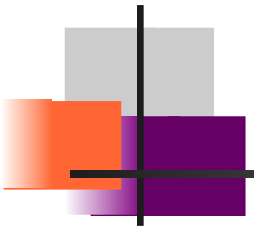
Invasive Software Composition

Uwe Aßmann

Research Center for Integrational Software Engineering (RISE)

PELAB IDA

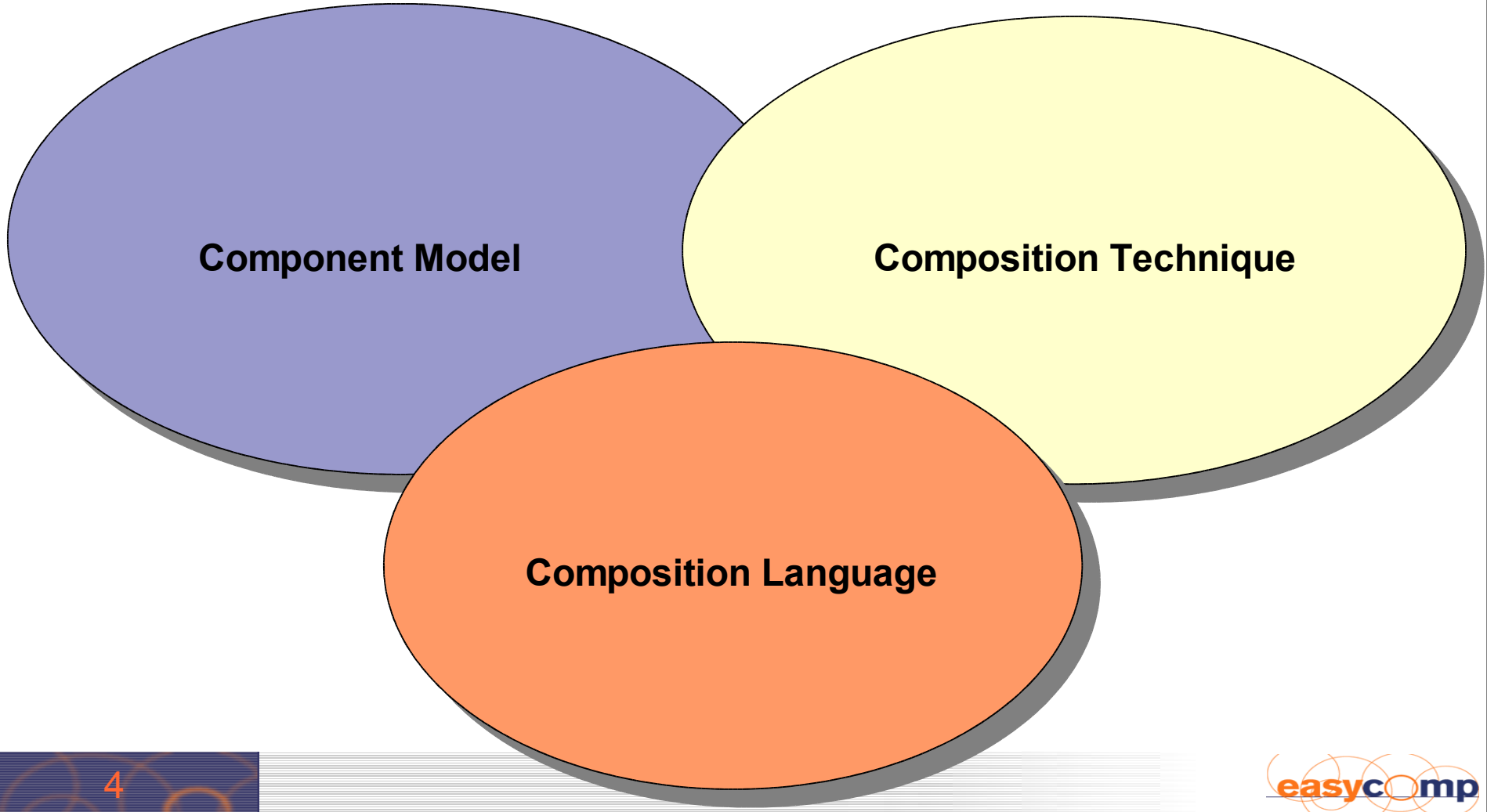
Linköpings Universitet

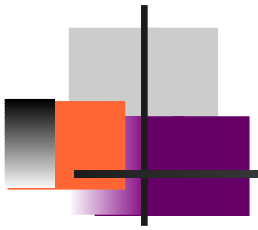


Contents

- A little history of software composition
  - Comparison criteria for composition
- How it is realized for Invasive Software Composition
- Future software composition systems

