- Applet java appaiono di frequente nelle pagine web
- Come funziona l'interprete contenuto in ogni browser di un certo livello?
- Per approfondire il funzionamento della Java Virtual Machine (JVM):
- "The Java Virtual Machine Specification" di Tim Lindholm e Frank Yellin disponibile on line su http://java.sun.com

Funzionamento di Java

- A differenza degli altri linguaggi di programmazione per Java lo scopo fondamentale e' funzionare su ogni tipo di hadware che possegga un’implementazione della Java Virtual Machine (JVM).
- Il .class che otteniamo dalla compilazione non e' codificato per il linguaggio macchina
- Ad eseguire il .class non sara', quindi, il processore ma un programma che interpreta i bytecode e trasmette i comandi corrispondenti al processore.

The Java Virtual Machine

"Java Architecture"

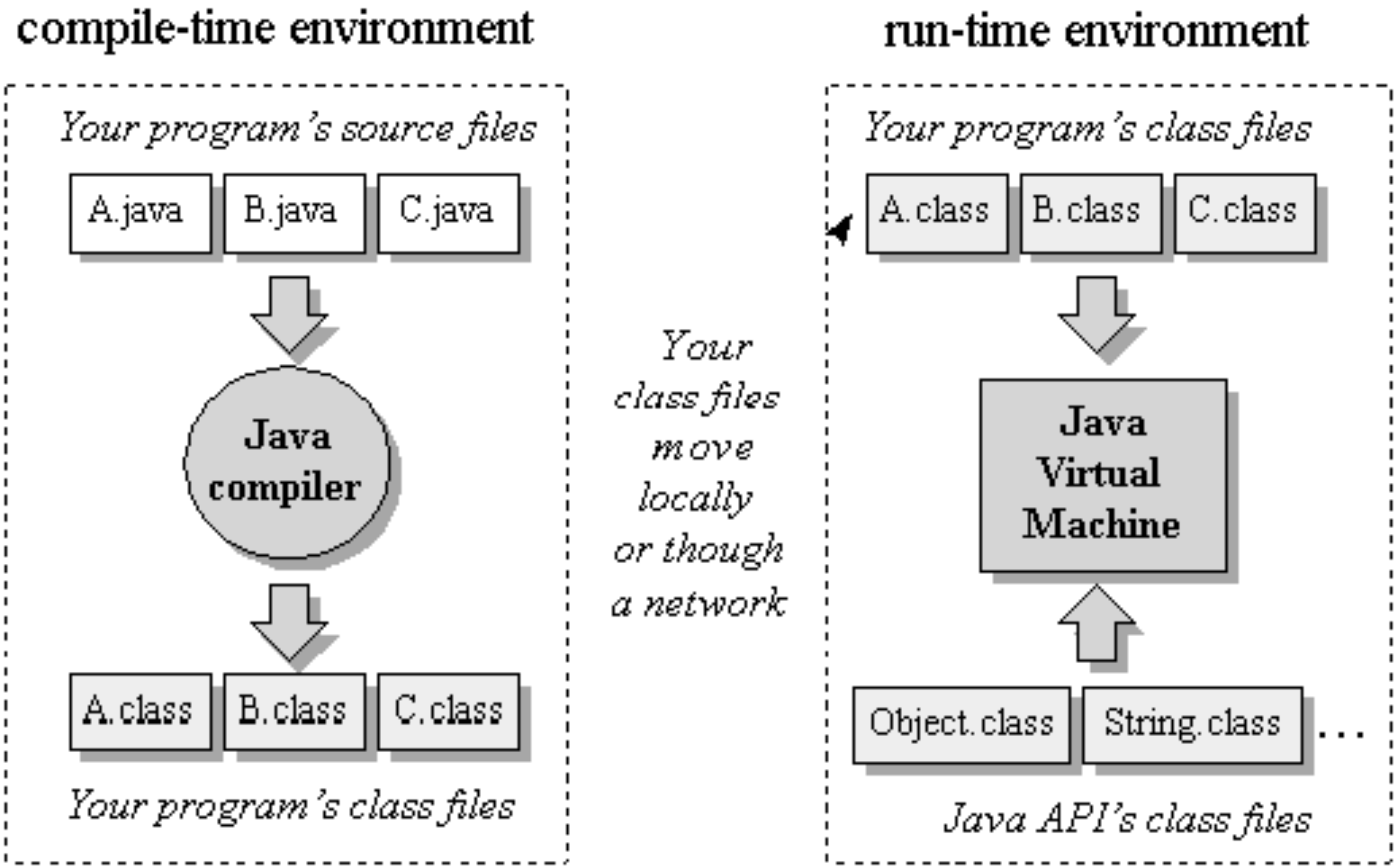
- Java Programming Language
- Java Virtual Machine (JVM)
- Java API

Reference

The content of this lecture is based on *Inside the Java 2 Virtual Machine* by Bill Venners

- Chapter 1 Introduction to Java's Architecture
  - » http://www.artima.com/insidejvm/ed2/introarchP.html
- Chapter 5 The Java Virtual Machine
  - » http://www.artima.com/insidejvm/ed2/jvmP.html
- Interactive Illustrations
  - » http://www.artima.com/insidejvm/applets/index.html

