Announcements

- Project 6 Due Tomorrow, 11:59pm
  - As usual, can use <= 2 late days
  - If you hand it in on time, I'll try to grade it and send solutions
- Final review session, Monday 11-12 on Zoom
  - Come with questions about lectures, projects, practice final
- Final Exam: Tuesday, 12/6; 10:30am-12:30pm; SB 113
  - I will try to have a practice exam up today or tomorrow

Final Exam

- (All percentages are approximate and subject to change)
- Cumulative: covers all lectures (including post-Proj6), all projects
  - But heavily (~60-75%) post-midterm material
- 10-20% multiple choice
- ~5 longer questions with multiple parts
  - Some parts are MC/short answer

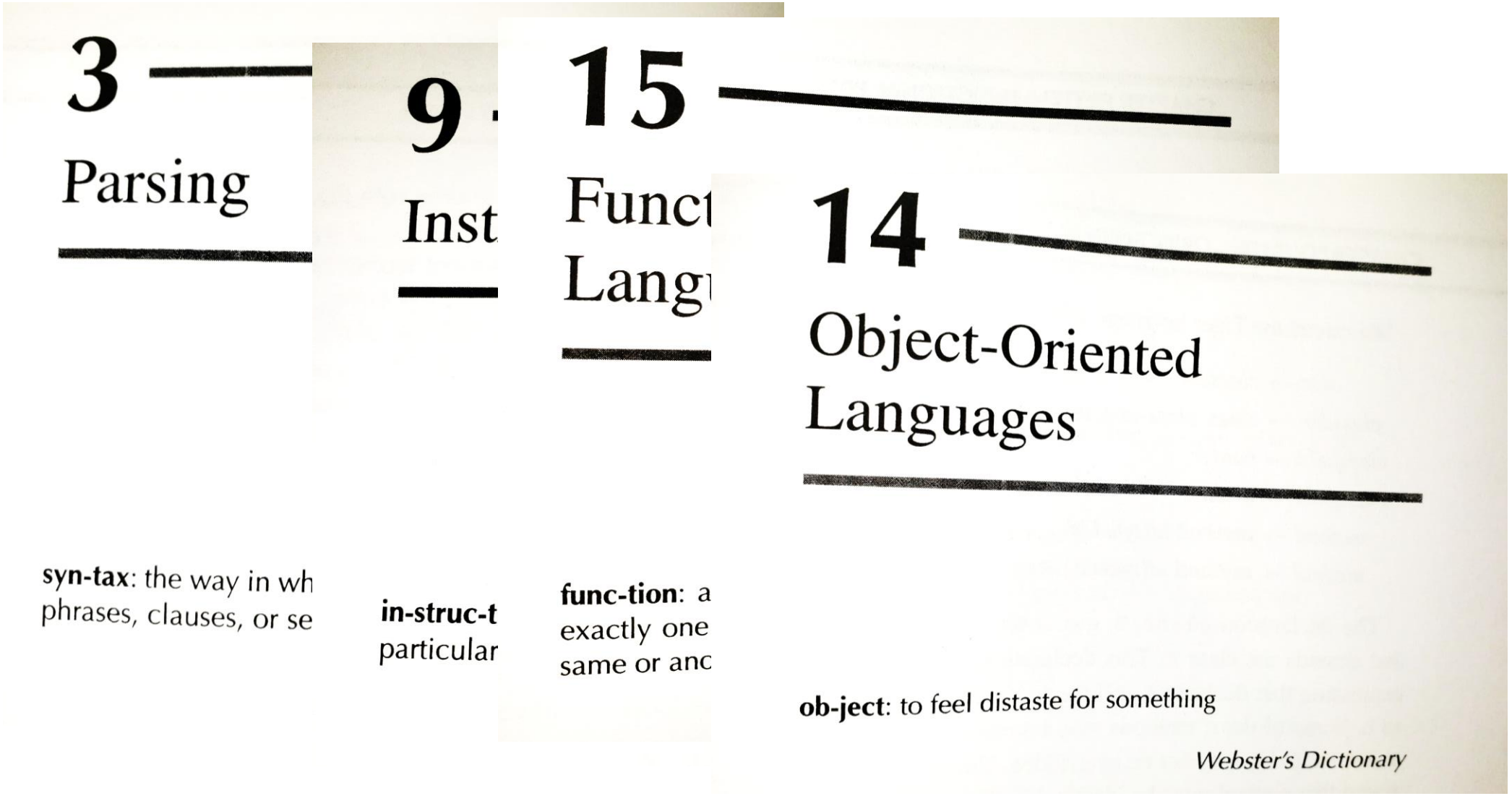
Final Exam

- I will provide some reference material
  - Mostly the same as the midterm + RISC-V Green Sheet
- Open book/Open notes
  - Can bring any amount of printed/written reference material
  - Please don’t murder too many trees
- No other aids/electronics

CS443: Compiler Construction

Lecture 27: Compiling Object-Oriented Programs

Stefan Muller


3
Parsing
syn-tax: the way in wh
phrases, clauses, or se
9
Inst
in-struc-t
particular
15
Funct
Langu
func-tion: a
exactly one
same or anc
14
Object-Oriented
Languages
ob-ject: to feel distaste for something
Webster's Dictionary

Mini-Java

```
public class F {
  int x    // Field
  public int F(int x) {  // Constructor
    this.x = x; // “this” bound in every method
  }
  public int foo(int n) {   // Method
    return this.x + n;
  }
}

F a = new F(5);   // Instantiate constructor
F b = new F(10);
return a.foo(15) + b.foo(20);    // Method calls
```

