Pharo Bytecode Compiler Roadmap 2022

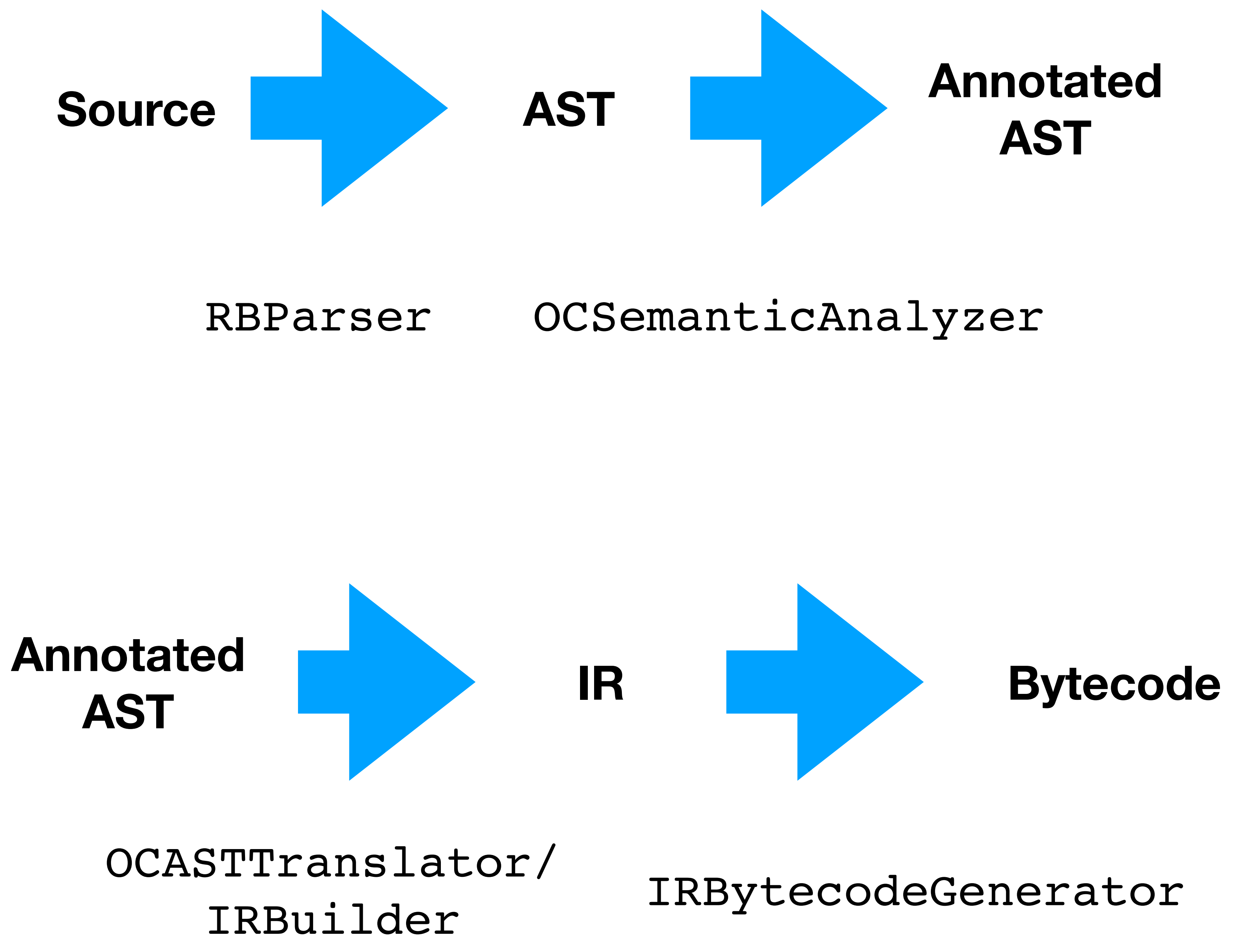
Marcus Denker, Inria

http://marcusdenker.de

The Compiler

- `Smalltalk compiler` -> Compiler Facade
- But the real work is done by a set of visitors

The Compiler



Done in 2021

- Simplified AST: RBVariableNode subclasses removed
- Name Analysis redone: uses Variable Hierarchy
  - Much simpler!
  - Debugger uses variable meta objects to read/write
  - Old DebuggerMethodMap API finally removed

Done in 2021

- Removed support for inlined Blocks and old Bytecode set
- Clean Blocks: pre-compile blocks that do not access outer variables / need outer block (not active)
- Literals are compiled as read-only objects

What needs improvement?

