Virtual Memory I

CSE 410 Winter 2017

Instructor:

Justin Hsia

Teaching Assistants:

Kathryn Chan, Kevin Bi, Ryan Wong, Waylon Huang, Xinyu Sui

Amazon AWS S3 outage is breaking things for a lot of websites and apps

Amazon's S3 web-based storage service is experiencing widespread issues, leading to service that's either partially or fully broken on websites, apps and devices upon which it relies. Connected lightbulbs, thermostats and other IoT hardware is also being impacted, with many unable to control these devices as a result of the outage.

It's used by 0.8 percent of the top 1 million websites, which is actually quite a bit smaller than CloudFlare, which is used by 6.2 percent of the top 1 million websites globally – and yet it's still having this much of an effect.

- https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/

Administrivia

- Lab 4 due next Tuesday (3/7)
- Homework 5 released today, due next Thursday (3/9)

- Optional Section: Processes, VM Intro

- **Final Exam:** Tuesday, March 14 @ 2:30pm (MGH 241)
  - Review Session: Sunday, March 12 @ 1:30pm in SAV 264
  - Cumulative (midterm clobber policy applies)
  - TWO double-sided handwritten 8.5×11" cheat sheets
    - Recommended that you reuse or remake your midterm cheat sheet

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

- Memory & data
- Integers & floats
- x86 assembly
- Procedures & stacks
- Executables
- Arrays & structs
- Memory & caches
- Processes
- Virtual memory
- Operating Systems

