

# **CSCI 2400**

# **Computer Systems**

**Spring 2017**

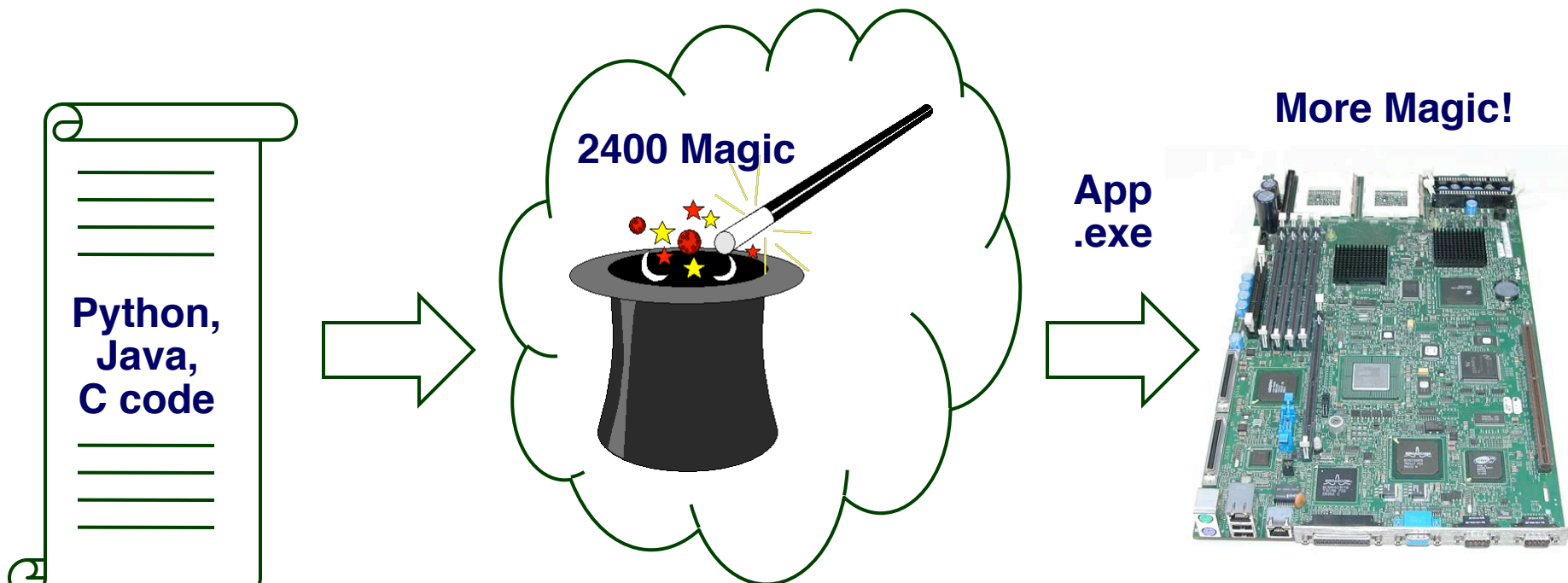
**Professor Rick Han**  
**Department of Computer Science**  
**University of Colorado at Boulder**

**[rhan@cs.colorado.edu](mailto:rhan@cs.colorado.edu)**  
**<http://www.cs.colorado.edu/~rhan>**

# Goal of 2400:

- **Learn how a computer works!**

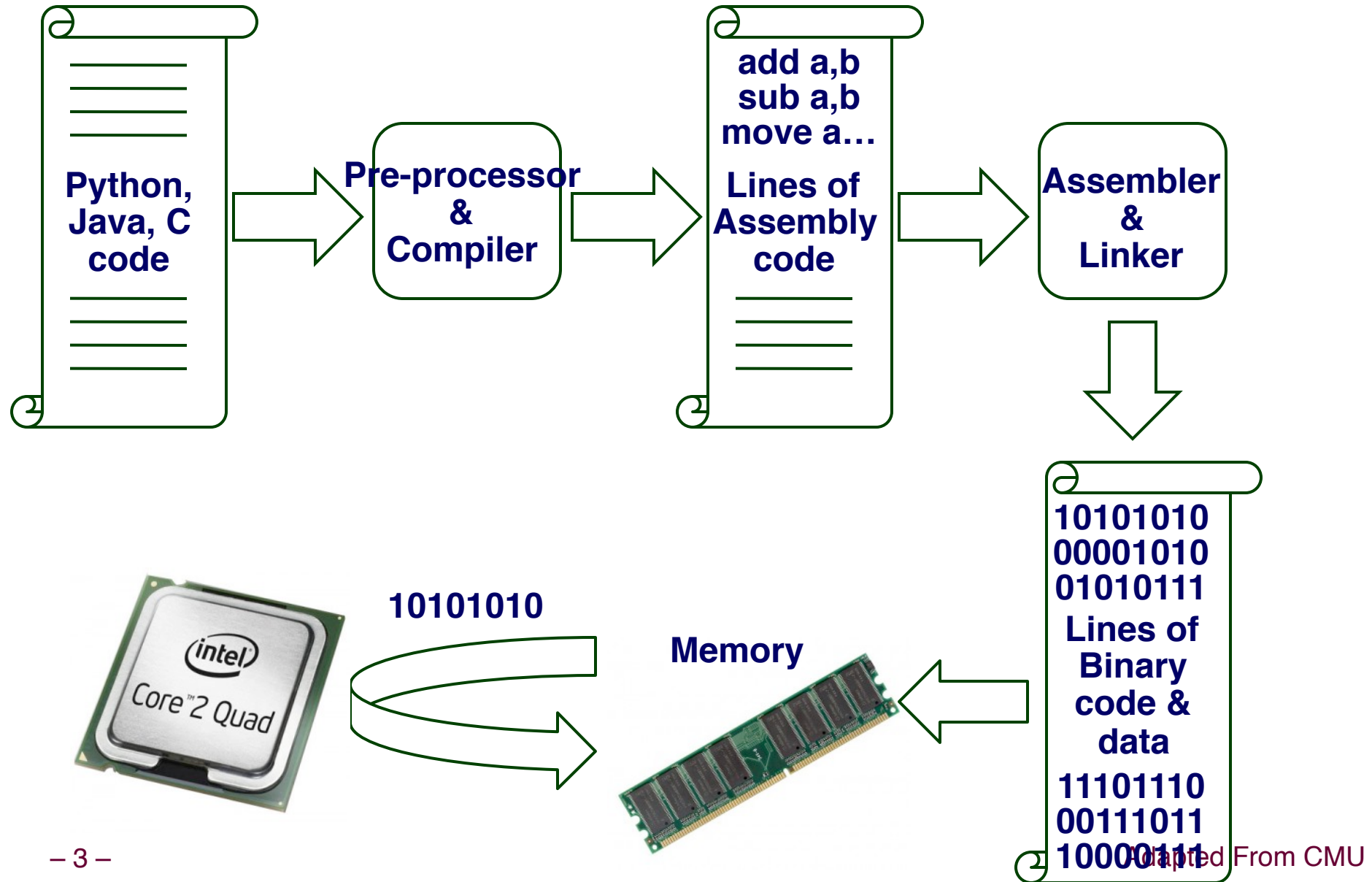
More precisely, how software executes on modern computer hardware



