

Reflection in Pharo: Beyond Smalltak

Marcus Denker

<http://www.pharo.org>



Talk held at PharoDays 2016

Everything is an Object

Classes

Methods

The Stack

Everything is an Object

Code?

AST: Abstract Syntax Tree

AST in Pharo5

- AST of the Refactoring browser
 - Transformation
 - Visitors
 - Annotations (properties)
- Deeper integrated:
 - Pretty Printing, Syntax Highlight, Suggestions
 - Compiler uses RB AST

AST in Pharo5

- Easy access
 - #ast
 - Demo: method and block

DEMO

(OrderedCollection>>#do:) ast.

[1 + 2] sourceNode == thisContext method ast blockNodes first

- ASTCache: as twice, get the same answer
(flushed on image save for now)

AST + Tools

The image shows a 'Playground' application with two main panels. The left panel displays an Abstract Syntax Tree (AST) for a method node, and the right panel displays the corresponding source code.

Left Panel (AST Tree):

- ▼ RBMethodNode(do: aBlock "Override the supercl...
 - RBArgumentNode(aBlock)
 - ▼ RBSequenceNode(firstIndex to: lastIndex do: [:index |
 - ▼ RBMessageNode(firstIndex to: lastIndex do:
 - RBVariableNode(firstIndex)
 - RBVariableNode(lastIndex)
 - ▼ RBBlockNode([:index | aBlock value: (array at: index)
 - RBArgumentNode(index)
 - ▼ RBSequenceNode(aBlock value: (array at: index)
 - ▼ RBMessageNode(aBlock value: (array at: index)
 - RBArgumentNode(aBlock)
 - ▼ RBMessageNode((array at: index))
 - RBVariableNode(array)
 - RBArgumentNode(index)

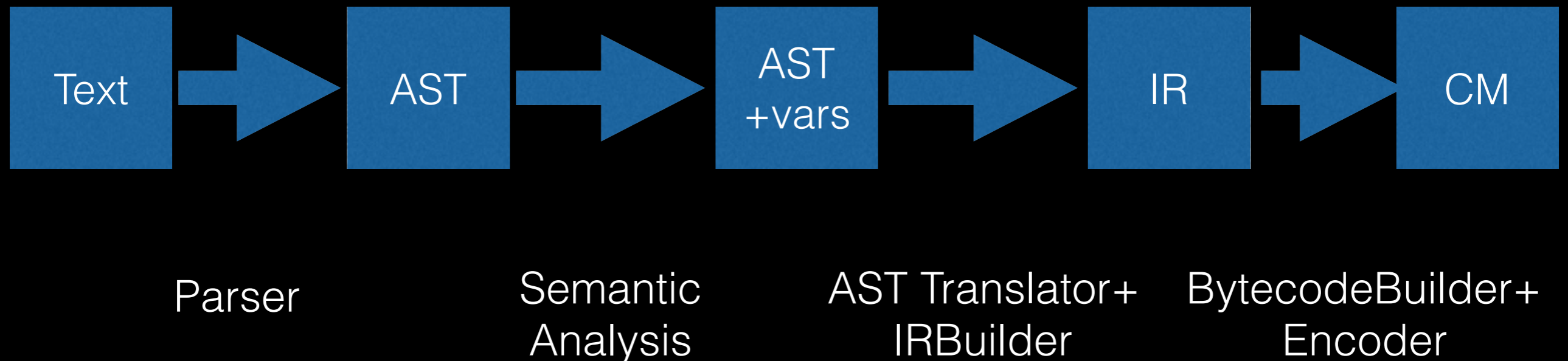
Right Panel (Code Editor):

```
do: aBlock
  "Override the superclass for performance
  reasons."

  firstIndex to: lastIndex do: [ :index |
    aBlock value: (array at: index) ]
```

