CSE 390B, Spring 2023

Building Academic Success Through Bottom-Up Computing

Design Decisions & Code Generation

Design Decisions in Computing, Compilers: Code Generation, Two-Tier Compilation

Lecture Outline

- **Design Decisions in Computing**
  - **Understanding Design and Its Importance**
  - **Design Decisions in Computing**
- Compilers: Code Generation
  - Generating Target Code from an AST
- Two-Tier Compilation
  - Intermediate Programs and The Java Virtual Machine (JVM)

What is Design?

- The way something works, including how someone uses it
  - Almost always includes some element of interaction
- Design could have different definitions, goals, and interpretations in different contexts
  - It's also not always about the end-user of a product
  - For example, you might design a codebase that's easier to maintain
- Everything we create has design, but there is a range to how intentional the design of something is
  - Could be completely forgotten
  - Could be focused on throughout the creation of something

Why Talk About Design?

- If design is "the way something works, including how someone uses it," then it dictates the interactions between us and the designs we interact with

- Those interactions have a range of consequences
  - Positive: When you go to a website, and you are easily able to find all the information you need
  - Unideal: If a person can't easily drink from a certain cup
  - Harmful: If a person can't easily use emergency equipment

Why Talk About Design?

- Seemingly harmless interactions can have a negative impact on certain people, especially if repeated
  - E.g., Unable to use any door you see will make you feel unwelcome
- How can we design to create more positive reactions for more people while mitigating negative interactions?
  - Tough question in a world with so many diverse people
- What accountability should there be for more harmful interactions caused by the design of something?
  - A significant question with a muddy web of answers

An Aside: Bias

- Biases are the beliefs we have, often formed by our experiences
  - Can be **explicit**: We consciously have a belief about something and it may intentionally impact us
  - Can be **implicit**: Unconscious or impact us unintentionally

- We all have bias, and it is not inherently “good” or “bad”
  - Both potentially beneficial and potentially harmful consequences

- Eliminating bias is not a realistic goal
  - Attempting to mitigate negative consequences that come from bias is more realistic

Designer's Bias

- People often think of the "typical user" as someone who is similar to them or those they are close to
  - An example of the influence of their biases
- Even if we try to think beyond what is familiar to us, it is unlikely we will remove bias from the design process
  - Opinions about what something "should" do are inherently biased
- Ideally, we would develop processes that mitigate the negative effects of biases as much as possible
  - Recall biases can be both known (explicit) and unknown (implicit)

Bias and Design

- Following slides include some ideas and frameworks people have come up with related to bias and design

- Not meant to be the most important ideas
  - Think of it more as a few reference points that you can learn more about beyond this lecture
  - Discussions about bias and design are very nuanced and constantly evolving

- None completely solve these issues, but they can be used to think about them and build better practices

Universal Design

- Big idea: Design things that can be used by as many people as easily as possible
- Designing things that work well for a wide range of people includes those who might usually be excluded
  - For example: Video captioning
- The process of "including everyone" leads us to better design

Inclusive Design

- Including as diverse a range of perspectives when designing something as possible
  - Similar to universal design, but you may offer different solutions for different types of people (rather than one solution for all)
  - "Including" a diverse perspective does not just mean having a diverse team of people
  - It means valuing a diversity of opinions and experiences

- If we prioritize diverse perspectives, especially those that have been typically excluded, it will lead to things that benefit more people

Affordance Theory

- Way of thinking about things around us

- Things provide different affordances to people
  - A way of defining what the capabilities of something are

- Can group these affordances into different categories:
  - What affordances does someone think/perceive something provides them?
  - What affordances does something actually provide someone?

Affordance Types

- Four types of affordances (in reality, it's more of a spectrum)
  - **Perceptible affordance:** something does what someone thinks it can
  - **Hidden affordance:** something does what someone thinks it can't
  - **False affordance:** something doesn't do what someone thinks it can
  - **Correct rejection:** something doesn't do what someone thinks it can't

Design Principles in Practice

- In groups, discuss the following questions:
  - Observations of design in the real world
  - Experiences you have had with technology that has privileged or discriminated against you
  - How might you design these technologies differently to be more inclusive?

Design in Computing

- Design discussions are relevant to computing
  - Many were developed with design in mind

- Technology can be biased
  - Design is part of almost everything in computing
  - Our biases influence the design of things

- A lack of diverse perspectives can lead to many harmful designs

Design in Computing: Accessibility

- There is a large community in CSE focused on making technology more accessible for people
  - E.g., making web pages easily navigable for people who are blind
  - E.g., expanding internet access to remote populations

- Connection: Elements of both universal design and inclusive design
  - Universal design: Designing products that work for as many people as possible
  - Inclusive design: Including more perspectives in the design process, and potentially developing specific solutions aimed at including different groups of people

Design in Computing: Algorithmic Bias

- Research related to bias in AI/ML algorithms
  - E.g., Facial recognition technology not working as well on people of color (trained on primarily white datasets)
  - E.g., Racial bias in crime prediction algorithms (reflects the bias of our criminal justice system)

- These results biases reflect biased design decisions throughout development
  - Picking datasets biased towards certain communities
  - Testing applications in biased environments
  - Bias in what is prioritized within an algorithm

Moving Towards Inclusive Design

- Design is often categorized as being separate from other parts of the development process
  - In reality, happens in almost every stage of developing something
- You can voice feedback and concerns in design
  - You are ultimately contributing to the design of it
  - What conversations already occur, then ask how we can do better
- Different vision of how to approach building technology
  - Slogan offered by Animikii: “Move slow and empower people”

Next Steps in Design

- Brief overview of design that only scratches the surface

- Entire fields and majors related to design and computing
  - Human Computer Interaction (HCI)
  - User Experience (UX/UI)
  - Human Centered Design and Engineering (HCDE, major at UW)

- Related courses:
  - CSE 340: Interactive computing
  - CSE 440: Introduction to HCI
  - SOC 225: Data and society
  - HCDE department has some neat courses on this topic too!

