# Component-Based Software Engineering (CBSE) 1) Introduction

1. Basics of Composition Systems
2. Historic Approaches to Black-Box Composition
3. Gray-Box Composition
4. Ubiquitous Component Models

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de

13-1.1, 18.04.13

## The Power of Components

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

https://en.wikipedia.org/wiki/Container_ship

## Goals

- Understand what a *composition system* is
  - 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

## Obligatory Reading

- [ISC], Chapter 1, Chapter 2
- Douglas McIlroy's home page
  http://cm.bell-labs.com/who/doug/
- [McIlroy] Douglas McIlroy.  Mass Produced Software Components. In P. Naur and B. Randell, "Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968", Scientific Affairs Division, NATO, Brussels, 1969, 138-155.
  http://cm.bell-labs.com/cm/cs/who/doug/components.txt

# 1.1. 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

- Divide-and-conquer (Alexander the Great)
  - Well known in other disciplines
    - Mechanical engineering (e.g., German VDI 2221)
    - Electrical engineering
    - Architecture
- Outsourcing to component producers
  - Components off the shelf (COTS)
  - Goal:
    - Reuse of partial solutions
    - Easy configurability of the systems: variants, versions, product families
- Mass Produced Software Components [McIlroy]
  - Garmisch 68, NATO conference on software engineering
  - Every ripe industry is based on components, since these allow to manage large systems
  - Components should be produced in masses and composed to systems afterwards

# Mass-produced Software Components

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.

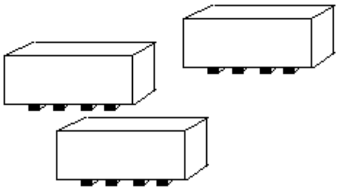
Yet this fragile analogy is belied when we seek for analogues of other tangible symbols of mass production.
- There do not exist manufacturers of standard parts, much less catalogues of standard parts.
- One may not order parts to individual specifications of size, ruggedness, speed, capacity, precision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

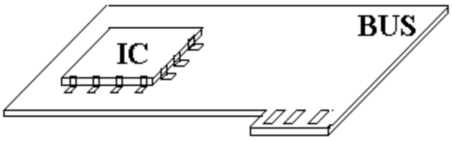
# Mass-produced Software Components

- 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?

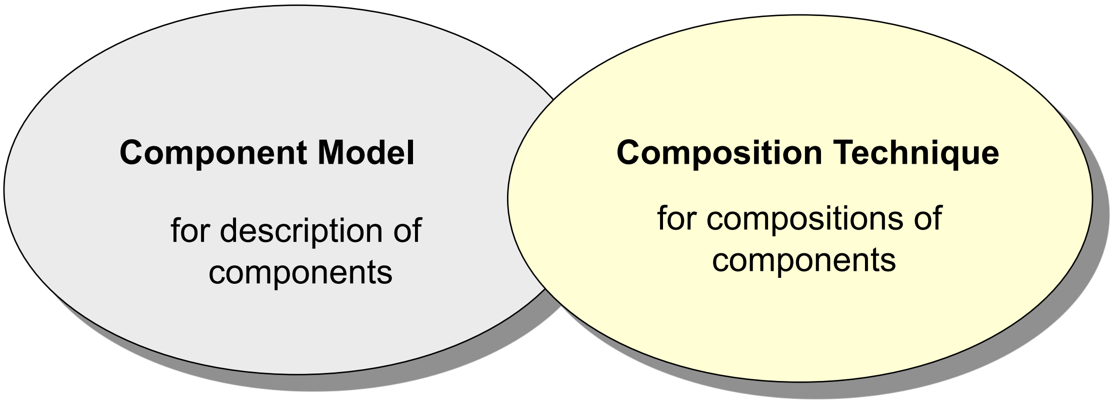
- A component is a *container with*
  - *content* (most often code snippets/fragments)
  - *variation points*
  - *extension points*
  - that are adapted during composition
- A component is a reusable *unit for composition*
- A component underlies *a component model*
  - that fixes the abstraction level
  - 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**

