CSCI 2400

Introduction to Computer Systems

Fall 2014

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

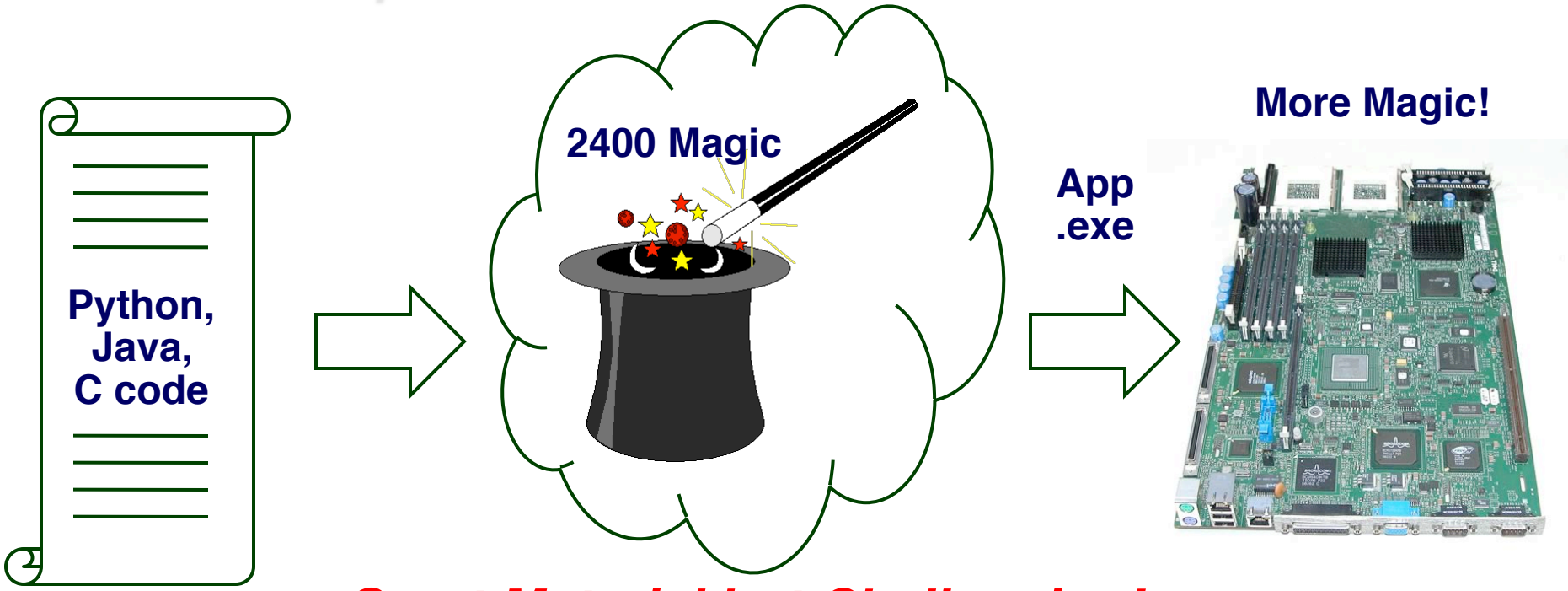
http://www.cs.colorado.edu/~rhan
rhan@cs.colorado.edu

Goal of this course

- **Understand how software executes on computer hardware**

or…

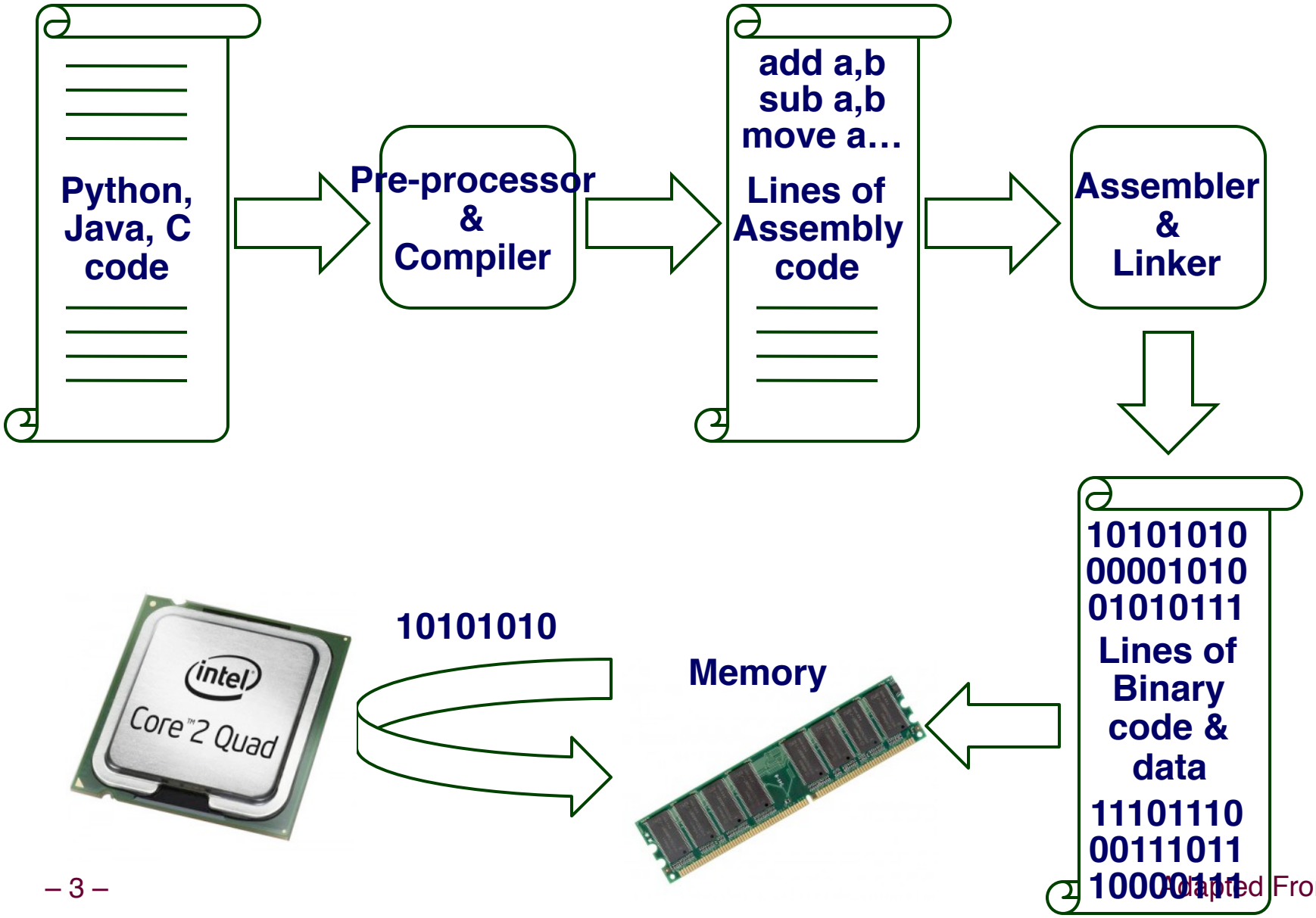
How a computer works!



