CS 153
Design of Operating Systems

Fall 20

Final Review

Objectives of this class

- In this course, we will study typical problems that an OS to address and the corresponding solutions
  - Focus on **concepts** rather than particular OS
  - Specific OS for examples
- Practice your engineering skills
  - Abstraction and implementation
  - The projects are very close to real projects in industry
- Develop an understanding of how OS and hardware impacts application performance and reliability

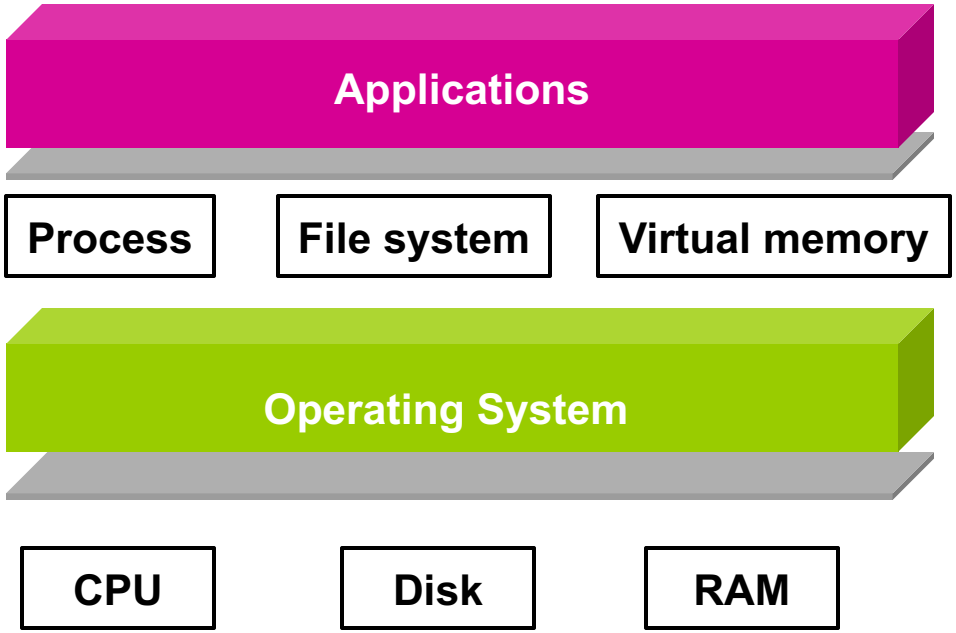
Projects are HARD?

- Probably the hardest class you will take at UCR in terms of development effort
  - You must learn gdb if you want to preserve your sanity! ☺
- Working on the projects will take most of your time in this class
- Biggest reason the projects are hard: legacy code
  - You have to understand existing code before you can add more code
  - Preparation for main challenge you will face at any real job

Roles an OS

- **Abstraction**: defines a set of logical resources (objects) and well-defined operations on them (interfaces)
  - **Why?** Easier app programming
  - Humans are good at abstraction instead of details
- **Virtualization**: Isolates and multiplexes physical resources via spatial and temporal sharing
  - **Why?** Better hardware utilization
- **Schedule/Control**:
  - **Why?** Fairness, performance, security, privacy, etc.

OS Abstractions



What have we learned?

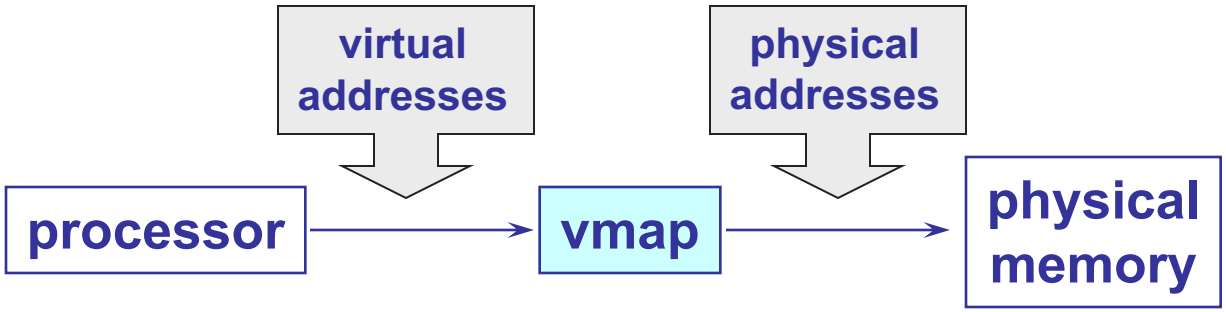
- First half: schedule and synchronization
  - Queueing theory*
  - Consensus*
    - » blockchain
- Second half: translation and cache
  - *There are only two hard things in Computer Science: cache invalidation and naming things. -- Phil Karlton*