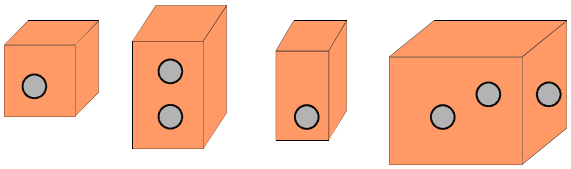
Software Composition



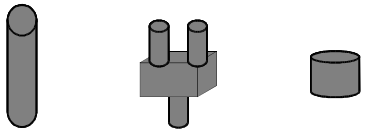


Historical Approaches to Components

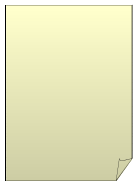
Blackbox Composition



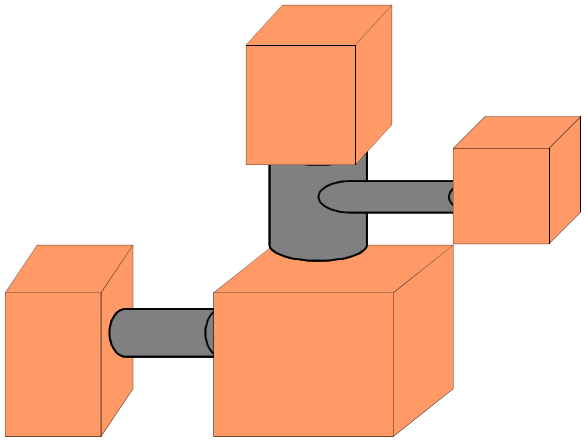
Components



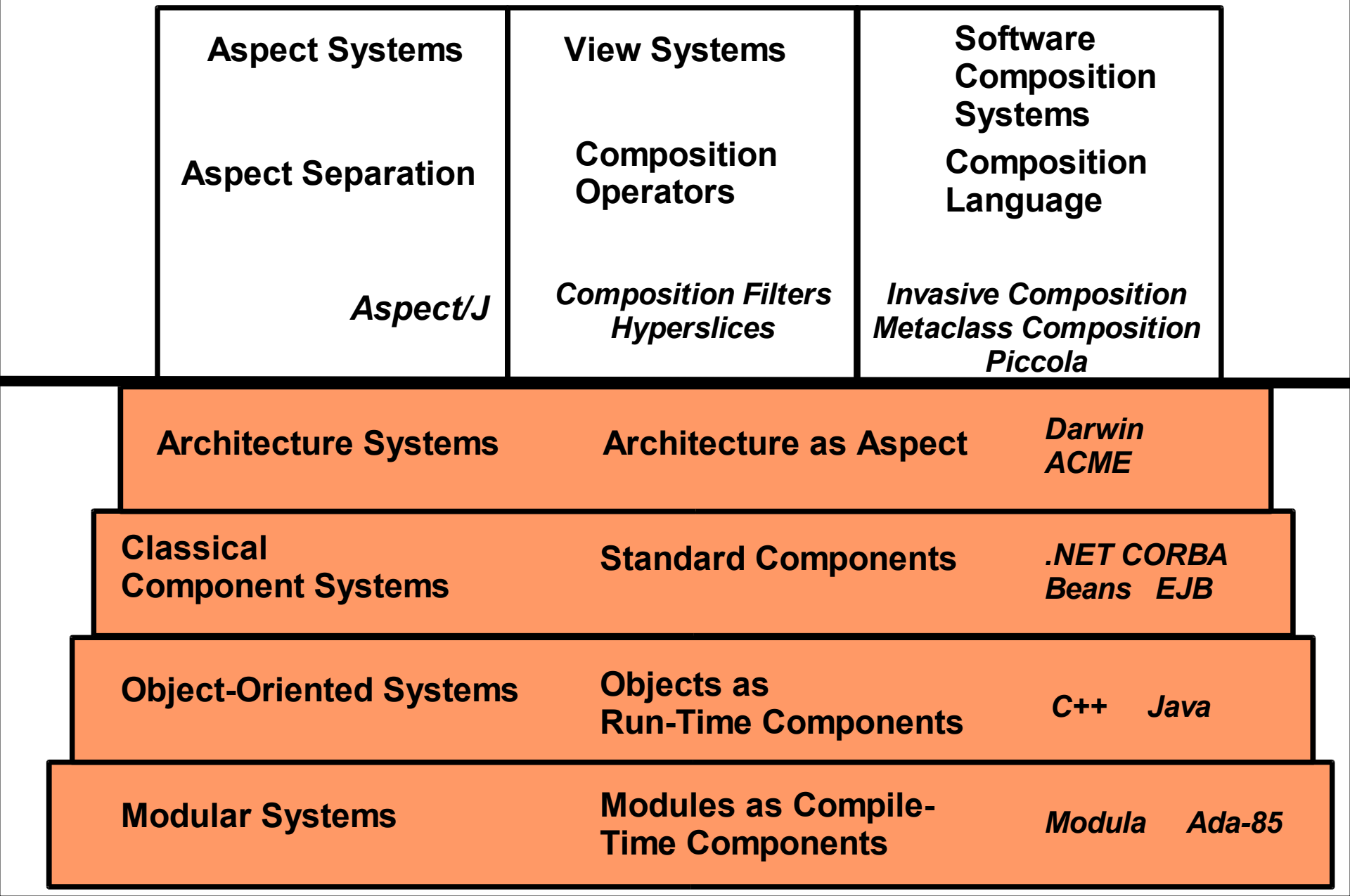
Connectors



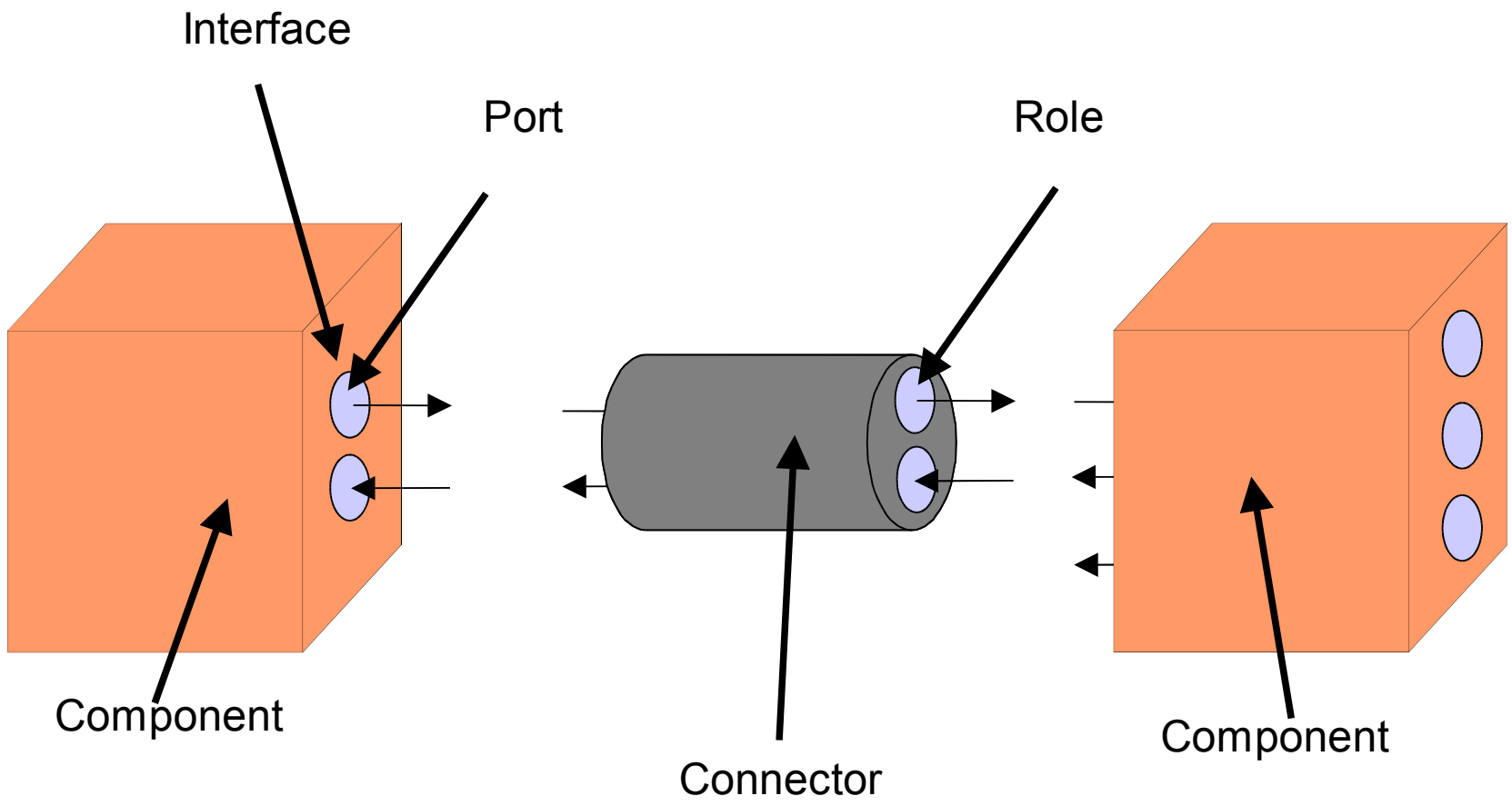
Composition recipe



Component-based applications

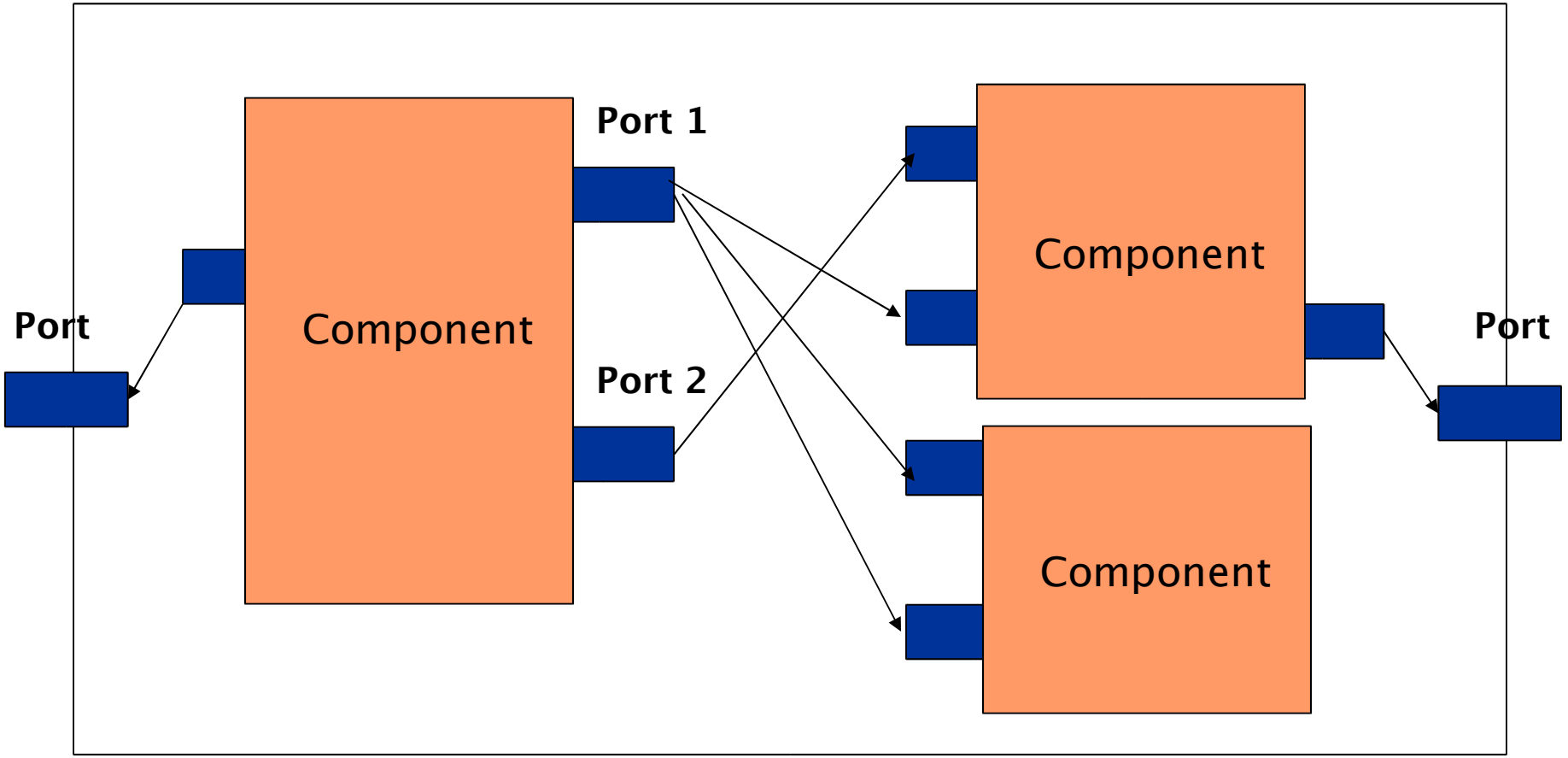

Aspect Systems
Aspect Separation
Aspect/J
View Systems
Composition Operators
Composition Filters
Hyperslices
Software Composition Systems
Composition Language
Invasive Composition
Metaclass Composition
Piccola
Architecture Systems
Architecture as Aspect
Darwin
ACME
Classical Component Systems
Standard Components
.NET CORBA
Beans EJB
Object-Oriented Systems
Objects as Run-Time Components
C++ Java
Modular Systems
Modules as Compile-Time Components
Modula Ada-85

Most Advanced: Software Architecture Systems



Architecture can be exchanged independently of components

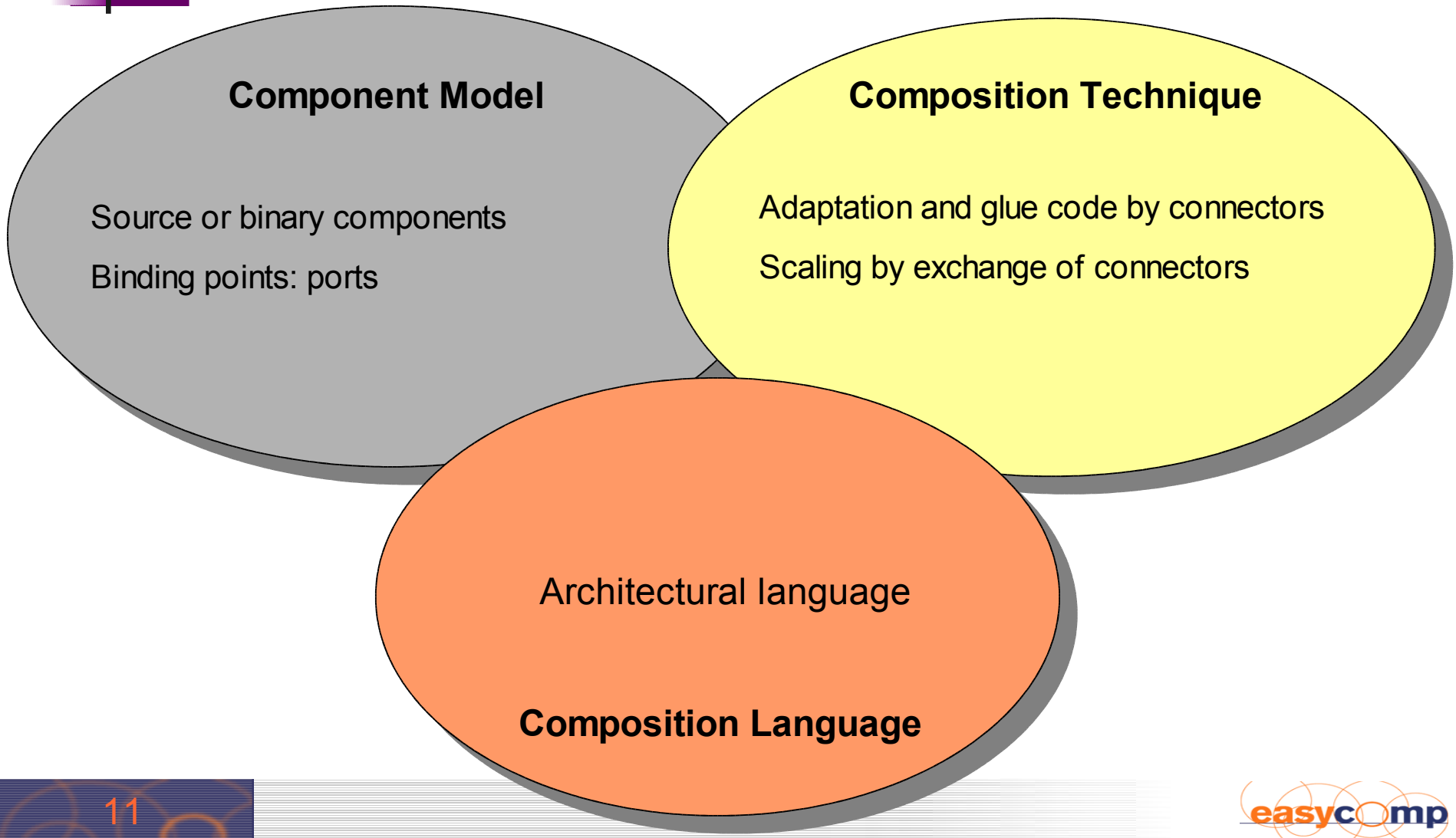
Reuse of components and architectures is fundamentally improved

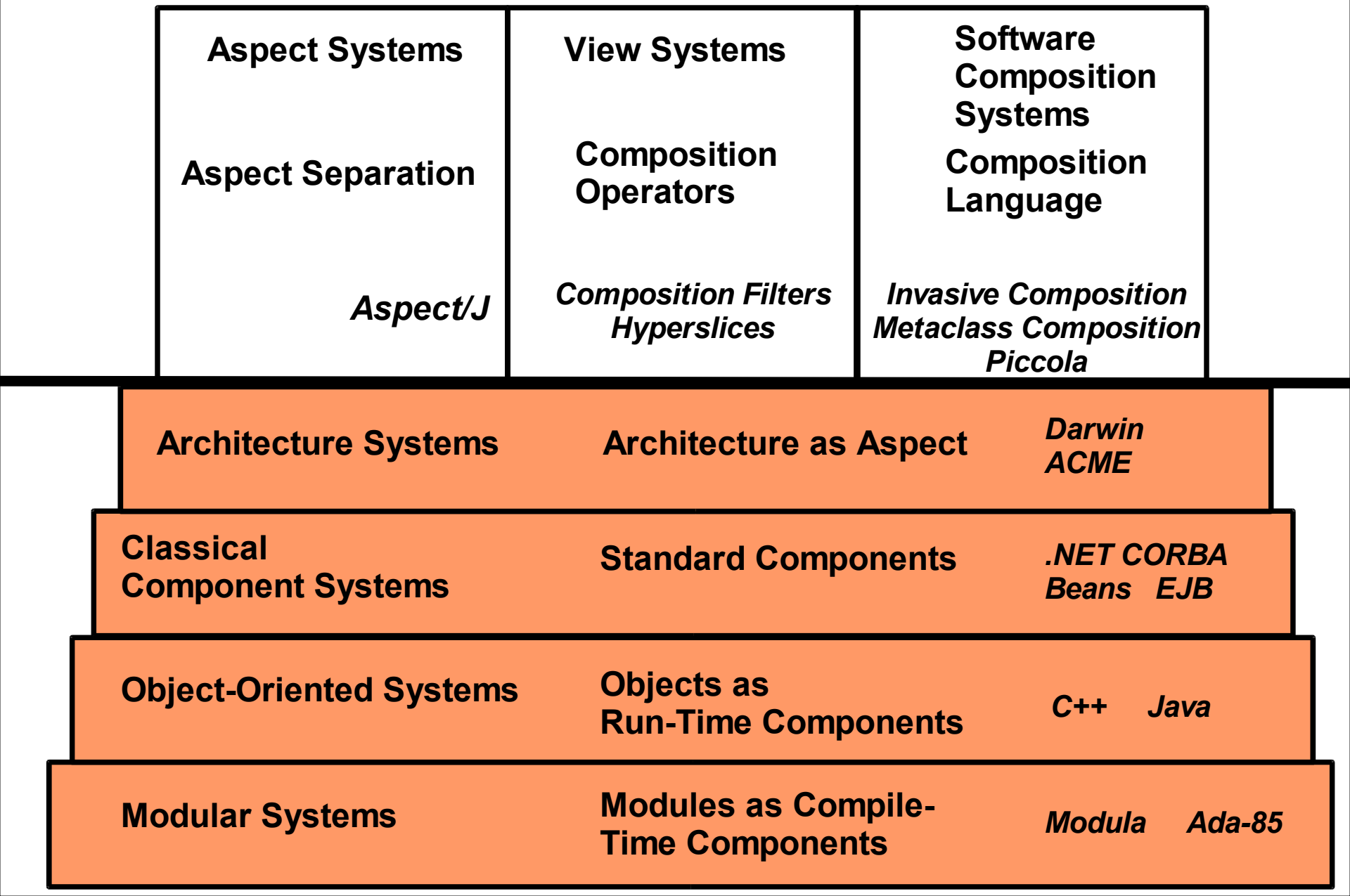


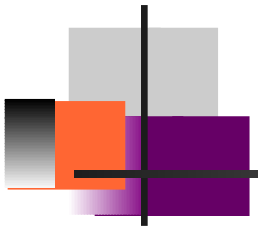
Architecture Systems

- ACME (Garlan, CMU)
- Darwin (Kramer, Magee, Imperial College)
- Unicon (Shaw, CMU)
- CoSy (ACE b.V., Amsterdam, commercialized for compilers of embedded systems, http://www.ace.nl)

Architecture Systems as Composition Systems




Aspect Systems
Aspect Separation
Aspect/J
View Systems
Composition Operators
Composition Filters
Hyperslices
Software Composition Systems
Composition Language
Invasive Composition
Metaclass Composition
Piccola
Architecture Systems
Architecture as Aspect
Darwin
ACME
Classical Component Systems
Standard Components
.NET CORBA
Beans EJB
Object-Oriented Systems
Objects as Run-Time Components
C++ Java
Modular Systems
Modules as Compile-Time Components
Modula Ada-85



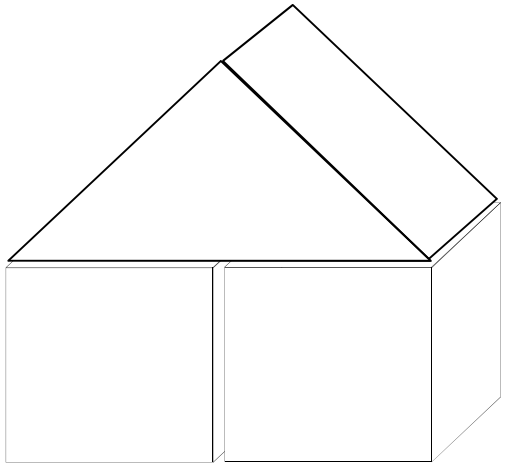
Graybox Component Models

The Essence of the Last 5 Years

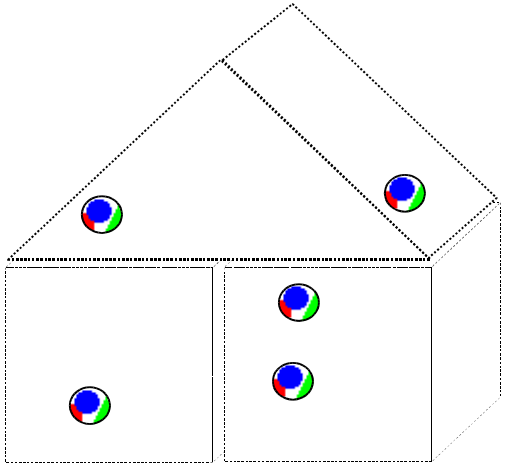
- **Aspect-oriented Programming**
- **View-based Programming**

