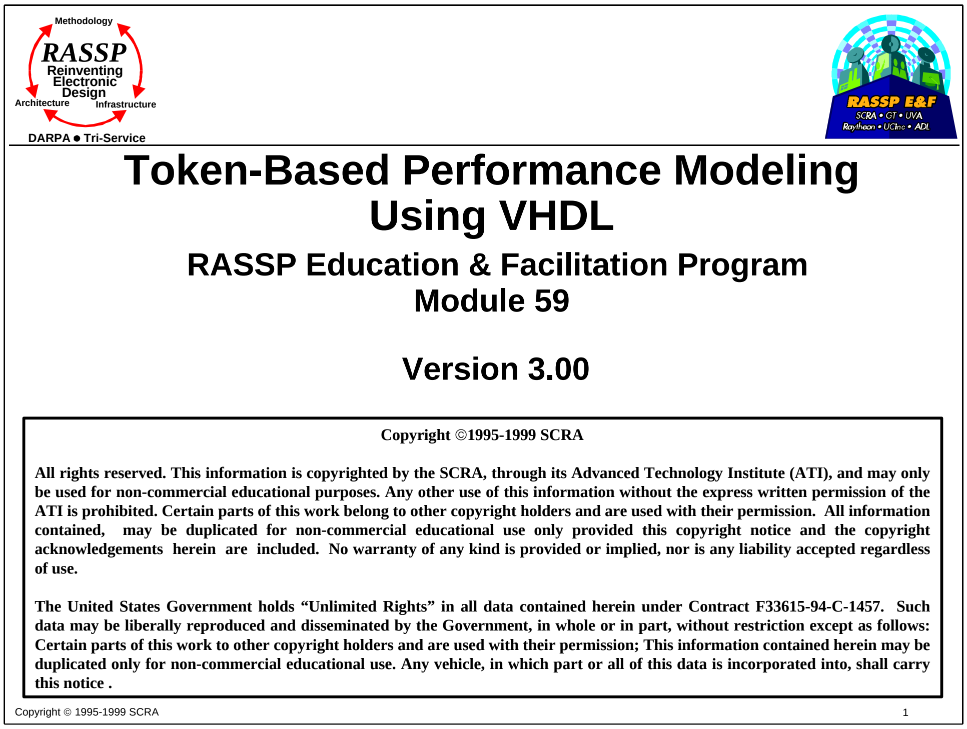
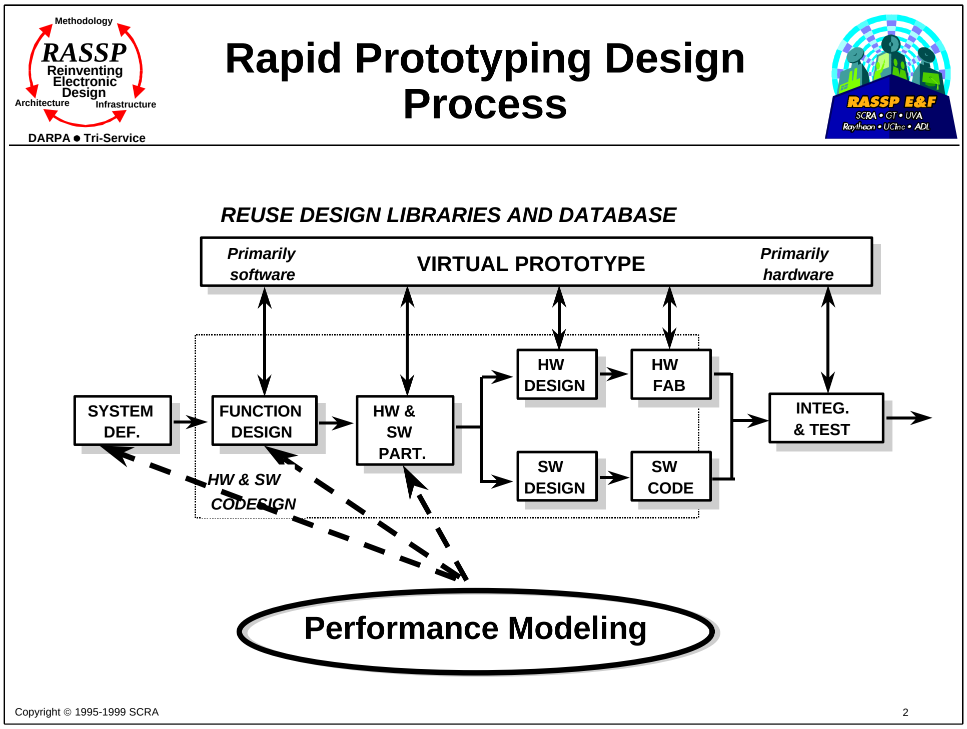


DARPA ● Tri-Service

Token-Based Performance Modeling Using VHDL

RASSP Education & Facilitation Program
Module 59

Version 3.00



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Rapid Prototyping Design Process

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

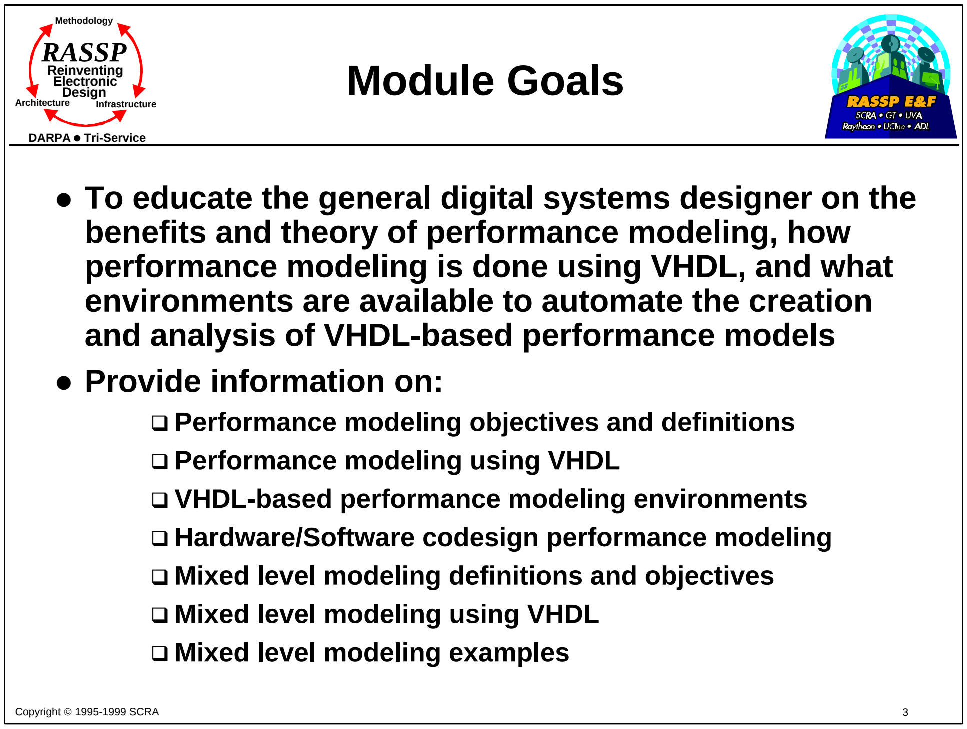
REUSE DESIGN LIBRARIES AND DATABASE

Primarily software VIRTUAL PROTOTYPE *Primarily hardware*

SYSTEM DEF.
FUNCTION DESIGN
HW & SW PART.
HW DESIGN
HW FAB
SW DESIGN
SW CODE
INTEG. & TEST
HW & SW CODESIGN

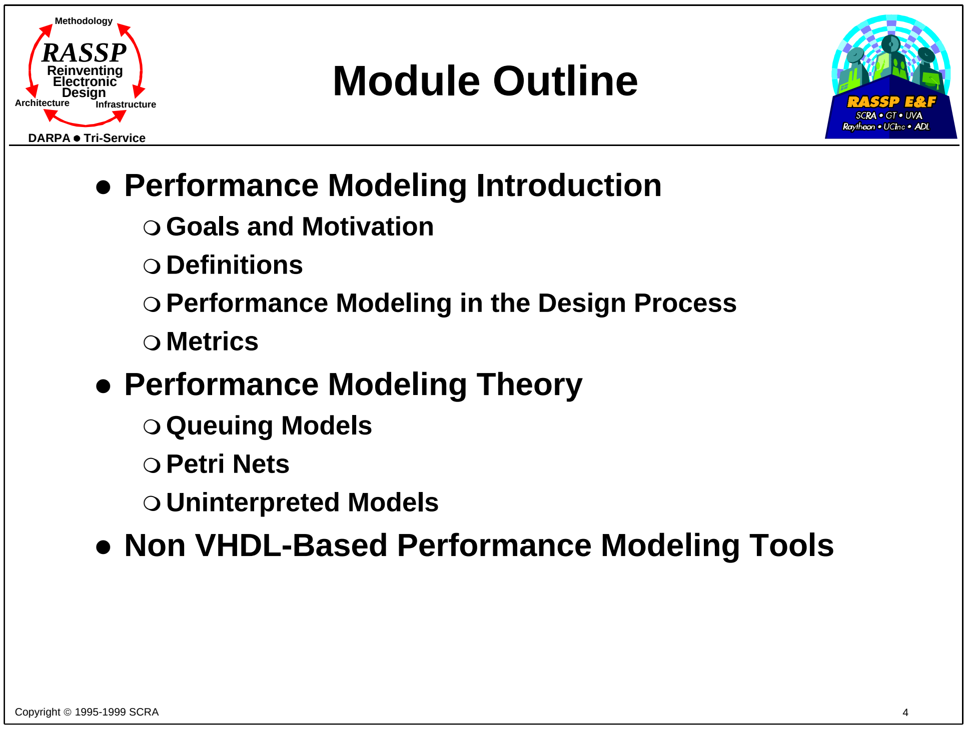
Performance Modeling

This slide shows the application area for performance modeling. It will be explained in more detail later in the module.



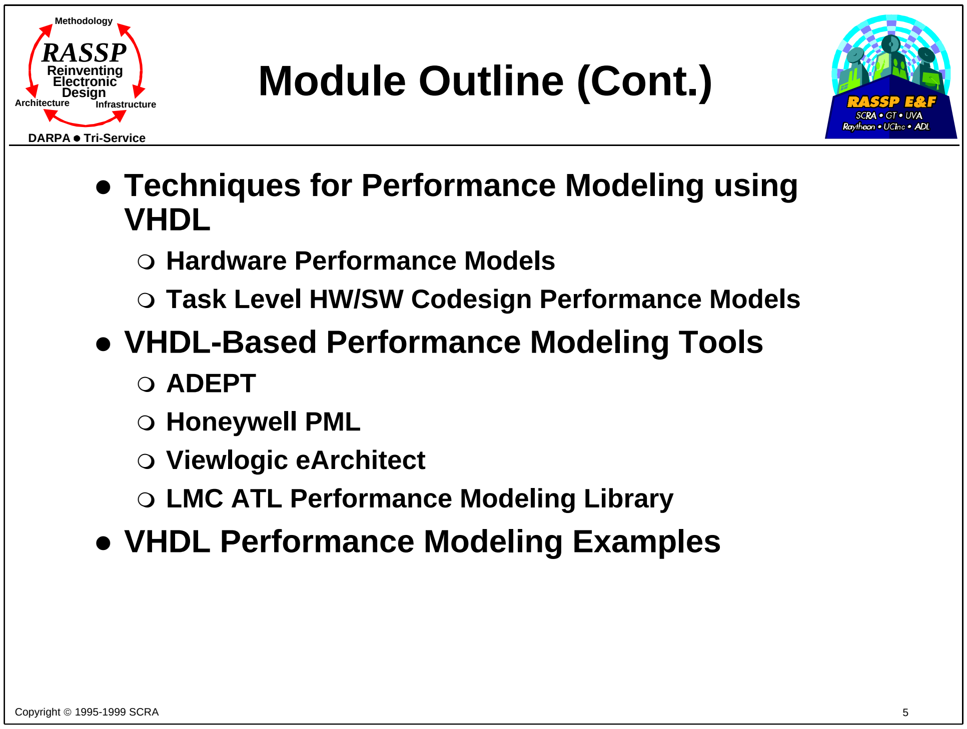
Module Goals

- To educate the general digital systems designer on the benefits and theory of performance modeling, how performance modeling is done using VHDL, and what environments are available to automate the creation and analysis of VHDL-based performance models
- Provide information on:
  - ❑ Performance modeling objectives and definitions
  - ❑ Performance modeling using VHDL
  - ❑ VHDL-based performance modeling environments
  - ❑ Hardware/Software codesign performance modeling
  - ❑ Mixed level modeling definitions and objectives
  - ❑ Mixed level modeling using VHDL
  - ❑ Mixed level modeling examples



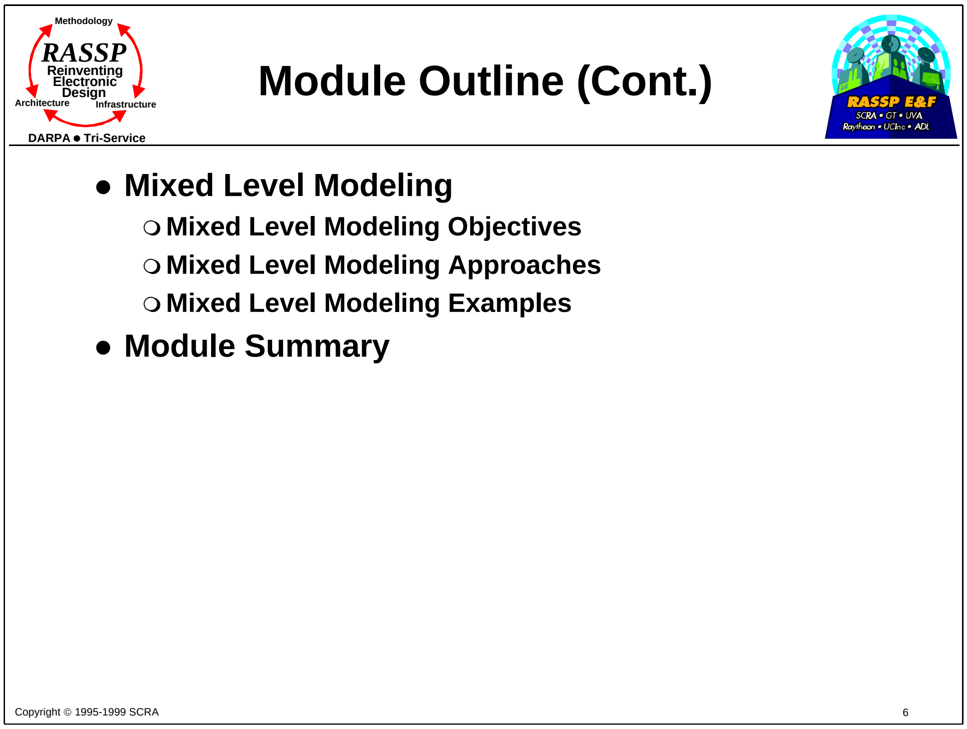
Module Outline

- Performance Modeling Introduction
  - ❍ Goals and Motivation
  - ❍ Definitions
  - ❍ Performance Modeling in the Design Process
  - ❍ Metrics
- Performance Modeling Theory
  - ❍ Queuing Models
  - ❍ Petri Nets
  - ❍ Uninterpreted Models
- Non VHDL-Based Performance Modeling Tools



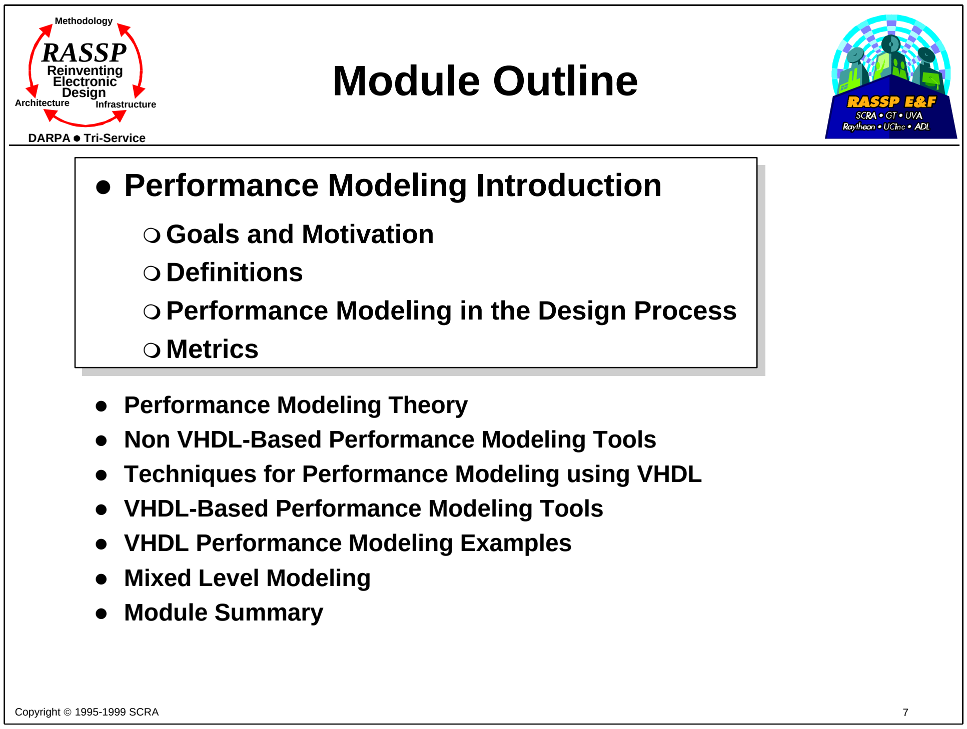
Module Outline (Cont.)

- Techniques for Performance Modeling using VHDL
  - Hardware Performance Models
  - Task Level HW/SW Codesign Performance Models
- VHDL-Based Performance Modeling Tools
  - ADEPT
  - Honeywell PML
  - Viewlogic eArchitect
  - LMC ATL Performance Modeling Library
- VHDL Performance Modeling Examples



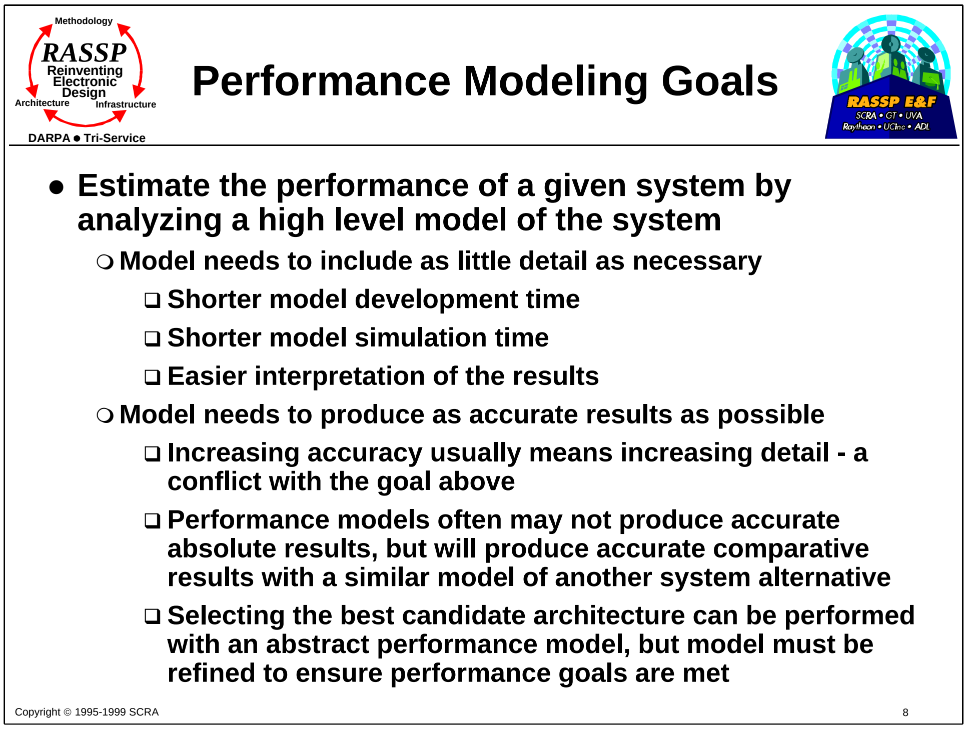
Module Outline (Cont.)

- Mixed Level Modeling
  - ❍ Mixed Level Modeling Objectives
  - ❍ Mixed Level Modeling Approaches
  - ❍ Mixed Level Modeling Examples
- Module Summary



Module Outline

- **Performance Modeling Introduction**
  - ❍ **Goals and Motivation**
  - ❍ **Definitions**
  - ❍ **Performance Modeling in the Design Process**
  - ❍ **Metrics**
- **Performance Modeling Theory**
- **Non VHDL-Based Performance Modeling Tools**
- **Techniques for Performance Modeling using VHDL**
- **VHDL-Based Performance Modeling Tools**
- **VHDL Performance Modeling Examples**
- **Mixed Level Modeling**
- **Module Summary**



Performance Modeling Goals

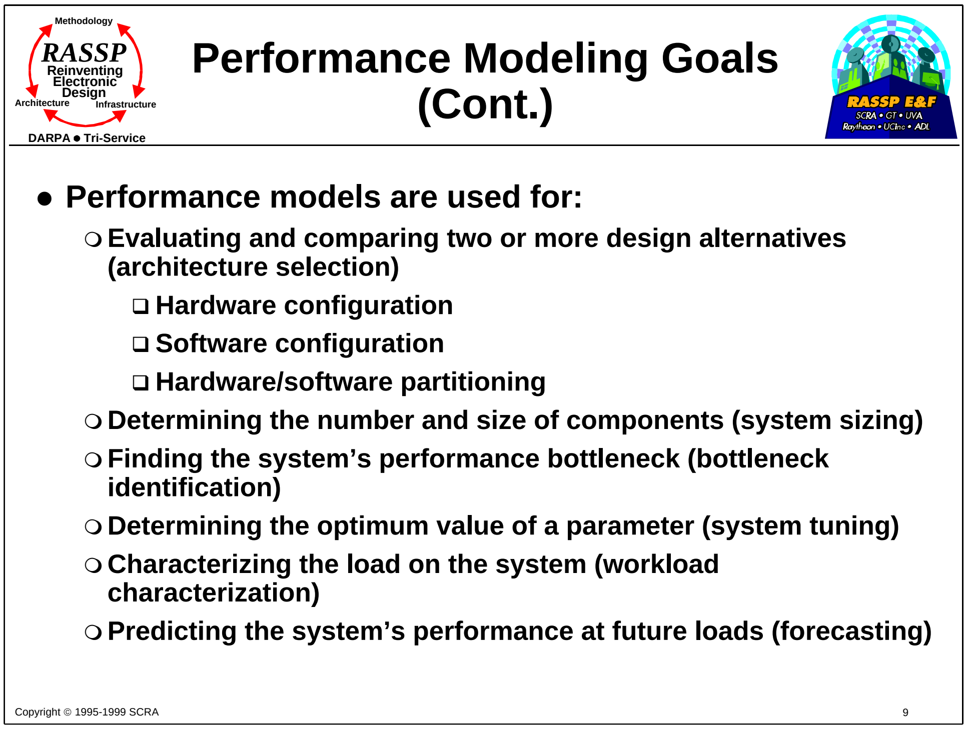
- Estimate the performance of a given system by analyzing a high level model of the system
  - ❍ Model needs to include as little detail as necessary
    - ❑ Shorter model development time
    - ❑ Shorter model simulation time
    - ❑ Easier interpretation of the results
  - ❍ Model needs to produce as accurate results as possible
    - ❑ Increasing accuracy usually means increasing detail - a conflict with the goal above
    - ❑ Performance models often may not produce accurate absolute results, but will produce accurate comparative results with a similar model of another system alternative
    - ❑ Selecting the best candidate architecture can be performed with an abstract performance model, but model must be refined to ensure performance goals are met

8

The goal of performance modeling is to analyze the performance model of a system using a high-level model. The model needs to be at as high (abstract) a level as possible to reduce model generation, verification, and simulation time, but at a low enough level that accurate results are obtained.

How to determine this level is not an easy process but is usually best approached from the “to little detail” side down.

Abstract performance models may not give completely accurate absolute results as in “this architecture will have a throughput of X jobs per second,” but can give accurate comparative results as in “architecture A has a 20% greater throughput than architecture B.”

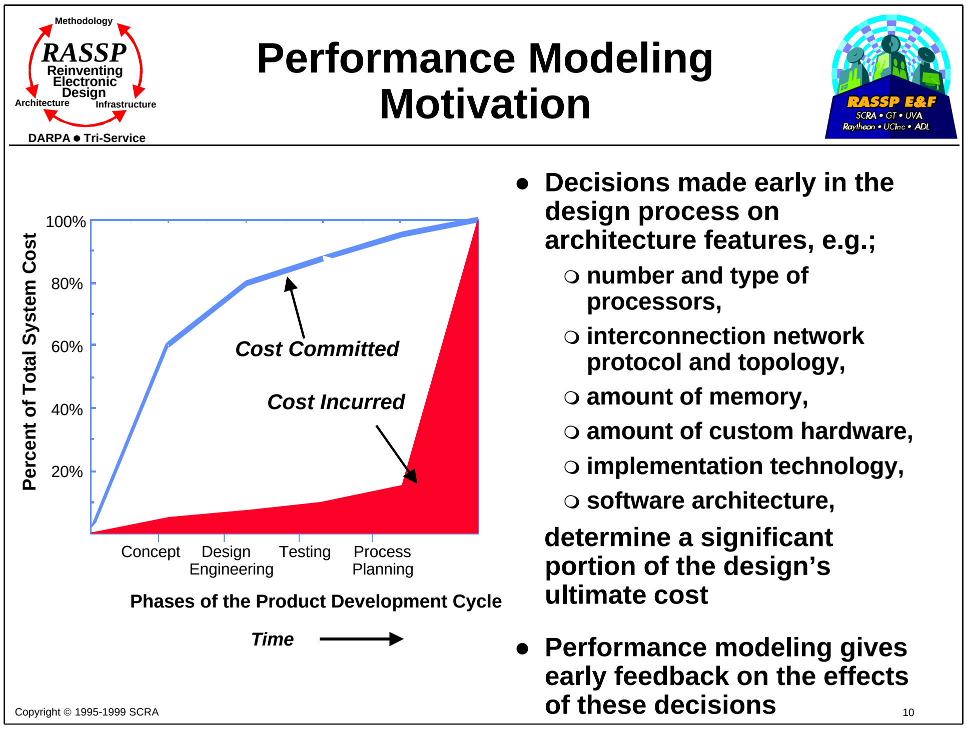


Performance Modeling Goals (Cont.)

- Performance models are used for:
  - ❍ Evaluating and comparing two or more design alternatives (architecture selection)
    - ❑ Hardware configuration
    - ❑ Software configuration
    - ❑ Hardware/software partitioning
  - ❍ Determining the number and size of components (system sizing)
  - ❍ Finding the system’s performance bottleneck (bottleneck identification)
  - ❍ Determining the optimum value of a parameter (system tuning)
  - ❍ Characterizing the load on the system (workload characterization)
  - ❍ Predicting the system’s performance at future loads (forecasting)

This list comes from many of the references, but mainly from [Jain91]

In this module, we are discussing the mainly the application of performance modeling to the architecture selection process.



DARPA ● Tri-Service

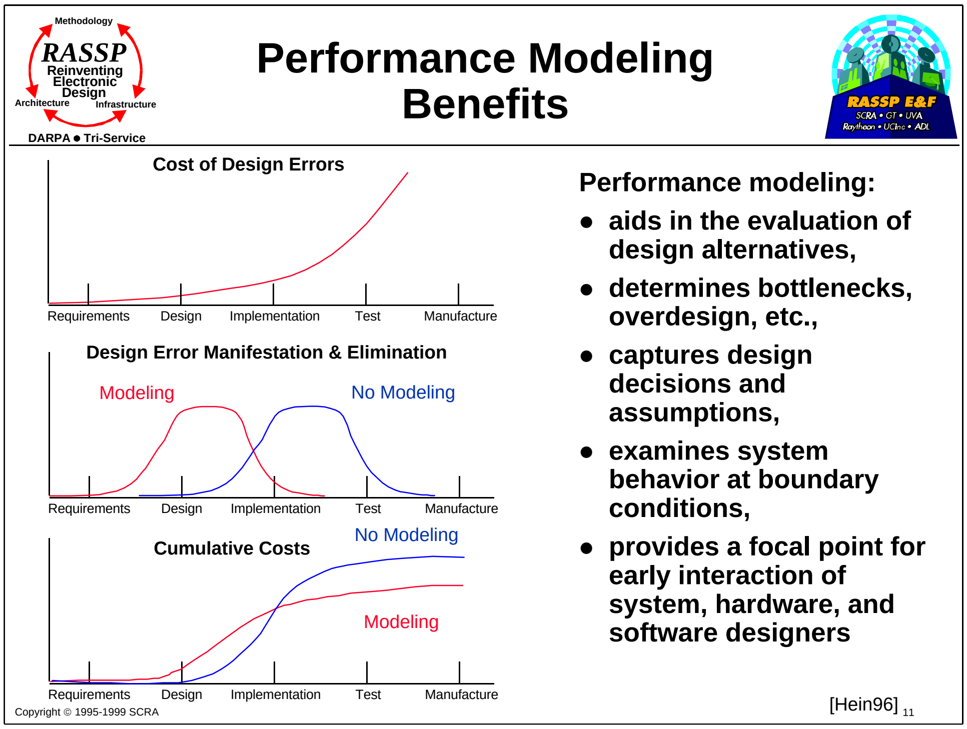
Performance Modeling Motivation

- Decisions made early in the design process on architecture features, e.g.;
  - ❍ number and type of processors,
  - ❍ interconnection network protocol and topology,
  - ❍ amount of memory,
  - ❍ amount of custom hardware,
  - ❍ implementation technology,
  - ❍ software architecture,

  determine a significant portion of the design's ultimate cost
- Performance modeling gives early feedback on the effects of these decisions

This graph shows that most of the final cost of a system is locked in during the early phases of the design process when the architecture of the system is selected. However, the cost incurred in designing and producing the system does not reach its peak until the product is going out the door. Therefore, spending some time (and money) looking at the final cost of candidate architectures and their performance, early in the design process can save a great deal.

Note that these curves will change some if performance modeling is used in that more cost will be incurred early as design cost for the early stages increases, and the cost committed early will be less as the actual selection of the architecture is done later in the design cycle.



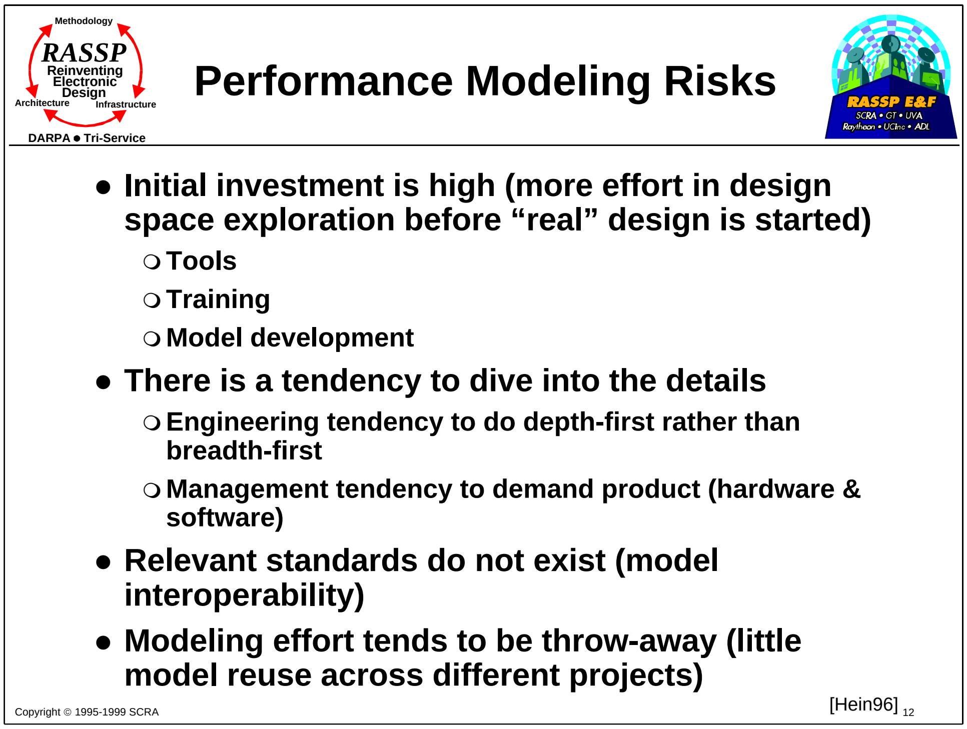
Performance Modeling Benefits

Performance modeling:

- **aids in the evaluation of design alternatives,**
- **determines bottlenecks, overdesign, etc.,**
- **captures design decisions and assumptions,**
- **examines system behavior at boundary conditions,**
- **provides a focal point for early interaction of system, hardware, and software designers**

[Hein96] 11

This slide shows some of the benefits of performance modeling as seen by some industrial users of the technique. Note that using performance modeling results in design errors being manifested and eliminated earlier in the design process where they are less costly. Also note that initially, the cost of a design process with performance modeling is higher, but the overall cost (area under the curve) is lower.

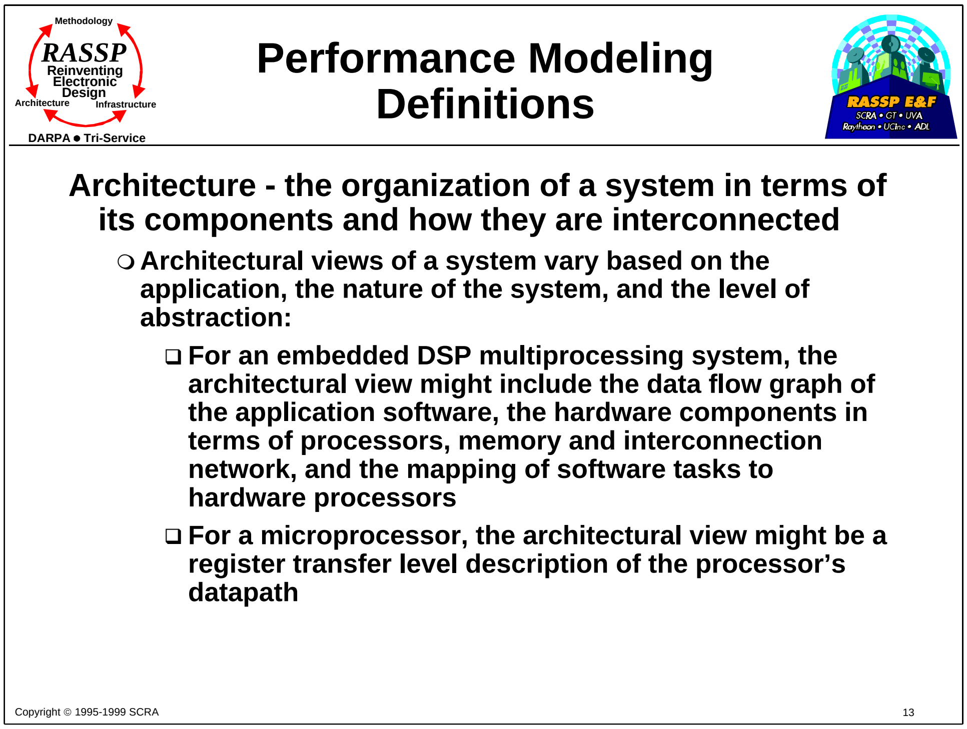

Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Performance Modeling Risks

- Initial investment is high (more effort in design space exploration before “real” design is started)
  - ❍ Tools
  - ❍ Training
  - ❍ Model development
- There is a tendency to dive into the details
  - ❍ Engineering tendency to do depth-first rather than breadth-first
  - ❍ Management tendency to demand product (hardware & software)
- Relevant standards do not exist (model interoperability)
- Modeling effort tends to be throw-away (little model reuse across different projects)

[Hein96] 12

The initial investment in performance modeling is high in that it increases the time spent in design space exploration before the design of the chosen architecture is actually started. This is increased by the fact that often, designers need to be trained to use the tools and develop the models necessary for performance modeling. However, the goal of performance modeling is to significantly reduce the detailed design time and cost for the chosen system by eliminating costly redesigns and design errors, thereby decreasing the overall design time and cost.



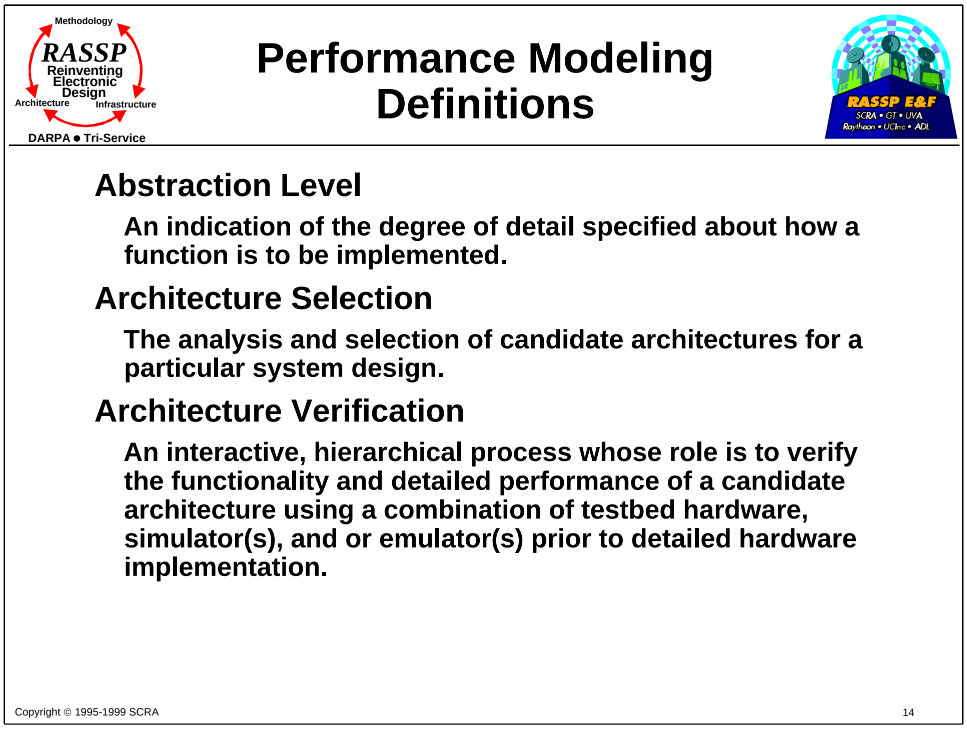
Performance Modeling Definitions

Architecture - the organization of a system in terms of its components and how they are interconnected

- ❍ **Architectural views of a system vary based on the application, the nature of the system, and the level of abstraction:**
  - ❑ **For an embedded DSP multiprocessing system, the architectural view might include the data flow graph of the application software, the hardware components in terms of processors, memory and interconnection network, and the mapping of software tasks to hardware processors**
  - ❑ **For a microprocessor, the architectural view might be a register transfer level description of the processor's datapath**

13

The definition of architecture is different for different systems and different levels of abstraction.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

Performance Modeling Definitions

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Abstraction Level

An indication of the degree of detail specified about how a function is to be implemented.

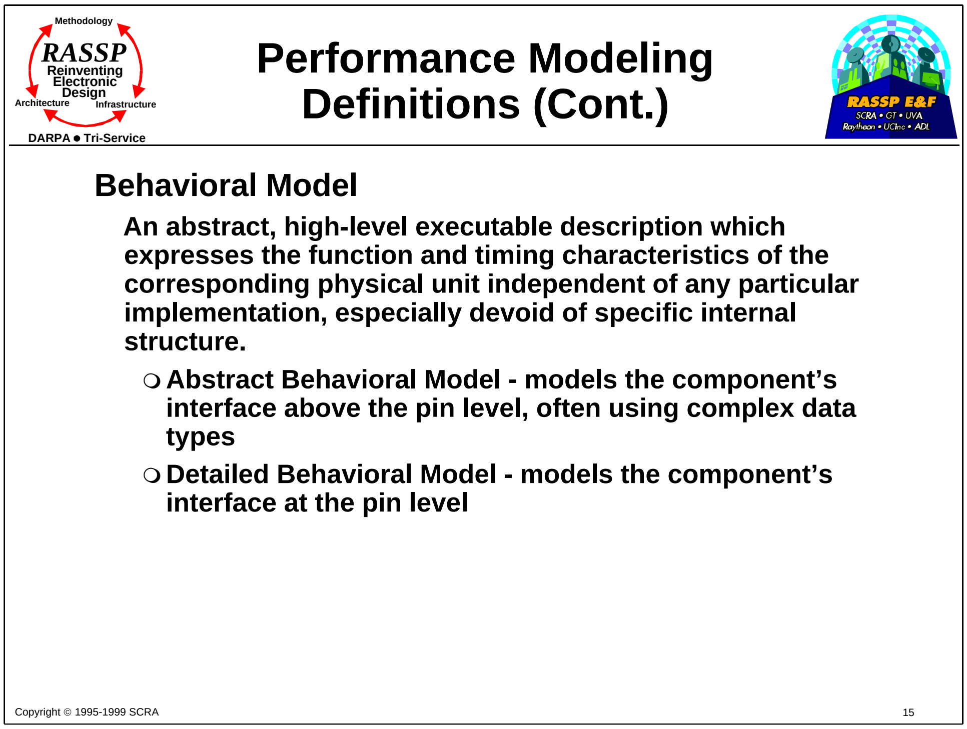
Architecture Selection

The analysis and selection of candidate architectures for a particular system design.

Architecture Verification

An interactive, hierarchical process whose role is to verify the functionality and detailed performance of a candidate architecture using a combination of testbed hardware, simulator(s), and or emulator(s) prior to detailed hardware implementation.

Architecture selection and architecture verification will be explained in more detail later in the module.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

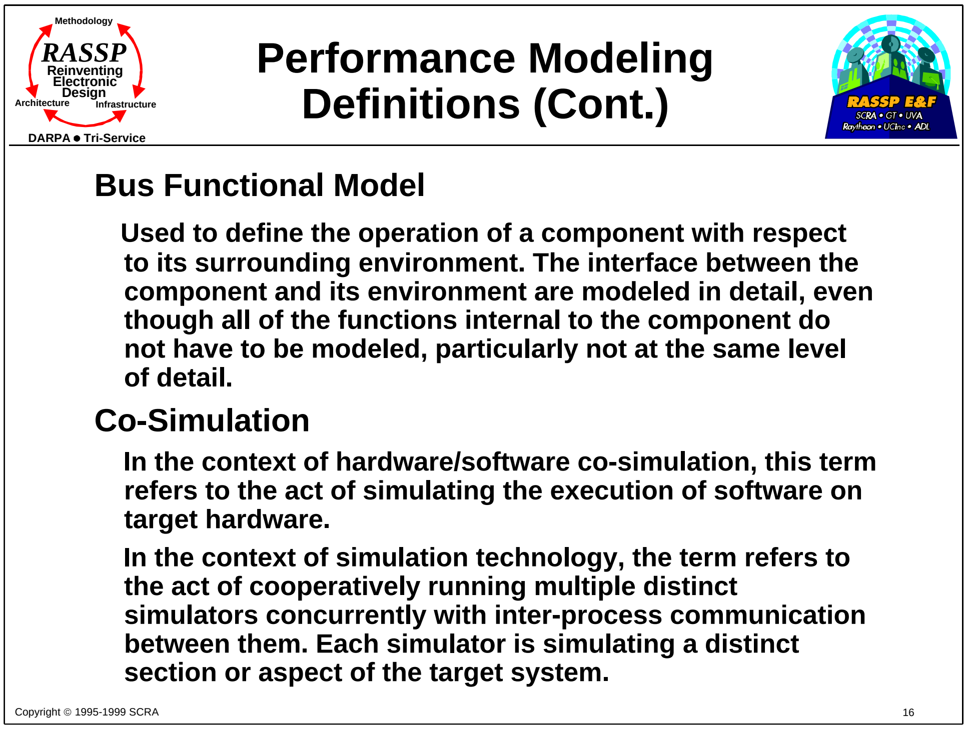
Performance Modeling Definitions (Cont.)

Behavioral Model

An abstract, high-level executable description which expresses the function and timing characteristics of the corresponding physical unit independent of any particular implementation, especially devoid of specific internal structure.

- ❍ **Abstract Behavioral Model - models the component’s interface above the pin level, often using complex data types**
- ❍ **Detailed Behavioral Model - models the component’s interface at the pin level**

All definitions of model types are consistent with the RASSP Taxonomy [Hein97]



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Performance Modeling Definitions (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Bus Functional Model

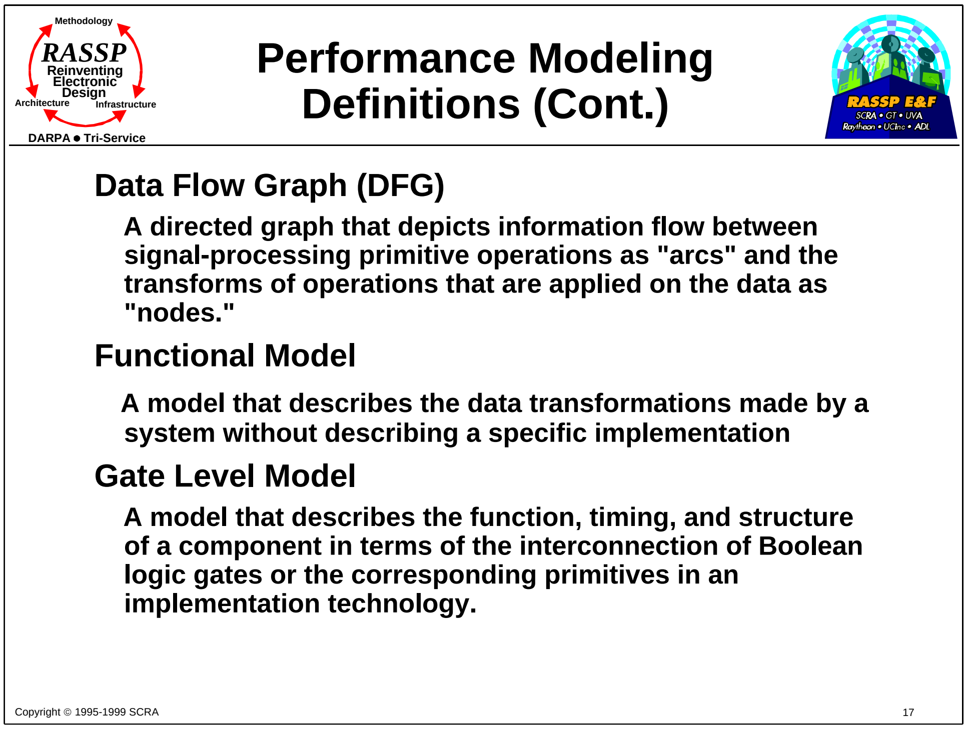
Used to define the operation of a component with respect to its surrounding environment. The interface between the component and its environment are modeled in detail, even though all of the functions internal to the component do not have to be modeled, particularly not at the same level of detail.

Co-Simulation

In the context of hardware/software co-simulation, this term refers to the act of simulating the execution of software on target hardware.

In the context of simulation technology, the term refers to the act of cooperatively running multiple distinct simulators concurrently with inter-process communication between them. Each simulator is simulating a distinct section or aspect of the target system.

No notes necessary.



Performance Modeling Definitions (Cont.)

Data Flow Graph (DFG)

A directed graph that depicts information flow between signal-processing primitive operations as "arcs" and the transforms of operations that are applied on the data as "nodes."

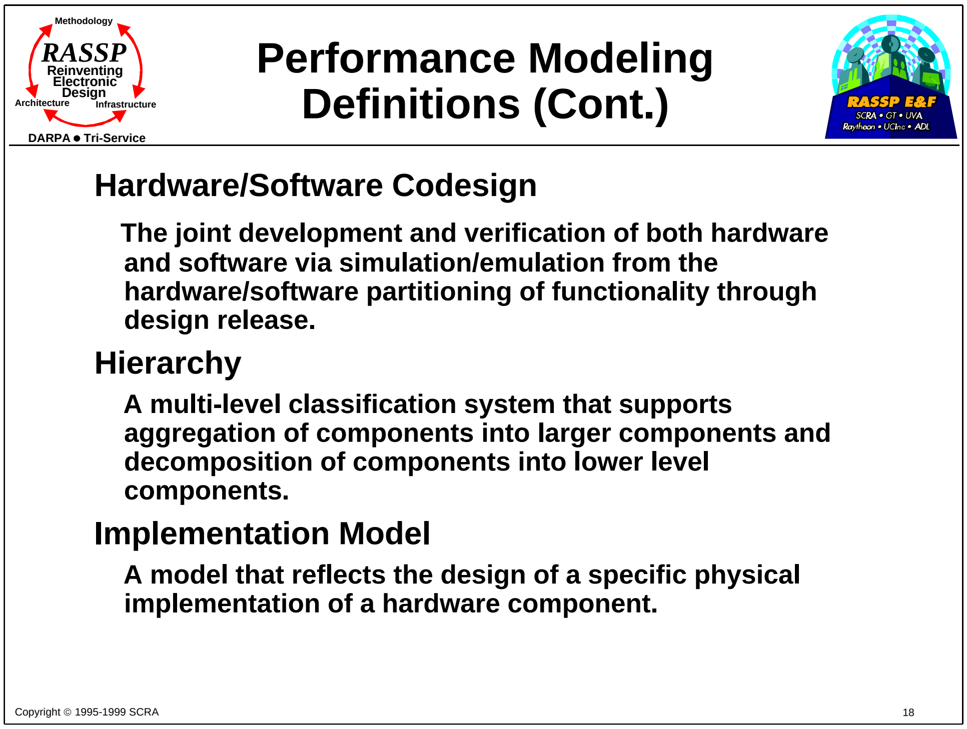
Functional Model

A model that describes the data transformations made by a system without describing a specific implementation

Gate Level Model

A model that describes the function, timing, and structure of a component in terms of the interconnection of Boolean logic gates or the corresponding primitives in an implementation technology.

No notes necessary.



Performance Modeling Definitions (Cont.)

Hardware/Software Codesign

The joint development and verification of both hardware and software via simulation/emulation from the hardware/software partitioning of functionality through design release.

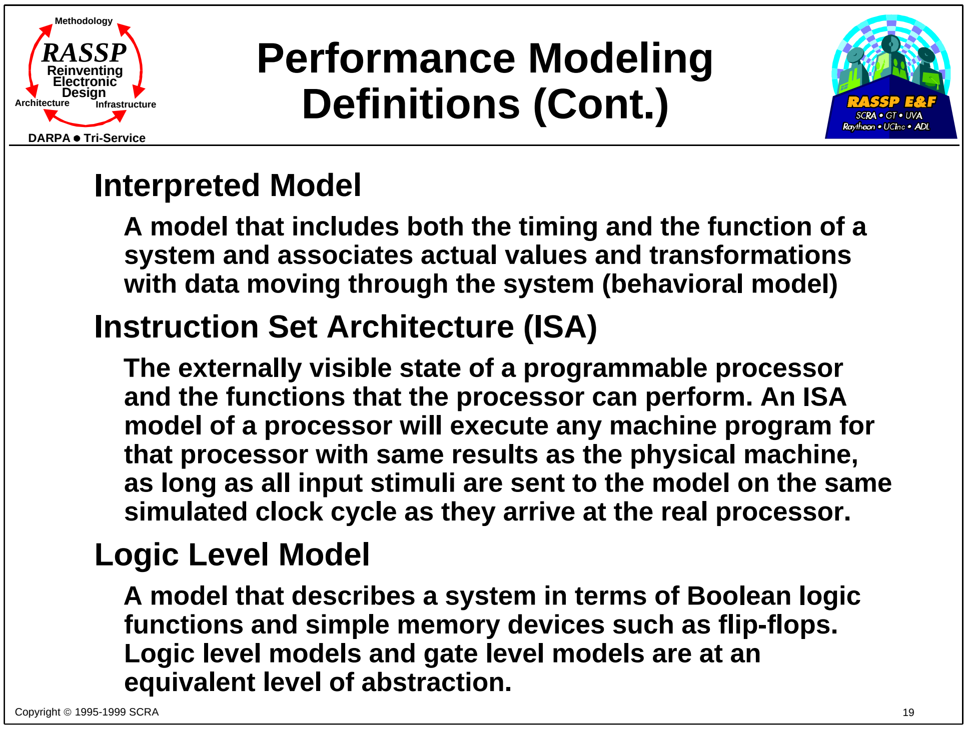
Hierarchy

A multi-level classification system that supports aggregation of components into larger components and decomposition of components into lower level components.

Implementation Model

A model that reflects the design of a specific physical implementation of a hardware component.

No notes necessary.



Performance Modeling Definitions (Cont.)

Interpreted Model

A model that includes both the timing and the function of a system and associates actual values and transformations with data moving through the system (behavioral model)

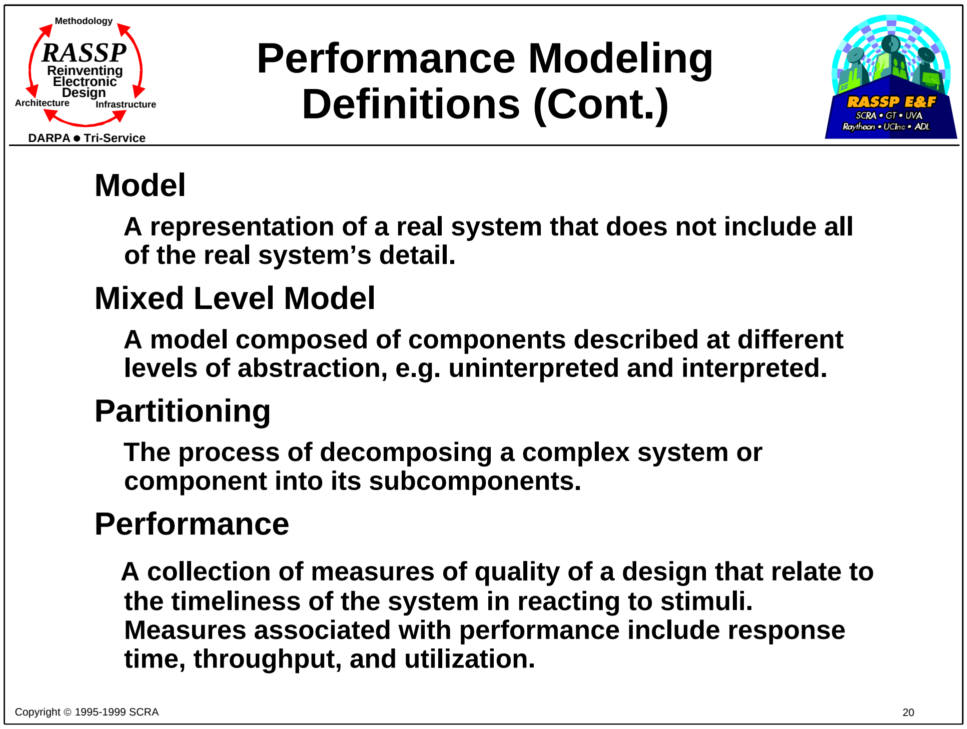
Instruction Set Architecture (ISA)

The externally visible state of a programmable processor and the functions that the processor can perform. An ISA model of a processor will execute any machine program for that processor with same results as the physical machine, as long as all input stimuli are sent to the model on the same simulated clock cycle as they arrive at the real processor.

Logic Level Model

A model that describes a system in terms of Boolean logic functions and simple memory devices such as flip-flops. Logic level models and gate level models are at an equivalent level of abstraction.

No notes necessary.



Performance Modeling Definitions (Cont.)

Model

A representation of a real system that does not include all of the real system's detail.

Mixed Level Model

A model composed of components described at different levels of abstraction, e.g. uninterpreted and interpreted.

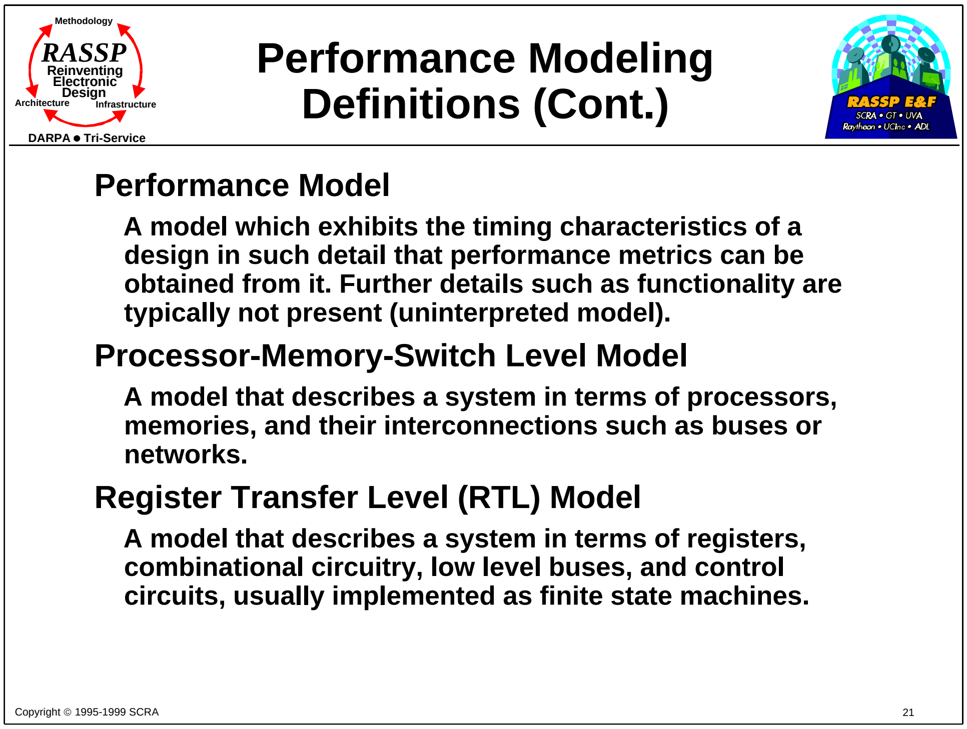
Partitioning

The process of decomposing a complex system or component into its subcomponents.

Performance

A collection of measures of quality of a design that relate to the timeliness of the system in reacting to stimuli. Measures associated with performance include response time, throughput, and utilization.

No notes necessary.



Performance Modeling Definitions (Cont.)

Performance Model

A model which exhibits the timing characteristics of a design in such detail that performance metrics can be obtained from it. Further details such as functionality are typically not present (uninterpreted model).

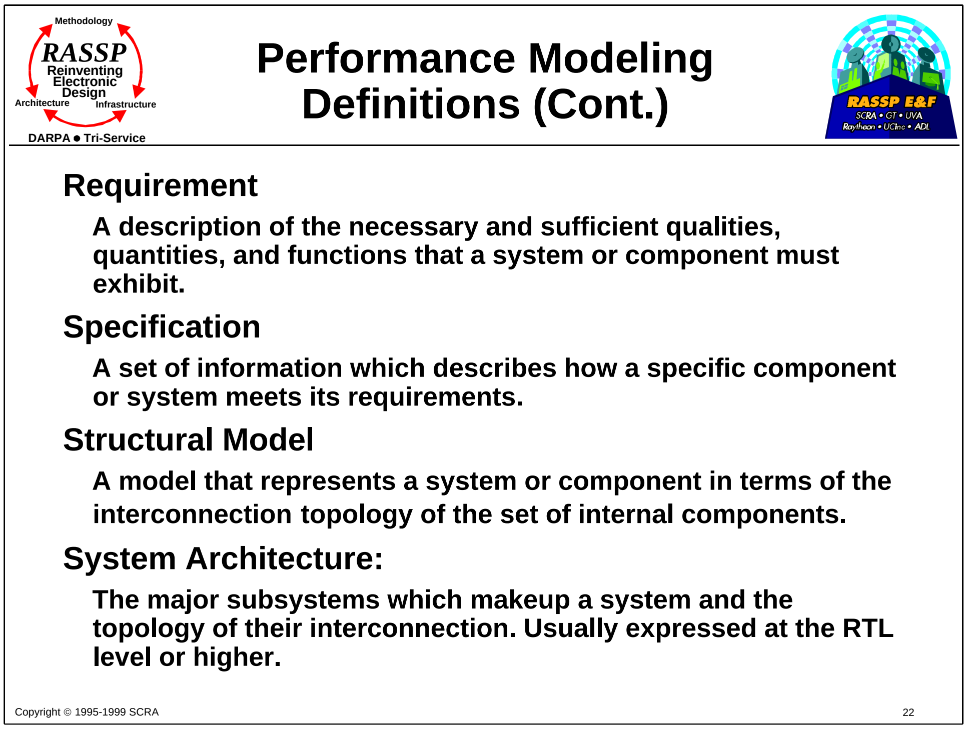
Processor-Memory-Switch Level Model

A model that describes a system in terms of processors, memories, and their interconnections such as buses or networks.

Register Transfer Level (RTL) Model

A model that describes a system in terms of registers, combinational circuitry, low level buses, and control circuits, usually implemented as finite state machines.

No notes necessary.



Performance Modeling Definitions (Cont.)

Requirement

A description of the necessary and sufficient qualities, quantities, and functions that a system or component must exhibit.

Specification

A set of information which describes how a specific component or system meets its requirements.

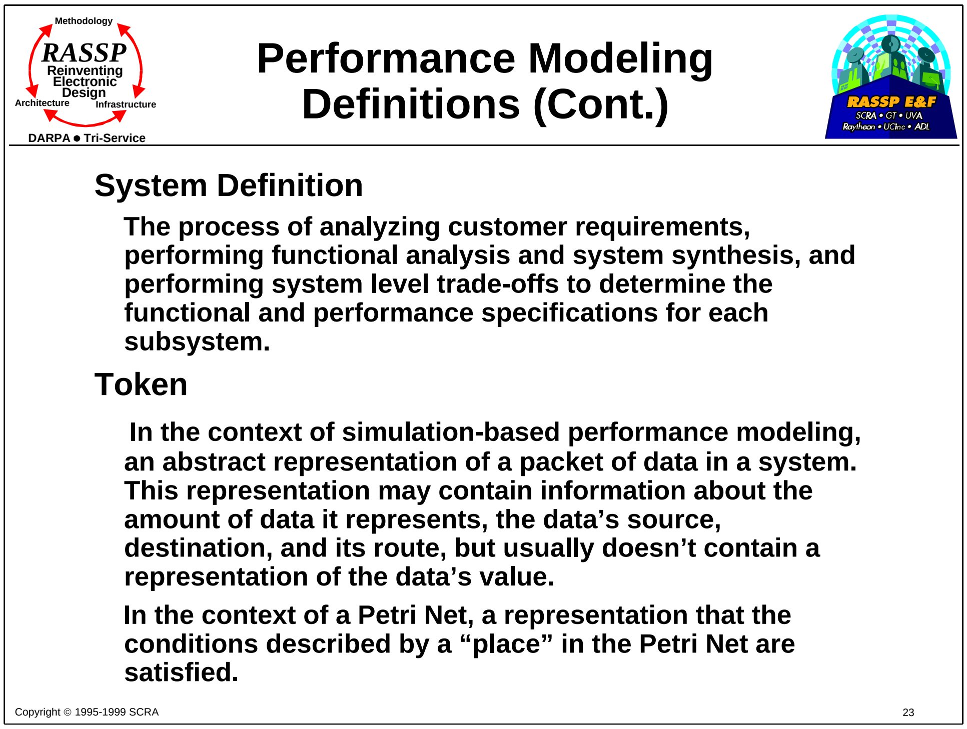
Structural Model

A model that represents a system or component in terms of the interconnection topology of the set of internal components.

System Architecture:

The major subsystems which makeup a system and the topology of their interconnection. Usually expressed at the RTL level or higher.

No notes necessary.



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Performance Modeling Definitions (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

System Definition

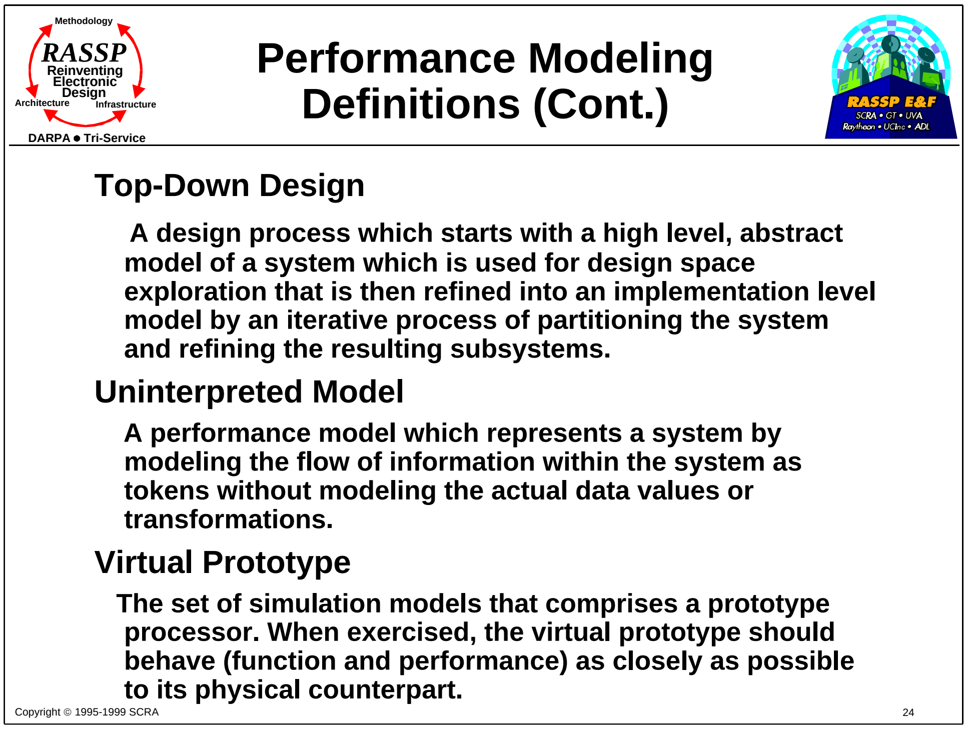
The process of analyzing customer requirements, performing functional analysis and system synthesis, and performing system level trade-offs to determine the functional and performance specifications for each subsystem.

Token

In the context of simulation-based performance modeling, an abstract representation of a packet of data in a system. This representation may contain information about the amount of data it represents, the data's source, destination, and its route, but usually doesn't contain a representation of the data's value.

In the context of a Petri Net, a representation that the conditions described by a "place" in the Petri Net are satisfied.

No notes necessary.



Performance Modeling Definitions (Cont.)

Top-Down Design

A design process which starts with a high level, abstract model of a system which is used for design space exploration that is then refined into an implementation level model by an iterative process of partitioning the system and refining the resulting subsystems.

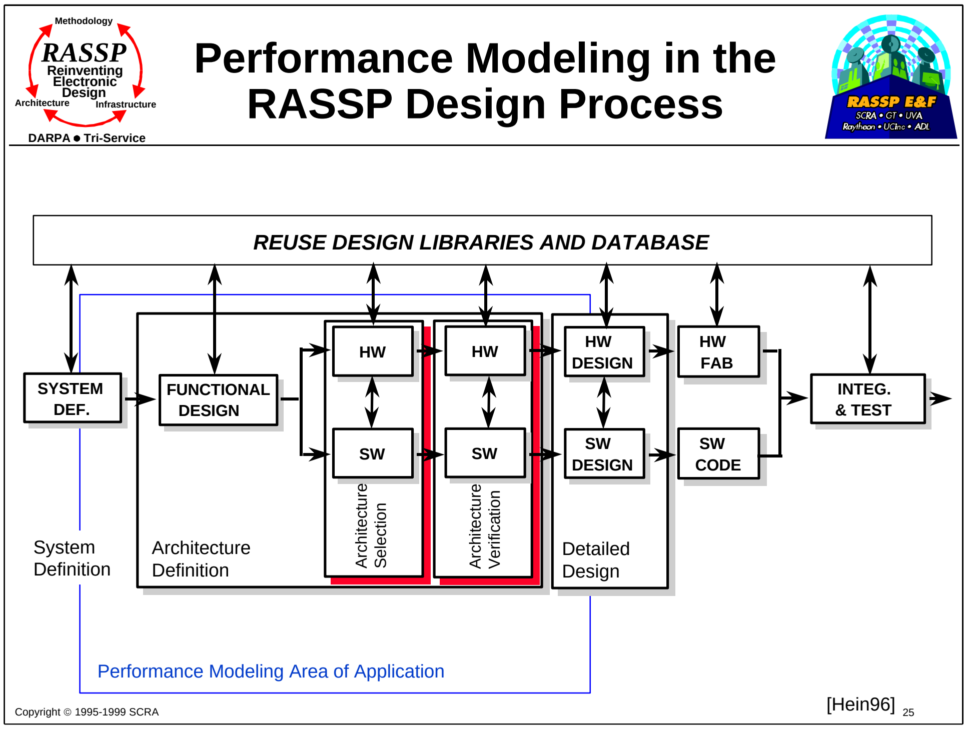
Uninterpreted Model

A performance model which represents a system by modeling the flow of information within the system as tokens without modeling the actual data values or transformations.

Virtual Prototype

The set of simulation models that comprises a prototype processor. When exercised, the virtual prototype should behave (function and performance) as closely as possible to its physical counterpart.

No notes necessary.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

Performance Modeling in the RASSP Design Process

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

REUSE DESIGN LIBRARIES AND DATABASE

SYSTEM DEF.
FUNCTIONAL DESIGN
HW
SW
HW
SW
HW DESIGN
SW DESIGN
HW FAB
SW CODE
INTEG. & TEST

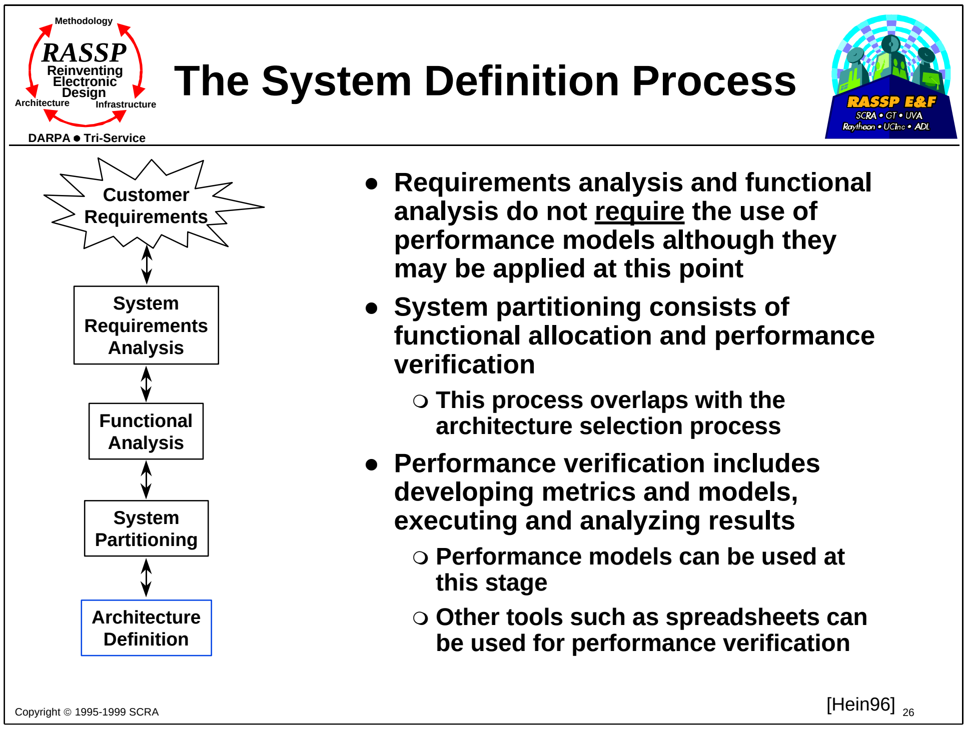
System Definition
Architecture Definition
Architecture Selection
Architecture Verification
Detailed Design

Performance Modeling Area of Application

[Hein96] 25

This slide shows the RASSP (Rapid Prototyping of Application Specific Signal Processors) design process and where performance modeling fits into it. This includes the processes of System Definition, Architecture Definition, and portions of Detailed Design. Note that Architecture Definition encompasses Functional Design and the processes of Architectural Selection Architectural Verification.

How performance modeling is used within these processes is covered in the following slides...


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

The System Definition Process

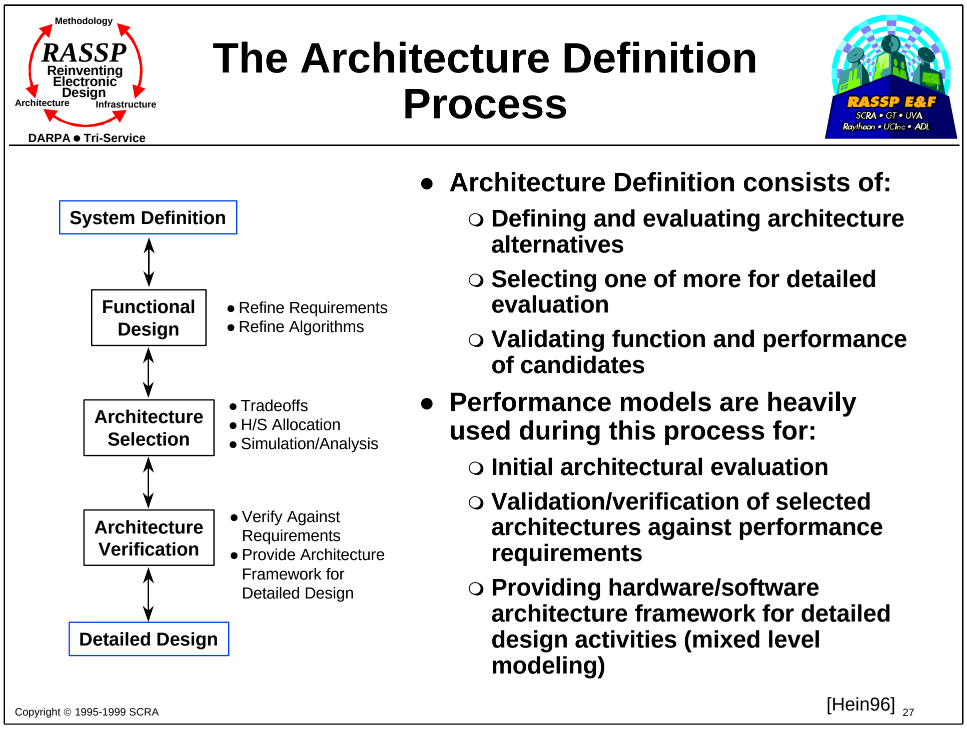
Customer Requirements
System Requirements Analysis
Functional Analysis
System Partitioning
Architecture Definition

- Requirements analysis and functional analysis do not require the use of performance models although they may be applied at this point
- System partitioning consists of functional allocation and performance verification
  - ❍ This process overlaps with the architecture selection process
- Performance verification includes developing metrics and models, executing and analyzing results
  - ❍ Performance models can be used at this stage
  - ❍ Other tools such as spreadsheets can be used for performance verification

[Hein96] 26

This slide presents the functions in the system definition process, which begins with customer requirements (which may be executable) and flows into the architecture definition process.

Performance models are not required at the upper levels of the system definition process although they can be applied at any point. In the performance verification phase, some type of performance modeling is required for all but the most trivial of systems.



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

The Architecture Definition Process

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL

System Definition

Functional Design

- Refine Requirements
- Refine Algorithms

Architecture Selection

- Tradeoffs
- H/S Allocation
- Simulation/Analysis

Architecture Verification

- Verify Against Requirements
- Provide Architecture Framework for Detailed Design

Detailed Design

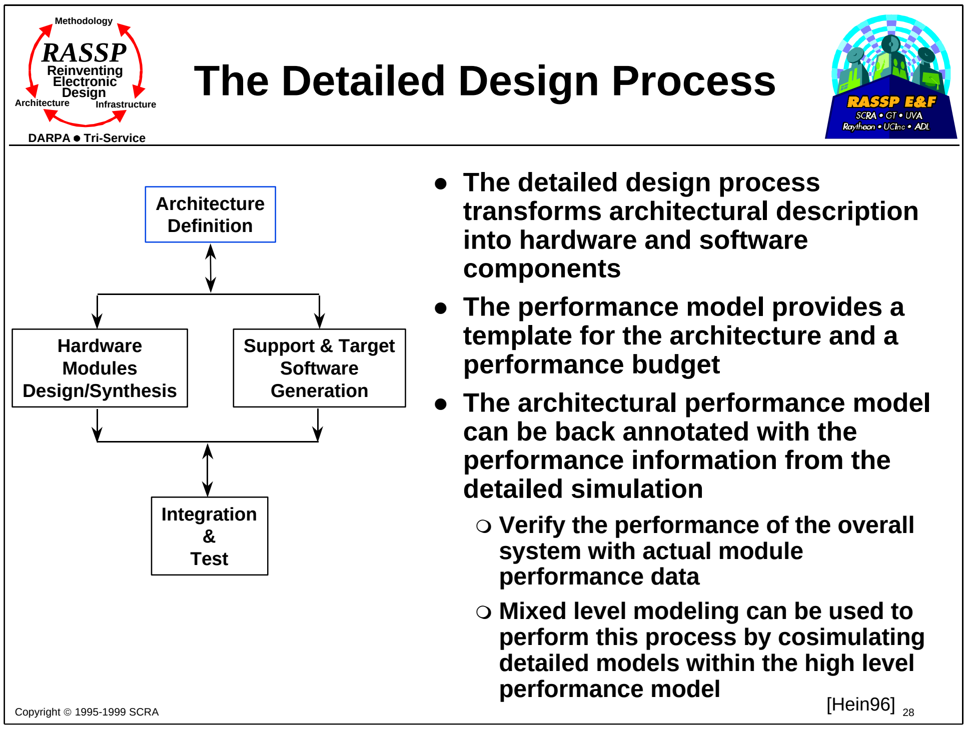
- **Architecture Definition consists of:**
  - ❍ **Defining and evaluating architecture alternatives**
  - ❍ **Selecting one of more for detailed evaluation**
  - ❍ **Validating function and performance of candidates**
- **Performance models are heavily used during this process for:**
  - ❍ **Initial architectural evaluation**
  - ❍ **Validation/verification of selected architectures against performance requirements**
  - ❍ **Providing hardware/software architecture framework for detailed design activities (mixed level modeling)**

[Hein96] 27

The architecture definition process is fed by the system definition process and in turn feeds into the detailed design process.