Overview

- Translation
  - Virtual address space
    - » Virtual address → physical address
  - Disk and File Systems
    - » Name → file object (e.g., inode)
    - » Offset → disk blocks
- Cache
  - Program Locality
  - Memory Hierarchy

Virtual Addresses

What is the virtualization/illusion we created?

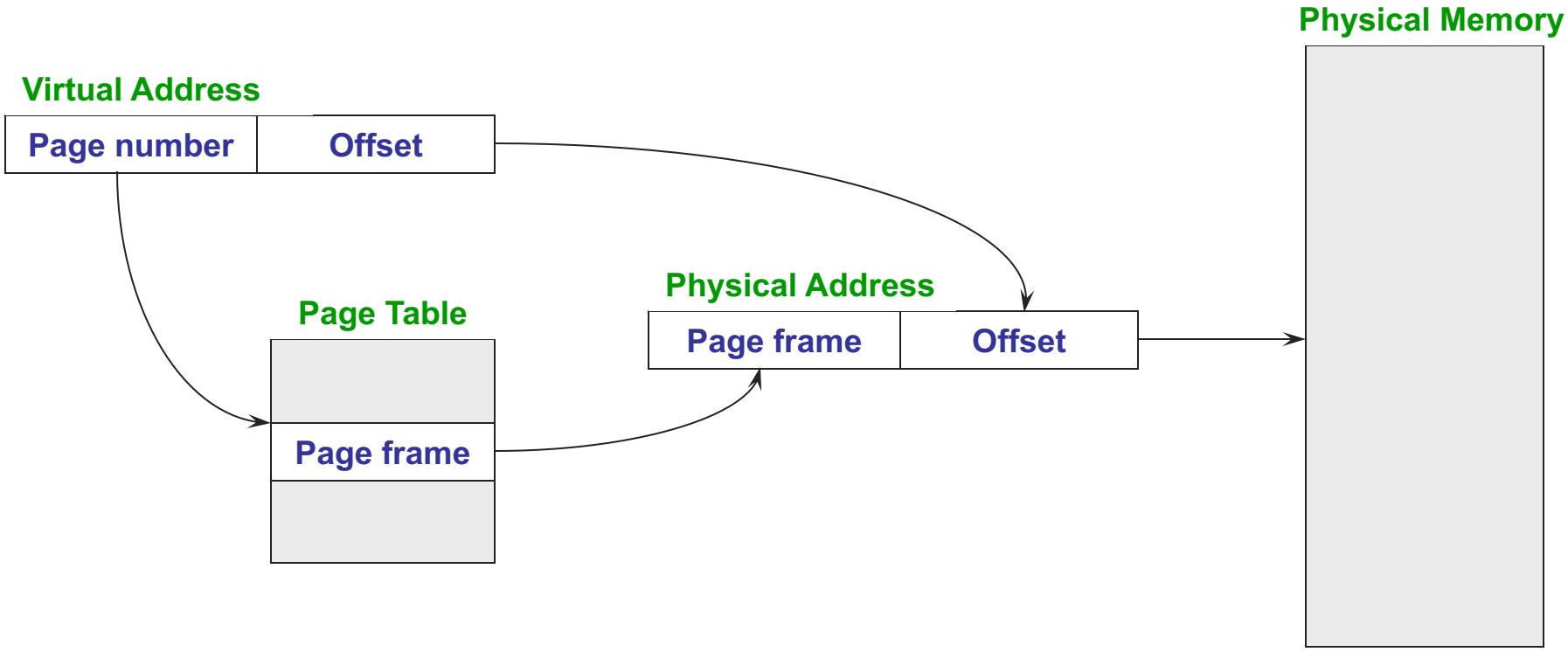


- Many ways to do this translation…
  - Need hardware support and OS management algorithms
- Requirements
  - Need protection – restrict which addresses jobs can use
  - Fast translation – lookups need to be fast
  - Fast change – updating memory hardware on context switch