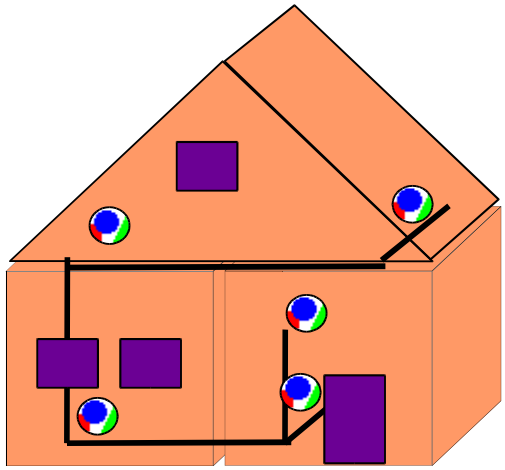
Component Integration

Structure



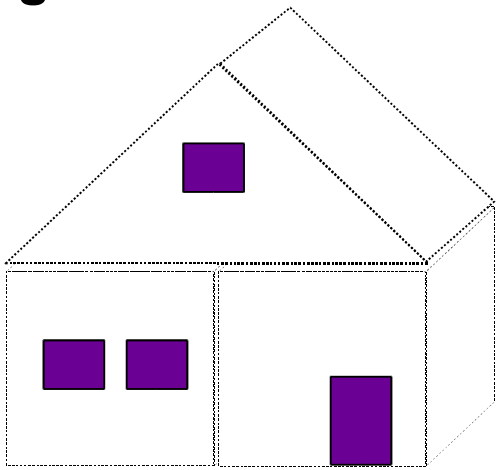
Interfaces



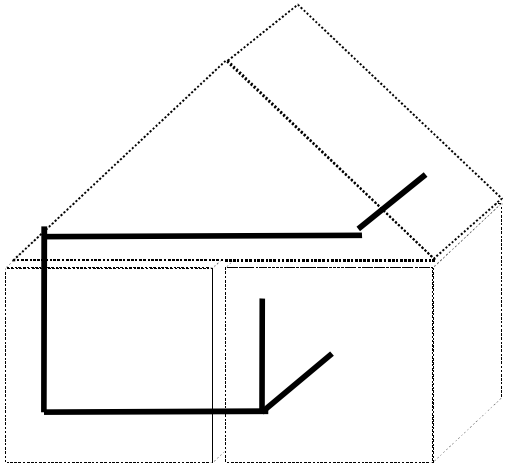


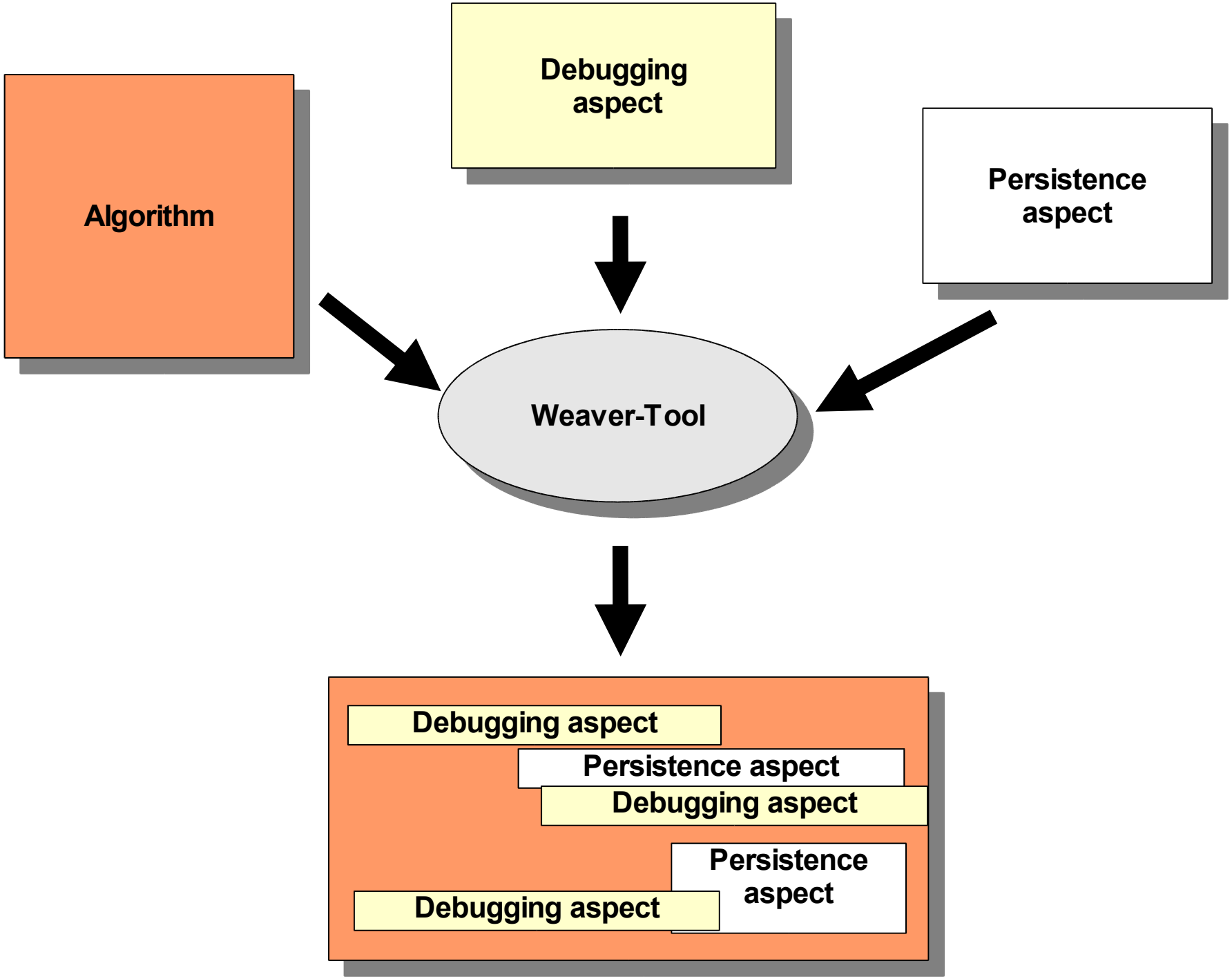
Integrated House

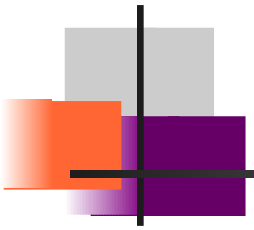
Light Plan



Pipe Plan




Debugging aspect
Algorithm
Persistence aspect
Weaver-Tool
Debugging aspect
Persistence aspect
Debugging aspect
Persistence aspect
Debugging aspect



Aspect Systems

- Aspect languages
  - Every aspect in a separate language
  - Domain specific
  - Weaver must be build (is a compiler, much effort)
- Script based Weavers
  - The weaver interprets a specific script or aspect program
  - This introduces the aspect into the core

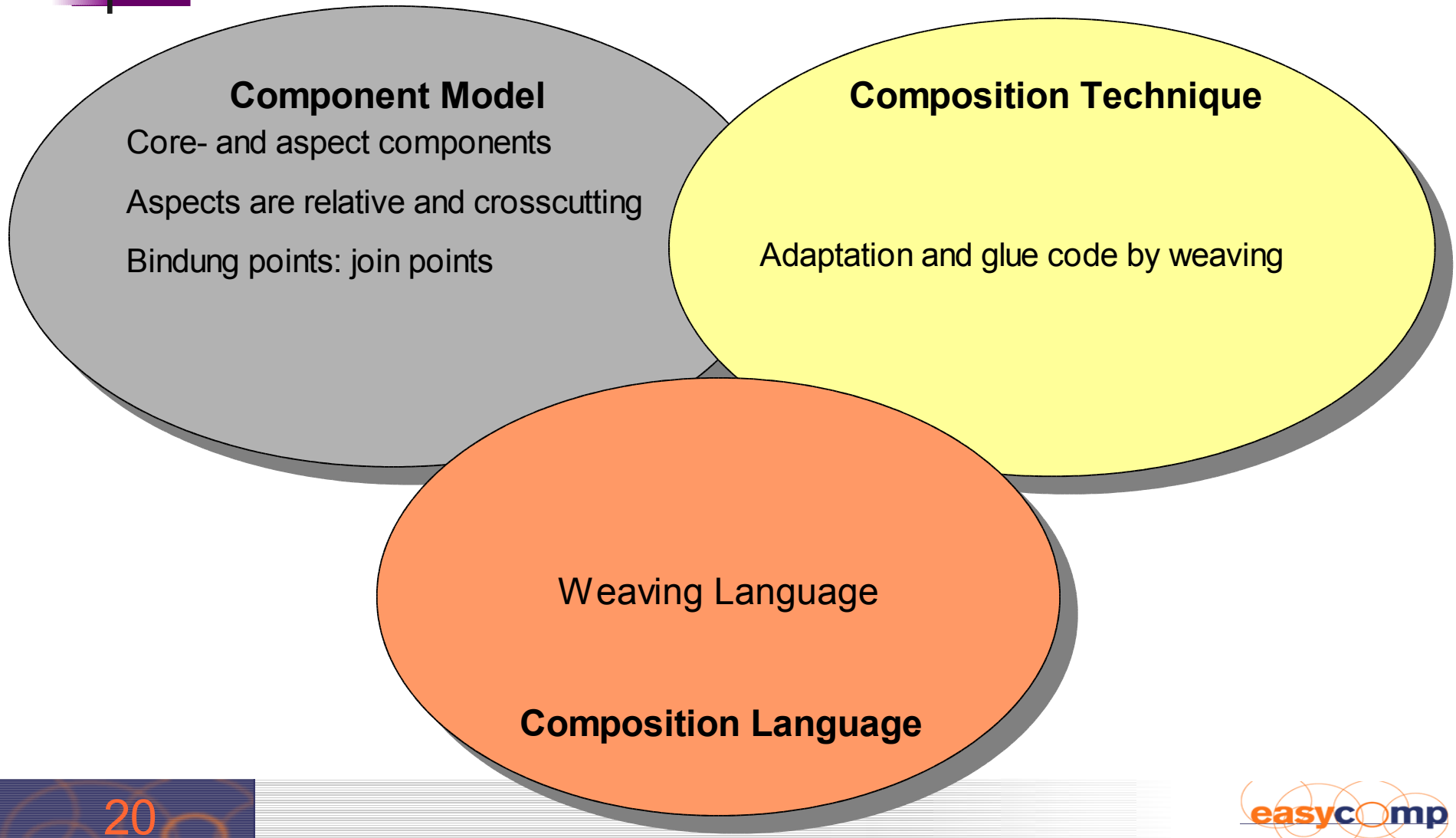
Example: Inject/J injectj.fzi.de

- Script based weaver (T. Genssler)
  - More powerful composition language than Aspect/J
- Based on explicit static metaprogramming
  - Navigations on classes and methods of the core
  - Pattern matching
  - Weaving in code at arbitrary places
- Builds on Java RECODER http://recoder.sf.net
- Useful for
  - Automated refactorings
  - Compositions
  - Generative Programming

Inject/J

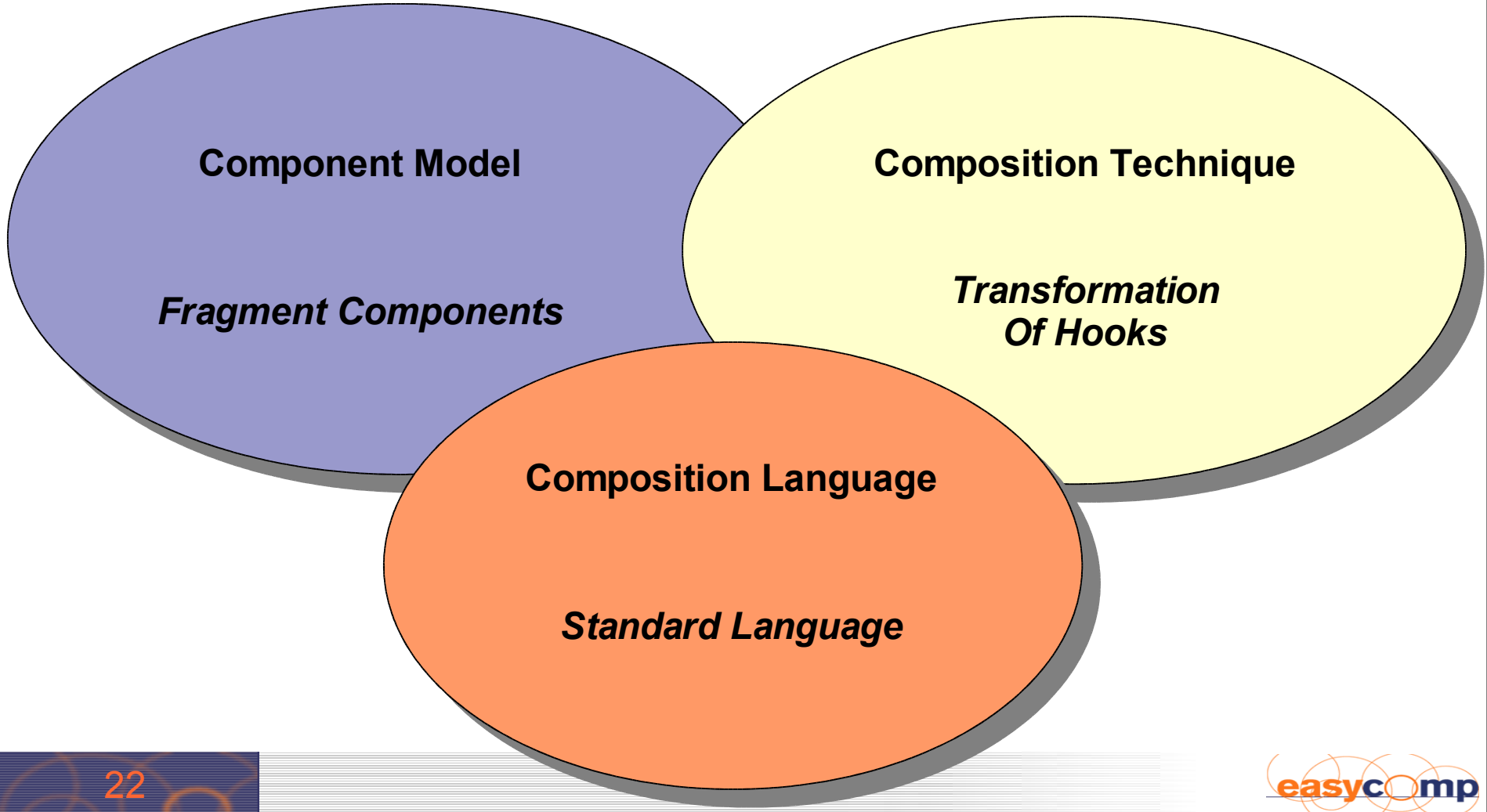
```
script BeforeAfterExample {
        // Only visit classes in package Testpackage
       foreach class 'Testpackage.*' <=c> do {
              // In this class, visit all methods with no parameters
             foreach method '*()' <=m> do {
                    // Now insert in some debug code in the method body...
                   before ${
                      System.out.println("Entering <m.signature> in class <c.name>");
                   }$;
                   after ${
                      System.out.println("Leaving ..");
                   }$;
             }
       }
}
```

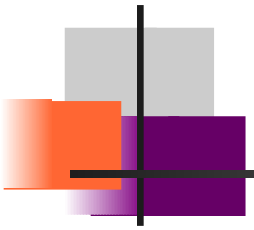
Aspect Systems As Composition Systems



Invasive Software Composition - A Fragment-Based Composition Technique

Invasive Composition



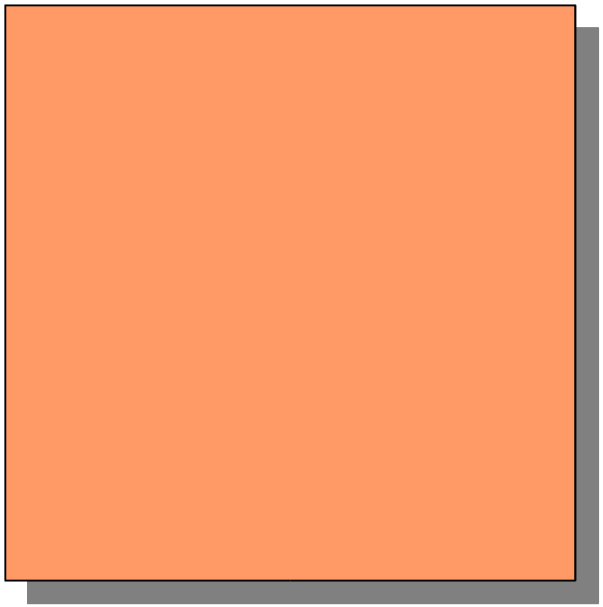


Invasive Composition

Invasive composition
adapts and extends
components
at hooks
by transformation

The Component Model of Invasive Composition

- **The component is a *fragment container (fragment box)***
  - a set of fragments/tag elements
- **Uniform representation of**
  - a software component
  - a class, a package, a method
  - an aspect
  - a meta description
  - a composition program



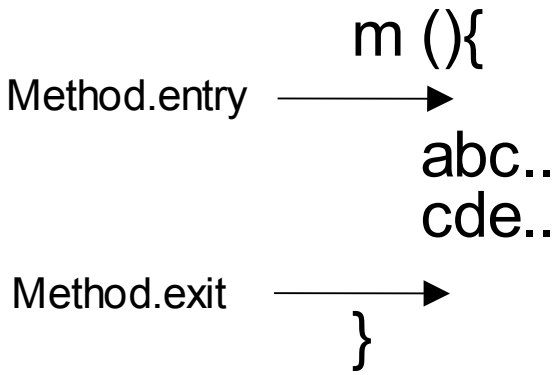
Fragment Components Have Hooks

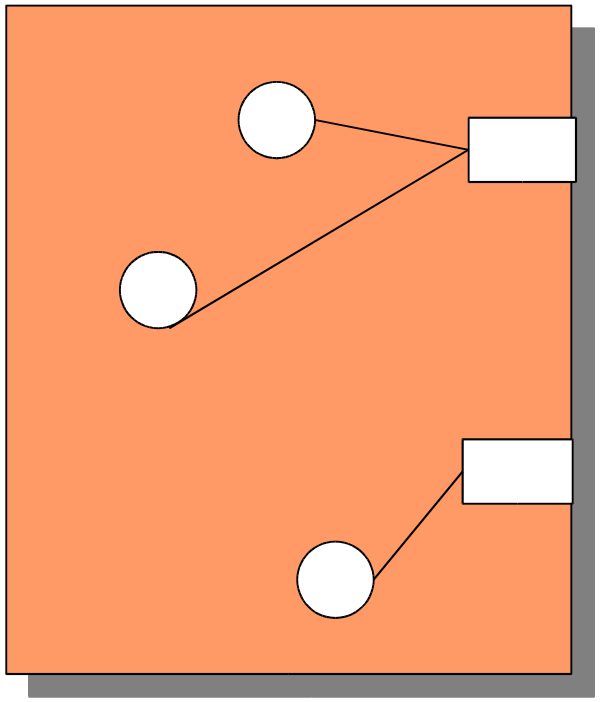
Hooks are variation points of a component:
fragments or positions,
which are subject to change