Great Material but Challenging!

Adapted From CMU

2400 In a Nutshell



Chapter Mapping

Chapter 3

Lines of Source code

Pre-processor & Compiler

add a,b
sub a,b
move a…

Lines of Assembly code

Chapter 7

Assembler & Linker

Chapter 5

Chapter 8

Operating System

Chapter 2

10101010
00001010
01010111

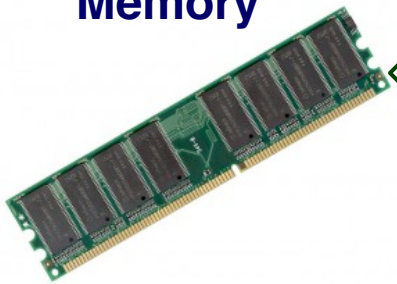
Lines of Binary code & data

11101110
00111011
10000111

10101010

Chapter 9

Memory



Chapters 3, 4 and 6

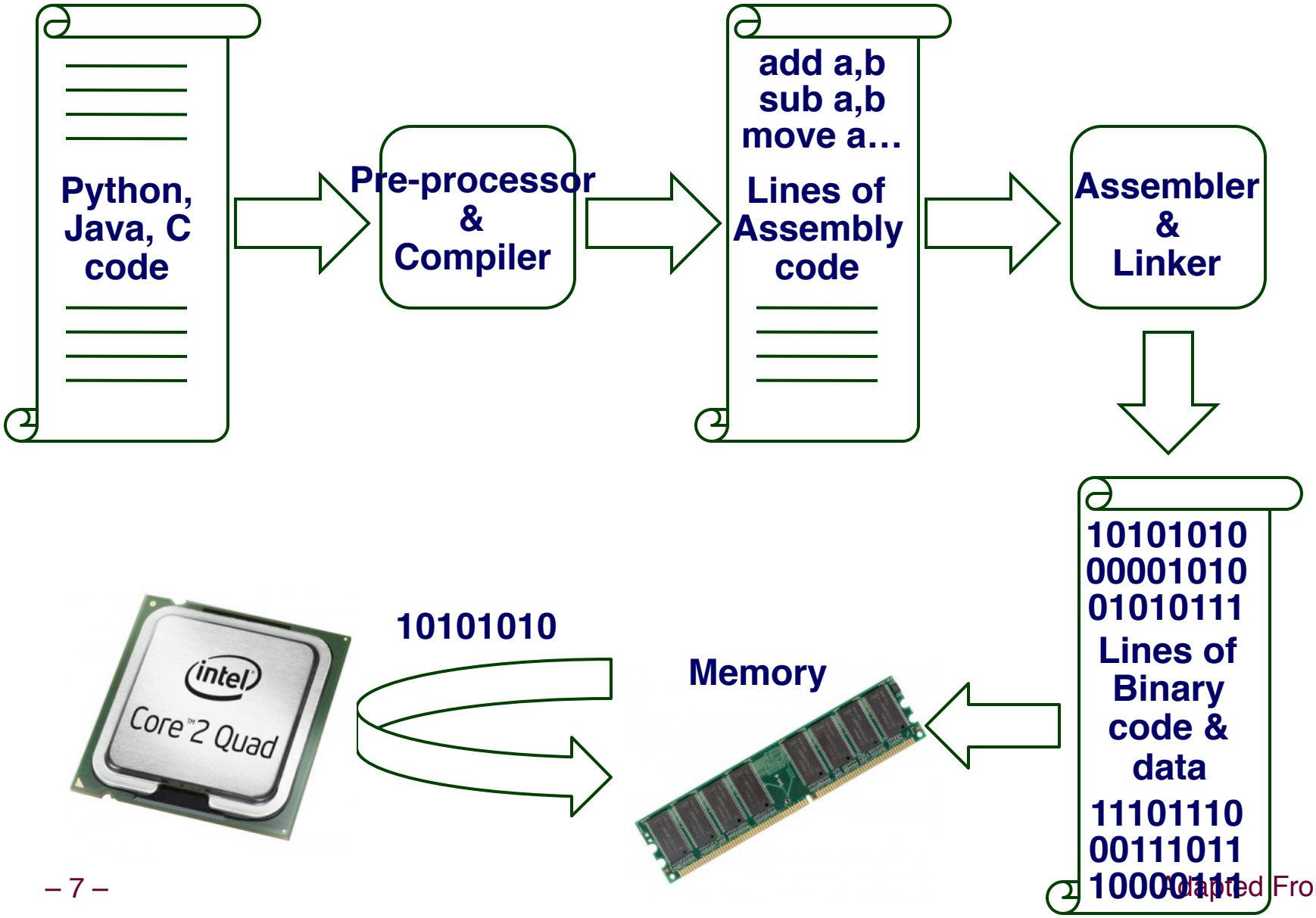
Approximate Timeline

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	
5	Ch 3: Assembly: stacks, arrays	
6	Ch 3: Assembly: structs, 64-bit	Bomb Lab
7	Ch 2: Floating point	MT 1
8	Ch 4: ISA, Pipelining	
9	Ch 4-5: Pipelining, Optimization	Buffer Lab
10	Ch 5: Performance Optimization	
11	Ch 6: Caching	Performance Lab
12	Ch 6: Caching, Storage	MT 2
13	Ch 8: Exceptions, Signals	
14	Ch 9: Virtual Memory	Shell Lab
15	Ch 7: Linking	
Finals	Final Exam Thursday Dec 18, 4:30-7 pm	Final Exam

