CSE 390 B Spring 2021

Mid-Quarter Retrospective & Reflection

CSE 390B Midterm Debrief, Time Management Check-in, Nand2Tetris Reflection/Look Ahead

Agenda

- Midterm Recap
- Revisiting Time Management
- Where we've been, where we're at, & where we're going
- Nand2Tetris Reflection/Look Ahead

Midterm Overview

A Few Midterm Notes

- Overall awesome work!
- For almost all questions everyone seemed to understand the general idea/start in the right direction
- Figuring out the details was where most of the mistakes were made
  - Likely this is a product of the time pressure?

A Few More Midterm Notes

- If you think a problem was graded unfairly or wrong, submit a regrade request in Gradescope!
  - Don't be too scared to do so; this is a great learning opportunity both for you and me!

- If you are unhappy with your score, you will have a chance to get points back w/midterm corrections as part of project 6

Agenda

- Midterm Recap
- **Revisiting Time Management**
- Where we've been, where we're at, & where we're going
- Nand2Tetris Reflection/Look Ahead

Time Management Revisited

Identify which **time management interval** (quarter, week, day, length of project) you feel you've improved your time management skills in the most and why? Which area(s) do you still want to improve in?
Try to be as specific as possible.

What are specific time management strategies that you think would be helpful but have yet to regularly practice?

What are some ways for you to shift from thinking/knowing a practice could be helpful to actually implementing that practice?

- Quarterly Calendar
- Weekly Time Commitments schedule
- 24-Hour Time Audit
- Project planning document

Time Management Revisited

Identify which **time management interval** (quarter, week, day, length of project) you feel you've improved your time management skills in the most and why? Which area(s) do you still want to improve in?
Try to be as specific as possible.

What are specific time management strategies that you think would be helpful but have yet to regularly practice?

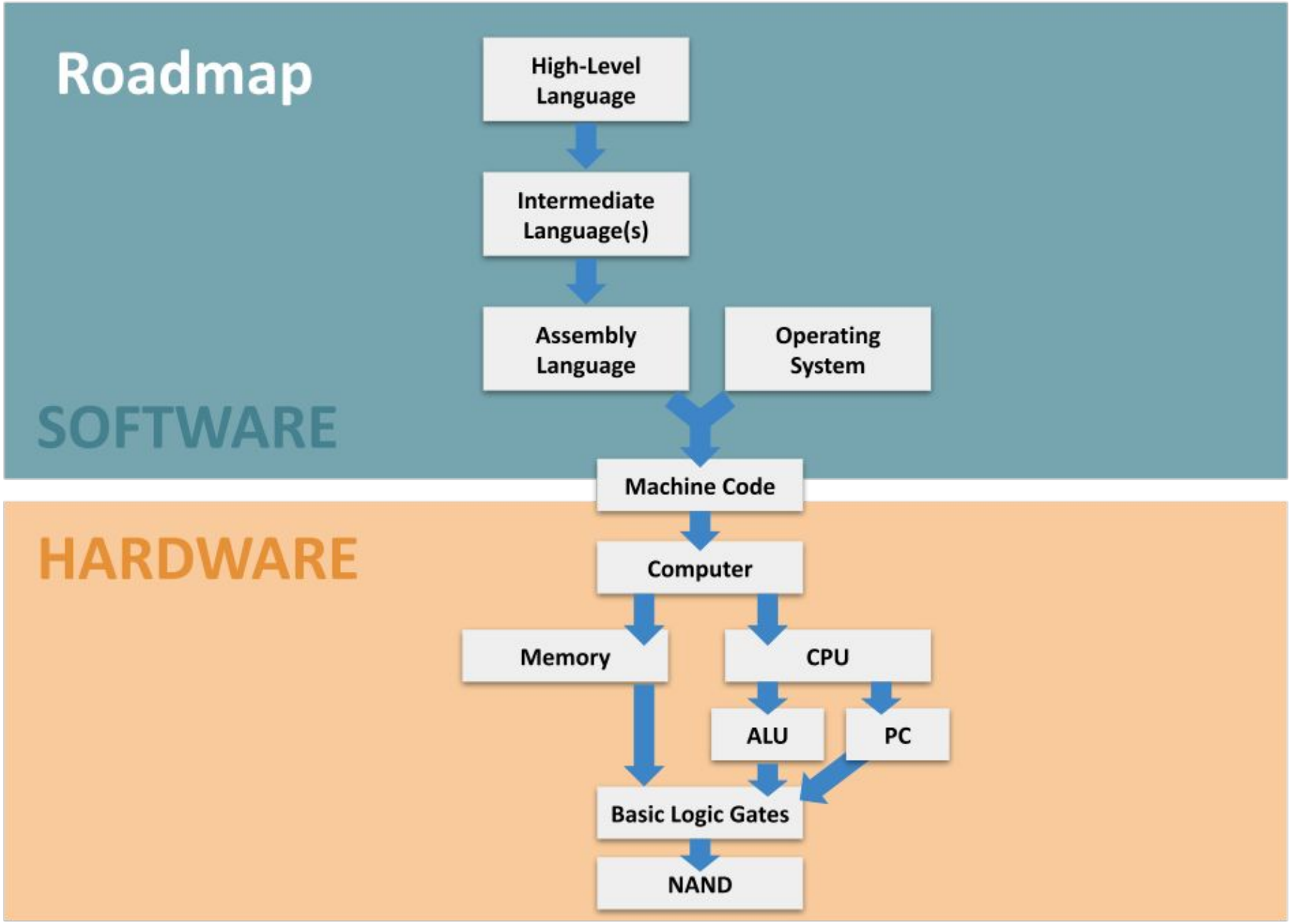
What are some ways for you to shift from thinking/knowing a practice could be helpful to actually implementing that practice?

