Compiling Object-Oriented Languages

Andrew P. Black

How are OO Languages Different?

- methods instead of procedures
  - method request instead of procedure call
  - “full upward funargs"
  - inheritance & encapsulation
    ⇒ frequent method requests

How are OO Languages Different?

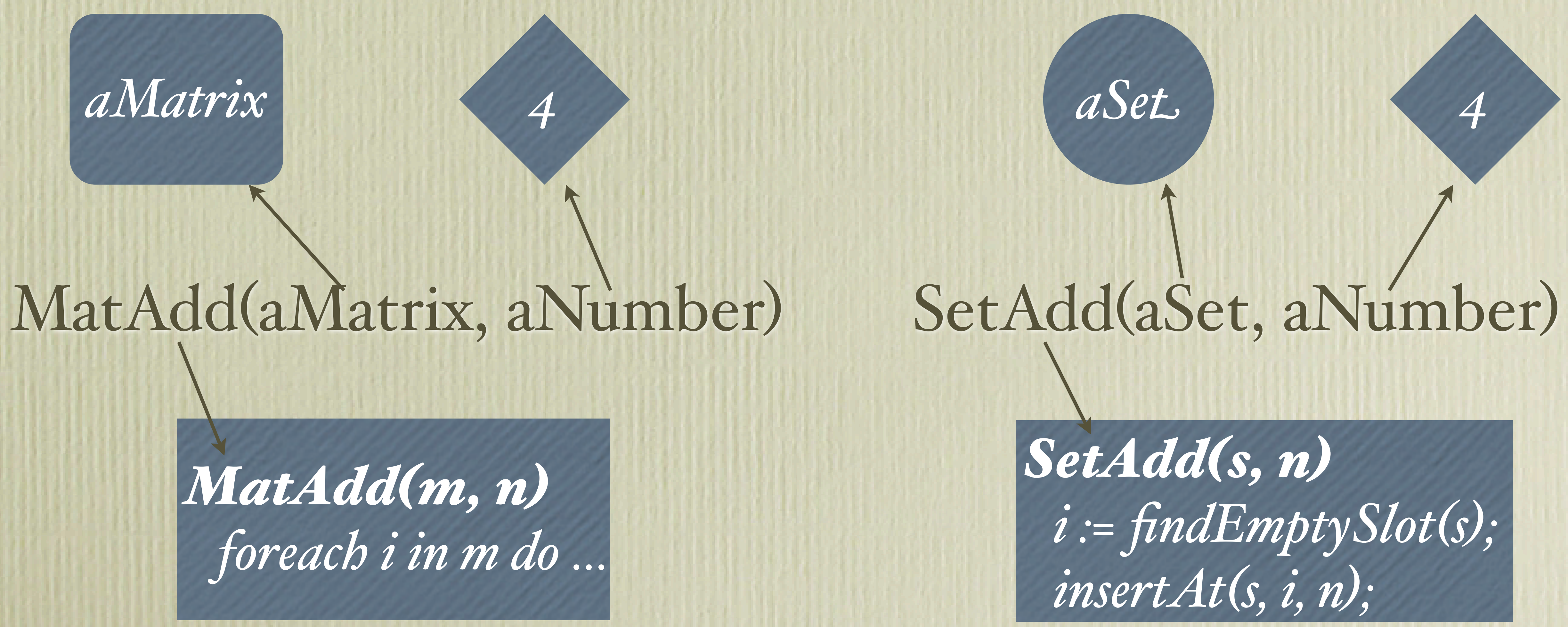
- subtyping
  - types dictate interface, not implementation
    - not in all languages
  - code to be executed not known at time of request

