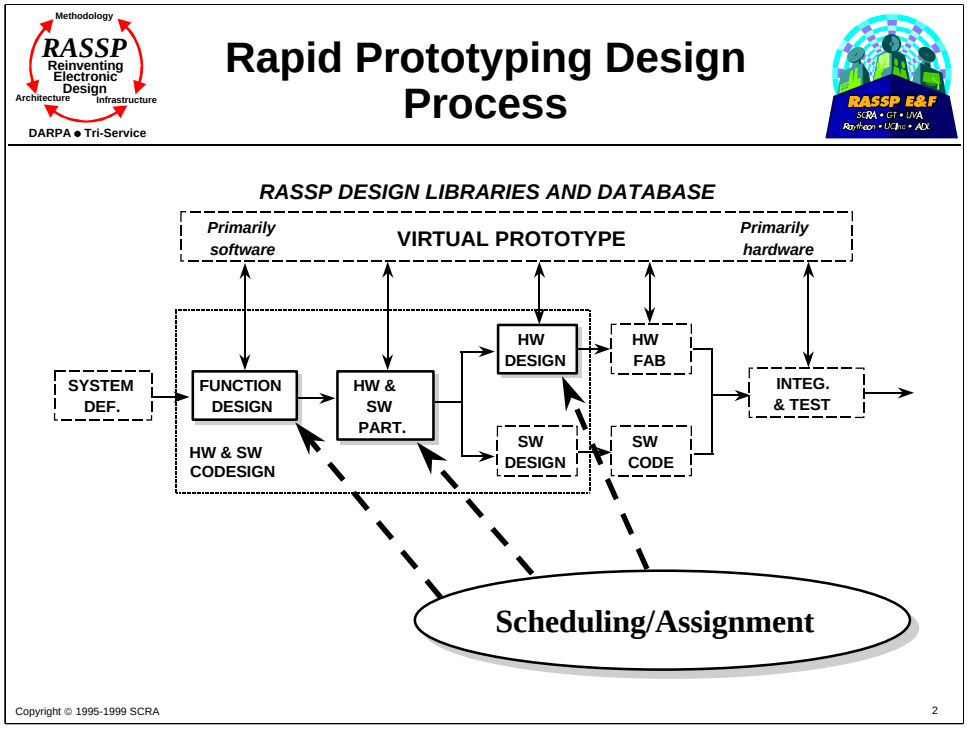
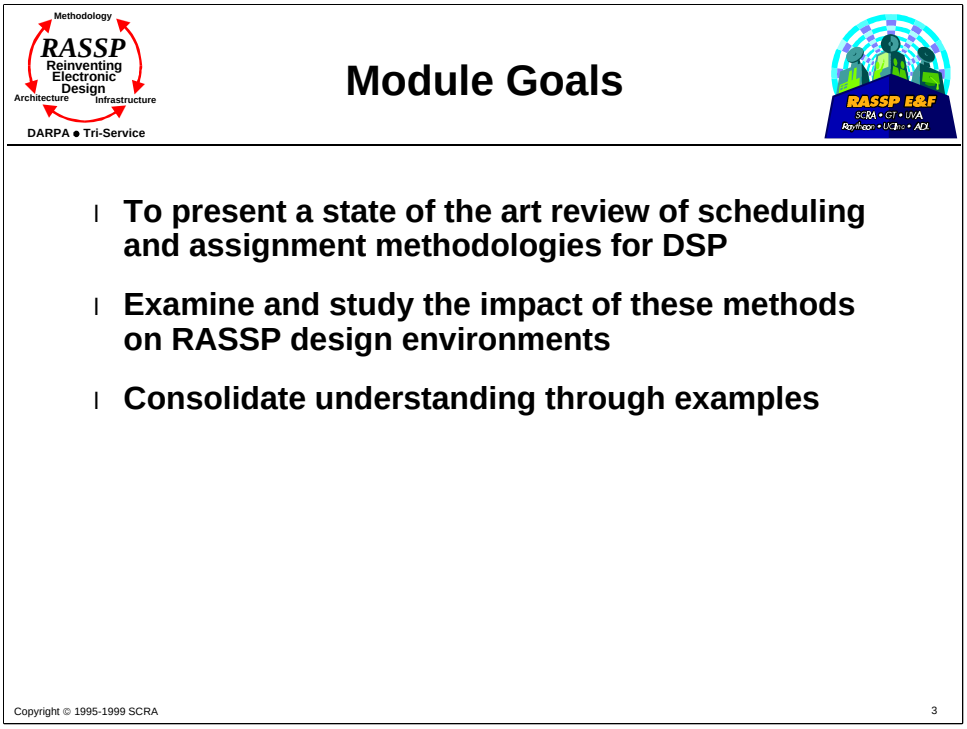
Scheduling and Assignment for DSP

RASSP Education & Facilitation Program
Module 22

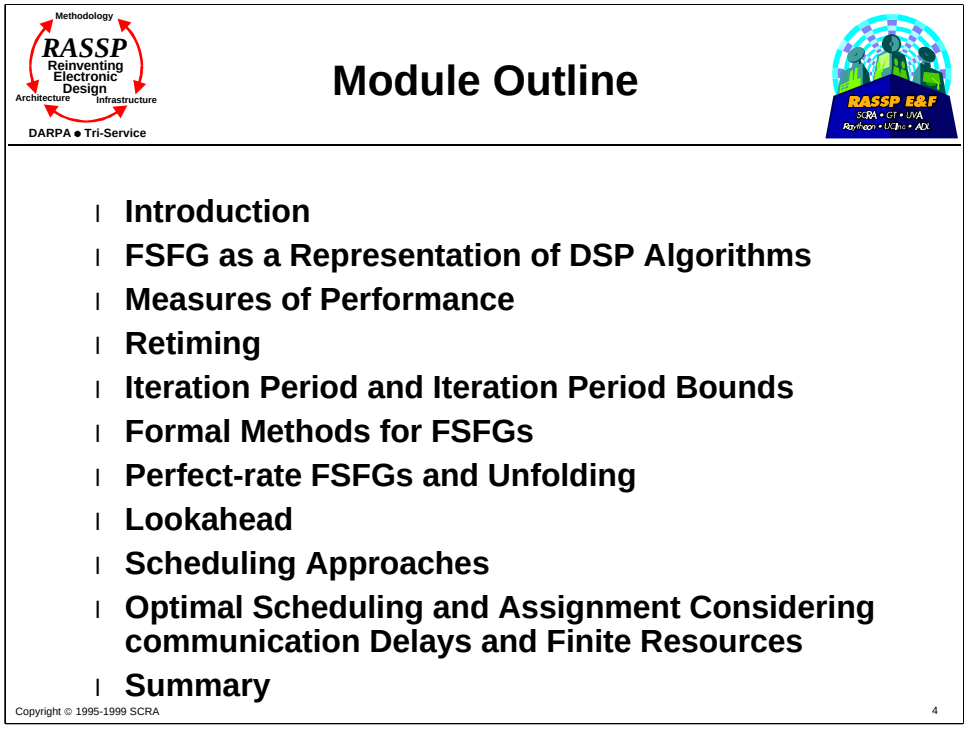
Version 03.00



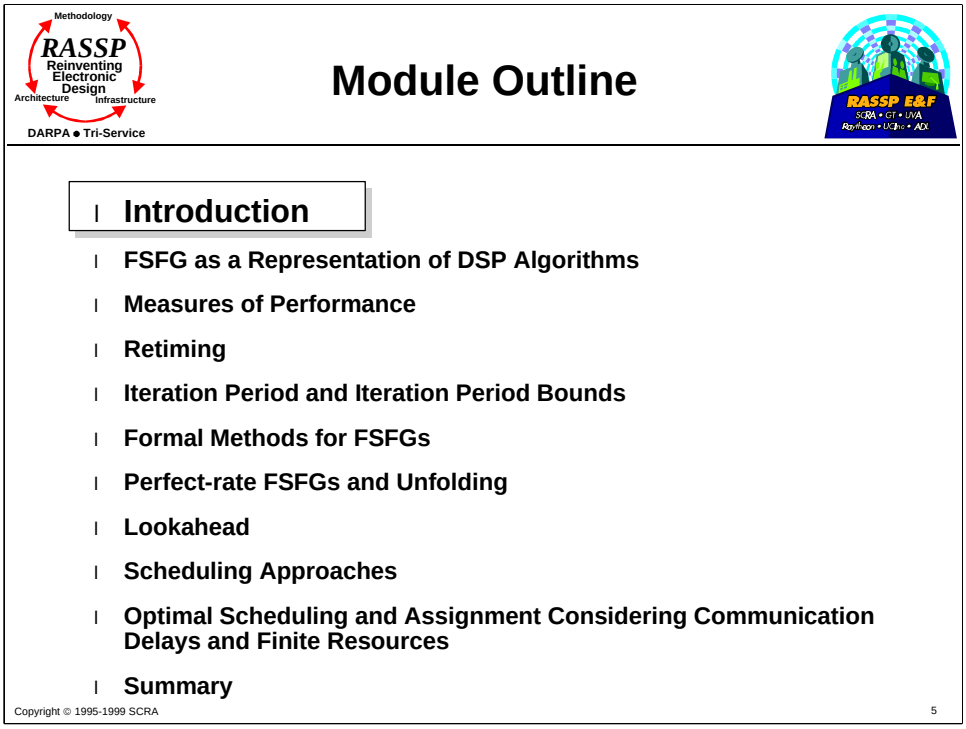
Architecture design is based on the functional computational and communications requirements of the algorithm or algorithms selected to meet the specification. These trade-offs fall under the functional design and partitioning sections of the RASSP process.



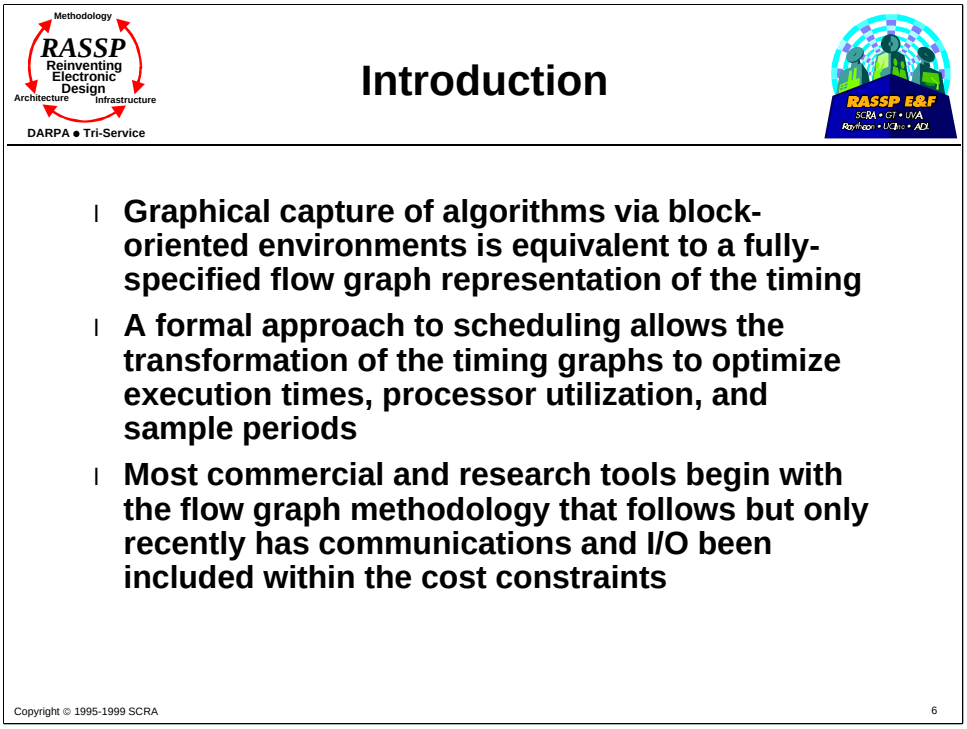
This module will cover the areas listed above. The purpose of this module is to expose the user of the RASSP process to the area of architecture design.



The module describes in detail the process of scheduling, allocation, and mapping of DSP algorithms onto multiprocessor DSP architectures.



We first describe how the input to the scheduling environment is specified (in terms of a Fully Specified Flow Graph).

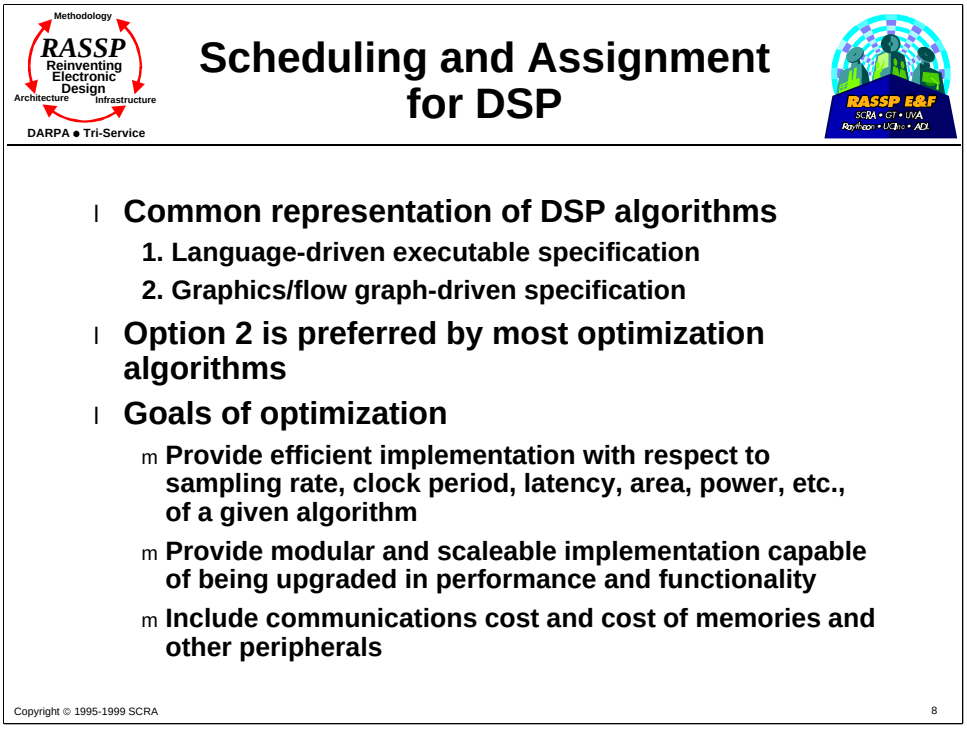


A unified representation of DSP algorithms (at the level of a block oriented graphical language) is used to describe the computational flow and concurrency in the DSP algorithm. An equivalent language based specification could be used, if available.

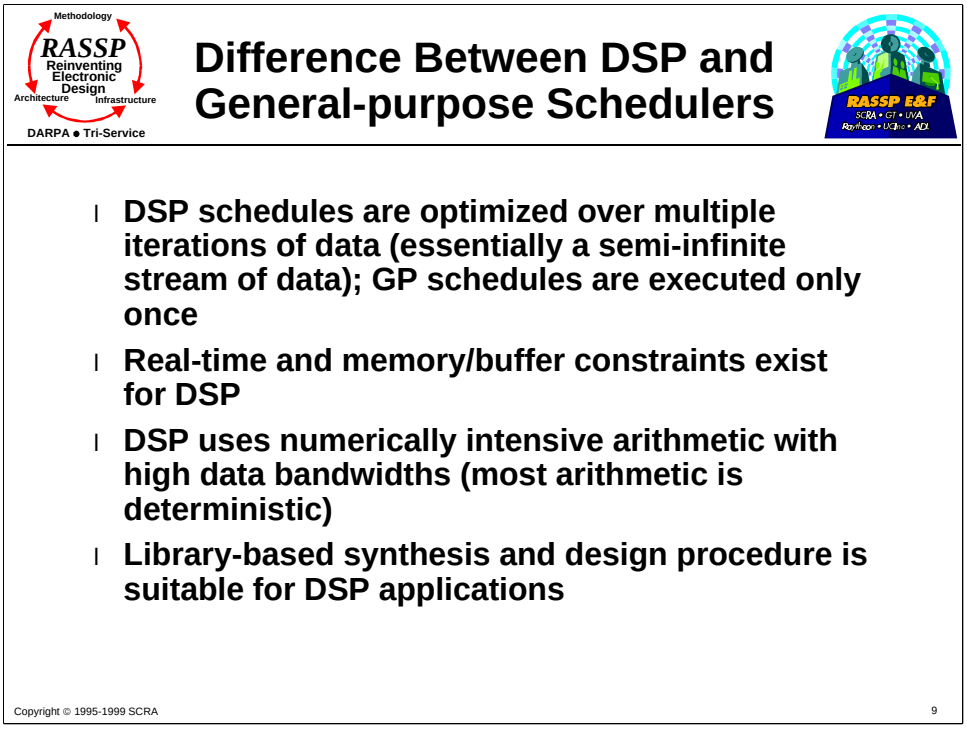
Module Outline

- Introduction
- **FSFG as a Representation of DSP Algorithms**
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- Formal Methods for FSFGs
- Perfect-rate FSFGs and Unfolding
- Lookahead
- Scheduling Approaches
- Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- Summary

.