Opal Compiler

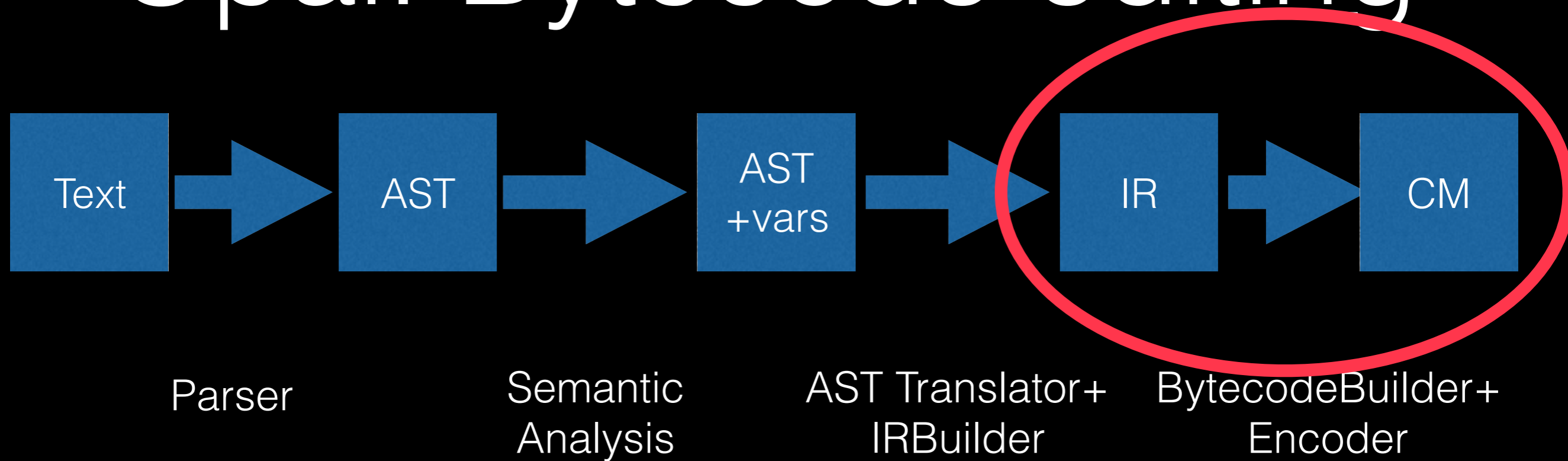
- Uses RB AST
- Based on Visitors



Opal: API

- All staged are Pluggable
 - e.g Semantic Analyzer or Code Generator can be changed.
 - compiler options

Opal: Bytecode editing



- IR can be used to manipulate methods on a bytecode level

Too complicated

Too low level

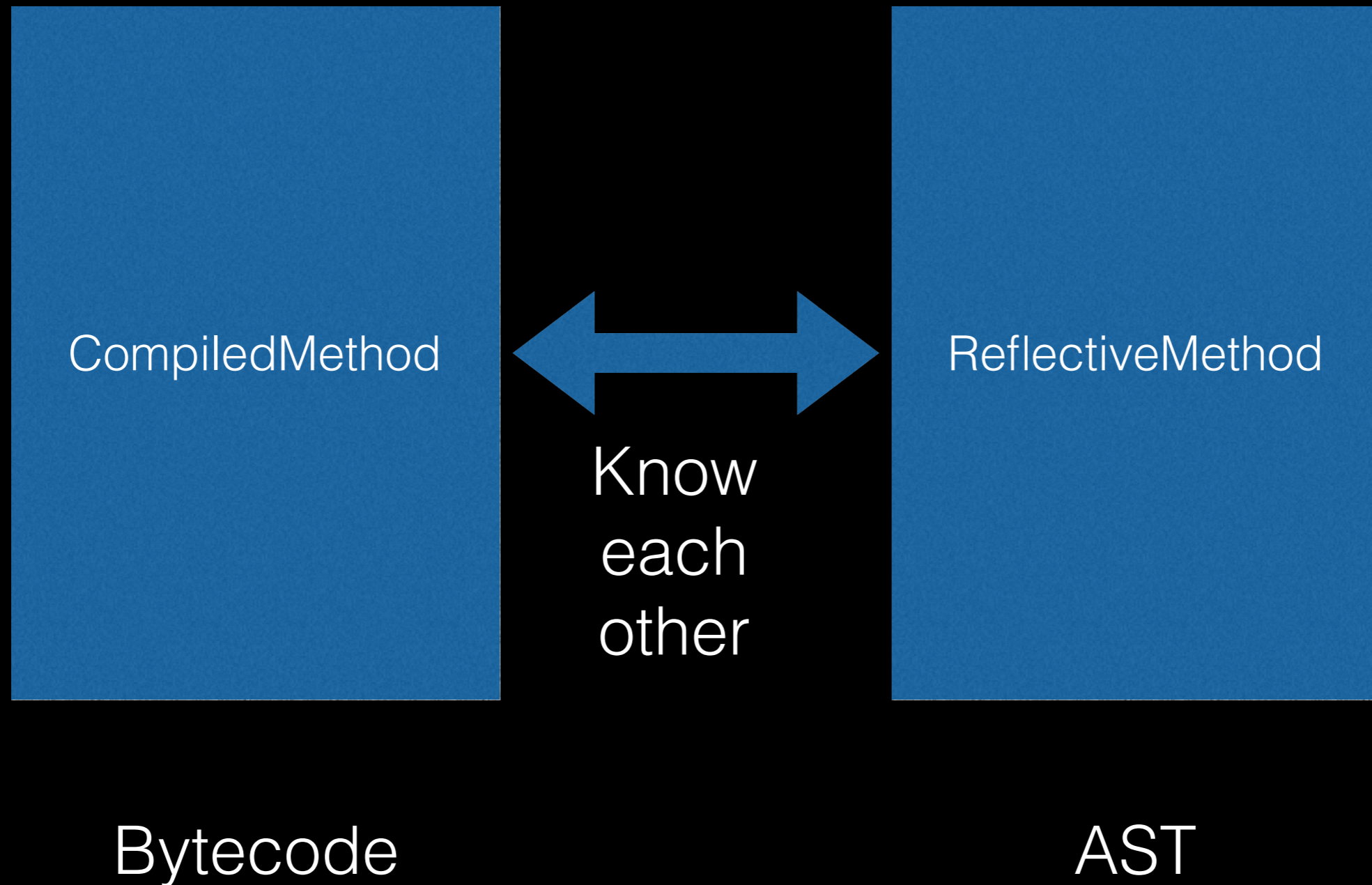
Can we do better?

