

FAC

Fractal Aspect Component

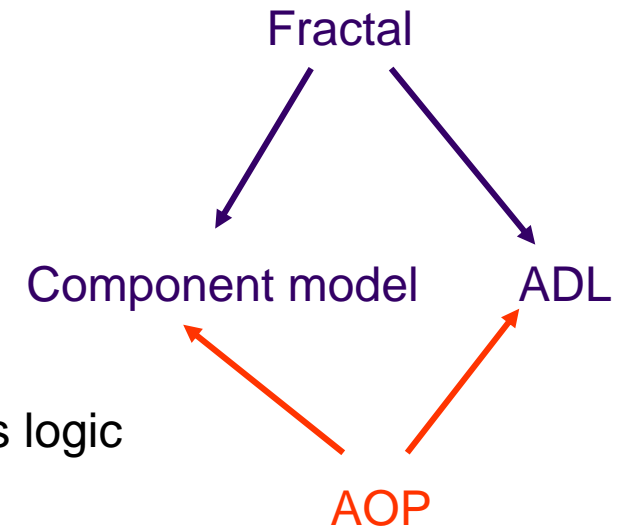
N. Pessemier, L. Seinturier, L. Duchien
T. Coupaye

INRIA Futurs, USTL-LIFL,
Jacquard project



FAC project & context

- Get 2 development styles closer:
 - Component assembly:
 - Packages software entities in modules
 - Increases reusability
 - Architecture Description Language (ADL):
 - Clarifies software entities interactions
 - Fractal component model
 - Aspect-Oriented Programming (AOP):
 - Separation of Concerns (SoC)
 - Separates technical services from business logic
 - Modularizes crosscutting concerns
 - JAC model
- Fractal Aspect Component:
 - enhance Fractal component model and Fractal-ADL with AOP principles

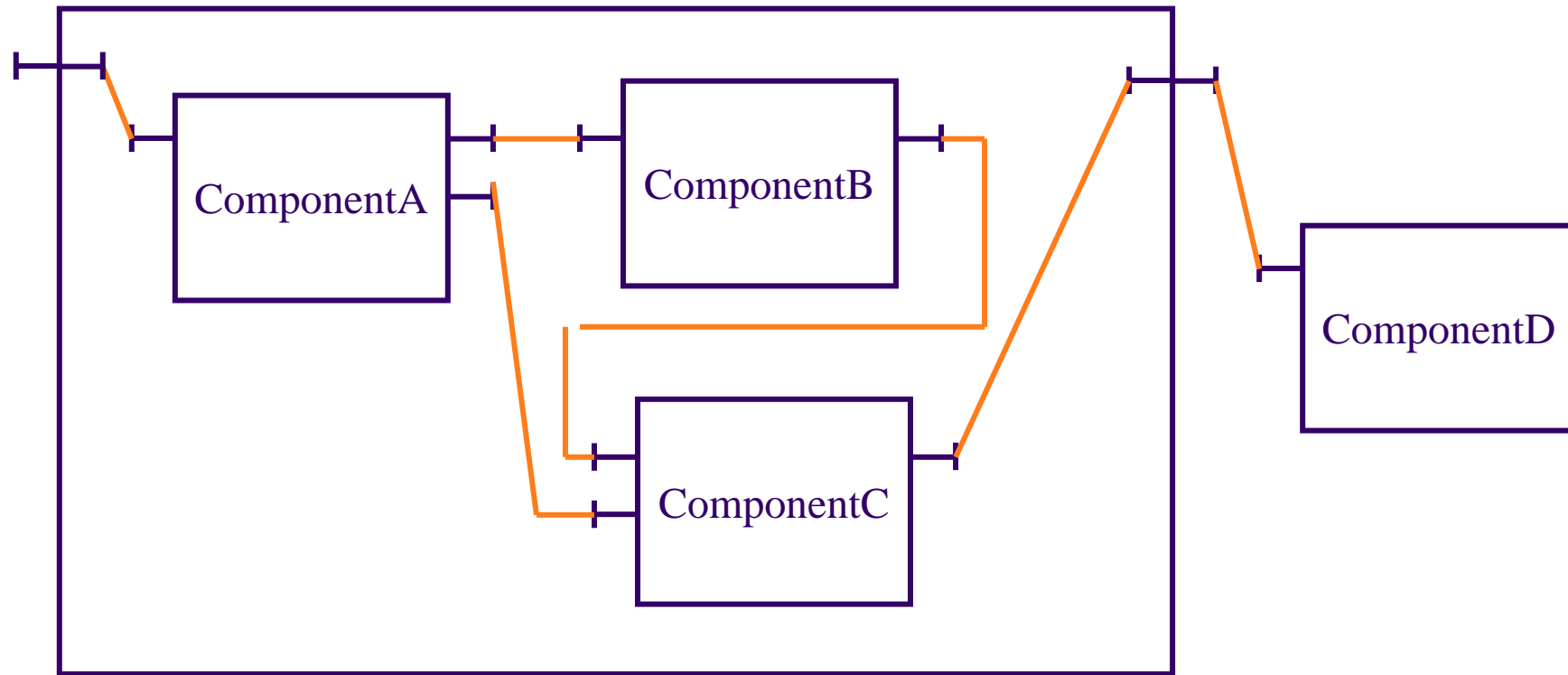


- Fractal Crosscutting Issues
- FAC Abstract Model
- FAC Concrete Model
- Tools:
 - Fractal-ADL extension
 - Fractal Explorer extension
- Conclusion & Future Work

