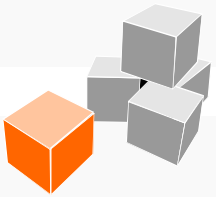
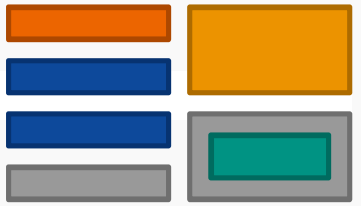
Twente Research and Education on Software Engineering, Universiteit Twente



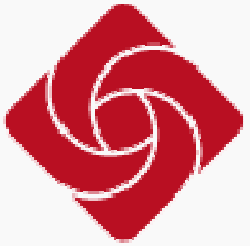
Software Technology Group
TU Darmstadt | FB Informatik



www.alia4j.org

ALIA4J's [(Just-In-Time) Compile-Time] MOP for Advanced Dispatching

Christoph Bockisch[1], Andreas Sewe[2], Martin Zandberg[1]
[1] Universiteit Twente, The Netherlands
[2] Technische Universität Darmstadt, Germany



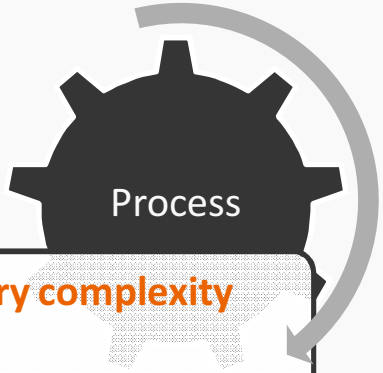
CASED

Motivation and Goal

- Much research in programming languages
  - Increase modularity
  - Variations of late-binding ("**advanced dispatching**")
- Language creation involves **two roles**

Language **designer** → Language **implementer**

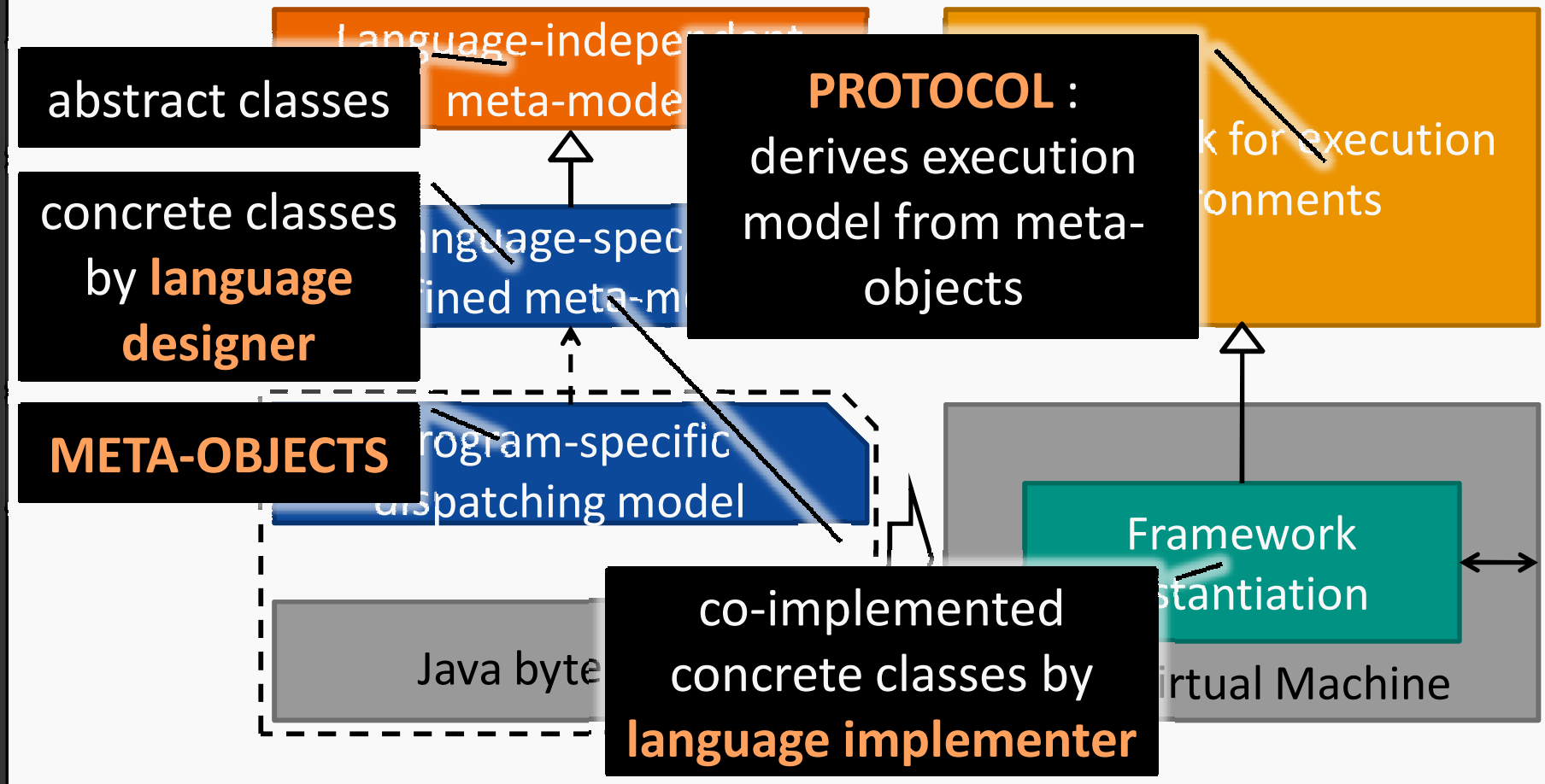
- High-level semantics
- Prototype based on **interpretation**
- Semantic interactions of language concepts