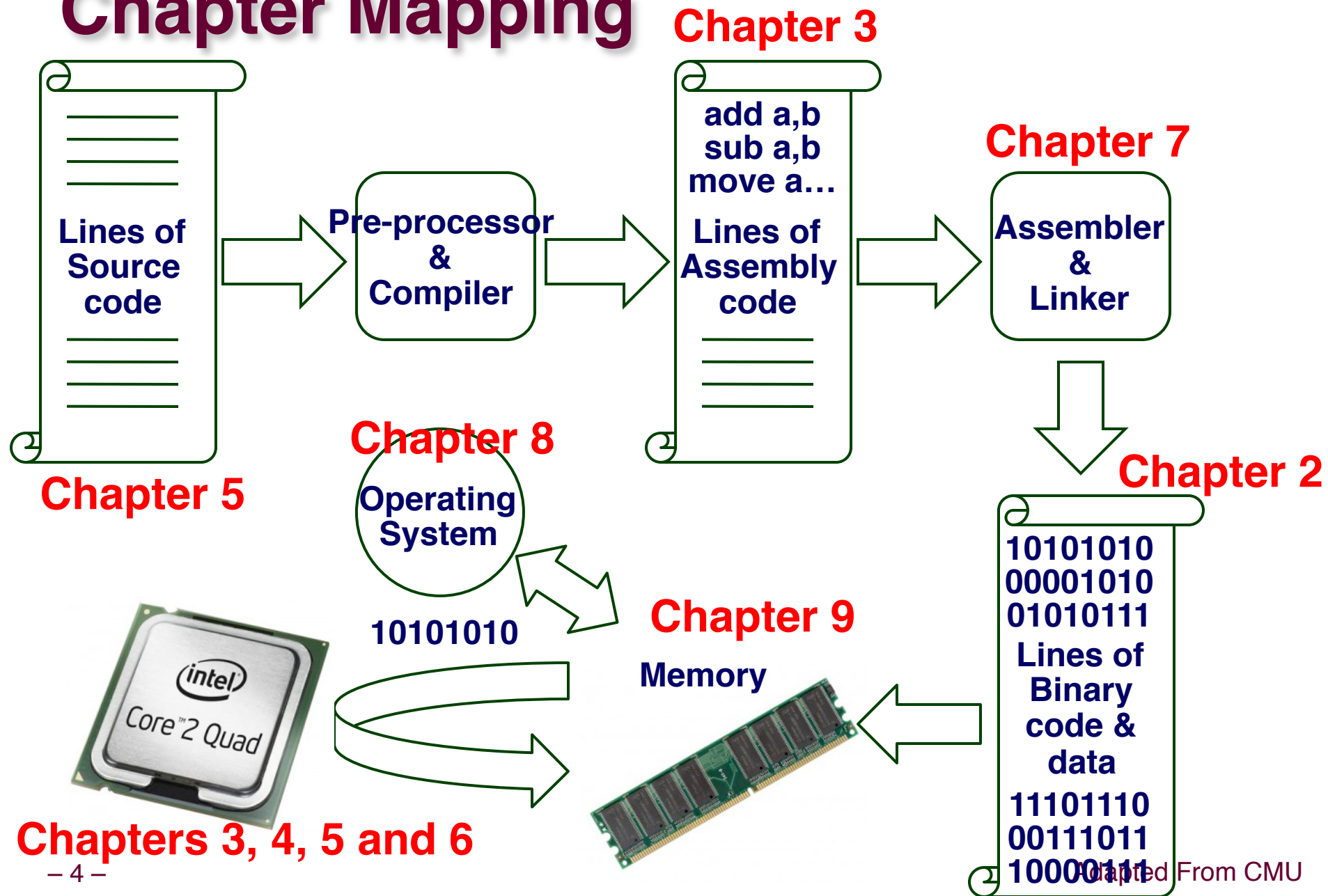
***Great Material but Challenging!***

Adapted From CMU

# 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	
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	Attack 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	Final Exam

# Announcements

- Let's go to [moodle.cs.colorado.edu](https://moodle.cs.colorado.edu), the primary rendezvous point for CS 2400
  - Sign up for an account, using '2400rocks2017' as the enrollment key
  - Walk through the syllabus

