Virtual Virtual Memory

Jason Power

3/20/2015

With contributions from Jayneel Gandhi and Lena Olson

Virtual Machine History

- 1970's: VMMs
- 1997: Disco
- 1999: VMWare (binary translation)
- 2003: Xen (para-virtualization)
- 2006(ish): Hardware support

VM Origins

1974

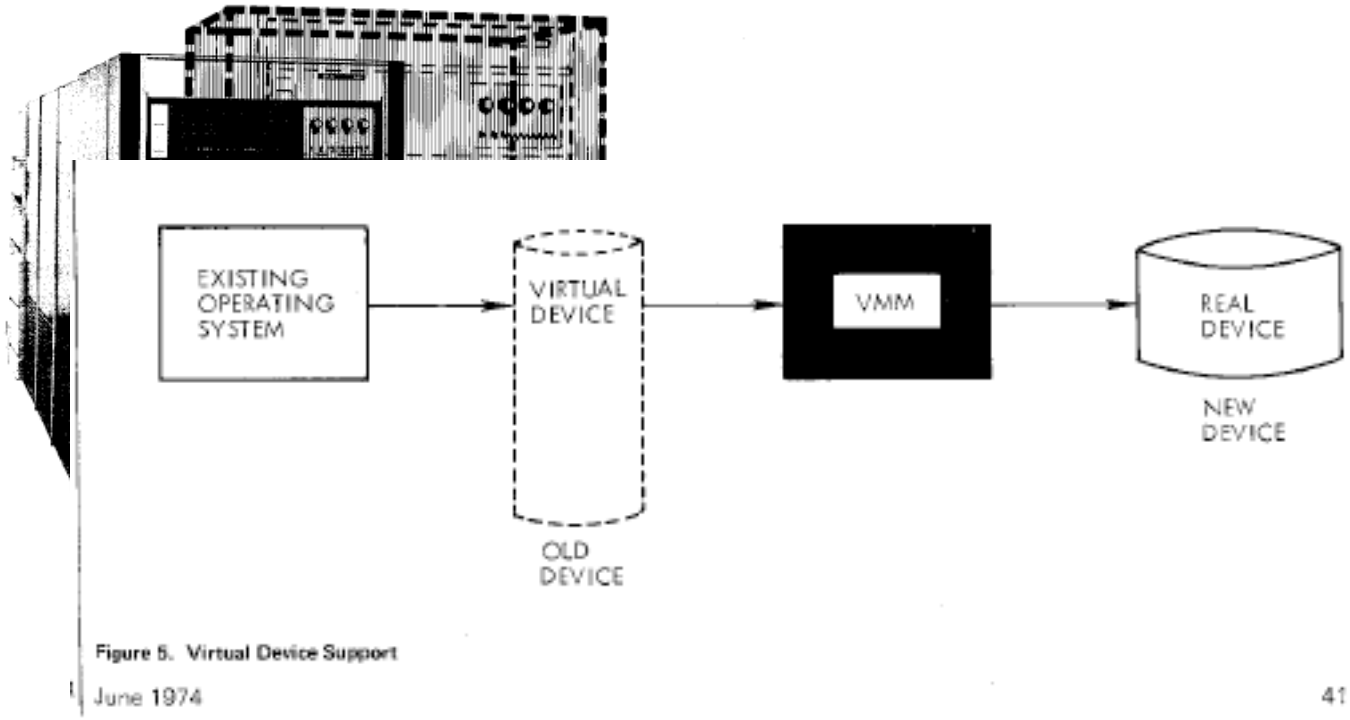


Figure 6. Virtual Device Support

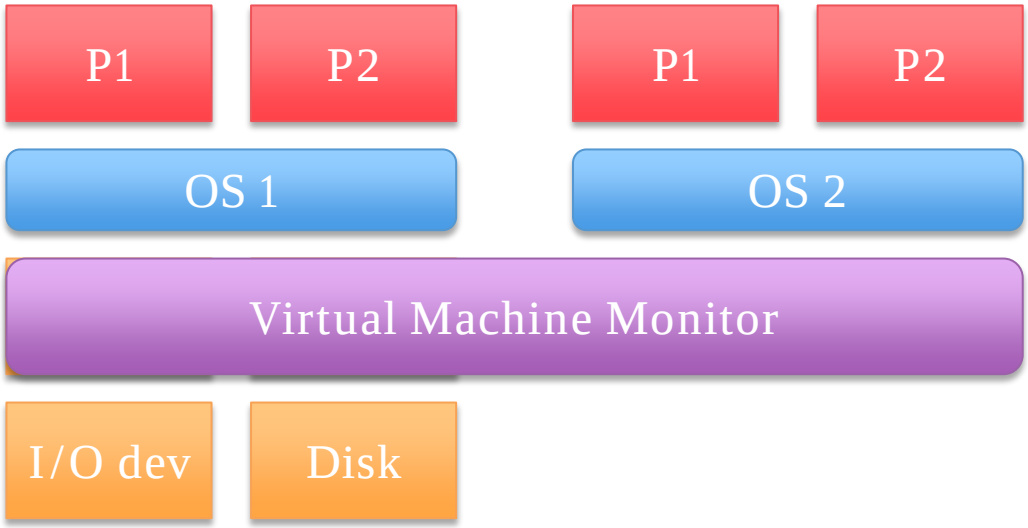
June 1974 41

Survey of Virtual Machine Research

Robert P. Goldberg

Honeywell Information Systems
and Harvard University

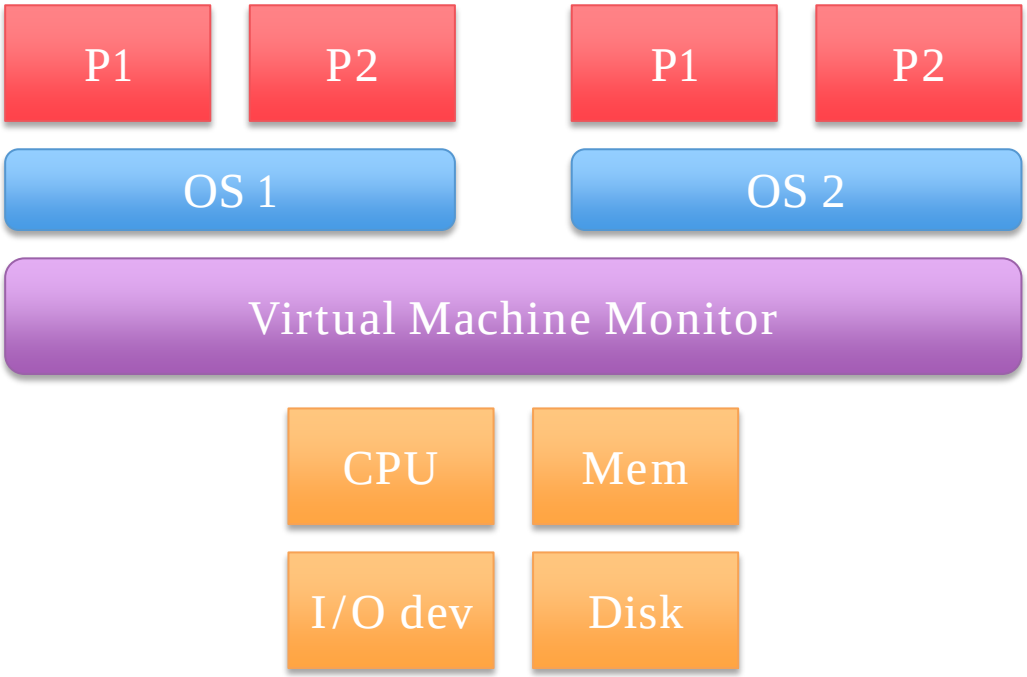
Virtual Machine Monitor (VMM)



VMM also called a hypervisor

Disco

- Trap and emulate



What about x86?

