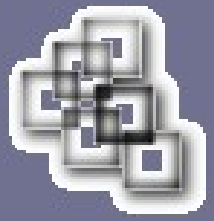
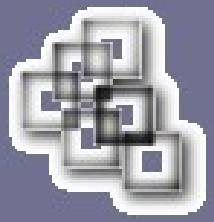


Are Pointcuts a First-Class Language Construct?

Stephan Herrmann
Technische Universität Berlin

stephan@cs.tu-berlin.de

www.ObjectTeams.org



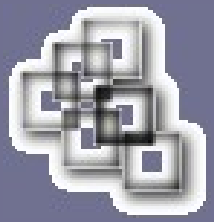
Join Points?

program element

„A join point is a point of interest in some artefact ... through which two or more concerns may be composed“ [Crosscut 1st Issue]

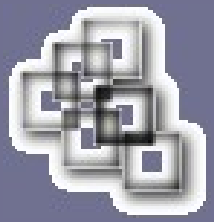
runtime event

„A join point is a point in the execution of a program ...“



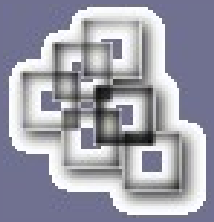
A Calculus for Pointcut Composition?

- **&&, || and !** ≠ **∧, ∨ and ¬**
  - distributive law does not hold in AspectJ
- **event negation?**
  - debatable semantics
- **intersection of join point kinds?**
  - `call(T C.foo()) && set(T C.bar)` ??



Outline

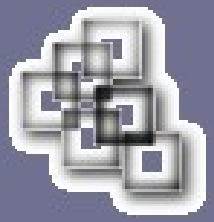
- **Minimal AOP w/o „pointcut“**
  - Bottom-up construction of AOP
    - economy of concepts – „pointcut“ is an expensive concept
  - Terminology of „Join Point Interception“
  - Meta model for join points
- **The Delta**
- **„Pointcuts“**
  - Reverse methods
  - Model: pointcuts as classes
  - Compositionality for free



Minimal AOP

- **Join points**
  - **elements** of the program, defined by meta model
- **Join point queries**
  - matching (wildcards etc.) ∨ functional **queries**
    - kind & scope & constrain
- ~~advice~~
- **Join point interception**
  - **binding**: aspect method ← set of join points
    - before | after | replace
    - possibly **guarded** (run-time filter)
    - overridable (needs a name)
  - execution of join point may trigger aspect method