Nothing magic so far

```
module F =
struct
  type t = { x: int }
  let new (x: int): t = { x = x }
  let foo (this: t) (n: int) = this.x + n
end
```

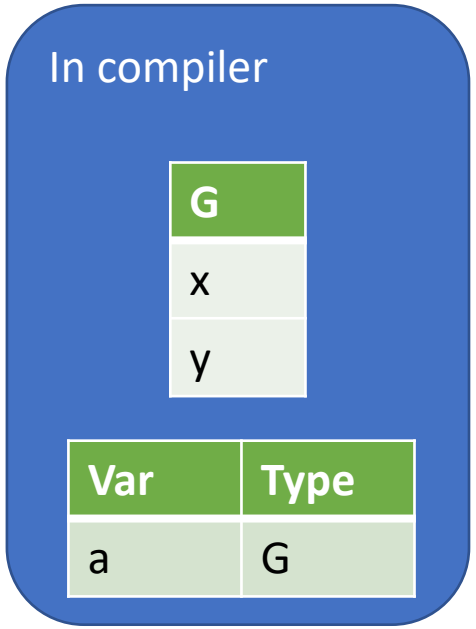
Classes can *inherit* from other classes

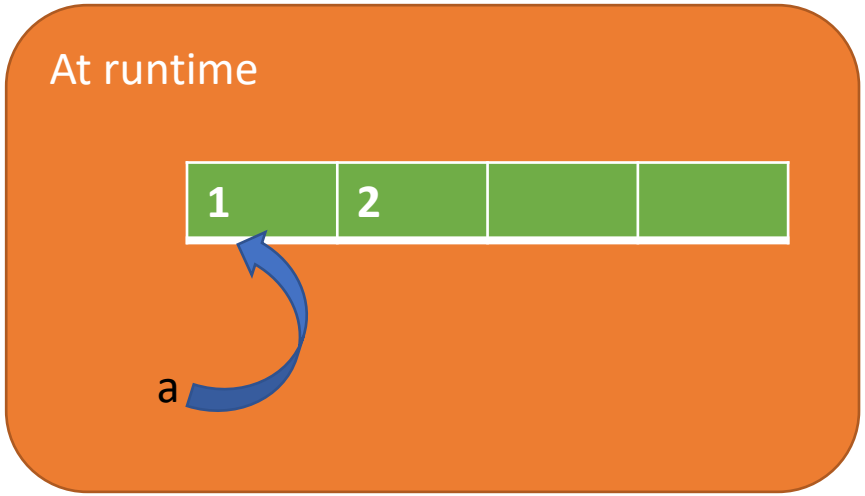
```
public class F {
  public int x;
}
public class G extends F {
  public int y;  // also has field x
}

G a = new G();
a.x = 1;
a.y = 2;
```

