

The Hardware/Software Interface

CSE351 Winter 2013

Java vs. C

Roadmap

Data & addressing
 Integers & floats
 Machine code & C
 x86 assembly
 programming
 Procedures &
 stacks
 Arrays & structs
 Memory & caches
 Processes
 Virtual memory
 Memory allocation

Java vs. C

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

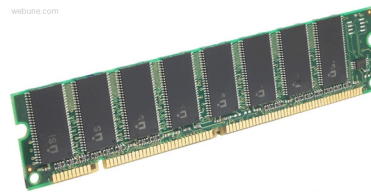
Assembly language:

```
get_mpg:
    pushq    %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

Computer system:



OS:



Java vs. C

■ Reconnecting to Java

- Back to CSE143!
- But now you know a lot more about what really happens when we execute programs

■ We've learned about the following items in C; now we'll see what they look like for Java:

- Representation of data
- Pointers / references
- Casting
- Function / method calls
- Runtime environment
- Translation from high-level code to machine code

Meta-point to this lecture

- None of the data representations we are going to talk about are *guaranteed* by Java
- In fact, the language simply provides an *abstraction*
- We can't easily tell how things are really represented
- But it is important to understand *an* implementation of the lower levels – useful in thinking about your program

Data in Java

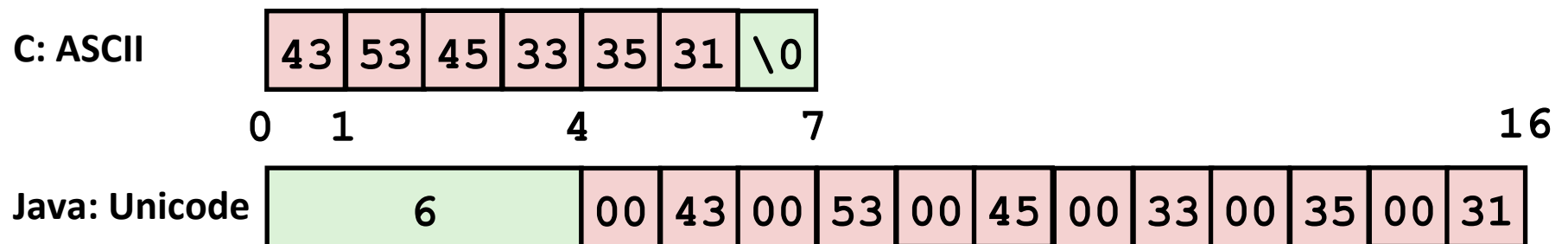
- **Integers, floats, doubles, pointers – same as C**
 - Yes, Java has pointers – they are called ‘references’ – however, Java references are much more constrained than C’s general pointers
- **Null is typically represented as 0**
- **Characters and strings**
- **Arrays**
- **Objects**

Data in Java

■ Characters and strings

- Two-byte Unicode instead of ASCII
 - Represents most of the world's alphabets
- String not bounded by a '\0' (null character)
 - Bounded by hidden length field at beginning of string

the string 'CSE351':

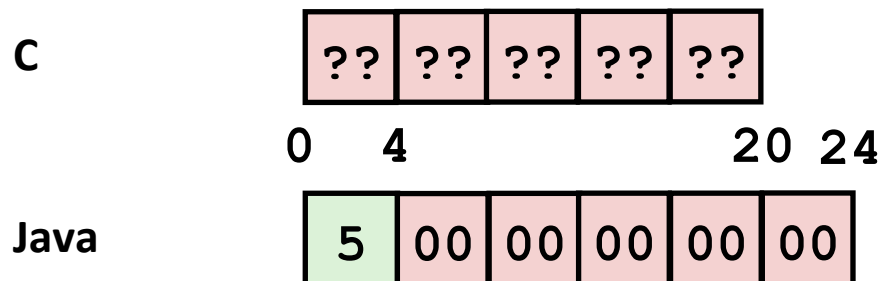


Data in Java

■ Arrays

- Every element initialized to 0
- Bounds specified in hidden fields at start of array (int – 4 bytes)
 - *array.length* returns value of this field
- Every access triggers a bounds-check
 - Code is added to ensure the index is within bounds
 - Exception if out-of-bounds

int array[5]:



Data structures (objects) in Java

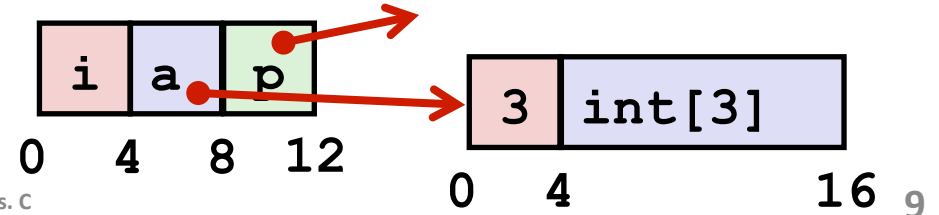
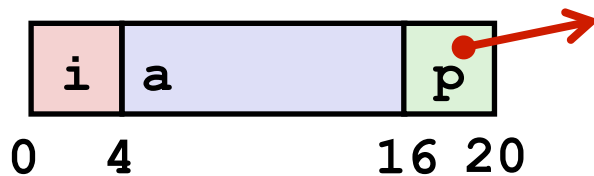
- **Objects (structs) can only include primitive data types**
 - Include complex data types (arrays, other objects, etc.) using *references*

