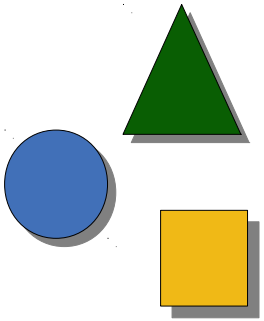
25. Declarative Aspect Weaving with Cross-Cut Graphs and Graph Rewriting

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 13-1.0, 2013-06-29

1) Introduction
   1) Aspect Oriented Development (AOSD) and -Programming (AOP)
   2) Graph Rewrite Systems (GRS)
2) Categories of GRS-based Weaving
3) Generation of Aspect Weavers
4) Conclusion



Optional

Literature

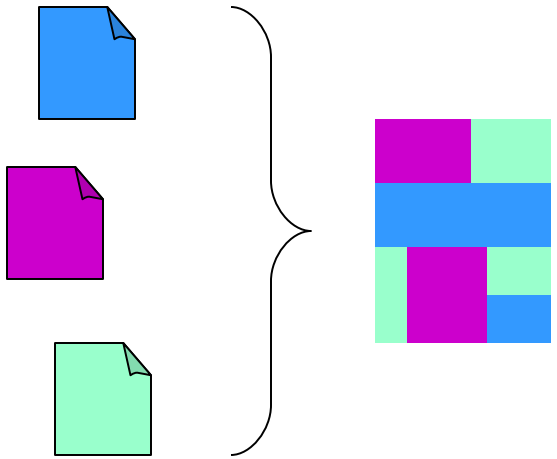
- Uwe Aßmann and Andreas Ludwig. Introducing Connections into Classes with Static Metaprogramming. In Paolo Ciancarini and Alexander Wolf, editors, 3rd Int. Conf. on Coordination, volume 1594 of Lecture Notes in Computer Science, pages 371-383. Springer, Heidelberg, April 1999.
  - google for it.

25.1 Aspect-Oriented Development

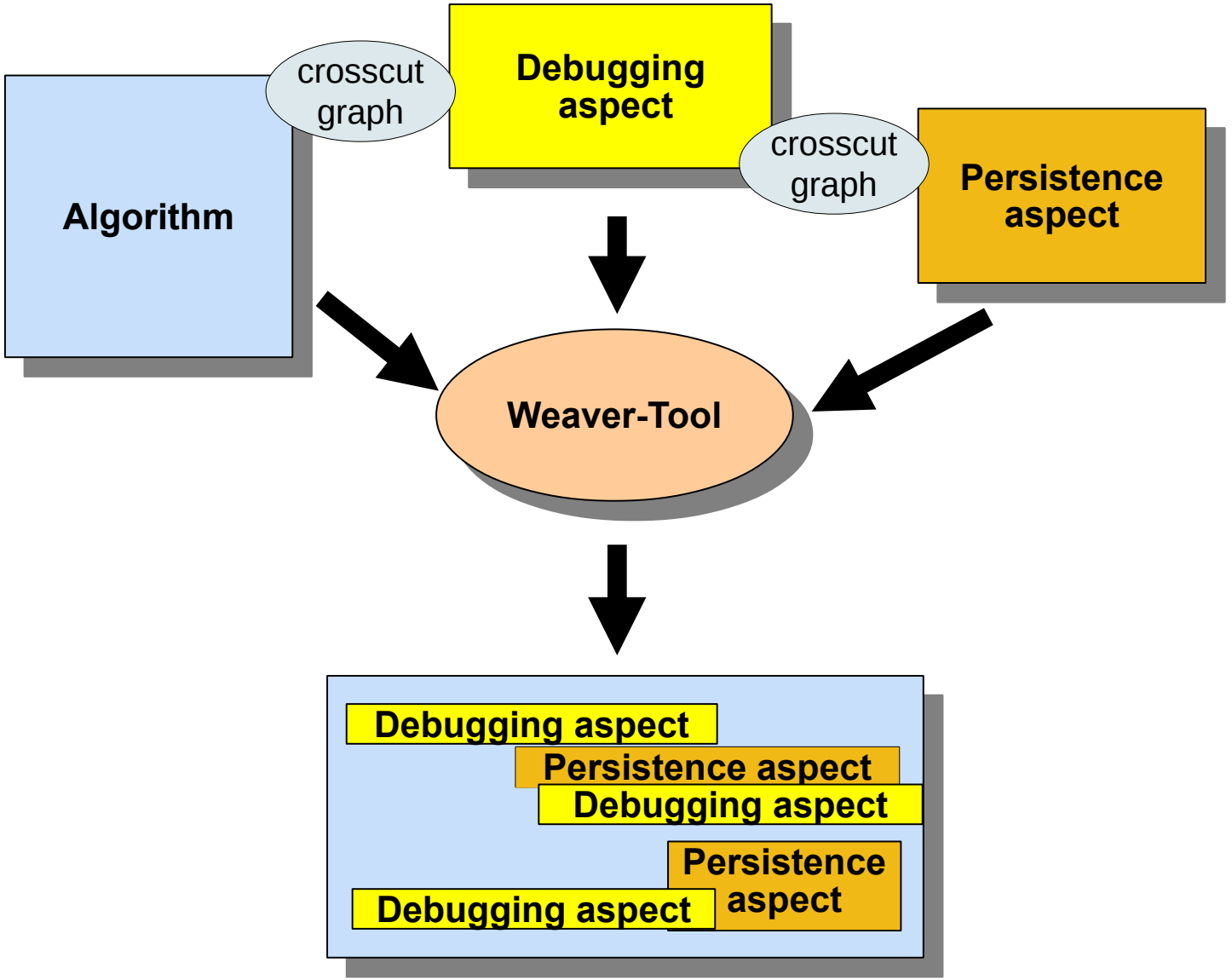
- Motivation: Separation of Concerns
  - a new kind of modularization
  - separation of cross-cutting code parts
- Technique: Integration by (Static) Weaving
  - Based on Crosscut Graphs