Assembly language:

```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

OS:

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

Computer system:

Virtual Memory (VM*)

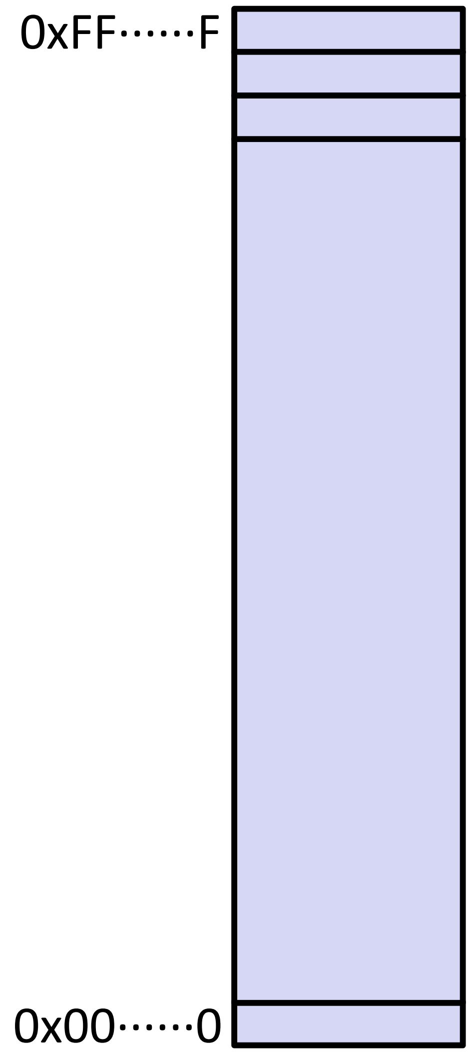
- **Overview and motivation**
- **VM as a tool for caching**
- Address translation
- VM as a tool for memory management
- VM as a tool for memory protection

Warning: Virtual memory is pretty complex, but crucial for understanding how processes work and for debugging performance

**Not to be confused with "Virtual Machine" which is a whole other thing.*

Memory as we know it so far… is *virtual!*

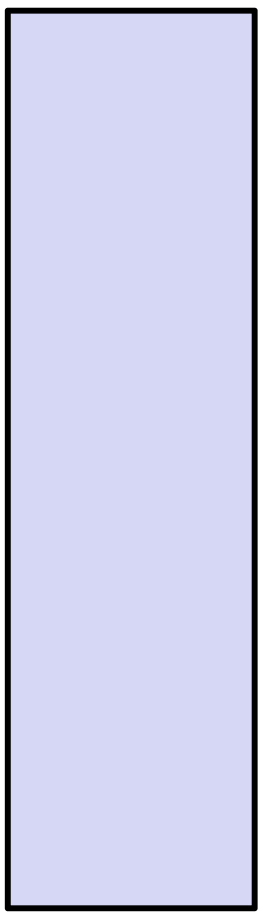
- Programs refer to virtual memory addresses
  - `movq (%rdi),%rax`
  - Conceptually memory is just a very large array of bytes
  - System provides private address space to each process
- Allocation: Compiler and run-time system
  - Where different program objects should be stored
  - All allocation within single virtual address space
- But…
  - We *probably* don't have 2^w bytes of physical memory
  - We *certainly* don't have 2^w bytes of physical memory ***for every process***
  - Processes should not interfere with one another
    - Except in certain cases where they want to share code or data

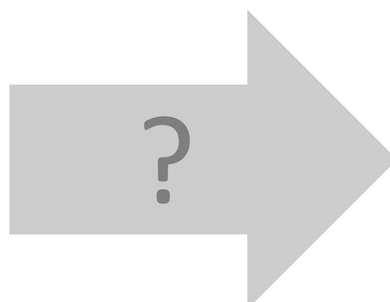


Problem 1: How Does Everything Fit?

64-bit virtual addresses can address several exabytes (18,446,744,073,709,551,616 bytes)

16 EiB





Physical main memory offers a few gigabytes (*e.g.* 8,589,934,592 bytes)

8 GiB

(Not to scale; physical memory would be smaller than the period at the end of this sentence compared to the virtual address space.)

