Component-Based Software Engineering (CBSE) 10. Introduction

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de/teaching/cbse
05.04.2017

Lecturer: Dr. Sebastian Götz

The Power of Components

http://upload.wikimedia.org/wikipedia/commons/thumb/1/13/Container_ship_Hanjin_Taipei.jpg/800px-Container_ship_Hanjin_Taipei.jpg

Goals

- Component-based software engineering (CBSE) is the generalization of object-oriented software engineering (OOSE)
  - Understand how to reuse software
  - Component models are the basis of all engineering
- What is a *composition system?*
  - The difference of component-based and composition-based systems
  - The difference of component and composition systems
  - What is a composition operator? composition expression? composition program? composition language?
- Understand the difference between graybox and blackbox systems (variability vs. extensibility)
- Understand the ladder of composition systems
  - Understand the criteria for comparison of composition systems

The Destructive Power of Ill-Used Components: The Ariane 5 Launcher Failure

June 4th 1996

Total failure of the Ariane 5 launcher on its maiden flight

The following slides are from

Ian Summerville, Software Engineering

Credit: DLR/Thilo Kranz (CC-BY 3.0) 2013
http://commons.wikimedia.org/wiki/File:Ariane_5ES_with_ATV_4_on_its_way_to_ELA-3.jpg

http://www.astronews.com/news/artikel/2002/12/0212-009.shtml

Ariane 5 Launcher Failure

- Ariane 5 can carry a heavier payload than Ariane 4
  - Ariane 5 has more thrust (Schub), launches *steeper*
- 37 seconds after lift-off, the Ariane 5 launcher lost control
  - Incorrect control signals were sent to the engines
  - These swivelled so that unsustainable stresses were imposed on the rocket
  - It started to break up and self-destructed
- The system failure was a software failure

Ian Summerville, Software Engineering

The Problem of Component Reuse

- The attitude and trajectory of the rocket are measured by a computer-based inertial reference system
  - This transmits commands to the engines to maintain attitude and direction
  - The software failed and this system and the backup system shut down
- Diagnostic commands were transmitted to the engines
  - ..which interpreted them as real data and which swivelled to an extreme position
- Technically: Reuse Problem
  - Integer overflow failure occurred during converting a 64-bit floating point number to a signed 16-bit integer
  - There was no exception handler
  - So the system exception management facilities shut down the software

Ian Summerville, Software Engineering

Software Reuse Error

- The erroneous software component (Ada-83) was reused from the Ariane 4 launch vehicle.
- The computation that resulted in overflow was not used by Ariane 5.
- Decisions were made in the development
  - Not to remove the facility as this could introduce new faults
  - Not to test for overflow exceptions because the processor was heavily loaded.
  - For dependability reasons, it was thought desirable to have some spare processor capacity
- Why not in Ariane 4?
  - Ariane 4 has a lower initial acceleration and build up of horizontal velocity than Ariane 5
  - The value of the variable on Ariane 4 could never reach a level that caused overflow during the launch period.
- That had been proved (**proven component contract** for Ariane 4)!
- The contract was not re-proven for Ariane-5
- There was also no run-time check for contract violation in Ariane-5

10.1. Basics of Composition Systems

- Component-based software engineering is built on **composition systems.**
- A composition system has a component model, a composition technique, and a composition language.

Motivation for Component-Based Development

- Component-Based Development is the basis of *all* engineering
  - Development by "divide-and-conquer" (Alexander the Great)
  - Well known in other disciplines
    - Mechanical engineering (e.g., German VDI 2221)
    - Electrical engineering
    - Architecture
- "Make, reuse or buy" decisions (reuse decisions):
  - Outsourcing to component producers (Components off the shelf, COTS)
    - Reuse of partial solutions
    - Easy configurability of the systems: variants, versions, product families
- Scaling business by Software Ecosystems
  - Component models and composition systems are the technical basis for all modern software ecosystems: Linux, Eclipse, AutoSAR, openHAB,…

Mass-produced Software Components

- Mass Produced Software Components [McIlroy, Garmisch 68, NATO conference on software engineering]:
  - Every ripe industry is based on components, to manage large systems
  - Components should be produced in masses and composed to systems afterwards

In the phrase `mass production techniques,' my emphasis is on `techniques' and not on mass production plain. Of course mass production, in the sense of limitless replication of a prototype, is trivial for software.

But certain ideas from industrial technique I claim are relevant.
•The idea of subassemblies carries over directly and is well exploited.
•The idea of interchangeable parts corresponds roughly to our term `modularity,' and is fitfully respected.
•The idea of machine tools has an analogue in assembly programs and compilers.

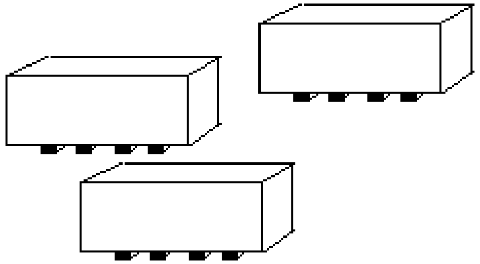
Yet this fragile analogy is belied when we seek for analogues of other tangible symbols of mass production.
•There do not exist manufacturers of standard parts, much less catalogues of standard parts.
•One may not order parts to individual specifications of size, ruggedness, speed, capacity, precision or character set.

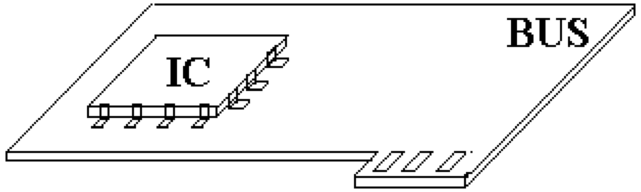
Mass-produced Software Components

- Later McIlroy was with Bell Labs,
  - ..and invented pipes, diff, join, echo (UNIX).
  - Pipes are still today the most employed component system!
- Where are we today?

"Real" Component Systems

- Lego
- Square stones
- Building plans
- IC's
- Hardware bus
- How do they differ from software?





Definitions of Software Components

A software component is a unit of composition

- with contractually specified interfaces
- and explicit context dependencies only.

A software component

- can be deployed independently and
- is subject to composition by third parties.

(ECOOP Workshop WCOP 1997 Szyperski)

A reusable software component is a

- logically cohesive,
- loosely coupled module
- that denotes a single abstraction. (Grady Booch)

A software component is a static abstraction with plugs.

(Nierstrasz/Dami)

What is a Software Component?

- A component is a *container with*
  - *Hidden inner*
  - *Public outer interface, stating all dependencies explicitly*

- Example: a snippet component is a snippet with
  - Inner: content (most often code snippets/fragments)
  - Outer: variation points, extension points that are adapted during composition
- Example: a class with provided and required interfaces
  - Inner: methods as usual

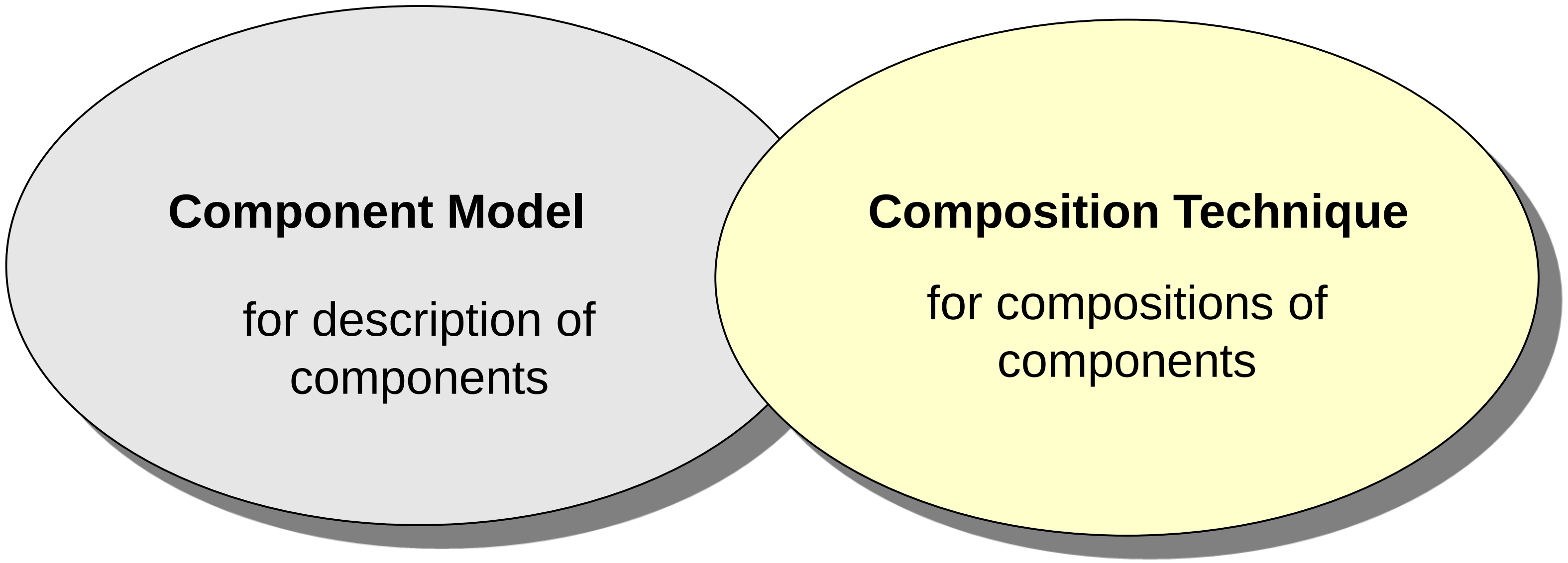
- A component is a reusable *unit for composition*
- A component underlies *a component model*
  - that fixes the abstraction level
  - that fixes the grain size (widget or OS?)
  - that fixes the time (static or runtime?)

What Is a Component-Based System?

- A component-based system has the following **divide-and-conquer feature**:
  - A component-based system is a system in which a major relationship between the components is **tree-shaped or reducible**.
  - See course Softwaretechnologie-II
- Consequence: the entire system can be reduced to one abstract node
  - at least along the structuring relationship
  - Systems with layered relations (dag-like relations) are not necessarily component-based.
  - Because they cannot be reduced
- Because of the divide-and-conquer property, component-based development is attractive.
  - However, we have to choose the structuring relation and the composition model
- Mainly, 2 types of component models are known
  - Modular decomposition (blackbox)
  - Separation of concerns (graybox)

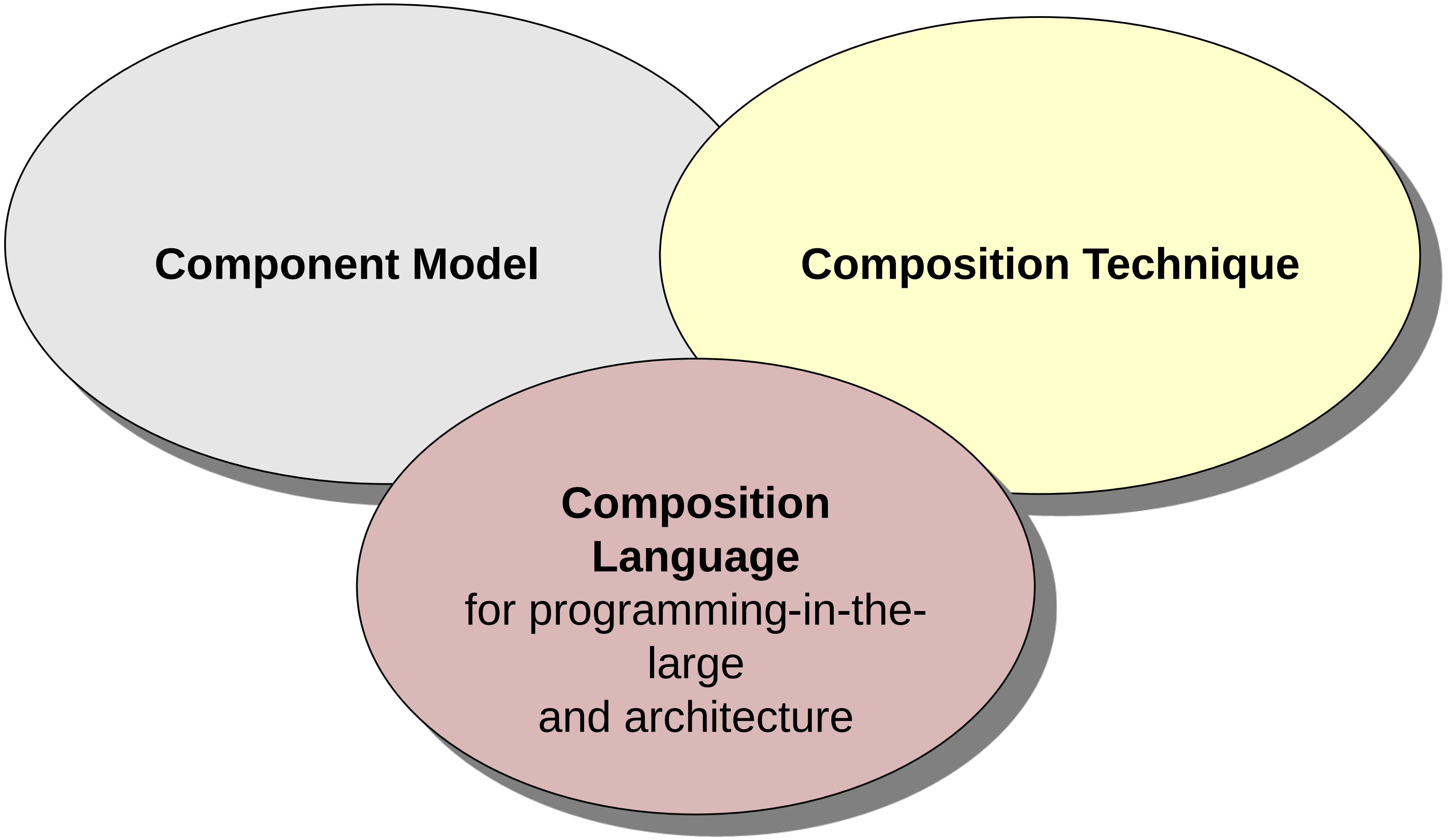
Component Systems (Component Platforms)

- We call a technology in which component-based systems can be produced a *component system* or *component platform.*
- A component system has


Component Model
for description of components
Composition Technique
for compositions of components

Composition Systems

- A composition system has



The Ladder of Composition Systems

Software Composition Systems	Composition Language	*Invasive Composition Piccola Gloo*
Aspect Systems	Aspect Separation Crosscutting	*Aspect/J AOM*
View Systems	Composition Operators	*Composition Filters Hyperspaces*
Architecture Systems	Architecture as Aspect Connectors	*Darwin BPMN HRC*
Classical Component Systems	Standard Components Reflection	*.NET CORBA Beans EJB ArchJava*
Object-Oriented Systems	Objects as Run-Time Components	*C++ Java UML components*
Modular Systems	Modules as Compile-Time Components	*Shell scripts Modula Ada-85*

