Computer Systems

CSE 410 Winter 2022

Instructor:

John Zahorjan

Teaching Assistants:

Yixiao Li, Suzanne Piver, Jack Zhang

Today's Agenda

- Administration
  - Course overview
  - Staff
  - General organization
  - Requirements, assignments, grading
  - Texts and references
  - Policies
- The course
  - What it's about, our perspective

Organization and Administration

Everything is on the course web page:

http://www.cs.washington.edu/410

Including

- General information, policies, syllabus
- Staff information, office hours (still working on that)
- Link to discussion board (still working on that too!)
- Calendar(s) with lecture slides, links to assignments, etc.
- Information and links to computing resources and reference info
- Etc

By the way

- You should have received email with your account information for klaatu.cs.washington.edu
- Homework 0 is out

Us

Instructor

John Zahorjan, CSE 434, zahorjan@cs

TAs

Yixiao Li

Suzanne Piver

Jack Zhang

Use the discussion board for most general interest communications.

Use cse410-staff@cs.washington.edu to contact (all) the course staff.

You

CSE 410 — Computer Systems (3)

CSE 373 — Data Structures and Algorithms (4)

CSE 143 — Computer Programming 2 (5)

CSE 142 — Computer Programming 1 (4)

You and CSE 410

- Our goal is to maximize useful things learned per minute of your time spent
- There will be some programming
  - In assembler
    - You'll never do this again
  - In C
    - We will not even attempt to give you enough experience to be a skilled C programmer
  - You should work on Linux machine klaatu.cs.washington.edu
- There will be "book questions"
- (There will be reading)

Hardware Architecture

- We'll be using RISC-V
  - Descendant of MIPS, ARM, PowerPC
  - Open source architecture
    - riscv.org
- We won't be using x86/AMD64
  - "Intel architectures"

Textbooks

- **None**
  - Course Documentation page
  - Google

- Computer Organization and Design: RISC-V Edition
  - David Patterson and John Hennessy
- Operating Systems: Principles and Practice
  - Tom Anderson and Mike Dahlin
- Computer Networks
  - Peterson and Davie
- The C Programming Language
  - Kernighan and Ritchie

Course Components

- 3 lectures per week (~30 total)
- Written assignments
- Programming assignments ("a few")
  - Assembly language is closely related to architecture
  - C is closely related to much of the course material
- Exams (midterm + final)
  - Taken remotely ("take home")
  - Test your understanding of concepts and principles

We have no idea whether we'll go back to in-person or will stay remote.

Policies: Grading

- Exams: midterm 10%, final 25% of total grade
- Written assignments: weighted according to effort required
- Programming assignments: weighted according to effort
  - These will likely increase in weight as the quarter progresses
- Grading (aprox.):
  - 60% assignments
  - 35% exams
  - 5% other

- Late policy
  - Use your judgement
  - Don't be unreasonable

- Academic integrity: policy on course web
  - I trust you to do what best helps your learn the material
    - The goal isn't a completed homework…
  - I have no sympathy for trust violations – nor should you

End of Part 1

- Questions?

What is this class about?

- You've done extensive Java programming
- You understand computers at the level of the Java language
- How is that language supported? What is required to execute your program?
  - What does computer hardware do?
  - How is it built?
  - What is the role of the compiler?
  - What is the role of the Java runtime system?
  - What is the role of the operating system?
  - How do these components support building and running applications?
  - How do networks work?

Computer Systems

- What do we mean by "systems"?
  - Hardware and software whose purpose is to enable/facilitate creating other hardware or software
  - A "system" doesn't do anything itself, but enables efficiently creating an application, say, that does

- Efficiency
  - One version is how much work it is to create a correctly functioning application
    - "Static"
  - Another is how much time that application takes to do its job when it is run
    - "Dynamic"

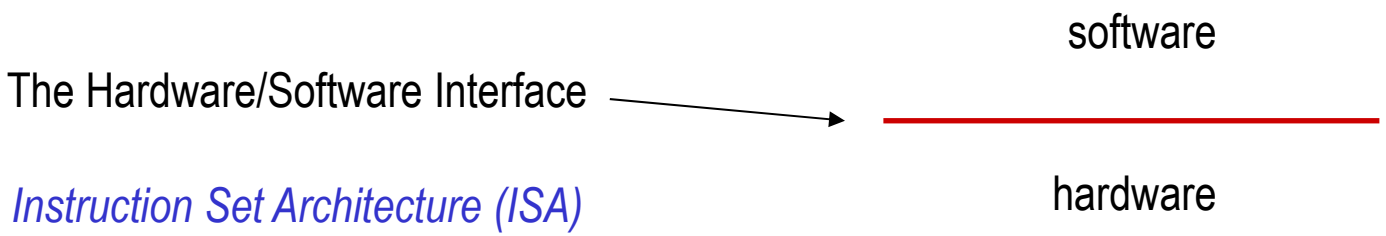
This Course

- It's about interfaces
  - and the implementation of those interfaces.
- We intend to go broad rather than deep
  - Maximize useful information per minute of effort
  - Limit workload to what's appropriate for a 3 hour course
- When done, you should have a big picture understanding of how computer systems work
  - From the idea of a Java program in my head to the implementation of that program writing data into a file, what has happened?
- You'll end up knowing things most CSE majors do not. (But they'll know many things you don't as well.)
- I hope we can identify "themes" that apply at all levels

Some Themes

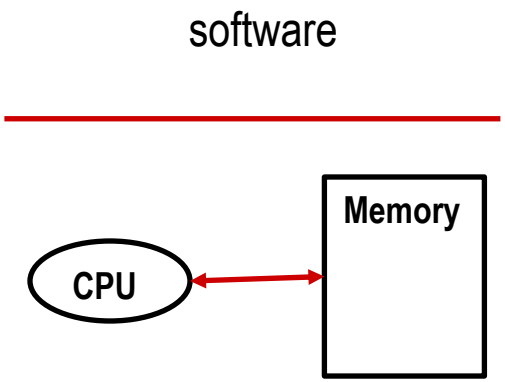
- "Simpler is faster"
- Static vs. Dynamic Evaluation
- Representation and Translation
- Interfaces vs. Implementation
  - Layers, not options
  - Policy vs. mechanism
  - Interposition to evolve functionality
- Naming / Virtualization
- Parallelism / Concurrency
  - Atomicity
- Trading space for time

