UNIVERSITÄT BERN

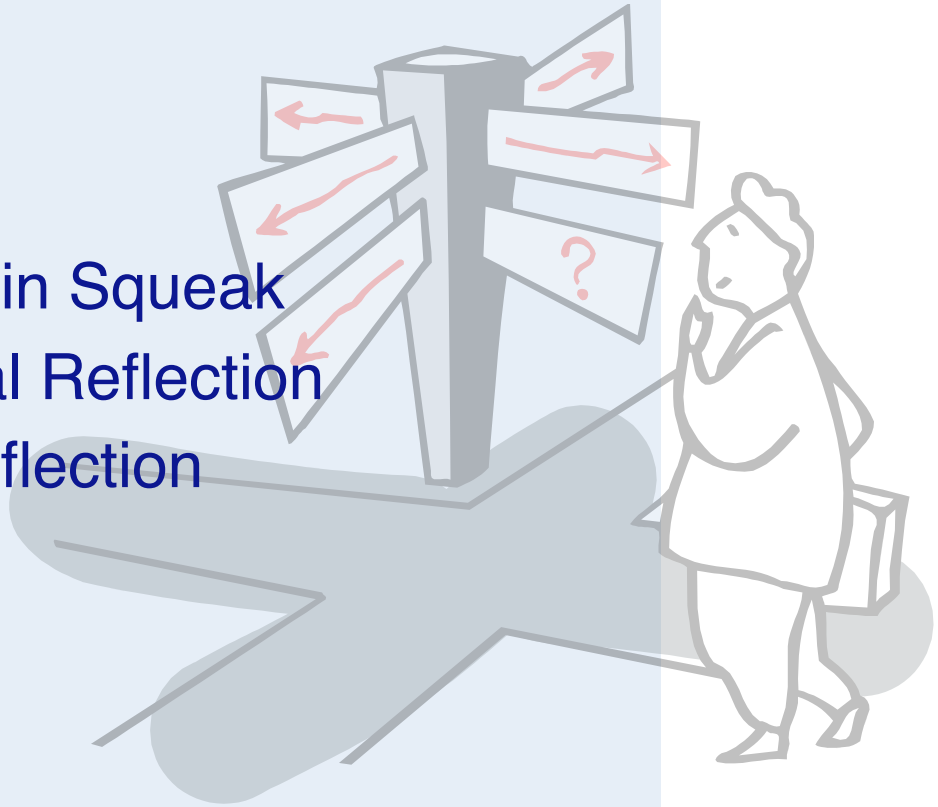
Part II: Reflection

Marcus Denker
denker@iam.unibe.ch

Universität Bern

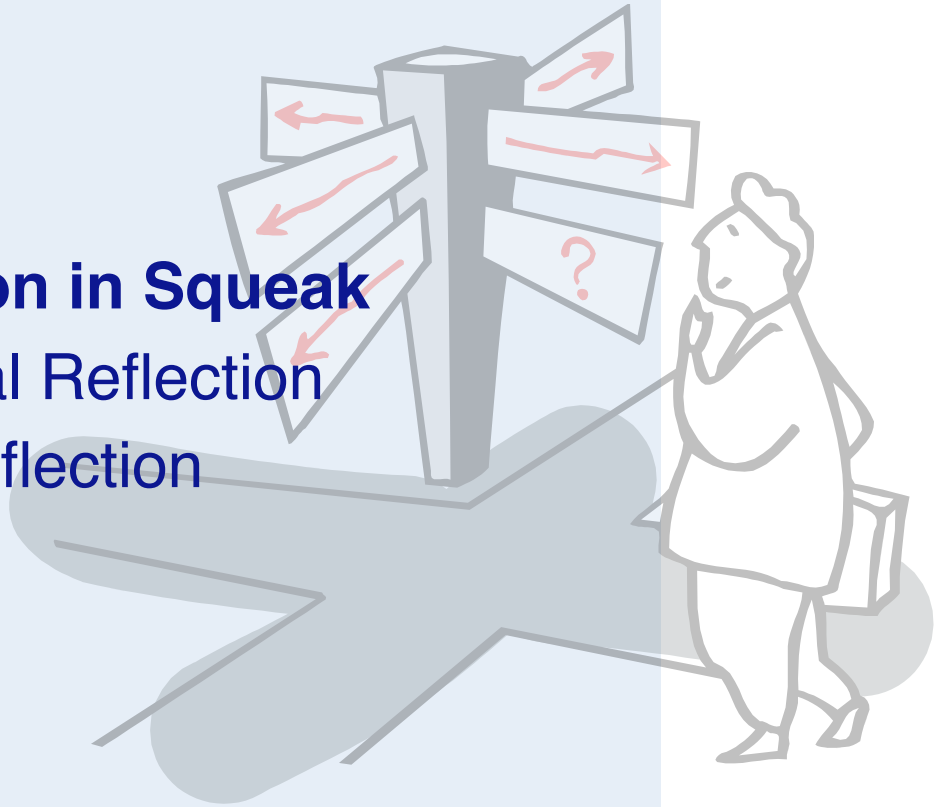
Roadmap

- Introduction: Reflection in Squeak
- I. Sub-Method Structural Reflection
- II. Partial Behavioral Reflection



Roadmap

- **Introduction: Reflection in Squeak**
- I. Sub-Method Structural Reflection
- II. Partial Behavioral Reflection



Reflection

- Reflection: computation about computation
  - Base level / meta level
  - Causally connected
- Structural Reflection
  - Reification of structure
- Behavioral Reflection
  - Reification of execution

