

CS330 - Computer Organization and Assembly Language Programming

Lecture 21 Virtual Memory

Professor : Mahmut Unan – UAB CS

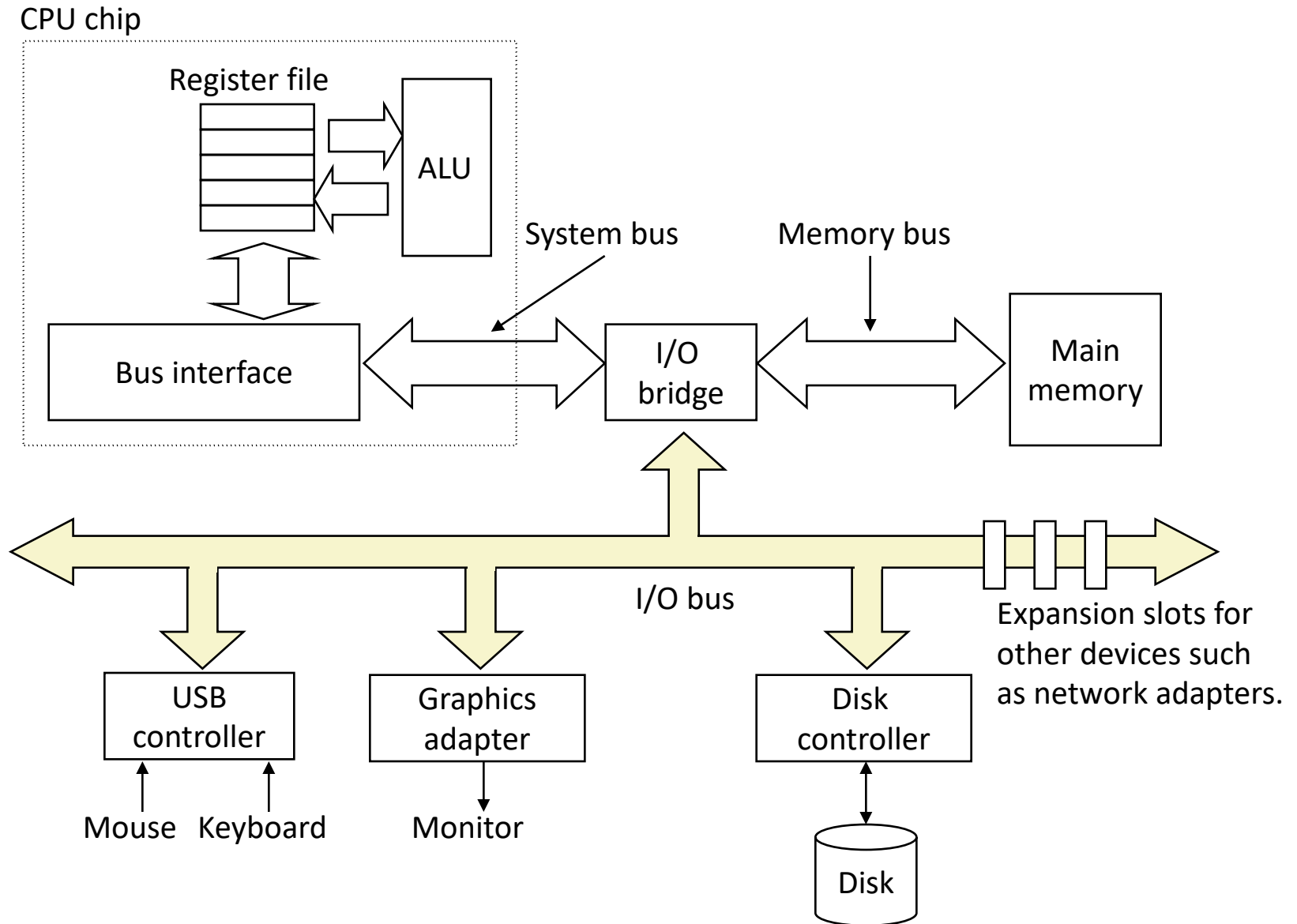
Agenda

- Locality of reference
- Caching in the memory hierarchy
- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

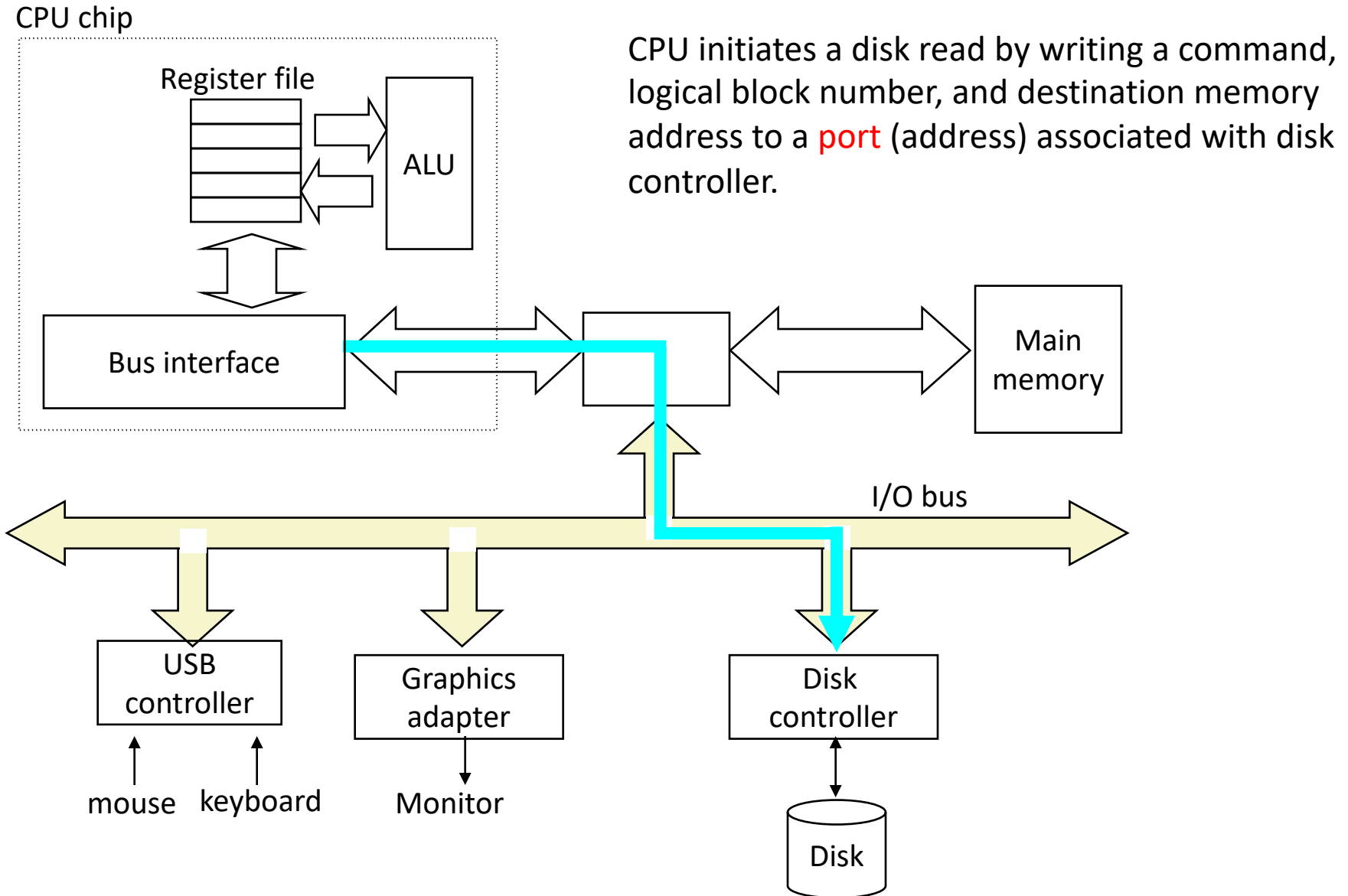
Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
 - The set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
 - Maintained by hardware/firmware device called disk controller.
 - Converts requests for logical blocks into (surface, track, sector) triples.
- Allows controller to set aside spare cylinders for each zone.
 - Accounts for the difference in “formatted capacity” and “maximum capacity”.

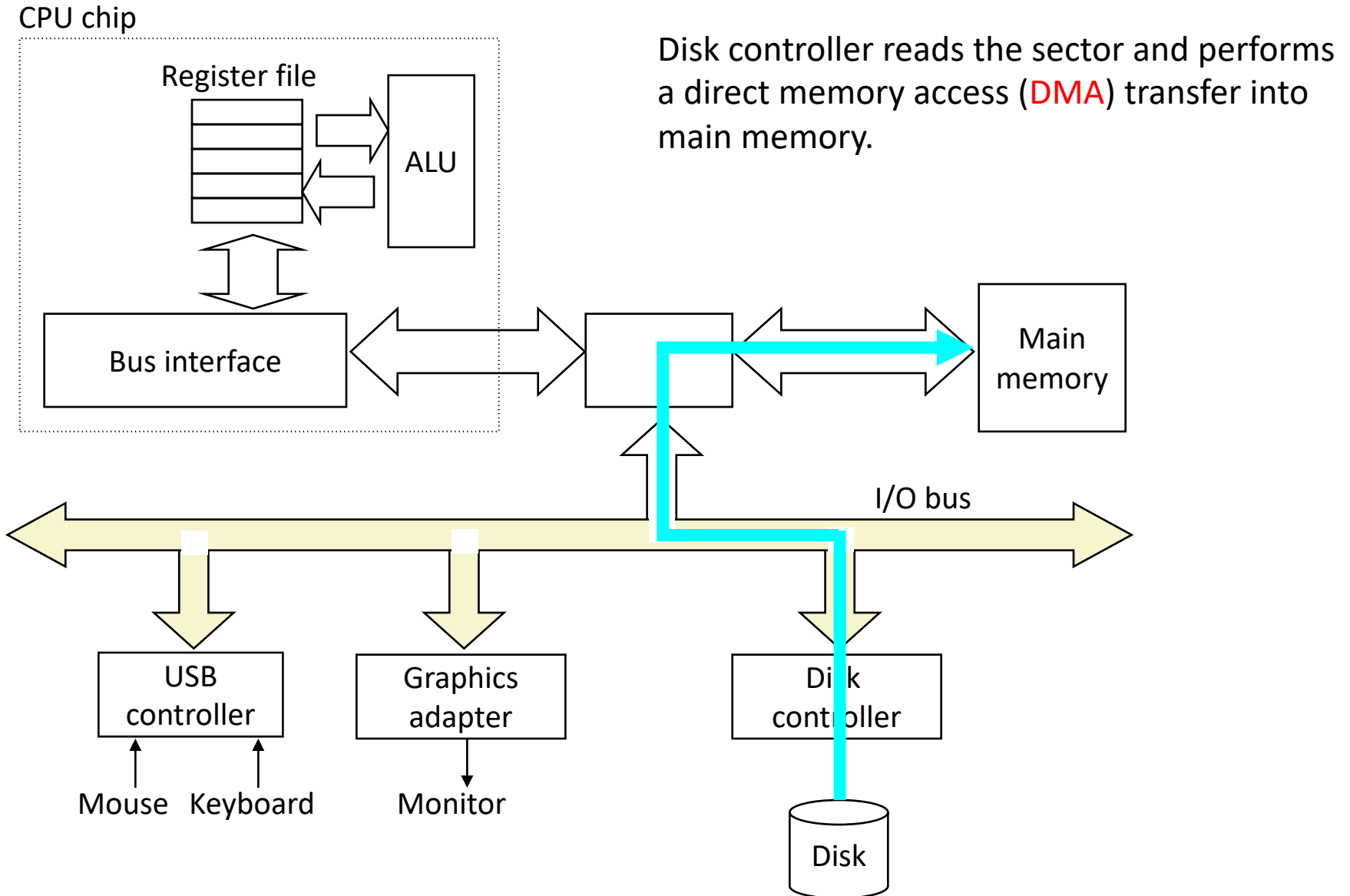
I/O Bus



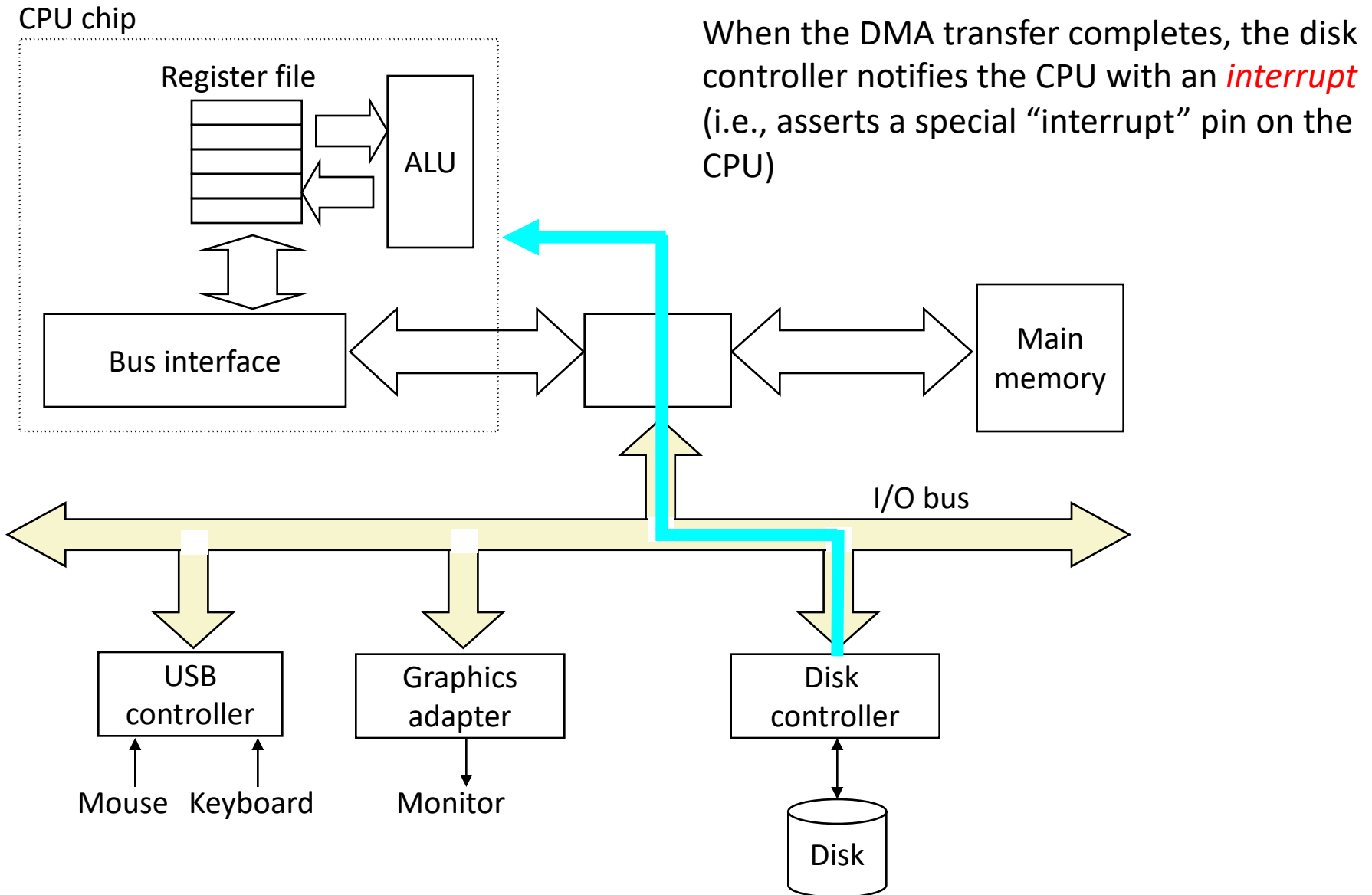
Reading a Disk Sector (1)