What This Course is About: The Instruction Set Architecture



What This Course is About: Hardware Components

int x;
*x = 10 + 3 * 4;*



CPU: Central Processing Unit
Executes instructions

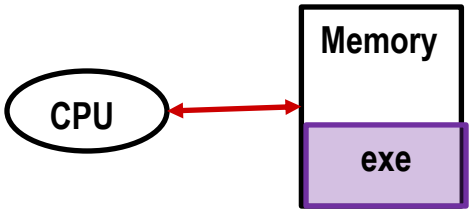
Memory (RAM)
Holds values

What This Course is About: Static vs Dynamic

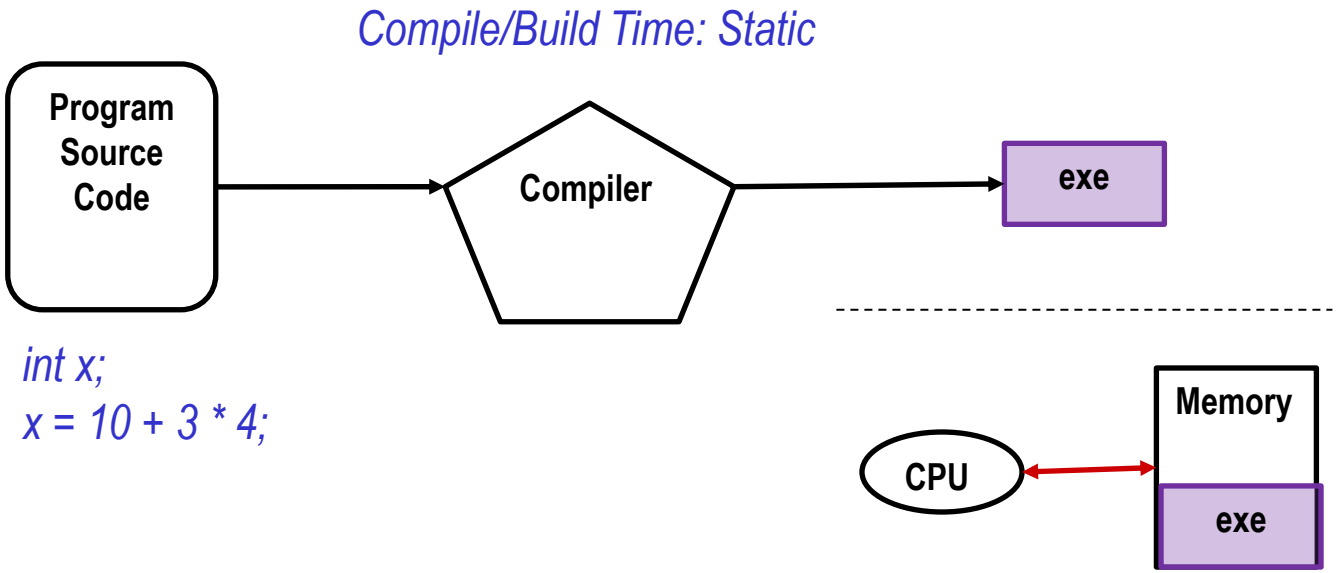
