Lecture 16
Virtual Memory

CS213 – Intro to Computer Systems

Branden Ghena – Spring 2021

Slides adapted from:
St-Amour, Hardavellas, Bustamente (Northwestern), Bryant, O'Hallaron (CMU), Garcia, Weaver (UC Berkeley)

Administrivia

- Bomb Lab Secret Phase
  - Completed by:
  - Charles, Amogh, Joshua, Ben, Akhil, Hannah, Kelley, Dilan, and Alan

  - Winner of TIS-100 is…

Administrivia

- Bomb Lab Secret Phase
  - Completed by:
  - Charles, Amogh, Joshua, Ben, Akhil, Hannah, Kelley, Dilan, and Alan

  - Winner of TIS-100 is…

    - Akhil!! 🎉🎉🎮

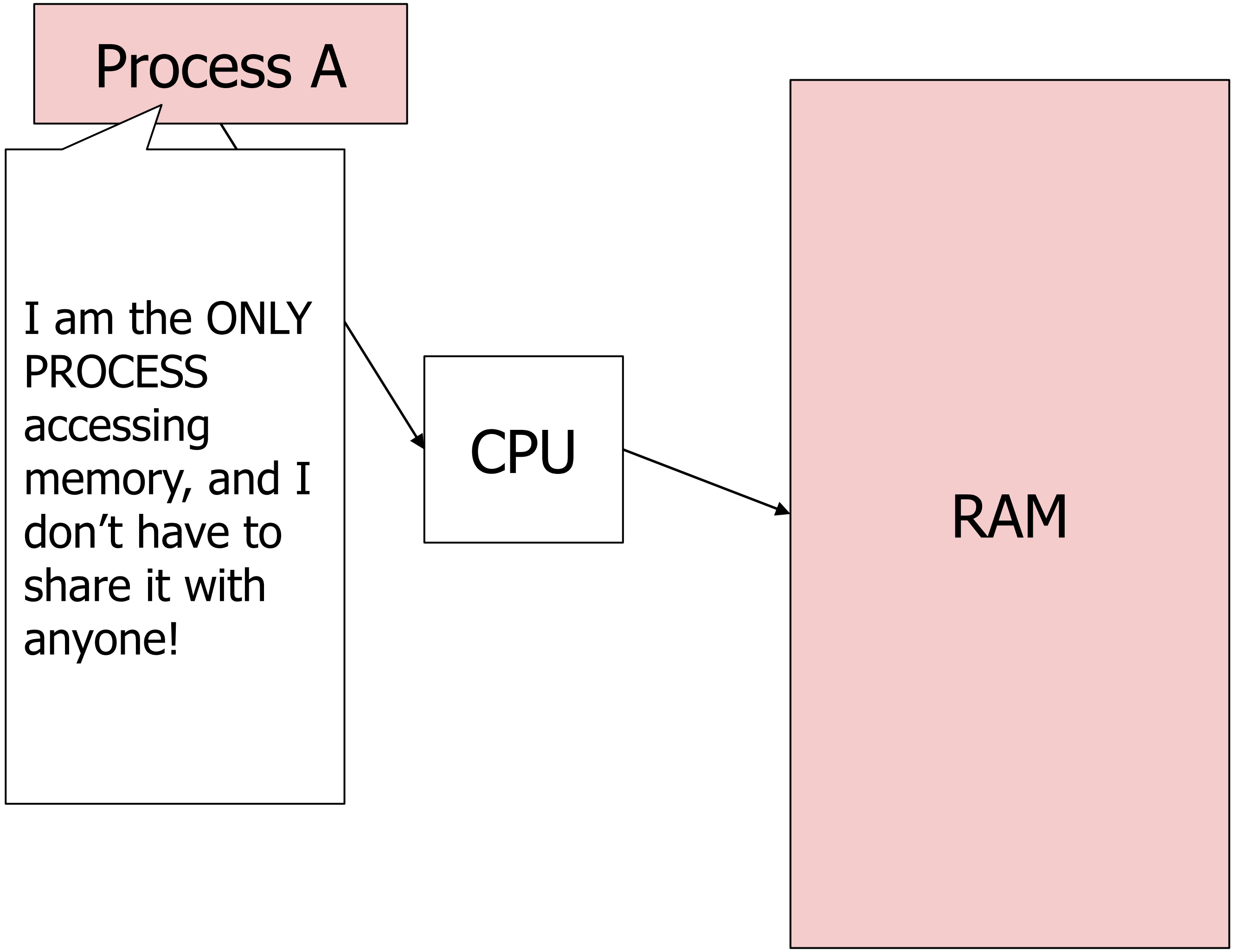
Today’s Goals

- Understand goals and application of virtual memory
- Explore how virtual memory resolves memory problems
- Practice translating virtual addresses to physical addresses

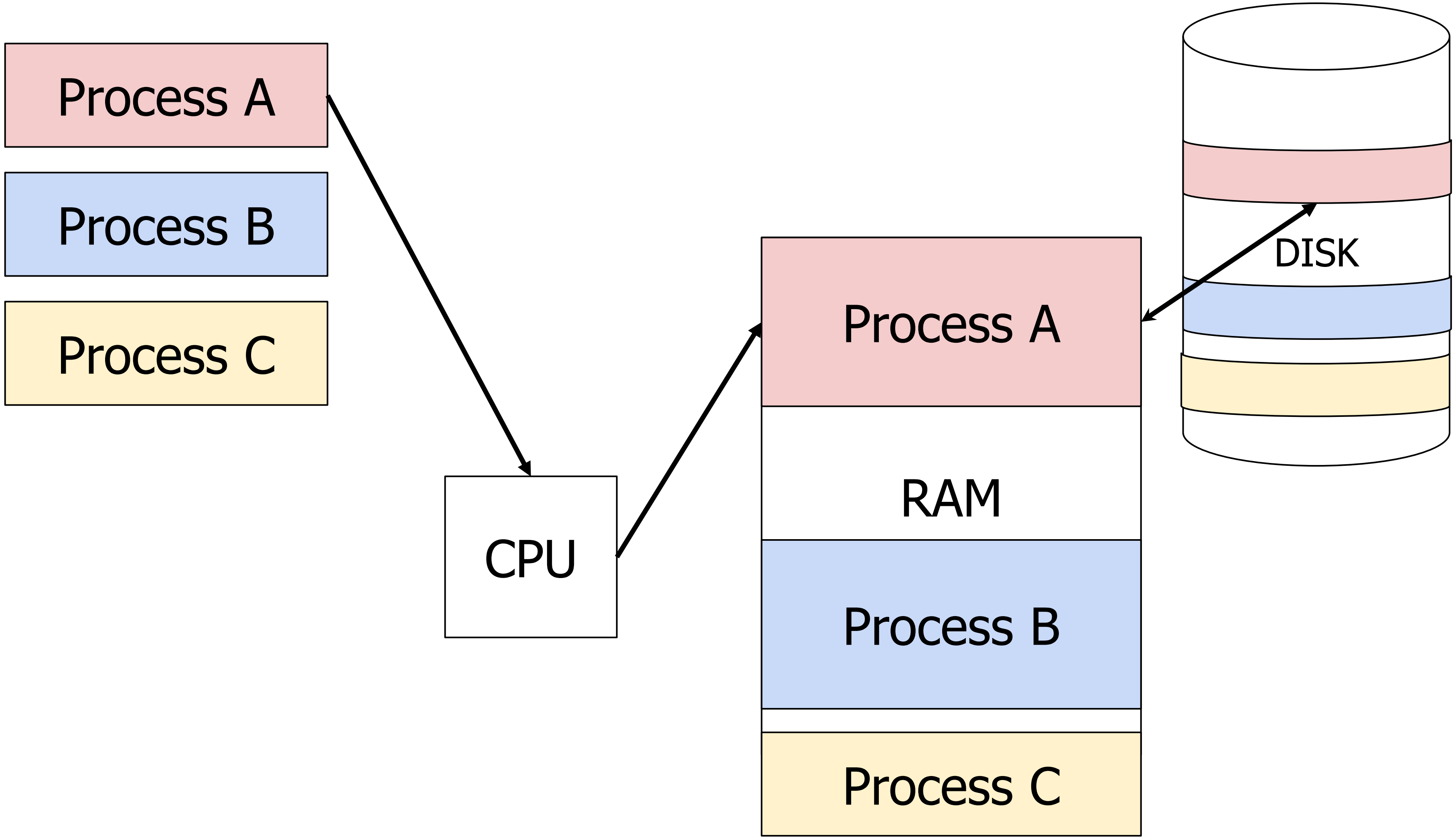
Outline

- **Memory Problems**
- Virtual Memory Concept
- Main Memory as a Cache
- Memory Problems Solved
- Address Translation
- Caching Page Table Entries

The Illusion!



The Reality!



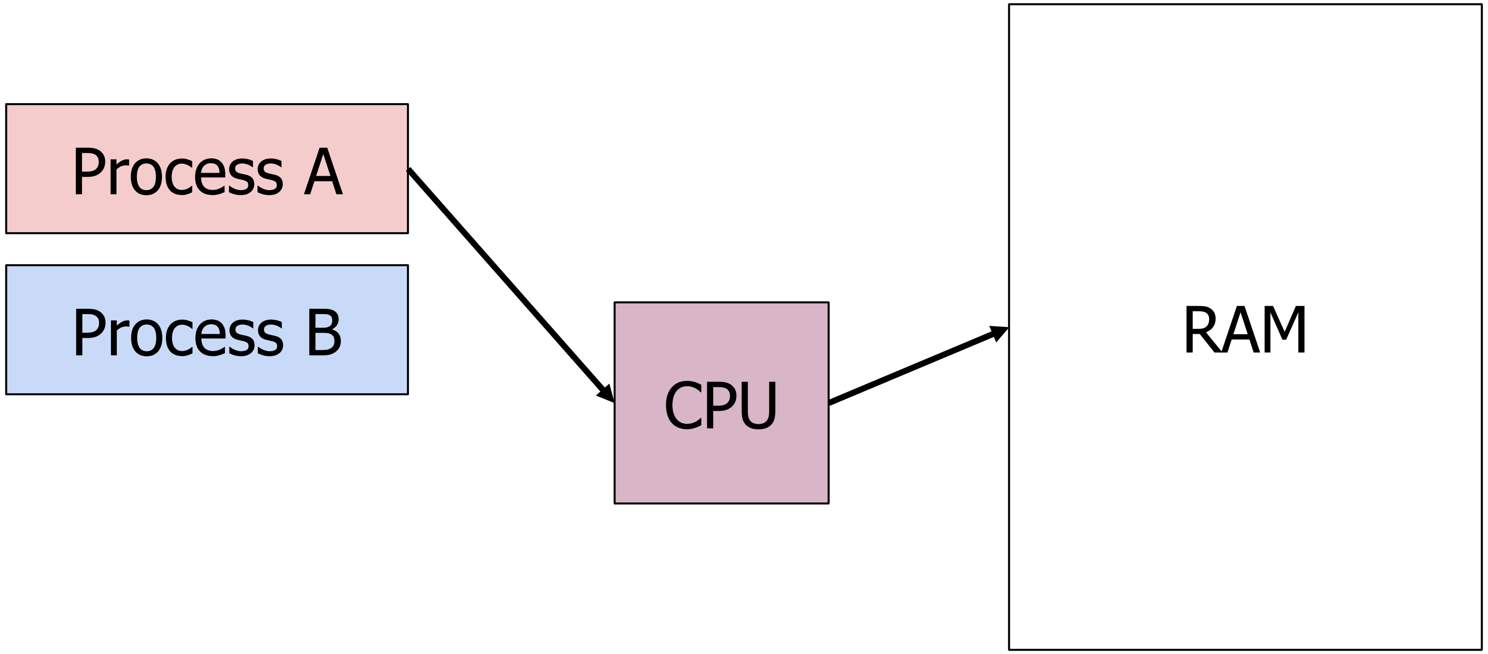
Process A
Process B
Process C
CPU
Process A
RAM
Process B
Process C
DISK

Memory problems

- What are the challenges to supporting this reality?

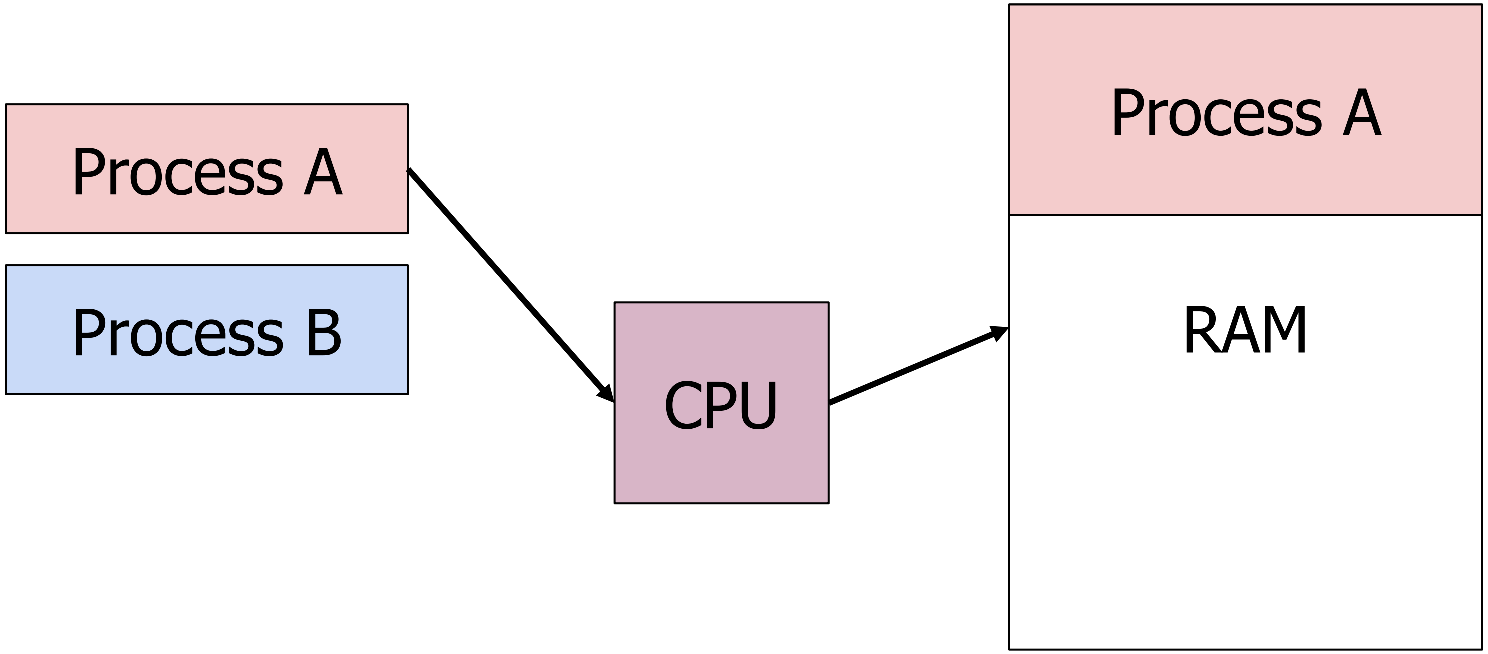
    1. Which addresses does each process get?

Multiple applications share RAM

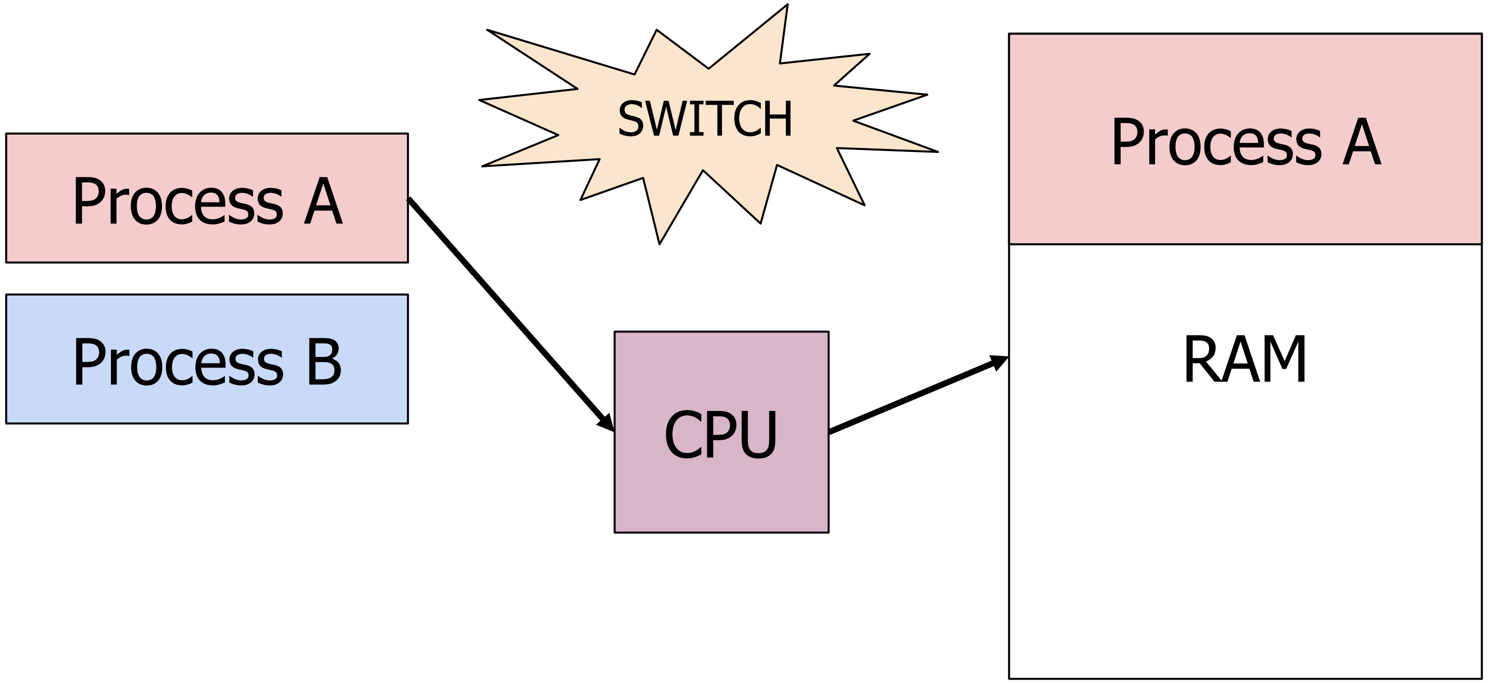


Both processes assume they start at the beginning of RAM and use as much as they need