- x86 can't use trap and emulate
- Classic Example: ***popf*** instruction
  - Same instruction behaves differently depending on execution mode
  - User Mode: changes ALU flags
  - Kernel Mode: changes ALU **and** system flags
  - **Does not generate a trap in user mode**

VMWare

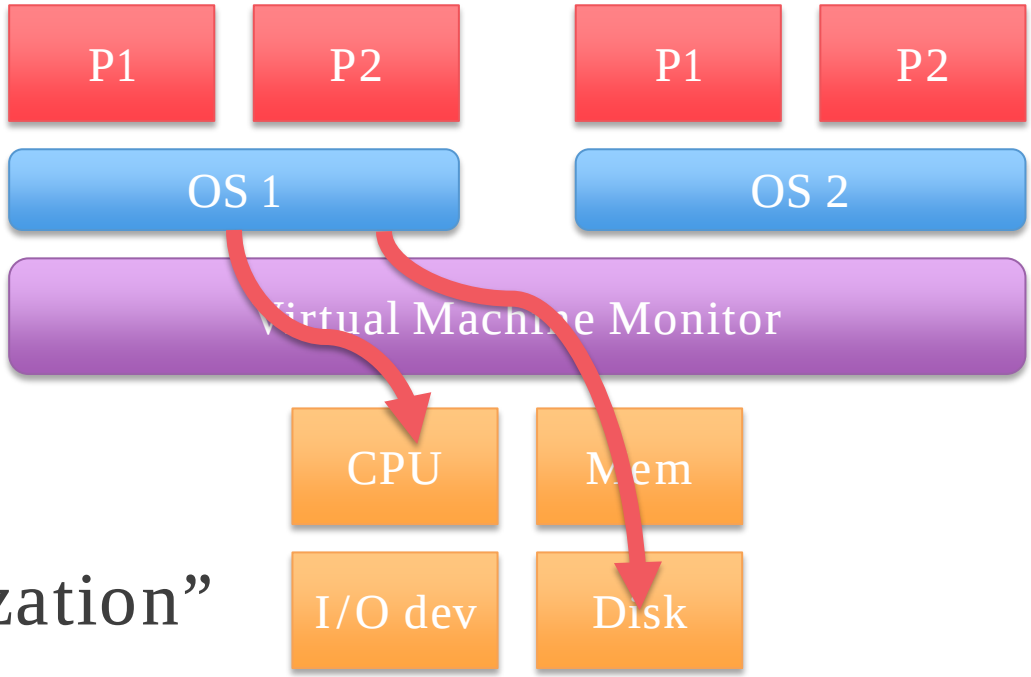
- **Solution:** binary translation
- Only need to translate OS code
  - Makes SPEC run fast by default
- Most instruction sequences don't change
- Instructions that **do change:**
  - Indirect control flow: call/ret, jmp
  - PC-relative addressing
  - Privileged instructions

Overheads

- Traps are heavy weight
- Binary translation
  - Bad for OS-heavy workloads (many server apps)
- What if you're allowed to change OS a little?

Paravirtualization and Xen

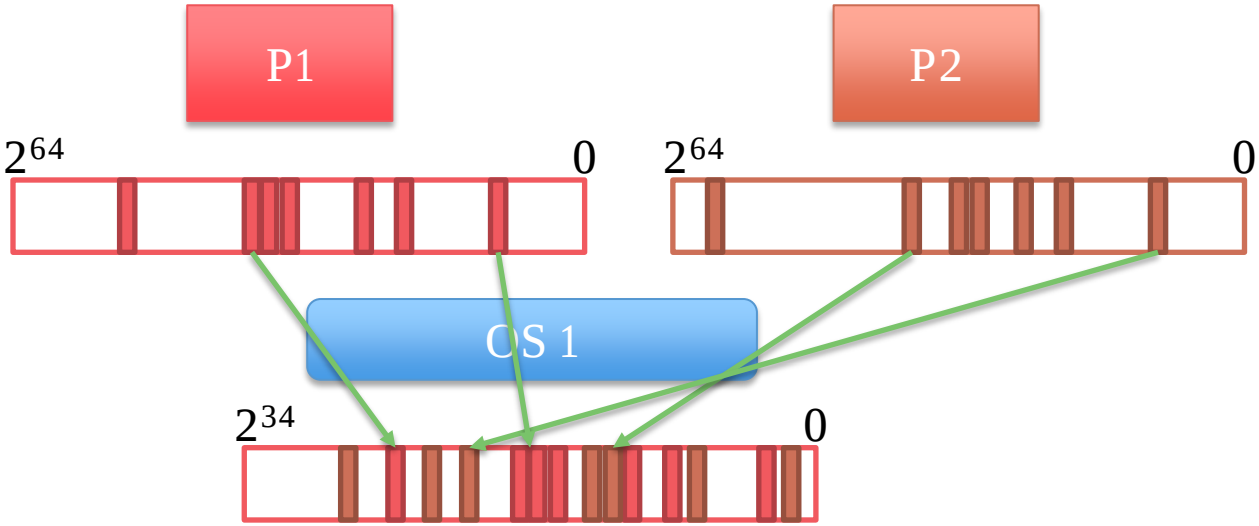
- Use ***hypercalls*** to bypass VMM
- Still emulate for corner cases & safety reasons
- Commonly used!
  - Amazon EC2
- Not “full virtualization”



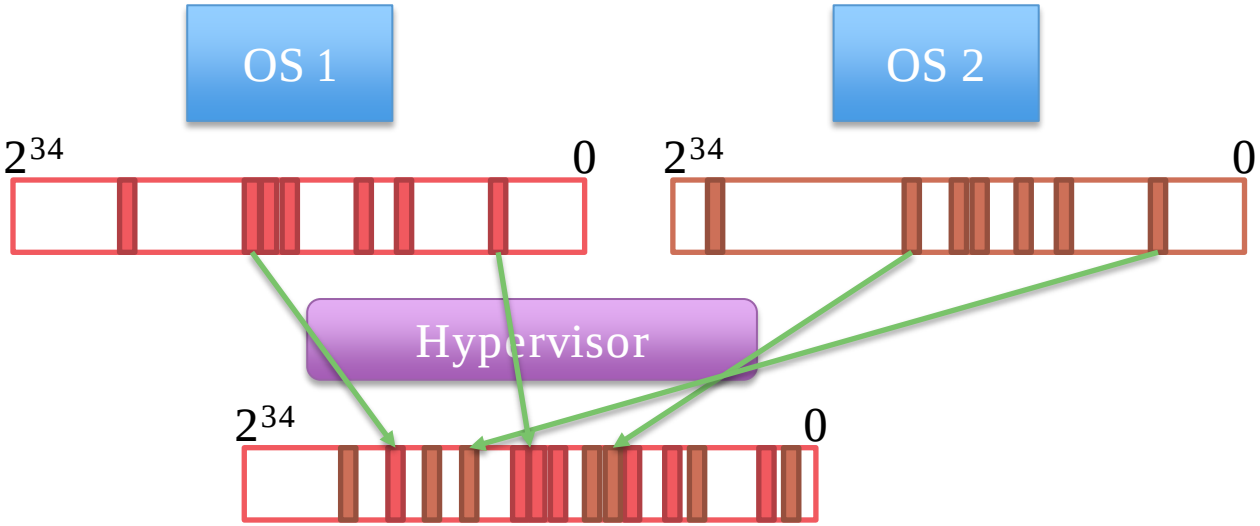
Hardware Support

- Another ring
  - `int` moves from user-mode to kernel-mode
  - `vmrun` moves from kernel-mode to vmm-mode
- Many other instructions
- What about virtual memory?