Fractal Crosscutting Issues



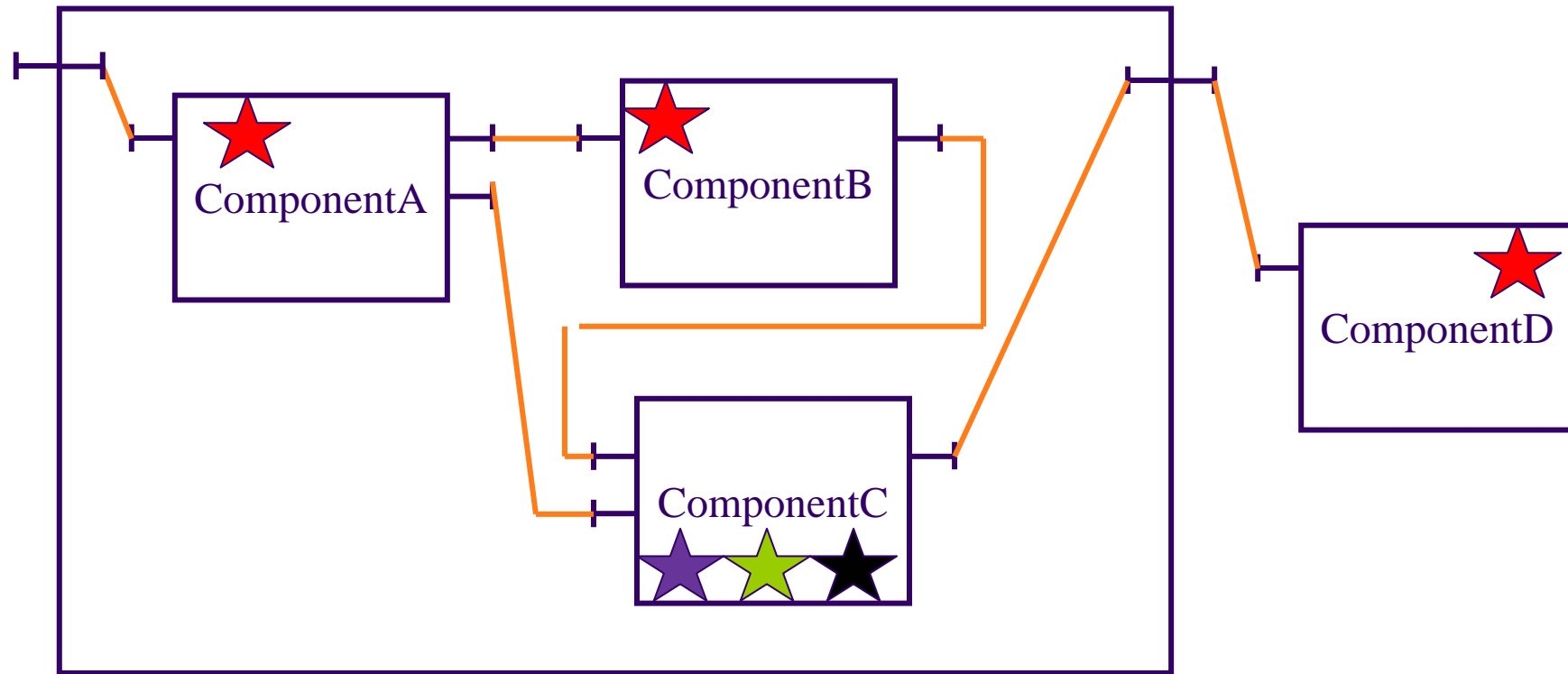
Crosscutting issues (1/3)



Crosscutting issues (2/3)