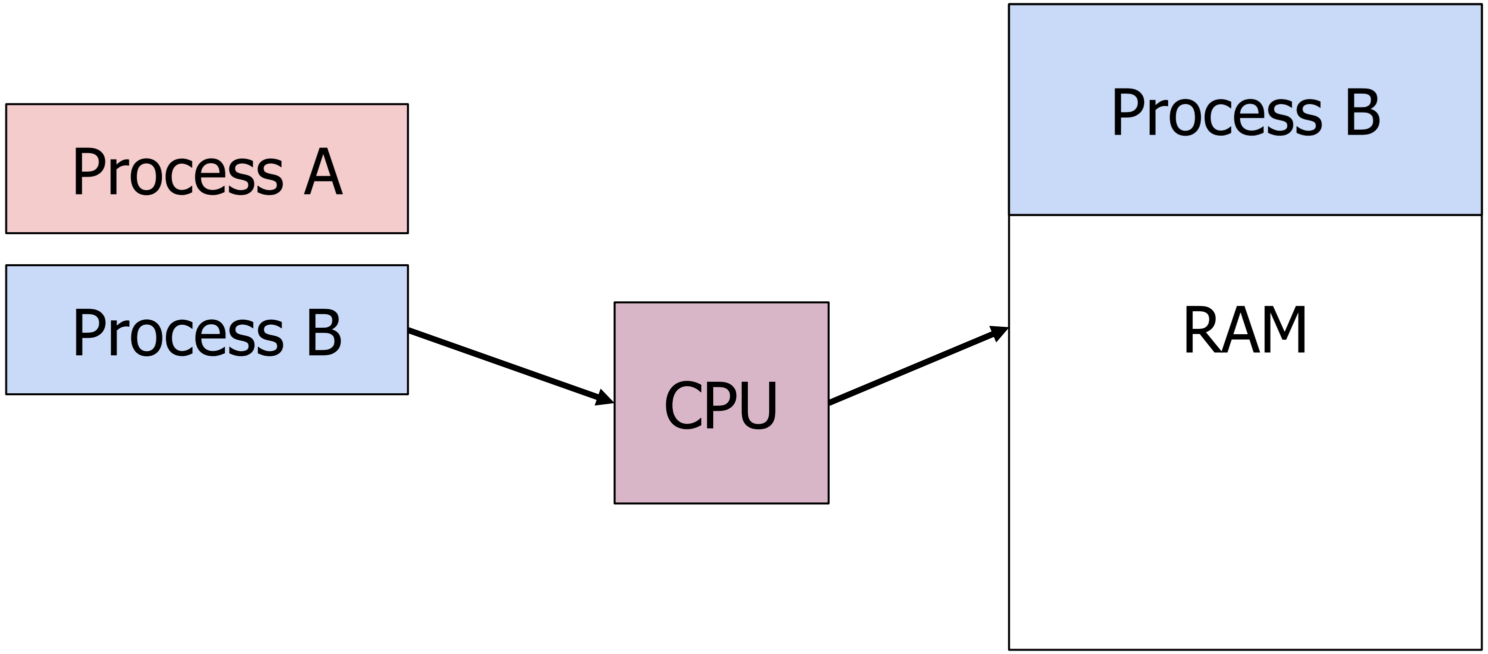
Multiple applications share RAM



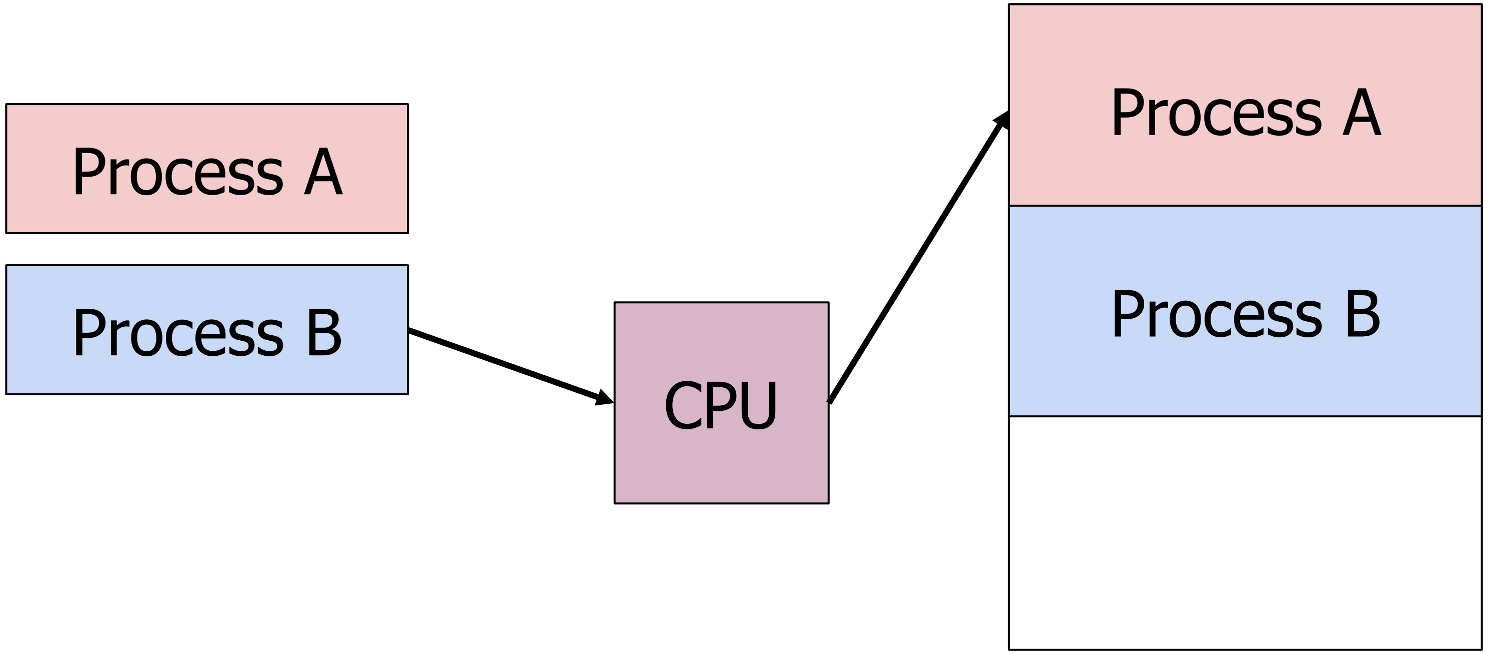
Multiple applications share RAM



Multiple applications share RAM



Multiple applications share RAM



There's enough RAM for both. Why should we have to swap?

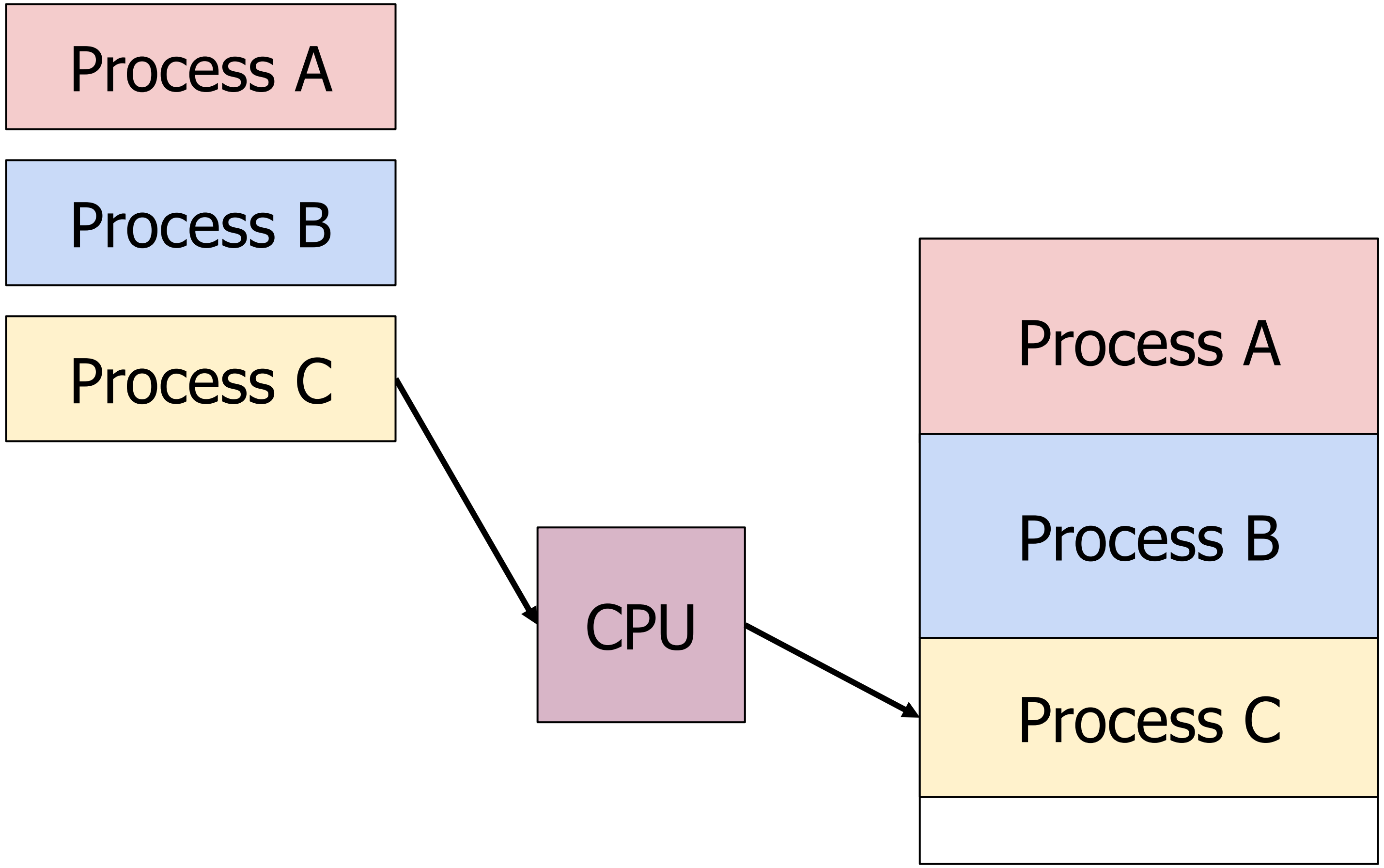
Memory problems

- What are the challenges to supporting this reality?

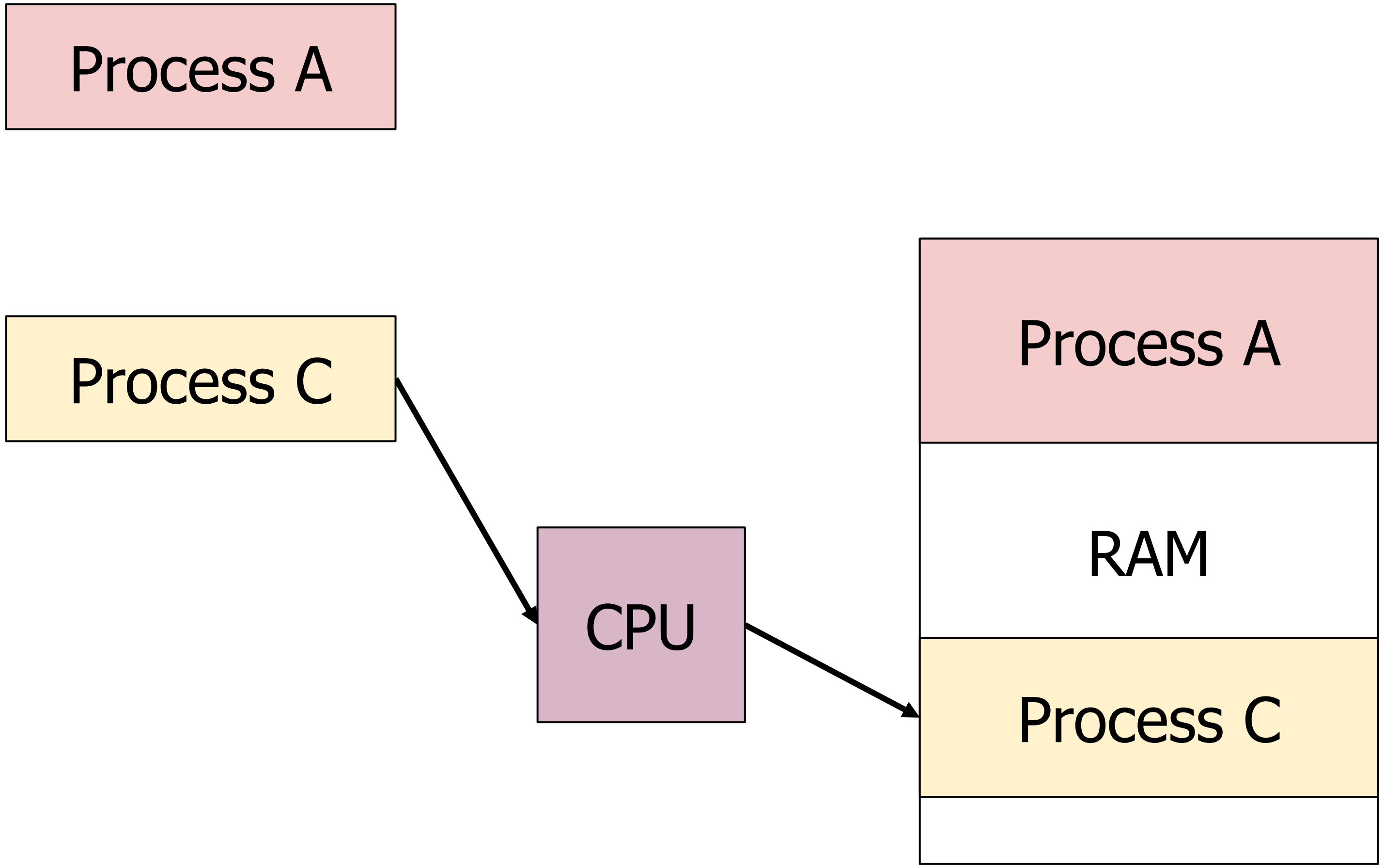
  1. Which addresses does each process get?

  2. How do we move memory around?

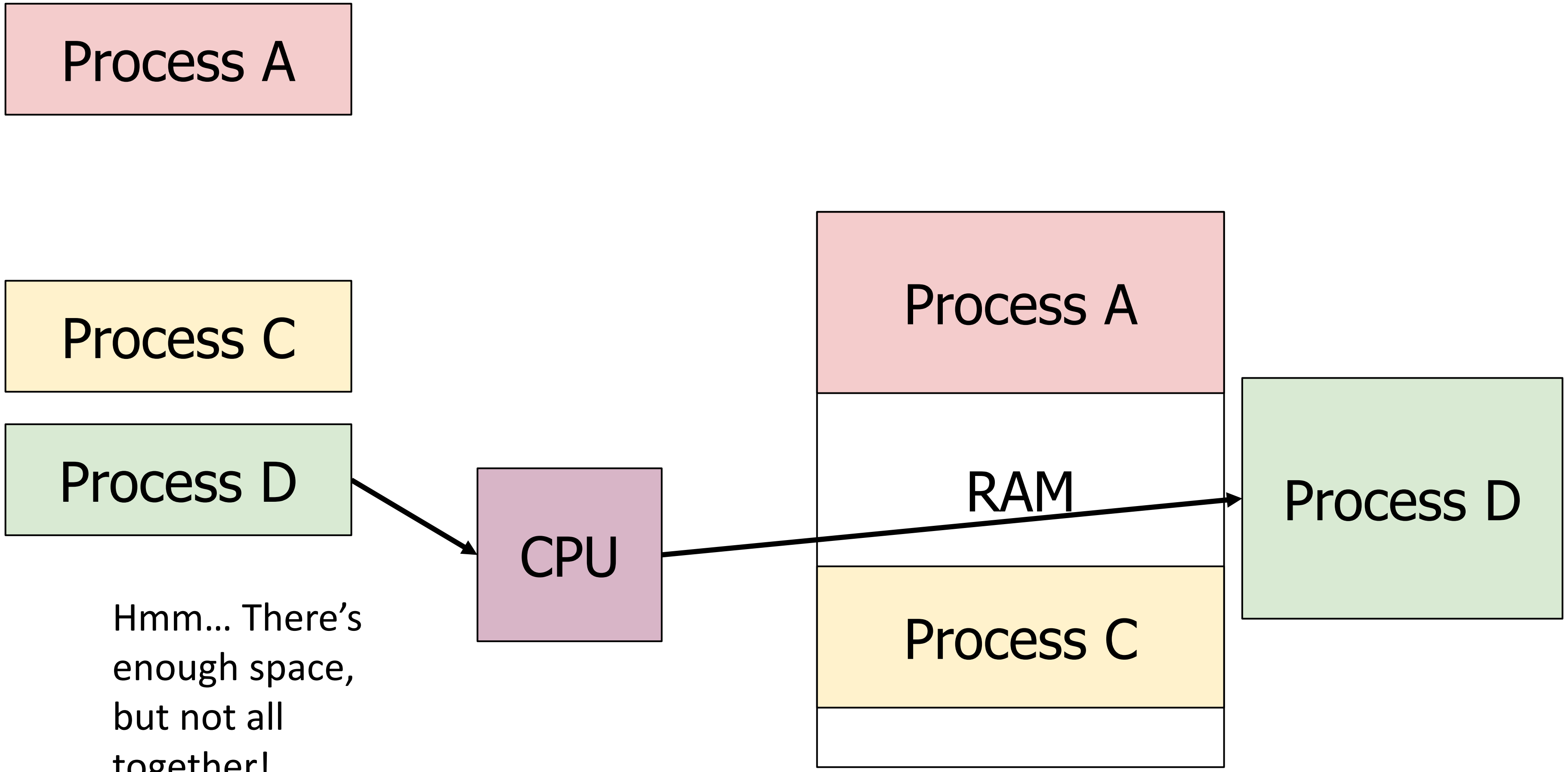
Memory fragmentation



Memory fragmentation

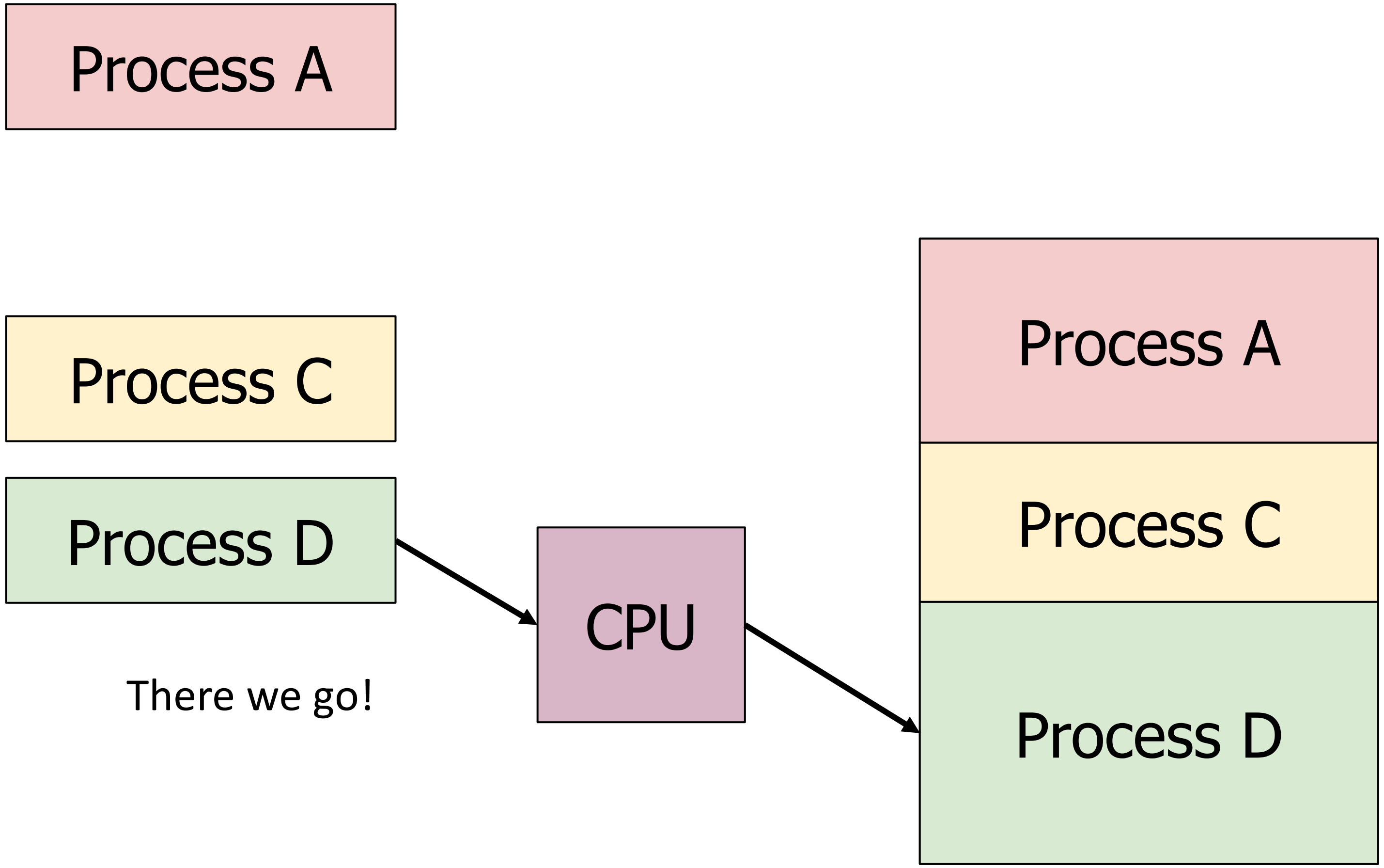


Memory fragmentation



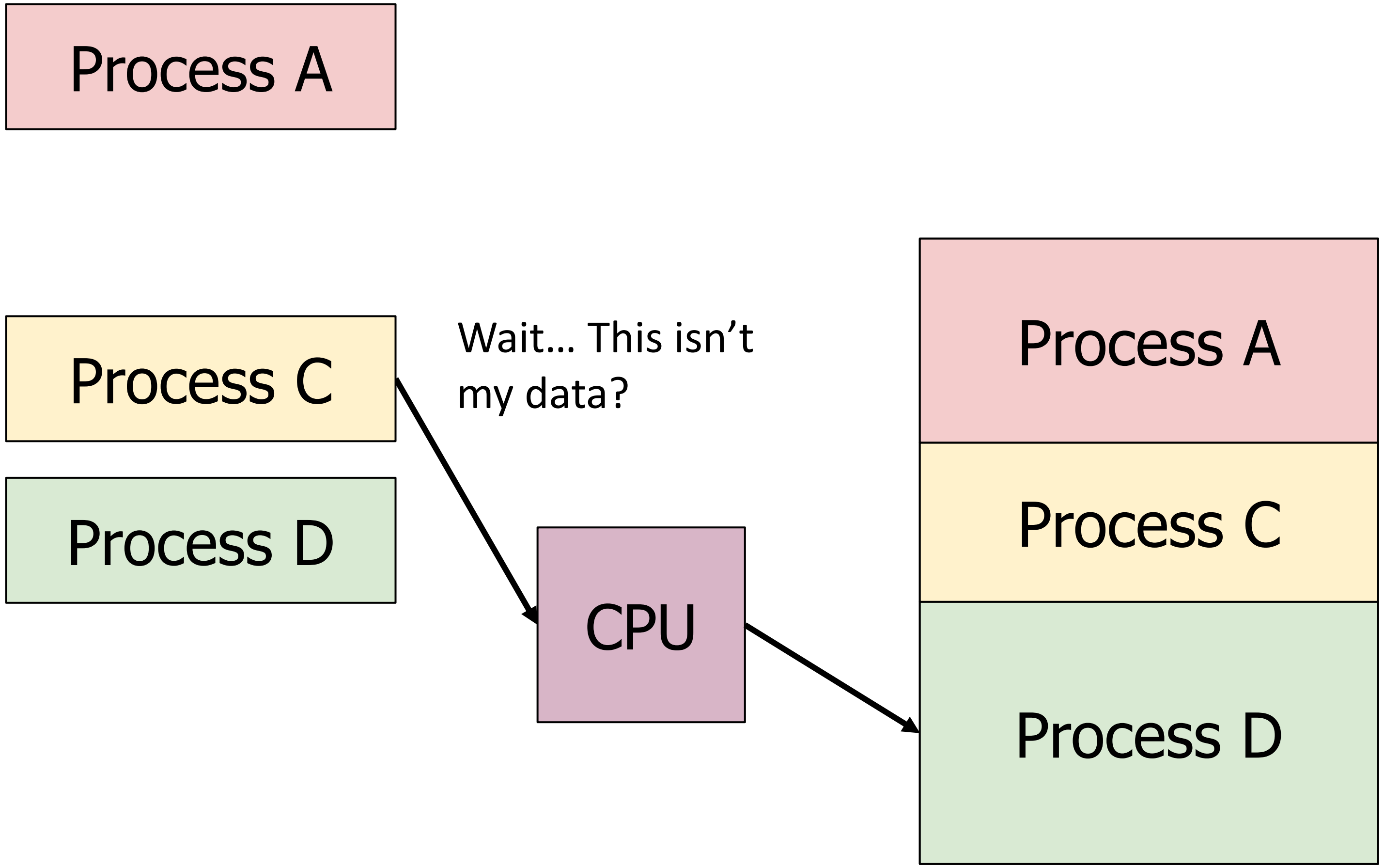
Hmm… There's enough space, but not all together!