smaller than this!

1 virtual address space per process, with many processes…

Problem 2: Memory Management

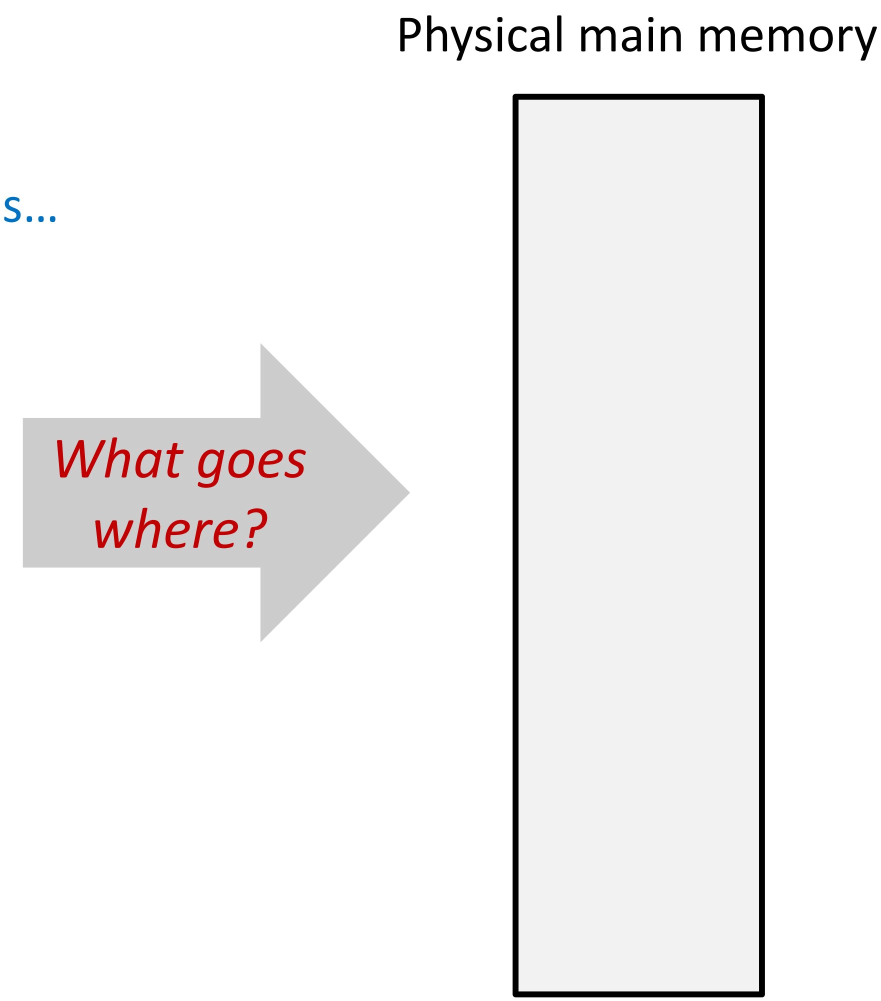
We have multiple processes:

Process 1
Process 2
Process 3
…
Process n

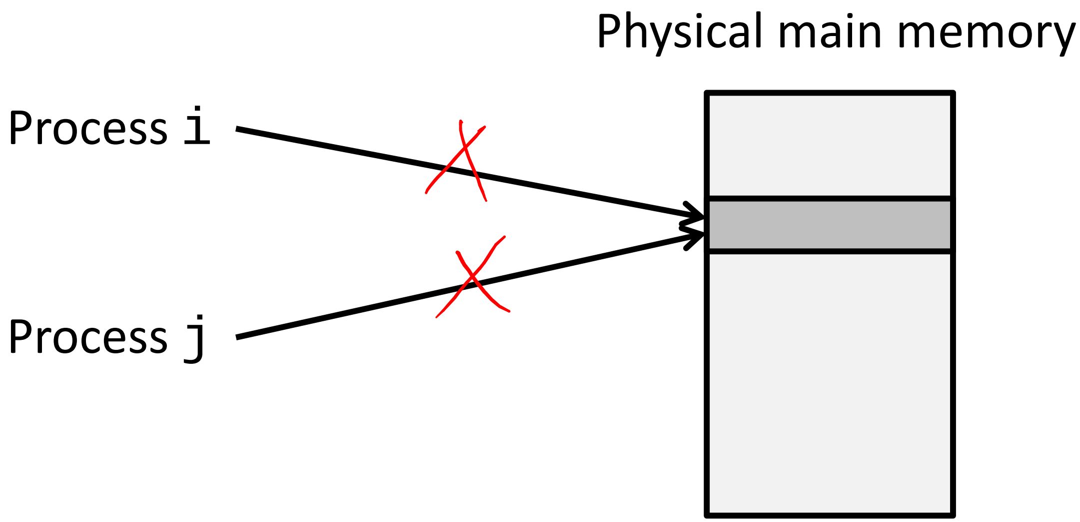
X

Each process has…

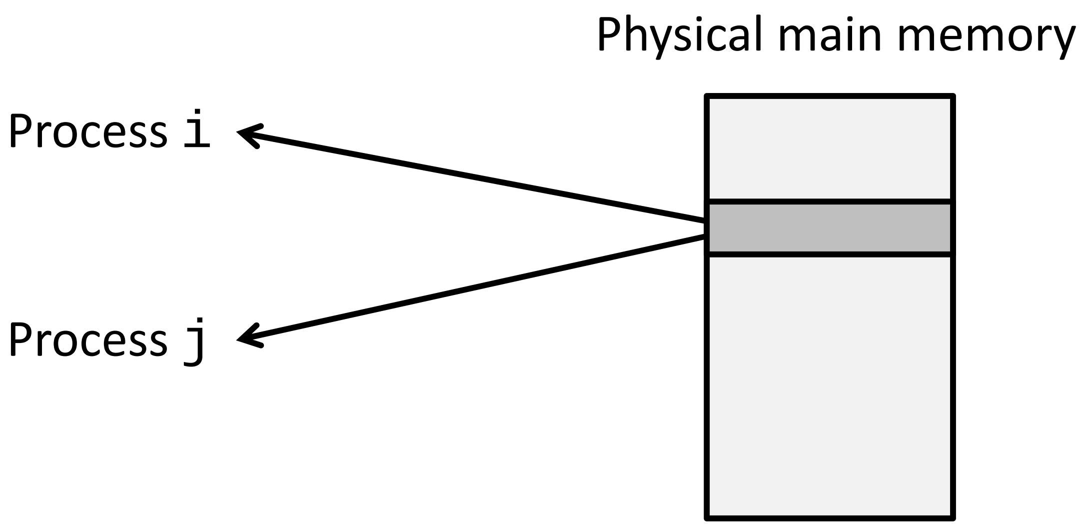
stack
heap
`.text`
`.data`
…



Problem 3: How To Protect



Problem 4: How To Share?

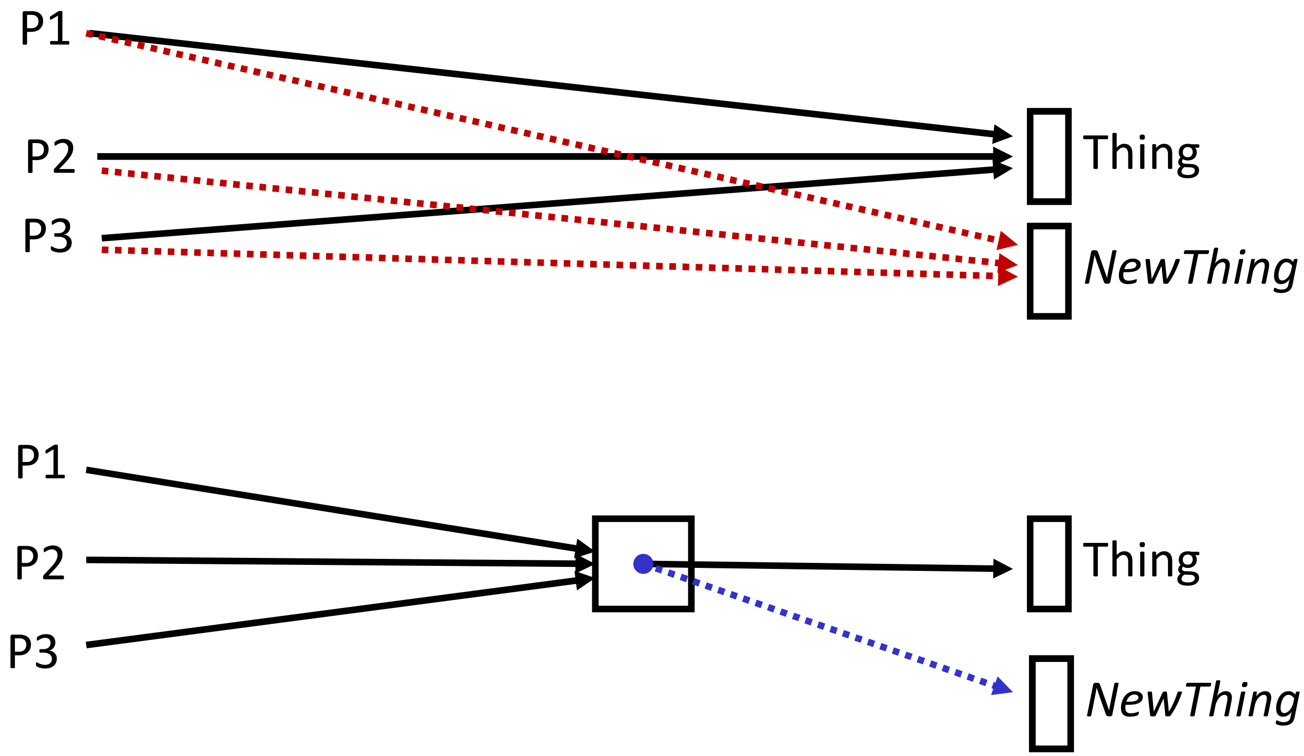


How can we solve these problems?

1) Fitting a huge address space into a tiny physical memory
2) Managing the address spaces of multiple processes
3) Protecting processes from stepping on each other's memory
4) Allowing processes to share common parts of memory

Indirection

- "Any problem in computer science can be solved by adding another level of indirection." *–David Wheeler, inventor of the subroutine*



- Without Indirection
- With Indirection

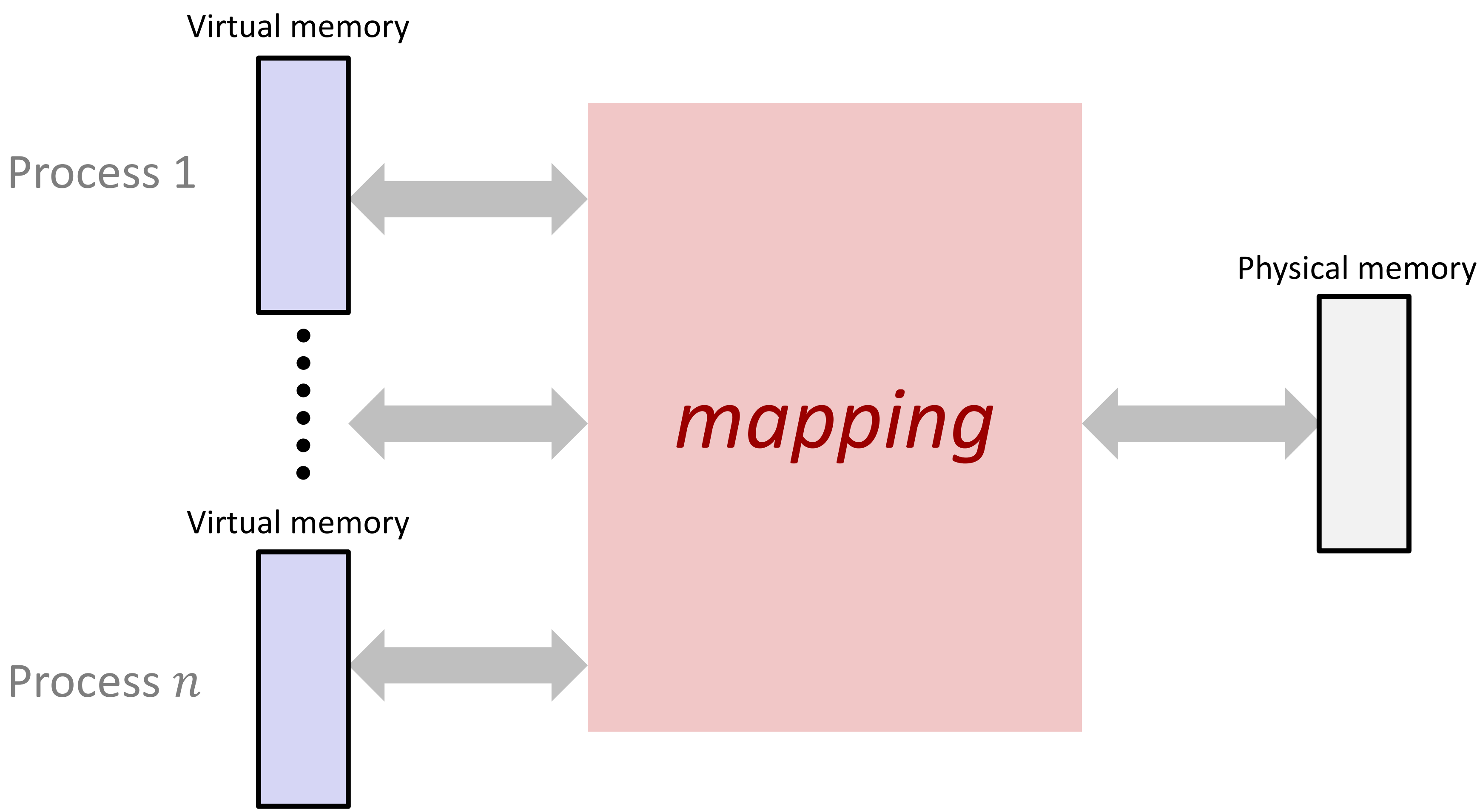
What if I want to move Thing?

Indirection

- *Indirection*: The ability to reference something using a name, reference, or container instead of the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.
  - Adds some work (now have to look up 2 things instead of 1)
  - But don’t have to track all uses of name/address (single source!)

- Examples:
  - **Phone system:** cell phone number portability
  - **Domain Name Service (DNS):** translation from name to IP address
  - **Call centers:** route calls to available operators, etc.
  - **Dynamic Host Configuration Protocol (DHCP):** local network address assignment