We can compile this like we compiled structs

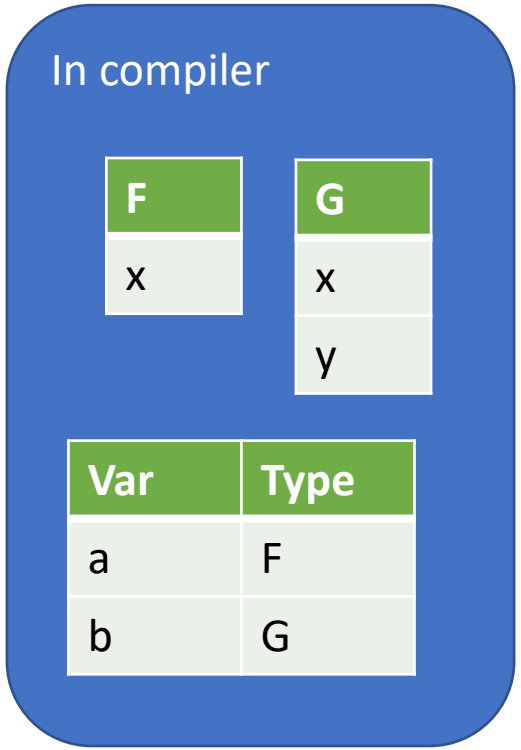
```
a.x = 1;
a.y = 2;
```

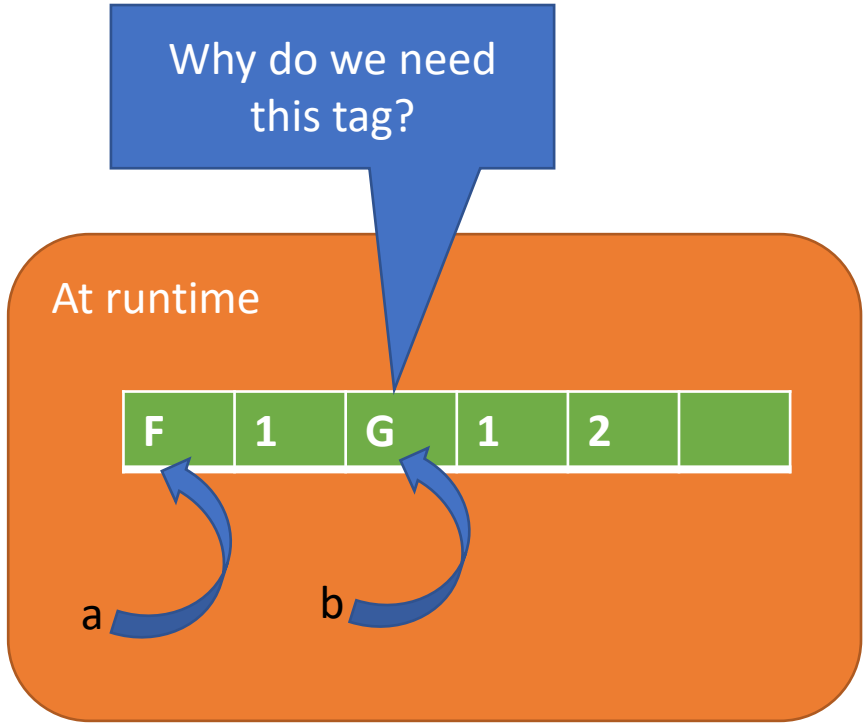




Put the inherited fields at the beginning so they overlap*

```
F a = new F();
G b = new G();
a.x = 1;
b.x = 2;
b.y = 3;
```





*This works because Java only has single inheritance

`instanceof` is evil, but we compile the language we have, not the one we want

```
int add_x_and_maybe_y (F a) {
  int s = a.x;
  if (a instanceof G) {
    s += ((G)a).y;
  }
  return s;
}
```

Classes can inherit methods too

```
class F {
  public void say_hi() { System.out.println("Hi from F!"); }
}
class G extends F {
  public void say_hi() { System.out.println("Hi from G!"); }
}
class Test {
  public void say_with(F a) {
    a.say_hi();
  }
}
F a = new G();
test.say_with(a);
```

Java puzzle: What does this print?

Answer: “Hi from G!”

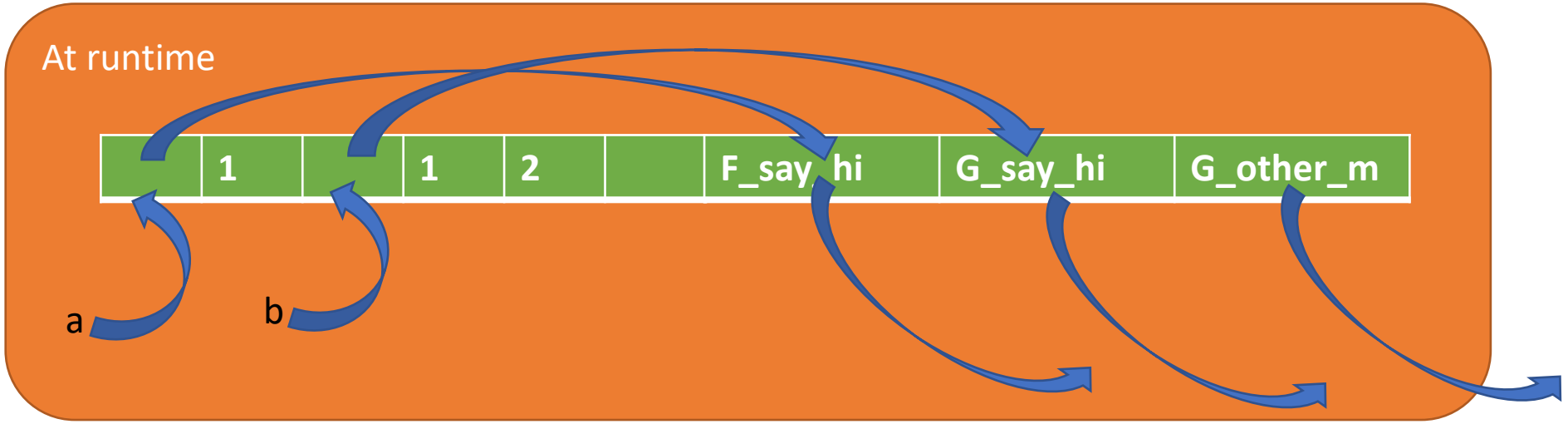
Imagine compiling to Mini-C: Hoist all functions to top level

```
public void F_say_hi(F this) {
  System.out.println("Hi from F!");
}
public void G_say_hi(G this) {
  System.out.println("Hi from G!");
}
public void Test_say_with(Test this, F a) {
  a.say_hi();
}
```

Which say_hi do we call?

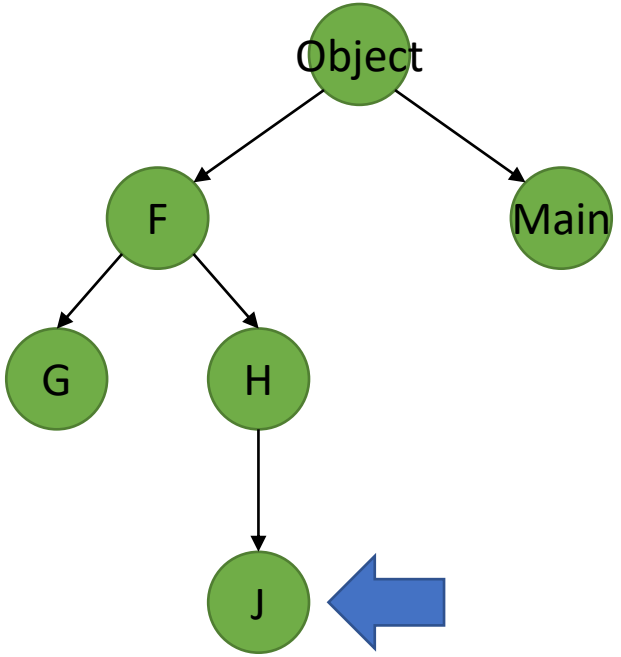
Each class has a method table (at runtime); each object has a pointer to its class's table

```
F a = new F();
G b = new G();
```



