Java Virtual Machine

A7 due Monday
Final Dec. 14

Implementation of Java Abstraction

You write:

Java code

?

Processor (CPU) runs:

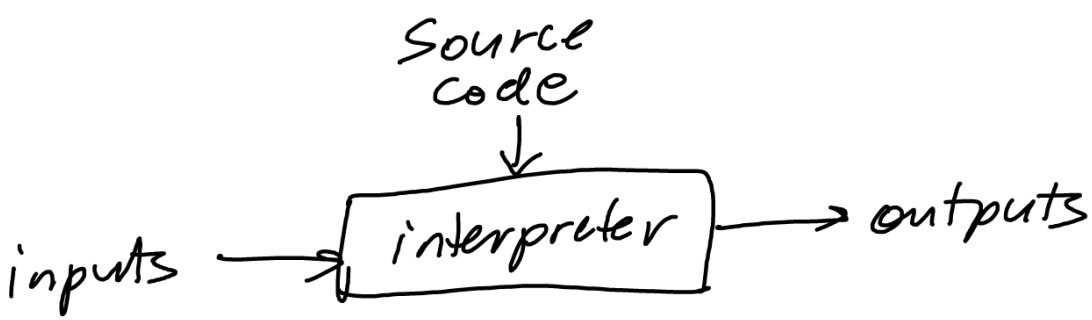
machine code

Two way to handle code:

1. interpret — interpreter
2. compile — compiler

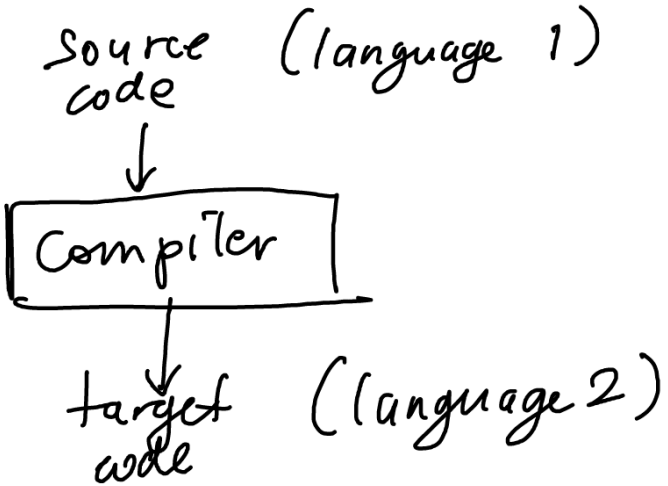
Both are used by Java.

Interpreting



- software interpreter
  can handle any language
  portable to any computer
- much slower than machine code. 10–1000x
  Python: 50x
- processor is an interpreter (hardware)
  - limited to machine code.