Indirection in Virtual Memory



- Each process gets its own private virtual address space
- Solves the previous problems!

Address Spaces

bits

ceiling function (round up)

$n = \lceil \log_2 N \rceil$

- Virtual address space: Set of $N = 2^n$ virtual addr
  - {0, 1, 2, 3, …, N-1}

bytes

$m = \lceil \log_2 M \rceil$

- Physical address space: Set of $M = 2^m$ physical addr
  - {0, 1, 2, 3, …, M-1}

- Every byte in main memory has:
  - one physical address (PA)
  - zero, one, *or more* virtual addresses (VAs)

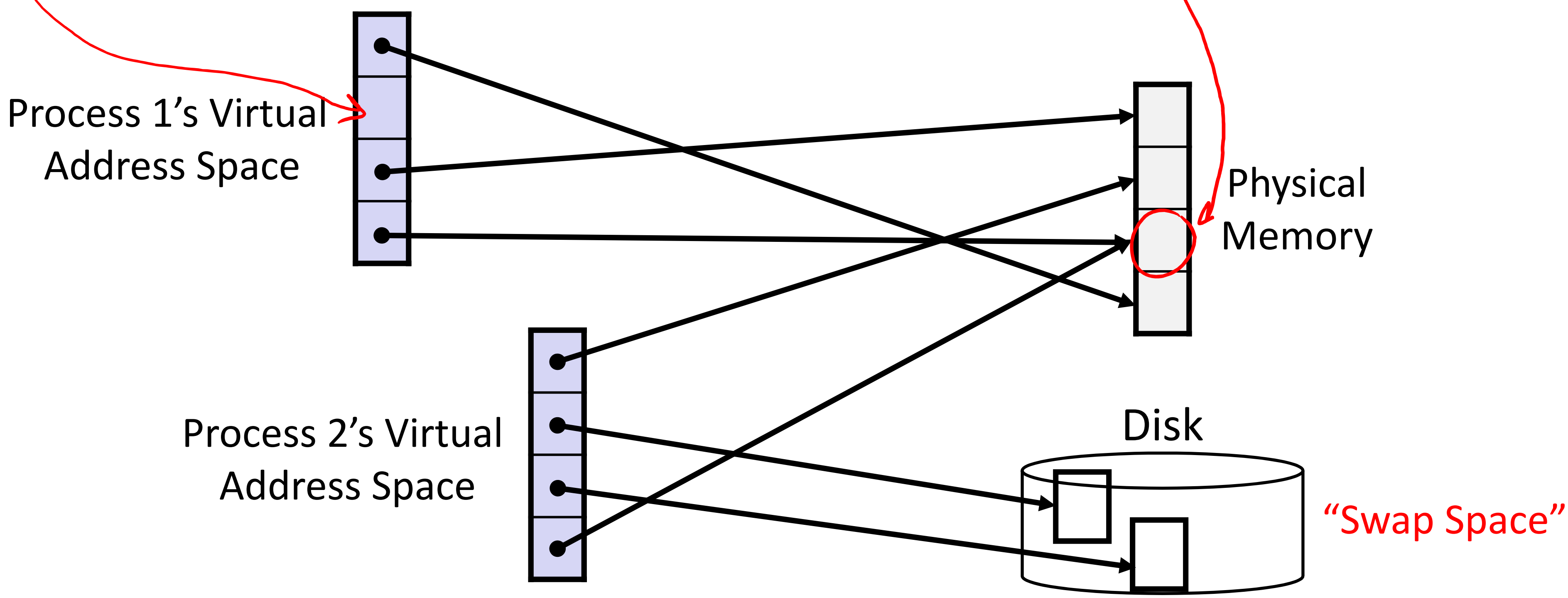
unused

used by one process

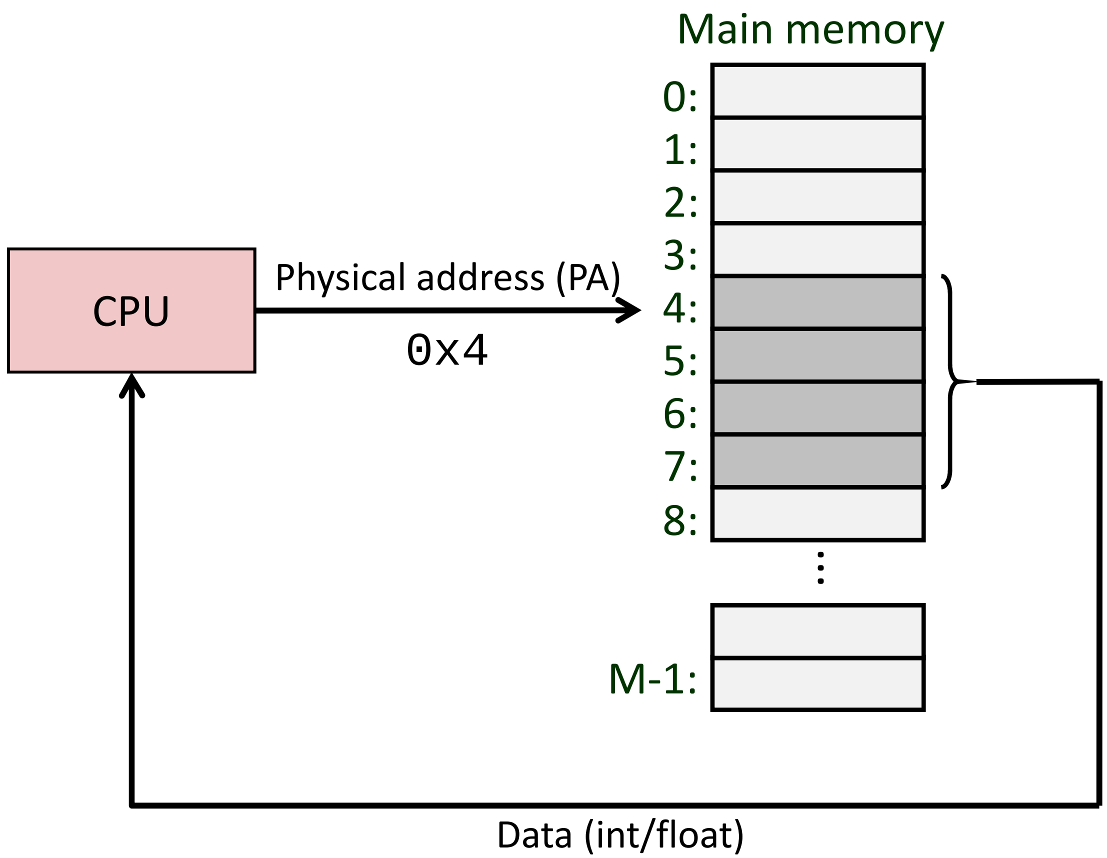
used by many processes

Mapping

- A virtual address (VA) can be mapped to either physical memory or disk
  - Unused VAs may not have a mapping
  - VAs from *different* processes may map to same location in memory/disk

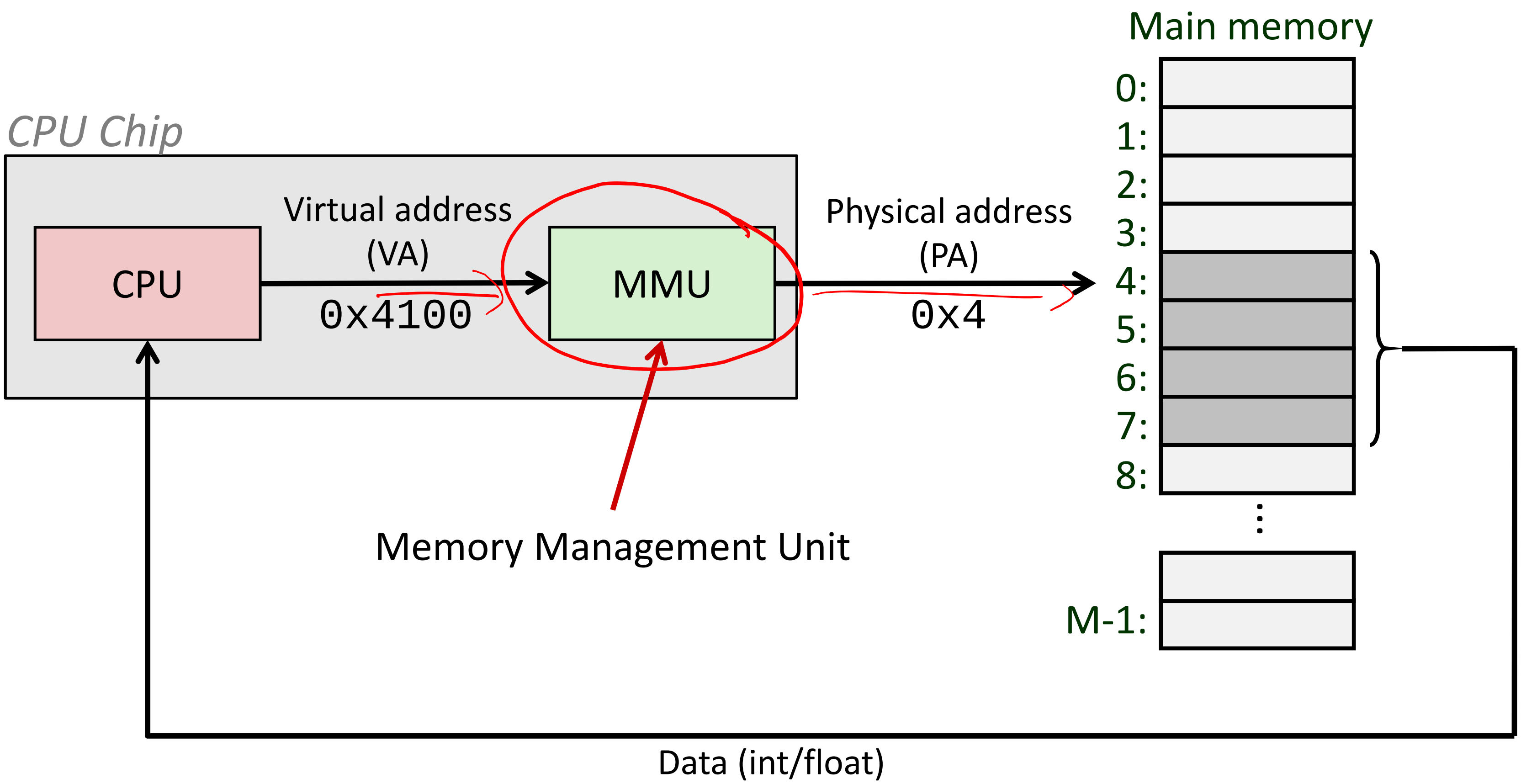


A System Using Physical Addressing



- Used in “simple” systems with (usually) just one process:
  - Embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing



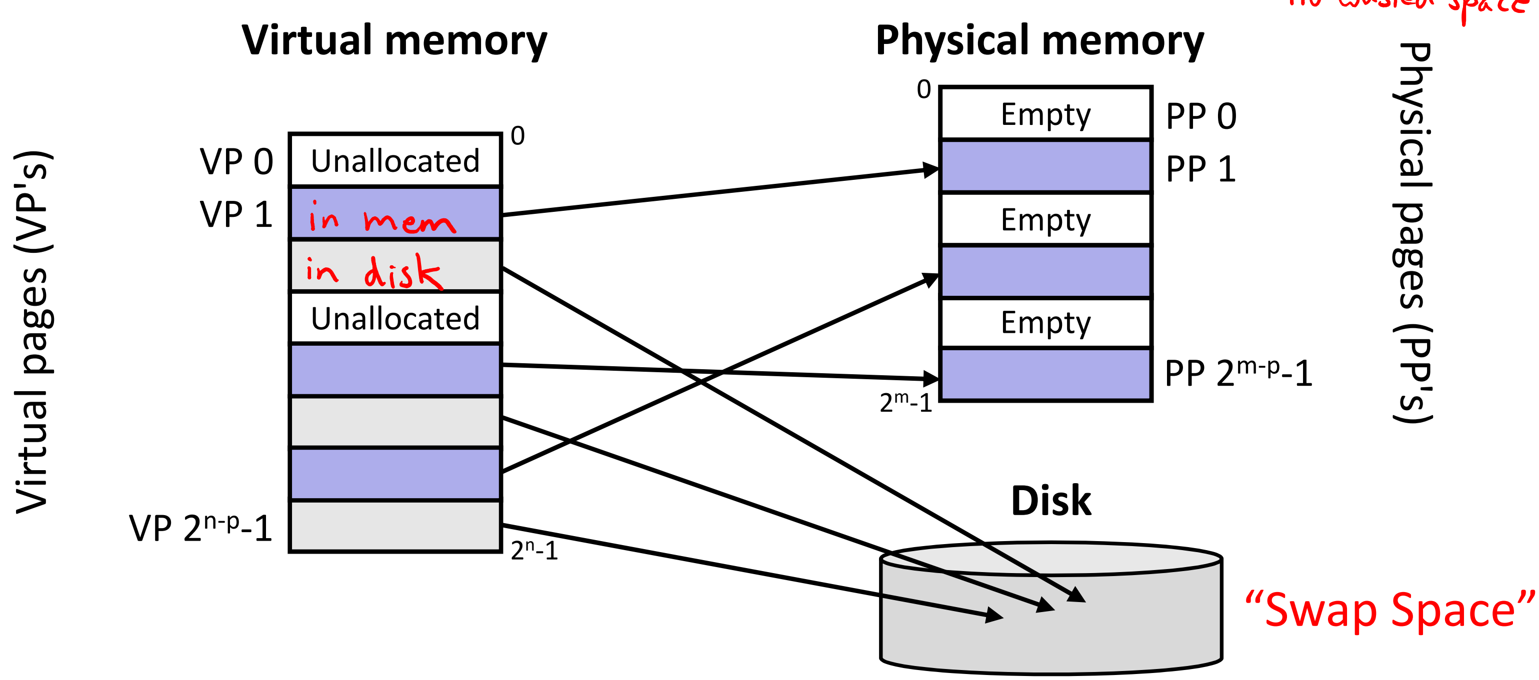
- Physical addresses are *completely invisible to programs*
  - Used in all modern desktops, laptops, servers, smartphones…
  - One of the great ideas in computer science

Why Virtual Memory (VM)?

- Efficient use of limited main memory (RAM)
  - Use RAM as a cache for the parts of a virtual address space
    - Some non-cached parts stored on disk
    - Some (unallocated) non-cached parts stored nowhere
  - Keep only active areas of virtual address space in memory
