CS5460: Operating Systems

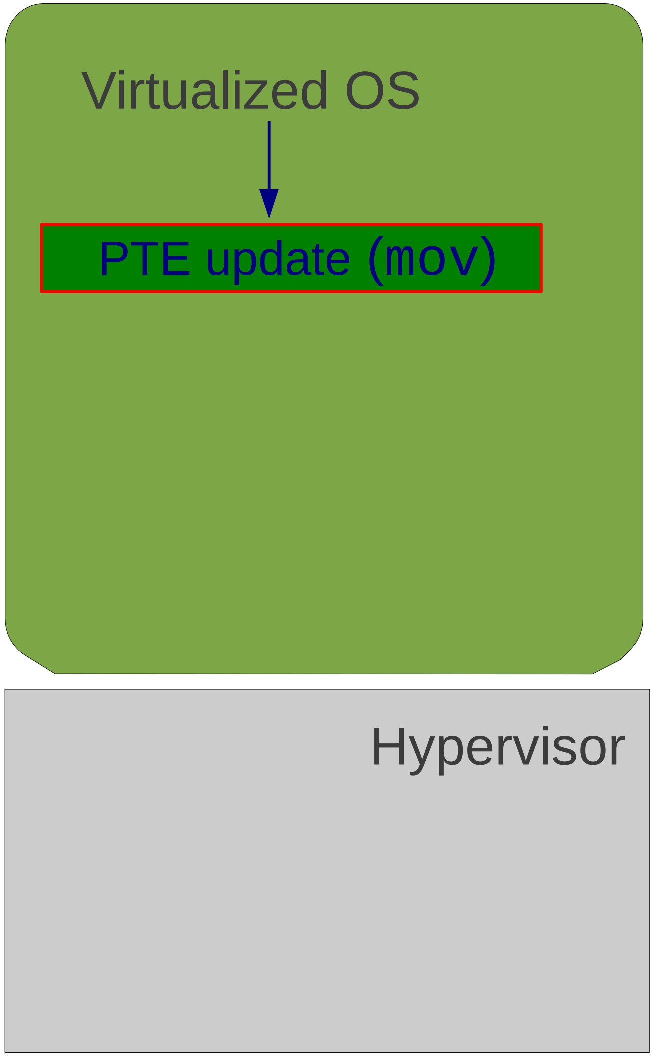
Lecture: Virtualization 2

Anton Burtsev
March, 2013

Paravirtualization: Xen

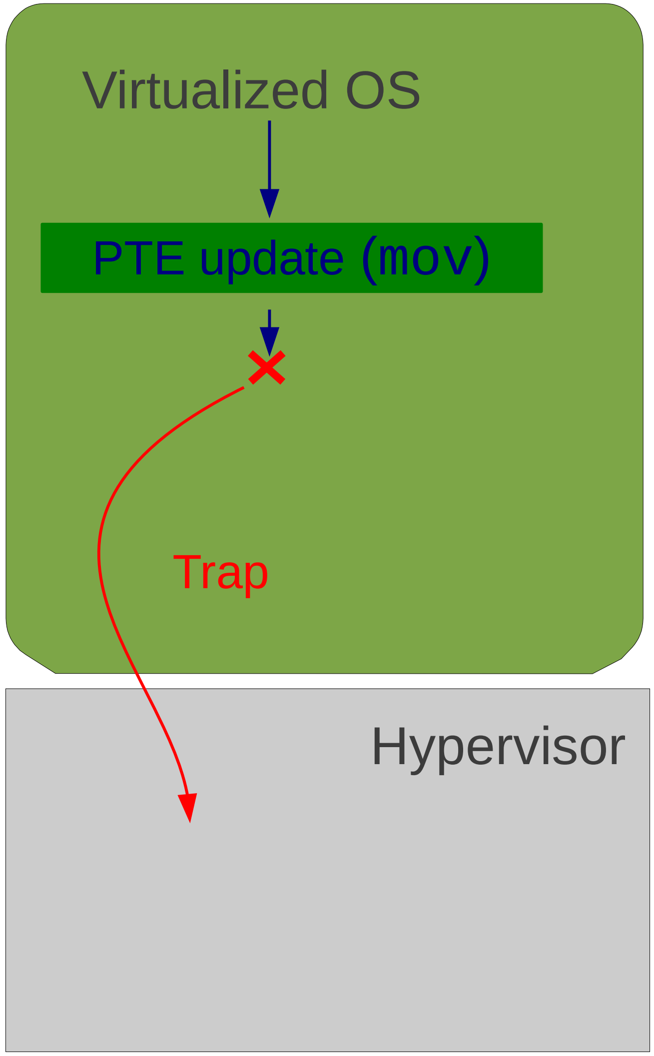
Full virtualization

- Complete illusion of physical hardware
  - Trap _all_ sensitive instructions
  - Example: page table update



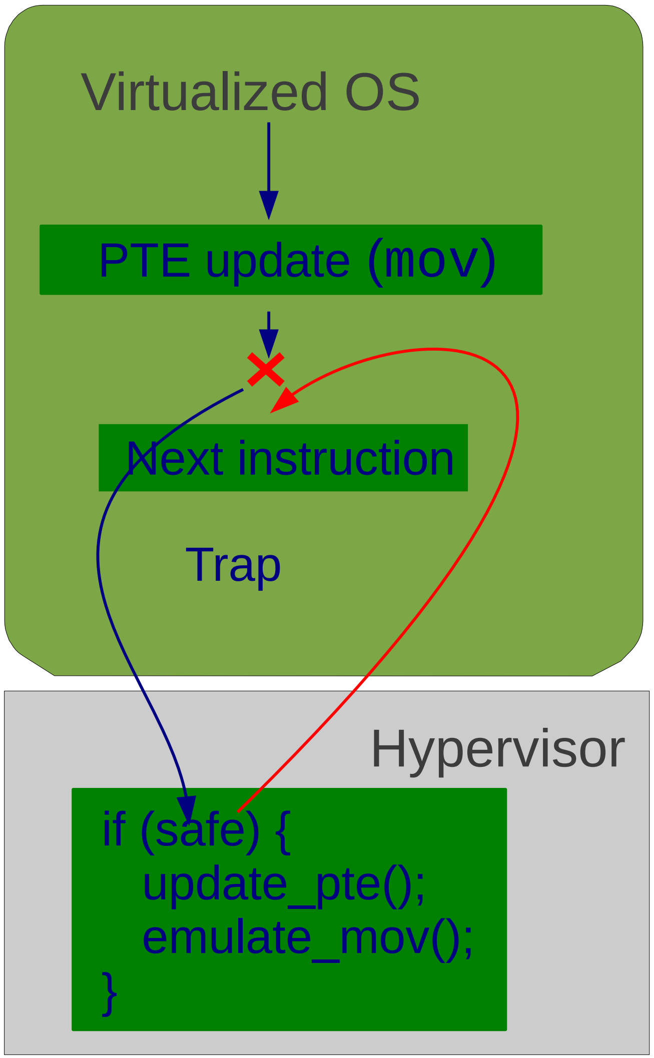
Full virtualization

- Complete illusion of physical hardware
  - Trap _all_ sensitive instructions
  - Example: page table update



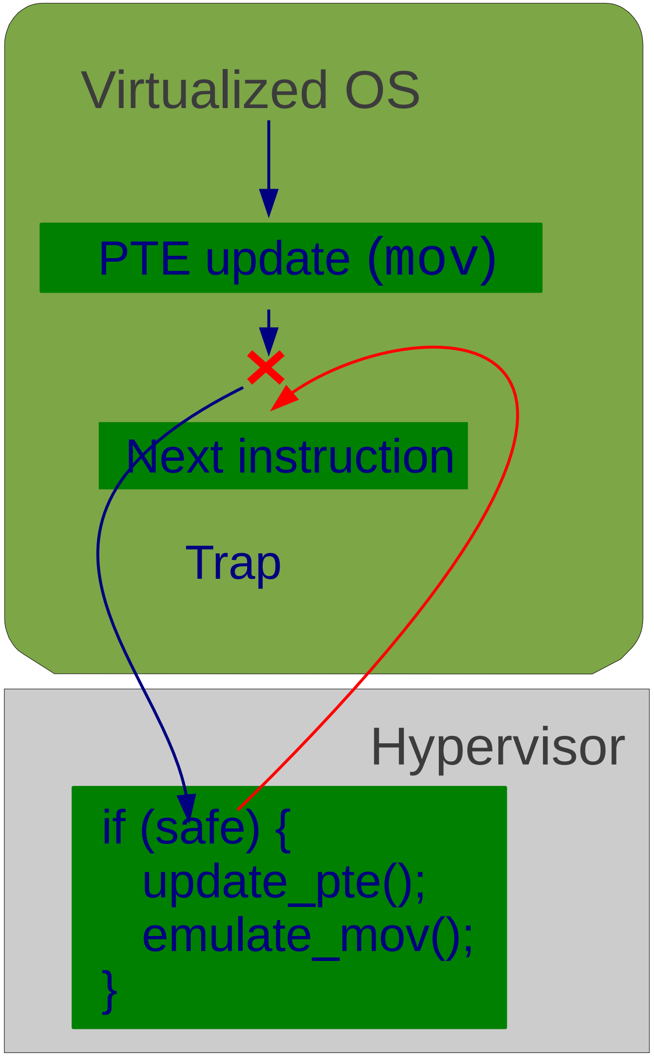
Full virtualization

- Complete illusion of physical hardware
  - Trap _all_ sensitive instructions
  - Example: page table update
- Traps are slow
- Binary translation is faster, for some events
  - Not for PTE updates, why?



Performance problems

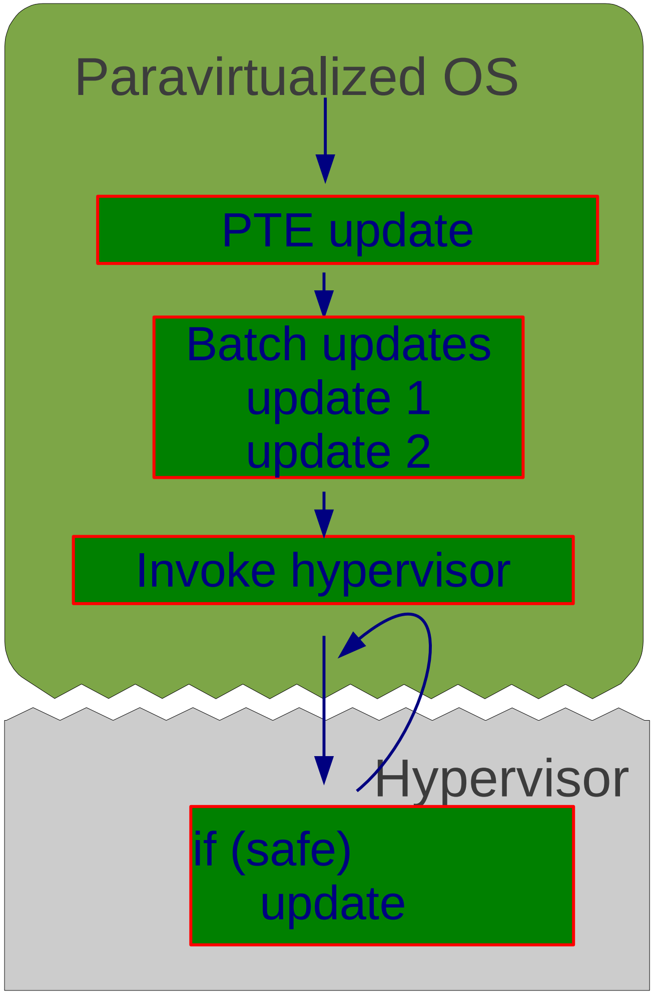
- Traps are slow
- Binary translation is faster
  - For some events
  - Not for PTE updates, why?



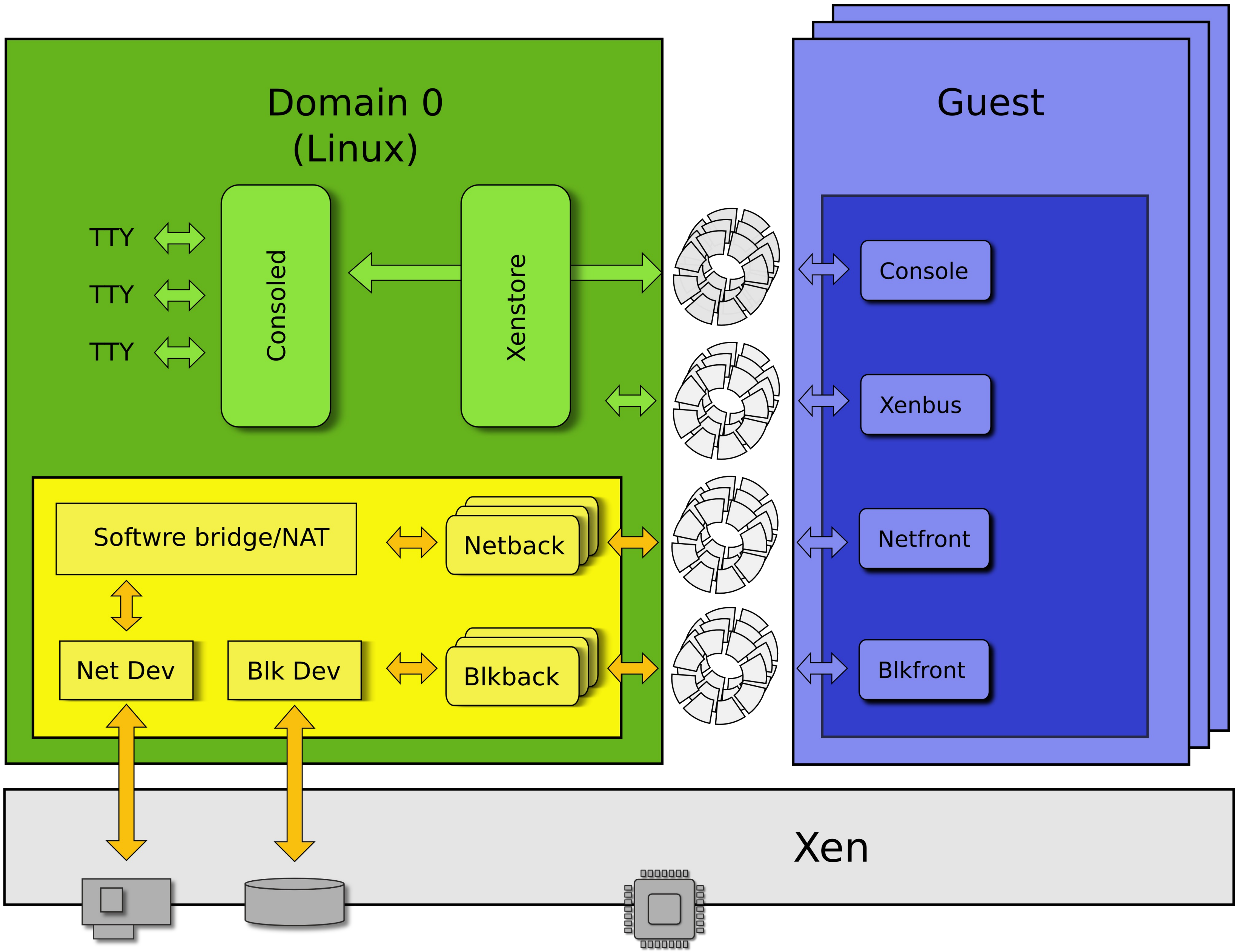
Paravirtualization

- No illusion of hardware
- Instead: paravirtualized interface
  - Explicit hypervisor calls to update sensitive state
    - Page tables, interrupt flag
- But Guest OS needs porting
  - Applications run natively in Ring 3