Compiling

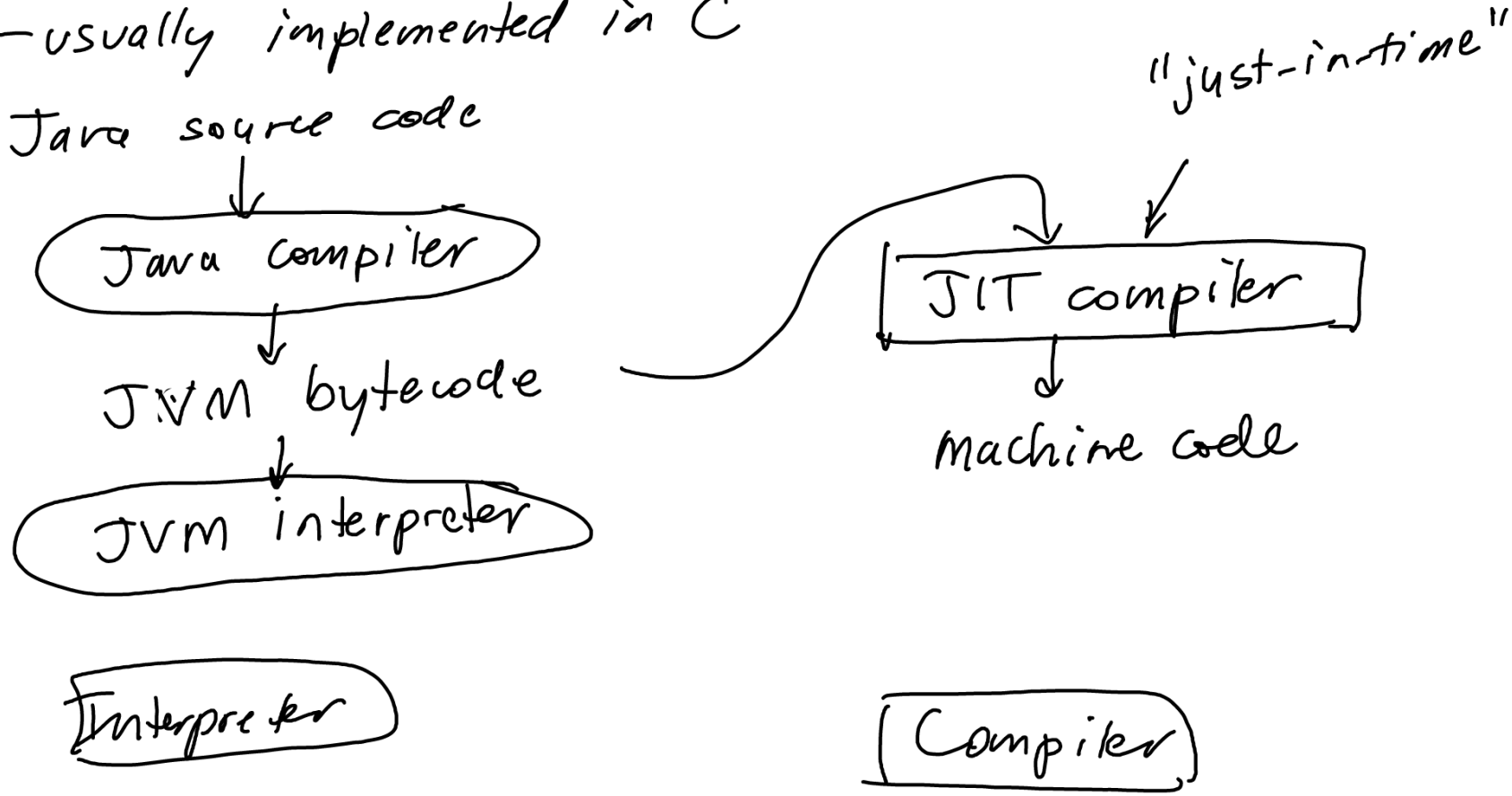


- target language = machine code
- efficient, can run if you have interpreter for target
- hard to retarget

Java: interprets & compiles.

Interpreter: Java Virtual Machine (JVM)

- runs on many architectures (portable)
- usually implemented in C



Two machine-code programs running:

1. JVM interpreter
2. output of JIT

compiling is expensive ⇒ not all code gets JIT-compiled.

- JVM interpreter profiles execution to identify hot spots
- JIT compiler invoked on hot spots
  e.g. HotSpot compiler

JVM bytecode

Compiled code is in .class files.

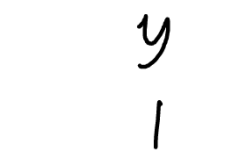
JVM bytecode is stack-based language.