Don't need an explicit tag anymore: can use the location of the class descriptor

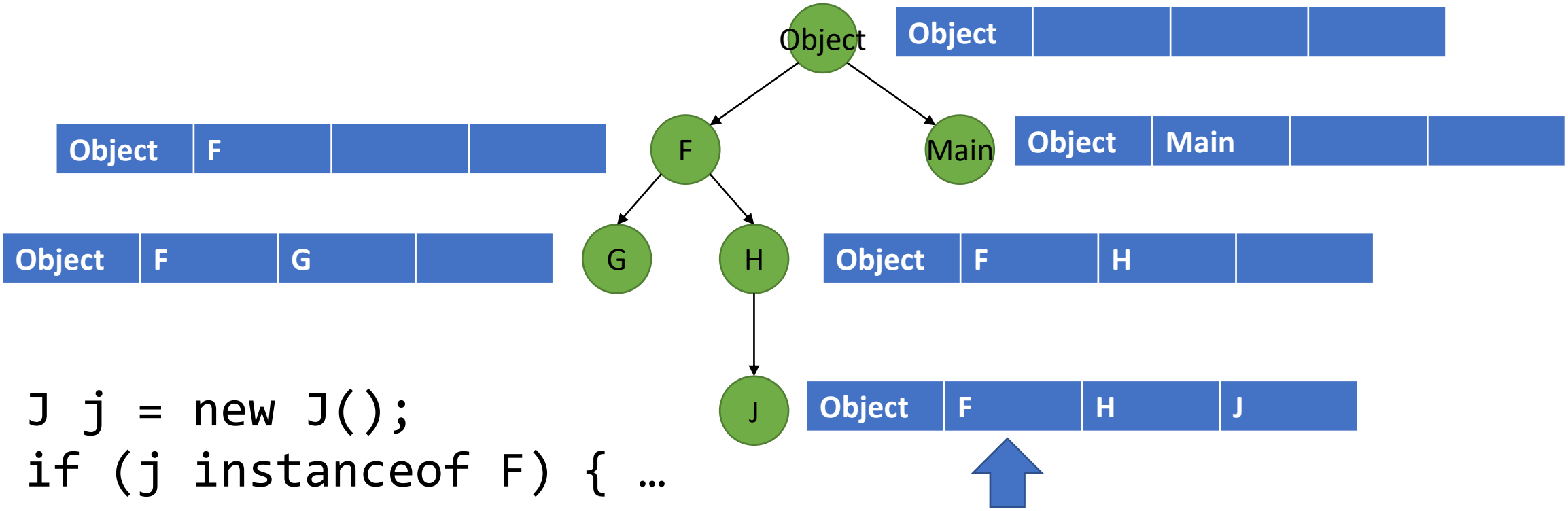
Implementing instanceof: Option 1: Store the inheritance tree in memory




```
J j = new J();
if (j instanceof F) { …
```

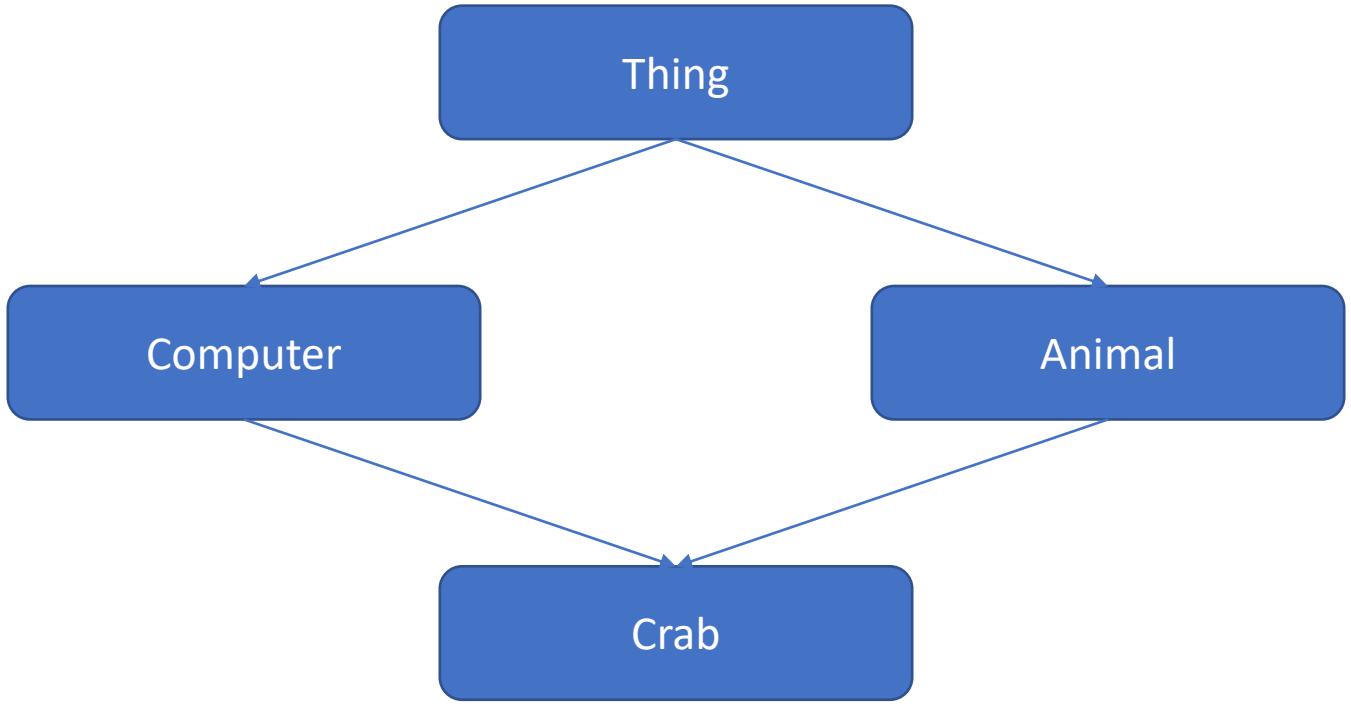
Implementing instanceof: Option 2: Have a "display" of superclasses

