Principles of Programming Languages

Lecture 1 – Overview and Introduction to OCaml

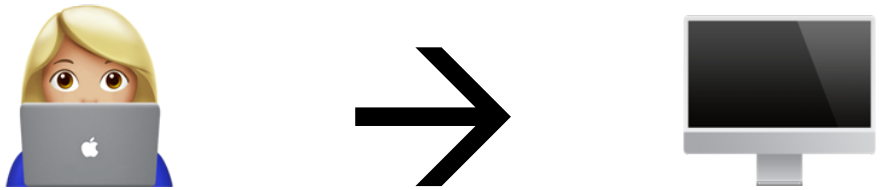
Ziyang Li

Instructor

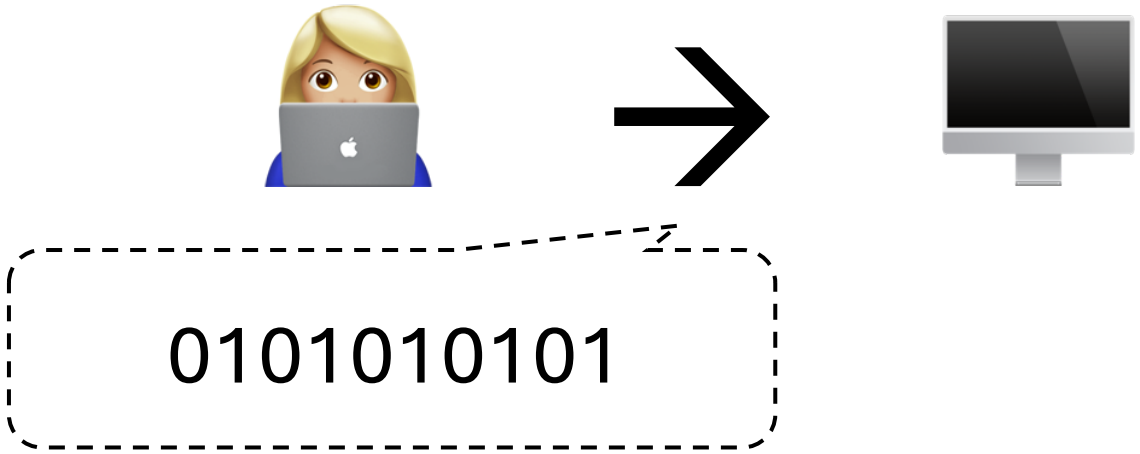
Ziyang Li

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

Programming...

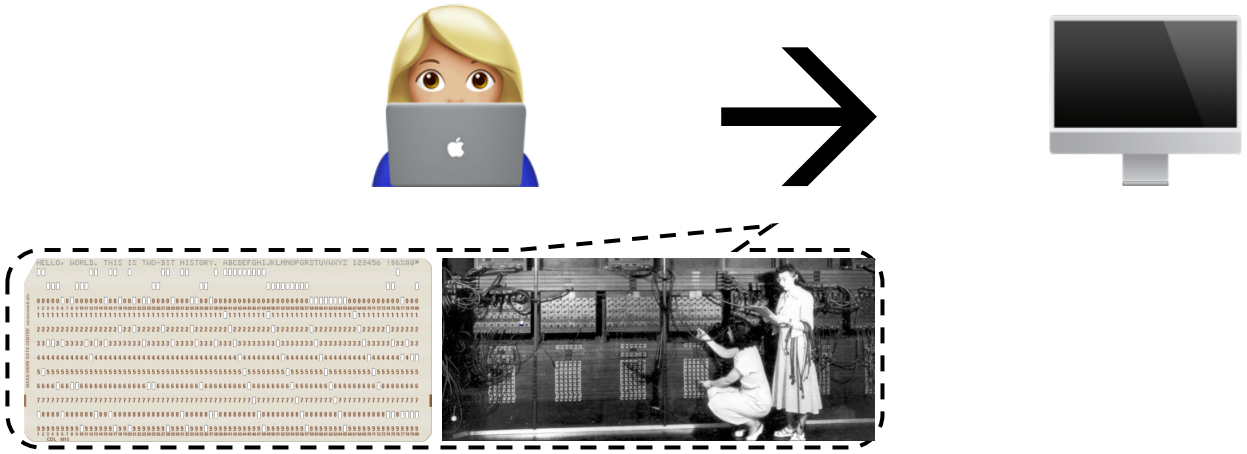


Programming...

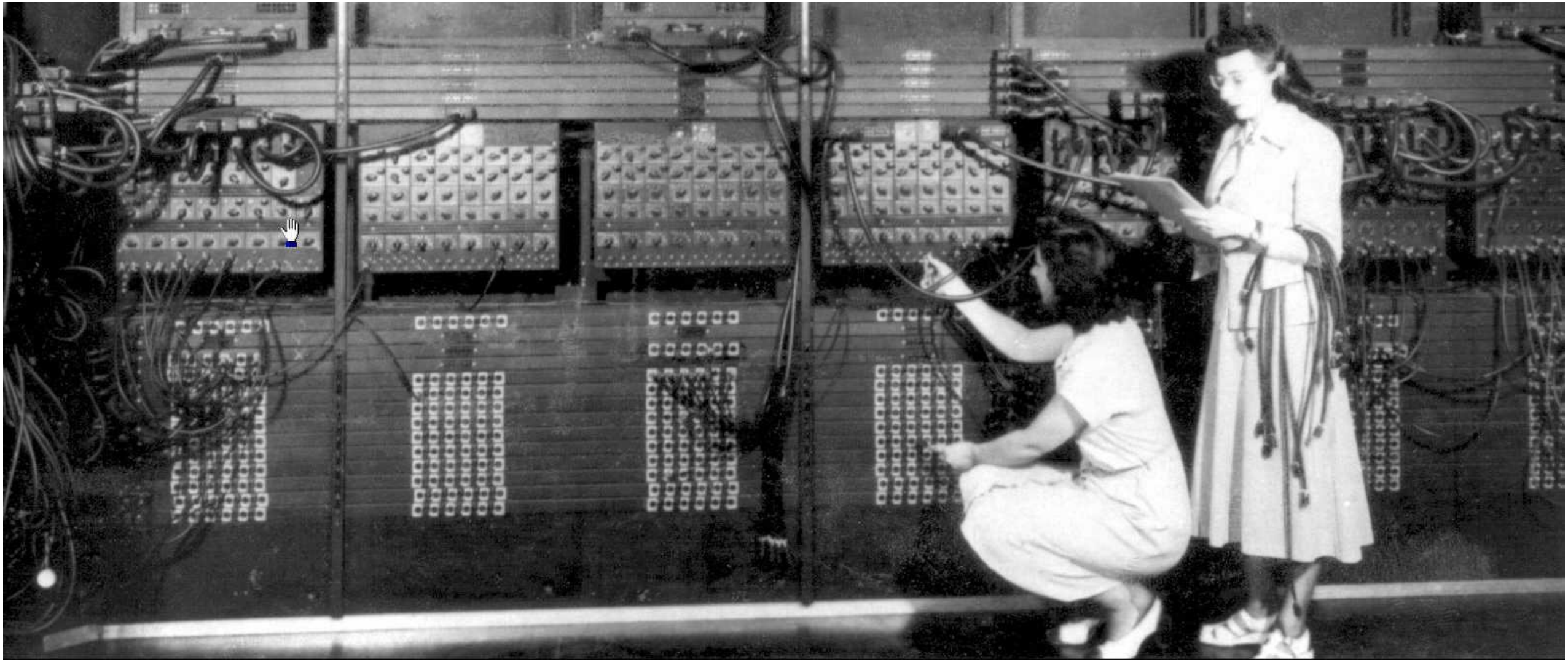


1940s–1950s: Binary

Programming…

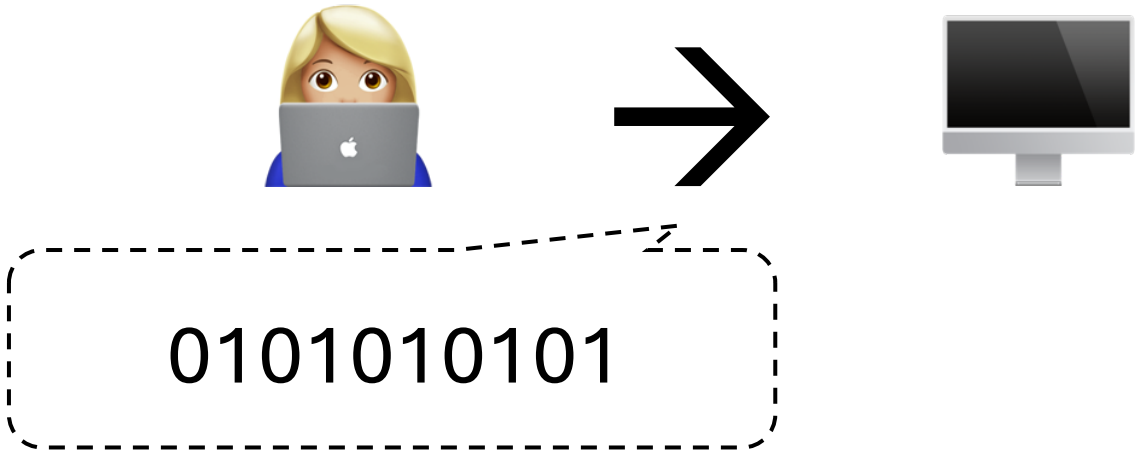


1940s–1950s



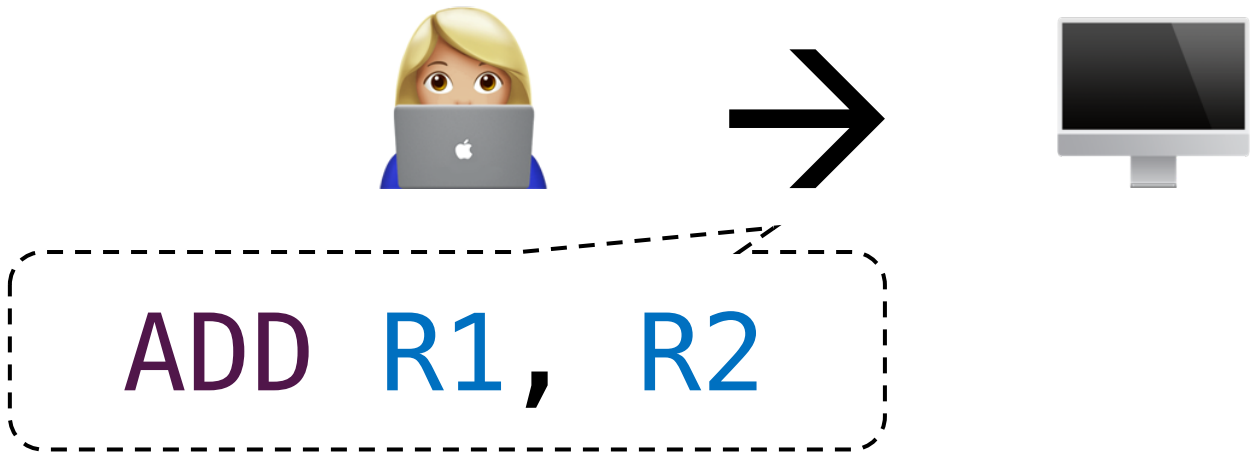

HELLO, WORLD. THIS IS TWO-BIT HISTORY. ABCDEFGHIJKLMNOPQRSTUVWXYZ 123456 !$&%#@*
MASS:WERK DATA CENTER <masswerk.at>
CDL 0815

Programming...