exe: executable file

Program: Static

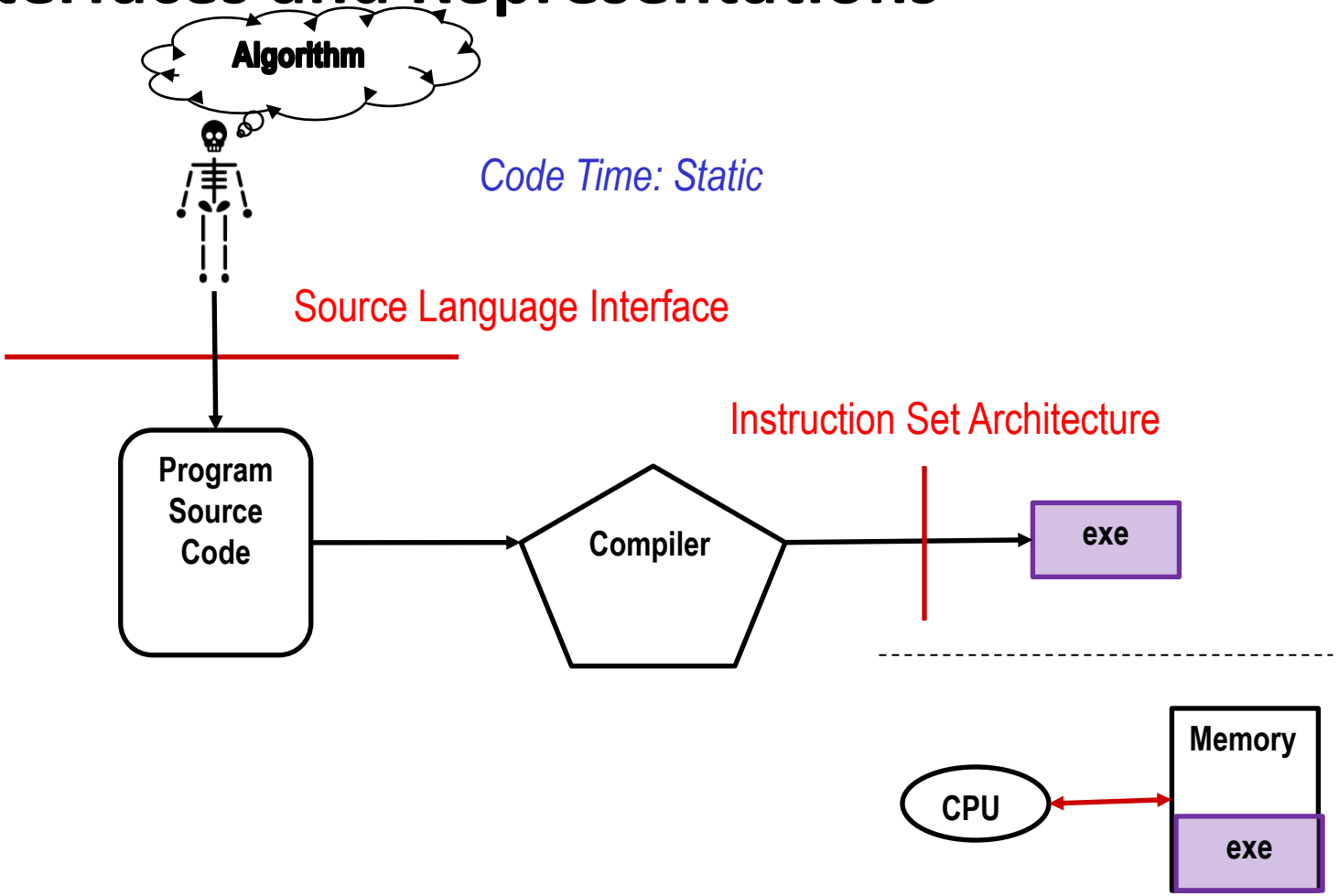
Execution: Dynamic



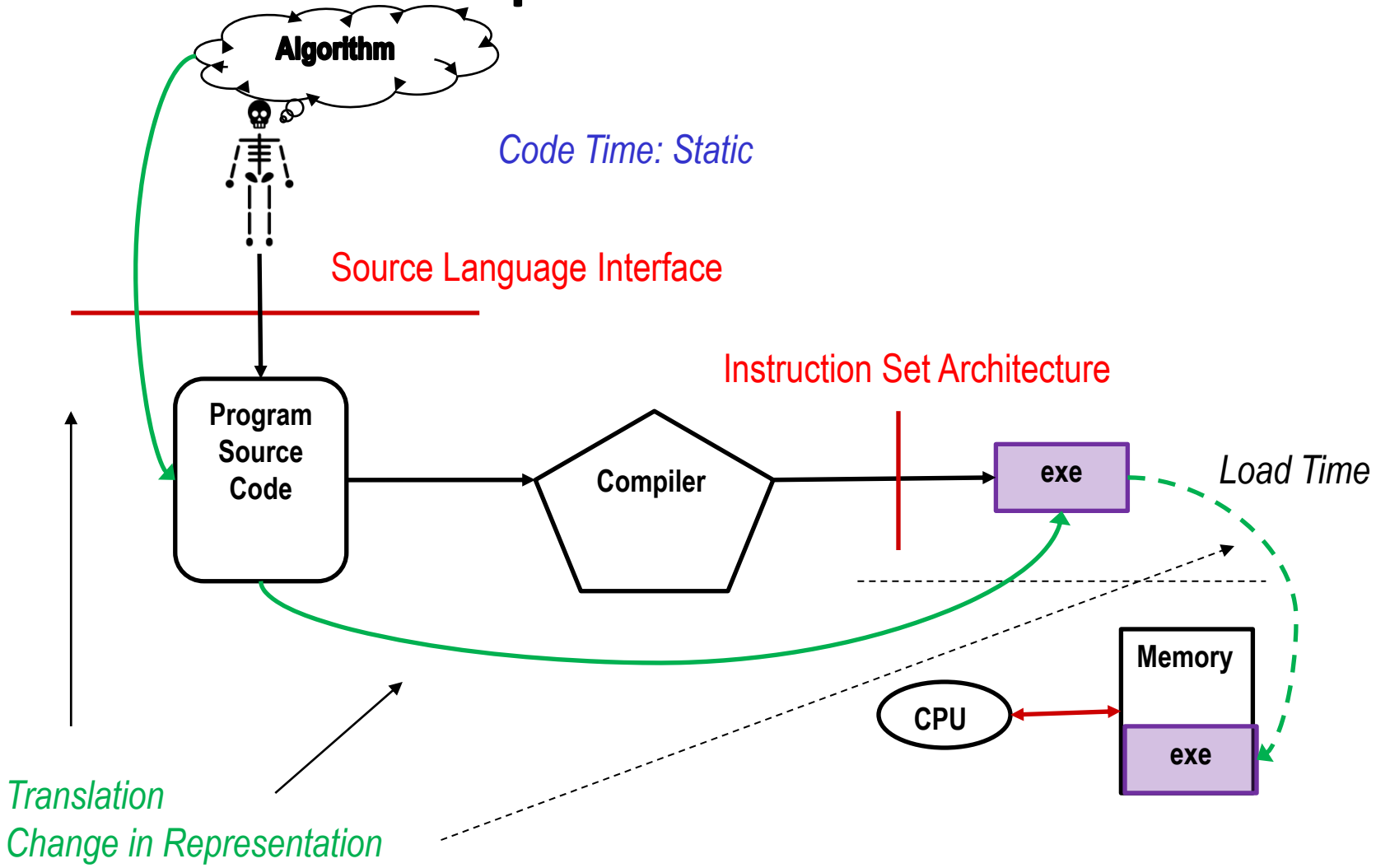
What This Course is About: Compiling / Building Applications



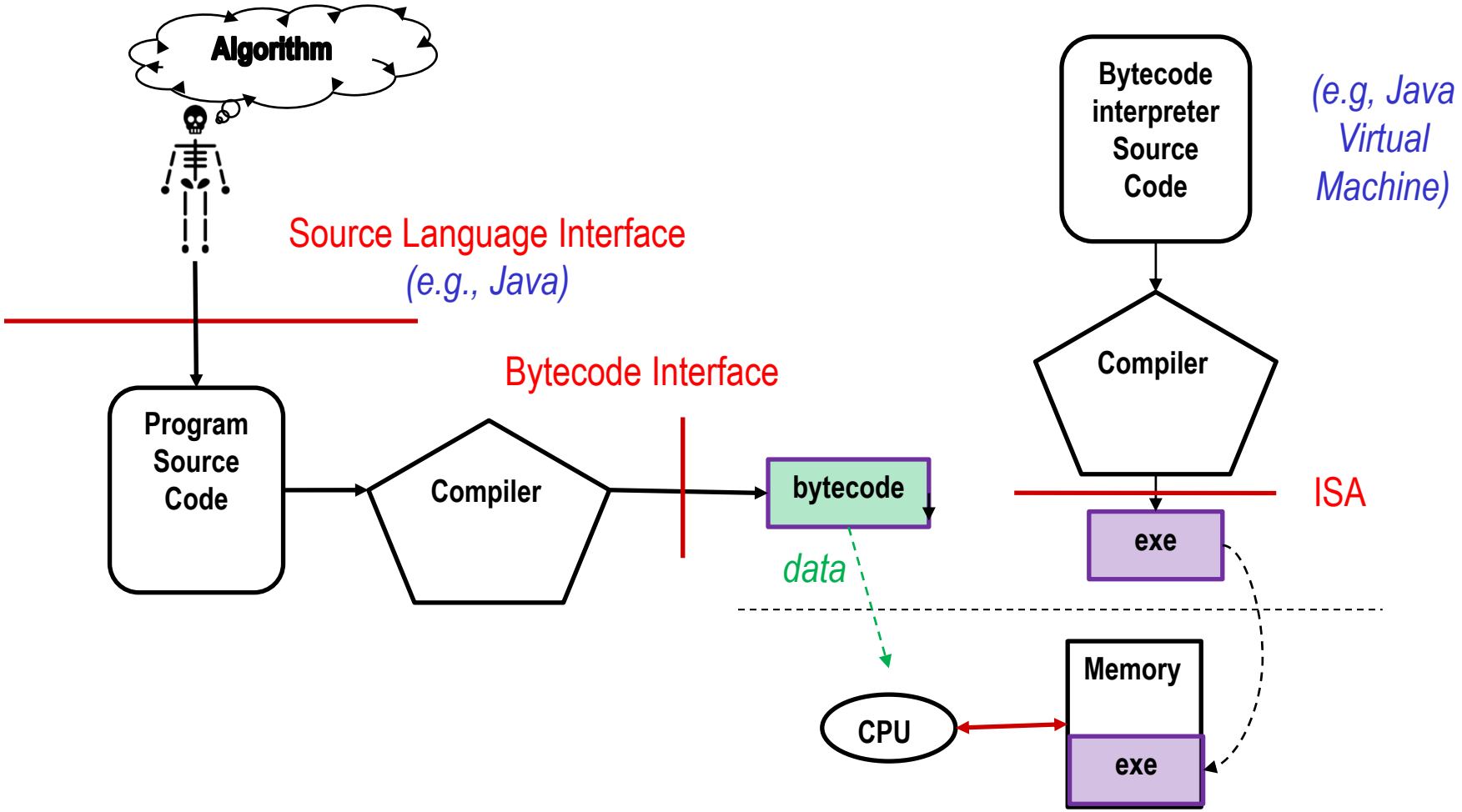
What This Course is About: Interfaces and Representations



