OpenSPARC Slide-Cast

In Twelve Chapters
Presented by OpenSPARC designers, developers, and programmers
• to guide users as they develop their own OpenSPARC designs and
• to assist professors as they teach the next generation

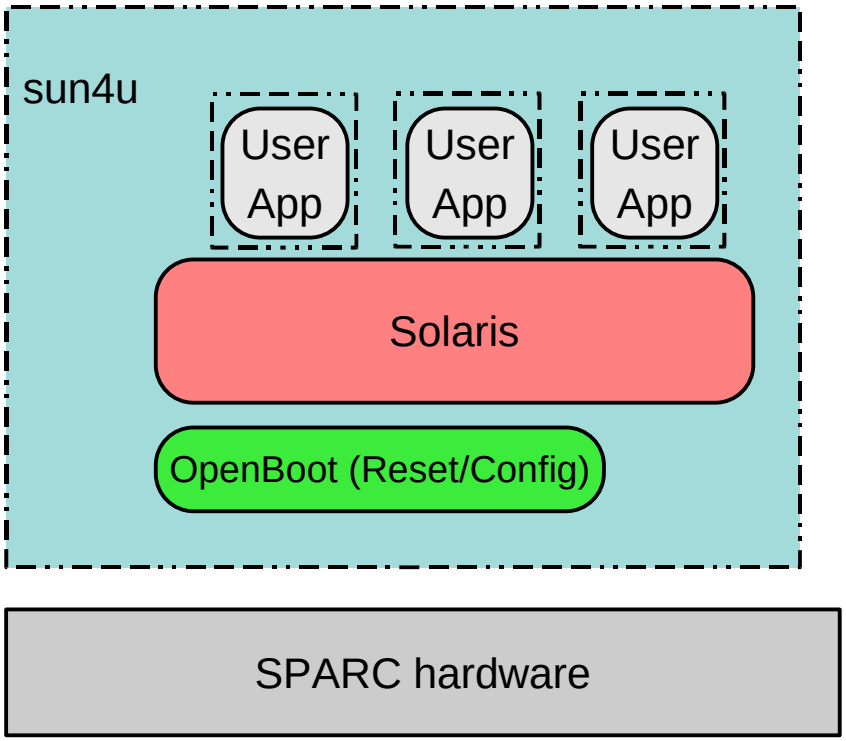
Chapter Nine

HYPERVISOR AND VIRTUALIZATION

Maran Wilson
Software Developer
Sun Microsystems

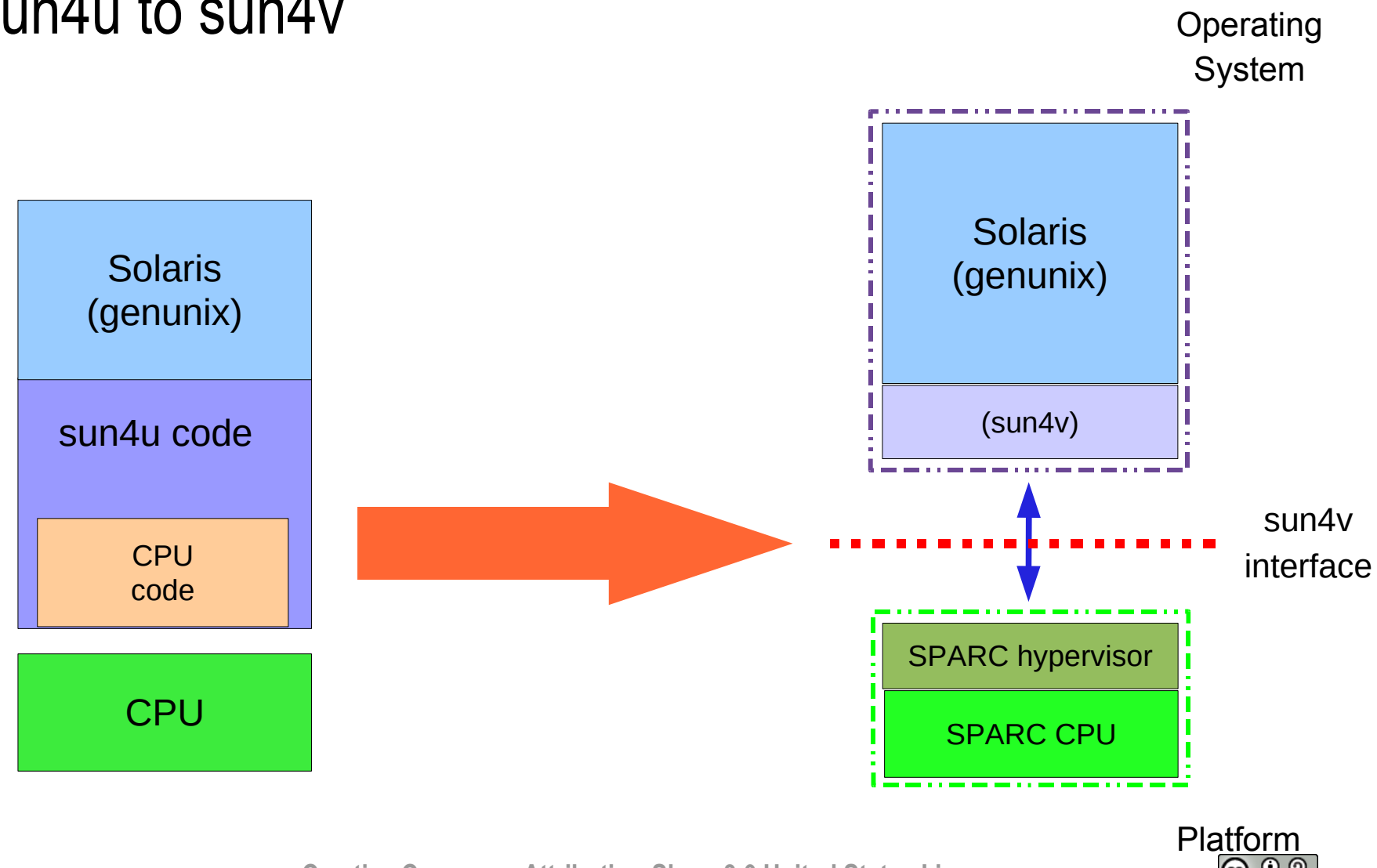
Sun4u Software/Firmware stack

- In the 'good old days' of sun4u, SW/FW was pretty simple.
  - FW would Power on cpu, init registers, hand off to Solaris.
  - Solaris reboots, do a clean fresh reset of the HW



Sun4v Virtualization for SPARC platforms

- sun4u to sun4v



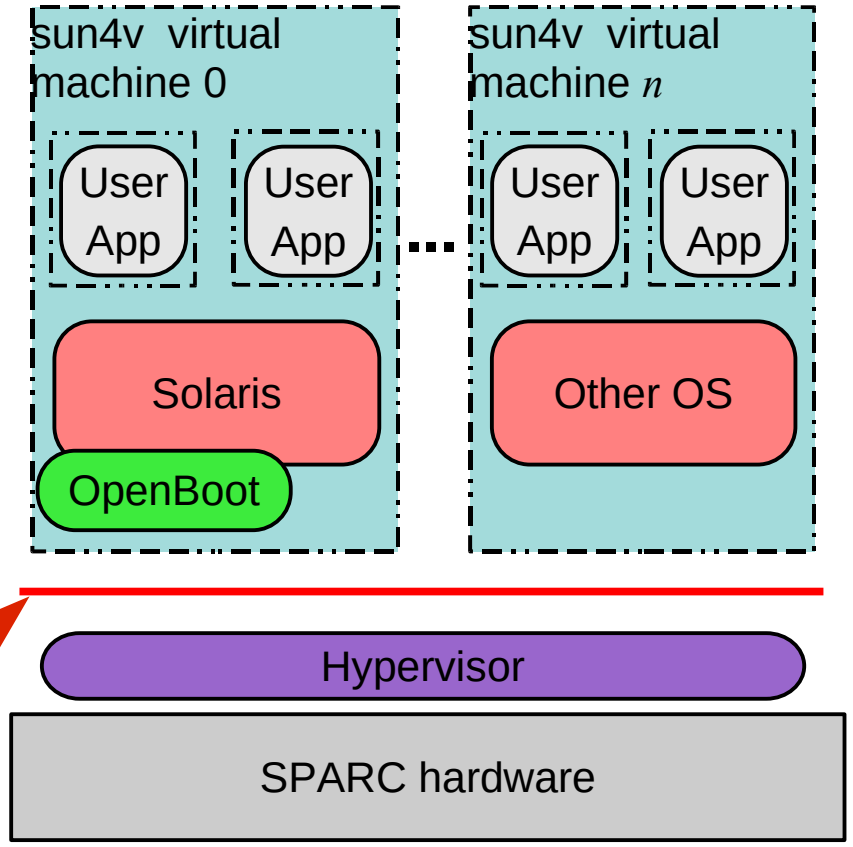
Virtual Machine approaches

- True virtualization
  - Emulate machine architecture down to register level
  - Can run any OS / software unmodified
  - Performance penalty
- Para-virtualization
  - Define a SW interface to hypervisor
  - Requires OS modification
  - More efficient implementation - no emulation required
- Sun's Approach
  - A hybrid - the processors are designed for the sun4v privileged model, but the hypervisor impedance matches anything not handled by the hardware, and there are sun4v APIs implemented by the hypervisor.