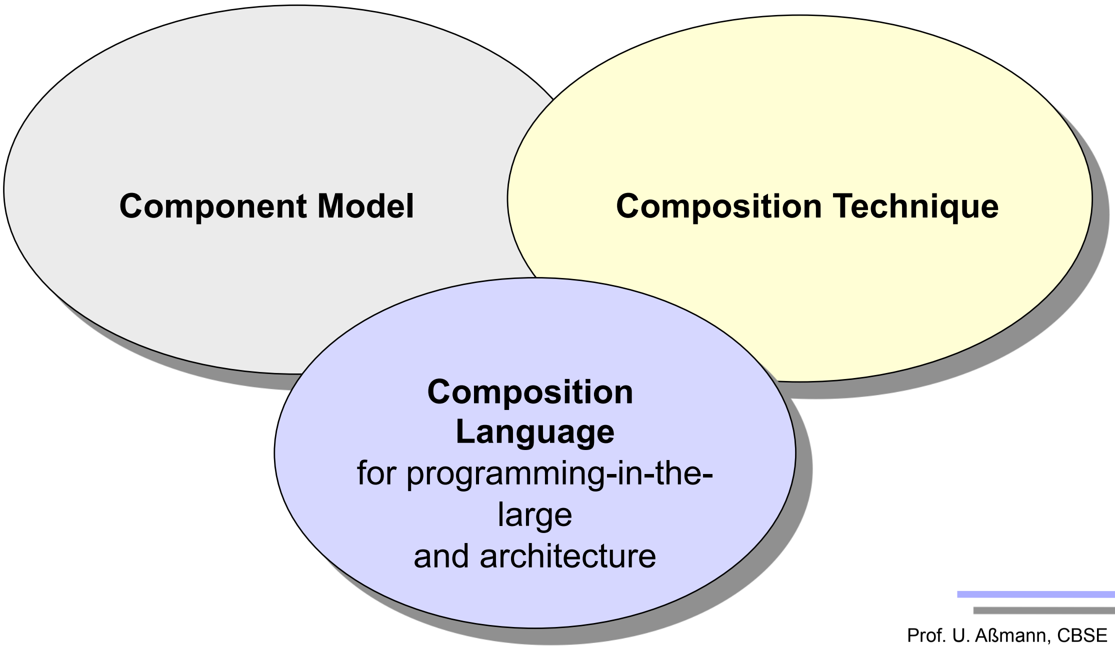
for description of components

**Composition Technique**

for compositions of components

## Composition Systems

- A composition system has

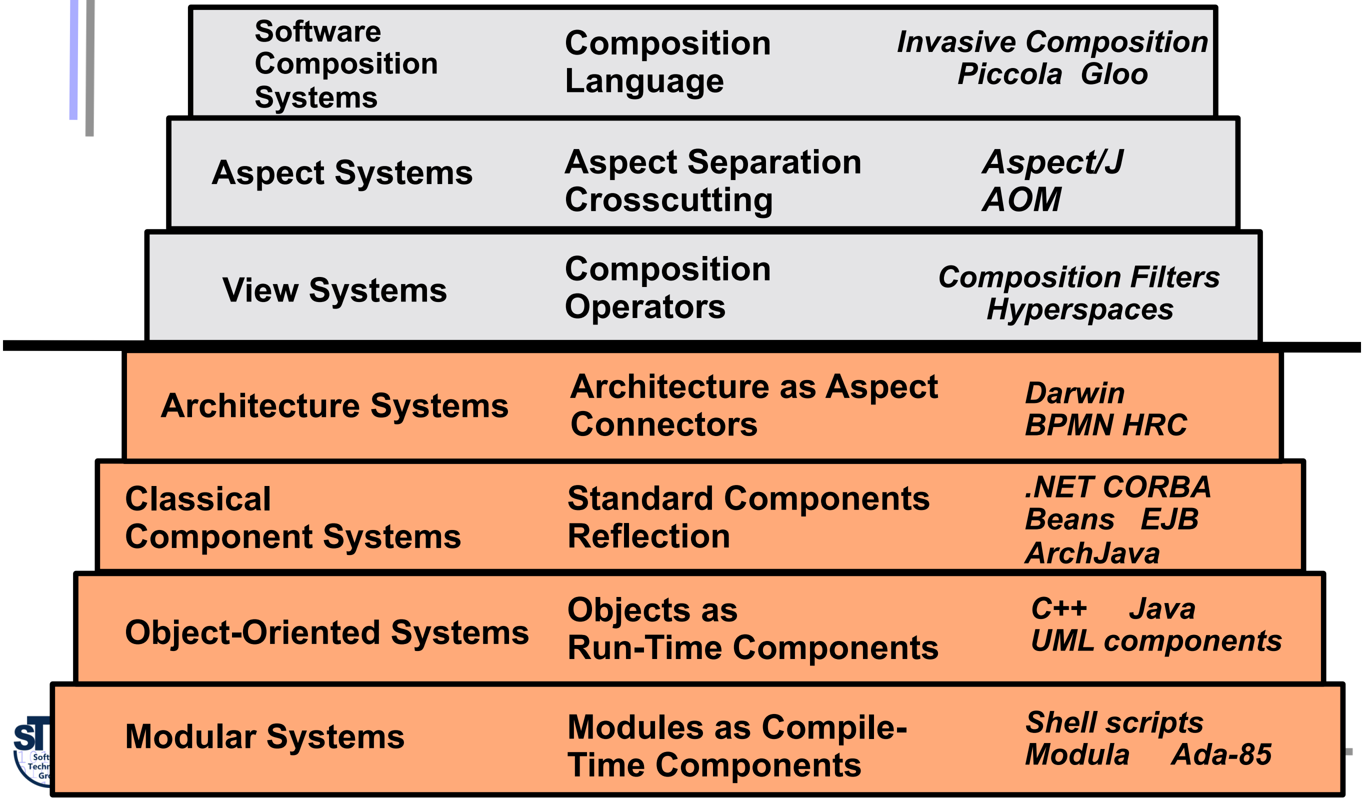


**Component Model**

**Composition Technique**

**Composition Language**
for programming-in-the-large
and architecture

## The Ladder of Composition Systems



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

# Desiderata Component Model

- **CM-M: Modularity**
  - M1 Component interfaces and secrets (information hiding):
    - Explicit specification of interfaces (contact points, exchange points, binding points, variation points, extension points)
    - Explicit specification of dependencies: Provided and required interfaces
    - Location, way of deployment
    - Component lifetime
  - M2  Semantic substitutability (conformance, contracts)
    - CM-M2.1 Syntactic substitutability (typing)
    - CM-M2.2 Functional contracts
    - CM-M2.3 Quality contracts
  - M3 Content
    - Component language metamodel
- **CM-P: Parameterization** of components to their reuse context
  - P1 Generic type parameters
  - P2 Generic program elements
  - P3 Property parameterization
- **CM-S: Standardization**
  - S1 Open standards – or proprietary ones
  - S2 Standard components
  - S3 Standard services

# Desiderata Composition Technique

- **CT-C: Connection and Adaptation**
  - C1: Automatic Component Adaptation: adapt the component interface to another interface
