

# CSE 120

# Operating Systems Principles

Spring 2025

Lecture 19: Final Review

Amy Ousterhout

# Related Courses

---

- If you enjoyed CSE 120, consider taking other systems courses:
- CSE 123: Computer Networks (Winter '26)
- CSE 124: Networked Services
- CSE 125: Software System Design and Implementation (Spring '26)
- CSE 127: Computer Security (Fall '25, Winter '26, Spring '26)
- CSE 190: Data Center Systems (Winter '26)
- CSE 221: Graduate Operating Systems (Fall '25, Winter '26)

# Tutoring

- If you enjoyed CSE 120 and enjoy helping other students, consider applying to tutor!
  - You don't need to get an A to be a great tutor
- Geoff Voelker teaches 120 in the fall
  - Same Nachos projects
  - Very similar lecture content
- Applications for Fall open June 9th

Join a supportive community

Apply now to become a **CSE Tutor**

<https://go.ucsd.edu/4ifkGzh>

Industry connections & networking opportunities

Become a role-model & inspire others like you!

Improve your understanding and sharpen your own skills as you teach

Help your peers & strengthen your community

Boost your resume with hands-on teaching experience

Make money \$23.73/hr

All majors welcome to apply!

**Am I qualified?**  
Almost certainly! You don't need to have gotten an A to tutor. Often people who struggled a little can relate to and support students better.

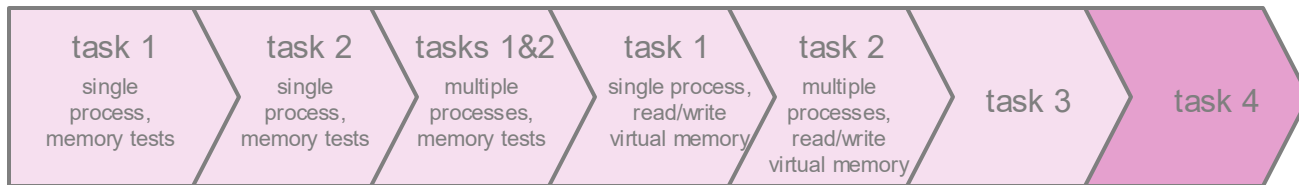
Participation Quarter	Application Opens	Priority Deadline**
Summer 2025	3/31/25	4/11/25
Fall 2025	6/9/25	6/27/25

\*\*Applications still accepted after the deadline!

# Upcoming Tasks

---

- Discussion tomorrow: Q&A
  - Ask about the projects, homework, concepts, etc.
- Project 3
  - Due Friday June 6<sup>th</sup> at 11:59 pm, no option to submit late



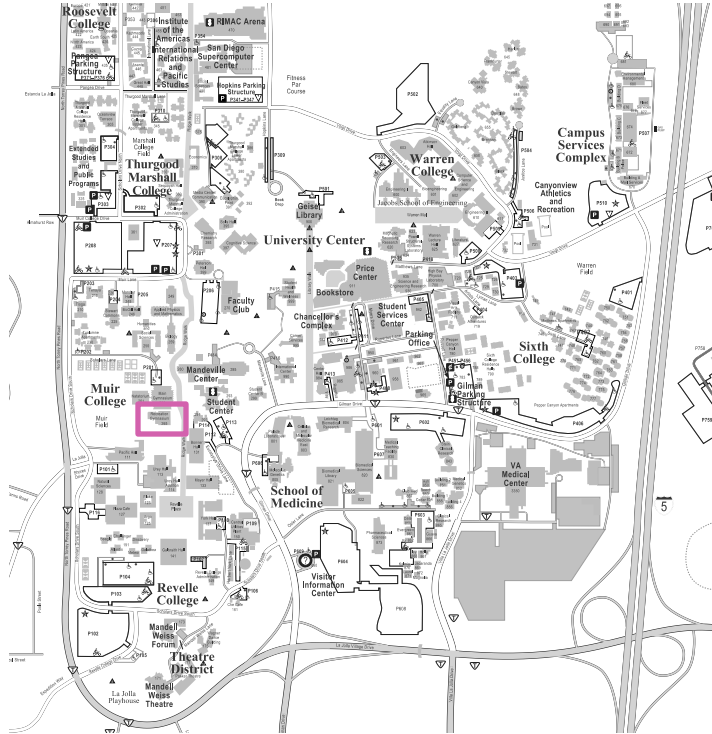
- Course Evaluations: <https://academicaffairs.ucsd.edu/Modules/Evals>
  - Due Saturday, June 7<sup>th</sup>, at 8:00 am
- Final Exam
  - Monday, June 9<sup>th</sup>, 3:00-6:00 pm in the REC gym

# Today's Outline

---

- Final exam logistics
- Final exam material
- Practice problems

# Final Exam Location: REC Gym (not York!)



From Ridge Walk



# Final Exam Logistics

---

- Monday June 9<sup>th</sup>, REC Gym, 3:00-6:00 pm
- We will assign seats
  - Seat assignments will be posted on Piazza and in the exam room
- You may bring **one 8.5"x11" double-sided sheet of notes** to the exam
  - Handwritten or printed
- **Bring your ID**
  - Place it on the table in front of you during the exam
  - Show it to a proctor when you hand in your exam
- **Obligatory: complete your exam yourself without assistance from others**
  - Sign an agreement on the first page indicating this

# Final Exam Content

---

- The final exam is cumulative (covers the entire quarter)
  - But, it will focus on the content covered since the midterm
- Covers:
  - Lectures
  - Homework
  - Nachos programming projects

# Types of Questions

---

- True or False
- Multiple choice
- Short answer
- Multi-part problems
  - May include reading and writing code
  - See questions 4, 5, 6, and 8 in the sample final for examples (and questions from the midterm)

# How to Prepare

---

- Practice, practice, practice!
- We will work through practice questions today
- Make sure you can solve problems on your own:
  - Homework questions
    - » All solutions are now available on the course website
  - In-class poll questions
  - Sample exams
  - Synchronization practice
- Want even more practice?
  - Search online for other undergrad OS courses that post problems or exams
  - Pair up with a peer and make practice questions for each other

# Today's Outline

---

- Final exam logistics
- Final exam material
  - Lecture material since the midterm
- Practice problems