Translation methods

- Linear/contiguous mapping: y = x + b
  - Physical address = base register + virtual address
  - Disk block = starting block + offset / block size
  - Fast, easy to implement
  - Inflexible, causes fragmentation (either internal or external)
- **Indexed mapping**: y = map[x]
  - Physical page number = page table[virtual page number]
  - File = directory[file name]
  - Disk block = inode[virtual block number]
  - Could go through multiple levels
  - Flexible, easy to manage free space, no external fragmentation
  - Translation is slower

Page Lookups



Intel IA32-e Paging

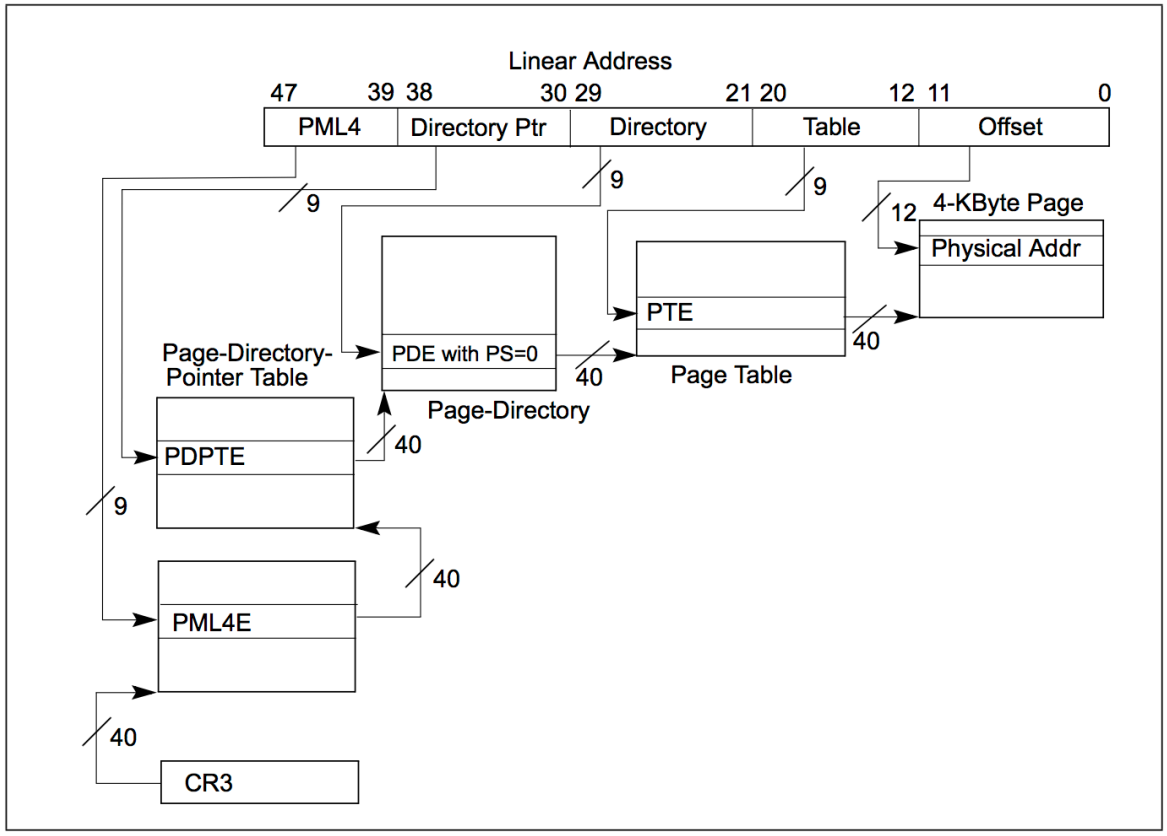


Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

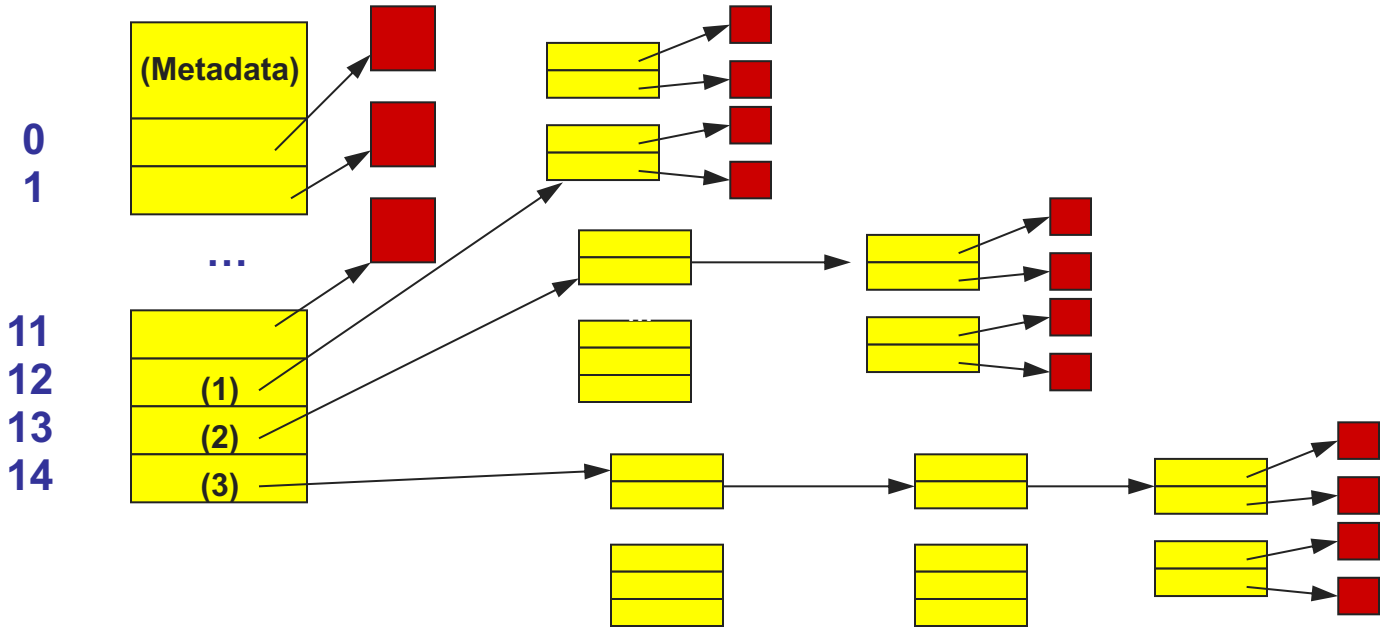
Source: Intel Architecture Software Developer Manuals

Path Name Translation

- Let's say you want to open "/one/two/three"
- What does the file system do?
  - Open directory "/" (well known, can always find)
  - Search for the entry "one", get location of "one" (in dir entry)
  - Open directory "one", search for "two", get location of "two"
  - Open directory "two", search for "three", get location of "three"
  - Open file "three"

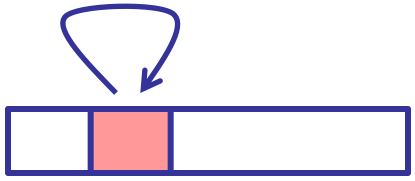
Unix Inodes

- Unix inodes implement an indexed structure for files
  - Also store metadata info (protection, timestamps, length, ref count…)
- Each inode contains 15 block pointers
  - First 12 are direct blocks (e.g., 4 KB blocks)
  - Then single, double, and triple indirect

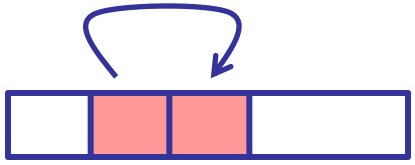


Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently
- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- Data references
  - Reference array elements in succession (stride-1 reference pattern). **Spatial locality**
  - Reference variable **`sum`** each iteration. **Temporal locality**
- Instruction references
  - Reference instructions in sequence. **Spatial locality**
  - Cycle through loop repeatedly. **Temporal locality**