```
C
struct rec {
    int i;
    int a[3];
    struct rec *p;
};
```

```
Java
class Rec {
    int i;
    int[] a = new int[3];
    Rec p;
...
};
```

```
struct rec *r = malloc(...);
struct rec r2;
r->i = val;
r->a[2] = val;
r->p = &r2;
```

```
r = new Rec;
r2 = new Rec;
r.i = val;
r.a[2] = val;
r.p = r2;
```

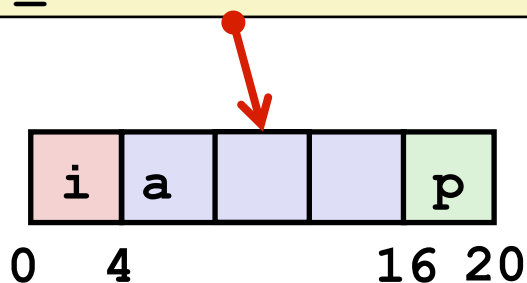


Pointers/References

- Pointers in C can point to any memory address
- References in Java can only point to an object
 - And only to its first element – not to the middle of it

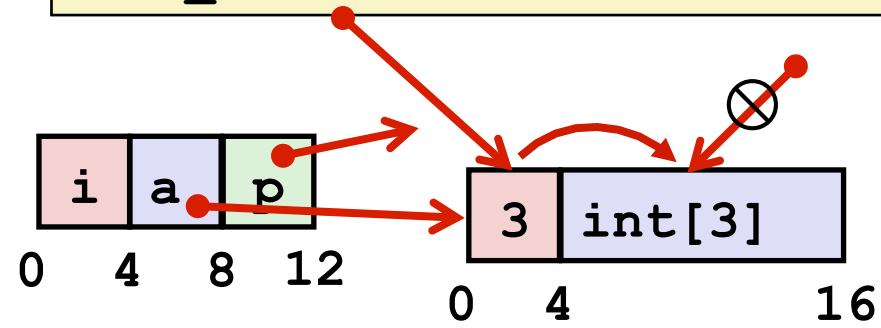
C

```
struct rec {
    int i;
    int a[3];
    struct rec *p;
};
some_fn(&(r.a[1])) //ptr
```



Java

```
class Rec {
    int i;
    int[] a = new int[3];
    Rec p;
...
};
some_fn(r.a, 1) //ref & index
```



Pointers to fields

- In C, we have “->” and “.” for field selection depending on whether we have a pointer to a struct or a struct
 - (*r).a is so common it becomes r->a
- In Java, *all variables are references to objects*
 - We always use r.a notation
 - But really follow reference to r with offset to a, just like C’s r->a

Casting in C

- We can cast any pointer into any other pointer

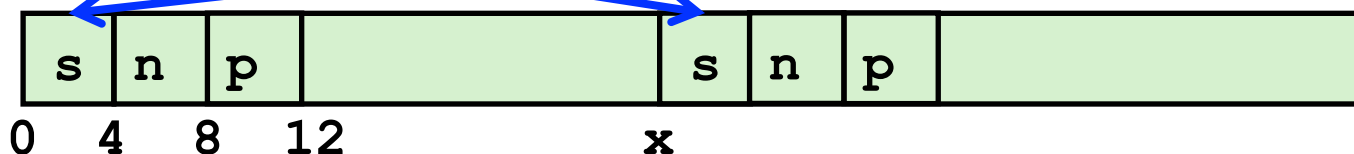
```