Memory fragmentation



There we go!

Memory fragmentation



Memory problems

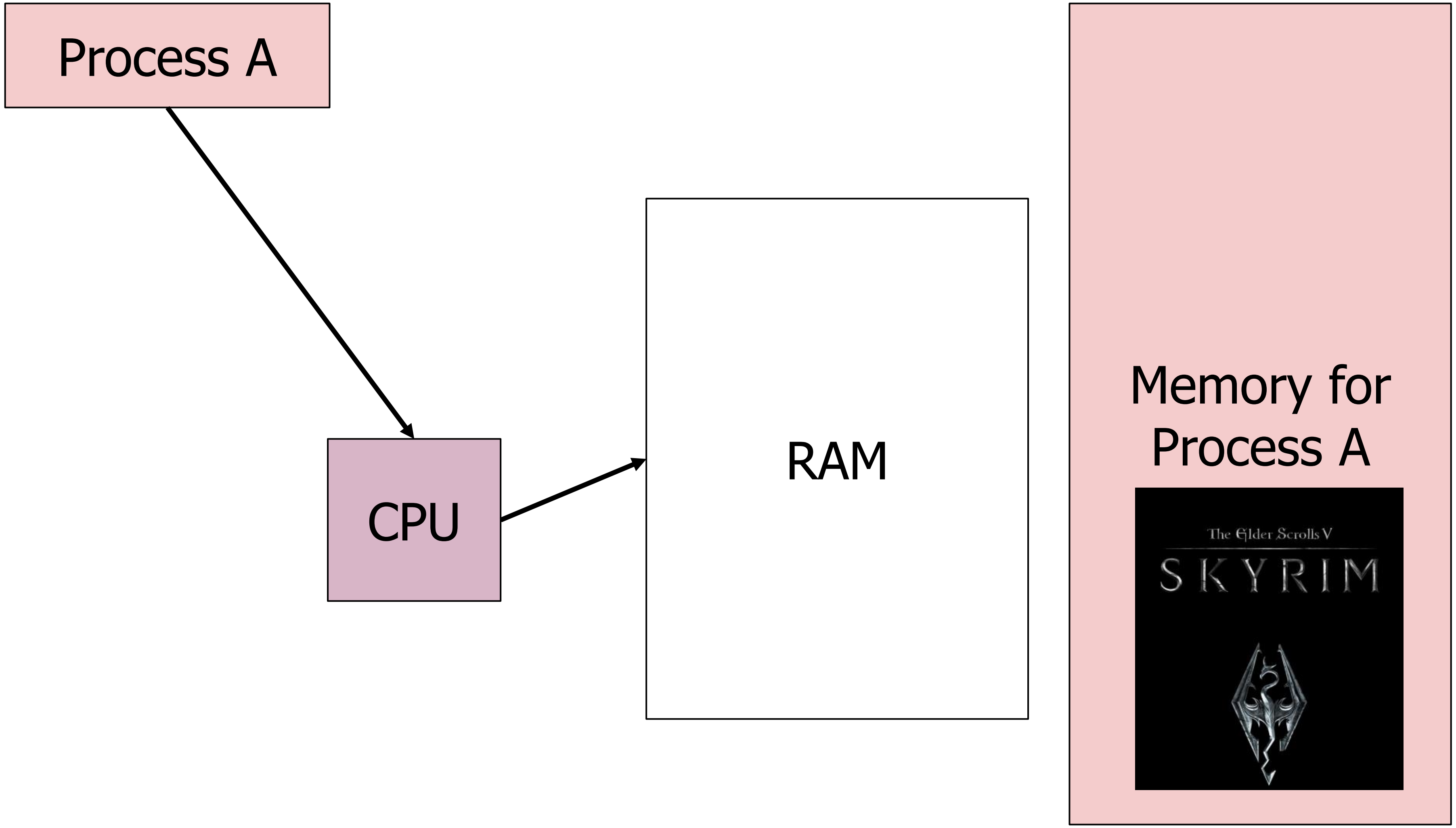
- What are the challenges to supporting this reality?

  1. Which addresses does each process get?

  2. How do we move memory around?

  3. How do we support processes bigger than RAM?

Processes might be bigger than RAM



Memory problems

- What are the challenges to supporting this reality?

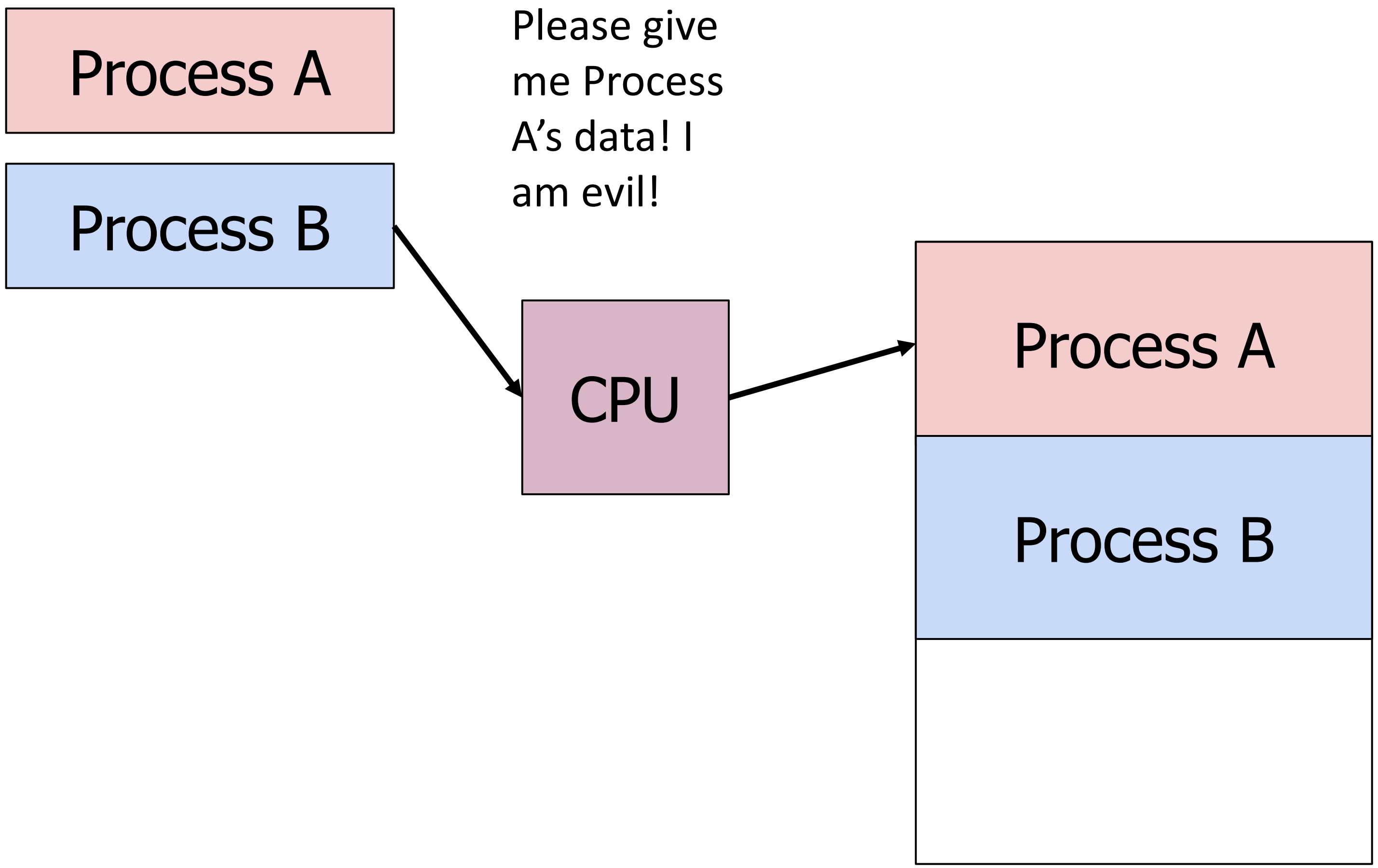
    1. Which addresses does each process get?

    2. How do we move memory around?

    3. How do we support processes bigger than RAM?

    4. How do we protect processes from each other?

Processes can’t be trusted



Memory problems

- What are the challenges to supporting this reality?

  1. Which addresses does each process get?
  2. How do we move memory around?
  3. How do we support processes bigger than RAM?
  4. How do we protect processes from each other?
  5. **How do we deal with how incredibly slow disk is?**

Computing timescales

- Assuming 4 GHz processor, Instruction (with registers): 0.25 ns

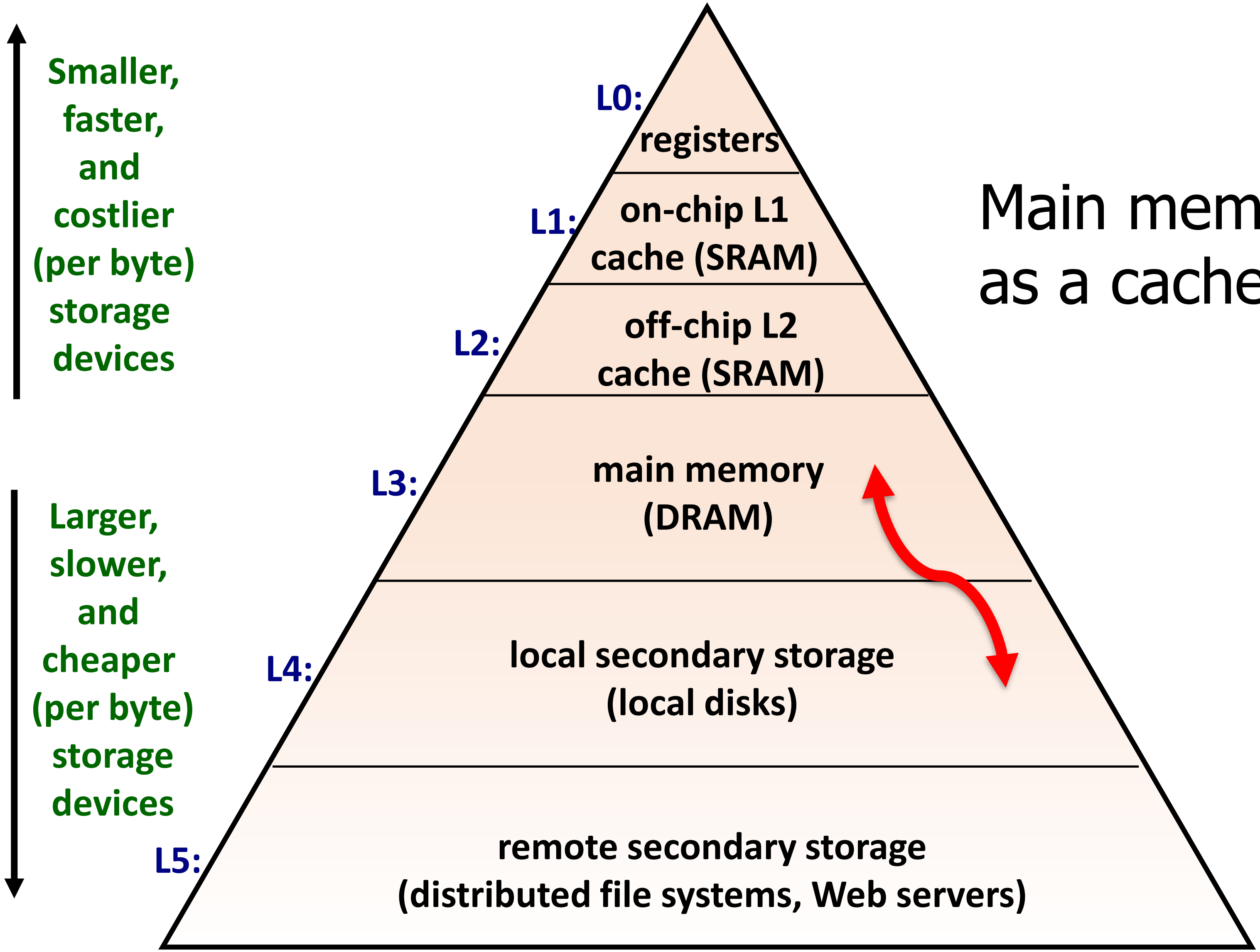
Jeff Dean (Google AI): "Numbers Everyone Should Know"

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Jim Gray's analogy:

- Registers are in your apartment
- Disk is on Mars

Caching disks



Main memory should act as a cache for disk!

Memory problems

- What are the challenges to supporting this reality?
    1. Which addresses does each process get?
    2. How do we move memory around?
    3. How do we support processes bigger than RAM?
    4. How do we protect processes from each other?
    5. **How do we deal with how incredibly slow disk is?**
- Virtual memory addresses all of these problems!

Outline

- Memory Problems
- **Virtual Memory Concept**
- Main Memory as a Cache
- Memory Problems Solved
- Address Translation
- Caching Page Table Entries

Virtual memory concept

- Disconnect reality of RAM from illusion of main memory

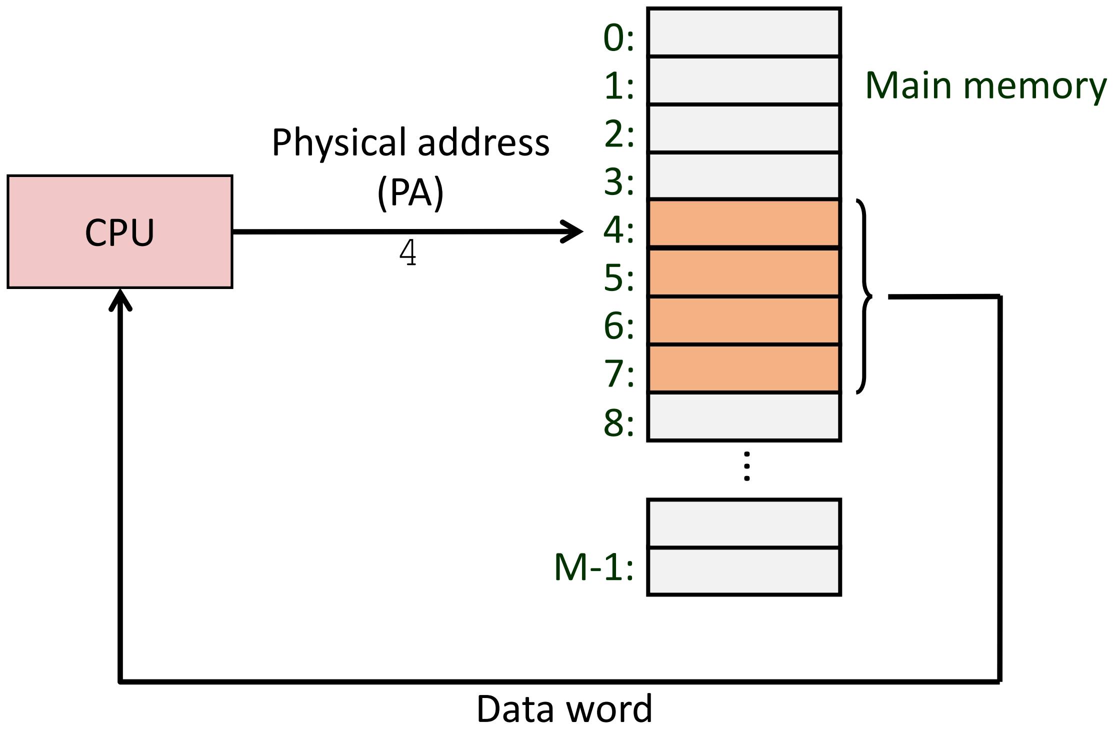
- Processes work with the illusion
  - They use **virtual addresses** to reference where their memory is

- Computer (and OS) work with the reality
  - They use **physical addresses** that are real locations in RAM

- The computer translates virtual addresses into physical addresses

A system using physical addresses

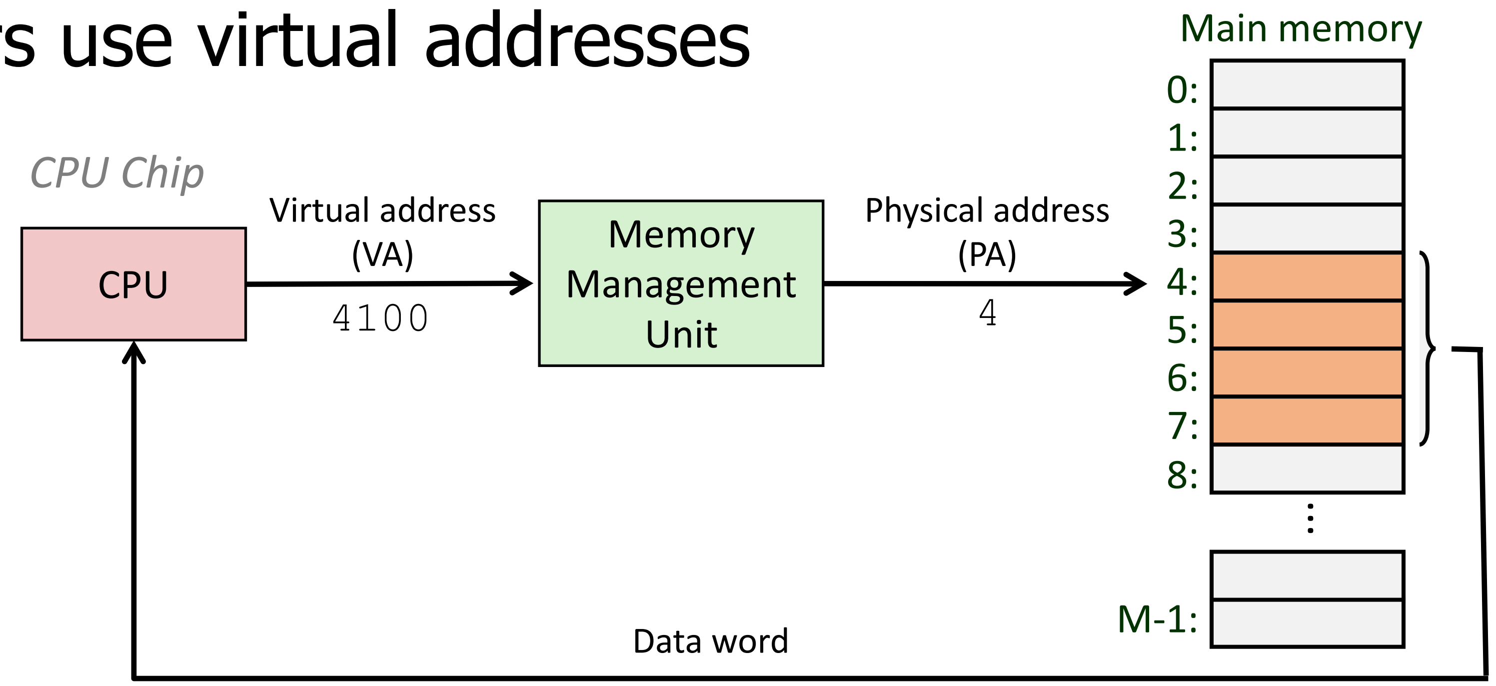
- Main memory - An array of M contiguous byte-sized cells, each with a unique physical address
- Physical addressing
  - Most natural way to access it
    - Addresses used by the CPU correspond to bytes in memory
  - Used in simple systems like early PCs and embedded microcontrollers



A system using virtual addresses

- The CPU generates virtual address
  - Address translation is done by dedicated hardware (memory management unit) via OS-managed lookup table
  - Resulting physical address is looked up in memory (or memory hierarchy)

- Modern processors use virtual addresses
  - All addresses your programs work with are virtual!



