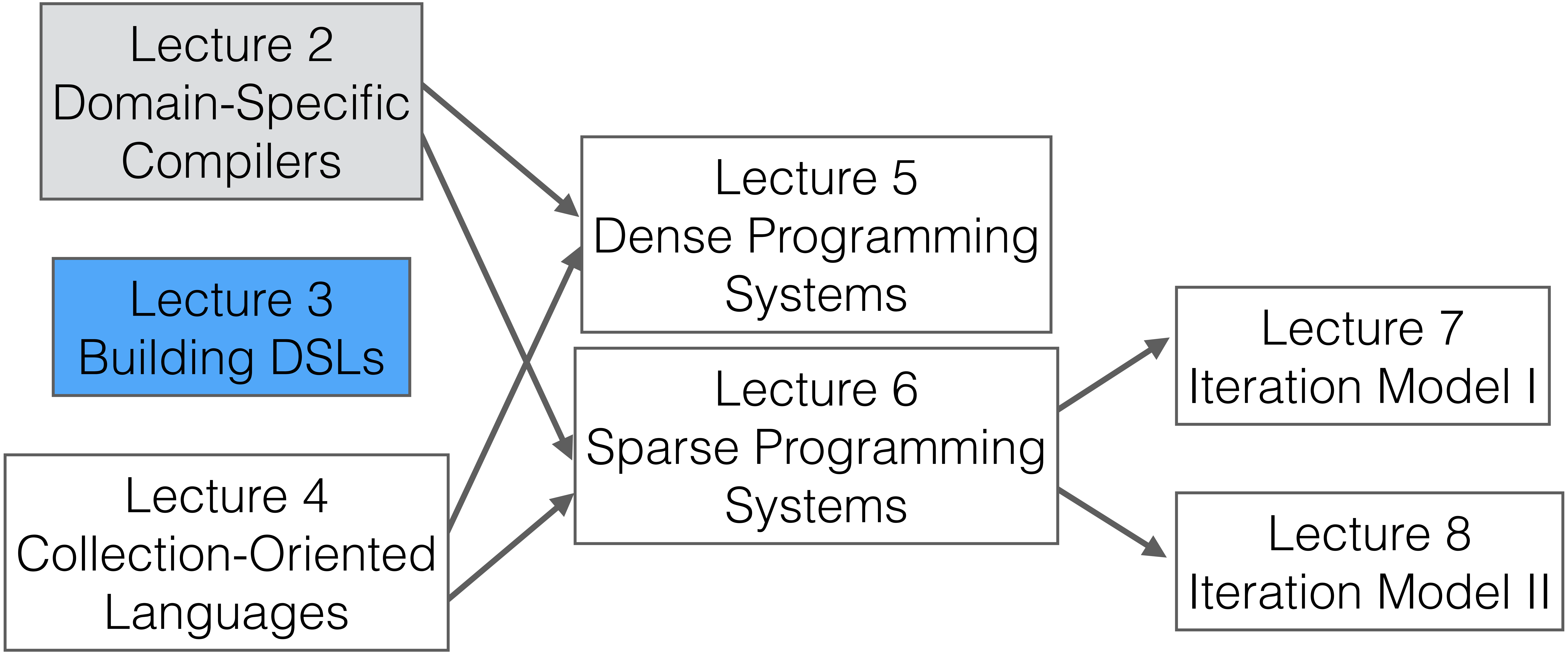
Lecture 3 — Building DSLs

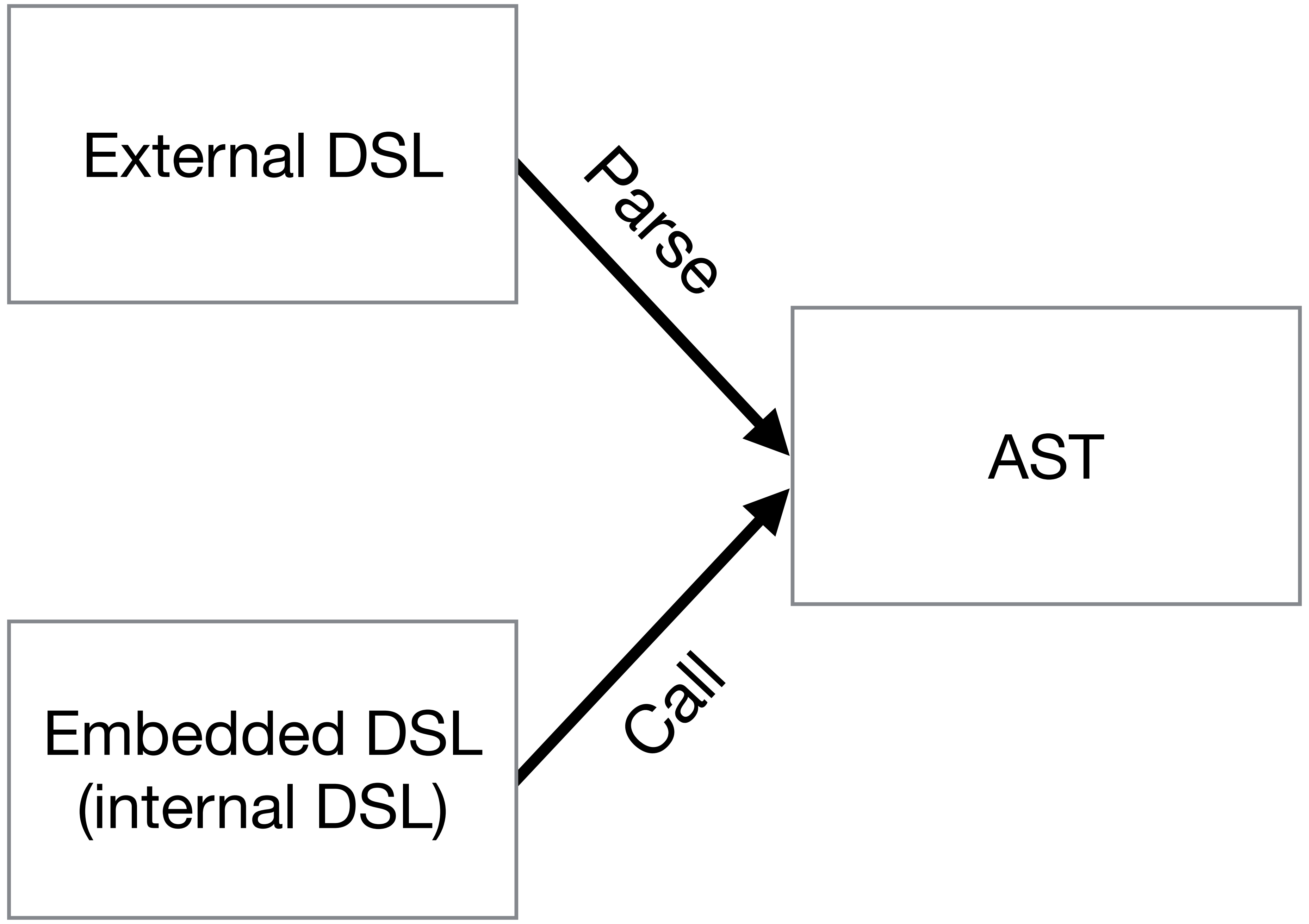
Stanford CS343D (Winter 2023)
Fred Kjolstad

Slides based on lecture by Pat Hanrahan in CS343D Fall 2020


Lecture 2
Domain-Specific Compilers
Lecture 3
Building DSLs
Lecture 4
Collection-Oriented Languages
Lecture 5
Dense Programming Systems
Lecture 6
Sparse Programming Systems
Lecture 7
Iteration Model I
Lecture 8
Iteration Model II

Types of DSLs — languages or libraries?