Both language-driven and graphical-driven front ends have their advantages and disadvantages, though the graphical approach is utilized in this module for purposes of ease of representation of the underlying scheduling, mapping, and allocation technologies.

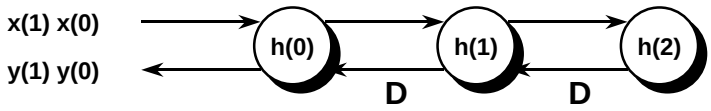


Application specificity results from determinism. An ever-present stream of data adds to the efficiency of the DSP scheduling process.

FSFG Used to Represent DSP Functional/Timing

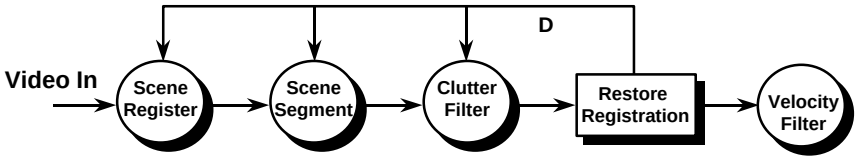
FSFG = Fully Specified Flow Graph (Data/Control Flow Graph)

- A convolution - A fine-grain depiction

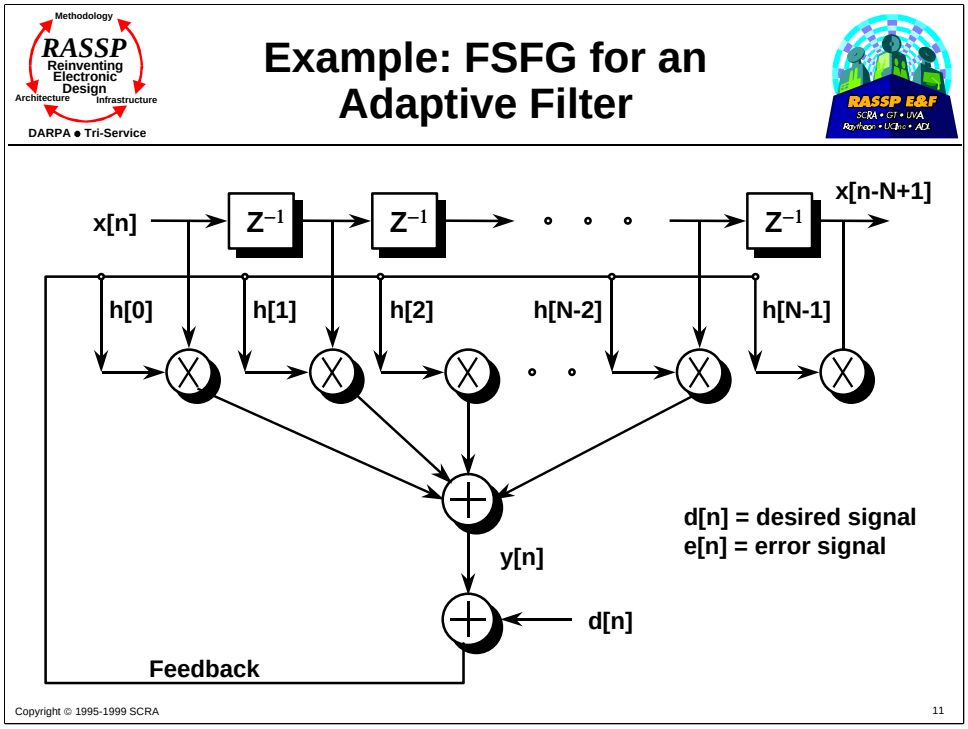


x(.) = data input y(.) = data output h(.) = coefficients D = unit delays

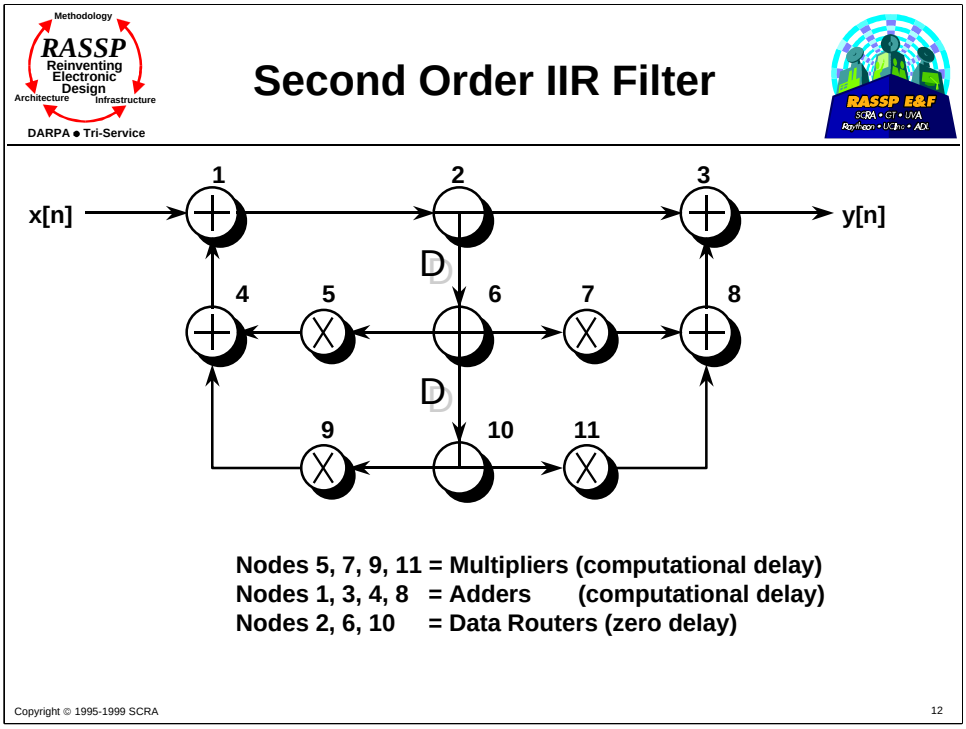
- IRST - A coarse-grain depiction



FSFGs can be used to represent a variety of classes of algorithms.

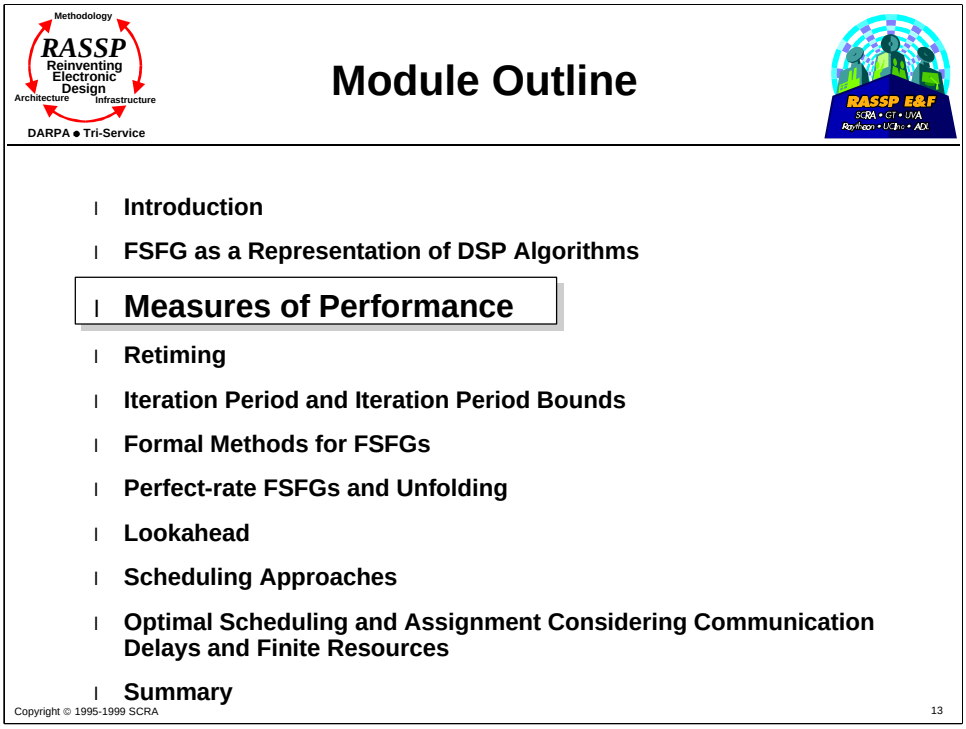


FSFGs are particularly suited for DSP applications - examples. Note that it is not necessary to have N multipliers in the eventual implementation, and the FSFG only represents the input specification (including its concurrency) and not the final design.

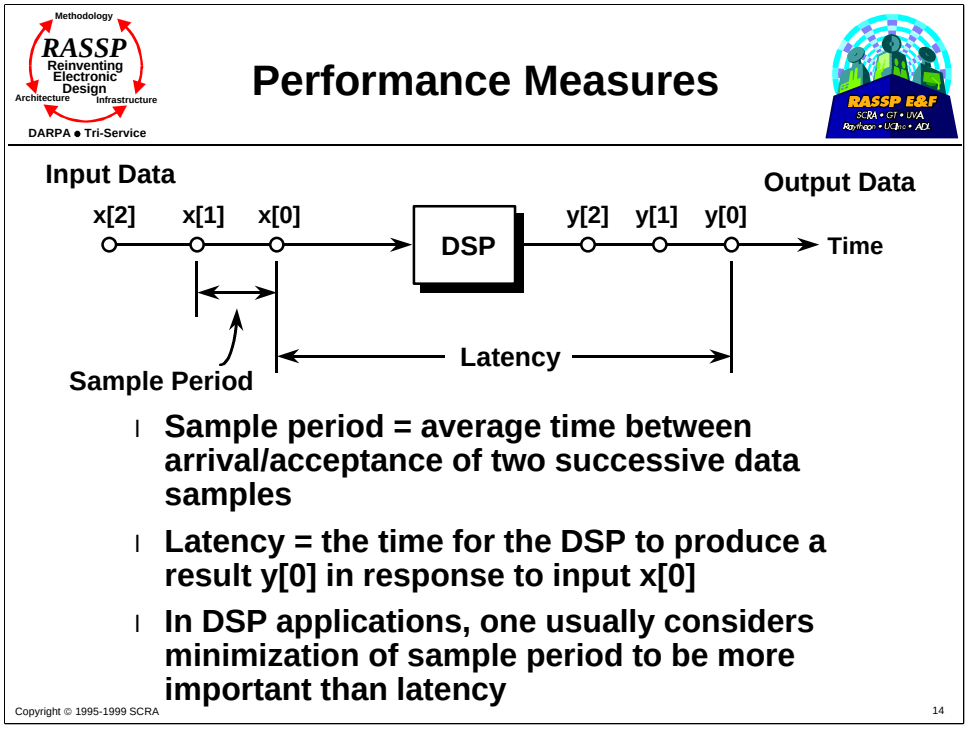


This example is used throughout the module to represent the various scheduling technologies available. The delays (D) represent logical delays. E.g., x[n] when passed through a D operator results in x[n-1] and represent storage elements for intermediate computation.

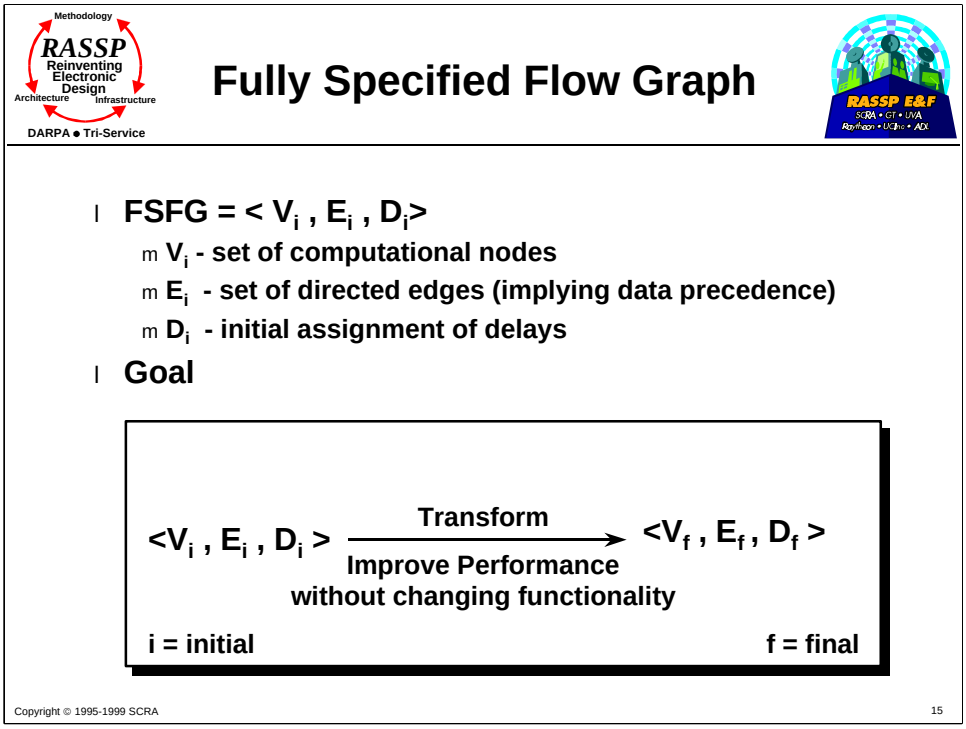
Another type of delay is called the computational delay that represents the wall-clock delay due to the computational time requirements of the datapath operators.



One cannot define how well a scheduling technology performs without introducing some metrics or measures of performance. These will be described in the following slides together with their bounds.

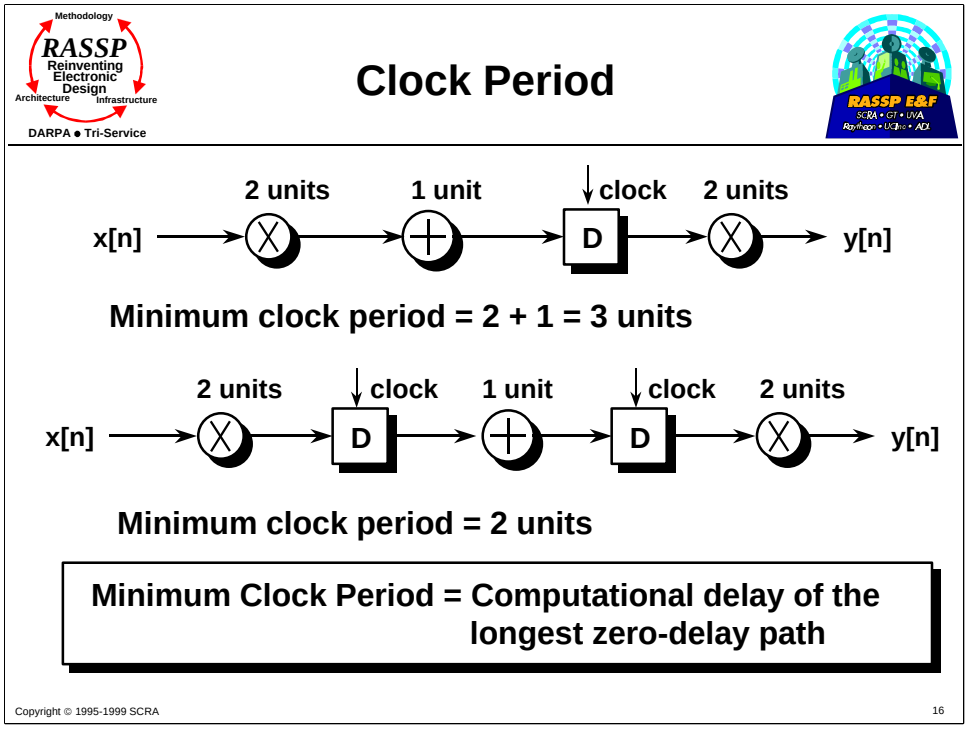


Of the performance measures, the Sample Period is more important than the Latency.