Performance models can be used in the functional design process to help refine requirements and algorithms. They most definitely are used in the architecture selection and verification process for evaluation.

Note that this slide show one view of the architecture definition process, but it can be pursued in other ways (more of an iterative process, less of a waterfall, etc.)



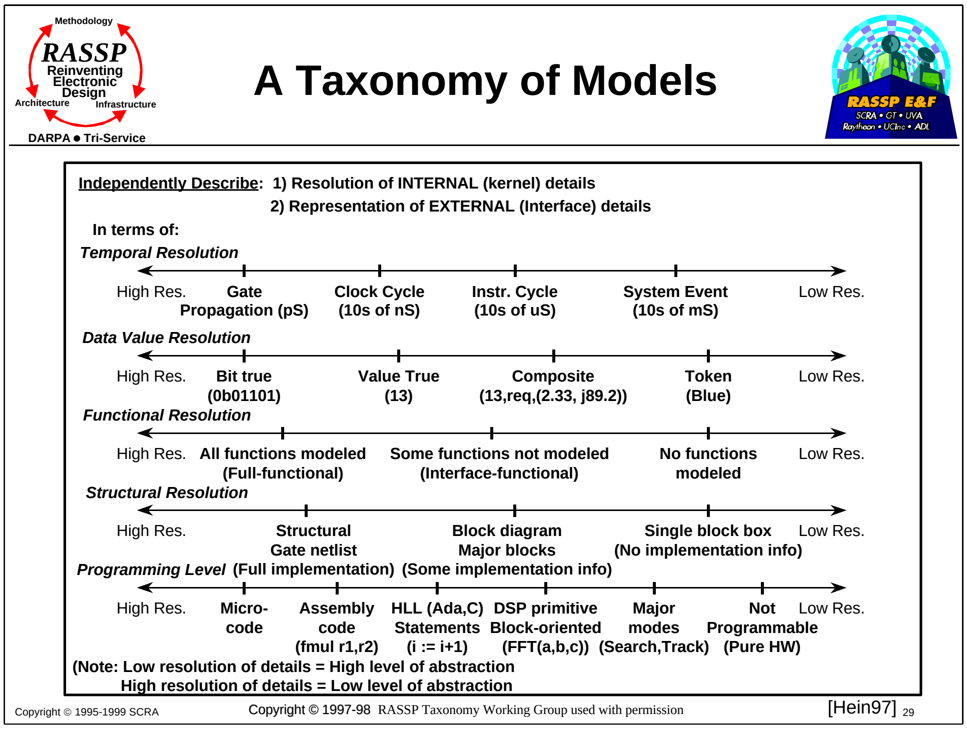
The Detailed Design Process

Architecture Definition
Hardware Modules Design/Synthesis
Support & Target Software Generation
Integration & Test

- The detailed design process transforms architectural description into hardware and software components
- The performance model provides a template for the architecture and a performance budget
- The architectural performance model can be back annotated with the performance information from the detailed simulation
  - ❍ Verify the performance of the overall system with actual module performance data
  - ❍ Mixed level modeling can be used to perform this process by cosimulating detailed models within the high level performance model

[Hein96] 28

This slide shows the detailed design process and how performance modeling is used in it. Note that this is where mixed level modeling, the notion of cosimulating performance and behavioral models, is introduced.



A Taxonomy of Models

Independently Describe: 1) Resolution of INTERNAL (kernel) details
2) Representation of EXTERNAL (Interface) details

In terms of:

Temporal Resolution

High Res. — **Gate Propagation (pS)** — **Clock Cycle (10s of nS)** — **Instr. Cycle (10s of uS)** — **System Event (10s of mS)** — Low Res.

Data Value Resolution

High Res. — **Bit true (0b01101)** — **Value True (13)** — **Composite (13,req,(2.33, j89.2))** — **Token (Blue)** — Low Res.

Functional Resolution

High Res. — **All functions modeled (Full-functional)** — **Some functions not modeled (Interface-functional)** — **No functions modeled** — Low Res.

Structural Resolution

High Res. — **Structural Gate netlist (Full implementation)** — **Block diagram Major blocks (Some implementation info)** — **Single block box (No implementation info)** — Low Res.

Programming Level

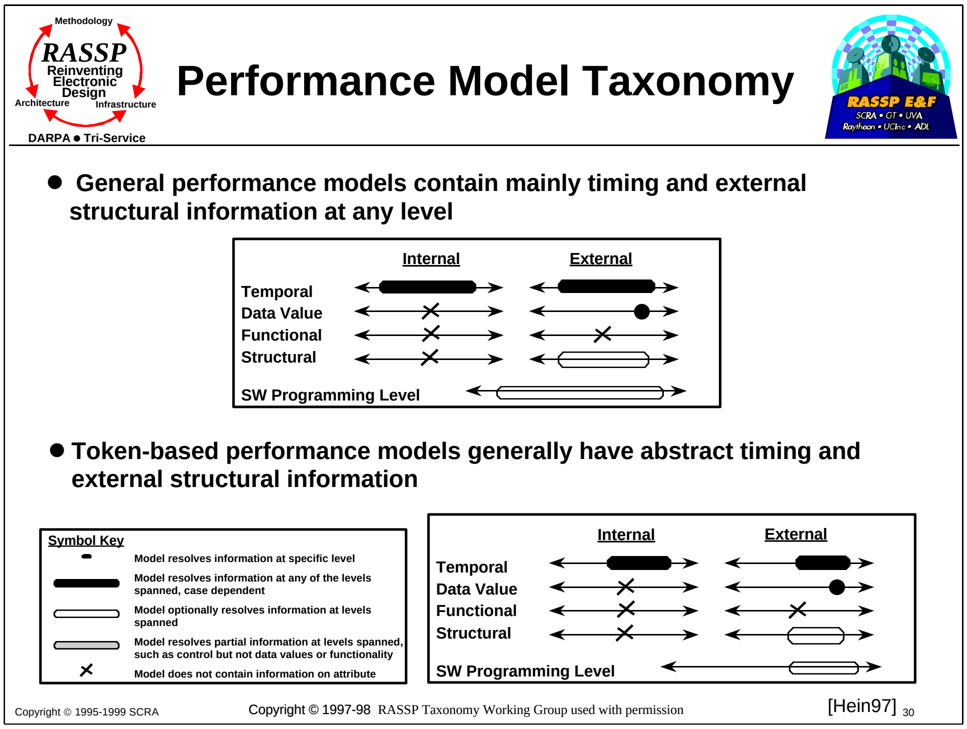
High Res. — **Micro-code** — **Assembly code (fmul r1,r2)** — **HLL (Ada,C) Statements (i := i+1)** — **DSP primitive Block-oriented (FFT(a,b,c))** — **Major modes (Search,Track)** — **Not Programmable (Pure HW)** — Low Res.

(Note: Low resolution of details = High level of abstraction
High resolution of details = Low level of abstraction

[Hein97] 29

This slide shows the 5 elements of model characteristics that determines its place in the overall taxonomy of models. The position on each scale that a model occupies determines what type of model it is classified as.

This slide is taken from the RASSP taxonomy document.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Performance Model Taxonomy

- General performance models contain mainly timing and external structural information at any level

	Internal	External
Temporal	▬	▬
Data Value	×	●
Functional	×	×
Structural	×	▭
SW Programming Level	▭	▭

- Token-based performance models generally have abstract timing and external structural information

	Internal	External
Temporal	▬	▬
Data Value	×	●
Functional	×	×
Structural	×	▭
SW Programming Level		▭

Symbol Key

▪	Model resolves information at specific level
▬	Model resolves information at any of the levels spanned, case dependent
▭	Model optionally resolves information at levels spanned
▨	Model resolves partial information at levels spanned, such as control but not data values or functionality
×	Model does not contain information on attribute

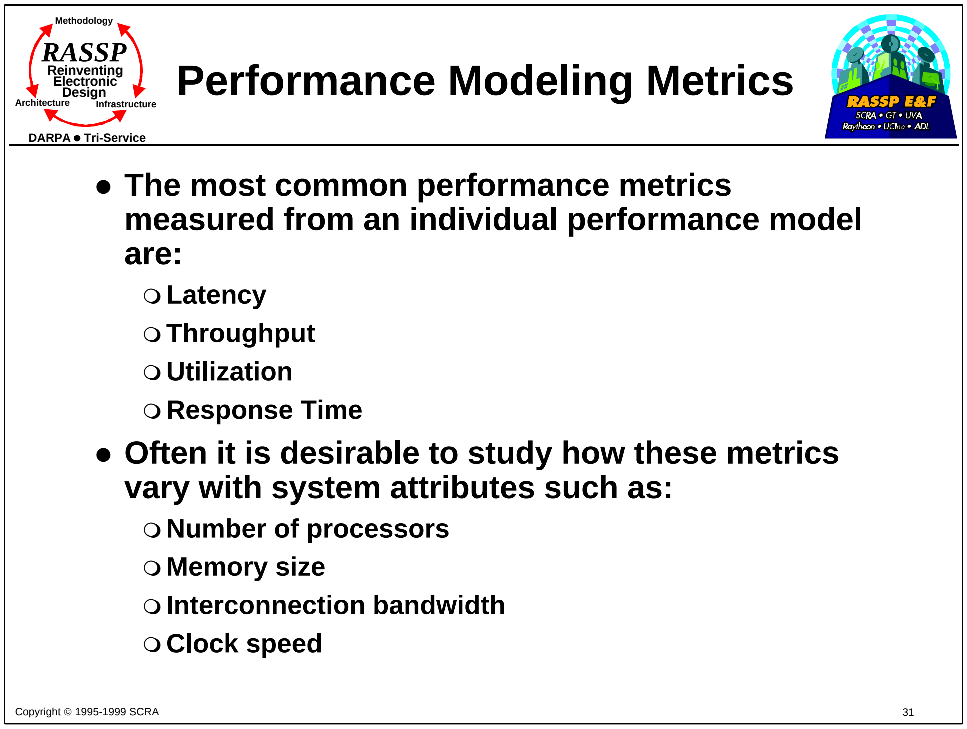
Copyright © 1995-1999 SCRA

Copyright © 1997-98 RASSP Taxonomy Working Group used with permission

[Hein97] 30

General performance models have temporal data (both internal and external) that can be at essentially any level of abstraction. They have no internal data value information, and only high level external data value information (e.g. memory address, size, etc.), no functional information and only external structural information. Software can be represented at any level.

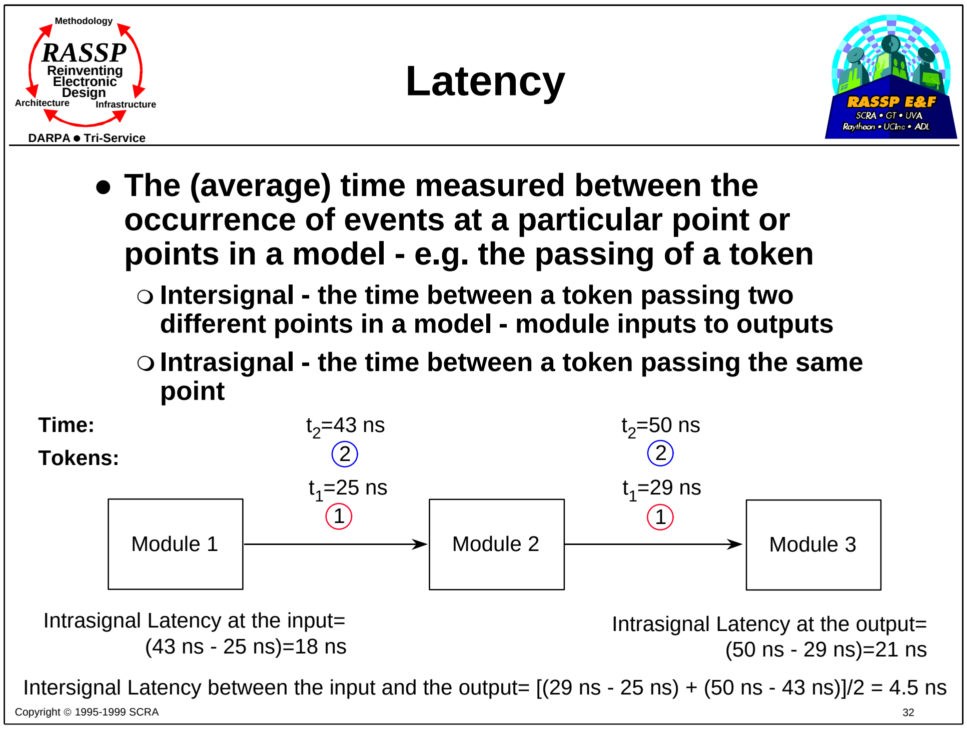
Token-based performance models have higher levels of timing information (e.g. at the task level or data packet level, not instruction level or individual word level), and higher levels of external structure. Software is represented at the task level and above.



Performance Modeling Metrics

- The most common performance metrics measured from an individual performance model are:
  - ❍ Latency
  - ❍ Throughput
  - ❍ Utilization
  - ❍ Response Time
- Often it is desirable to study how these metrics vary with system attributes such as:
  - ❍ Number of processors
  - ❍ Memory size
  - ❍ Interconnection bandwidth
  - ❍ Clock speed

This section will present the classical performance modeling metrics of latency, throughput, utilization, and others.



Latency

- The (average) time measured between the occurrence of events at a particular point or points in a model - e.g. the passing of a token
  - ❍ Intersignal - the time between a token passing two different points in a model - module inputs to outputs
  - ❍ Intrasignal - the time between a token passing the same point

Intrasignal Latency at the input=
(43 ns - 25 ns)=18 ns

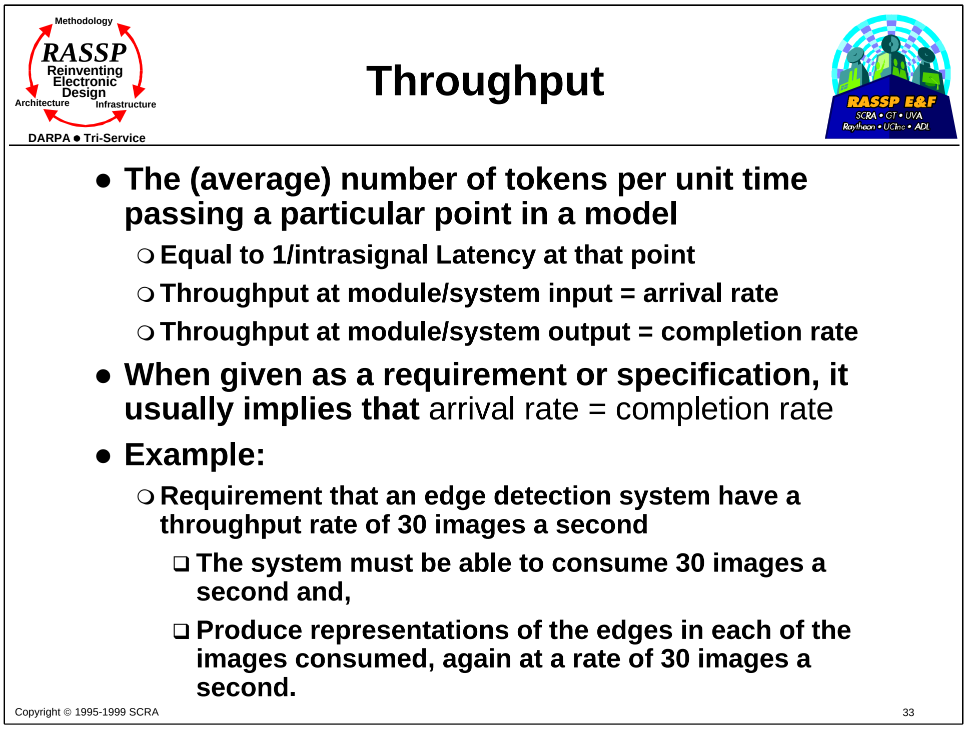
Intrasignal Latency at the output=
(50 ns - 29 ns)=21 ns

Intersignal Latency between the input and the output= [(29 ns - 25 ns) + (50 ns - 43 ns)]/2 = 4.5 ns

Latency is the time between two events.

Usually, latency is the time between two events on different signals, or in different parts of the model, e.g., the time between the arrival of a memory request and a memory access - memory latency, or the time between the sending and receiving of a message - communications latency. For lack of a better term, this is called intersignal latency.

Sometimes however, the latency between events on the same signal is important, e.g., the time between subsequent memory accesses or the time between the processing of RADAR pulses by a SAR system. This is termed intrasignal latency.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Throughput

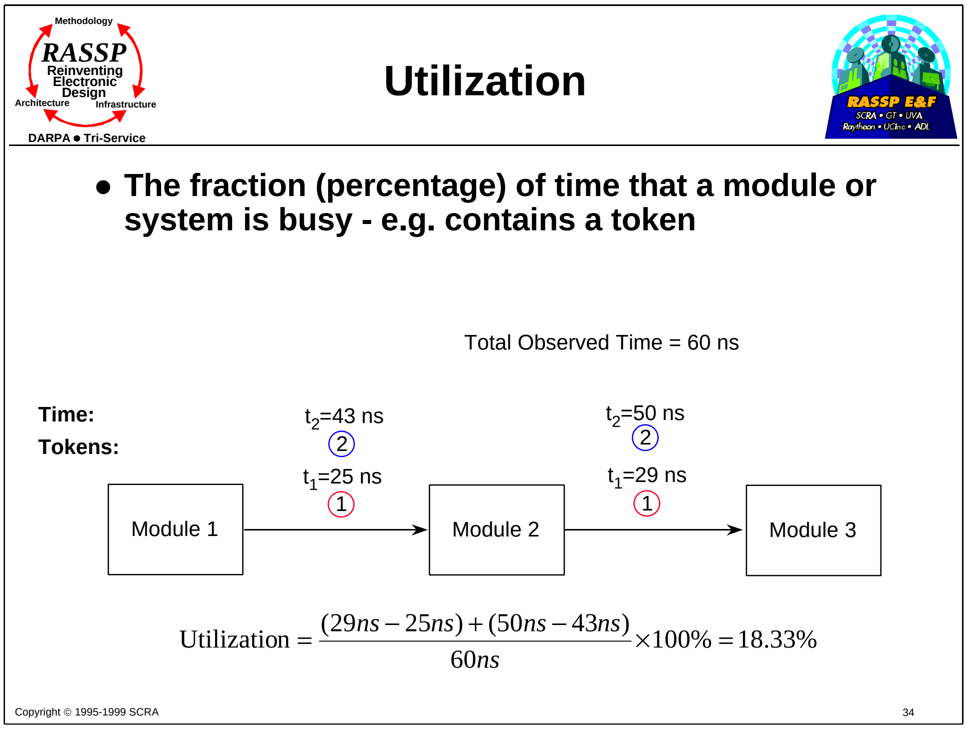
- **The (average) number of tokens per unit time passing a particular point in a model**
  - ❍ **Equal to 1/intrasignal Latency at that point**
  - ❍ **Throughput at module/system input = arrival rate**
  - ❍ **Throughput at module/system output = completion rate**
- **When given as a requirement or specification, it usually implies that** arrival rate = completion rate
- **Example:**
  - ❍ **Requirement that an edge detection system have a throughput rate of 30 images a second**
    - ❑ **The system must be able to consume 30 images a second and,**
    - ❑ **Produce representations of the edges in each of the images consumed, again at a rate of 30 images a second.**

33

Throughput is basically 1/some type of latency.

Arrival rate is 1/ the intrasignal latency at the system’s input, Completion rate is 1/ the intrasignal latency at the system’s output.

When used as a requirement, throughput usually means completion rate.



Utilization

- The fraction (percentage) of time that a module or system is busy - e.g. contains a token

Total Observed Time = 60 ns

Time:

Tokens:

t_2=43 ns
2
t_1=25 ns
1

t_2=50 ns
2
t_1=29 ns
1

Module 1

Module 2

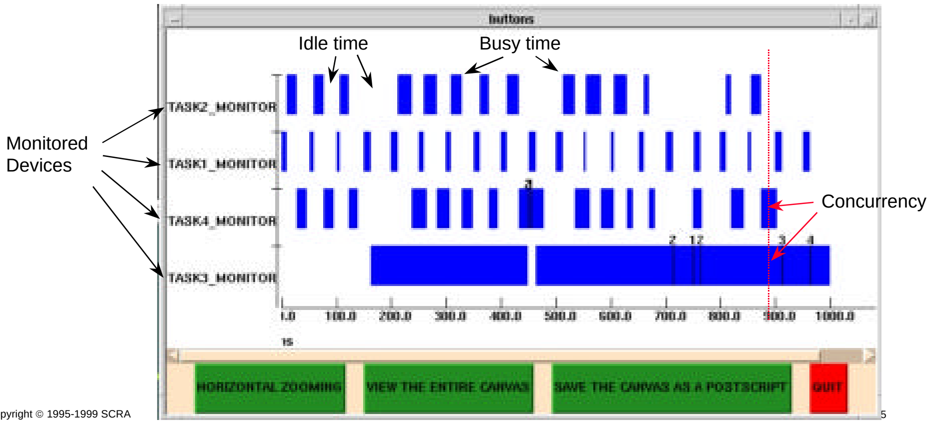
Module 3

$$\text{Utilization} = \frac{(29ns - 25ns) + (50ns - 43ns)}{60ns} \times 100\% = 18.33\%$$

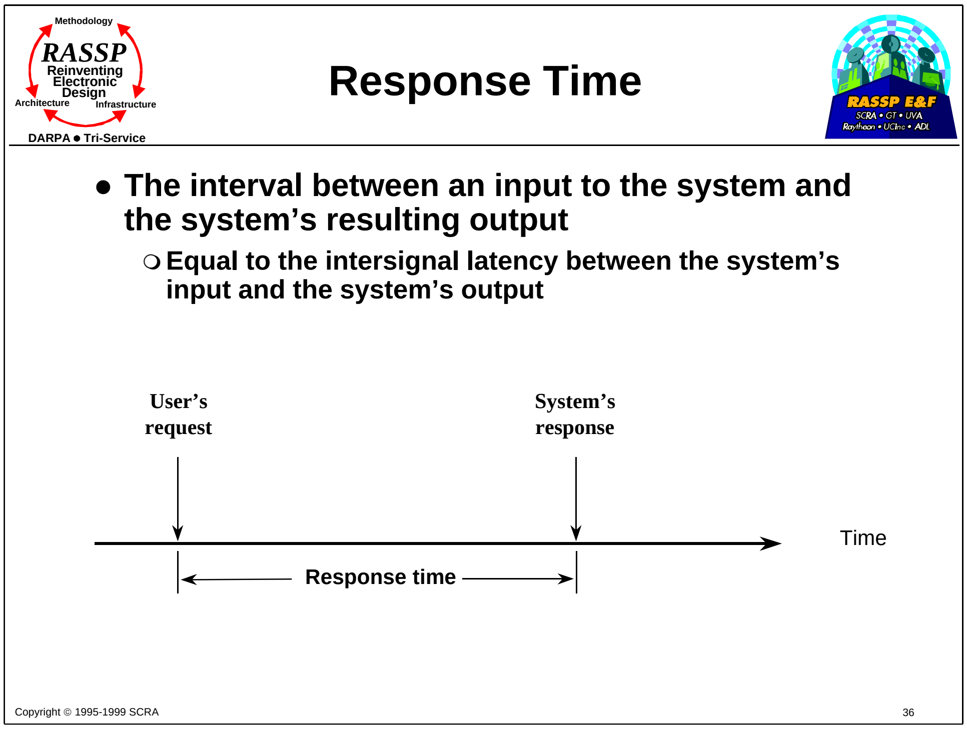
Utilization is simply the percentage of time (that the system is simulated for) that the system is actually busy i.e., it contains a token.

Utilization (Cont.)

- **Activity Time Lines**
  - **Display individual device utilization as horizontal bar graphs**
  - **Useful in visualizing idle time and concurrency**



Activity time lines are a helpful way to visualize utilization, especially in a system where some concurrency is possible because they allow that concurrency to be visualized. This helps to see points where concurrency is or isn’t happening



Response Time

- **The interval between an input to the system and the system's resulting output**
  - ❍ **Equal to the intersignal latency between the system's input and the system's output**

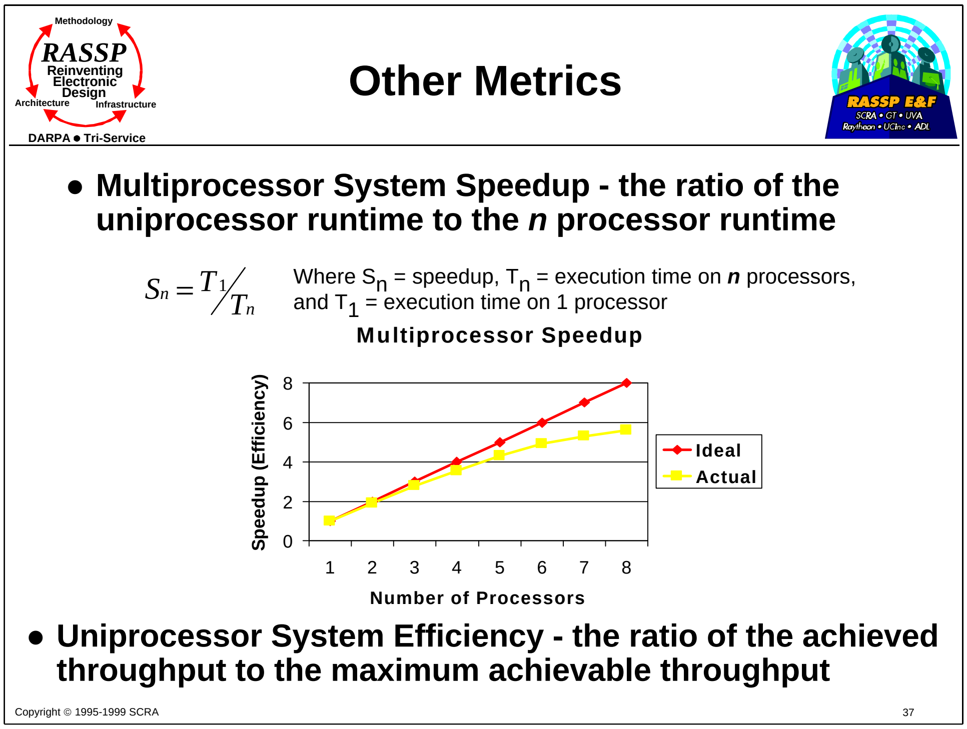
User's request

System's response

Time

Response time

Response time is a metric that is sometimes used in "user driven systems" because it measures how long the user must wait from their input to the desired output.



Other Metrics

- **Multiprocessor System Speedup - the ratio of the uniprocessor runtime to the *n* processor runtime**

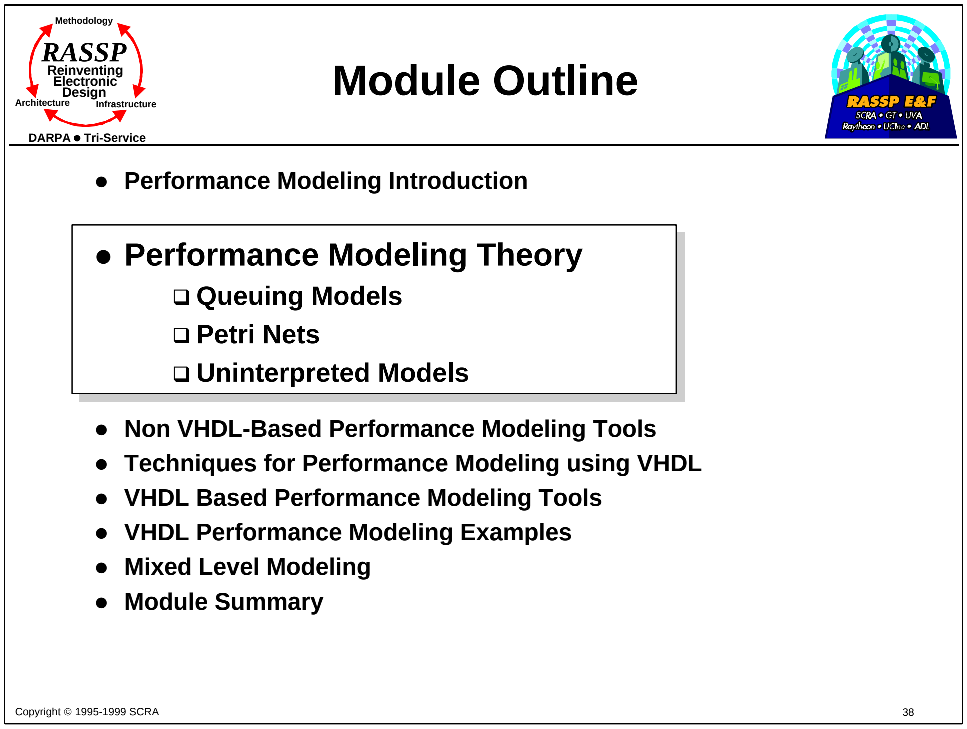
$$S_n = {}^{T_1}\!/_{T_n}$$

Where S_n = speedup, T_n = execution time on ***n*** processors, and T_1 = execution time on 1 processor

Multiprocessor Speedup

- **Uniprocessor System Efficiency - the ratio of the achieved throughput to the maximum achievable throughput**

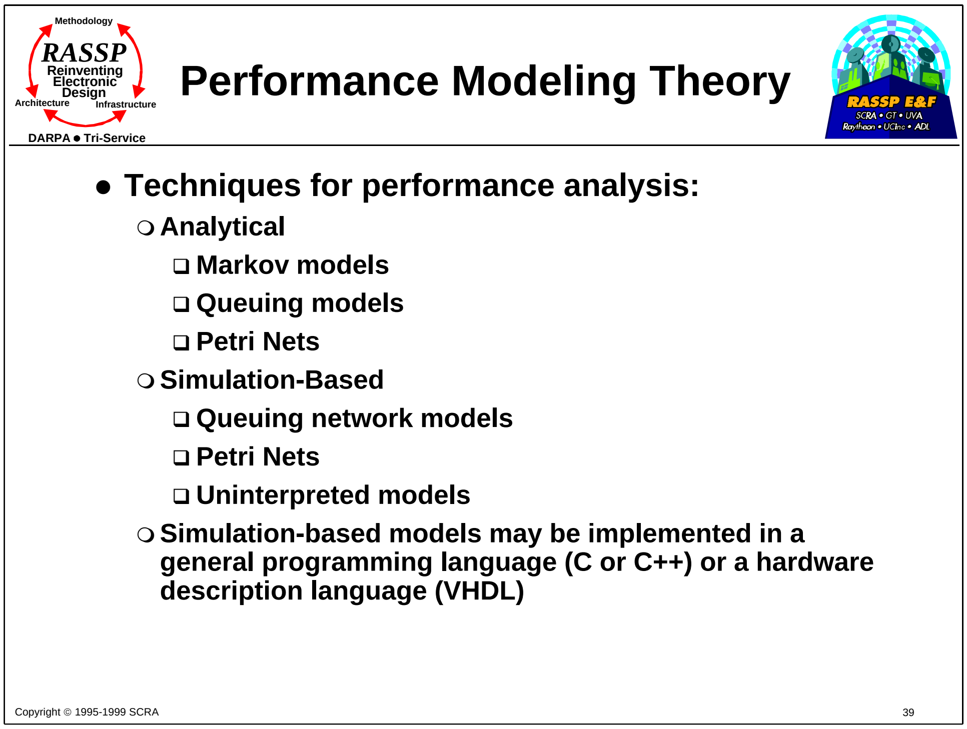
Other metrics typically used in system performance analysis include speedup for a multiprocessor system, and efficiency for a uniprocessor system (these two are related in that they are both basically the ratio of the achieved throughput to the theoretical maximum throughput, or visa versa).



Module Outline

- Performance Modeling Introduction
- **Performance Modeling Theory**
  - ❑ Queuing Models
  - ❑ Petri Nets
  - ❑ Uninterpreted Models
- Non VHDL-Based Performance Modeling Tools
- Techniques for Performance Modeling using VHDL
- VHDL Based Performance Modeling Tools
- VHDL Performance Modeling Examples
- Mixed Level Modeling
- Module Summary

Module Outline

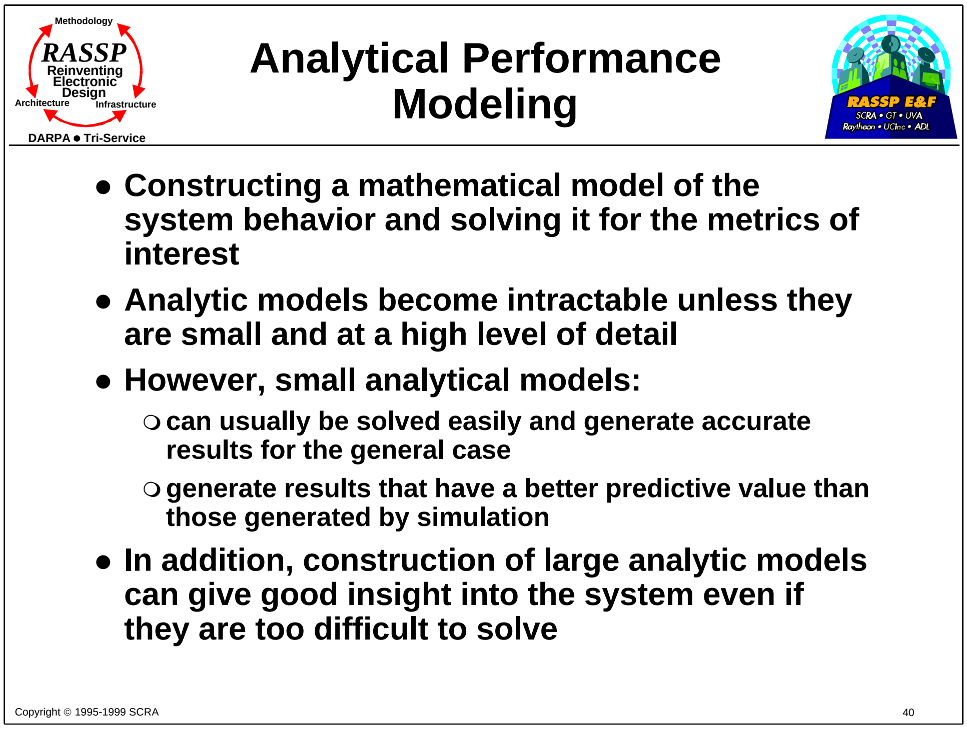

Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Performance Modeling Theory

- Techniques for performance analysis:
  - ❍ Analytical
    - ❑ Markov models
    - ❑ Queuing models
    - ❑ Petri Nets
  - ❍ Simulation-Based
    - ❑ Queuing network models
    - ❑ Petri Nets
    - ❑ Uninterpreted models
  - ❍ Simulation-based models may be implemented in a general programming language (C or C++) or a hardware description language (VHDL)

There are two basic techniques for performance modeling, analytical, and simulation-based. The advantages and disadvantages of each will be explained in each section.

Token-based performance modeling using VHDL is a simulation-based technique, but the analytical techniques will be introduced here to provide background for the simulation-based techniques. This section of the module can be omitted from discussion if this background material is not required for the given audience.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

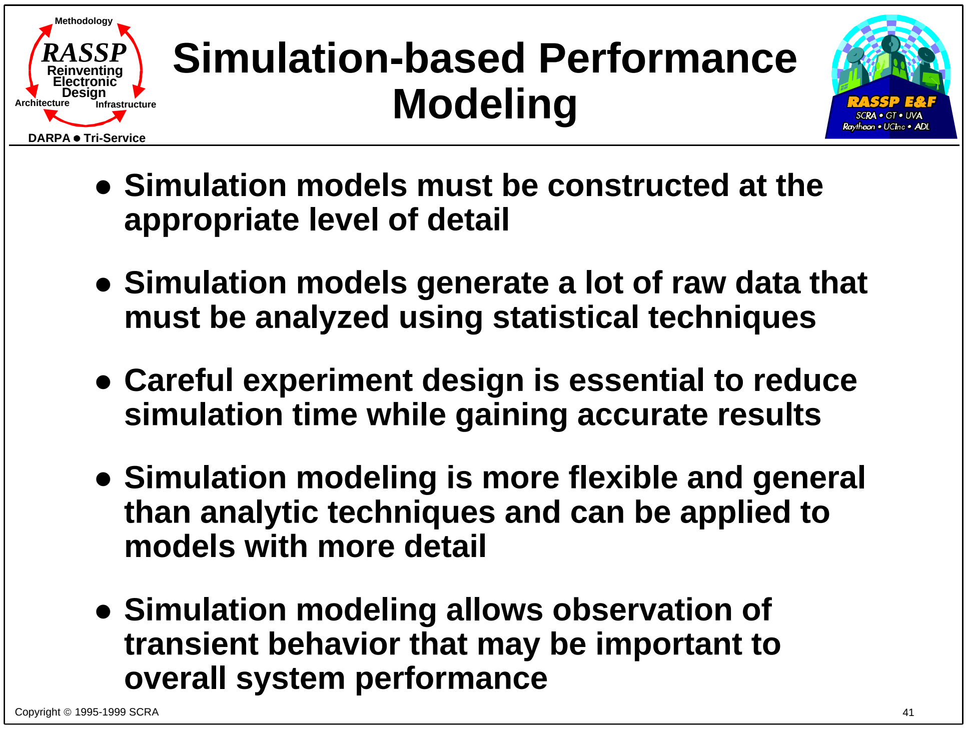
Analytical Performance Modeling

- Constructing a mathematical model of the system behavior and solving it for the metrics of interest
- Analytic models become intractable unless they are small and at a high level of detail
- However, small analytical models:
  - ❍ can usually be solved easily and generate accurate results for the general case
  - ❍ generate results that have a better predictive value than those generated by simulation
- In addition, construction of large analytic models can give good insight into the system even if they are too difficult to solve

40

Analytical performance modeling techniques consist of constructing and solving a mathematical model of the system. Their main advantage is their accuracy and the speed with which they can be solved. Their main disadvantage is the fact that they become intractable for all but the smallest systems.

[Kant92]


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Simulation-based Performance Modeling

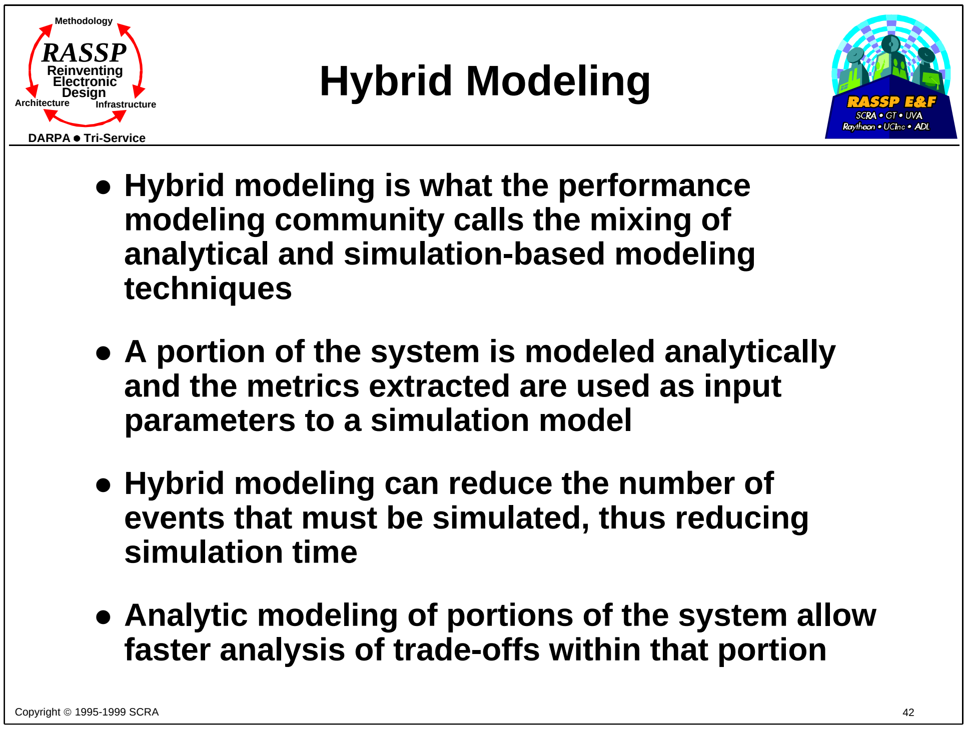
- Simulation models must be constructed at the appropriate level of detail
- Simulation models generate a lot of raw data that must be analyzed using statistical techniques
- Careful experiment design is essential to reduce simulation time while gaining accurate results
- Simulation modeling is more flexible and general than analytic techniques and can be applied to models with more detail
- Simulation modeling allows observation of transient behavior that may be important to overall system performance

41

Simulation-based techniques consist of constructing and executing a model of the system in a high-level programming language or hardware description language (hdl). Simulation-based models are more generally applicable and can handle larger systems. The simulation execution time can become excessive for very complex systems however, if the level of detail of the model becomes too high. Unlike analytical models, which just give indications of system steady-state behavior, simulation-based models allow observation of the transient behavior of the system which may be important.

In addition, simulation-based models typically generate large amounts of data that have to be analyzed using statistical techniques.

[Kant92]

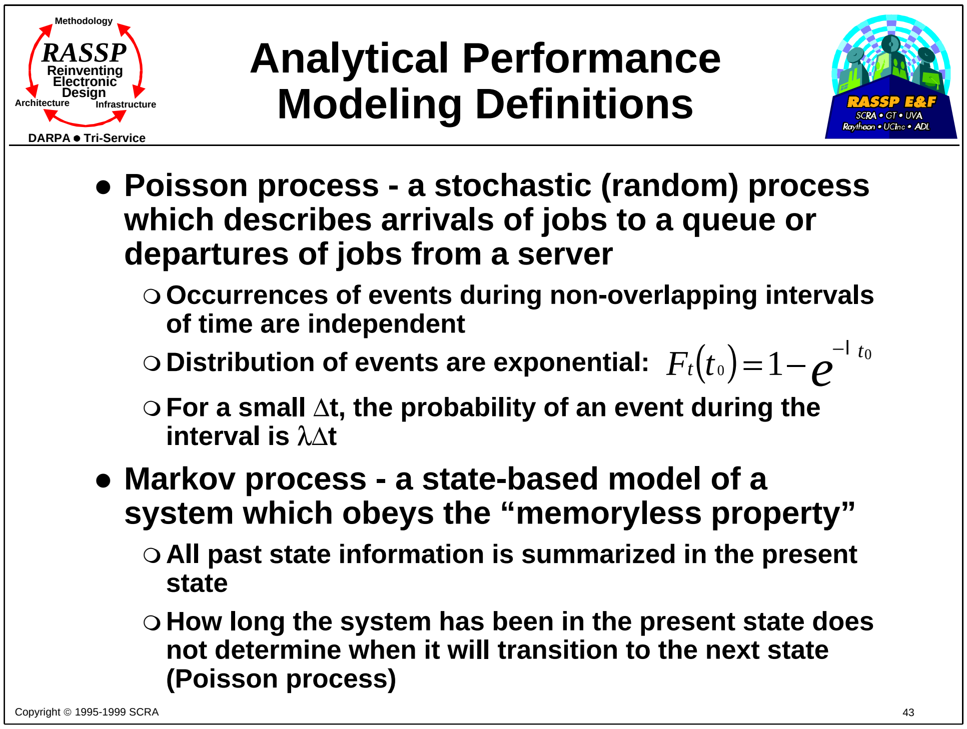


Hybrid Modeling

- Hybrid modeling is what the performance modeling community calls the mixing of analytical and simulation-based modeling techniques
- A portion of the system is modeled analytically and the metrics extracted are used as input parameters to a simulation model
- Hybrid modeling can reduce the number of events that must be simulated, thus reducing simulation time
- Analytic modeling of portions of the system allow faster analysis of trade-offs within that portion

Hybrid modeling is the term used in the queuing model and Petri Net community to describe mixed analytical and simulation based performance modeling. It is a somewhat overloaded term in that hybrid modeling has also been used to describe the mixture of performance and behavioral models although the preferred term for that is “mixed level modeling.”

Hybrid modeling attempts to incorporate the benefits of both analytical and simulation-based modeling techniques.



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

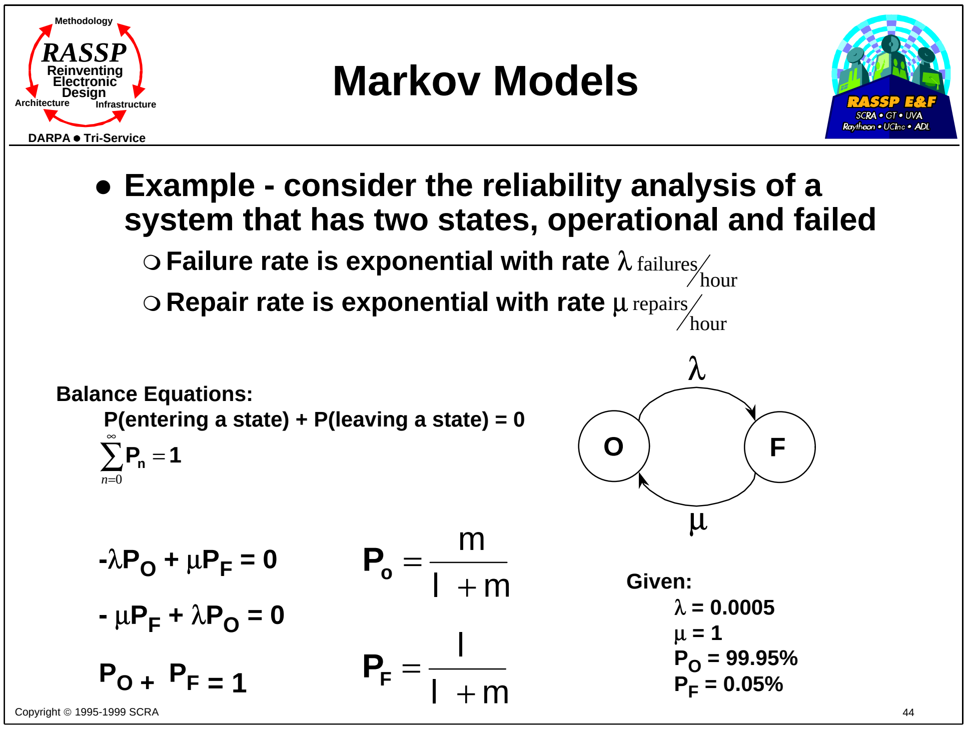
Analytical Performance Modeling Definitions

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Poisson process - a stochastic (random) process which describes arrivals of jobs to a queue or departures of jobs from a server**
  - ❍ **Occurrences of events during non-overlapping intervals of time are independent**
  - ❍ **Distribution of events are exponential:** $F_t(t_0)=1-e^{-l\ t_0}$
  - ❍ **For a small Δt, the probability of an event during the interval is λΔt**
- **Markov process - a state-based model of a system which obeys the "memoryless property"**
  - ❍ **All past state information is summarized in the present state**
  - ❍ **How long the system has been in the present state does not determine when it will transition to the next state (Poisson process)**

Most analytical performance modeling techniques are based on a Poisson process. This is a stochastic process in which the distribution of events are exponential and occurrences of events in non-overlapping time intervals are independent. Because of this property, the probability that an event occurs in a small interval of time is proportional to the probability distribution.

The Markov model is the basic modeling paradigm. A Markov model is a state based model where the probability of transitioning from one state to another is a Poisson process. This allows the model to be easily solved as will be seen.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL
λ
O
F
μ

Markov Models

- Example - consider the reliability analysis of a system that has two states, operational and failed
  - ❍ Failure rate is exponential with rate λ failures/hour
  - ❍ Repair rate is exponential with rate μ repairs/hour

Balance Equations:

P(entering a state) + P(leaving a state) = 0

$$\sum_{n=0}^{\infty} P_n = 1$$

$$-\lambda P_O + \mu P_F = 0$$

$$-\mu P_F + \lambda P_O = 0$$

$$P_O + P_F = 1$$

$$P_o = \frac{m}{l + m}$$

$$P_F = \frac{l}{l + m}$$

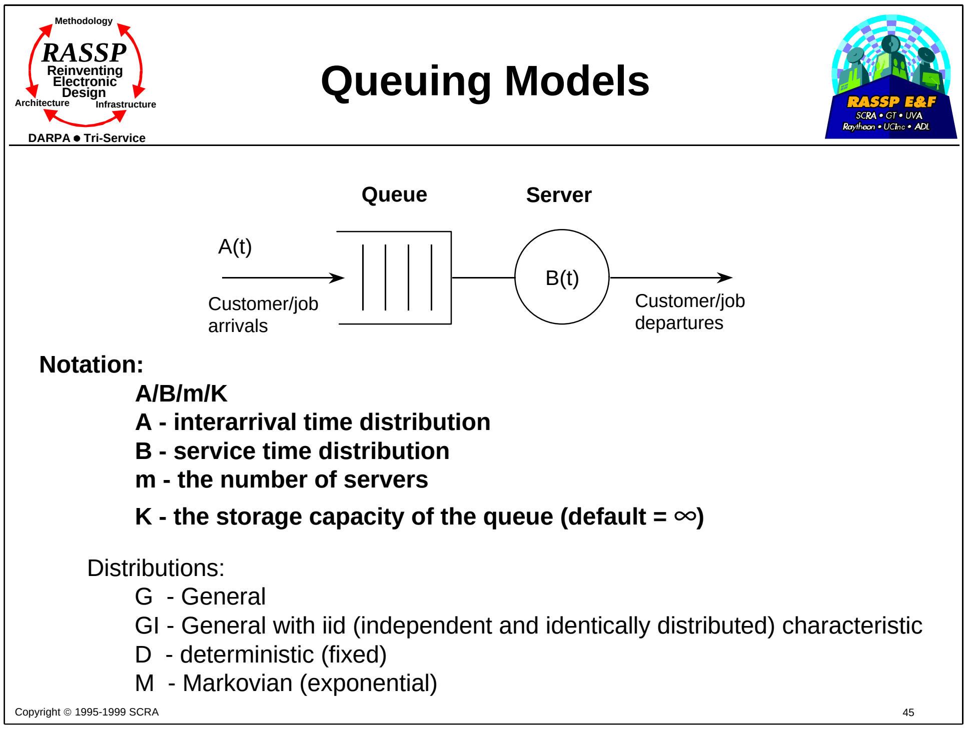
Given:

$\lambda = 0.0005$
$\mu = 1$
$P_O = 99.95\%$
$P_F = 0.05\%$

44

This simple two state example (even though it is derived from reliability analysis) shows how a Markov model is solved.

Balance equations that are derived from the fact that the sum of all probabilities entering a state must be equal to all probabilities leaving that state and all probabilities must sum to 1. These balance equations can then be solved to determine the probability of being in each state.



Queuing Models

Notation:

A/B/m/K

A - interarrival time distribution

B - service time distribution

m - the number of servers

K - the storage capacity of the queue (default = ∞)

Distributions:

G - General

GI - General with iid (independent and identically distributed) characteristic

D - deterministic (fixed)

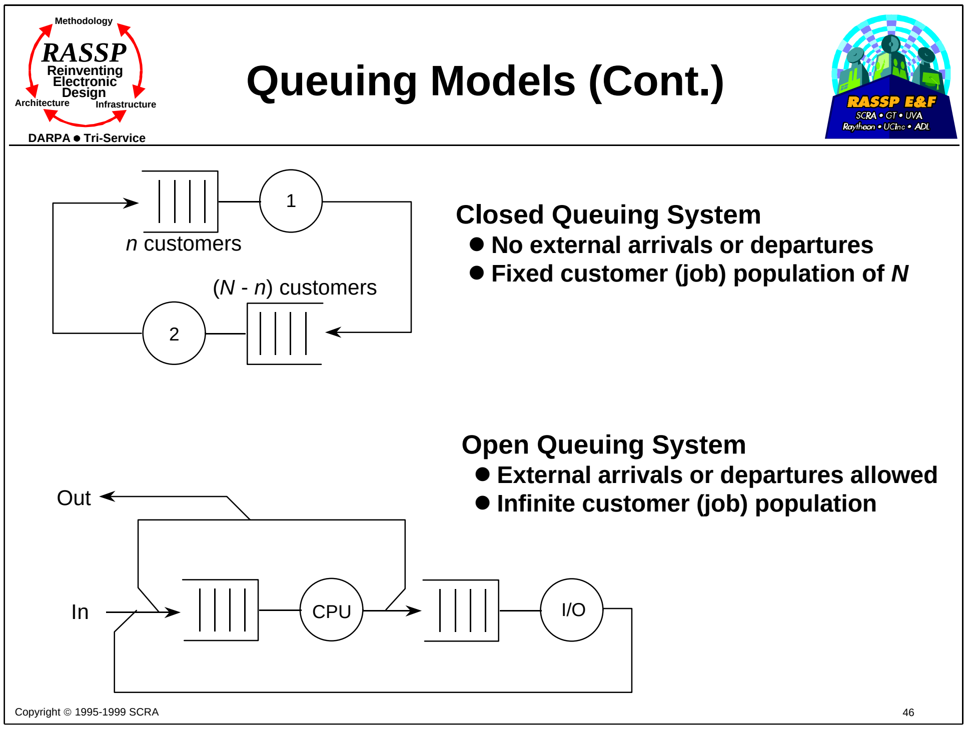
M - Markovian (exponential)

This slide describes the convention with which queues are specified.

The discussion here will be limited to M/M queues since they can be described as Markov models, as will be shown.

iid - independent and identically distributed

[Cassandras93] has probably the best description of queuing networks and how they can be analyzed as Markov models, but [Sauer81] is also good and has some good examples.



Queuing Models (Cont.)

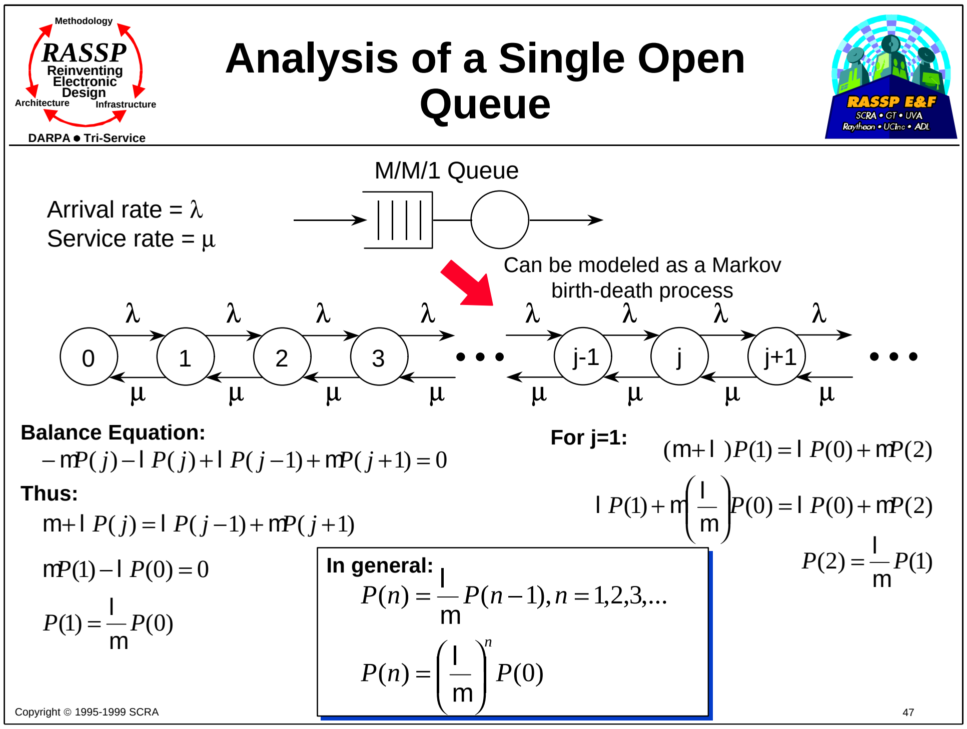
Closed Queuing System

- No external arrivals or departures
- Fixed customer (job) population of ***N***

Open Queuing System

- External arrivals or departures allowed
- Infinite customer (job) population

Both open and closed queuing networks can be analyzed, but there must be some restrictions on the arrivals and departures in an open queuing network so that it may be analyzed using these techniques.



Analysis of a Single Open Queue

Arrival rate = λ
Service rate = μ

M/M/1 Queue
Can be modeled as a Markov birth-death process
λ
μ
0
1
2
3
j-1
j
j+1
• • •

Balance Equation:

$$-mP(j)-l\,P(j)+l\,P(j-1)+mP(j+1)=0$$

Thus:

$$m+l\,P(j)=l\,P(j-1)+mP(j+1)$$

$$mP(1)-l\,P(0)=0$$

$$P(1)=\frac{l}{m}P(0)$$

For j=1:

$$(m+l\,)P(1)=l\,P(0)+mP(2)$$

$$l\,P(1)+m\left(\frac{l}{m}\right)P(0)=l\,P(0)+mP(2)$$

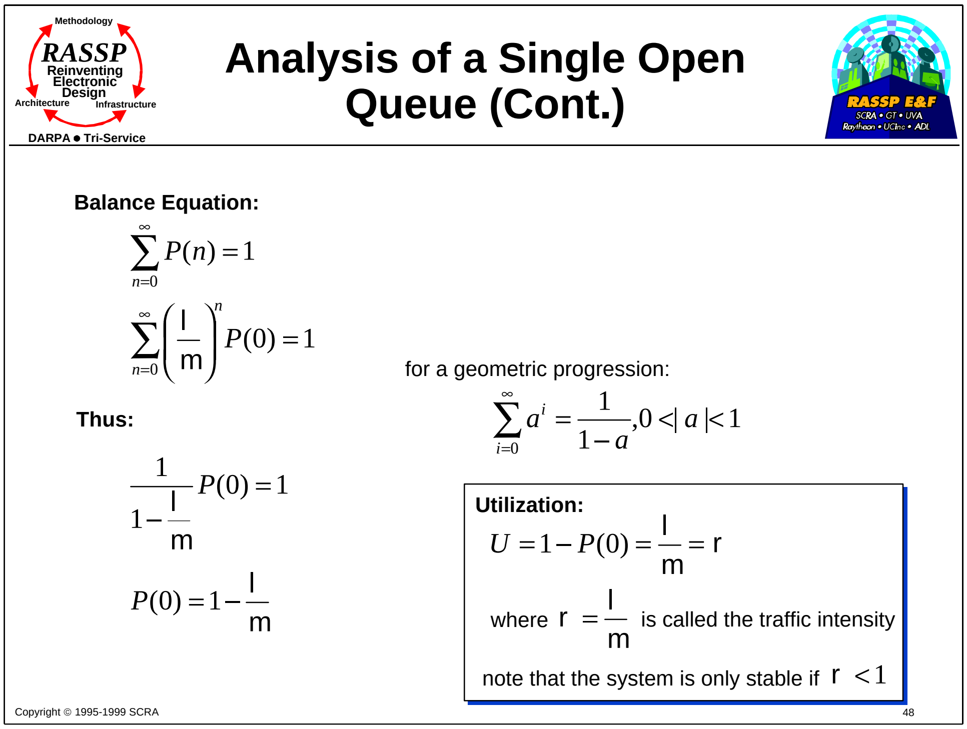
$$P(2)=\frac{l}{m}P(1)$$

In general:

$$P(n)=\frac{l}{m}P(n-1),n=1,2,3,\ldots$$

$$P(n)=\left(\frac{l}{m}\right)^{n}P(0)$$

This slides shows the analysis if a single queue if infinite size with exponential arrival and service rates. As shown, the queue can be modeled with a Markov birth-death process. This allows the steady state behavior of the queue to be modeled analytically. Note that the service rate must be greater than the arrival rate for the model to be stable.



Analysis of a Single Open Queue (Cont.)

Balance Equation:

$$\sum_{n=0}^{\infty} P(n) = 1$$

$$\sum_{n=0}^{\infty} \left(\frac{l}{m}\right)^{n} P(0) = 1$$

for a geometric progression:

$$\sum_{i=0}^{\infty} a^{i} = \frac{1}{1-a}, 0 < |a| < 1$$

Thus:

$$\frac{1}{1-\frac{l}{m}} P(0) = 1$$

$$P(0) = 1 - \frac{l}{m}$$

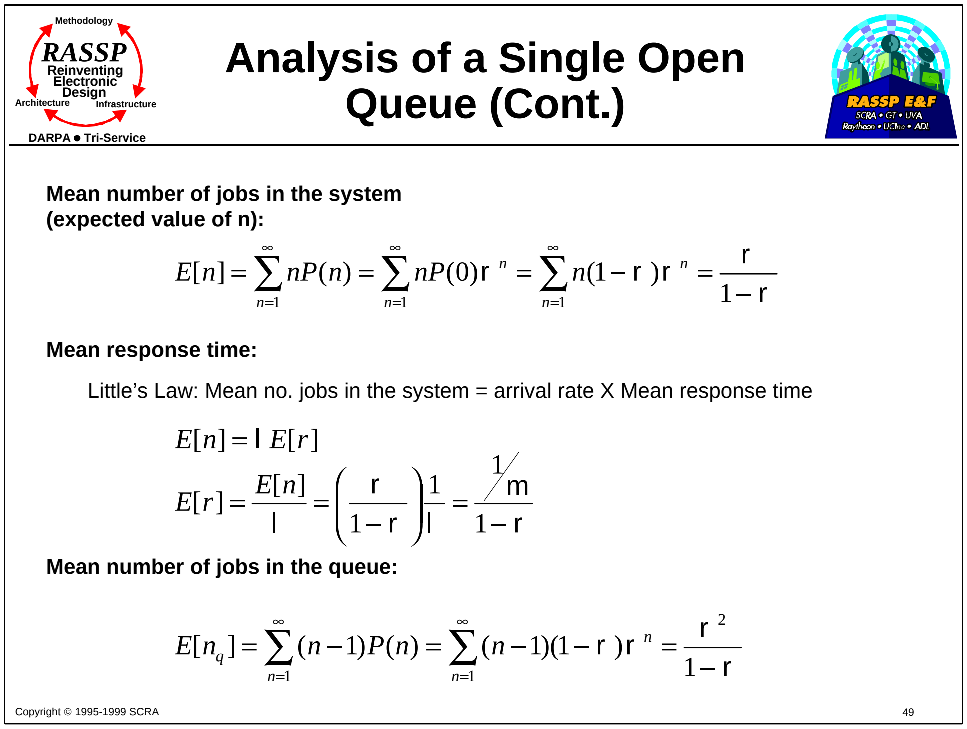
Utilization:

$$U = 1 - P(0) = \frac{l}{m} = r$$

where $r = \frac{l}{m}$ is called the traffic intensity

note that the system is only stable if $r < 1$

This is calculation of utilization of the server in the single queue system.



Analysis of a Single Open Queue (Cont.)

Mean number of jobs in the system (expected value of n):

$$E[n] = \sum_{n=1}^{\infty} nP(n) = \sum_{n=1}^{\infty} nP(0)\mathrm{r}^{n} = \sum_{n=1}^{\infty} n(1-\mathrm{r})\mathrm{r}^{n} = \frac{\mathrm{r}}{1-\mathrm{r}}$$

Mean response time:

Little’s Law: Mean no. jobs in the system = arrival rate X Mean response time

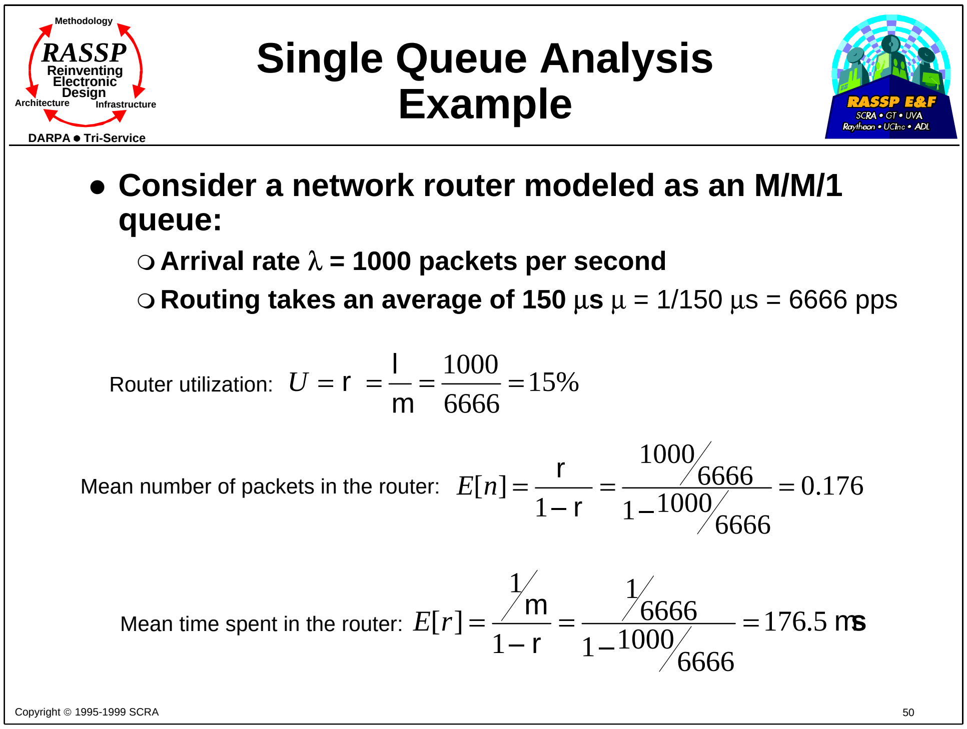
$$E[n] = \mathrm{l}\, E[r]$$

$$E[r] = \frac{E[n]}{\mathrm{l}} = \left(\frac{\mathrm{r}}{1-\mathrm{r}}\right)\frac{1}{\mathrm{l}} = \frac{{}^{1}\!/_{\mathrm{m}}}{1-\mathrm{r}}$$

Mean number of jobs in the queue:

$$E[n_q] = \sum_{n=1}^{\infty} (n-1)P(n) = \sum_{n=1}^{\infty} (n-1)(1-\mathrm{r})\mathrm{r}^{n} = \frac{\mathrm{r}^{2}}{1-\mathrm{r}}$$

This is the calculation of mean number of jobs in the system, mean response time, and mean number of jobs in the queue. Note that this slide introduces Little’s Law, an important theorem in queue analysis.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Single Queue Analysis Example

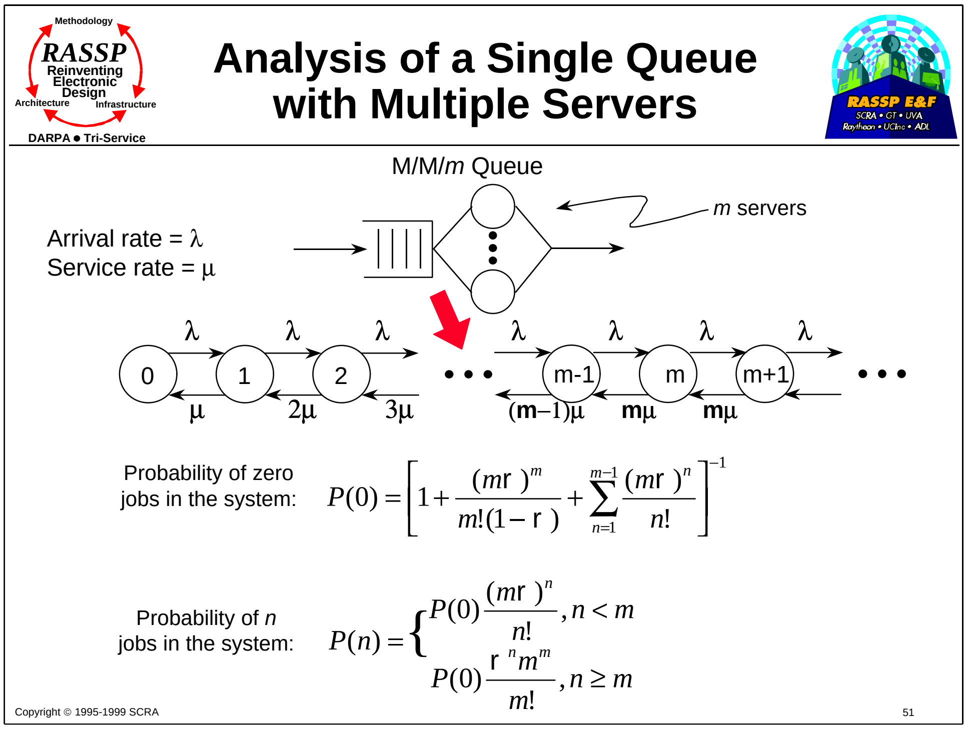
- **Consider a network router modeled as an M/M/1 queue:**
  - ❍ **Arrival rate λ = 1000 packets per second**
  - ❍ **Routing takes an average of 150 µs** µ = 1/150 µs = 6666 pps

Router utilization: $U = r = \frac{l}{m} = \frac{1000}{6666} = 15\%$

Mean number of packets in the router: $E[n] = \frac{r}{1-r} = \frac{1000/6666}{1-1000/6666} = 0.176$

Mean time spent in the router: $E[r] = \frac{1/m}{1-r} = \frac{1/6666}{1-1000/6666} = 176.5\ ms$

This is an example of how a real life system can be analyzed as a M/M/1 queue. Note that the analysis of a system with a limited queue size (M/M/1/N), which covers more real-life systems, is equally simple.



Analysis of a Single Queue with Multiple Servers

M/M/*m* Queue

Arrival rate = λ
Service rate = μ

m servers

Probability of zero jobs in the system:

$$P(0) = \left[1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$$

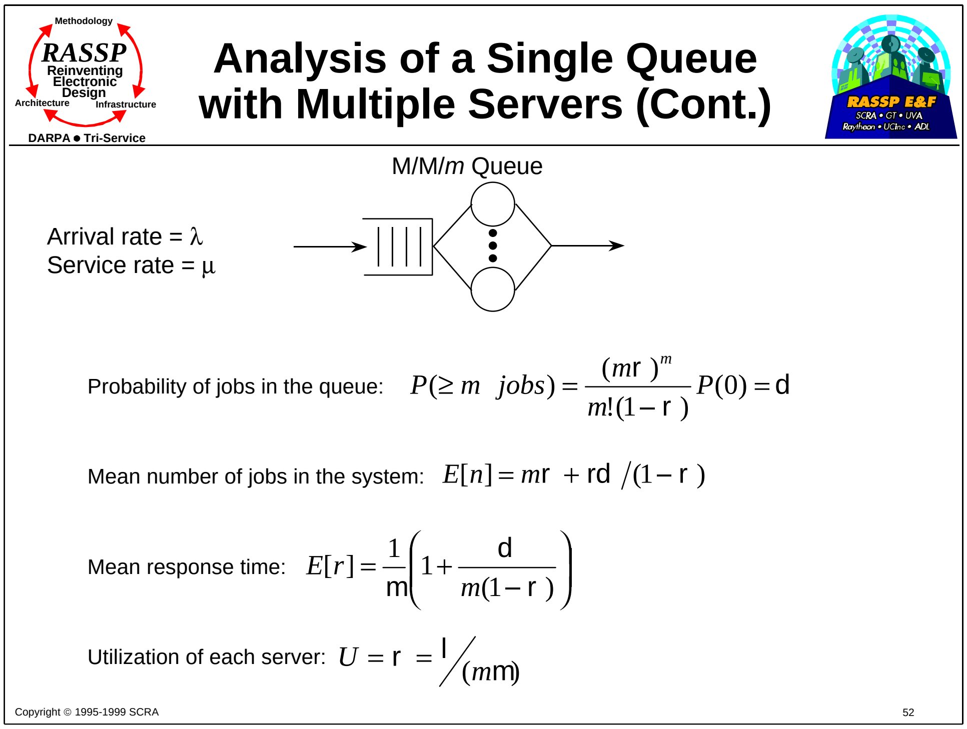
Probability of *n* jobs in the system:

$$P(n) = \begin{cases} P(0)\dfrac{(m\rho)^n}{n!}, n < m \\ P(0)\dfrac{\rho^n m^m}{m!}, n \geq m \end{cases}$$

Copyright © 1995-1999 SCRA

51

This is the analysis if an M/M/n system, one with a single exponential queue but multiple servers, e.g. a multiprocessor system for transaction processing.



Analysis of a Single Queue with Multiple Servers (Cont.)

M/M/*m* Queue

Arrival rate = λ
Service rate = μ

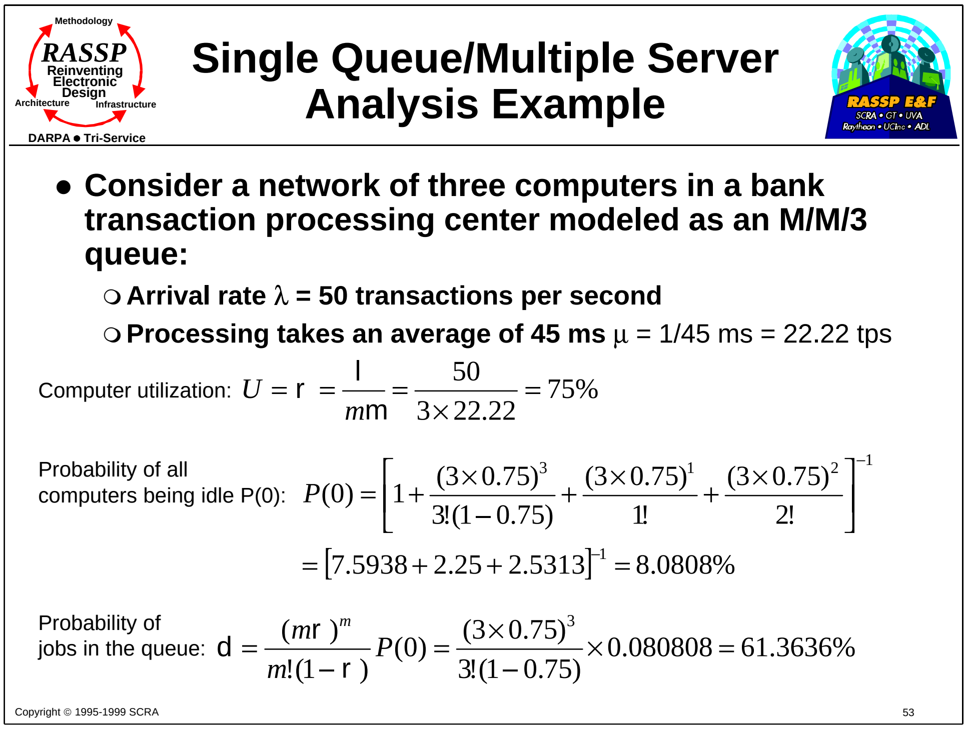
Probability of jobs in the queue: $P(\geq m \;\; jobs) = \frac{(m\rho)^m}{m!(1-\rho)} P(0) = \varpi$

Mean number of jobs in the system: $E[n] = m\rho + \rho\varpi / (1-\rho)$

Mean response time: $E[r] = \frac{1}{\mu}\left(1 + \frac{\varpi}{m(1-\rho)}\right)$

Utilization of each server: $U = \rho = \lambda / (m\mu)$

This is the remainder of the analysis.



Single Queue/Multiple Server Analysis Example

- **Consider a network of three computers in a bank transaction processing center modeled as an M/M/3 queue:**
  - **Arrival rate λ = 50 transactions per second**
  - **Processing takes an average of 45 ms** μ = 1/45 ms = 22.22 tps

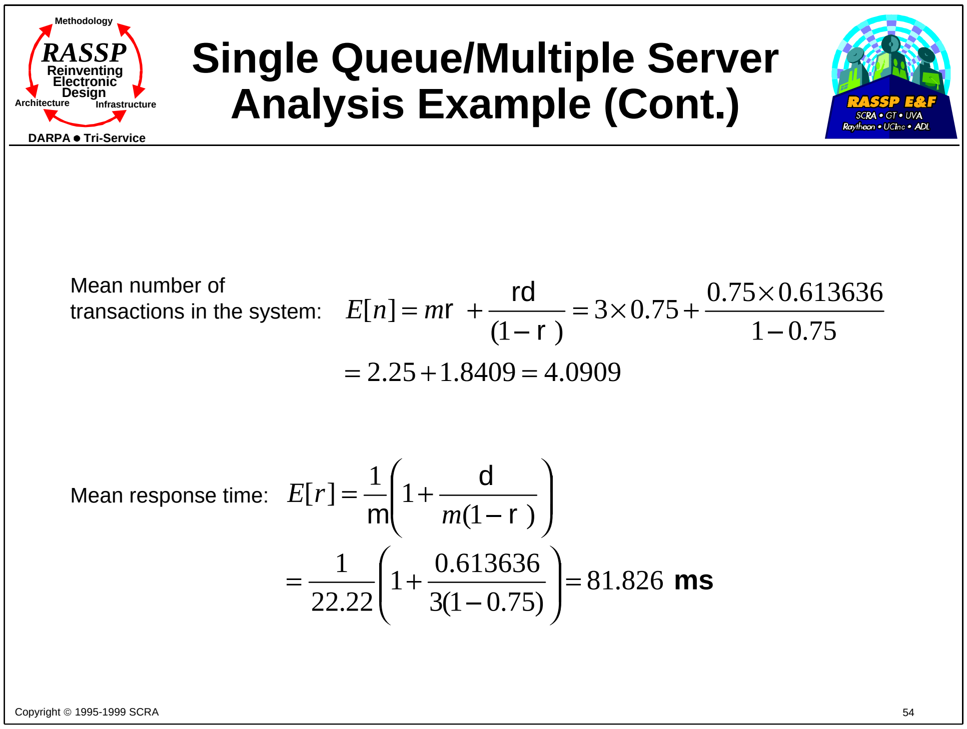
Computer utilization: $U = r = \frac{l}{m m} = \frac{50}{3 \times 22.22} = 75\%$

Probability of all computers being idle P(0): $P(0) = \left[1 + \frac{(3 \times 0.75)^3}{3!(1-0.75)} + \frac{(3 \times 0.75)^1}{1!} + \frac{(3 \times 0.75)^2}{2!} \right]^{-1}$

$$= [7.5938 + 2.25 + 2.5313]^{-1} = 8.0808\%$$

Probability of jobs in the queue: $d = \frac{(m r)^m}{m!(1 - r)} P(0) = \frac{(3 \times 0.75)^3}{3!(1-0.75)} \times 0.080808 = 61.3636\%$

An example of an M/M/n queue model.



Single Queue/Multiple Server Analysis Example (Cont.)