- **Software variation points**
  - method entries/exits
  - generic parameters

Implicit Hooks In Software

- Example Method Entry/Exit

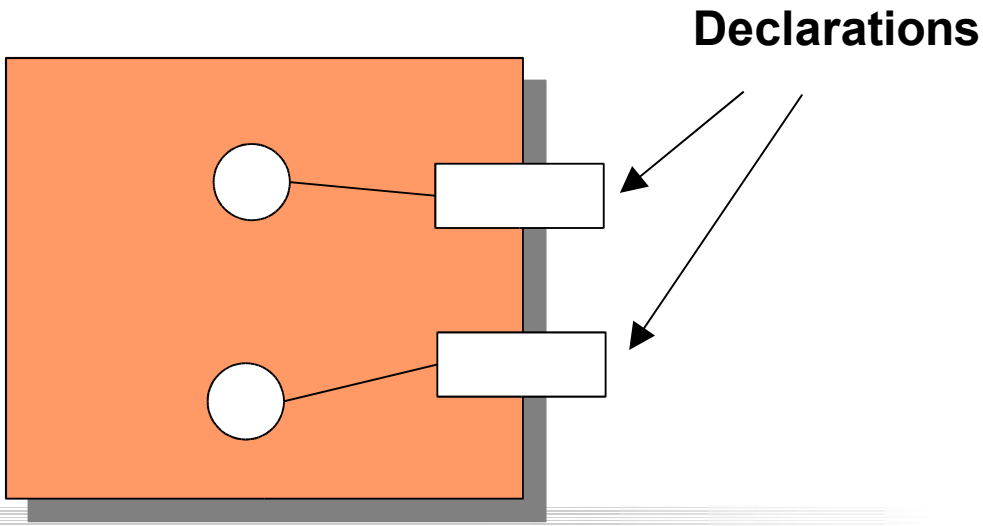




Given by the programming language

Declared Hooks

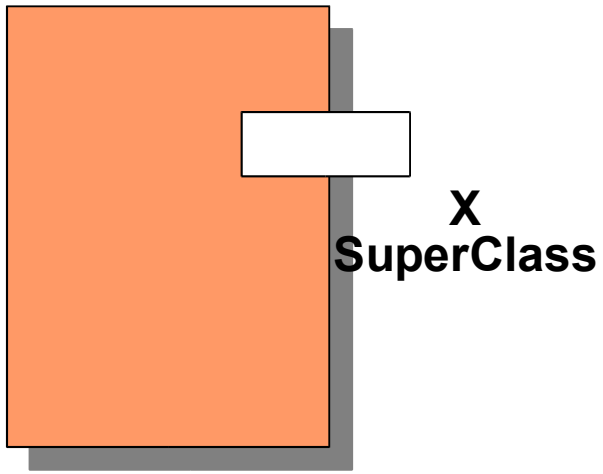
Declared Hooks are declared by the box writer as variables in the hook's tags.



Declaration of Hooks

- **Language extensions with new keywords**
- **Markup Tags**
- **Standardized Names (Hungarian Notation)**
- **Comment Tags**

```
<superclasshook> X </superclasshook>

class Set extends genericXSuperClass {  }

class Set  /* @superClass */
```



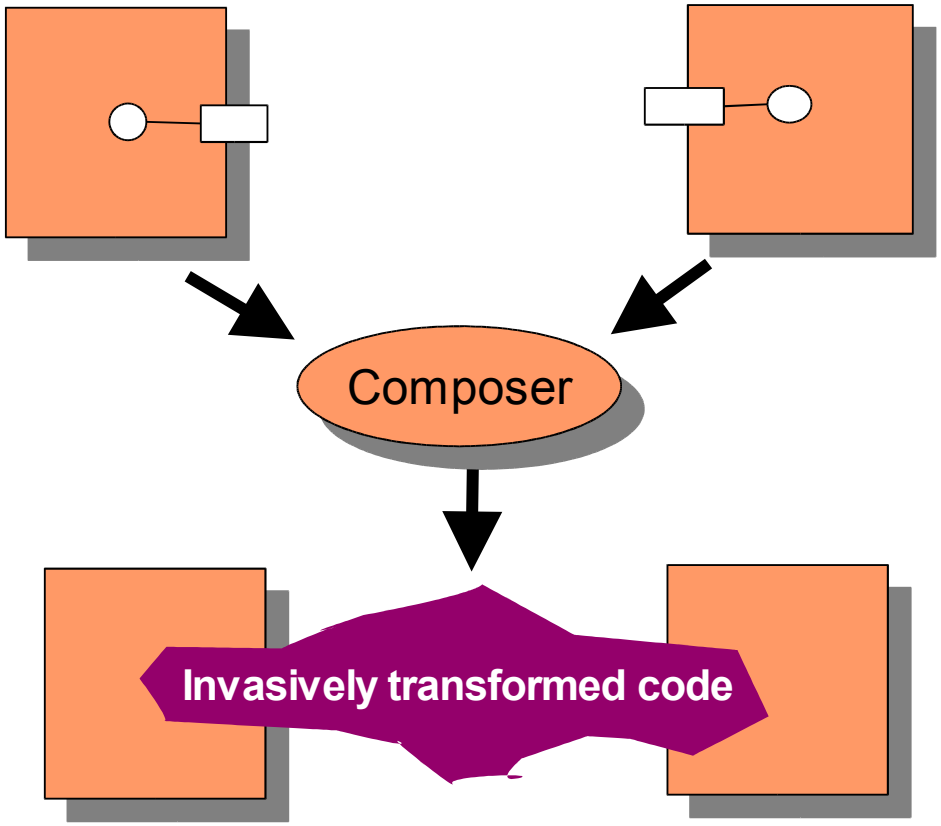
The Composition Technique of Invasive Composition

Invasive Composition
adapts and extends
components
at hooks
by transformation

A composer transforms unbound to bound hooks

***composer*: box with hooks --> box with tags**

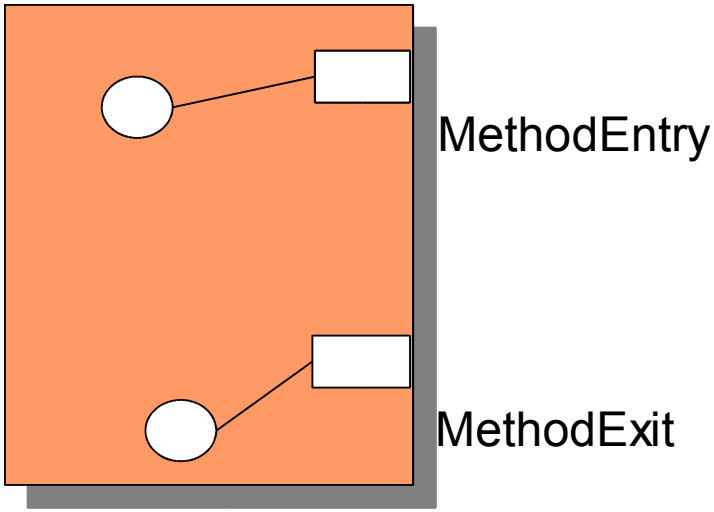
The Composition Technique of Invasive Composition

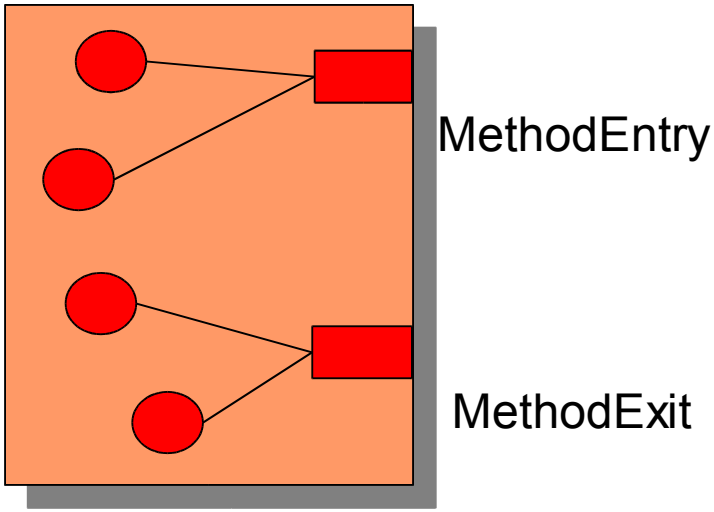


Static Metaprogram

Transformer Generator

Uniform for declared and implicit hooks






```
m (){

    abc..
    cde..

}
```

```
m (){
    print("enter m");
    abc..
    cde..
    print("exit m");
}
```

```
component.findHook(„MethodEntry“).extend(“print(\”enter m\”);”);

component.findHook(„MethodExit“).extend(“print(\”exit m\”);”);
```

The Composition Language of Invasive Composition

- **For combination of the basic composition operations**
- ***Composition programs result***
- **Using standard languages**
  - XML itself
  - Java
- **Enables us to describe large systems**

Composition program size	1
System size	10

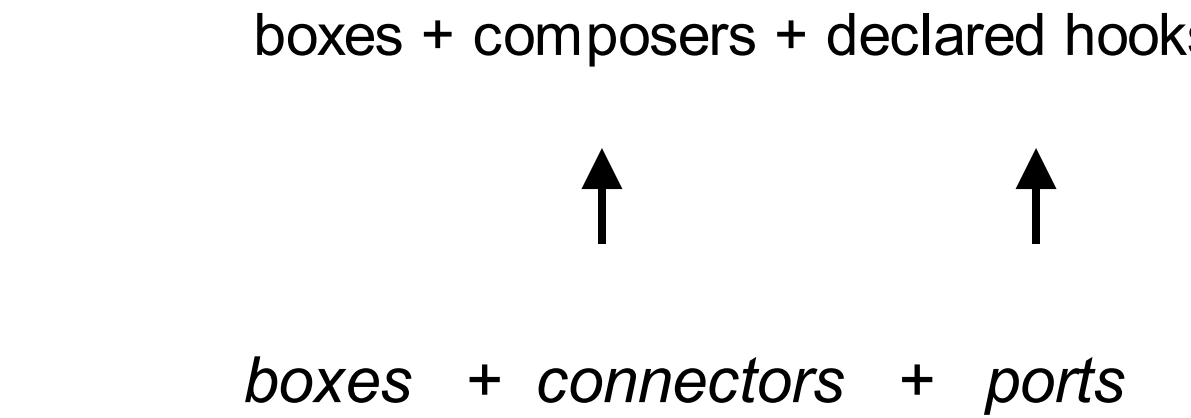
What Can You Do With Invasive Composition?

Atomic and Compound Composition Operators

- bind hook (parameterization)
  - generalized generic program elements
- rename component, rename hook
- remove value from hook (unbind)
- extend
  - extend in different semantic versions
  - Inheritance
  - view-based programming
  - intrusive data functors
- connect (bind hook 1 and 2)
- distribute
  - aspect weaving

