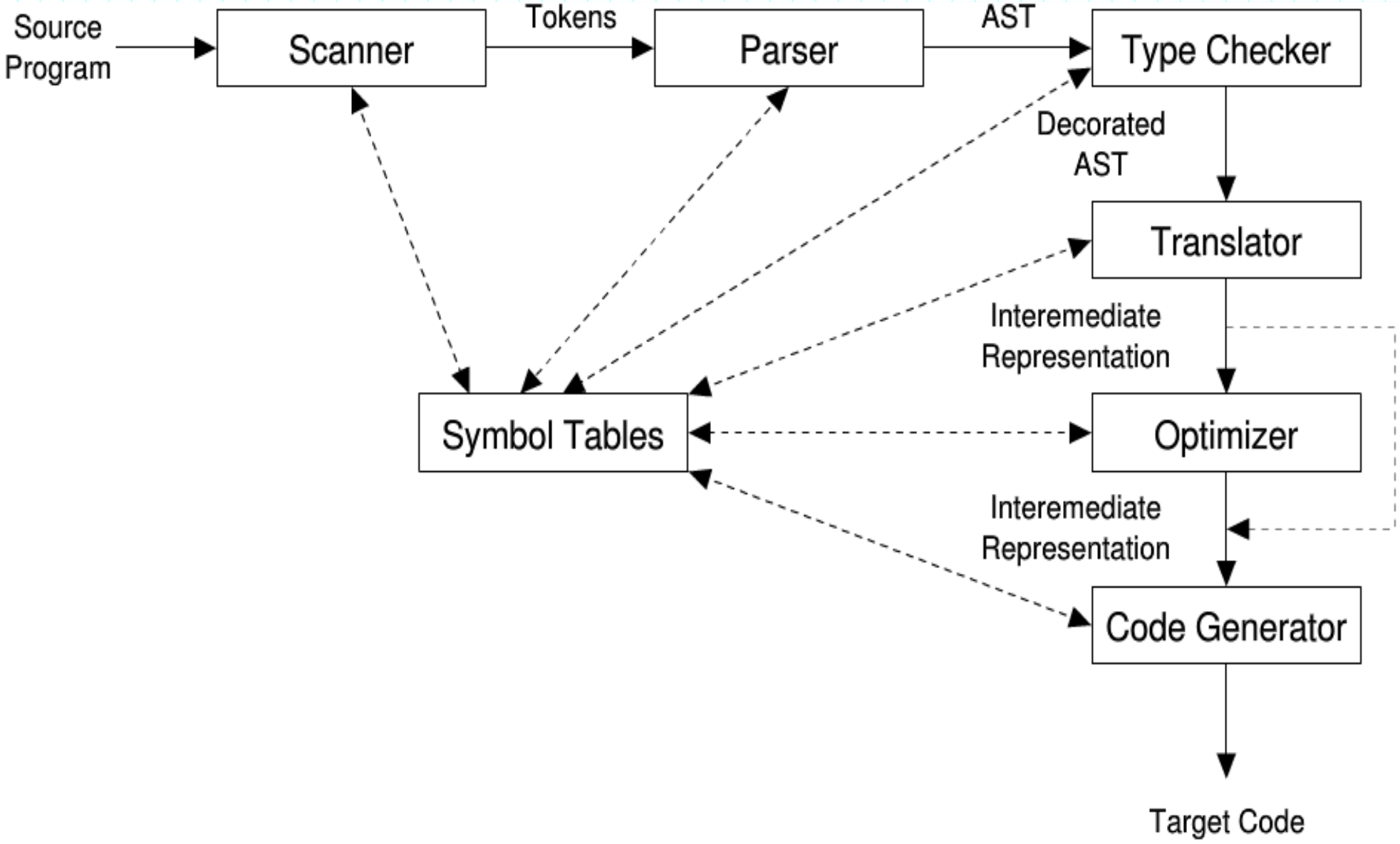
CSE431 – Translation of Computer Languages