Mean number of transactions in the system:

$$E[n] = m\mathrm{r} + \frac{\mathrm{rd}}{(1-\mathrm{r})} = 3\times 0.75 + \frac{0.75\times 0.613636}{1-0.75}$$

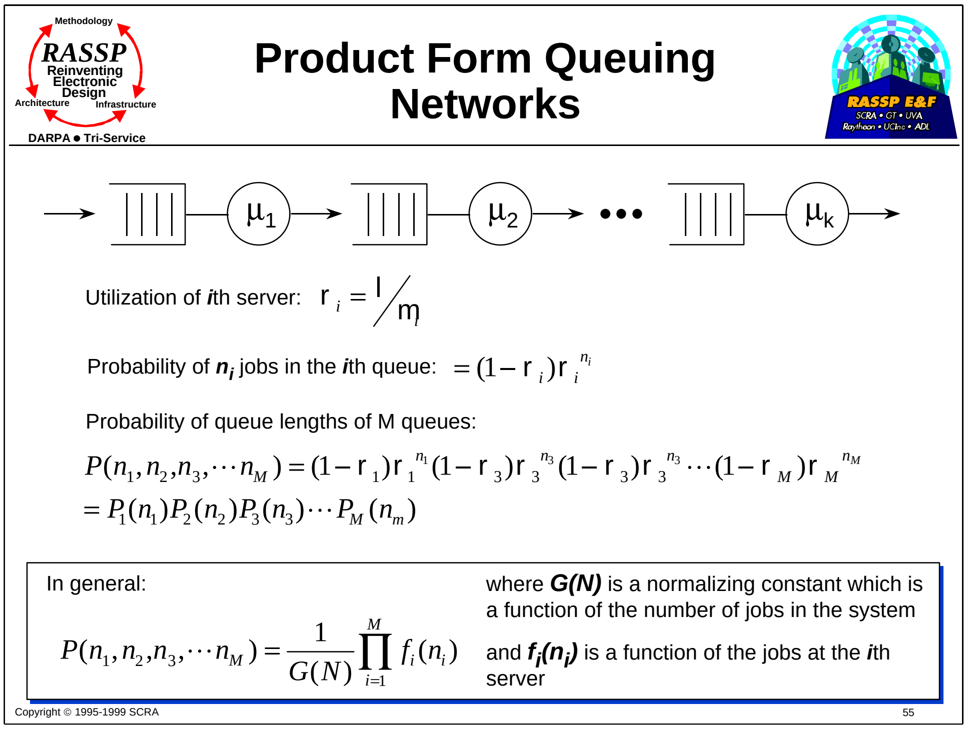
$$= 2.25 + 1.8409 = 4.0909$$

Mean response time:

$$E[r] = \frac{1}{\mathrm{m}}\left(1+\frac{\mathrm{d}}{m(1-\mathrm{r})}\right)$$

$$= \frac{1}{22.22}\left(1+\frac{0.613636}{3(1-0.75)}\right) = 81.826\ \textbf{ms}$$

M/M/n example continued.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

Product Form Queuing Networks

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL
μ1
μ2
•••
μk

Utilization of *i*th server: $r_i = l \big/ m_i$

Probability of $\boldsymbol{n_i}$ jobs in the *i*th queue: $= (1 - r_i) r_i^{n_i}$

Probability of queue lengths of M queues:

$$P(n_1, n_2, n_3, \cdots n_M) = (1 - r_1) r_1^{n_1} (1 - r_3) r_3^{n_3} (1 - r_3) r_3^{n_3} \cdots (1 - r_M) r_M^{n_M}$$
$$= P_1(n_1) P_2(n_2) P_3(n_3) \cdots P_M(n_m)$$

In general:

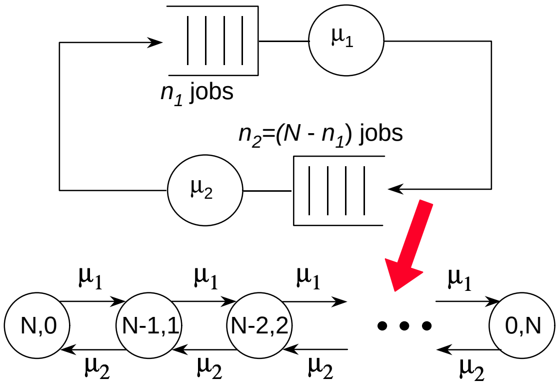
$$P(n_1, n_2, n_3, \cdots n_M) = \frac{1}{G(N)} \prod_{i=1}^{M} f_i(n_i)$$

where ***G(N)*** is a normalizing constant which is a function of the number of jobs in the system

and $\boldsymbol{f_i(n_i)}$ is a function of the jobs at the ***i***th server

This is a brief presentation of the analysis of a chain of M/M/1 queues. Note the form that the solution takes is the general form of the solution of a closed network of M/M/1 queues. Queuing networks whose solution takes this form are called “product form networks.”

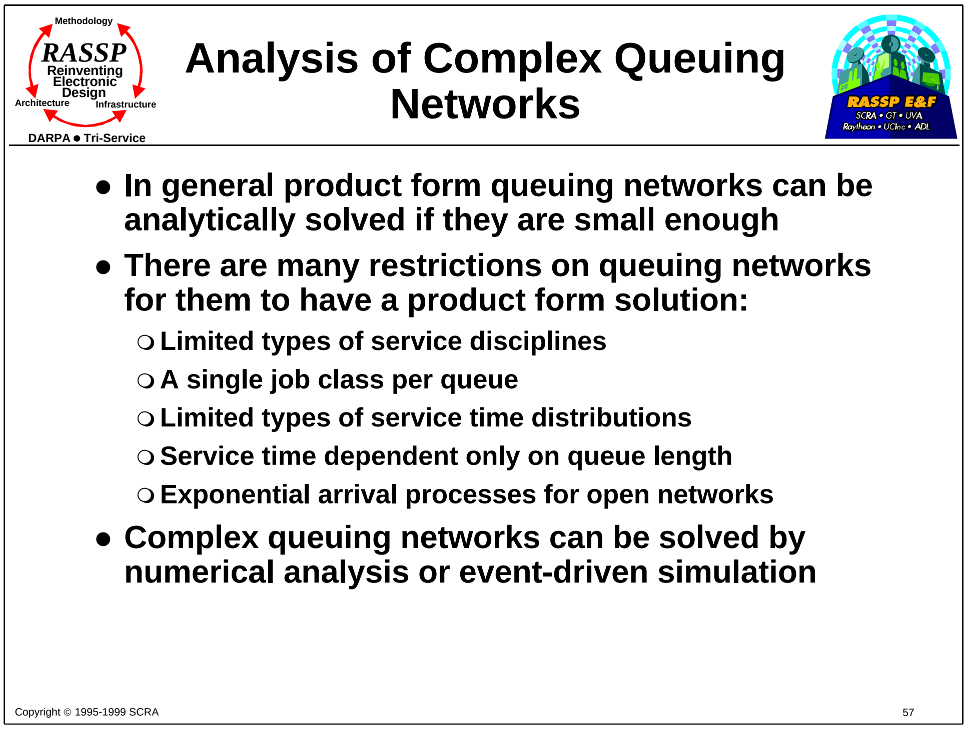
Product Form Queuing Network Example



$$P(n_1,n_2) = \frac{1}{\mu_2^{N+1} - \mu_1^{N+1}}(\mu_1^{n_1} \times \mu_2^{n_2})$$

where $G(N) = \mu_2^{N+1} - \mu_1^{N+1}$

This is an example of the solution of a closed network of M/M/1 queues. Note how the solution takes the general product form.

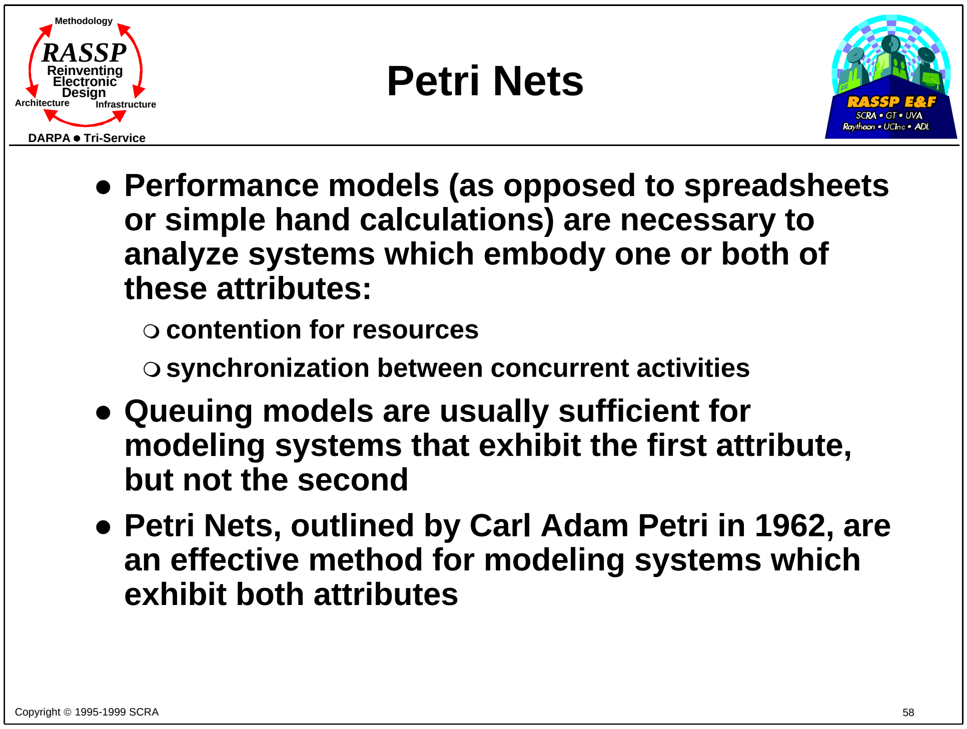


Analysis of Complex Queuing Networks

- In general product form queuing networks can be analytically solved if they are small enough
- There are many restrictions on queuing networks for them to have a product form solution:
  - ❍ Limited types of service disciplines
  - ❍ A single job class per queue
  - ❍ Limited types of service time distributions
  - ❍ Service time dependent only on queue length
  - ❍ Exponential arrival processes for open networks
- Complex queuing networks can be solved by numerical analysis or event-driven simulation

Product form queuing networks have a very mathematically “clean” solution, but there are many restrictions on the queuing networks such that they are “product form networks.”

Note that complex queuing networks can be solved numerically or by event driven simulation. This is the basis of many performance tools like SES Workbench, Extend, Foresight, etc.


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

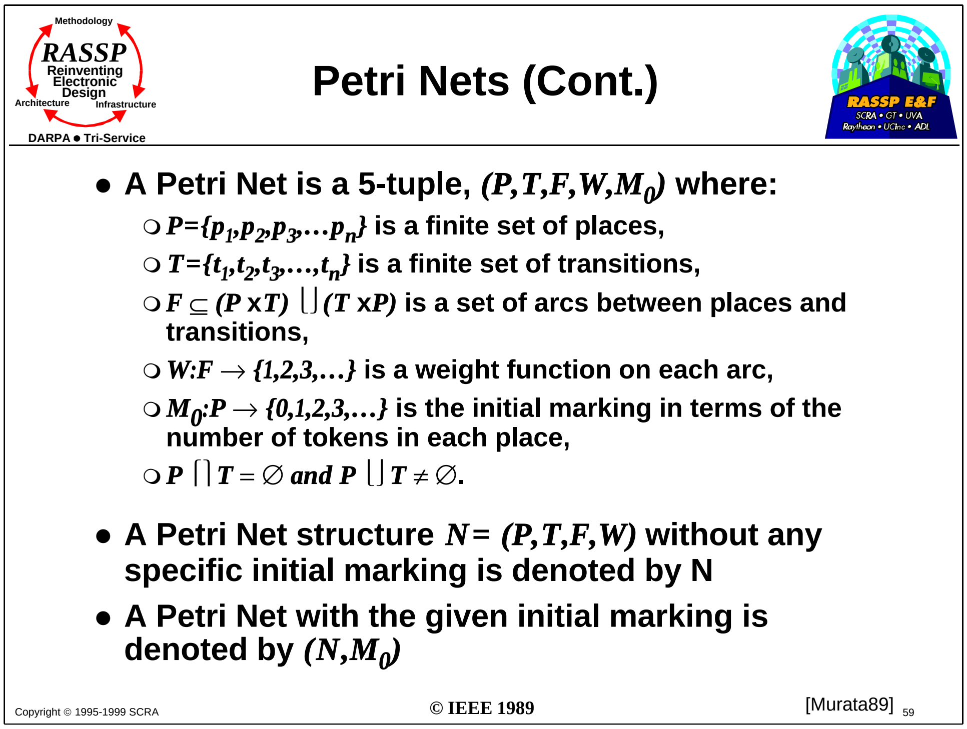
Petri Nets

- Performance models (as opposed to spreadsheets or simple hand calculations) are necessary to analyze systems which embody one or both of these attributes:
  - ❍ contention for resources
  - ❍ synchronization between concurrent activities
- Queuing models are usually sufficient for modeling systems that exhibit the first attribute, but not the second
- Petri Nets, outlined by Carl Adam Petri in 1962, are an effective method for modeling systems which exhibit both attributes

For simple systems that do not exhibit concurrency and contention, detailed performance modeling may not be necessary, a simple “spread sheet” approach might suffice. For systems that exhibit concurrency, and contention (like the transaction system example), queuing models are applicable. However, for systems that exhibit synchronization between concurrent activities, queuing models are not adequate.

Petri Nets, developed in 1962, are suited to modeling systems that have concurrency, contention, and synchronization.

The major reference for Petri Nets is the paper by Murata [Murata89], but [Cassandras93] is a good text reference.



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Petri Nets (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

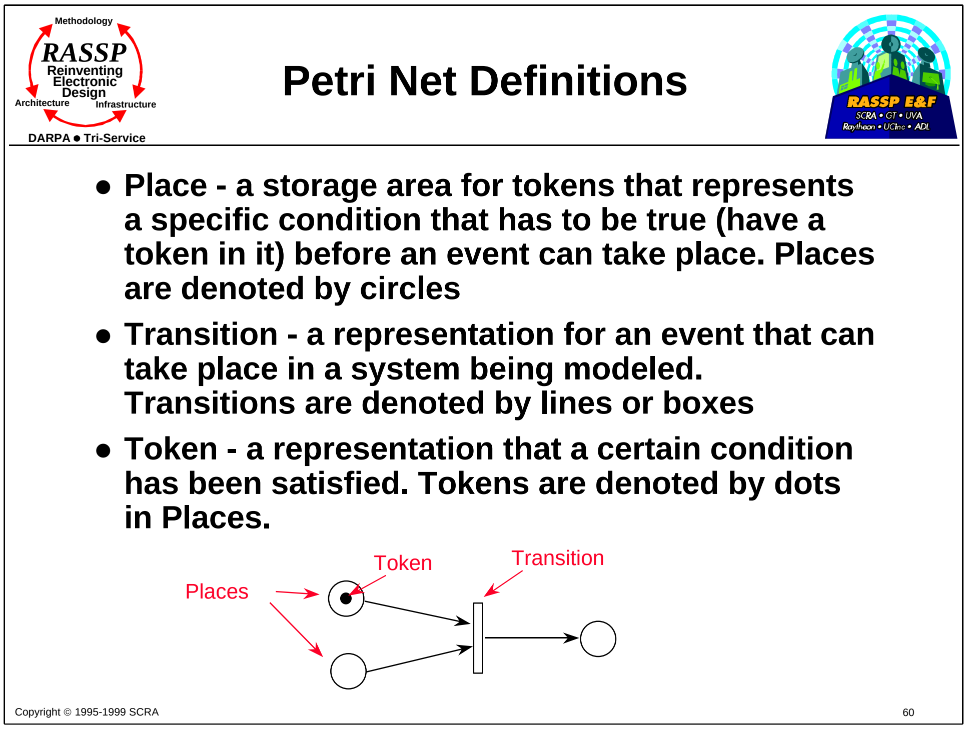
- A Petri Net is a 5-tuple, (P,T,F,W,M_0) where:
  - ❍ $P=\{p_1,p_2,p_3,\ldots p_n\}$ is a finite set of places,
  - ❍ $T=\{t_1,t_2,t_3,\ldots,t_n\}$ is a finite set of transitions,
  - ❍ $F \subseteq (P \times T) \bigcup (T \times P)$ is a set of arcs between places and transitions,
  - ❍ $W{:}F \rightarrow \{1,2,3,\ldots\}$ is a weight function on each arc,
  - ❍ $M_0{:}P \rightarrow \{0,1,2,3,\ldots\}$ is the initial marking in terms of the number of tokens in each place,
  - ❍ $P \bigcap T = \varnothing$ *and* $P \bigcup T \neq \varnothing$.
- A Petri Net structure $N= (P,T,F,W)$ without any specific initial marking is denoted by N
- A Petri Net with the given initial marking is denoted by (N,M_0)

Copyright © 1995-1999 SCRA

© IEEE 1989

[Murata89] 59

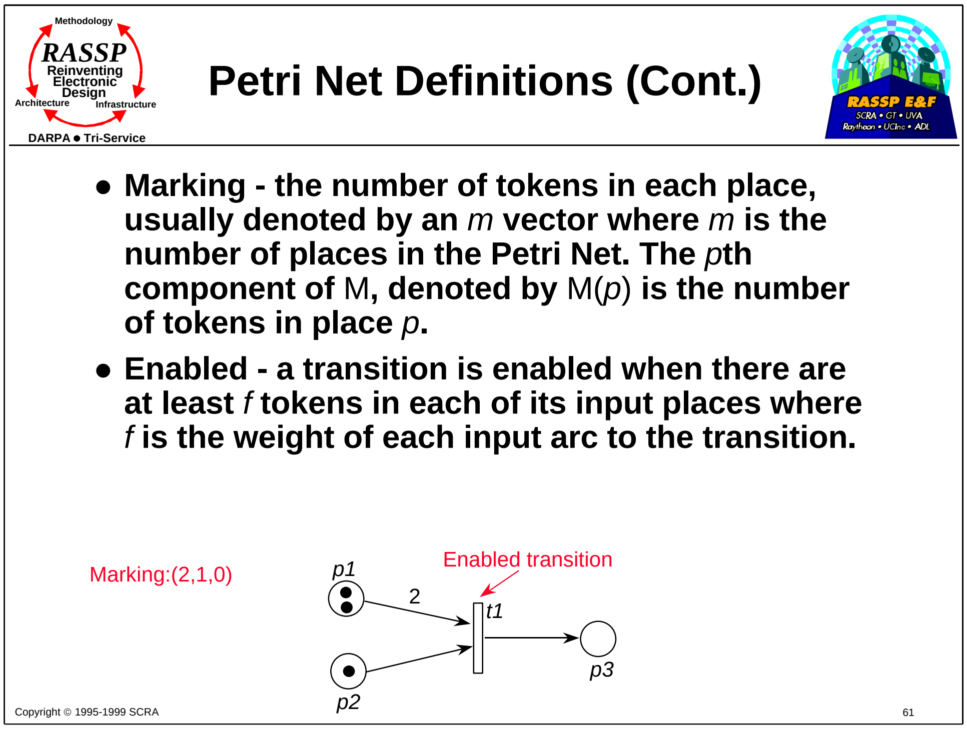
This is the basic definition of a Petri Net. Note that the basic Petri Net contains no notion of time or values on the data modeled in the system.



Petri Net Definitions

- Place - a storage area for tokens that represents a specific condition that has to be true (have a token in it) before an event can take place. Places are denoted by circles
- Transition - a representation for an event that can take place in a system being modeled. Transitions are denoted by lines or boxes
- Token - a representation that a certain condition has been satisfied. Tokens are denoted by dots in Places.

The basic definitions of the things that make up a Petri Net. Note that the Petri Net definition of a token is slightly different than the definition that will be used in the uninterpreted modeling section. In a Petri Net, a token is a representation that a certain condition, that will cause a transition to fire, has been satisfied. It does not necessarily denote actual data that is moving in the system, as is the case with most (but not all) uninterpreted modeling systems.



Petri Net Definitions (Cont.)

- Marking - the number of tokens in each place, usually denoted by an *m* vector where *m* is the number of places in the Petri Net. The *p*th component of M, denoted by M(*p*) is the number of tokens in place *p*.
- Enabled - a transition is enabled when there are at least *f* tokens in each of its input places where *f* is the weight of each input arc to the transition.

Marking:(2,1,0)
Enabled transition
p1
2
t1
p2
p3

No additional notes necessary.

RASSP Reinventing Electronic Design
Methodology Architecture Infrastructure
DARPA ● Tri-Service

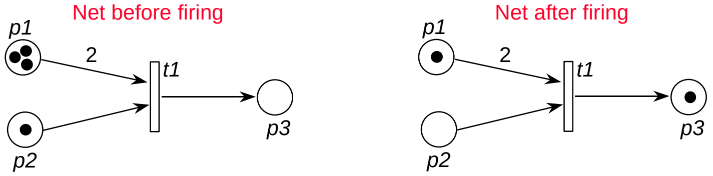
Petri Net Definitions (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

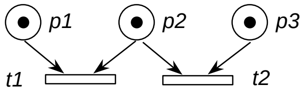
- Firing - the activation of an enable transition
  - ❍ it consumes the required amount of tokens at its input(s) and produces the required amount of tokens at it output(s)

Net before firing

Net after firing

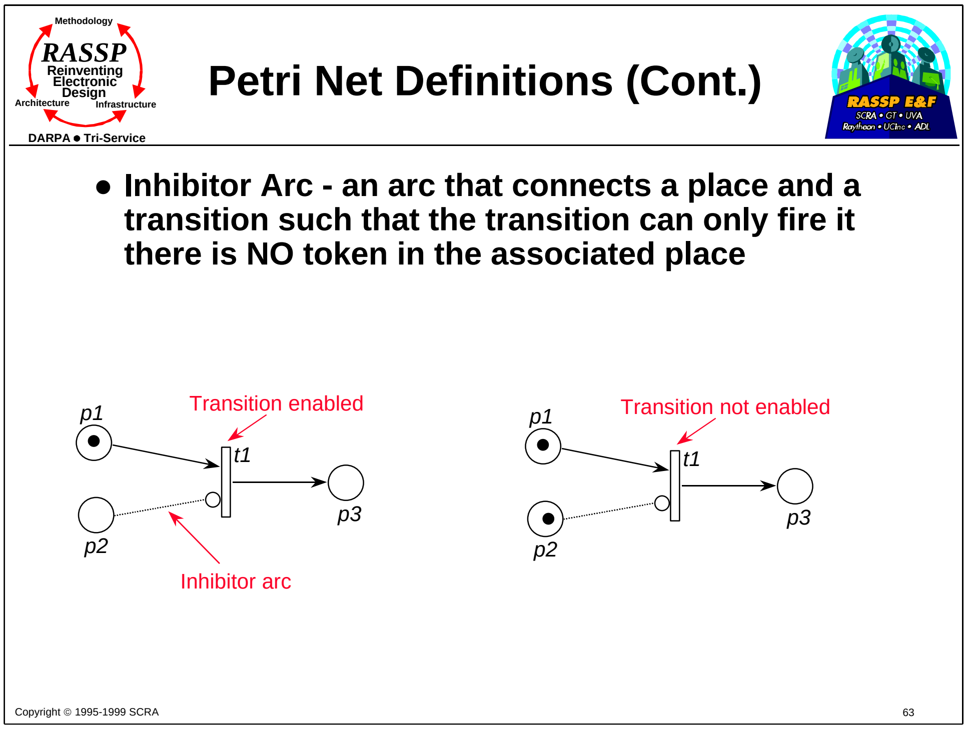


- Nondeterminism - when several transitions are simultaneously enabled, any one may fire first
- Conflict - when the firing of one enabled transition would disable another enabled transition



Transitions t1 and t2 conflict

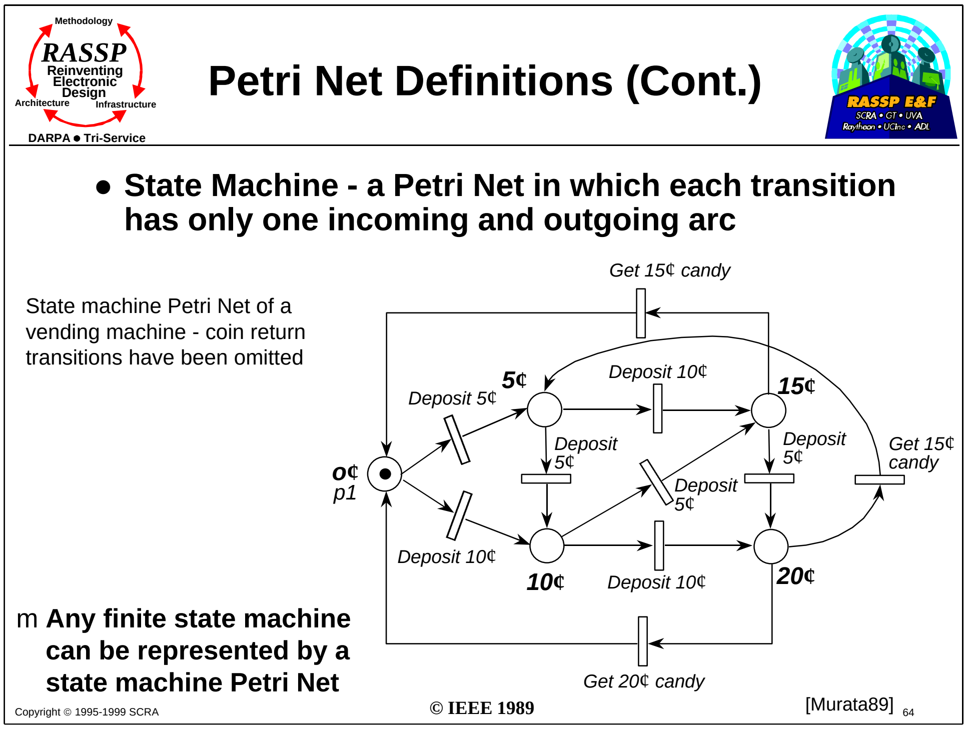
The nondeterminism of Petri Nets is a significant difference between them and other uninterpreted modeling techniques. Where two conflicting transitions are enabled, which one fires first can make a significant difference in how the model behaves.



Petri Net Definitions (Cont.)

- Inhibitor Arc - an arc that connects a place and a transition such that the transition can only fire it there is NO token in the associated place

No additional notes necessary.



Petri Net Definitions (Cont.)

- **State Machine - a Petri Net in which each transition has only one incoming and outgoing arc**

State machine Petri Net of a vending machine - coin return transitions have been omitted

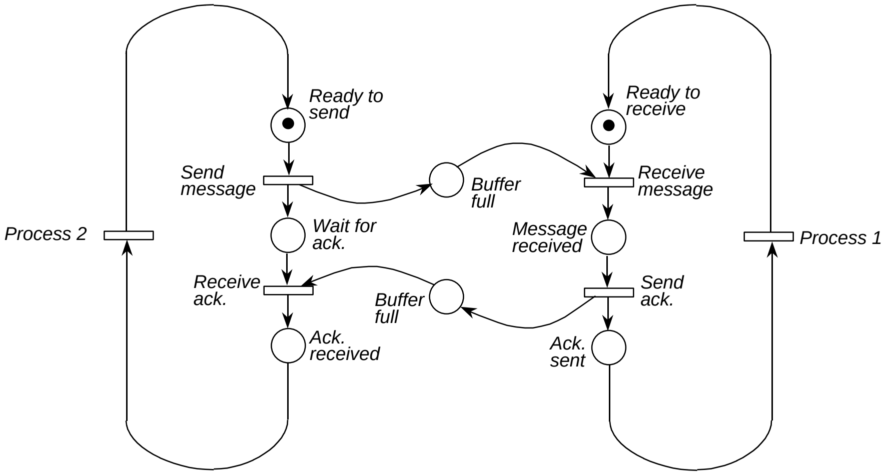
m **Any finite state machine can be represented by a state machine Petri Net**

© IEEE 1989

[Murata89] 64

This is a Petri Net model of a finite state machine (FSM). By definition, any FSM can be modeled with a Petri Net. One thing to note here is that in the real state machine, the firing of each transition is triggered by an external event, either the insertion of a coin or the pressing of a “get candy” button. However, in the true Petri Net model, which transition would fire, in the case where two or more are enabled (0¢, 15¢, 20¢ state), is non-deterministic.

Petri Net Examples

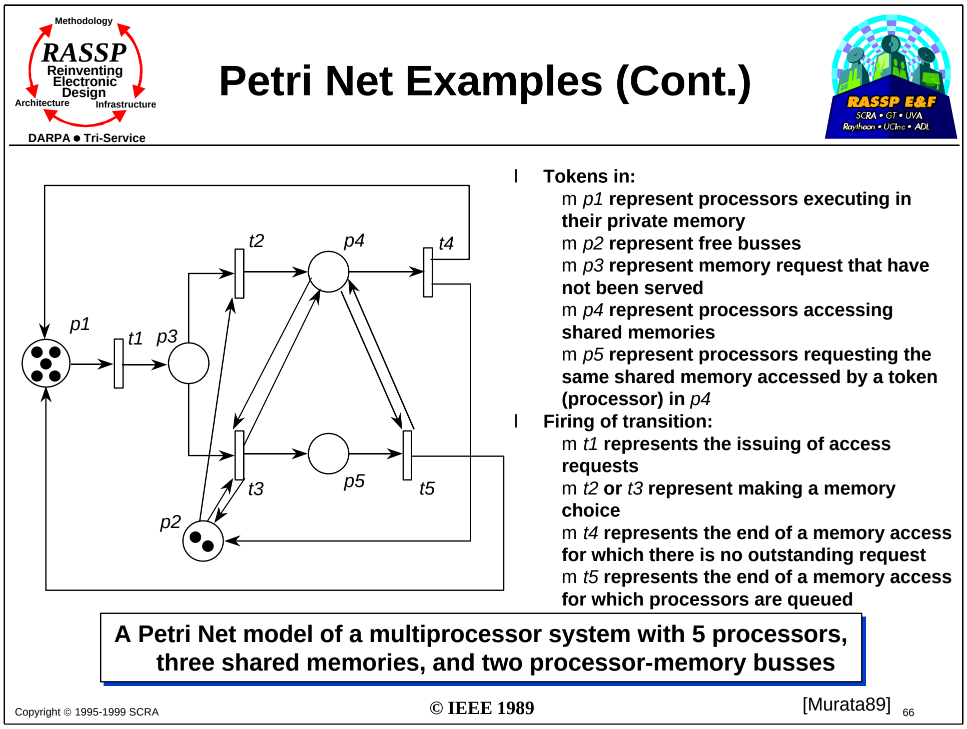


A Petri Net model of a simple communications protocol

© IEEE 1989

[Murata89]

This is a Petri Net model of a simple interlocking communications protocol. In fact, both hardware and software systems can be modeled with Petri Nets - a powerful feature.



Petri Net Examples (Cont.)

- Tokens in:
  - *p1* represent processors executing in their private memory
  - *p2* represent free busses
  - *p3* represent memory request that have not been served
  - *p4* represent processors accessing shared memories
  - *p5* represent processors requesting the same shared memory accessed by a token (processor) in *p4*
- Firing of transition:
  - *t1* represents the issuing of access requests
  - *t2* or *t3* represent making a memory choice
  - *t4* represents the end of a memory access for which there is no outstanding request
  - *t5* represents the end of a memory access for which processors are queued

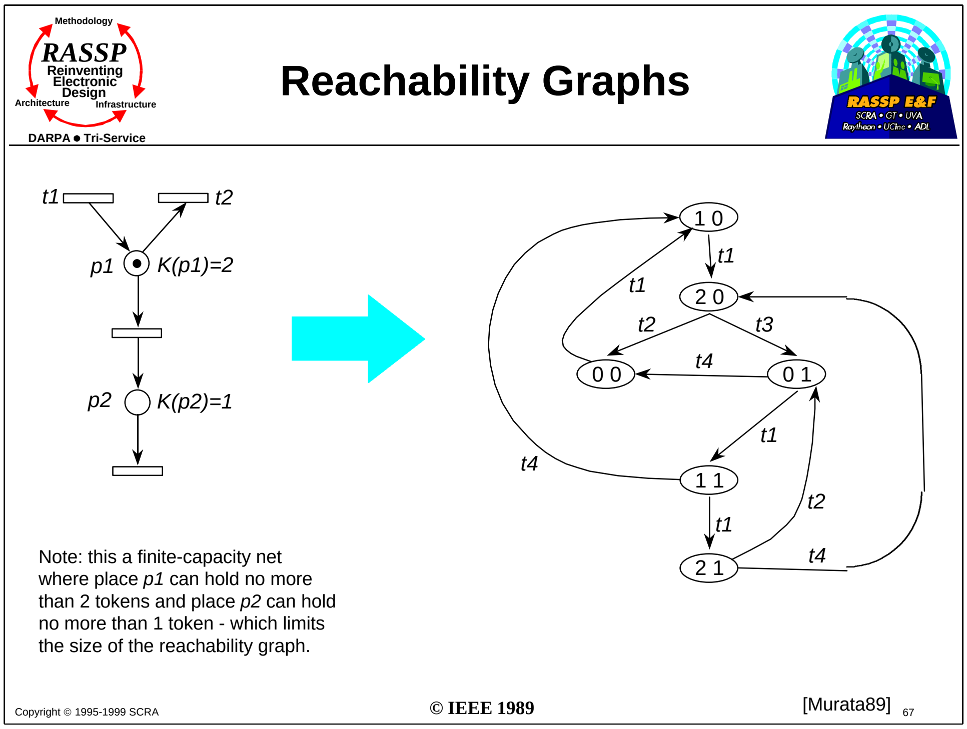
A Petri Net model of a multiprocessor system with 5 processors, three shared memories, and two processor-memory busses

© IEEE 1989

[Murata89] 66

This is a more complex Petri Net model of a multiprocessor system with 5 processors, three shared memories, and two processor-memory busses. It is intended to show how systems of this type can be modeled with Petri Nets and that there is not a one-to-one correspondence between tokens, place, and transitions and hardware components or data packets in a real system - which sometimes makes them difficult to conceive.

See [Murata89] for more details on this example.



Reachability Graphs

Note: this a finite-capacity net where place *p1* can hold no more than 2 tokens and place *p2* can hold no more than 1 token - which limits the size of the reachability graph.

[Murata89]

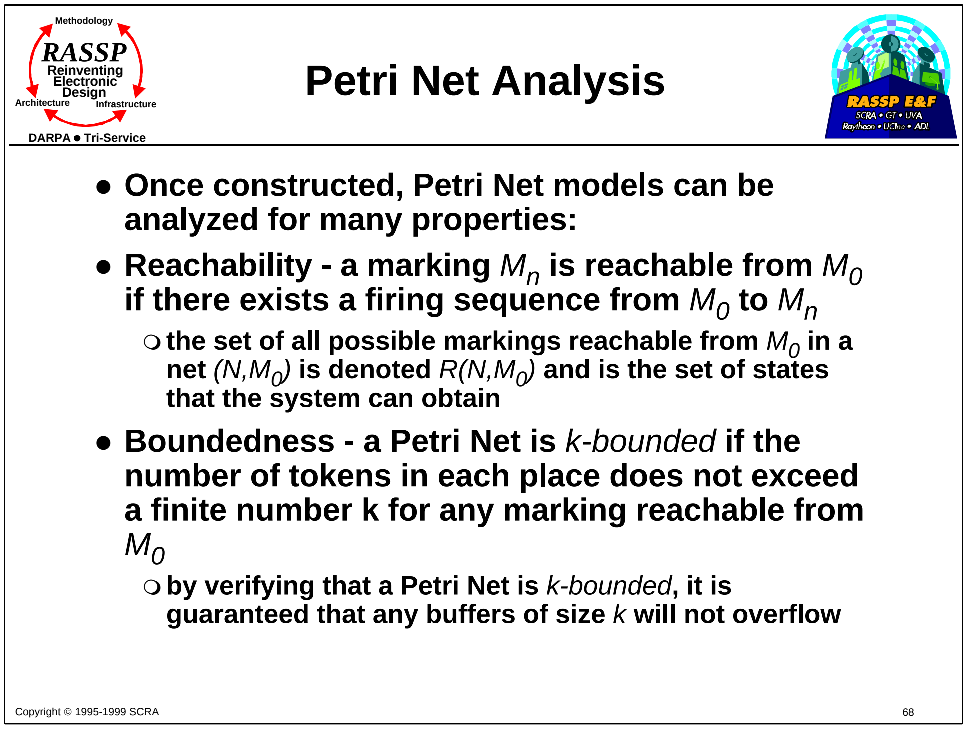
This slide introduces reachability graphs which are representations of the “states” or markings of a Petri Net and how they are reached by various transition firings.

The nodes in the reachability graph are markings (e.g., 1 0 is the marking where there is one token in p_1 and 0 tokens in p_2.

The arcs in the reachability graph are the transitions that move the Petri Net from one marking to another.

Note that in order to make the reachability graph for this example tractable (as far as drawing it), the example is a finite capacity net in that p_1 can hold no more than 2 tokens and p_2 can hold no more than 1 token.

Once the reachability graph is constructed, it can be analyzed using various graph algorithms.

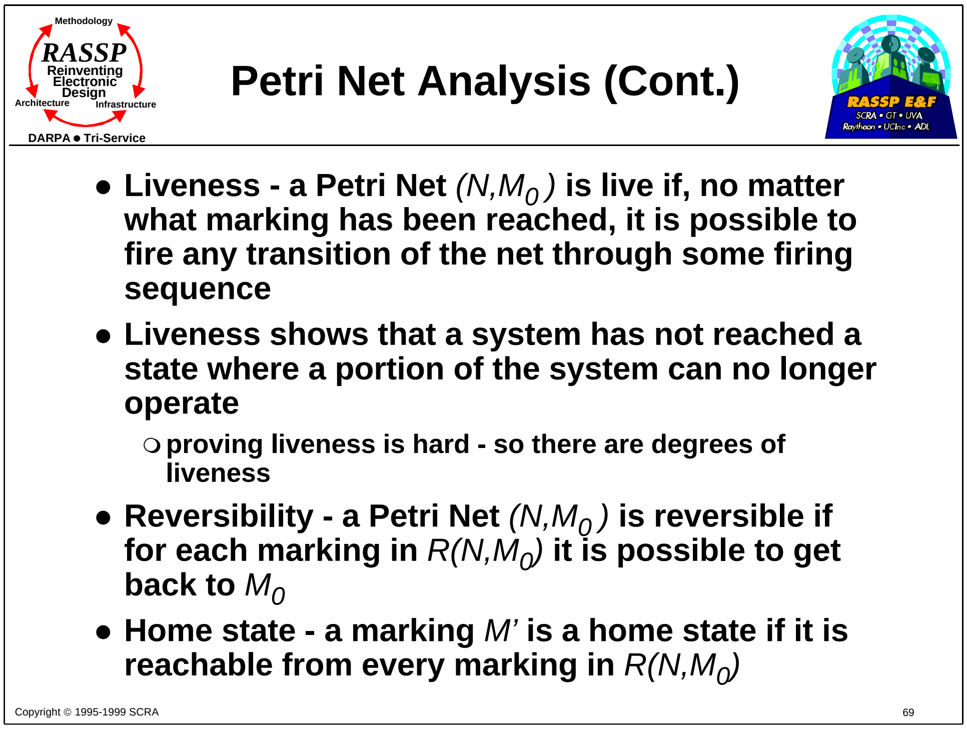

Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Petri Net Analysis

- Once constructed, Petri Net models can be analyzed for many properties:
- Reachability - a marking M_n is reachable from M_0 if there exists a firing sequence from M_0 to M_n
  - ❍ the set of all possible markings reachable from M_0 in a net (N,M_0) is denoted $R(N,M_0)$ and is the set of states that the system can obtain
- Boundedness - a Petri Net is *k-bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0
  - ❍ by verifying that a Petri Net is *k-bounded*, it is guaranteed that any buffers of size *k* will not overflow

Here are some of the attributes that the Petri Net can be analyzed for. All of these attributes can be examined analytically using the reachability graph and do not require simulating or “animating” the Petri Net.

Reachability analysis can be used to see if the Petri Net can attain any “undesirable” state. Boundedness can be used to determine if the “capacity of any state (e.g. buffer size) can be overflowed.

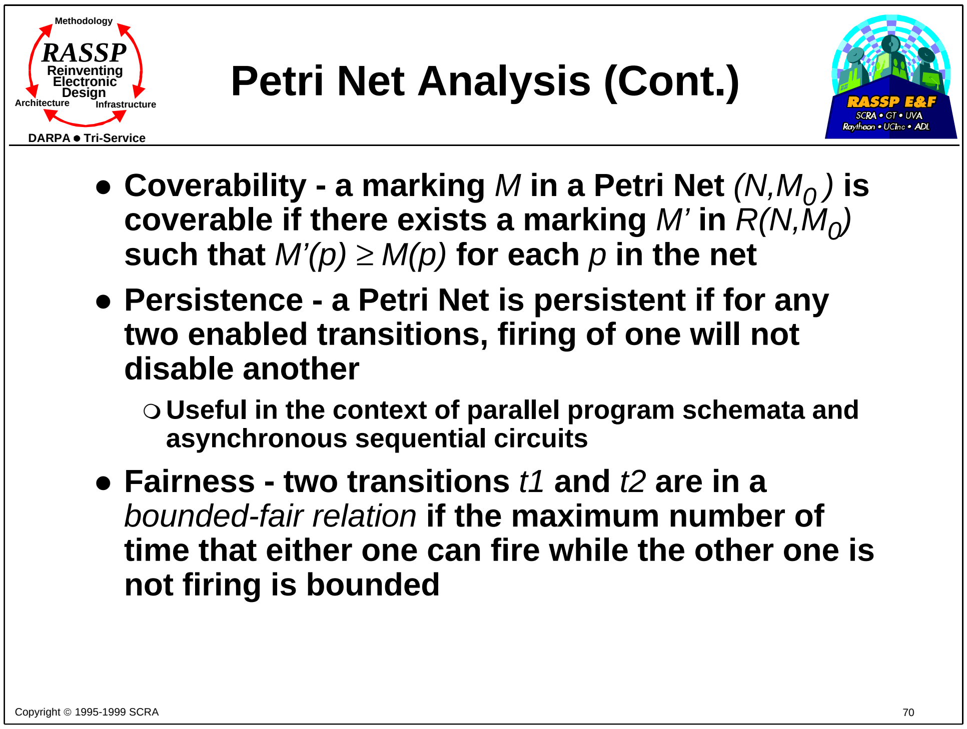


Petri Net Analysis (Cont.)

- Liveness - a Petri Net *(N,M_0)* is live if, no matter what marking has been reached, it is possible to fire any transition of the net through some firing sequence
- Liveness shows that a system has not reached a state where a portion of the system can no longer operate
  - ❍ proving liveness is hard - so there are degrees of liveness
- Reversibility - a Petri Net *(N,M_0)* is reversible if for each marking in *R(N,M_0)* it is possible to get back to *M_0*
- Home state - a marking *M'* is a home state if it is reachable from every marking in *R(N,M_0)*

Liveness can again show that the Petri Net does not attain an “undesirable” state in which its not exactly deadlocked, but some transitions can no longer be fired.

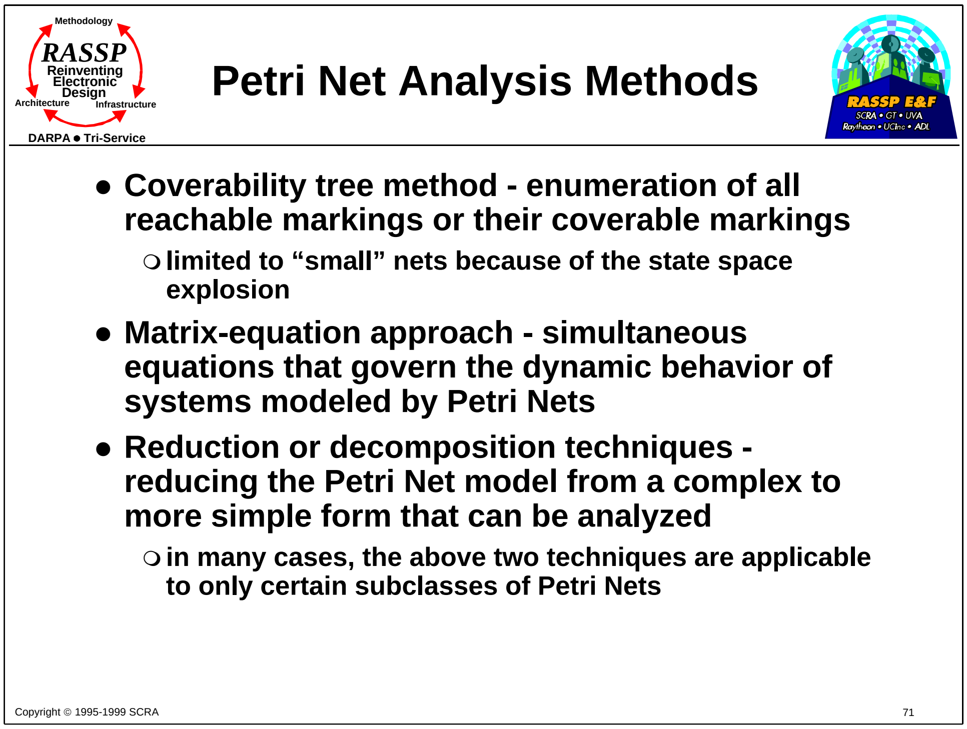
Reversibility shows that a Petri Net can regain its “home state” from any state it can attain.



Petri Net Analysis (Cont.)

- **Coverability - a marking** *M* **in a Petri Net** *(N,M_0)* **is coverable if there exists a marking** *M'* **in** *$R(N,M_0)$* **such that** *$M'(p) \geq M(p)$* **for each** *p* **in the net**
- **Persistence - a Petri Net is persistent if for any two enabled transitions, firing of one will not disable another**
  - **Useful in the context of parallel program schemata and asynchronous sequential circuits**
- **Fairness - two transitions** *$t1$* **and** *$t2$* **are in a** *bounded-fair relation* **if the maximum number of time that either one can fire while the other one is not firing is bounded**

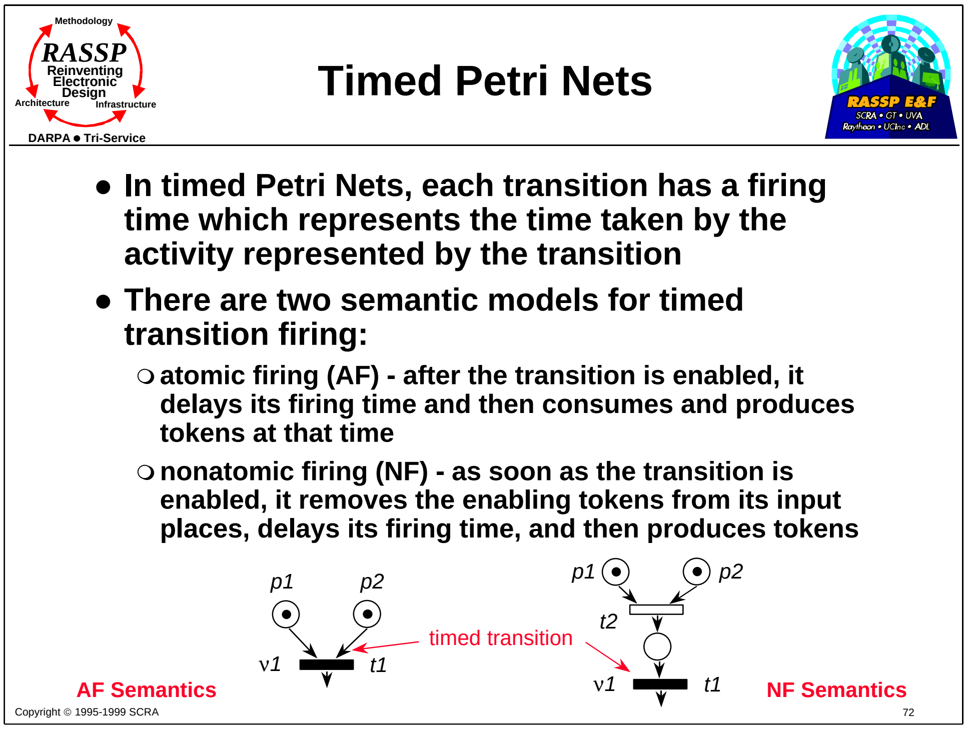
Here are more attributes that can be determined from the analysis of a Petri Net and its reachability graph.



Petri Net Analysis Methods

- Coverability tree method - enumeration of all reachable markings or their coverable markings
  - ❍ limited to “small” nets because of the state space explosion
- Matrix-equation approach - simultaneous equations that govern the dynamic behavior of systems modeled by Petri Nets
- Reduction or decomposition techniques - reducing the Petri Net model from a complex to more simple form that can be analyzed
  - ❍ in many cases, the above two techniques are applicable to only certain subclasses of Petri Nets

Various methods for analyzing Petri Nets for the metrics discussed.



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

Timed Petri Nets

- In timed Petri Nets, each transition has a firing time which represents the time taken by the activity represented by the transition
- There are two semantic models for timed transition firing:
  - ❍ atomic firing (AF) - after the transition is enabled, it delays its firing time and then consumes and produces tokens at that time
  - ❍ nonatomic firing (NF) - as soon as the transition is enabled, it removes the enabling tokens from its input places, delays its firing time, and then produces tokens

AF Semantics

NF Semantics

Timed Petri Nets are the more useful form for performance analysis. Both NF and AF semantics can be employed although AF is more general in that NF can be described in AF.

A potential problem with AF is that in conflicting transitions, an enabled transition may be disabled during its delay time by the firing of another transition.

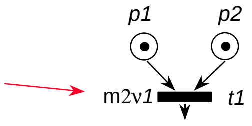
Methodology RASSP Reinventing Electronic Design Architecture Infrastructure DARPA ● Tri-Service

Petri Net Timing Functions

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

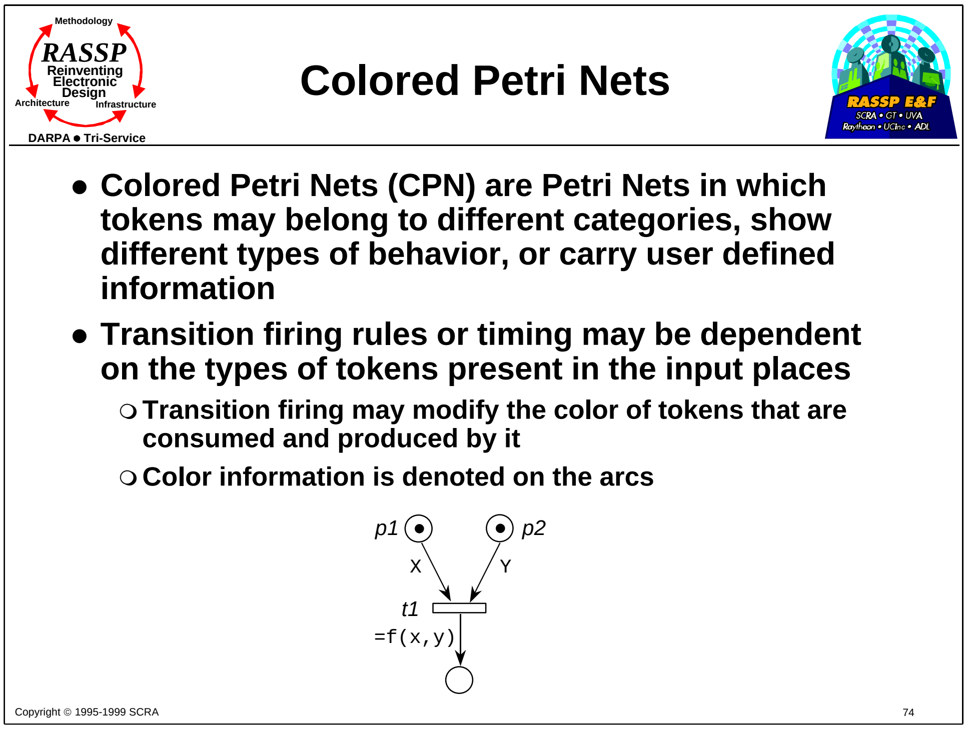
- Transition timing functions can depend on the number of tokens in a specific place in the Petri Net

transition timing is based on m2, the number of tokens in place *p2*



- Transition timing functions can be deterministic or stochastic
- Transition timing functions can be continuous time or discrete time

Timing functions for transitions can be a function of the number of tokens in a place. Also, timing functions can be deterministic of stochastic. General Stochastic Petri Nets can be analyzed as Markov Models (as will be shown).



Colored Petri Nets

- Colored Petri Nets (CPN) are Petri Nets in which tokens may belong to different categories, show different types of behavior, or carry user defined information
- Transition firing rules or timing may be dependent on the types of tokens present in the input places
  - ❍ Transition firing may modify the color of tokens that are consumed and produced by it
  - ❍ Color information is denoted on the arcs

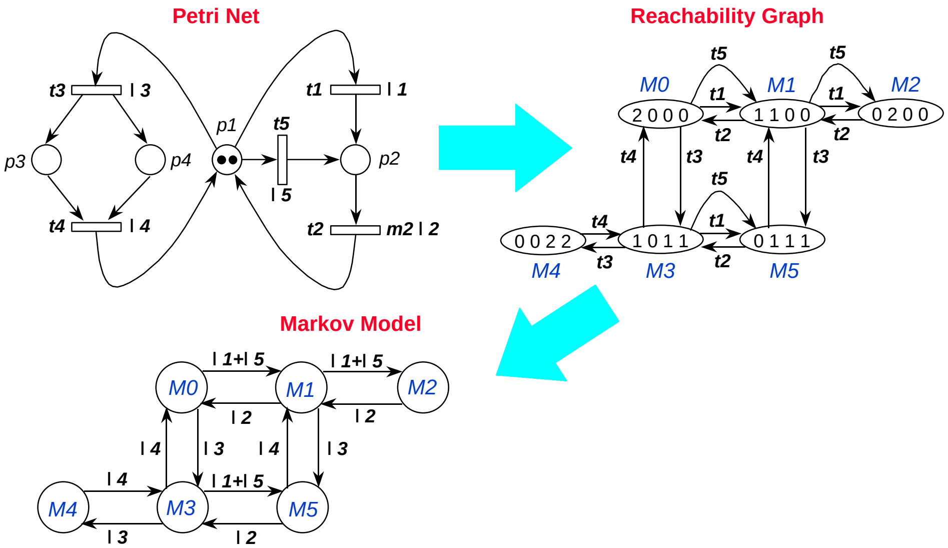
Copyright © 1995-1999 SCRA

74

Colored Petri Nets (CPN) include the notion of values (or classes) on the tokens. Note that CPNs are what is used as the mathematical foundation for UVa’s ADEPT tool.

In this example, the color of the token produced by the firing of transition t1 is a function [f(x,y)] of the color of the tokens in the p1 and p2 places.

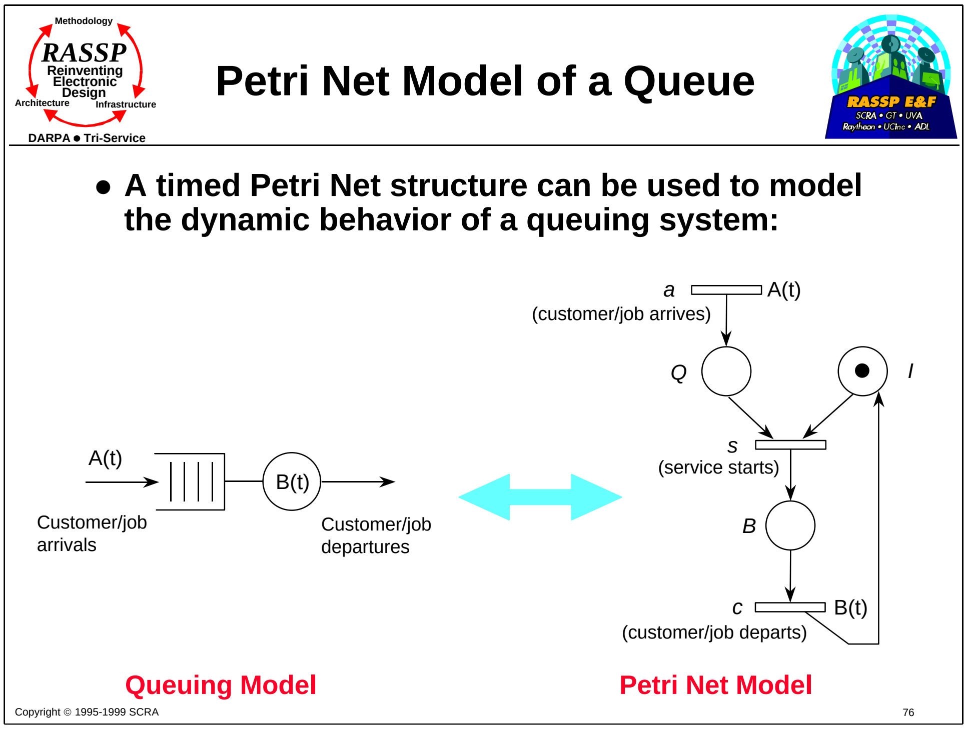
Stochastic Petri Net Analysis


Petri Net

Reachability Graph

Markov Model

As shown here, a Stochastic Petri Net can be translated into a Markov Model via its reachability graph.



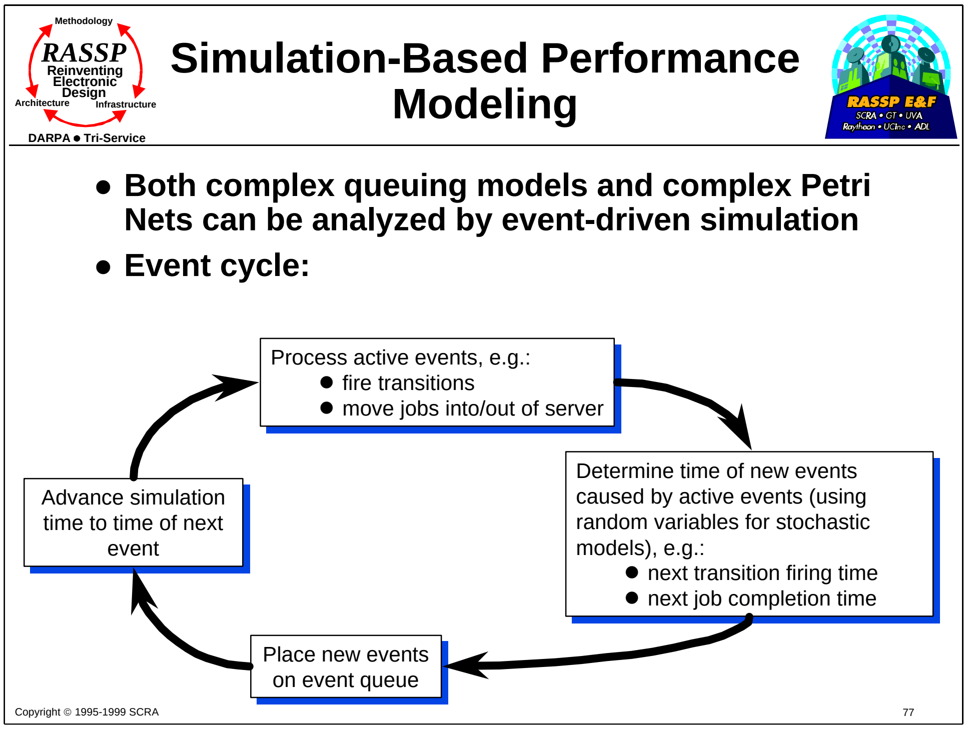
Petri Net Model of a Queue

- A timed Petri Net structure can be used to model the dynamic behavior of a queuing system:

Queuing Model

Petri Net Model

Here is an example of how a queuing model can be modeled using Petri Nets - a further demonstration of their modeling power.



Simulation-Based Performance Modeling

- Both complex queuing models and complex Petri Nets can be analyzed by event-driven simulation
- Event cycle:

Process active events, e.g.:
- fire transitions
- move jobs into/out of server

Determine time of new events caused by active events (using random variables for stochastic models), e.g.:
- next transition firing time
- next job completion time

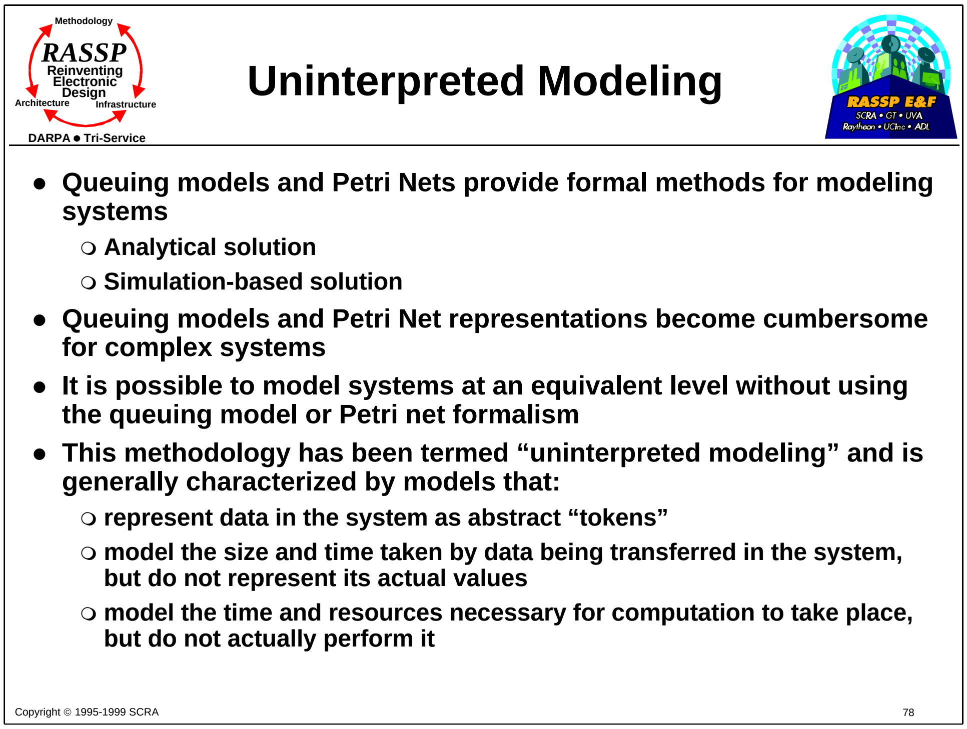
Place new events on event queue

Advance simulation time to time of next event

As mentioned before, complex queuing models and Petri Nets, although they may not be solvable via analytical techniques, can be solved by simulation. There are many commercial tools available that do this.

This is an illustration of the basic event driven simulation cycle. You simply process all events scheduled for a given time, and determine what new events are generated for what future times. These events are added to the “event queue” and time is advanced to the earliest future time in the event queue. All events at that time are then processed and the cycle begins again.

Alternatively to event-driven simulation, the simulation cycle can be done on a discrete time interval (e.g. 1 ns) and simulation time advances at regular intervals. All signals can be updated to new values (which may be the same as old ones) at each time interval. This eases the management of simulation time and the event queue.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure DARPA ● Tri-Service

Uninterpreted Modeling

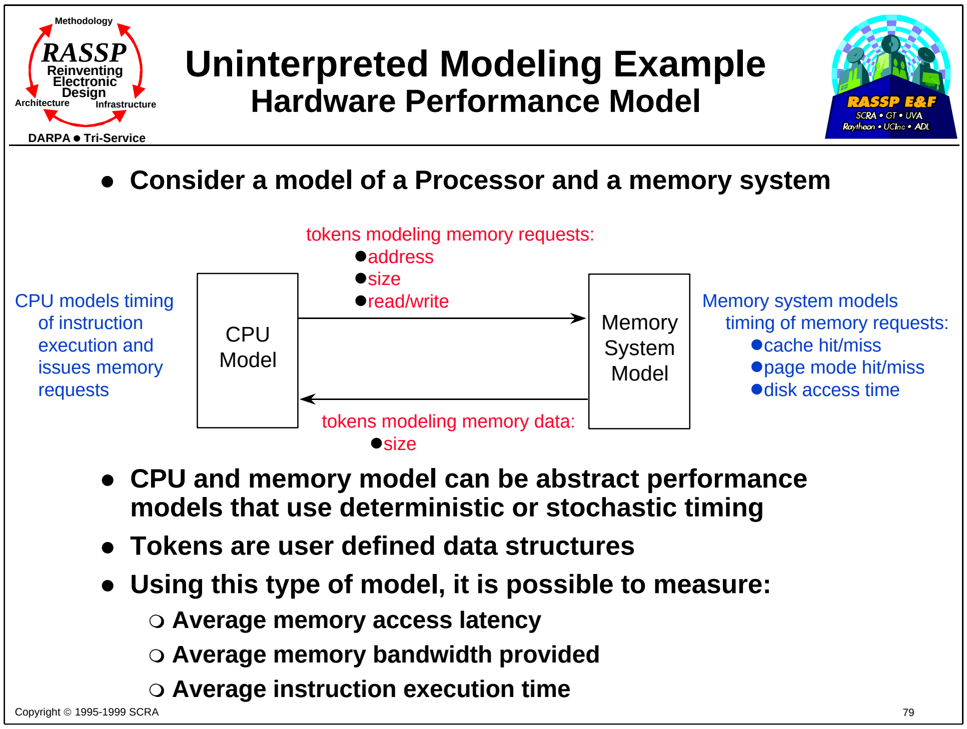
RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

- Queuing models and Petri Nets provide formal methods for modeling systems
  - ❍ Analytical solution
  - ❍ Simulation-based solution
- Queuing models and Petri Net representations become cumbersome for complex systems
- It is possible to model systems at an equivalent level without using the queuing model or Petri net formalism
- This methodology has been termed “uninterpreted modeling” and is generally characterized by models that:
  - ❍ represent data in the system as abstract “tokens”
  - ❍ model the size and time taken by data being transferred in the system, but do not represent its actual values
  - ❍ model the time and resources necessary for computation to take place, but do not actually perform it

78

It is possible to model systems at a high level without using either the queuing model or Petri Net formalism. This is a separate issue from the analytical vs. simulation-based solution issue, although models that do not have the queuing model or Petri Net formalism obviously have to use simulation-based solutions.

In general “uninterpreted modeling” the system is modeled at such a level as the data in the system that is moved from component to component is modeled, but its values and transformations performed on it are not. Timing is modeled, but usually at a high level. Recall that the taxonomy of performance models showed this level of abstraction. In general, all of the modeling environments discussed from her on out will be general “uninterpreted modeling” environments although some of them may include elements of queuing models (SES Workbench) and Petri Nets (ADEPT)



Uninterpreted Modeling Example
Hardware Performance Model

- Consider a model of a Processor and a memory system

tokens modeling memory requests:
- address
- size
- read/write

CPU models timing of instruction execution and issues memory requests

CPU Model

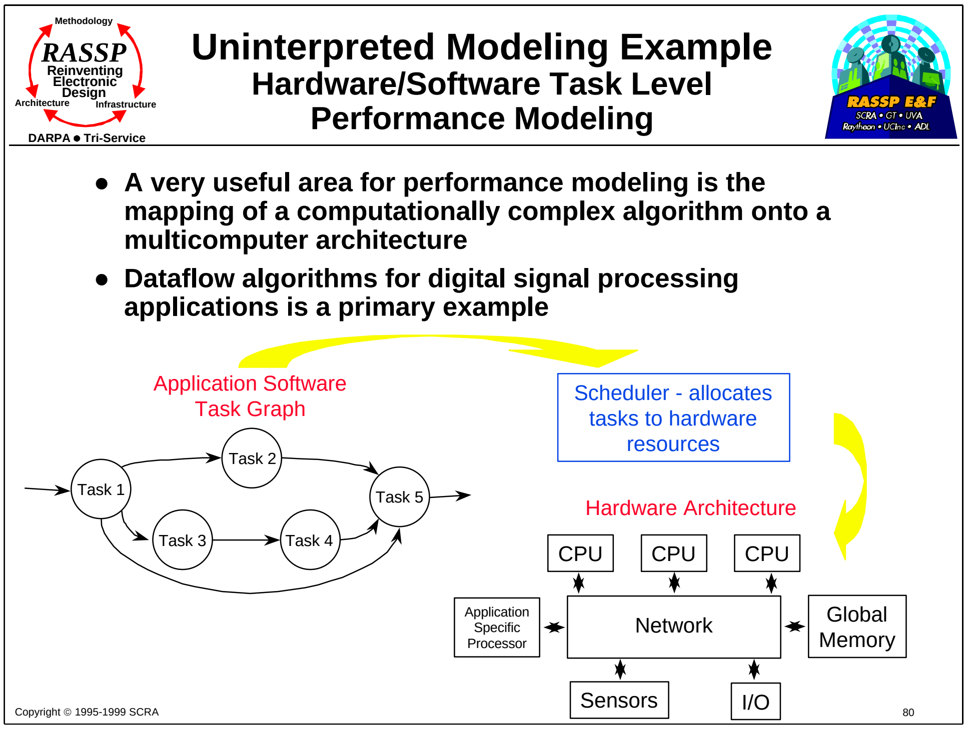
Memory System Model

Memory system models timing of memory requests:
- cache hit/miss
- page mode hit/miss
- disk access time

tokens modeling memory data:
- size

- CPU and memory model can be abstract performance models that use deterministic or stochastic timing
- Tokens are user defined data structures
- Using this type of model, it is possible to measure:
  - ❍ Average memory access latency
  - ❍ Average memory bandwidth provided
  - ❍ Average instruction execution time

Here is an example of an uninterpreted model of a CPU and memory system. This is an example that will be utilized in the section on VHDL performance modeling examples. Notice that the tokens in the model actually model the passing of data between the CPU and the memory and are fairly abstract in nature, as are the CPU and memory component models.



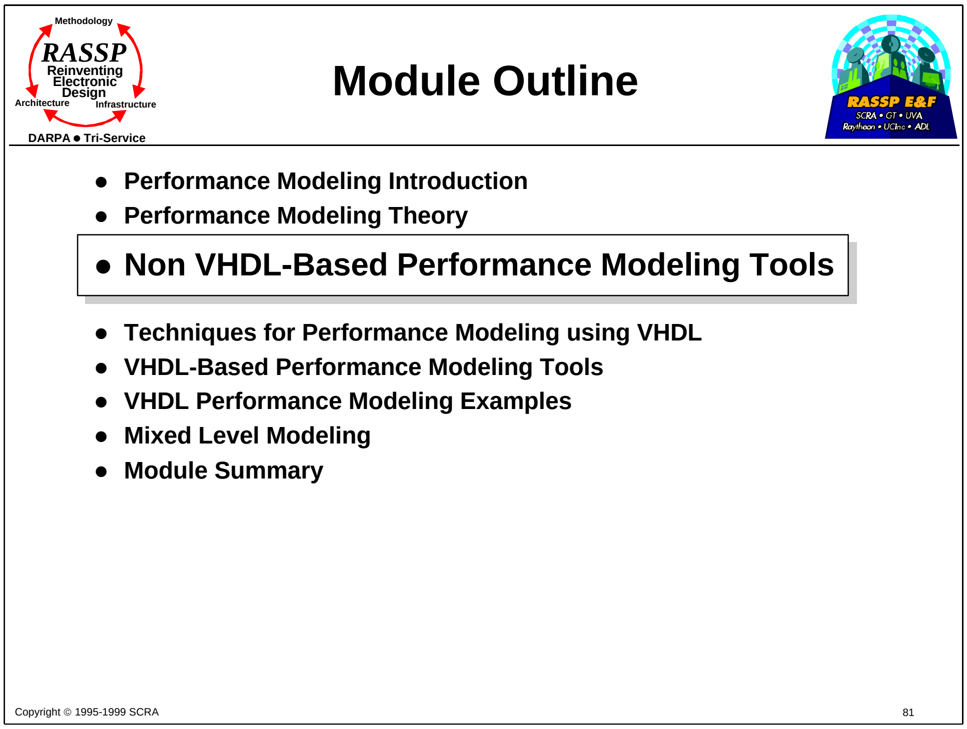
Uninterpreted Modeling Example

Hardware/Software Task Level Performance Modeling

- A very useful area for performance modeling is the mapping of a computationally complex algorithm onto a multicomputer architecture
- Dataflow algorithms for digital signal processing applications is a primary example

80

This is another type of uninterpreted model that will also be used in the example section, a hardware/software task level model. Here the software is a set of tasks, often modeled as a dataflow graph, that communicates with a “scheduler” to obtain hardware resources (processors, memories, switches) on which to execute. Usually, the software tasks provide information on how much hardware resources they require (data size, number of floating point instructions, etc.) and the hardware model actually delays the required simulated time.

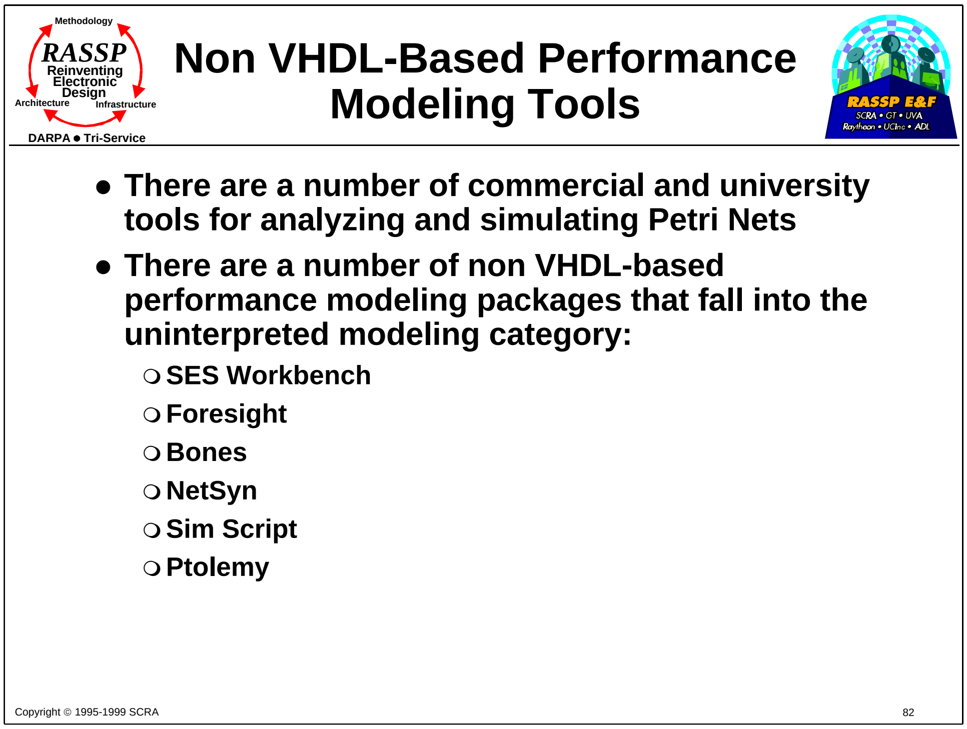


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Module Outline

- Performance Modeling Introduction
- Performance Modeling Theory
- **Non VHDL-Based Performance Modeling Tools**
- Techniques for Performance Modeling using VHDL
- VHDL-Based Performance Modeling Tools
- VHDL Performance Modeling Examples
- Mixed Level Modeling
- Module Summary

Module Outline



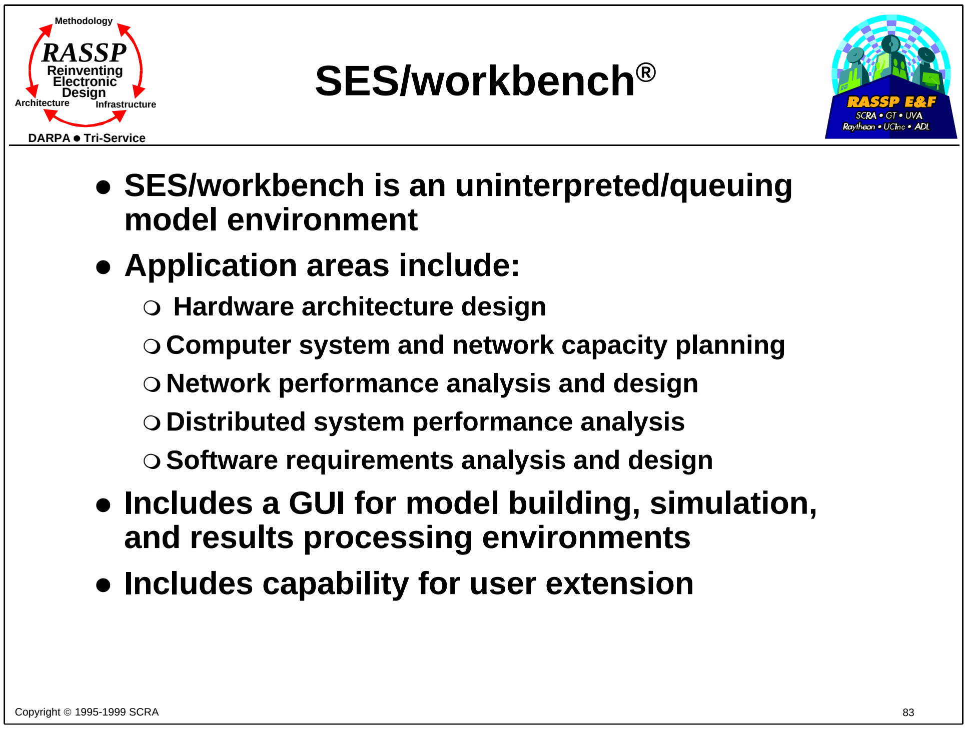
RASSP Reinventing Electronic Design
Methodology Architecture Infrastructure
DARPA ● Tri-Service

Non VHDL-Based Performance Modeling Tools

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- There are a number of commercial and university tools for analyzing and simulating Petri Nets
- There are a number of non VHDL-based performance modeling packages that fall into the uninterpreted modeling category:
  - ❍ SES Workbench
  - ❍ Foresight
  - ❍ Bones
  - ❍ NetSyn
  - ❍ Sim Script
  - ❍ Ptolemy

There are a number of commercial and educational packages available for Petri Net analysis and general “uninterpreted” performance modeling. Most of these are implemented in C or C++ and as such, are a bit divorced from the electronic system design process. However, because of their number and popularity, some discussion of them is warranted here.



