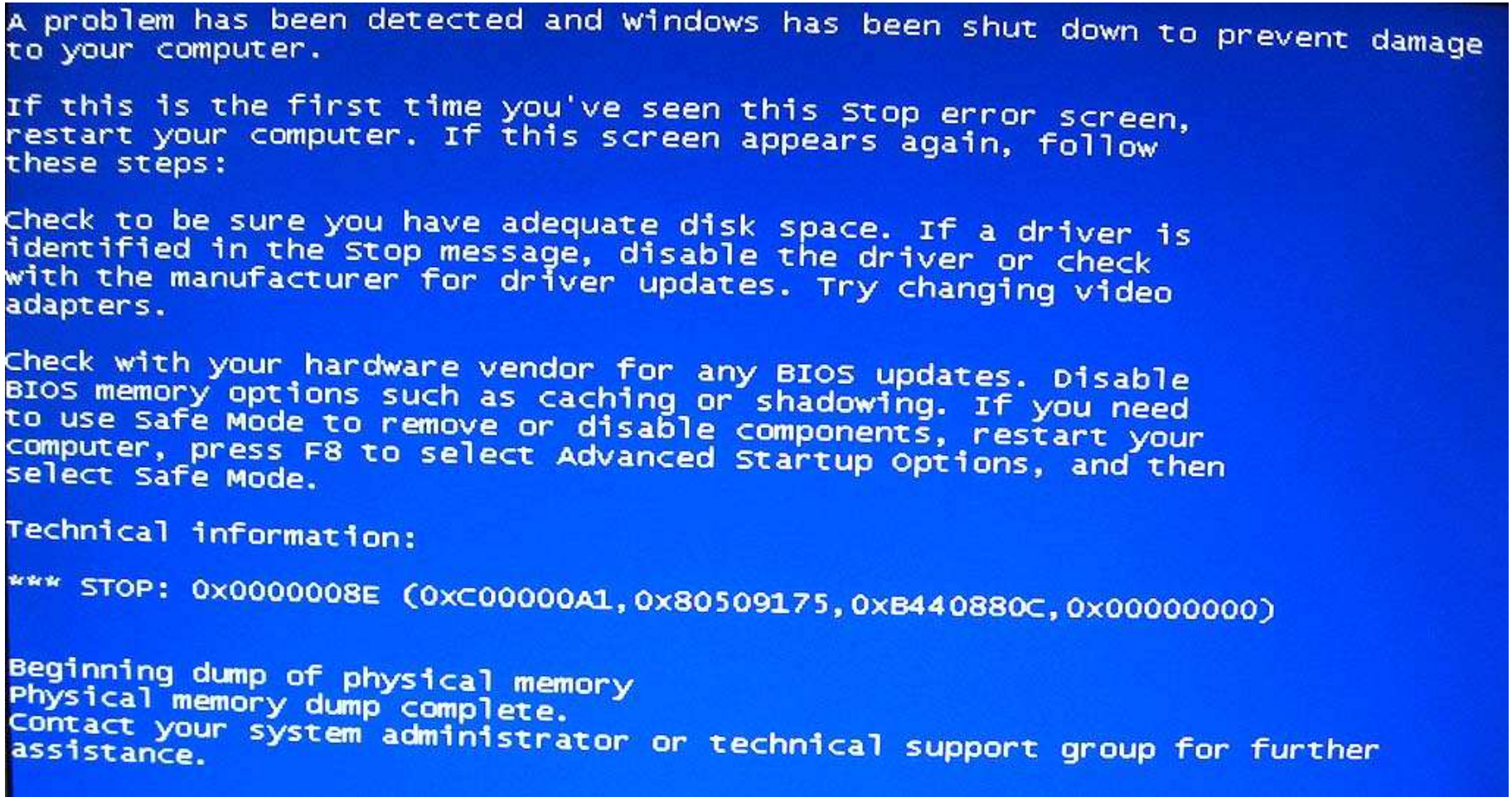
Scalable I/O—A Well-Architected Way to Do Scalable, Secure and Virtualized I/O

Julian Satran Leah Shalev Muli Ben-Yehuda Zorik Machulsky

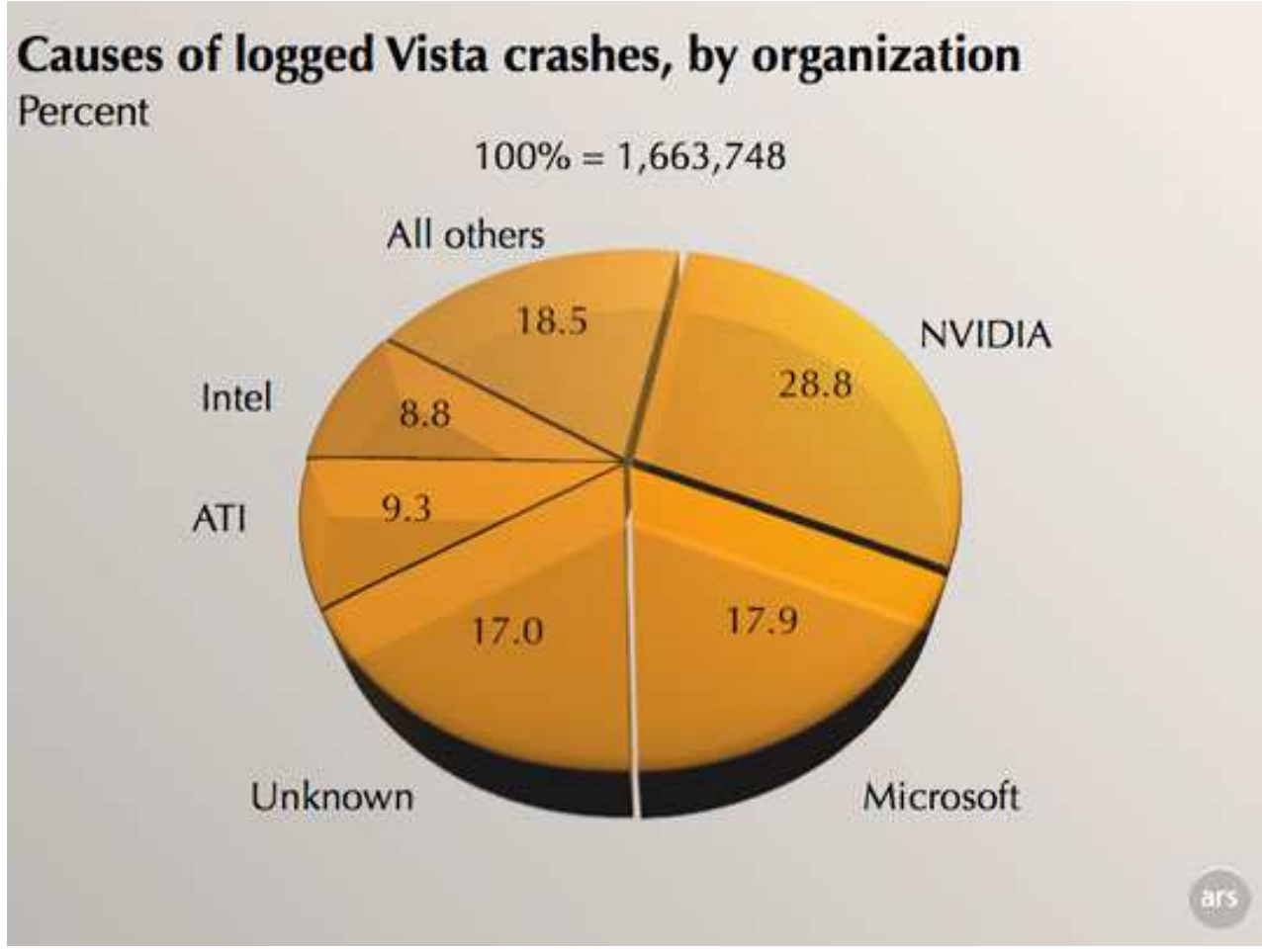
satran@il.ibm.com leah@il.ibm.com muli@il.ibm.com machulsk@il.ibm.com

IBM Haifa Research Lab, Haifa, Israel

The Problem...


