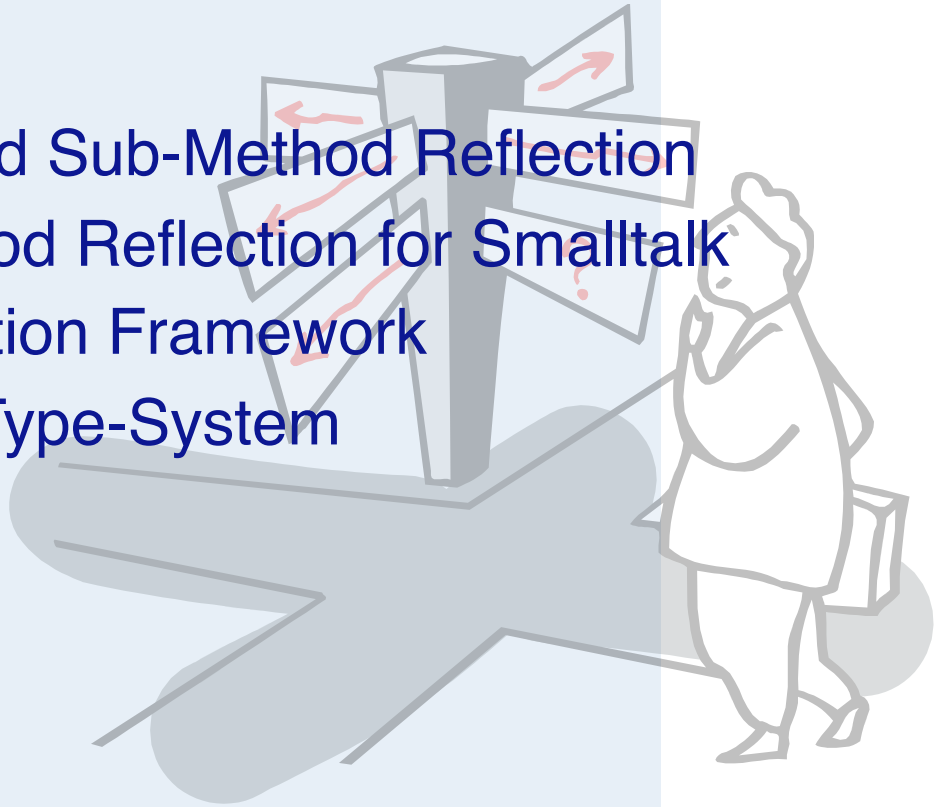
UNIVERSITÄT
BERN

Sub-Method Reflection

Marcus Denker
Stéphane Ducasse
Adrian Lienhard
Philippe Marschall

Roadmap

- Structural Reflection and Sub-Method Reflection
- Persephone: Sub-Method Reflection for Smalltalk
- Example I: Instrumentation Framework
- Example II: Pluggable Type-System
- Benchmarks + Memory
- Future Work



Structural Reflection

- Structure modeled as objects
  - e.g. Classes, methods
  - Causally connected
- Uses:
  - Development environments
  - Language extensions and experiments

Methods and Reflection

- Method are Objects
  - e.g in Smalltalk
- No high-level model for sub-method elements
  - Message sends
  - Assignments
  - Variable access
- Structural reflection stops at the granularity of methods

Sub-Method Reflection

- Many tools work on sub method level
  - Profiler, Refactoring Tool, Debugger, Type Checker
- Communication between tools needed
  - example: Code coverage
- All tools use different representations
  - Tools are harder to build
  - Communication not possible

Existing Method Representations

- Existing representations for Methods
  - Text
  - Bytecode
  - AST

Requirements

- Causal Connection
- Abstraction Level
- Extensibility
- Persistency
- Size and Performance

Text

- Low level abstraction
  - String of Characters
- Not causally connected
  - Need to call compiler

