KU | Fall 2018 | Drew Davidson

# EECS 665 COMPILER CONSTRUCTION

## 22 – AST Transformation

# Last Lecture

- What it is
- Why its done

- Translation rules on node type

- Bottom-up / Inside-out AST traversal

Program Semantics: Typechecking

Concepts

Implementation

Operation

# This Lecture

- What IRs are
- Why IRs are used
- Some IR forms

- IR Rewriting

Intermediate Representation

AST Processing

Concepts

Construction

# What IRs are

*Intermediate Representations*

Src->IR->Target

- Such a big, basic concept that its hard to get a precise definition
  - "An encoding of a program"
  - "What is output by the frontend of the compiler and processed by the backend of the compiler"

Good Defintion

  - "A representation of what a compiler knows about a program"
  - "A simpler language to which the source language is mapped"

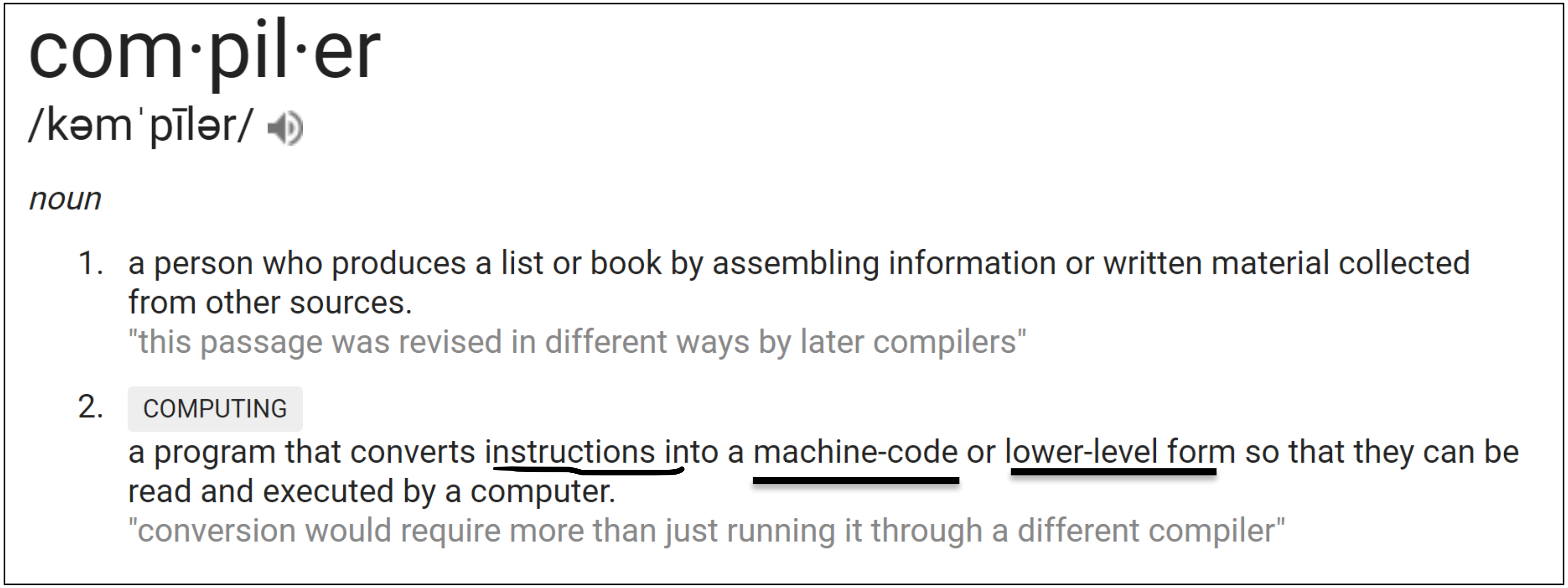
# Why use Intermediate Representations?

Basic answer: help the compiler achieve its goals

… but why use a compiler?