Cache

- **Cache**: a smaller, faster storage that acts as a staging area for a subset of the data in a larger, slower storage.
  - The storage could be a software data structure (like index tables) or a hardware device (like spinning disk)
- Why does cache work?
  - Because of locality!
    - » Hit fast storage much more frequently even though its smaller

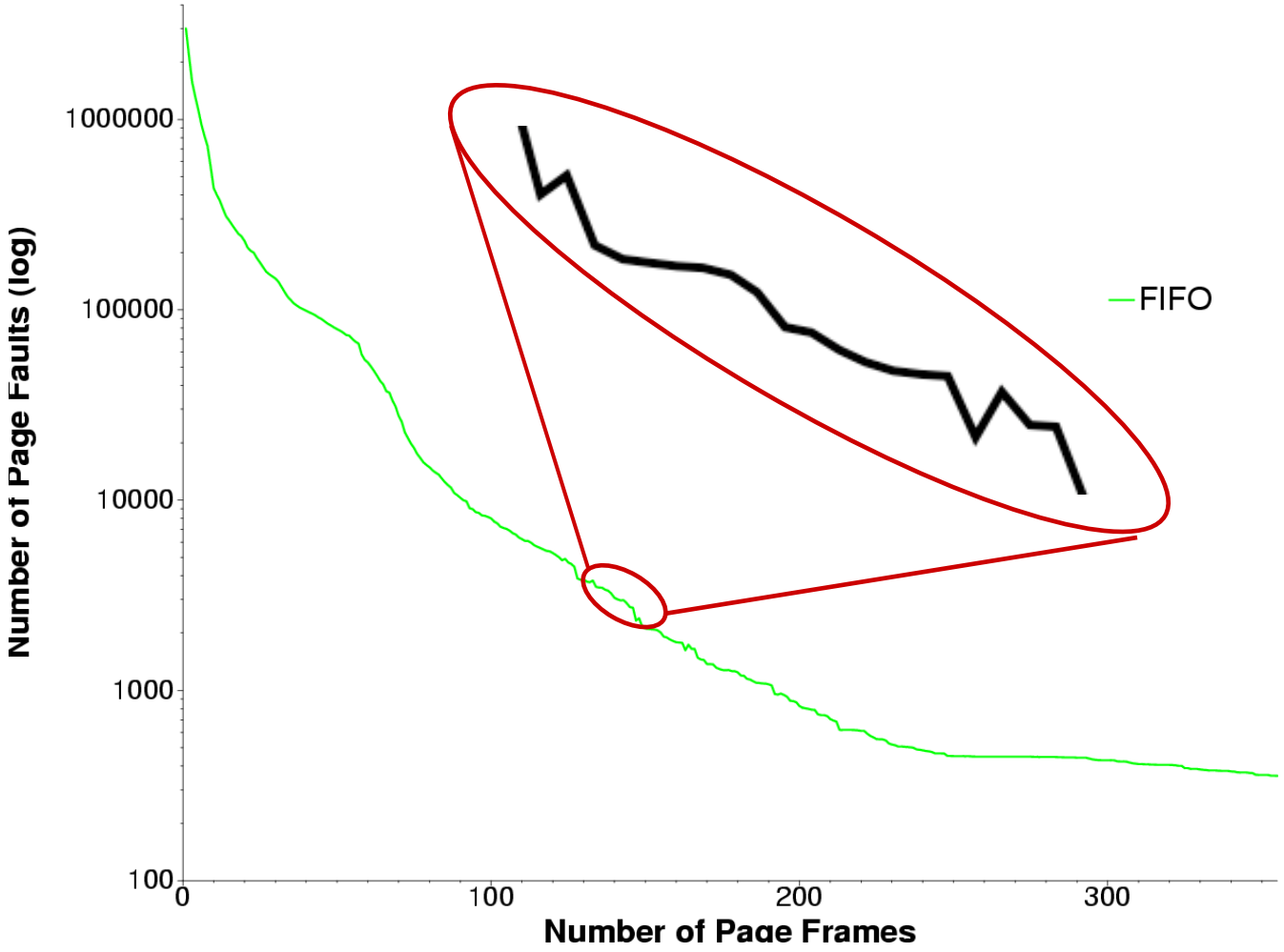
General Cache Concepts

- Hit
  - Data needed is in the cache
- Miss
  - Data is not in the cache
  - Types of misses
    - » Cold (compulsory) miss: cache is empty
    - » Conflict miss: cache is not full but due to placement policy, the slot the data will be mapped to is occupied
    - » Capacity miss: the set of active cache blocks (working set) is larger than the cache size