Announcements

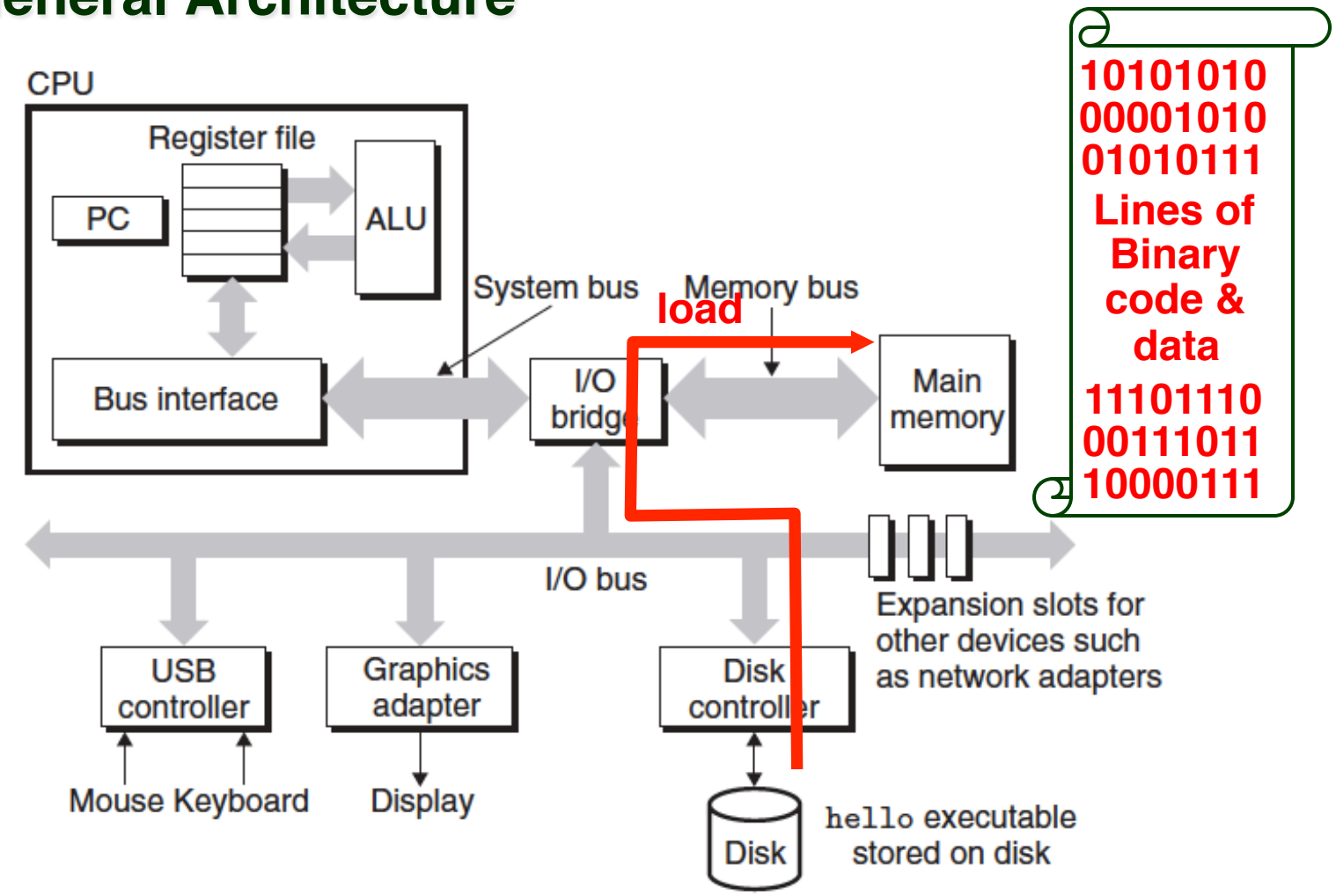
- Let's go to moodle.cs.colorado.edu, the primary rendezvous point for CS 2400
  - Sign up for an account, using 'systems14' as the enrollment key
  - Walk through the syllabus
- First data lab already released on moodle, due in ~3 weeks
- First recitation with TAs was on Monday
  - Install the 2400 class VM (VirtualBox-based) on your laptops
  - More install sessions later this week – see link on moodle
- Sign up for CSEL accounts @ csel.cs.colorado.edu
- Read Chapter 1 and 2.1-2.3 (skip floating point)

2400 In a Nutshell



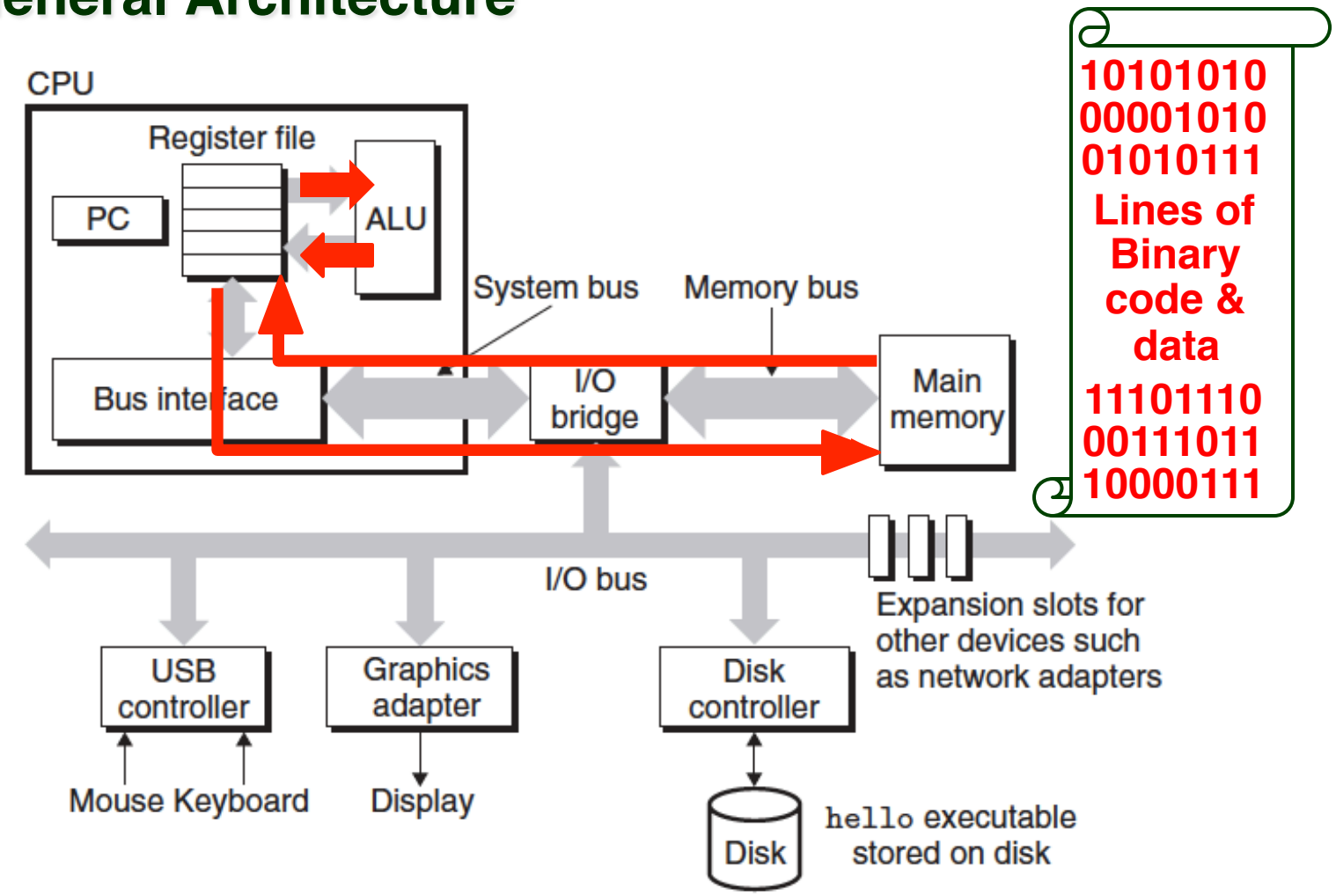
Hardware Organization of a Computer System (Von Neumann)