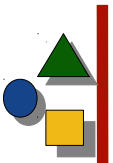
Examples for Aspects:

- Synchronization
- Communication
- Instrumentalization
- Memory Management



Aspect-Oriented Development





Motivation

AOSD/AOP aims at different problem domains...

.. weaving requires different specification languages.

A new weaver for every weave scenario!

Weavers are compilers... Weaving can become complicated...

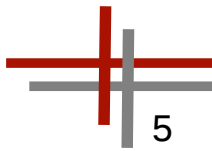
We need a uniform and formal technique to

classify and specify AOP weavers.

Idea: Programs and models can be represented as typed graphs (abstract syntax graphs)...

Describe aspect weaving as cross-cut graphs

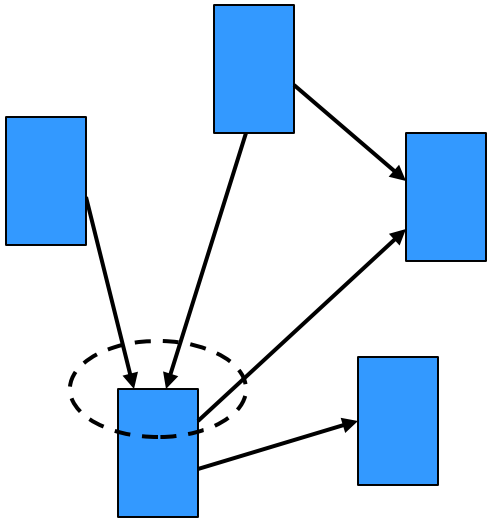
Produce the cross-cut graphs by declarative graph-rewriting

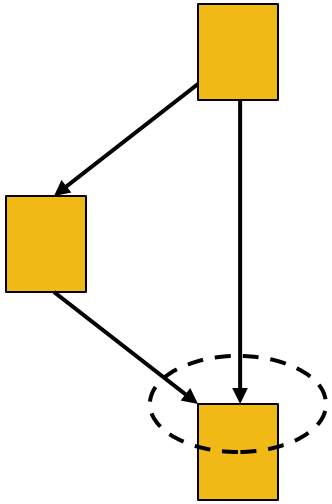


Classes of AOP Systems

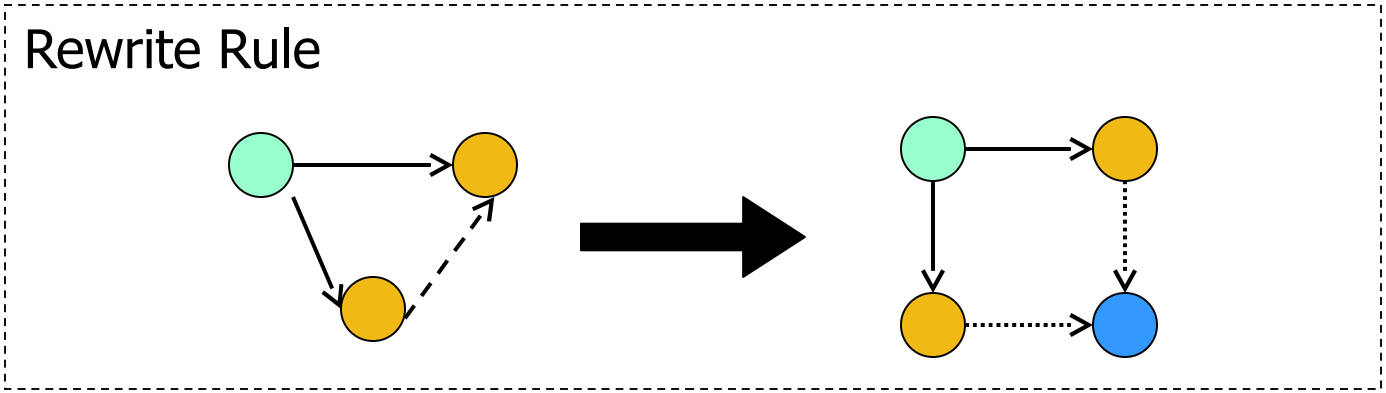
- Script-based AOP (e.g. *RG, AspectJ, InjectJ*)
  - aspects are modification rules
- Language-based AOP (e.g. *D, AML*)
  - aspects are specialized languages
- Declarative AOP
  - crosscut graphs are described by a declarative language
  - e.g., logic-based AOP
- Graph-rewriting-based AOP
  - rewriting rules combine aspect fragments