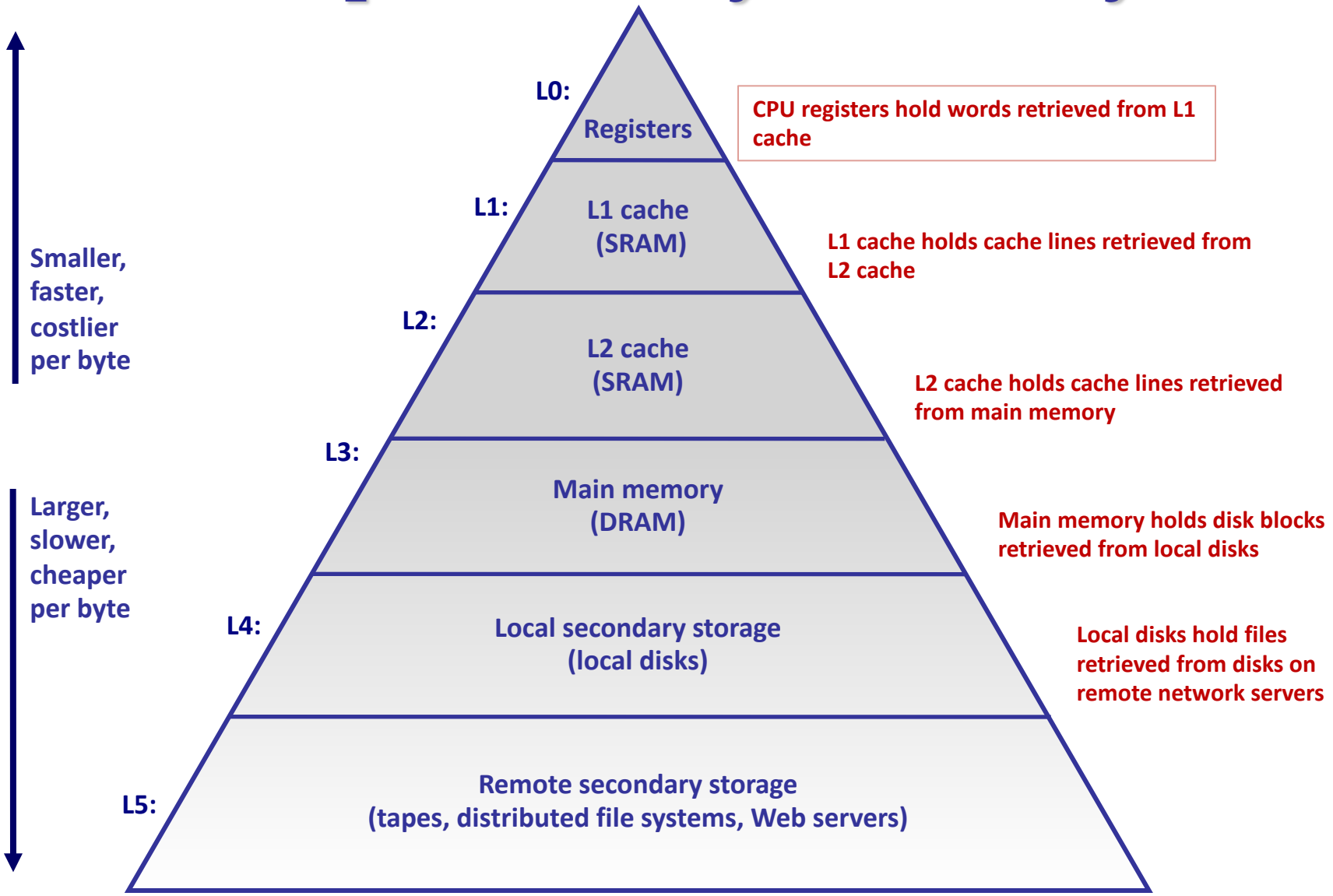
Cache replacement policy

- Cache replacement policy: determine which data to remove when there is a miss
- Belady's algorithm
  - Idea: Replace the page that will not be used for the longest time in the future
  - Optimal but not practical: have to predict the future
  - Serves as the baseline to measure other algorithms
- Common replacement algorithms
  - ~~FIFO~~, Least Recently Used (LRU), access bit, LRU clock