Virtual Machine for SPARC

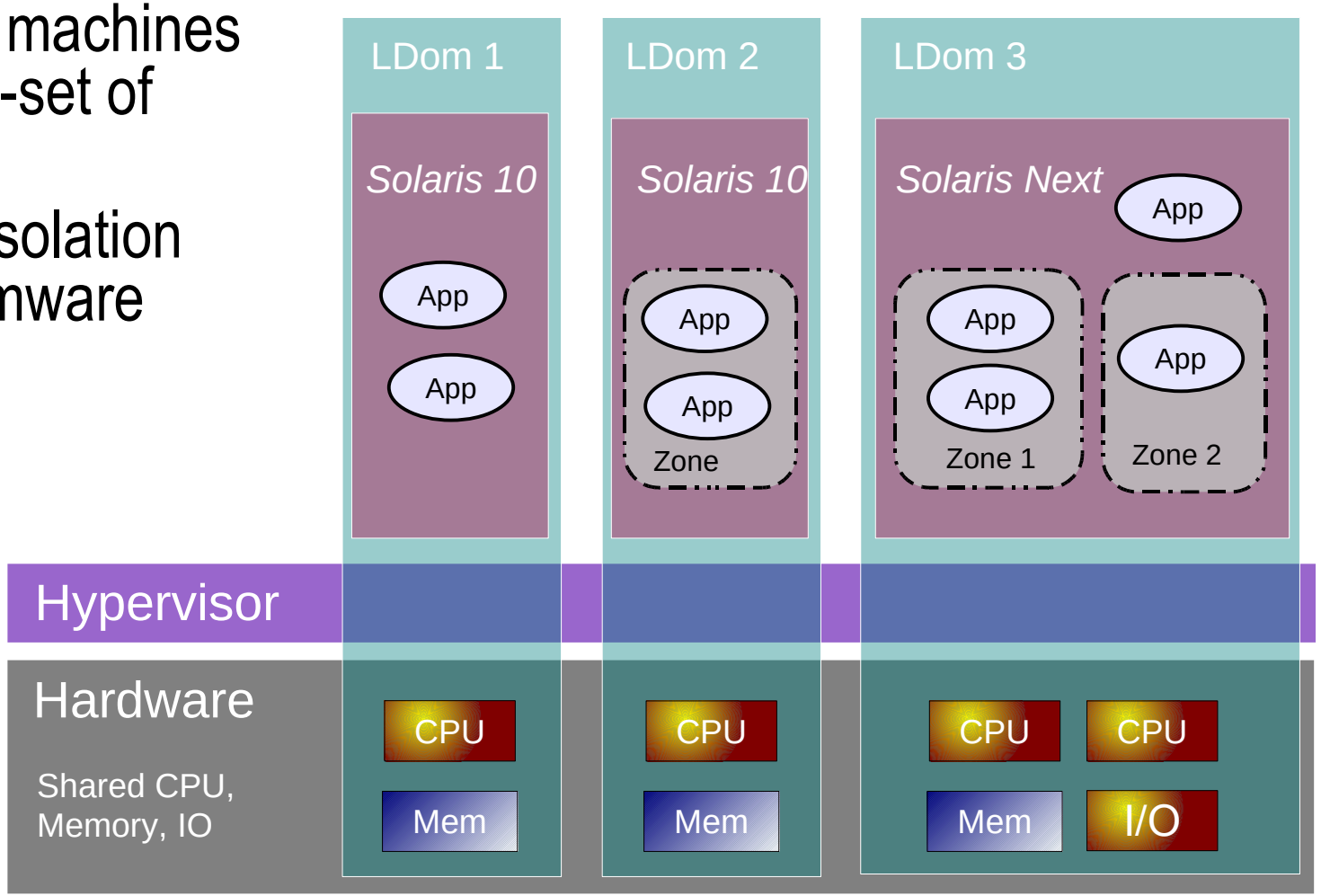
- Thin software layer between OS and platform hardware
- Para-virtualised OS
- Hypervisor + sun4v interface
  - Virtualises machine HW and isolates OS from register-level
  - Delivered with platform not OS
  - Not itself an OS

stable interface "**sun4v**"



Logical Domains

- Partitioning capability
  - > Create virtual machines each with sub-set of resources
  - > Protection & Isolation using HW+firmware combination

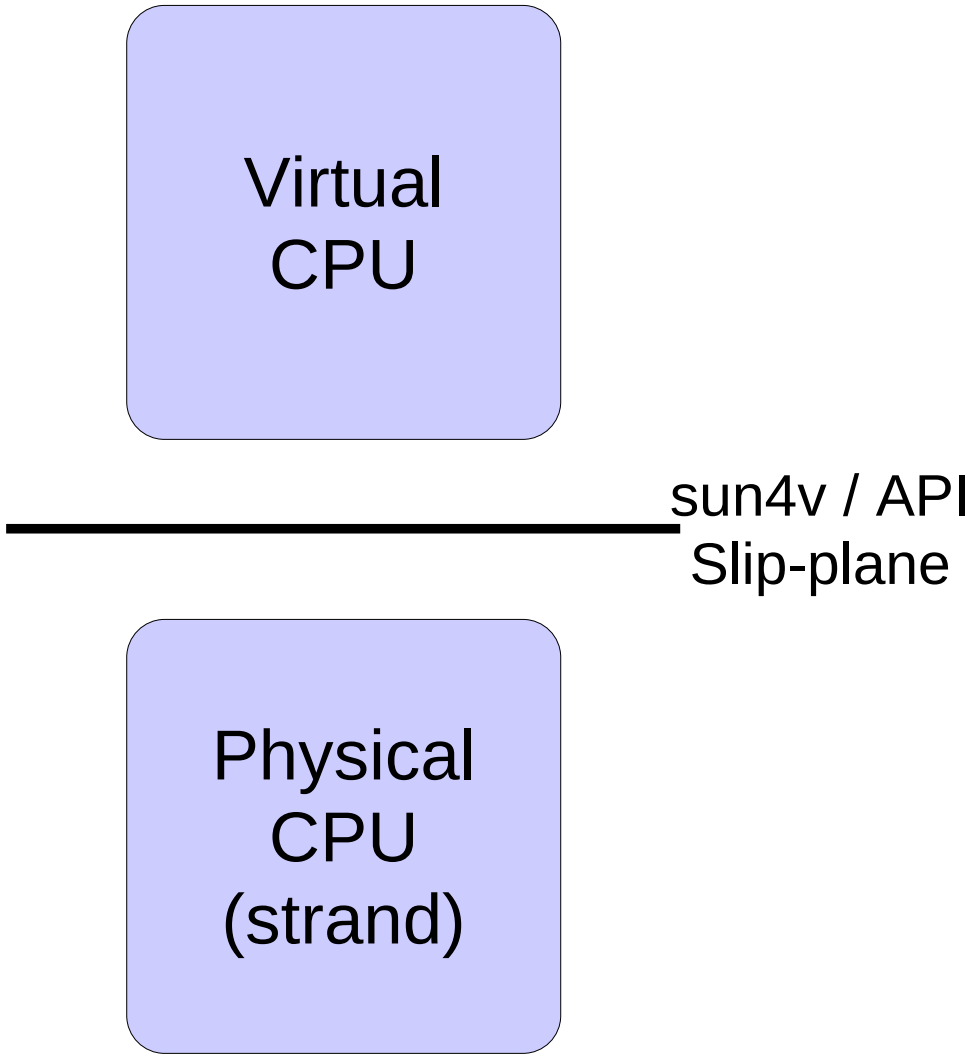


Topics

- CPU changes
- Memory management
- I/O
- Interrupts
  - x-cpu & devices
- Multiple Domains
- Additional Topics
  - Error handling
  - Machine Description
  - Boot process

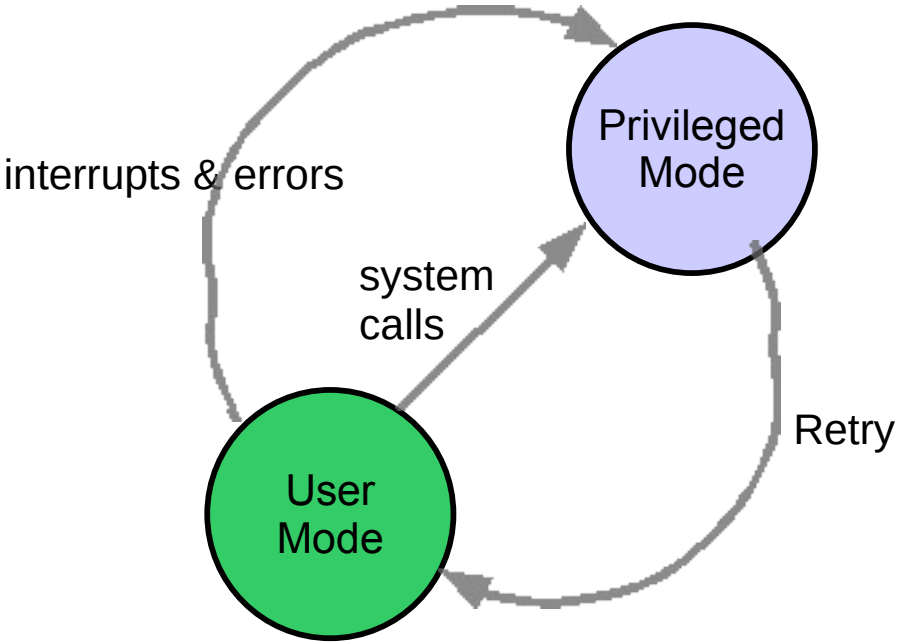
Basic Principles

- Ability to rebind virtual resources to physical components at any time
- Minimal state held in Hypervisor to describe guest OS
- *Never* trust Guest OS