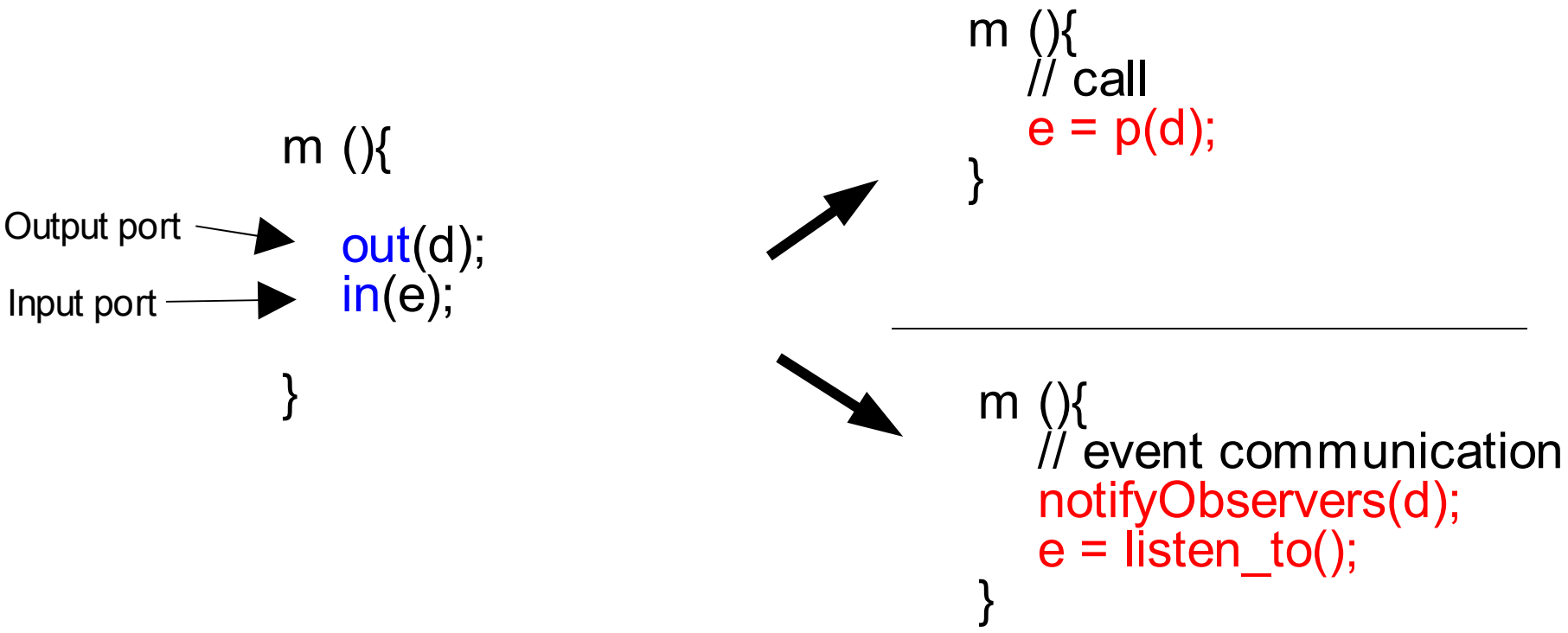
Basic Composition Algebra

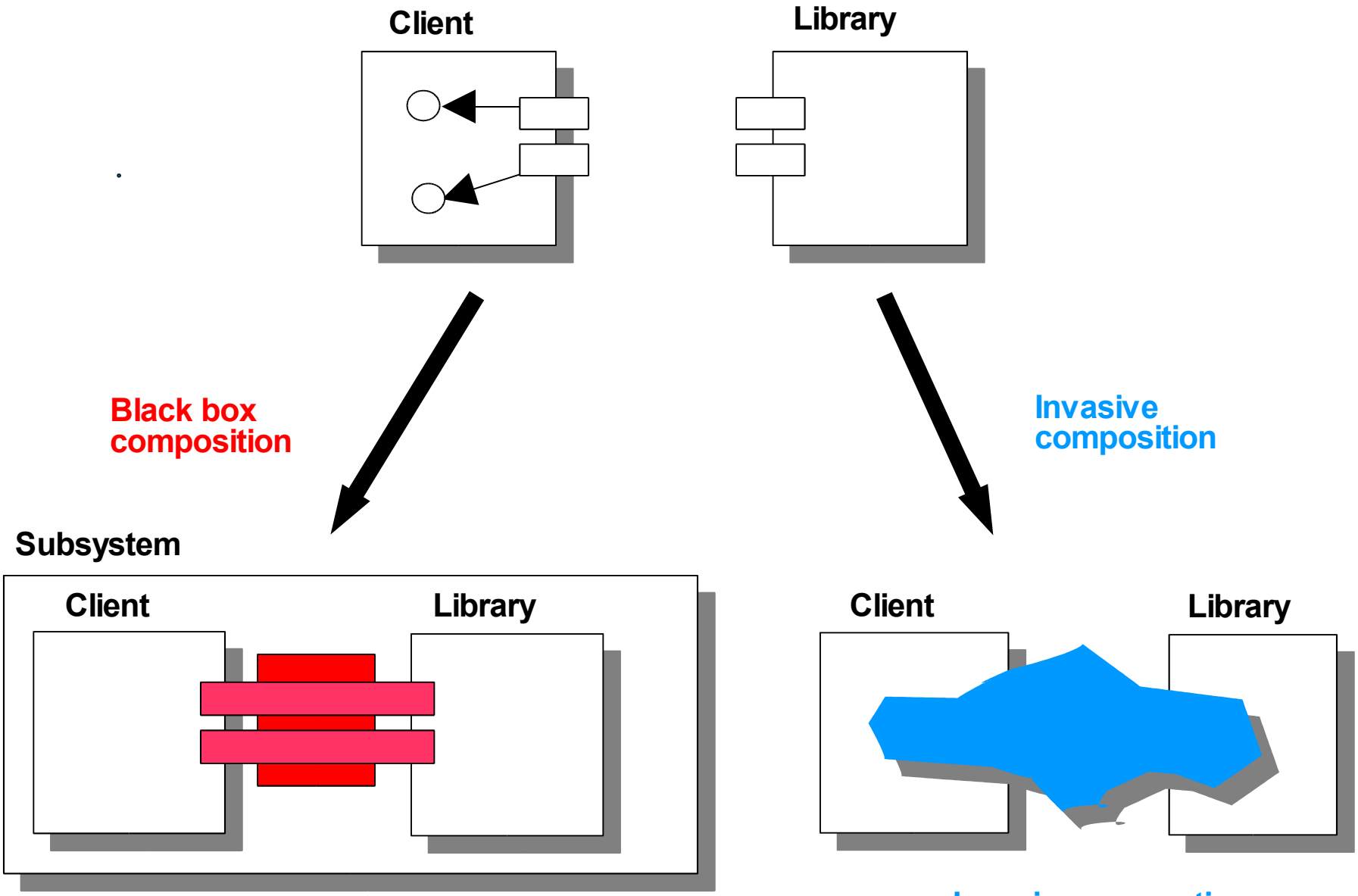
Composers Generalize Connectors (ADL Component Model)



Hooks for Communications (Ports)

Can be declared by calls to standard methods (as in Linda)

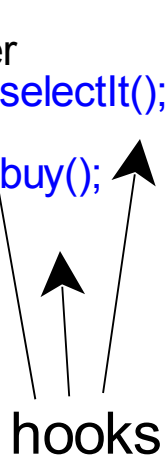



Client
Library
Black box composition
Invasive composition
Subsystem
Client
Library
Client
Library
Black box connection with glue code
Invasive connection

[TOOLS 2000]

Invasive Connection with CORBA

```
import Library;

public class Client {
  public order(String serverName)
  {
    // Get the seller
    Library library
              = getLibrary();

    // Order
    library.selectIt();

    library.buy();
  }
}
```

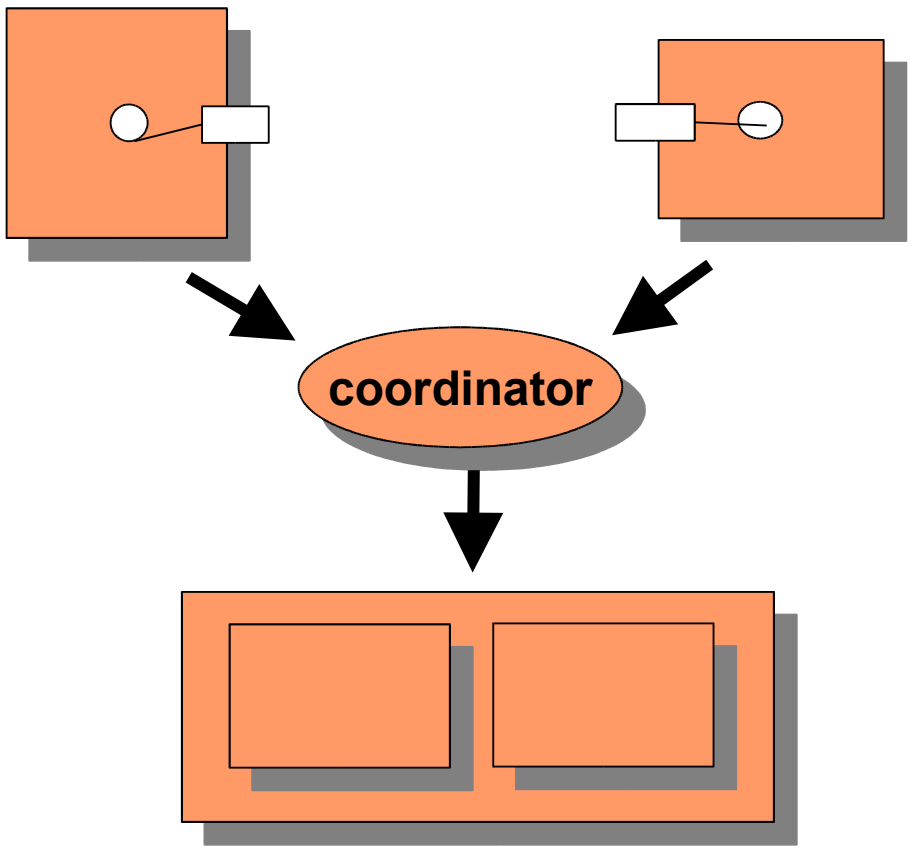


hooks

```
import org.omg.CORBA.*
import Library;

public class Client extends CORBA.client {
  public order(String name) {
    // Initialize CORBA Broker
    ORB orb = ORB.init(args,new Properties());
    // Get the seller
   Library farAway =
          orb.string_to_object(name);

    // Order
    farAway.schaueNach();
    farAway.order();
  }
}
```

[TOOLS 2000]

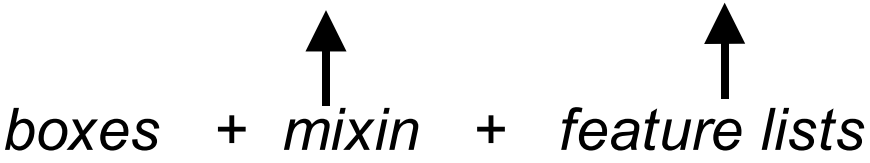
Composers Can Be Used For Skeletons (Coordinator)



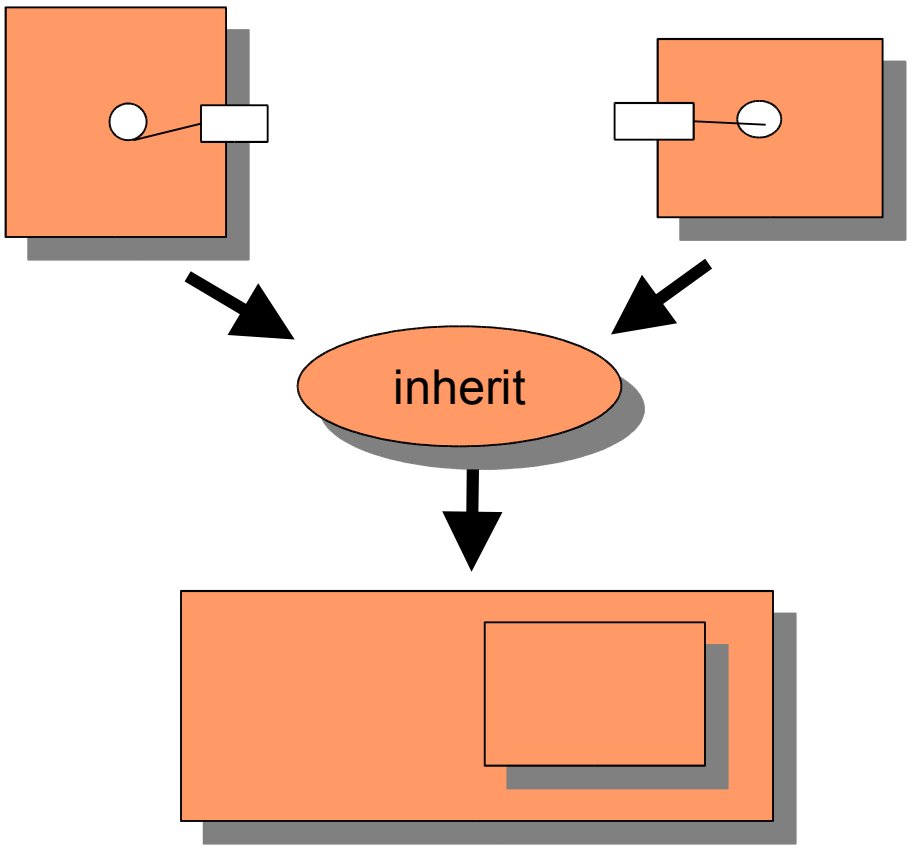
- Instead of functions or modules, skeletons can be defined over fragment components
- CoSy coordination schemes (ACE compiler component framework www.ace.nl)
  - Compose basic components with coordinating operators

Composers Generalize Inheritance Operators (Classes as Components)

boxes + composers + declared hooks

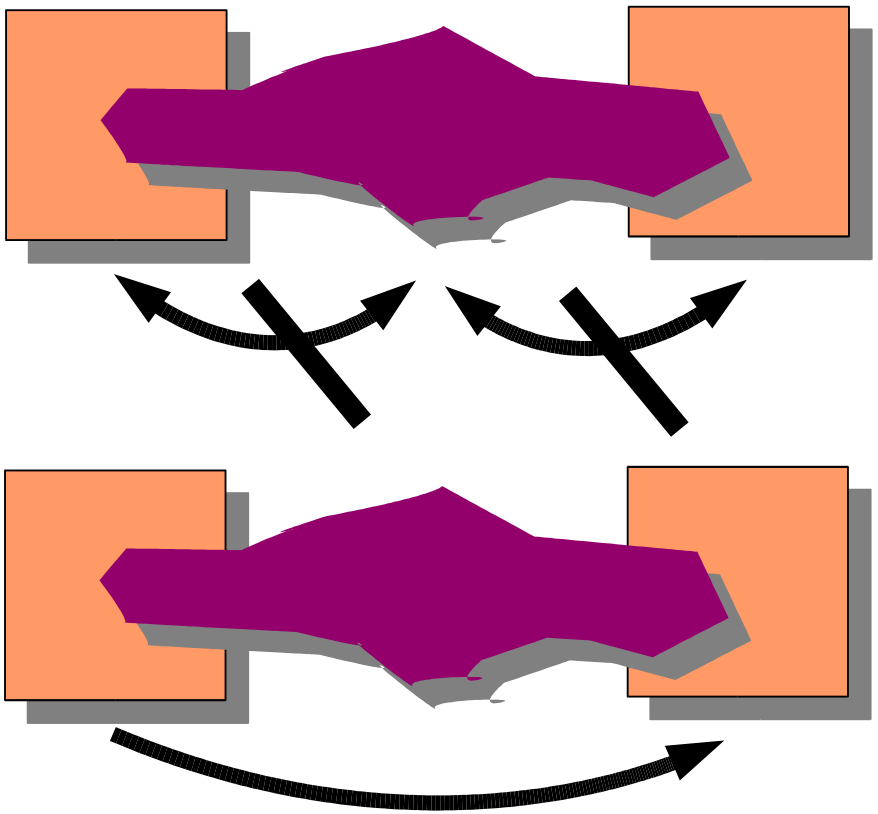


Composers Can Be Used For Inheritance



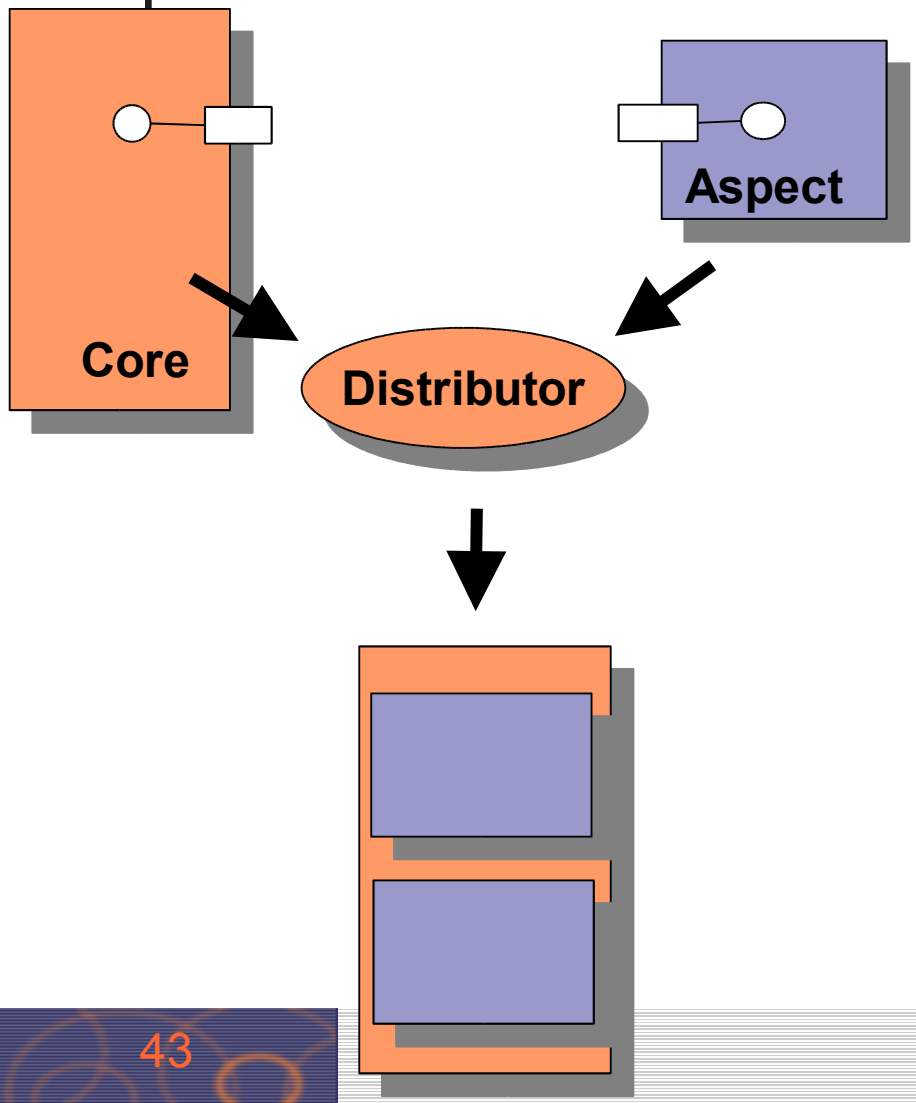
- **Extension can be used for inheritance (mixins)**
- **inheritance :=**
  - copy first super document
  - extend with second super document