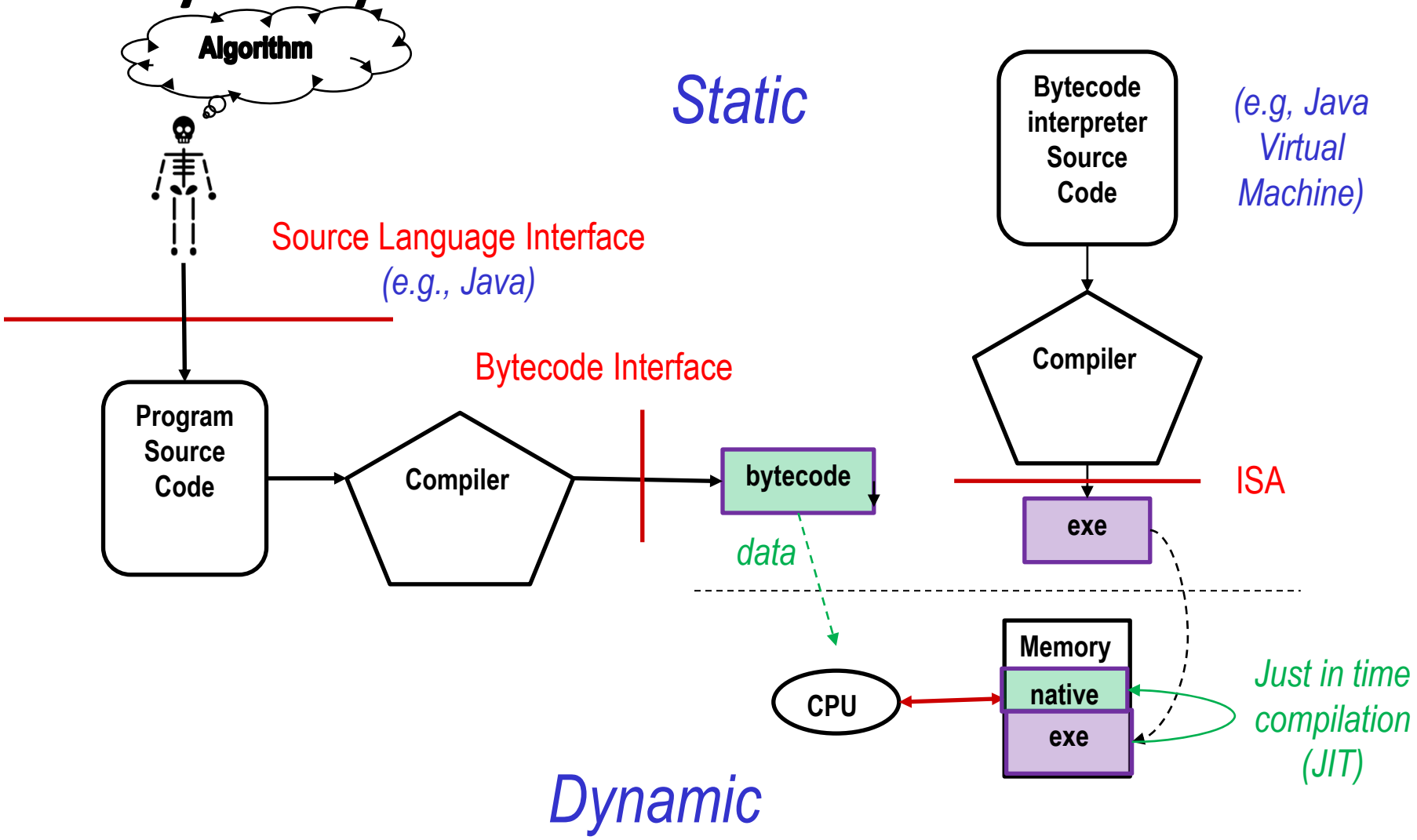
What This Course is About: Interfaces and Representations



What This Course is About: Many Choices for Layers

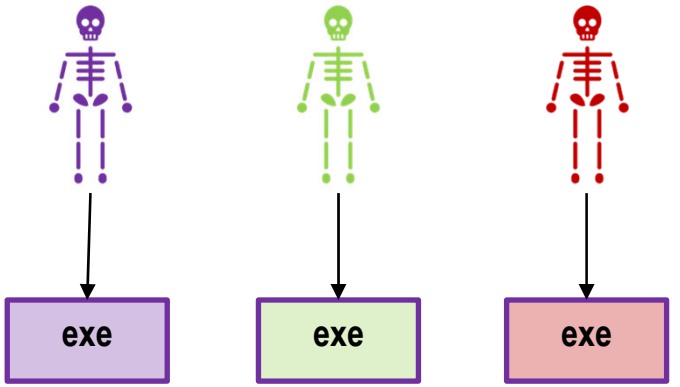


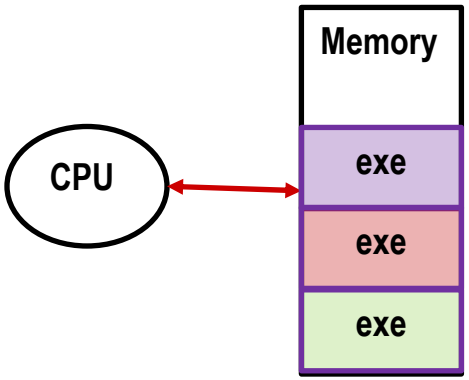
What This Course is About: Really Many Choices



What This Course is About: Shared Use of the Hardware

1. *Who loads the exe file into memory in the first place?*
2. *How do programs that know nothing about each other share the hardware?*