Lecture Outline

- Design Decisions in Computing
  - Understanding Design and Its Importance
  - Design Decisions in Computing
- **Compilers: Code Generation**
  - ***Generating Target Code from an AST***
- Two-Tier Compilation
  - Intermediate Programs and The Java Virtual Machine (JVM)

Software Overview

High-Level Language

Java
Python
C/C++
Jack

Compiler

Intermediate Language(s)

Java Byte Code
Jack VM Code

Compiler (VM Translator)

Compiler

(Project 8)

Assembly Language

x86, x86-64
ARM
RISC-V
HACK

Operating System

Windows
Mac
Unix/Linux
Android
Hack OS

Assembler

Machine Code

SOFTWARE

The Compiler: Implementation

```
public int fact(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

High-Level Language

```
(fact)
  @R0
  M=M+1
  @R1
  D=A
  @ifbranch
  D;JEQ
```

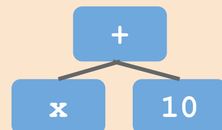
Assembly Language

Scanner → Parser → Type Checker → Optimizer → Code Generator

Scanner: Break string into discrete **tokens**:

IF | (| ID(n) | == | NUM(0) etc.

Parser: Arrange tokens into **syntax tree**:



Type Checker: Verify the syntax tree is **semantically correct**

Optimizer: Rearrange the code to be **more efficient**

Code Generator: Convert the syntax tree to the **target language**

Lecture 15: Design Decisions & Code Generation

When poll is active, respond at **PollEv.com/cse390b**

What of the following is FALSE about the Abstract Syntax tree (AST)?

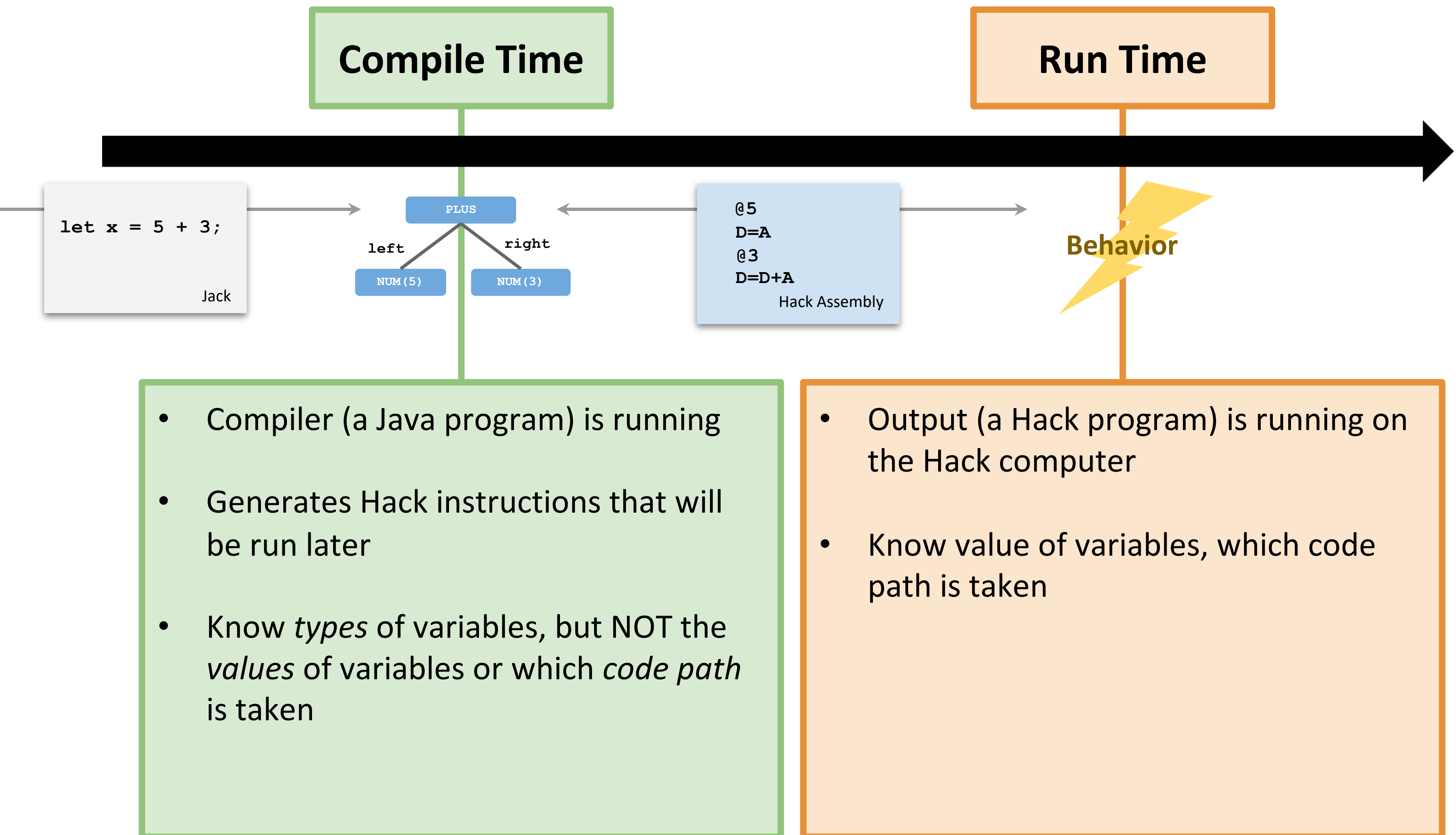
- ASTs are a general, recursive structure that allows us to describe any program
- ASTs allow the compiler to extract relevant parts of a computer program
- ASTs enable the compiler to perform type checking of the input program
- ASTs perform optimizations in the intermediate representations of a program
- We're lost…

Total Results: 0

Code Generation: The Task

- Convert the AST into **target language code** that produces the same result
- Project 8 goal: Produce **reliable**, not efficient, compiler
- The tricky bit: Do it automatically for all possible arrangements of code
  - To stay sane, we'll break the task down: Generate code *for each node type* in the AST

Compile Time vs. Run Time

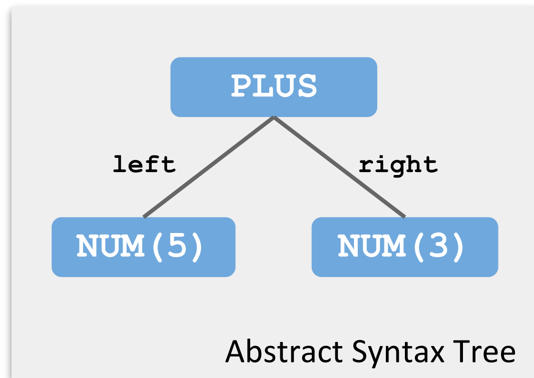


- Compiler (a Java program) is running
- Generates Hack instructions that will be run later
- Know *types* of variables, but NOT the *values* of variables or which *code path* is taken

- Output (a Hack program) is running on the Hack computer
- Know value of variables, which code path is taken

Code Generation: Example

- How would you, a brilliant human, translate this abstract syntax tree into Hack Assembly?



Abstract Syntax Tree

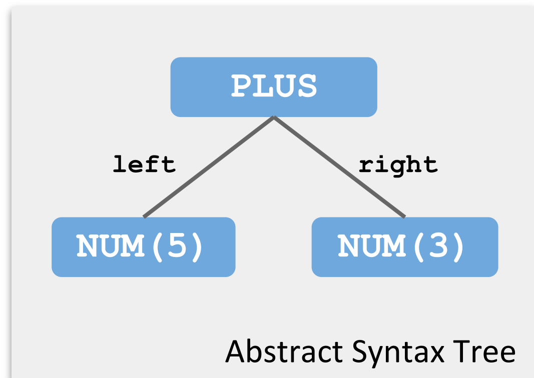
Code Generation: Example

- Here's how you, a brilliant human, would likely translate this syntax tree into Hack:

```
@5
D=A
@3
D=D+A
```

Hack Assembly

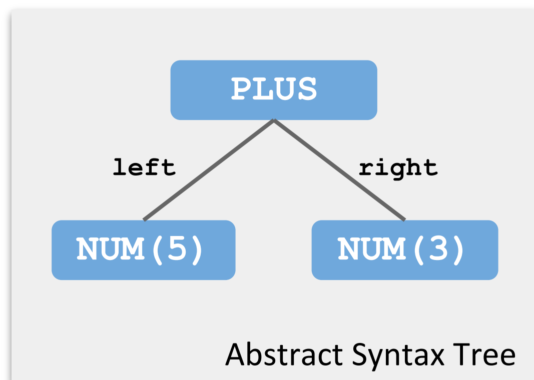
Human (genius)



Abstract Syntax Tree

Code Generation: Example

- Here's how you, a brilliant human, would likely translate this syntax tree into Hack:



Abstract Syntax Tree

Human (genius)

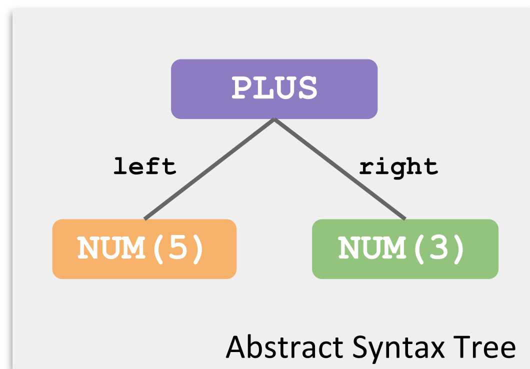
```
@5
D=A
@3
D=D+A
```
Hack Assembly

Computer (trying its best)

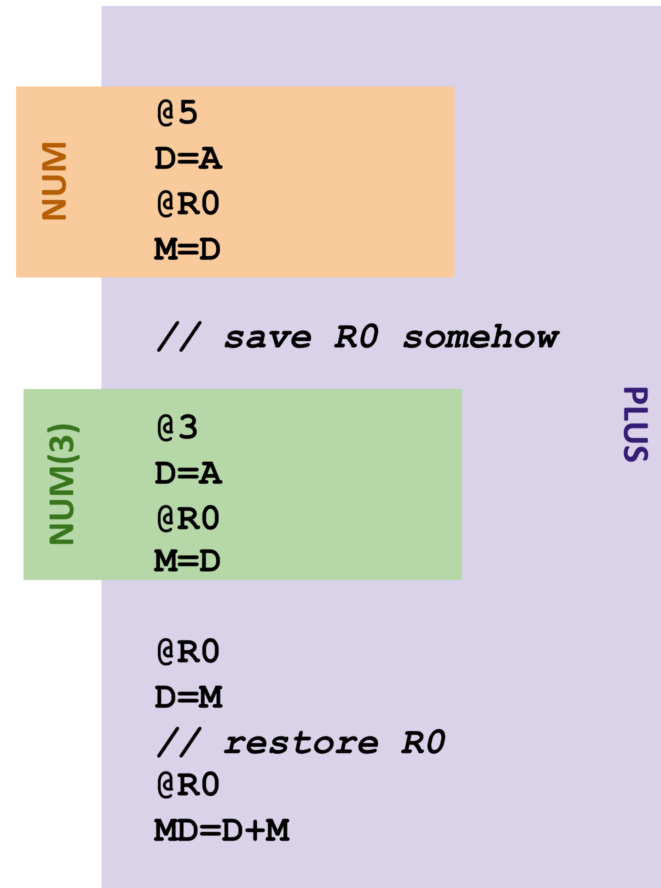
```
@5
D=A
@R0
M=D
// save R0 somehow

