Virtual Memory Start

Computer Operating Systems, Fall 2023

Instructor: Travis McGaha

Head TAs: Nate Hoaglund & Seungmin Han

TAs:

Andy Jiang	Haoyun Qin	Kevin Bernat	Ryoma Harris
Audrey Yang	Jason hom	Leon Hertzberg	Shyam Mehta
August Fu	Jeff Yang	Maxi Liu	Tina Kokoshvili
Daniel Da	Jerry Wang	Ria Sharma	Zhiyan Lu
Ernest Ng	Jinghao Zhang	Rohan Verma	

Poll Everywhere

pollev.com/tqm

- Should we go over check-ins in lecture?

Administrivia

- **Partner sign up**
  - **CHECK CANVAS NOW IF YOU AREN'T IN A GROUP WITH YOUR PARTNER, LET US KNOW AS SOON AS POSSIBLE**
- I am going to set people up tonight and post an announcement once random pairs are "done"
- Check-in Quiz 3 Due in on Tuesday

Administrivia

- Peer Evaluation: out now, due ~~Friday 9/22~~ **Tuesday 9/26 @ 11:59 pm** (extended)
  - Please do it, it shouldn't take long
  - Mostly completion, don't just say "this is fine" for everything

- Project 1 is out now
  - The milestone is due Wed 9/27 @ 11:59 pm
    late deadline: 11:59 pm on Sun, Oct 01
  - Project is due 11:59 pm on Wed, Oct 11
    late deadline 11:59 pm on Sun, Oct 15
  - Demo of project 1 in the last half of this class
  - After this class, should have everything needed to complete the project

Follow-ups

- **`setitimer`**
  - Can set a timer to be less than a second, **`setitimer`** allows you to specify microseconds
- Typos: I do care
- View open file descriptors for a process:
  - **`ls -l /proc/<PID>/fd/`**
  - **`lsof -p <PID>`**

Poll Everywhere

pollev.com/tqm

- Any questions, comments or concerns from last lecture?

Lecture Outline

- **Problems with old memory model**
- Virtual Memory High Level
- Address Translation

Poll Everywhere

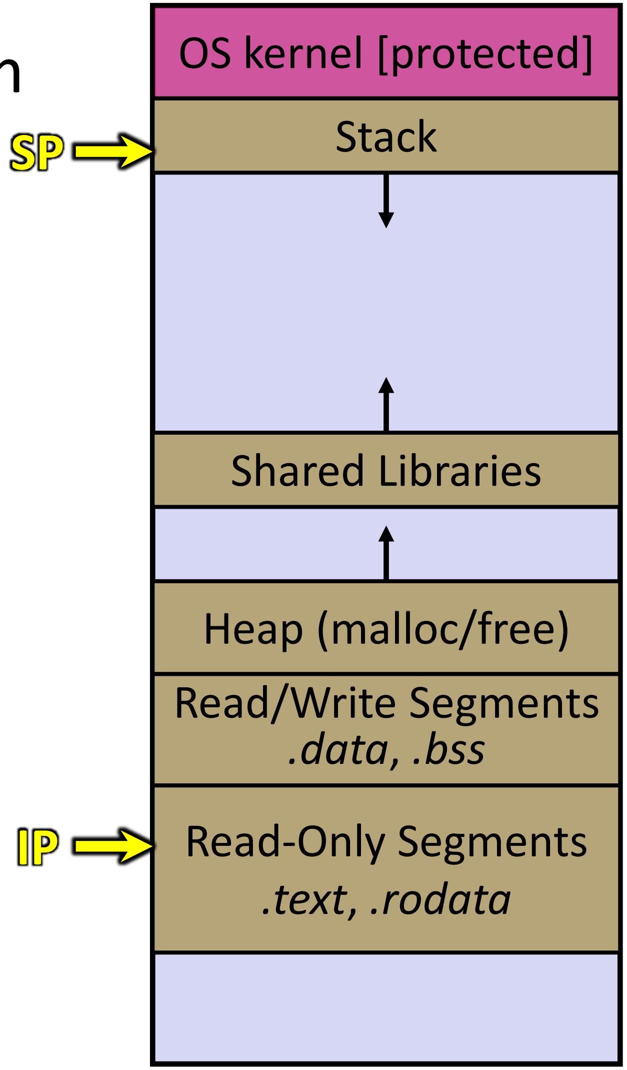
pollev.com/tqm

- What does this print for **`x`** and the **`ptr`** at all three points in the code?

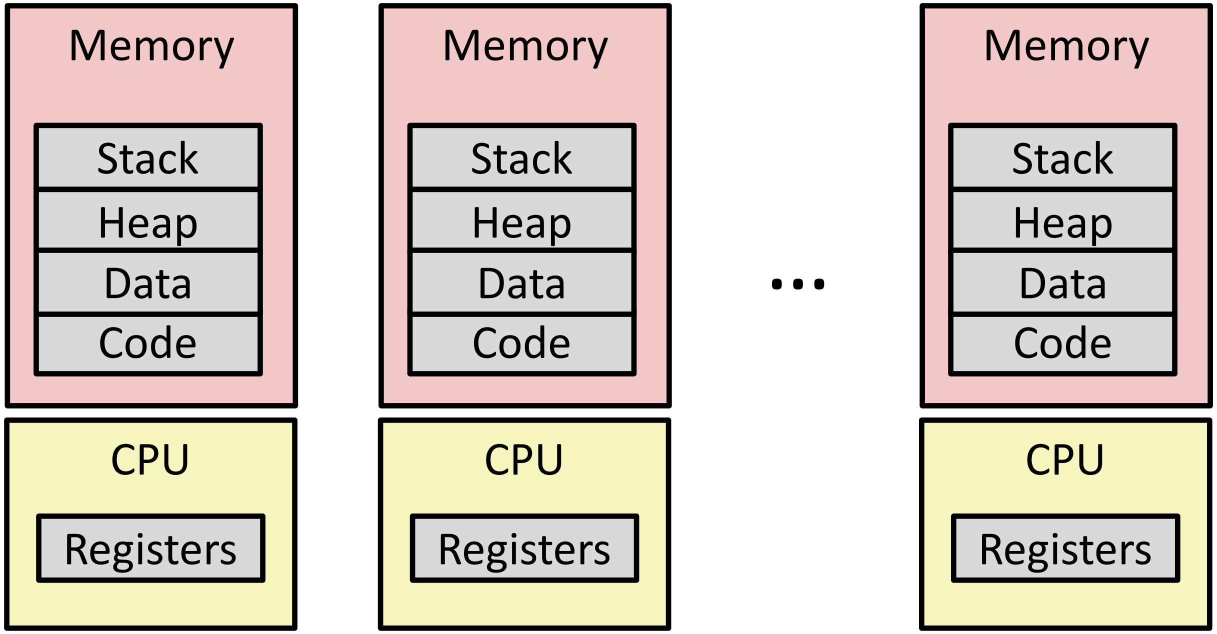
```c
int main() {
  int x = 3;
  int *ptr = &x;

  printf("[Before Fork]\t x = %d\n", x);
  printf("[Before Fork]\t ptr = %p\n", ptr);

  pid_t pid = fork();
  if (pid < 0) {
    perror("fork errored");
    return EXIT_FAILURE;
  }

  if (pid == 0) {
    x += 2;
    printf("[Child]\t\t x = %d\n", x);
    printf("[Child]\t\t ptr = %p\n", ptr);

    return EXIT_SUCCESS;
  }

  // assume no error
  waitpid(pid, NULL, 0);

  x -= 2;
  printf("[Parent]\t x = %d\n", x);
  printf("[Parent]\t ptr = %p\n", ptr);

  return EXIT_SUCCESS;
}
```

