Direct Device Assignment for Untrusted Fully-Virtualized Virtual Machines

Ben-Ami Yassour **Muli Ben-Yehuda** Orit Wasserman

benami@il.ibm.com muli@il.ibm.com oritw@il.ibm.com

IBM Haifa Research Lab

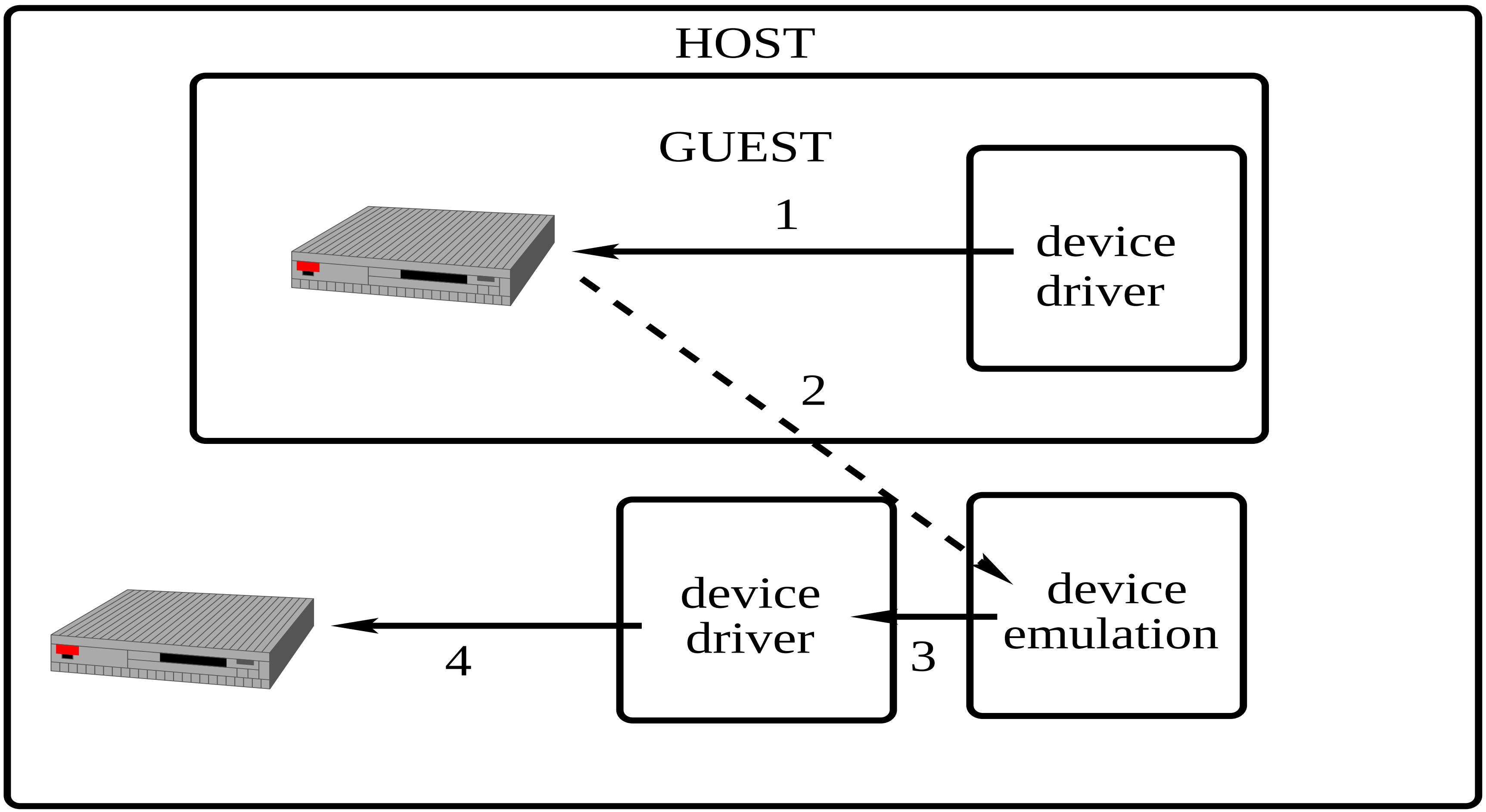
Table of Contents

x86 Virtualization

Virtual Machine I/O

- Virtual machines use three models for I/O:
  - Emulation
  - Para-virtualized drivers
  - Pass-through access

