

FU05 Computer Architecture

12. Memory Hierarchy: Virtual Memory (メモリ階層化: 仮想メモリ)

Ben Abdallah Abderazek

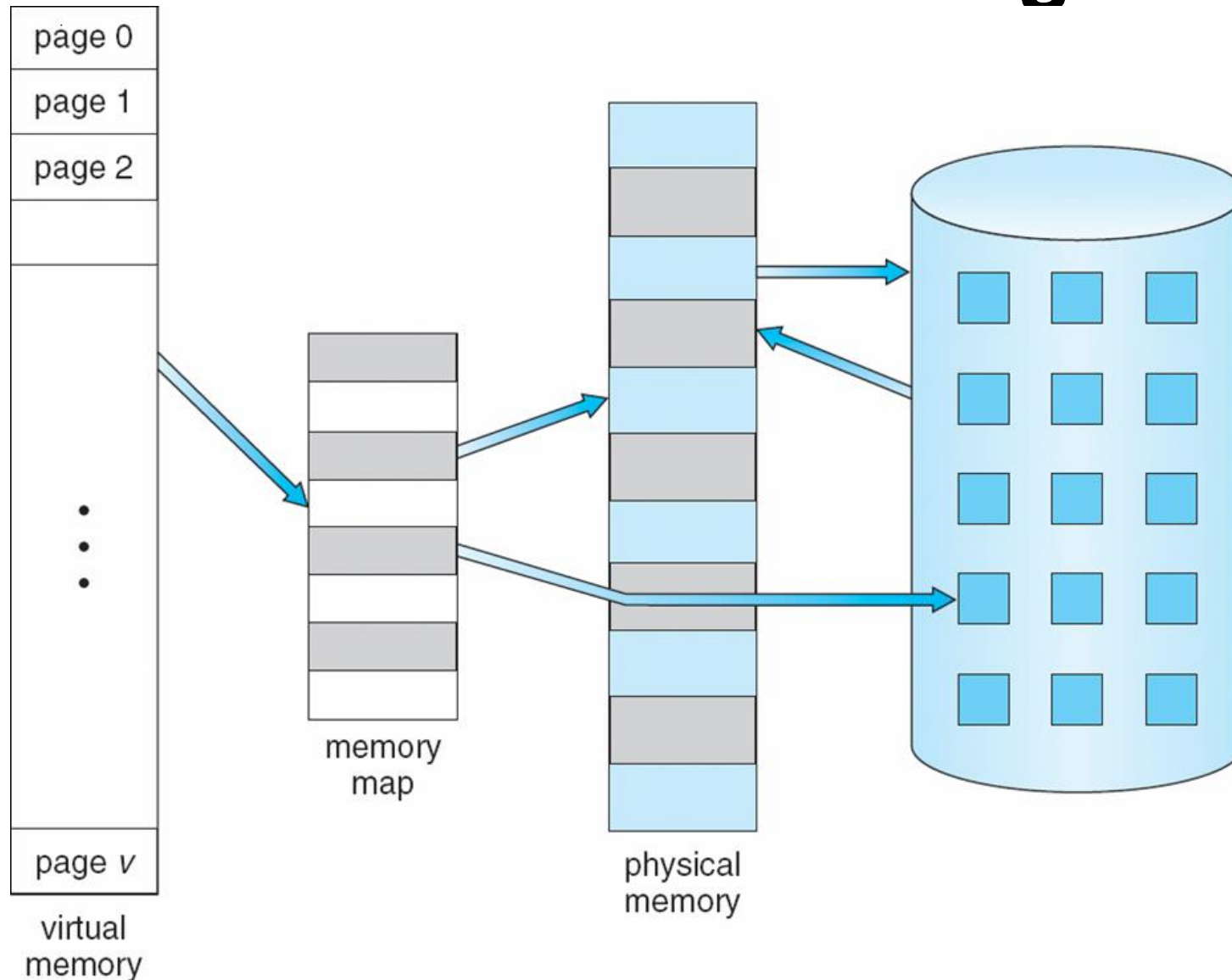
E-mail: benab@u-aizu.ac.jp

Virtual Memory

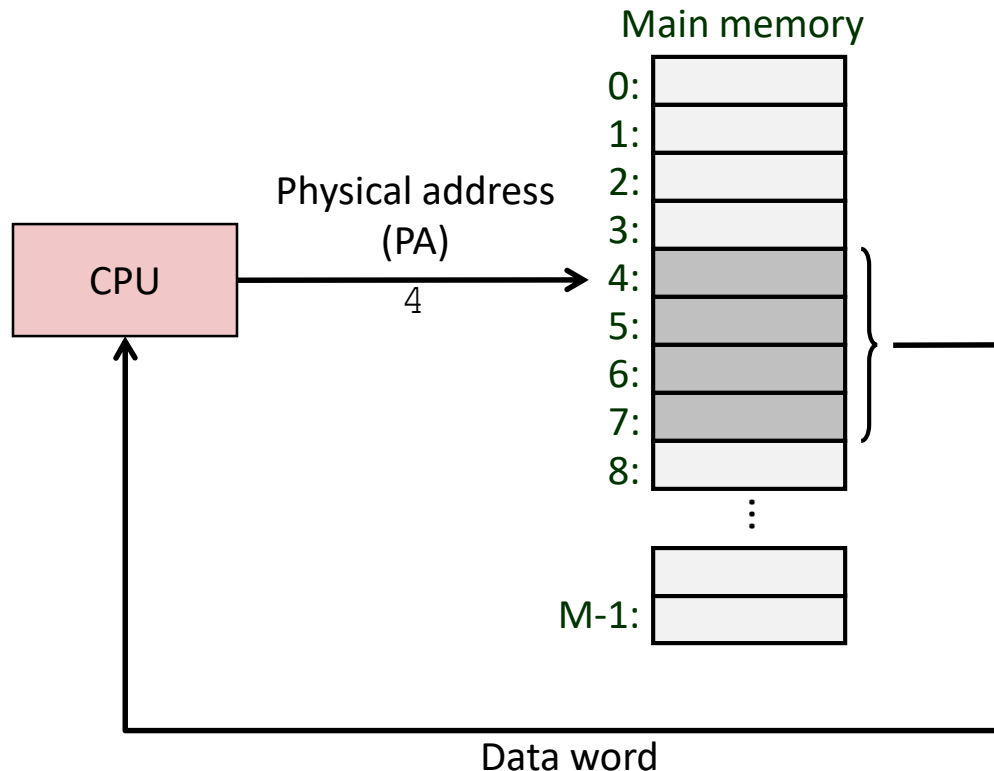
仮想メモリ

- Use main memory as a “cache” for secondary (disk) storage
 - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
 - Each gets a private virtual address space holding its frequently used code and data
 - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
 - VM “block” is called a page
 - VM translation “miss” is called a page fault