- computation uses an operand stack (instructions)

$x = y + 1$ ⇝

```
iload 4
iconst_1
iadd
istore 3
```

```
// y
// 1
// x
```



instructions encoded as bytes in .class file

method dispatch

```
Animal x = ...
x.eat(5)
```

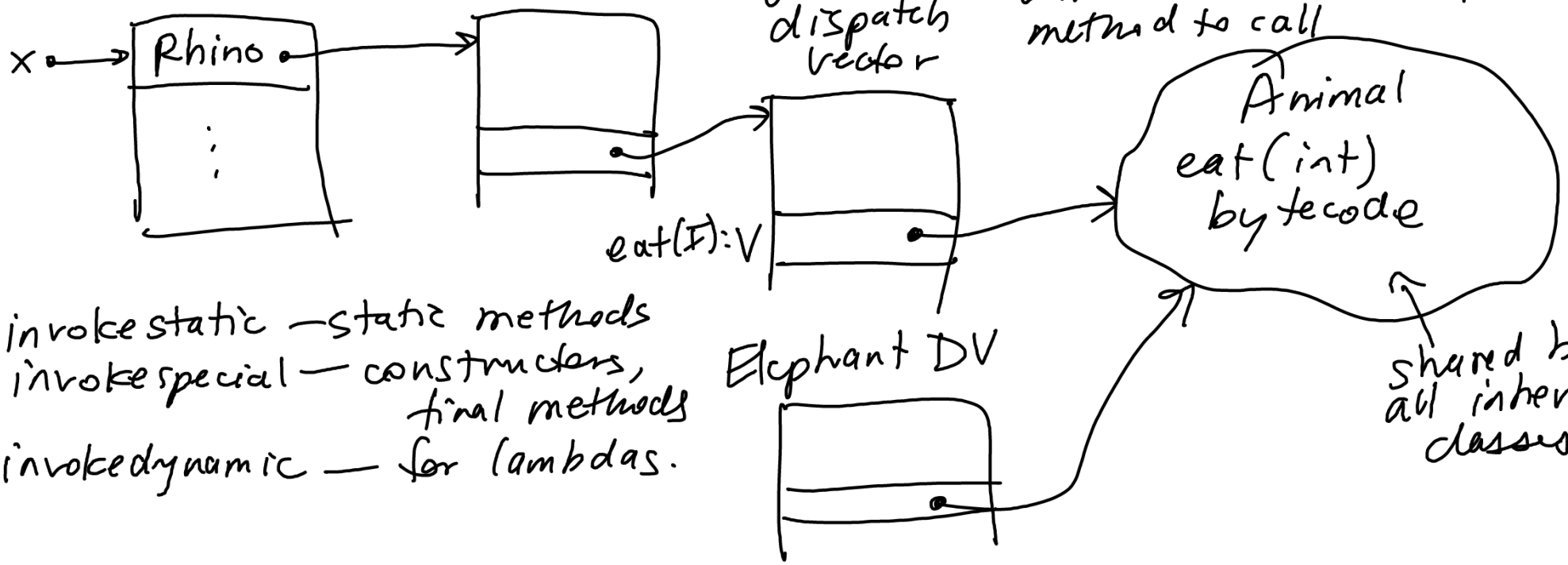
~~> invokevirtual

encodes signature ↓ "eat(I):V"

actually: #23 (constant pool index)

JVM knows which overloaded method to call

```
x = new Rhino();
```



invokestatic — static methods

invokespecial — constructors, final methods

invokedynamic — for lambdas.

Bytecode verification

JVM type-checks bytecode
— allows more efficient compilation.

```
Elephant x = ...
x.f = 0;
```

```
aload   1   // x
iconst_0
putfield #7  // f
```

generic type erasure → JVM doesn't check!

no run-time check that x is an Elephant

Runtime Code Generation

Java compiler is a library

⇒ you can generate code at run time

API: javax.tools.ToolProvider.getSystemJavaCompiler()

$y = f(x)$ → Java code (faster) + class file —(class loader)→ running class

Goal: make you comfortable with idea → code

Software engineering +

- Specifications
- invariants
- testing
- concurrency
- GUI + event-driven

Theory

- Algorithms
- Data structures
- Complexity

Beyond 2110 ?

- ACSU, WICC, ...
- research
- project teams
- teaching staff