

Design & Implementation of a High-performance I/O Path for Data Protection

M. Marazakis, M. Fountoulakis, M. Flouris, and A. Bilas

maraz@ics.forth.gr

CARV Laboratory

Institute of Computer Science

Foundation for Research and Technology – Hellas



Motivation

- ▶ Ever increasing data volumes
- ▶ Ever demanding user expectations for diverse applications
 - ▶ Capacity
 - ▶ Performance
 - ▶ Data Integrity
- ▶ Direct-attached storage will not go away any time soon
- ▶ Is RAID enough?
 - ▶ ... for protecting data integrity
 - ▶ ... for recovery from user errors

Data Protection Techniques

- ▶ **RAID**
 - ▶ Protection from device failures
- ▶ **Error detection/correction codes**
 - ▶ Protection from (silent) data corruption
- ▶ **Point-in-Time Snapshots**
 - ▶ Protection from user errors (timeline of volume states)
- ▶ **Live Volume Migration**
 - ▶ Maintenance without service disruption

Data Integrity

- ▶ **Error-Detecting vs. Error-Correcting Codes**
 - ▶ Per 4KB data block
 - ▶ CRC32 -> 4 bytes, MD5 -> 16 bytes, SHA1 -> 20 bytes
 - ▶ Reed/Solomon (N,K) for up to $(N-K)/2$ byte errors
 - ▶ Potential for hardware-assisted acceleration
 - ▶ Computed & stored during i/o writes
 - ▶ Retrieved & checked during i/o reads
- ▶ **Enhancements to RAID layer**
 - ▶ Capability to restore data integrity & re-alloc/re-map damaged data blocks
 - ▶ Current prototype: RAID 1+0 (stripes of mirrors)

Context

- ▶ Data protection feature-sets are currently provided in enterprise-grade storage systems
 - ▶ Combined with component/path redundancy
 - ▶ Software-only (host-based) implementations are also available
- ▶ Can this functionality be achieved (efficiently) with commodity-grade components?
 - ▶ Within the I/O controller, rather than at the host?
 - ▶ Using an OS-based run-time, rather than embedded firmware?
- ▶ Starting with a programmable I/O controller, we enhance it with data protection features
 - ▶ Prototype (H/W + S/W) & evaluate with realistic workloads

Challenges

- ▶ **Challenges related to performance**
 - ▶ System performance is often much less than the nominal performance of each of the components
 - ▶ Systems software overheads
 - ▶ Increased complexity
 - ▶ H/W + S/W orchestration (explicit scheduling)
 - ▶ Host + I/O controller
 - ▶ Co-ordination?
 - ▶ “Asymmetry” of resources?
- ▶ **Data protection relies on extensive metadata**
 - ▶ ... that needs to be managed & protected

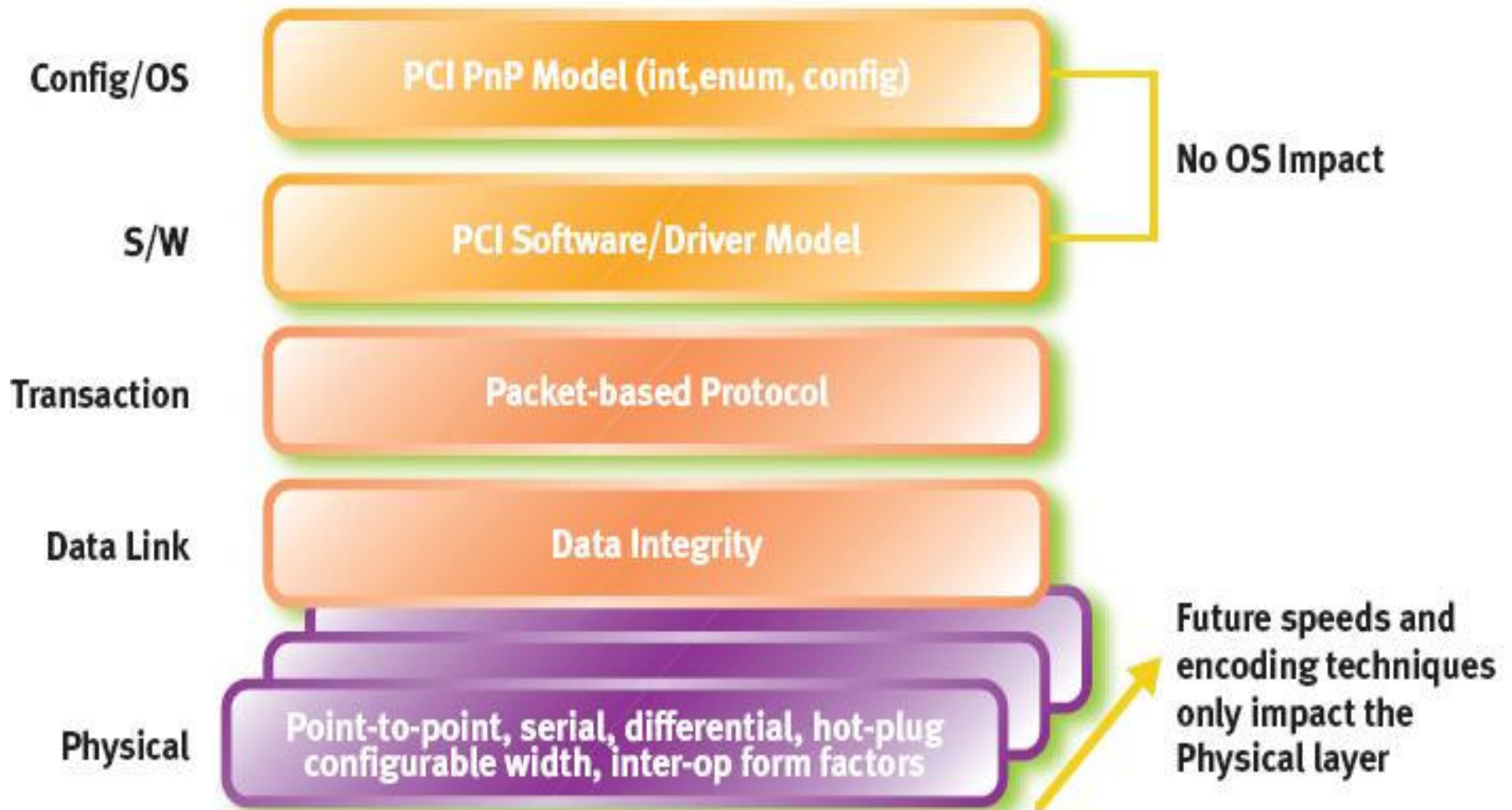
Goal of this talk

- ▶ Describe I/O path via a walkthrough of real prototype
- ▶ Outline challenges in providing new functions *and* achieving high-performance
- ▶ Ponder about what is important for I/O subsystems

Outline

- ▶ **Background**
 - ▶ **Peripheral Communication Protocols**
 - ▶ **PCIe, SCSI, RAID**
- ▶ DARC controller
- ▶ Extensible Block-level Storage
 - ▶ Very briefly discuss metadata issue
- ▶ Evaluation
- ▶ Summary & Conclusions

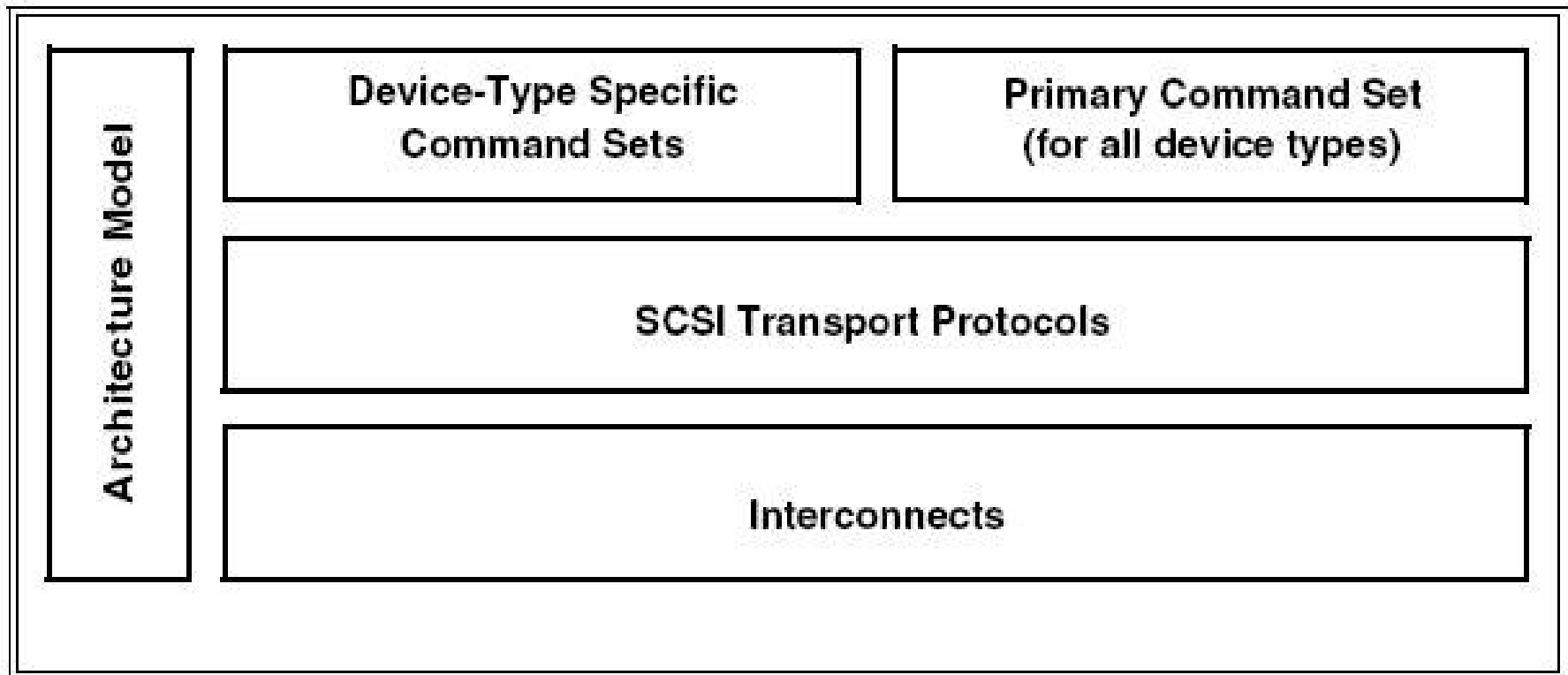
PCI Express: Layers



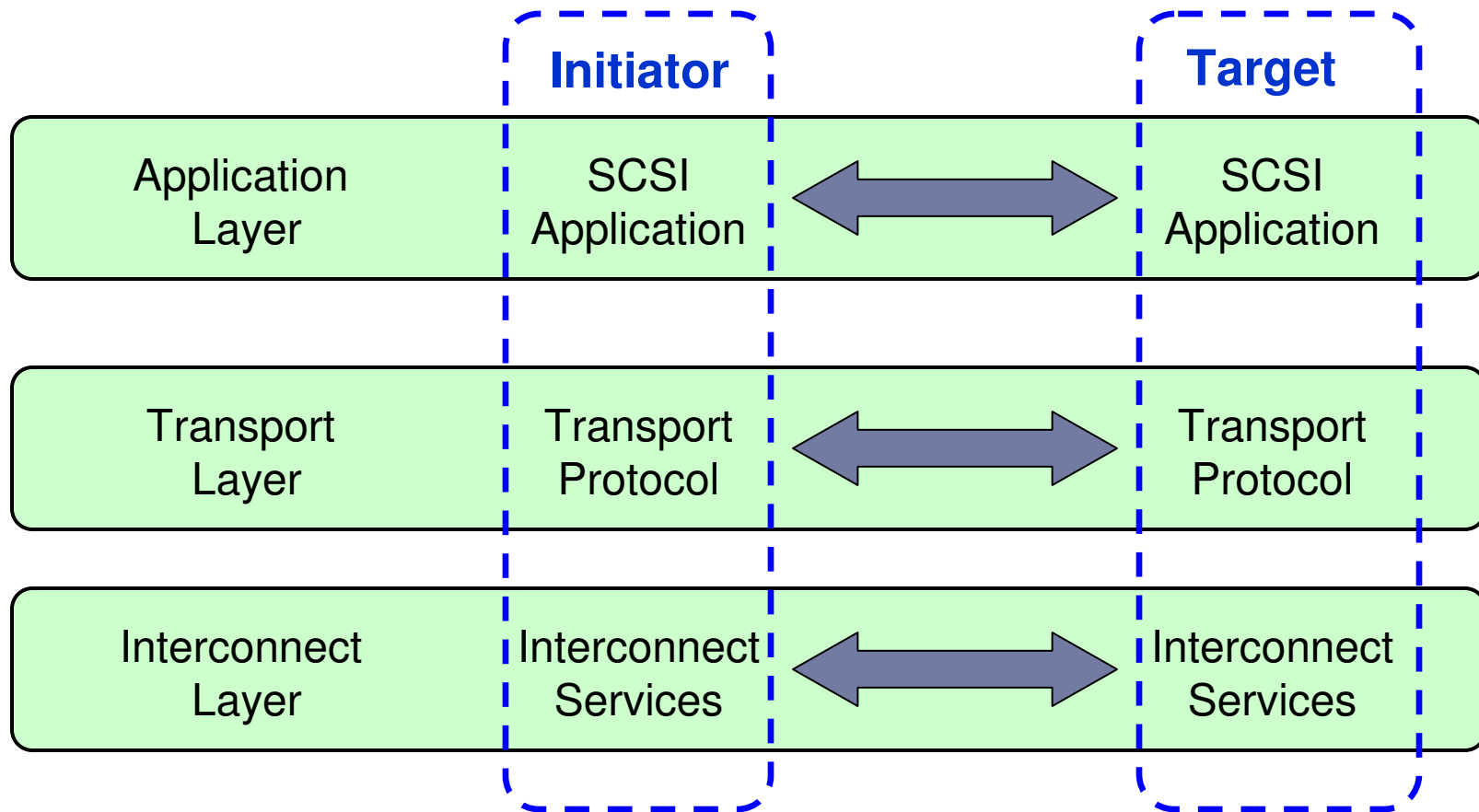
Disk Interface Technologies

	Parallel ATA	Parallel SCSI	Fibre Channel	SATA	Serial Attached SCSI ¹
Performance					
Technology Introduction ²	2000	2002	2001	2002	2004
Maximum Speed ³	100 MB/s	320 MB/s	4.2 Gb/s (400 MB/s)	3.0 Gb/s (300 MB/s)	3.0 Gb/s (300 MB/s)
Topology	Shared bus master/slave	Shared bus	Arbitrated loop/ switched fabric	Point-to-point	Point-to-point
Number of Devices	2	15	1,000s	up to 15	100s