Virtual Addressing

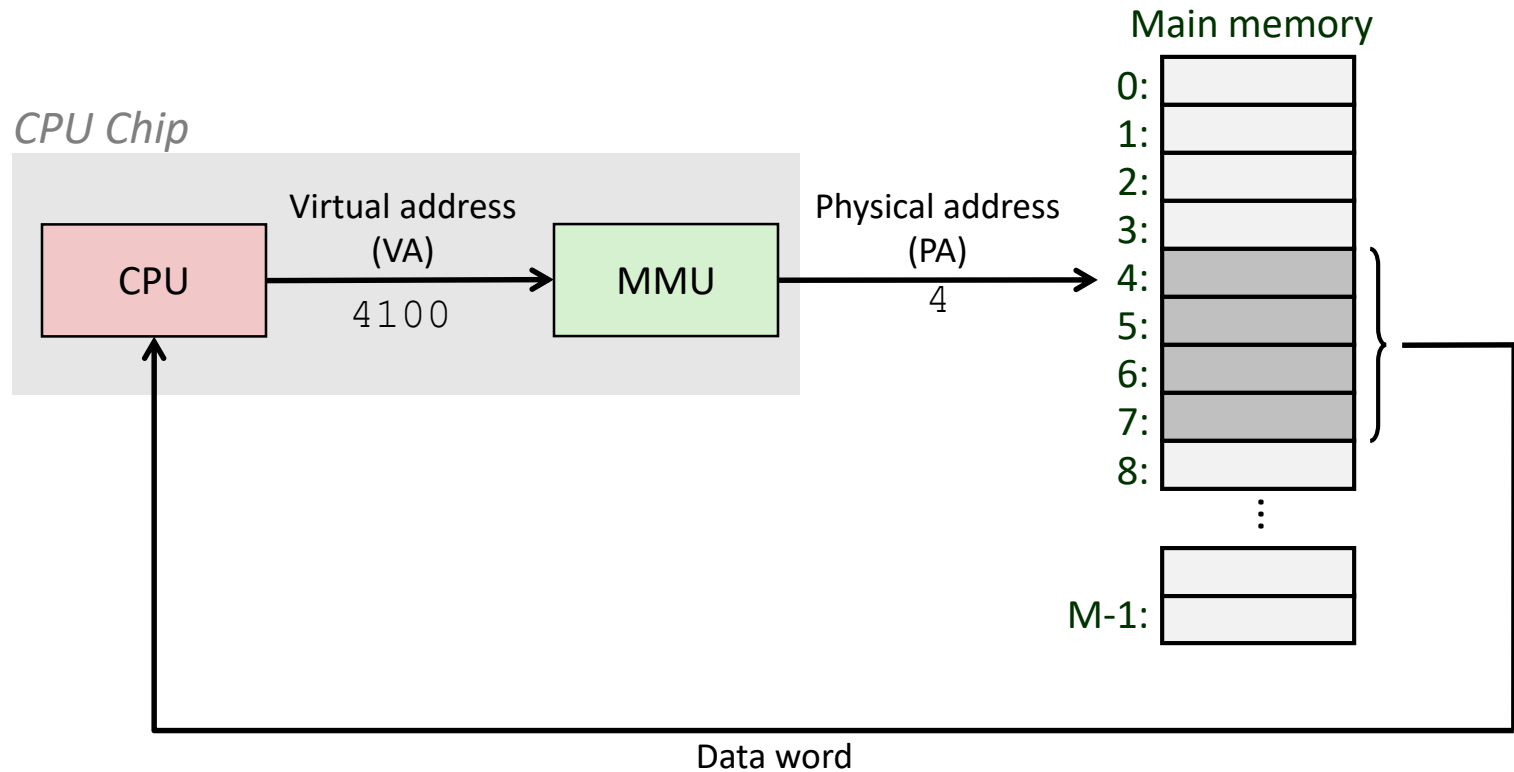


Physical Addressing



- Used in “simple” systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

Virtual Addressing



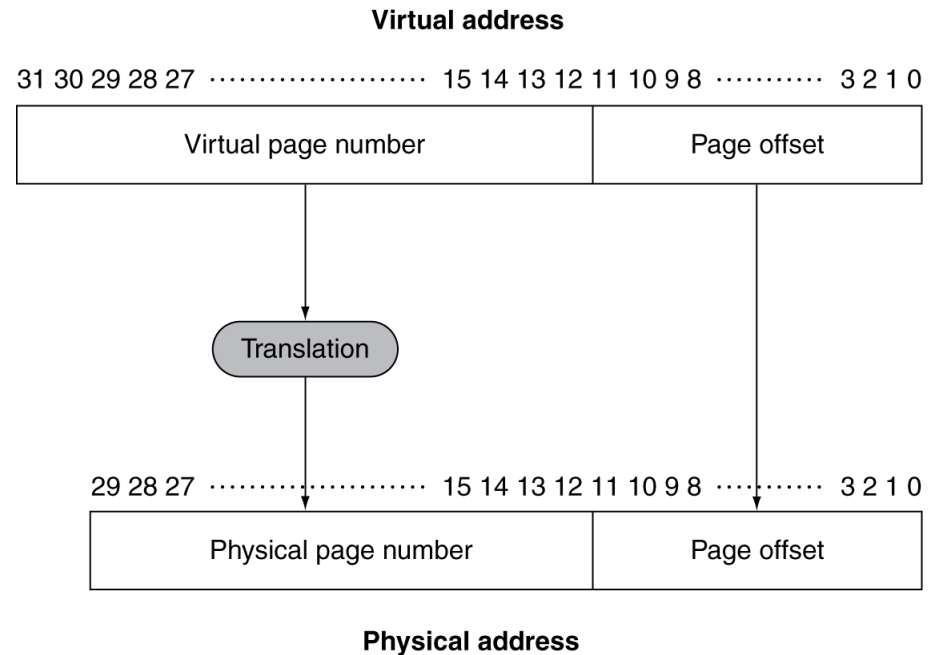
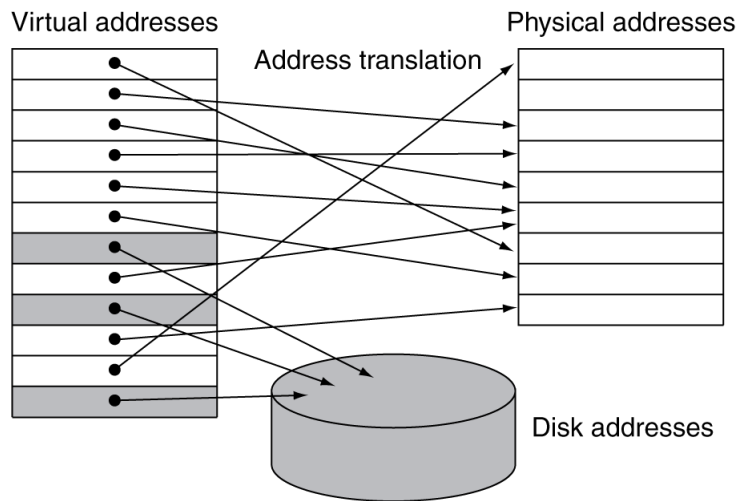
- Used in all modern servers, desktops, and laptops
- One of the great ideas in computer science

Why Virtual Memory (VM)?

- **Uses main memory efficiently**
 - Use DRAM as a cache for the parts of a virtual address space
- **Simplifies memory management**
 - Each process gets the same uniform linear address space
- **Isolates address spaces**
 - One process can't interfere with another's memory
 - User program cannot access privileged kernel information

Address Translation

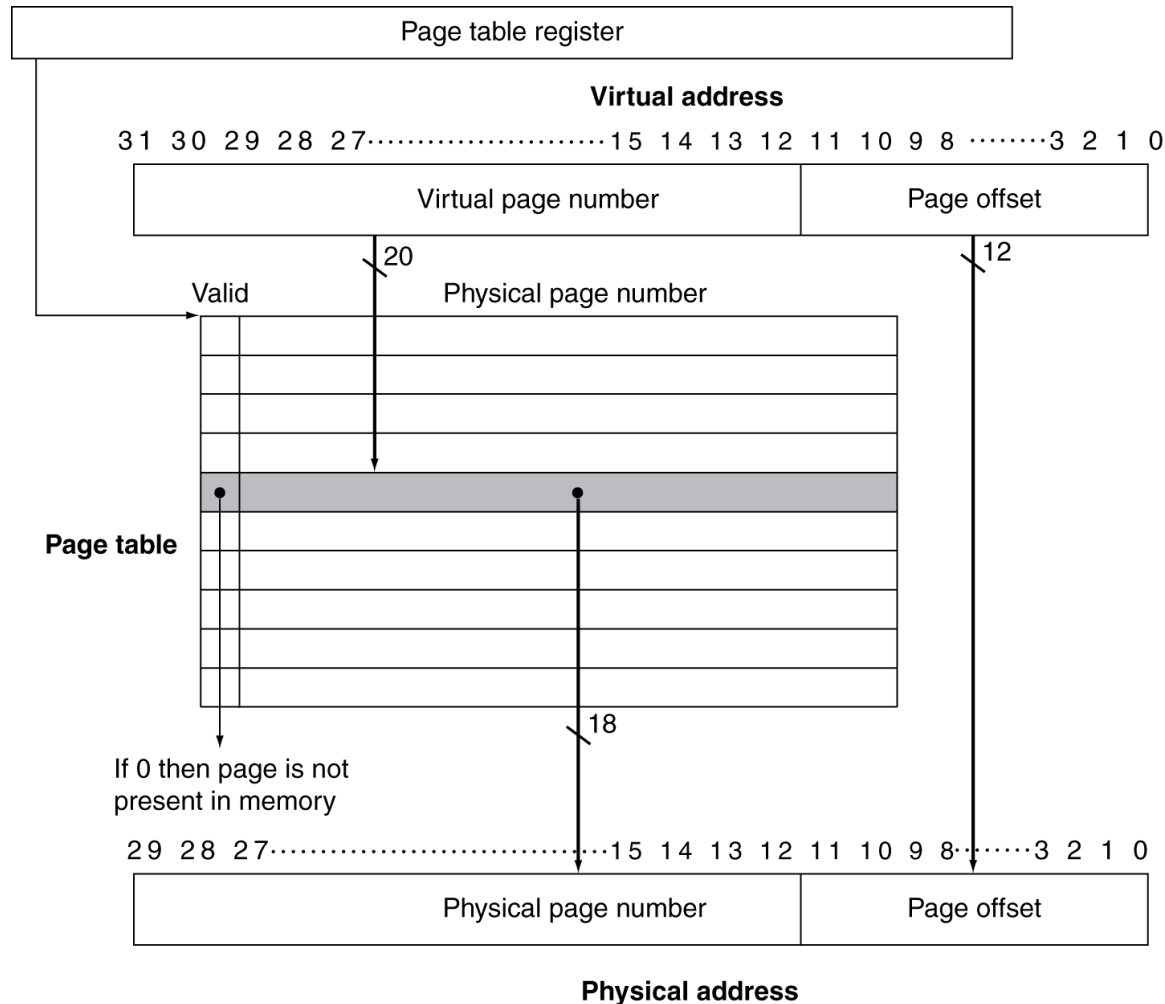
- Fixed-size pages (e.g., 4K)