  - C2: Automatic Glueing: Generation of glue code for communication, synchronization, distribution. Consists of a sequence of adaptations
- **CT-E: Extensibility**
  - E1: Base Class Extension: can base classes be extended?
    - E1.1 Generated factories: can factories be generated
    - E1.2 Generated access layers
  - E2: Views. Use-based extensions: Can a use of a component extend the component?
  - E3: Integrated Extensions. Can extensions be integrated?
- **CT-A: Aspect separation**
  - AS1: Aspect weaving: Extension by crosscutting views
  - AS2: Multiple interfaces of a component
- **CT-S: Scalability (Composition time)**
  - SC1: Binding time hiding
  - SC2: Binding technique hiding
- **CT-M: Metamodelling**
  - MM1: Introspection and reflection (metamodel). Can other components be  introspected? The component itself?
  - MM2: Metaobject protocol: is the semantics of the component specified reflectively?
- **CT-I: Tool support** for composition
  - Editors, checkers, validators

# Desiderata Composition Language

- **CL-C: Product Consistency**
  - Variant cleanness: consistent configurations
  - Robustness: absence of run-time exceptions
- **CL-P: Software Process Support**
  - Build management automation
- **CL-M: Meta-composition**
  - Is the composition language component-based, i.e., can it be composed itself?
  - Reuse of architectures
- **CL-A: Architectural styles** (composition styles)
  - Constraints for the composition