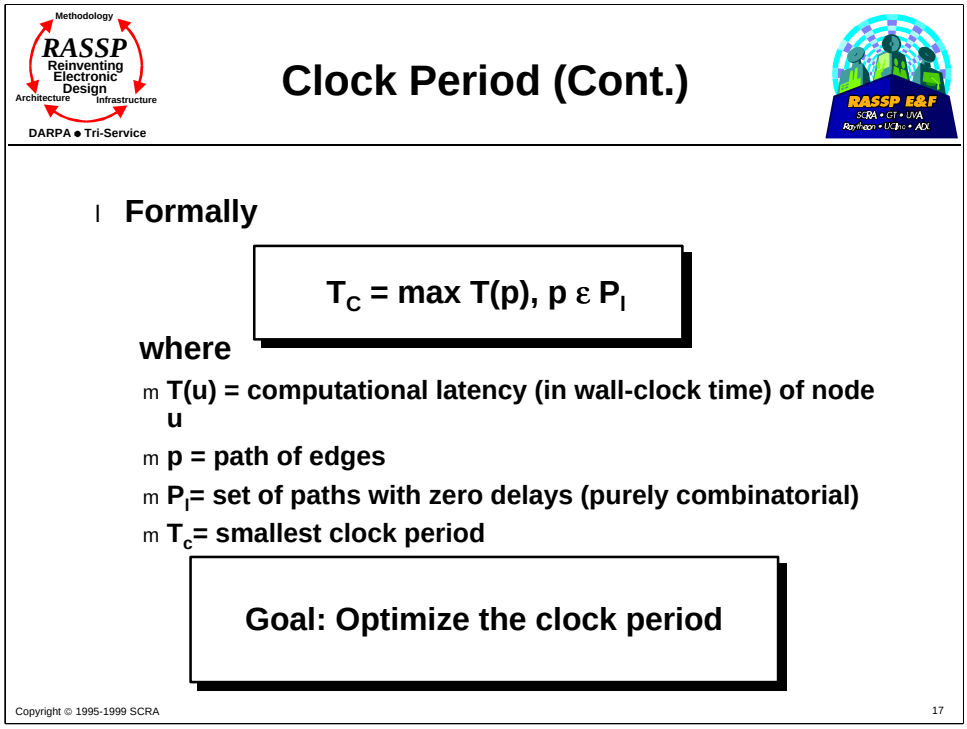


This slide formally defines the FSFG.

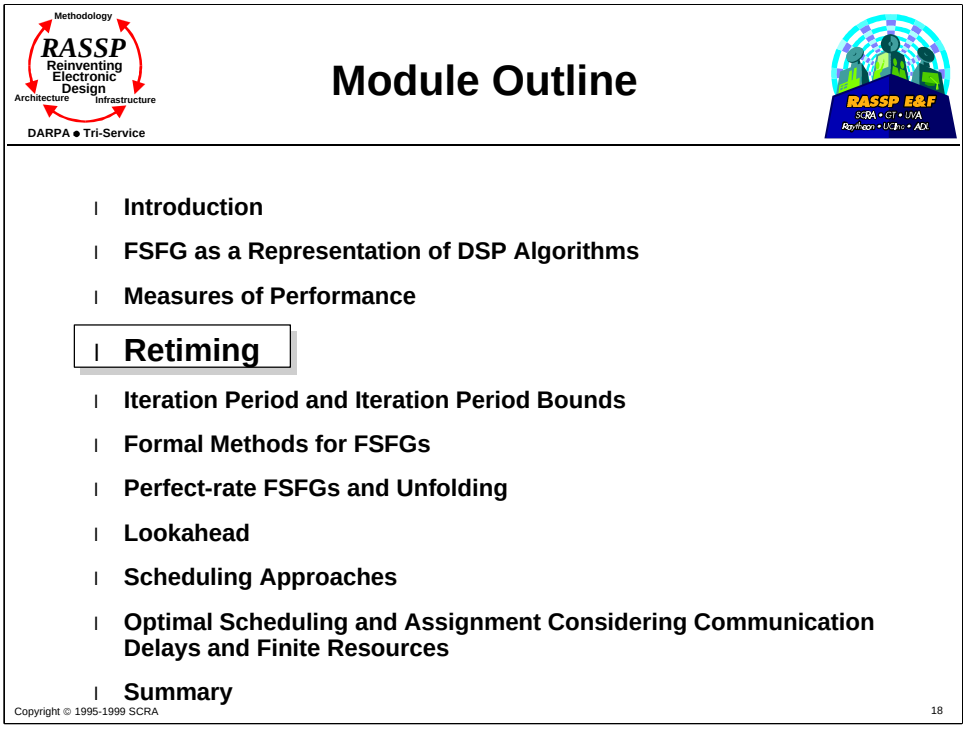
The nodes are non-preemptible.



This slide describes a simple optimization (visually) of a functional unit chain.



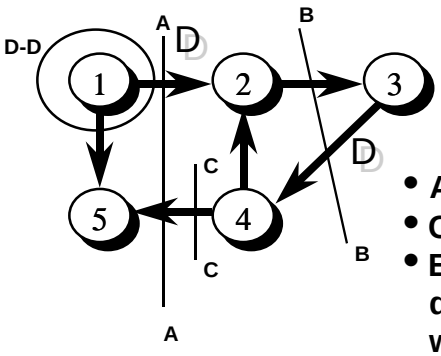
Note the difference between delay elements [D] and wall clock computational delays within arithmetic and other operators that contribute to the clock period and latency.



Retiming is the reassignment of delays within a FSFG without changing its algorithmic behavior.

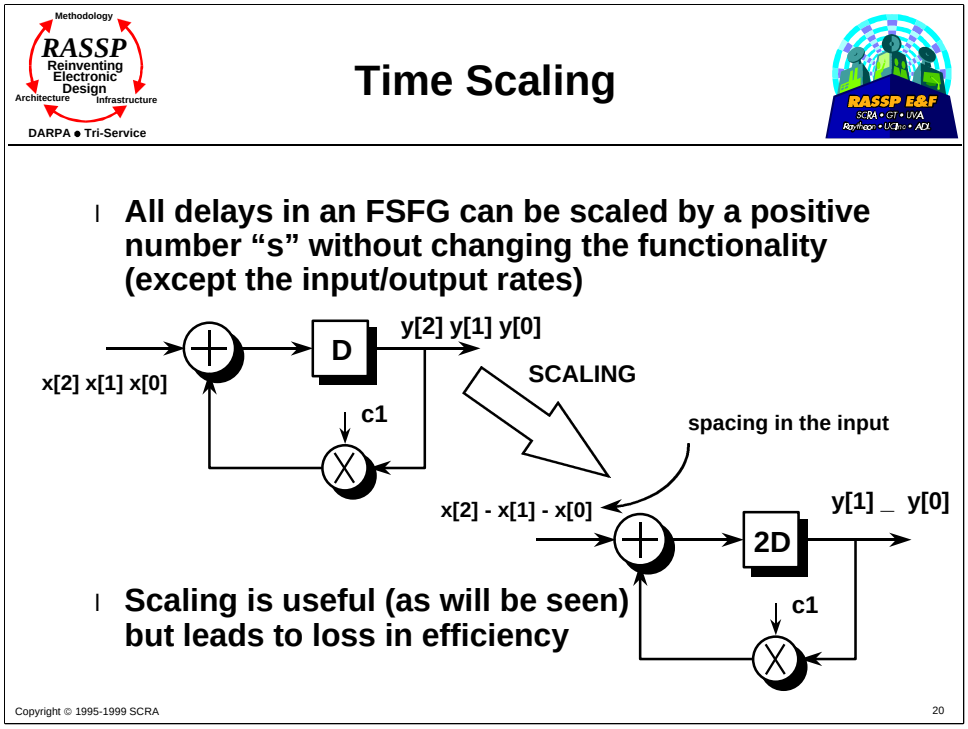
Cutset Definition

- A Cutset consists of the minimal set of edges that divide an FSFG into two disjoint parts

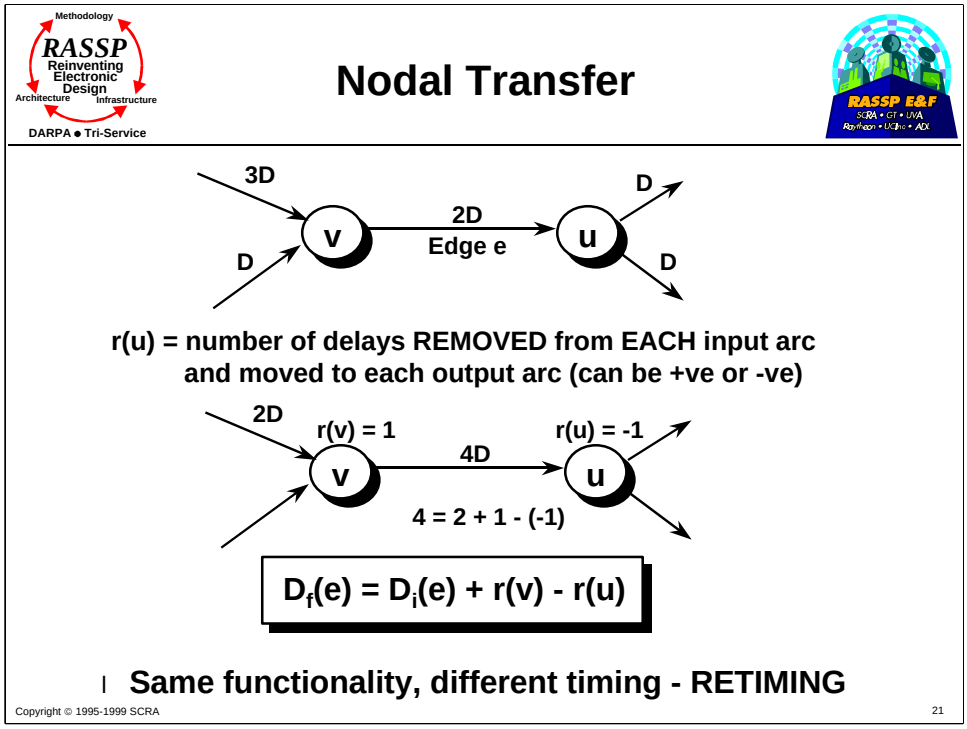


- A-A, B-B, D-D are valid cutsets
- C-C is not a valid cutset
- Edge 1-2 and edge 4-5 are directed in opposite directions with respect to cutset A-A

Cutsets will be important to the scheduling methodologies introduced in the remainder of the module.



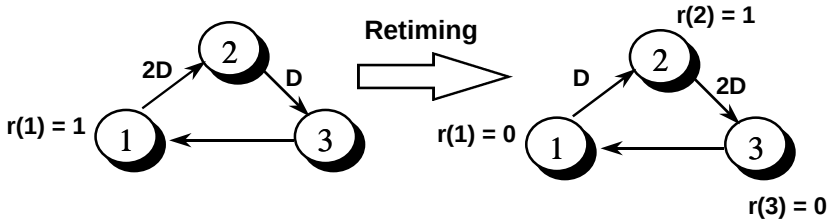
A number of examples of time scaling can be introduced at this juncture to illustrate time scaling.



We illustrate the process of transfer of delays through a node without changing functionality (See [Madisetti95] for further details).

Legal Retiming

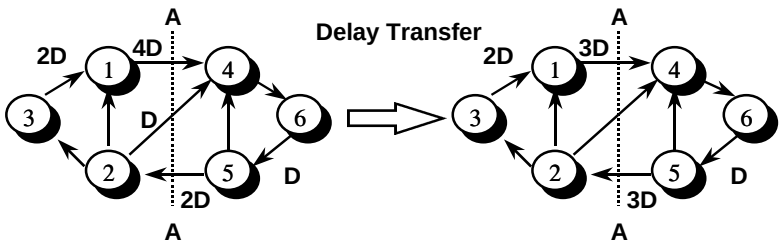
- $D_f(e) >= 0$ for all edges
- $D_f(e) >= 1$ for a SYSTOLIC FSFG
  - Systolic FSFGs have minimum clock period, or the highest clock frequency
- Total number of delays in a loop remains constant with and without retiming (no loop can have a zero delay)



A neat way to convert FSFGs to systolic arrays.

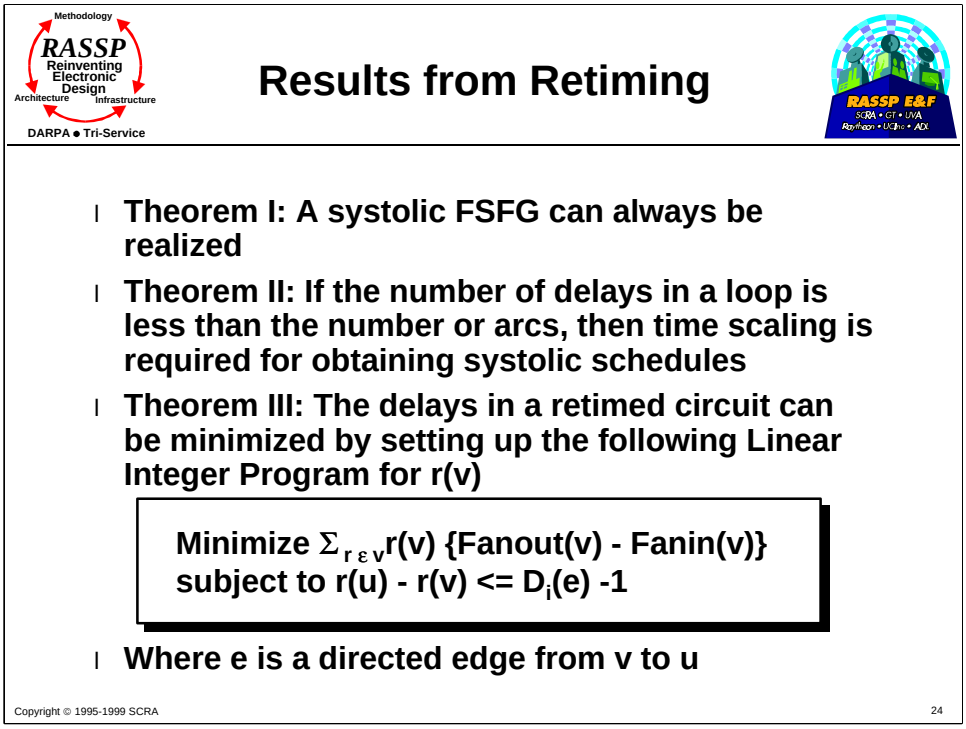
Systolic arrays are not efficient though.