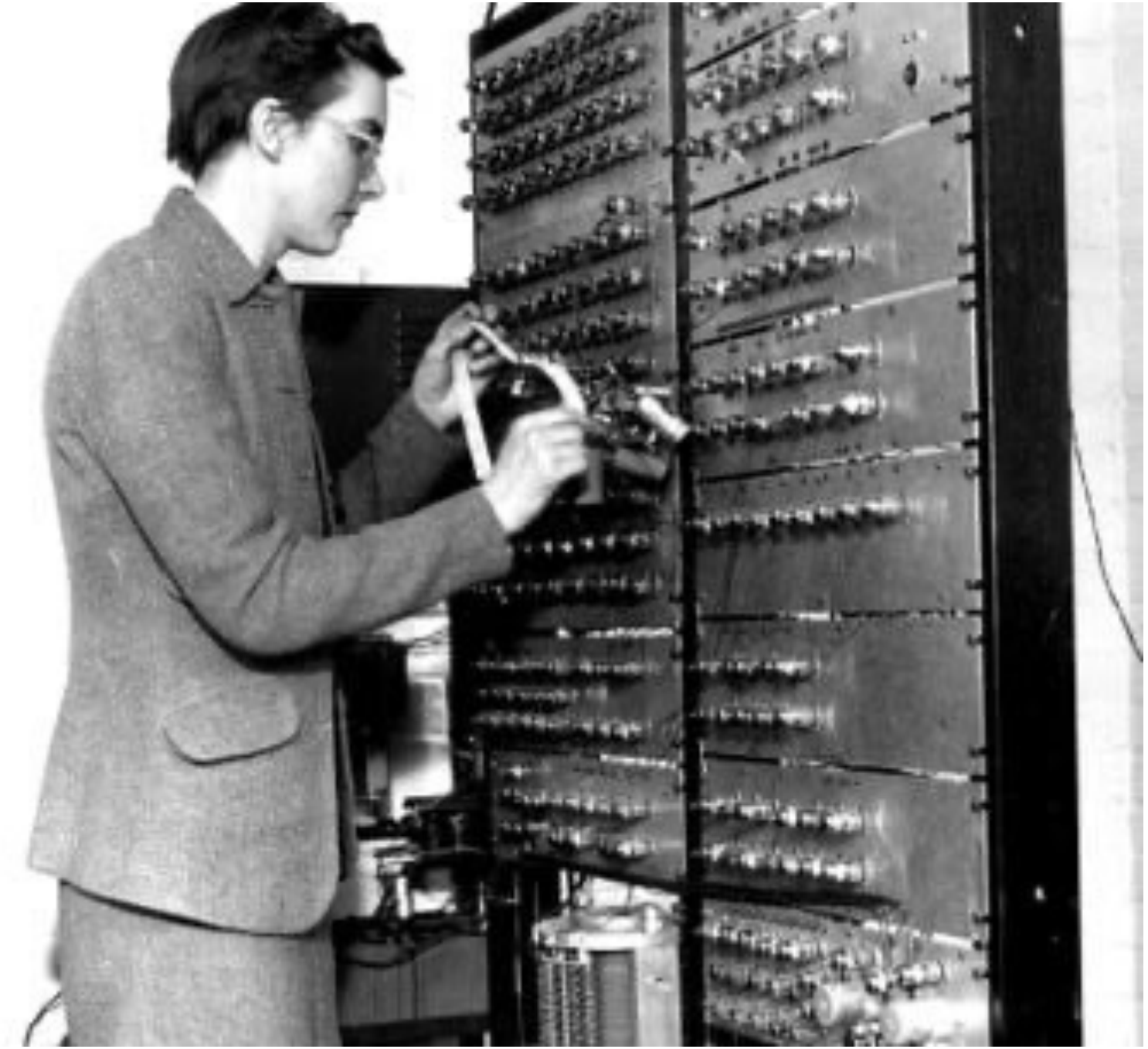


1940s–1950s: Binary

Programming…



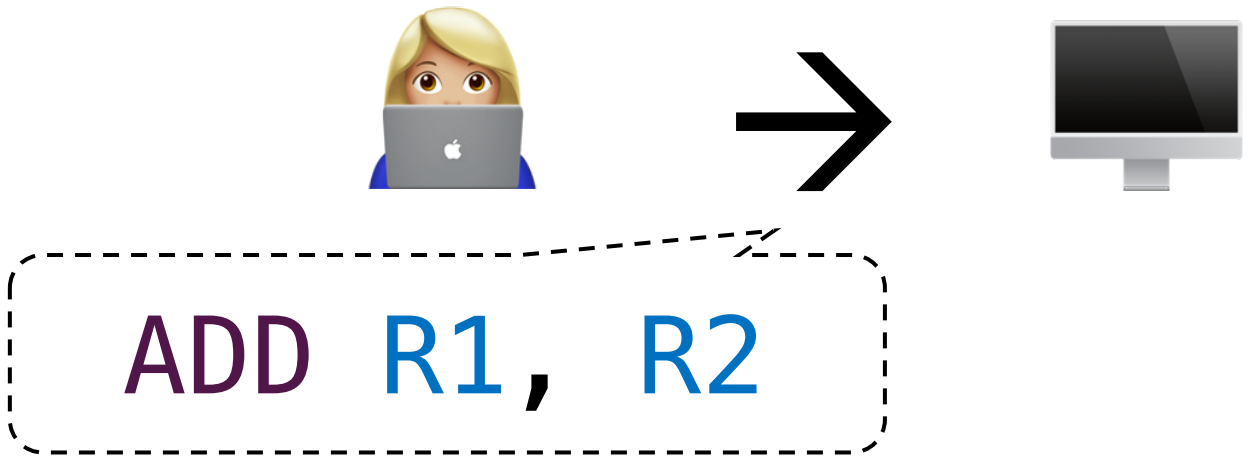
1950s–1960s: Assembly



Kathleen Booth

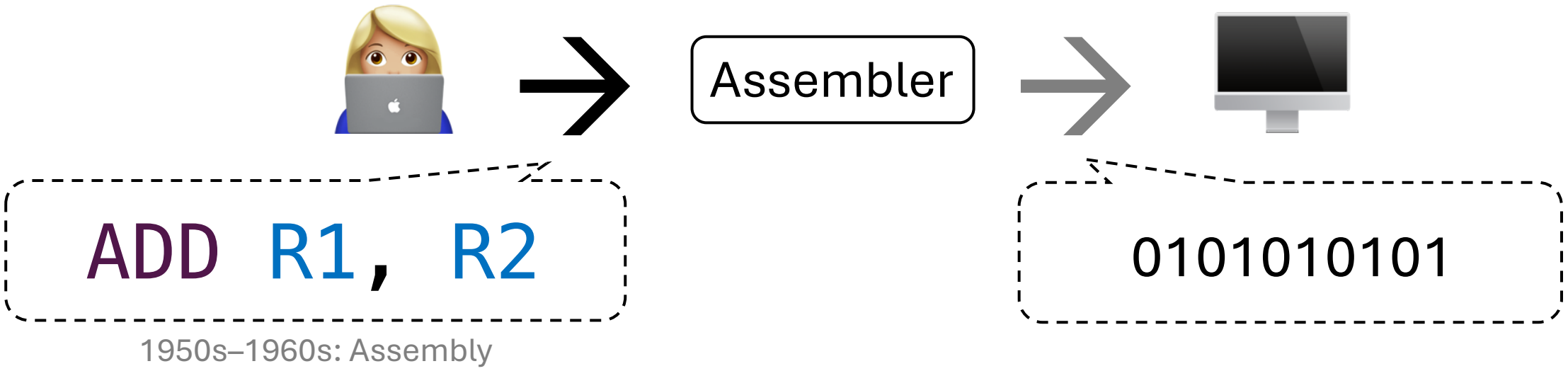
1922–2022

Programming…



1950s–1960s: Assembly

Programming...