Paravirtualization

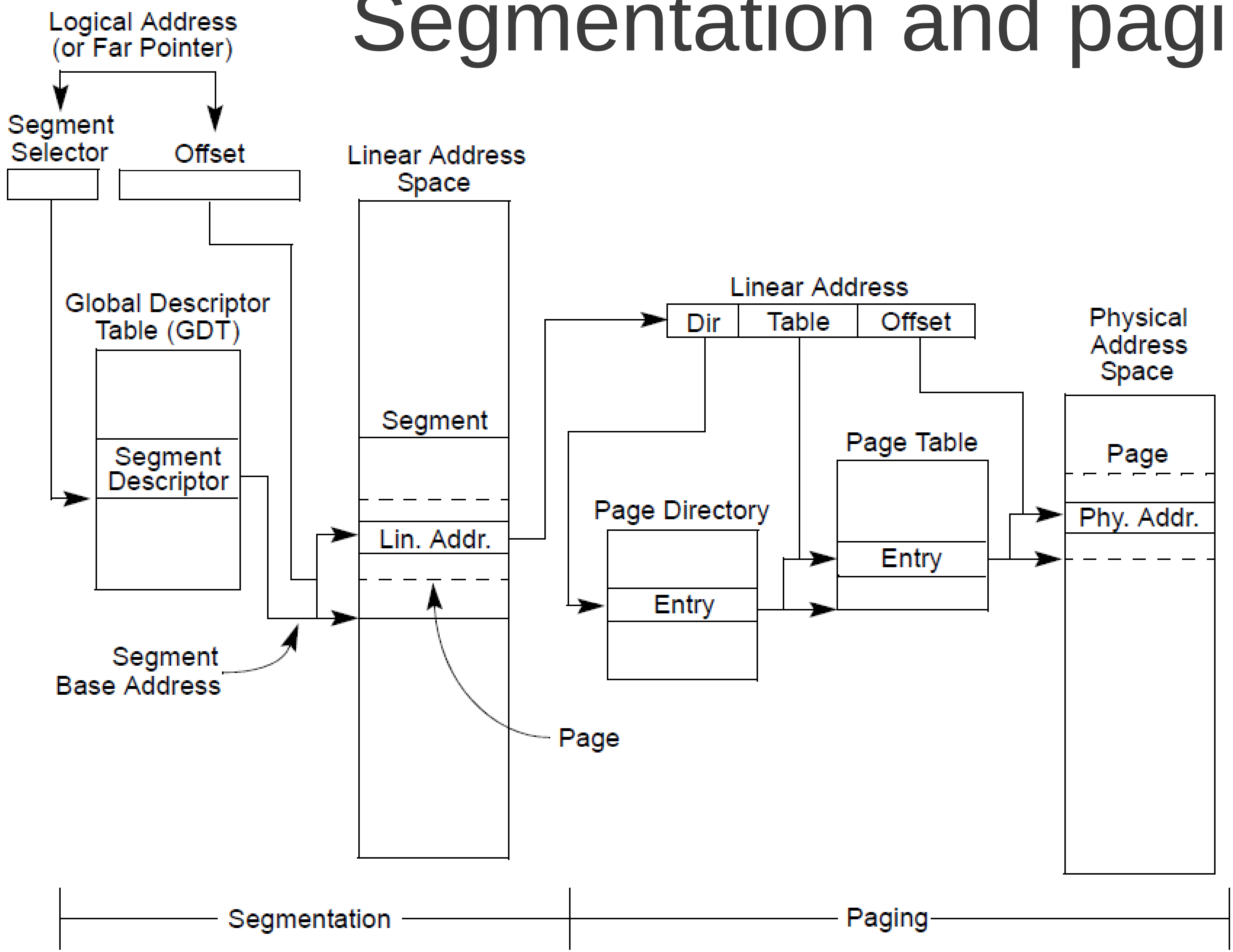


Xen

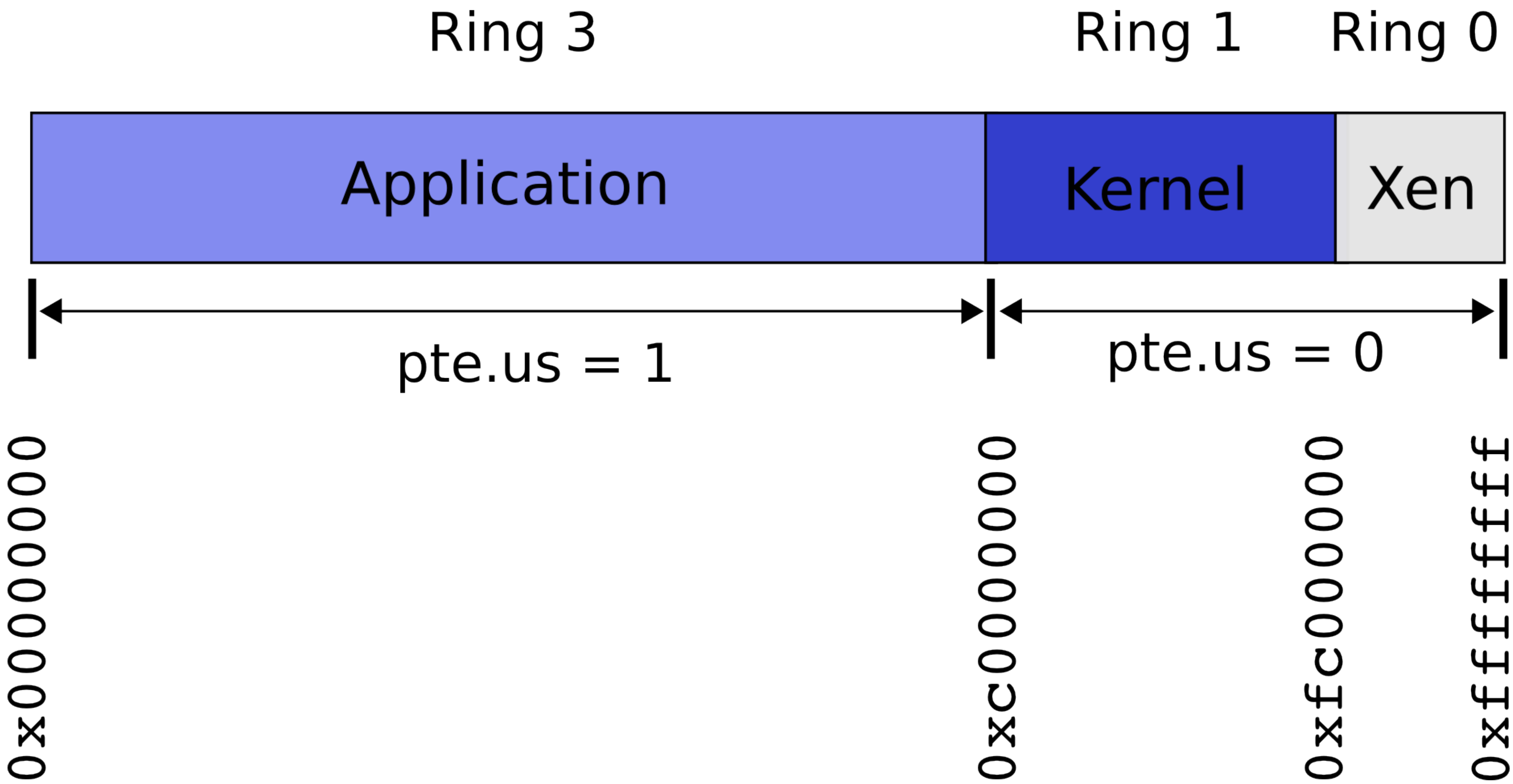


Domain 0
(Linux)
TTY
TTY
TTY
Consoled
Xenstore
Softwre bridge/NAT
Netback
Net Dev
Blk Dev
Blkback
Guest
Console
Xenbus
Netfront
Blkfront
Xen

Segmentation and paging

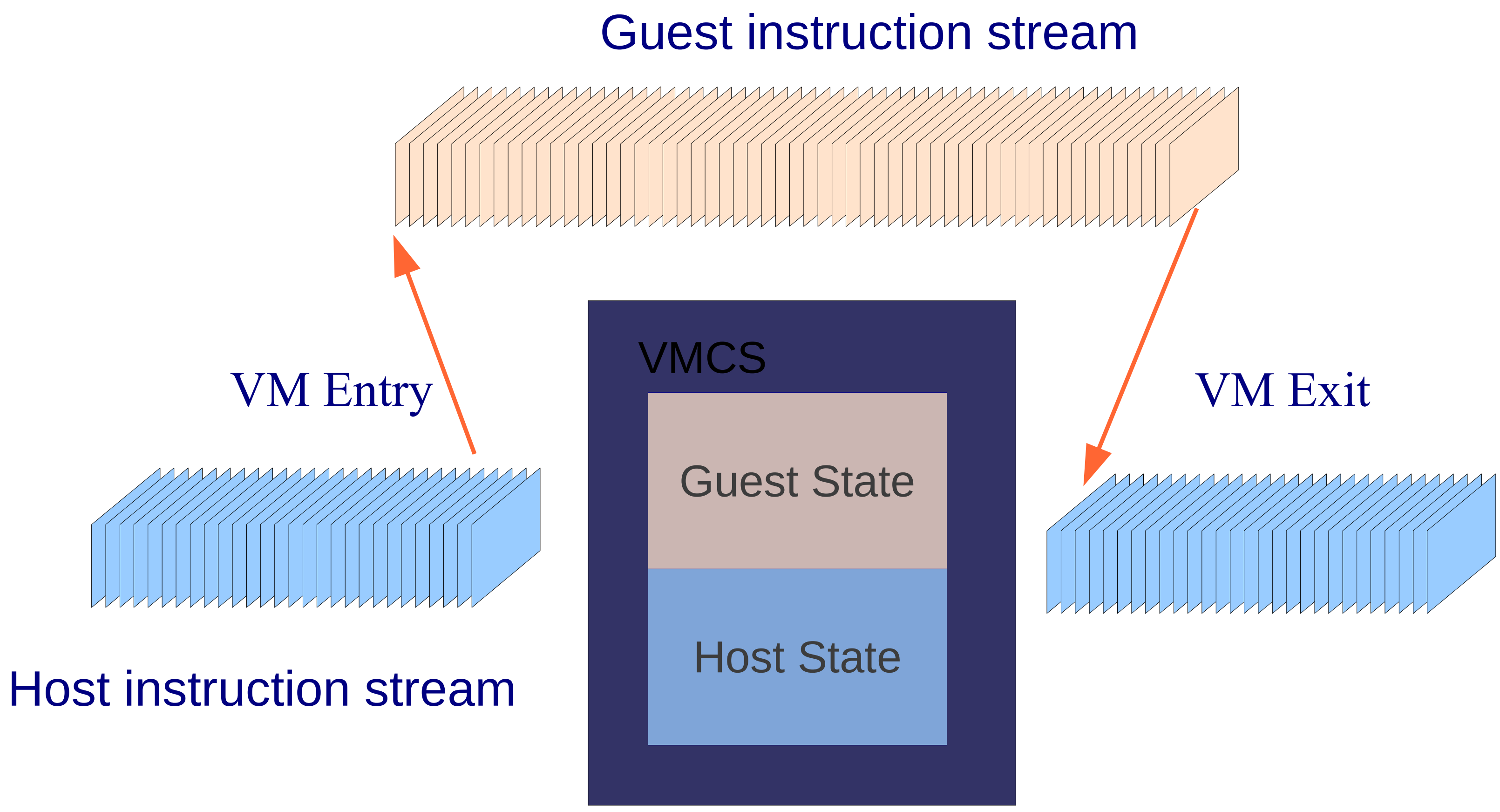


Hypervisor protection



Hardware support for virtualization: KVM

Basic idea


Guest instruction stream
VM Entry
VMCS
Guest State
Host State
VM Exit
Host instruction stream

New mode of operation:VMX root

- VMX root operation
  - 4 privilege levels
- VMX non-root operation
  - 4 privilege levels as well, but unable to invoke VMX root instructions
  - Guest runs until it performs exception causing it to exit
  - Rich set of exit events
  - Guest state and exit reason are stored in VMCS

Virtual machine control structure (VMCS)

- Guest State
  - Loaded on entries
  - Saved on exits
- Host State
  - Saved on entries
  - Loaded on exits
- Control fields
  - Execution control, exits control, entries control

Guest state

- Register state
- Non-register state
  - Activity state:
    - active
    - inactive (HLT, Shutdown, wait for Startup IPI interprocessor interrupt))
  - Interruptibility state

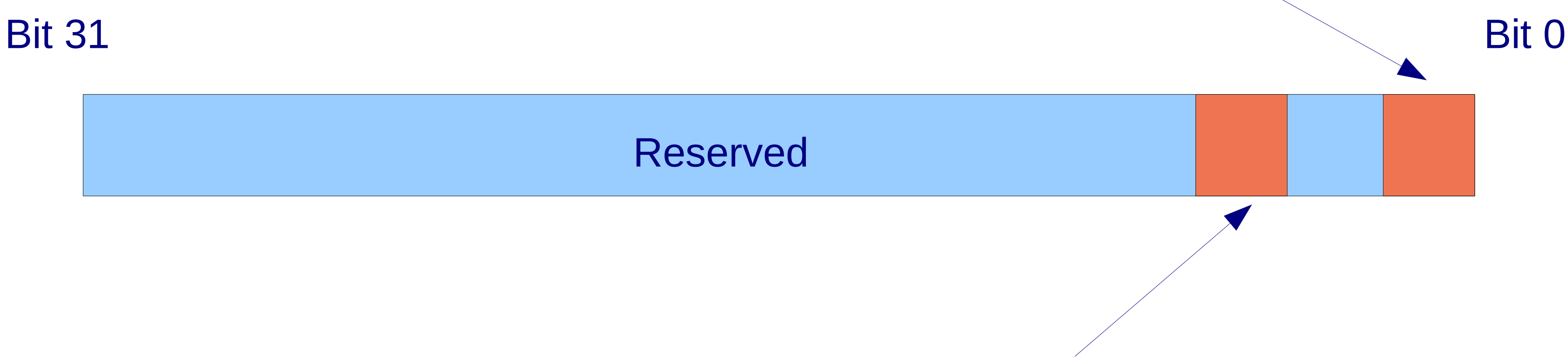
Host state

- Only register state
  - ALU registers,
- also:
  - Base page table address (CR3)
  - Segment selectors
  - Global descriptors table
  - Interrupt descriptors table

VM-execution controls

(asynchronous events control)

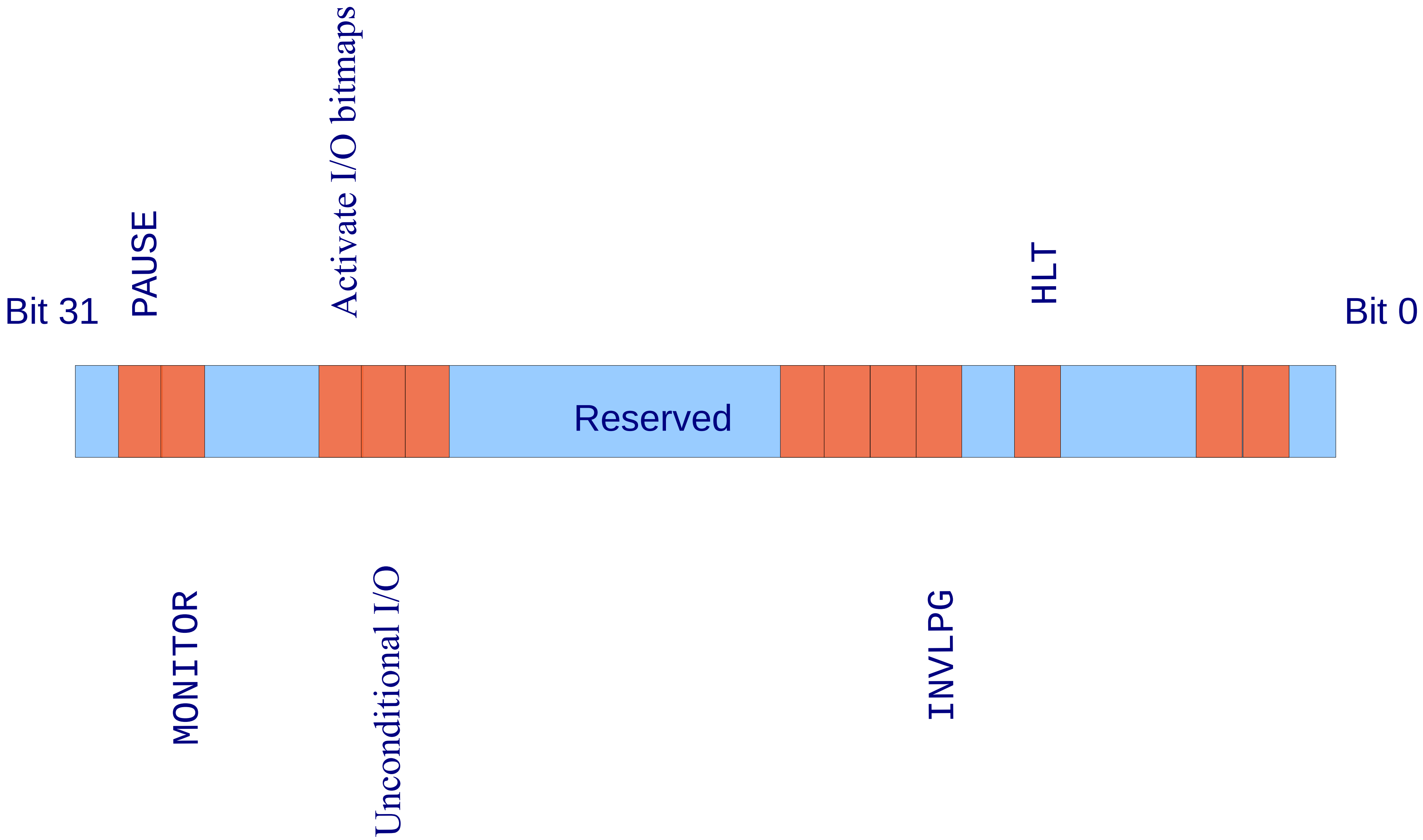
External interrupts (maskable or IRQs) cause exits(yes/no)
If not, then they delivered through guest IDT



NMI cause exits (yes/no)
If not, then they are delivered normally through guest IDT (descriptor 2)

VM-execution controls

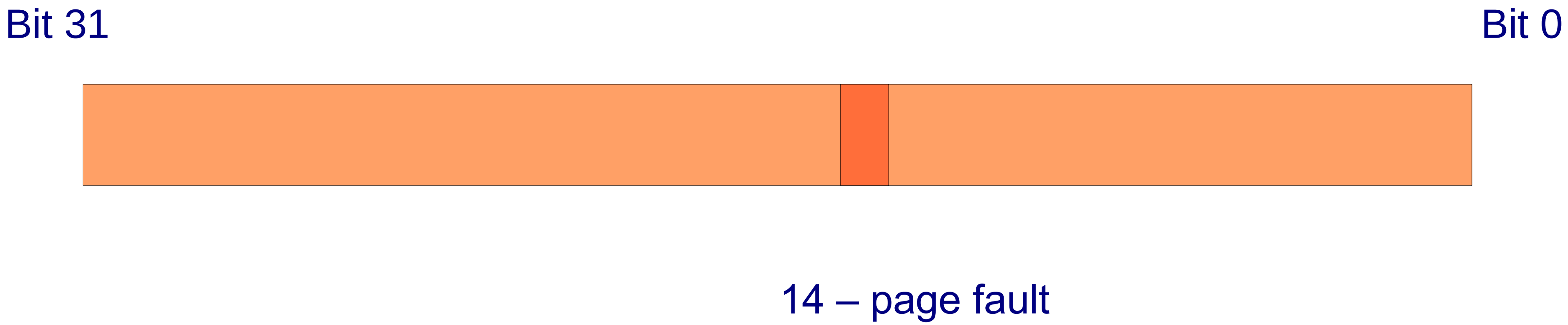
(synchronous events control, not all reasons are shown)



