# CS 202
# Advanced Operating Systems

## Winter 26

Lecture 1: Course Introduction

Instructor: Chengyu Song
Associate Professor
Department of Computer Science and Engineering

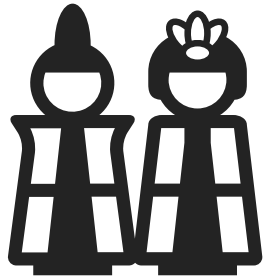
# Teaching Staff

## Chengyu Song

I am an Associate Professor in CSE

Office hours TTh 11:00pam-12:00pm or by appointment

- In person and online



## TA for Labs

Jiajun Chen (PhD student)

Office hours and links on Canvas

Leads for Labs

# Expectations

- Advanced
  - You should have already learnt about basic operating system concepts
    - » We will do a quick review
  - **Read papers** (together)
    - » Advanced topics
  - **Discussion and active class participation** (10% of final grade)
  - More challenging labs

# What is this course about?

- How has the role of the operating system **evolved** over time?
  - How does the past inform the present?
- What are the **principles** that underlie Operating Systems?
- What are current and future **trends** in OS?
- Make it real: lab assignments to get some experience with OS development
- *Get you ready to do systems research*

# Some topics we will cover

- Operating Systems models and how they evolved
  - Extensibility, protection, performance
- Concurrency:
  - Synchronization and Scheduling
  - Multicore OS
- File systems:
  - Local, networked, distributed, internet scale
- Other advanced topics

# Class format

- For every topic:
  - Some overview
  - **Read related book chapters** before the class
  - **Discuss research papers**
- Research papers:
  - Critique for some required papers (1-2 paper per topic)
  - You are responsible for required papers and material discussed in class

# Reading Research Papers

- Guidelines for reading papers
  - Make sure to identify authors' goals and assumptions. Not always directly stated.
  - Look for high-level takeaways.
  - **Simulate the whole process in your head**
  - Follow a multi-pass reading strategy
    - Pass1: Get overview. Pass2: Read details and make notes. Pass3: Re-read details to evaluate.
  - Think how techniques used are applicable today. Identify extensions.

# Questions while Reading Papers

- What are the primary goals (hypothesis)?
  - 2 sentence elevator pitch
- Why did the authors do what they did the way they did?
  - Understand motivation for design
- What were the driving forces for the paper at the time it was written?
- What parts still hold? What parts don't?
- How did they experiment to support their ideas?
- **Write the review by yourself**

# Projects

- 3 Lab Assignments
  - Modifications on xv6 (https://github.com/mit-pdos/xv6-riscv)
  - Related to the topics discussed
- All labs are done individually
- Submission
  - Design documents & test plans
  - Code & tests
  - Short demo
- Deadlines are **hard**!
- Late policy
  - 10% penalty per day; up to 60% late penalties

# Course Materials

- Textbook: Apraci-Dessau and Apraci-Dessau, **OS, 3 easy pieces**
  - https://pages.cs.wisc.edu/~remzi/OSTEP/
  - Please read, most chapters are only a few pages
- Lecture notes and Papers: on Canvas
- Other good books:
  - Anderson and Dahlin, ***Operating Systems: Principles and Practice (recommended)***
  - Silberschatz, Galvin, and Gagne, ***Operating System Concepts***, John Wiley and Sons, 8th Edition **(recommended)**

# Grading Policy

- Lab Assignments: 40%
  - Lab 1: 10%, Lab 2: 15%, Lab 3:15%
- Reading and critiquing papers: 20%
- Attendance & Class Participation: 10%
- Final (cumulative, closed book): 30%

# Cheating and Plagiarism

- Your code and critiques may be compared against each other
- Academic integrity policies and procedures
  - https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures
- What consequences can happen?
  - Will be reported to the Academic Integrity Committee (AIC)
  - Grade penalty
  - Course failure
  - Academic sanctions: probation, suspension, dismissal

# Policy on AI

- AI is replacing many entry-level positions
  - You do not want to become replaceable
    - If you just copy&paste whatever AI told you, you're replaceable, elementary school kids can do this
    - If you stop thinking, you become replaceable, as we're teaching AI to think/reason
- Learning to work with AI is necessary
  - Use AI to help you understand research papers better, but write the review by yourself and using your own words
  - Use AI to help you understand existing code better, but put your own thoughts into design and testing
  - Use AI to help you code, but follow your design and know how to test and reason

# Quick Review of Operating System

# Questions for today

- What is “operating”?
- What is an operating system?
- Why do we need operating systems?
- What do operating systems do?

# Operating

- Ask ChatGPT: what is operating

- Get things done (hide the details)
- Management of resources
- Optimization (reduce cost, improve deliveries, robustness)

# Operating System

- Why we have computers?