SES/workbench®

- SES/workbench is an uninterpreted/queuing model environment
- Application areas include:
  - ❍ Hardware architecture design
  - ❍ Computer system and network capacity planning
  - ❍ Network performance analysis and design
  - ❍ Distributed system performance analysis
  - ❍ Software requirements analysis and design
- Includes a GUI for model building, simulation, and results processing environments
- Includes capability for user extension

As an example of the types of tools in the general uninterpreted performance modeling category that are available, SES workbench® will be presented in some detail. SES does have some basis in queuing network modeling, but performance models that do not include queues can be built with it, so it falls into the more general category.

This presentation was taken from the Scientific and Engineering Software, Inc. web page: http://www.ses.com

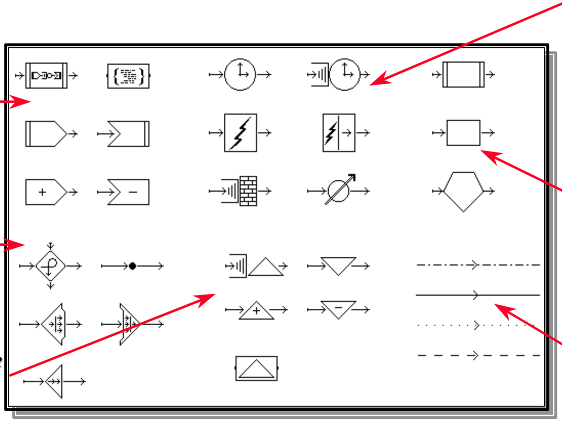
A through reading of the material on Workbench there will suffice as background to present these slides.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

SES/workbench Building Blocks

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

SES/*workbench* provides 25 primitive building blocks for creating models



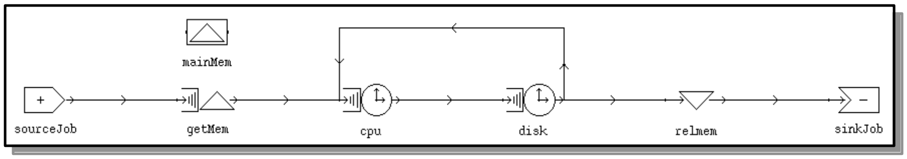
- *Submodel management nodes*
- *Flow Control nodes*
- *Passive Resource management nodes*
- *Active Resource management nodes*
- *User Extension/ Custom Function nodes*
- *Connection/ Statistical arcs*

Copyright 1999, SES, Inc. All Rights Reserved. [SES]

See http://www.ses.com

SES/workbench Model Development

- **SES workbench performance models are created using a GUI interface**
  - **placing and interconnecting building blocks to represent system function/structure**

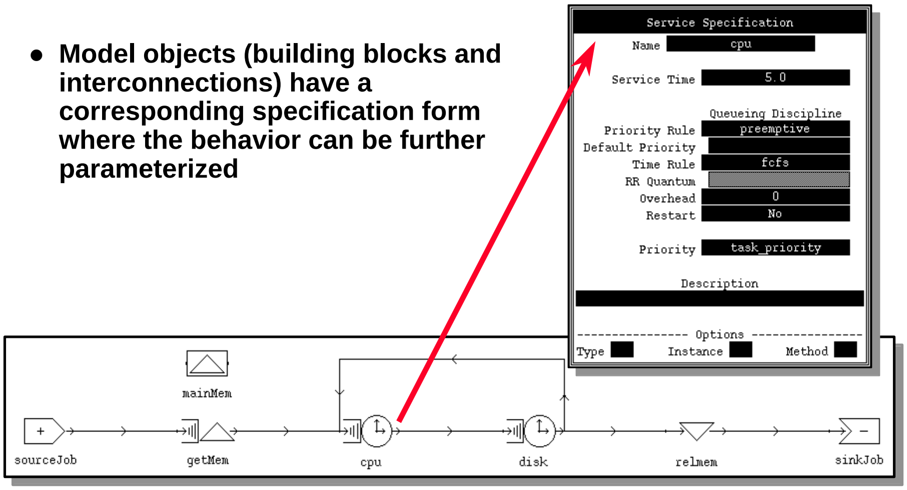


[SES]

See http://www.ses.com

SES/workbench Model Parameterization

- **Model objects (building blocks and interconnections) have a corresponding specification form where the behavior can be further parameterized**



Service Specification

Name cpu

Service Time 5.0

Queueing Discipline

Priority Rule preemptive

Default Priority

Time Rule fcfs

RR Quantum

Overhead 0

Restart No

Priority task_priority

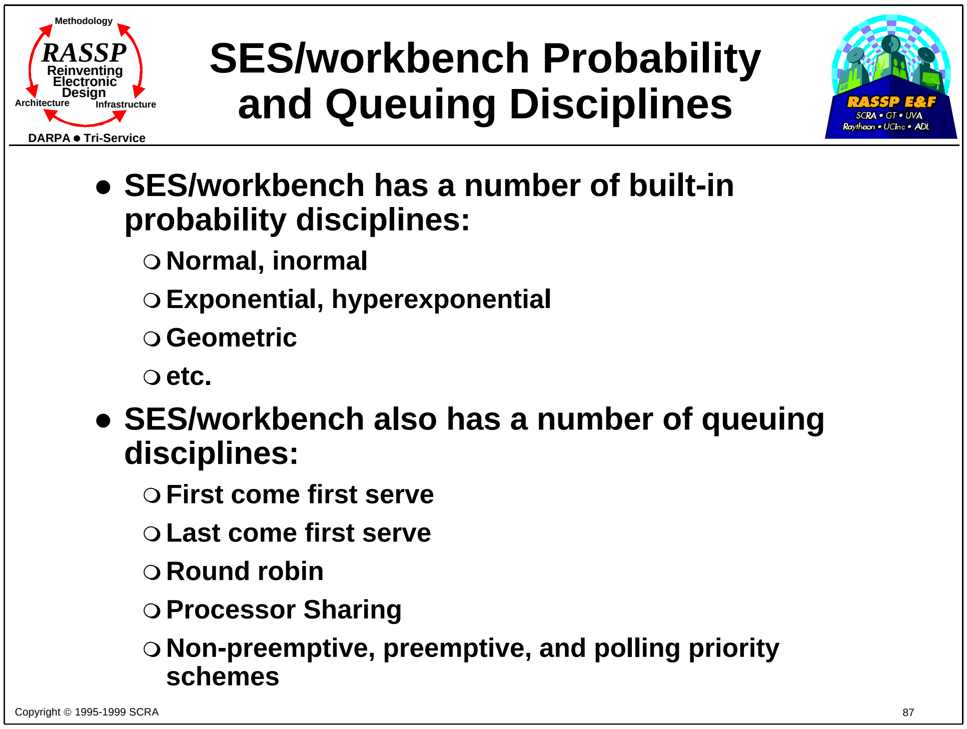
Description

---------------- Options ----------------

Type ■ Instance ■ Method ■

[SES]

See http://www.ses.com


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

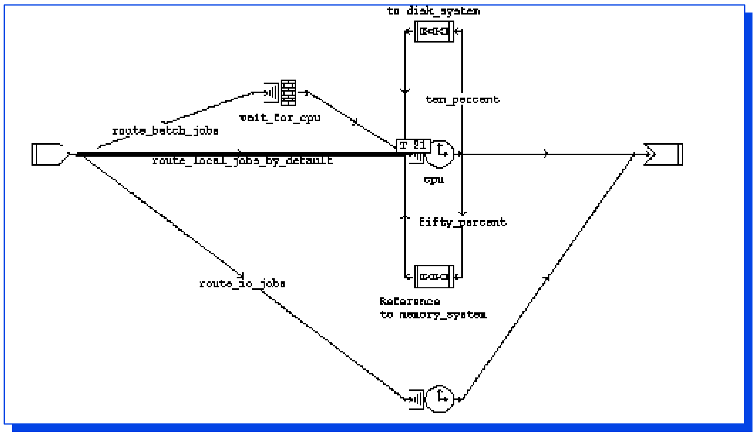
SES/workbench Probability and Queuing Disciplines

- SES/workbench has a number of built-in probability disciplines:
  - ❍ Normal, inormal
  - ❍ Exponential, hyperexponential
  - ❍ Geometric
  - ❍ etc.
- SES/workbench also has a number of queuing disciplines:
  - ❍ First come first serve
  - ❍ Last come first serve
  - ❍ Round robin
  - ❍ Processor Sharing
  - ❍ Non-preemptive, preemptive, and polling priority schemes

See http://www.ses.com

SES/workbench Model Simulation

- **SES/workbench models can be animated to show the flow of information**



Copyright 1999, SES, Inc. All Rights Reserved.

[SES]

See http://www.ses.com

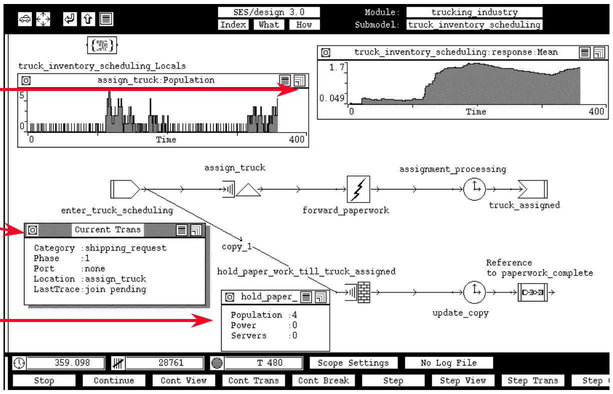
SES/workbench Model Simulation (Cont.)

- **SES/workbench includes the capability of viewing the model statistics as the model executes**

• Gather statistics on workload, environment and application performance

• Inspect the current work in your system

• Analyze the application load on the execution environment



Module: trucking_industry
Submodel: truck_inventory_scheduling

```
Category  :shipping_request
Phase     :1
Port      :none
Location  :assign_truck
LastTrace :join pending
```

```
Population :4
Power      :0
Servers    :0
```

```
359.098    28761    T 480
```

Copyright 1999, SES, Inc. All Rights Reserved.

[SES]

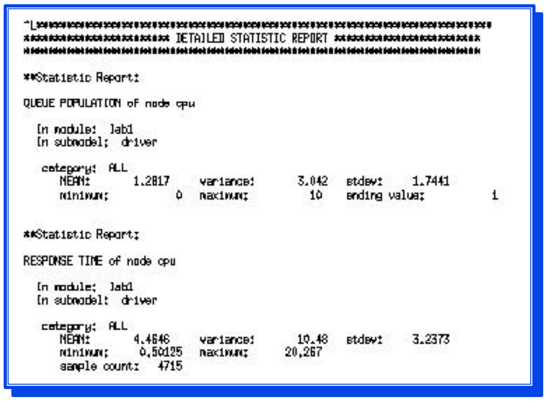
See http://www.ses.com

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

SES/workbench Model Simulation (Cont.)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- SES/workbench provides model statistics on system performance that permit verification, debugging, and optimization of system designs
- Statistics may be built-in or user-defined



```
^L*****************************************************************
********************** DETAILED STATISTIC REPORT ********************
*****************************************************************

**Statistic Report:

QUEUE POPULATION of node cpu

  In module:  lab1
  In submodel;  driver

   category:  ALL
      MEAN:       1.2817     variance:       3.042   stdev:     1.7441
      minimum;           0   maximum;           10   ending value;          1


**Statistic Report;

RESPONSE TIME of node cpu

  In module;  lab1
  In submodel:  driver

   category;  ALL
      MEAN:       4.4646     variance:       10.48   stdev:     3.2373
      minimum;     0.50125   maximum;      20.267
      sample count:   4715
```

[SES]

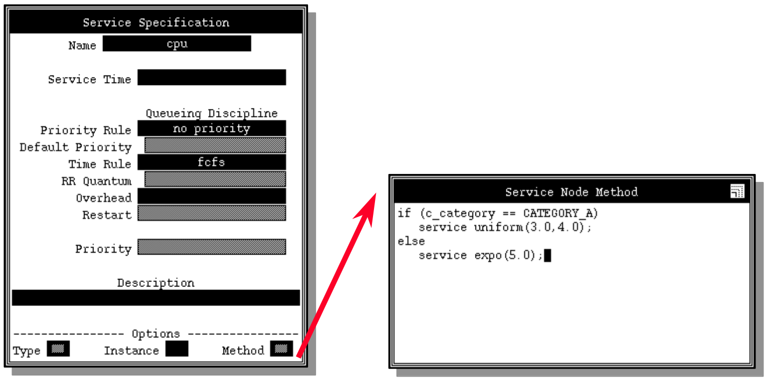
See http://www.ses.com

Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service

User Extensions to SES/workbench

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Users can extend the graphical modeling icons to represent unique system behaviors**



Service Specification

Name	cpu
Service Time	
	Queueing Discipline
Priority Rule	no priority
Default Priority	
Time Rule	fcfs
RR Quantum	
Overhead	
Restart	
Priority	
Description	

---------------- Options ----------------

Type ▨ Instance ■ Method ▨

Service Node Method

```
if (c_category == CATEGORY_A)
   service uniform(3.0,4.0);
else
   service expo(5.0);
```

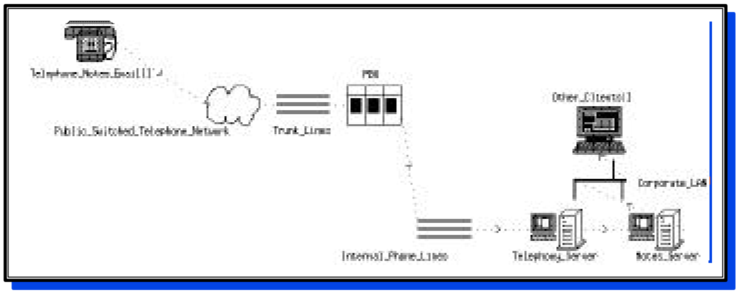
Copyright 1999, SES, Inc. All Rights Reserved.

[SES]

See http://www.ses.com

User Extensions to SES/workbench (Cont.)

- Users can add custom icons to the SES/workbench to represent portions of the modeled system in a more self-explanatory manner

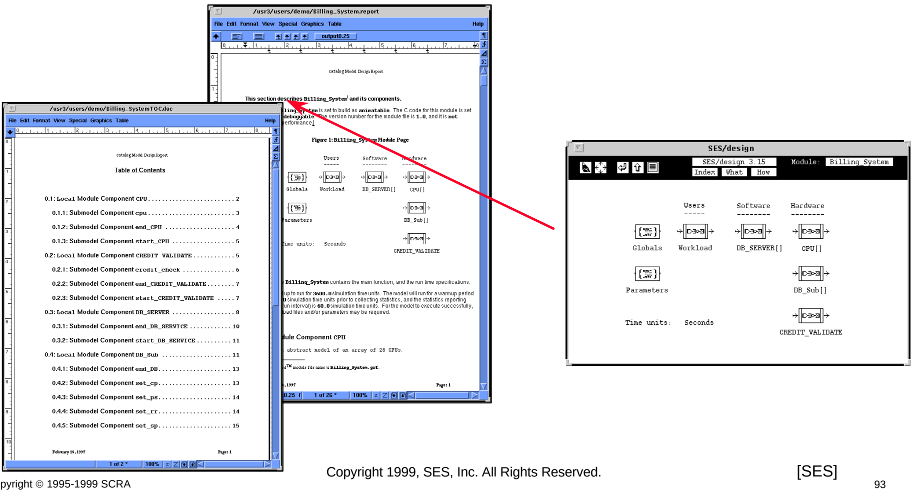


[SES]

See http://www.ses.com

User Extensions to SES/workbench (Cont.)

- **Users can create custom documentation of the system design from the SES/workbench model files**



/usr3/users/demo/Billing_System.report

catalog Model Design Report

This section describes Billing_System[1] and its components.

...ling_System is set to build as animatable. The C code for this module is set ...debuggable. The version number for the module file is 1.0, and it is not ...erformance.

Figure 1: Billing_System Module Page

...Billing_System contains the main function, and the run time specifications.

...up to run for 3600.0 simulation time units. The model will run for a warmup period ...0 simulation time units prior to collecting statistics, and the statistics reporting ...un interval) is 60.0 simulation time units. For the model to execute successfully, ...oad files and/or parameters may be required.

...dule Component CPU

... abstract model of an array of 28 CPUs.

...[TM] module file name is Billing_System.grf.

...,1997 Page: 1

/usr3/users/demo/Billing_SystemTOC.doc

catalog Model Design Report

Table of Contents

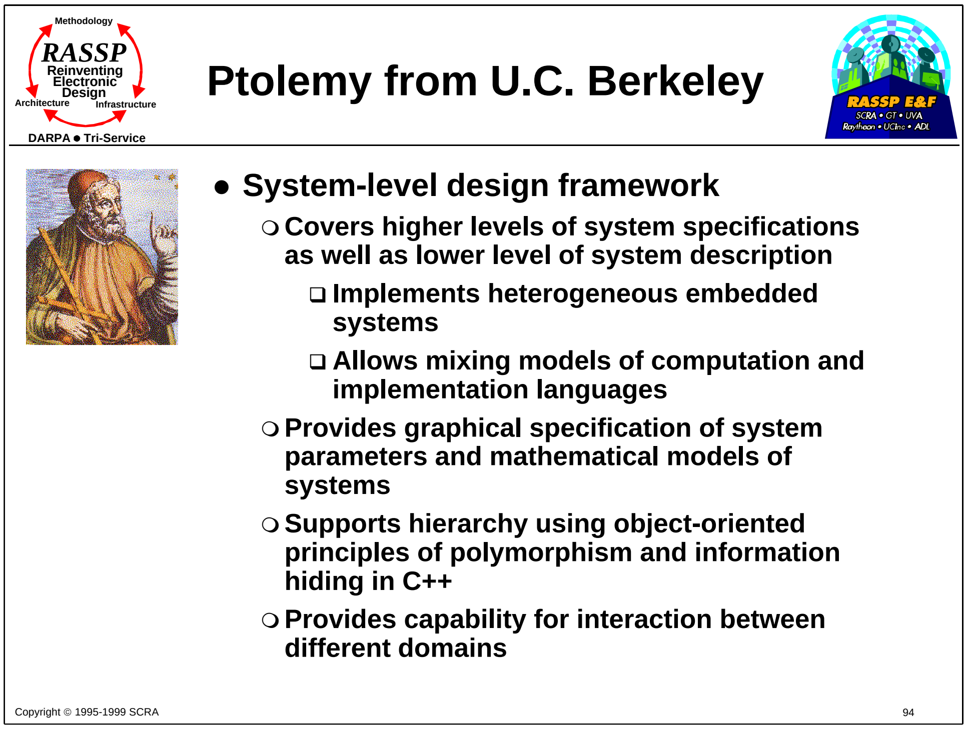
February 18, 1997 Page: 1

SES/design

Copyright 1999, SES, Inc. All Rights Reserved.

[SES]

See http://www.ses.com


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Ptolemy from U.C. Berkeley

- System-level design framework
  - ❍ Covers higher levels of system specifications as well as lower level of system description
    - ❑ Implements heterogeneous embedded systems
    - ❑ Allows mixing models of computation and implementation languages
  - ❍ Provides graphical specification of system parameters and mathematical models of systems
  - ❍ Supports hierarchy using object-oriented principles of polymorphism and information hiding in C++
  - ❍ Provides capability for interaction between different domains

94

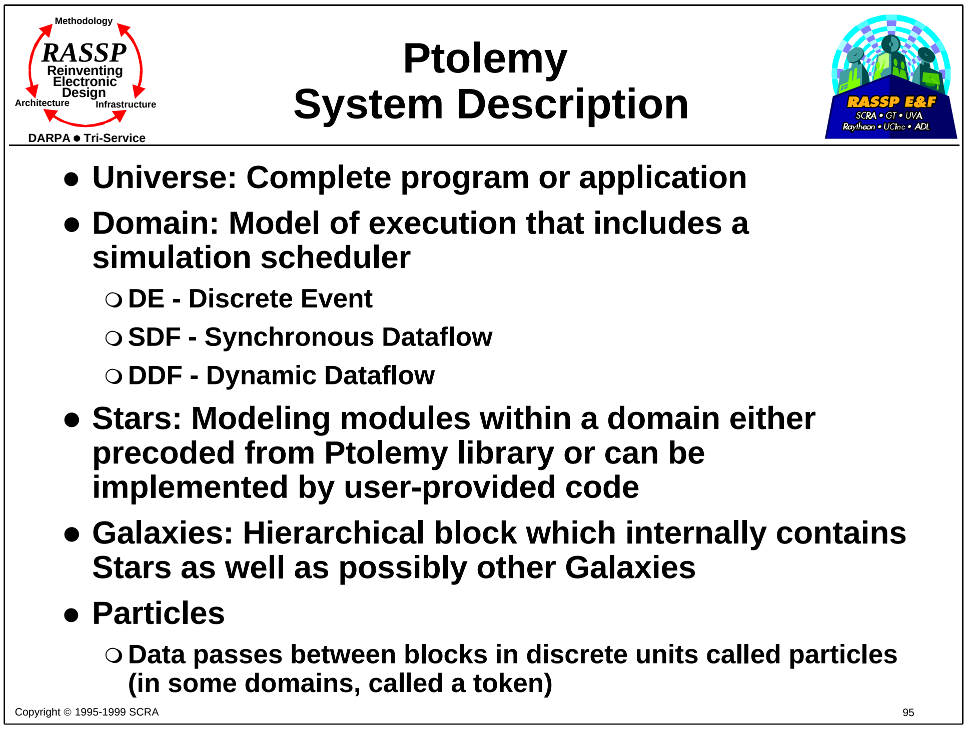
[Ptolemy96].

This section describes UC Berkeley's Ptolemy functional modeling tool. Ptolemy is targeted as a tool to model and simulate the function of a DSP system, but, as is described in this section, it has been used to perform uninterpreted performance modeling.

Biographical Names

Ptol-e-my \'ta^:l-e-me^-\

2d cent. A.D. Claudius Ptolemaeus - Alexandrian astronomer



Ptolemy System Description

- Universe: Complete program or application
- Domain: Model of execution that includes a simulation scheduler
  - ❍ DE - Discrete Event
  - ❍ SDF - Synchronous Dataflow
  - ❍ DDF - Dynamic Dataflow
- Stars: Modeling modules within a domain either precoded from Ptolemy library or can be implemented by user-provided code
- Galaxies: Hierarchical block which internally contains Stars as well as possibly other Galaxies
- Particles
  - ❍ Data passes between blocks in discrete units called particles (in some domains, called a token)

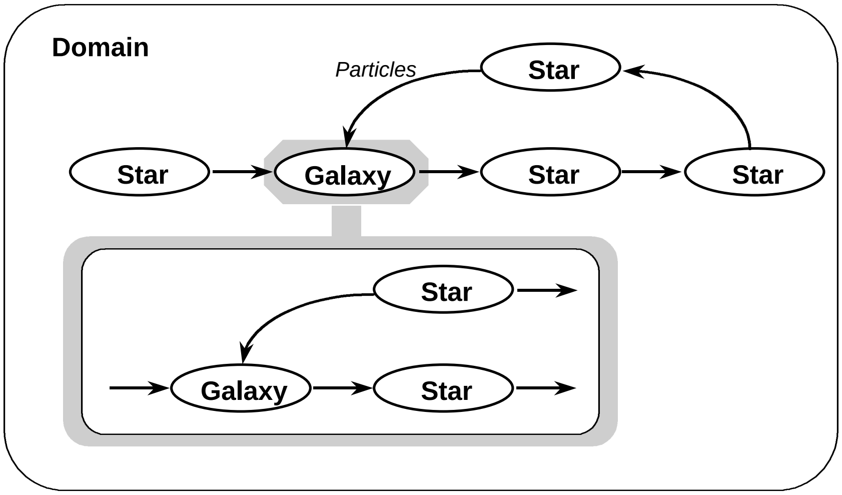
This slide outlines the parts of a Polemy simulation.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Ptolemy System Description (Cont.)

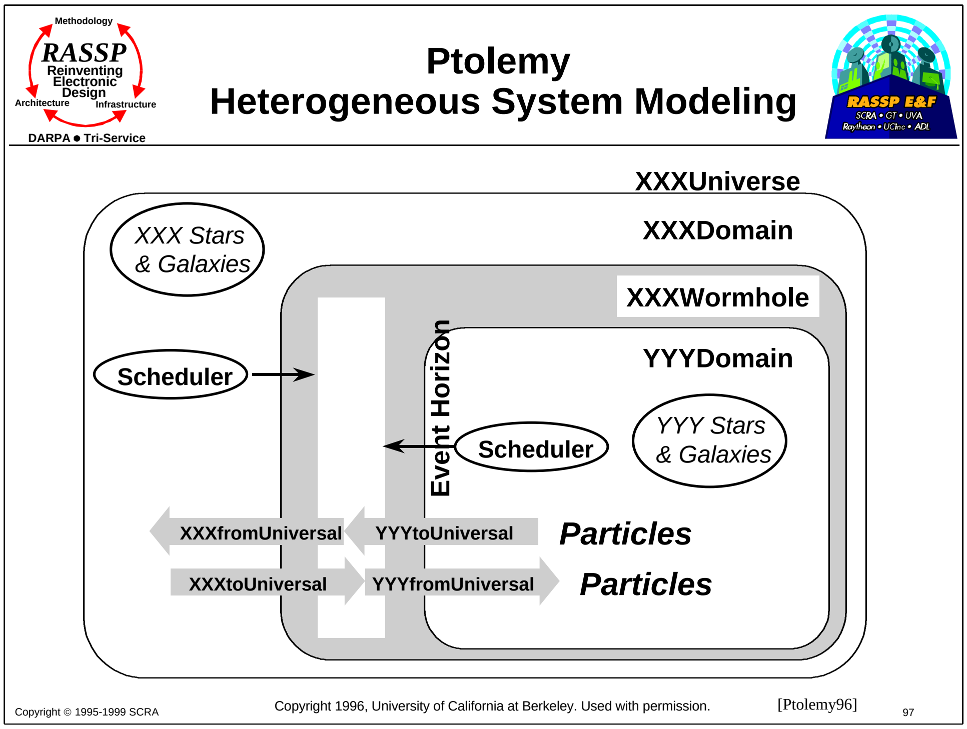
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Universe



Copyright 1996, University of California at Berkeley. Used with permission. [Ptolemy96] 96

This figure shows the general outline of a system model in Ptolemy. General modeling blocks in Ptolemy are called “stars.” A hierarchical collection of stars used to model a large piece of functionality is called a Galaxy. Stars communicate with each other by passing particles (similar to tokens). A specific modeling paradigm in Ptolemy is called a domain. An entire model is Ptolemy is called a Universe.

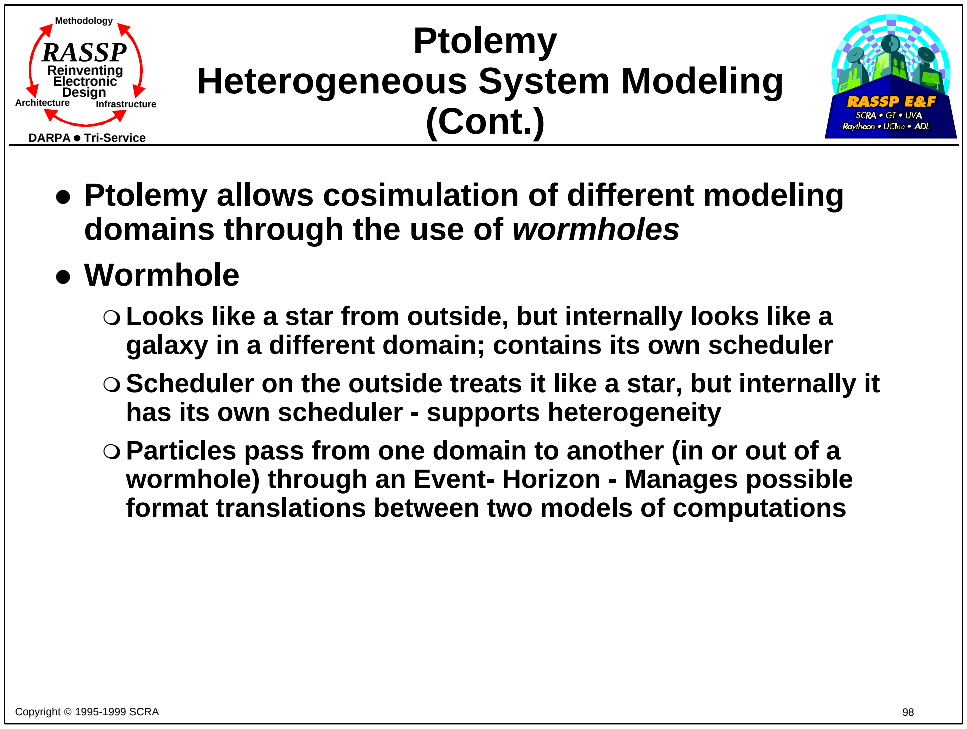


Ptolemy Heterogeneous System Modeling

Copyright 1996, University of California at Berkeley. Used with permission.

[Ptolemy96]

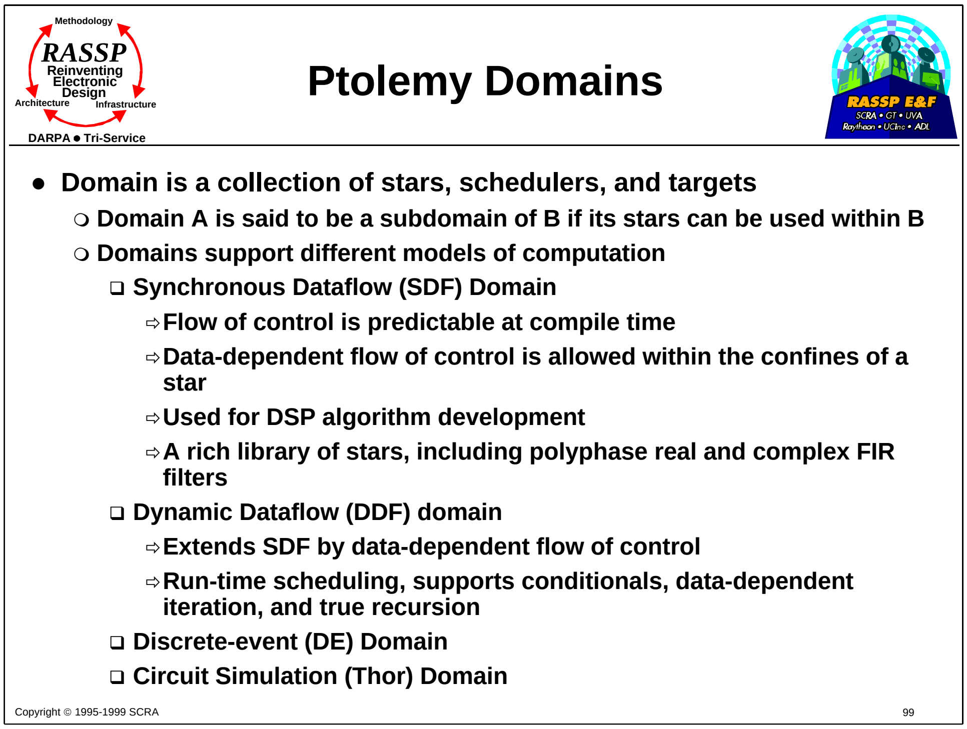
A model of computation (such as discrete event, synchronous dataflow, dynamic dataflow, etc.) is called a Domain in Ptolemy. Each domain includes building blocks, or stars (which the user can add to by writing their own), a scheduler that executes the portion of a model that resides in its domain, and wormholes that interface data and events between domains.


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Ptolemy
Heterogeneous System Modeling (Cont.)

- Ptolemy allows cosimulation of different modeling domains through the use of *wormholes*
- Wormhole
  - ❍ Looks like a star from outside, but internally looks like a galaxy in a different domain; contains its own scheduler
  - ❍ Scheduler on the outside treats it like a star, but internally it has its own scheduler - supports heterogeneity
  - ❍ Particles pass from one domain to another (in or out of a wormhole) through an Event- Horizon - Manages possible format translations between two models of computations

Stars communicate across different domains using wormholes. Wormholes allow heterogeneous models with stars from different domains to be constructed.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

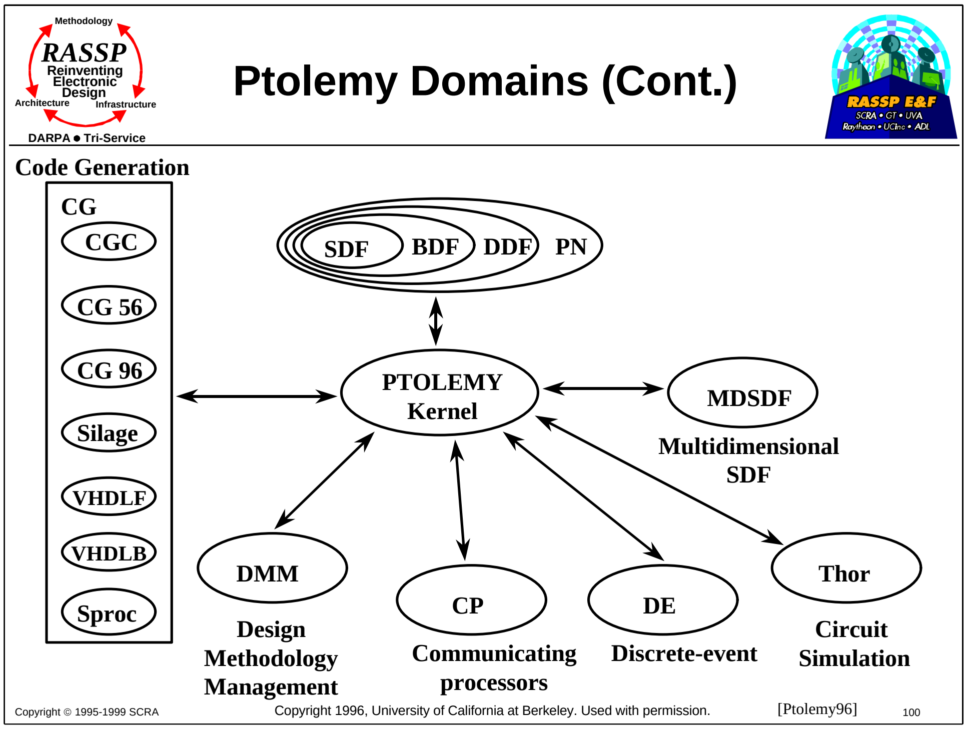
Ptolemy Domains

- Domain is a collection of stars, schedulers, and targets
  - ❍ Domain A is said to be a subdomain of B if its stars can be used within B
  - ❍ Domains support different models of computation
    - ❑ Synchronous Dataflow (SDF) Domain
      - ⇨ Flow of control is predictable at compile time
      - ⇨ Data-dependent flow of control is allowed within the confines of a star
      - ⇨ Used for DSP algorithm development
      - ⇨ A rich library of stars, including polyphase real and complex FIR filters
    - ❑ Dynamic Dataflow (DDF) domain
      - ⇨ Extends SDF by data-dependent flow of control
      - ⇨ Run-time scheduling, supports conditionals, data-dependent iteration, and true recursion
    - ❑ Discrete-event (DE) Domain
    - ❑ Circuit Simulation (Thor) Domain

More discussion of domains.

Example:

> A high-level dataflow model of a signal processing system can be connected to a hardware simulator that in turn may be connected to a discrete-event model of a communication network
>
> BDF domain implements a compile-time scheduler for DDF graphs that supports run-time flow of control; similar to SDF. Attempts to construct a compile-time scheduler - like DDF
>
> - achieves the efficiency of SDF with the generality of DDF.
>
> HOF domain: takes a function as an argument and/or returns a function. It implements a star called Map, that can apply any other star (or galaxy) to the sequence(s) at its inputs thereby “mapping” itself to the other star or galaxy.



Ptolemy Domains (Cont.)

Code Generation

CG

CGC

CG 56

CG 96

Silage

VHDLF

VHDLB

Sproc

SDF BDF DDF PN

PTOLEMY Kernel

MDSDF

Multidimensional SDF

DMM

Design Methodology Management

CP

Communicating processors

DE

Discrete-event

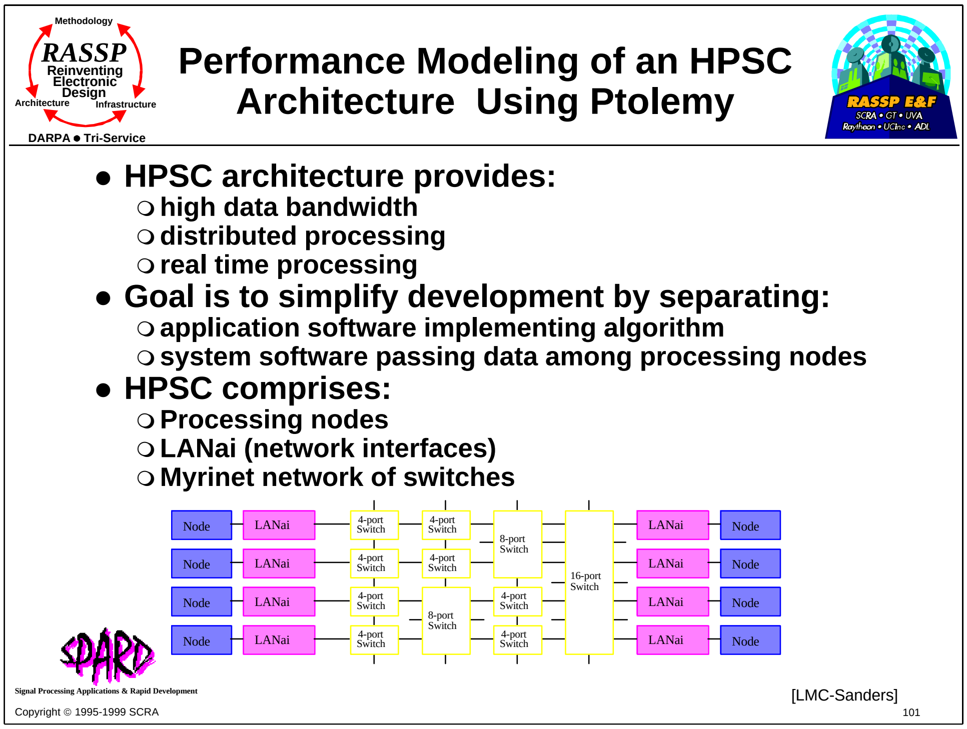
Thor

Circuit Simulation

Copyright 1996, University of California at Berkeley. Used with permission.

[Ptolemy96]

This is a graphical representation of the domains available within Ptolemy and how they interact with the Ptolemy kernel.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

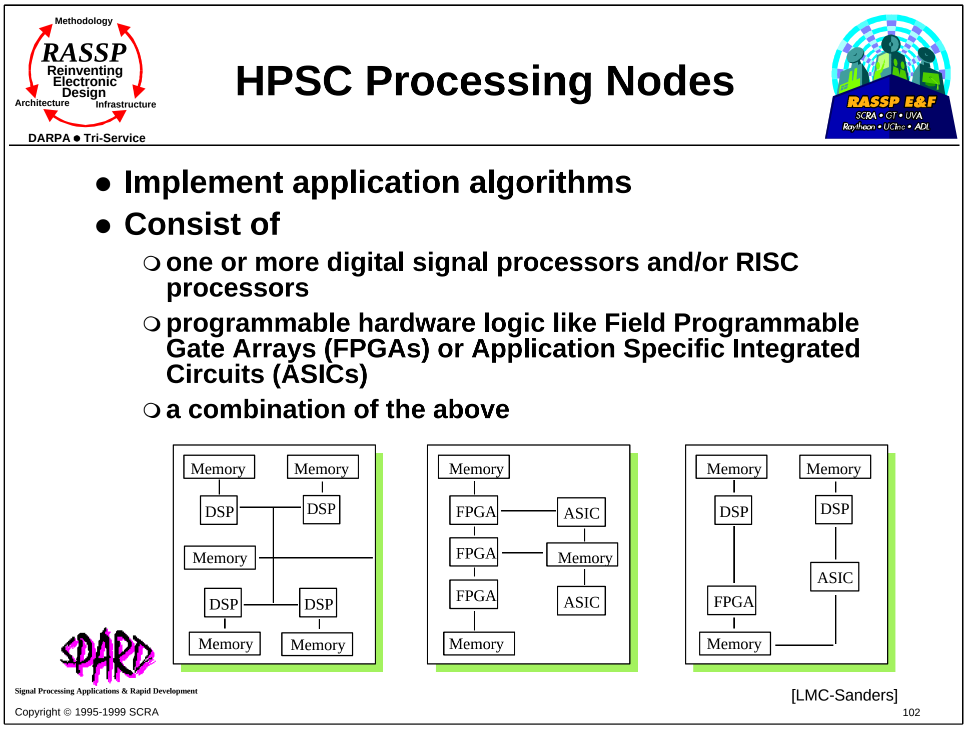
Performance Modeling of an HPSC Architecture Using Ptolemy

- HPSC architecture provides:
  - ❍ high data bandwidth
  - ❍ distributed processing
  - ❍ real time processing
- Goal is to simplify development by separating:
  - ❍ application software implementing algorithm
  - ❍ system software passing data among processing nodes
- HPSC comprises:
  - ❍ Processing nodes
  - ❍ LANai (network interfaces)
  - ❍ Myrinet network of switches

[LMC-Sanders]

101

This is a presentation of how High Performance Scalable Computing systems can be accomplished using Ptolemy. HPSC systems are those types of systems utilized in the RASSP program. This method for performance modeling is described in detail in [Pauer97], so a through reading of that paper will suffice to explain these slides.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

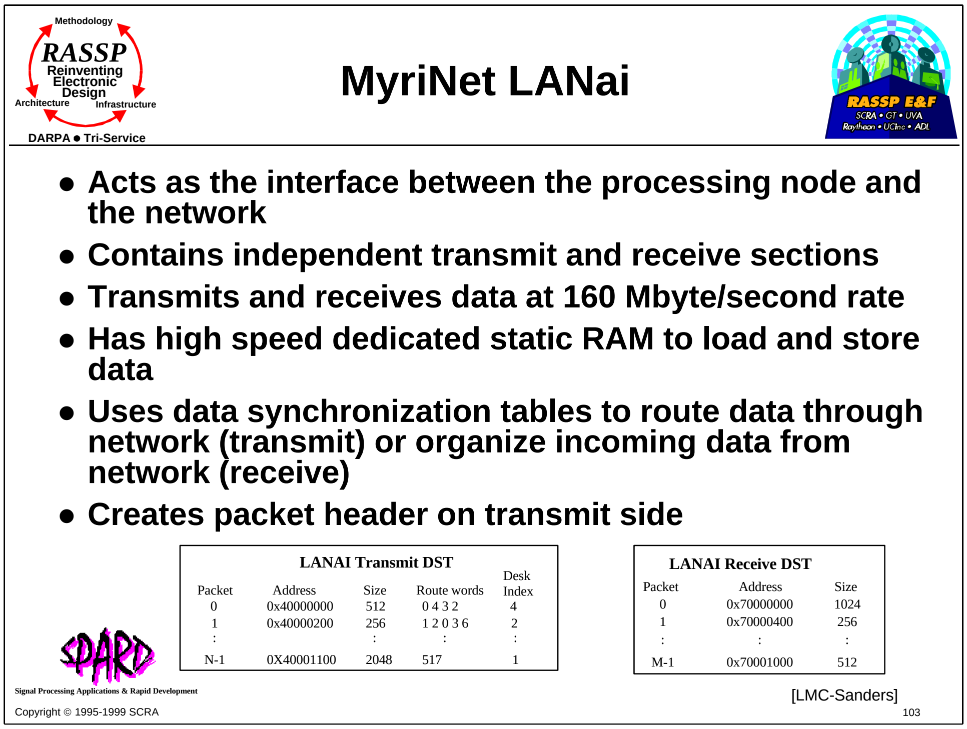
HPSC Processing Nodes

- Implement application algorithms
- Consist of
  - ❍ one or more digital signal processors and/or RISC processors
  - ❍ programmable hardware logic like Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs)
  - ❍ a combination of the above

Memory
Memory
DSP
DSP
Memory
DSP
DSP
Memory
Memory
Memory
FPGA
ASIC
FPGA
Memory
FPGA
ASIC
Memory
Memory
Memory
DSP
DSP
ASIC
FPGA
Memory
SPARD
Signal Processing Applications & Rapid Development

[LMC-Sanders]

See [Pauer97].



Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service

MyriNet LANai

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Acts as the interface between the processing node and the network
- Contains independent transmit and receive sections
- Transmits and receives data at 160 Mbyte/second rate
- Has high speed dedicated static RAM to load and store data
- Uses data synchronization tables to route data through network (transmit) or organize incoming data from network (receive)
- Creates packet header on transmit side

LANAI Transmit DST

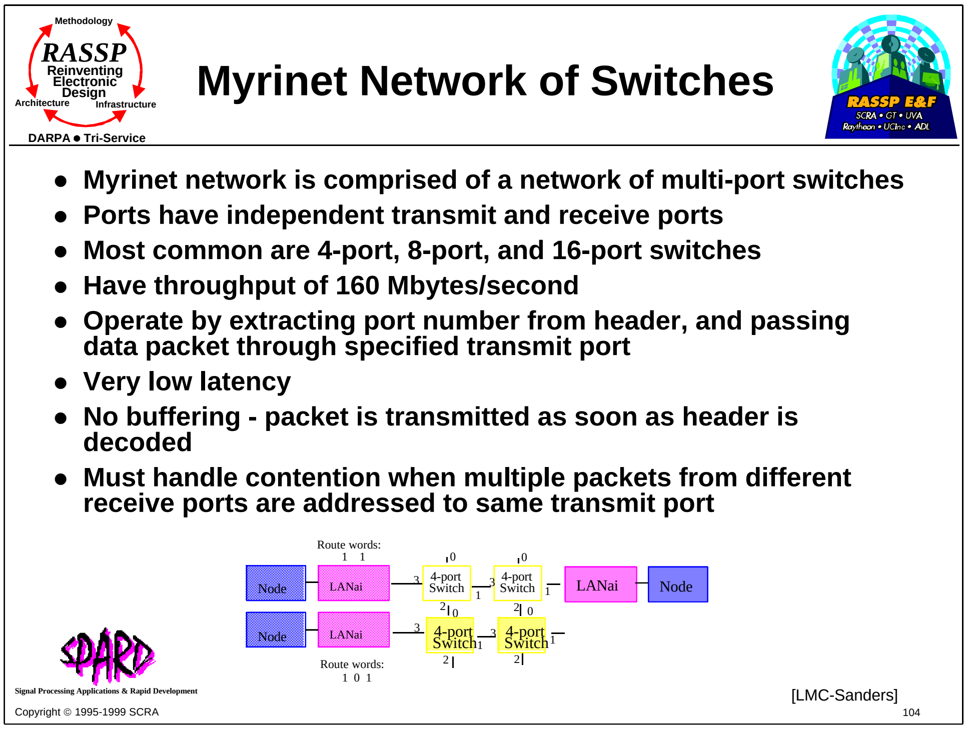
Packet	Address	Size	Route words	Desk Index
0	0x40000000	512	0 4 3 2	4
1	0x40000200	256	1 2 0 3 6	2
:		:	:	:
N-1	0X40001100	2048	517	1

LANAI Receive DST

Packet	Address	Size
0	0x70000000	1024
1	0x70000400	256
:	:	:
M-1	0x70001000	512

[LMC-Sanders]

See [Pauer97].


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Myrinet Network of Switches

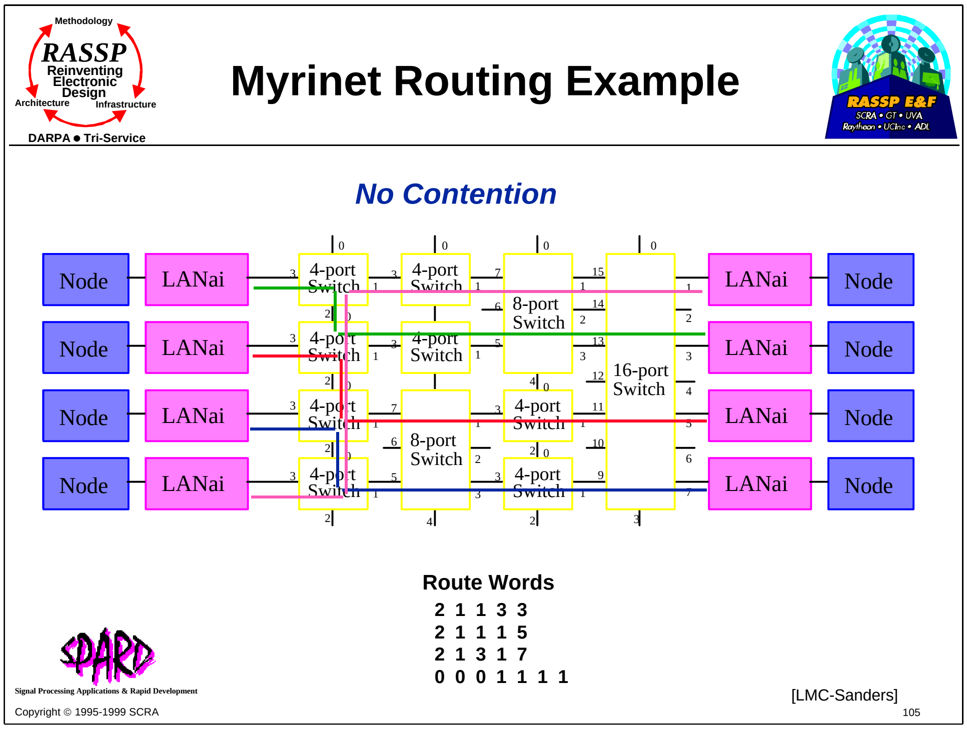
- Myrinet network is comprised of a network of multi-port switches
- Ports have independent transmit and receive ports
- Most common are 4-port, 8-port, and 16-port switches
- Have throughput of 160 Mbytes/second
- Operate by extracting port number from header, and passing data packet through specified transmit port
- Very low latency
- No buffering - packet is transmitted as soon as header is decoded
- Must handle contention when multiple packets from different receive ports are addressed to same transmit port

[LMC-Sanders]

Copyright © 1995-1999 SCRA

104

See [Pauer97].



Myrinet Routing Example

No Contention

Route Words

2 1 1 3 3
2 1 1 1 5
2 1 3 1 7
0 0 0 1 1 1 1

[LMC-Sanders]

105

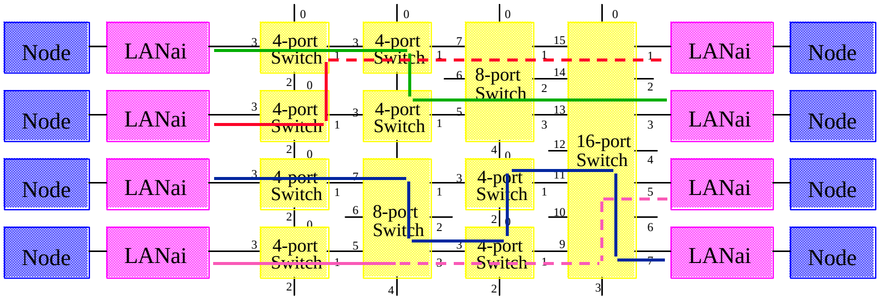
See [Pauer97].

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Myrinet Routing Example

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Contention



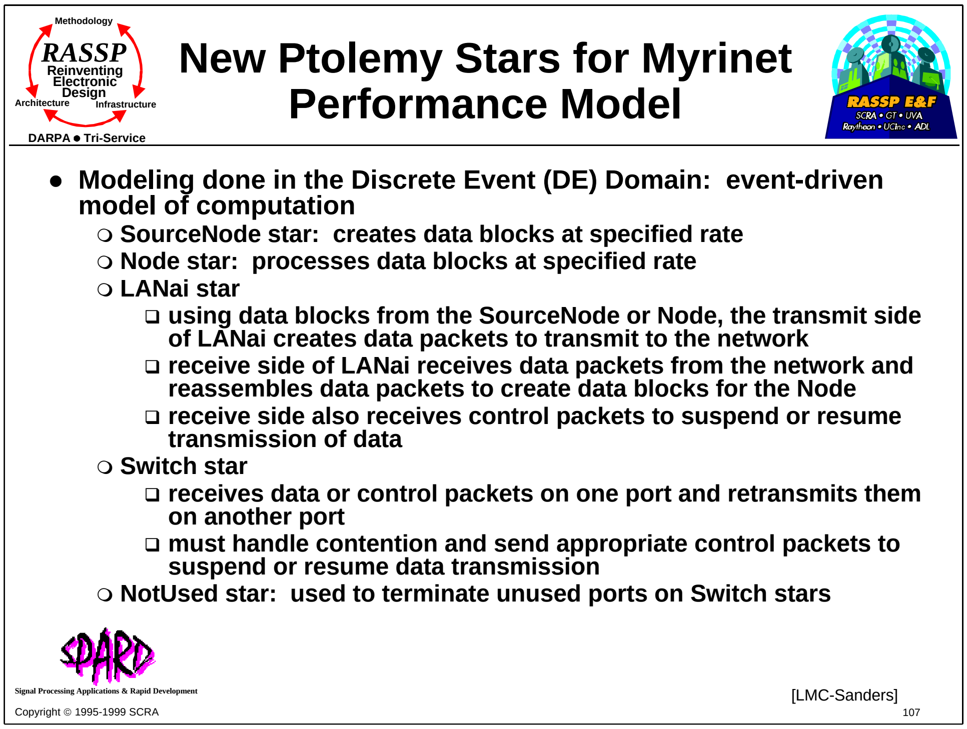
Route Words

1 2 1 3 3
0 1 1 1 1
1 3 0 1 7
1 3 1 5

Signal Processing Applications & Rapid Development

[LMC-Sanders]

See [Pauer97].



New Ptolemy Stars for Myrinet Performance Model

- Modeling done in the Discrete Event (DE) Domain: event-driven model of computation
  - ❍ SourceNode star: creates data blocks at specified rate
  - ❍ Node star: processes data blocks at specified rate
  - ❍ LANai star
    - ❑ using data blocks from the SourceNode or Node, the transmit side of LANai creates data packets to transmit to the network
    - ❑ receive side of LANai receives data packets from the network and reassembles data packets to create data blocks for the Node
    - ❑ receive side also receives control packets to suspend or resume transmission of data
  - ❍ Switch star
    - ❑ receives data or control packets on one port and retransmits them on another port
    - ❑ must handle contention and send appropriate control packets to suspend or resume data transmission
  - ❍ NotUsed star: used to terminate unused ports on Switch stars

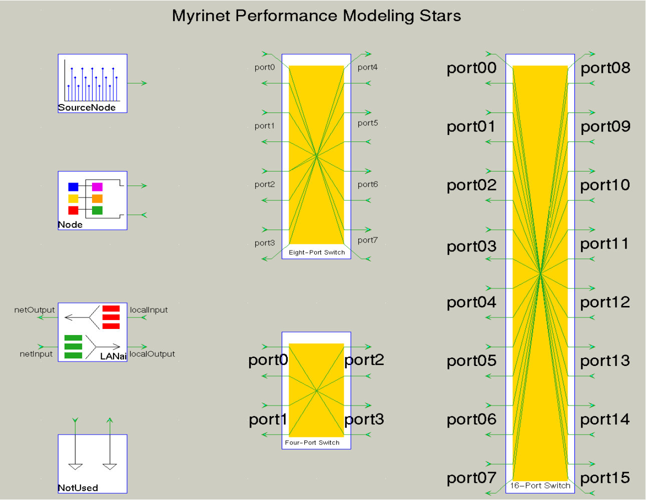
[LMC-Sanders]

See [Pauer97].

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

New Ptolemy Performance Modeling Stars for Myrinet

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

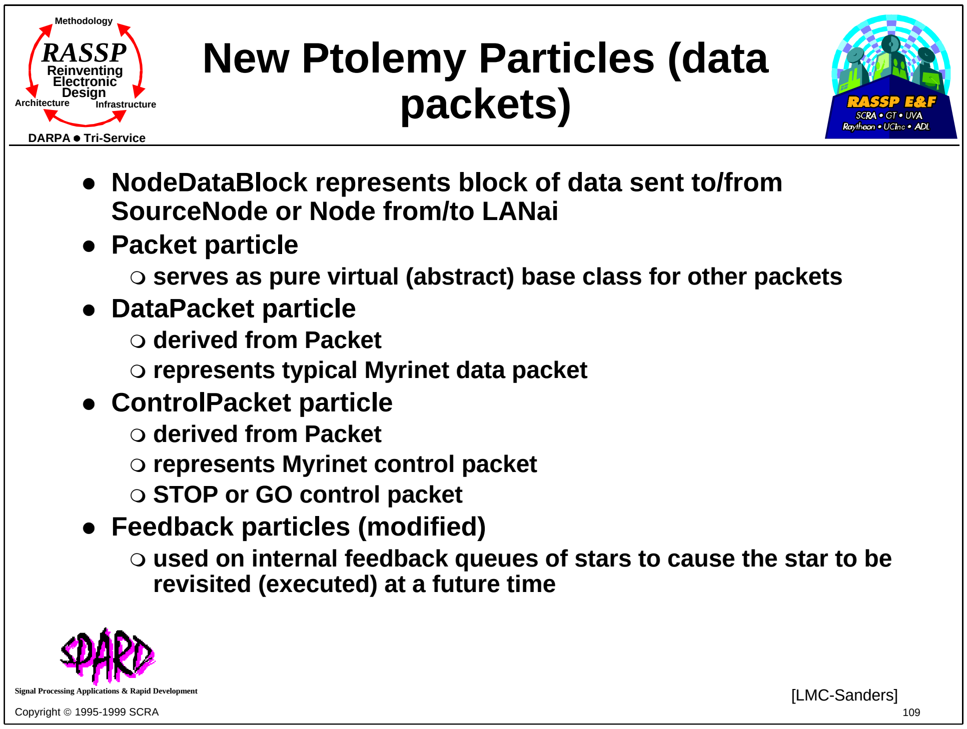


SPARD
Signal Processing Applications & Rapid Development

[LMC-Sanders]

108

See [Pauer97].



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

New Ptolemy Particles (data packets)

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL

- NodeDataBlock represents block of data sent to/from SourceNode or Node from/to LANai
- Packet particle
  - ❍ serves as pure virtual (abstract) base class for other packets
- DataPacket particle
  - ❍ derived from Packet
  - ❍ represents typical Myrinet data packet
- ControlPacket particle
  - ❍ derived from Packet
  - ❍ represents Myrinet control packet
  - ❍ STOP or GO control packet
- Feedback particles (modified)
  - ❍ used on internal feedback queues of stars to cause the star to be revisited (executed) at a future time

SPARD

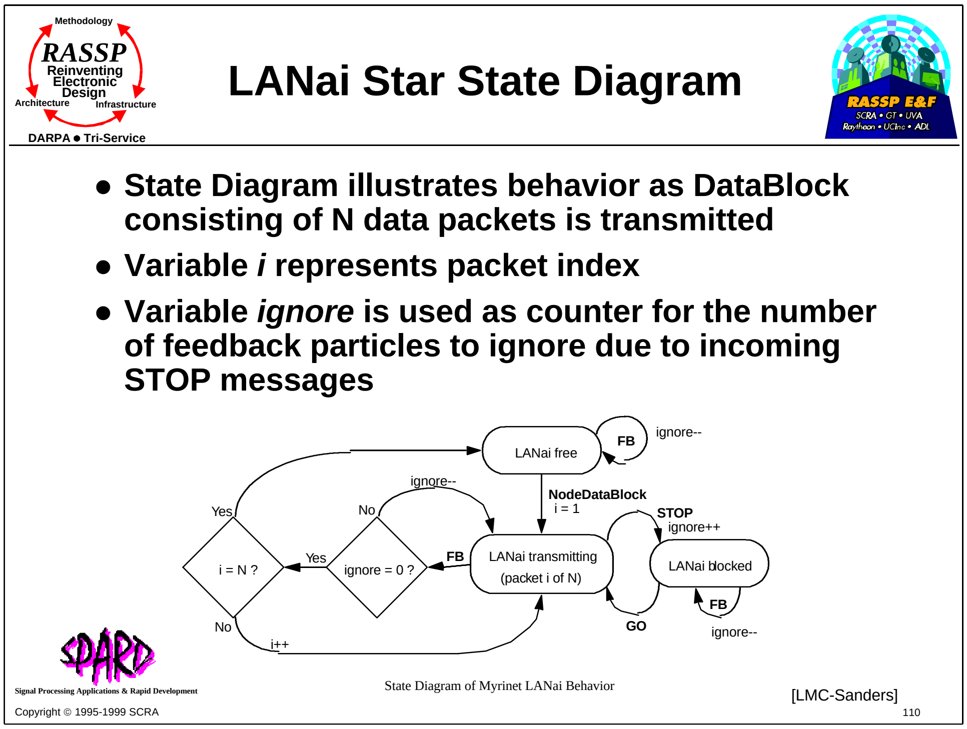
Signal Processing Applications & Rapid Development

Copyright © 1995-1999 SCRA

[LMC-Sanders]

109

See [Pauer97].



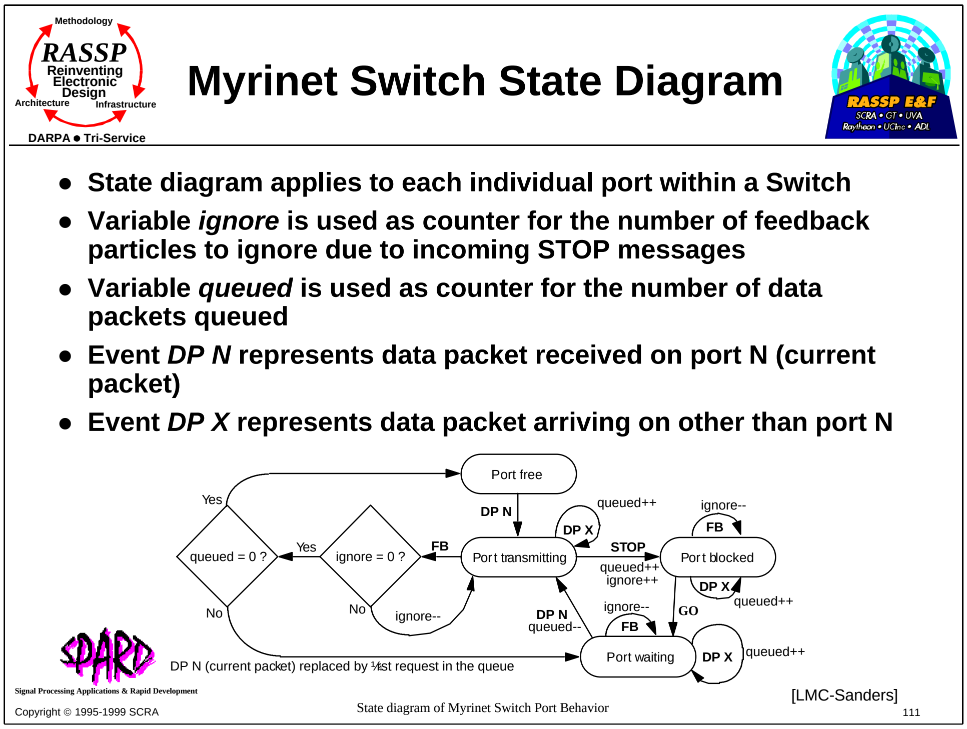
LANai Star State Diagram

- State Diagram illustrates behavior as DataBlock consisting of N data packets is transmitted
- Variable *i* represents packet index
- Variable *ignore* is used as counter for the number of feedback particles to ignore due to incoming STOP messages

State Diagram of Myrinet LANai Behavior

[LMC-Sanders]

See [Pauer97].



Myrinet Switch State Diagram

- State diagram applies to each individual port within a Switch
- Variable ***ignore*** is used as counter for the number of feedback particles to ignore due to incoming STOP messages
- Variable ***queued*** is used as counter for the number of data packets queued
- Event ***DP N*** represents data packet received on port N (current packet)
- Event ***DP X*** represents data packet arriving on other than port N

State diagram of Myrinet Switch Port Behavior

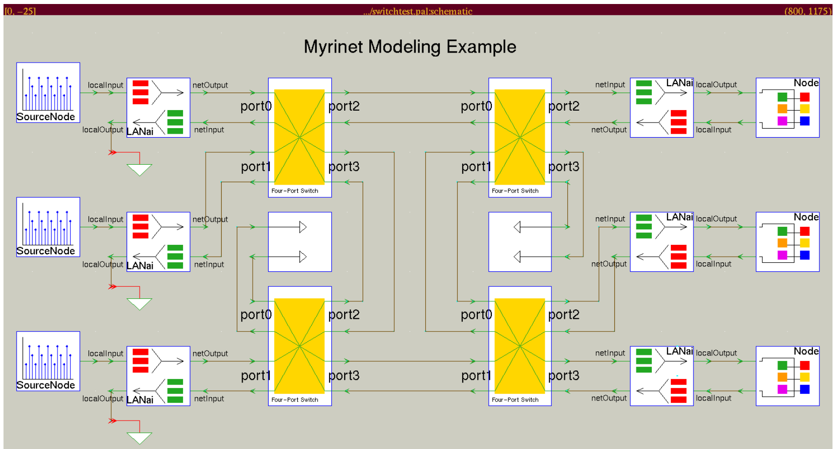
[LMC-Sanders]

See [Pauer97].

Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service

Simple 4 Switch Network Modeling Example

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL



Signal Processing Applications & Rapid Development

[LMC-Sanders]

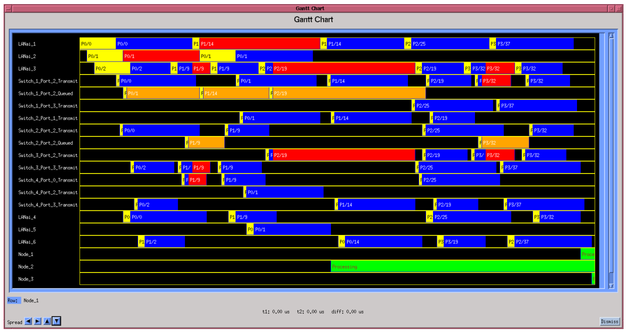
See [Pauer97].

Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

Results for Simple Network Example

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL



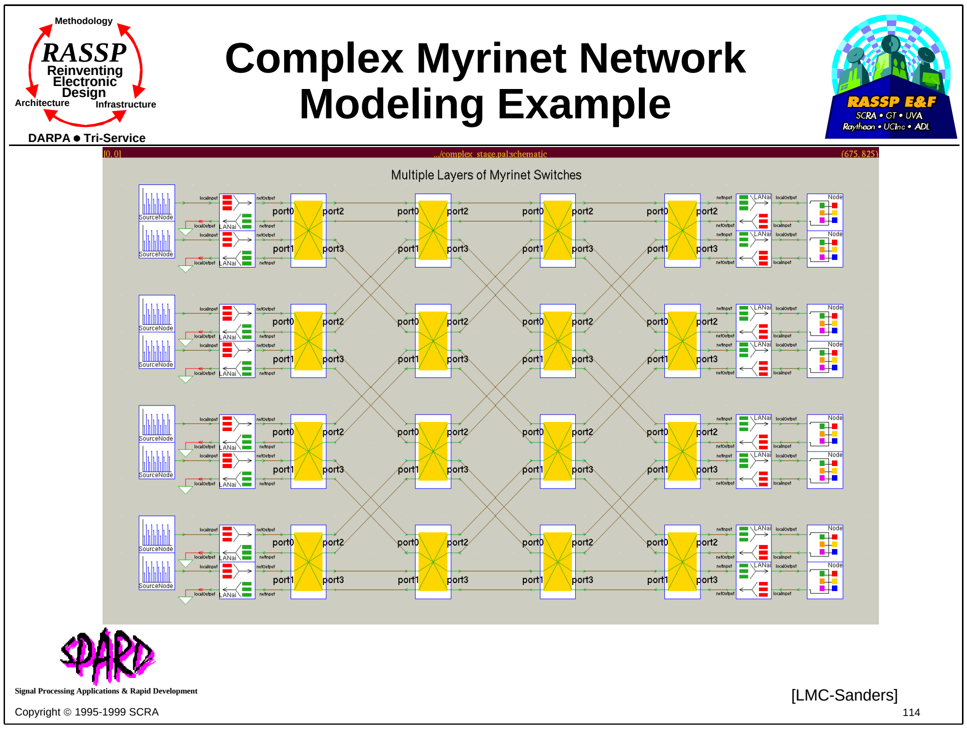
Gantt Tool Display of Simple Myrinet Modeling Example

- **Yellow: start-up latency**
- **Blue: normal transmission/reception**
- **Green: processing of data on Node**
- **Orange: origin of contention, one or more packets queued in the switch**
- **Red: propagating effect of switch contention down current data path**

Signal Processing Applications & Rapid Development

[LMC-Sanders]

See [Pauer97].



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

Complex Myrinet Network Modeling Example

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL

[0, 0]
.../complex_stage.pal.schematic
(675, 825)
Multiple Layers of Myrinet Switches
SourceNode
LANai
port0
port1
port2
port3
Node

Signal Processing Applications & Rapid Development

[LMC-Sanders]

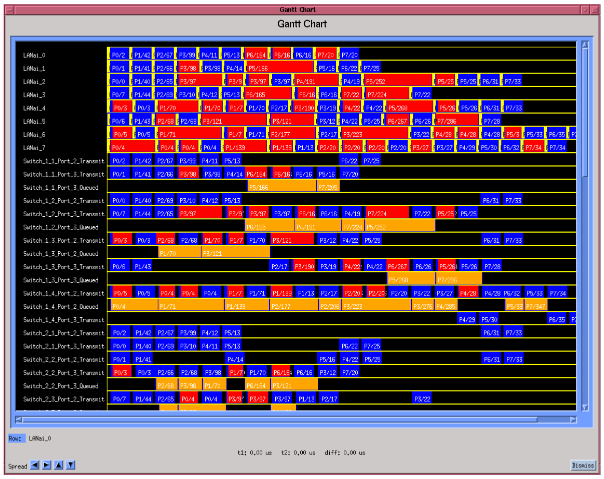
See [Pauer97].

Methodology
RASSP Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Results for Complex Myrinet Network Example

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Yellow: start-up latency
- Blue: normal transmission/reception
- Green: processing of data on Node
- Orange: origin of contention, one or more packets queued in the switch
- Red: propagating effect of switch contention down current data path

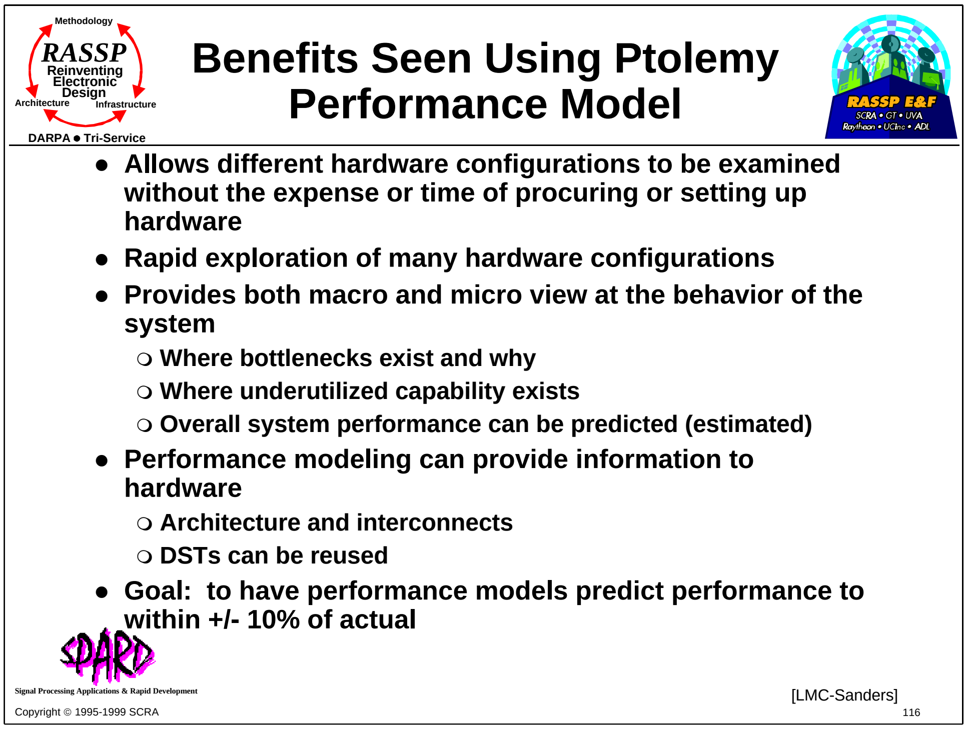


SPARD
Signal Processing Applications & Rapid Development

[LMC-Sanders]

115

See [Pauer97].


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL
SPARD

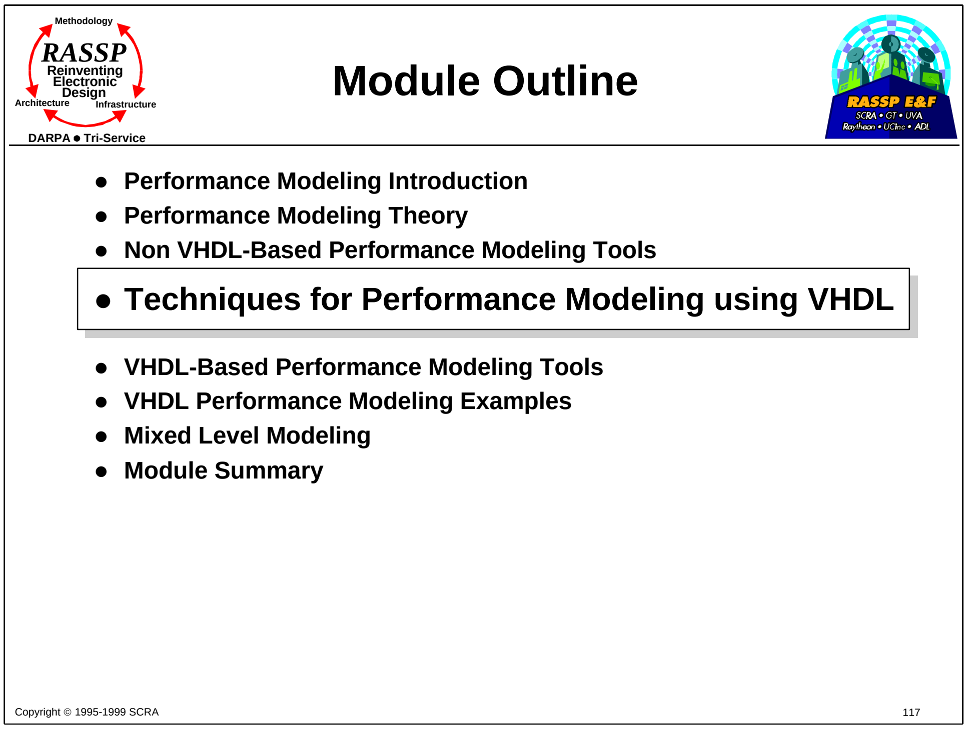
Benefits Seen Using Ptolemy Performance Model

- Allows different hardware configurations to be examined without the expense or time of procuring or setting up hardware
- Rapid exploration of many hardware configurations
- Provides both macro and micro view at the behavior of the system
  - ❍ Where bottlenecks exist and why
  - ❍ Where underutilized capability exists
  - ❍ Overall system performance can be predicted (estimated)
- Performance modeling can provide information to hardware
  - ❍ Architecture and interconnects
  - ❍ DSTs can be reused
- Goal: to have performance models predict performance to within +/- 10% of actual

Signal Processing Applications & Rapid Development

[LMC-Sanders]

See [Pauer97].



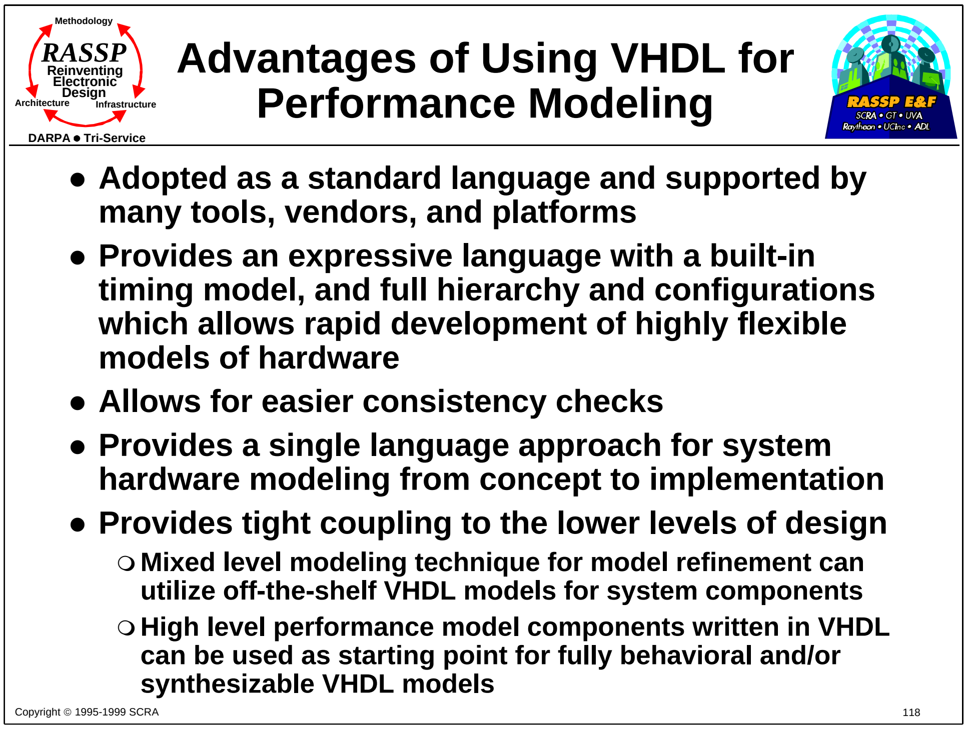
Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Module Outline

- Performance Modeling Introduction
- Performance Modeling Theory
- Non VHDL-Based Performance Modeling Tools
- **Techniques for Performance Modeling using VHDL**
- VHDL-Based Performance Modeling Tools
- VHDL Performance Modeling Examples
- Mixed Level Modeling
- Module Summary

Module Outline

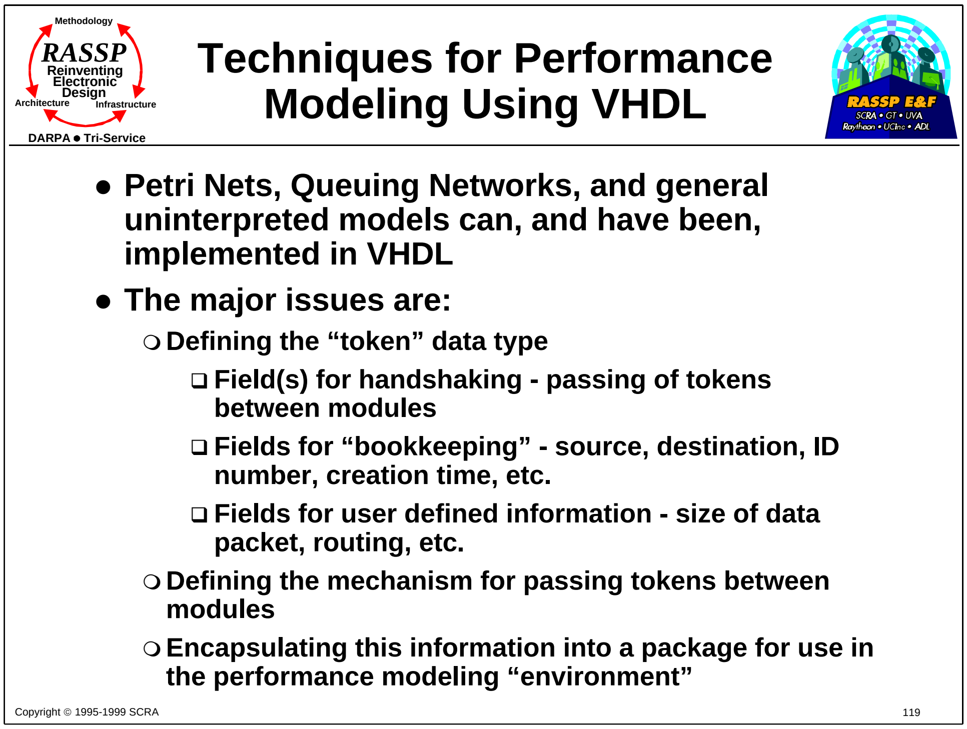


Advantages of Using VHDL for Performance Modeling

- Adopted as a standard language and supported by many tools, vendors, and platforms
- Provides an expressive language with a built-in timing model, and full hierarchy and configurations which allows rapid development of highly flexible models of hardware
- Allows for easier consistency checks
- Provides a single language approach for system hardware modeling from concept to implementation
- Provides tight coupling to the lower levels of design
  - Mixed level modeling technique for model refinement can utilize off-the-shelf VHDL models for system components
  - High level performance model components written in VHDL can be used as starting point for fully behavioral and/or synthesizable VHDL models

As a hardware description language, VHDL has many desirable features for describing hardware already built-in such a a timing model, support for design hierarchy and configuration, etc. A general programming language such as C or C++ has none of these things.