- **General Architecture**



Hardware Organization of a Computer System

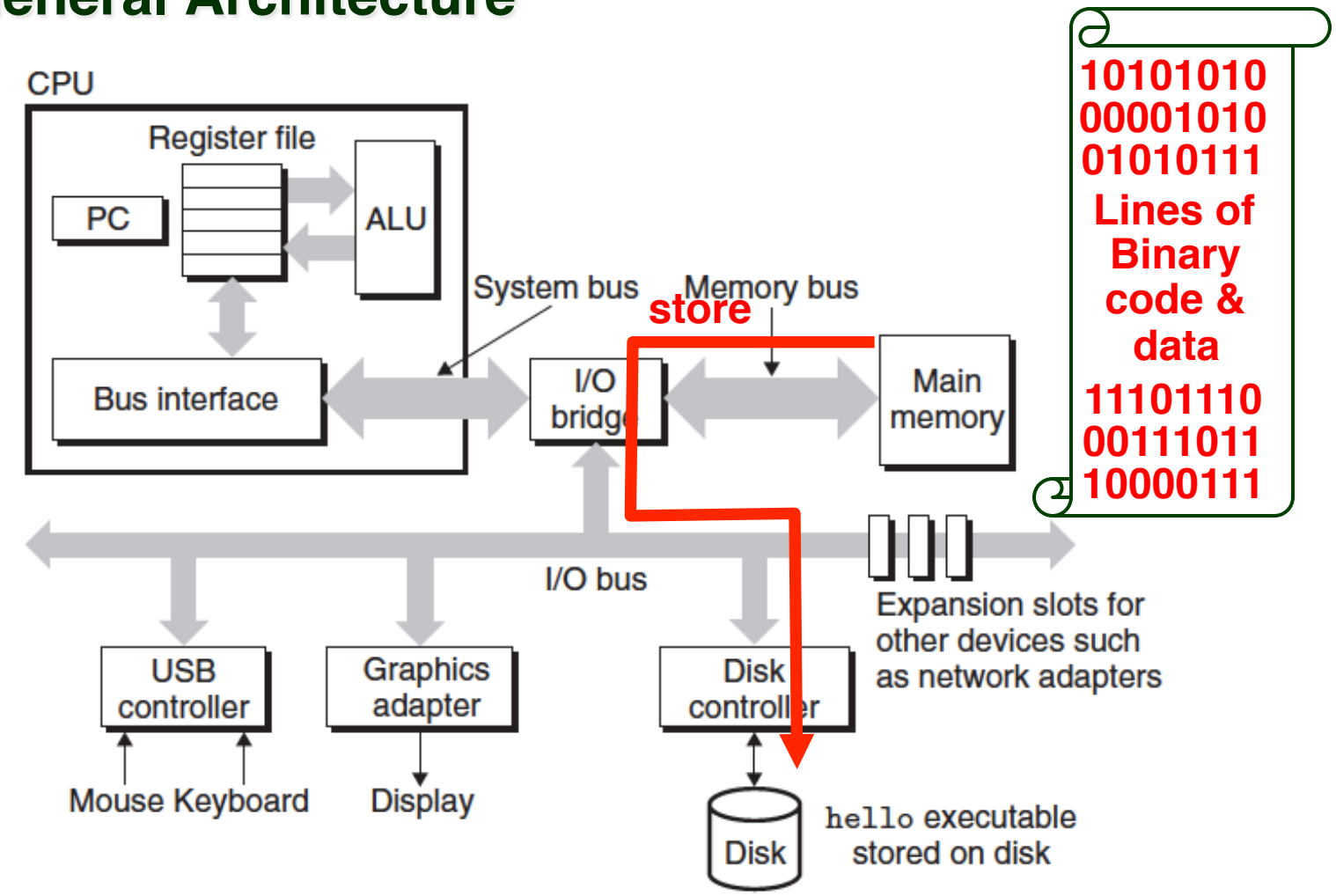
- **General Architecture**



Adapted From CMU

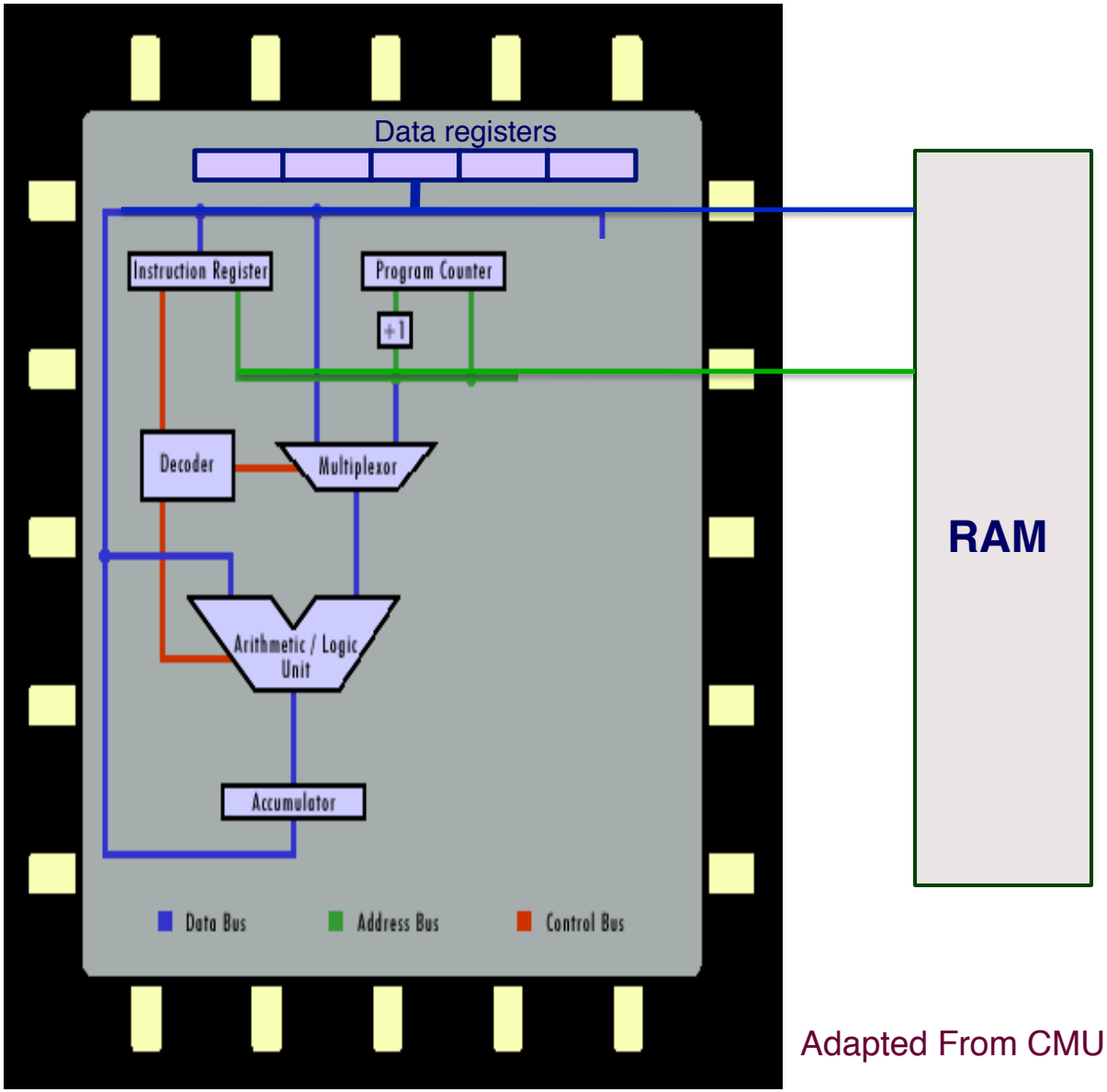
Hardware Organization of a Computer System

- **General Architecture**



Inside the CPU - Building Blocks

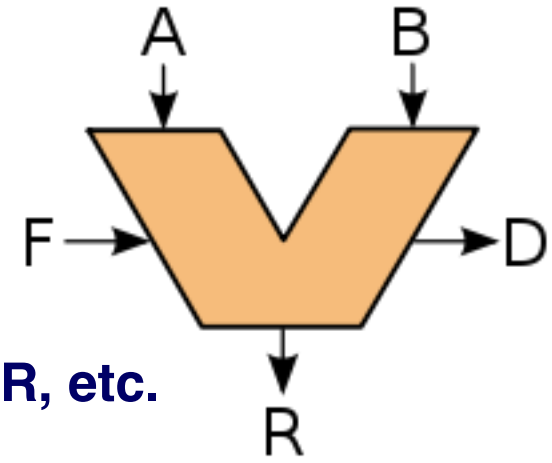
- **Registers**
  - Data (8 in IA32)
  - Instruction register (IR) = current instruction
  - Program counter (PC) = pointer to next instruction in memory