Review: Processes

- Definition: An instance of a program that is being executed (or is ready for execution)
- Consists of:
  - Memory (code, heap, stack, etc)
  - Registers used to manage execution (stack pointer, program counter, ...)
  - Other resources

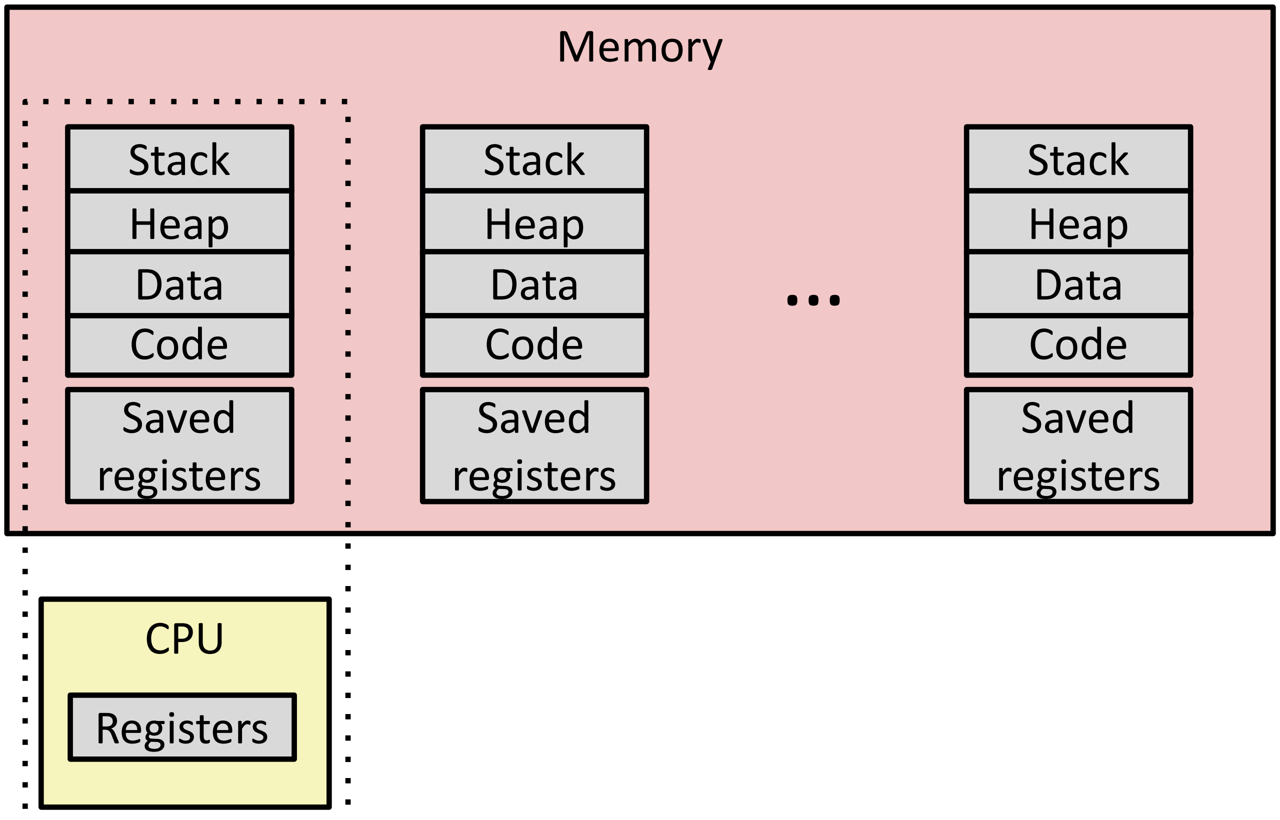


Multiprocessing: The Illusion



- Computer runs many processes simultaneously
  - Applications for one or more users
    - Web browsers, email clients, editors, …
  - Background tasks
    - Monitoring network & I/O devices

Multiprocessing: The (Traditional) Reality



- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (later in course)
  - Register values for nonexecuting processes saved in memory

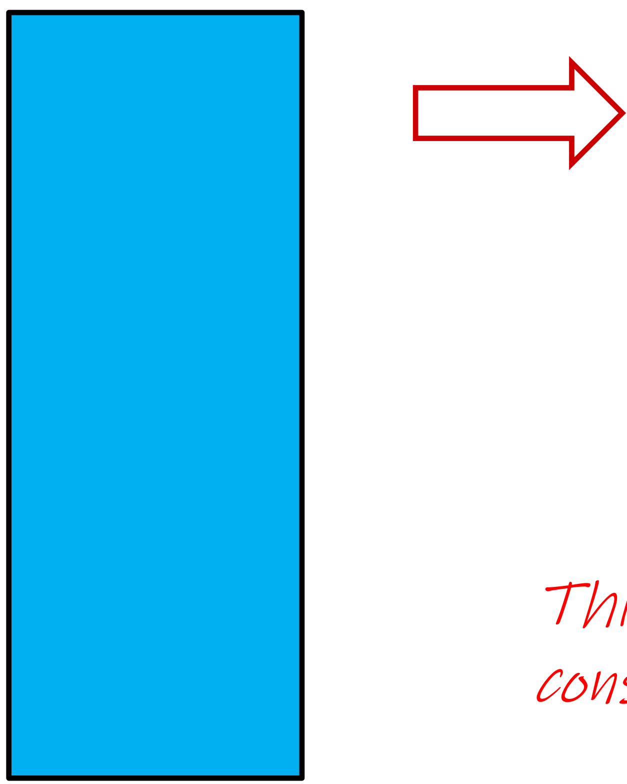
Memory (as we know it now)

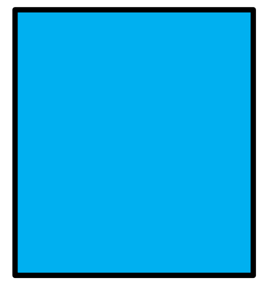
- The CPU directly uses an address to access a location in memory

Problem 1: How does everything fit?