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to be sure you have adequate disk space. If a driver is
identified in the Stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.
Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x0000008E (0xC00000A1,0x80509175,0xB440880C,0x00000000)
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

... is Device Drivers



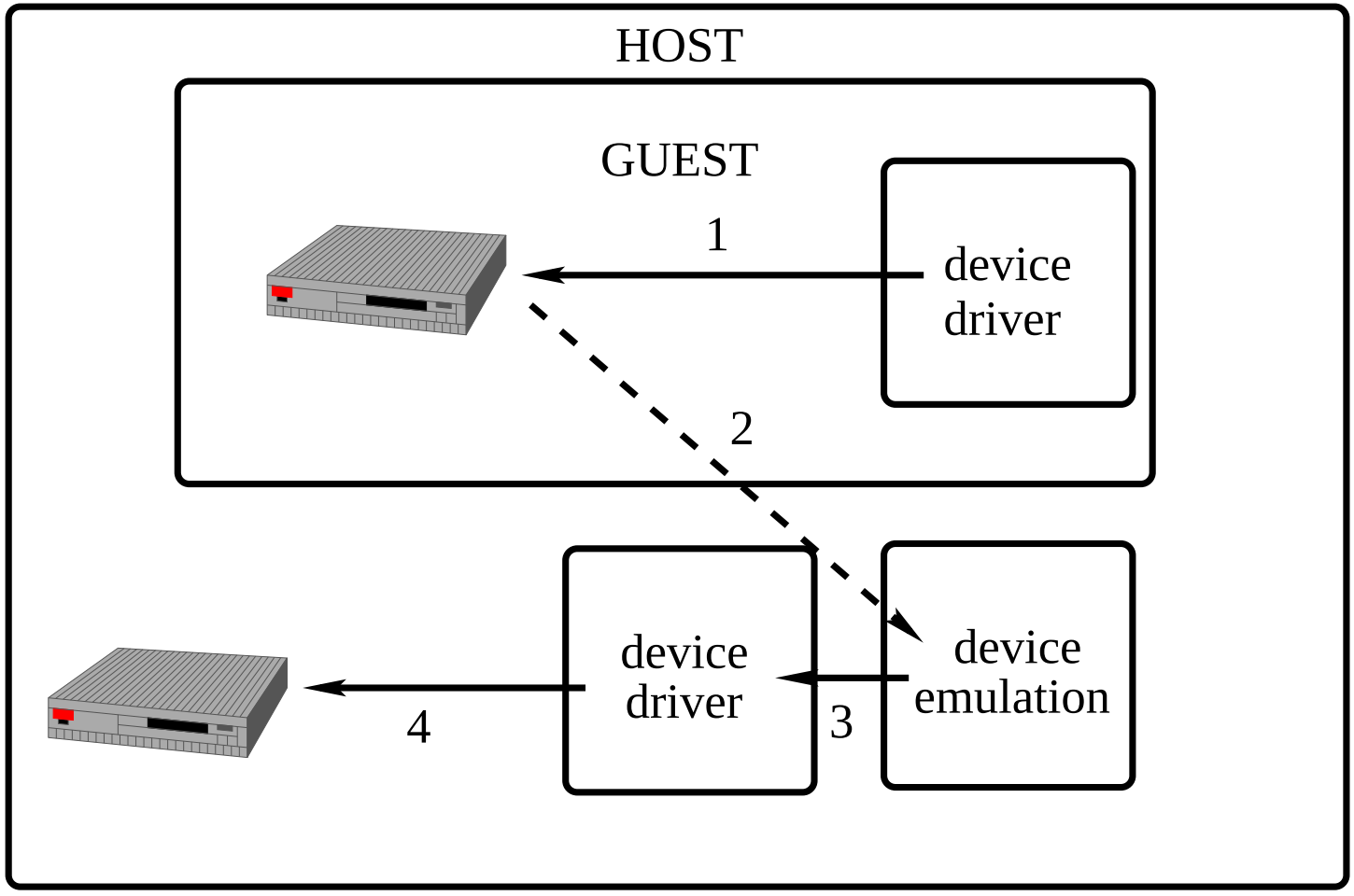
ars technica: `http://is.gd/384`

Virtualization makes it worse

I/O virtualization

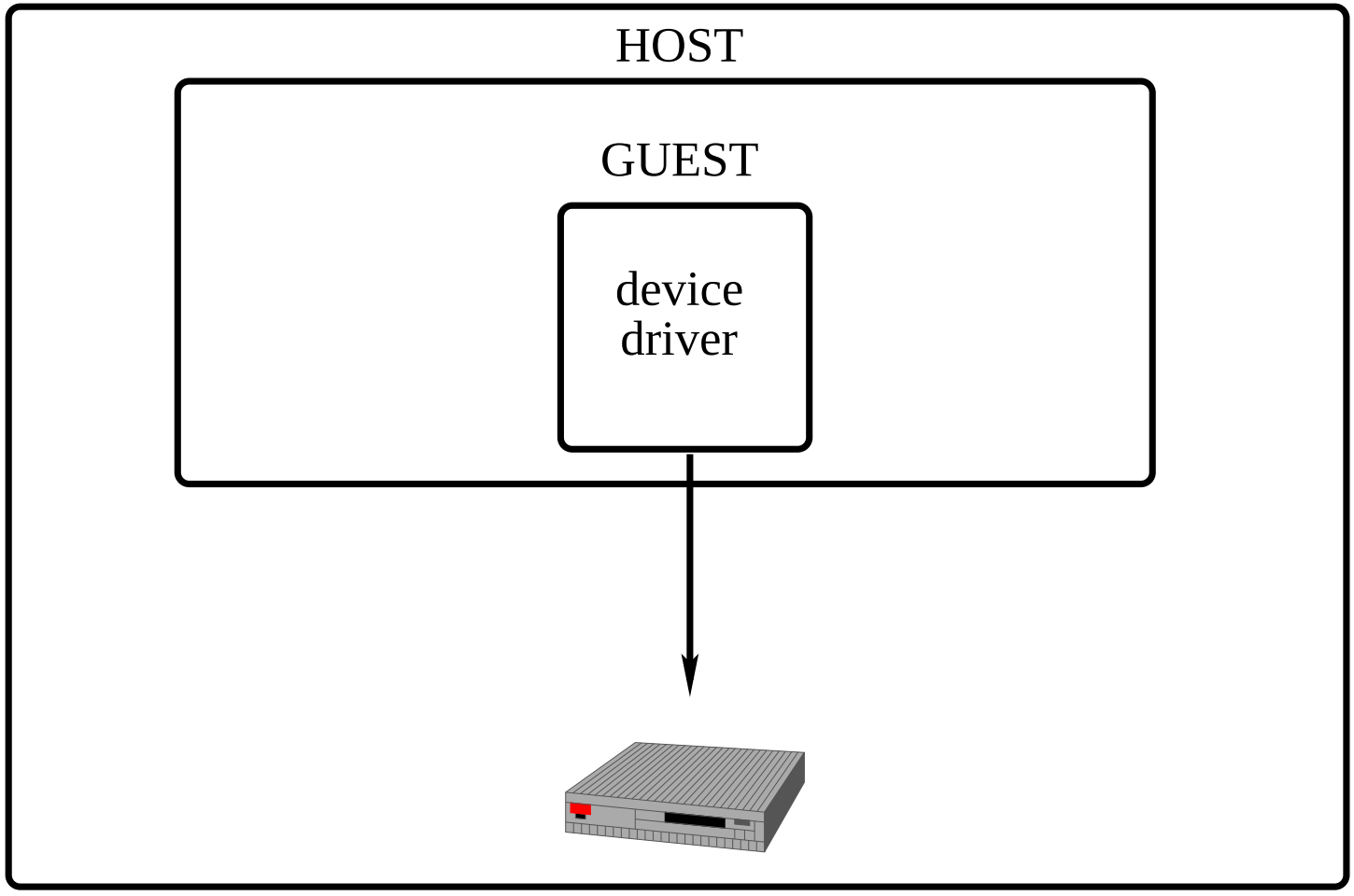
An extremely short primer on I/O virtualization