    - Transfer data back and forth as needed
- Simplifies memory management for programmers
  - Each process "gets" the same full, private linear address space
- Isolates address spaces (protection)
  - One process can't interfere with another's memory
    - They operate in *different address spaces*
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions

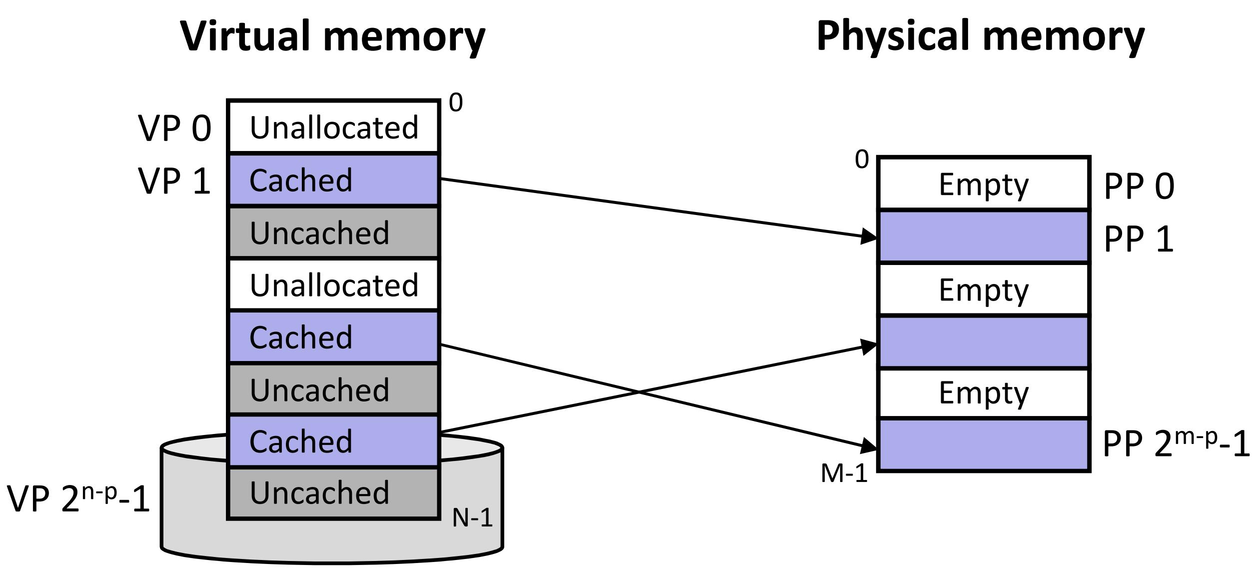
VM and the Memory Hierarchy

- Think of virtual memory as array of $N = 2^n$ contiguous bytes
- *Pages* of virtual memory are usually stored in physical memory, but sometimes spill to disk
  - Pages are another unit of aligned memory (size is $P = 2^p$ bytes) $p = \lceil \log_2 P \rceil$
  - Each virtual page can be stored in *any* physical page (no fragmentation!) no wasted space/gaps



or: Virtual Memory as DRAM Cache for Disk

- Think of virtual memory as an array of $\mathrm{N} = 2^n$ contiguous bytes stored *on a disk*
- Then physical main memory is used as a *cache* for the virtual memory array
  - These "cache blocks" are called *pages* (size is $\mathrm{P} = 2^p$ bytes)

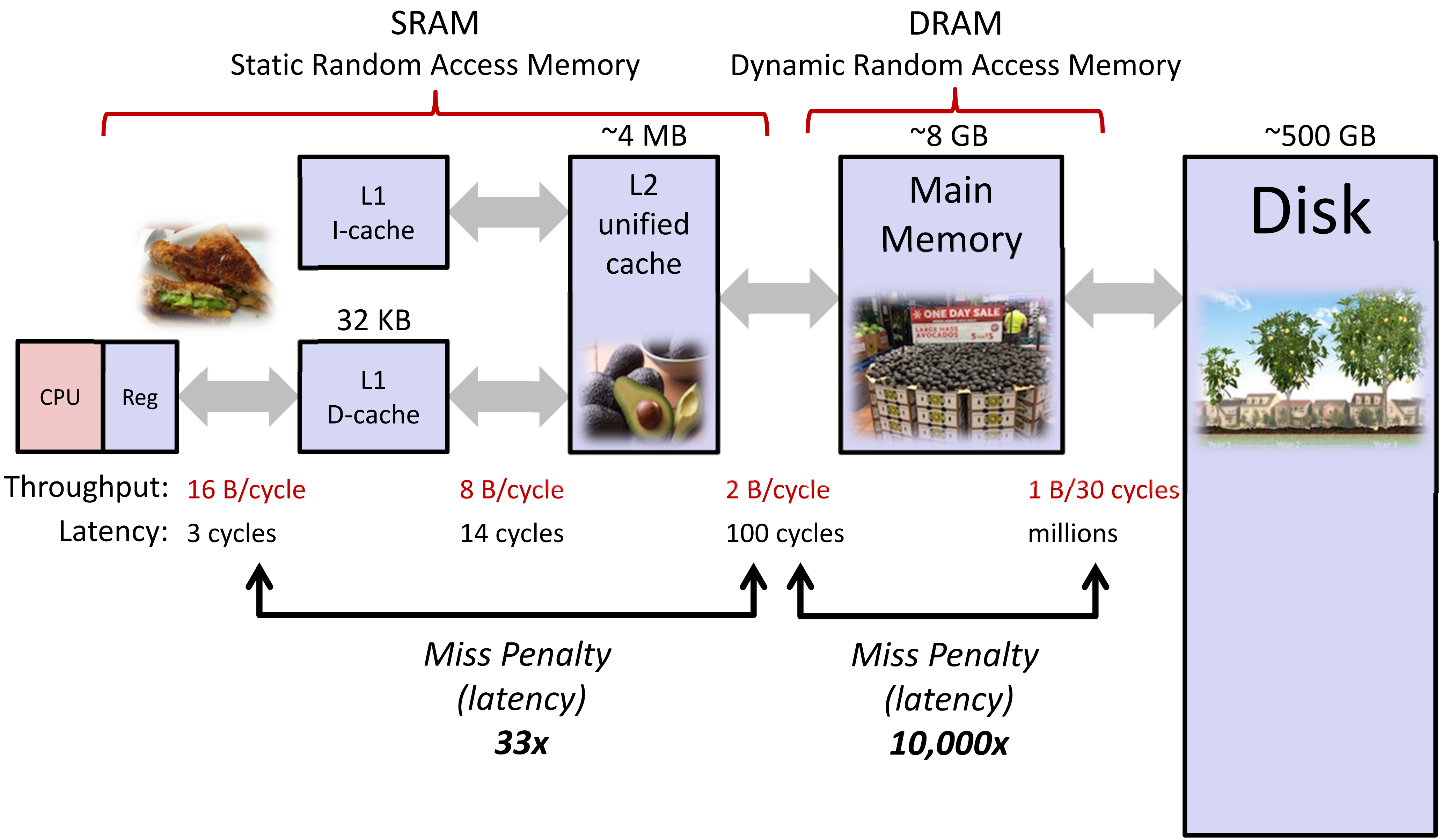


Virtual pages (VPs) "stored on disk"

Physical pages (PPs) cached in DRAM

Memory Hierarchy: Core 2 Duo

Not drawn to scale



Virtual Memory Design Consequences

- Large page size: typically 4-8 KiB or 2-4 MiB
  - *Can* be up to 1 GiB (for "Big Data" apps on big computers)
  - Compared with 64-byte cache blocks
- Fully associative (physical memory is single set)
  - Any virtual page can be placed in any physical page
  - Requires a "large" mapping function – different from CPU caches