Example: Belady's Anomaly



An Example Memory Hierarchy



Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Locality and Performance

Matrix Multiply: relative speedup to a Python version (18 core Intel)

Version	Speed-up	Optimization
Python	1	
C	47	Translate to static, compiled language
C with parallel loops	366	Extract parallelism
C with loops & memory optimization	6,727	Organize parallelism and memory access
Intel AVX instructions	62,806	Use domain-specific HW

from: "There's Plenty of Room at the Top," Leiserson, et. al., *to appear*.

Security and Reliability

- Access control
  - DAC: access control list and capabilities
  - CS 165
- Reliability
  - Crash consistency: journaling file systems (**JFS**)
  - Disk failures: Redundant Array of Inexpensive Disks (**RAID**)

Let's do some problems

- Check iLearn for keys to homework and previous exams

Virtual Address Translation

- Consider the segment table
  - Is the information in the segment table consistent?
  - If there are possible errors that you identify, what will their implication be?
  - What will be the result of translating the following virtual addresses?
    - #4, 80
    - #1, 200

Segment	Base	Limit
0	800	600
1	1200	140
2	70	100
3	1350	880
4	2200	96

Paging

- An OS is using two-level paging to implement a 28-bit virtual address space per process. The page size is 256-bytes, and the machine does not have a TLB. Assume an even split of address bits between first- and second-level page tables.
  - Explain the steps involved in looking up the virtual address 0x03bf04d, when all pages are present in memory.
  - For the system above, what is the maximum number of page faults that could be generated in response to a memory access?

Cache Replacement Policy

- Consider a process that has been allocated 5 pages of memory: P1, P2, P3, P4, and P5. The process accesses these pages in the following order: `P1 P2 P3 P4 P1 P2 P5 P1 P2 P3 P4 P5`
  - What is the Belady's anomaly for cache replacement policies?
  - Illustrate Belady's anomaly by precisely describing the execution of the FIFO page eviction algorithm in two cases and by comparing the number of page faults incurred in these two cases: 3 physical pages & 4 physical pages.
  - Show how the LRU page eviction algorithm would work in the same scenarios.

Locality and cache

- Caching is the core technique to bridge the gap of access performance between different layers in the memory hierarchy. Caching works because of the program locality. Explain what spatial and temporal locality is.

File System

- On the original UNIX file system, we are reading the file “/one/two/three.txt” All the directories fit in a single block each.
  - Describe and count the number of disk reads involved in reading the file. Assume no disk cache/buffer is used.
  - Similar to TLB, the OS uses a path cache to reduce the number of disk reads when traversing a path.
  - How does FFS improve performance over the basic UNIX file system?

File System

- How does using the `open()` system call help the performance of the file system? In other words, what's the difference when `read()` and `write()` system calls use file name as argument instead of file descriptors?
- Let’s create a symbolic link called `/one/three` pointing to three.txt – Assume that `three` has a single block of data. List all i-nodes that are modified or created.
- Explain **two** specific scenarios of problems that can happen if the system crashes in the middle of creating symbolic link in Q1.5. What consequences can these inconsistencies cause?

File System Implementation

- Consider a file system that uses a structure similar to an `i-node` but with the following differences. If the file size is less than 100 bytes, the data is stored directly in the i-node. If it is larger, there are 6 direct links (point to a data block), 1 single-indirect links, 2-double indirect links and 1 triple indirect link. Assuming the pointer size is 4 bytes and the disk block size is 512 bytes.
  - Describe how many disk blocks can a single file index in this system?
  - How many blocks (including `i-node` and index blocks) are needed to address a file of size 50 bytes, 500 bytes, 50Kbytes, and 5Mbytes?

Access Control

- The slides of file system on the class website has the following permission: `-rw-r--r-- csong csprofs fs.pdf`
  - Who is the owner of the file? What operations can the owner perform? What special operations can owner perform?
  - What is the group of the file? What operations can the group perform?
  - What operations can other users perform?