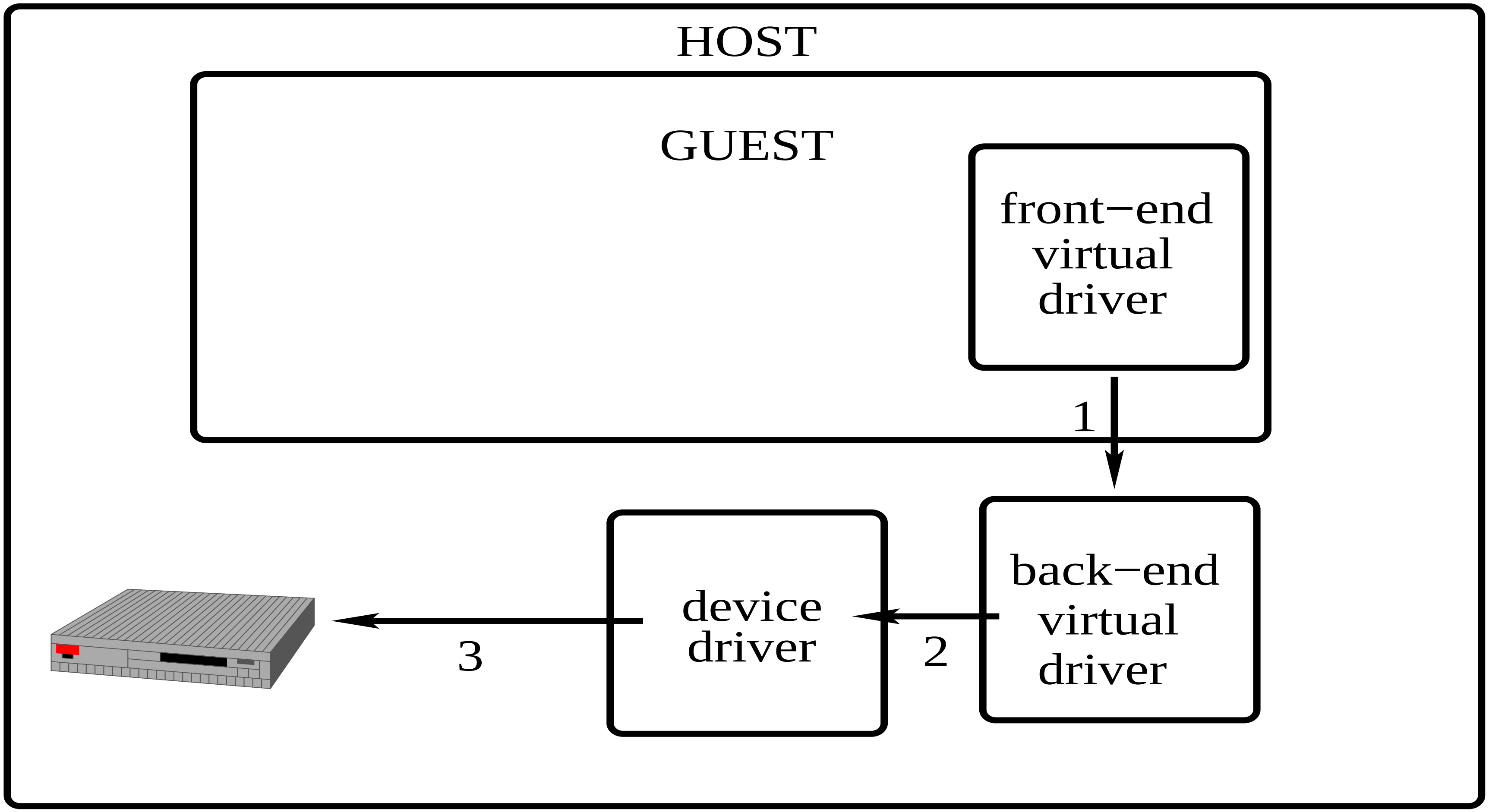
I/O: Emulation



I/O: Emulation cont'

- Hypervisor emulates real I/O devices [Sugerman01]
- Virtual machine uses its standard drivers
- Hypervisor traps device accesses (MMIO, PIO)
- Hypervisor emulates interrupts and DMA
- Interface limited to low-level, real device interface!
    - Which is not a good fit for software emulation
- $\rightarrow$ High compatibility but low performance.