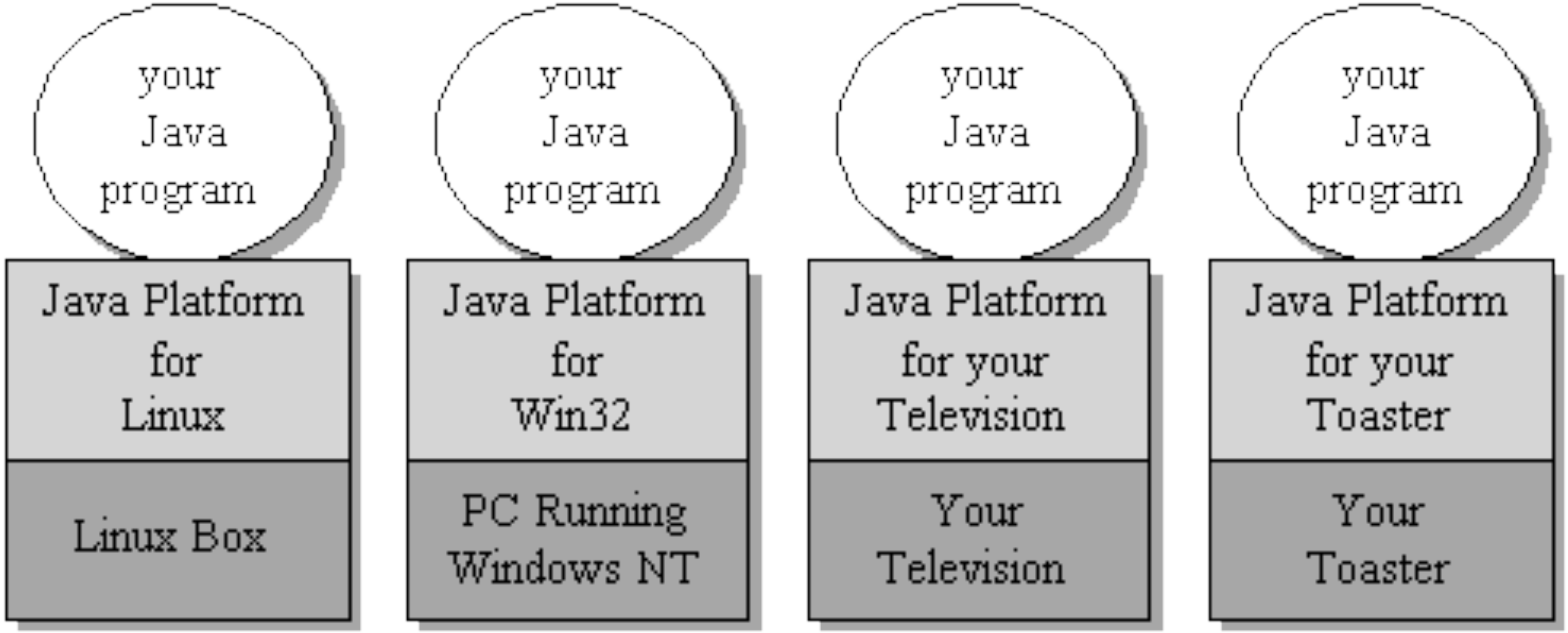
The Java Programming Environment



The Java Platform

The byte code generated by the Java front-end is an *intermediate form*

- Compact
- Platform-independent



your Java program

Java Platform for Linux

Linux Box

your Java program

Java Platform for Win32

PC Running Windows NT

your Java program

Java Platform for your Television

Your Television

your Java program

Java Platform for your Toaster

Your Toaster

The Class File

Java class file contains

- Byte code for data and methods (intermediate form, platform independent)
- *Symbolic* references from one class file to another
  - Class names in text strings
  - Decompiling/reverse engineering quite easy
- Field names and descriptors (type info)
- Method names and descriptors (num args, arg types)
- Symbolic refs to other class methods/fields, own methods/fields

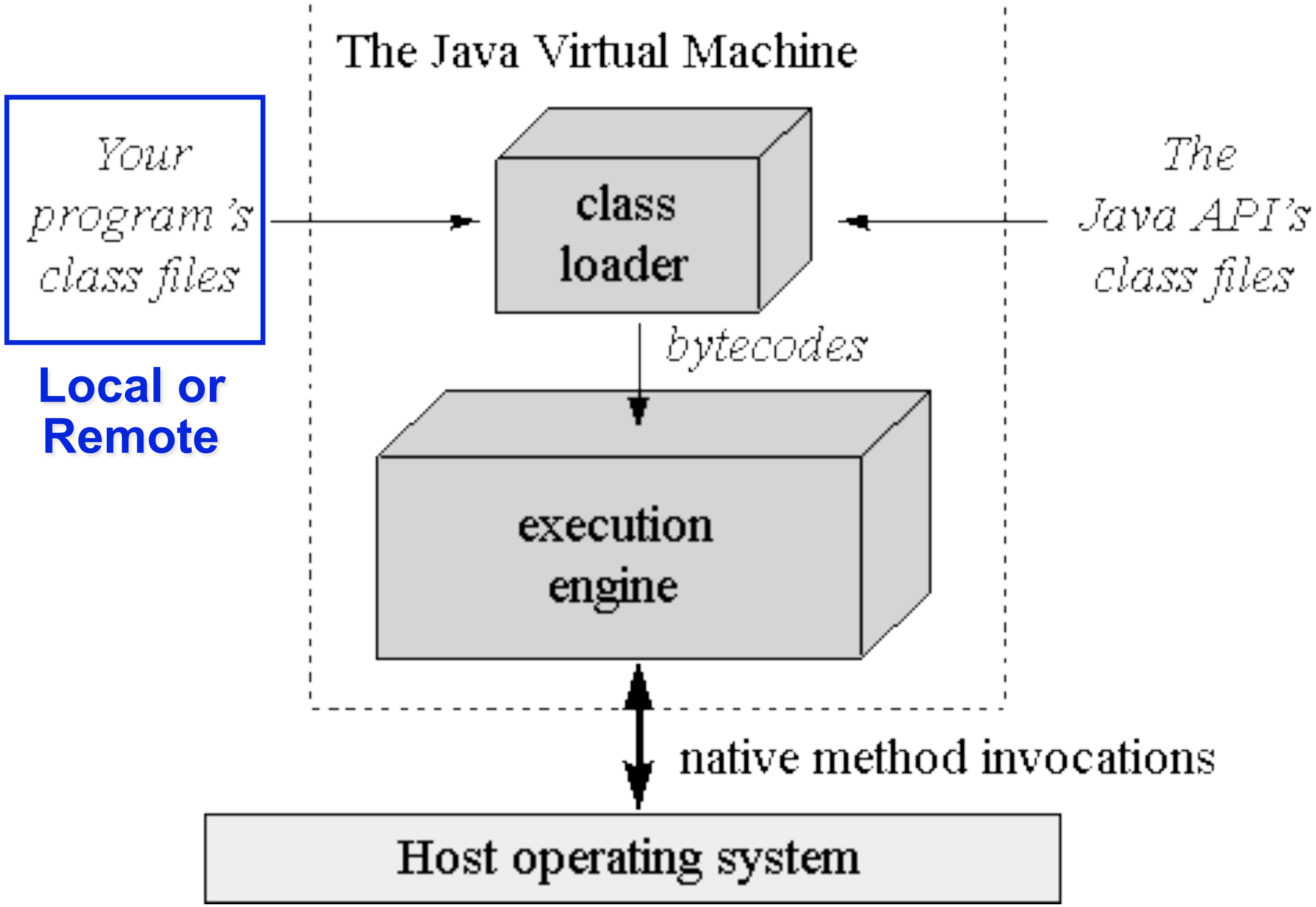
Bytecode Basics

- Bytecodes are the machine language of the Java virtual machine.
- A method's bytecode stream is a sequence of instructions for the Java virtual machine. Each instruction consists of a one-byte *opcode* followed by zero or more *operands*. The opcode indicates the action to take.
- Each type of opcode has a mnemonic. In the typical assembly language style, streams of Java bytecodes can be represented by their mnemonics followed by any operand values.