# Yeah, why DO we need a Compiler?

- The obvious answer: PL implementation
  - Avoid dealing with target language directly
- By strict definition of “compiler”, other options may exist



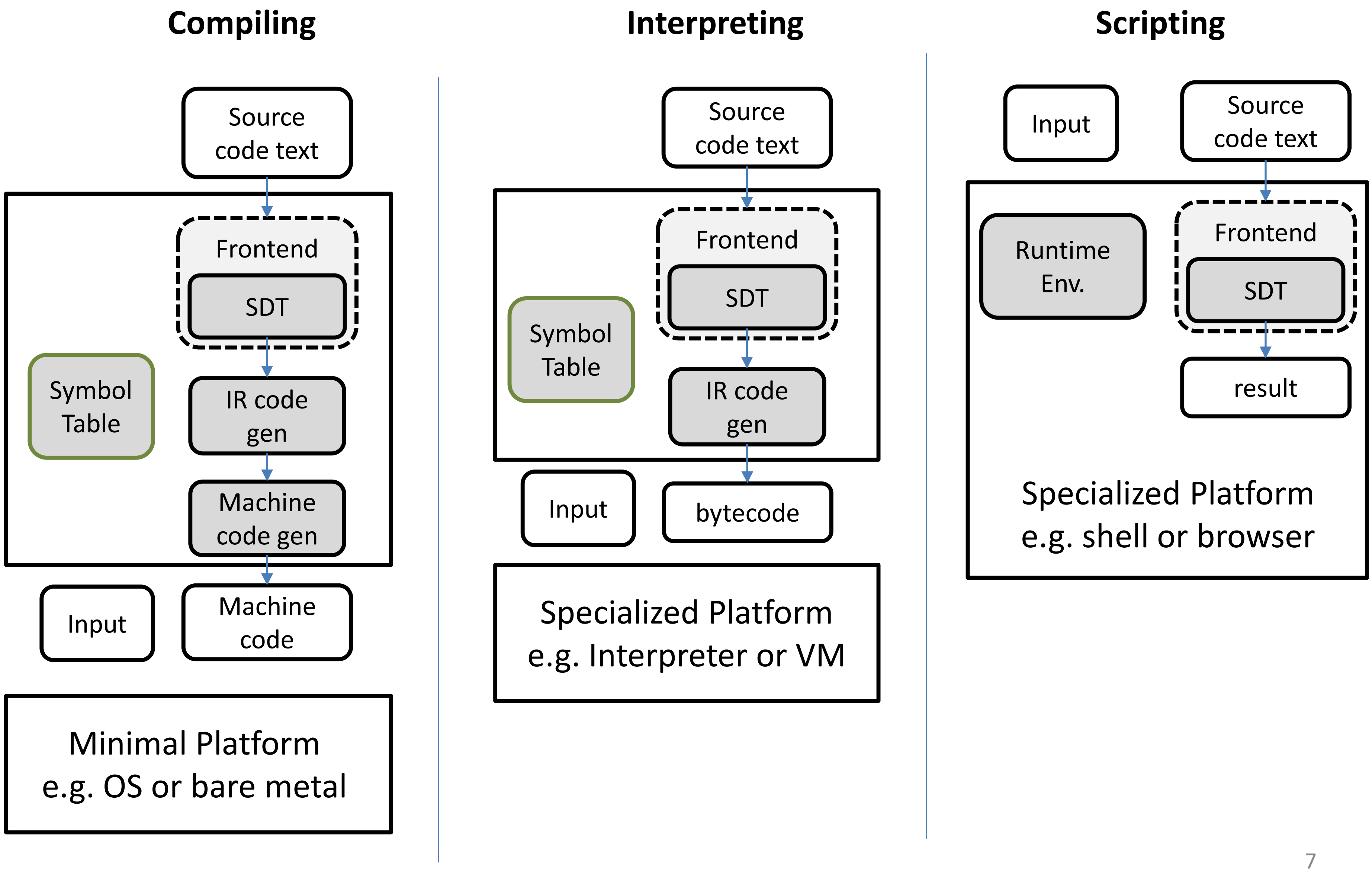
com·pil·er

/kəmˈpīlər/

*noun*

1. a person who produces a list or book by assembling information or written material collected from other sources.
   "this passage was revised in different ways by later compilers"
2. COMPUTING
   a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.
   "conversion would require more than just running it through a different compiler"