★ Code scattering

★ ★ ★ Code tangling

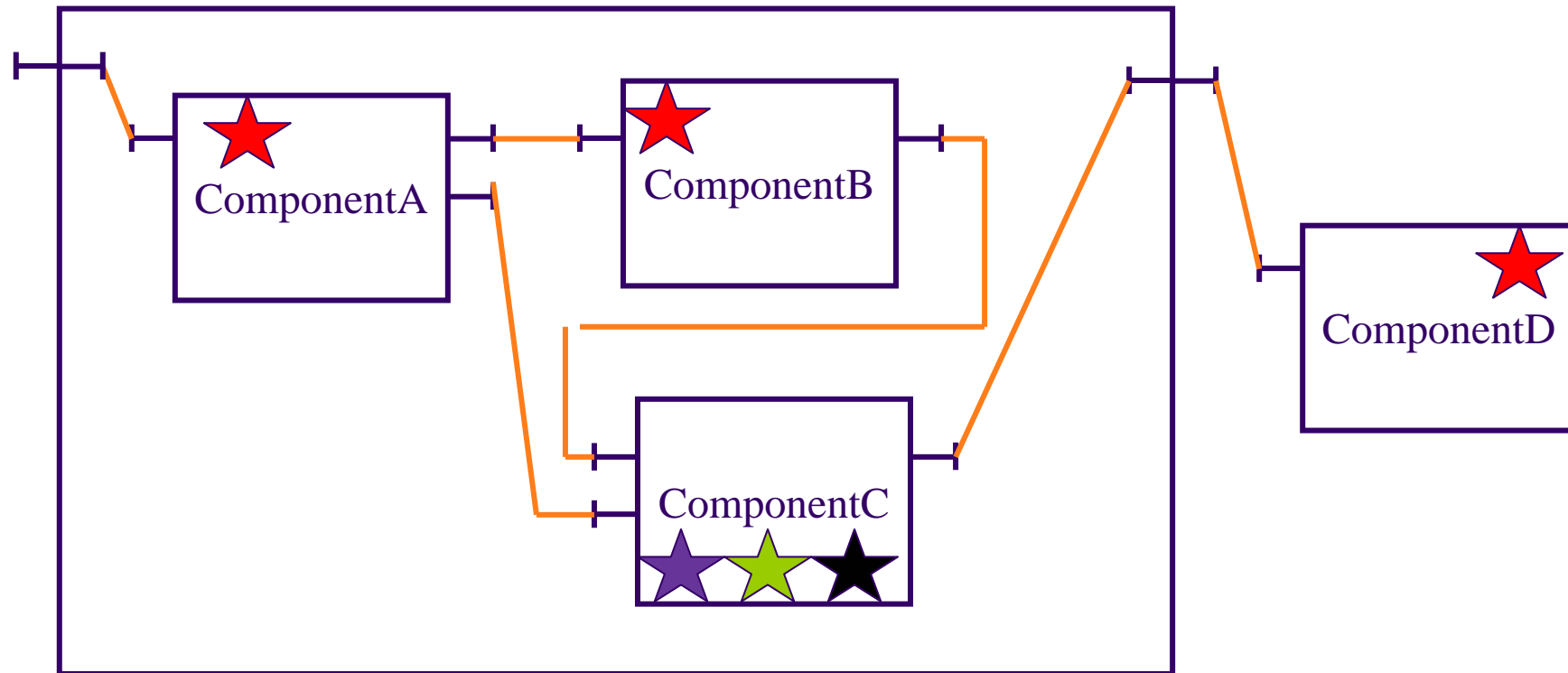


Crosscutting issues (3/3)

★ Code scattering

★ ★ ★ Code tangling

Apply AOP to components ? Fractal Aspect Component



FAC Abstract Model



➤ 2 main notions:

➤ Aspect Component (AC): --> WHAT ?

- Embodies a crosscutting concern
- Is a Fractal component with a server interface that describes the behavior to apply
- Components and Aspects are components:
 - Symmetric approach

➤ Crosscutting Binding (CB): --> WHERE ?

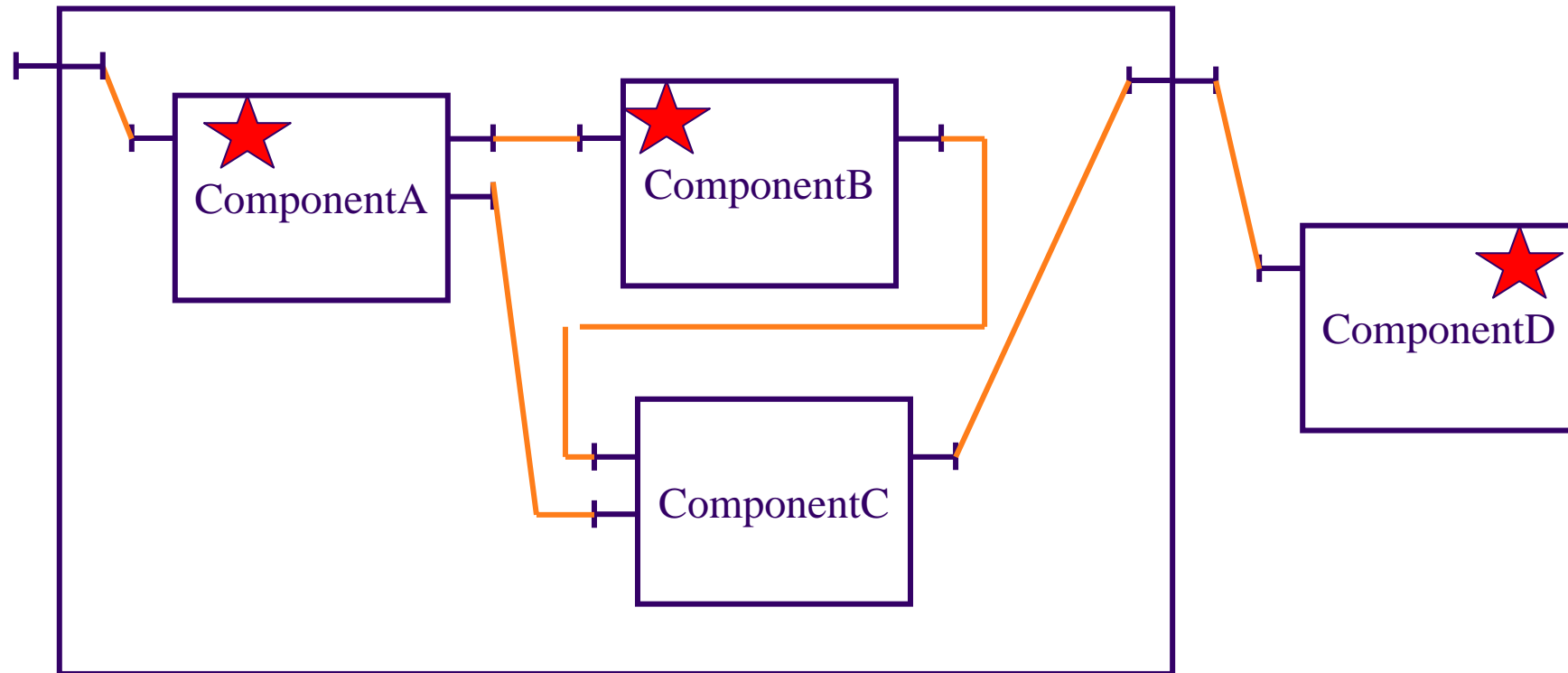
- A new kind of binding between an **aspect controller** and an **AC**
- New control interface: the **aspect controller**

Abstract model (1/4)