Bytecode Basics ctd.

```
// Bytecode stream:
03 3b 84 00 01 1a 05 68 3b a7 ff f9
// Disassembly:
iconst_0        // 03
istore_0        // 3b
iinc 0, 1       // 84 00 01
iload_0         // 1a
iconst_2        // 05          ...
```

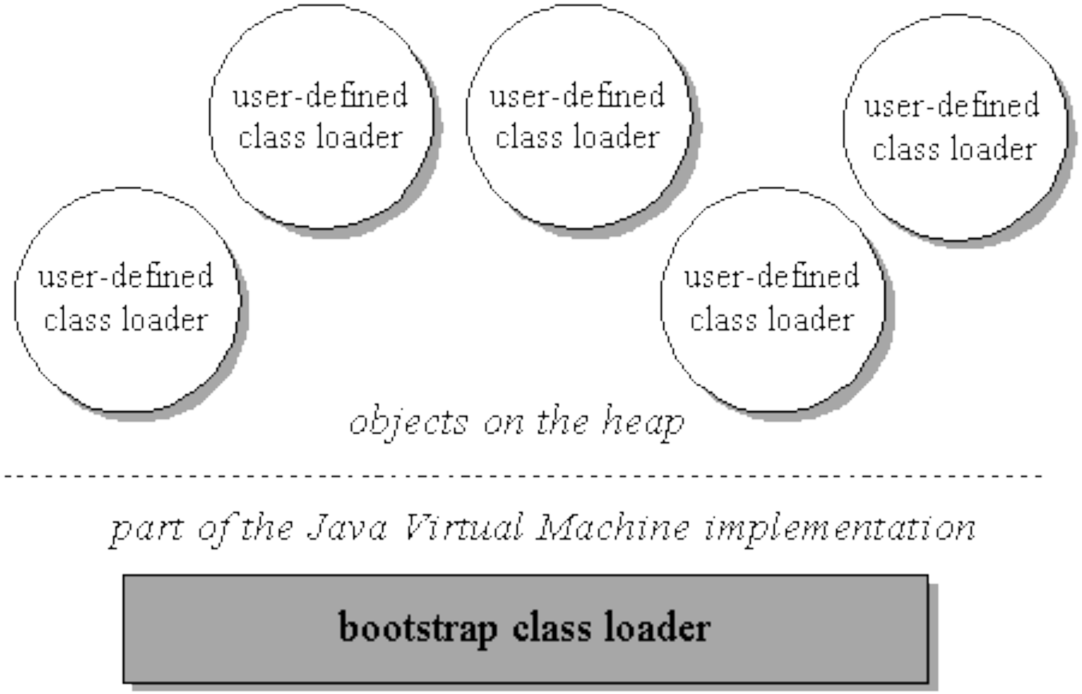
The Role of the Virtual Machine



Local or Remote

Class Loaders

- Bootstrap (default) loader (in the JVM)
- User-defined (custom) loaders


user-defined class loader
user-defined class loader
user-defined class loader
user-defined class loader
user-defined class loader
objects on the heap
part of the Java Virtual Machine implementation
bootstrap class loader

Dynamic Class Loading

- You don't have to know at compile-time all the classes that may ultimately take part in a running Java application.

  User-defined class loaders enable you to dynamically extend a Java app at run-time

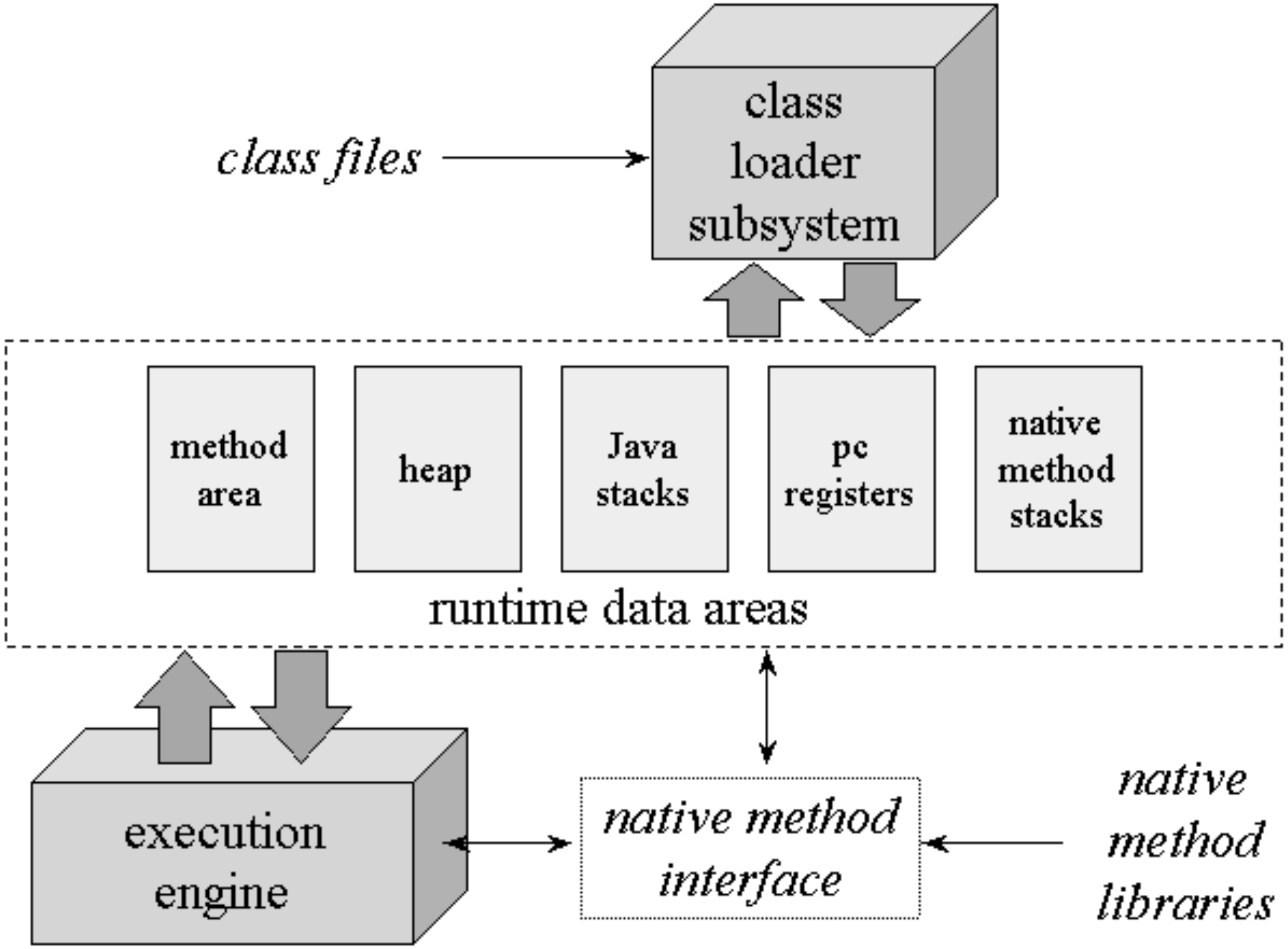
- As it runs, your app can determine what extra classes it needs and load them
- Custom loaders can download classes across a network (applets), get them out of some kind of database, or even calculate them on the fly.