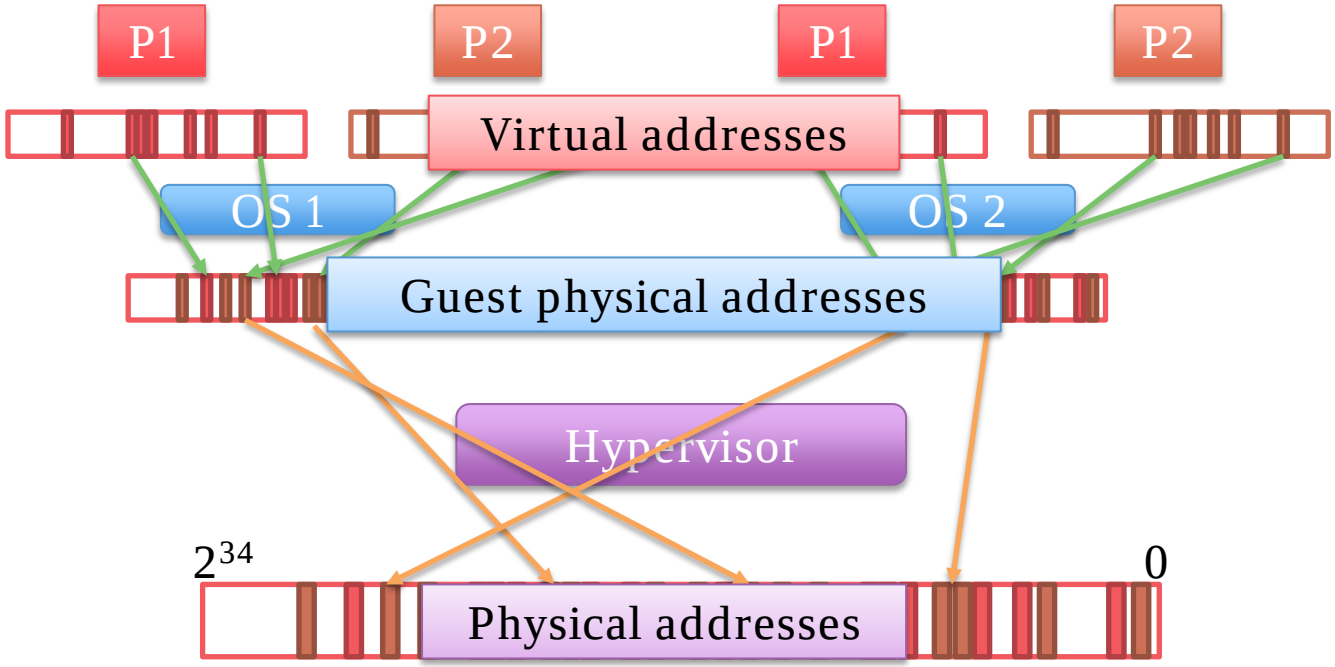
Let's recall virtual memory



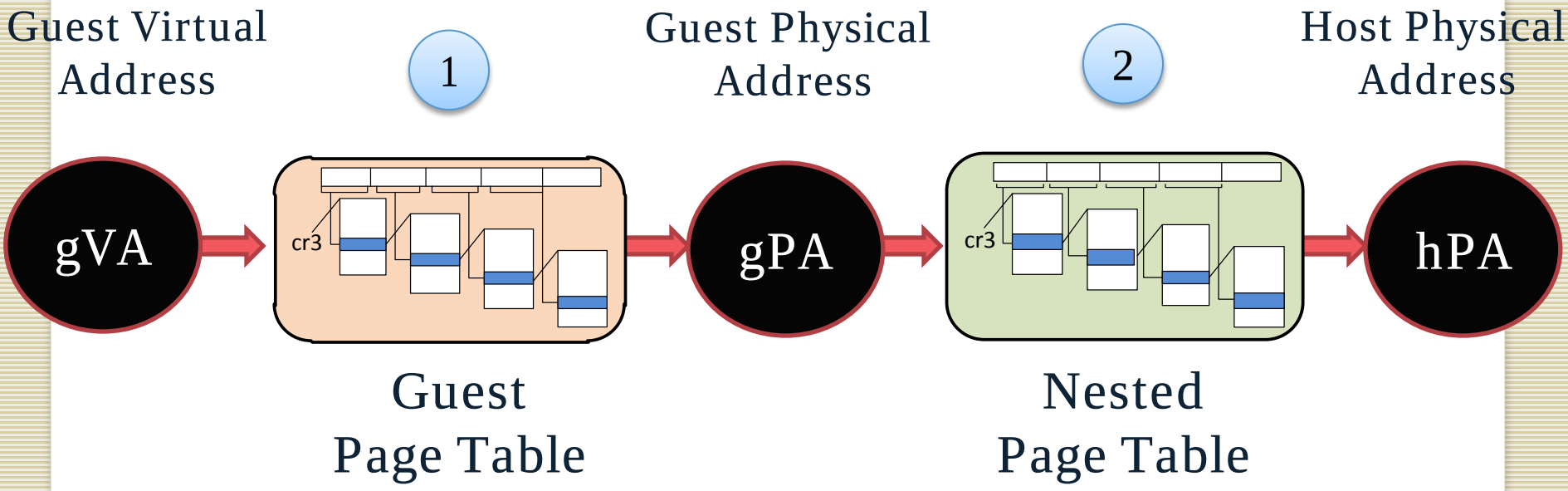
Virtualize the OS's memory?



Two *dimensions* of translation



Two *dimensions* of translation



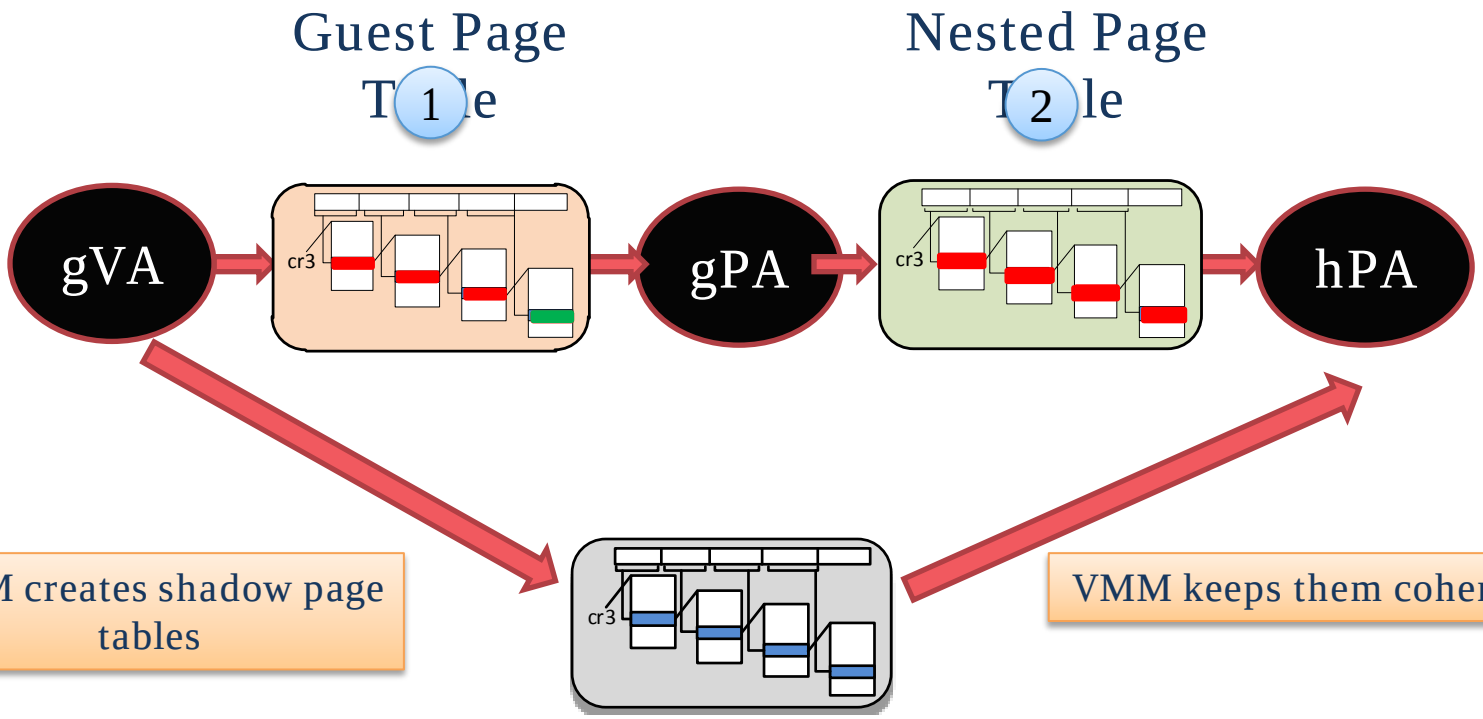
What about the TLB?

- Want to cache virtual → machine in TLB
- (Relatively) Easy with software-loaded TLBs
  - TLB miss is a trap (virtual → guest physical)
  - Guest OS loads TLB (VMM trap)
  - Translates guest physical → machine physical
  - VMM actually does the TLB insert
- Problem?