# “Alternatives” to “Compiling”



# So Compiling is Optional???

- In some contexts, no
  - If you have no runtime support, you're compiling (or coding ASM)
- In some contexts, yes
  - Interpreted languages might forgo machine code
    - May evaluate IR directly
  - Scripts might forgo source translation entirely
    - Evaluate source code directly

# Then Why Compile at All?!?!

- Abstraction
  - Don’t have to deal with the target language
- Analysis
  - Error checking: Predict bugs before they strike
  - Optimization: Generate more efficient code

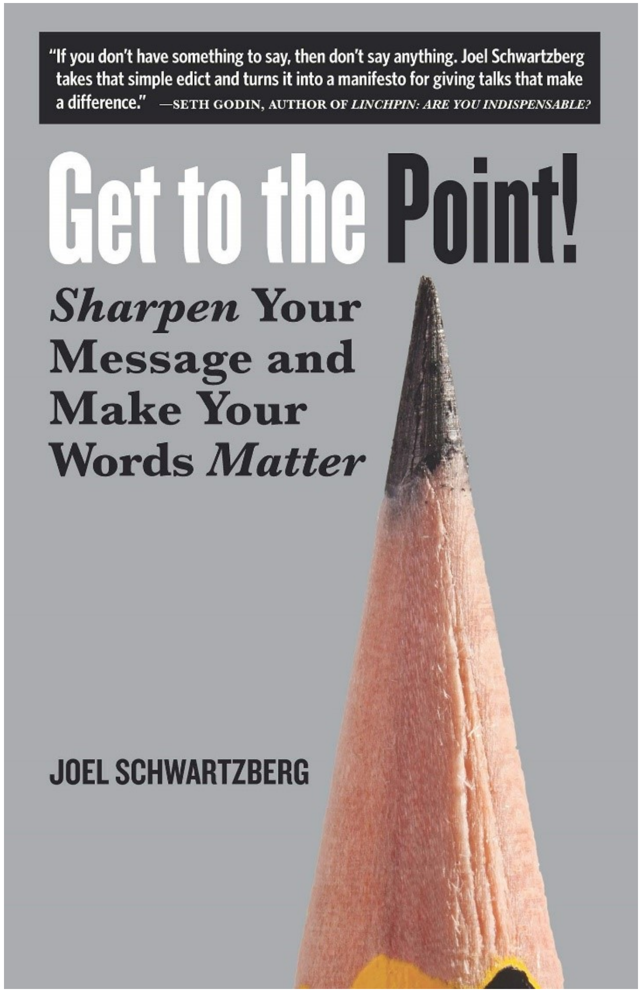
```
Commence Existential Crisis?
(y/n)
>
```

# Cool… Why did we detour?

- Minor: contextualize interpreted and scripting languages
- Major: The goals of the IR serve the goals of the compiler

Aid Abstraction

Aid Analysis


"If you don't have something to say, then don't say anything. Joel Schwartzberg takes that simple edict and turns it into a manifesto for giving talks that make a difference." —SETH GODIN, AUTHOR OF LINCHPIN: ARE YOU INDISPENSABLE?
Get to the Point!
Sharpen Your Message and Make Your Words Matter
JOEL SCHWARTZBERG