Virtual Memory

- From here on out, we'll be working with two different memory spaces:
  - **Virtual Memory (VM)**: A large (~infinite) space that a process believes it, and only it, has access to
  - **Physical Memory (PM)**: The limited RAM space your computer must share among all processors
- This idea is independent of caches
  - Could still be multiple layers of memory caches in the CPU
  - They might use virtual or physical addresses
    - We'll assume caches use physical addresses for this class

Outline

- Memory Problems
- Virtual Memory Concept
- **Main Memory as a Cache**
- Memory Problems Solved
- Address Translation
- Caching Page Table Entries

VM as a Tool for Caching

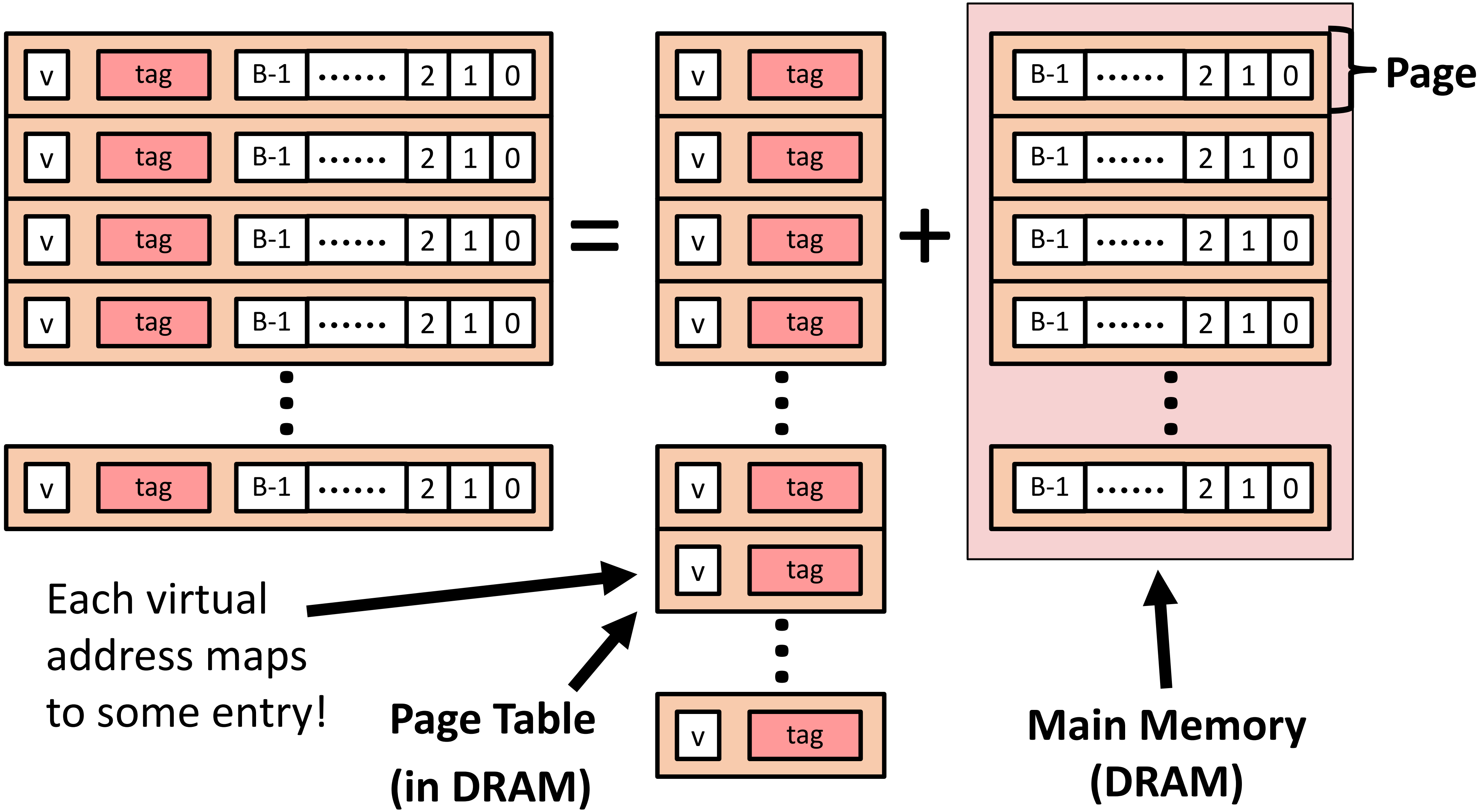
- Physical memory: how much RAM does a computer actually have
  - These days, a few gigabytes is typical
- Address space: 2^{64} possible addresses on x86-64!
  - That's 2^{64} possible bytes of memory, or ***17,179,869,184*** GB!
  - (In reality, architecture limits addresses to 48 bits, soon 57. Still huge!)
- Across all your programs, may need more data than fits in physical memory

- Solution: use physical memory as a *cache*! (called: DRAM cache)
  - Store the bulk of your data on disk (very large, very cheap, but very slow)
  - And store the currently-used data in main memory (very fast by comparison)
  - Get the best of both worlds! Large capacity and fast access!
- DRAM cache organization driven by the *enormous* miss penalty
  - DRAM is about ***10x*** slower than SRAM
  - Disk is about ***100,000x*** slower than DRAM

Picking Cache Design Parameters

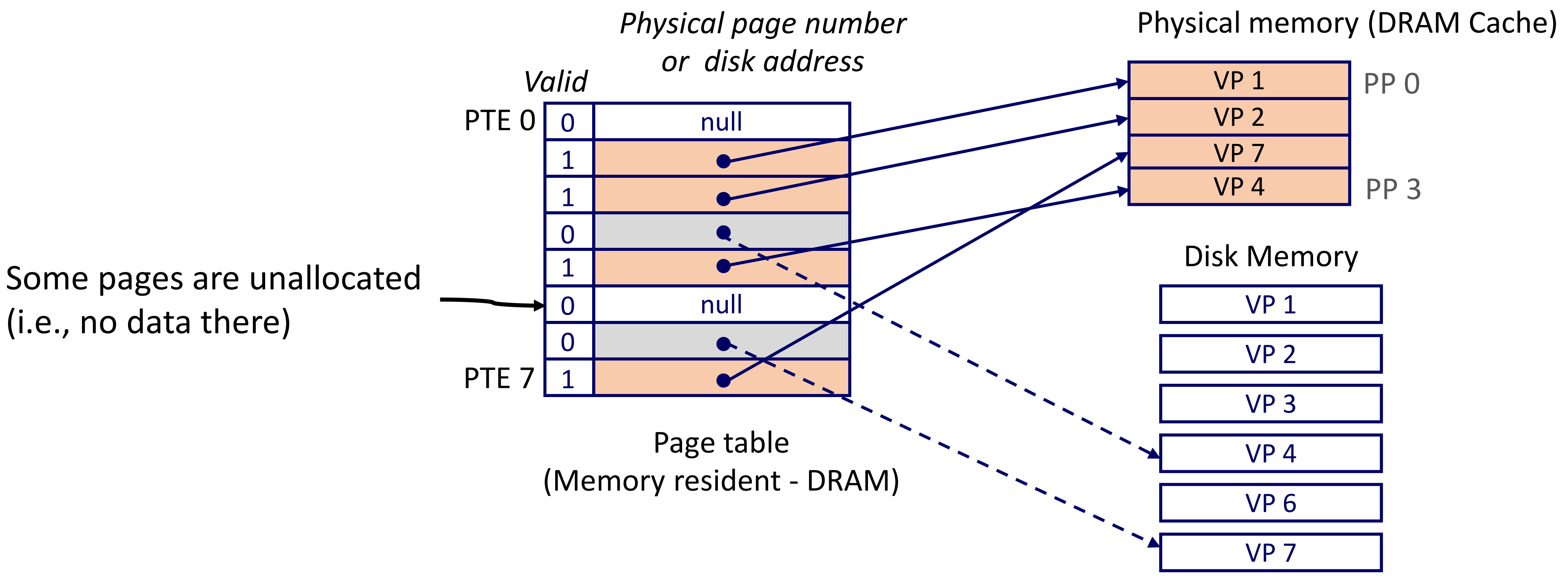
- Block size?
  - Disks are better at transferring large chunks of data (the first byte incurs a long delay, the rest come really fast afterwards)
  - Large block size: typically 4-8 KB
- Associativity?
  - DRAM cache misses incur enormous penalties; have to go to disk. Yikes!
  - Associativity is high to minimize miss rate
    - Fully associative (one huge set): any block can go anywhere in cache
    - Requires a “large” mapping function – but managed in software, so ok
- Write-back or write-through?
  - Disk cannot keep up with a firehose of small writes
  - Use write-back
- Replacement algorithms?
  - Not limited by hardware; hardware strongly favors simple methods
  - Highly sophisticated, expensive, open-ended replacement algorithms

DRAM Cache Analogy to Cache Memory



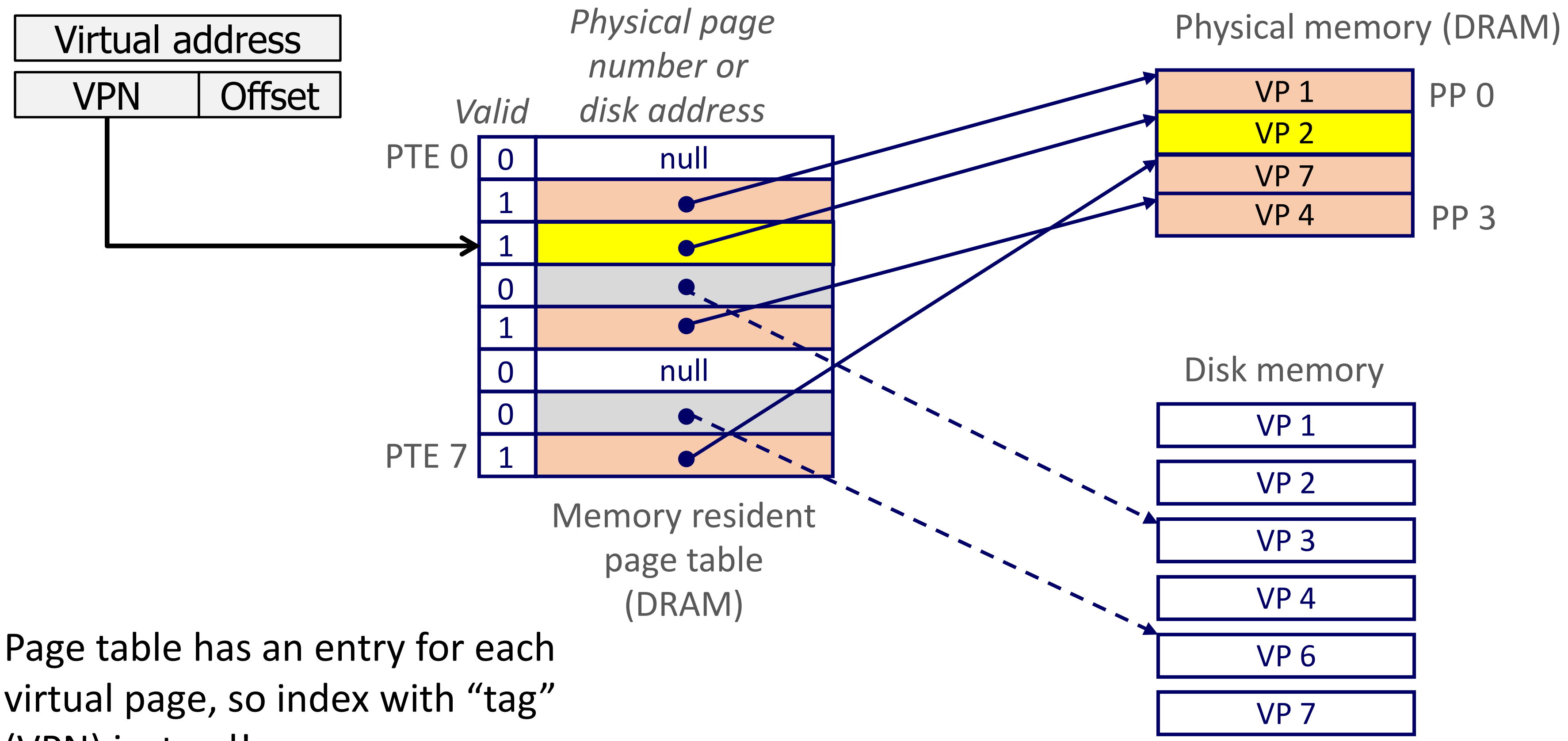
Locating an object in DRAM Cache: Page Tables

- A **page table** maps virtual pages to physical pages
  - One page table entry (PTE) per virtual page (possible page in VM)
    - Fully-associative → one big set. Use the "tag" to index into table!
  - Each PTE specifies either a physical page (in DRAM) or a disk address
    - Valid bit tells us which
    - Instead of data in the "cache block", "pointer" to where data is



Page Hit

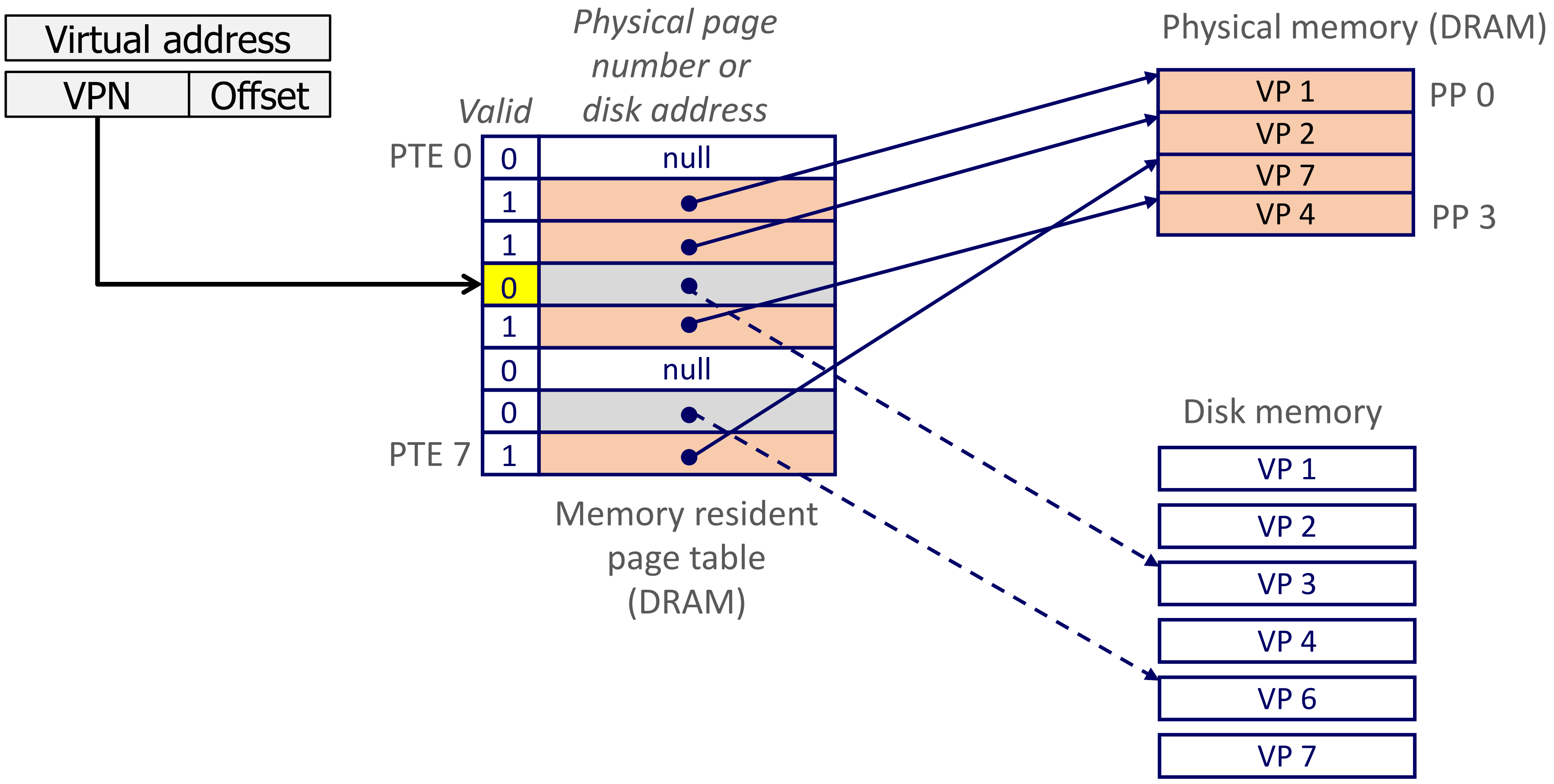
- *Page hit:* reference to a VM word that is in physical memory (hit on the DRAM cache)



Page table has an entry for each virtual page, so index with “tag” (VPN) instead!