struct BlockInfo {
    int sizeAndTags;
    struct BlockInfo* next;
    struct BlockInfo* prev;
};
typedef struct BlockInfo BlockInfo;
...
int x;
BlockInfo *b;
BlockInfo *newBlock;
...
newBlock = (BlockInfo *) ( (char *) b + x );
...

```

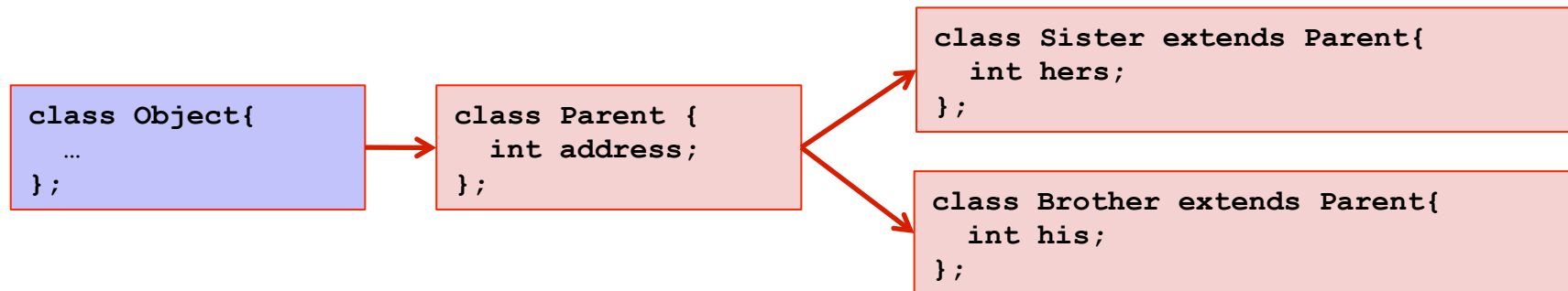
Cast b into char pointer so that you can add byte offset without scaling

Cast back into BlockInfo pointer so you can use it as BlockInfo struct



Casting in Java

■ Can only cast compatible object references



```

// Parent is a super class of Brother and Sister, which are siblings
Parent    a = new Parent();
Sister    xx = new Sister();
Brother   xy = new Brother();
Parent    p1 = new Sister(); // ok, everything needed for Parent
                                     // is also in Sister
Parent    p2 = p1;           // ok, p1 is already a Parent
Sister    xx2 = new Brother(); // incompatible type - Brother and
                                     // Sisters are siblings
Sister    xx3 = new Parent(); // wrong direction; elements in Sister
                                     // not in Parent (hers)
Brother   xy2 = (Brother) a; // run-time error; Parent does not contain
                                     // all elements in Brother (his)
Sister    xx4 = (Sister) p2; // ok, p2 started out as Sister
Sister    xx5 = (Sister) xy; // inconvertible types, xy is Brother

```

How is this implemented / enforced?

Creating objects in Java

```
class Point {  
    double x;  
    double y;  
  