Delay Transfer Across Cutsets

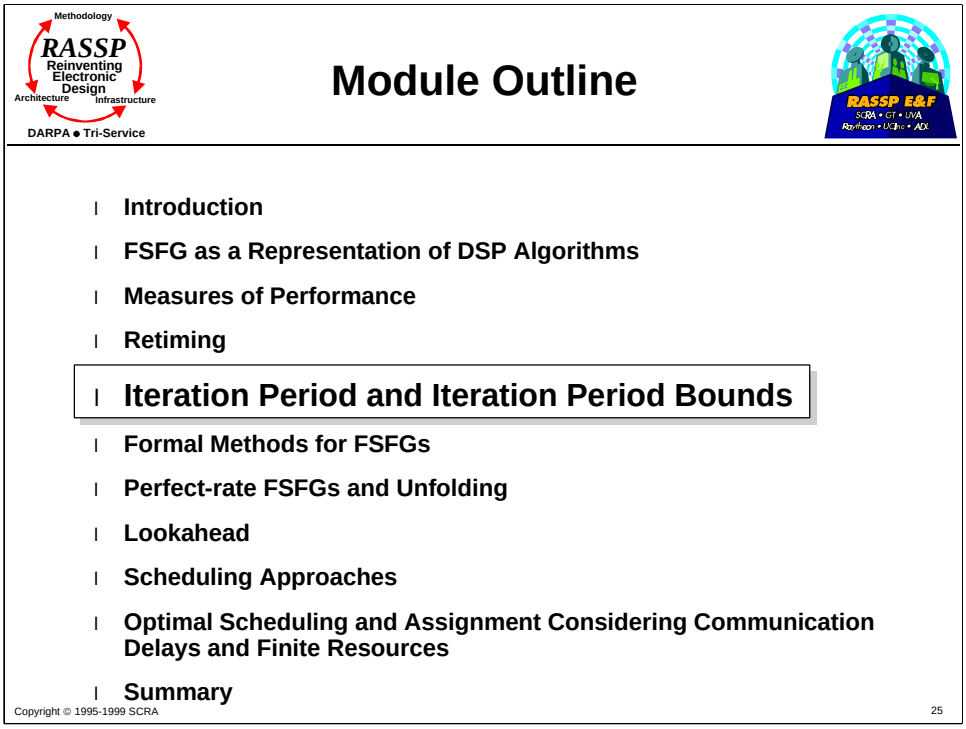


- **Remove k delays from positive edges and move them to negative edges (k can be positive or negative)**
- **No algorithmic change in behavior**

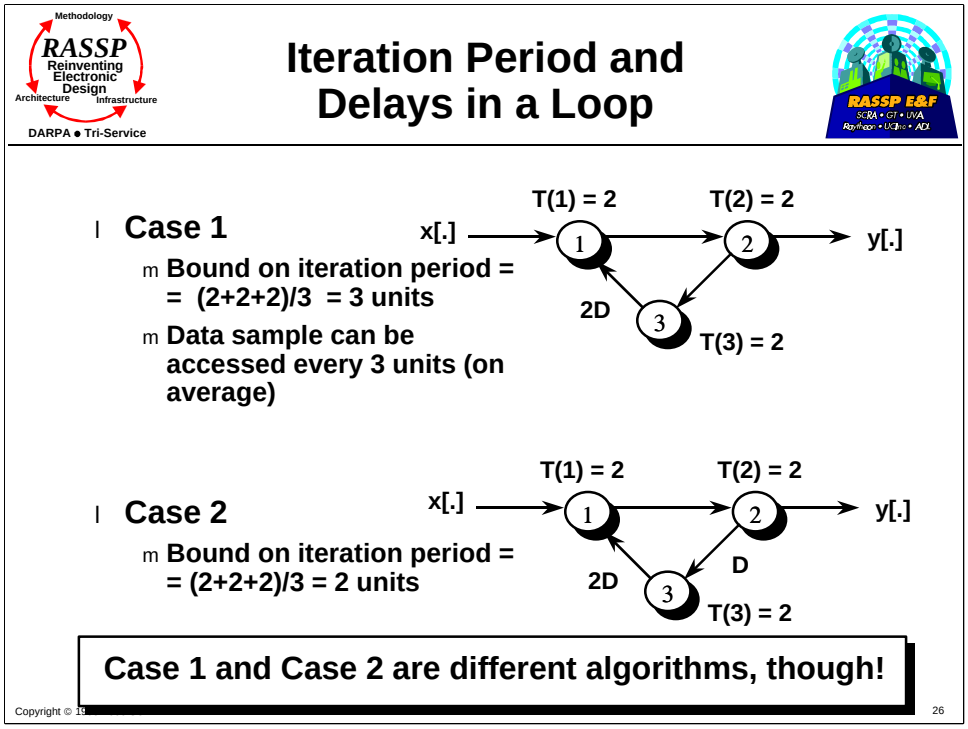
We will use these properties to optimize schedules.



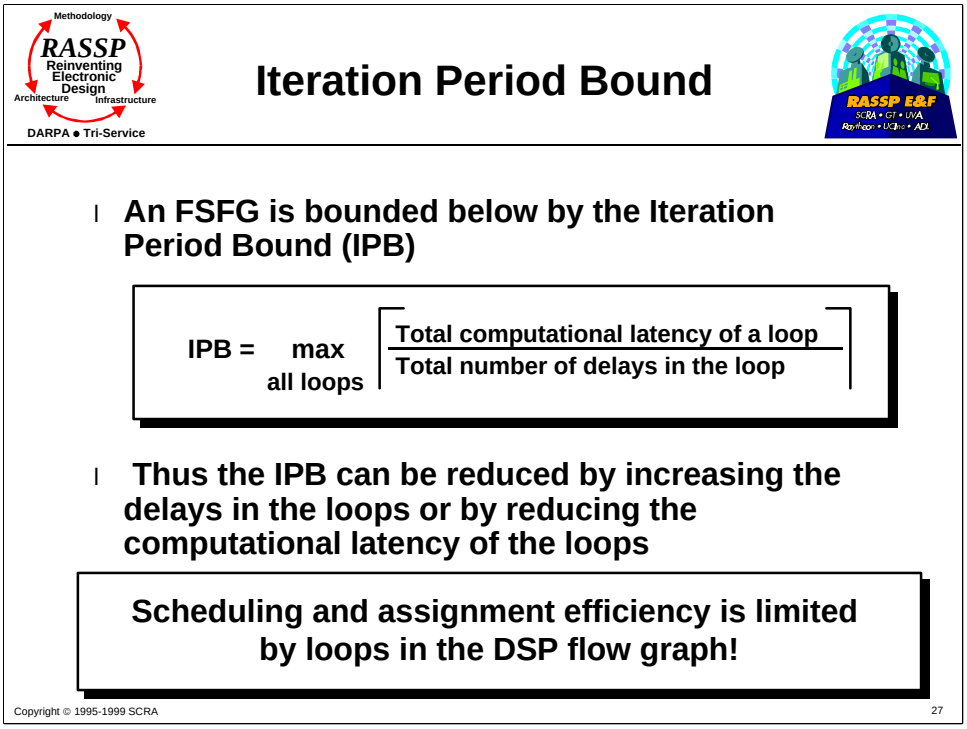
Proofs of these results are better illustrated by examples.



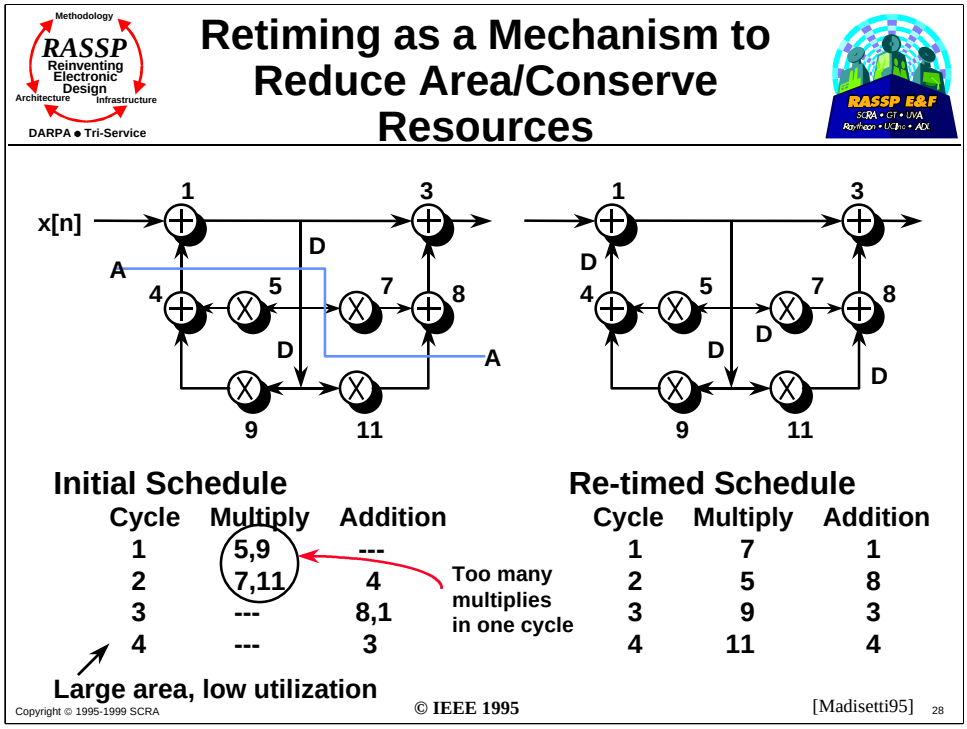
A few bounds that relate to the iteration period between successive periods of a DSP algorithm are now covered.



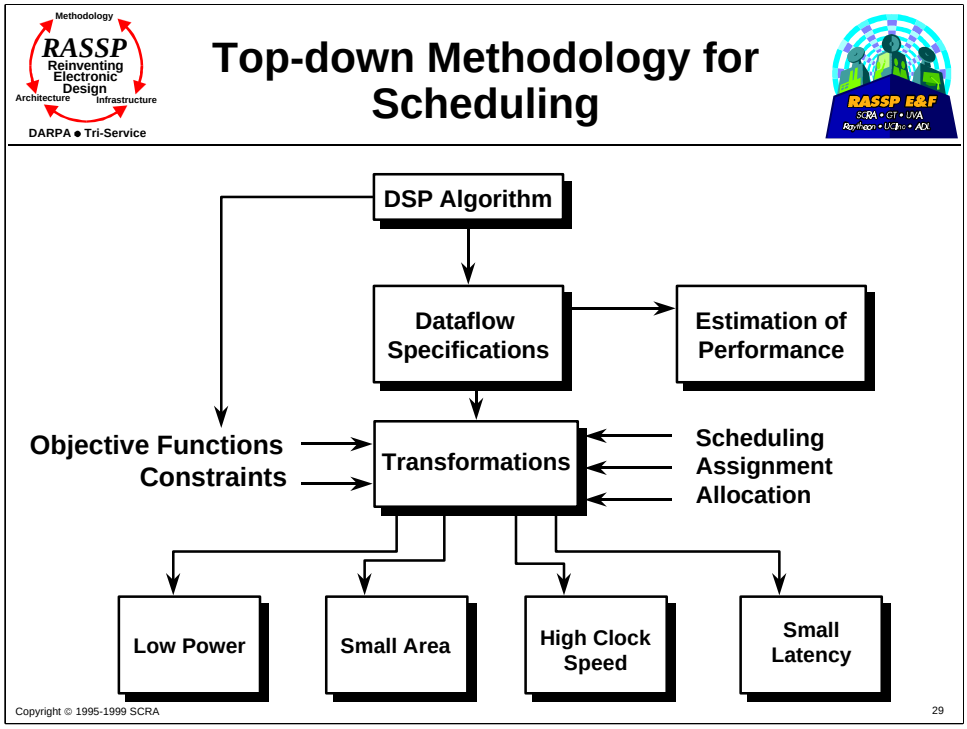
Iteration Period is related to the sample period in that the former is suitable for loops.



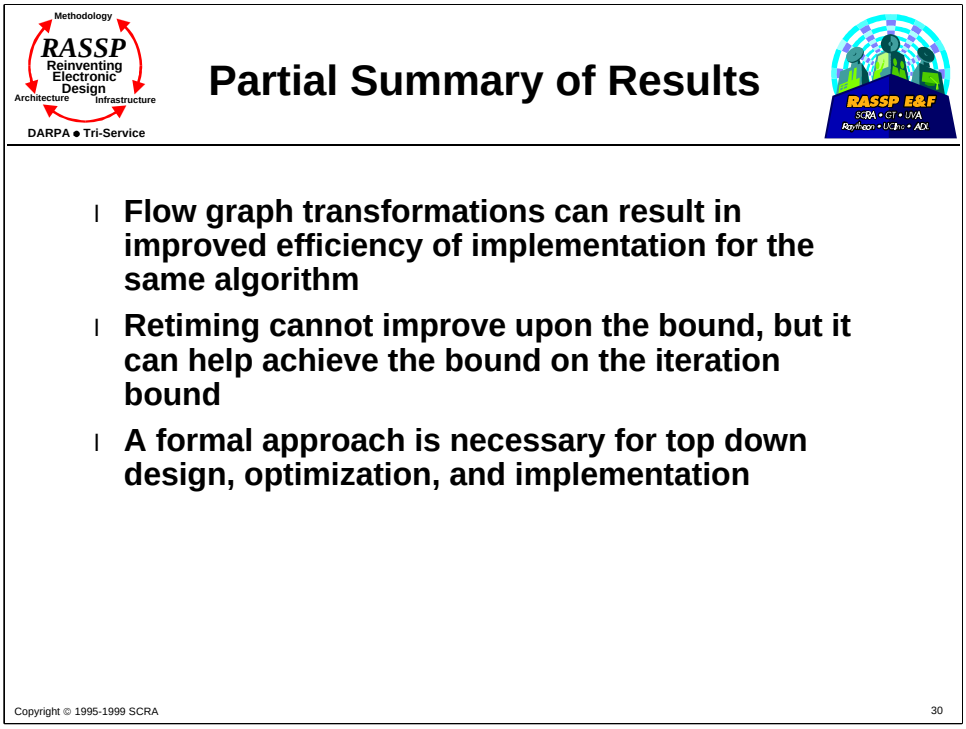
The IPB is a fundamental bound to scheduling efficiency.



We repeat the example to show how retiming “flattens” resource utilization.



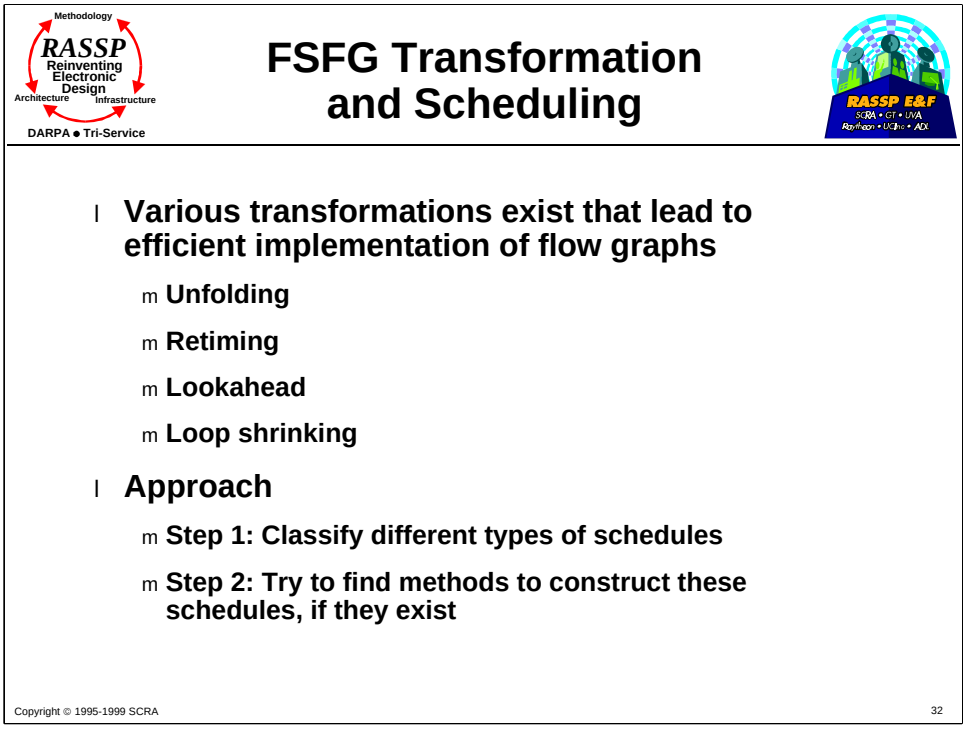
The DSP algorithm is first analyzed to derive its bounds and then suitable transformations of the flow graph are considered while keeping the original intent of the scheduling algorithm to suit the implementation objectives of low power, small area, or small latency, etc.



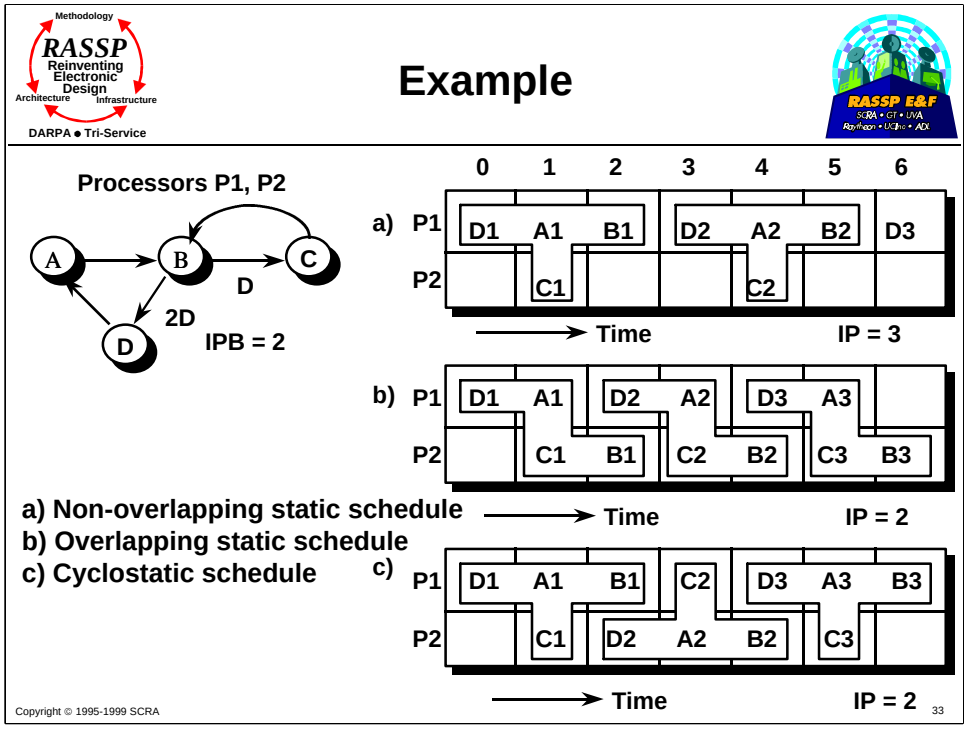
So far we have shown that retiming and transformations help, without formally proving anything --- so now we move ahead with a formal procedure.