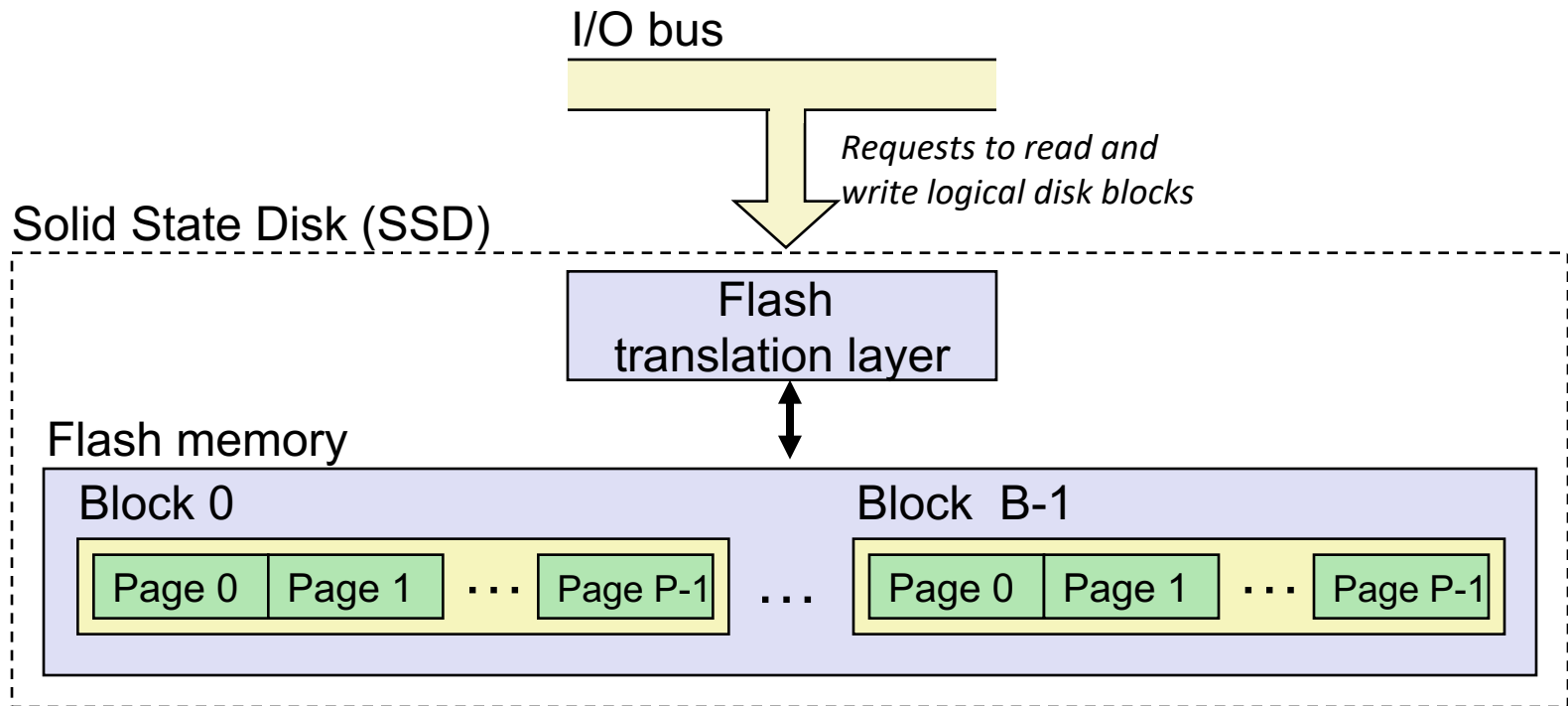
Reading a Disk Sector (2)



Reading a Disk Sector (3)



Solid State Disks (SSDs)



- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after about 100,000 repeated writes.

SSD Performance Characteristics

Sequential read tput	550 MB/s	Sequential write tput	470 MB/s
Random read tput	365 MB/s	Random write tput	303 MB/s
Avg seq read time	50 us	Avg seq write time	60 us

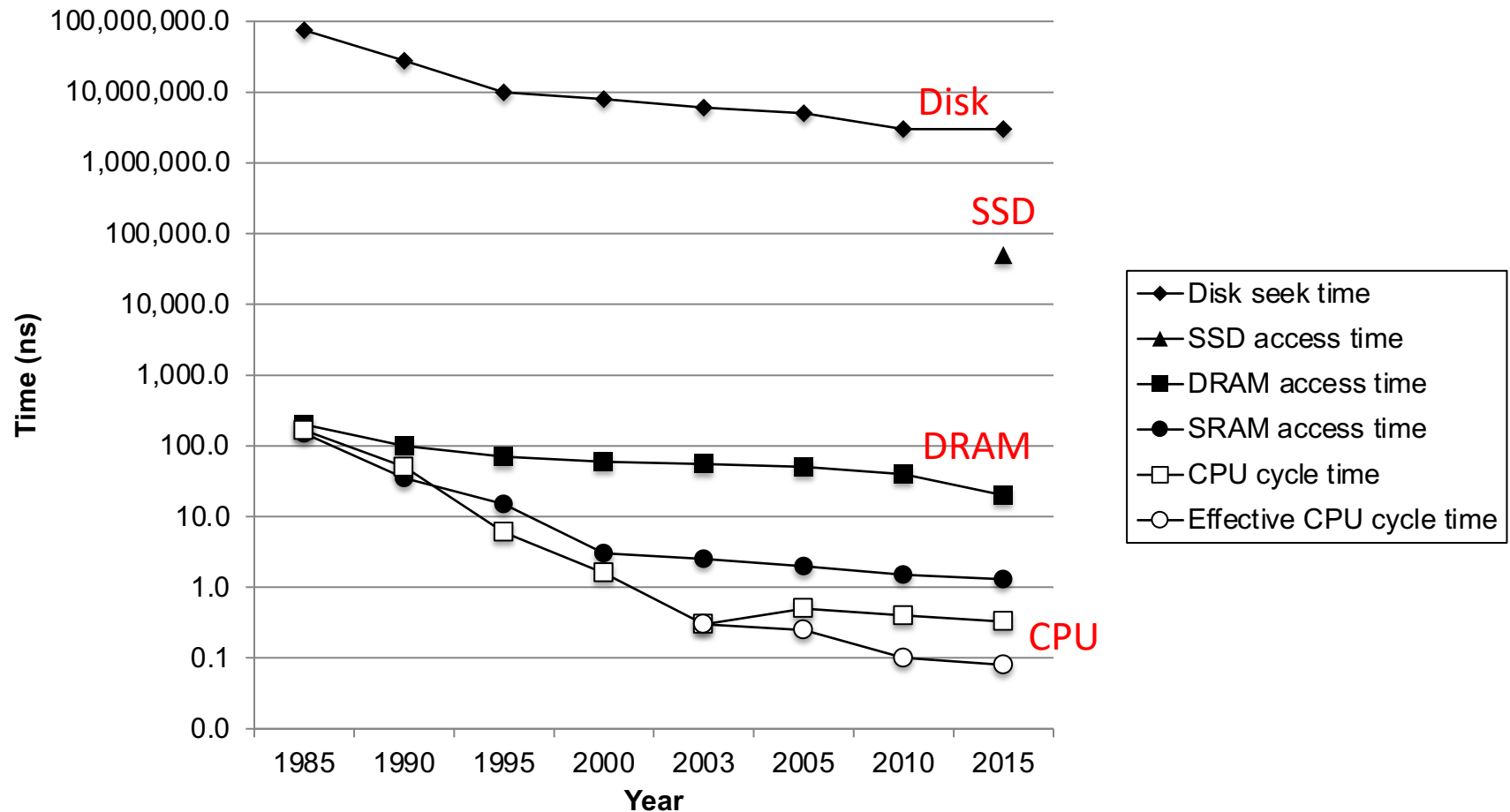
- Sequential access faster than random access
 - Common theme in the memory hierarchy
- Random writes are somewhat slower
 - Erasing a block takes a long time (~1 ms)
 - Modifying a block page requires all other pages to be copied to new block
 - In earlier SSDs, the read/write gap was much larger.

SSD Tradeoffs vs Rotating Disks

- Advantages
 - No moving parts → faster, less power, more rugged
- Disadvantages
 - Have the potential to wear out
 - Mitigated by “wear leveling logic” in flash translation layer
 - E.g. Intel SSD 730 guarantees 128 petabyte (128×10^{15} bytes) of writes before they wear out
 - In 2015, about 30 times more expensive per byte
- Applications
 - MP3 players, smart phones, laptops
 - Beginning to appear in desktops and servers

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Locality to the Rescue!

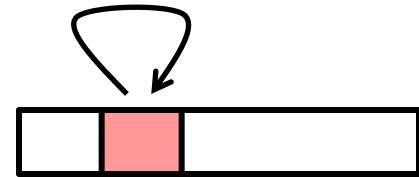
The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as **locality**

- **Locality of reference**

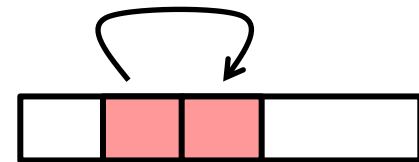
Locality

- **Principle of Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

- **Temporal locality:**
 - Recently referenced items are likely to be referenced again in the near future



- **Spatial locality:**
 - Items with nearby addresses tend to be referenced close together in time



Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- Data references
 - Reference array elements in succession (stride-1 reference pattern).
Spatial locality
 - Reference variable `sum` each iteration.
Temporal locality
- Instruction references
 - Reference instructions in sequence.
Spatial locality
 - Cycle through loop repeatedly.
Temporal locality

Qualitative Estimates of Locality

- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- **Question:** Does this function have good locality with respect to array `a`?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```


Locality Example

- **Question:** Does this function have good locality with respect to array *a*?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Locality Example

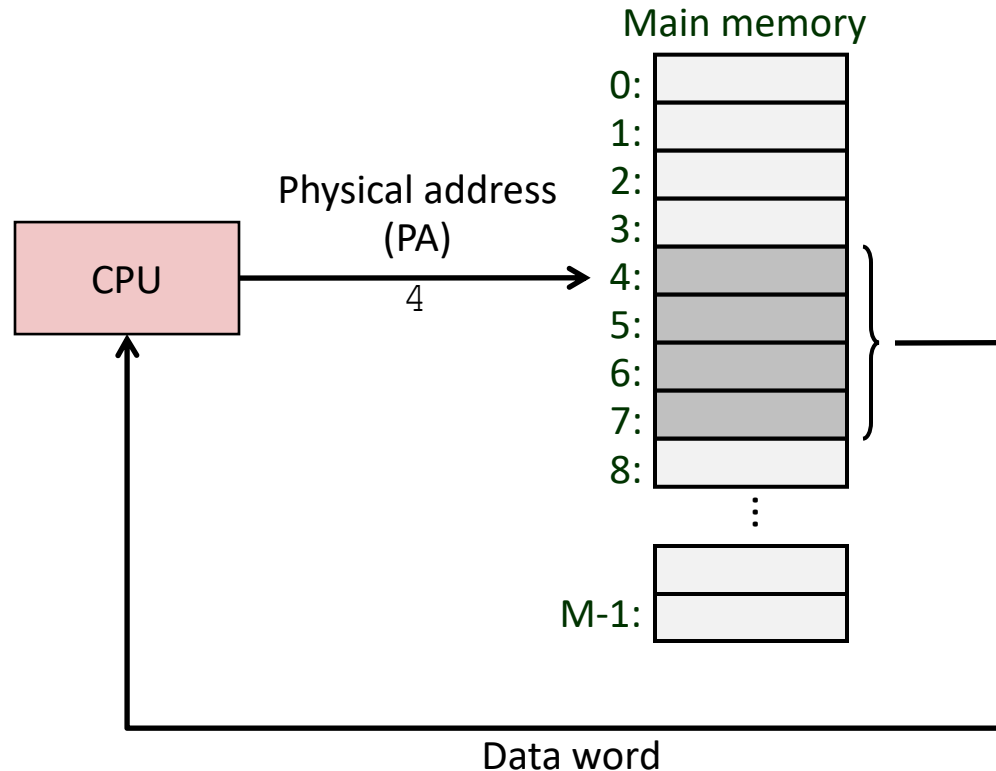
- **Question:** Can you permute the loops so that the function scans the 3-d array `a` with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];