Method Request

- Method request, aka message send, is not the same as procedure call

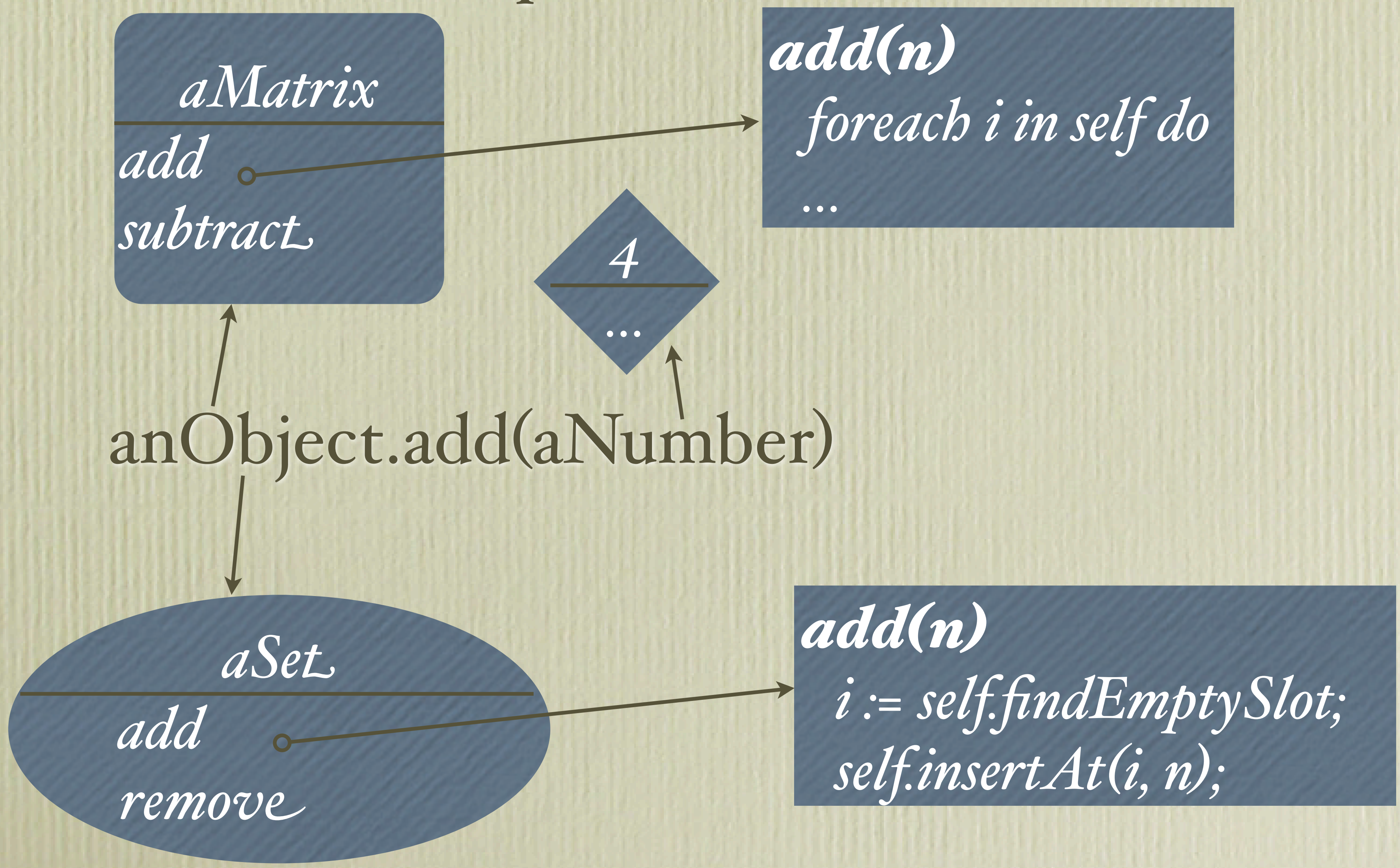
Procedure Call

- Code to be executed is identified by name at call site
  - Compiler's job:



Method Request

- Code to be executed depends on the **receiver** of the request

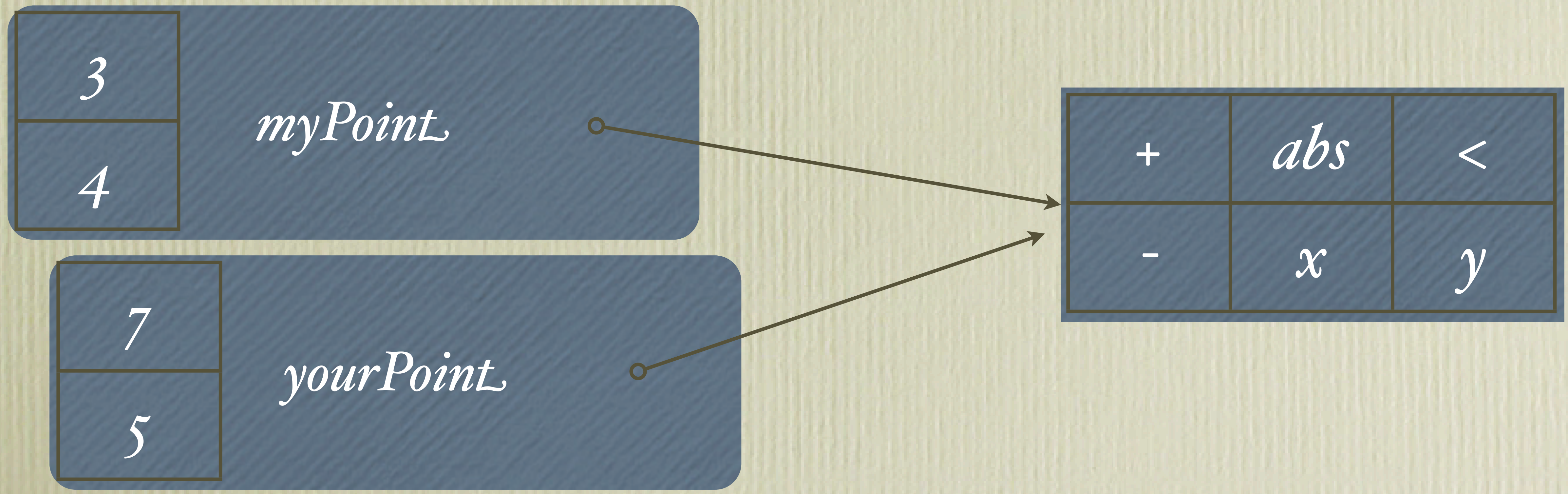


Implementing Objects

- Each object contains, *conceptually*:
  - a set of named methods
  - a set of named instance variables

Implementing Objects

- Each object contains, *in practice*:
  - a reference to a shared set of named methods
  - a set of named instance variables



Points in Smalltalk

```
Object subclass: #Point
    instanceVariableNames: 'x y'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Graphics-Primitives'
```

Point class » x:y:

```
x: xInteger y: yInteger
    "Answer an instance of me with coordinates xInteger and yInteger."

    ^self basicNew setX: xInteger setY: yInteger
```

```
21 <70> self
22 <D1> send: basicNew
23 <10> pushTemp: 0
24 <11> pushTemp: 1
25 <F0> send: setX:setY:
26 <7C> returnTop
```

Point » setX:setY:

```
setX: xValue setY: yValue
    x := xValue.
    y := yValue
```

```
13 <10> pushTemp: 0
14 <60> popIntoRcvr: 0
15 <11> pushTemp: 1
16 <61> popIntoRcvr: 1
17 <78> returnSelf
```