Module Outline

- Introduction
- FSFG as a Representation of DSP Algorithms
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- **Formal Methods for FSFGs**
- Perfect-rate FSFGs and Unfolding
- Lookahead
- Scheduling Approaches
- Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- Summary

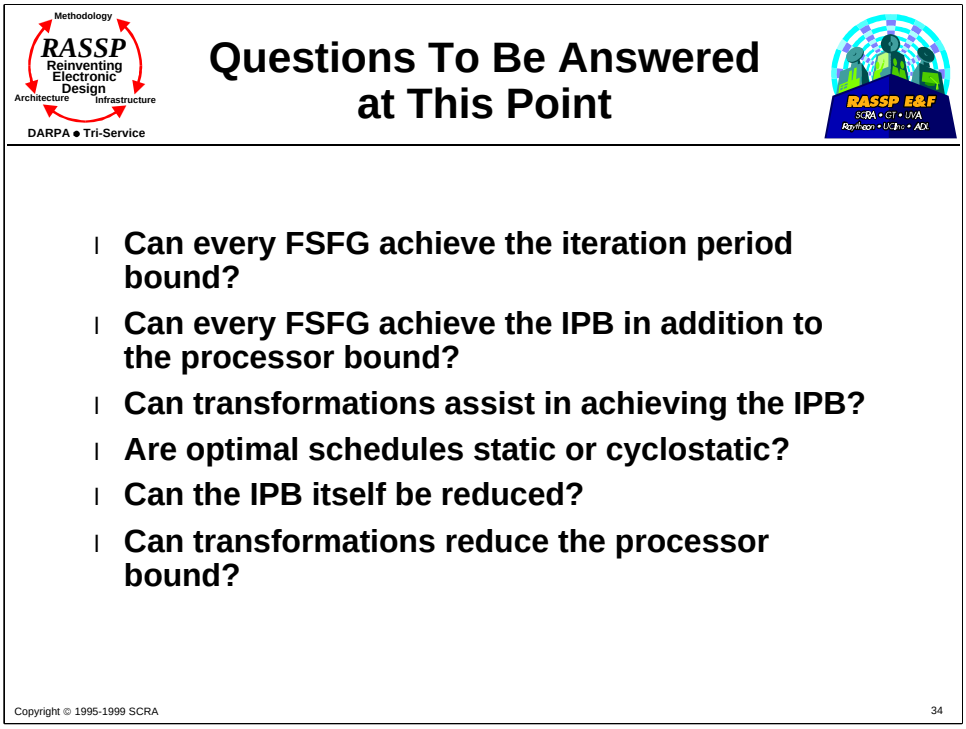


We follow a 2 step procedure - first identify what schedules we like, and then look or construct these schedules.

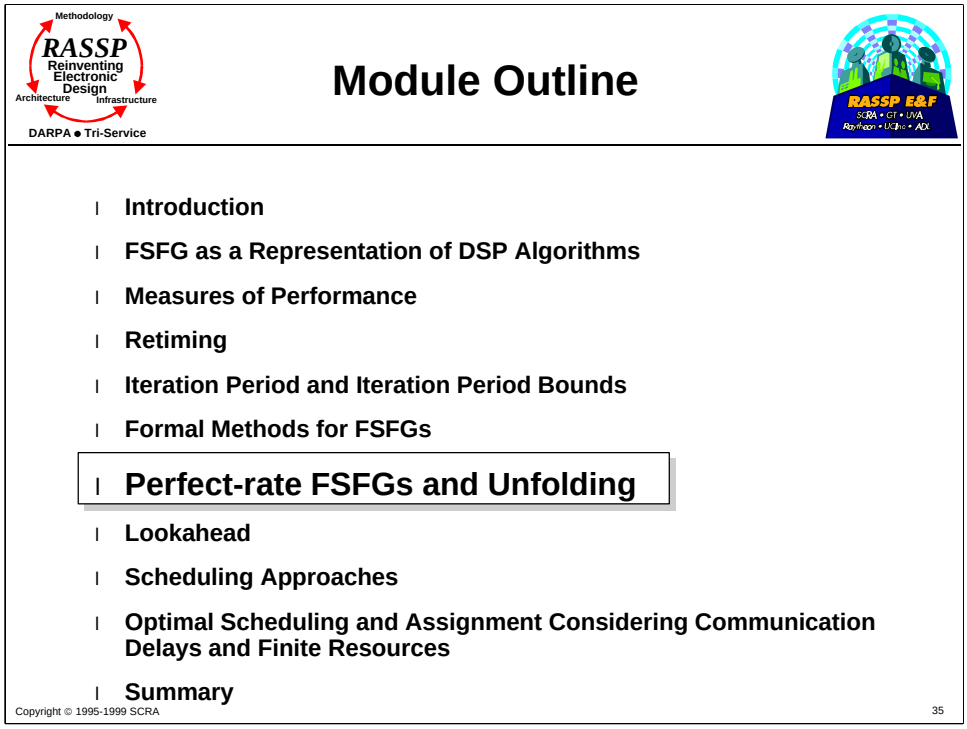


Though the cyclostatic and the static schedules both have the same IP here, the cyclostatic has, in general, a smaller delay although with more complex control.

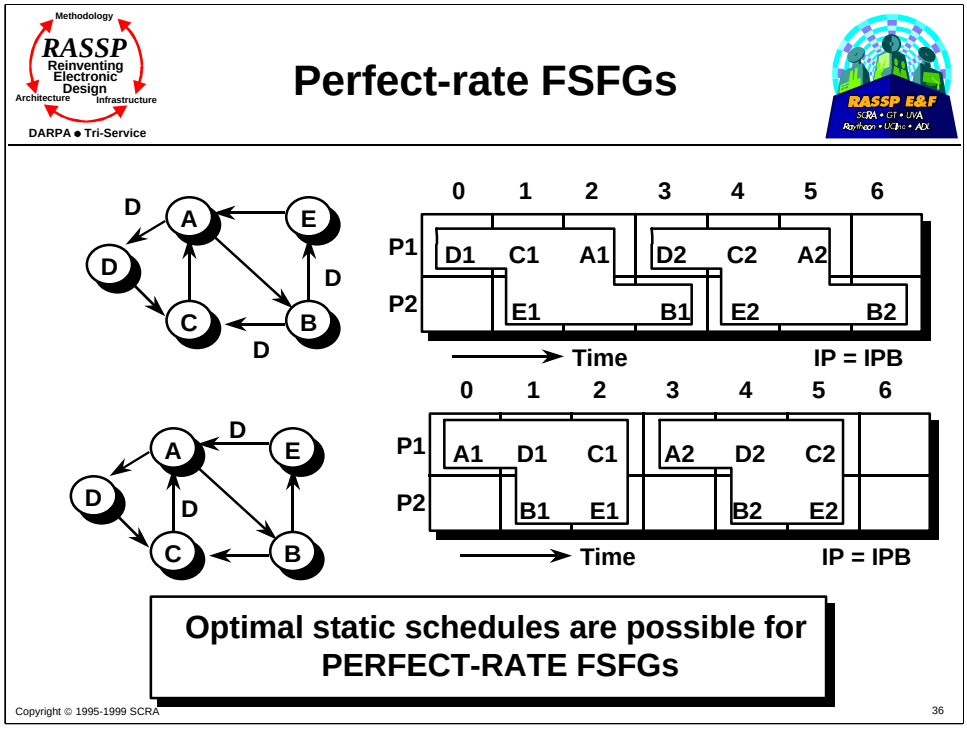
We prefer schedules of type (b).



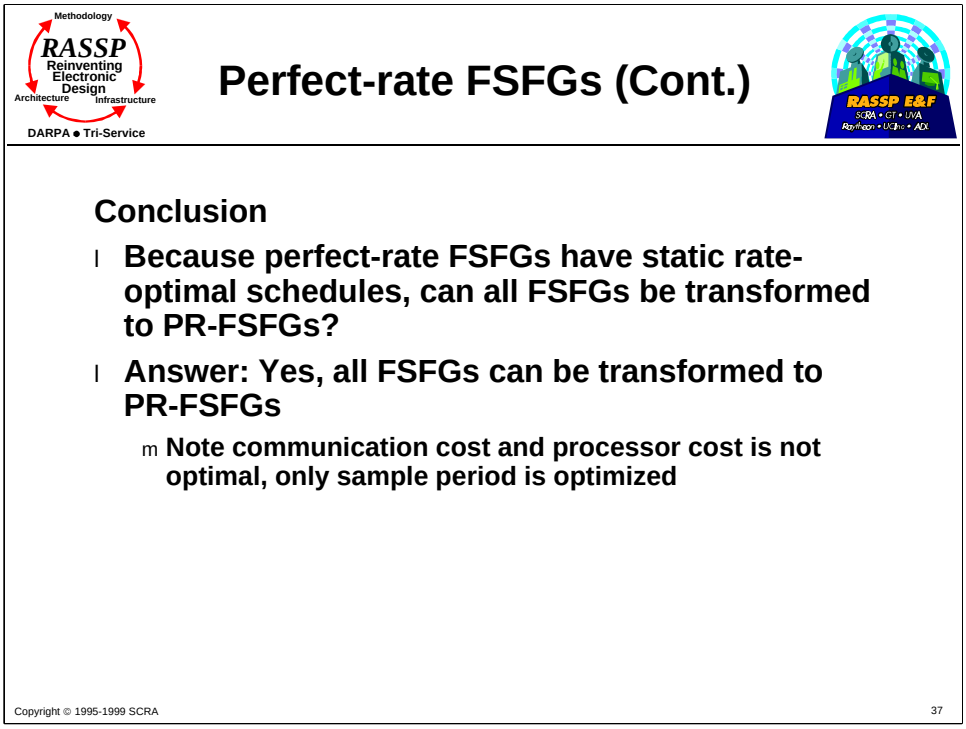
These questions will be answered in the following discussion.



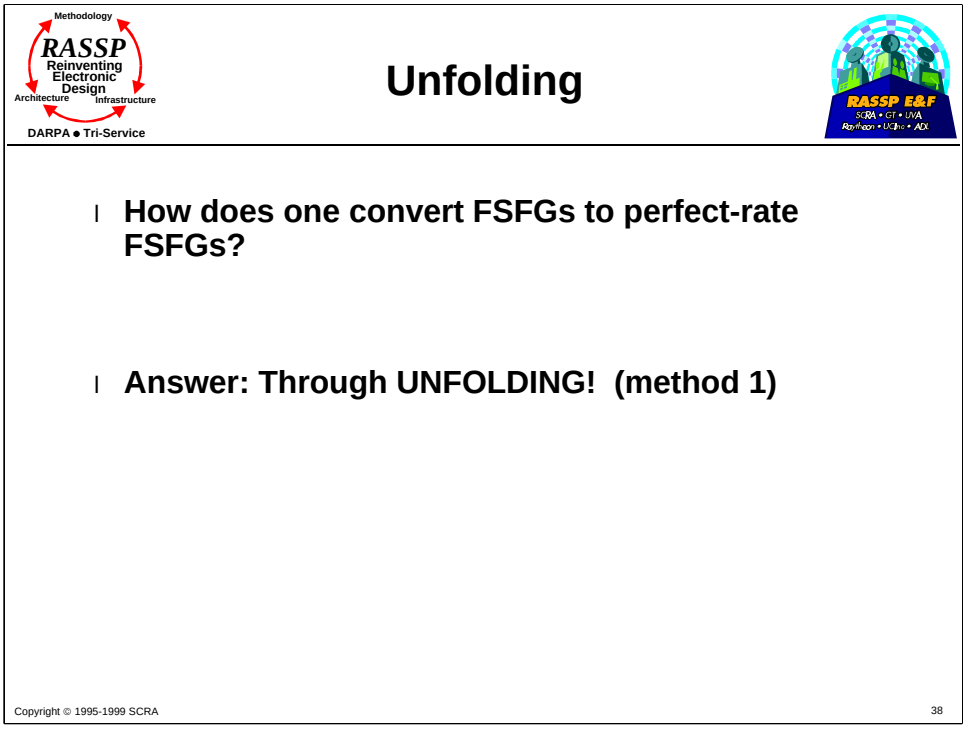
We now describe those graphs that have optimal rate schedules, called perfect-rate schedules.



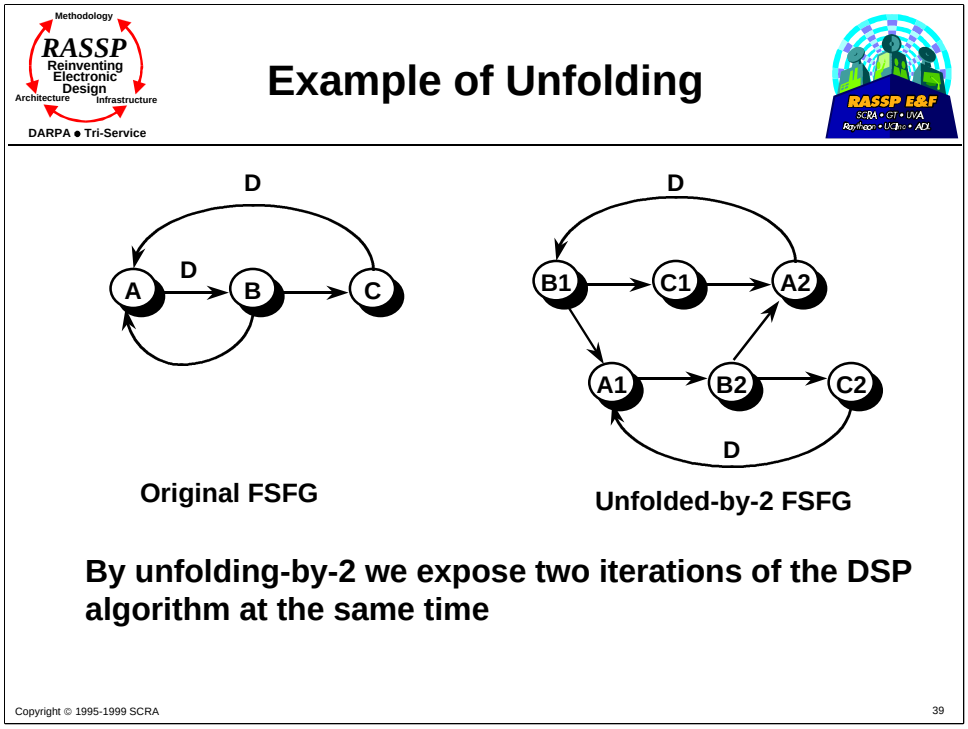
A perfect-rate FSFG is one which has only one delay in each loop - no more and no less. Perfect rate graphs on the left have the rate-optimal schedules shown on the right.



All perfect-rate FSFGs have rate optimal static schedules.



Unfolding is a method by which one executes two or more iterations of an FSFG in one step.

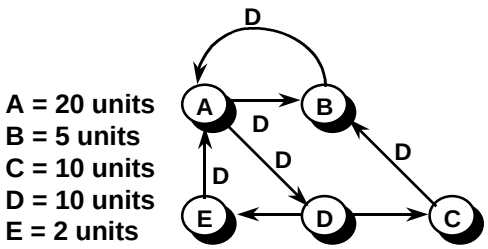


Here note that the unfolded graph is perfect-rate.

Also the IPB is 4, but two iterations get done in one step.