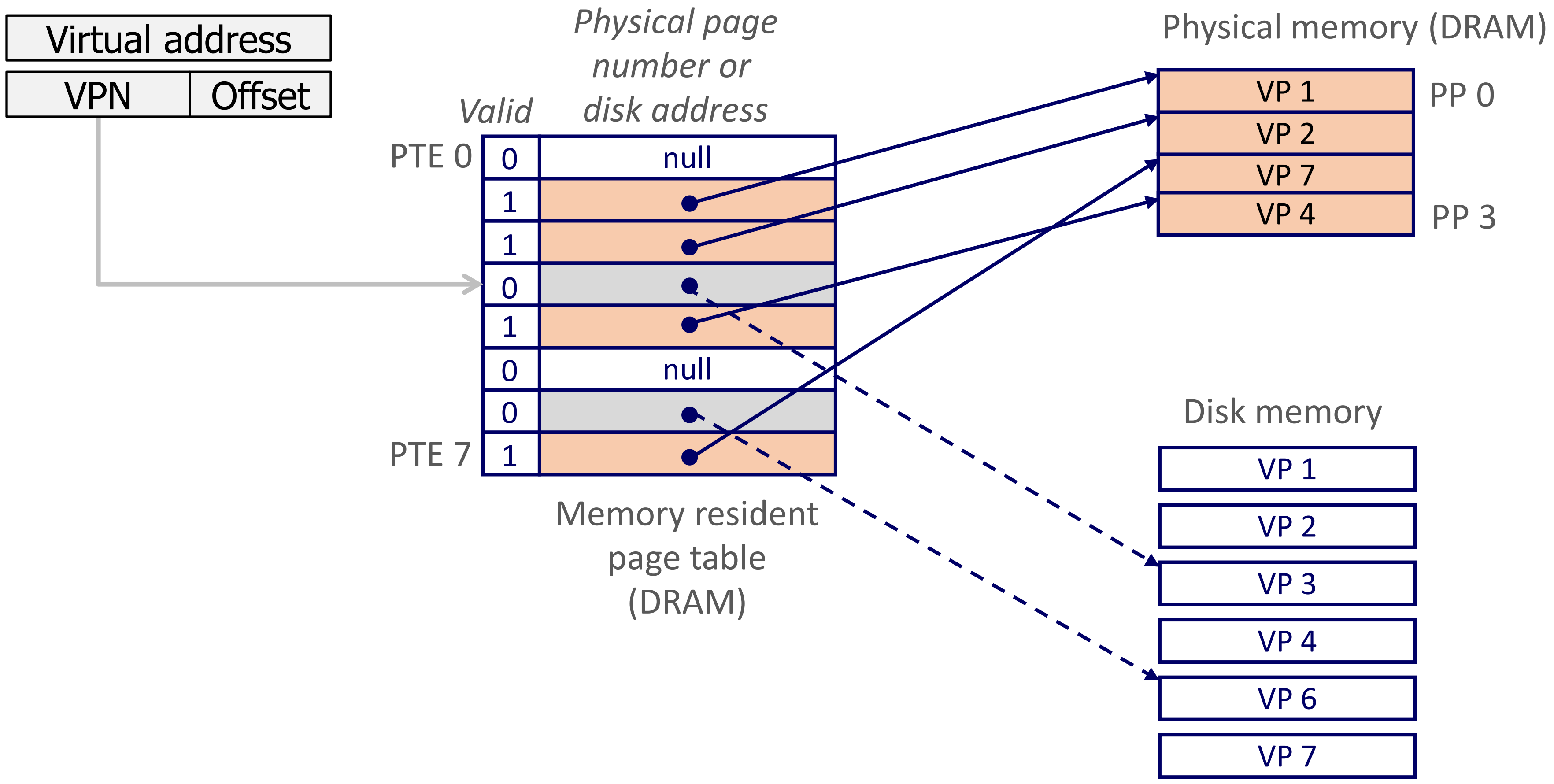
Page Fault

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



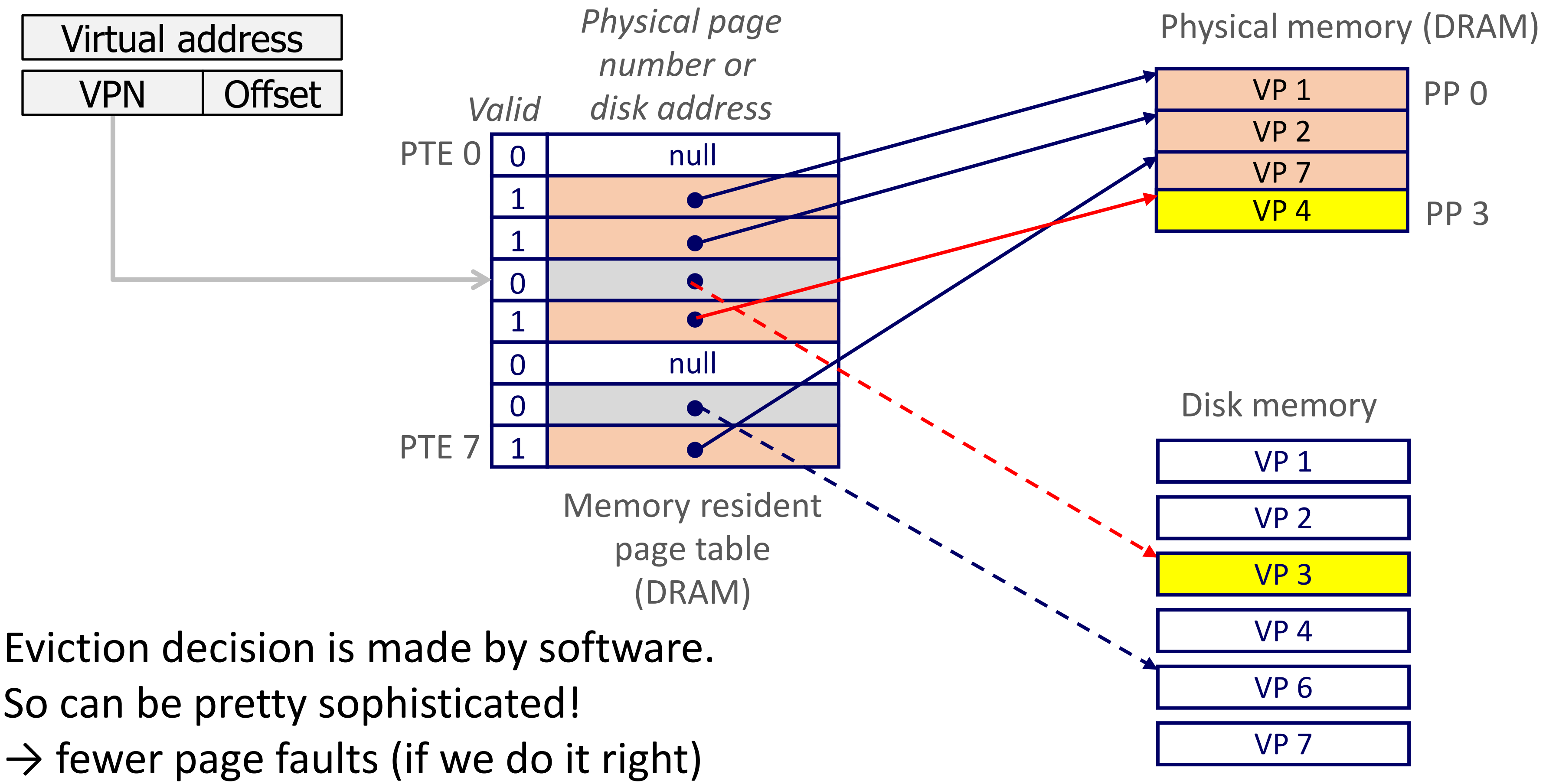
Handling Page Fault

- Page miss causes page fault (a HW exception, OS code kicks in to handle)



Handling Page Fault

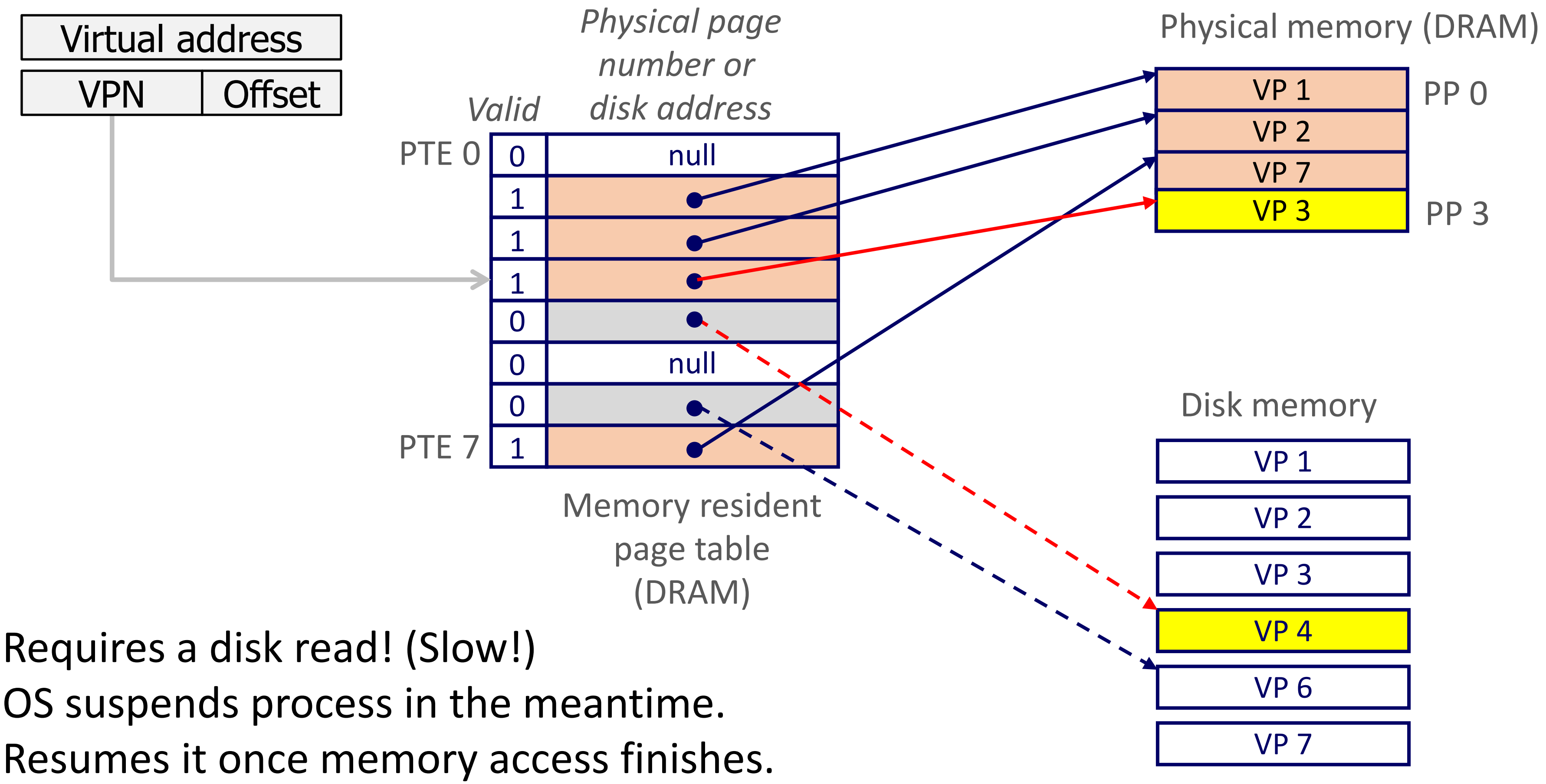
- Page miss causes page fault (a HW exception, OS code kicks in to handle)
- Page fault handler selects a victim to be evicted (here VP 4)



Eviction decision is made by software.
So can be pretty sophisticated!
→ fewer page faults (if we do it right)

Handling Page Fault

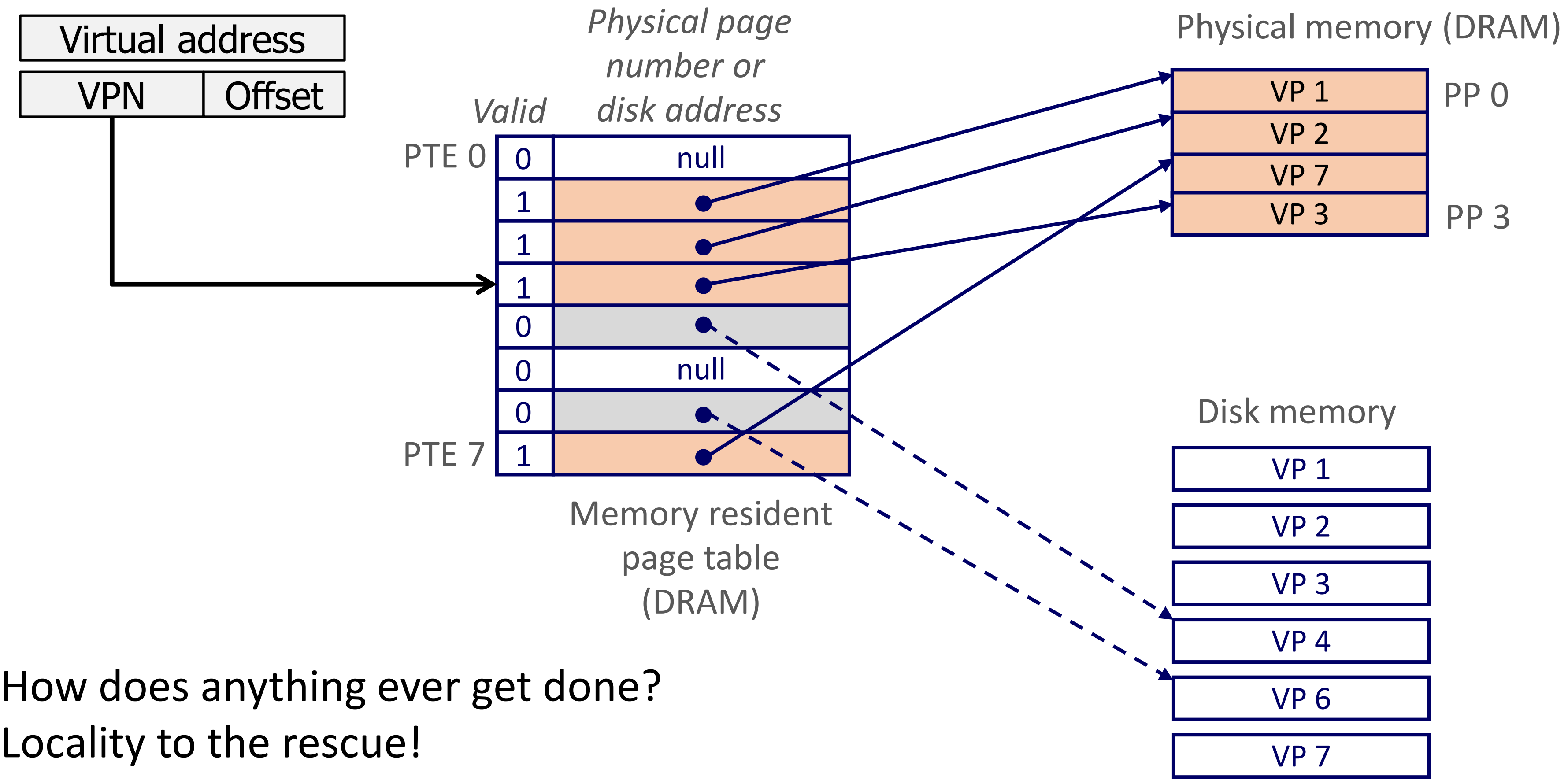
- Page miss causes page fault (a HW exception, OS code kicks in to handle)
- Page fault handler selects a victim to be evicted (here VP 4)
- The victim page is swapped with the disk block of the requested address



Requires a disk read! (Slow!)
OS suspends process in the meantime.
Resumes it once memory access finishes.

Handling Page Fault

- Offending instruction is restarted: page hit this time!



How does anything ever get done?
Locality to the rescue!

Locality saves the day (as usual)

- At any point in time, programs tend to access a small set of active virtual pages called the **working set**
  - Programs with higher temporal locality will have smaller working sets

- If (working set size < main memory size)
  - High performance for one process after compulsory misses
  - Fully-associative cache, so no conflicts. Only capacity matters.
  - Life is good!

- If (SUM(working set sizes) > main memory size)
  - **Thrashing**: Performance meltdown where pages are swapped to and from disk continuously

  - When cache memory is thrashing, CPU runs at the speed of memory. Ow.
  - When virtual memory is thrashing, CPU runs at the speed of disk. Yikes!
    - Hope you enjoy the commute to Pluto
    - Because that's where your data is

Break + Question

- Computer has:
  - 8 pages of Virtual Memory
  - 4 pages of Physical Memory

- How many entries (rows) does a page table have?
- How many entries can be valid at any time?

Break + Question

- Computer has:
  - 8 pages of Virtual Memory
  - 4 pages of Physical Memory

- How many entries (rows) does a page table have? **8 entries**
- How many entries can be valid at any time? **4 valid**

- Page Table translates Virtual to Physical
  - It needs an entry for each virtual page, so 8 entries

  - Rows are valid if they point at physical memory
    - So only four entries can be valid (unless they share a physical page)

Outline

- Memory Problems
- Virtual Memory Concept
- Main Memory as a Cache
- **Memory Problems Solved**
- Address Translation
- Caching Page Table Entries

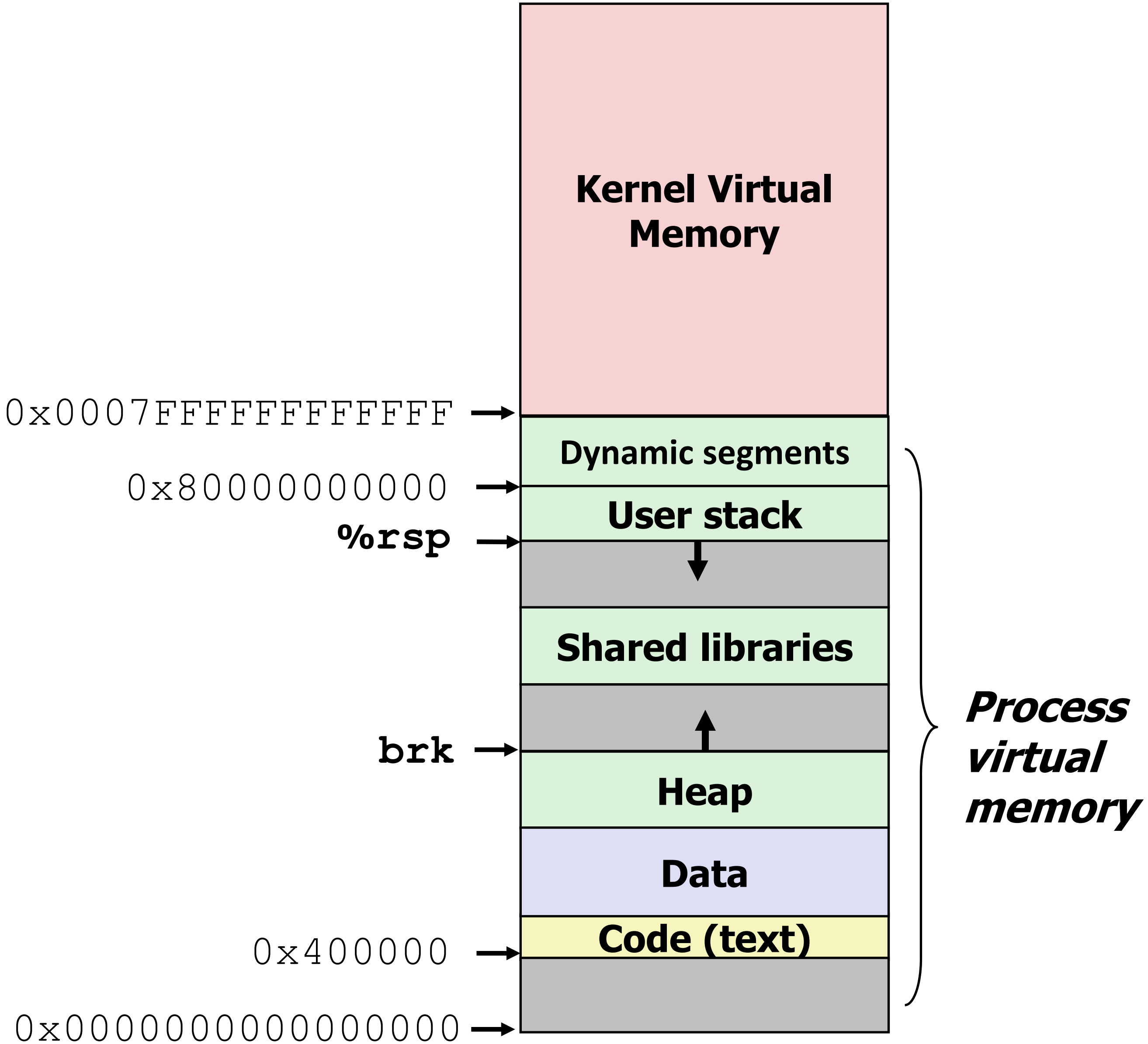
Memory problems

- What are the challenges to supporting this reality?

1. Which addresses does each process get?
2. How do we move memory around?
3. How do we support processes bigger than RAM?
4. How do we protect processes from each other?
5. How do we deal with how incredibly slow disk is? ✔
   Use RAM as a cache for disk

Which addresses do processes get?

- Programs can use whatever virtual addresses they want
- OS controls physical addresses



Memory problems

- What are the challenges to supporting this reality?