Bytecode

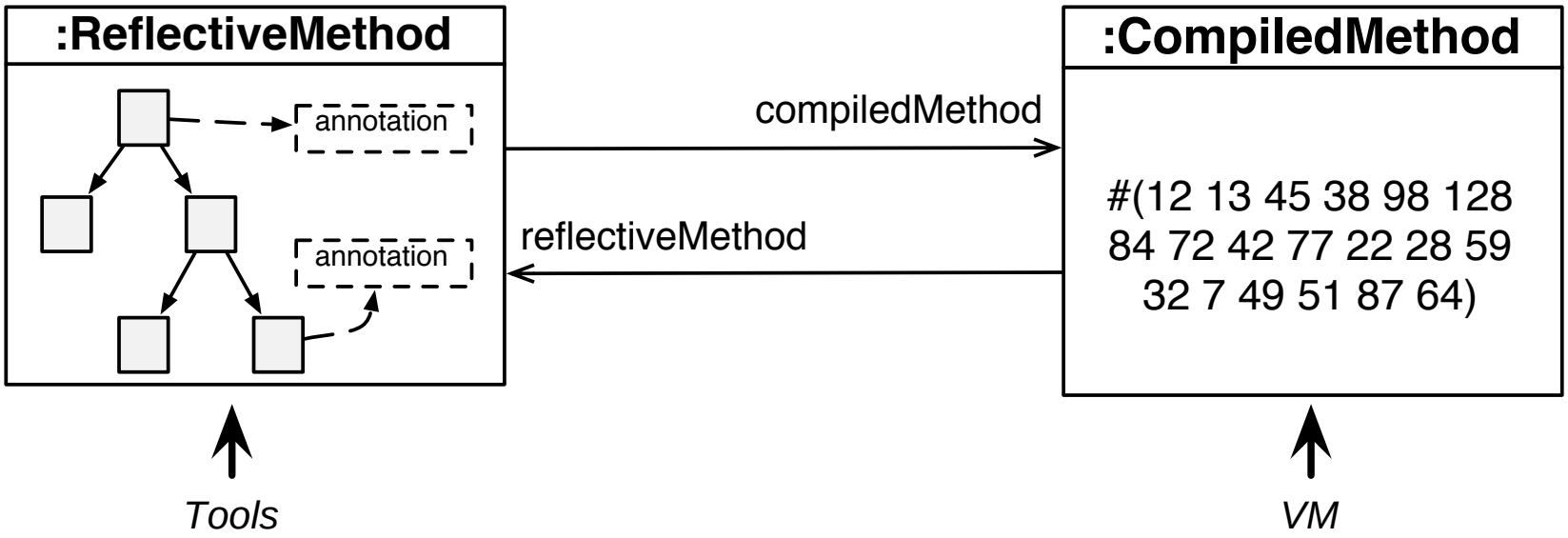
- Low level abstraction
  - Array of Integers
- Missing extensibility
  - e.g. for tools
- Mix of base- and meta-level code
  - Problems with synthesized code when changing code
  - Examples: AOP point-cut residues, reflection hooks

Abstract Syntax Tree

- Not causally connected
  - Need to call compiler
- Not extensible
  - Fixed set of codes, no way to store meta data
- Not persistent
  - Generated by compiler from text, never stored

Solution: Reflective Methods

- Annotated, persistent AST
- Bytecode generated on demand and cached



Persephone

- Implementation of Reflective Methods for Squeak Smalltalk
- Smalltalk Compiler generates Reflective Methods
  - Translated to Bytecode on demand
- Open Compiler: Plugins
  - Called before code generation
  - Transform a copy of the AST

Requirements revisited

- Abstraction Level OK
- Causal Connection OK
- Extensibility OK
- Persistency OK
- Size and Performance OK

Reflective Methods: Annotations

- Source visible annotations
  - extended Smalltalk syntax

```
(9 raisedTo: 10000) <:evaluateAtCompiletime:>
```

