CSCI 2400

Computer Systems

Fall 2017

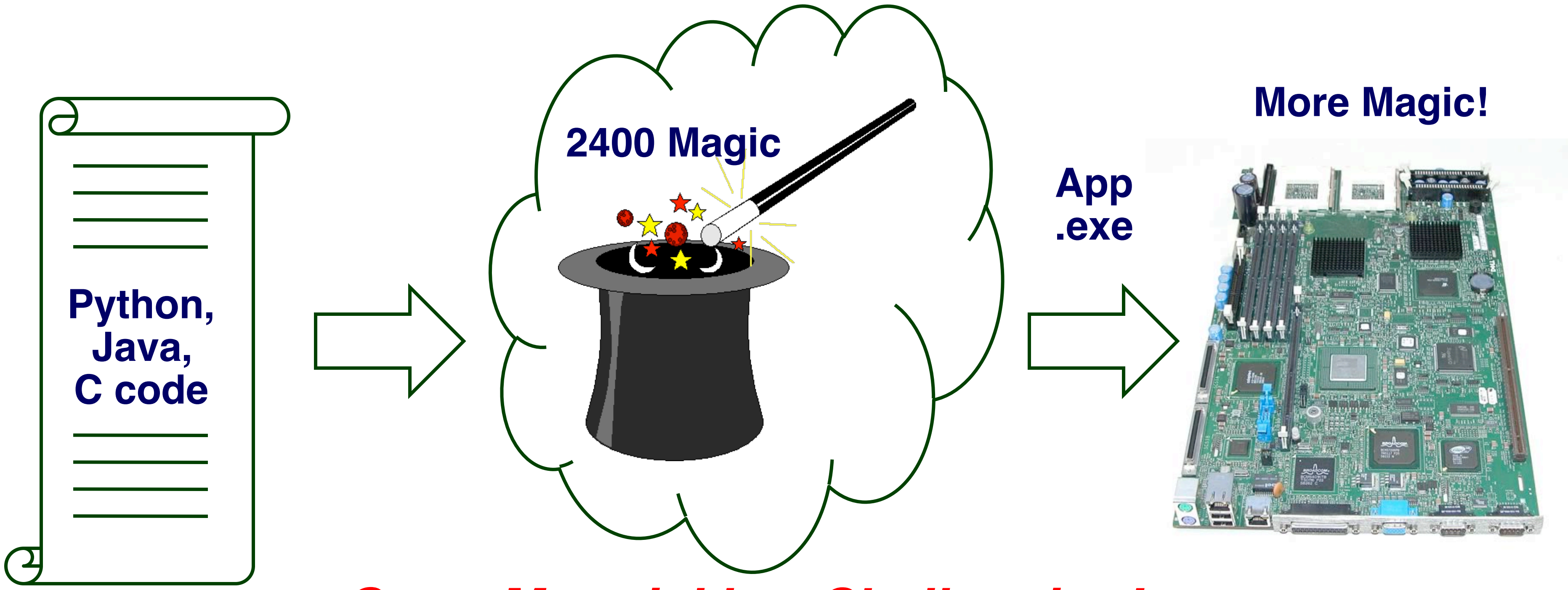
Professor Rick Han
Department of Computer Science
University of Colorado at Boulder

rhan@cs.colorado.edu
http://www.cs.colorado.edu/~rhan

Goal of 2400:

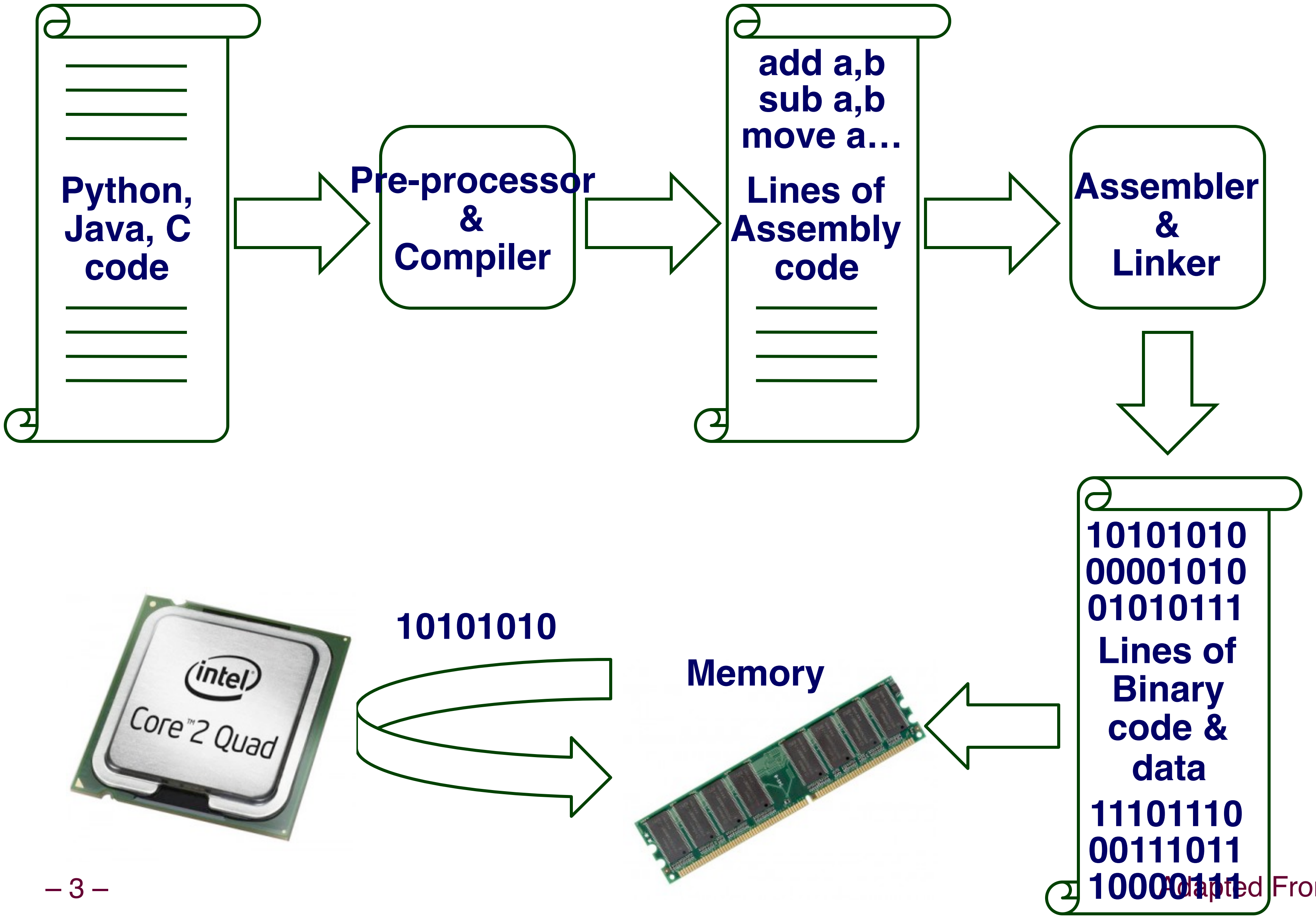
- **Learn how a computer works!**

More precisely, how software executes on modern computer hardware

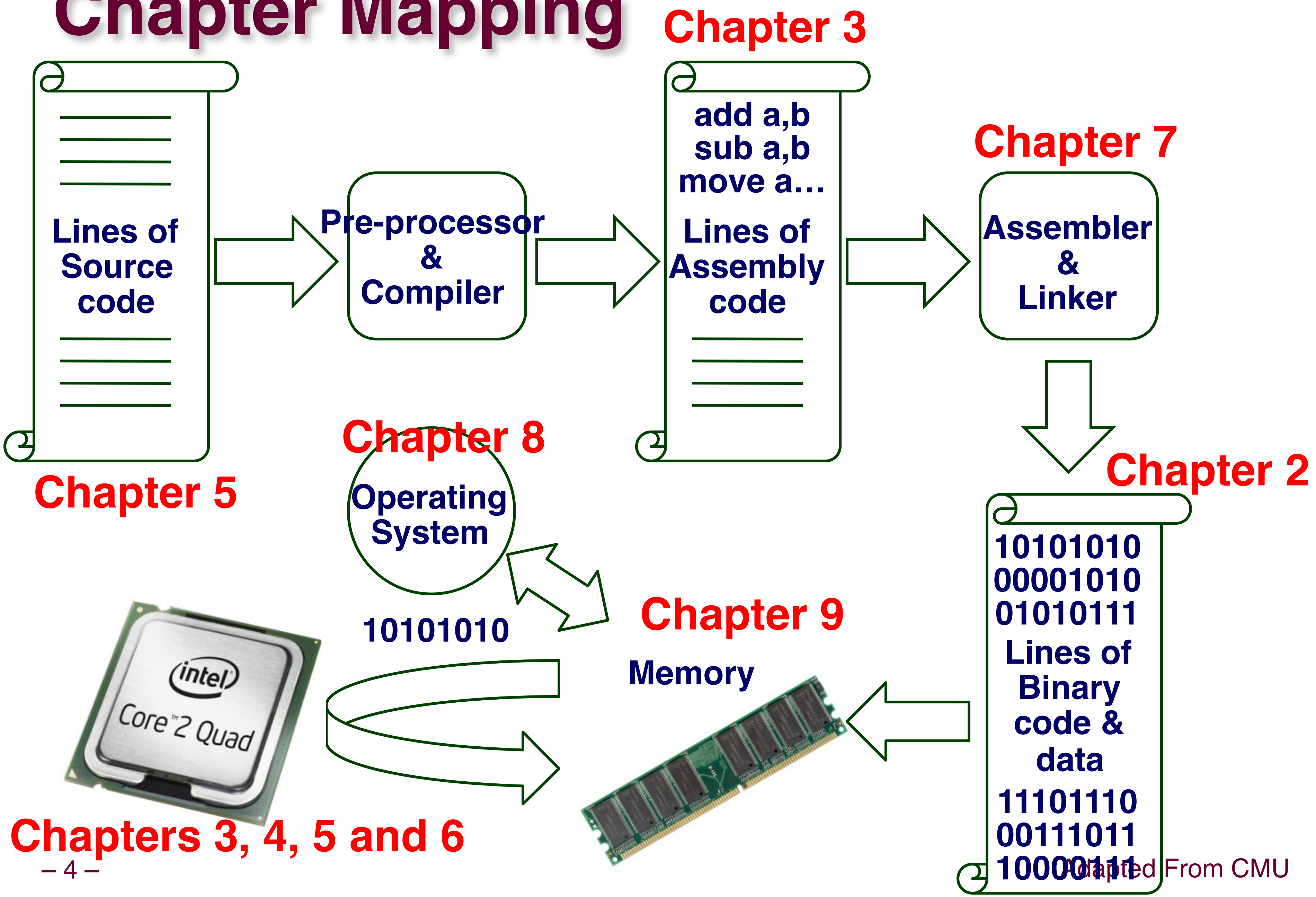


Great Material but Challenging!

2400 In a Nutshell



Chapter Mapping



Approximate Timeline

Week	Topics	Due
1	Ch 1-2: Intro, Two's Complement	
2	Ch 2: Integer Arithmetic	
3	Ch 3: Assembly memory, arithmetic	Data Lab
4	Ch 3: Assembly control flow, loops	MT 1
5	Ch 3: Assembly: stacks, arrays	
6	Ch 3: Assembly: buffer overflow	Bomb Lab
7	Ch 2: Floating point	
8	Ch 4: ISA, Pipelining	MT 2
9	Ch 4-5: Pipelining, Optimization	Attack Lab
10	Ch 5: Performance Optimization	
11	Ch 6: Caching	Performance Lab
12	Ch 6: Caching, Storage	MT3
13	Ch 8: Exceptions, Signals	
14	Ch 9: Virtual Memory	Shell Lab
15	Ch 7: Linking	
Finals	Final Exam	Final Exam

Announcements

- Let’s go to moodle.cs.colorado.edu, the primary rendezvous point for CS 2400
  - Sign up for an account, using ‘systemsrocks17’ as the enrollment key
  - Walk through the syllabus

Announcements

- C Assessment quiz due Friday by 12 pm noon
  - Make sure you understand C pointers
  - Go to the moodle and do the C Assessment quiz by the deadline this week
  - Suggested C refresher tutorial/study links:
    - http://www.tutorialspoint.com/cprogramming/
    - http://learn-c.org/
  - Unlimited # of attempts
  - If your score <= 70%, you need to attend Friday's C tutorial
  - More announcements on this quiz will be made on the moodle
  - Strongly recommended but optional