Class descriptor = method table + display

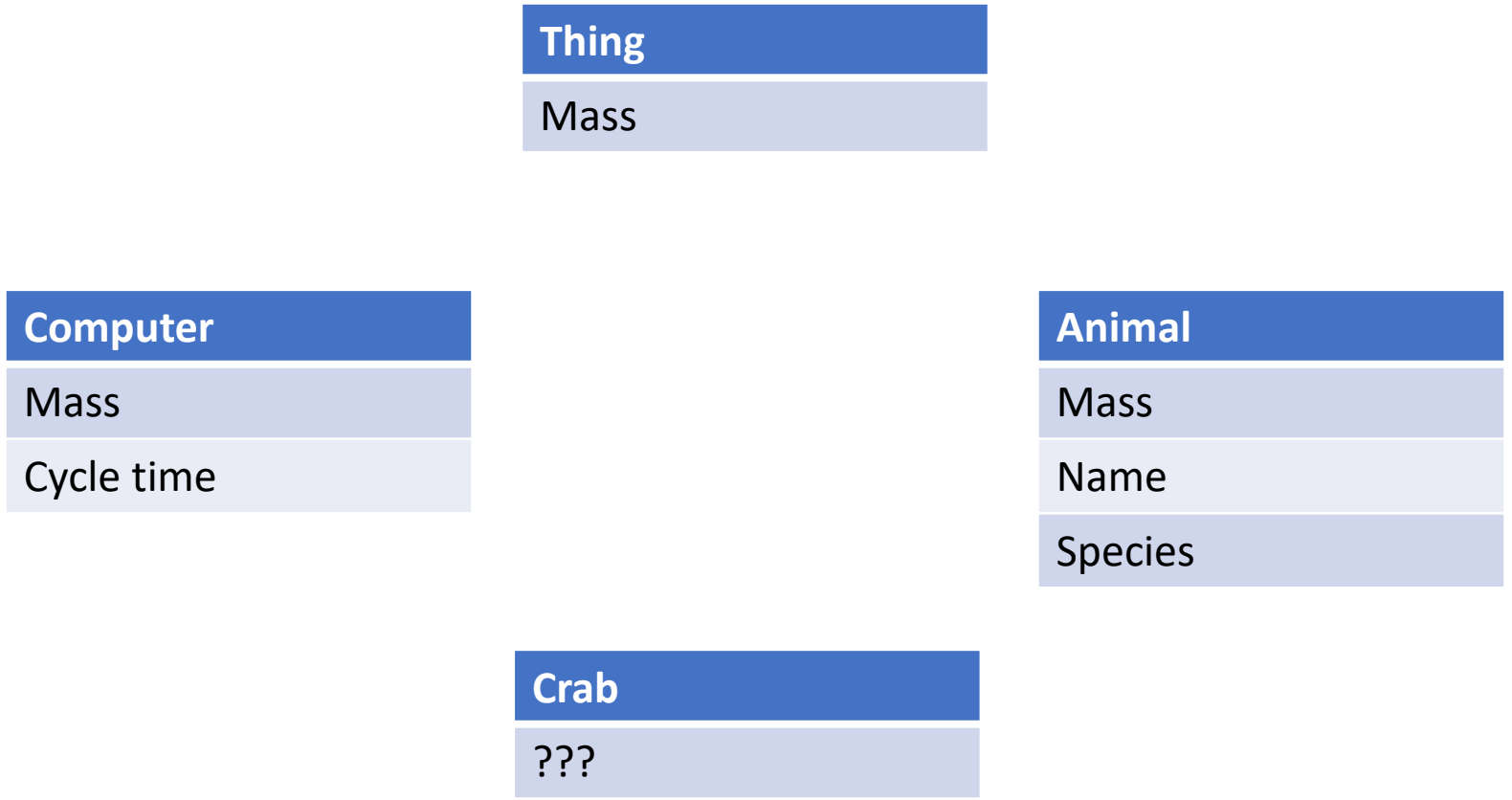



```
J j = new J();
if (j instanceof F) { …
```