Hardware walked pagetable

- Page table walker walks nested pagetable
- Need a “fake” page table

Shadow Paging

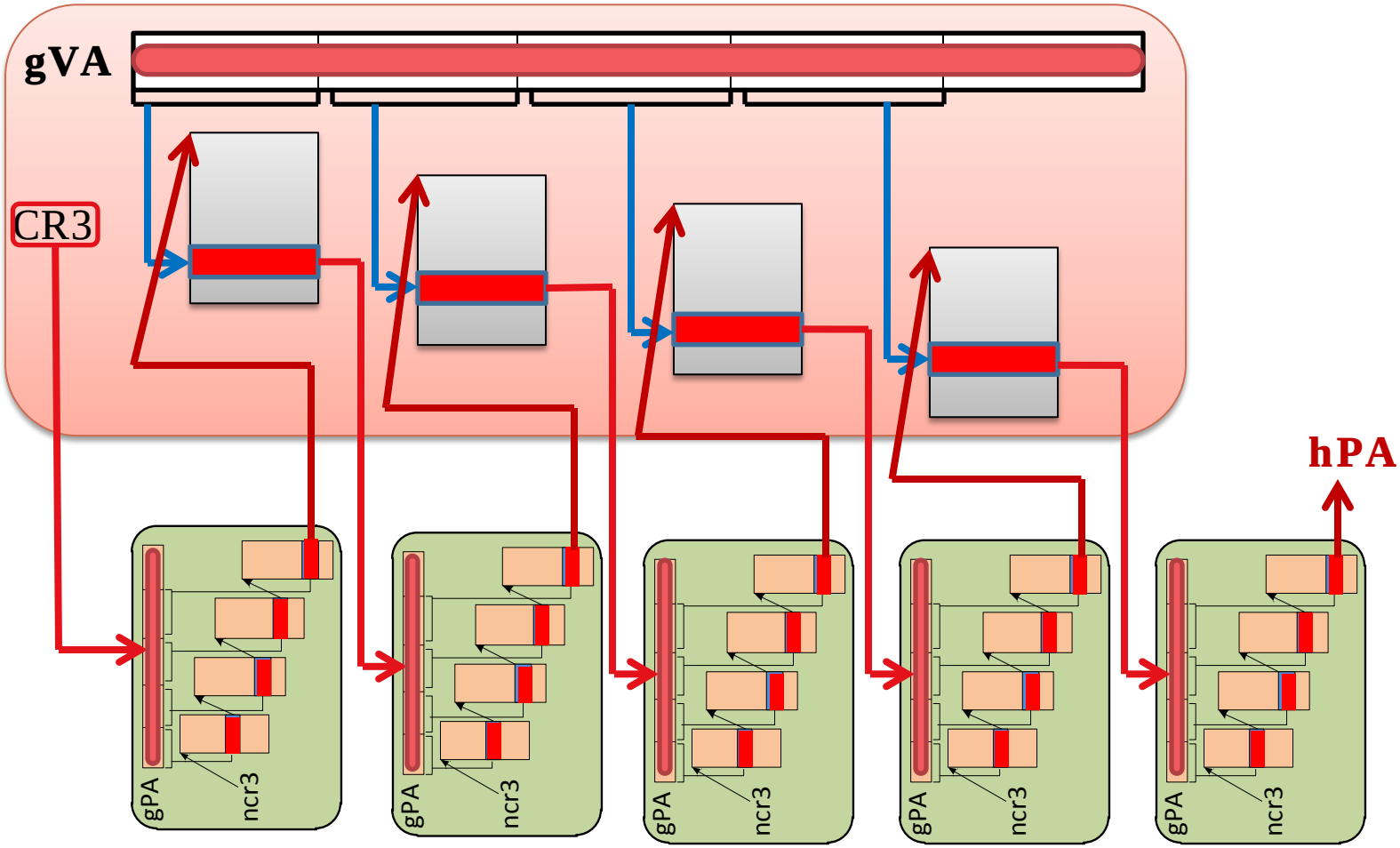


Keeping shadow page table coherent introduces overheads

Hardware support

- Today's hardware is aware of nested pagetable
- Nested page table walk
  - For each level, must do a full pagetable walk
  - Can be very high overhead

Support for Virtualizing Memory



Tradeoffs

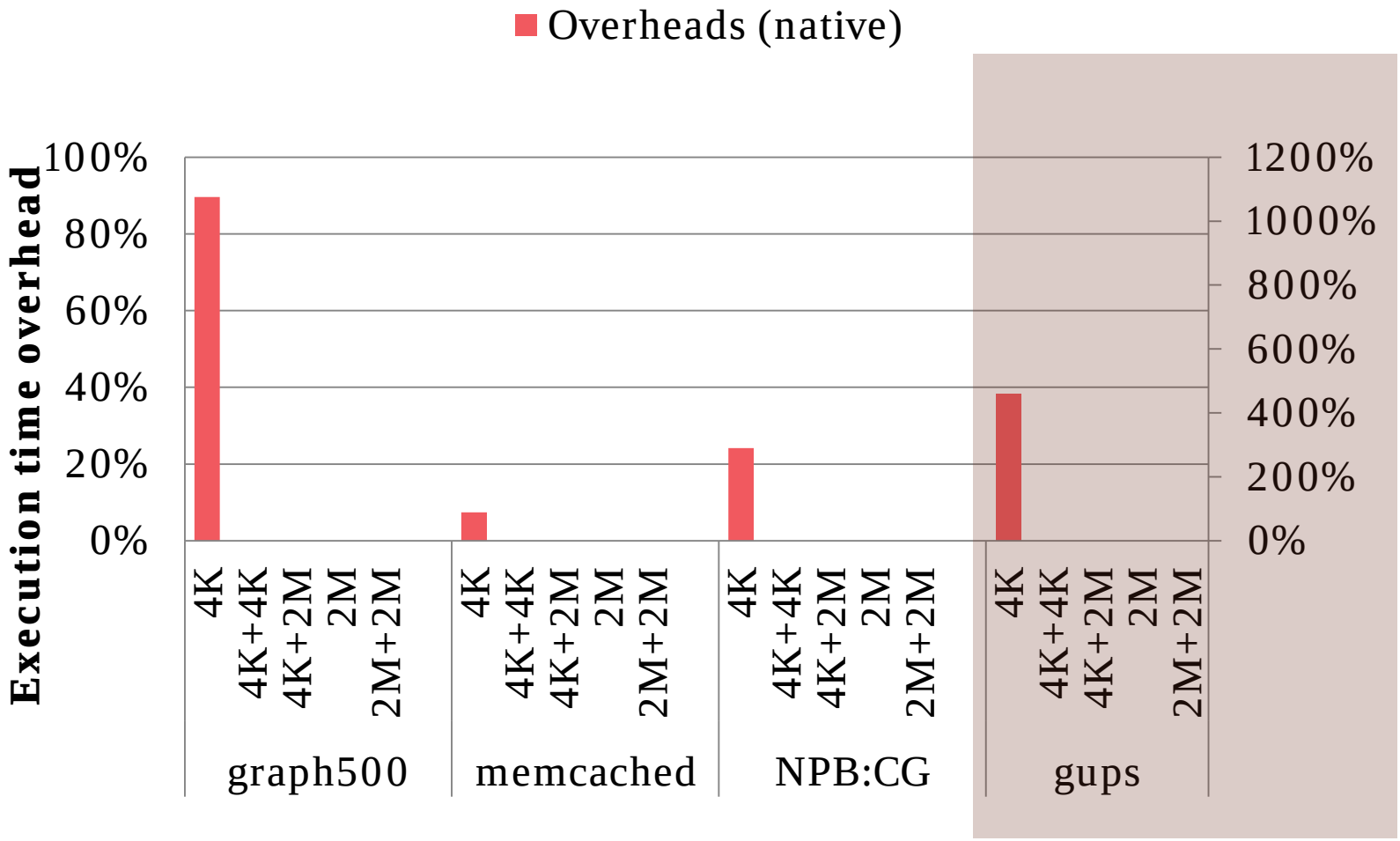
Nested Paging

- Up to 24 memory references
- Updates to either page tables without VMM intervention
- Beneficial with
  - Low TLB miss rate
  - High page table updates

