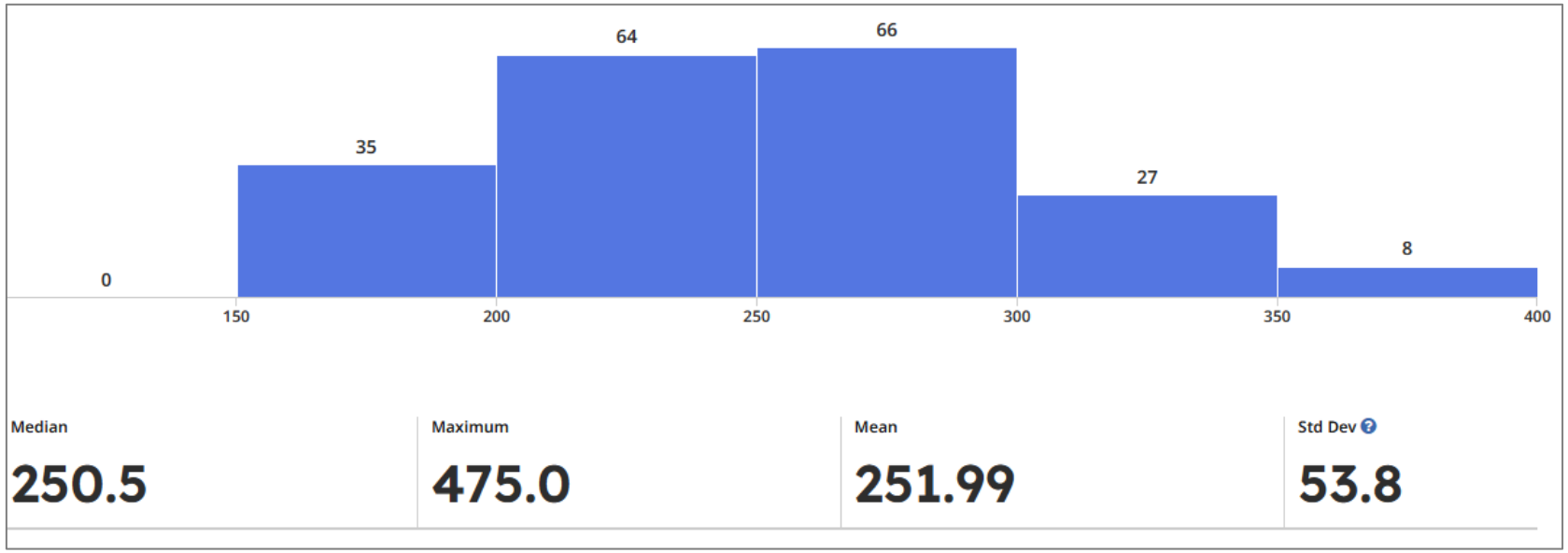
CS 211: Intro to Computer Architecture

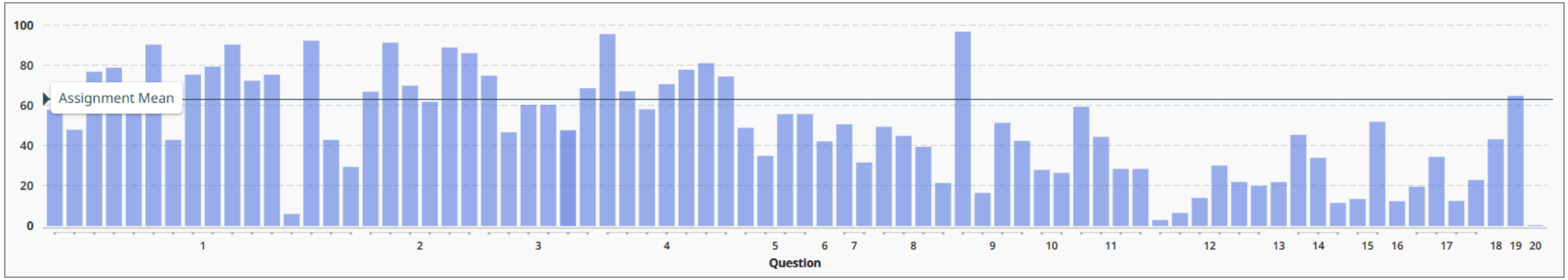
9.1: The Hardware-Software Interface

Minesh Patel

Spring 2025 – Tuesday 25 March

Midterm Exam Statistics





Announcements

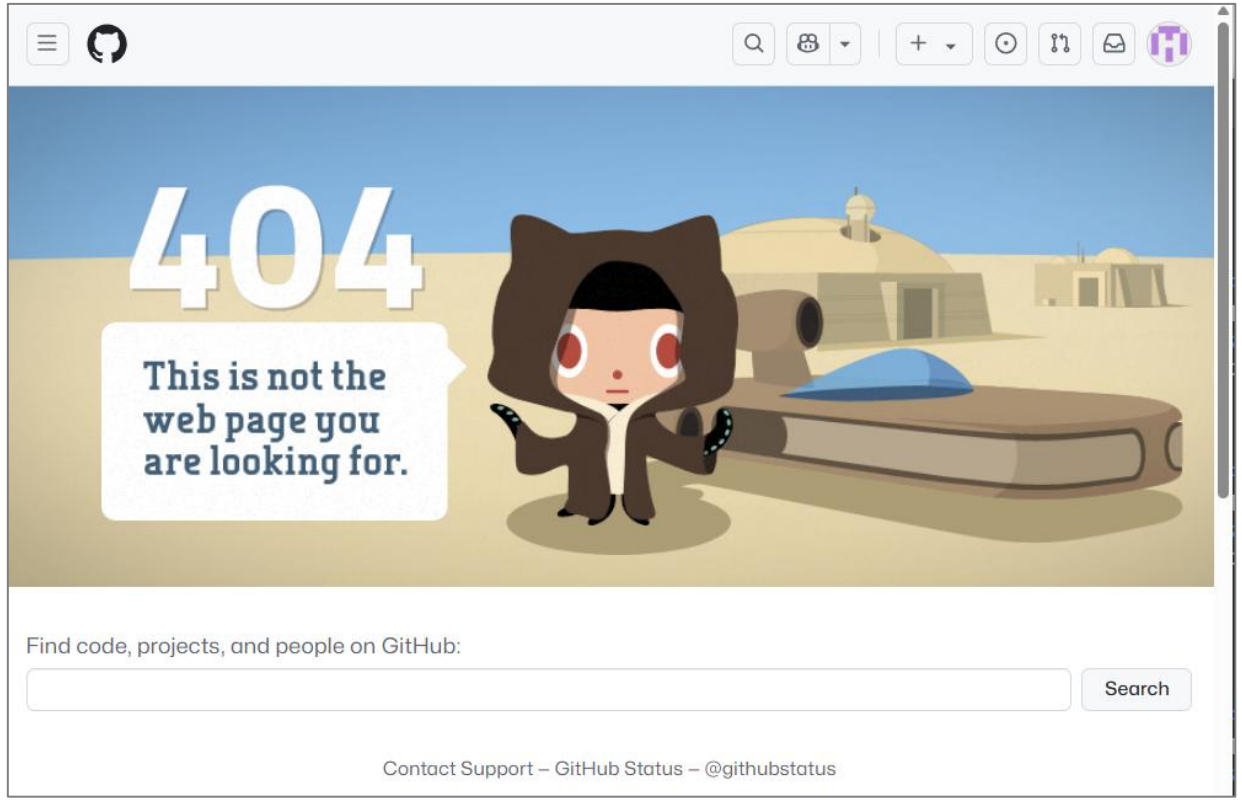
- Assignments
  - **PA3**: due **Friday @ 23:59**
  - **Extra Credit:** replaces WA6, due in **two weeks**

Agenda

- **Aside**: PA3 Filesystem Permissions

Filesystem Permissions

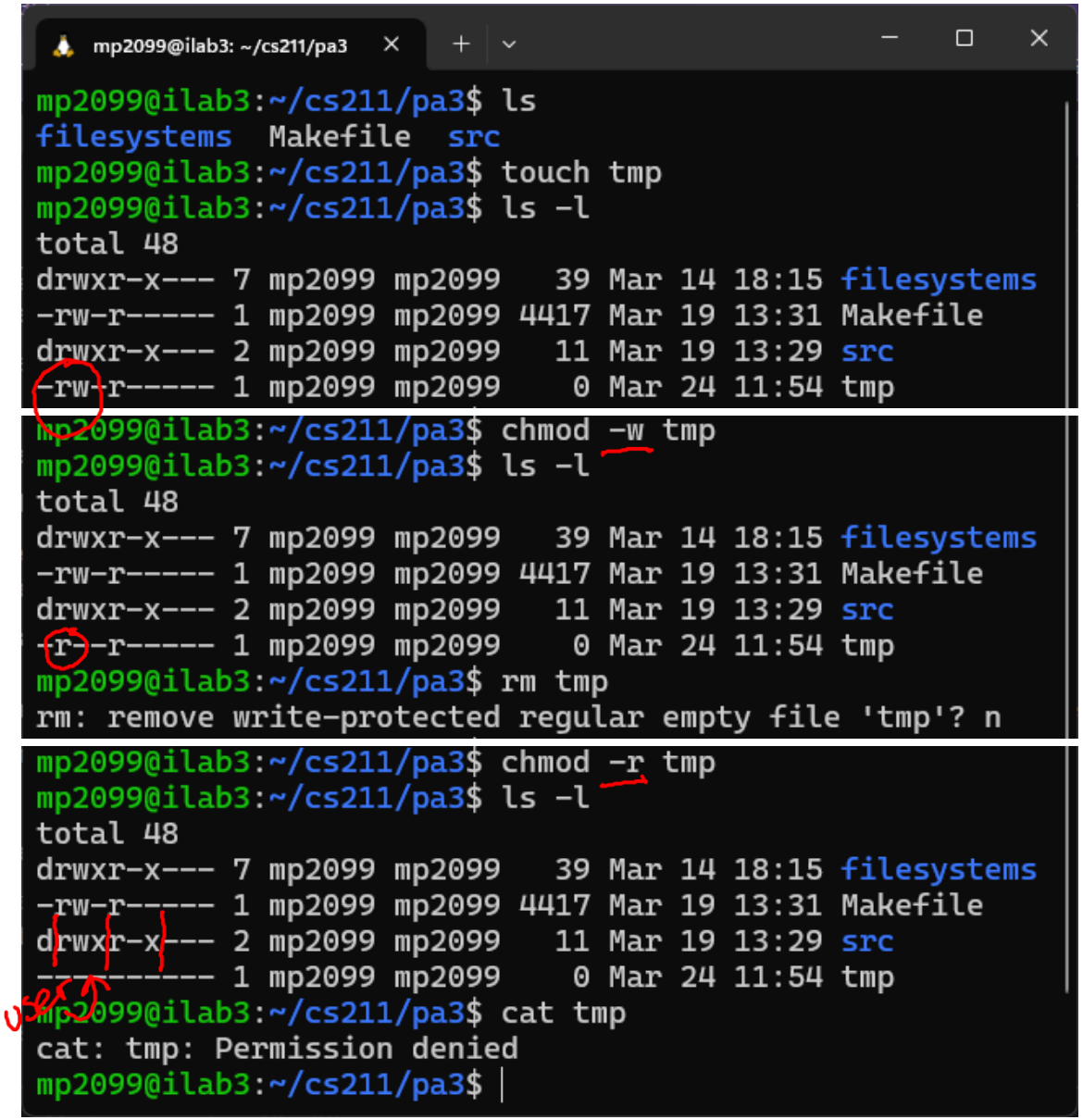
- Controls who can access a file




```
mp2099@ilab3:~/cs211/pa3$ ls -l
total 48
drwxr-x--- 7 mp2099 mp2099   39 Mar 14 18:15 filesystems
-rw-r----- 1 mp2099 mp2099 4417 Mar 19 13:31 Makefile
drwxr-x--- 2 mp2099 mp2099   11 Mar 19 13:29 src
---------- 1 mp2099 mp2099    0 Mar 24 11:54 tmp
mp2099@ilab3:~/cs211/pa3$ cat tmp
cat: tmp: Permission denied
mp2099@ilab3:~/cs211/pa3$
```

- Fundamental part of most operating systems

Filesystem Permissions on Linux



Creating a new file
(read || write by default)

chmod to read-only

chmod to no permissions at all

Filesystem Permissions for PA3

Think about whether you should allow:

- creating a file…
  - In a read-only directory?
  - In a r+w subdirectory of a read-only directory?
- removing a file…
  - That has read-only permissions?
  - Inside a read-only directory?
- changing permissions on a file/directory…
  - That has no access permissions?
  - What about its children?
- …

Agenda

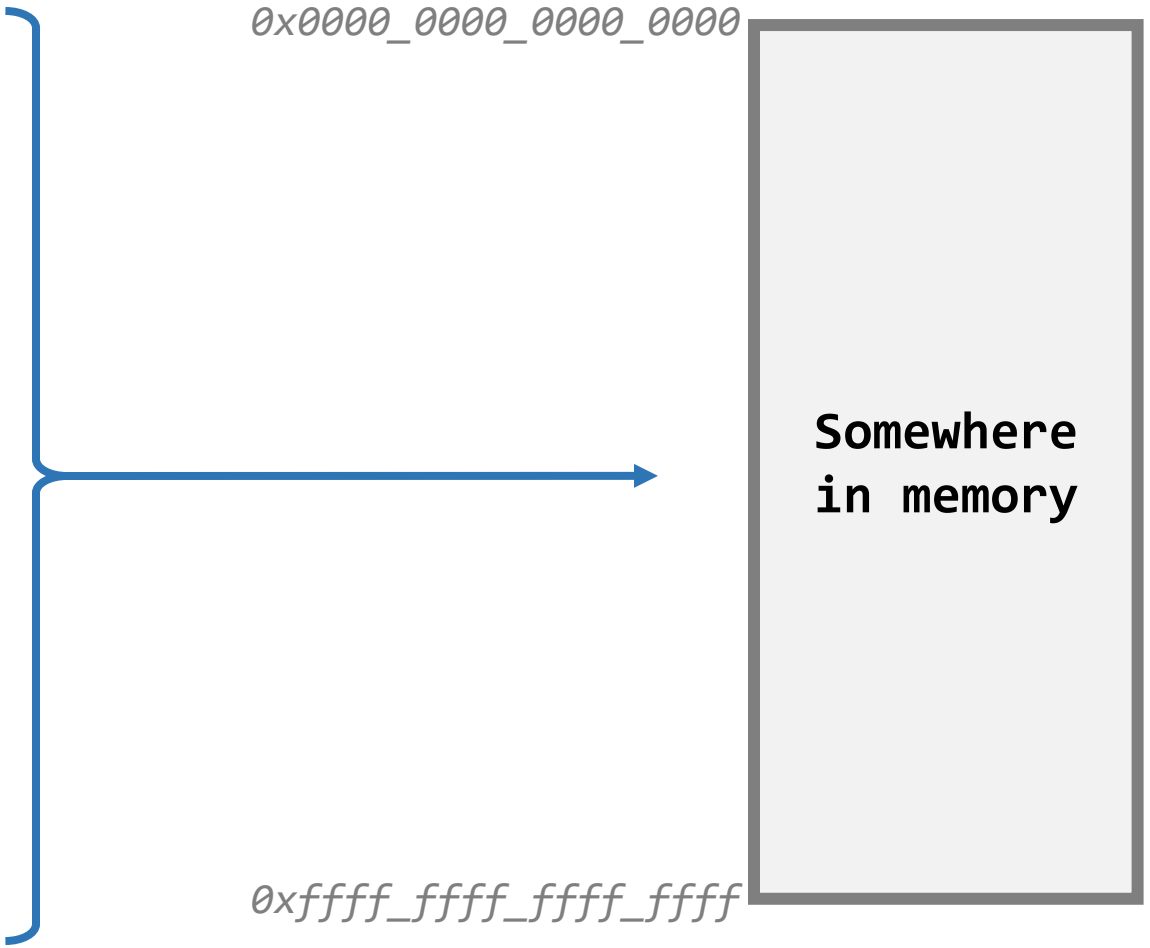
- **What's Next: Layers of Abstraction**

- Computer Organization Revisited

- The Representation of Code
  - Machine Language and ISAs
  - Choosing an ISA

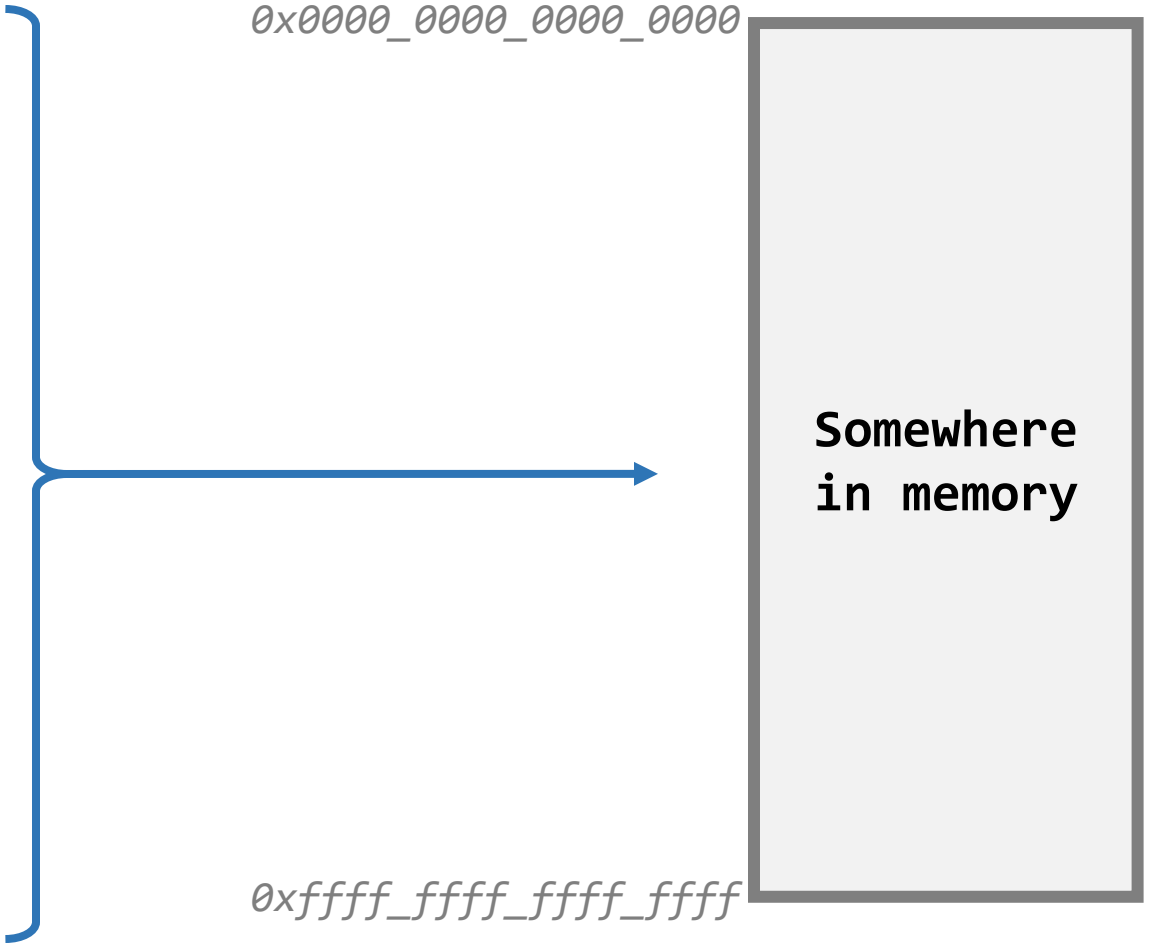
So Far: Representation of Data

- **Characters**
  `'a' '\n' '\0'`
- **Numbers**
  `0b1011` -6½
  `0xae.14` π^{-2}
- **Arrays / strings**
  `[0,1,2]` "me"
- …



Up Next: Representation of Code

```
*a += 3;

b = b ? 1 : 0;

while(c) {...}

func(a, b, c);

...
```



Layers of Abstraction for CS 211

Real-world Demands

Problem

Algorithm

Program

CS 415: Compilers

Runtime

CS 416: Operating Systems

System software

HW/SW Interface (ISA, ABI)

Microarchitecture

Logic gates

Circuits

Technology

Physics

Real-world Constraints

High-Level Language Program

```
int i;
for(i = 0; i < 10; i++)
    a[i] = b[i] * c[i];
```

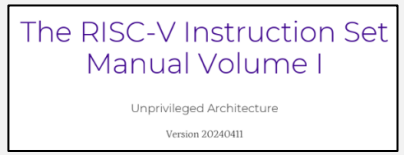
Assembly Language Program

```
loop:  ld   a4, 0(a5)
       ld   a3, 8(a5)
       addi a5, a5,   -8
       add  a4, a4,   a3
       sd   a4, 8(a5)
       bne  a0, a5,   loop
```

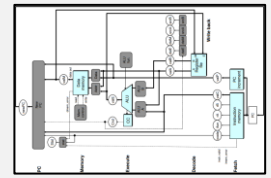
Machine Language Program

```
0000_0000_0000_0111_1011_0111_0000_0011
0000_0000_1000_0111_1011_0110_1000_0011
1111_1111_1000_0111_1000_0111_1001_0011
0000_0000_1101_0111_0000_0111_0011_0011
0000_0000_1110_0111_1011_0100_0010_0011
1111_1110_1111_0101_0001_0110_1110_0011
```

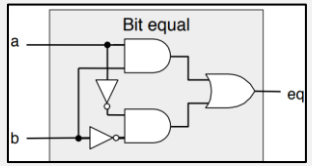
Instruction Set Architecture (ISA)



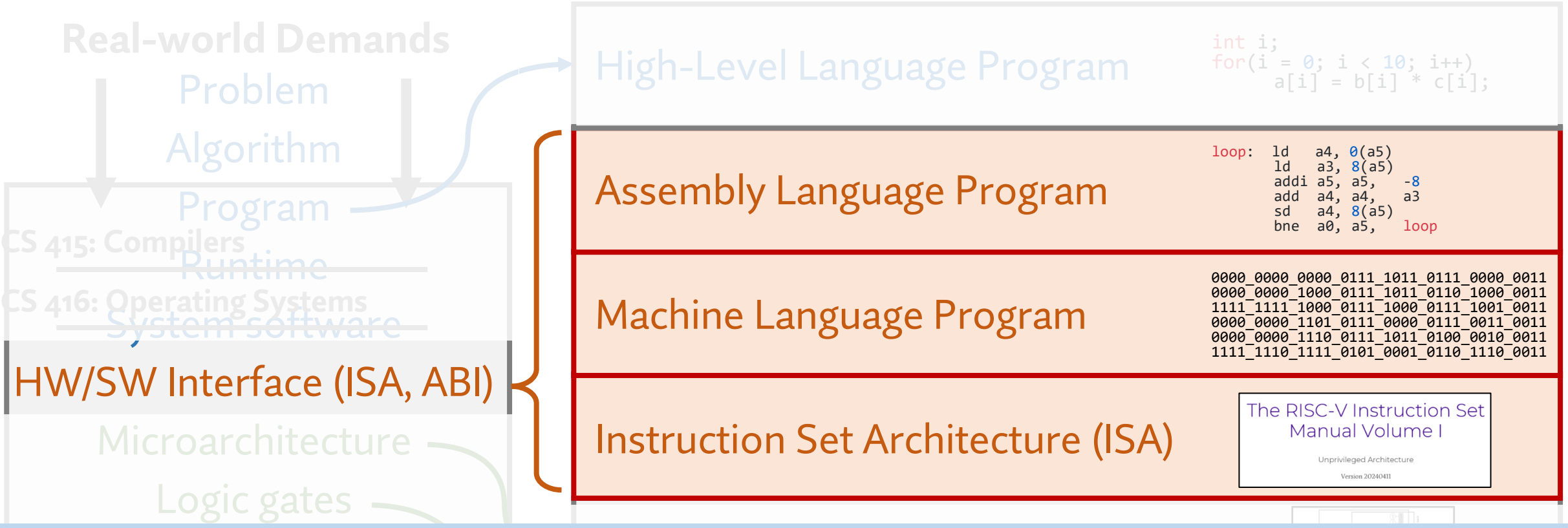
Simple RISC-V Processor



Logic Circuits



Layers of Abstraction for CS 211: What's Next



Next ~2-3 weeks:

The representation of code:

assembly and **machine** language programs

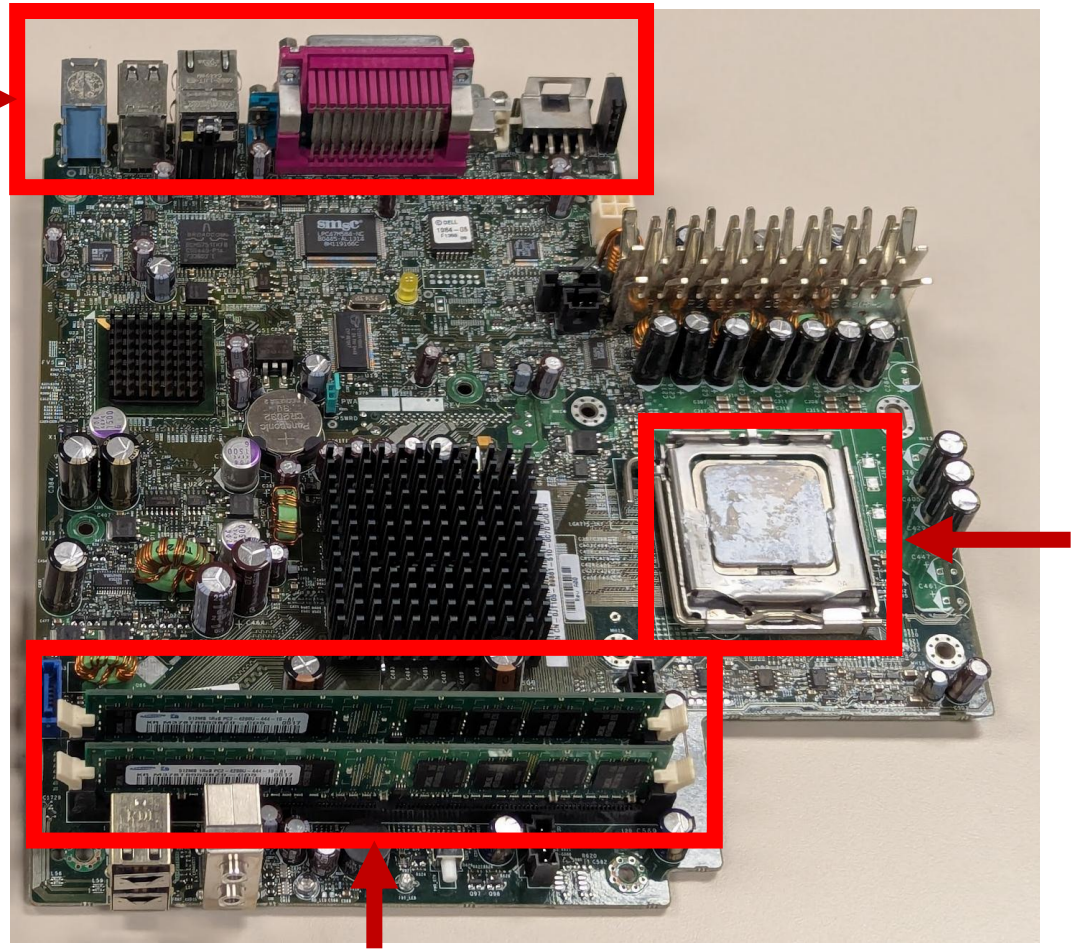
Agenda

- What's Next: Layers of Abstraction
- **Computer Organization Revisited**
- The Representation of Code
  - Machine Language and ISAs
  - Choosing an ISA

Recap: Computer Organization

Input/Output
(network, USB, etc.)

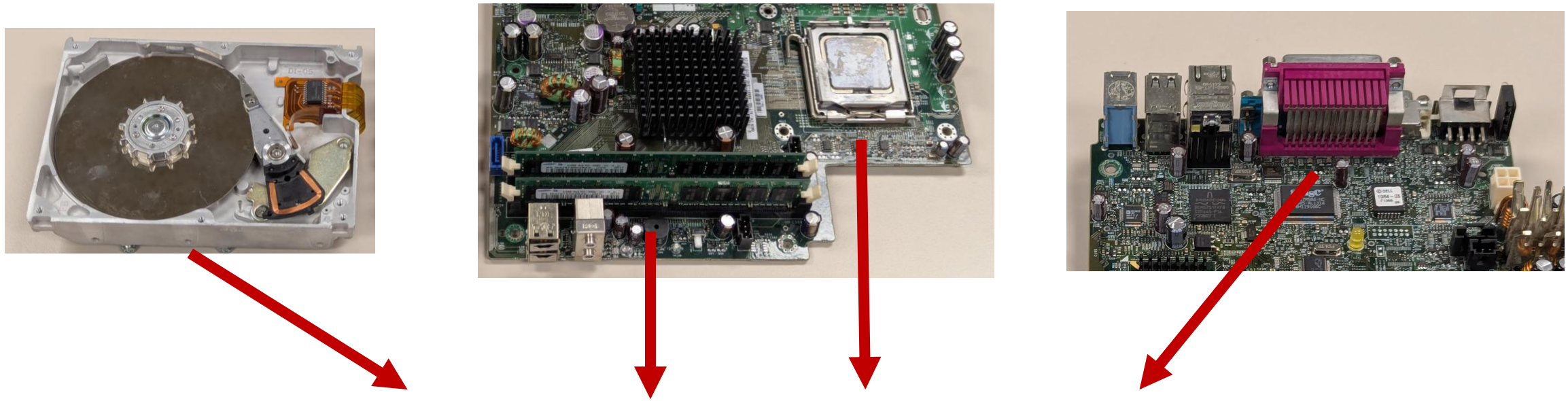
Storage
(big, slow, cheap memory)



Processor
(where code runs)

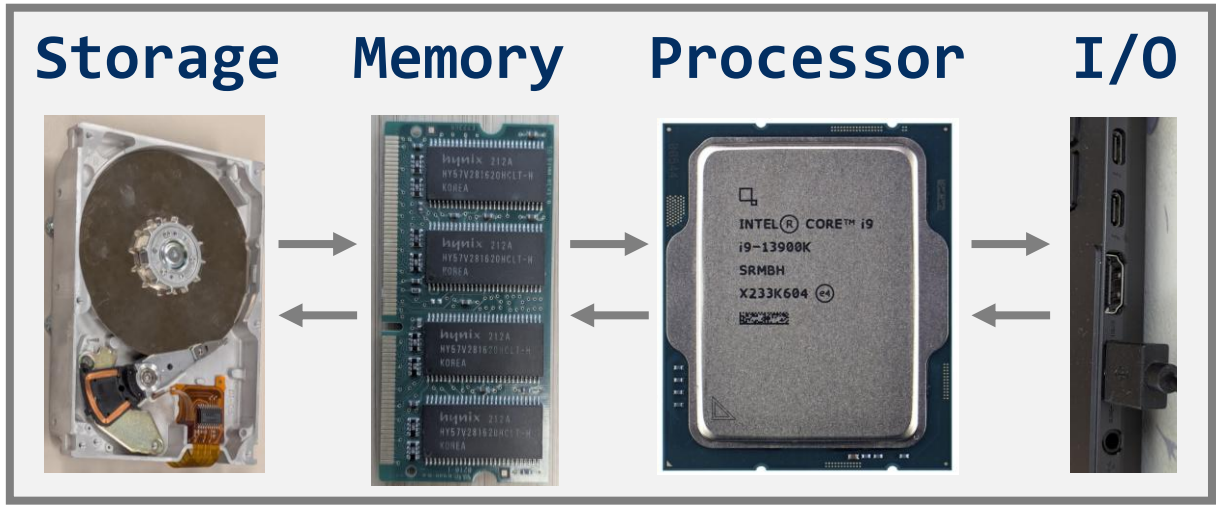
Main Memory
(small, fast, expensive)

Computer Organization: Mental Model

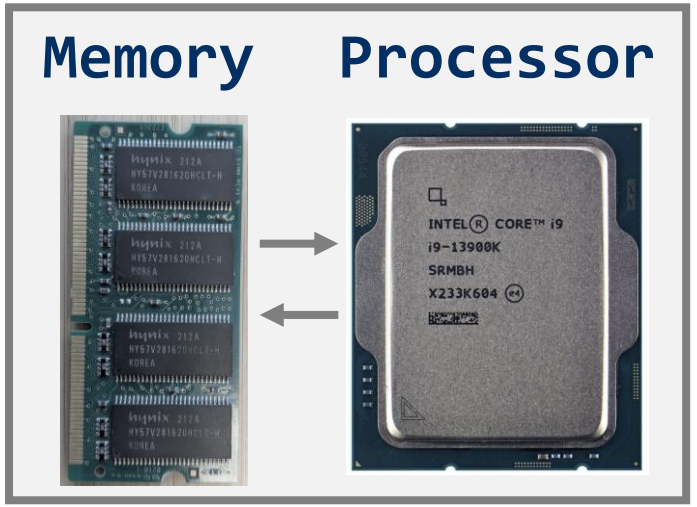


Computer Organization: Simplified

Computer



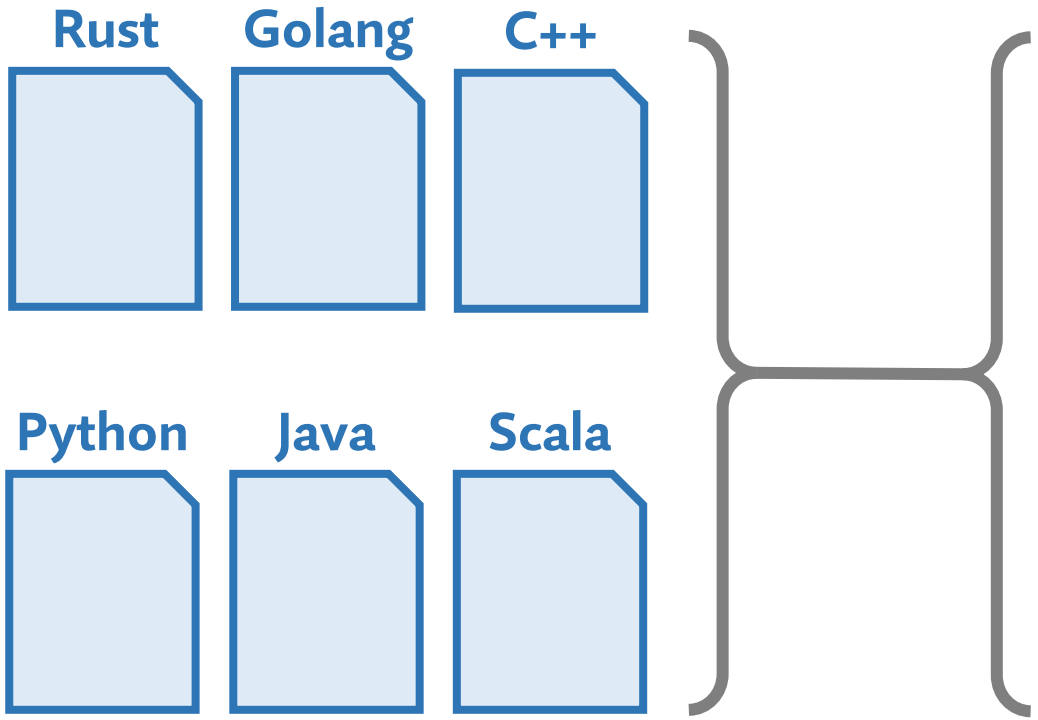
Simplified Mental Model



Abstract View: What Do We Want from a Computer?

- **Ideal:** runs **everything** quickly, efficiently, securely, reliably, etc.
  - Independent of your favorite programming language
- **Reality:** no single hardware design can achieve all of the above

Theme 1: Tradeoffs - pick the right tools



https://dlcdnwebimgs.asus.com/gain/d47ac467-7878-47b7-a29a-c9adfb80b9bd/

https://www.ebay.com/itm/154513065385?epid=1252327726&hash=item23f9b245a9:g:GoIAAOSwi7VcJULN

https://www.amazon.com/Google-Pixel-Unlocked-Smartphone-Ultrawide/dp/B09HJZPFDD

https://www.ultrabookreview.com/64501-asus-vivobook-pro-16x-oled-review/

Abstract View: What Do We Want from a Computer?

