COMP2310/COMP6310
Systems, Networks, & Concurrency

Convener: Prof John Taylor

Australian
National
University

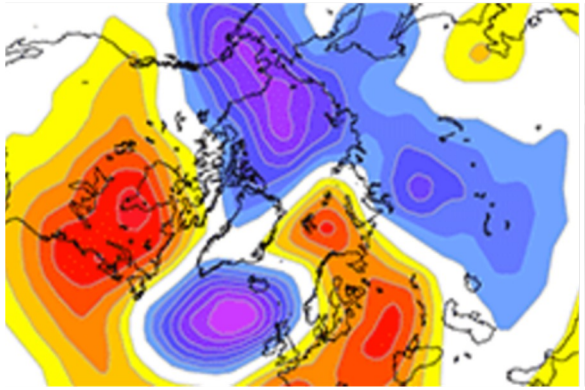
Teaching: Comp4300 - Parallel Systems (semester 1)

Systems, Network, and Concurrency (semester 2)

Research: AI for Science, High Performance Computing

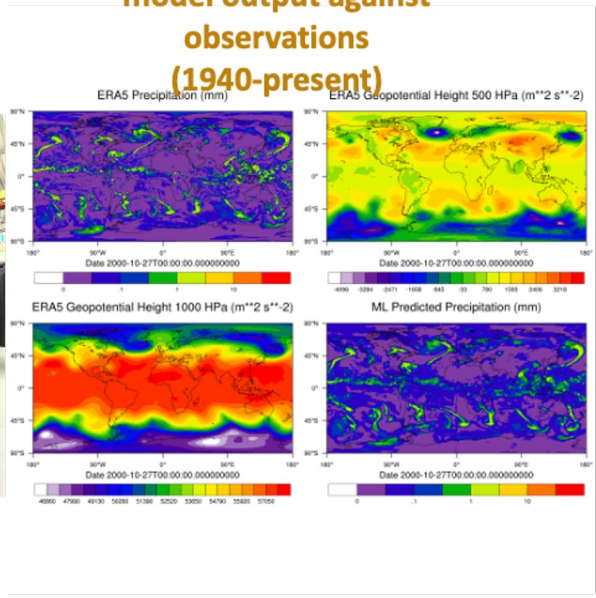
Feature Selection → ***Model Training*** → ***Model Inference***

ERA5 Global high resolution (30km) reanalysis datasets (1940-present)



Build AI/ML models and scale computation on GPGPU HPC systems (Keras, TensorFlow, Pytorch)

Infer, verify, analyse, display model output against observations (1940-present)



Quick Logistics

Course webpage: https://comp.anu.edu.au/courses/comp2310/

Lectures (on the website**)**

- Lecture slides
- Lecture videos (*Echo360*)
- 2 hours reserved (some lectures may be shorter, demos etc)

Policies

- General conduct, assignment submissions, support, management, grading

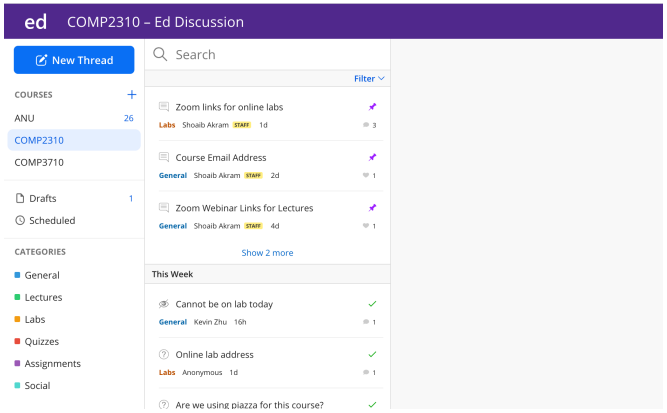
Resources

- Past exams
- Information needed to finish the labs and assignments

Edstem

We will use `edstem` for all communication

- If you ignore `edstem`, *you will miss key announcements*
  - Drop-in sessions, make-up lectures, problems, exercises, corrections, lecture timing
  - Ask questions on `edstem` first (most likely you will receive a response quickly)
- *Ask instructors private questions on* `edstem`
- Students are added/dropped automatically



Course Email

comp2310@anu.edu.au

- Do not send me a direct email except for requests:
  - Super urgent
  - Personal
  - EAP-related

ANU COLLEGE OF SYSTEMS AND SOCIETY

CLASS REPRESENTATIVES

- Class Student Representation is an important component of the teaching and learning quality assurance and quality improvement processes within the ANU College of Systems and Society (CSS).

- Each semester, we put out a call for Class Representatives for all ANU College of Systems and Society (CSS) courses. Students can nominate themselves for one or more of the courses they are enrolled in.

Australian National University

Roles and responsibilities:

- The role of Student Representatives is to provide ongoing constructive feedback on behalf of the student cohort to Course Conveners and to Associate Directors (Education) for continuous improvements to the course.

- Act as the official liaison between your peers and convener.
- Be available and proactive in gathering feedback from your classmates.
- Provide reports on course feedback to your course convener.
- Close the feedback loop by reporting back to the class the outcomes of your feedback.

- Note: Class representatives will need to be comfortable with their contact details being made available via Wattle to all students in the class.

- For more information regarding roles and responsibilities, contact:
- ANUSA CSS representatives (sa.cecc@anu.edu.au).

– Why become a class representative?

- **Ensure students have a voice** to their course convener, lecturer, tutors, and College.
- **Develop skills sought by employers**, including interpersonal, dispute resolution, leadership and communication skills.
- **Become empowered**. Play an active role in determining the direction of your education.
- **Become more aware of issues influencing your University** and current issues in higher education.
- **Course design and delivery.** Help shape the delivery of your current courses, as well as future improvements for following years.

Want to be a class representative?
Nominate today!

Please nominate yourself to your course convener by end of Week 2

Motivation

Recall: How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Devices

Recall: How do we make electrons do the work?

- Using a sequence of systematic transformations
  - Developed over six decades
- Each step must be studied and improved for the whole stack to work efficiently

Recall: Transformation Hierarchy

- We call the steps of the process: Levels of transformation OR Transformation hierarchy
- At each level of the stack, we have choices
  - **Language:** Java, Python, Ruby, Scala, C++, C#
  - **ISA:** ARM, x86, SPARC, PowerPC, RISC-V
  - **Microarchitecture:** Intel, AMD, IBM
- If we ignore any of the steps, then we cannot: -
  - Make the best use of computer systems
  - Build the **best** system for a set of programs

Problem

Algorithm

Program

Architecture

micro-arch

circuits

devices

Recall: Transformation Hierarchy & Us

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

2310 Compiler and Third-Party Libraires/Binaries

2310 Operating System

2300 Instruction Set Architecture (ISA)

2300 Microarchitecture

2300 Circuits

Devices

Program Execution

Computer Architecture

Computer Organization

Recall: Hardware and Software

Problem Statement: “Save the planet”

Software	The Algorithm
	Program in a High-Level Language
	Compiler and Third-Party Libraires/Binaries
	Operating System

ISA = Hw/Sw

boundary/interface Instruction Set Architecture (ISA)

	Microarchitecture
	Circuits
Hardware	Devices 

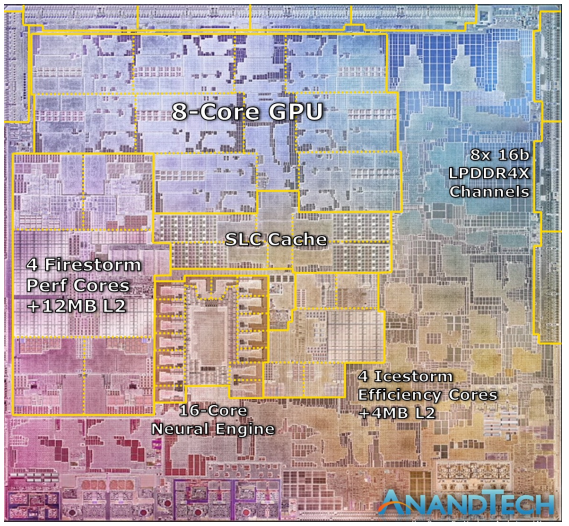
Recall: Two Recurring Themes

- The notion of abstraction
- Hardware versus software

Recall: The Notion of Abstraction

- **Abstraction:** Know components from a high level of detail

No human (programmer) can track 10 billion elements. **Computer systems work because of abstraction!**



Apple M1 Chip
Billions of transistors
All working in parallel

Recall: The Notion of Abstraction

- **Abstraction**: View the world from a higher level
- Focus on the important aspects
  - Input? Output? X = ADD or MULTIPLY

- Raise the level of abstraction for productivity and efficiency
- But what if the world below does not work as expected?
  - To deal with it, we need to go below the abstraction layer
- **Deconstruction**: To un-abstract when needed
  - Important skill

Recall: The Notion of Abstraction

- We will use this theme a lot!
  - Each layer in the transformation hierarchy is an abstraction layer!

Problem

Algorithm

Program

Architecture

micro-arch

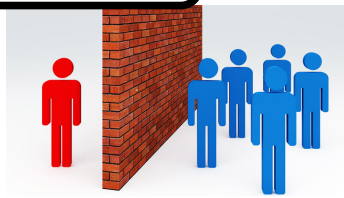
circuits

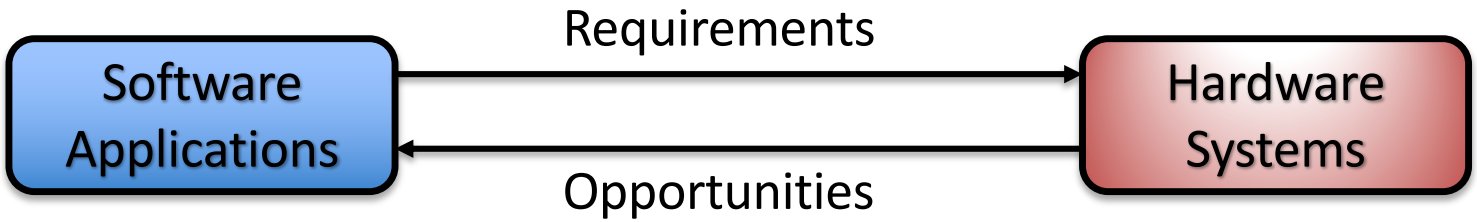
devices

Recall: Hardware versus Software

- Hardware versus software
  - Hardware: Physical computer
  - Software: Programs, operating systems, compilers
- **One view:** Ok to be an expert at one of these
- Hw and Sw: Two parts of the computer system
  - **COMP2300 view:** Knowing the capabilities/limitations of each leads to better overall systems

