Virtualizing Memory

Yiying Zhang

Outline

- Software-based memory virtualization
- Hardware-assisted memory virtualization
- Memory management
  - Reclaiming
  - Sharing
  - Allocation

Acknowledgment: some slides from Carl Waldspurger's OSDI'02 presentation

Lec2: Virtualizing Memory

- Extra level of indirection

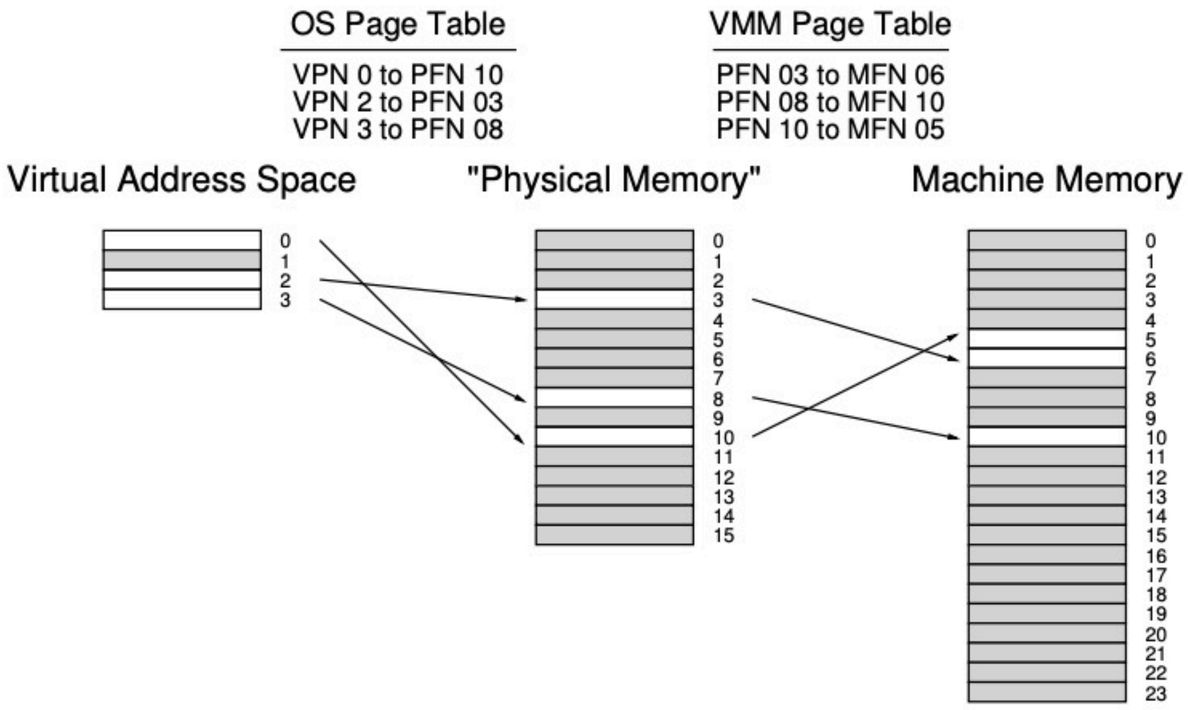
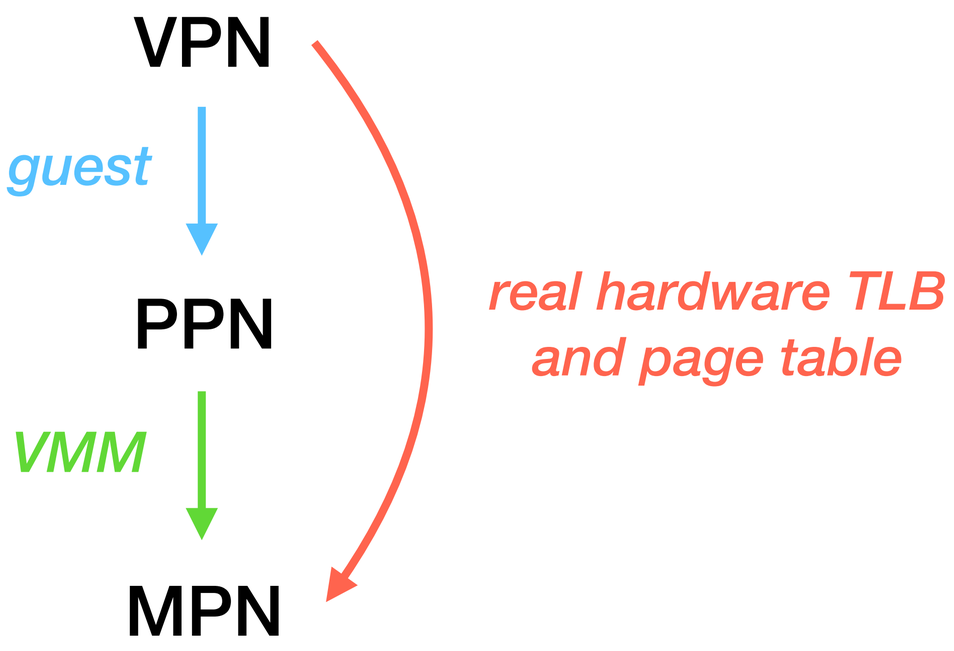


Figure B.4: **VMM Memory Virtualization**



Lec2: Virtualizing Memory

- TLB miss flow with software-managed TLB

Process	Operating System
1. Load from memory: TLB miss: Trap	
	2. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid: get PFN, update TLB; Return from trap
3. Resume execution (@PC of trapping instruction); Instruction is retried; Results in TLB hit	

