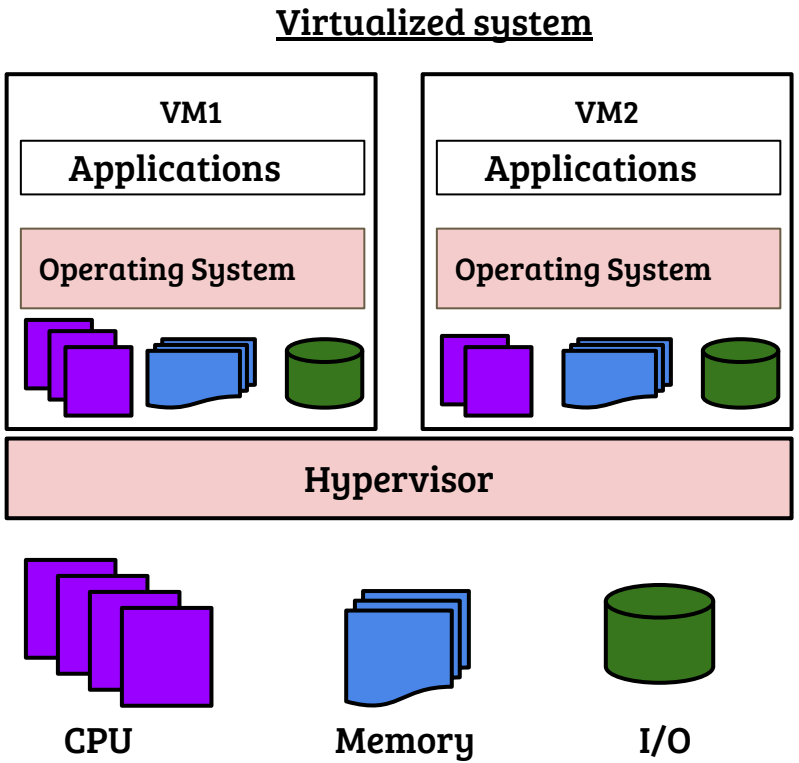
Topics in Operating Systems

Advanced isolation: Virtualization (memory)

Debadatta Mishra, CSE, IITK

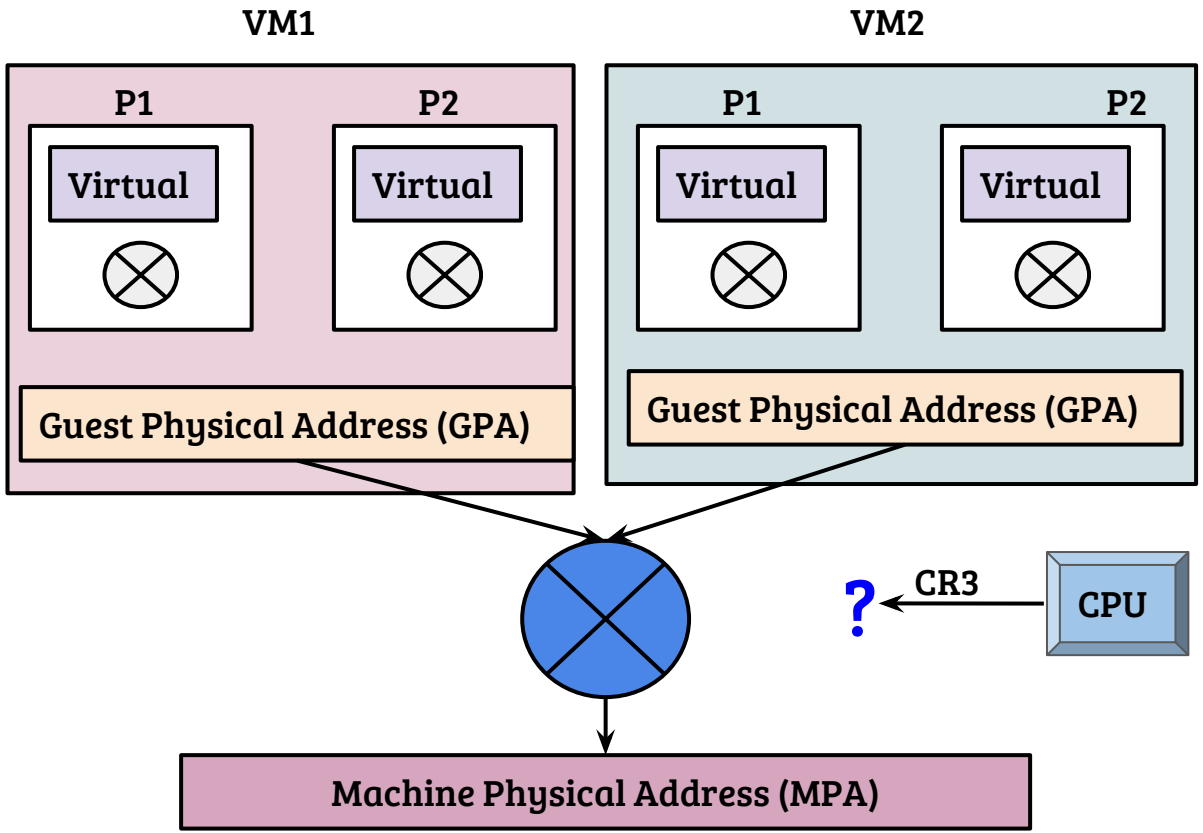
Virtualization: Resource multiplexing with isolation



- Definition [1] "Not physically existing as such but made by software to appear to do so."
- Objectives
  - Equivalence
  - Isolation
  - Resource control
  - Efficiency

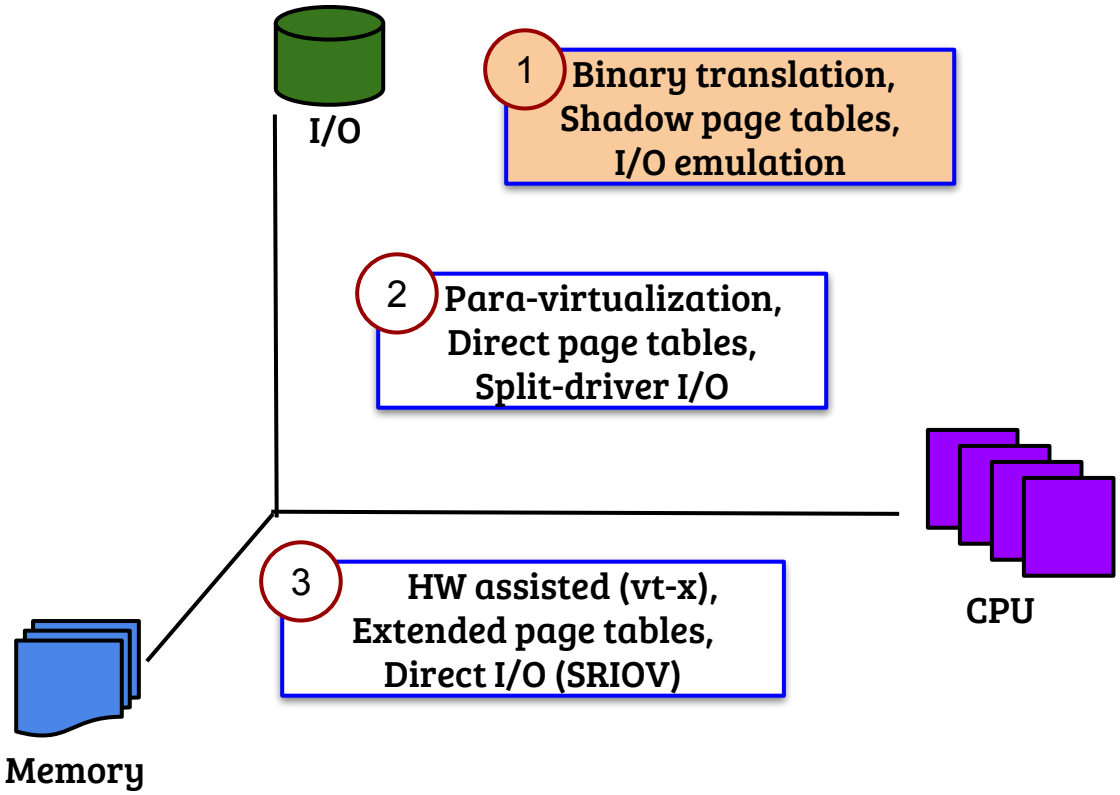
1. Oxford dictionary : https://en.oxforddictionaries.com/definition/virtual

Doubly virtualized memory!