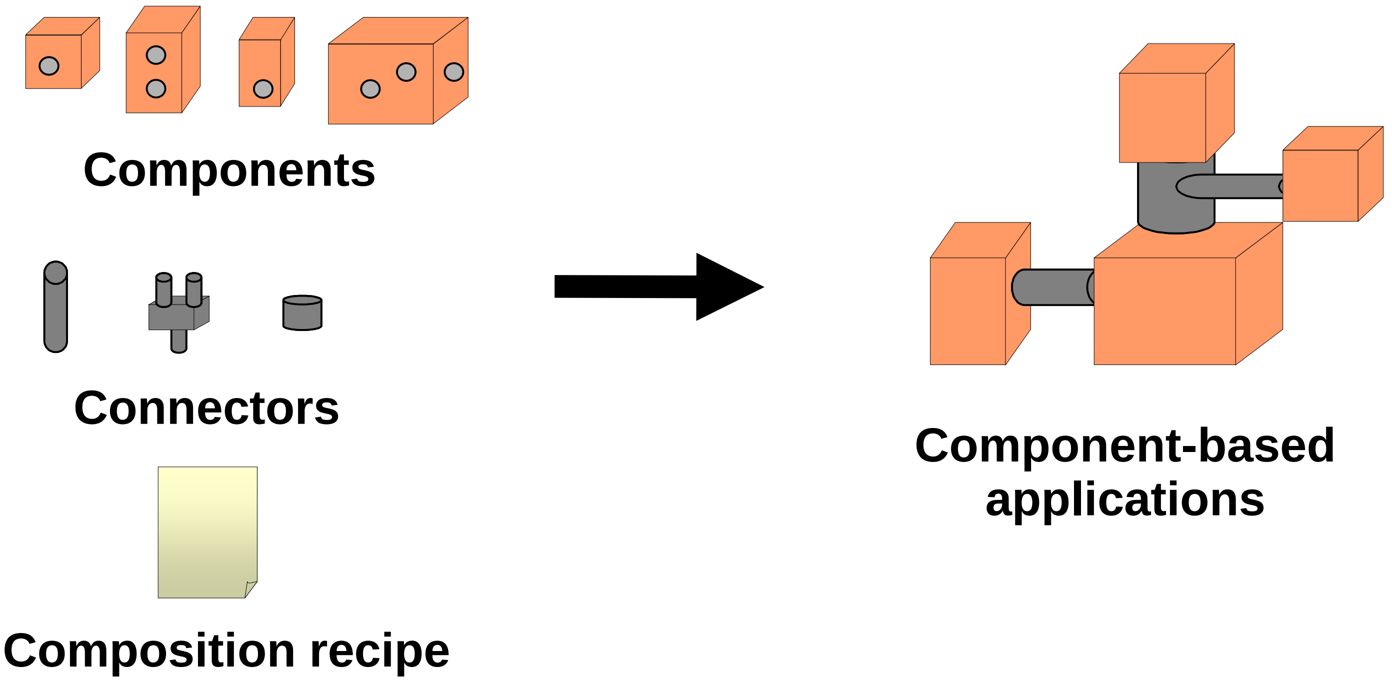
Desiderata for Flexible Software Composition

- Component Model:
  - How do components look like?
  - Secrets, interfaces, substitutability
- Composition Technique
  - How are components plugged together, composed, merged, applied?
  - Composition time (Deployment, Connection, ...)
- Composition Language
  - How are compositions of large systems described?
  - How are system builds managed?
- Be aware: this list is NOT complete!

10.2 Historical Approaches to Components

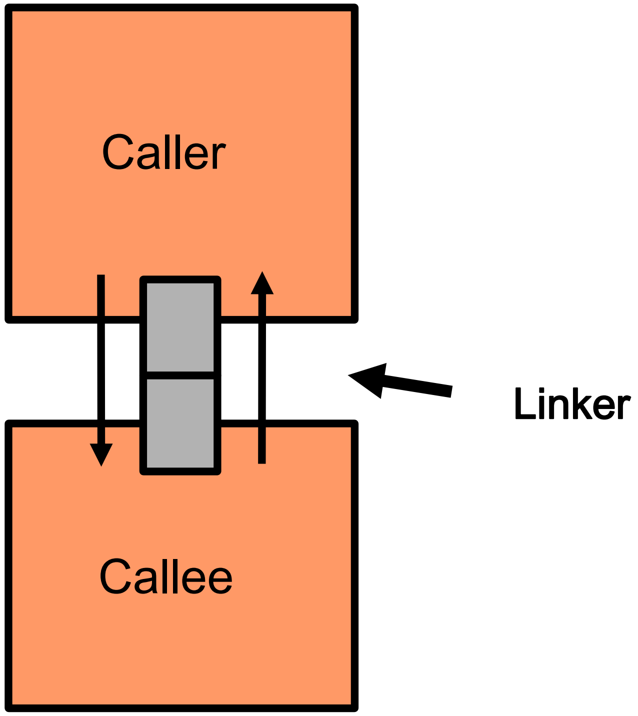
The Essence of the 60s-90s: LEGO Software with Black-Box Composition

- Procedural systems, stream-based systems
- Modular systems
- Object-oriented technology
- Component-based programming
  - CORBA, EJB, DCOM, COM+, .NET, OSGI
- Architecture languages

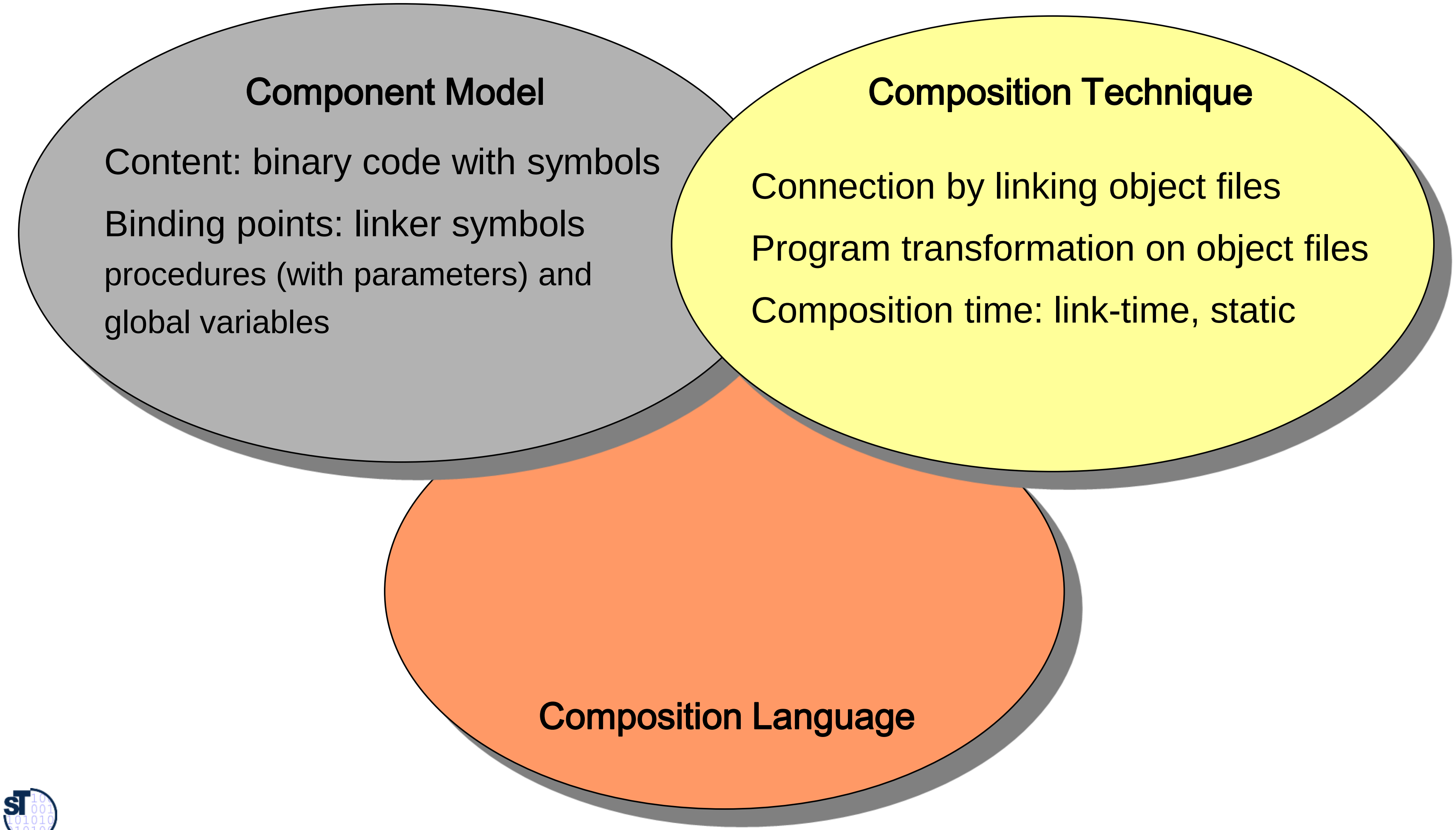


Procedure Systems

- Fortran, Algol, C
- The procedure is the static component
- The activation record the dynamic one
- Component model is supported by almost all chips directly
  - jumpSubroutine -- return



Procedures as Composition System



Component Model

Content: binary code with symbols

Binding points: linker symbols

procedures (with parameters) and global variables

Composition Technique

Connection by linking object files

Program transformation on object files

Composition time: link-time, static

Composition Language

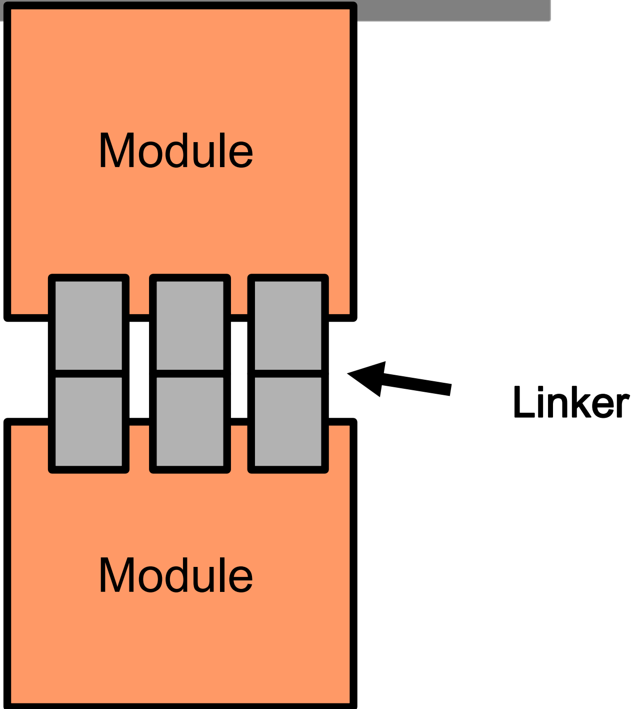
Modules (Information-Hiding-Based Design a la Parnas)

- Every module hides an important design decision behind a well-defined interface which does not change when the decision changes.

We can attempt to define our modules “around” assumptions which are likely to change. One then designs a module which “hides” or contains each one.

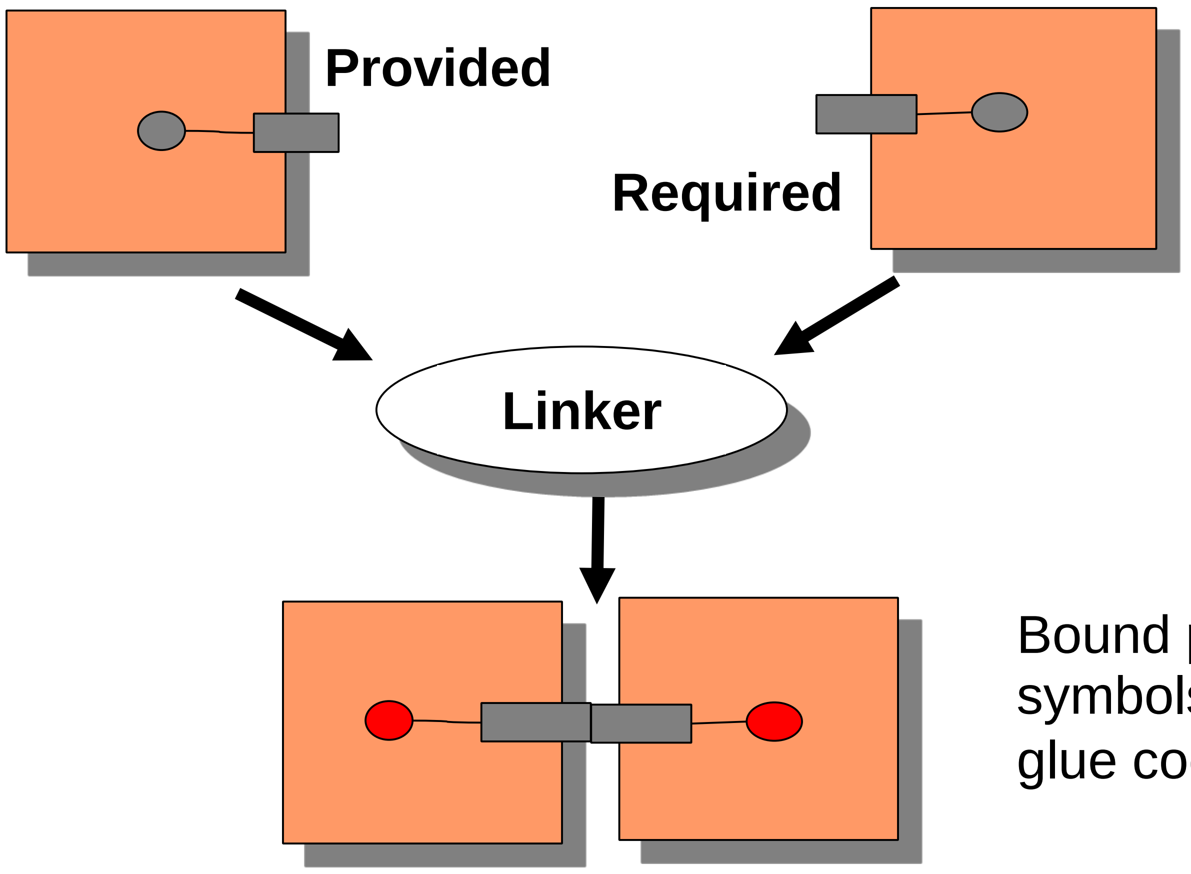
Such modules have rather abstract interfaces which are relatively unlikely to change.