- **Ideal:** runs **everything** quickly, efficiently, securely, reliably, etc.
  - Independent of your favorite programming language
- **Reality:** no single hardware design can achieve all of the above

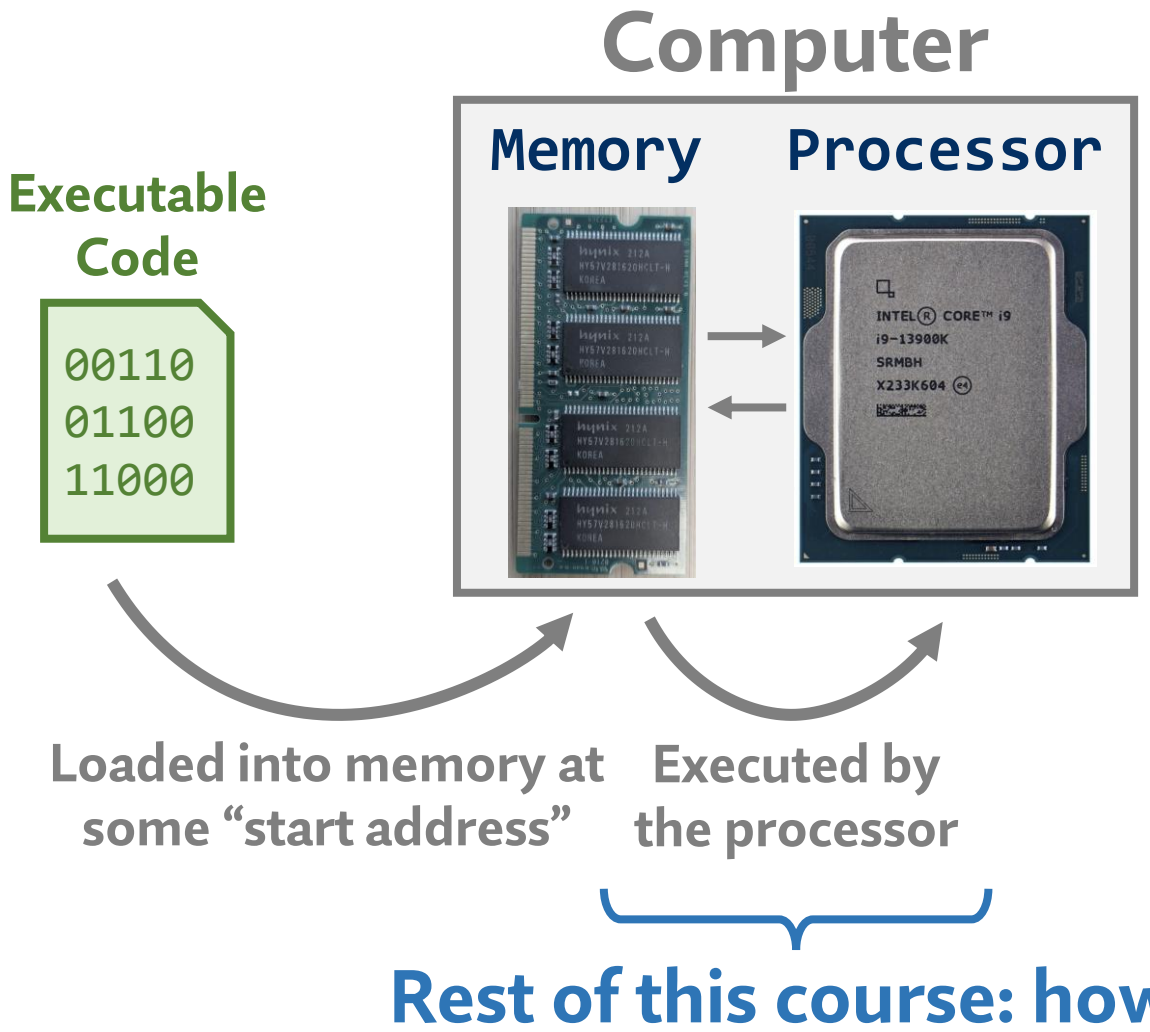
Theme 2: Separation of Concerns

Agenda

- What's Next: Layers of Abstraction
- Computer Organization Revisited
- **The Representation of Code**
  - Machine Language and ISAs
  - Choosing an ISA

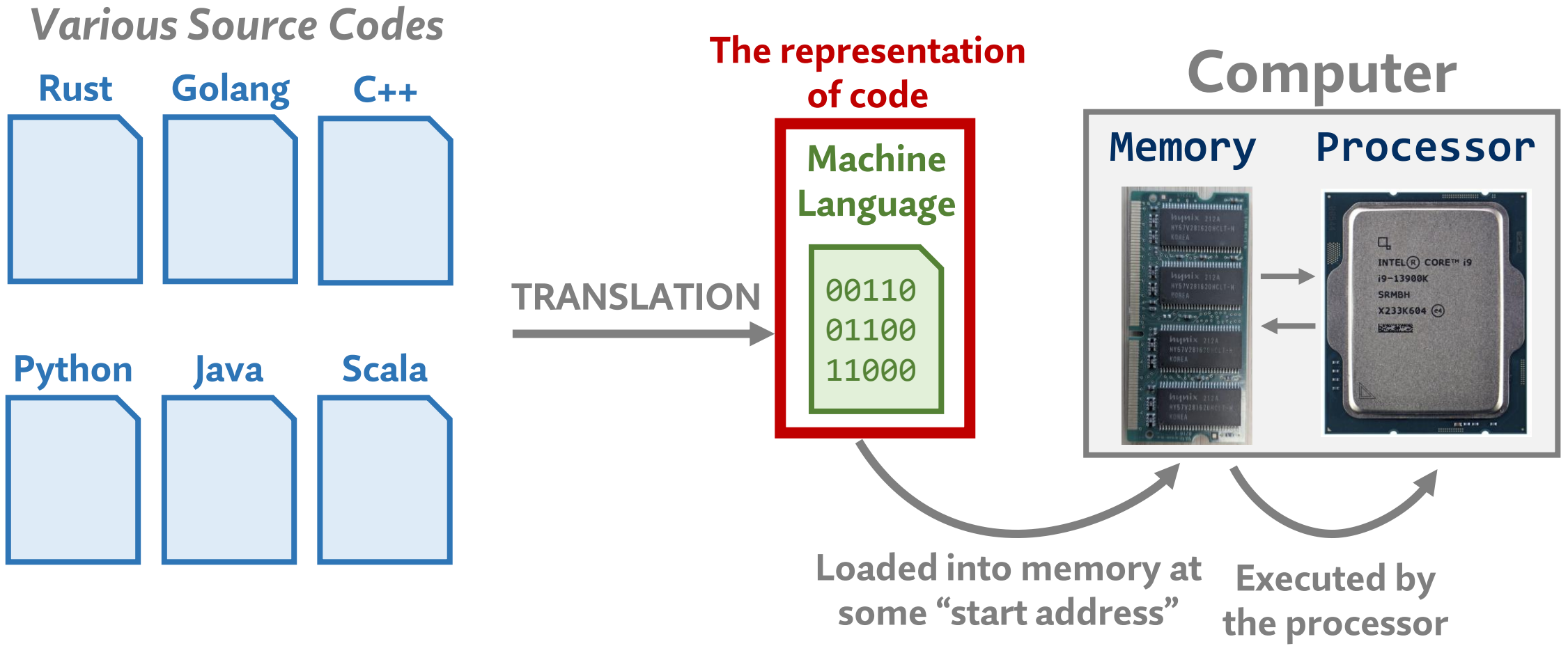
Running Code on the Processor

- Code lives somewhere **in memory**

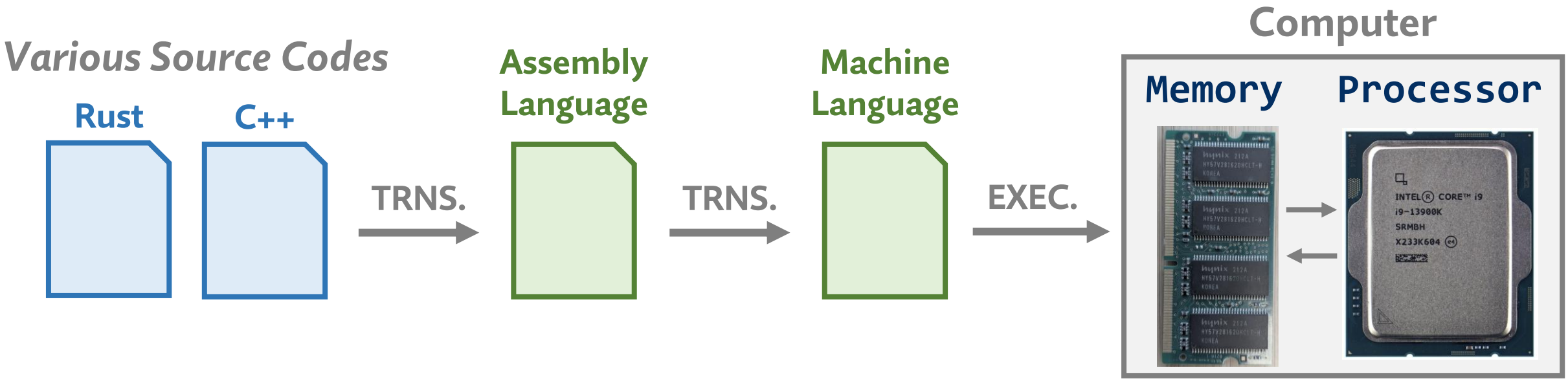


From Source Code to Execution

- The processor only understands **machine language code**
  - Anything that runs is **translated to machine language** (via compiler, interpreter, etc.)

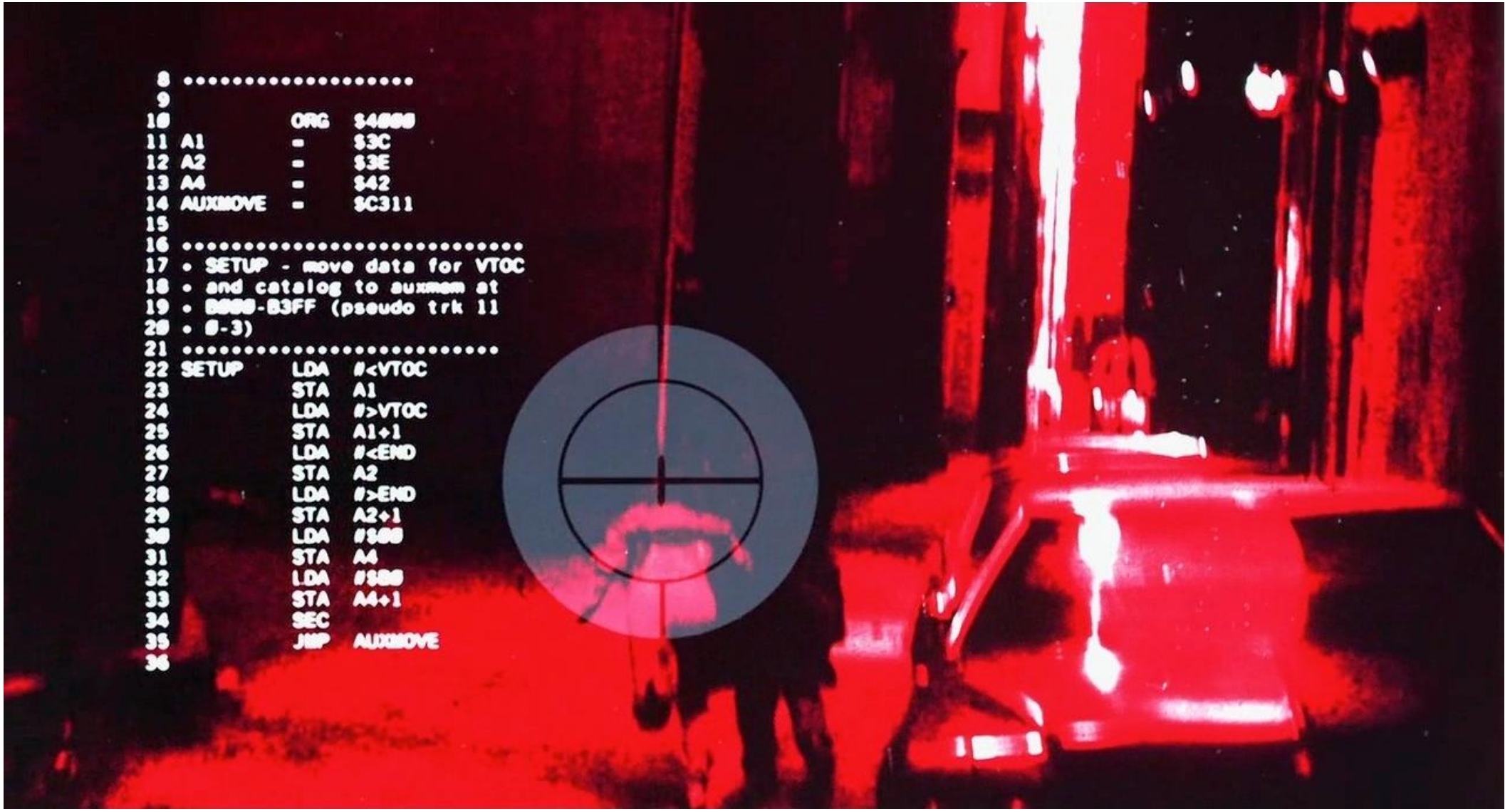


Assembly Language: Human-Readable Machine Code



- Machine language **exposes the raw capabilities** of a processor
  - E.g., **primitive operations** such as arithmetic, memory access, etc.
  - **Not** meant for humans to read/write
- Assembly language provides an **(almost) 1:1 abstraction** of machine code