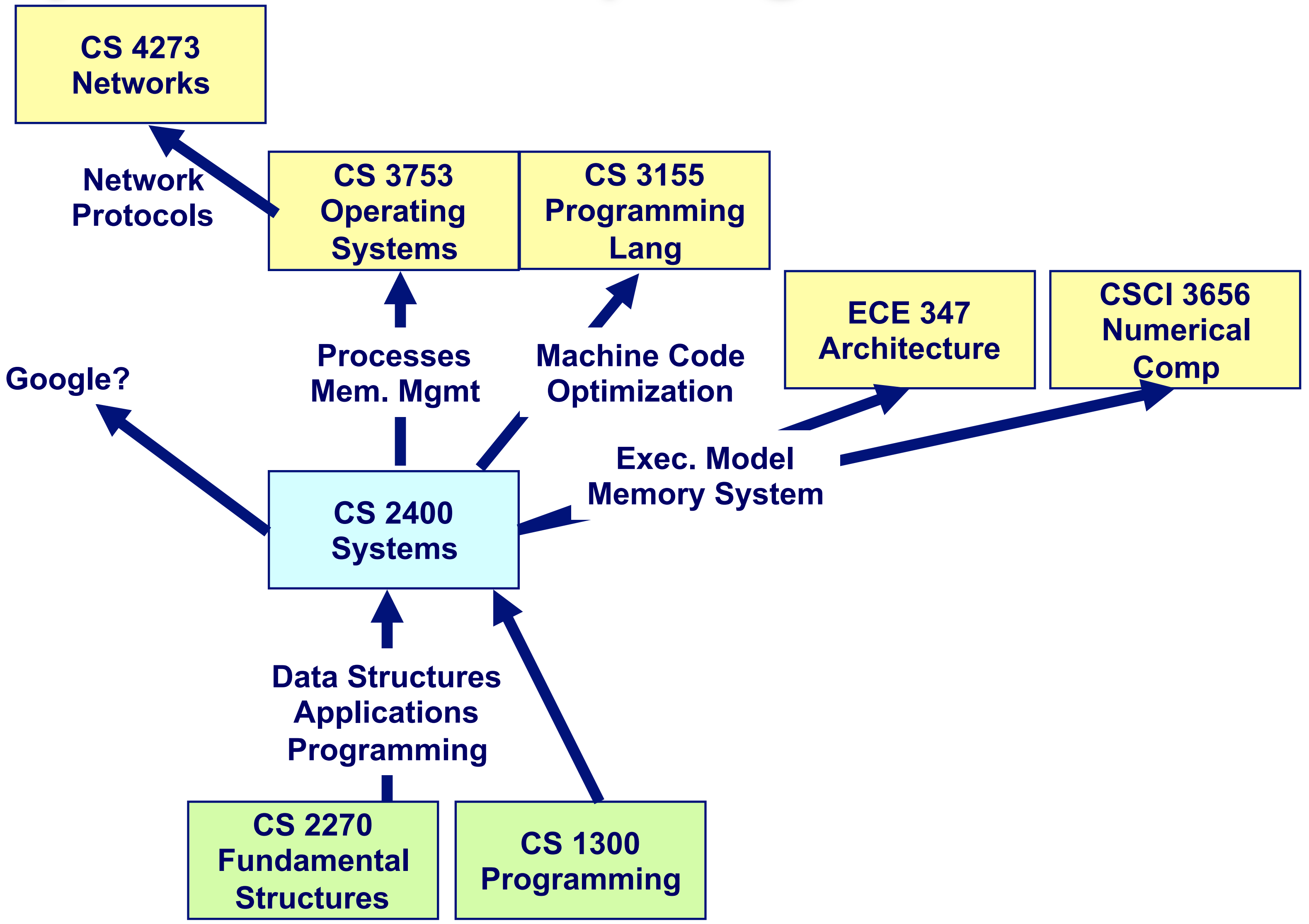
Announcements

- First data lab to be released on moodle, due in ~3 weeks
- First recitation with TAs this week
  - Install the 2400 class VM (VirtualBox-based) on your laptops
  - Introduce data lab
- Read Chapter 1 and 2.1-2.3 (skip floating point)

Hints!

1. Do not fall behind on the reading in this class!
2. Do all practice problems in the textbook! (solutions are there)
3. Attend lectures!
4. Skip roadblocks and circle back

Systems as a Springboard…

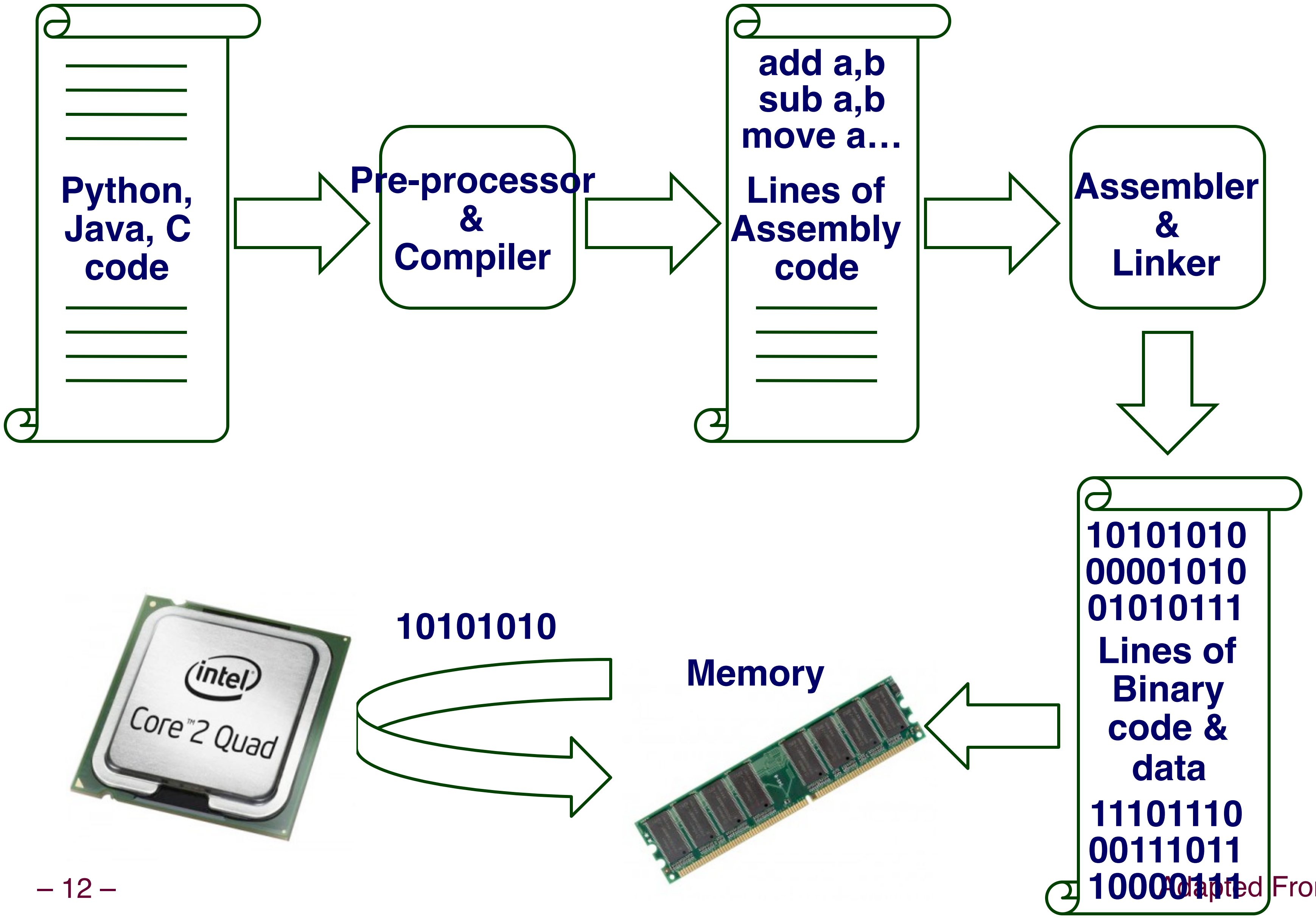


Adapted From CMU

Why C?