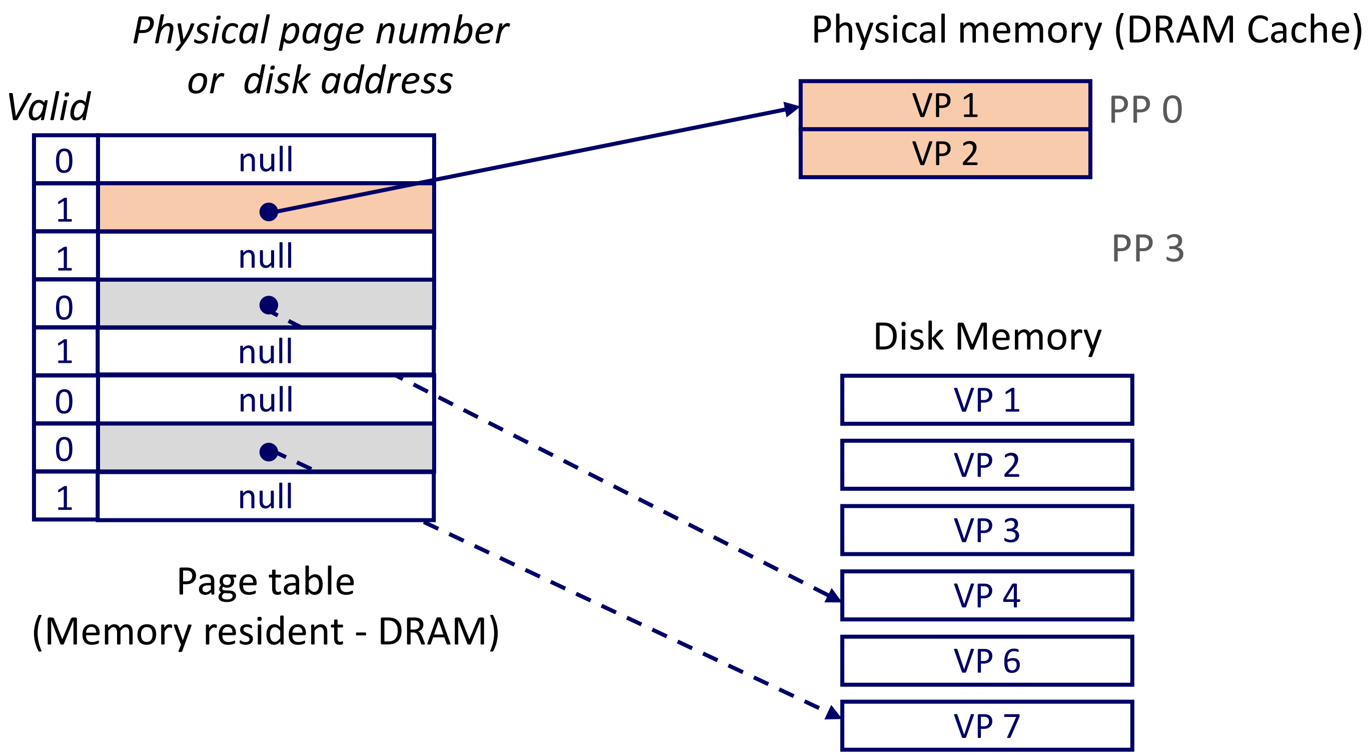
1. Which addresses does each process get? ✔
   Whatever virtual addresses they want
2. How do we move memory around?

3. How do we support processes bigger than RAM? ✔
   Leave some pages on disk
4. How do we protect processes from each other?

5. How do we deal with how incredibly slow disk is? ✔
   Use RAM as a cache for disk

How do we move memory around?

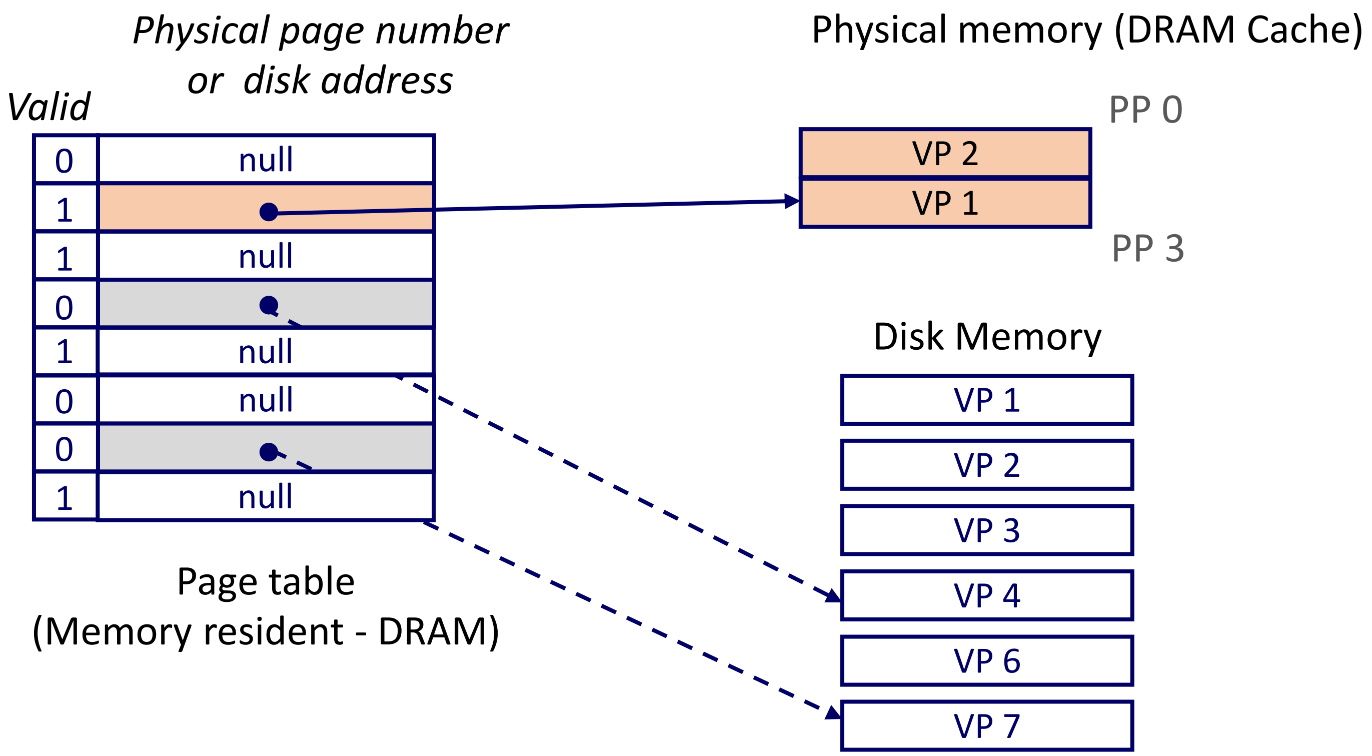
- Just change the page table entry!



How do we move memory around?

- Just change the page table entry!
    - Same virtual address points at a different physical address

- Usually only happens when pages are swapped to disk and then later brought back



Memory problems

- What are the challenges to supporting this reality?

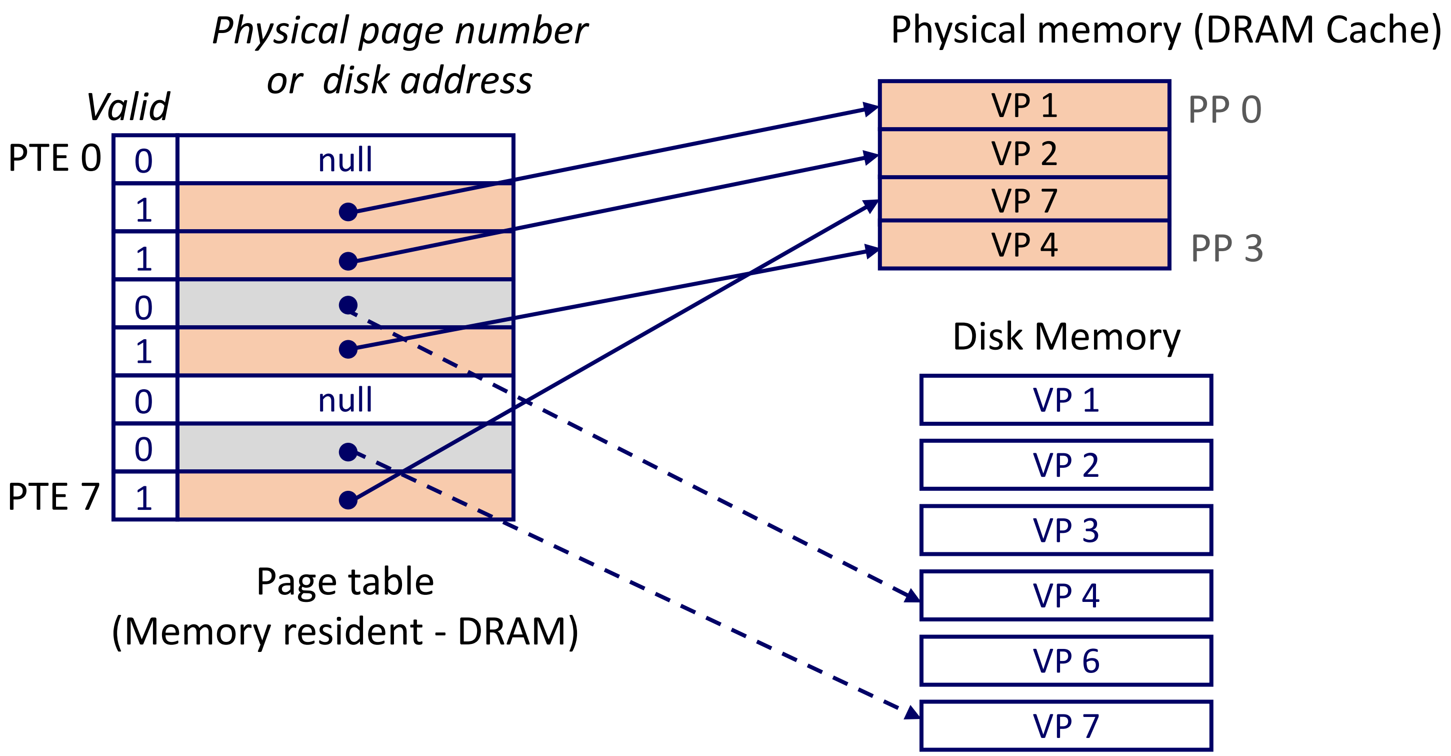
1. Which addresses does each process get? ✔
   Whatever virtual addresses they want
2. How do we move memory around? ✔
   Update page table entries
3. How do we support processes bigger than RAM?

4. How do we protect processes from each other?

5. How do we deal with how incredibly slow disk is? ✔
   Use RAM as a cache for disk

How do we support processes bigger than RAM?

- Just leave some pages for that process on disk
- Page table entry still exists for each virtual page
- Hopefully working set is smaller than program memory



Memory problems

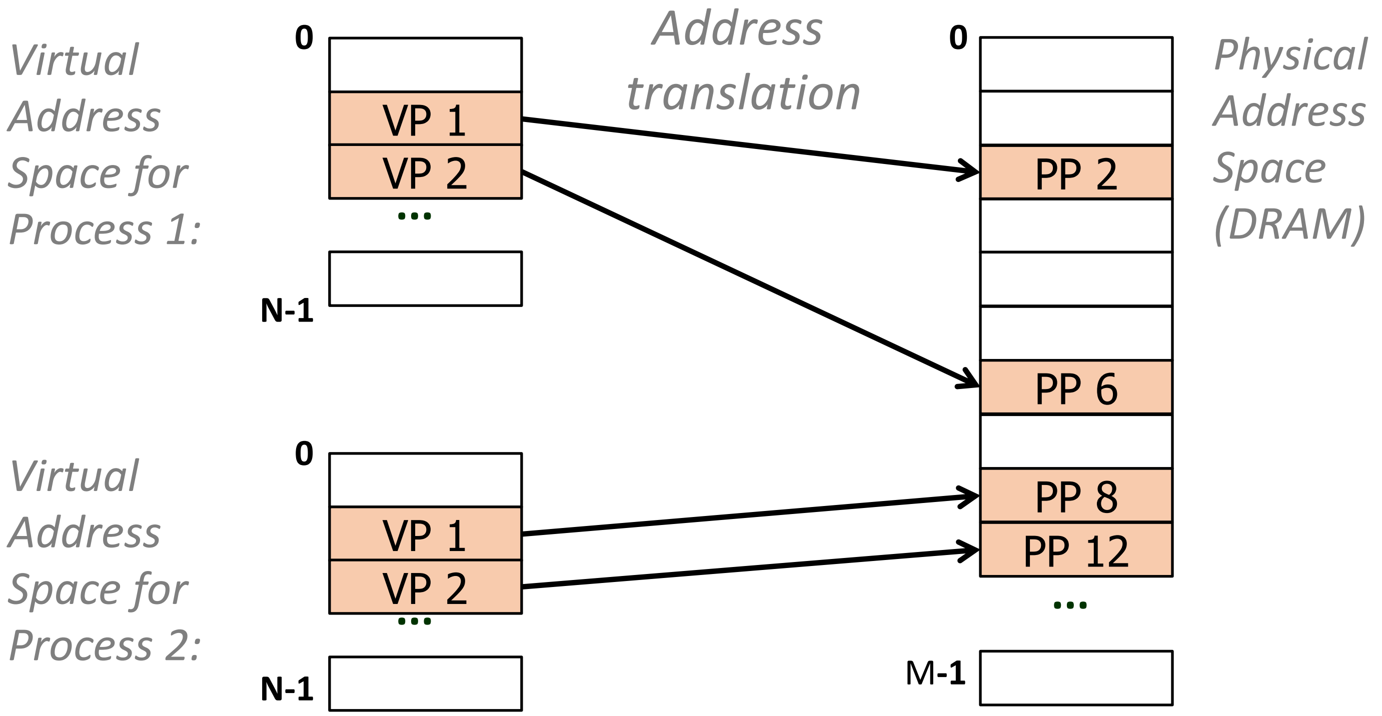
- What are the challenges to supporting this reality?

1. Which addresses does each process get? ✔
   Whatever virtual addresses they want
2. How do we move memory around? ✔
   Update page table entries
3. How do we support processes bigger than RAM? ✔
   Leave some pages on disk
4. How do we protect processes from each other?

5. How do we deal with how incredibly slow disk is? ✔
   Use RAM as a cache for disk

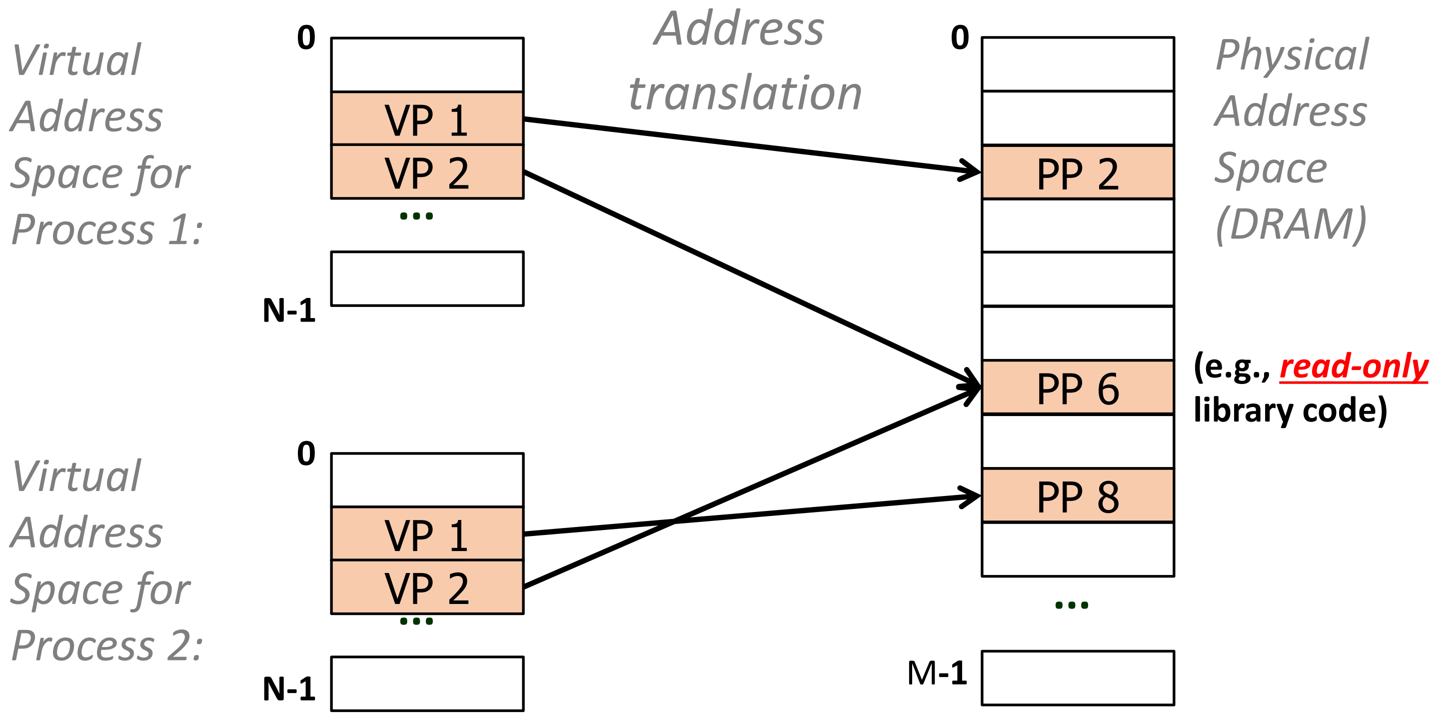
How do we protect processes from each other?

- Each process has separate virtual memory spaces
  - No way to access another process's physical memory unless it is mapped to one of your virtual addresses



Enabling shared libraries

- We could share some physical pages across processes to enable shared libraries or shared memory



VM as a Tool for Memory Protection

- What if we want better protection?
  - Mark a page as read-only
  - Keep a page in memory, but only the OS can touch it
- Extend PTEs with permission bits!
  - Page fault handler checks these before remapping
- HW enforces this protection (trap into OS if violation occurs)

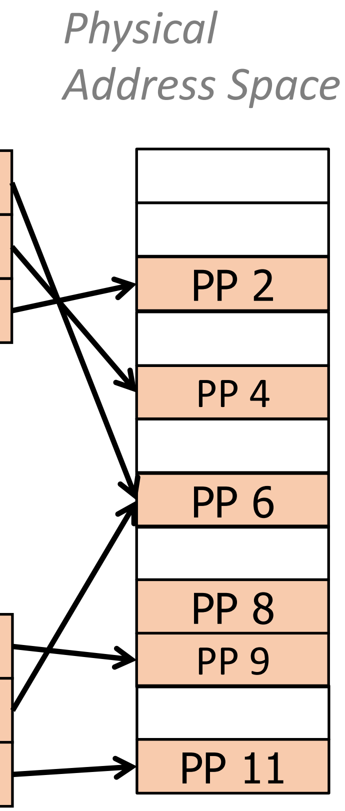
Must be running in kernel (supervisor mode)

Process i

	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	Yes	PP 6
VP 1:	No	Yes	Yes	No	PP 4
VP 2:	Yes	Yes	Yes	No	PP 2

Process j

	SUP	READ	WRITE	EXEC	Address
VP 0:	No	Yes	No	No	PP 9
VP 1:	Yes	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	No	PP 11



Memory problems

- What are the challenges to supporting this reality?