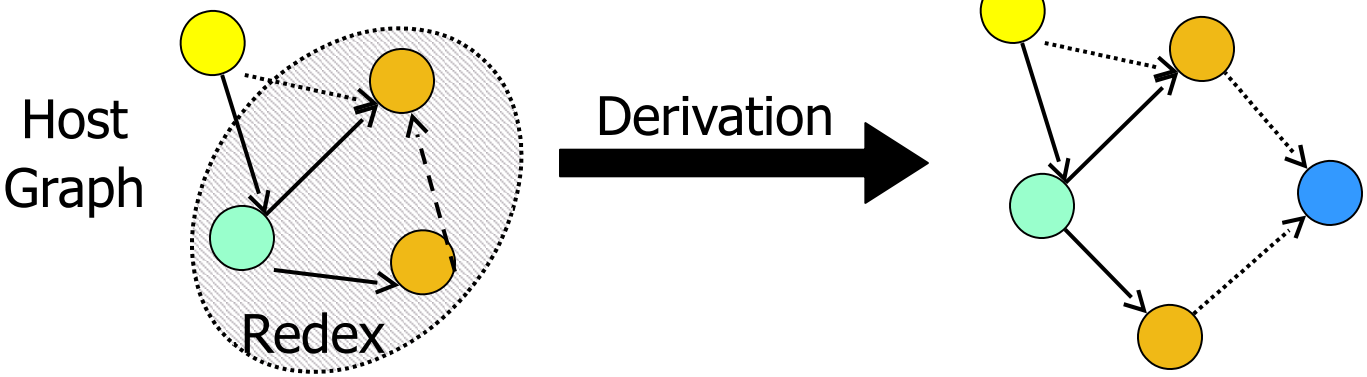
Core + Aspect Graphs





GRS - Basics





25.2 AOP as GRS

Program Graph: Representation of the program (or of a model) in form of a graph

- *Core Graph:* Program graph of the core
- *Aspect Graph:* Program graph of the aspect

