Reflex in a (Big) Nutshell

A Versatile Kernel for Multi-Language AOP in Java

Éric Tanter

Motivation

- Different approaches to AOP
  - models, general purpose VS domain specific
- Combining approaches
  - many aspects in a given application
  - ideally, one DSAL per aspect
  - BUT: need to manage aspect composition (across languages)

Why Domain Specific?

- Domain specificity brings:
  - declarative representation
  - simpler analysis and reasoning
  - domain-level error checking and optimizations

```
synchronize: Buffer
```

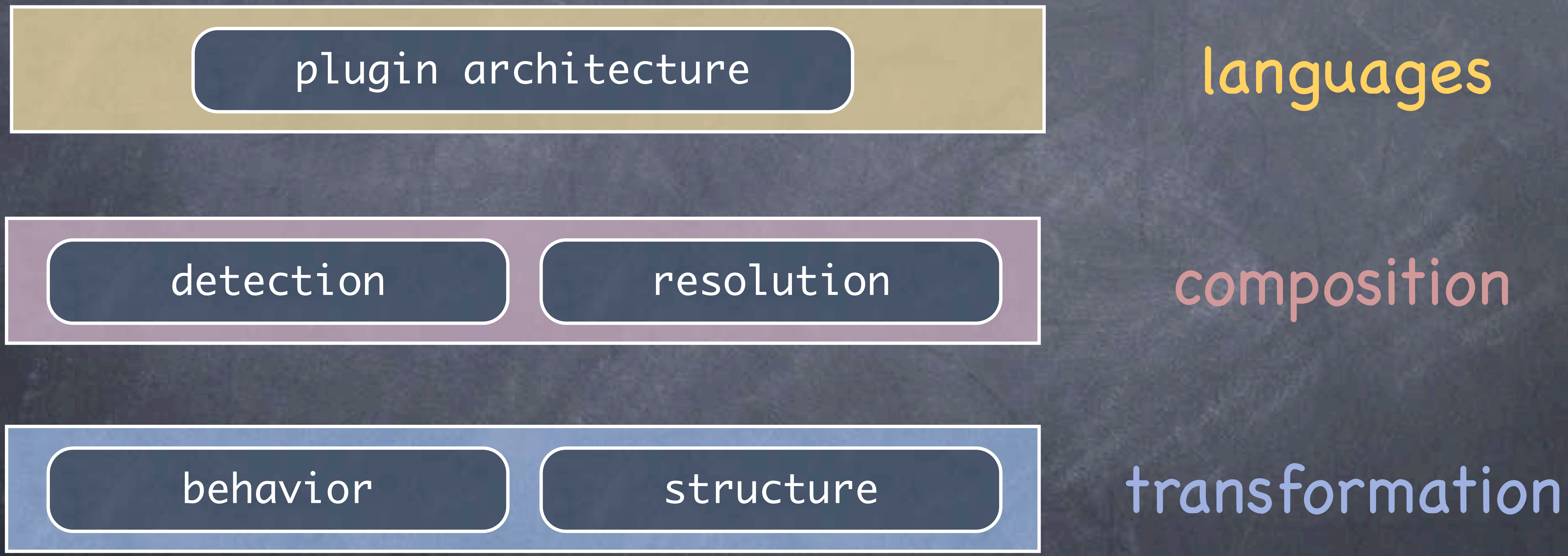
```
public aspect Synchronize {
  pointcut mutex(Buffer b):
    execution(Buffer.*(..)) && !cflowbelow(mutex)
                            && this(b);
  before(Buffer b): mutex(b) { LockMgr.enter(b); }
  after(Buffer b): mutex(b) { LockMgr.exit(b); }
}
```

AOP Kernel

a mediator for multi-language AOP

- Facilitate definition of new aspect languages
  - convenient API for transformation
  - mechanism for modular definition (plugins)
- Ensures proper composition of aspects
  - detection of aspect interactions
  - expressive/extensible means for their resolution

AOP Kernel Architecture



plugin architecture

languages

detection

resolution

composition

behavior

structure

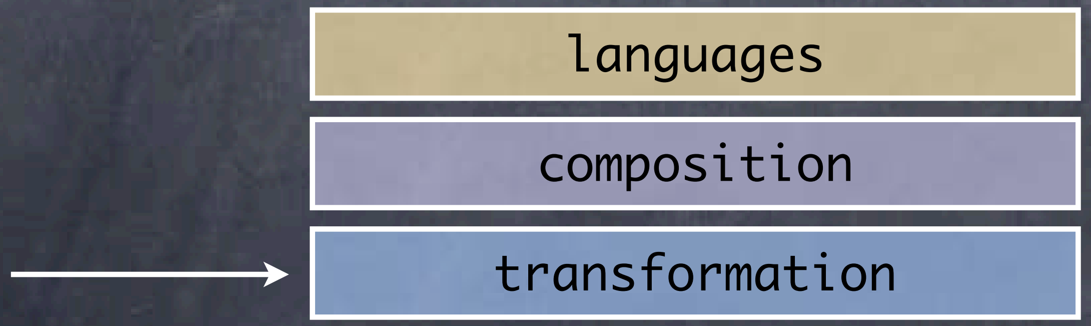
transformation

Reflex

- Basic Topics
  - model: explicit links
  - behavioral and structural links
  - operational schema
  - configuration
- Advanced Topics
  - composition
  - plugins for aspect languages