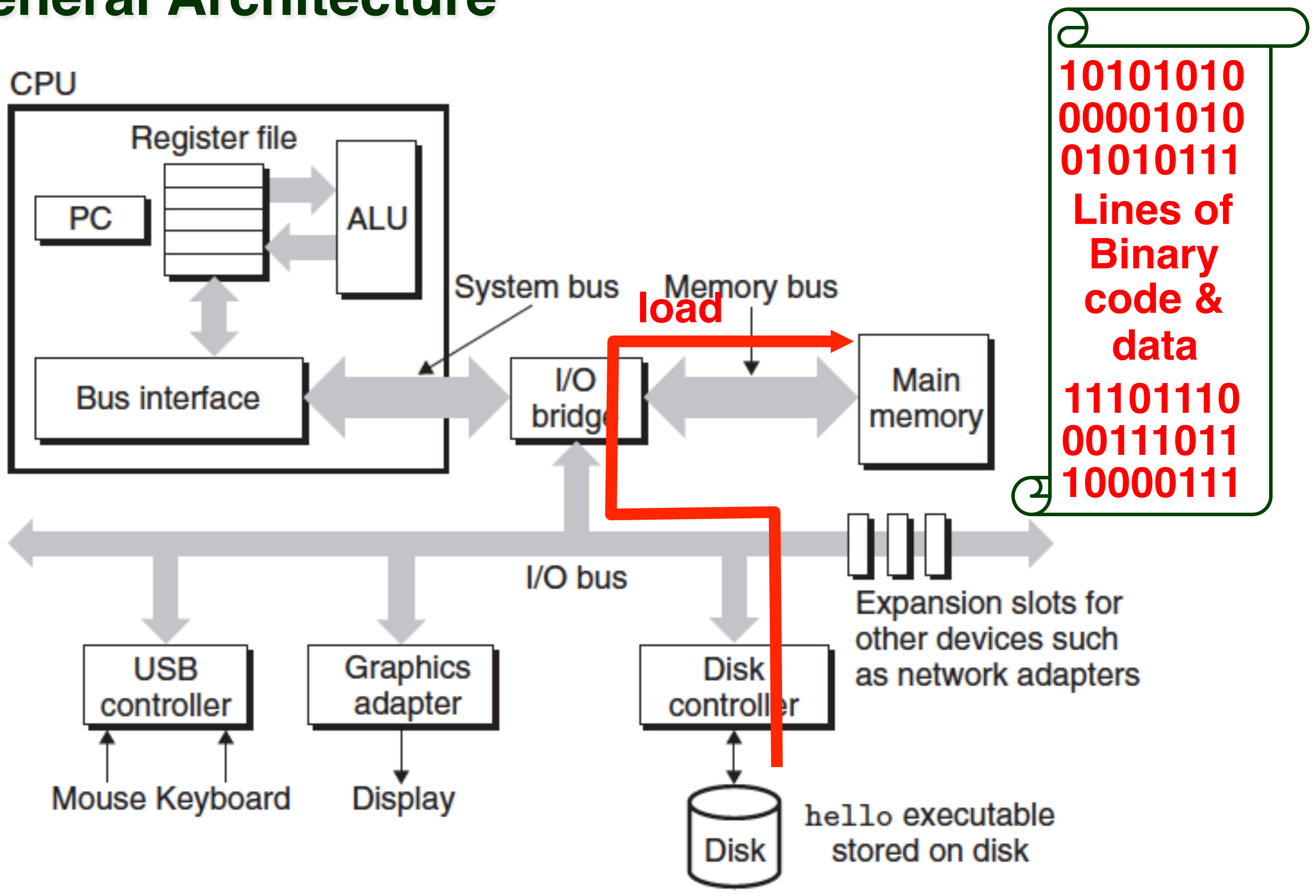
- A good compromise between a high-level programming language and low-level hardware
  - Maps well to assembly and binary machine code instructions
  - Low level enough to manipulate memory with pointers
    - Learn from C about the dangers of pointer arithmetic, array out-of-bounds memory accesses, etc. – we'll study some of these
- Most operating systems and network protocol stacks are built with C, as are many high-performance applications

2400 In a Nutshell



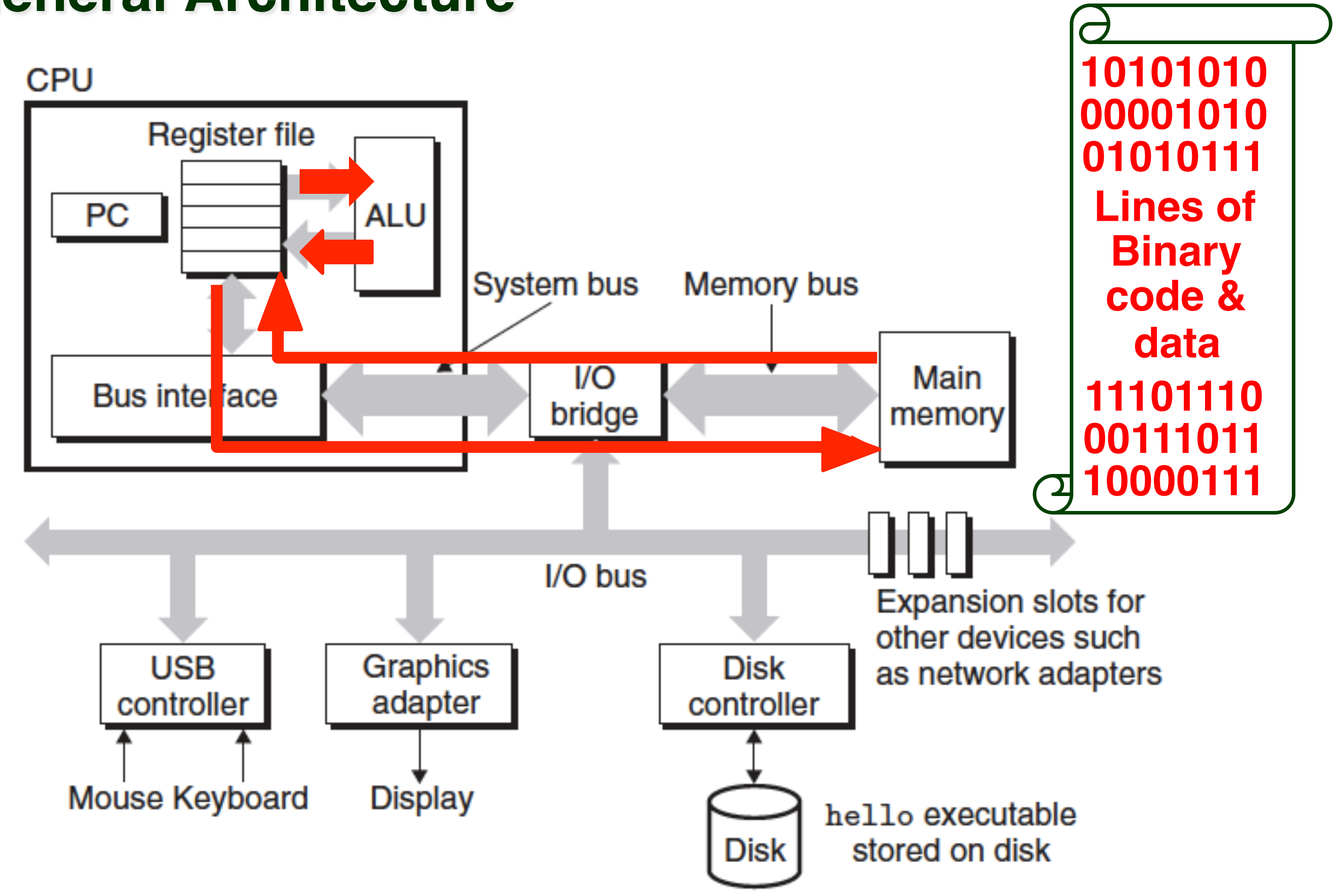
Hardware Organization of a Computer System (Von Neumann)