- Java Virtual Machine

Doug Shook

Road Map

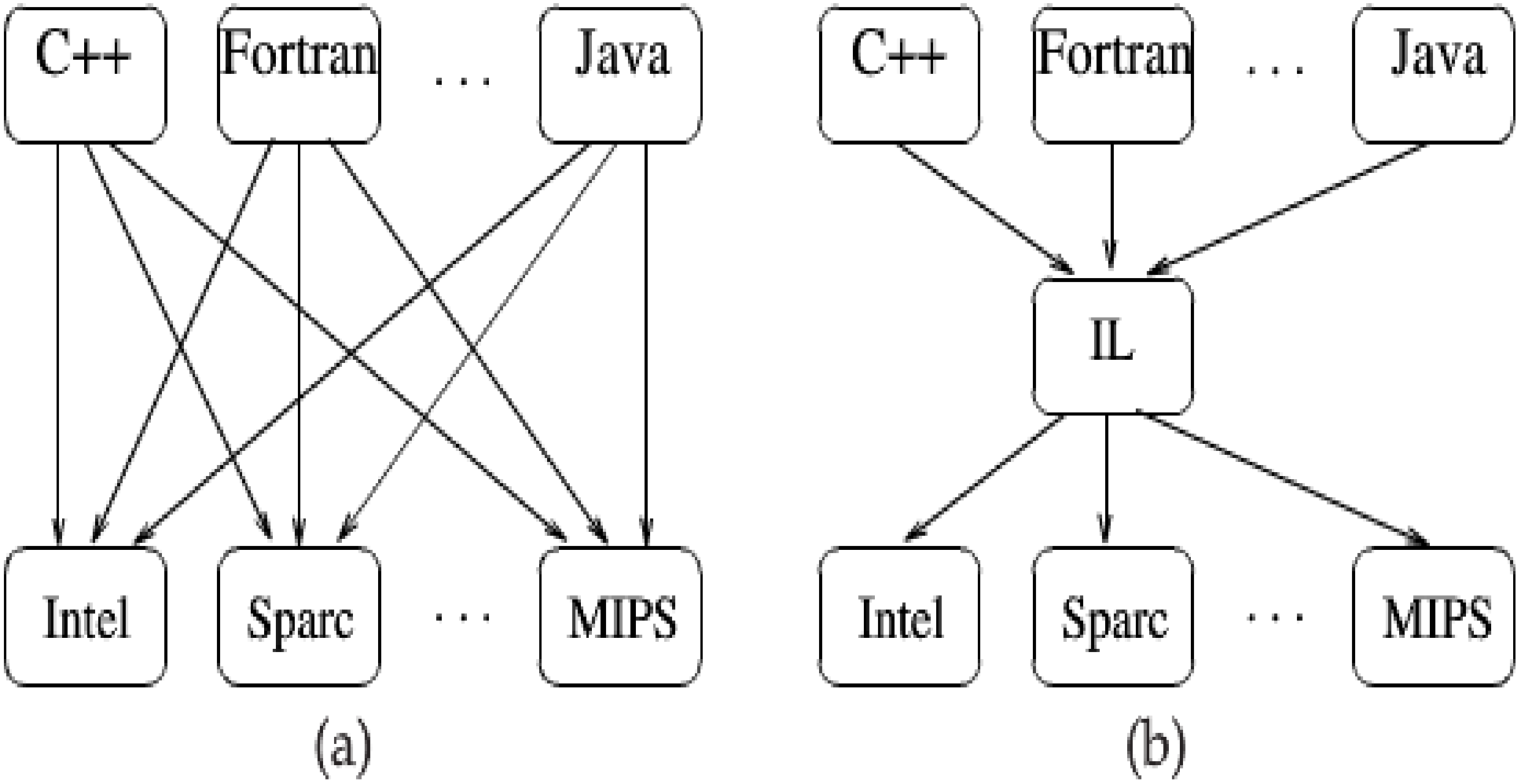

Source Program
Scanner
Tokens
Parser
AST
Type Checker
Decorated AST
Translator
Interemediate Representation
Symbol Tables
Optimizer
Interemediate Representation
Code Generator
Target Code

Intermediate Representations

- Abstract Syntax Tree
  - Useful for many purposes
  - Can be used for entire code generation process
    - Usually not the only step
  - Is this an intermediate representation?