Reflection in Squeak

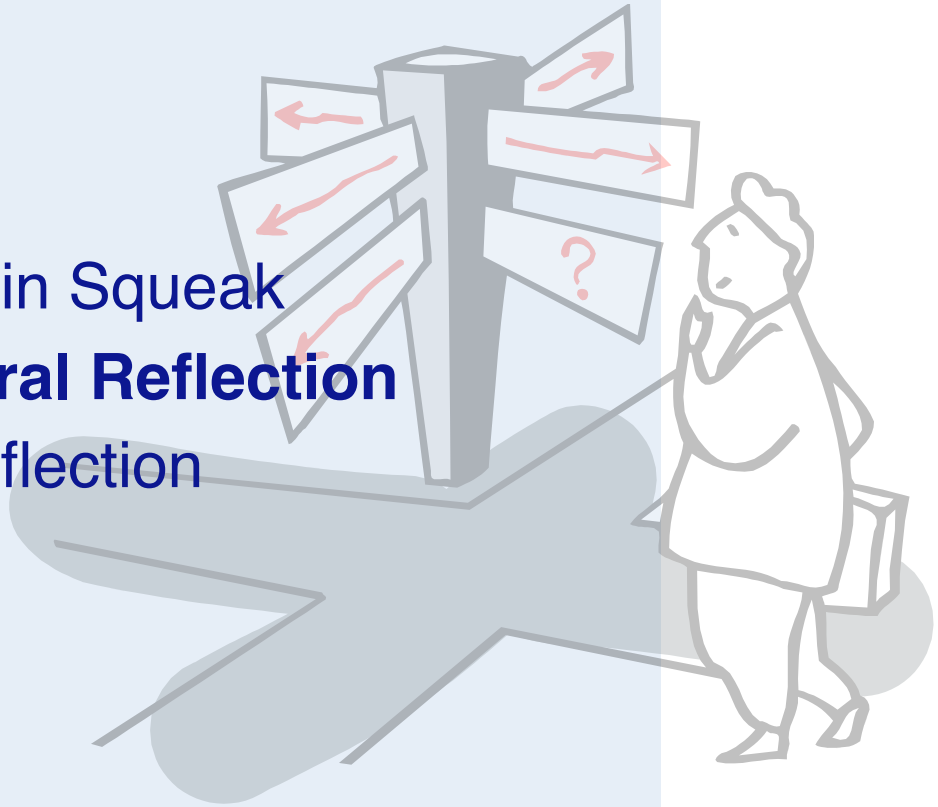
- Squeak has support for reflection
- Structural Reflection
  - Classes / Methods are Objects
  - Can be changed at runtime
- Behavioral Reflection
  - Current execution reified (thisContext)
  - #doesNotUnderstand / MethodWrappers

Can we do better?

- Structural Reflection stops at method level
  - Bytecode in the CompiledMethod: Numbers
  - Text: Just a String, needs to be compiled
- Behavior hard coded in the Virtual Machine
  - Message Sending
  - Variable Access
- Both Structural and Behavioral Reflection is limited in Squeak
  - We should do better!