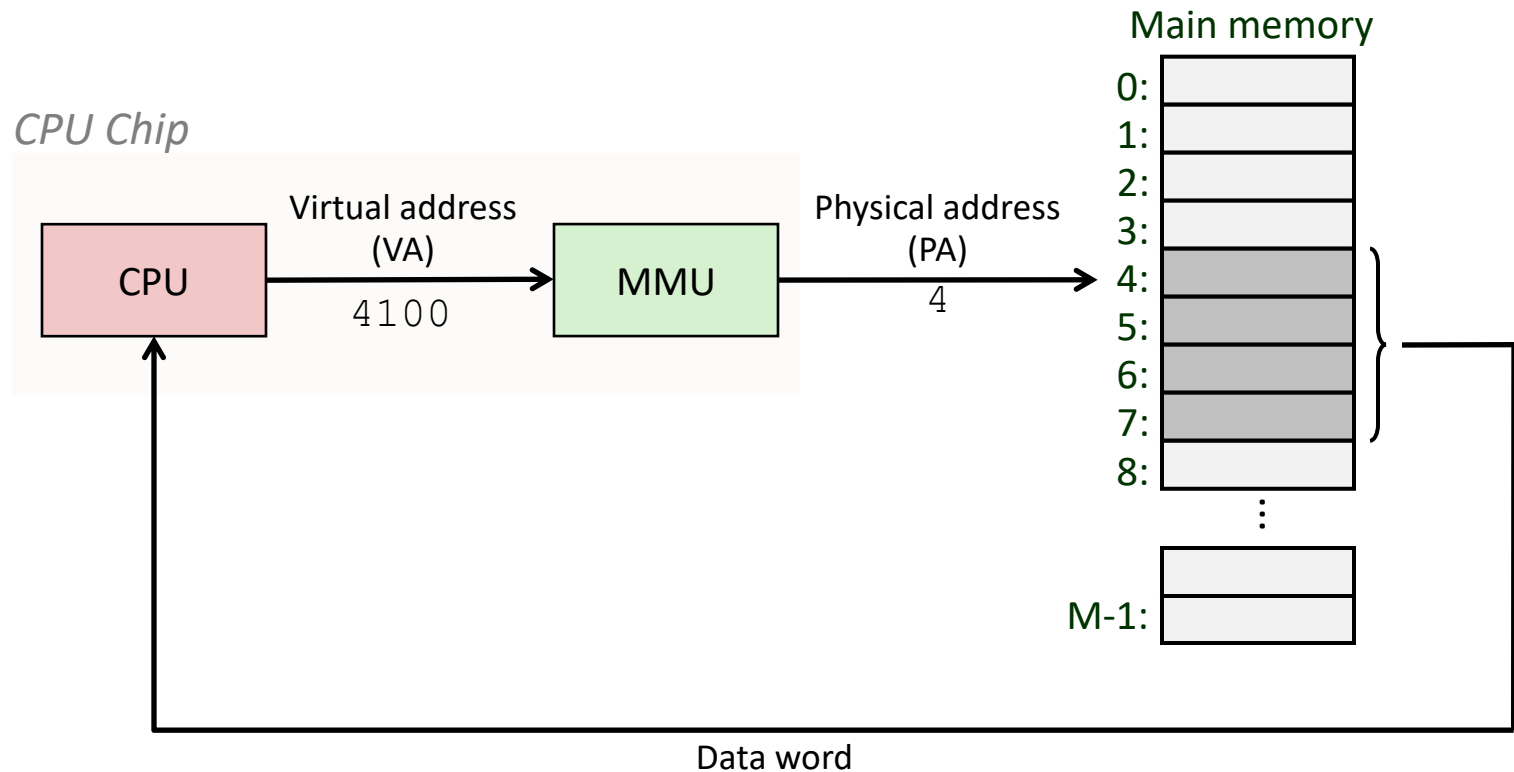
    return sum;
}
```

A System Using Physical Addressing



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

A System Using Virtual Addressing



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

Address Spaces

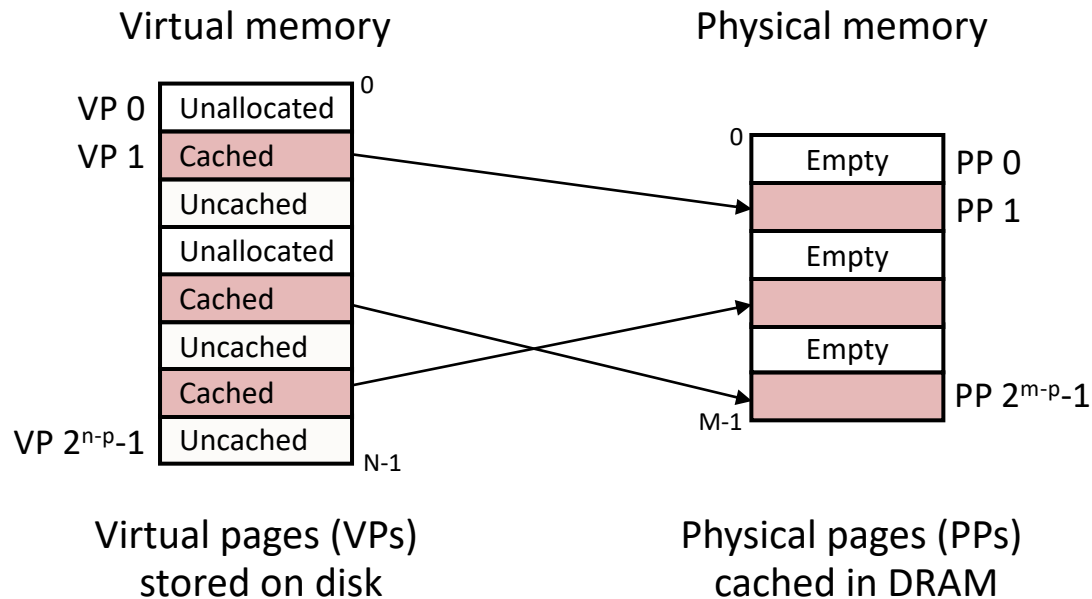
- **Linear address space:** Ordered set of contiguous non-negative integer addresses:
 $\{0, 1, 2, 3 \dots \}$
- **Virtual address space:** Set of $N = 2^n$ virtual addresses
 $\{0, 1, 2, 3, \dots, N-1\}$
- **Physical address space:** Set of $M = 2^m$ physical addresses
 $\{0, 1, 2, 3, \dots, M-1\}$

Why Virtual Memory (VM)?

- Uses main memory efficiently
 - Use DRAM as a cache for 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 and code

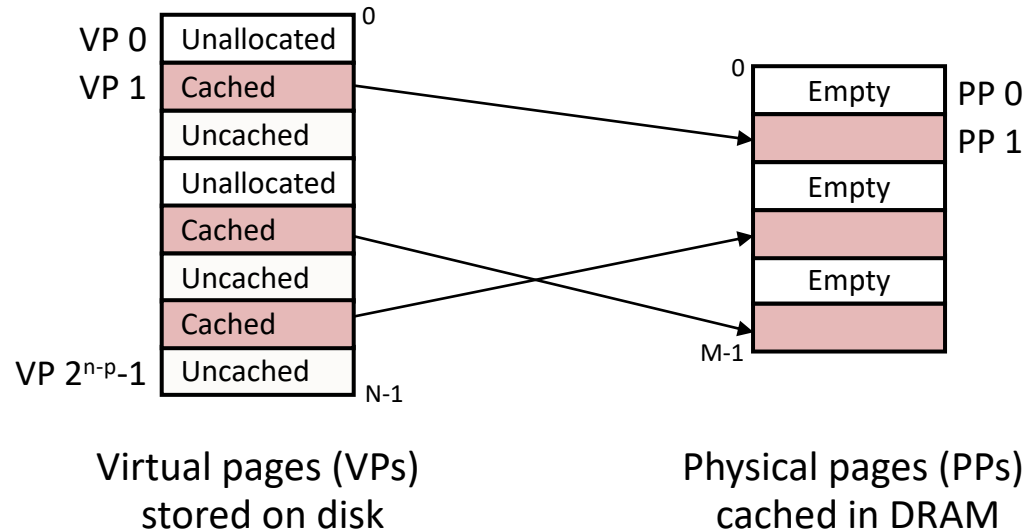
VM as a Tool for Caching

- Conceptually, *virtual memory* is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in *physical memory* (*DRAM cache*)
 - These cache blocks are called *pages* (size is $P = 2^p$ bytes)



Virtual Pages

- At any point in time, the set of virtual pages is partitioned into three disjoint subsets;
- Unallocated:** pages that have not been allocated by VM system. Unallocated blocks do not have any data associated with them, thus do not occupy any space on the disk
- Cached:** allocated pages that are currently cached in physical memory