# Why Use Intermediate Representations?

- Abstraction

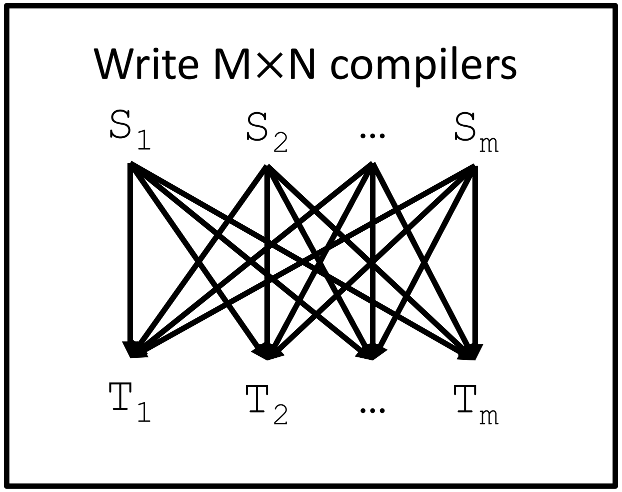
- Analysis

# Why Use Intermediate Representations?

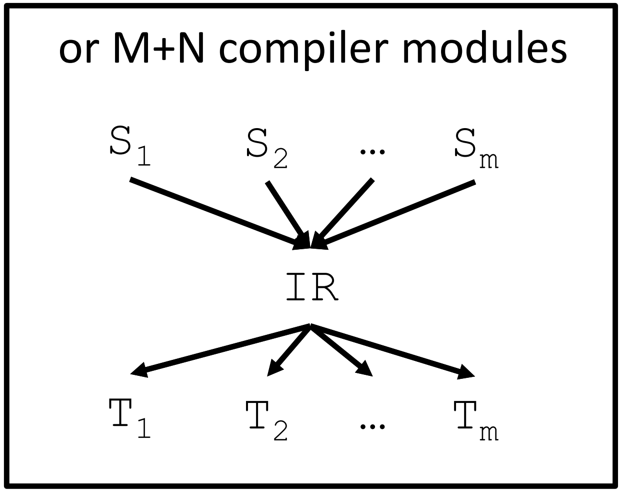
- Abstraction
  - Decouple compiler frontend from backend
- Analysis

M source languages

N target languages



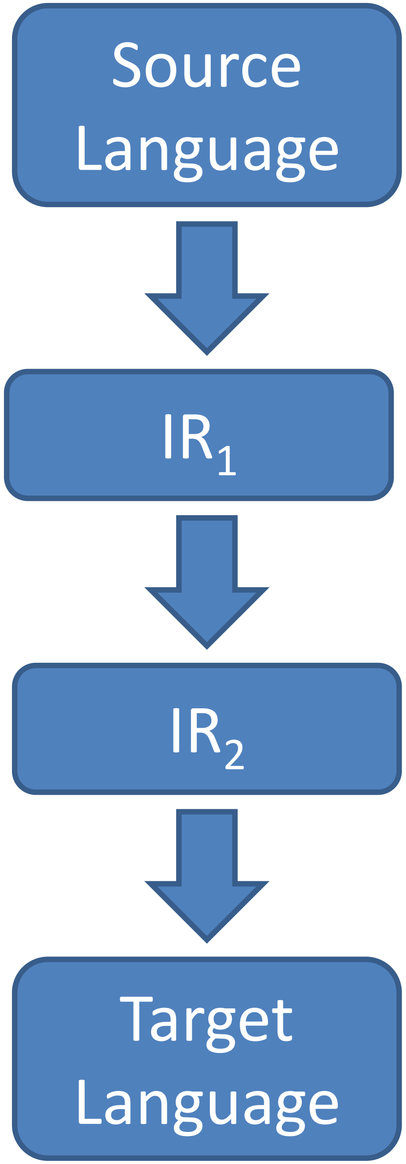
Write M×N compilers



or M+N compiler modules

# Why Use Intermediate Representations?

- Abstraction
  - Decouple compiler frontend from backend
  - Break down source language constructs into target language over several steps
- Analysis



# Why Use Intermediate Representations?

- Abstraction
  - Decouple compiler frontend from backend
  - Break down source language constructs into target language over several steps
- Analysis
  - Optimize programs

Improve…

- Runtime
- Memory usage
- Power usage
- Security

# Why Use Intermediate Representations?

- Abstraction
  - Decouple compiler frontend from backend
  - Break down source language constructs into target language over several steps
- Analysis
  - Optimize programs
  - Predict faults

<u>For example…</u>

- typechecking

But isn't this an analysis on the AST?