Roadmap

- Introduction: Reflection in Squeak
- **I. Sub-Method Structural Reflection**
- II. Partial Behavioral Reflection



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

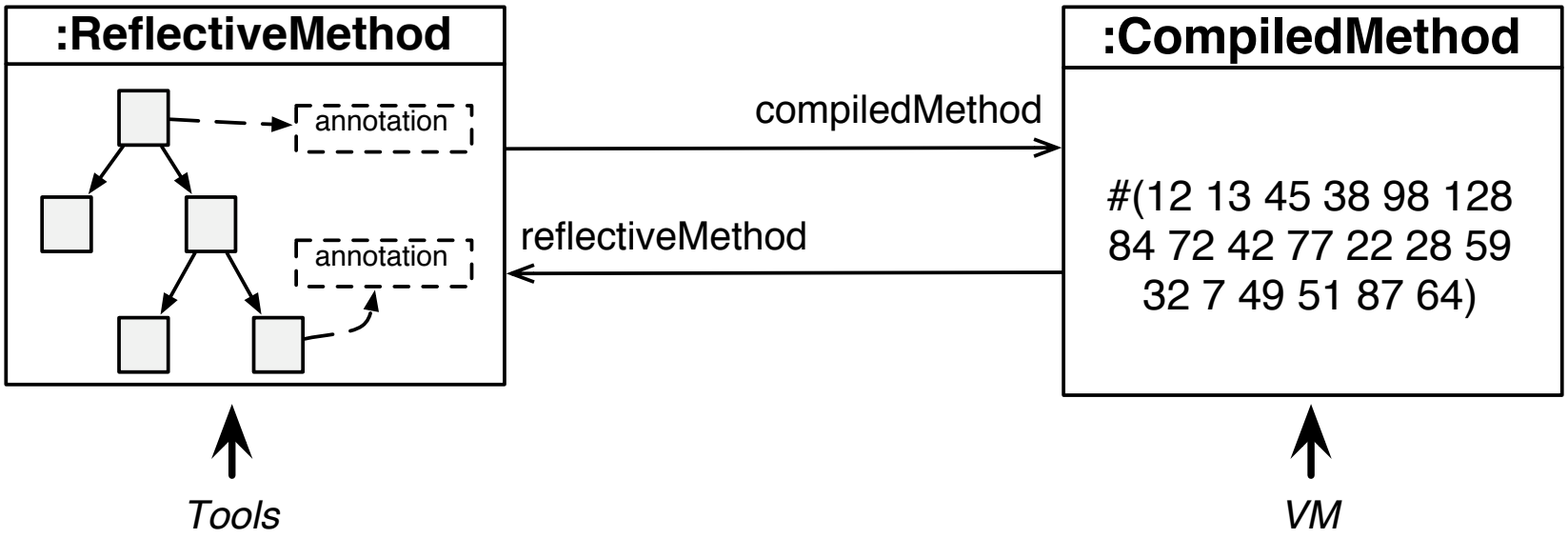
- Low level abstraction
  - Array of Integers
- Missing extensibility
  - e.g. for tools
- Mix of base- and meta-level code
  - Problems with synthesized code when changing code
  - Examples: AOP point-cut residues, reflection hooks