- Uncached:** allocated pages that are not cached in physical memory

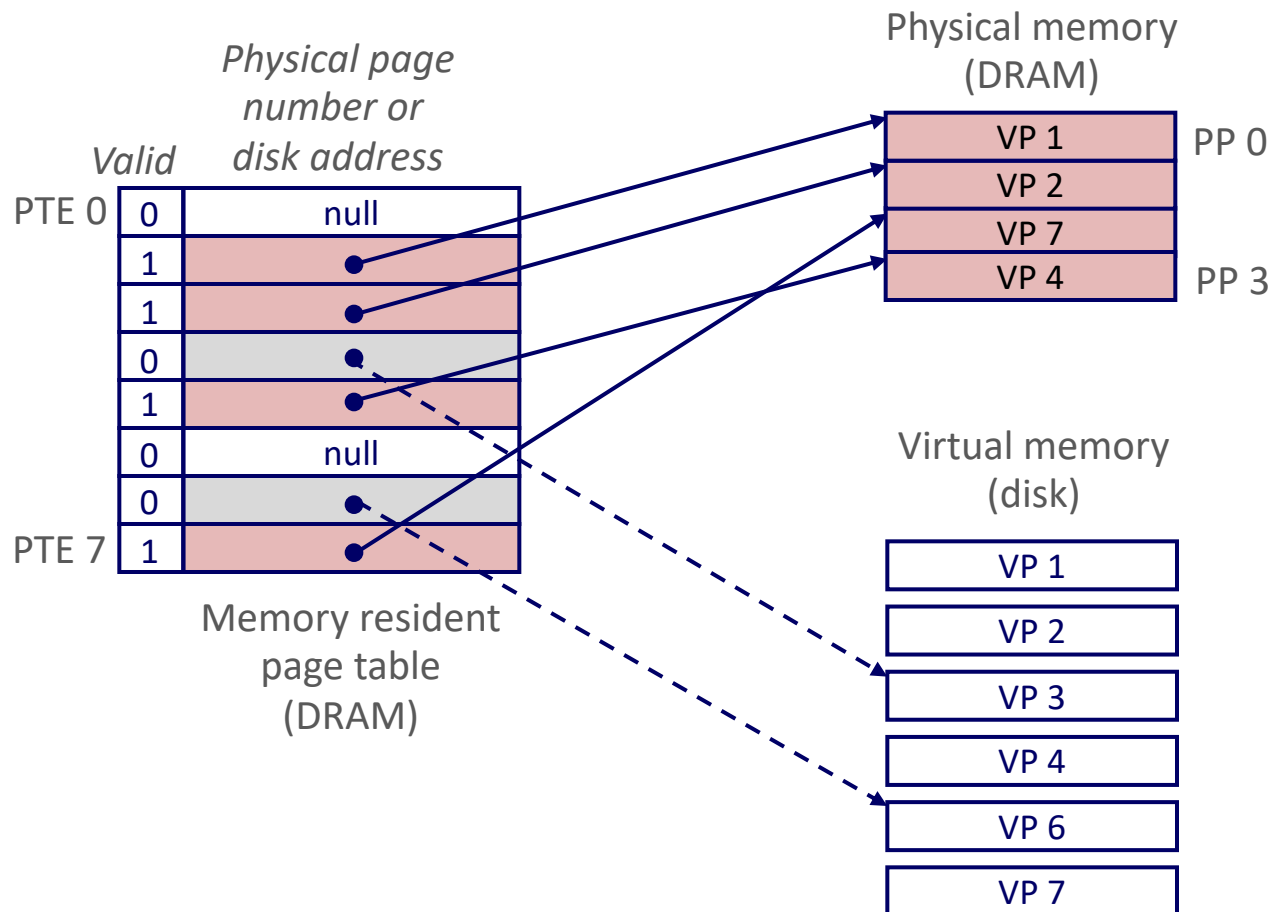


DRAM Cache Organization

- DRAM cache organization driven by the enormous miss penalty
 - DRAM is about **10x** slower than SRAM
 - Disk is about **10,000x** slower than DRAM
- Consequences
 - Large page (block) size: typically 4 KB, sometimes 4 MB
 - Fully associative
 - Any VP can be placed in any PP
 - Requires a “large” mapping function – different from cache memories
 - Highly sophisticated, expensive replacement algorithms
 - Too complicated and open-ended to be implemented in hardware
 - Write-back rather than write-through

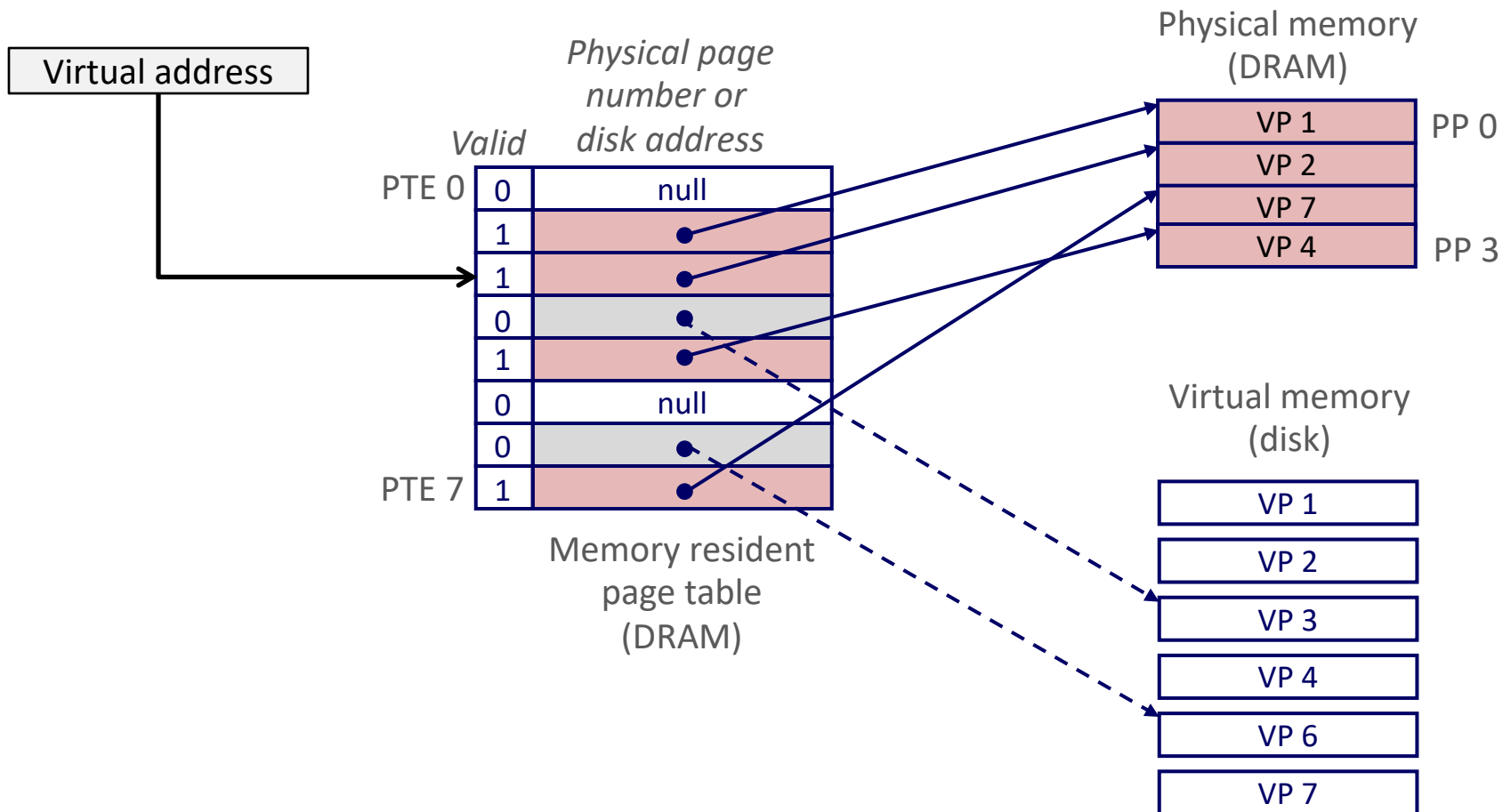
Enabling Data Structure: Page Table

- A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.
 - Per-process kernel data structure in DRAM



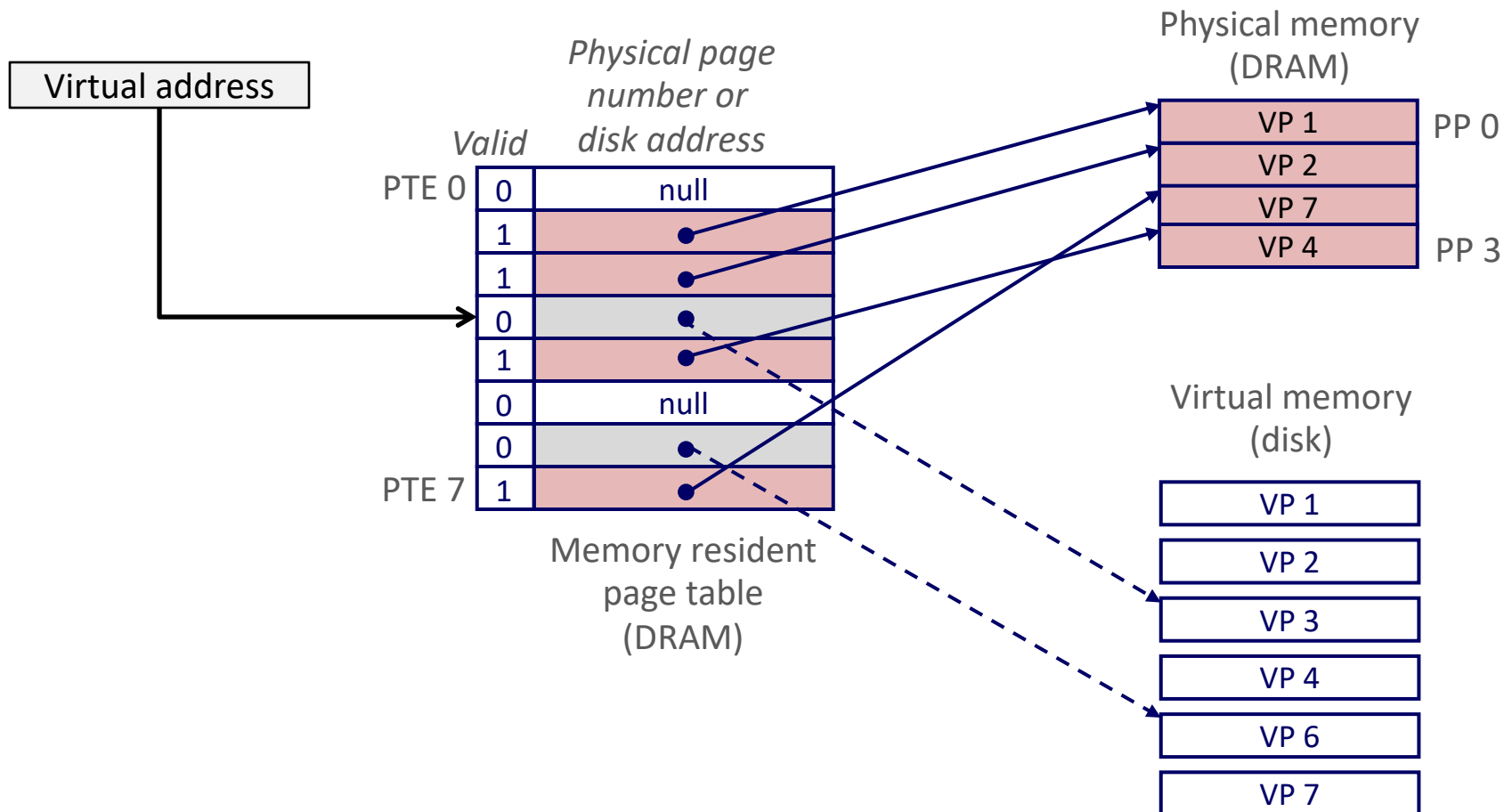
Page Hit

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



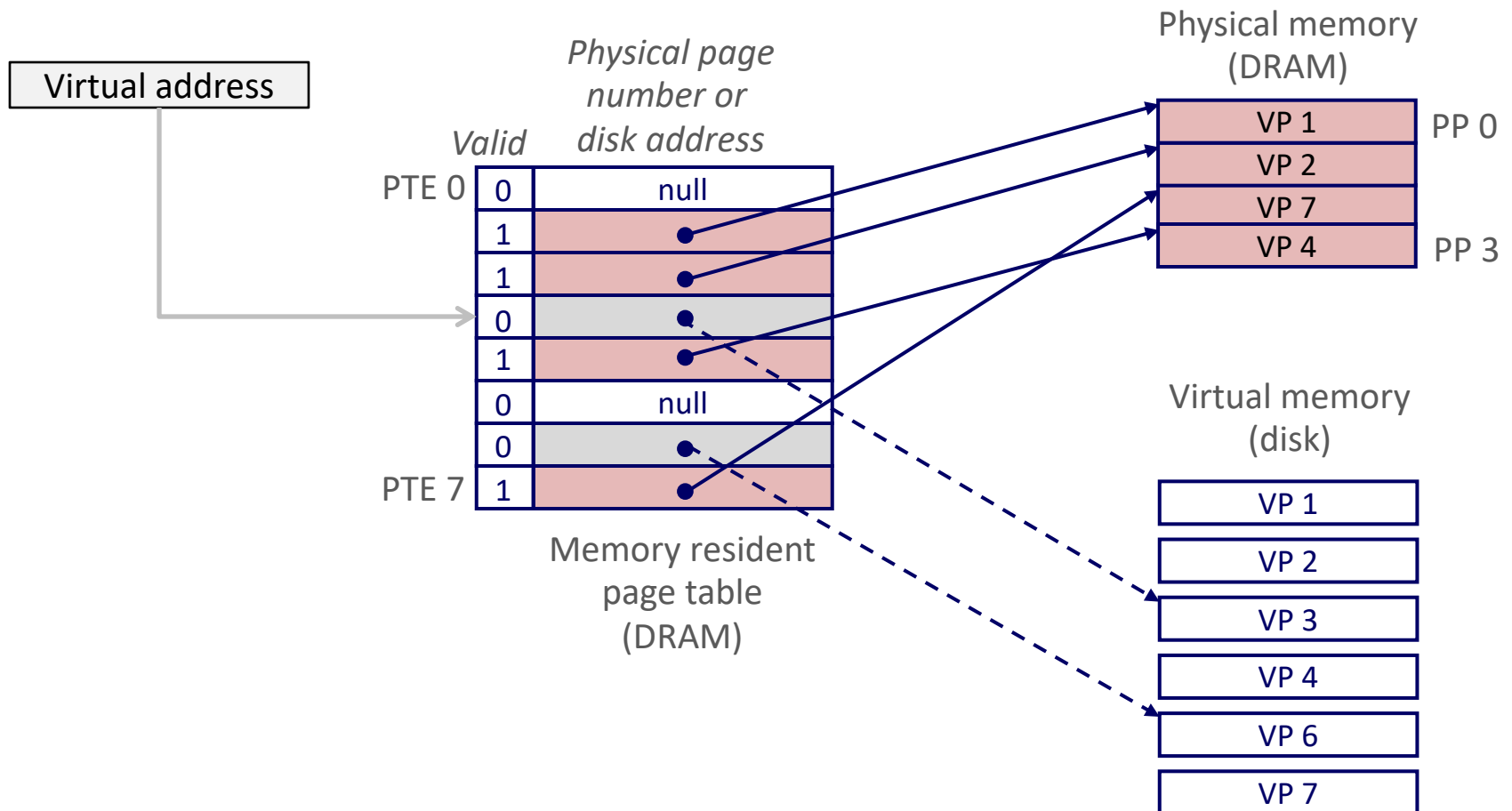
Page Fault

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



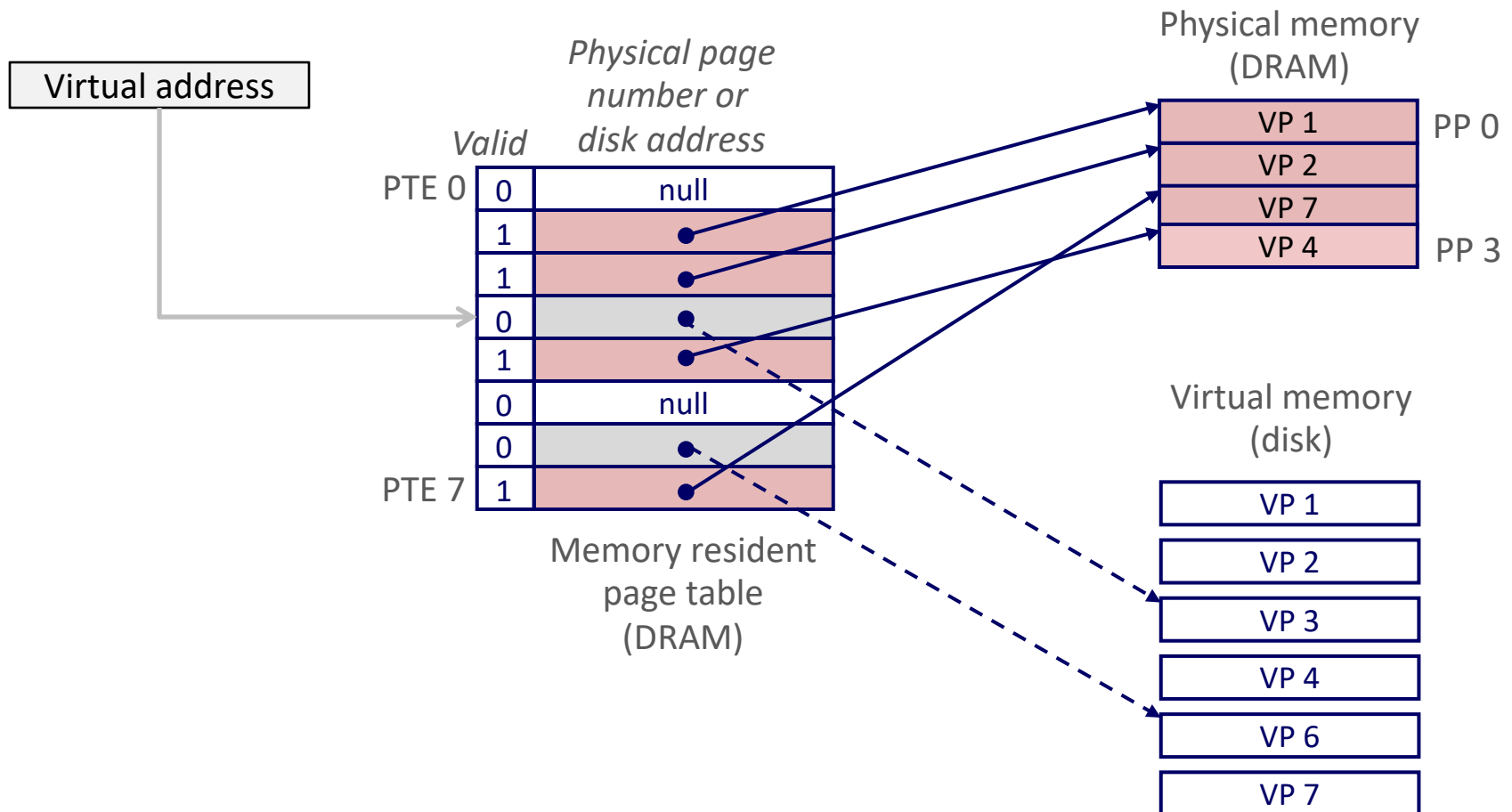
Handling Page Fault

- Page miss causes page fault (an exception)



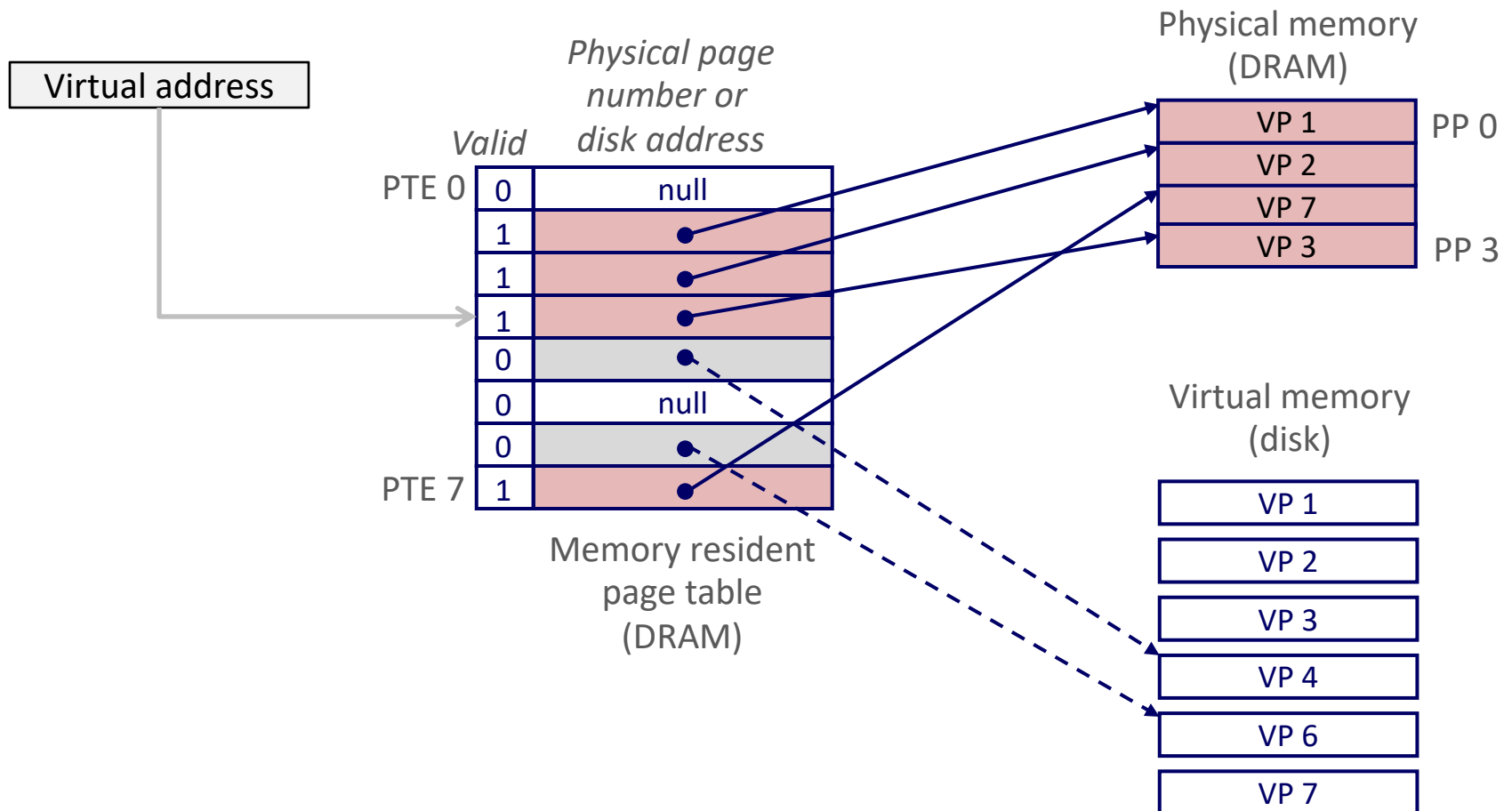
Handling Page Fault

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