Figure B.5: **TLB Miss Flow without Virtualization**

Process	Operating System	Virtual Machine Monitor
1. Load from mem TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap
7. Resume execution (@PC of instruction); Instruction is retried; Results in TLB hit		

Figure B.6: **TLB Miss Flow with Virtualization**

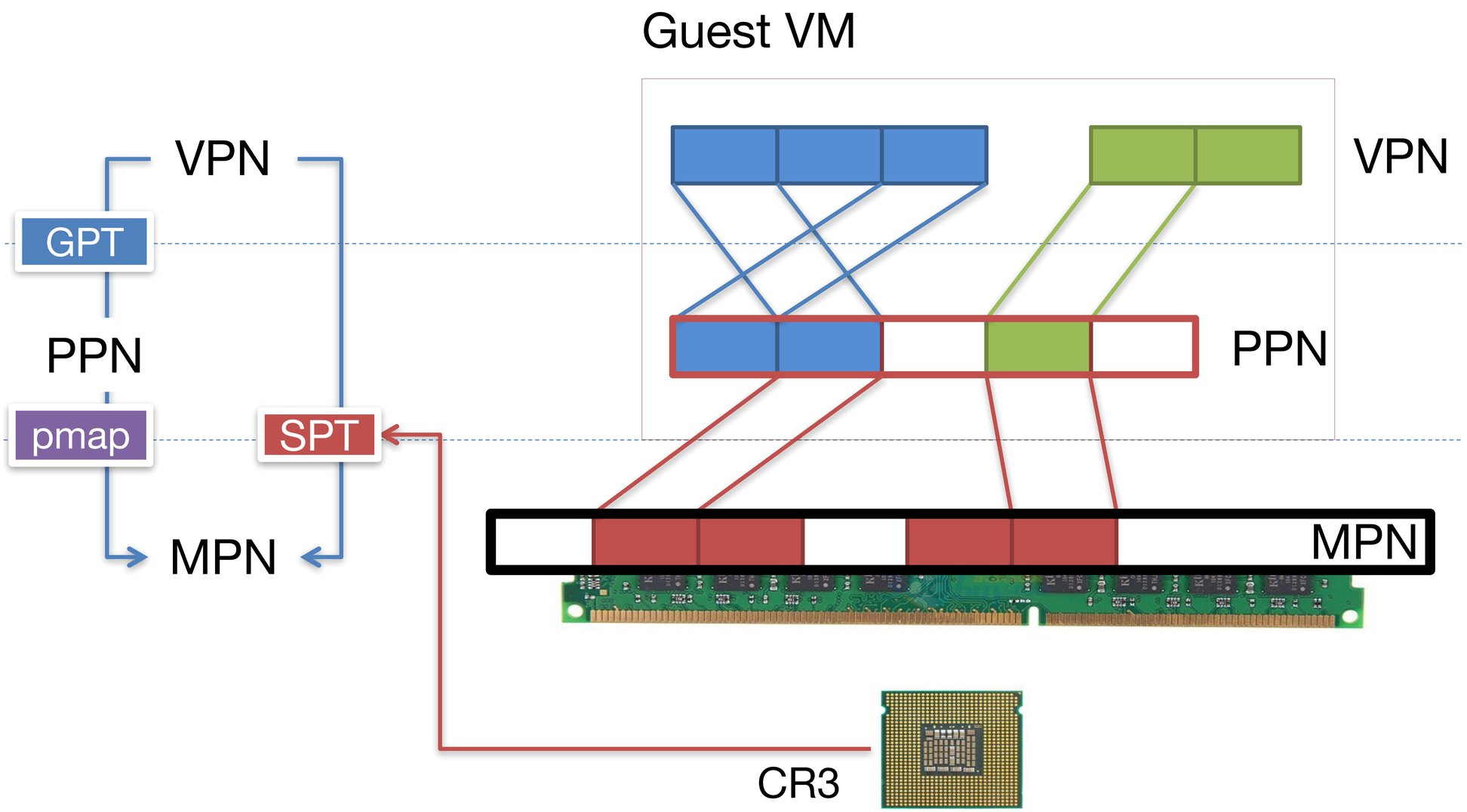
Lec2: Difficulty in Virtualizing Hardware-Managed TLB

- Hardware-managed TLB
  - Hardware does page table walk on each TLB miss
  - and fills TLB with the found PTE

- Hypervisor doesn't have chance to intercept on TLB misses

- Solution-1: **shadow paging**
- Solution-2: **direct paging (para-virtualization)** *later this quarter*
- Solution-3: **new hardware**

Shadow Paging



Set Up Shadow Page Table

1. VMM intercepts guest OS setting the virtual CR3
2. VMM iterates over the guest page table, constructs a corresponding shadow page table
3. In shadow PT, every guest physical address is translated into host physical address (machine address)
4. Finally, VMM sets the real CR3 to point to the shadow page table

Set Up Shadow Page Table

```
set_cr3 (guest_page_table):
    for VPN in 0 to 220
        if guest_page_table[VPN] & PTE_P /* PTE_P: valid bit */
            PPN = guest_page_table[VPN]
            MPN = pmap[PPN]
            shadow_page_table[VPN] = MPN | PTE_P
        else
            shadow_page_table = 0
    CR3 = PHYSICAL_ADDR(shadow_page_table)
```

Question

- Assume that:
  - There are 10 VMs running on a machine
  - Each VM contains 10 applications
- **Q**: How many shadow page tables in total?
  - Shadow page tables are per application
  - Guest page tables are per application
  - pmaps are per VM

What if Guest OS Modifies Its Page Table?