- Highly sophisticated, expensive replacement algorithms in OS
  - Too complicated and open-ended to be implemented in hardware
- *Write-back* rather than *write-through* (track dirty pages)
  - *Really* don't want to write to disk every time we modify something in memory
  - Some things may never end up on disk (*e.g.* stack for short-lived process)

Why does VM work on RAM/disk?

- Avoids disk accesses because of *locality*
  - Same reason that L1 / L2 / L3 caches work

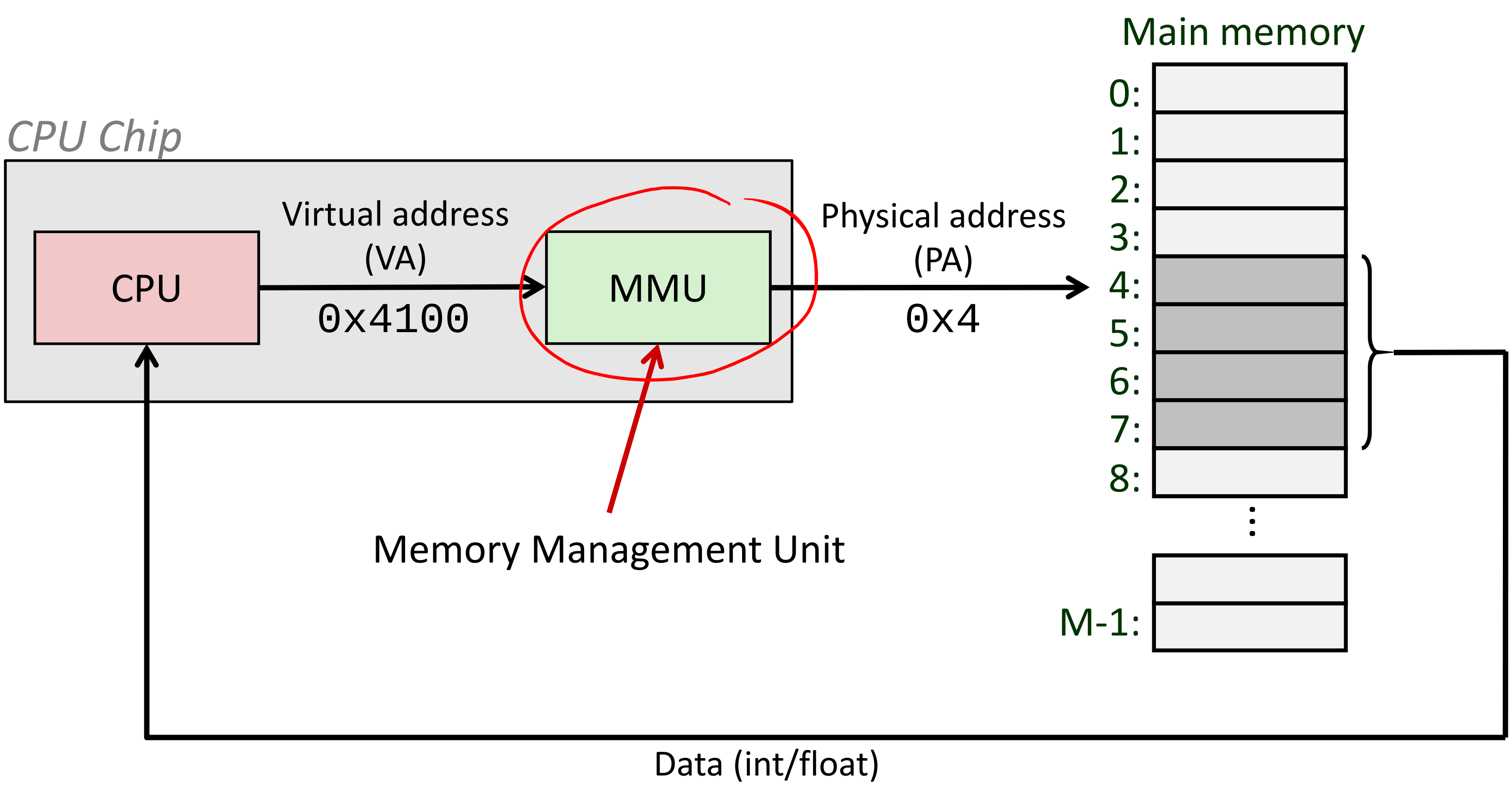
- The set of virtual pages that a program is "actively" accessing at any point in time is called its *working set*
  - If (*working set of one process* ≤ *physical memory*):
    - Good performance for one process (after compulsory misses)
  - If (*working sets of all processes* > *physical memory*):
    - ***Thrashing:*** Performance meltdown where pages are swapped between memory and disk continuously (CPU always waiting or paging)
    - This is why your computer can feel faster when you add RAM

Virtual Memory (VM)

- Overview and motivation
- VM as a tool for caching
- **Address translation**
- VM as a tool for memory management
- VM as a tool for memory protection

Address Translation

How do we perform the virtual → physical address translation?



Address Translation: Page Tables

VPN width $n-p \Leftrightarrow$ we have 2^{n-p} pages in VA space

page size P bytes $\Leftrightarrow p = \lceil \log_2 P \rceil$ bits

- CPU-generated address can be split into:

n-bit address: $\underbrace{\text{Virtual Page Number}}_{n-p\text{ bits}}$ | $\underbrace{\text{Page Offset}}_{p\text{ bits}}$