- **Control Unit (not shown)**
  - Communicate with RAM



Adapted From CMU

Arithmetic Logic Unit (ALU) of CPU

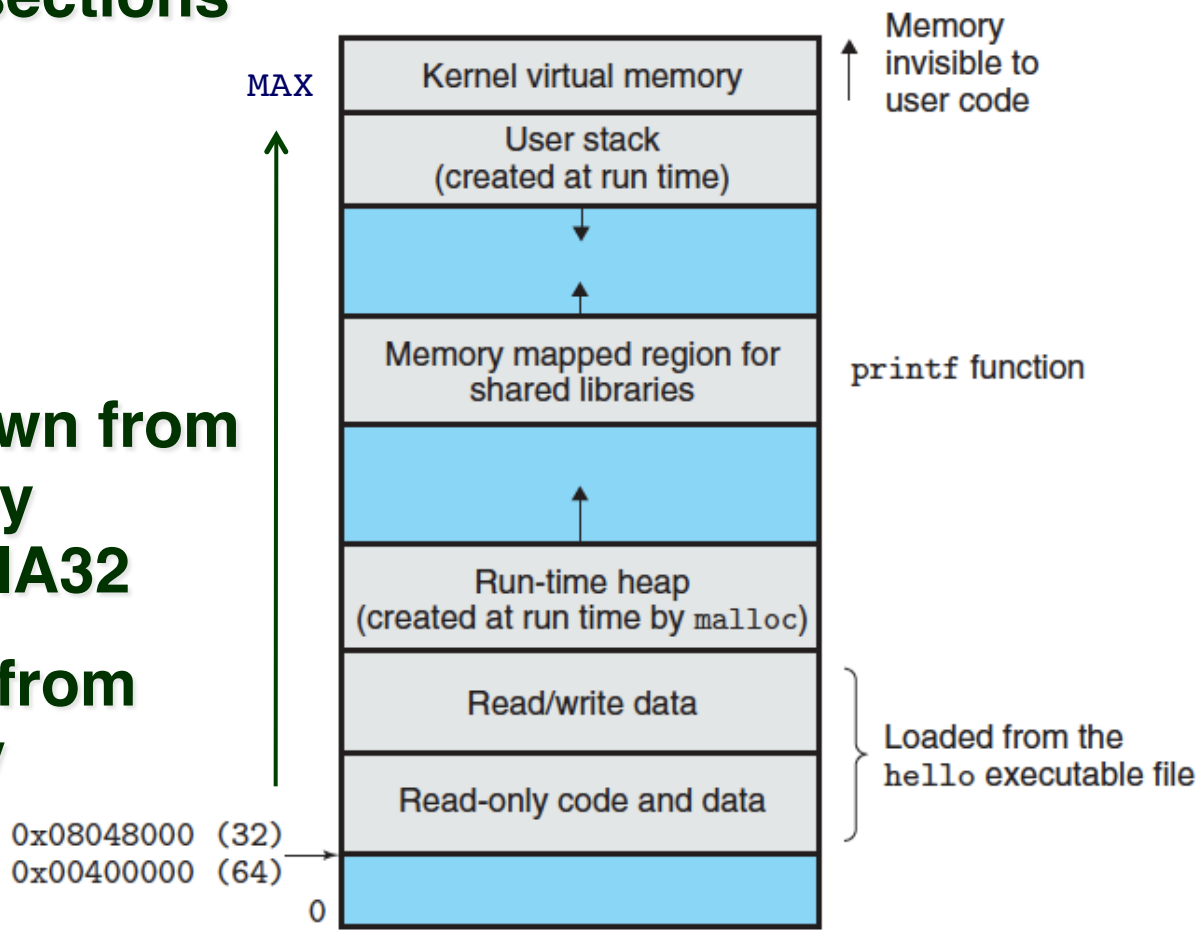
- Performs two types of operations
  - Arithmetic: ADD, SUBTRACT, etc.
  - Logic: AND, OR, NOT, XOR, >, <
- In the picture,
  - A and B are the input data or operands
  - F = function to perform, i.e. +, -, AND, XOR, etc.
  - R = result
  - D = any flags due to operation, such as a carry, overflow, sign,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