1950s–1960s: Assembly

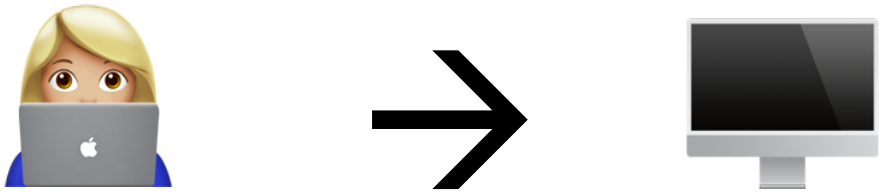
The FORTRAN Automatic Coding System

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

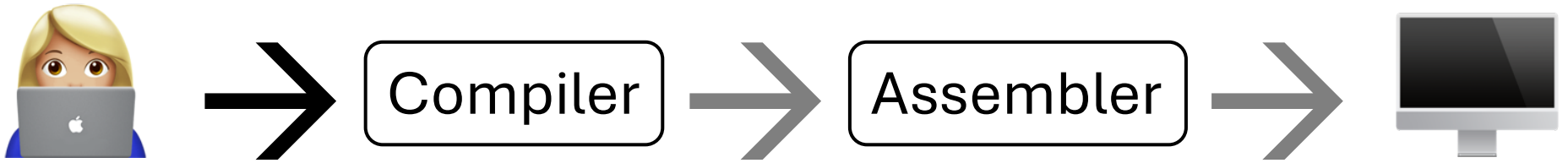
Programming…



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

1970s: High-level Language

Programming…




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

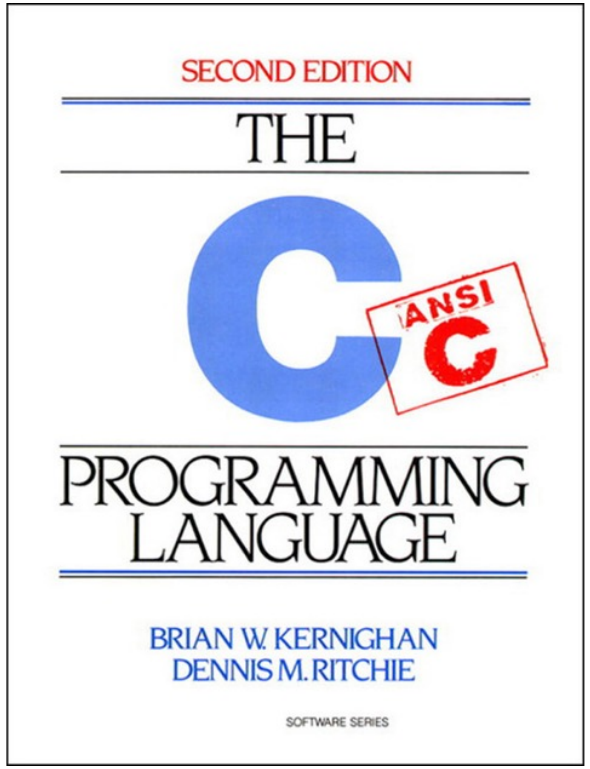
```
ADD R1, R2
```

```
0101010101
```

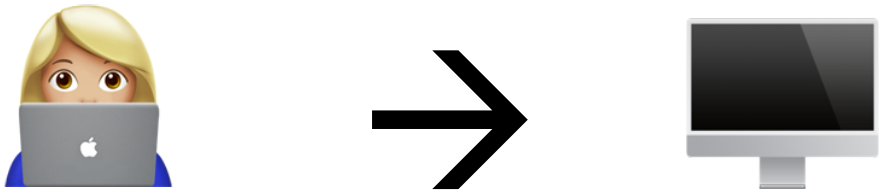
1970s: High-level Language

Dennis M. Ritchie

1941–2011



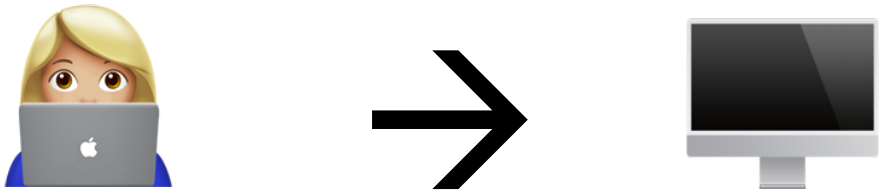
Programming…



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

1970s: High-level Language

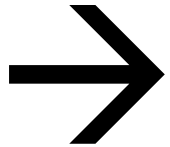
Programming...



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

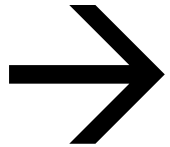
1980s onwards: Modern High-Level Languages

Programming...




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

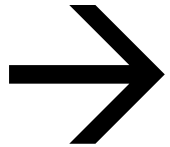
Programming...




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


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

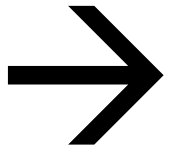
Programming...




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


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

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



Compiler

Assembler

Interpreter

VM

...

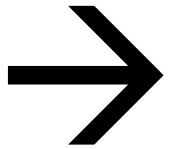
python

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

JS

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

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



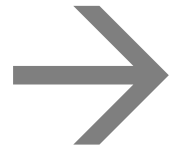
Compiler

Assembler

Interpreter

VM

...



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

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

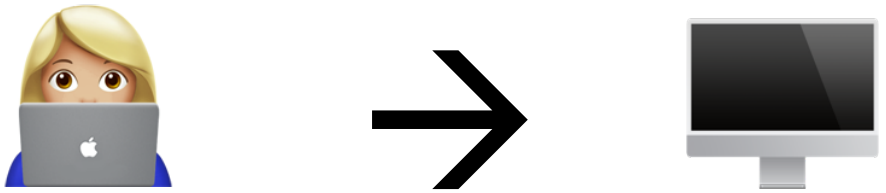
```
total :: Num a => [a] -> a
total a = foldl (+) 0 a
```

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
```

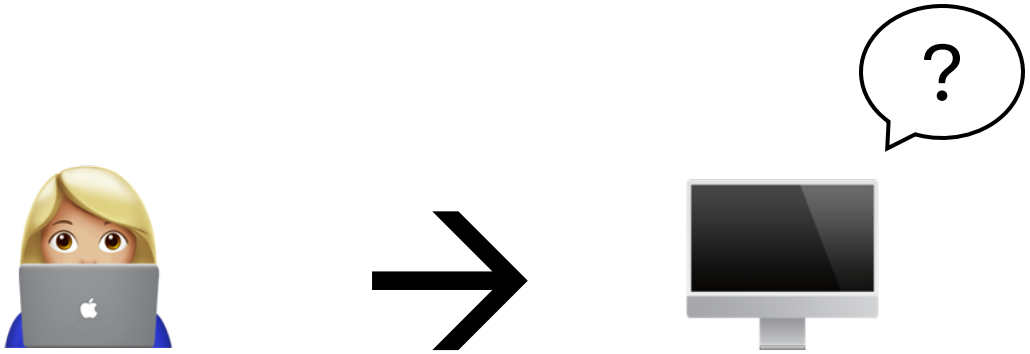
C

```
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;
    }
  }
}
```

OCaml

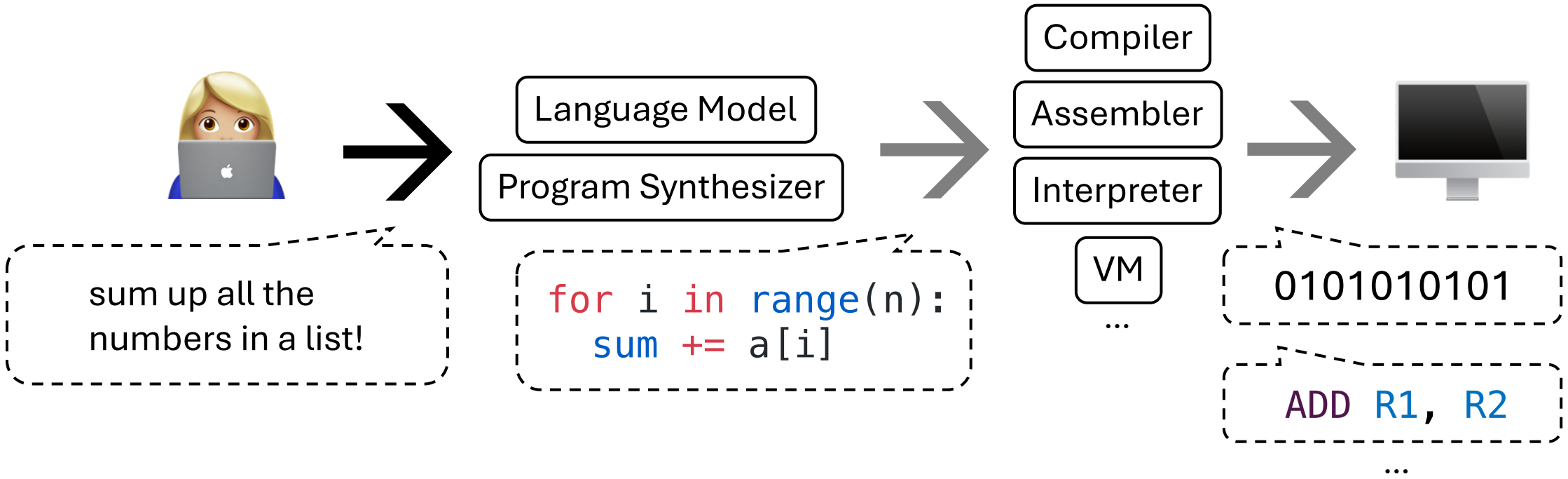
```
let rec insert x ys =
  match ys with
  | [] -> [x]
  | y :: rs ->
    if x <= y then x :: y :: rs
    else y :: (insert x rs)
```

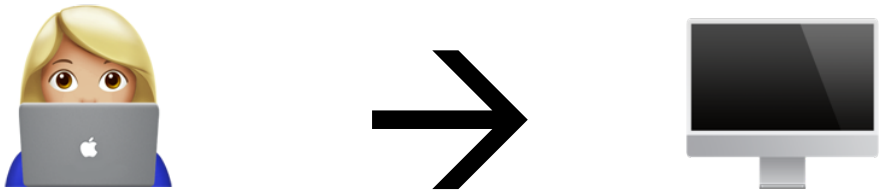
Programming → Talking to Computers



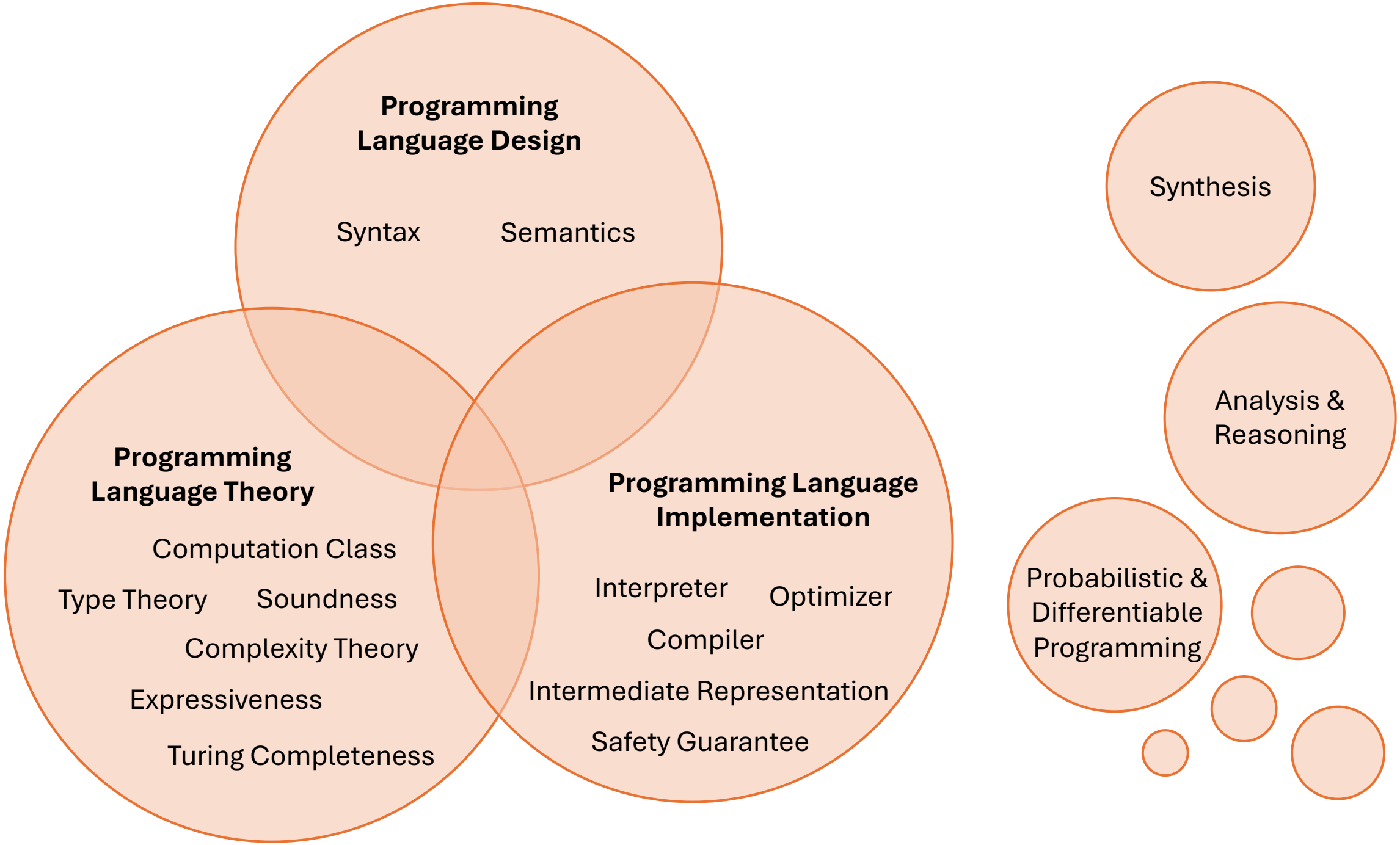
Natural language as programming languages?

Programming → Talking to Computers





Principles of Programming Languages


Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Safety Guarantee
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming

Programming Language Design

Syntax Semantics

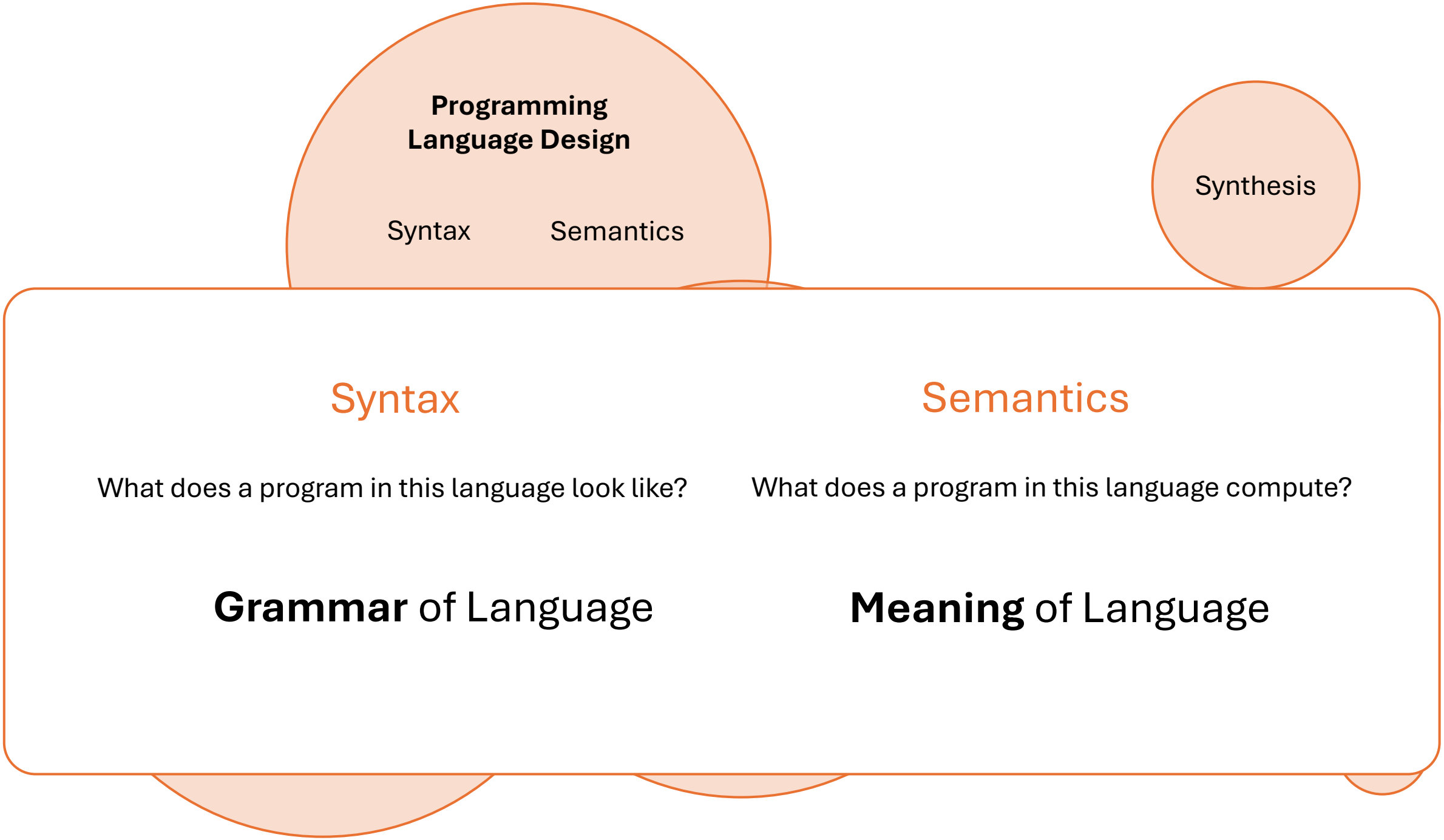
Synthesis

Syntax

What does a program in this language look like?

Semantics

What does a program in this language compute?


Programming
Language Design
Syntax
Semantics
Synthesis
Syntax
What does a program in this language look like?
Grammar of Language
Semantics
What does a program in this language compute?
Meaning of Language