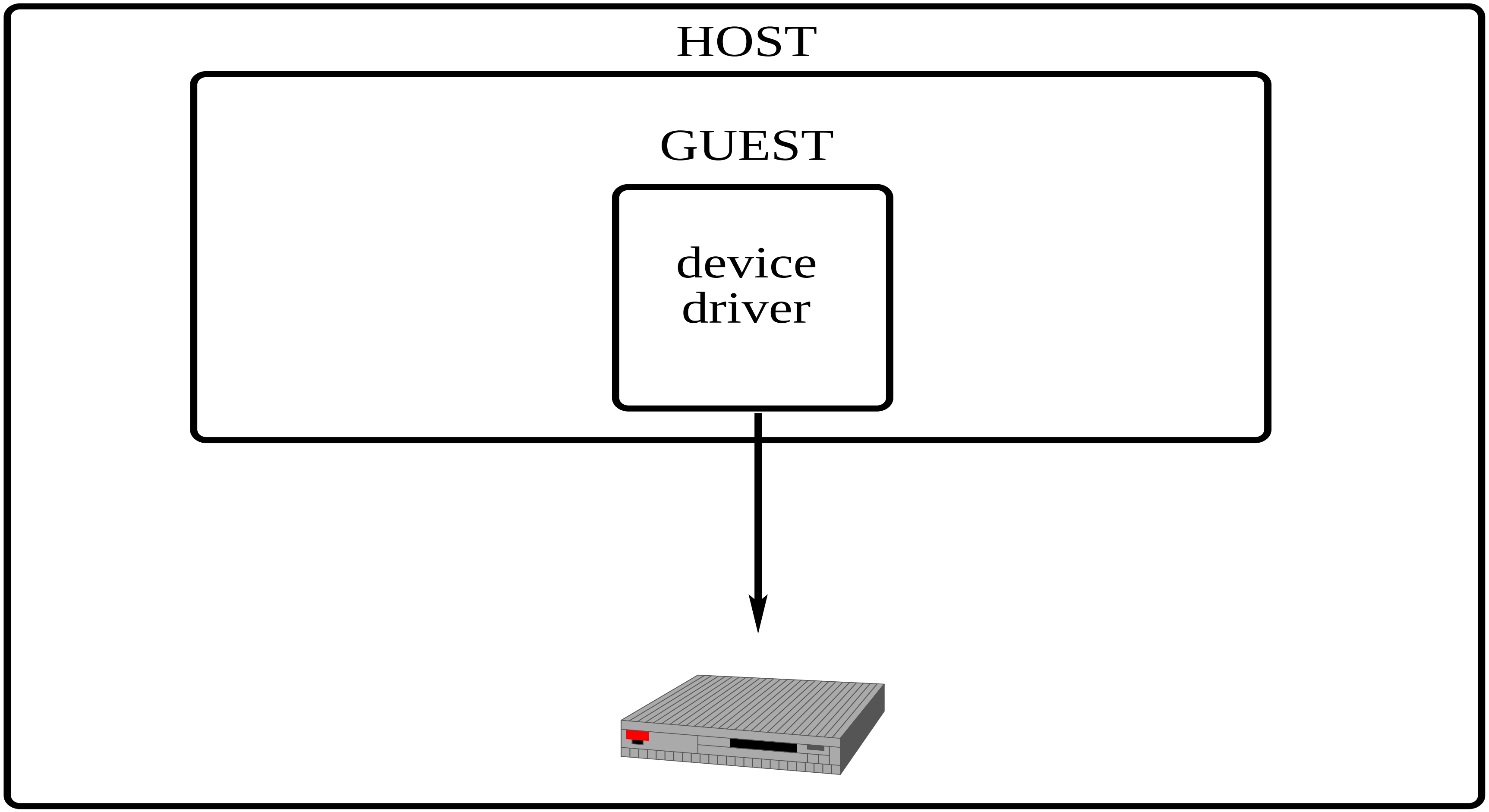
I/O: Para-virtualized Drivers



I/O: Para-virtualized Drivers cont’

- Hypervisor and VM cooperate for more efficient I/O [Barham03] [Russell08]
- Hypervisor specific drivers installed in the VM
- Network device level or higher up the stack
- $\rightarrow$ Low compatibility but better performance [Santos08].

I/O: Direct Device Assignment



I/O: Direct Device Assignment cont’

- Give VM direct access to a hardware device
- Without any software intermediaries between the virtual machine and the device
- Examples:
  - Legacy adapters [Ben-Yehuda06]
  - Self-virtualizing adapters [Liu06], [Willman07]
- $\rightarrow$ Best performance—but at a price.

I/O: Device Assignment Pros and Cons

- Pros
  - Best performance compared to other methods
  - Supporting odd-ball devices that don't have emulation support or equivalent PV drivers
  - Supporting self-virtualizing devices (SRIOV/MRIOV)
- Cons
  - Reduces the level of virtualization
  - Make harder to migrate a virtual machine
  - Legacy device can not be shared

The Linux/KVM Hypervisor

- A hypervisor extension for the Linux kernel [Kivity07]
- Makes extensive use of Intel and AMD hardware virtualization extensions
- Full featured, open source, and hacker friendly
- `http://www.linux-kvm.org`