Join Point: Redex in core graph

Aspect Fragment: Redex in Aspect Graph

Aspect Composer: Graph Rewrite Rule

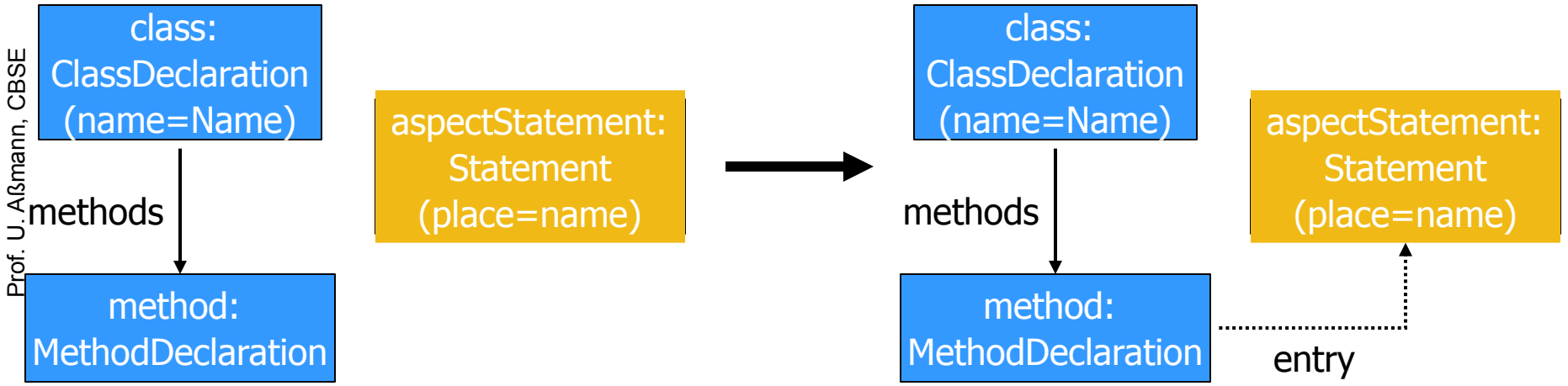
Weave Operation: Direct Derivation

Weaver: Graph Rewrite System with

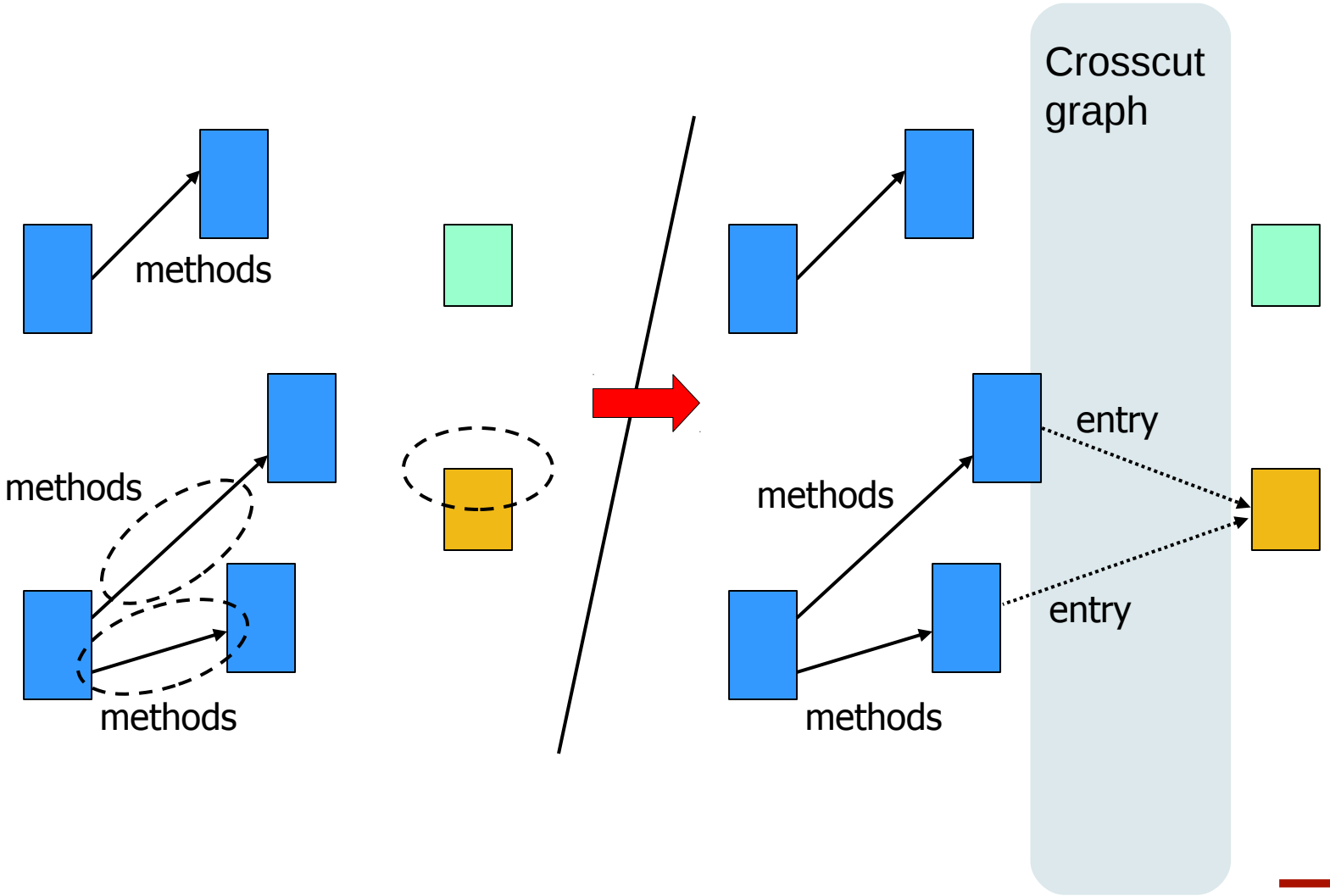
- a set of aspect composers,
- a component graph, and
- a set of aspect graphs (context-sensitive rules).