On a 64-bit machine, there are 2^{64} bytes, which is:
18,446,744,073,709,551,616 Bytes
(1.844 x 10^{19})

Laptops usually have around 8GB which is 8,589,934,592 Bytes (8.589 x 10^{9})

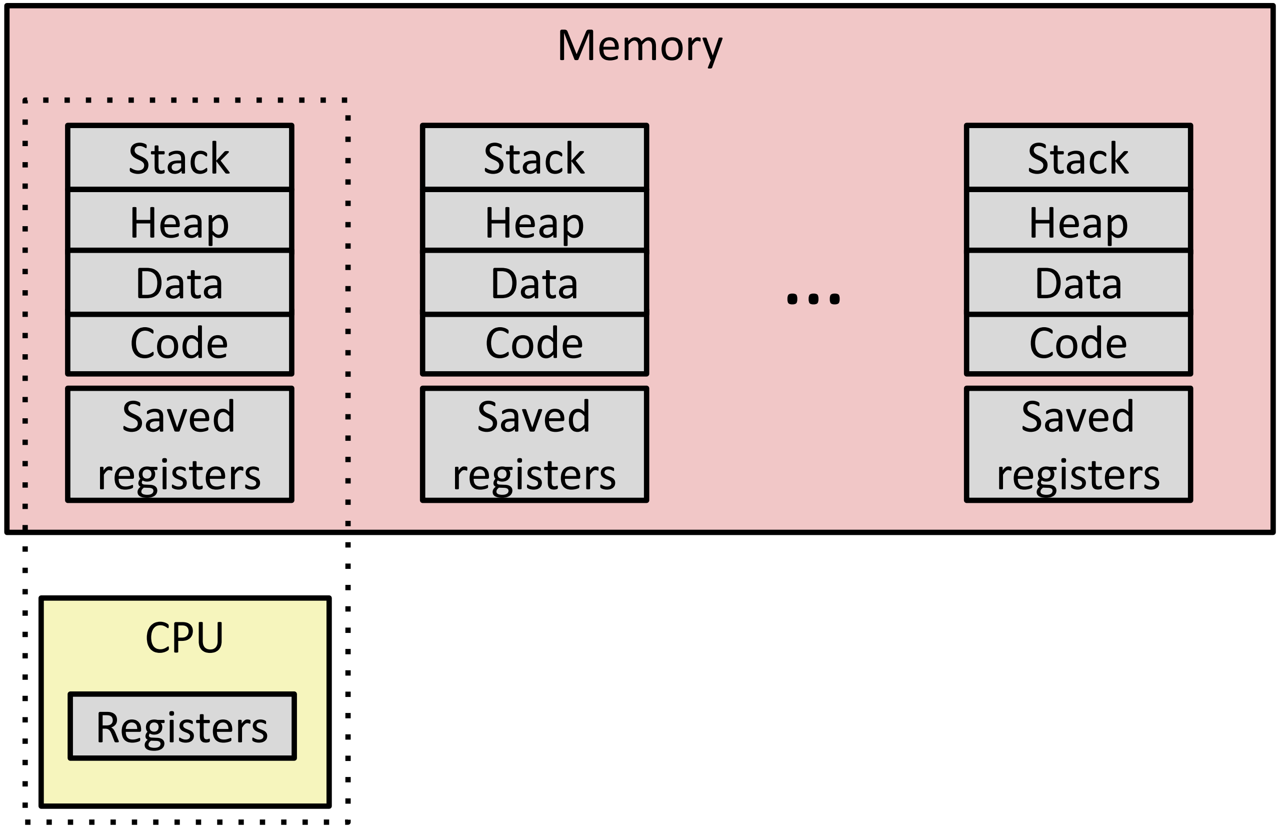




(Not to scale; physical memory is smaller than the period at the end of the sentence compared to the virtual address space.)