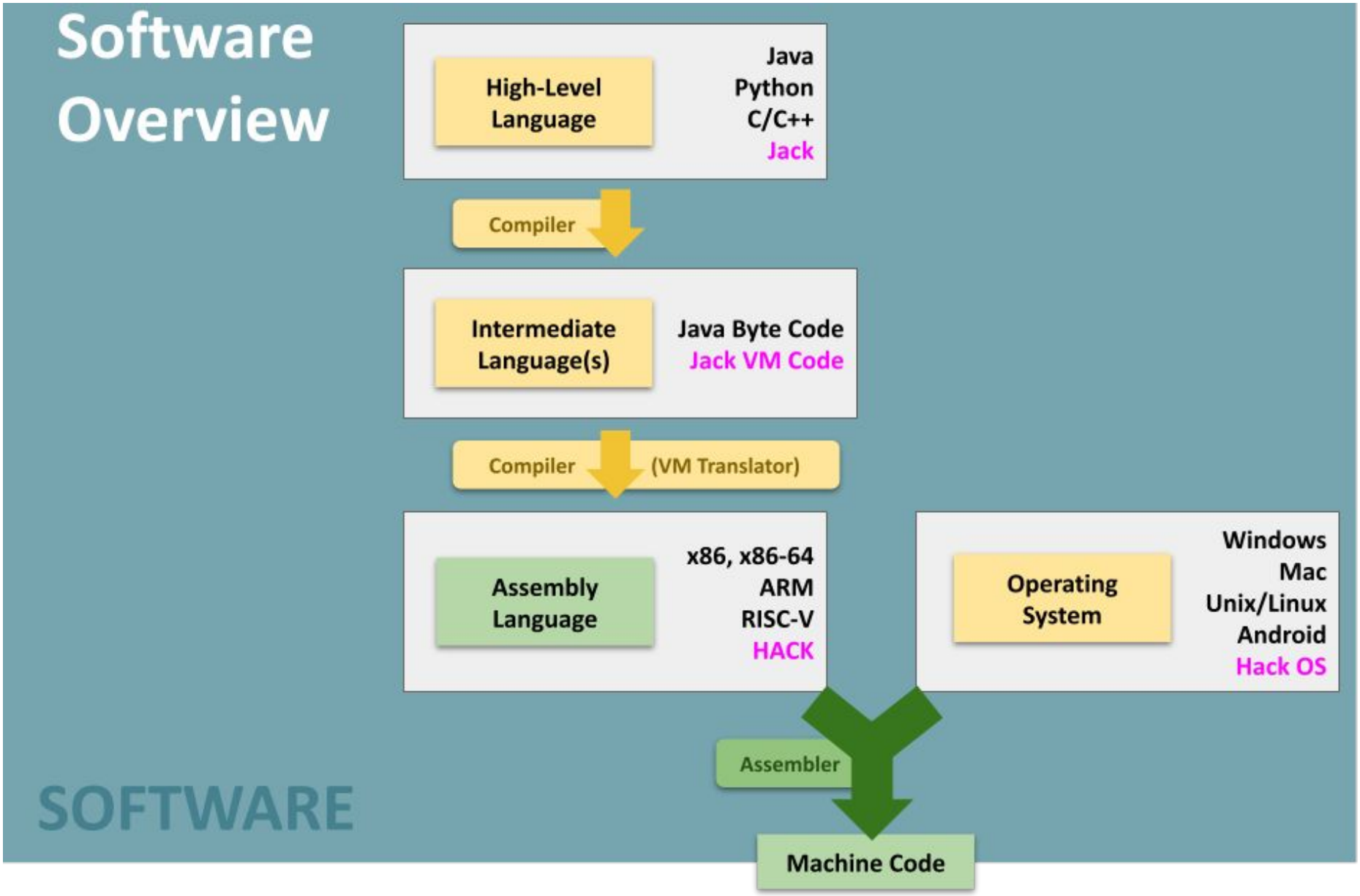
- Quarterly Calendar
- Weekly Time Commitments schedule
- 24-Hour Time Audit
- Project planning document

Before leaving the breakout session, execute **one** of the practices you identified.