- Static binding of functional interfaces to each other
- Concept has penetrated almost all programming languages (Modula, Ada, Java, C++, Standard ML, C#)



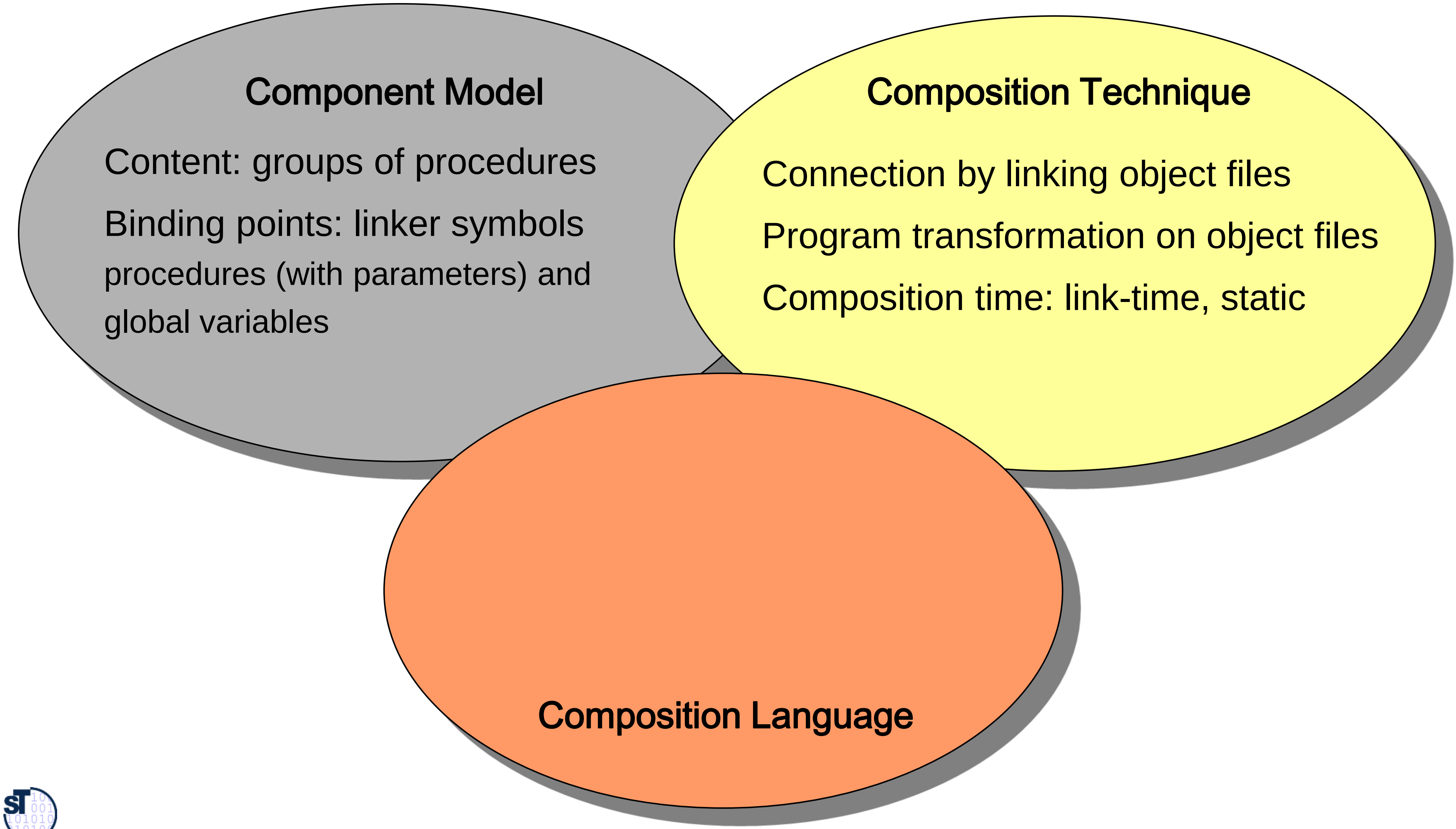
A Linker is a Static Composition Operator

- Static linkers compose modules at link time
- Dynamic linkers at run time



Bound procedure symbols, no glue code

Modules as Composition System



Component Model

Content: groups of procedures

Binding points: linker symbols

procedures (with parameters) and global variables

Composition Technique

Connection by linking object files

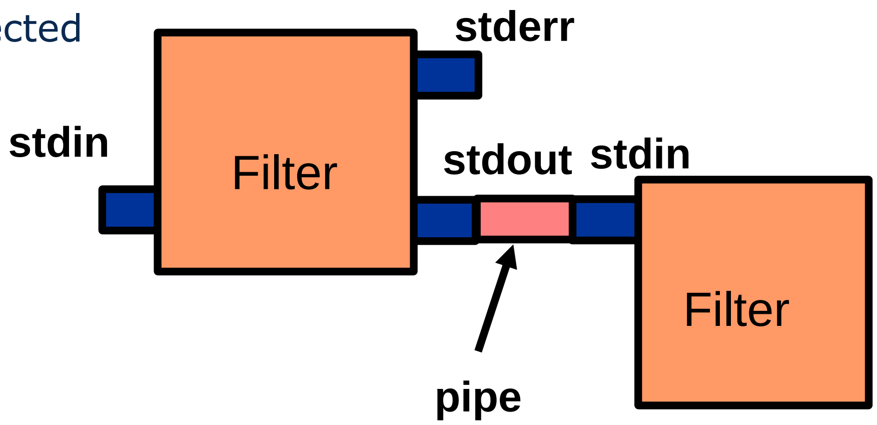
Program transformation on object files

Composition time: link-time, static

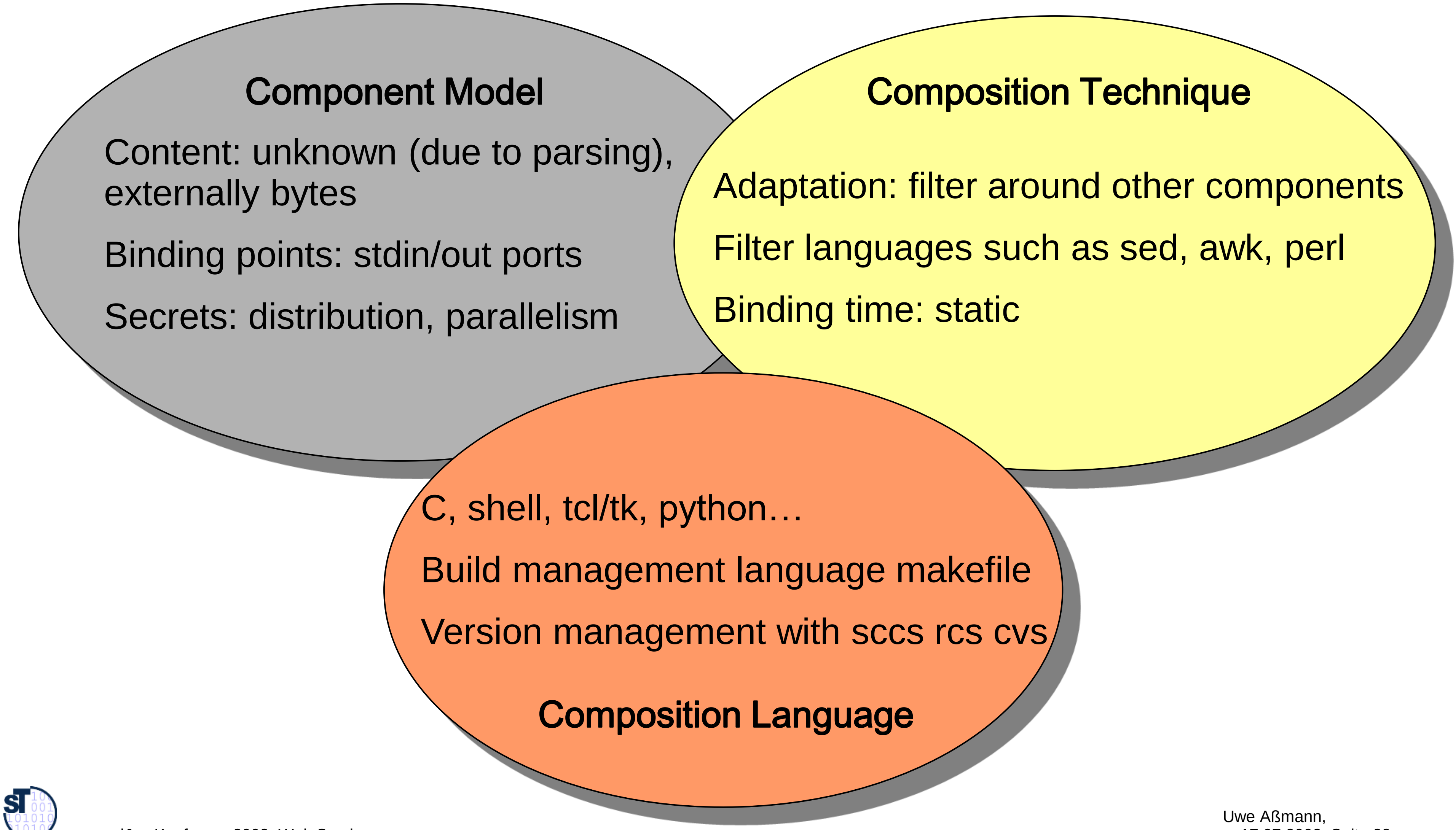
Composition Language

UNIX Pipes and Filters (McIlroy)

- Communication can take place once or many times
  - By **Calls** (singular) or **Streams** (continuous)
- UNIX shells offer a component model for streams
  - Extremely flexible, simple
  - Communication with byte streams, parsing and linearizing the objects
- Component model
  - Content: unknown (depens on parsing), externally bytes
  - Binding points: stdin/stdout/stderr ports
  - More secrets: distribution, parallelism etc
- Composition technique: manipulation of byte streams
  - Adaptation: filter around other components. Filter languages such as sed, awk, perl
  - Binding time: static, streams are connected (via filters) during composition
- Composition languages
  - C, shell, tcl/tk, python, perl…
  - Build management language makefile



Shells and Pipes as Composition System



Component Model

Content: unknown (due to parsing), externally bytes

Binding points: stdin/out ports

Secrets: distribution, parallelism

Composition Technique

Adaptation: filter around other components

Filter languages such as sed, awk, perl

Binding time: static

C, shell, tcl/tk, python…

Build management language makefile

Version management with sccs rcs cvs

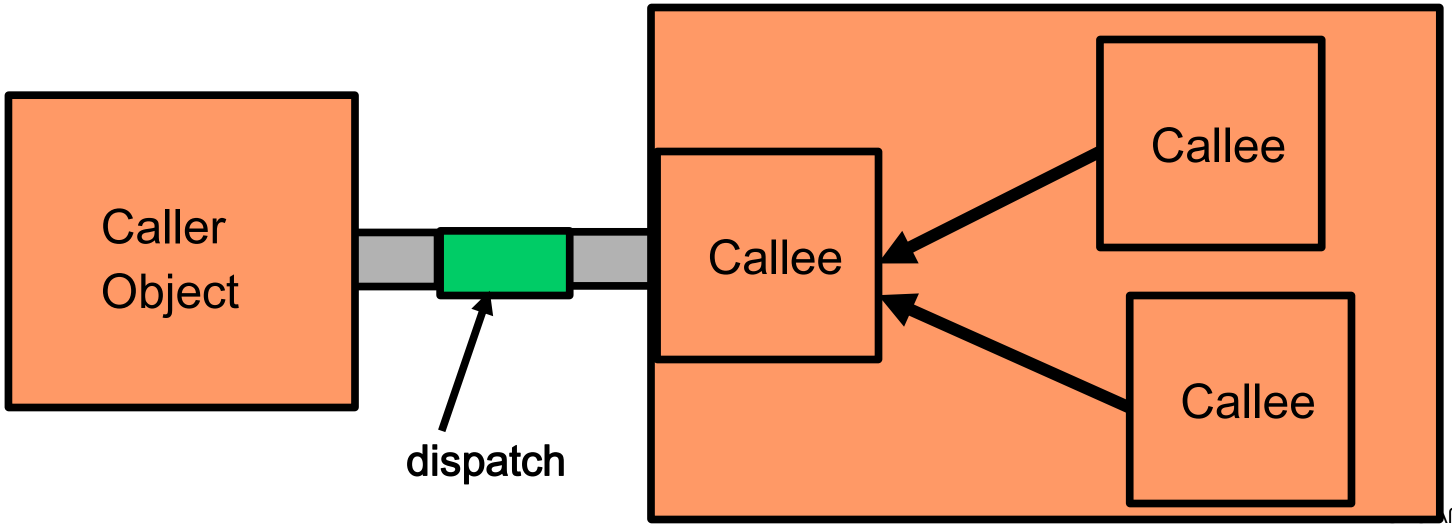
Composition Language

Communication

- Black-box components communicate either
  - Via calls (singular): → algebraic data types, induction
  - Via streams (continuous) → coalgebraic data types, coinduction

Object-Oriented Systems

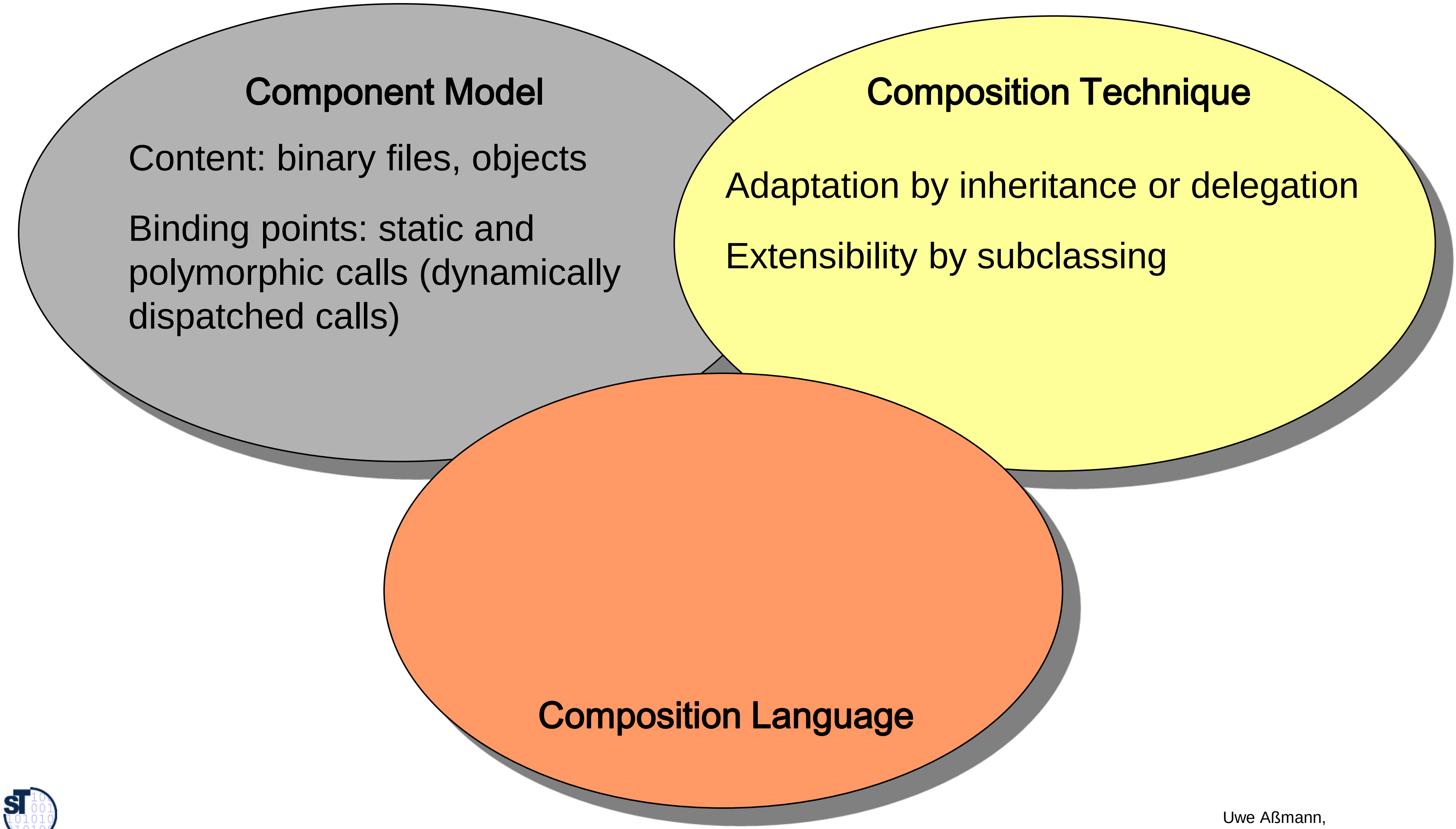
- Two sorts of components: objects (runtime) and classes (compile time)
  - Objects are instances of classes (modules) with unique identity
  - Objects have runtime state
  - Late binding of calls by search at runtime