    Point() {  
        x = 0;  
        y = 0;  
    }  
  
    boolean samePlace(Point p) {  
        return (x == p.x) && (y == p.y);  
    }  
  
}  
...  
Point newPoint = new Point();  
...
```

fields

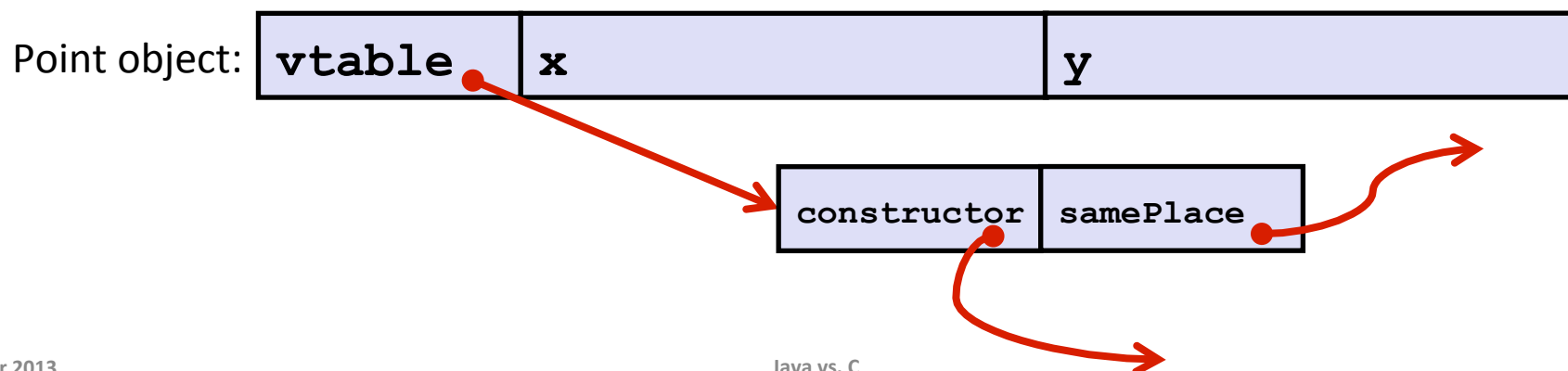
constructor

method

creation

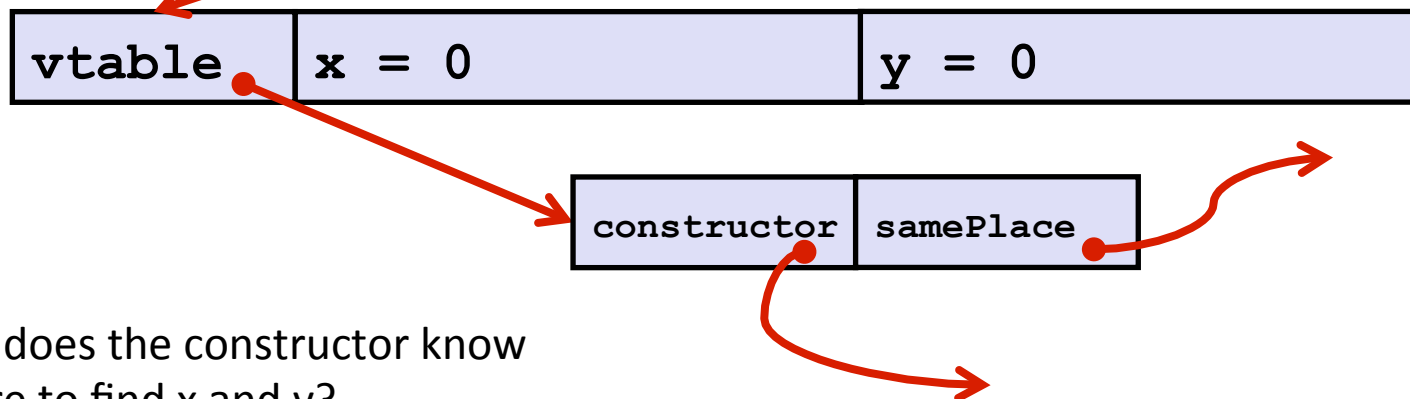
Creating objects in Java

- “new”
 - Allocates space for data fields
 - Adds pointer in object to “virtual table” or “vtable” for class
 - vtable is shared across all objects in the class!
 - Includes space for “static fields” and pointers to methods’ code
 - Returns reference (pointer) to new object in memory
 - Runs “constructor” method
- **The new object is eventually garbage collected if all references to it are discarded**



Initialization

- `newPoint`'s fields are initialized starting with the vtable pointer to the vtable for this class
- The next step is to call the 'constructor' for this object type
- Constructor code is found using the 'vtable pointer' and passed a pointer to the newly allocated memory area for `newPoint` so that the constructor can set its `x` and `y` to 0