- Should not allow it to happen directly
  - Since CR3 is now pointing to the shadow page table
  - Need to synchronize the shadow page table with guest page table

- VMM needs to intercept when guest OS modifies page table, and updates the shadow page table accordingly

  1. Mark the guest table pages as read-only (in the shadow page table)
  2. If guest OS tries to modify its page tables, it triggers page fault
  3. VMM handles the page fault by updating shadow page table

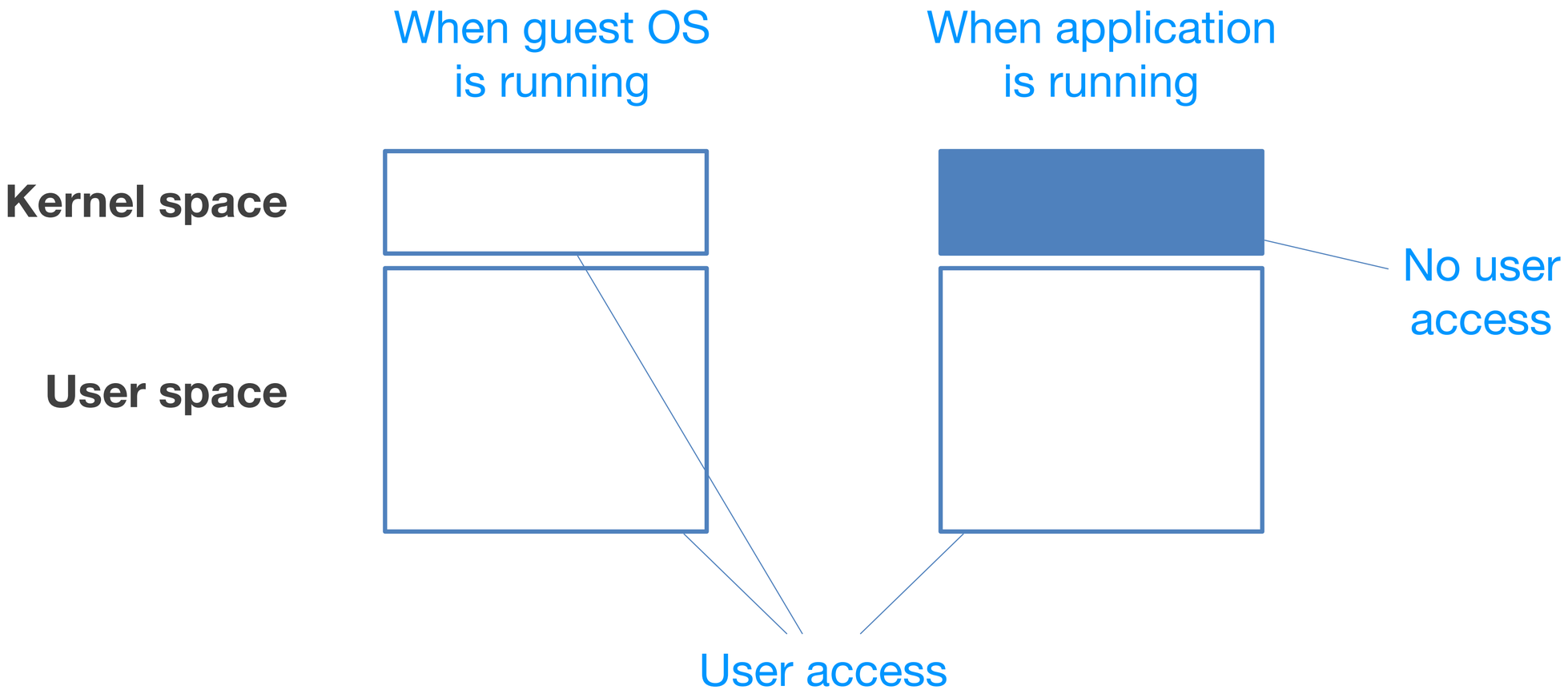
Dealing with Page Faults

- When page fault occurs, traps to VMM
- If present bit is 0 in the guest page table entry, guest OS needs to handle the fault
  - Guest OS load page from virtual disk to guest physical memory and sets present bit to 1
  - Guest OS returns from page fault, which traps into VMM again
  - VMM sees that present is 1 in guest PTE and creates entry in shadow page table
  - VMM returns from the original page fault
- If present is 1: guest OS thinks page is present (but VMM may have swapped it out), VMM handles transparently
  - VMM locates the corresponding physical page, loads it in memory if needed
  - VMM creates entry in shadow page table
  - VMM returns from the original page fault

What if a Guest App Access its Kernel Memory?

- How do we selectively allow / deny access to kernel-only pages?
- One solution: split a shadow page table into two tables
  - Two shadow page tables, one for user, one for kernel
  - When guest OS switches to guest applications, VMM will switch the shadow page table as well, vice versa

Two Memory Views of Guest VM



The Same Question

- Assume that:
  - There are 10 VMs running on a machine
  - Each VM contains 10 applications
- **Q**: Now, how many shadow page tables in total?

What about Memory for Translation Cache (BT)?