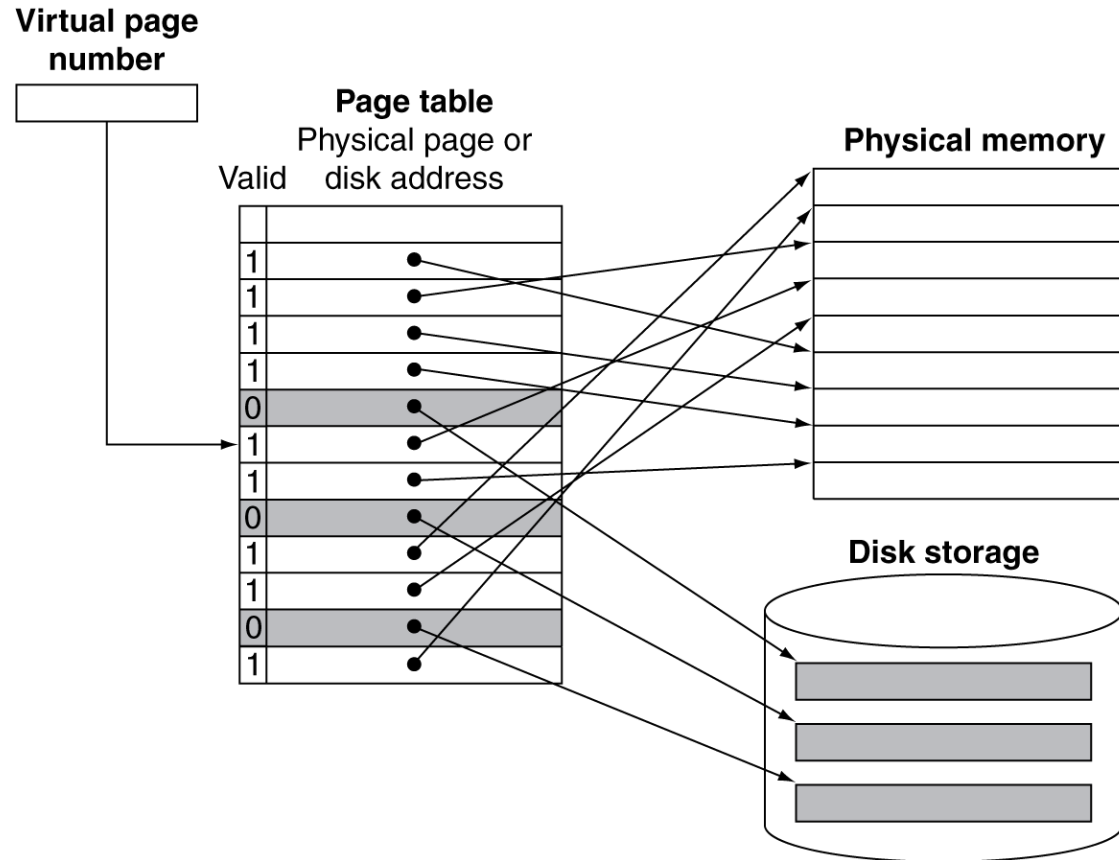
Page Tables

- Stores placement information
 - Array of page table entries, indexed by virtual page number
 - Page table register in CPU points to page table in physical memory
- If page is present in memory
 - PTE stores the physical page number
 - Plus other status bits (referenced, dirty, ...)
- If page is not present
 - PTE can refer to location in swap space on disk

Translation Using a Page Table

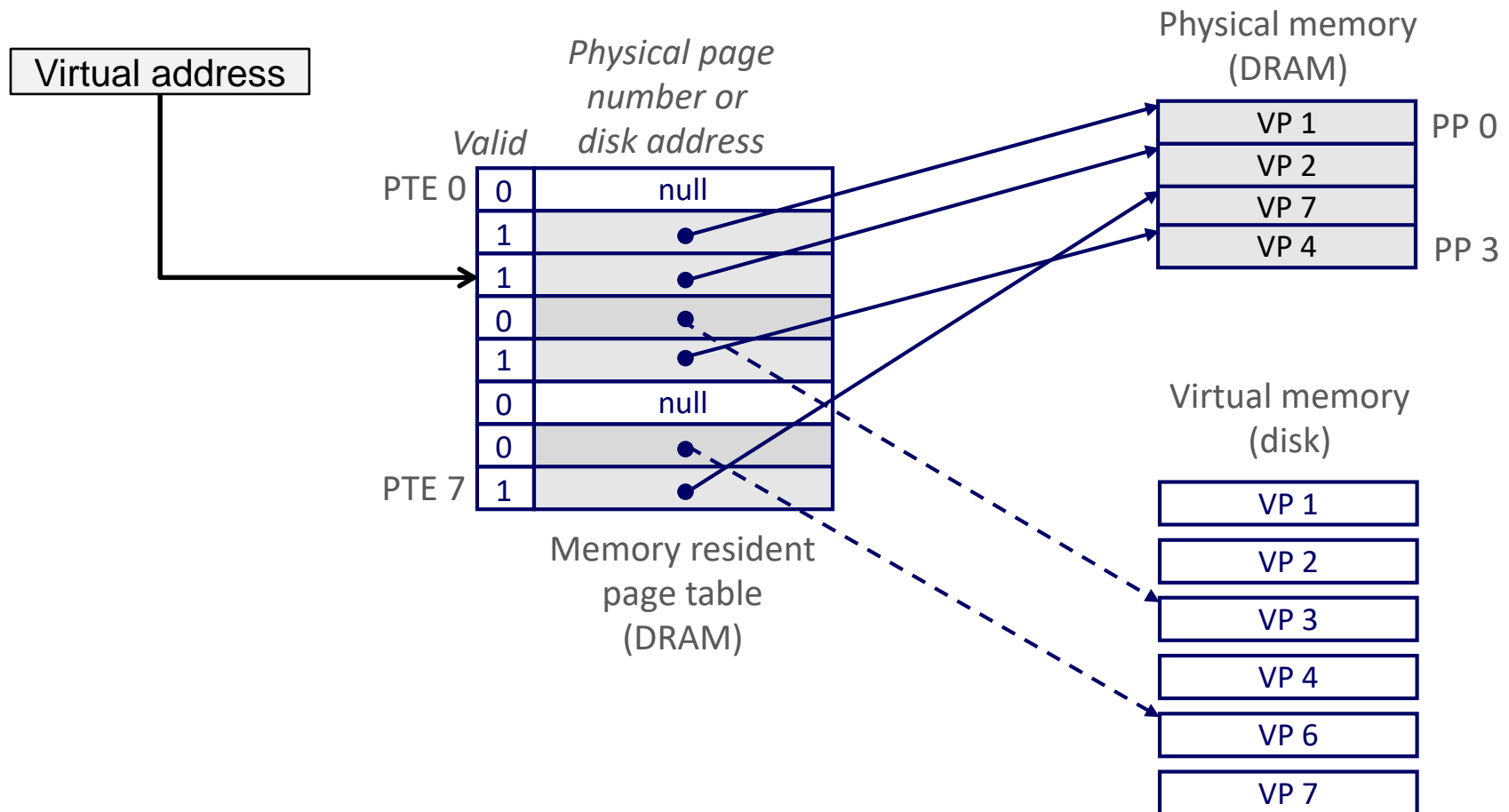


Mapping Pages to Storage



Page Hit

- **Page hit:** reference to VM word that is in physical memory (DRAM cache hit)