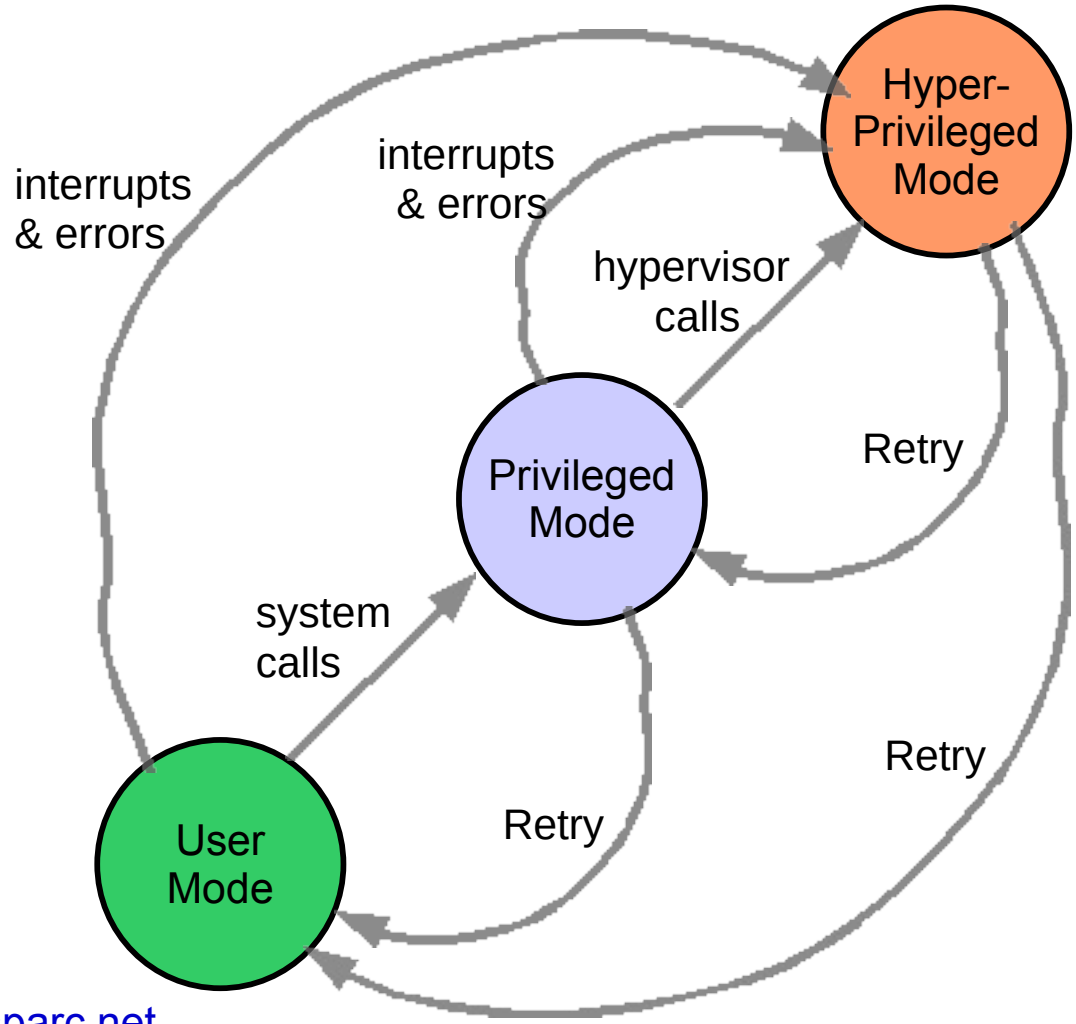


Legacy SPARC execution mode

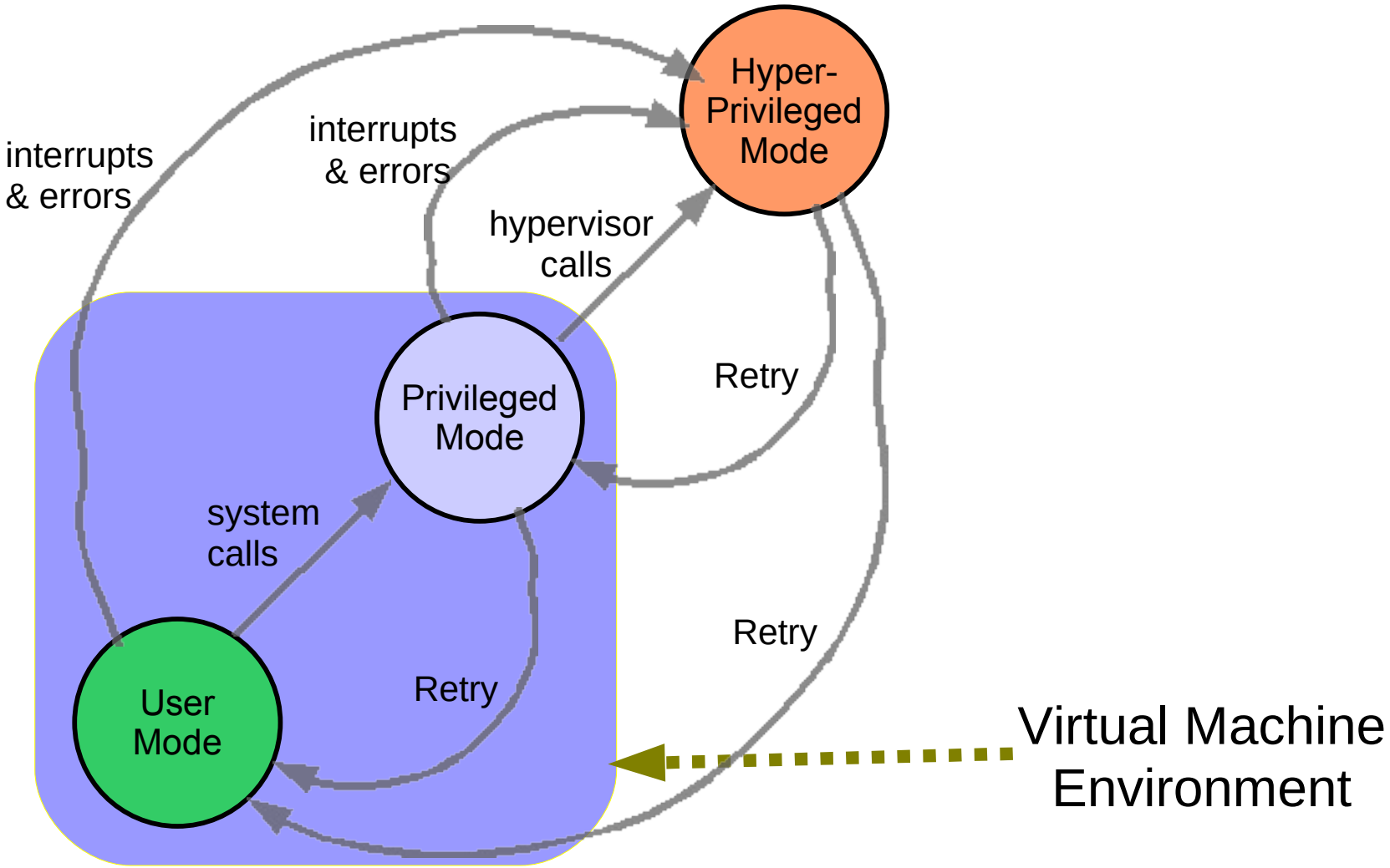
- Older sun4u chips (UltraSPARC I, II, III, IV)



New SPARC Execution mode



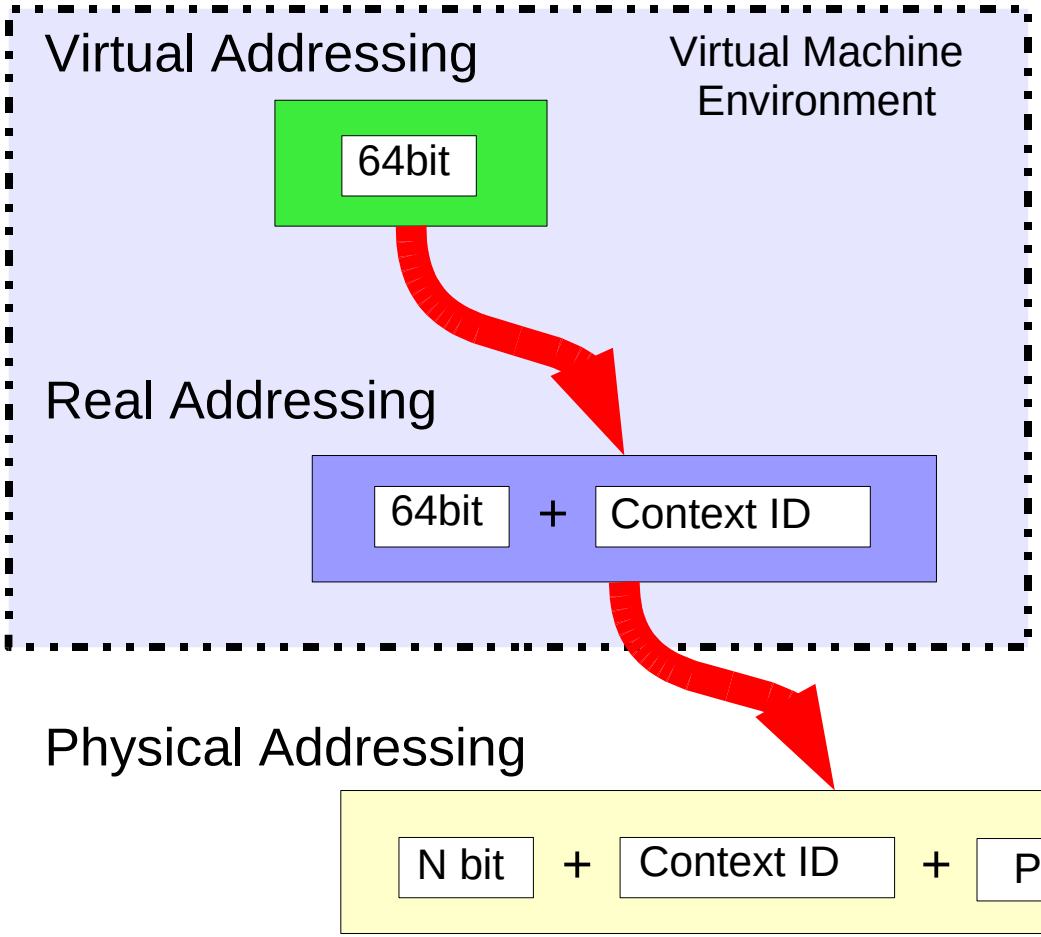
New SPARC Execution mode



Privileged mode constrained

- Close derivative of legacy privileged mode, but:
  - No access to diagnostic registers
  - No access to MMU control registers
  - No access to interrupt control registers
  - No access to I/O-MMU control registers
  - All replaced by Hypervisor API calls
- UltraSPARC-ness remains with minor changes
  - timer tick registers
  - softint registers etc.
  - trap-levels & global registers etc.
  - register window spill/fill