Points in Smalltalk

```
Object subclass: #Point
    instanceVariableNames: 'x y'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Graphics-Primitives'
```

```
+ arg
    "Answer a Point that is the sum of the receiver and arg."

    arg isPoint ifTrue: [^ (x + arg x) @ (y + arg y)].
    ^ arg adaptToPoint: self andSend: #+
```

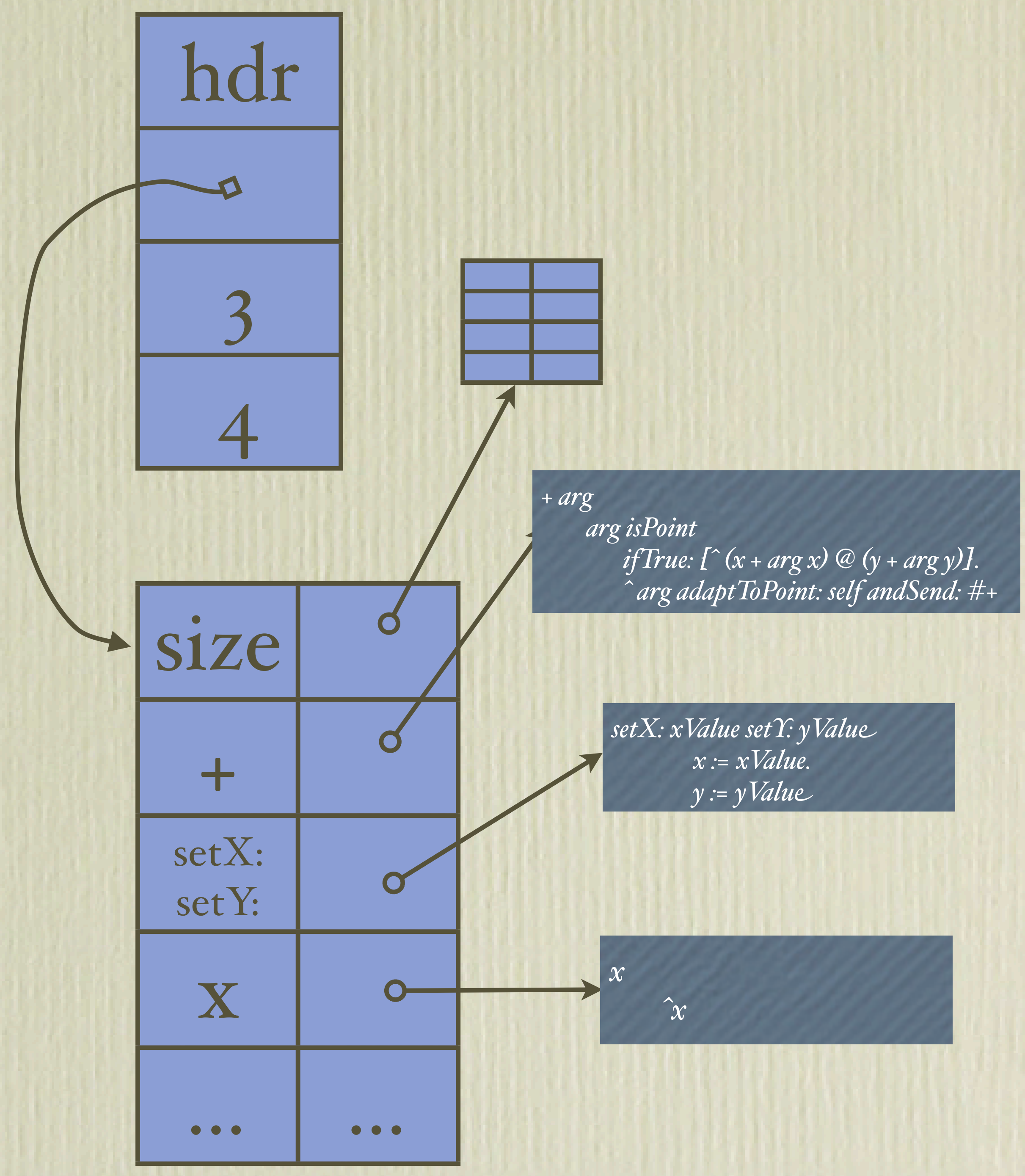
```
x
    "Answer the x coordinate."

    ^x
```

```
25 <10> pushTemp: 0
26 <D0> send: isPoint
27 <AC 0A> jumpFalse: 39
29 <00> pushRcvr: 0
30 <10> pushTemp: 0
31 <CE> send: x
32 <B0> send: +
33 <01> pushRcvr: 1
34 <10> pushTemp: 0
35 <CF> send: y
36 <B0> send: +
37 <BB> send: @
38 <7C> returnTop
39 <10> pushTemp: 0
40 <70> self
41 <22> pushConstant: #+
42 <F1> send: adaptToPoint:andSend:
43 <7C> returnTop
```

What does “send x” mean?

