Virtual Memory Details

Computer Systems Programming, Spring 2024

Instructor: Travis McGaha

TAs:

Ash Fujiyama	Lang Qin
CV Kunjeti	Sean Chuang
Felix Sun	Serena Chen
Heyi Liu	Yuna Shao
Kevin Bernat	

Poll Everywhere

pollev.com/tqm

- Why do we store data in physical memory? Why don't we store all of the pages and data in disk?

Poll Everywhere

pollev.com/tqm

- On a 32-bit machine, one address space is 4 GB. If we have 8 GB of RAM installed, is it possible to "run out" of physical memory? How?

Poll Everywhere

pollev.com/tqm

- Any questions, comments or concerns so far about **anything?**

Upcoming Due Dates

- HW2 (Threads)
  - Due a week from Thursday

- Midterm
  - Exams still being graded
  - A few makeups still happening

Lecture Outline

- **Review**
- Virtual Memory Details

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 of 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)

Definitions

Sometimes called "virtual memory" or "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 NOT EXIST ON REAL HARDWARE

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

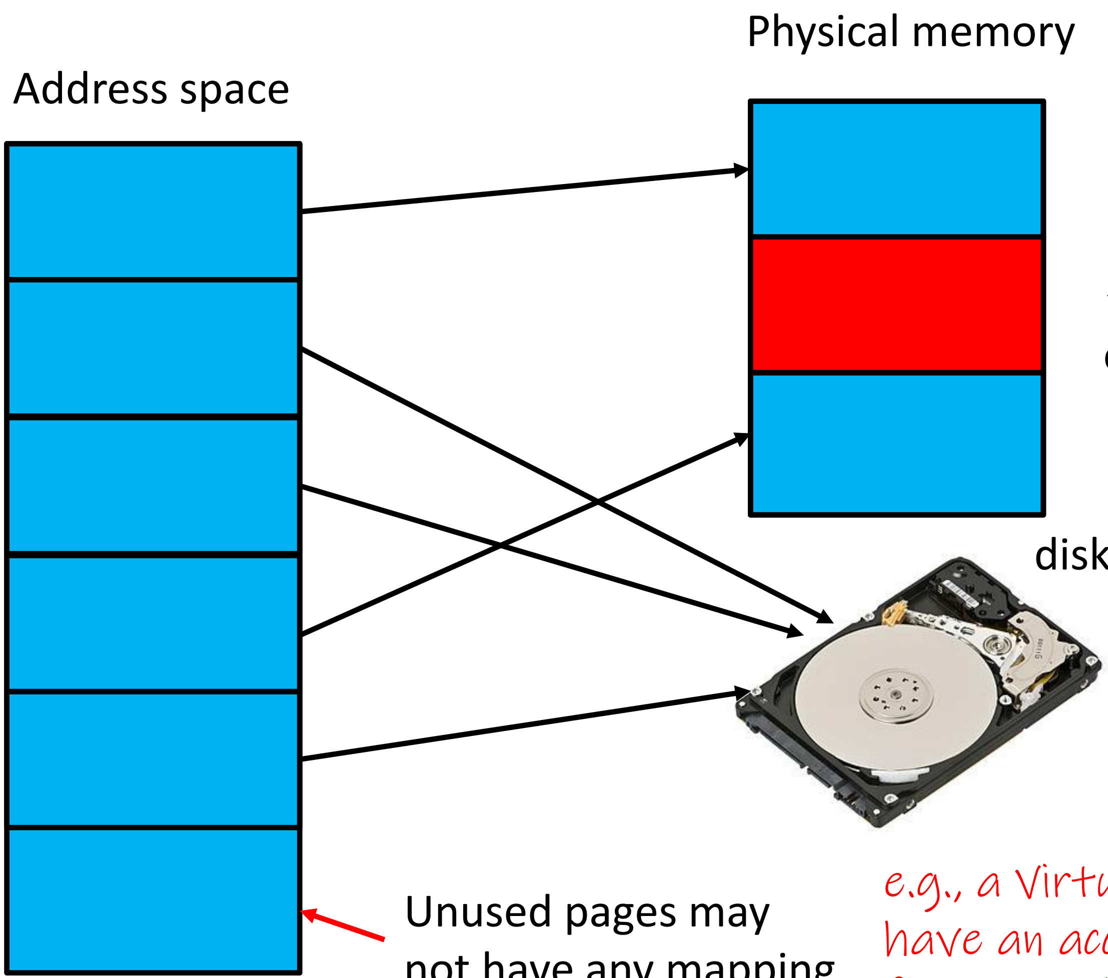
Adding Addressable Memory + Physical Memory doesn't make sense