- Operating system: the COO of computer systems
  - Get things done (hide the details)
  - Management of recourses
  - Optimization

# Goals of an OS/Platform

- **Enabling**: allow users and developers to use (new) hardware to do new things that are not possible before






- **Usability**: make it easy to use / **program with**

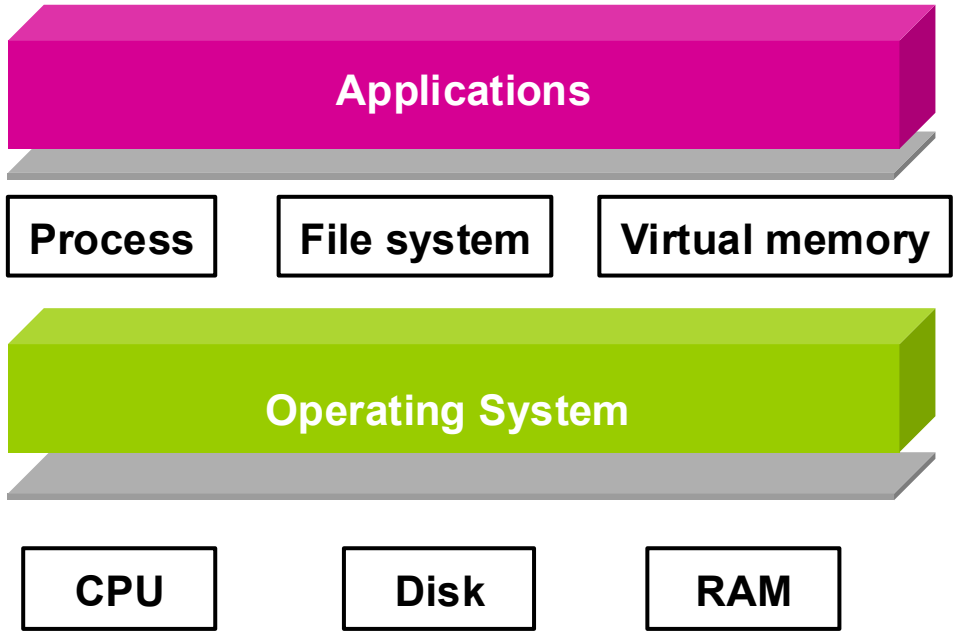

- **Efficiency**: are we wasting our money/time/energy?

- **Fairness and protection**: make sure good things will happen and bad things will not

# Roles of 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?** Easier programming, better hardware utilization
- **Control**:
  - **Why?** Fairness, performance, security, privacy, etc.

# OS Abstractions



# How to Share Computing/Processors

- Time sharing
  - Events and controlled/limited direct execution
  - Context switch (user <-> kernel, thread1 <-> thread2)
  - Scheduling (OS controls)
  - Synchronization (application controls)
- Parallel and concurrent
  - Parallel: (really/physically) at the same time, with none or minimal dependencies
  - Concurrent: (conceptually) at the same time, likely with shared resources
  - Consensus: who gets the lock / bitcoin

# How to Share/Access Storages

- 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

# 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 (no cache needed)
  - 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]
  - IP address = DNS[domain name]
  - Could go through multiple levels
  - Flexible, easy to manage free space, no external fragmentation
  - Translation is slower (cache to the rescue)