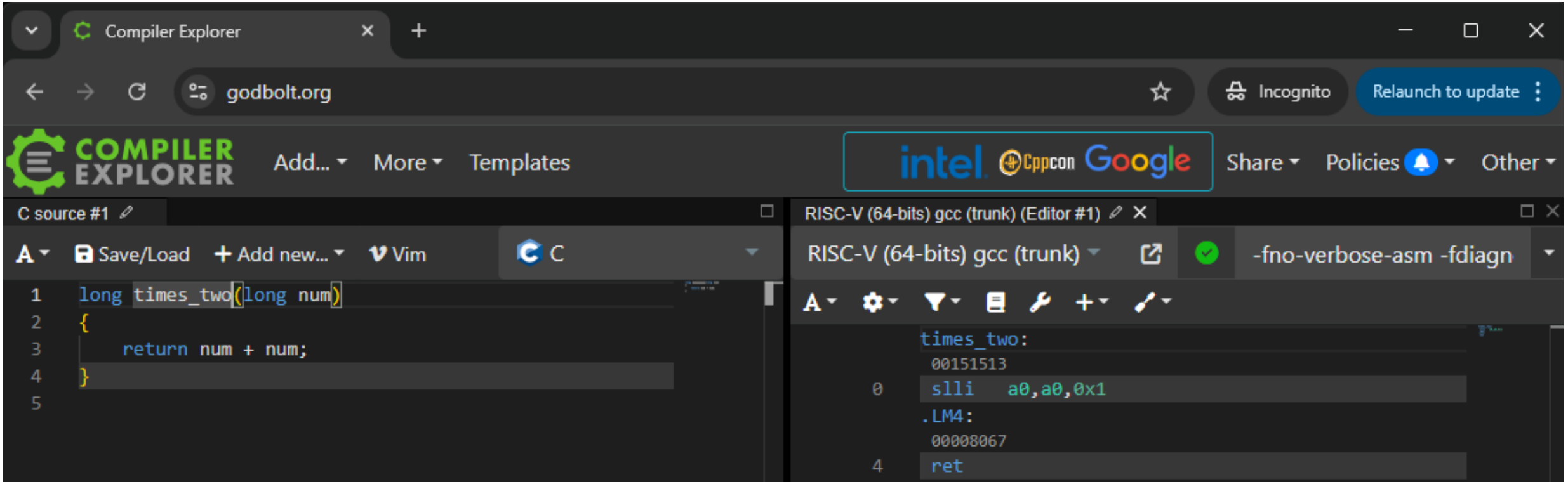
The Terminator T-800 Runs MOS 6502 Assembly



https://www.pagetable.com/docs/terminator/00-37-23.jpg

Example: Godbolt Compiler Explorer

https://godbolt.org/z/cqvME3xEn



C Code

```
long times_two(long num)
{
    return num + num;
}
```

Assembly Code

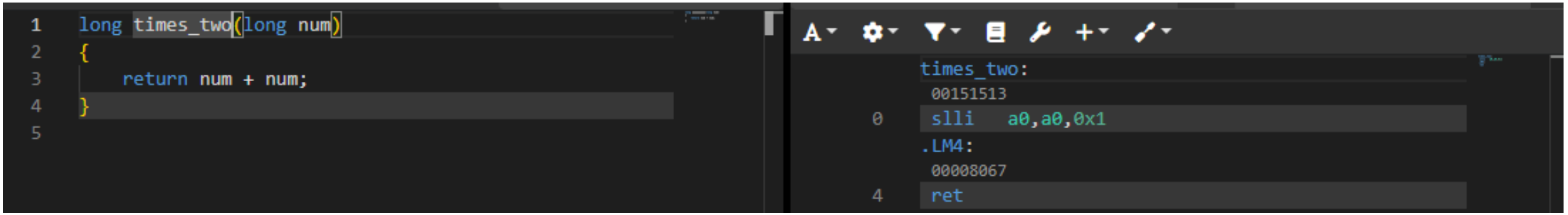
```
.times_two
    slli a0, a0, 0x1
    ret
```

Machine Code

```
0x00151513
0x00008067
```

Aside: Compiler Transformation

https://godbolt.org/z/cqvME3xEn



- Consider what I wrote in C vs. what processor will execute

Supplied C Code

```
return num + num;
```

"Decompiled" C Code

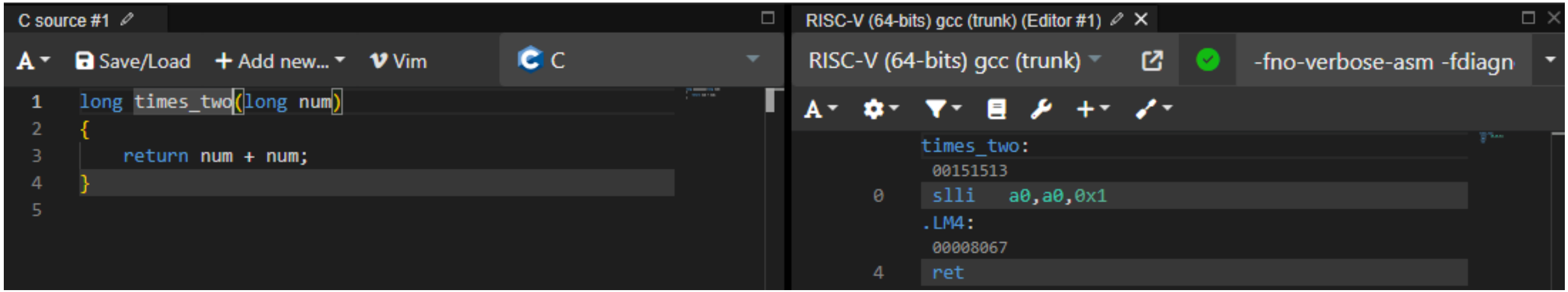
```
return num << 1;
```

↓

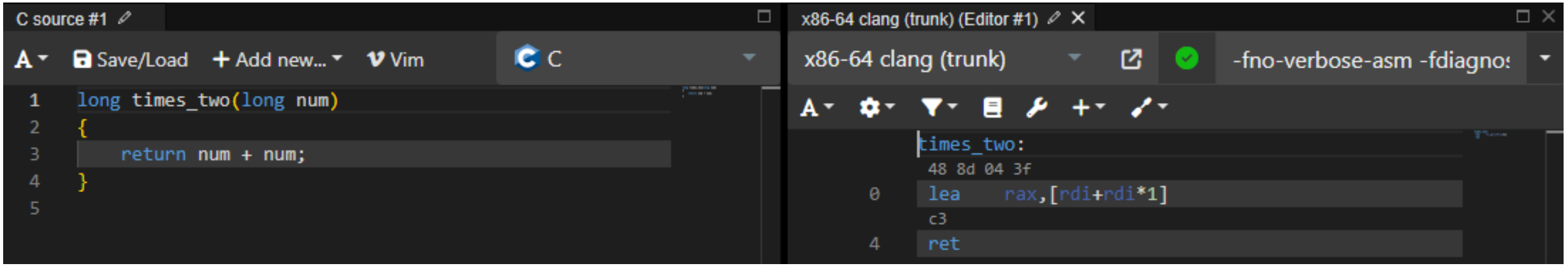
Technically, this is undefined behavior in C!

Example: RISC-V vs. x86-64

https://godbolt.org/z/cqvME3xEn



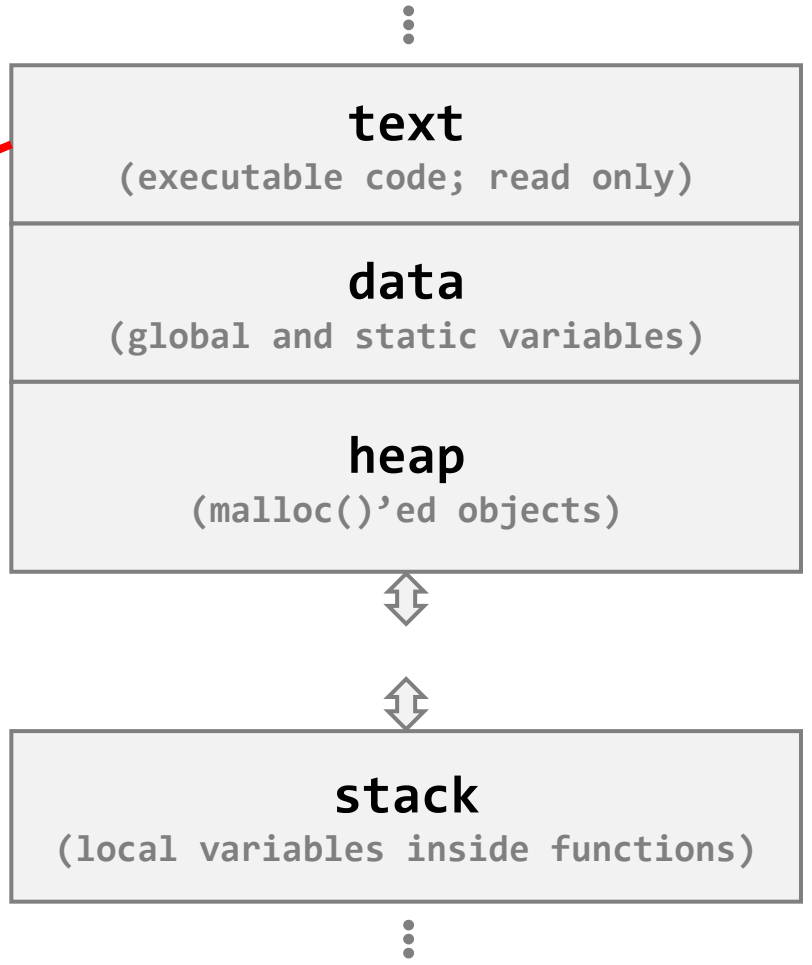
https://godbolt.org/z/bxaxzEaKo



Sections of an Executable File

The tool "**objdump**" allows us to inspect sections of a program

```
mp2099@ilab4: ~/cs211/expe
mp2099@ilab4:~/cs211/experiment$ /common/users/shared/cs211_s25_5678/toolchain
_glibc3/bin/riscv64-unknown-linux-gnu-objdump --section=.text hello.o -h

hello.o:     file format elf64-littleriscv

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
 10 .text         000000c0  0000000000010460  0000000000010460  00000460  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
mp2099@ilab4:~/cs211/experiment$
```