A single language approach is beneficial because it means that hardware designers can work in VHDL to describe their components at all levels from the system level on down. Also, the system level VHDL models can be a starting point for fully behavioral or even synthesizable VHDL models of components.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure DARPA ● Tri-Service

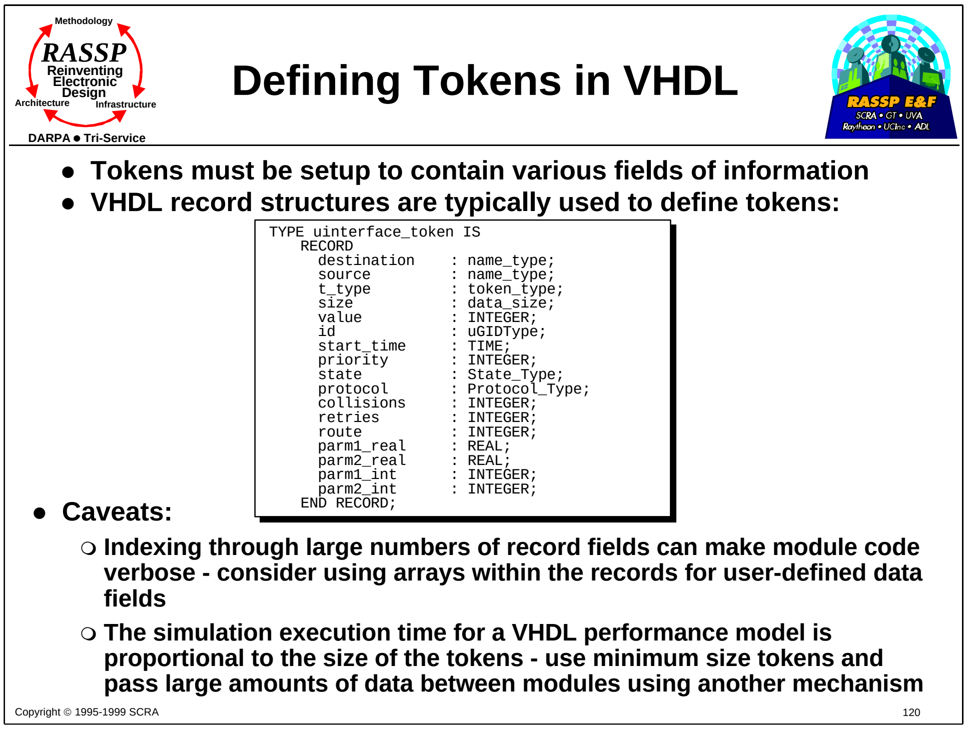
Techniques for Performance Modeling Using VHDL

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

- Petri Nets, Queuing Networks, and general uninterpreted models can, and have been, implemented in VHDL
- The major issues are:
  - ❍ Defining the “token” data type
    - ❑ Field(s) for handshaking - passing of tokens between modules
    - ❑ Fields for “bookkeeping” - source, destination, ID number, creation time, etc.
    - ❑ Fields for user defined information - size of data packet, routing, etc.
  - ❍ Defining the mechanism for passing tokens between modules
  - ❍ Encapsulating this information into a package for use in the performance modeling “environment”

Traditional performance modeling methods such as queuing models and Petri Nets, have been implemented in VHDL by UVa and others, as have more general uninterpreted performance modeling environments.

The major issues in this type of modeling effort in VHDL are discussed above.


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Defining Tokens in VHDL

- Tokens must be setup to contain various fields of information
- VHDL record structures are typically used to define tokens:

```
TYPE uinterface_token IS
    RECORD
      destination    : name_type;
      source         : name_type;
      t_type         : token_type;
      size           : data_size;
      value          : INTEGER;
      id             : uGIDType;
      start_time     : TIME;
      priority       : INTEGER;
      state          : State_Type;
      protocol       : Protocol_Type;
      collisions     : INTEGER;
      retries        : INTEGER;
      route          : INTEGER;
      parm1_real     : REAL;
      parm2_real     : REAL;
      parm1_int      : INTEGER;
      parm2_int      : INTEGER;
    END RECORD;
```

- Caveats:
  - ❍ Indexing through large numbers of record fields can make module code verbose - consider using arrays within the records for user-defined data fields
  - ❍ The simulation execution time for a VHDL performance model is proportional to the size of the tokens - use minimum size tokens and pass large amounts of data between modules using another mechanism

This slide includes the source code (somewhat modified) for the generic interface token developed by Honeywell Technology Center as an example.

Tokens in VHDL are probably best described as records. However, if large numbers of user defined fields are to be included, it is sometimes better to define those as arrays within the record structure. This allows the code that accesses the user defined fields to do so with loops and to index them easily (e.g., token.user_array(value_one)).

Another issue to consider is that it has become apparent that the size of the token has a great influence on the simulation time of the model, especially if a bus resolution function is used to pass tokens between modules.

RASSP Reinventing Electronic Design

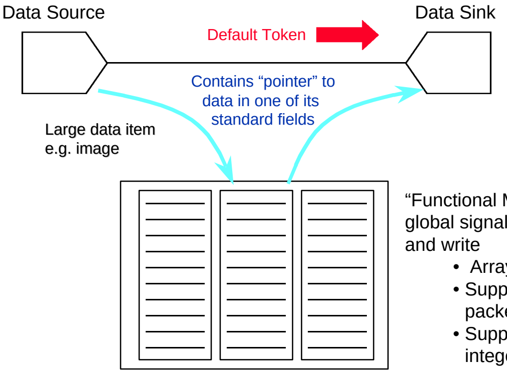
Methodology Architecture Infrastructure

DARPA ● Tri-Service

Passing Large Amounts of Data Between Modules in VHDL

RASSP E&F SCRA • GT • UVA Raytheon • UCInc • ADL

- **Define token as small as possible to reduce simulation time**
- **Use Honeywell's "functional memory" concept to pass data that will not fit into the standard token**



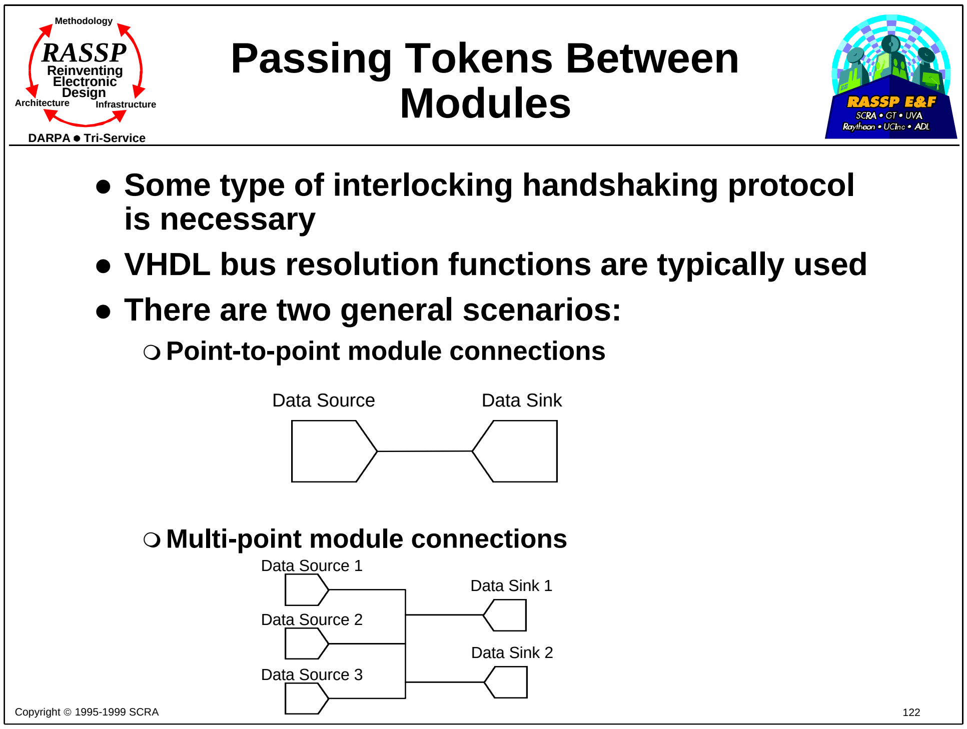
Data Source
Data Sink
Default Token
Contains "pointer" to data in one of its standard fields
Large data item e.g. image

"Functional Memory" implemented as global signal - all modules can read and write

- Array of stacks
- Support for variable size data packets
- Support for standard types - integer, real, etc.

The problem with passing large amounts of data in a token is that large tokens slow down the VHDL simulation greatly. Also, if only one token signal in a given model needs to carry a large amount of information, all tokens will be large (because they all have to be the same size) which is a waste of simulation speed and memory.

A solution developed by Honeywell as part of their PML (to be presented later) is to have a global signal, declared in a package and visible to all architectures, that can be used as a storage space to pass large amounts of data. Modules that want to pass data write it into this "functional memory" which is implemented as an array of stacks supporting generic types like integers and reals, and pass pointers to the information to other modules in one of the standard token fields. These other modules can then read the information out of the functional memory as required.



Passing Tokens Between Modules

- Some type of interlocking handshaking protocol is necessary
- VHDL bus resolution functions are typically used
- There are two general scenarios:
  - ❍ Point-to-point module connections

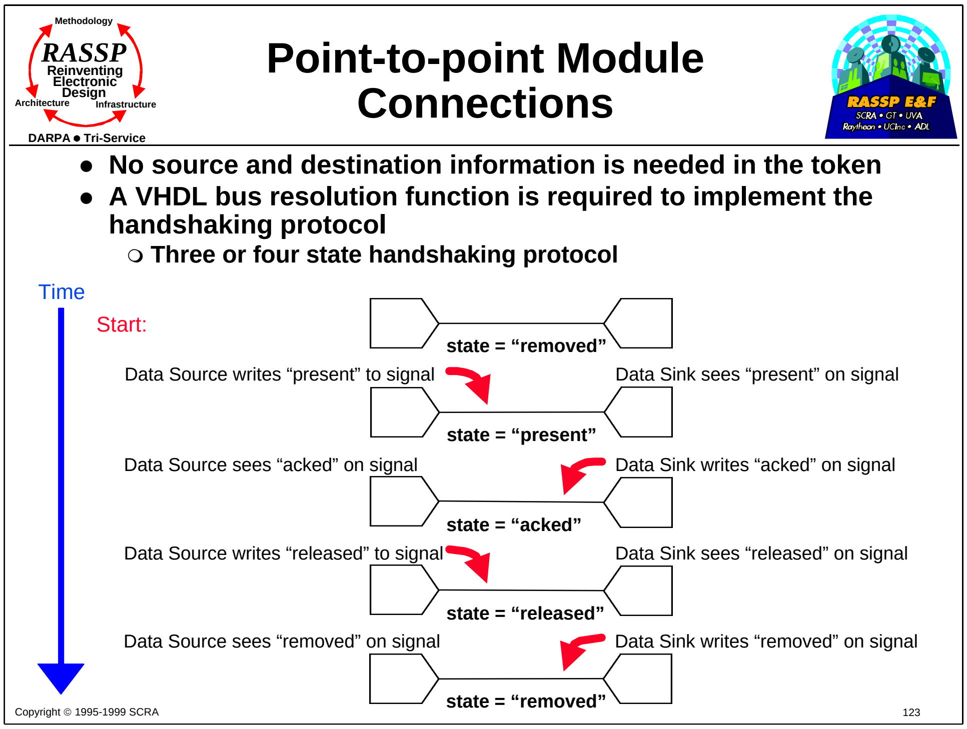
Data Source
Data Sink

  - ❍ Multi-point module connections

Data Source 1
Data Source 2
Data Source 3
Data Sink 1
Data Sink 2

Some type if interlocking mechanism to pass tokens from one module to another is necessary. VHDL bus resolution functions are typically used, both in the point-to-point and multiple driver/reader case, because the token signal is bi-directional. That is, the data source has to be able to drive the new token onto the signal and the data destination has to be able to drive the acknowledgement onto the signal. The two sources require a resolution function.

An alternative (used in the ATL models and in the latest version of ADEPT) is to have unidirectional signals, one from source to destination to place the initial token, and another from the destination to the source to acknowledge the token.



Point-to-point Module Connections

- No source and destination information is needed in the token
- A VHDL bus resolution function is required to implement the handshaking protocol
  - ❍ Three or four state handshaking protocol

Time

Start:

state = “removed”

Data Source writes “present” to signal

Data Sink sees “present” on signal

state = “present”

Data Source sees “acked” on signal

Data Sink writes “acked” on signal

state = “acked”

Data Source writes “released” to signal

Data Sink sees “released” on signal

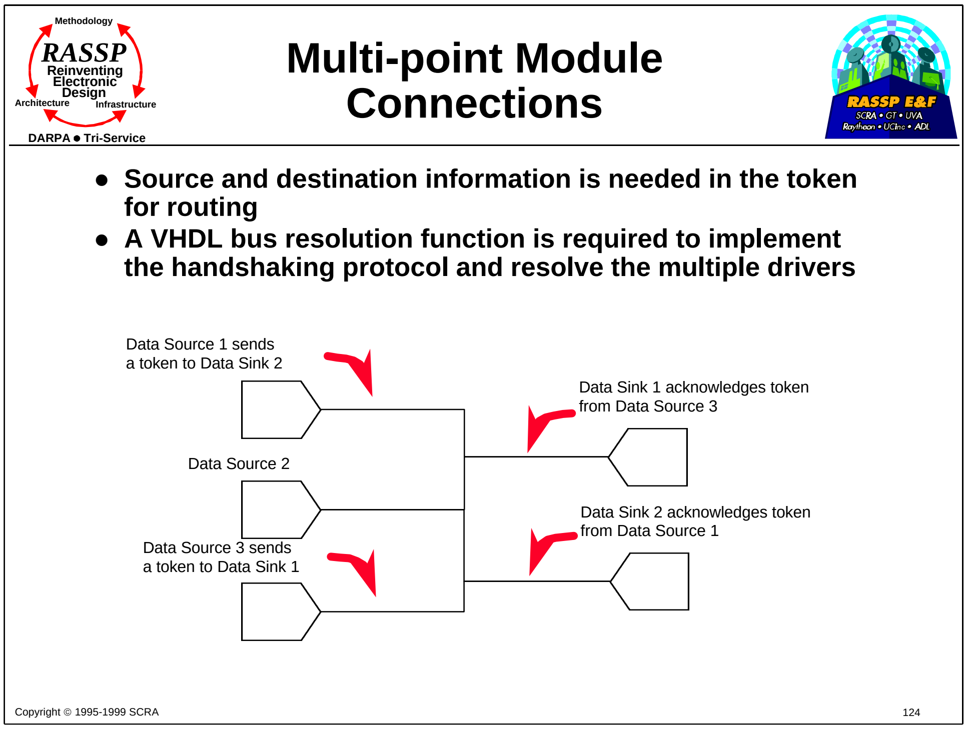
state = “released”

Data Source sees “removed” on signal

Data Sink writes “removed” on signal

state = “removed”

This is an example of how a four state, point-to-point token passing protocol works and why it need a resolution function (taken from ADEPT).



Multi-point Module Connections

- Source and destination information is needed in the token for routing
- A VHDL bus resolution function is required to implement the handshaking protocol and resolve the multiple drivers

Data Source 1 sends a token to Data Sink 2

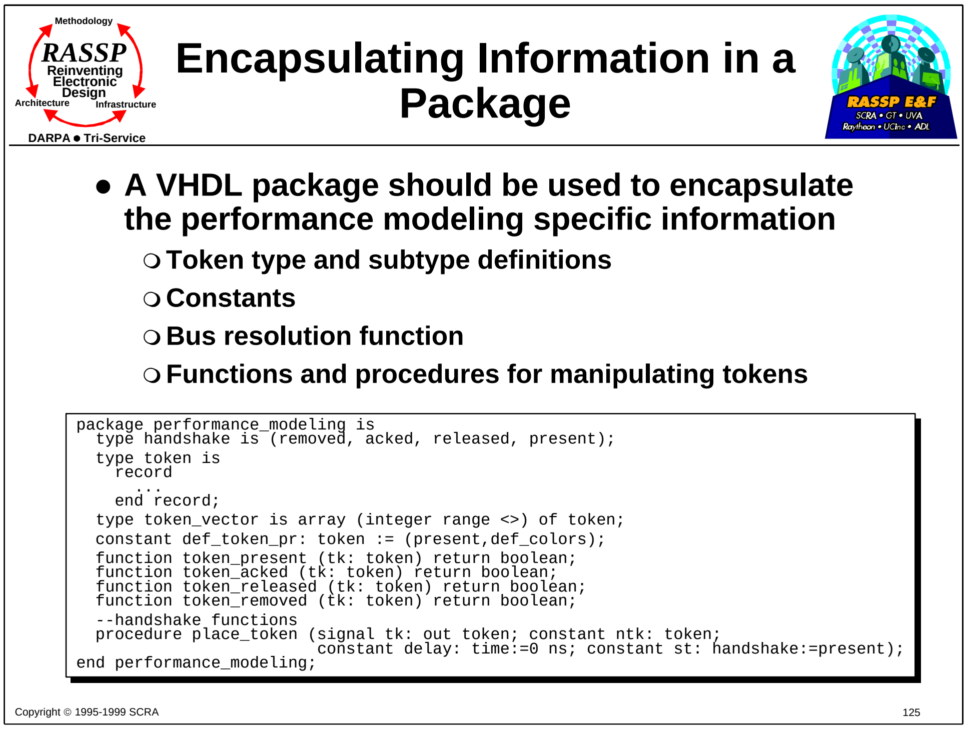
Data Sink 1 acknowledges token from Data Source 3

Data Source 2

Data Sink 2 acknowledges token from Data Source 1

Data Source 3 sends a token to Data Sink 1

This is a multipoint communications protocol. Why a bus resolution function is needed here is self-evident. This is the token passing protocol used in the Honeywell PML, eArchitect.

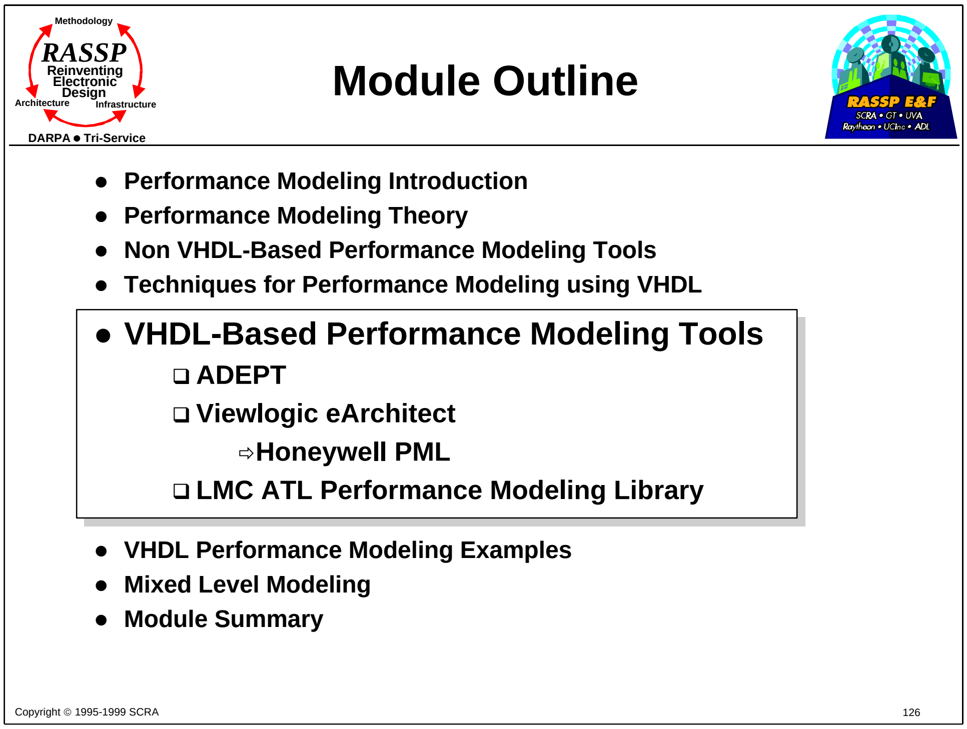


Encapsulating Information in a Package

- A VHDL package should be used to encapsulate the performance modeling specific information
  - ❍ Token type and subtype definitions
  - ❍ Constants
  - ❍ Bus resolution function
  - ❍ Functions and procedures for manipulating tokens

```
package performance_modeling is
  type handshake is (removed, acked, released, present);
  type token is
    record
      ...
    end record;
  type token_vector is array (integer range <>) of token;
  constant def_token_pr: token := (present,def_colors);
  function token_present (tk: token) return boolean;
  function token_acked (tk: token) return boolean;
  function token_released (tk: token) return boolean;
  function token_removed (tk: token) return boolean;
  --handshake functions
  procedure place_token (signal tk: out token; constant ntk: token;
                         constant delay: time:=0 ns; constant st: handshake:=present);
end performance_modeling;
```

Finally, once all of the information necessary to do performance modeling is defined (types, functions, procedures), it should be encapsulated into a package that can be made visible to any performance modeling component that needs it.



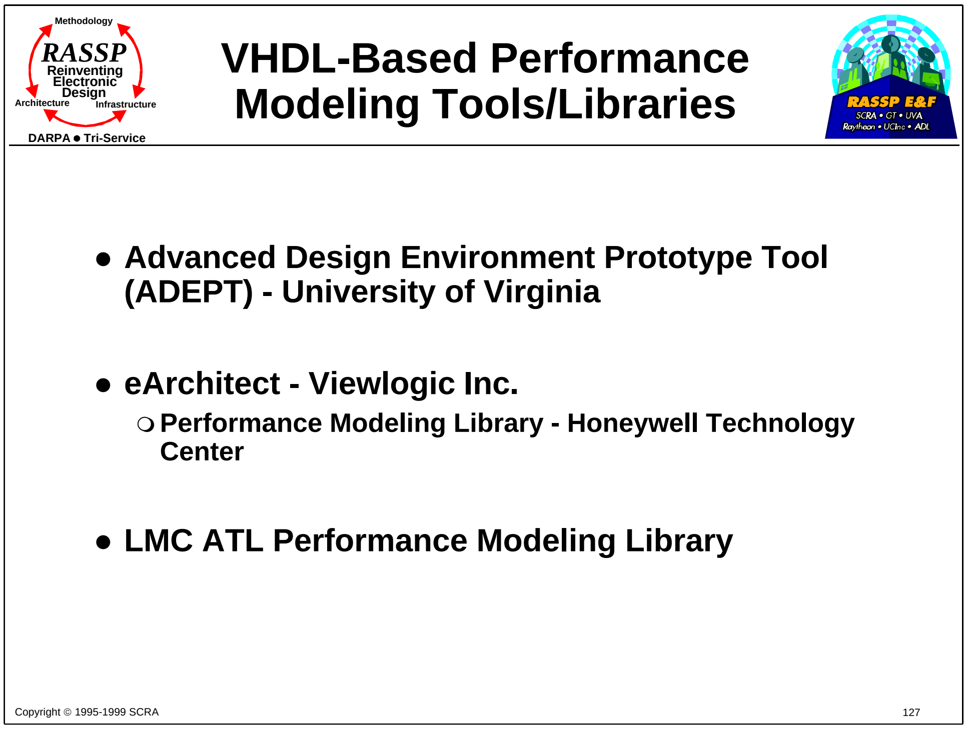
Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Module Outline

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Performance Modeling Introduction
- Performance Modeling Theory
- Non VHDL-Based Performance Modeling Tools
- Techniques for Performance Modeling using VHDL
- **VHDL-Based Performance Modeling Tools**
  - ❑ ADEPT
  - ❑ Viewlogic eArchitect
    - ⇨Honeywell PML
  - ❑ LMC ATL Performance Modeling Library
- VHDL Performance Modeling Examples
- Mixed Level Modeling
- Module Summary

Module Outline



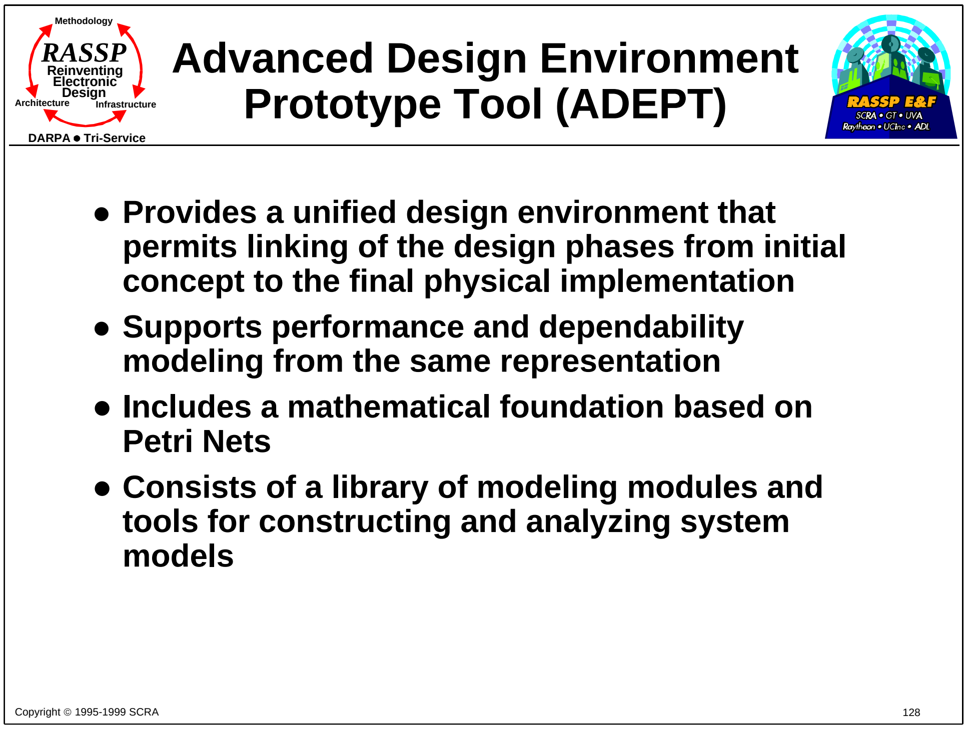
VHDL-Based Performance Modeling Tools/Libraries

- Advanced Design Environment Prototype Tool (ADEPT) - University of Virginia
- eArchitect - Viewlogic Inc.
  - Performance Modeling Library - Honeywell Technology Center
- LMC ATL Performance Modeling Library

UVa's ADEPT system is a set of library elements and a set of tools for constructing VHDL performance models.

Viewlogic's eArchitect product is a set of tools for constructing and analyzing the results of, VHDL performance models. It includes a performance modeling library based on the Performance Modeling Library developed by Honeywell Technology Center.

The Lockheed Martin, Advanced Technology Laboratory has developed a small library of VHDL performance modeling elements, specifically targeted at modeling Mercury Race Multicomputers, and a few tools for analyzing their results.


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

Advanced Design Environment Prototype Tool (ADEPT)

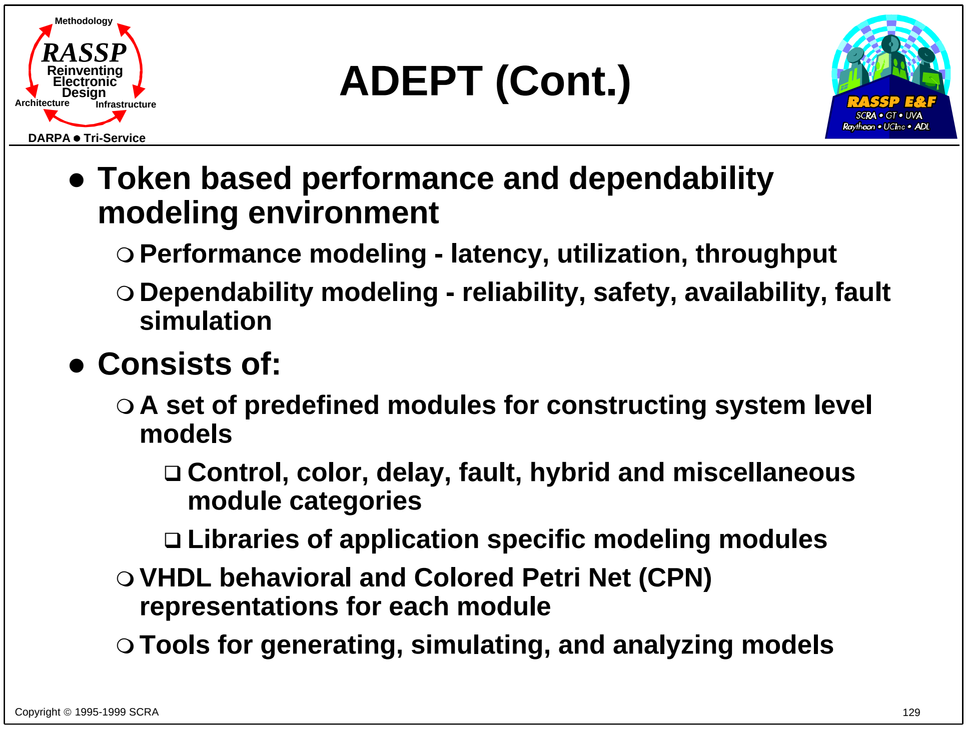
- **Provides a unified design environment that permits linking of the design phases from initial concept to the final physical implementation**
- **Supports performance and dependability modeling from the same representation**
- **Includes a mathematical foundation based on Petri Nets**
- **Consists of a library of modeling modules and tools for constructing and analyzing system models**

The Advanced Prototype Design Environment from UVa is a general VHDL-based uninterpreted modeling environment that also includes a Petri Net foundation (as will be explained). It consists of a library of modules for constructing system-level performance and Dependability models, and a set of tools for constructing and analyzing those models.

More information, including complete documentation and source code for ADEPT can be found on the UVa Center For Semicustom Integrated Systems web page:

http://csis.ee.virginia/

under the Publications and Tools sections. This includes some more detailed examples of performance, dependability, and mixed level modeling using ADEPT.


Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

ADEPT (Cont.)

- Token based performance and dependability modeling environment
  - ❍ Performance modeling - latency, utilization, throughput
  - ❍ Dependability modeling - reliability, safety, availability, fault simulation
- Consists of:
  - ❍ A set of predefined modules for constructing system level models
    - ❑ Control, color, delay, fault, hybrid and miscellaneous module categories
    - ❑ Libraries of application specific modeling modules
  - ❍ VHDL behavioral and Colored Petri Net (CPN) representations for each module
  - ❍ Tools for generating, simulating, and analyzing models

ADEPT’s strengths consist of:

• the inclusion of a mathematical foundation which makes analytical analysis of ADEPT models possible,

• the capability to perform performance and reliability modeling from the same ADEPT model without modification,

• the inclusion of a library of elements with which interfaces to behavioral models can be easily constructed for mixed level modeling, and

• the ability of the user to easily extend the ADEPT libraries.

ADEPT’s weaknesses include:

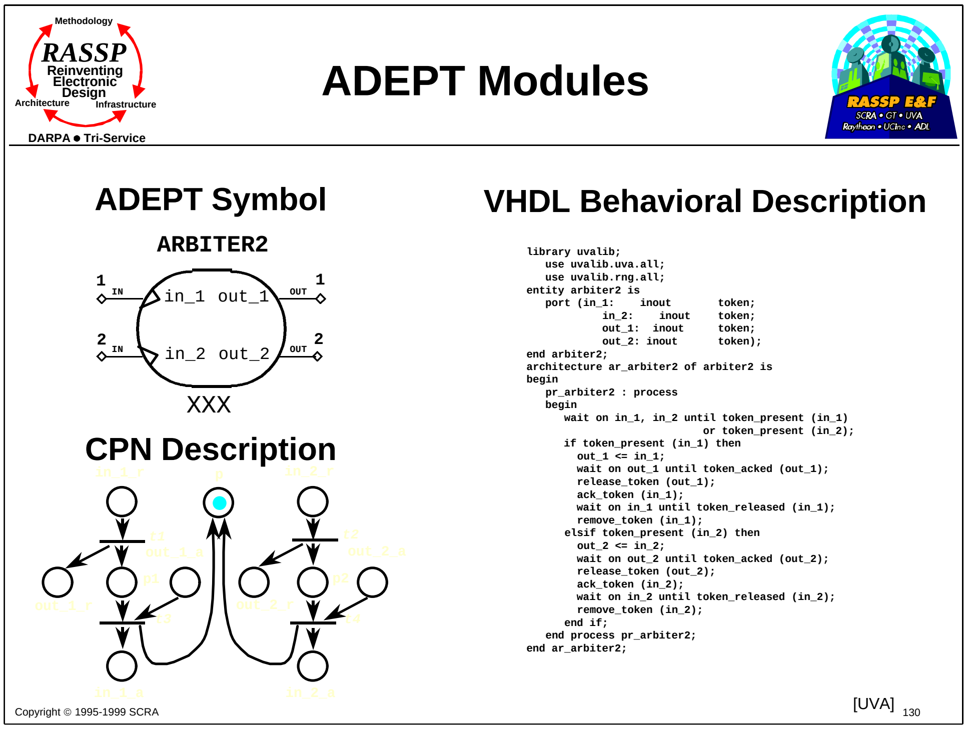
• the fact that the low level nature of the ADEPT modules sometimes makes model construction difficult and time consuming[1], and

• the fact that because its VHDL based, simulation of ADEPT models can take a long time[2].

Notes:

1) This is being alleviated somewhat by the addition of libraries of more complex modules, although these modules often lack the Petri Net representation.

2) This is being addressed by an effort to simplify and speedup the simulation of ADEPT models.



ADEPT Modules

ADEPT Symbol

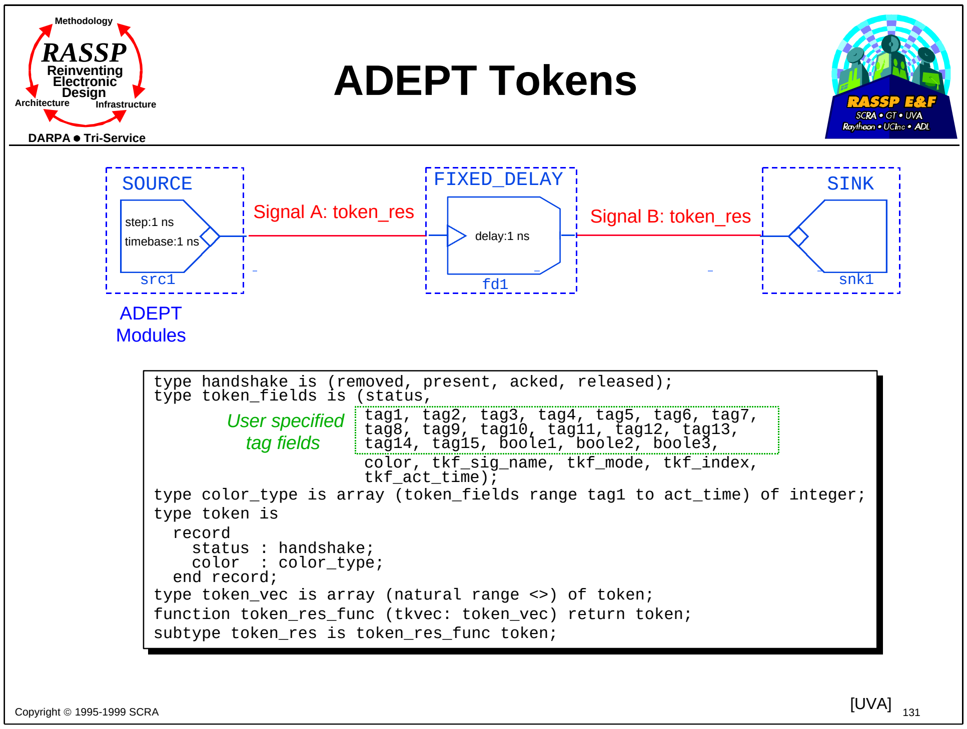
CPN Description

VHDL Behavioral Description

```
library uvalib;
   use uvalib.uva.all;
   use uvalib.rng.all;
entity arbiter2 is
   port (in_1:    inout       token;
            in_2:    inout    token;
            out_1:  inout     token;
            out_2: inout      token);
end arbiter2;
architecture ar_arbiter2 of arbiter2 is
begin
   pr_arbiter2 : process
   begin
      wait on in_1, in_2 until token_present (in_1)
                             or token_present (in_2);
      if token_present (in_1) then
        out_1 <= in_1;
        wait on out_1 until token_acked (out_1);
        release_token (out_1);
        ack_token (in_1);
        wait on in_1 until token_released (in_1);
        remove_token (in_1);
      elsif token_present (in_2) then
        out_2 <= in_2;
        wait on out_2 until token_acked (out_2);
        release_token (out_2);
        ack_token (in_2);
        wait on in_2 until token_released (in_2);
        remove_token (in_2);
      end if;
   end process pr_arbiter2;
end ar_arbiter2;
```

[UVA] 130

This figure shows the ADEPT symbol for an arbiter module - a module that serializes two tokens that arrive simultaneously on its inputs - its corresponding VHDL behavioral description, and its corresponding Colored Petri Net description. All of the ADEPT modules have a symbol and VHDL behavioral description that can be used for simulation. The ADEPT primitive modules - those in the Control, Color, Delay, Fault, Miscellaneous, and Hybrid categories - have colored Petri Net descriptions.



ADEPT Tokens

```
type handshake is (removed, present, acked, released);
type token_fields is (status,
                      tag1, tag2, tag3, tag4, tag5, tag6, tag7,
                      tag8, tag9, tag10, tag11, tag12, tag13,
                      tag14, tag15, boole1, boole2, boole3,
                      color, tkf_sig_name, tkf_mode, tkf_index,
                      tkf_act_time);
type color_type is array (token_fields range tag1 to act_time) of integer;
type token is
  record
    status : handshake;
    color  : color_type;
  end record;
type token_vec is array (natural range <>) of token;
function token_res_func (tkvec: token_vec) return token;
subtype token_res is token_res_func token;
```

User specified tag fields

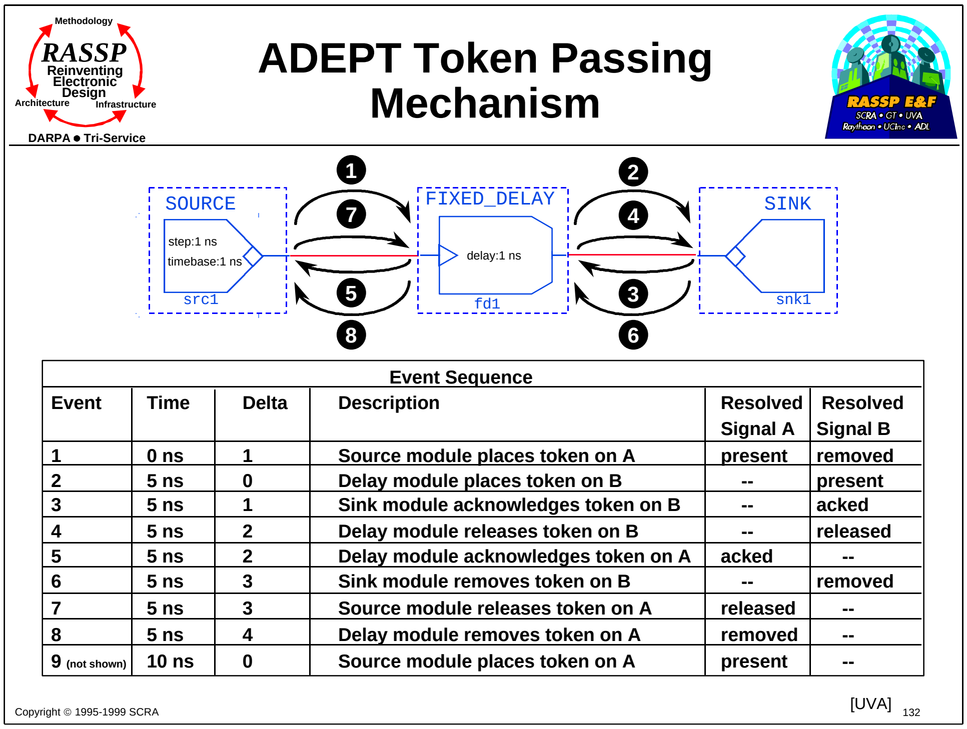
Copyright © 1995-1999 SCRA

[UVA] 131

ADEPT modules are connected via VHDL signals. These signals carry the tokens between the modules. The ADEPT tokens are implemented in VHDL as a record structure with two fields, a status field that is used to implement the 4 state handshaking, and a color field which is an array of integers used to hold user-defined information.

A VHDL bus resolution function, called token_res_function, is used to implement the point-to-point token passing mechanism as described earlier.

The point-to-point token mechanism uses a 4 state, fully-interlocked protocol. The states (enumerated in the handshake type) are “present,” “ack(nowledg)ed,” “released,” and “removed.”



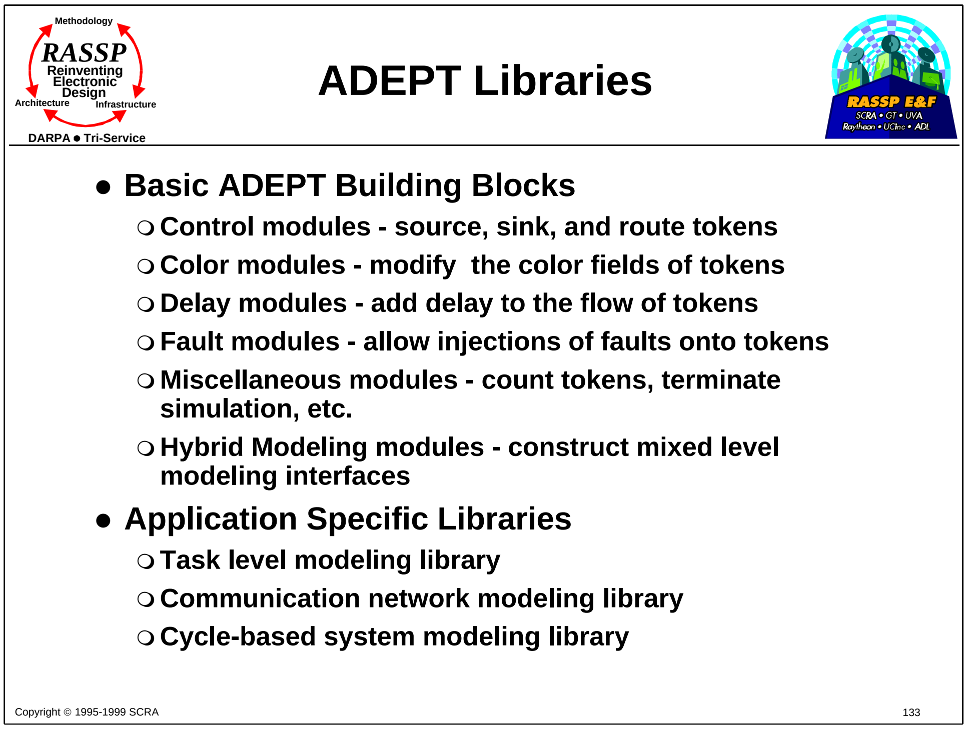
ADEPT Token Passing Mechanism

Event Sequence

Event	Time	Delta	Description	Resolved Signal A	Resolved Signal B
1	0 ns	1	Source module places token on A	present	removed
2	5 ns	0	Delay module places token on B	--	present
3	5 ns	1	Sink module acknowledges token on B	--	acked
4	5 ns	2	Delay module releases token on B	--	released
5	5 ns	2	Delay module acknowledges token on A	acked	--
6	5 ns	3	Sink module removes token on B	--	removed
7	5 ns	3	Source module releases token on A	released	--
8	5 ns	4	Delay module removes token on A	removed	--
9 (not shown)	10 ns	0	Source module places token on A	present	--

[UVA] 132

This is a detailed description of the ADEPT token passing protocol using a simple source/delay/sink model. Note that the only time that actually passes in the model is that taken up by the delay module - the token handshaking takes place in VHDL delta cycles with no time delay. In general, only delay module in ADEPT have actual time delays associated with them. All other modules use only delta delay. This fact can sometimes cause problems (delta cycle races) in constructing an ADEPT model.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

ADEPT Libraries

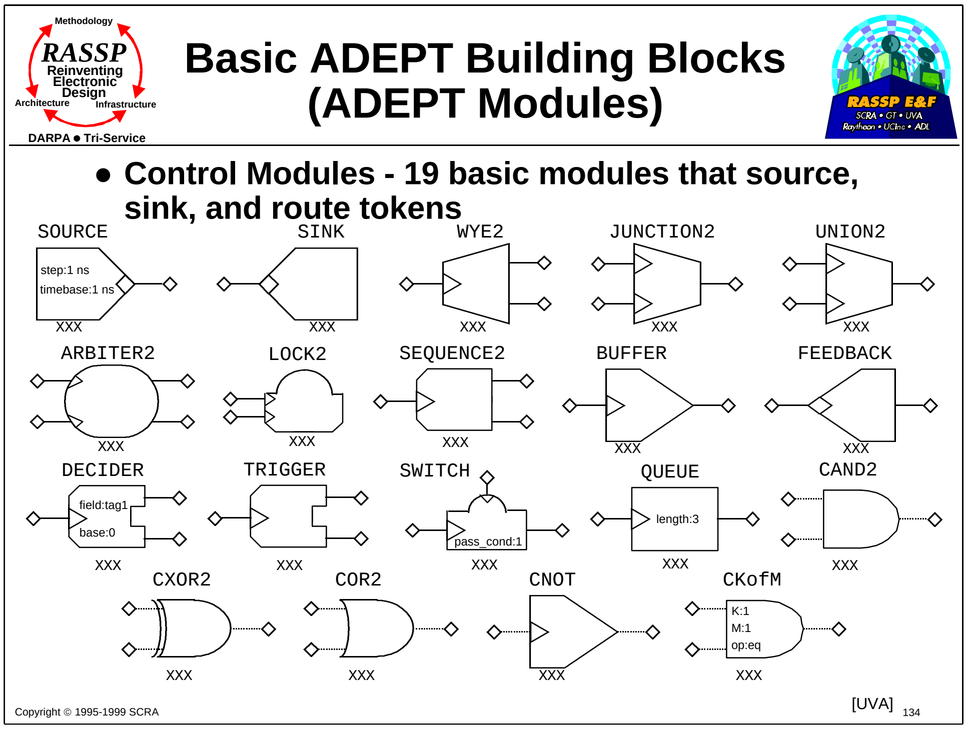
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Basic ADEPT Building Blocks**
  - ❍ **Control modules - source, sink, and route tokens**
  - ❍ **Color modules - modify the color fields of tokens**
  - ❍ **Delay modules - add delay to the flow of tokens**
  - ❍ **Fault modules - allow injections of faults onto tokens**
  - ❍ **Miscellaneous modules - count tokens, terminate simulation, etc.**
  - ❍ **Hybrid Modeling modules - construct mixed level modeling interfaces**
- **Application Specific Libraries**
  - ❍ **Task level modeling library**
  - ❍ **Communication network modeling library**
  - ❍ **Cycle-based system modeling library**

There are six categories of basic ADEPT building blocks out of which general system models can be constructed. As stated previously, these module have both a VHDL behavioral description and the Colored Petri Net description.

Because of the difficulty with which users have been constructing complex models out of the basic building blocks, libraries of more complex constructs and modeling modules have been developed. The elements in these libraries, which are targeted towards modeling systems in certain application areas, have only the VHDL behavioral description for simulation.

See [ADEPT_LR96] for more details on all of the ADEPT modules.



Basic ADEPT Building Blocks (ADEPT Modules)

- Control Modules - 19 basic modules that source, sink, and route tokens

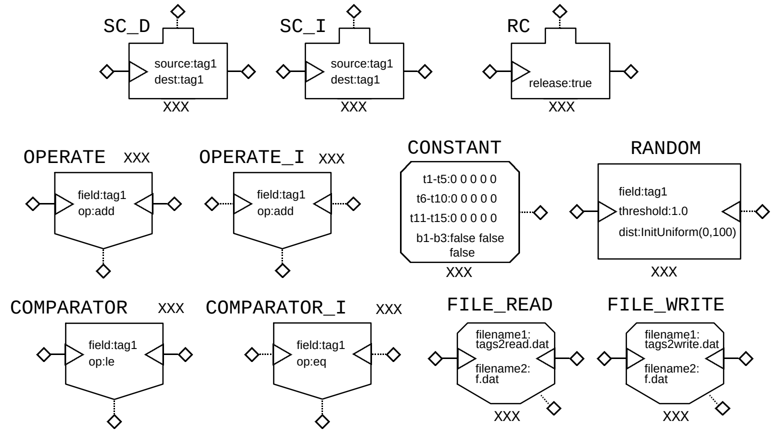
[UVA] 134

There are 19 modules in the Control category. These modules include the source and sink module for creating and destroying tokens, the wye, junction and union modules for fanning in and fanning out tokens, the buffer and feedback modules for buffering parts of the a system model from others, queue modules, for storing tokens, and other modules for routing tokens within a model.

There are also the “C” modules, like the CNOT and CXOR, that manipulate so called “control,” or independent tokens. In ADEPT, the tokens that are passed between modules using the 4 state interlocked protocol, are called “data” or dependent tokens. Independent or “control” tokens are tokens which have one source, but no real sinks. Then can take on only two of the 4 states in the protocol, present and released. They are generally used to carry routing and control information. For example, the output from the queue module which tells if the queue is full or not, and the inputs to the decider and switch module which determine if, and which output is active, are “control” tokens. See [ADEPT_UM96] for more details.

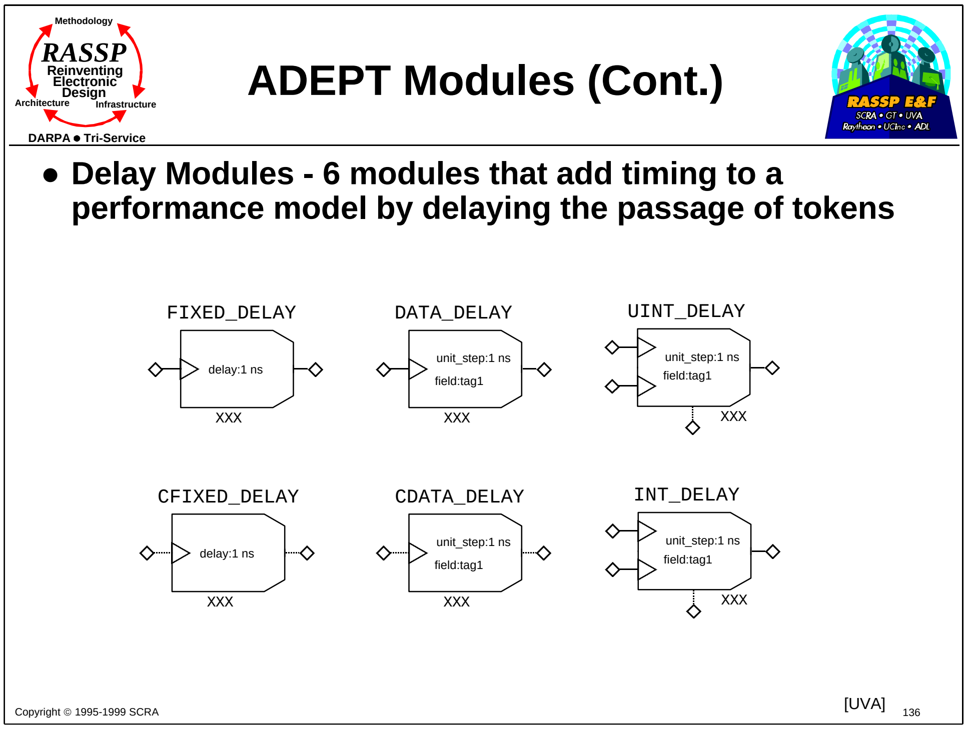
ADEPT Modules (Cont.)

- **Color Modules - 11 modules that manipulate (read, write, modify) the user-defined color fields of a token**



[UVA] 135

The color modules are used to access the user-defined (color fields) of the tokens. The set color (SC_D, SC_I) modules set values on tokens passing through them, and does the file_read module the read color (RC) module and the file_write module read color fields and write them onto other tokens or a file. The operator and comparator modules allow arithmetic and logical operations with token color fields, and the random module puts a random value on a color field.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

ADEPT Modules (Cont.)

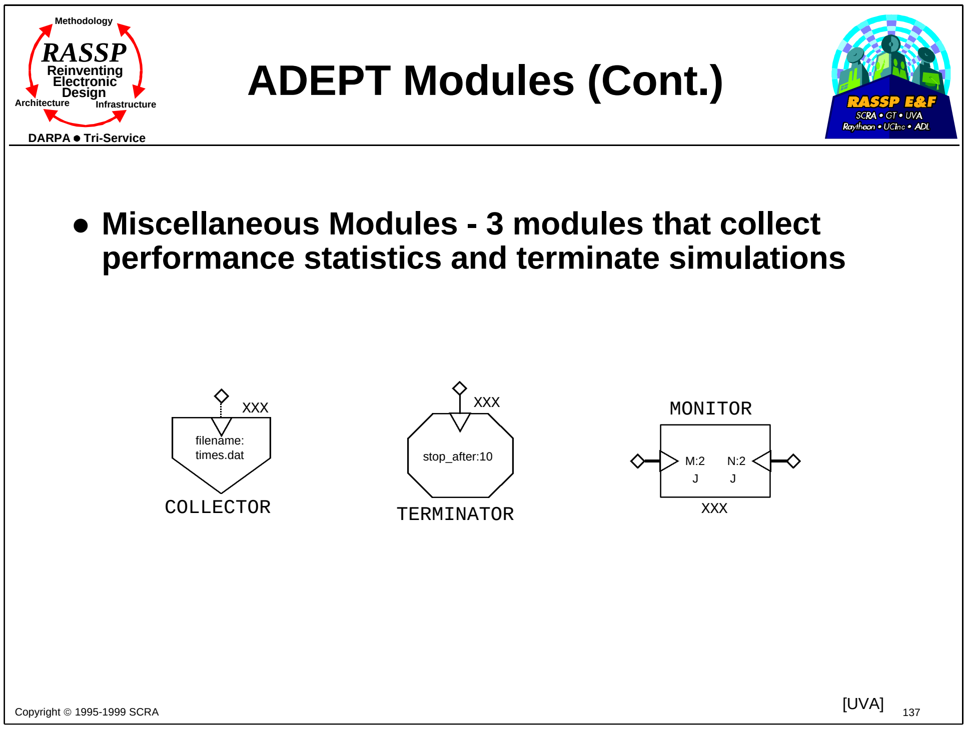
RASSP E&F SCRA • GT • UVA Raytheon • UCInc • ADL

- **Delay Modules - 6 modules that add timing to a performance model by delaying the passage of tokens**

Copyright © 1995-1999 SCRA

[UVA] 136

As stated previously, the delay modules are the only modules in the basic ADEPT set that have simulation time associated with them. There are fixed and data dependent delays for both “data” and “control” type tokens and more complex delay modules for modeling synchronization type events.

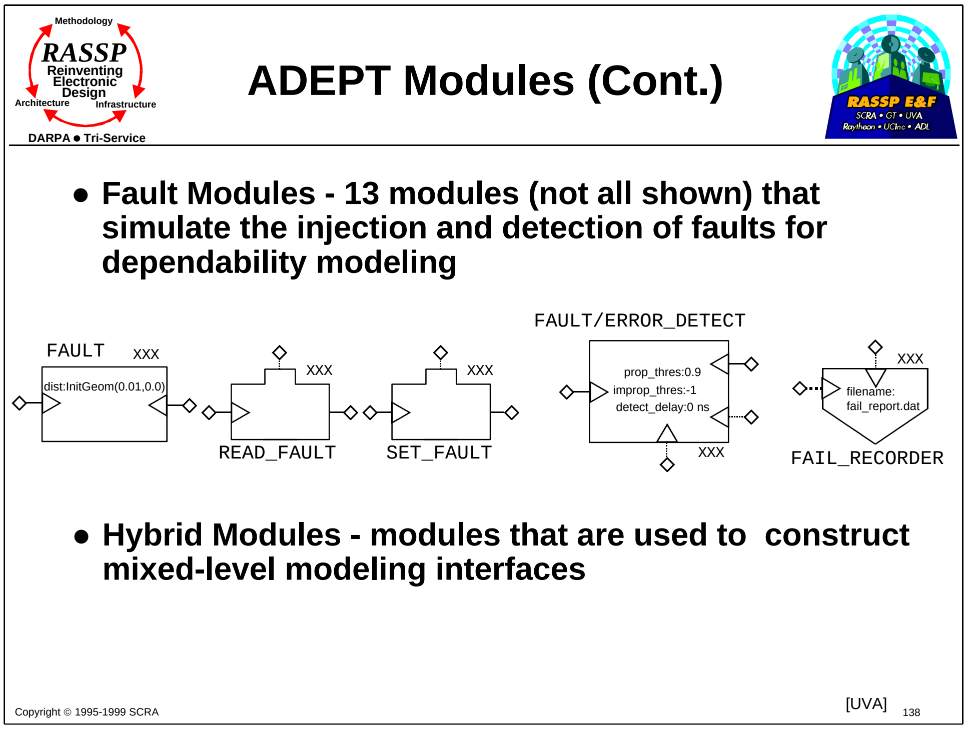


ADEPT Modules (Cont.)

- **Miscellaneous Modules - 3 modules that collect performance statistics and terminate simulations**

[UVA] 137

The miscellaneous module category includes the collector, which writes the time that a token passes a certain point in the model to a file, the terminator module, which can stop a simulation after a chosen number of tokens have gone past a specific point, and the monitor module, which writes latency and utilization data out to a file for post-processing by the ADEPT tools.



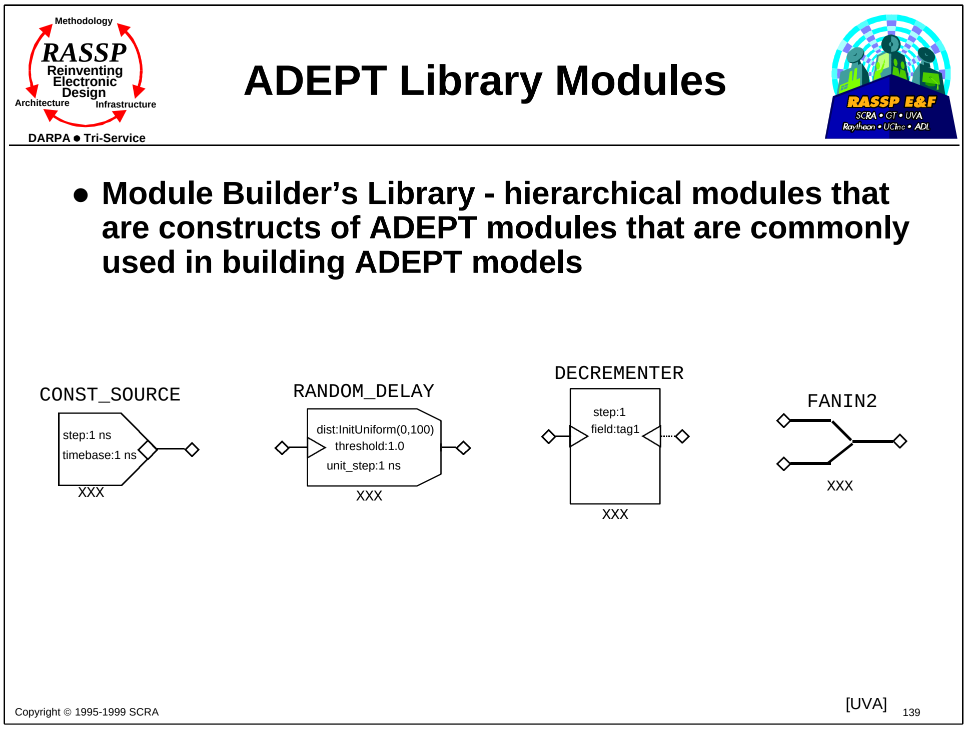
ADEPT Modules (Cont.)

- **Fault Modules - 13 modules (not all shown) that simulate the injection and detection of faults for dependability modeling**

- **Hybrid Modules - modules that are used to construct mixed-level modeling interfaces**

[UVA]

The fault modules allow the insertion and detection of faults into an ADEPT model for reliability analysis.

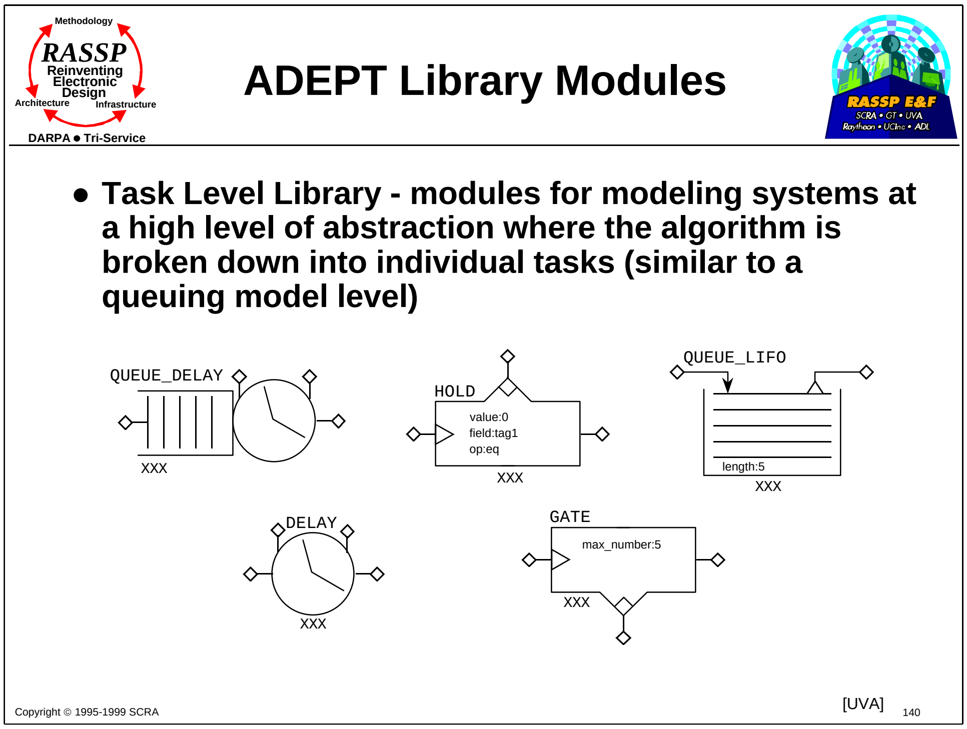


ADEPT Library Modules

- Module Builder’s Library - hierarchical modules that are constructs of ADEPT modules that are commonly used in building ADEPT models

[UVA] 139

The Module Builders Library is a library of constructs commonly used in constructing ADEPT models. For example, the random delay module delays a token according to a random number. It is a hierarchical module built up mainly from a Random module and a Data Delay module. The Decrementer module will decrement the value on a token tag by a set amount. It is built up from a Read Color, Operator, and Set Color module.



ADEPT Library Modules

- Task Level Library - modules for modeling systems at a high level of abstraction where the algorithm is broken down into individual tasks (similar to a queuing model level)

[UVA] 140

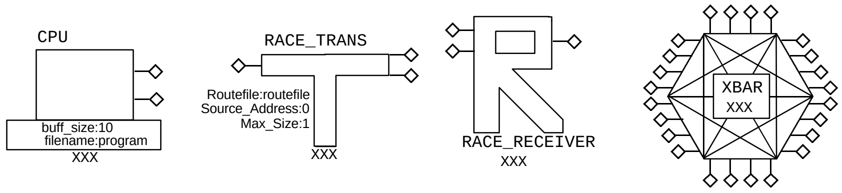
The Task Level Library is intended to allow users to build high level models of various application areas. The elements in this library consist of various Server module, various type of queue, like FIFO, LIFO, and Priority, and special routing modules like the gate and hold. The modules in this library were modeled, to some extent, on the types of modules available in the Extend tool from Imagine That Inc.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

ADEPT Library Modules

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

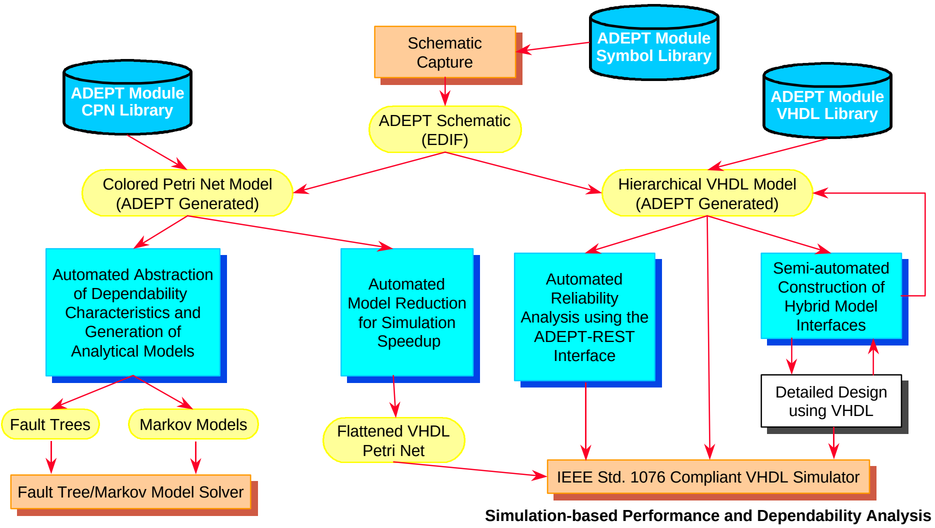
- **Multiprocessor Communications Network Modeling Library - modules for modeling systems at the processor/memory/switch level**
  - ❍ **Includes generic CPU plus models of ATM, SCI, Ethernet, Mercury Raceway, and Myrinet network components**
  - ❍ **Network models consist of routers and transmitters and receivers to interface CPUs to specific network routers**



[UVA] 141

The Multiprocessor Communication Network Modeling library was developed under the RASSP program to ease modeling of embedded multicomputer applications. It includes a generic CPU, much like the ATL CPU model to be discussed, and network modules to model Raceway, Myrinet, SCI, Ethernet, and ATM networks.

ADEPT Modeling Flows



Schematic Capture

ADEPT Module Symbol Library

ADEPT Module CPN Library

ADEPT Schematic (EDIF)

ADEPT Module VHDL Library

Colored Petri Net Model (ADEPT Generated)

Hierarchical VHDL Model (ADEPT Generated)

Automated Abstraction of Dependability Characteristics and Generation of Analytical Models

Automated Model Reduction for Simulation Speedup

Automated Reliability Analysis using the ADEPT-REST Interface

Semi-automated Construction of Hybrid Model Interfaces

Detailed Design using VHDL

Fault Trees

Markov Models

Flattened VHDL Petri Net

Fault Tree/Markov Model Solver

IEEE Std. 1076 Compliant VHDL Simulator

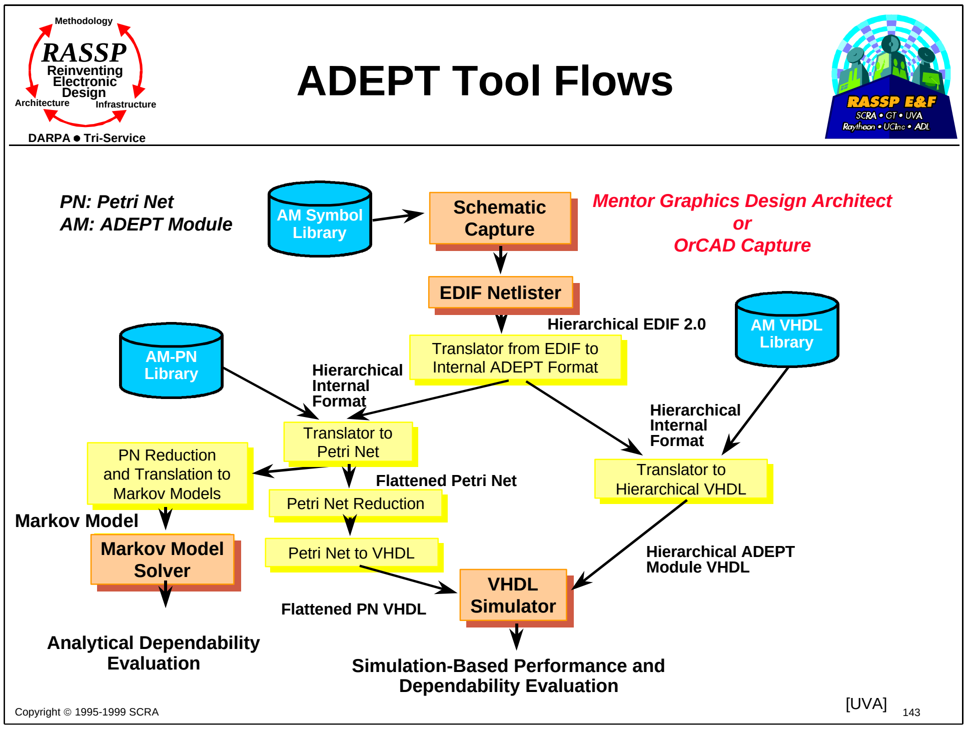
Simulation-based Performance and Dependability Analysis

Analytical Dependability Analysis

Copyright © 1995-1999 SCRA

[UVA] 142

This is a representation of the ADEPT modeling flows. Notice that there are two basic types of analysis, analytical (mainly for dependability modeling) and simulation-based (for both dependability and performance modeling). The boxes shown in blue are processes that are automated by tools developed for the ADEPT environment and the blue drums are ADEPT libraries of symbols, VHDL behavioral descriptions, and CPN descriptions.



ADEPT Tool Flows

PN: Petri Net
AM: ADEPT Module

AM Symbol Library

Schematic Capture

Mentor Graphics Design Architect
or
OrCAD Capture

EDIF Netlister

Hierarchical EDIF 2.0

AM VHDL Library

Translator from EDIF to Internal ADEPT Format

AM-PN Library

Hierarchical Internal Format

Translator to Petri Net

Hierarchical Internal Format

PN Reduction and Translation to Markov Models

Flattened Petri Net

Translator to Hierarchical VHDL

Petri Net Reduction

Markov Model

Markov Model Solver

Petri Net to VHDL

Hierarchical ADEPT Module VHDL

VHDL Simulator

Flattened PN VHDL

Analytical Dependability Evaluation

Simulation-Based Performance and Dependability Evaluation

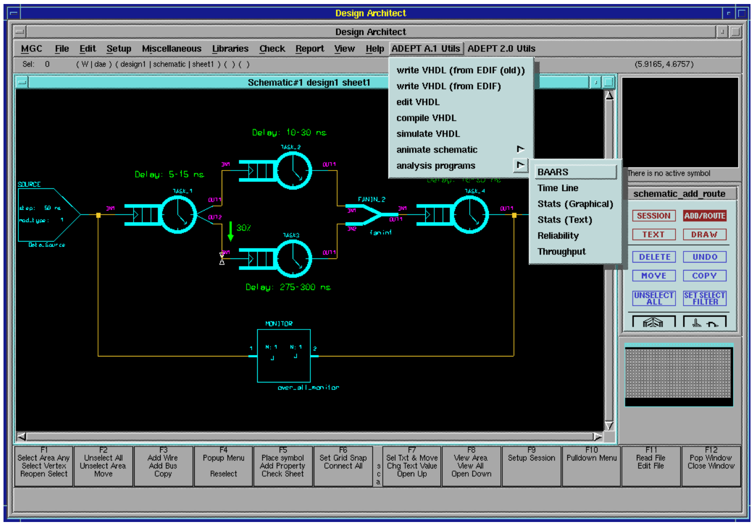
[UVA]

This slide shows how the actual ADEPT tools fit together with the various intermediate formats. Unfortunately, not all tools are available in all versions of ADEPT. Specifically, only the EDIF to structural VHDL path is supported on the PC platform with the OrCAD Capture tool.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

ADEPT Schematic Capture

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL



Copyright © 1995-1999 SCRA

[UVA] 144

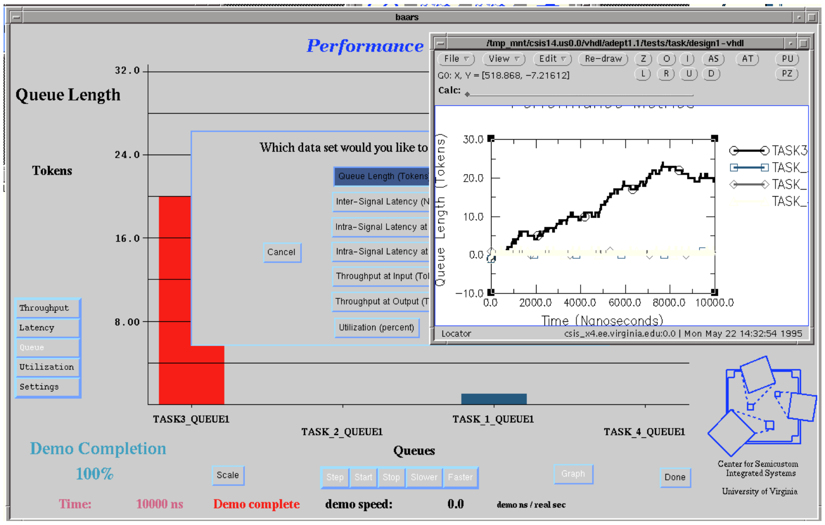
This screen shot shows the construction of an ADEPT schematic within Design Architect. Notice that all of the ADEPT utilities for constructing, simulation, and analyzing the results of an ADEPT model are available via pull-down menus.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

ADEPT Post Processing Tools

BAARS Dynamic Metric Display

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL



Copyright © 1995-1999 SCRA

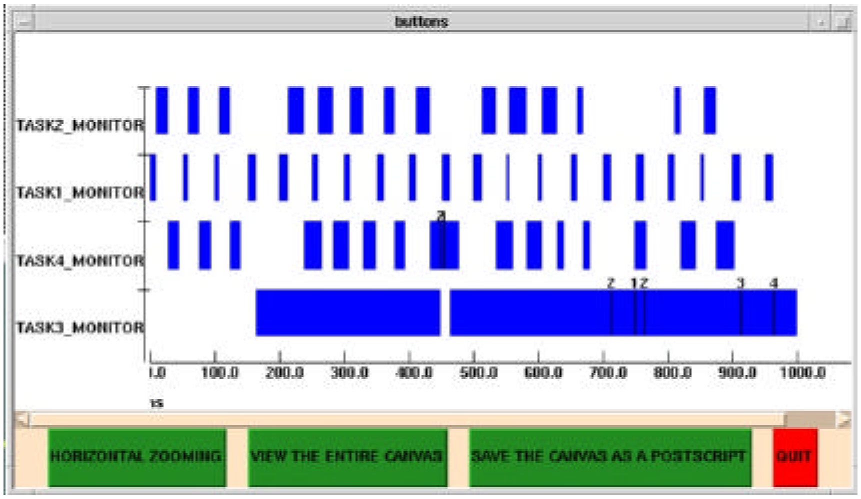
[UVA]

145

This is a screen shot of one of the available ADEPT post processing tools. This tool will give the user a dynamic playback of queue lengths, and module latency, utilization, and throughput over simulation time and then graph the results.