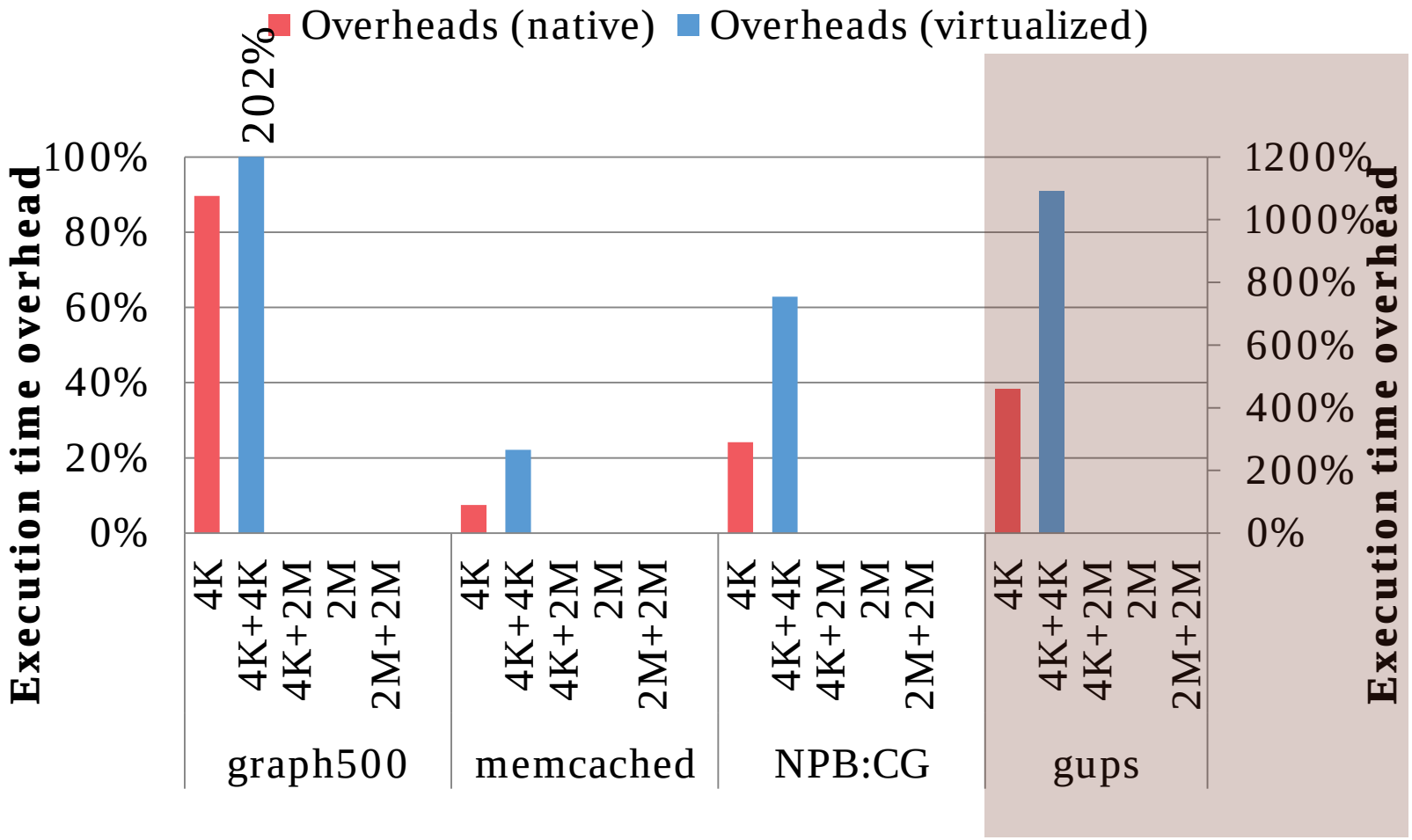
Shadow Paging

- Up to 4 memory references
- Updates to either page tables requires costly VMM intervention
- Beneficial with
  - High TLB miss rate
  - Low page table updates