1. Find the representation of the receiver
2. Find its list of methods
3. Look for a method named “x”
4. If there is none, repeat above in the methods of the receiver’s superclass …



Points in Java

```java
interface Point{
    Point plus(Point p);
    boolean greaterThan(Point p);
    double x();
    double y();
}
```

```java
public class CartesianPoint implements Point{
    private double x;
    private double y;

    // constructor
    CartesianPoint(double xCoord, double yCoord) {
        x = xCoord;
        y = yCoord;
    }

    public double x() { return x ;}
    public double y() { return y ;}
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y()); }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y()); }
}
```

```java
public class PolarPoint implements Point{
    private double r;
    private double theta;

    // constructor
    PolarPoint(double xCoord, double yCoord) {
        r = java.lang.Math.sqrt((xCoord*xCoord) + (yCoord*yCoord));
        theta = java.lang.Math.atan2(yCoord, xCoord);
    }

    public double x() { return r * java.lang.Math.cos(theta) ;}
    public double y() { return r * java.lang.Math.sin(theta) ;}
    public Point plus(Point p) {
        return new PolarPoint(this.x()+p.x(), this.y()+p.y()); }
    public boolean greaterThan(Point p) {
        return (this.x()>p.x()) & (this.y()>p.y()); }
}
```

Points in Java

```
public class CartesianPoint implements Point{
    private double x;
    private double y;

    // constructor
    CartesianPoint(double xCoord, double yCoord) {
        x = xCoord;
        y = yCoord;
    }

    public double x() { return x ;}
    public double y() { return y ;}
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y()); }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y()); }
}
```

$ javap -c CartesianPoint

```
Compiled from "CartesianPoint.java"
public class CartesianPoint extends java.lang.Object implements Point{
CartesianPoint(double, double);
  Code:
   0:  aload_0
   1:  invokespecial #1; //Method java/lang/Object."<init>":()V
   4:  aload_0
   5:  dload_1
   6:  putfield   #2; //Field x:D
   9:  aload_0
   10: dload_3
   11: putfield   #3; //Field y:D
   14: return
```

Points in Java

```
public class CartesianPoint implements Point{
    private double x;
    private double y;

    // constructor
    CartesianPoint(double xCoord, double yCoord) {
        x = xCoord;
        y = yCoord;
    }

    public double x() { return x ;}
    public double y() { return y ;}
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y()); }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y()); }
}
```

```
$ javap -c CartesianPoint
```

```
public double x();
  Code:
   0:   aload_0
   1:   getfield   #2; //Field x:D
   4:   dreturn

public double y();
  Code:
   0:   aload_0
   1:   getfield   #3; //Field y:D
   4:   dreturn
```

Points in Java

```
public class CartesianPoint implements Point{
    private double x;
    private double y;

    // constructor
    CartesianPoint(double xCoord, double yCoord) {
        x = xCoord;
        y = yCoord;
    }

    public double x() { return x ;}
    public double y() { return y ;}
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y()); }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y()); }
}
```

$ javap -c CartesianPoint

```
public Point plus(Point);
  Code:
   0:   new #4; //class CartesianPoint
   3:   dup
   4:   aload_0
   5:   getfield   #2; //Field x:D
   8:   aload_1
   9:   invokeinterface   #5,   1; //InterfaceMethod Point.x:()D
   14:  dadd
   15:  aload_0
   16:  getfield   #3; //Field y:D
   19:  aload_1
   20:  invokeinterface   #6,   1; //InterfaceMethod Point.y:()D
   25:  dadd
   26:  invokespecial #7; //Method "<init>":(DD)V
   29:  areturn
```

```java
        x = xCoord;
        y = yCoord;
    }

    public double x() { return x ;}
    public double y() { return y ;}
    public Point plus(Point p) {
        return new CartesianPoint(x+p.x(), y+p.y()); }
    public boolean greaterThan(Point p) {
        return (x>p.x()) & (y>p.y()); }
}
```

$ javap -c CartesianPoint

```
public boolean greaterThan(Point);
  Code:
   0:   aload_0
   1:   getfield   #2; //Field x:D
   4:   aload_1
   5:   invokeinterface   #5,  1; //InterfaceMethod Point.x:()D
   10:  dcmpl
   11:  ifle   18
   14:  iconst_1
   15:  goto   19
   18:  iconst_0
   19:  aload_0
   20:  getfield   #3; //Field y:D
   23:  aload_1
   24:  invokeinterface   #6,  1; //InterfaceMethod Point.y:()D
   29:  dcmpl
   30:  ifle   37
   33:  iconst_1
   34:  goto   38
   37:  iconst_0
   38:  iand
   39:  ireturn

}
```

```java
    private double theta;

    // constructor
    PolarPoint(double xCoord, double yCoord) {
        r = java.lang.Math.sqrt((xCoord*xCoord) + (yCoord*yCoord));
        theta = java.lang.Math.atan2(yCoord, xCoord);
    }

    public double x() { return r * java.lang.Math.cos(theta) ;}
    public double y() { return r * java.lang.Math.sin(theta) ;}
    public Point plus(Point p) {
        return new PolarPoint(this.x()+p.x(), this.y()+p.y()); }
    public boolean greaterThan(Point p) {
        return (this.x()>p.x()) & (this.y()>p.y()); }
}
```

```
public Point plus(Point);
  Code:
   0:   new #8; //class PolarPoint
   3:   dup
   4:   aload_0
   5:   invokevirtual #9; //Method x:()D
   8:   aload_1
   9:   invokeinterface  #10,  1; //InterfaceMethod Point.x:()D
   14:  dadd
   15:  aload_0
   16:  invokevirtual #11; //Method y:()D
   19:  aload_1
   20:  invokeinterface  #12,  1; //InterfaceMethod Point.y:()D
   25:  dadd
   26:  invokespecial #13; //Method "<init>":(DD)V
   29:  areturn
```

Why is method request slow?

1. String compare
2. Linear Search
3. Chaining through super dictionaries

Why does it matter?

It doesn’t matter

- So long as there is a virtual machine interpreting the byte-code instructions, the overhead of method request is not much of a problem

How to speed-up OO?

- Compile them!
- Translate each byte code into the equivalent series of machine instructions
    - the very same instructions that the interpreter would have *executed*
- *method Request* is now a subroutine

  ... and it's time-consuming

  Recall why:

String Compare

- String comparison is slow (linear in the length of the shorter string)
  - Avoid by using the Flyweight Pattern
    - see Smalltalk class Symbol

Linear Search

- Linear Search is slow
  - Linear in the number of methods
- Avoid by hashing
  - hash can be generated at compile time
    - hash function should be part of the language!
  - Hashing is constant time, provided ____________
  - Space is not free

Why is this slow?