Page Fault

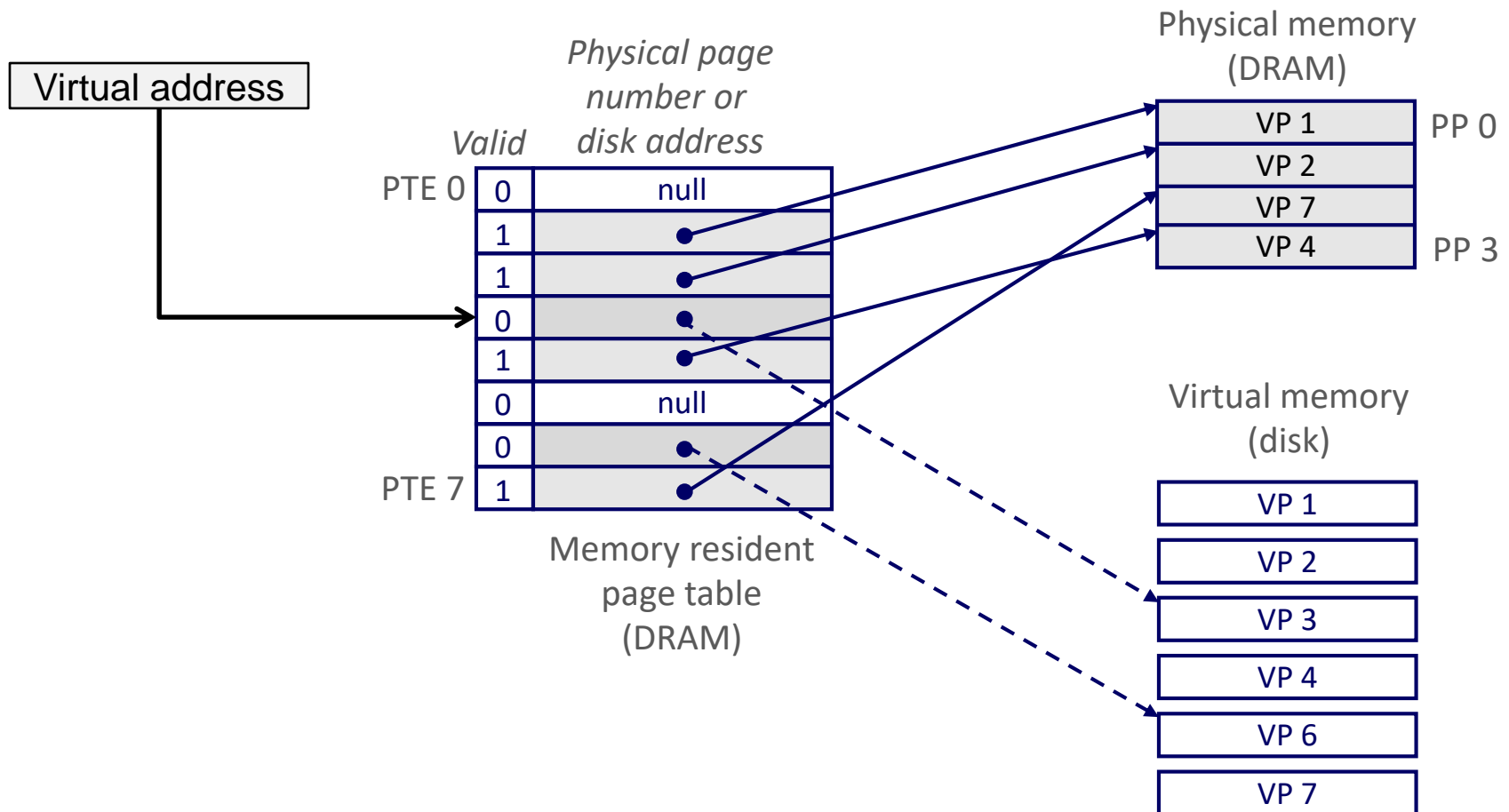
- If there is a reference to a page, first reference to that page will trap to operating system:

page fault

1. Operating system looks at another table to decide:
 - Invalid reference \Rightarrow abort
 - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **V**
6. Restart the instruction that caused the page fault

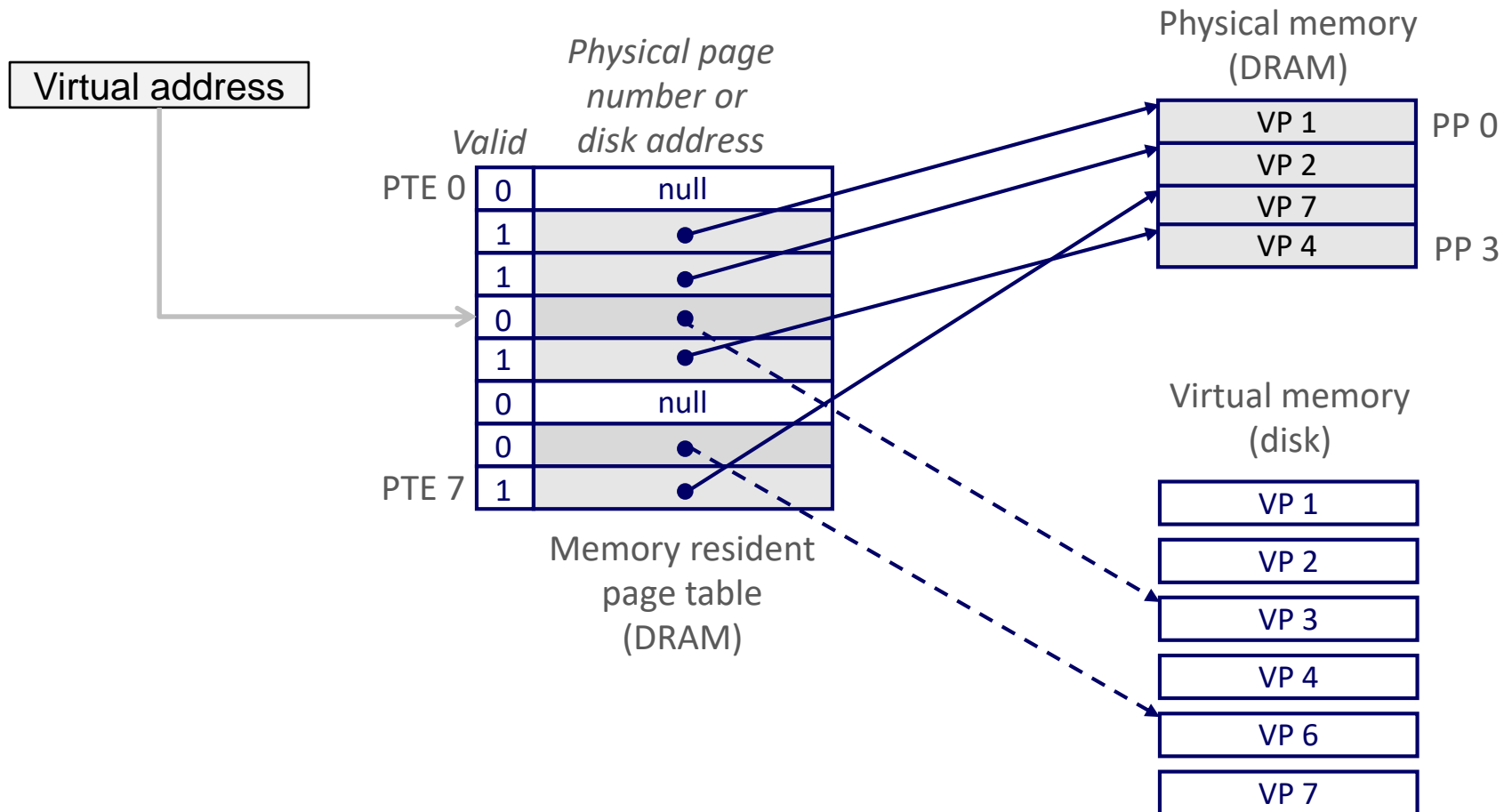
Page Fault

- **Page fault:** reference to VM word that is not in physical memory (DRAM cache miss)



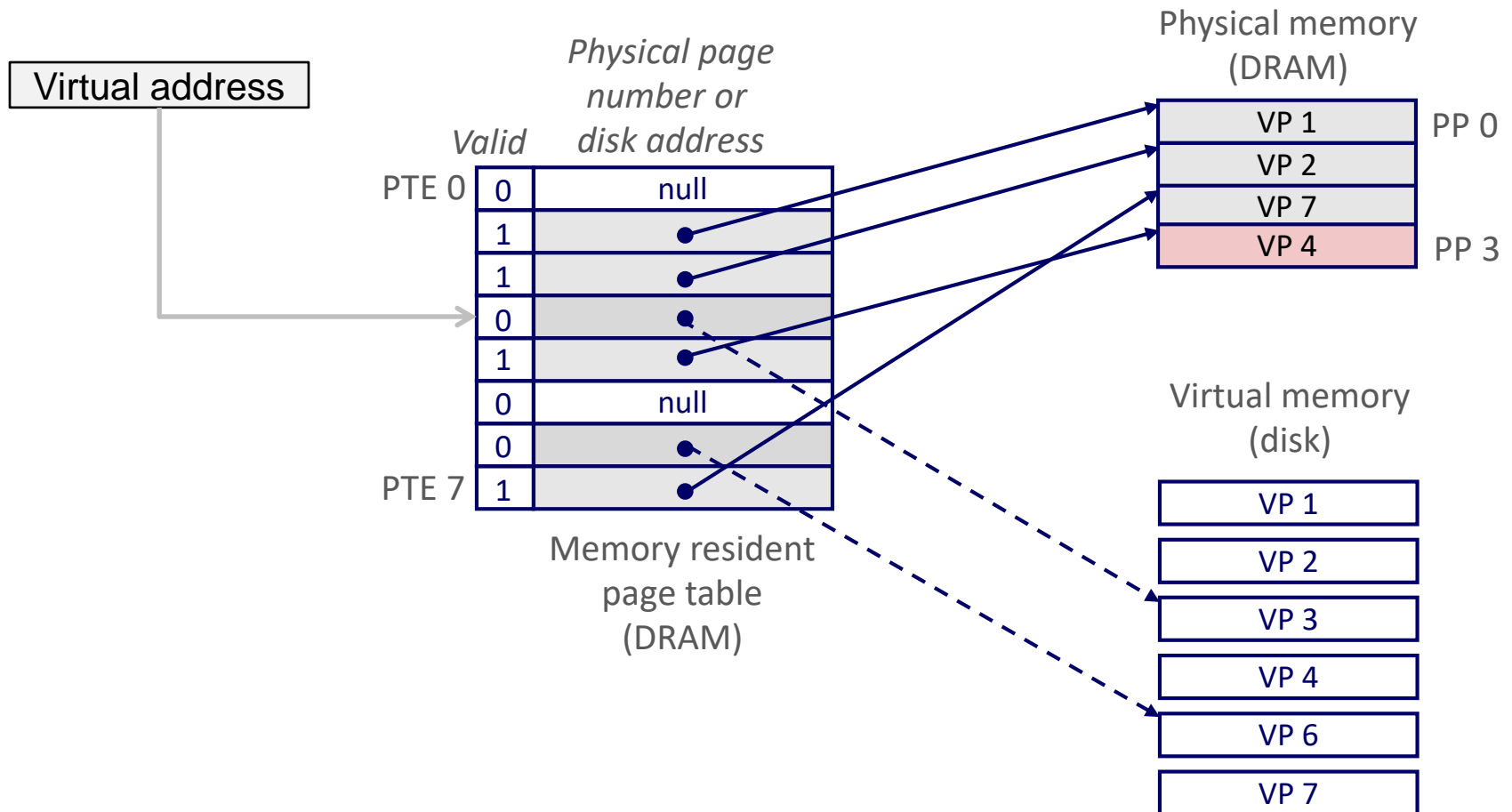
Handling Page Fault

- Page miss causes page fault (an exception)