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

Pages

Pages are fixed size chunks ~4KB
(4 * 1024 = 4096 bytes)

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



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

← Ram may contain pages from other active processes

Pages on physical storage are called a "Page Frame"

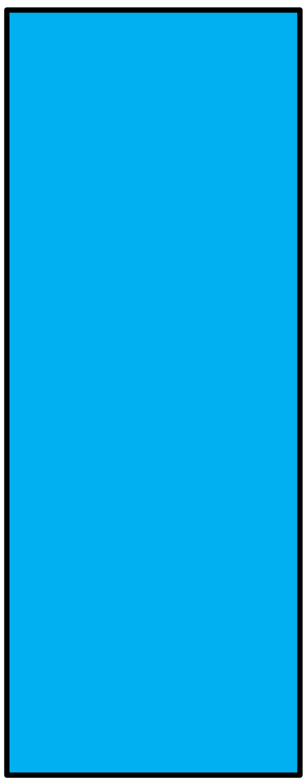
Pages not currently in use are stored on disk

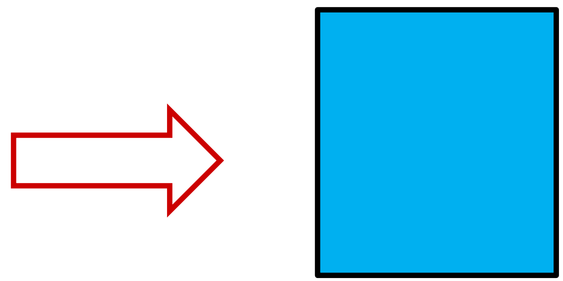
e.g., a Virtual page may not have an accompanying page frame until the page is used

Unused Pages

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

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





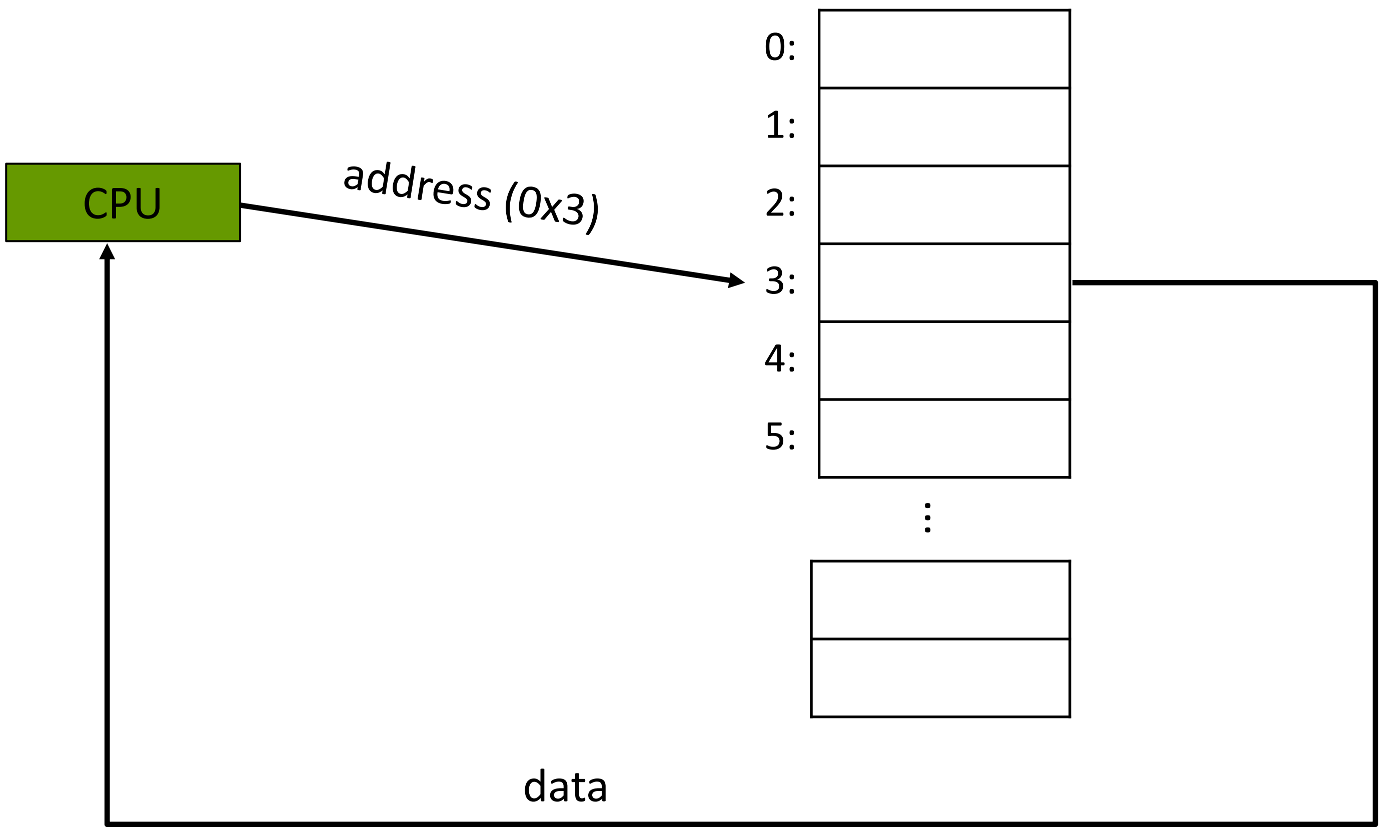
(Not to scale; physical memory is smaller than the period at the end of the sentence compared to the virtual address space.)