Optimum Unfolding

- **Unfold the flow graph by the LCM of the number of delays in each loop**
- **Example: Unfold by 6 results in IP = 16 units on 4 processors**



A = 20 units
B = 5 units
C = 10 units
D = 10 units
E = 2 units

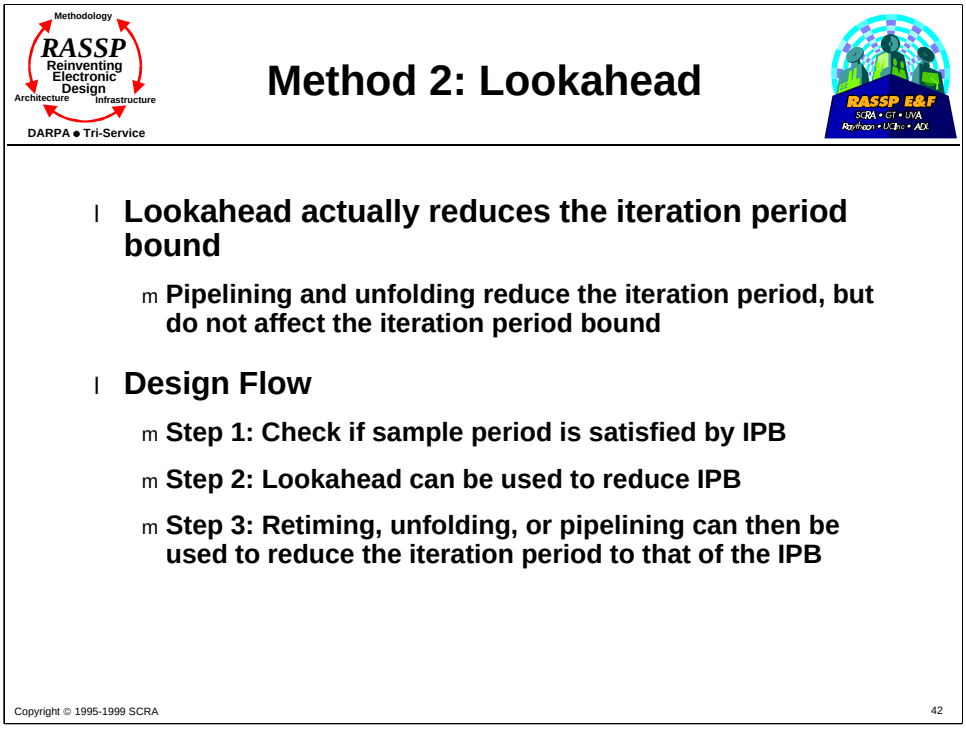
Note: Node A = 20 units which is larger than IPB

IP = 20 units
IPB = 16 units

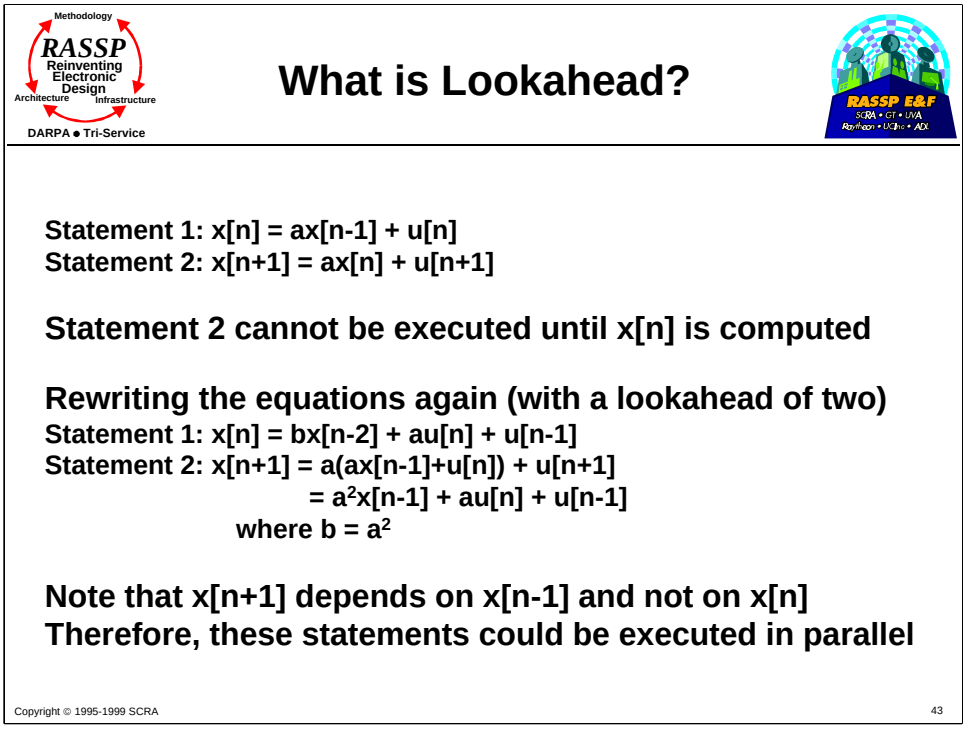
This example shows that unfolding is sufficient but not necessary for rate optimal schedules. Unfolding also is NOT as efficient.

Module Outline

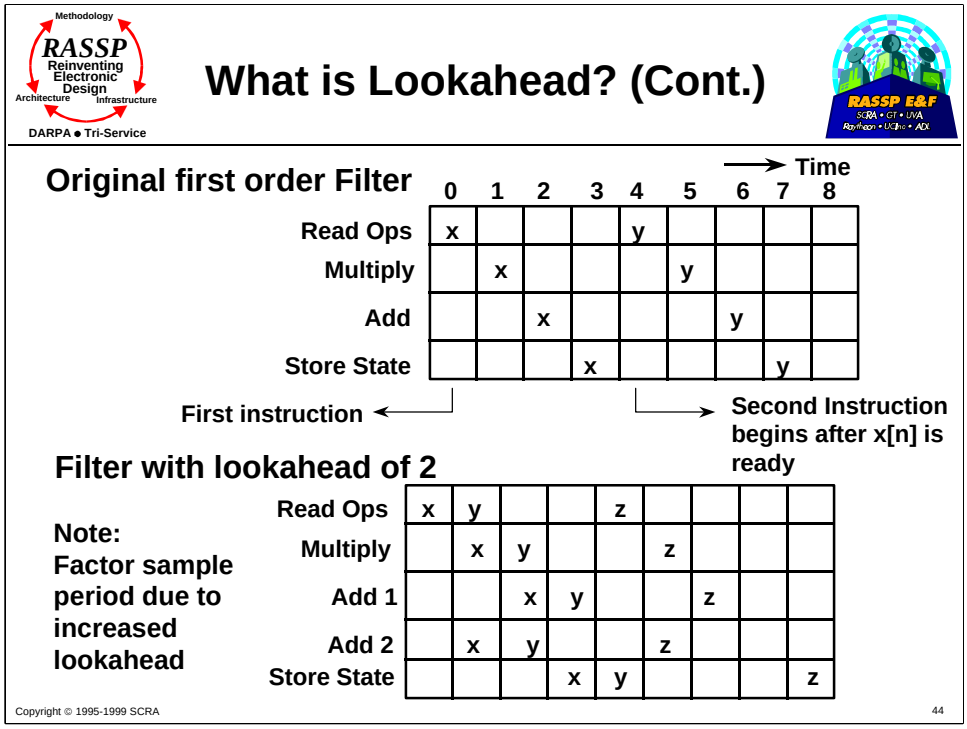
- Introduction
- FSFG as a Representation of DSP Algorithms
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- Formal Methods for FSFGs
- Perfect-rate FSFGs and Unfolding
- **Lookahead**
- Scheduling Approaches
- Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- Summary



Lookahead is best illustrated by the example that follows.



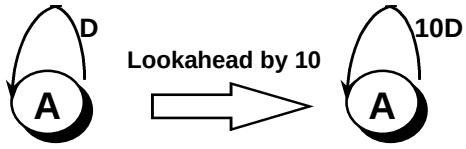
Note that in lookahead, we precompute intermediate steps of future values, thus reduces the iteration period bottleneck through dependencies between successive iterations.



The crosses mark the time units (along x-axis) when the operation is active.

What is Lookahead? (Cont.)

- **Lookahead reduces the IPB, but at the cost of additional hardware (adders and multipliers)**
- **Lookahead adds delay (or memory) into the feedback loop**
  - **E.g., x[n] depends on x[n-2] => Two delays**
    x[n] depends on x[n-1] => One delay
- **Iteration period bound = $\frac{\text{Total Computational Latency in Loop}}{\text{Total Number of Delays}}$**
  - **Lookahead increases the denominator**
- **Alternative**

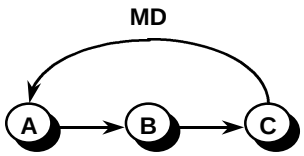


Lookahead improves the sample period bound at the cost of additional hardware (similar to the carry lookahead adder).

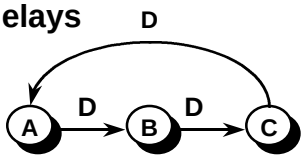
Flowchart

Flowchart for optimization with Lookahead and Retiming

- **Step 1: Use Lookahead by M**



- **Step 2: Retime to distribute delays**

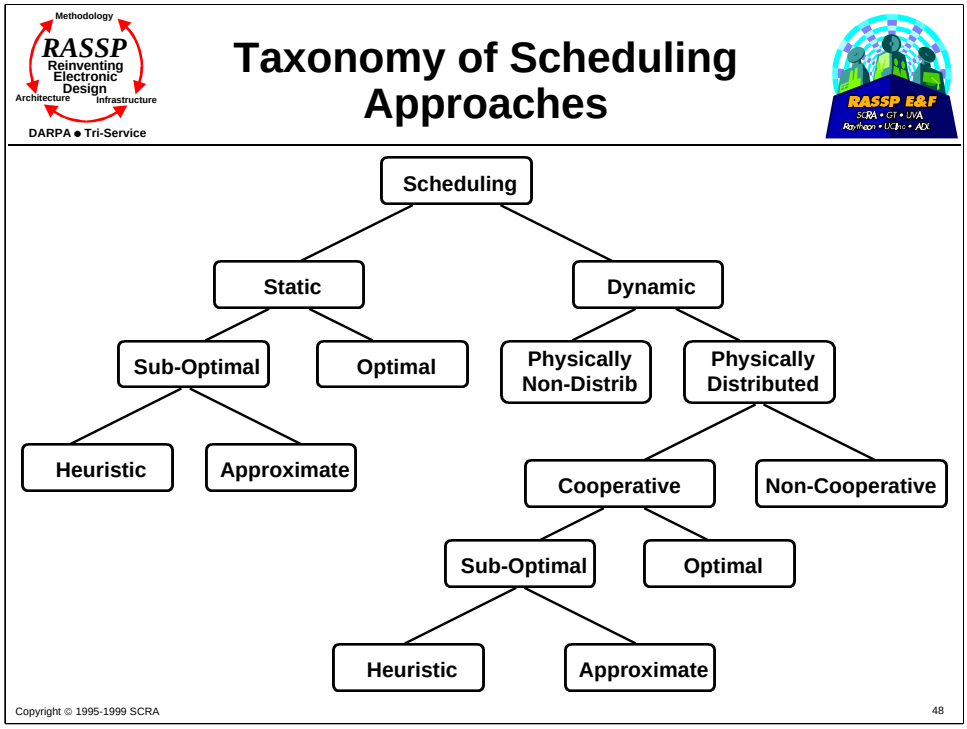


- **Step 3: Obtain a static rate-optimal schedule**

The above three-step procedure leads to optimal schedules.

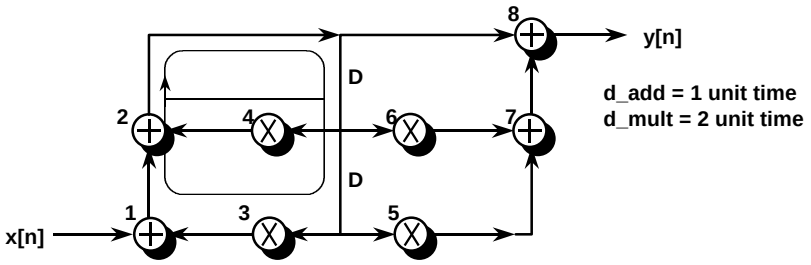
Module Outline

- Introduction
- FSFG as a Representation of DSP Algorithms
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- Formal Methods for FSFGs
- Perfect-rate FSFGs and Unfolding
- Lookahead
- **Scheduling Approaches**
- Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- Summary



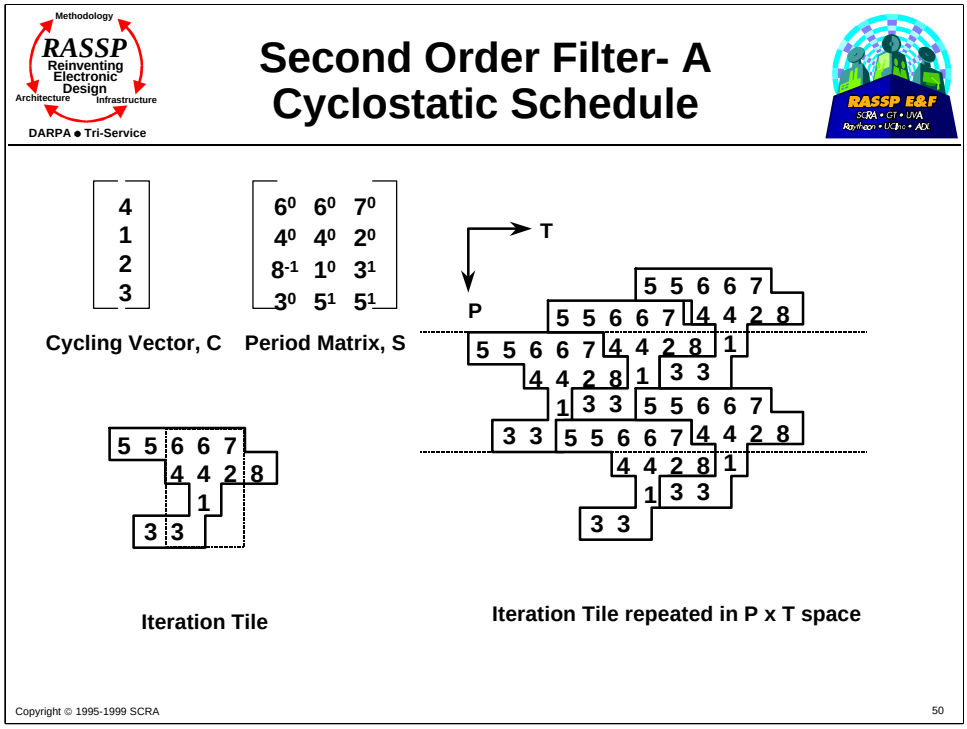
This chart lists various possible heuristic and non-heuristic methods.

FSFG for Second Order IIR Filter



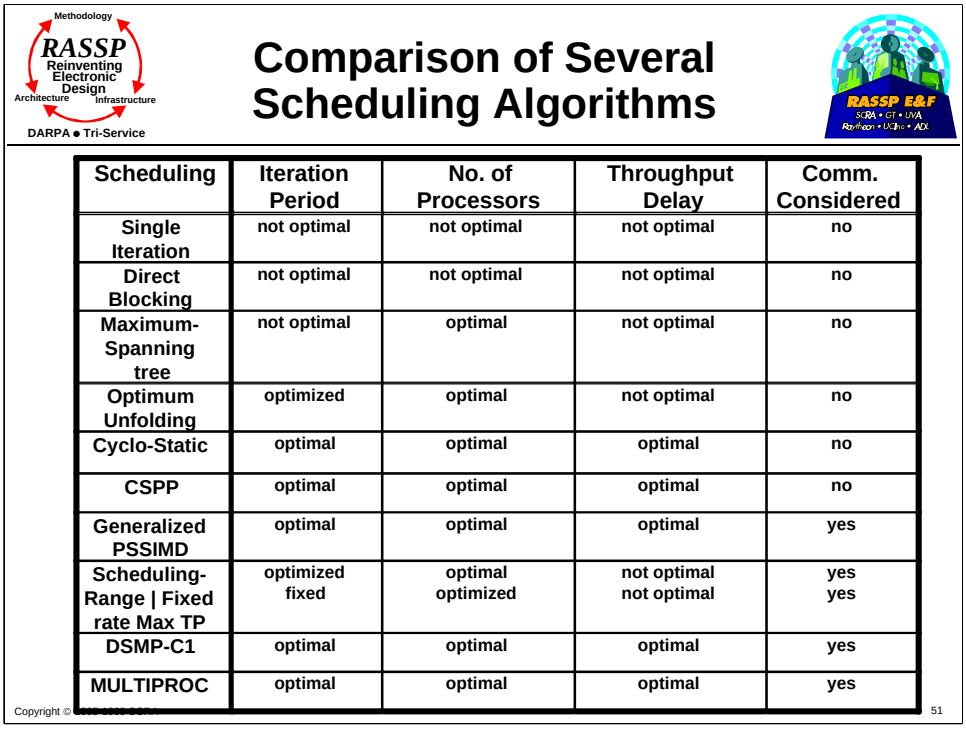
Flow Graph Bound	Definition	Value
Iteration Period Bound	$IPB = \max \left\lceil \frac{\Sigma_{j \varepsilon l} d_j}{n_l} \right\rceil, l \varepsilon L$	3
Period Delay Bound	$D_{i/o} = \max \left\lceil \Sigma_{j \varepsilon p} d_j - n_p IPB \right\rceil, r \varepsilon P$	3
Iteration Period Bound	$PB = \left\lceil \frac{\Sigma_{j \varepsilon l} d_j}{IPB} \right\rceil$	4

We now introduce methods for non-ideal systems (e.g., communication delays and finite resources).