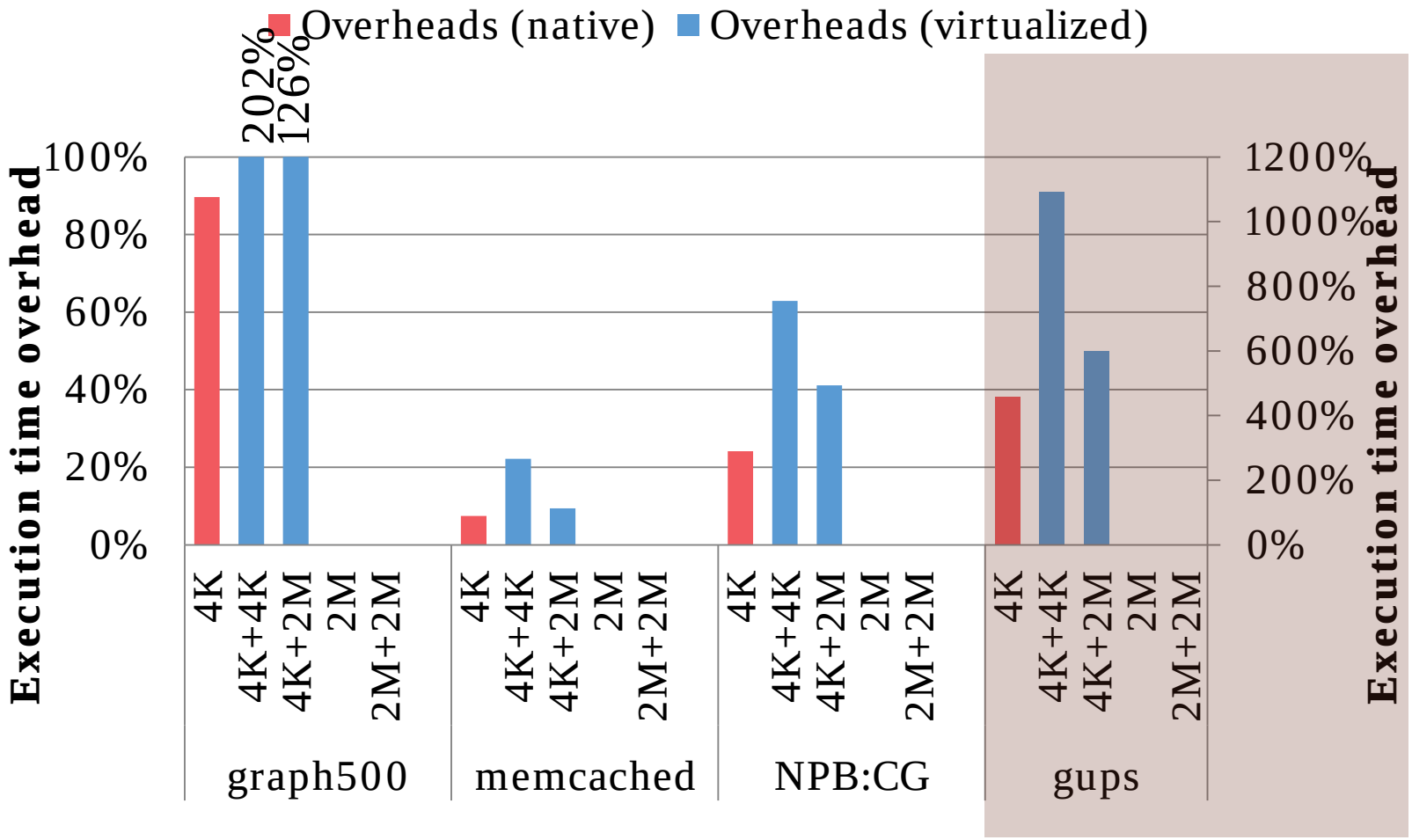
Cost of virtualization



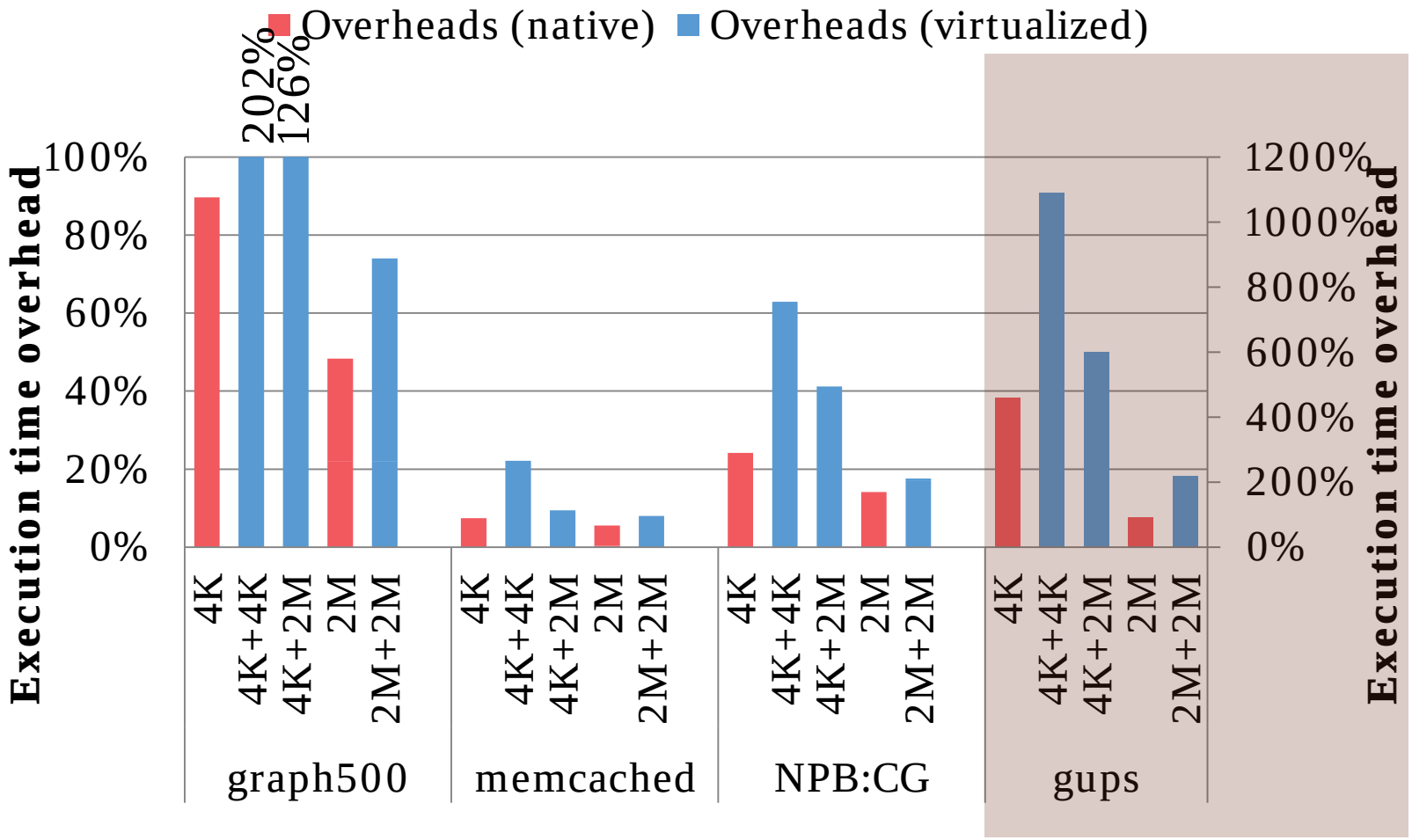
Cost of virtualization



Cost of virtualization



Cost of virtualization



Reducing translation overhead

Bhargava et. al: page walk cache

- Opportunity?
  - PTE reuse (10% of entries cover 90% of accesses)
    - Why?
  - Nested translations are redundant

Reducing translation overhead

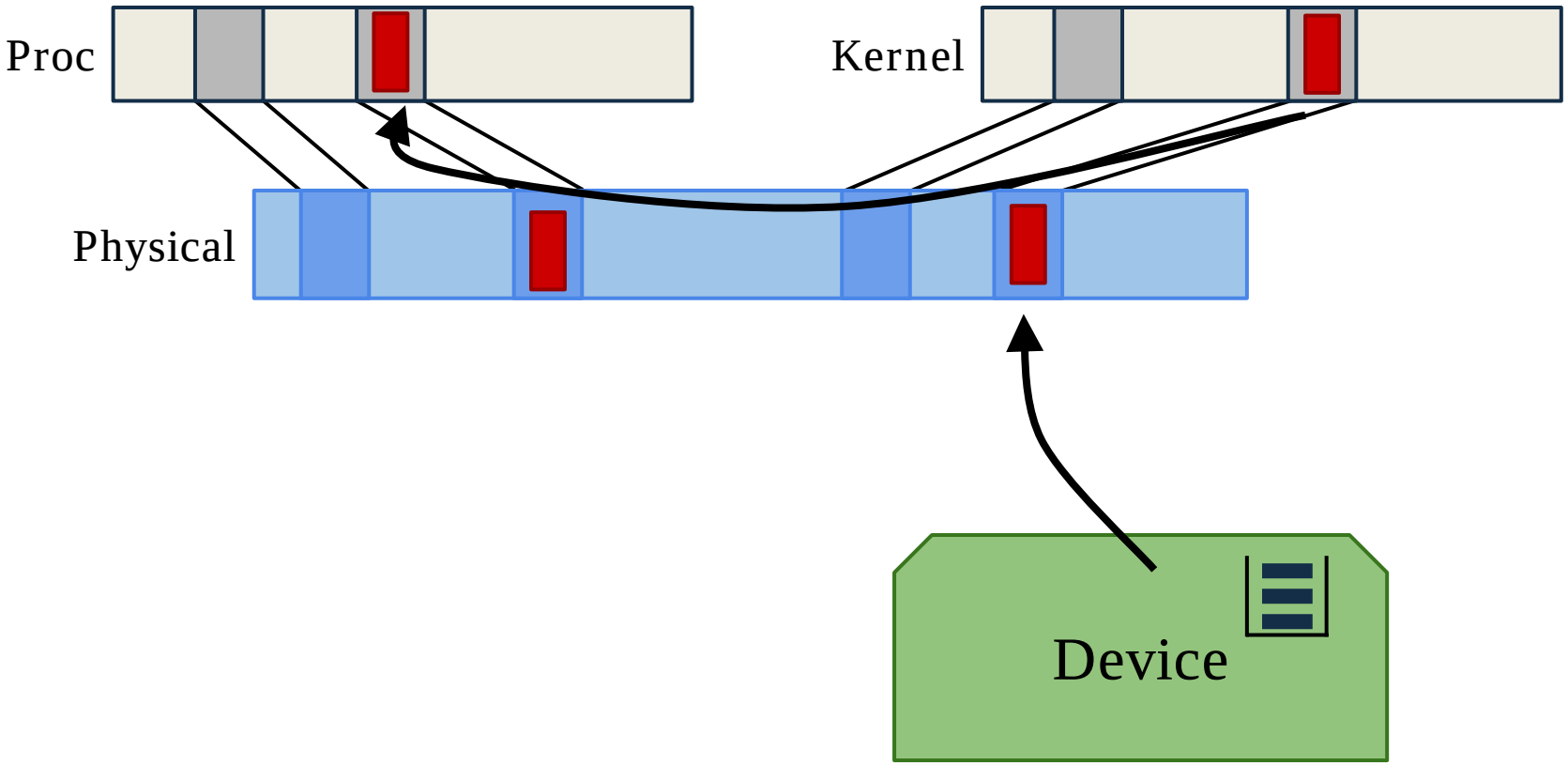
Bhargava et. al: page walk cache

- Page walk cache
  - Why not cache L1 entries?
- What is the NTLB?
  - Caches guest physical to system physical
  - Skips the 2nd dimension walk

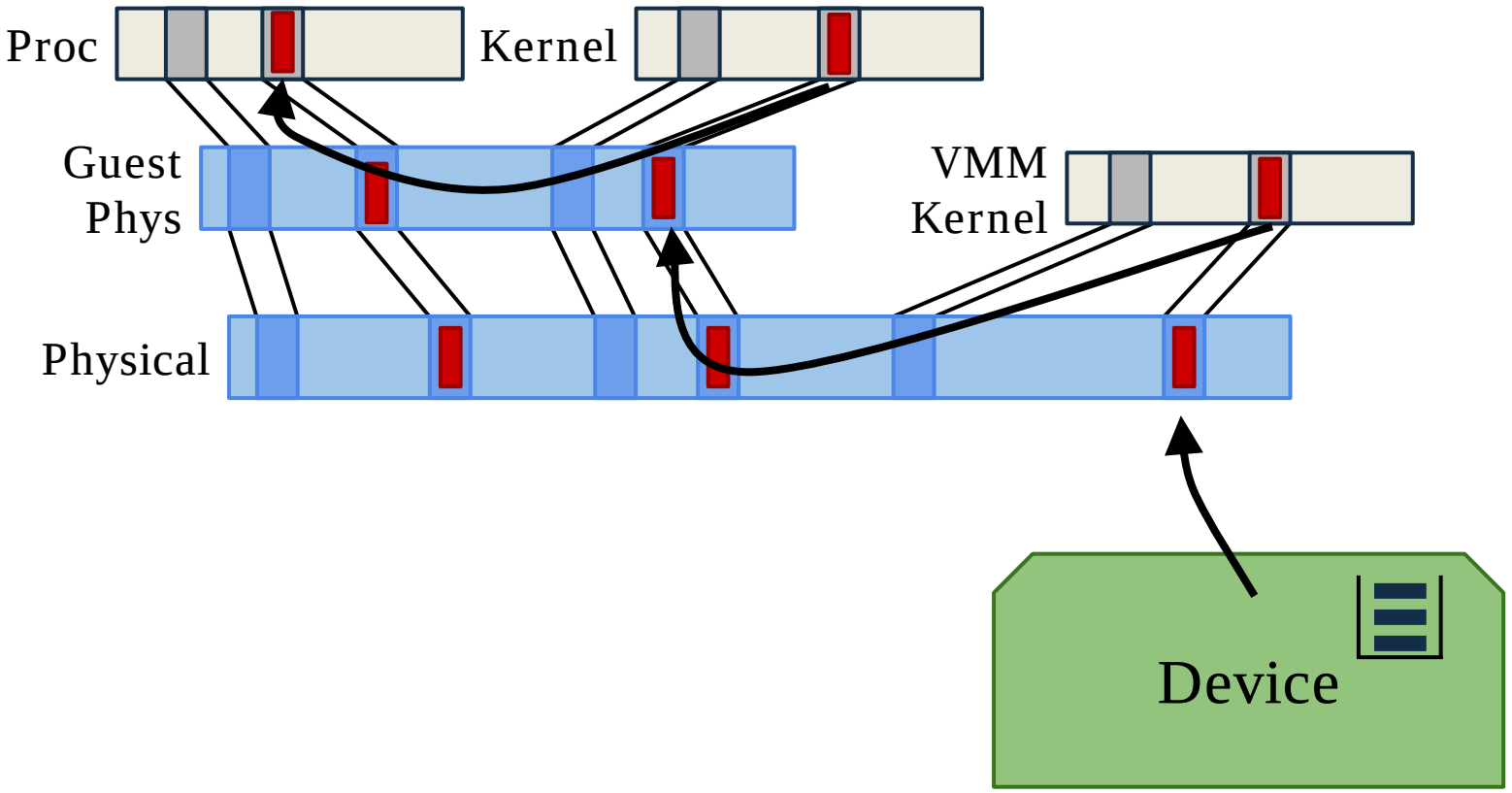
That was fun, let's make it more complicated...