Server interface *AspectComponent*

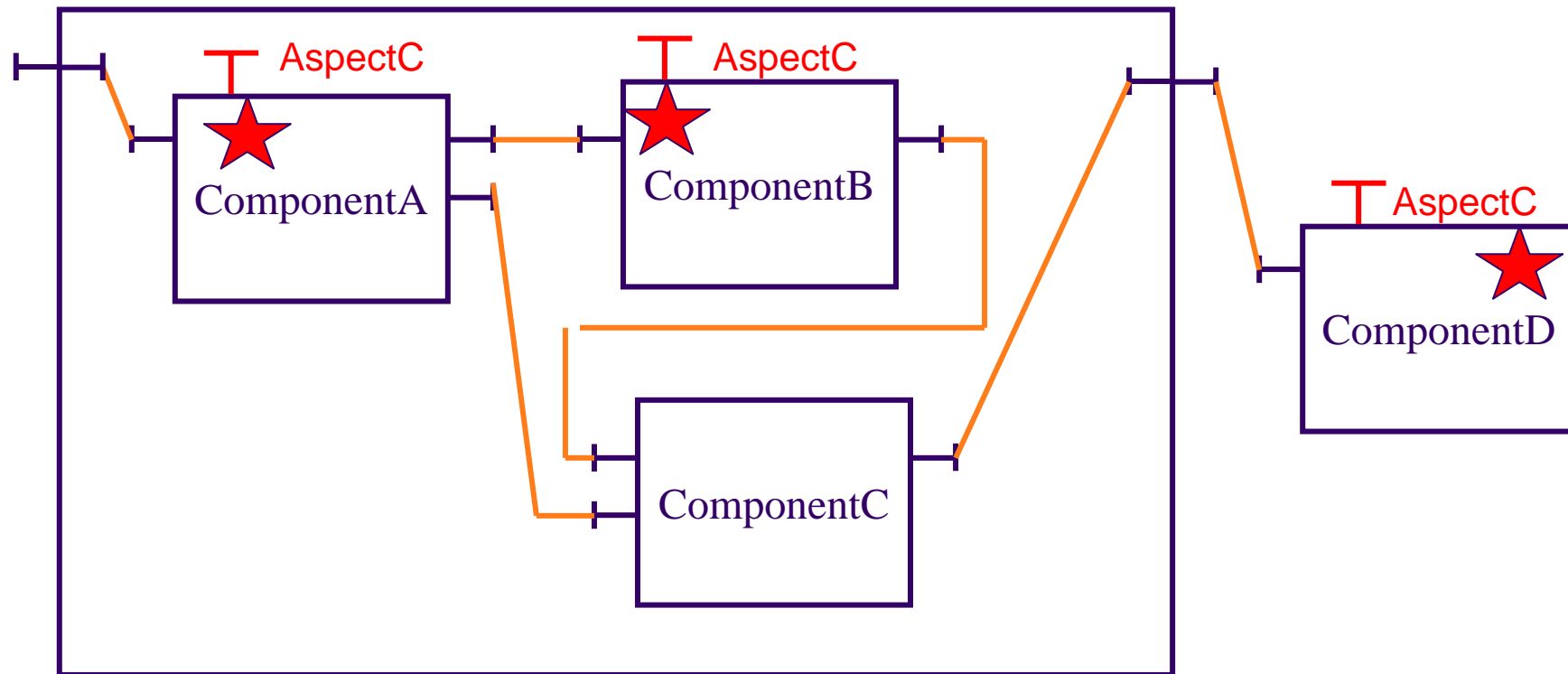
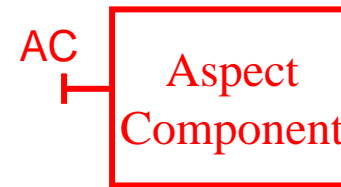
Aspect
Component

An Aspect Component (AC) embodies a crosscutting concern



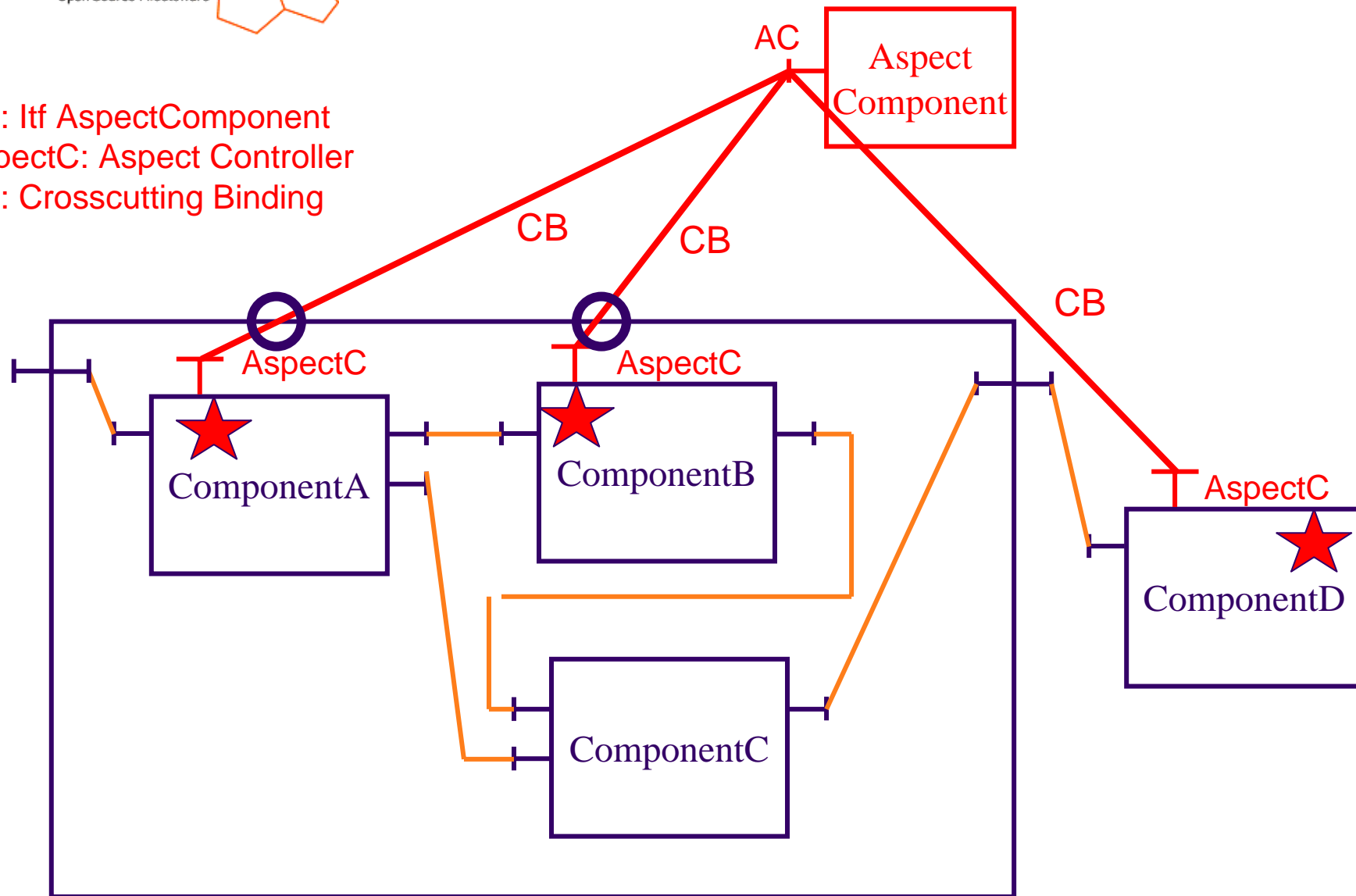
Abstract model (2/4)