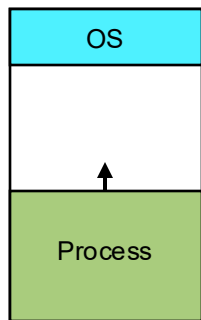
# Memory Management

---

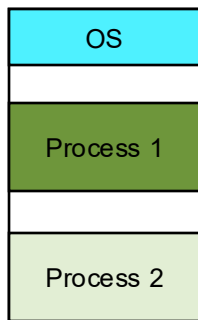
- Understand why memory management is useful
  - What are the benefits of virtual memory?
- Describe the **mechanisms** for implementing memory management
  - Physical and virtual addressing
  - Static relocation, base and bound, segmentation, paging
  - Page tables, TLB
- Understand the **policies** related to memory management
  - Page replacement
- Describe the overheads related to providing memory management

# Virtualizing Memory

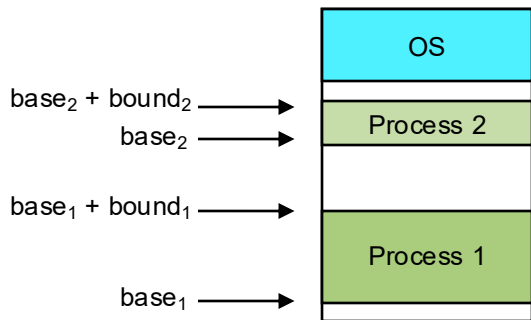
- Understand the difference between a physical and a virtual address
- Multiple approaches to memory management
  - Describe them
  - Compare/contrast them, explain their pros/cons



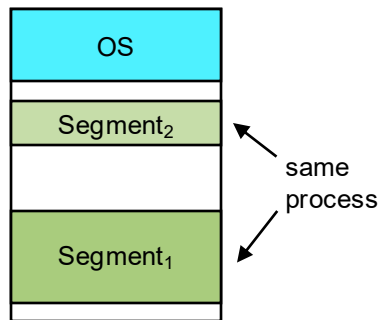
single tasking



static relocation



base and bound



segmentation

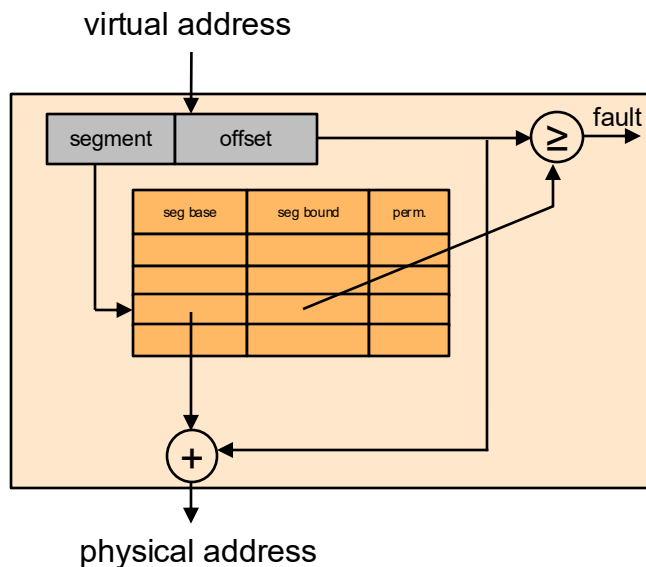
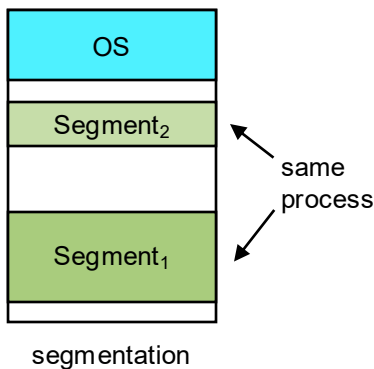
# Virtualizing Memory

---

- Internal and external fragmentation
  - Differentiate between them
  - Identify whether each can occur or not
- Understand what kinds of faults can occur due to memory management
  - Page faults
  - Protection faults

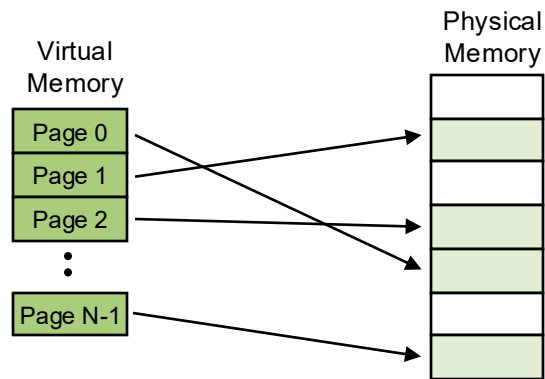
# Segmentation

- Understand what segmentation is
- Compare/contrast it with base and bound and paging
- Describe its pros/cons relative to base and bound and paging
- Be able to translate addresses with segmentation
  - What is a segment table and how is it used?



# Paging

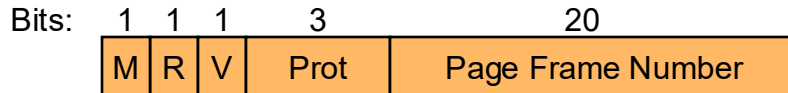
- Understand what paging is
- Compare/contrast paging with other approaches
- Understand paging mechanisms and how to use them in address translation
  - Page tables
  - Page table entries (PTEs)
  - Virtual page number (VPN)
  - Physical page number (PPN)/page frame number (PFN)
  - Offset