Direct Access Challenges

- PIO and MMIO
- Interrupts
- DMA—Security and Address Translation

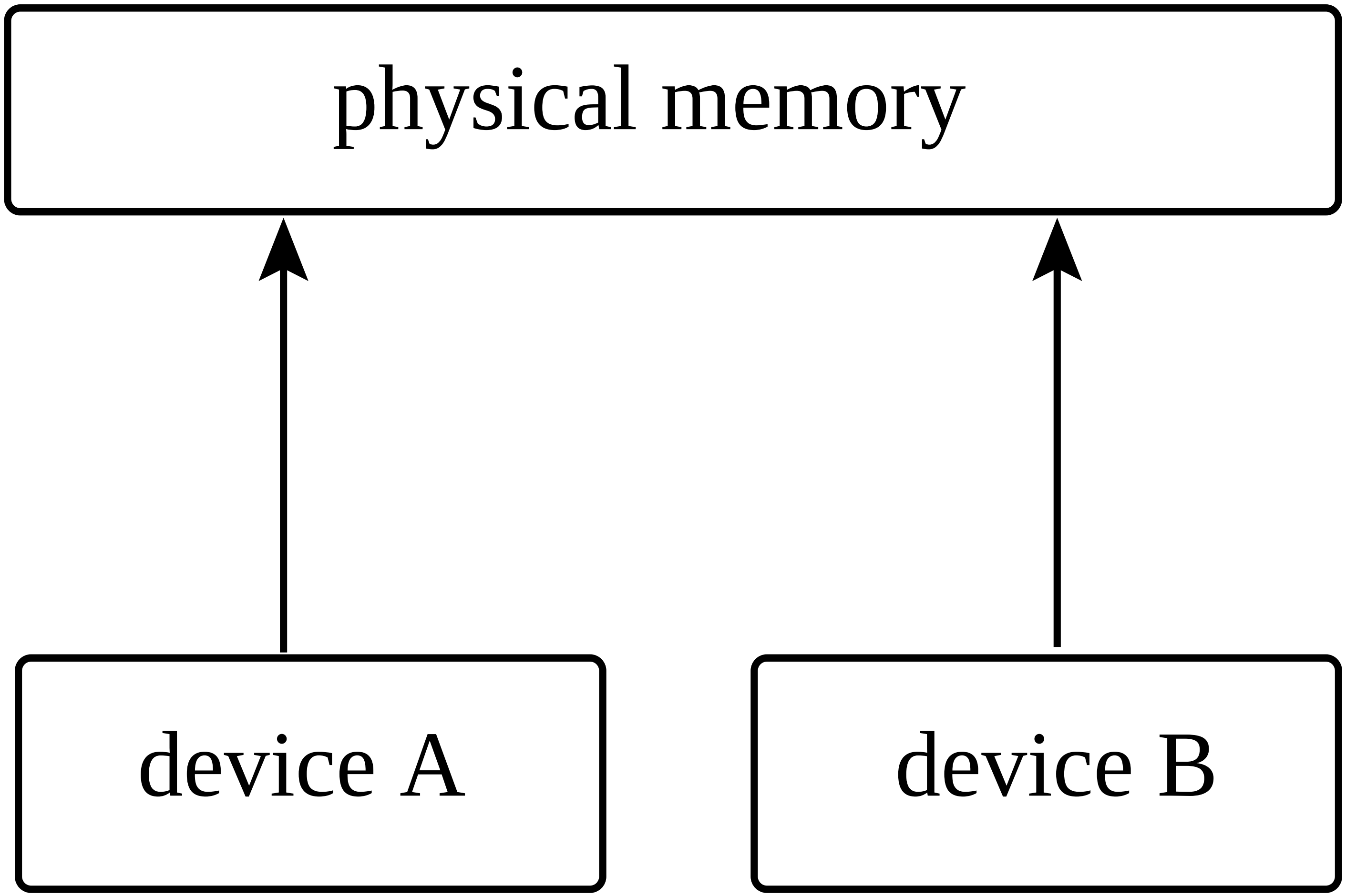
PIO and MMIO

- PIO/MMIO can be trapped by hypervisor and replayed to the device
- PIO can be passed directly via VMCS I/O bitmaps
- MMIO can be passed directly via mapping device BARs to guest
- Some PIO/MMIO accesses must be trapped (PCI config space)
- Direct-MMIO gives a nice performance improvement

Interrupts

- Host registers a direct access interrupt handler for IRQ
- Interrupt received $\rightarrow$ disable IRQ line
- Host injects interrupt to the guest
- Guest acks virtual APIC
- Host enables IRQ line
- Currently, no shared interrupts support
- MSI also supported