- Source invisible annotations
  - Reflective API
  - Can reference any object

- Every node can be annotated
- Semantics: Compiler Plugins

Example I: Instrumentation

- Goal: Code Instrumentation
  - Similar to Javassist, but at runtime
  - Insert code before/after, replace
  - Access to runtime data (e.g. receiver of send)

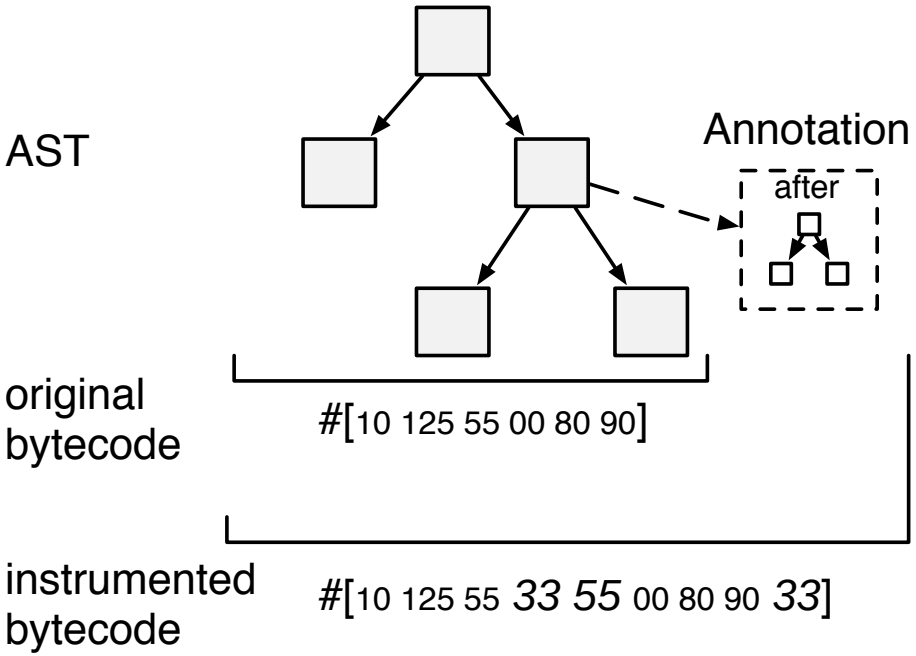
```
a := 1 max: 3
```

Original Code

```
a := 1 max: 3.
AssignmentCounter inc.
```

Instrumented Code

Instrumentation using Annotations



- On demand code generation
  - Faster!
- Better code
  - No preamble code to access data on stack
- Annotations are metadata
  - Original code untouched

Example II: Pluggable Type-System

- Example for textual annotations

```
bitFromBoolean: aBoolean <:type: Boolean :>
^ (aBoolean ifTrue: [1] ifFalse: [0]) <:type: Integer :>
```

- Optional, pluggable type-system
- Types stored as annotations in the Reflective Methods

Performance

Squeak tinyBenchmarks

Caching scheme	Runtime
unmodified Squeak	6.9 seconds
Persephone, no cache	>1 hour
Persephone, cache	6.9 seconds

Memory

	number of classes	*memory*
Squeak 3.9	*2040*	*15.7 MB*
Persephone *no reflective methods*	*2224*	*20 MB*
Persephone *reflective methods*	*2224*	*123 MB*

Future Work

- Optimize Size of AST Representation
  - Simpler AST
  - AST Compression
- Behavioral Reflection
  - Implement Reflex model of partial behavioral reflection
- Beyond Text
  - Store only AST (no text)
  - Build text from annotated AST

Conclusion

- Motivated the need for Reflective Methods
- Implementation: Persephone
- Examples
  - Instrumentation framework
  - Pluggable type-system
- Benchmarks / Memory

Conclusion

- Motivated the need for Reflective Methods
- Implementation: Persephone
- Examples
  - Instrumentation framework
  - Pluggable type-system
- Benchmarks / Memory

Questions?