- **General Architecture**



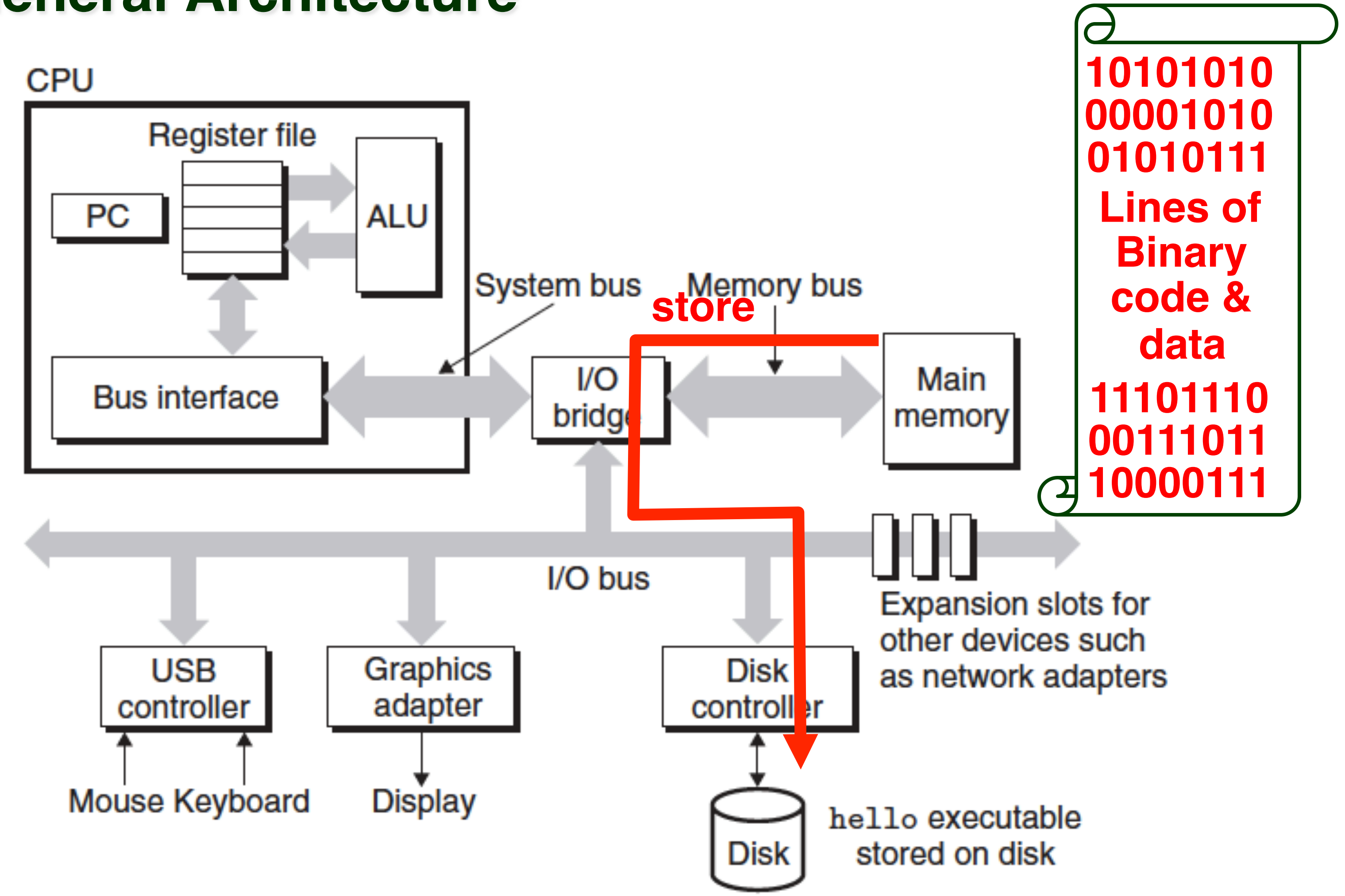
Hardware Organization of a Computer System