I/O Virtualization: Emulation



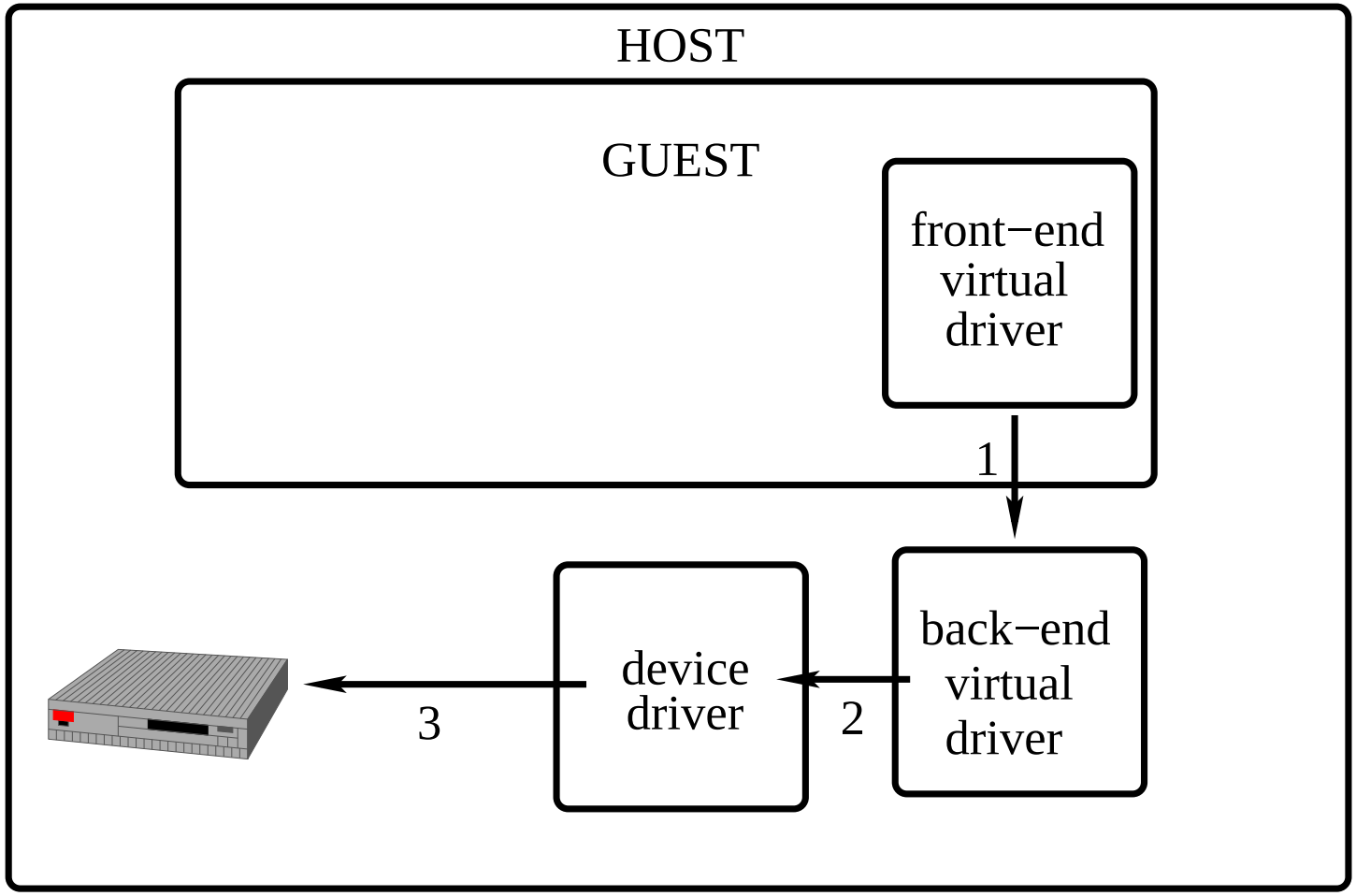
[Sugerman '01]

I/O Virtualization: Direct Access



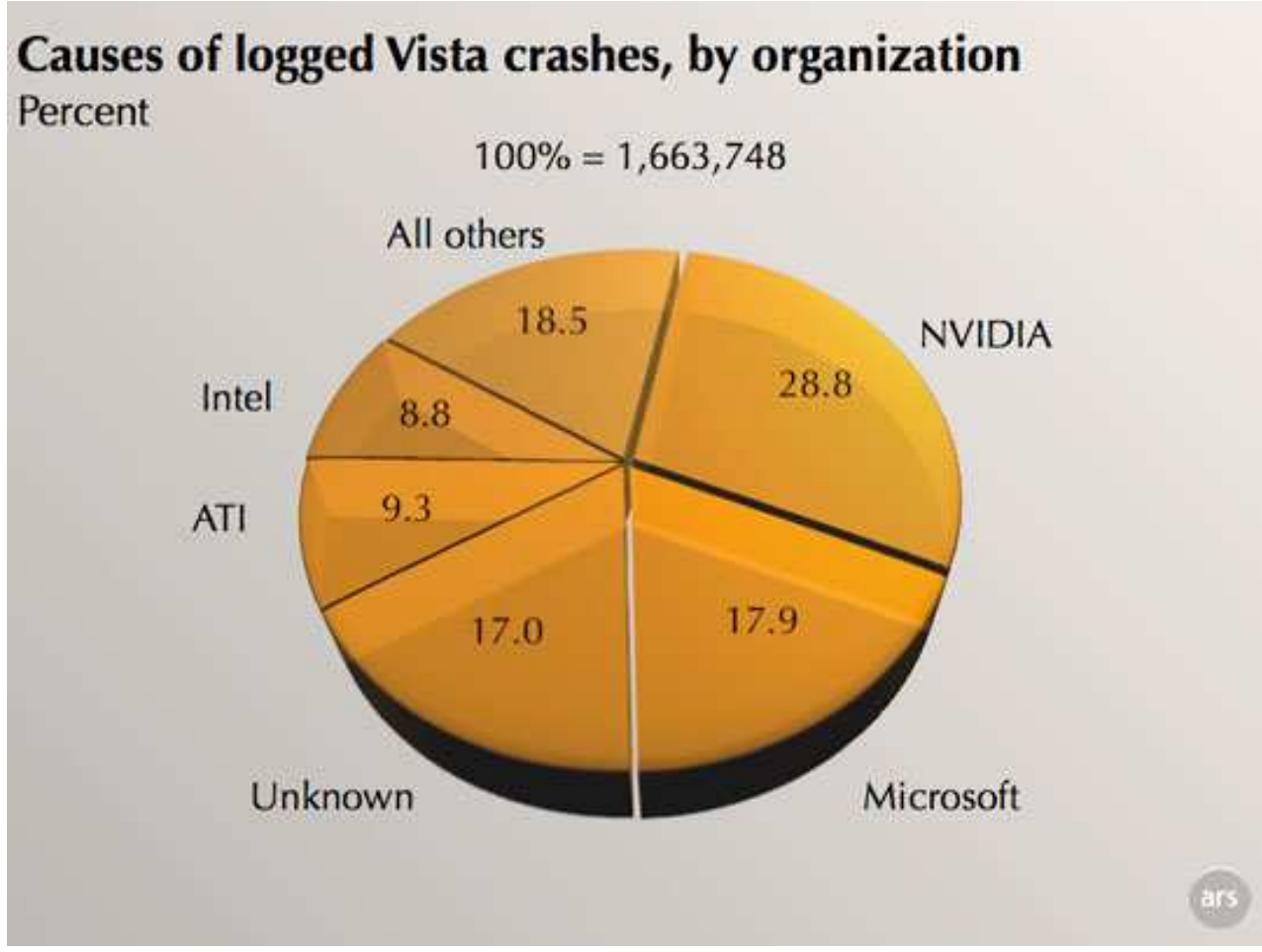
[Levasseur ’04, Liu ’06, Ben-Yehuda ’06, Willman ’07, Raj ’07]

I/O Virtualization: Virtual Drivers

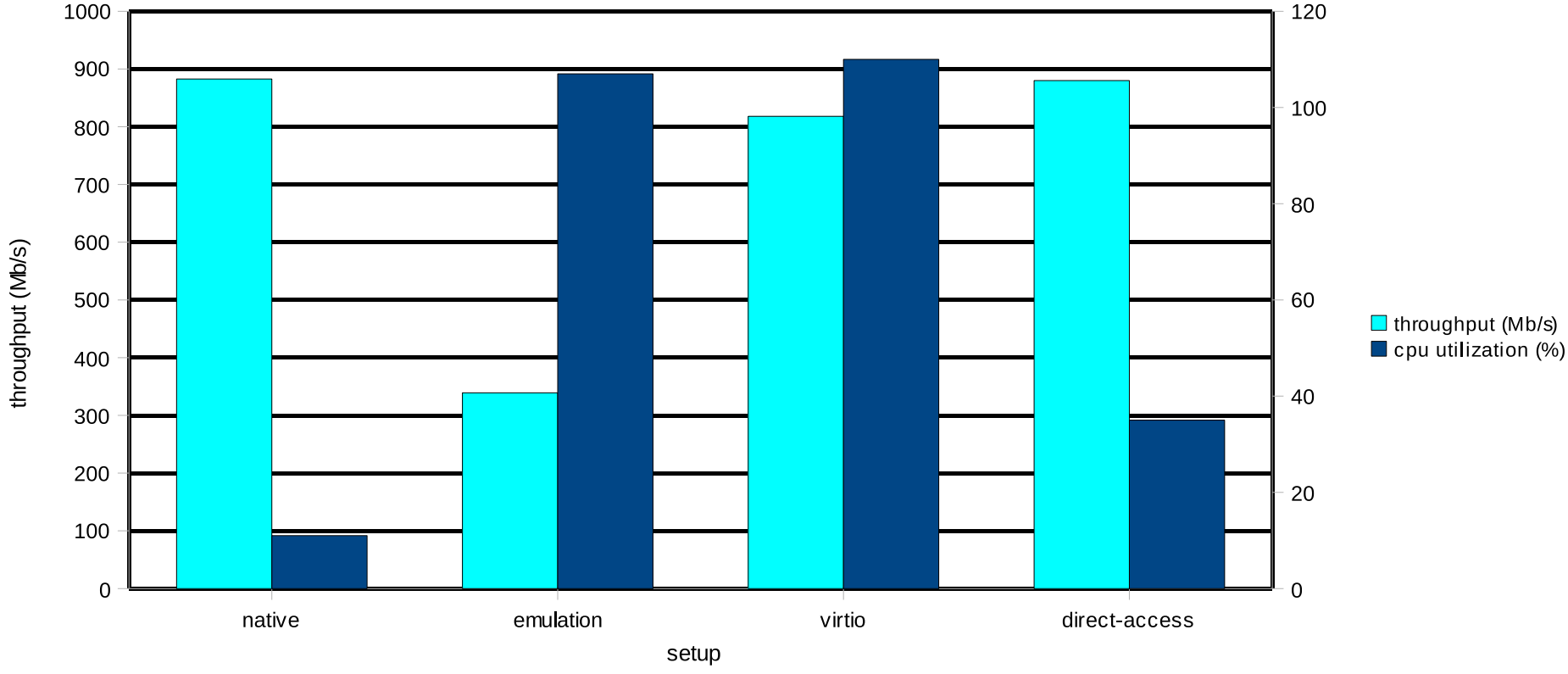


[Barham ’03, Kivity ’07, Russell ’08]

Reliable?