- Chaining through super dictionaries
  - Avoid by copying down super methods at compile time
  - *e.g.,* Point inherits Object»printString, so copy the pair 〈 #printString, code ptr 〉 into Point's method dictionary.
- Two problems:
  1. super-sends
  2. space consumption

Simple Cache

- Small cache indexed by pair
  ⟨ receiver class, method name ⟩
- Speeds–up overall system by 20% to 30% [Krasner 1983], 37% [Hölzle 1981]
- But: there are lots of classes in the system!

Per request-site Cache

- Idea: use a separate cache for each method request site.
  [Deutsch POPL 1983]: Efficient Implementation of Smalltalk
  - Locality says that most of the receivers at a given site will be of the same class
  - *e.g.*, list.collect { each → each.display }
    - if list is homogeneous, all of the convert requests will be to the same method
  - Also: method name is now a constant

How to find the Cache?

- if you use one cache for each method request in the program, there will be a *lot* of caches
  - make caches small, *e.g.*, one entry!
- How do we find the right cache?
  - Simple and effective solution: place the cache “in-line”: in the *code* in place of the original request!

(3@4) display

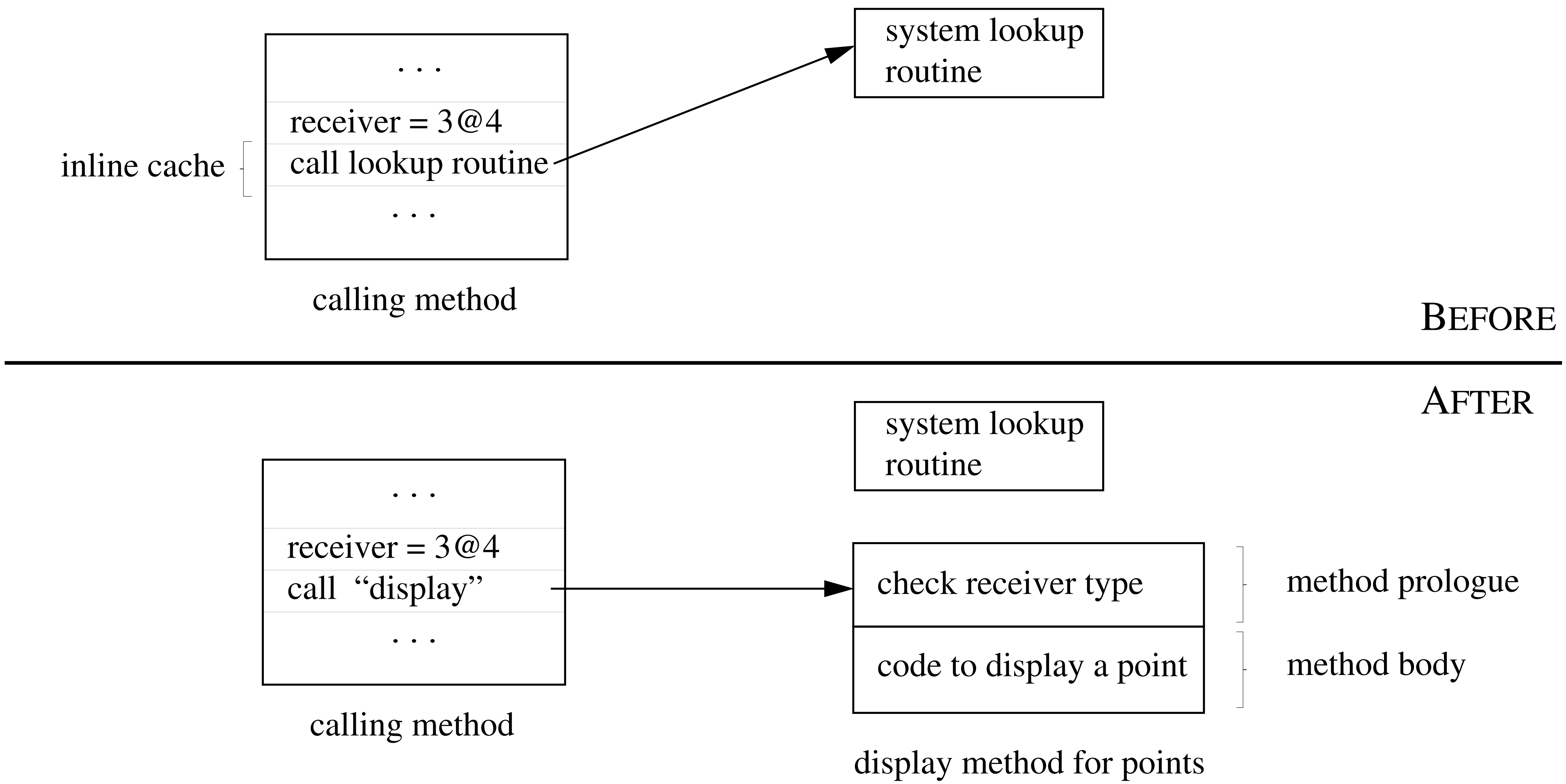


Figure 1. Inline Caching

Figure from Hölzle, U., Chambers, C., and Ungar, D. 1991. Optimizing dynamically-typed object-oriented languages with polymorphic inline caches. In Proceedings ECOOP '91.