# Page Table Entries

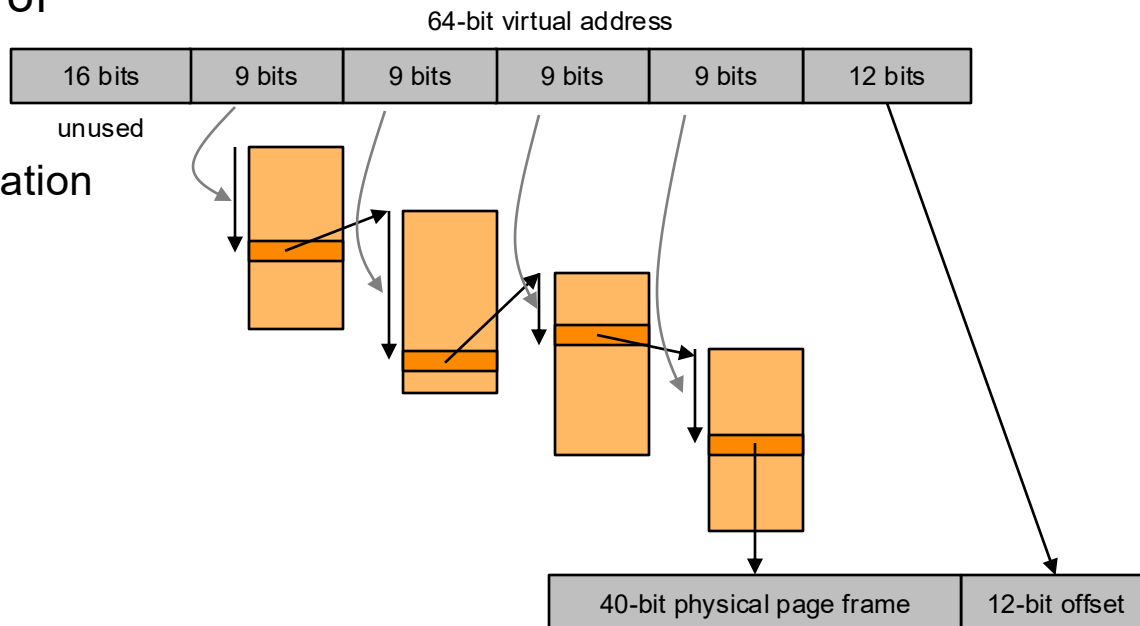
---

- Understand the components of a page table entry
  - Modify bit (dirty)
  - Reference bit (used)
  - Valid bit
  - Protection bits
  - Page frame number (PFN)



# Multi-Level Page Tables

- Understand the overheads of page tables
  - Space for storing them
  - Time to use them for translation
- Describe multi-level page tables and their pros/cons



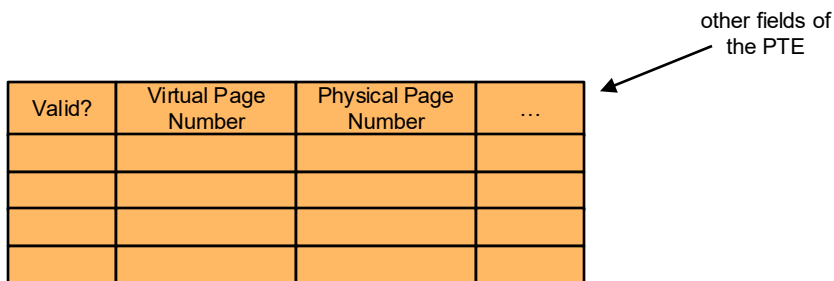
# Translation Lookaside Buffers (TLBs)

---

- Understand what problem TLBs solve
- Explain what information TLBs store and how they are used and managed
  - How are they involved in address translation?
  - What happens on a TLB miss?
- Understand why TLBs are effective

Valid?	Virtual Page Number	Physical Page Number	...

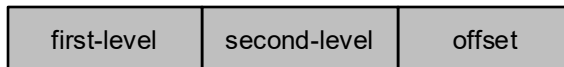
other fields of the PTE

A diagram showing a table with four columns: 'Valid?', 'Virtual Page Number', 'Physical Page Number', and '...'. The table has five rows. An arrow points from the text 'other fields of the PTE' to the '...' column, indicating that the table represents a TLB entry that stores virtual-to-physical page mappings and validity information, with other fields from the original Page Table Entry (PTE) also being stored.

# Address Translation with Paging

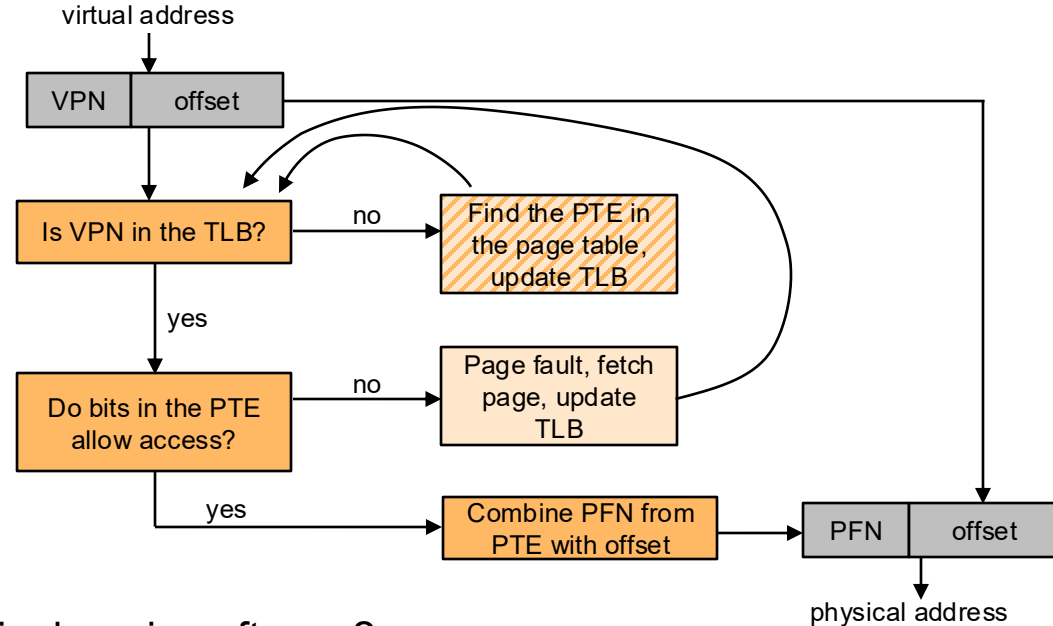
---

- Identify what different bits in a virtual address represent
  - Index into first-level page table, second-level, etc.
  - Offset within a page
- Be able to translate from virtual to physical addresses
- Explain the effect of changing virtual memory parameters. For example:
  - The page size
  - Single-level vs. multi-level page tables



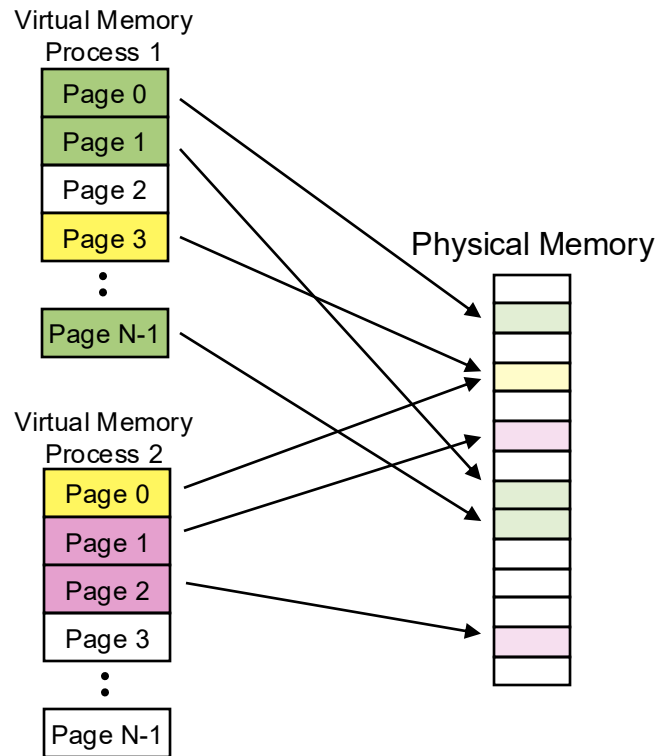
# Page Faults

- Describe what a page fault is
  - What causes one?
  - How does the OS handle them?
  - How are they used to implement demand-paged virtual memory?
- Understand the sequence of steps for translating a virtual address to a physical address
  - What is done in hardware, what is done in software?