COMP2310 deepens this knowledge

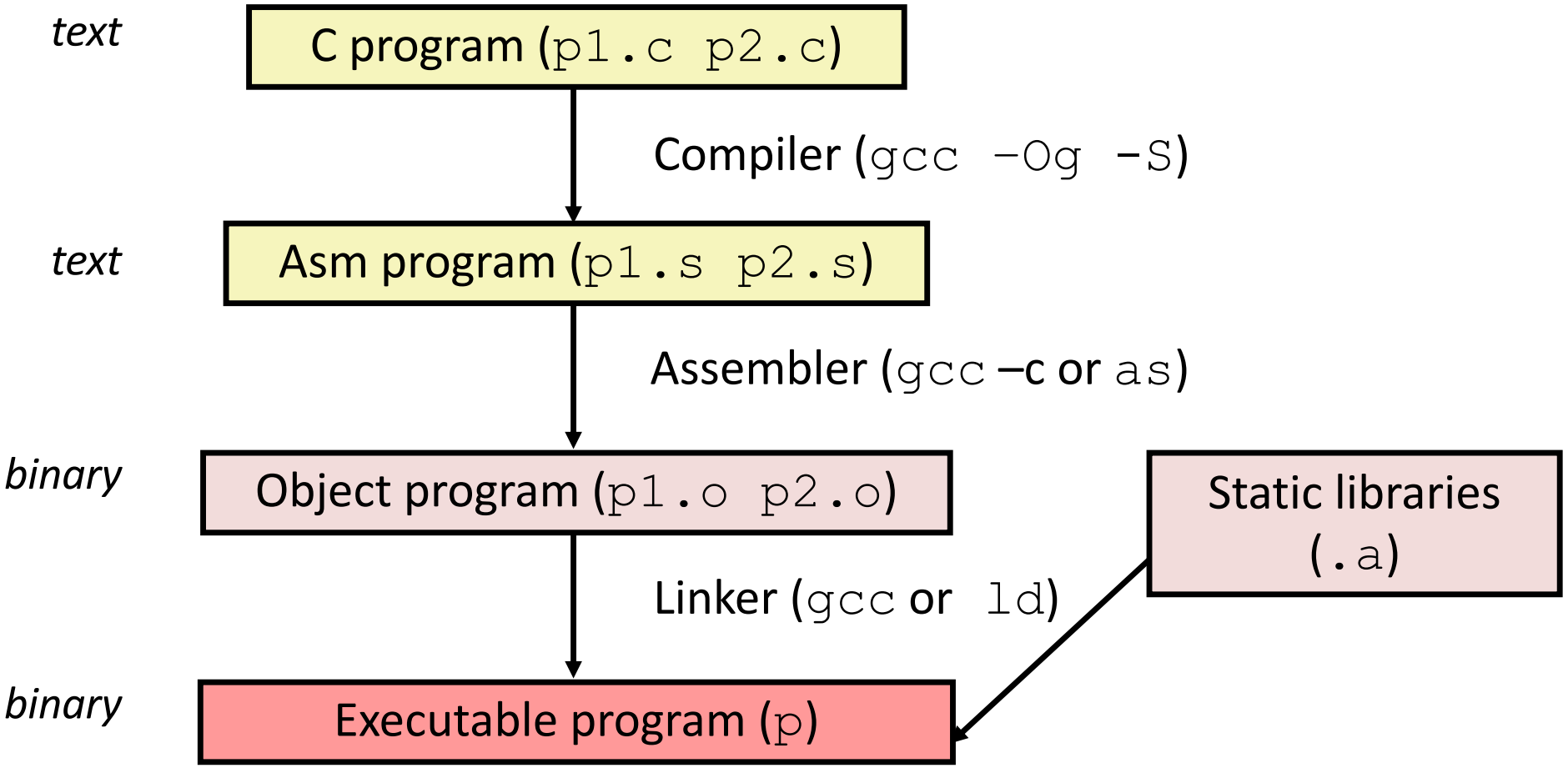




Role of Compiler

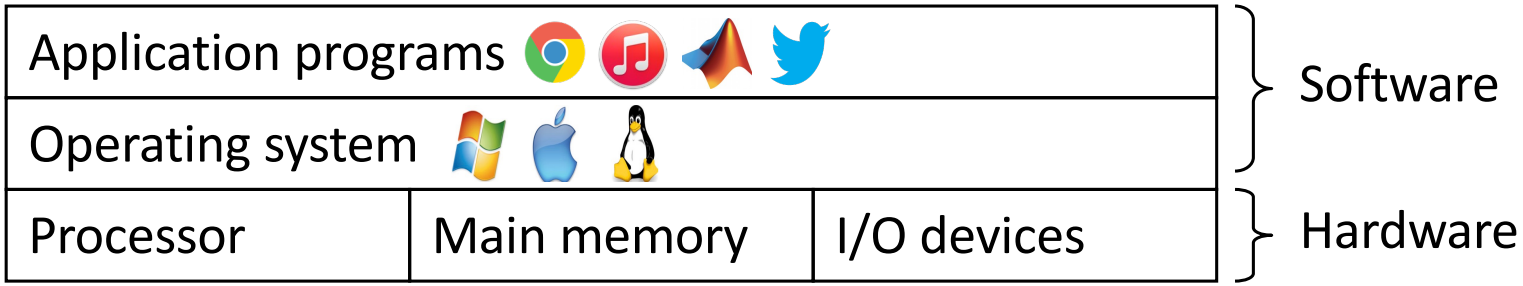
- What does a compiler do?
  - Translates high-level code into assembly
- More generally, the compilation toolchain generates machine code in a sequence of stages:
  - translate a group of related source files into assembly
  - resolve inter-dependencies between source files (*linking*)
  - handle the *linking* of any external libraries
  - perform *optimizations* (make use of special hardware features)
- It is more complex than line-by-line C to assembly translation
- Learning the process is **important** from a performance, efficiency, security, and hacker perspective

Turning C into Object Code (details later)



Role of Operating System (OS)

- Operating system



- Enables safe abstractions of hardware resources
- Virtualizes hardware for use by programs
  - Gives each program the **illusion** that it has the entire resource for itself
- Manages the hardware resources for efficient and safe working of the system

COMP2310, Goal # 1

- Deepen the understanding of how applications interact with compiler and OS, and hardware

- Today, critical for software to be correct, performant, efficient, secure



- Demystify how programs are loaded into memory and executed
  - What happens when you click an icon to start an application?
  - Or type the program name into a shell program and press enter

COMP2310, Goal # 1 (cont'd)

- How are C programs translated into x86-64 assembly?
  - Compilation and linking fundamentals
  - Object files, executable formats, etc
  - Implementation of loops, procedure calls, data structures (reprise)
  - Optimizations done (not done) by the compiler

Assembly is Important!

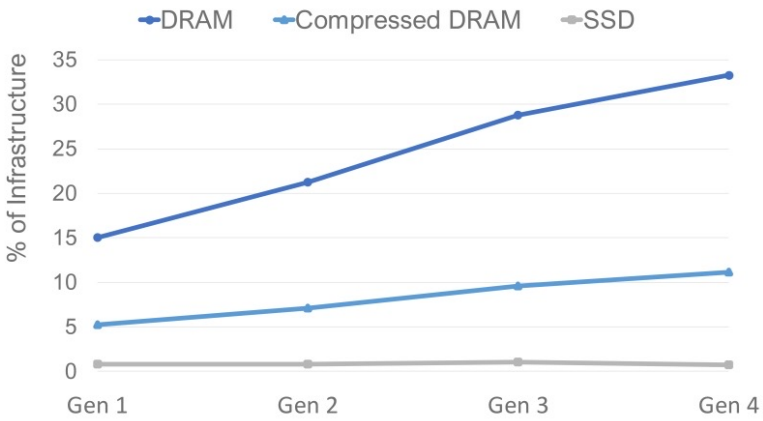
- Intel x86-64 ISA widely used in server hardware
- **Tuning** performance
  - Understanding optimization done (not done) by the compiler
  - Understanding behavior of programs and exploiting choice via compile-time options
- Writing **systems software** (device drivers)
- Fighting security vulnerabilities
- Behavior of buggy programs

COMP2310, Goal # 1 (cont'd)

- What does the memory hierarchy look like?
  - How do caches work in more detail?
    - What is their impact on program behavior? (programmer's perspective)
  - How does main memory differ from a disk drive?
  - How does device behavior impact the design of computer programs?

Memory matters!

- Memory is a limited resource
  - Must be carefully managed
- Memory bugs are hard to detect
  - Understanding pointers and memory allocators helps
- Memory performance is not always uniform
  - Caches, virtual memory effects need to be understood



Meta datacenters, 2022

Memory matters!

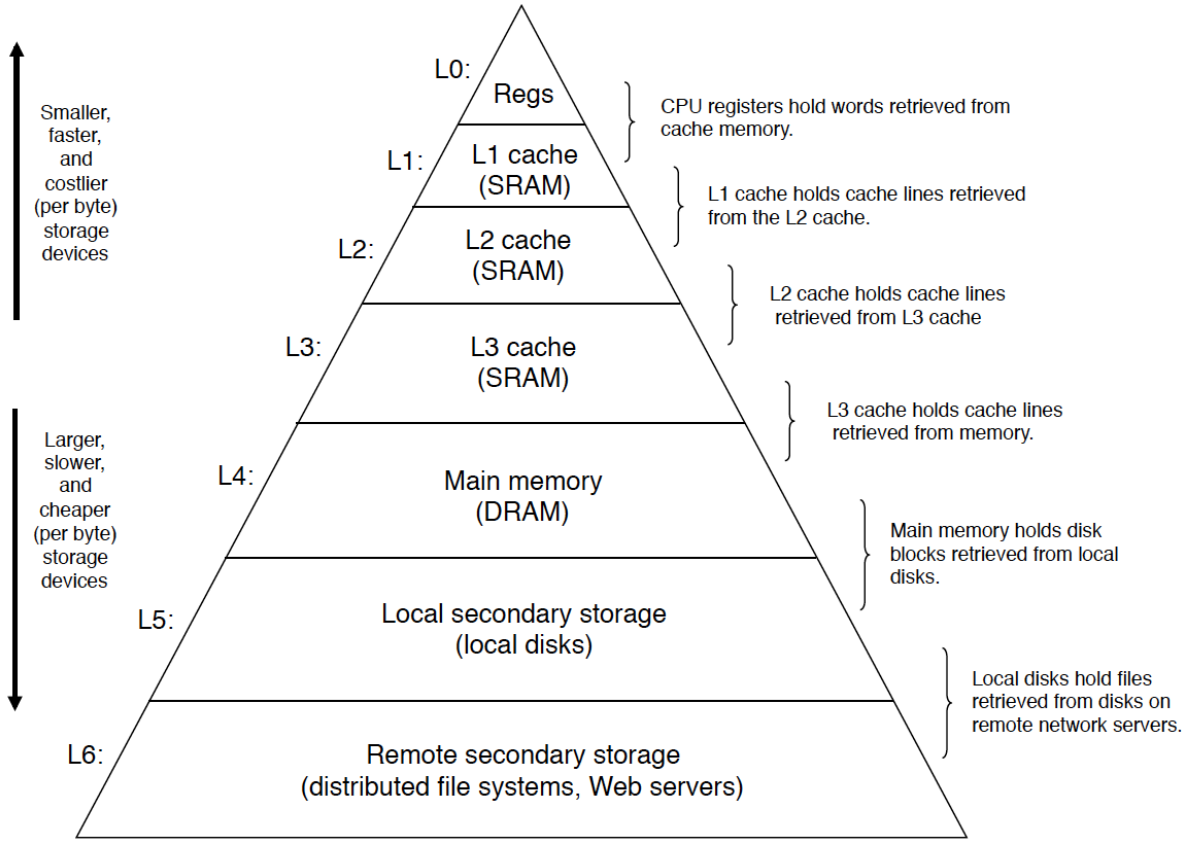
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms 2.0 GHz Intel Core i7 Haswell 81.8ms

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how to step through a multi-dimensional array

Memory Hierarchy



COMP2310, Goal # 1 (cont'd)

- How does the operating system abstract hardware resources for use by application programs?
  - Processes
  - Virtual memory
  - Files

- All these are abstractions the OS uses to isolate computer programs from each other
  - Our focus is **NOT** on (re)building these "mechanisms" but writing programs to use them

COMP2310, Goal # 2

- Understand how applications use the operating system (OS) and the C standard library for writing real-world applications
- Search engines
- Databases

COMP2310, Goal # 2

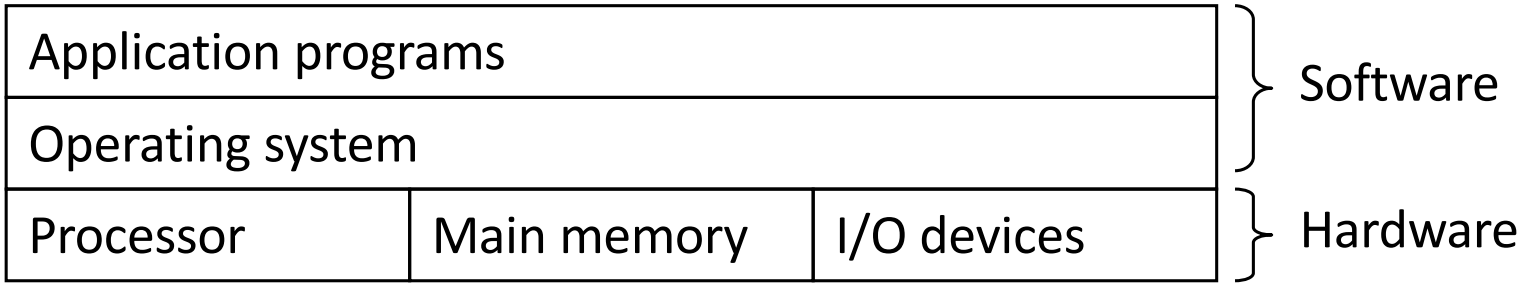
- Write low-level code that interfaces with the operating system kernel and C library

- What interface (and interesting system calls) does the operating system provide?

- Learn to: -
  - Implement memory allocators, read/write from/to storage disks and SSDs
  - Communicate with the outside world (networking), and manage concurrently running processes and applications

User Code vs. Kernel Code

- OS manages the hardware, interposed b/w the program and hardware



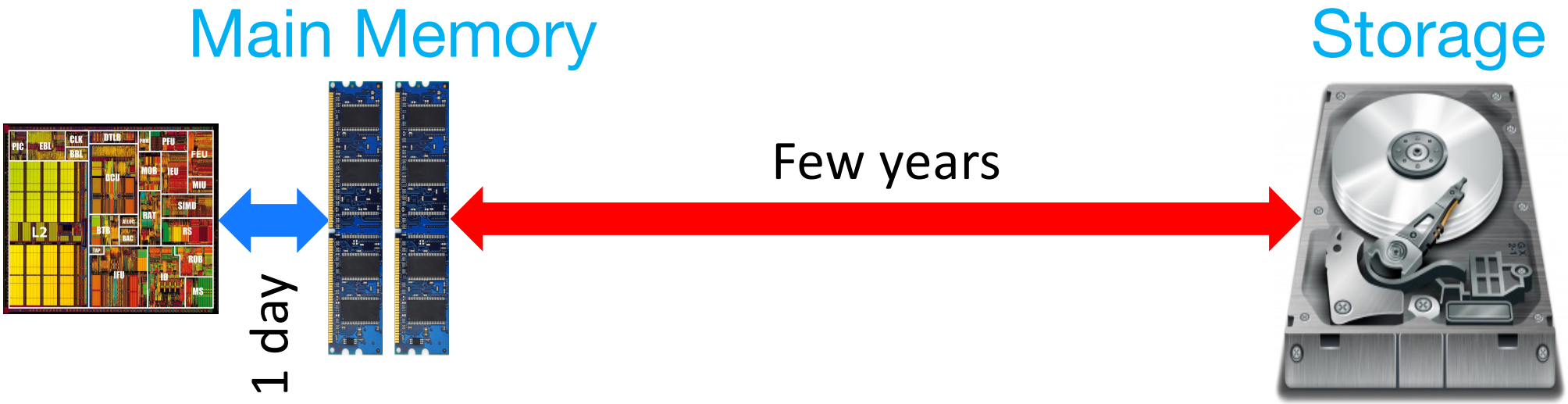
- Application code that runs on top of OS or any resource manager in general is user code (or user program)
- The code that manages the hardware is kernel code
- The CPU is either in user mode or kernel mode

What is OS kernel?

- Core component of OS that manages the hardware
  - Device management (keyboard, mouse, display, etc)
  - Memory (RAM) management
  - Network management
  - Storage management
  - Filesystem code
- What else is in the OS?
  - Shell
  - GUI
  - Utilities

What does an OS do?

- Manages the hardware, interposed b/w the program and hardware
- Our high-level view of system (no network for simplicity)



- OS manages CPU (processor), memory (RAM), input/output (I/O) devices (keyboard, disk, display, network), and files on disk

How do applications use the OS?

- OS provides services to be accessed by user programs
- Programs can make use of "system calls" on Linux and Windows application programming interface ("**API**")
  - Allocate memory for me
  - Read "**N**" bytes from file F into memory location "**M**"
  - Write "**N**" bytes from memory location "**M**" into file **F**
  - Establish a network connection to www.anu.edu.au
  - Write "**N**" bytes to the network connection
  - Put me to sleep

How applications use the OS?