This is just one address space, consider multiple processes…

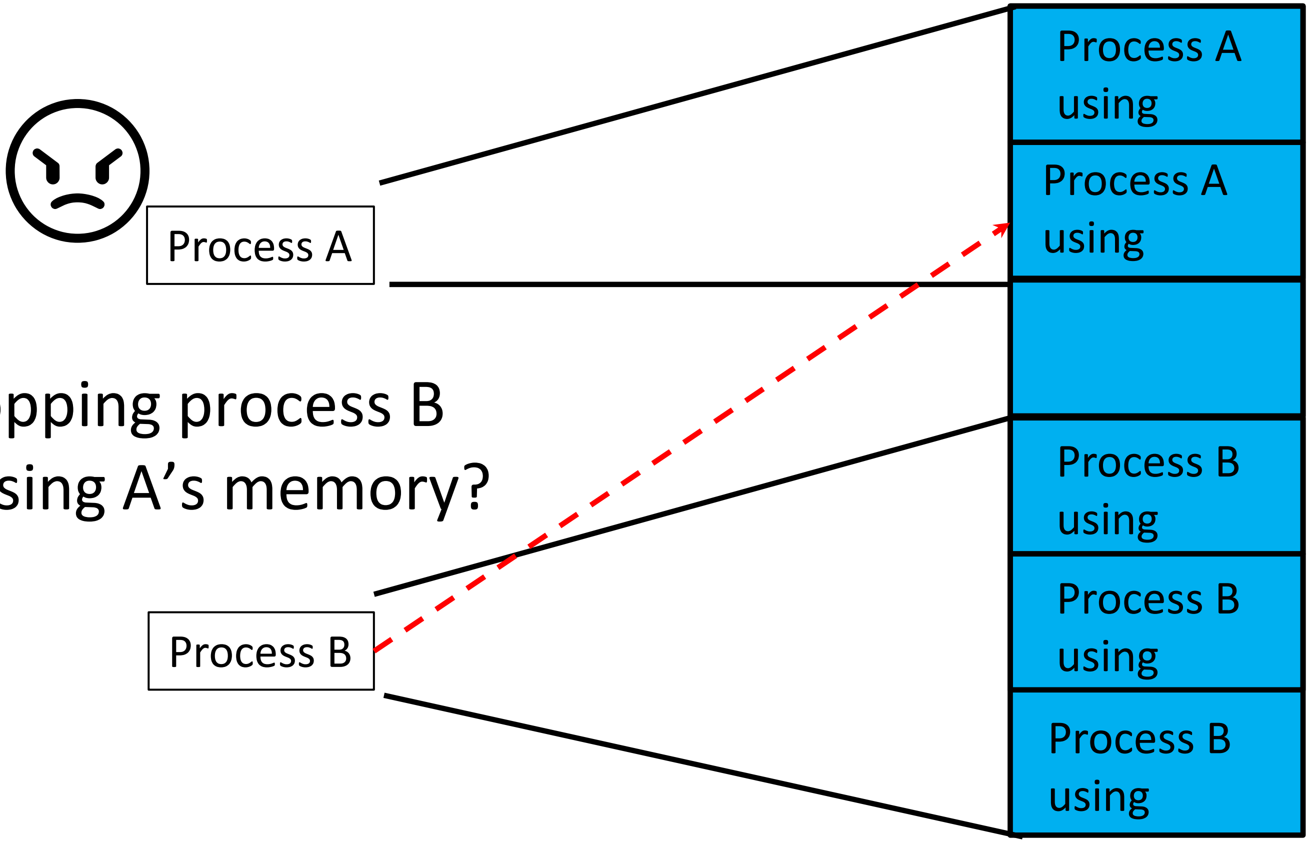
Problem 2: Sharing Memory



- How do we enforce process isolation?
  - Could one process just calculate an address into another process?

Problem 2: Sharing Memory

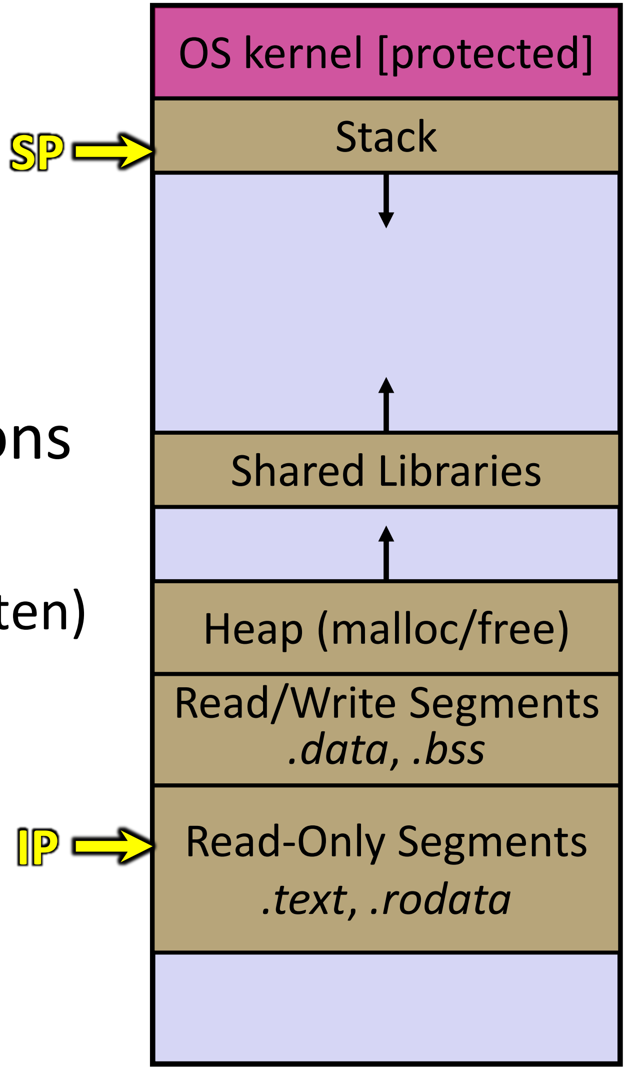
- How do we enforce process isolation?
  - Could one process just calculate an address into another process?



- What is stopping process B from accessing A's memory?

Problem 3: How do we segment things

- A process' address space contains many different "segments"
- How do we keep track of which segment is which and the permissions each segment may have?
  - (e.g., that Read-Only data can't be written)



Lecture Outline

- Problems with old memory model
- **Virtual Memory High Level**
- Address Translation

This doesn't work anymore

- The CPU directly uses an address to access a location in memory

Indirection

- "Any problem in computer science can be solved by adding another level of indirection."
  - David wheeler, inventor of the subroutine (e.g. functions)

- The ability to indirectly reference something using a name, reference or container instead of the value itself. A flexible mapping between a name and a thing allows chagcing the thing without notifying holders of the name.
  - May add some work to use indirection
  - Example: Phone numbers can be transferred to new phones

- Idea: instead of directly referring to physical memory, add a level of indirection

Idea:

- We don’t need all processes to have their data in physical memory, **just the ones that are currently running**

- For the process’ that are currently running: we don’t need all their data to be in physical memory, **just the parts that are currently being used**