# Why Use Intermediate Representations?

- Abstraction
  - Deco
    front
  - Brea
    lang
    targ
    seve
- Analys
  - Opti
  - Pred

For example

sis

**ASTs are an example of an IR!!**

We've been talking about IR for a week!!

*M. Night Shyamalan, famous for (ill-considered) plot twists in movies he writes/directs*

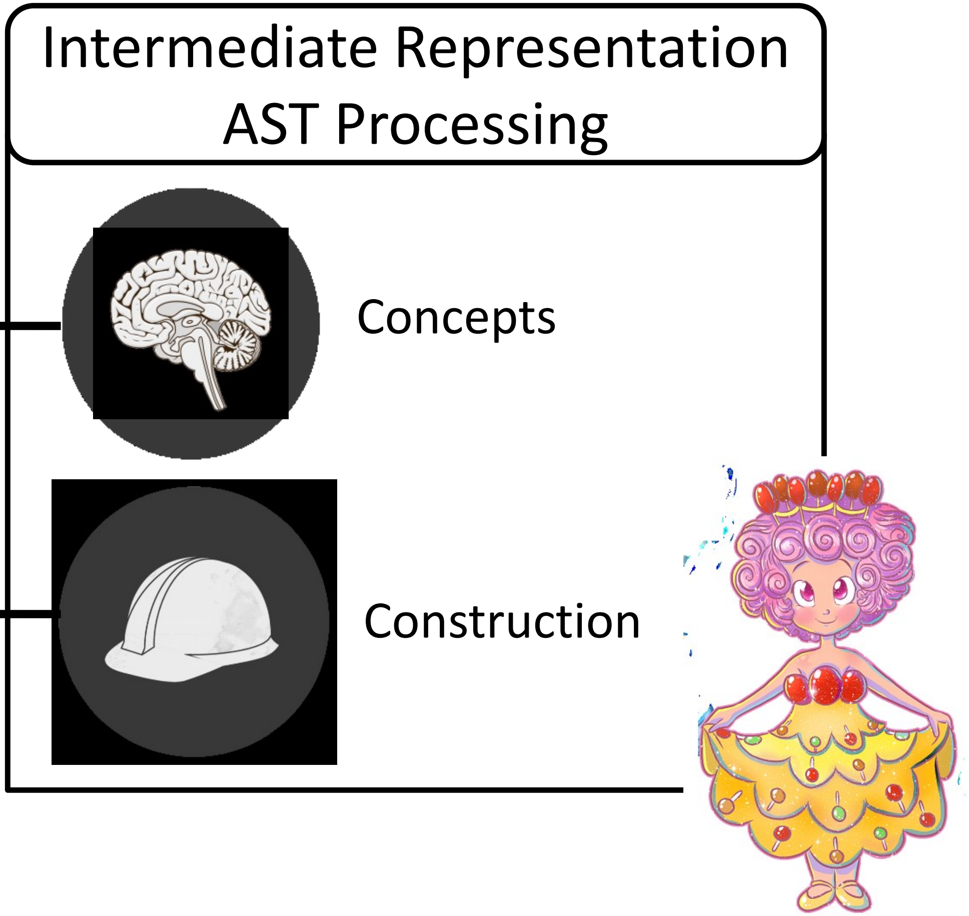
# Summary

Intermediate representations (like the AST) are a compiler’s “internal formats” for the program, aid in abstraction and analysis

