

# Machine Programming

Lecture 1 – Overview and Introduction to Synthesis

Ziyang Li

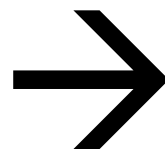
# Instructor



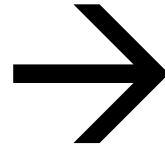
## Ziyang Li

- Assistant professor @JHU CS, 2025-
- Before that: PhD at University of Pennsylvania
- Research areas: Programming Language + Machine Learning
- Favorite PL: Rust & JavaScript

# Programming...



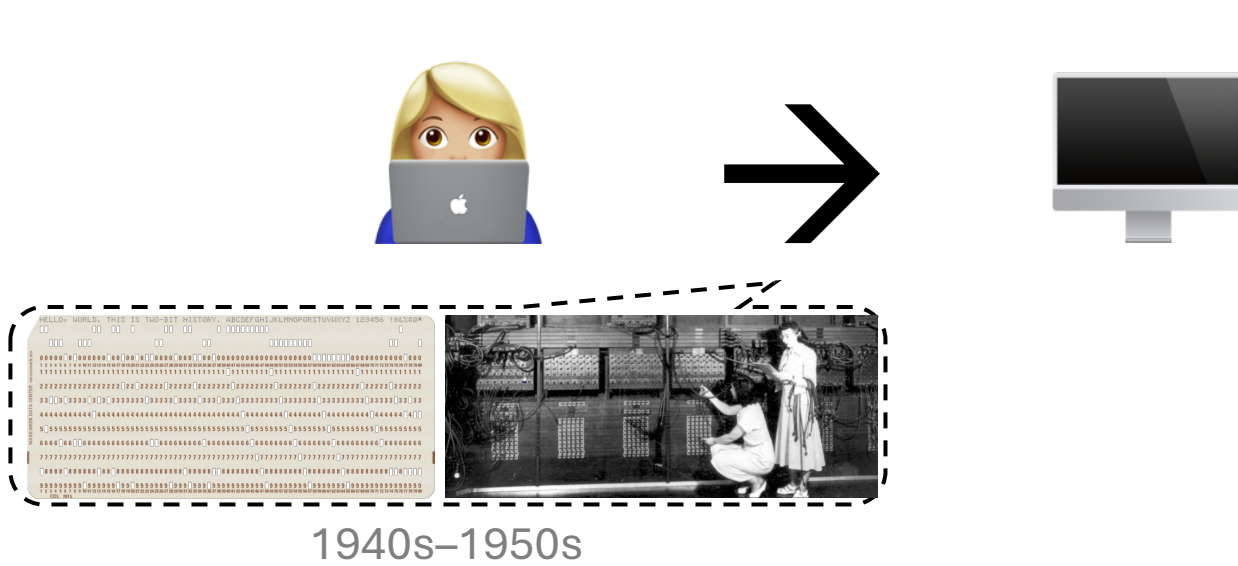
# Programming...

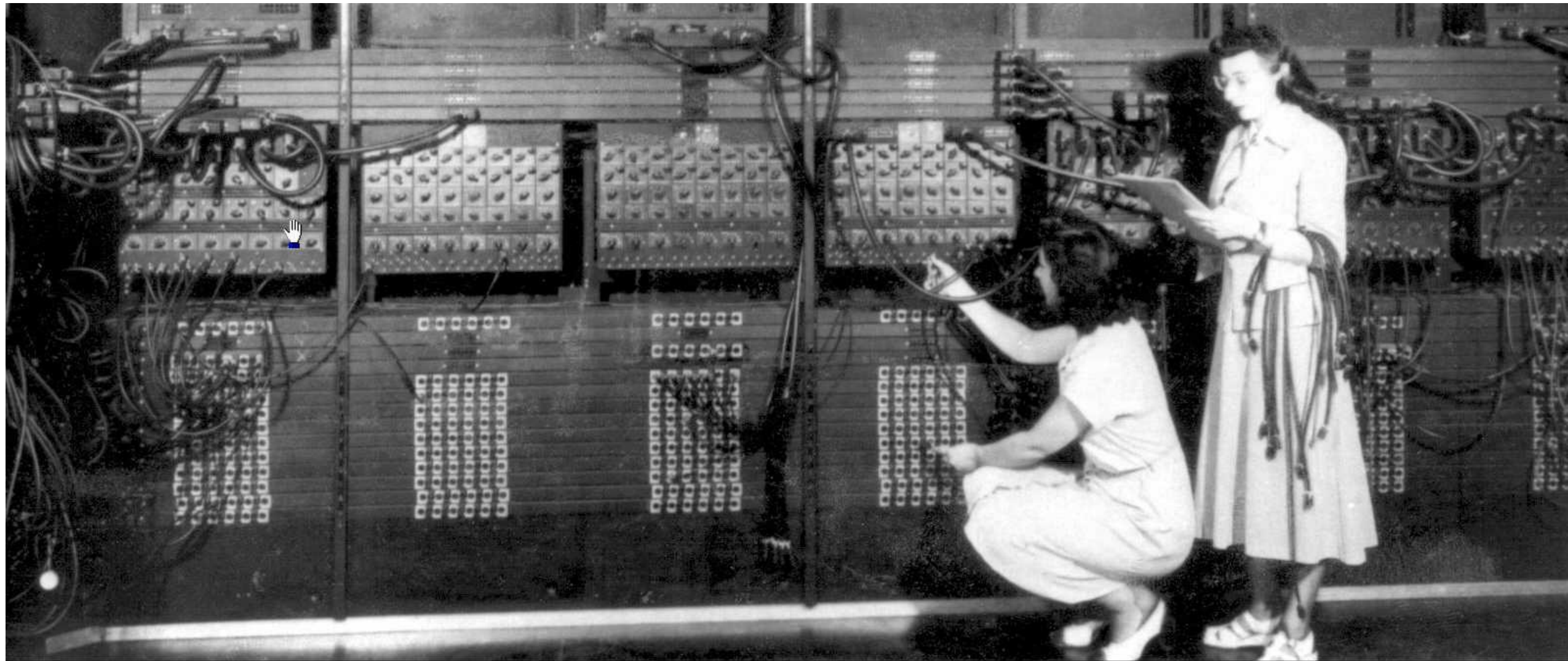


0101010101

1940s–1950s: Binary

# Programming...







1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

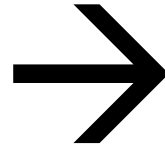
[illegible][illegible][illegible]

////////////////////

[illegible]