- Translation cache intermingles guest and monitor memory accesses
  - Need to distinguish these accesses
  - Monitor accesses have full privileges
  - Guest accesses have lesser privileges
- On x86 can use segmentation
  - Monitor lives in high memory
  - Guest segments truncated to allow no access to monitor
  - Binary translator uses guest segments for guest accesses and monitor segments for monitor accesses

Pros and Cons of Shadow Paging

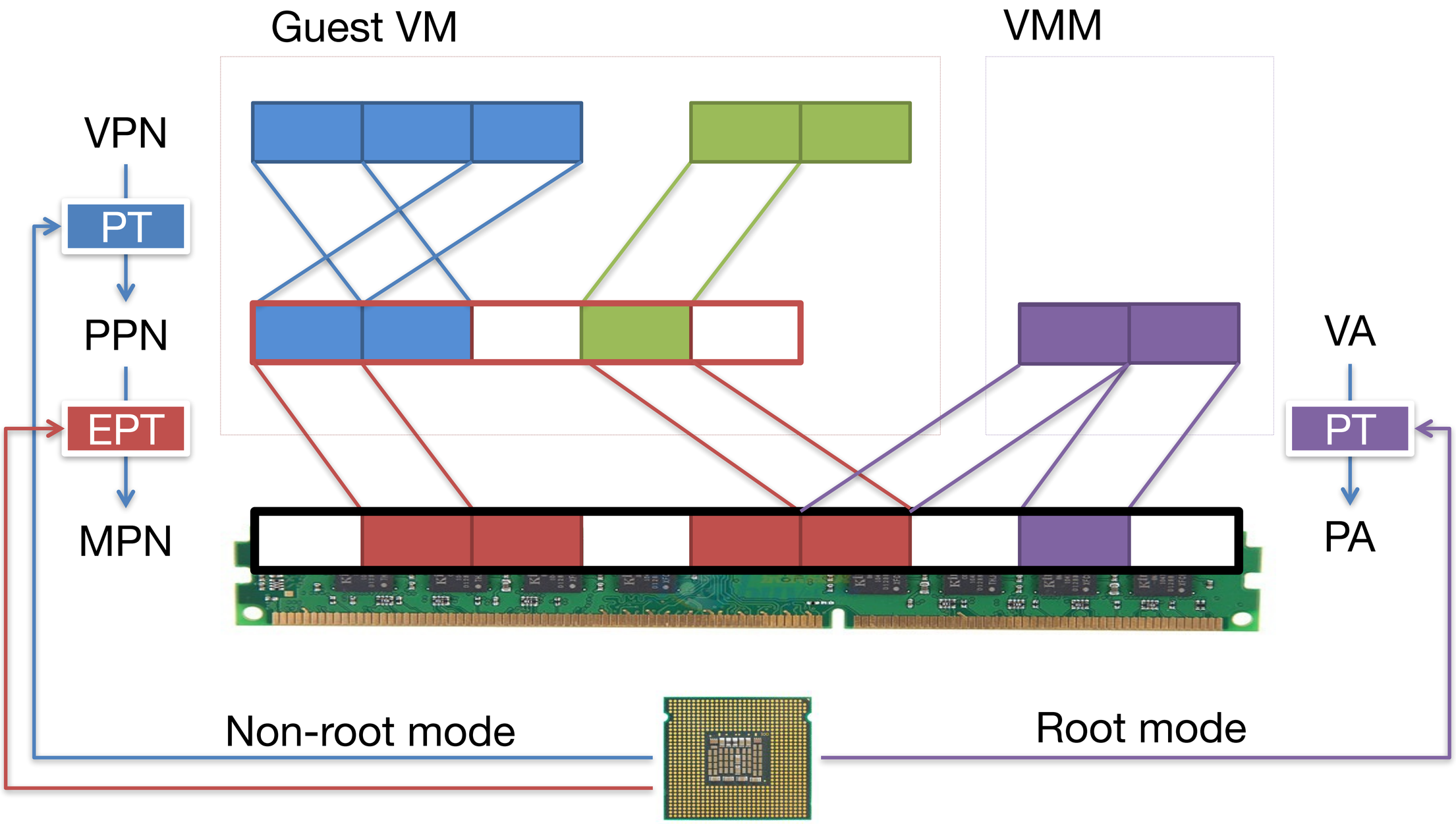
- Pros
  - When shadow PT is established, memory accesses are very fast
- Cons
  - Maintaining consistency between guest PTs and shadow PTs involve VMM traps, can be costly
  - TLB flush on every “world switch”
  - Memory space overhead to maintain *pmap*