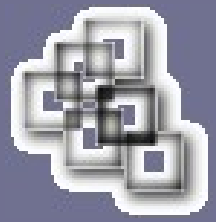
E(C)/A



Discussion

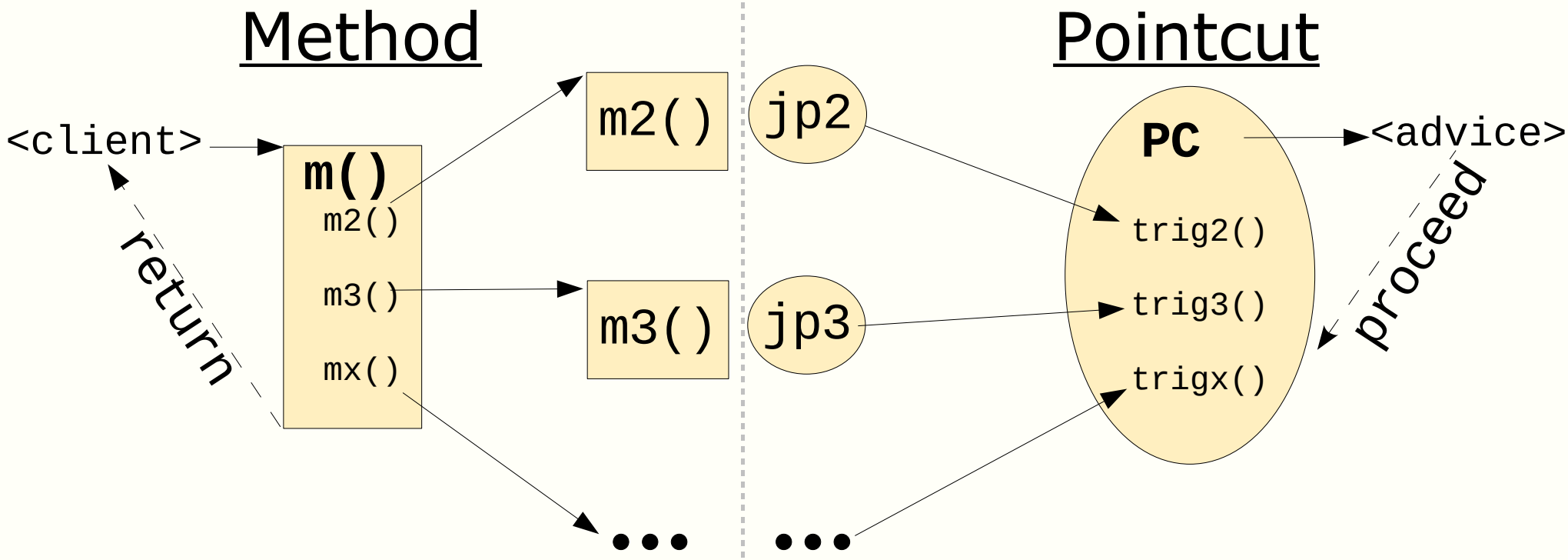
- **Powerful AOP without**
  „points in the execution of a program“
  - amenable to formal, static analysis
  - students can implement/understand the language
- **What is missing?**
  - Regarding AspectJ:
    - `cflow`
  - Other dynamic approaches
    - stateful aspects
    - trace matches
    - ...
  - What do these have in common?

consume multiple events to trigger one action



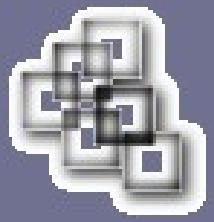
Reverse Methods

- **Definition by Anti-Symmetry:**



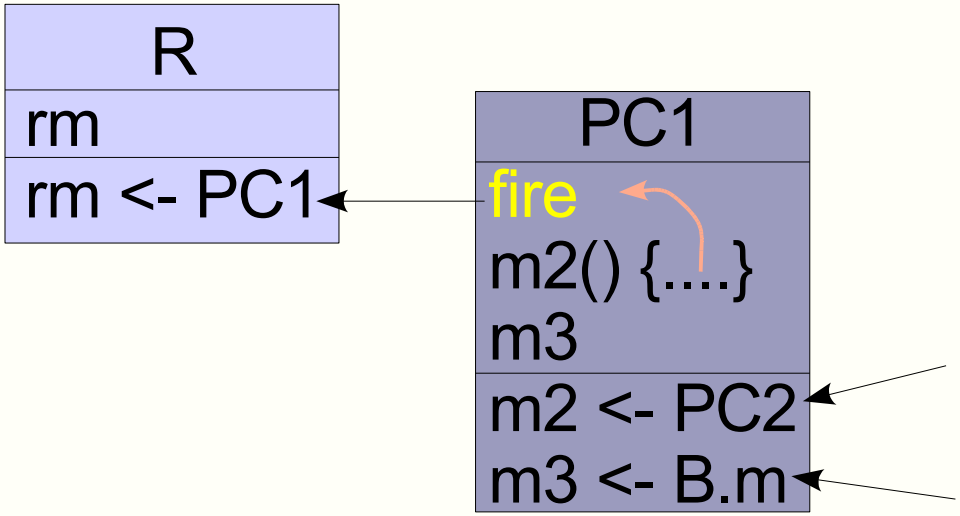
***consume* one *incoming* call event**
***produce* sequence of *outgoing* events**

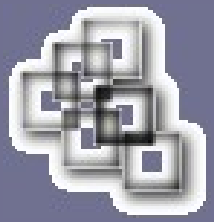
***produce* one *outgoing* call event**
***consume* sequence of *incoming* events**