Reflex in a (Big) Nutshell

Basic Topics



languages

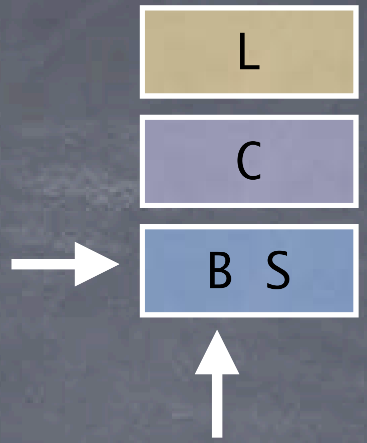
composition

transformation

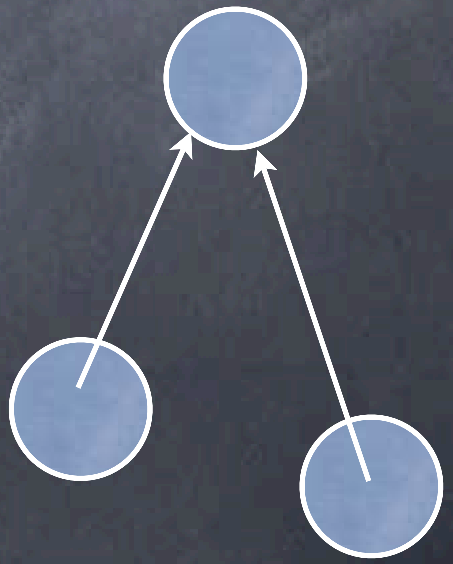
Model: Explicit Links

- Intermediate abstraction for driving transformation
- Link binding a cut to an action
  - cut = where in a program?
  - action = what to do?
- Links are first class
  - used for reporting/resolving composition issues

Behavior



- Essence of MOPs and AOP: implicit invocation
  - some "modules" "talking to each other" without explicit calls
- technically:
  - referencing (from who to whom)
  - marshalling (which info, how)
  - calling (which method / interface)



Referencing: from who...

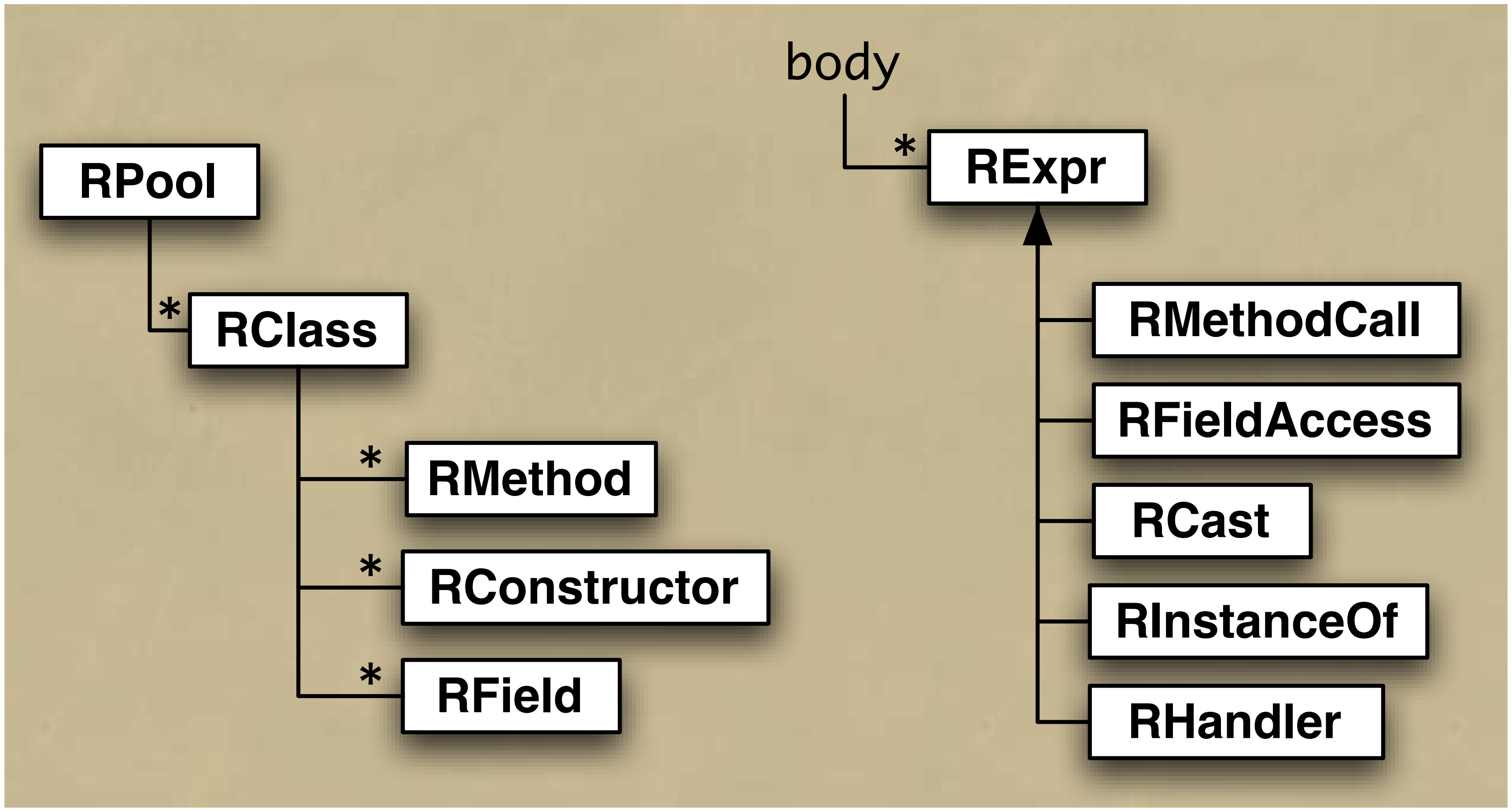
- Static specification
  - hooksets: composable sets of points where delegation will occur
  - link control: before, after, around
- Dynamic specification
  - link activation
    - select specific instances, cflow, etc.

```
Hookset hs = new PrimitiveHookset(
   FieldAccess.class,               // operation
   new ExtendsCS(“FigElement”),   // class predicate
   new PrivateOS());                // op predicate

BLink l = API.links().createBLink(hs, ...);
l.setControl(Control.AFTER);
l.setActivation(new isDisplayed());
```

```
MsgReceive, MsgSend, FieldAccess, Cast, Instantiation, etc.
```

```
interface ClassSelector {
  boolean accept(RClass c);
}
```

```
interface OperationSelector {
  boolean accept(Operation op,
                 RClass c);
}
```