CDL 0815

# Programming...

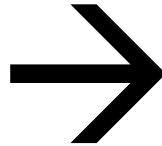


0101010101

1940s–1950s: Binary

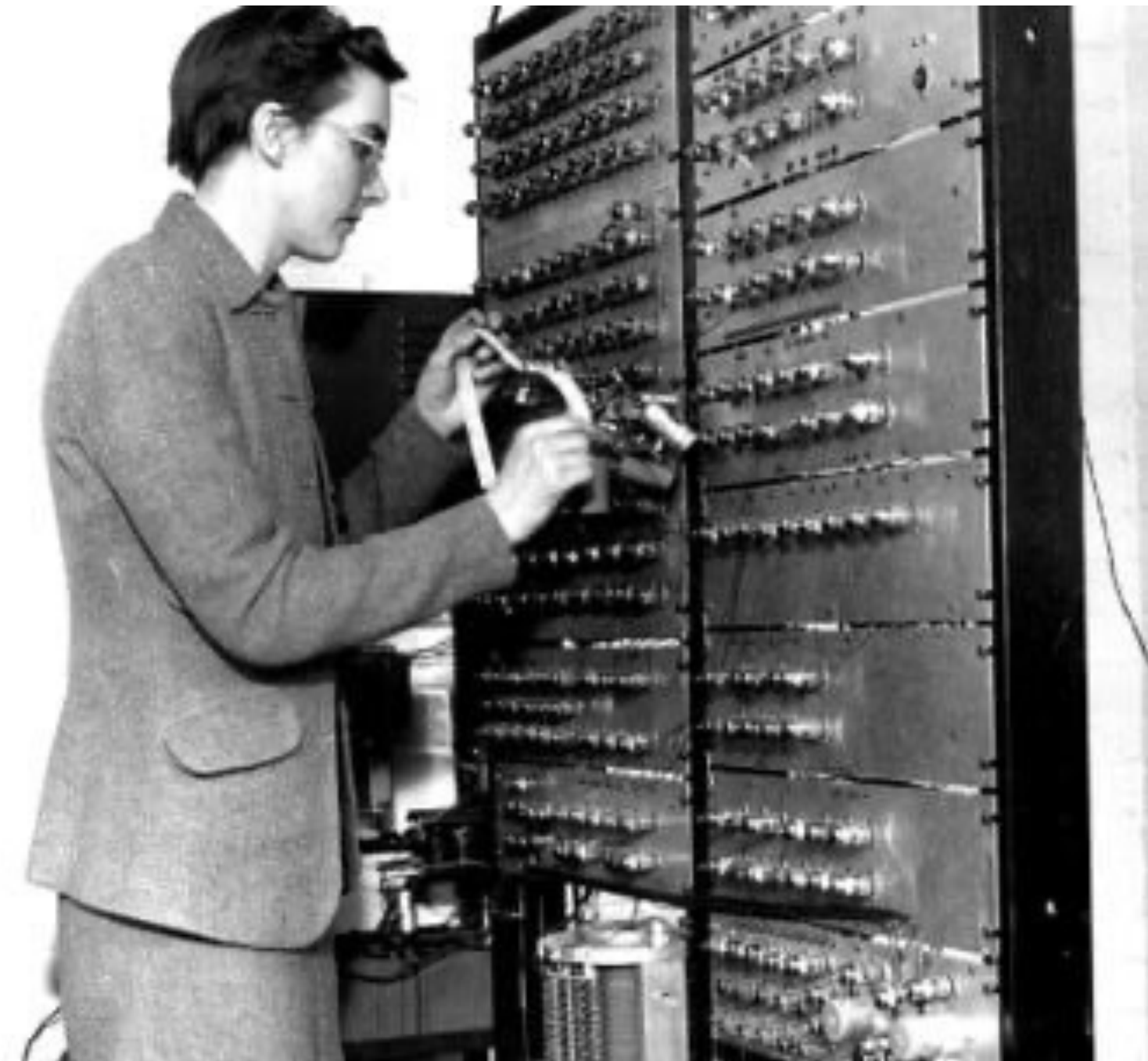


# Programming...



ADD R1, R2

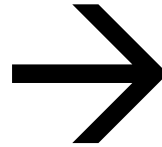
1950s–1960s: Assembly



Kathleen Booth

1922–2022

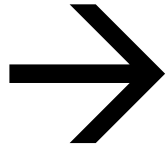
# Programming...



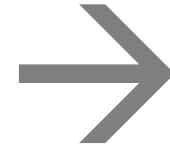
ADD R1, R2

1950s–1960s: Assembly

# Programming...



Assembler



ADD R1, R2

1950s–1960s: Assembly

0101010101

# The FORTRAN Automatic Coding System

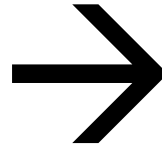
J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†,  
H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†,  
H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT||

## INTRODUCTION

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as

system is now complete. It has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs. Descriptions of the FORTRAN language and the translator form the principal

# Programming...

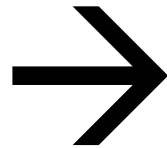


```
for (i = 0; i < n; i++)  
    sum += a[i];
```

1970s: High-level Language



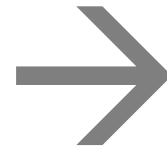
# Programming...



Compiler



Assembler



```
for (i = 0; i < n; i++)  
    sum += a[i];
```

1970s: High-level Language

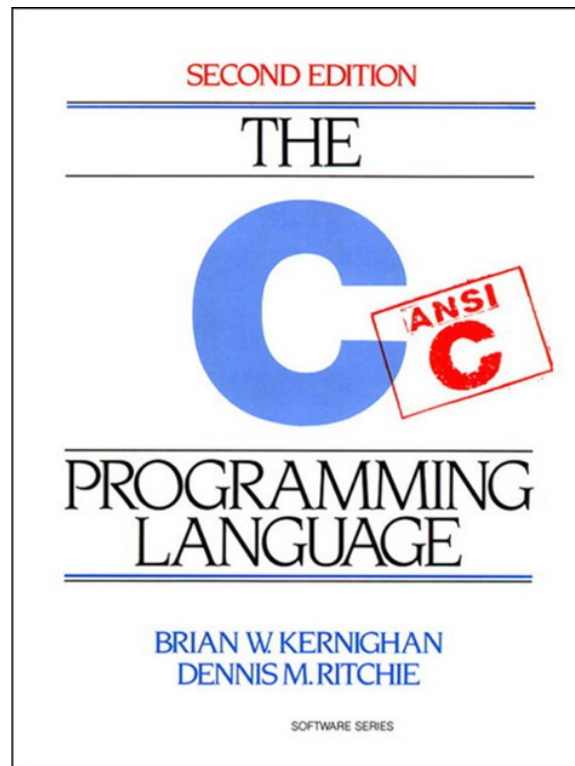
```
ADD R1, R2
```

```
0101010101
```

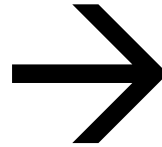


# Dennis M. Ritchie

1941–2011



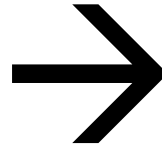
# Programming...



```
for (i = 0; i < n; i++)  
    sum += a[i];
```

1970s: High-level Language

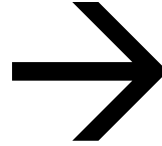
# Programming...



```
for i in range(n):  
    sum += a[i]
```

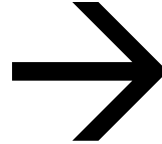
1980s onwards: Modern High-Level Languages

# Programming...



```
for i in range(n):  
    sum += a[i]
```

# Programming...



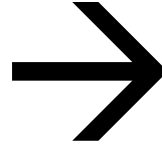
```
for i in range(n):  
    sum += a[i]
```



```
for (var e of a) {  
    sum += e; }  
}
```



# Programming...



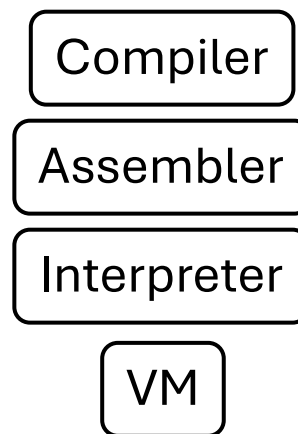
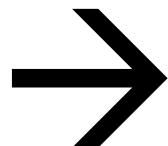
```
for i in range(n):  
    sum += a[i]
```



```
for (var e of a) {  
    sum += e; }  
}
```



```
total :: Num a => [a] -> a  
total a = foldl (+) 0 a
```



...