- Intermediate languages
  - Designed to be neutral
  - Easy to translate
  - Abstracts away runtime issues

Middle-End


C++
Fortran
...
Java
Intel
Sparc
...
MIPS
(a)
C++
Fortran
...
Java
IL
Intel
Sparc
...
MIPS
(b)

Java Bytecode

- The two biggest design goals for Java:
  - Speed
  - Portability
- Other design considerations:
  - Compact
  - Easily verifiable
  - Ability to deal with runtime issues

Java Bytecode

- Stack-based
  - Vs. register based
  - Operand stack
- Instructions
  - Each opcode is 1 byte long

Runtime Issues

- Register Allocation
  - Nearly zero-address...
- Memory Management
- Optimizing for size
- Conversion to native code

Class Files

- Binary
- Platform independent
  - Run on any JVM

- 10 Sections
  - Constant Pool
  - This class
  - Super class
  - Interfaces
  - Fields
  - Methods

Constant Pools

- Used for constants, literals, types
- Consider loading an arbitrary integer constant
  - Or even worse, a string constant
  - What would the assembly instruction look like?
- Having a constant pool simplifies this
  - Put each constant in the pool
  - How to reference a constant?

Methods

- Each method call is encapsulated by a stack frame
  - Pushed onto the Java stack
- Stack frame contains:
  - Local variable pool
  - Local operand stack
  - Frame data
    - Constant pool resolution
    - Exception handling
    - Method return
- Method limits are verified upon class load

Verification

- Executed upon class load
  - Register / stack limits
  - Branches are always to a valid location
    - Includes return
  - Not possible to modify other stack frames
  - Types are respected
  - Access restrictions are respected
  - It must not be possible to reach the same instruction with different stack sizes
    - Why?

Jasmin

- Java Bytecode is not human readable
  - Though there are ways...
- Jasmin *assembles* into java bytecode
  - Manages constant pool
  - Instruction set is very similar to bytecode instructions

Jasmin example

```
.class public hello
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 2
    .limit locals 1

    getstatic java/lang/System/out
       Ljava/io/PrintStream;
    ldc "Hello World!"
    invokevirtual java/io/PrintStream println
       (Ljava/lang/Object;)V
    return
.end method
```

JVM Types

Type	JVM designation
`boolean`	`Z`
`byte`	`B`
`double`	`D`
`float`	`F`
`int`	`I`
`long`	`J`
`short`	`S`
`void`	`V`
Reference type *t*	L*t*;
Array of type *a*	[*a*

Stack Instructions

- aload
- astore
- istore, iload
- Similar for other types

- Constants
  - ldc 8
    - In Jasmin this loads 8
    - What does this do in bytcode?
  - iconst_0

Integer Arithmetic

- How do these work in a stack-based language?

- iadd
- imul
- iand
- ishr, ishrl
- Etc.

Invoking Methods

- Typically either:
  - invokevirtual
  - invokestatic
- Top of stack should be:
  - Instance (if necessary)
  - Parameter list
    - Where do these end up?
- Method signature is supplied as an argument
- Return values are left on the stack
  - Transferred where?

Fields

- Must be able to get and set:
  - putfield
    - Object
    - Value to store
  - getfield
    - Object
    - Pushes value to stack

Control Flow

- Bytecode
  - Unconditional
    - goto, jsr, ret
  - Conditional
    - ifgt, iflt, ifeq, ifneq ifle, ifge
      - All compare against 0

- Jasmin allows labels:

```
looptop:
    ...
    goto looptop
```

Stack/Memory Management

- Stack
  - dup
  - pop
  - swap
- Memory
  - new
    - Operand points to a type (where?)
    - Must explicitly call constructor on new object
  - newarray
    - TOS is array size
    - Operand must be a type

Demo

- Java Bytecode
- Jasmin

Exercises

- Give a java expression which is functionally equivalent to the jasmin fragment below. Assume that registers 1,2,3 correspond to integer variables x,y,z.

```
iconst_2
iload_1
iload_2
idiv
iconst_5
iload_3
iload_2
iadd
idiv
isub
imul
```

Exercises

- The following bytecode fragment has three problems which will not pass bytecode verification. What are they?

```
.method public static f(I)I
  .limit locals 1
  .limit stack 2
  iload_0
  dup
  istore_1
A:
  ifle B
  goto C
B:
  iconst_0
  iload_1
  iload_0
  iadd
  iadd
C:
  iconst_m1
  ireturn
.end method
```

Exercises

- Write Jasmin code for a fibonacci method. This method should take in a single argument, n, and return the n*th* fibonacci number.

fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
etc.

Putting it all together

- For the following:
  - AST -> Type Check -> Reachability -> Jasmin

```
public boolean withdraw(float amount) {
    if (this.balance – amount <= 0) {
       System.out.println(“Insufficient funds”);
       return false;
    }
    else {
       this.balance = this.balance – amount;
       return true;
    }
}
```