SCSI Standards



SCSI Communication Model



Initiator: issues commands to Target, to perform a Task
(example: HBA)

Target: executes commands to perform Task received from an Initiator
(example: peripheral device)

SCSI: CDBs, Tasks, Ports

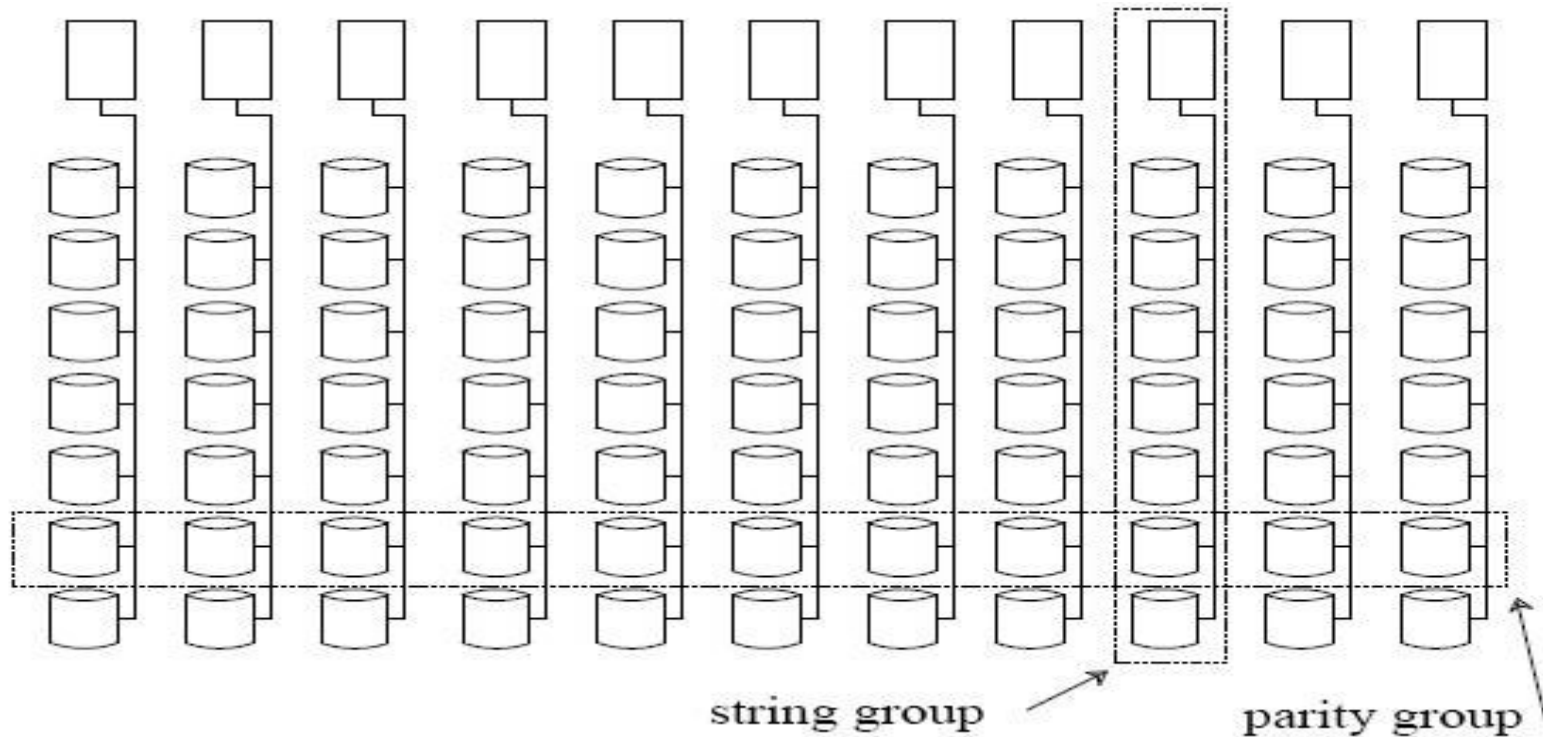
- ▶ Each device service request contains a **Command Descriptor Block (CDB)**:
 - ▶ Command to be executed + command-specific inputs
 - ▶ # bytes: 6, 10, 12, 16 (or variable)
- ▶ **Task**: object within LUN representing the work associated with a command or a series of linked commands
- ▶ **Port**: physical connector for communication
 - ▶ Initiator, Target, Combined Initiator/Target, Target with multiple ports

SCSI RBC: Reduced Block Command Set

Command name	OpCode	Command Support		Reference
		Fixed	Removable	
FORMAT UNIT	04h	O	O	RBC
INQUIRY	12h	M	M	SPC-2 ¹
MODE SELECT(6)	15h	M	M	SPC-2 ¹
MODE SENSE(6)	1Ah	M	M	SPC-2 ¹
PERSISTENT RESERVE IN	5Eh	O	O	SPC-2 ¹
PERSISTENT RESERVE OUT	5Fh	O	O	SPC-2 ¹
PREVENT/ALLOW MEDIUM REMOVAL	1Eh	N/A	M	SPC-2 ¹
READ (10)	28h	M	M	RBC
READ CAPACITY	25h	M	M	RBC
RELEASE(6)	17h	O	O	SPC-2 ¹
REQUEST SENSE	03h	O	O	SPC-2 ¹
RESERVE(6)	16h	O	O	SPC-2 ¹
START STOP UNIT	1Bh	M	M	RBC
SYNCHRONIZE CACHE	35h	O	O	RBC
TEST UNIT READY	00h	M	M	SPC-2 ¹
VERIFY (10)	2Fh	M	M	RBC
WRITE (10)	2Ah	M	M	RBC
WRITE BUFFER	3Bh	M	O	SPC-2 ¹

RAID: Redundant Array of Independent (Inexpensive) Disks

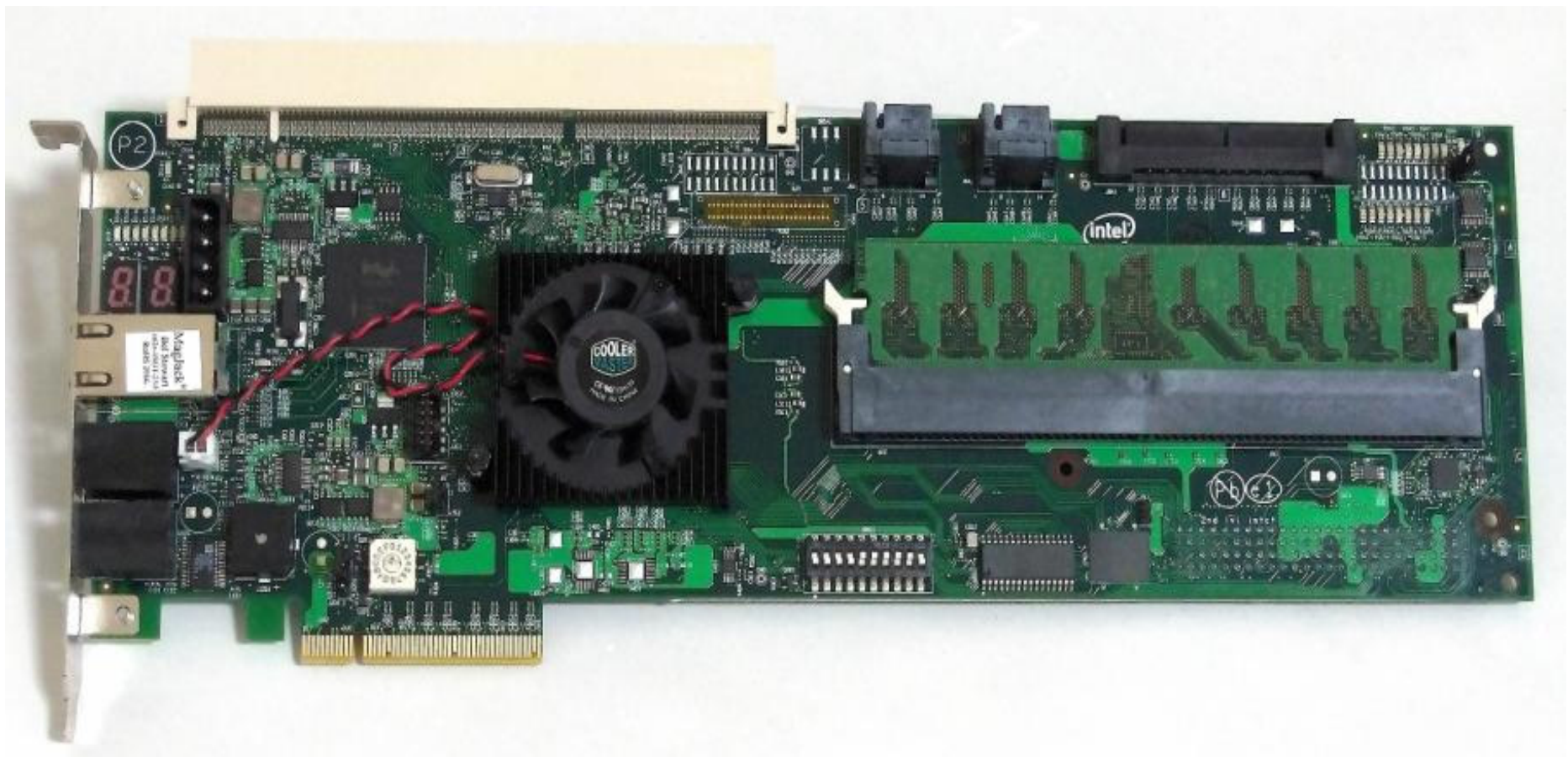
- ▶ Striping for parallel data transfer & load balancing
- ▶ Redundancy for failure protection
- ▶ Error-correcting codes



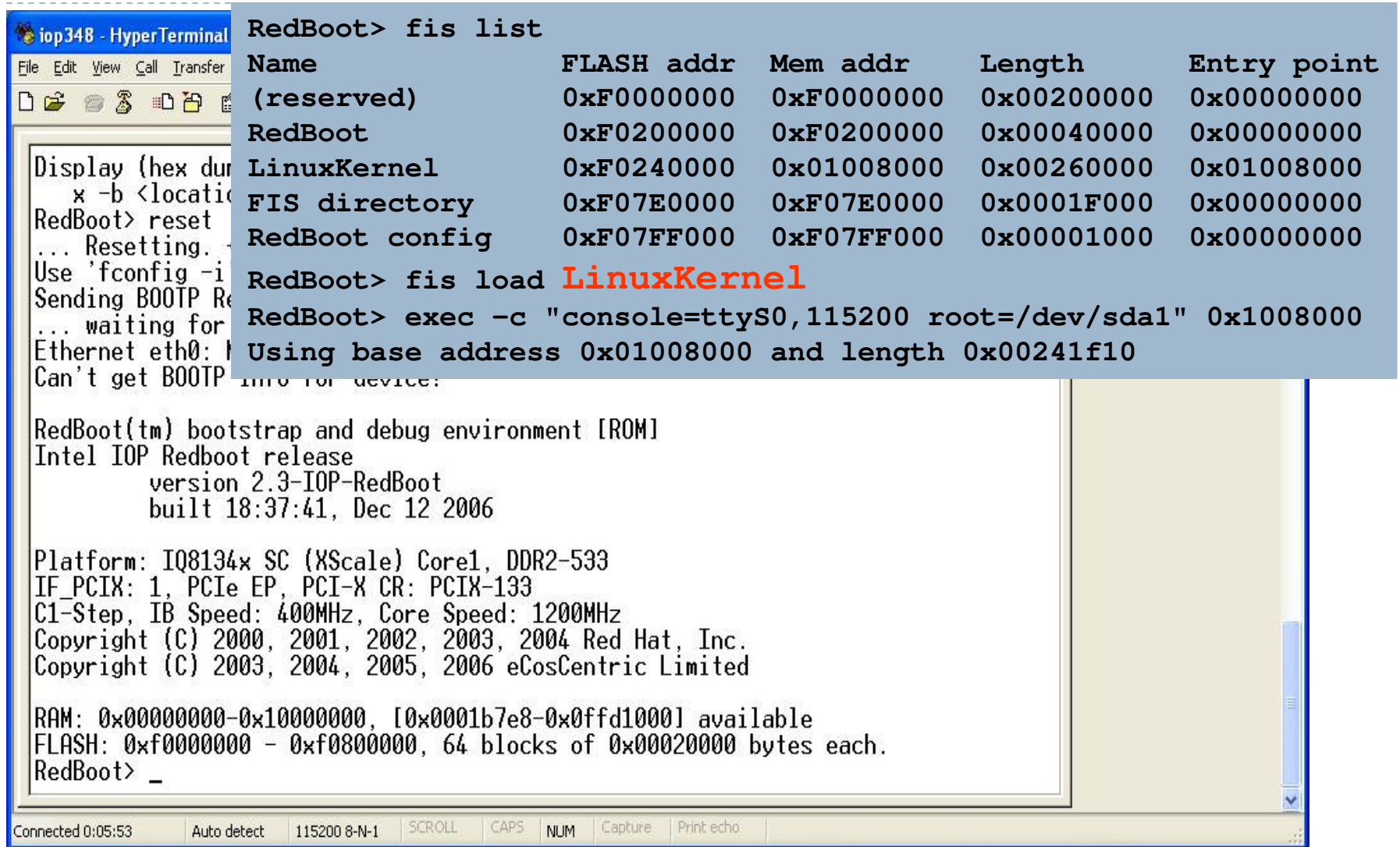
Outline

- ▶ Background
 - ▶ Peripheral Communication Protocols
 - ▶ PCIe, SCSI, RAID
- ▶ **DARC controller**
- ▶ Violin: A Framework for Extensible Block-level Storage
- ▶ Evaluation
- ▶ Summary & Conclusions