1. Which addresses does each process get? ✔
   Whatever virtual addresses they want
2. How do we move memory around? ✔
   Update page table entries
3. How do we support processes bigger than RAM? ✔
   Leave some pages on disk
4. How do we protect processes from each other? ✔
   Don't overlap virtual address spaces + permission bits
5. How do we deal with how incredibly slow disk is? ✔
   Use RAM as a cache for disk

Outline

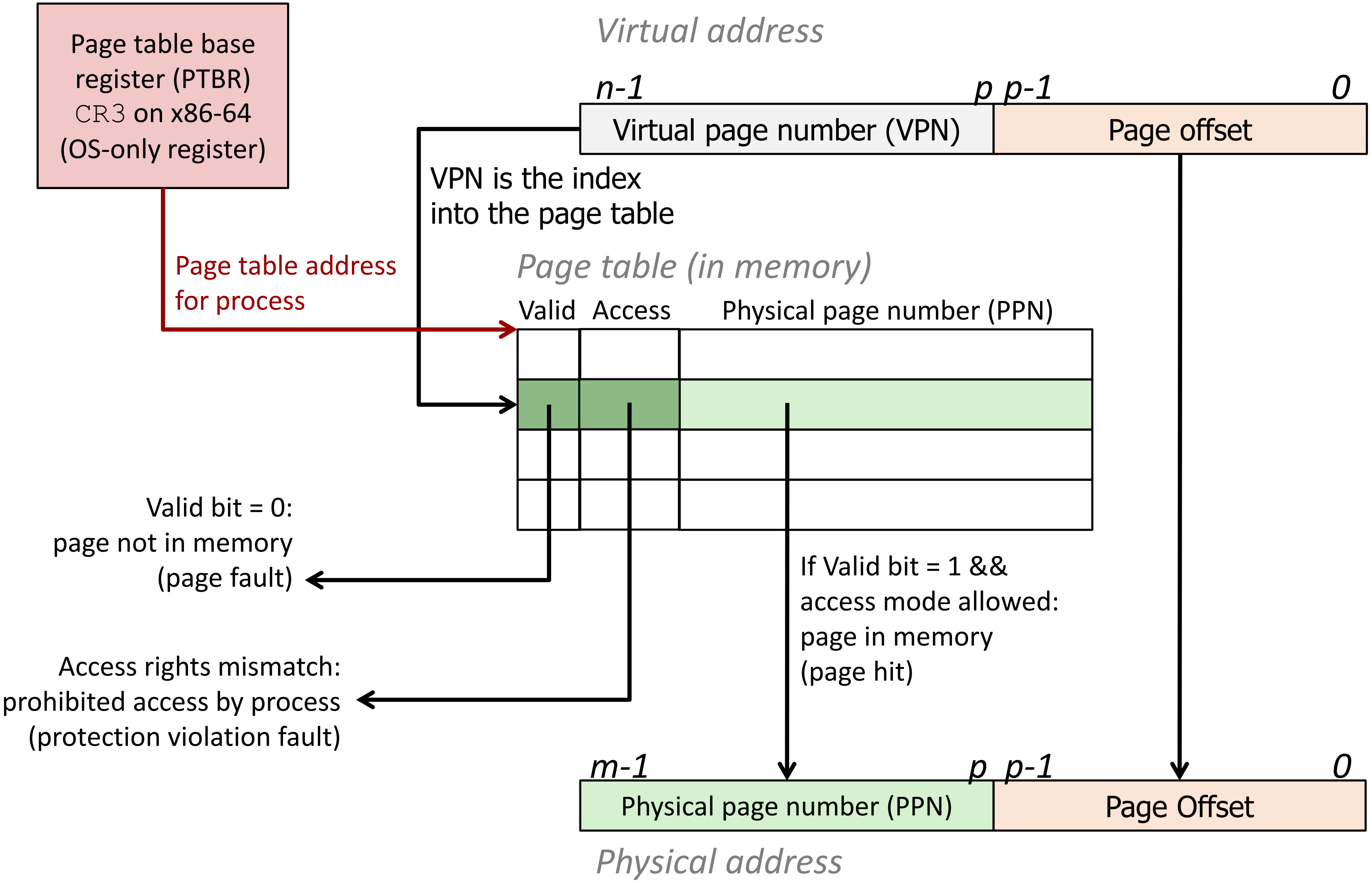
- Memory Problems
- Virtual Memory Concept
- Main Memory as a Cache
- Memory Problems Solved
- **Address Translation**
- Caching Page Table Entries

Address Translation

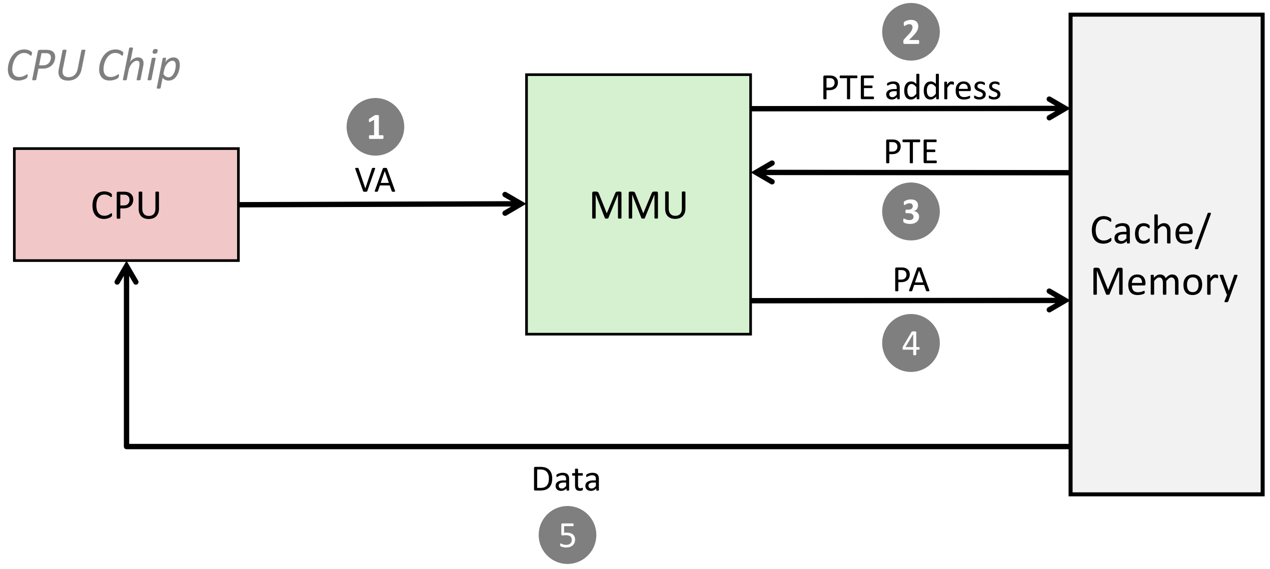
- Goal: Given virtual address, find corresponding physical address
  - (Or get a page fault if the page is not in memory)
  - Translation done by Memory Management Unit (hardware)
  - But mapping itself is maintained by OS (software)
    - Just a table in memory!

- Basic Parameters
  - **$N = 2^n$** : Number of addresses in virtual address space
  - **$M = 2^m$** : Number of addresses in physical address space. $m \leq n$ (usually much less)
  - **$P = 2^p$** : Page size (bytes)
- Components of the virtual address (VA)
  - Virtual page number (VPN)
  - Page Offset
- Components of the physical address (PA)
  - Physical page number (PPN)
  - Page Offset (same offset as VA)

Address Translation With a Page Table



Memory Access: Page Hit



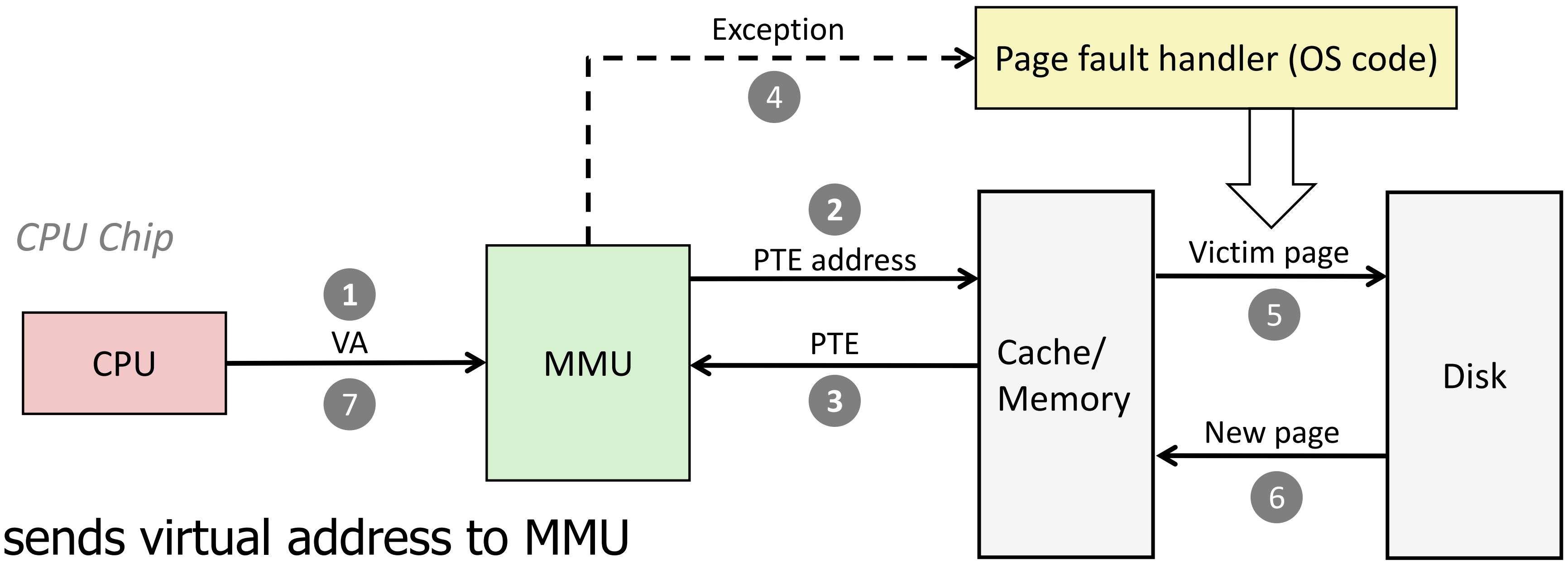
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in cache/memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Memory Access: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in cache/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

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

Mapping can be anything, which is bigger doesn't really matter!

1. How do we split Virtual Addresses into VPN and Offset?

11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

Mapping can be anything, which is bigger doesn't really matter!

1. How do we split Virtual Addresses into VPN and Offset?
   - Offset is based on page size: 64-bytes ⇨ 6 bits

11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Virtual Page Number						Page Offset					

2. How big are Physical Page Numbers?

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

Mapping can be anything, which is bigger doesn't really matter!

1. How do we split Virtual Addresses into VPN and Offset?
   - Offset is based on page size: 64-bytes ⇨ 6 bits

11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Virtual Page Number						Page Offset					

2. How big are Physical Page Numbers? 16-6 = 10 bits

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x3F0
  - VPN:
  - Offset:

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x3F0
  - VPN: 0b001111
  - Offset: 0b110000

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x3F0
  - VPN: 0b001111
  - Offset: 0b110000

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN:
- Offset:

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x3F0
  - VPN: 0b001111
  - Offset: 0b110000

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN: 0b0111110000
- Offset: 0b110000

- Physical address:

Virtual memory example

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x3F0
  - VPN: 0b001111
  - Offset: 0b110000

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN: 0b0111110000
- Offset: 0b110000

- Physical address:
  - 0b0111110000110000
  - 0x7C30

Practice again

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x0D6
  - VPN:
  - Offset:

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN:
- Offset:

- Physical address:

Practice again

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x0D6
  - VPN: 0b000011
  - Offset: 0b010110

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN: 0b0101101111
- Offset: 0b010110

- Physical address:
  - 0b0101101111010110
  - 0x5BD6

Break + Question

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x500
  - VPN:
  - Offset:

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN:
- Offset:

- Physical address:

Break + Question

- Parameters
  - Virtual addresses are 12-bits
  - Physical addresses are 16-bits
  - Page size is 64 bytes

11	10	9	8	7	6	5	4	3	2	1	0
Virtual Page Number						Page Offset					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Page Number										Page Offset					

- Translate:
- Virtual address: 0x500
  - VPN: 0b010100
  - Offset: 0b000000

VPN	PPN	Valid
0x00	0x123	1
0x01	0x156	1
0x02	0x143	1
0x03	0x16F	1
0x04	0x1FF	0
0x05	0x107	0
0x06	0x100	0
0x07	0x1C0	0
0x08	0x1D8	0
0x09	0x1BF	0
0x0A	0x000	1
0x0B	0x3FF	1
0x0C	0x308	0
0x0D	0x3FD	0
0x0E	0x111	1
0x0F	0x1F0	1

VPN	PPN	Valid
0x10	0x237	1
0x11	0x236	1
0x12	0x2B0	1
0x13	0x280	0
0x14	0x120	0
Continues on…		

- PPN: INVALID
- Offset:
- Physical address:
  - **Page Fault**

Outline

- Memory Problems
- Virtual Memory Concept
- Main Memory as a Cache
- Memory Problems Solved
- Address Translation
- **Caching Page Table Entries**

Accessing page tables is slow

- Problem: page tables are in memory
  - And we need to access them to find our address to access memory
  - Two memory accesses per access!!! 😱

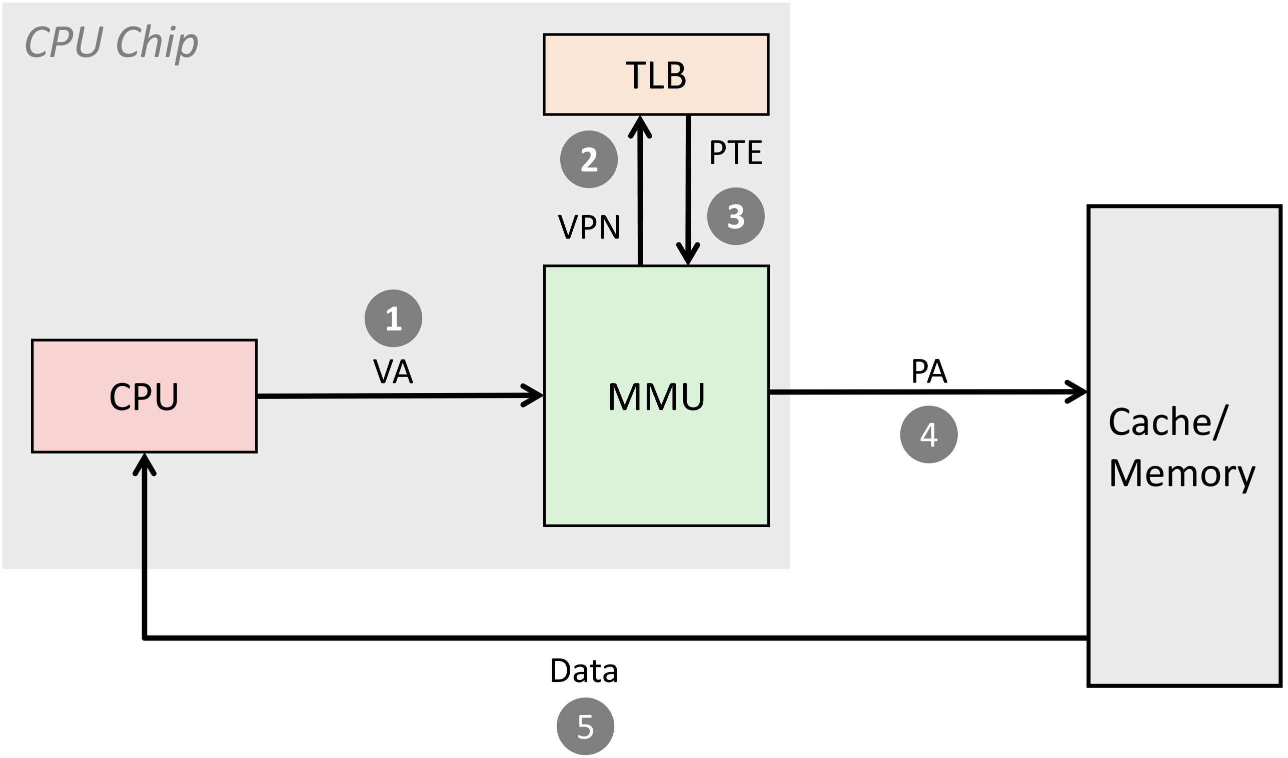
- Page table entries (PTEs) are cached in L1, L2, etc, like any other data in memory
  - PTEs may be evicted by other data references. Oops.
  - PTE access still requires average effective memory access delay

Speeding up Translation with a TLB

- Solution: *Translation Lookaside Buffer* (TLB)
  - Small hardware cache memory inside MMU
  - Contains page table entries for a small number of pages
  - Maps virtual page numbers to physical page numbers
  - Reduces issues with data kicking PTEs out of caches!

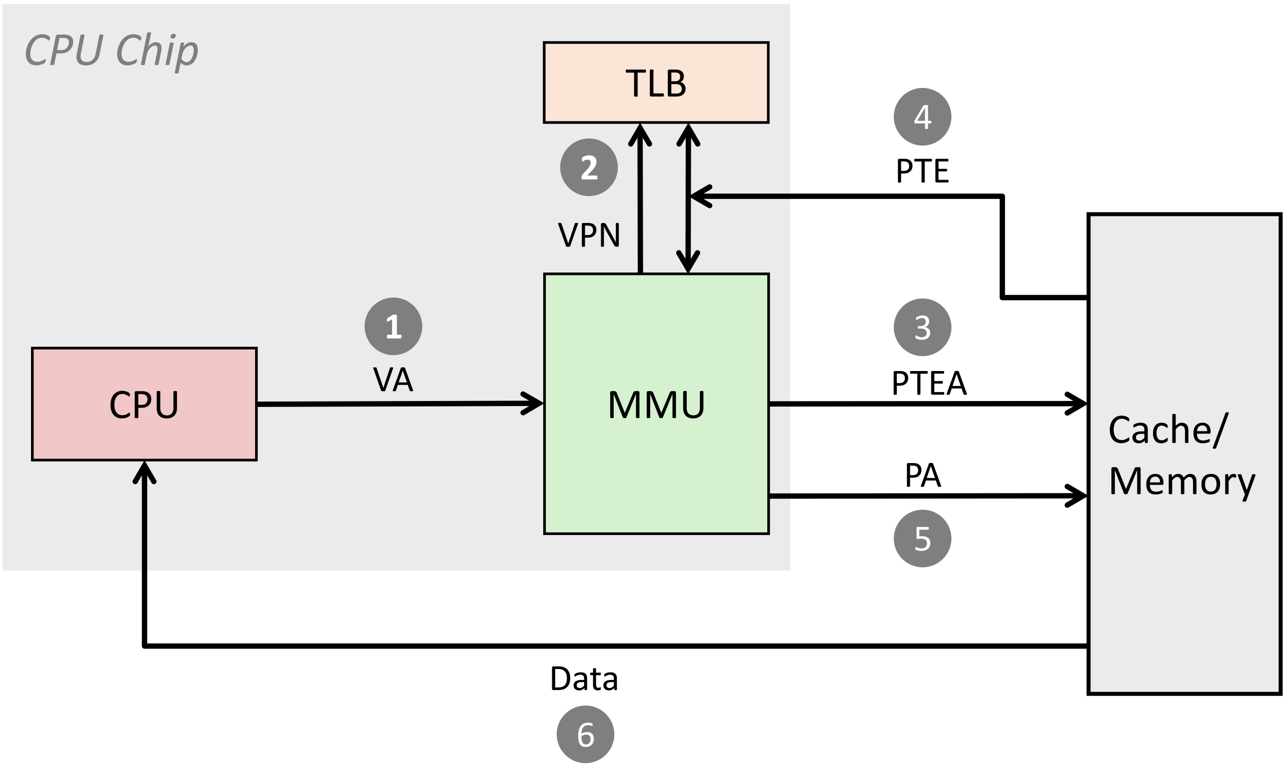
- Like cache memories, uses set indices, tags, and valid bits
  - VPN split into: TLB tag and TLB index (just like caches, because it is one!)
  - No need for a block offset equivalent (PTEs have a single value)