Inline Caching

- Exploits locality of call site
- site is originally "unlinked":
  - jumps to the general lookup routine
- After first request, site is over-written with call to the "prologue" of the found method
  - prologue checks that the class of the receiver is that expected by the method
  - if it's not, jump to general lookup routine

Inline Caching is Effective

- 95% effective for Smalltalk
- Overall speedup of 50% on SOAR
- Can be combined with simple ⟨ receiver class, method name ⟩ cache to handle misses.

What about Polymorphic Sends?

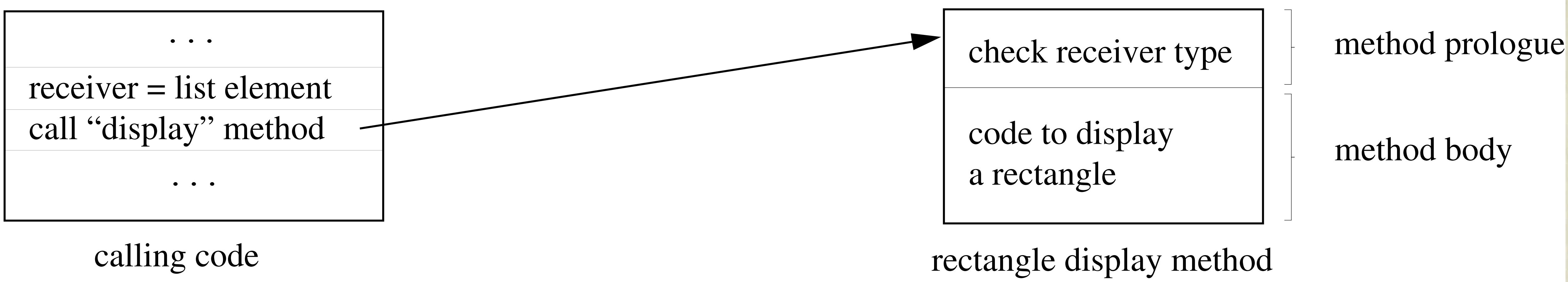
- Example:
```
array := #(1 'a' 2 'b' 3 'c' 4 'd' 5 'e')
array do: [ :each |  each printOn: Transcript]
```
- Worst case for inline-cache:
  - Why?

Polymorphic Sends

- Degree of Polymorphism is usually small
  - less than 10
- If it's not small, then it's large
  - Trimodal distribution: monomorphic, polymorphic, megamorphic.

Polymorphic Inline Caches

- Suppose that we are displaying the elements of a list
  - So far, every element has been a Rectangle



- Now suppose that the next element is a circle

- Inline cache calls prologue of display method for Rectangles.
- Prologue detects the cache miss, and calls system lookup routine
- lookup routine finds the correct method
    - constructs a stub, and *replaces* original inline cache with call to this stub (stub is the PIC)
- PIC stub checks if receiver is a Rectangle or a Circle, and jumps to the start of the appropriate method.
    - No need to jump to the prologue