# Advanced Virtual Memory Topics

- Describe what shared memory is
  - How do OSes implement it?
- Describe what copy-on-write is
  - How do OSes implement it?
  - How is it used in process creation?
- Describe what mapped files are
  - How do OSes implement them?
- Understand the benefits of each



# Page Replacement

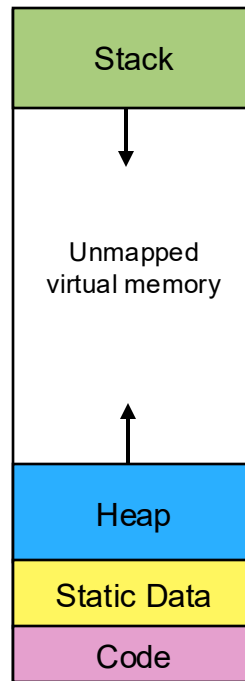
---

- Understand the purpose of page replacement algorithms and when they are used
- Understand what application behavior page replacement tries to exploit
- Describe how each of the following work
  - Belady's MIN, FIFO, LRU, Clock
- Understand Belady's anomaly, working sets, and thrashing
- Spatial and temporal locality
  - Differentiate between them
  - Determine if each is present in a given workload or setting

# Virtual Memory Allocation

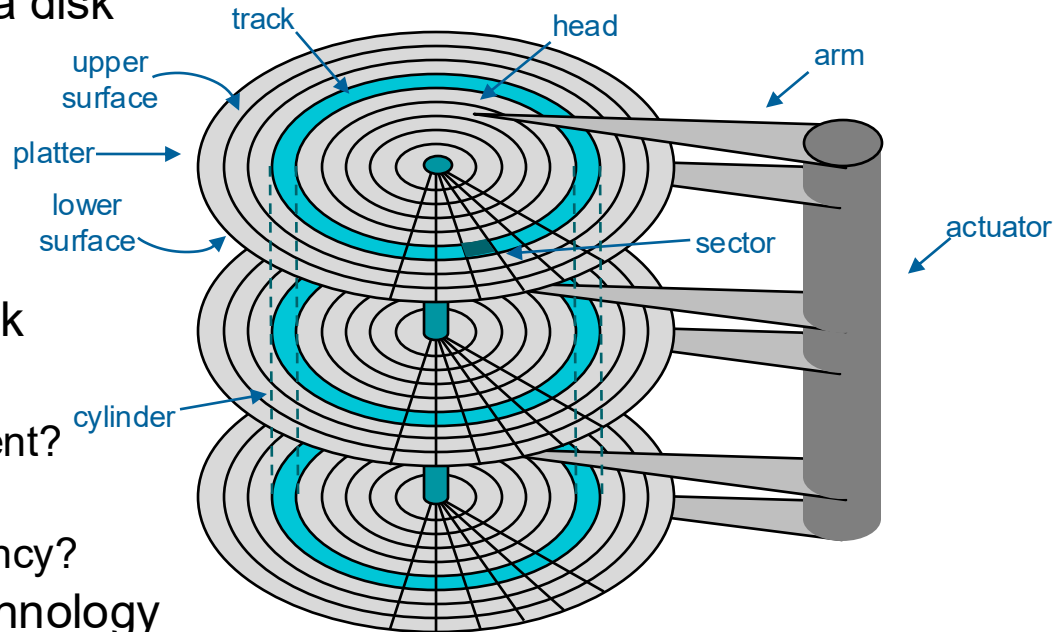
---

- Understand how OSes manage and grow the stack
- Understand how OSes manage and grow the heap



# Disk Components and Latency

- Understand the components of a disk
  - Sectors
  - Tracks
  - Heads
  - etc.
- Describe what contributes to disk access latency
  - How significant is each component?
  - How does disk access latency compare to memory access latency?
- Predict how changes in disk technology would impact performance



# Disks

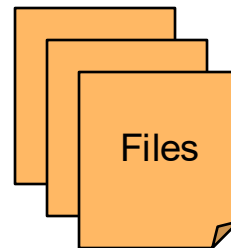
---

- Compare/contrast the block interface with older disk interfaces
- Understand how disk scheduling policies work
  - FIFO, shortest seek time first, elevator (SCAN)
- Compare/contrast different disk scheduling policies
- Understand how RAID uses striping, mirroring, and parity disks
  - What metrics does RAID improve?

# File Systems and their Abstractions

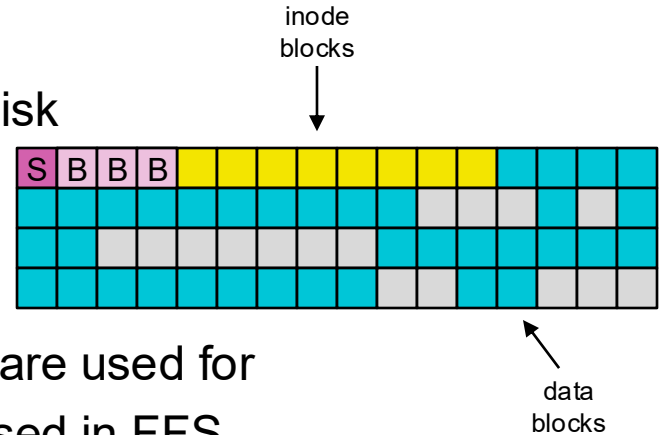
---

- Understand what a file system is
  - What hardware resource(s) does it manage?
  - What abstractions does it provide to applications and why are they useful?
- Describe the file abstraction
  - What operations are supported?
  - What characteristics do they have?
  - What are file access methods?
- Describe the directory abstraction
  - What are they used for?
  - What is a directory entry?