IOP348-based I/O Controller



IOP348 – Boot Loader



The screenshot shows a HyperTerminal window titled "iop348 - HyperTerminal" with a menu bar (File, Edit, View, Call, Transfer) and a toolbar. The terminal output is as follows:

```
RedBoot> fis list
Name                FLASH addr  Mem addr    Length     Entry point
(reserved)          0xF0000000 0xF0000000 0x00200000 0x00000000
RedBoot              0xF0200000 0xF0200000 0x00040000 0x00000000
LinuxKernel         0xF0240000 0x01008000 0x00260000 0x01008000
FIS directory       0xF07E0000 0xF07E0000 0x0001F000 0x00000000
RedBoot config      0xF07FF000 0xF07FF000 0x00001000 0x00000000
RedBoot> fis load LinuxKernel
RedBoot> exec -c "console=ttyS0,115200 root=/dev/sda1" 0x1008000
Using base address 0x01008000 and length 0x00241f10
Info for device:

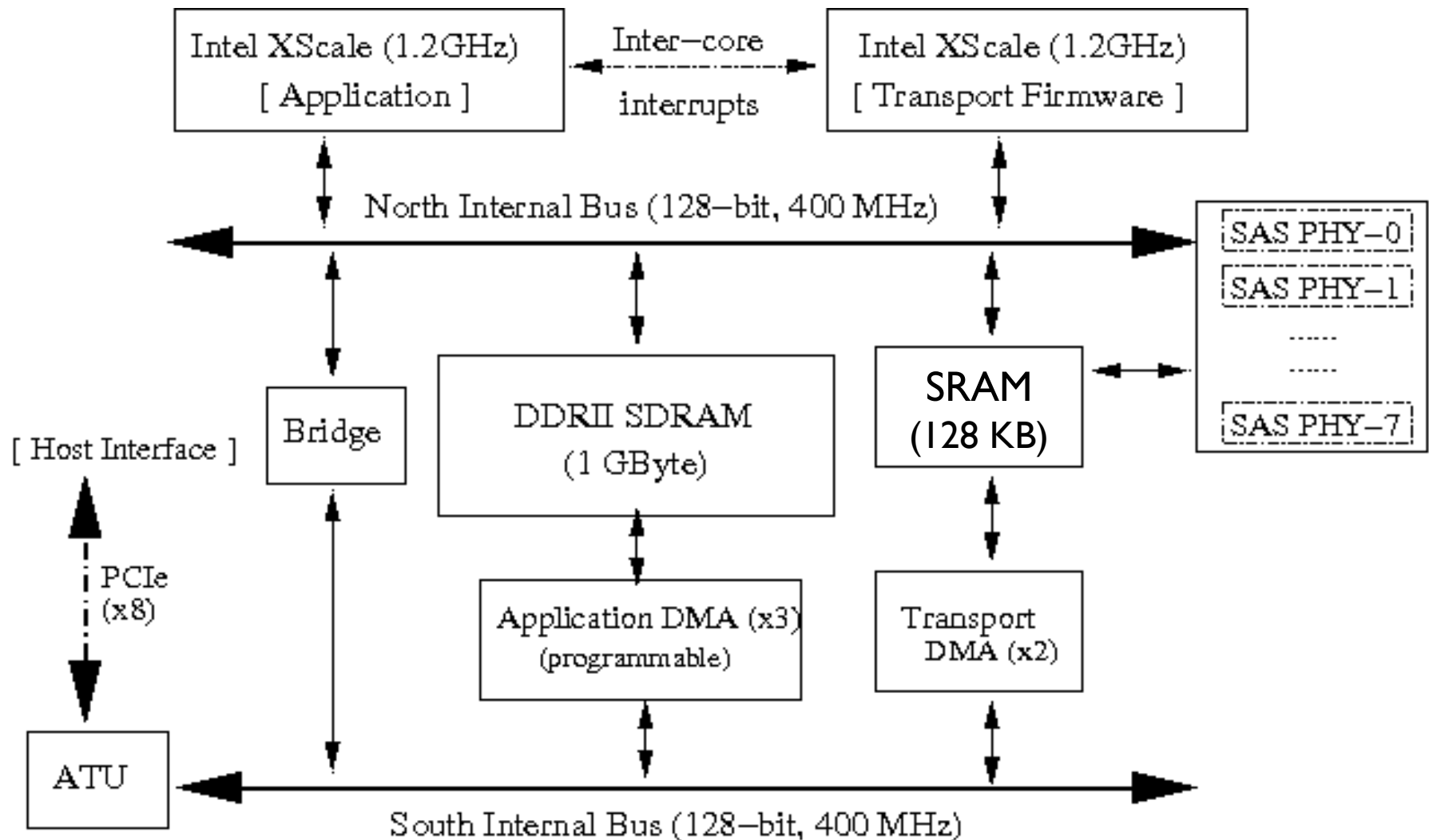
RedBoot(tm) bootstrap and debug environment [ROM]
Intel IOP Redboot release
    version 2.3-IOP-RedBoot
    built 18:37:41, Dec 12 2006

Platform: IQ8134x SC (XScale) Core1, DDR2-533
IF_PCI-X: 1, PCIE EP, PCI-X CR: PCI-X-133
C1-Step, IB Speed: 400MHz, Core Speed: 1200MHz
Copyright (C) 2000, 2001, 2002, 2003, 2004 Red Hat, Inc.
Copyright (C) 2003, 2004, 2005, 2006 eCosCentric Limited

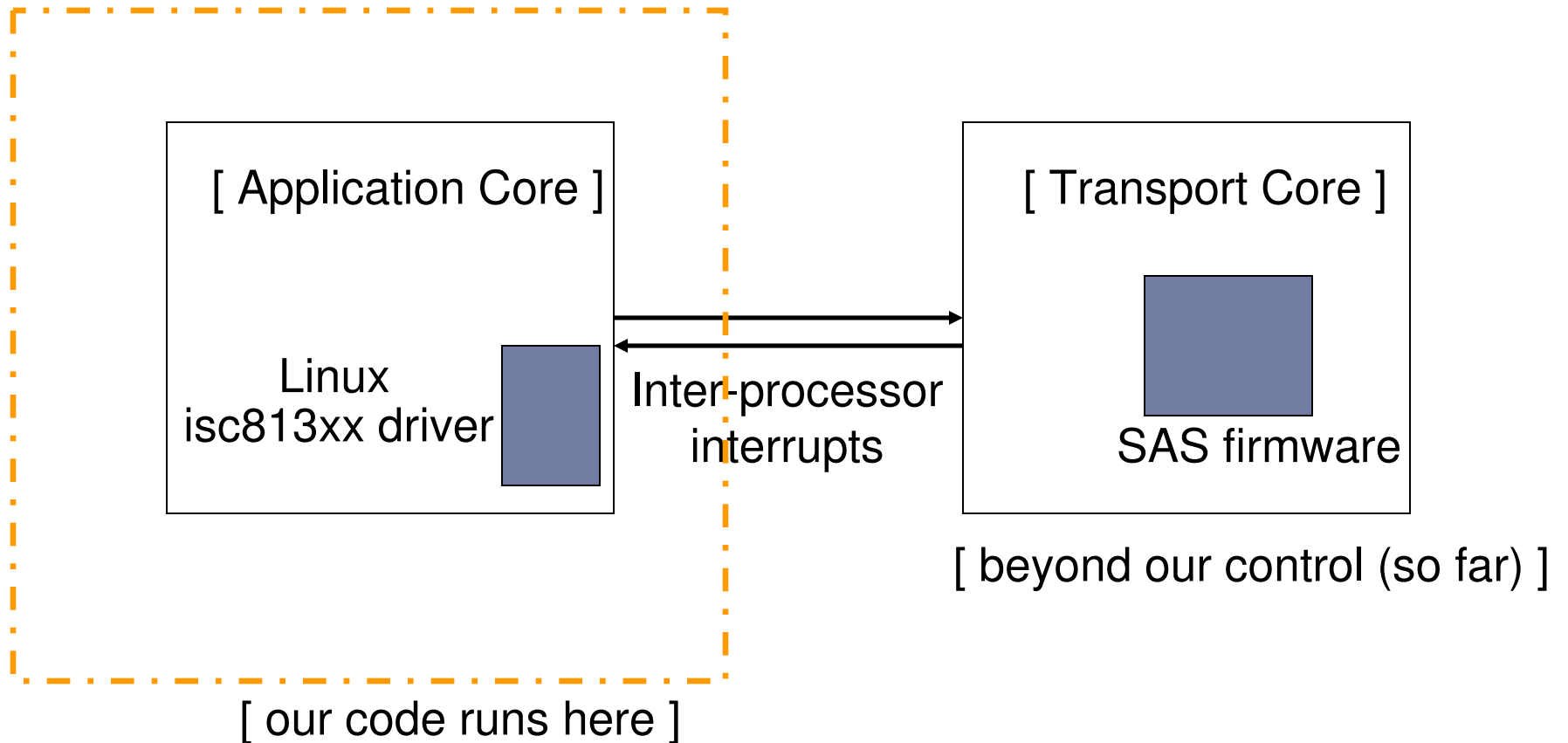
RAM: 0x00000000-0x10000000, [0x0001b7e8-0x0ffd1000] available
FLASH: 0xf0000000 - 0xf0800000, 64 blocks of 0x00020000 bytes each.
RedBoot> _
```

At the bottom of the window, the status bar shows: "Connected 0:05:53", "Auto detect", "115200 8-N-1", "SCROLL", "CAPS", "NUM", "Capture", "Print echo".

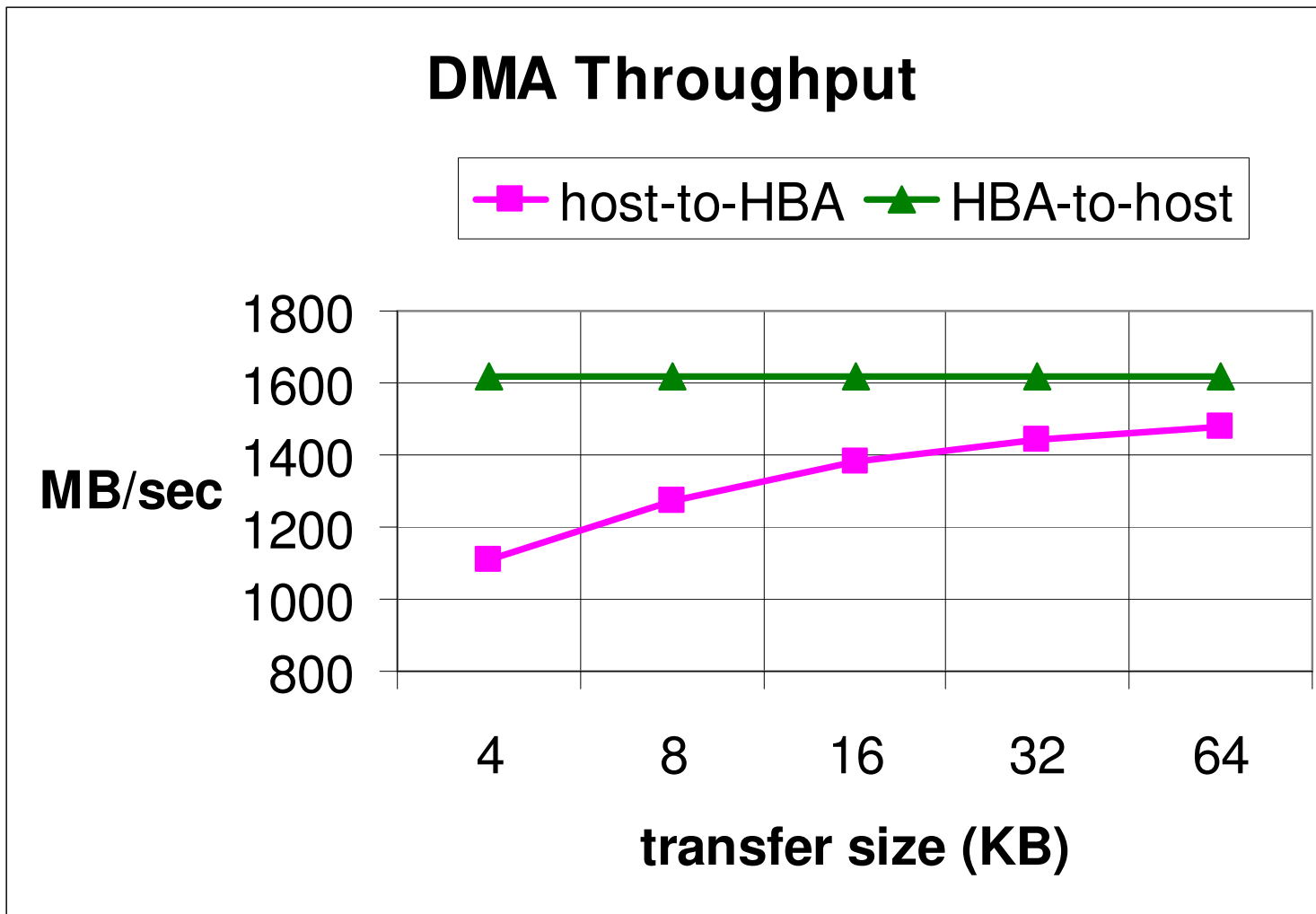
I/O Controller Data Path



2 Cores: Application, Transport

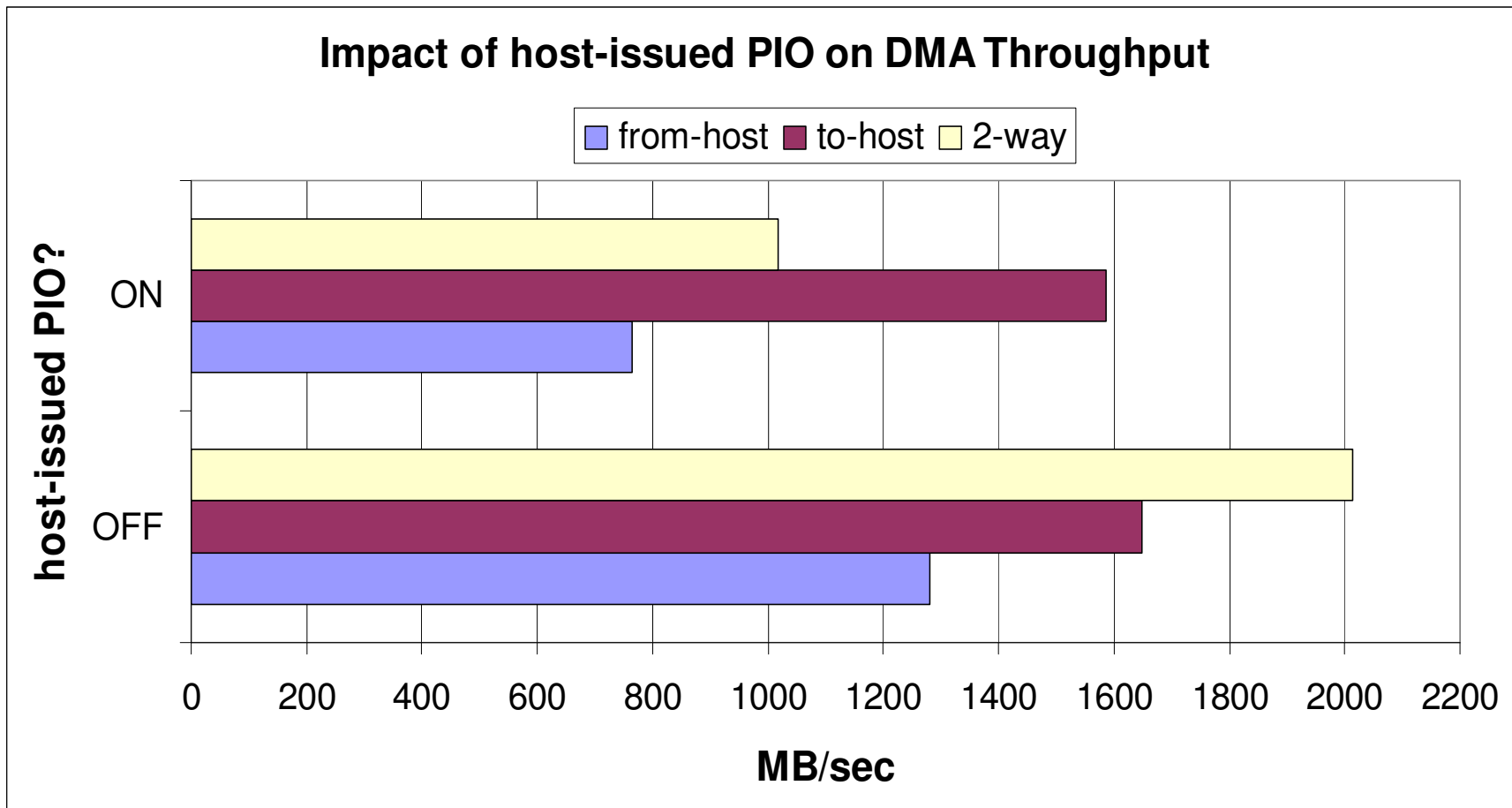


IOP348 Micro-benchmarks (I)