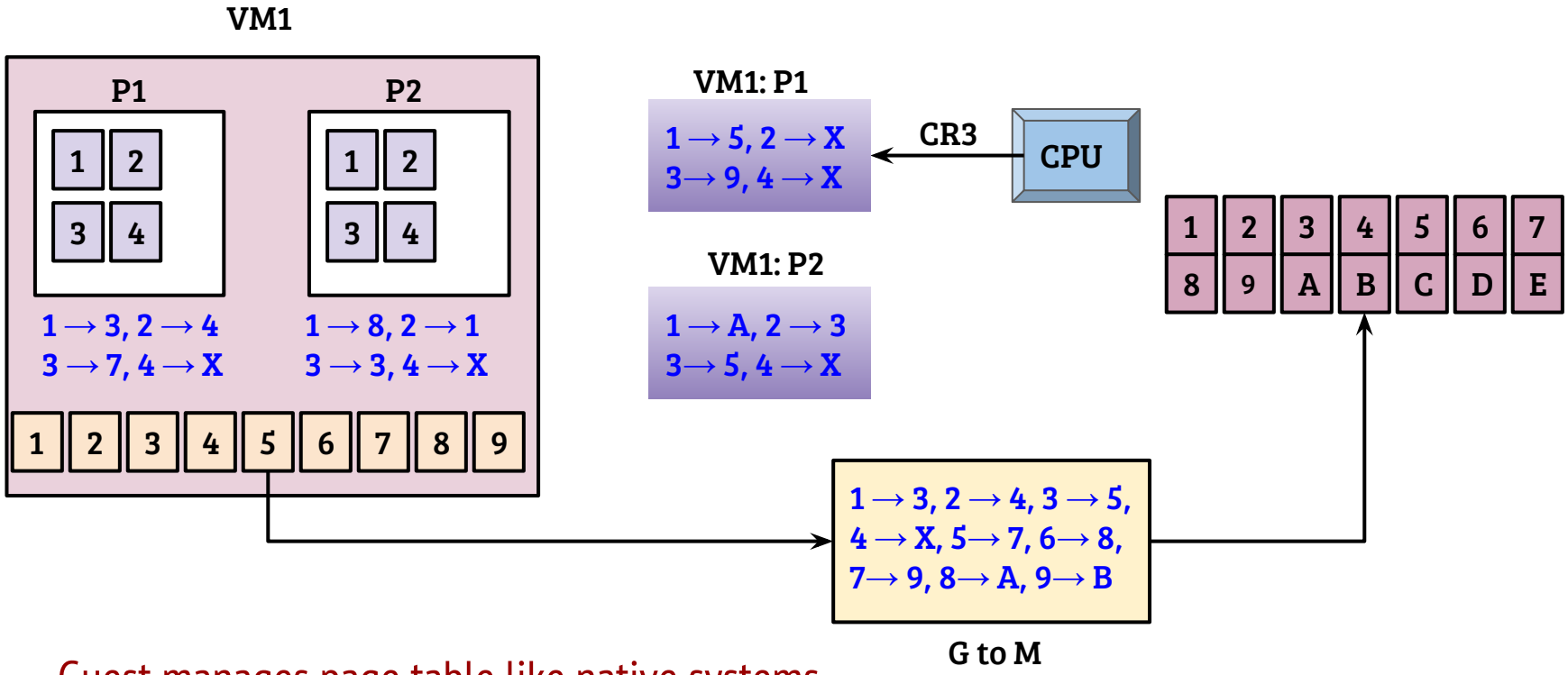
- Two levels of translation
  - V ⟶ GPA
  - GPA ⟶ MPA
- Two types of context switch
  - Intra-VM
  - Inter-VM
- Two sources of Page fault
  - Invalid {V ⟶ GPA}
  - Invalid {P ⟶ MPA}

Overview of virtualization approaches



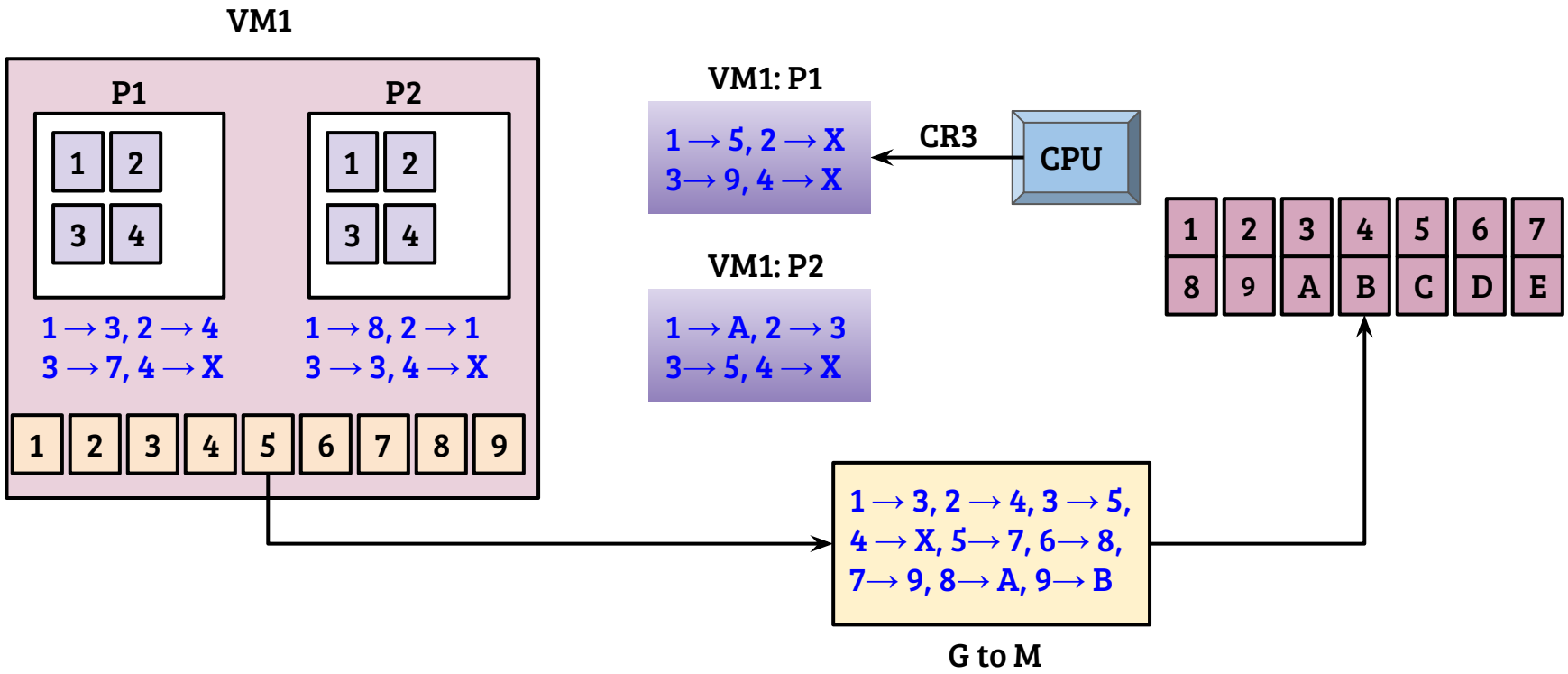
- Agenda for today's lecture: Memory virtualization
- Software only techniques: shadow page tables and direct page tables
- Hardware assisted: nested/extended page tables

Shadow paging: Basic design



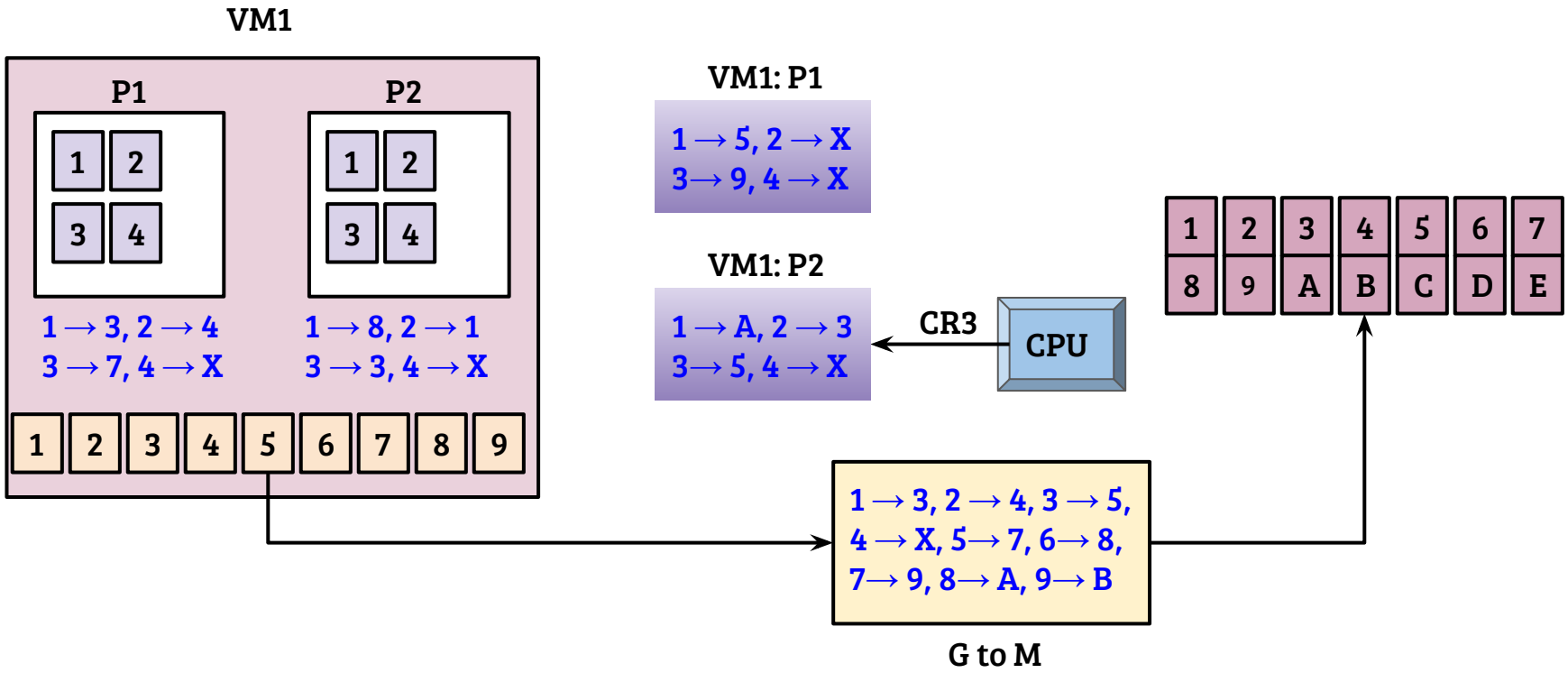
- Guest manages page table like native systems
- Hypervisor maintains a per-process shadow page table and per-process GPFN ⟶ MFN information (G to M)