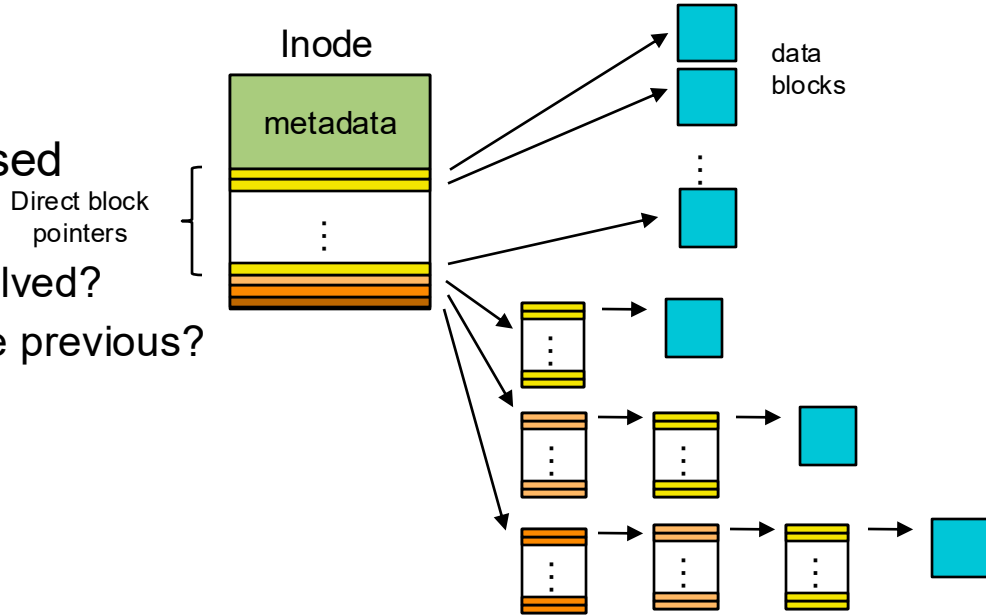
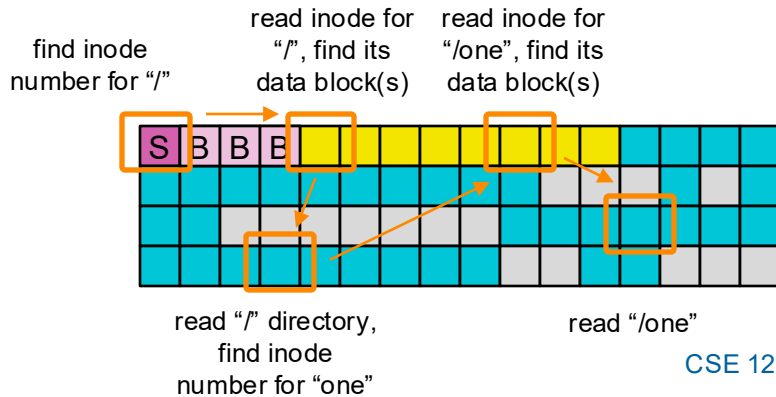
# File System Layout

- Understand how file systems manage information on disk
  - What are the advantages of using disk blocks?
  - What kind of fragmentation do they suffer from?
- Different strategies for laying out information on disk
  - Contiguous, linked, indexed
  - Describe them
  - Compare/contrast them
- Understand what bitmap blocks and superblocks are used for
- Describe how cylinder groups/block groups are used in FFS



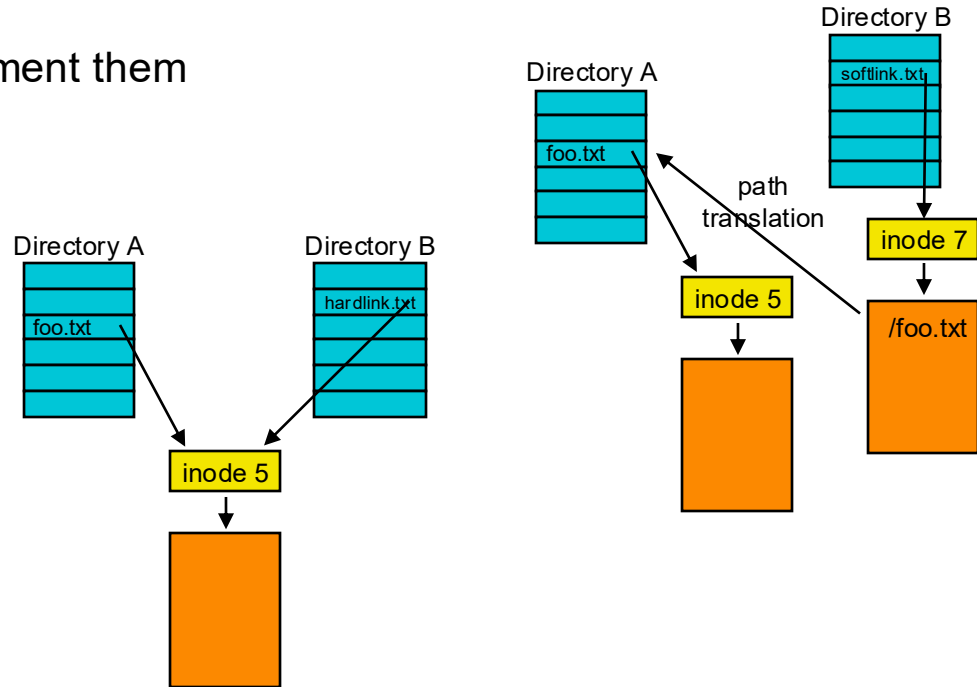
# Inodes and Path Translation

- Describe what an inode is
  - What information does it contain?
- Determine which blocks are accessed during path translation
  - How are inodes and directories involved?
  - How do you find each block from the previous?



# File Operations

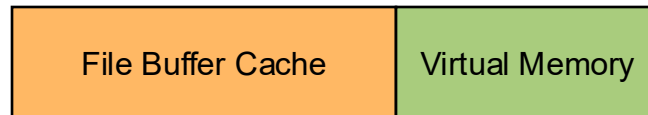
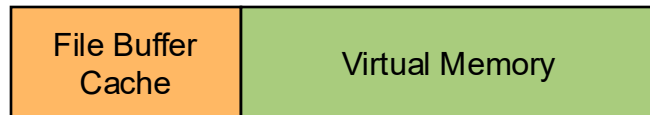
- Hard links and soft links
  - Understand how file systems implement them
  - Compare/contrast them
- Understand how create, delete, and rename work
  - Which blocks are modified during each?