```
if_stmt:
    | 'if' named_expression ':' block elif_stmt
    | 'if' named_expression ':' block [else_block]
elif_stmt:
    | 'elif' named_expression ':' block elif_stmt
    | 'elif' named_expression ':' block [else_block]
else_block:
    | 'else' ':' block

# While statement
# ---------------

while_stmt:
    | 'while' named_expression ':' block [else_block]

# For statement
# -------------

for_stmt:
    | 'for' star_targets 'in' ~ star_expressions ':' [TYPE_COMMENT] block [else_block]
    | 'async' 'for' star_targets 'in' ~ star_expressions ':' [TYPE_COMMENT] block [el

# With statement
# --------------

with_stmt:
    | 'with' '(' ','.with_item+ ','? ')' ':' [TYPE_COMMENT] block
    | 'with' ','.with_item+ ':' [TYPE_COMMENT] block
    | 'async' 'with' '(' ','.with_item+ ','? ')' ':' block
    | 'async' 'with' ','.with_item+ ':' [TYPE_COMMENT] block

with_item:
    | expression 'as' star_target &(',' | ')' | ':')
    | expression

# Try statement
# -------------

try_stmt:
    | 'try' ':' block finally_block
    | 'try' ':' block except_block+ [else_block] [finally_block]
    | 'try' ':' block except_star_block+ [else_block] [finally_block]
```

What language is this?

Concrete and Abstract Syntax

Python Grammar (Concrete Syntax)

```
if_stmt:
    | 'if' named_expression ':' block elif_stmt
    | 'if' named_expression ':' block [else_block]
elif_stmt:
    | 'elif' named_expression ':' block elif_stmt
    | 'elif' named_expression ':' block [else_block]
else_block:
    | 'else' ':' block

# While statement
# ---------------

while_stmt:
    | 'while' named_expression ':' block [else_block]

# For statement
# -------------

for_stmt:
    | 'for' star_targets 'in' ~ star_expressions ':' [TYPE_COMMENT] block [else_block
    | 'async' 'for' star_targets 'in' ~ star_expressions ':' [TYPE_COMMENT] block [el

# With statement
# --------------

with_stmt:
    | 'with' '(' ','.with_item+ ','? ')' ':' [TYPE_COMMENT] block
    | 'with' ','.with_item+ ':' [TYPE_COMMENT] block
    | 'async' 'with' '(' ','.with_item+ ','? ')' ':' block
    | 'async' 'with' ','.with_item+ ':' [TYPE_COMMENT] block

with_item:
    | expression 'as' star_target &(',' | ')' | ':')
    | expression

# Try statement
# -------------

try_stmt:
    | 'try' ':' block finally_block
    | 'try' ':' block except_block+ [else_block] [finally_block]
    | 'try' ':' block except_star_block+ [else_block] [finally_block]
```

https://docs.python.org/3/reference/grammar.html

Python Abstract Syntax

```
      -- use 'orelse' because else is a keyword in target languages
      | For(expr target, expr iter, stmt* body, stmt* orelse, string?
      | AsyncFor(expr target, expr iter, stmt* body, stmt* orelse, st
      | While(expr test, stmt* body, stmt* orelse)
      | If(expr test, stmt* body, stmt* orelse)
      | With(withitem* items, stmt* body, string? type_comment)
      | AsyncWith(withitem* items, stmt* body, string? type_comment)

      | Match(expr subject, match_case* cases)

      | Raise(expr? exc, expr? cause)
      | Try(stmt* body, excepthandler* handlers, stmt* orelse, stmt*
      | TryStar(stmt* body, excepthandler* handlers, stmt* orelse, st
      | Assert(expr test, expr? msg)

      | Import(alias* names)
      | ImportFrom(identifier? module, alias* names, int? level)

      | Global(identifier* names)
      | Nonlocal(identifier* names)
      | Expr(expr value)
      | Pass | Break | Continue

      -- col_offset is the byte offset in the utf8 string the parser
      attributes (int lineno, int col_offset, int? end_lineno, int? e