 - `Point.constructor()`



How does the constructor know where to find `x` and `y`?

Java Methods

- **Methods in Java are just functions (as in C) but with an extra argument: a reference to the object whose method is being called**
 - E.g., `newPoint.samePlace` calls the `samePlace` method with a pointer to `newPoint` (called 'this') and a pointer to the argument, `p` – in this case, both of these are pointers to objects of type `Point`
 - Method becomes `Point.samePlace(Point this, Point p)`
 - `return x==p.x && y==p.y;` becomes something like:
`return (this->x==p->x) && (this->y==p->y);`

Subclassing

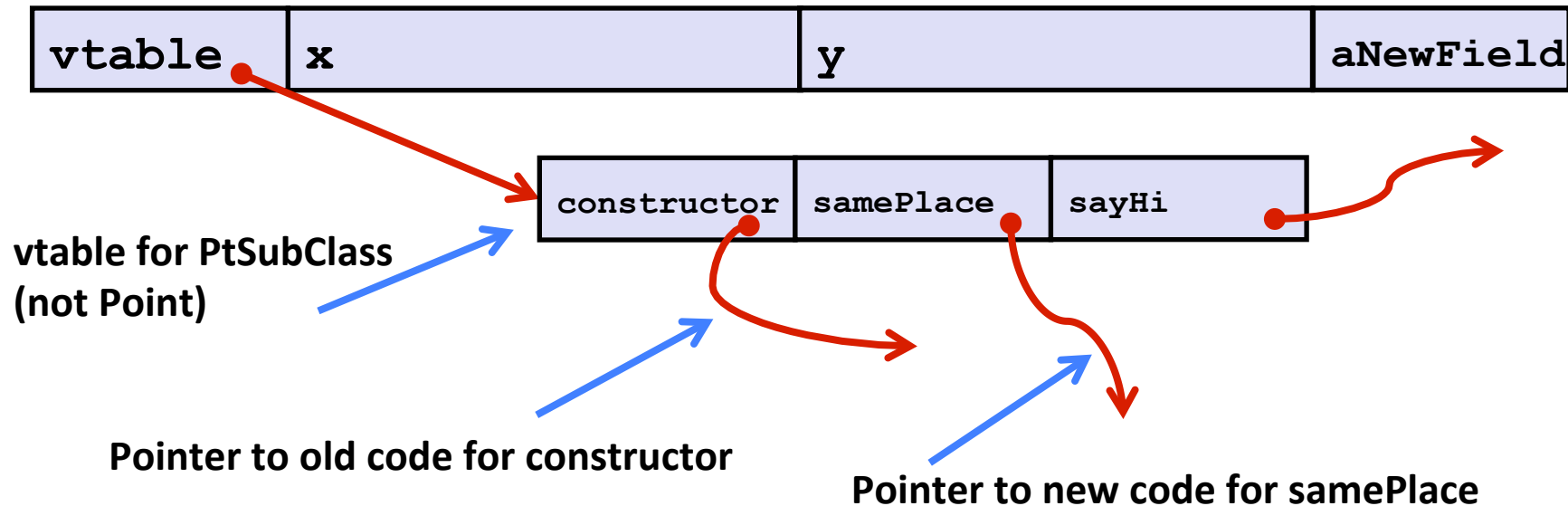
```
class PtSubClass extends Point{
    int aNewField;
    boolean samePlace(Point p2) {
        return false;
    }
    void sayHi() {
        System.out.println("hello");
    }
}
```

- **Where does “aNewField” go?**
 - At end of fields of Point – allows easy casting from subclass to parent class!
- **Where does pointer to code for two new methods go?**
 - To override “samePlace”, write over old pointer
 - Add new pointer at end of table for new method “sayHi”

Subclassing