AC: Itf AspectComponent
AspectC: Aspect Controller



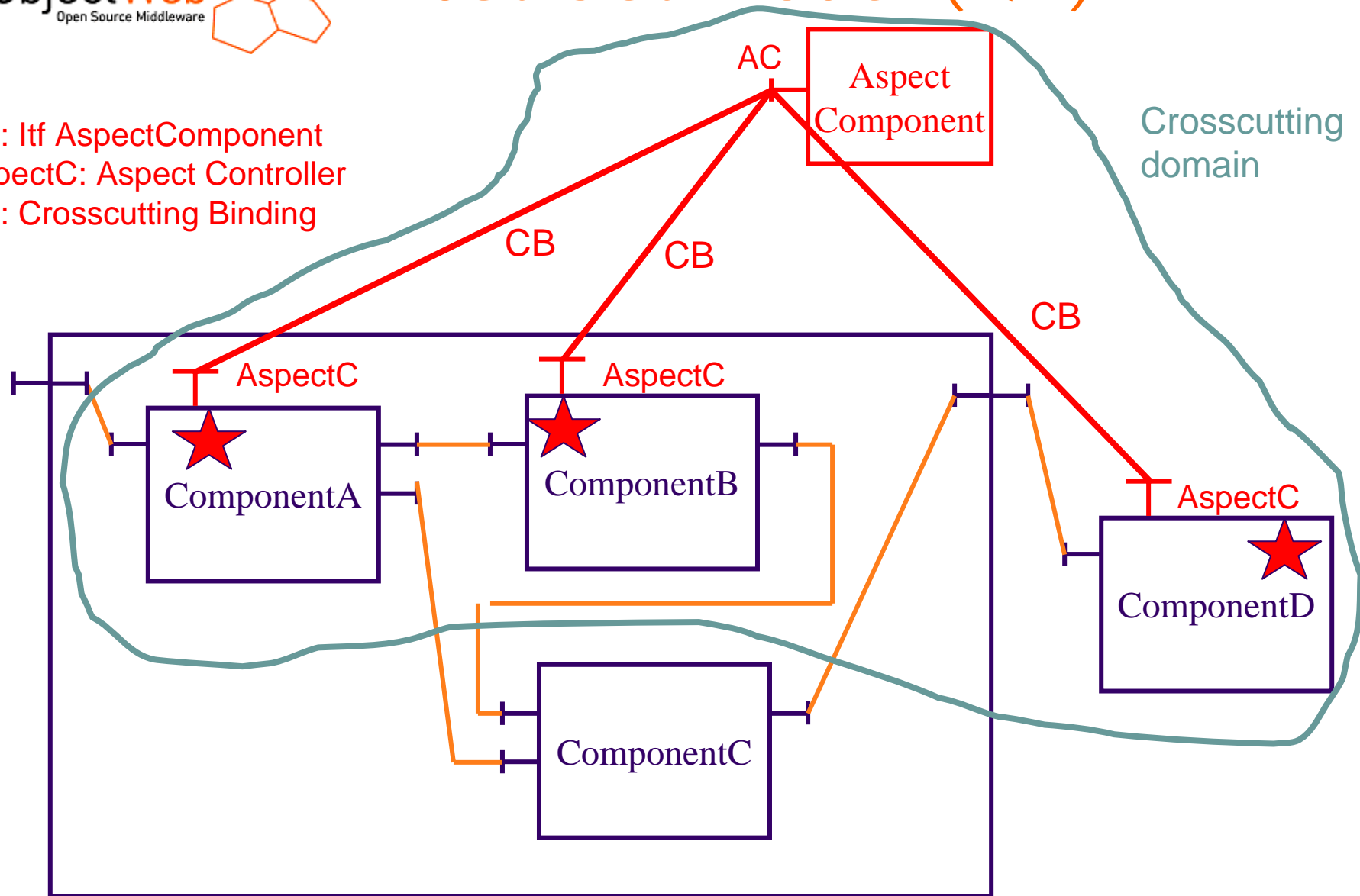
Abstract model (3/4)