# This Lecture

- What IRs are
- Why IRs are used
- Some IR forms

- IR Rewriting



Intermediate Representation
AST Processing

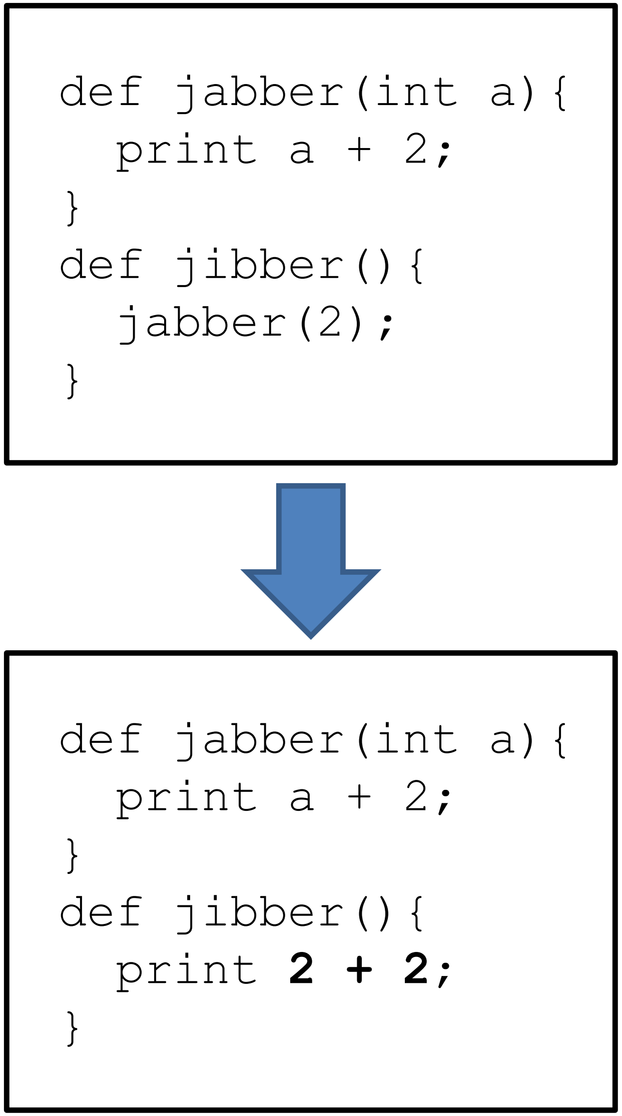
Concepts

Construction

# AST Transformations

What Optimizations can we do on the AST?

- Function inlining
  - Replace the function call with the entire body of the function
- Constant folding
  - Replace computations with their results, if statically available



```
def jabber(int a){
  print a + 2;
}
def jibber(){
  jabber(2);
}
```

```
def jabber(int a){
  print a + 2;
}
def jibber(){
  print 2 + 2;
}
```

# Function Inlining: Why/Why not?

Why

- *Might* simplify code
- Function calls are (relatively) expensive

Why Not

- Might *not* simplify code
- Likely to increase overall code size

# Function Inlining: How

pros and cons of function inling

- Place the function body into the caller's context
- Disambiguate name clashes
- Replace parameter passing with assignments
- Replace function return with assignments
- Remove function definition ?

```
int g(int b){
int a;
print(a+b)
}
int f(){
int a =4;
g(a)
}
```