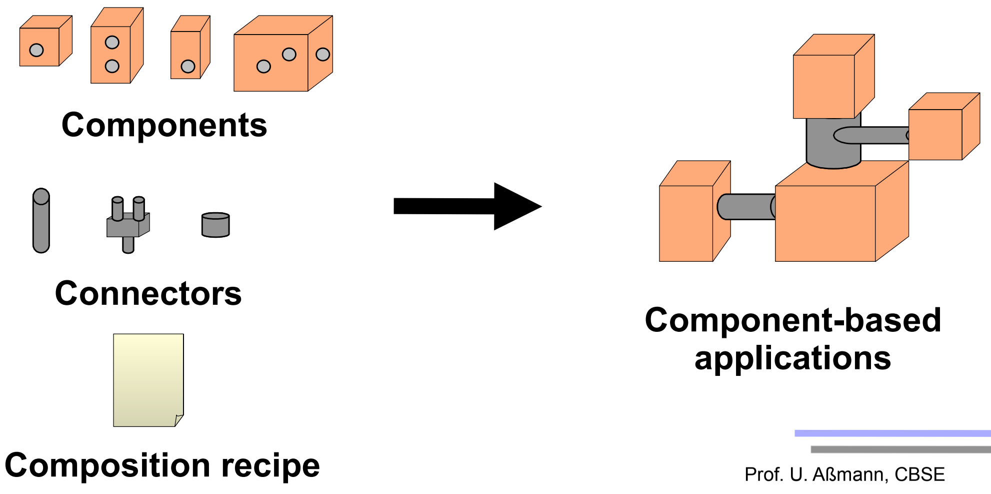
# Service Components

- A *service component* is a software component whose location, style of deployment, and name is not known.
  - It is described by metadata (attributes)
  - [from Greenfield/Short, Software Factories, AWL]

# 1.2 Historical Approaches to Components

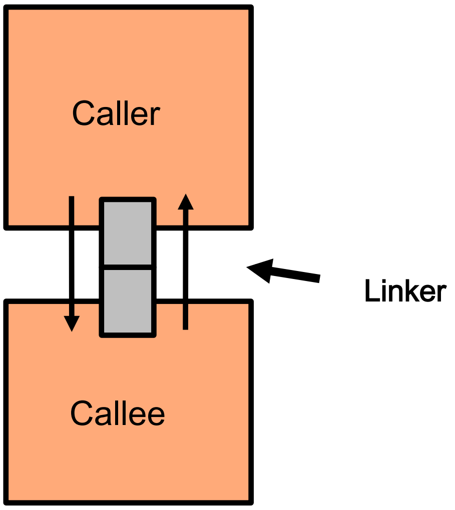
# The Essence of the 60s-90s: LEGO Software with Black-Box Composition

