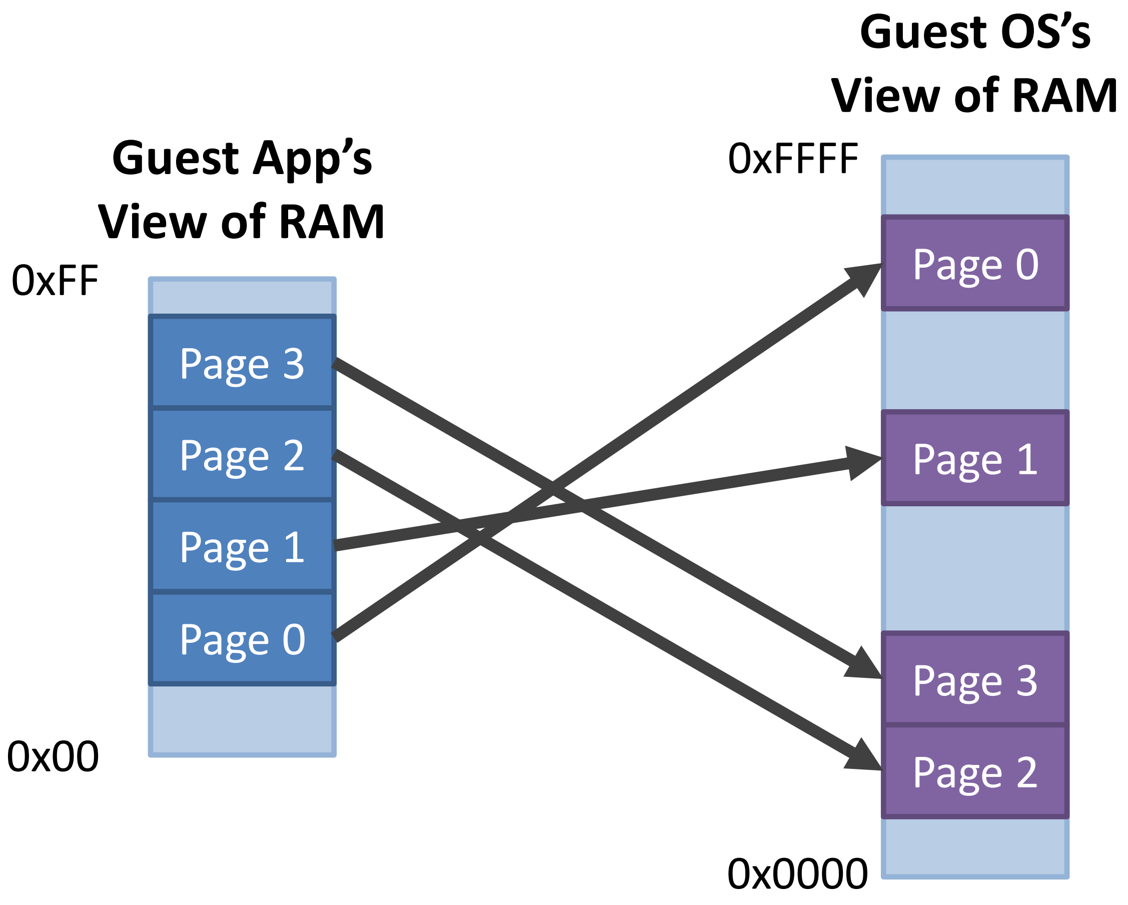
x86 Memory Virtualization

Dr. Amit Praseed

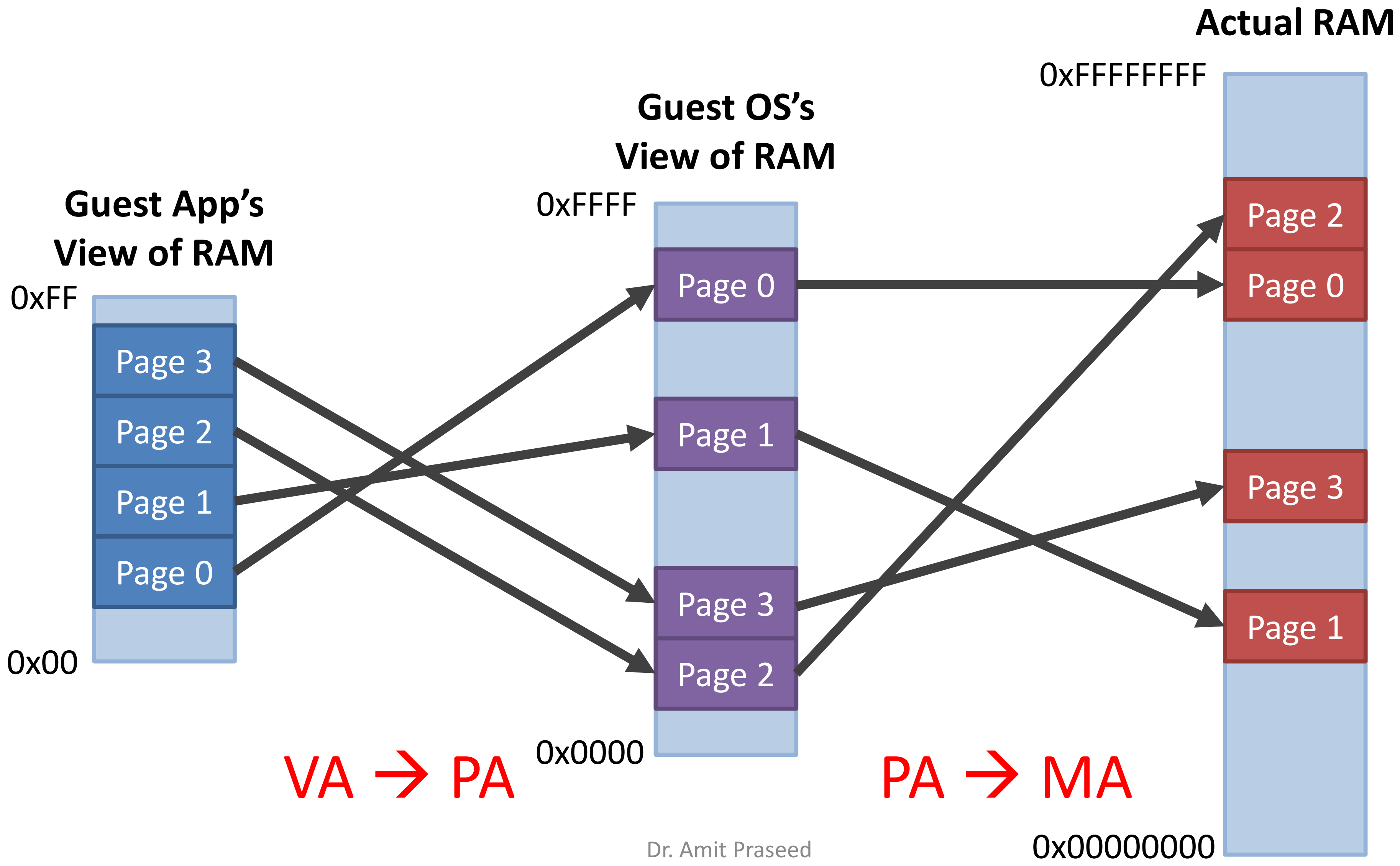
Memory Virtualization

- Typical x86 architecture has a virtual to physical address mapping (VA → PA)
- Virtualized x86 architecture requires a two level address translation
  - VA → PA
  - Physical address (PA) → Machine Address (MA)
- Guest OS has no idea about this translation
  - Guest continues to maintain page tables containing VA → PA mappings