As I write this slide, PowerPoint is using 212.7MB which is: 223,032,115 Bytes (2.230×10^{7})

Some programs don't need 2^{64} bytes, so several pages may never be used

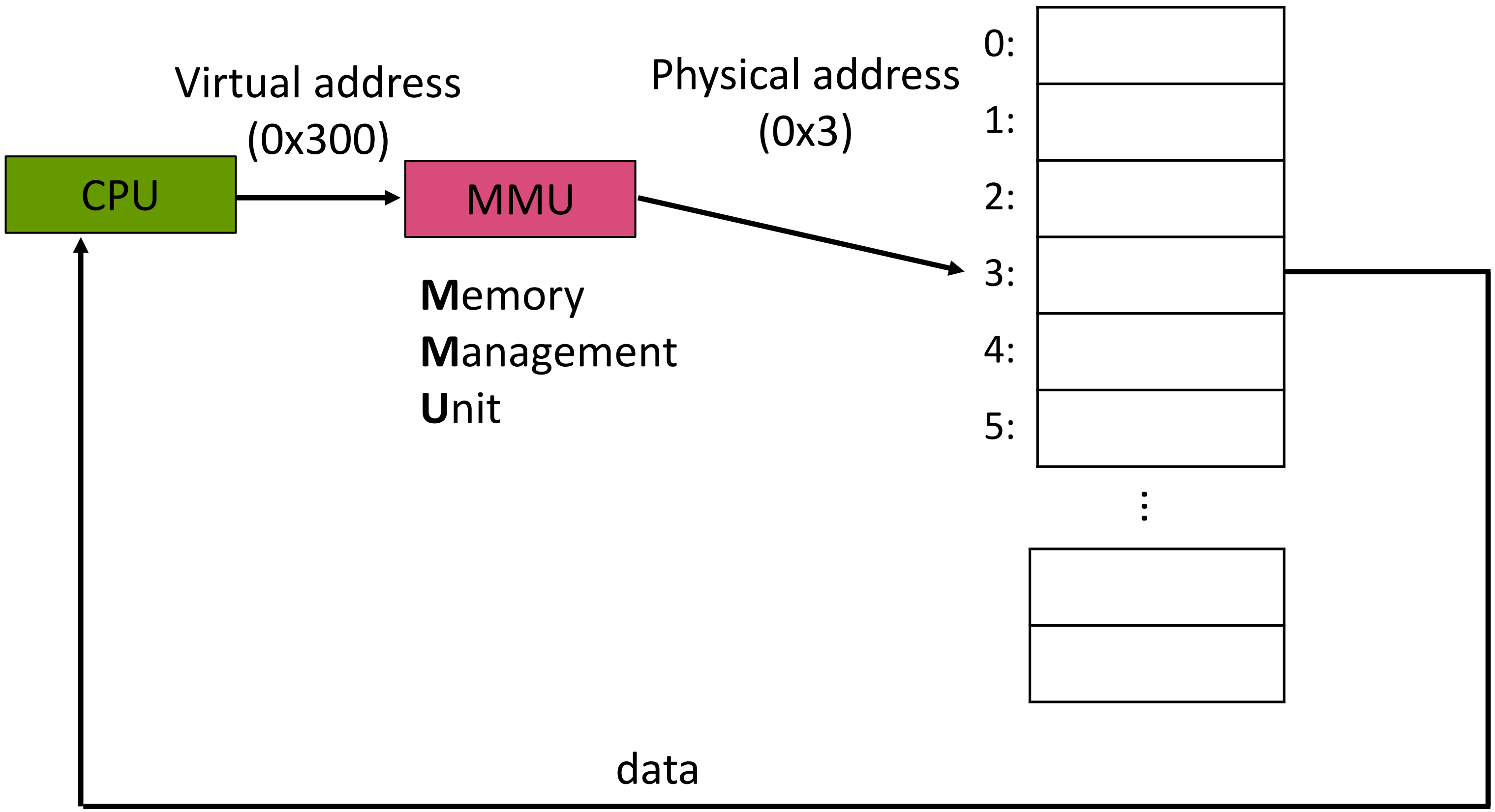
This doesn't work anymore

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



Virtual Address Translation

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



Page Tables

More details about translation on Wednesday

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

- There is **<u>one page table per process</u>**, managed by the MMU. Has one entry per virtual page.

Virtual page #	Valid	Physical Page Number
0	0	null
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 Replacement

- We don’t have space to store all active pages in physical memory.

- If we need to load in a page from disk, how do we decide which page in physical memory to “evict”