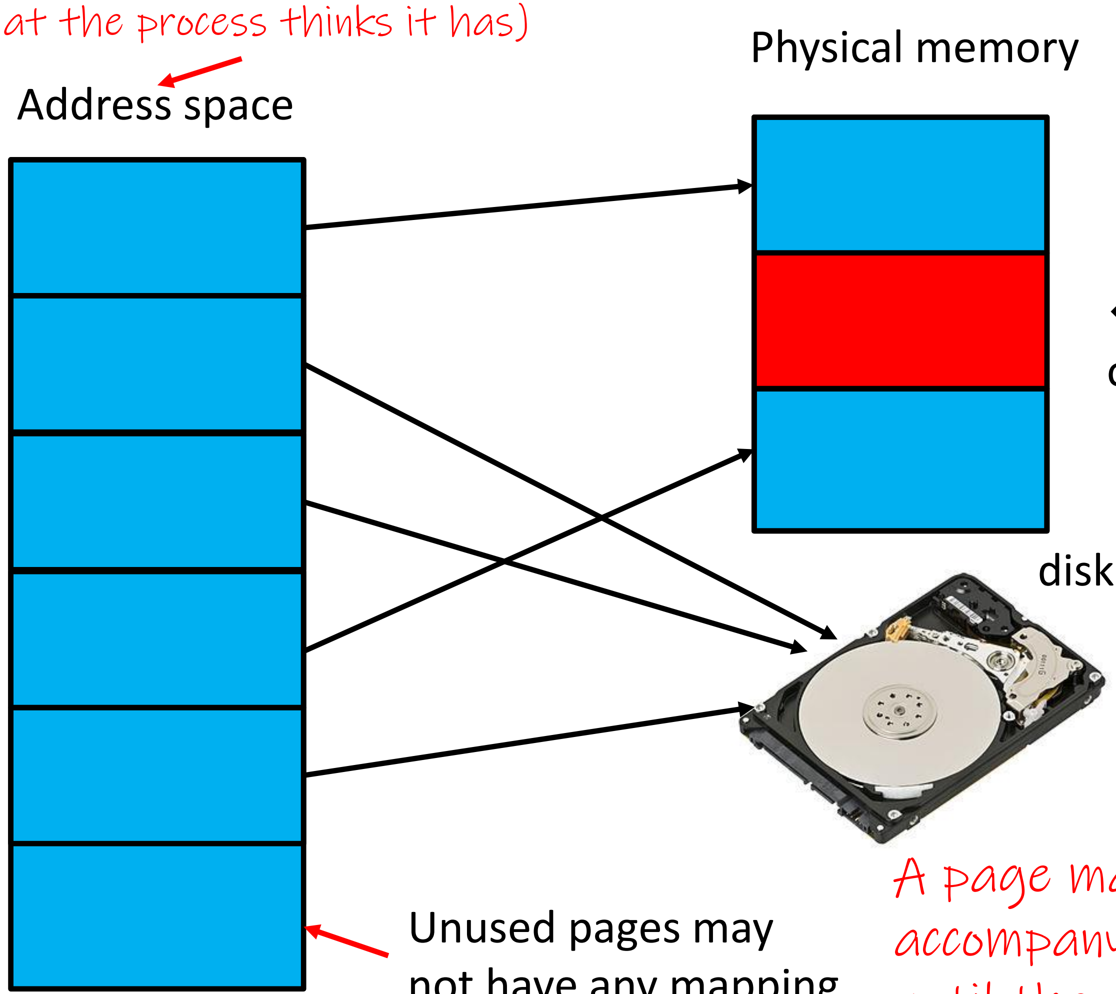
- Data that isn’t currently stored in physical memory, can be stored elsewhere (disk).
  - Disk is "permanent storage" usually used for the file system
  - Disk has a longer access time than physical memory (RAM)

Pages

Pages are of fixed size ~4KB
4KB -> (4 * 1024 = 4096 bytes.)

- Memory can be split up into units called "pages"

(what the process thinks it has)



Pages currently in use are stored in physical memory (RAM)

← Ram may contain pages from other active processes

Pages in physical memory are called "Page frames"

Pages not currently in use (but were used in the past) are stored on disk

A page may not have an accompanying page frame until the page is lost

Definitions

Sometimes called "virtual memory" or the "virtual address space"

- Addressable Memory: the total amount of memory that can be theoretically be accessed based on:
  - number of addresses ("address space")
  - bytes per address ("addressability")

IT MAY OR MAY NOT EXIST ONHARDWARE (like if that memory is never used)

- Physical Memory: the total amount of memory that is physically available on the computer

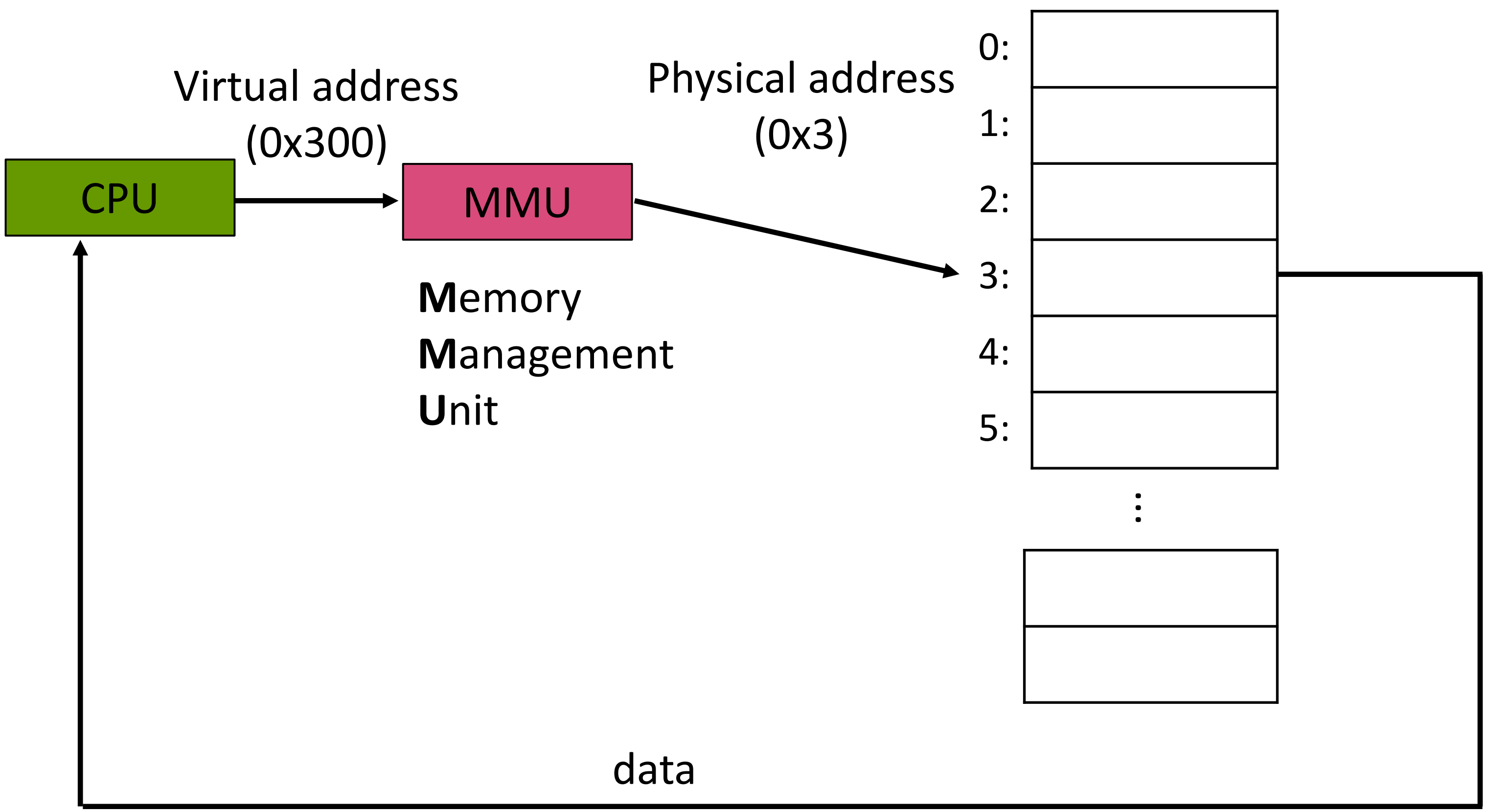
Physical memory holds a subset of the addressable memory being used