      -- BoolOp() can use left & right?
expr = BoolOp(boolop op, expr* values)
     | NamedExpr(expr target, expr value)
     | BinOp(expr left, operator op, expr right)
     | UnaryOp(unaryop op, expr operand)
     | Lambda(arguments args, expr body)
     | IfExp(expr test, expr body, expr orelse)
     | Dict(expr* keys, expr* values)
     | Set(expr* elts)
     | ListComp(expr elt, comprehension* generators)
```

https://docs.python.org/3/library/ast.html

Formal Semantics

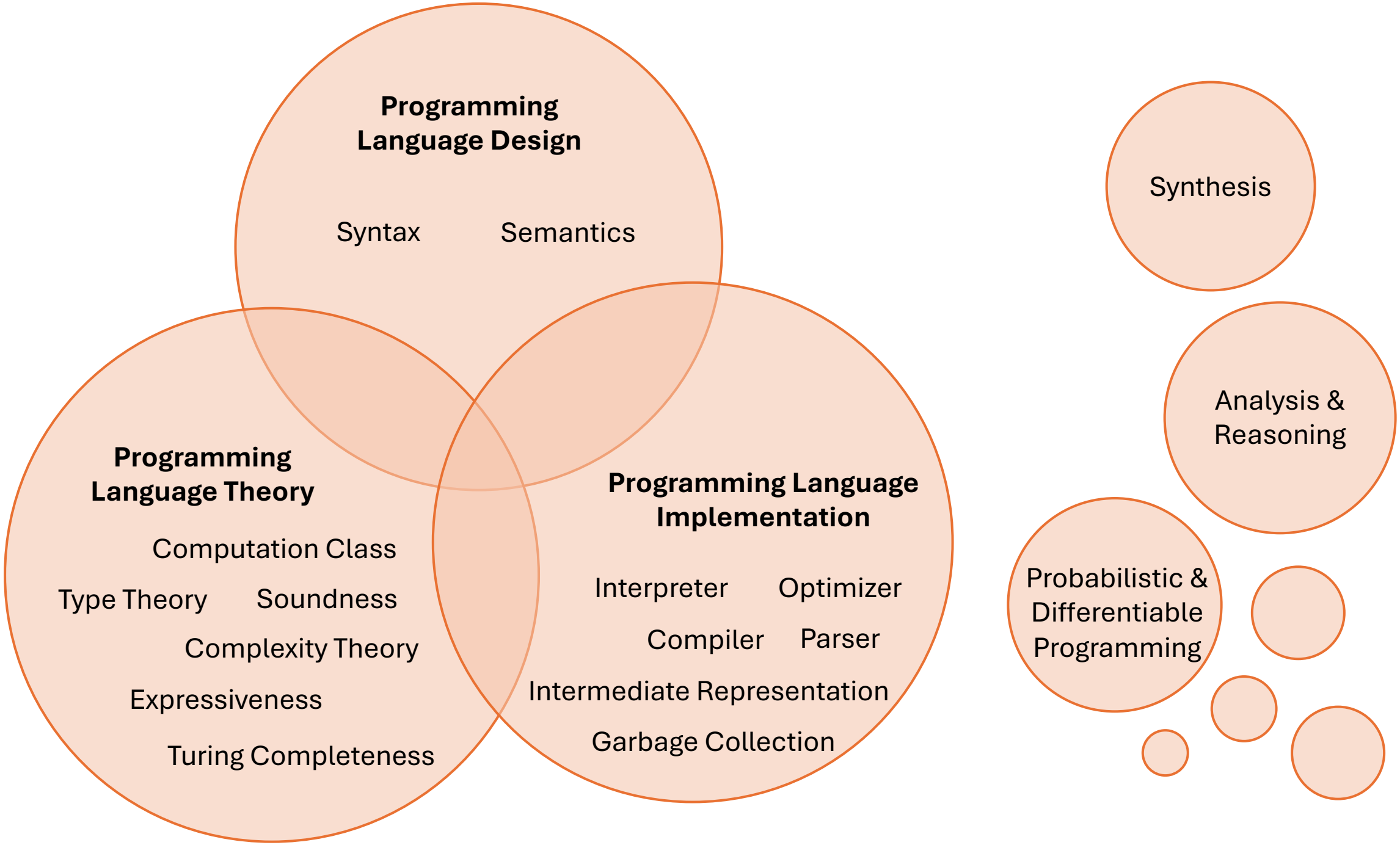
WebAssembly (WASM)

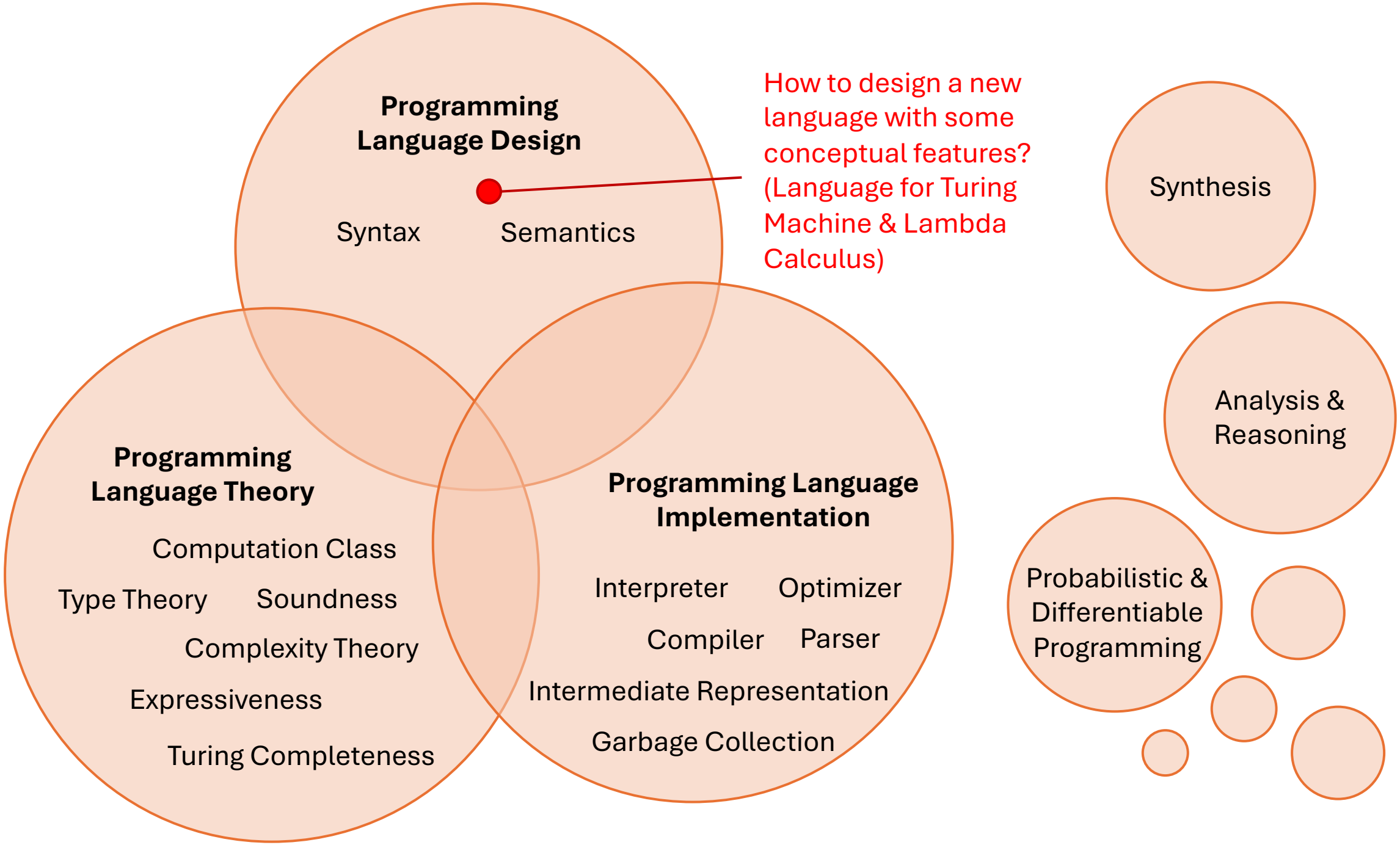
Datafun

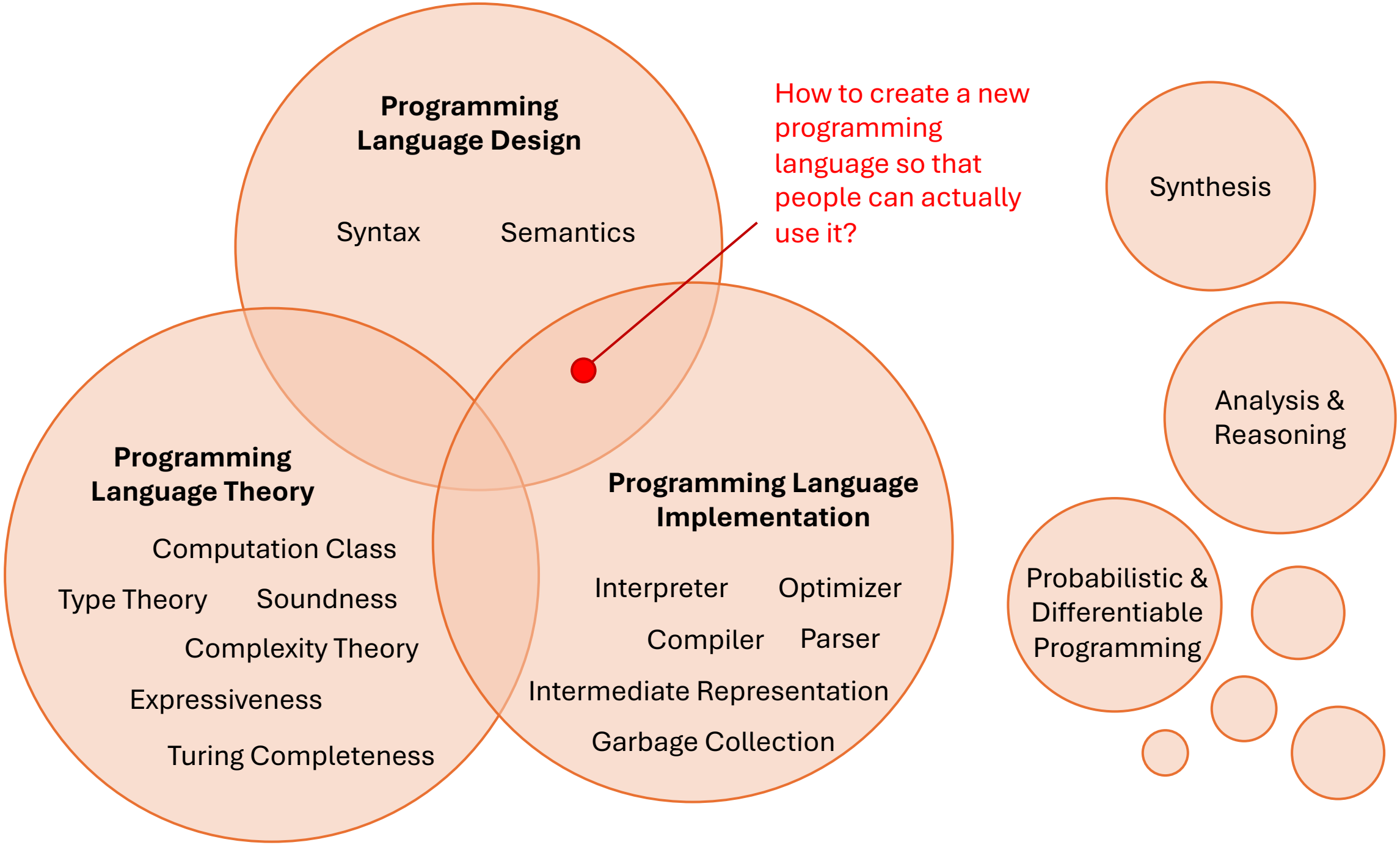
Lobster

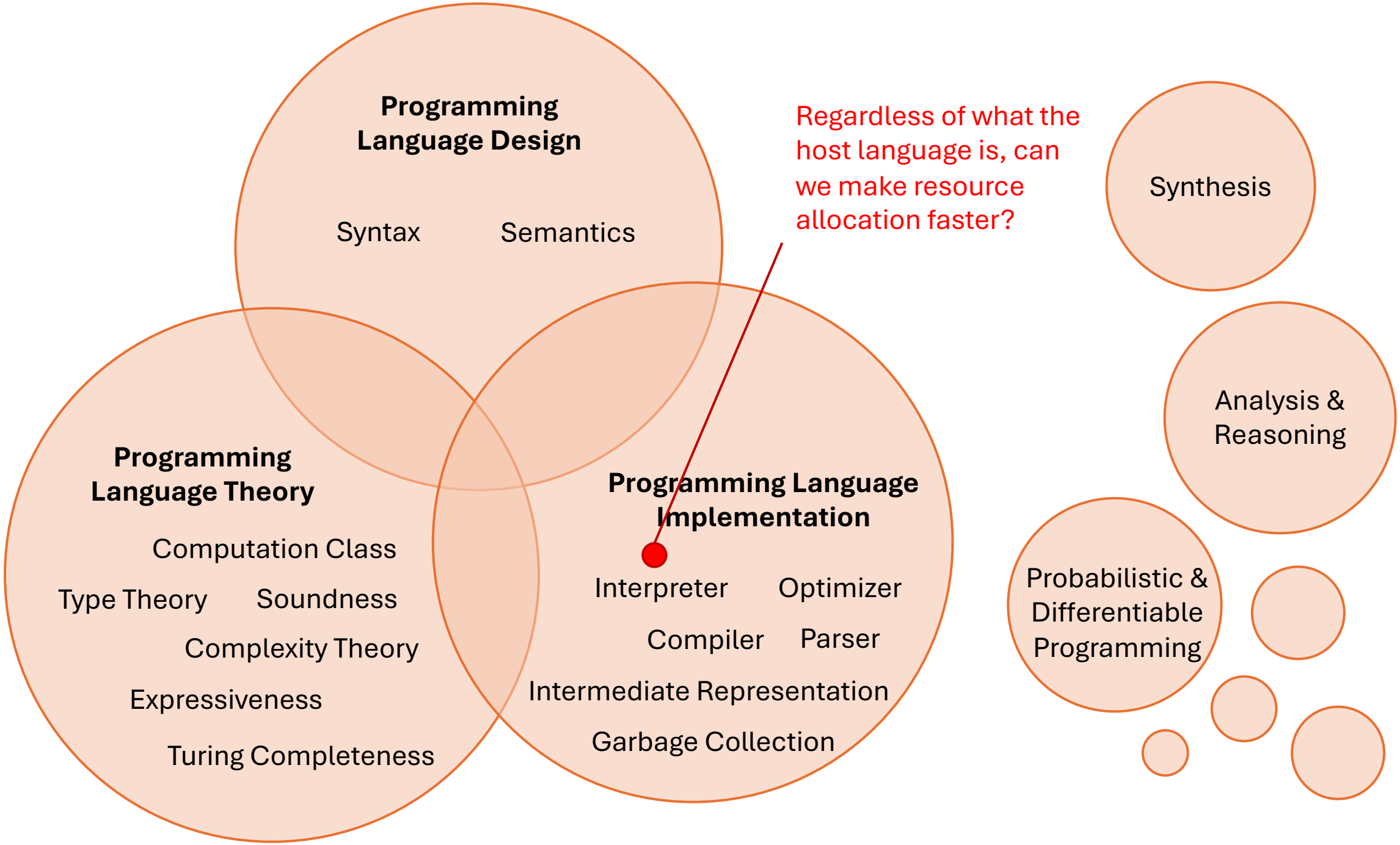
SQL

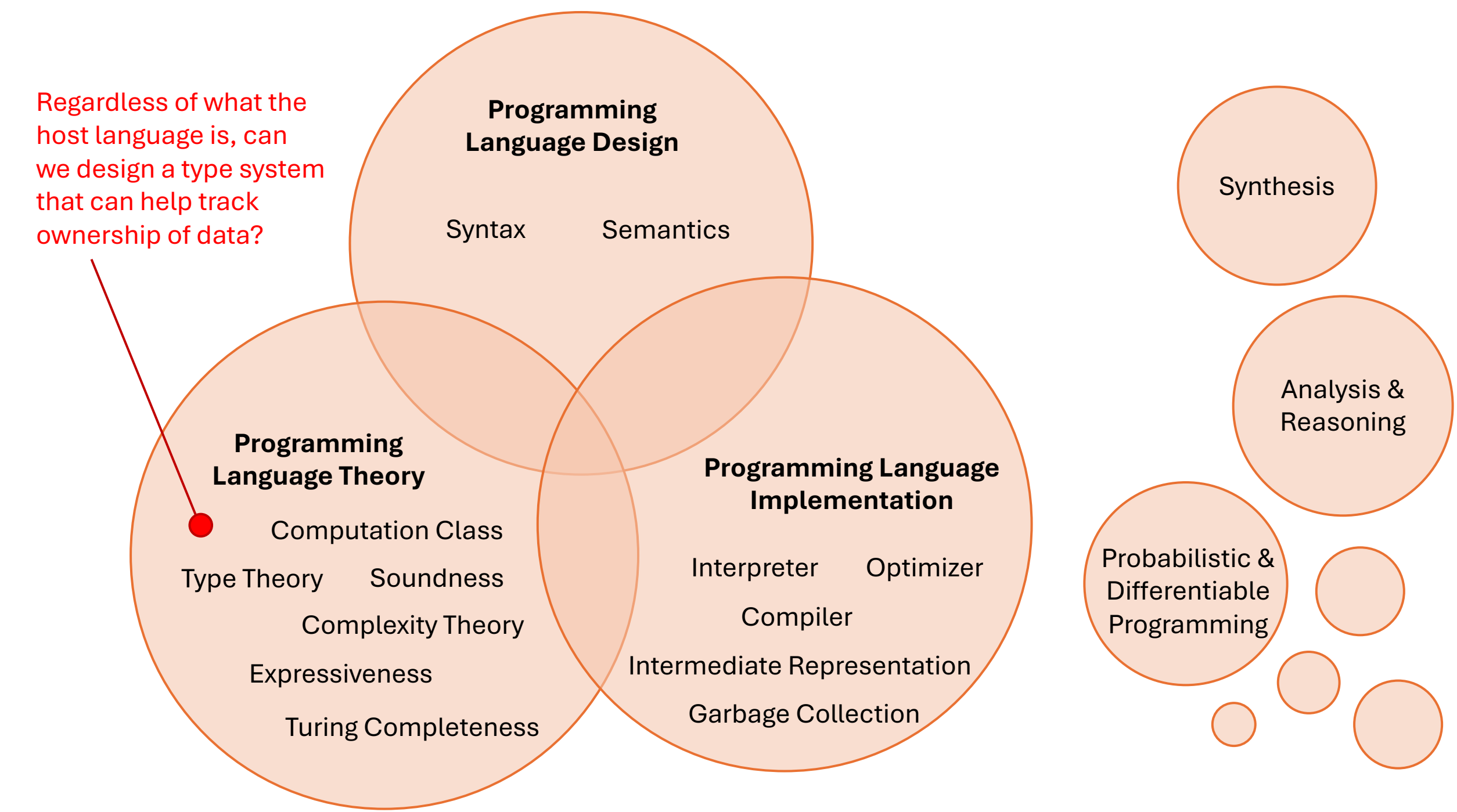
Scallop

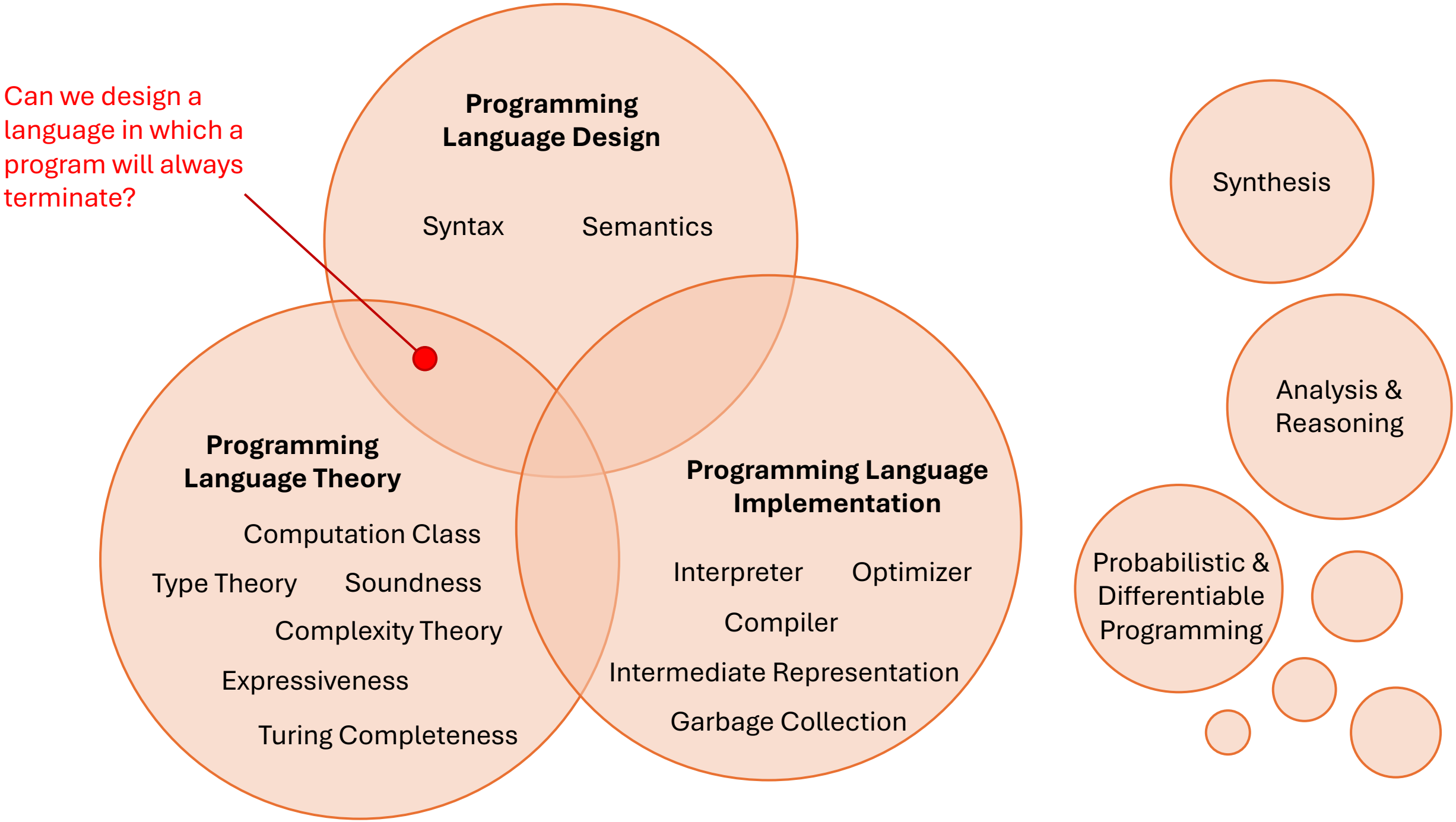

Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Parser
Intermediate Representation
Garbage Collection
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming

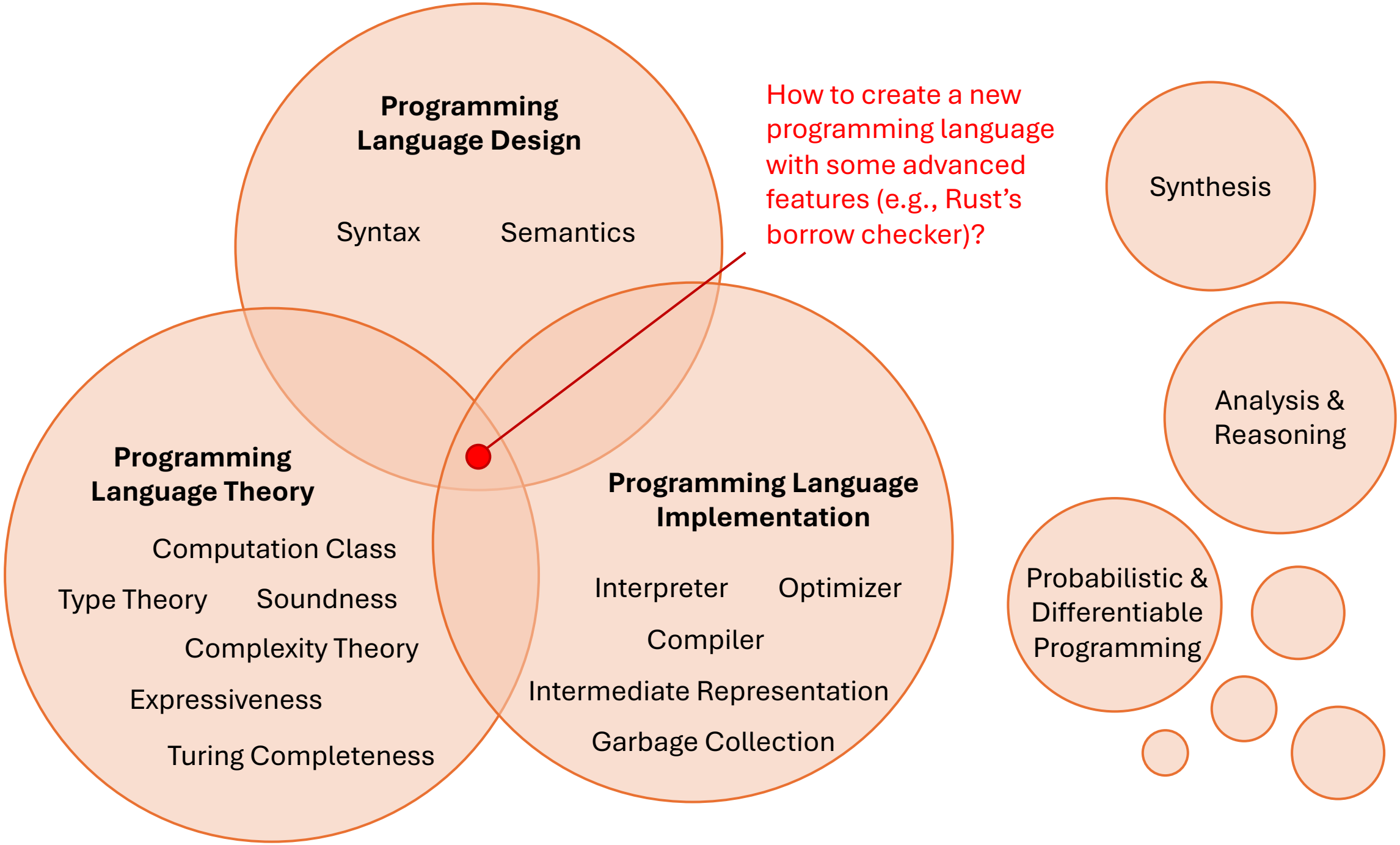

Programming Language Design
Syntax
Semantics
How to design a new language with some conceptual features? (Language for Turing Machine & Lambda Calculus)
Synthesis
Analysis & Reasoning
Probabilistic & Differentiable Programming
Programming Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language Implementation
Interpreter
Optimizer
Compiler
Parser
Intermediate Representation
Garbage Collection

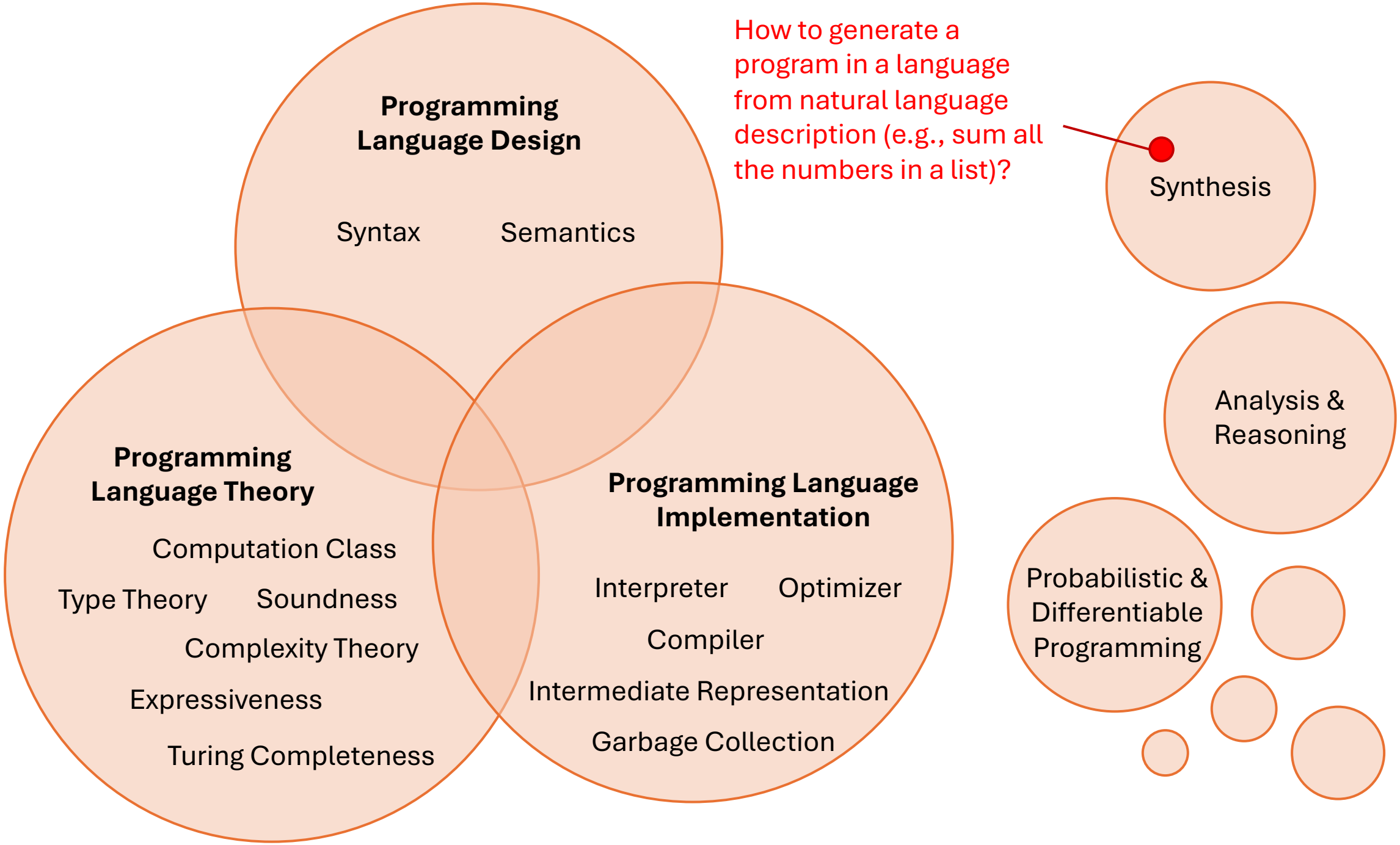

Programming
Language Design
Syntax
Semantics
How to create a new programming language so that people can actually use it?
Synthesis
Analysis & Reasoning
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Parser
Intermediate Representation
Garbage Collection
Probabilistic & Differentiable Programming

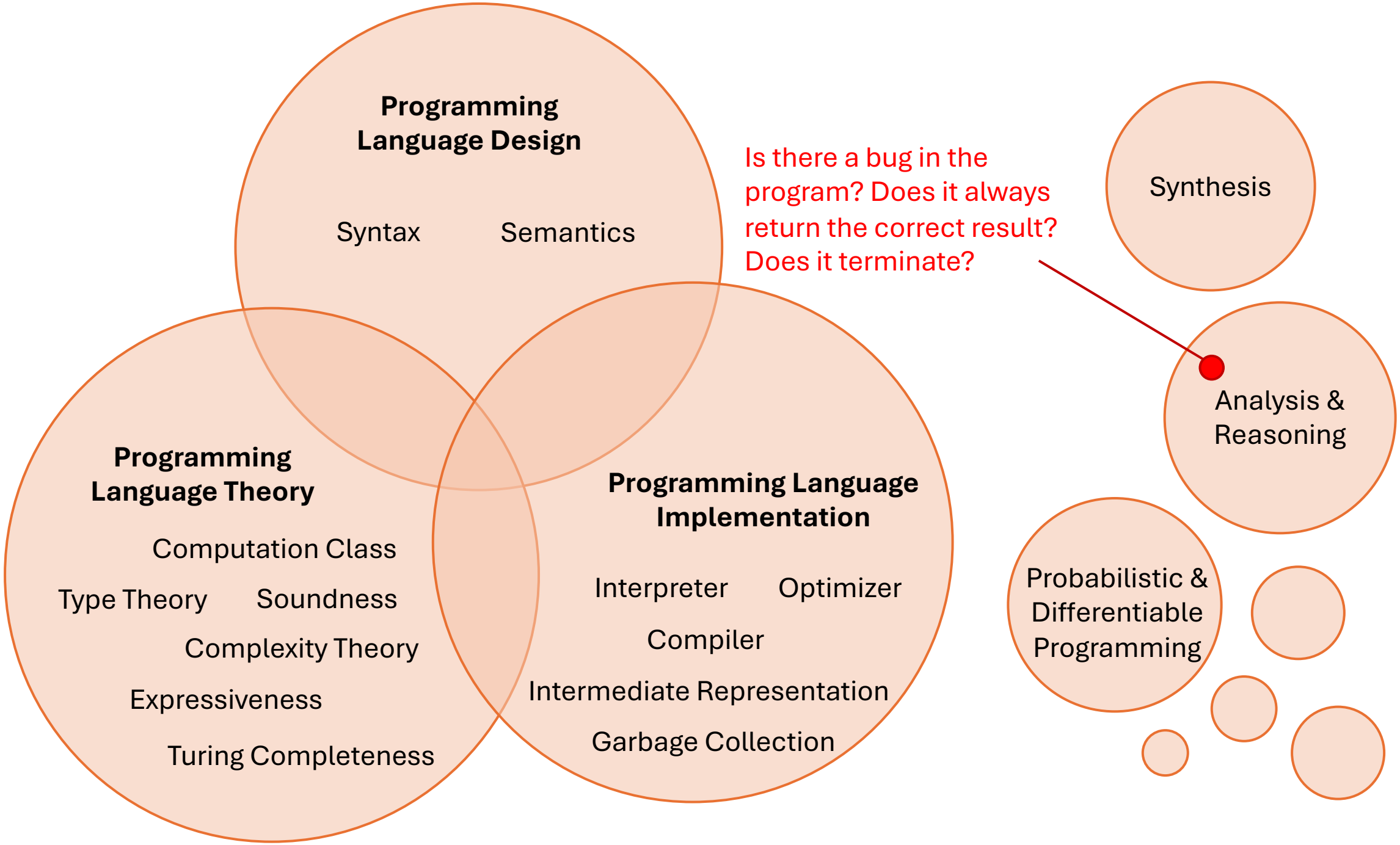

Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Parser
Intermediate Representation
Garbage Collection
Regardless of what the host language is, can we make resource allocation faster?
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming

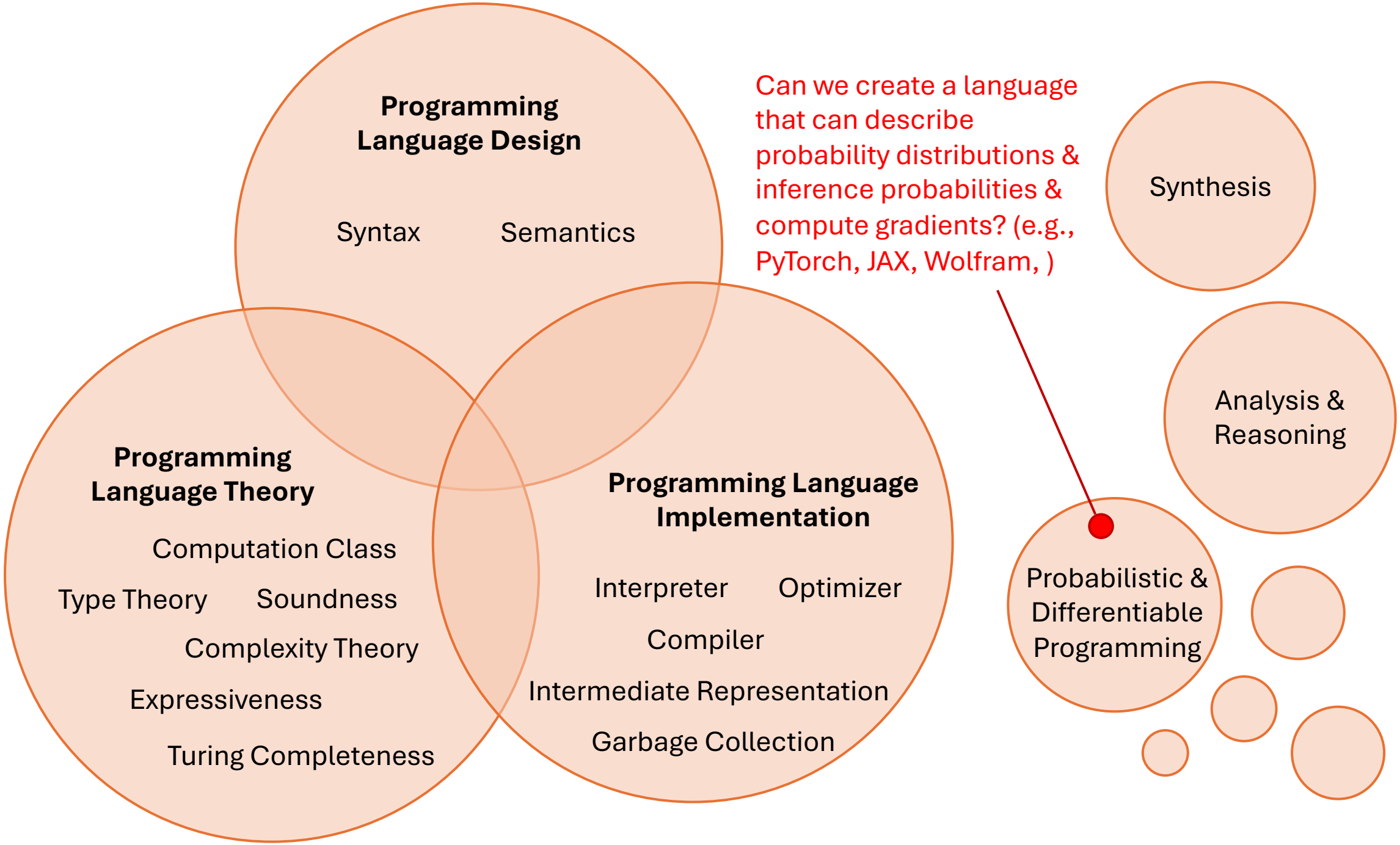

Regardless of what the host language is, can we design a type system that can help track ownership of data?
Programming Language Design
Syntax
Semantics