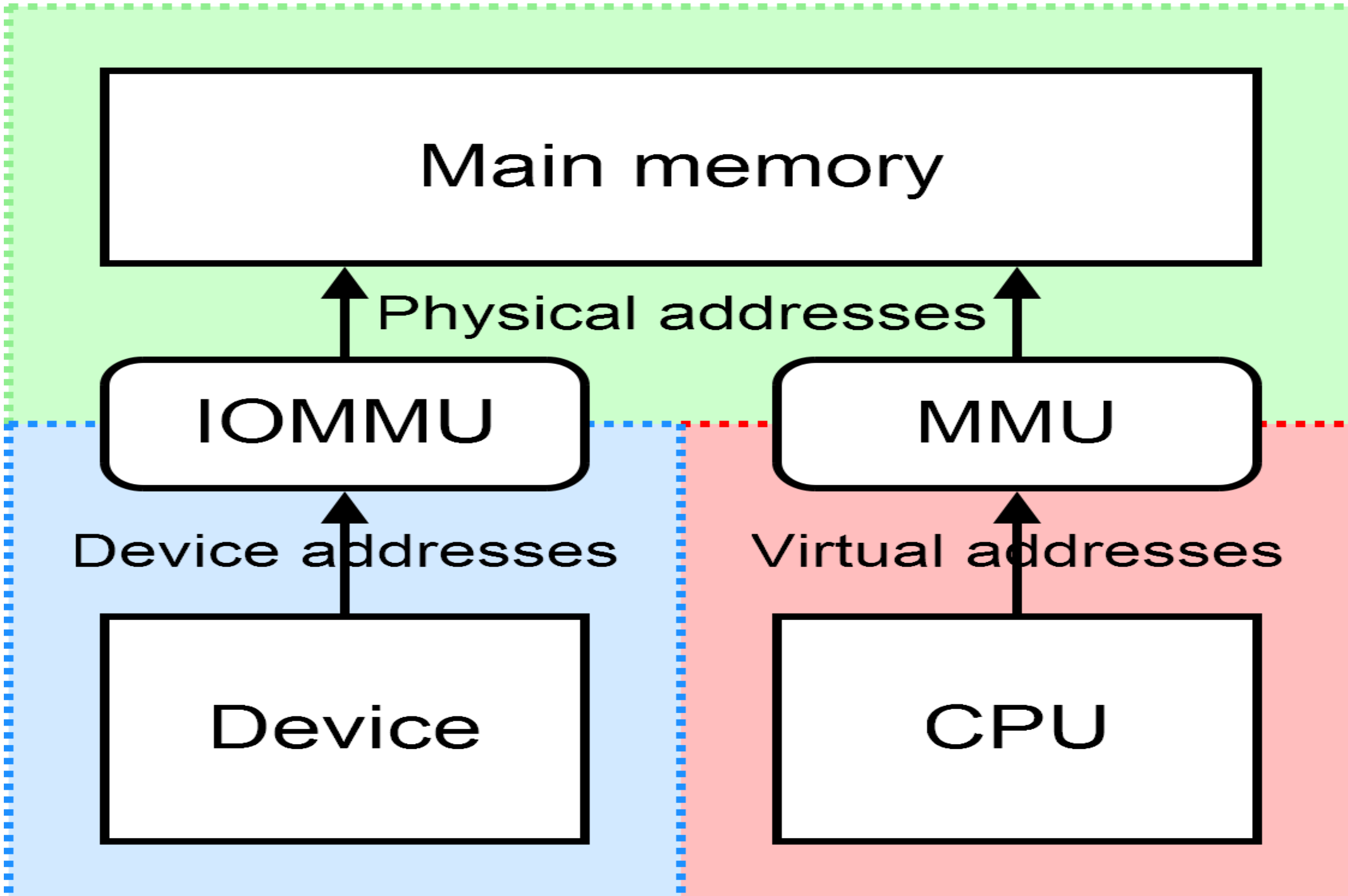
DMA



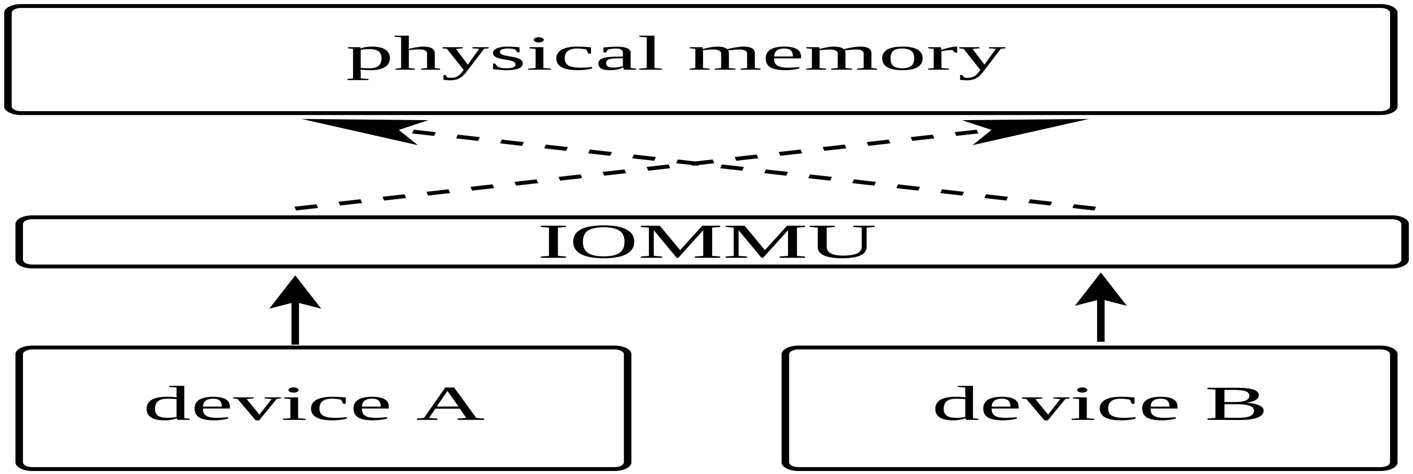
DMA Security

- Untrusted guest programs a device, without any supervision.
- Device is DMA capable (all modern devices are).
  - Which means the guest can program the device to overwrite any memory location.
- . . . including where the hypervisor lives . . . game over.

IOMMU


Main memory
Physical addresses
IOMMU
MMU
Device addresses
Virtual addresses
Device
CPU