Translation hierarchy



User Level

Privileged Level

Hyper-privileged Level

Translation management

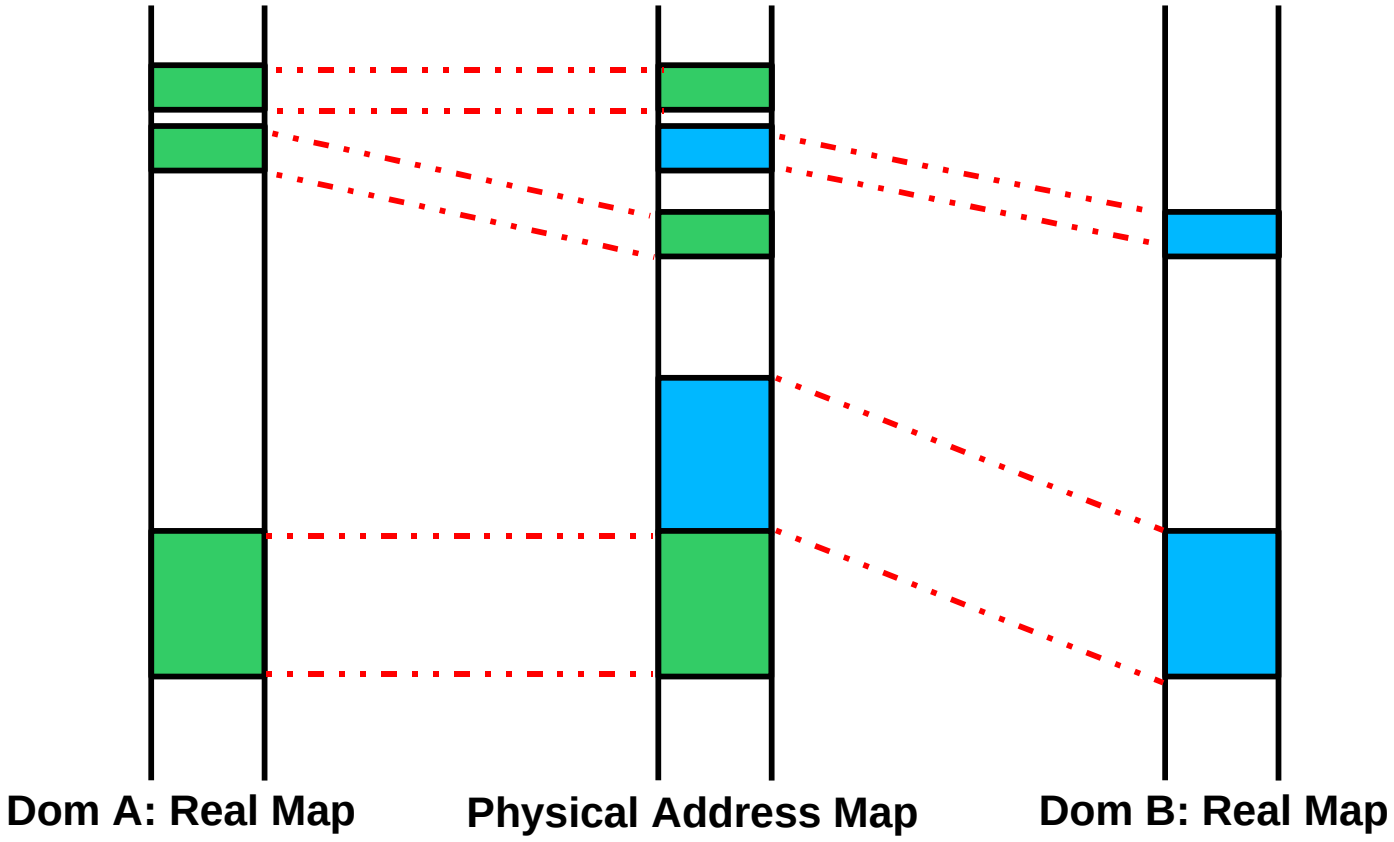
- Guest OS defines a “fault-status” area of memory for each virtual CPU (vCPU)
  - Hypervisor fills in info for each MMU exception
- UltraSPARC does not use page-tables
  - traditionally a software loaded TLB
- Hypervisor APIs to support direct software management of TLB entries
  - map, demap
  - Simple guests like OBP can use this

Translation Storage Buffers

- Guest OS managed cache of translations stored in memory
  - Guest allocates memory for buffer
  - Guest places translation mappings into buffer when needed
  - Hypervisor fetches from this cache into TLBs
- Guest specifies virtual -> real mappings
  - Hypervisor translates real->physical to load into TLB
  - TLB holds virtual -> physical mappings
- Multiple TSBs used simultaneously for multiple page sizes and contexts

Address space control

- Hypervisor limits access to memory and devices -- creating partitions (logical domains)



Virtual I/O devices

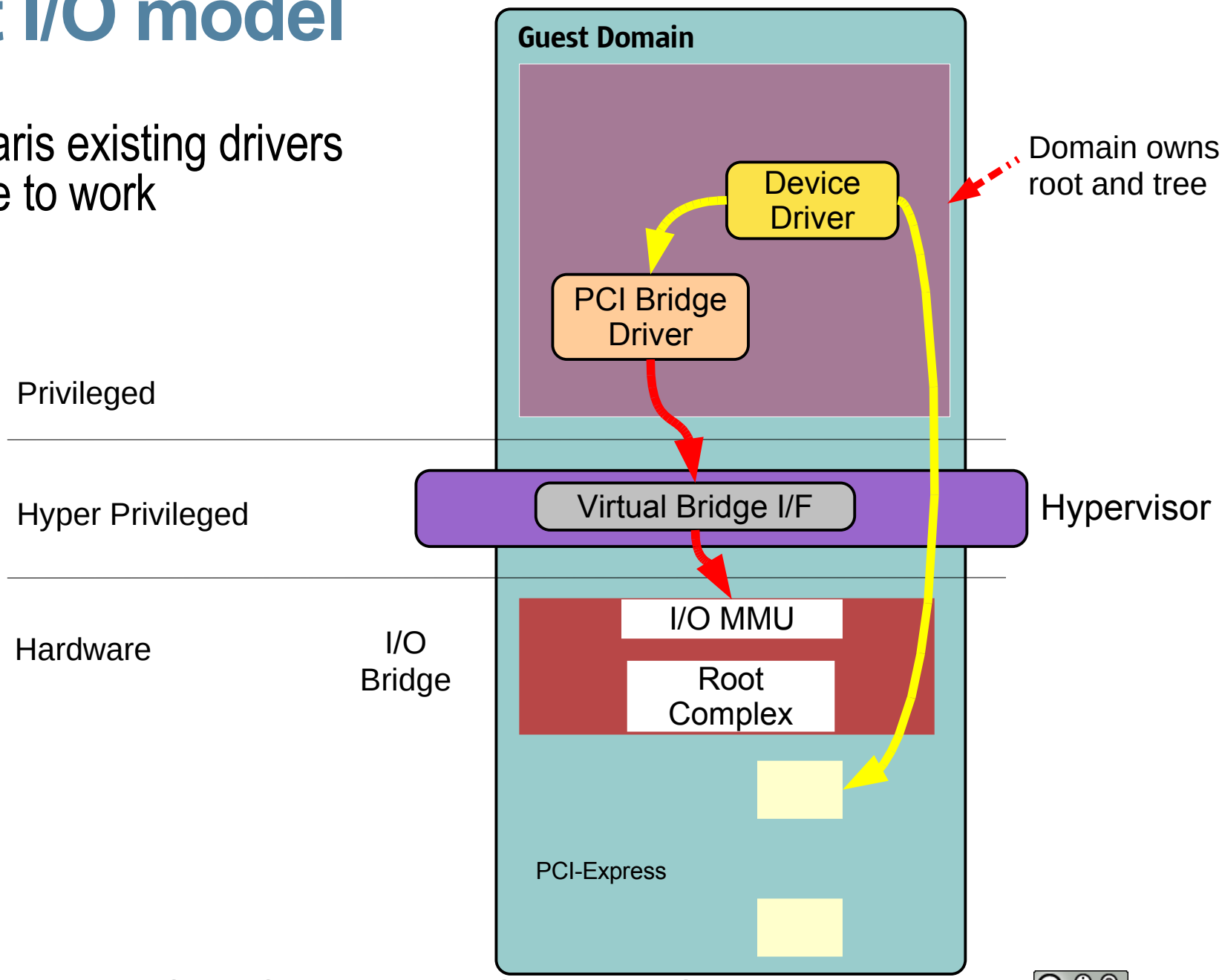
- Provided via Hypervisor
  - e.g. Console - getchar / putchar API calls
  - Hypervisor generates virtual interrupts

Physical I/O devices

- PCI-Express root complex mapped into real address space of guest domain
- Direct access to device registers
  - OBP probes and configures bus and devices
- I/O Bridge and I/O MMU configuration virtualized by hypervisor APIs
  - Ensures that I/O MMU translations are validated by hypervisor
  - Device interrupts are virtualized for delivery