Object-Oriented Systems

- Component Model
  - Content: classes (code, static) and objects (values, dynamic)
  - Binding points:
    - monomorphic calls (static calls)
    - polymorpic calls (dynamically dispatched calls)
- Composition Technique
  - Adaptation by inheritance or delegation
  - Extensibility by subclassing
- Composition Language: none

Object-Orientation as Composition System



Component Model

Content: binary files, objects

Binding points: static and polymorphic calls (dynamically dispatched calls)

Composition Technique

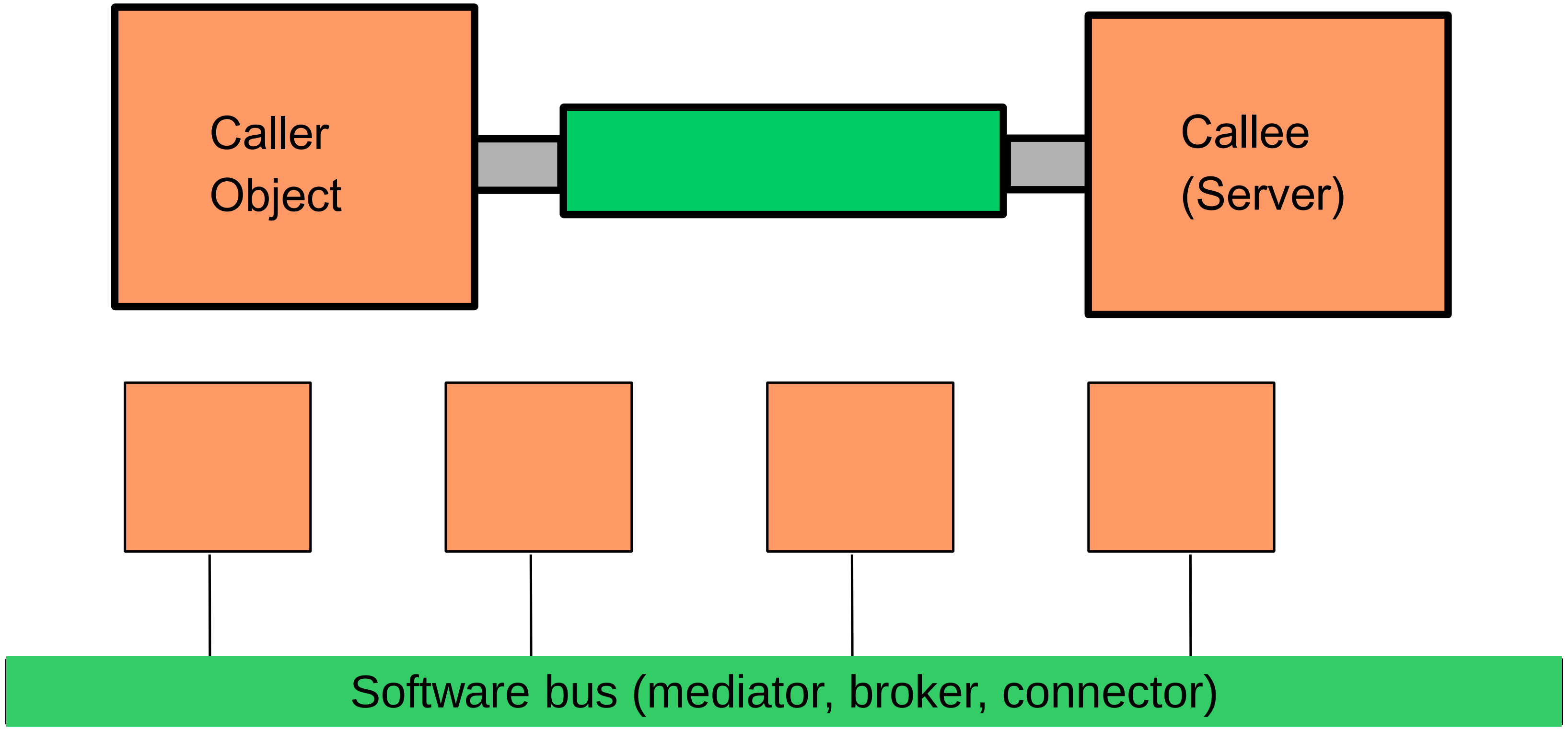
Adaptation by inheritance or delegation

Extensibility by subclassing

Composition Language

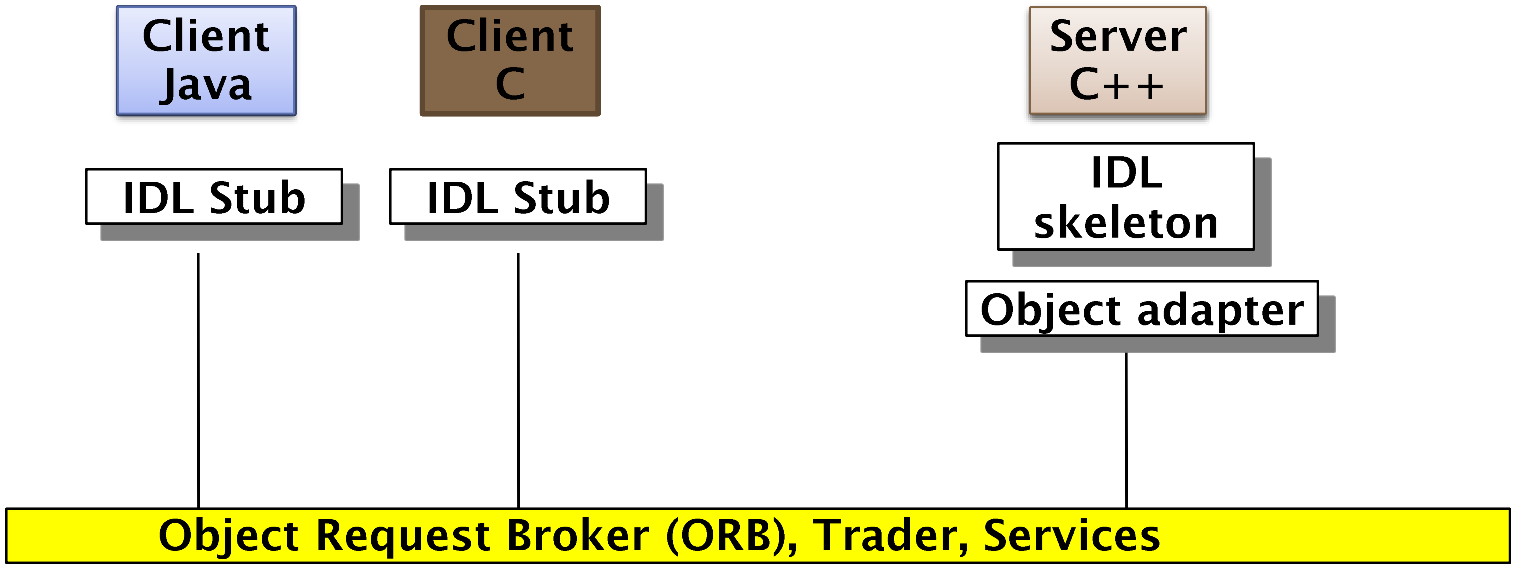
Commercial Component Systems (COTS, Components off the Shelf)

- CORBA/DCOM/.NET/JavaBeans/EJB
- Although different on the first sight, turn out to be rather similar



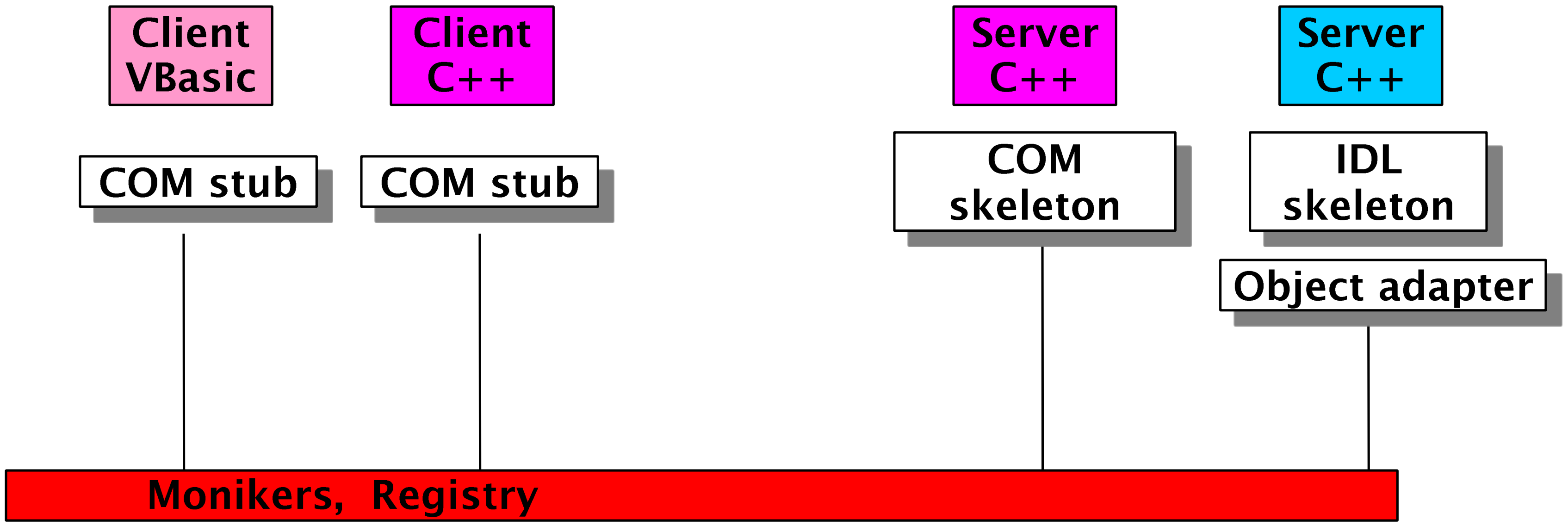
CORBA
http://www.omg.org/corba

- Language independent, distribution transparent
- interface definition language IDL
- source code or binary



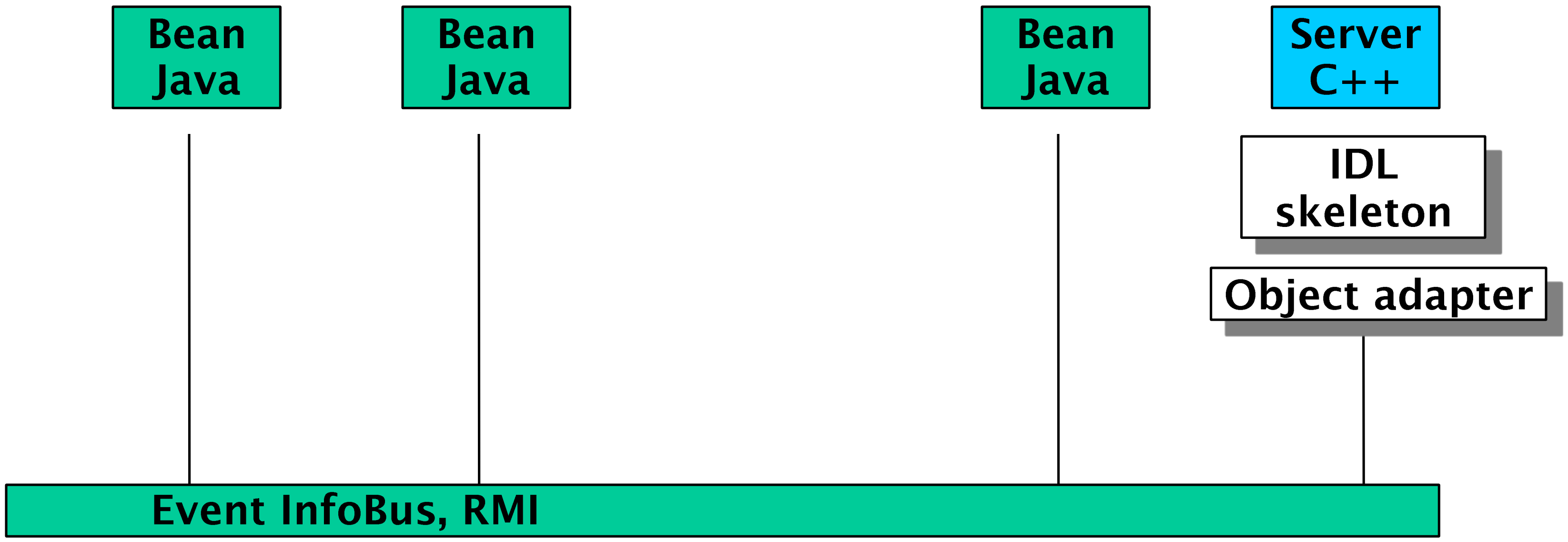
(D)COM(+), ActiveX
http://www.activex.org