Some languages allow multiple inheritance



The prefix trick for field/method layout doesn't work anymore

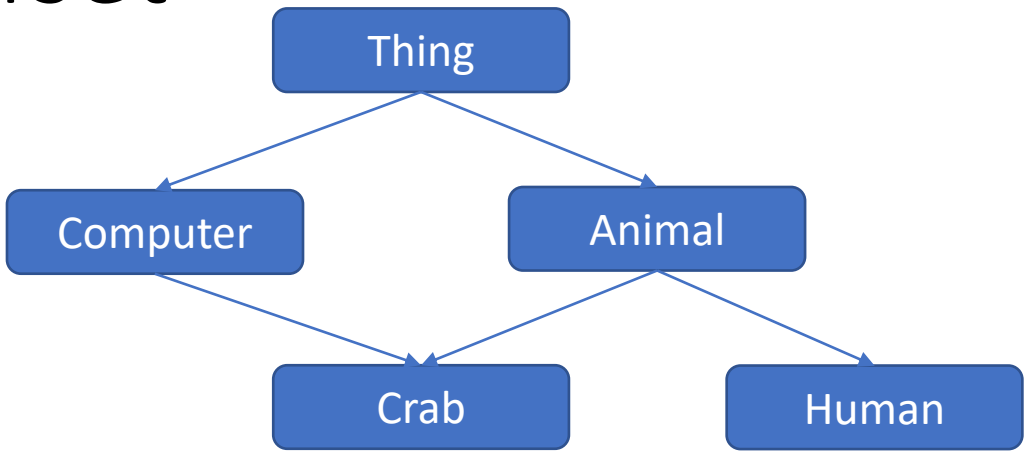


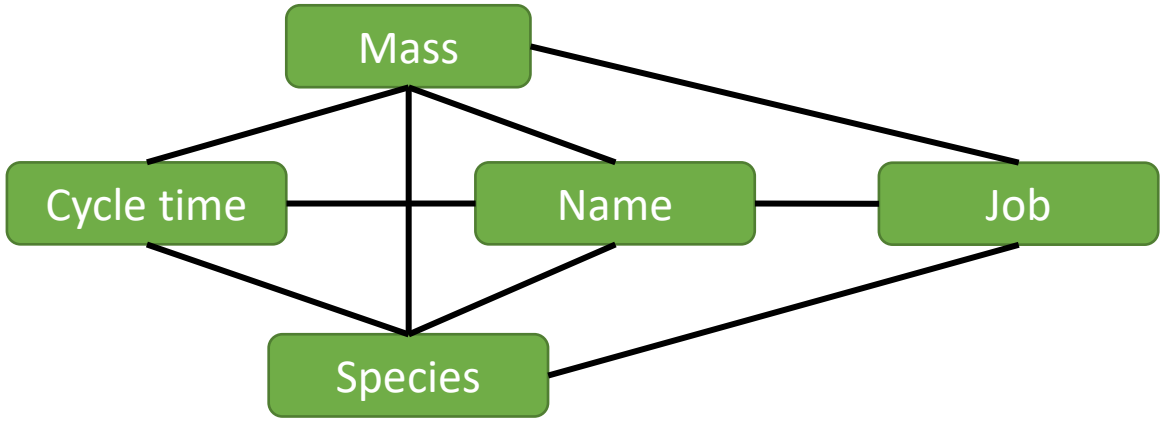
Need a global mapping from fields to offsets

Mass
Cycle time
Name
Species

If two fields are never used in the same object, can reuse that offset

Mass
Cycle time/Job
Name
Species





To minimize wasted space in objects, we'll have the class descriptor tell us where each field is