- It is still too complex for what it is
- We need some new features
- Better Documentation / Tutorial

Why is it difficult

- The Compiler is not just a library that is used
- Improvements are often not just internal, but have impact everywhere
    - Lots of interactive usage via complex APIs
    - Good solutions often at the Language Kernel level (e.g. reflective API), not the compiler

Backward Compatibility

- Compiler was designed to be backward compatible
  - Exceptions
  - API
  - Exact Bytecode emitted

Exceptions are a mess

- A syntax errors is no Error but SyntaxErrorNotification
- OCSemanticWarning is not a Warning
- Really complex:
  - call #notify:at:in: to print into the editor
  - Suggest what to do (UI!) for Undeclared vars

Exceptions are a mess

- We still support ST80 #failBlock: (exception handling of before exceptions where introduced)
- Oh, and there is ReparseAfterSourceEditing (aargh…)
- And then, with all that, in non-interactive mode we just write to the transcript and compile

Proposal: No Exceptions

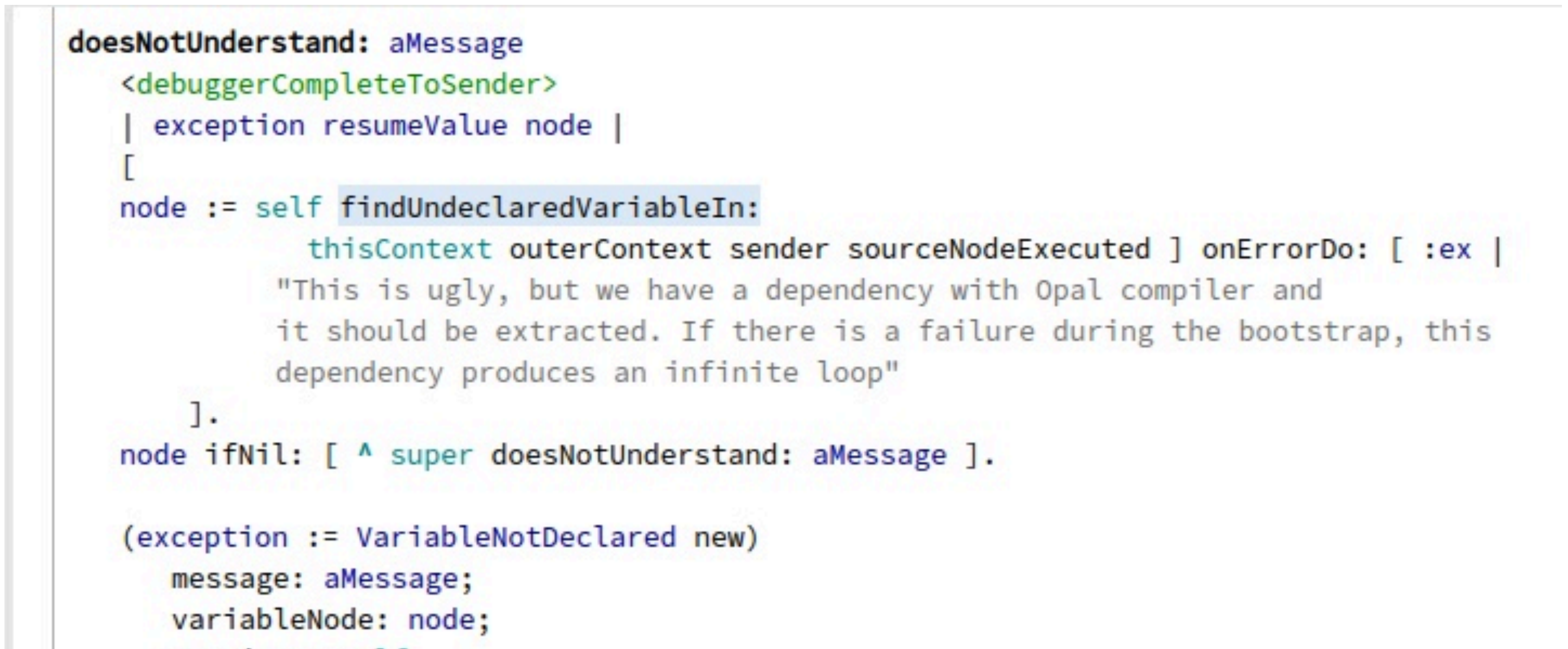
- Why not compile and let the Tools handle what to do?
  - Remember: this is all *only* for interactive use !
- We log the errors
- Tools provide UI to the user to fix what is broken