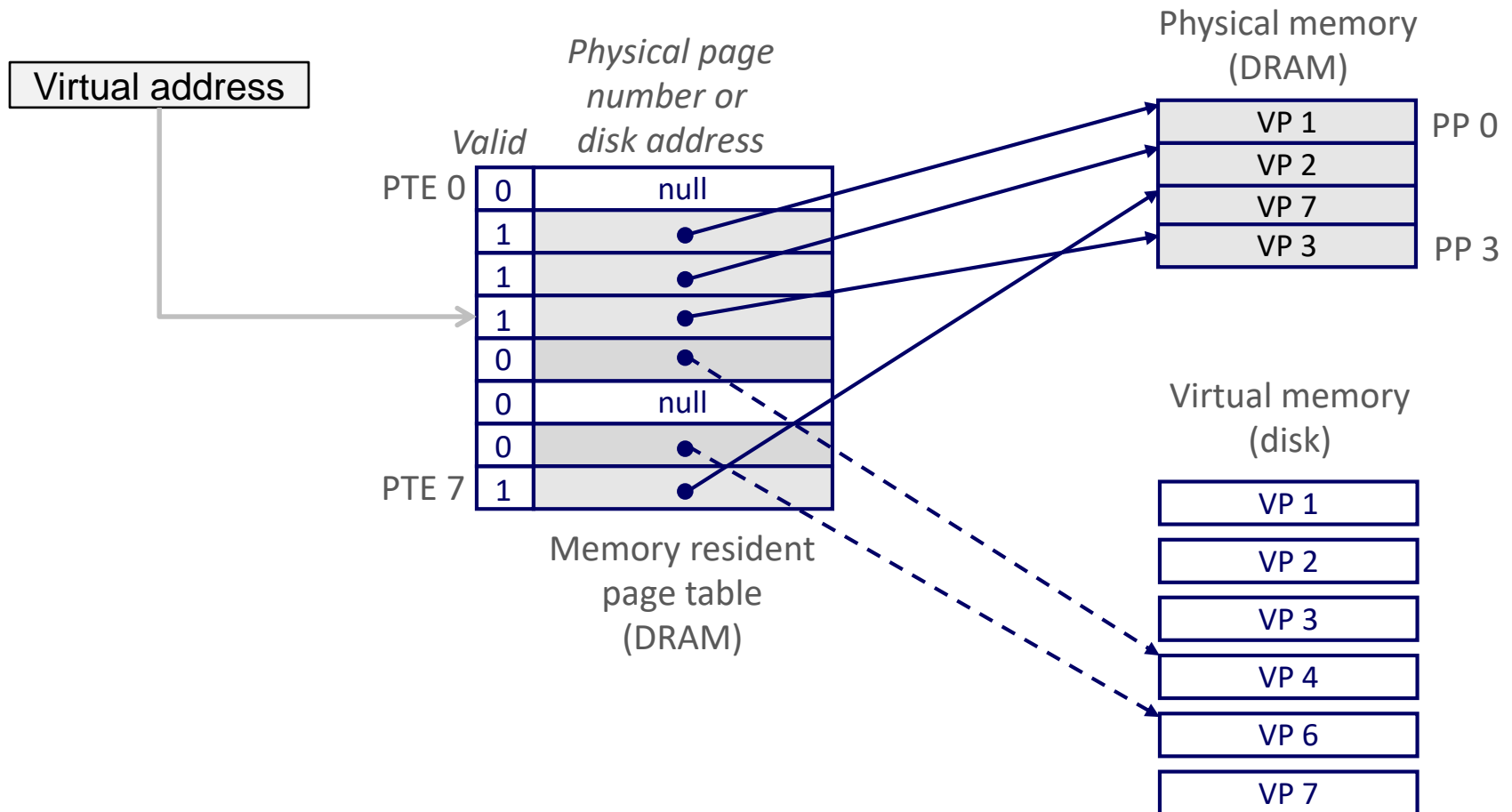
Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)



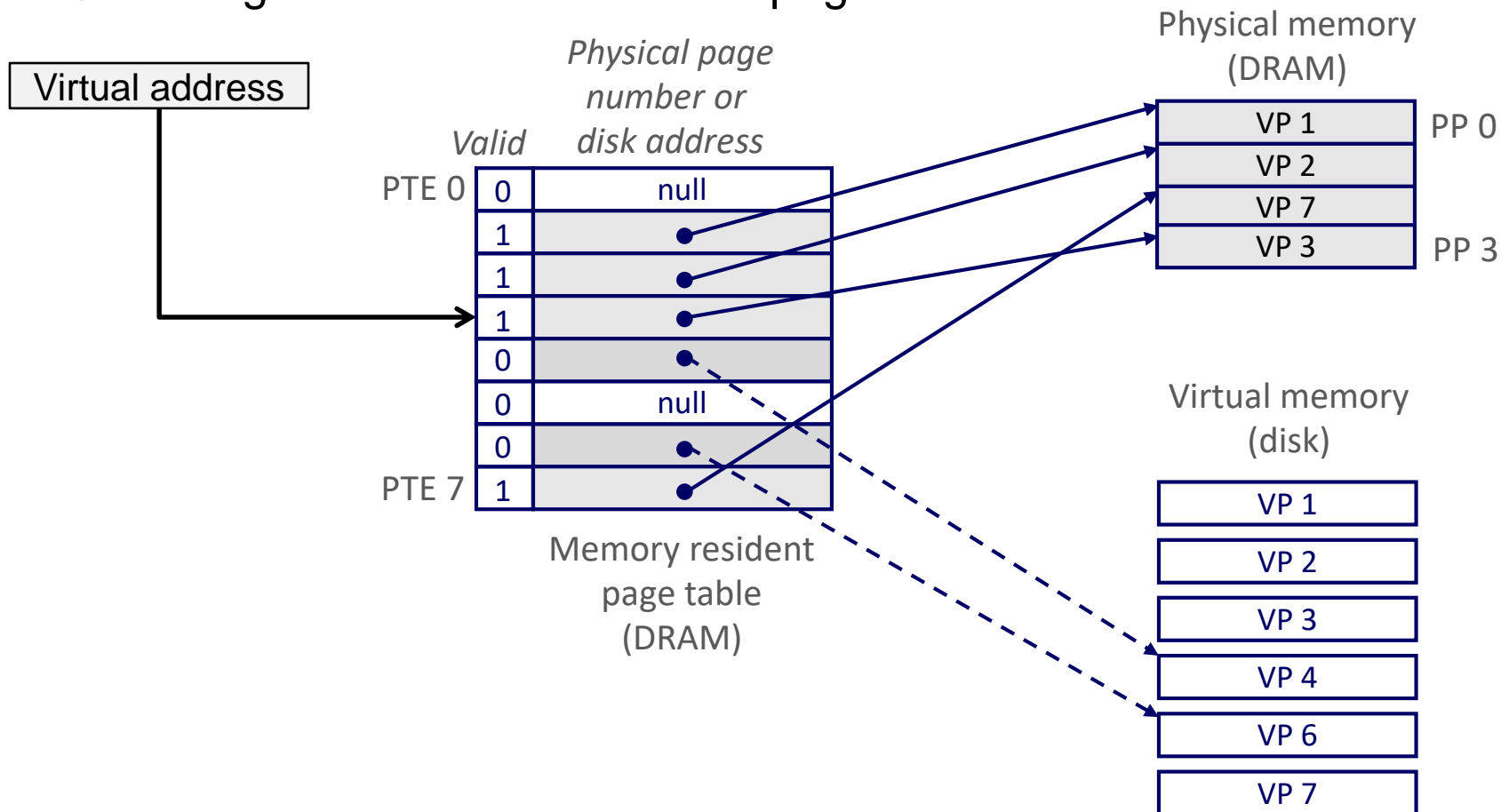
Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)