@3
D=A
@R0
M=D

@R0
D=M
// restore R0
@R0
MD=D+M
```
Hack Assembly

Code Generation: Example

- Why? Modularity: We can fit any expression in that slot, as long as **its result ends up in R0!**

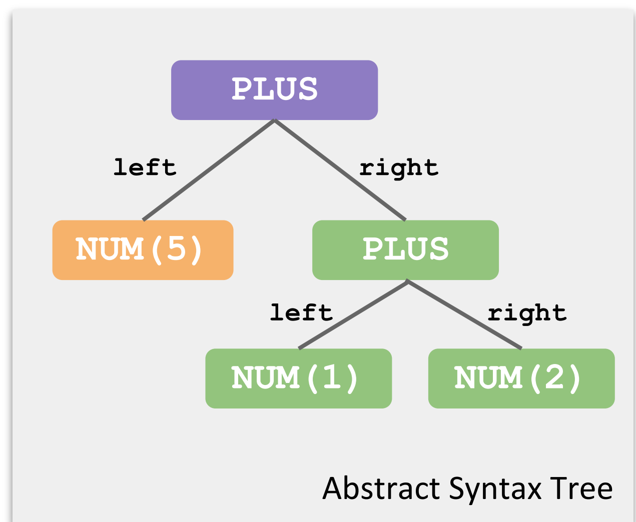


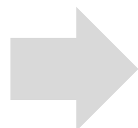
Computer (actually, quite clever!)

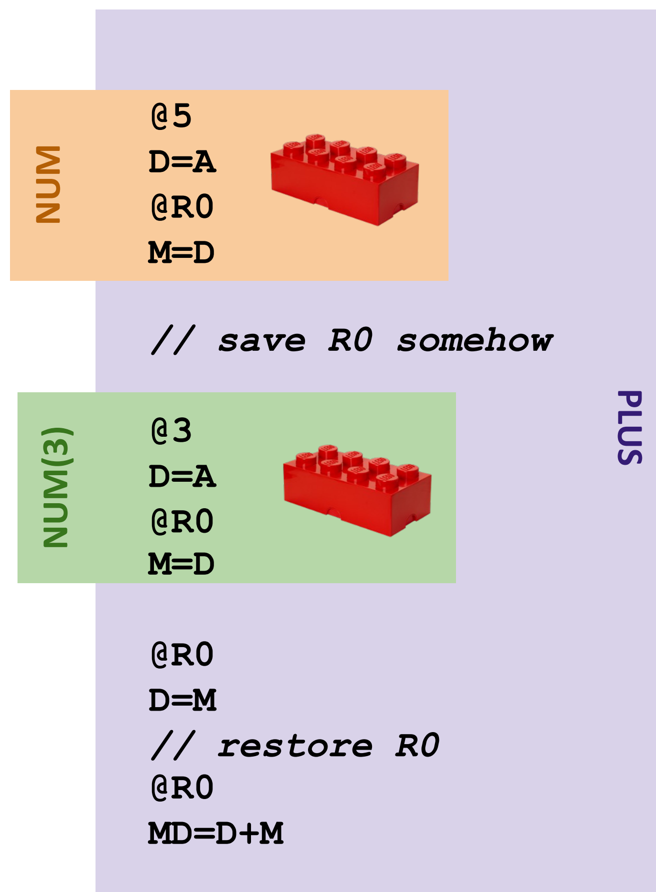


Code Generation: Example

- Why? Modularity: We can fit any expression in that slot, as long as **its result ends up in R0!**
  - Even another PLUS

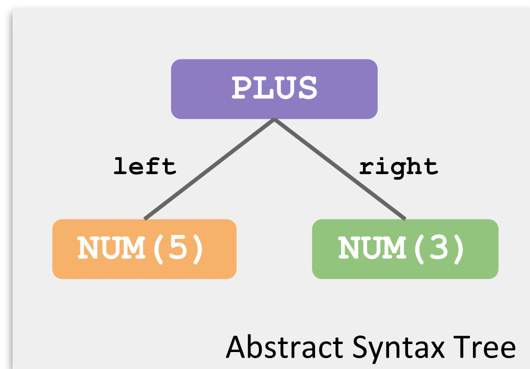


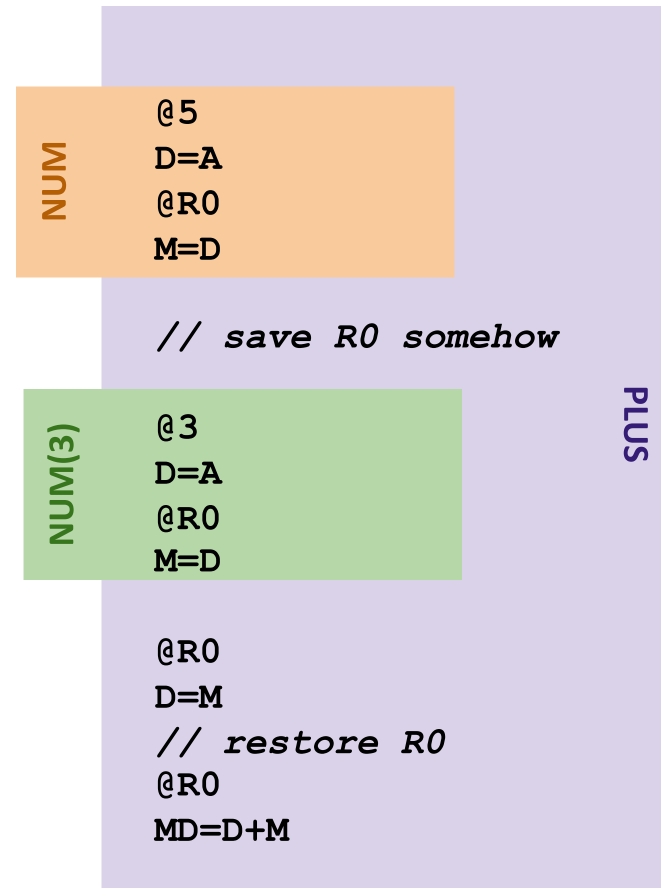




Code Generation: Example

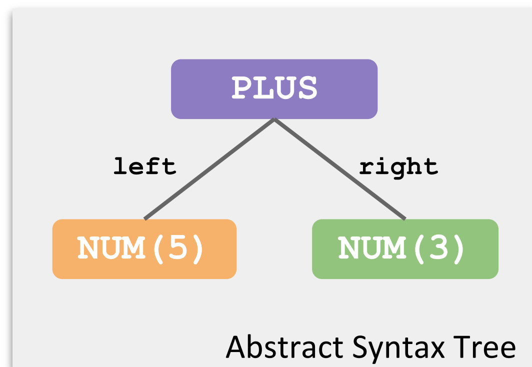
- Now, we need to save R0 somehow

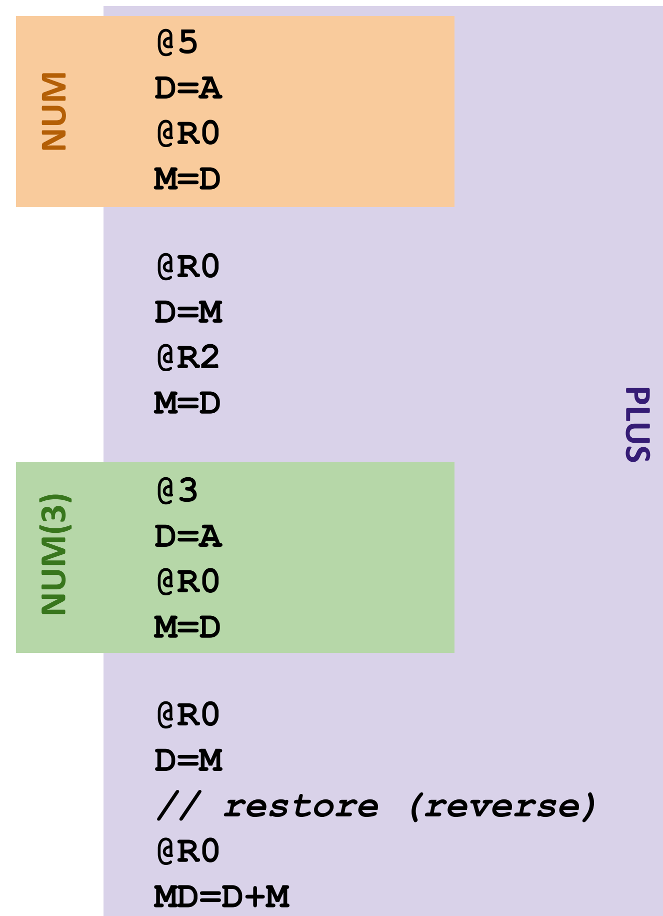




Code Generation: Example

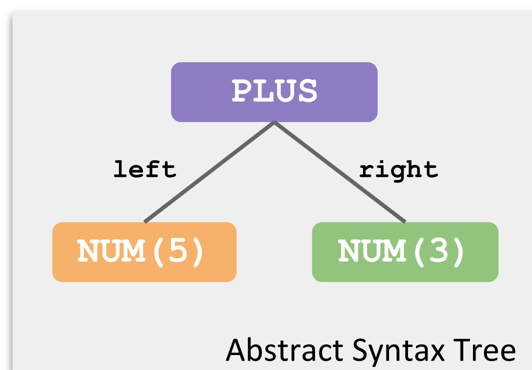
- Now, we need to save R0 somehow
  - What if we save it in a temporary register? Let's pick R2