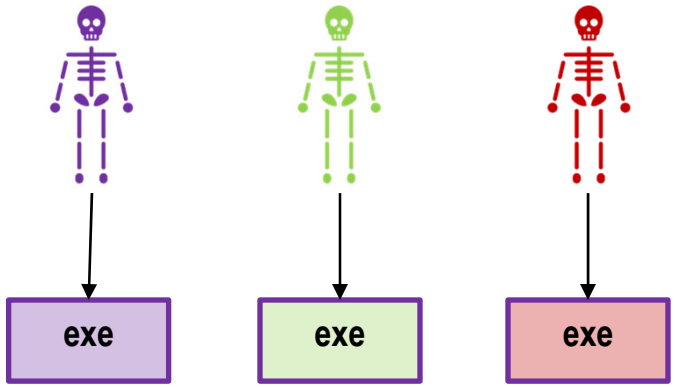


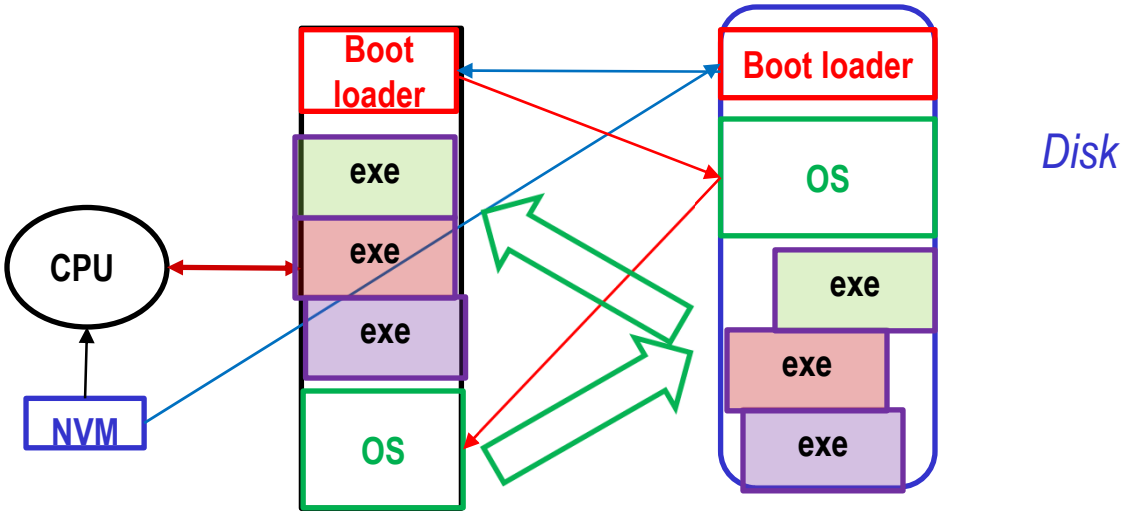
What This Course is About: The OS

1. *Who loads the exe file into memory in the first place?*
   A: The operating system (OS)

   Who loads the OS?!
   A: The boot loader program

   Who loads the boot loader?!
   A: Program in non-volatile memory



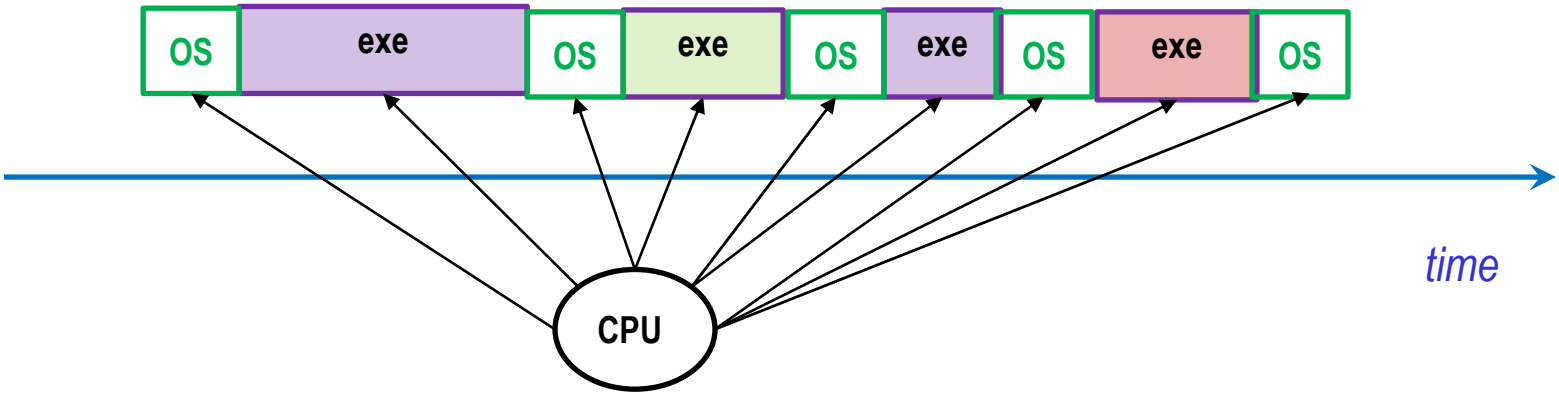


"Boot"

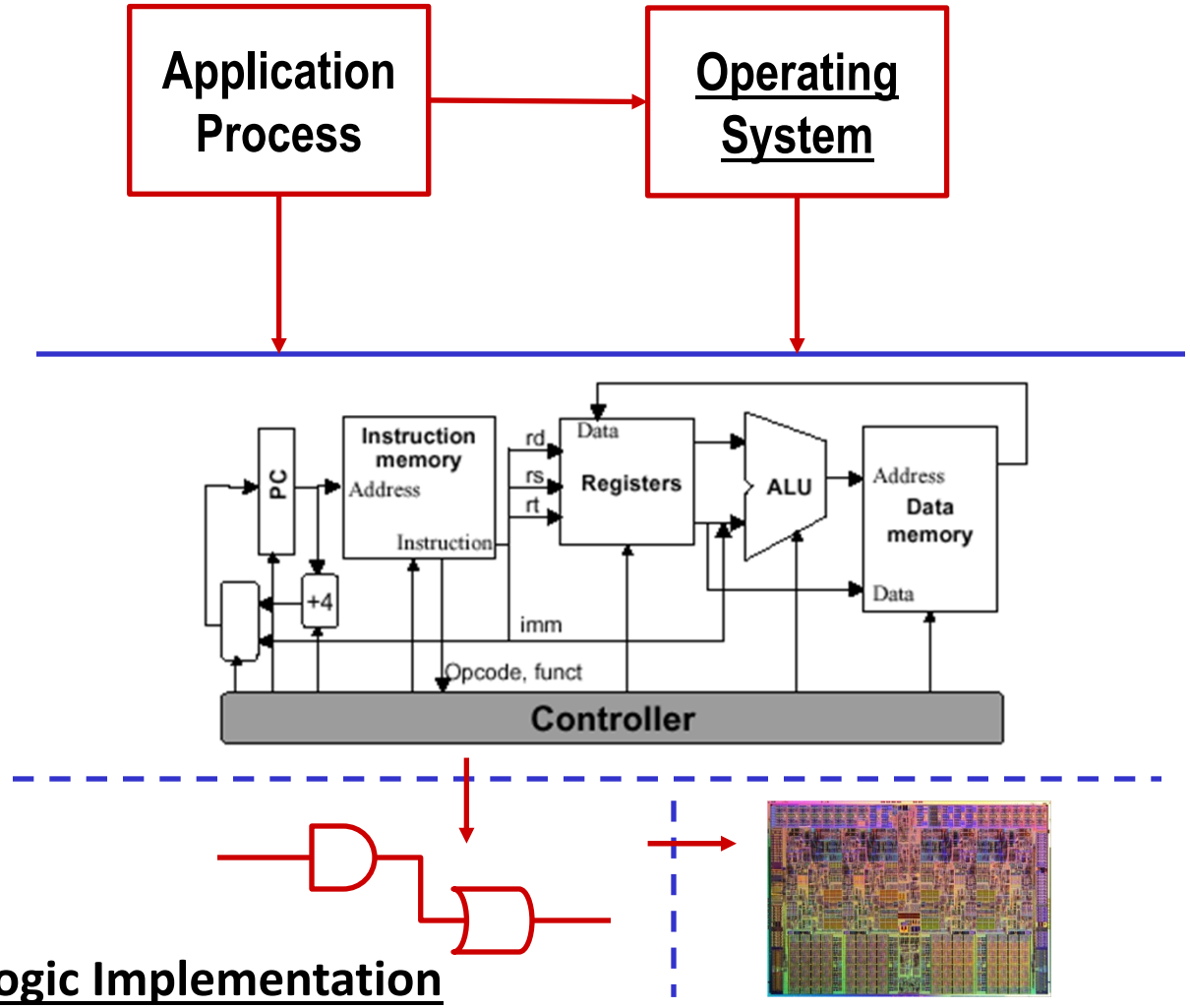
Non-volatile memory

What This Course is About: The OS

2. *How do programs that know nothing about each other share the hardware?*
   A: The operating system (OS) and the hardware together allow the OS to yank the CPU away from a program while it's running and give it to a different program