- Low-level implementation
- Product based on **compilation**
- Indentify optimizable special cases

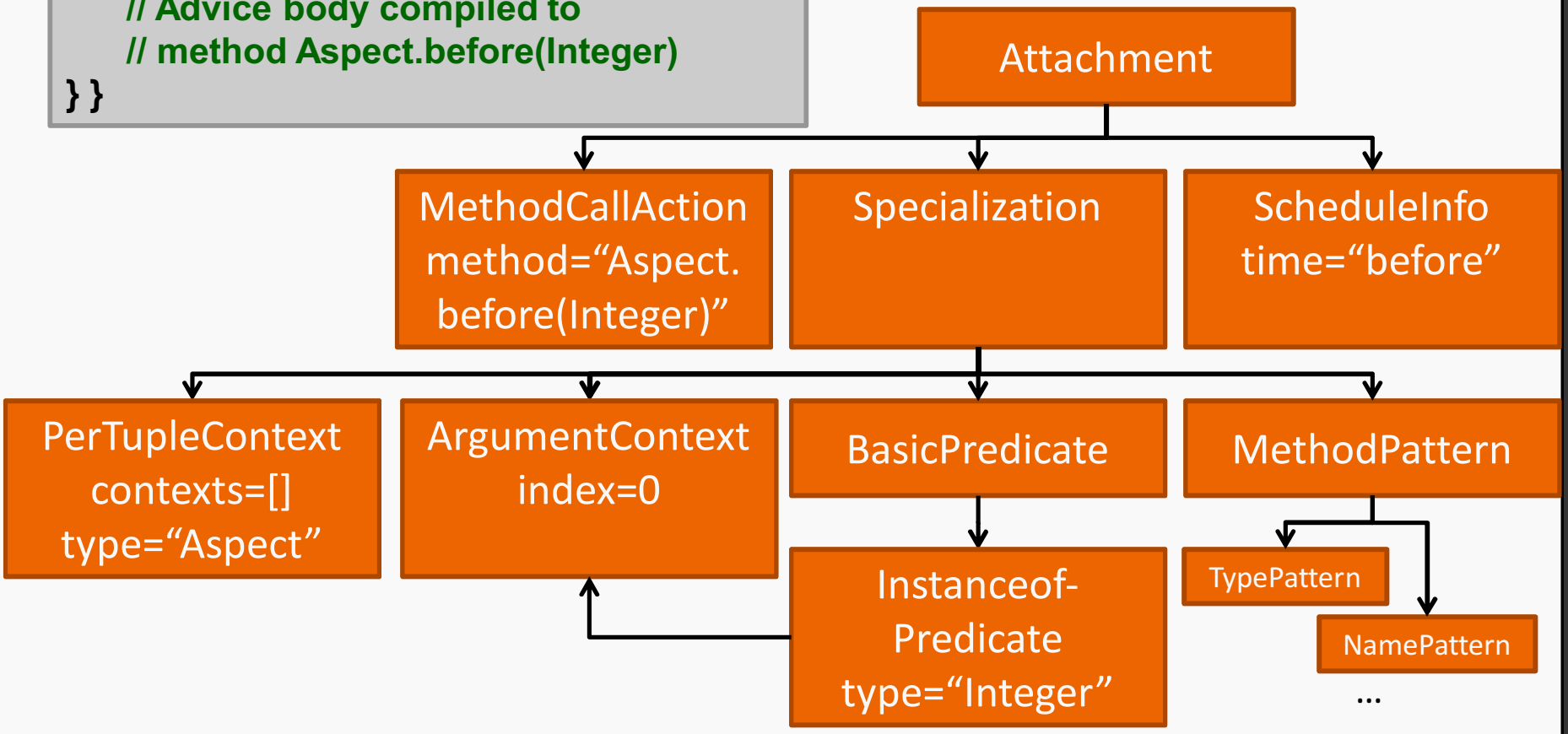
- **no unnecessary complexity**
- **correctness**
- **reusability**