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

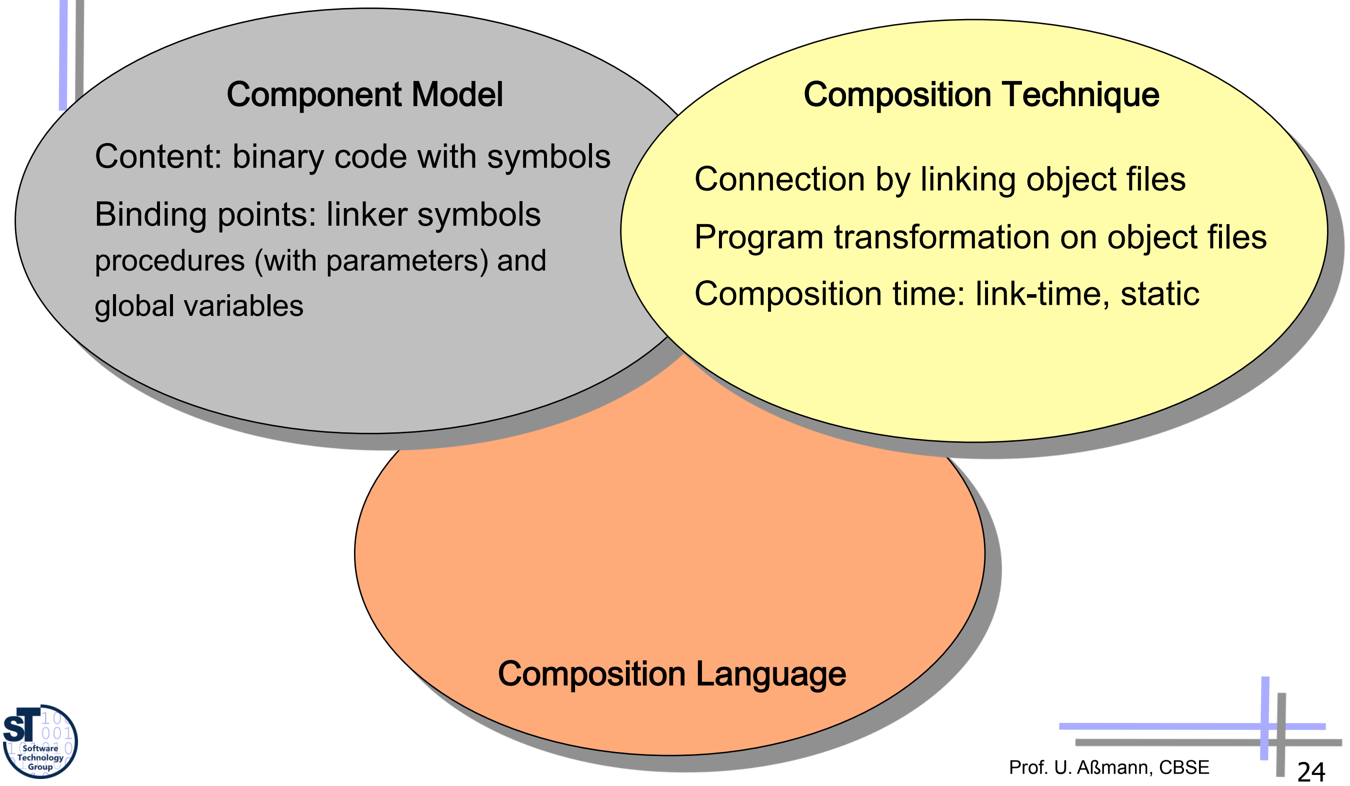


# Procedure Systems

- 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

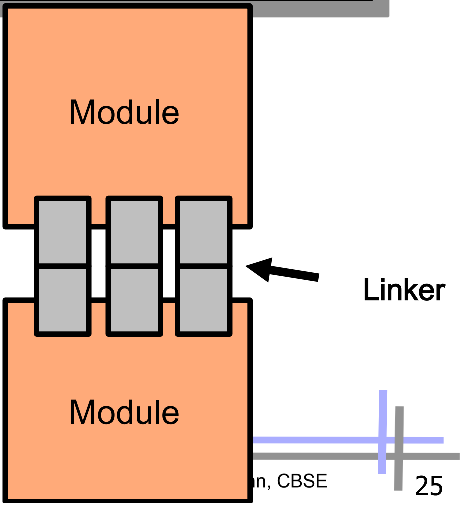
**Composition Language**

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

- Every module hides the 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.

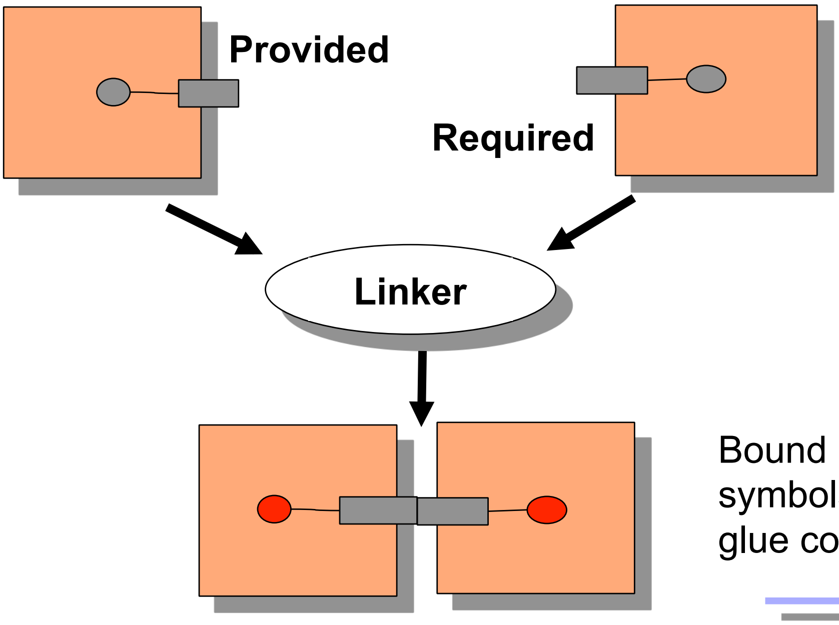
Such modules have rather abstract interfaces which are relatively unlikely to change.