Exception bitmap

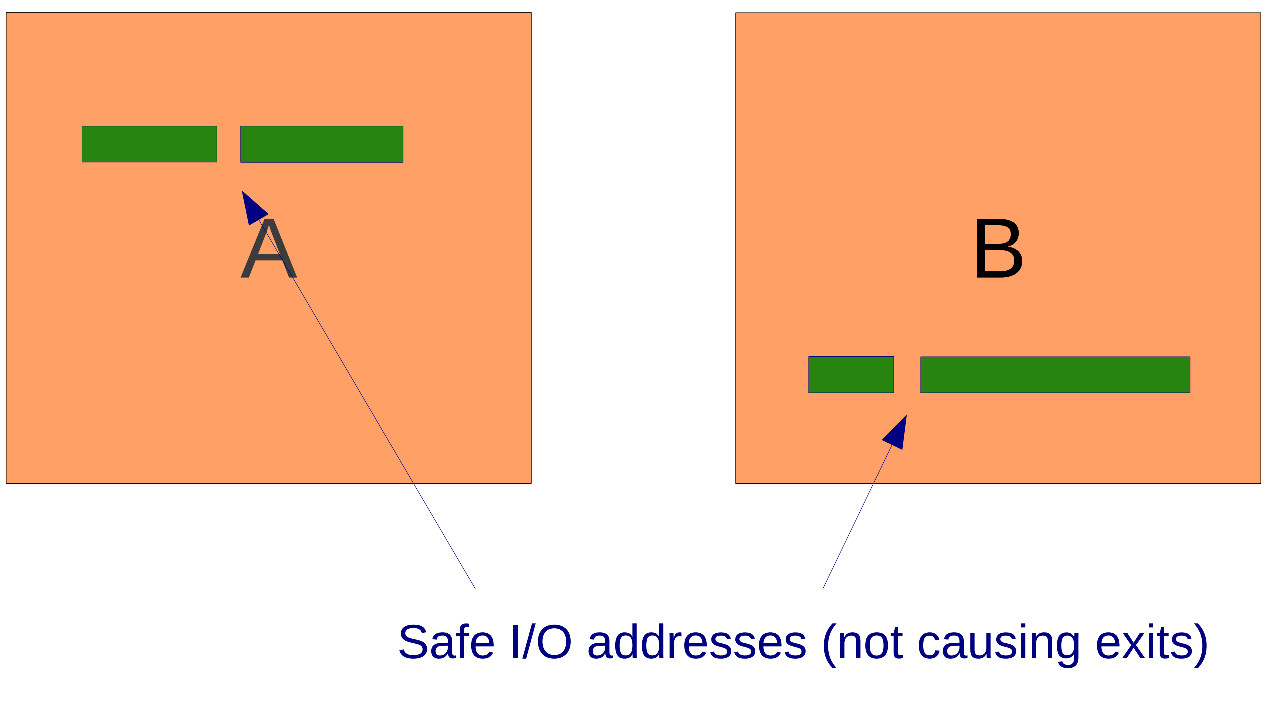
(one for each of 32 IA-32 exceptions)

- IA-32 defines 32 exception vectors (interrupts 0-31)
- Each of them is configured to cause or not VM-exit



I/O Bitmaps

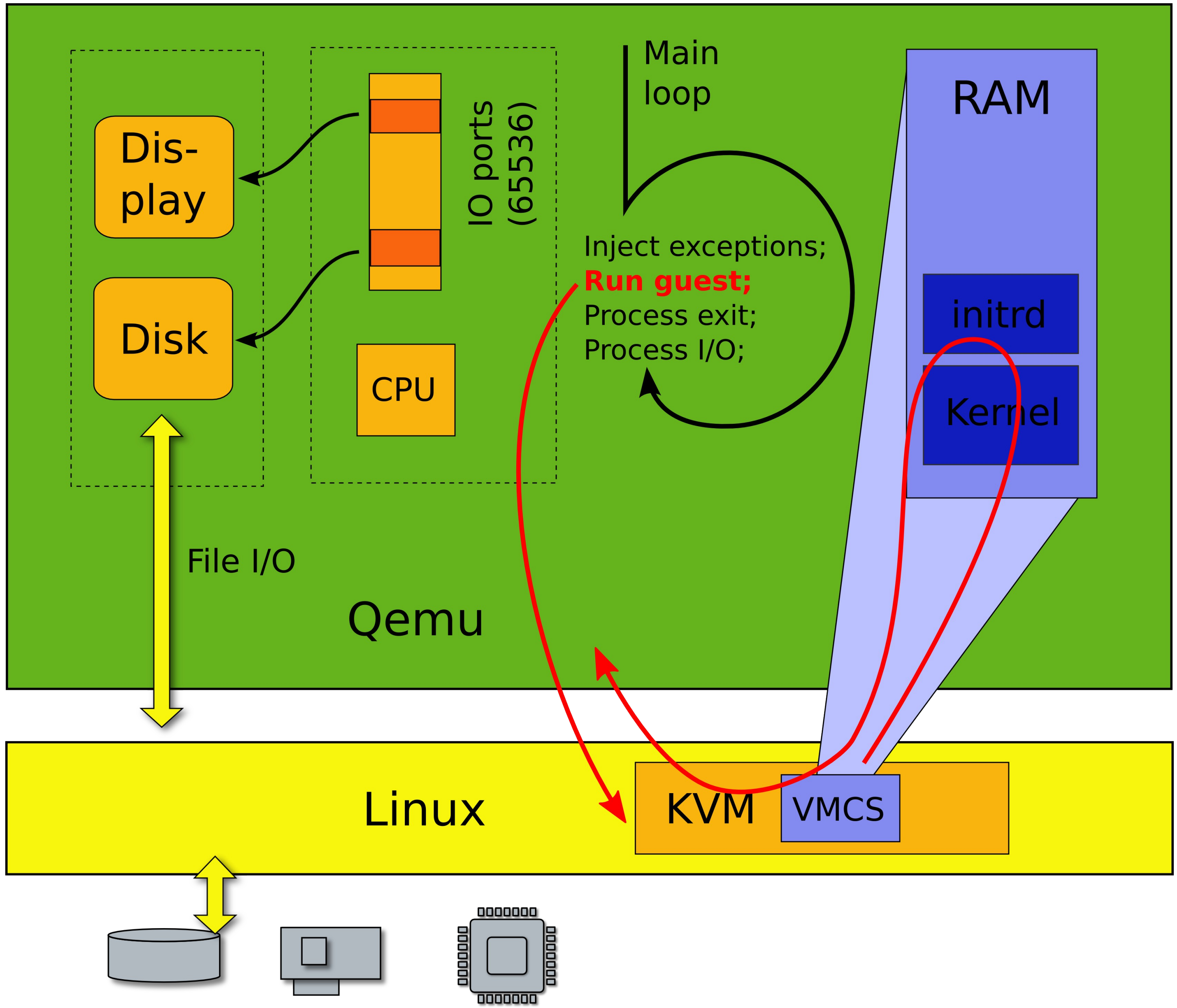
- Two addresses on 4KB memory areas (A and B)



Exit information

- Information describing conditions of VM-exit is saved in VMCS
  - It's different for different types of event

KVM

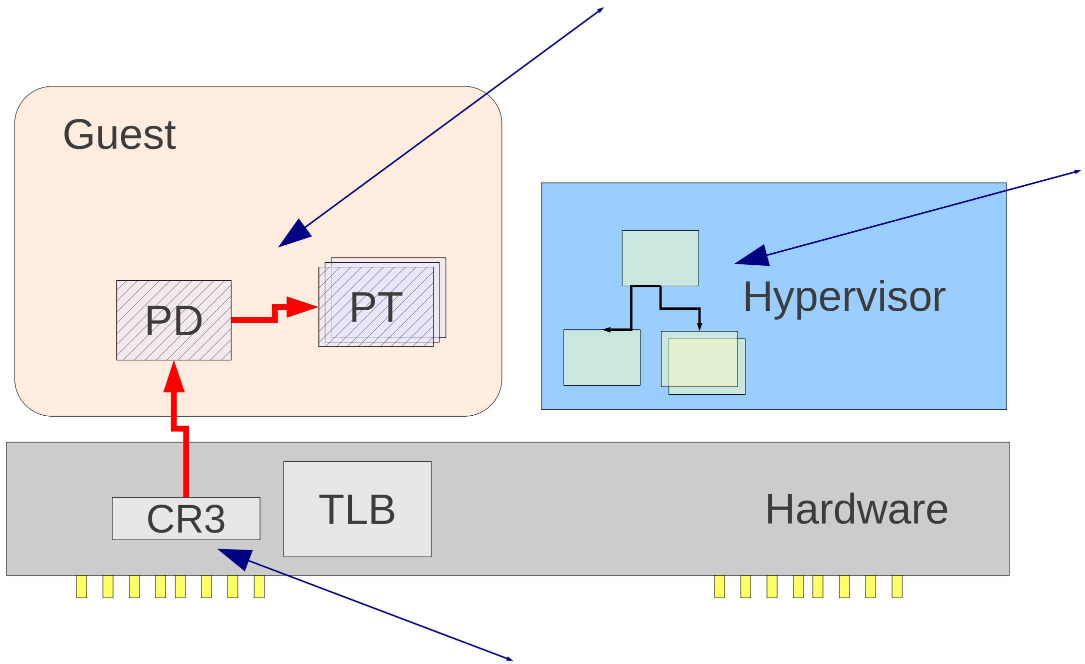


Dis-
play
Disk
IO ports
(65536)
CPU
Main
loop
Inject exceptions;
Run guest;
Process exit;
Process I/O;
RAM
initrd
Kernel
File I/O
Qemu
Linux
KVM
VMCS

Memory virtualization: brute force.

Write / read protected
page table area.

Every access results in VM-Exit and
passes control to hypervisor



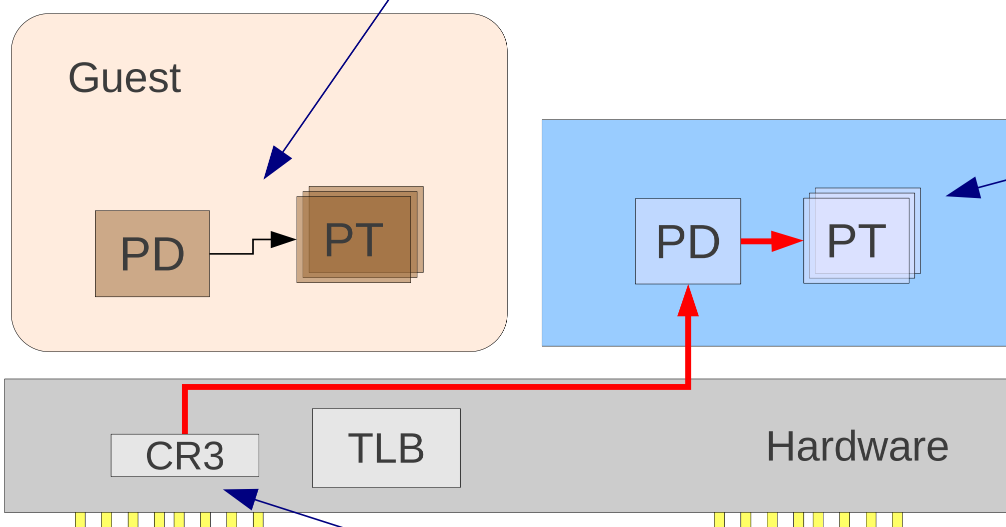
Helper structures describe
actual guest VM layout

Maintained for each guest. On
VM-Exit hypervisor adjusts guest
page accordingly.

CPU stores pointer on
guest page table directory

Memory virtualization: shadow page tables

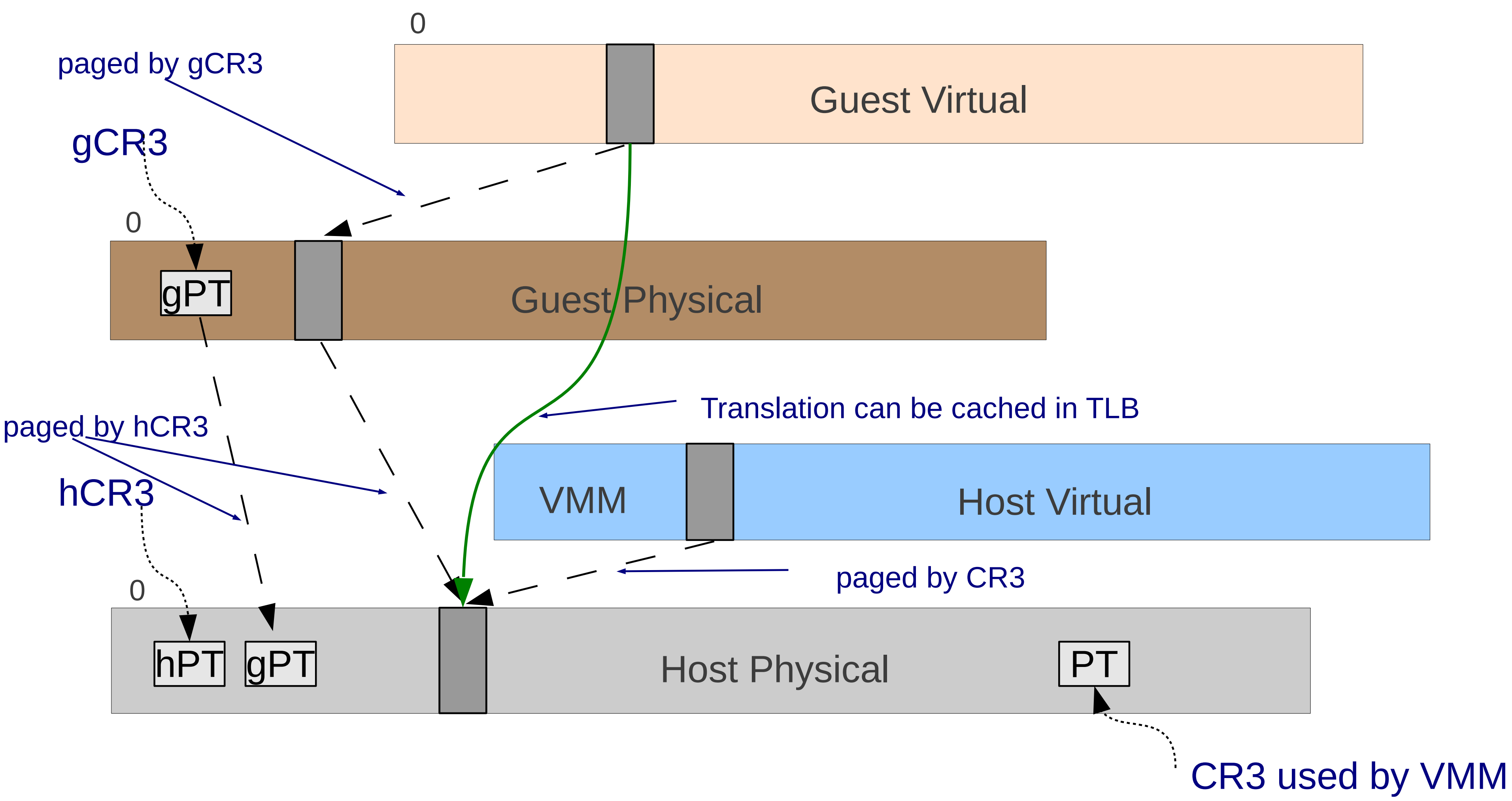
Guest page table hierarchy
It's writable, but can be inconsistent with active page table hierarchy stored by the hypervisor