Thing	
Mass	0

Computer	
Mass	0
Cycle time	1

Crab	
Mass	0
Cycle time	1
Name	2
Species	3

Animal	
Mass	0
Cycle time	
Name	1
Species	2

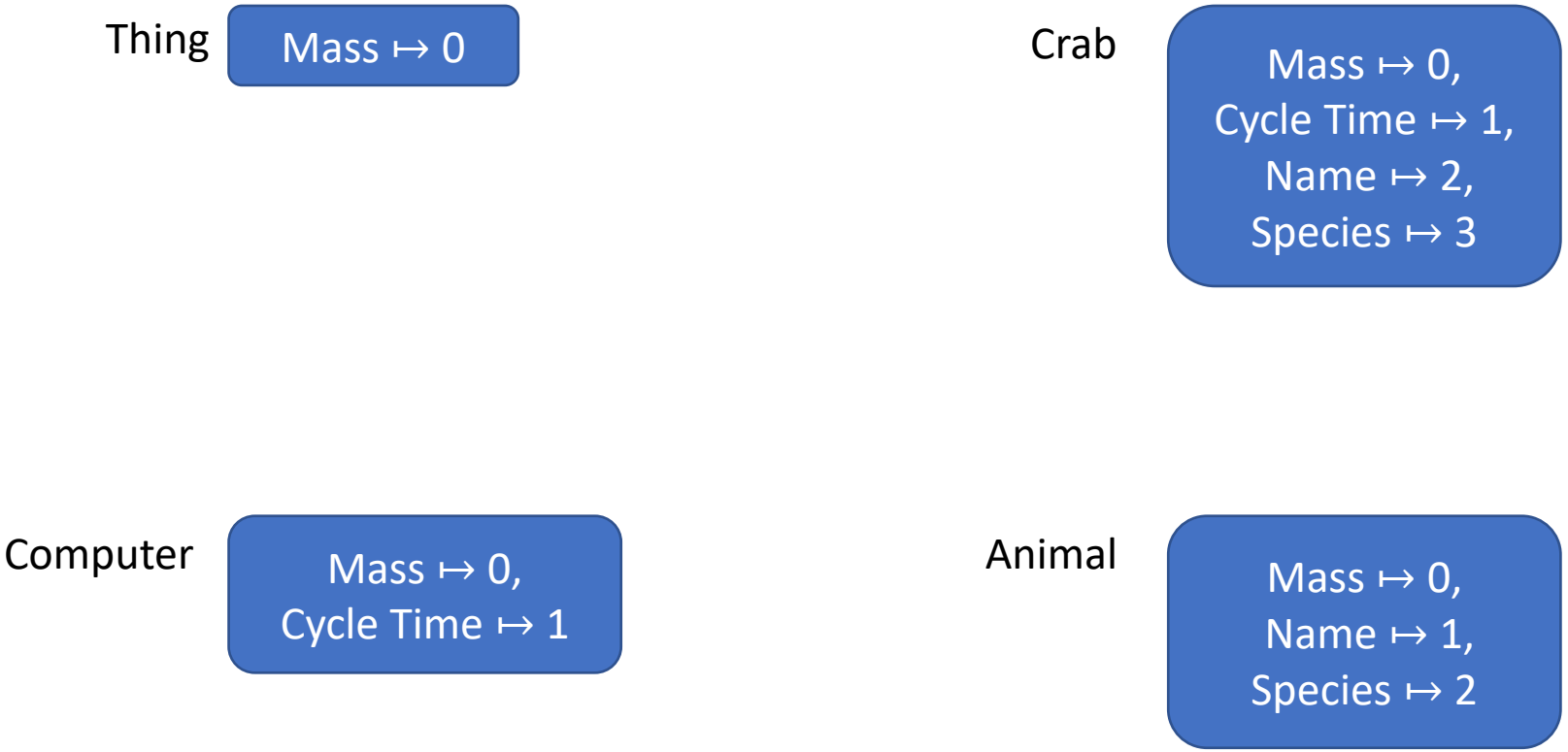
My_laptop

5lbs
0.4ns

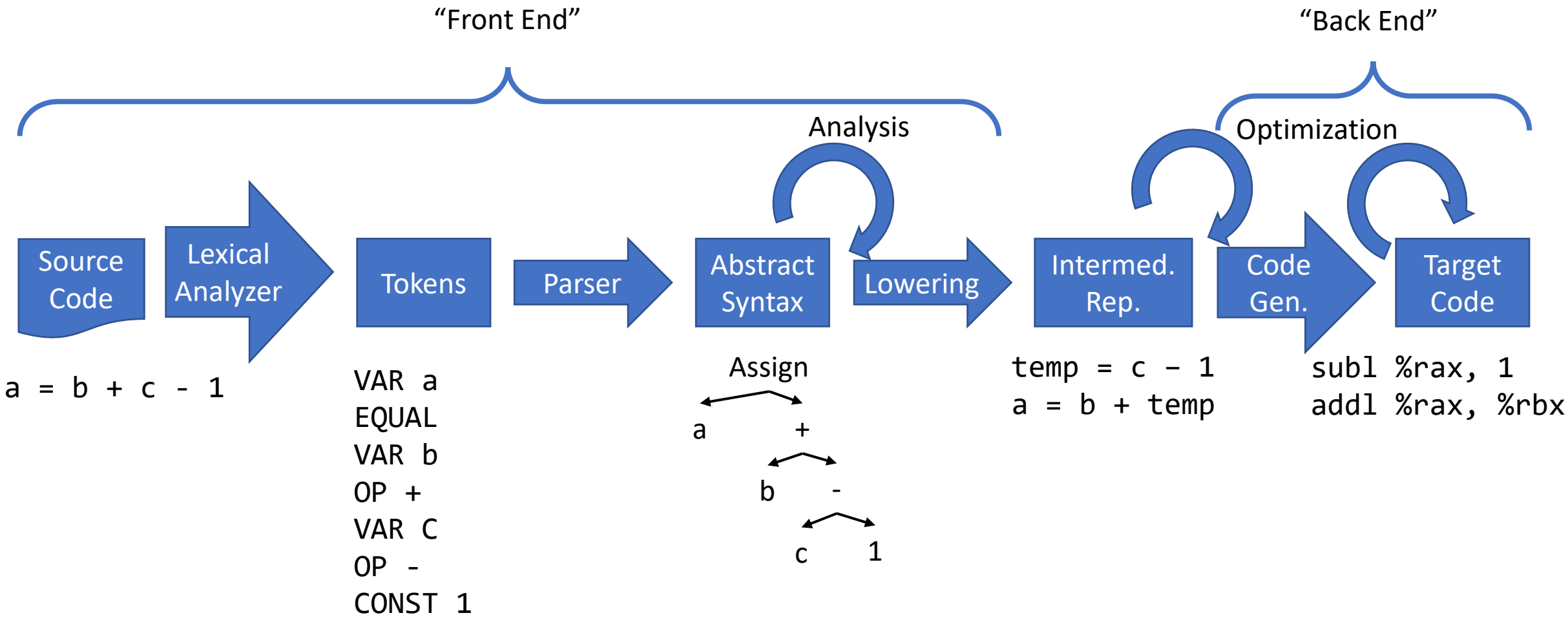
My_dog

96lbs
Hugo
Dog

Or, just have a hash table from field names to offsets in the class descriptor



Compilers translate code in phases



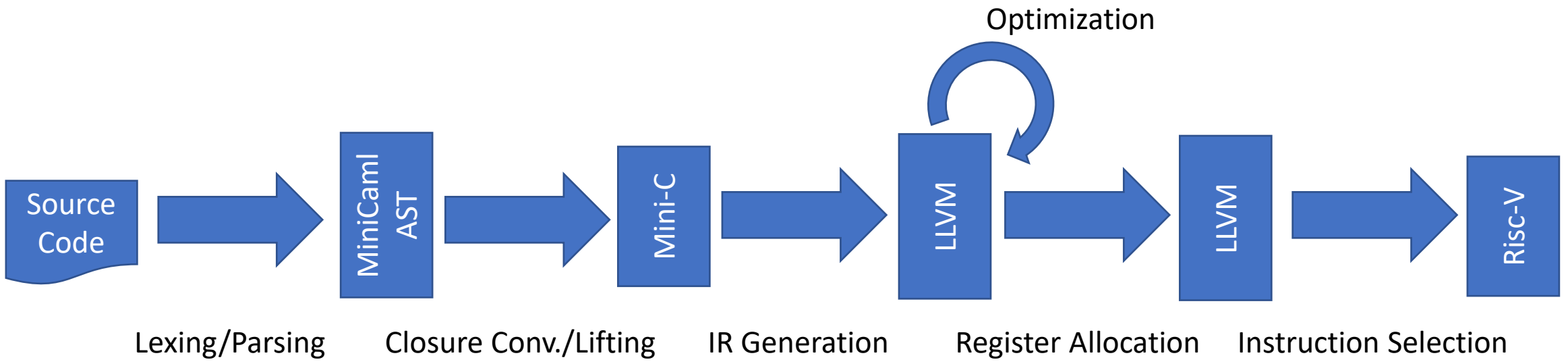
A Small ML Compiler

Higher-order
Typed
Structured Data