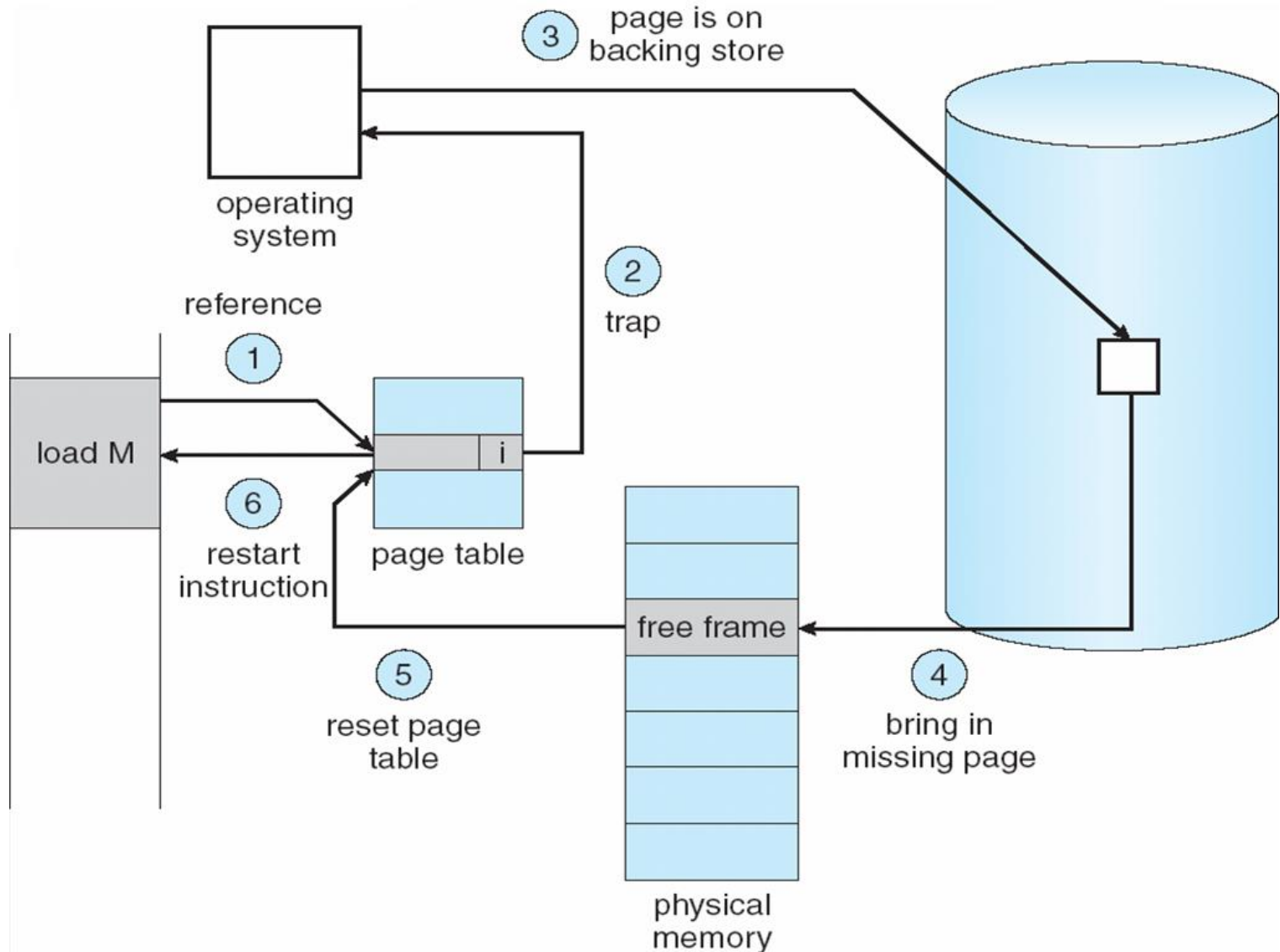


Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



Summary of Handling a Page Fault



Page Fault Penalty

- On page fault, the page must be fetched from disk
 - Takes millions of clock cycles
 - Handled by OS code
- Try to minimize page fault rate
 - Fully associative placement
 - Smart replacement algorithms

Locality to the Rescue Again!

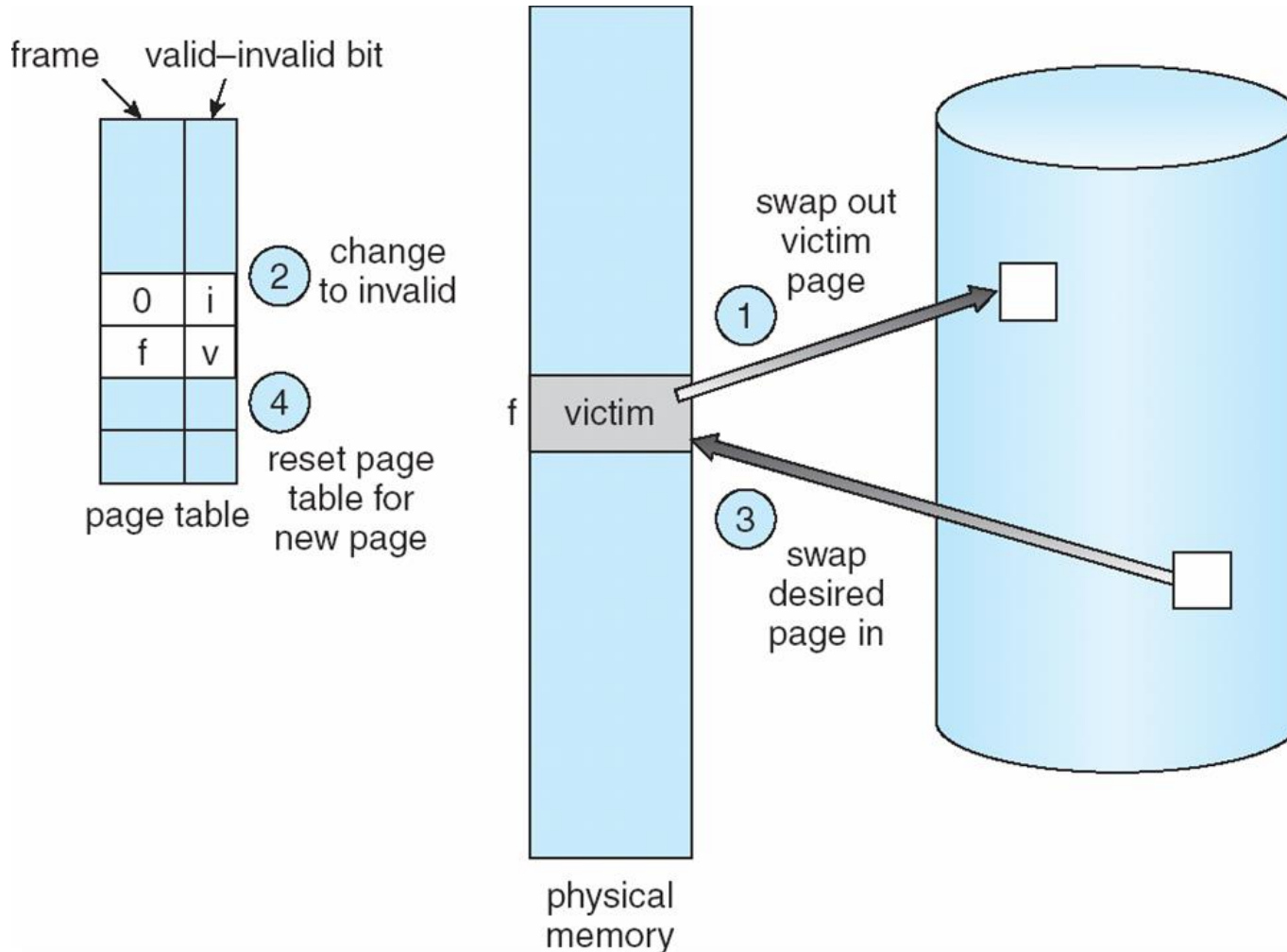
- Virtual memory works because of locality
- At any point in time, programs tend to access a set of active virtual pages called the *working set*
 - Programs with better temporal locality will have smaller working sets
- If (working set size < main memory size)
 - Good performance for one process after compulsory misses
- If (SUM(working set sizes) > main memory size)
 - *Thrashing*: Performance meltdown where pages are swapped (copied) in and out continuously

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
 - If there is a free frame, use it
 - If there is no free frame, use a page replacement algorithm to select a **victim** frame

Bring the desired page into the (newly) free frame; update the page and frame tables
3. Restart the process