# File Buffer Cache

---

- Understand what the file buffer cache is and its benefits
  - What is cached in the file buffer cache?
- Explain how caching reads and caching writes work
- Understand how physical memory is shared between the file buffer cache and virtual memory



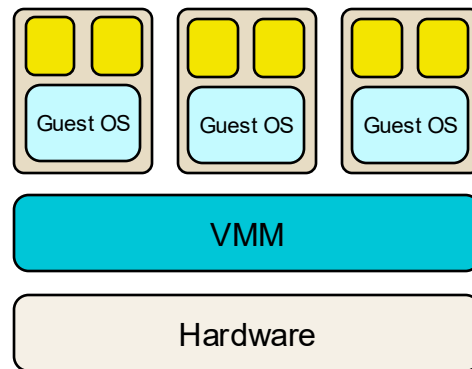
# Crash Consistency

---

- Understand why crash consistency is a problem
- Describe things that can go wrong with the file system when a crash occurs
  - Space leaks, accessing garbage data, inconsistencies, etc.
- Given a file system operation, assess what inconsistencies could arise due to a crash
- Describe techniques for keeping a file system consistent
  - File system checker (fsck)
  - Ordered writes
  - Journaling
- Compare/contrast these different approaches
- Determine how each could be used to prevent or recover from inconsistencies

# Virtual Machines

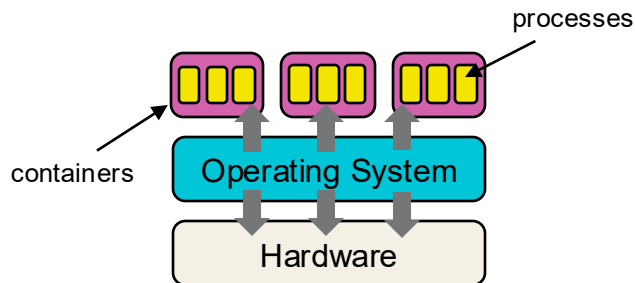
- Understand what a Virtual Machine Monitor (VMM) is
  - What abstraction do they provide to virtual machines (VMs)?
  - Why are virtual machines useful?
- Explain what trap and emulate is
  - How is it involved in different actions that a VM might take?
- Compare and contrast different abstractions for virtualization:
  - VMs
  - Containers
  - Processes



# Containers

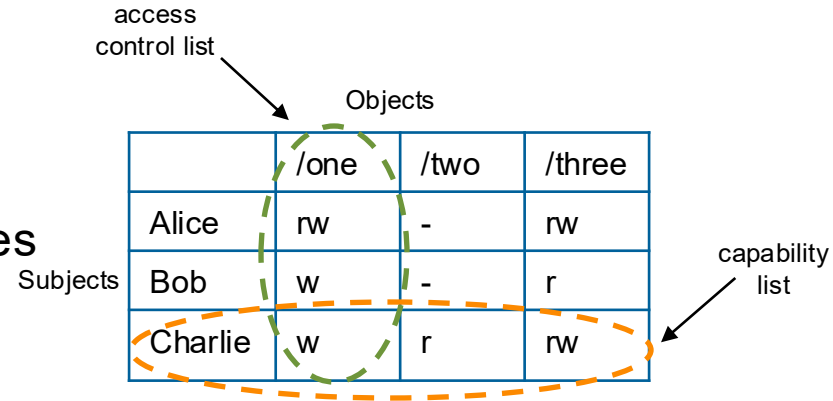
---

- Understand what abstraction containers provide
- Describe what namespaces and resource management are
  - How do containers use them?



# Protection

- Understand the principles of protection
- Understand how user identity is used in protection
- Describe access control lists and capabilities
  - How does each one organize authorization information?
  - Articulate their similarities/differences and pros/cons
  - Identify whether something is an access control list or a capability
- Understand how OSes implement protection for files and virtual memory
  - How do they use access control lists and capabilities?



# Today's Outline

---

- Final exam logistics
- Final exam material
- Practice problems
  - I'll give you 5-10 minutes to work on each one, then we'll discuss
  - Try them on your own first, then feel free to discuss with a neighbor
  - Feel free to work ahead (see slides on the course website)
  - I will walk around and answer questions

# File System Crash Consistency (Q)

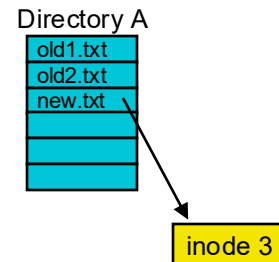
---

- Your goal is to implement a `creat` system call, which creates a new empty file:
  - `creat(char *name)`
- List the blocks that must be modified and written to disk during a call to `creat`
- Give two examples of what could go wrong after a crash (without any crash-consistency techniques)
- Suppose this file system uses ordered writes. In what order should the file system write the blocks to disk? Give one example.
- How could you solve this problem with journaling instead?

# File System Crash Consistency (A1)

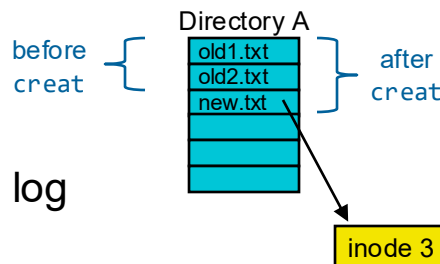
---

- `int creat(char *name)`
- List the blocks that must be modified and written to disk during a call to `creat`
  - Inode block containing the new inode, bitmap block for the inode
  - Data block for the directory, inode for the directory
- Give two examples of what could go wrong after a crash (without any crash-consistency techniques)
  - Space leak if only the bitmap block is written
  - Multiple files try to use the same inode if all blocks except the bitmap are written
  - Pointer to uninitialized inode if all blocks except the inode are written
  - Invalid directory entry if the directory's inode is written first



# File System Crash Consistency (A2)

- Suppose this file system uses ordered writes. In what order should the file system write the blocks to disk? Give one example.
  - Constraints:
    - » Write the inode bitmap before the inode is accessible
    - » Write the new inode block and the directory data block before the directory inode
  - Any ordering in which the directory inode is written last is safe
    - » Before then, the directory entry isn't valid
  - Run the file system checker to address the space leak
- How could you solve this problem with journaling?
  - Create a transaction containing those 4 blocks, write it to the log



# Segmentation and Paging (Q)

---

- Consider a virtual memory system that uses both segmentation and paging. The virtual address space is first divided into segments and then further divided into pages.
  - Virtual addresses are 32 bits and a page is 1024 ( $2^{10}$ ) bytes.
  - A page table must fit on one physical page, the size of a PTE is 4 bytes, and it has 12 bits of flags.
  - One process may have at most 64 ( $2^6$ ) segments.
- What is the maximum size of a virtual memory segment?
- How many pages are there per segment?
- How many levels of page tables are required for virtual address translation?
- Draw an address translation diagram showing the tables used and how the bits of the virtual and physical addresses are divided.