Causes of logged Vista crashes, by organization
Percent
100% = 1,663,748
All others
18.5
NVIDIA
28.8
Intel
8.8
ATI
9.3
17.0
17.9
Unknown
Microsoft
ars

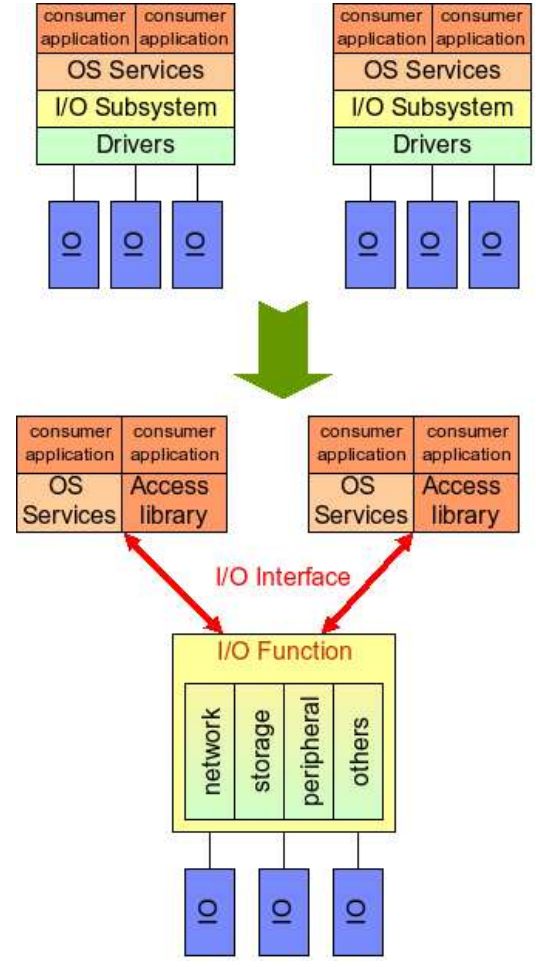
Performant?



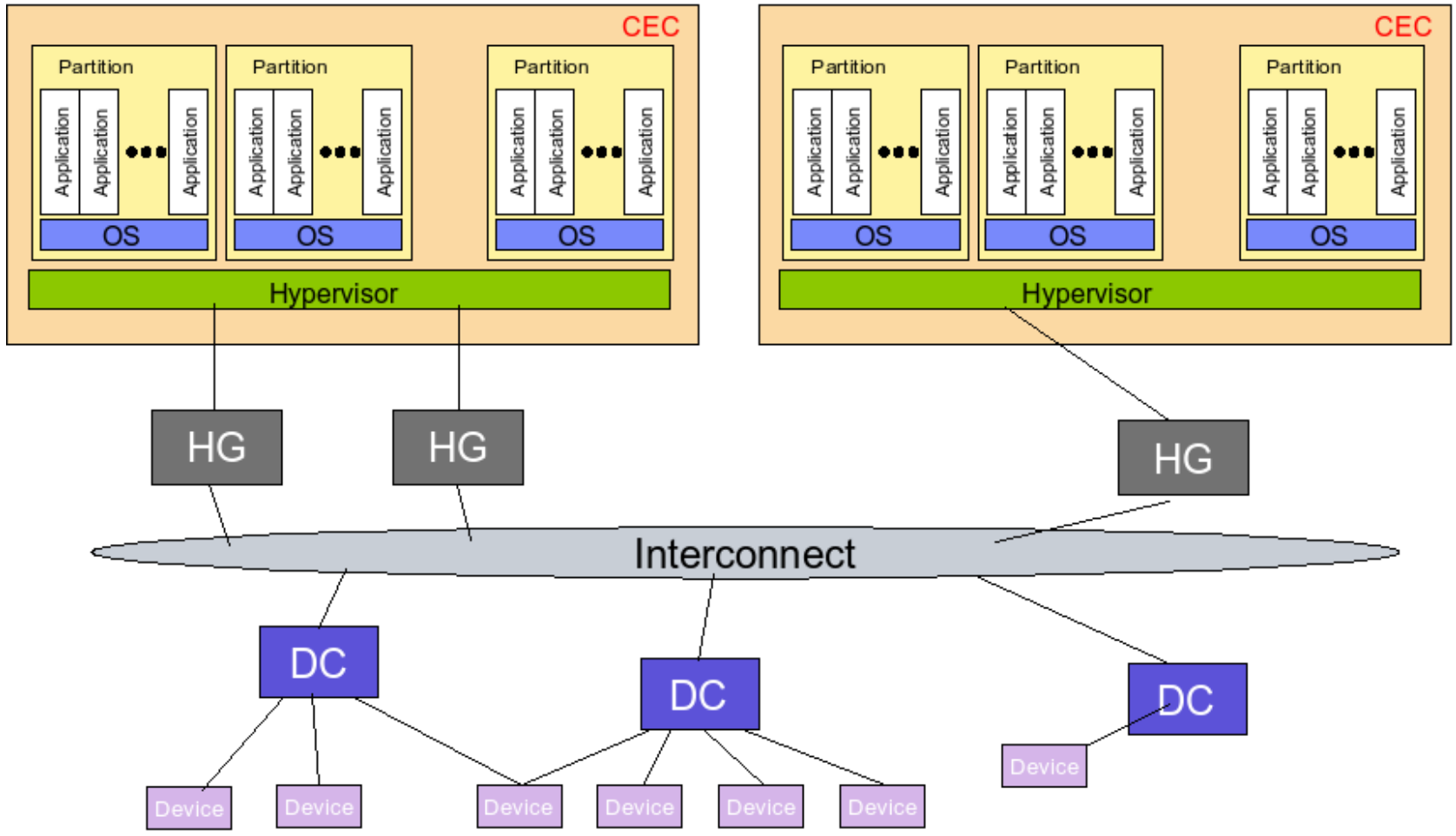
[Yassour ’08, Santos ’08, Ram ’09]

Scalable I/O

Virtualizes the
entire I/O subsystem
rather than specific
drivers
or
devices.