AC: Itf AspectComponent
 AspectC: Aspect Controller
 CB: Crosscutting Binding



Abstract model (4/4)

AC: Itf AspectComponent
AspectC: Aspect Controller
CB: Crosscutting Binding



FAC 3 levels

- 3 levels to apply AOP:
 - Object:
 - Content of components
 - Traditional AOP: AspectJ
 - Component:
 - Aspect component (AC)
 - Crosscutting binding (CB)
 - Current FAC
 - Architecture:
 - Architectural pattern transformation
 - Future work

FAC Concrete Model





FAC Concrete Model

- AspectComponent server interface
 - Defines an Aspect Component

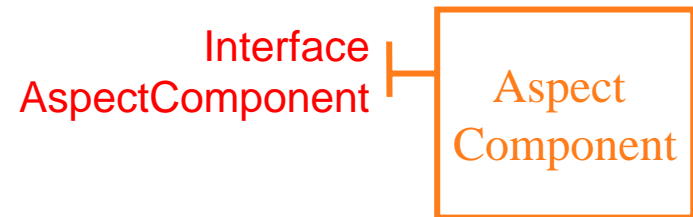
- Aspect Control Interface
 - Supports the Crosscutting Bindings

- FAC Pointcut Language
 - Captures a set of components and interfaces



Aspect Component Interface

- An AC embodies a crosscutting concern
- ACs can be bound to business components



- **Server interface *AspectComponent*:**
 - Relies on **AOP Alliance API**: open source initiative to define a common API for AOP frameworks (Spring, Dynaop, Joyaop, JAC)
 - Applies on client and server component interfaces

Aspect Component (AC)

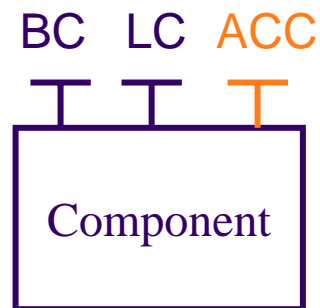
Aspect Component interface

Method to implement

```
public class MyFirstAC
    implements AspectComponent {
    public Object invoke (MethodInvocation m) {
        // before method call
        Object ret=proceed();
        // after method call
        return ret;
    }
}
```

Client interfaces
to deal with business
components

- The **Aspect Controller** supports Crosscutting Bindings:
 - Manages a list of ACs
 - Allows **local order policy** on each component
 - Relies on interception mechanism :
 - **Julius** (**Julia** extension): byte code transformation with ASM
 - **FACAOKell** (**AOKell** extension): AspectJ around advice.



BC : *Binding Controller*
LC : *Life Cycle Controller*
ACC : *Aspect Component Controller*



Aspect Controller API

```
Interface AspectComponent{
```

```
// weaving part
```

```
void weave(Component rootComp,  
           AspectComponent ac,  
           ItfPointcutExp pointcutExp,  
           String cbName);
```

```
void unweave(Component rootComp, Component ac);
```

```
// Pointcut introspection part
```

```
Component[] listAC();
```

```
Component[] listCrosscutComps(Component rComp, Component ac);
```

```
Pointcut aspectizableComps(Component rComp,  
                             ItfPointcutExp pExp);
```

```
}
```

Pointcut language: structural pointcuts

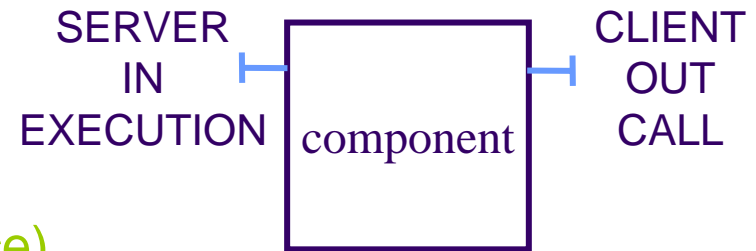
- Currently, 3 regular expressions declared in the ADL:

- Component name

- Interface name

- Method signature

CLIENT/SERVER tag (type of interface)



- Pointcut examples :

- “*.*.*”

- Intercepts every methods on every interfaces of all components

- “SERVER server.* print.*:void”

- Intercepts incoming void methods that begin with “print” in component “server”

Pointcut language: behavioral pointcuts

- The Aspect Component behavior is triggered on a sequence of messages (calls and executions of component interfaces)
- Historic/path of a method call

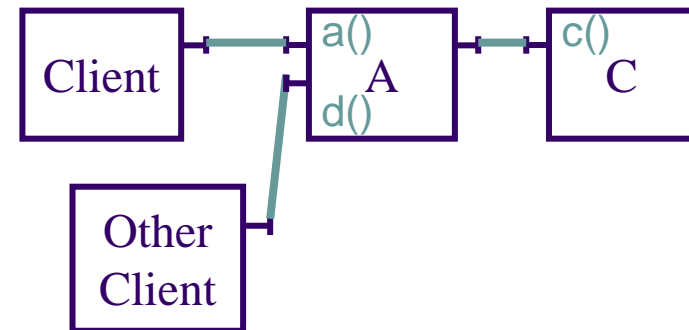
- A state machine is generated
- The expression describes its transitions

- Example:

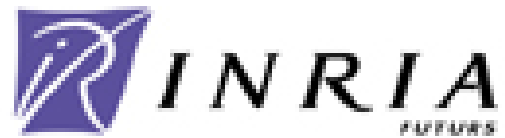
- `Client.a();compA.b();compC.c()` and
`OtherClient.d();compA.b();methodC.c()`

- Wildcards authorized:

- `(a+b);c*;(d+e)` matching sequences: ace, acd, bd, bce, ...