TLB Hit



A TLB hit eliminates a memory access

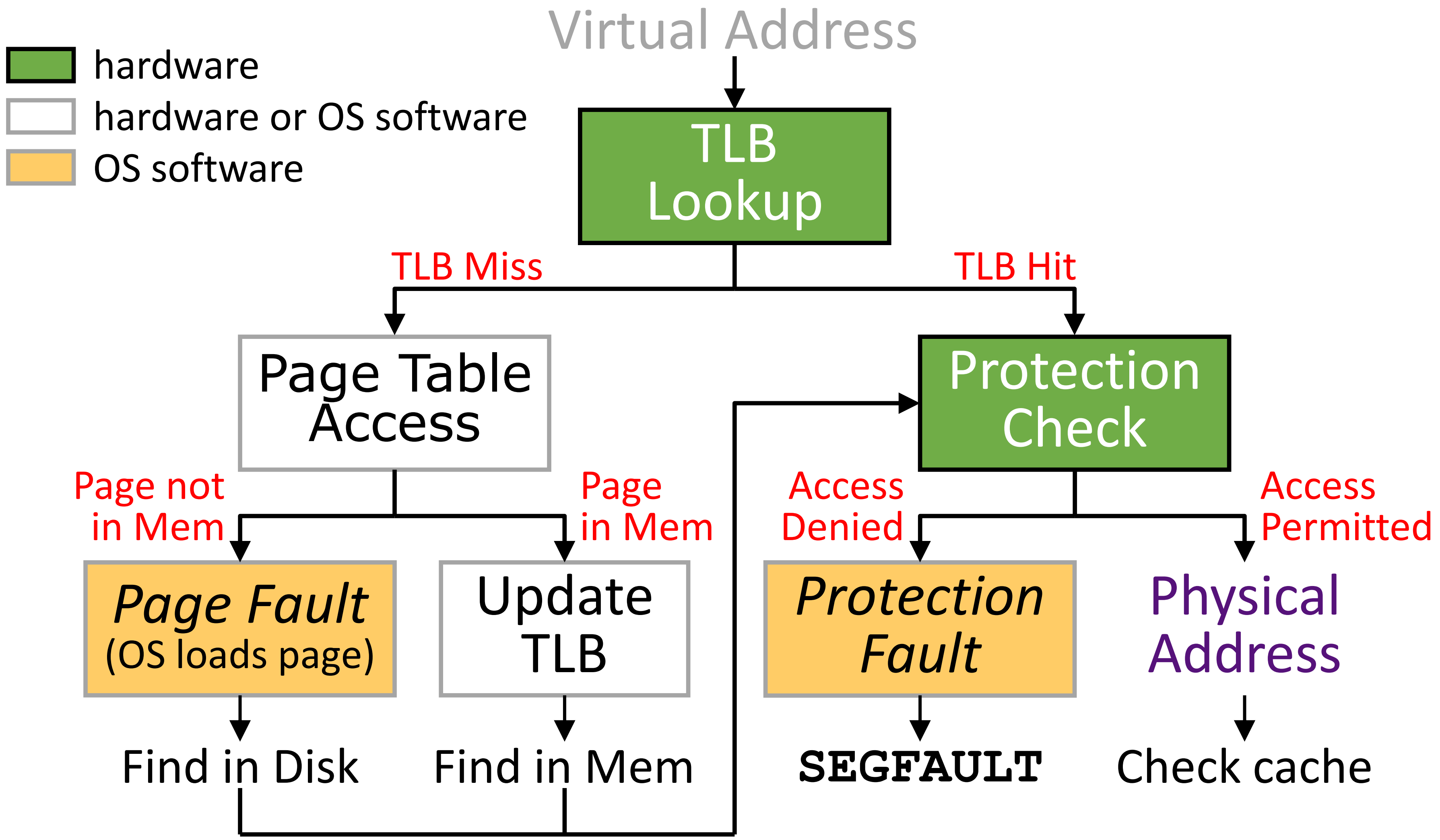
TLB Miss



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

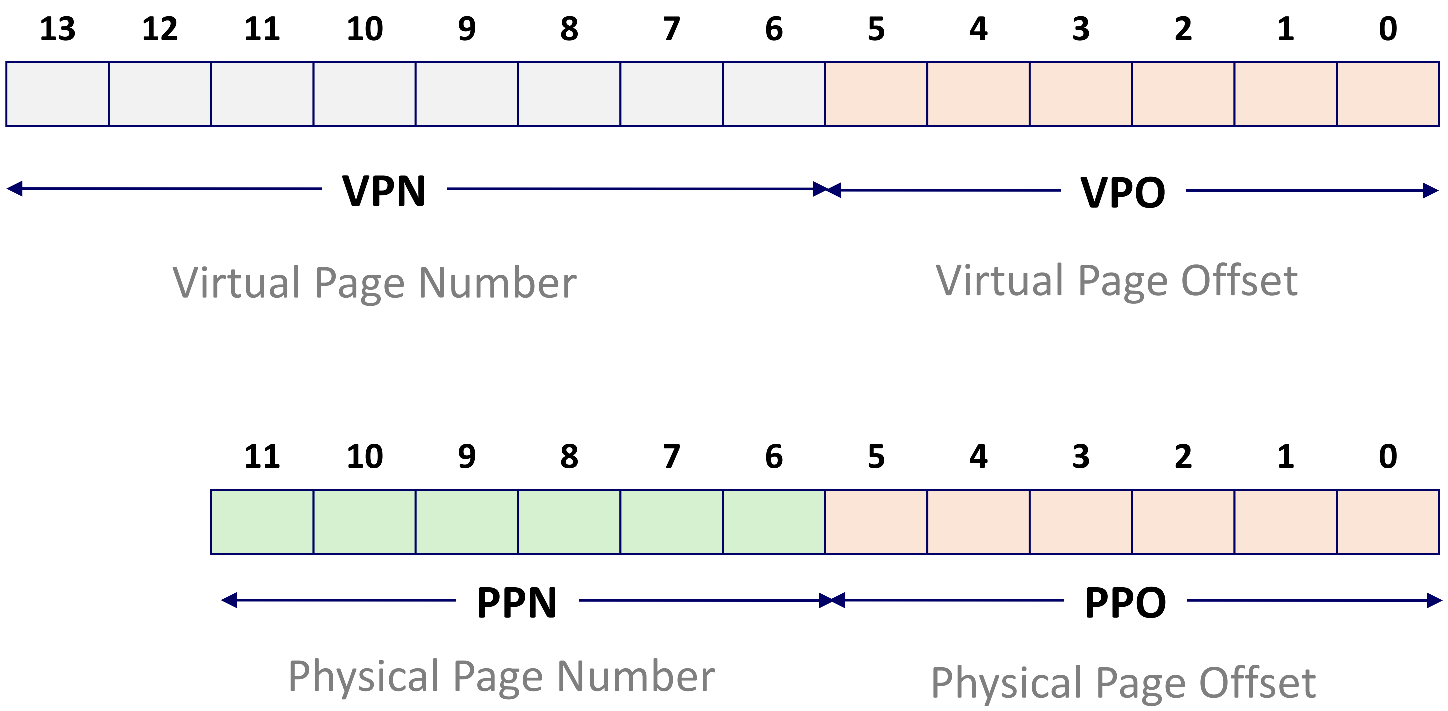
Fortunately, TLB misses are rare. Why? Locality. It's always locality.

Address translation process



Simple Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes



Simple Memory System: Page Table

We only show a few entries (out of 256)

VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

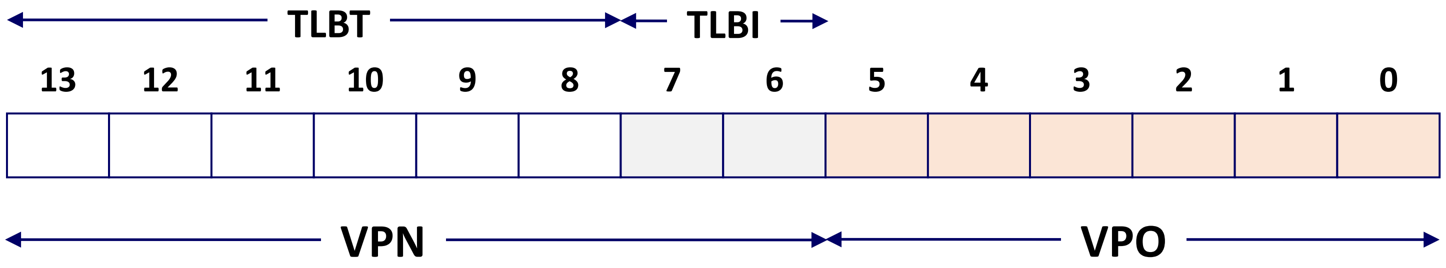
…

VPN	PPN	Valid
2E	-	0

⋮

Simple Memory System: TLB

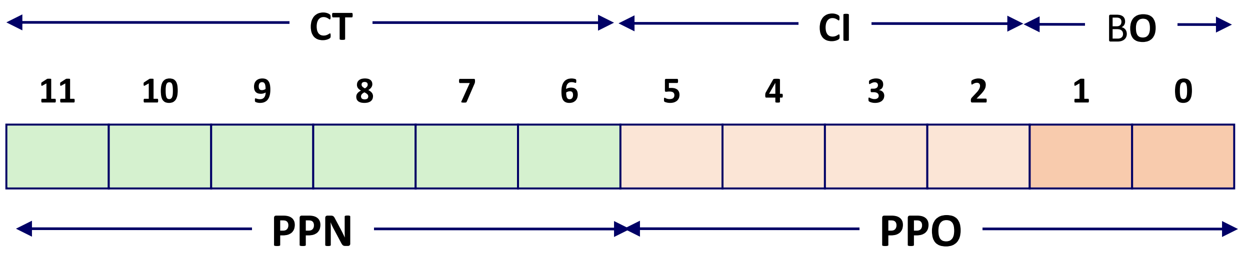
- 16 entries
- 4-way associative



Set	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*	*Tag*	*PPN*	*Valid*
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Simple Memory System: L1 Cache

- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped



Idx	*Tag*	*Valid*	*B3*	*B2*	*B1*	*B0*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	1D	72	F0	36
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Idx	*Tag*	*Valid*	*B3*	*B2*	*B1*	*B0*
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

Address Translation Example #1

(using the Page Table, TLB, and L1 cache shown in the preceding slides)

movb (%rcx), %al

Address space: 14-bit VAddr, 12-bit PAddr, 64-byte page
TLB: 16 entries, 4-way
L1 Cache: 16 lines, 4-byte block, direct mapped,
Physically addressed

Virtual Address: 0x03D4

TLBT: bits 13–8; TLBI: bits 7–6
VPN: bits 13–6; VPO: bits 5–0

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	1	0	0

VPN **0x0F** TLBI **0x3** TLBT **0x03** TLB Hit? **Y** Page Fault? **N** PPN: **0x0D**

Physical Address

CT: bits 11–6; CI: bits 5–2; BO: bits 1–0
PPN: bits 11–6; PPO: bits 5–0

11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0

BO **0** CI **0x5** CT **0x0D** Hit? **Y** Byte: **0x36**

Address Translation Example #2

(using the Page Table, TLB, and L1 cache shown in the preceding slides)

Virtual Address: `0x0B8F`

TLBT: bits 13–8 | TLBI: bits 7–6

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	1	**0**	**1**	**1**	**1**	0	**0**	0	1	**1**	1	1

VPN: bits 13–6 | VPO: bits 5–0

VPN **0x2E** TLBI **2** TLBT **0x0B** TLB Hit? **N** Page Fault? **Y** PPN: **TBD**

Physical Address

CT: bits 11–6 | CI: bits 5–2 | BO: bits 1–0

11	10	9	8	7	6	5	4	3	2	1	0

PPN: bits 11–6 | PPO: bits 5–0

BO ___ CI___ CT ____ Hit? __ Byte: ____

Likely invalid page. Maybe needs to read from disk. Either way we don't know the PPN.

Address Translation Example #3

(using the Page Table, TLB, and L1 cache shown in the preceding slides)

Virtual Address: `0x0020`

	TLBT					TLBI							
13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0
VPN								VPO					

VPN **0x00** TLBI 0 TLBT **0x00** TLB Hit? N Page Fault? N PPN: **0x28**

Physical Address

CT						CI				BO	
11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0	0	0	0	0
PPN						PPO					

BO **0** CI **0x8** CT **0x28** Hit? N Byte: Mem

Cache miss, so needs to read byte values from main memory

Outline

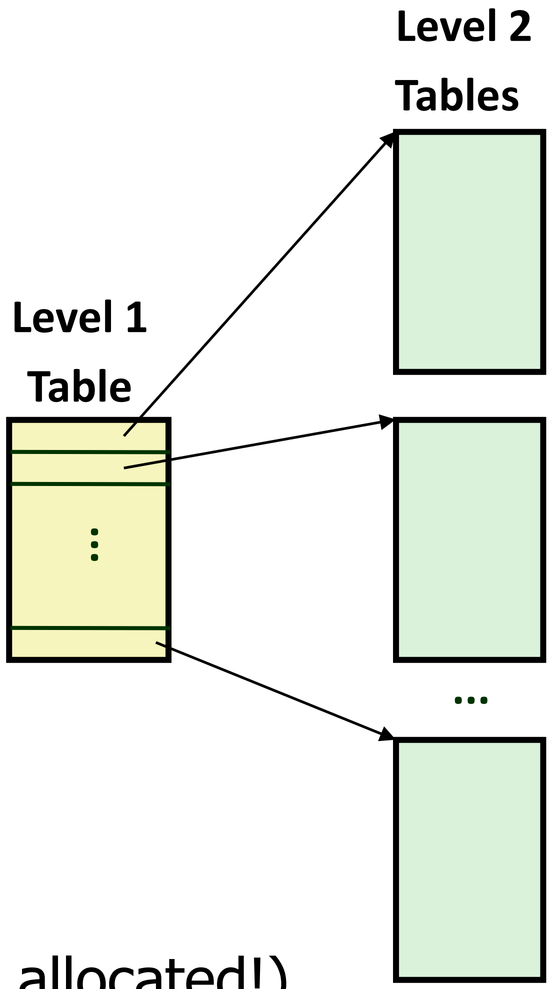
- Memory Problems
- Virtual Memory Concept
- Main Memory as a Cache
- Memory Problems Solved
- Address Translation
- Caching Page Table Entries

Outline

- Bonus: Multi-level Page Tables

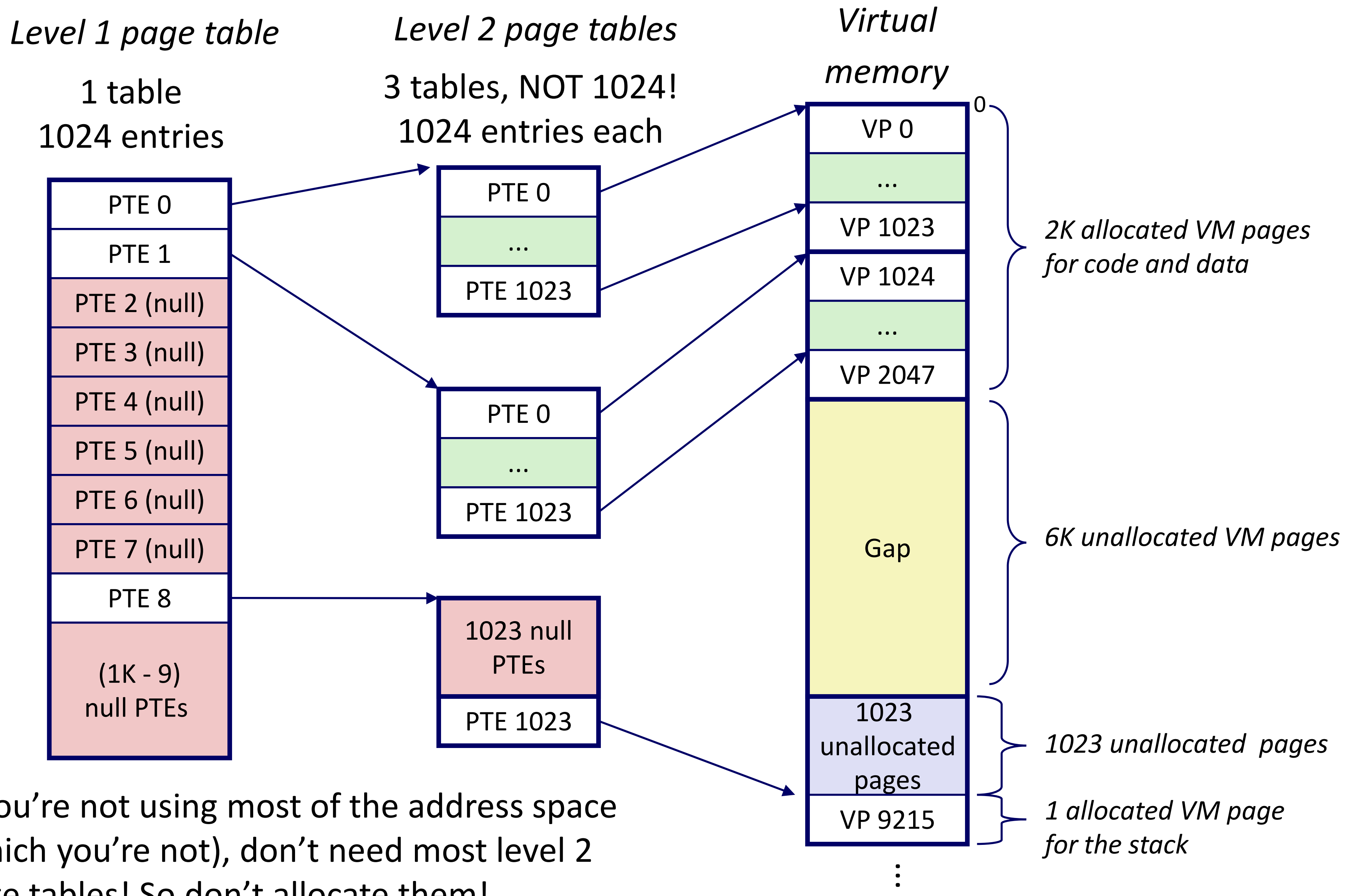
Multi-Level Page Tables

- Suppose:
  - 4KB (2^{12}) page size, 48-bit address space, 8-byte PTE
- How big is the page table?
  - Would need a 512 GB page table!
    - $2^{48} * 2^{-12} * 2^{3} = 2^{39}$ bytes
  - That's just meta-data!
    Where does the data go?
- Common solution:
  - Multi-level page tables
  - Split the VPN into multiple pieces, 1 per level
  - Example: 2-level page table
    - Level 1 table: each PTE points to a level 2 page table
      (always memory resident)
    - Level 2 table: each PTE points to a page
      (paged in and out like any other data, maybe not even allocated!)



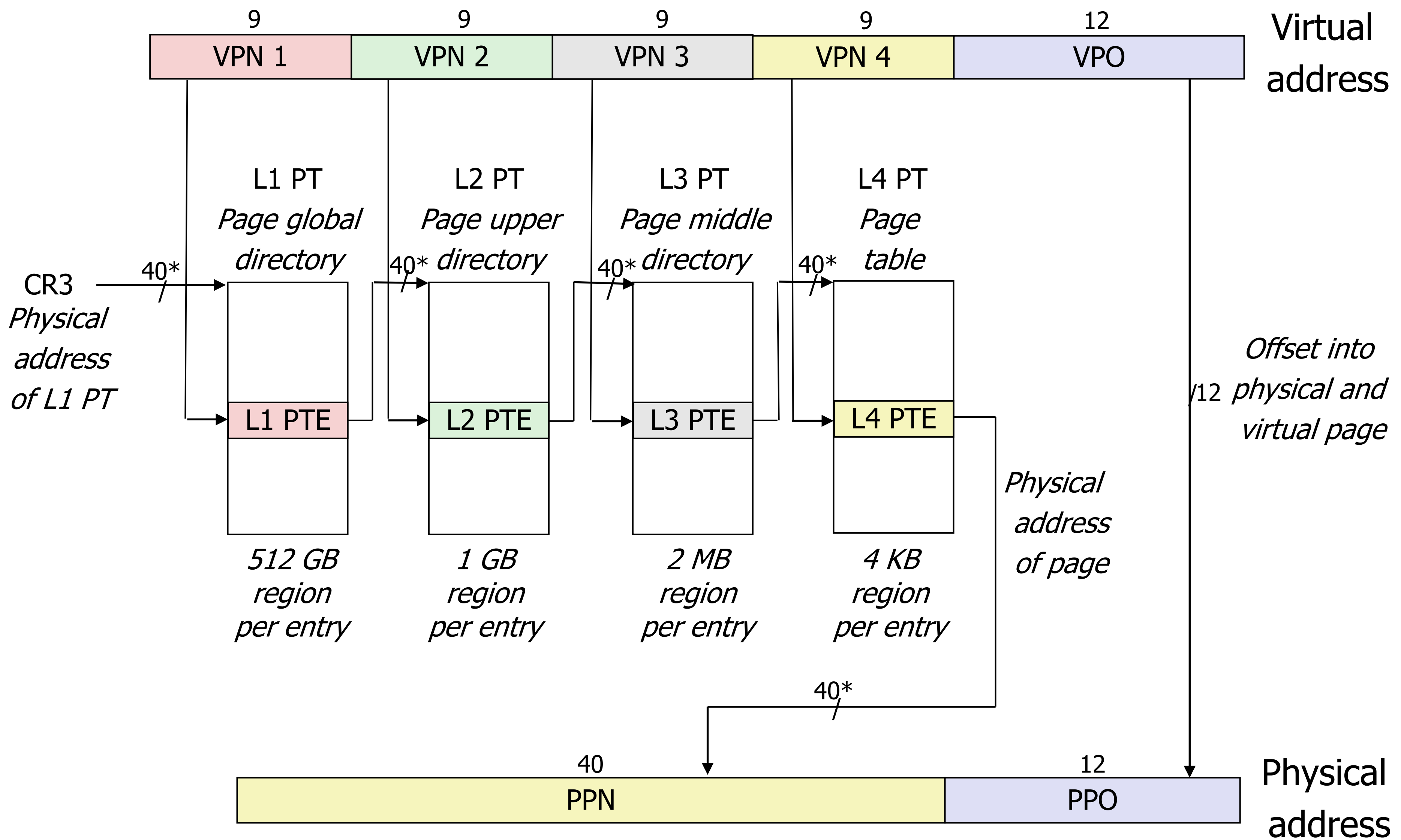
A Two-Level Page Table Hierarchy

32 bit addresses, 4KB pages, 4-byte PTEs



If you're not using most of the address space (which you're not), don't need most level 2 page tables! So don't allocate them!

Multi-level page table: Core i7



*aligned to a 4K-boundary

End-to-end Core i7 Data Address Translation