A JIT as a Central System Service

Marcus Denker

What's this all about?

* "Normal" Point of View:

A JIT makes the sytem faster

* This is nice, but what else can it do?

--> Run time translation can make the System simpler

This talk was put together pretty quickly, so it's more like a braindumb...

SXTalk.image

Overview

1. Some Basics:
   -> The current System
   -> J3
   -> AOStA

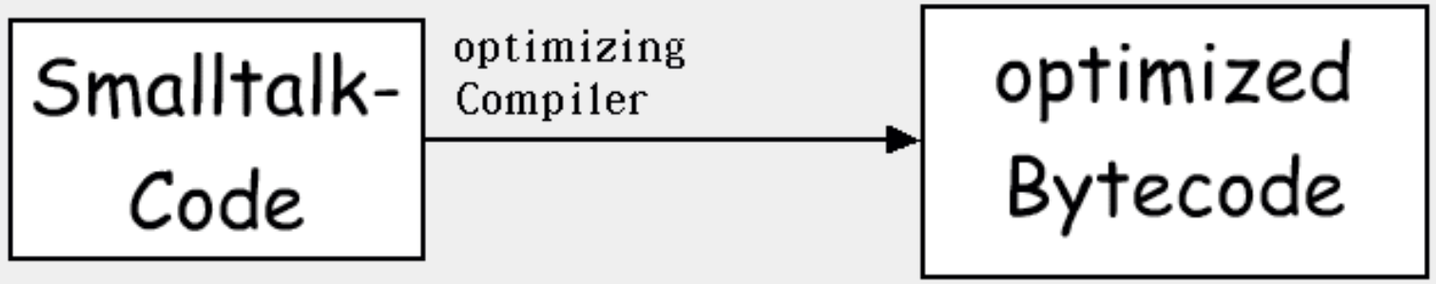
2. An idea for a future System
   -> Overview

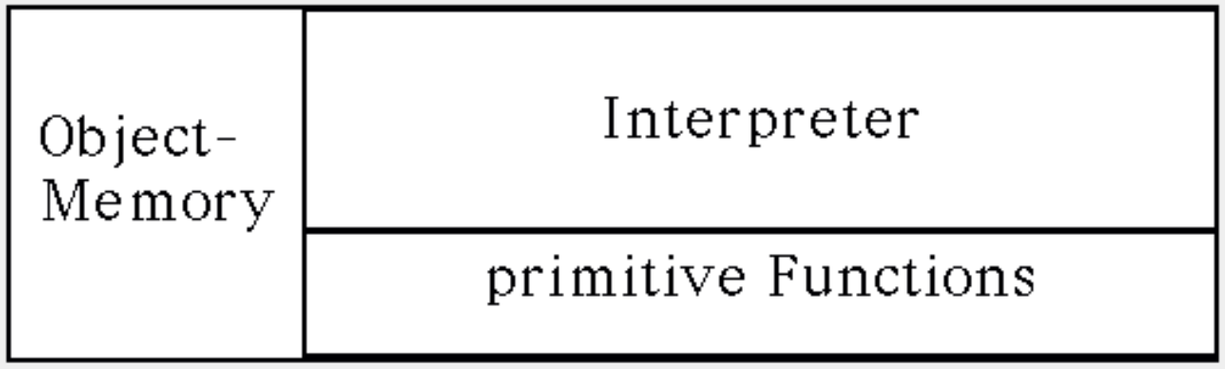
3. Current state of the System

4. Consequences:
   -> Late binding the execution

The current System

Current System: - Virtual Machine
- Bytecode Interpreter





Problems Interpreter

Positive:

- extremely portable
- fast enough
- simple

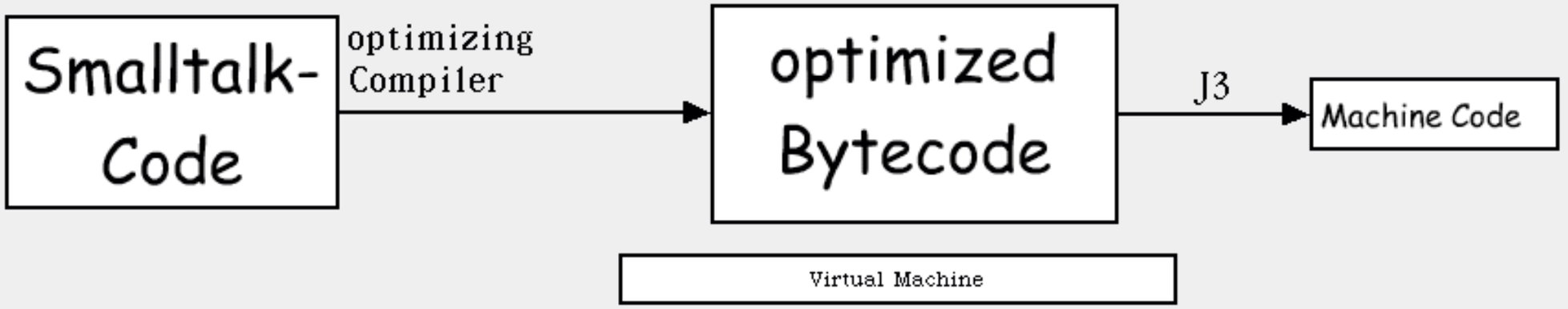
Problems:

- Bytecode very "low level"
- e.g. offset-encoded instVars,
  hard-coded Globals
- Optimizations semantically *wrong*
- could be faster

J3 Jit Compiler

- classical runtime translator

- Goal: Faster Bytecode execution



- J5: Same, but with dynamic feedback optimization

SXTalk.image

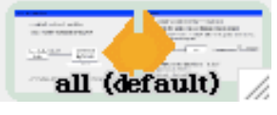
Problems J3

Positive: Faster

- 3 times: Bytecode Execution
- 7 times: Sends

Problems:

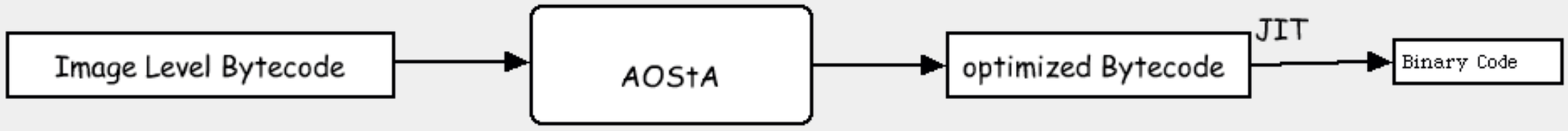
- Implemented in C++
- No Solution for the bytecode optimization problem
- The Goals:
  - -> Perfomance
  - -> Full Transparency: Do not change anything



SXTalk.image

AOStA

- Adaptive Optimization in Smalltalk
- Eliot Miranda, Claus Gittinger, Paolo Bonzini
- Idea: Optimizer in Smalltalk, Code generation using normal JIT Compiler



- Optimizing Compiler (uses dynamic FeedBack)
- implemented in Smalltalk
- Bytecode --> Bytecode translator

Problems AOStA

Postitive:

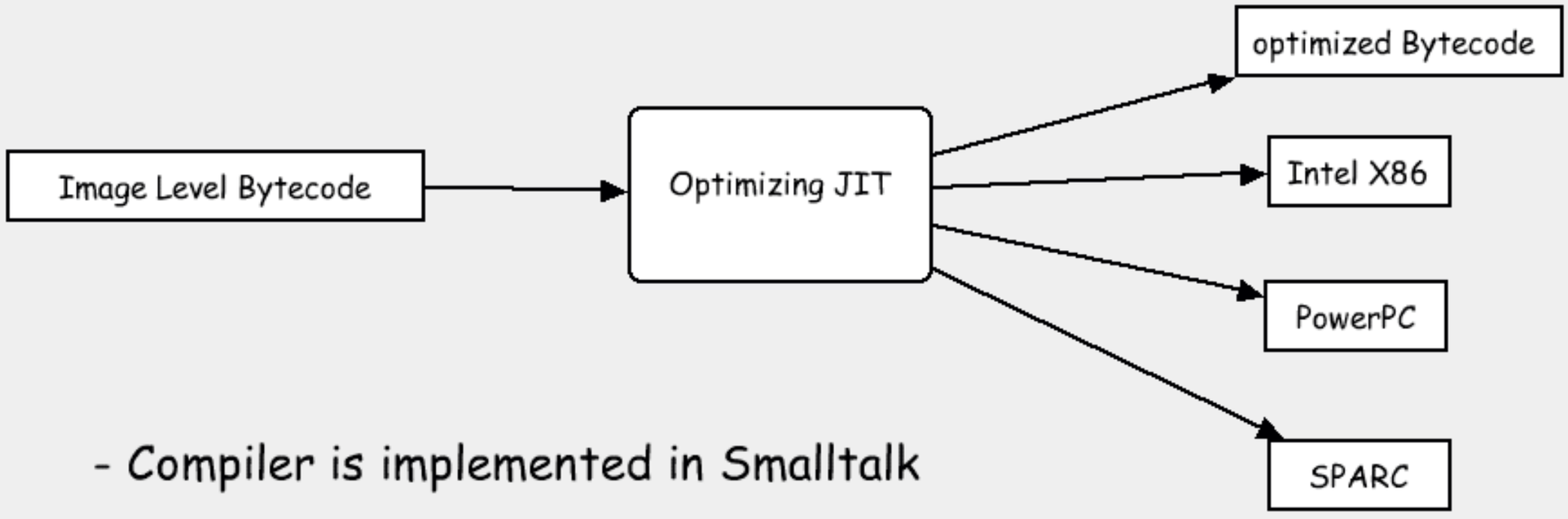
- written in Smalltalk
- overall a nice Design

Problem:

- Goal is "performance only"

JIT as a central service

- We JIT everytime, even when running the Interpreter



- Compiler is implemented in Smalltalk

SXTalk.image

Implemented in Smalltalk

- We do not want C++ (Ian might disagree ;-)

- Slang is Evil!

JIT Should be implemented in Smalltalk

-> much easier to understand and modify

SXTalk.image

What's there

-> Only some Experiments based on AOStA

1. Experimental System for compilinng
   methods "Just in time" before execution.

2 . AOStA SSA-Framework ported to Squeak.

3 . Started a CodeGen for AOStA

SXCompiledMethods

- very simple, prototypical system
- "compile before run"
- based on ObjectsAsMethods feature of Squeak 3.6

- Idea: Compiler installs instances of "SXCompiledMethod" in MethodDictonary.

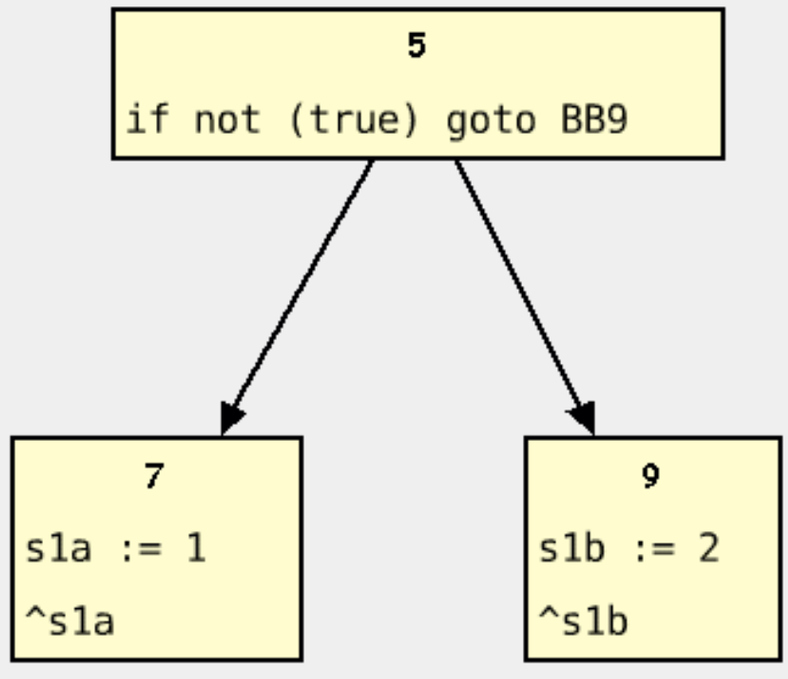
- SXCompiledMethod generates a "real" CompiledMethod on first execution.