Sound Extensions (Views That Do Not Destroy Contracts)



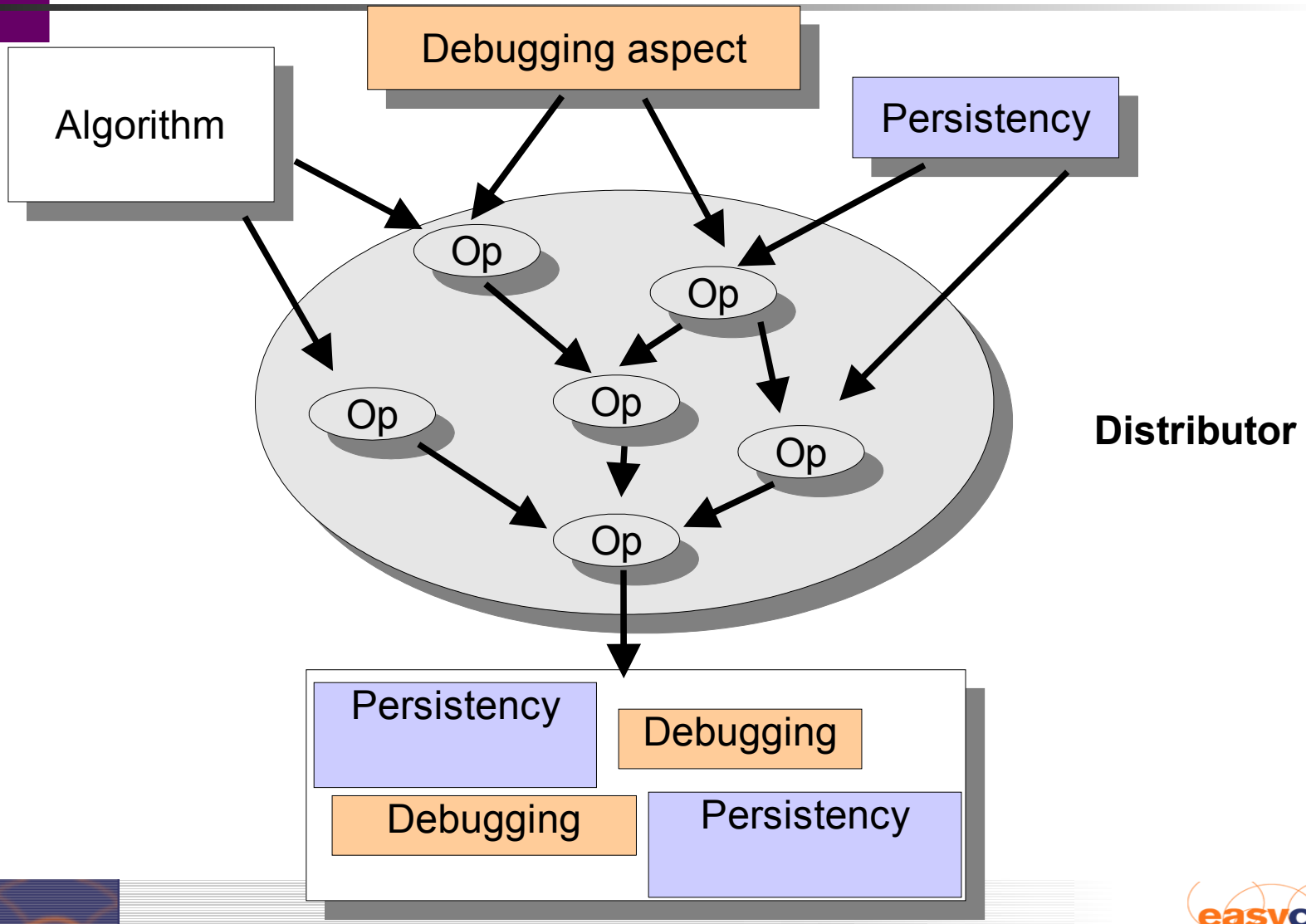
- Invasive Composition works if dependencies are
  - Absent
  - Forward flow
- Core components don't change
- Can be checked with slicing or analysis, or regression testing

Composers can be Used for AOP (Core and Aspect Components)

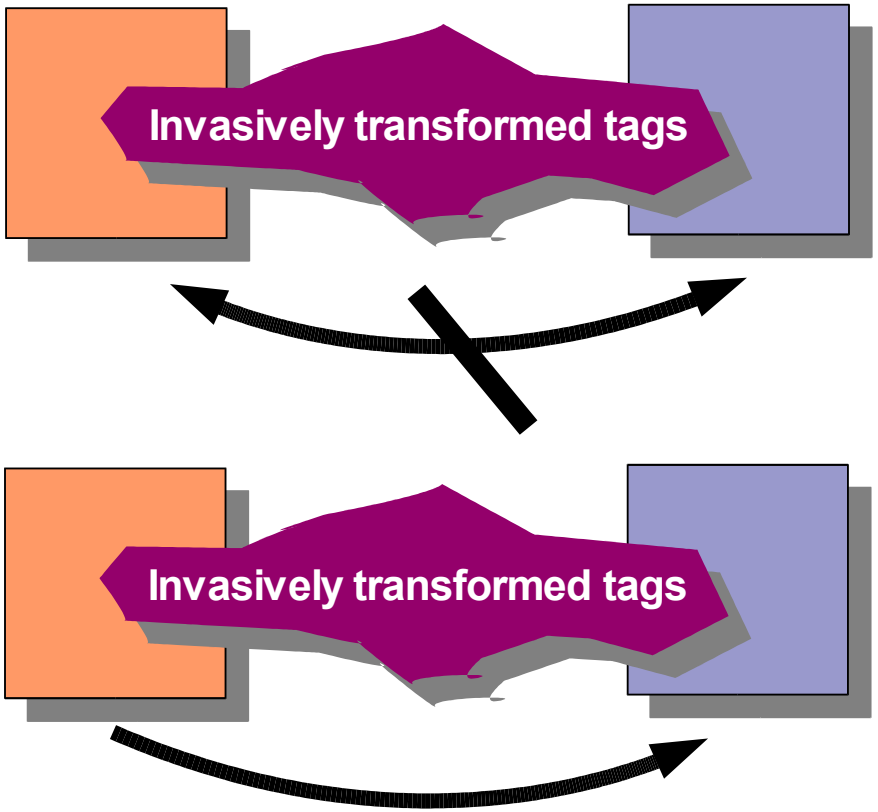


- Complex composers distribute aspect fragments over core fragments
- *Distributors* extend the core
- Distributors are more complex operators, defined from basic ones

Weavers As Distributors



Sound Aspects (Aspects That Do Not Destroy Contracts)



- Invasive Aspect Weaving works if dependencies are
  - Absent
  - Forward flow
- Core components don't change
- Can be checked with slicing or analysis, or regression testing

Simple Weavers

- distributeOverMethods
  - Weave a prologue and an epilogue into a class or package tree
  - implemented as a navigator over the tree
  - applies simple hook extensions on entry and exit hook

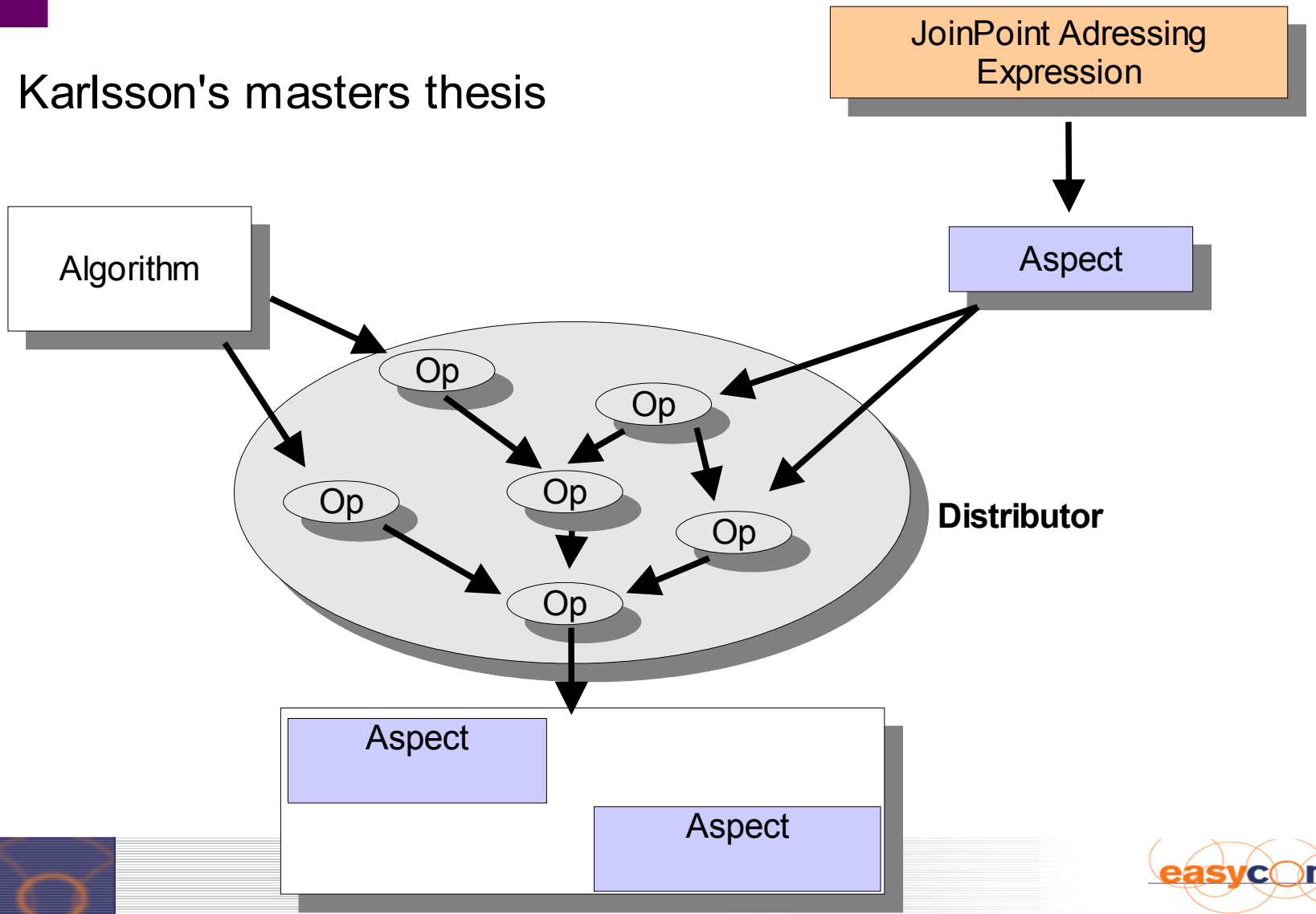
- Hungarian aspect boxes
  - Carry an aspect with Hungarian notation
  - Weavers weave with naming conventions

A Simple Weaver

```
// Initialize composition system
JavaCompositionSystem cs = new JavaCompositionSystem(outputPath);

// Loading components.

// The core component
CompilationUnitBox cuToBeExtended = cs.createCompilationUnitBox("DemoClass");
// The aspect
ClassBox aspectClass = cs.createClassBox("BeforeAfterAspect.java");

// Now distribute the aspect over the core
cuToBeExtended.distributeMethods(aspectClass);

// Export
cs.printAll();
```

Weaving with Modular Join Point Adressing

- M. Karlsson's masters thesis



The COMPOsition SysTem COMPOST

- COMPOST is the first system to support invasive composition for Java
  - Library of static meta-programs
  - Composition language Java
  - Reifies concepts Boxes, Hooks, Composers
- and many other things

COMPOST for Everybody

- 0.78 is out (Uni Karlsruhe/Uni Linköping)
  - http://www.the-compost-system.org
  - We expect a new major version in April 2004
- Contains refactoring engine RECODER as transformation subsystem
  - http://recoder.sf.net
- Invasive Software Composition, U. Aßmann, Springer.
- Developed within the EASYCOMP project
  - EU FET Basic Research “Easy Composition in Future Generation Component Systems”
  - New component models for XML, COTS, runtime components (Uniform composition)
- We are refactoring towards a uniform XML version

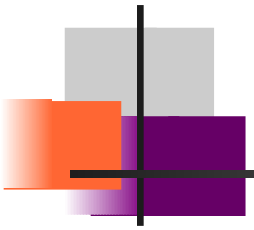
Invasive Software Composition as Composition Technique

Invasive Composition: Component Model

- Graybox components instead of black box ones
  - Composition interfaces with declared hooks
  - Implicit composition interfaces with implicit hooks
  - The composition programs produce the functional interfaces
    - Resulting in efficient systems, because superfluous functional interfaces are removed from the system
  - Content: source code
    - binary components also possible, poorer metamodel
- Aspects are just a new type of component
- Fragment-based Parameterisation a la BETA slots
  - Type-safe parameterization on all kinds of fragments

Invasive Composition: Composition Technique

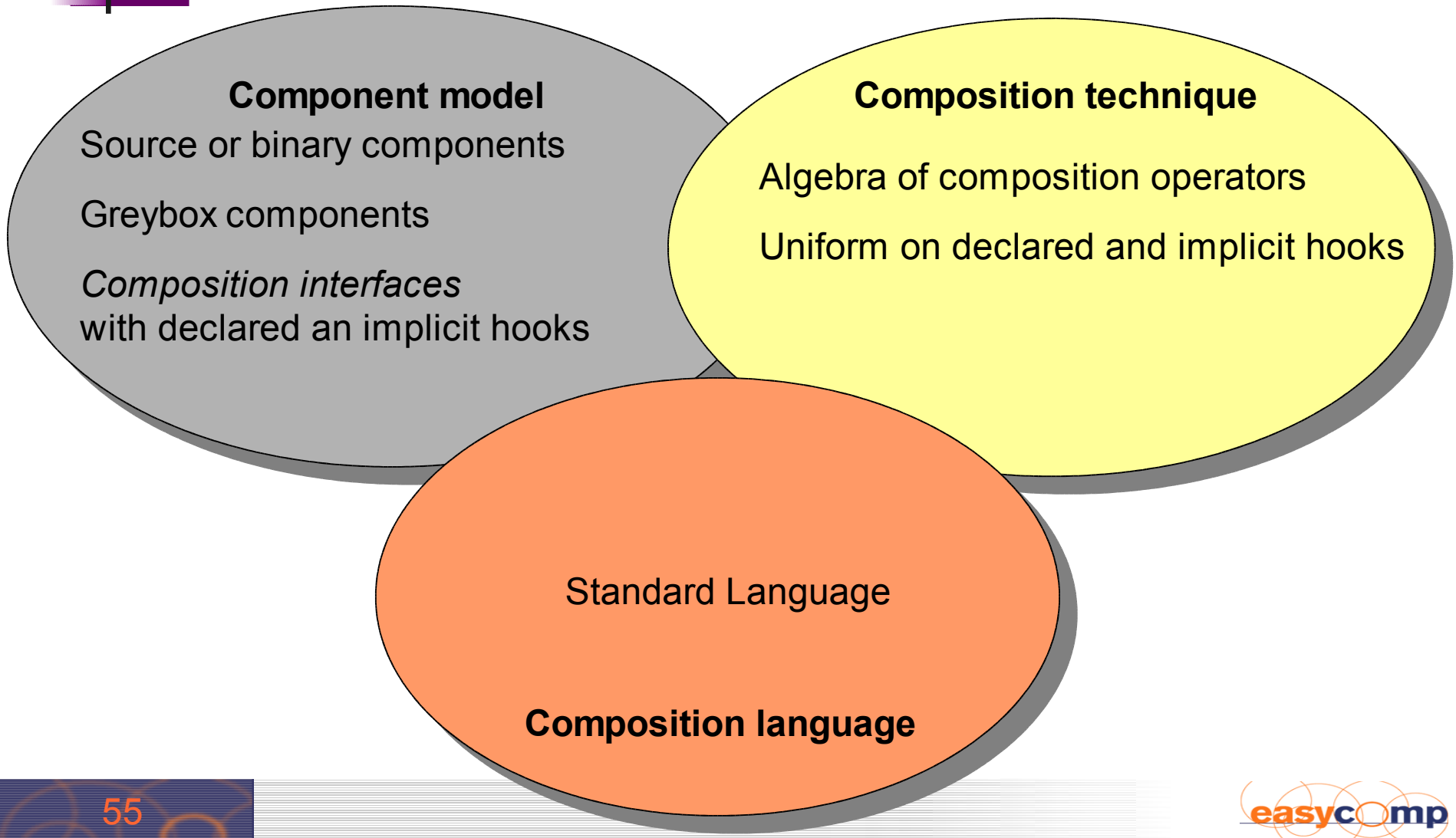
- Adaptation and glue code: good, composers are program transformers and generators
- Aspect weaving
  - Parties may write their own weavers
  - No special languages
- Extensions:
  - Hooks can be extended
  - Soundness criteria of lambdaN still apply
  - Metamodelling employed
- Not yet scalable to run time