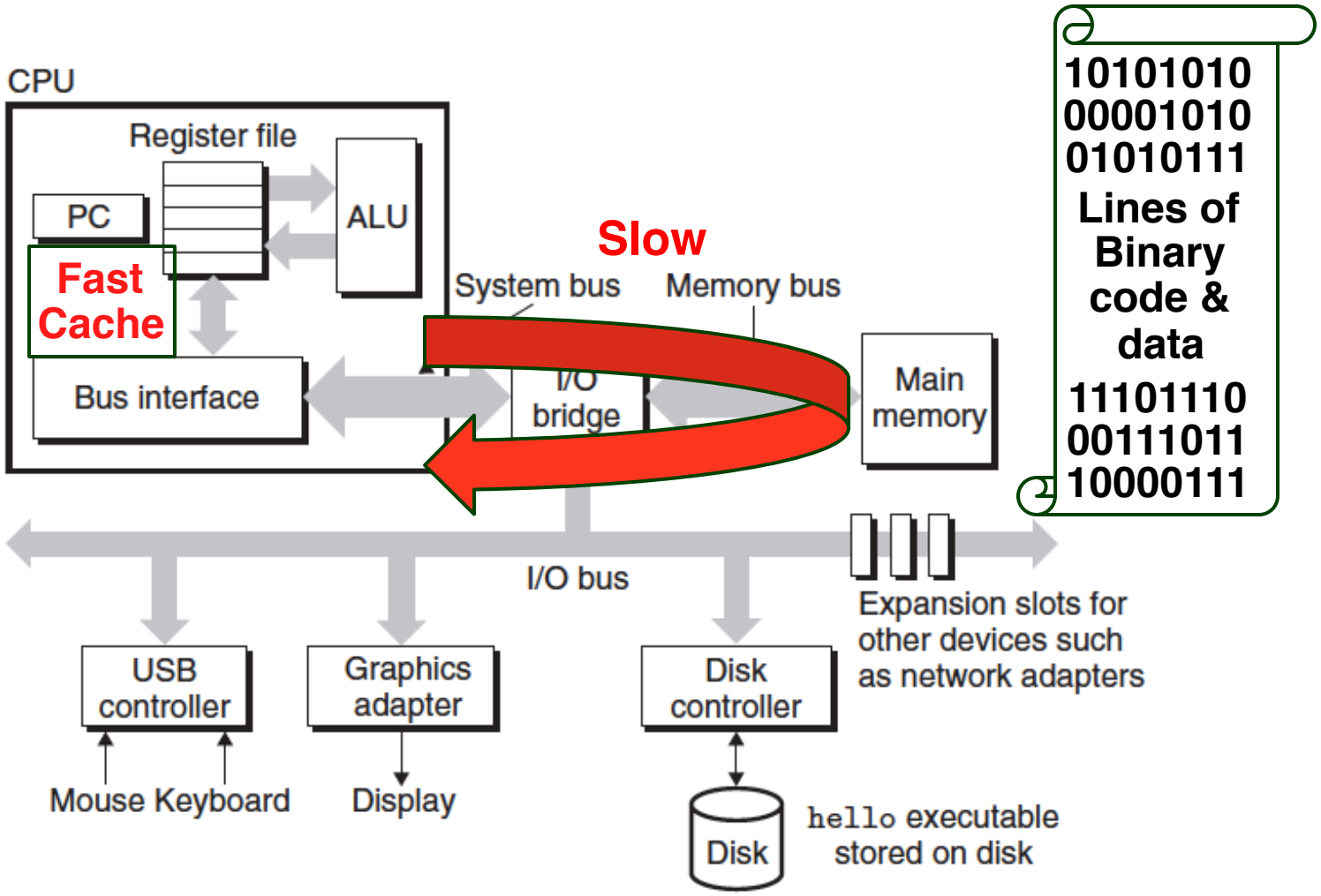
Layout of a Program in Memory

- Program memory is divided into 4 sections
  1. Code
  2. Data
  3. Heap
  4. Stack
- Stack grows down from high memory by convention on IA32
- Heap grows up from low memory by convention