- Virtual Memory: An abstraction technique for making memory look larger than it is and hides many details from the programs.

Virtual Address Translation

THIS SLIDE IS KEY TO THE WHOLE IDEA

- Programs don't know about physical addresses; virtual addresses are translated into them by the MMU



Page Tables

More details about translation later

- Virtual addresses can be converted into physical addresses via a page table.

- There is one page table per processes, managed by the MMU

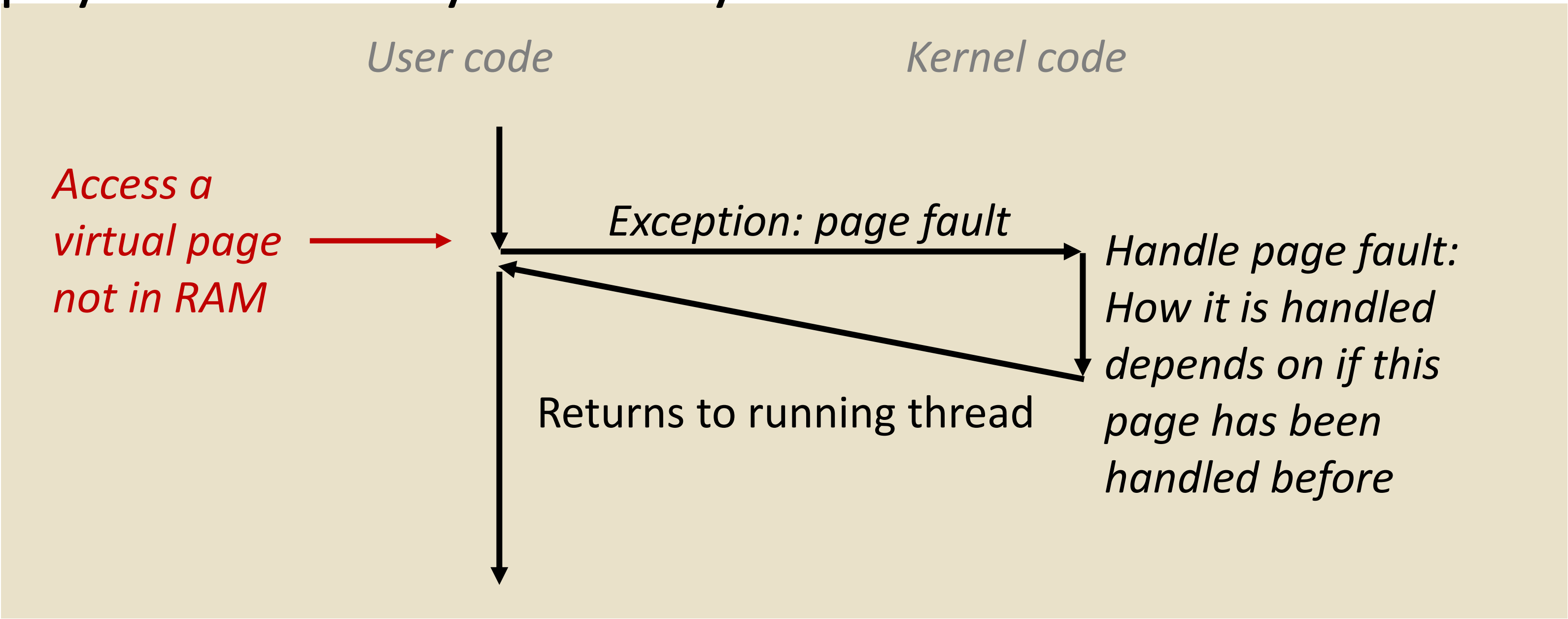
Virtual page #	Valid	Physical Page Number
0	0	null //page hasn't been used yet
1	1	0
2	1	1
3	0	disk

Valid determines if the page is in physical memory

If a page is on disk, MMU will fetch it

Page Fault Exception

- An *Exception* is a transfer of control to the OS *kernel* in response to some ***synchronous event*** *(directly caused by what was just executed)*
- In this case, writing to a memory location that is not in physical memory currently