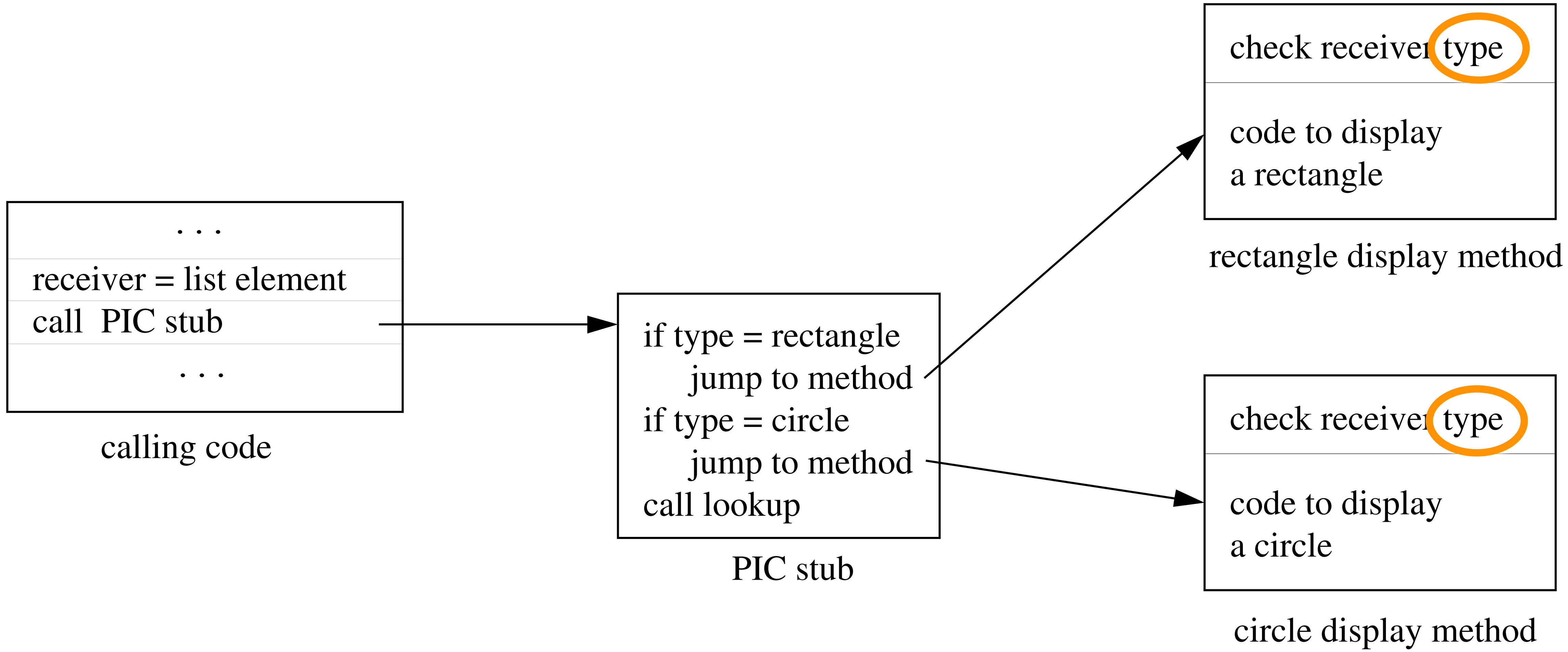


Figure 3. Polymorphic inline cache

- Suppose the next object is a Triangle
  - PIC stub routine misses, but is extended with a third case:
    - PIC now handles Rectangles, Circles and Triangles.
- Eventually, the PIC will handle all cases seen in practice.
- If the size of the PIC grows too large:
  - Mark request site as megamorphic and quit caching.

Variations

- Inline small methods into PIC stub
- Order classes in PIC by frequency
- Replace linear search by hashing, binary search, *etc.*
- Sharing PICS between request sites that have same method name
  - saves space, looses locality