---> Hook for a Compiler

AOStA: Good intermediate form for optimizations

- Has jumps (direct and conditional) in addition to Assignment and Methodcalls
- Good for optimizations
- Simple Example: IfTrue:

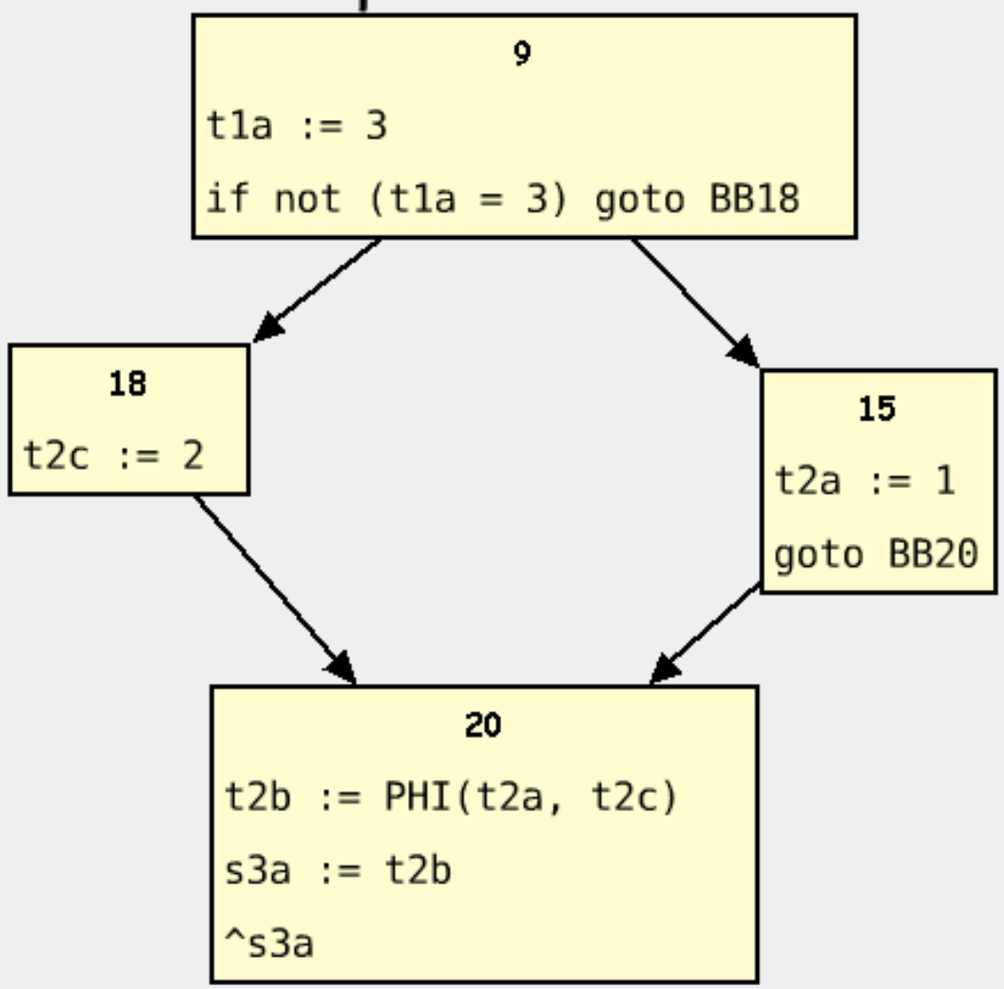
```
t4
  ^true ifTrue: [1] ifFalse: [2]
```