# Intel IA32-e Paging

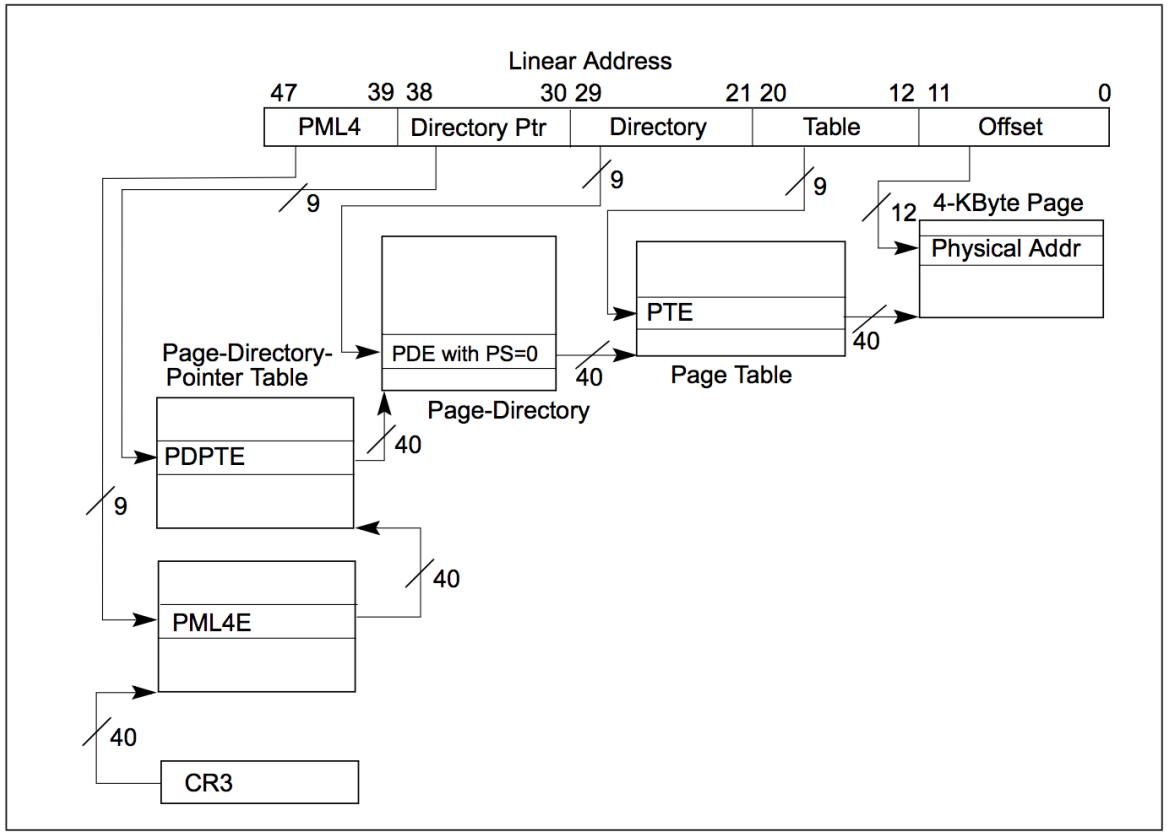



Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

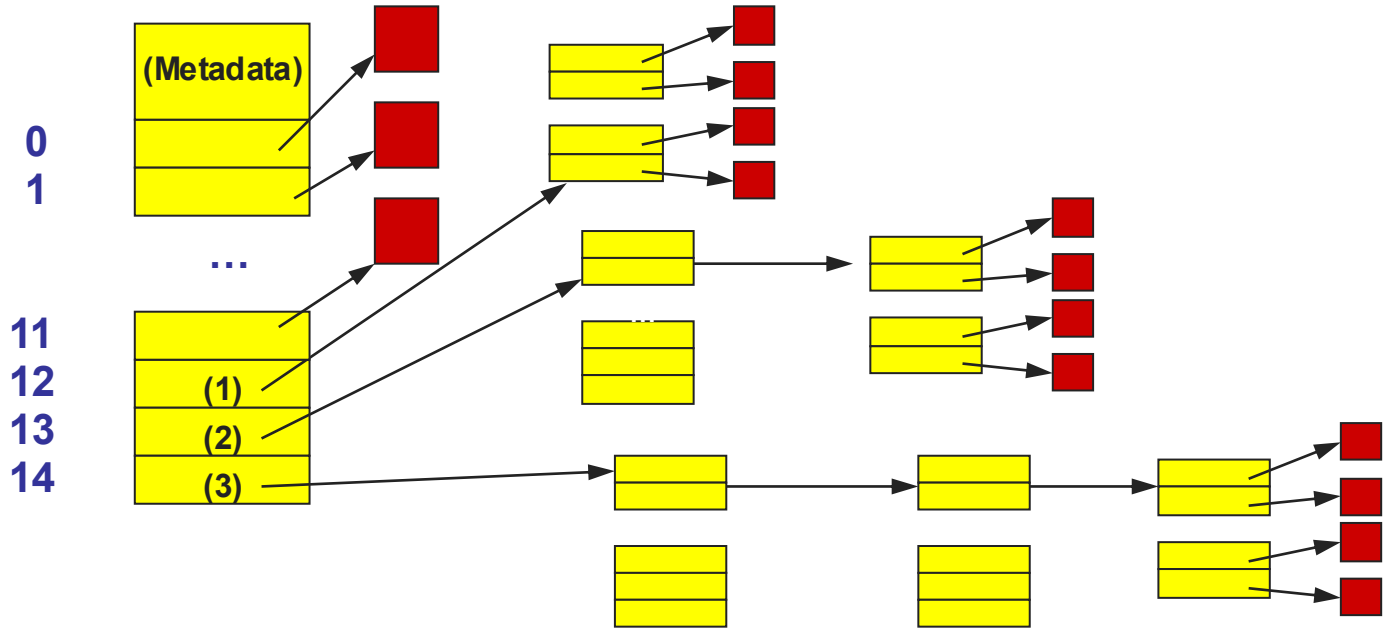
Source: Intel Architecture Software Developer Manuals