Architecture

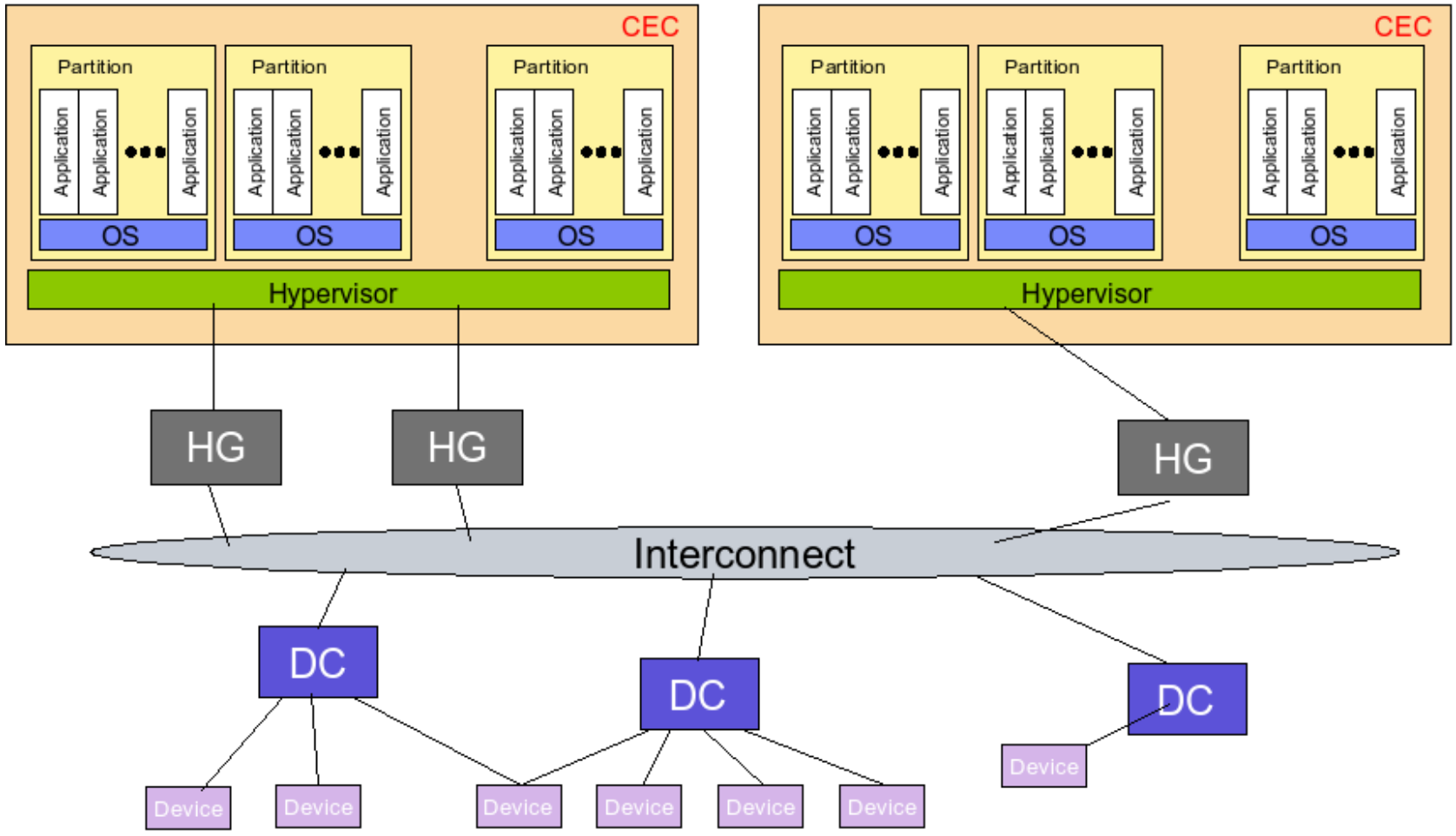


Mainframe I/O channels [Cormier '83]

Main Components and Concepts

- Device Controllers.
- Host Gateways.
- I/O consumers (clients).
- Security model.
- Protected DMA (PDMA).

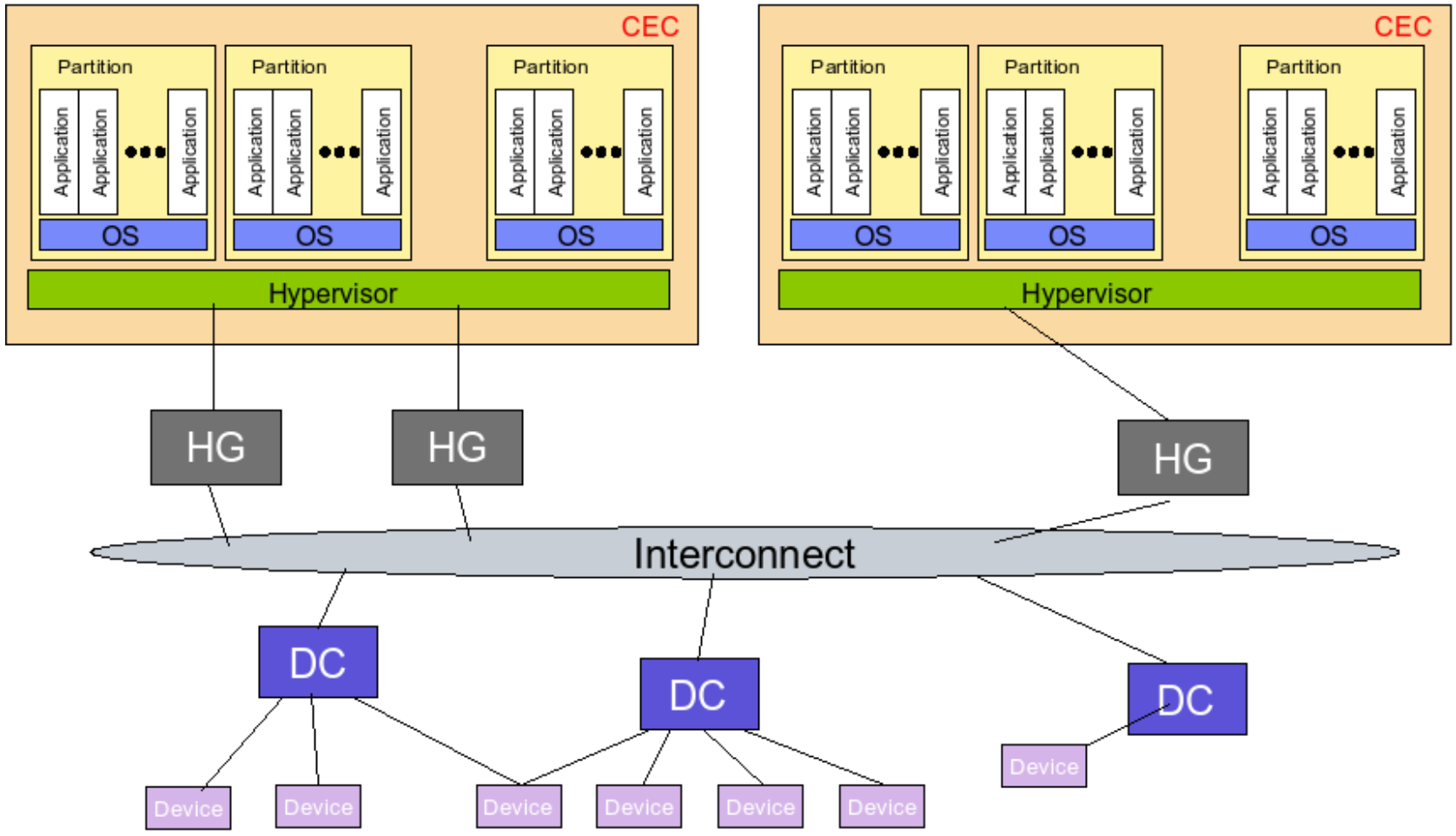
Architecture



Device Controllers

- DCs implement I/O services.
- DCs control I/O devices and run device specific code.
- May be implemented in software or hardware (separate core, separate VM, device firmware, etc.)
- Can service multiple *I/O consumers*.
- DCs protect devices from unauthorized access.

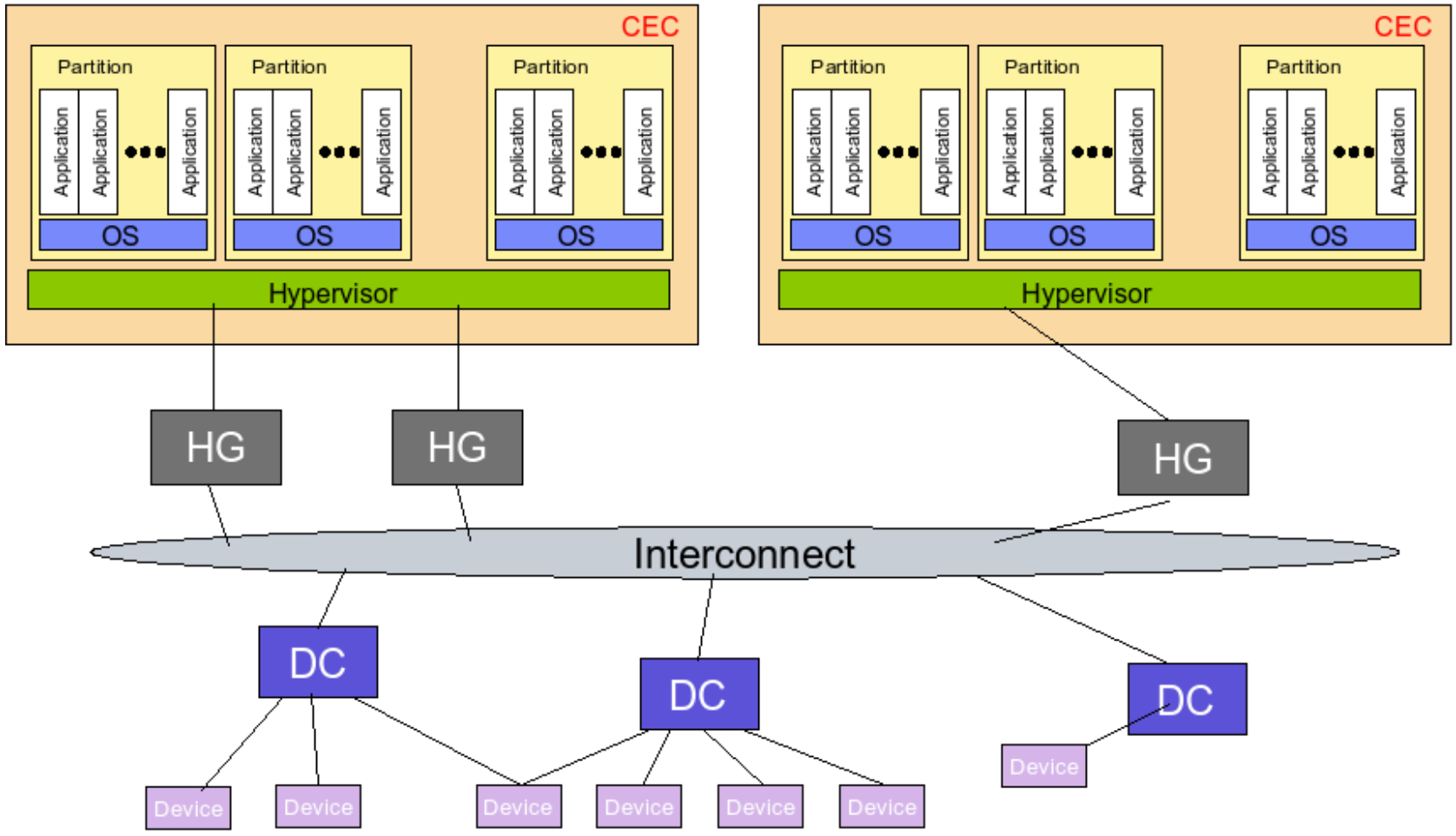
Architecture



I/O Consumers

- I/O consumers (clients) are applications or virtual machines.
- Access I/O devices through DCs.
- Use class libraries to access devices (have no knowledge of device specific bits).
- May be an application, an OS kernel, a virtual machine, or any other entity.
- Access the I/O fabric through a *Host Gateway*—directly. (OS and hypervisor bypass).