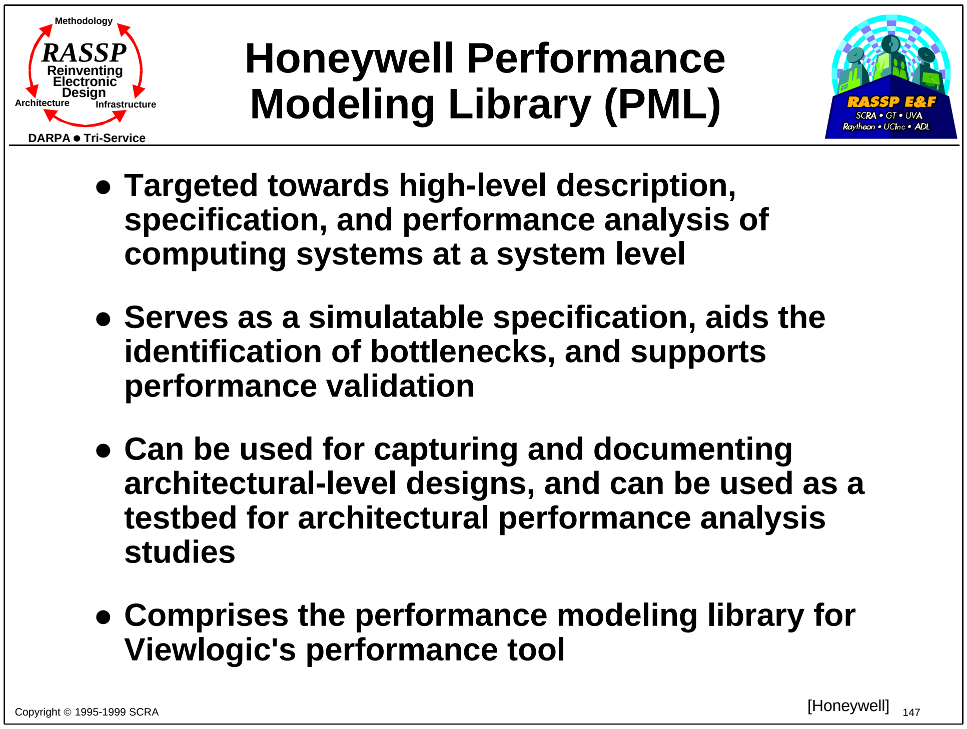
ADEPT Post Processing Tools (Cont.)

Timeline Utilization Display



[UVA]

This is a screen shot of another of the available ADEPT post processing tools. This tool presents utilization as a standard timeline display.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure DARPA ● Tri-Service

Honeywell Performance Modeling Library (PML)

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

- Targeted towards high-level description, specification, and performance analysis of computing systems at a system level
- Serves as a simulatable specification, aids the identification of bottlenecks, and supports performance validation
- Can be used for capturing and documenting architectural-level designs, and can be used as a testbed for architectural performance analysis studies
- Comprises the performance modeling library for Viewlogic's performance tool

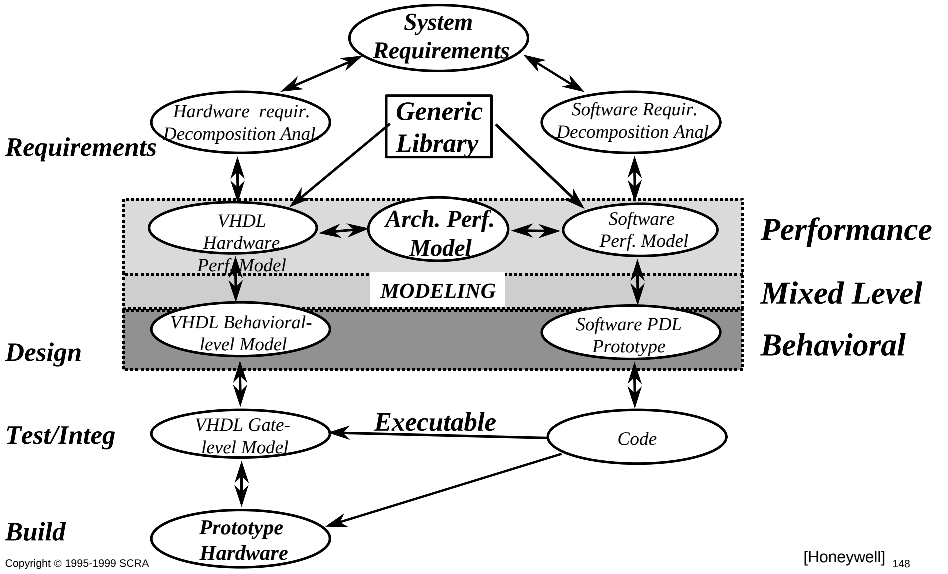
[Honeywell] 147

Now the Performance Modeling Library (PML) developed by Honeywell Technology Center in Minneapolis MN will be discussed. PML is a VHDL-based performance modeling library of elements targeted towards modeling a system at the processor-memory-switch level. It allows the modeling and simulation of the system’s hardware and software. PML is the basis of the Viewlogic eArchitect performance modeling tool.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

PML in the Design Process

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL



System Requirements

Hardware requir. Decomposition Anal

Generic Library

Software Requir. Decomposition Anal

Requirements

VHDL Hardware Perf. Model

Arch. Perf. Model

Software Perf. Model

Performance

MODELING

Mixed Level

VHDL Behavioral-level Model

Software PDL Prototype

Design

Behavioral

Test/Integ

VHDL Gate-level Model

Executable

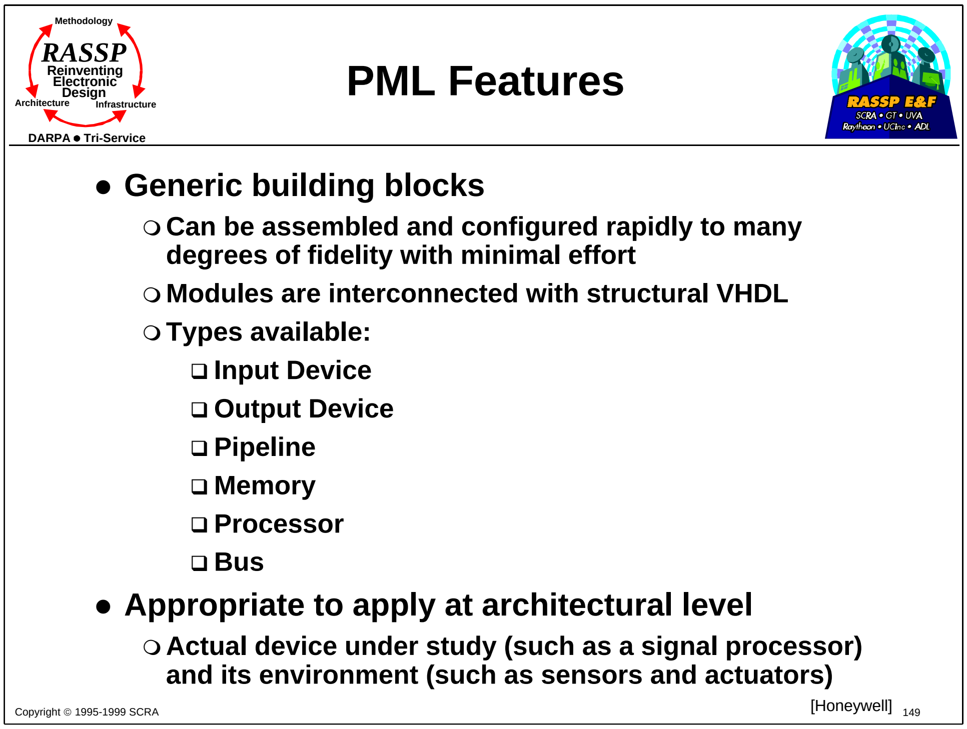
Code

Build

Prototype Hardware

[Honeywell] 148

This figure illustrates where the PML (and eArchitect) are intended to be used in the design process. Note that a capability for mixed level modeling (explained in the next section) is built into PML/eArchitect.

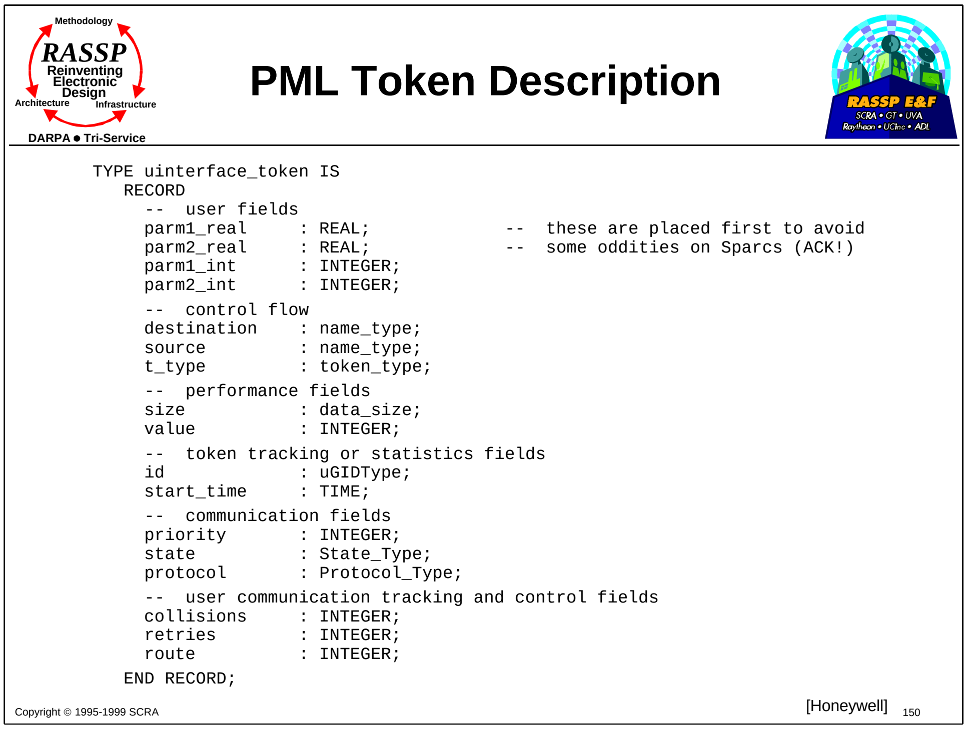


PML Features

- Generic building blocks
  - ❍ Can be assembled and configured rapidly to many degrees of fidelity with minimal effort
  - ❍ Modules are interconnected with structural VHDL
  - ❍ Types available:
    - ❑ Input Device
    - ❑ Output Device
    - ❑ Pipeline
    - ❑ Memory
    - ❑ Processor
    - ❑ Bus
- Appropriate to apply at architectural level
  - ❍ Actual device under study (such as a signal processor) and its environment (such as sensors and actuators)

[Honeywell]

The overall approach in PML was to develop a small library of generic building blocks with many generic inputs that allowed them to be parameterized to model many different devices. The library actually contains only 5 modules and several different bus resolution functions to model communications protocols. These devices are targeted at modeling the architectural (PMS) level.

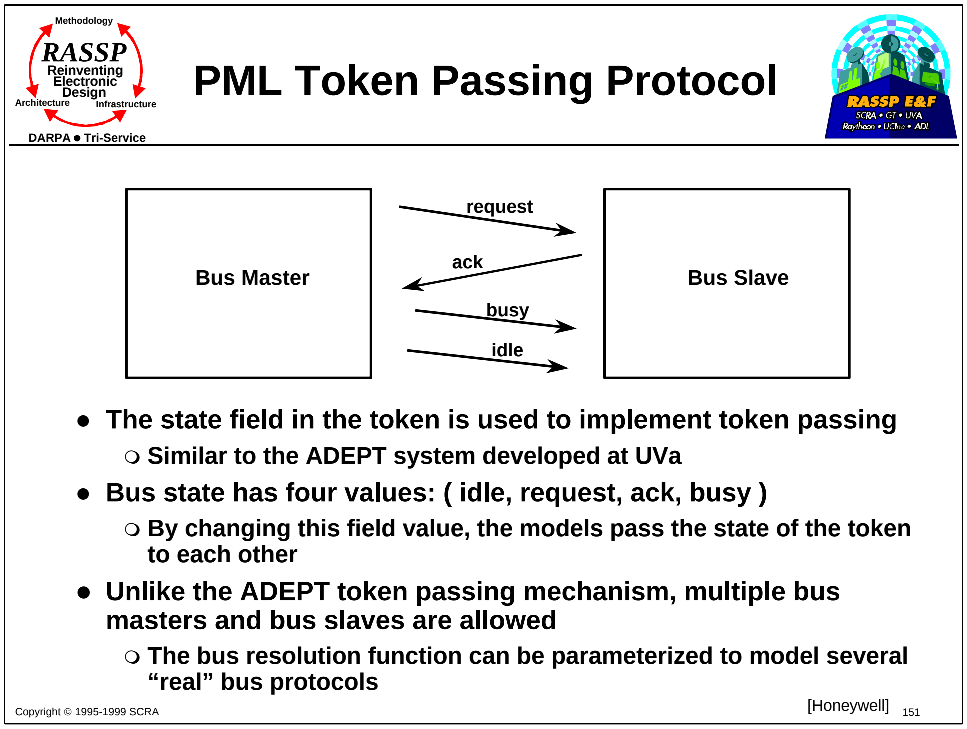


PML Token Description

```
TYPE uinterface_token IS
   RECORD
     --  user fields
     parm1_real     : REAL;             --  these are placed first to avoid
     parm2_real     : REAL;             --  some oddities on Sparcs (ACK!)
     parm1_int      : INTEGER;
     parm2_int      : INTEGER;
     --  control flow
     destination    : name_type;
     source         : name_type;
     t_type         : token_type;
     --  performance fields
     size           : data_size;
     value          : INTEGER;
     --  token tracking or statistics fields
     id             : uGIDType;
     start_time     : TIME;
     --  communication fields
     priority       : INTEGER;
     state          : State_Type;
     protocol       : Protocol_Type;
     --  user communication tracking and control fields
     collisions     : INTEGER;
     retries        : INTEGER;
     route          : INTEGER;
   END RECORD;
```

[Honeywell]

Here is a description of the generic token defined by Honeywell Technology Center for interoperability of performance models [HTC97]. The actual token used inside of PML is proprietary and slightly different than this, but this example gives the overall structure and how it is different from the ADEPT token.



PML Token Passing Protocol

Bus Master

request

ack

busy

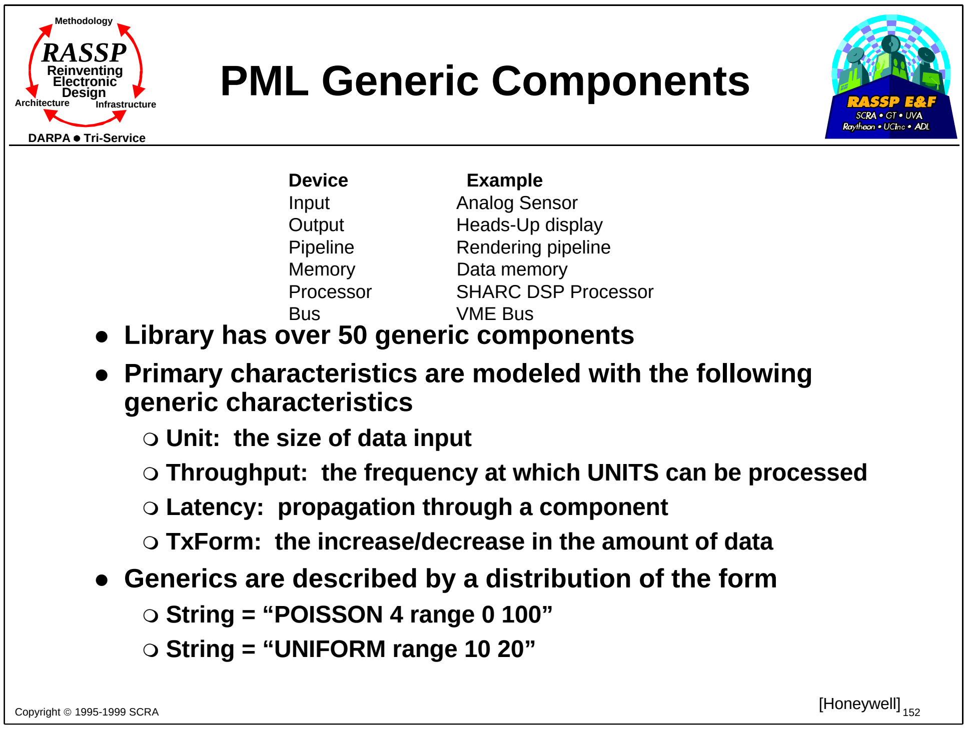
idle

Bus Slave

- The state field in the token is used to implement token passing
  - ❍ Similar to the ADEPT system developed at UVa
- Bus state has four values: (idle, request, ack, busy)
  - ❍ By changing this field value, the models pass the state of the token to each other
- Unlike the ADEPT token passing mechanism, multiple bus masters and bus slaves are allowed
  - ❍ The bus resolution function can be parameterized to model several “real” bus protocols

[Honeywell]

The VHDL bus resolution function (BRF) used in PML uses four states to pass tokens on busses that have multiple drives and sources. For simple point-to-point connections, only three states are used for simulation efficiency. The BRF can be parameterized (or modified) to model several “real” bus protocols - thus the VHDL BRF is actually part of the model.



PML Generic Components

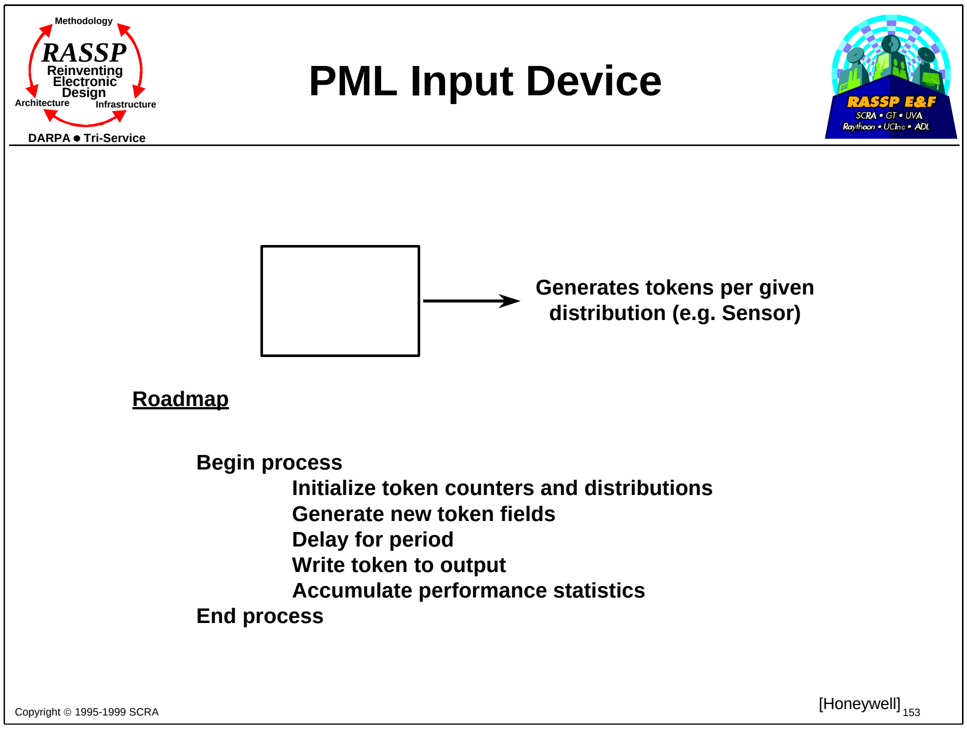
Device	Example
Input	Analog Sensor
Output	Heads-Up display
Pipeline	Rendering pipeline
Memory	Data memory
Processor	SHARC DSP Processor
Bus	VME Bus

- **Library has over 50 generic components**
- **Primary characteristics are modeled with the following generic characteristics**
  - **Unit: the size of data input**
  - **Throughput: the frequency at which UNITS can be processed**
  - **Latency: propagation through a component**
  - **TxForm: the increase/decrease in the amount of data**
- **Generics are described by a distribution of the form**
  - **String = “POISSON 4 range 0 100”**
  - **String = “UNIFORM range 10 20”**

[Honeywell] 152

As stated previously, the PML library consists of 5 major modules, but there are many examples of modules parameterized to model specific devices in the library.

PML contains a sophisticated string processing language for specification of complex generic parameters to the models.



PML Input Device

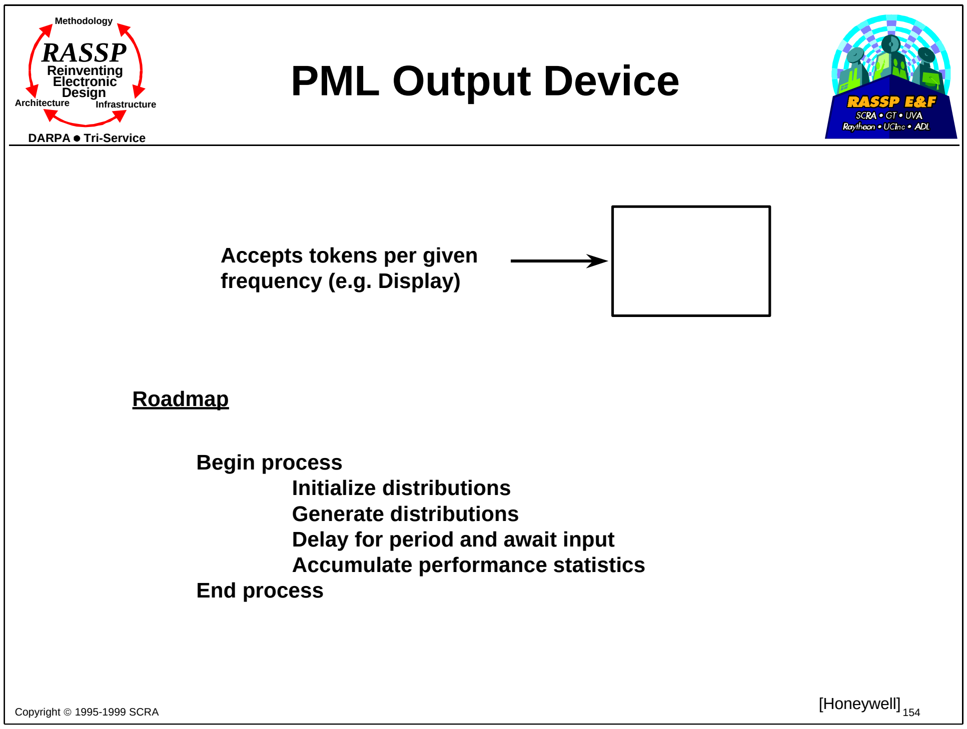
Generates tokens per given distribution (e.g. Sensor)

Roadmap

```
Begin process
        Initialize token counters and distributions
        Generate new token fields
        Delay for period
        Write token to output
        Accumulate performance statistics
End process
```

[Honeywell]

A PML input device is like a Source module in ADEPT, it creates tokens at a specified rate. Note that all modules in PML participate in the generation of performance statistics like latency and utilization.



Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

PML Output Device

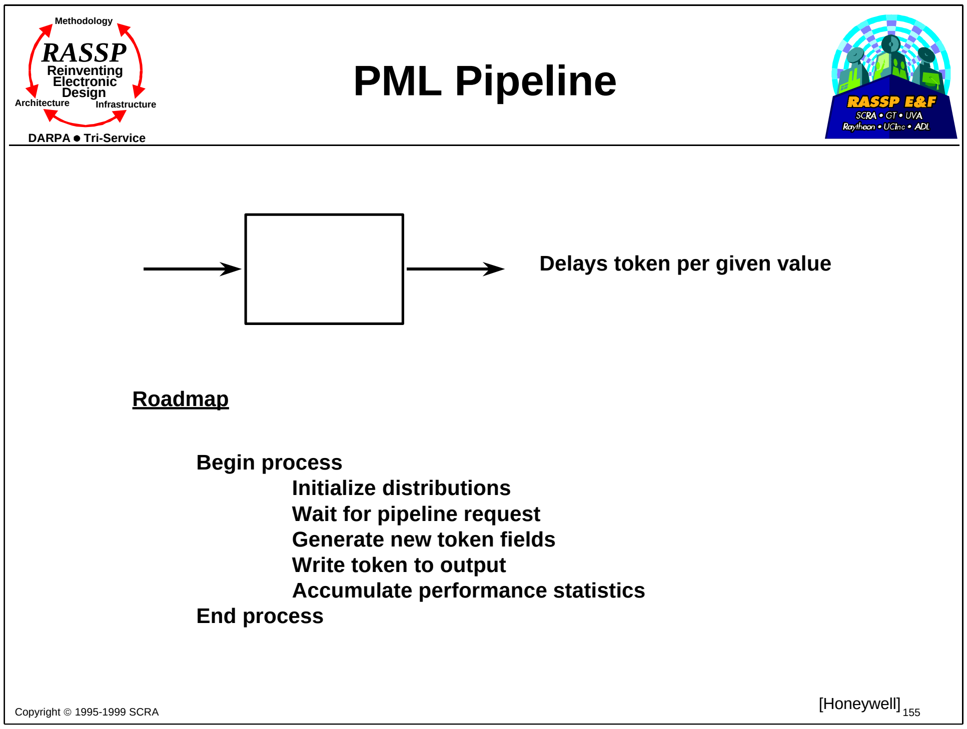
Accepts tokens per given frequency (e.g. Display)

Roadmap

```
Begin process
    Initialize distributions
    Generate distributions
    Delay for period and await input
    Accumulate performance statistics
End process
```

[Honeywell] 154

An output device is like a Sink module in ADEPT. It consumes tokens.



PML Pipeline

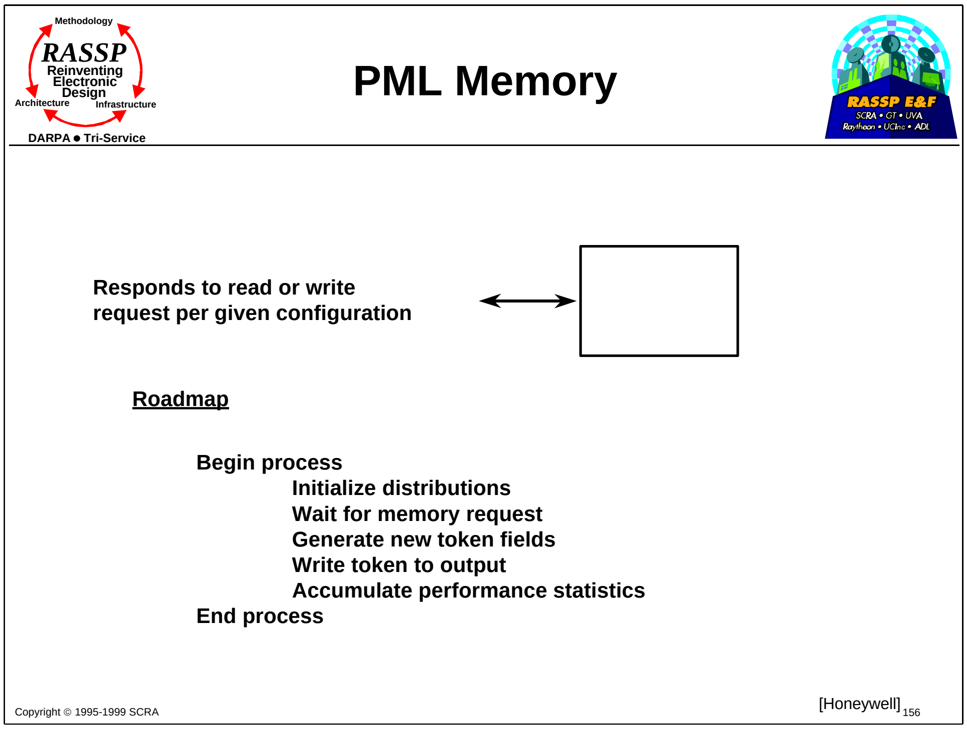
Delays token per given value

Roadmap

```
Begin process
        Initialize distributions
        Wait for pipeline request
        Generate new token fields
        Write token to output
        Accumulate performance statistics
End process
```

[Honeywell] 155

The pipeline component delays tokens. It can also, by changing token fields, route tokens.



PML Memory

Responds to read or write request per given configuration

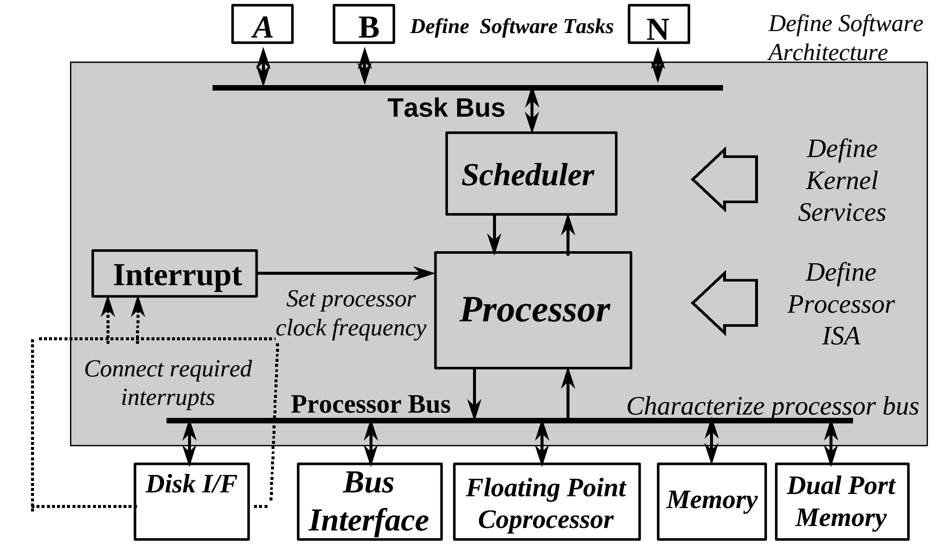
Roadmap

```
Begin process
        Initialize distributions
        Wait for memory request
        Generate new token fields
        Write token to output
        Accumulate performance statistics
End process
```

[Honeywell] 156

The memory component consumes memory request tokens and after a specified delay, generates memory access tokens.

PML Processor Model



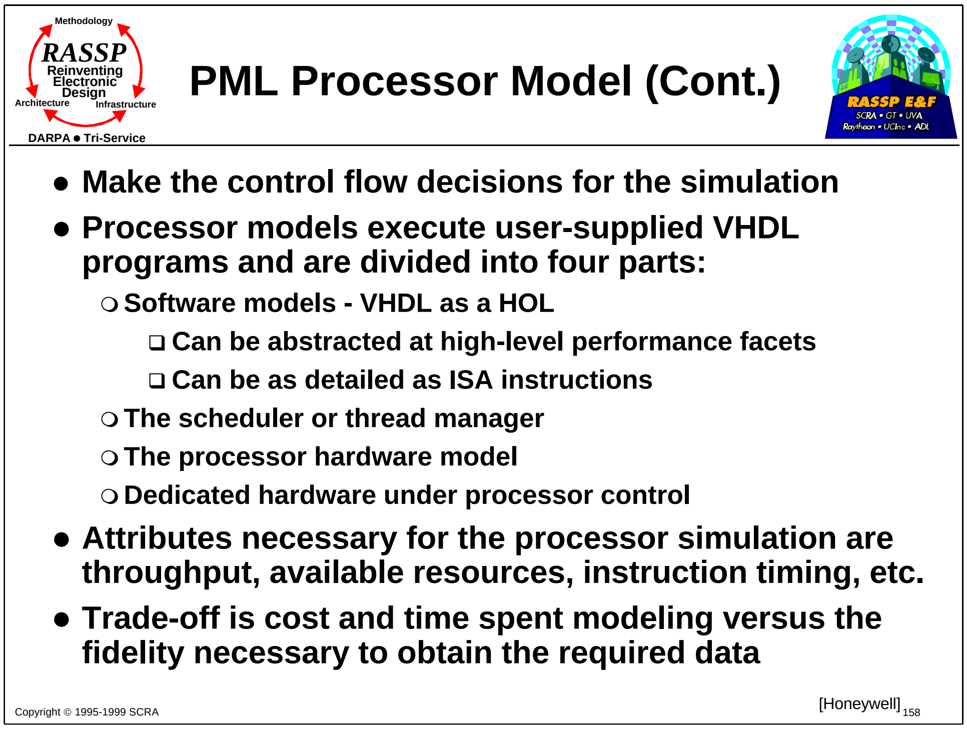
A
B
Define Software Tasks
N
Define Software Architecture
Task Bus
Scheduler
Define Kernel Services
Interrupt
Set processor clock frequency
Processor
Define Processor ISA
Connect required interrupts
Processor Bus
Characterize processor bus
Disk I/F
Bus Interface
Floating Point Coprocessor
Memory
Dual Port Memory

[Honeywell] 157

The processor model is the heart of the PML. It is capable of running a representation of the software that the real system will execute. That software representation, while written in VHDL can be at a level of abstraction that ranges from the task level down to the detailed functional level.

The PML processor is basically a request-resource model. The software representation executes and a specified point, requests resources (e.g. memory access, 1000 floating point multiplies, 100 integer adds, etc.) from the processor. The processor schedules these operations on the hardware resource when it is available and delays the software execution until they are completed. The software continues from that point until more hardware resources are needed.

The processor is parameterized by specifying its Instruction Set Architecture (ISA) and what and how many resources are consumed by each instruction in the ISA. Sophisticated operating system constructs such as interrupts and multitasking can be modeled as well.

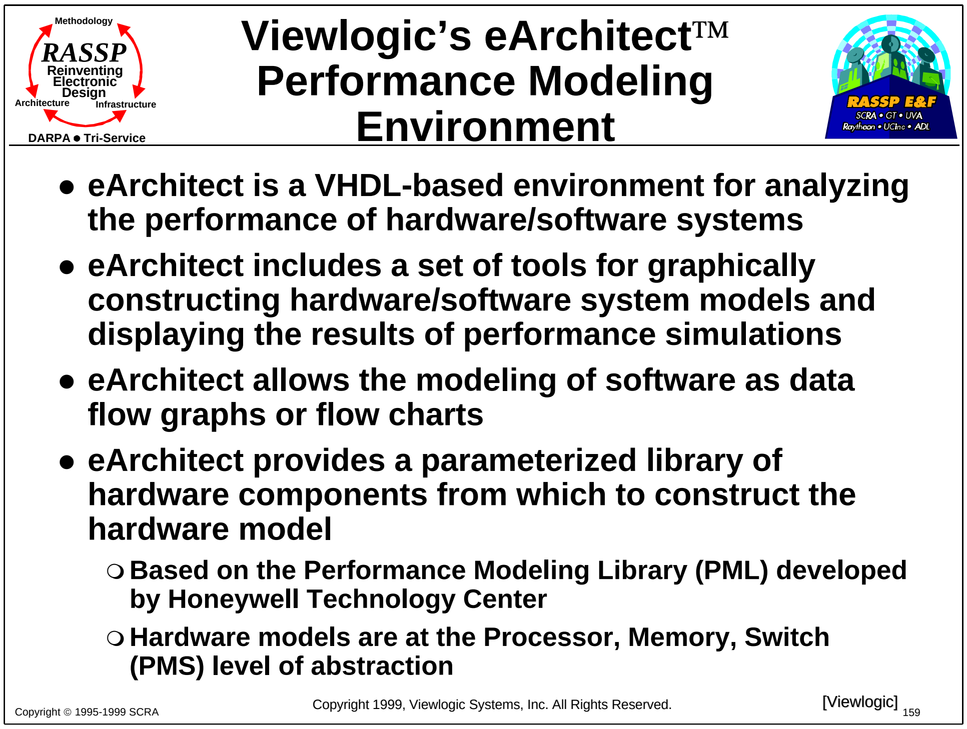


PML Processor Model (Cont.)

- Make the control flow decisions for the simulation
- Processor models execute user-supplied VHDL programs and are divided into four parts:
  - ❍ Software models - VHDL as a HOL
    - ❑ Can be abstracted at high-level performance facets
    - ❑ Can be as detailed as ISA instructions
  - ❍ The scheduler or thread manager
  - ❍ The processor hardware model
  - ❍ Dedicated hardware under processor control
- Attributes necessary for the processor simulation are throughput, available resources, instruction timing, etc.
- Trade-off is cost and time spent modeling versus the fidelity necessary to obtain the required data

[Honeywell] 158

The processor model allows detailed modeling of software at various levels of abstraction executing on different types and speeds of processors. One drawback of this fidelity (and its associated complexity) is long simulation times.



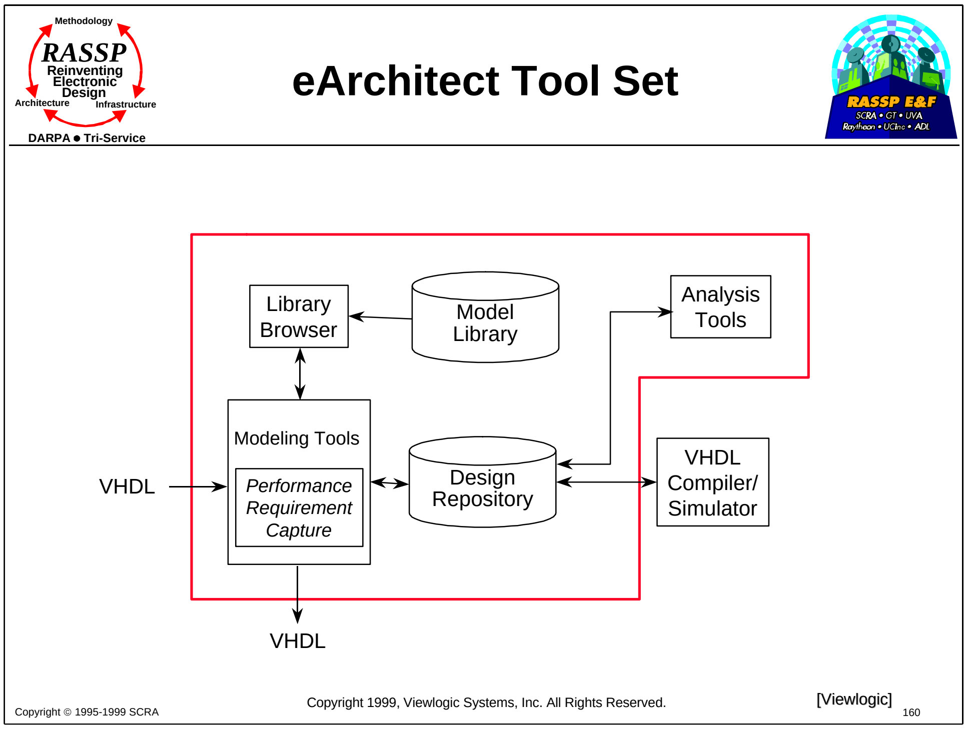
Viewlogic's eArchitect™ Performance Modeling Environment

- eArchitect is a VHDL-based environment for analyzing the performance of hardware/software systems
- eArchitect includes a set of tools for graphically constructing hardware/software system models and displaying the results of performance simulations
- eArchitect allows the modeling of software as data flow graphs or flow charts
- eArchitect provides a parameterized library of hardware components from which to construct the hardware model
  - ❍ Based on the Performance Modeling Library (PML) developed by Honeywell Technology Center
  - ❍ Hardware models are at the Processor, Memory, Switch (PMS) level of abstraction

[Viewlogic]

This section describes Viewlogic's eArchitect ™ tool. eArchitect is very ADEPT like in that it includes tools for constructing, simulating, and analyzing performance models in VHDL. It uses the Performance Modeling Library (PML) developed by Honeywell Technology Center as its module library. The development of eArchitect was funded as part of the RASSP program.

Note that unlike ADEPT, eArchitect (like PML) is targeted at one specific level of performance modeling (the processor, memory, switch (PMS) level) and does not have a mathematical foundation or support dependability analysis.



eArchitect Tool Set

Library Browser

Model Library

Analysis Tools

Modeling Tools

Performance Requirement Capture

Design Repository

VHDL Compiler/ Simulator

VHDL

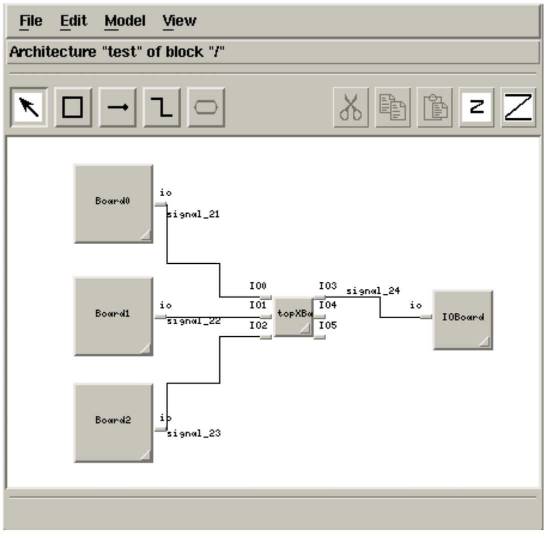
VHDL

Copyright 1999, Viewlogic Systems, Inc. All Rights Reserved.

[Viewlogic]

This is the eArchitect tool set. Like ADEPT, a commercial, third party, VHDL simulator is used as the simulation engine and must be obtained separately.

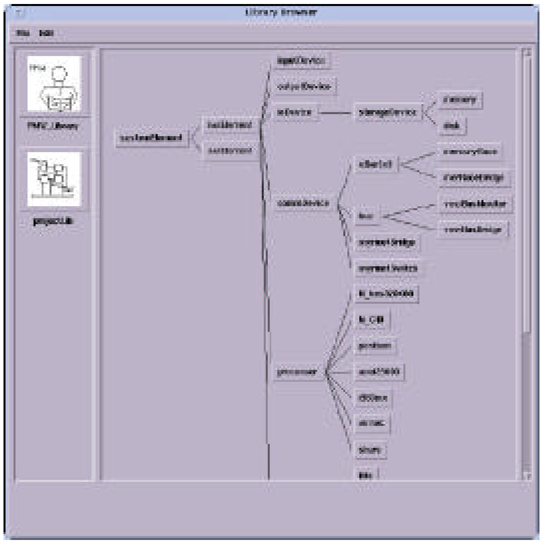
Hardware Design in eArchitect



[Viewlogic]

This is an illustration of the construction of the hardware model in eArchitect. The hardware model consists of processor models and communications switch models from the PML library (as will be presented). The modules used in the model can be parameterized, via the GUI, to model different types of processors and networks.

eArchitect Library Browser



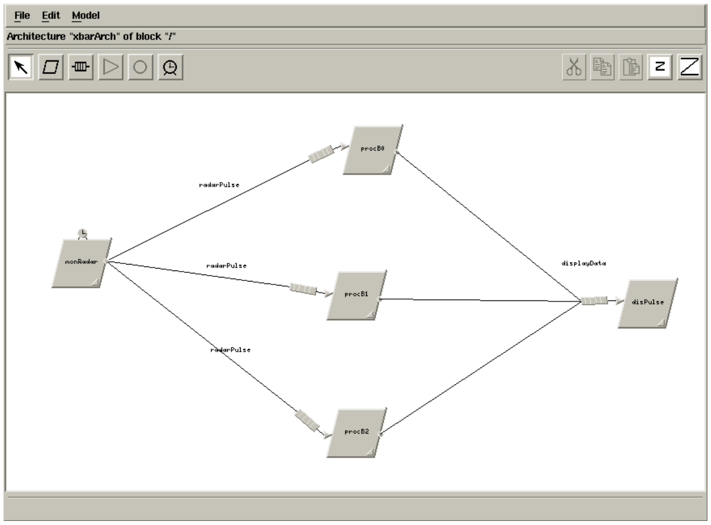
Copyright 1999, Viewlogic Systems, Inc. All Rights Reserved.

[Viewlogic]

This is the eArchitect library browser. It is used to select standard hardware components out of the library for instantiation into a performance model. eArchitect comes with the complete PML library of generic elements and several specific components (like a Mercury RaceWay crossbar switch) built out of those generic components.

Software Design in eArchitect

Data Flow Graph

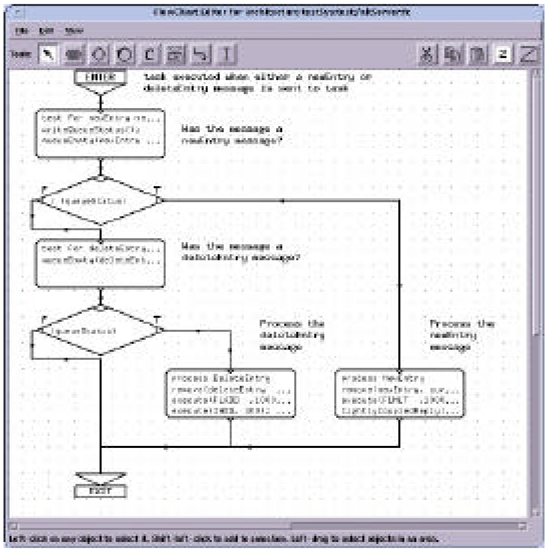


[Viewlogic] 163

This is an illustration of the description of the software application in eArchitect. Here, the software is described as a dataflow graph as is common in embedded DSP applications.

Software Design in eArchitect

Flow Chart


File Edit View
Tools
ENTER
task executed when either a newEntry or deleteEntry message is sent to task
Has the message a newEntry message?
Has the message a deleteEntry message?
Process the deleteEntry message
Process the newEntry message
process DeleteEntry
process NewEntry
EXIT

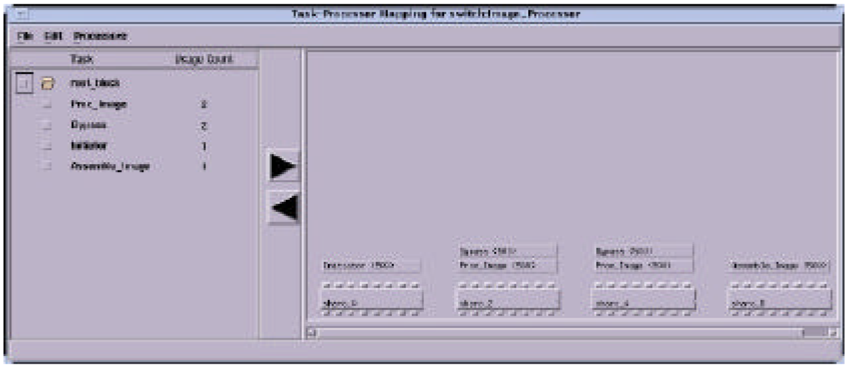
Copyright 1999, Viewlogic Systems, Inc. All Rights Reserved.

[Viewlogic] 164

Software can also be described as a control flow graph in eArchitect as shown here.

In addition to the two methods shown in this slide and the previous one, software in eArchitect can be coded directly in VHDL by the user (with appropriate calls to the hardware resource models), and included in the eArchitect model.

Software to Hardware Mapping in eArchitect



Task	Image Count
root_block	
Proc_Image	2
Bypass	2
Initiator	1
Assemble_Image	1

[Viewlogic]

Once the hardware and software models are completed, the next step is to map the software tasks onto specific hardware processors for execution. This is done with the software mapping tool as shown here.

Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

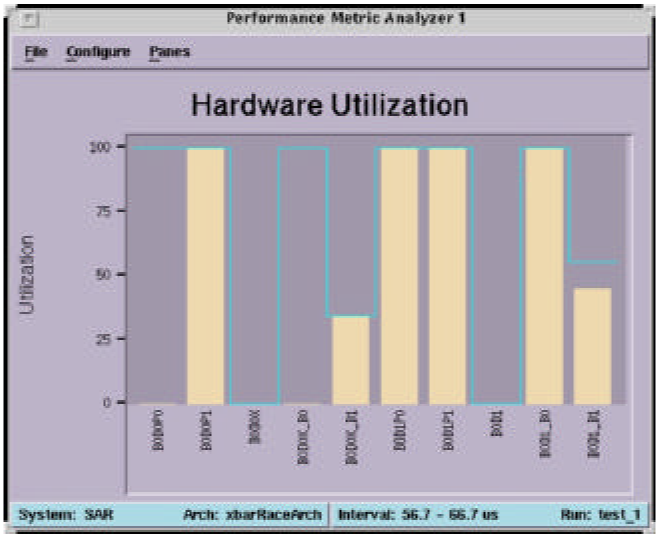
DARPA ● Tri-Service

Analysis of Results in eArchitect Utilization

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL



System: SAR Arch: xbarRaceArch Interval: 56.7 - 66.7 us Run: test_1

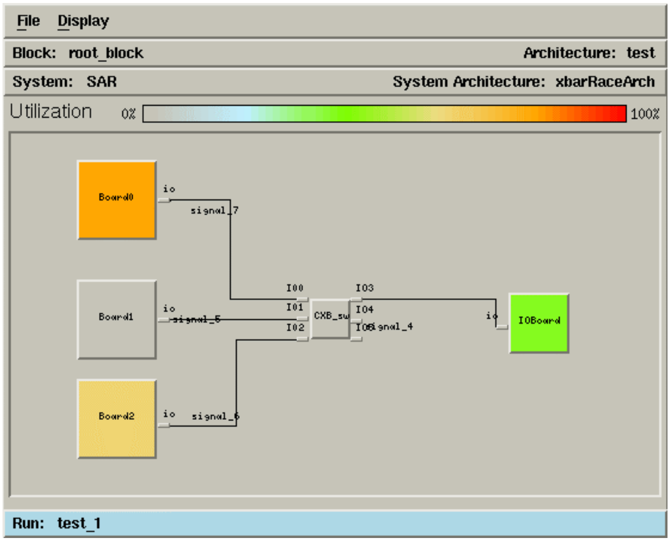
[Viewlogic] 166

Like ADEPT, eArchitect contains a number of tools for analyzing the data from the performance model simulation. This is the eArchitect utilization tool display. It displays specific processor utilization as a moving horizontal bar graph.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Analysis of Results in eArchitect
Utilization (Hot Spots)

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL



Block: root_block
Architecture: test
System: SAR
System Architecture: xbarRaceArch
Run: test_1

[Viewlogic] 167

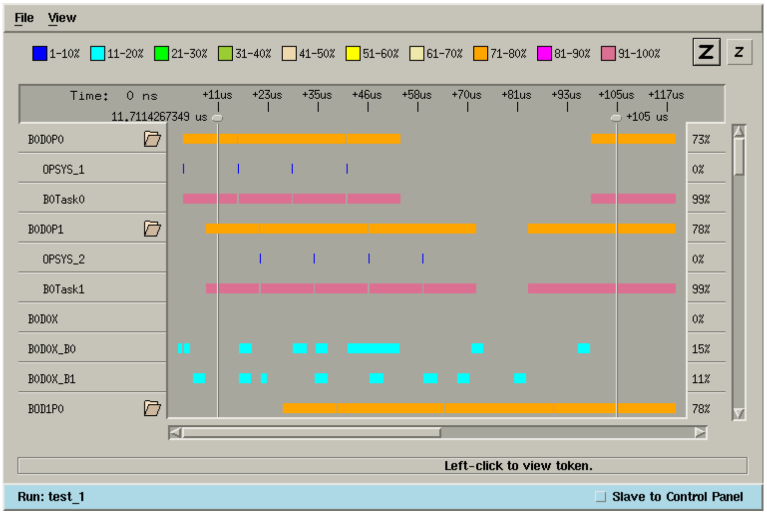
Here is another eArchitect post-simulation data display tool. In this case, its a “hot spot” display which show module utilization in color codes. Modules that appear towards the red side of the spectrum are highly utilized and may represent a bottleneck in the computation. If however, all modules are towards the blue side of the spectrum, the overall system may be over designed resulting in wasted resources.

Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

Analysis of Results in eArchitect
Activity Time Lines

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL



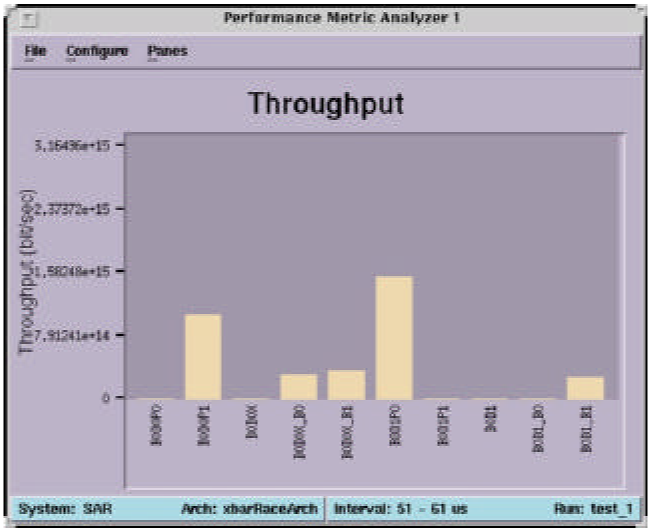
Left-click to view token.

Run: test_1

[Viewlogic] 168

This is a screen shot of the activity time line display available in eArchitect. It is fairly standard.

Analysis of Results in eArchitect
Throughput



System: SAR
Arch: xbarRaceArch
Interval: 51 - 61 us
Run: test_1

[Viewlogic]

Here is the throughput display from eArchitect.

RASSP Reinventing Electronic Design

Methodology · Architecture · Infrastructure

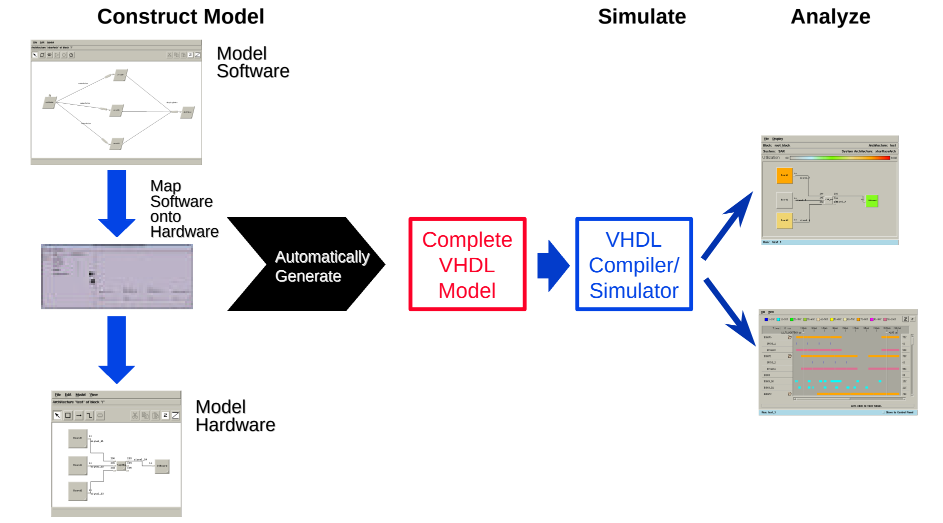
DARPA ● Tri-Service

eArchitect Design Flow

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL



Construct Model

Simulate

Analyze

Model Software

Map Software onto Hardware

Model Hardware

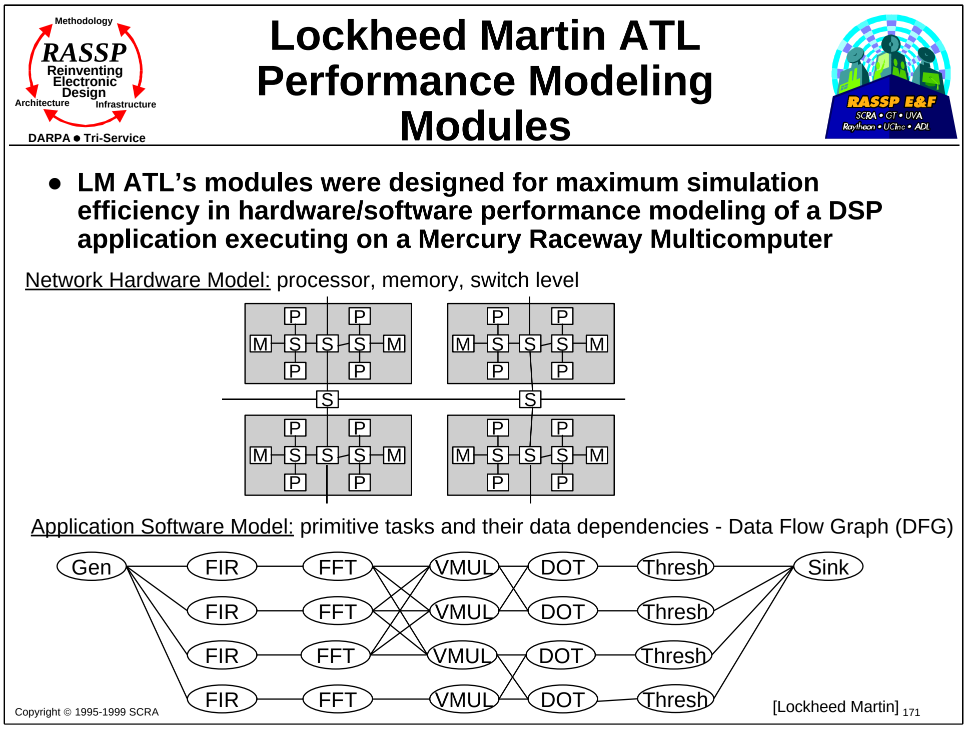
Automatically Generate

Complete VHDL Model

VHDL Compiler/ Simulator

[Viewlogic] 170

This slide shows the overall design flow in eArchitect. Again, the hardware architecture is modeled using the PML library modules configured to model the chosen hardware architecture. This includes specifying the ISA of the chosen processors and their execution rates, and the network configuration and its communication rates. The software is modeled as a set of tasks that communicate in a specific way and take a certain amount of resources in terms of computation and communication. Finally, the mapping of software tasks to processors is specified. The eArchitect tools then generate a VHDL model of the complete system which is then compiled and simulated on the chosen commercial VHDL simulator. The data that results from that simulation can then be displayed graphically by the eArchitect post-simulation analysis tools.



Lockheed Martin ATL Performance Modeling Modules

- **LM ATL's modules were designed for maximum simulation efficiency in hardware/software performance modeling of a DSP application executing on a Mercury Raceway Multicomputer**

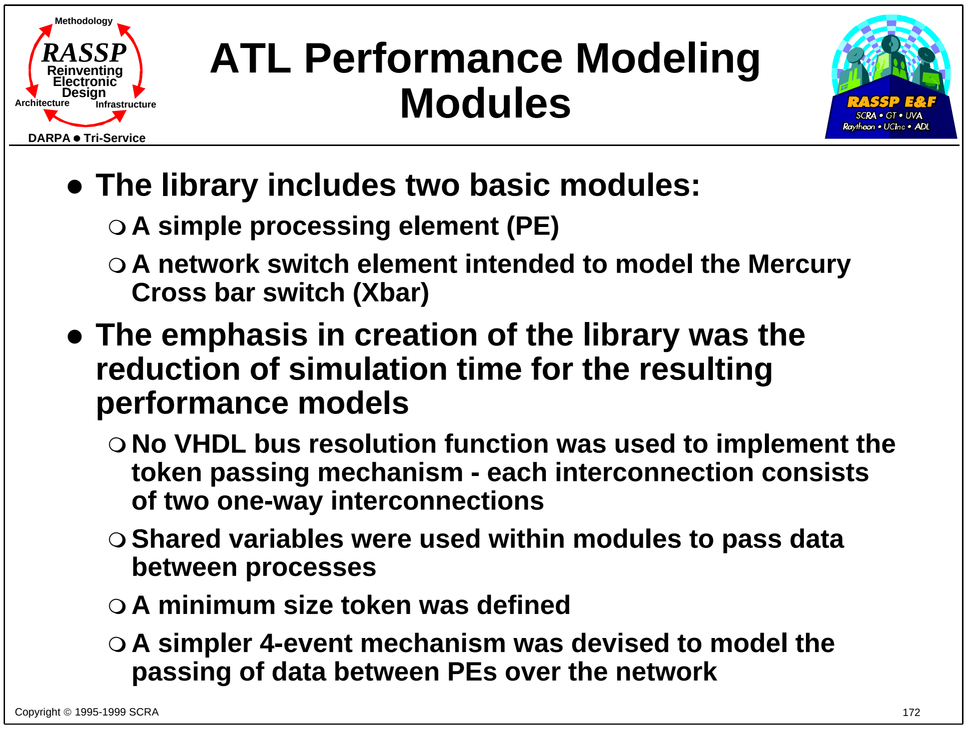
Network Hardware Model: processor, memory, switch level

Application Software Model: primitive tasks and their data dependencies - Data Flow Graph (DFG)

[Lockheed Martin] 171

As part of the RASSP program, ATL was tasked to use performance modeling in the design of several benchmark embedded DSP systems. Their efforts to use PML and ADEPT at an early point in the program were hindered by the long simulation times of both ADEPT and PML models and by the unavailability, at that time, of the eArchitect tool and a suitable PMS level modeling library in ADEPT. In response, they developed a very lightweight PMS level modeling environment for Mercury Raceway systems with an emphasis on reduced simulation times.

Note that both ADEPT and PML have since addressed the simulation time problem with good results.



Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service

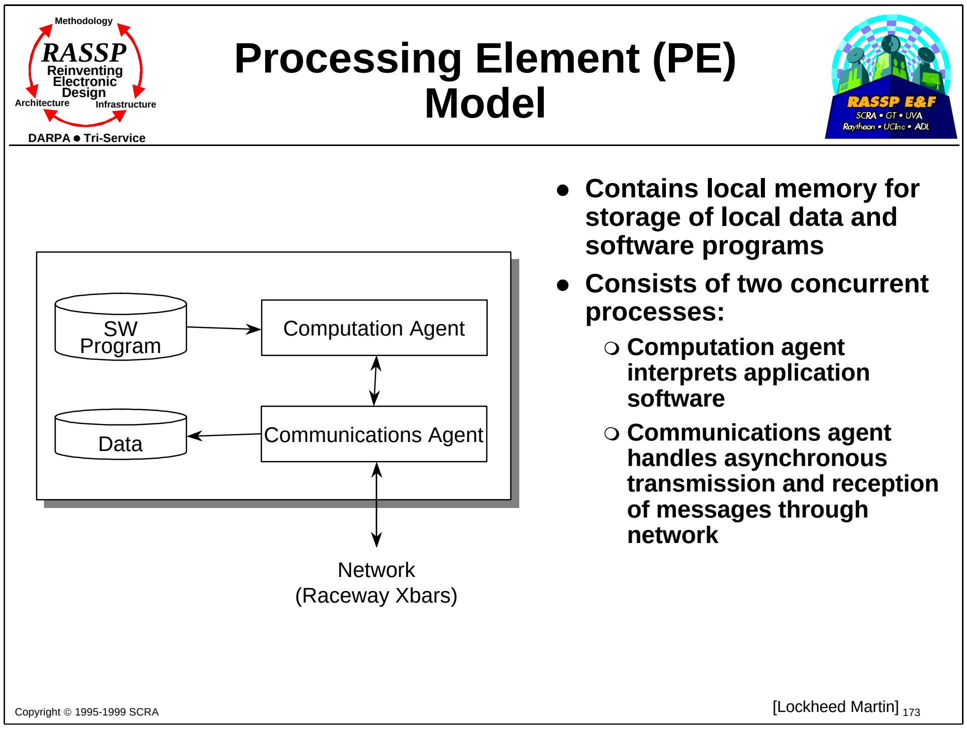
ATL Performance Modeling Modules

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- The library includes two basic modules:
  - ❍ A simple processing element (PE)
  - ❍ A network switch element intended to model the Mercury Cross bar switch (Xbar)
- The emphasis in creation of the library was the reduction of simulation time for the resulting performance models
  - ❍ No VHDL bus resolution function was used to implement the token passing mechanism - each interconnection consists of two one-way interconnections
  - ❍ Shared variables were used within modules to pass data between processes
  - ❍ A minimum size token was defined
  - ❍ A simpler 4-event mechanism was devised to model the passing of data between PEs over the network

The ATL library consists of two components, a processor model (which includes a network interface), and a switch model. The switch is intended to model the Mercury Raceway crossbar switch.

Much emphasis was placed on reducing simulation times and the results were very good in that regard - ATL VHDL performance models of the Raceway system simulate in an equivalent time to models written in C. However, the disadvantage of this more ad hoc approach over ADEPT or PML is the limited library of components available (which had to be written specifically for this network model) and a less general applicability.



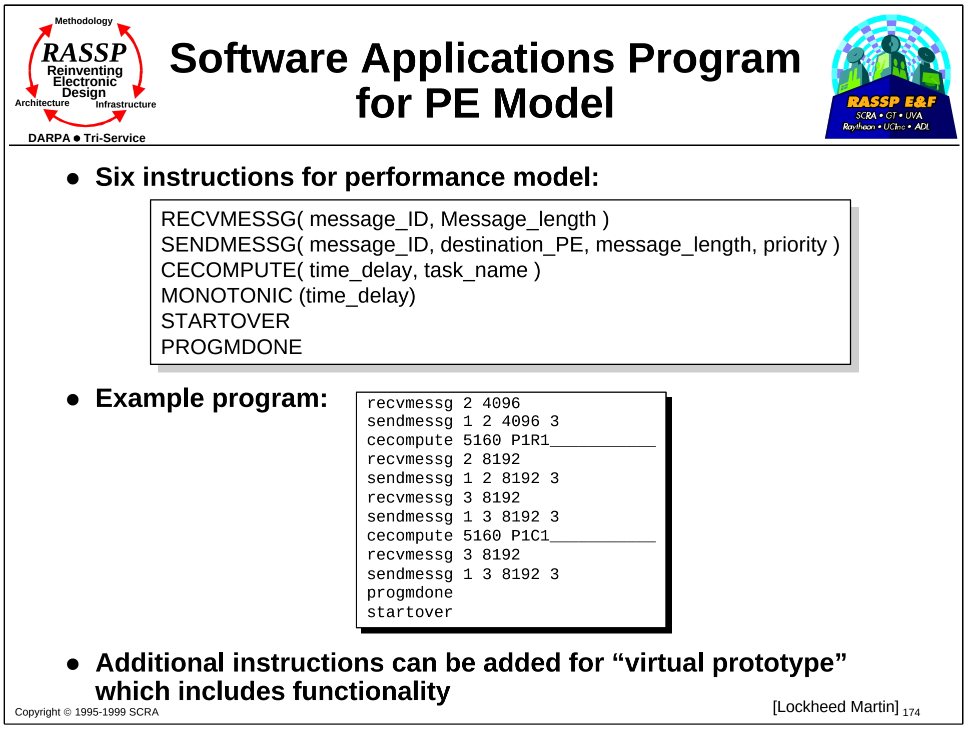
Processing Element (PE) Model

SW
Program
Computation Agent
Data
Communications Agent
Network
(Raceway Xbars)

- Contains local memory for storage of local data and software programs
- Consists of two concurrent processes:
  - ❍ Computation agent interprets application software
  - ❍ Communications agent handles asynchronous transmission and reception of messages through network

[Lockheed Martin] 173

The ATL processing element (PE) consists of two parts; the computation agent that reads CPU instruction from a file and executes them, and a communications agent that interfaces to the network model and handles message sends and receives.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

Software Applications Program for PE Model

RASSP E&F
SCRA • GT • UVA
Raytheon • UCInc • ADL

- **Six instructions for performance model:**

```
RECVMESSG( message_ID, Message_length )
SENDMESSG( message_ID, destination_PE, message_length, priority )
CECOMPUTE( time_delay, task_name )
MONOTONIC (time_delay)
STARTOVER
PROGMDONE
```

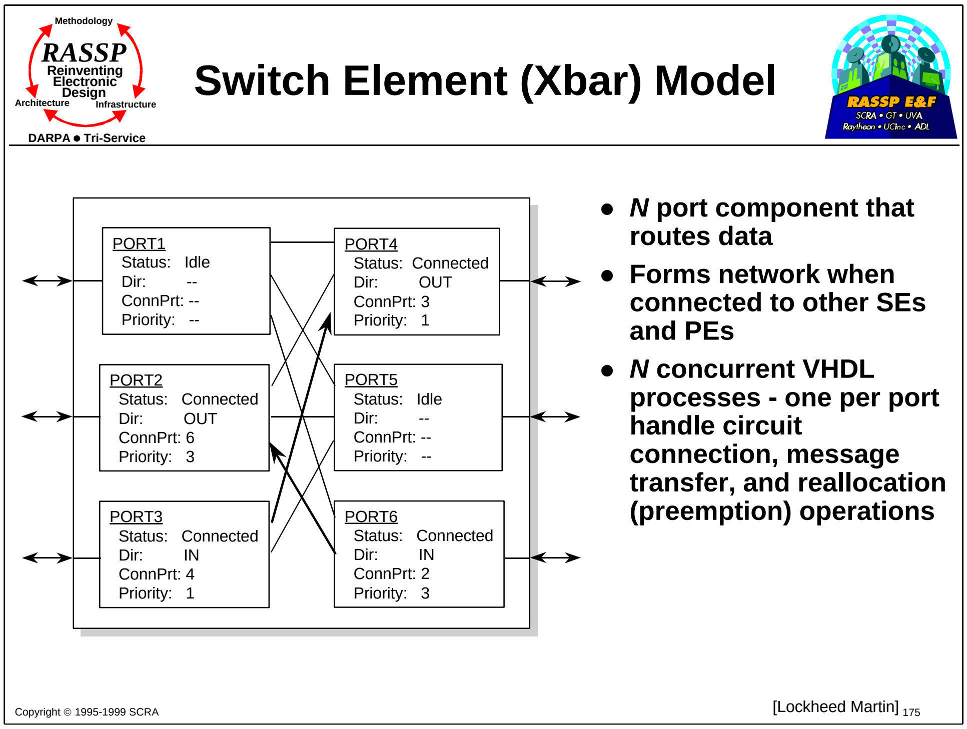
- **Example program:**

```
recvmessg 2 4096
sendmessg 1 2 4096 3
cecompute 5160 P1R1___________
recvmessg 2 8192
sendmessg 1 2 8192 3
recvmessg 3 8192
sendmessg 1 3 8192 3
cecompute 5160 P1C1___________
recvmessg 3 8192
sendmessg 1 3 8192 3
progmdone
startover
```

- **Additional instructions can be added for “virtual prototype” which includes functionality**

[Lockheed Martin] 174

The ATL CPU model has 6 instructions that fall into three basic modes, compute, send and receive. Additional instructions that perform actual data translations (complex multiply, matrix operations, etc.) can be added in the first “virtual prototype” stage when some functionality is added to the model.



Switch Element (Xbar) Model

PORT1
Status: Idle
Dir: --
ConnPrt: --
Priority: --

PORT2
Status: Connected
Dir: OUT
ConnPrt: 6
Priority: 3

PORT3
Status: Connected
Dir: IN
ConnPrt: 4
Priority: 1

PORT4
Status: Connected
Dir: OUT
ConnPrt: 3
Priority: 1

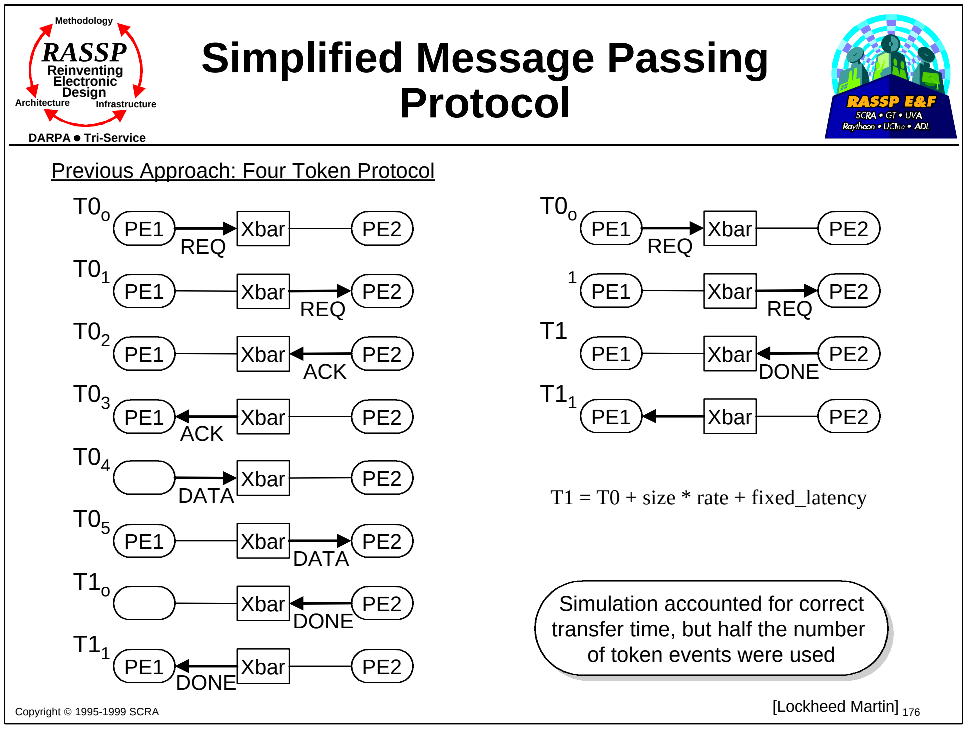
PORT5
Status: Idle
Dir: --
ConnPrt: --
Priority: --

PORT6
Status: Connected
Dir: IN
ConnPrt: 2
Priority: 3

- ***N*** **port component that routes data**
- **Forms network when connected to other SEs and PEs**
- ***N*** **concurrent VHDL processes - one per port handle circuit connection, message transfer, and reallocation (preemption) operations**

[Lockheed Martin] 175

The switch element in the ATL library models a 6 port Mercury Raceway crossbar switch. This crossbar is circuit switched and can handle up to three simultaneous connections. It is modeled in VHDL using 6 concurrent VHDL processes, one to handle each port on the crossbar. The crossbar functions of circuit setup, teardown and preemption are handled.



Simplified Message Passing Protocol

Previous Approach: Four Token Protocol

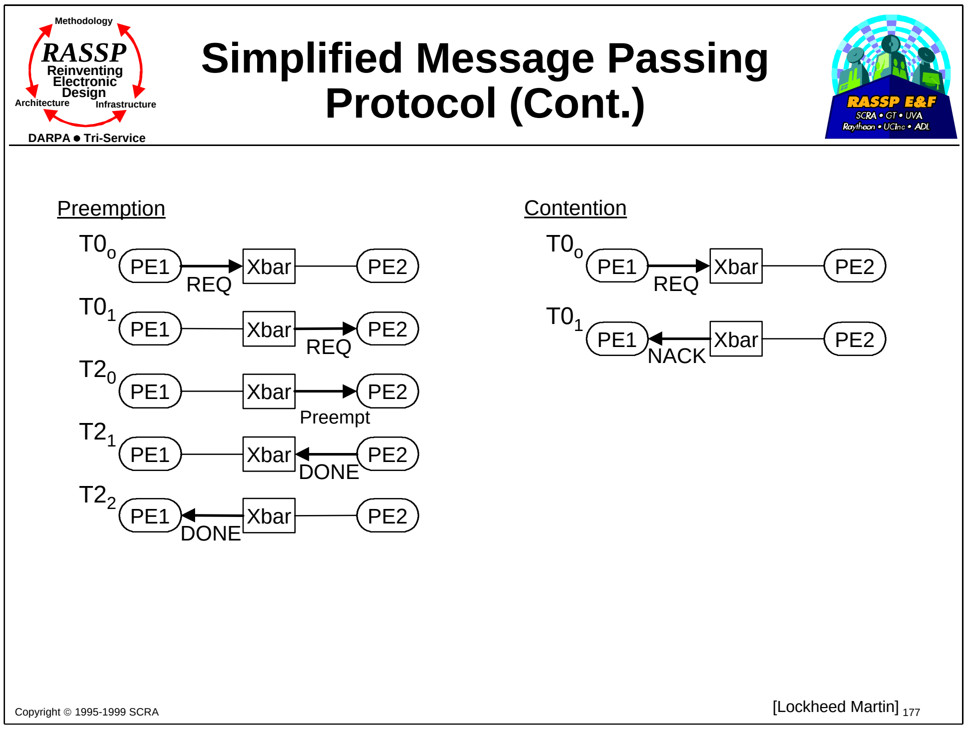
T1 = T0 + size * rate + fixed_latency

Simulation accounted for correct transfer time, but half the number of token events were used

[Lockheed Martin] 176

This is an illustration of how the normal message passing protocol, as modeled in a performance modeling environment, was simplified to reduce the number of tokens needed. Note that this token passing mechanism is a modeling artifact, it is not how the Raceway actually passes data, so changing it does not affect the model fidelity as long as care is taken to keep the timing the same.

Also note that the ATL module do not use bus resolution functions to pass tokens - they use two unidirectional signals - further decreasing the execution time of the simulation.



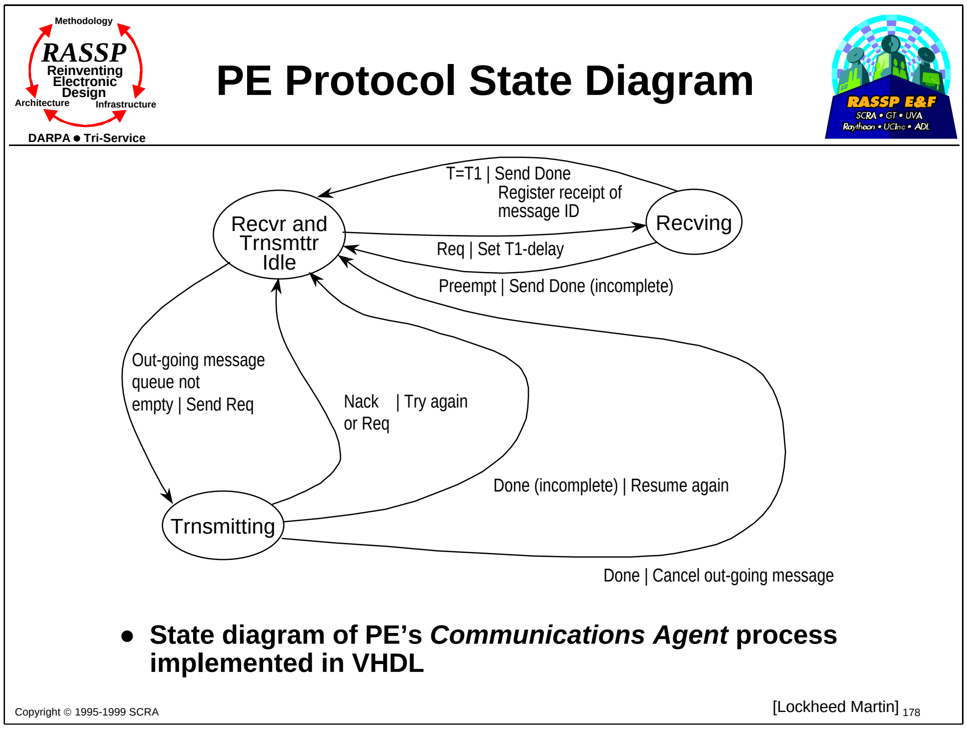
Simplified Message Passing Protocol (Cont.)

Preemption

Contention

[Lockheed Martin] 177

These figures illustrate how preemption and contention (requesting a busy path) are handled in the simplified ATL protocol.