Shadow paging: Basic design



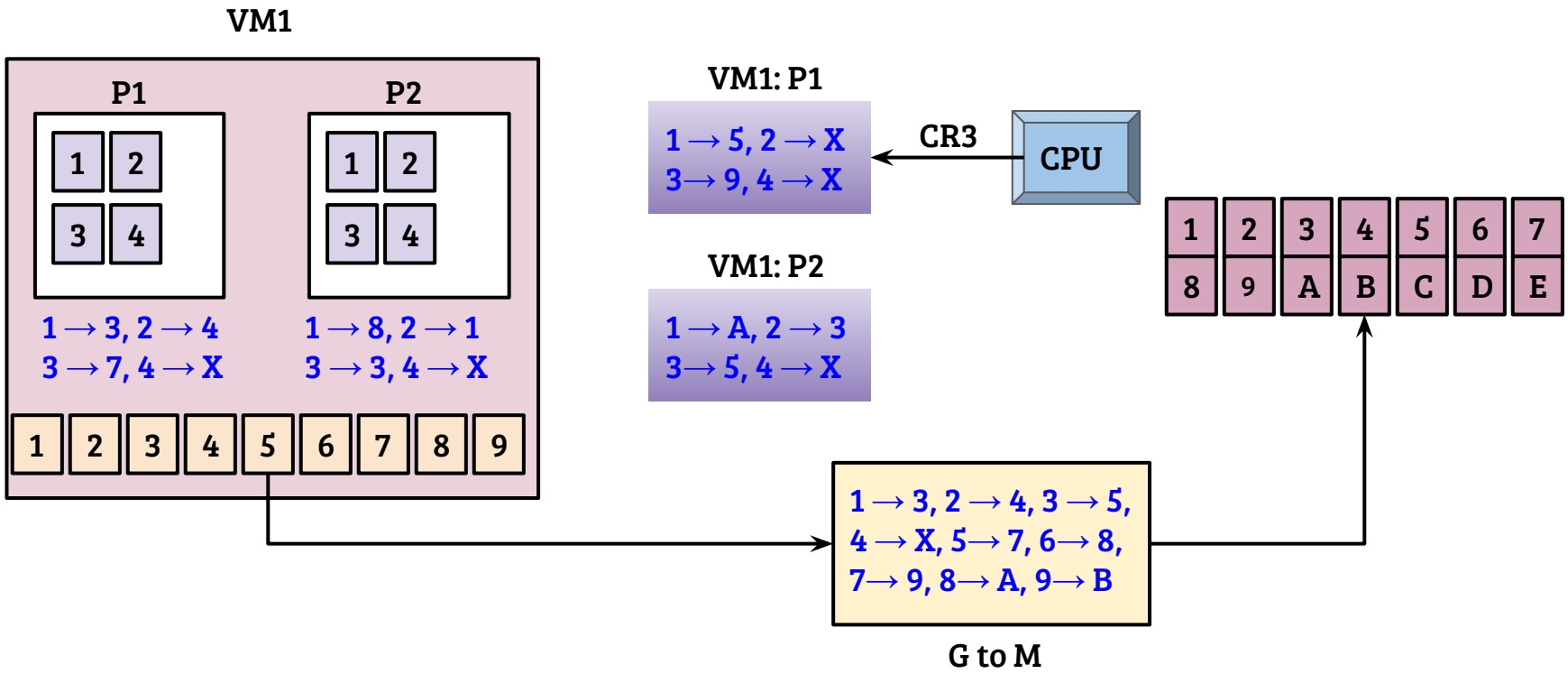
Event: Context switch from P1 to P2

Shadow paging: Context switch



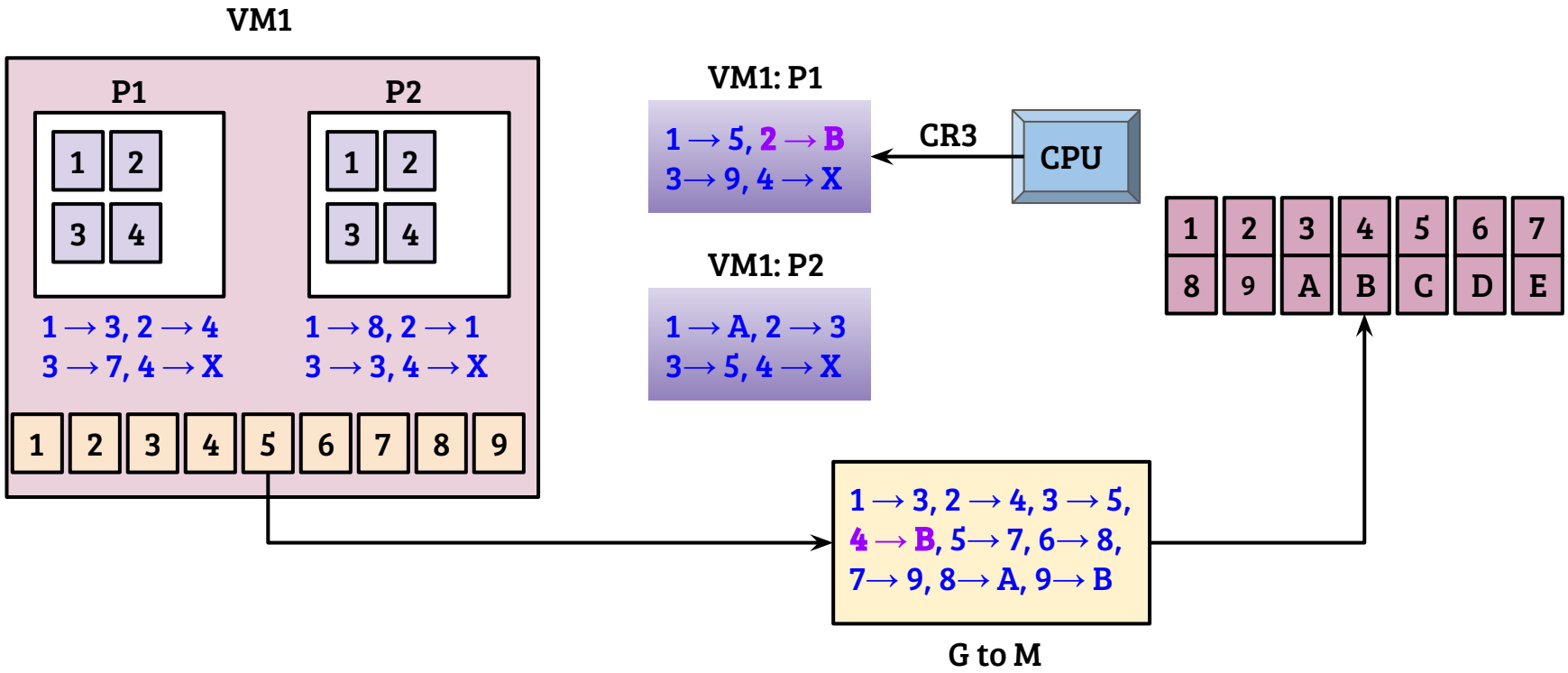
- Hypervisor updates the CR3 (CR3 write trap) to point to the shadow page table of P2

Shadow paging: Page fault handling



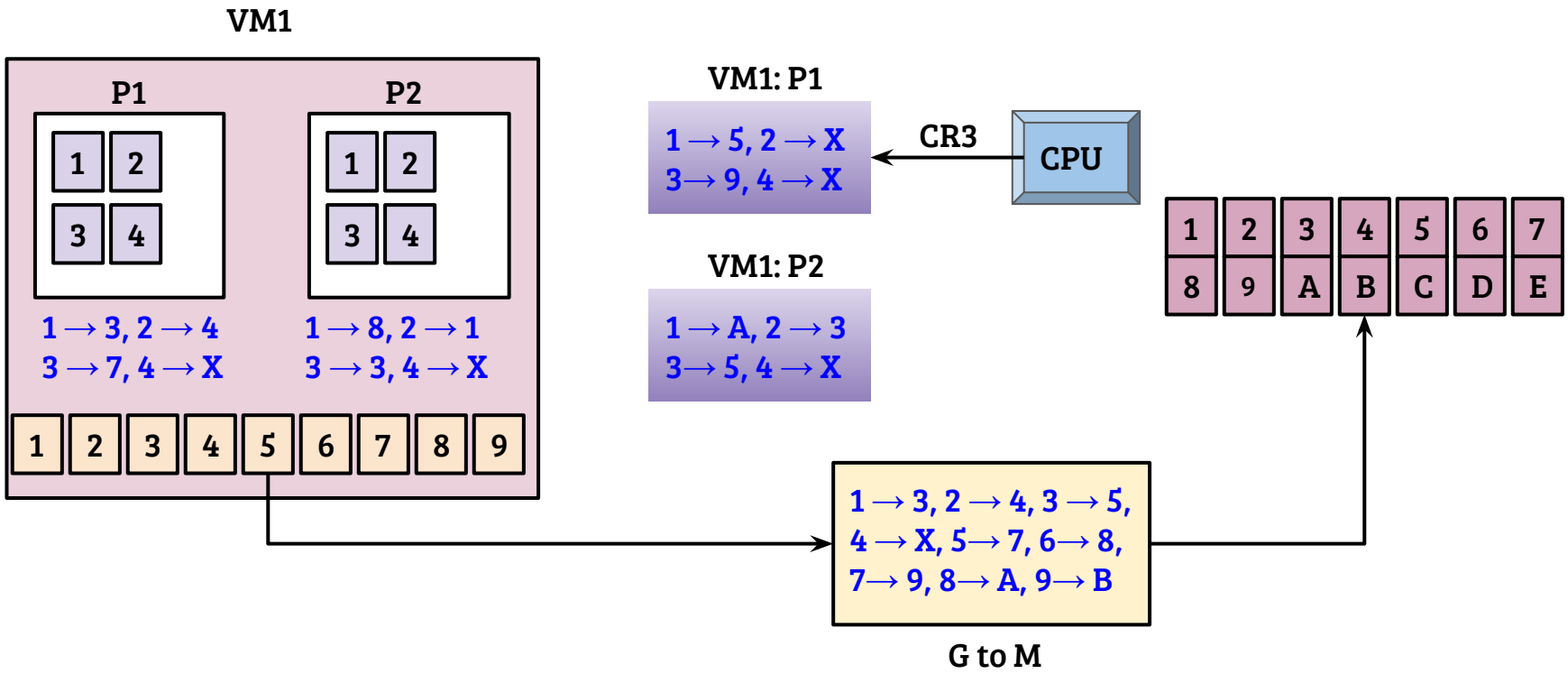
P1 access virtual address 2 ⟶ Page fault ⟶ Handled @ hypervisor