Example

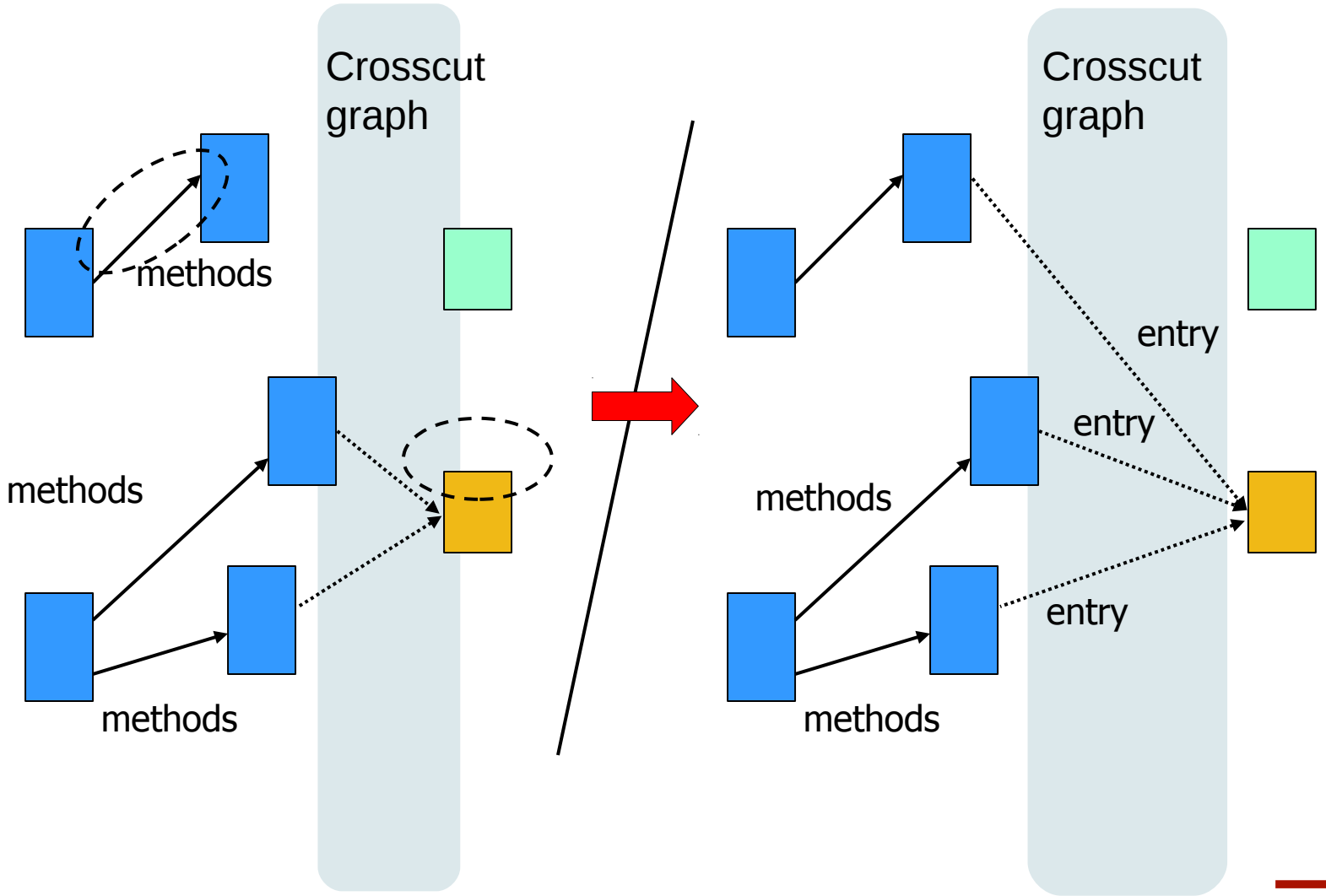
GRS rule to prepend statements to method entries



Example (2): Working with the Rule on a ProgramGraph



Example (3)

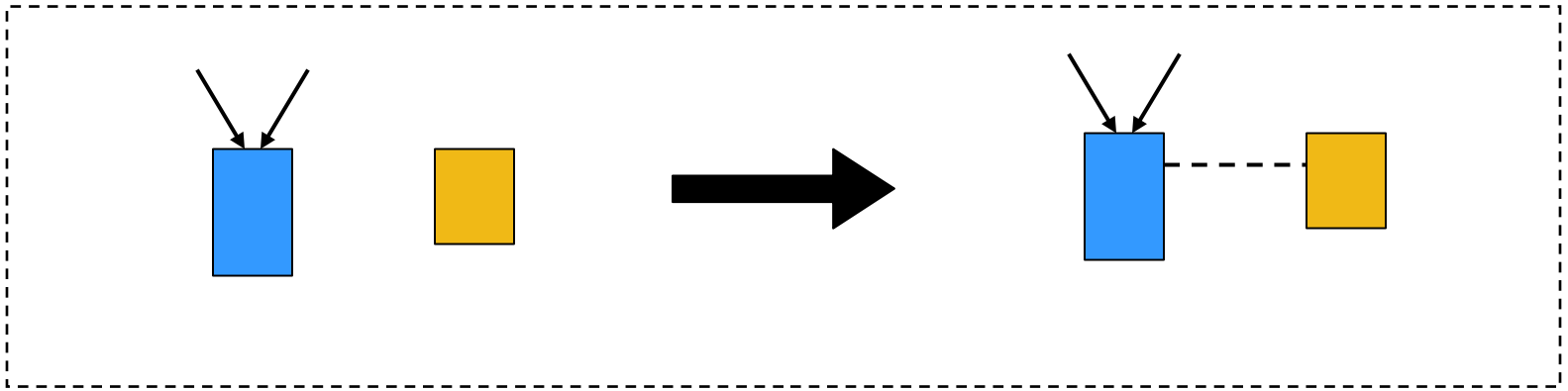

Crosscut graph
methods
methods
methods
Crosscut graph
entry
entry
entry
methods
methods