- OS provides an **interface** for applications to use
  - Programs access hardware/device capabilities through this interface
  - Different hardware → Same interface
  - Interface is constant, its implementation is OS specific

- **We need to learn this interface to write interesting applications**
  - Learning “just enough” details of the implementation to write correct, efficient, secure programs

Goal # 3, Systems Programming

- All these aspects will help you become a systems programmer
- Systems programmers
  - write low-level tools such as compilers, operating systems, and debuggers
  - they must have an acute awareness of the environment, e.g., Linux versus Windows
  - they must use system calls for the specific OS
  - contrast with Python, Ruby, Java programs for business or ML
    - high-level libraries abstract OS and hardware details
  - C library abstracts OS/hardware but many Linux C programs interface with the kernel API

Example of Pure User-Level C Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LEN 10000000

// struct of arrays
// all i points are stored in a single contiguous array
// all j points are stored in a single contiguous array
// all k points are stored in a single contiguous array

struct pointarray3D {
        int i[LEN];
        int j[LEN];
        int k[LEN];
};

struct pointarray3D points;

int sum_k (struct pointarray3D *points) {
        int sum = 0;
        return sum;
}

int main() {
        for (int idx = 0; idx < LEN; idx++) {
                points.i[idx] = 1;
                points.j[idx] = 1;
                points.k[idx] = 1;
        }
        int sum = sum_k(&points);
        printf("sum of all k points is %i \n", sum);

        return 0;
}
```

- Will not crash your machine if you did something wrong
- Programmer's creativity is more critical in solving the problem
  - Can get by not knowing how an array looks like in memory
- Uses a C library function for printing to the screen
- C library takes care of making it happen for the programmer

Example of Pure System-Level C Program

```c
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#define alpha           "abcdefghijklmnopqrstyvwxyz"
#define BUF_SIZE        32

/*
            Do while having super-user privillages:
-------------------------------------------------------------------
|              ' insmod mine.ko' :      insert module into the kernel.|
|              ' rmmod mine.ko'  :      remove module from the kernel.|
-------------------------------------------------------------------
*/
// The entry will be created into the ' /proc ' directory
// the directory that will hold the new device.
static struct proc_dir_entry *ent;
static char message[BUF_SIZE];

static ssize_t mwrite(struct file *file, const char __user *ubuf, size_t count, loff_t *offset)
{
        unsigned int i;
        int rv;
        char __user *p = ubuf;

        printk(KERN_INFO "Write Handler\n");
        printk(KERN_INFO "Size: %d , Offset: %d \n", count, *offset);

        if(count > BUF_SIZE)
                return -EFAULT;

        rv = copy_from_user(message, p, count);

        printk(KERN_INFO "Byte not copied: %d\n",rv);
        printk(KERN_INFO "device have been written\n");

        return count;
}

static ssize_t mread(struct file *file, char __user *ubuf, size_t count, loff_t *ppos)
{
        char __user *ptr;

        printk(KERN_INFO "Read Handler\n");
        printk(KERN_ALERT "Count : %d\n",count);
        ptr = ubuf;

        if(count > BUF_SIZE) {
                printk(KERN_INFO "Adjusting size:\n");
                count = BUF_SIZE;
                printk(KERN_INFO "Size: %d\n",count);
                return (-1);
        }
        copy_to_user(ubuf, message, count);

        return count;
}

// File operations.
static struct file_operations fops =
{
        .owner = THIS_MODULE,
        .read  = mread,
        .write = mwrite,
```

- Device driver code
- Most likely crash your machine if you did something wrong
- Requires intricate knowledge of the hardware for which driver is being written
- Uses Linux kernel sources to reuse functionality
- Even “printf()” is not available
- No C library

Example of User-Space System-Level C Program

Focus of this course!

```c
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <stdio.h>
#include <sys/wait.h>

char ptype[10];
int main()
{
    int size = 50 * sizeof(int);
    void *addr = mmap(0, size, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    printf("Mapped at : %p \n\n", addr);