The Execution Engine

Back-end transformation and execution

- *Simple JVM*
  - byte code interpretation
- *Just-in-time compiler*
  - Method byte codes are compiled into machine code the first time they are invoked
  - The machine code is cached for subsequent invocation
  - It requires more memory
- *Adaptive optimization*
  - The interpreter monitors the activity of the program, compiling the heavily used part of the program into machine code
  - It is much faster than simple interpretation, a little more memory
  - The memory requirement is only slightly larger due to the 20%/80% rule of program execution *(In general, 20% of the code is responsible for 80% of the execution)*

The Java Virtual Machine

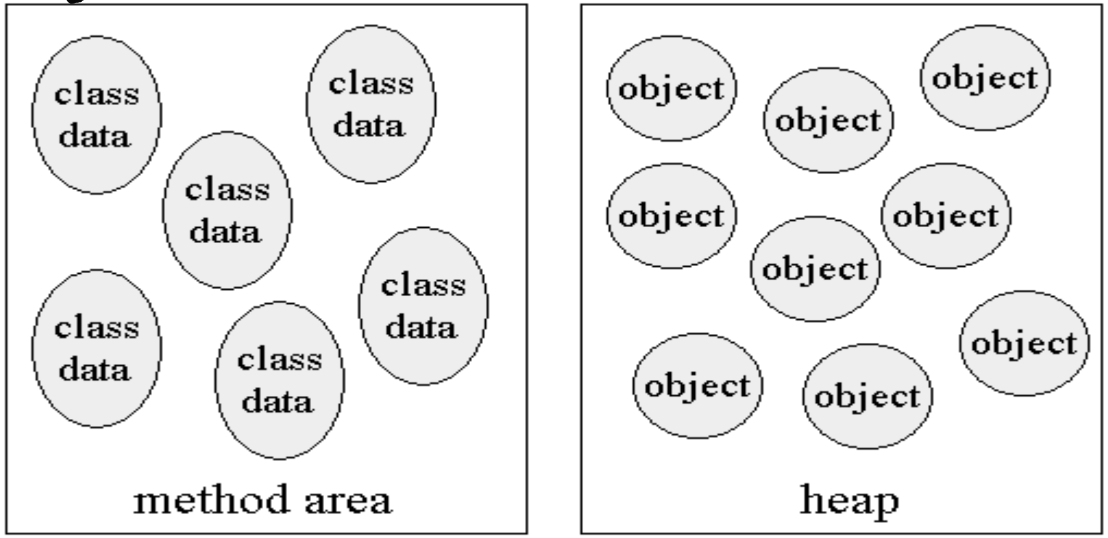


Shared Data Areas

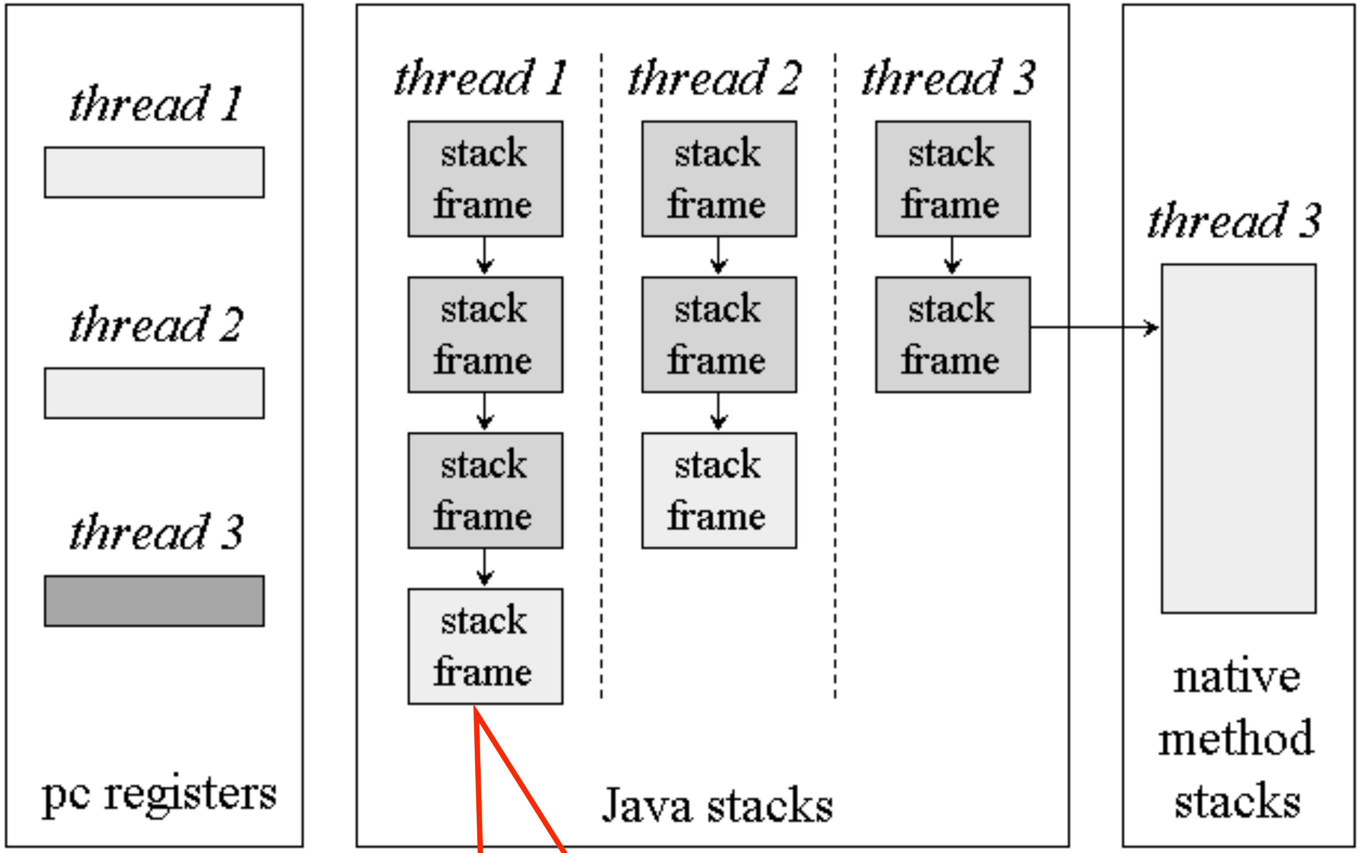
Each JVM has one of each:

✓Method area: byte code and class (static) data storage

✓Heap: object storage



Thread Data Areas



Frame in Execution

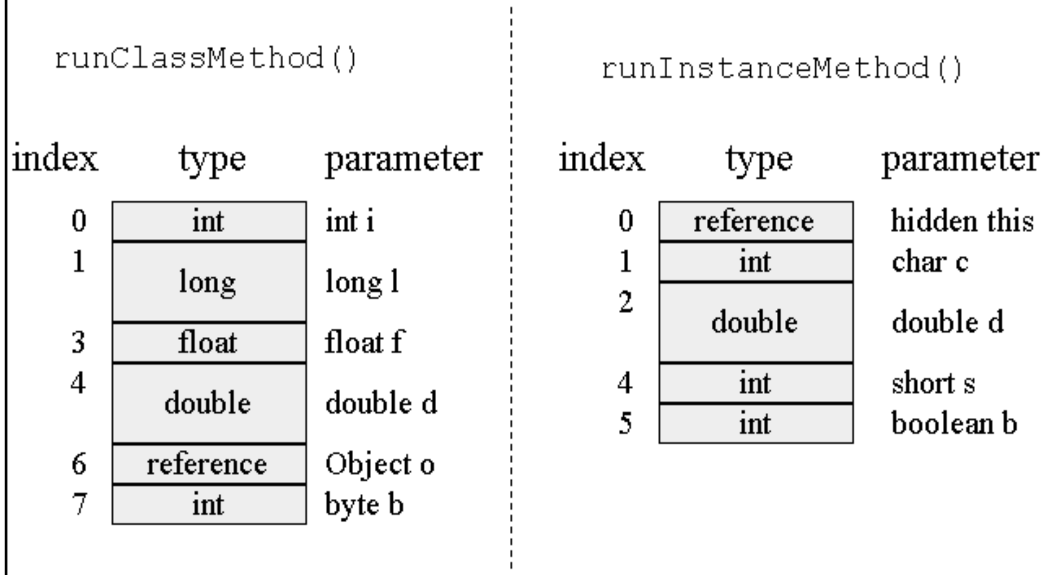
Stack Frames

Stack frames have three parts

- Local variables
- Operand stack
- Frame data

Stack Frame
Local Variables

```
class Example3a {
    public static int
runClassMethod(int i, long
l, float f, double d, Object
o, byte b) {
        return 0;
    }
    public int
runInstanceMethod(char c,
double d, short s, boolean
b) {
        return 0;
    }
}
```



runClassMethod()

index	type	parameter
0	int	int i
1	long	long l
3	float	float f
4	double	double d
6	reference	Object o
7	int	byte b

runInstanceMethod()

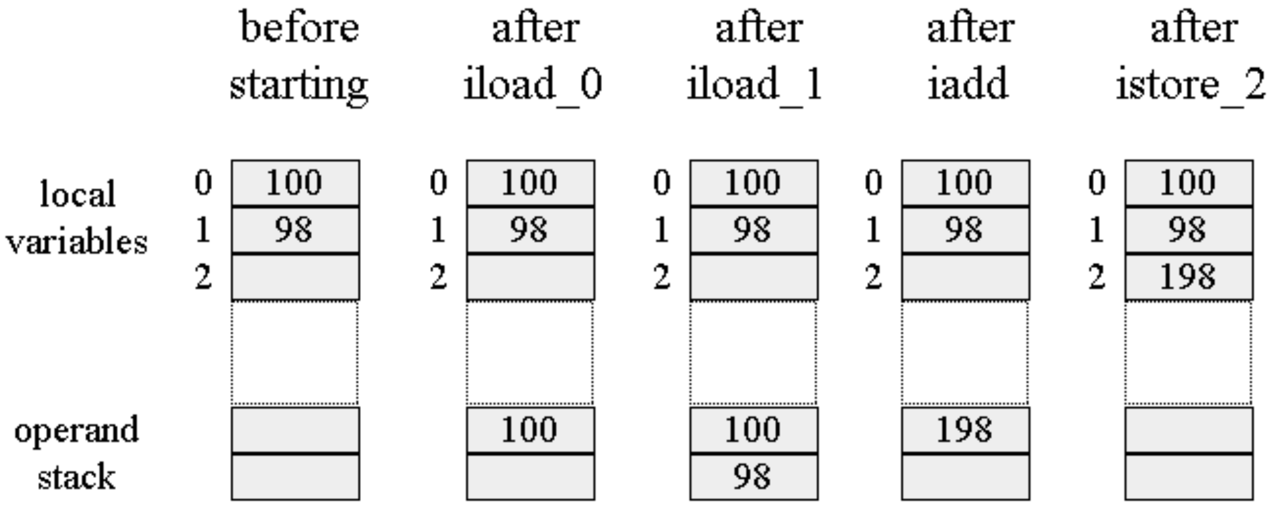
index	type	parameter
0	reference	hidden this
1	int	char c
2	double	double d
4	int	short s
5	int	boolean b

Stack Frame
Operand Stack

Adding 2 numbers

```
iload_0
iload_1
Iadd
istore_2
```

Compiler can tell how many slots the op stack will need for a method



		before starting	after iload_0	after iload_1	after iadd	after istore_2
local variables	0	100	100	100	100	100
	1	98	98	98	98	98
	2					198
operand stack			100	100	198	
				98		