- Microsoft's model is similar to CORBA. Proprietary
- DCOM is a binary standard



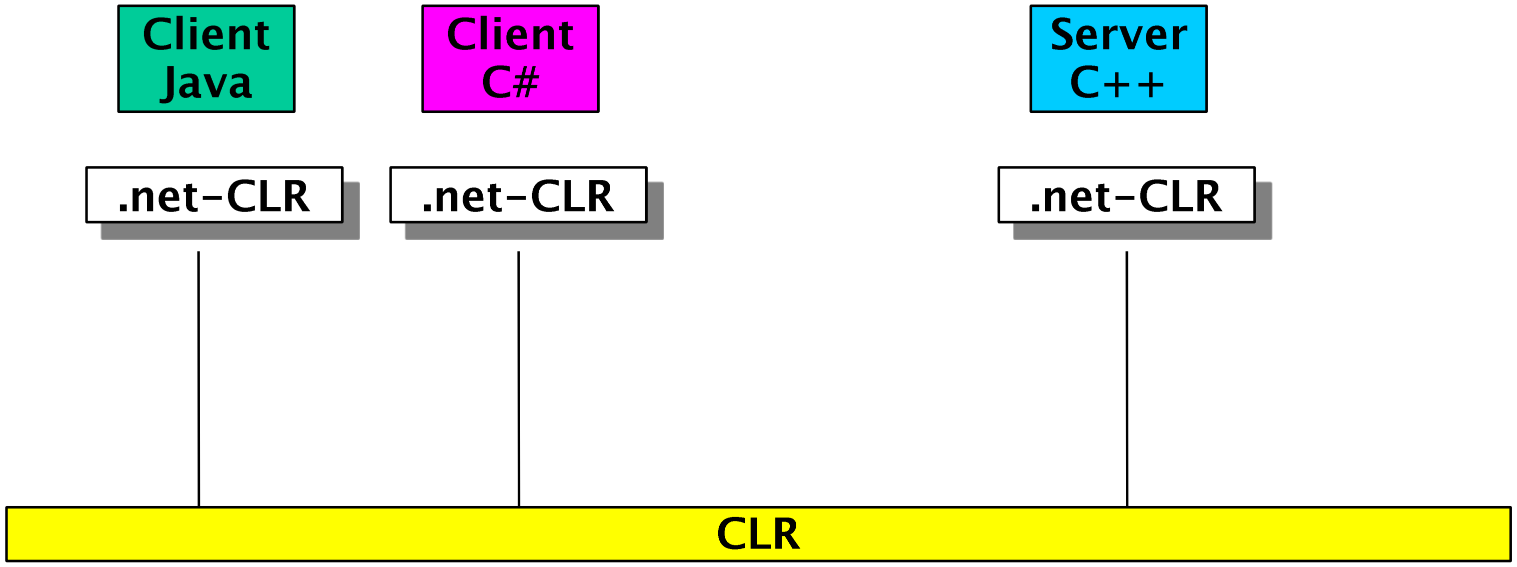
Java Enterprise Beans

- Java only, event-based, transparent distribution by remote method invocation (RMI)
- source code/bytecode-based



.NET
http://www.microsoft.com

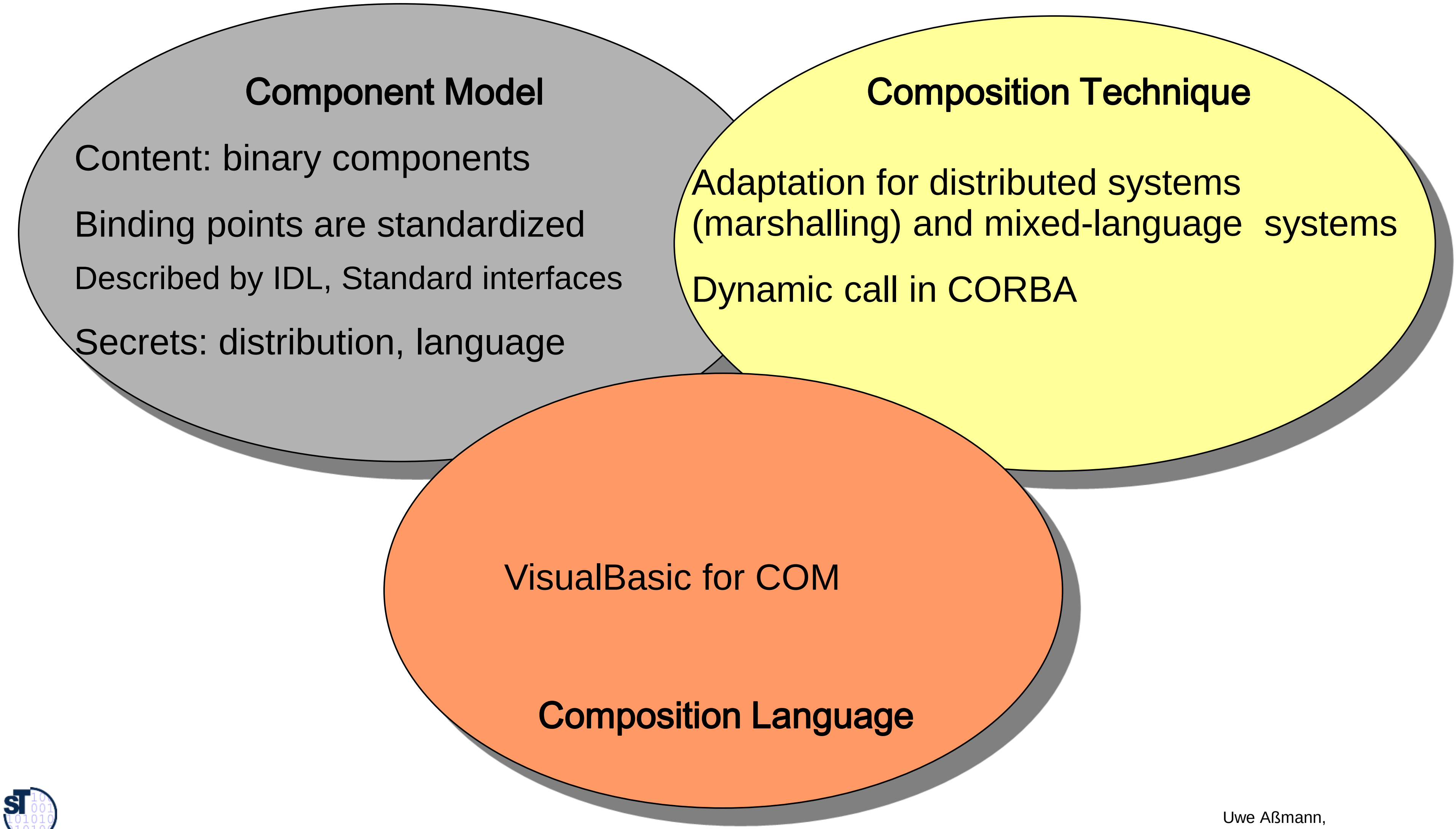
- Language independent, distribution transparent
- NO interface definition language IDL (at least for C#)
- source code or bytecode MSIL
- Common Language Runtime CLR



COTS

- Component Model
  - Content: binary components
  - Secrets: Distribution, implementation language
  - Binding points are standardized
    - Described by IDL languages
    - set/get properties
    - standard interfaces such as IUnknown (QueryInterface)
- Composition Technique
  - External adaptation for distributed systems (marshalling) and mixed-language systems (IDL)
  - Dynamic call in CORBA
- Composition Language
  - e.g., Visual Basic for COM

COTS as Composition System



Component Model

Content: binary components

Binding points are standardized

Described by IDL, Standard interfaces

Secrets: distribution, language

Composition Technique

Adaptation for distributed systems (marshalling) and mixed-language systems

Dynamic call in CORBA

VisualBasic for COM

Composition Language

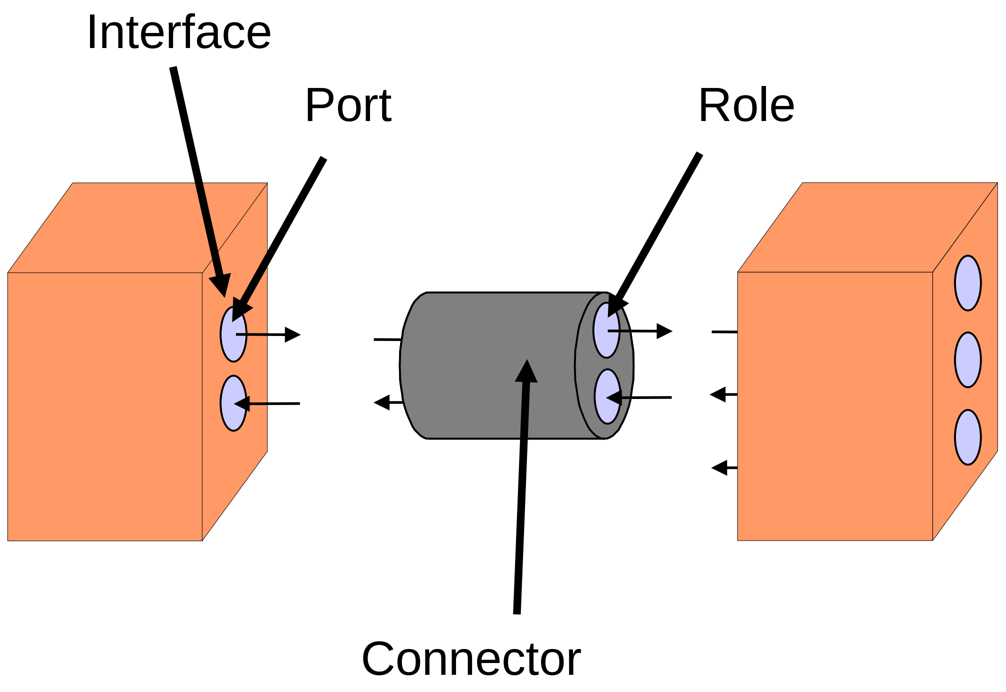
Architecture Systems

- Unicon, ACME, Darwin, Reo (research languages)
  - feature an Architecture Description Language (ADL)
- EAST-ADL, Artop are ADL in Embedded Software
- BPEL, BPMN in Web Services

- Split an application into:
  - Application-specific part (encapsulated in components)
  - Architecture and communication (in architectural description in ADL)
  - Better reuse since both dimensions can be varied independently

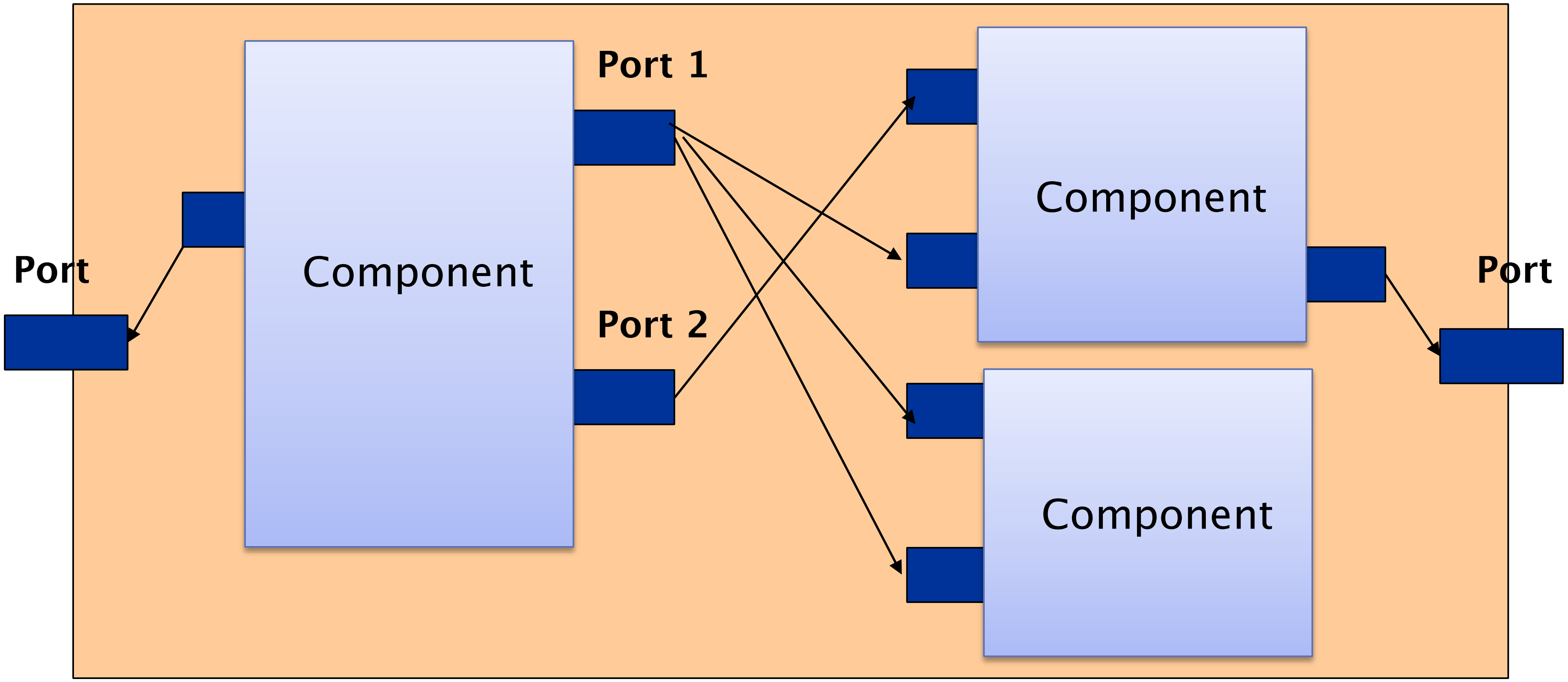
Component Model in Architecture Systems

- **Ports** abstract interface communication points
  - in(data), out(data)
  - Components may be nested
- **Connectors** as special communication components