PICs first Implemented for Self

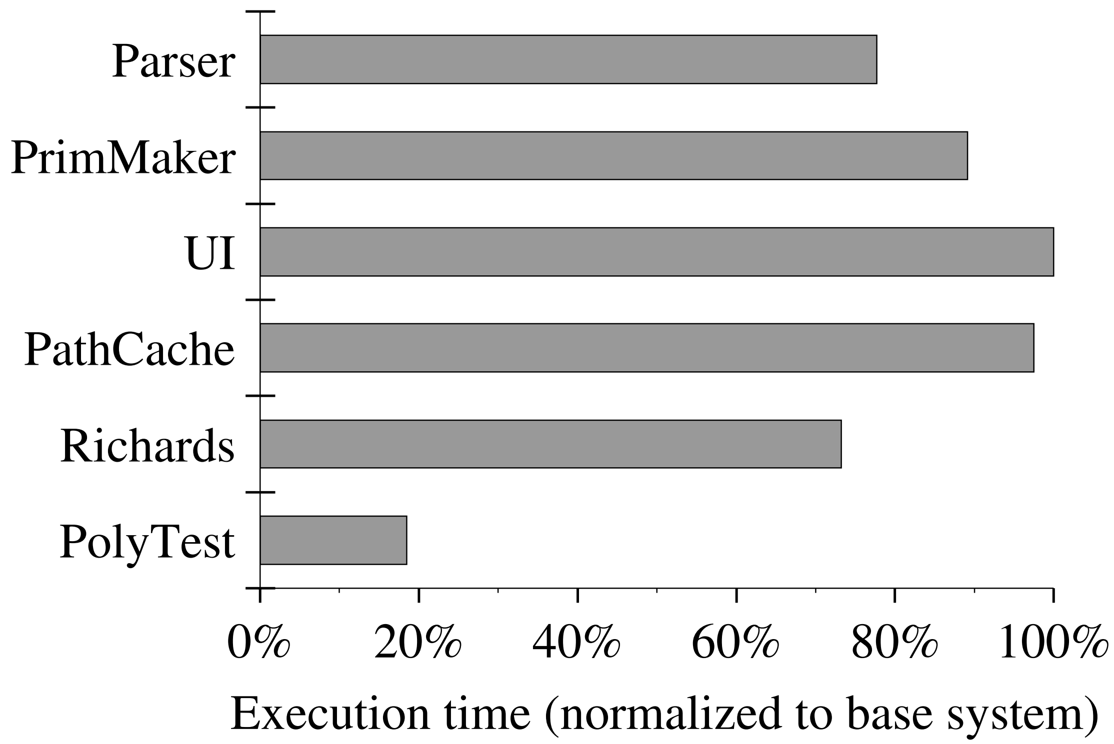


Figure 5. Impact of PICs on performance

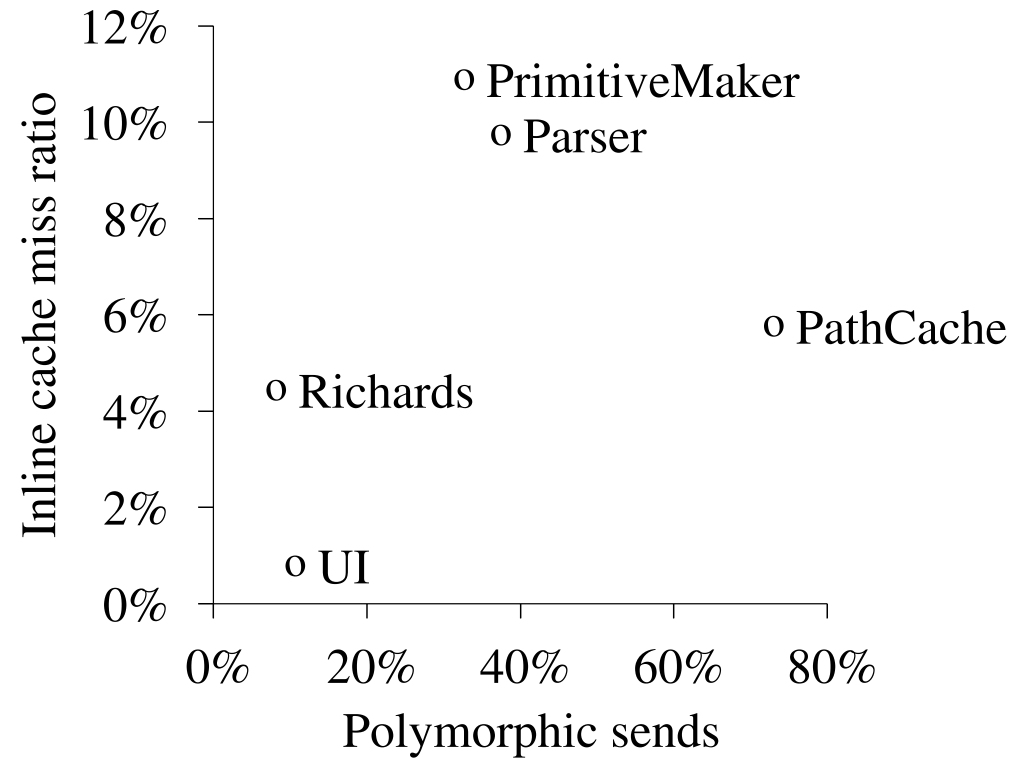


Figure 6. Inline cache miss ratios

Execution times relative to Self system with inline cache

`PolyTest.` An artificial benchmark (20 lines) designed to show the highest possible speedup with PICs. `PolyTest` consists of a loop containing a polymorphic send of degree 5; the send is executed a million times. Normal inline caches have a 100% miss rate in this benchmark (no two consecutive sends have the same receiver type).

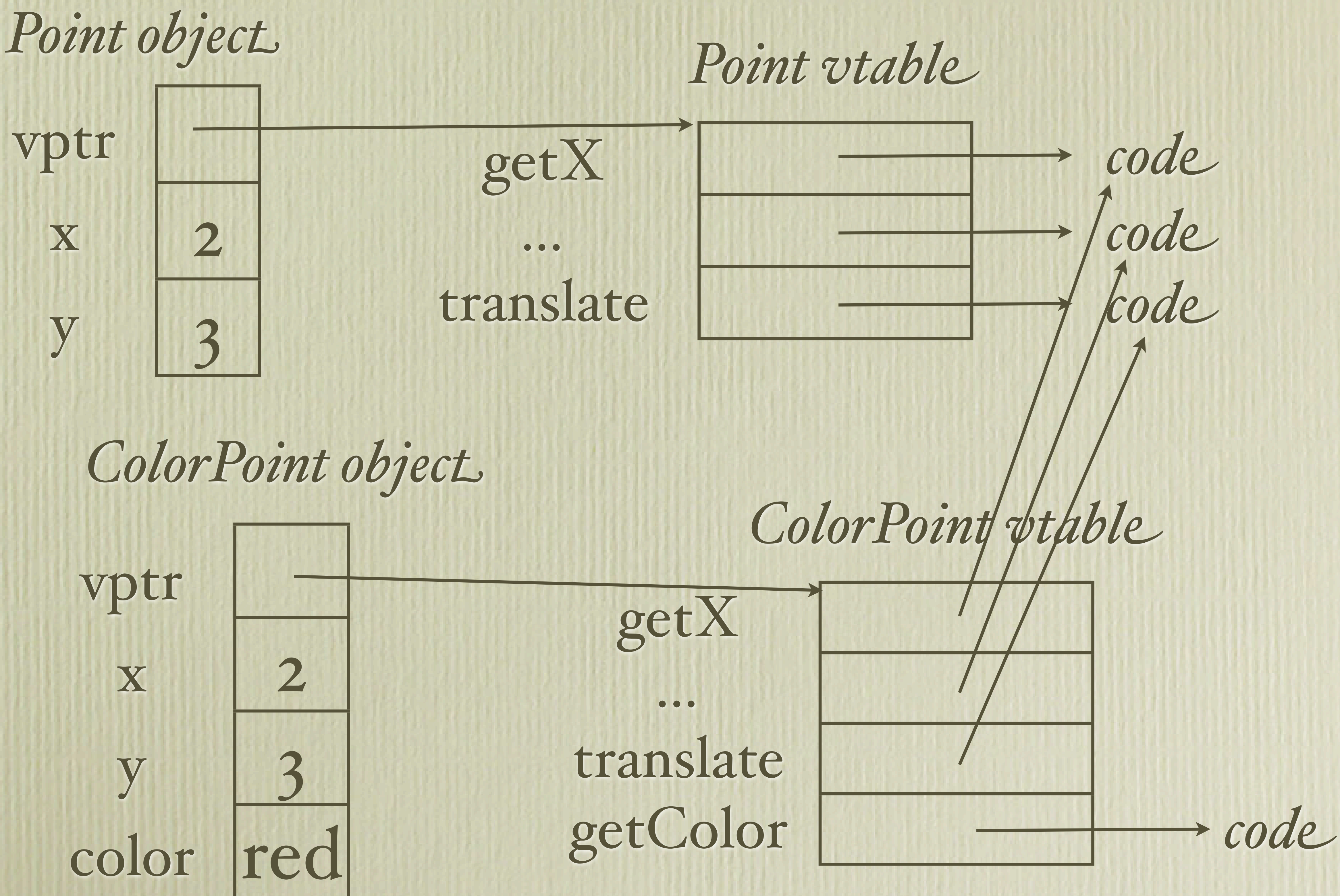
Why Inline Caches Win

- They replace indirect calls by direct calls
- Modern hardware optimizes direct calls, *e.g.*, with pipelining and lookahead
- The direct call is "right" 95% of the time

Another Approach

- Use indirect calls
  - Compile the method name to a small integer that is used as a table index
- Every class has it's
  - x method at offset 0, its
  - y method at offset 1, its
  - printOn method at offset 2, etc.

VTable for Virtual methods



vTables

- use multiple indirection instead of search
- hard to do with multiple inheritance
- a great source of research papers
- loose on modern architectures
    - no branch prediction through indirect call

AbCon Vectors