Architecture



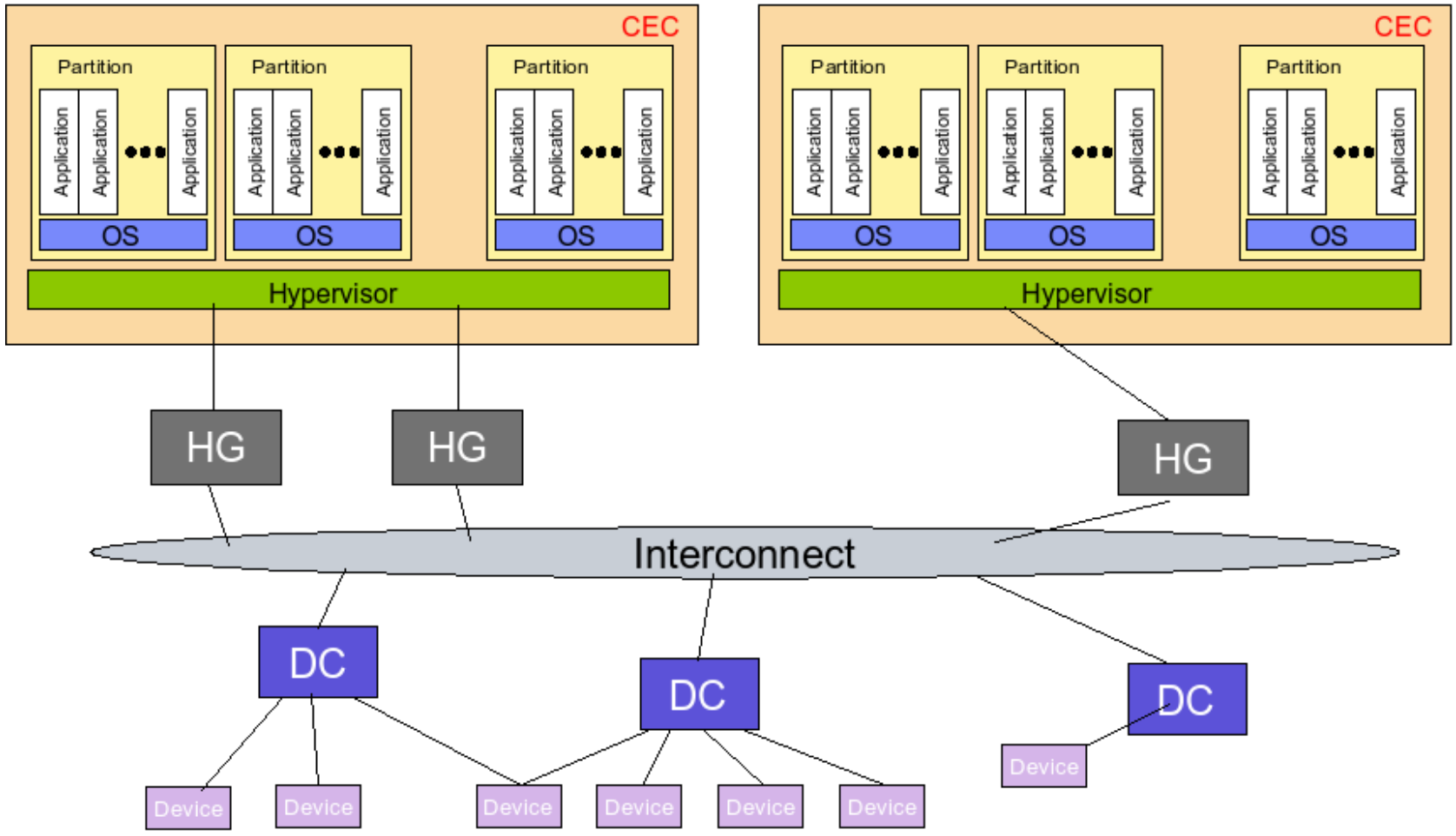
Host Gateway

- Think of it as an Infiniband HCA on steroids.
- Provides a protected I/O mechanism for all of the host's I/O consumers (i.e., protects consumers from rogue DCs).
- Or think of it as an elaborate DMA engine which provides DMA access to virtual memory.

Security Model

- Loosely based on the Object Disk (OSD) security model [Factor ’05].
- Credential/Capability (no local state required!)
- Both devices (DCs) and consumers are protected.

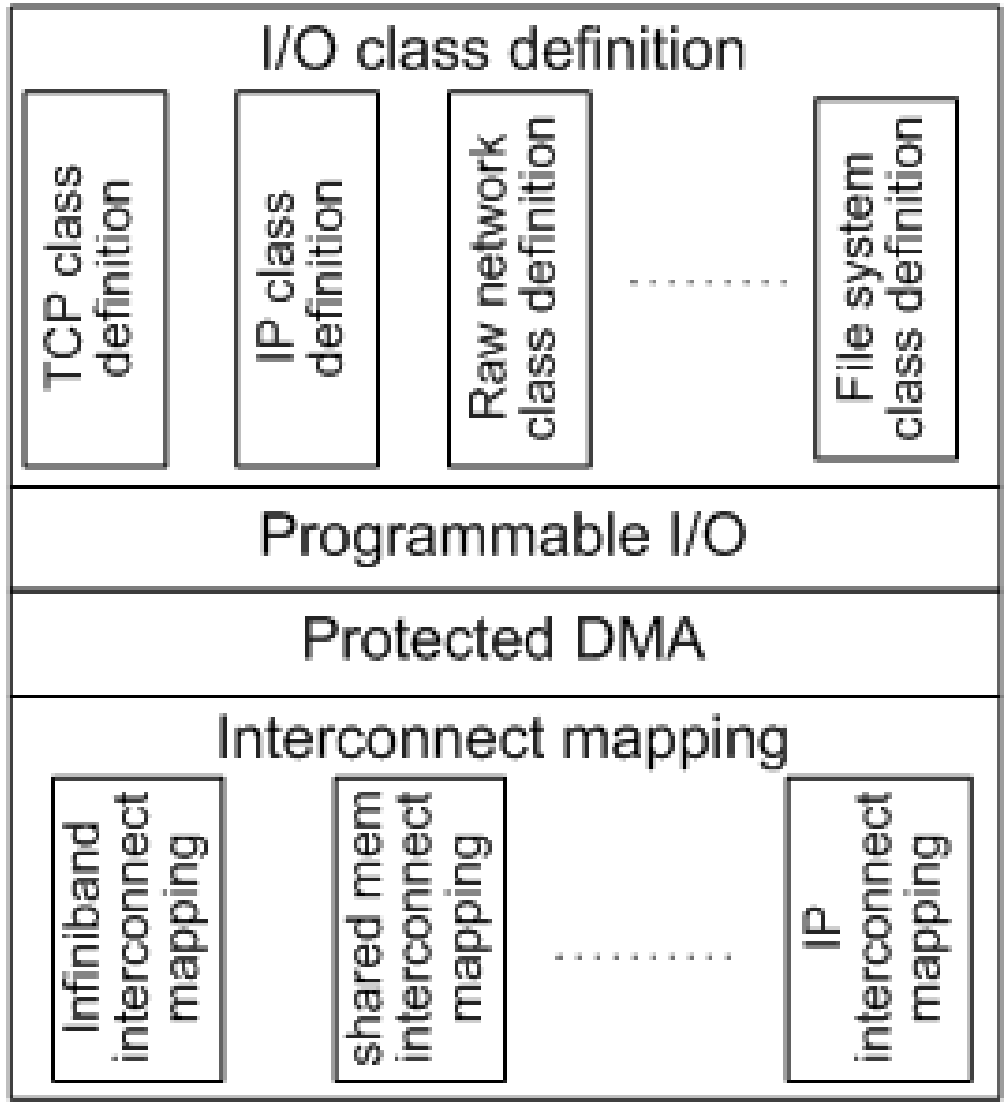
Architecture



Protected DMA

- DMA operations are secured by address protection code.
  - HG *generates* memory credential when I/O is sent from consumer to DC.
  - HG *validates* memory credential when DC tries to access consumer memory.
- PDMA allows remote DMAs over different interconnects.
- PDMA allows local DMAs between partitions.
- Consumer is oblivious to the I/O interconnect used.