Stack Frame

Frame Data

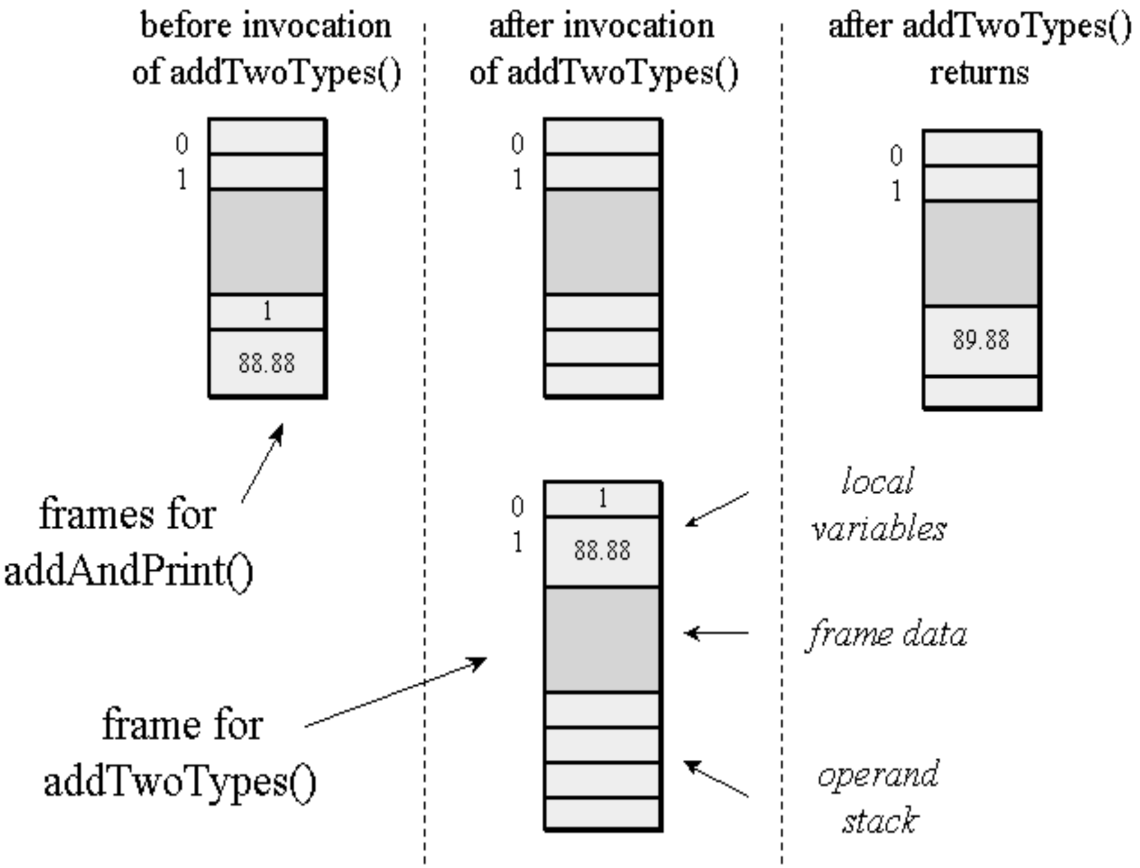
The stack frame also supports

- Constant pool resolution
- Normal method return
- Exception dispatch

Stack Frame

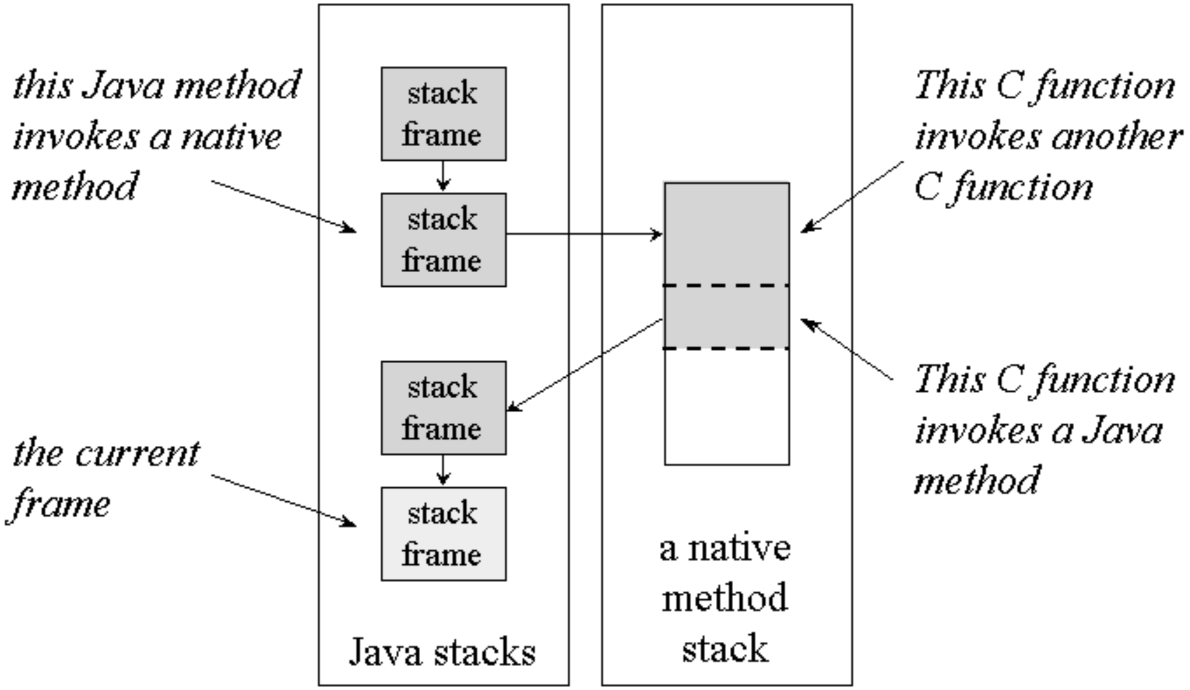
Frame Allocation in a Heap

```
class Example3c {

    public static void
addAndPrint() {

        double result =
addTwoTypes(1, 88.88);


System.out.println(result)
;

    }



    public static double
addTwoTypes(int i, double
d) {

        return i + d;

    }

}
```



Stack Frame
Native Method

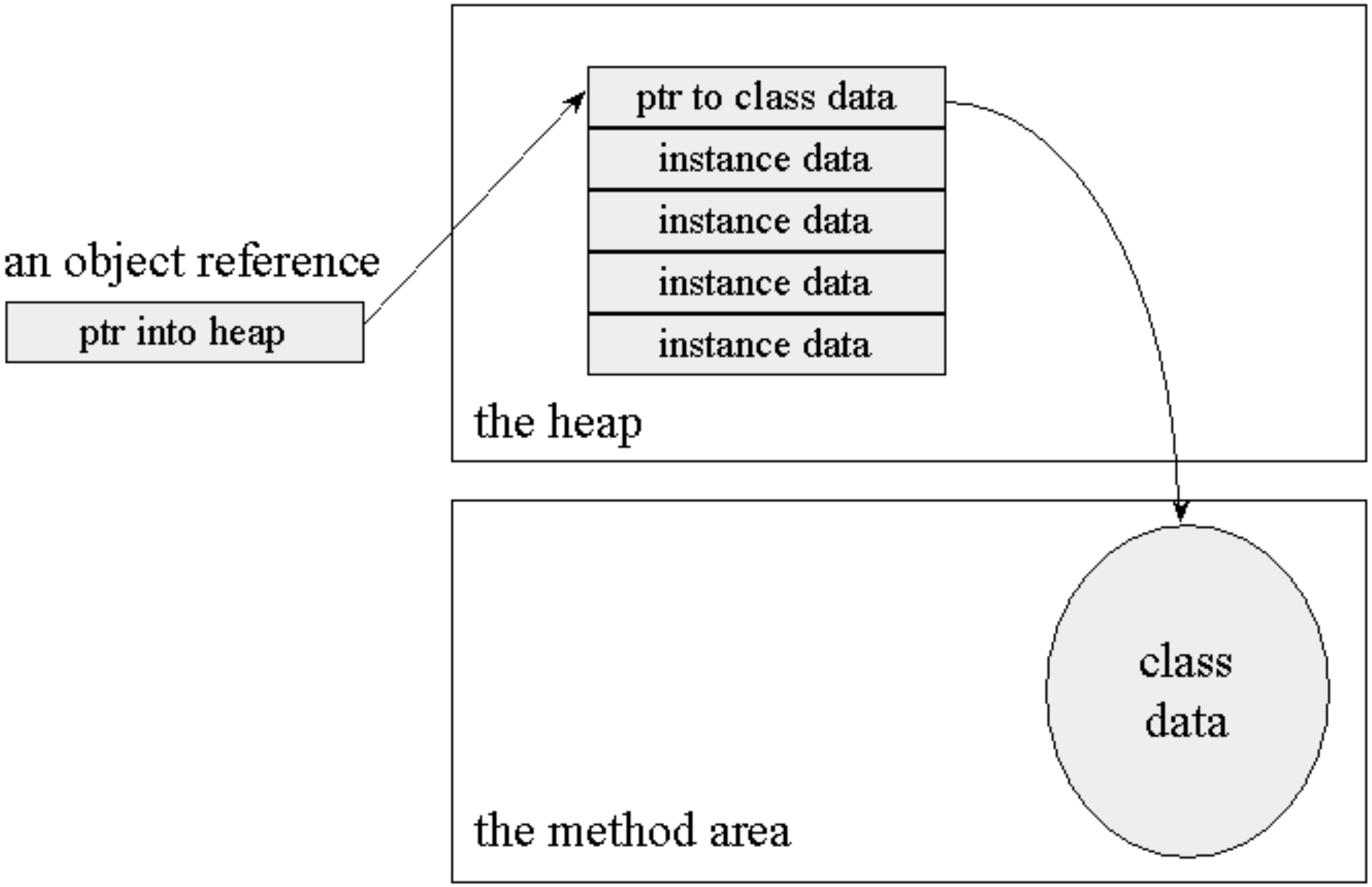
A simulated stack of the target language (e.g. C) is created for JNI