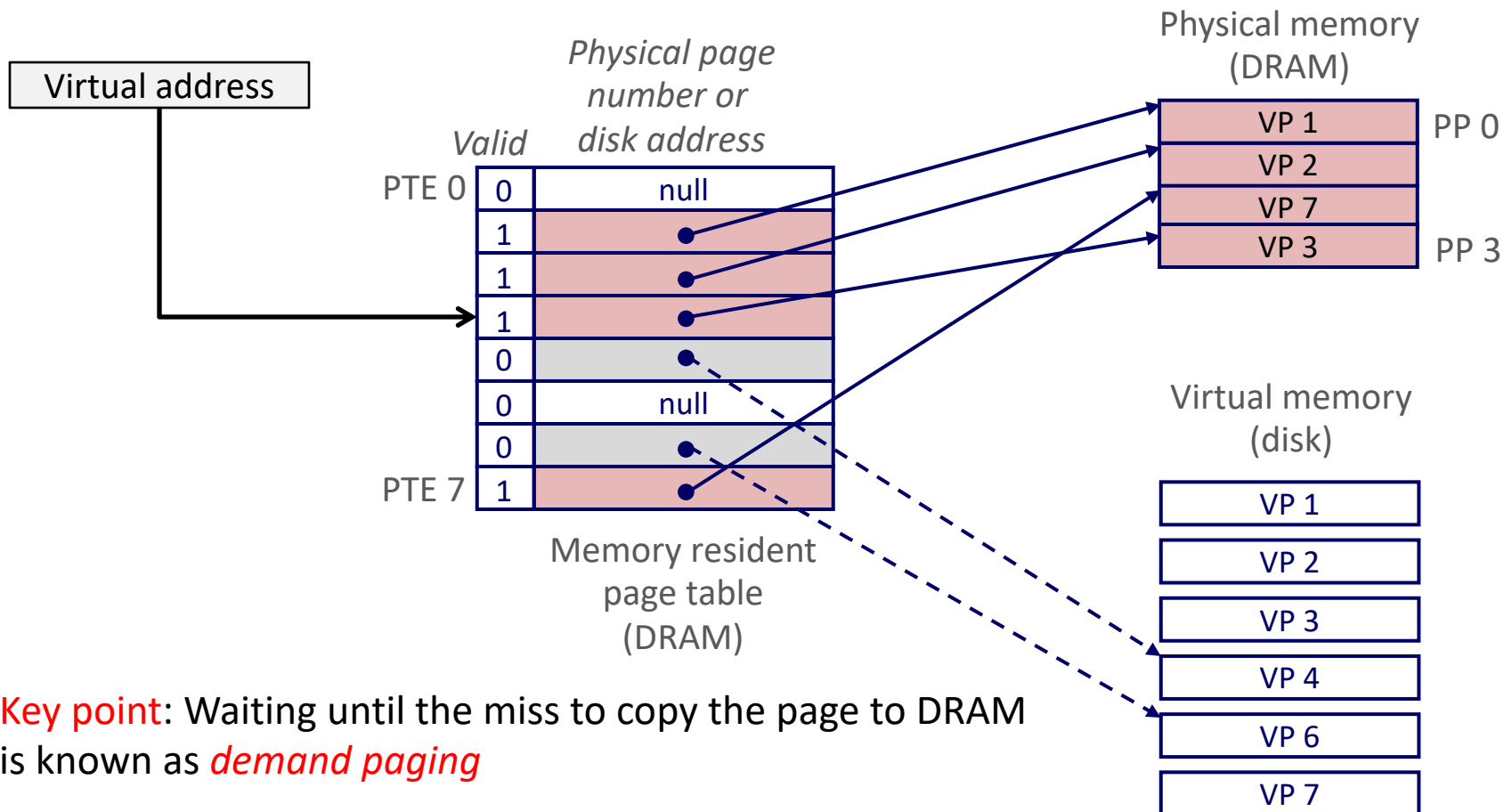
Handling Page Fault

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



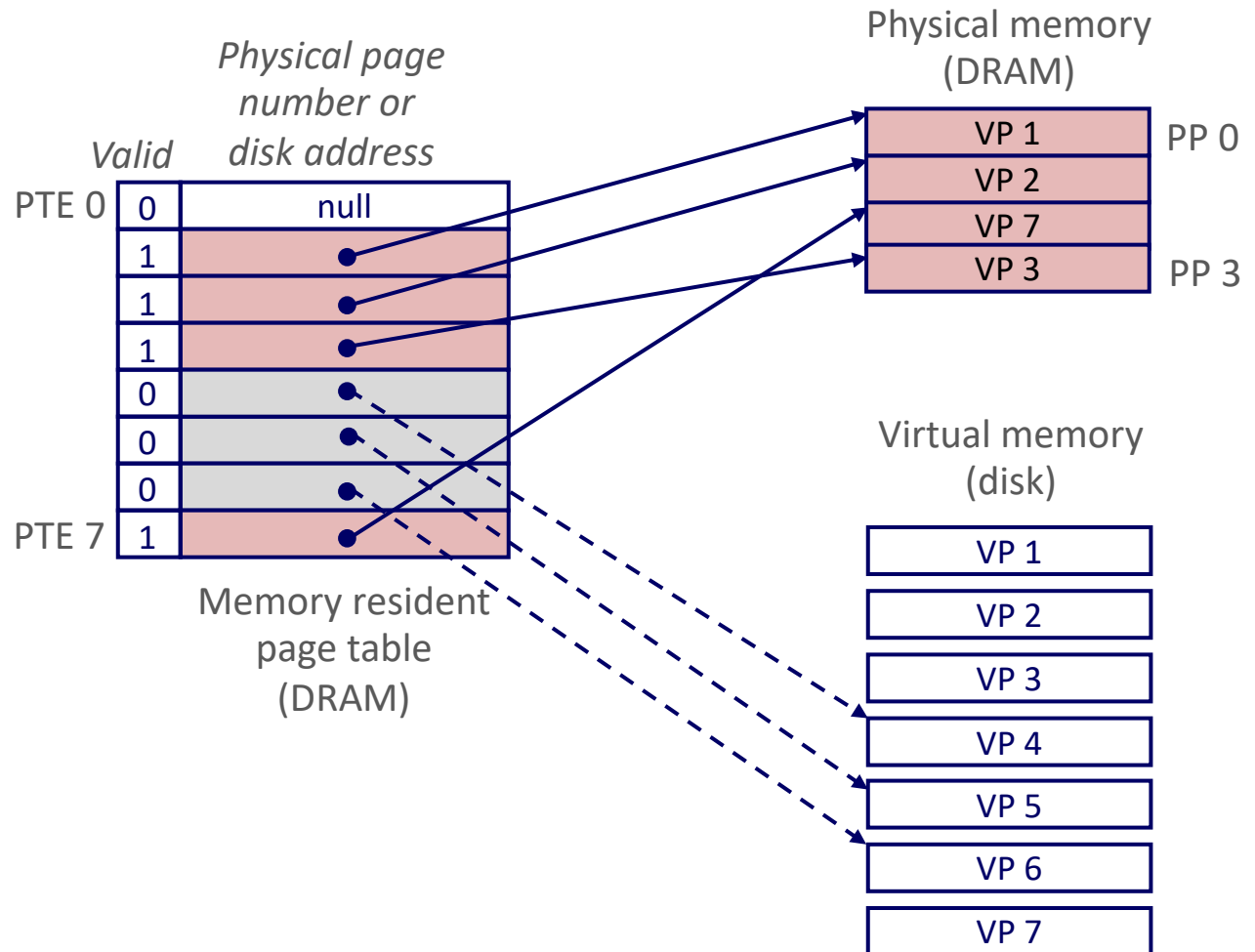
Handling Page Fault

- 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!



Allocating Pages

- Allocating a new page (VP 5) of virtual memory.

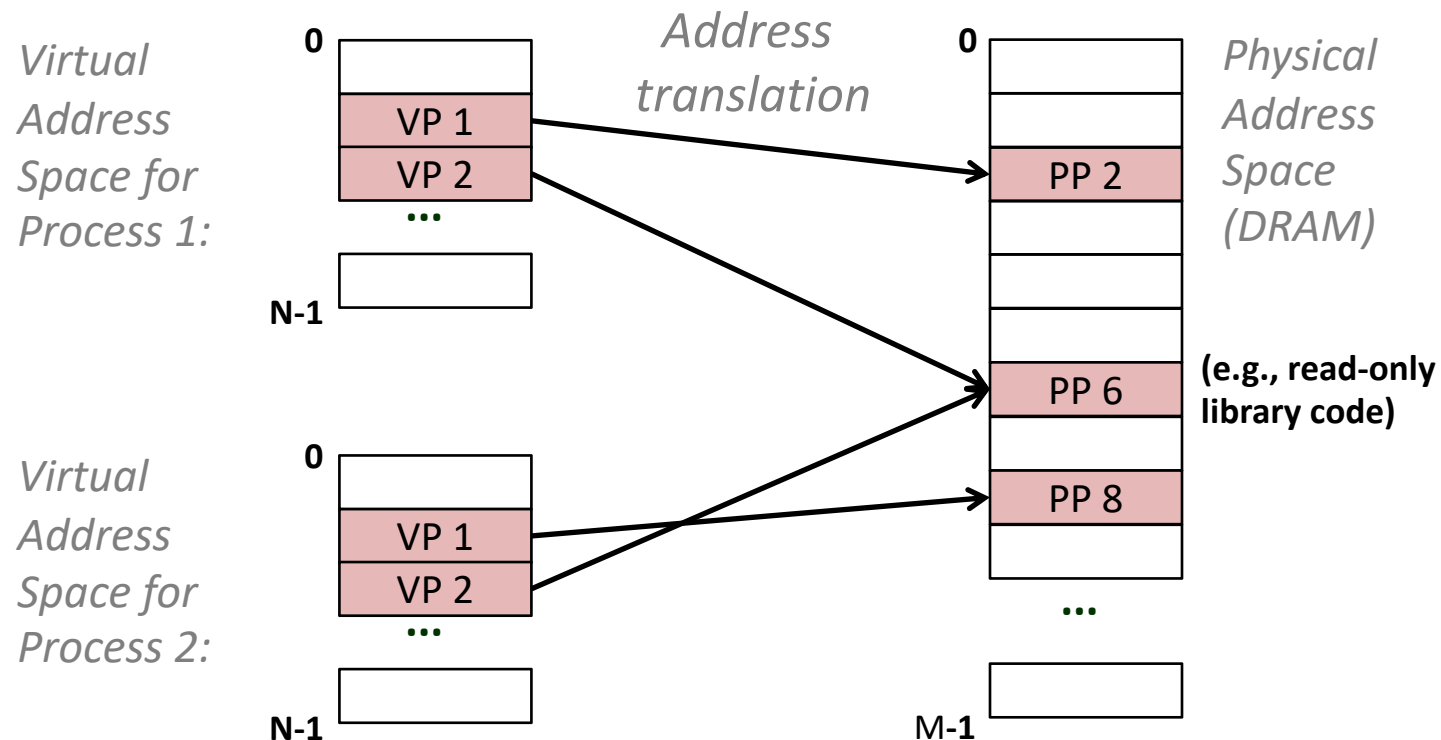


Locality to the Rescue Again!

- Virtual memory seems terribly inefficient, but it 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

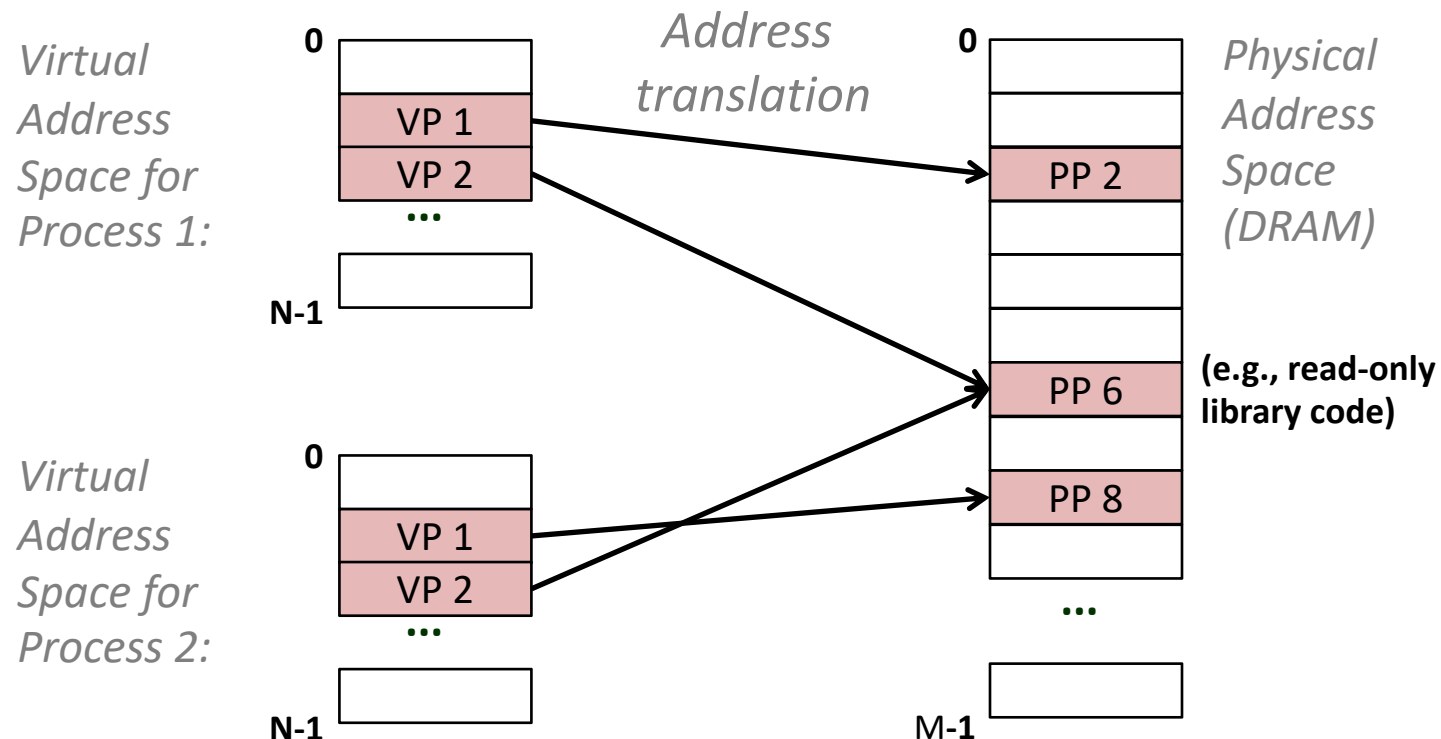
VM as a Tool for Memory Management

- Key idea: each process has its own virtual address space
 - It can view memory as a simple linear array
 - Mapping function scatters addresses through physical memory
 - Well-chosen mappings can improve locality



VM as a Tool for Memory Management

- **Simplifying memory allocation**
 - Each virtual page can be mapped to any physical page
 - A virtual page can be stored in different physical pages at different times
- **Sharing code and data among processes**
 - Map virtual pages to the same physical page (here: PP 6)



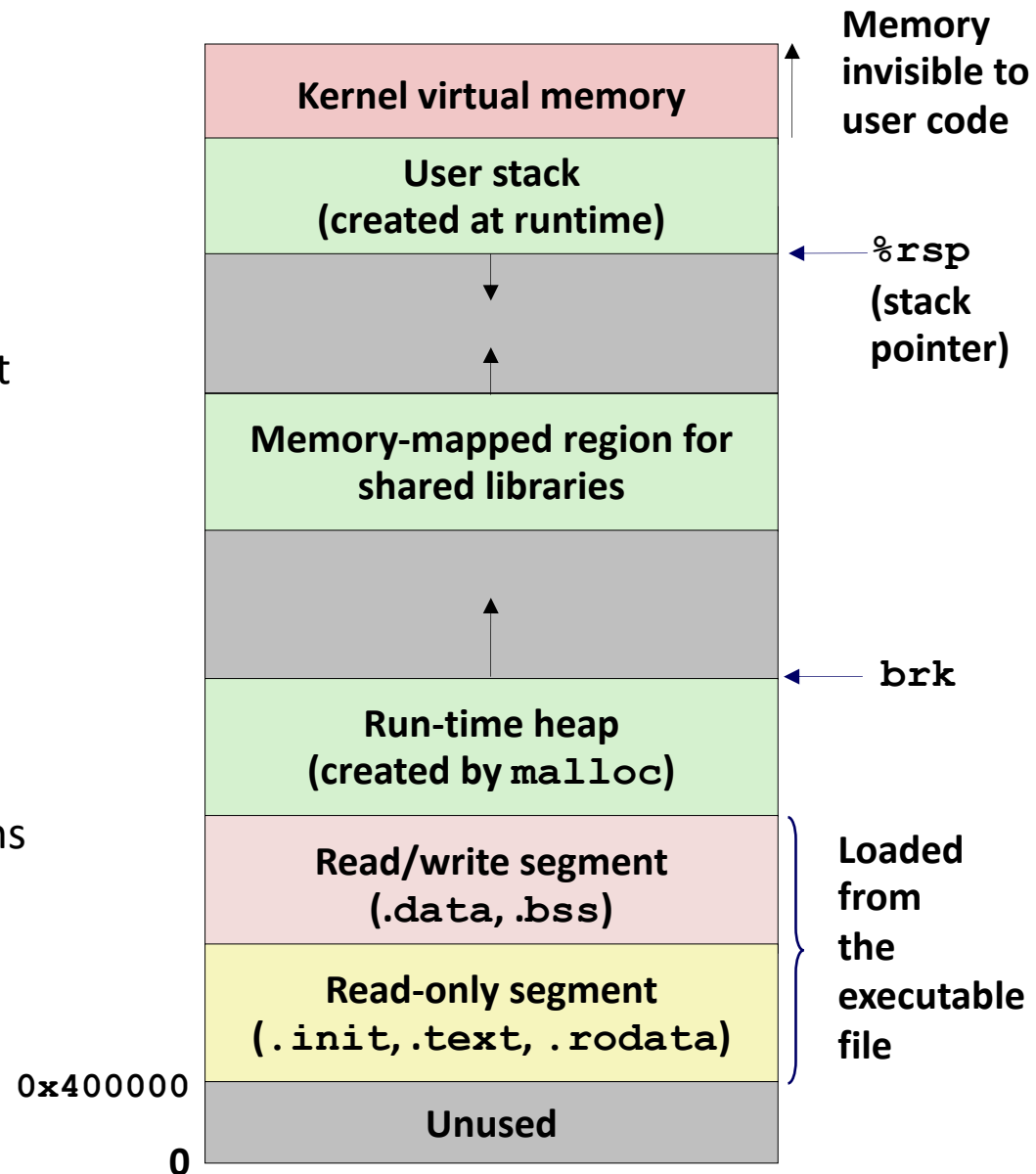
Simplifying Linking and Loading

- Linking

- Each program has similar virtual address space
- Code, data, and heap always start at the same addresses.

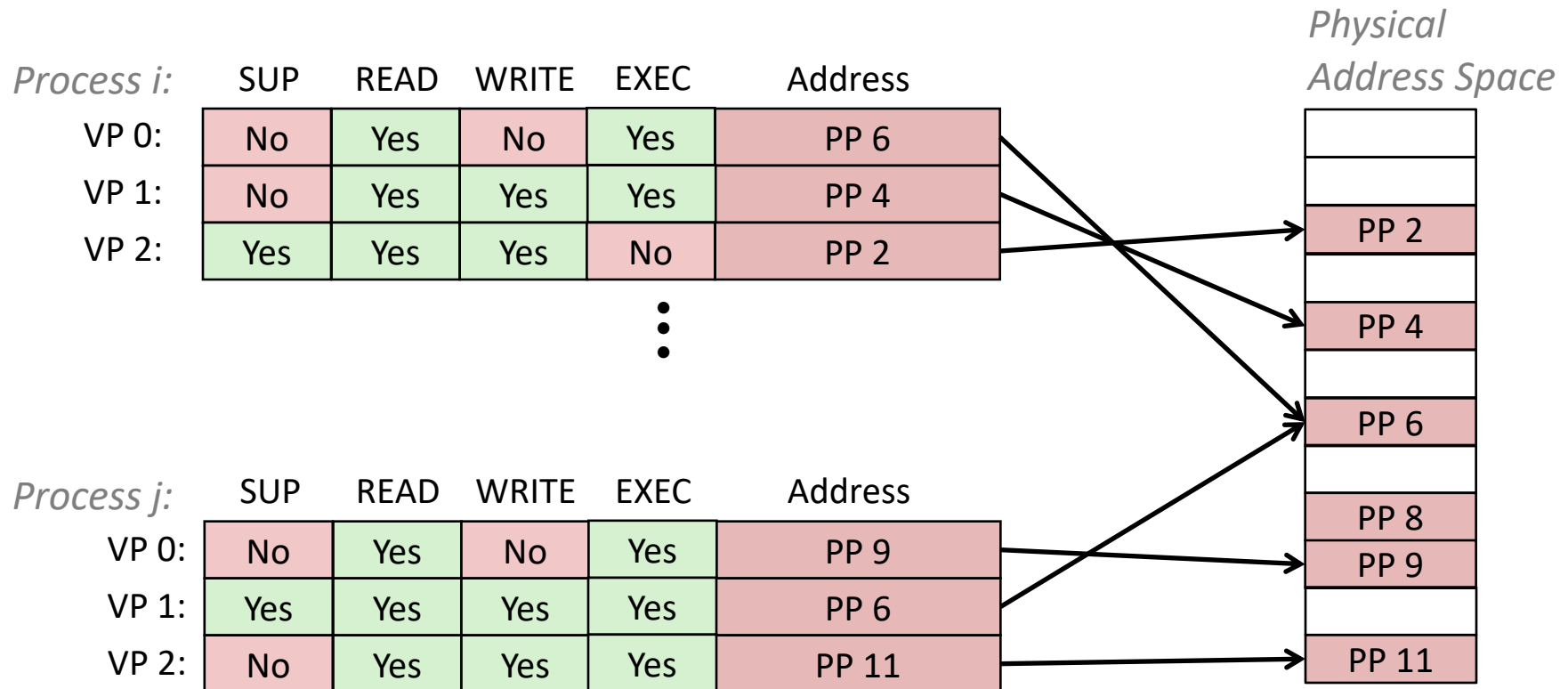
- Loading