Problem: Paging Replacement

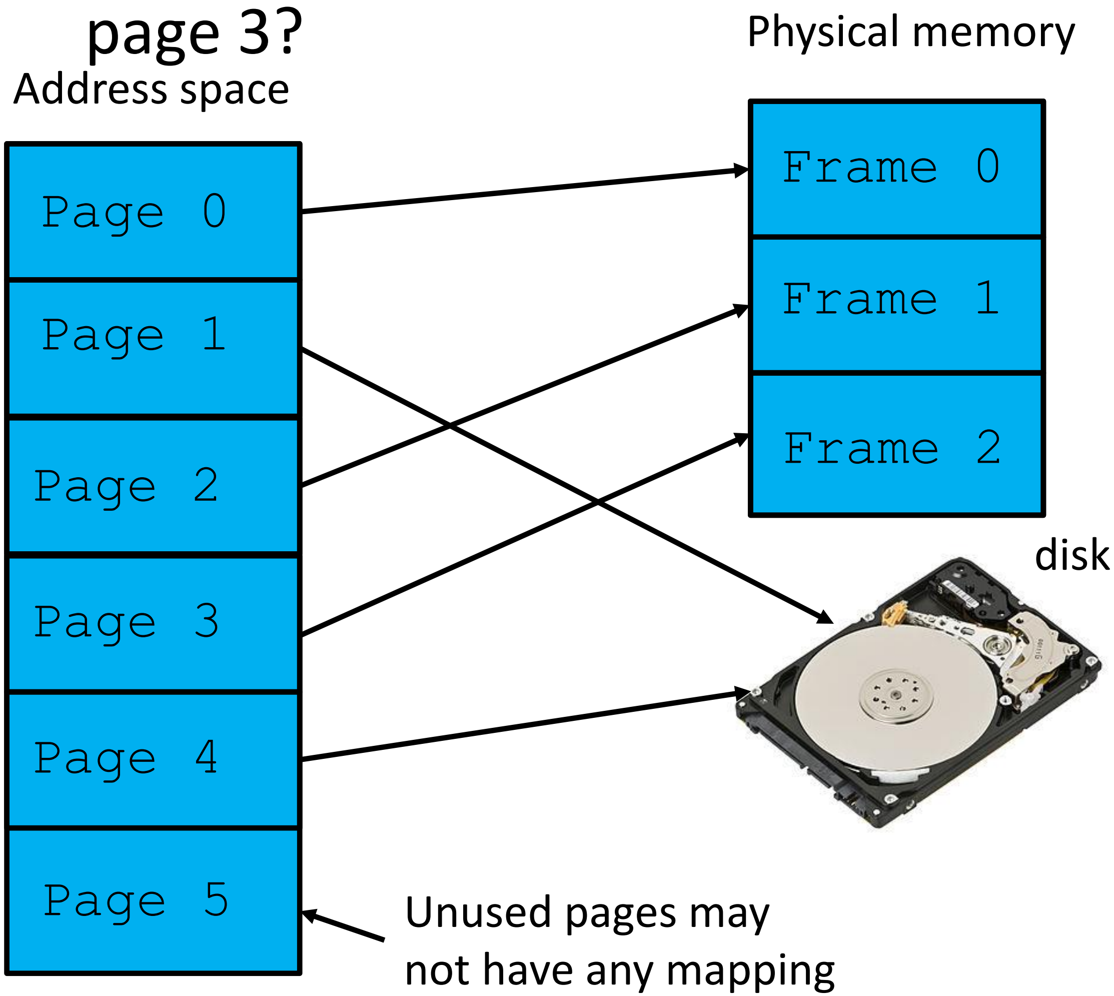
More details about page replacement later

- We don’t have space to store all active pages in physical memory.
- If physical memory is full and we need to load in a page, then we choose a page in physical memory to store on disk in the **swap file**
- If we need to load in a page from disk, how do we decide which page in physical memory to “evict”
- Goal: Minimize the number of times we have to go to disk. It takes a while to go to disk.

Poll Everywhere

pollev.com/tqm

- What happens if this process tries to access an address in page 3?



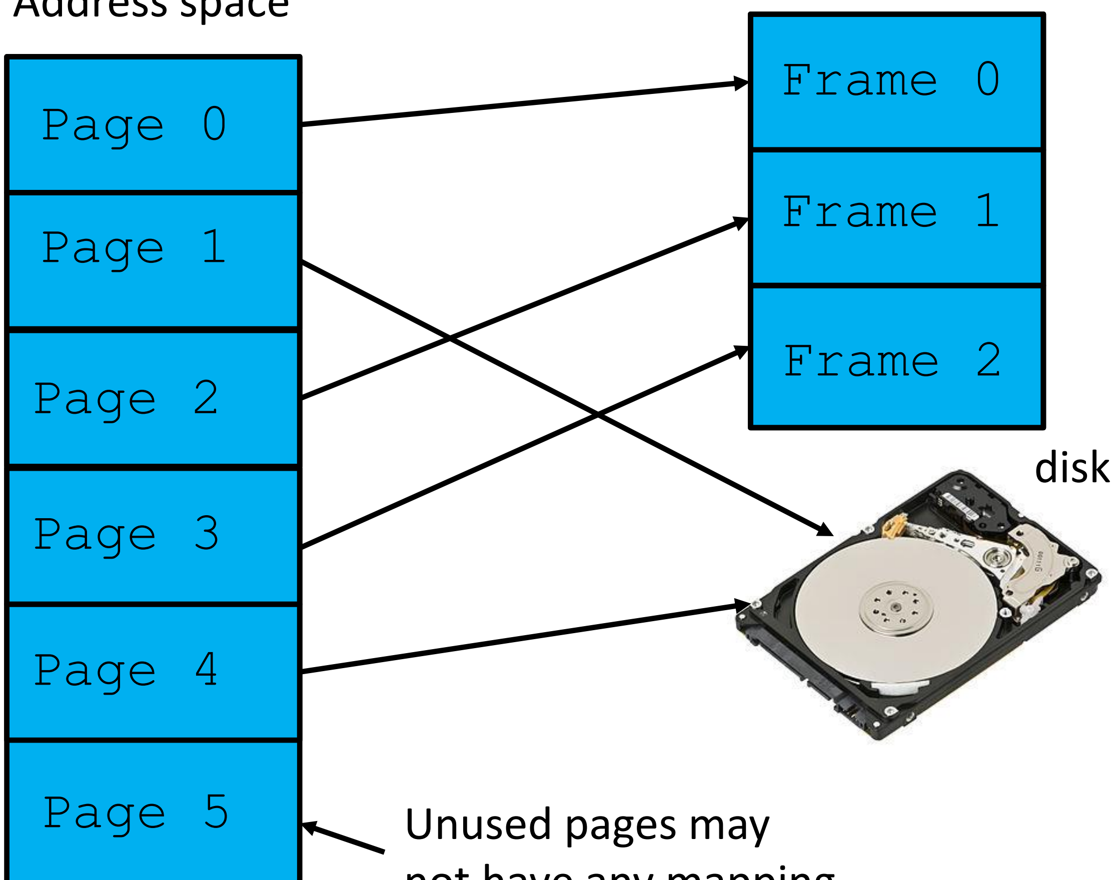
Pages currently in use are stored in physical memory (RAM)

Pages not currently in use (but were used in the past) are stored on disk

Poll Everywhere

pollev.com/tqm

- What happens if we need to load in page 1 and physical memory is full?