PE Protocol State Diagram

Recvr and Trnsmttr Idle

Recving

Trnsmitting

T=T1 | Send Done Register receipt of message ID

Req | Set T1-delay

Preempt | Send Done (incomplete)

Out-going message queue not empty | Send Req

Nack or Req | Try again

Done (incomplete) | Resume again

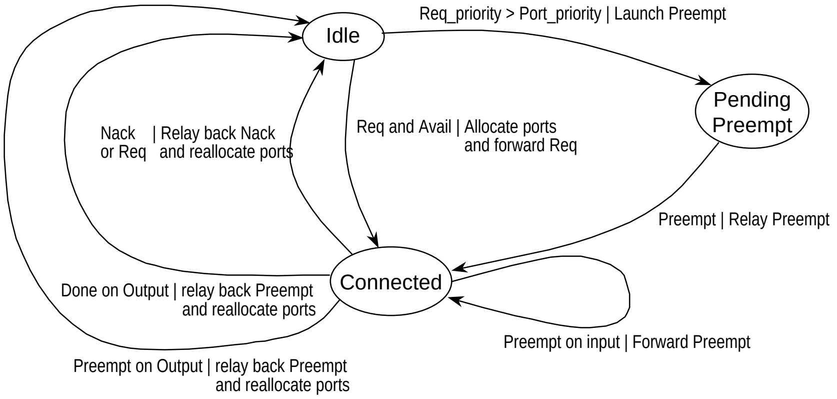
Done | Cancel out-going message

- **State diagram of PE's *Communications Agent* process implemented in VHDL**

[Lockheed Martin] 178

The communications agent and how it handles the various network functions such as requesting a path for a message, sending the message, and responding to preemption, is fairly complex, so it was designed as a state machine. This state machine was then implemented in VHDL to perform the required function. Note that within the PE VHDL code, the communications agent and computation agent pass data back and forth using shared variable instead of signals, further reducing simulation execution time.

SE Port Process State Diagram


Idle
Pending Preempt
Connected
Req_priority > Port_priority | Launch Preempt
Req and Avail | Allocate ports and forward Req
Nack or Req | Relay back Nack and reallocate ports
Preempt | Relay Preempt
Done on Output | relay back Preempt and reallocate ports
Preempt on input | Forward Preempt
Preempt on Output | relay back Preempt and reallocate ports

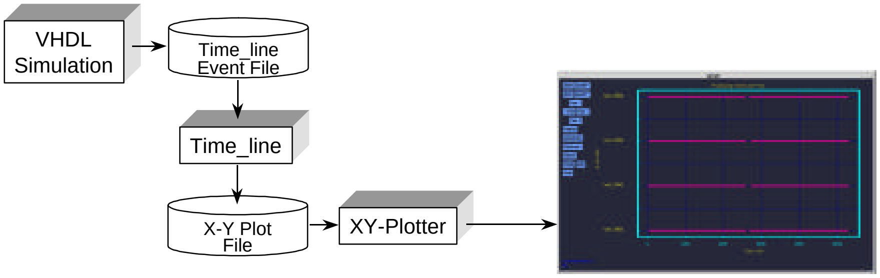
- **State diagram of VHDL process for each port of the SE**

[Lockheed Martin] 179

This is the state diagram for the VHDL process that implements the procedures of the port in the switch element (crossbar).

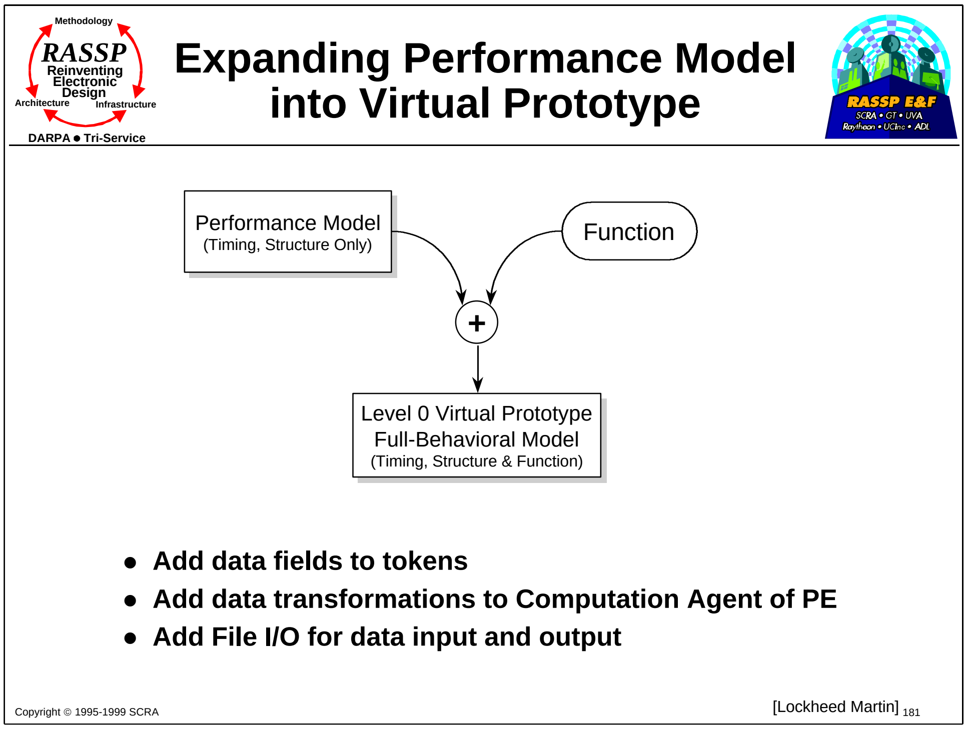
Performance Metrics from ATL Performance Models

- Statistics are recorded using shared variables
- Simulation output includes:
  - ❍ Link and PE utilization
  - ❍ Resource and link contentions
  - ❍ Processor and communications time-lines



[Lockheed Martin] 180

A simple set of tools for collecting and analyzing performance metrics from the ATL modules was devised. The main tool is a time line utilization analysis tool that is capable of displaying both the times when the PEs are busy computing and when the communications network is busy.



Expanding Performance Model into Virtual Prototype

Performance Model
(Timing, Structure Only)

Function

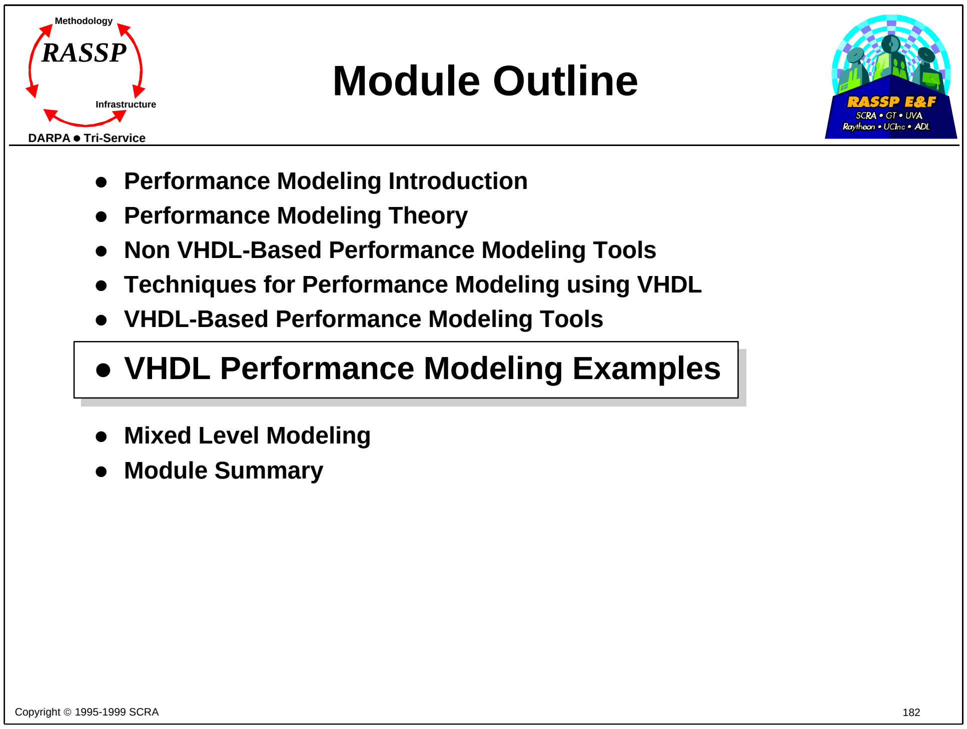
+

Level 0 Virtual Prototype
Full-Behavioral Model
(Timing, Structure & Function)

- **Add data fields to tokens**
- **Add data transformations to Computation Agent of PE**
- **Add File I/O for data input and output**

[Lockheed Martin] 181

After a high level performance model (with timing, but no functional information) is developed and analyzed, function can be added in terms of data values and data transformations. This forms what is termed in the Virtual Prototyping module as a level 0 virtual prototype (high level function plus timing).



Methodology RASSP Infrastructure

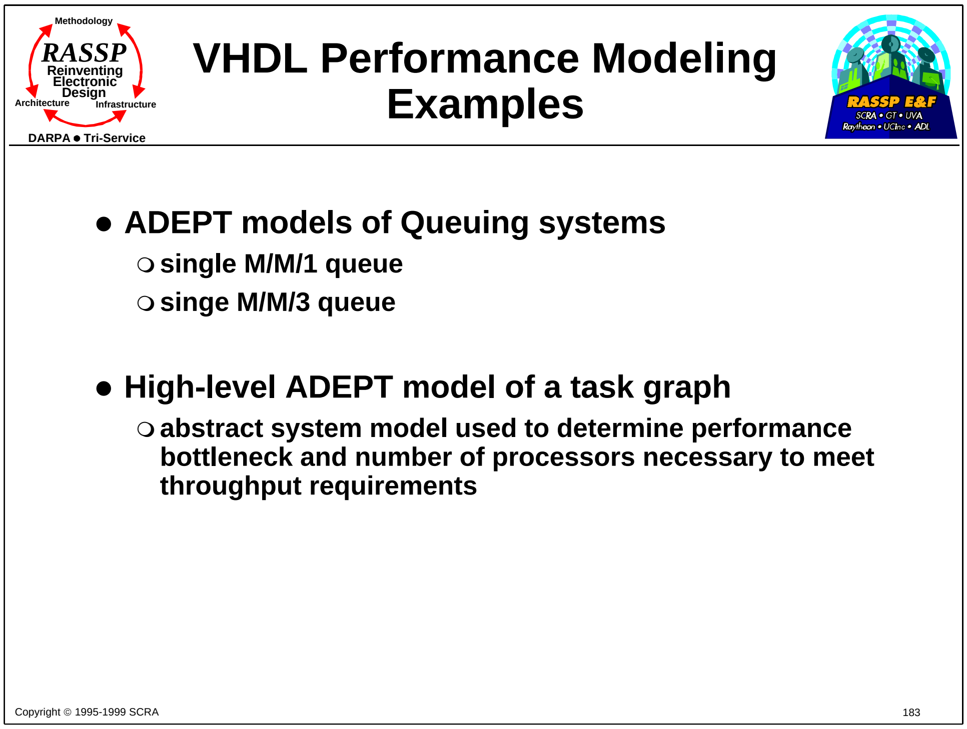
DARPA ● Tri-Service

Module Outline

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

- Performance Modeling Introduction
- Performance Modeling Theory
- Non VHDL-Based Performance Modeling Tools
- Techniques for Performance Modeling using VHDL
- VHDL-Based Performance Modeling Tools
- **VHDL Performance Modeling Examples**
- Mixed Level Modeling
- Module Summary

Module Outline

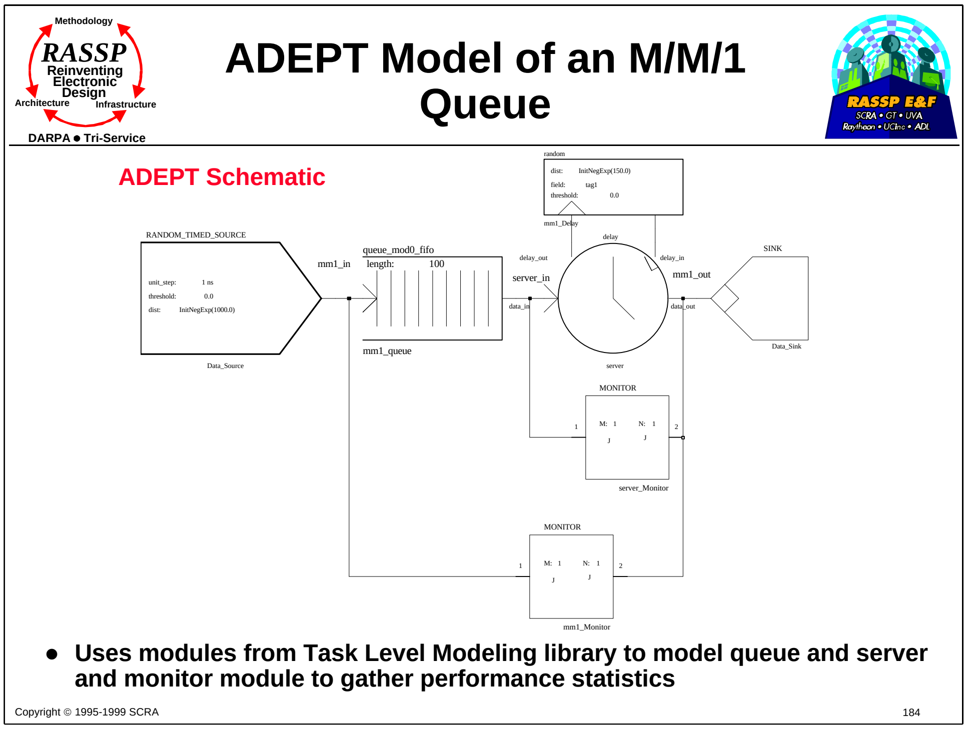


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

VHDL Performance Modeling Examples

- ADEPT models of Queuing systems
  - ❍ single M/M/1 queue
  - ❍ singe M/M/3 queue
- High-level ADEPT model of a task graph
  - ❍ abstract system model used to determine performance bottleneck and number of processors necessary to meet throughput requirements

There are several examples of VHDL based performance models included in this module. Most are based on the ADEPT system, but on uses the ATL performance modeling modules. However, there are many more examples available in the documentation for eArchitect and ADEPT and in the applications notes and case studies prepared as part of the RASSP program.



Methodology RASSP Reinventing Electronic Design Architecture Infrastructure

DARPA ● Tri-Service

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

ADEPT Model of an M/M/1 Queue

ADEPT Schematic

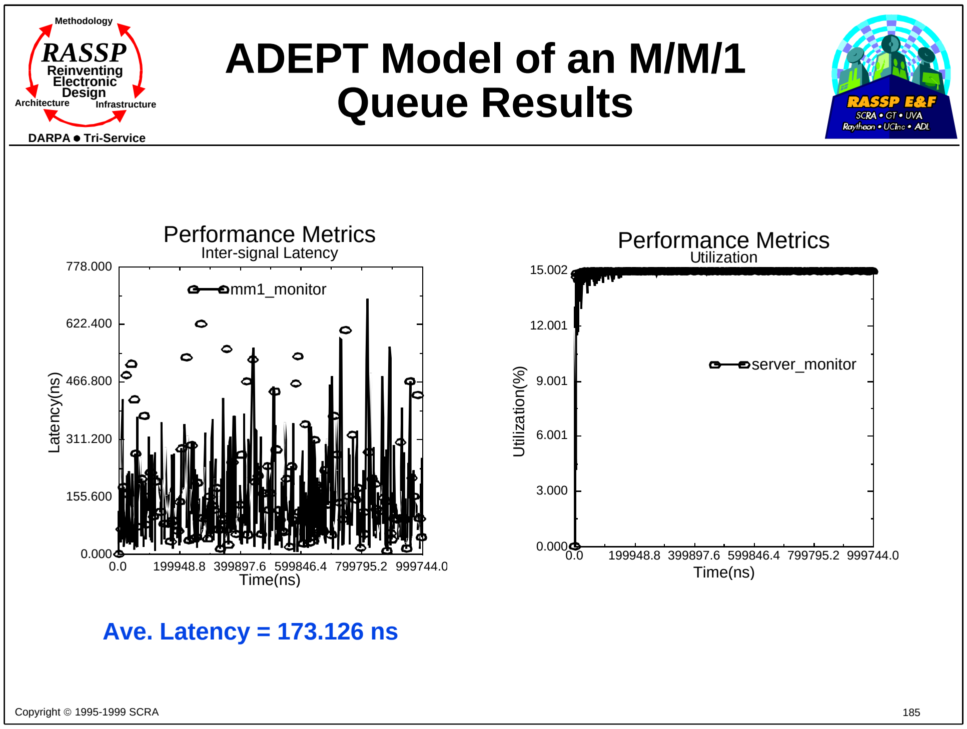
- **Uses modules from Task Level Modeling library to model queue and server and monitor module to gather performance statistics**

Copyright © 1995-1999 SCRA 184

This is a simple model of the M/M/1 queuing system presented and analyzed earlier, using the ADEPT system. The modules used to construct this model come from the ADEPT Task Level Modeling and Module Builder’s libraries.

The random_timed_source module generates a token with a random exponential arrival rate with a mean of 1000 ns (this example is modeled on a ns time scale instead of the ms time scale of the analytical example - the results are the same however). The delay module is connected to a random module such that it has a random, exponential service rate with a mean of 150 ns.

The monitor modules are standard ADEPT modules that are place in an ADEPT model to measure standard performance metrics. They record tokens as the pass by their inputs and outputs and write the information into files that are then interpreted and displayed by the ADEPT post-simulation analysis tools.



ADEPT Model of an M/M/1 Queue Results

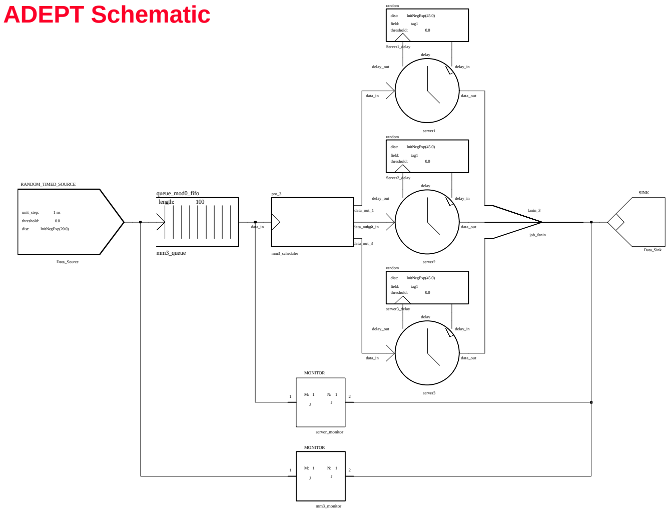
Ave. Latency = 173.126 ns

These are the results of the simulation of the M/M/1 ADEPT model. Note that the average latency of jobs (tokens) within the system is 173.126 ns as reported by the ADEPT analysis tools and that the average utilization of the server is 15%.

Recall that the analytical results for this model were 176.5 ns and 15% respectively.

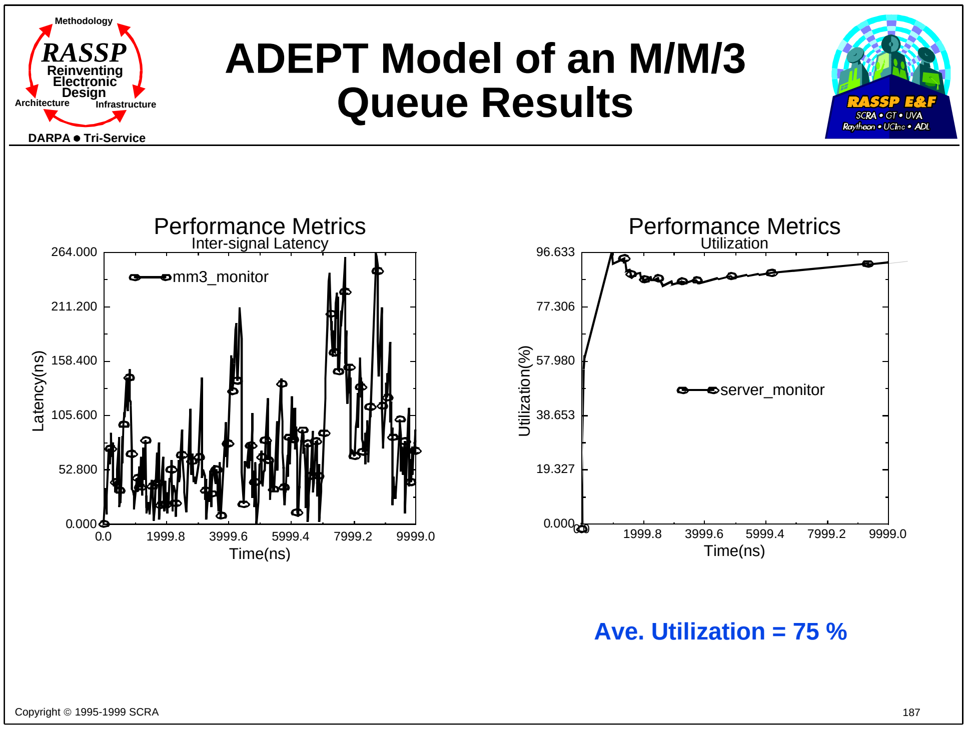
ADEPT Model of an M/M/3 Queue

ADEPT Schematic



186

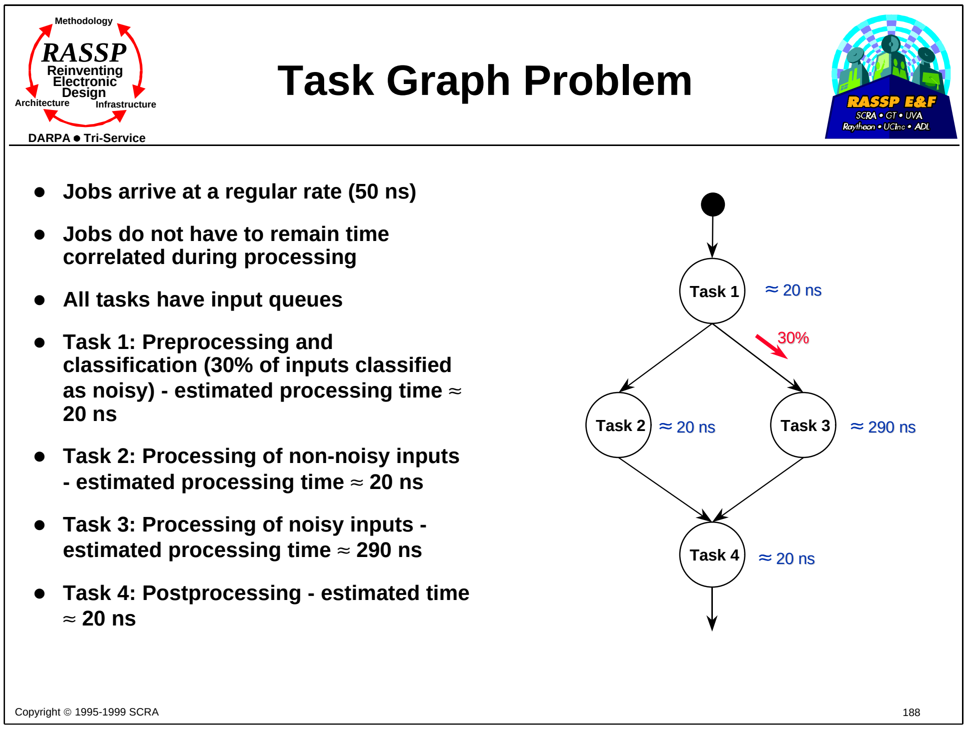
This an ADEPT model of an M/M/3 queue. It is similar to the M/M/1 model except that it obviously has three servers (delay/random module combinations). The pro_3 module is from the Task Level Modeling library and it routes tokens on its input, from the queue, to any output that is free (I.e. any server that is not busy). Note that it has a built-in priority that if more than one server is free, then it routes the token (job) to the lowest numbered output first, but that is immaterial to this model.



ADEPT Model of an M/M/3 Queue Results

Ave. Utilization = 75 %

Here are the results of the ADEPT M/M/3 model. Note that the average utilization for the servers is 75% which agrees with the analytical results and the average latency seems to be close to the analytical result of 81 ns (again, this simulation was on a ns scale as opposed to the ms scale of the analytical analysis).



Task Graph Problem

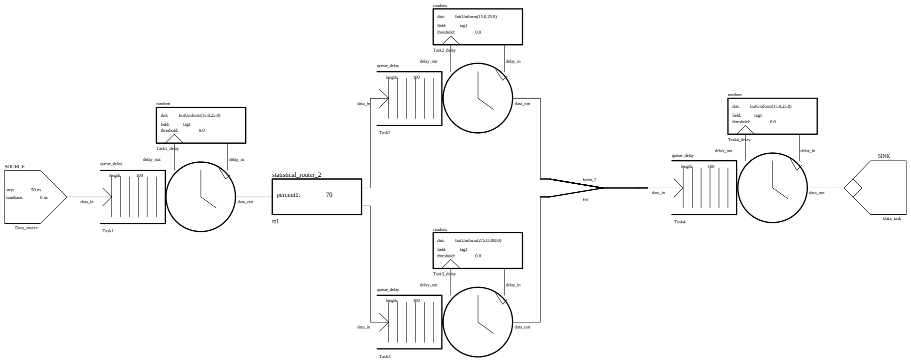
- **Jobs arrive at a regular rate (50 ns)**
- **Jobs do not have to remain time correlated during processing**
- **All tasks have input queues**
- **Task 1: Preprocessing and classification (30% of inputs classified as noisy) - estimated processing time ≈ 20 ns**
- **Task 2: Processing of non-noisy inputs - estimated processing time ≈ 20 ns**
- **Task 3: Processing of noisy inputs - estimated processing time ≈ 290 ns**
- **Task 4: Postprocessing - estimated time ≈ 20 ns**

This is a simple task graph problem that further illustrates the ADEPT performance modeling environment. In this problem, there is a set of jobs (say images to process) that arrive from a sensor at a regular rate. The first task is to classify the images as to their clarity - noisy or non-noisy. An average of 30% of the images are classified as noisy and must be filtered. The remaining non-noisy images must be formatted, but that takes much less time than the filtering operation. Finally, all images must be compressed for storage. Images do not need to remain correlated in the time that they arrived as they pass through the system, I.e., non-noisy images may move ahead of noisy images during processing.

An ADEPT model will be constructed to explore the issue of how many processors are required to perform the noisy image filtering to meet throughput requirements. A more detailed version of this model, with links to lower levels of hardware/software codesign and mixed level modeling, is available in the standard ADEPT deliverable.

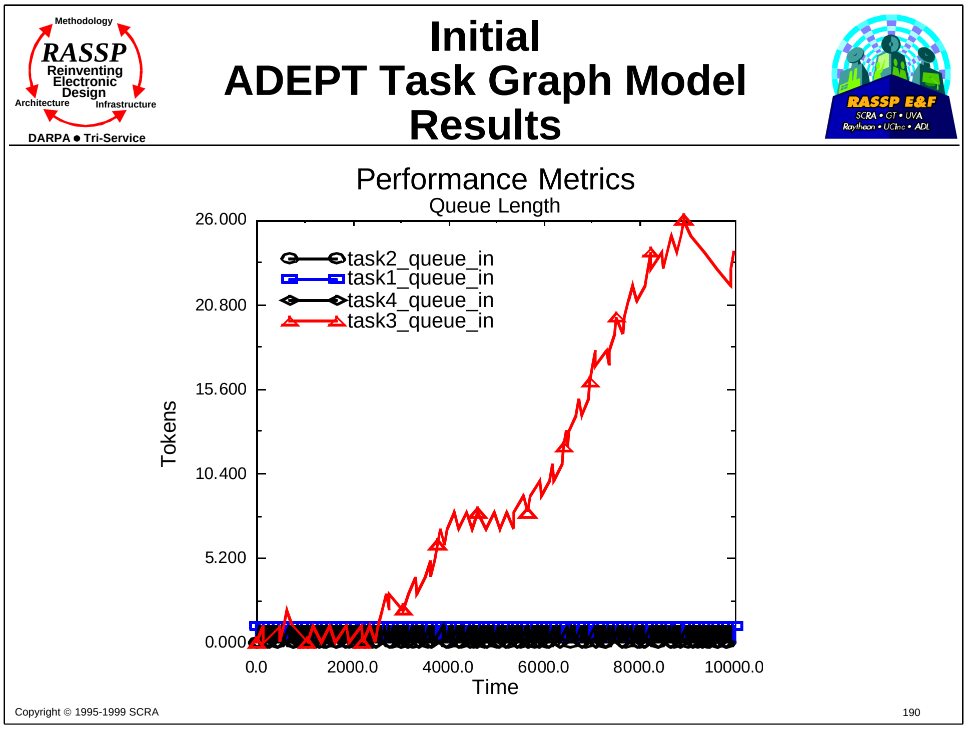
Initial
ADEPT Task Graph Model

ADEPT Schematic



- **Queuing network model constructed using Task Level Modeling library modules**

This is the initial ADEPT performance model of the task graph problem. It is a high-level queuing network model with only one processor performing the noisy image filtering process.



RASSP Reinventing Electronic Design

Methodology Architecture Infrastructure

DARPA ● Tri-Service

Initial ADEPT Task Graph Model Results

RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

Performance Metrics

190

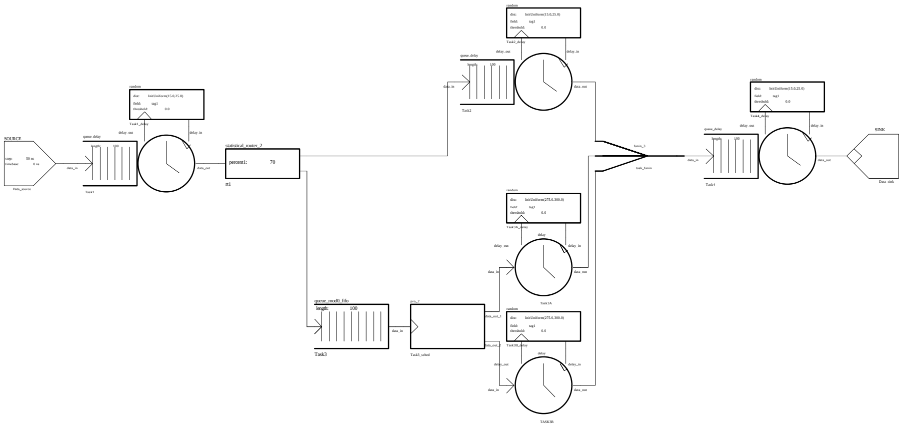
This is a plot of the number of items in the input queues to each task. Note that the number of items in the input queue to task 3 is increasing. Despite the slight decrease in the number of images in the queue towards the end of the simulation, it is clear that one processor is not enough to keep up with the number of filtering requests and that at least one more processor performing that task will be necessary.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Revised ADEPT Task Graph Model

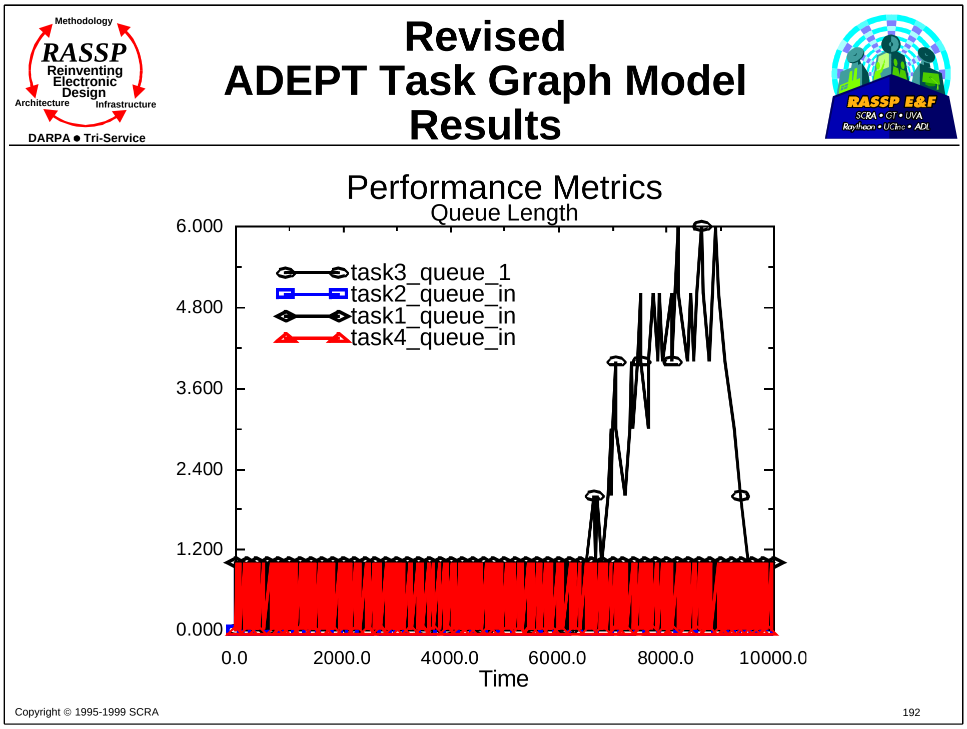
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

ADEPT Schematic



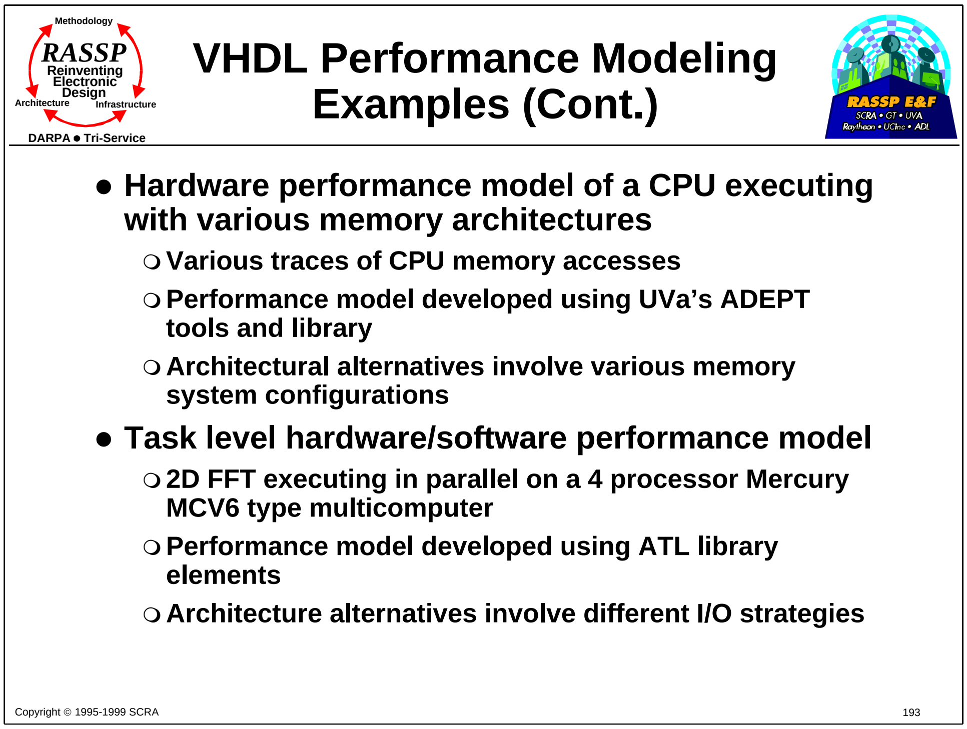
Here is a model with two processors for task 3. Again, a pro_2 module is used to schedule jobs from the task 3 queue onto idle task 3 processors.

Again, a more detailed model of this scenario, where task 3 is taken down one more level to model actual software algorithms executing on a Digital Signal Processor, and task 4 is taken down to a behavioral model of an ALU using mixed level modeling, is included in the ADEPT package.



Revised ADEPT Task Graph Model Results

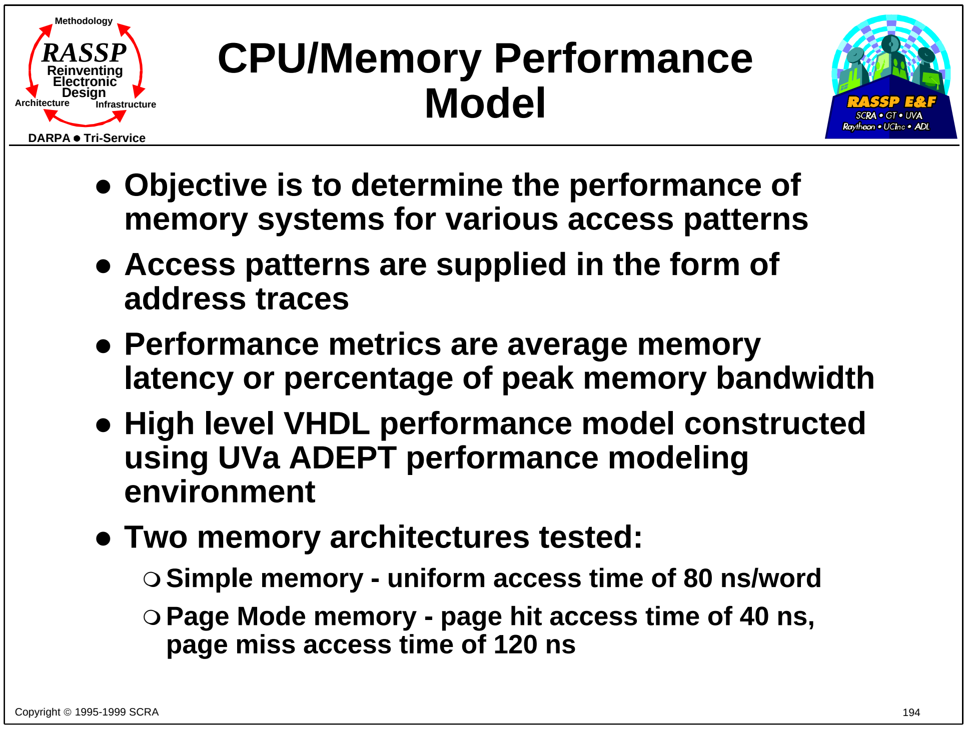
Here is the plot of queue depths for the two task 3 processor model and it shows that the depth of the task 3 queue is bounded, so two processors for that task should be enough. However, more detail should be added to the model to further prove this conclusion as the results show that the task 3 queue still may fill up if the estimate of the time required to perform the filtering is optimistic.



VHDL Performance Modeling Examples (Cont.)

- Hardware performance model of a CPU executing with various memory architectures
  - ❍ Various traces of CPU memory accesses
  - ❍ Performance model developed using UVa’s ADEPT tools and library
  - ❍ Architectural alternatives involve various memory system configurations
- Task level hardware/software performance model
  - ❍ 2D FFT executing in parallel on a 4 processor Mercury MCV6 type multicomputer
  - ❍ Performance model developed using ATL library elements
  - ❍ Architecture alternatives involve different I/O strategies

Next will be presented two more performance modeling example. One, a performance model of a CPU and memory modeled with ADEPT, and another, a hardware/software task level performance model done with the ATL performance modeling modules.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

CPU/Memory Performance Model

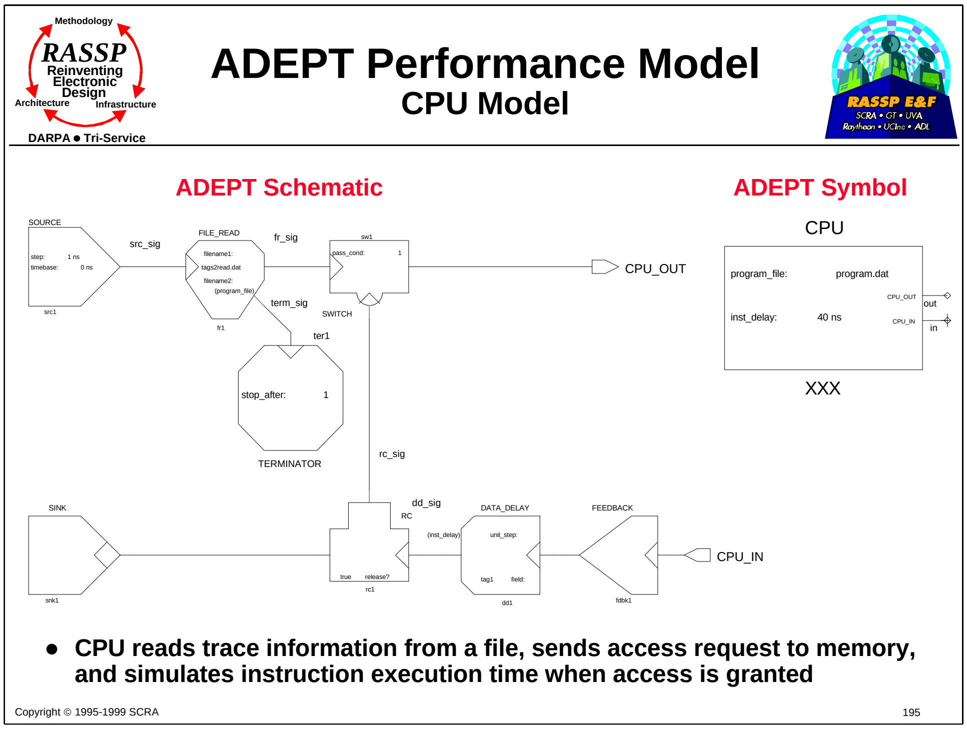
- Objective is to determine the performance of memory systems for various access patterns
- Access patterns are supplied in the form of address traces
- Performance metrics are average memory latency or percentage of peak memory bandwidth
- High level VHDL performance model constructed using UVa ADEPT performance modeling environment
- Two memory architectures tested:
  - ❍ Simple memory - uniform access time of 80 ns/word
  - ❍ Page Mode memory - page hit access time of 40 ns, page miss access time of 120 ns

The CPU/memory performance model is a simple example of a “hardware only” type of performance model. The objective of the performance model is to be able to determine the performance of various memory system architectures on typical memory traces.

At this point, only two different memory architectures were tested:

- a simple memory model in which each access takes a uniform time (based on the size of the access) of 80 ns per word.

- a page mode dram memory model where the memory system is divided up into “pages” of a specified size. If an access is made to a memory location that is on the same page as the one immediately preceding in, the “page hit access time” is 40 ns. If the access is on a different page, then the current page has to be closed and a new one opened which results in a “page miss access time” of 120 ns. Therefore, grouping accesses into groups that hit the same page (as will be seen in the DAXPY example trace) can result in significantly decreased access time.



ADEPT Performance Model

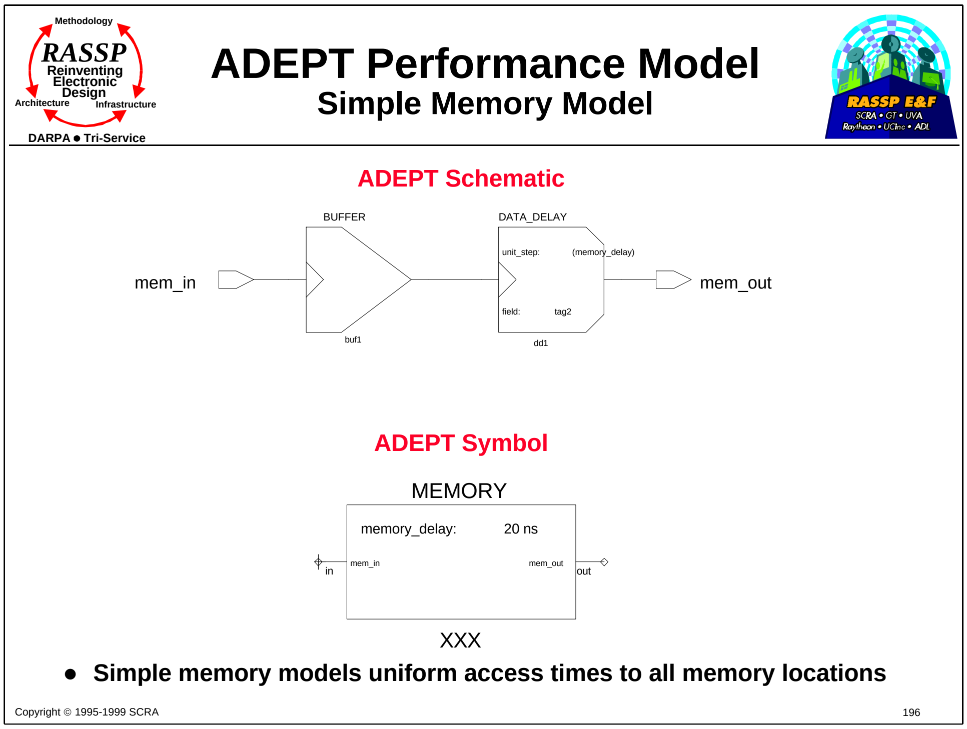
CPU Model

ADEPT Schematic

ADEPT Symbol

- **CPU reads trace information from a file, sends access request to memory, and simulates instruction execution time when access is granted**

This is the simple CPU model. At the start of simulation time, the Source module generates a token which passes through the File_read module and picks up the first set of trace information. The token then weights at the Switch module until it is released by it. Also at time zero, the Feedback module generates an initial token (once at time 0 only) which enters the Data_delay module. The Data_delay module models the actual execution of instructions by the CPU and delays the CPU's instruction time (10 ns) times the number of instructions the current memory access allows to execute (contained on tag1 of the token). The initial token from the feedback module delays for one instruction (10 ns) and then passes through the RC module. The RC module produces a "control" token on its output which is connected to the Switch module which causes the Switch to release the next token to the memory system. The token from the RC module is then consumed by the Sink module. After the token leaves the switch module and is passed to the memory system model (through the CPU_OUT port), the Source module produces another token which passes through the File_Read module and waits at the Switch module until it is released by the token returning from the memory model (through the CPU_IN port). When the File_read module reaches the end of the address trace file, it sends a "control" token to the terminator module which terminates the simulation.



ADEPT Performance Model

Simple Memory Model

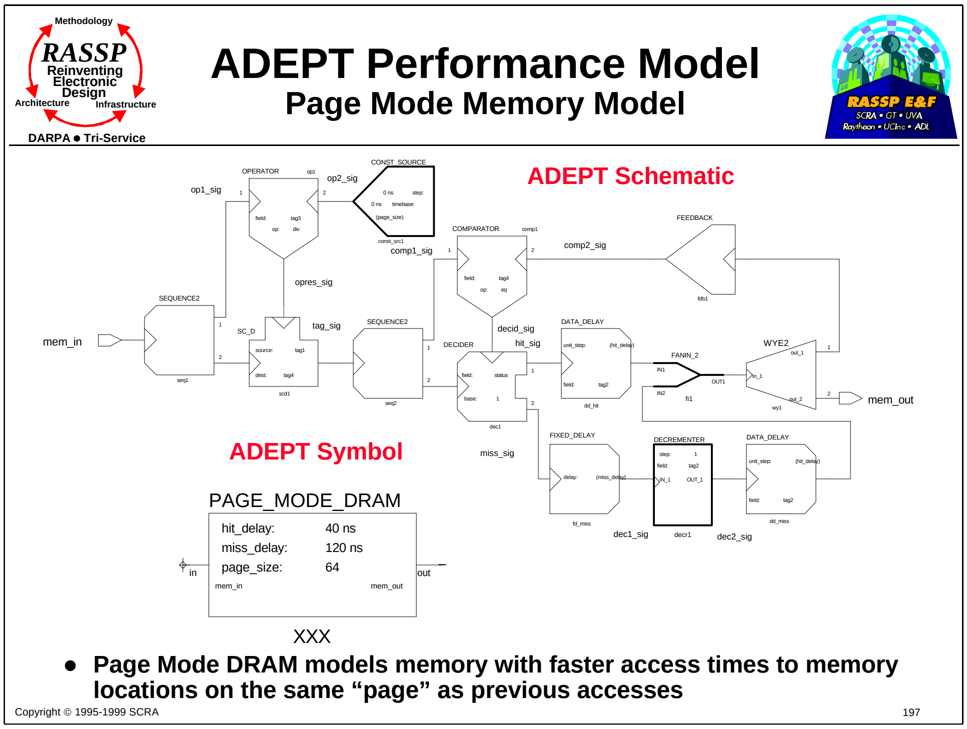
ADEPT Schematic

ADEPT Symbol

- **Simple memory models uniform access times to all memory locations**

196

This is the simple memory system model. When the token arrives from the CPU (through the MEM_IN port), it is buffered by the buffer module and then waits at the data delay module for a time determined by the number of words the access is for (determined by tag2 of the incoming token). Notice that the access time is independent of the actual address that is addresses (specified by tag3 of the token). Also note that the default delay time is 20 ns per word, but that is overwritten by the 80 ns specified on the top level schematic (as seen in the coming slide).



ADEPT Performance Model

Page Mode Memory Model

ADEPT Schematic

ADEPT Symbol

- **Page Mode DRAM models memory with faster access times to memory locations on the same “page” as previous accesses**

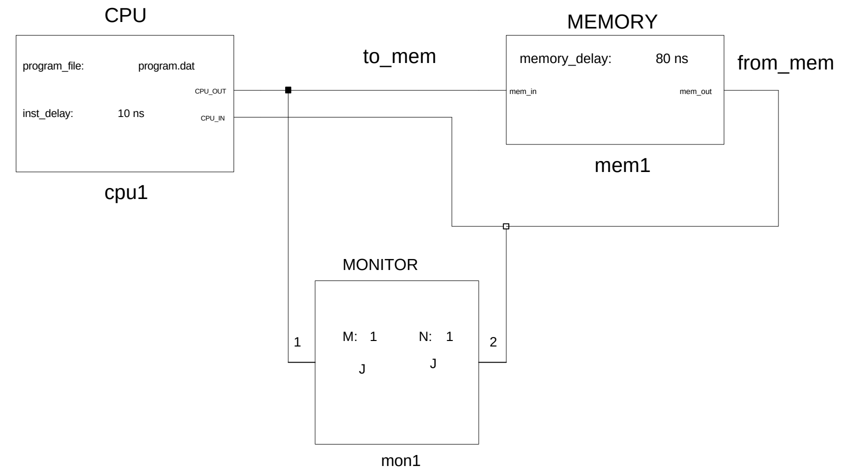
197

This is the model of the page mode dram which is more complex than the simple memory model, but still very straight froward. When a token enters the model (through the MEM_IN port), the Sequence module creates a copy of it and send it to the Operator module. The address of that token (on tag3) is divided by the specified page size (provided on tag3 of the other token input to the Operator by the Constant Source module and the Page_size generic on the overall symbol) to generate the resulting page number on tag1 of the output at the bottom of the Operator module. Once this process is complete, the first Sequence module passes the original token to the SC_D module where the page number is written onto tag4 for the token. It then passes to the second Sequence module which creates a copy of the token and send it to the Comparator module. The comparator module compares the page number on tag4 of the token to the previous page number stored on its other input token. If they are equal, the Comparator signals the Decider module to send the original token through the Data_delay that has the hit_delay. If they are not equal, the Decider sends the token though the lower path. In the lower path, the token is delayed for one miss_delay time to simulate the opening of the new page and the accessing of the first word of the request. Then the number of words requested is decremented by one and the token is delayed for the remaining number of words times the hit_delay. Finally the token passes through the Wye module which sends one copy of the token, containing the new current page number on its tag4, to the Comparator module and another copy out of the memory back to the CPU.

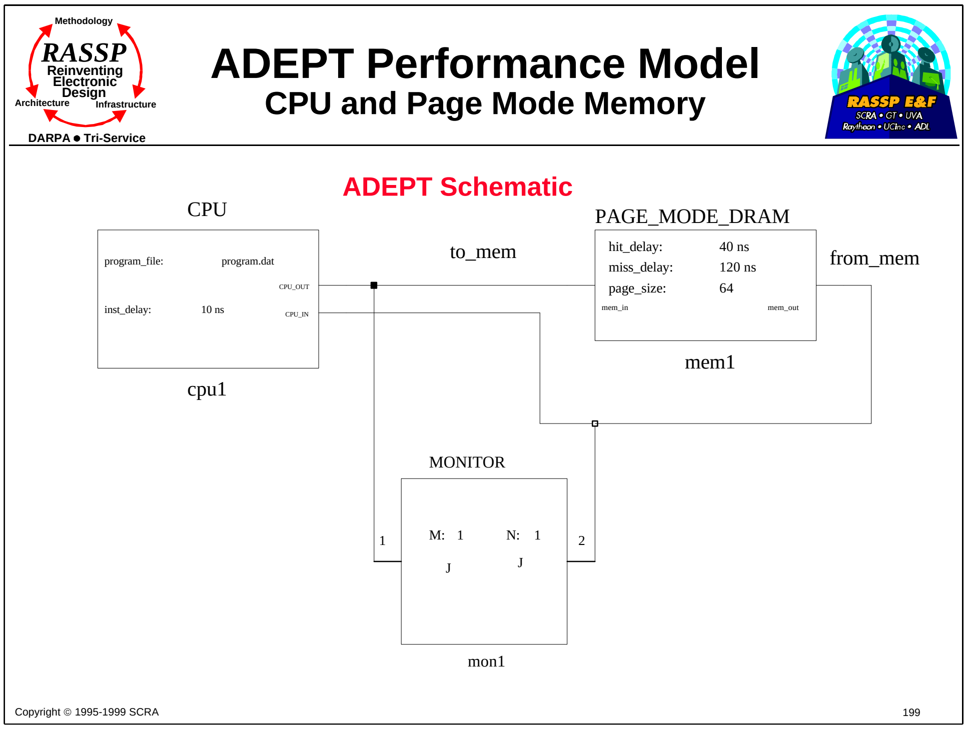
ADEPT Performance Model

CPU and Simple Memory

ADEPT Schematic



This is the ADEPT schematic of the overall model with the simple memory. Notice that the memory access time on the memory model has been changed to 80 ns which will override the 20 ns default as explained before.

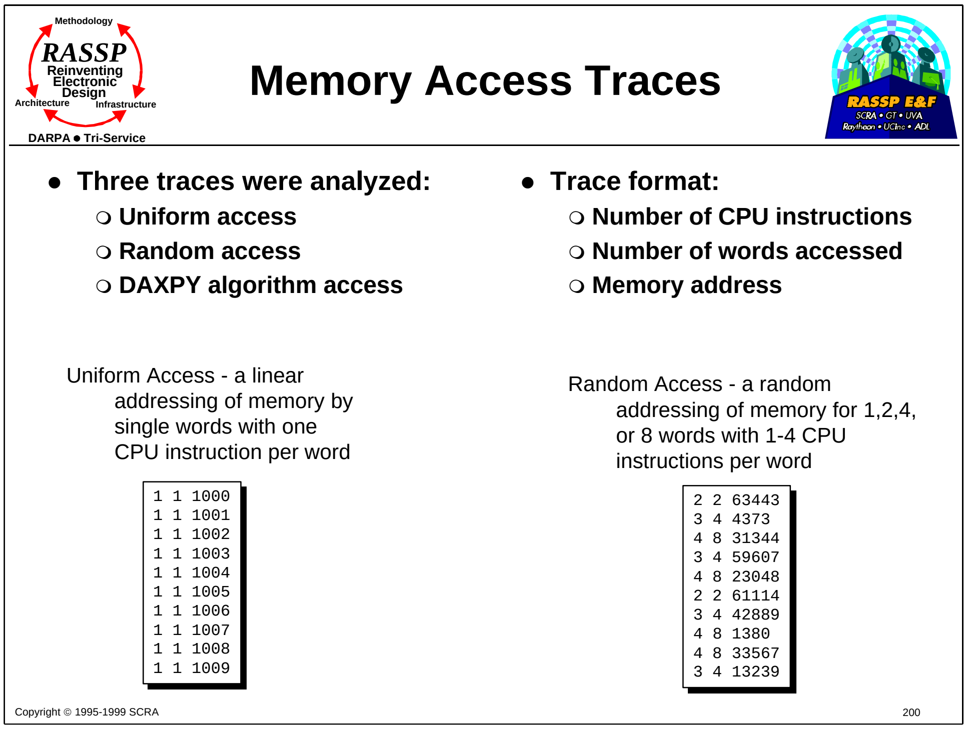


ADEPT Performance Model

CPU and Page Mode Memory

ADEPT Schematic

This is the ADEPT schematic of the CPU with the page mode memory model.



Memory Access Traces

- Three traces were analyzed:
  - ❍ Uniform access
  - ❍ Random access
  - ❍ DAXPY algorithm access
- Trace format:
  - ❍ Number of CPU instructions
  - ❍ Number of words accessed
  - ❍ Memory address

Uniform Access - a linear addressing of memory by single words with one CPU instruction per word

```
1 1 1000
1 1 1001
1 1 1002
1 1 1003
1 1 1004
1 1 1005
1 1 1006
1 1 1007
1 1 1008
1 1 1009
```

Random Access - a random addressing of memory for 1,2,4, or 8 words with 1-4 CPU instructions per word

```
2 2 63443
3 4 4373
4 8 31344
3 4 59607
4 8 23048
2 2 61114
3 4 42889
4 8 1380
4 8 33567
3 4 13239
```

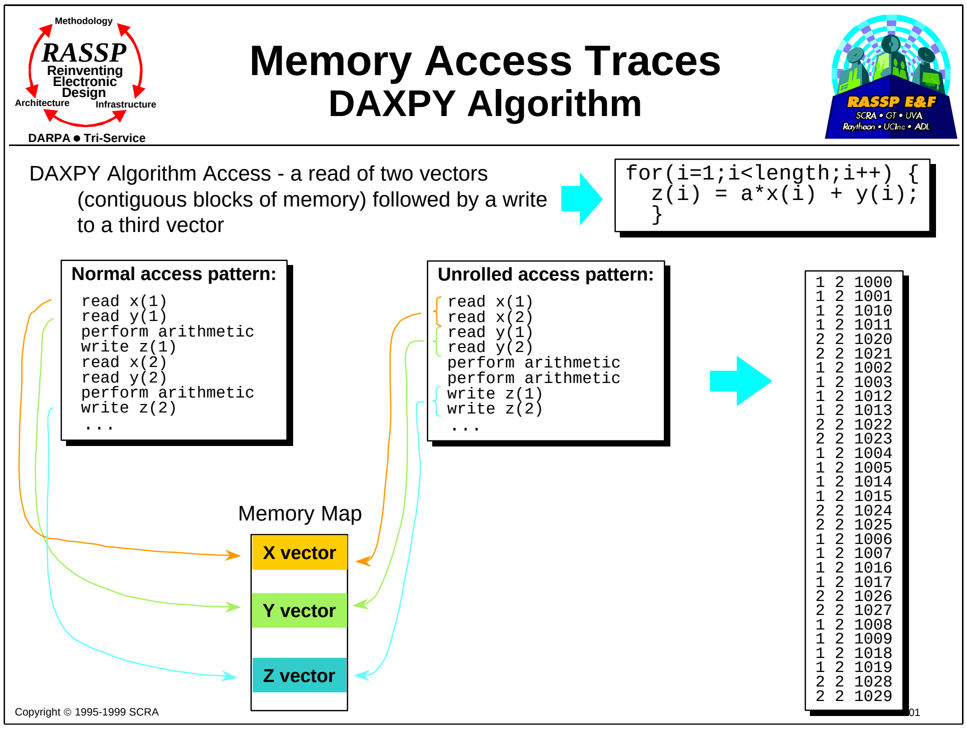
Three traces were run through the two memory system models, a simple uniform access, a random access, and a DAXPY algorithm access with loop unrolling. The traces were in the following format:

<number of instructions> <number of words> <memory address>

where number of instructions is the number of CPU instruction (time 10 ns) that the CPU will delay for after the access is granted, number of words is the number (times the access time) that the memory will delay in returning the access, and memory address is just that.

The uniform access is a single instruction, single word access where the address starts at a specify point (1000 in this example) and increments by 1 for each successive access.

The random access is an access where the number of instruction is random uniformly distribute between 1 and 4, the number of words is random uniformly distributed over the values of 1,2,4, and 8, and the address is a uniform randomly distributed number.



Memory Access Traces
DAXPY Algorithm

DAXPY Algorithm Access - a read of two vectors (contiguous blocks of memory) followed by a write to a third vector

```
for(i=1;i<length;i++) {
  z(i) = a*x(i) + y(i);
 }
```

Normal access pattern:

```
read x(1)
read y(1)
perform arithmetic
write z(1)
read x(2)
read y(2)
perform arithmetic
write z(2)
 ...
```

Unrolled access pattern:

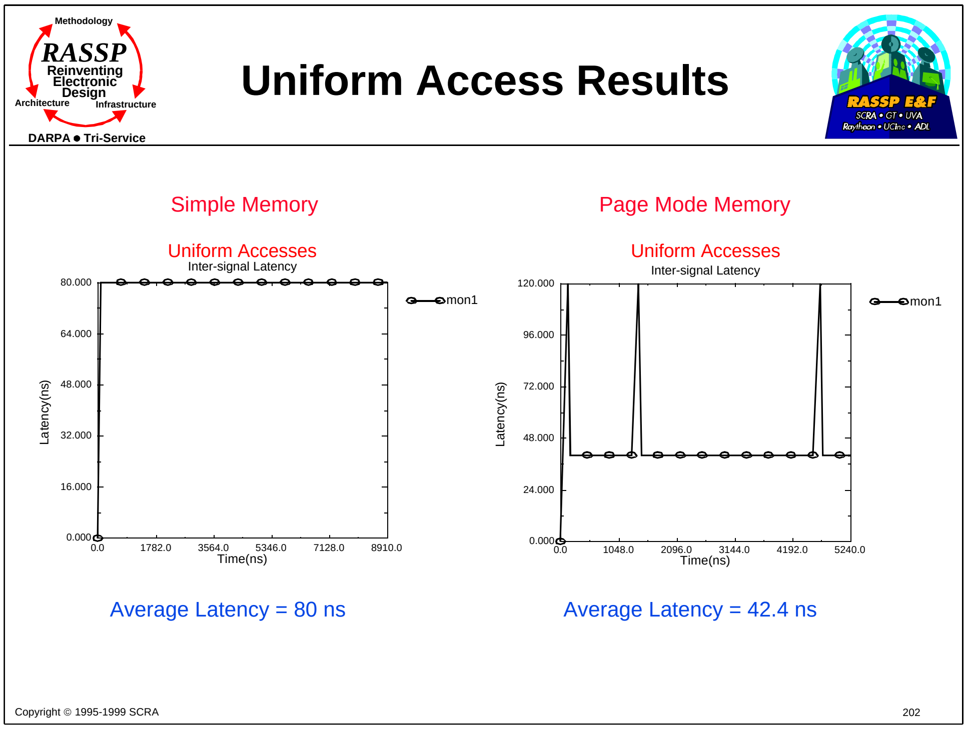
```
read x(1)
read x(2)
read y(1)
read y(2)
perform arithmetic
perform arithmetic
write z(1)
write z(2)
 ...
```

```
1 2 1000
1 2 1001
1 2 1010
1 2 1011
2 2 1020
2 2 1021
1 2 1002
1 2 1003
1 2 1012
1 2 1013
2 2 1022
2 2 1023
1 2 1004
1 2 1005
1 2 1014
1 2 1015
2 2 1024
2 2 1025
1 2 1006
1 2 1007
1 2 1016
1 2 1017
2 2 1026
2 2 1027
1 2 1008
1 2 1009
1 2 1018
1 2 1019
2 2 1028
2 2 1029
```

The DAXPY algorithm access is the simulation of the accesses that would happen if the CPU was running the algorithm to add two vectors (one times a constant), resulting in a third vector as shown. The vectors are stored in contiguous areas of memory as arrays that are typically on different memory pages.

If the DAXPY algorithm is executed in its native form, it will result in the pattern: read first X value, read first Y value, write first Z value, read second X value, etc. The problem with this is that if the vectors are indeed on different pages, each memory access will result in a page miss.

One solution to this is to “unroll” the loop so as to group accesses to the same page together. For example, in a twice unrolled case (loop unrolling factor of 2) the access pattern would be: read first X (and store in register) read second X, read first Y, read second Y, perform two multiply/adds, write first Z, write second Z. In this case, the first read or write would be a page miss, and the second would be a page hit. Obviously, the ideal would be to unroll the loop many times, but in reality, the amount of unrolling that can be done is limited by the size of the processor’s register file.



Uniform Access Results

Simple Memory

Page Mode Memory

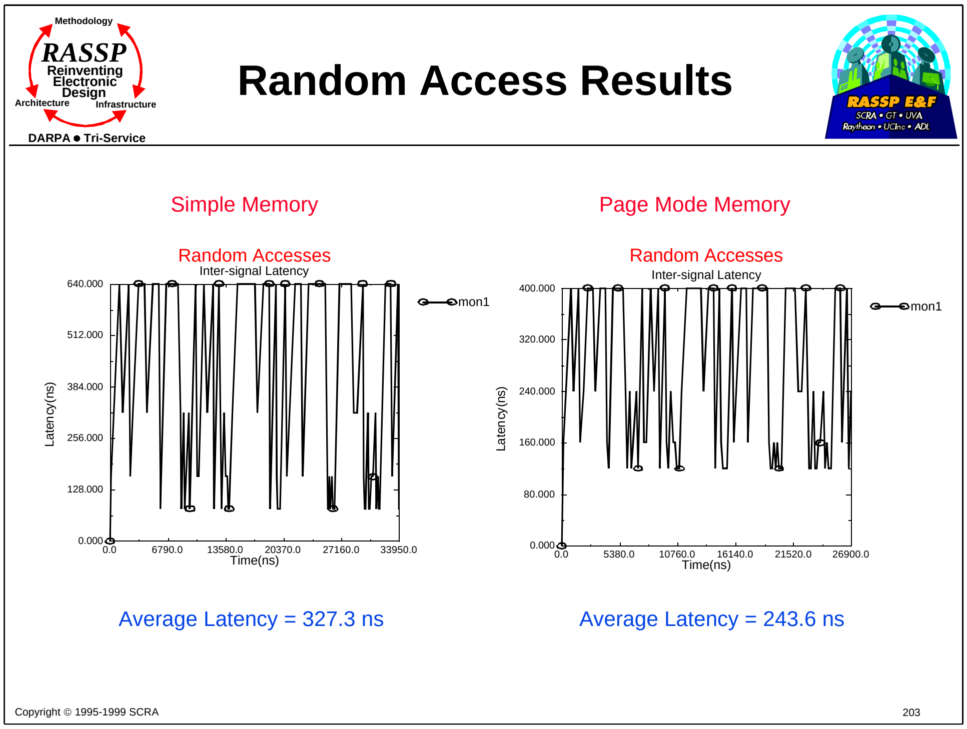
Uniform Accesses

Uniform Accesses

Average Latency = 80 ns

Average Latency = 42.4 ns

Here are the results for the uniform access trace for both the simple memory and the page mode DRAM. The simple memory has a uniform access time of 80 ns for each request (of one word size). The page mode DRAM has an initial access time of 120 ns, but then subsequent accesses have times of only 40 ns until the address jumps to the next page. Note that the pages in this example are 64 words long and the addresses start in the middle of a page, that’s why the second miss comes earlier than the third.



Random Access Results

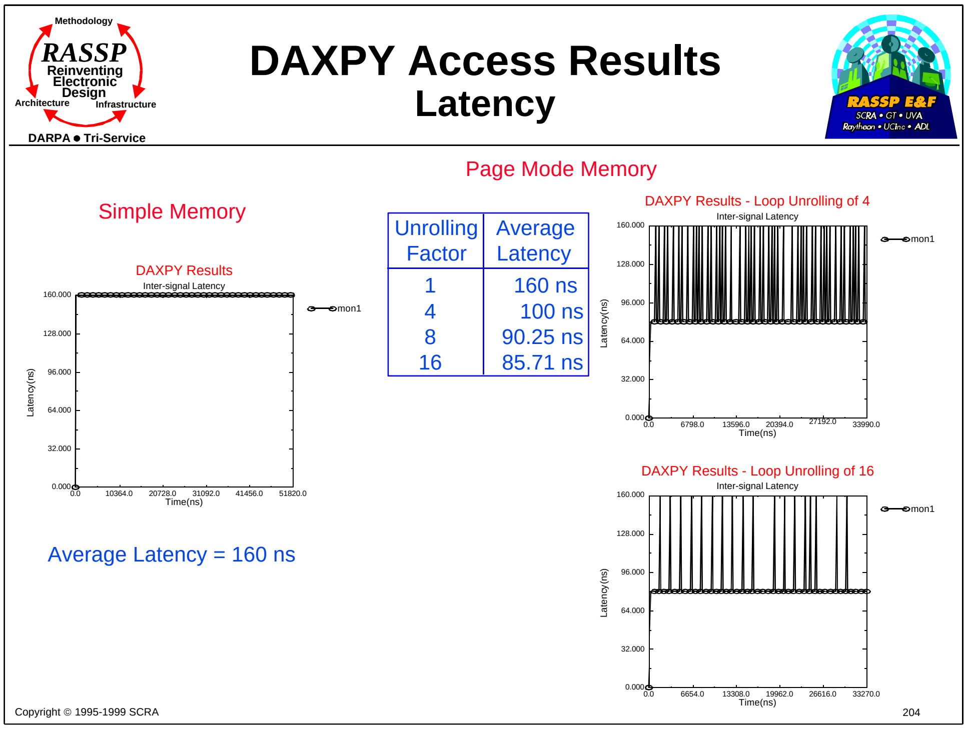
Simple Memory

Page Mode Memory

Average Latency = 327.3 ns

Average Latency = 243.6 ns

These are the results for the random access traces. The page mode DRAM is somewhat better than the simple memory here in spite of the fact that the addresses are random because many of the accesses are for multiple words and the page mode DRAM has a lower overall access time for them.



DAXPY Access Results
Latency

Page Mode Memory

Simple Memory

DAXPY Results

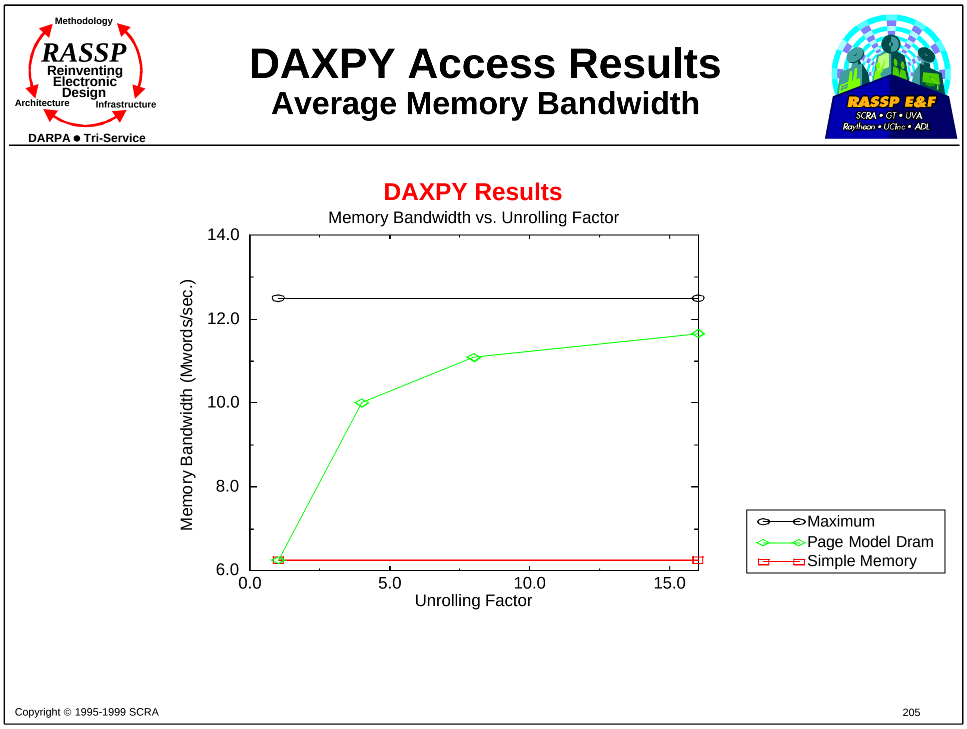
Average Latency = 160 ns

Unrolling Factor	Average Latency
1	160 ns
4	100 ns
8	90.25 ns
16	85.71 ns

DAXPY Results - Loop Unrolling of 4

DAXPY Results - Loop Unrolling of 16

Here are the results for the DAXPY accesses. The time for the simple memory is fixed because the access size is fixed. However, for the page mode DRAM, the results vary with the unrolling factor - more unrolling, lower average latency as the page misses are amortized over more page hits.



DAXPY Access Results

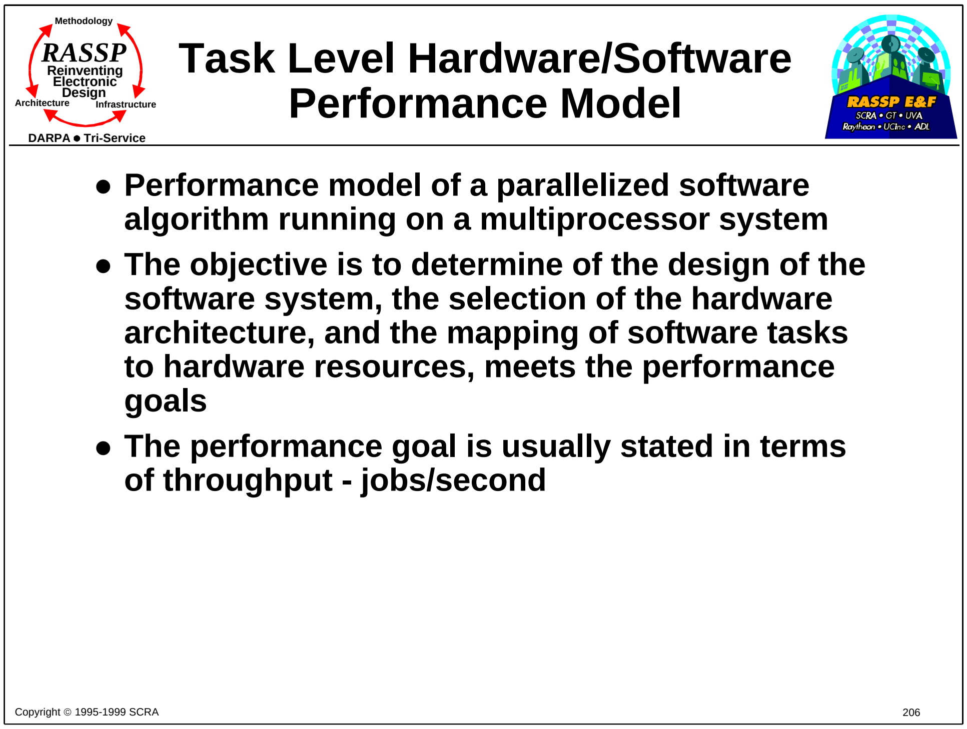
Average Memory Bandwidth

DAXPY Results

Memory Bandwidth vs. Unrolling Factor

205

Here are the results graphed as memory bandwidth (1/average latency). Note that as the unrolling factor goes up, the average latency for the page mode DRAM approaches the theoretical maximum (1/page hit time).



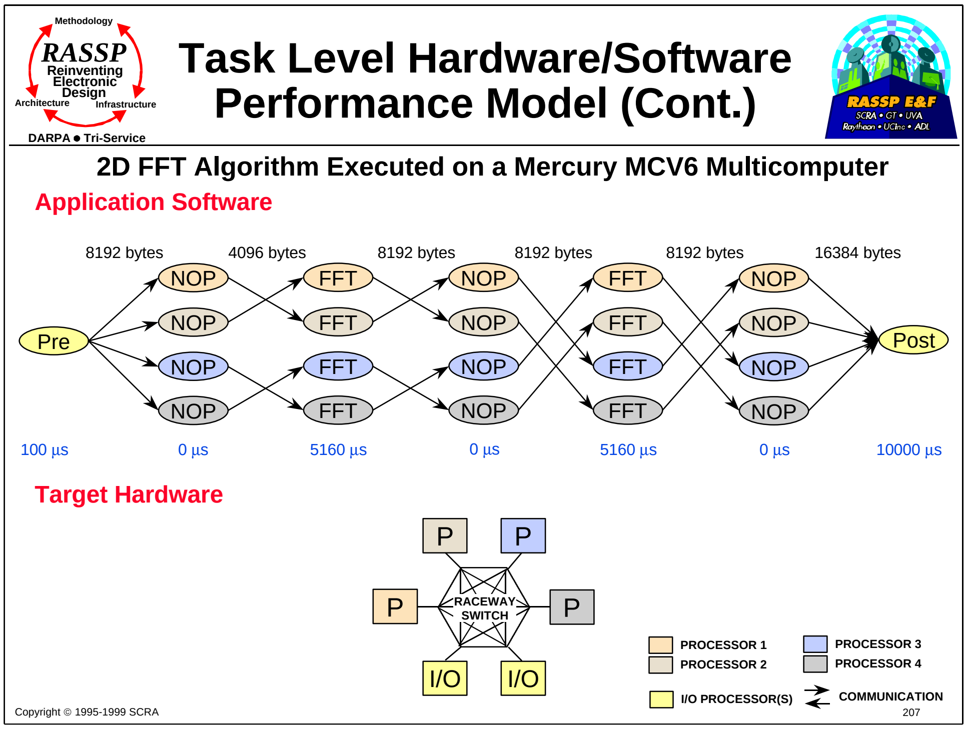
Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

Task Level Hardware/Software Performance Model

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- Performance model of a parallelized software algorithm running on a multiprocessor system
- The objective is to determine of the design of the software system, the selection of the hardware architecture, and the mapping of software tasks to hardware resources, meets the performance goals
- The performance goal is usually stated in terms of throughput - jobs/second

This section describes an example of a hardware/software performance model constructed using the ATL model elements.



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

Task Level Hardware/Software Performance Model (Cont.)