Recapitulation of scheduling. The cycling vector and the period matrix together determine the P x T tiling.

The superscripts represent the iteration numbers.

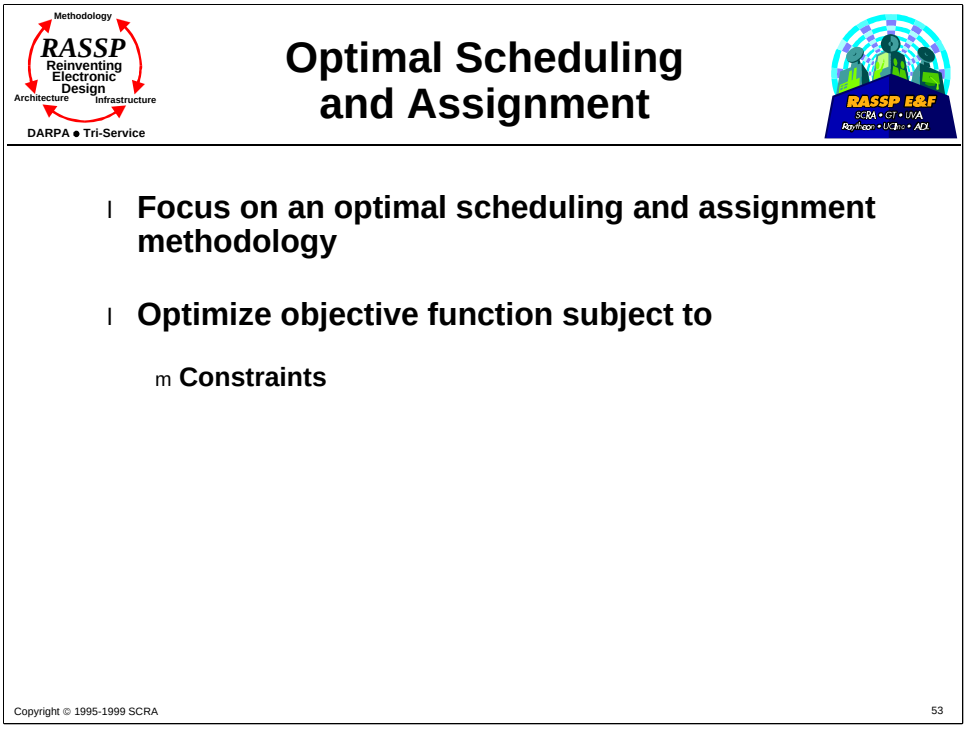


Scheduling	Iteration Period	No. of Processors	Throughput Delay	Comm. Considered
Single Iteration	not optimal	not optimal	not optimal	no
Direct Blocking	not optimal	not optimal	not optimal	no
Maximum-Spanning tree	not optimal	optimal	not optimal	no
Optimum Unfolding	optimized	optimal	not optimal	no
Cyclo-Static	optimal	optimal	optimal	no
CSPP	optimal	optimal	optimal	no
Generalized PSSIMD	optimal	optimal	optimal	yes
Scheduling-Range | Fixed rate Max TP	optimized fixed	optimal optimized	not optimal not optimal	yes yes
DSMP-C1	optimal	optimal	optimal	yes
MULTIPROC	optimal	optimal	optimal	yes

Note that we now have a formal approach of comparing various scheduling methods based on our optimality criteria (See [Madisetti95]).

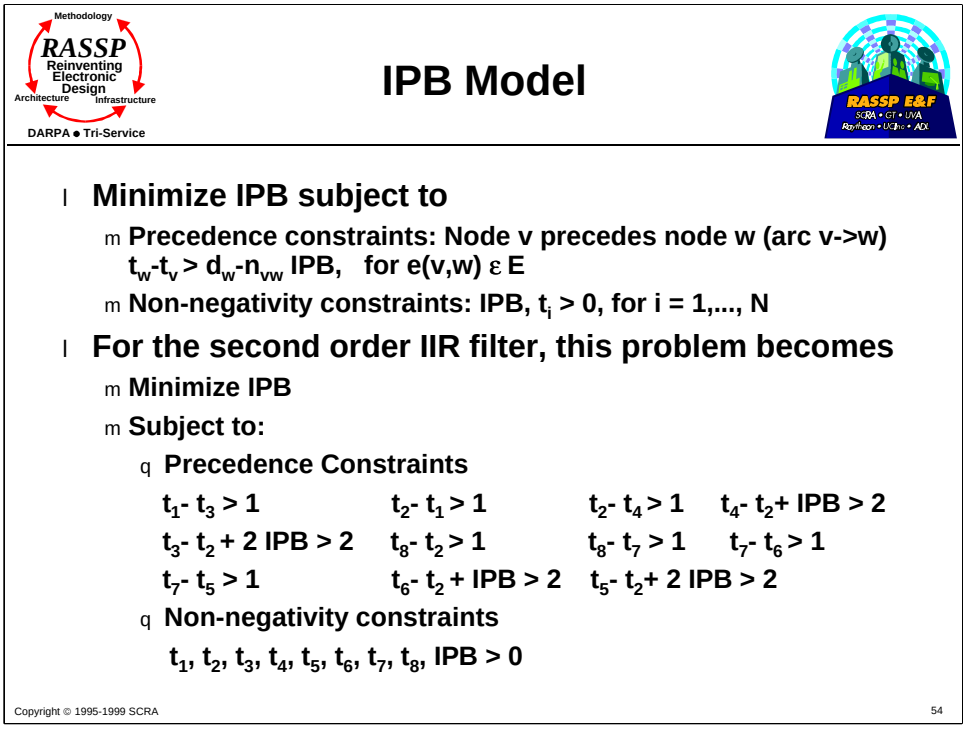
Module Outline

- Introduction
- FSFG as a Representation of DSP Algorithms
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- Formal Methods for FSFGs
- Perfect-rate FSFGs and Unfolding
- Lookahead
- Scheduling Approaches
- **Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources**
- Summary

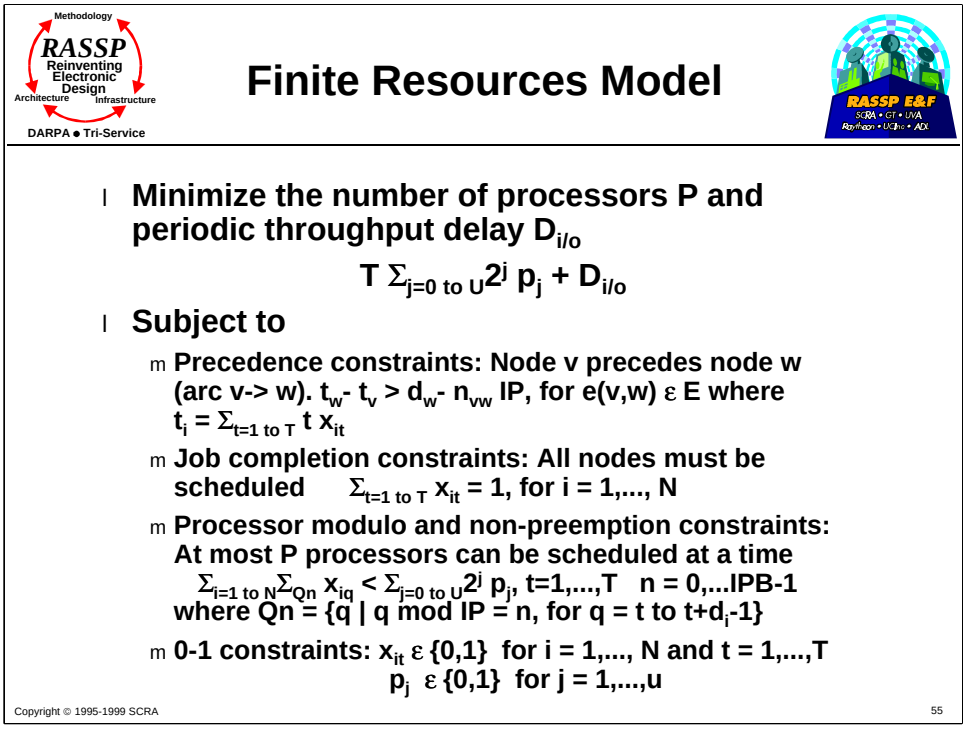


This optimization-based methodology can be found in

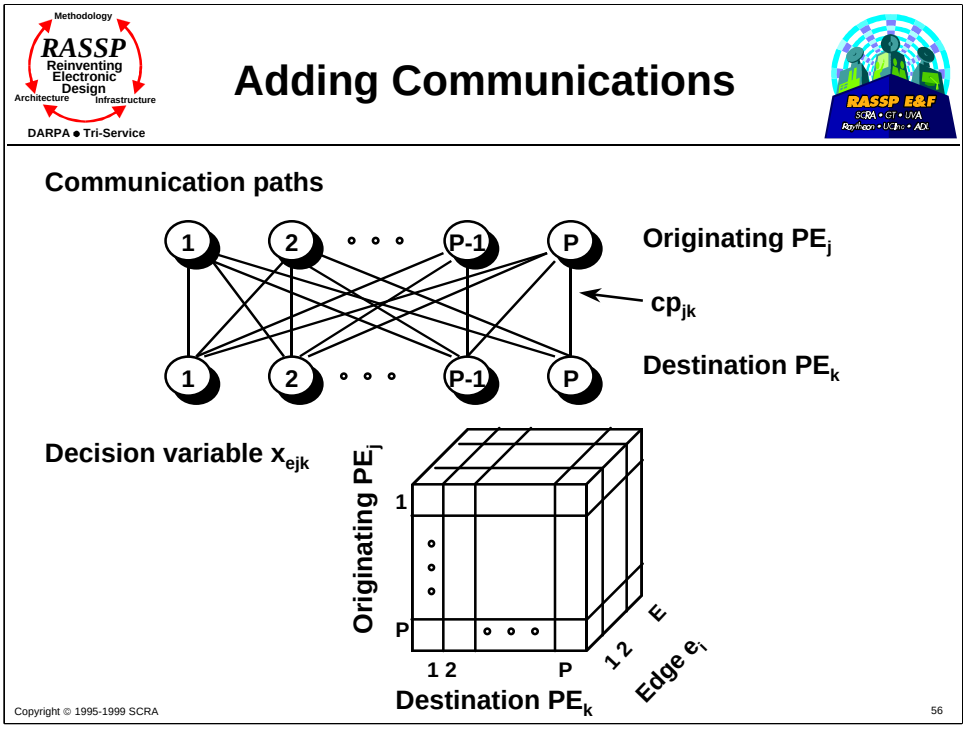
[Madisetti95] V.Madisetti, “VLSI Digital Signal Processors”, IEEE Press, 1995.



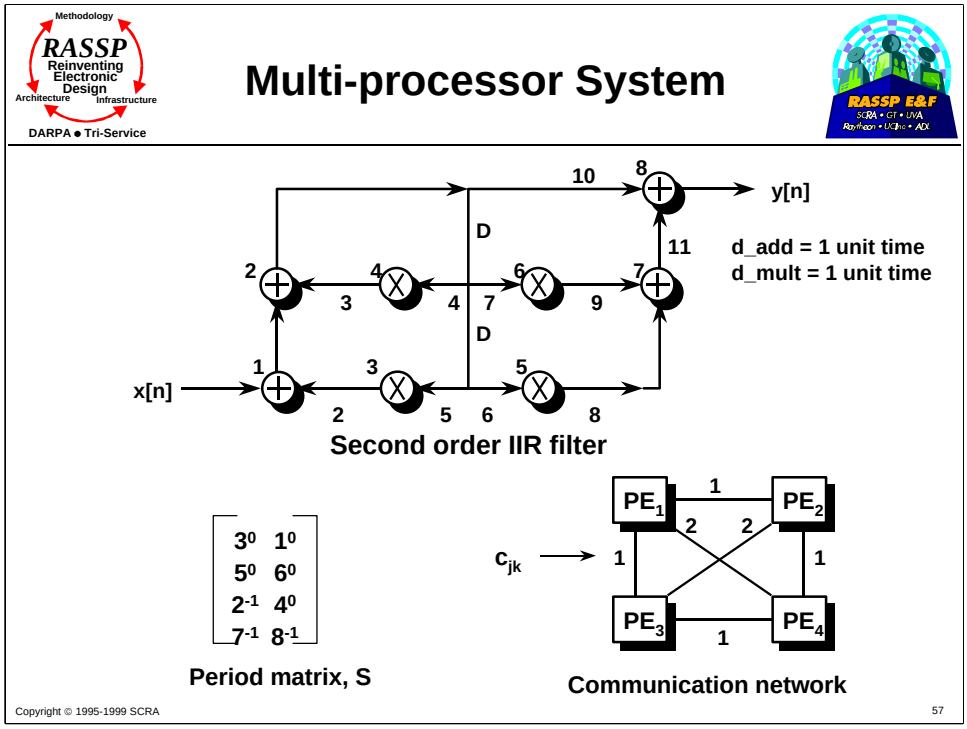
We illustrate the typical scheduling problem as an integer programming problem. Efficient methods exist to solve integer programs.



We can also solve the scheduling problem for a finite number of processors.



The real benefit of this approach comes when it handles non-zero communication costs.



We now show how one can map a schedule on a target multiprocessor architecture with known costs.

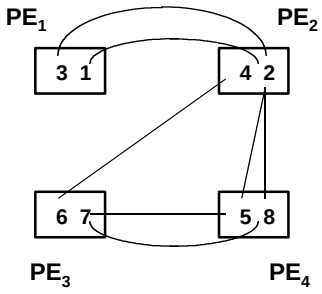
Second Order IIR Filter

Edge	Comm. Path
1	1 --> 2
2	1 --> 1
3	2 --> 2
4	2 --> 2
5	2 --> 1
6	2 --> 4
7	2 --> 3
8	4 --> 3
9	3 --> 3
10	2 --> 4
11	3 --> 4

Edge mapping

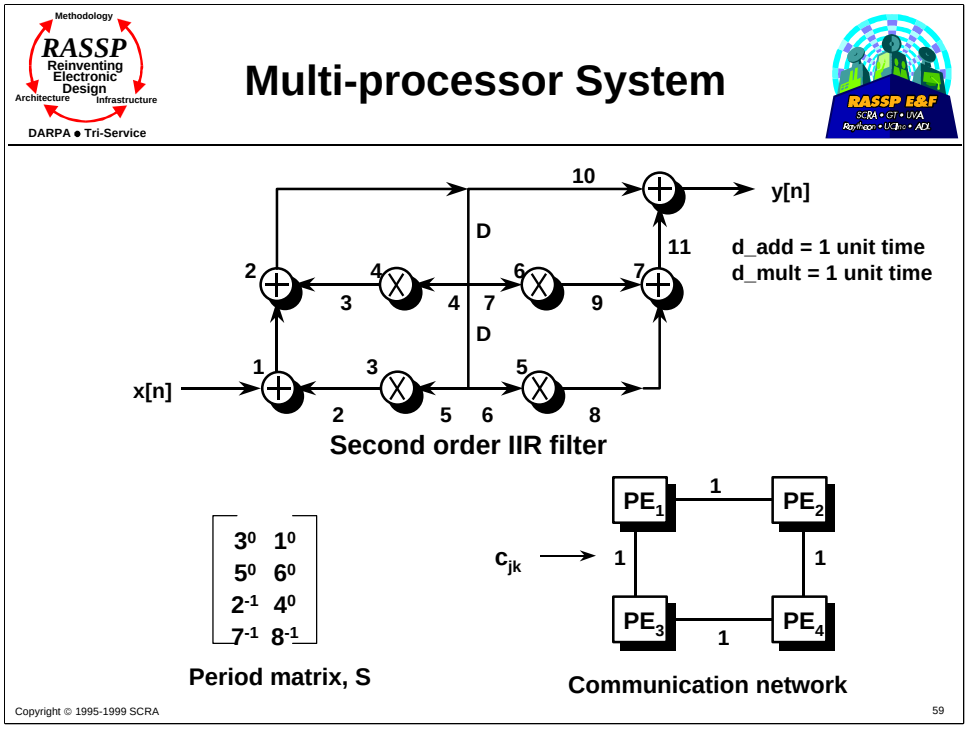
PE	Nodes
1	3, 1
2	4, 2
3	6, 7
4	5, 8

PE assignment



Communications

Note the two steps involved in the edge mapping and PE assignment.