Shadow paging: Page fault handling



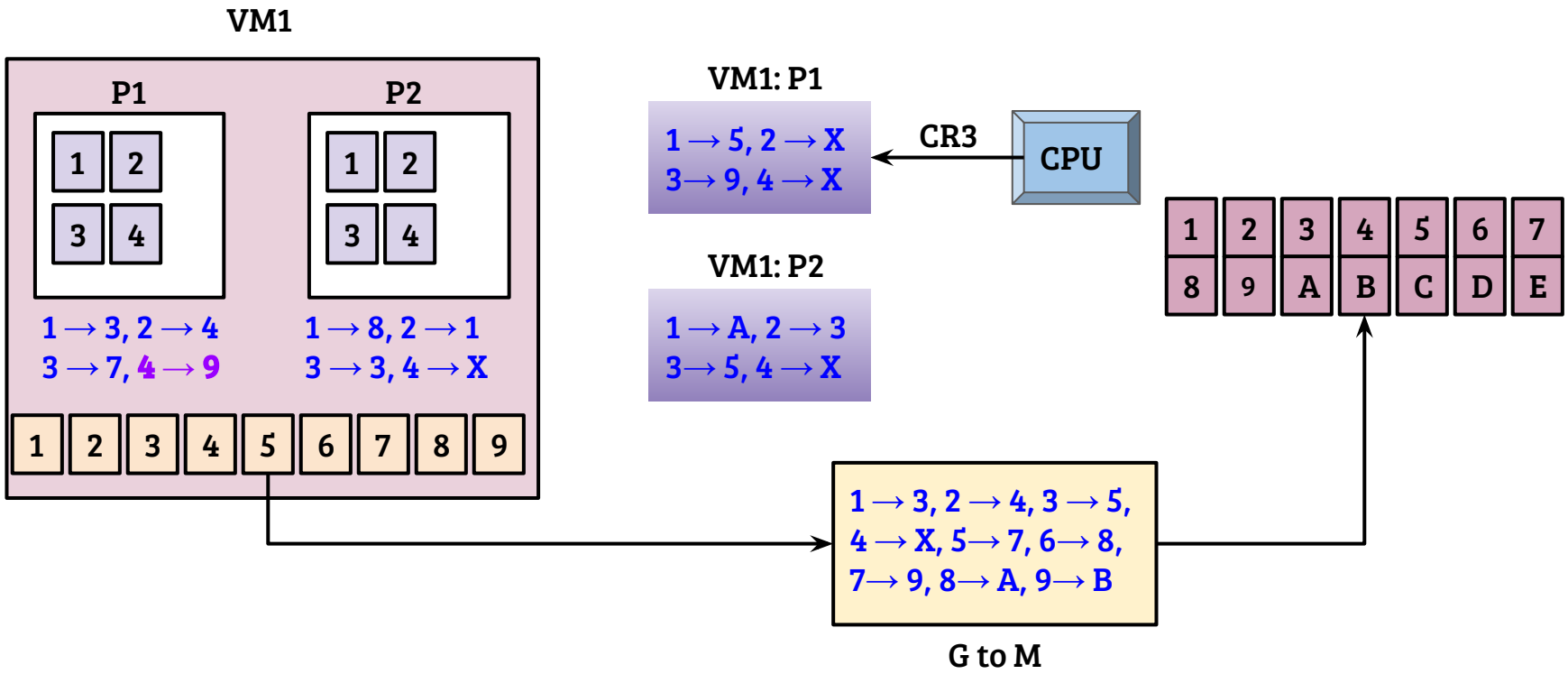
- Hypervisor updates the shadow page table and G to M mapping information

Shadow paging: Page fault handling



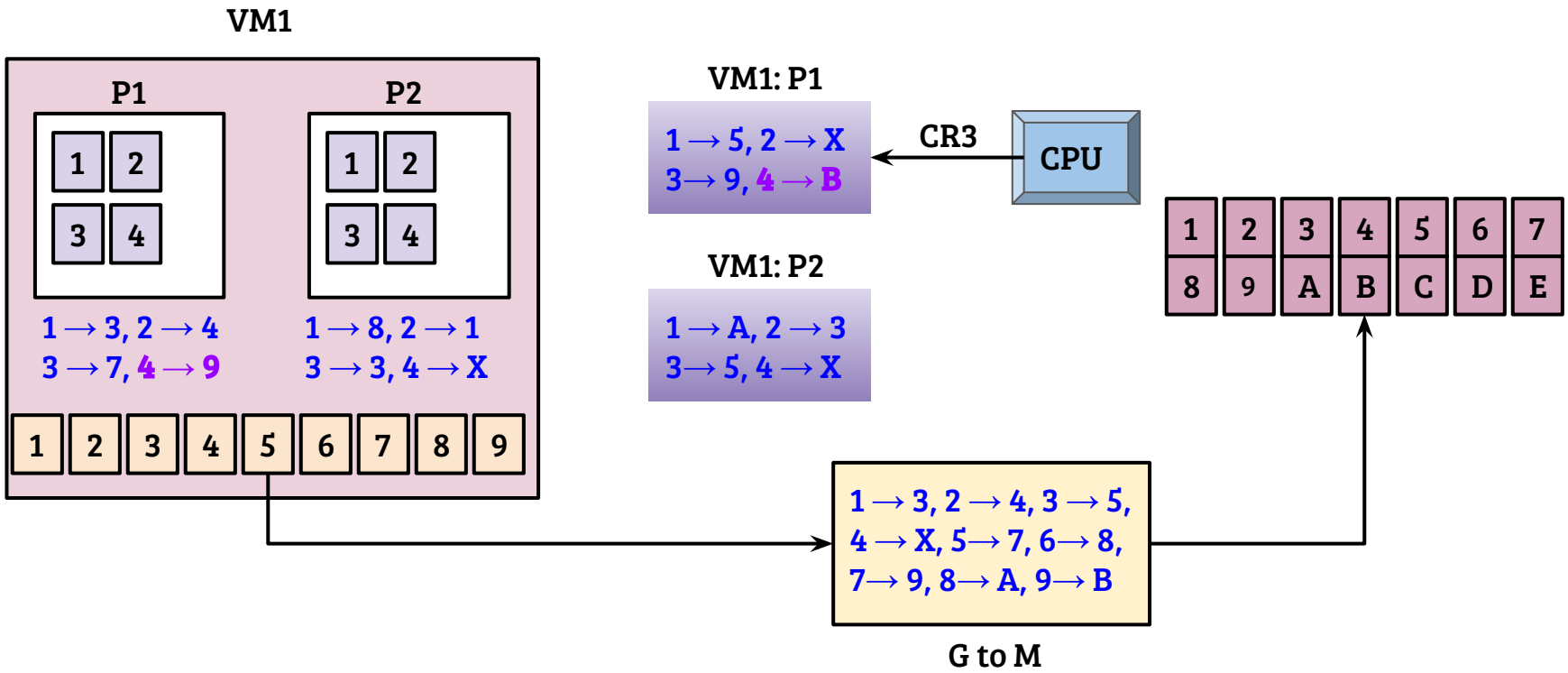
P1 access VA 4 ⟶ Page fault ⟶Who handles Page fault?

Shadow paging: Page fault handling



- Guest OS updates the page table (4 → 9), how will the shadow page table be in sync?

Shadow paging: Page fault handling



- Trap (VMExit) as the guest page tables are read-only
- Therefore, no issues with isolation enforcement

Shadow paging: conclusion

- Shadow paging does not require guest OS modification (equivalence)
- Quiz: Assume a scenario when for any given process
  - All virtual addresses are mapped
  - No updates to page table mappings
  - Shadow paging performance?

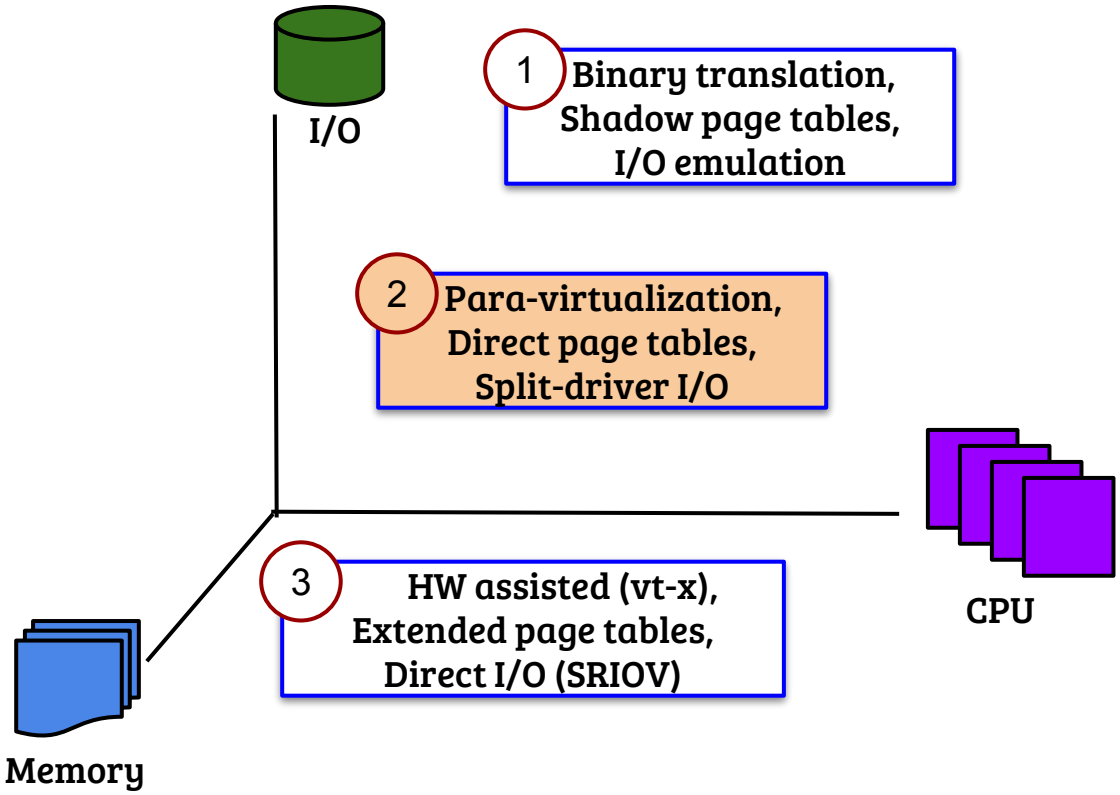
- How many shadow pages to be maintained if there are **N** VMs with **P** processes each?