```
class PtSubClass extends Point{
    int aNewField;
    boolean samePlace(Point p2) {
        return false;
    }
    void sayHi() {
        System.out.println("hello");
    }
}
```

aNewField tacked on at end



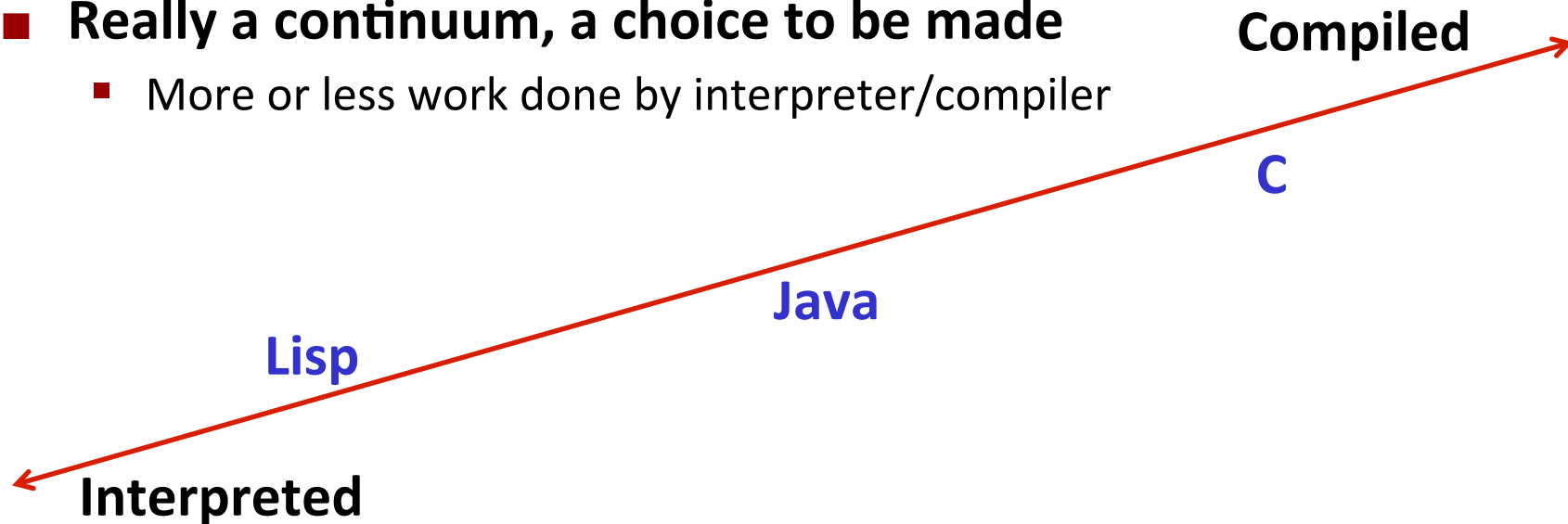
Implementing Programming Languages

- Many choices in how to implement programming models
- We've talked about compilation, can also *interpret*
 - Execute line by line in original source code
 - Less work for compiler – all work done at run-time
 - Easier to debug – less translation
 - Easier to run on different architectures – runs in a simulated environment that exists only inside the *interpreter* process
- Interpreting languages has a long history
 - Lisp – one of the first programming languages, was interpreted
- Interpreted implementations are very much with us today
 - Python, Javascript, Ruby, Matlab, PHP, Perl, ...

Interpreted vs. Compiled