Direct I/O model

- For Solaris existing drivers continue to work

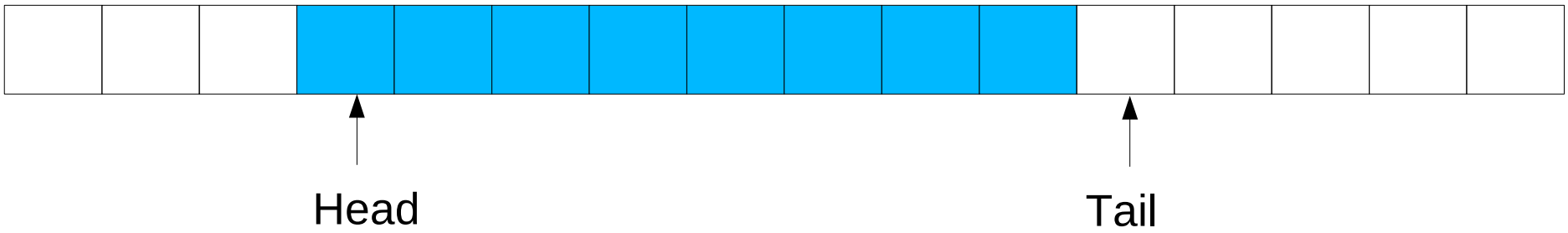


Interrupt and event delivery

- Device interrupts and error events need a mechanism to asynchronously cause an exception for a virtual CPU
- Typically also require some data to identify reason and source and notification
- How to do this in an abstract manner?
  - What if the virtual CPU is not currently bound to a physical CPU?
  - Can't block physical interrupt source until virtual CPU is rescheduled

Interrupts & CPU mondos

- Delivered to privileged mode via four in-memory FIFO queues:
  - cpu-mondo dev-mondo resumable error non-resumable error
- 64-byte entries carry cause information (interrupt numbers)
- Head and Tail offsets available as CPU registers to privileged code
  - Tail manipulated by hypervisor, head by guest OS
  - For either queue, head != tail causes trap

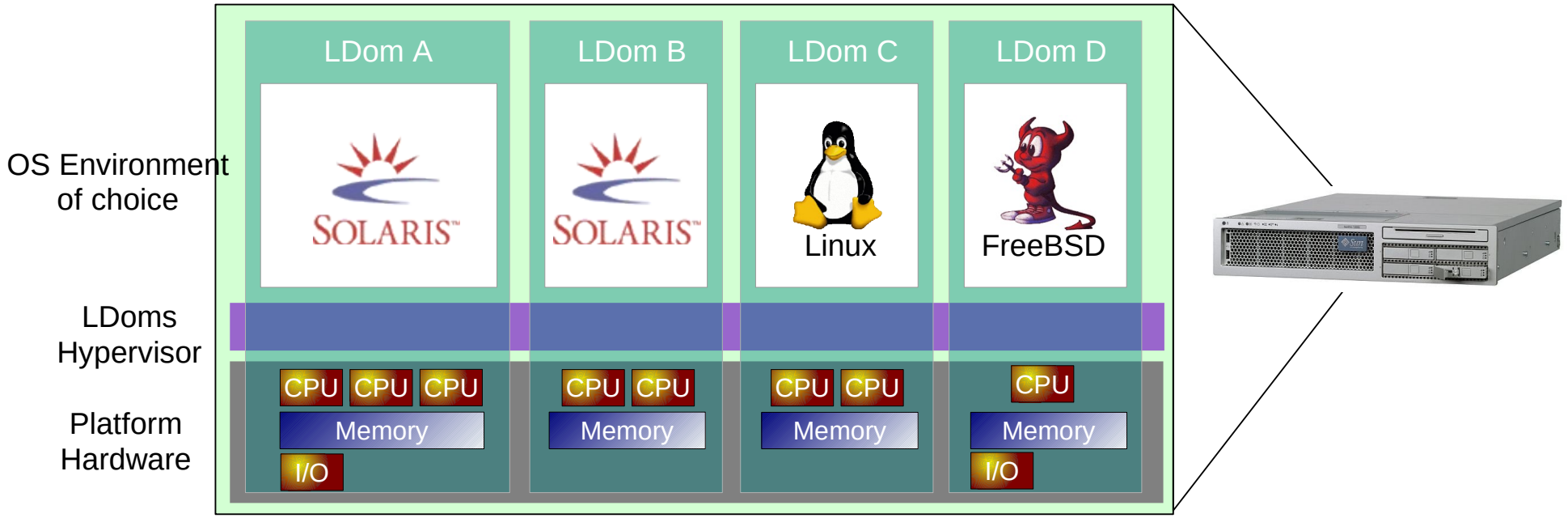


Queue constraints

- Must be a power-of-2 number of entries, minimum of 2
  - Entries always 64 bytes in size
  - (tail+1)%size == head defines full state
- Must be aligned on a real address boundary identical to Q size
  - Designed to make hardware mondo delivery easier
- May have queues defined for each virtual CPU
  - dev_mondo queue must be sized for all possible interrupt sources
  - dev_mondo queue may never contain more than one entry for same source
- Hypervisor API to send 64Byte mondo to CPU queue
  - Used for CPU to CPU x-calls
  - Queue may fill and sender's API call fails

Logical Domaining Technology

- Virtualization and partitioning of resources
  - > Each domain is a full virtual machine, with a dynamically re-configurable sub-set of machine resources, and its own independing OS instance
  - > Protection & isolation via SPARC hardware and Ldoms firmware



Fundamentals

- Each virtual machine should appear as an entirely independent machine
  - own kernel, patches, tuning parameters
  - own user accounts, administrators
  - own disks
  - own network interfaces, MAC & IP addresses
  - Start, stop and reboot independently of each other

Hypervisor Support

- Hypervisor software is responsible for maintaining separation between domains
  - Using extensions built into a sun4v CPU
- Also provides Logical Domain Channels (LDCs) so that domains can communicate with each other
  - Mechanism by which domains can be virtually networked with each other, or provide services to each other
- Domain Roles: Control, I/O and Service

LDom Manager

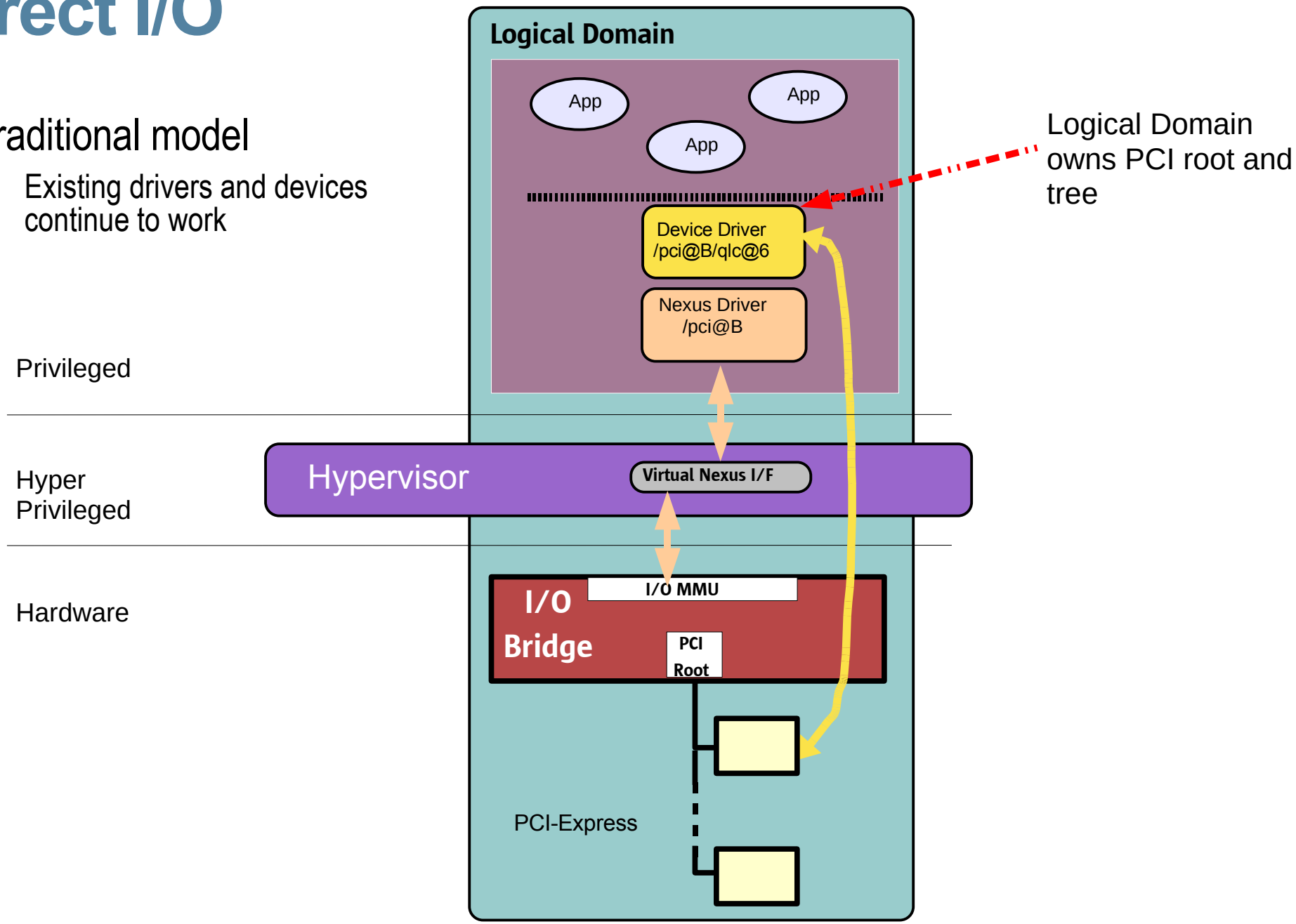
- One Manager per machine
  - Can be run in any domain, but only 1 domain at a time
    - The “Control Domain”
  - Controls Hypervisor and all its LDoms
- Exposes CLI to administrator
- Maps Logical Domains to physical resources
  - Constraint engine
  - Heuristic binding of LDoms to resources
    - Assists with performance optimization
    - Assists in even of failures / blacklisting

