

Variables in Pharo5

ESUG 2015

Marcus Denker

<http://www.pharo.org>

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Everything is an Object

Everything?

Classes, yes.

Methods, yes

But Variables?

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?

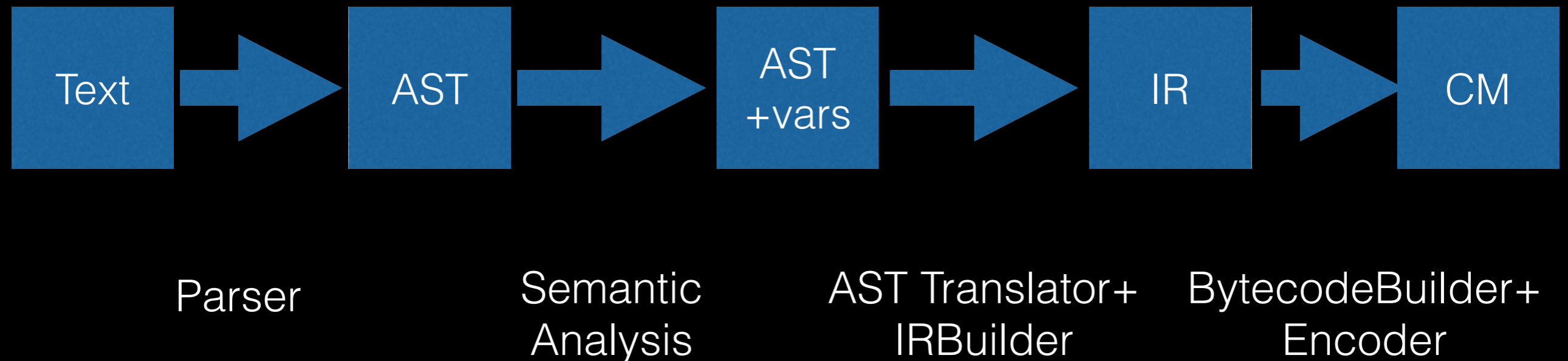
Variables + Links

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

(not yet in Pharo5)

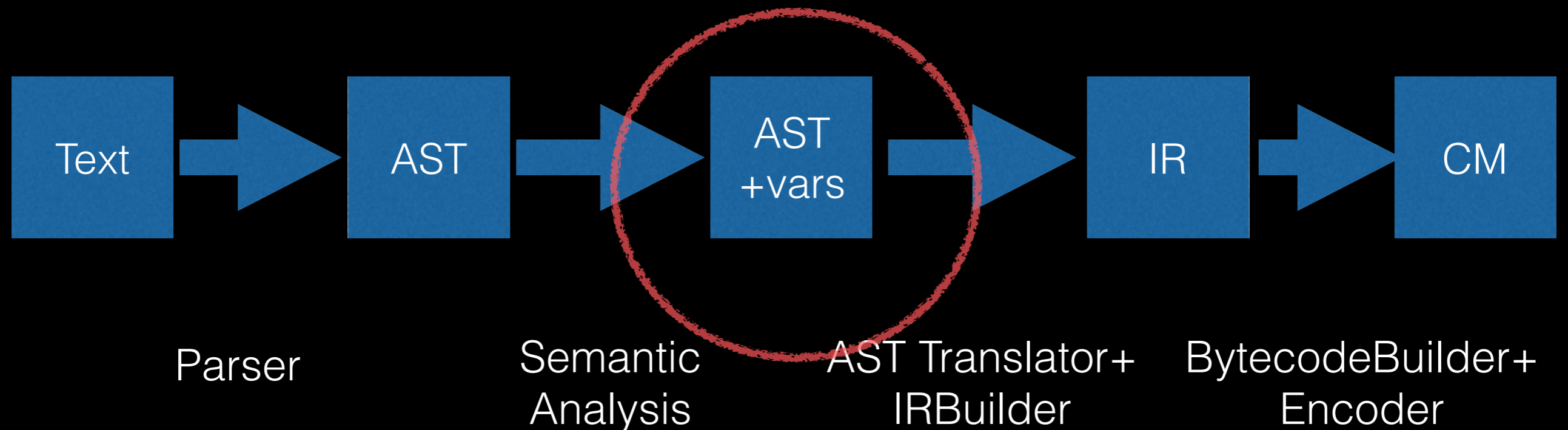
Opal Compiler

- Uses RB AST
- Based on Visitors



Opal Compiler

- Name analysis finds the variables
- Code generator can delegate to them



Gobals: code read

- By default compile reflective read

```
emitValue: aMethodBuilder  
  aMethodBuilder  
  pushLiteralVariable: #slot->self;  
  send: #read
```

Gobals: code write

- By default compile reflective write

```
emitStore: aMethodBuilder
| tempName |
tempName := Object new.
aMethodBuilder
  addTemp: tempName;
  storeTemp: tempName;
  popTop;
  pushLiteralVariable: #global -> self;
  pushTemp: tempName;
  send: #write:
```

Gobals: code write

- ClassVariable and GlobalVariable override

```
emitStore: methodBuilder
```

```
    methodBuilder storeIntoLiteralVariable: self.
```

Same for Slots

- Slot generates reflective read/write
- InstanceVariableSlot overrides for fast instance access via byte code

What does that mean?

- Slots and Globals are instances of a class
- The compiler delegates code generation to the variable meta object
- Which means: We can define our own variables!

Class Template

```
Object subclass: #Point  
  slots: { #x. #y }  
  classVariables: { }  
  category: 'Kernel-BasicObjects'
```


Class Template

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

Examples: DEMO

- Simple Slot
- WeakSlot
- Property Slot
- Boolean

RoadMap

- Pharo3:
 - Layout+Slots (hidden), Opal
- Pharo4
 - Slots: Monticello support, class template
- Pharo5
 - Remove old Compiler/AST
 - Slots + Reflectivity: First finished version

RoadMap

- Pharos6:
 - Library of useful Slots
 - Use e.g. Property Slots in Bloc/Morphic

Future

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

Thanks!

- Work of many people...
- special thanks to... Toon Verwaest, Camillo Bruni, Martin Dias, Stephane Ducasse, Max Mattone and Camille Teruel

Questions ?