Implemented as standalone language

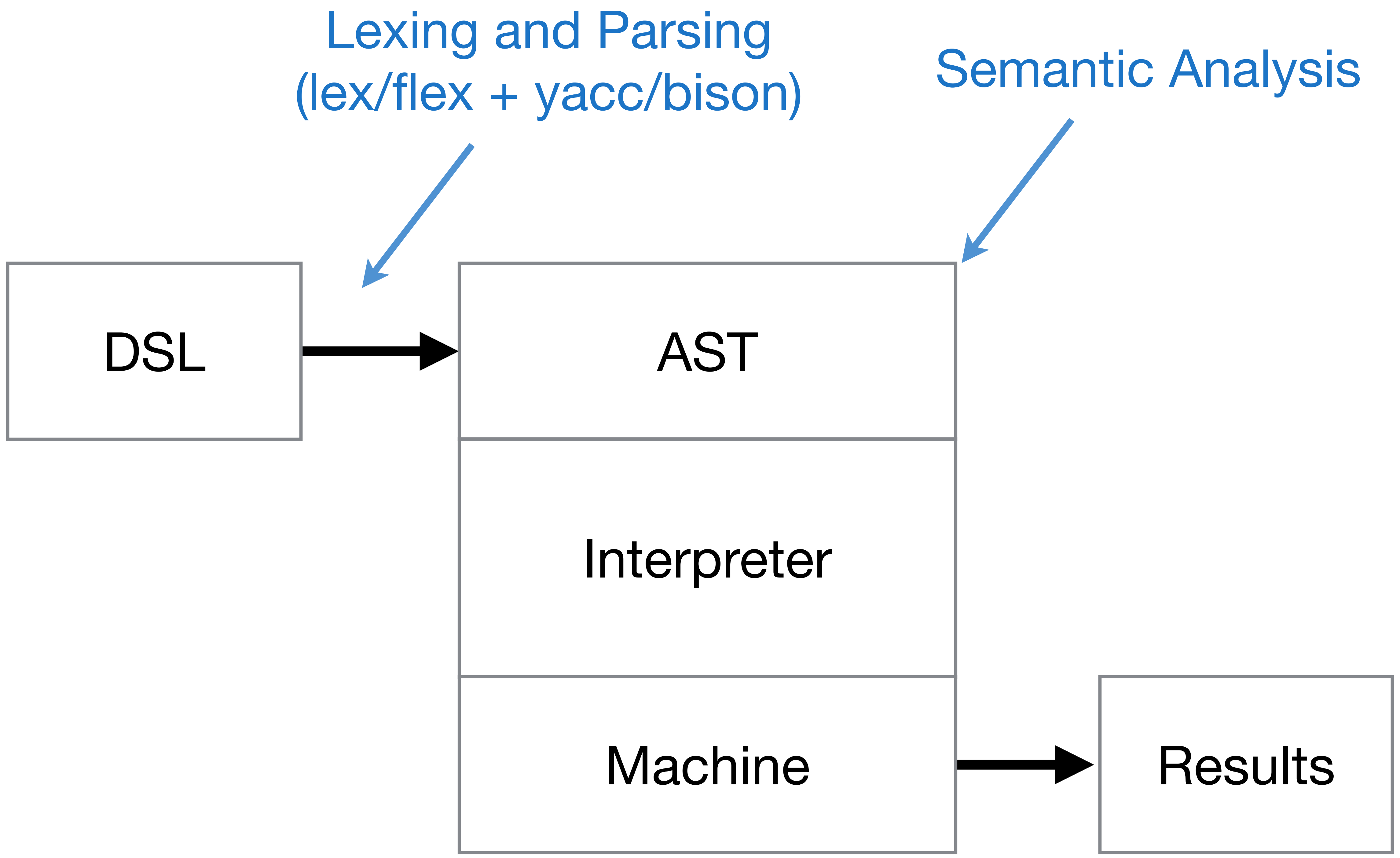


Embedded inside another language.
Ideally the host language has features to make it easy to embed DSLs.