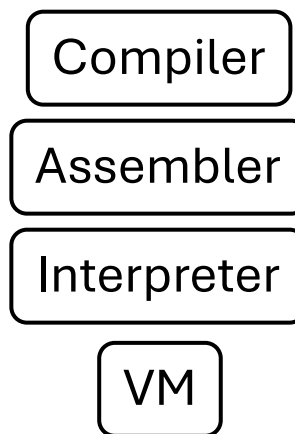
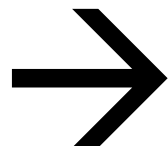
```
for i in range(n):  
    sum += a[i]
```



```
for (var e of a) {  
    sum += e; }  
}
```



```
total :: Num a => [a] -> a  
total a = foldl (+) 0 a
```



```
for i in range(n):  
    sum += a[i]
```



```
for (var e of a) {  
    sum += e; }  
}
```

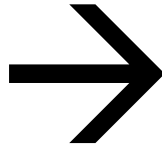


```
total :: Num a => [a] -> a  
total a = foldl (+) 0 a
```

0101010101

ADD R1, R2

# Programming...



```
for i in range(n):  
    sum += a[i]
```

## Assembly

```
append:
    push ebp
    mov ebp, esp
    push eax
    push ebx
    push len
    call malloc
    mov ebx, [ebp + 12]
    mov [eax + info], ebx
    mov dword [eax + next], 0
    mov ebx, [ebp + 8]
    cmp dword [ebx], 0
    je null_pointer
    mov ebx, [ebx]
```

```
next_element:
    cmp dword [ebx + next], 0
    je found_last
    mov ebx, [ebx + next]
    jmp next_element
```

```
found_last:
    push eax
    push addMes
    call puts
    add esp, 4
    pop eax
    mov [ebx + next], eax
```

```
go_out:
    pop ebx
    pop eax
    mov esp, ebp
    pop ebp
    ret 8
```

```
null_pointer:
    push eax
    push nullMes
    call puts
    add esp, 4
    pop eax
    mov [ebx], eax
    jmp go_out
```

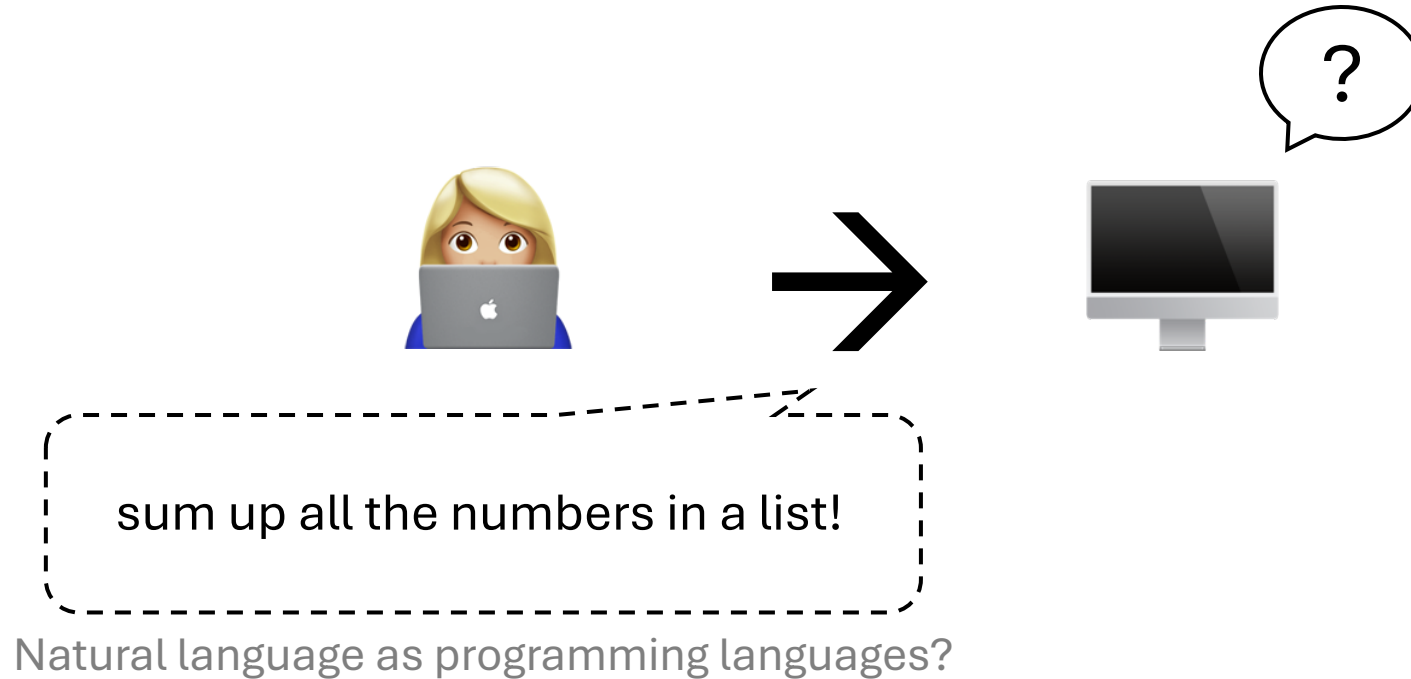
C

```
void insert(node *xs, int x) {
    node *new;
    node *temp;
    node *prev;
    new = (node *)malloc(sizeof(node));
    if(new == NULL) {
        printf("Insufficient memory.");
        return;
    }
    new->val = x;
    new->next = NULL;
    if (xs == NULL) {
        xs = new;
    } else if(x < xs->val) {
        new->next = xs;
        xs = new;
    } else {
        prev = xs;
        temp = xs->next;
        while(temp != NULL && x > temp->val) {
            prev = temp;
            temp = temp->next;
        }
        if(temp == NULL) {
            prev->next = new;
        } else {
            new->next = temp;
            prev->next = new;
        }
    }
}
```

Haskell

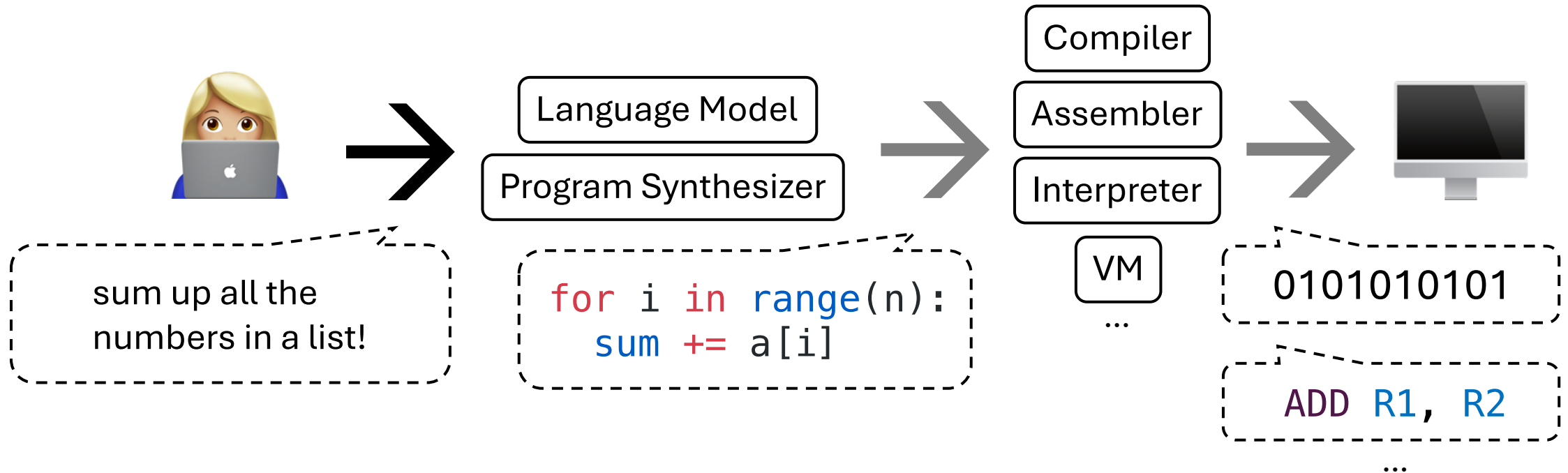
```
insert x [] = [x]
insert x (y:ys)
    | x ≤ y = x:y:ys
    | otherwise = y:(insert x ys)
```