Composition Language

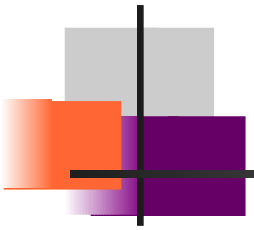
- Various languages can be used
- Product quality improved by metamodel-based typing of compositions
- Metacomposition possible
  - Architectures can be described in a standard object-oriented language and reused
- An *assembler* for composition
  - Other, more adequate composition languages can be compiled

Invasive Composition as Composition System



Unification of Development Techniques

- With the uniform treatment of declared and implicit hooks, several technologies can be unified:
  - Generic programming
  - Inheritance-based programming
  - Connector-based programming
  - View-based programming
  - Aspect-based programming



Conclusions for ISC

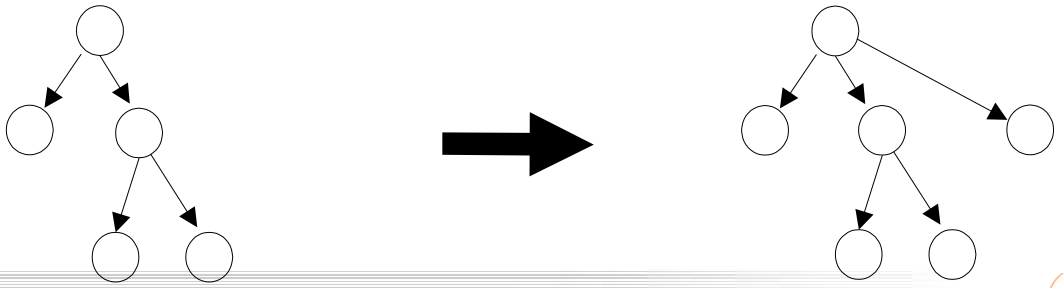
- Fragment-based composition technology
  - Graybox components
  - Producing tightly integrated systems
- Components have *composition interface*
  - From the composition interface, the functional interface is derived
  - Composition interface is different from functional interface
  - Overlaying of classes (role model composition)

Different Forms of Greyboxes

Refactoring as Whitebox Operation

- Refactoring works directly on the AST/ASG
- Attaching/removing/replacing fragments
- Whitebox reuse

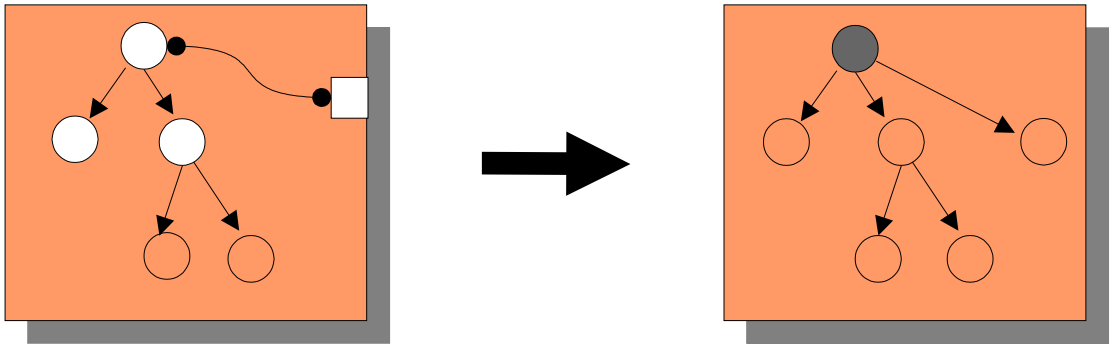
Refactorings
Transformations



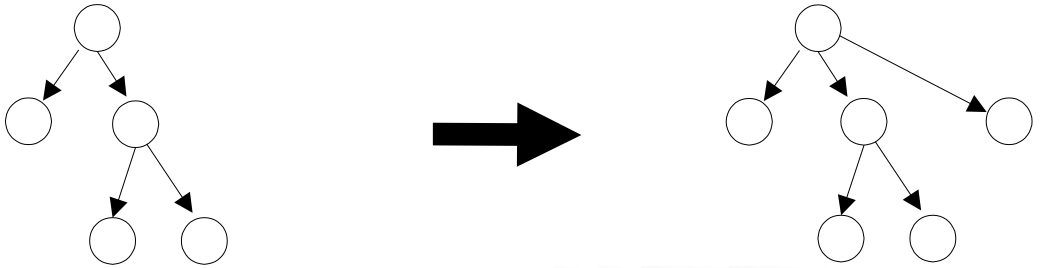
Weaving as Light-Grey Operation

- Aspect weaving and view composition works on implicit hooks *(join points)*
- *Implicit composition interface*

Composition with implicit hooks



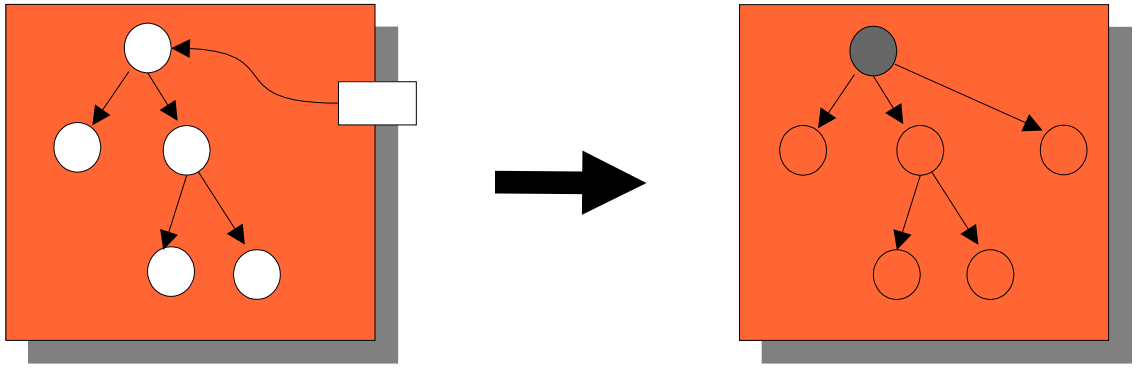
Refactorings Transformations



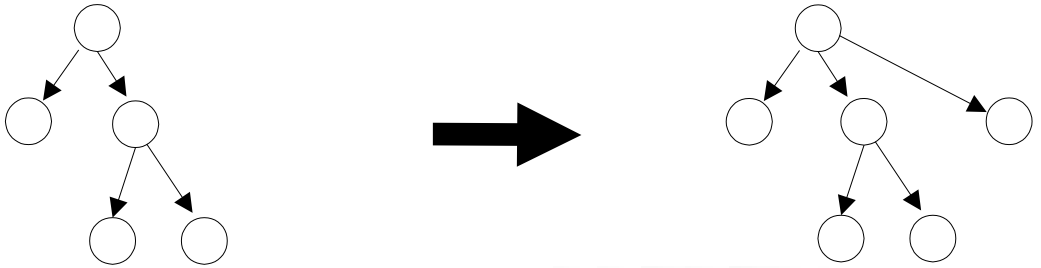
Parameterization as Darker-Grey Operation