Adapted From CMU

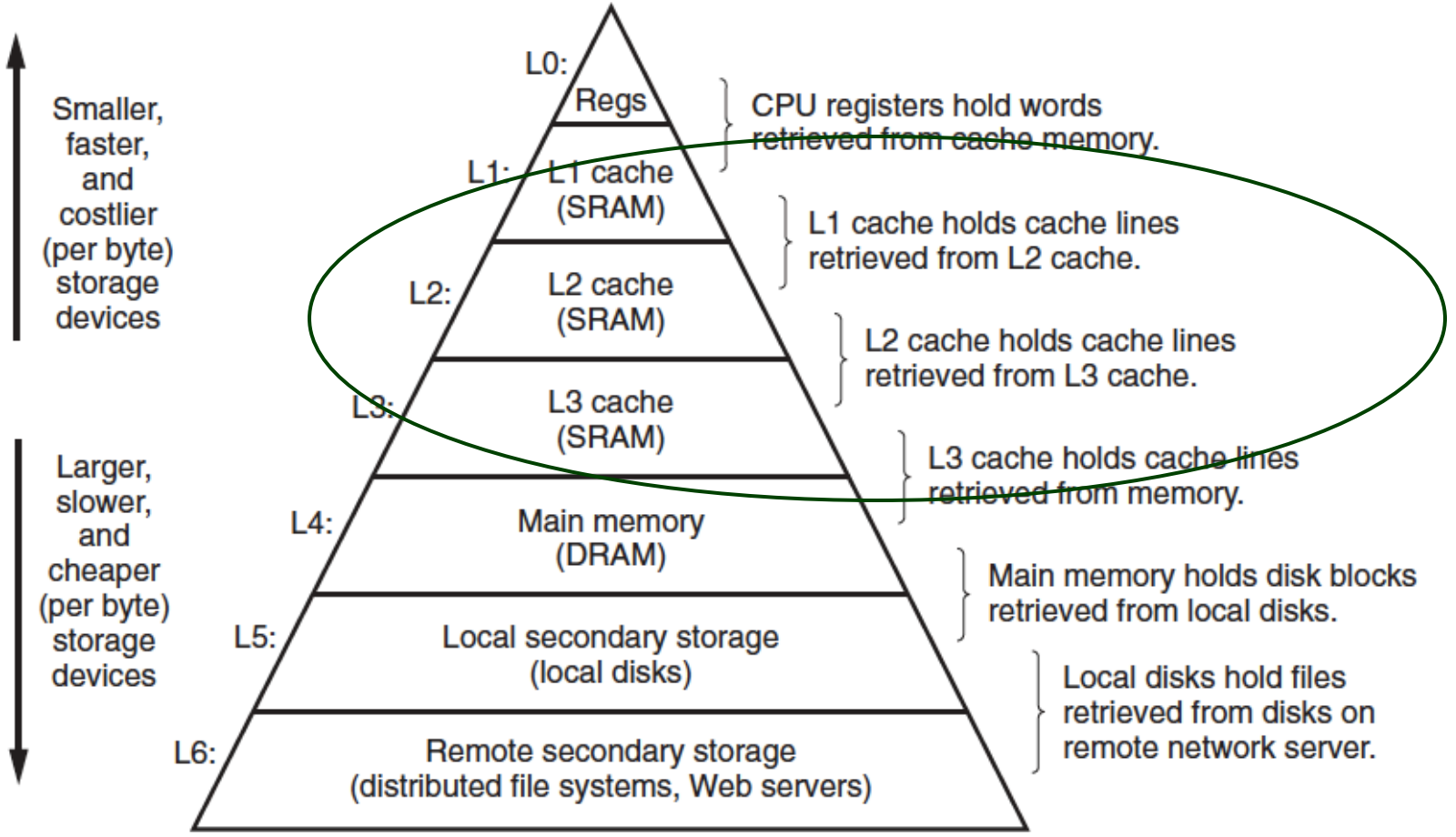
Cache For Faster Data/Code Access



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
    - In reality, cache data that was more recently used
  - Why not cache all data in registers?
    - Too expensive/byte