- **General Architecture**



Hardware Organization of a Computer System

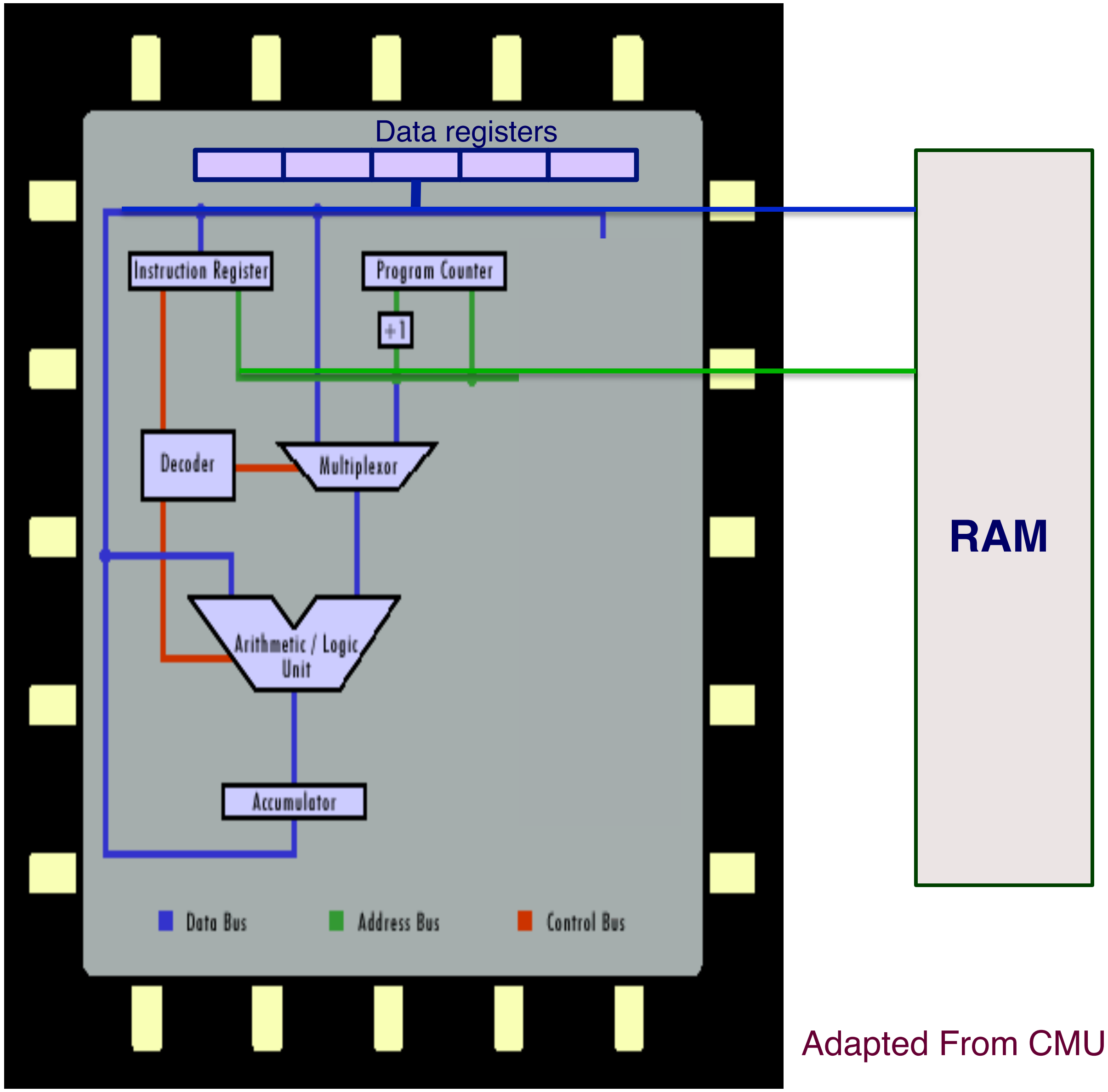
- General Architecture



Adapted From CMU

Inside the CPU - Building Blocks

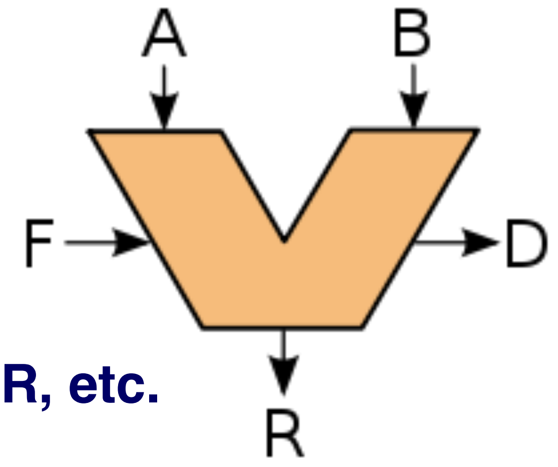
- Registers
  - Data (16 in IA64)
  - Instruction register (IR) = current instruction
  - Program counter (PC) = pointer to next instruction in memory
- Control Unit (not shown)
  - Communicate with RAM



Adapted From CMU

Arithmetic Logic Unit (ALU) of CPU

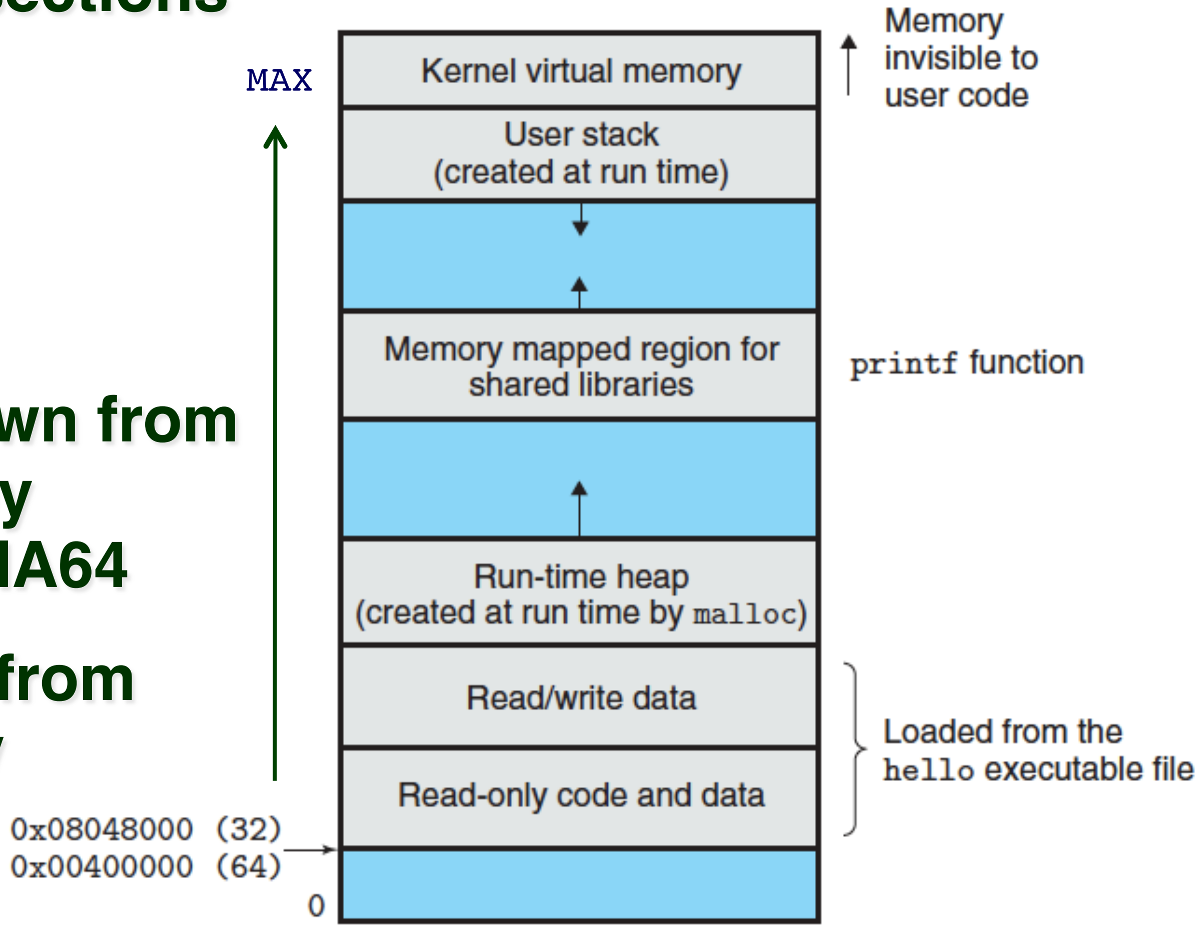
- Performs two types of operations
  - Arithmetic: ADD, SUBTRACT, etc.
  - Logic: AND, OR, NOT, XOR, >, <
- In the picture,
  - A and B are the input data or operands
  - F = function to perform, i.e. +, -, AND, XOR, etc.
  - R = result
  - D = any flags due to operation, such as a carry, overflow, sign, etc.