Dynamic Reconfiguration

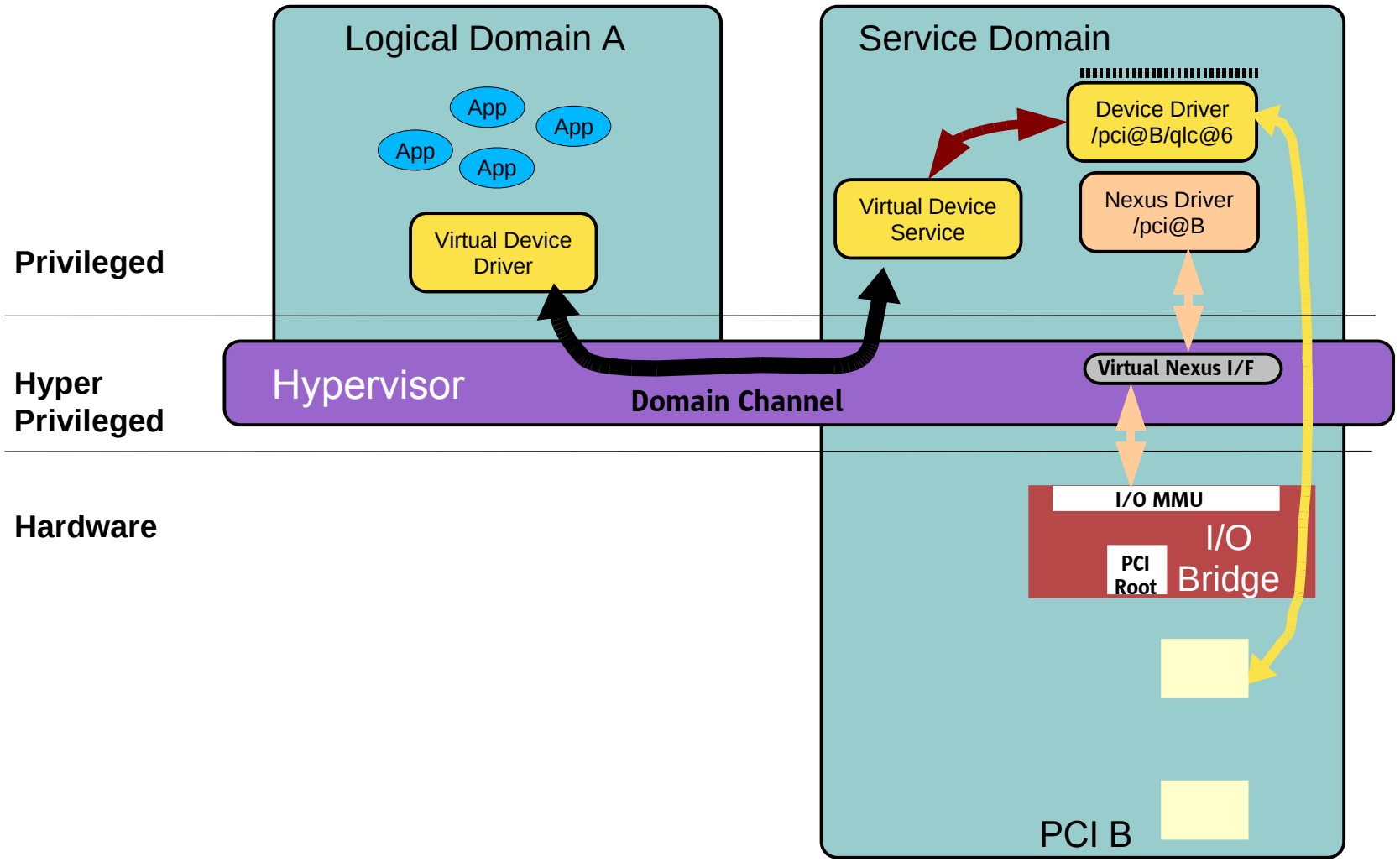
- Hypervisor has ability to dynamically shrink or grow LDoms upon demand
- Simply add/remove cpus, memory & I/O
  - Ability to cope with this without rebooting depends on guest OS capabilities
  - Guest OS indicates its capabilities to the domain manager
- Opportunity to improve utilization by balancing resources between domains

Direct I/O

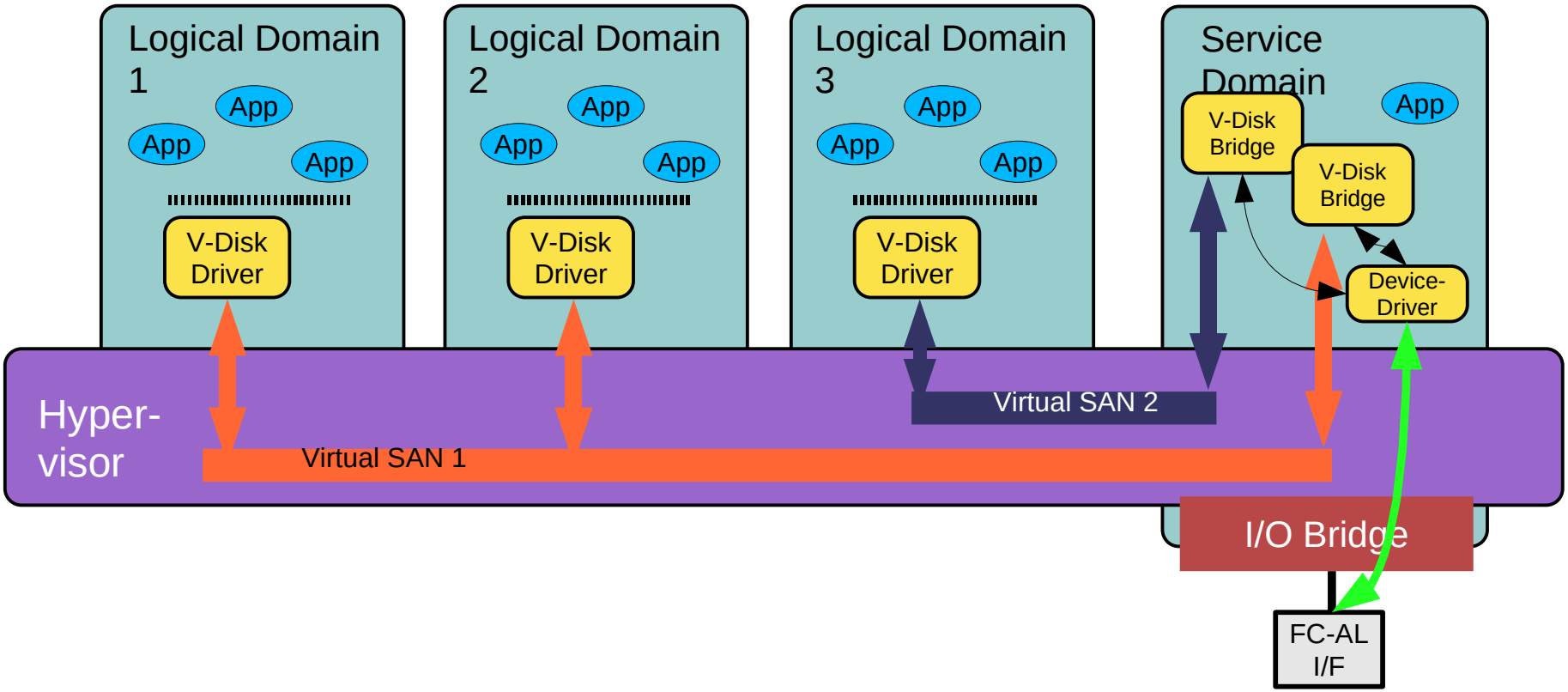
- Traditional model
  - > Existing drivers and devices continue to work