# Programming → Talking to Computers

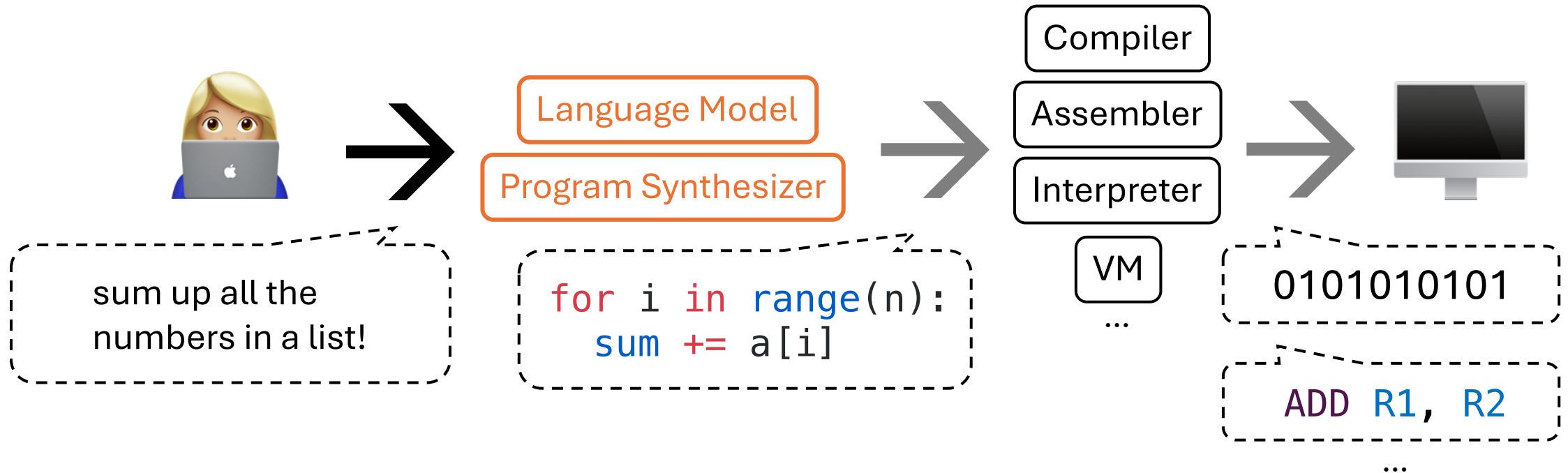


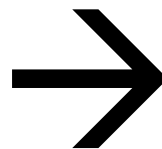


# Programming → Talking to Computers



# Programming → Talking to Computers





# Course Roadmap & Logistics

# Logistics

- Lecture
  - When: Tue/Thu 12:00 – 1:15pm
  - Where: Maryland Hall 310
- Office Hours
  - Instructor: Wed 3:00 – 4:00pm, Zoom (<https://wse.zoom.us/my/ziyang>)
  - TA: Tue 3:30 – 5:30pm, Malone 216
- Course Website
  - <https://machine-programming.github.io>
  - Discussions: [counselore](#)

# Roadmap

Foundations + Applications

of machine programming



# Topics

- Foundations

- **Programming language, syntax and semantics**
- **Classical synthesis**, e.g., inductive synthesis, bottom-up and top-down synthesis, type-guided synthesis, specification guided synthesis
- **LLM-based synthesis**, e.g., natural language guided synthesis, agentic frameworks, model context protocol, language server protocol

- Applications

- **Software Engineering & Security**, e.g., testing, transpilation, verification
- **Math & Theorem Proving**, e.g., auto-formalization, automated proving
- **Data Wrangling**, e.g., database querying, code querying
- **Planning & Cyber-Physical Systems**, e.g. robotics, simulation, reward

# Topics that YOU Can Explore

- Search algorithms
- Neurosymbolic method
- Feedback engineering
- Example guided synthesis
- Probabilistic method
- Human-in-the-loop synthesis / Vibe Coding

## 1. Synthesis methodologies

## 3. Synthesis applications

- General programming
- Data wrangling
- Program repair & optimization
- Verification & security
- Scientific domains & math
- Embodied AI & robotics
- Creative coding
- Visualization, games, and graphics

- MCP
- Reinforcement learning
- Agentic behavior
- Evaluation of coding capabilities
- Adversarial attack
- Prompting strategies
- Synthetic data generation
- Context engineering
- Fine-tuning strategies

## 2. Foundation models

## 4. Programming languages

- Syntax & semantics
- Type systems for synthesis
- Synthesis for domain-specific languages
- Synthesis for low-resource languages
- Language design
- Specification languages
- Language server protocol
- Safety properties

# Topics that YOU Can Explore

- Search algorithms
- Feedback engineering
- Example guided synthesis
- Human-in-the-loop synthesis / Vibe Coding
- Neurosymbolic method
- Probabilistic method

## 1. Synthesis methodologies

## 3. Synthesis applications

- General programming
- Program repair & optimization
- Verification & security
- Embodied AI & robotics
- Visualization, games, and graphics
- Data wrangling
- Scientific domains & math
- Creative coding

- MCP
- Evaluation of coding capabilities
- Prompting strategies
- Context engineering
- Reinforcement learning
- Adversarial attack
- Synthetic data generation
- Fine-tuning strategies
- Agentic behavior

## 2. Foundation models

## 4. Programming languages

- Syntax & semantics
- Synthesis for domain-specific languages
- Synthesis for low-resource languages
- Language design
- Language server protocol
- Type systems for synthesis
- Specification languages
- Safety properties

# Topics that YOU Can Explore

- Search algorithms
- Feedback engineering
- Example guided synthesis
- Human-in-the-loop synthesis / Vibe Coding
- Neurosymbolic method
- Probabilistic method

## 1. Synthesis methodologies

- General programming
- Program repair & optimization
- Verification & security
- Embodied AI & robotics
- Visualization, games, and graphics
- Data wrangling
- Scientific domains & math
- Creative coding

## 3. Synthesis applications

- MCP
- Evaluation of coding capabilities
- Prompting strategies
- Context engineering
- Reinforcement learning
- Adversarial attack
- Synthetic data generation
- Fine-tuning strategies
- Agentic behavior

## 2. Foundation models

## 4. Programming languages

- Syntax & semantics
- Synthesis for domain-specific languages
- Synthesis for low-resource languages
- Language design
- Language server protocol
- Type systems for synthesis
- Specification languages
- Safety properties

# Topics that YOU Can Explore

- Search algorithms
- Neurosymbolic method
- Feedback engineering
- Example guided synthesis
- Probabilistic method
- Human-in-the-loop synthesis / Vibe Coding

## 1. Synthesis methodologies

## 3. Synthesis applications

- General programming
- Data wrangling
- Program repair & optimization
- Verification & security
- Scientific domains & math
- Embodied AI & robotics
- Creative coding
- Visualization, games, and graphics

- MCP
- Reinforcement learning
- Agentive behavior
- Evaluation of coding capabilities
- Adversarial attack
- Prompting strategies
- Synthetic data generation
- Context engineering
- Fine-tuning strategies

## 2. Foundation models

## 4. Programming languages

- Syntax & semantics
- Type systems for synthesis
- Synthesis for domain-specific languages
- Synthesis for low-resource languages
- Language design
- Specification languages
- Language server protocol
- Safety properties

# Topics that YOU Can Explore

- Search algorithms
- Neurosymbolic method
- Feedback engineering
- Example guided synthesis
- Probabilistic method
- Human-in-the-loop synthesis / Vibe Coding

## 1. Synthesis methodologies

- General programming
- Data wrangling
- Program repair & optimization
- Verification & security
- Scientific domains & math
- Embodied AI & robotics
- Creative coding
- Visualization, games, and graphics