# Segmentation and Paging – Intuition, part 1

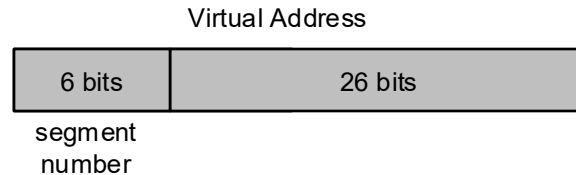
---

- We want to be able to use the entire virtual address space ( $2^{32}$  bytes)
- One way to approach this problem is to think about the **sizes** of different components.
  - We are given that the address space is  $2^{32}$  bytes, there are  $2^6$  segments, and a page is  $2^{10}$  bytes
- Another way to approach this problem is to think about the **bits in the virtual address**
  - From above we can see that there are 32 bits in the virtual address, 6 bits in the segment number, and 10 bits in the page offset
- We can use either of these approaches to answer the first 2 questions

# Segmentation and Paging (A1)

---

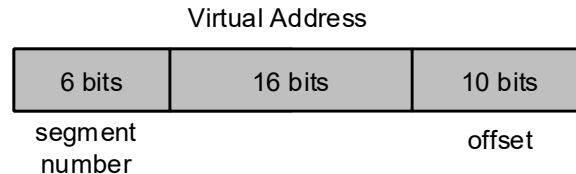
- Consider a virtual memory system that uses both segmentation and paging.
  - Virtual addresses are 32 bits and a page is 1024 ( $2^{10}$ ) bytes.
  - A page table must fit on one physical page, the size of a PTE is 4 bytes and it has 12 bits of flags.
  - One process may have at most 64 ( $2^6$ ) segments.
- What is the maximum size of a virtual memory segment?
  - $2^6$  segments so 6 bits for segment number
  - Leaves  $32 - 6 = 26$  bits for addressing within a segment
  - Segment size is  $2^{26}$  bytes



# Segmentation and Paging (A2)

---

- Consider a virtual memory system that uses both segmentation and paging.
  - Virtual addresses are 32 bits and a page is 1024 ( $2^{10}$ ) bytes.
  - A page table must fit on one physical page, the size of a PTE is 4 bytes and it has 12 bits of flags.
  - One process may have at most 64 ( $2^6$ ) segments.
- How many pages are there per segment?
  - Segment size is  $2^{26}$  bytes (previous slide)
  - Page size =  $2^{10}$  bytes
  - $2^{26}$  bytes per segment /  $2^{10}$  bytes per page =  $2^{16}$  pages per segment

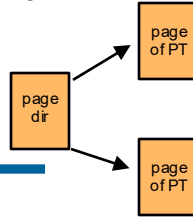


# Segmentation and Paging – Intuition, part 2

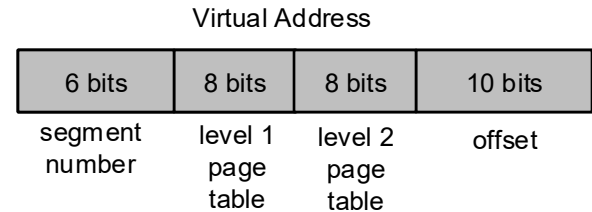
---

- Each page table page can only fit so many entries in it
- If one page table per segment does not have enough entries for all of the pages in a segment, then we will need multiple levels of page tables
- Think about:
  - How many entries can fit in one page table?
  - When we add an additional level of page tables, how does that change the number of pages that we can address?
    - » E.g., if page tables can fit 4 entries, with one level we can address 4 pages. With 2 levels we can address  $4 \times 4 = 16$  pages. With 3 levels we can address  $4 \times 4 \times 4 = 64$  pages...

# Segmentation and Paging (A3)

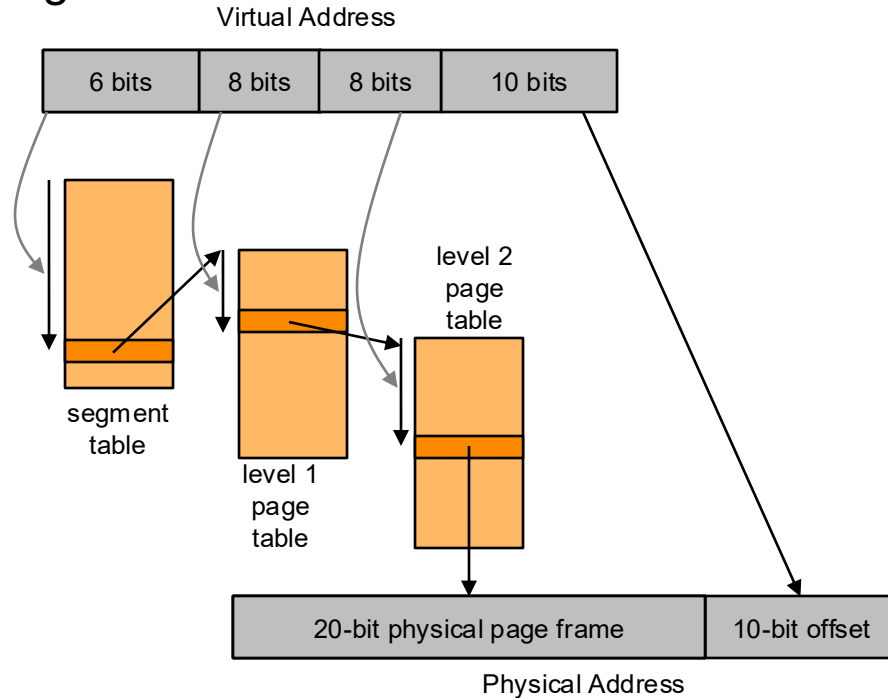


- Consider a virtual memory system that uses both segmentation and paging.
  - Virtual addresses are 32 bits and a page is 1024 ( $2^{10}$ ) bytes.
  - A page table must fit on one physical page, the size of a PTE is 4 bytes and it has 12 bits of flags.
  - One process may have at most 64 ( $2^6$ ) segments.
- How many levels of page tables are required for virtual address translation?
  - Page table size =  $2^{10}$  bytes and 4 bytes per PTE
  - $2^{10} / 4 = 2^8$  PTEs per page
  - 16 bits for addressing page tables / 8 bits per level
  - 2 levels of page tables