- Templates work on *declared hooks*
- *Declared composition interface*

Composition with declared hooks

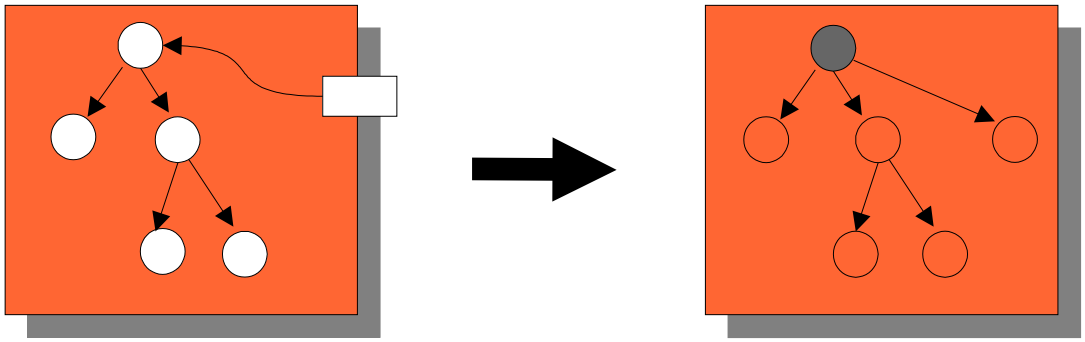


Refactorings Transformations

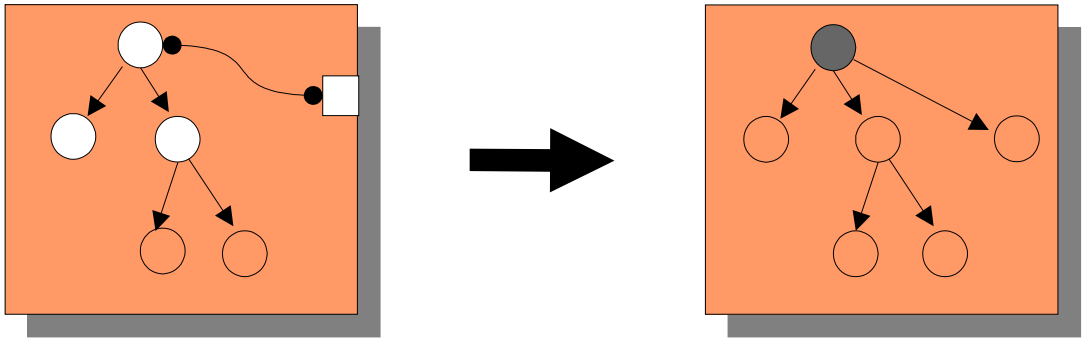


Systematization Towards Greybox Component Models

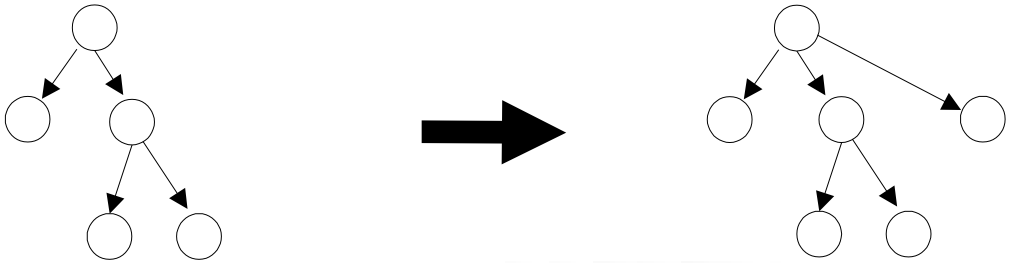
Composition with declared hooks



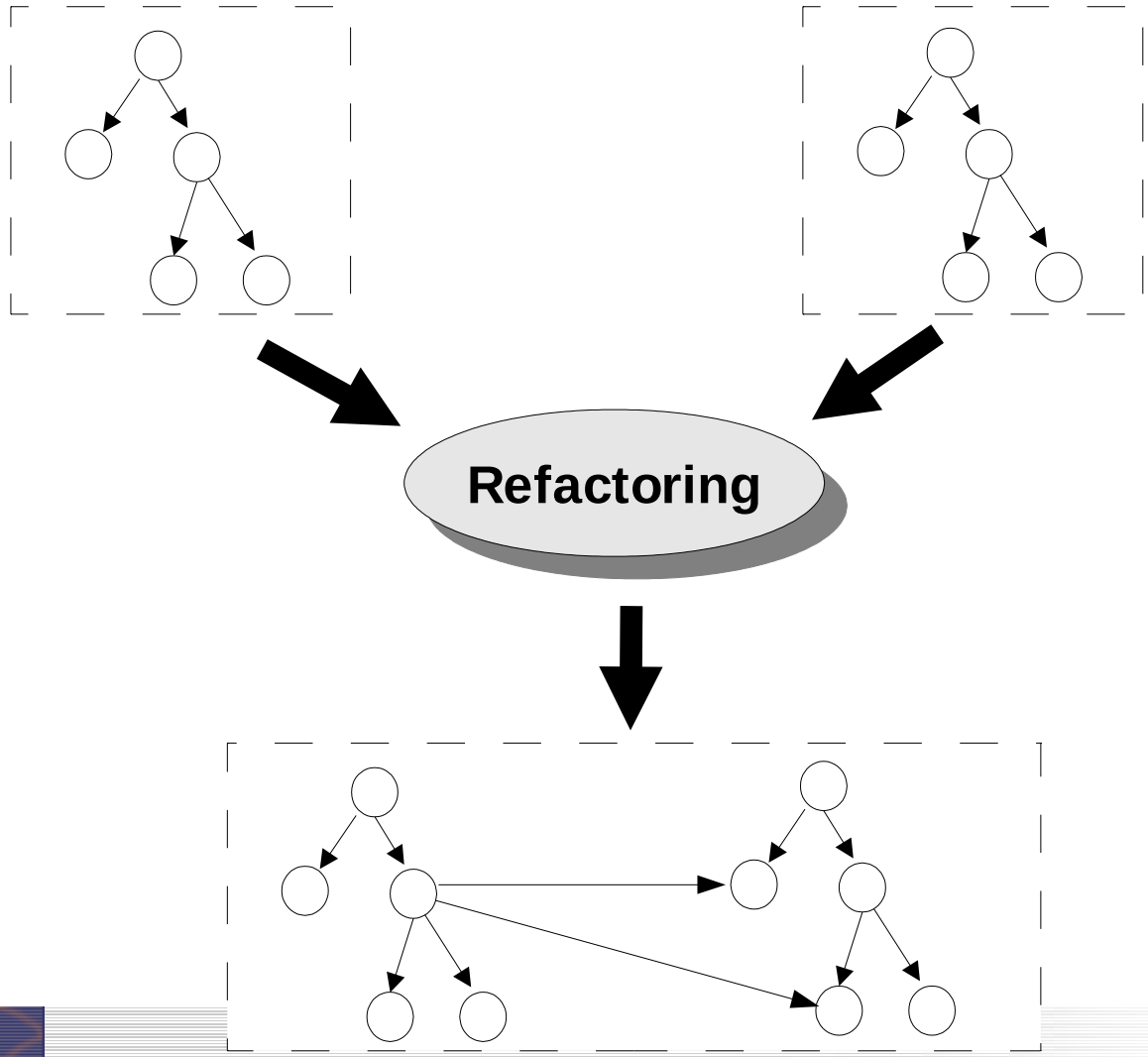
Composition with implicit hooks



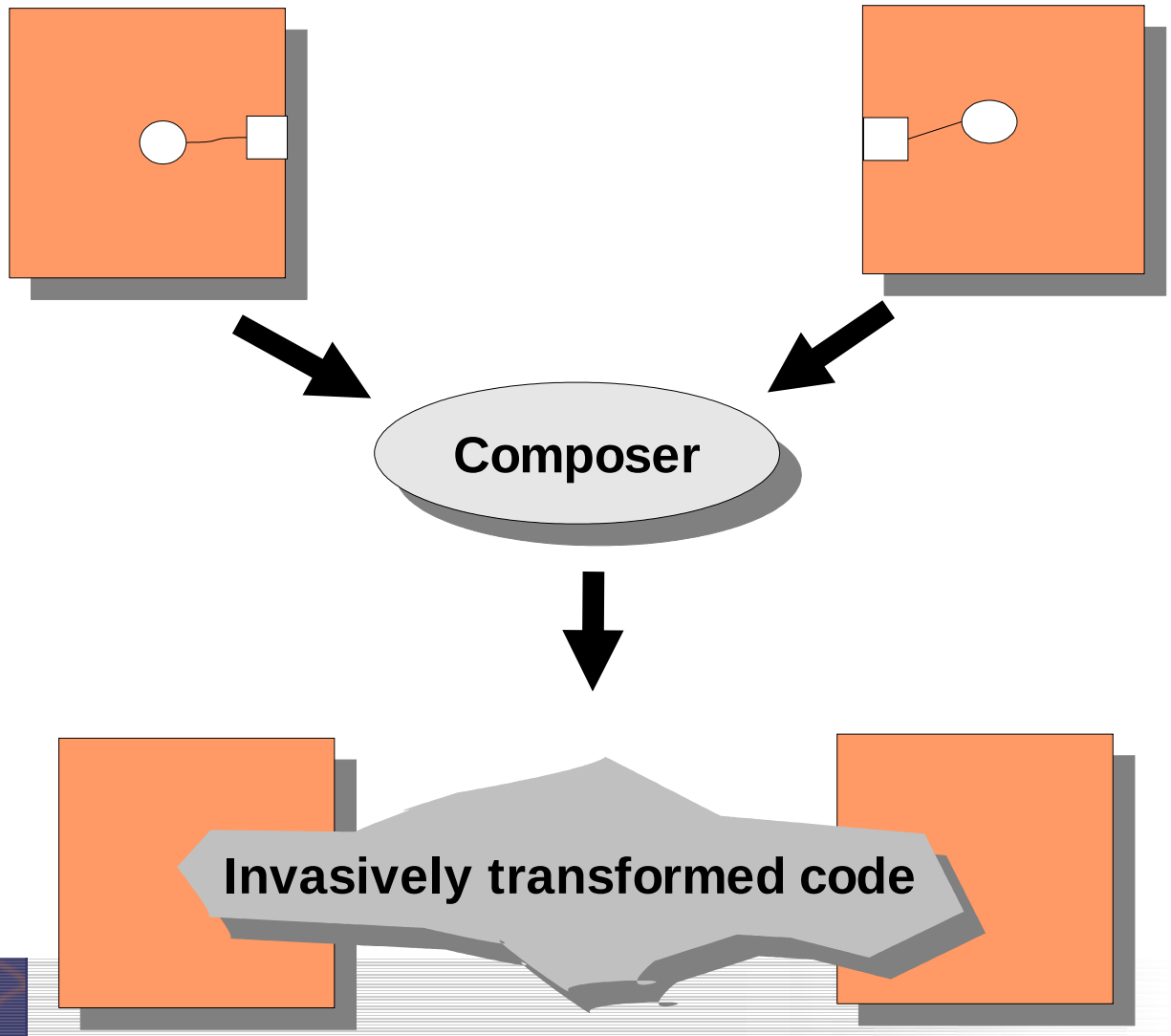
Refactorings Transformations



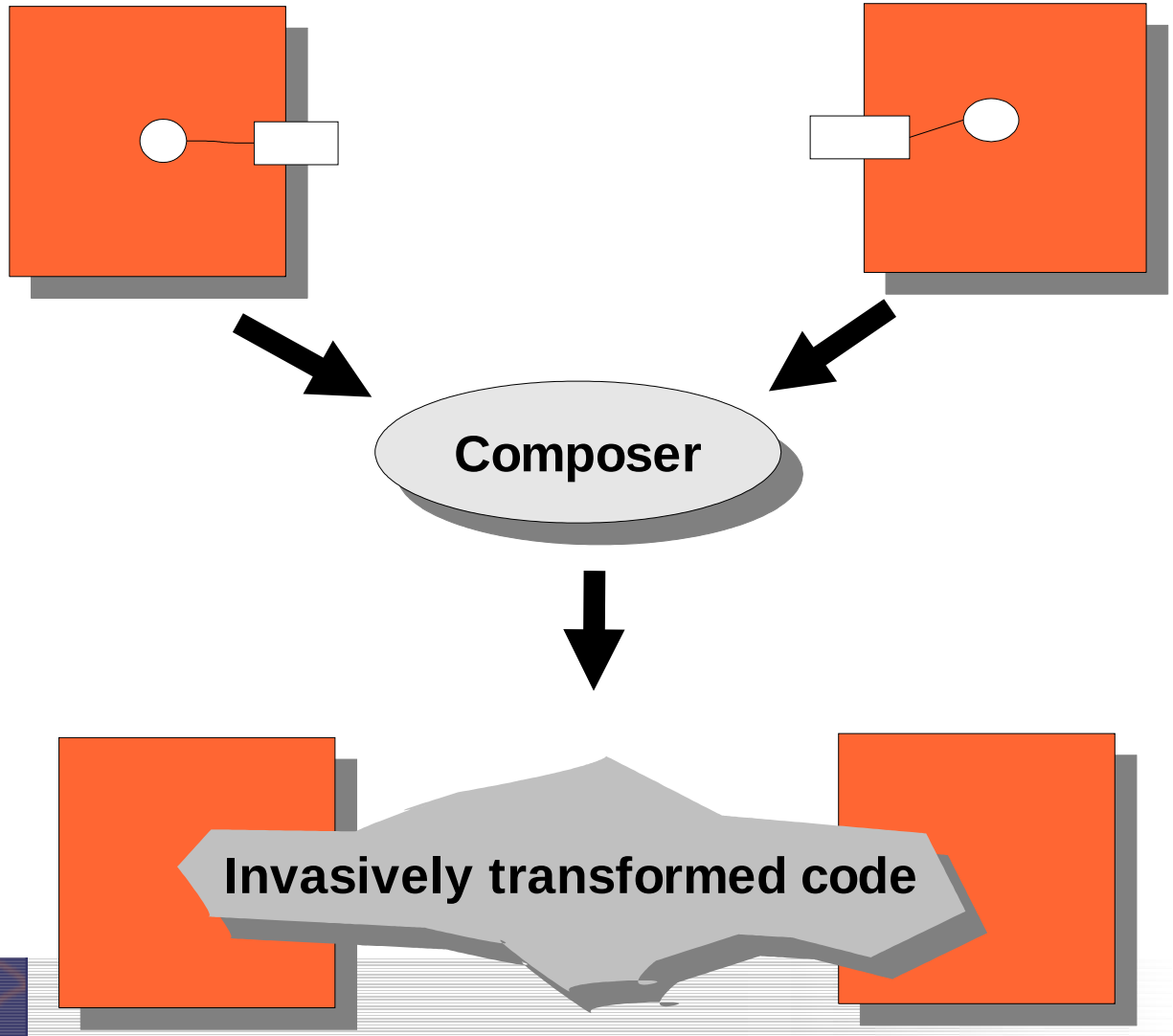
Refactoring Builds On Transformation Of Abstract Syntax

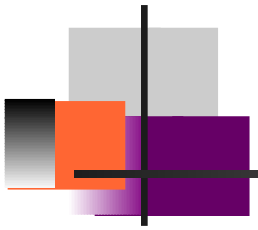


Invasive Composition Builds On Transformation Of Implicit Hooks



Invasive Composition Builds On Transformation on Declared Hooks

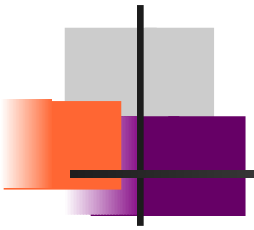




Future Composition Systems

What Is A Component?

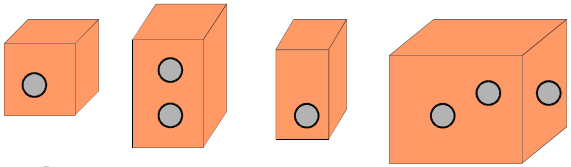
- Cannot be stated in general
  - Component models must be defined
- We must investigate composition techniques
- And languages
  - Domain-specific ones (composition-oriented composition languages)
  - General ones
- We should build frameworks for all component models
  - Generic component models
  - Generic composition technique
  - Scalability!



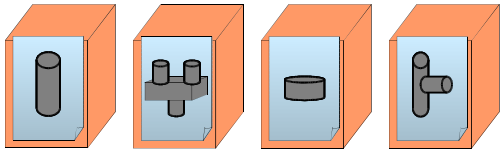
Types of Composition Systems

- **Software Composition Systems**
  - Blackbox Composition Systems
  - Graybox Composition Systems (Integrational Systems)
  - Turing-complete composition languages
  - [Invasive Software Composition, Aßmann, Springer 2003]
- **Uniform Composition Systems**
  - Supporting multiple languages
  - Supporting XML
  - Active documents
  - Uniform treatment of software and data
  - Based on software composition systems

Integrational Software Engineering

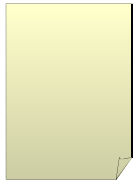


Components



Invasive composition

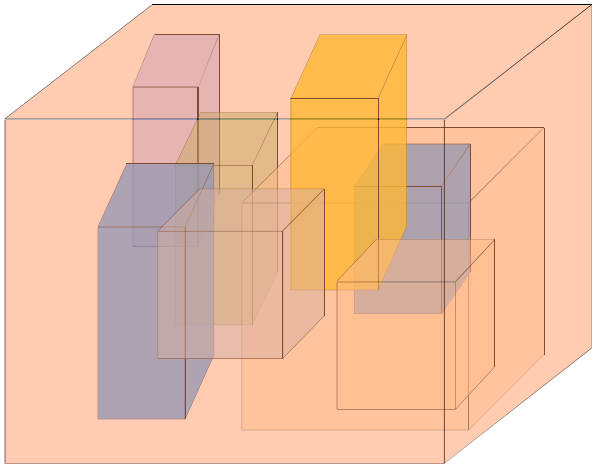
operations



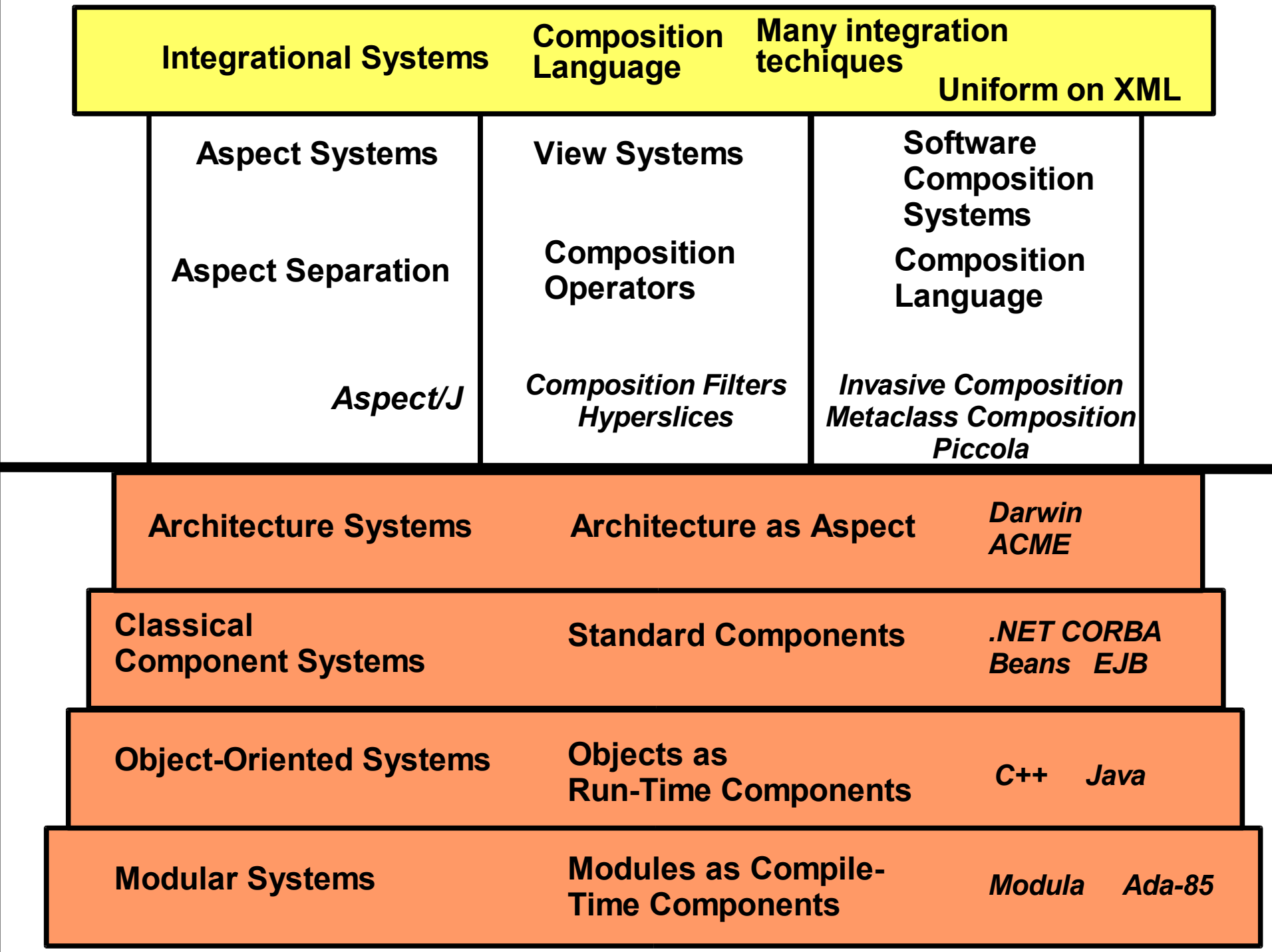
Composition recipe

Software composition

Uniform composition



Integrated system


Integrational Systems
Composition Language
Many integration techiques
Uniform on XML
Aspect Systems
Aspect Separation
Aspect/J
View Systems
Composition Operators
Composition Filters
Hyperslices
Software Composition Systems
Composition Language
Invasive Composition
Metaclass Composition
Piccola
Architecture Systems
Architecture as Aspect
Darwin
ACME
Classical Component Systems
Standard Components
.NET CORBA
Beans EJB
Object-Oriented Systems
Objects as Run-Time Components
C++ Java
Modular Systems
Modules as Compile-Time Components
Modula Ada-85

The End

- http://www.easycomp.org
- http://www.the-compost-system.org
- http://recoder.sf.net
- http://injectj.fzi.de

- Invasive Software Composition, U. Aßmann, Springer.