External DSLs

That is, DSLs as textual languages

External DSLs — Implementation



External DSLs — Demo

calc1.py
calc2.py

lexical analysis
syntactic analysis
interpretation
ASTs

External DSLs — Advantages and Disadvantages

Advantages

+ Flexibility (syntax and semantics)
+ Easy to make a small textual language

Disadvantages

- Yet another programming language
- Syntactic cacophony
- Slippery slope towards generality
- Hard to interoperate with other languages
- No tool chain: IDE, debuggers, profilers

Embedded DSLs

That is, DSLs as a library

Embedded DSL — Language implemented as a library

OpenGL

```
glMatrixMode(GL_PROJECTION);
glPerspective(45.0);

for(;;) {
    glBegin(TRIANGLES);
        glVertex(…);
        glVertex(…);
        …
    glEnd();
}

glSwapBuffers();
```

Fluent Interfaces — Composable API calls with method chaining

html

```html
<ul>
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
</ul>
```

jquery

```javascript
// turn first element green
$("li:first").css("color", "green");
```

Sophisticated data rendering with embedded DSL

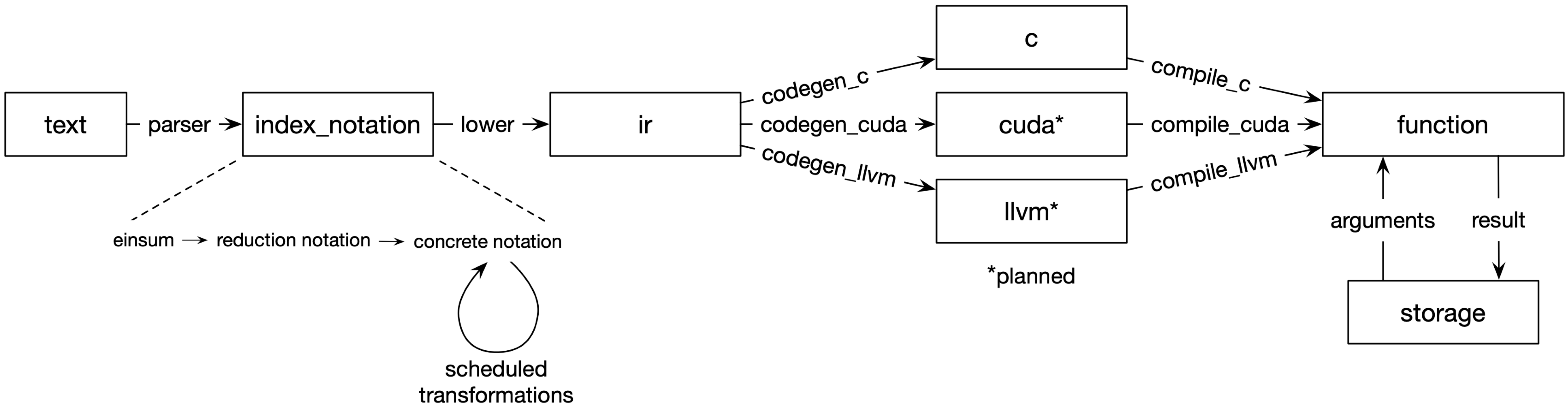
https://www.d3-graph-gallery.com/graph/density_basic.html

http://d3js.org/

Sparse Tensor Algebra DSL in C++ (taco)

```
Format dv({dense});
Format csr({dense,compressed});

Tensor<double> a({m},   dv);
Tensor<double> c({n},   dv);
Tensor<double> B({m,n}, csr);

// Load data

IndexVar i,j,i1,i2;
a(i) = sum(j, B(i,j) * c(j));

a.split(i, i1, i2, Down, 32);
 .parallelize(i1, CPUThread, NoRaces);

std::cout << a << std::endl;
```

taco — many languages



C-like DSL (Pochi) embedded in C++ for online code generation

```
Function* regexfn = codegen(“ab.d*e”);
using Regexs = int(*)(vector<string>*);
auto [regexs, inputs] = newFunction<Regexs>(“regexs”);
auto result = regexs.newVariable<int>();
auto it = regexs.newVariable<vector<string>::iterator>();
regexs.setBody(
   Declare(result, 0),
   For(Declare(it, inputs->begin()),
       it != inputs->end(),
       it++
   ).Do(
     result += StaticCast<int>(
        Call<RegexFn>(regexfn, it->c_str()))
   ),
   Return(result)
);

vector<string> input {“abcde”, “abcdde”,  // good input
                      “abde” , “abcdef”}; // bad input
buildModule();
Regexs match = getFunction<Regexs>(“regexs”);
assert(match(&input) == 2);
```