AST Meta Annotation

- We have an AST with properties
- We have Opal with Pluggable API

Can't we use that?

Basis: the Evil Twin



Basis: the Evil Twin

```
run: aSelector with: anArray in: aReceiver  
  self installCompiledMethod.  
  self recompileAST.  
  self installCompiledMethod.  
  ^compiledMethod  
    valueWithReceiver: aReceiver  
    arguments: anArray
```



ReflectiveMethod

AST

Demo: Morph

- Morph methods do: `#createTwin`
- Morph methods do: `#invalidate`
- inspect “Morph methods”

Putting it together

- Annotate the AST
 - Create Twin if needed
 - Invalidate method
- Next call: generate code changed by annotation

Annotations?

MetaLink

DEMO: Simple Link

```
node := (ReflectivityExamples>>#exampleMethod) ast.
```

```
link := MetaLink
```

```
  new metaObject: (Object new);
```

```
  selector: #halt.
```

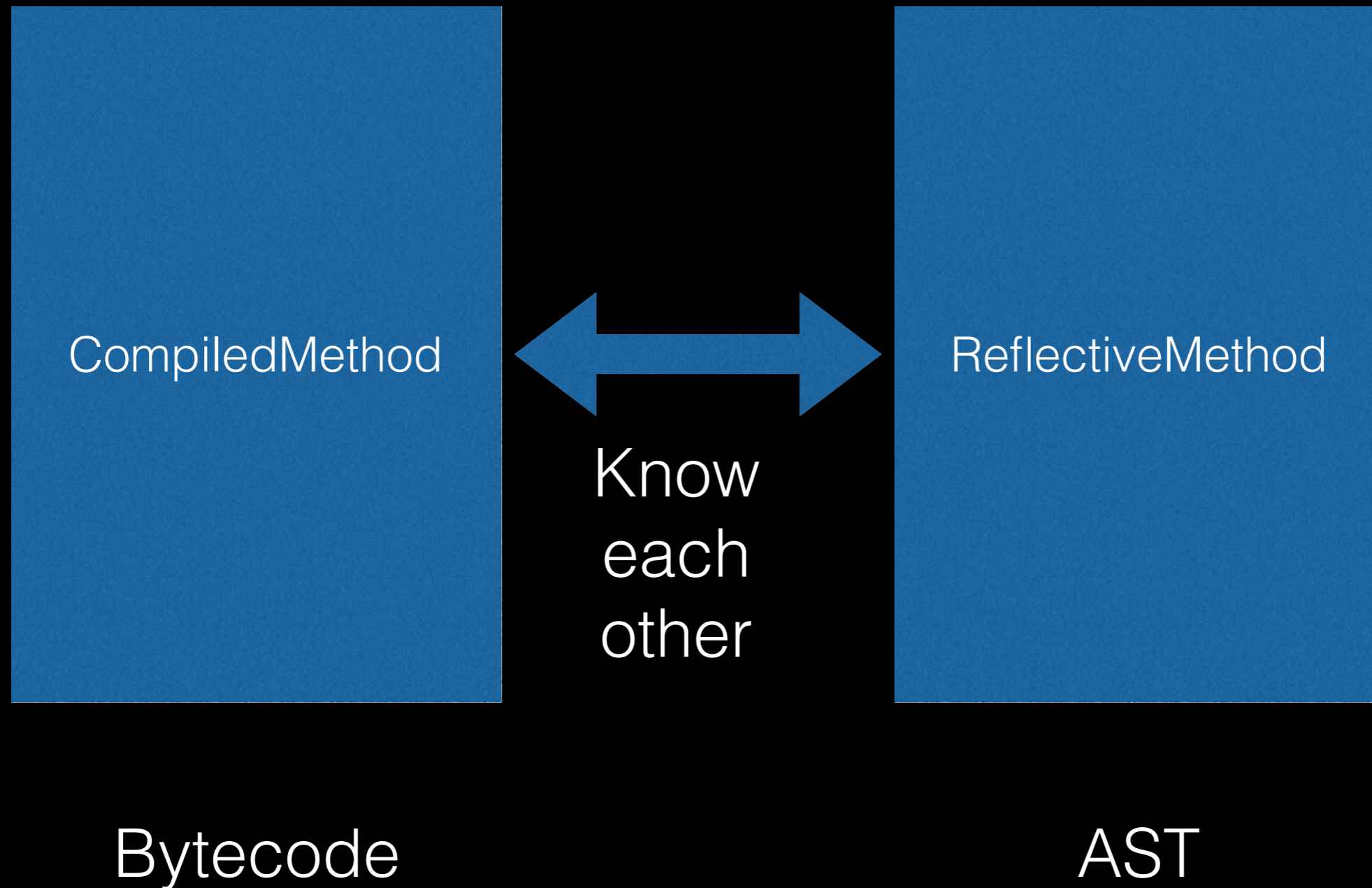
```
node link: link.
```

```
ReflectivityExamples new exampleMethod
```