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



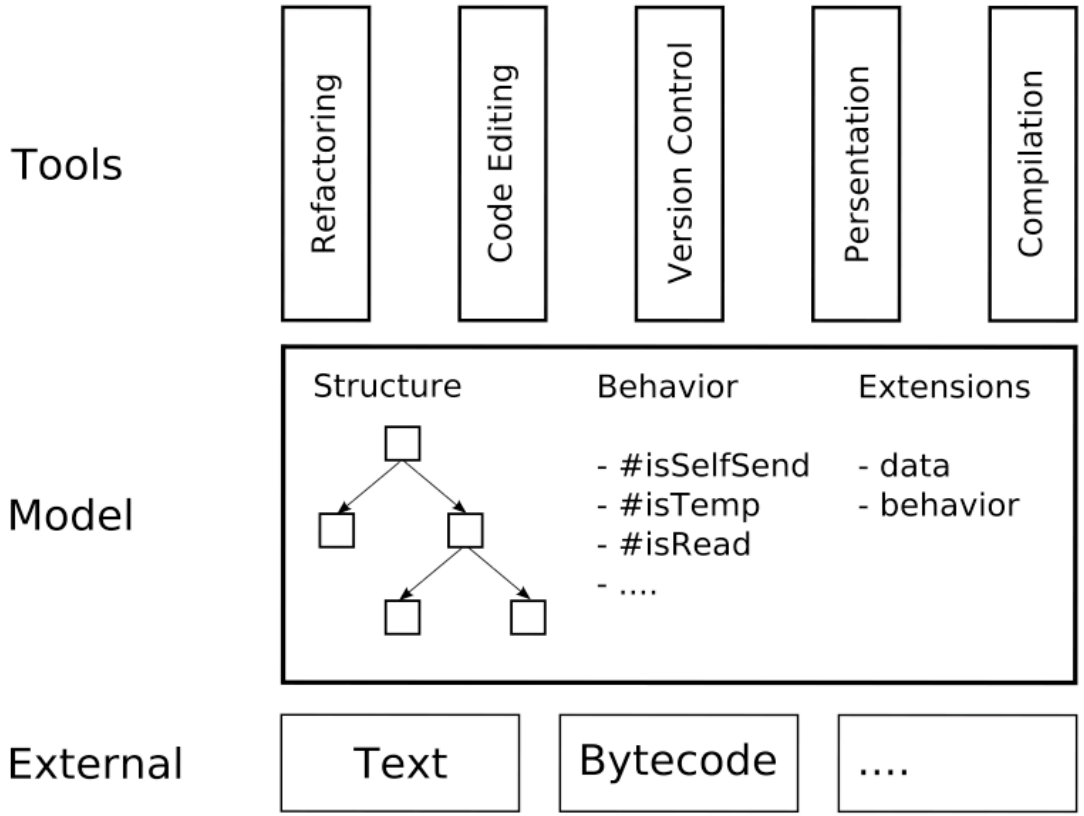
Persephone

- 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

From the last lecture...



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: Pluggable Type-System

- Example for textual annotations

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

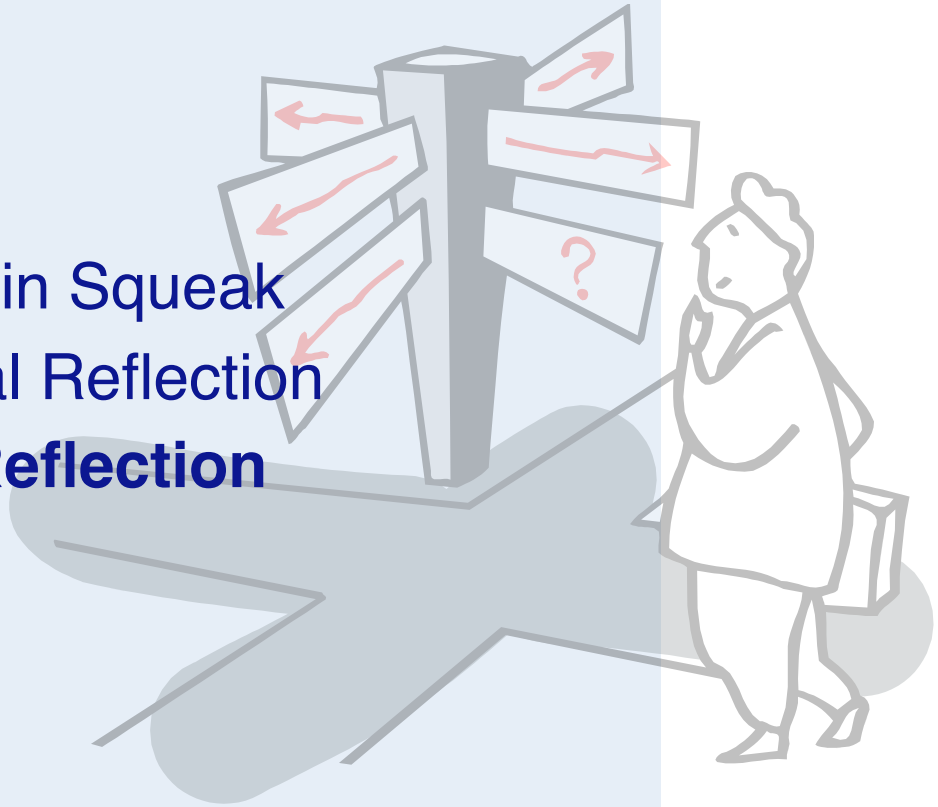
- Optional, pluggable type-system
- Types stored as annotations in the Reflective Methods