Advanced-Dispatching Language-Implementation Architecture for Java (ALIA4J)

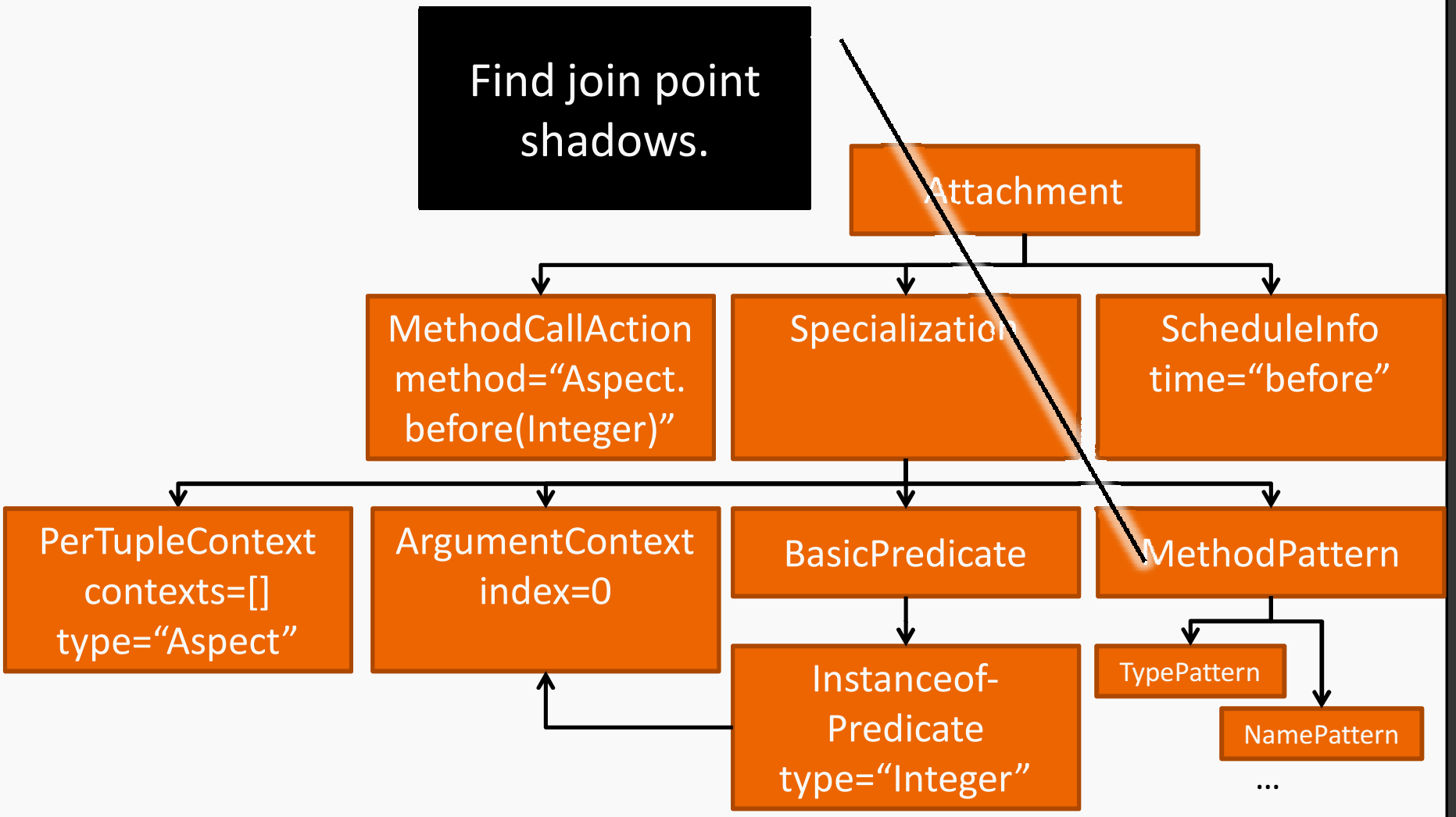


Meta-Objects by example

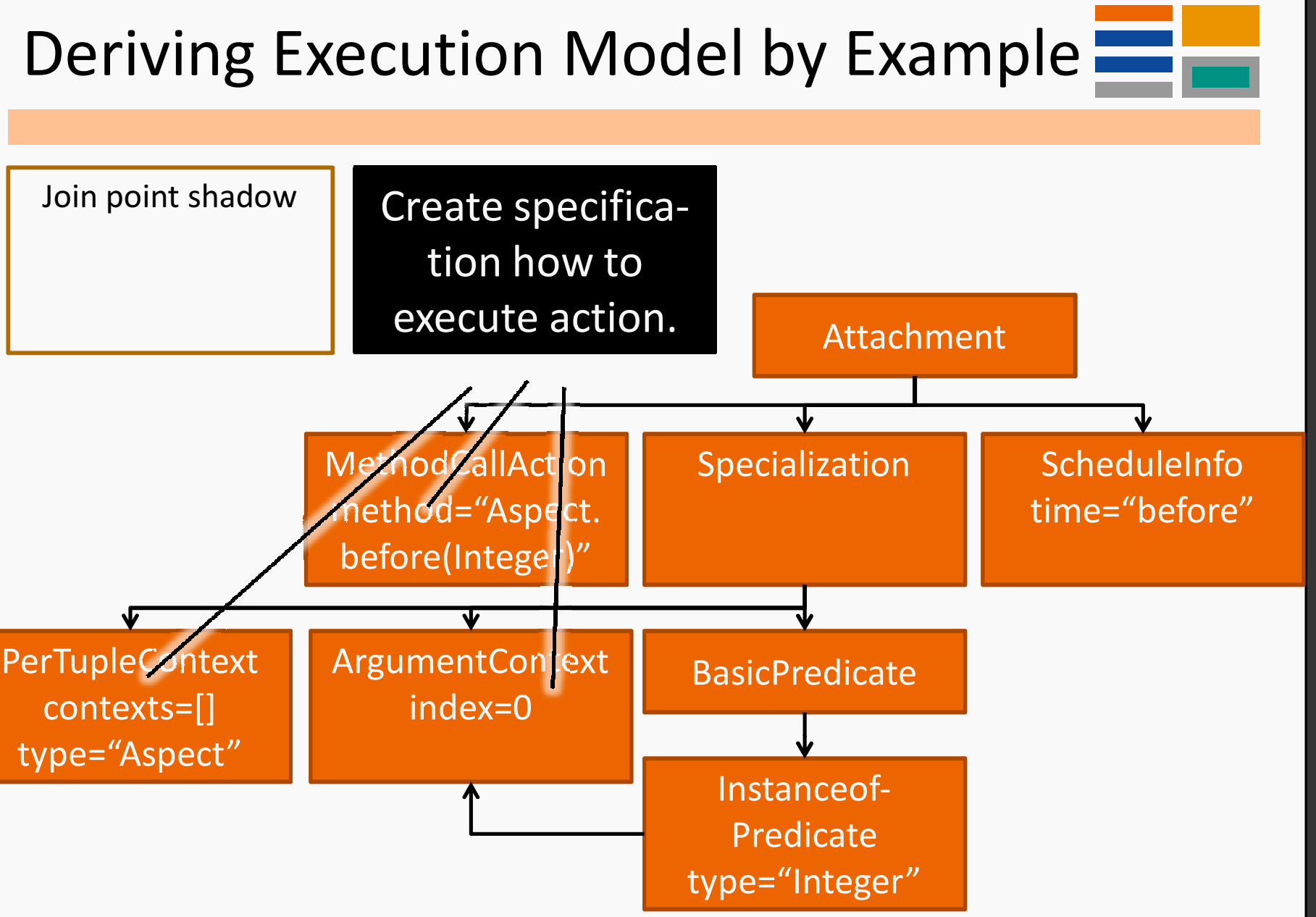
```
public aspect Aspect issingleton() {
  before(Integer i) :
    call(void Main.m(Number)) && args(i) {
      // Advice body compiled to
      // method Aspect.before(Integer)
} }
```