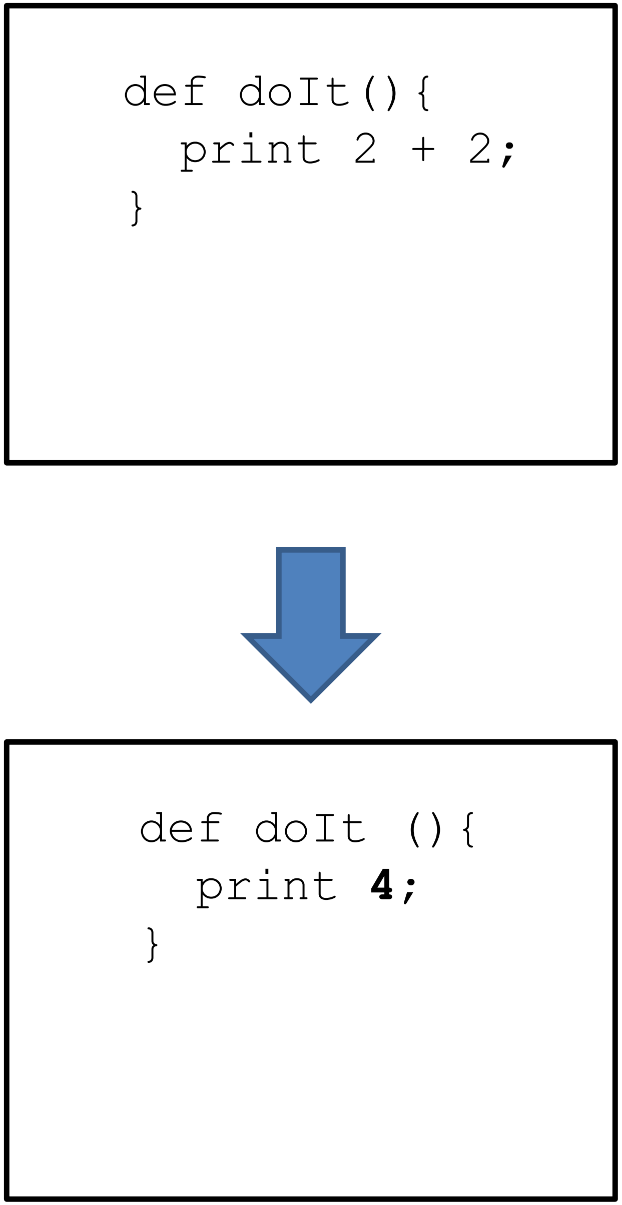
function inline clashes

```
int f(){
int a =4;
int a;
print(a+b);
}
```

# AST Transformations

What Optimizations can we do on the AST?

- Function inlining
  - Replace the function call with the entire body of the function
- Constant folding
  - Replace computations with their results, if statically available`



```
def doIt(){
  print 2 + 2;
}
```