Tools



Fractal-ADL Extension

```
<component name="traceAC">
  <interface name="aspectComponent"
    role="server"
    signature="org.objectweb.fractal.fac.api.Aspe
ctComponent"/>
  <content class="TraceAC"/>
  <controller desc="primitive"/>
</component>
```

```
<weave
  root="this"
  ac="traceAC"
  componentExp="server"
  interfaceExp="s"
  methodExp=".*"
  pointcutName="traceCo"/>
```

```
<definition name="HelloWorld">
  <component name="client">
    <interface name="r" role="server"
      signature="java.lang.Runnable"/>
    <interface name="s" role="client"
      signature="Service"/>
    <content class="ClientImpl"/>
    <controller desc="primitive"/>
  </component>

  <component name="server">
    <interface name="s" role="server"
      signature="Service"/>
    <content class="ServerImpl"/>
    <controller desc="FACprimitive"/>
  </component>
  (...)
  <binding client="client.s"
    server="server.s"/>
</definition>
```


Fractal Explorer Extension

The screenshot shows the Fractal Explorer application interface. On the left is a project tree for 'helloworld' with the following structure:

- helloworld
 - controllers
 - name-controller
 - super-controller
 - aspect-controller
 - binding-controller
 - lifecycle-controller
 - content-controller
 - components
 - client
 - server
 - controllers
 - s
 - traceAC
 - traceCo










On the right is a component diagram for 'helloworld'. The diagram shows three main components: 'client', 'server', and 'traceCo'. 'client' and 'server' are connected by a dependency arrow labeled 's'. 'traceAC' is an aspect component that provides advice to 'client' and 'server'. 'traceCo' is a component that provides advice to 'client', 'server', and 'traceAC'. The diagram includes various role labels (BC, C, LC, NC, SC, AC) and a toolbar at the top with icons for TAC, TBC, TC, TCC, TLC, TNC, and TSC.

At the bottom of the window, the text reads: helloworld: org.objectweb.fractal.api.Component

Fractal Explorer: introspection

List of ACs

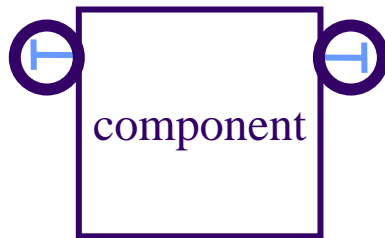
List of ACs	
Aspect Components	PointcutExpression
Sh: traceAC	

List of methods	
methods	
start():void	 traceAC
giveCard(station.C...	 traceAC
giveReceipt(station...	 traceAC
askAuth(java.lang....	 traceAC
sale(java.lang.Strin...	 traceAC
stop(java.lang.Dou...	 traceAC
putCard(station.Ca...	 traceAC
enterCode(station....	 traceAC
	 cryptAC

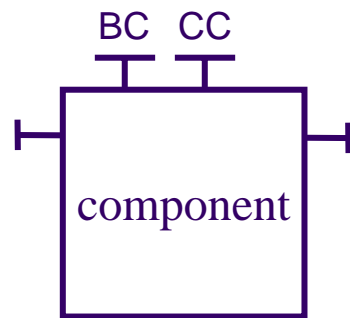
ACs at the
method granularity

- Weaving + Pointcut = new binding
- Symmetric approach: aspects & components are components
- 2 implementations: Julius & FACAOKell
- 3 levels:
 - Object
 - Component
 - Architecture