## 3. Synthesis applications

- MCP
- Reinforcement learning
- Agentic behavior
- Evaluation of coding capabilities
- Adversarial attack
- Prompting strategies
- Synthetic data generation
- Context engineering
- Fine-tuning strategies

## 2. Foundation models

## 4. Programming languages

- Syntax & semantics
- Type systems for synthesis
- Synthesis for domain-specific languages
- Synthesis for low-resource languages
- Language design
- Specification languages
- Language server protocol
- Safety properties

# Programming Languages

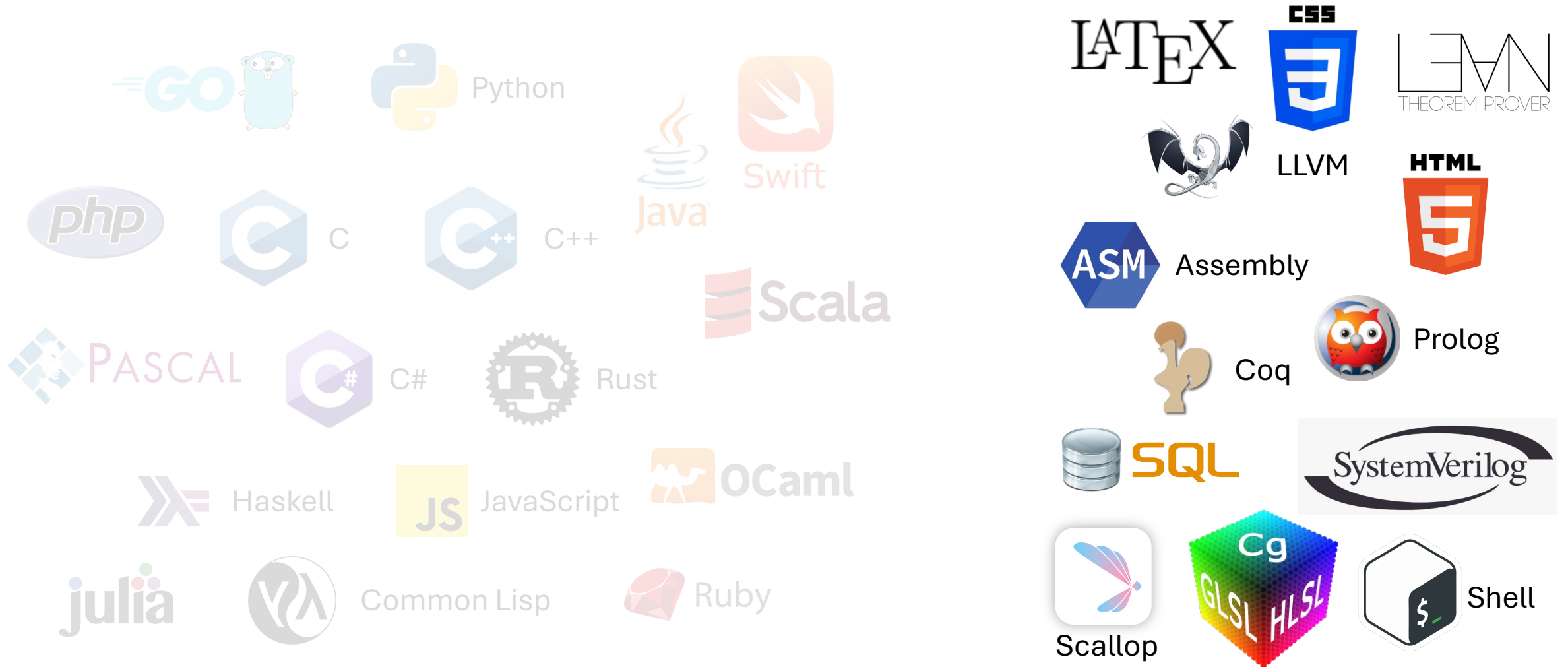


# General Purpose Programming Languages





# Domain Specific Programming Languages



# Grading

- Attendance (10%)
- Assignment 1: Inductive synthesis (15%)
- Assignment 2: Evaluating coding LLMs (15%)
- Assignment 3: Coding agents (15%)
- Oral presentation (10%)
- Final Project (35%)

# Assignments (45% total, 15% each)

- There will be **three assignments**:
  - Assignment 1: Inductive Synthesis & Basics of LLM synthesis
  - Assignment 2: Evaluating Coding LLMs
  - Assignment 3: Coding Agents
- Timeline: **~2 weeks** for each assignment
- Submission: via **GradeScope**; submit .zip files
- Late submission: up to 3 days, -20% per day
- Extensions: possible, please request via email

# Assignments (45% total, 15% each)

- Collaboration policy:
  - **Encouraged**, please acknowledge your collaborator
  - Still need to be your own work
- Use of AI:
  - **Encouraged**, please acknowledge your collaborator AI
  - Be specific about the model, version, programming environment, etc.
  - Document important conversations, learn about its good or bad

# Assignments (45% total, 15% each)

- **API Key** for LLM use

- Assignments will include code for you to invoke LLMs, which require API keys
- We are going to supply an API key for Google's Gemini Model
  - The key will be sent privately to each of you via email
- Do not share API key with others, even your collaborators
- We keep track of API usage (#requests, #tokens, \$, etc.)
- Overused API key will be revoked

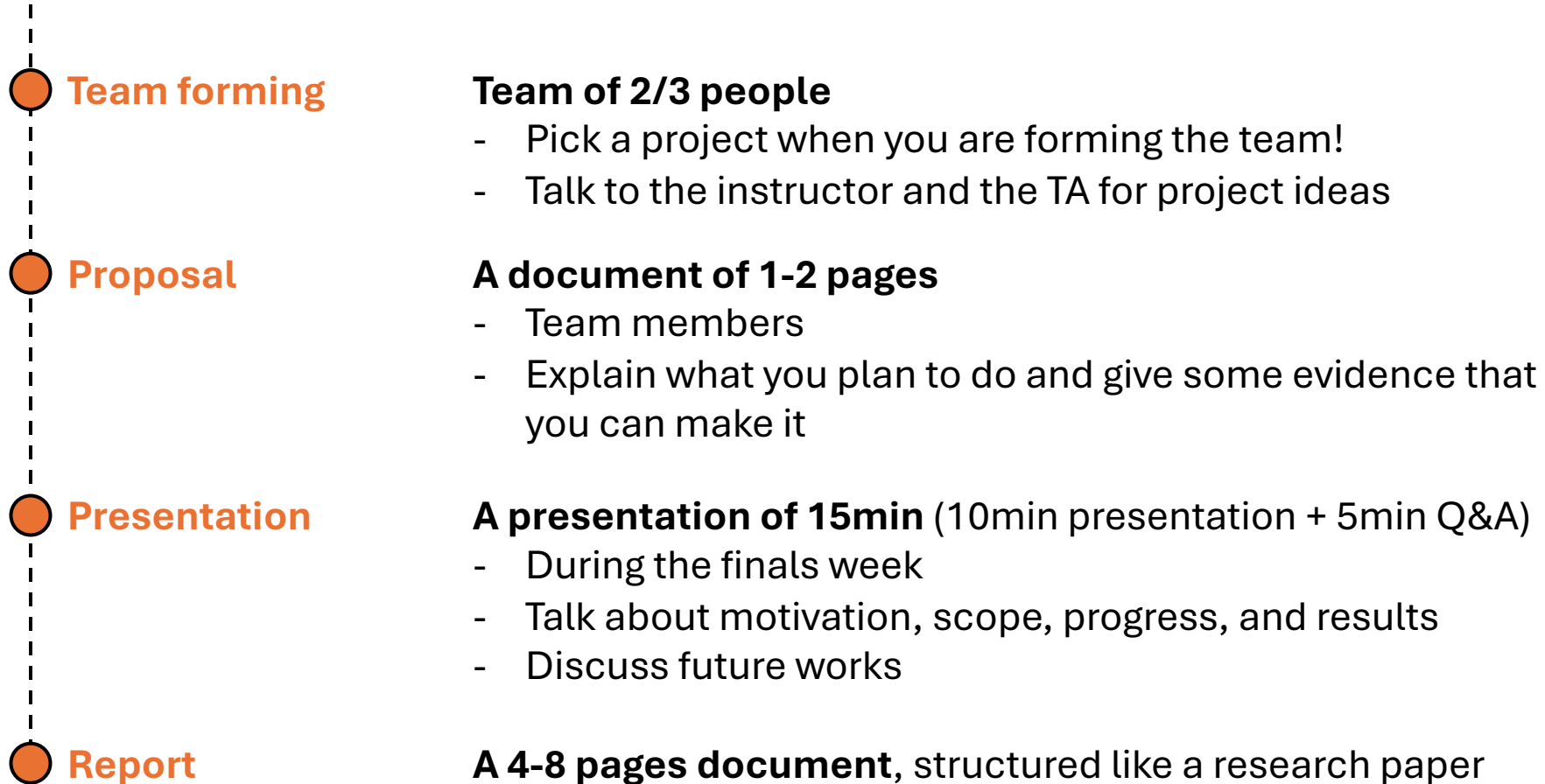
# Oral Presentation (10%)