# Announcements

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

# Announcements

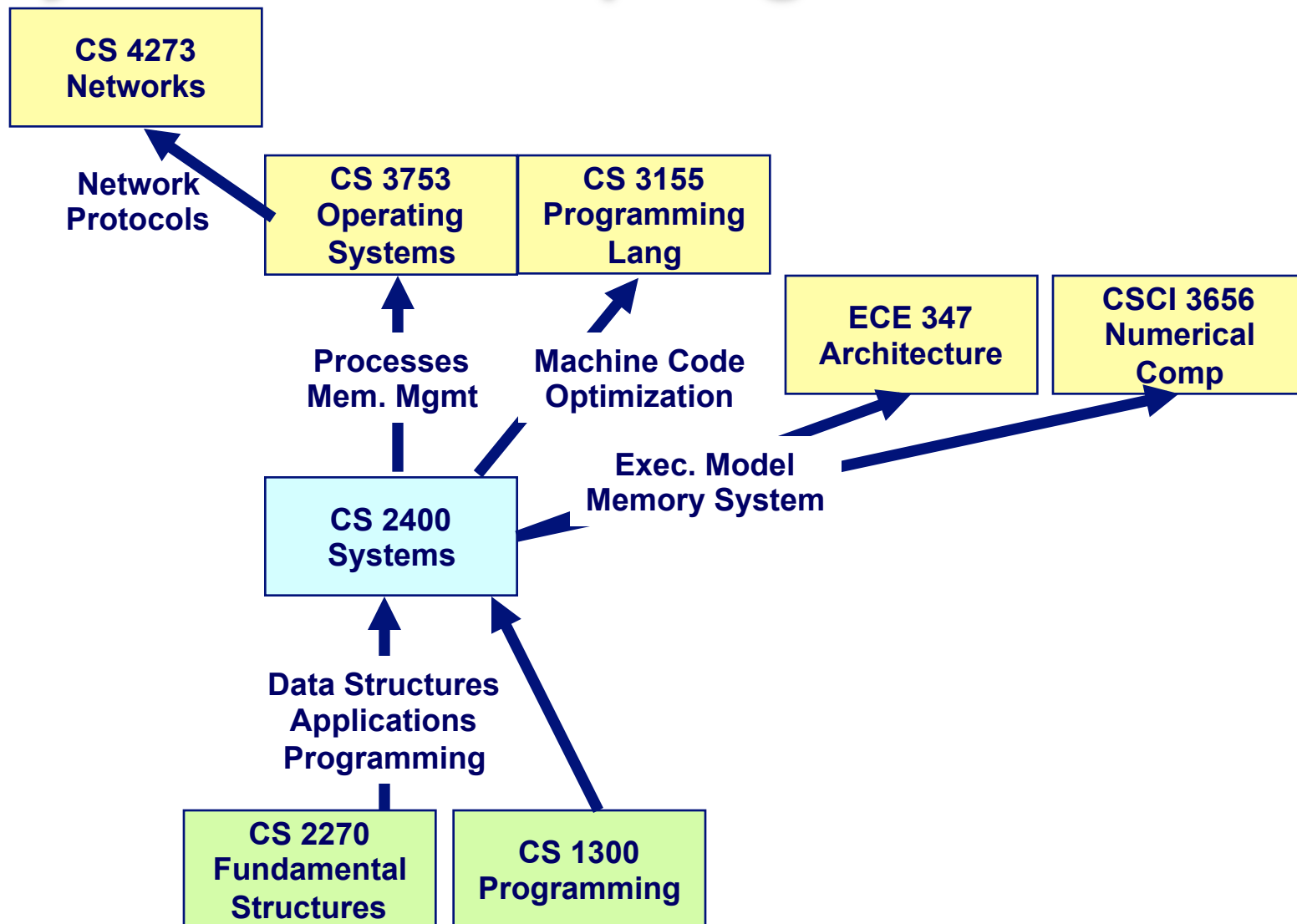
- **First data lab to be released on moodle, due in ~3 weeks**
- **First recitation with TAs this week**
  - **Install the 2400 class VM (VirtualBox-based) on your laptops**
  - **Introduce data lab**
  - **Bit manipulation operations**
- **Read Chapter 1 and 2.1-2.3 (skip floating point)**



# 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**

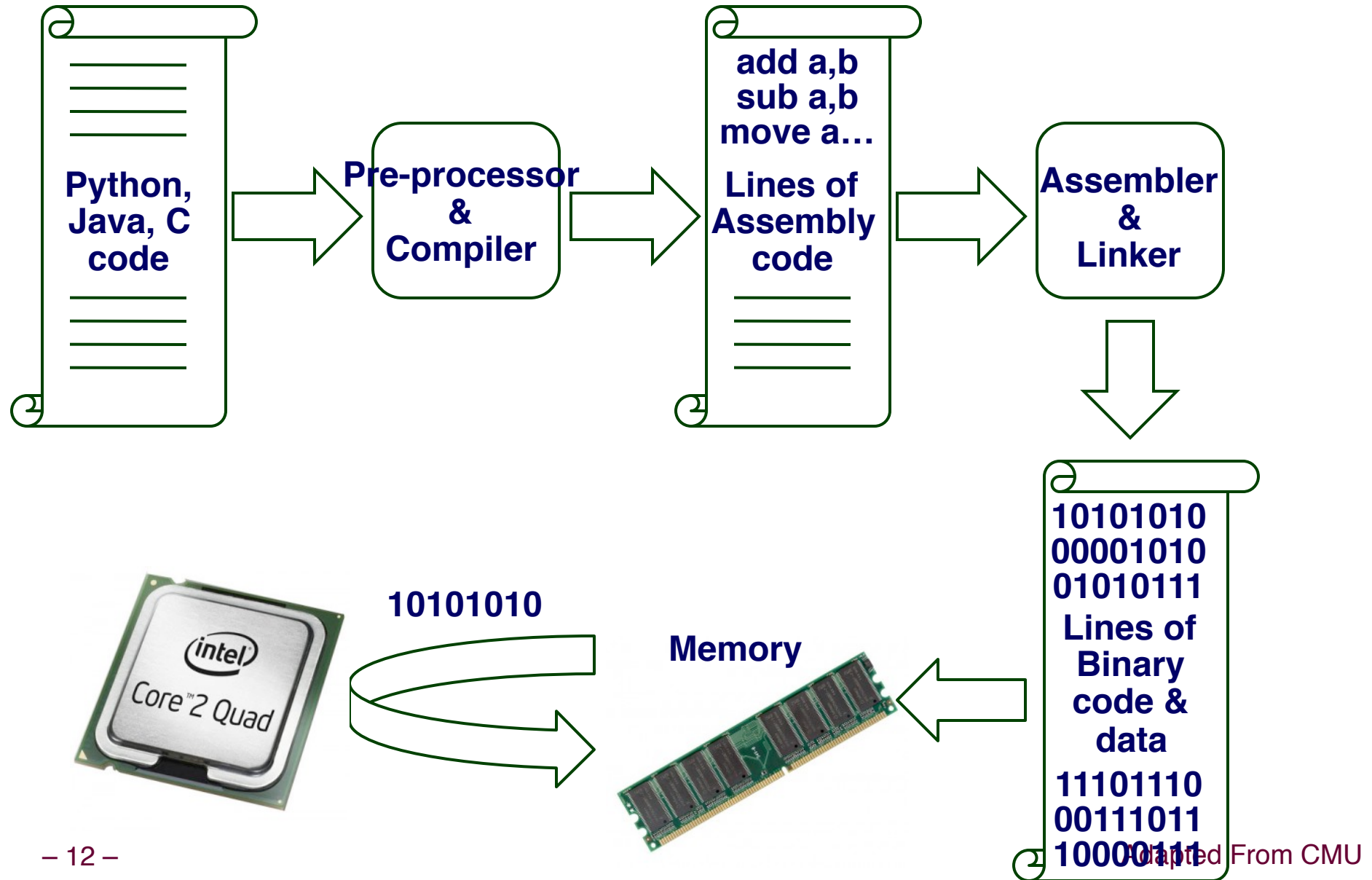
# Systems as a Springboard...