Proposal: Logging

- Instead of Transcript, log Objects describing what happened
- The UI of the logger can provide “fix it” buttons for all problems encountered.
- Both useful in interactive and non-interactive use

Problem with TDD

- If we have compiled an Undeclared, there is no way to interact with the programmer
- How we turn a DNU on UndefinedObject into a variable definition popUp.



```
doesNotUnderstand: aMessage
   <debuggerCompleteToSender>
   | exception resumeValue node |
   [
   node := self findUndeclaredVariableIn:
              thisContext outerContext sender sourceNodeExecuted ] onErrorDo: [ :ex |
            "This is ugly, but we have a dependency with Opal compiler and
            it should be extracted. If there is a failure during the bootstrap, this
            dependency produces an infinite loop"
      ].
   node ifNil: [ ^ super doesNotUnderstand: aMessage ].

   (exception := VariableNotDeclared new)
      message: aMessage;
      variableNode: node;
```

Idea to improve TDD

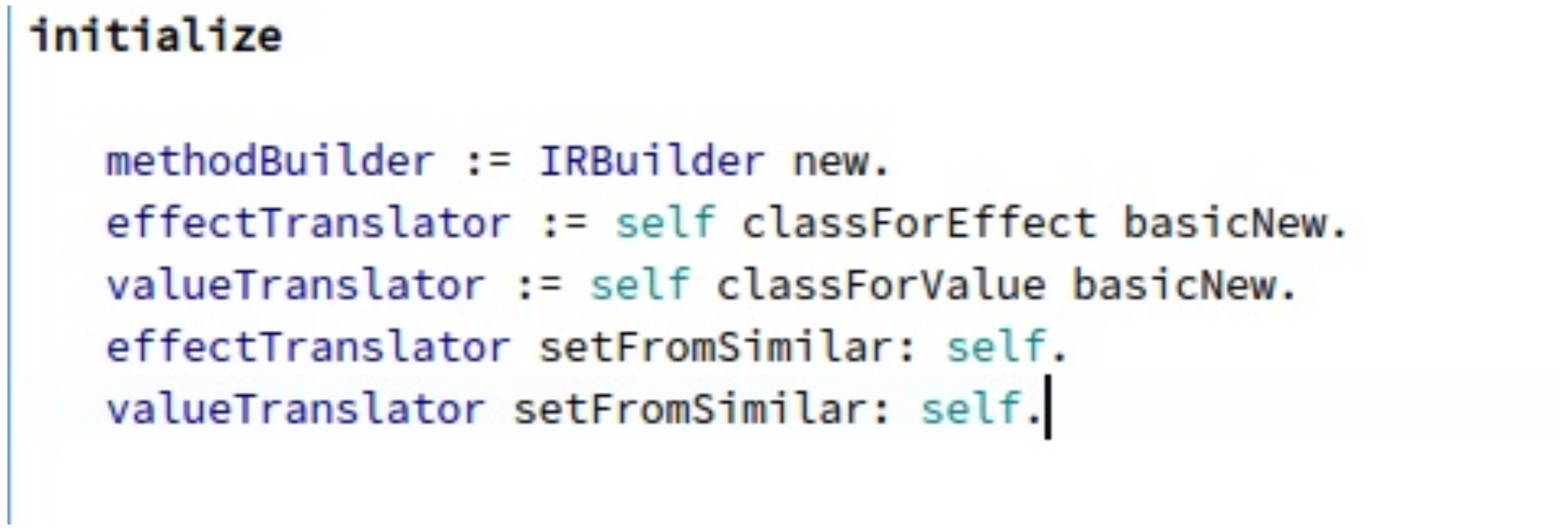
- Compile Undeclared access as message send to the UndeclaredVariable instance
- This would then allow us to execute code at read / write to prompt interaction to declare the variable
    - evaluating “<undeclared> new” then would work as it does now: prompting a fix in interactive mode