  - Have a page replacement algorithm (e.g. LRU)

- Goal: Minimize the number of times we have to go to disk. It takes a while to go to disk.

Lecture Outline

- Review
- **Virtual Memory Details**

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 -> 13
  - $(1 * 10^1) + (3 * 10^0)$
- In binary -> 0b1101
  - $(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0)$ -> 8 + 4 + 0 + 1 -> 13

- 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

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)

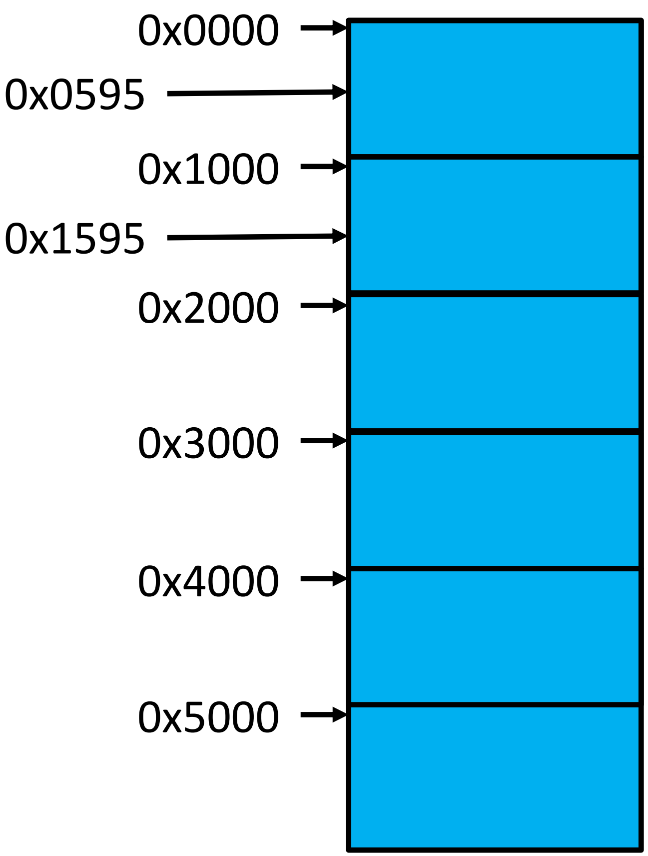
Page Offset

- 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

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: 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 `0001 0010 0011 0100`
  - What is the page number? `0001 -> 0x1`
  - What is the offset? `0010 0011 0100 -> 0x234`
  - 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