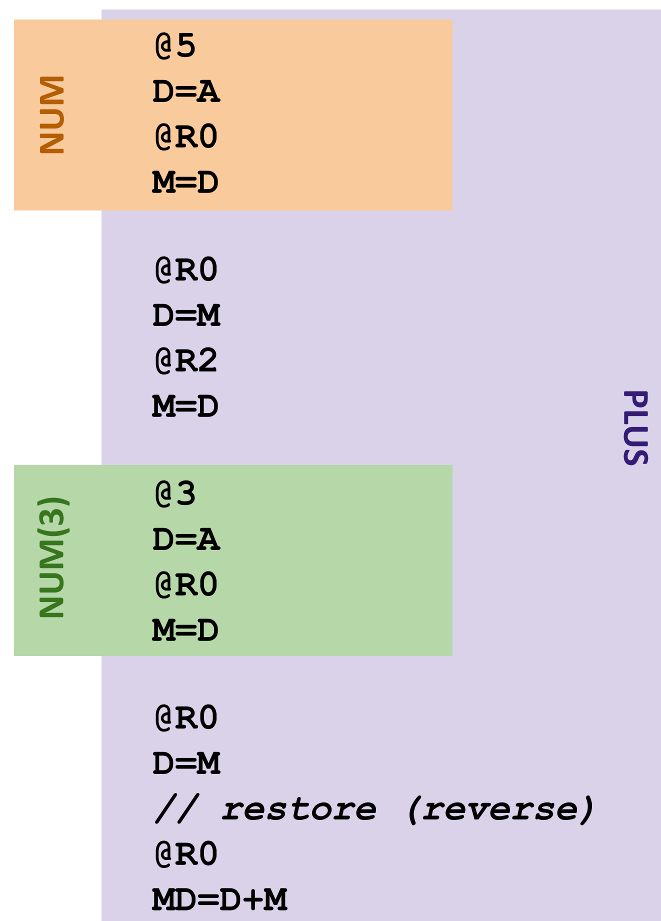




Code Generation: Example

- Now, we need to save R0 somehow
  - What if we save it in a temporary register? Let's pick R2

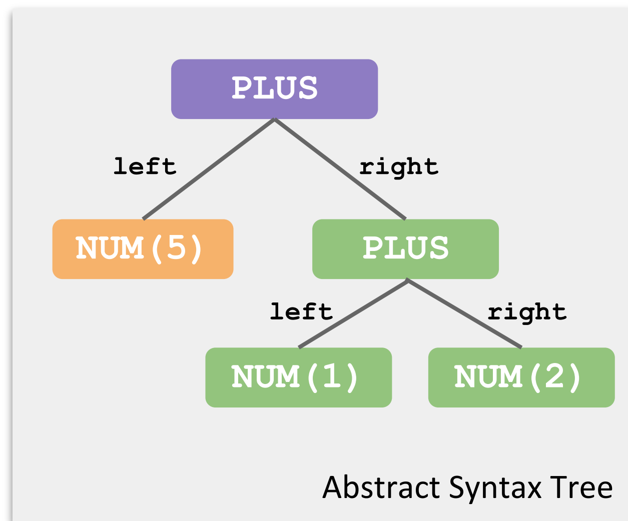




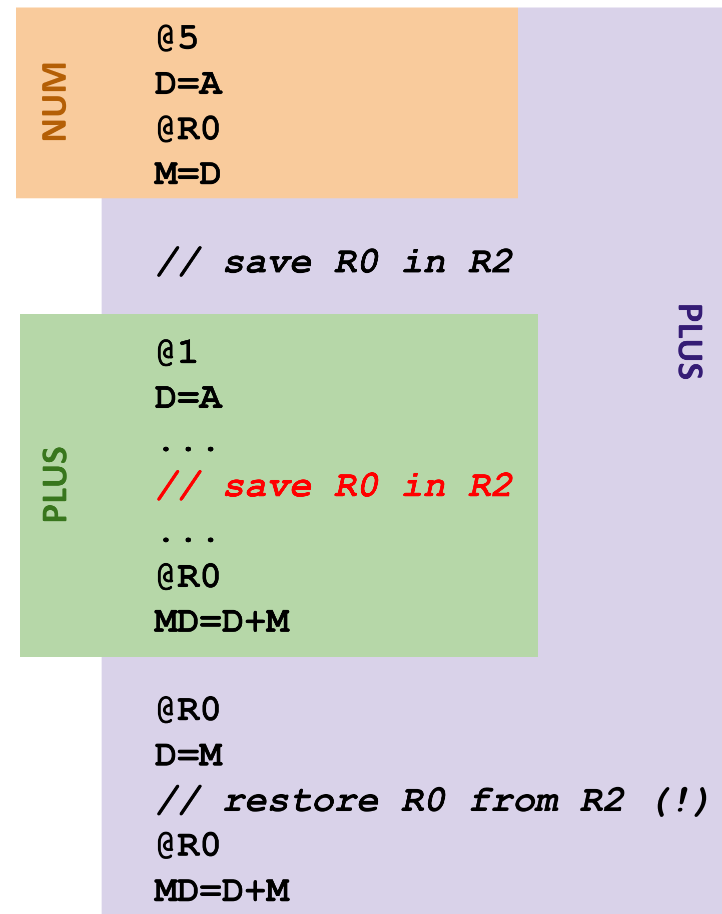
Why won't this always work?

Code Generation: Example

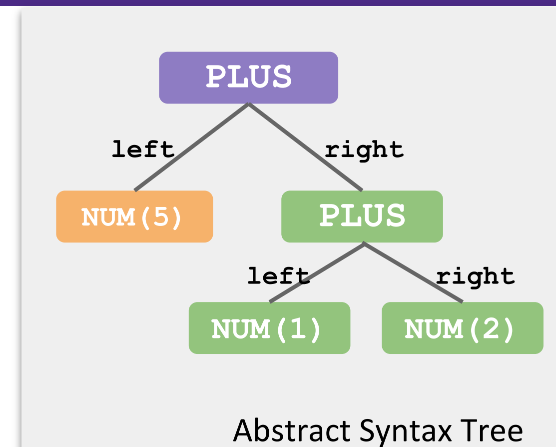
- It's those pesky nested expressions! The outer PLUS saves a value in R2, but the inner PLUS overwrites that value during its computation



Abstract Syntax Tree



Code Generation: Example



- Solution: Store "saved" values in a stack
  - Not quite the same as "The Stack" or function call stack frames (but used for a similar reason)
- We'll keep a stack starting at memory address 1024
  - R1 is our *stack pointer*: always stores address of last used stack position
  - No built-in Hack push: manually copy to memory and increment R1

NUM