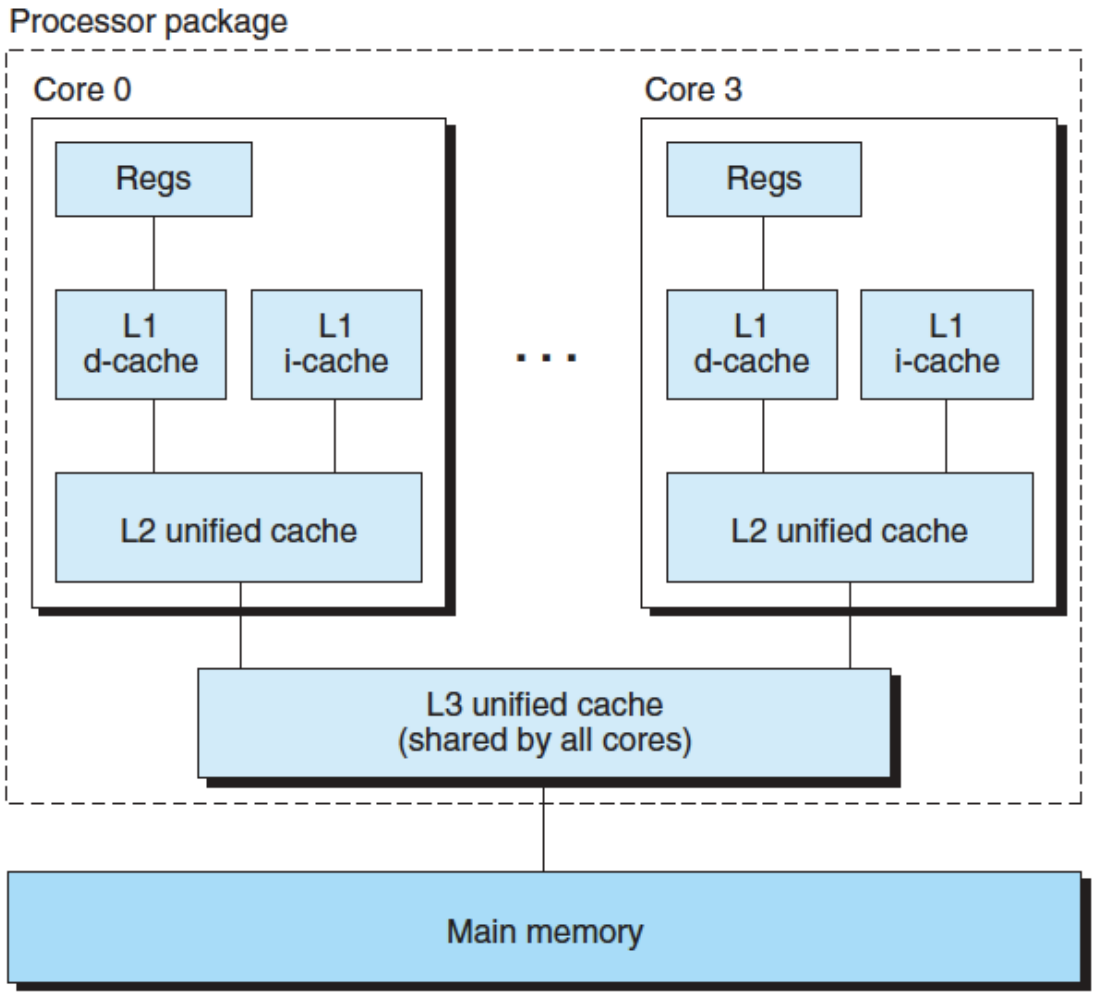
Memory Hierarchy



Adapted From CMU

Memory Hierarchy

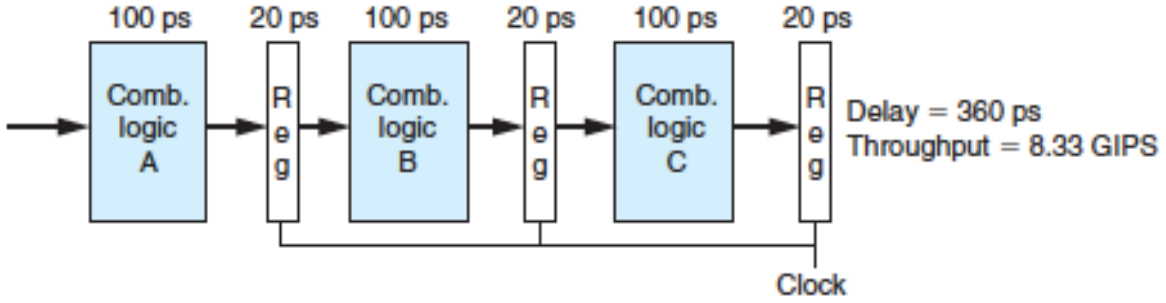
- **Intel Core i7 organization**
  - **4 cores**
  - **Each core has an L1 data cache, an L1 instruction cache, and a larger but slower L2 unified cache**
  - **Across cores, there is an L3 unified cache**



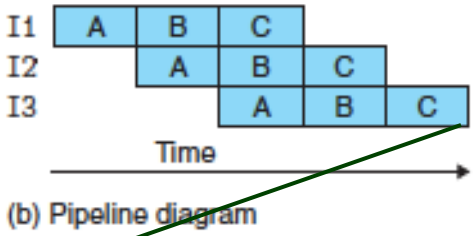
Adapted From CMU

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.

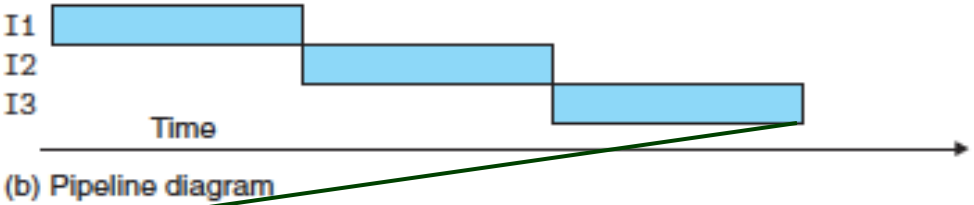


(a) Hardware: Three-stage pipeline



(b) Pipeline diagram

(a) Hardware: Unpipelined



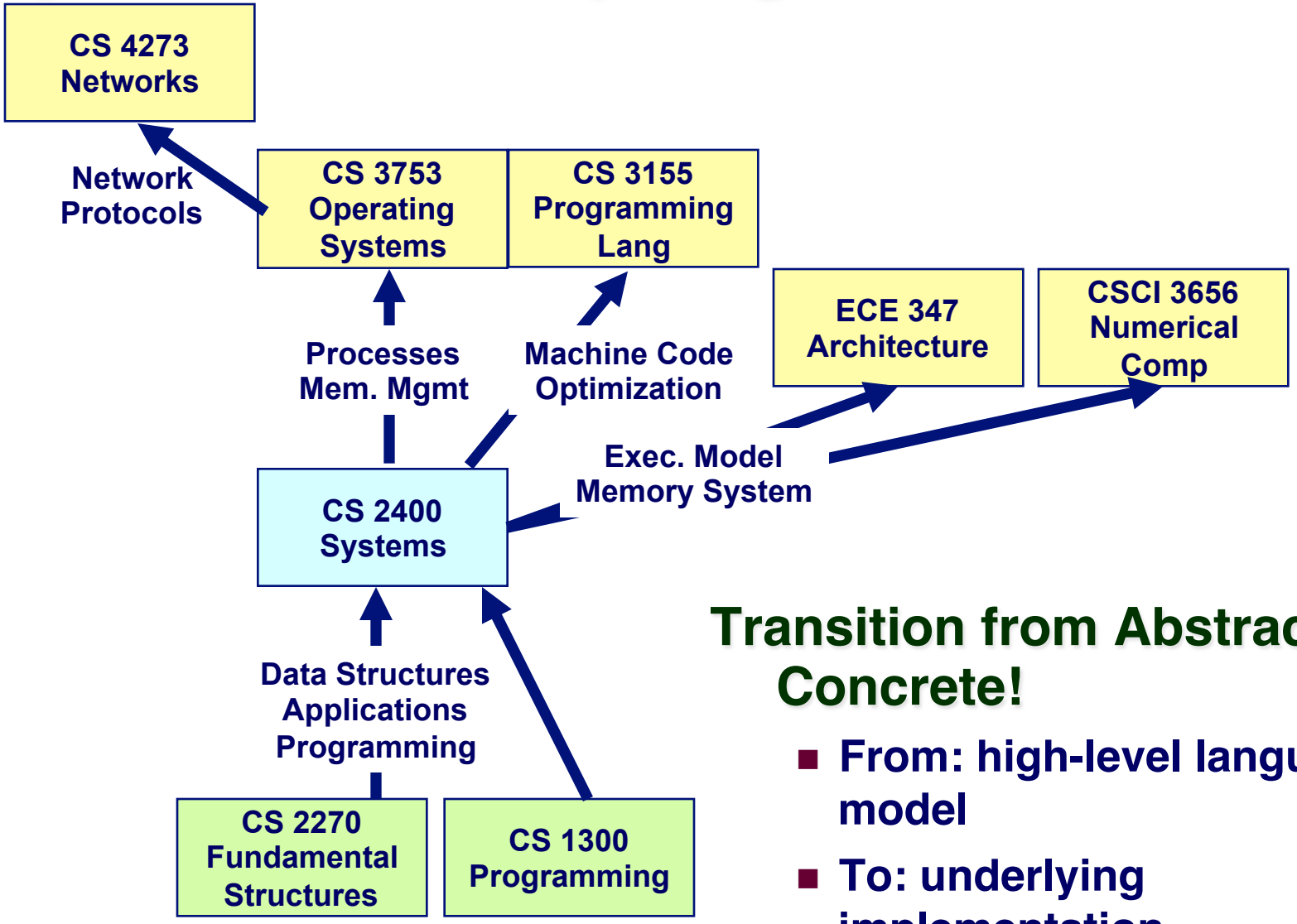
(b) Pipeline diagram

Adapted From CMU

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

Systems as a Springboard…



Transition from Abstract to Concrete!

- From: high-level language model
- To: underlying implementation

Adapted From CMU