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

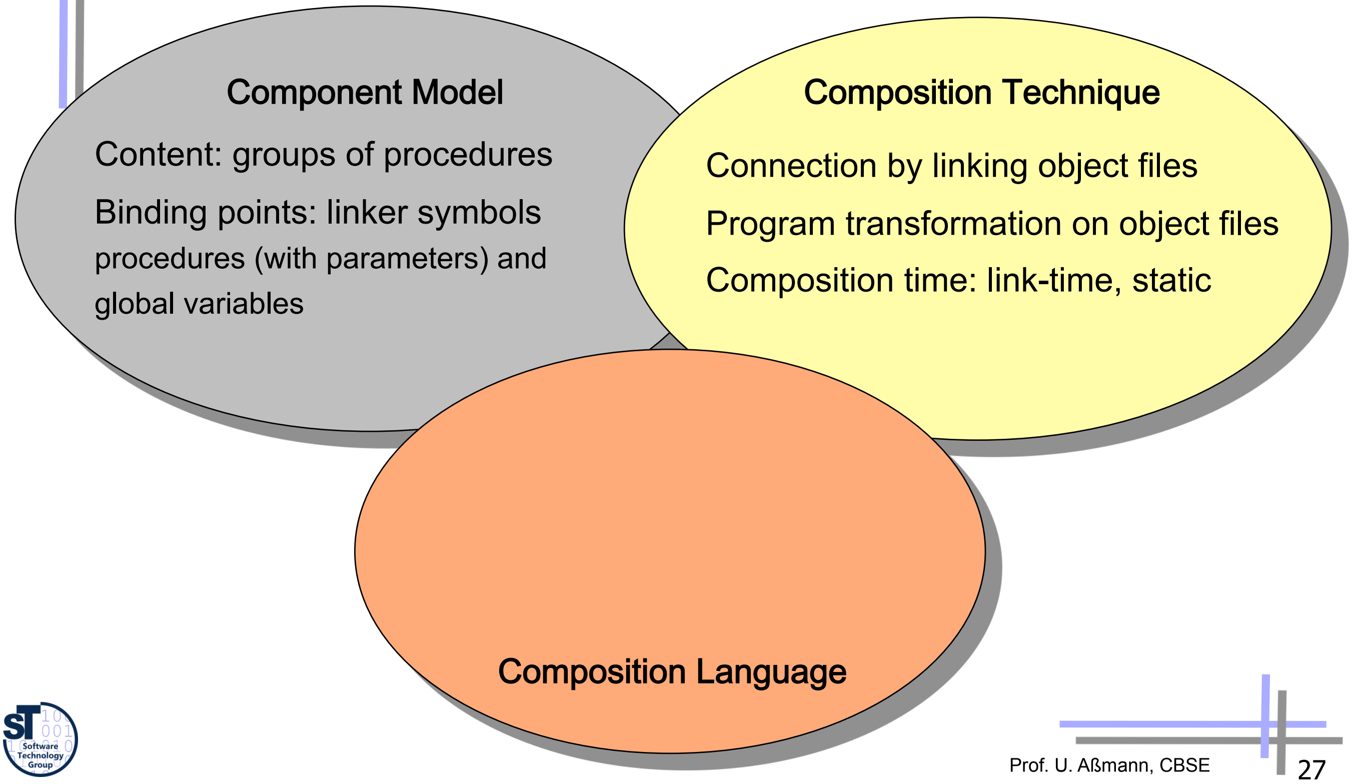
# A Linker is a Static Composition Operator