    int *shared = addr;
    pid_t fork_return = fork();
    if (fork_return > 0)
    {
        shared[0] = 40;
        shared[1] = -20;
        strcpy(ptype, "Parent");
        int status;
        waitpid(-1, &status, 0);
    }
    else
    {
        sleep(1);
        printf("Child  : shared[0] = %d  , shared[1] =  %d  \n", shared[0], shared[1]);
        shared[1] = 120;
        strcpy(ptype, "Child ");
    }
    printf("%s : shared[0] : %d\n", ptype, shared[0]);
    printf("%s : shared[1] : %d\n", ptype, shared[1]);
    munmap(addr, size);
    return 0;
}
```

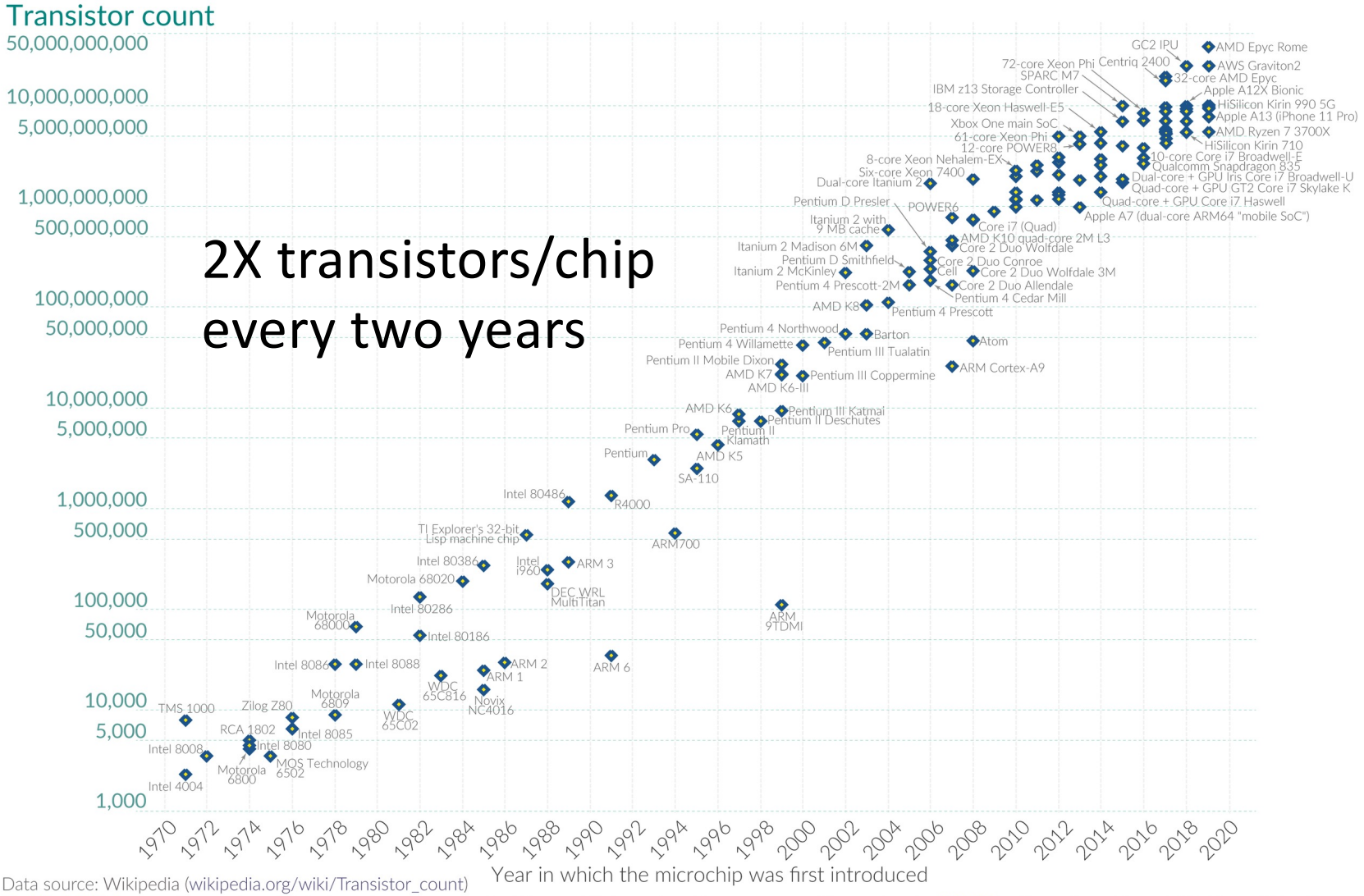
- Won't crash your machine
  - But program is likely to crash if something is wrong
- Uses system call wrappers provided by C library
- Uses "interesting" system calls
  - `fork()` spawn a new virtual CPU
  - `mmap()` instantiates a region in the process' address space

CPU Trends

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

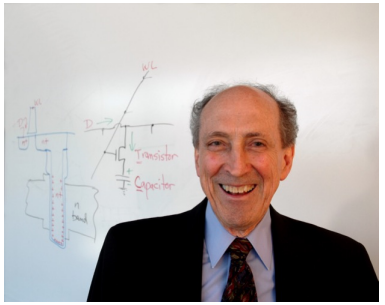
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world's largest problems.

End of Dennard Scaling



- **Dennard scaling:** As transistors get smaller, their density stays constant

- In every technology generation, the area and power consumption of individual transistors is halved
  - With twice the number of transistors, power consumption still stays the same

Dennard scaling broke down b/w 2005-2007

→ As we add more transistors, power consumption for for a chip with the same area increases

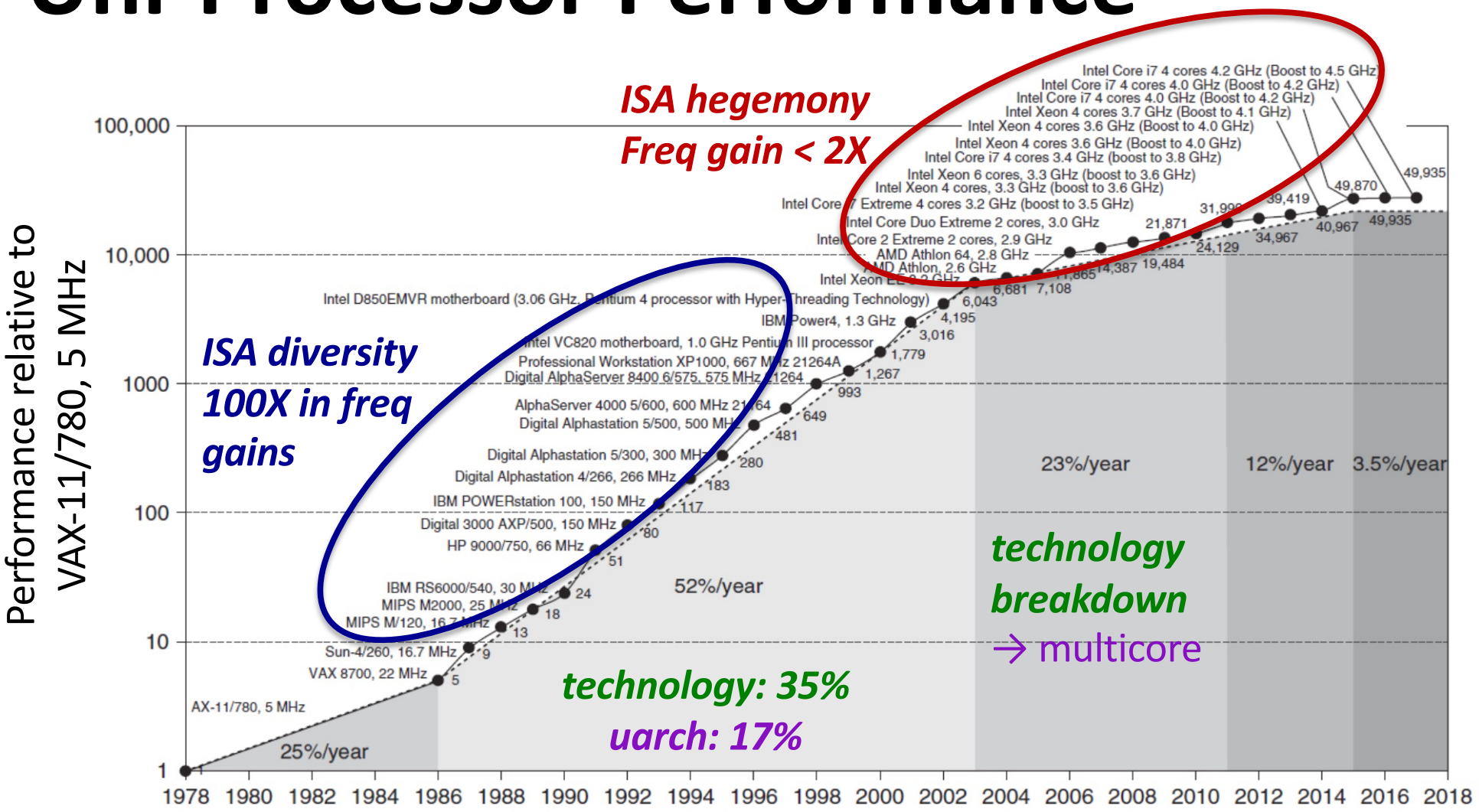
End of Dennard Scaling

Implication: Frequency cannot increase any further because that would make the power problem even worse → Industry shifted to multicores!

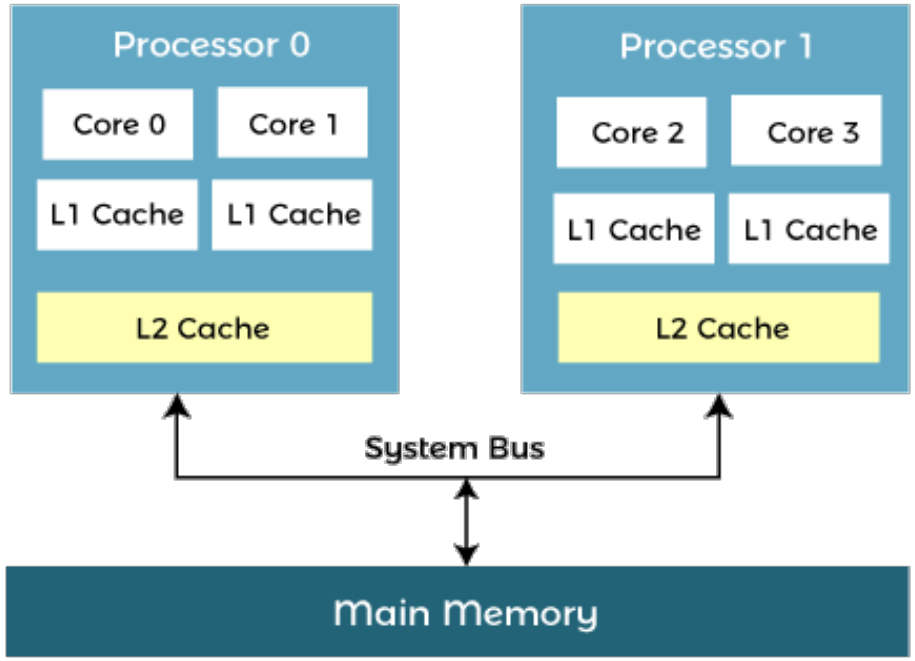
https://silvanogai.github.io/posts/dennard/

https://www.maketecheasier.com/why-cpu-clock-speed-isnt-increasing/

Uni-Processor Performance



Modern System



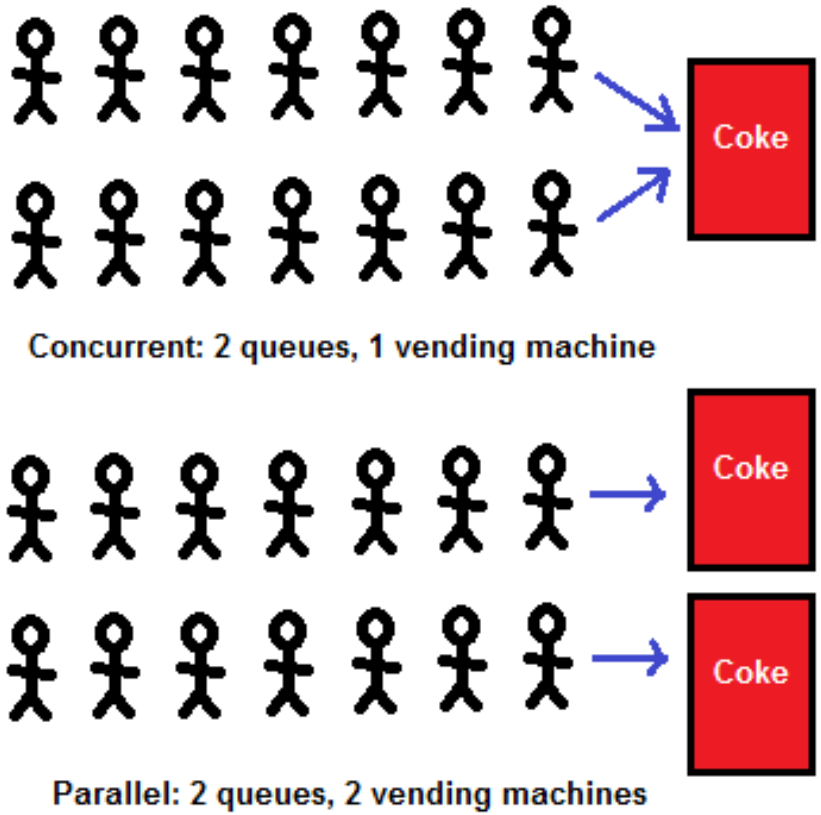
- Software must exploit parallelism for performance

Concurrency and Parallelism

- **Concurrency:** When the execution of two processes overlap in time. Think of a process as an instance of a program
  - Concurrency has always been important
  - It is a style of programming to solve a problem
    - Multiple users time-sharing a uniprocessor system
    - Handle an incoming request from the network, while the user is watching a video recording

- **Parallelism:** When two processes use dedicated resources (separate CPUs) to execute at the same time
  - Multicores have made parallelism critical

Concurrency and Parallelism Real-Life Example



Networking

- Web, social media, email, online games, all use the network
- We will learn the basics of client-server model
- Writing simple networking applications in C

Managing System Resources

- Many interesting debates in computer systems
  - RISC vs. CISC
  - Compiler vs. hardware exploitation of ILP
  - Manual (C++) vs. automatic memory management (Java)
- One more debate
  - Should "certain" hardware features be **exposed** to user-level applications or not?
  - **One camp:** User-level programmer possesses better knowledge of application logic than hardware or compiler or OS
    - They can "tune" the feature to make the optimal use of it
  - **Other camp:** They may also do something wrong
    - Leave it to the hardware or compiler or OS

Managing System Resources: Examples

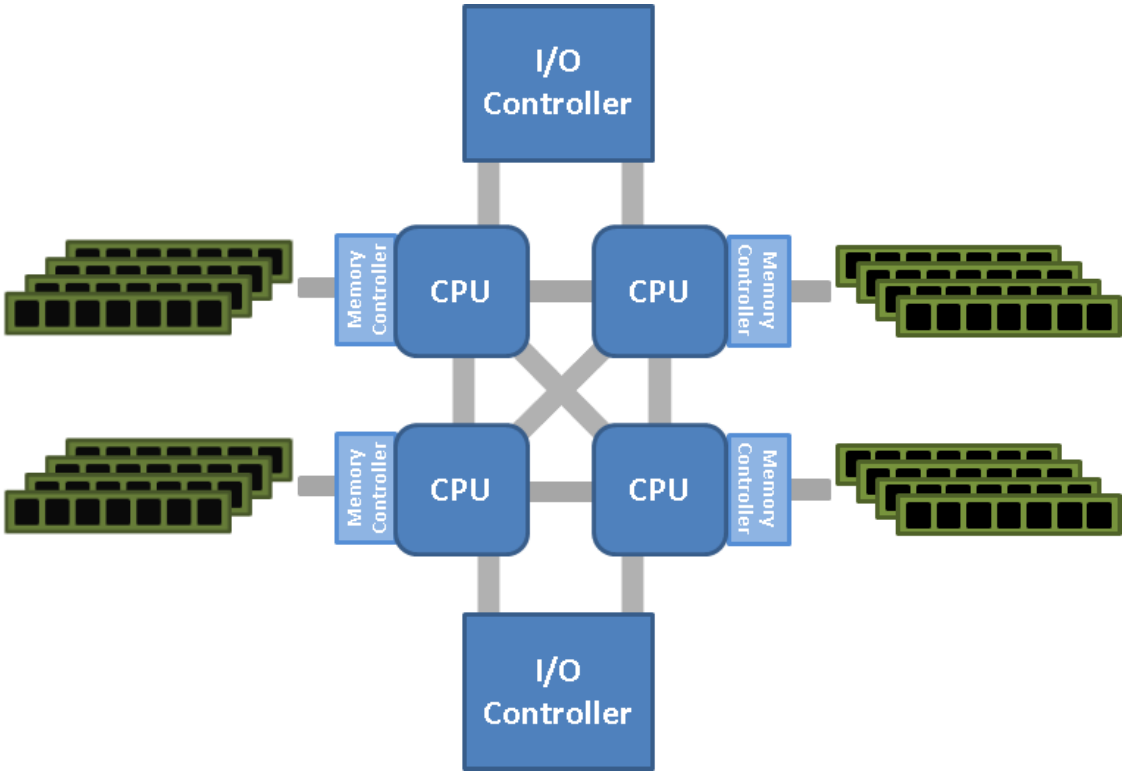
- CPU registers are exposed to software (OS and user-level)

- CPU caches are managed by hardware
  - We say caches are transparent to software

- A feature X is exposed to software, but OS utilizes the feature and user-level code has no way to access it
  - Feature X is transparent to user-level code
  - Feature X is visible to OS, or X is exposed to OS
  - Physical memory is an example (what? that's why 2310 exists!)

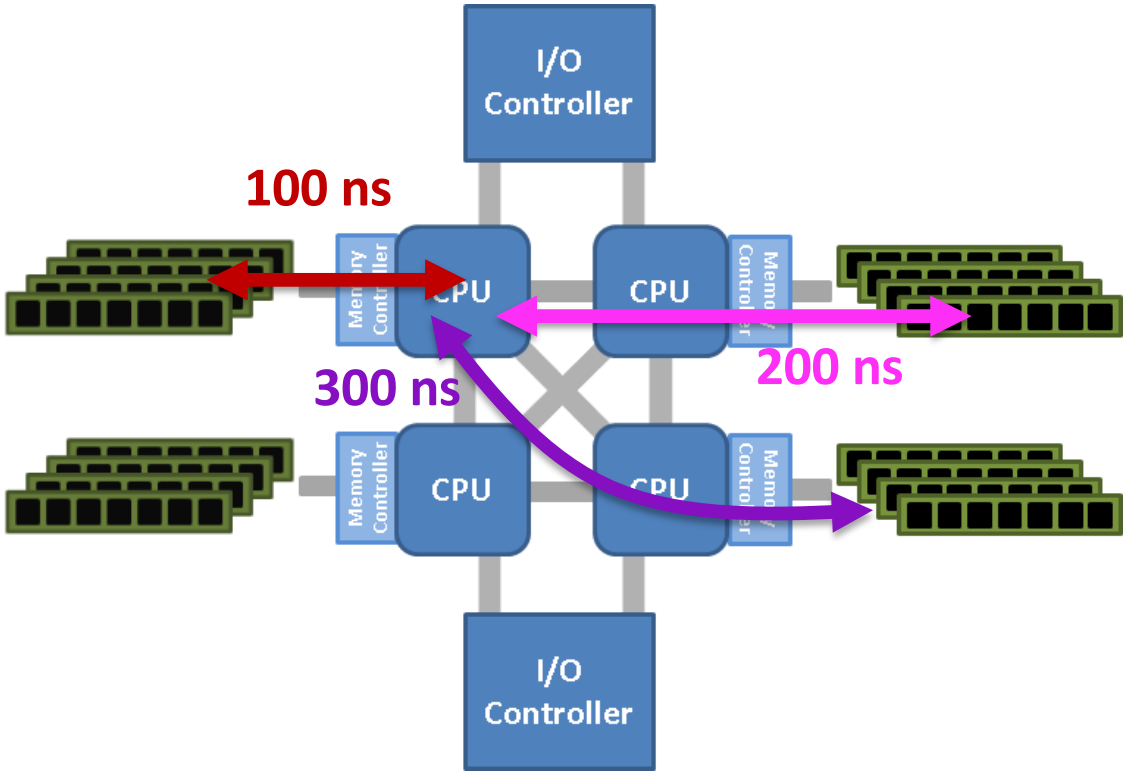
Modern NUMA System

Some memories are closer to the CPU, while others are far away. Wrong data placement can hurt performance.



User-space software must be aware of **N**on-**U**niform **M**emory **A**ccess (**NUMA**) architectures **(one view)**

Modern NUMA System

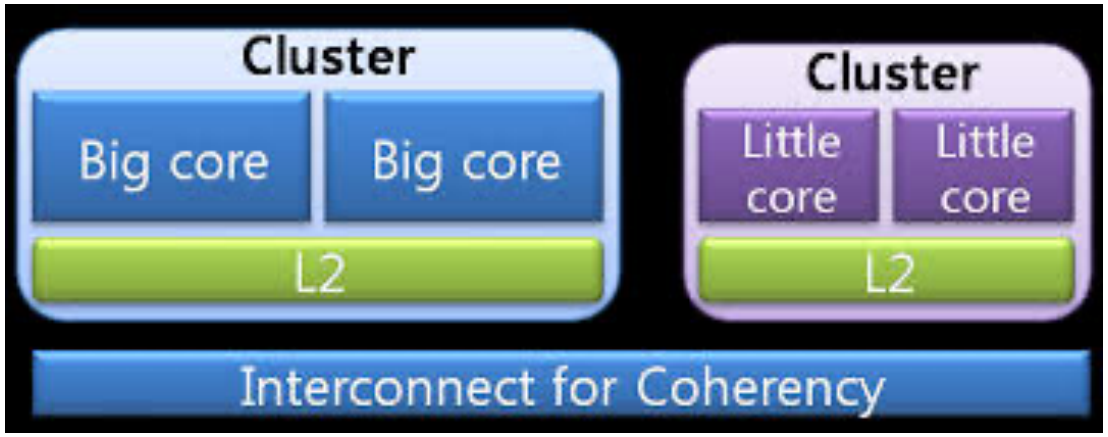


ns = nanoseconds = 1 billionth of a second (10^{-9} seconds)

User-space software must be aware of **N**on-**U**niform **M**emory **A**ccess (**NUMA**) architectures **(one view)**

Another system (cell phones)

ARM big.LITTLE



Power-hungry

Performance-driven

mission-critical tasks

Energy-efficient

background tasks

Why learn systems programming?

- **Key takeaway:** As a systems programmer, you can advise the OS (or any resource manager) to make the best use of the underlying hardware

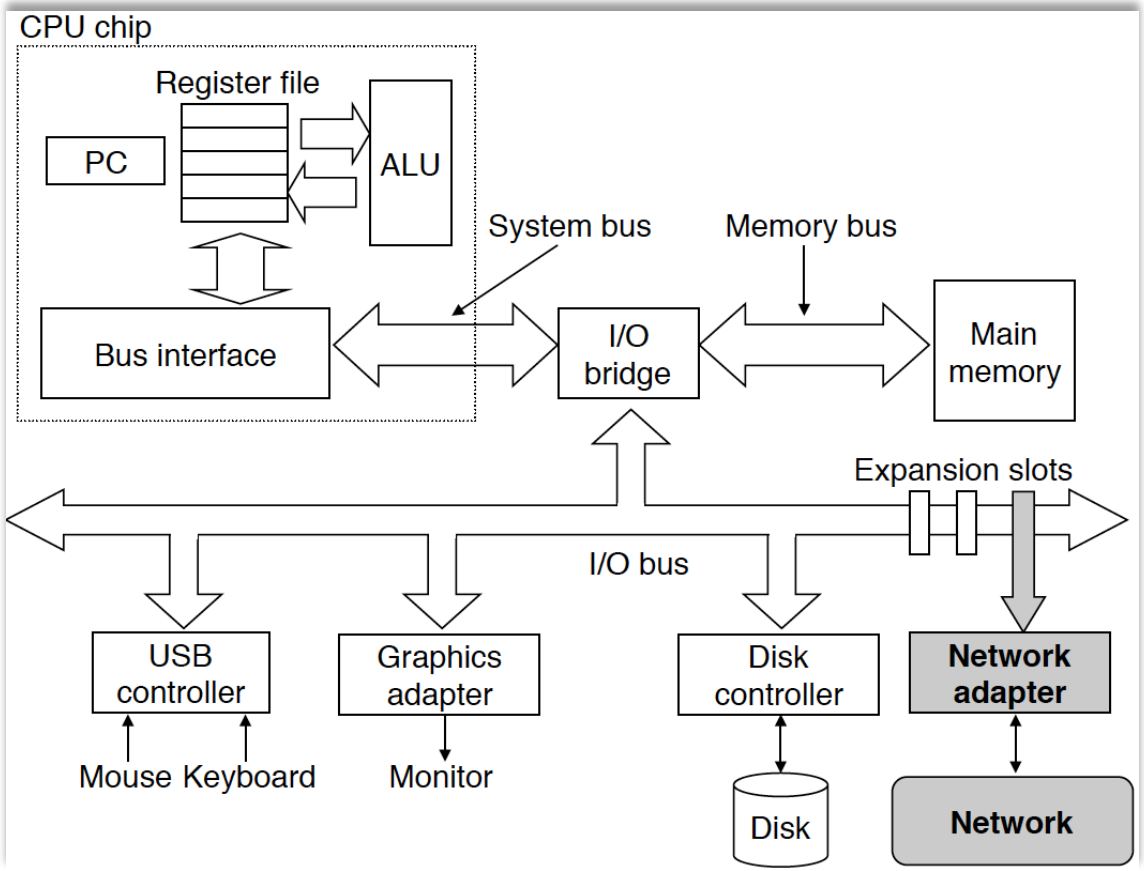
- We will teach you how you can build applications that hook up with the kernel and do just that and other interesting things

- We **won't** build: CPU, OS, compiler, here ....

More Examples of System Features

- NUMA
- Heterogeneous multicore processors (e.g., big.LITTLE)