Example: Hello World on ilab

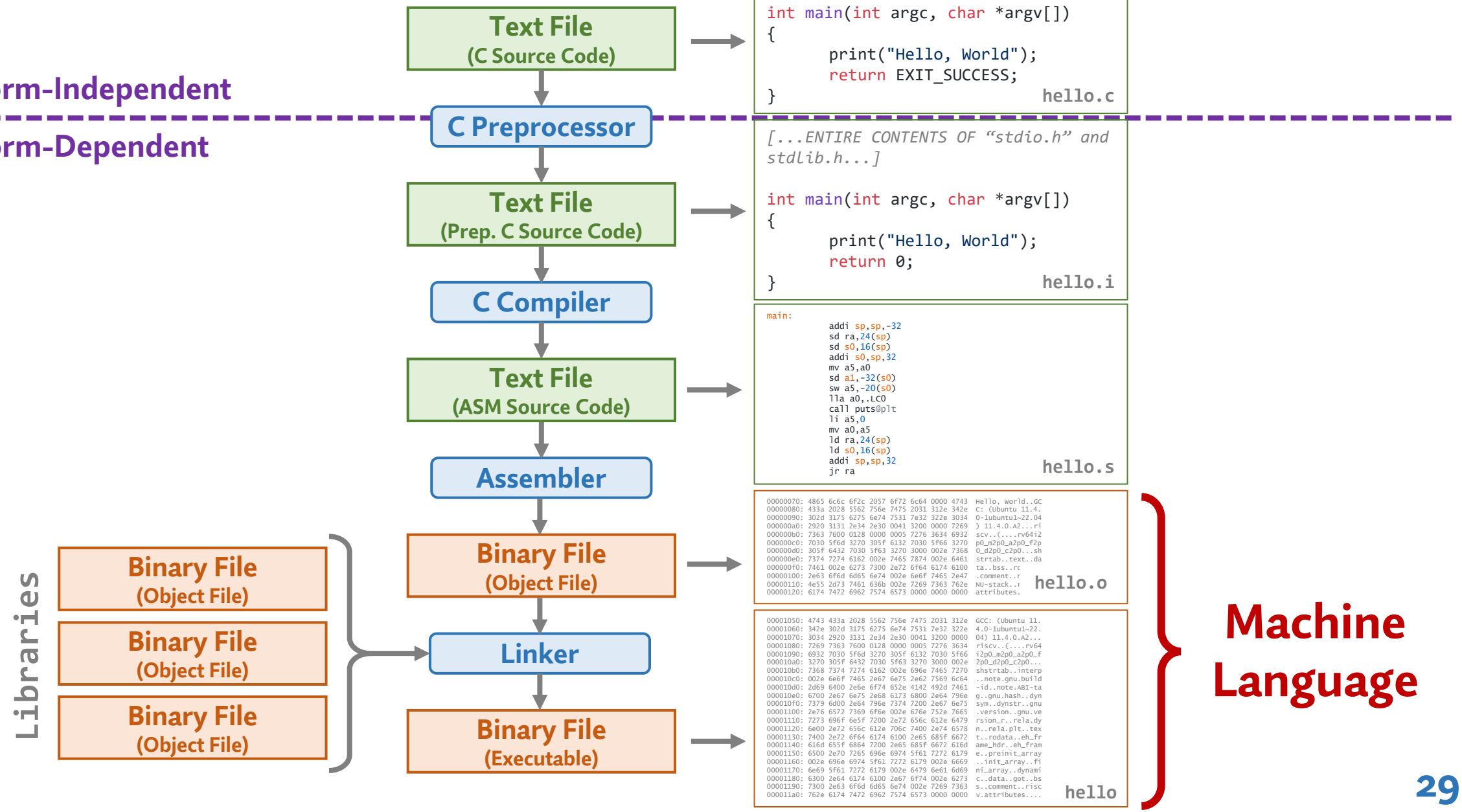
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      print("Hello, World");
      return EXIT_SUCCESS;
}
```

hello.c

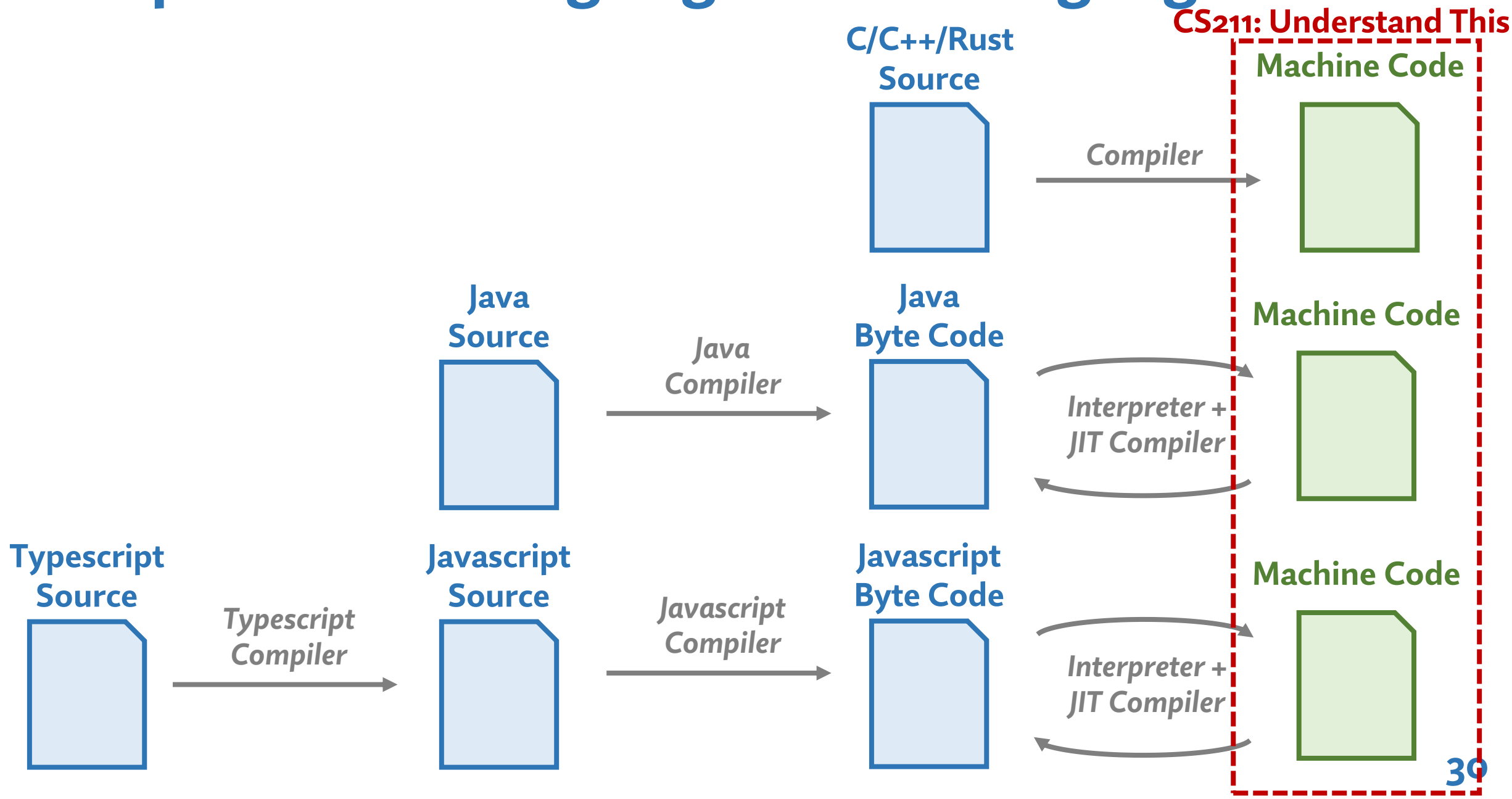
```
mp2099@ilab2: ~/cs211/expe
mp2099@ilab2:~/cs211/experiment$ /common/users/shared/cs211_s25_5678/toolchain_glibc3/bin/riscv64-unk
nown-linux-gnu-gcc hello.c -o hello.o -mabi=lp64 -march=rv64i -Os -fno-inline -fomit-frame-pointer
mp2099@ilab2:~/cs211/experiment$ /common/users/shared/cs211_s25_5678/toolchain_glibc3/bin/riscv64-unk
nown-linux-gnu-objdump -L -S --section=.text hello.o