# Why C?

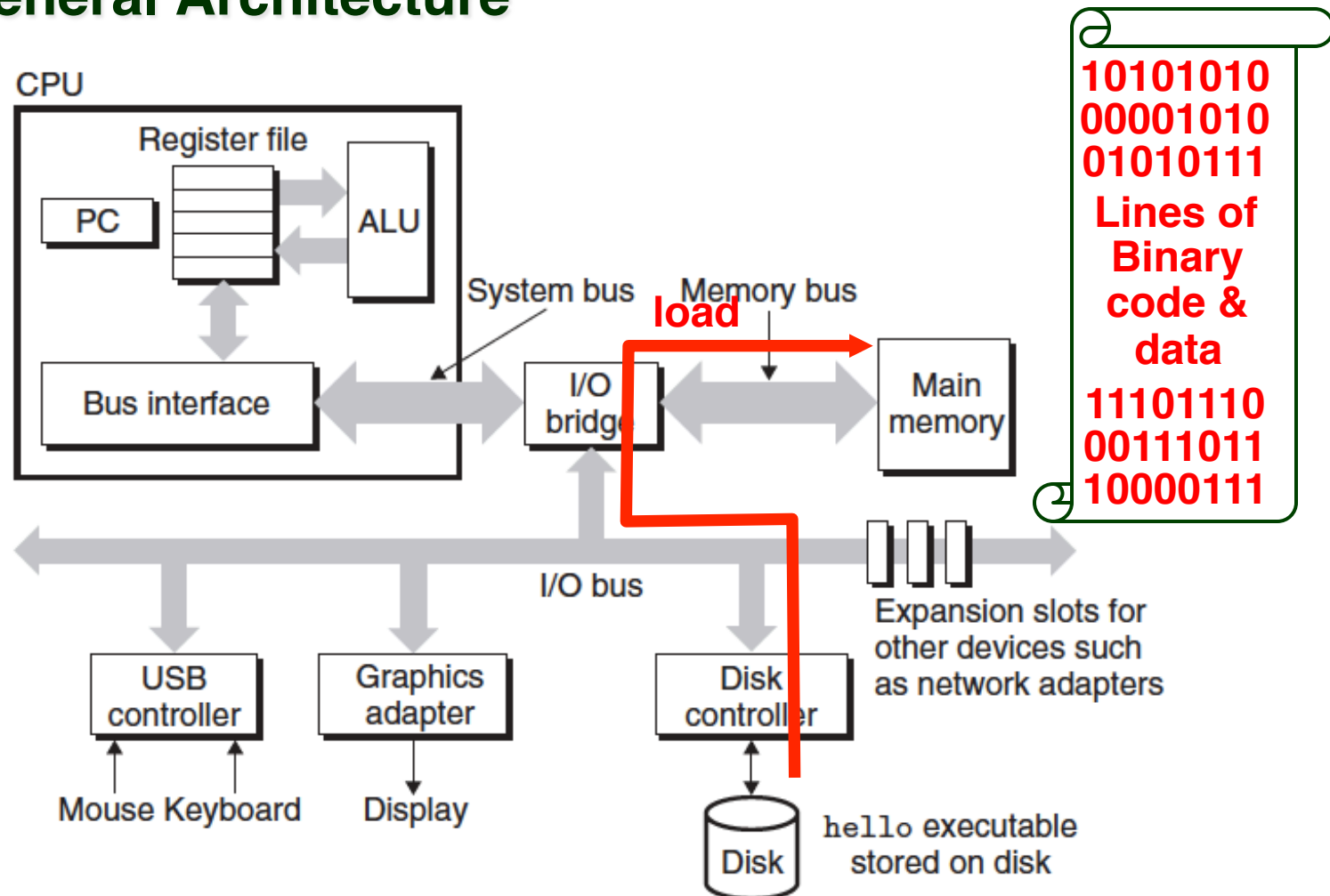
- **A good compromise between a high-level programming language and low-level hardware**
  - **Maps well to assembly and binary machine code instructions**
  - **Low level enough to manipulate memory with pointers**
    - **Learn from C about the dangers of pointer arithmetic, array out-of-bounds memory 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



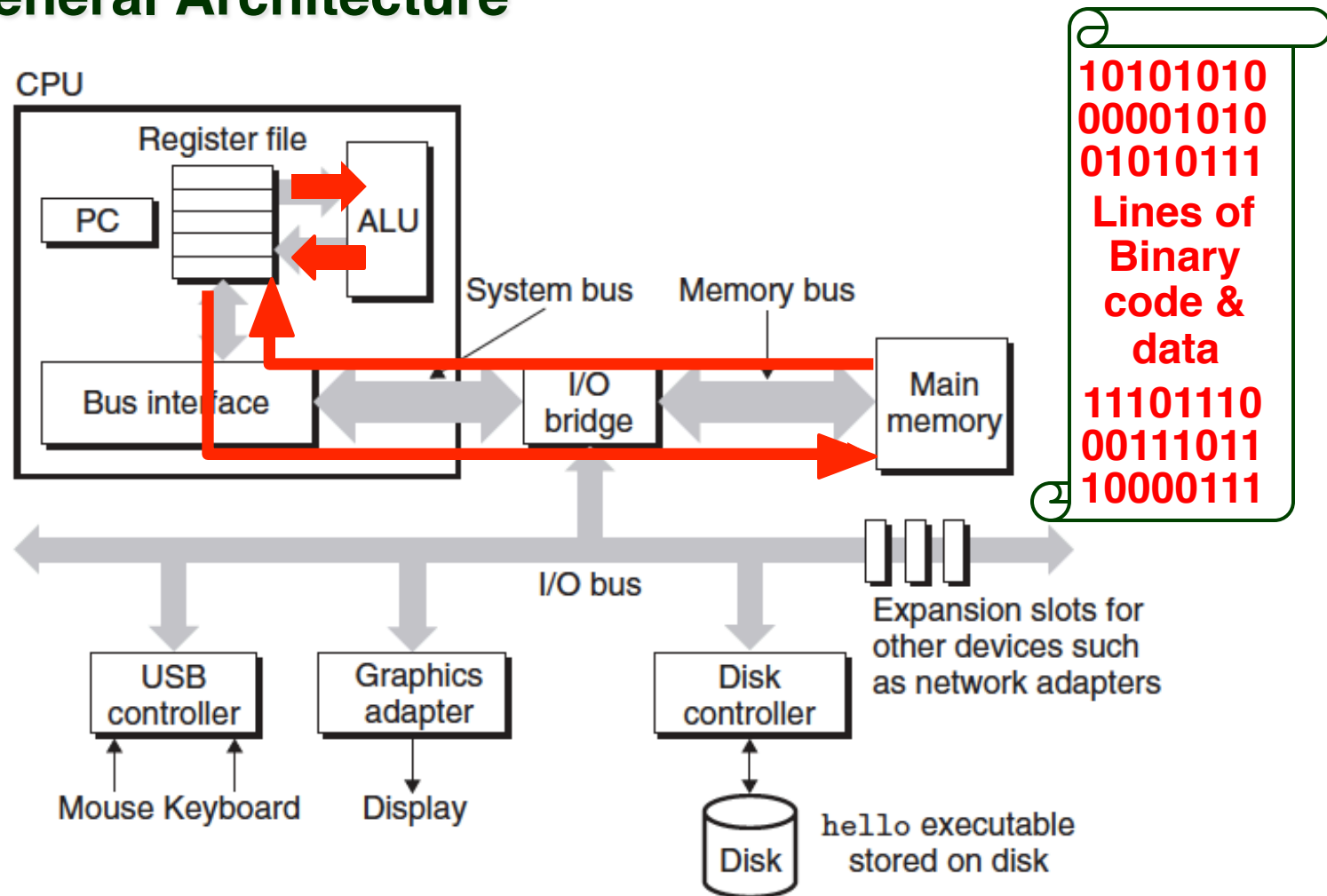
# Hardware Organization of a Computer System (Von Neumann)