What This Course is About: Machine Organization



Software in Execution

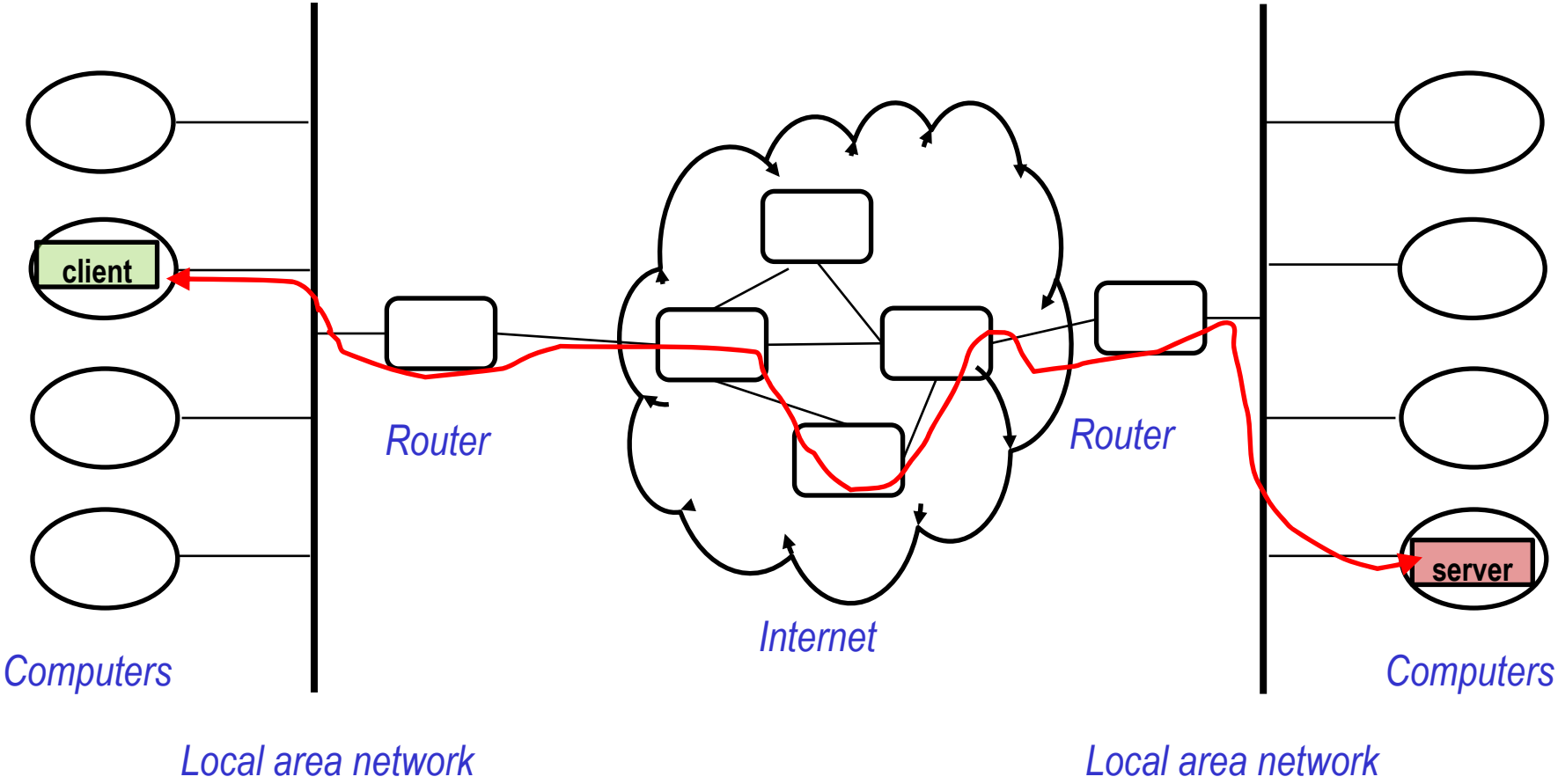
Instruction Set Architecture
(e.g., x86-64)

Machine Organization
(e.g., Core i7-8550)