DEVICES AND VIRTUAL MEMORY

No IOMMU: No Virtualization



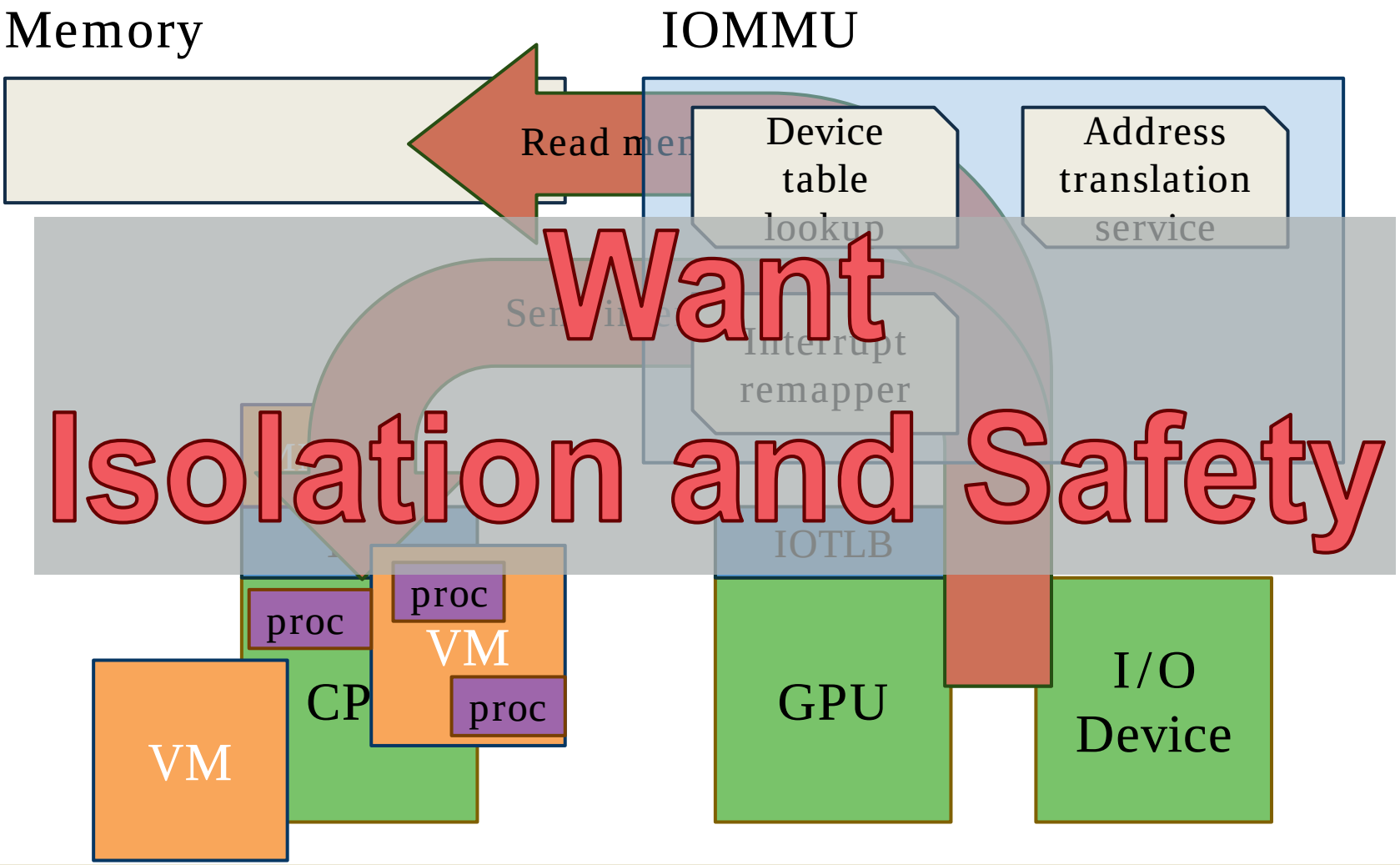
No IOMMU (virtualized)



Virtualization

- Devices accessed by physical addresses
  - Emulation of IO devices is too expensive!
- Approach 1: VMM driver (paravirtualization)
  - Protection domains: IOMMU checks permissions for the memory location; use physical address
  - Need to rewrite drivers!
- Approach 2: Guest driver (true virtualization)
  - Direct Assignment: driver uses guest physical address
  - IOMMU translates to machine physical address

IOMMU Overview



Want

Isolation and Safety

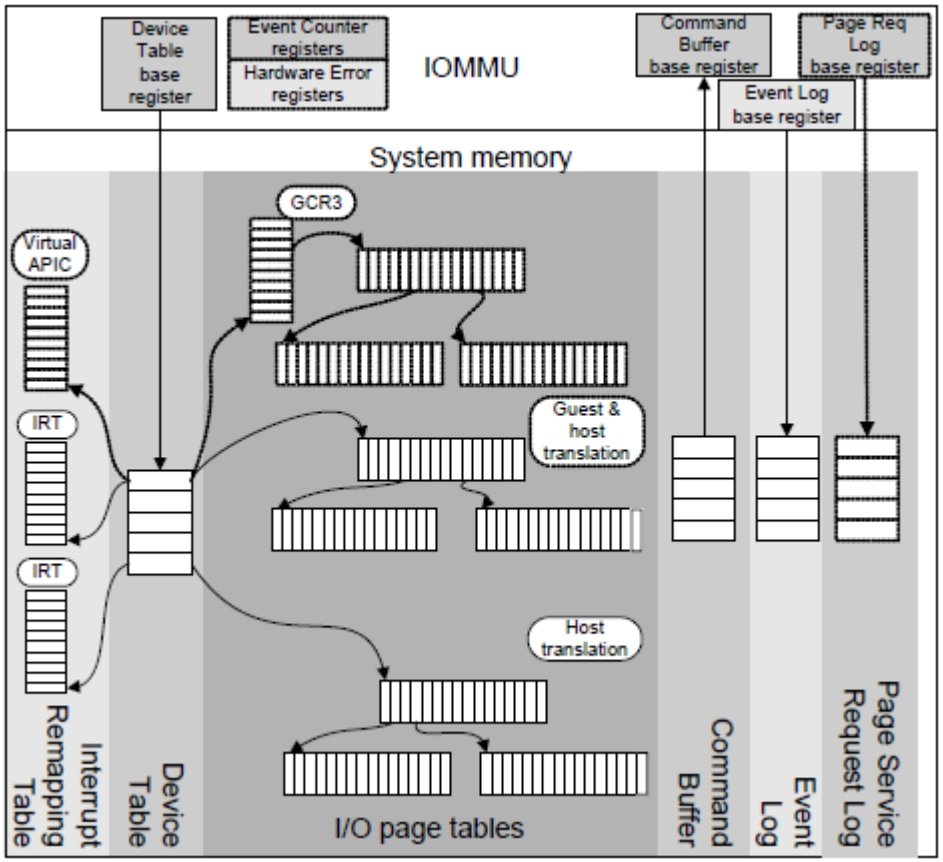
History

- Initially combination of
  - GART (graphics aperture remapping table) and
  - DEV (device exclusion vector)
- GART
  - Physical-to-physical translation so graphics addresses appear contiguous
  - IOMMU is a generalization
- DEV
  - Devices classified into domains
  - Each domain is allowed to access a set of physical addresses