Improve Playground Variables

- Review automatic variable definition in the Playground
  - Source of many problems in the past
  - Code very hard to understand
  - Better: ask the developer before defining a var

Improve Structure

- Revise the odd implementation idea off the two subclasses of OCASTTranslator (one for value, one for effect)



```
initialize

   methodBuilder := IRBuilder new.
   effectTranslator := self classForEffect basicNew.
   valueTranslator := self classForValue basicNew.
   effectTranslator setFromSimilar: self.
   valueTranslator setFromSimilar: self.
```

- Lots of logic is implemented on the level of the RBMethodNode and IRMethod (Idea from ST80)
  - AST holds on to IR representation

Simplify code gen optimised code

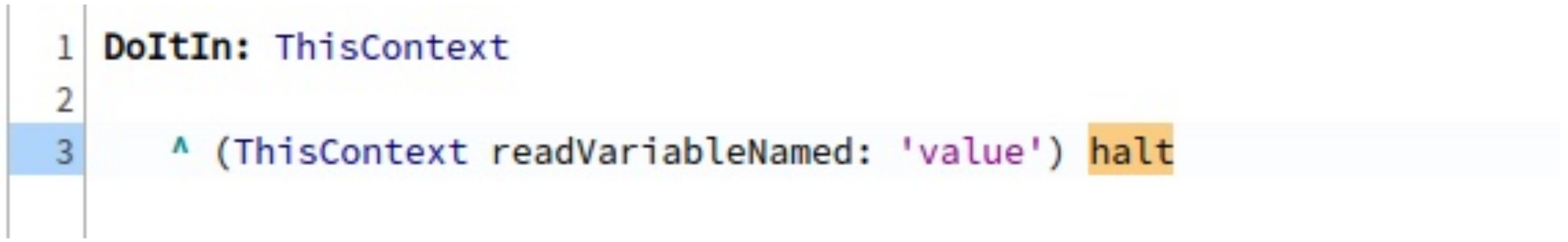
- Originally, we tried to emit the same bytecode as ST80 Compiler
  - Lots of cases hard coded
- Do we need it from the VM side?
  - Benchmark!

Evaluate Code Gen

- OCASTTranslator now directly emit optimised code (ifTrue: …)
    - Evaluate if we can not do this as a second pass
- Start to evaluate backend: Too complex!
- This is for later

Simplify DoItIn:

- Simplify DoItIn: Evaluation
  - Very slow and odd due to AST rewrites



```
DoItIn: ThisContext

    ^ (ThisContext readVariableNamed: 'value') halt
```

  - Denis did a first step to simplify
  - Evaluate: Can we compile DoIts to Closures, not Methods?

Simplify Tool API

- SpCodeInteractionModel shows the complexity that the compiler forces on the tools
  - 5 subclasses, lot of methods
  - This should not need so much code!
  - And definitely not on this level!

Compiler Plugins

- Better Compiler Plugin Infrastructure
  - Plugins for different phases
- Revisit #compilerClass / #compiler
- Allow the compiler to be set in the Fluid Class Definition

Meta Data

- Compiler meta-data need to be stored in the CompiledMethod
  - We e.g do not know if a method was compiled with non-standard options
  - Maybe encode flags as an integer in the literals?

Clean Blocks

- We should enable Clean Blocks by default
- Provides some speedup (~5-7% for e.g. compiler recompiling the image)
- reduces memory due to not referencing the outer context
- Compiler + Debugger works, but Fuel needs work
  - Need to test the new version

Documentation

- Need to have a booklet describing the compiler
- Need a tutorial
    - Show how to extend the compiler by subclassing
    - Show how to define code gen for your own Variables