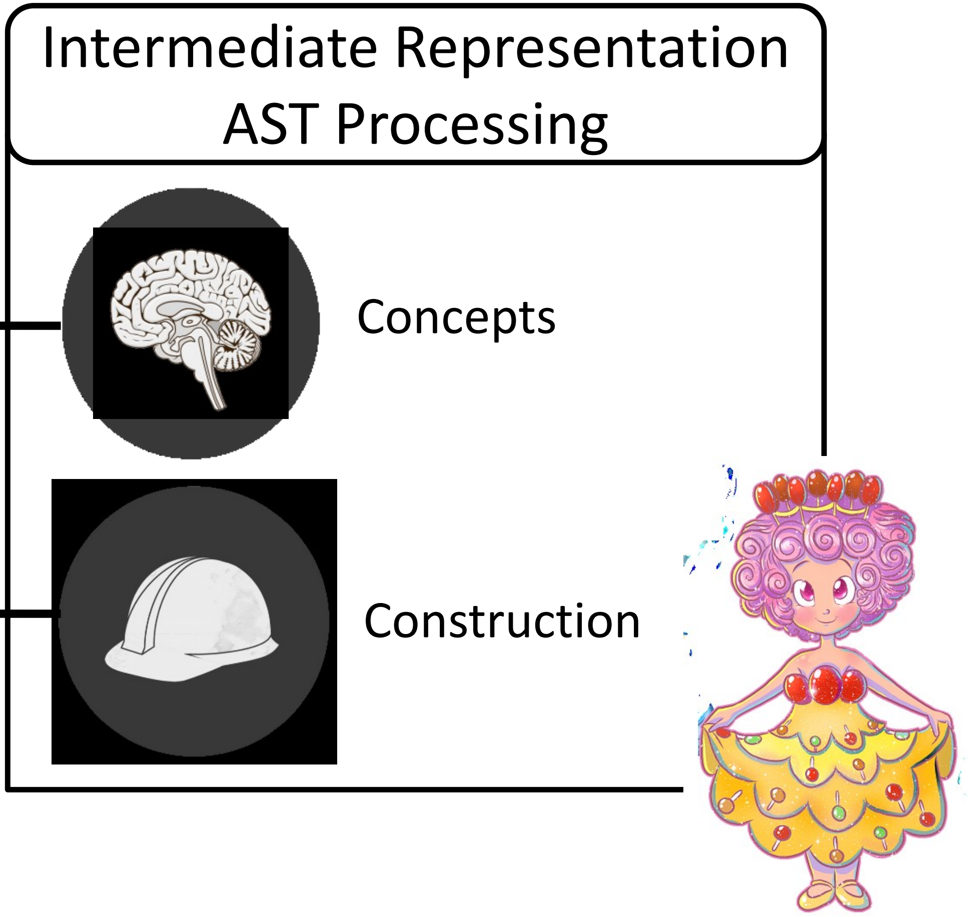
```
def doIt (){
  print 4;
}
```

# Summary

The AST can be a good target for analysis such as function inlining, typechecking, and constant folding

# This Lecture

- What IRs are
- Why IRs are used
- Some IR forms
  ^
  Other
- IR Rewriting



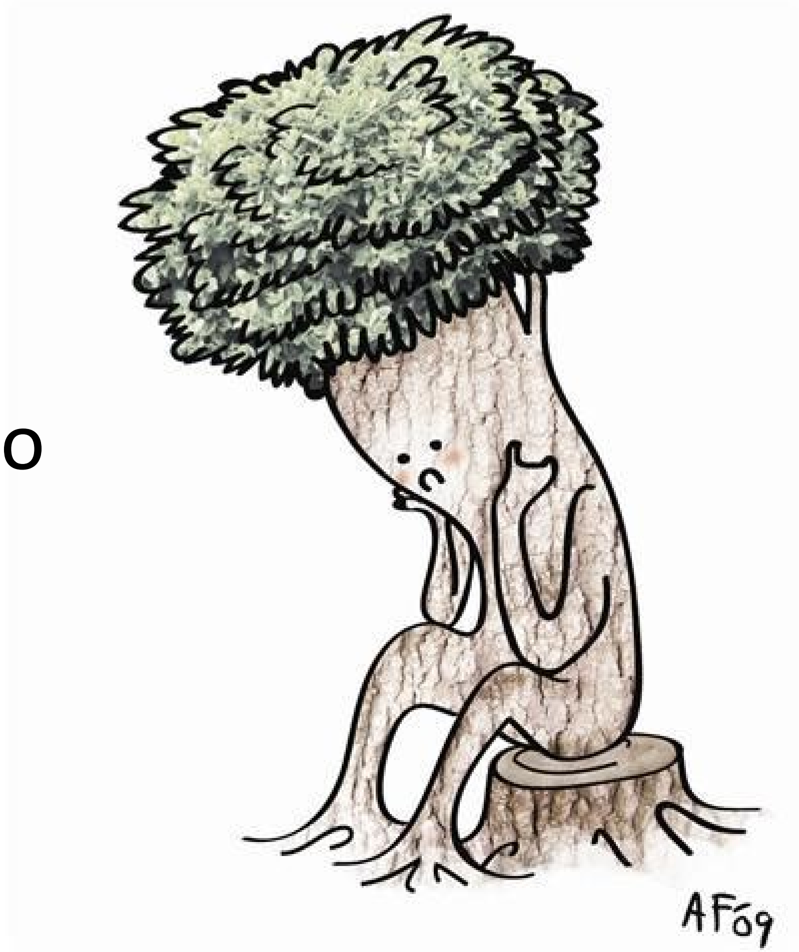
Intermediate Representation
AST Processing

Concepts

Construction

# Other IR Forms: Why?

- AST is great for some things, but not everything
  - Doesn’t represent control flow very well
- (NB: compilers could go directly from AST to machine code)



# Other IR Forms: Examples

- Structural
  - Abstract-Syntax Tree (AST)
  - Abstract Syntax DAG
- Linear
  - Three-Address Code (3AC)
  - Stack machine code
- Hybrid
  - Control-Flow Graph

Next time

Next next time