- Happening on the latter half of the semester
- Two students will be paired to explore the direction of that week
  - Each one will read a paper within the direction
  - The paper could be from the list or proposed by yourself (talk to me!)
  - Lead a 25min in-class discussion; 15-20min presentation & 5-10min Q&A
- A presentation sign-up form will be sent out later

# Final Project

- On any aforementioned topics
  - Re-implement a technique from a paper
  - Applying existing synthesis framework to a new domain
  - Extend/improve existing synthesis algorithm or tool
  - Develop a new synthesis algorithm or tool
  - New dataset, new benchmark, or a novel evaluation
  - ...
- Judged in terms of
  - Quality of execution
  - Originality
  - Scope

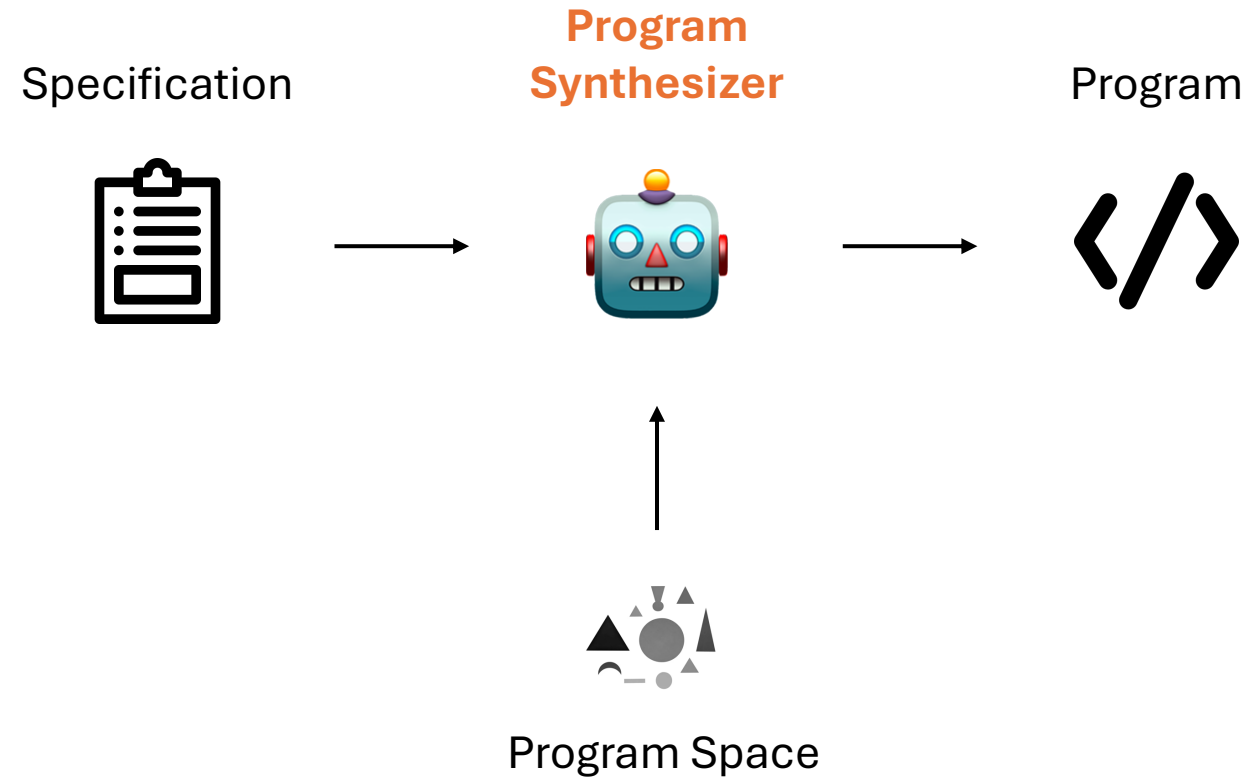
# Final Project



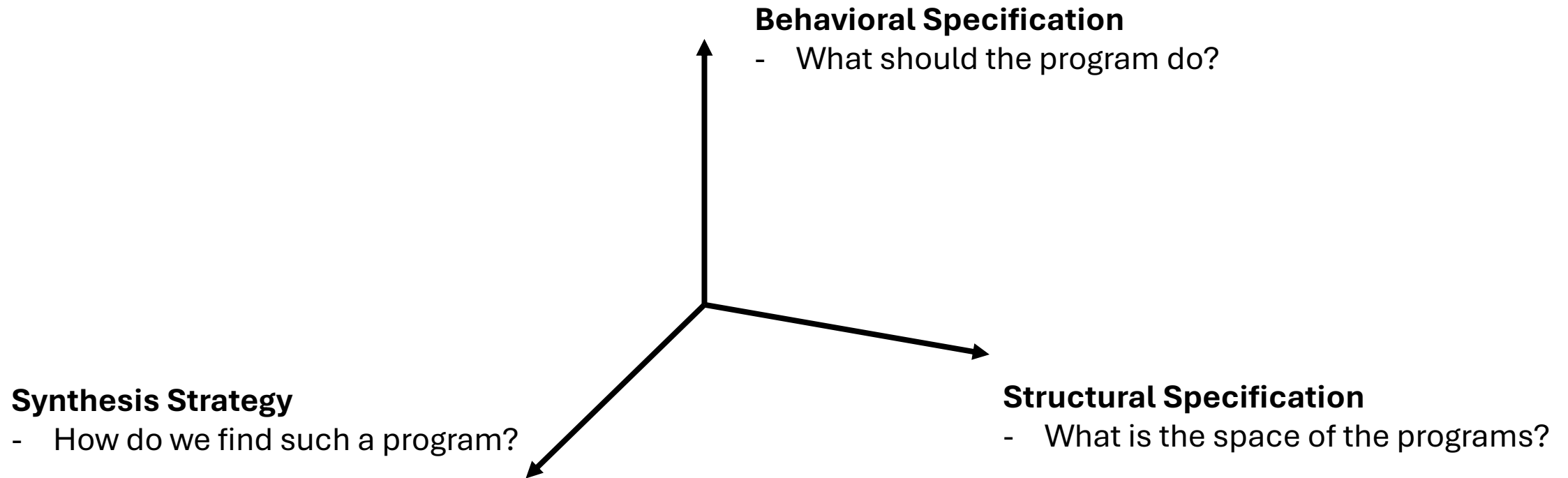