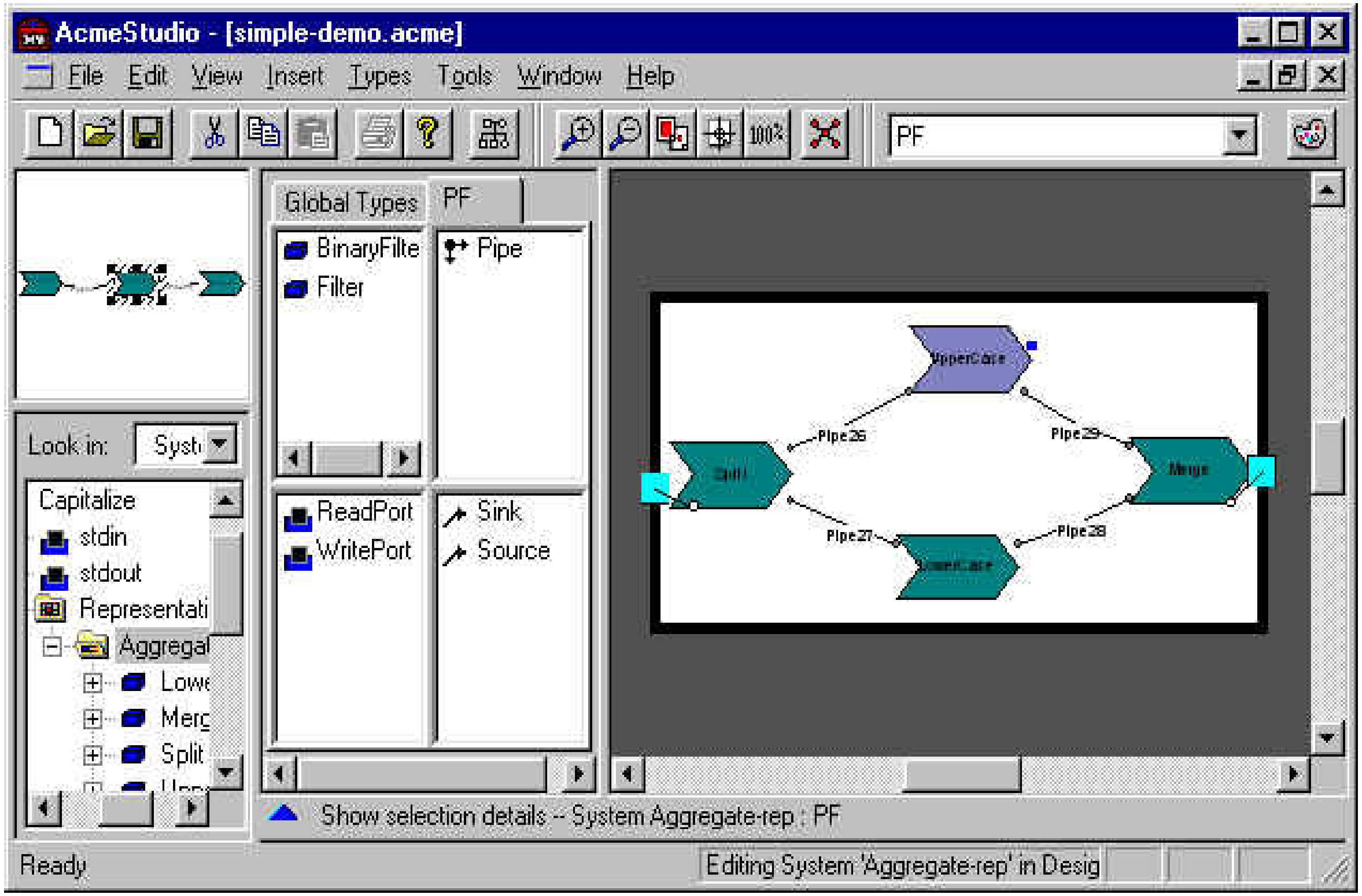


Architecture can be exchanged independently of components

- Reuse of components and architectures is fundamentally improved



ACME Studio

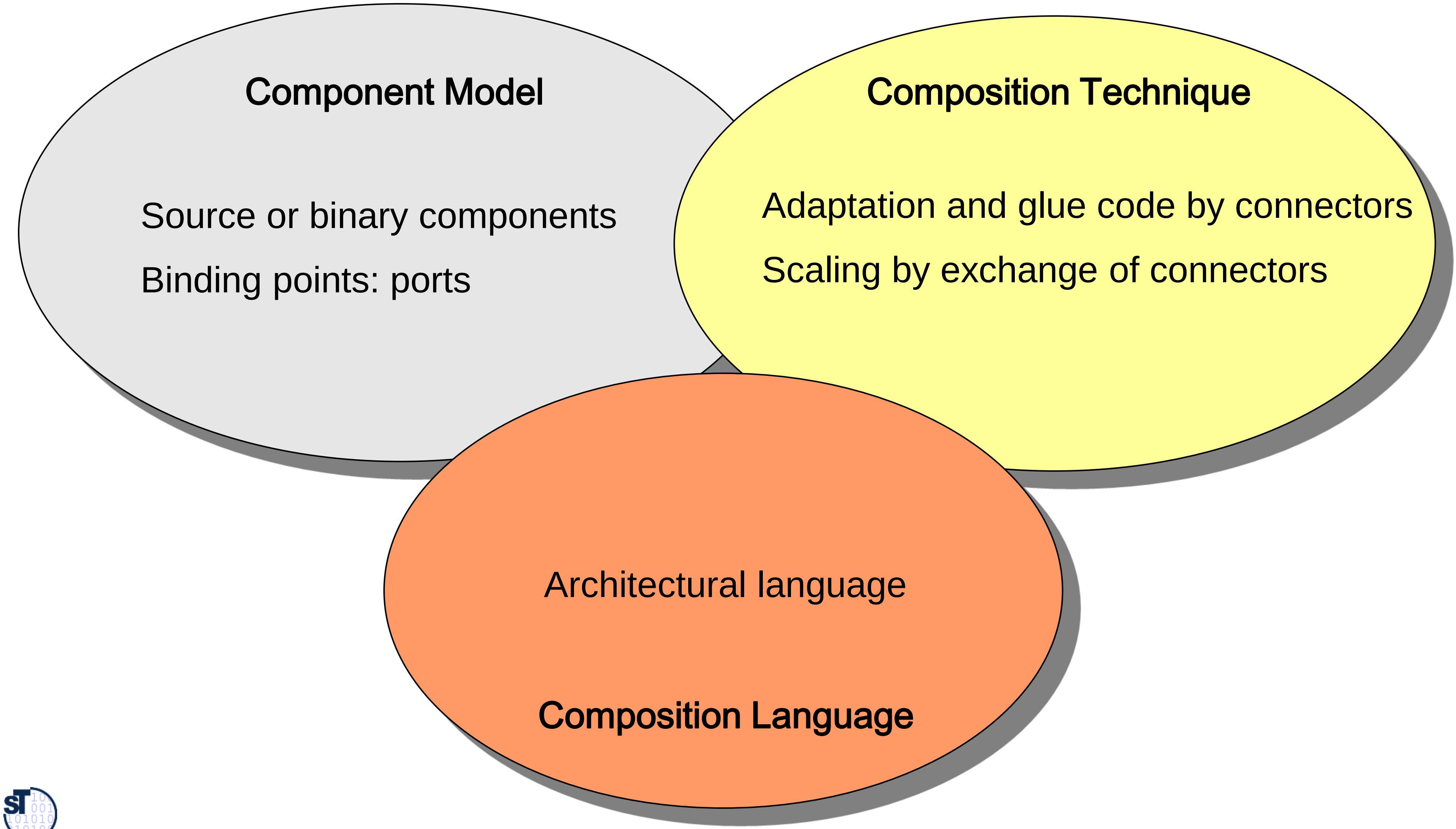


Show selection details -- System Aggregate-rep : PF

Ready

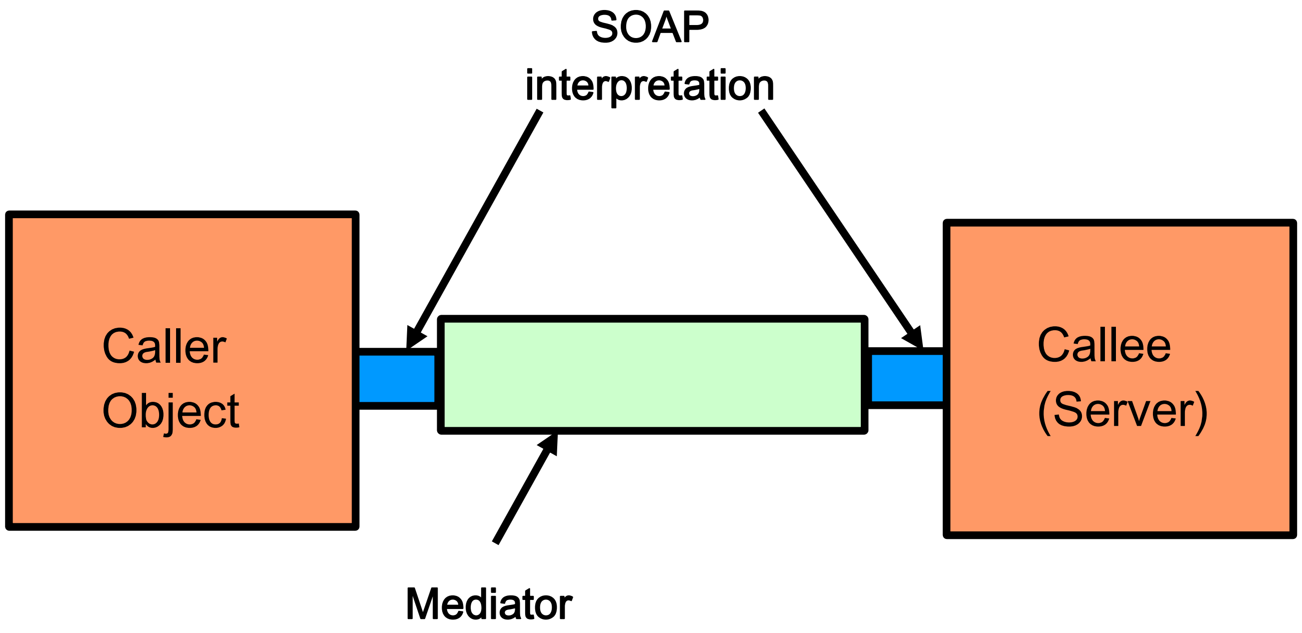
Editing System 'Aggregate-rep' in Desig

Architecture Systems as Composition Systems

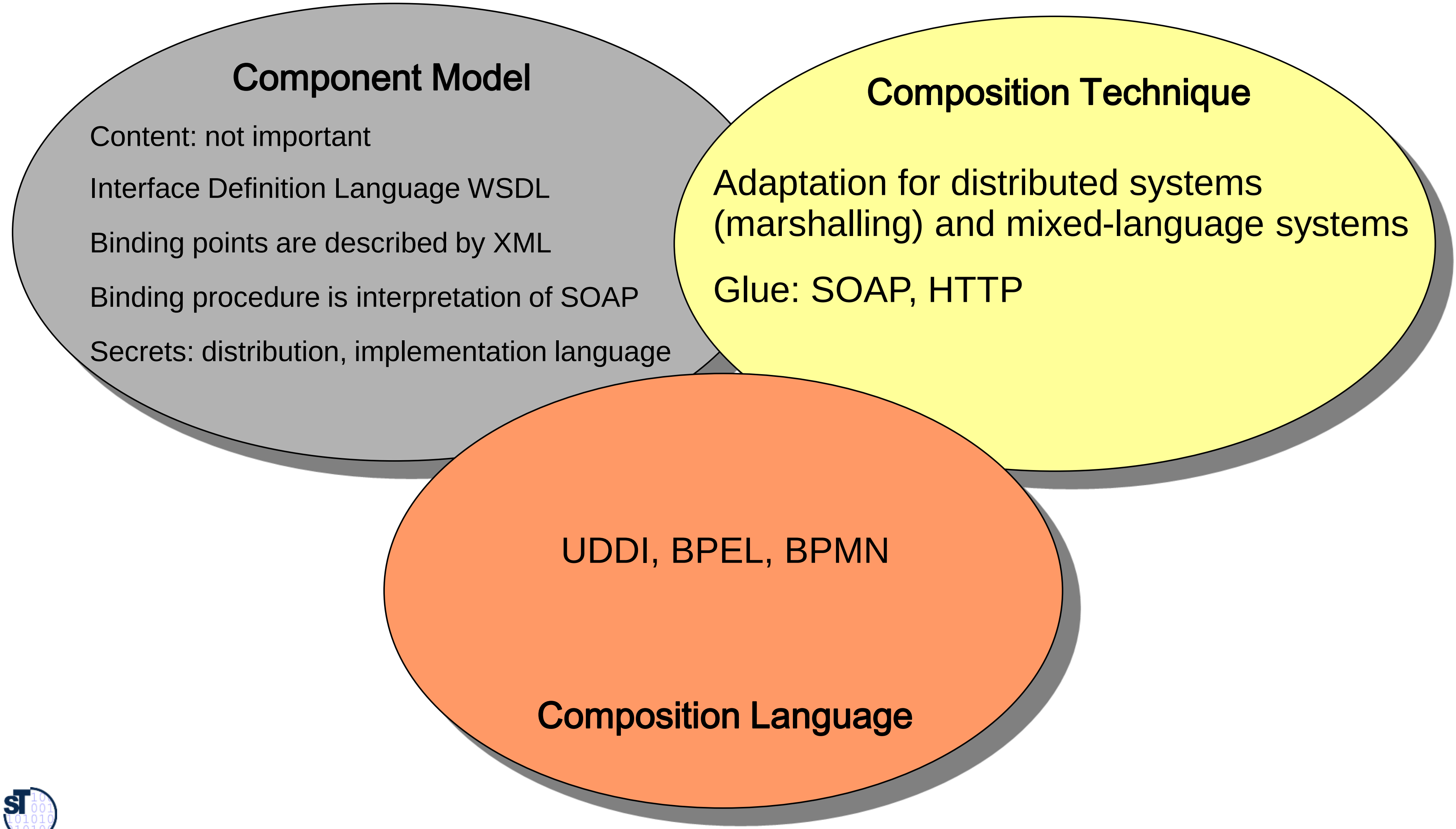


Web Services and their Languages as Specific ADL

- Languages: BPEL, BPMN
- Binding procedure is interpreted, not compiled
- More flexible than binary connectors:
  - When interface changes, no recompilation and rebinding
  - Protocol-independent