IOP348 Micro-benchmarks (II)

8KB DMA transfers



IOP348 Micro-benchmarks (III)

- ▶ **Interrupt delay: 837 nsec (similar for CTX SW)**
 - ▶ With host-initiated PIO: 880 nsec
- ▶ **IOP348**
 - ▶ One cycle ~ 0.833 nsec (rate: 1.2 GHZ)
 - ▶ Local-bus store: 30 nsec
 - ▶ With host-initiated PIO: 37 nsec
 - ▶ Memory-store: 99 nsec
 - ▶ Outbound store: 114 nsec
 - ▶ With Host-initiated PIO + IOP348-initiated DMA: 198 nsec

IOP348 Micro-benchmarks (IV)

- ▶ IOP348 Outbound load: 674 ns

- ▶ With Host-initiated PIO: 679 ns



- ▶ With IOP348-initiated DMA: **3390 ns**

- ▶ With Host-initiated PIO + IOP348-initiated DMA: **5970 ns**

- ▶ Is PCI-Express “slow”?

- ▶ D. Miller, et al: “Motivating Future Interconnects: A Differential Measurement Analysis of PCI Latency“, In: Proc. ANCS 2009

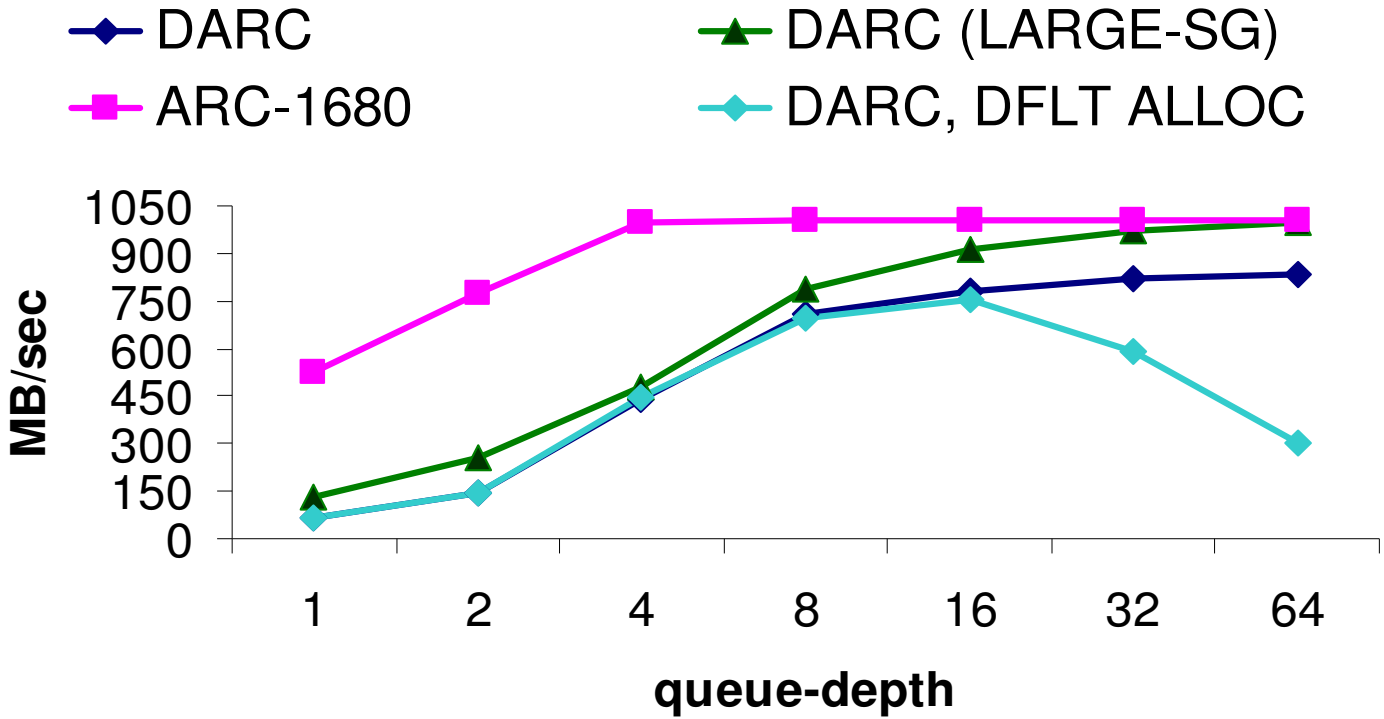
- ▶ “... we illustrate how an evolution in the common PCI interconnect architecture has worsened latency by a factor of between 3 and 25 over earlier incarnations.”

Memory Allocation Effects

RAID-0, IOmeter RS pattern

[8 SAS HDDs]

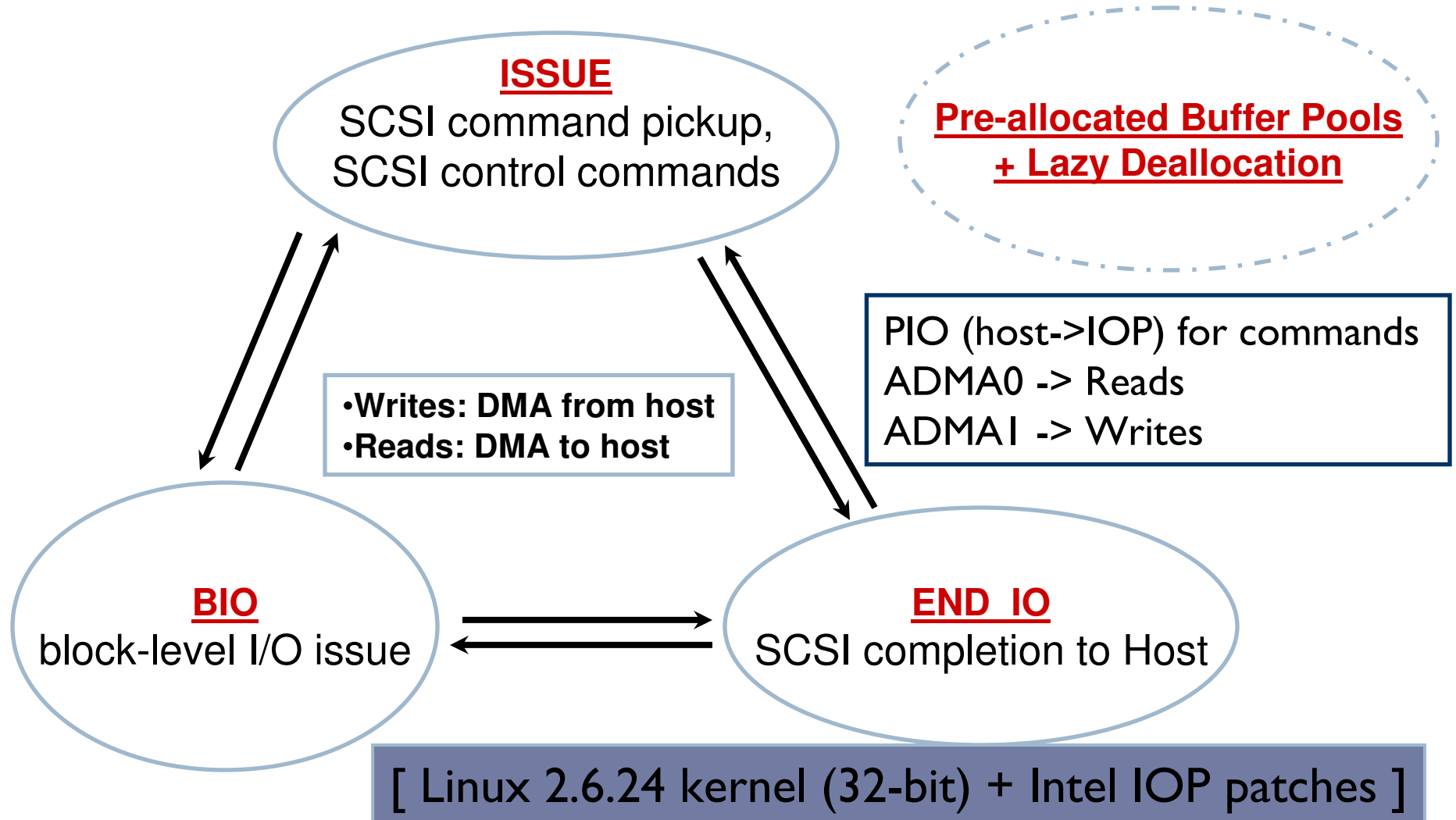
RS IOmeter Pattern



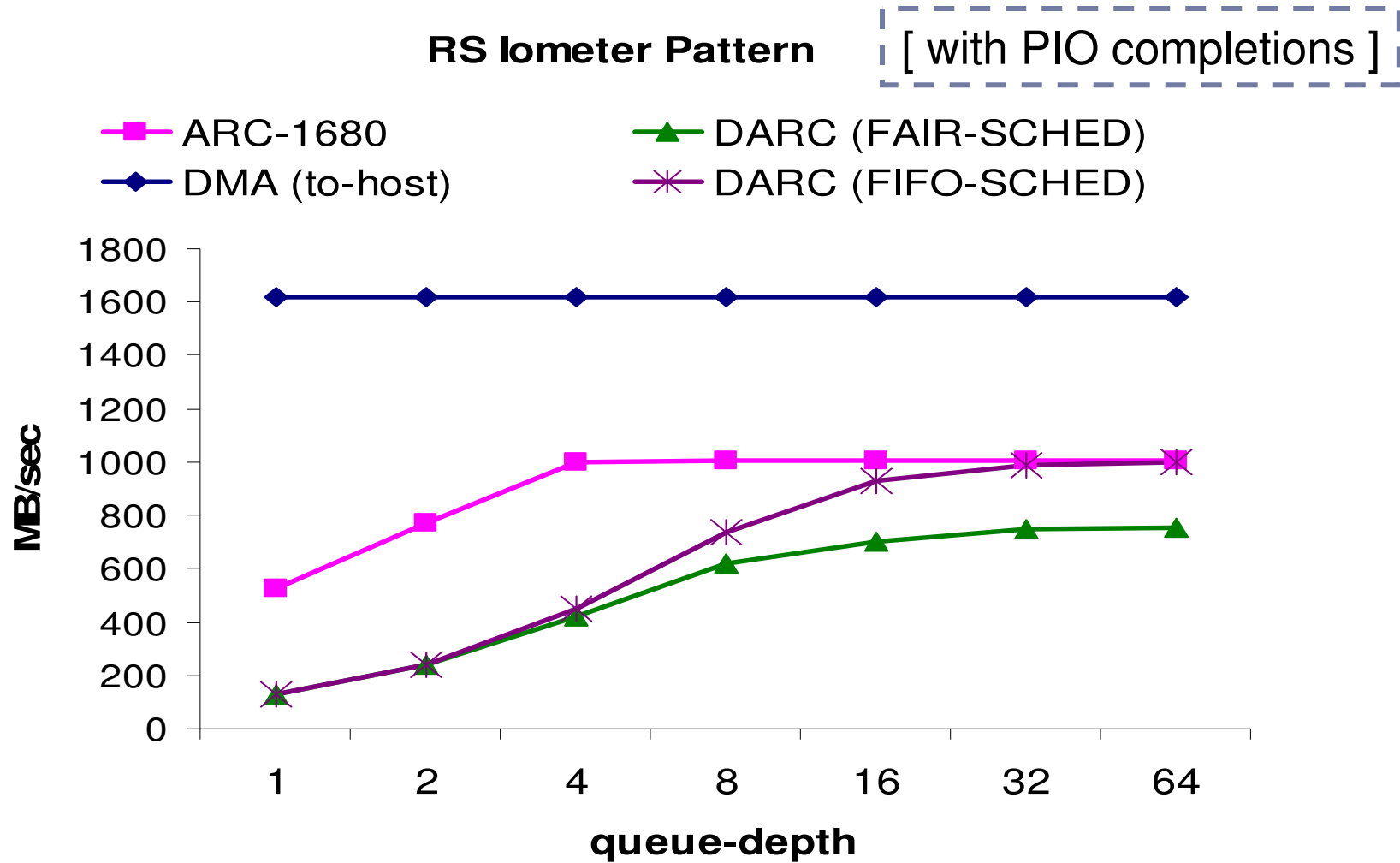
Scheduling

- ▶ **Threads (work-queues) instead of FSMs**
 - ▶ Simpler to develop/re-factor code & debug
 - ▶ Can block independently from one another
- ▶ **Fair scheduling (SCHED_OTHER) is not optimal**
 - ▶ Threads need to be explicitly pre-empted when polling on a resource
- ▶ **SCHED_FIFO scheduler**
 - ▶ Static priorities, no time-slicing
 - ▶ With explicit "yields" when polling or when "enough" work has been done - always yield when a resource is unavailable

Co-operating Contexts (simplified)



Scheduler Effects



Xscale + Linux: Memory Regions

- ▶ **Cacheable + write-combining**
 - ▶ Explicit cache invalidations & writebacks to sync w/ peripherals
 - ▶ At the cache-line granularity (32-bytes)
- ▶ **Non-cacheable, no write-combining**
 - ▶ Every load/store goes through the CPU and cache hierarchy
 - ▶ ... which can be slow
- ▶ **Non-cacheable + write-combining**
 - ▶ Stores go through the CPU write buffer → fast
 - ▶ Does not work if R/W side-effects are expected
 - ▶ e.g. memory-mapped device registers