## General Architecture



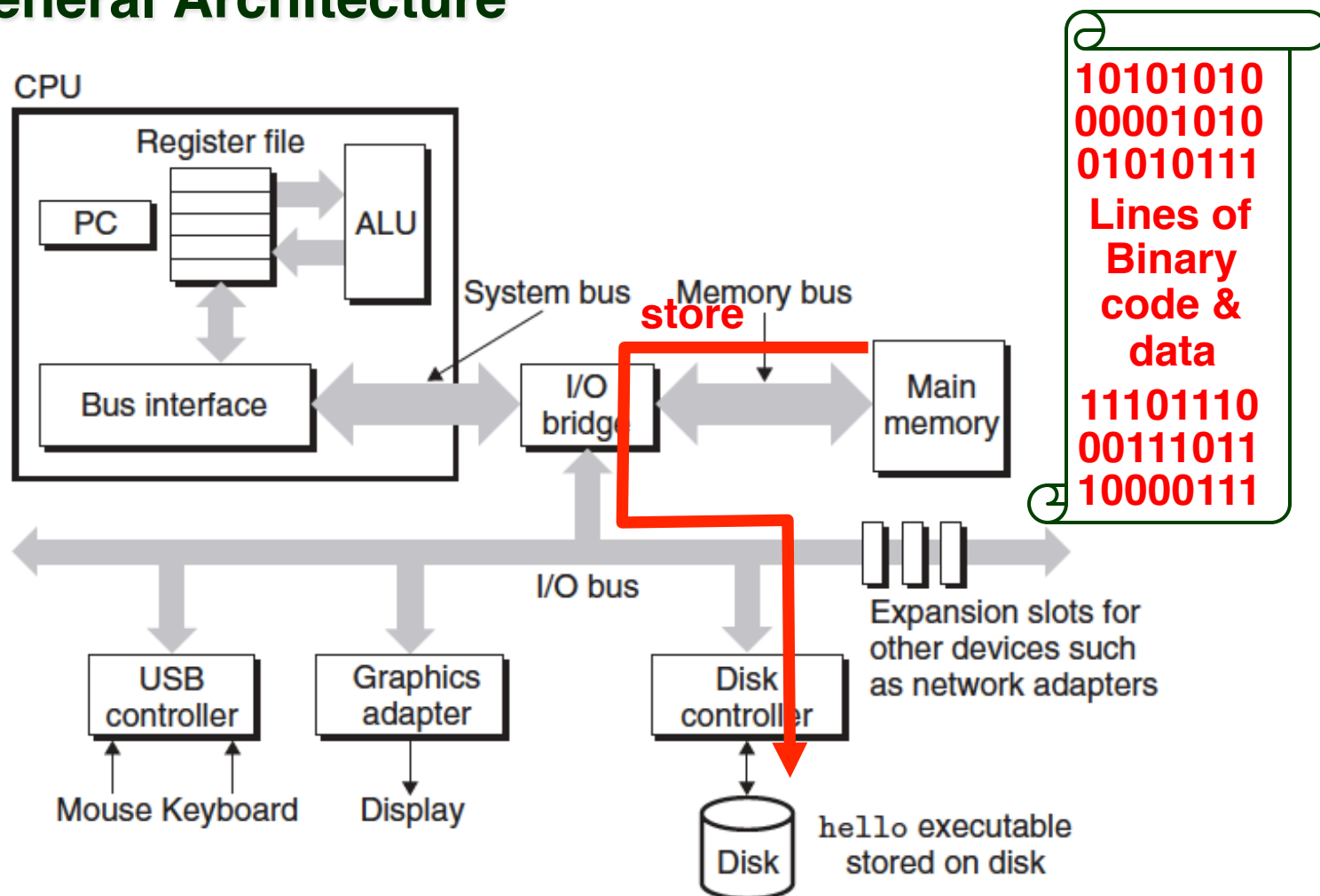
# Hardware Organization of a Computer System