Benefits

- Handling of all kinds of aspects possible
  - all we need is abstract syntax of programs or models
- Universally for all programming and modeling languages
- Uniform specification allows a classification of aspect weaving systems
- Certain classes of rewrite systems guarantee
  - **termination**
  - **confluence** (= deterministic results)

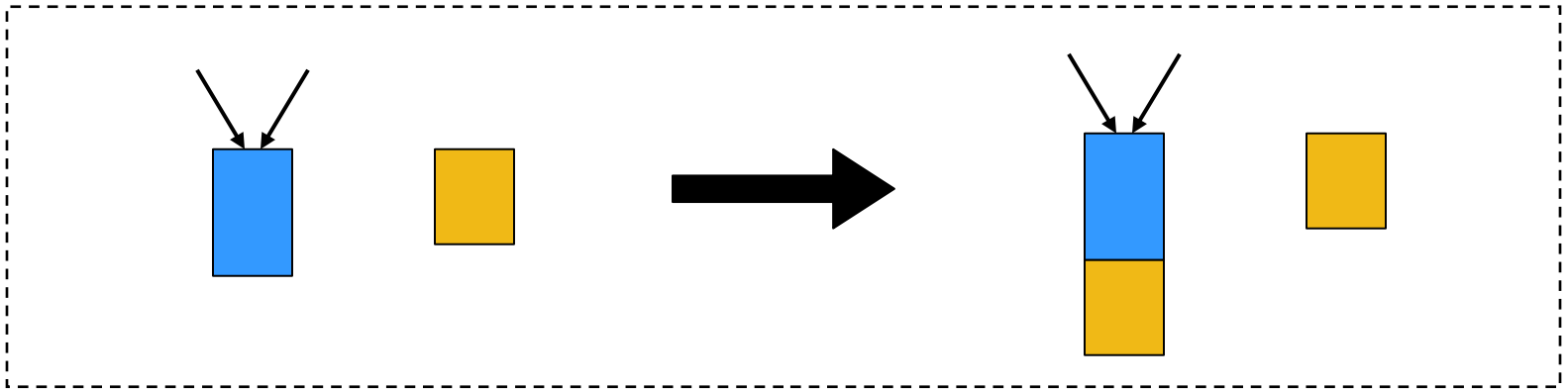
Category I: Aspect-Relating Rules

- Edge-addition rewrite system (EARS)
  - always congruent (= terminating + confluent)
  - weaving operation becomes a function
- Ideal for simple property aspects
  - e.g. persistency, synchronization, ...



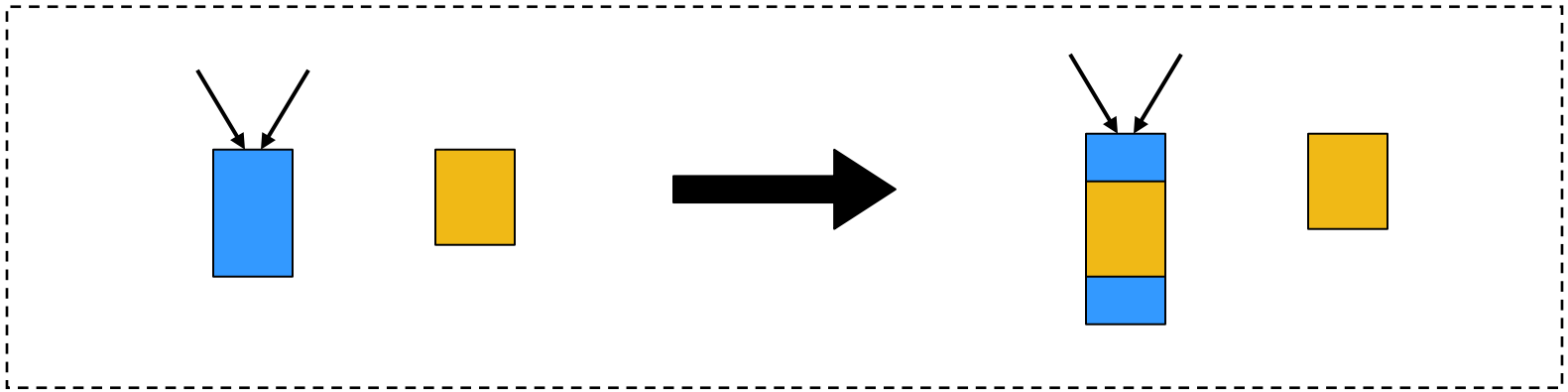
Category II: Aspect-Additive Rules

- if it is an eXhaustive Graph Rewrite System (XGRS) and does not modify the redex
  - always congruent (= terminating + confluent)
- Ideal for orthogonal aspect code
  - e.g. Adaptive Programming
-