- Worst case scenario for shadow page tables?

Shadow paging: conclusion

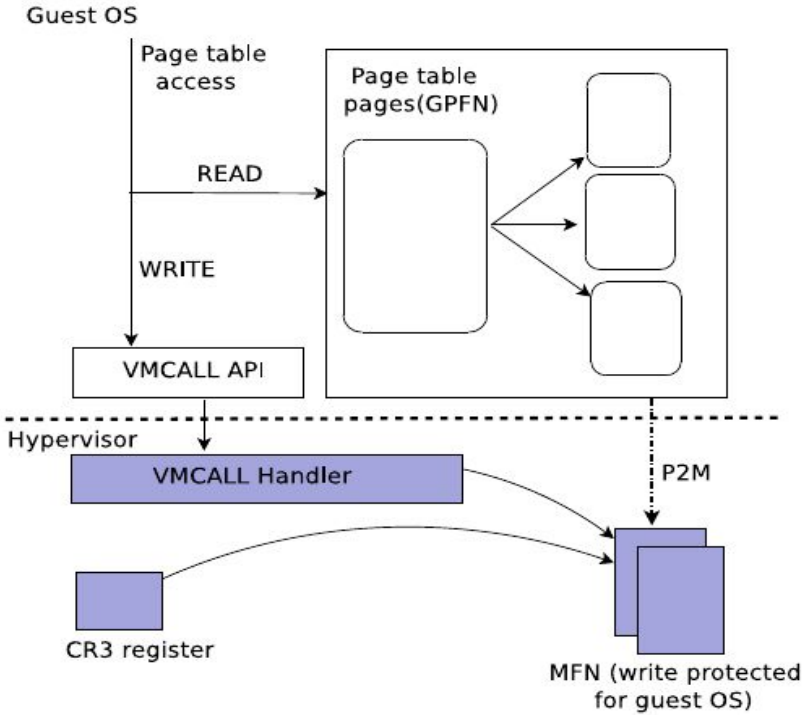
- Shadow paging does not require guest OS modification (equivalence)
- Quiz: Assume a scenario when for any given process
  - All virtual addresses are mapped
  - No updates to page table mappings
  - Shadow paging performance?
  - Comparable to native paging performance
- How many shadow pages to be maintained if there are **N** VMs with **P** processes each?
- **N X P**
- Worst case scenario for shadow page tables?
  - A lot of page faults (memory alloc and dealloc)
  - Frequent context switch

Overview of virtualization approaches



- Agenda for today's lecture: Memory virtualization
- Software only techniques: shadow page tables and direct page tables
- Hardware assisted: nested/extended page tables

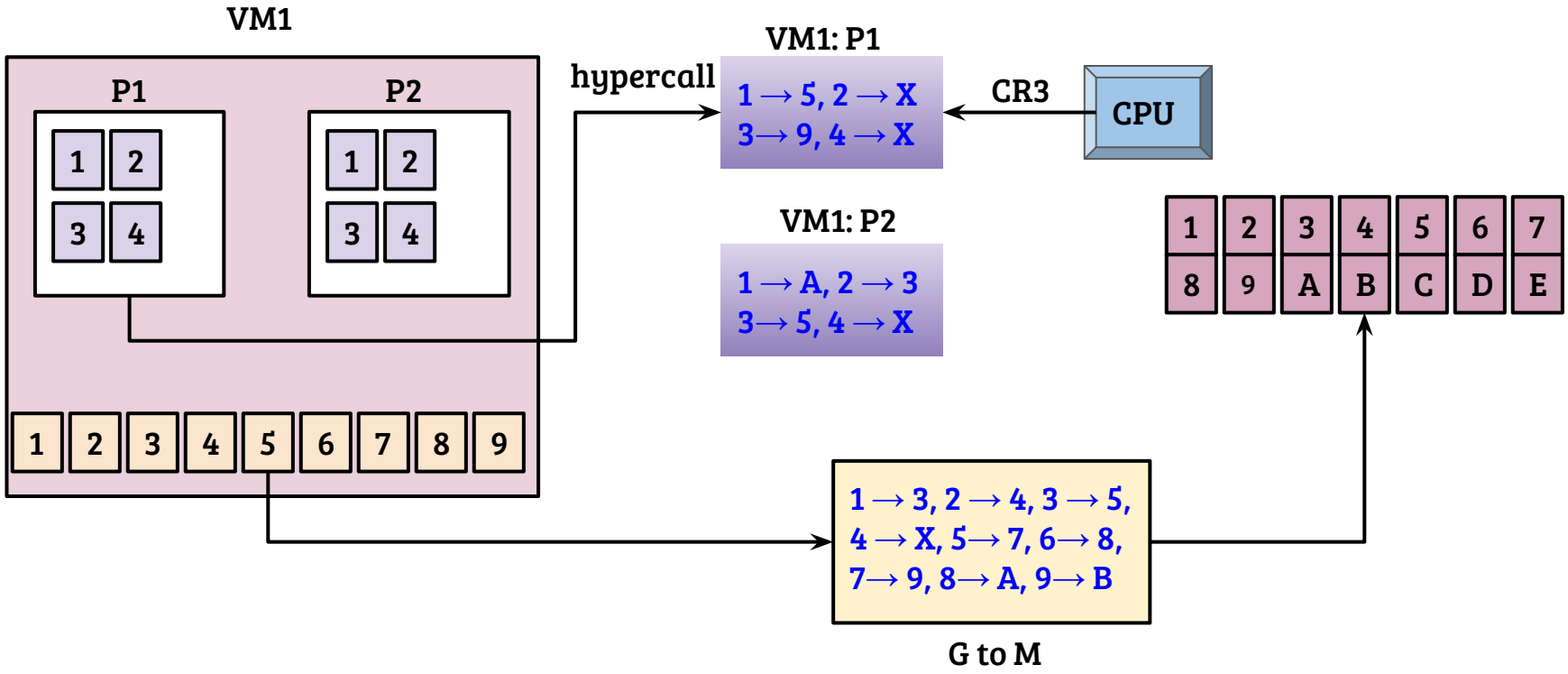
Direct page tables (Xen) [1]



- Guest OS is aware that the page table is maintained by the hypervisor ⟶ illusion of GPFN is not required
- CR3 updates generate a trap
- Guest OS is modified to invoke hypercall to perform updates to the page table
- Hypervisor maintains the PFN to MFN mapping information

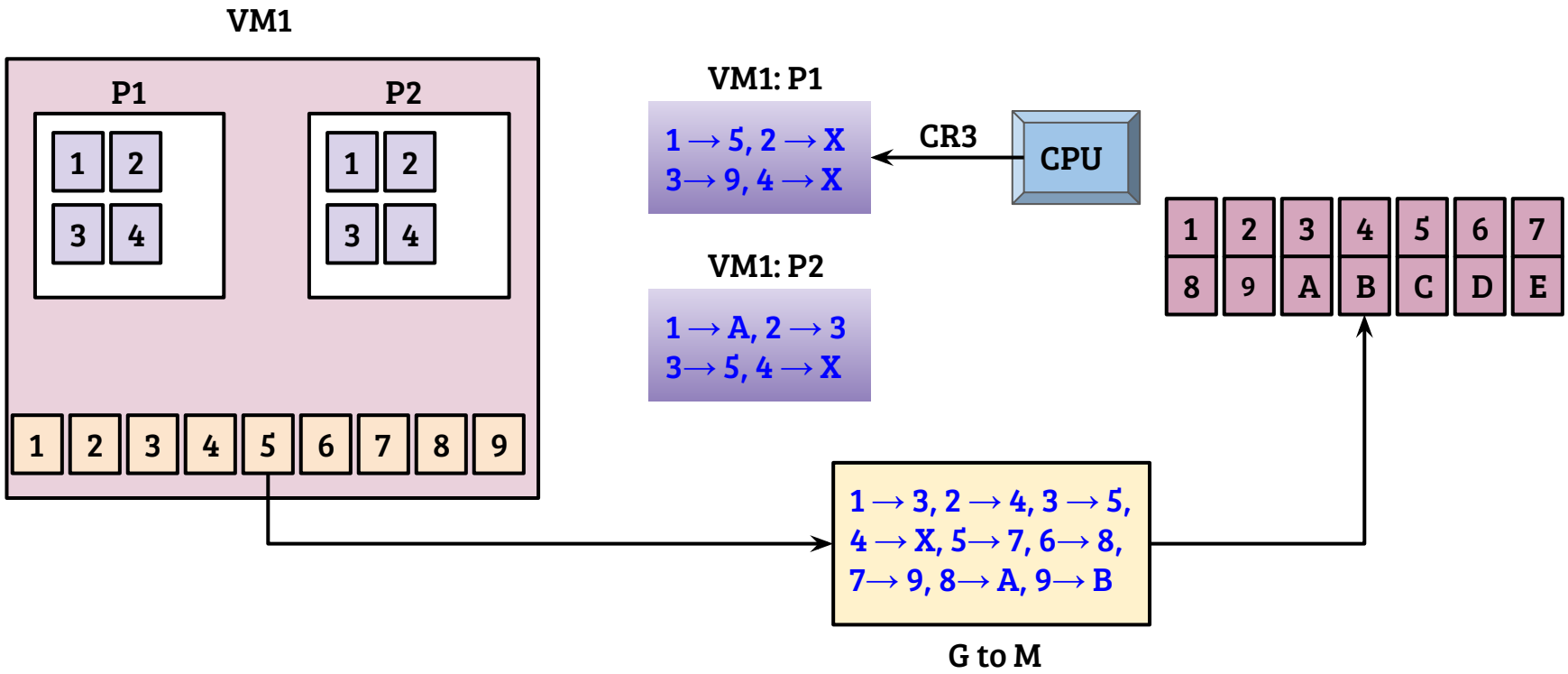
1. Xen and the art of virtualization, https://dl.acm.org/citation.cfm?id=945462