- Persistent memory (e.g., Intel Optane Persistent memory)
- CPU-FPGA platforms
- Computational storage devices (CSD)
- Programmable network interface cards
- Hyperthreading
- Turbo boosting, low-power modes
- Single Instruction Multiple Data (SIMD) instructions
- ML accelerators
- Some recent additions to ISA for hardware cache mgmt.
- Dynamic voltage and frequency scaling (DVFS)
- Processing in Memory (PIM)
- Heterogeneous-ISA multicore processors
- Remote memory
- Single-ISA multicore processors
- Intel Cache Allocation and Monitoring Technology
- Memory-semantic solid-state drives (SSDs)
- CXL-based memory expansion
- Software defined storage…

COMP2310: Holistic View of Computer System



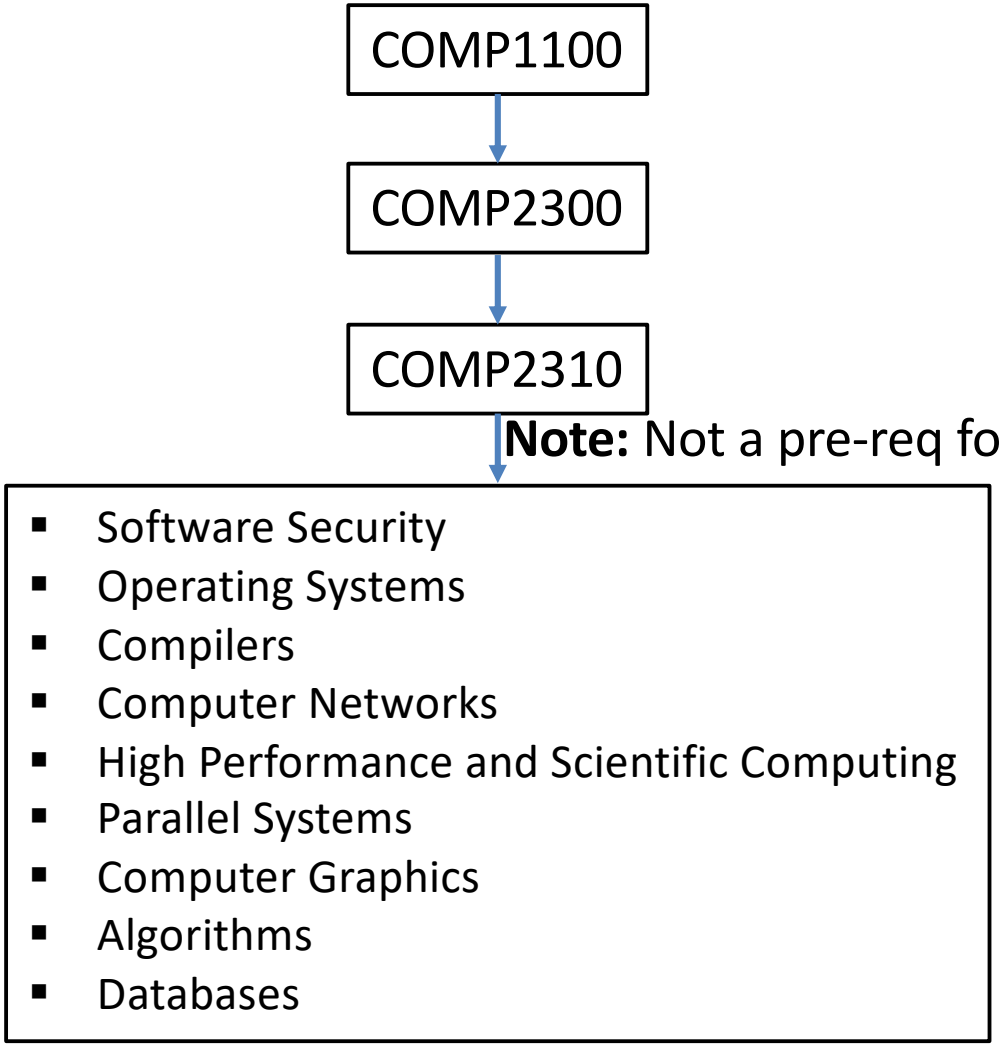
Course Perspective

- Most systems courses are Builder-Centric
  - Computer Organization (COMP2300), Microarchitecture
    - Build a CPU. Implement an ISA
  - Operating Systems (COMP3300)
    - Implement portions of operating system
  - Compilers (COMP3710)
    - Write compiler for a simple language
  - Computer Networks (COMP3310)
    - Implement and simulate network protocols

Course Perspective

- COMP2310 is programmer-centric
  - By knowing more about the underlying system, you can be more **effective** as a programmer
  - Enable you to
    - Write programs that are more **reliable** and **efficient**
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Things you will not see elsewhere or are required background knowledge
  - Not a course for **dedicated** hackers
    - We aim to **bring the hidden hacker** inside you!

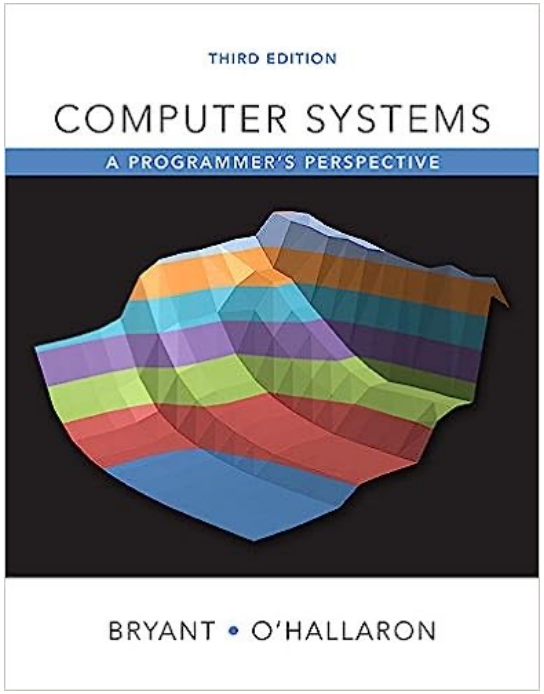
Role within CS Curriculum

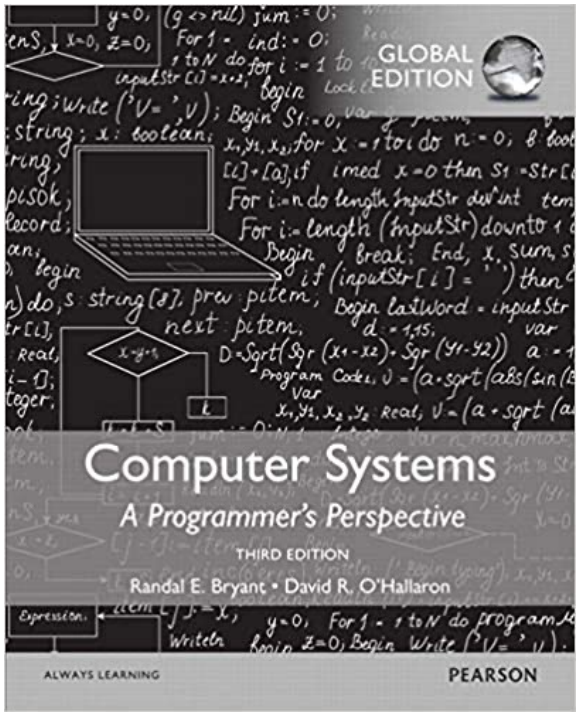


Content & Topics

Primary Textbook

- Textbook really matters for the course (problems, lectures, labs)
  - Textbook is not “just” a recommendation
- **Warning:** Paperback international version has “some” errors

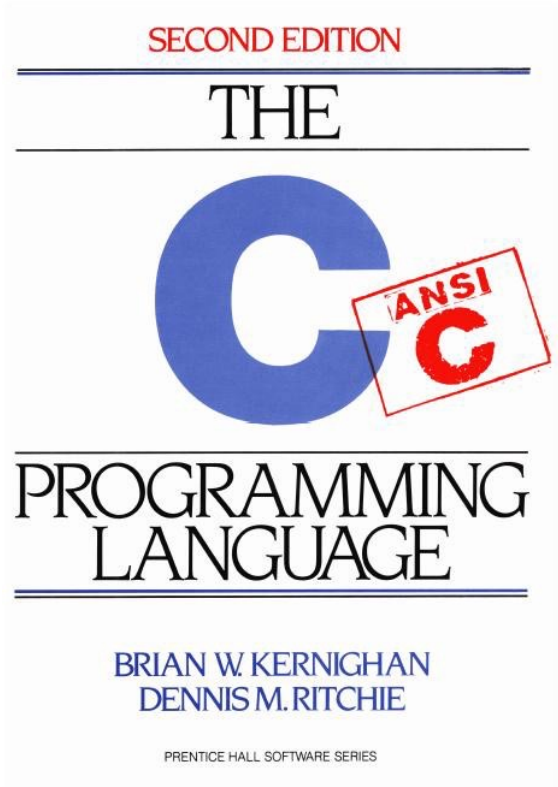




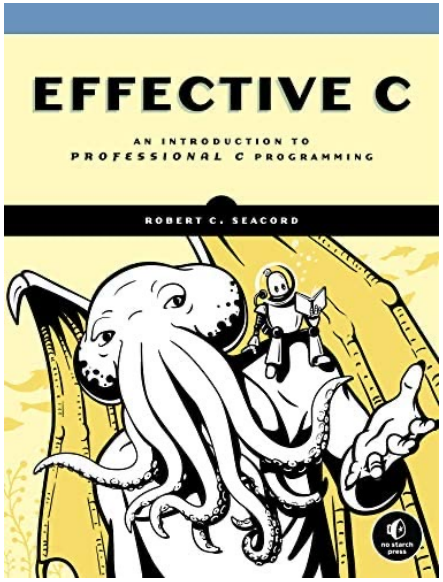
Useful Books on C (optional)

Kernighan & Ritchie, The C Programming Language, 2nd Edition

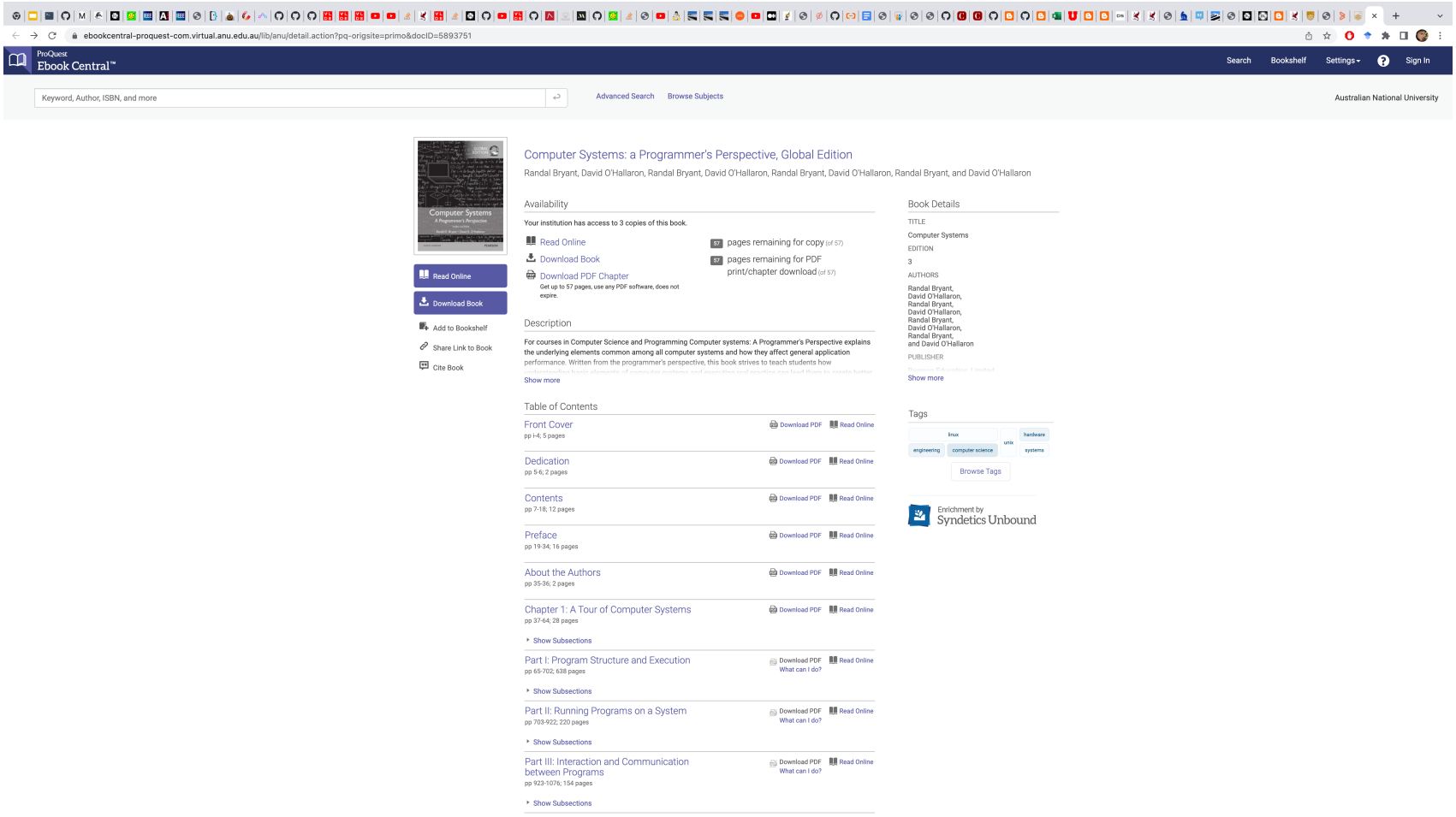
- "ANSI" (old-school) C
- Not too serious about things we now consider critical



Slightly advanced, more practical advice, modern



Textbook: Electronic edition available for ANU students

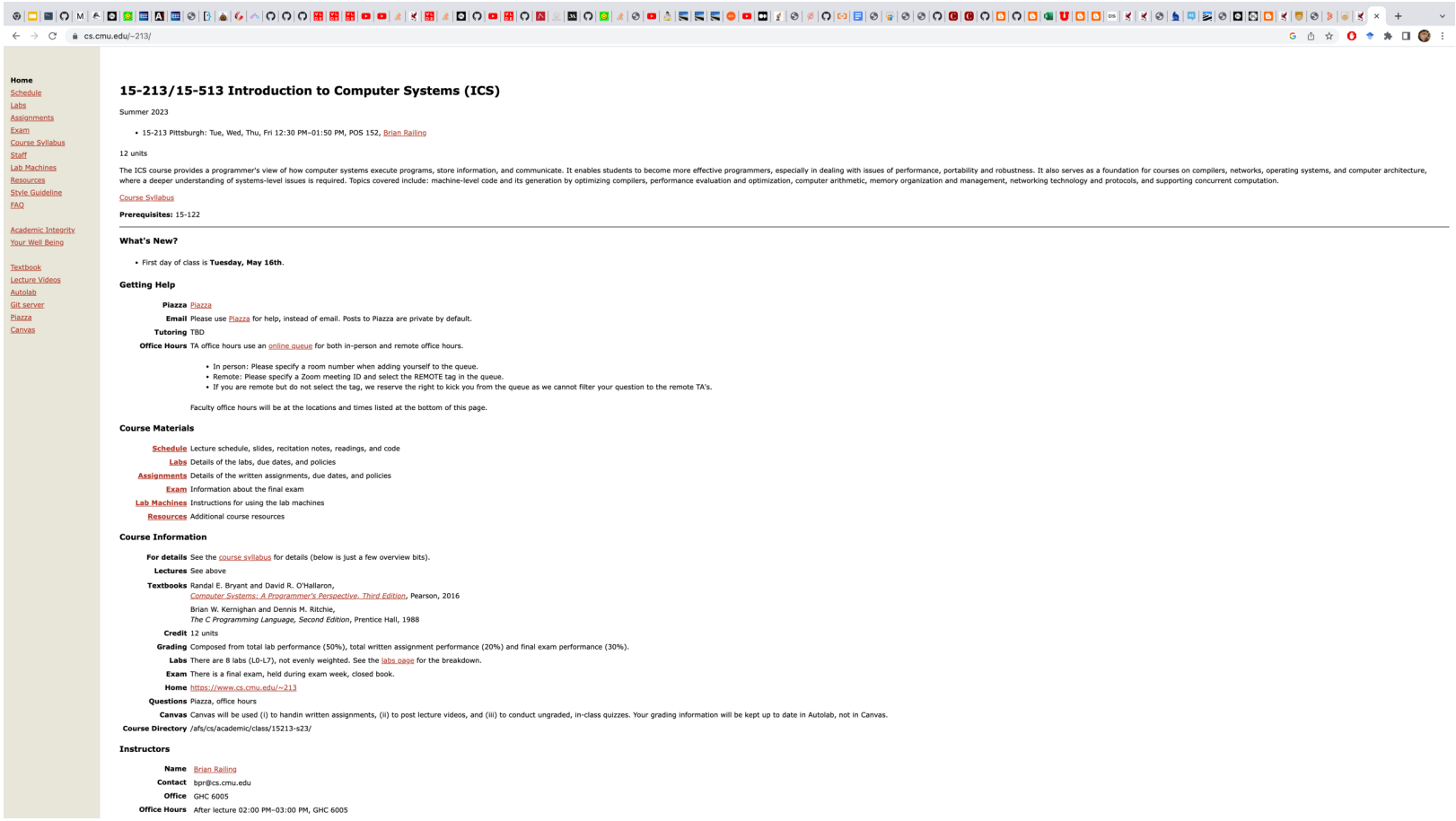


CMU 213

- Authors of the book at the **Carnegie Mellon University** created a course to accompany with the book
  - Lecture slides, problem sets, exams, labs, etc
  - (**Acknowledgement**) We use the material from the course

- We encourage you to explore the CMU course website
  - **Note:** Their course combines aspects of COMP2300 and COMP2310 into one course
  - **Their starting point:** COMP2300 starting point
  - Their CPU coverage is limited (programmer's perspective)
  - **Key Point:** Do not ignore COMP2310 & blindly follow CMU213

CMU 213


cs.cmu.edu/~213/
Home
Schedule
Labs
Assignments
Exam
Course Syllabus
Staff
Lab Machines
Resources
Style Guideline
FAQ
Academic Integrity
Your Well Being
Textbook
Lecture Videos
Autolab
Git server
Piazza
Canvas
15-213/15-513 Introduction to Computer Systems (ICS)
Summer 2023
15-213 Pittsburgh: Tue, Wed, Thu, Fri 12:30 PM–01:50 PM, POS 152, Brian Railing
12 units
The ICS course provides a programmer's view of how computer systems execute programs, store information, and communicate. It enables students to become more effective programmers, especially in dealing with issues of performance, portability and robustness. It also serves as a foundation for courses on compilers, networks, operating systems, and computer architecture, where a deeper understanding of systems-level issues is required. Topics covered include: machine-level code and its generation by optimizing compilers, performance evaluation and optimization, computer arithmetic, memory organization and management, networking technology and protocols, and supporting concurrent computation.
Course Syllabus
Prerequisites: 15-122
What's New?
First day of class is Tuesday, May 16th.
Getting Help
Piazza Piazza
Email Please use Piazza for help, instead of email. Posts to Piazza are private by default.
Tutoring TBD
Office Hours TA office hours use an online queue for both in-person and remote office hours.
In person: Please specify a room number when adding yourself to the queue.
Remote: Please specify a Zoom meeting ID and select the REMOTE tag in the queue.
If you are remote but do not select the tag, we reserve the right to kick you from the queue as we cannot filter your question to the remote TA's.
Faculty office hours will be at the locations and times listed at the bottom of this page.
Course Materials
Schedule Lecture schedule, slides, recitation notes, readings, and code
Labs Details of the labs, due dates, and policies
Assignments Details of the written assignments, due dates, and policies
Exam Information about the final exam
Lab Machines Instructions for using the lab machines
Resources Additional course resources