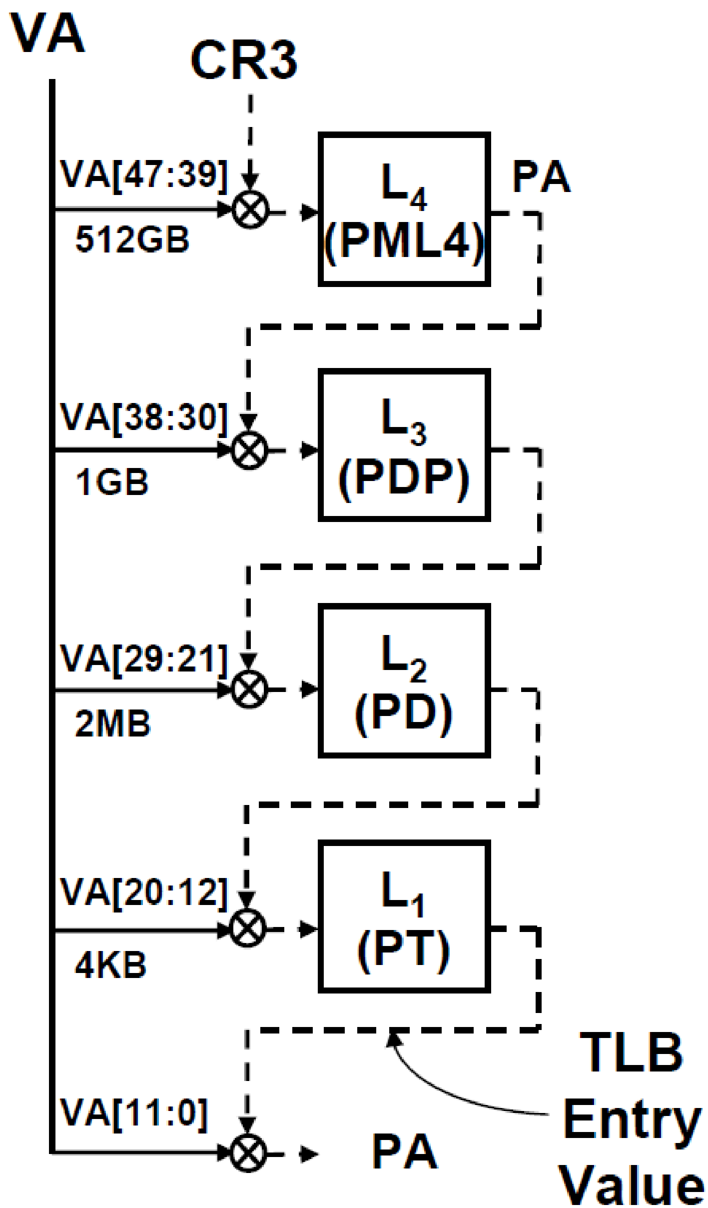
Active page table hierarchy
VMM maintains it for each VM that it supports

CPU stores pointer on active page table hierarchy.
On Intel CPUs TLB is always refilled from active page table directory

Nested page tables

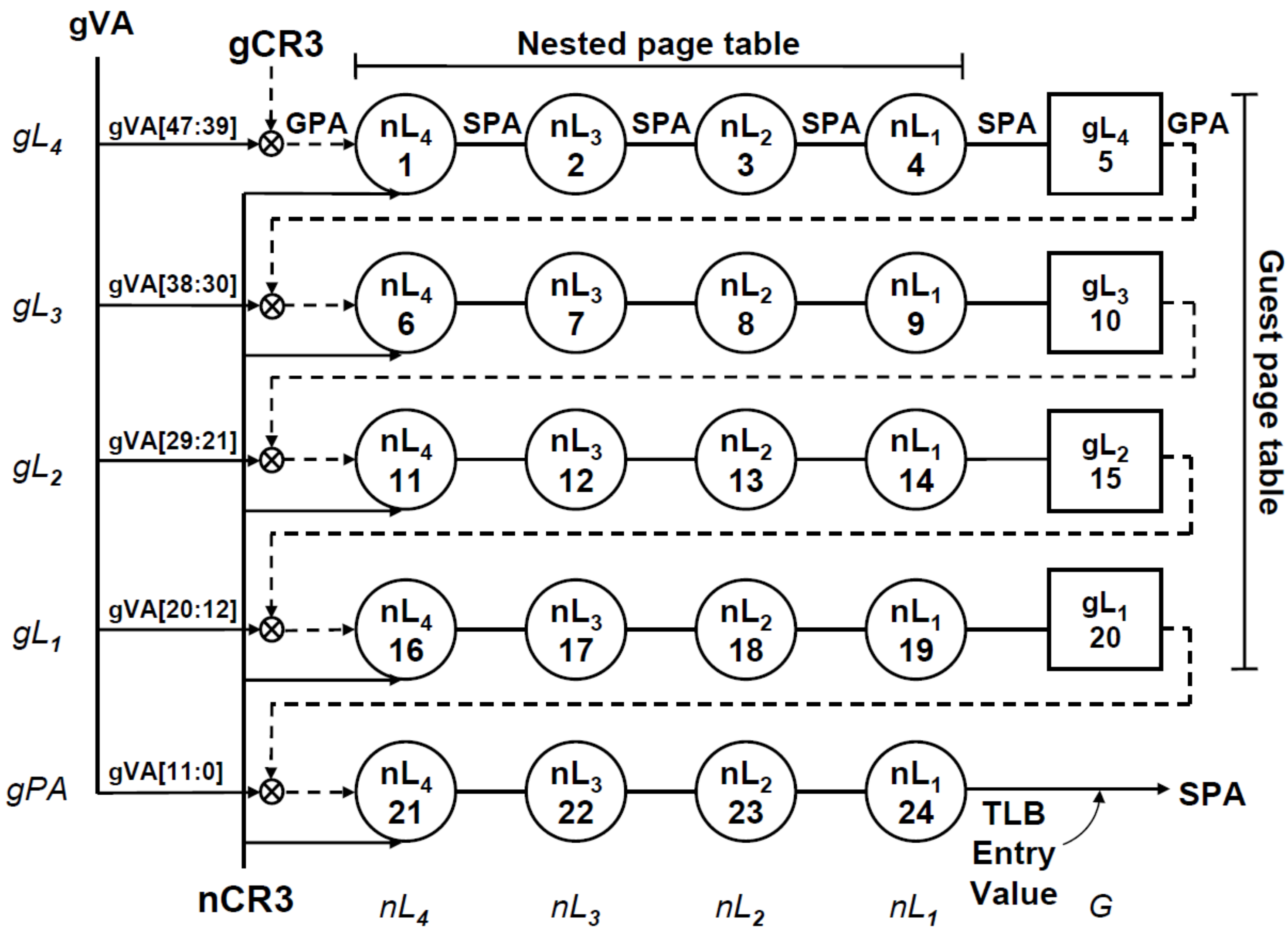


Page table lookup



- 4-level page table

Nested page table lookup



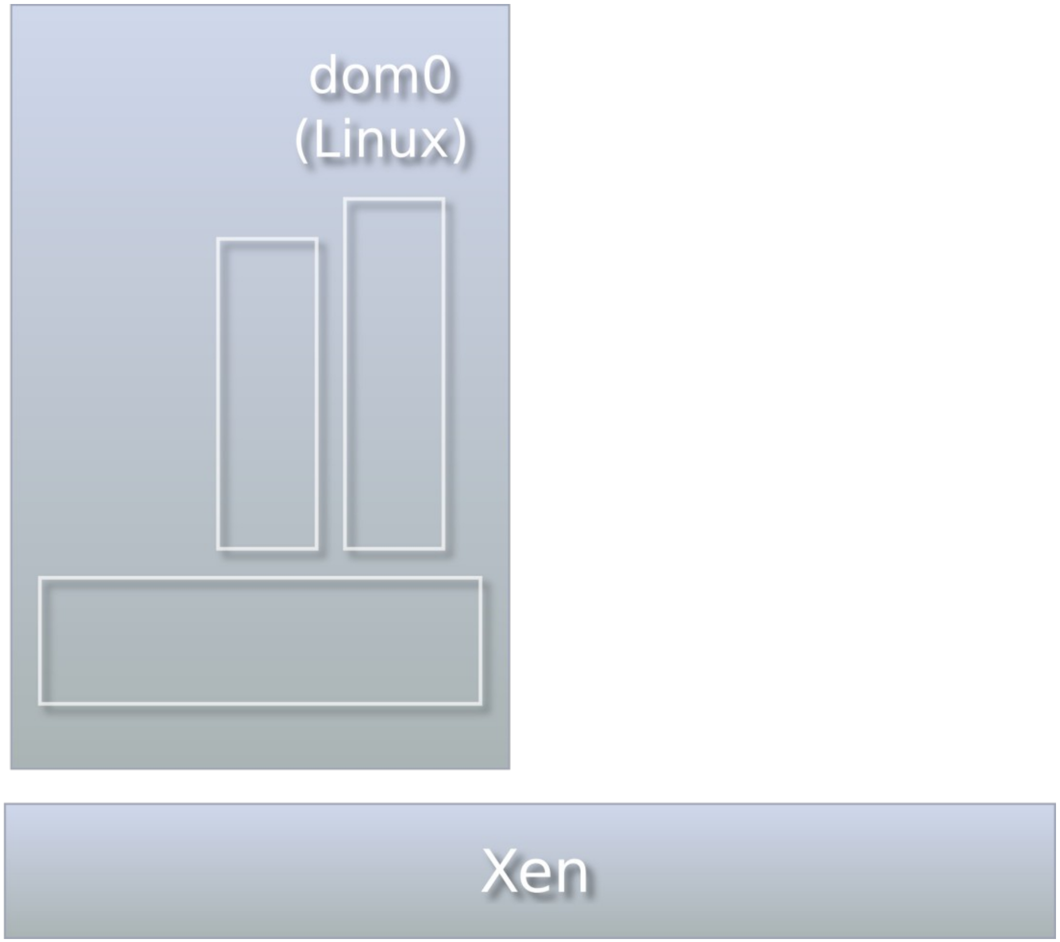
Efficient I/O

Where is the bottleneck

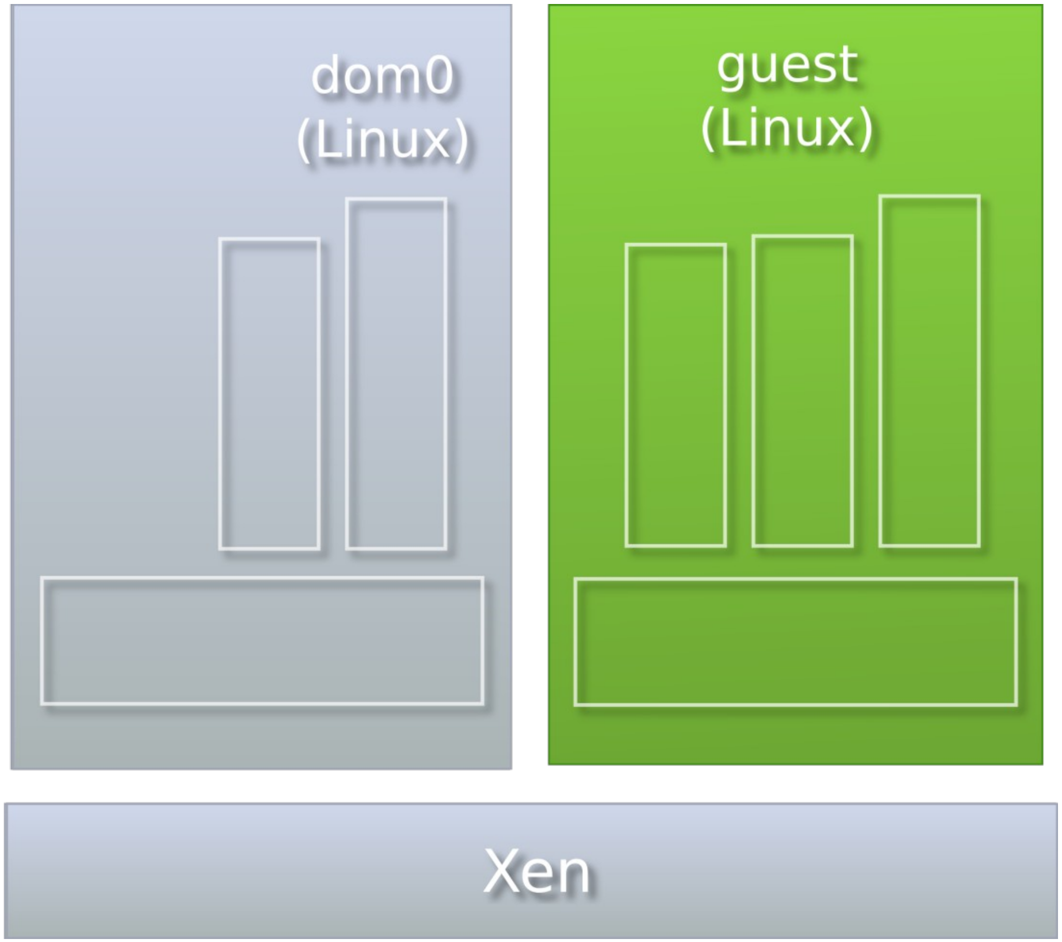
- What is the bottleneck in case of virtualization?
  - CPU?
    - CPU bound workloads execute natively on the real CPU
    - Sometimes JIT compilation (binary translation makes them even faster [Dynamo]
  - Everything what is inside VM is fast!
- What is the most frequent operation disturbing execution of VM?
- Device I/O!
  - Disk, Network, Graphics