this Java method invokes a native method
stack frame
stack frame
stack frame
stack frame
the current frame
Java stacks
This C function invokes another C function
This C function invokes a Java method
a native method stack

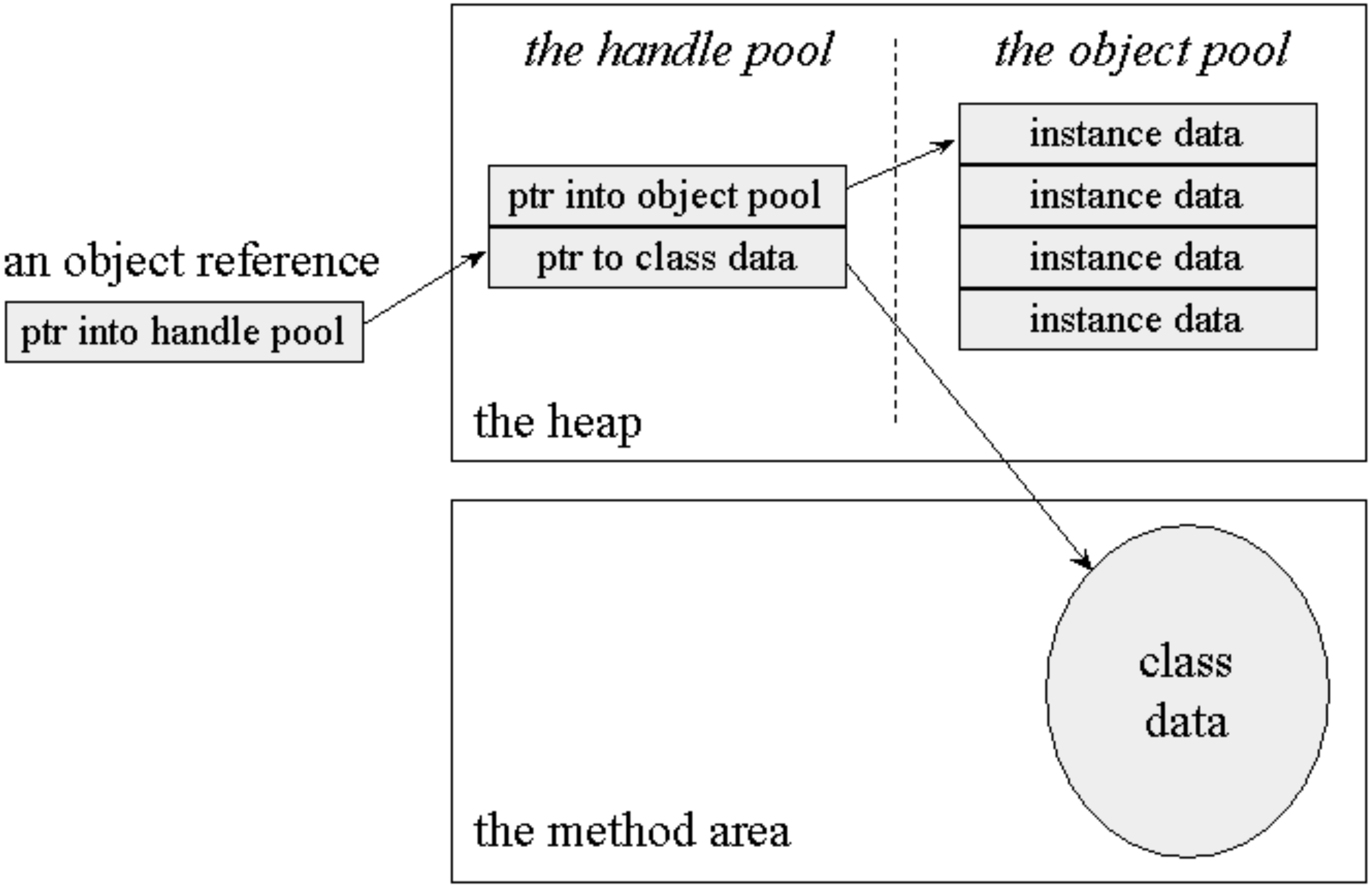
The Heap

- Class instances (objects) and arrays are stored in a single, shared heap
- Each Java application has its own heap
  - Isolation
  - But a JVM crash will break this isolation
- JVM heaps always implement garbage collection mechanisms