Pages currently in use are stored in physical memory (RAM)

Pages not currently in use (but were used in the past) are stored on disk

Unused pages may not have any mapping

Lecture Outline

- Problems with old memory model
- Virtual Memory High Level
- **Address Translation**

Aside: Bits

- We represent data on the computer in binary representation (base 2)
- A bit is a single “digit” in a binary representation.
- A bit is either a 0 or a 1

- In decimal -> 243
- In binary -> 0b11110011

Hexadecimal

- Base 16 representation of numbers
- Allows us to represent binary with fewer characters
  - 0b11110011 == 0xF3
    ^ binary ^ hex

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

Poll Everywhere

pollev.com/tqm

- A page is typically 4 KiB -> 2^{12} -> 4096 bytes

- If physical memory is 32 KiB, how many page frames are there?

  A. 5 B. 4 C. 32 D. 8 E. We're lost…

- If addressable memory for a single process consists of 64 KiB bytes, how many pages are there for one process?

  A. 64 B. 16 C. 20 D. 6 E. We're lost…

- If there is one page table per process, how many entries should there be in a single page table?

  A. 6 B. 8 C. 16 D. 5 E. We're lost…

Addresses

- Virtual Address:
  - Used to refer to a location in a virtual address space.
  - Generated by the CPU and used by our programs
- Physical Address
  - Refers to a location on physical memory
  - Virtual addresses are converted to physical addresses

Page Offset

- This idea of Virtual Memory abstracts things on the level of Pages (4096 bytes == 2^{12} bytes)
- On almost every machine, memory is ***byte-addressable*** meaning that each byte in memory has its own address
- How many different addresses correspond to the same page? 4096 addresses to a single page
- How many bits are needed in an address to specify where in the page the address is referring to?

  12 bits

Poll Everywhere

pollev.com/tqm

- If there are 16 pages, how many bits would you need to represent the number of pages?
- If there are 8 pages frames, how many bits would we need to represent the number of page frames?

	Page bits	Frame bits
A.	4	2
B.	4	3
C.	3	3
D.	5	3

E. We're lost…

Steps For Translation

- Derive the virtual page number from a virtual address
- Look up the virtual page number in the page table
  - Handle the case where the virtual page doesn't correspond to a physical page frame
- Construct the physical address

Address Translation: Virtual Page Number

- A virtual address is composed of two parts relevant for translating:

Virtual Page Number	Page Offset

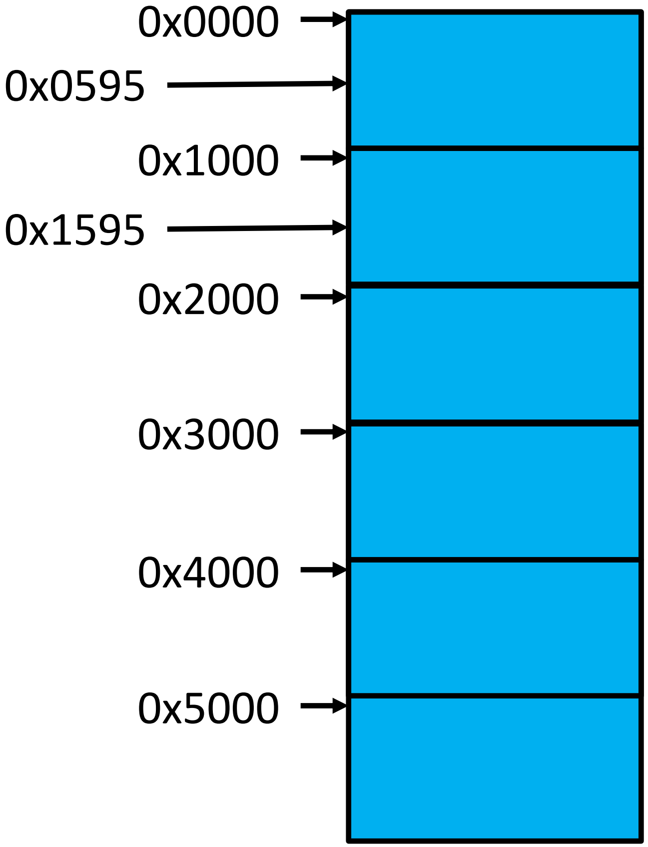
  - Virtual Page Number length = bits to represent number of pages
  - Page offset length = bits to represent number of bytes in a page

- The virtual page number determines which page we want to access

- The page offset determines which location within a page we want to access.
  - Remember that a page is many bytes (~4KiB -> 4096 bytes)

Virtual Address High Level View

- High level view:
  - Each page starts at a multiple of 4096 (0X1000)
  - If we take an address and add 4096 (0x1000) we get the same offset but into the next page



Address Translation: Virtual Page Number

- A virtual address is composed of two parts relevant for translating:

Virtual Page Number	Page Offset

  - Virtual Page Number length = bits to represent number of pages
  - Page offset length = bits to represent number of bytes in a page

pollev.com/tqm

- Example address: 0x1234
  - What is the page number?
  - What is the offset?
  - Reminder: there are 16 virtual pages, and a page is 4096 bytes

Address Translation: Lookup & Combining

- Once we have the page number, we can look up in our page table to find the corresponding physical page number.
  - For now, we will assume there is an associate page frame

Virtual page #	Valid	Physical Page Number
0x0	0	null
0x1	1	0x5
…	…	…

- With the physical page number, combine it with the page offset to get the physical address

Physical Page Number	Page Offset

  - Since we only need 3 bits to represent the physical page number, we only 15 bits for the address (as opposed to 16).
  - In our example, with 0x1234, our physical address is 0x5234

Translation Done!

Page Faults

- What if we accessed a page whose page frame was not in physical memory?

Virtual page #	Valid	Physical Page Number
0x0	0	null
0x1	1	0x0
0x2	1	0x5
0x3	0	Disk
…	…	…

- In this example, Virtual page 0x0 and 0x3

Page Faults

Virtual page #	Valid	Physical Page Number
0x0	0	null
0x1	1	0x0
0x2	1	0x5
0x3	0	Disk
…	…	…

- In this example, Virtual page 0x3, whose frame is on disk (page 0x3 handled before, but was evicted at some point)
  - MMU fetches the page from disk
  - Evicts an old page from physical memory if necessary
    - Uses LRU or some page replacement algorithm
    - Writes the contents of the evicted page back to disk
  - Store the previously fetched page to physical memory

Page Faults

Virtual page #	Valid	Physical Page Number
0x0	0	null
0x1	1	0x0
0x2	1	0x5
0x3	0	Disk
…	…	…

- In this example, Virtual page 0x0, which has never been accessed before
  - Evict an old page if necessary
  - Claim an empty frame and use it as the frame for our virtual page