Pochi loop iterates over a C++ STL iterator

```
using RegexFn = bool(*)(char* /*input*/);
Function* codegen(const char* regex) {
  auto [regexfn, input] = newFunction<RegexFn>();
  if (regex[0] == ‘\0’) {
    regexfn.setBody(
      Return(*input == ‘\0’)
    );
  } else if (regex[1] == ‘*’) {
    regexfn.setBody(
      While(*input == regex[0]).Do(
        input++,
        If (Call<RegexFn>(codegen(regex+2), input)).Then(
          Return(true)
        )
      ),
      Return(false)
    );
  } else if (regex[0] == ‘.’) {
    regexfn.setBody(
      Return(*input != ‘\0’ &&
             Call<RegexFn>(codegen(regex+1), input+1))
    );
  } else {
    regexfn.setBody(
      Return(*input == *regex &&
             Call<RegexFn>(codegen(regex+1), input+1))
    );
  }
  return regexfn;
}
```

Pochi test on runtime regex

C# language designed for libraries and DSLs

linq

```
int count =
   (from character in Characters
   where character.Episodes > 120
   select character).Count();
```

Embedded DSLs — Advantages and Disadvantages

Advantages

+ Familiar host language syntax
+ Can combine DSL code with host language features
+ Can interoperate with other libraries
+ Complete host language toolchain

Disadvantages

- Host language syntax can be rigid and verbose
- Hard to debug DSL with host language tools
- Hard to restrict features in DSL
- Still hard to develop

DSL Construction Features

Type system: algebraic types or classes with inheritance

Polymorphism (multiple interpretation of the same AST)

Higher-order functions and lamdas (insert code)

Flexible syntax (e.g., operator overloading)

Shallow Embedding

A shallow embedding is when the expressions are interpreted in the semantics of the base language

calc1.py: direct interpretation of arithmetic

Deep Embedding

A deep embedding first builds an abstract syntax tree (AST). The abstract syntax tree is typically an algebraic data type. The AST is then evaluated with an interpreter.