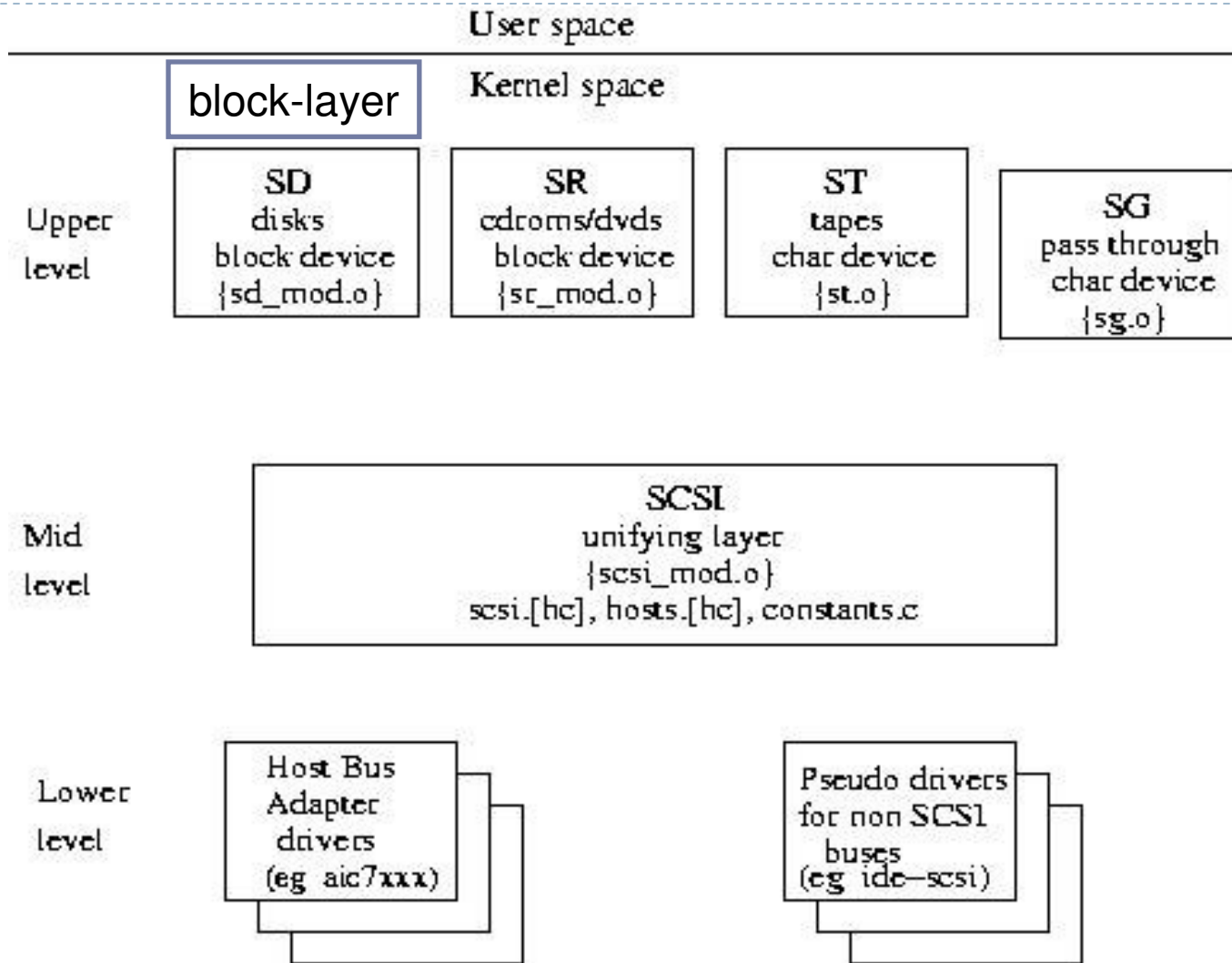
Memory Regions for DARC (1)

- ▶ **Cacheable + write-combining for**
 - ▶ every piece of information that does NOT need to be communicated to external hardware
- ▶ **Non-cacheable, no write-combining for**
 - ▶ all peripheral device registers
 - ▶ Xscale outbound PIO to host memory

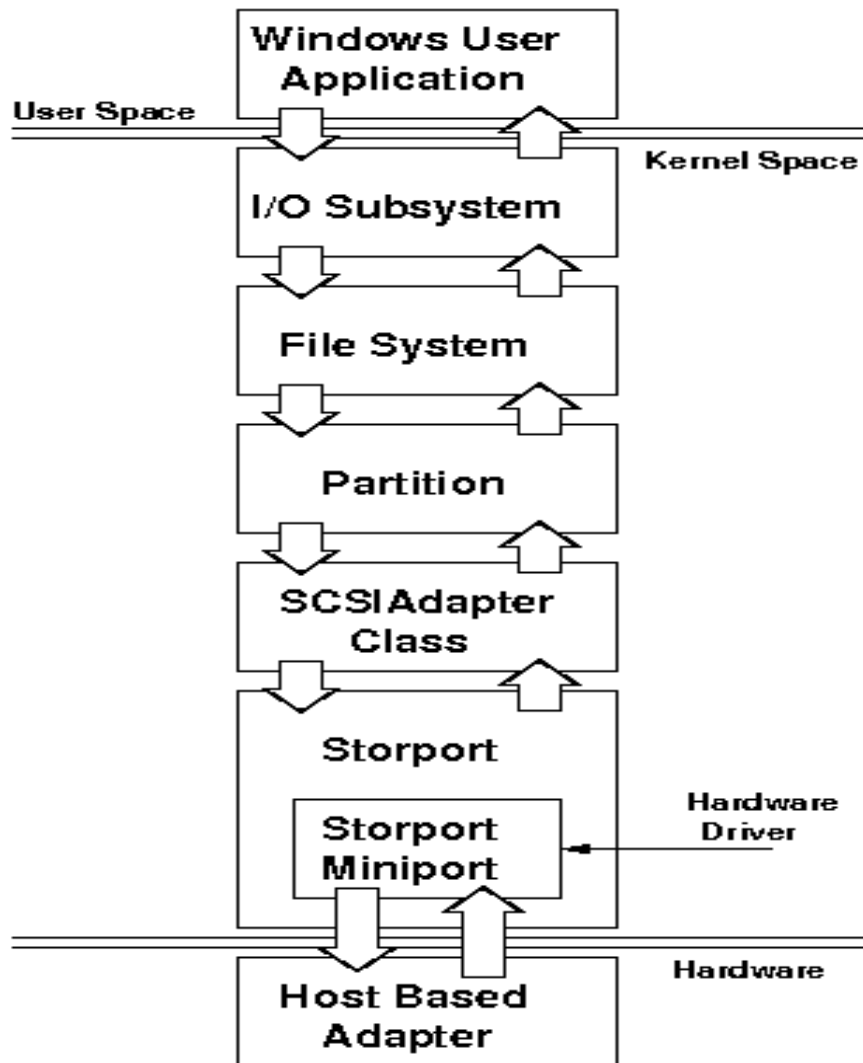
Memory Regions for DARC (2)

- ▶ **Non-cacheable + write-combining for**
 - ▶ DMA descriptors
 - ▶ Only fast stores are required
 - ▶ SCSI driver buffer allocations
 - ▶ Initially implemented (within isc813xx driver) as Non-cacheable, no write-combining
- ▶ **Cacheable + write-combining**
 - ▶ CRCs: allocated along with other data to be processed, and their cache entries are managed explicitly.
 - ▶ The Command FIFO buffer
 - ▶ Dequeuing demands both efficient loads and stores → explicit cache management is the fastest alternative.

Host (Linux)



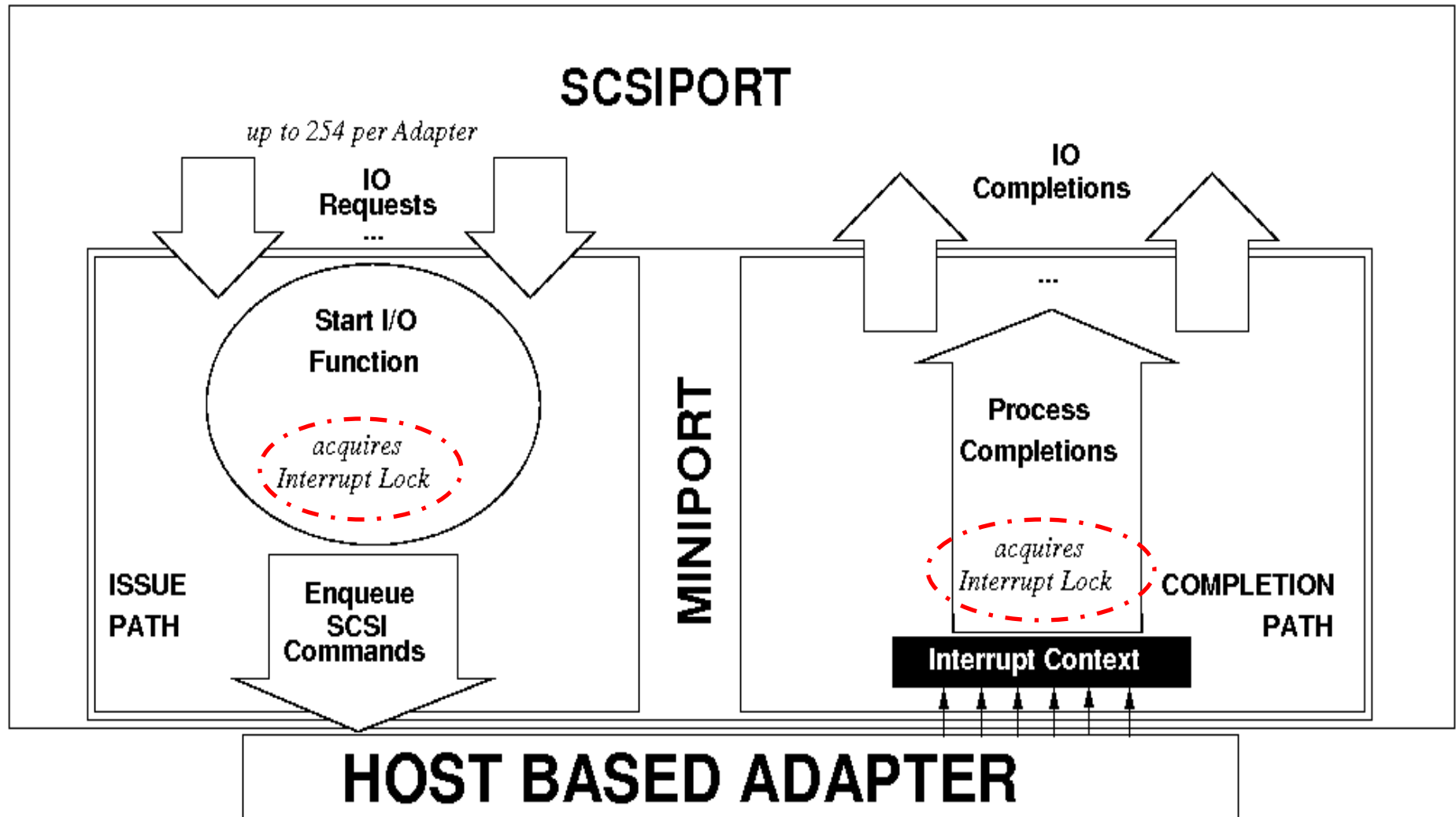
MS Windows Host S/W Stack



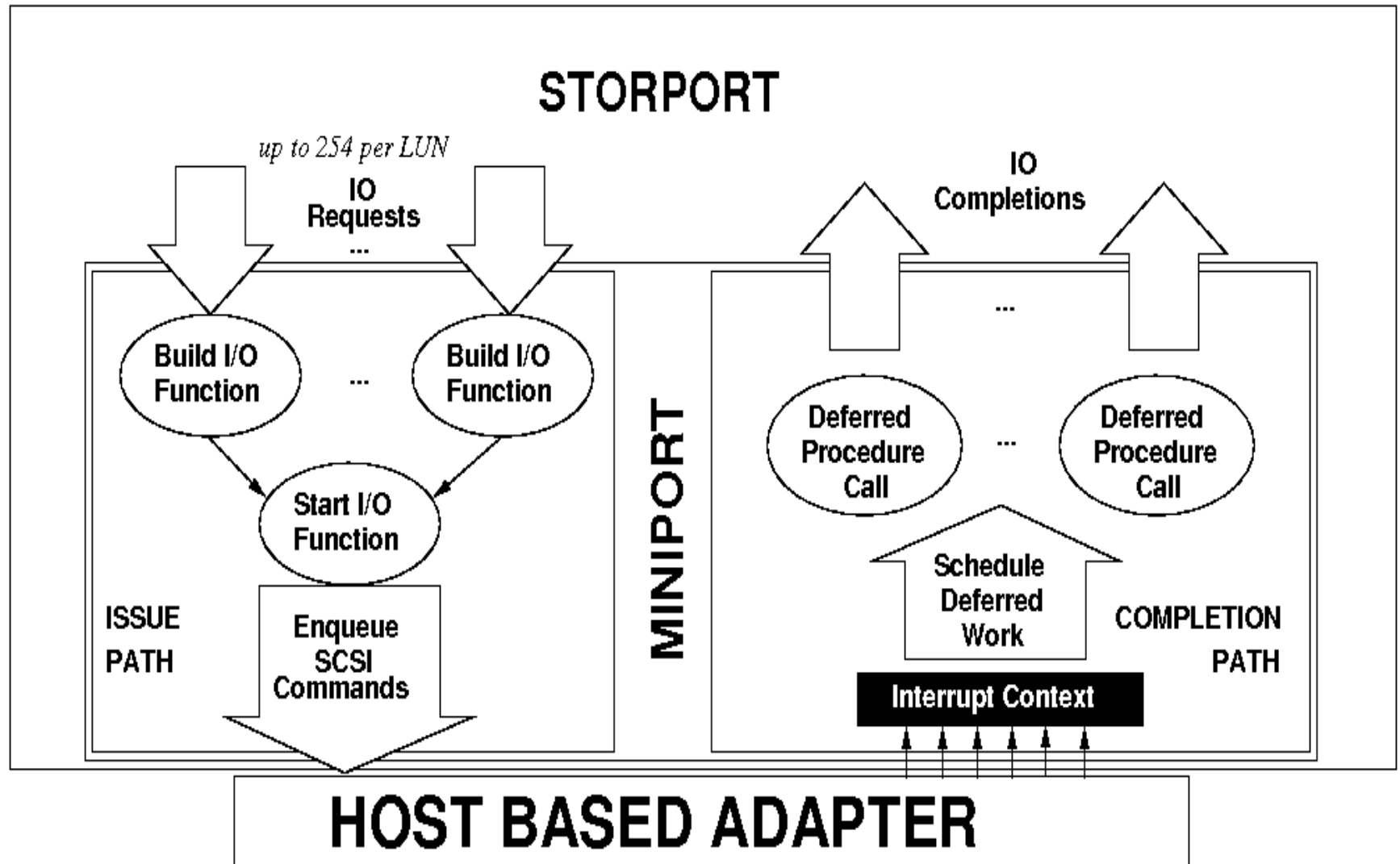
- ScsiPort: half-duplex
- StorPort: full-duplex