Virtual devices in Xen

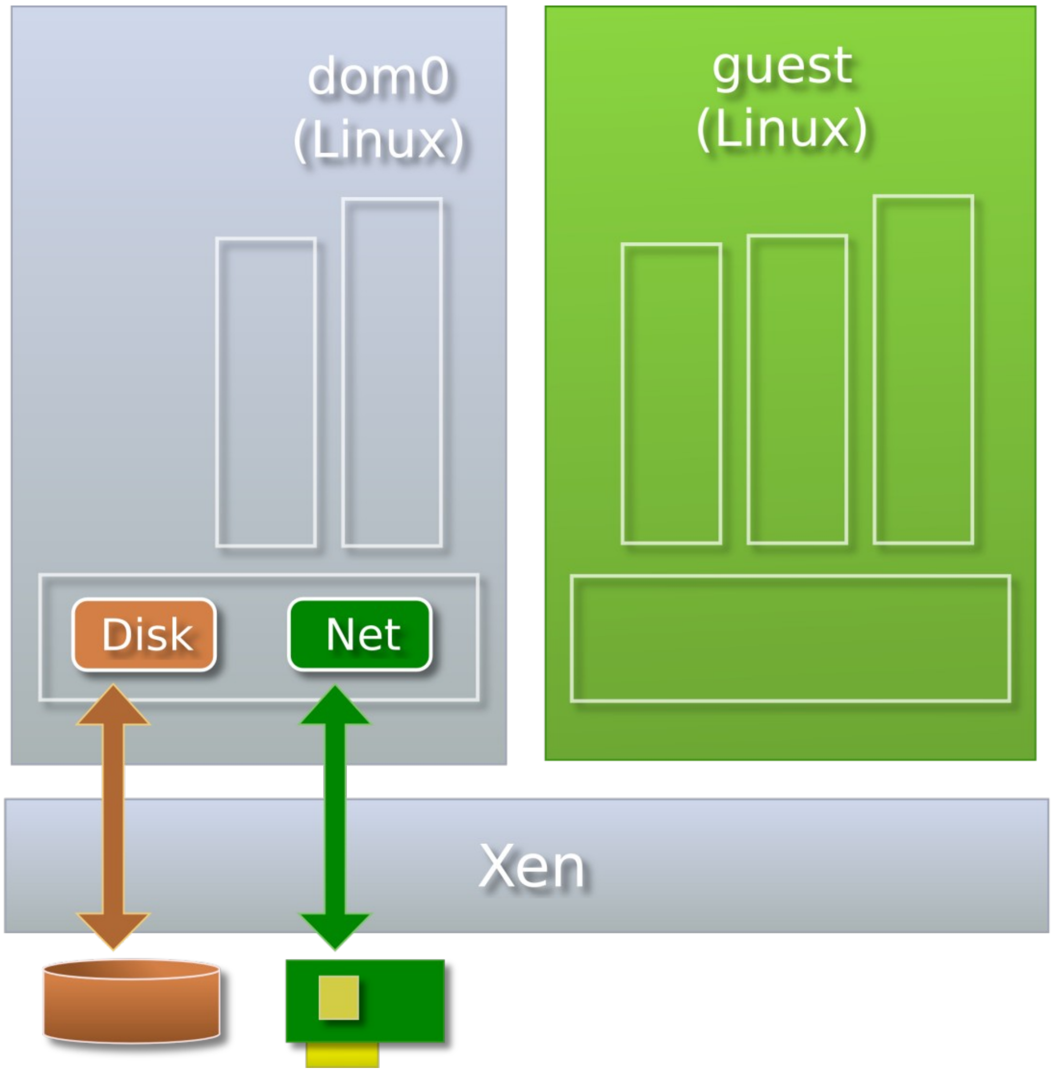
Virtual devices in Xen



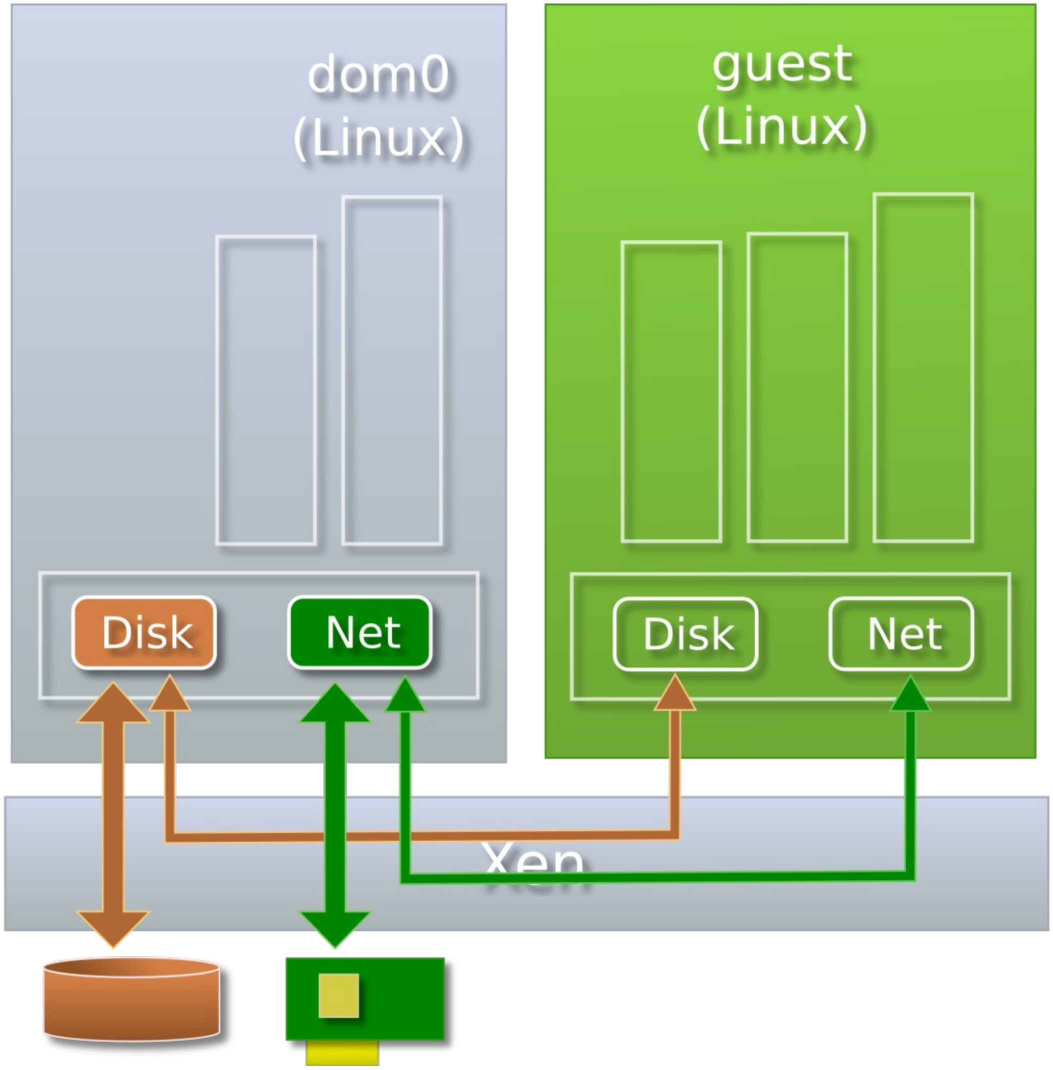
Virtual devices in Xen



Virtual devices in Xen



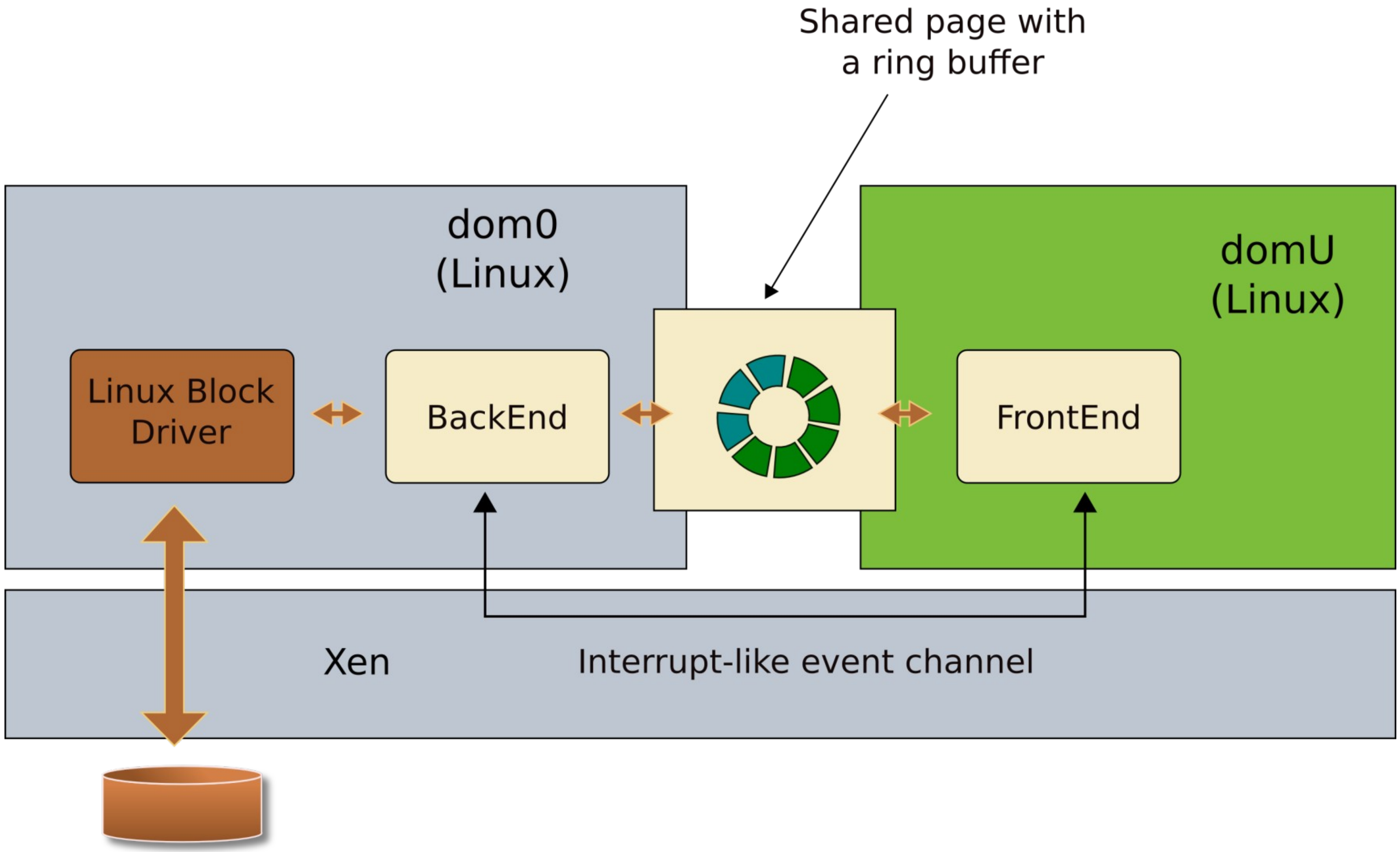
Virtual devices in Xen



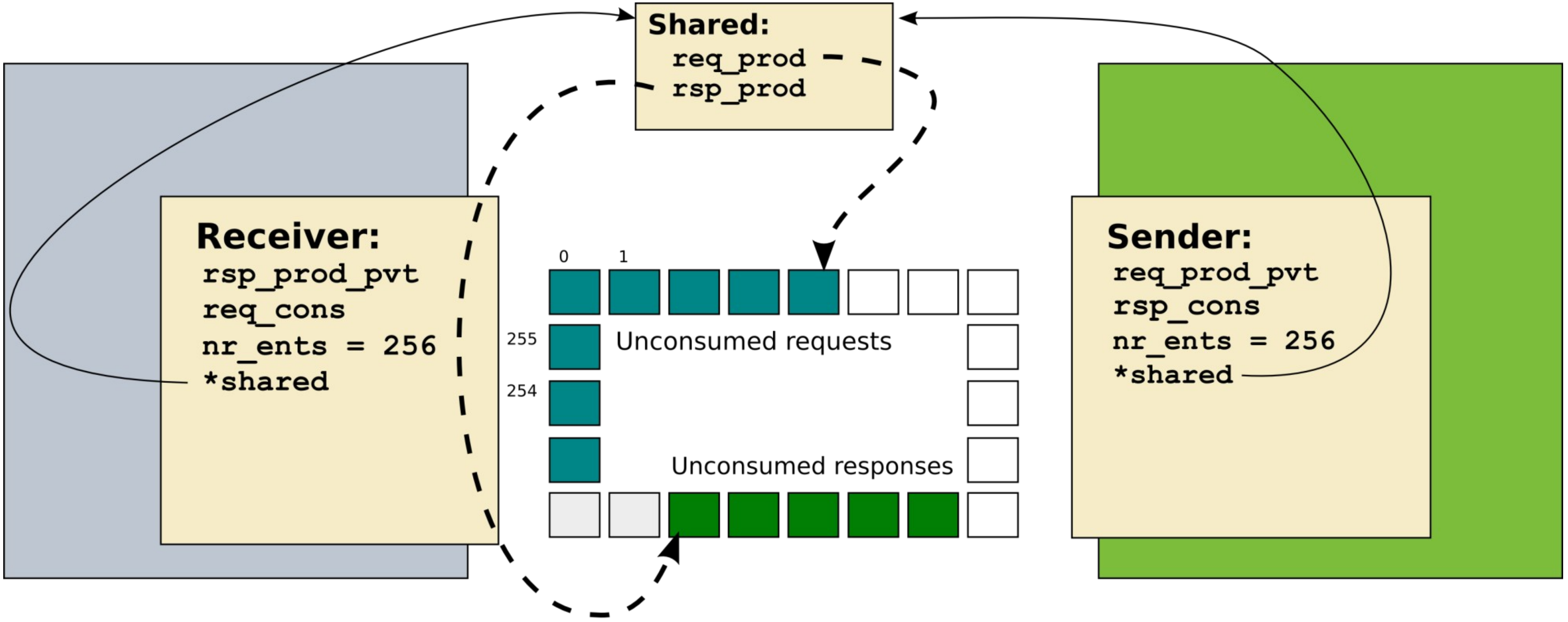
How to make the I/O fast?

- Take into account specifics of the device-driver communication
  - Bulk
    - Large packets (512B – 4K)
  - Session oriented
    - Connection is established once (during boot)
    - No short IPCs, like function calls
    - Costs of establishing an IPC channel are irrelevant
  - Throughput oriented
    - Devices have high delays anyway
  - Asynchronous
    - Again, no function calls, devices are already asynchronous