- Really a continuum, a choice to be made

- More or less work done by interpreter/compiler

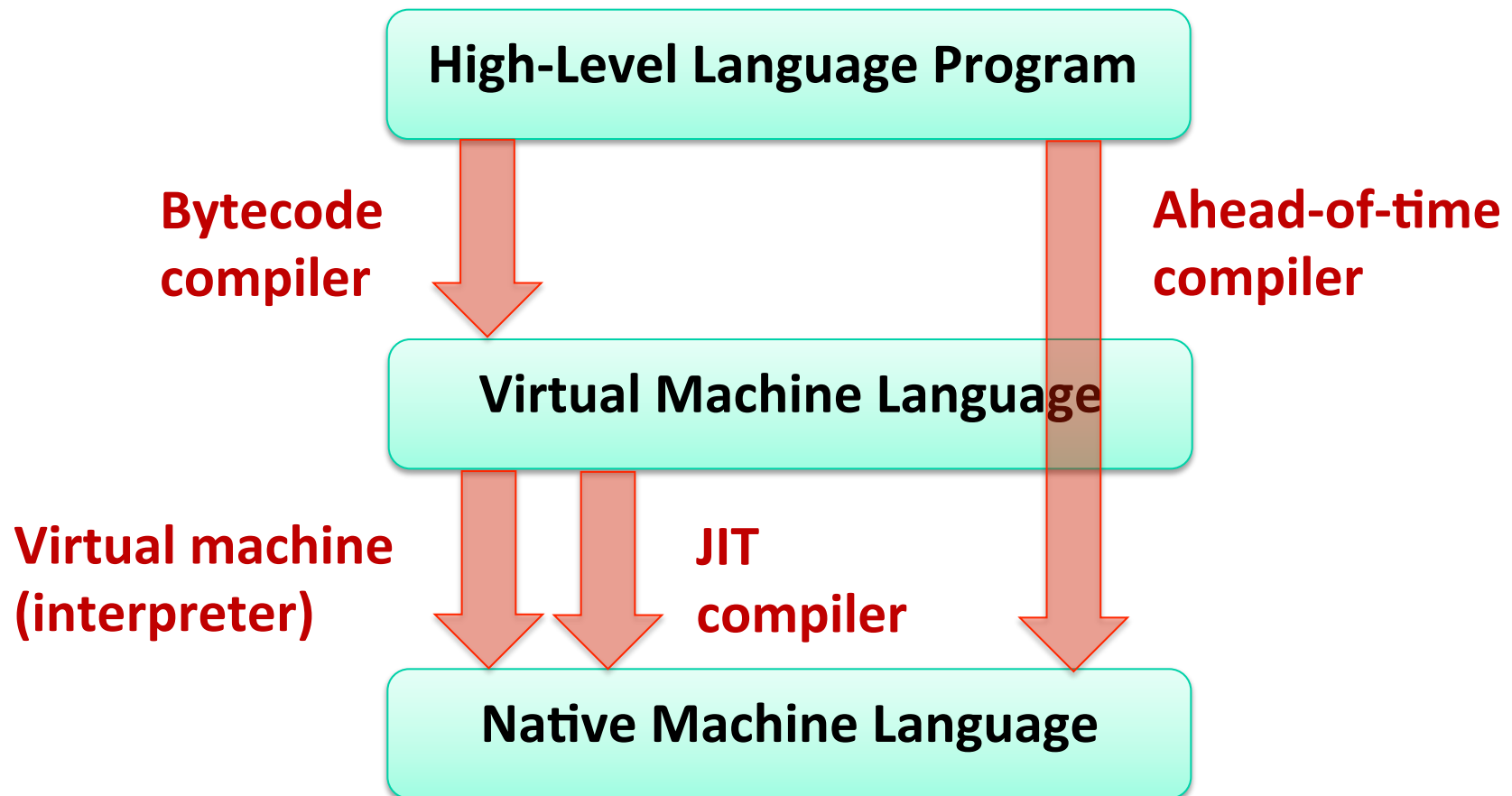


- Java programs are usually run by a ***virtual machine***

- VMs interpret an intermediate, “partly compiled” language called *bytecode*

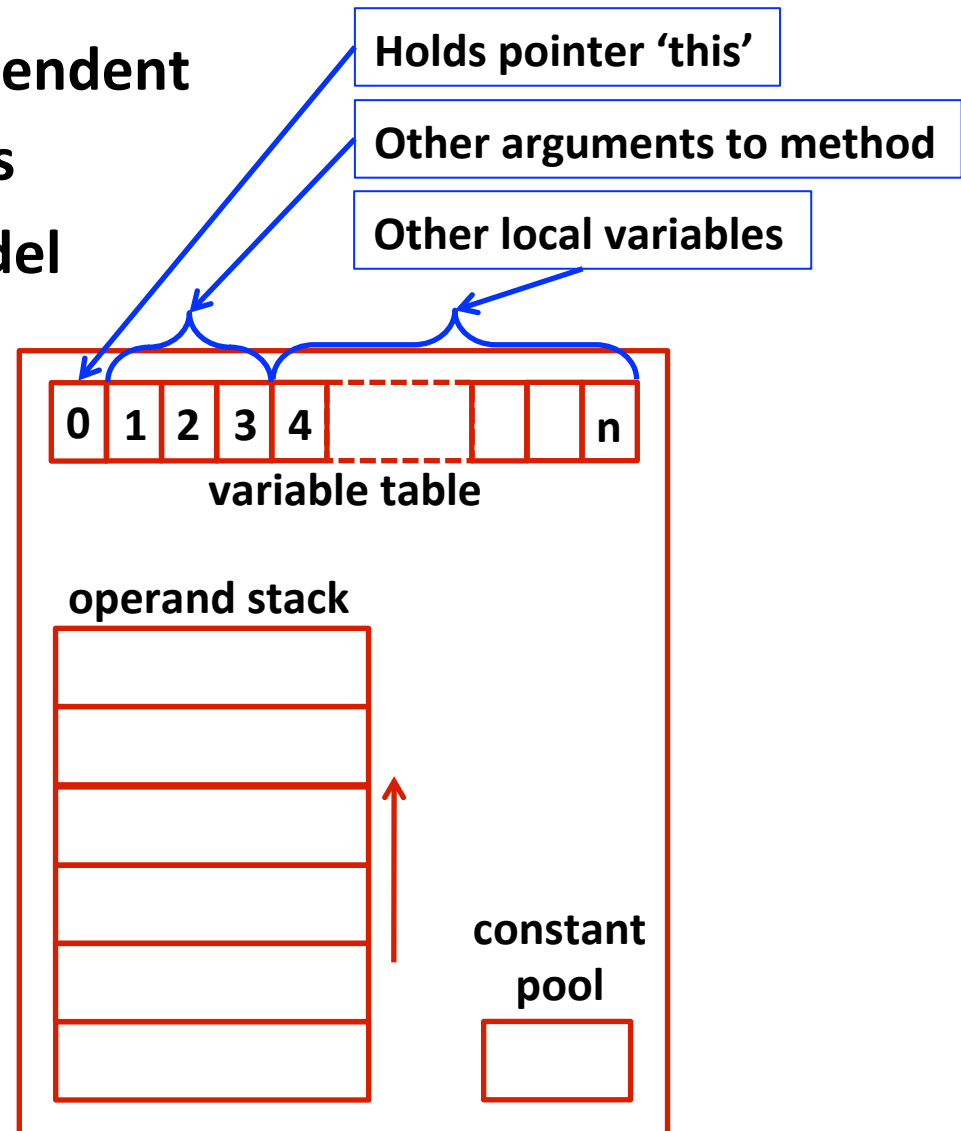
- Java can also be compiled ahead of time (just as a C program is) or at runtime by a ***just-in-time (JIT) compiler***

Virtual Machine Model



Java Virtual Machine

- **Makes Java machine-independent**
- **Provides strong protections**
- **Stack-based execution model**
- **There are many JVMs**
 - Some interpret
 - Some compile into assembly
 - Usually implemented in C



JVM Operand Stack Example

'i' stands for integer,
'a' for reference,
'b' for byte,
'c' for char,
'd' for double, ...

No knowledge
of registers or
memory locations
(each instruction
is 1 byte – bytecode)