hello.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000010460 <main>:
   10460:       00010537                lui     a0,0x10
   10464:       ff010113                addi    sp,sp,-16
   10468:       52850513                addi    a0,a0,1320 # 10528 <_IO_stdin_used+0x8>
   1046c:       00113423                sd      ra,8(sp)
   10470:       fe1ff0ef                jal     10450 <puts@plt>
   10474:       00813083                ld      ra,8(sp)
   10478:       00000513                li      a0,0
   1047c:       01010113                addi    sp,sp,16
   10480:       00008067                ret
```

"disassembly"

Recap: C Compilation



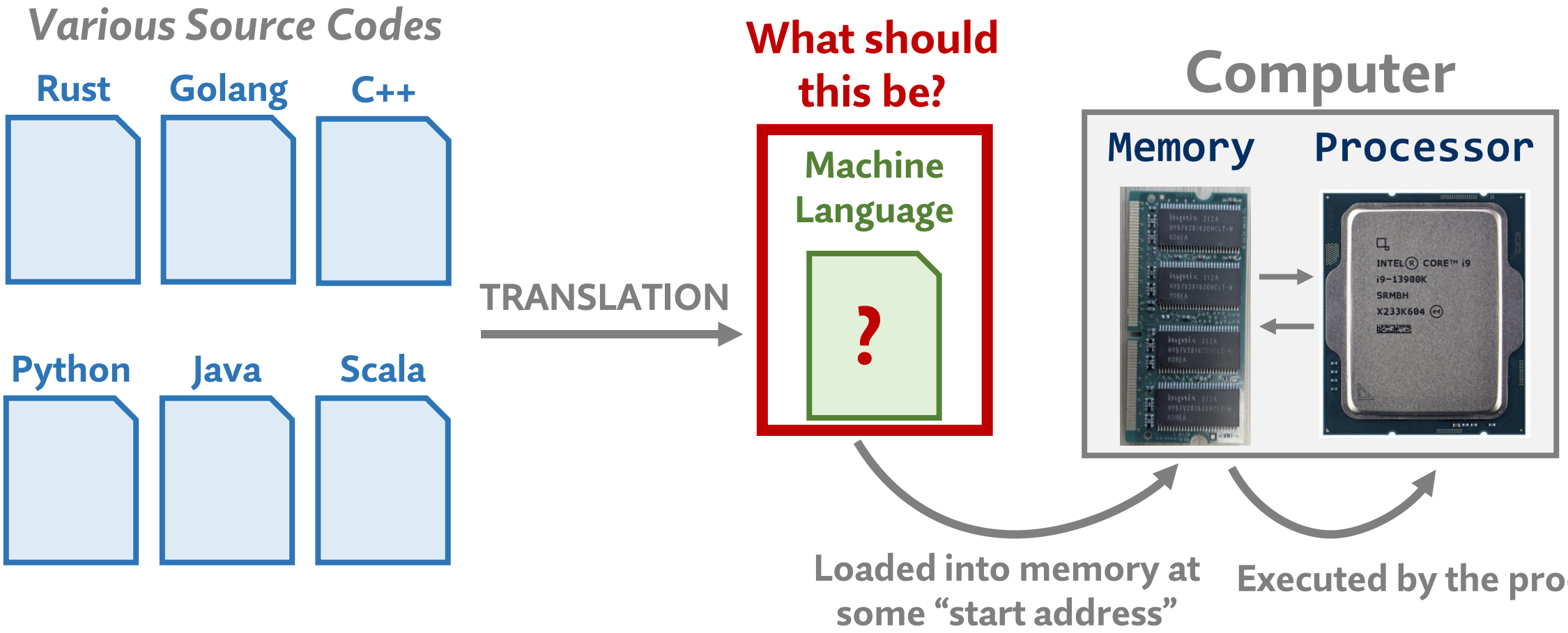
Example: Translating High-Level Languages



Agenda

- What's Next: Layers of Abstraction
- Computer Organization Revisited
- The Representation of Code
  - **Machine Language and ISAs**
  - Choosing an ISA

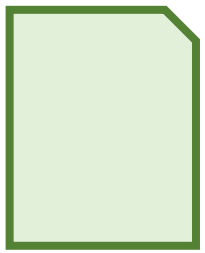
Designing a Machine Language



Instruction Set Architecture: The HW-SW Interface

- **Instruction Set Architecture (ISA)**: the contract between HW and SW
  - Precisely specifies the machine language
  - Abstracts away *how* the hardware implements the machine language

Software
(Machine Language)



731-page manual

The RISC-V Instruction Set Manual
Volume I

Unprivileged Architecture

Version 20240411

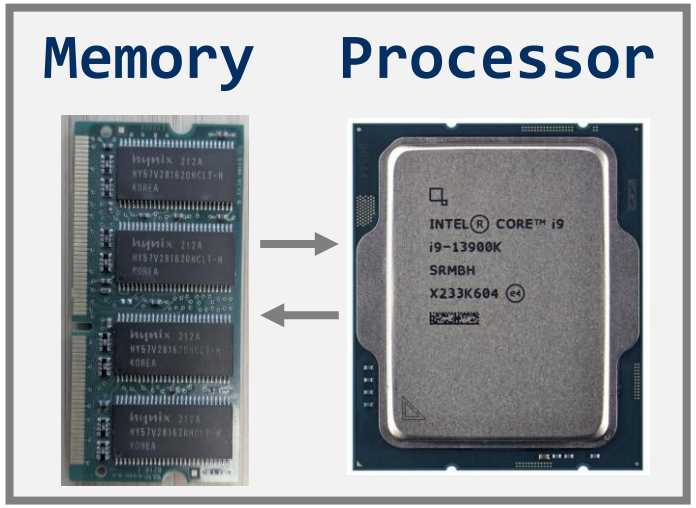
2606-page manual

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 2 (2A, 2B, 2C, & 2D):
Instruction Set Reference, A-Z

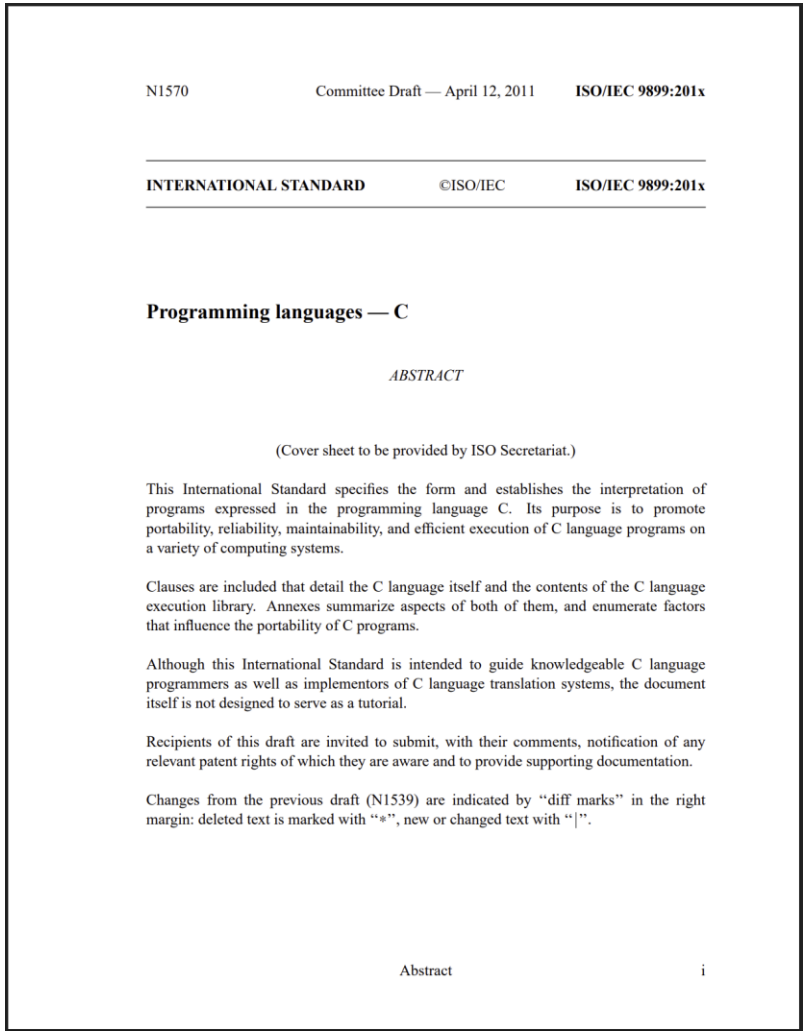
ISA

Hardware
(Microarchitecture)



Language Specifications

C Programming Language Standard



N1570 Committee Draft — April 12, 2011 **ISO/IEC 9899:201x**

INTERNATIONAL STANDARD ©ISO/IEC **ISO/IEC 9899:201x**

Programming languages — C

ABSTRACT

(Cover sheet to be provided by ISO Secretariat.)

This International Standard specifies the form and establishes the interpretation of programs expressed in the programming language C. Its purpose is to promote portability, reliability, maintainability, and efficient execution of C language programs on a variety of computing systems.

Clauses are included that detail the C language itself and the contents of the C language execution library. Annexes summarize aspects of both of them, and enumerate factors that influence the portability of C programs.

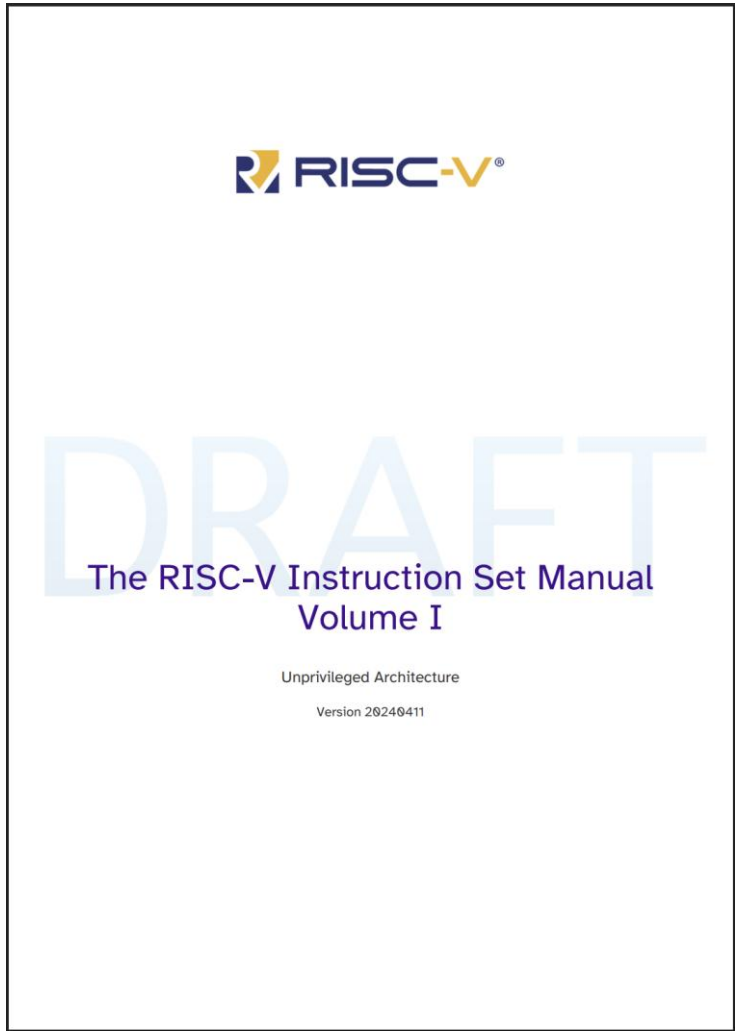
Although this International Standard is intended to guide knowledgeable C language programmers as well as implementors of C language translation systems, the document itself is not designed to serve as a tutorial.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Changes from the previous draft (N1539) are indicated by "diff marks" in the right margin: deleted text is marked with "*", new or changed text with "|".

Abstract i

RISC-V ISA Manual



RISC-V®

DRAFT

The RISC-V Instruction Set Manual
Volume I

Unprivileged Architecture

Version 20240411

What Does the ISA Specify?

- Describes:
  - All operations a given processor ***can*** perform
  - The ***binary representations*** of each operation
- Abstracts away ***how*** the processor does any of it

- Example operations:
  - **Scalar arithmetic:** Add, subtract, multiply, divide, etc.
  - **Vector arithmetic:** Dot product, element shuffling
  - **Memory access:** load/store to a given memory address
  - **Function calls:** Call and return
  - **Security primitives:** Hash functions, polynomial computations
  - **Configuration:** configure the hardware (e.g., enable/disable features)
  - … and more

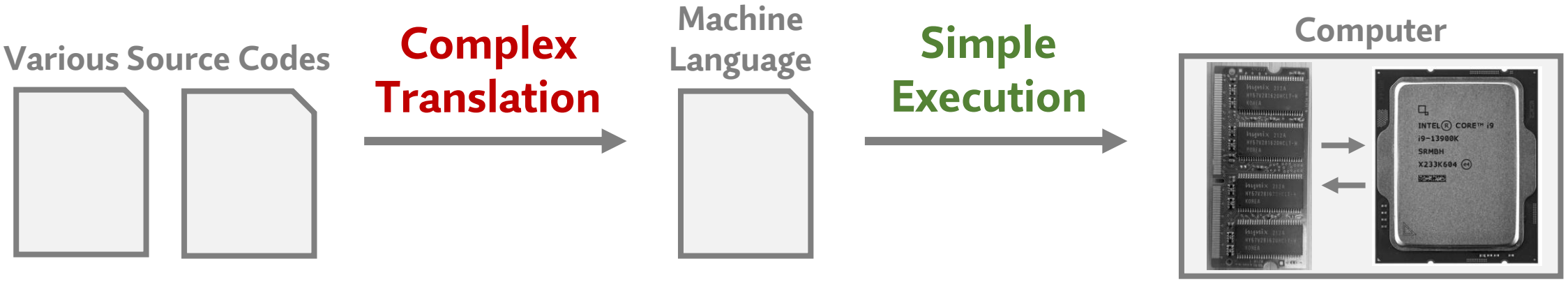
Agenda

- What's Next: Layers of Abstraction
- Computer Organization Revisited
- The Representation of Code
  - Machine Language and ISAs
  - **Choosing an ISA**

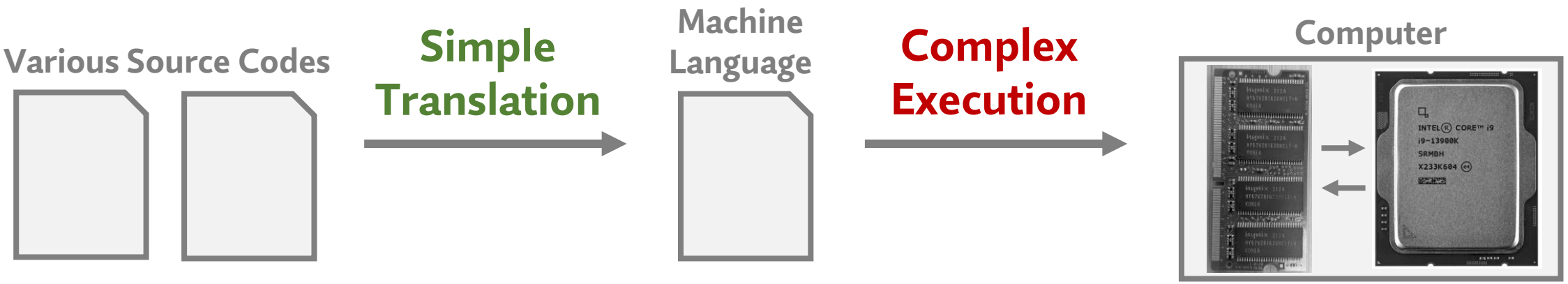
Choice of ISA Impacts HW/SW Design Complexity

- **There's no free lunch:** you can move complexity around but not remove it

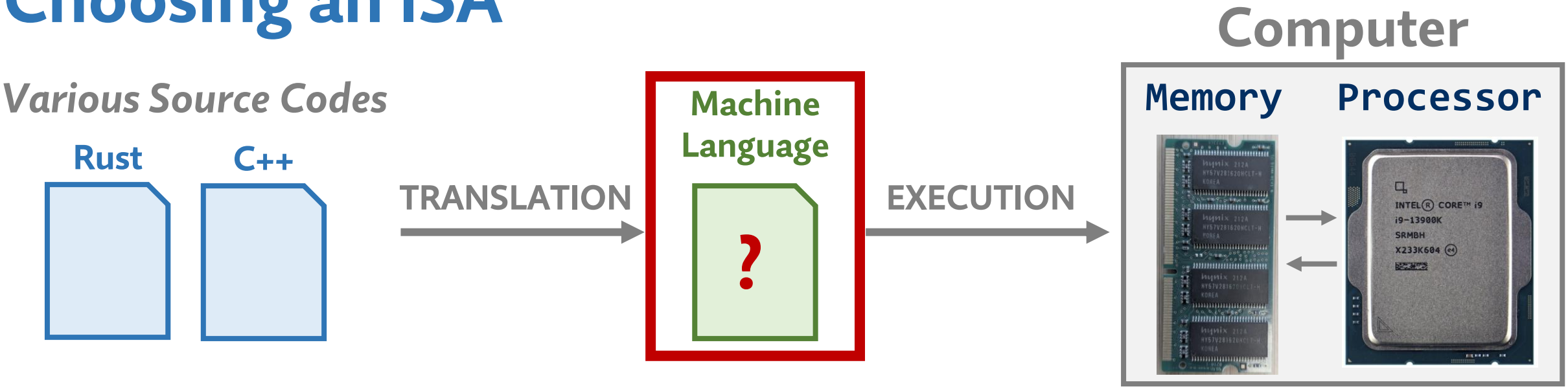
Simple Machine Language



Complex Machine Language



Choosing an ISA



- **No single answer:** need to know your end-to-end system design goals
- **Complex tradeoffs** between different capabilities
  - Design complexity (software & hardware)
  - Performance optimizations
  - Debuggability
  - Ability to patch problems in the field
  - Security considerations
  - …

Choosing an ISA: High Level Machine Language

- **Option 1:** high-level machine language (e.g., C, Java, Fortran, etc.)

tl;dr: it's been tried

Retrospective on High-Level Language Computer Architecture

David R. Ditzel

Bell Laboratories
Computing Science Research Center
Murray Hill, New Jersey

David A. Patterson

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California

Introduction

High-level language computers (HLLC) have attracted interest in the architectural and programming community during the last 15 years; proposals have been made for machines directed towards the execution of various languages such as ALGOL,[1,2] APL,[3,4,5] BASIC,[6,7] COBOL,[8,9] FORTRAN,[10,11] LISP,[12,13] PASCAL,[14] PL/I,[15,16,17]

- Esoteric: Aesthetics or no stated advantages.

An almost universal justification for high-level language comput-

https://courses.cs.washington.edu/courses/cse548/05wi/files/Ditzel-Retrospective-on-HLL-Computer-Architecture.pdf

High-Level Machine Languages

- **Older (1965 – 1980):** Fortran, Lisp, Pascal, Algol, and many others
- **ARMv5 (1998):** "Java mode" side-by-side with ARM mode

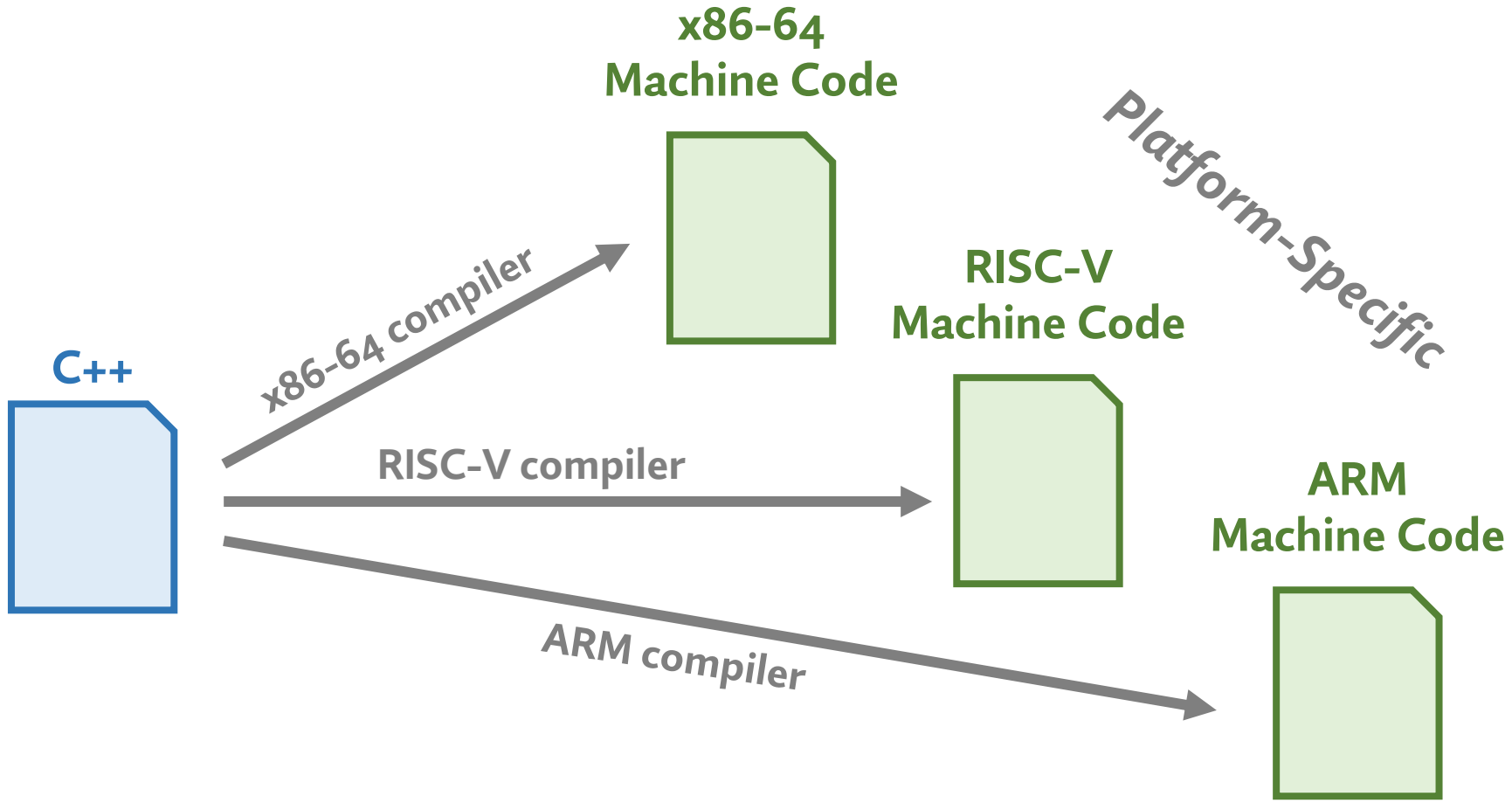
- Vision:
  - **Simple, cheap software** (e.g., simple compilers, software debugging tools)

- Reality:
  - **Moves complexity** to the hardware: not simpler or cheaper!
  - **No real difference** (or advantage) to application-level programmers
  - Often gives software **less control** over the processor and memory

https://courses.cs.washington.edu/courses/cse548/05wi/files/Ditzel-Retrospective-on-HLL-Computer-Architecture.pdf

Choosing an ISA: Low Level Machine Language

- Break things down to **small primitive operations** (i.e., "**instructions**")
- Let compilers **translate** from HLLs to instructions



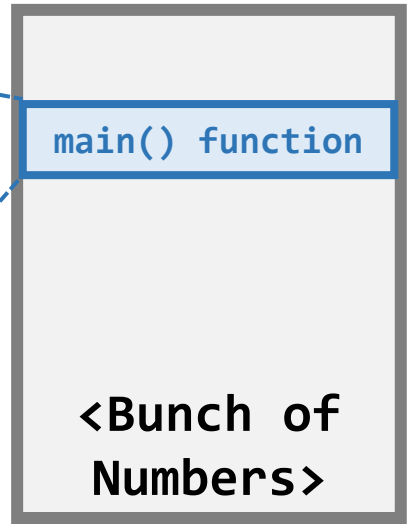
Modern ISAs

- Today's systems almost all use some variant of low-level instructions
  - All data and instructions are represented as **numbers in memory**
  - The processor **executes instructions** out of memory one-by-one
  - Every instruction updates the **numbers in memory**

Program
(instructions for number manipulation)