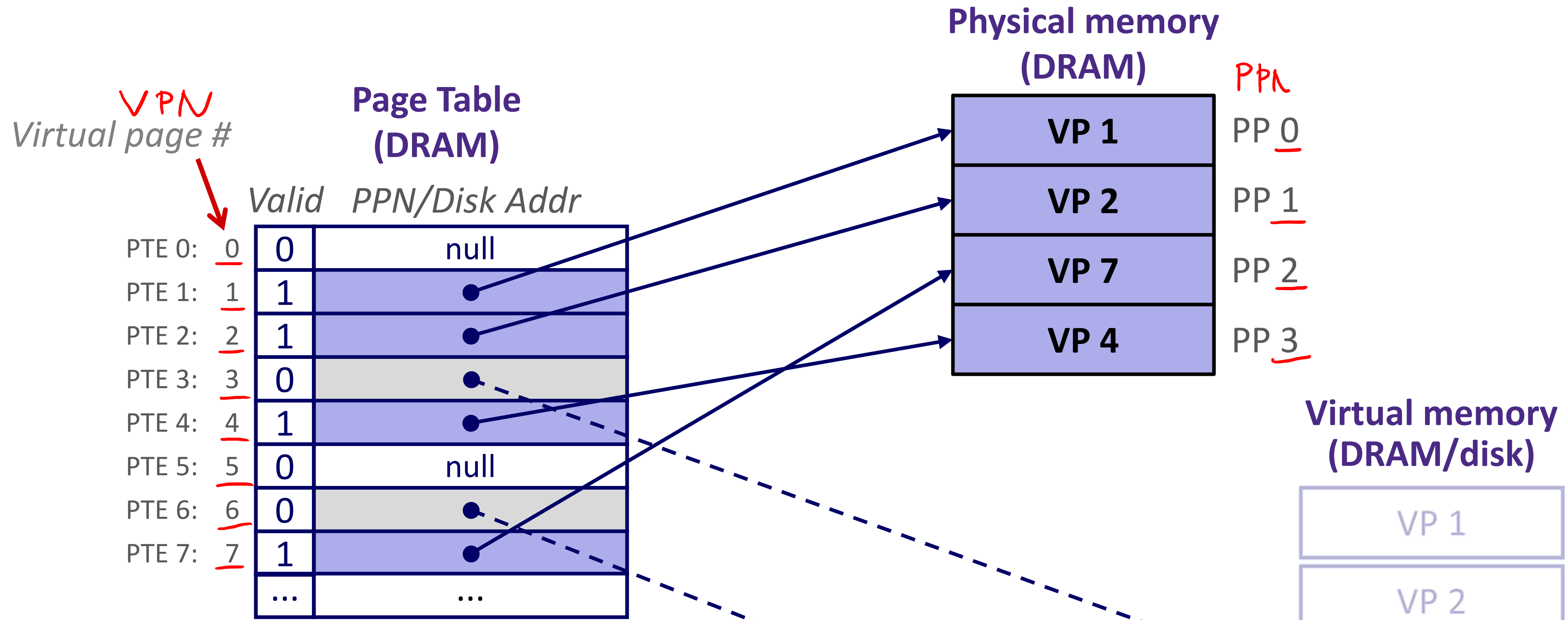
analogous to: block number | block offset for caches

  - Request is Virtual Address (VA), want Physical Address (PA)
  - Note that Physical Offset = Virtual Offset (page-aligned)

- Use lookup table that we call the *page table* (PT)
  - Replace Virtual Page Number (VPN) for Physical Page Number (PPN) to generate Physical Address
  - Index PT using VPN: page table entry (PTE) stores the PPN plus management bits (*e.g.* Valid, Dirty, access rights)
  - Has an entry for *every* virtual page – why?

    no backup for mappings (which can be anything)

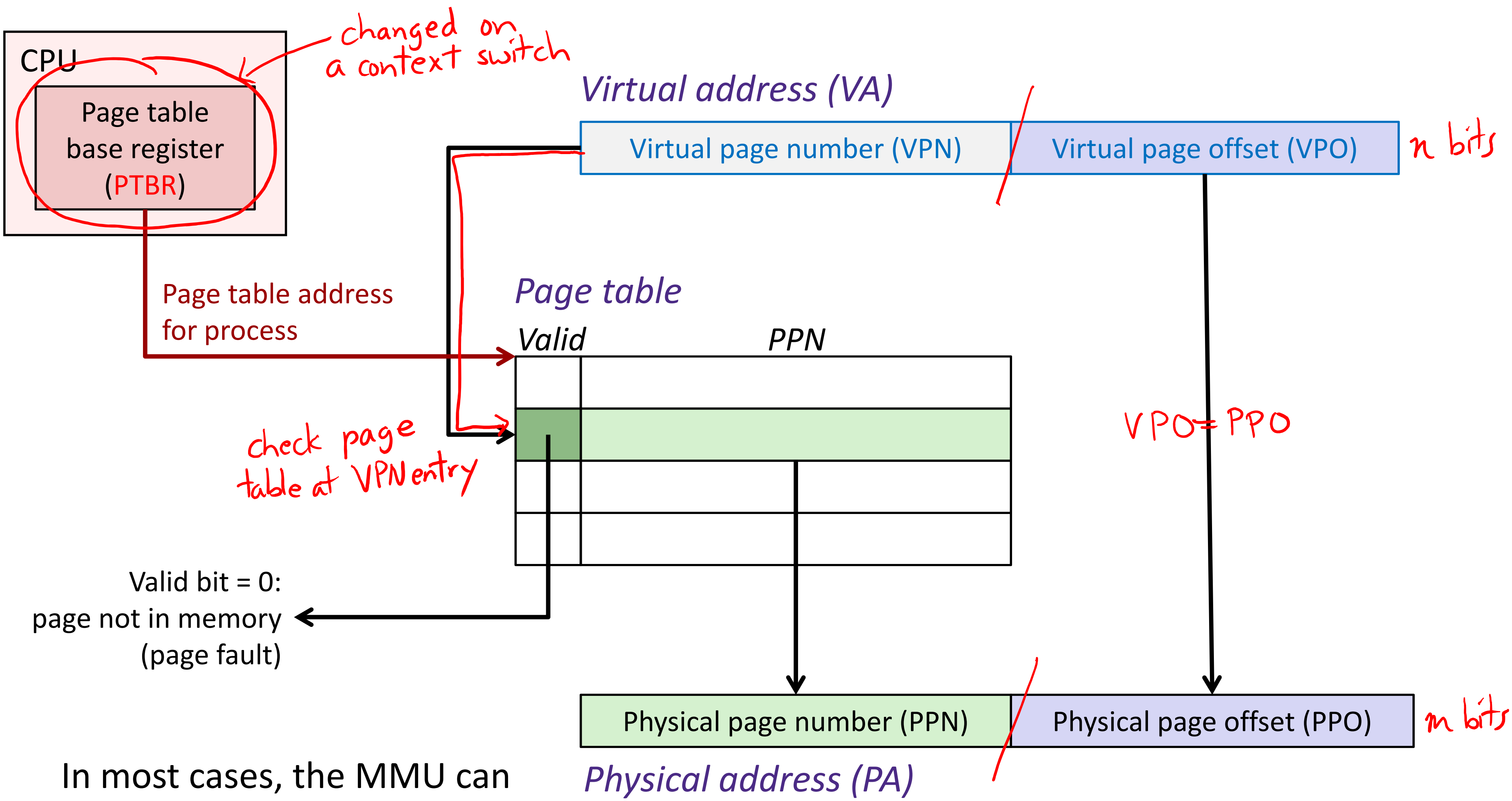
Page Table Diagram



page table has 2^{n-p} entries!

- Page tables stored in physical memory
  - Too big to fit elsewhere – managed by MMU & OS
- How many page tables in the system?
  - *One per process*

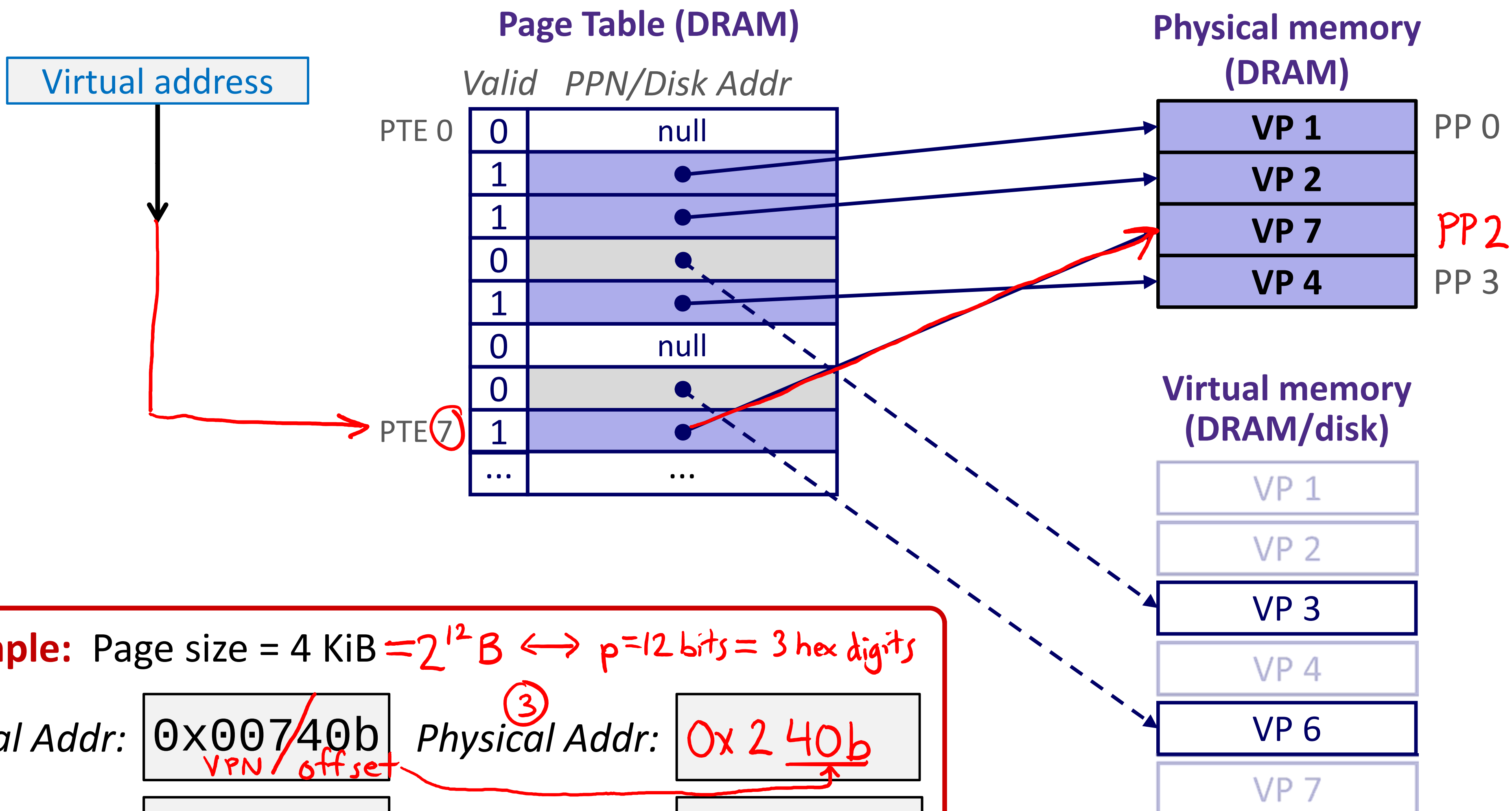
Page Table Address Translation



In most cases, the MMU can perform this translation without software assistance

Page Hit

- ***Page hit:*** VM reference is in physical memory



Example: Page size = 4 KiB $= 2^{12}$ B $\leftrightarrow$ p=12 bits = 3 hex digits

Virtual Addr: 0x00740b VPN / offset

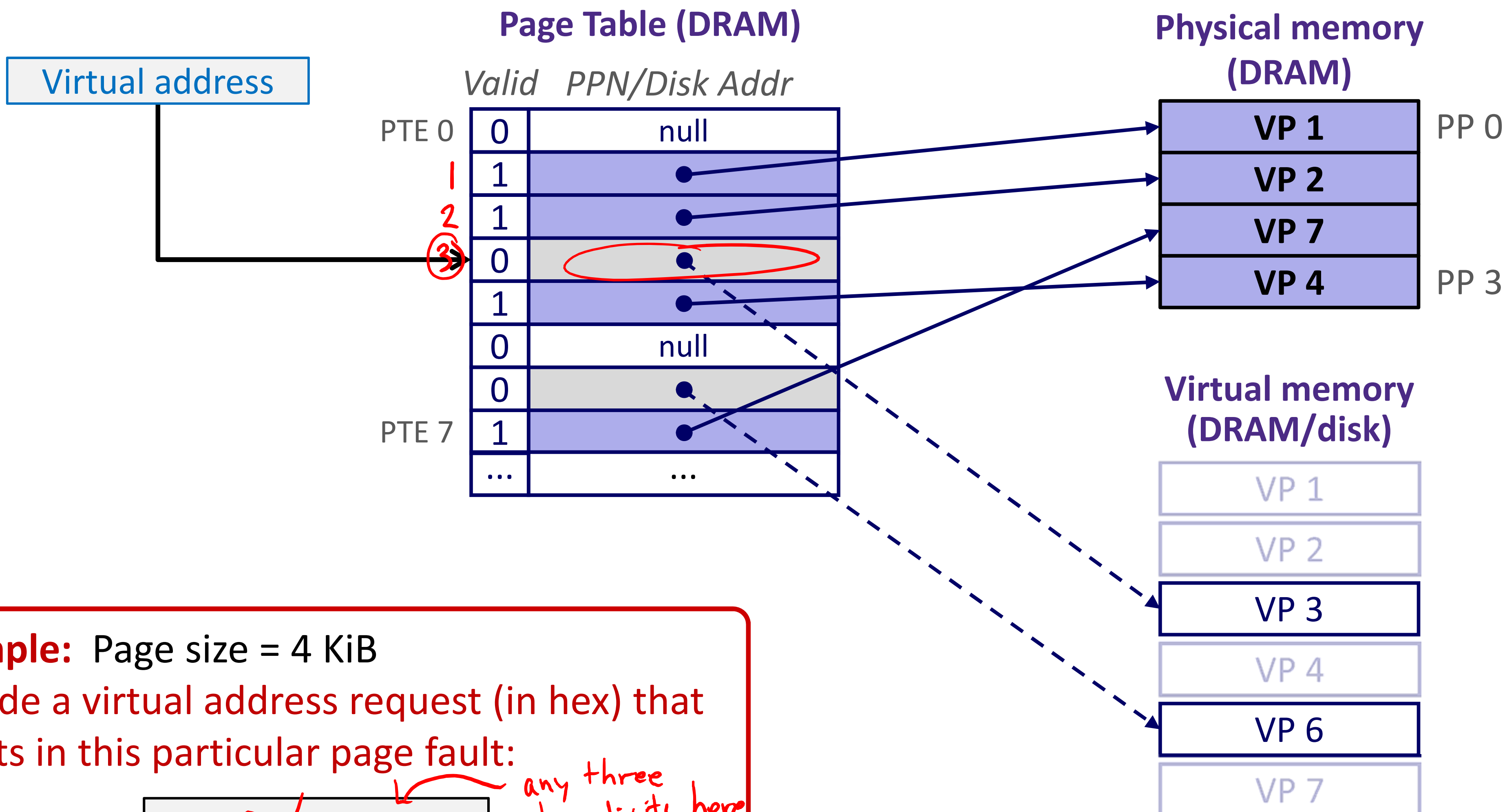
3 *Physical Addr:* 0x 2 40b

1 *VPN:* 7

2 *PPN:* 2

Page Fault

- ***Page fault:*** VM reference is NOT in physical memory



Example: Page size = 4 KiB
Provide a virtual address request (in hex) that results in this particular page fault:

Virtual Addr: 0x 3/000

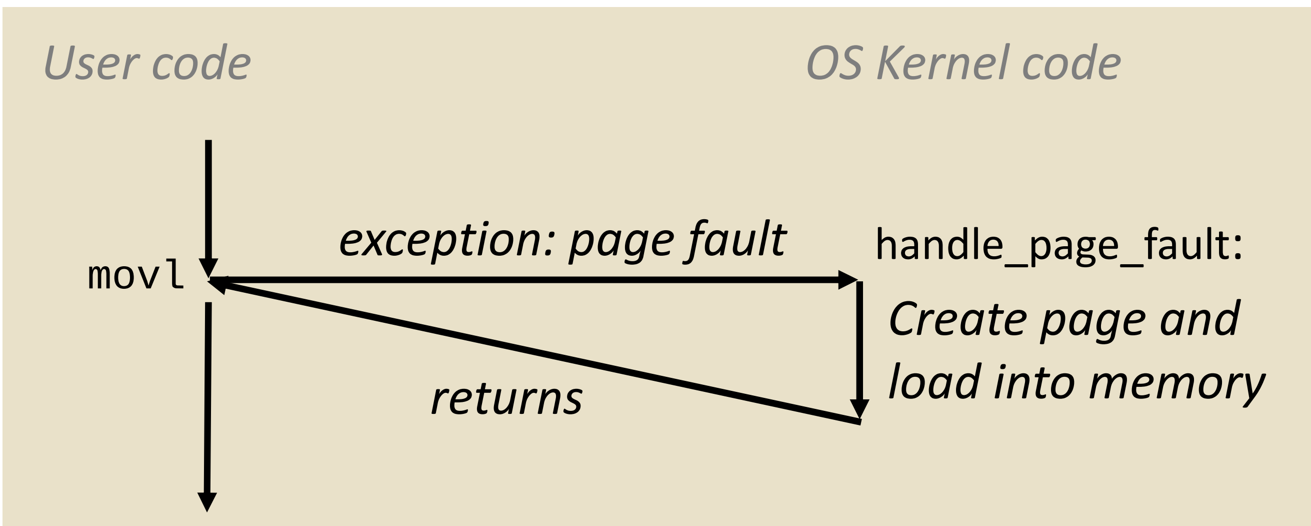
any three hex digits here

Page Fault Exception