Web Services as Composition System



Component Model

Content: not important

Interface Definition Language WSDL

Binding points are described by XML

Binding procedure is interpretation of SOAP

Secrets: distribution, implementation language

Composition Technique

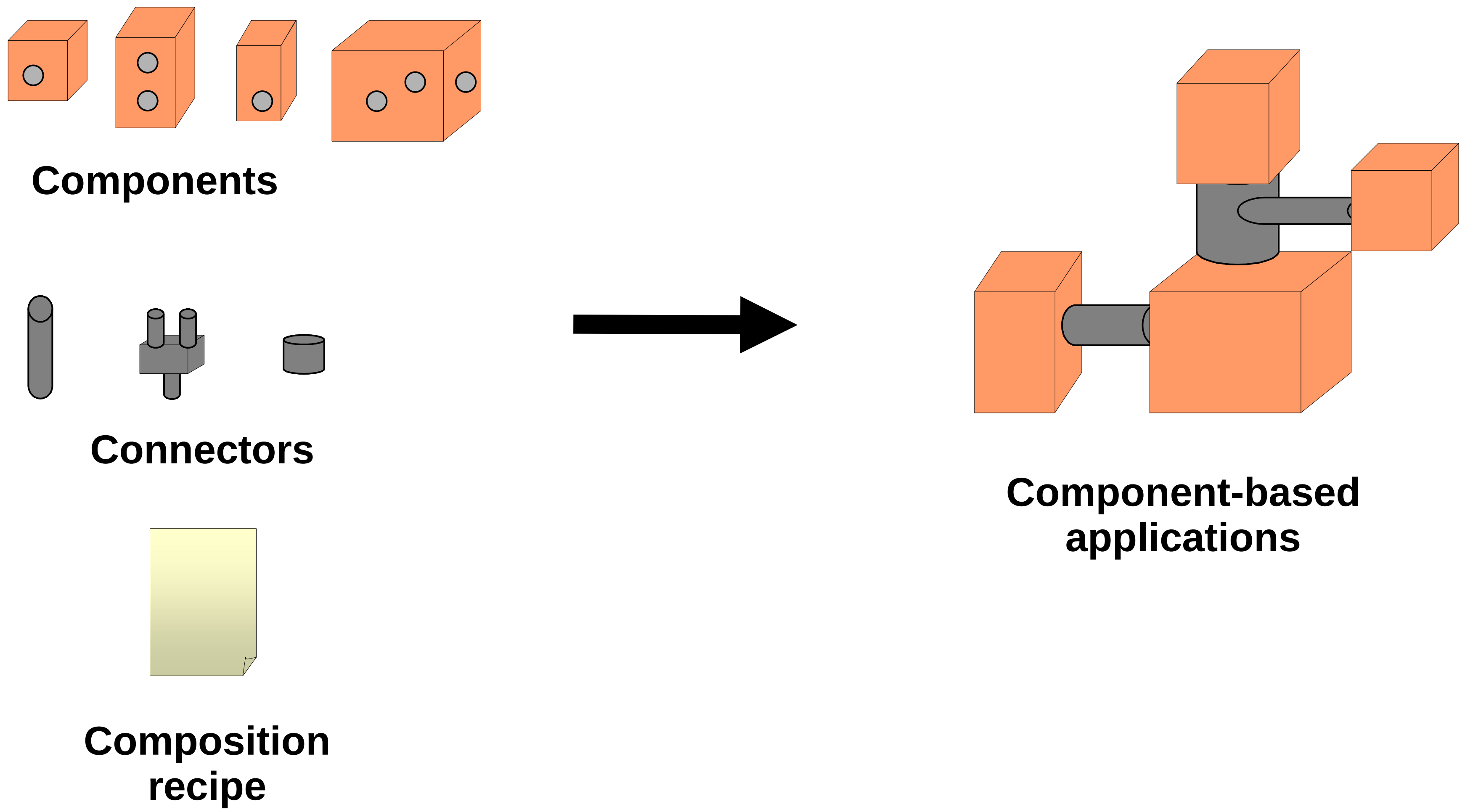
Adaptation for distributed systems (marshalling) and mixed-language systems

Glue: SOAP, HTTP

UDDI, BPEL, BPMN

Composition Language

Black-Box Composition



The Essence of Black-Box Composition

- 3 Problems in System construction
  - Variability
  - Extensibility
  - Adaptation
- In “Design Patterns and Frameworks”, we learned about design patterns to tackle these problems
- Black-box composition supports variability and adaptation
  - not extensibility

The Ladder of Composition Systems

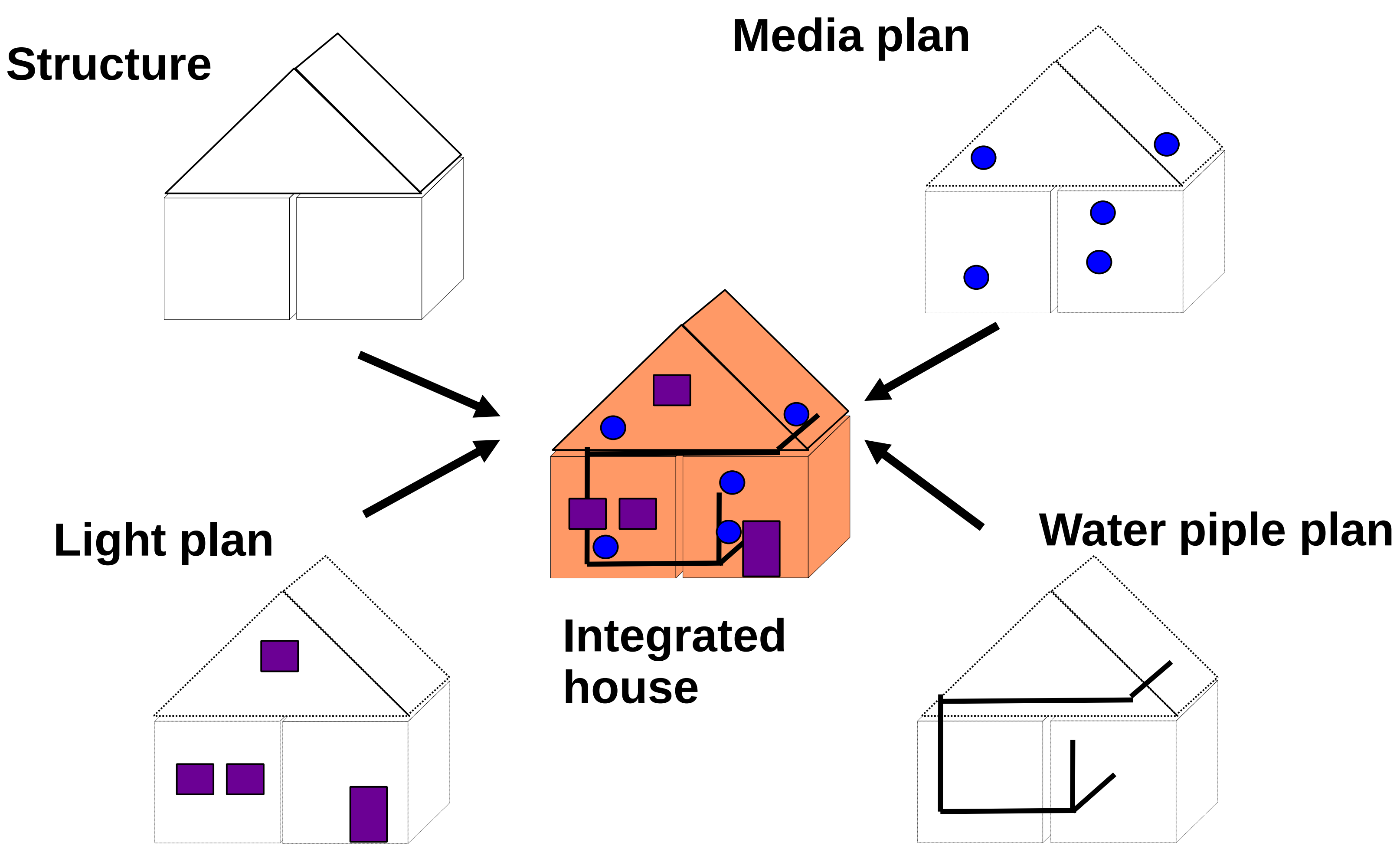
Software Composition Systems	Composition Language	*Invasive Composition Piccola Gloo*
Aspect Systems	Aspect Separation Crosscutting	*Aspect/J AOM*
View Systems	Composition Operators	*Composition Filters Hyperspaces*
Architecture Systems	Architecture as Aspect Connectors	*Darwin BPMN HRC*
Classical Component Systems	Standard Components Reflection	*.NET CORBA Beans EJB ArchJava*
Object-Oriented Systems	Objects as Run-Time Components	*C++ Java UML components*
Modular Systems	Modules as Compile-Time Components	*Shell scripts Modula Ada-85*

10.3 Gray-box Component Models

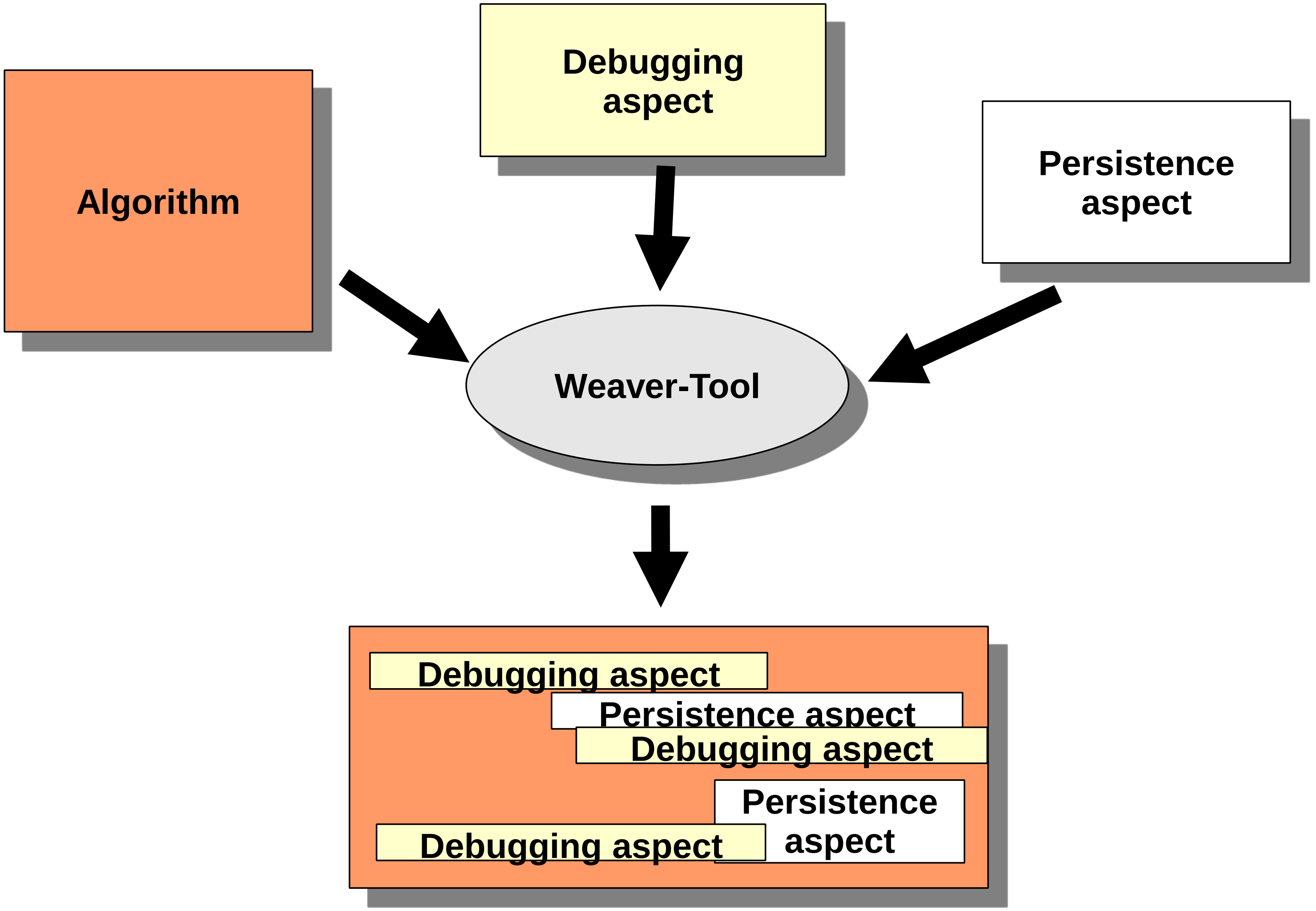
Grey-Box Component Models: The Development of the Last Years

- **View-based Programming**
  - **Component merge (integration)**
  - **Component extension**
- **Aspect-oriented Programming**
  - **Views can cross-cut components**
  - **Component distribution**