# Segmentation and Paging (A4)

- Draw an address translation diagram.
  - How many bits per PFN?
  - 32 bits per PTE
  - 12 bits of flags per PTE
  - $32 - 12 = 20$  bits for PFN



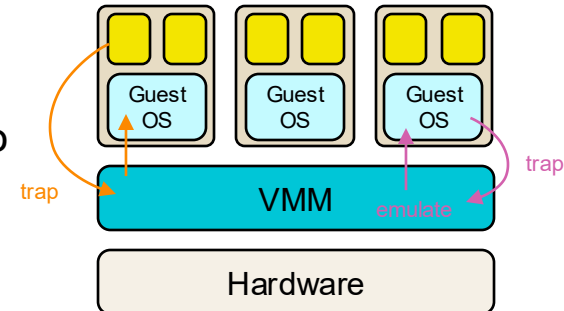
# Virtual Machines (Q)

---

- For these questions, assume a type 1 hypervisor, no binary translation, no paravirtualization, and no hardware support for virtualization.
- Consider a process running in a VM that tries to **dereference a NULL pointer**. What are the steps for handling this?
  
- Consider a Guest OS in a VM that tries to **write to the register containing a pointer to the current page table**. What are the steps for handling this?

# Virtual Machines (A)

- Consider a process running in a VM that tries to **dereference a NULL pointer**. What are the steps for handling this?
  - Virtual address 0 is not mapped, causing a page fault
  - CPU traps to the VMM
  - VMM calls into the Guest OS which kills the process
- Consider a Guest OS in a VM that tries to **write to the register containing a pointer to the current page table**. What are the steps for handling this?
  - Changing the page table pointer is a privileged operation
  - This causes a trap to the VMM
  - VMM emulates the behavior (chooses which page tables to use)
  - VMM returns to the Guest OS in the VM



# Page Replacement (Q)

---

- Consider a machine with 8 GB of memory. The OS's replacement policy is: when a page needs to be evicted, evict the page that has been in memory the longest. The owner notices that some workloads incur a lot of page faults and installs 2 GB more memory. Answer true/false for the following:
- There are workloads for which the extra memory will *decrease* the number of page faults
- There are workloads for which the extra memory will *have no effect on* the number of page faults
- There are workloads for which the extra memory will *increase* the number of page faults

# Page Replacement (A)

---

- There are workloads for which the extra memory will *decrease* the number of page faults
  - True – the working set might fit into 10 GB but not 8 GB
- There are workloads for which the extra memory will *have no effect on* the number of page faults
  - True – the workload might loop through a large amount of memory (e.g., 12 GB) such that every access incurs a page fault
- There are workloads for which the extra memory will *increase* the number of page faults
  - True – this is Belady's anomaly, where for some access patterns and replacement policies (including FIFO), increasing the size of a cache can increase the number of cache misses

# Memory Management Benefits (Q & A)

---

- What is the main benefit (or benefits) of each technique?
- Multi-level page tables
  - Reduce memory used by page tables
- Huge pages
  - Reduce memory used by page tables, speed up address translations
- Effective page replacement algorithms
  - Reduce page faults
- TLBs
  - Speed up address translations
- Copy on write
  - Reduce memory usage, reduce CPU time spent copying

Hint: reduce page faults,  
reduce memory usage, etc.

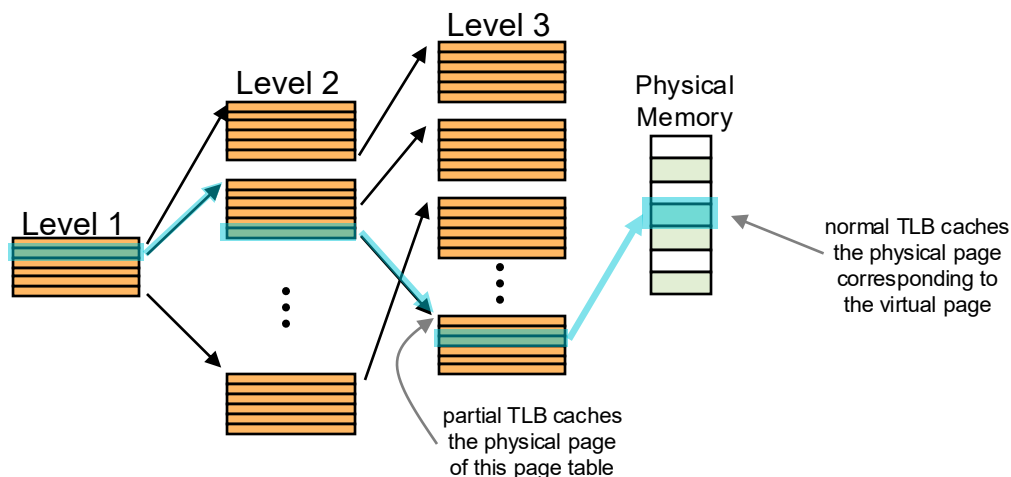
# TLBs and Multi-level Page Tables (Q)

---

- Your friend proposes a new hardware structure that they call a **partial TLB**. A partial TLB maps virtual pages to the physical page of their final page map (bypassing intermediate levels of page tables). It has the same number of entries as a regular TLB.
- Describe a workload where this would perform worse than a regular TLB.
- Describe a workload where this would perform better than a regular TLB.
- Do you think partial TLBs are a good approach overall? Why/why not?

# TLBs and Multi-level Page Tables (A1)

- A normal TLB maps from a virtual page number to a physical page number
- The proposed partial TLB maps from a virtual page number to the physical page number of the last-level page table
  - Address translation still requires one memory access even when you hit in the partial TLB



# TLBs and Multi-level Page Tables (A2)

---

- Describe a workload where this would perform worse than a regular TLB.
  - A workload where the number of pages in the working set is  $\leq$  the number of entries in a regular TLB
- Describe a workload where this would perform better than a regular TLB.
  - A workload where the number of pages in the working set is  $>$  the number of entries in a regular TLB but the number of last-level page tables is  $\leq$  the number of entries in a partial TLB (some pages use the same page table)
- Do you think partial TLBs are a good approach overall? Why/why not?
  - No, TLBs already have high hit rates and this approach would always add an extra memory access to translation

# Protection (Q & A)

---

- Explain what a capability is and give an example, either from operating systems or from real life.
  - A capability specifies what actions a subject is allowed to perform on an object.
  - Examples: file descriptor, PTE, physical key
- Explain what an access control list is and give an example, either from operating systems or from real life.
  - An access control list specifies which users are allowed to perform what actions on a given object
  - Examples: UNIX's permissions on files, list of people allowed to enter an event, list of users who are allowed to access a shared Google doc