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*

Roadmap

- Introduction: Reflection in Squeak
- I. Sub-Method Structural Reflection
- **II. Partial Behavioral Reflection**



Behavioral Reflection

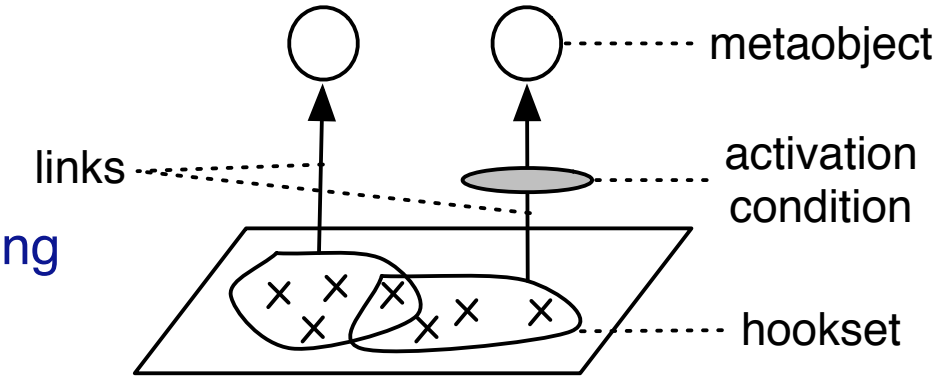
- Reflect on the execution
  - method execution
  - message sending, variable access
- In Smalltalk
  - No model of execution below method body
  - message sending / variable access hard coded by VM
  - #doesNotUnderstand / MethodWrappers
- Reflective capabilities of Smalltalk should be improved!

MetaclassTalk

- Extends the Smalltalk metaclass model
  - Similar to CLOS MOP
- Metaclass defines
  - message lookup
  - access to instance variables
- Problems:
  - Reflection only controllable at class boundaries
  - No fine-grained selection (e.g. single operations)
  - Protocol between base and meta level is fixed

Reflex: Partial Behavioral Reflection

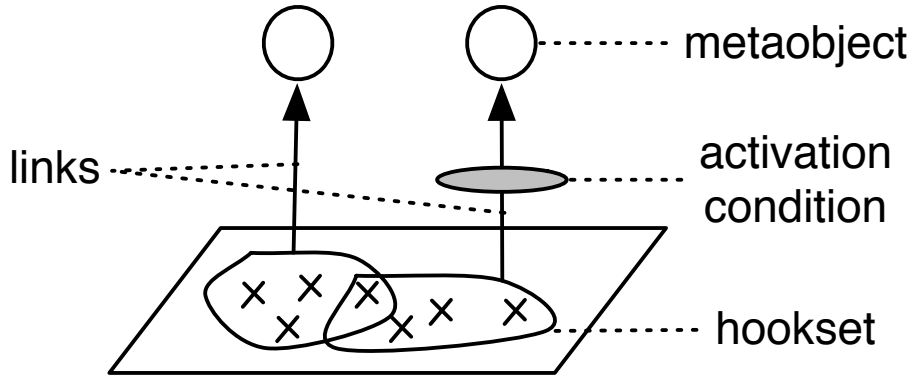
- Hooksets: collection of operation occurrences
- Links
  - Bind hooksets to metaobjects
  - Define Protocol between base and meta
- Goals
  - Highly selective reification
  - Flexiblel metalevel engineering
    - *Protocol specification*
    - *Cross-cutting hooksets*



Tanter, OOPSLA03

Example: Profiler

- Operation:
  - Method Execution (around)
- Hookset:
  - All execution operations in a package
- Metaobject:
  - A profiling tool