```
@5
D=A
@R0
M=D
```

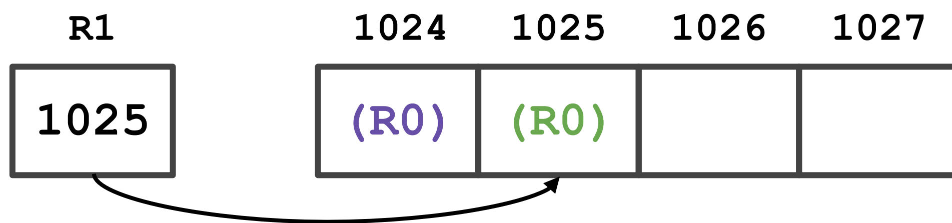
PLUS

```
// push R0 to slot 0
```

PLUS

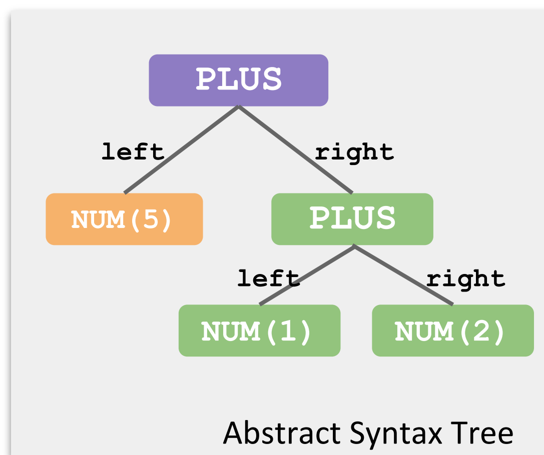
```
@1
...
// push R0 to slot 1
...
// pop R0 from slot 1
@R0
MD=D+M
```

```
@R0
D=M
// pop R0 from slot 0
@R0
MD=D+M
```

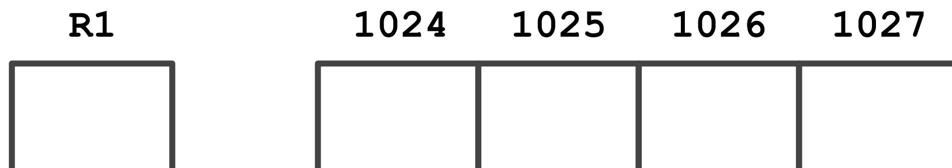


Code Generation: Example

- Let's trace through an example of using the stack in code generation for the following AST:



Abstract Syntax Tree




```
NUM:
@5
D=A
@R0
M=D

// Push R0 to addr 1024

PLUS:
@1
D=A
@R0
M=D
// Push R0 to addr 1025
@2
D=A
@R0
M=D

@R0
D=M
// Pop R0 from addr 1025
@R0
MD=D+M

@R0
D=M

// Pop R0 from addr 1024

@R0
MD=D+M
```
(PLUS)

Code Generation: Example

- What about variables?