Nested Expressions
Unlimited Variables

First-order
Typed
Structured Data
Nested Expressions
Unlimited Variables

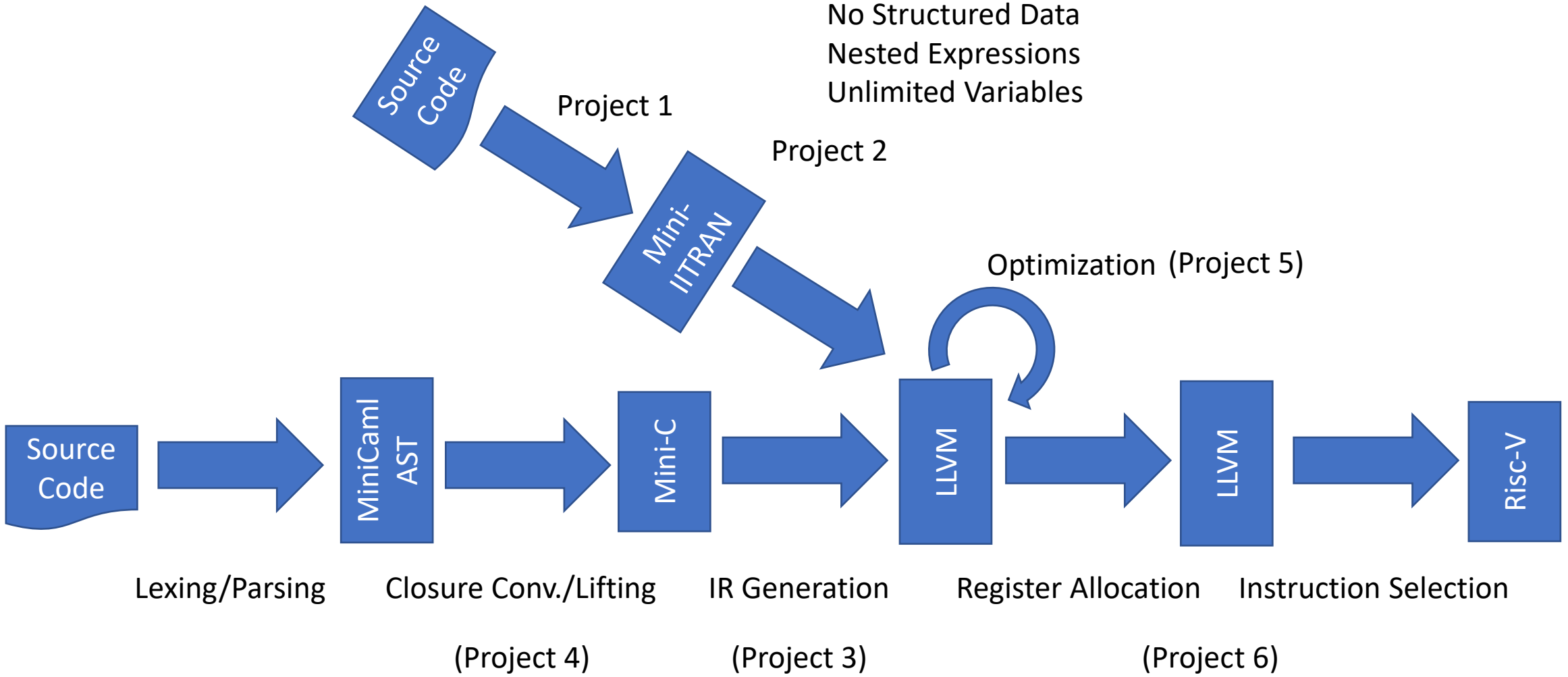
First-order
Typed
Structured Data
Flat Expressions
Unlimited Variables

First-order
Untyped
No Structured Data
Flat Expressions
32 Hardware Registers



Course Projects

First-order
Typed
No Structured Data
Nested Expressions
Unlimited Variables



Go off and make languages

- (You don't even need to write a full compiler)