Course Information
For details See the course syllabus for details (below is just a few overview bits).
Lectures See above
Textbooks Randal E. Bryant and David R. O'Hallaron,
Computer Systems: A Programmer's Perspective, Third Edition, Pearson, 2016
Brian W. Kernighan and Dennis M. Ritchie,
The C Programming Language, Second Edition, Prentice Hall, 1988
Credit 12 units
Grading Composed from total lab performance (50%), total written assignment performance (20%) and final exam performance (30%).
Labs There are 8 labs (L0-L7), not evenly weighted. See the labs page for the breakdown.
Exam There is a final exam, held during exam week, closed book.
Home https://www.cs.cmu.edu/~213
Questions Piazza, office hours
Canvas Canvas will be used (i) to handin written assignments, (ii) to post lecture videos, and (iii) to conduct ungraded, in-class quizzes. Your grading information will be kept up to date in Autolab, not in Canvas.
Course Directory /afs/cs/academic/class/15213-s23/
Instructors
Name Brian Railing
Contact bpr@cs.cmu.edu
Office GHC 6005
Office Hours After lecture 02:00 PM–03:00 PM, GHC 6005

High-Level to Low-Level Translation

- C programming to x86-64 assembly
- Compilation steps
- Array allocation and access
- Heterogenous data structures
- Optimizations
- Security vulnerabilities

COMP2310 is not a C Programming Course

- Emphasis is on program transformation
- How does high-level code look in assembly?
- **Do compilers always do the right thing?**
- Programmers **WILL** write more efficient code if they have insight into transformation steps
- The power to reverse engineer object code and binaries
  - A.k.a. hackers! Security professionals' bread and butter

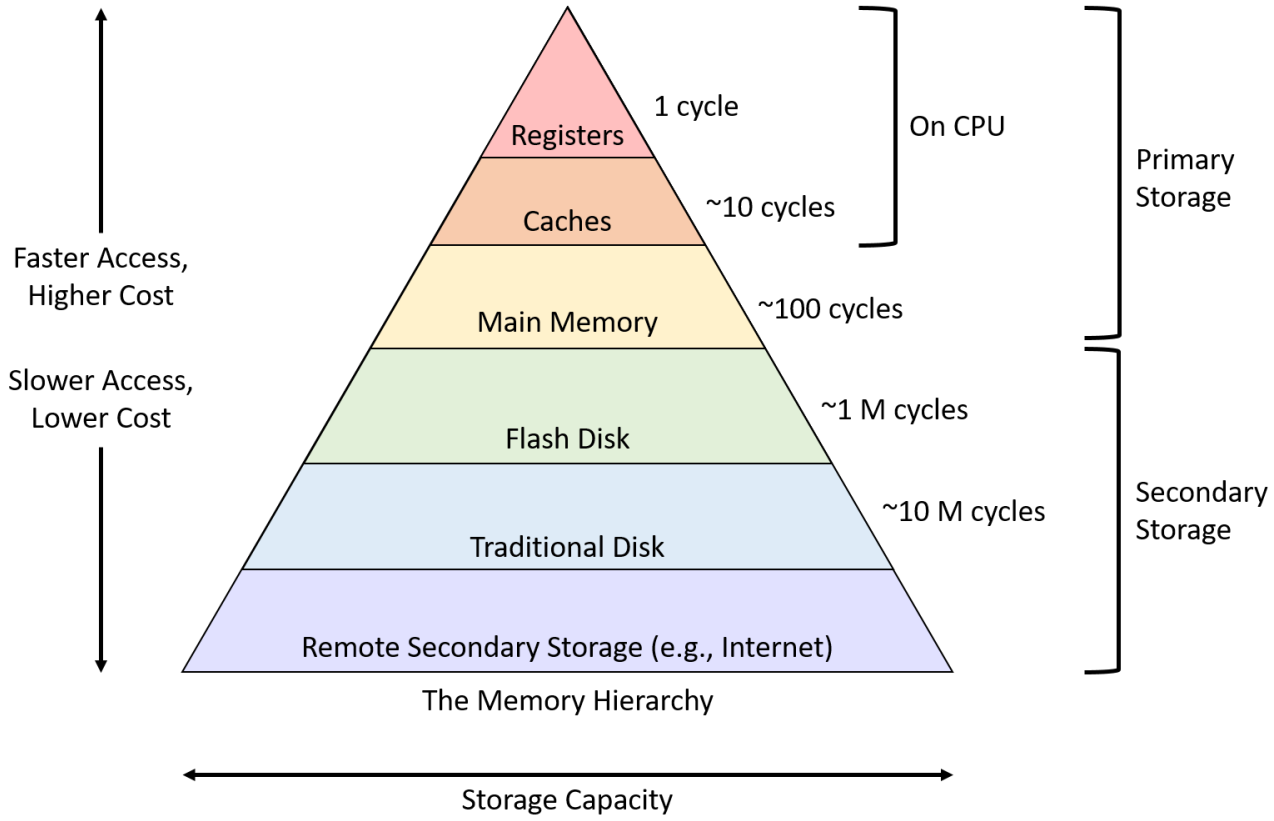
Qualified Answers to C Questions

- What are pros and cons of programming in C?
- Why should you NEVER use C in 2022? Why should everyone learn C (and then program in whatever language they like?
- Why is C insecure and what can be done about that?
- Why is Linux OS written in C? And many other datacenter software stacks?
- Why is C dominant in the **embedded domain**?

Exceptional Control Flow

- Processes
  - The illusion that each program has the entire CPU for its own use even though many programs might be co-running
- Exceptions and signals
- Address spaces
- How does Unix-like systems enable the process abstraction
- Linux API and its use. (Key idea: not implementation of API)

Memory Hierarchy



Linking

- The process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory executed

- **Topics**
  - Static and dynamic linking
  - Object files, relocatable code
  - Symbols, symbol resolution, symbol tables
  - Position independent code
  - Library interpositioning

Virtual Memory

- Illusion that a program has the entire physical address space for its own use even though many programs may be co-running

- **Topics**
  - Address translation
  - Translation-lookaside buffers
  - Page tables and page fault
  - Dynamic storage allocation
  - Garbage collection

System-Level I/O

- Managing storage device (e.g., disk) as a reliable and easy-to-use persistent storage resource

- **Topics**
  - How to use the **Linux filesystem API**
    - Not a course for learning to implement filesystems
    - Appropriate API usage is *an art* in its own right!
  - System call and memory-mapped I/O
  - Includes aspects of virtual memory

Network Programming

- High-level and low-level I/O contd., with extension to network programming
- Very similar API for storage and networking I/O
- Internet services, web servers

Concurrent Programming

- Concurrent server design
- Threaded server versus process-based server
  - Last year's assignment's key theme
- I/O **multiplexing** with select
- Some aspects of parallel programming
  - Stepping-stone to parallel systems course

Big Data Frameworks

- Big data frameworks today process very large datasets

- They stress every aspect of key COMP2310 topics
  - Memory
  - Storage
  - CPU
  - Concurrency and networking

Big Data Frameworks

- Typical data processing framework you can **aim to** implement after COMP2310

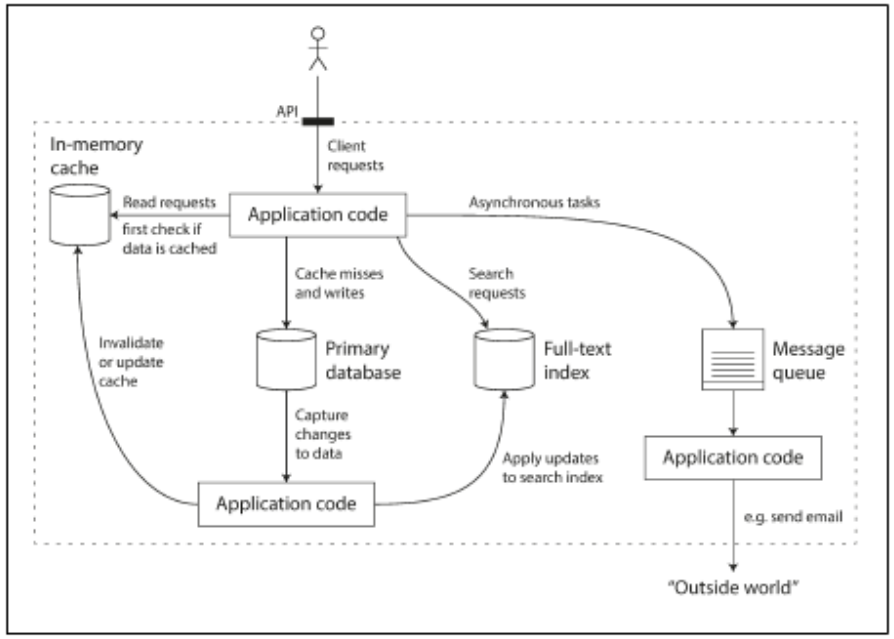


Figure 1-1. One possible architecture for a data system that combines several components.

- *From book: Designing Data Intensive Applications, Page 5*

Assessment

Checkpoint 1

- Reverse engineering x86-64 object code
- Proficient in low-level C programming
  - pointers, string manipulation, etc
- Week 4 – **during Labs**

Assignment 1

- Implementing a memory allocator or malloc() from scratch
- Open-ended extensions on top of a base spec
- Released in week 6

Checkpoint 2

- Concurrency fundamentals
- Pthread synchronization
- Week 9 **- during labs**

Assignment 2

- Related to networking with aspects of concurrency
- Last year assignment was a web proxy with a user-level cache
- Some changes this year but similar in inspiration
- Released Week 11

Quiz 1

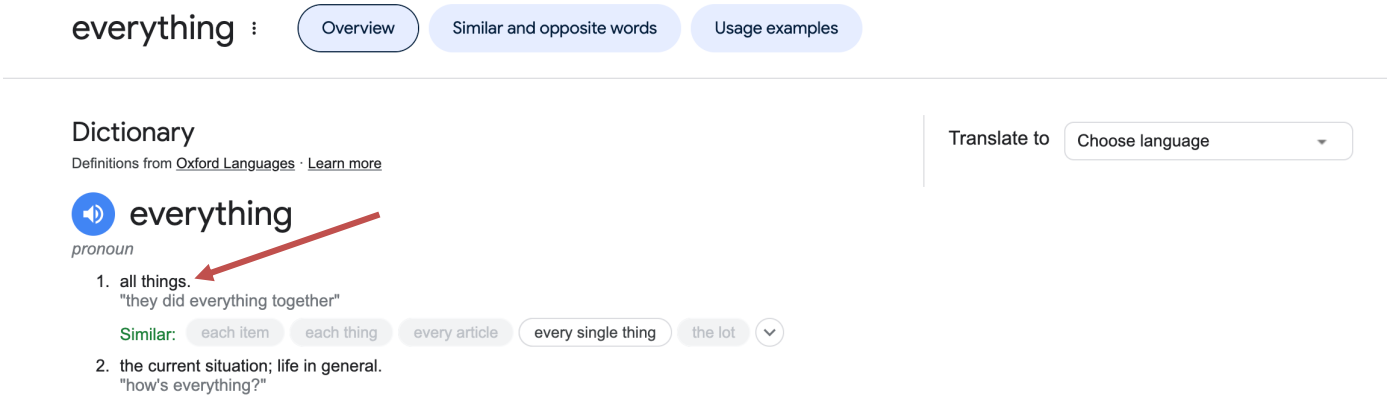
- Processes and signals
- Tests first four weeks of content