# Program Synthesis

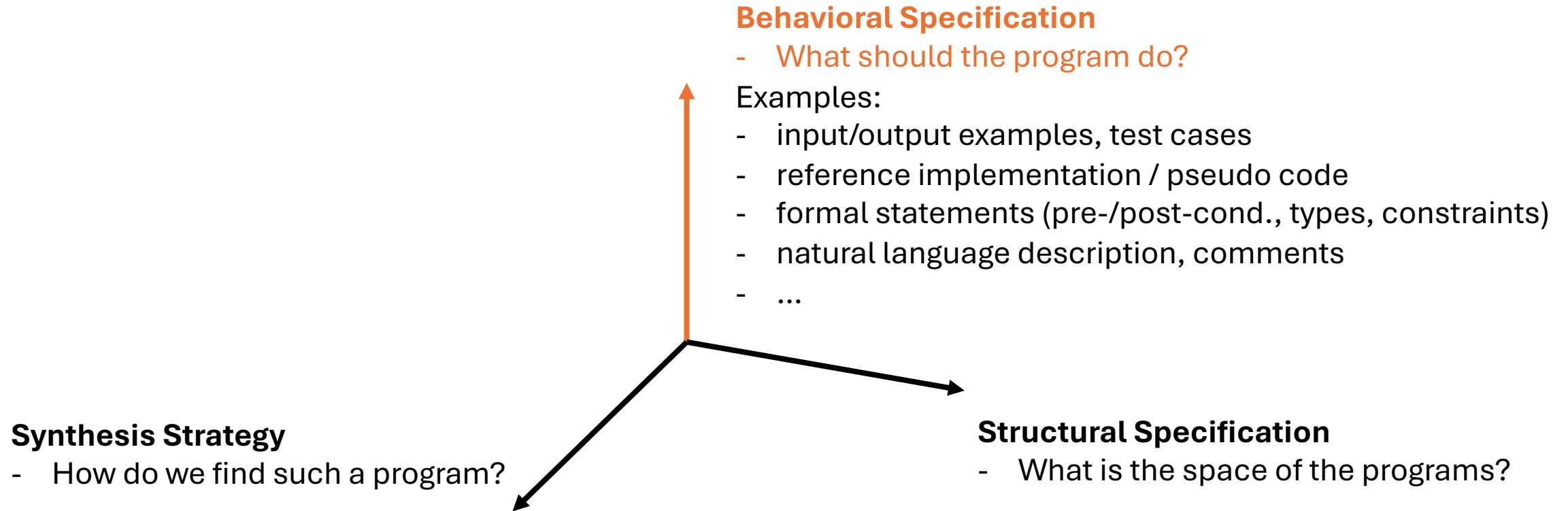
# Program Synthesizers



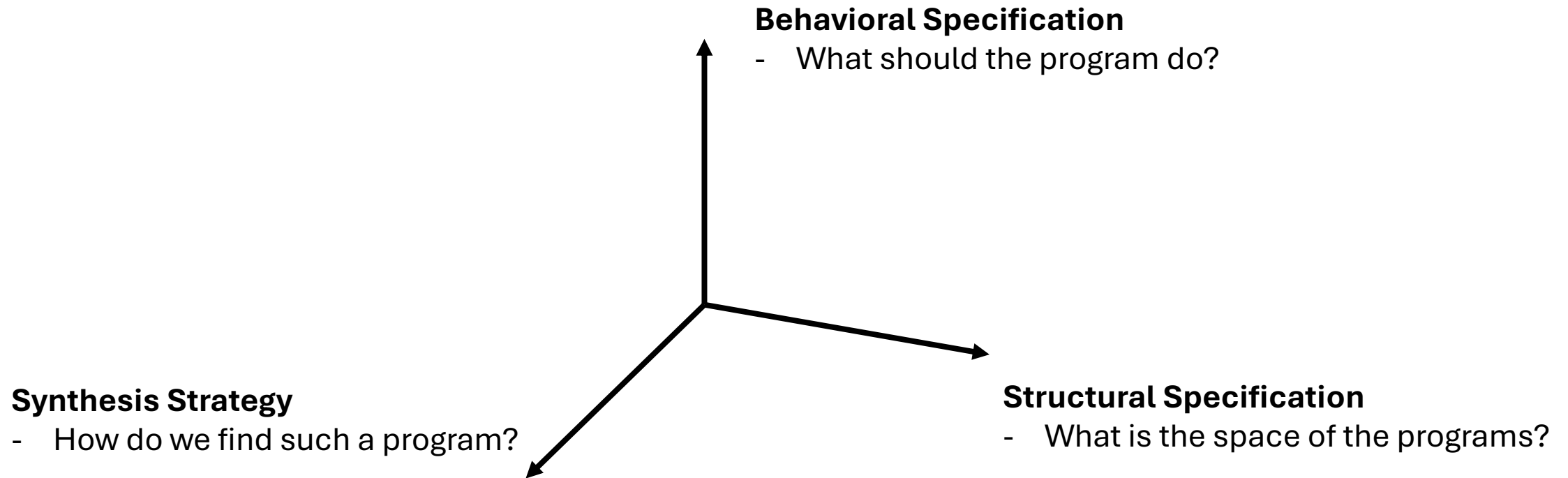
# Dimensions in Program Synthesis



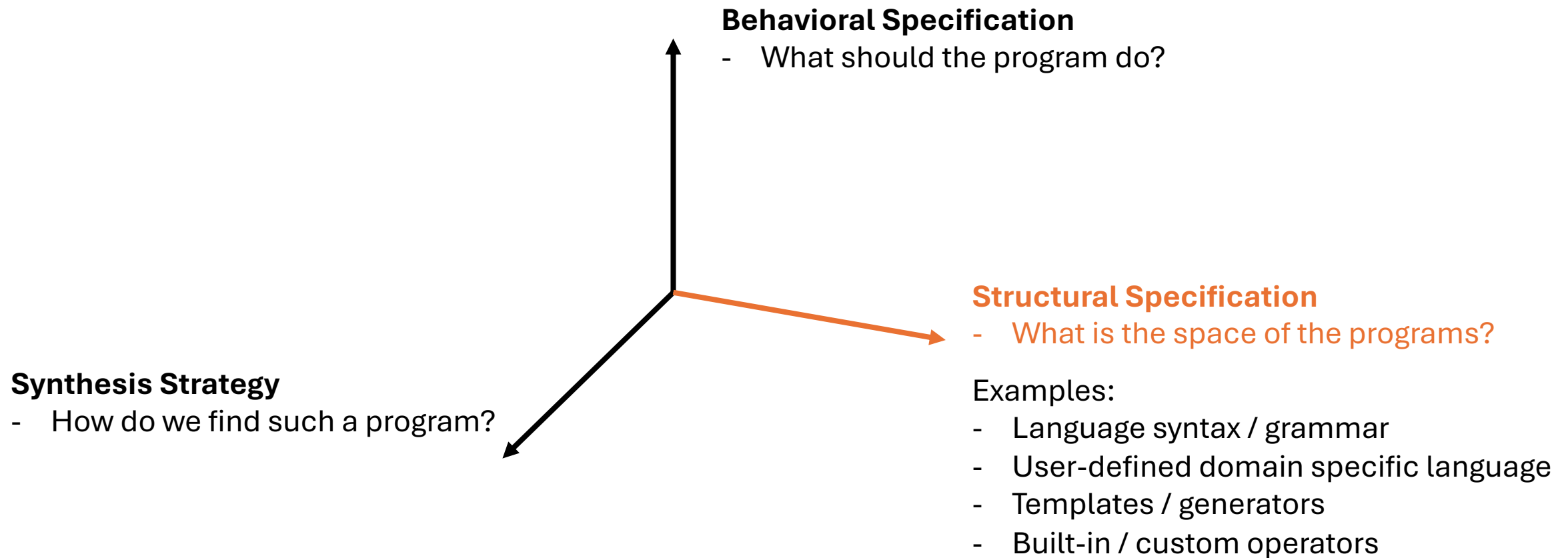
# Dimensions in Program Synthesis



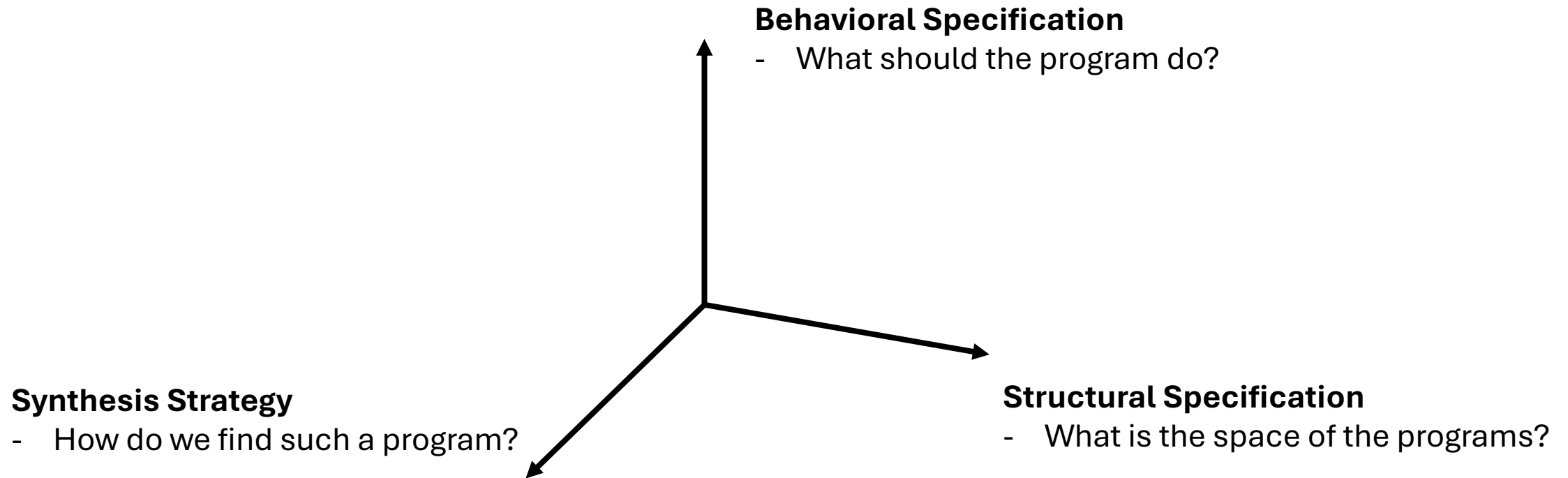
# Dimensions in Program Synthesis



# Dimensions in Program Synthesis



# Dimensions in Program Synthesis



# Dimensions in Program Synthesis

## Behavioral Specification

- What should the program do?

## Synthesis Strategy

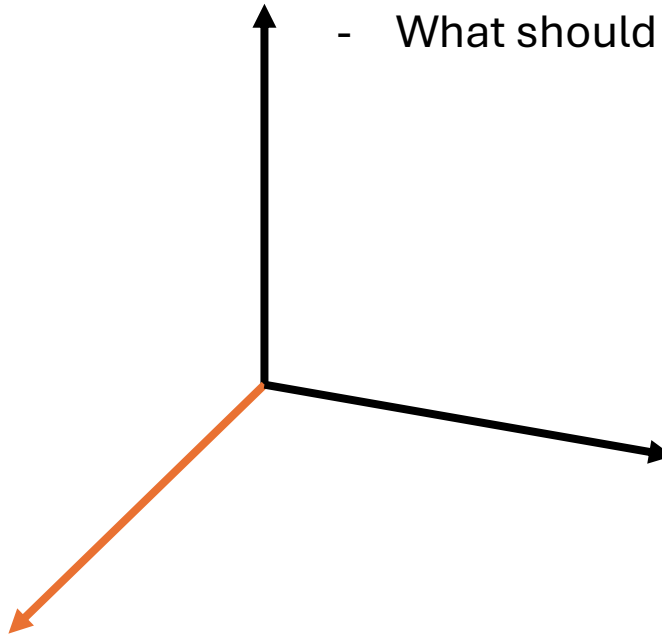
- How do we find such a program?

Examples:

- Bottom-up / top-down search
- Stochastic / constraint-based search
- LLM zero-shot / few-shot prompting
- Chain-of-thought reasoning
- Iterative / agentic self-refinement

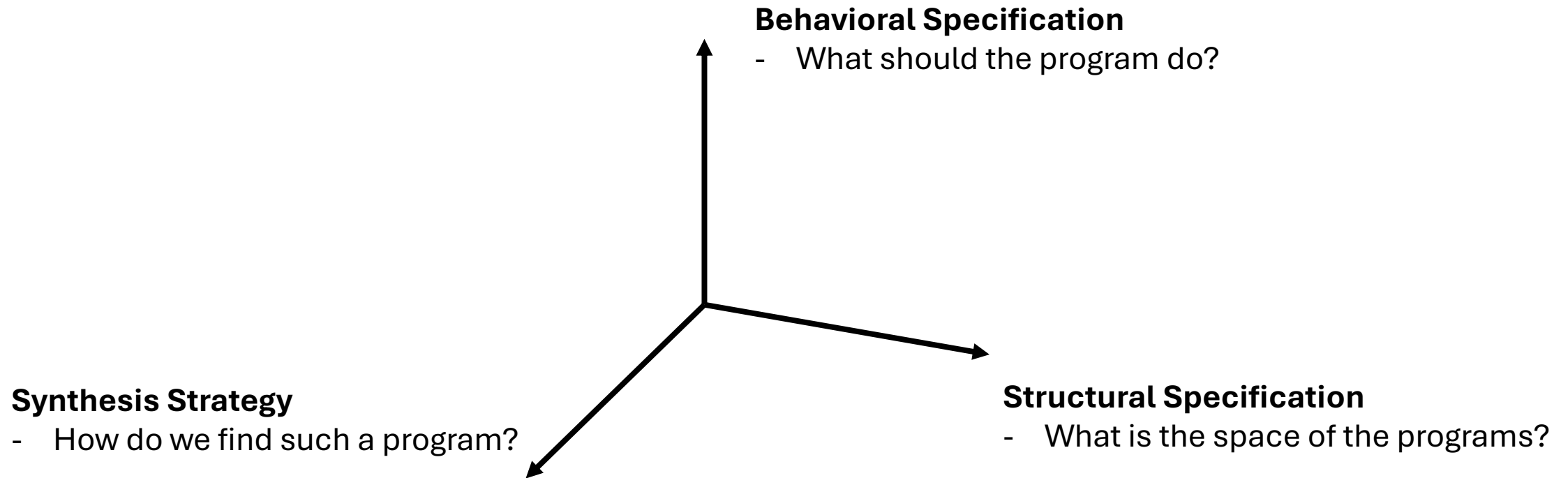
## Structural Specification

- What is the space of the programs?





# Dimensions in Program Synthesis



# Structure of the course

- **Module 1: Foundations of Program Synthesis**
  - Programming language syntax and semantics
  - Classical methods for synthesizing programs
- **Module 2: Program Synthesis in the era of Foundation Models**
  - Prompting strategies, evaluating coding LLMs
  - Iterative synthesis, MCP and tool use, agentic program synthesis
- **Module 3: Applications of Program Synthesis**
  - Software engineering, theorem proving, planning, interactive
  - Other advanced topics

# Week 1

- Topic:
  - Programming language syntax & semantics
  - Bottom-up enumerative synthesis from examples
- Assignment 1:
  - Released: <https://github.com/machine-programming/assignment-1>
  - API keys will be sent shortly
  - Submission on GradeScope will be opened next Tuesday