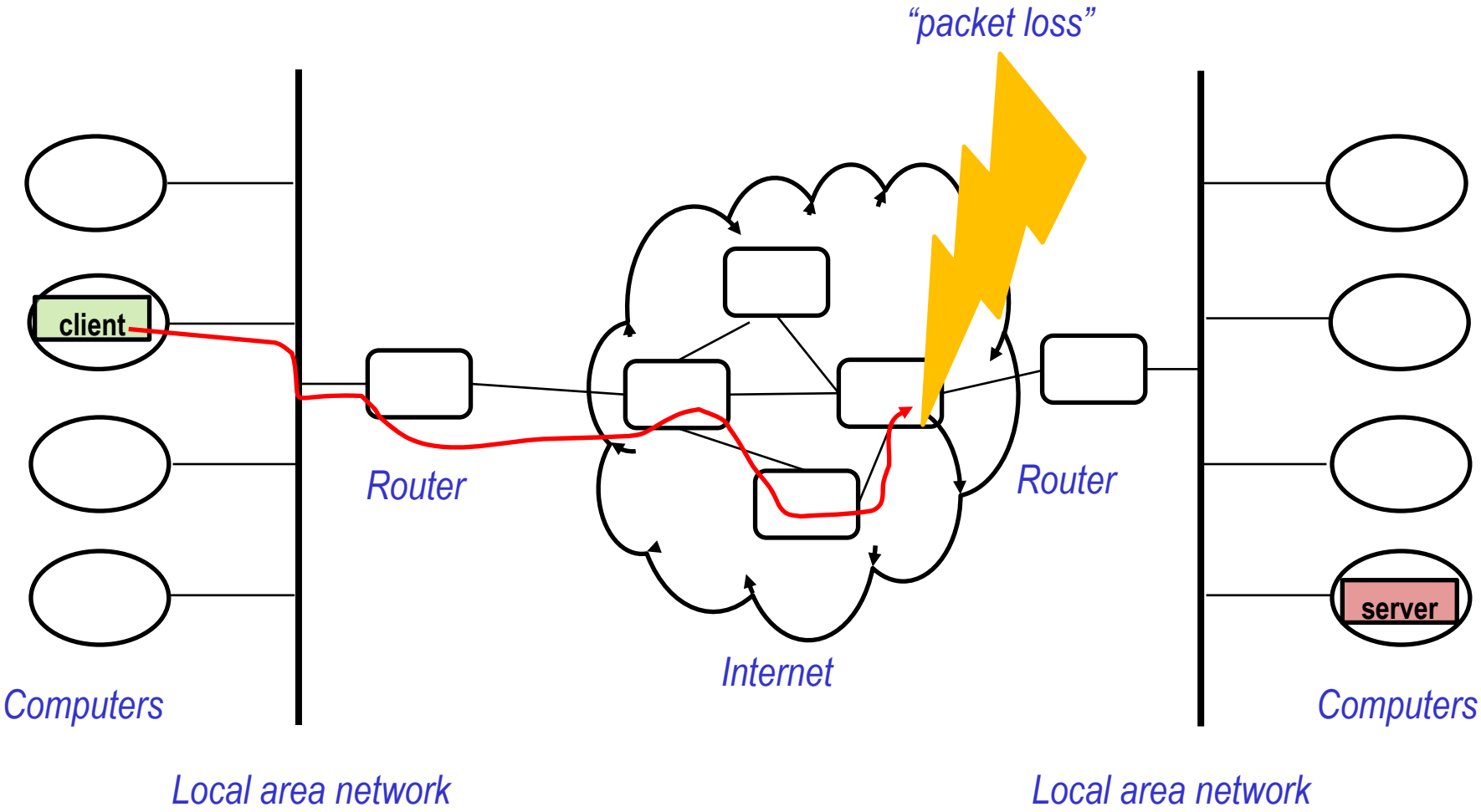
Hardware

Logic Implementation

What This Course is About: Networking



What This Course is About: Networking / Errors



Themes: Interfaces

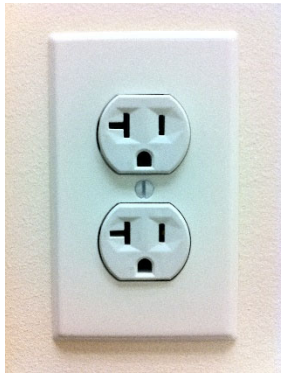
- Interfaces provide abstraction
  - They separate how to use a component from how the component is implemented
  - Here's an interface:

  - The interface stays the same even if what's behind it changes (hydro vs. coal vs nuclear vs wind ...)
  - The interface makes few requirements on what uses it (toasters, USB chargers, lamps, …)
  - ⇨The interface promotes innovation
    - Both above it and **below it**

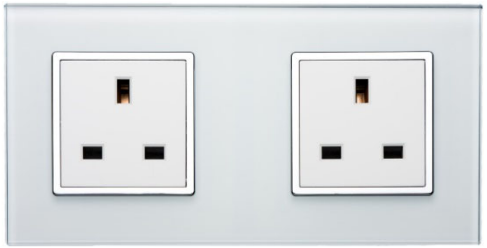
Interfaces

- Backward compatible changes to interfaces are good





- Incompatible changes are bad

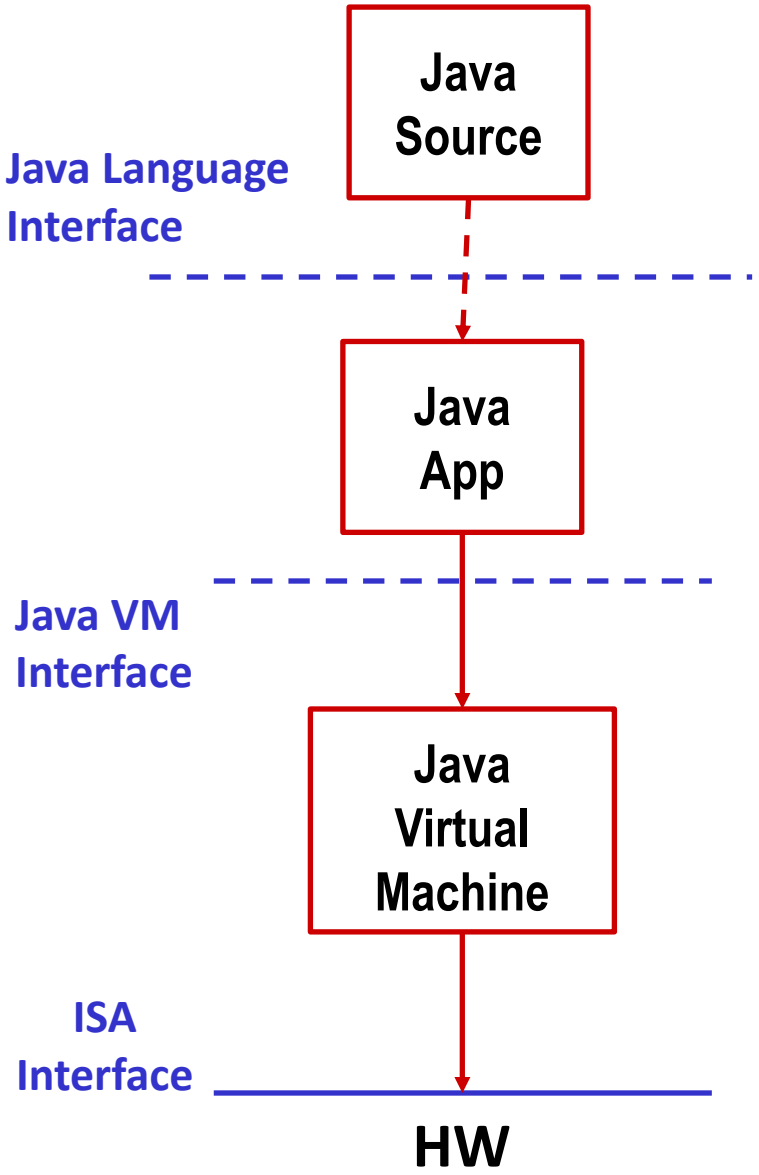


Layering?