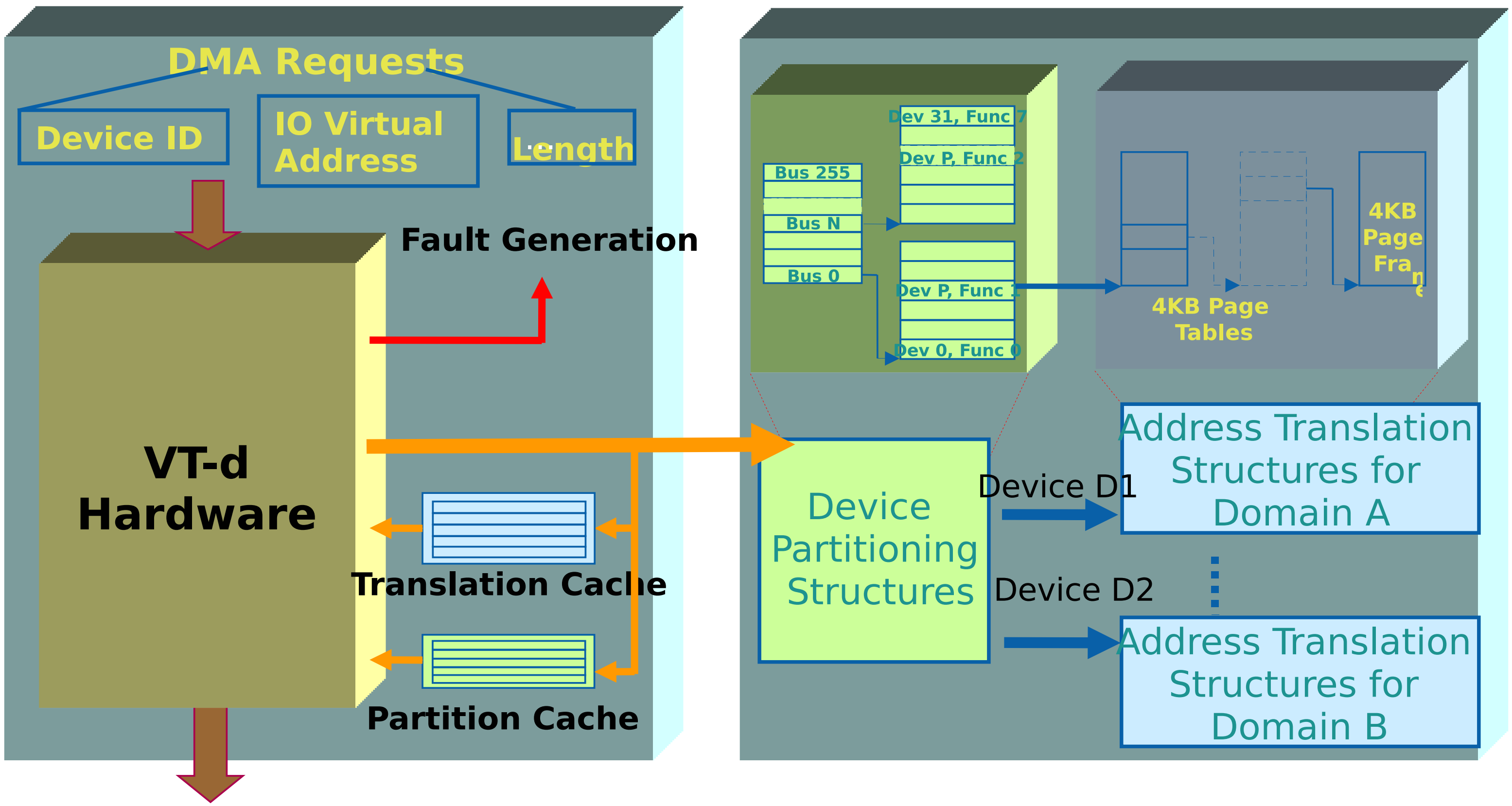
IOMMU to the rescue



- IOMMU—think MMU for I/O devices—separate address spaces, protection from malicious devices!
- IOMMUs enable direct assignment for VMs.
- Intra-VM vs. Inter-VM protection [Willman08]
- But: IOMMUs have costs too [Ben-Yehuda07]

The Intel VT-d IOMMU

VT-d Hardware Overview



©IBM Corporation 2009

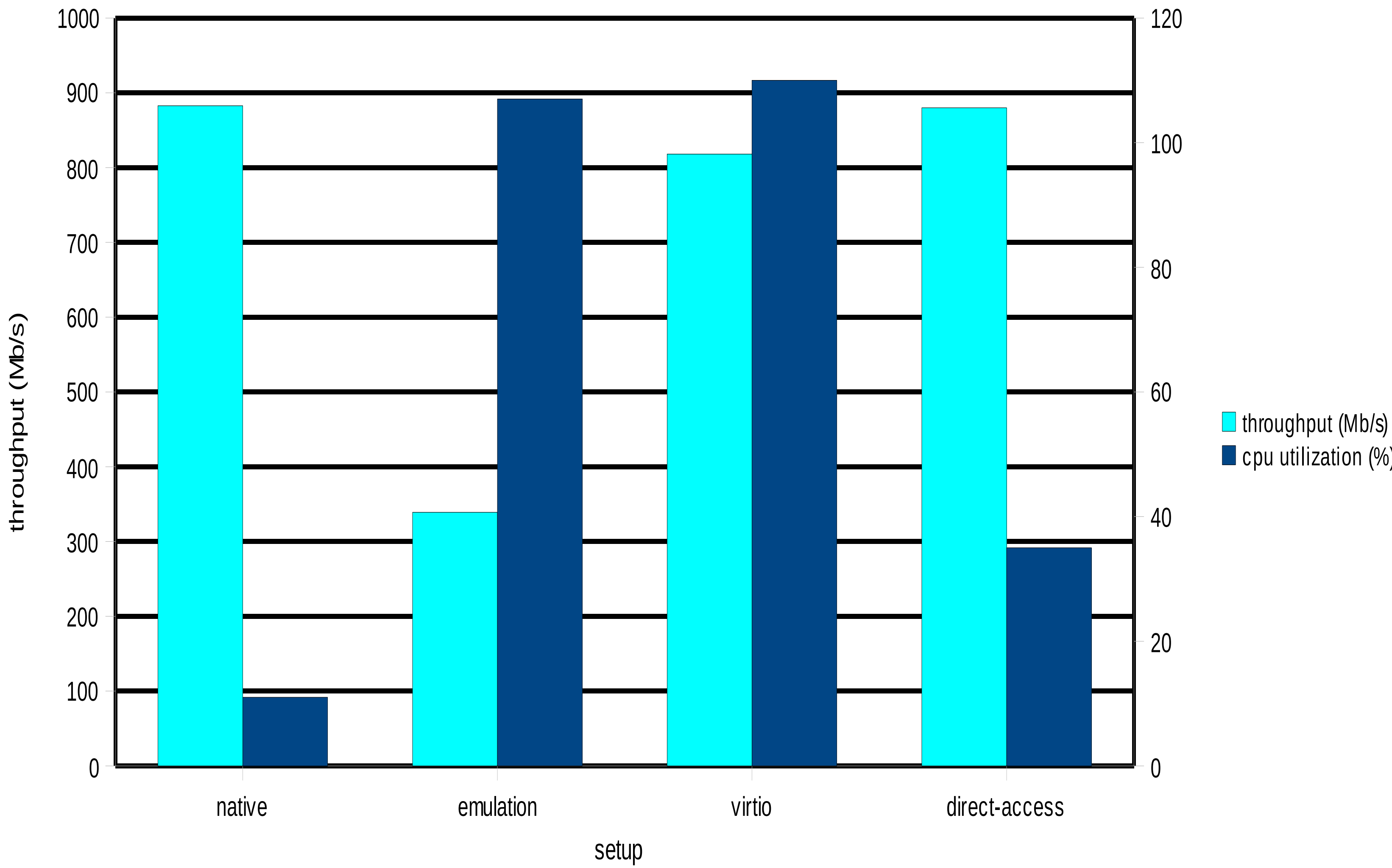
IOMMU Protection Strategies

As defined by Willman, Rixner and Cox [Willman08]:

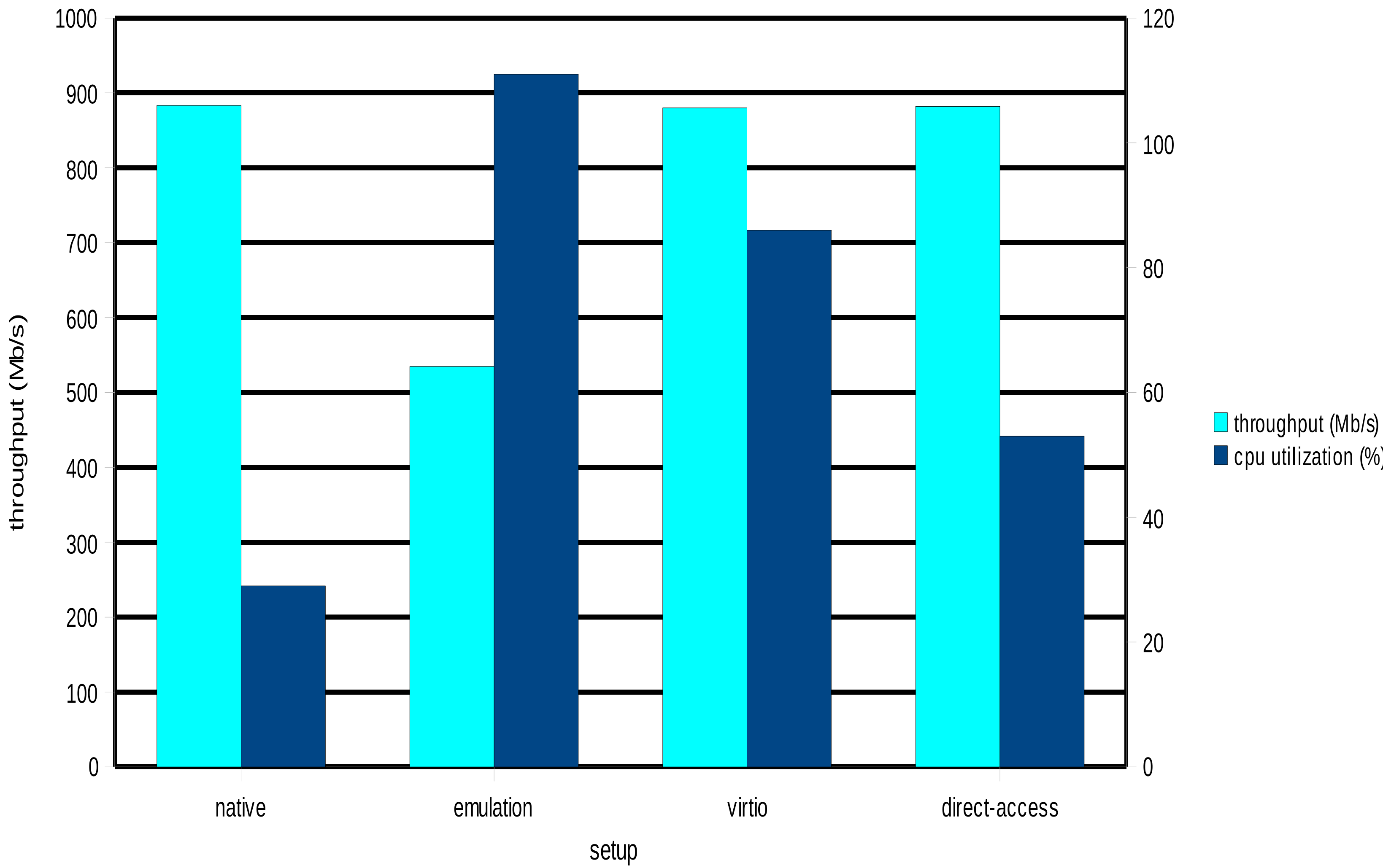
- Single-use → **Intra**-guest protection, expensive!
- Shared → Relaxed protection, expensive.
- Persistent → Inter-guest protection, pins all of memory.
- Direct-map → Inter-guest protection, no run-time cost, pins all of memory.

Our initial direct-access implementation (which is included in KVM today) used direct-mapping.

Direct-map Performance—Send



Direct-map Performance—Receive



IOMMU Protection Strategies Revisited

Single-mapping is very expensive, but pinning all of the guest's memory (no over-commit) is not acceptable. How can we balance performance and memory requirements?