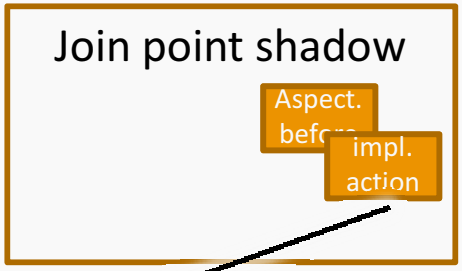
Deriving Execution Model by Example



Deriving Execution Model by Example

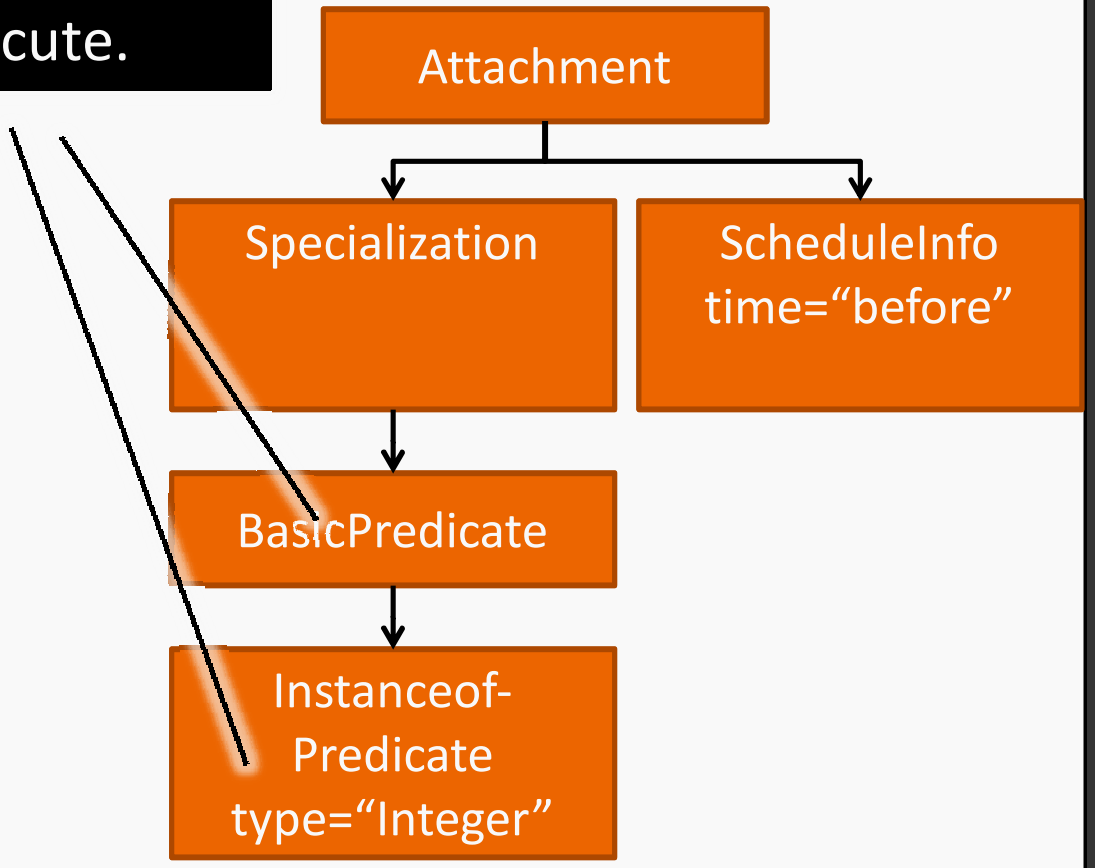


Deriving Execution Model by Example