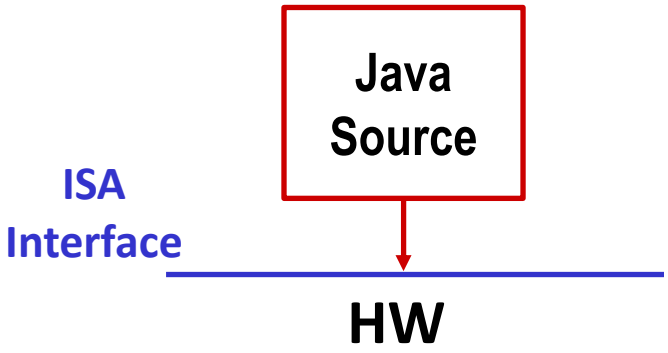
No Layering

Layering & Translation

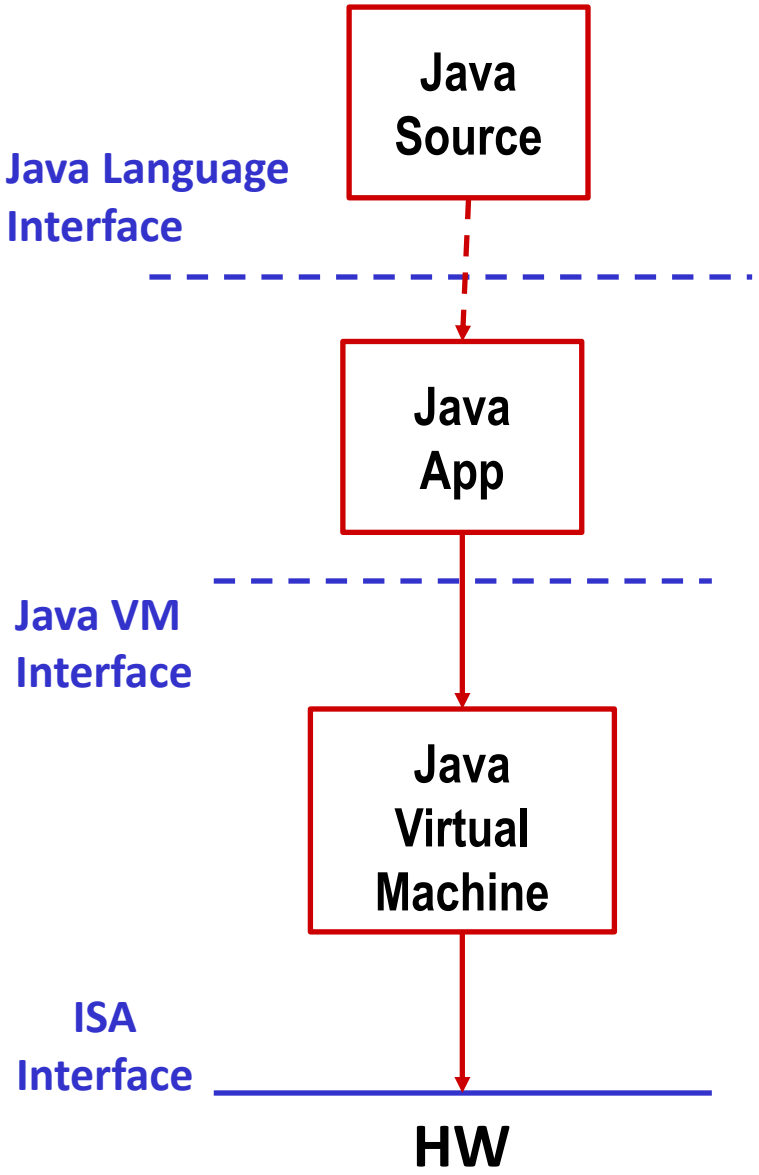
Themes: Layering *(Simpler is Faster)*



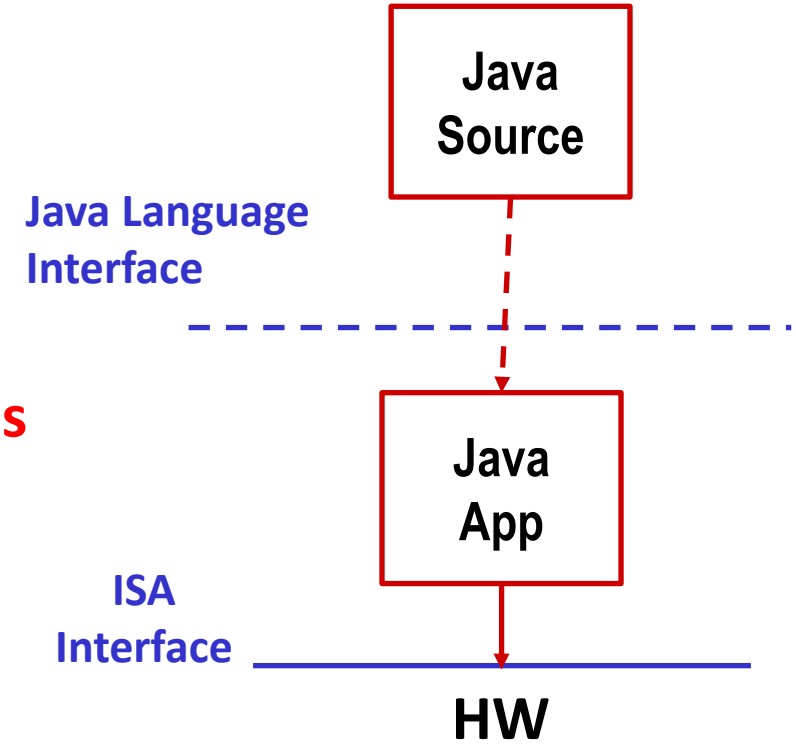
versus



How About This?



versus



Themes: Translation / Representation

- A "program" is written against (using) some interface
  - Java program → Java language interface
    - plus Java library interfaces
  - C program → C language interface
  - Code running on HW → ISA interface
  - ISA interface → machine organization interface
  - machine organization interface → logic interface
  - logical interface → hw implementation
- In general, higher level interfaces are more expressive
  - We prefer them because it's easier to say what we want
  - Except that if they're very expressive in some domain they're probably very clumsy to use in other domains
- Actual execution, though, relies on low level interfaces
  - For example, it's faster for the hw to be primitive? Why?
- Main idea: write to a high level interface and use a program to automatically translate to an equivalent lower level interface for execution
  - A "compiler"

Example: C, assembly, and machine code languages (interfaces)

```
if (x != 0) y = (y+z)/x;
```

```
cmpl    $0, -4(%ebp)
je      .L2
movl    -12(%ebp), %eax
movl    -8(%ebp), %edx
leal    (%edx, %eax), %eax
movl    %eax, %edx
sarl    $31, %edx
idivl   -4(%ebp)
movl    %eax, -8(%ebp)
```

```
10000011011111000010010000011100000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111111010000011111
11110111011111000010010000011100
10001001010001000010010000011000
```

Example of Translation / Representation

```
if (x != 0) y = (y+z)/x;
```

```
cmpl    $0, -4(%ebp)
je      .L2
movl    -12(%ebp), %eax
movl    -8(%ebp), %edx
leal    (%edx, %eax), %eax
movl    %eax, %edx
sarl    $31, %edx
idivl   -4(%ebp)
movl    %eax, -8(%ebp)
```

```
10000011011111000010010000011100000000000
0111010000011000
10001011010001000010010000010100
10001011010001100010010100010100
10001101000001000000010
1000100111000010
11000001111110100001111
11110111011111000010010000011100
10001001010001000010010000011000
```

- The three program fragments are equivalent
- You'd would rather write C! – a more human-friendly language

Course Outcomes

- Understanding the fundamentals of what is happening in going from creating a source file to running a program and obtaining its output
- Understanding some of the abstractions that exist between programs and the hardware they run on, why they exist, and how they build upon each other
- Knowledge of key details of underlying implementations
- Become better at thinking about problem solving in ways that have proven effective in computing