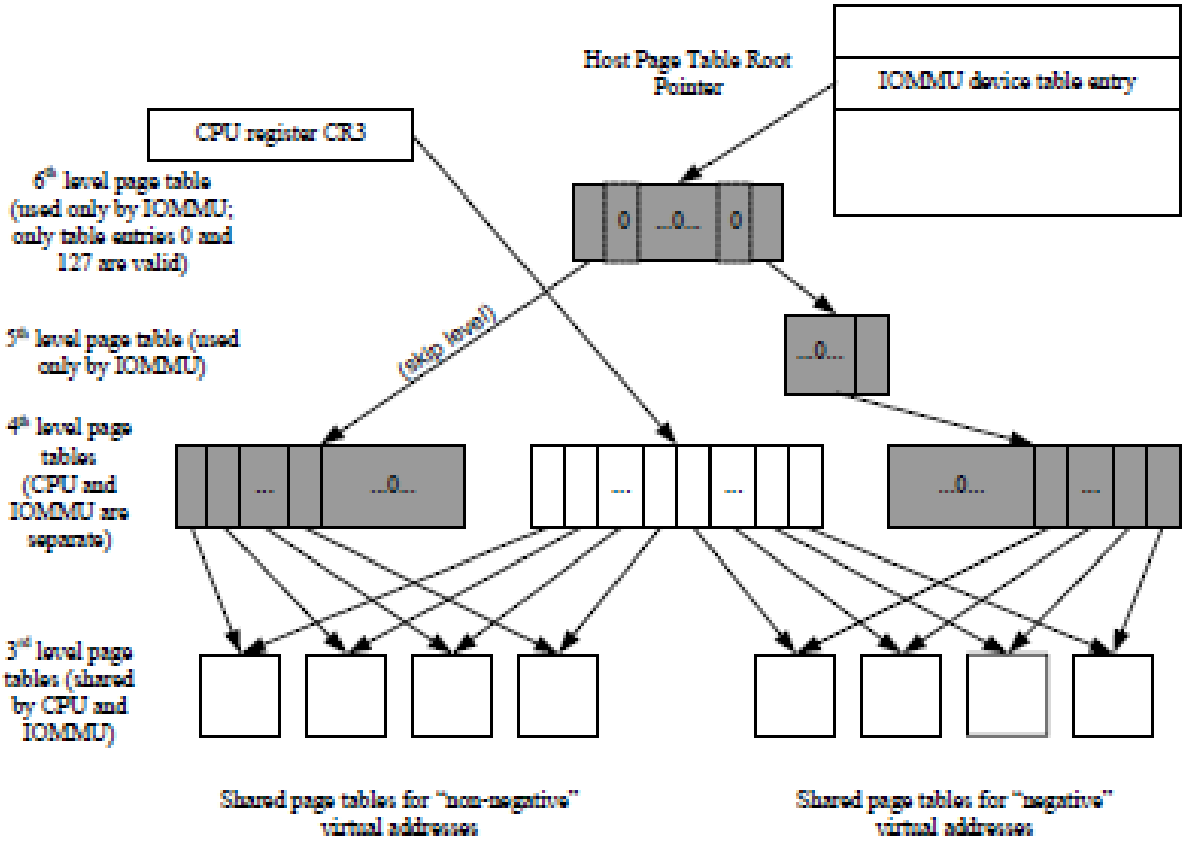
Laundry list of features

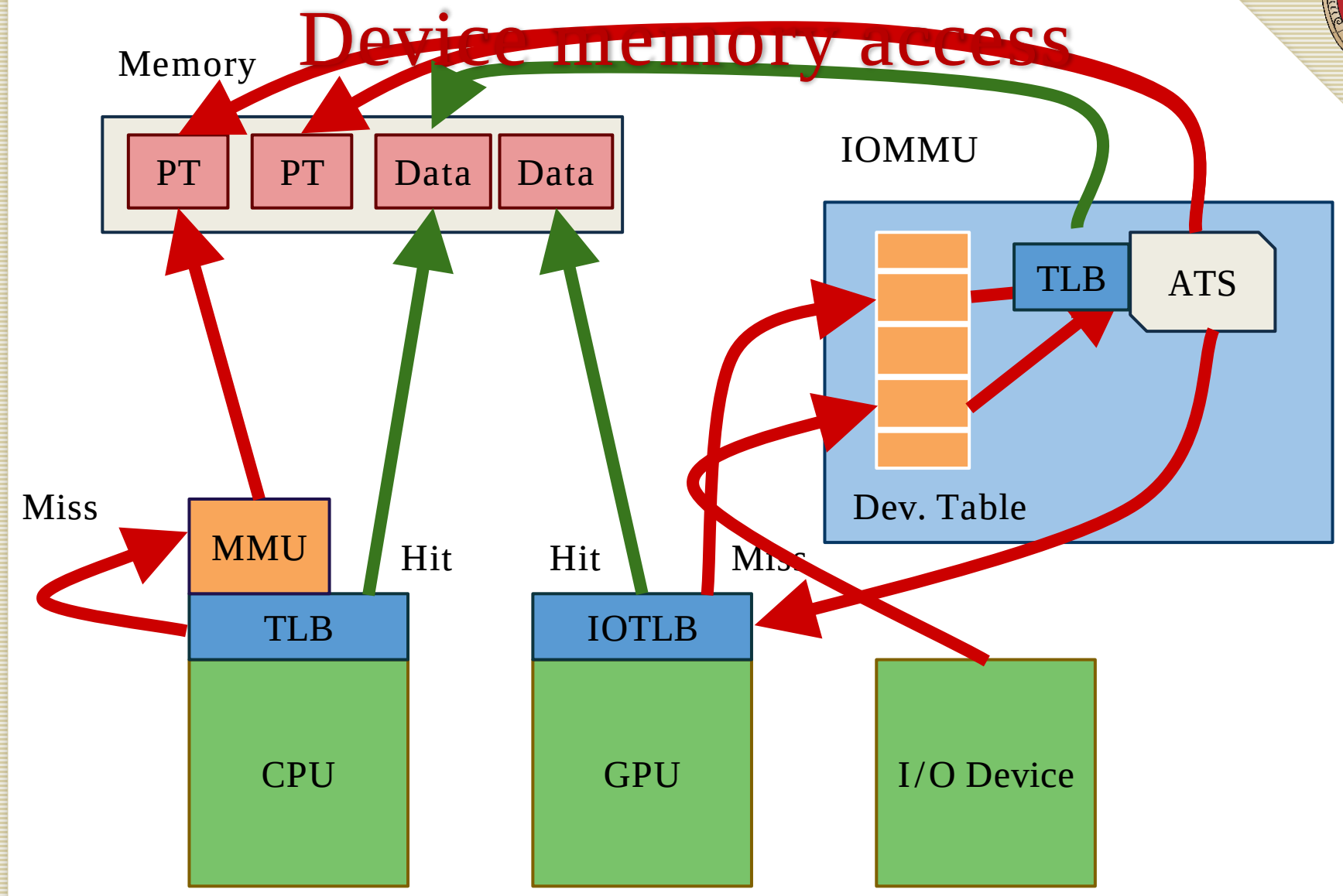
- I/O page tables for I/O devices to access memory
  - permission checking
  - virtual address translation
- Interrupt remapping for I/O interrupts
- Service page faults from I/O devices
- Legacy I/O
- User mode device access
- VM guest device access
- Virtualized user mode device access
- Two-level address translation
- Interrupt virtualization
- …

IOMMU data structures



I/O page tables




Device memory access
Memory
PT
PT
Data
Data
IOMMU
TLB
ATS
Dev. Table
Miss
MMU
Hit
Hit
TLB
IOTLB
CPU
GPU
I/O Device

Page faults (before)

- Generated if the I/O device accesses unallowed memory
- Fatal error
- Written to a log

- Requires pinned memory

Page faults (now)

- Generated if the I/O device accesses unallowed memory
- Written to a buffer
- Interrupt raised on CPU core
  - (Kernel) driver handles the fault
- No support to notify the device it should retry
  - Device keeps on executing/waiting for the TLB miss