```
int a[1000];
int main ()
{
    a[500] = 13;
}
```

- User writes to memory location
- That portion (page) of user's memory is currently on disk

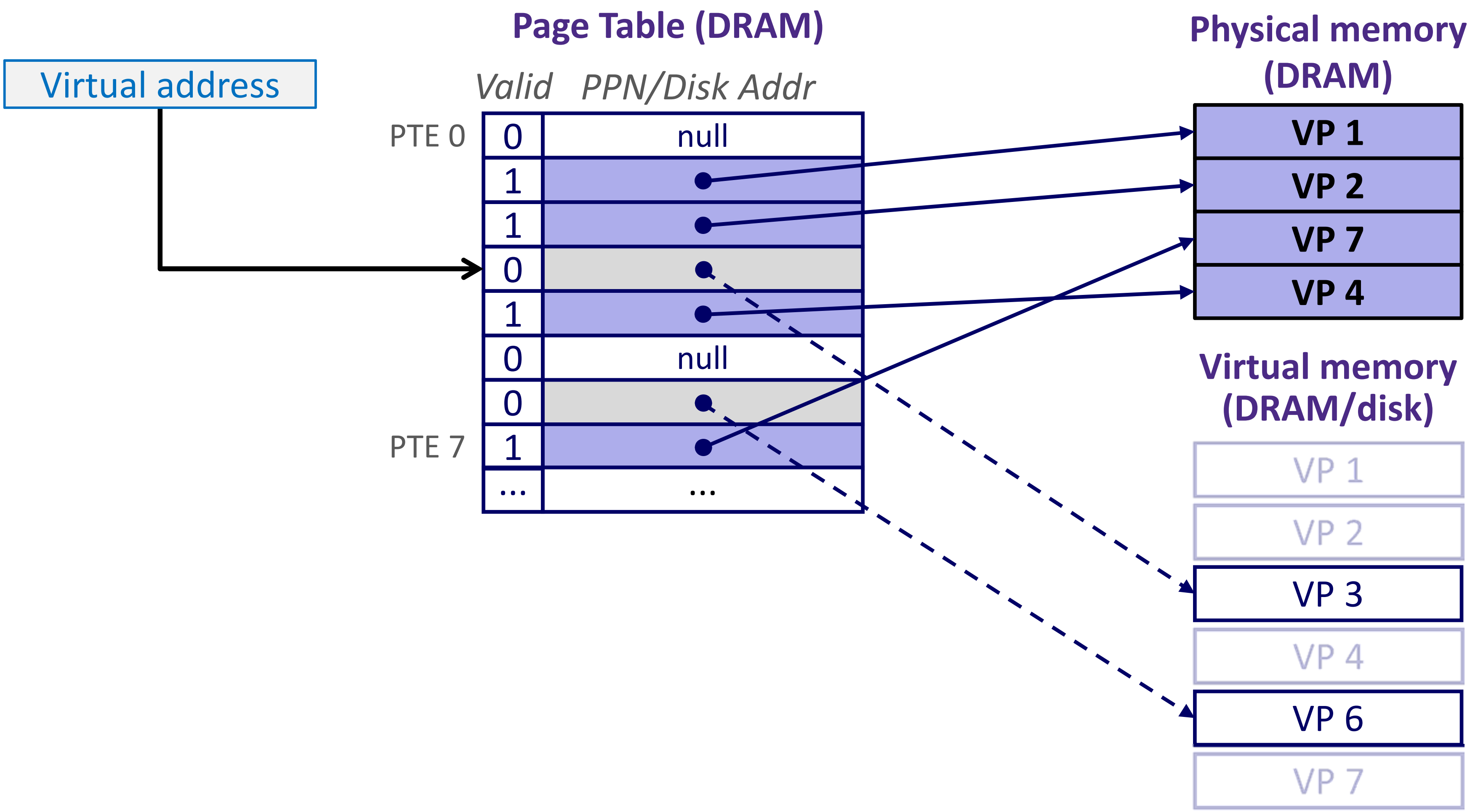
```
80483b7:       c7 05 10 9d 04 08 0d    movl    $0xd,0x8049d10
```



- Page fault handler must load page into physical memory
- Returns to faulting instruction: `mov` is executed again!
  - Successful on second try

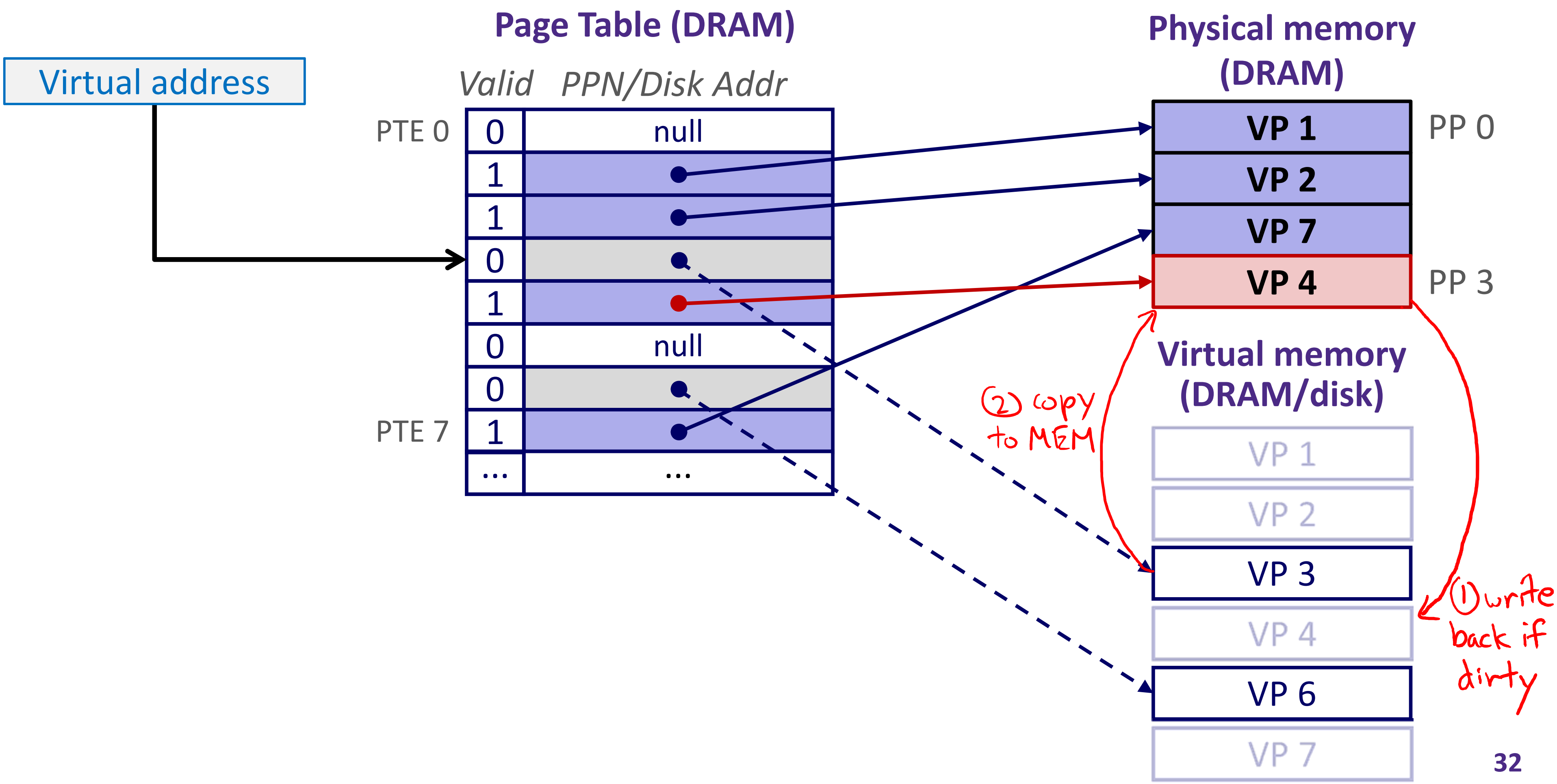
Handling a Page Fault

- Page miss causes page fault (an exception)



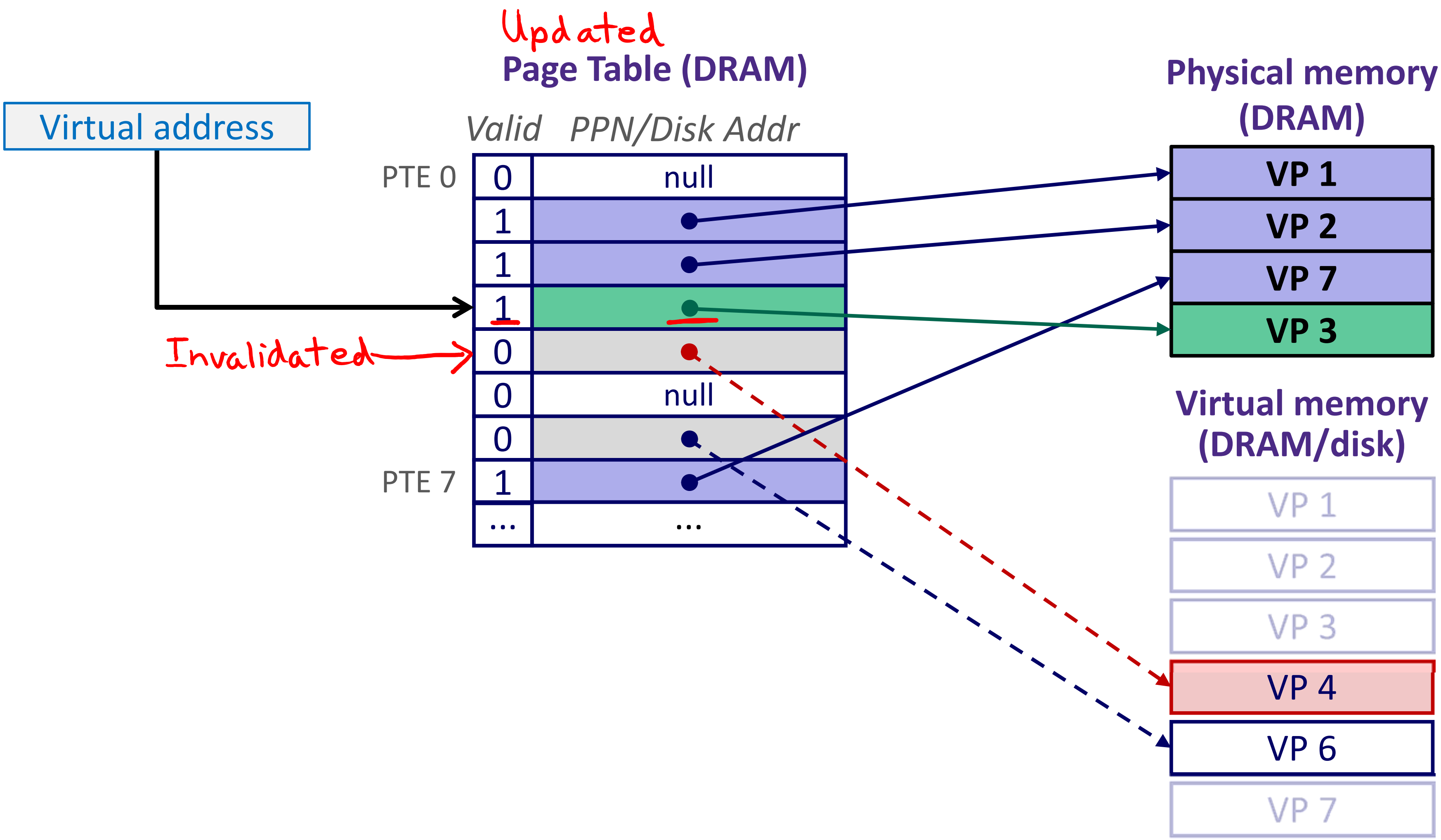
Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a *victim* to be evicted (here VP 4) PP3



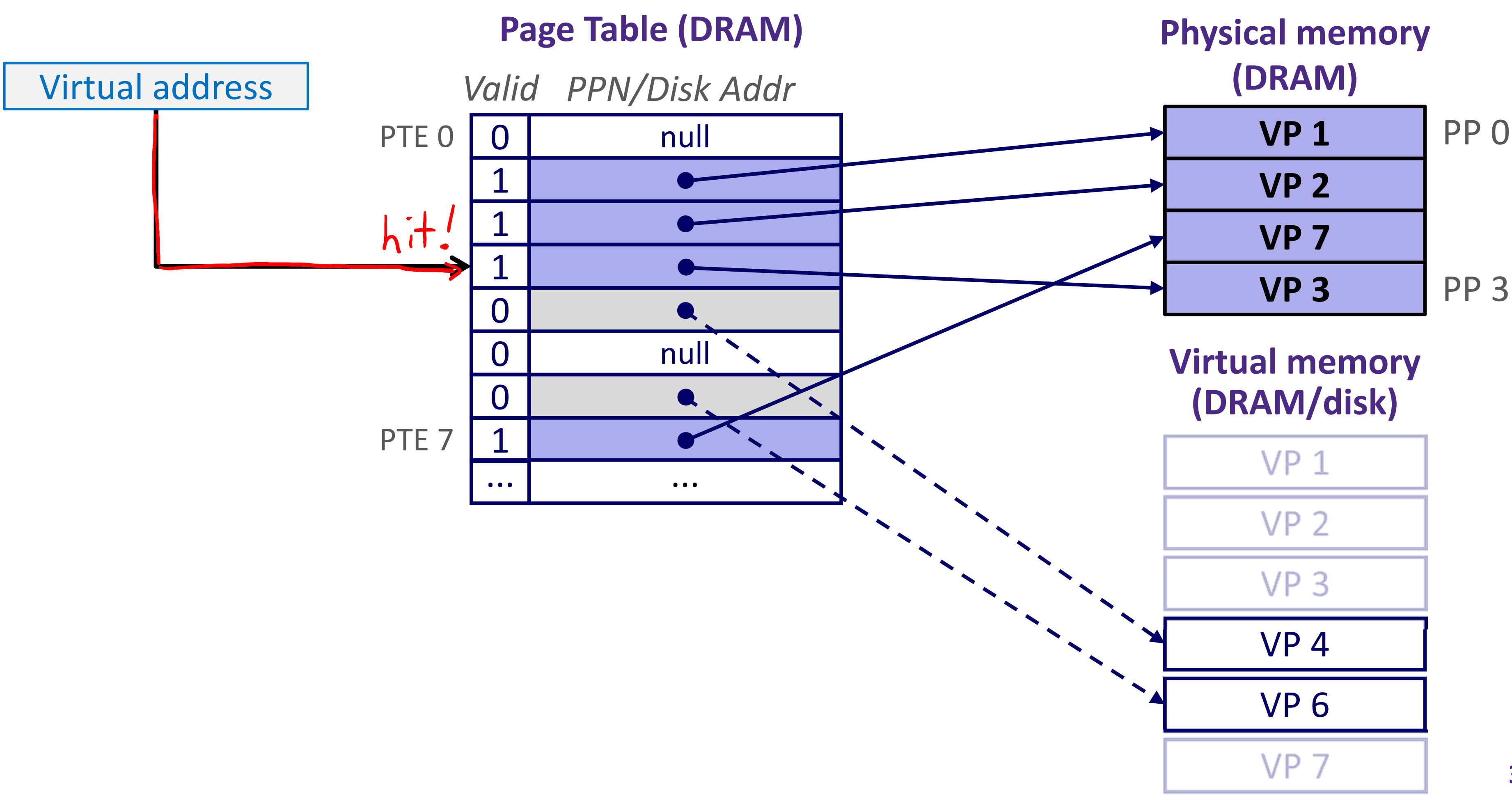
Handling a Page Fault

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



Handling a 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!



Peer Instruction Question

- How many bits wide are the following fields?
  - 16 KiB pages (2^4 2^{10}) $p = 14$ bits
  - 48-bit virtual addresses $n = 48$ bits $\longleftrightarrow$ 256 TiB virtual memory
  - 16 GiB physical memory (2^4 2^{30}) $m = 34$ bits
  - Vote at: http://PollEv.com/justinh

	VPN	PPN
(A)	34	24
(B)	32	18
(C)	30	20
(D)	34	20

VPN width $= n - p = 34$ bits $\longleftrightarrow 2^{34}$ pages in virtual address space

PPN width $= m - p = 20$ bits $\longleftrightarrow 2^{20}$ pages in physical address space

Summary

- Virtual memory provides:
  - Ability to use limited memory (RAM) across multiple processes
  - Illusion of contiguous virtual address space for each process
  - Protection and sharing amongst processes
- Indirection via address mapping by page tables
  - Part of memory management unit and stored in memory
  - Use virtual page number as index into lookup table that holds physical page number, disk address, or NULL (unallocated page)
  - On page fault, throw exception and move page from swap space (disk) to main memory