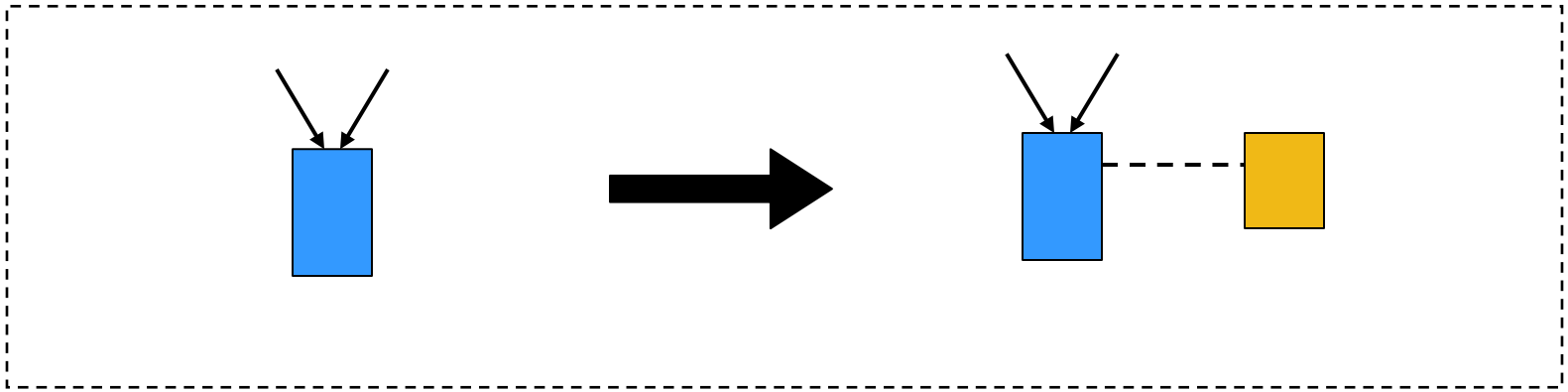
Category III: Core-Modifying Rules

- Exchange parts of cores.
- Confluence and termination are not guaranteed.
- Indeterminism is acceptable if all normal forms are semantically equivalent.



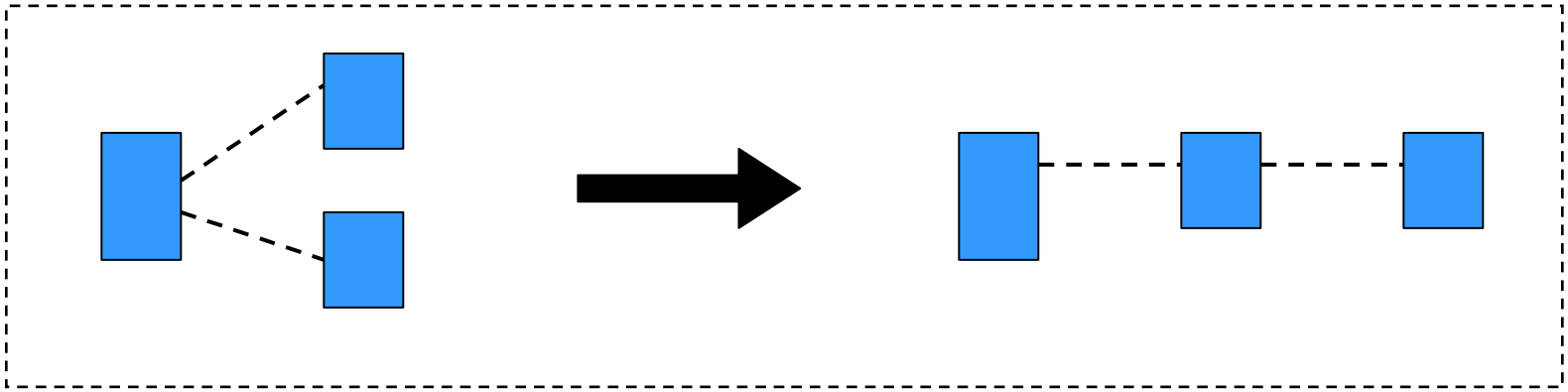
Special Category: Aspects in Rules

- Aspect fragments are part of the right-hand sides.
- Similar to script-based AOP.
- Ideal for aspects with finite variability (because of finite set of rules).