calc2.py: AST represented as lists of lists

Operator Overloading

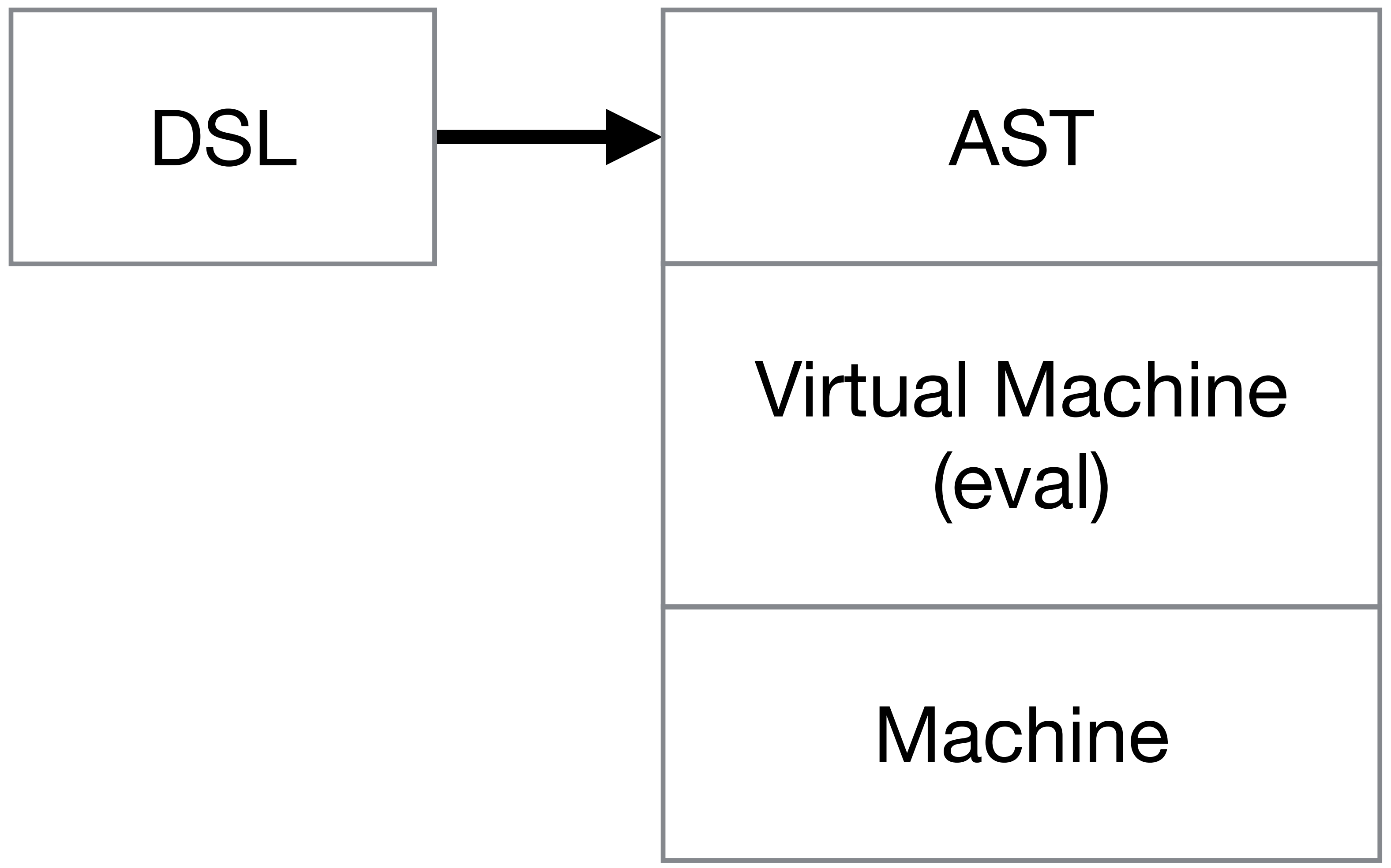
Not all “operations” can be intercepted

- Arithmetic operators
- Iteration operators
- Function definition?
- Type/class definition?
- Equality?
- Assignment?

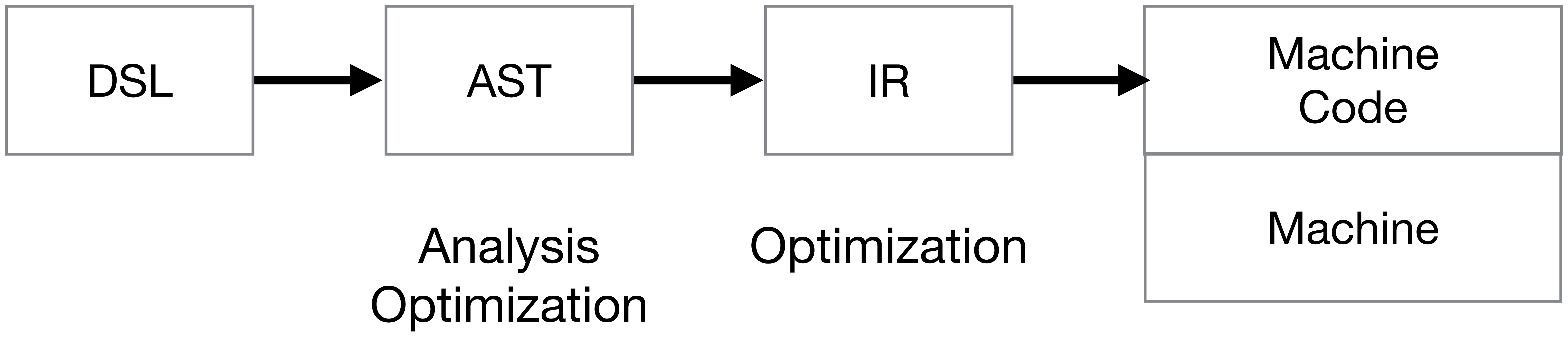
“Monkey patching” like this can be dangerous

Interpretation vs. Compilation

Interpreter



Compiler



Mini-APL Assignment

- Implement simple array processing language in C++
- We provide recursive descent parser that builds an AST
- Lower the AST to LLVM; use LLVM to generate machine code!
- The LLVM Kaleidoscope tutorial contains most of what you need to know: https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl03.html
- Assignment released today and due February 1st