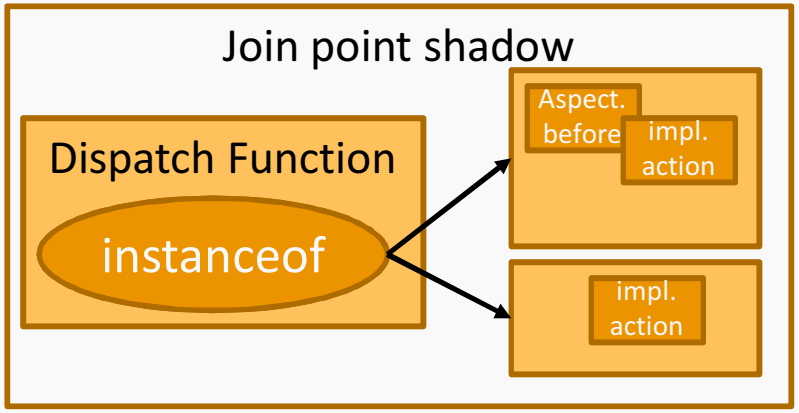


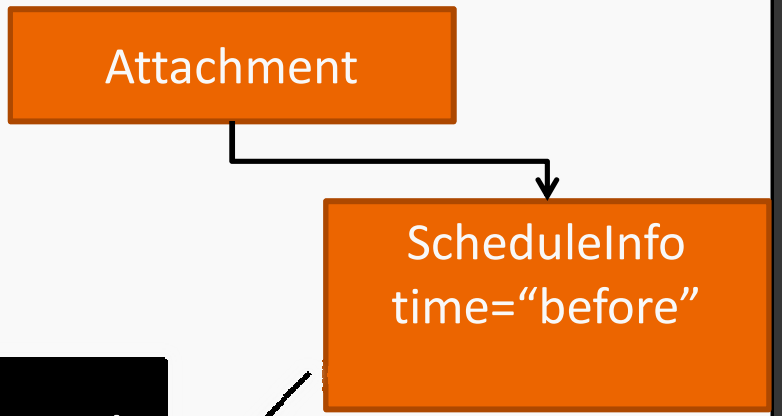
Dynamically select actions to execute.

Every join point has implicit attachment of called Java method.