# 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"

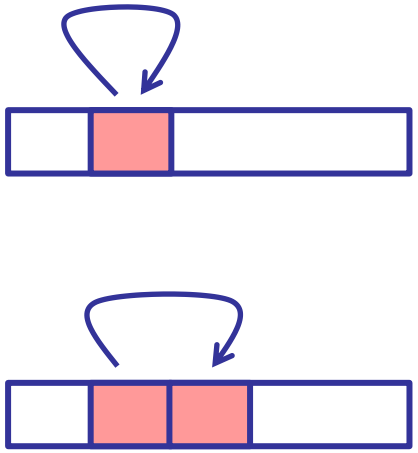
# Unix Inodes

- 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

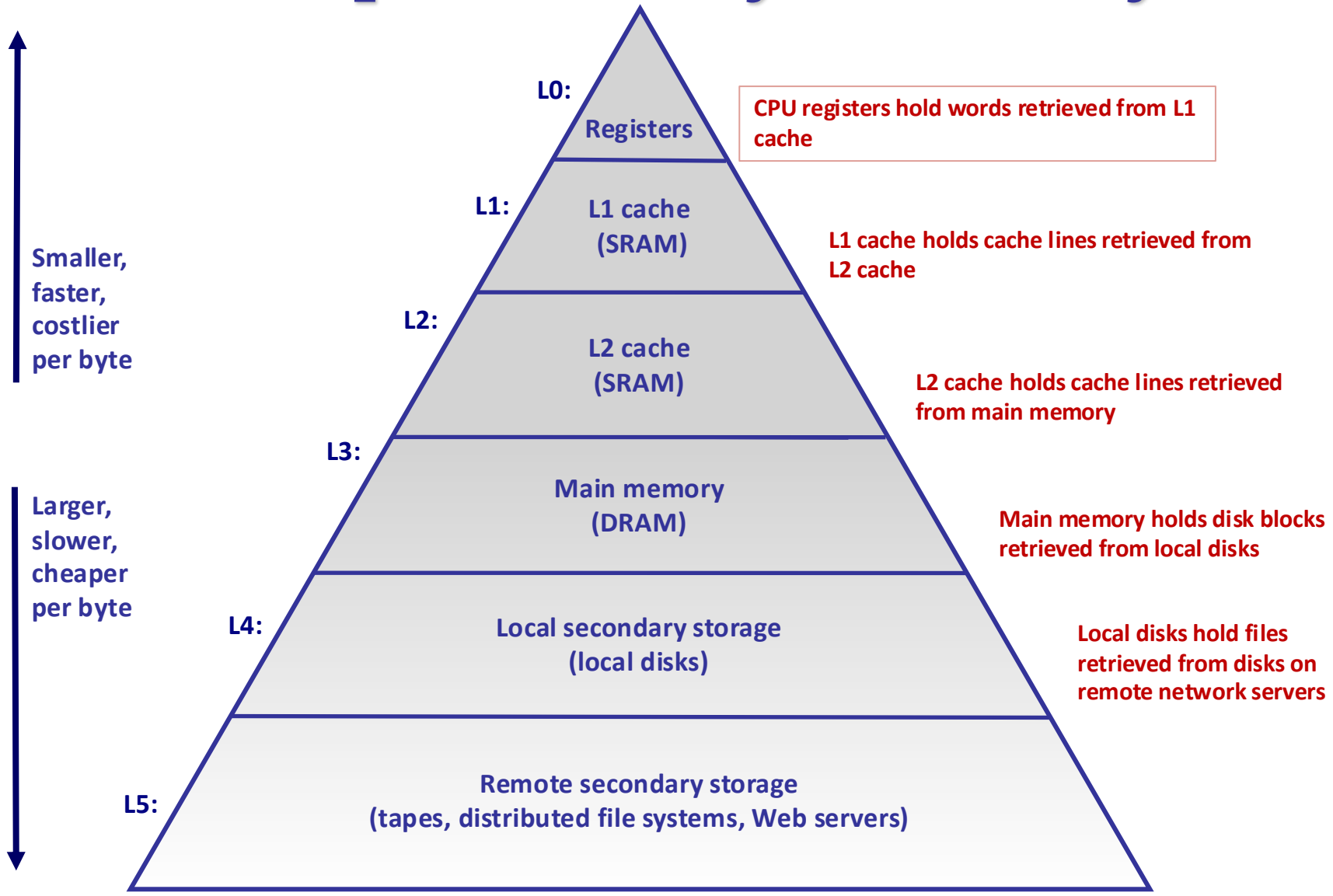
# General Cache Concepts

- 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: must predict the future
  - Serves as the baseline to measure other algorithms
- Common replacement algorithms
  - FIFO, Least Recently Used (LRU), access bit, LRU clock

# 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 |

# 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**)