- **execve** allocates virtual pages for **.text** and **.data** sections & creates PTEs marked as invalid
- The **.text** and **.data** sections are copied, page by page, on demand by the virtual memory system



VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- MMU checks these bits on each access



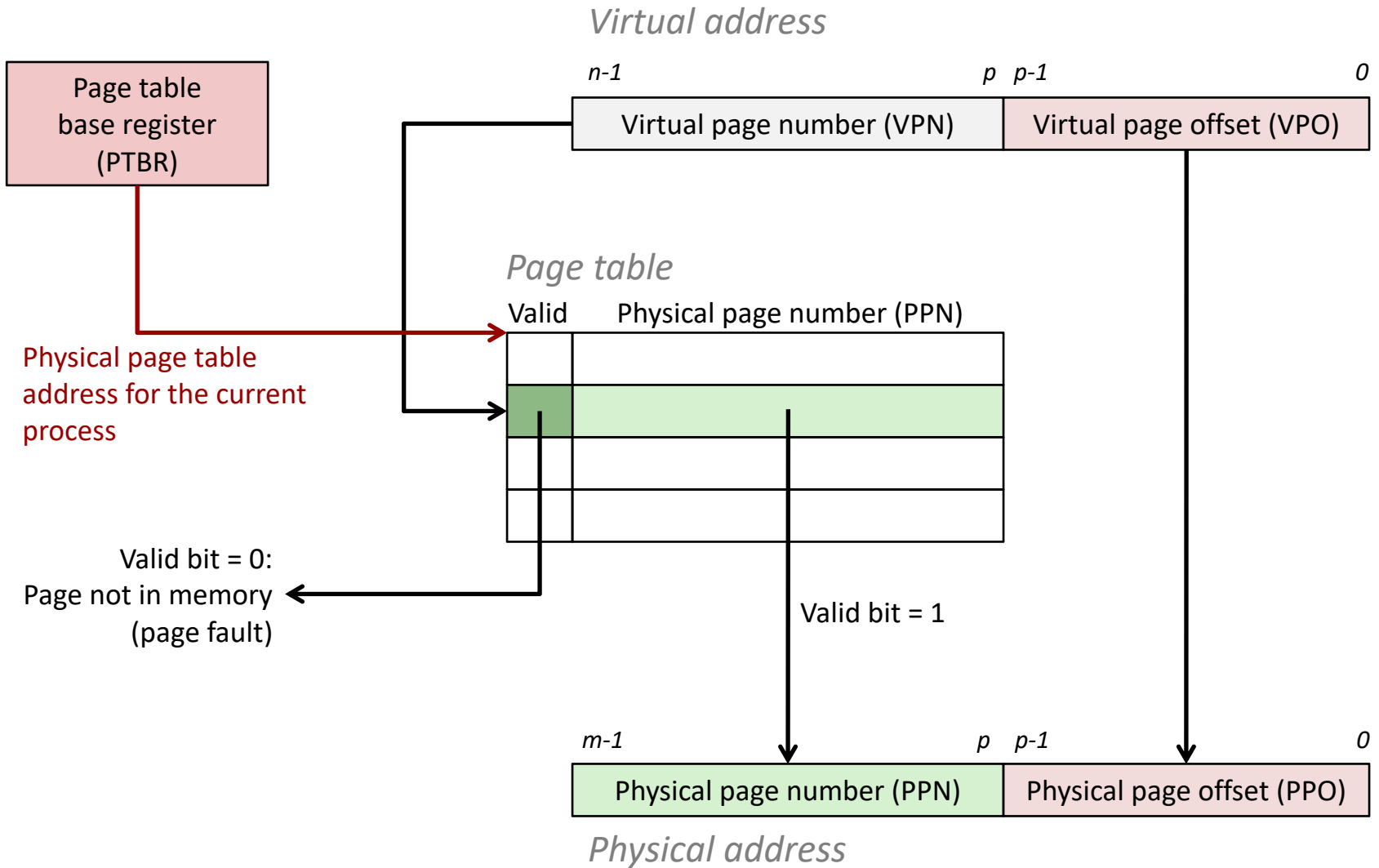
VM Address Translation

- Virtual Address Space
 - $V = \{0, 1, \dots, N-1\}$
- Physical Address Space
 - $P = \{0, 1, \dots, M-1\}$
- Address Translation
 - **$MAP: V \rightarrow P \cup \{\emptyset\}$**
 - For virtual address **a** :
 - **$MAP(a) = a'$** if data at virtual address **a** is at physical address **a'** in **P**
 - **$MAP(a) = \emptyset$** if data at virtual address **a** is not in physical memory
 - Either invalid or stored on disk

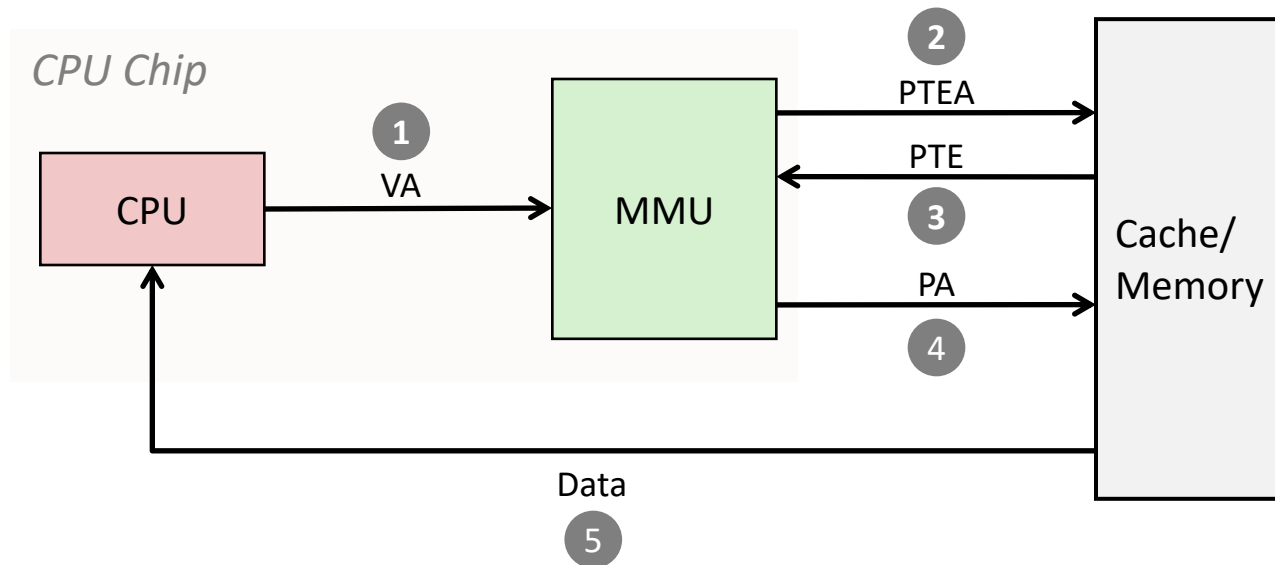
Summary of Address Translation Symbols

- Basic Parameters
 - **N** = 2^n : Number of addresses in virtual address space
 - **M** = 2^m : Number of addresses in physical address space
 - **P** = 2^p : Page size (bytes)
- Components of the virtual address (VA)
 - **TLBI**: TLB index
 - **TLBT**: TLB tag
 - **VPO**: Virtual page offset
 - **VPN**: Virtual page number
- Components of the physical address (PA)
 - **PPO**: Physical page offset (same as VPO)
 - **PPN**: Physical page number