Referencing: ...to whom

- metaobjects
  - can be any object!
  - bootstrapping: MODefinition
- binding: link scope
  - per caller / per caller class / unique

```
MODefinition def;
def = new MODefinition.Class(“Foo”); // new Foo instance
def = new MODefinition.Factory(F);   // query factory F
def = new MODefinition.SharedMO(o);  // shared object o

BLink l = API.links().createBLink(hs, def);
l.setScope(Scope.OBJECT);
```

Marshalling

- Which information to pass...
  - Parameter objects
    - open set: this, Xth arg, arg array, thread, time, etc.
    - may depend on the operation being intercepted (target type, method object, declared exceptions...)
    - standard parameters available
- and how
  - PassingMode

none	o.foo()
plain	o.foo(a,b,c)
array	o.foo([a, b, c])

encapsulated	o.foo(new A(a,b,c))
	o.foo(new A([a,b,c]))

Calling

- Which method to call
  - Call Descriptor (MOP specialization)
  - method name + declaring type

```
l.setMOCall(“Display”, “update”, new Parameter[]{
      Parameter.CONTEXT      // this (or class if static)
});
```

  - statically type-checked

Link attributes

- More attributes:
  - mintypes: type restrictions
  - declared type: avoid cast
  - initialization: eager/lazy (thread-safe or not)
  - updatable: change metaobject at runtime?

```
l.setDeclaredType(new DT(“Display”));
```

```
Display _mo_l1 = ...; // field
...
_mo_l1.update(this); // call
...
```

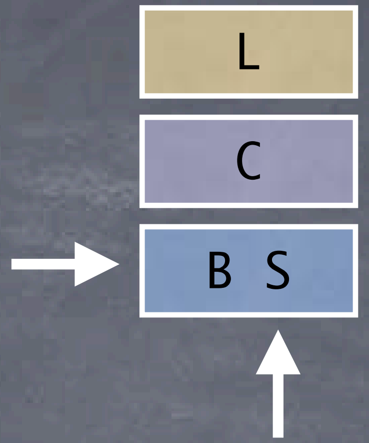
Runtime API

- Links are reified at runtime
  - RTLink
  - used to access/change metaobject (restricted)
  - and activation condition

```
API.links().addBLink(l); // link is registered
....
DisplayAPI.setLink(l.getRTLink()); // export RTLink
```

```
class DisplayAPI {
 ..turnOff(FigEl o){ l.setActive(o, Active.OFF); }
 ..displayOn(FigEl o, Display d){ l.setMetaobject(o, d); }
}
```

Structure


L

C

B S

- Perform structural modifications
  - add method, field, interface, change name, etc.
- Structural link: SLink
  - class selector
  - structural metaobject

```
API.links().addSLink(
  new MyClassSelector(), // cut condition
  new AddLoaderTrace()       // action
);
```

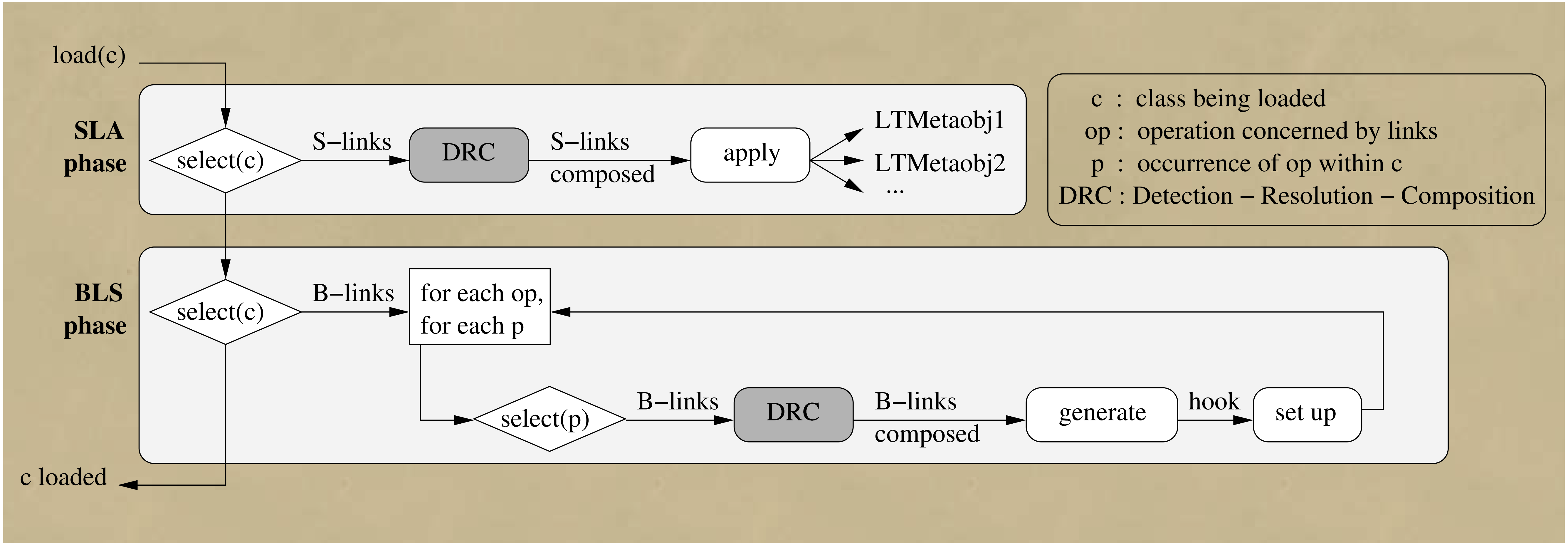
```
class AddLoaderTrace implements SMetaobject {

  void handleClass(RClass c) {
    String init =
      “{ print(” + c.getName() + “.class.getClassLoader()); }”;

    c.addClassInit(MemberFactory.makeClassInit(init, c));
  }
}
```

```
class A {
  static {
   print(A.class.getClassLoader());
  } ...
}
```

```
class B {
  static {
   print(B.class.getClassLoader());
  } ...
}
```