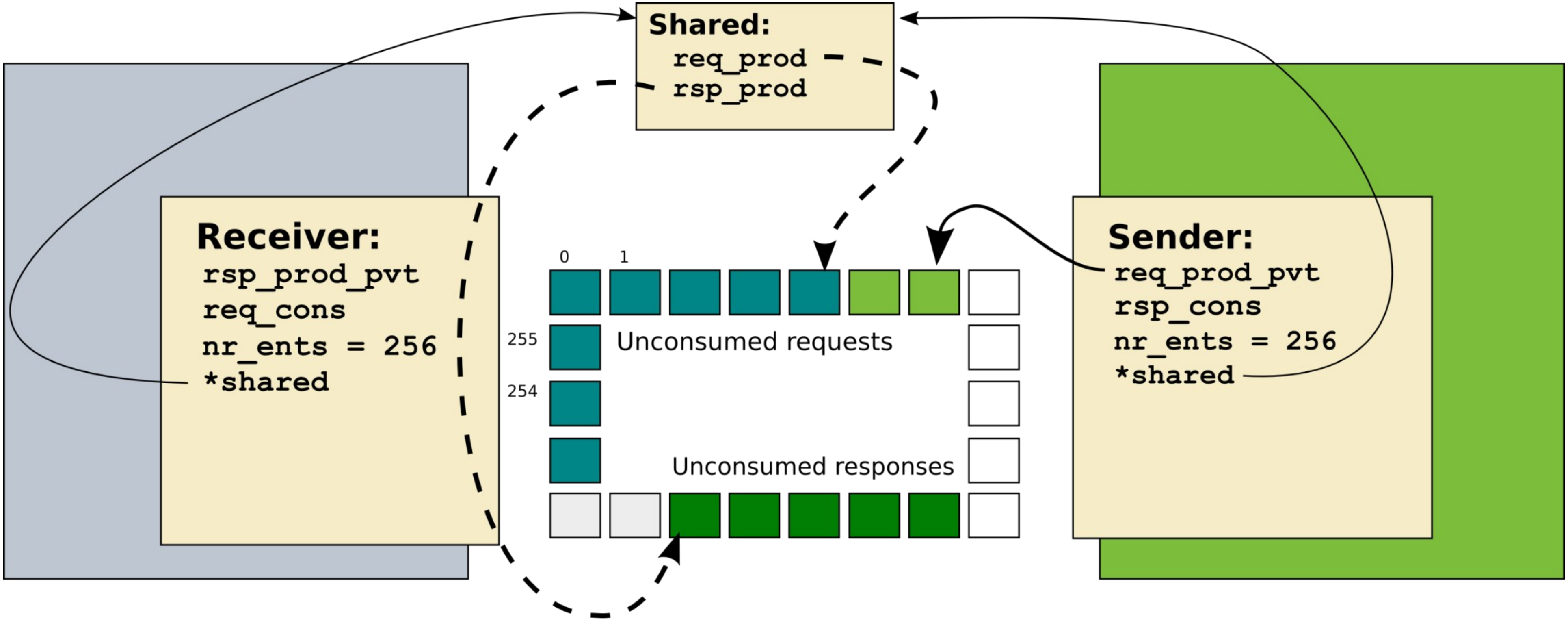
Shared rings and events



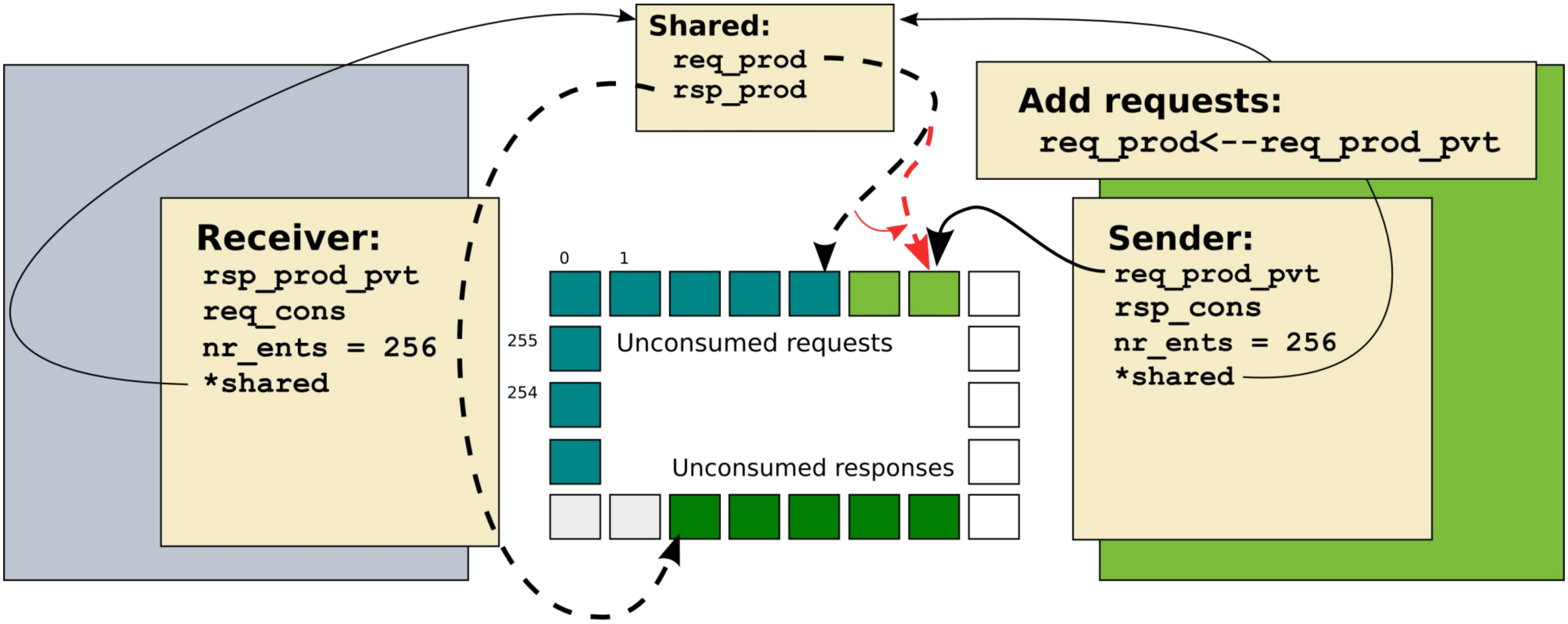
Shared rings



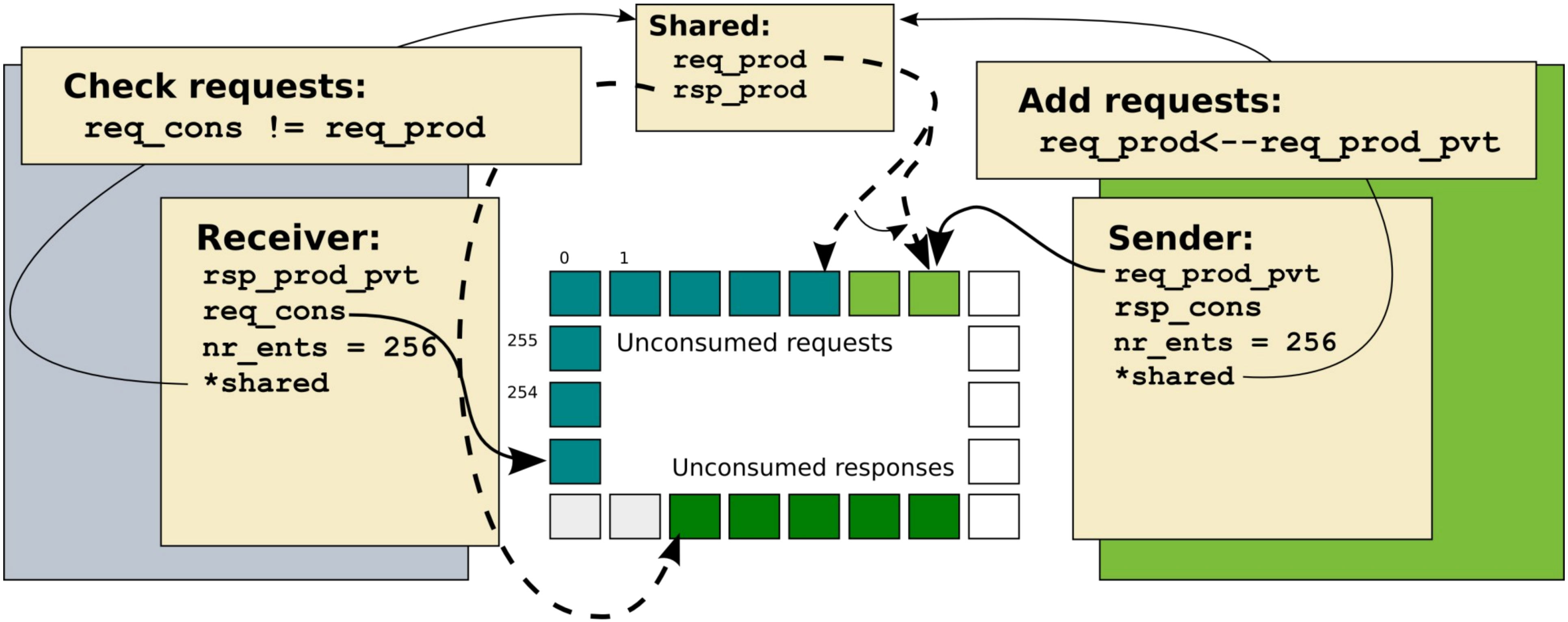
Shared rings



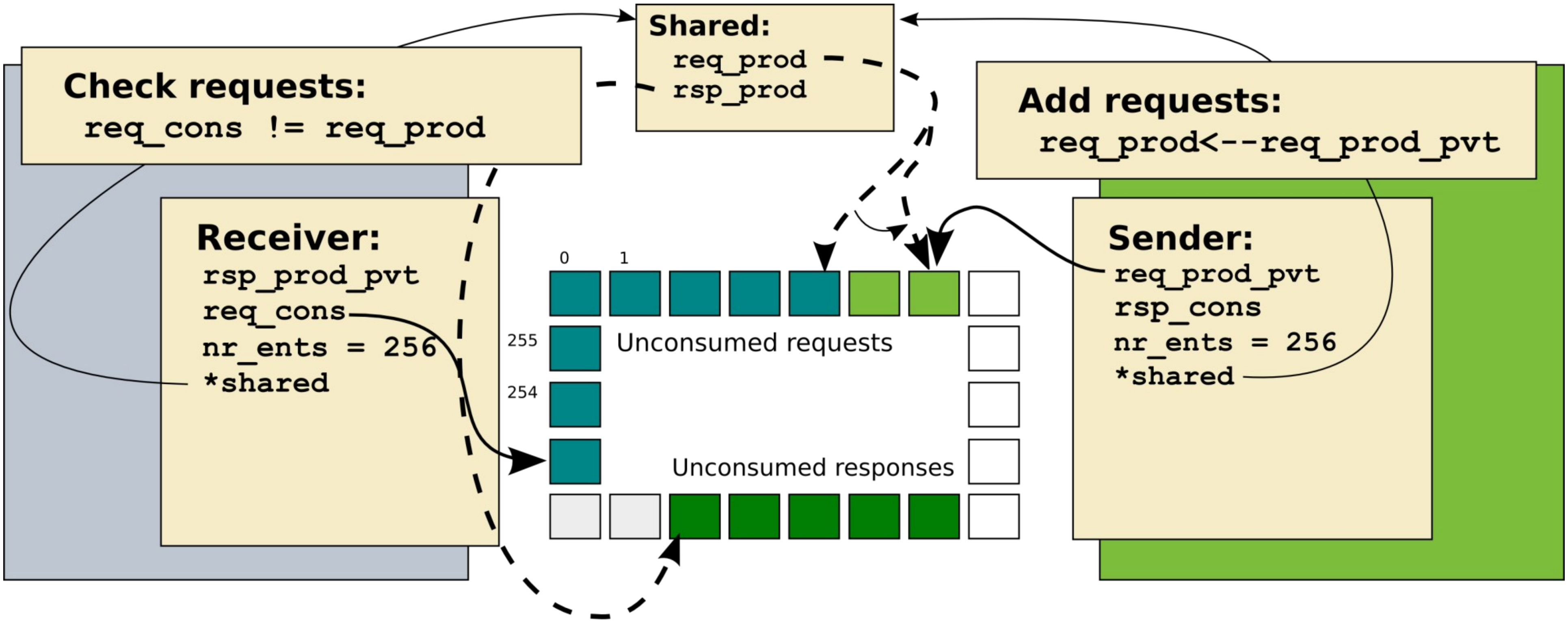
Shared rings



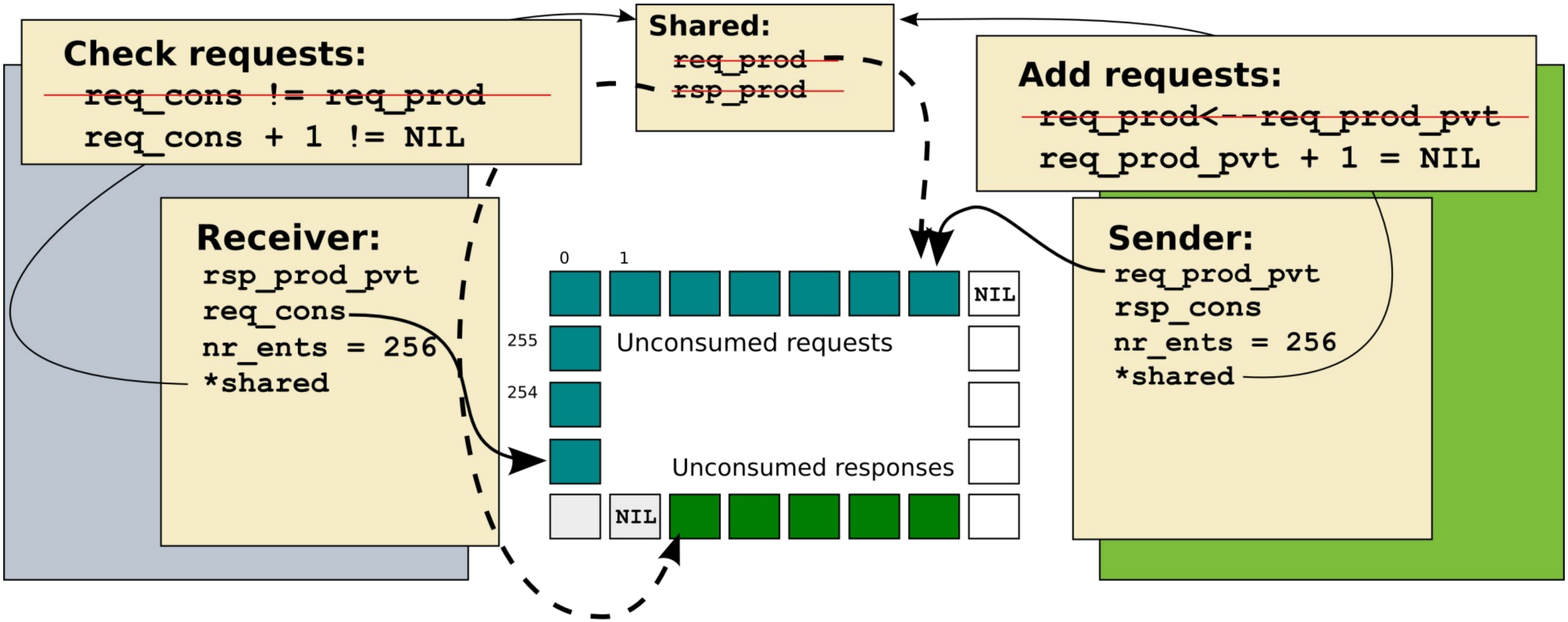
Shared rings



Where is a performance bottleneck here?



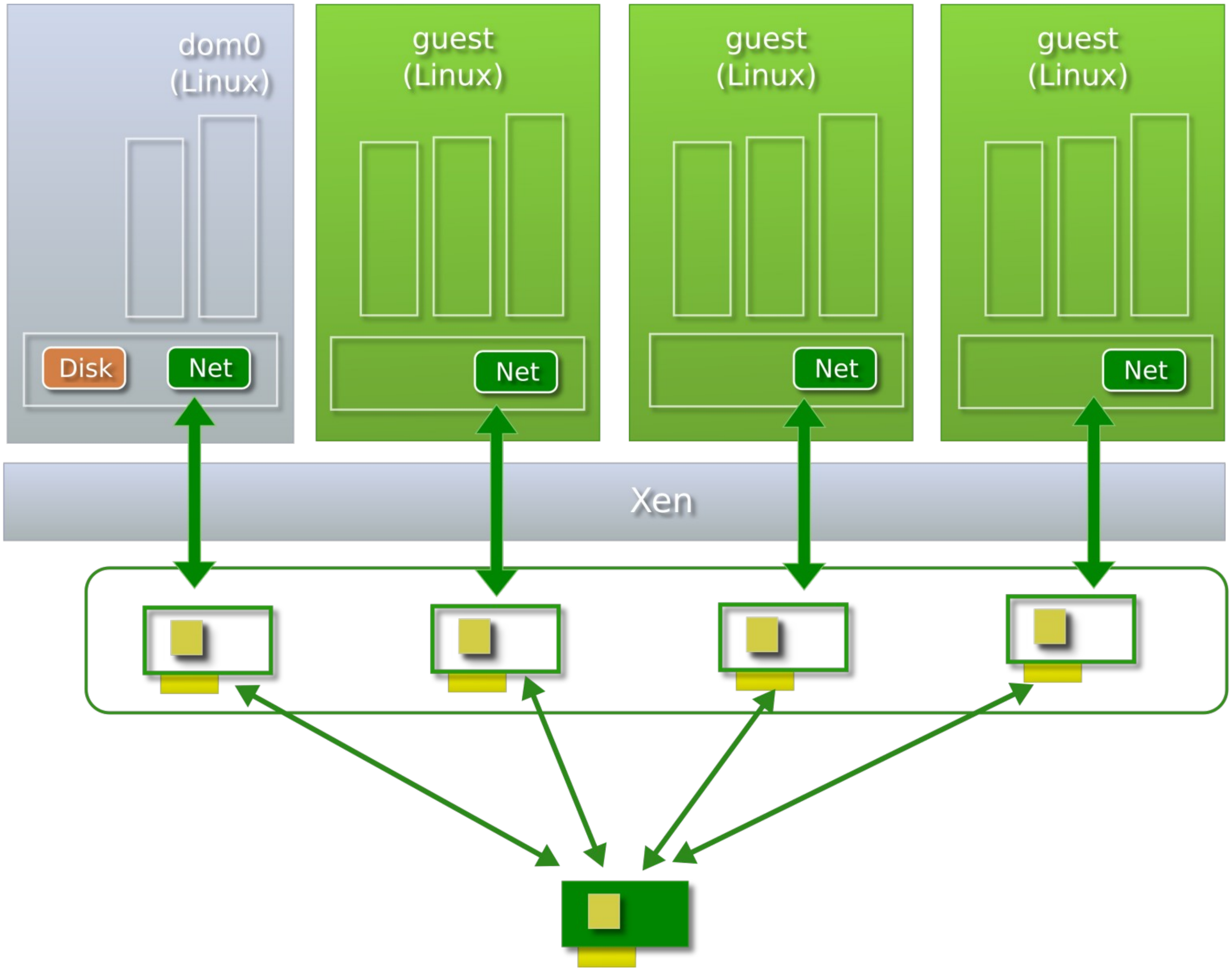
Eliminate cache thrashing



GPUs

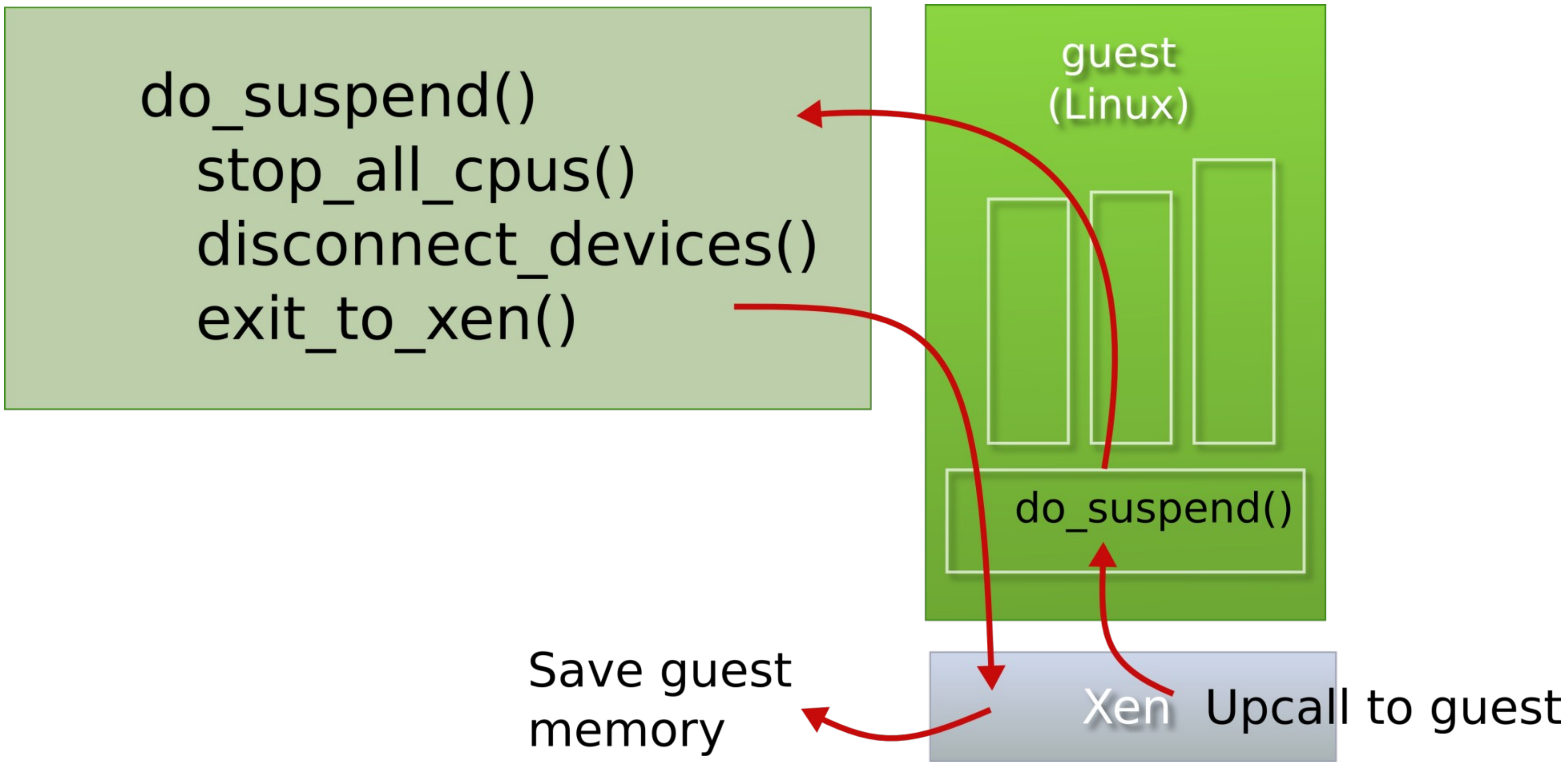
- Sending frames from the framebuffer
  - No hardware acceleration
  - Too slow
- OpenGL/DirectX level virtualization
  - Send high-level OpenGL commands over rings
  - OpenGL operations will be executed on the real GPU

Devices supporting virtualization

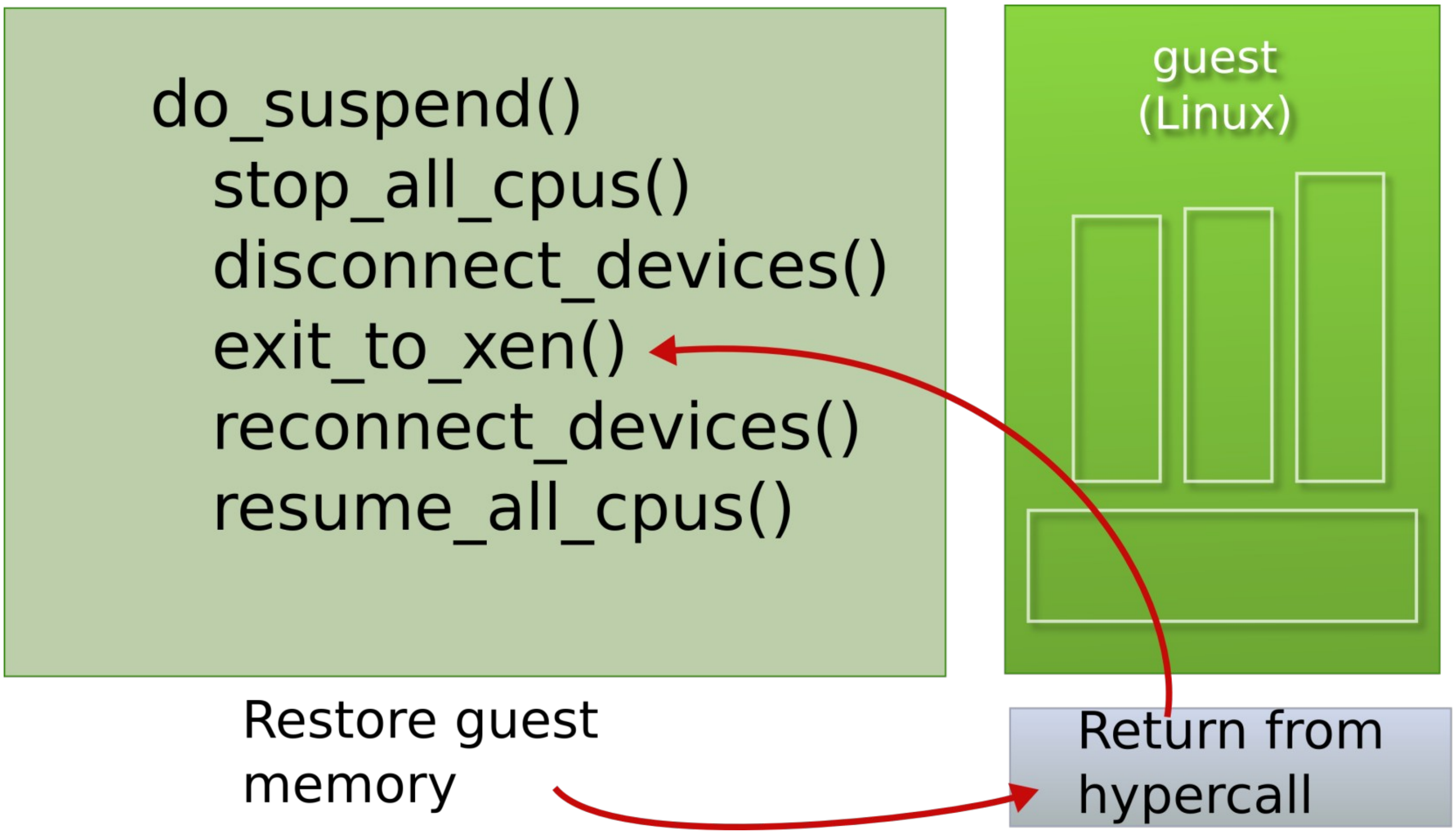


Some VM tricks: suspend/resume, checkpoints migration

Suspend


do_suspend()
stop_all_cpus()
disconnect_devices()
exit_to_xen()
guest
(Linux)
do_suspend()
Save guest
memory
Xen
Upcall to guest