Direct manipulation of SCSI CDBs

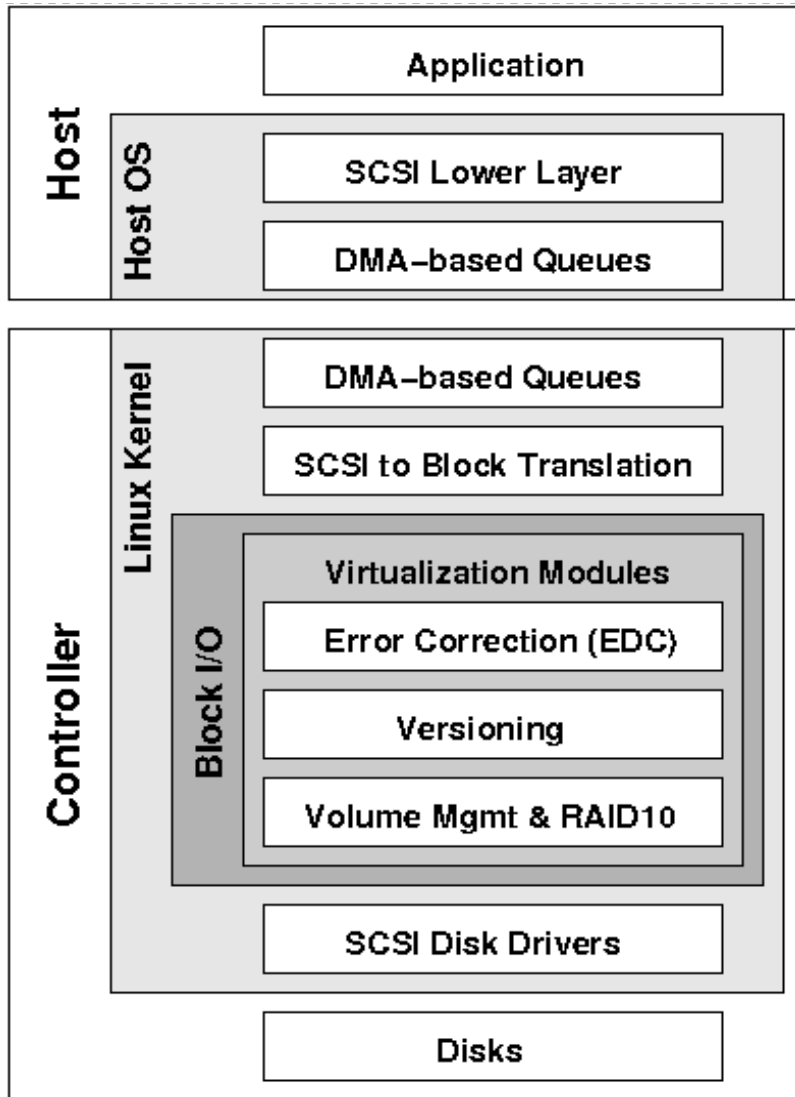
Half-Duplex: ScsiPort



Full-duplex: StorPort



DARC codebase



Module	Codelines
Virtualization Framework	32.5K
RAID-10	2K
Versioning	5K
EDC layer	0.5K
SCSI Host (Windows)	2.1K
SCSI Host (Linux)	2K
DMA queues, host (Linux, Windows)	2 x 0.85K
DMA queues, controller	1.65K
SCSI command processor	0.9K
I/O path control	1.6K
Buffer allocator	0.5K
Total	50K

Outline

- ▶ Background
 - ▶ Peripheral Communication Protocols
 - ▶ PCIe, SCSI, RAID
- ▶ DARC controller
- ▶ **Violin: A Framework for Extensible Block-level Storage**
- ▶ Evaluation
- ▶ Summary & Conclusions

Violin Block-level Framework

- ▶ **Goals**

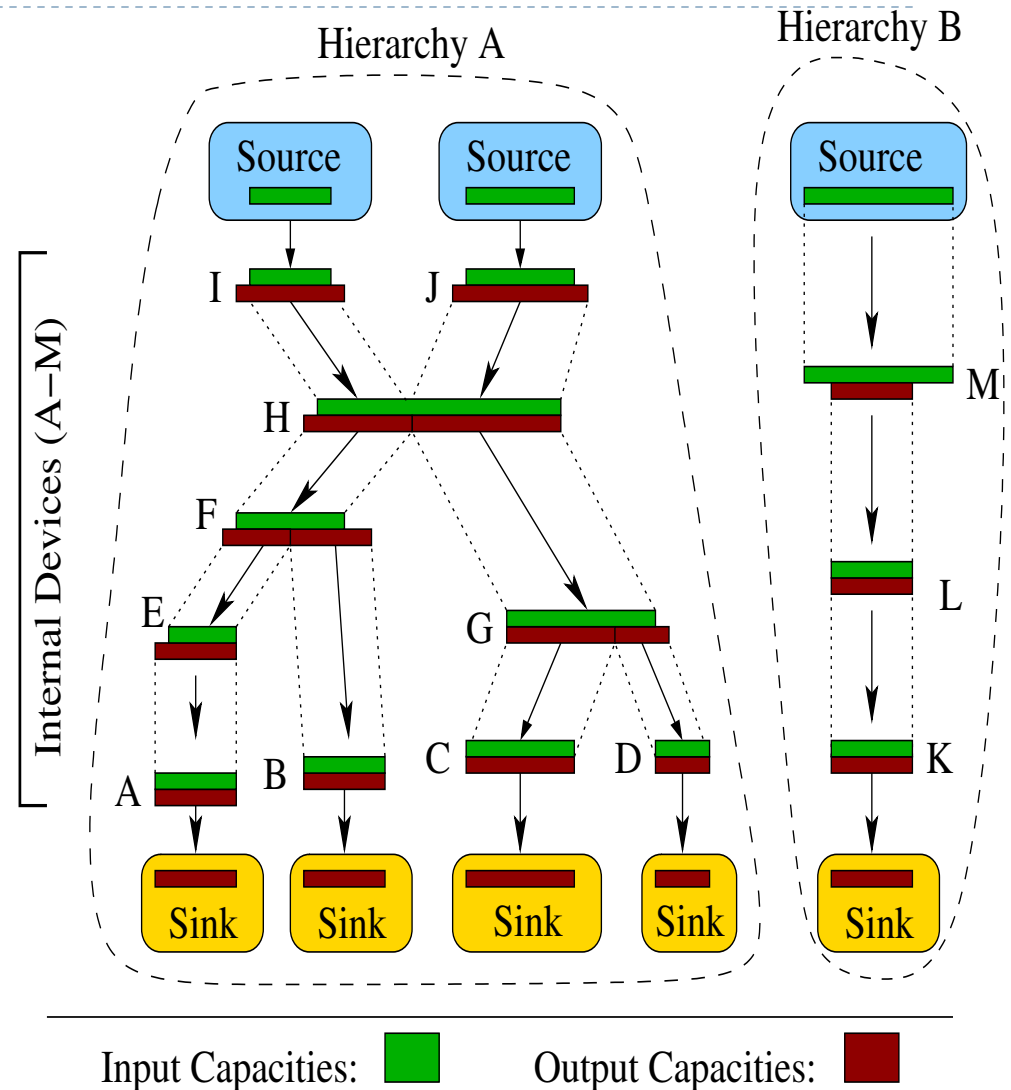
- ▶ Easy to develop new extensions
- ▶ Easy to combine them in I/O hierarchy
- ▶ Low overhead

- ▶ **Violin achieves this by providing**

- ▶ Convenient semantics and mappings
- ▶ Simple control of the I/O request path
- ▶ Persistent metadata support

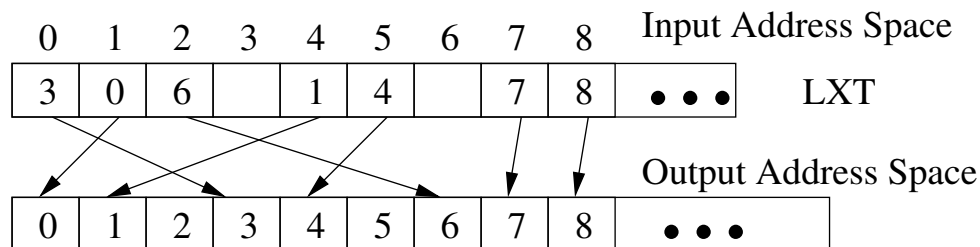
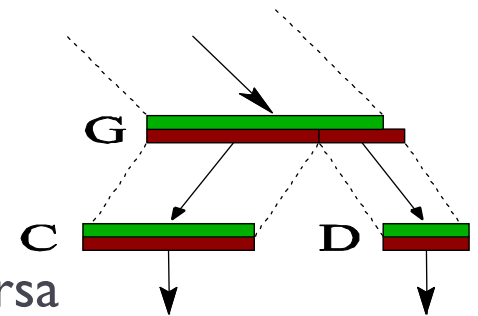
Virtualization Hierarchies

- ▶ Device graph maps I/O sources to I/O sinks
 - ▶ I/O requests pass through devices (layers)
- ▶ Nodes are virtual devices
- ▶ Edges are mappings
- ▶ Hierarchies: connected device sub-graphs, or independent I/O stacks
- ▶ Directed acyclic graph



Virtual Devices

- ▶ A virtual device is driven by an extension module
 - ▶ Device/Layer is runtime instance of module
 - ▶ Sees input/output address spaces, and one or more output devices
 - ▶ Maps arbitrarily blocks between devices
 - ▶ Transforms data between input & output, and vice versa
- ▶ Some modules need logical translation table (LXT)
 - ▶ A type of logical device metadata



Control of I/O Requests

- ▶ Violin API for layers to control I/O requests passing through them
- ▶ Layer can initiate, forward, complete or terminate I/O requests using simple tags or calls
- ▶ Initiating I/O: fan-out of I/O flows
 - ▶ Layers initiate asynchronous I/O requests using callback handlers, executed on completion
- ▶ Forward I/O: Send I/O request to a lower layer
- ▶ Complete I/O: Can be used by caching layers
- ▶ Terminate I/O: Error-handling

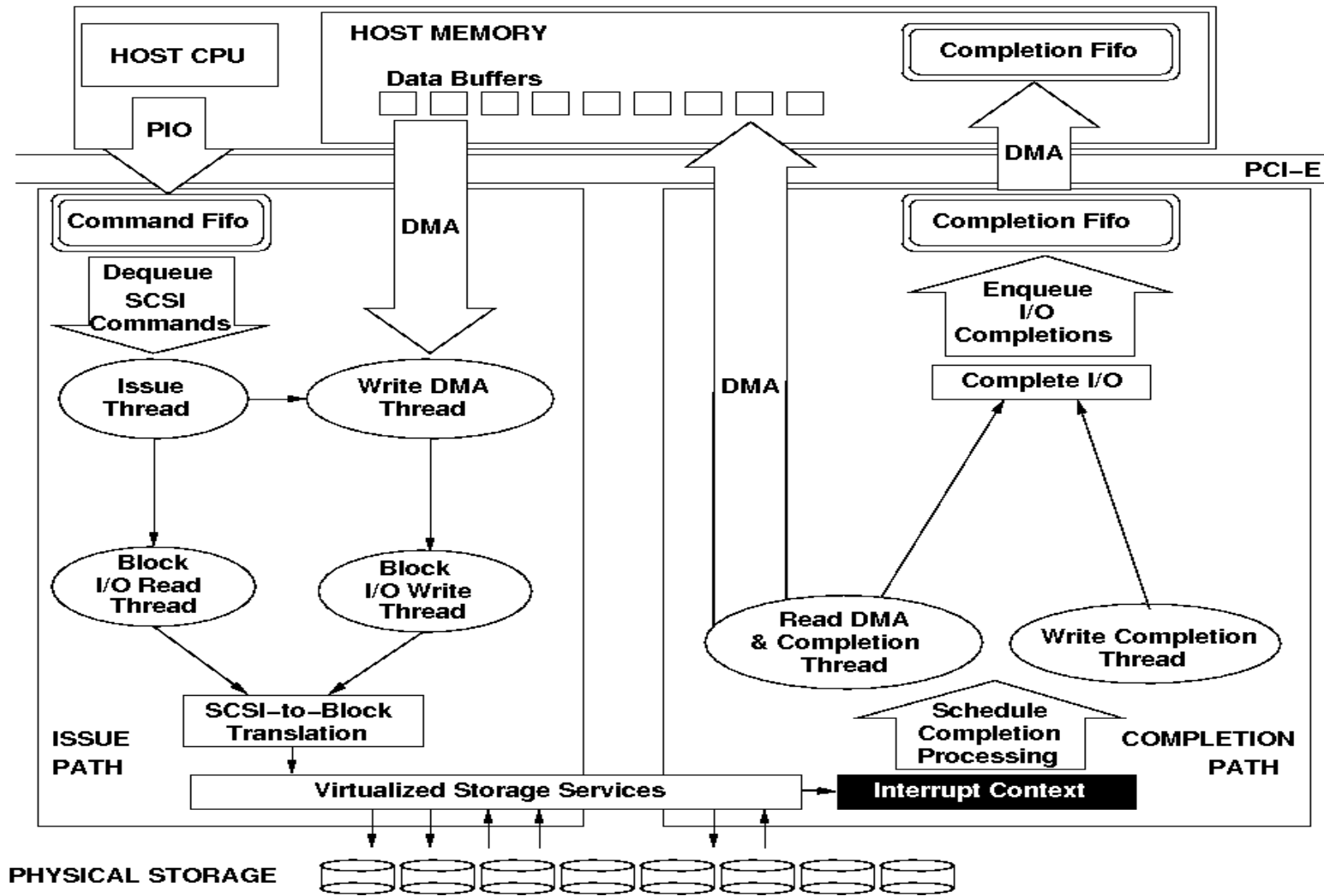
Persistent Metadata

- ▶ **Storage layers need persistent state**
 - ▶ For superblocks, partition tables, block maps, etc.
- ▶ **Violin offers persistent objects for layer metadata**
 - ▶ Persistent Objects are memory-mapped storage blocks, accessed as generic memory objects
 - ▶ Automatically synchronized to stable storage periodically
 - ▶ Automatically loaded / unloaded during startup / shutdown
 - ▶ Layers need only allocate objects once
- ▶ **Violin internal metadata are also persistent objects**
 - ▶ Device graph and hierarchy info stored at superblocks