Programming Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
Synthesis
Analysis & Reasoning
Probabilistic & Differentiable Programming


Can we design a language in which a program will always terminate?
Programming Language Design
Syntax
Semantics
Programming Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
Synthesis
Analysis & Reasoning
Probabilistic & Differentiable Programming


Programming
Language Design
Syntax
Semantics
How to create a new programming language with some advanced features (e.g., Rust's borrow checker)?
Synthesis
Analysis &
Reasoning
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
Probabilistic &
Differentiable
Programming


Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
How to generate a program in a language from natural language description (e.g., sum all the numbers in a list)?
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming


Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
Is there a bug in the program? Does it always return the correct result? Does it terminate?
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming


Programming
Language Design
Syntax
Semantics
Programming
Language Theory
Computation Class
Type Theory
Soundness
Complexity Theory
Expressiveness
Turing Completeness
Programming Language
Implementation
Interpreter
Optimizer
Compiler
Intermediate Representation
Garbage Collection
Can we create a language that can describe probability distributions & inference probabilities & compute gradients? (e.g., PyTorch, JAX, Wolfram,)
Synthesis
Analysis &
Reasoning
Probabilistic &
Differentiable
Programming

Logistics

- Lecture
  - When: Tue/Thu 3:00 – 4:15pm
  - Where: Krieger Hall 170
- Course Website
  - https://pl.cs.jhu.edu/pl/
  - Discussions: courselore
  - Grading: gradescope

Programming Languages

Python

C

C++

C#

Rust

Haskell

JavaScript

Common Lisp

Ruby

LLVM

Assembly

Coq

Prolog

Scallop

Shell

General Purpose Programming Languages

GO

Python

Swift

Java

php

C

C++

Scala

PASCAL

C#

Rust

Haskell

JS JavaScript

OCaml

julia

Common Lisp

Ruby

LaTeX

CSS

LEAN THEOREM PROVER

LLVM

HTML

ASM Assembly

Coq

Prolog

SQL

SystemVerilog

Scallop

Cg GLSL HLSL

Shell