Pointcut Class

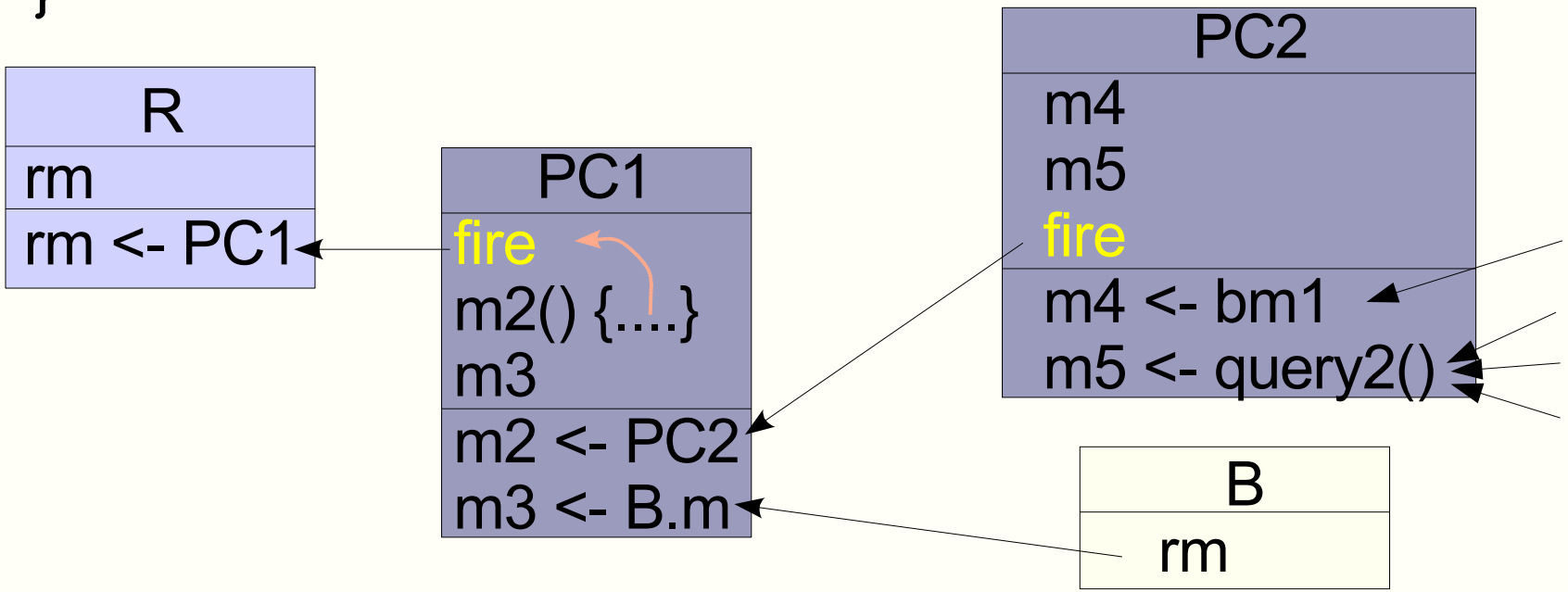
- **Why invent something new?**
  - ```
    public team class T {
        protected class R playedBy ? {
            void rm() { ... }
            rm <- replace PC1.fire;
        }
    }
    ```

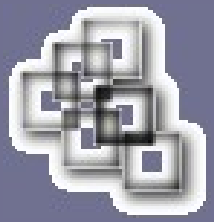

    ```



Compositional Pointcut Binding

- **Why invent something new?**
  - ```
    public team class T {
        protected class R playedBy ? {
            void rm() { ... }
            rm <- replace PC1.fire;
        }
    }
    ```


    ```



Economy

- **Join point interception**
  - a low-cost concept
  - statically determined

more details at:
www.objectteams.org/publications

- **Multi-event triggers**
  - generalized/simulated by class
  - specialized syntax deferred
- **Calculi**
  - join points: functional queries (meta model + set theory)
  - aspect binding: E(C)/A + overriding
  - composition as aspects-of-aspects