Outline

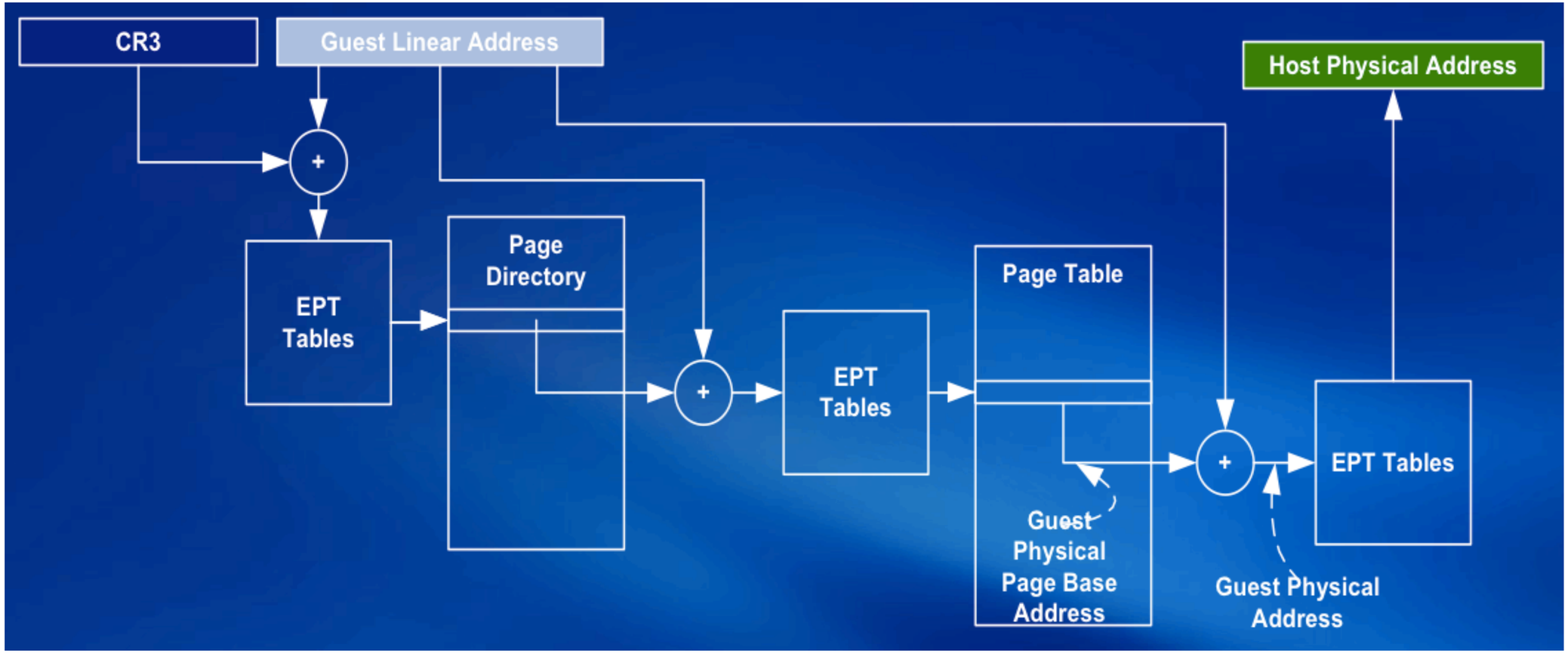
- Software-based memory virtualization
- Hardware-assisted memory virtualization
- Memory management
  - Reclaiming
  - Sharing
  - Allocation

Hardware-Assisted Memory Virtualization

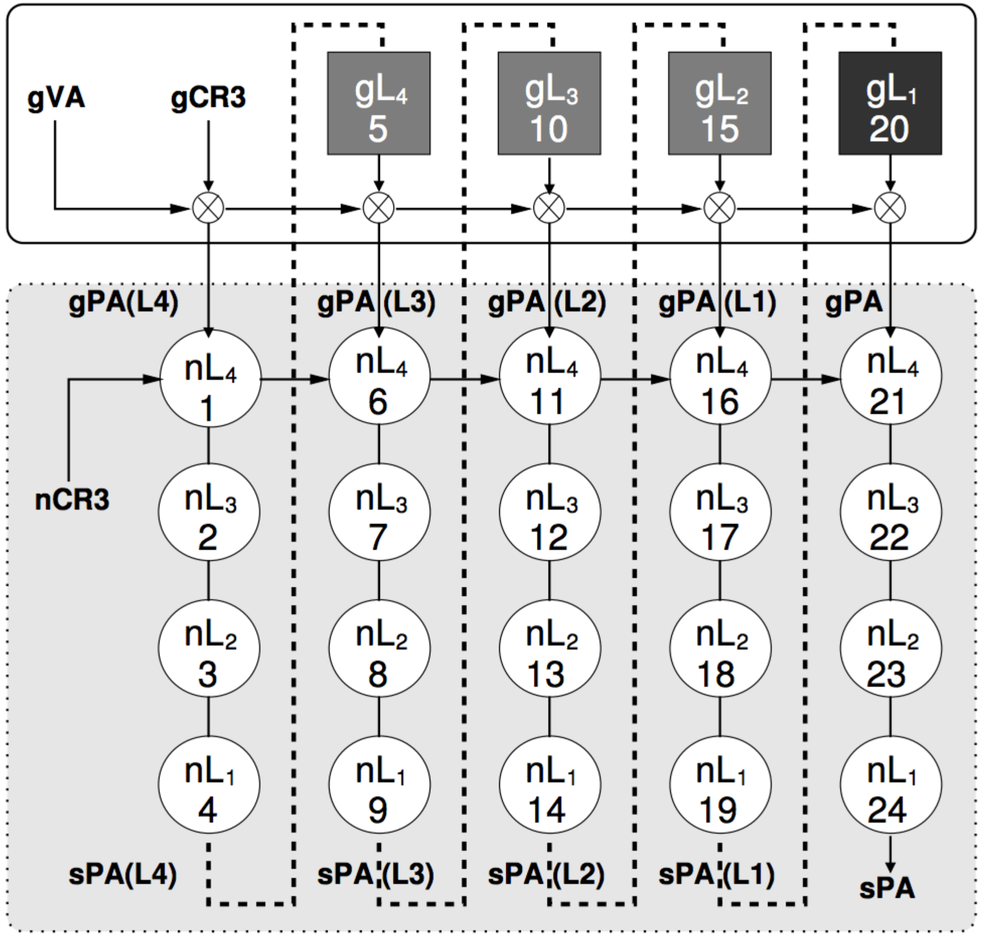
- Hardware support for memory virtualization
  - Intel EPT (Extended Page Table) and AMD NPT (Nested Page Table)
  - EPT: a per VM table translating PPN -> MPN, referenced by EPT base pointer
  - EPT controlled by the hypervisor, guest page table (GPT) controlled by guest OS (both exposed to hardware)
  - Hardware directly walks GPT + EPT (for each PPN access during GPT walk, needs to walk the EPT to determine MPN)
  - No VM exits due to page faults, INVLPG, or CR3 accesses


Guest VM
VMM
VPN
PT
PPN
EPT
MPN
VA
PT
PA
Non-root mode
Root mode

EPT Translation: Details



EPT Increases Memory Access



sPA: machine address
nCR3: root of EPT table
nL_k: EPT table level

One memory access from the guest VM may lead up to 20 memory accesses!