AOStA

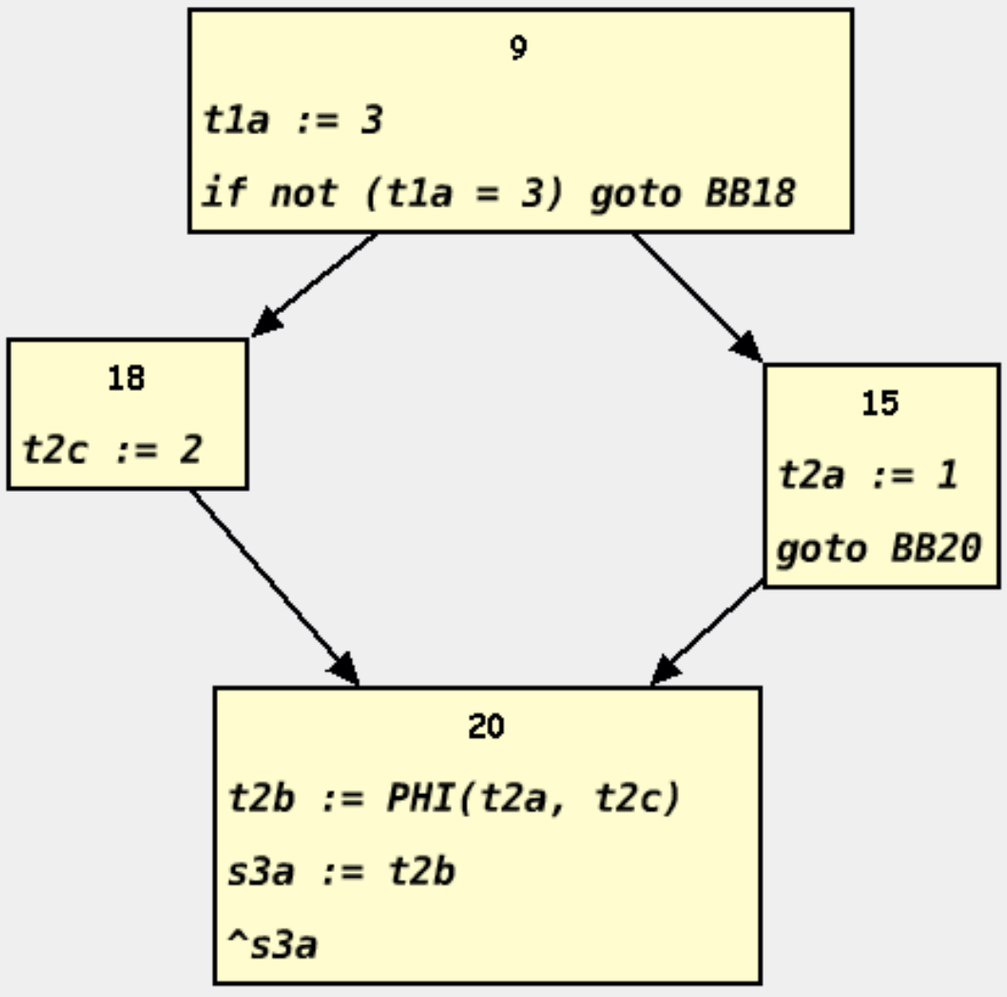
- SSA: Single Static Assignment Form:
  - Each Variable has *one* assignment
- If Controllflow merges, we need to select the variable comming from the path we came from.