```
iload 1 // push 1st argument from table onto stack
iload 2 // push 2nd argument from table onto stack
iadd    // pop top 2 elements from stack, add together, and
        // push result back onto stack
istore 3 // pop result and put it into third slot in table
```

```
mov 0x8001, %eax
mov 0x8002, %edx
add %edx, %eax
mov %eax, 0x8003
```

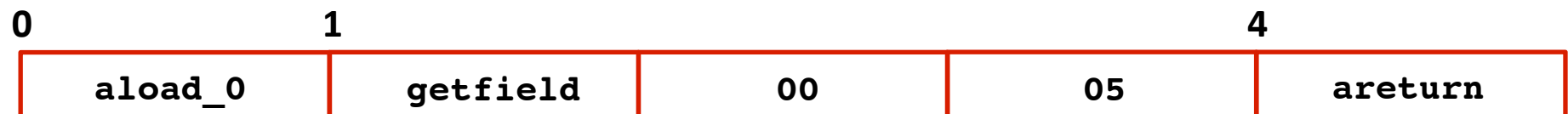

A Simple Java Method

```
Method java.lang.String getEmployeeName()

0 aload 0          // "this" object is stored at 0 in the var table

1 getfield #5 <Field java.lang.String name> // takes 3 bytes
    // pop an element from top of stack, retrieve its
    // specified field and push the value onto stack.
    // "name" field is the fifth field of the class

4 areturn         // Returns object at top of stack
```



In the .class file:

2A	B4	00	05	B0
----	----	----	----	----

http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

Class File Format

- **Every class in Java source code is compiled to its own class file**
- **10 sections in the Java class file structure:**
 - Magic number: 0xCAFEBAE (legible hex from James Gosling – Java’s inventor)
 - Version of class file format: the minor and major versions of the class file
 - Constant pool: set of constant values for the class
 - Access flags: for example whether the class is abstract, static, etc.
 - This class: The name of the current class
 - Super class: The name of the super class
 - Interfaces: Any interfaces in the class
 - Fields: Any fields in the class
 - Methods: Any methods in the class
 - Attributes: Any attributes of the class (for example the name of the source file, etc.)
- **A *.jar* file collects together all of the class files needed for the program, plus any additional resources (e.g. images)**

Disassembled Java Bytecode

```
javac Employee.java
javap -c Employee > Employee.bc
```

Compiled from Employee.java

```
class Employee extends java.lang.Object {
public Employee(java.lang.String,int);
public java.lang.String getEmployeeName();
public int getEmployeeNumber();
}
```

Method Employee(java.lang.String,int)

```
0  aload_0
1  invokespecial #3 <Method java.lang.Object()>
4  aload_0
5  aload_1
6  putfield #5 <Field java.lang.String name>
9  aload_0
10 iload_2
11 putfield #4 <Field int idNumber>
14 aload_0
15 aload_1
16 iload_2
17 invokespecial #6 <Method void
    storeData(java.lang.String, int)>
20 return
```

Method java.lang.String getEmployeeName()

```
0  aload_0
1  getfield #5 <Field java.lang.String name>
4  areturn
```

Method int getEmployeeNumber()

```
0  aload_0
1  getfield #4 <Field int idNumber>
4  ireturn
```

Method void storeData(java.lang.String, int)

...

Java vs. C

Other languages for JVMs

- **JVMs run on so many computers that compilers have been built to translate many other languages to Java bytecode:**
 - AspectJ, an aspect-oriented extension of Java
 - ColdFusion, a scripting language compiled to Java
 - Clojure, a functional Lisp dialect
 - Groovy, a scripting language
 - JavaFX Script, a scripting language targeting the Rich Internet Application domain
 - JRuby, an implementation of Ruby
 - Jython, an implementation of Python
 - Rhino, an implementation of JavaScript
 - Scala, an object-oriented and functional programming language
 - And many others, even including C

Microsoft's C# and .NET Framework

- C# has similar motivations as Java
- Virtual machine is called the Common Language Runtime; Common Intermediate Language is the bytecode for C# and other languages in the .NET framework