Page Replacement



Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

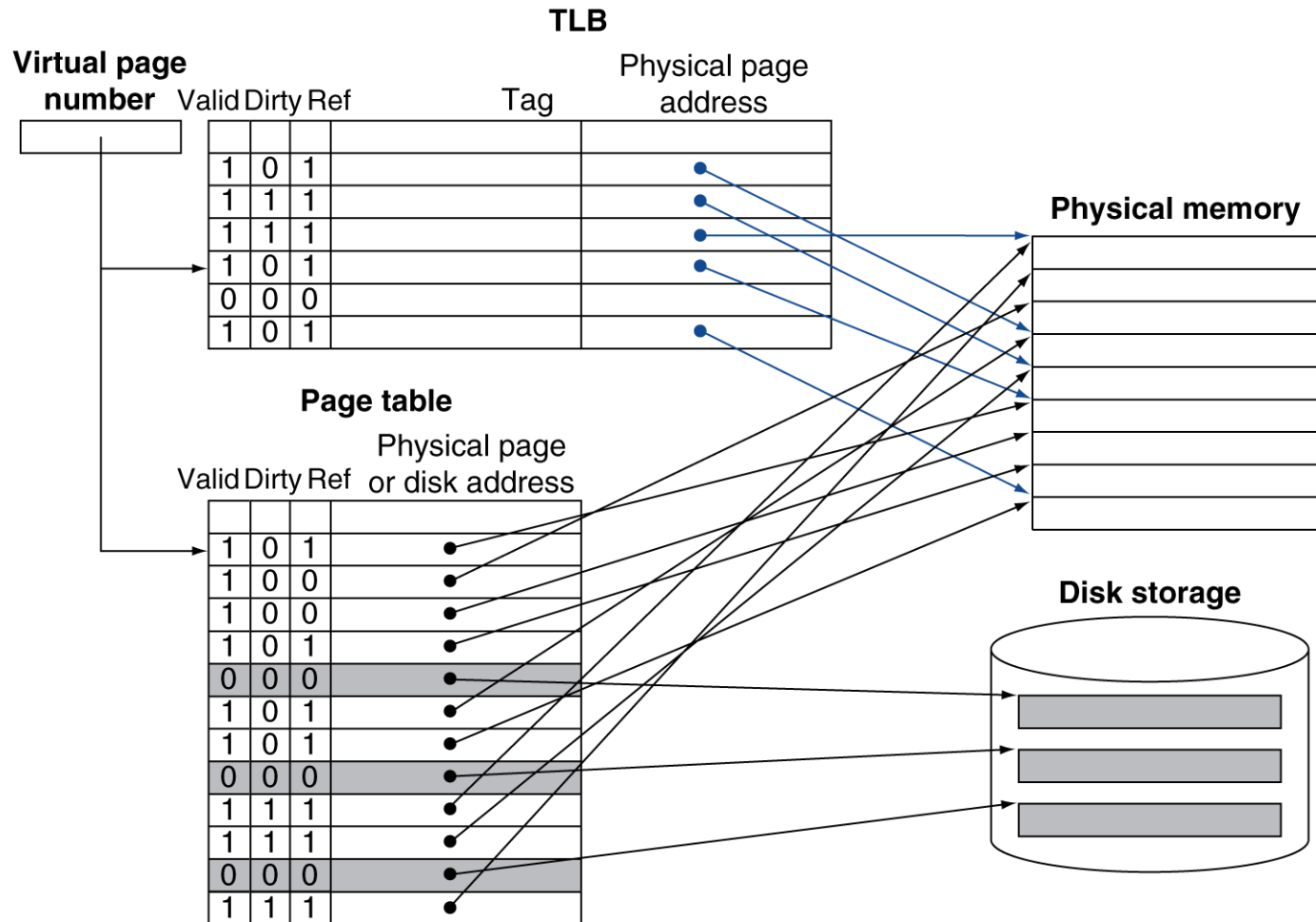
1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

- Counter implementation
 - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
 - When a page needs to be changed, look at the counters to determine which are to change

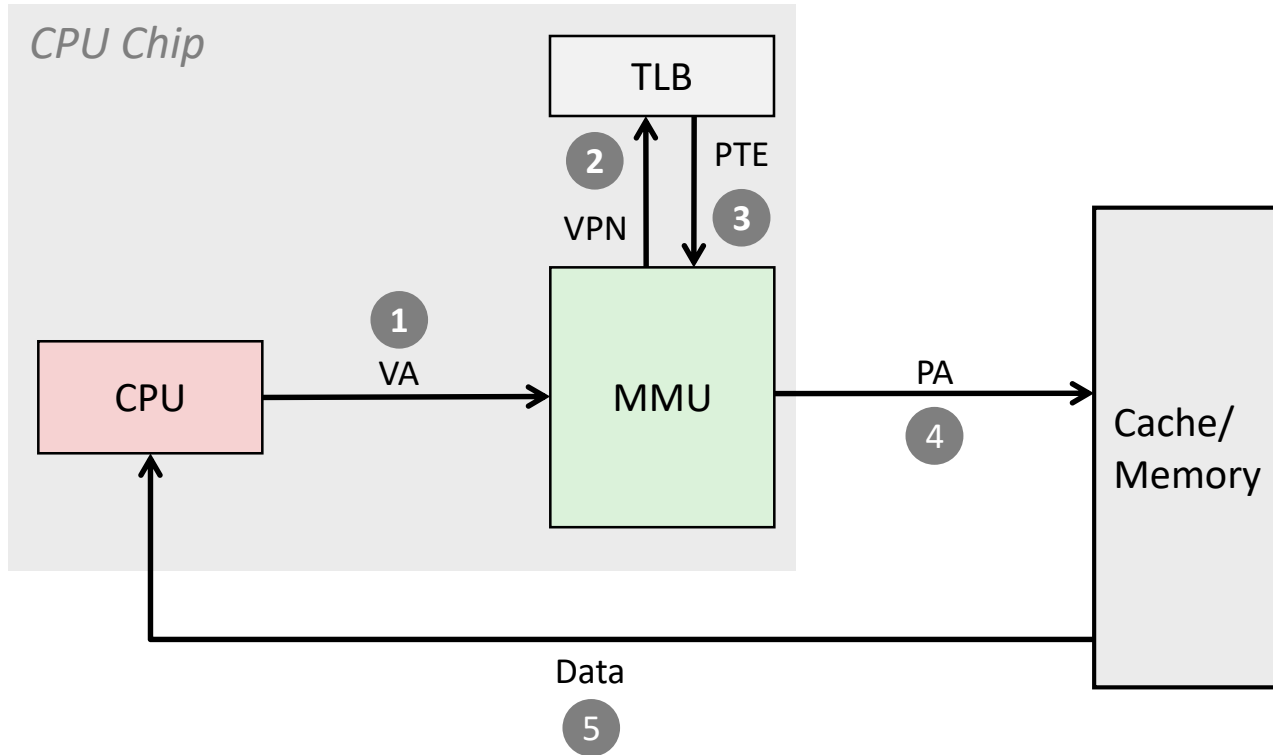
Fast Translation Using a TLB

- Address translation would appear to require extra memory references
 - One to access the PTE
 - Then the actual memory access
- But access to page tables has good locality
 - So use a fast cache of PTEs within the CPU
 - Called a Translation Look-aside Buffer (TLB)
 - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
 - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB Hit

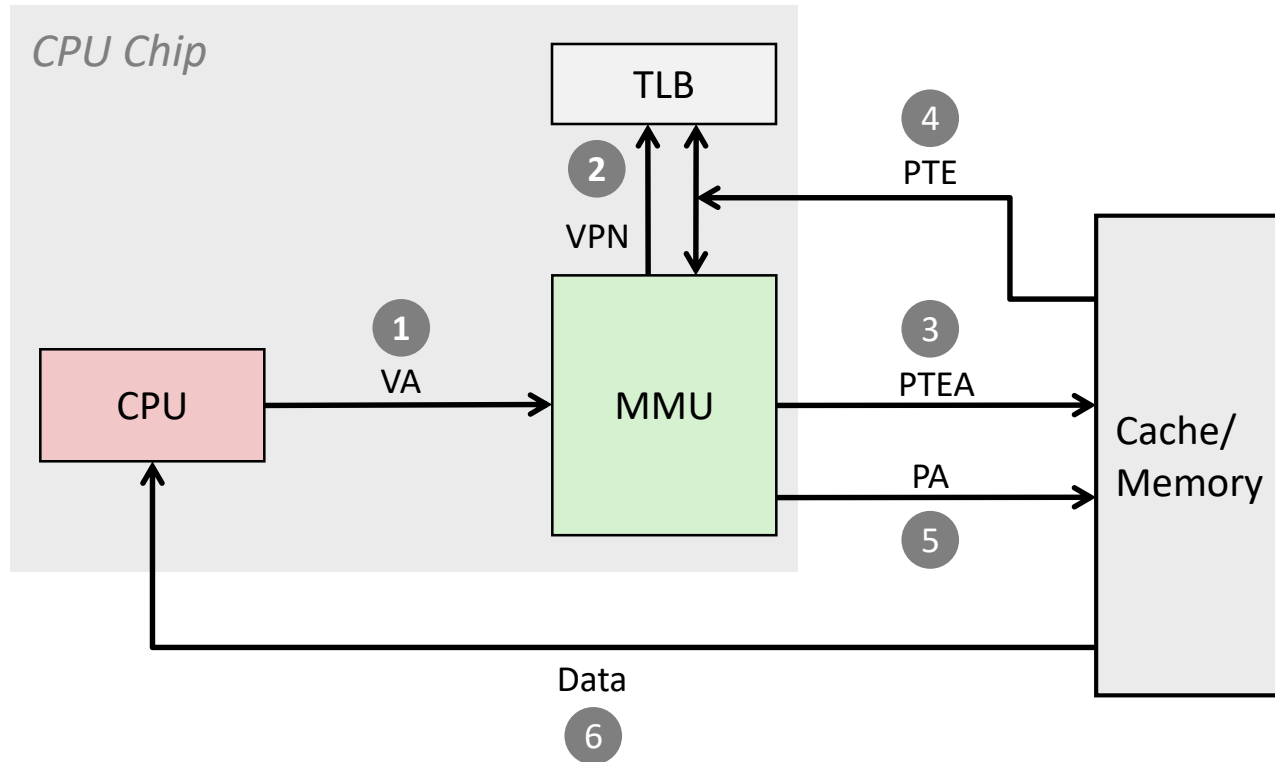


A TLB hit eliminates a memory access

TLB Misses

- If page is in memory
 - Load the PTE from memory and retry
 - Could be handled in hardware
 - Can get complex for more complicated page table structures
 - Or in software
 - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
 - OS handles fetching the page and updating the page table
 - Then restart the faulting instruction