Direct paging: Basic design



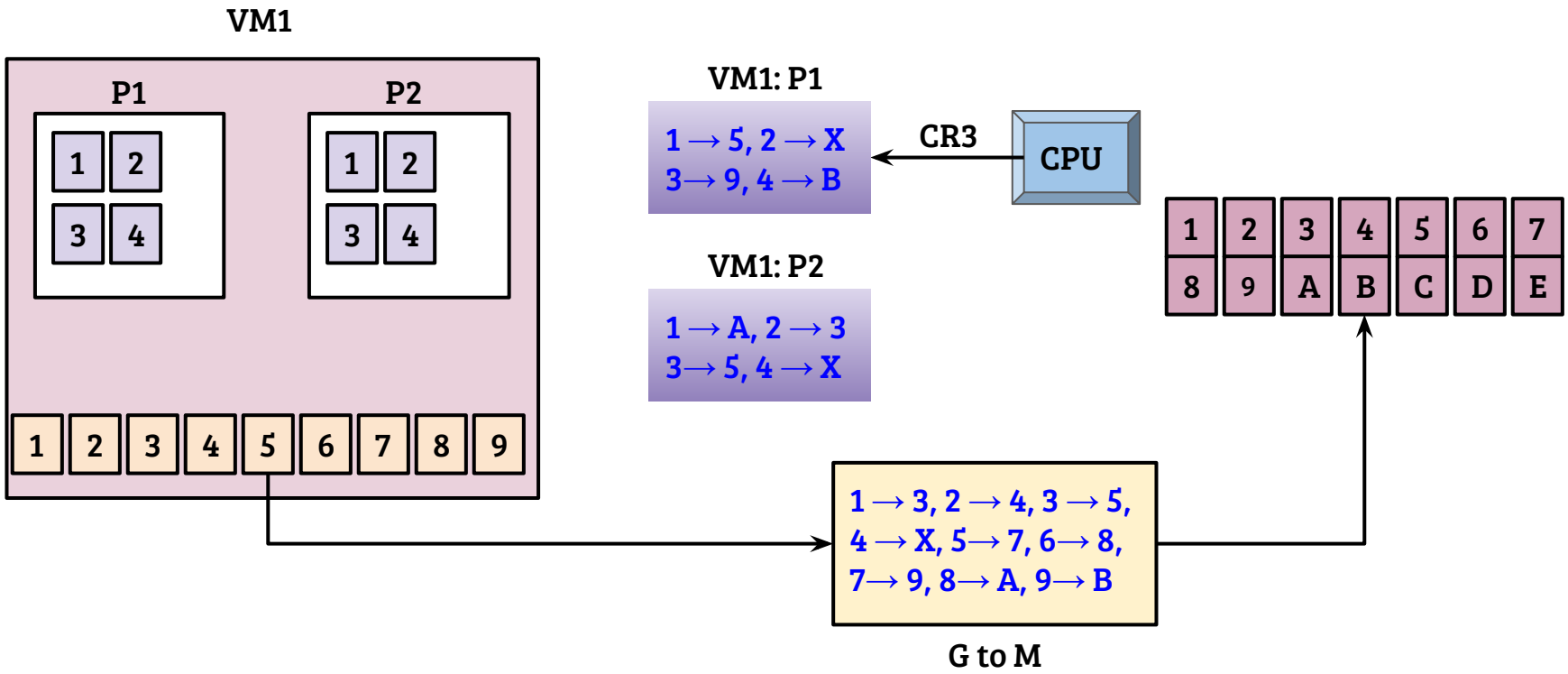
- Single per-process page table set during context switch
- A per-VM GPFN → MFN information (G to M) is maintained by the hypervisor

Direct paging: Page fault handling



P1 access VA 4 ⟶ Page fault ⟶Who handles Page fault?

Direct paging: Page fault handling

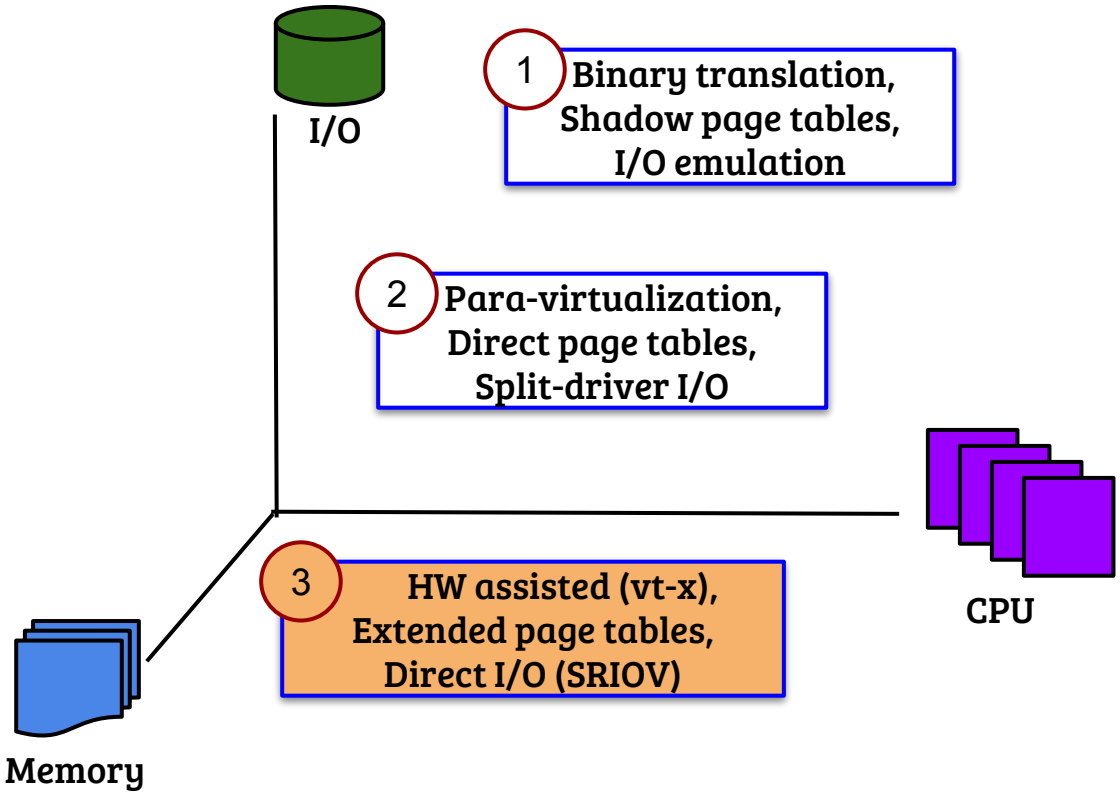


- Guest OS updates the page table (4 → 9) by invoking a hypercall

Direct paging: conclusion

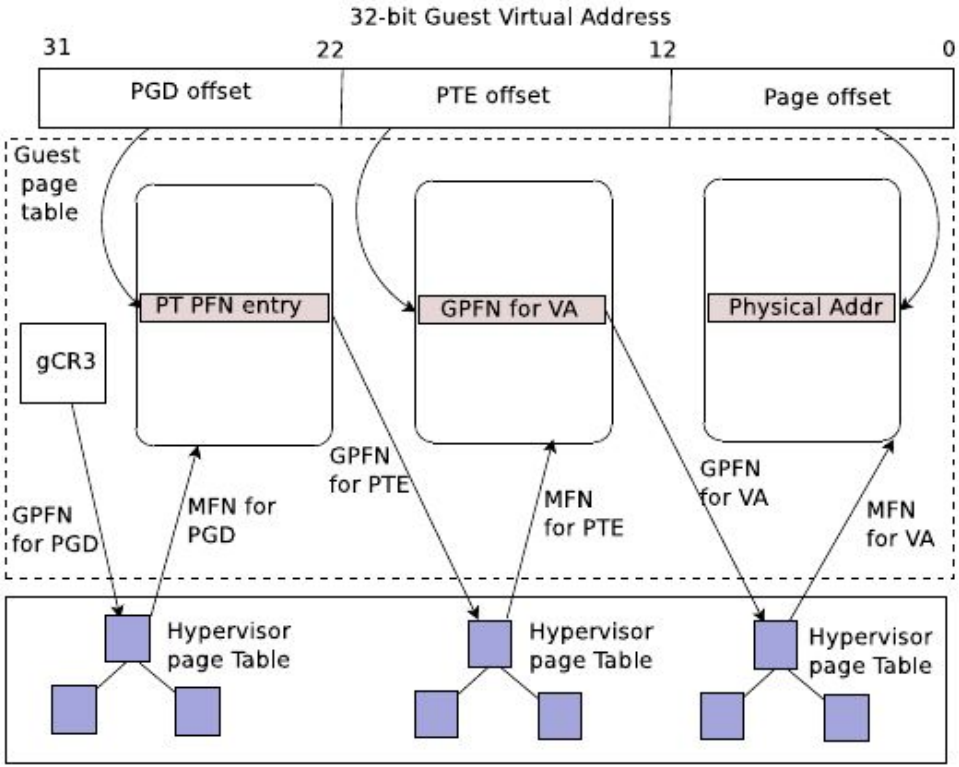
- Simple and efficient memory virtualization in expense of equivalence
- The guest read access and the hardware access to page table does not involve the hypervisor
- Possible issues
  - The guest OS is aware of the physical addresses
  - Hypervisor must keep track of “MFN usage type” to track the page table pages
- Optimizations
  - Batched updates --- efficient mmap implementation
  - Reuse page table pages to avoid overheads to ensure that PTs readonly

Overview of virtualization approaches



- Agenda for today's lecture: Memory virtualization
- Software only techniques: shadow page tables and direct page tables
- Hardware assisted: nested/extended page tables