Operational Schema

load-time phases



c : class being loaded
op : operation concerned by links
p : occurrence of op within c
DRC : Detection – Resolution – Composition

Configuration

- reflex.API
  - links(): manage BLinks and SLinks
  - rules(): manage composition rules (more on this later)
- Initial (static) configuration
  - configuration classes
  - plugins for aspect languages (more on this later)
- Runtime configuration
  - API accessible
  - implementation restriction: no class reloading

Configuration Classes

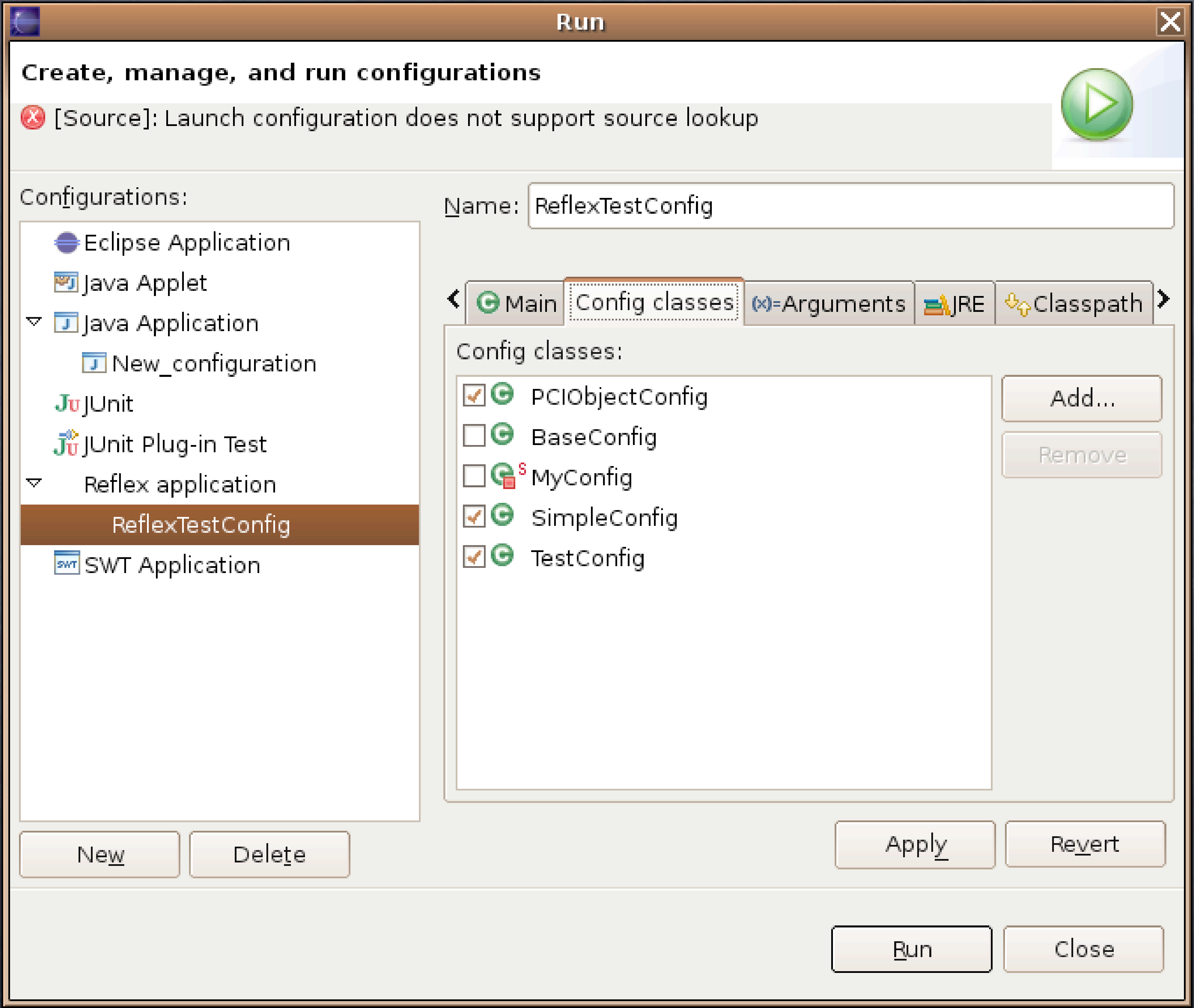
```
interface IReflexConfig {
  void initReflex();
}
```

```
class DisplayConf implements IReflexConfig {
  void initReflex() {
    Hookset h = ...;
    BLink l = API.links().createBLink(...);
    bl.set...; // set attributes
    API.links().addBLink(bl);
    DisplayAPI.setLink(bl);
} }
```

```
class TraceLoading implements IReflexConfig {
  void initReflex() {
    API.links().addSLink(...);
} }
```

```
java reflex.Run -configClasses DisplayConf:TraceLoading
                DrawingApp
```