Address Translation With a Page Table



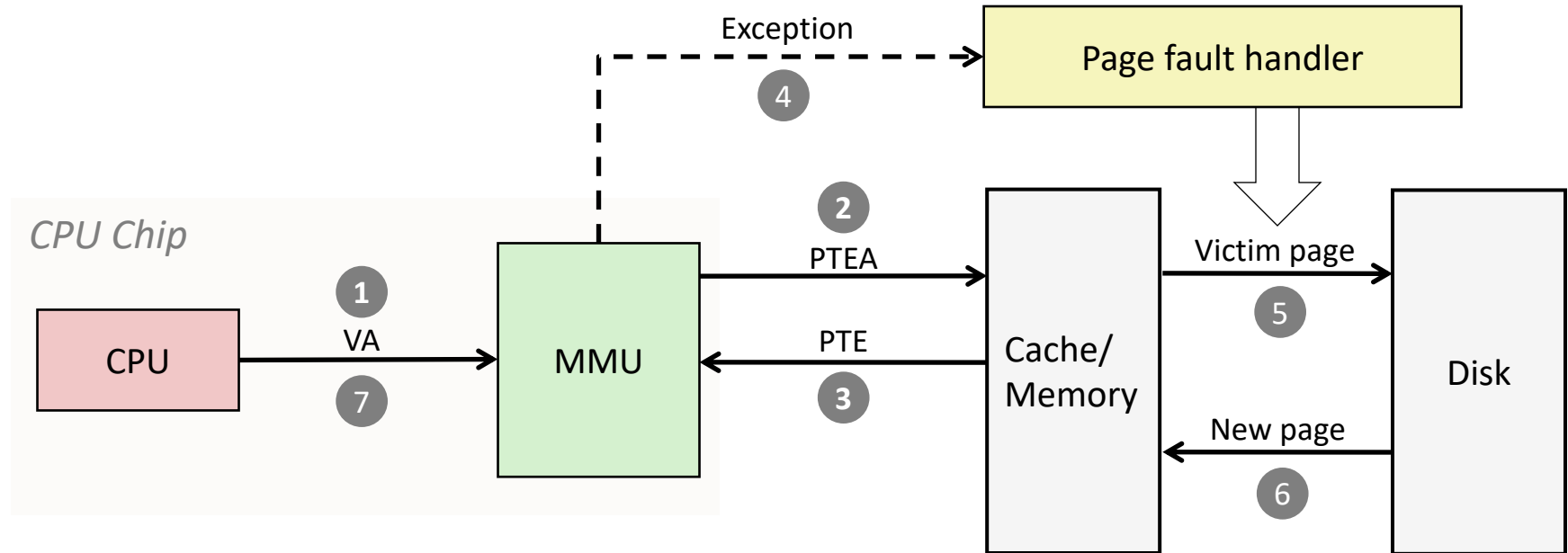
Address Translation: Page Hit



- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) MMU sends physical address to cache/memory
- 5) Cache/memory sends data word to processor

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

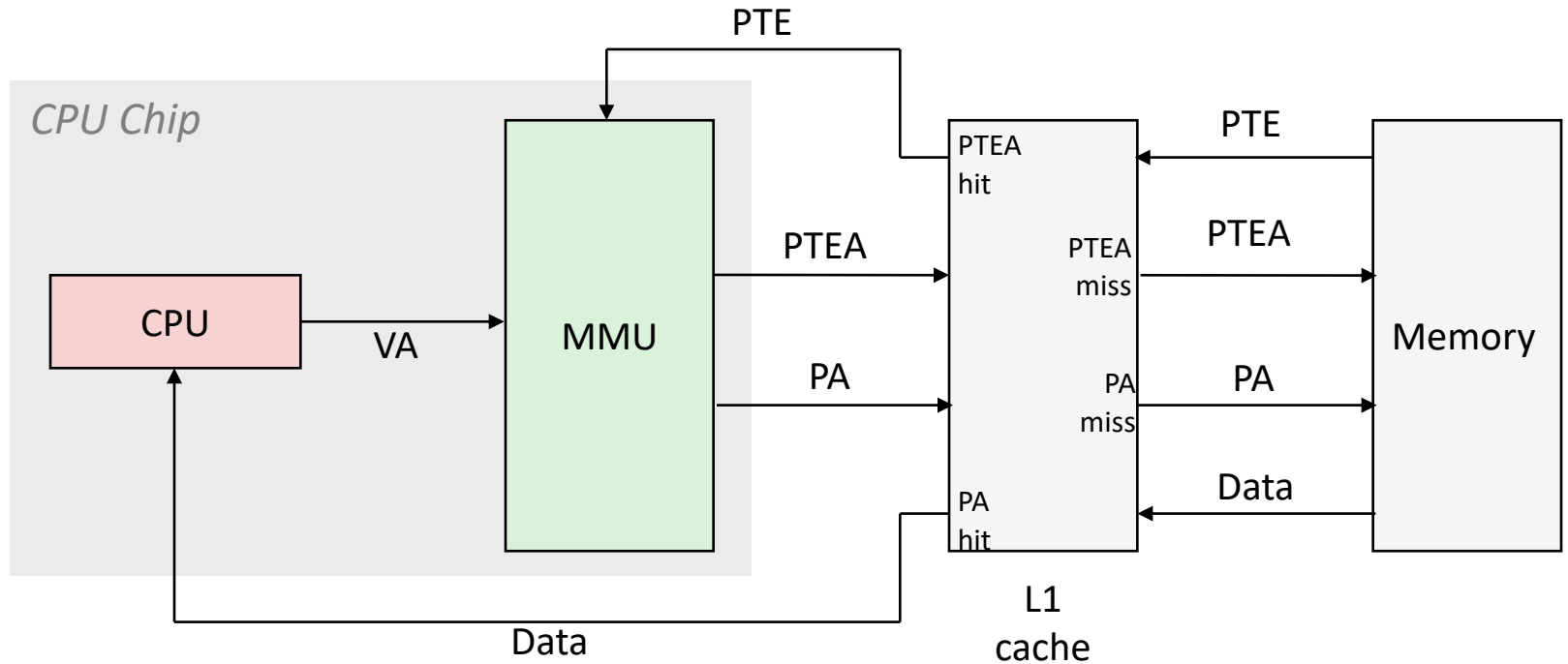
Address Translation: Page Fault



- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) Valid bit is zero, so MMU triggers page fault exception
- 5) Handler identifies victim (and, if dirty, pages it out to disk)
- 6) Handler pages in new page and updates PTE in memory
- 7) Handler returns to original process, restarting faulting instruction

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

Integrating VM and Cache



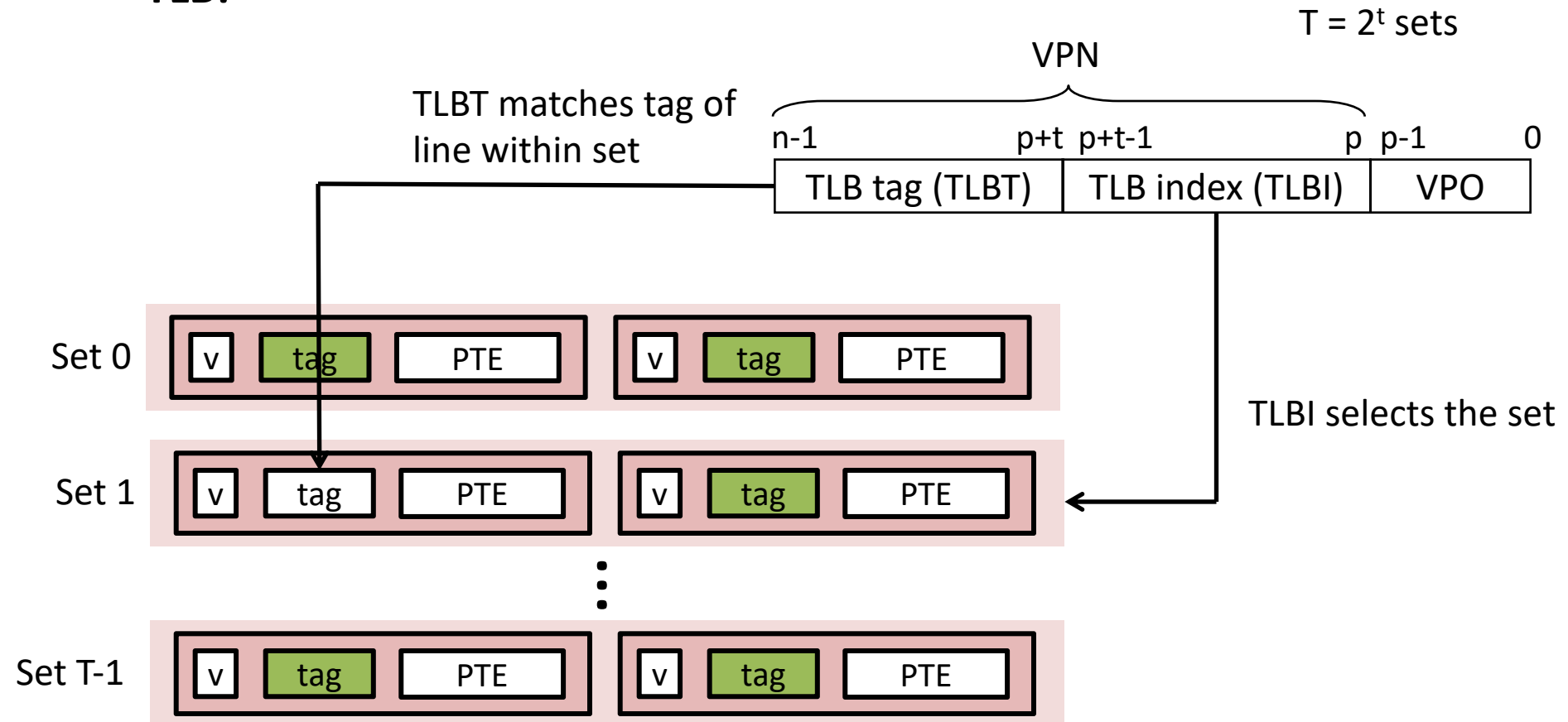
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Speeding up Translation with a TLB

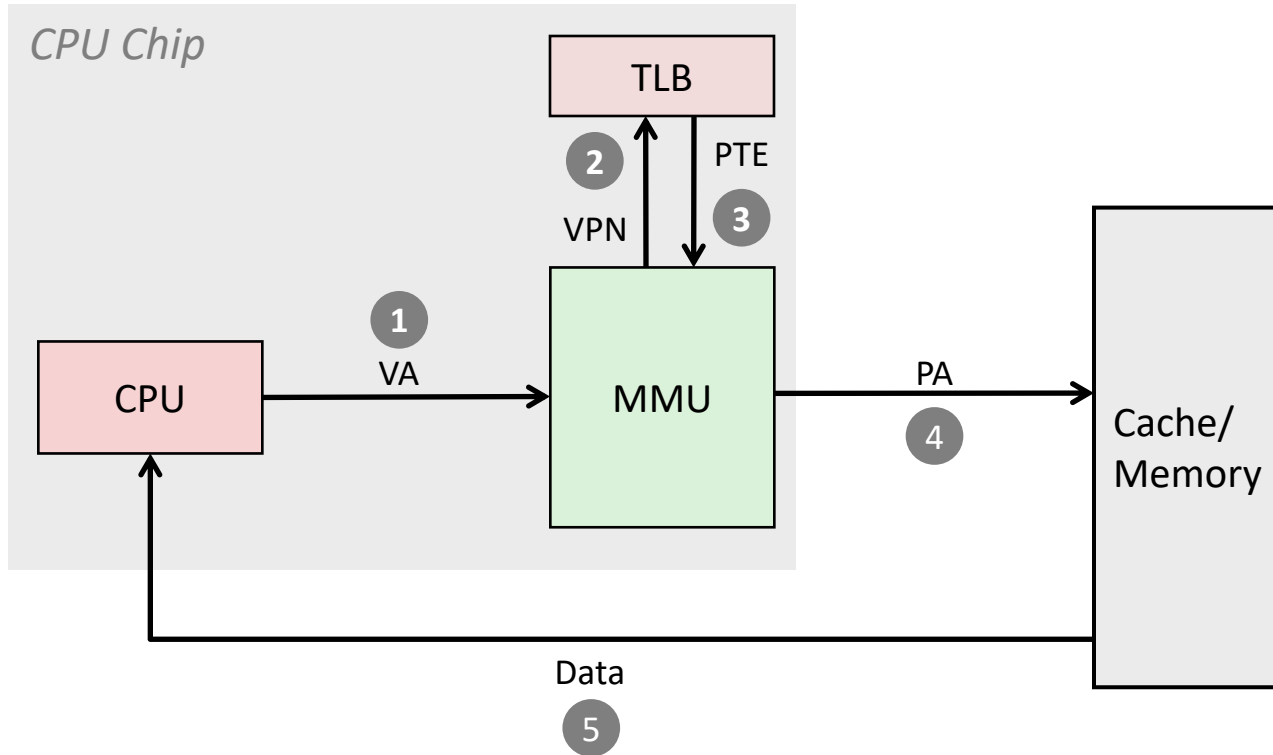
- **Page table entries (PTEs) are cached in L1 like any other memory word**
 - PTEs may be evicted by other data references
 - PTE hit still requires a small L1 delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
 - Small set-associative hardware cache in MMU
 - Maps virtual page numbers to physical page numbers
 - Contains complete page table entries for small number of pages