Agenda

- Midterm Recap
- Revisiting Time Management
- **Where we've been, where we're at, & where we're going**
- Nand2Tetris Reflection/Look Ahead


Roadmap
High-Level Language
Intermediate Language(s)
Assembly Language
Operating System
SOFTWARE
Machine Code
HARDWARE
Computer
Memory
CPU
ALU
PC
Basic Logic Gates
NAND


Software Overview
High-Level Language
Java
Python
C/C++
Jack
Compiler
Intermediate Language(s)
Java Byte Code
Jack VM Code
Compiler
(VM Translator)
Assembly Language
x86, x86-64
ARM
RISC-V
HACK
Operating System
Windows
Mac
Unix/Linux
Android
Hack OS
Assembler
Machine Code
SOFTWARE

Agenda

- Midterm Recap
- Revisiting Time Management
- Where we've been, where we're at, & where we're going
- **Nand2Tetris Reflection/Look Ahead**

Bells and Whistles… Why Bother?

- Tradeoff: Adding convenience for programmer makes it harder to build programming languages
  - E.g. Removing symbols from Hack assembly would make Assembler much simpler, still possible to write all the same programs!
  - But language would be far more annoying to use

Bells and Whistles… Why Bother?

- Tradeoff: Adding convenience for programmer makes it harder to build programming languages
  - E.g. Removing symbols from Hack assembly would make Assembler much simpler, still possible to write all the same programs!
  - But language would be far more annoying to use

- **Don't underestimate the importance of convenience**!
  - Put another way: adding these extra features makes programmers more **productive**
  - Language design can also reduce the number of bugs, some of which can have serious real-world consequences

Bells and Whistles… Why Bother?

- Tradeoff: Adding convenience for programmer makes it harder to build programming languages
  - E.g. Removing symbols from Hack assembly would make Assembler much simpler, still possible to write all the same programs!
  - But language would be far more annoying to use

- **Don't underestimate the importance of convenience**!
  - Put another way: adding these extra features makes programmers more **productive**
  - Language design can also reduce the number of bugs, some of which can have serious consequences

- This is a general theme of programming language design!

Bells and Whistles… Why Bother?

- Example: HTTP (HyperText Transfer Protocol)
  - Loading a website: (1) Browser sends HTTP request to Server, (2) Server sends HTTP response back with website data

```
GET /midterm.pdf HTTP/1.1
Host: cs.washington.edu:80
Connection: close
```

Bells and Whistles… Why Bother?

- Example: HTTP (HyperText Transfer Protocol)
  - Loading a website: (1) Browser sends HTTP request to Server, (2) Server sends HTTP response back with website data
- Until 2015, every request was plain text!
  - Request MUST have hostname… but MUST precede with "Host:"
  - Connection can be "close" or "keep-alive". Why not 0 or 1?

```
GET /midterm.pdf HTTP/1.1
Host: cs.washington.edu:80
Connection: close
```

Bells and Whistles… Why Bother?

- Example: HTTP (HyperText Transfer Protocol)
  - Loading a website: (1) Browser sends HTTP request to Server, (2) Server sends HTTP response back with website data
- Until 2015, every request was plain text!
  - Request MUST have hostname… but MUST precede with "Host:"
  - Connection can be "close" or "keep-alive". Why not 0 or 1?
- Inefficient, but easy for programmers to develop with/debug → **widespread adoption!**

```
GET /midterm.pdf HTTP/1.1
Host: cs.washington.edu:80
Connection: close
```

Where We've Been: Hardware

- The physical devices that execute instructions
- Made up of a combination of both combinational logic and sequential logic
- Takes our theoretical code and actually performs computations to make it "real"
- Uses **instructions** that are part of an **instruction set** to determine what computations it should perform

Where We've Been: Machine Code

- Binary instructions that tell the hardware what to execute
- Hardware interprets the binary to know what the instruction means
  - Example: Our Hack instructions have 3 bits detailing where to store the result of a computation
- Really annoying to program in… hard to remember binary encoding, easy to introduce bugs, hard to find them, etc.

Where We've Been: Assembly

- Human readable format of machine code
- Each assembly instruction corresponds to exactly one binary instruction
- Adds some human-friendly features such as symbols and labels
  - These aren't represented explicitly in the machine code
- The **assembler** translates the human-readable assembly to binary machine code

Where We Are Going: Compiler

- Assembly is better to program in than machine code, but not as nice as a high level language like Java
  - If statements and for loops are much nicer than jumps...
- In order to program in a high level language like Java, need a tool to convert to assembly/machine code
- A **compiler** translates from one programming language to another (e.g. from Java to assembly code)
  - Our next big topic - we will build part of one in Project 7!

Reminders!

- **Eric & Margot's Office Hours** are happening right after class!

- **Project 5** due this Thursday 11:59PM PDT
  - Timed Mock Exam
  - Build a Computer!
  - Social Computing Reflection II

- **Autumn 2021 registration** has started

- **"Caring for myself is not self-indulgence, it is self-preservation…"**
  - Audre Lorde

  https://www.washington.edu/counseling/resources/