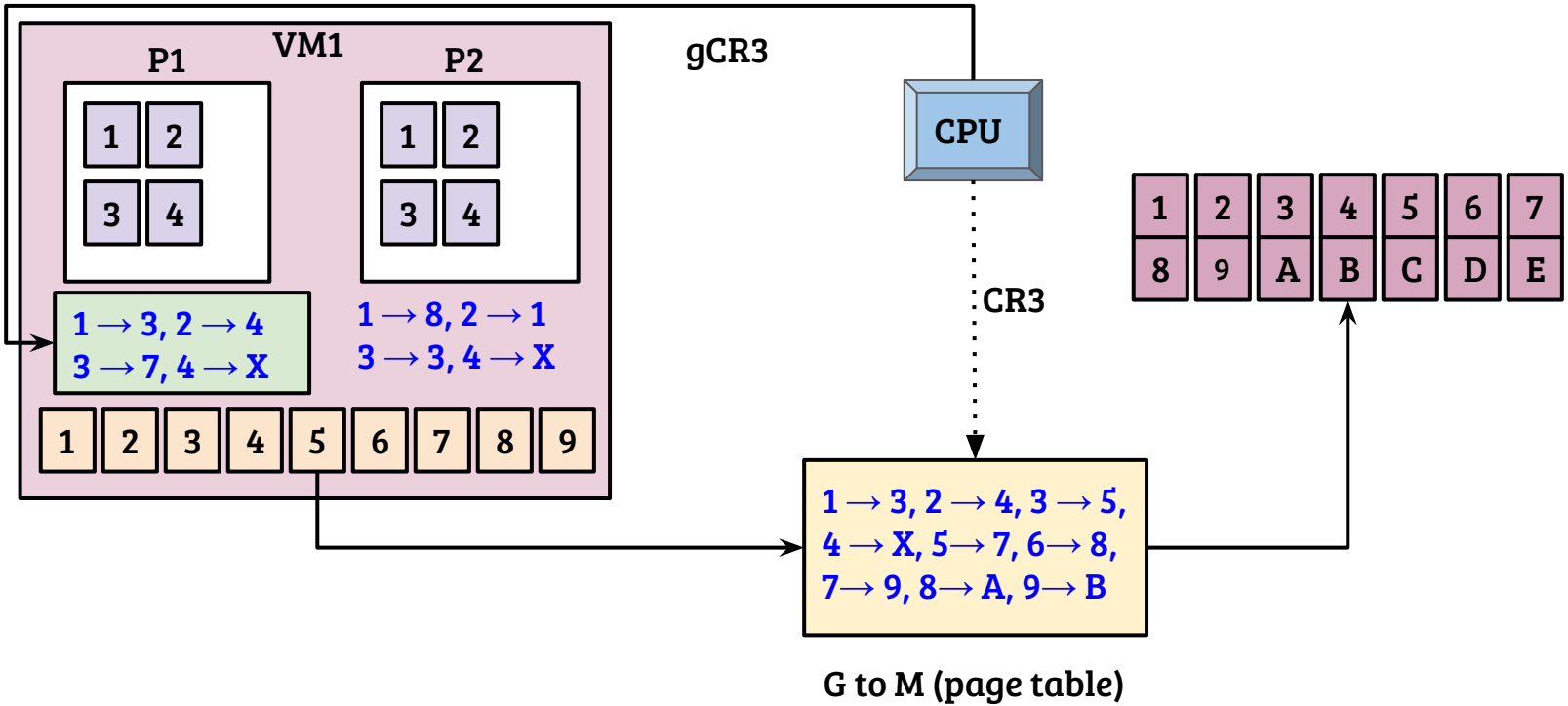
Nested/Extended page tables [1,2]



- Paging structures replicated in the hardware --- nested page walk
- In non-root mode, CR3 == gCR3
- gCR3 is used to maintain per-process virtual to guest PFN mapping
- Hypervisor maintains the PFN to MFN mapping in the second level translation tables

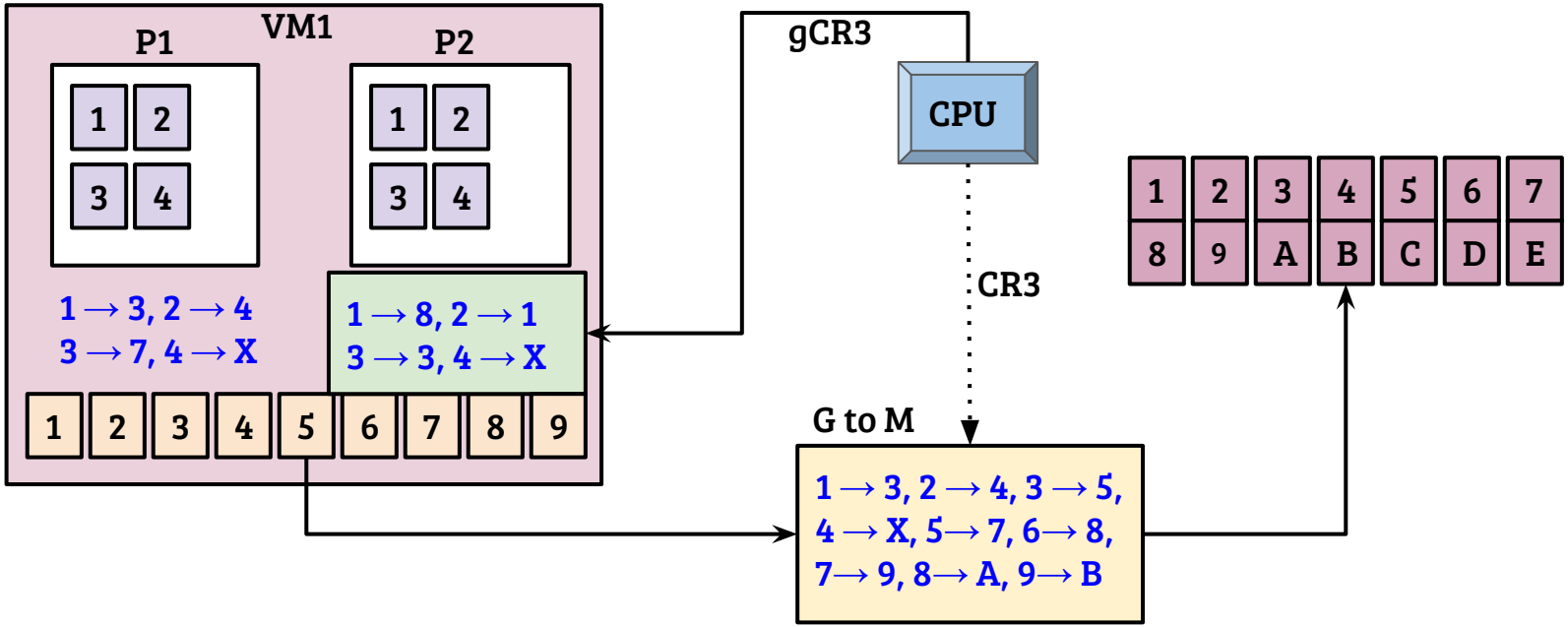
1. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf
2. http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf

H/W assisted paging: Basic working



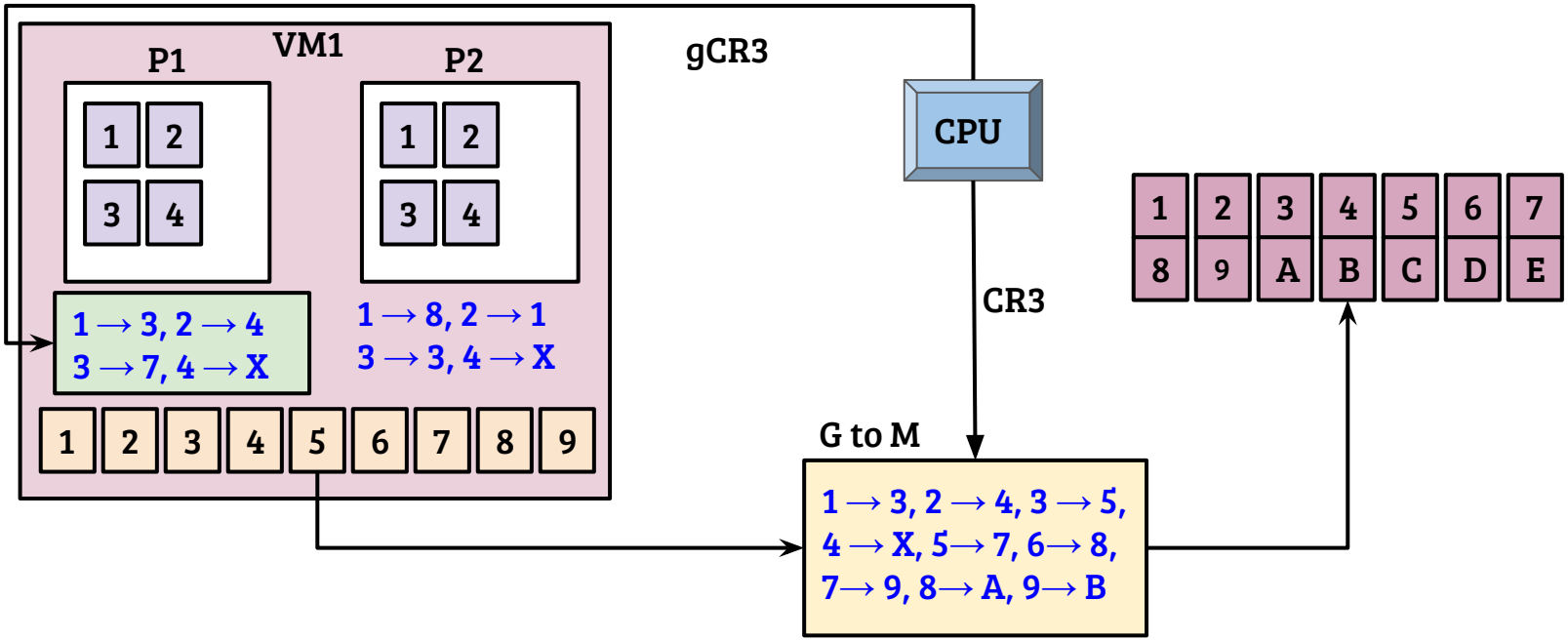
- gCR3 and CR3 are part of guest and host state areas of VMCS structure
- Loaded on VMEntry and VMExit