Pros and Cons of EPT

- Pros
  - Simplified VMM design (all handled by hardware)
  - Guest PT changes do not trap, minimize VM exits
  - Lower memory space overhead (no need for pmap in memory)
- Cons
  - TLB miss is costly: can involve many memory accesses to finish the walk!

Outline

- Software-based memory virtualization
- Hardware-assisted memory virtualization
- Memory management
  - Reclaiming
  - Sharing
  - Allocation

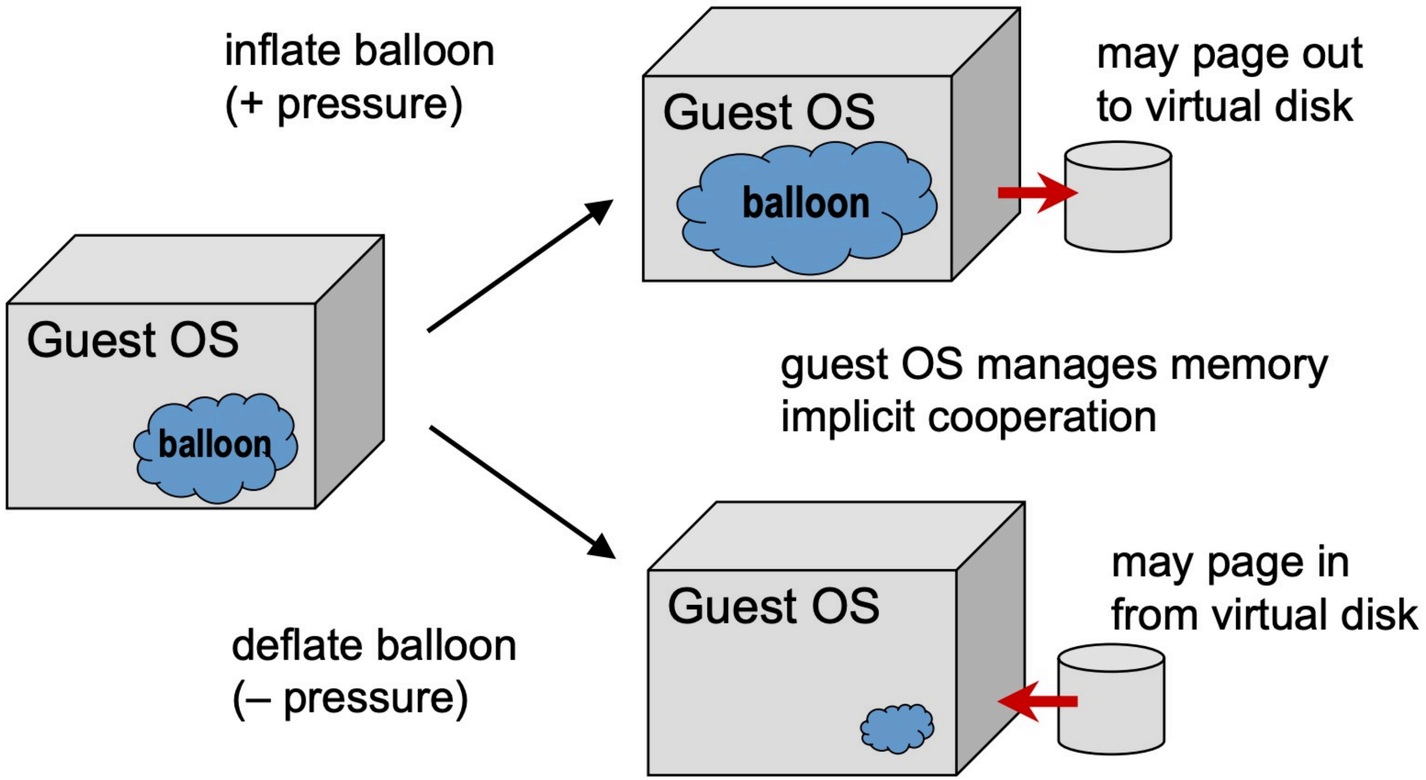
Reclaiming Memory

- ESX (and other hypervisors) allow overcommitment of memory
  - Total memory size of all VMs can exceed actual machine memory size
  - ESX must have some way to reclaim memory from VMs (and swap to disk)

Reclaiming Memory

- Traditional: add transparent swap layer
  - Requires “meta-level” decisions: which page from which VM to swap
  - Best data to guide decisions known only by guest OS
  - Guest and meta-level policies may clash, resulting in double paging
- Alternative: implicit cooperation
  - Coax guest OS into doing its own page replacement
  - Avoid meta-level policy decisions

Ballooning


inflate balloon
(+ pressure)
Guest OS
balloon
may page out
to virtual disk
Guest OS
balloon
guest OS manages memory
implicit cooperation
deflate balloon
(– pressure)
Guest OS
may page in
from virtual disk

Ballooning Details

- Guest drivers
  - Inflate: allocate pinned PPNs; backing MPNs reclaimed
  - Use standard Windows/Linux/BSD kernel APIs
- Performance benchmark
  - Linux VM, memory-intensive dbench workload
  - Compare 256 MB with balloon sizes 32 - 128 MB vs. static VMs
  - Overhead 1.4% - 4.4%