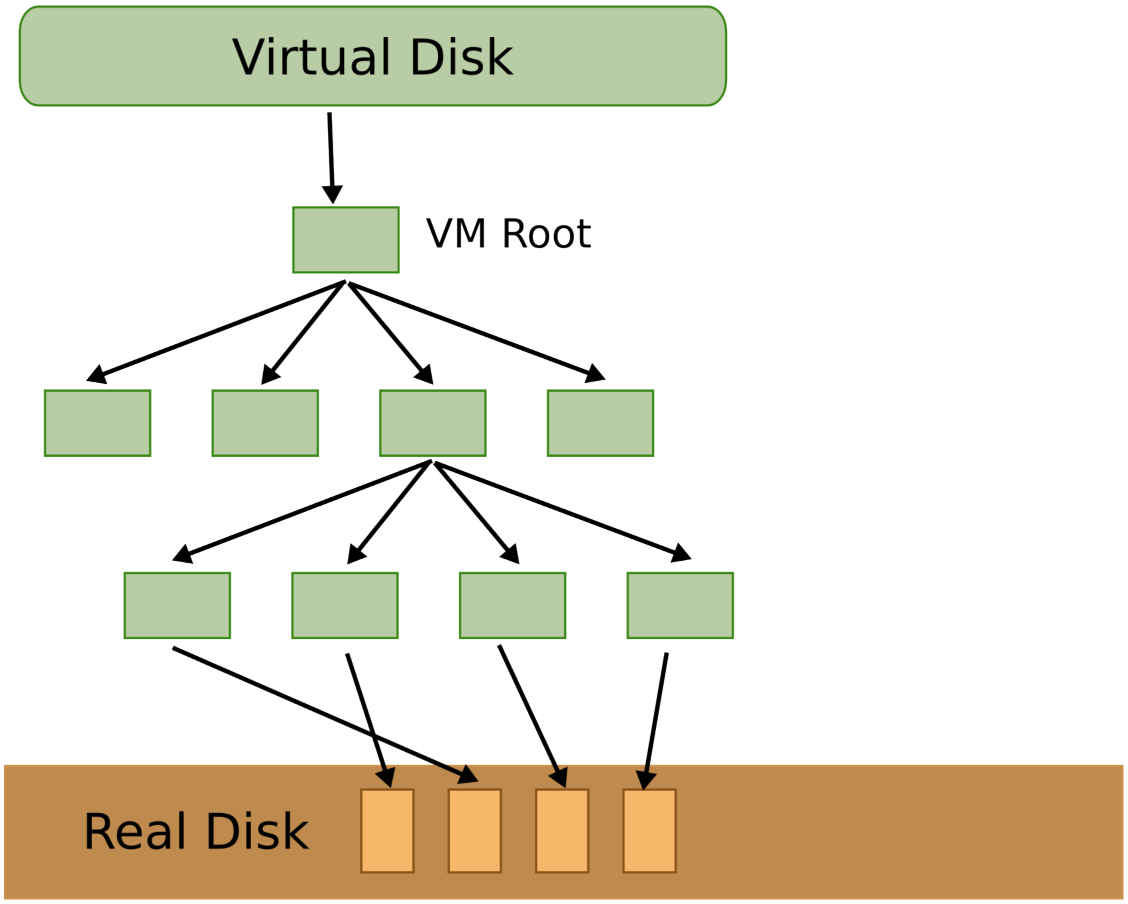
Resume


do_suspend()
stop_all_cpus()
disconnect_devices()
exit_to_xen()
reconnect_devices()
resume_all_cpus()
guest
(Linux)
Restore guest
memory
Return from
hypercall

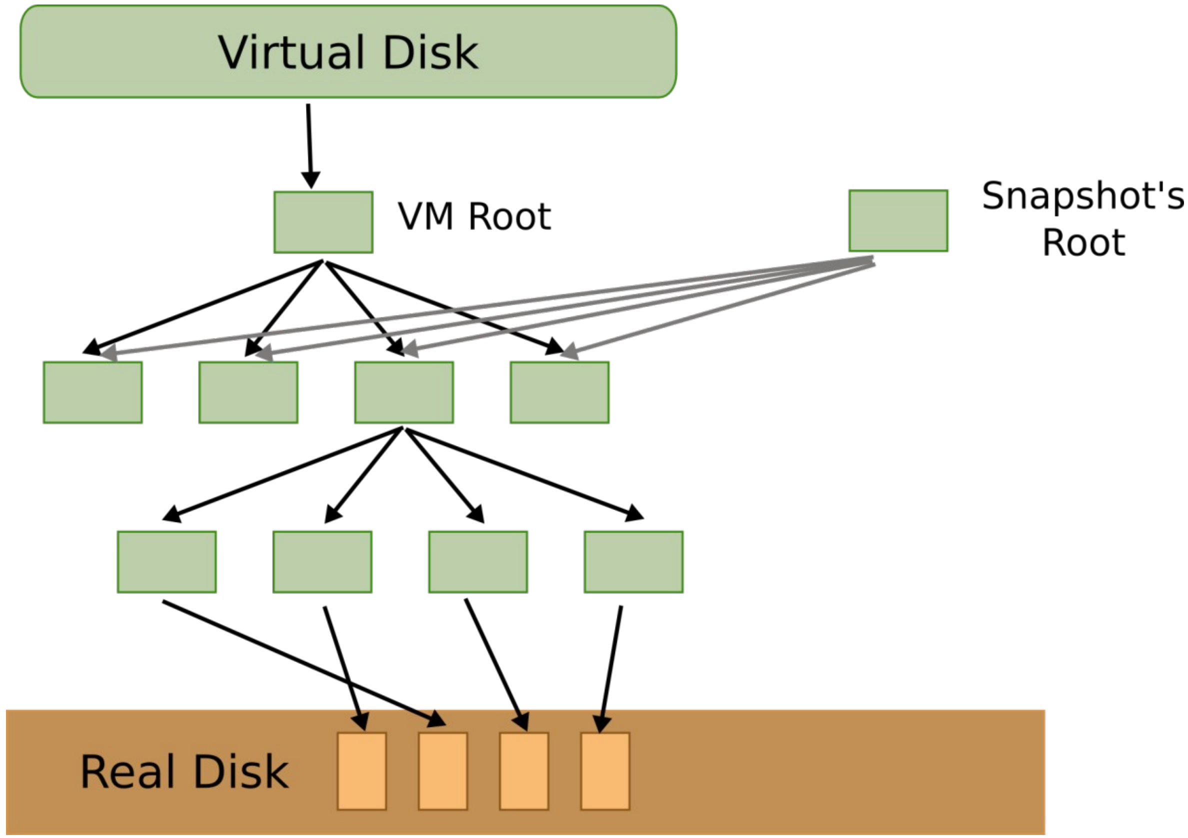
Checkpoints

- Checkpoints are almost suspend/resume
- Except that a copy of the entire VM's state has to be saved
  - Memory
    - OK, it's relatively small 128MB-4GB
  - Disk
    - Problem: disks are huge 100GB-1TB
- How to save storage efficiently?