- A, B, and R are all stored in registers
  - As we will see, one type of instruction moves instructions and data to/from CPU from/to memory
    - i.e. fetch A and B from memory, move R to memory
  - Another type of instruction uses ALU to operate on this data

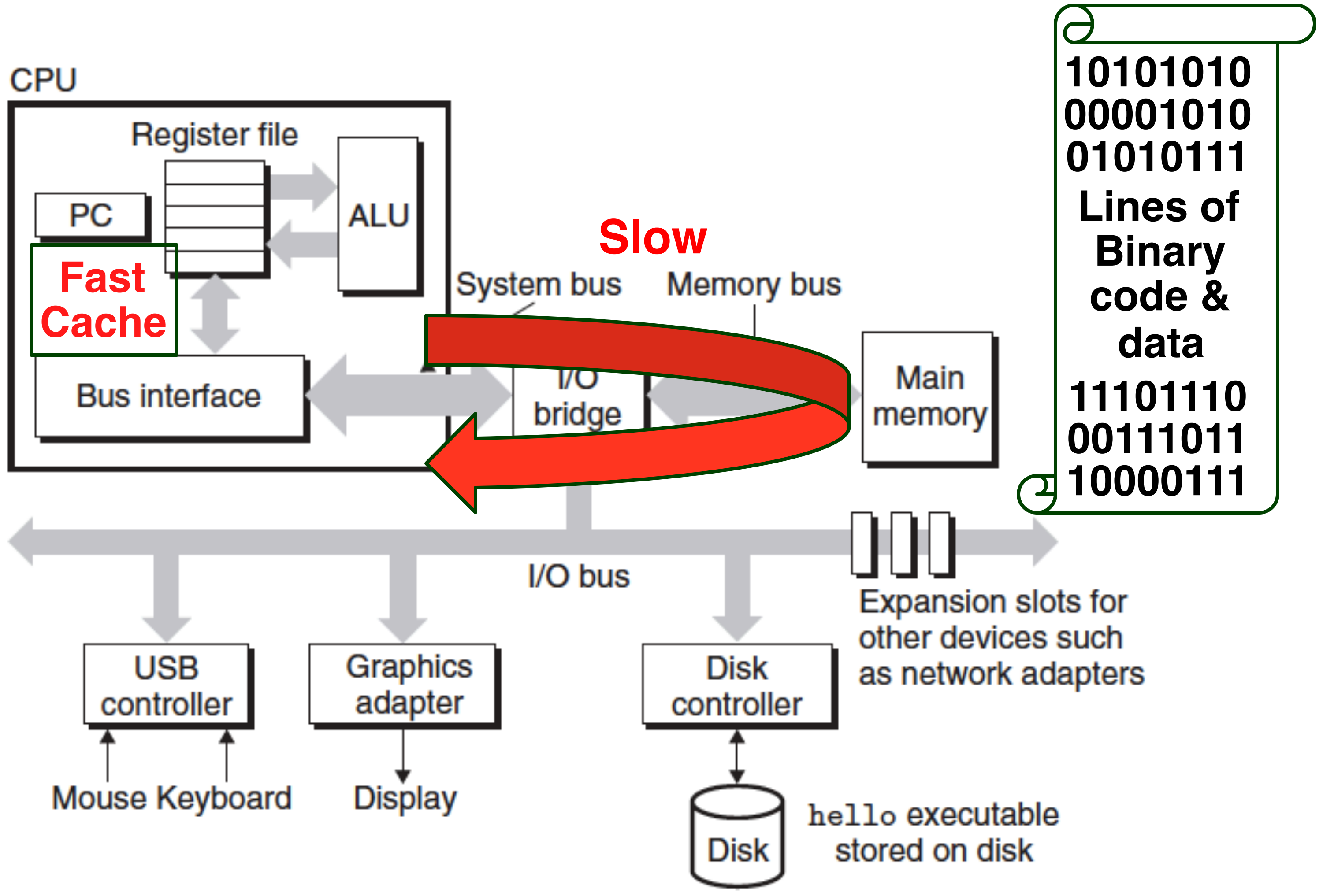
Layout of a Program in Memory

- Program memory is divided into 4 sections
  1. Code
  2. Data
  3. Heap
  4. Stack
- Stack grows down from high memory by convention on IA64
- Heap grows up from low memory by convention


MAX
Memory invisible to user code
Kernel virtual memory
User stack (created at run time)
Memory mapped region for shared libraries
printf function
Run-time heap (created at run time by malloc)
Read/write data
Read-only code and data
Loaded from the hello executable file
0x08048000 (32)
0x00400000 (64)
0

Adapted From CMU

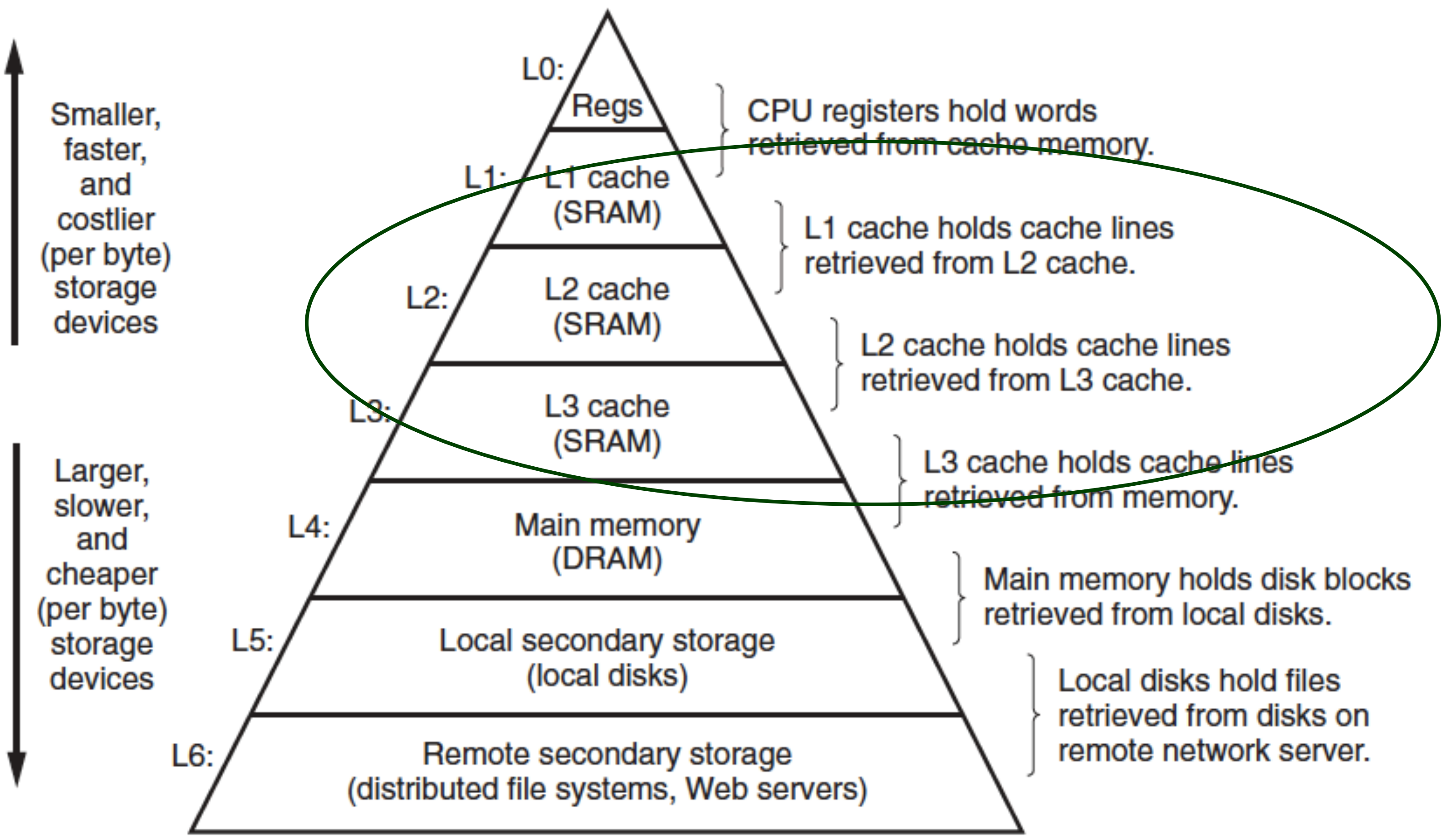
Cache For Faster Data/Code Access



Cache For Faster Data/Code Access

- **Caching**
  - Going to memory is quite slow compared to the speed of the CPU
    - RAM access time is ~ microseconds.
    - CPU executes multiple instructions per nanosecond.
    - So CPU waits ~1000 cycles to fetch data from memory!
  - Solution: create a smaller but faster buffer called a cache, and cache data there that is going to be used soon
    - Soon => program must exhibit locality of code/data access
    - In reality, cache data that was more recently used
  - Why not cache all data in registers?
    - Too expensive/byte