Config in Eclipse



Create, manage, and run configurations

[Source]: Launch configuration does not support source lookup

Configurations:

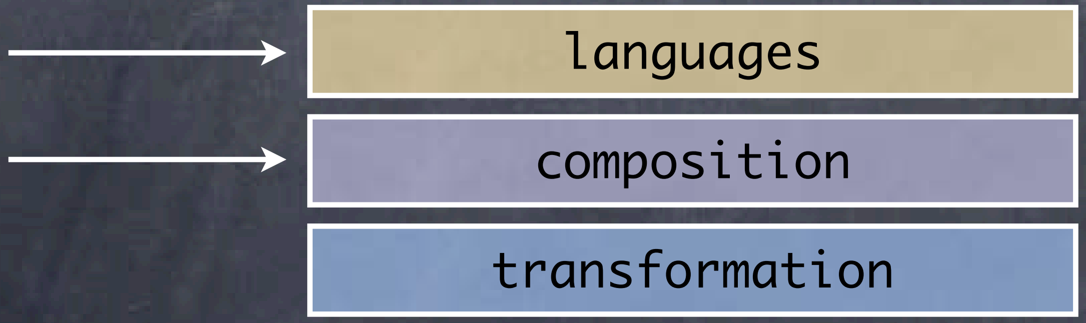
Name: ReflexTestConfig

Config classes:

☑ PCIObjectConfig
□ BaseConfig
□ MyConfig
☑ SimpleConfig
☑ TestConfig

Reflex in a (Big) Nutshell

Advanced Topics



Composition

L

→ C

T

- implicit cut
  - B applies whenever A does

share cut (hookset/class set)

- mutual exclusion
  - B never applies when A does
- aspects of aspects
  - B applies on A

B’s cut on A’s metaobjects

- visibility of structural changes
  - A adds a field, should B see it?
- order of application
  - both A and B apply, which goes first?

Interaction Detection

- Reflex detects interactions
  - at the hook level (lazy detection)
- and reports on interactions
  - warning (trace, GUI)
  - error
  - silent (arbitrary composition)

Interaction Resolution

- Interaction selectors
  - for mutual exclusion and other dependencies
  - attached to links
  - for now, only static

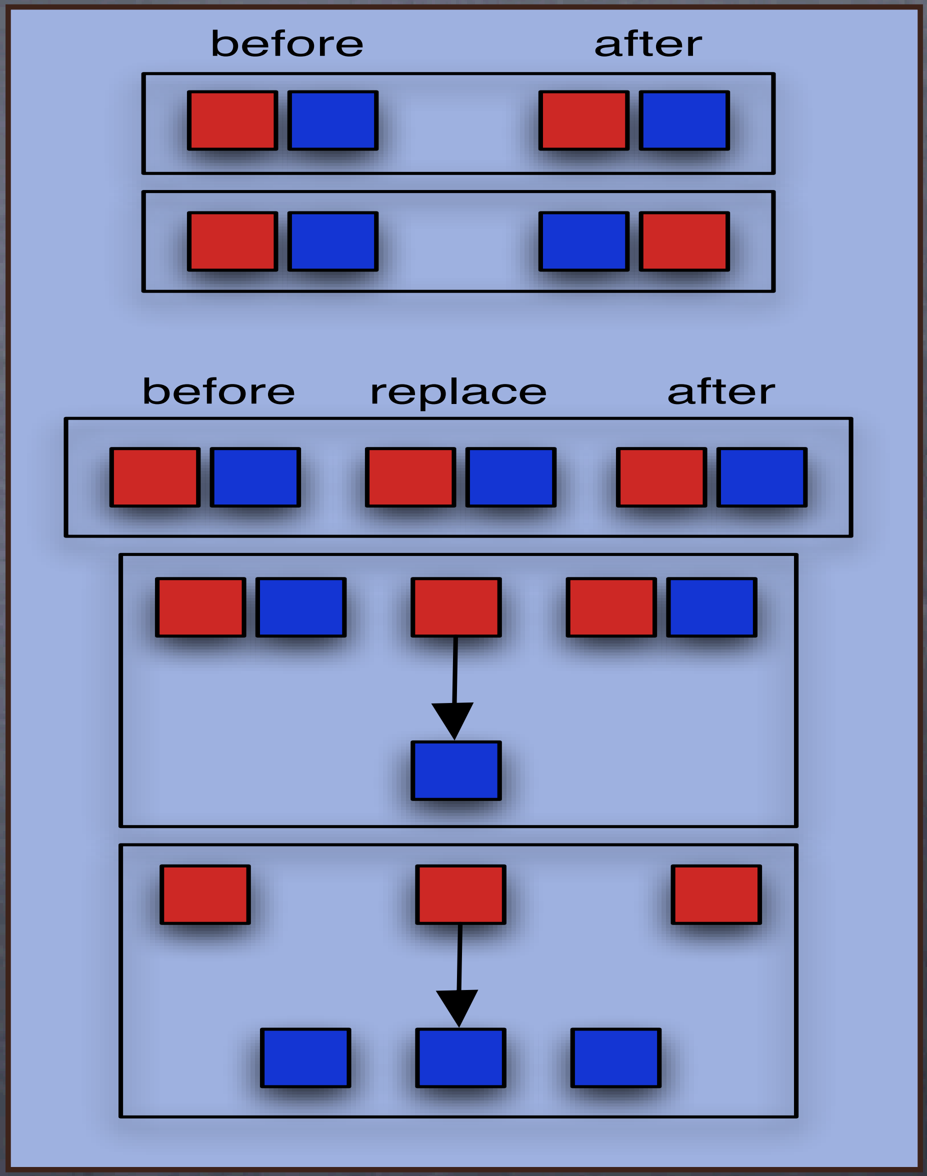
```
interface InteractionSelector {
 bool accept(LinkInteraction li);
}
```

```
link.setInteractionSelector(is);
```