Domain Specific Programming Languages

Programming Languages that we will learn

Programming Languages that we will learn

Primary language we are going to learn and use

JavaScript

Python

Rust

Supplementary languages to illustrate generalizability

Key Learning Objectives

- Programming Paradigms:
  - Imperative, Functional, Declarative, …
- Programming Principles:
  - Lambda Functions, Recursion, Types, Records, Algebraic Data Structures, State, Monad, Effect, Concurrency, Exceptions
- Mathematical Foundation:
  - Syntax, Semantics, Proof Tree, Deductive Reasoning, Equivalence
- Programming Languages:
  - OCaml (main), Rust, Python, JavaScript/TypeScript

What (I hope) you can do AFTER the course:

- Spend less time on learning individual programming language!
  - Just jump right in and be proficient!
  - Why? Because you already know all the principles
- Create your own programming language!
  - Design and implement a new language
  - Instead of making features as boilerplate APIs, turn features into a language!
- Be better at commanding LLMs to help you write programs!
  - Understand which language best suit your need
  - Be more precise at describing your desired program!

Grading

- Quiz & Attendance (10%)
- Homework Assignments (8 in total, 30%)
- Midterm Exam (25%)
- Final Exam (35%)

Quiz & Attendance (10%)

- 10 quizzes in total, 1% each
- Graded by completion, NOT by correctness
- The quiz will be on paper; please bring your own pen
- Quiz will be notified 1 week in advance, usually on Thursday
- Maximum 1 quiz per week
- You can miss 2 quizzes, meaning that you'll get 2% overall grade for free
- (no quiz in the first week; starting from the second week)

Quiz & Attendance (10%)

Example Quiz:

Name: _____

Complete the Proof Tree Below:

Homework Assignments (30%)

- Throughout the semester, we will be implementing an interpreter for a functional language called **F♭**
  - Pronunciation: F-Flat (if you know music theory… this is just E)
  - Written in ASC-II characters, we use Fb
- Each homework assignment corresponds to one language feature within **F♭**
  - Starting from operators to state, objects, and types
- You will be able to see all the assignments from the beginning