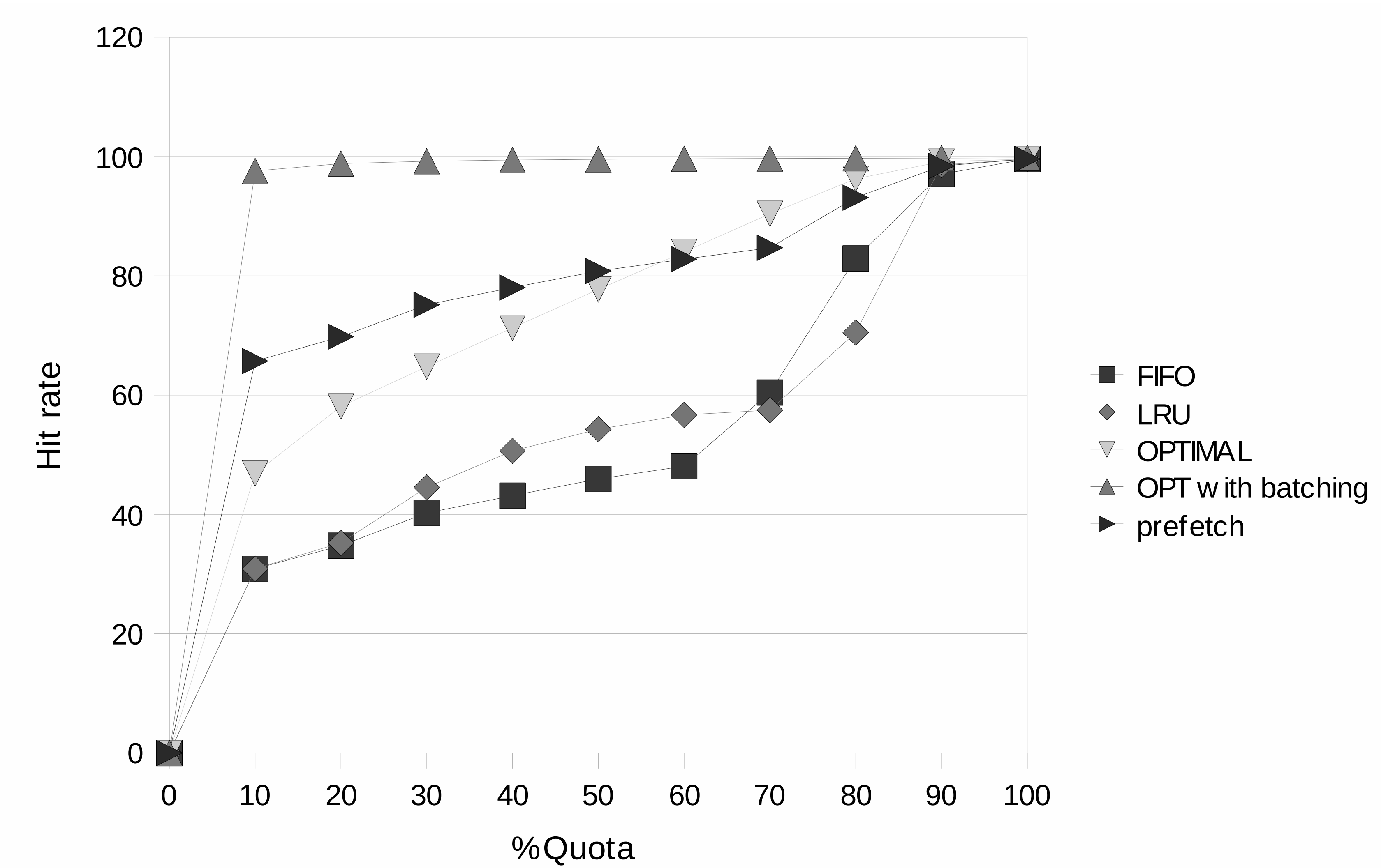
On-Demand Mapping Strategy

- IOMMU remappings are expensive (world switch, IOTLB flush)
- Solution: implemented a **map-cache** for caching IOMMU mappings. How big should it be?
- Observation: all guests have some memory pinned anyway.
- Second observation: common workloads do **not** need to use all of the guest's memory address space.
- Solution: defined a **quota** for map-cache: the amount of memory the guest can pin for DMA.
- Cooperative guests: defining a quota that is equal to their current memory requirements leads to no run-time IOMMU remappings—best performance!

On-Demand Mapping Strategy cont'

- Un-cooperative guests: smaller quota, hypervisor enforced.
- Now the question becomes: for a given quota that is smaller than the working set size, how to efficiently replace IOMMU mappings?
- Close resemblance to the classical page replacement problem.
- ... except I/O devices **do not** have page faults.
- Solution: batch map/unmap requests.
- Solution: prefetching of mappings (predict access patterns).

On-Demand Mapping Performance



Summary & Conclusions

- Direct device assignment gives best performance of all I/O virtualization methods [Yassour08].
- ... but also poses new problems.
- In particular, how to balance DMA mapping memory consumption and performance?
- ... via the on-demand mapping strategy (paper in preparation).
- Want to hear more?
- ... join us at the 2nd Workshop on I/O Virtualization!

Bibliography

- Barham03: “Xen and the Art of Virtualization”, SOSP ’03
- Bellard05: “QEMU, a Fast and Portable Dynamic Translator”, USENIX ’05
- Ben-Yehuda06: “Utilizing IOMMUs for Virtualization in Linux and Xen”, OLS ’06
- Ben-Yehuda07: “The Price of Safety: Evaluating IOMMU Performance”, OLS ’07
- Bhargave08: ”Accelerating two-dimensional page walks for virtualized systems”, ASPLOS ’08

Bibliography cont.

- Chen08: “Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems”, ASPLOS ’08
- Liu06: “High Performance VMM-Bypass I/O in Virtual Machines”, USENIX ’06
- Kivity07: “kvm: The Kernel-Based Virtual Machine for Linux”, OLS ’07
- Popek74: “Formal Requirements for Virtualizable Third Generation Architectures”, CACM 17(7), ’74
- Russell08: “virtio: Towards a De-factor Standard for Virtual I/O Devices”, OSR 42(6), ’08

Bibliography cont.

- Santos08: “Bridging the Gap between SW & HW Techniques for I/O Virtualization”, USENIX ’08
- Sugerman01: “Virtualizing I/O Devices on VMware Workstation’s Hosted Virtual Machine Monitor”, USENIX ’01
- Willman07: “Concurrent Direct Network Access for Virtual Machine Monitors”, HPCA ’07
- Willman08: “Protection Strategies for Direct Access to Virtualized I/O Devices”, USENIX ’08
- Yassour08: “Direct Device Assignment for Untrusted Fully-Virtualized Virtual Machines”, Ben-Ami Yassour, Muli Ben-Yehuda, Orit Wasserman, IBM Research Report H-0263, 2008