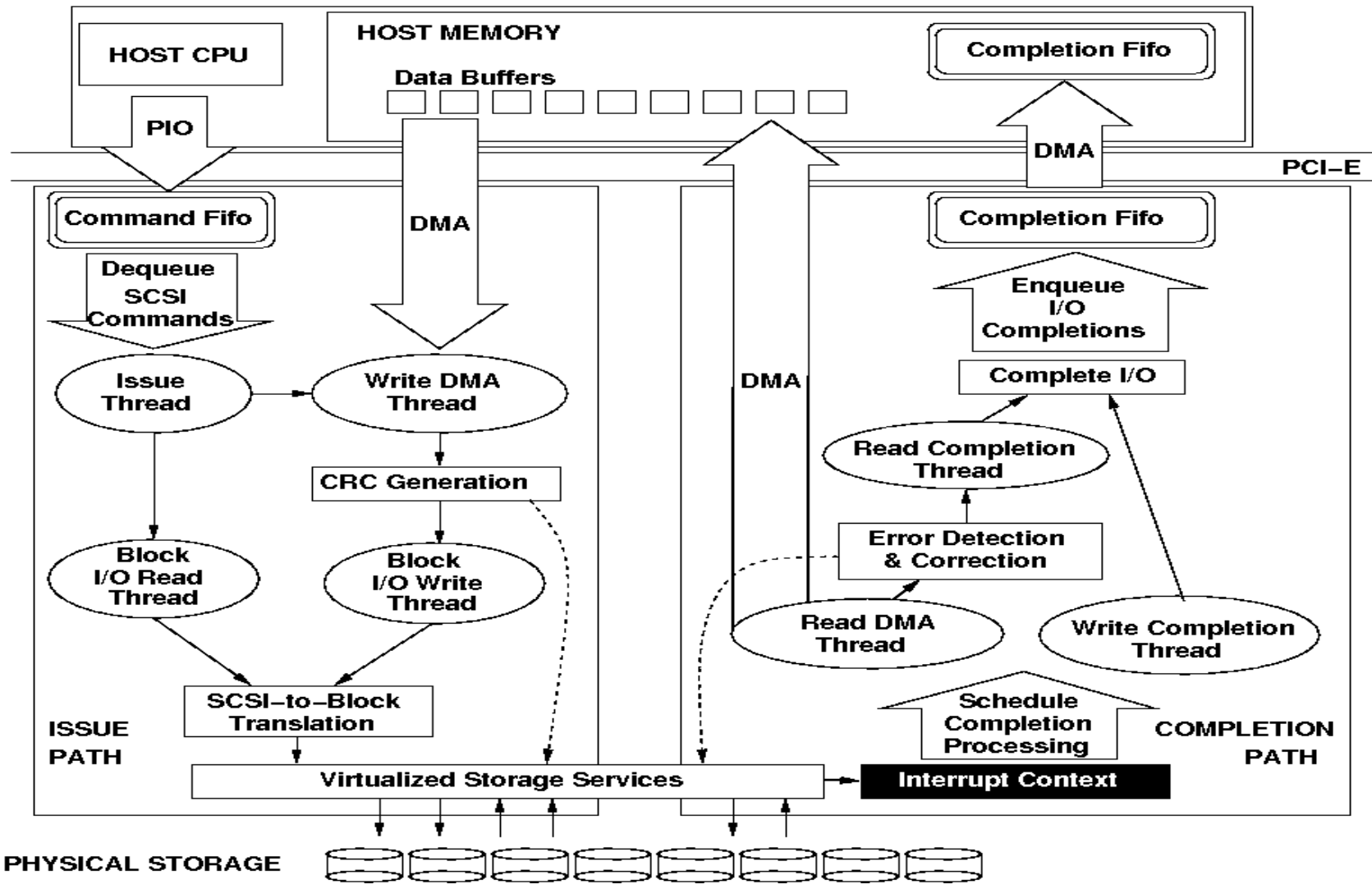
Design Choices for DARC

Challenge	Decision
Host-controller I/O path	PIO for commands, DMA for data and completions
Buffer Management	Pre-allocated buffer pools, lazy de-allocation, circular FIFO queues
Context Scheduling	Map I/O path stages to threads, explicit scheduling, no processing in IRQ context
Error Detection & Correction	CRC32-C checksums computed by DMA engines for 4KB blocks, persistently stored
Storage Virtualization	Block-level virtualization framework, supports RAID-10 volume management, versioning and EDC modules
Data-block cache	On-board cache <u>omitted</u> in <i>DARC</i>

I/O Path (no CRC checks)



I/O Path (with CRC checks)

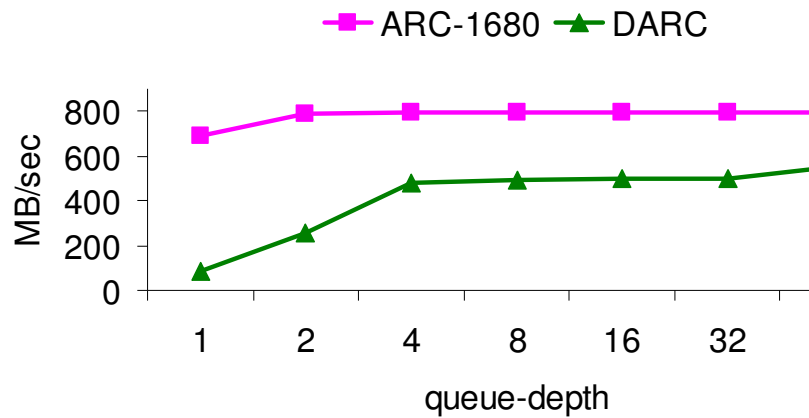


Outline

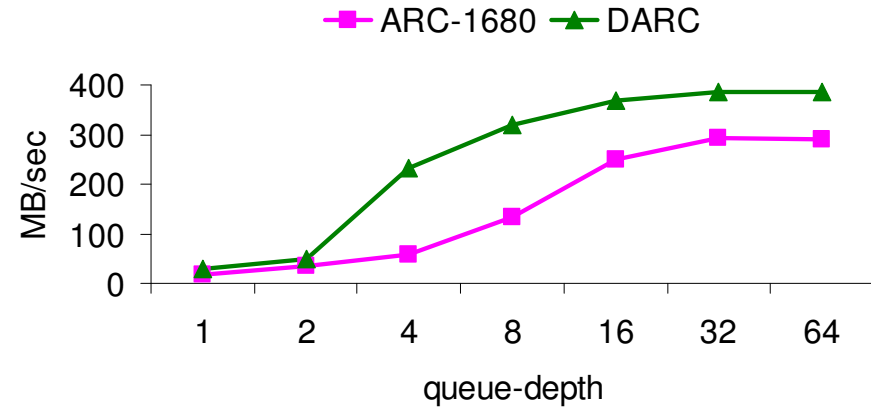
- ▶ **Background**
 - ▶ Peripheral Communication Protocols
 - ▶ PCIe, SCSI, RAID
- ▶ DARC controller
- ▶ Violin: A Framework for Extensible Block-level Storage
- ▶ **Evaluation**
- ▶ Summary & Conclusions

IOmeter results: RAID-10 (1)

RS Iometer Pattern

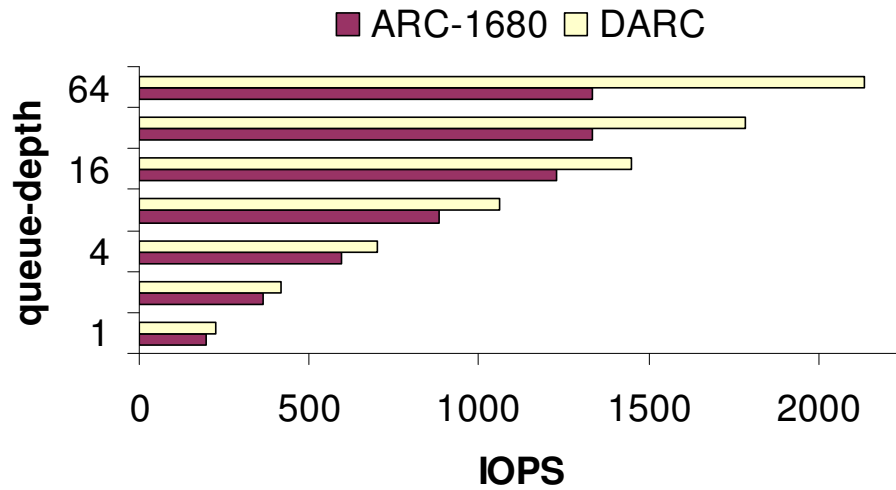


WS Iometer Pattern

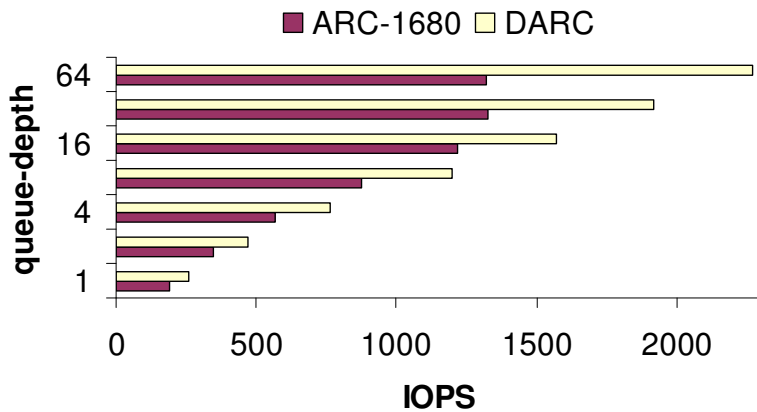
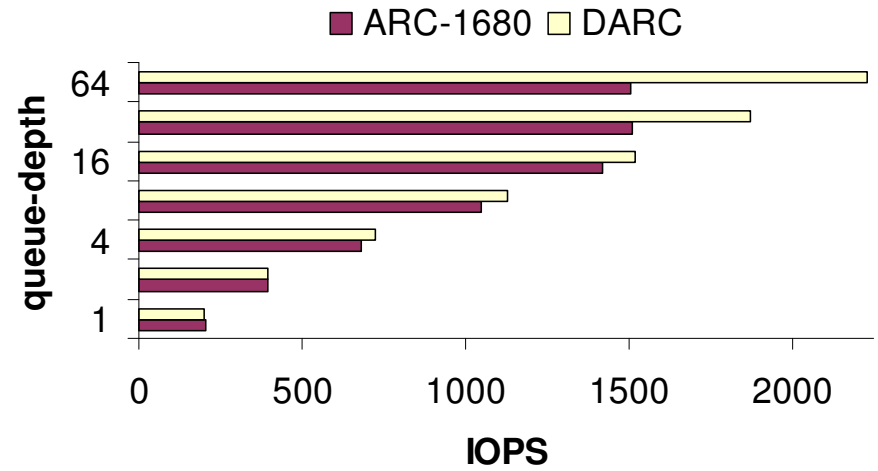


IOmeter results: RAID-10 (2)