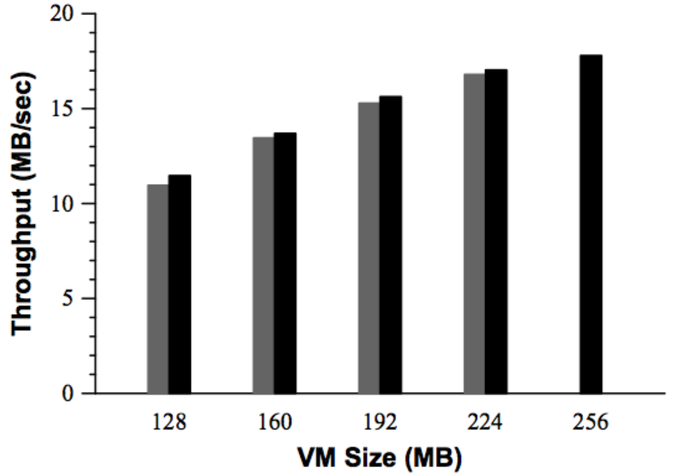
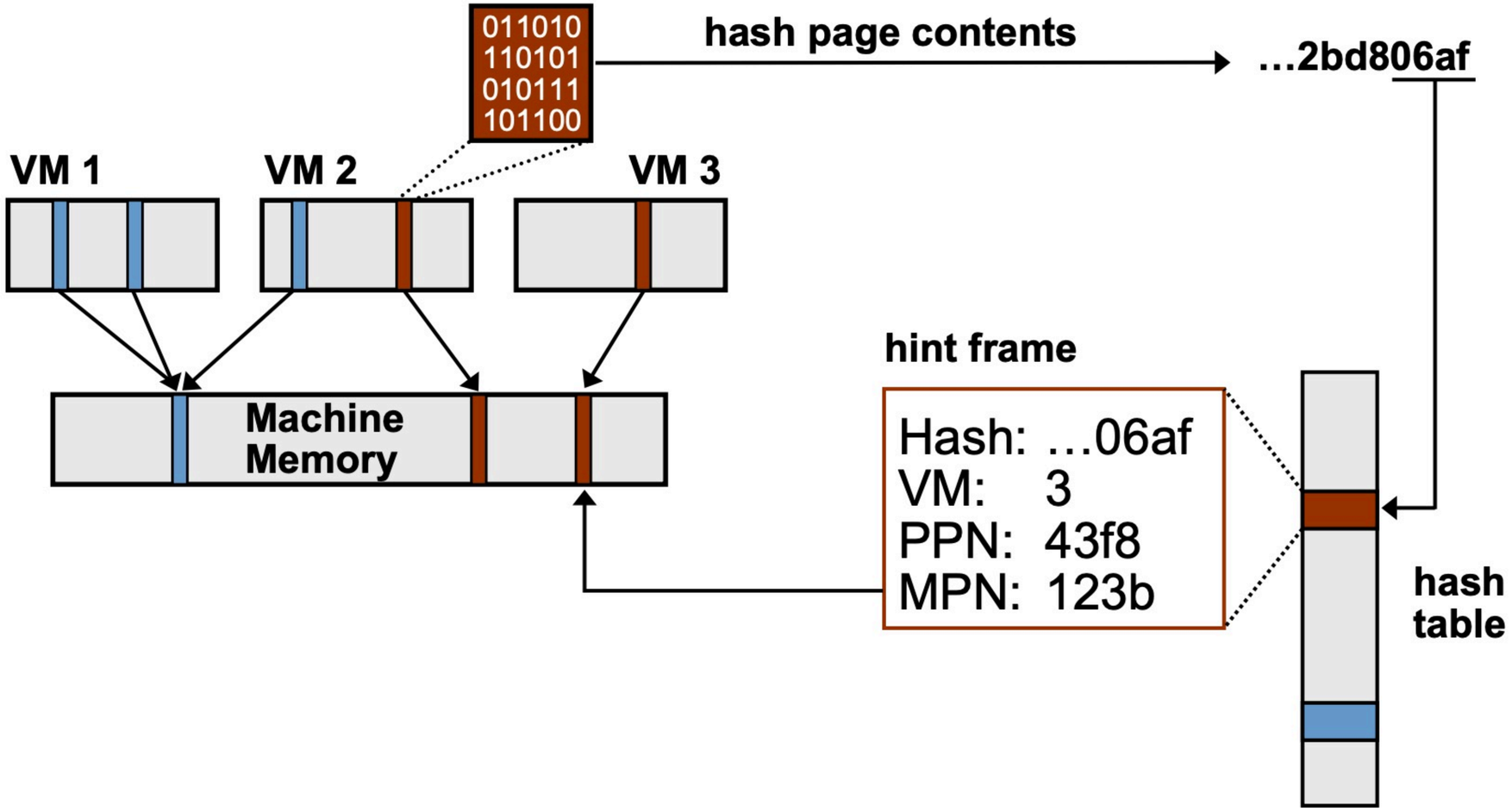


Figure 2: **Balloon Performance.** Throughput of single Linux VM running `dbench` with 40 clients. The black bars plot the performance when the VM is configured with main memory sizes ranging from 128 MB to 256 MB. The gray bars plot the performance of the same VM configured with 256 MB, ballooned down to the specified size.