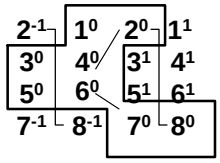


Here communication costs can be explicitly assigned.

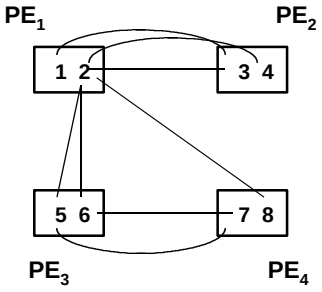
Randomly Connected Cost (RCC) Model for Second Order IIR Filter

Edge	Comm. Path
1	1 --> 1
2	2 --> 1
3	2 --> 1
4	1 --> 2
5	1 --> 2
6	1 --> 3
7	1 --> 3
8	3 --> 4
9	3 --> 4
10	1 --> 4
11	4 --> 4

Edge mapping



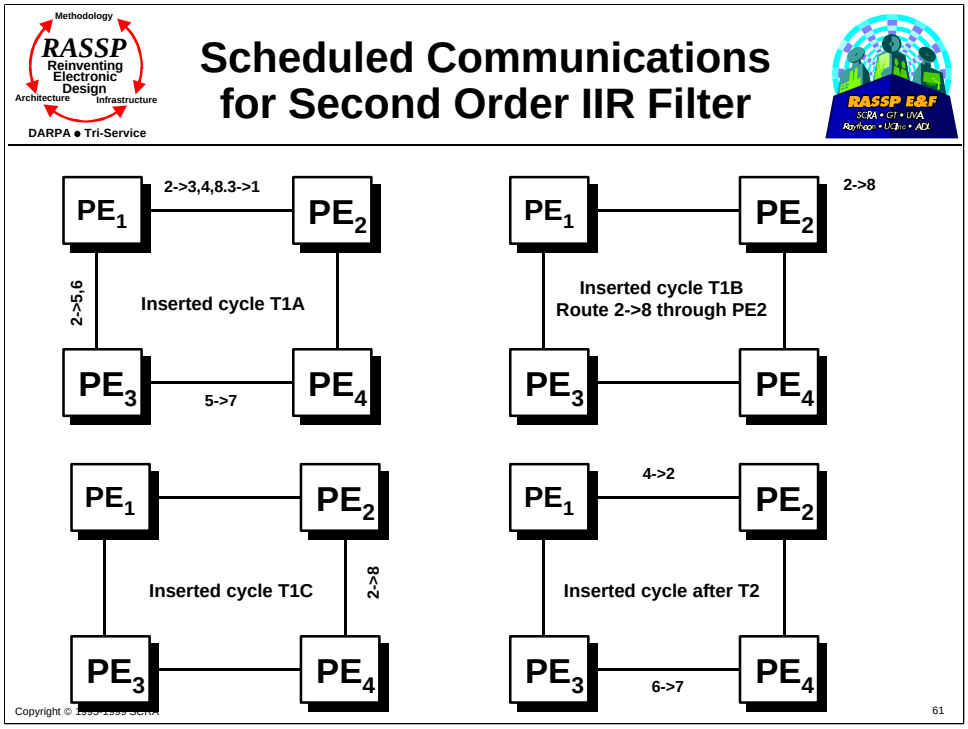
Comm. that increase the IP



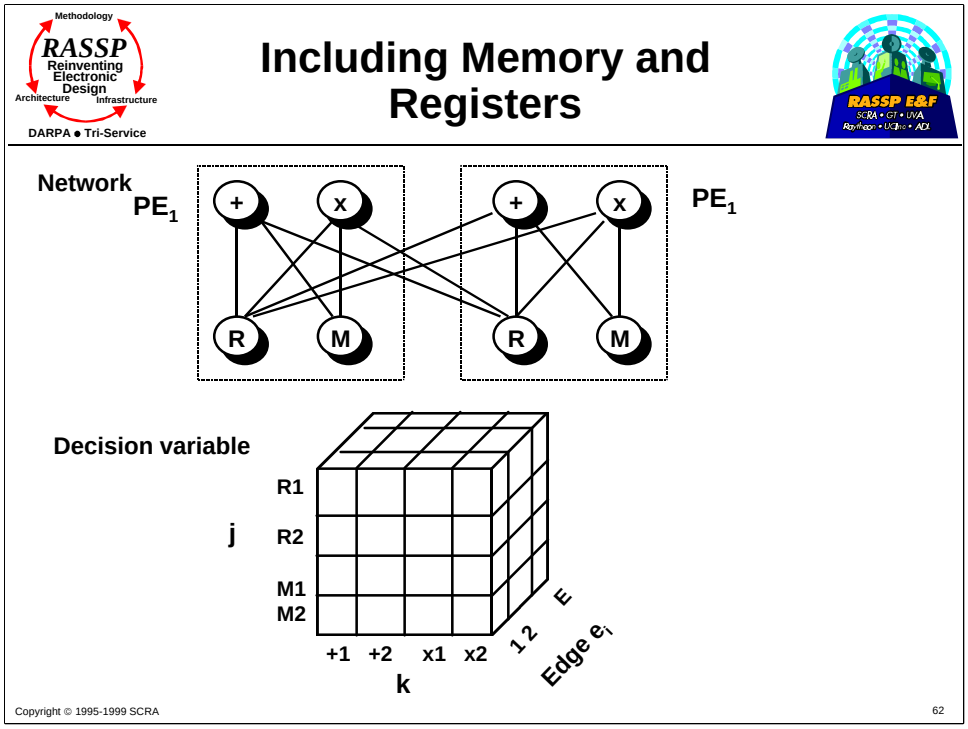
Communications

RCC Model = Randomly Connected Cost Model

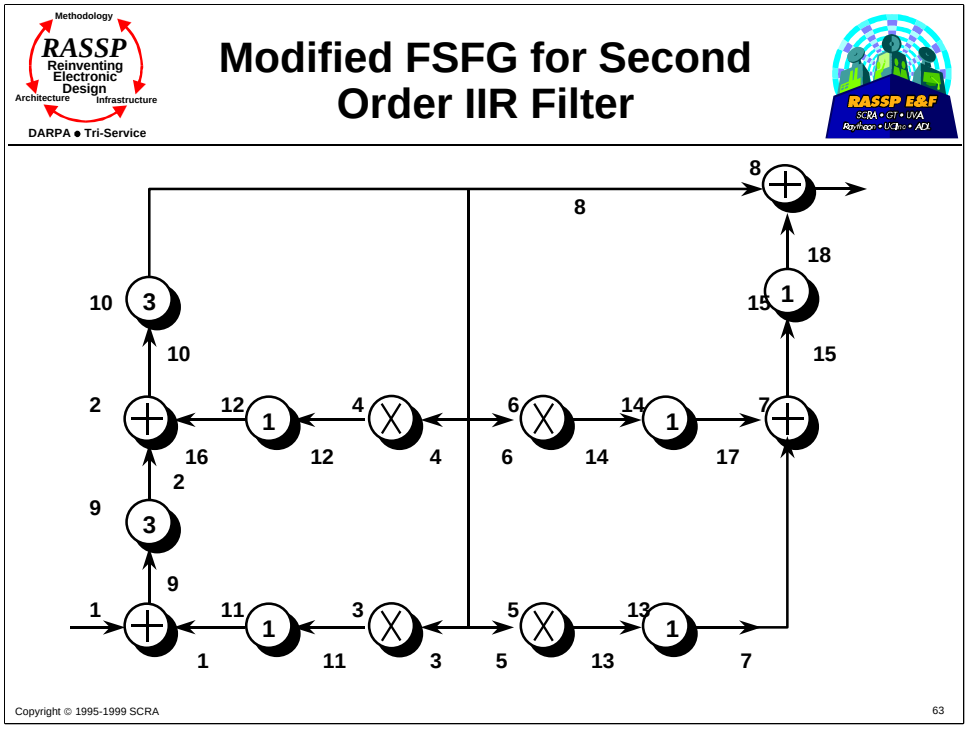
RCC = Randomly Connected Cost Model.



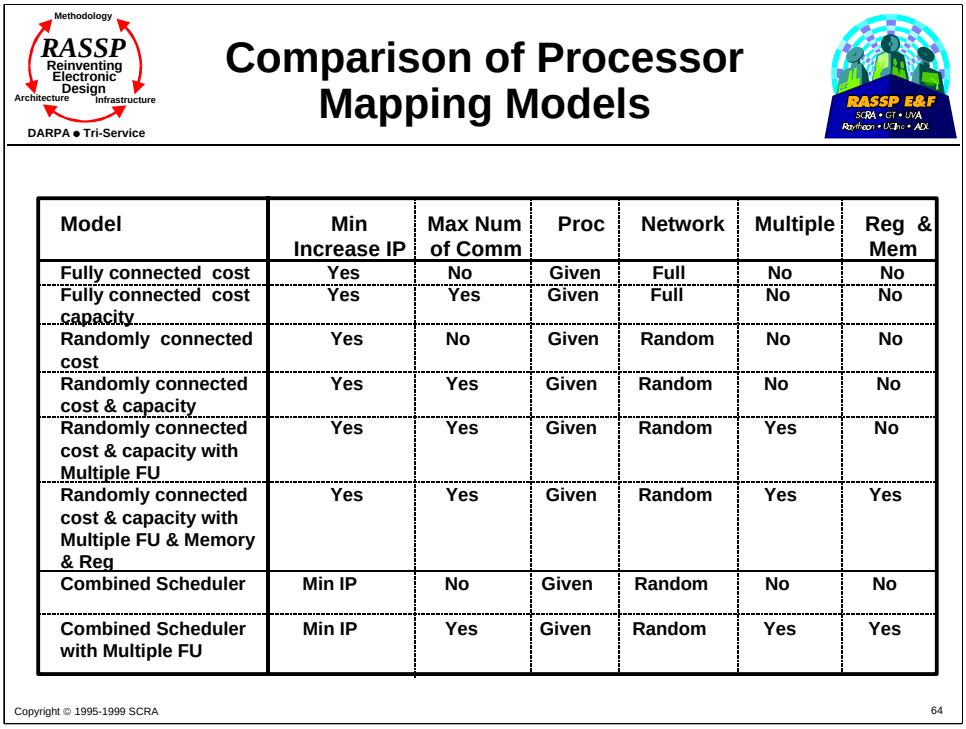
All communications are strictly scheduled due to the deterministic nature.



Memory and storage can also be included. This implies that the same framework can be used for synthesis.



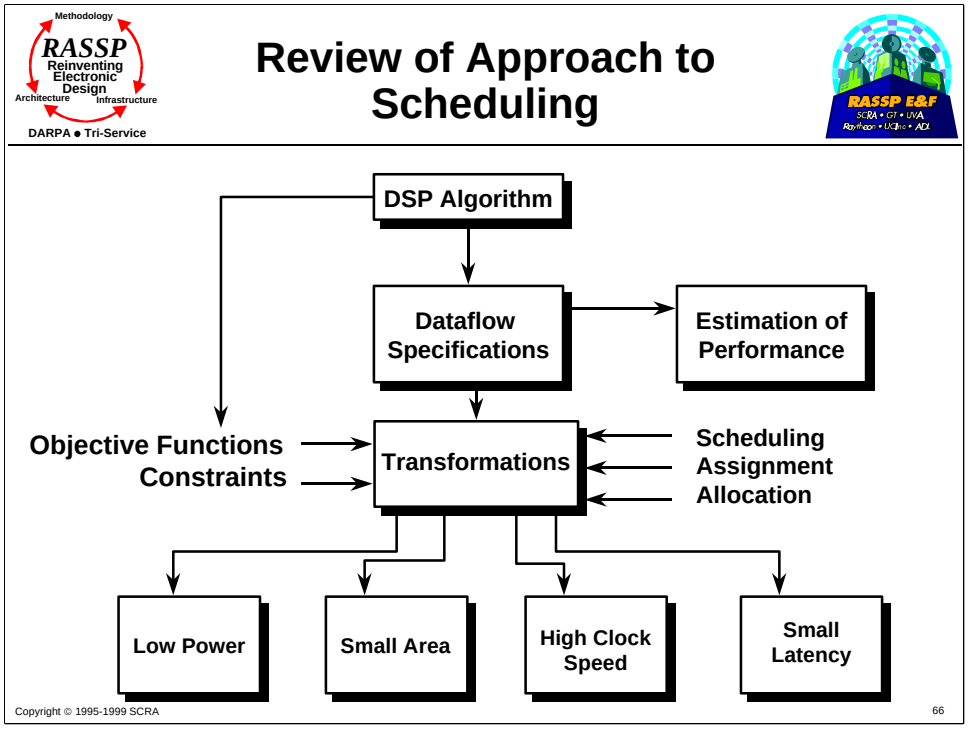
We now arrive at this intermediate FSFG that can be used as the input to the next steps in the scheduling process.



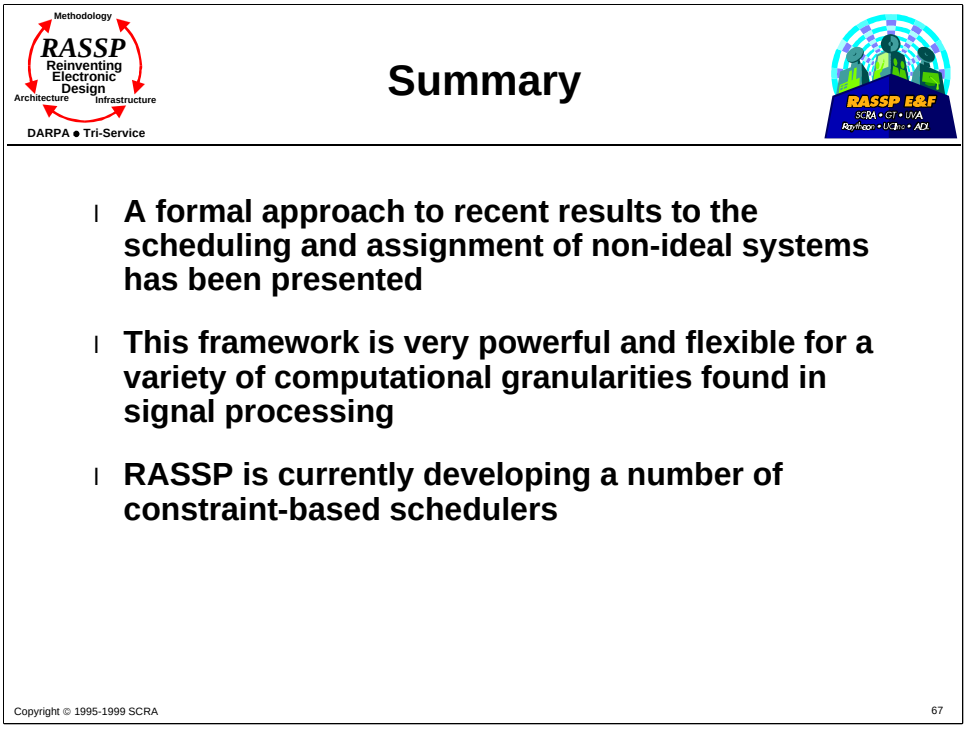
Summary of the constrained scheduling problem.

Module Outline

- Introduction
- FSFG as a Representation of DSP Algorithms
- Measures of Performance
- Retiming
- Iteration Period and Iteration Period Bounds
- Formal Methods for FSFGs
- Perfect-rate FSFGs and Unfolding
- Lookahead
- Scheduling Approaches
- Optimal Scheduling and Assignment Considering Communication Delays and Finite Resources
- **Summary**



We have presented a concise but complete introduction to the scheduling, assignment, and allocation problem for multiprocessor systems (both ideal and non-ideal cases).



RASSP has utilized the use of graphical driven front-end approaches that capture the algorithmic specification. In RASSP the building blocks within the FSFG are of a coarser granularity (e.g., FFTs as opposed to multiply/adds), but the results developed in this module do not change.

References

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

[Madisetti95] V.Madisetti, "VLSI Digital Signal Processors", *IEEE Press*, 1995; © IEEE 1995

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