```
var int arr[5];
var int bar, star;

let bar = star;
```

Jack

```
@261
D=M
@262
M=D
```

Hack Assembly

arr	256
bar	261
star	262
screen	16384

- Just like Assembler: Generate symbol table with mapping from variable names to spots in memory
  - Arrays get more (contiguous) spots
  - **`screen`** and **`keyboard`** are built-in array variables, allowing I/O

Code Generation: Takeaways

- Code Generation task: Writing several small snippets of Hack assembly
  - But need to be very generalizable
  - Whenever a PLUS expression is encountered, should generate almost the same code
- Conventions make the task much easier
  - For example, after any expression code runs, result should always be stored in R0
  - Then parent code can depend on it

Lecture Outline

- Design Decisions in Computing
  - Understanding Design and Its Importance
  - Design Decisions in Computing
- Compilers: Code Generation
  - Generating Target Code from an AST
- **Two-Tier Compilation**
  - **Intermediate Programs and The Java Virtual Machine (JVM)**

Software Overview

High-Level Language
Java
Python
C/C++
Jack

Compiler

Intermediate Language(s)
Java Byte Code
Jack VM Code

Compiler (VM Translator)

Compiler
Project 8

Assembly Language
x86, x86-64
ARM
RISC-V
HACK

Operating System
Windows
Mac
Unix/Linux
Android
Hack OS

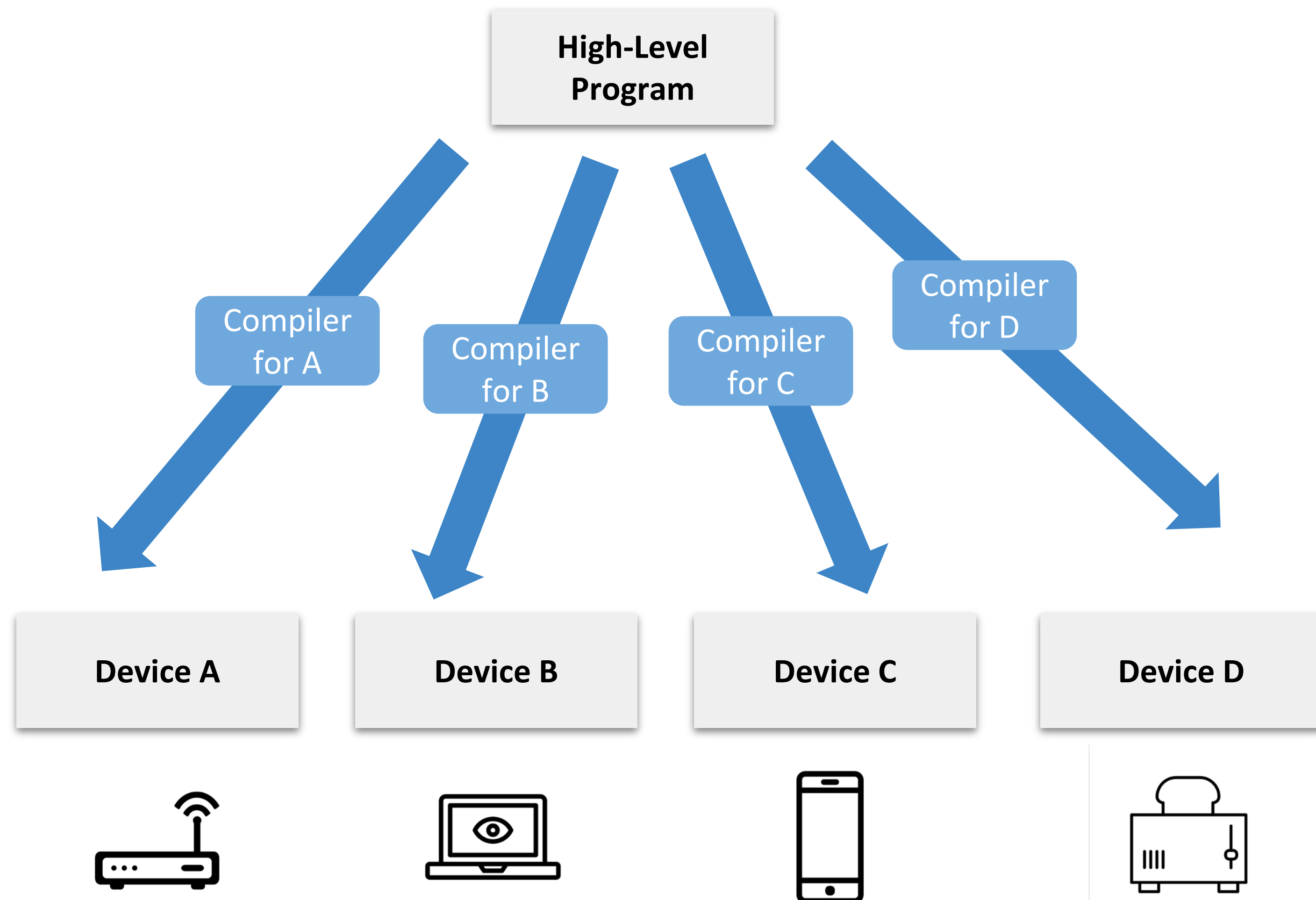
Assembler

Machine Code

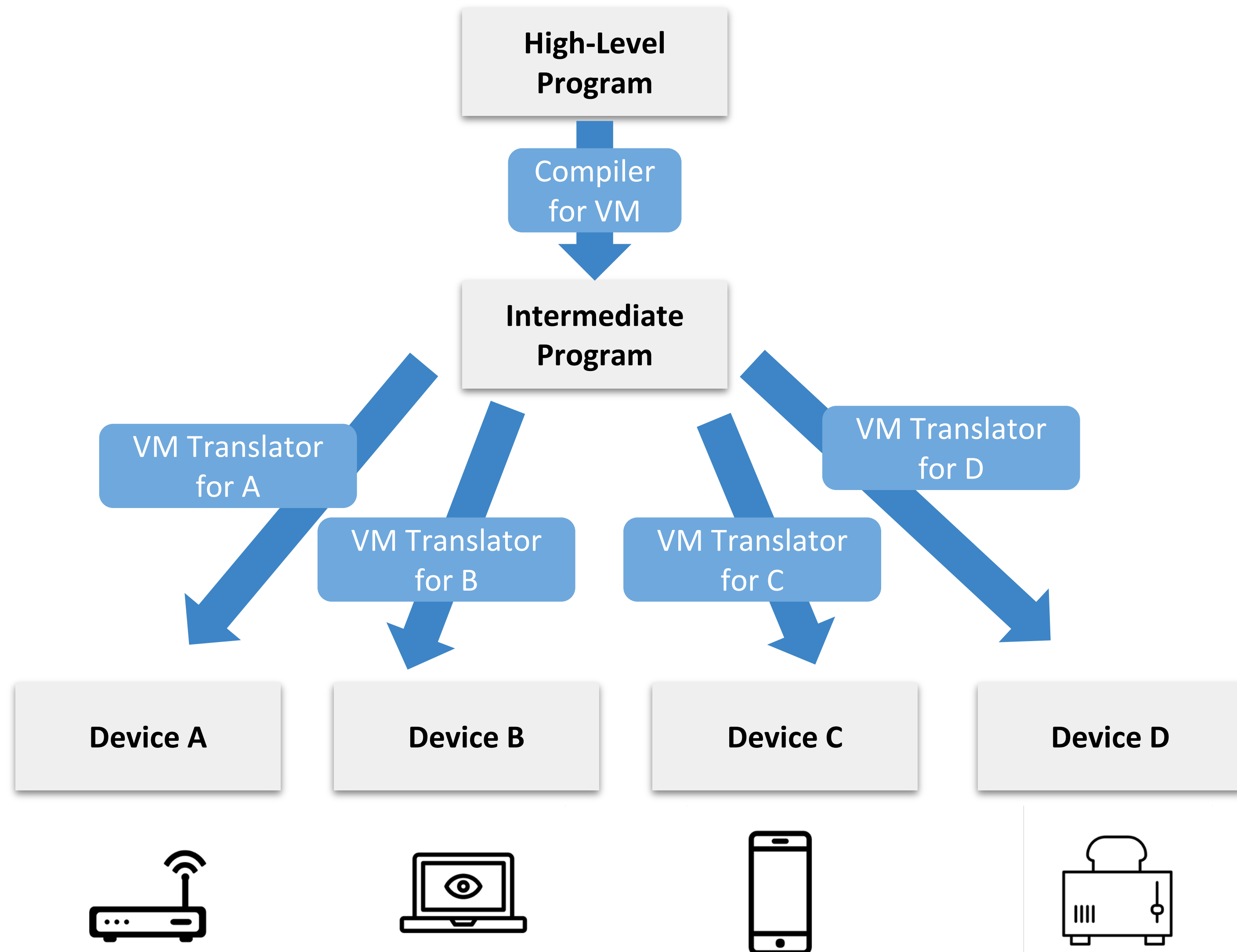
SOFTWARE

KEY: "Real-World" Examples
Our Computer

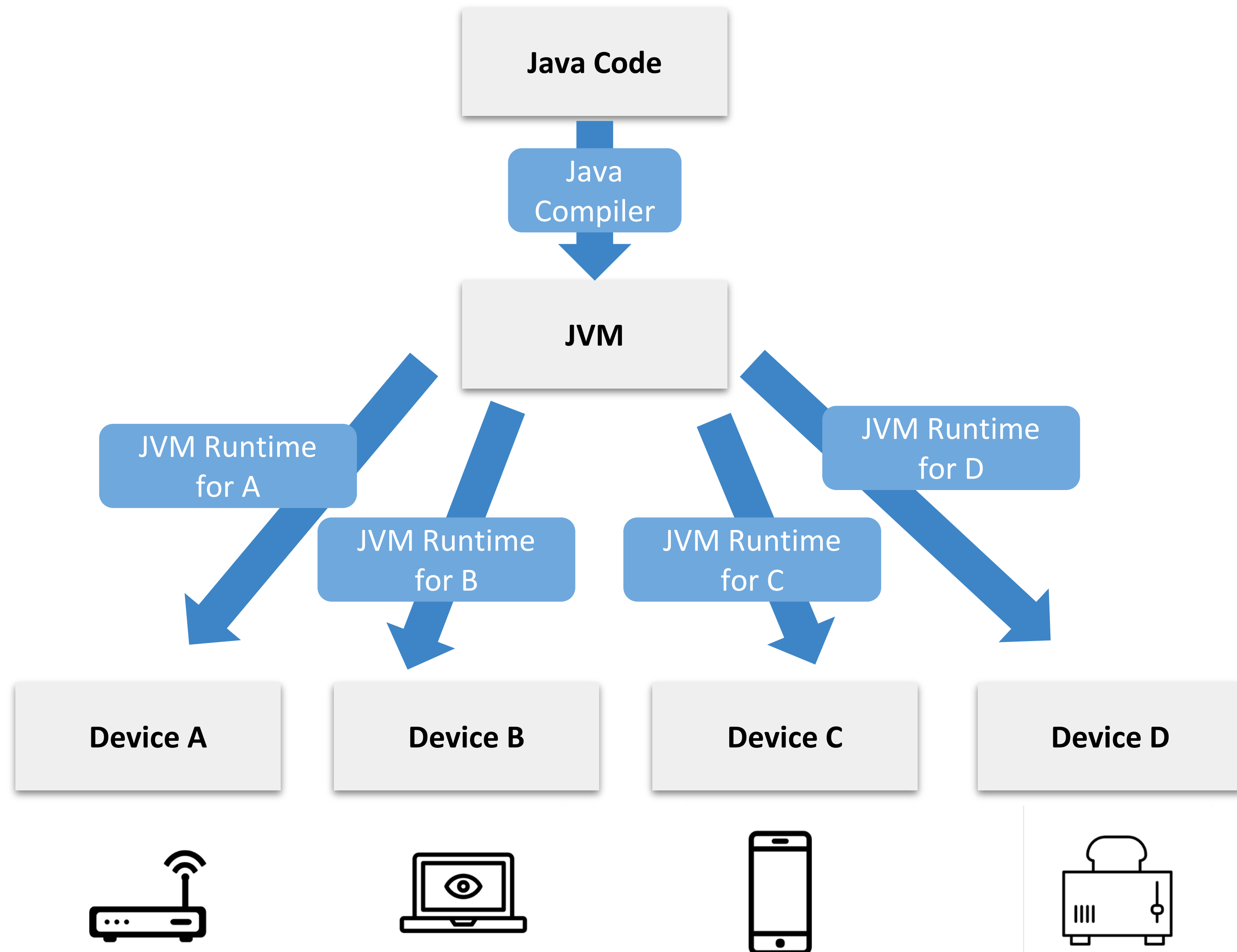
Compiling Code: Single Tier



Compiling Code: Two Tier



The Java Virtual Machine (JVM)



Post-Lecture 15 Reminders

- Project Reminders
  - **Project 7, Part I: Midterm Corrections due this Thursday (5/18) at 11:59pm (no late days may be used)**
  - Project 7, Part II: Professor Meeting Report due next Thursday (5/25) at 11:59pm

- Preston has office hours after class in CSE2 152
  - Feel free to post your questions on the Ed board as well