Special Category: Core-Modifying Rules

- Intra-core rules
  - rewrite the component graph only
  - resemble standard code motion optimizations
- Ideal for optimizing aspect weavers.
  - e.g. RG (Reverse Graphics of Xerox)



Comparison

System	terminating	deterministic	aspect graph
Aspect Fragment Matching			
Aspect-relating	yes	yes	yes
Aspect-additive	if exhaustive	yes	yes
Core-modifying	if exhaustive	usually not	yes
Aspects in Rules	depends	depends	no
Intra-Core	depends	depends	no

Aspect Weaving is similar to Program Optimization

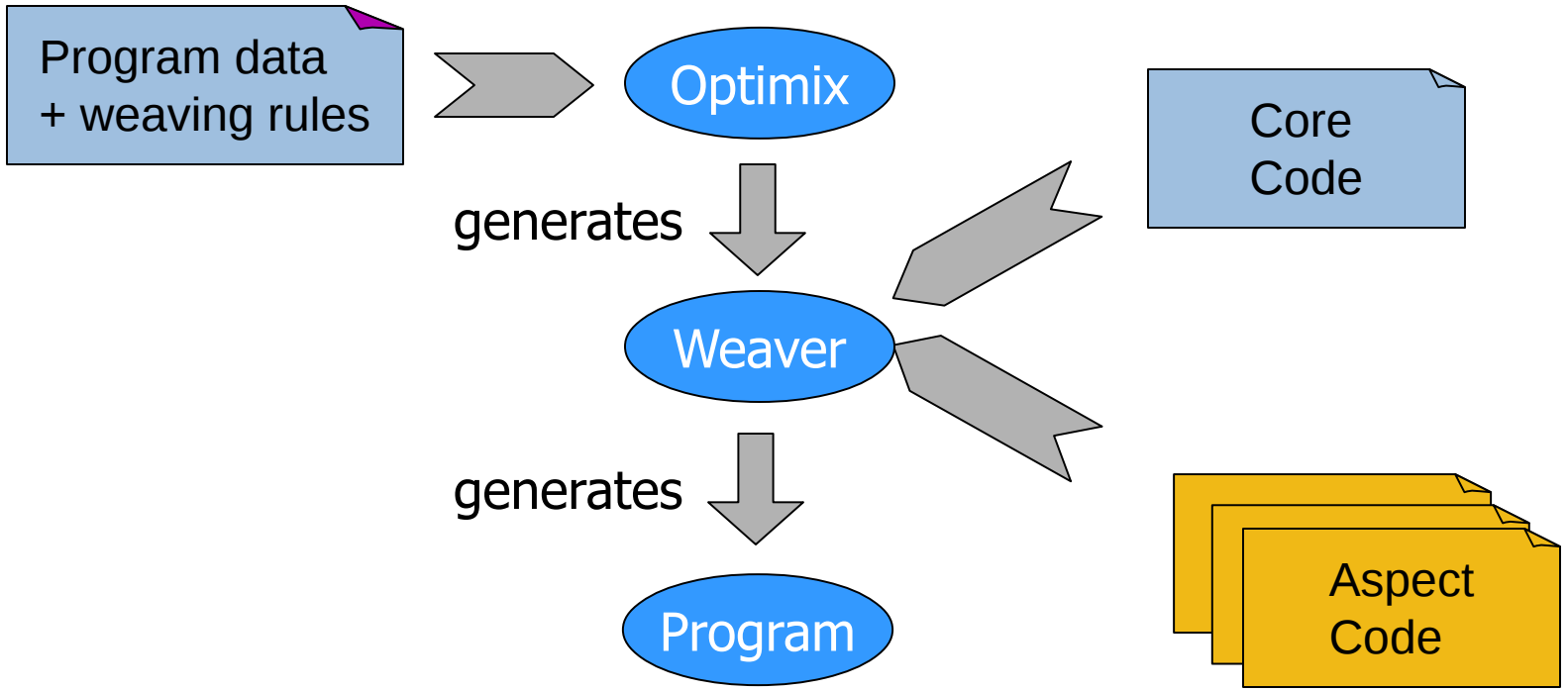
- Graph rewriting can express many program optimizations uniformly [Aßm96].
- Optimizations transform programs.
- Weavers transform programs.

So:

Graph rewriting can express many aspect weavings uniformly.

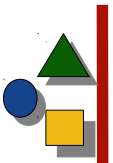
Generating Tools from Rewrite Specification

- [Alexander Christoph, PhD 2004, University of Karlsruhe]



Conclusion

- GRS provide a **uniform** and **formal** way to specify and classify aspect weavings.
- Tool support for weavers.
- Open question:
  - How much of AOP can be covered by this approach?
- Alternative approaches:
  - Prolog based pointcut specifications
  - Query-based pointcut specifications



The End

- Several slides are courtesy to Dr. Andreas Ludwig.