- Composition operators and rules
  - approach based on formal work of Douence et al. [reflection'01, gpce'02, aosd'04]

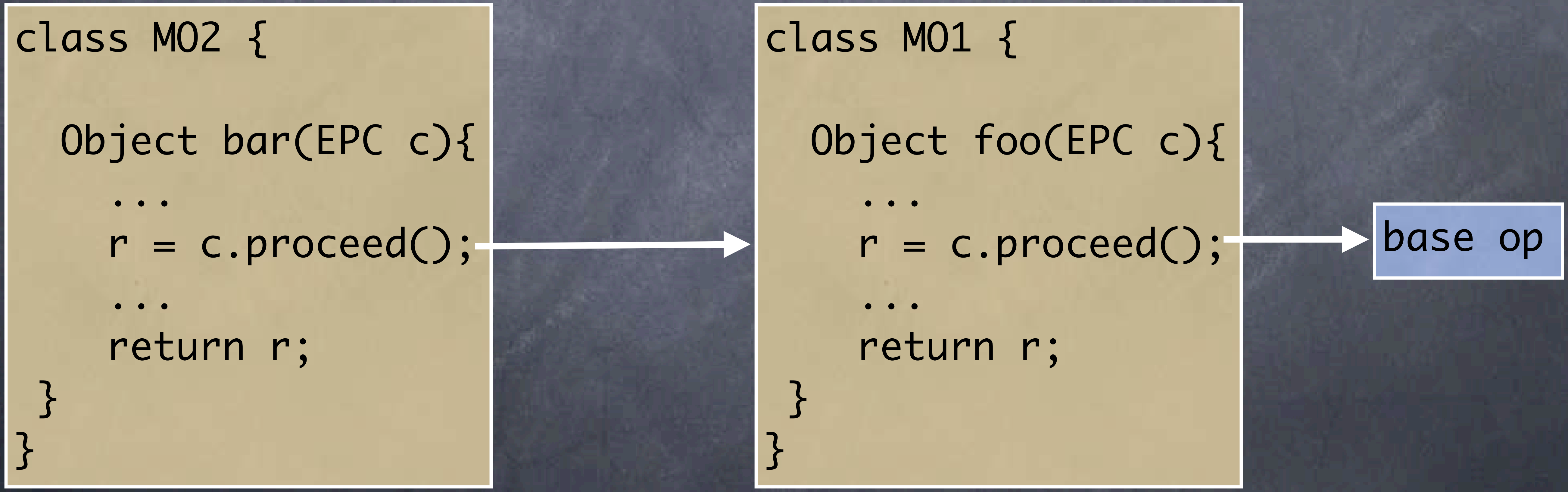
Interaction Resolution

- Operators for ordering and nesting (proceed)
- Kernel operators
  - ord(x,y) nest(x,y)
  - operate on link elements
- Composition operators
  - defined on top of k-ops
  - operate on links
  - eg. Seq, Wrap, Fst