Perspectives: reach the architecture level

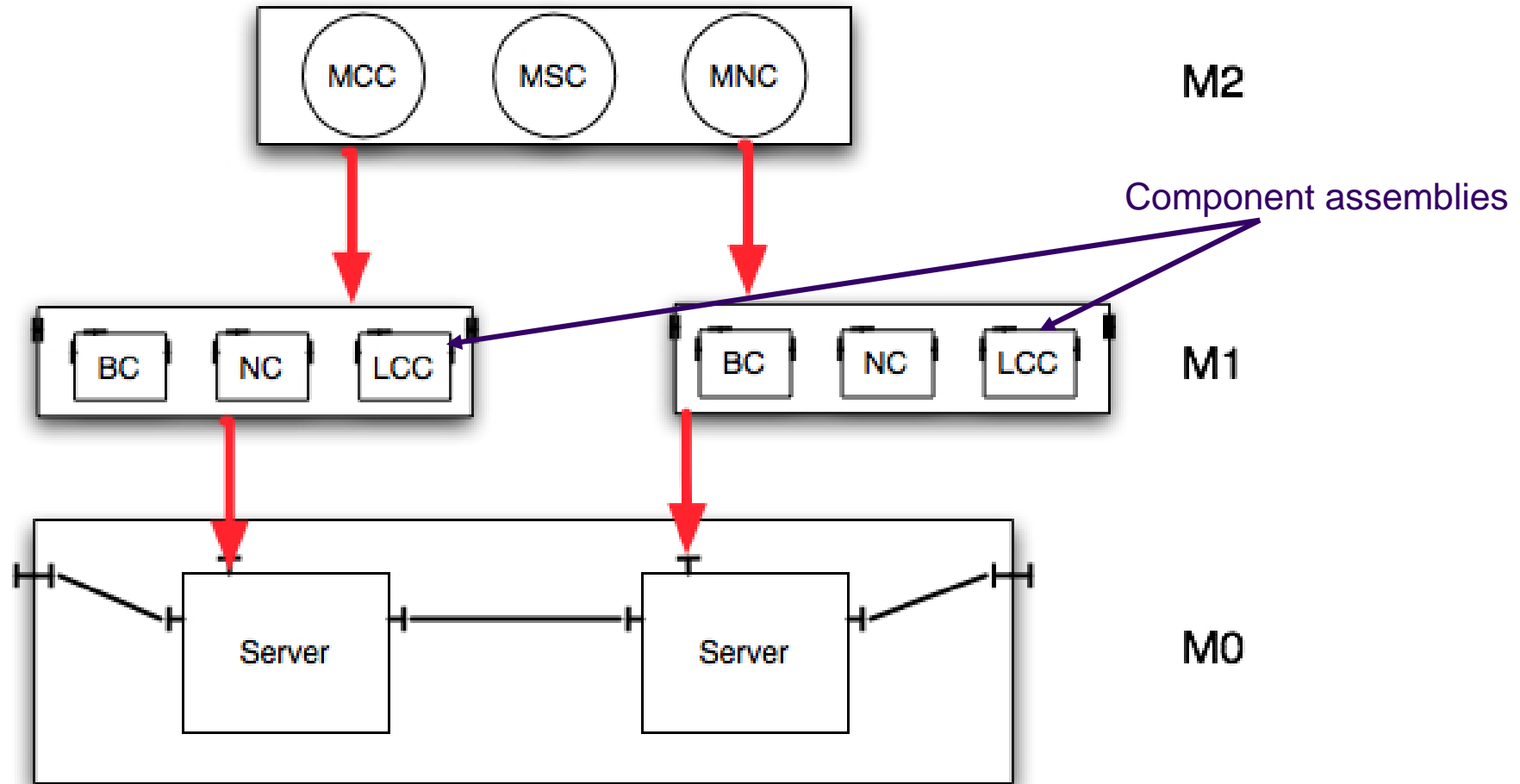


- Currently, advising of incoming and outgoing method calls on a component



- Get the pointcut language richer ?
- Capture a binding ?
- Capture a component that contains exactly 2 components ?

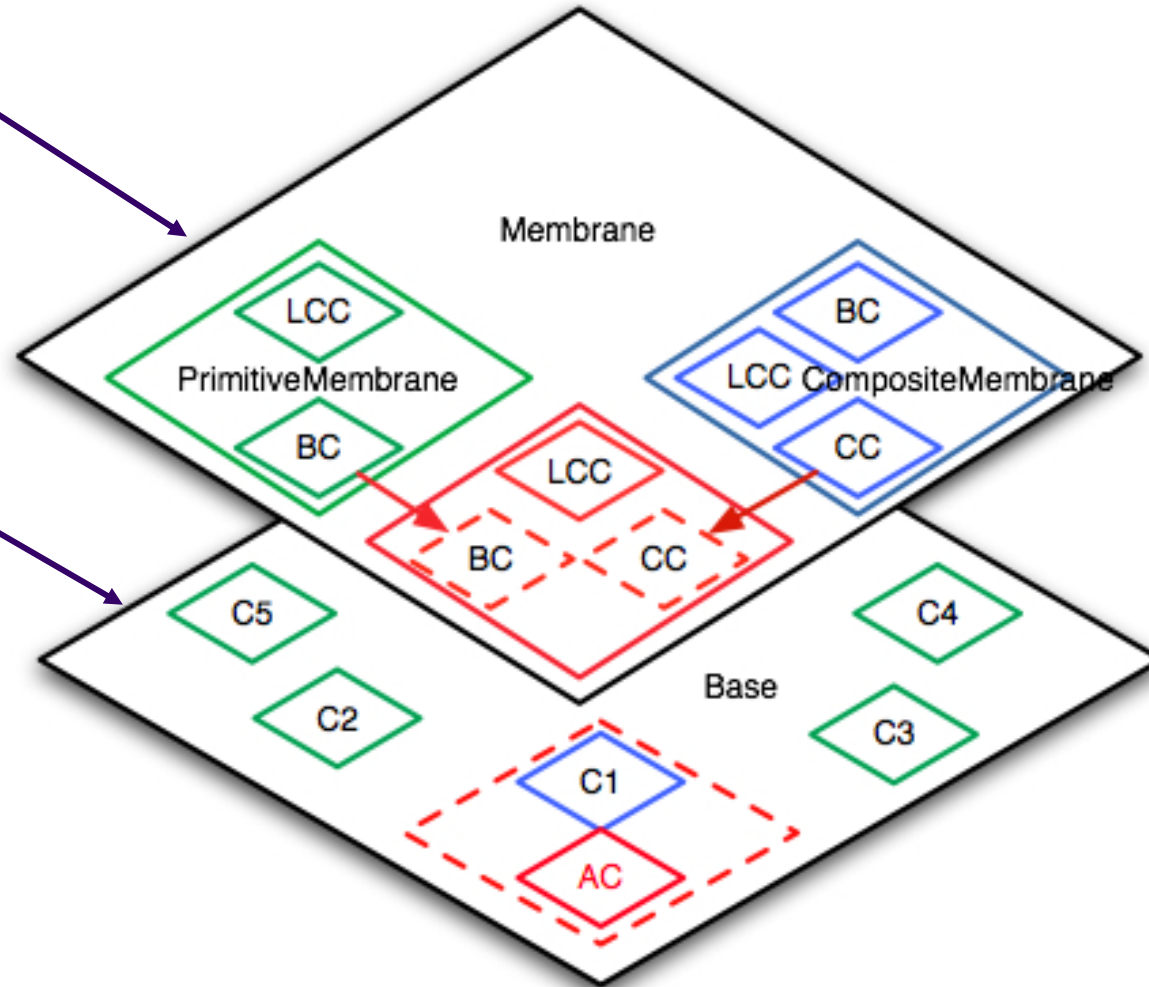
Perspectives: reach the architecture level



Perspectives: reach the architecture level

Controller plan

Base plan



Perspectives: reach the architecture level

- The composite membrane is an independent composite
- Build a new component type is like adding a new composite in level M1
- Different policies:
 - One instance per component type
 - One instance per station
 - ...
- Advising at level M1:
 - Capture the bindings?
 - Capture the recomposition?