## General Architecture



# Hardware Organization of a Computer System

## General Architecture



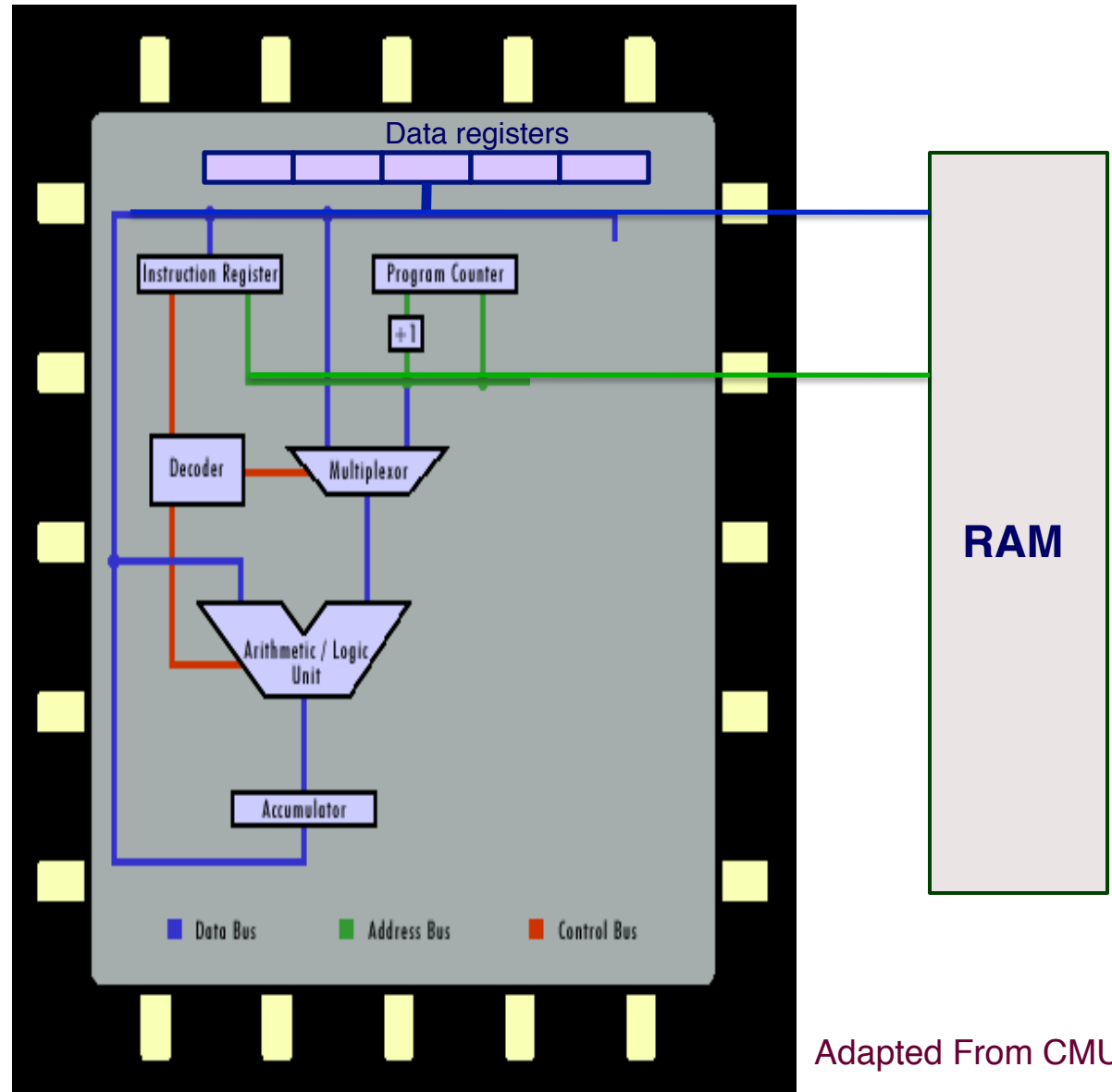
# Inside the CPU - Building Blocks

- 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





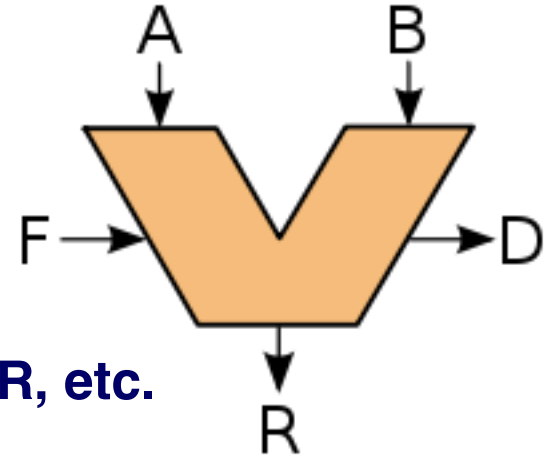
# Arithmetic Logic Unit (ALU) of CPU

- 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

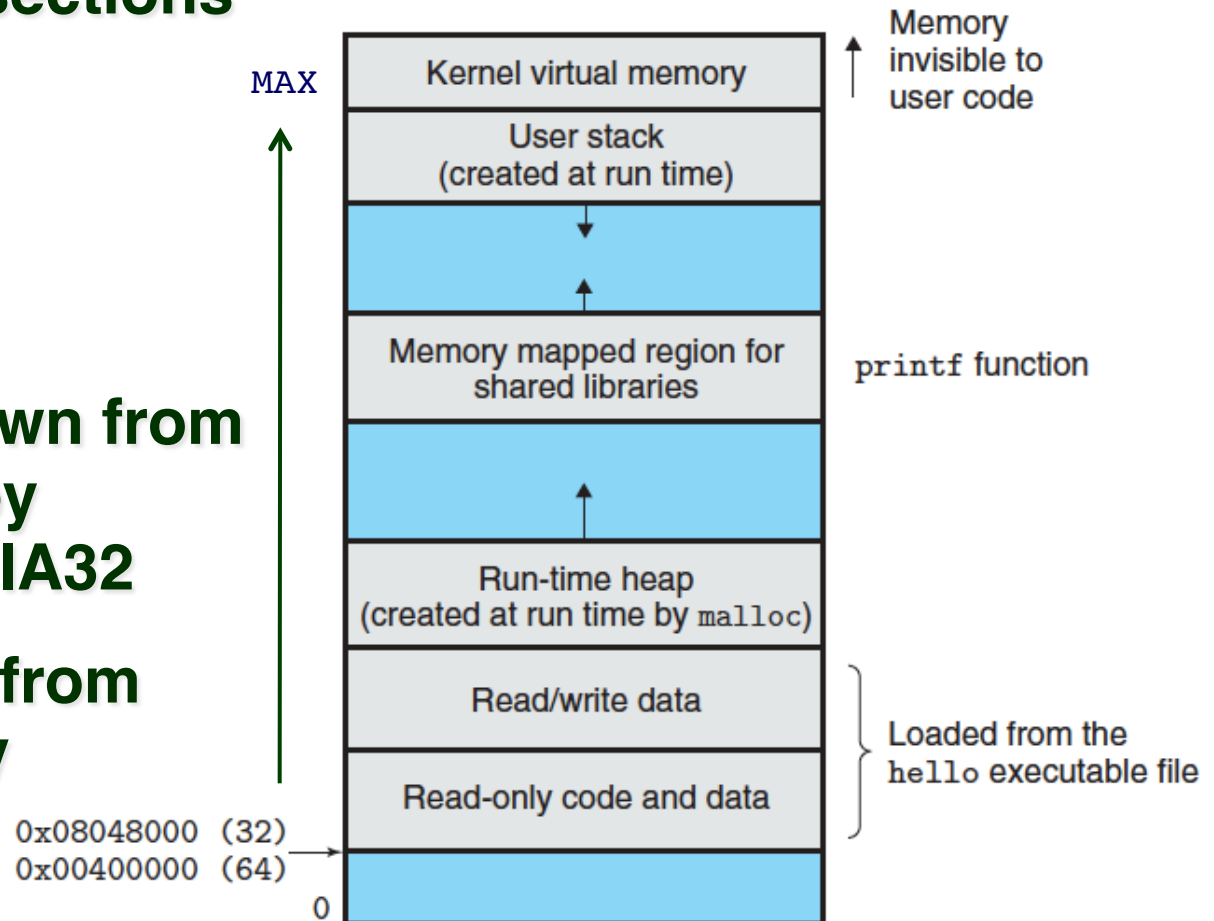
- As we will see, one type of instruction moves instructions and data to/from CPU from/to memory
  - i.e. fetch A and B from memory, move R to memory