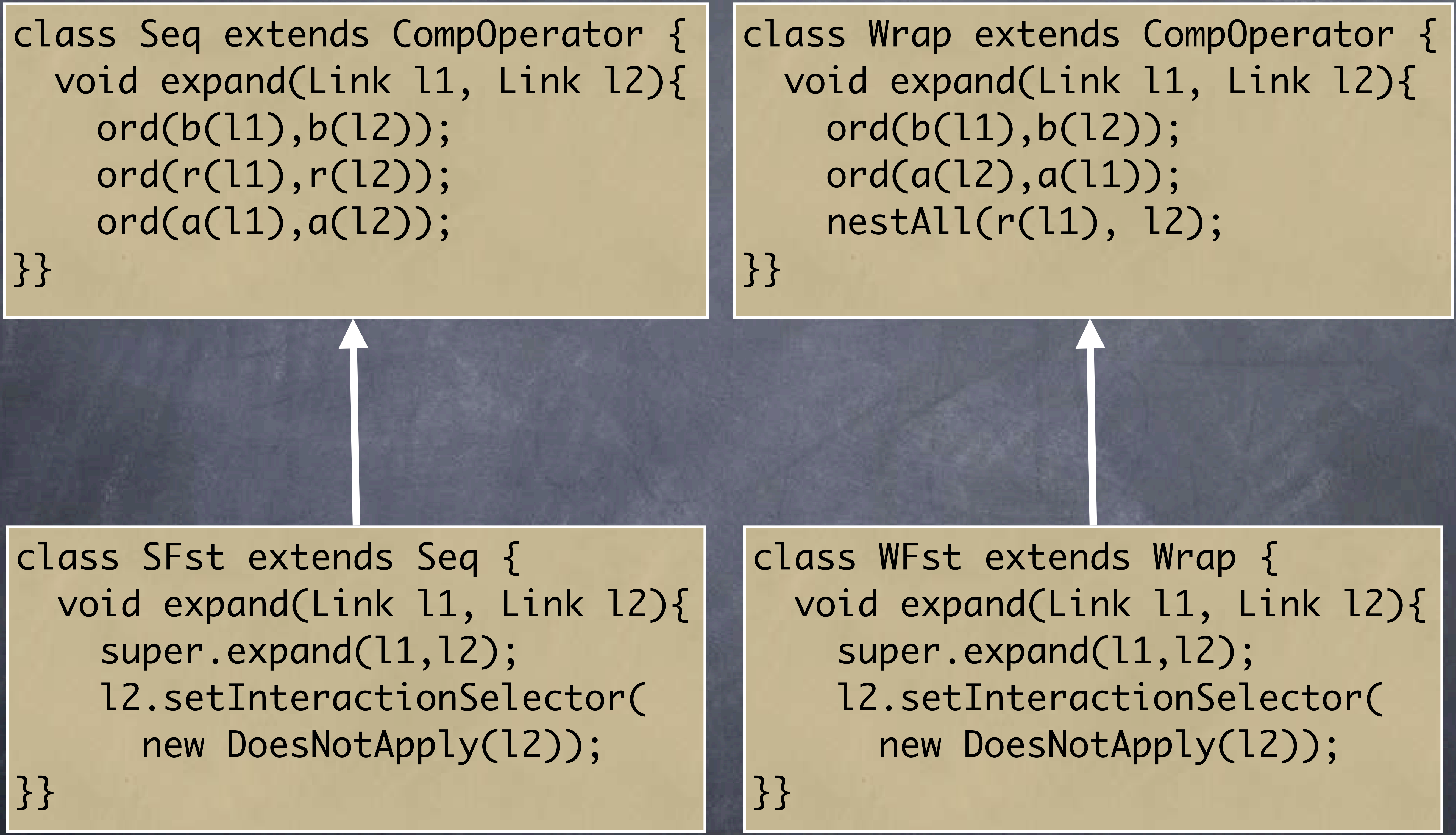
Nesting

- hierarchical precedence (cf. AspectJ)
- special parameter: execution point closure (EPC)



```
class MO2 {

  Object bar(EPC c){
    ...
    r = c.proceed();
    ...
    return r;
 }
}
```

```
class MO1 {

  Object foo(EPC c){
    ...
    r = c.proceed();
    ...
    return r;
 }
}
```

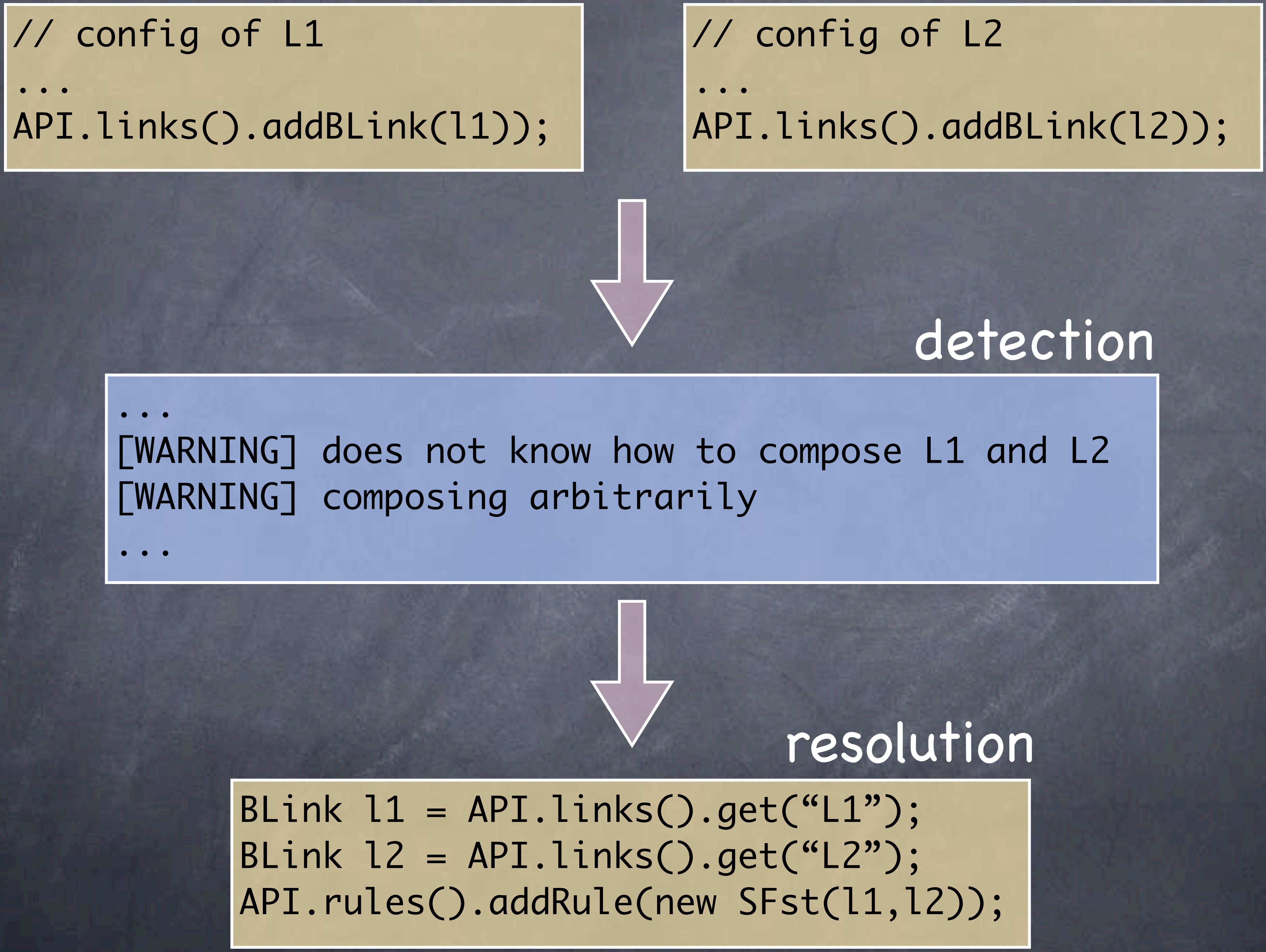


```
class Seq extends CompOperator {
  void expand(Link l1, Link l2){
    ord(b(l1),b(l2));
    ord(r(l1),r(l2));
    ord(a(l1),a(l2));
}}
```

```
class Wrap extends CompOperator {
  void expand(Link l1, Link l2){
    ord(b(l1),b(l2));
    ord(a(l2),a(l1));
    nestAll(r(l1), l2);
}}
```

```
class SFst extends Seq {
  void expand(Link l1, Link l2){
    super.expand(l1,l2);
    l2.setInteractionSelector(
      new DoesNotApply(l2));
}}
```

```
class WFst extends Wrap {
  void expand(Link l1, Link l2){
    super.expand(l1,l2);
    l2.setInteractionSelector(
      new DoesNotApply(l2));
}}
```



```
// config of L1
...
API.links().addBLink(l1));
```

```
// config of L2
...
API.links().addBLink(l2));
```