- Week 6 - during labs

Quiz 2

- Memory allocation, virtual memory, cache, storage, some database concepts
- Tests weeks 5 – 10 content
- Week 11 - during labs

Final Exam

- Everything!



  - Inclusive of week 12
  - Every lab
  - Every slide

Assessment Schedule

- Quizzes in labs
- Checkpoints in labs
- ~ 2 weeks for assignments

Assessment Item	Weighting	Released	Due Date
Checkpoint 1	7.5%	Week 4 (during labs)	Week 4 (during labs)
Quiz 1	5%	Week 6 (during labs)	Week 6 (during labs)
Assignment 1	15%	Week 6	14/09 11:59 PM
Checkpoint 2	7.5%	Week 9 (during labs)	Week 9 (during labs)
Quiz 2	5%	Week 11 (during labs)	Week 11 (during labs)
Assignment 2	15%	Week 11	02/11 11:59 PM
Final Exam	45%	-	Final Exam Period

70% of COMP2310 assessment will be invigilated

Breakdown

- Checkpoint 1 (7.5%)
- Checkpoint 2 (7.5%)

- Quiz 1 (5%)
- Quiz 2 (5%)

- Assignment 1 (15%)
- Assignment 1 (15%)

- Final Exam (45%)

- **70% of COMP2310 assessment tasks will be invigilated.**
- **GenAI usage undermines learning goals**
- **GenAI usage is not allowed in COMP2310 assessment tasks, it will be detected and reported as a breach of Academic Integrity**

Using Generative AI responsibly

AI can be useful to get a broad overview of a topic. But it shouldn't be used to generate content or to express ideas.

1. **Check the Class Summary**
   Your class summary or your convenor will have information on the use of AI in your course. Different assessment tasks may have different guidelines.

2. **Maintain Academic Integrity**
   Do not paste content generated by AI and claim it as your own. Write in your own words.

3. **Protect Your Details**
   Do not enter personal or private details into prompts. The ANU is licensed to use Microsoft Copilot (https://copilot.microsoft.com), which has data protecting features.

4. **Verify and Critique Outputs**
   AI can produce 'fabrications' or 'hallucinations' including incorrect information or made-up references. Always refer to real, reliable sources.

5. **Avoid Translation**
   Using AI to translate your work reduces opportunities for you to practice and receive feedback, and it may produce strange writing. AI can be used to clean up grammar (eg. Grammarly).

Find out more about responsible AI use at the ANU page on Best Practice Principles (https://www.anu.edu.au/students/academic-skills/academic-integrity/best-practice-principles/guide-for-students-best).

Admin & Logistics

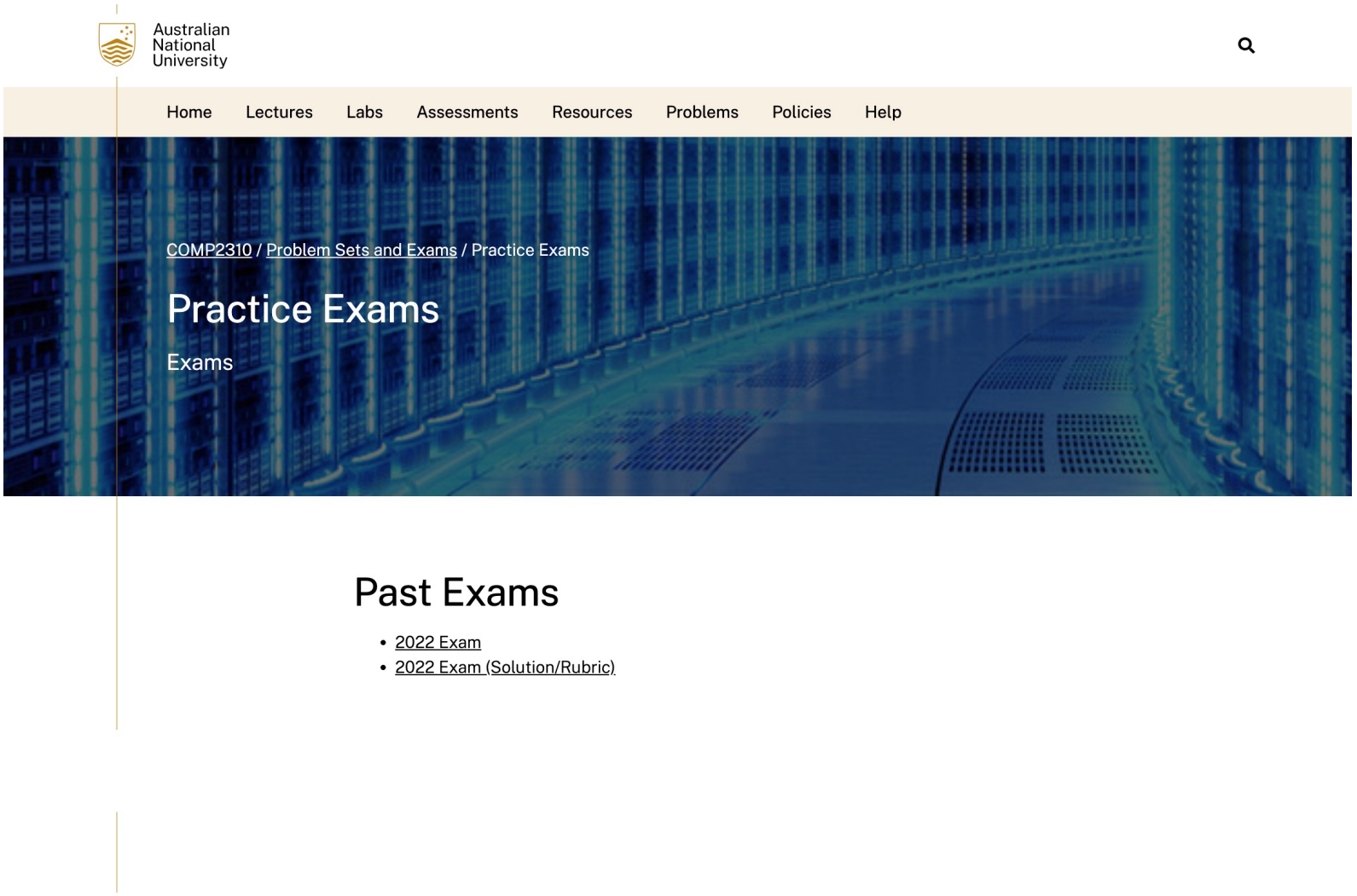
Succeeding in this course

- Pay attention to lecture content
- Finish all the labs
- Read the textbook
- Submit all assessments

Assessment Difficulty

- Assignments are manageable if you start early
- Possibly the most **“adventurous”** exam of your ANU journey
- **Check out the past year’s exam and rubric on the website**
- If you spend many hours finishing the first two labs and struggle with checkpoint 1
  - Make sure you finish COMP2300 first
  - Reconsider taking this course if it’s not compulsory
  - Focus on the key points in the last slide

Past Exam

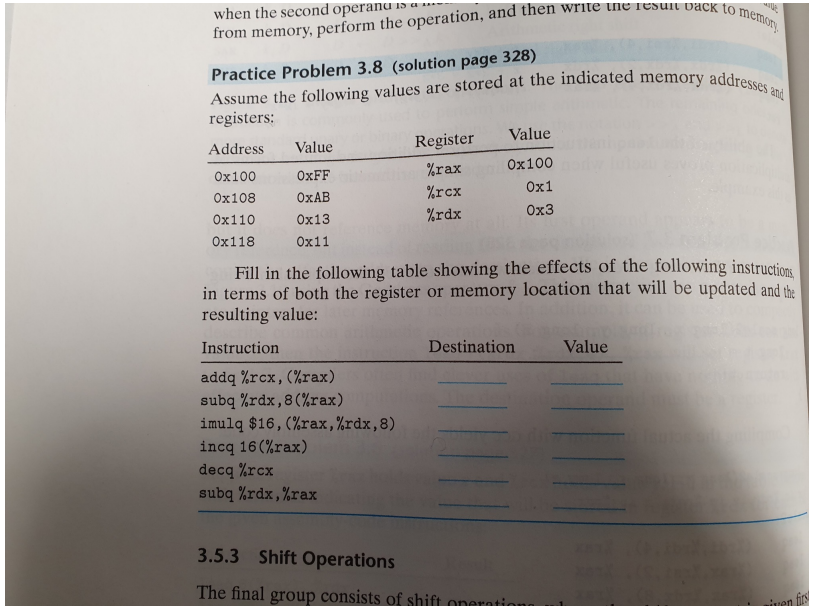

Australian National University
Home
Lectures
Labs
Assessments
Resources
Problems
Policies
Help
COMP2310 / Problem Sets and Exams / Practice Exams
Practice Exams
Exams
Past Exams
2022 Exam
2022 Exam (Solution/Rubric)

Readings: Book Chapters

Chapters	Topics/Weeks	2310 Coverage
1	COMP2300	Recommended
2	COMP2300	Not required
3	Weeks 1 – 2	Full except 3.11
4	COMP2300	Not Required
5	Weeks 1 – 2	Selected
6	Week 3	**Full**
7	Week 12	**Full**
8	Week 4	**Full**
9	Weeks 5 – 6	**Full**
10	Week 7	**Full**
11	Week 9 – 10	**Full**
12	Week 11	**Full**

Practice Problems

- Try practice questions in book (answers in the book)



when the second operand is a [illegible] from memory, perform the operation, and then write the result back to memory.

Practice Problem 3.8 (solution page 328)

Assume the following values are stored at the indicated memory addresses and registers:

Address	Value
0x100	0xFF
0x108	0xAB
0x110	0x13
0x118	0x11

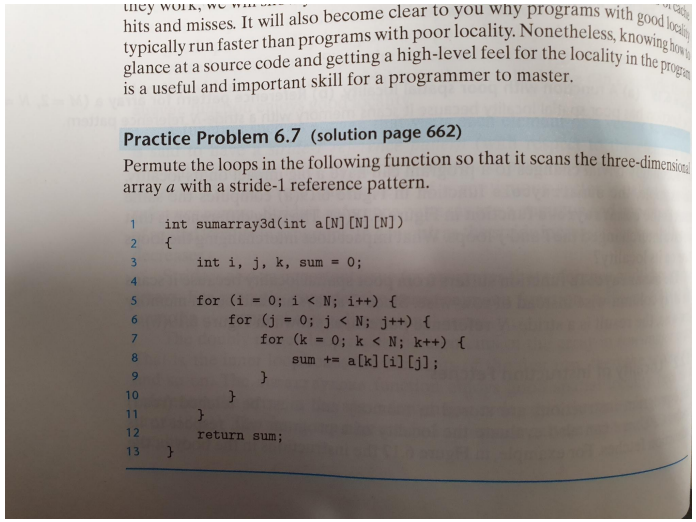
Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Fill in the following table showing the effects of the following instructions, in terms of both the register or memory location that will be updated and the resulting value:

Instruction	Destination	Value
addq %rcx,(%rax)		
subq %rdx,8(%rax)		
imulq $16,(%rax,%rdx,8)		
incq 16(%rax)		
decq %rcx		
subq %rdx,%rax		

3.5.3 Shift Operations

The final group consists of shift operations [illegible]



they work, we will [illegible] hits and misses. It will also become clear to you why programs with good locality typically run faster than programs with poor locality. Nonetheless, knowing how to glance at a source code and getting a high-level feel for the locality in the program is a useful and important skill for a programmer to master.

Practice Problem 6.7 (solution page 662)

Permute the loops in the following function so that it scans the three-dimensional array a with a stride-1 reference pattern.