# Layout of a Program in Memory

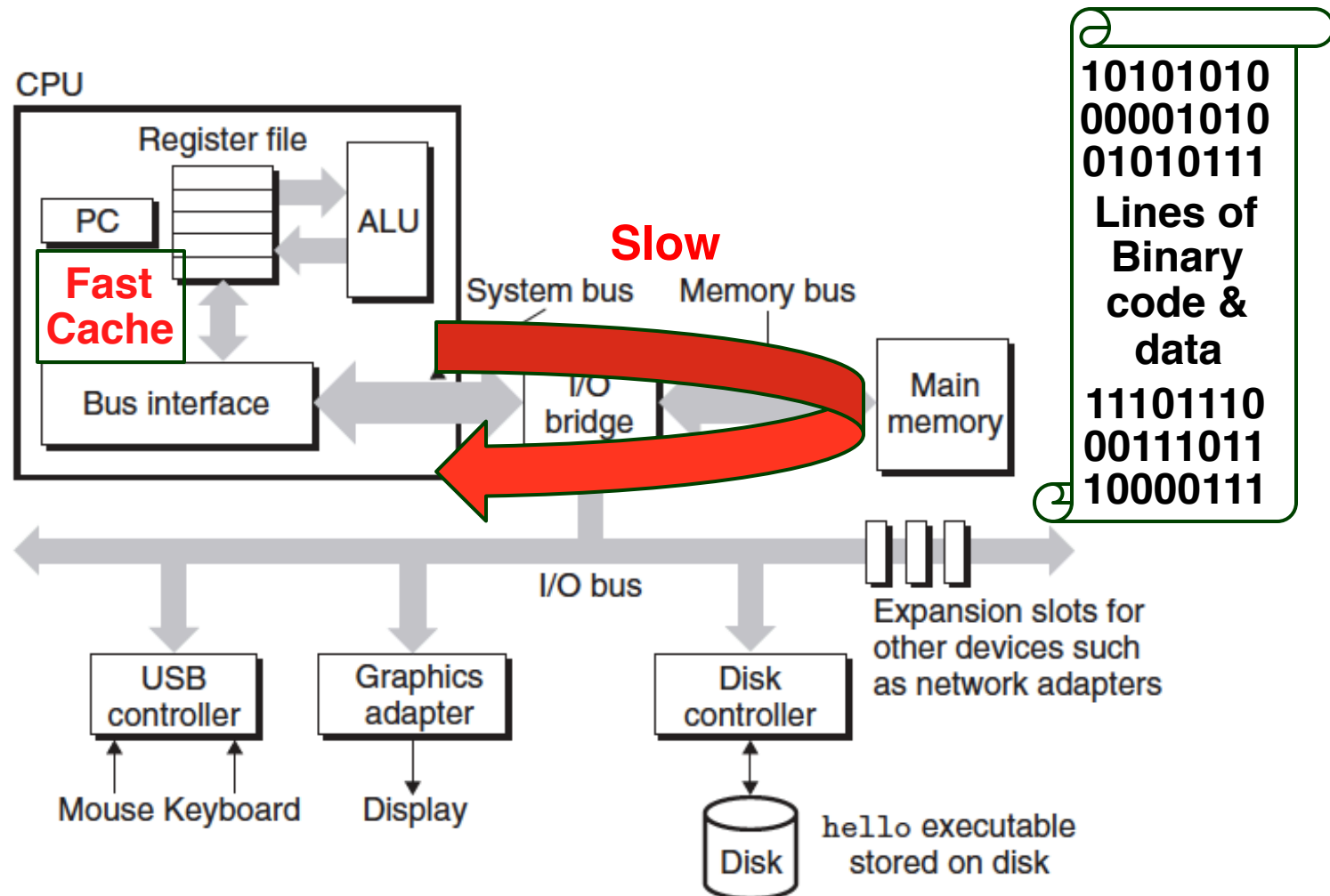
- Program memory is divided into 4 sections