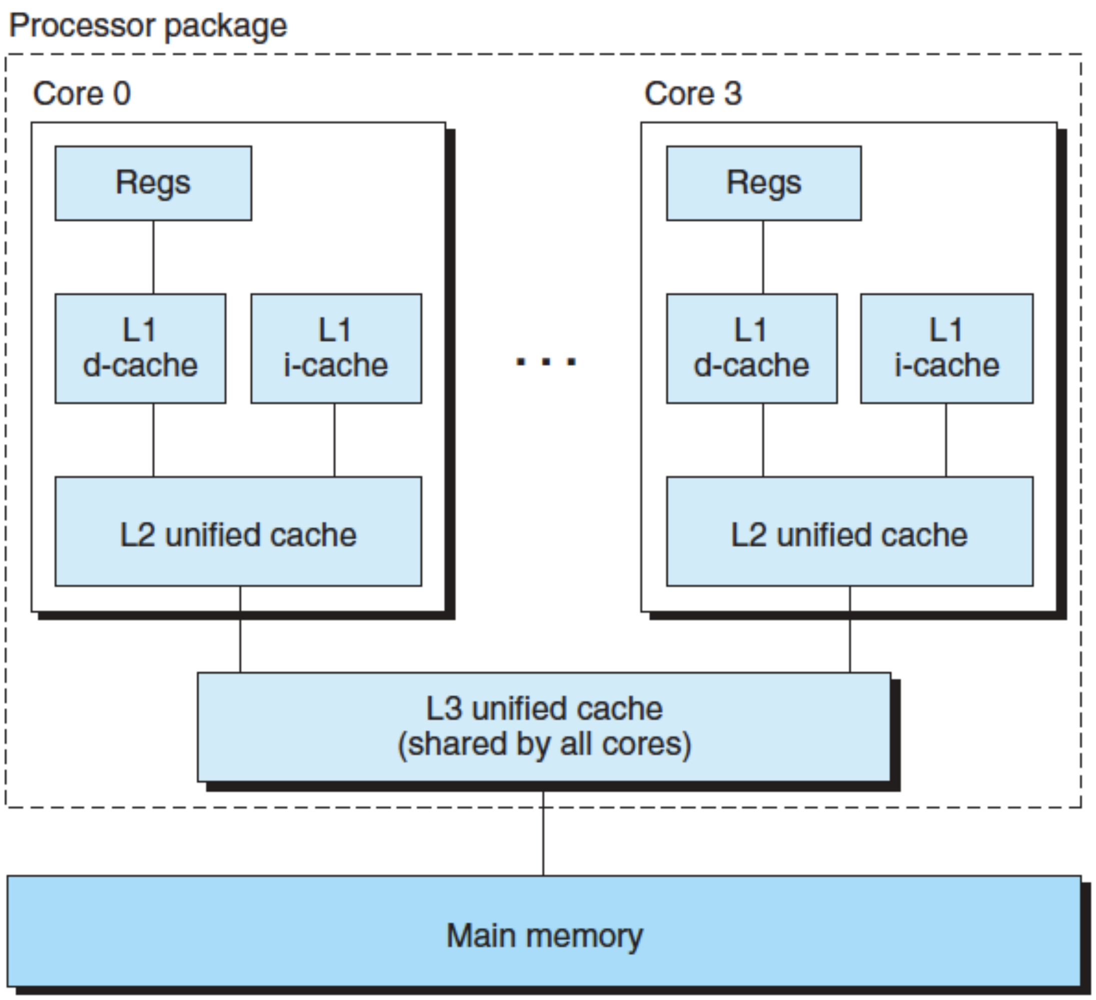
Memory Hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network server.

Adapted From CMU

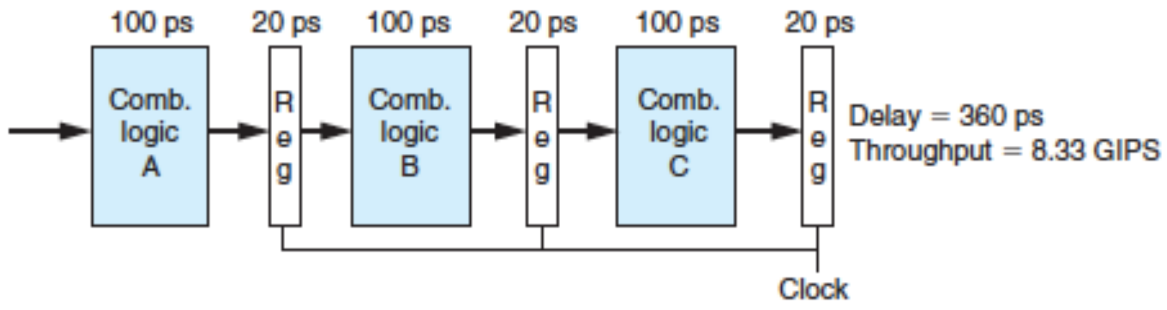
Memory Hierarchy

- **Intel Core i7 organization**
  - **4 cores**
  - **Each core has an L1 data cache, an L1 instruction cache, and a larger but slower L2 unified cache**
  - **Across cores, there is an L3 unified cache**

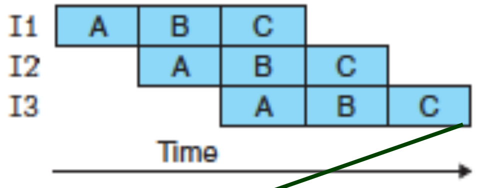


Pipelining to Improve Throughput

- **Pipelining**
  - **Instructions can be broken up into stages.**
  - **Design CPU with multiple stages or “pipeline”.**
  - **As a stage finishes, it accepts the result from the previous stage -> Faster!**
  - **Sequential non-pipelined is slower, some stages empty.**

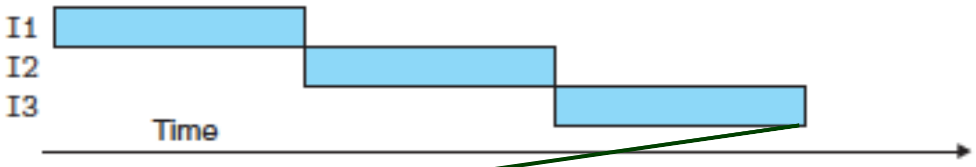


(a) Hardware: Three-stage pipeline



(b) Pipeline diagram

(a) Hardware: Unpipelined



(b) Pipeline diagram

Adapted From CMU