RASSP E&F

SCRA • GT • UVA

Raytheon • UCinc • ADL

2D FFT Algorithm Executed on a Mercury MCV6 Multicomputer

Application Software

Target Hardware

▨ PROCESSOR 1

▨ PROCESSOR 2

▨ PROCESSOR 3

▨ PROCESSOR 4

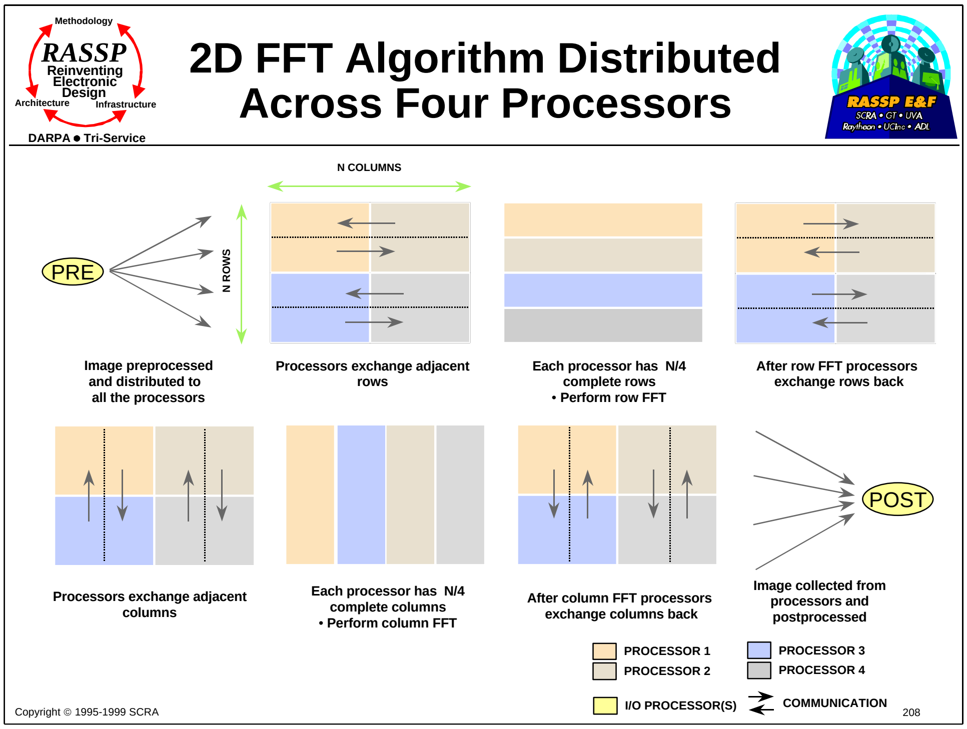
▨ I/O PROCESSOR(S)

⇄ COMMUNICATION

207

The upper part of the figure shows the overall structure of the software algorithm in terms of tasks, how long they require for computation (on the bottom in blue), and communications between them and the amounts of communication (in black above). The algorithm is a 2D Fast Fourier Transform (FFT). The NOPs in the algorithm are simply place holders to make the figure more clear. For example, after receiving the initial data from the pre-processing task, all of the processors, without doing any computation, exchange data with each other to perform the row FFT. This is shown in more detail on the next page.

The lower part of the figure show the hardware architecture. A 4 processor Mercury Race Multicomputer (called an MCV6), with either one or two I/O processors.



DARPA ● Tri-Service

2D FFT Algorithm Distributed Across Four Processors

Image preprocessed and distributed to all the processors

Processors exchange adjacent rows

Each processor has N/4 complete rows
• Perform row FFT

After row FFT processors exchange rows back

Processors exchange adjacent columns

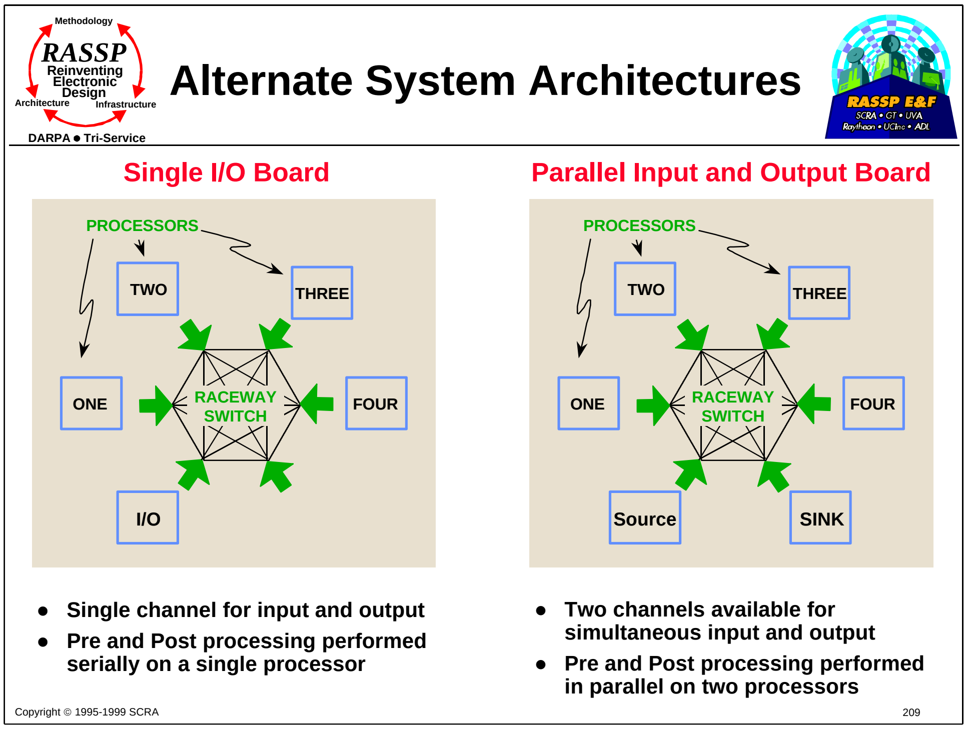
Each processor has N/4 complete columns
• Perform column FFT

After column FFT processors exchange columns back

Image collected from processors and postprocessed

🟧 PROCESSOR 1	🟦 PROCESSOR 3
🟫 PROCESSOR 2	⬜ PROCESSOR 4
🟨 I/O PROCESSOR(S)	⇄ COMMUNICATION

This is more detail on how the image data is allocated to the processors and how it is exchanged during the processing of the algorithm.



Alternate System Architectures

Single I/O Board

- Single channel for input and output
- Pre and Post processing performed serially on a single processor

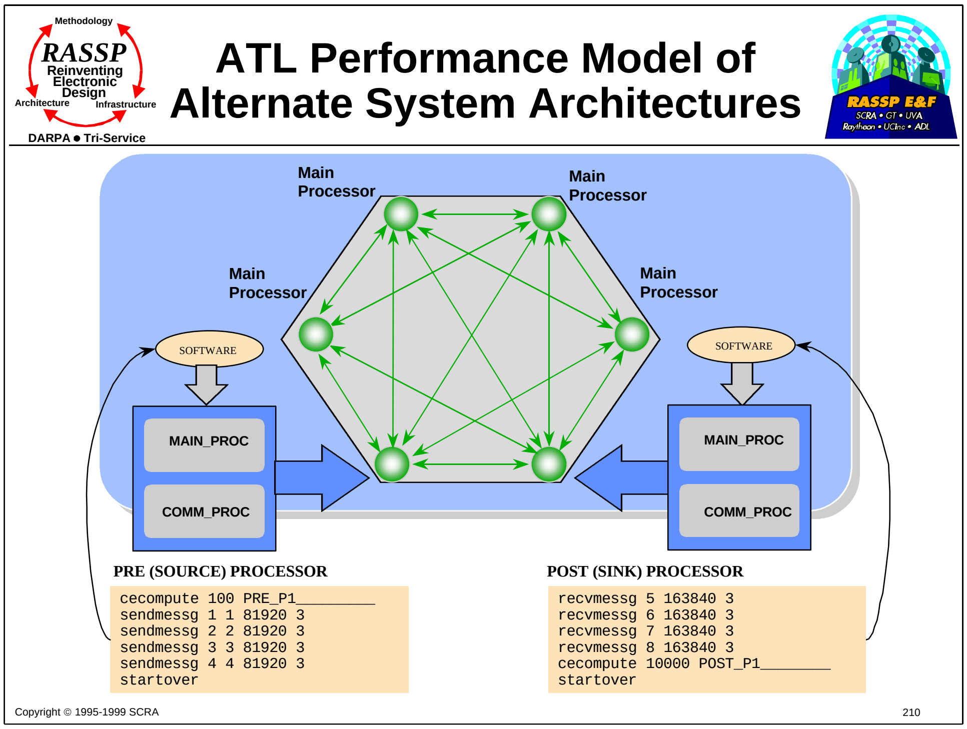
Parallel Input and Output Board

- Two channels available for simultaneous input and output
- Pre and Post processing performed in parallel on two processors

These are the two alternate systems architectures that are investigated using the performance model. Both architectures have 4 processors and a Raceway crossbar switch, but the first architecture has a single I/O board which must perform both the pre and post-processing tasks and sending and receiving images to/from the other processors must be serialized.

In the second architecture, there is a separate source and sink processor to perform the pre and post-processing task respectively, and sending and receiving images to/from the 4 processors can occur in parallel.

Note that in the ATL performance model, regular processing elements (PEs) are used to model the I/O, Source and Sink processors.



ATL Performance Model of Alternate System Architectures

Main Processor

Main Processor

Main Processor

Main Processor

SOFTWARE

MAIN_PROC

COMM_PROC

SOFTWARE

MAIN_PROC

COMM_PROC

PRE (SOURCE) PROCESSOR

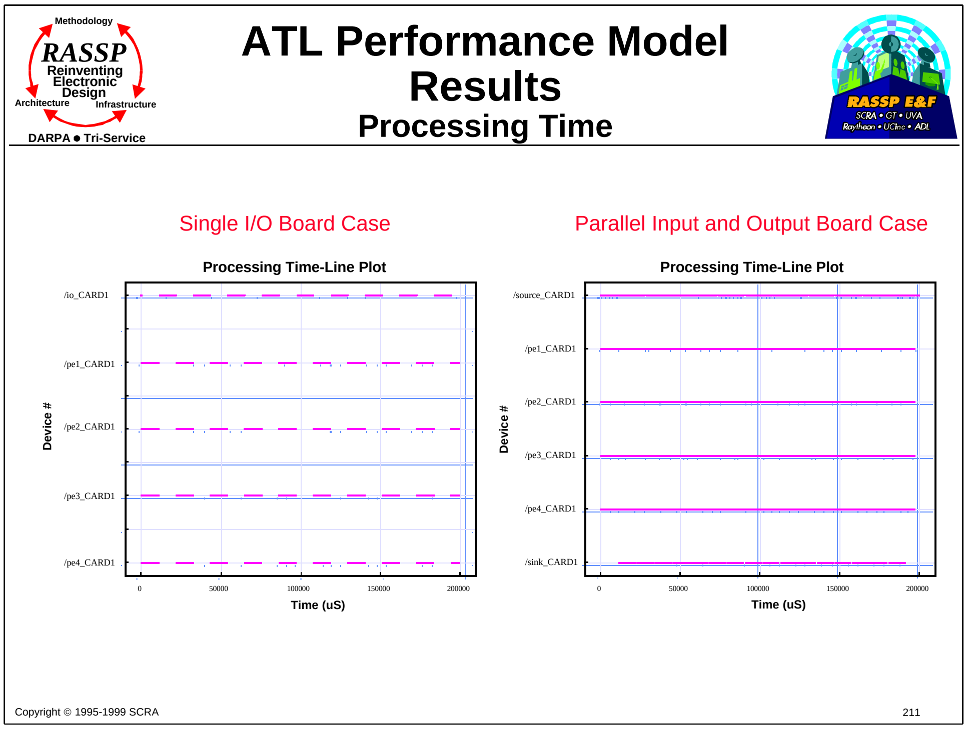
```
cecompute 100 PRE_P1_________
sendmessg 1 1 81920 3
sendmessg 2 2 81920 3
sendmessg 3 3 81920 3
sendmessg 4 4 81920 3
startover
```

POST (SINK) PROCESSOR

```
recvmessg 5 163840 3
recvmessg 6 163840 3
recvmessg 7 163840 3
recvmessg 8 163840 3
cecompute 10000 POST_P1________
startover
```

This slide shows more detail on the ATL performance model and how the PE modules (for the Source and Sink modules) read their programs out of a file.

Notice that the programs end in a “startover” command which makes them run the program in an endless loop. This way, the performance model can be simulated for some fixed amount of time and the number of loops which the model executes can be observed as a performance measure.



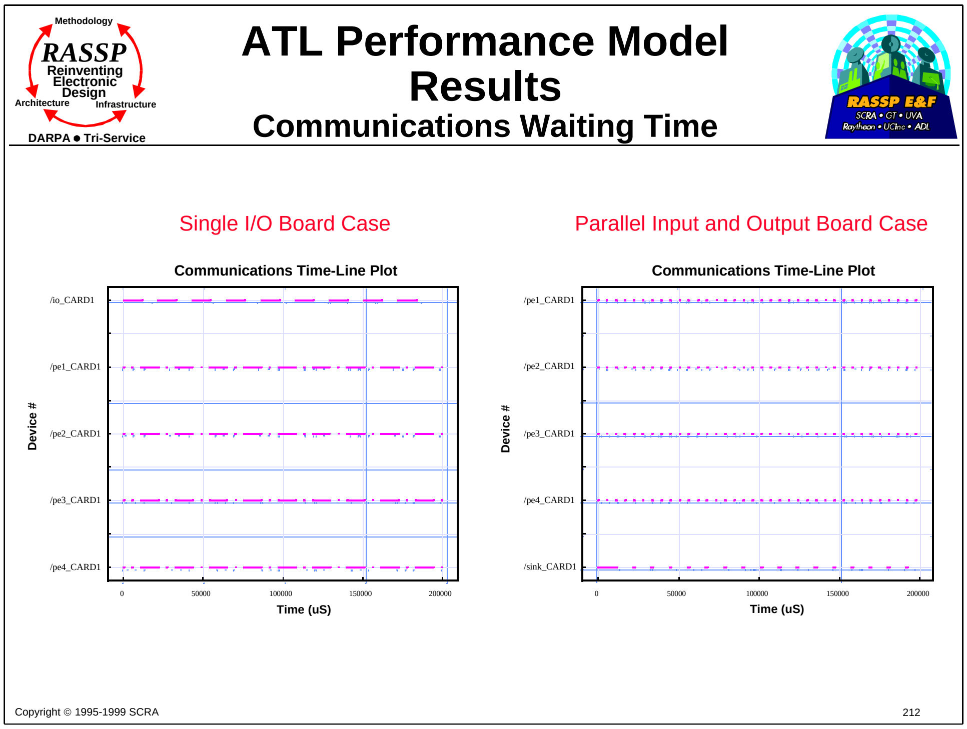
ATL Performance Model Results

Processing Time

Single I/O Board Case

Parallel Input and Output Board Case

Here are the results of the performance model from the STL time line tool. These graphs show the compute times for the modules. Notice that the second architectural alternative (with the Source and Sink processors) has much better throughput in terms of the number of loop iterations (> 20) than the first architectural alternative (<11).



Methodology

RASSP Reinventing Electronic Design

Architecture Infrastructure

DARPA ● Tri-Service

ATL Performance Model Results

Communications Waiting Time

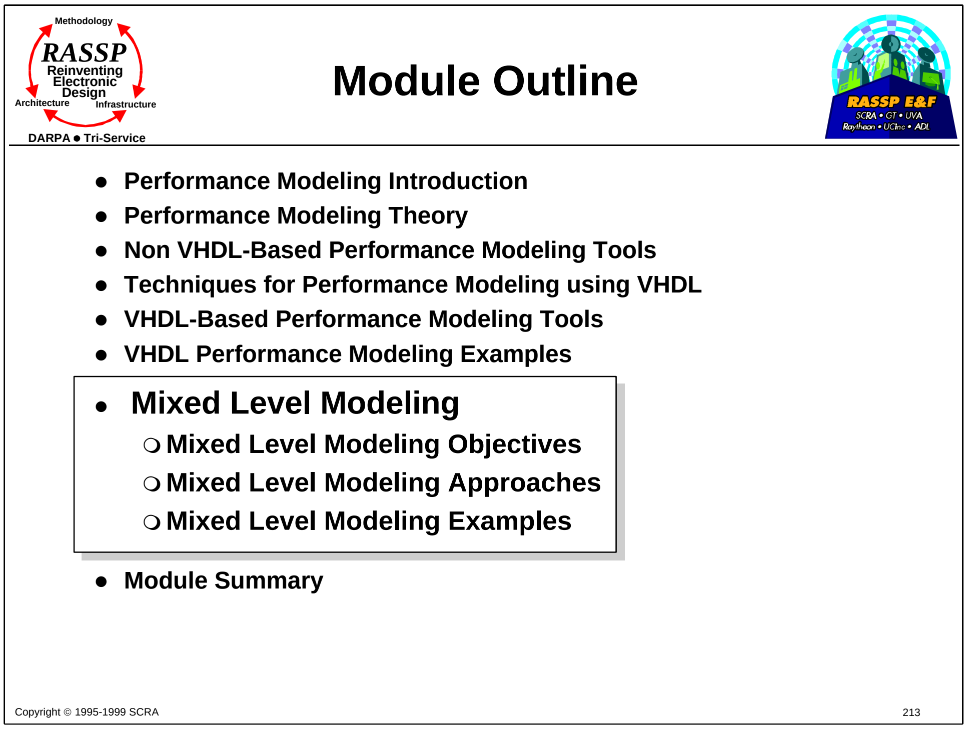
RASSP E&F SCRA • GT • UVA Raytheon • UCinc • ADL

Single I/O Board Case

Parallel Input and Output Board Case

212

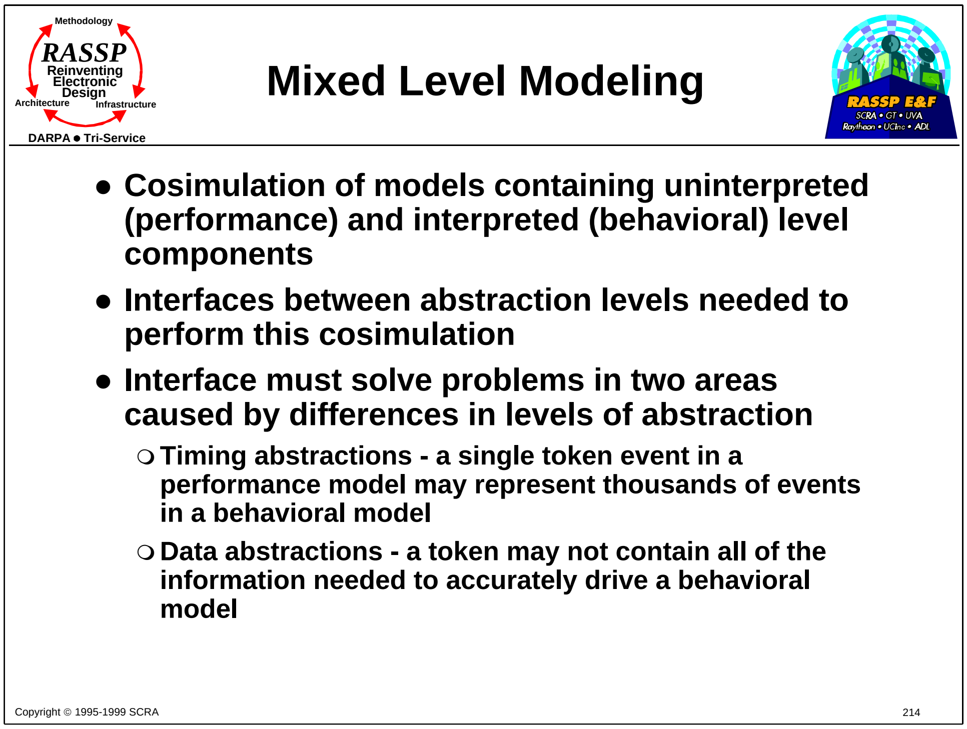
This is an activity time line plot of the communications (including waiting time) in the two alternative architectures. Note that the second architecture spends a great deal less time communicating or waiting for communications resulting in the higher throughput.


Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Module Outline

- Performance Modeling Introduction
- Performance Modeling Theory
- Non VHDL-Based Performance Modeling Tools
- Techniques for Performance Modeling using VHDL
- VHDL-Based Performance Modeling Tools
- VHDL Performance Modeling Examples
- **Mixed Level Modeling**
  - ❍ Mixed Level Modeling Objectives
  - ❍ Mixed Level Modeling Approaches
  - ❍ Mixed Level Modeling Examples
- Module Summary

Module Outline



Mixed Level Modeling

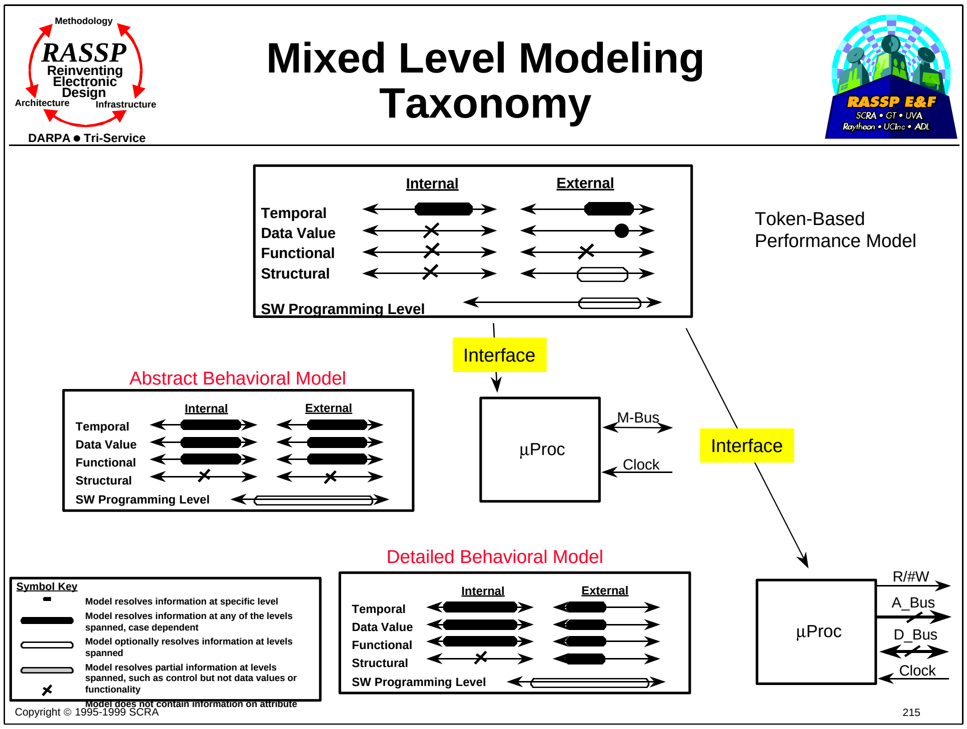
- Cosimulation of models containing uninterpreted (performance) and interpreted (behavioral) level components
- Interfaces between abstraction levels needed to perform this cosimulation
- Interface must solve problems in two areas caused by differences in levels of abstraction
  - ❍ Timing abstractions - a single token event in a performance model may represent thousands of events in a behavioral model
  - ❍ Data abstractions - a token may not contain all of the information needed to accurately drive a behavioral model

214

This section explains the concept of mixed level modeling, the cosimulation of performance and behavioral models, and how it is implemented in ADEPT. ADEPT was chosen as the example for this section as the theory and implementation of mixed level modeling is more advanced in ADEPT than other performance modeling environments as of this date. More information on this subject can be obtained from the UVa Center For Semicustom Integrated Systems web page:

http://csis.ee.virginia.edu/

eArchitect (through PML) includes the capability for constructing mixed level models, but the facilities for developing methods to resolve timing and data abstraction are less well developed and require more user interaction.



Mixed Level Modeling Taxonomy

DARPA ● Tri-Service

Token-Based
Performance Model

Internal External
Temporal
Data Value
Functional
Structural
SW Programming Level

Interface

Abstract Behavioral Model

Internal External
Temporal
Data Value
Functional
Structural
SW Programming Level

µProc
M-Bus
Clock

Interface

Detailed Behavioral Model

Internal External
Temporal
Data Value
Functional
Structural
SW Programming Level

µProc
R/#W
A_Bus
D_Bus
Clock

Symbol Key

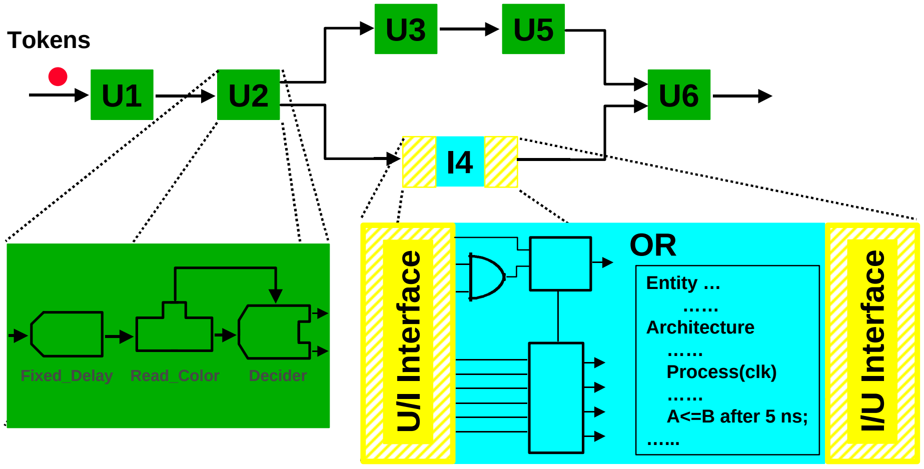
- Model resolves information at specific level
- Model resolves information at any of the levels spanned, case dependent
- Model optionally resolves information at levels spanned
- Model resolves partial information at levels spanned, such as control but not data values or functionality
- Model does not contain information on attribute

This figure illustrates where the components of mixed level models lie in the RASSP taxonomy. It is clear from this description that token-based performance models have abstract timing and little or no data values (and data transformations - function) and that behavioral models have more detailed timing and data values. Therefore, it is easy to see that an interface(s) is needed between them when they are simulated in the same model.

Methodology
RASSP
Reinventing Electronic Design
Architecture Infrastructure
DARPA ● Tri-Service

General Mixed Level Model Structure

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

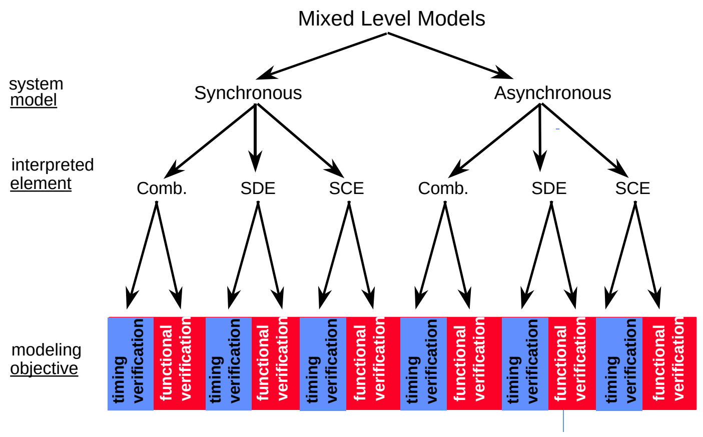


```
Entity ...
    ……
Architecture
    ……
    Process(clk)
    ……
    A<=B after 5 ns;
……
```

[UVA] 216

This is the general structure of a mixed level model. Here a single component in the performance model has been replaced with a behavioral component. Interfaces are required on its input and output to resolve the tokens to values and values to token conversion problem.

Mixed Level Modeling Interface Taxonomy



Mixed Level Models

system model

Synchronous

Asynchronous

interpreted element

Comb.

SDE

SCE

Comb.

SDE

SCE

modeling objective

timing verification

functional verification

timing verification

functional verification

timing verification

functional verification

timing verification

functional verification

timing verification

functional verification

timing verification

functional verification

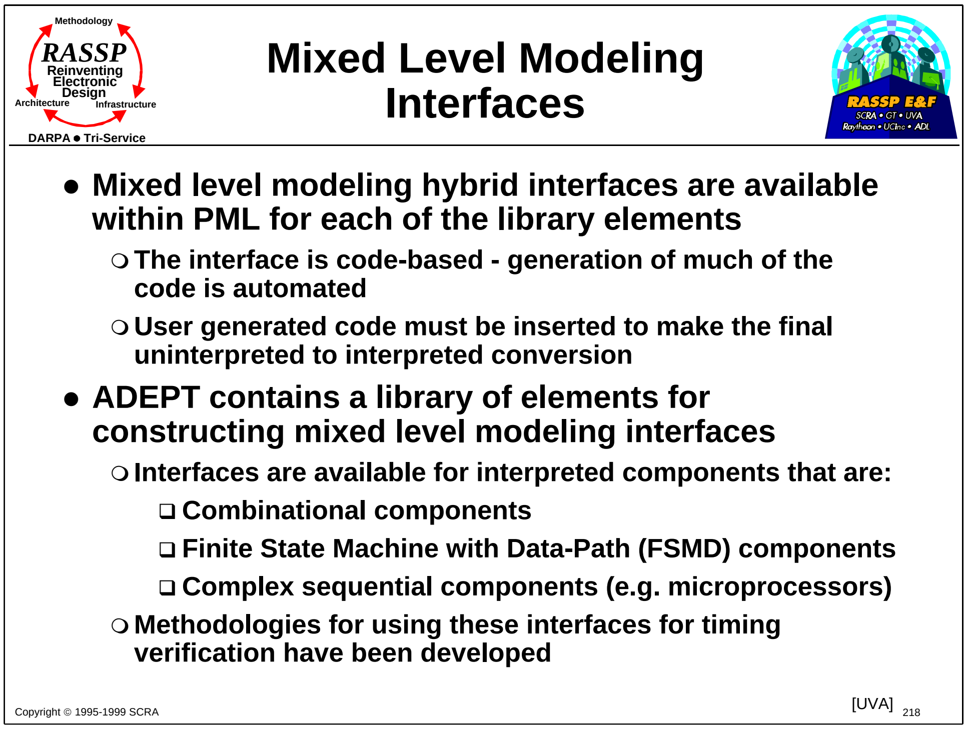
🟦 Interfaces and methodology available within ADEPT

🟥 Interfaces can be constructed within ADEPT

SDE - Sequential Dataflow Element
SCE - Sequential Control Element

[UVA] 217

This figure shows the taxonomy of hybrid models that was developed jointly between UVa and Honeywell Technology Center (their version is slightly different) to classify the solutions. Note that most work thus far has concentrated on the problem of timing verification.



Mixed Level Modeling Interfaces

- Mixed level modeling hybrid interfaces are available within PML for each of the library elements
  - ❍ The interface is code-based - generation of much of the code is automated
  - ❍ User generated code must be inserted to make the final uninterpreted to interpreted conversion
- ADEPT contains a library of elements for constructing mixed level modeling interfaces
  - ❍ Interfaces are available for interpreted components that are:
    - ❑ Combinational components
    - ❑ Finite State Machine with Data-Path (FSMD) components
    - ❑ Complex sequential components (e.g. microprocessors)
  - ❍ Methodologies for using these interfaces for timing verification have been developed

[UVA] 218

As stated previously, PML has a mixed level modeling interface capability, but it is mainly code based and the user must supply the VHDL code that performs the tokens to values and values to tokens conversion.

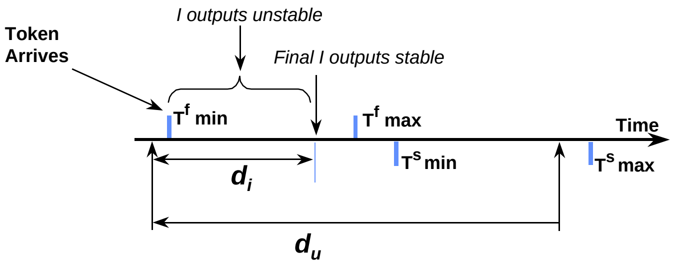
ADEPT has a library of standard “hybrid” elements out of which mixed level modeling interfaces can be developed. For some classes of models in the taxonomy, the interface can be generated with no user coding, or new modules required. In other cases, some generation of application specific modules by the user is required.

RASSP Reinventing Electronic Design
Methodology · Architecture · Infrastructure
DARPA ● Tri-Service

Mixed Level Interface for Combinational Interpreted Elements

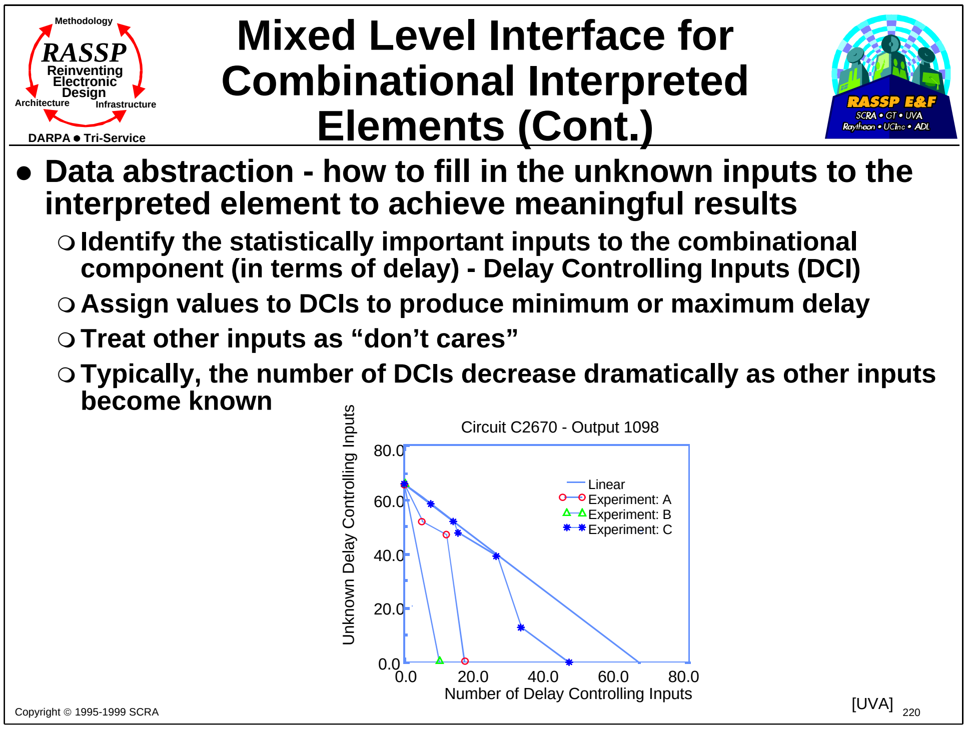
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

- **Timing abstraction - settling-time problem - how to determine the correct time to release token(s) from the hybrid element**
  - ❍ **Solution: *time expansion* technique**
    - ❑ **Execute the hybrid element in the fast time domain**
    - ❑ **Execute the remaining performance model in the slow time domain**



[UVA] 219

This figure illustrates the problem of timing in mixed level models when the behavioral (or interpreted) element is combinational. A token arriving at the interface to the hybrid element, which contains the interpreted combinational component, triggers application of the new values to the inputs of the combinational element. Then after some time, the generation of the final outputs from the combinational element will trigger the release of the token from the hybrid element. The problem is the fact that the outputs of the combinational element take variable times to settle to the final value and it is difficult to determine when that has happened. The solution, called time expansion, is to run the combinational element in “fast time” which is usually 10 times faster than the performance model time scale, wait the maximum delay time of the combinational element in fast time, observe when, in fast time, the combinational element’s outputs settled to their final values, and then scale this time up to slow time and release the token at the proper time in slow time.



Mixed Level Interface for Combinational Interpreted Elements (Cont.)

- Data abstraction - how to fill in the unknown inputs to the interpreted element to achieve meaningful results
  - ❍ Identify the statistically important inputs to the combinational component (in terms of delay) - Delay Controlling Inputs (DCI)
  - ❍ Assign values to DCIs to produce minimum or maximum delay
  - ❍ Treat other inputs as “don’t cares”
  - ❍ Typically, the number of DCIs decrease dramatically as other inputs become known

Circuit C2670 - Output 1098
Unknown Delay Controlling Inputs
80.0
60.0
40.0
20.0
0.0
0.0
20.0
40.0
60.0
80.0
Number of Delay Controlling Inputs
Linear
Experiment: A
Experiment: B
Experiment: C

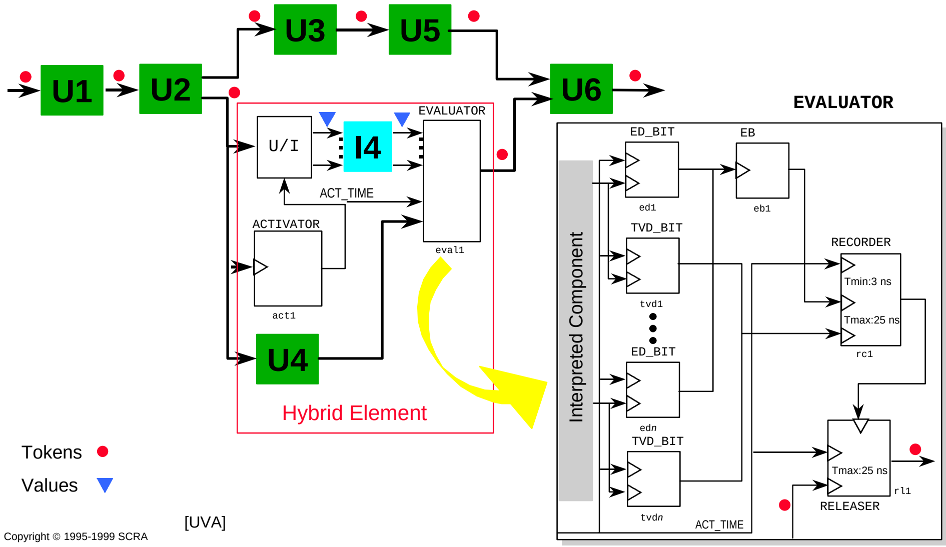
[UVA] 220

Another problem attacked in the mixed level area in ADEPT is the problem of specifying the inputs to the combinational element, that could not be derived from the incoming toke (called “unknown inputs”) in such a way as to generate meaningful results, usually either minimum or maximum delay.

A technique has been developed to determine the inputs that have the most influence on the delay of the combinational element (called DCIs) and setting them to the values that cause the best or worst case values. In theory, this is an exponentially complex problem, but the results, as shown here, have demonstrated that as a few inputs are known from the performance model, the number of DCIs drops dramatically, resulting in the problem quickly becoming tractable.

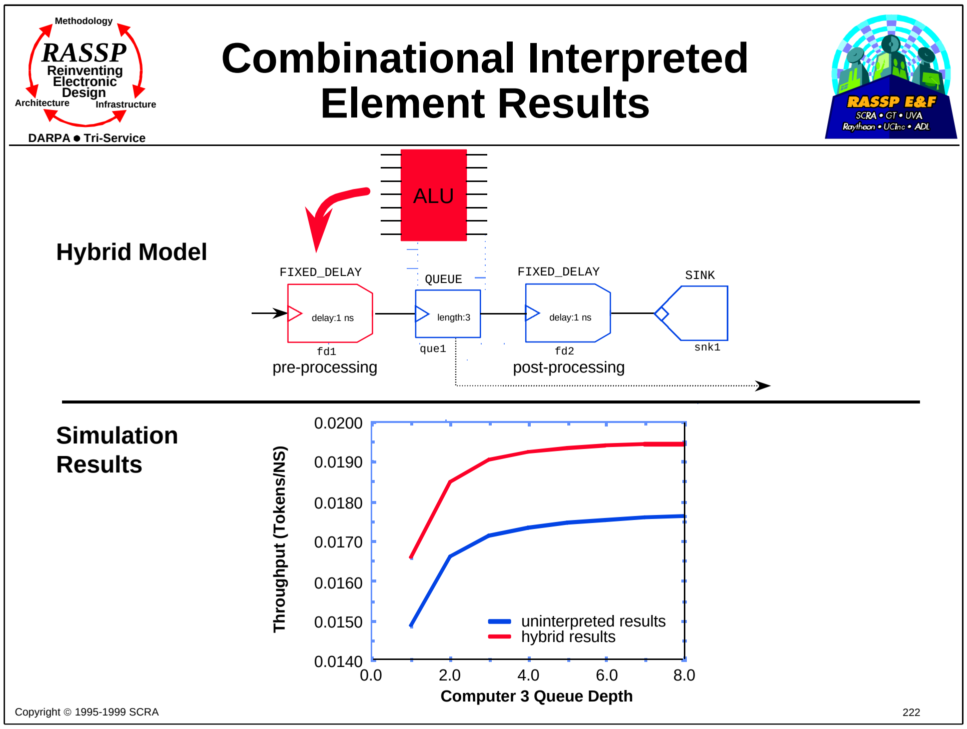
Mixed Level Interface for Combinational Interpreted Elements (Cont.)

- Interface Structure



[UVA]

This is the structure of the mixed level interface in ADEPT for combinational interpreted elements that implements time expansion. When a token arrives at the input to the hybrid element, the U/I component converts values on the token to values on the combinational element’s inputs and runs the DCI algorithm if need be. At the same time, the activator records the token arrival time and passes it to the evaluator. The evaluator waits the maximum combination delay time in fast time, measures the actual combination delay in fast time, and scales that up and releases the token at the proper time in slow time.



Methodology
RASSP
Reinventing Electronic Design
Architecture
Infrastructure
DARPA ● Tri-Service

Combinational Interpreted Element Results

RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

Hybrid Model

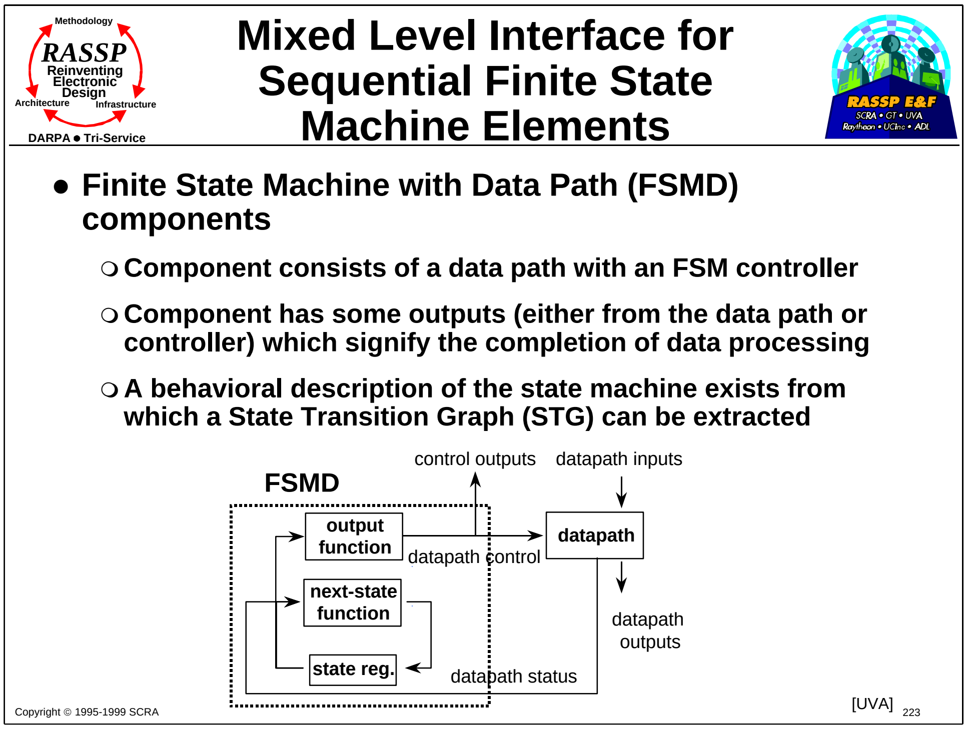
pre-processing

post-processing

Simulation Results

222

Here are some simple results from a mixed level model with a combinational element. Note that the throughput achieved by the mixed level model has the same shape as the original performance modeling results (which is good), but it is shifted as a result of having actual delay values from the behavioral component.



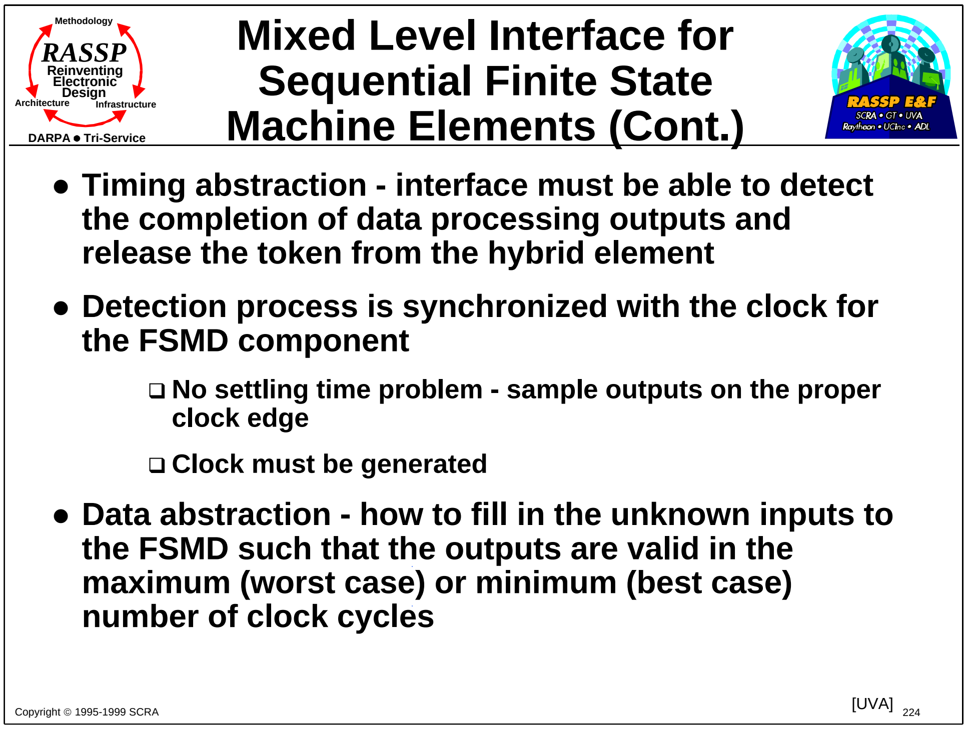
Mixed Level Interface for Sequential Finite State Machine Elements

- **Finite State Machine with Data Path (FSMD) components**
  - ❍ **Component consists of a data path with an FSM controller**
  - ❍ **Component has some outputs (either from the data path or controller) which signify the completion of data processing**
  - ❍ **A behavioral description of the state machine exists from which a State Transition Graph (STG) can be extracted**

[UVA] 223

Another area of mixed level modeling investigated in ADEPT was that of an interpreted component that was a finite state machine with datapath (FSMD). This is an interpreted component who's function can be described by a state transition graph (STG). This is important because it allows graph algorithms to be used to analyze the STG to determine maximum and minimum delay. In addition, a requirement is that there be some outputs, either from the state machine or datapath, the can be used to determine the completion of processing for a given token arrival event.

Examples of these types of elements include a dedicated FFT chip or a floating point coprocessor.



Mixed Level Interface for Sequential Finite State Machine Elements (Cont.)

- Timing abstraction - interface must be able to detect the completion of data processing outputs and release the token from the hybrid element
- Detection process is synchronized with the clock for the FSMD component
  - ❑ No settling time problem - sample outputs on the proper clock edge
  - ❑ Clock must be generated
- Data abstraction - how to fill in the unknown inputs to the FSMD such that the outputs are valid in the maximum (worst case) or minimum (best case) number of clock cycles

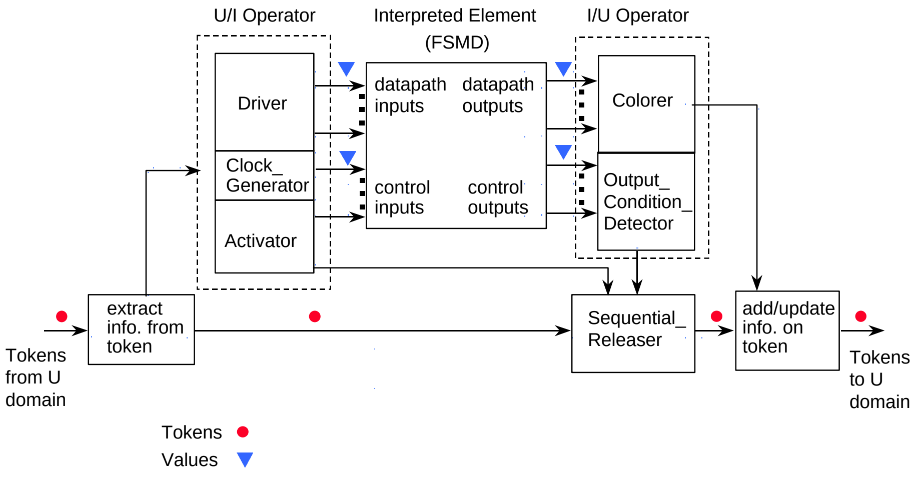
[UVA] 224

The timing abstraction problem is easier with FSMD components as there is no settling problem - everything is resolved on a clock edge. However, the clock input to the FSMD must be generated, usually by the mixed level interface elements.

The data abstraction problem is similar to the combinational element one - how to specify the unknown inputs such that minimum or maximum delay results.

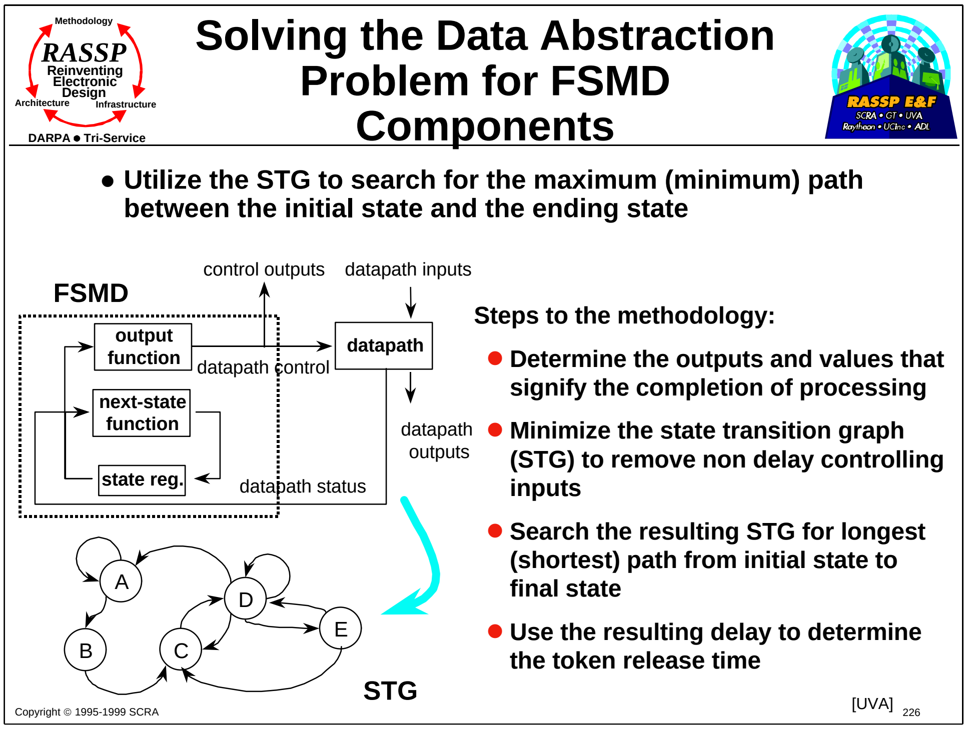
Mixed Level Interface for Sequential Finite State Machine Elements (Cont.)

- **Interface Structure**



[UVA] 225

This is the structure of the mixed level interface for FSMD components. The driver and clock generator perform the U/I function and the activator performs the same function as in the previous example. The Colorer, output_condition_detector, and sequential_releaser perform the functions of the evaluator, that is, determining when to release the token from the hybrid element after the proper delay time according to the interpreted component.



Solving the Data Abstraction Problem for FSMD Components

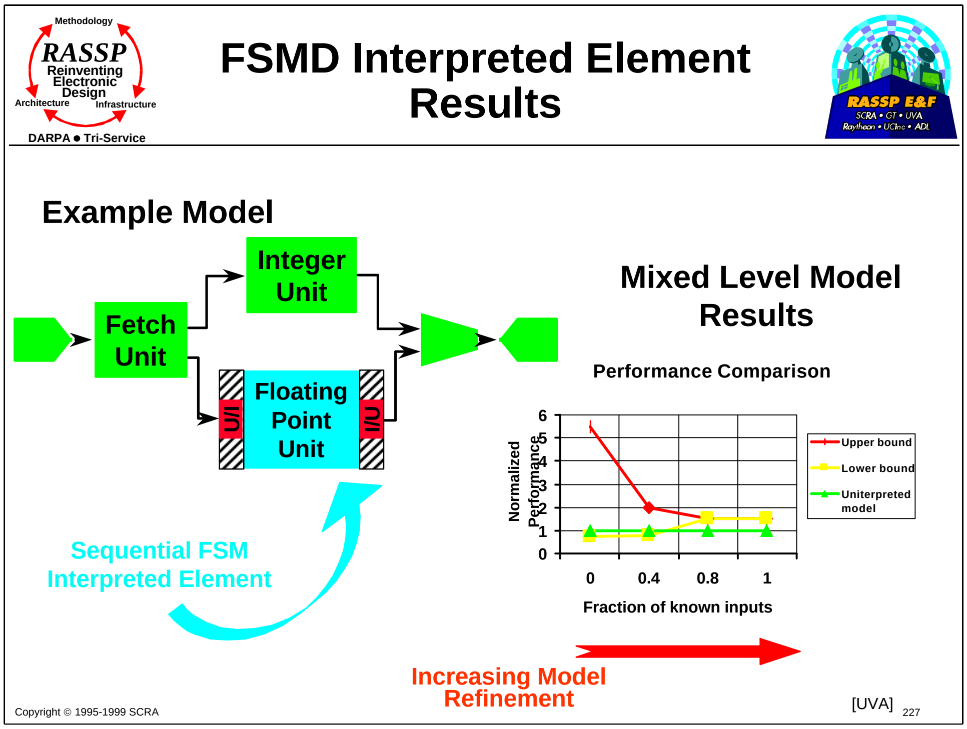
- Utilize the STG to search for the maximum (minimum) path between the initial state and the ending state

Steps to the methodology:

- Determine the outputs and values that signify the completion of processing
- Minimize the state transition graph (STG) to remove non delay controlling inputs
- Search the resulting STG for longest (shortest) path from initial state to final state
- Use the resulting delay to determine the token release time

[UVA] 226

This figure outlines the methodology used to determine the minimum or maximum delay, in terms of clock cycles, for the FSMD interpreted component using the component’s STG. First, the outputs that do not affect when the token is released are removed from the STG and the resulting STG is simplified. Next, the resulting STG is searched to find the shortest (minimum time) or longest (maximum time) path from the initial state to the final state. Finally, the inputs necessary to drive the FSMD along this path are applied to the interpreted component in the simulation.



FSMD Interpreted Element Results

Example Model

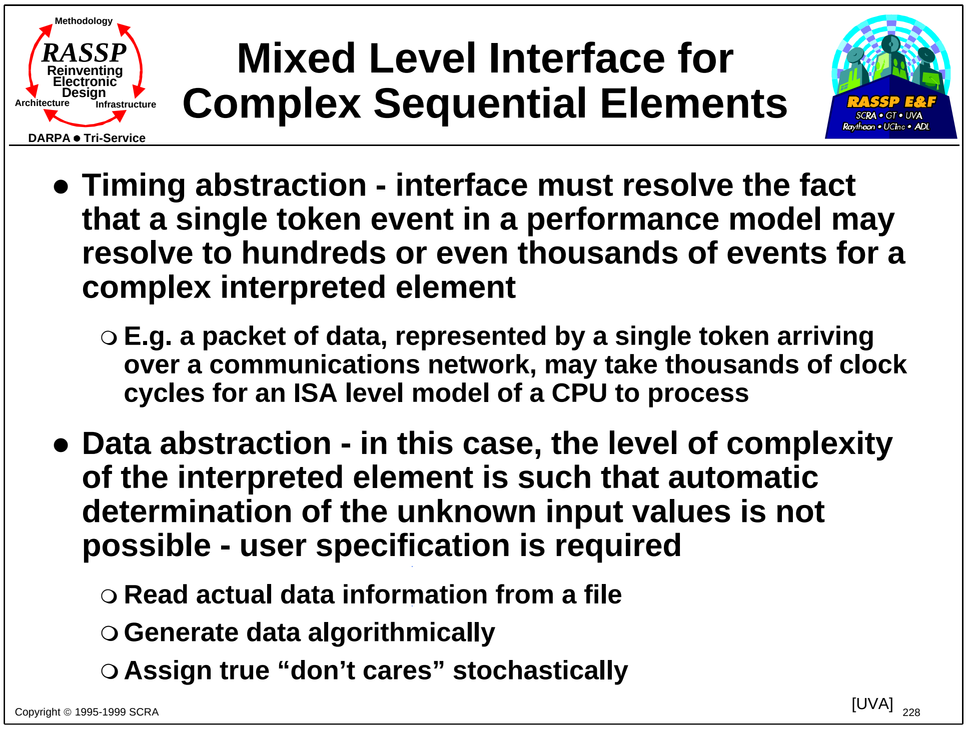
Mixed Level Model Results

Performance Comparison

Increasing Model Refinement

[UVA] 227

Here are some results from an example of applying the technique to an FSMD mixed level model. In this case, it was a performance model of a processor with a fetch unit, an integer unit and a floating point unit. The floating point unit was replaced with its interpreted (behavioral) representation. The results show how the upper and lower bounds (minimum and maximum delay) on performance can be generated for the model at various levels of refinement. As the model is refined, the fraction of inputs for which the actual values are known from the performance model increase, and the bounds get tighter and finally converge. Also notice that, as is quite typical, the initial estimate of the performance as used in the high level performance model, was inaccurate.



Methodology
RASSP
Reinventing
Electronic
Design
Architecture
Infrastructure
DARPA ● Tri-Service
RASSP E&F
SCRA • GT • UVA
Raytheon • UCinc • ADL

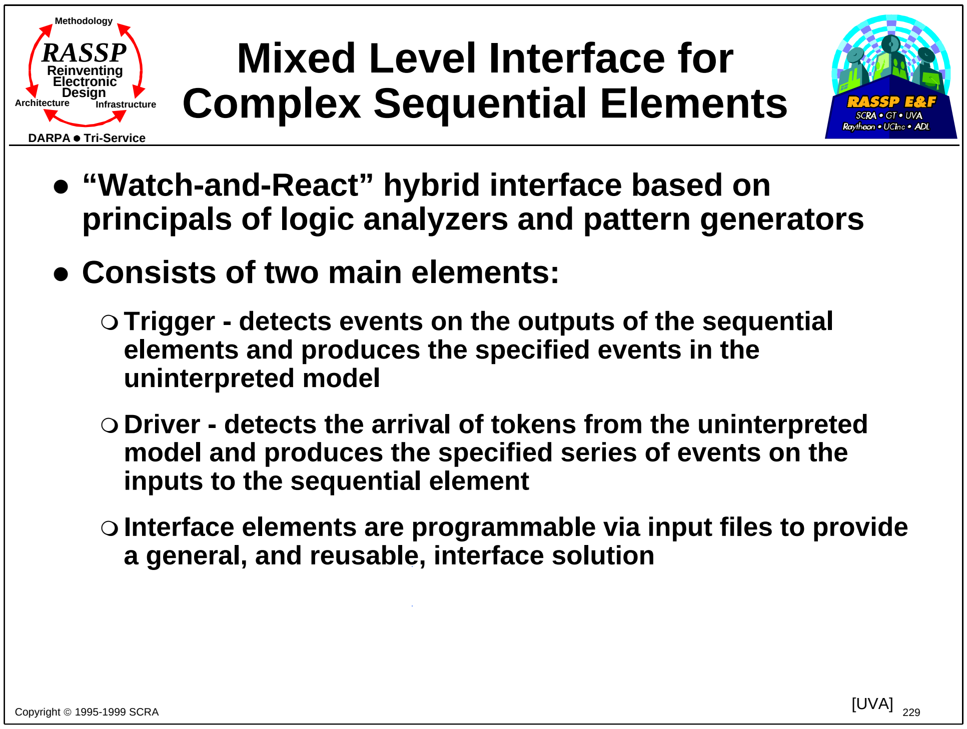
Mixed Level Interface for Complex Sequential Elements

- Timing abstraction - interface must resolve the fact that a single token event in a performance model may resolve to hundreds or even thousands of events for a complex interpreted element
  - ❍ E.g. a packet of data, represented by a single token arriving over a communications network, may take thousands of clock cycles for an ISA level model of a CPU to process
- Data abstraction - in this case, the level of complexity of the interpreted element is such that automatic determination of the unknown input values is not possible - user specification is required
  - ❍ Read actual data information from a file
  - ❍ Generate data algorithmically
  - ❍ Assign true “don’t cares” stochastically

[UVA] 228

Finally, mixed level interface elements were developed for “complex sequential elements” which are sequential elements that are too complex to describe as state machines. In this case, the interface is more ad hoc, and is targeted at solving the timing abstraction problem. The user must solve the data abstraction problem for interpreted elements such as these.

Elements that fall into this category include microprocessors, microcontrollers, and even entire computer systems.

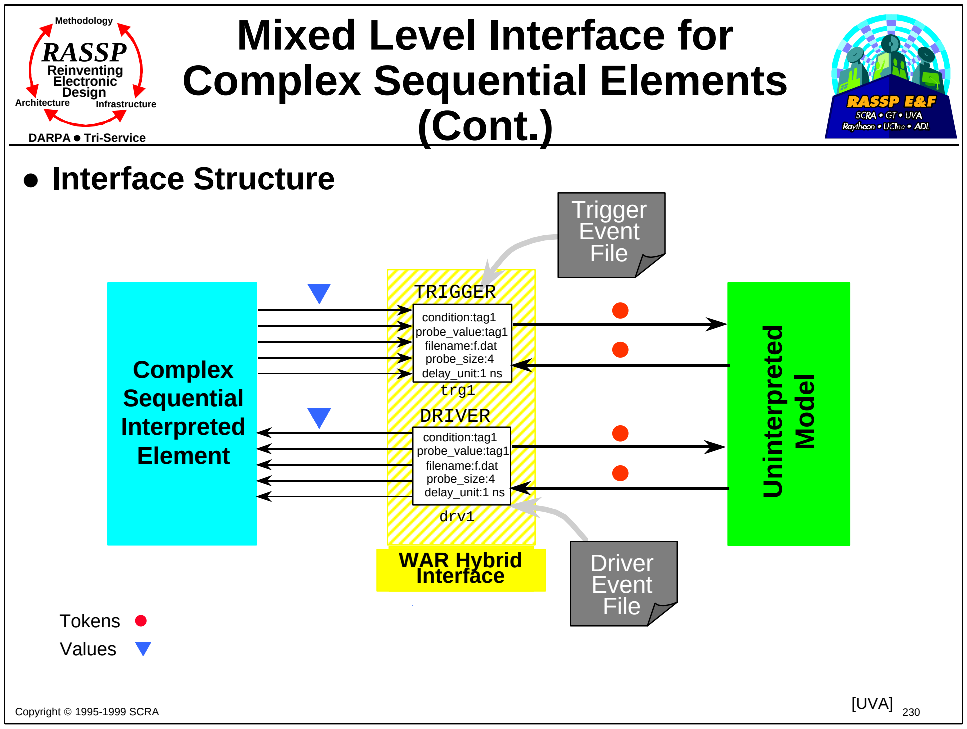


Mixed Level Interface for Complex Sequential Elements

- “Watch-and-React” hybrid interface based on principals of logic analyzers and pattern generators
- Consists of two main elements:
  - ❍ Trigger - detects events on the outputs of the sequential elements and produces the specified events in the uninterpreted model
  - ❍ Driver - detects the arrival of tokens from the uninterpreted model and produces the specified series of events on the inputs to the sequential element
  - ❍ Interface elements are programmable via input files to provide a general, and reusable, interface solution

[UVA] 229

The so called “watch and react” hybrid interface is build on the principals of logic analyzers. The interface watches the outputs of the interpreted element for certain “trigger” conditions, and when they occur, it takes the appropriate action. Likewise, when the performance model dictates that some new inputs be supplied to the interpreted component, a “program” can be executed that generates a complex set of input sequences to the interpreted component.



Mixed Level Interface for Complex Sequential Elements (Cont.)

- Interface Structure

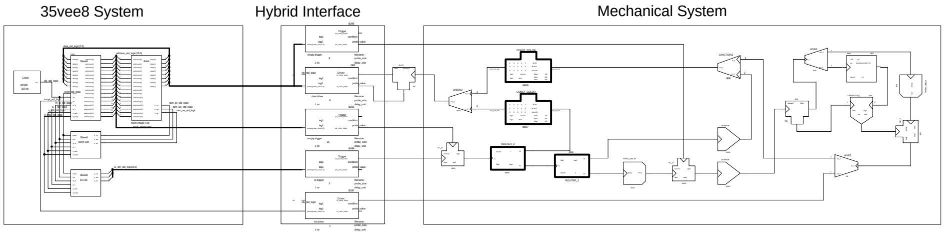
Tokens ●
Values ▼

Copyright © 1995-1999 SCRA

[UVA] 230

Here is the general structure of the watch and react interface. Both the trigger and driver element can be programmed by input files - which keeps them general in nature and avoids having the user to generate new VHDL code for a specific application.

Watch-and-React Interface Example



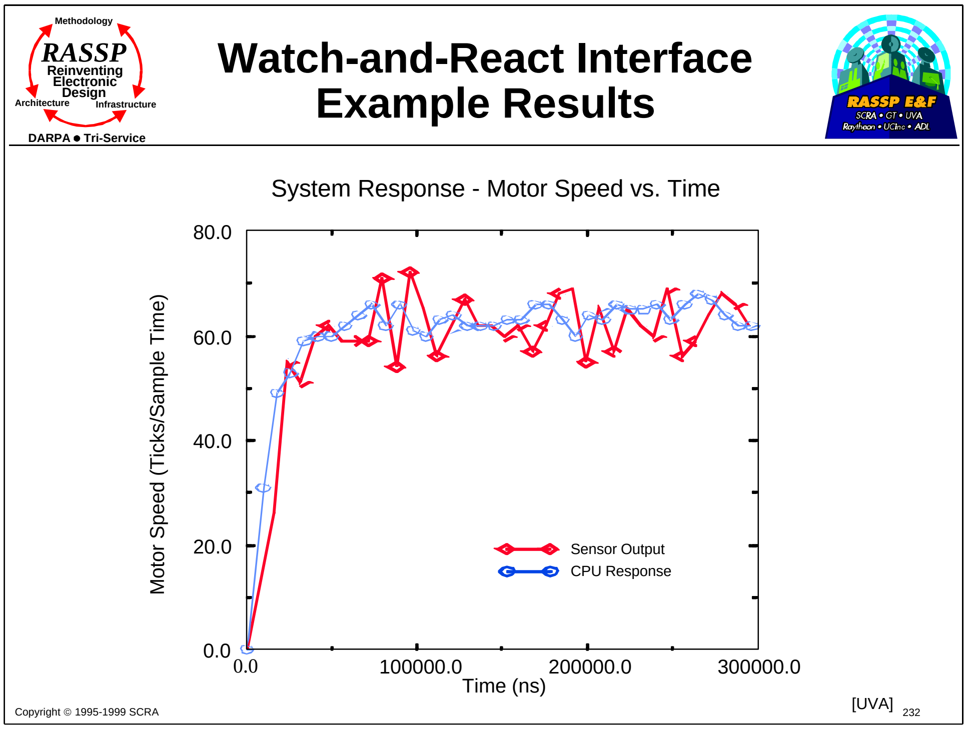
35vee8 System

Hybrid Interface

Mechanical System

[UVA] 231

Here is an example of the use of the watch and react interface. It is a motor control system in which an actual behavioral model of a microcontroller, along with its associated memory system has been inserted. The motor control system and its feedback mechanism are modeled at a system level using ADEPT modules.

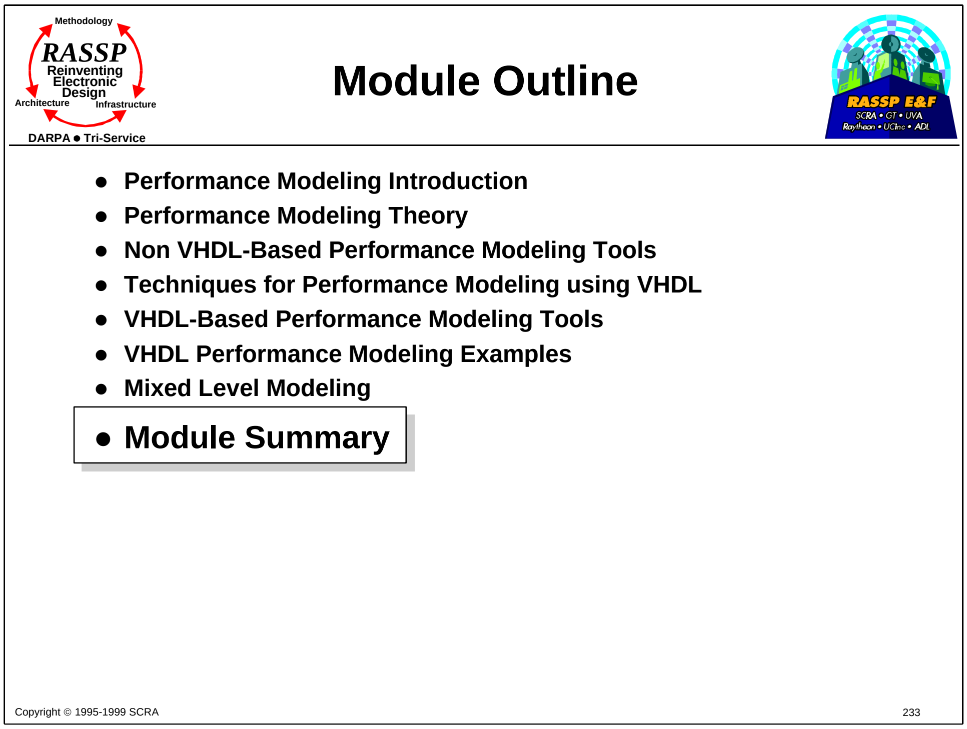


Watch-and-React Interface Example Results

System Response - Motor Speed vs. Time

[UVA] 232

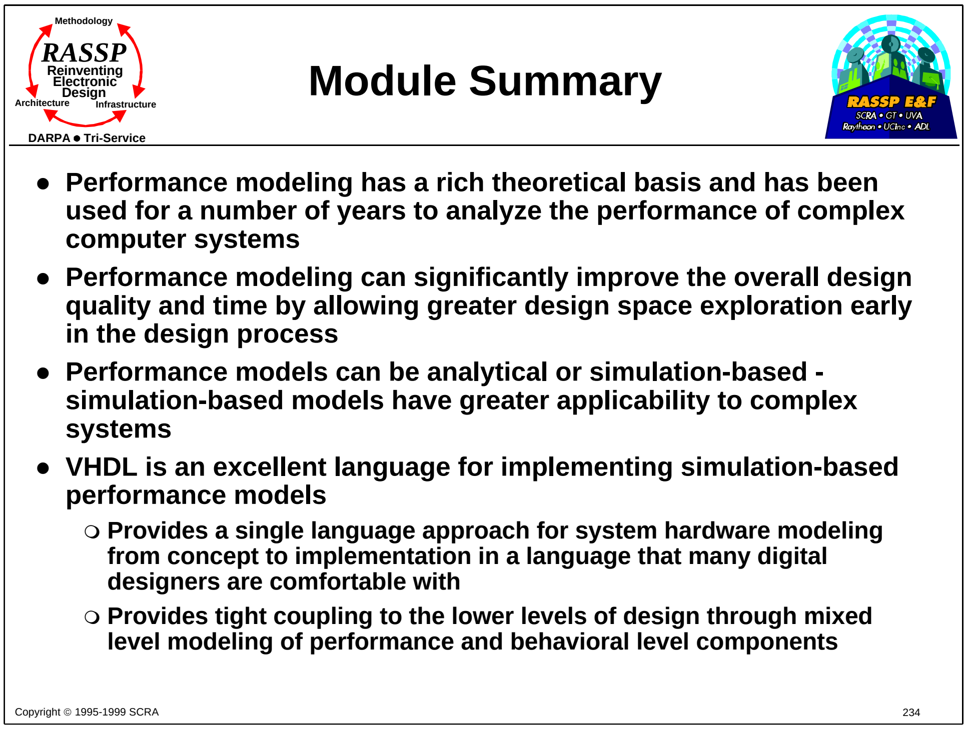
Here are some results from the mixed level model illustrating the proper control of the motor speed. Note that the behavioral model of the microcontroller is executing an actual control program from a memory model and responding to perturbations in the motor speed in the performance model of the motor system.



Module Outline

- Performance Modeling Introduction
- Performance Modeling Theory
- Non VHDL-Based Performance Modeling Tools
- Techniques for Performance Modeling using VHDL
- VHDL-Based Performance Modeling Tools
- VHDL Performance Modeling Examples
- Mixed Level Modeling
- **Module Summary**

Module Outline



Module Summary

- Performance modeling has a rich theoretical basis and has been used for a number of years to analyze the performance of complex computer systems
- Performance modeling can significantly improve the overall design quality and time by allowing greater design space exploration early in the design process
- Performance models can be analytical or simulation-based - simulation-based models have greater applicability to complex systems
- VHDL is an excellent language for implementing simulation-based performance models
  - ❍ Provides a single language approach for system hardware modeling from concept to implementation in a language that many digital designers are comfortable with
  - ❍ Provides tight coupling to the lower levels of design through mixed level modeling of performance and behavioral level components

References

[ADEPT_LR96] ADEPT A.1 Library Reference Manual, CSIS Technical Report No. 960625, Department of Electrical Engineering, University of Virginia, December, 1996. See http://www.ee.virginia.edu/research/CSIS/

[ADEPT_UM96] Unified Modeling Reference Manual (ADEPT Version A.1), CSIS Technical Report No. 960620.0, Department of Electrical Engineering, University of Virginia, December, 1996.

[Cassandras93] Cassandras, Christos G., *Discrete Event Systems, Modeling and Performance Analysis*, Aksen Associates Incorporated Publishers, 1993.

[IEEE] All referenced IEEE material is used with permission.

[Hein96] Hein, C., T. Carpenter, “Tutorial: VHDL-Based Rapid Prototyping for Large DSP Systems,” Presented at the Second Annual RASSP Conference, October 10th, 1996.

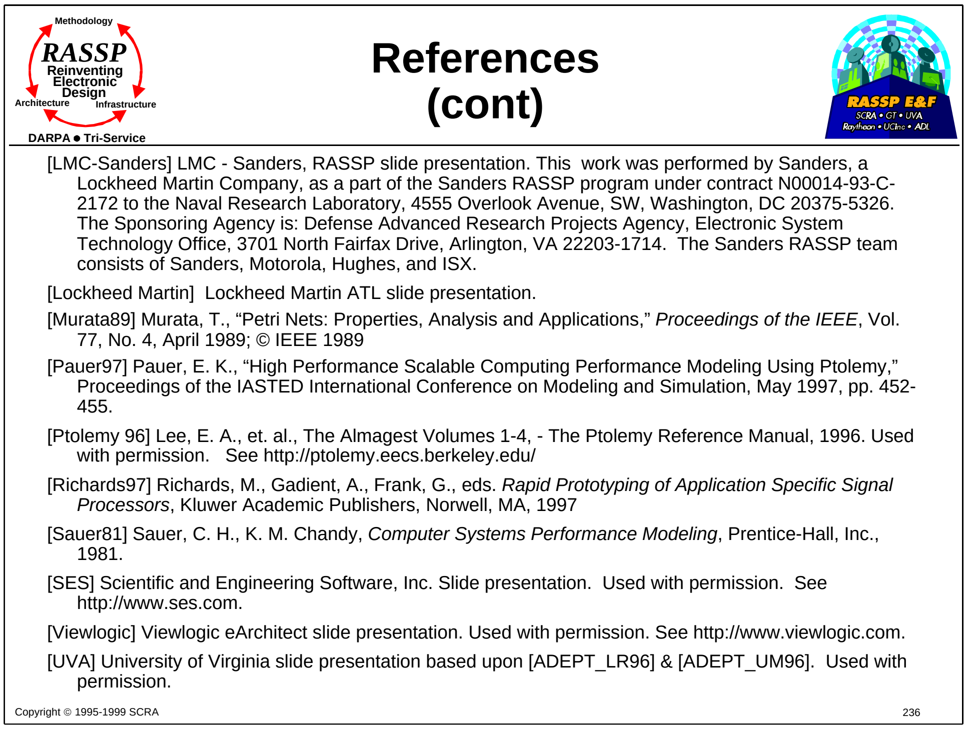
[Hein97] Hein, C., T. Carpenter, A. Gadient, R. Harr, P. Kalutkiewicz, V. Madisetti, “RASSP VHDL Modeling Terminology and Taxonomy,” Revision 2.2, March 27, 1997.

[Honeywell] Honeywell, RASSP slide presentation. Used with permission.

[HTC97] RASSP VHDL Performance Modeling Interoperability Guideline, Version 3.0, Honeywell Technology Center, March 31, 1997.

[Jain91] Jain, R., *The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, and Modeling*, John Wiley & Sons, Inc., 1991.

[Kant92] Kant, K., *Introduction to Computer System Performance Evaluation*, McGraw-Hill, Inc., 1992.



References (cont)

[LMC-Sanders] LMC - Sanders, RASSP slide presentation. This work was performed by Sanders, a Lockheed Martin Company, as a part of the Sanders RASSP program under contract N00014-93-C-2172 to the Naval Research Laboratory, 4555 Overlook Avenue, SW, Washington, DC 20375-5326. The Sponsoring Agency is: Defense Advanced Research Projects Agency, Electronic System Technology Office, 3701 North Fairfax Drive, Arlington, VA 22203-1714. The Sanders RASSP team consists of Sanders, Motorola, Hughes, and ISX.

[Lockheed Martin] Lockheed Martin ATL slide presentation.

[Murata89] Murata, T., “Petri Nets: Properties, Analysis and Applications,” *Proceedings of the IEEE*, Vol. 77, No. 4, April 1989; © IEEE 1989

[Pauer97] Pauer, E. K., “High Performance Scalable Computing Performance Modeling Using Ptolemy,” Proceedings of the IASTED International Conference on Modeling and Simulation, May 1997, pp. 452-455.

[Ptolemy 96] Lee, E. A., et. al., The Almagest Volumes 1-4, - The Ptolemy Reference Manual, 1996. Used with permission. See http://ptolemy.eecs.berkeley.edu/

[Richards97] Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997

[Sauer81] Sauer, C. H., K. M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall, Inc., 1981.

[SES] Scientific and Engineering Software, Inc. Slide presentation. Used with permission. See http://www.ses.com.

[Viewlogic] Viewlogic eArchitect slide presentation. Used with permission. See http://www.viewlogic.com.

[UVA] University of Virginia slide presentation based upon [ADEPT_LR96] & [ADEPT_UM96]. Used with permission.