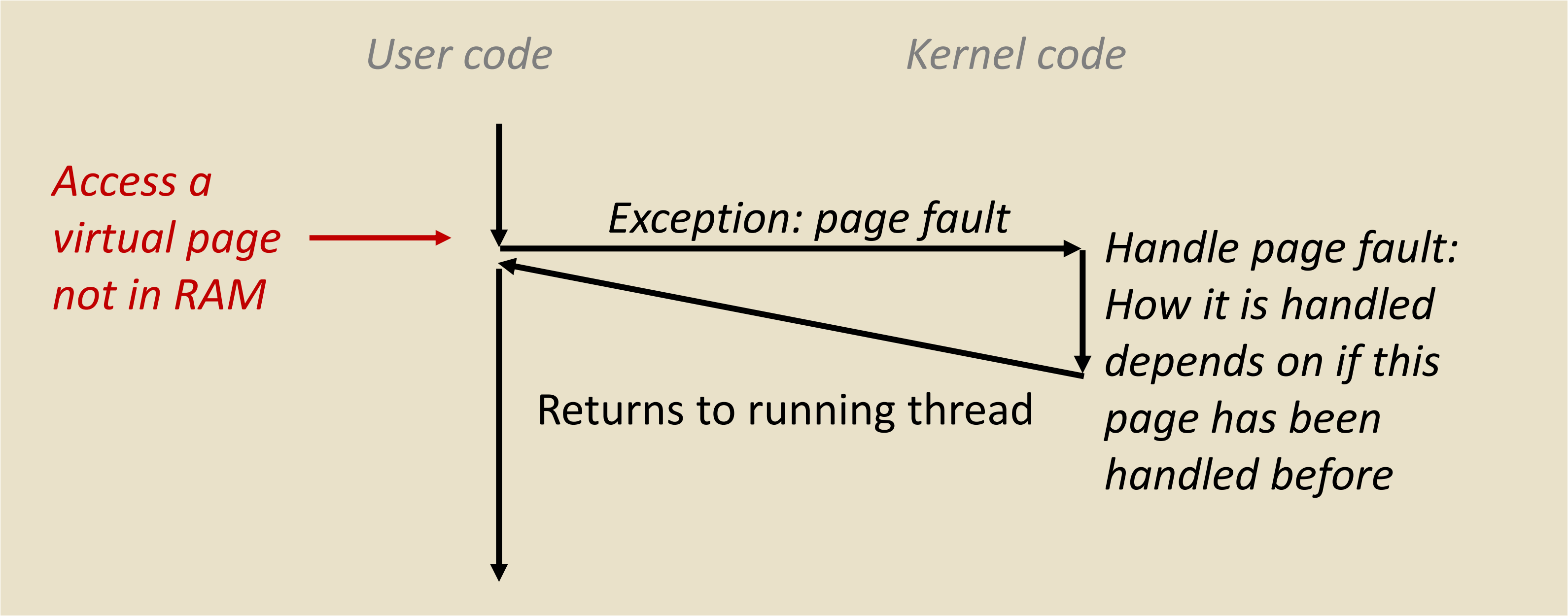
- What if we accessed a page whose page frame was not in physical memory?

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 Fault Exception

- An *exception* is a transfer of control to the OS *kernel* in response to some *event*
- In this case, writing to a memory location that is not in physical memory currently



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

Poll Everywhere

pollev.com/tqm

- There are 16 pages, 4 frames, and after starting from an empty page table, the following memory accesses are made in the listed order:
  - 0x4321, 0x1FEE, 0x1FEF, 0x2FFF, 0x3000, 0x400F
- If we are using Least Recently Used (LRU) for our replacement policy, what page would be evicted if we access memory address 0x5234

A. 0x4

B. 0x3

C. 0x2

D. 0x1

E. Nothing is evicted

Details left out

- Virtual Memory
  - COW Fork (Copy On Write)
  - Details about shared process memory
  - Transition Lookaside Buffers (TLB)

- Memory Hierarchy
  - Cache Associativity
  - Writing Policies
  - DRAM vs SRAM

- A bunch of details that would be system-specific