Paging without Virtualization



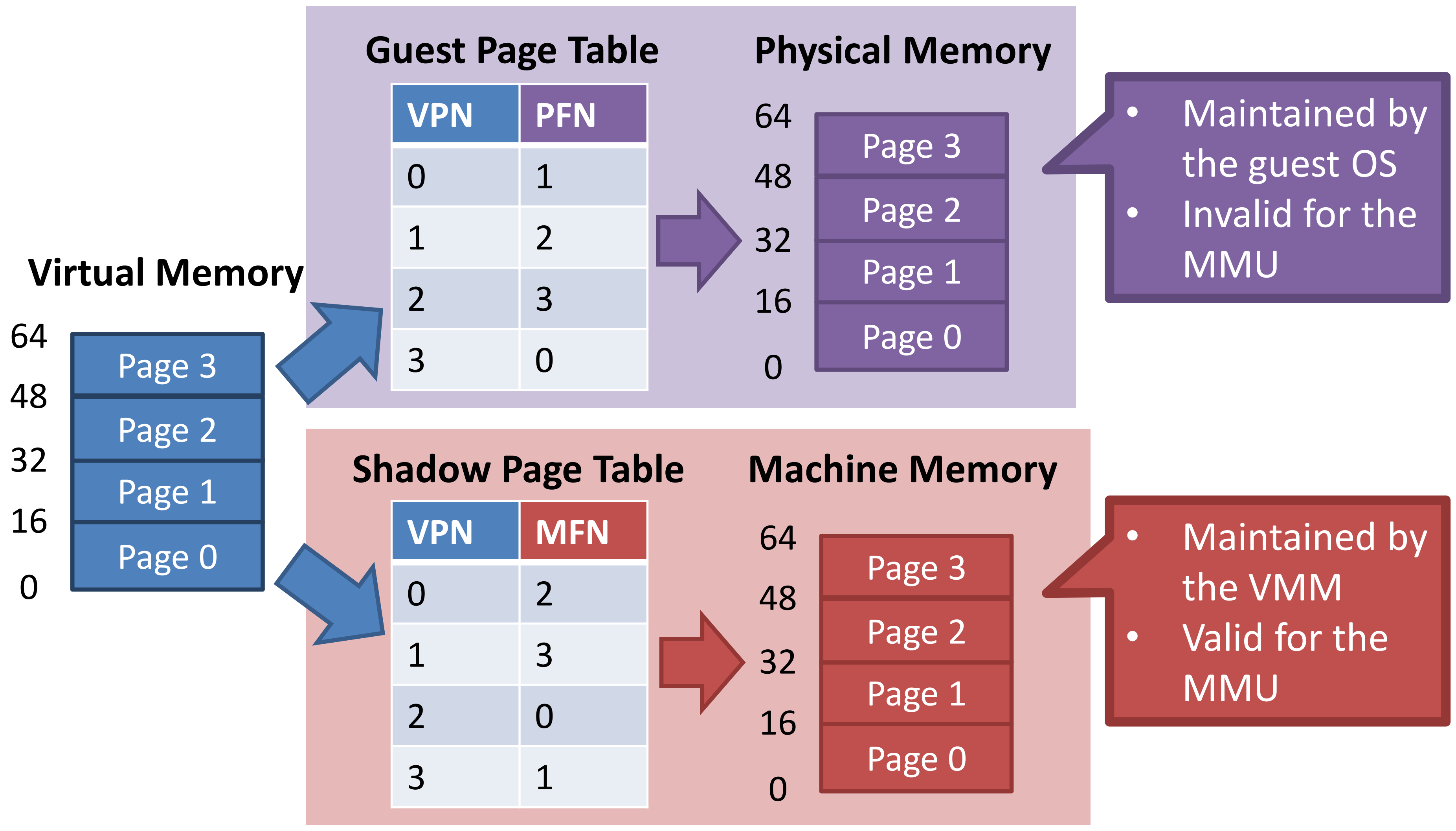
Paging with Virtualization



Does a 2 Level Indirection Work?

- Guest is only aware of VA → PA mapping
  - Issues only VA to hardware MMU
  - **MMU supports only a single mapping**
- Solution: Shadow Page Table
  - Hypervisor maintains a single shadow page table in the MMU
  - Shadow page table contains direct VA → MA mapping
  - Trick is to maintain consistency!!!

Shadow Page Tables



Guest Page Table

VPN	PFN
0	1
1	2
2	3
3	0

- Maintained by the guest OS
- Invalid for the MMU

Shadow Page Table

VPN	MFN
0	2
1	3
2	0
3	1

- Maintained by the VMM
- Valid for the MMU

Building Shadow Page Tables

- The guest can update its page tables at any time
  - Not a privileged instruction – not trapped!
  - Without knowing when the guest OS updates its page table, the hypervisor cannot maintain the correct entry in the shadow page table
- Solution: Mark the guest page tables as read only
  - Writing generates an exception, which can be trapped by hypervisor

What happens during page faults?

- Two kinds of page faults can occur:
  - True Miss : The mapping does not exist in the guest page table
  - Hidden Miss : The mapping exists in the guest page table, but is absent in the shadow page table
- The hypervisor should disambiguate between the two
- On every miss, the hypervisor walks the guest page table [Tracing]
  - If a mapping exists, the hypervisor silently updates the shadow page table and retries the instruction
  - Otherwise, the hypervisor forwards the page fault to the guest OS for handling

Pros and Cons

- The good: shadow tables allow the MMU to directly translate guest VPNs to hardware pages
  - Thus, guest OS code and guest apps can execute directly on the CPU
- The bad:
  - Double the amount of memory used for page tables
    - i.e. the guest's tables and the shadow tables
  - Overhead due to VMM traps
  - TLB flush whenever a "world switch" occurs
    - Loss of performance

Second Level Address Translation (SLAT)

- Hardware support for memory virtualization
  - Extended Page Tables (EPT) – Intel
  - Nested Page Tables (NPT) - AMD
- Walking the guest and host page tables can be combined into a single multilevel page table
  - Page table depth increases tremendously in some cases!!!
  - Extremely important to use an effective TLB to avoid expensive page table walks
- TLB modified to reduce miss rate
  - Larger TLB
    - More expensive, though!!!
  - Tagged TLB
    - Every entry in TLB now has an address space identifier
    - No need to flush TLB for every "world switch"