TLB Miss



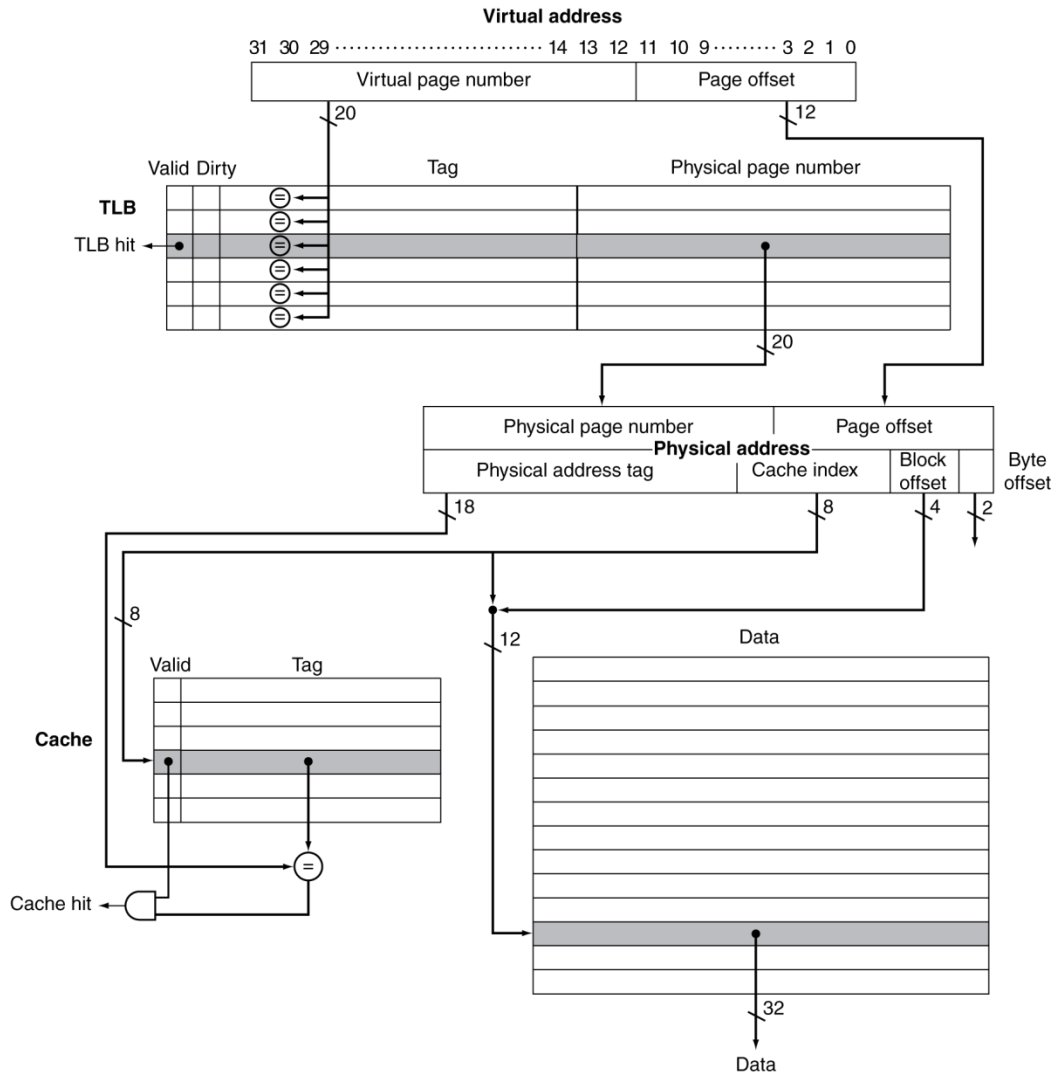
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

TLB Miss Handler

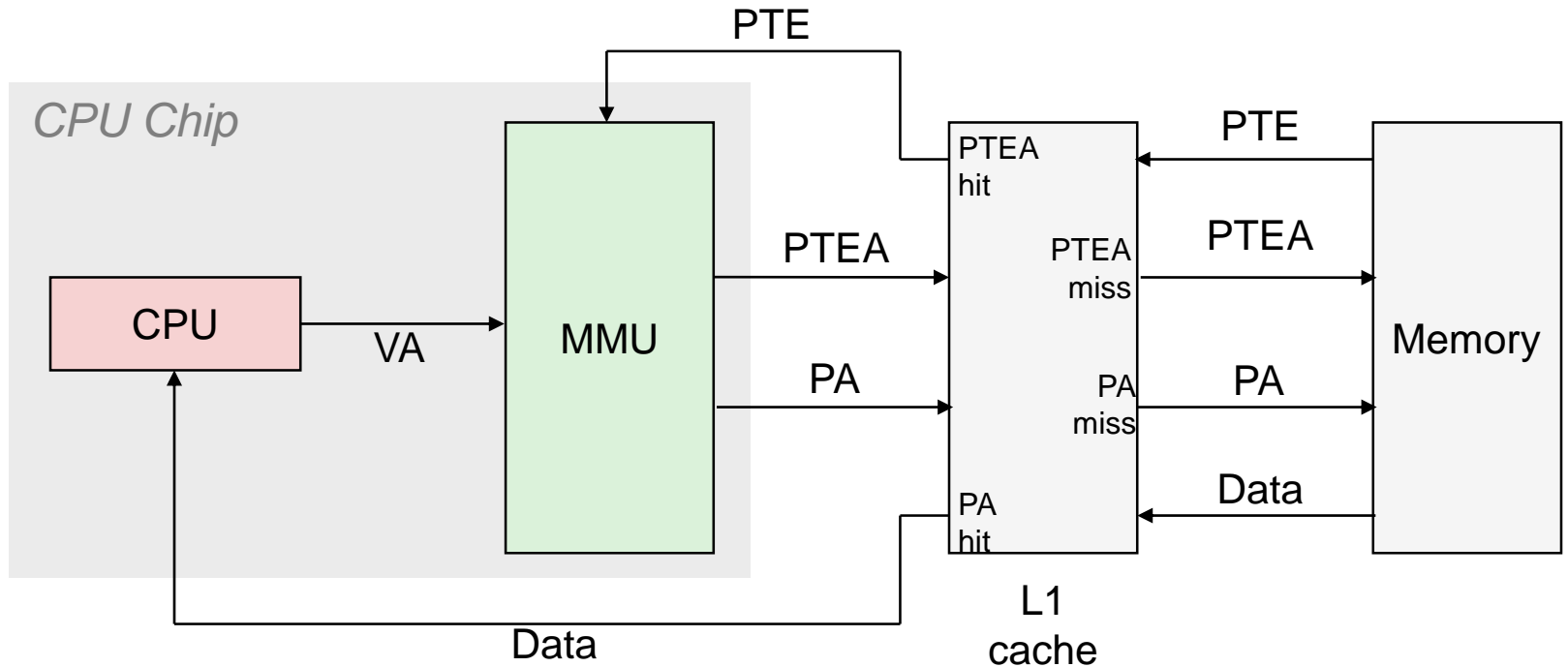
- TLB miss indicates
 - Page present, but PTE not in TLB
 - Page not present
- Must recognize TLB miss before destination register overwritten
 - Raise exception
- Handler copies PTE from memory to TLB
 - Then restarts instruction
 - If page not present, page fault will occur

TLB and Cache Interaction



- If cache tag uses physical address
 - Need to translate before cache lookup
- Alternative: use virtual address tag
 - Complications due to aliasing
 - Different virtual addresses for shared physical address

TLB and Cache Interaction



VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Summary

- Fast memories are small, large memories are slow
- Principle of locality
 - Programs use a small part of their memory space frequently
- Memory hierarchy
 - L1 cache \leftrightarrow L2 cache \leftrightarrow ... \leftrightarrow DRAM memory \leftrightarrow disk
- Memory system design is critical for multiprocessors