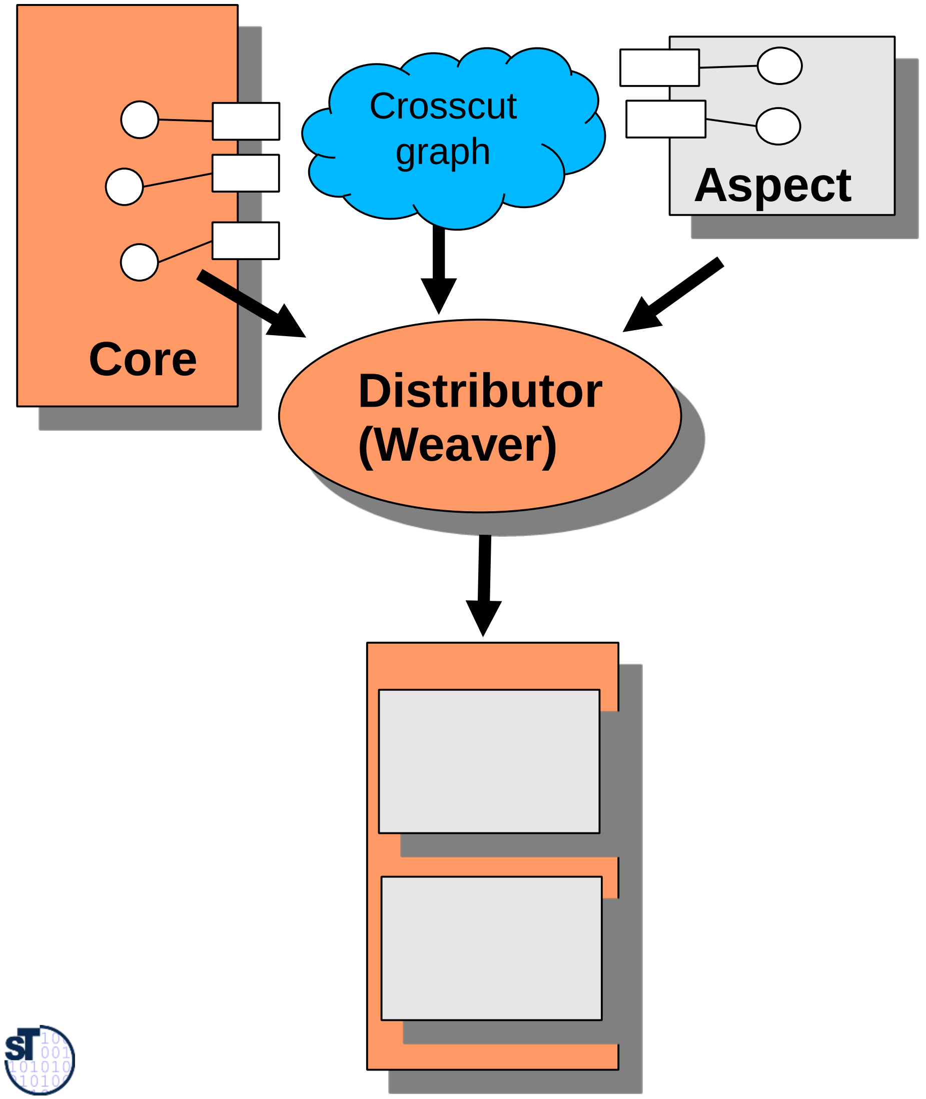
Aspects in Architecture



Aspects in Software

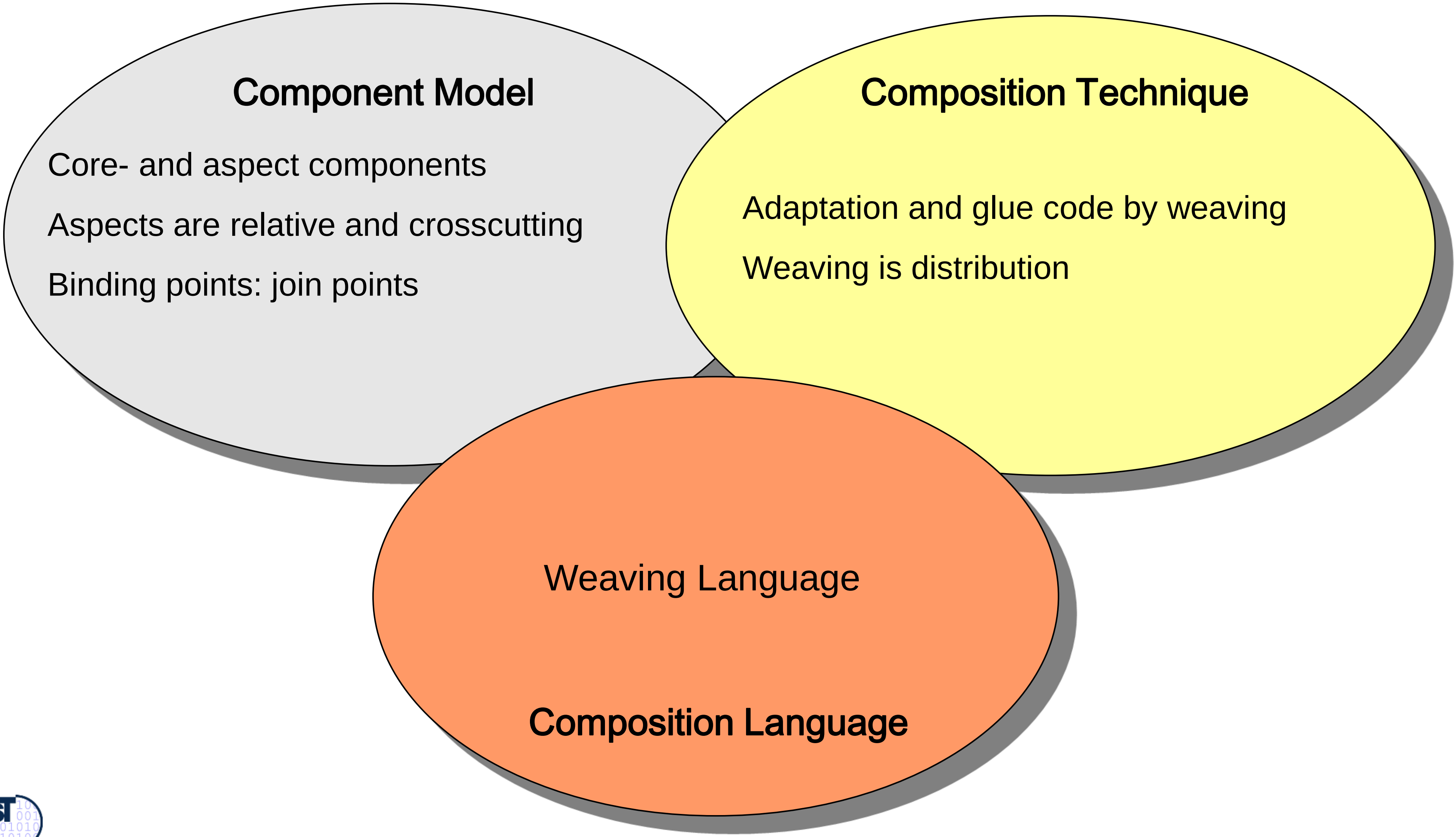


Aspect Weavers Distribute Advice Components over Core Components



- Aspects are *crosscutting*
- Hence, aspect functionality must be *distributed* over the core
- The distribution is controlled by a crosscut graph

Aspect Systems As Composition Systems


Component Model
Core- and aspect components
Aspects are relative and crosscutting
Binding points: join points
Composition Technique
Adaptation and glue code by weaving
Weaving is distribution
Weaving Language
Composition Language

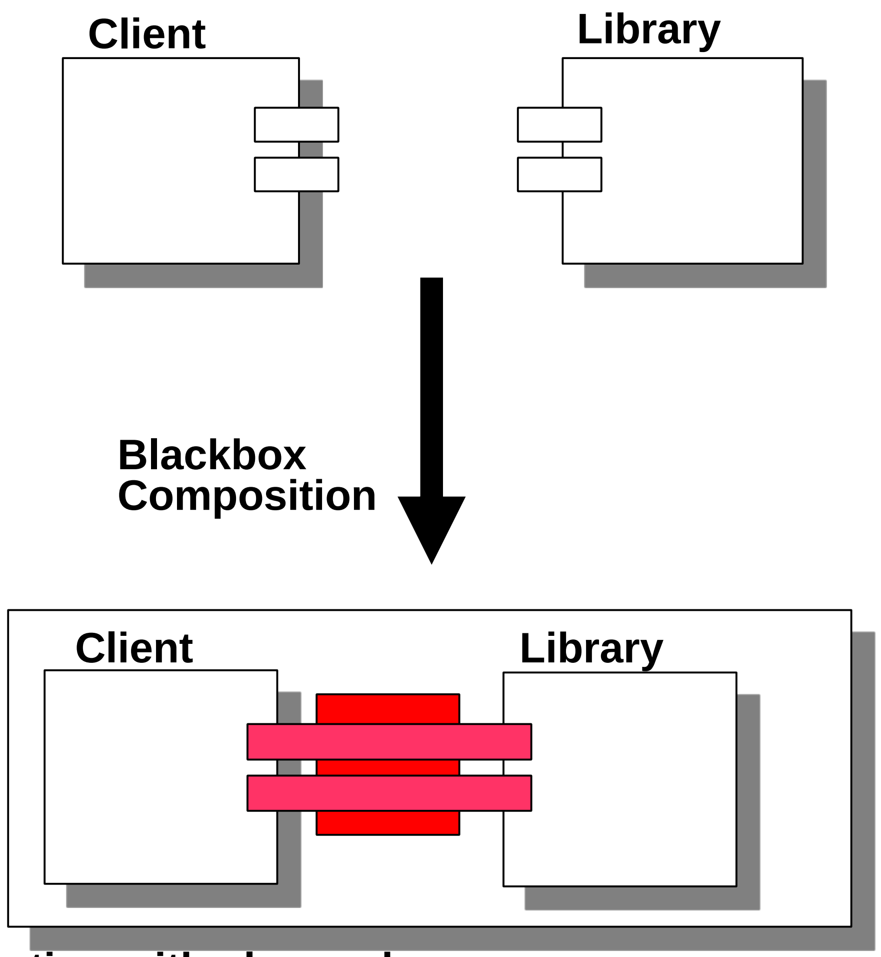
10.3.1 Full-Fledged Composition Systems

Composition Systems

- All the following composition systems support full black-box and grey-box composition, as well as full-fledged composition languages:
  - Composition filters [Aksit,Bergmans]
  - Hyperspace Programming [Ossher et al., IBM]
  - Invasive software composition (ISC) [Aßmann]

Connectors are Composition Operators

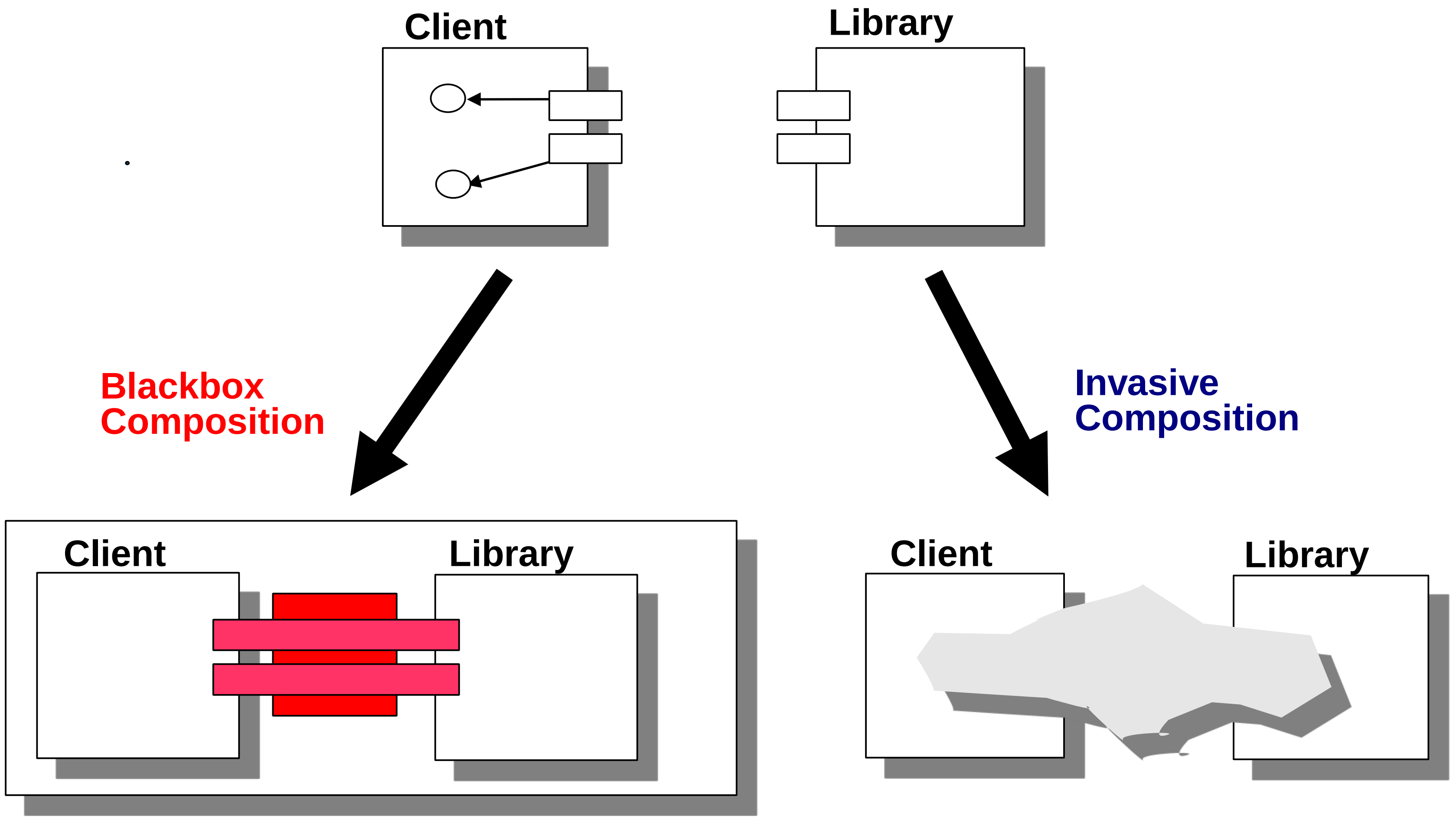
- Usually, connectors connect (glue) black-box components for communication



Blackbox connection with glue code

Connectors can be Grey-Box Composition Operators

- Connectors can work invasively, i.e., adapt components inside



Blackbox connection with glue code

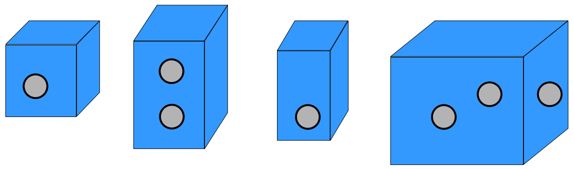
Grey-box (Invasive) Connection

Composition Languages in Composition Systems

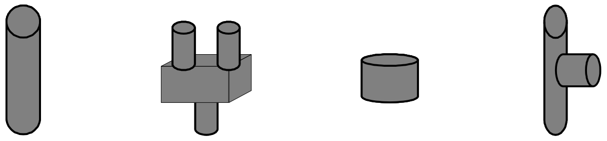
- Composition languages describe the structure of the system in-the-large ("programming in the large")
  - Composition programs combine the basic composition operations of the composition language

- Composition languages can look quite different
  - Imperative or rule-based
  - Textual languages
    - Standard languages, such as Java
    - Domain-specific languages (DSL) such as Makefiles or ant-files
  - Graphic languages
    - Architectural description languages (ADL)
- Composition languages enable us to describe large systems

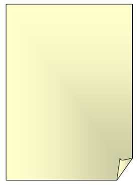
Composition Process in Grey-Box Composition Systems



Grey-box Components

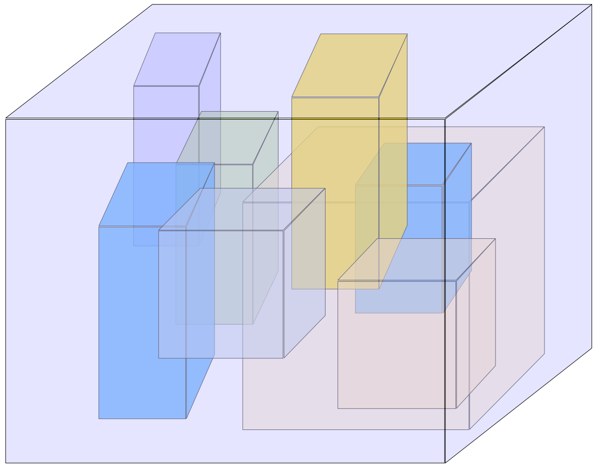


Composition Operators



Composition Recipe

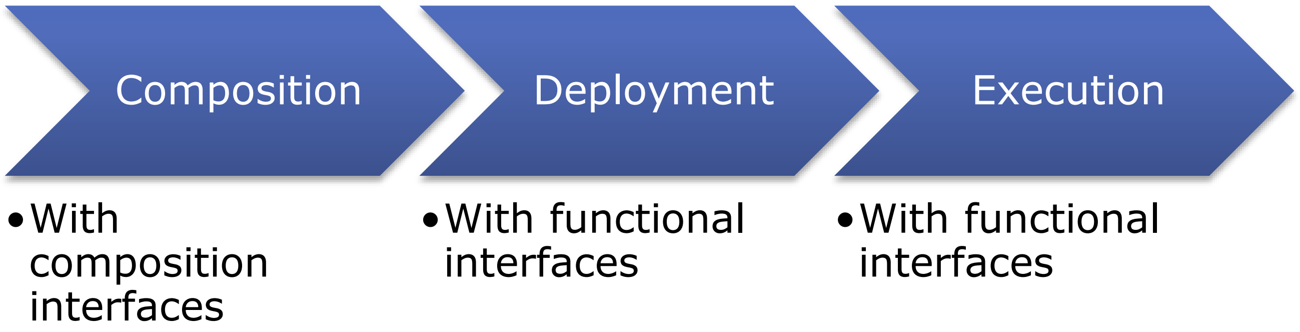
Invasive Software Composition



System Constructed with an Invasive Architecture

Conclusions for Composition Systems

- Components have a ***composition interface*** with variation and extension points
  - Composition interface is different from functional interface
  - The composition is running usually *before* the execution of the system
  - From the composition interface, the functional interface is derived
- System composition becomes a new step in system build


Composition
Deployment
Execution
• With composition interfaces
• With functional interfaces
• With functional interfaces

The End