Meta Link

- When setting link:
 - create twin if needed
 - install reflective method
- On execution
 - generate code and execute, install CM

Twin Switch



Link: metaobject

The object to send
a message to

```
link := MetaLink new  
      metaObject: [self halt]
```

Link: selector

The selector to send

```
link := MetaLink new  
.....  
selector: #value
```

Link: control

before, after, instead

```
link := MetaLink new
```

```
.....
```

```
control: #after
```


Link: control

after: #ensure: wrap

```
link := MetaLink new
```

```
.....
```

```
control: #after
```

Link: control

instead: last link wins
(for now no AOP *around*)

```
link := MetaLink new
```

```
.....
```

```
control: #instead
```

Link: condition

boolean or block

```
link := MetaLink new
```

```
.....
```

```
condition: [self someCheck]
```

Link: arguments

what to pass to the meta?

Reifications

- Every operation has data that it works on
- Send: #arguments, #receiver, #selector
- Assignment: #newValue, #name
- All: #node, #object, #context

Link: arguments

what to pass to the meta?

```
link := MetaLink new
```

```
.....
```

```
arguments: #(name newValue)
```

Reifications: condition

```
link := MetaLink new  
      condition: [: object | object == 1];
```

Virtual meta

- Reifications can be the meta object

```
link := MetaLink new
  metaObject: #receiver;
  selector: #perform:withArguments::;
  arguments: #(selector arguments).
```


Statement Coverage

```
link := MetaLink new  
    metaObject: #node;  
    selector: #tagExecuted.
```

“set this link on all the AST nodes”
(ReflectivityExamples>>#exampleMethod) ast
 nodesDo: [:node | node link: link].

Users

- BreakPoints Pharo5
- Coverage Kernel
-

Everything is an Object

Everything is an object?

SmalltalkImage classVarNamed: #CompilerClass
==> returns value

Object binding class
==> Association

Why not an Object?

Globals/ClassVariables

- We are close: bindings are associations
- Add subclass “LiteralVariable”
- Subclasses GlobalVariable, ClassVariable
- Enhance API

Globals/ClassVariables

SmalltalkImage classVariableNamed: #CompilerClass

Object binding class

Globals: Reflective API

```
global := SmalltalkImage classVariableNamed:  
#CompilerClass
```

```
global read  
global write: someObject
```

+ helper methods + compatibility methods

Everything is an object?

- Point instanceVariables
- 5@3 instVarNamed: 'x'
- 5@3 instVarNamed: 'y' put: 6

Why not an Object?

Slots

Point slots

(Point slotNamed: #x) read: (3@4)

(Point slotNamed: #x) write: 7 to: (3@4)

Variables+MetaLink

- Helper methods

Point assignmentNodes

- But: can't we annotate variables directly?

Variables + Links

- Object binding link: myMetaLink
- (Point slotNamed: #x) link: myMetaLink

(not yet in Pharo5)

Class Template

```
Object subclass: #MyClass  
  slots: { #x => WeakSlot }  
  classVariables: { }  
  category: 'Example'
```

Future

- Can't we model bit patterns and bind them to named virtual slots?
- How to model Array-like layouts better?

Questions ?