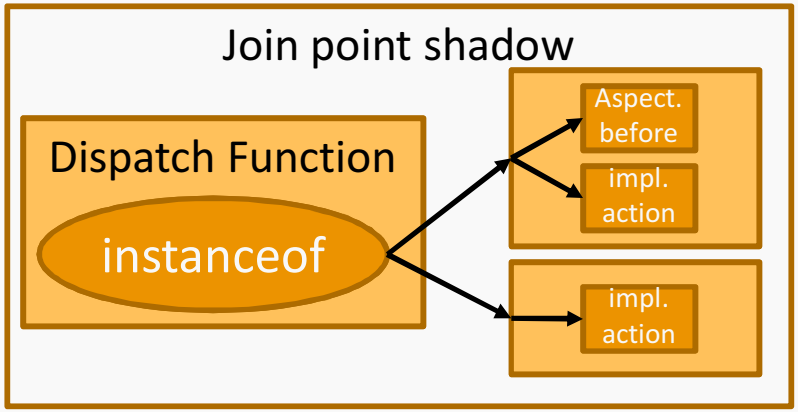
Deriving Execution Model by Example





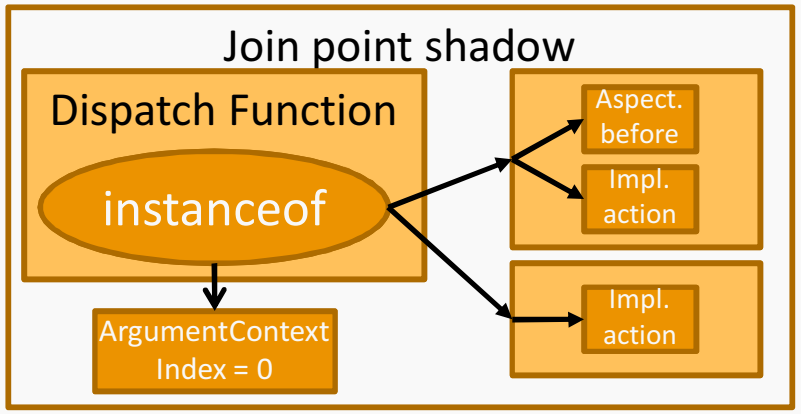
Order actions and consider other constraints.

Deriving Execution Model by Example



Meta-Object Protocol by Example

```
public class Main {
  public static void main(String[] args) {
    new Main().m(new Integer(1));
  }
  public void m(Number n) {
    ...
} }
```



1. Find corresponding execution model
2. Perform semantics of **ArgumentContext**
3. Perform semantics of **InstanceofPredicate**, **passing result** of step 2
4. Execute actions
   in top box, if step 3 returns **`true`**
   in bottom box, otherwise
   a. Evaluate exposed context values and execute actions

ALIA4J Execution Environments

- **Generic work flows** implemented in FIAL
  - Handle dynamic class loading and deployment
  - Derive execution model
  - Optimize execution model
  - …
- Concrete execution environments realize execution model by
  - **Interpretation** (Steamloom^{ALIA}, SiRIn, NOIRIn)
  - **Bytecode generation** (Steamloom^{ALIA}, SiRIn)
  - **Machine code generation** (Steamloom^{ALIA})

Strengths of MOPs

- Plain MOP
  - **No knowledge of execution environment**
  - Implemented in plain Java
- Compile-time MOP
  - Avoid indirection
  - **Access to compilation context**
  - Deferred code generation: can **consider runtime state**
- Just-in-time (JIT) compile-time MOP
  - As compile-time MOP
  - More **precise compilation context**
  - Access to **low-level operations**
    Direct memory access, runtime services, etc.

Serves language implementers as specification and test oracle.