Reflex for Squeak

UNIVERSITÄT BERN

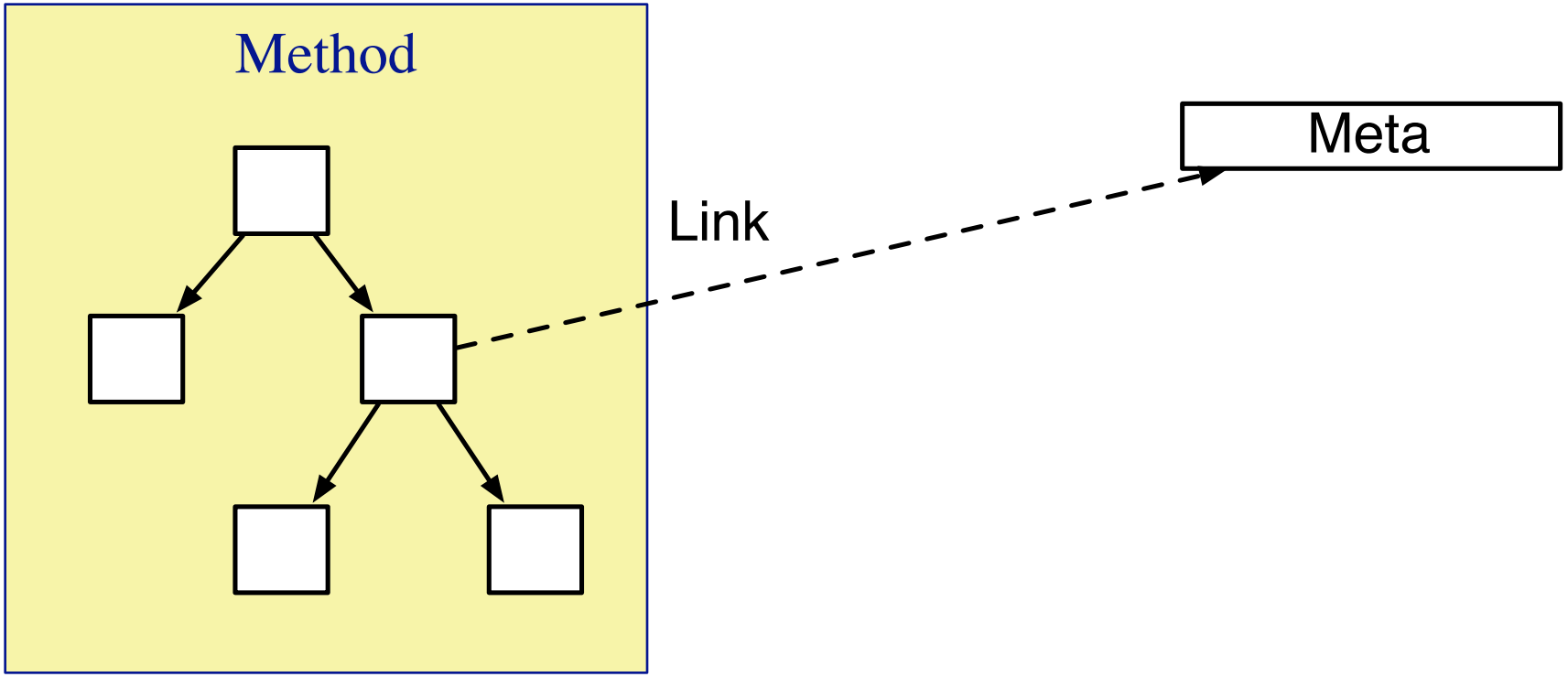
- Partial Behavioral Reflection pioneered in Java
  - Code transformation at load time
  - Not unanticipated (it's Java...)
- Geppetto: Partial Behavioral Reflection for Smalltalk
  - For Squeak 3.9 with Bytecode Transformation

Problems

- Annotation performance
  - Decompile bytecode
- Execution performance
  - Preambles for stack manipulation
- Low-level represenation
  - ifTrue:ifFalse:
  - Blocks
  - Global variables

Links as Annotations

> Links can be annotations on the AST



Properties

- Very fast annotations
  - No decompile!
- On-the-fly code generation
  - Only code executed gets generated
- Generated code is fast
  - Better then working on bytecode level

Reflectivity

- Prototype Implementation in Squeak
  - Sub-Method Structure
  - Partial Behavioral Reflection
- Download:

http:/scg.unibe.ch/Research/Reflectivity

What's next...

- Optimize Size of AST Representation
  - Simpler AST
  - AST Compression
- Implement Context Oriented Programming (COP)

License

UNIVERSITÄT BERN

> http://creativecommons.org/licenses/by-sa/2.5/

creative commons

COMMONS DEED

Attribution-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.