Virtualized I/O

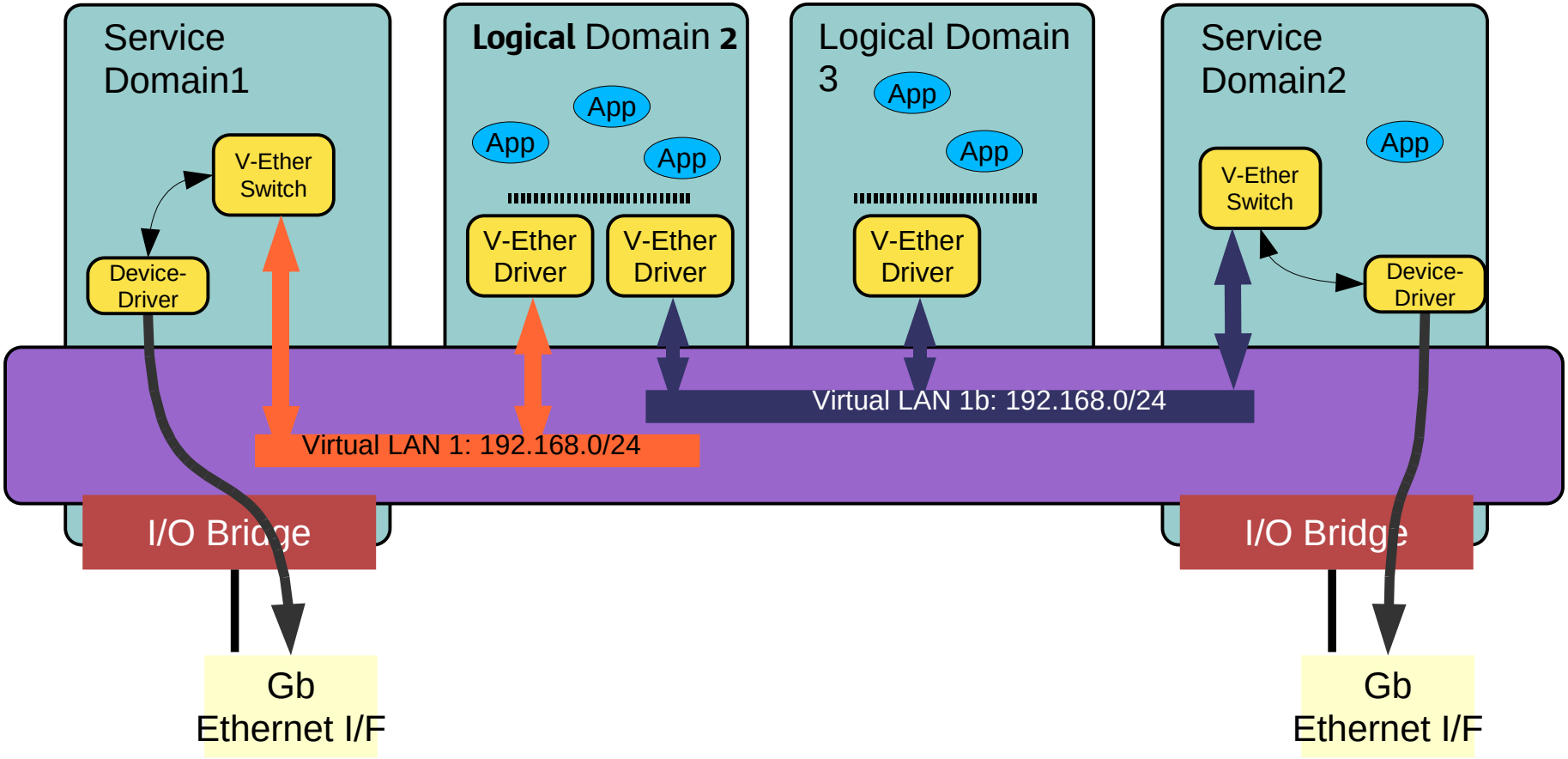


Virtual (Block) Disk device & server



Redundancy; Multi-path virtual I/O

- Virtualised devices can be used for redundant fail-over if guest OS supports it



Additional Topics

- Error handling
- Machine Descriptions
- Boot process

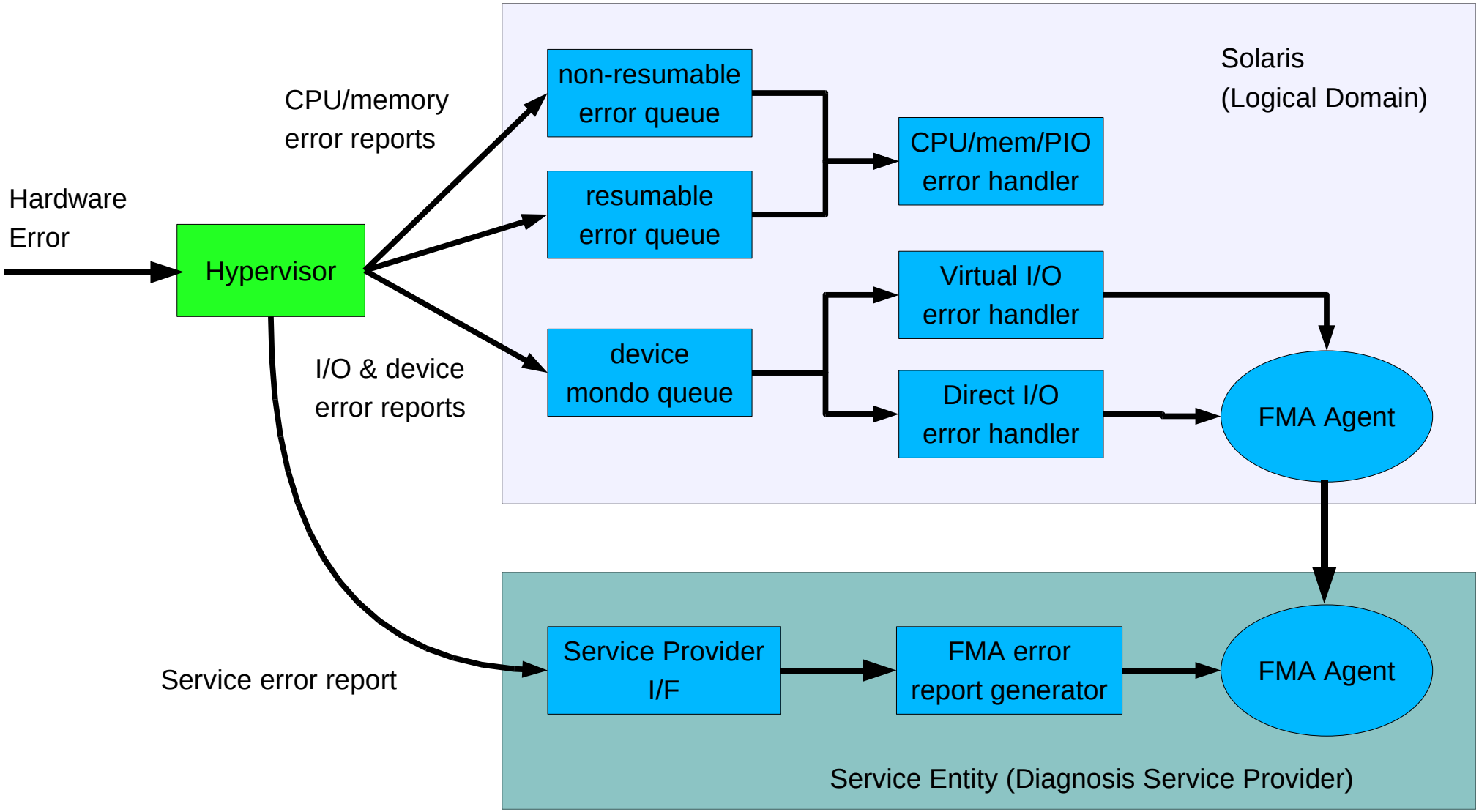
Error delivery philosophy

- Hypervisor handles and abstracts underlying hardware errors
- All errors logged on service processor
- Guest OSs are told about impact of error
  - No point in informing guest about correctable errors
- Legacy OS should be able to run on new platform

Error handling

- Two simple classificiations; “after handling the error, can I ...”
  - 1. Resume execution of what I was doing, or
  - 2. Can't resume execution ... some policy to handle this
- Simplest error handlers:
  - 1. Retry
  - 2. Panic
- 2 in-memory queues associated with these types, similar to interrupts
- Queue entries contain error reports distilled by hypervisor
- Hypervisor creates reports and attempts to correct errors when possible