H/W assisted paging: page table management



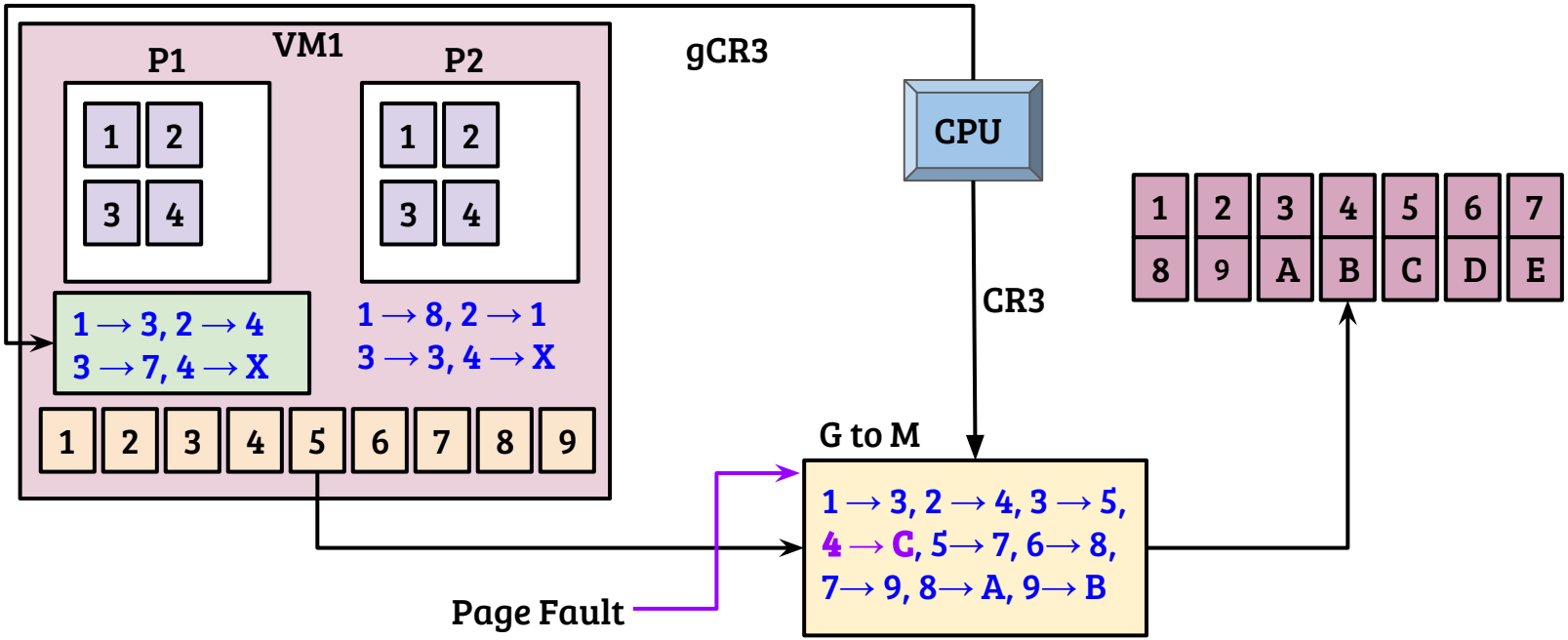
- Intra-VM context switch does not require hypervisor involvement
- What about page fault handling?

H/W assisted paging: Page fault handling



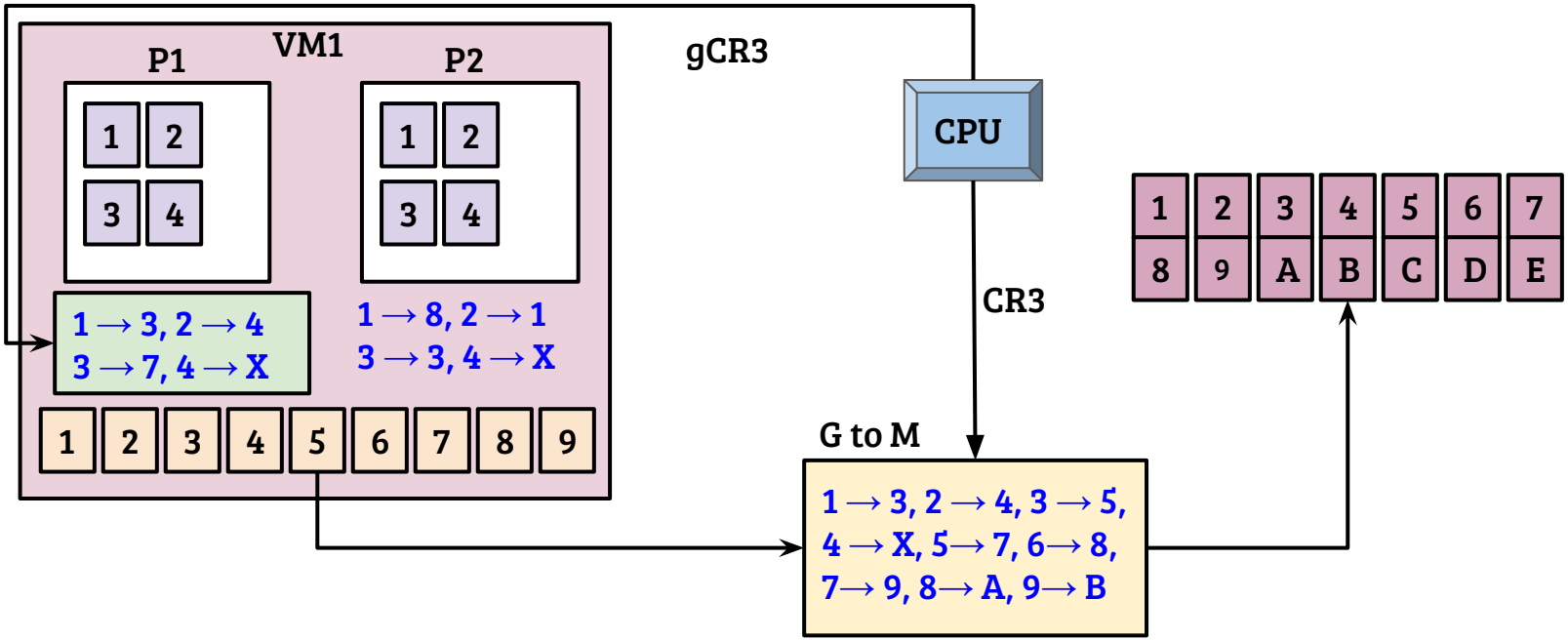
P1 access virtual address 2, how the page fault handled?

H/W assisted paging: Page fault handling



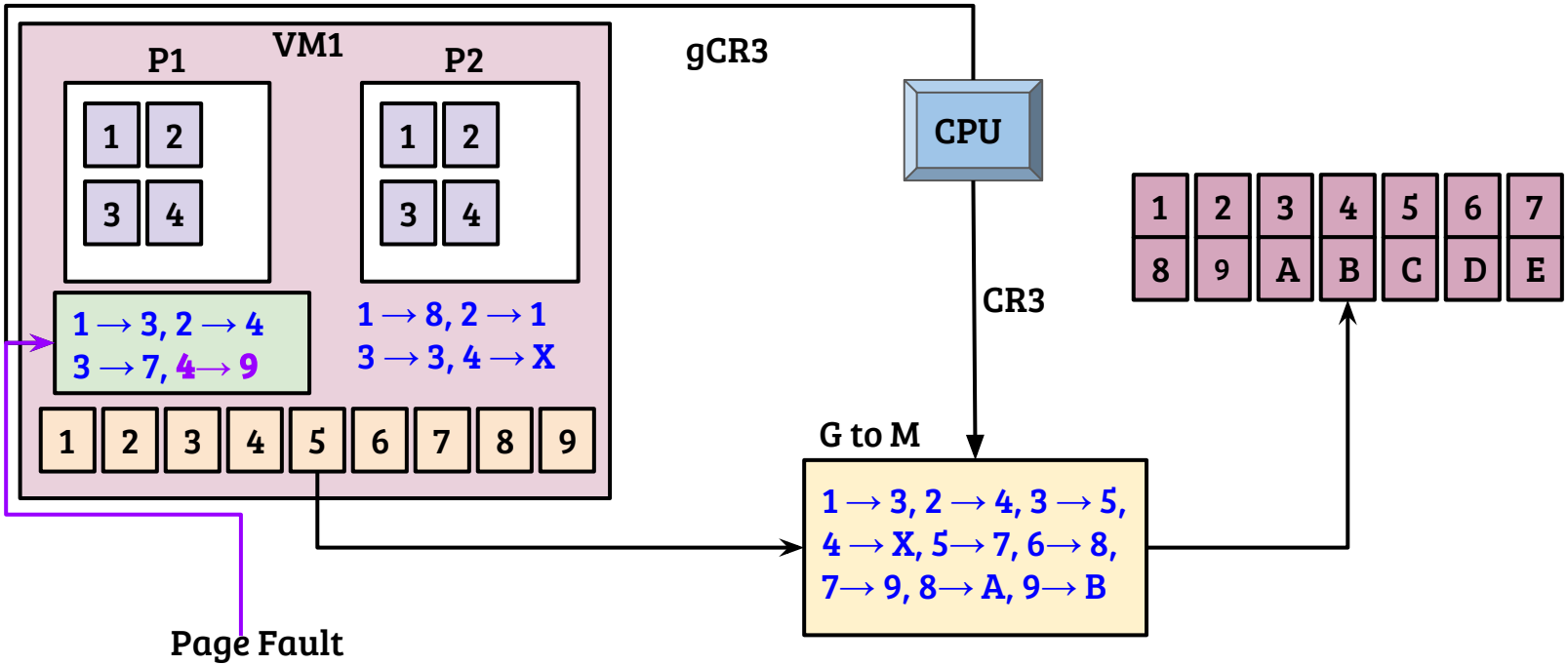
Page fault handled at the hypervisor in a guest OS transparent manner

H/W assisted paging: Page fault handling



P1 access virtual address 4, how the page fault handled?

H/W assisted paging: Page fault handling



Page fault handled by guest OS ⟶ no VMExit and hypervisor involvement

Nested paging: conclusion

- Architecture supported equivalence and resource control
- +ves
  - No VMExit, cumbersome page table sync
  - GPFN to MFN mapping is almost fixed
- -ves
  - Costly memory translation in case of TLB miss
  - For a N-level page table, how many memory walks are required to perform a single translations in the worst case?