detection

```
...
[WARNING] does not know how to compose L1 and L2
[WARNING] composing arbitrarily
...
```

resolution

```
BLink l1 = API.links().get(“L1”);
BLink l2 = API.links().get(“L2”);
API.rules().addRule(new SFst(l1,l2));
```

Structural Changes

- By default, changes are hidden
- Can be customized
  - always hidden, always visible
  - MemberSelector

```
c.addField(f);
```

```
Field[] fs = c.getFields();
```

```
c.addField(f, ALWAYS_VISIBLE);
```

```
f.setProperty(key,val);
c.addField(f);
```

```
sel = new MemberSelector(){
  boolean accept(RMember m){
    return m.getProperty(key) == val;
}}
Field[] fs = c.getFields(sel);
```

Aspect Languages

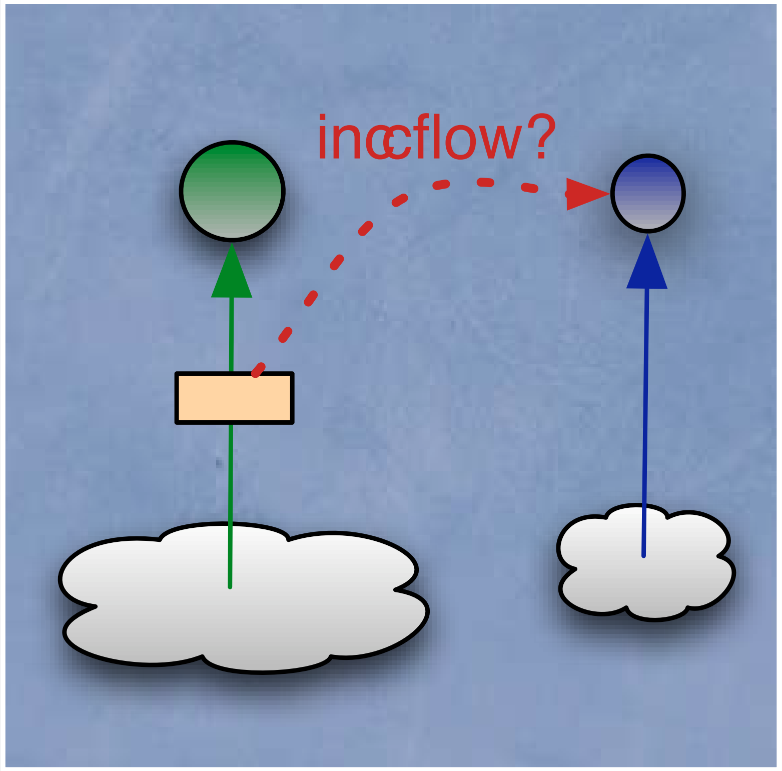
- Kernel API: mid-level abstractions
  - higher level than direct bytecode transformation
  - lower level than aspect languages (ALs)
- Plugin architecture
  - plugin = transformer from AL to kernel
  - reuse transformation and composition facilities
  - detection/resolution of interactions of aspects defined in different languages
- Current plugins: SOM, (subset of) AspectJ

Abstraction Gap

1 aspect

```
aspect DisplayUpdate {
  pointcut move(): execution(...);
  pointcut topLevelMove():
    move() && !cflowbelow(move());

  after(): move() { Display.update(); }
}
```

2 links



- Challenge: composition
  - intra-pointcut: cflow vs. cflowbelow
  - intra-aspect: no textual ordering!
  - inter-aspects : do not care about links
    - linksets: package related links together
    - 1 aspect = 1 linkset

DisplayUpdate.aj

```
aspect DisplayUpdate {
  pointcut move(): execution(...);
  pointcut topLevelMove():
    move() && !cflowbelow(move());

  after(): move() { Display.update(); }
}
```

sync.som

```
schedule: Buffer with: BufferScheduler;
schedule: Dictionary with: ReaderPriority;
```

```
java reflex.Run -aj DisplayUpdate.aj -som sync.som
                -configClasses TraceLoading
                DrawingApp
```

http://reflex.dcc.uchile.cl