  - Roughly speaking, you'll have 1-1.5 weeks per assignment

Homework Assignments (30%)

- The assignments are graded on GradeScope:
  - Autograders are available for most assignments; manual grading for a few
  - For autograded ones, you'll be able to see your score immediately after you submit. The score is based on the number of test cases passed
- Late Policy:
  - You will have 12 late days in total throughout the semester
  - For each assignment, you can use up to 3 late days
  - Please count the late days on your own; if the usage is exceeded, we will put 0 on some of your assignments

Homework Assignments (30%)

- Collaboration Policy:
  - Strongly encouraged
  - In the submission, please add “acknowledgement.md” when you have collaborated with other students; note their name down
- Use-of-AI Policy:
  - Okay to use but not entirely encouraged
  - In the submission, please add “acknowledgement.md” when you have used AI (LLMs, VLMs, etc.) Carefully note down the LLM and its version, copy-paste the entire conversation history (when available)

Homework Assignments (30%)

- Extra Credit
  - For programming assignment 4/5/6/7/8, +1% overall grade if:
    - You have implemented the entirety of the assignment with another programming language, including runnable test cases
    - Include a simple README.md telling us how to build & run
    - Recommended language: Rust, JavaScript/TypeScript, Haskell, Python, Julia
    - Other languages that you can try working with: C/C++, Java, Scala, Go, Ruby
  - If you have done all, they are +5% overall grade
  - Submission using .zip by email to the instructor (please CC TA & CA)
  - Submit anytime within 1 week after the corresponding assignment's due date
    - E.g., assignment 4 due Feb 18, the extra credit can be submitted before Feb 25.
  - Use of LLM welcomed, please acknowledge in the submission

Midterm (25%) & Final (35%)

- Midterm: Mar 10 (Tues) in-class, 1-hour
- Final: TBD, 2.5-hour
- Study materials & practice exams will be made available online
  - Answers will not be available, but we welcome you to come to the office hour or send private posts to check with the instructors & TA & CA
- You can bring one piece of cheat sheet (size TBD)