```
1   int sumarray3d(int a[N][N][N])
2   {
3       int i, j, k, sum = 0;
4
5       for (i = 0; i < N; i++) {
6           for (j = 0; j < N; j++) {
7               for (k = 0; k < N; k++) {
8                   sum += a[k][i][j];
9               }
10          }
11      }
12      return sum;
13  }
```

Cheating/Plagiarism

- Copying code, retyping by looking at a file
- Describing a solution to someone else so they can then type
- Searching the web for solutions to quiz or assignment
  - Last year's iteration of COMP2310, other universities' solutions in English or another language
- Copying from a github repository with minor or no modification
- Use of AI to generate your code

Cheating/Plagiarism

- Helping others by supplying code
- Debugging their code
- Telling them how to put together different code snippets to reach a working solution

Not Cheating

- Explaining how to use a tool
  - GDB, GCC, Valgrind, Editor, VSCode, Shell
- High level discussions
  - Not pseudo-code, not specific algorithms
- Using code supplied with the book
- Using Linux manpages
- Do not do this: COMP2310 malloc solution 2024

Cheating Consequences

- Action to uphold integrity begins at the time of discovery (not at the end of course)
- Last year, I read all submitted code from every student (but we will use automated tools as well)
- Some students were unable to pass the course due to academic integrity
- **Bottomline:** We want you to get the experience of dealing with systems programming issues from scratch!

Tutorials/Labs

- Labs are a critical component of this course (one every week)
- Handout will be posted on the website "Labs" before each lab
- First **2** labs
  - Becoming comfortable in C: pointers, bit-level manipulation, `malloc()/free()`
  - Lab 3 is assessed as the first checkpoint (no help from tutors)
- Lab 4
  - Process API and signal handling
- Lab 5 – 6
  - *We will teach you to write a basic memory allocator, i.e., implementation of* `malloc()`
- Lab 6 (assignment 1 week)
- Lab 7
  - **Storage I/O**
- Lab 8
  - Concurrency fundamentals
- Lab 9
  - Concurrency & Networking (sockets API)
- Labs 10, 11
  - Assignment 2, threads & concurrency (`pthreads`)

Systems Foundation

Networking & concurrency

Assignment Submission

- Extensions will be granted on a per-request basis
  - Via the extension app

- Assignment submissions are handled via Gitlab
  - You will learn more about it in the labs
  - Make a habit of using Git properly
  - Push often, always pull the latest

Each student submits their own work. No groups.

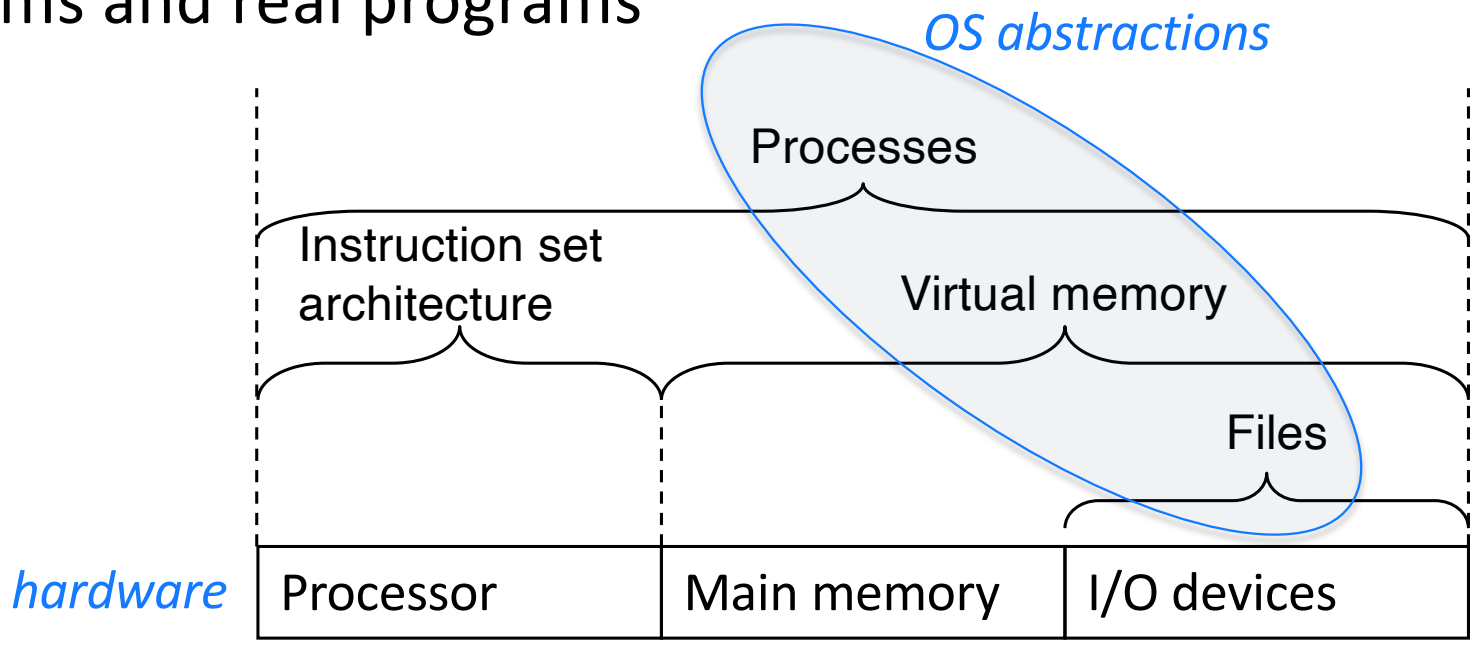
Note that: Student + AI = Group

Rough Plan for Lectures

1. Overview and x86 assembly
2. Optimizations and security implications (x86 as a vehicle)
3. Memory hierarchy
4. Processes and signals (abstraction for CPU, memory, I/O)
5. Virtual Memory (abstraction for main memory)
6. Dynamic memory allocation (memory allocator design)
7. Storage and File I/O (abstraction for I/O devices)
8. Networking
9. Concurrency
10. Linking
11. Revision (time permitting!)

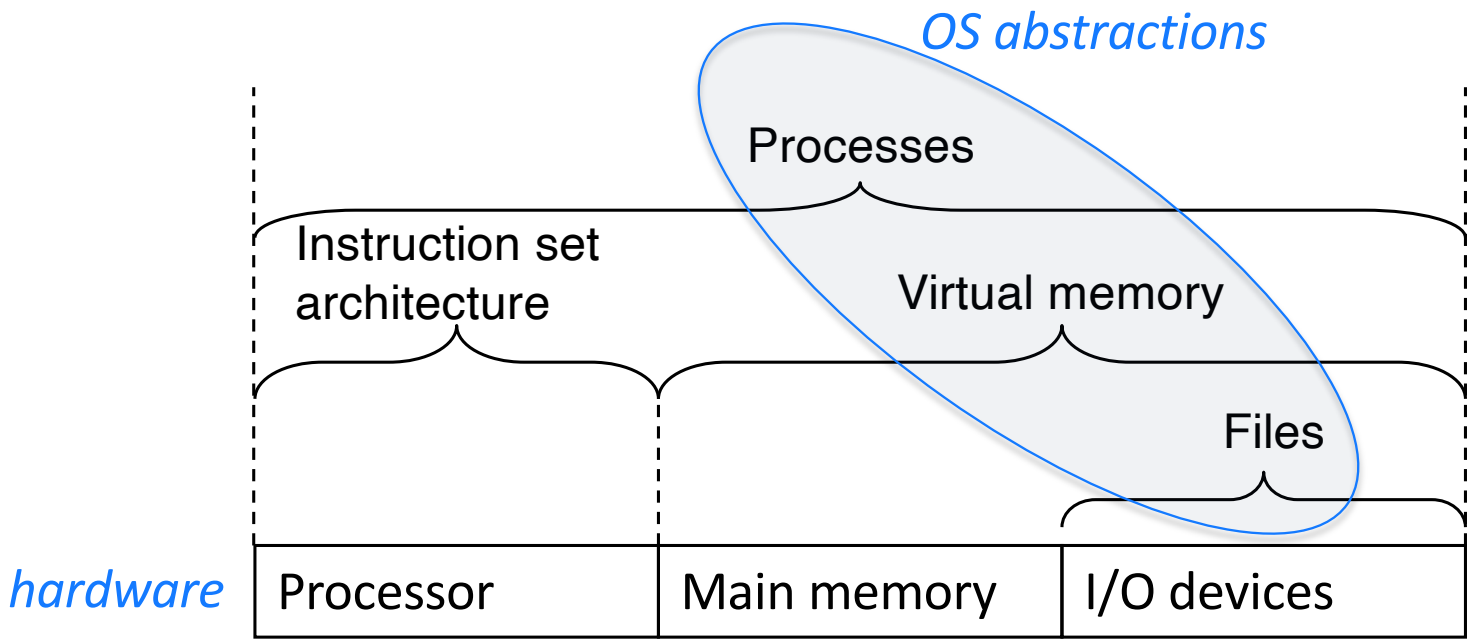
Course Organization (1)

- First 6 weeks lay the **foundation** of systems programming
  - They deal with CPU and memory virtualization
  - CPU and memory as a raw resource is not **safe** for multi-user systems and real programs



Course Organization (2)

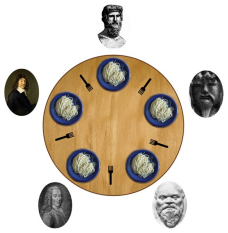
- Next week deals with abstraction for storage I/O devices
  - Without storage and files, no **serious** application can work



Course Organization (3)

- And finally, every system must communicate with other systems (world wide web)
  - We move to networking
  - Networking is also I/O so an extension of storage I/O

- A networked application must deal with multiple producers and consumers of information
  - In comes concurrency!



- Finally, we end the course with linking (how large programs that use external libraries are compiled efficiently and safely)
  - Ideally fits in week 4, but we need to approach memory early

Welcome to COMP2310!