Heap
Monolithic Object Representation

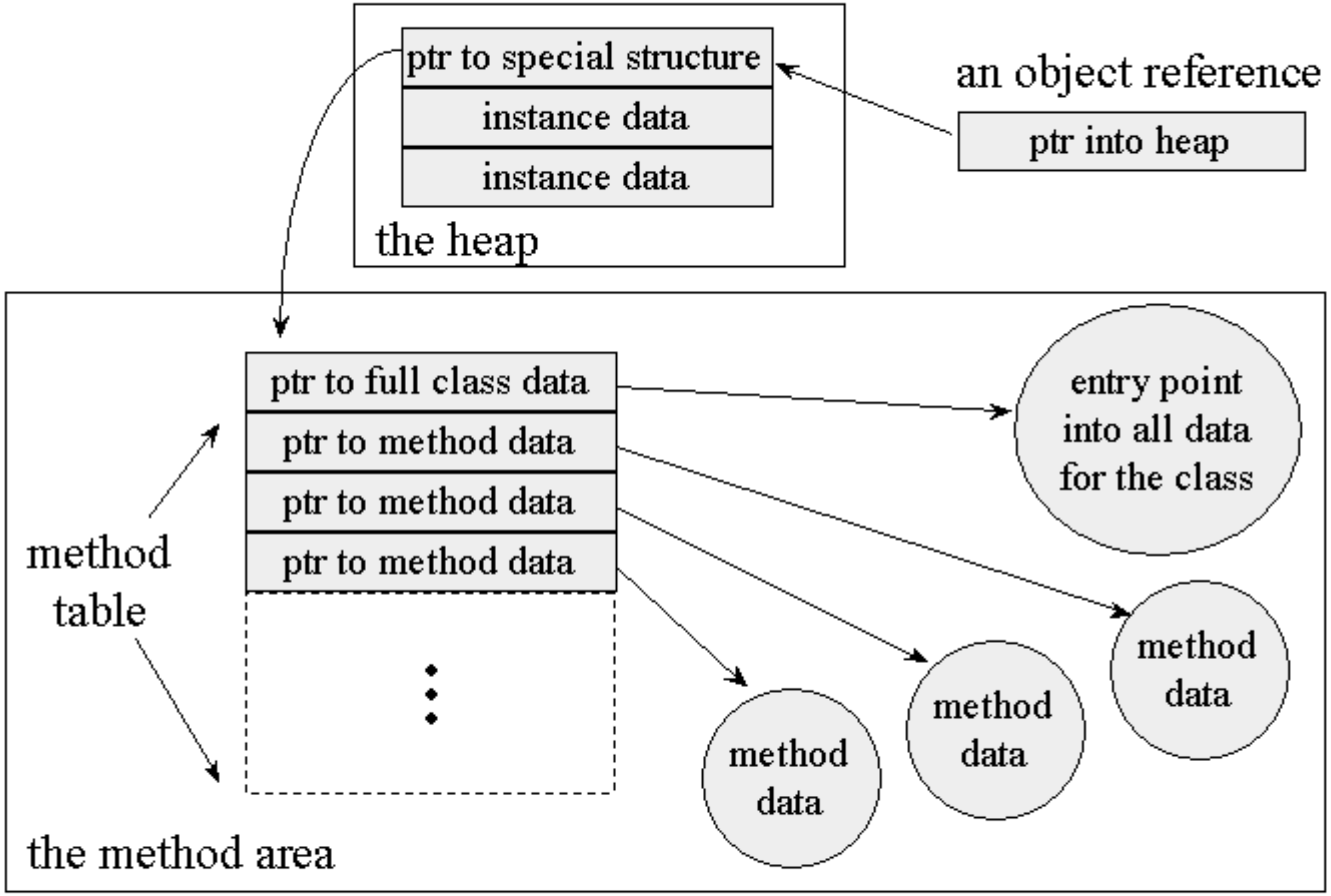


The Heap
Split Object Representation



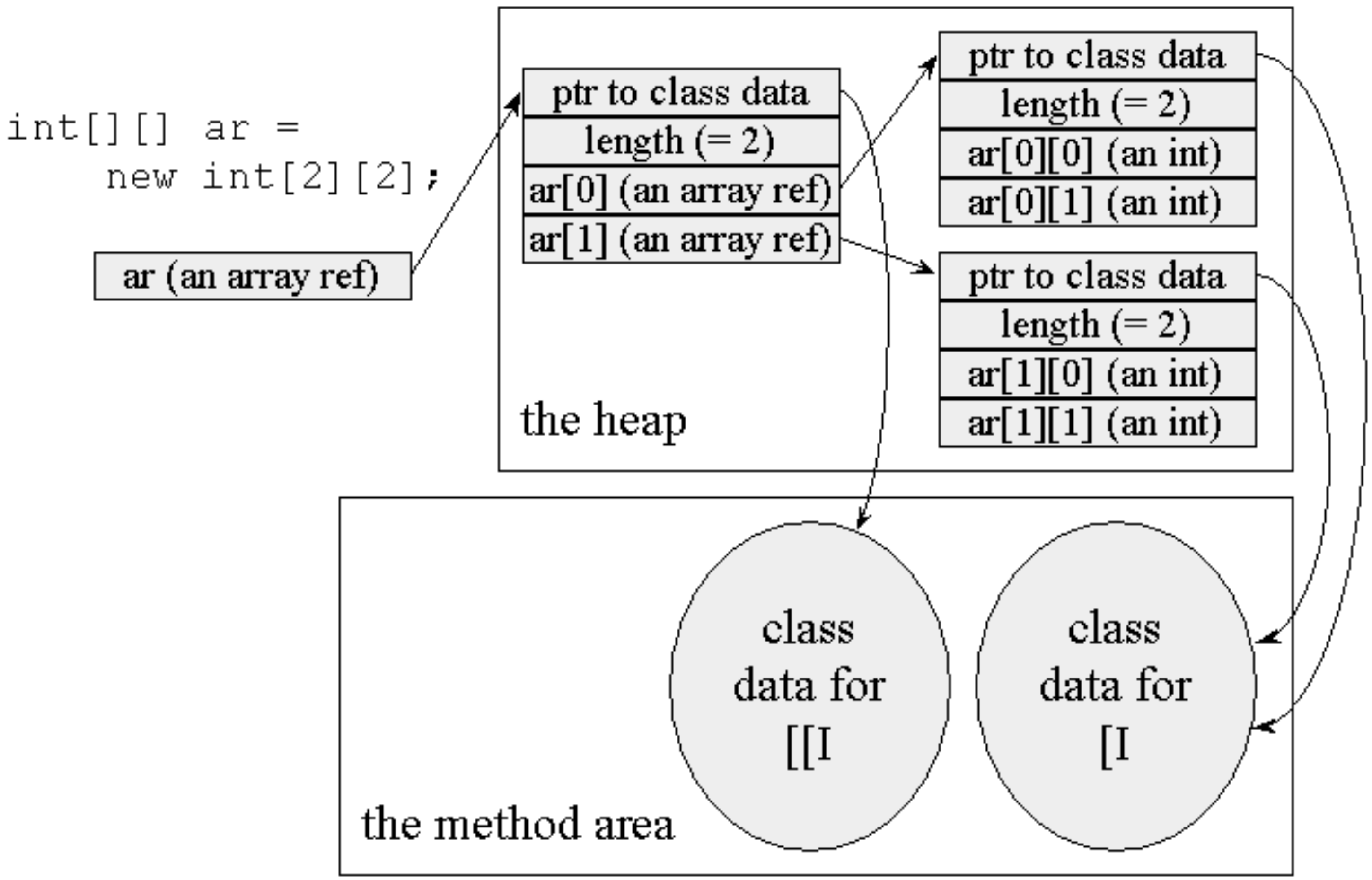
The Heap

Memory/Speed Tradeoff



The Heap

Arrays as Objects



```
int[][] ar =
    new int[2][2];
```

Examples

HeapOfFish

- http://www.artima.com/insidejvm/applets/HeapOfFish.html
- Object allocation illustration

Eternal Math Example

- http://www.artima.com/insidejvm/applets/EternalMath.html
- JVM execution, operand stack, illustration