SCIO Protocol

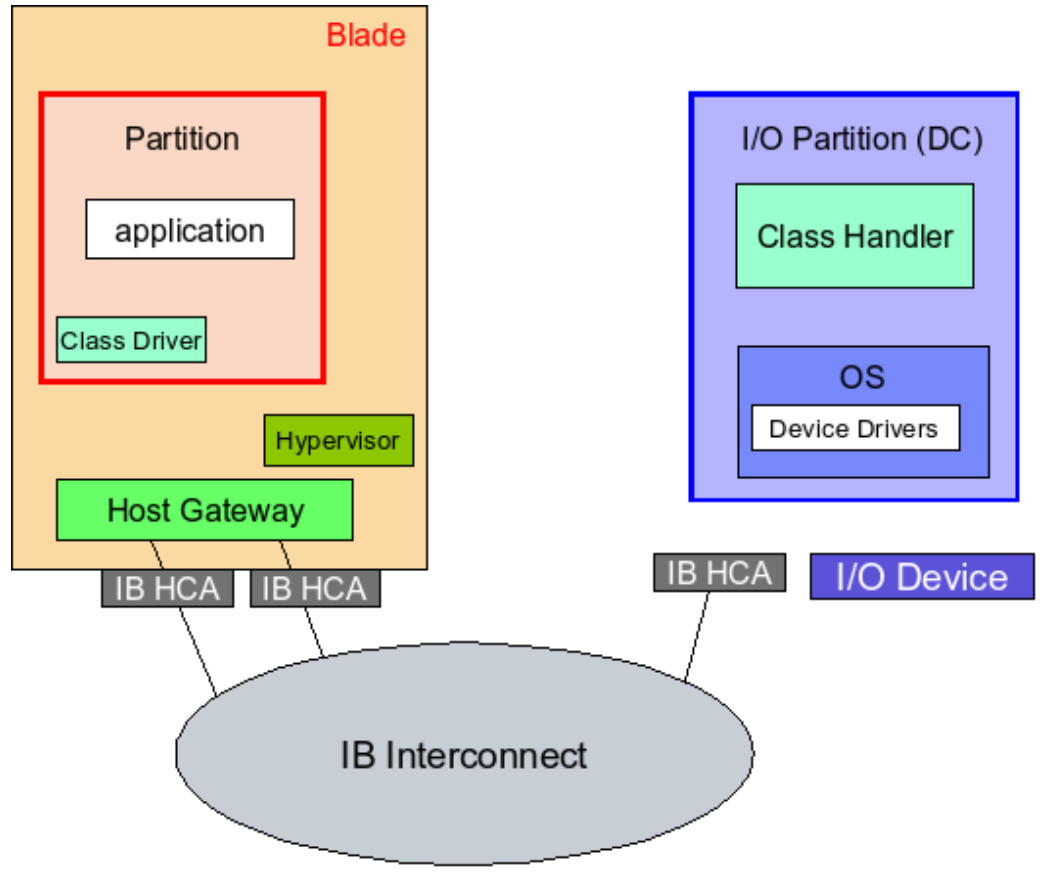


SCIO Protocol

- Consumers submit I/O programs to DCs (asynchronously).
- All programs share the same structure, but commands are class specific.
- PDMA is used by the programs to access host (consumer) memory.
- PDMA is transport independent: implemented over TCP, Infiniband, HiperSockets, shared memory.

Execution Flow

1. Application obtains device credentials.
2. Application invokes an I/O operation via class driver.
3. Class driver builds an I/O program; submits to DC.
4. HG generates memory credentials.
5. DC (class handler) executes I/O program.
6. DC accesses host memory through HG.
7. DC sends completion status message.



Key Differentiators

- A high-level interface.
- A *programmable* high-level interface.
- Scalability.
- Dynamic memory pinning.

High-level I/O Interface

- low-level: MMIO, PIO, IRQ, DMA.
- device-level: “send a packet”.
- SCIO high-level: “accept TCP connection”, “read file”.
- Advantages:
  - Reduced interaction improves performance (e.g., less protection-boundary crossings)
  - Isolate the I/O subsystem in a different, customized environment (reduces sharing, cache pollution, locking requirements, . . .)
  - TCP prototype achieved significant performance improvements [Shalev ’06].

Programmable I/O Interface

B 0	B 1	B 2	B 3	B 4	B 5	B 6	B 7

I/O Program Header

Class ID	Cmd Cnt	Cmd Len	Ext St Len
Sc Var Cnt	Buf Var Ler	Reserved	

I/O Program Commands

opcode+fl	Buf #	Length	
opcode+fl	Len	Imm Data	
Immediate Data (cont)			
opcode+f	Buf #	Length	
opcode+f	CondTag	MatchTag	Skip Cnt
opcode+f	Buf #	Length	

Programmable I/O Interface

- Compound operations a-la NFSv4.
- Basic I/O operations and conditions a-la mainframe Channel programs.
- Full-fledged programming environment: Java virtual machines, MapReduce?
- Benefits:
  - Reduced interaction overhead (bus crossings).
  - Reduced latency.
  - Execute where the data is.
  - Flexibility—avoid the SCSI “every command under the sun” phenomenon.

Scalability

- De-coupling of the compute and I/O functionality leads to better scalability.
- Off-loading of I/O functionality to dedicated environment.
- HG is not a bottleneck—there can be many HGs (almost completely stateless).

The Memory Pinning Problem

- Memory pinning is expensive:
  - Registration overhead (even if cached)
  - Memory consumption (must account for worst case)
  - Programmer burden
- Observation: an I/O device is just another core accessing memory. . .

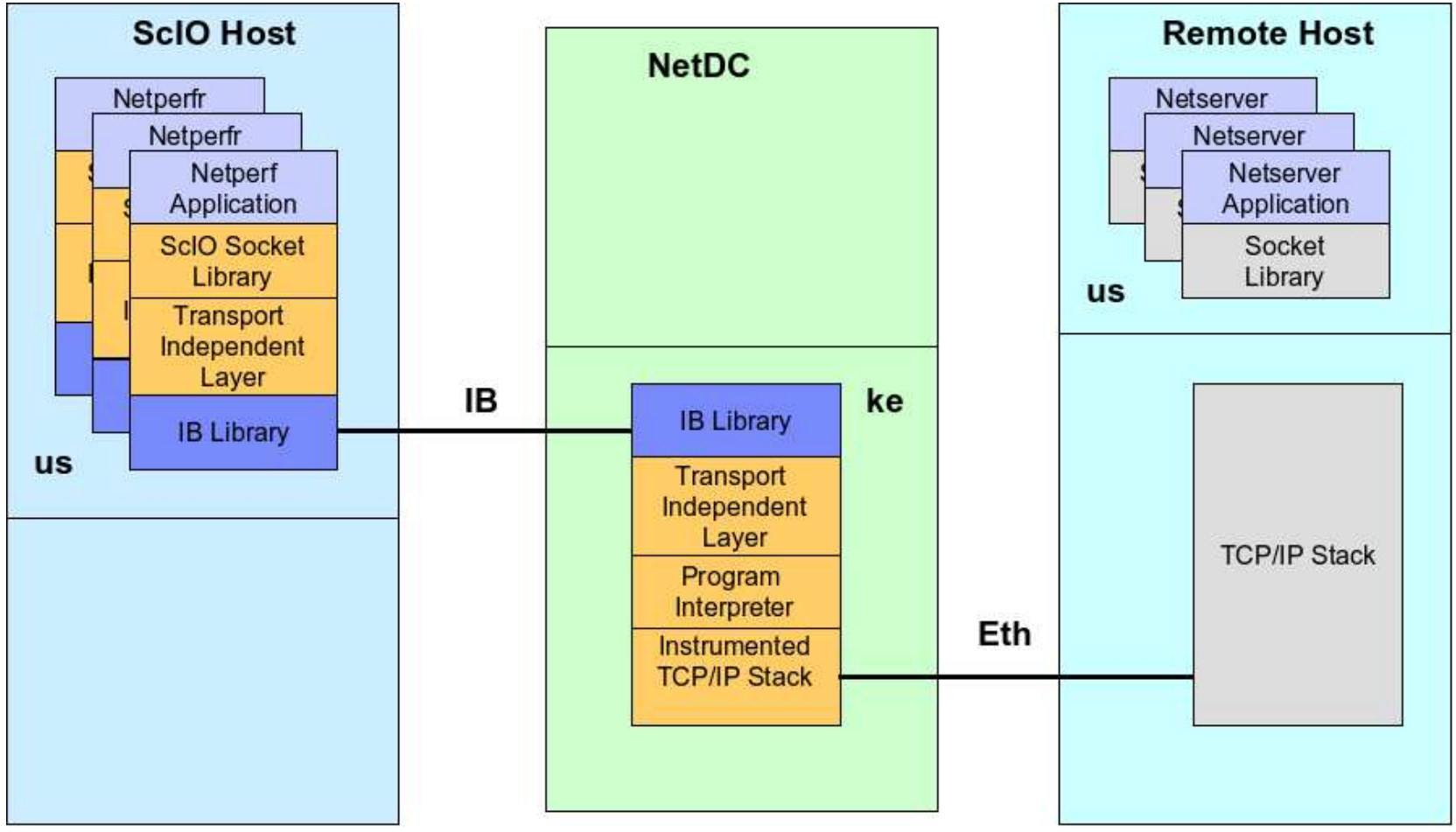
Dynamic Memory Pinning

- Key observation: memory buffers accessed by I/O devices are likely to have also been accessed by the CPU!
- Therefore, rather than requiring pinning buffers for I/O up-front, consider all I/O buffers likely to be already resident.
- Treat the case where something is not resident as an exception case.
- Unlike Infiniband, no need for registration or memory keys.
- Unlike PCI-SIG IOV ATS, DMA requests carry with them an *address space ID*, which is per consumer rather than per device.