Accessing the TLB

- MMU uses the VPN portion of the virtual address to access the TLB:

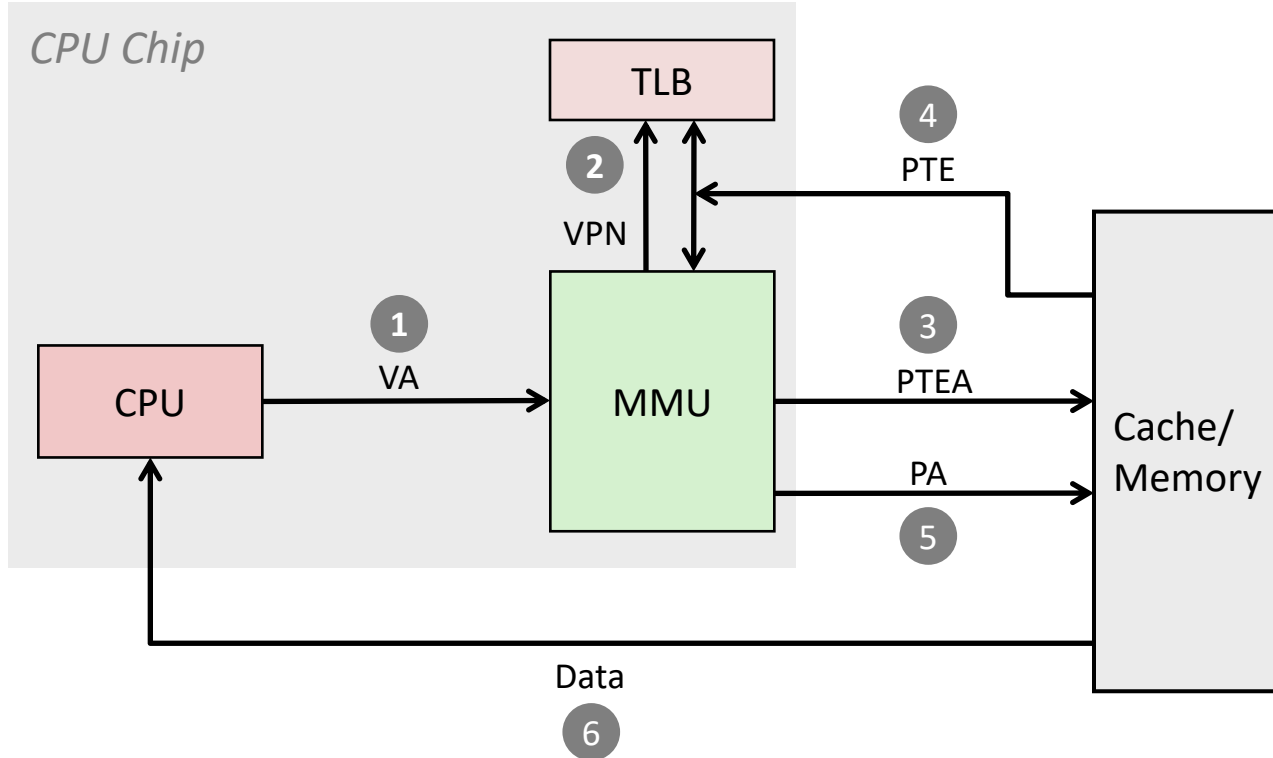


TLB Hit



A TLB hit eliminates a memory access

TLB Miss

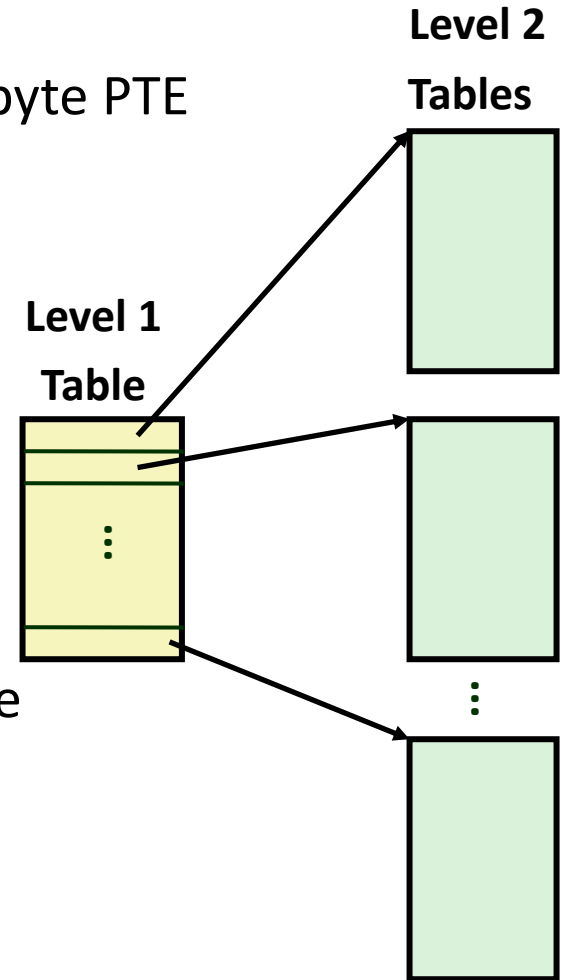


A TLB miss incurs an additional memory access (the PTE)

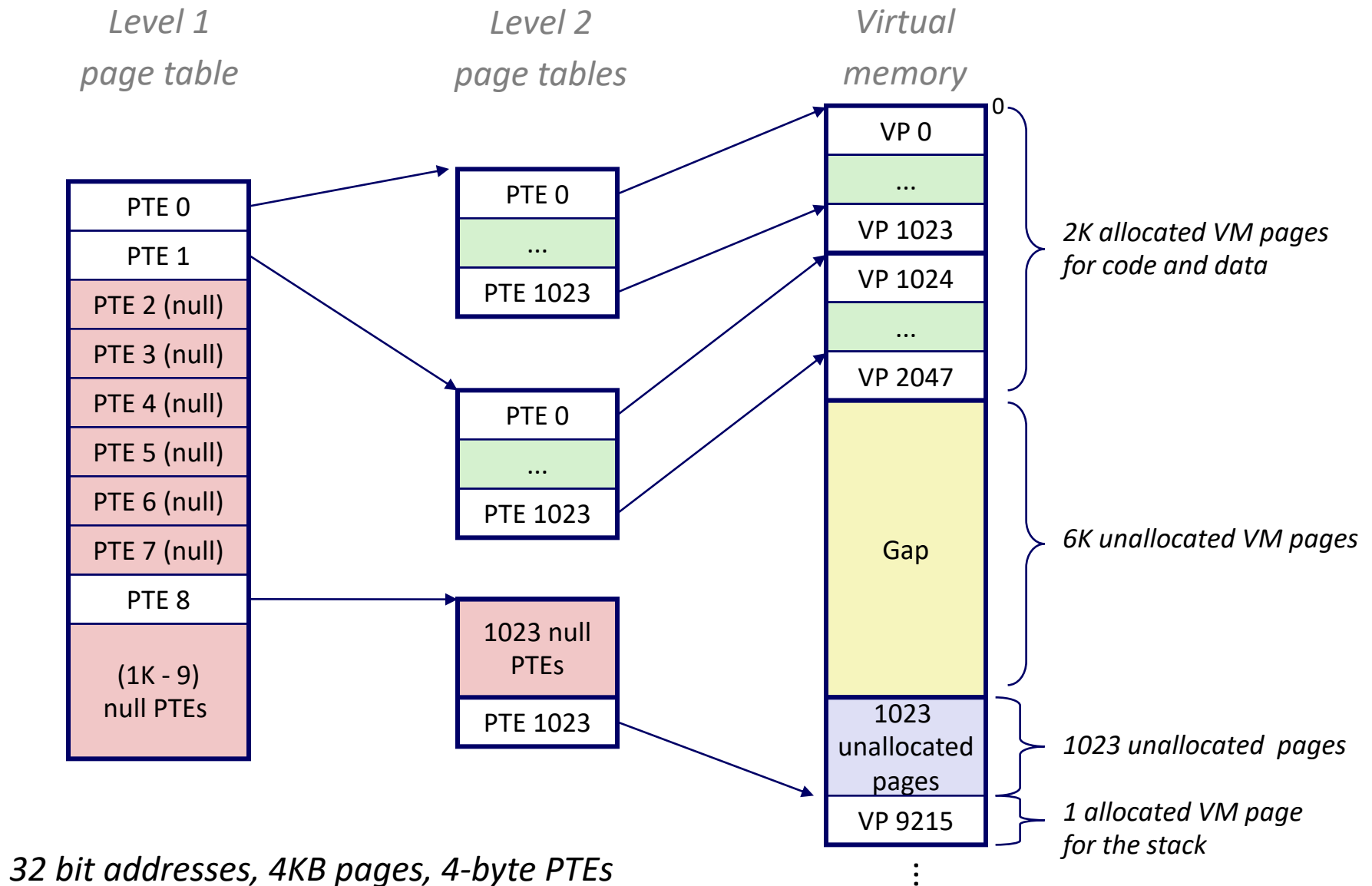
Fortunately, TLB misses are rare.

Multi-Level Page Tables

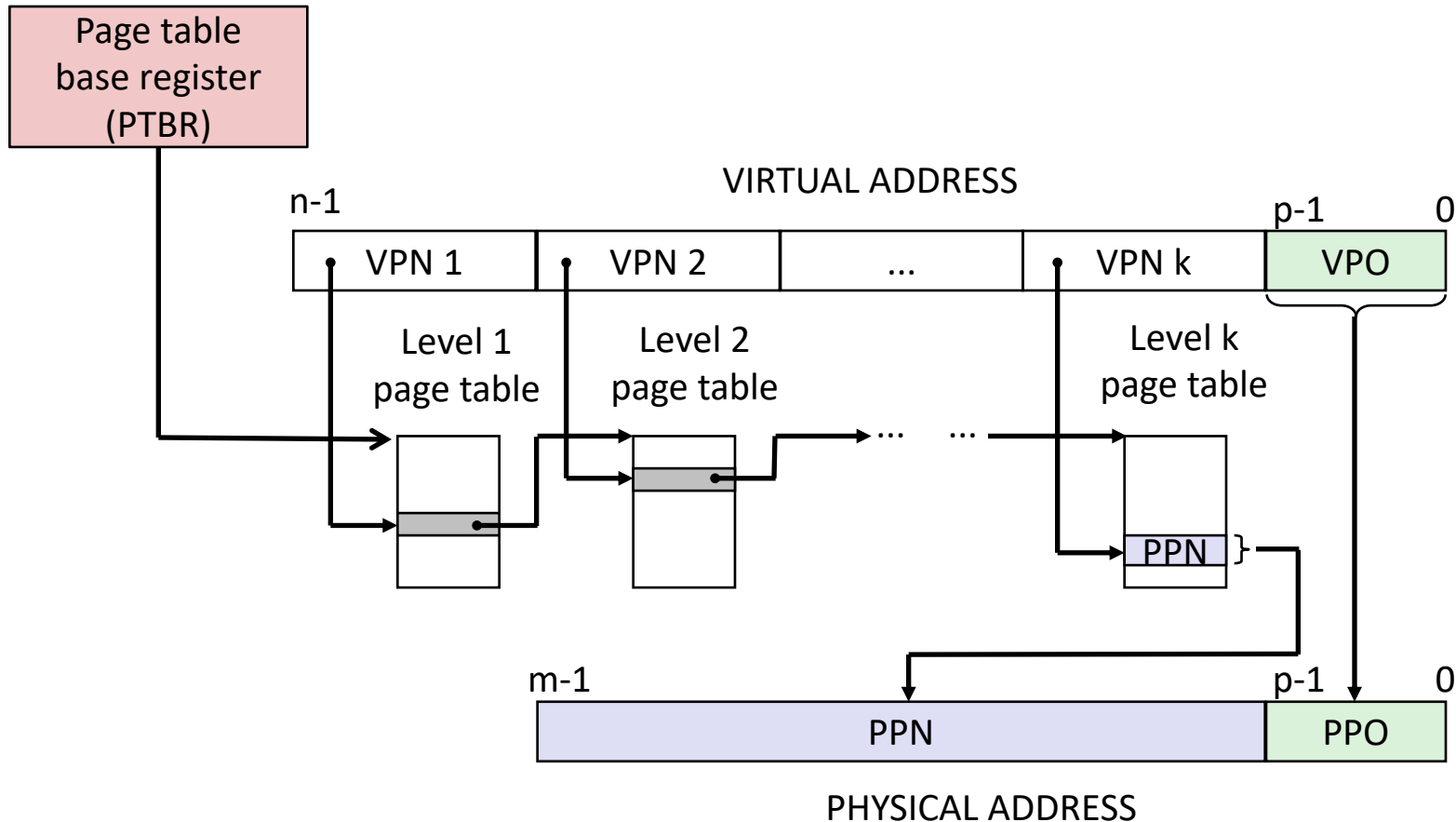
- Suppose:
 - 4KB (2^{12}) page size, 48-bit address space, 8-byte PTE
- Problem:
 - Would need a 512 GB page table!
 - $2^{48} * 2^{-12} * 2^3 = 2^{39}$ bytes
- Common solution: Multi-level page table
- Example: 2-level page table
 - Level 1 table: each PTE points to a page table (always memory resident)
 - Level 2 table: each PTE points to a page (paged in and out like any other data)



A Two-Level Page Table Hierarchy



Translating with a k-level Page Table



Summary

- Programmer's view of virtual memory
 - Each process has its own private linear address space
 - Cannot be corrupted by other processes
- System view of virtual memory
 - Uses memory efficiently by caching virtual memory pages
 - Efficient only because of locality
 - Simplifies memory management and programming
 - Simplifies protection by providing a convenient interpositioning point to check permissions