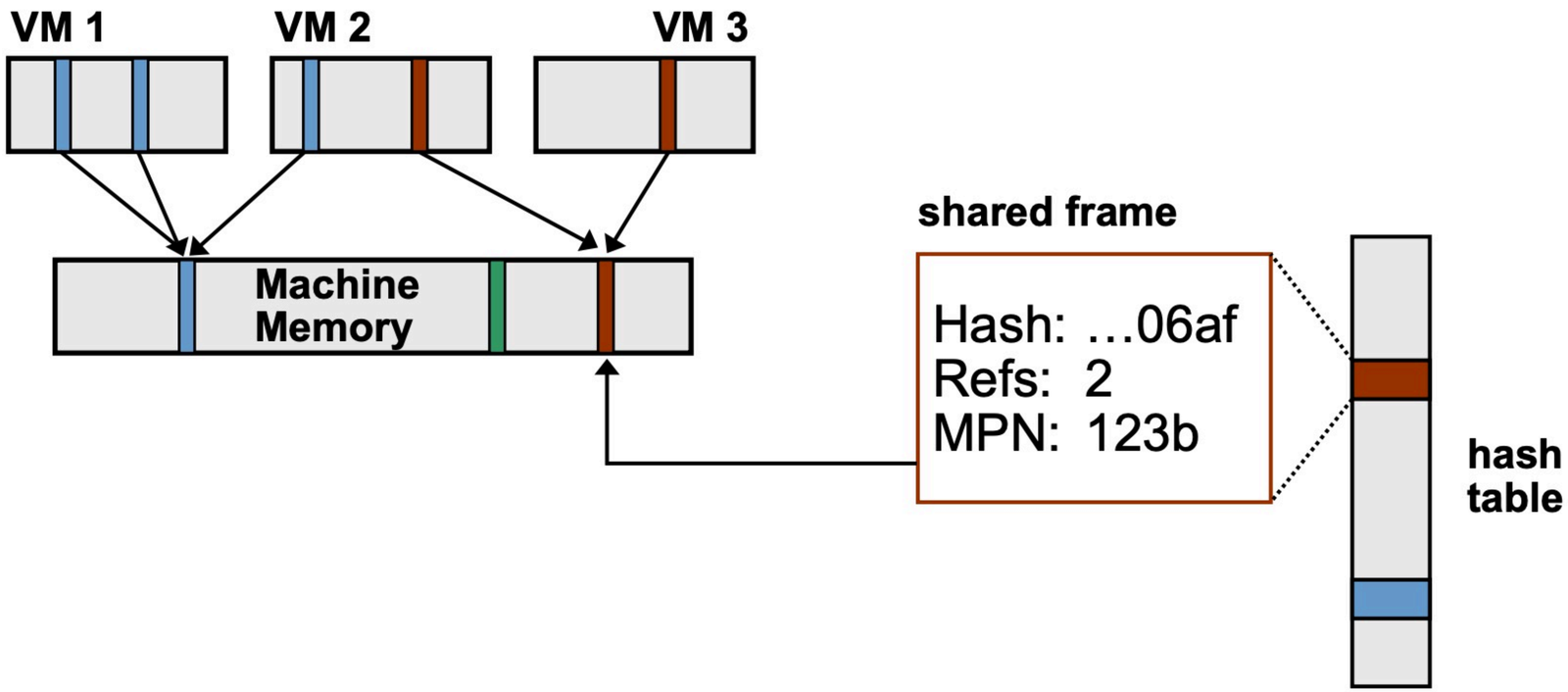
Memory Sharing

- Motivation
  - Multiple VMs running same OS, apps
  - Collapse redundant copies of code, data, zeros
- Transparent page sharing
  - Map multiple PPNs to single MPN (copy-on-write)
  - Pioneered by Disco, but required guest OS hooks
- New twist: content-based sharing
  - General-purpose, no guest OS changes
  - Background activity saves memory over time

Page Sharing: Scan Candidate PPN



Page Sharing: Successful Match



Real-World Page Sharing Results

		Total	Saved	
Workload	Guest Types	MB	MB	%
Corporate IT	10 Windows	2048	673	32.9
Nonprofit Org	9 Linux	1846	345	18.7
VMware	5 Linux	1658	120	7.2

Corporate IT – database, web, development servers (Oracle, Websphere, IIS, Java, etc.)

Nonprofit Org – web, mail, anti-virus, other servers (Apache, Majordomo, MailArmor, etc.)

VMware – web proxy, mail, remote access (Squid, Postfix, RAV, ssh, etc.)

Memory Allocation: Allocation Parameters

- Min size
  - Guaranteed, even when overcommitted
  - Enforced by admission control
- Max size
  - Amount of “physical” memory seen by guest OS
  - Allocated when undercommitted
- Shares
  - Specify relative importance
  - Proportional-share fairness

Allocation Policy

- Traditional approach
  - Optimize aggregate system-wide metric
  - Problem: no QoS guarantees, VM importance varies
- Pure share-based approach
  - Revoke from VM with min shares-per-page ratio [Waldspurger '95]
  - Problem: ignores usage, unproductive hoarding [Sullivan '00]
- Desired behavior
  - VM gets full share when actively using memory
  - VM may lose pages when working set shrinks

Reclaiming Idle Memory

- Tax on idle memory
  - Charge more for idle page than active page
  - Idle-adjusted shares-per-page ratio
- Tax rate
  - Explicit administrative parameter
  - 0% ≈ “plutocracy” … 100% ≈ “socialism”
- High default rate
  - Reclaim most idle memory
  - Some buffer against rapid working-set increases

Dynamic Reallocation

- Reallocation events
- Enforcing target allocations
  - Ballooning: common-case optimization
  - Swapping: dependable fallback, try sharing first
- Reclamation states
  - High: background sharing
  - Soft: mostly balloon
  - Hard: mostly swap
  - Low: swap and block VMs above target

Conclusion

- Software and hardware solutions for memory virtualization both have pros and cons
- More things to take care of besides the basic mechanism of memory virtualization
  - Allocation, sharing, overcommitment and reclamation