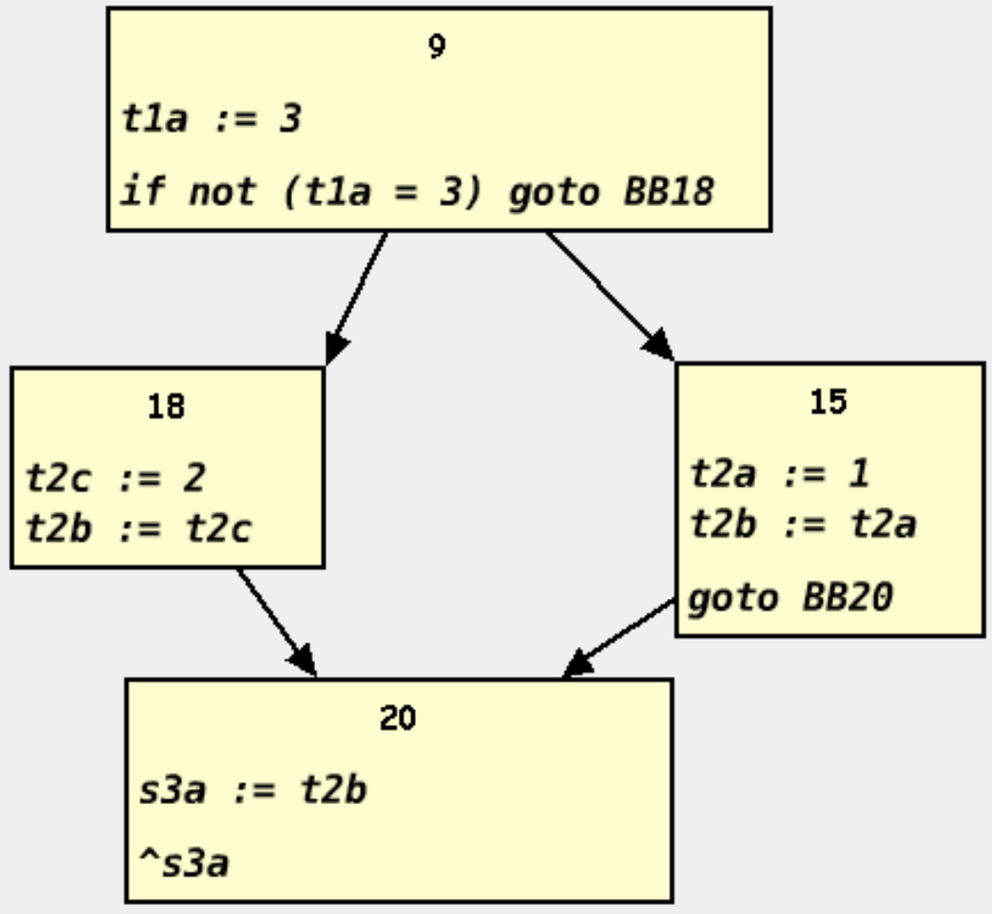
```
examplePhiSimple
   | a b |
a := 3.
a = 3
   ifTrue: [b := 1]
   ifFalse: [ b := 2 ].
^b
```



AOStA CodeGen 1: Remove PHI-Functions

- Before CodeGen is possible, PHI-Functions need to be removed.





Code Generation

Code Generation: a Simple Visitor

System Browser: AOStACodeGenVisitorTest

AOStA-Core
AOStA-Experiments
AOStA-Misc
Compiler-Standalone
Tests-Compiler-IR
Tests-AOStA
Tests-SX
Connectors-Base
Connectors-Demo
Connectors-FSM
Connectors-Info
People-nk-Broom

AOBytecodeToStaticSingleAssig
AOComputeStackHeightsTest
AOInstructionReaderTest
AOOptimizerTest
AOStACodeGenVisitorTest
AOStAExamples
RemovePhiFunctionVisitorTes
SSAGraphGenVisitorTest

instance | ? | class

-- all --
testing - executable
testing - mock
as yet unclassified

tests | ! | slint

testExecExampleIfTrueIfFalse
testExecExampleIfTrueIfFalse2
testExecExampleLocal
testExecExamplePhiSimple
testExecExampleReturn1
testExecExampleSend
testExecExampleSendCasCade
testExecExampleSendParam
testExecExampleSendPlus
testExecExampleWhileFalse
testExecExampleWhileTrue
testGenMethodT0

browse | senders | implementors | versions | inheritance | hierarchy | inst vars | class vars | :-) | source

```
testExecExamplePhiSimple
    | aOMethodNode method |
    aOMethodNode := AOBytecodeToStaticSingleAssignment
                        parse: (AOStAExamples compiledMethodAt: #examplePhiSimple).

    method := self genMethodForTree: aOMethodNode.

    self assert: ((method valueWithReceiver: nil arguments: #()) value = 1)
```

Next Steps.....

- More Debugging of AOStA:
  * Blocks

- Debug CodeGen

- Integrate with the SXCompiledMethdods

GOAL: Run the whole image this way

Future

- Bytecode can be optimized:
  Correct semantik preserving optimizations
  using TypeFeedback

- Imagelevel Bytecode and Interpreter-Bytecode
  can be different:
  => Latebinding of the execution format

- Imagelevel Bytecode can be simple:
  => No optimizations at all!

SXTalk.image

"2 Worlds"

Software-Eng

* AST

* late bound

* no Optimizations

JIT===>

Execution

* Bytecode or Binary

* optimized

* late Binding resolved

Squeak

Tools

Navigator

Connectors FSM Class Diagrams Widgets all (default)