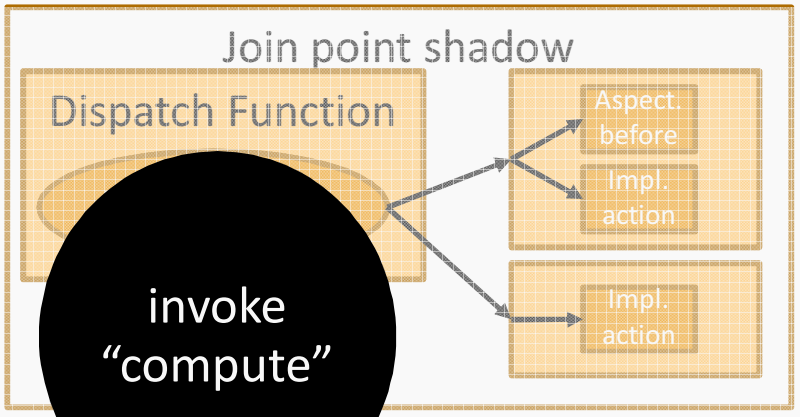
Implementing Semantics/Optimizations

```
public class PerTupleContext extends Context implements … {
  public Object getObjectValue (Object[] tuple) {
    …
  }                                          Interpretation strategy
  public void build(BytecodeBuilder builder, JoinPointSite call) {
    …
  }                                          Bytecode generation strategy
  public void generateASM(Assembler asm, BaselineCompilerState compilerState) {
    …
  }
  public void generateIR(IR2IR_bc2ir, GenerationContext gc) {
    …
  }                                          Just-in-time compilation strategy
}
```

compilation context

- Modular implementation in meta-model entity
- Choose best strategy supported by execution environment
- **Transparent to the user**

Levels of Meta-Object Protocol



1. Find corresponding execution mod
2. Perform semantics of **ArgumentContext**
3. Perform semantics of **InstanceofPred**
   passing result of step 2
4. Execute actions
   in top box, if step 3
   in bottom box, otherwi
   a. Evaluate exposed c ues and ex ions

JIT Compile-Time MOP: Case Study

AspectJ and other aspect-oriented languages:

- Aspects are types and can be instantiated to objects
- Advice execute in context of aspect instance
- Advice are invoked implicitly
  - ⇦ **aspect instance creation/selection specified by rules**

- PerTupleContext implements **generalized aspect-instantiation model**
  - Creation: on-demand (implicit) or in-advance (explicit)
  - Selection: based on context-value tuple

JIT Compile-Time MOP: Case Study

- Generic default implementation using map
- Optimizations depend on usage of PerTupleContext

- Example AspectJ issingleton():
  implicit, **context-insensitive**
  - Implicit
    - ⇦ Implementer has control over instantiation
  - Context-insensitive
    - ⇦ Result may be known at JIT compile-time

JIT Compile-Time MOP: Case Study

```
public class PerTupleContext extends Context implements … {
  public void generateASM(Assembler asm, BaselineCompilerState compilerState) {
    if (this.isInsensitive() && this.getContexts().isEmpty()) {
      if (!this.hasInstance()) {
        // instance still uninitialized?
        //   allocate instance in non-moving heap
        //   store instance
        // load instance
      }
      else {
        asm.emitPUSH_Imm(
          Magic.objectAsAddress(this.getInstance())
          .toInt()
        );
      }
    }
    else { … }
  }
}
```

instance not yet created

instance already created

Preliminary performance evaluation promising

Lessens Learnt

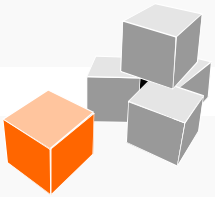
- Separation of roles works

Andre Loker → Martin Zandberg

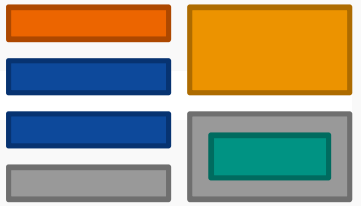
- ALIA4J's system of MOPs helps

Correct transition	• Regression test suite defined by Andre, applied by Martin
Re-use previous implementation	• Martin re-uses Andre's implementation for non-special cases
Avoiding unnecessary complexity	• See second case study in paper

Twente Research and Education on Software Engineering, Universiteit Twente



Software Technology Group
TU Darmstadt | FB Informatik



www.alia4j.org

http://www.alia4j.org

Christoph Bockisch, c.m.bockisch@cs.utwente.nl
Andreas Sewe, sewe@st.informatik.tu-darmstadt.de
Martin Zandberg

CASED

Re-use of LIAM entities

- ALIA4J includes implementations of many concepts
- **Most of them** implement at least **BytecodeSupport**
- Explored mapping many different languages