```
main:
        addi sp,sp,-32
        sd ra,24(sp)
        sd s0,16(sp)
        addi s0,sp,32
        mv a5,a0
        sd a1,-32(s0)
        sw a5,-20(s0)
        lla a0,.LC0
        call puts@plt
        li a5,0
        mv a0,a5
        ld ra,24(sp)
        ld s0,16(sp)
        addi sp,sp,32
        jr ra
```

Memory
(numbers in various representations)



Execute instructions one-by-one

Update numbers in memory

Processor
(executes instructions)

Example Instruction: RISC-V NOP

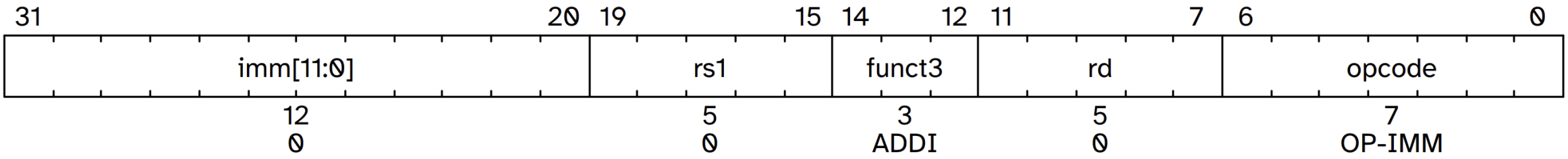
- A "No Operation (NOP)" instruction does **nothing**
  - Essential for debugging, performance, alignment, etc.

Functional Descripion of a NOP Instruction

The NOP instruction does not change any architecturally visible state, except for advancing the `pc` and incrementing any applicable performance counters. NOP is encoded as ADDI *x0*, *x0*, *0*.

Binary Representation of a NOP Instruction (32 Bits)

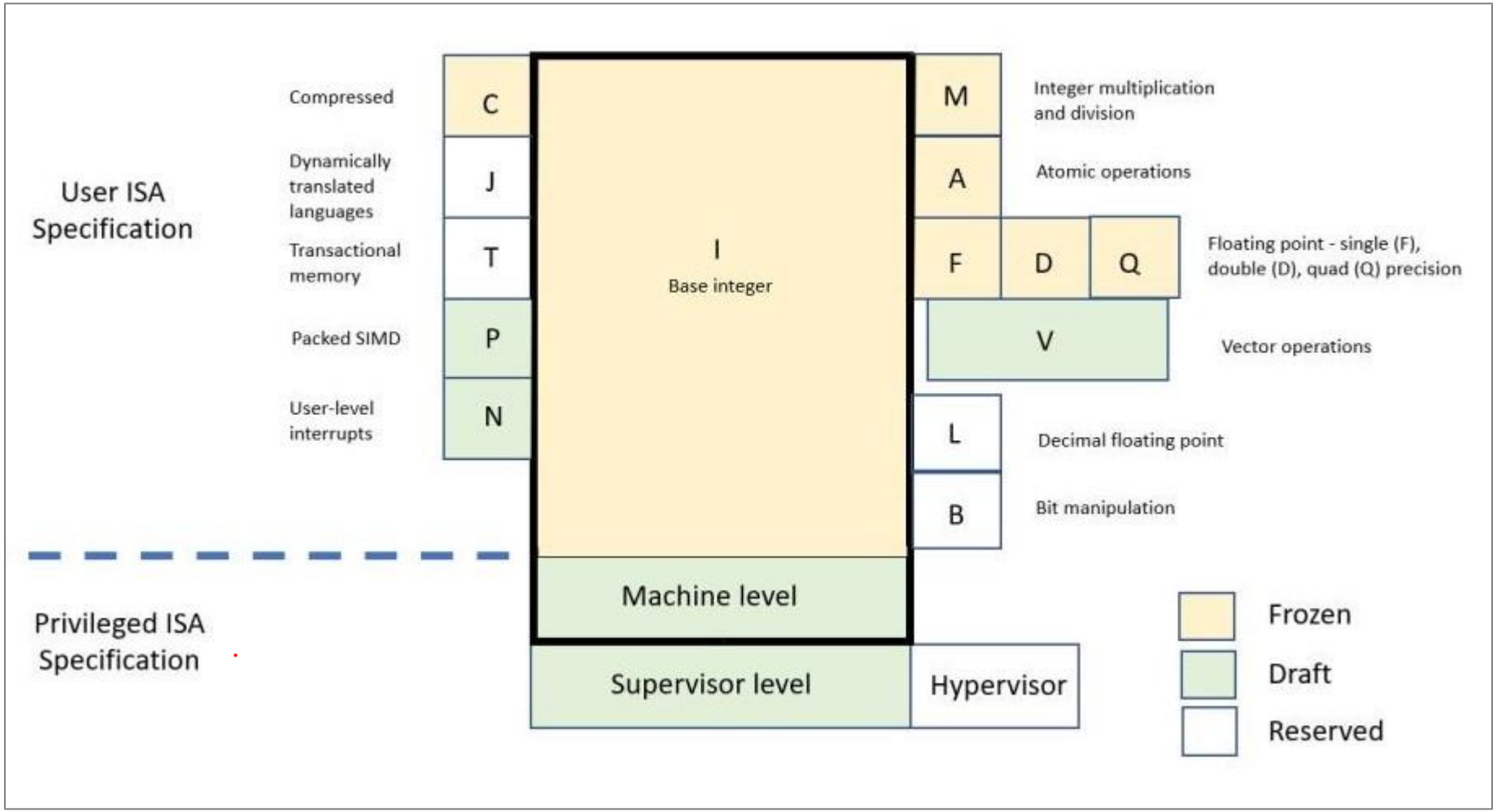
2.4.3. NOP Instruction



"The RISC-V Instruction Set Manual Volume I," Version 20240411.

RISC-V

- Good middle ground for this course: simple, yet realistic



https://www.microcontrollertips.com/wp-content/uploads/2020/11/RISC-V-ISA-block-diagram.jpg

CS 211: Intro to Computer Architecture

9.1: The Hardware-Software Interface

Minesh Patel

Spring 2025 – Tuesday 25 March