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



# Cache For Faster Data/Code Access

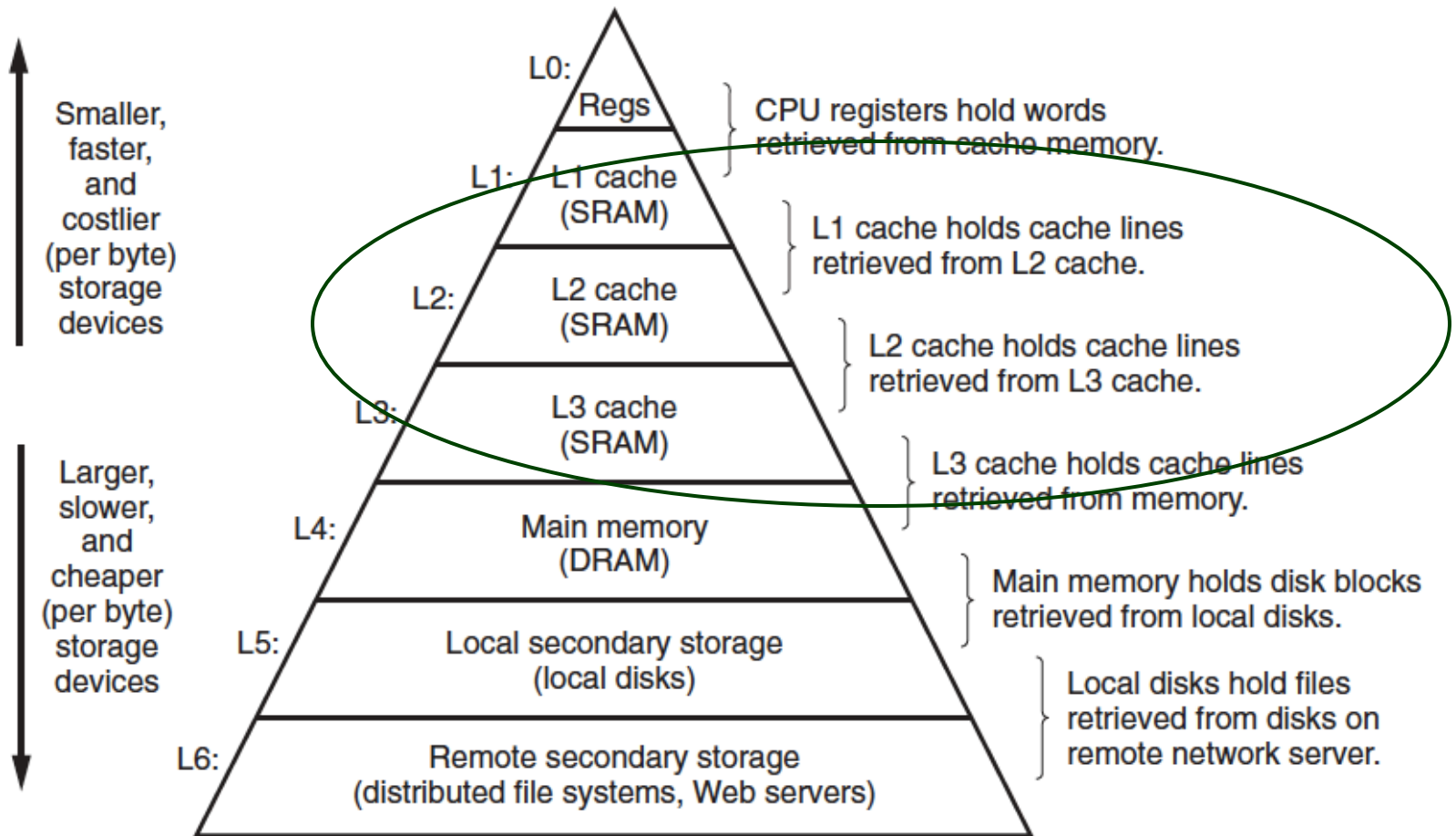


# Cache For Faster Data/Code Access

## ▪ Caching

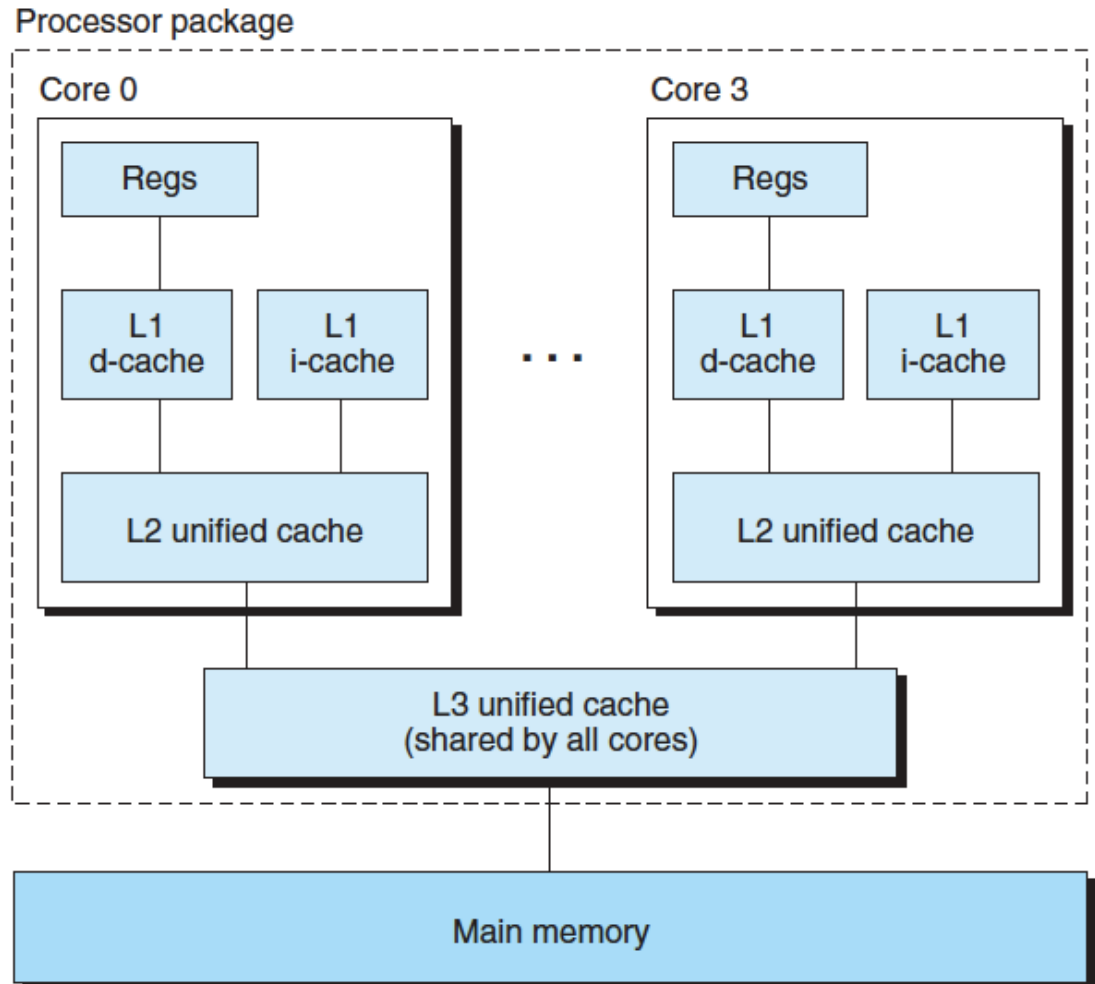
- Going to memory is quite slow compared to the speed of the CPU
  - RAM access time is ~ microseconds.
  - CPU executes multiple instructions per nanosecond.
  - So CPU waits ~1000 cycles to fetch data from memory!
- Solution: create a smaller but faster buffer called a cache, and cache data there that 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



# Memory Hierarchy

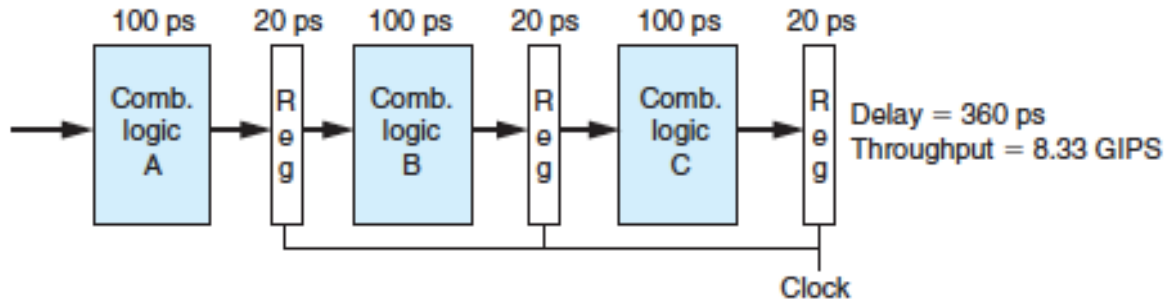
- **Intel Core i7 organization**
  - 4 cores
  - Each core has an L1 data cache, an L1 instruction cache, and a larger but slower L2 unified cache
  - Across cores, there is an L3 unified cache



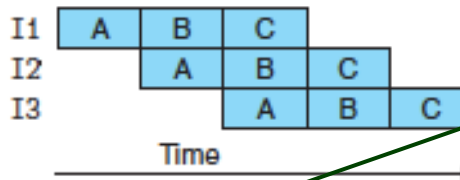
# Pipelining to Improve Throughput

## ■ 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