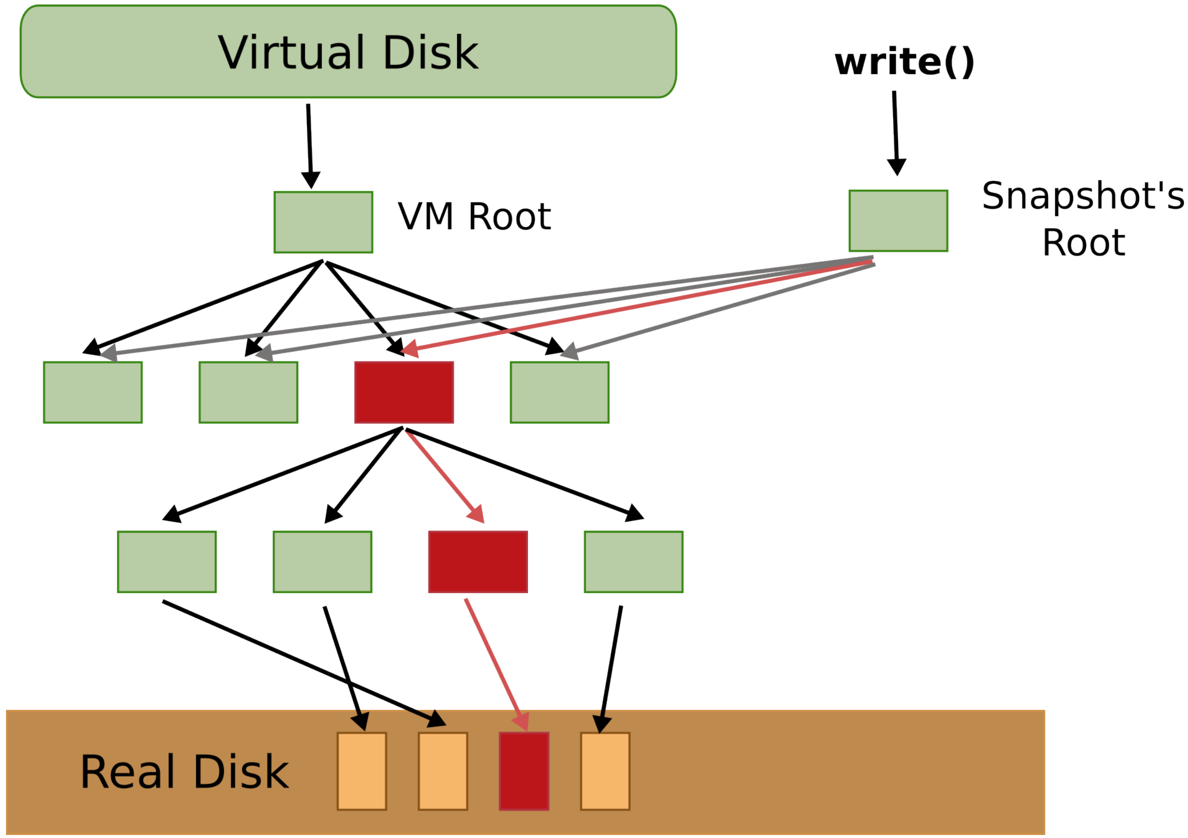
Branching storage



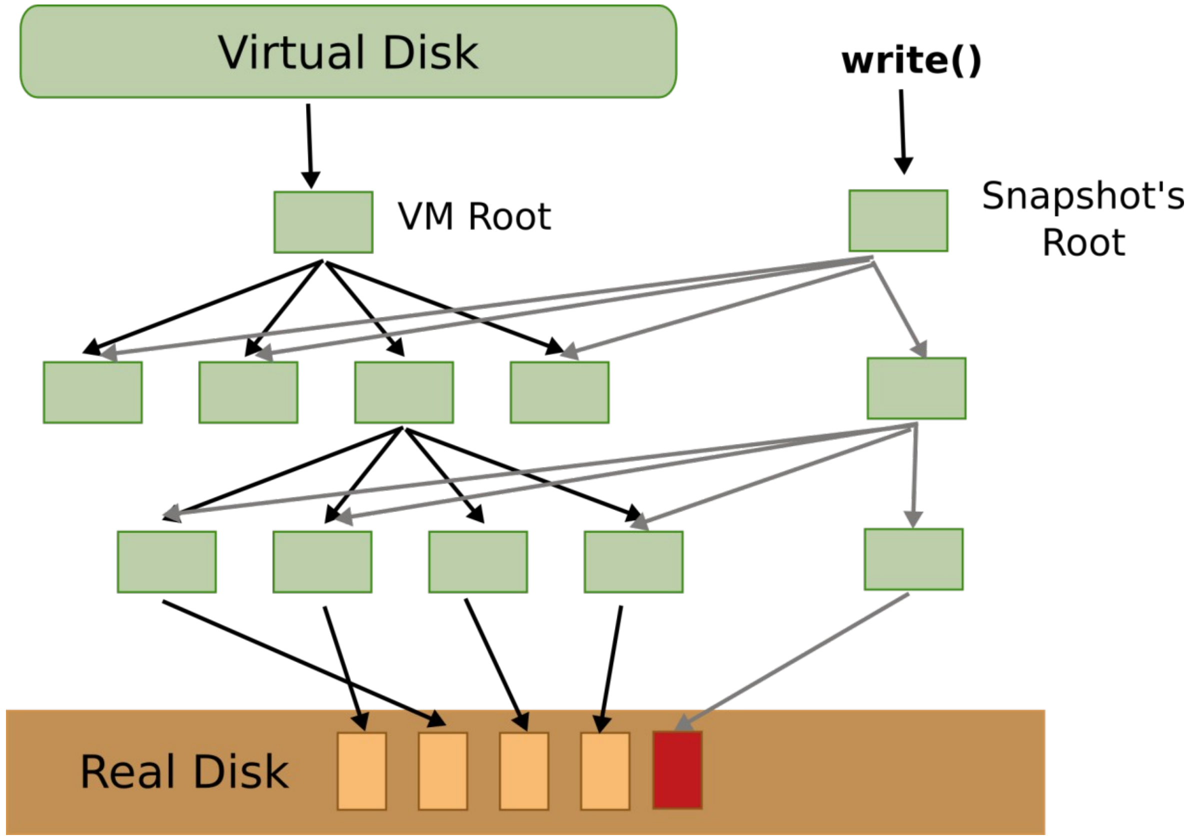
Branching storage: snapshot



Branching storage: writes



Branching storage: snapshot

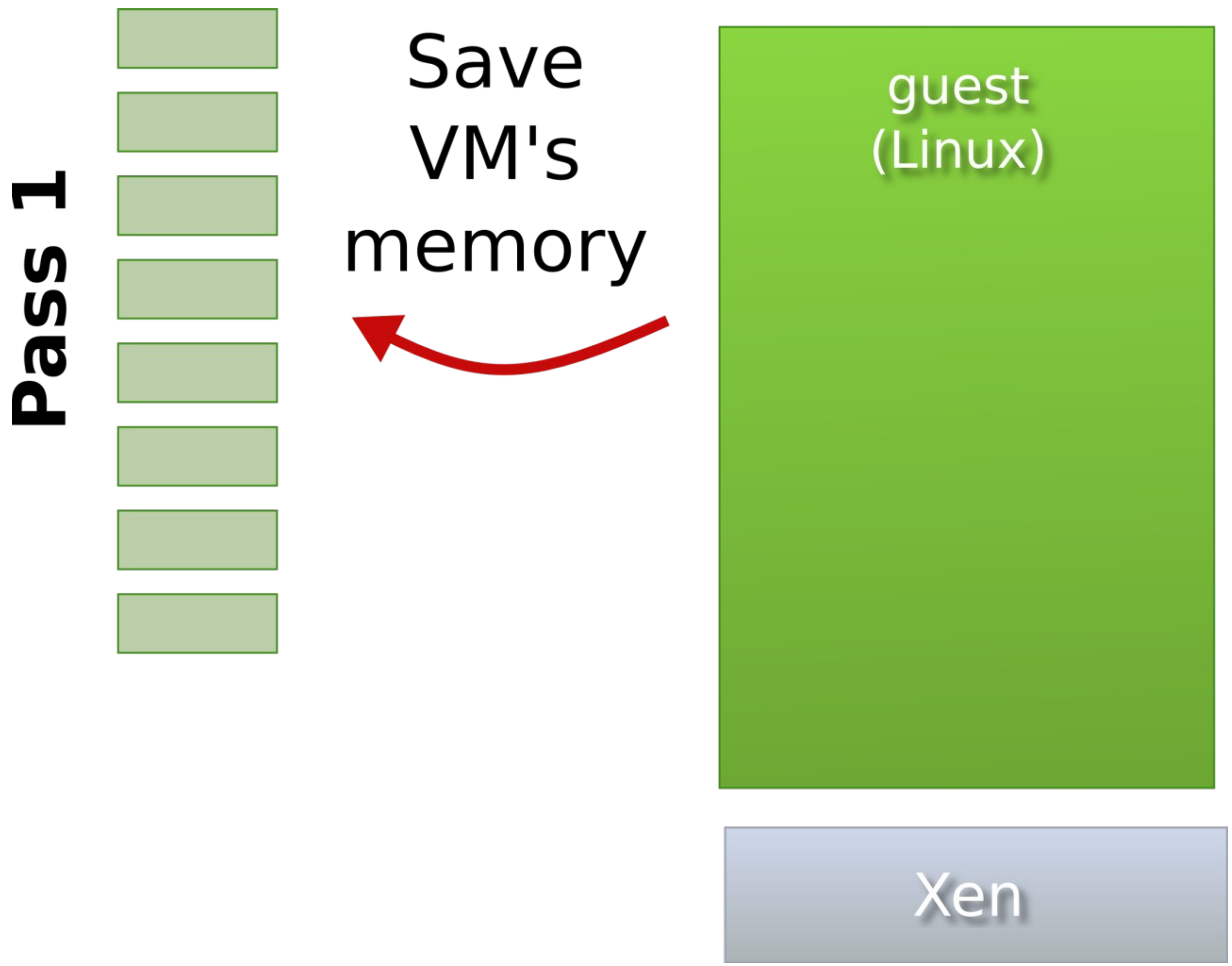


Migration

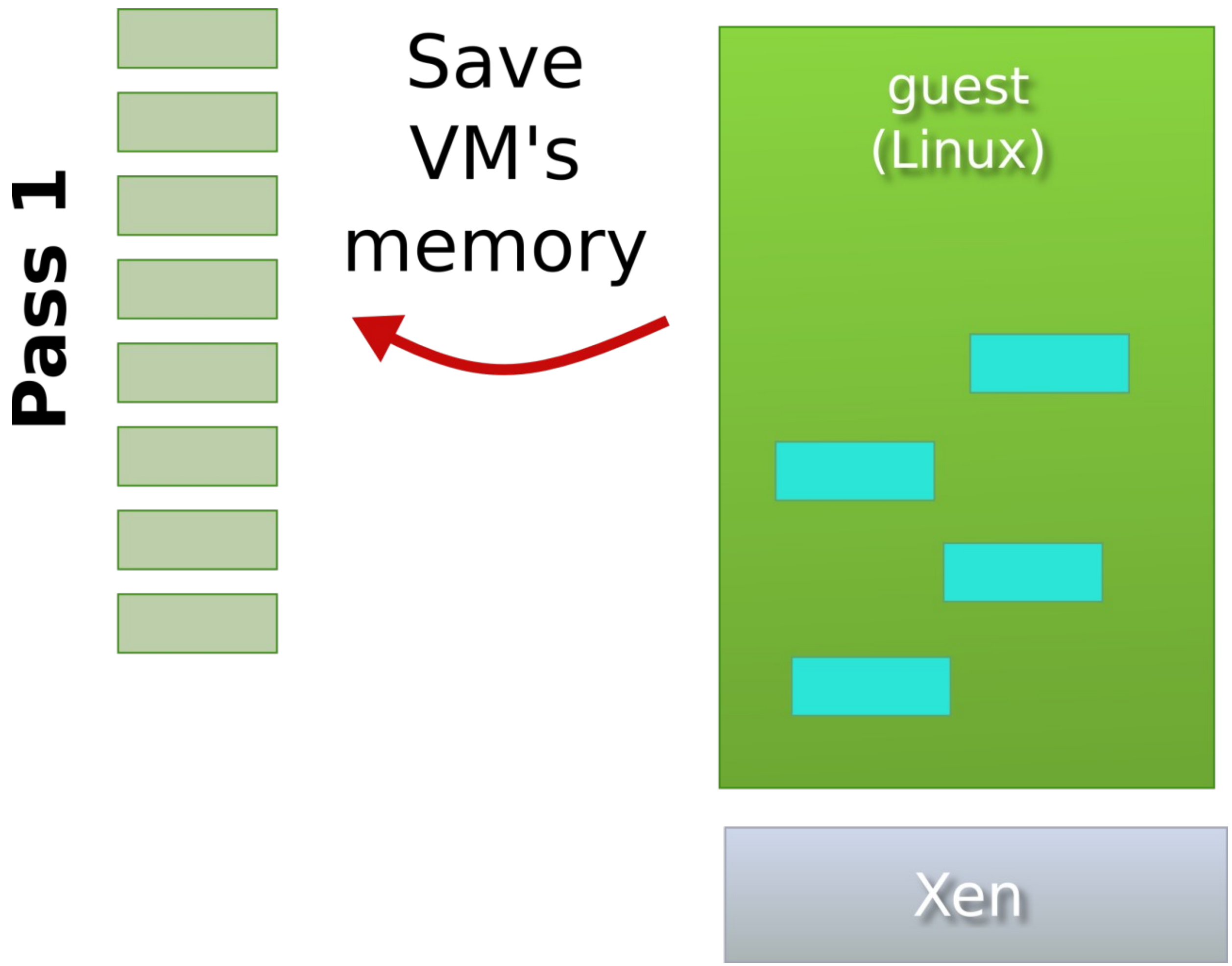
- Migration is essentially a live checkpoint between machines
- The goal: minimal downtime

- How to make the checkpoint faster?

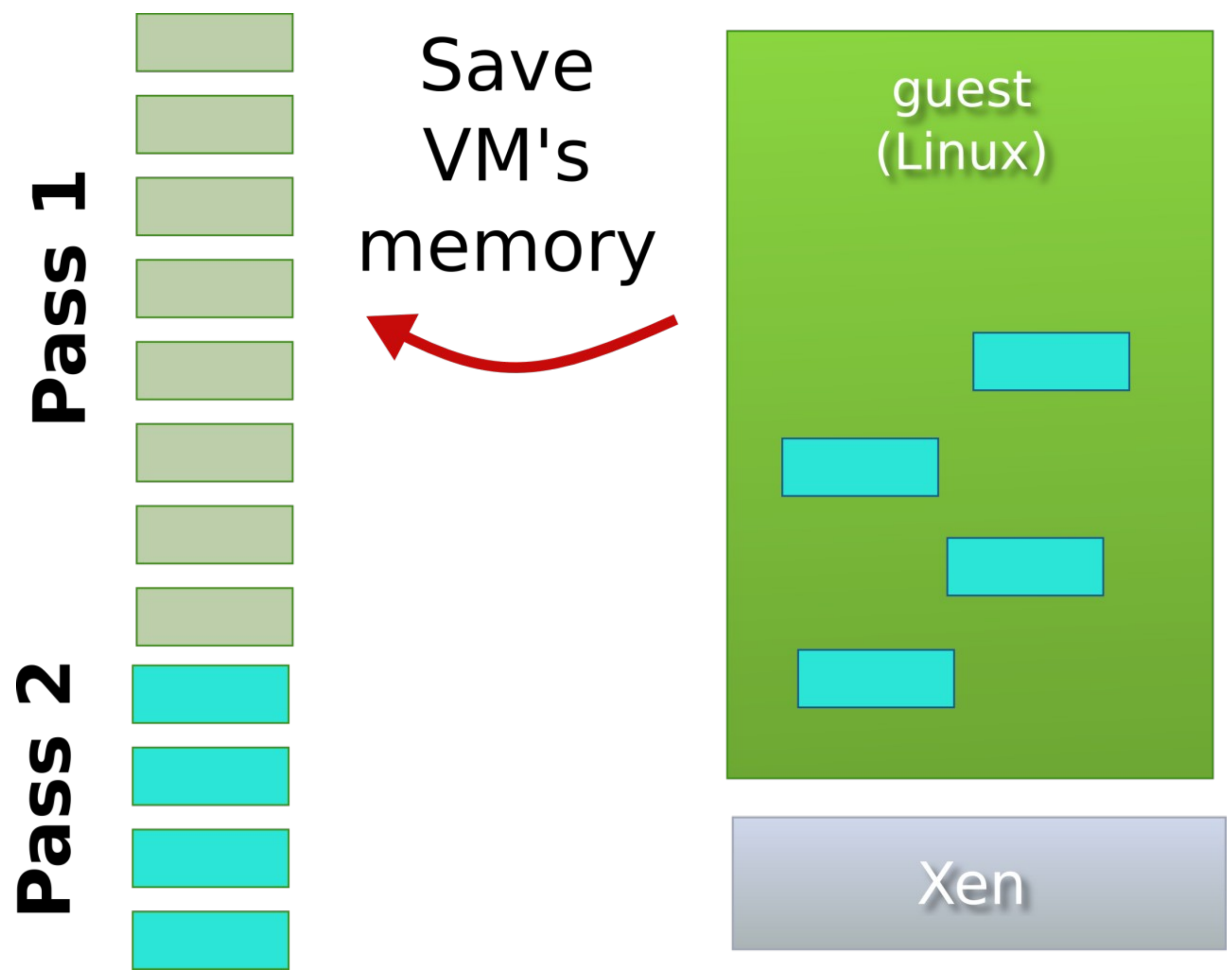
Migration: memory



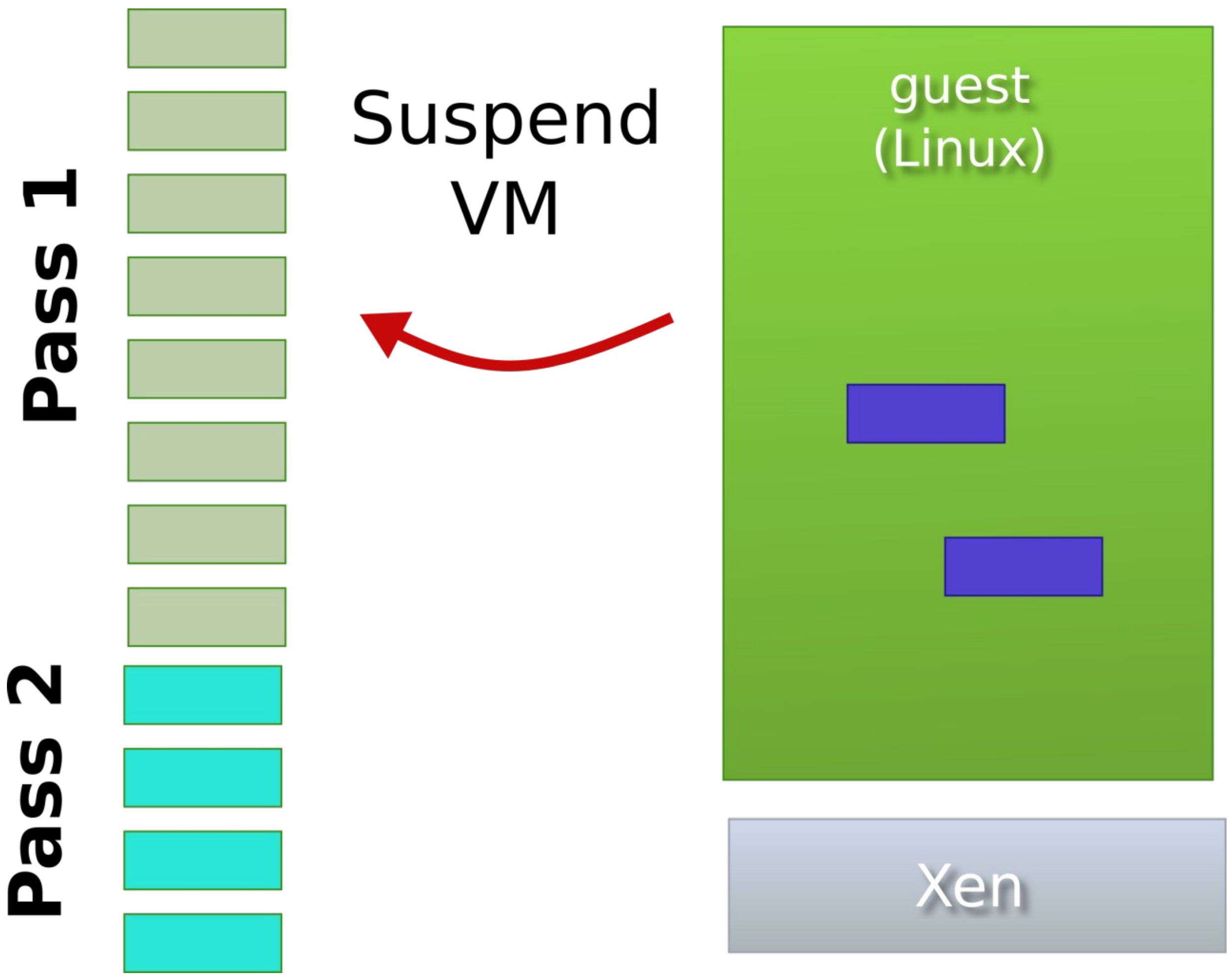
Migration: memory



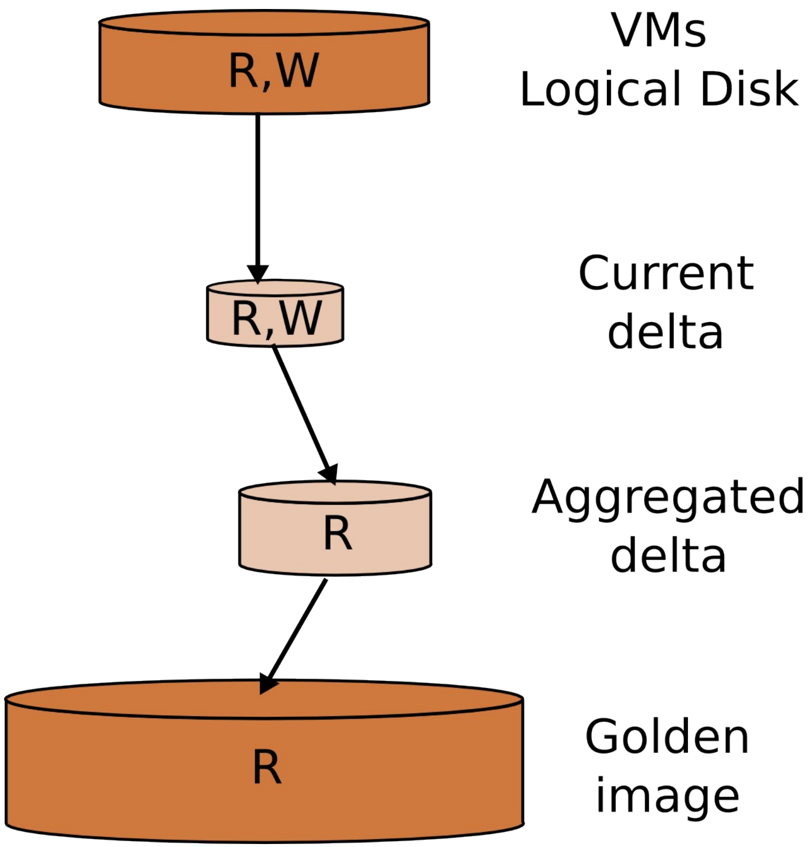
Migration: memory



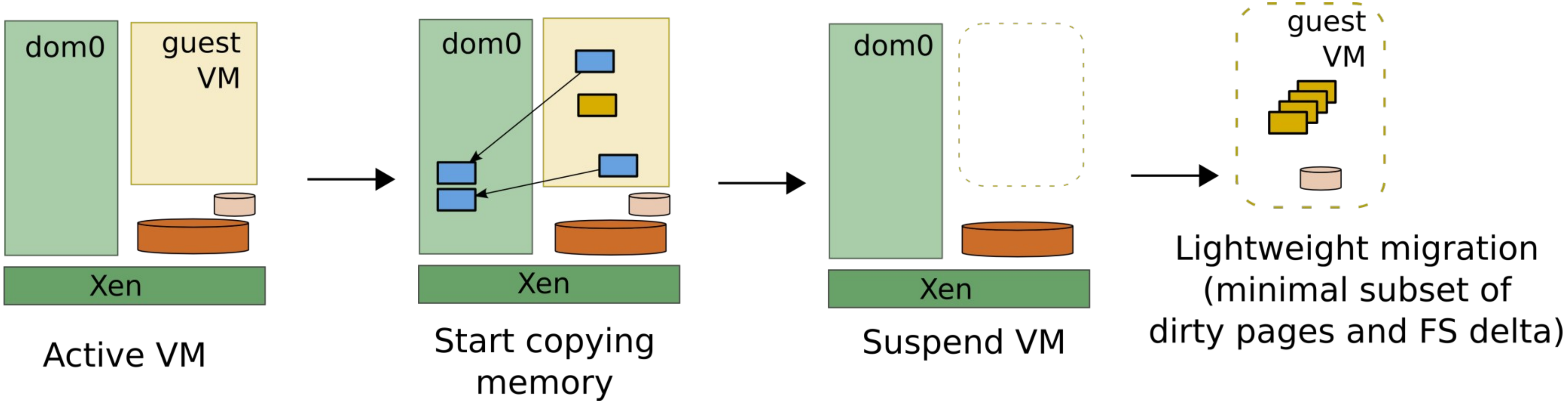
Migration: memory



Migration: storage



Migration


dom0
guest VM
Xen
Active VM
dom0
Xen
Start copying memory
dom0
Xen
Suspend VM
guest VM
Lightweight migration (minimal subset of dirty pages and FS delta)

References

- Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3C: System Programming Guide, Part 3
- Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. Accelerating two-dimensional page walks for virtualized systems. In ASPLOS'08.