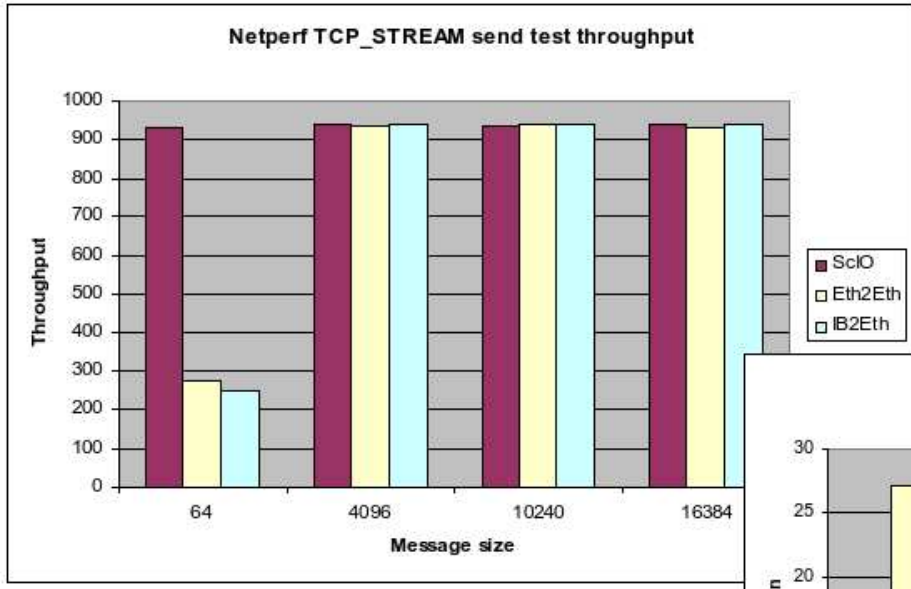
I/O Page Fault

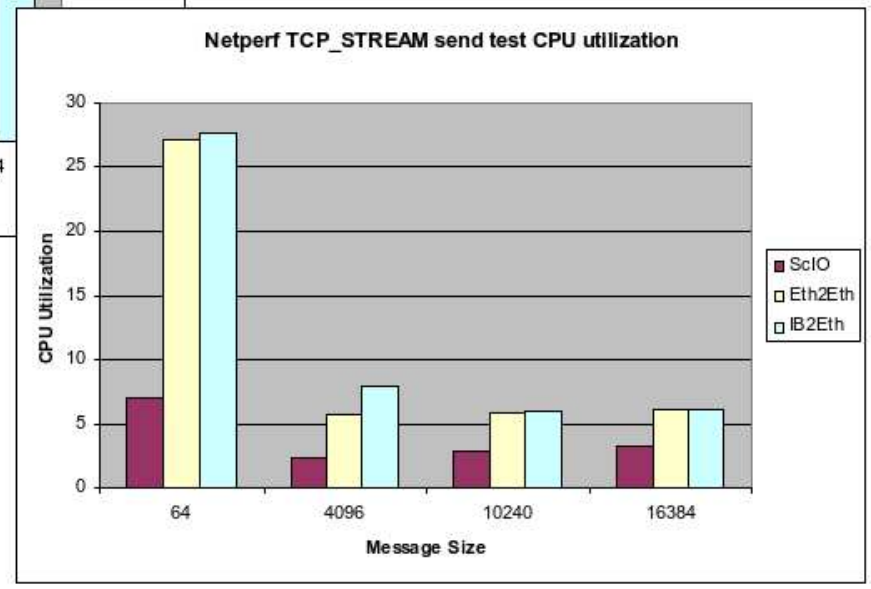
- A rare occurrence.
- Raised by the HG when no translation exists for a given (address space, virtual address) tuple.
- Handled by privileged software.
- HG and DC cooperate to avoid dropping data (e.g., due to lack of buffers space while page fault is resolved).

Prototype environment

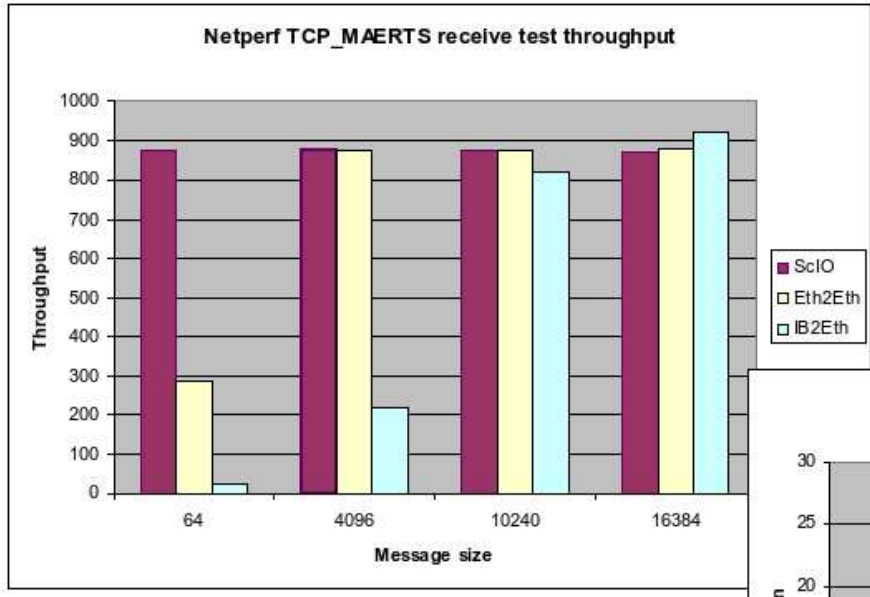


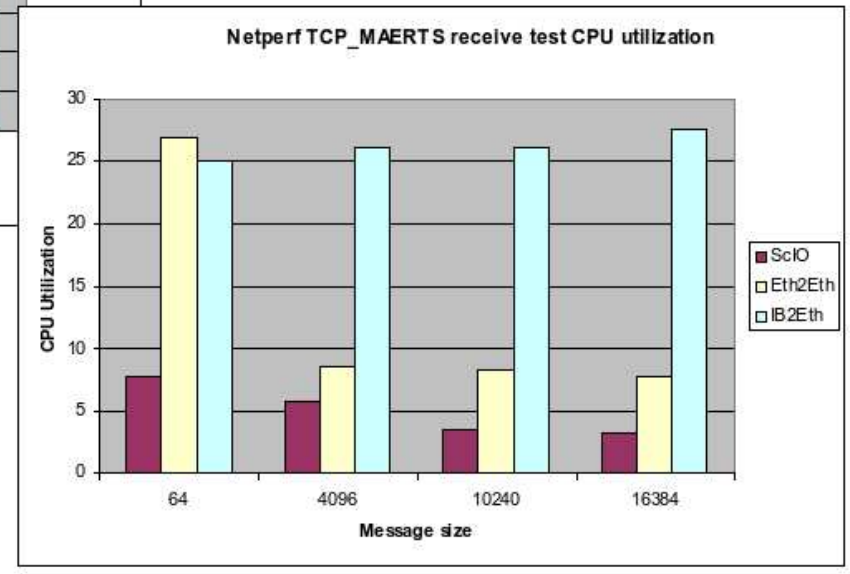
Performance: Send





Performance: Receive





Summary

- Scalable I/O simplifies I/O virtualization and improves I/O efficiency, security, performance and scalability.
- Scalable I/O gets rid of drivers, replacing them with class libraries and Host Gateways.
- Performance via subsystem isolation and a programmable high-level interfaces.
- Security via a credential/capability mechanism.
- Scalability via de-coupling of the compute and I/O functionality.
- Dynamic memory pinning is the future.

Acknowledgments

The authors would like to thank, in no particular order:

Ton Engbersen, Scott Guthridge, Orran Krieger, Vadim Makhervaks, Ilan Shimony, T. Basil Smith, John Tracey, Michael Waidner, Jimi Xenidis, Michal Ostrowski, Hubertus Franke, Ed Seminaro, Greg Still, Alan Benner, Tom Bradicich, and Micky Rodeh.

Questions?