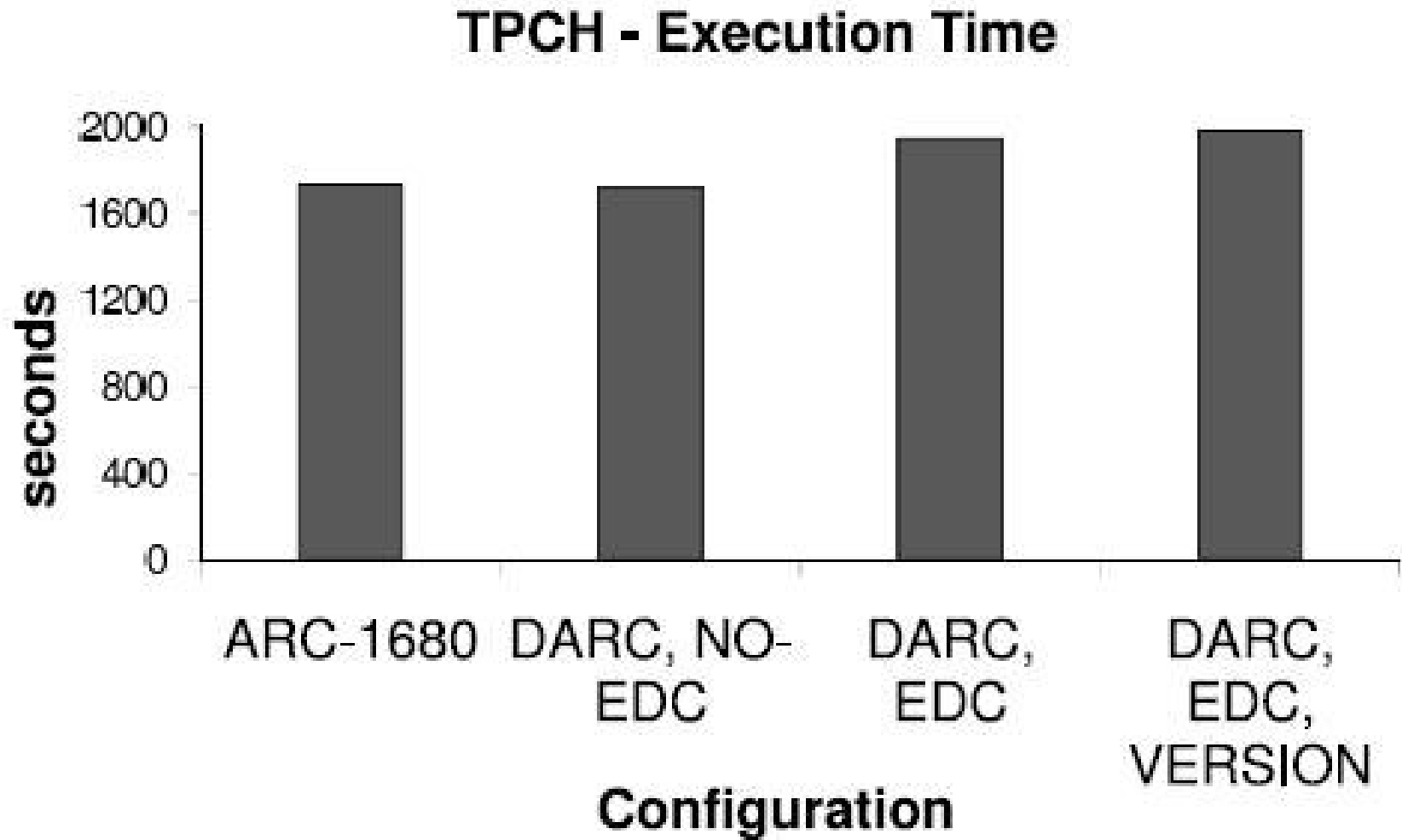
FS Iometer Pattern



WEB Iometer Pattern

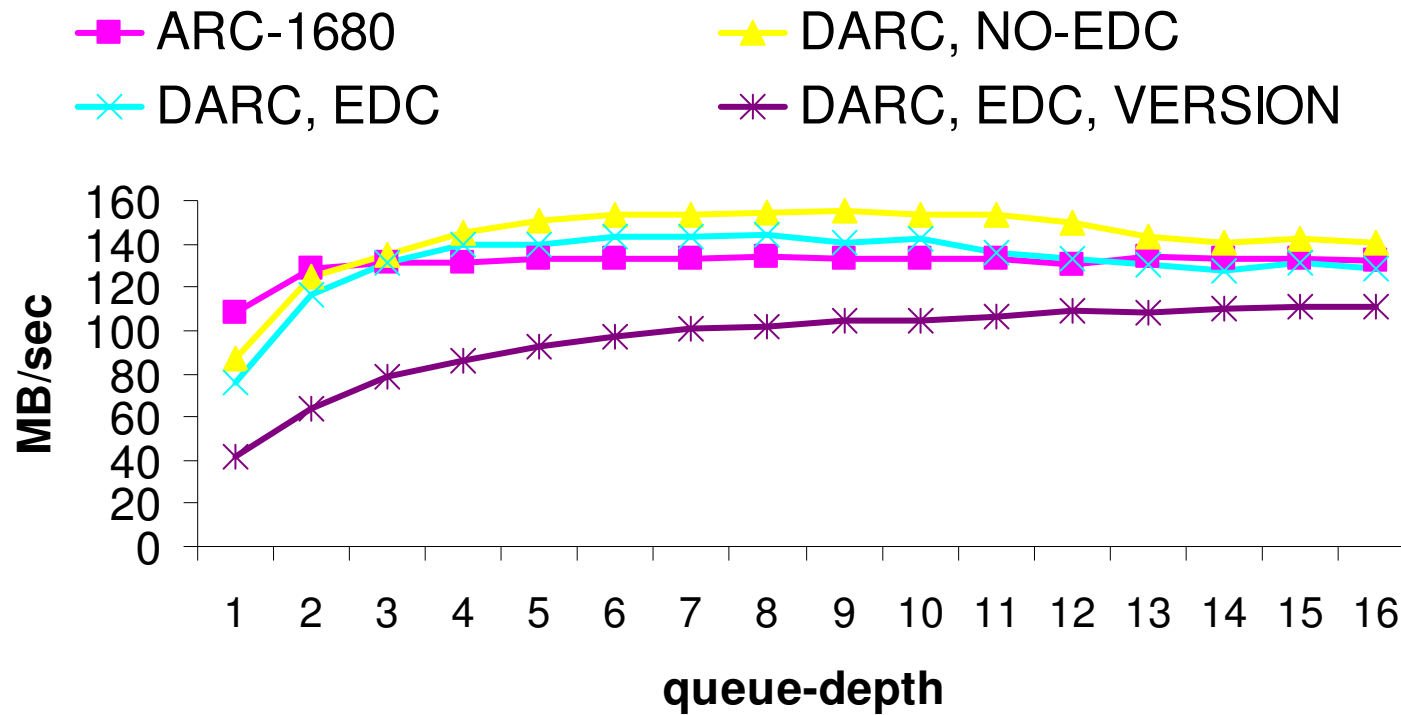


TPC-H results (RAID-10)



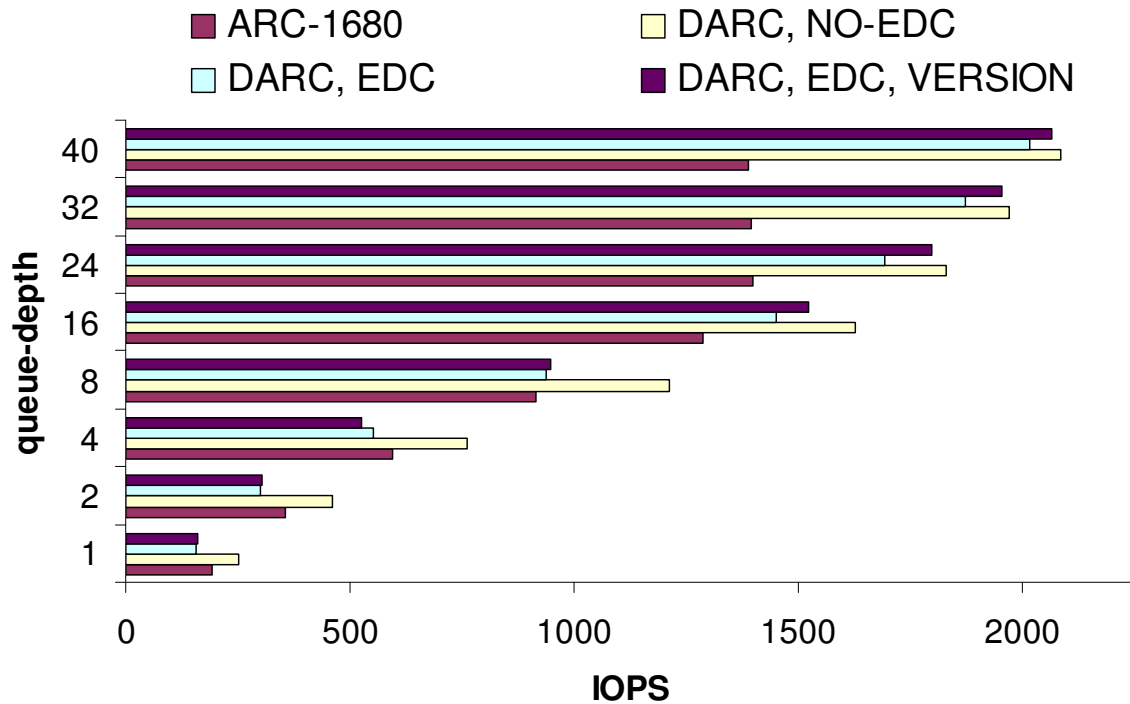
Orion - Throughput (RAID-10)

ORION - I/O Throughput (1MB)



Orion - IOPS (RAID-10)

ORION - IOPS (8KB)



Configuration	Max IOPS	MaxMB/sec
<i>DARC, NO-EDC</i>	2088	154.96
<i>DARC, EDC</i>	2018	144.39
<i>DARC, EDC, VERSION</i>	2067	110.82
<i>ARC-1680</i>	1402	133.90

JetStress results (RAID-10)

- ▶ DATA & LOG on same 8-disk volume
- ▶ 6-disk DATA volume, separate 2-disk LOG volume

Configuration	IOPS data-volume		IOPS log-volume
	READ	WRITE	
<i>DARC</i> , NO-EDC	679.32	556.79	70.32
<i>DARC</i> , EDC	544.41	456.17	54.2
<i>DARC</i> , EDC, VERSION	516.43	434.65	53.95
ARC-1680, write-back	774.94	703.1	626.4
ARC-1680, write-through	505.32	425.93	39.99

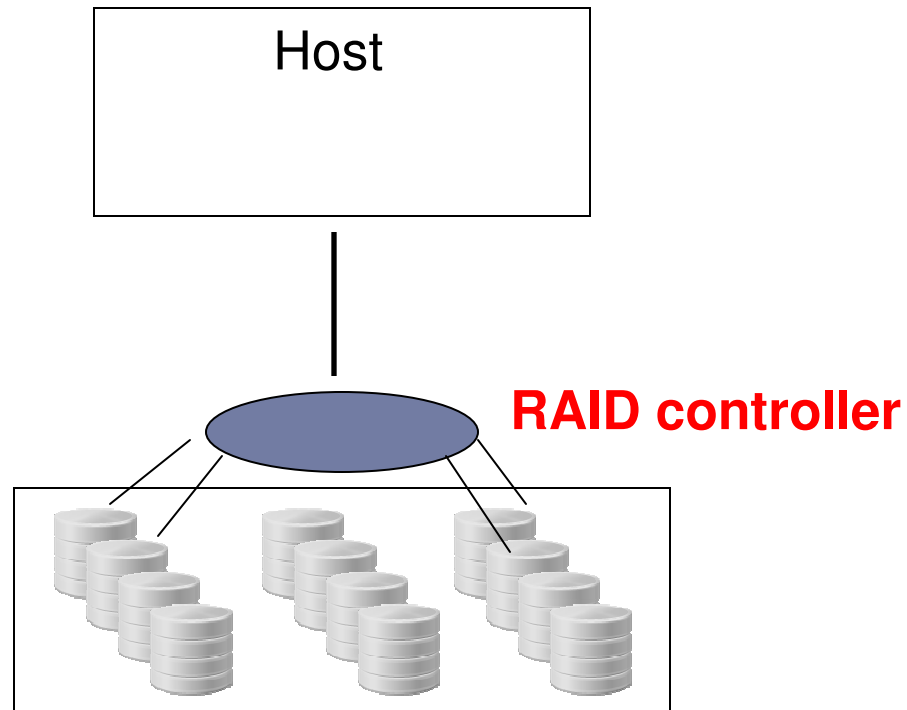
Conclusions (1)

- ▶ **Overhead of EDC checking: 12 - 20%**
 - ▶ Depending on # concurrent I/Os
- ▶ **Overhead of versioning: 2.5 - 5%**
 - ▶ With periodic (frequent) capture & purge
 - ▶ Depending on number and size of writes
- ▶ **CPU overhead at controller very important**
 - ▶ Memory management
 - ▶ Scheduling
 - ▶ Host PIO vs. DMA tradeoffs, especially completions

Conclusions (2)

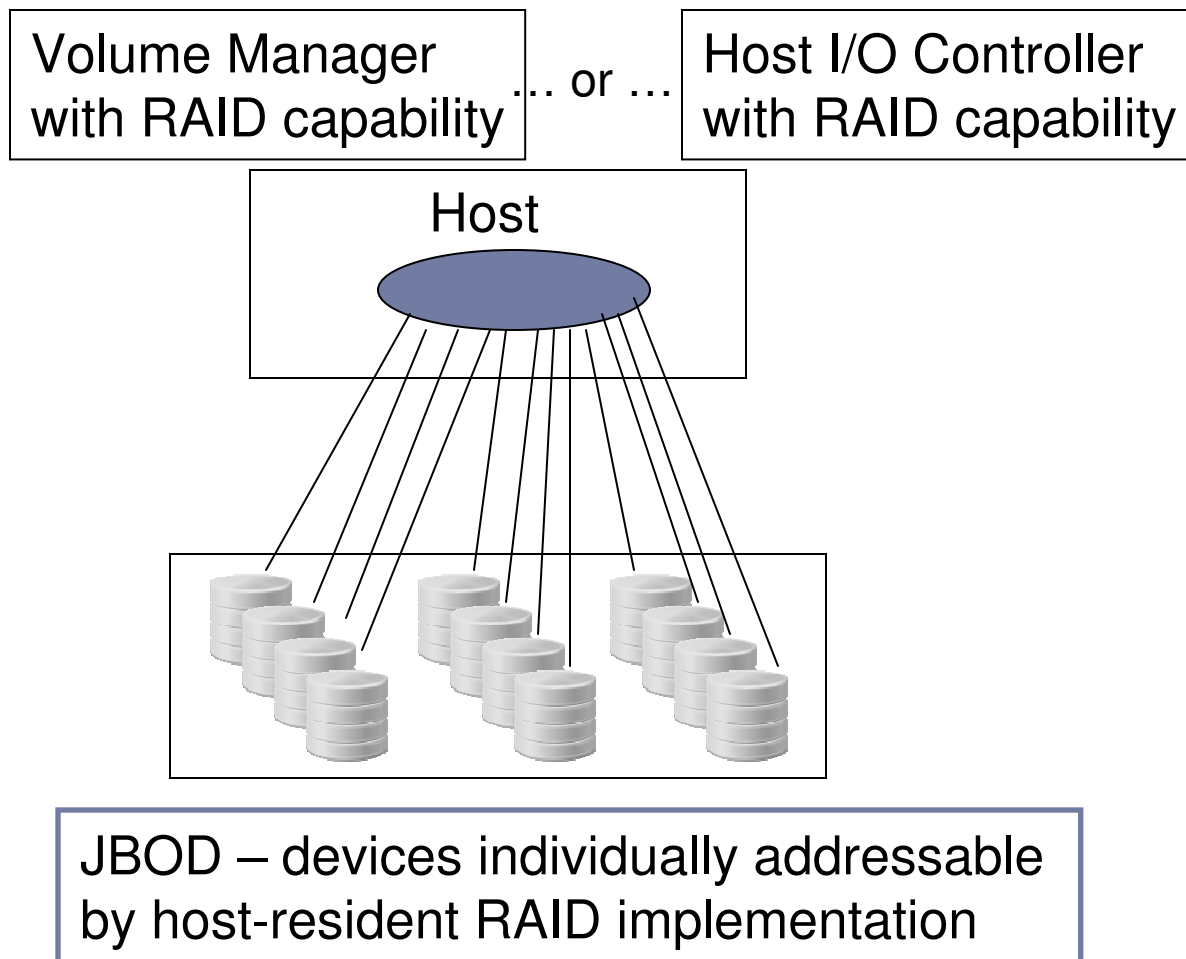
- ▶ **Replacing custom I/O firmware with OS-based run-time viable option**
 - ▶ Comparable performance to custom firmware
 - ▶ Transparent to host-side stack
 - ▶ Several implementation challenges
- ▶ **Data protection features beyond RAID**
 - ▶ Error-detecting/correcting codes, Timeline of Snapshots, Live Migration, Compression + Deduplication, ...
 - ▶ Many features required in the I/O path
 - ▶ But: performance + power not even close!

RAID Systems: Controller-based



RAID subsystem – presented as a single virtual device

RAID Systems: Host-based



What if ...

- ▶ What if we could blur the distinction between “memory” and “storage” ?
 - ▶ Memory → fast but volatile, expensive per capacity unit
 - ▶ Storage → slow but non-volatile, cheap per capacity unit
- ▶ Solid-state, instead of electromechanical ?
- ▶ Could we interface to this type of storage without a dedicated “peripheral” controller ?

Treat storage as a part of the memory hierarchy, not as an interaction with an external “foreign” environment

Design & Implementation of a High-performance I/O Path for Data Protection

Thank you for your attention!

Questions?

Manolis Marazakis

maraz@ics.forth.gr

<http://www.ics.forth.gr/carv>