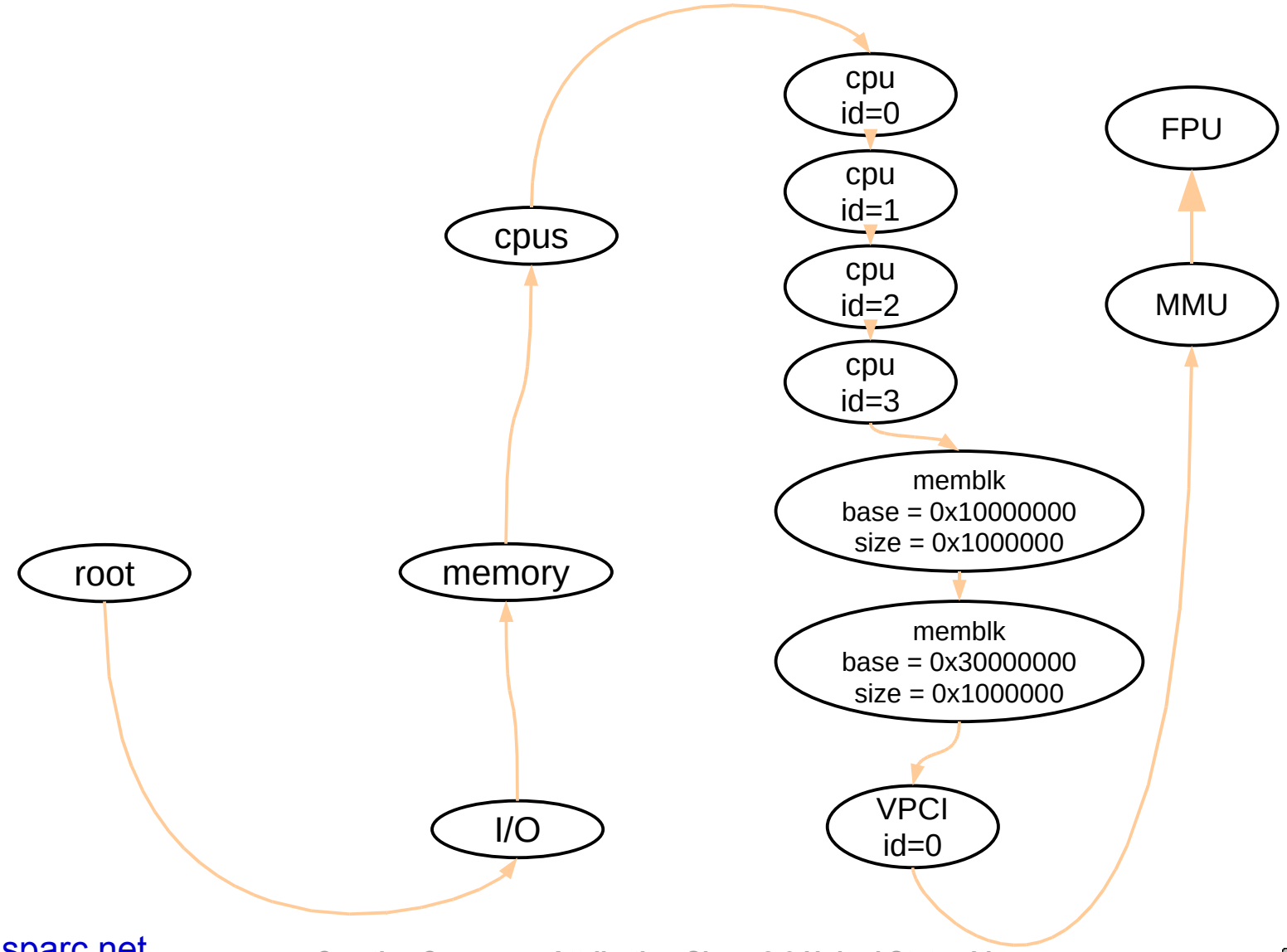
Error handling agents



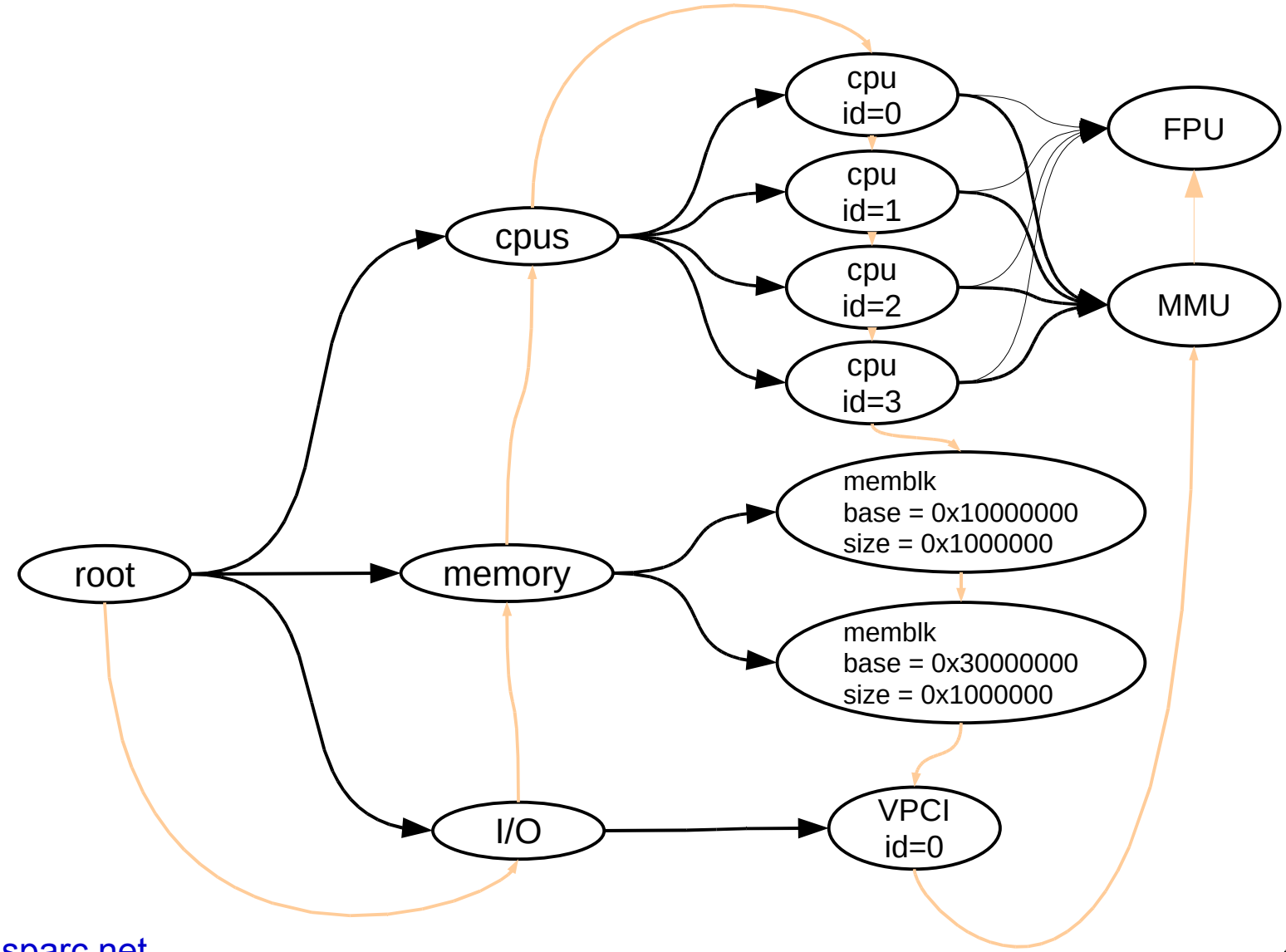
Machine description

- How the OS Inside a virtual machine finds its resources
- Trivial list of nodes that detail the contents of each domain
  - CPUs, Blocks of memory, I/O devices, I/O Ports etc.
- Nodes are also inter-linked to form a DAG to convey more advanced information for guest OSs that care
  - e.g. cache sharing, NUMA memory latencies etc.
- Key requirements;
  - Very simple to parse by the simplest of guests
  - Convey very complex information for guest OSs that care
  - A guest need not understand all the information presented
    - e.g. old OS running on a new platform

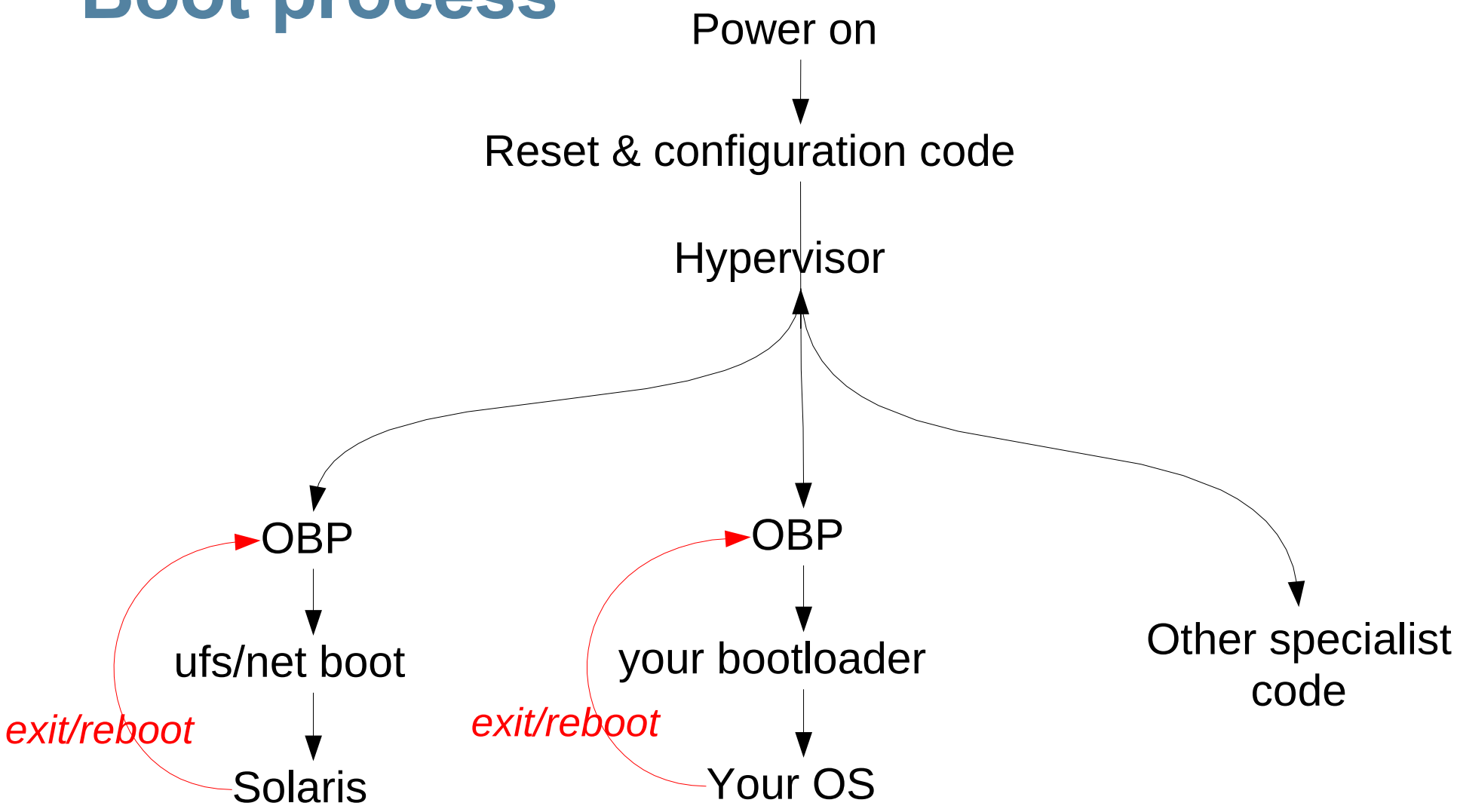
Simple list of nodes



Also arranged as a DAG



Boot process



Summary

- Specifications & code published:
  - http://www.opensparc.net
  - http://www.opensolaris.net
- “Legion” instruction level simulator available to assist with code development
  - Provides level of code execution visibility not possible on actual hardware
  - Source code available on http://www.opensparc.net
- Contact alias:
  - hypervisor@sun.com

Thanks for watching the OpenSPARC Slide-Cast!

Let us hear from you! The OpenSPARC Team would appreciate your feedback on this in the http://www.OpenSPARC.net forum.