- 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

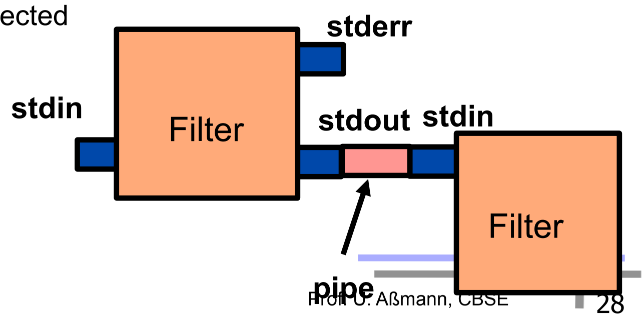
Program transformation on object files

Composition time: link-time, static

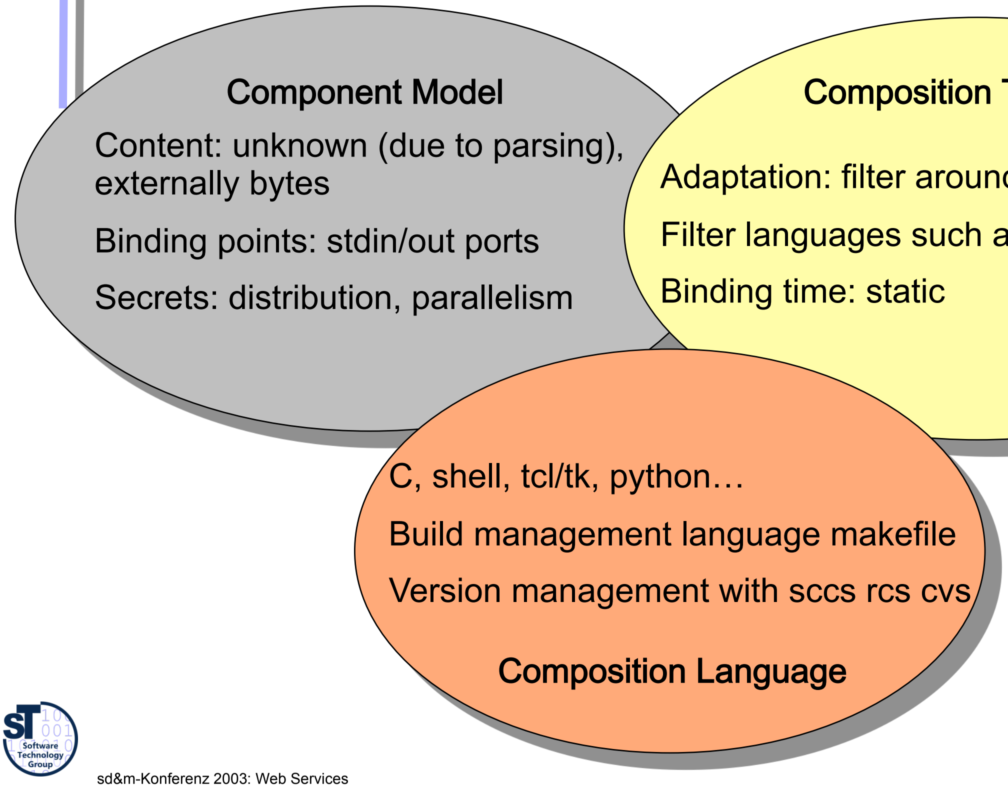
Composition Language

# 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

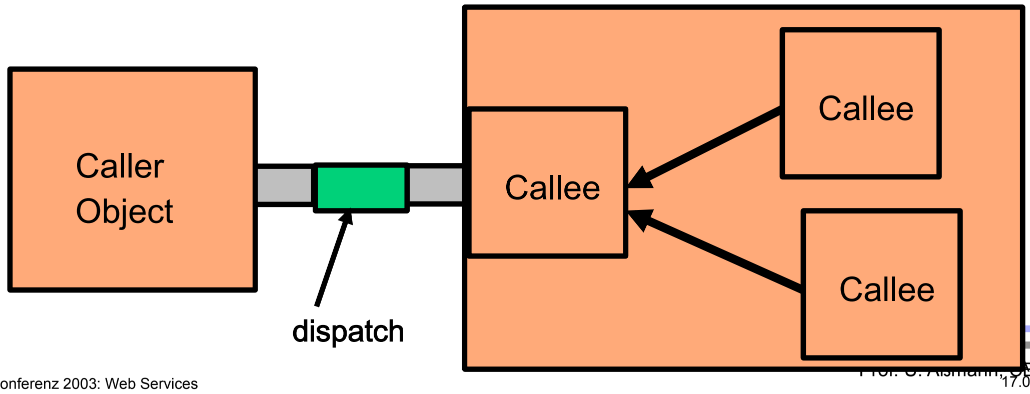
**Composition Language**

# Communication

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

# Object-Oriented Systems

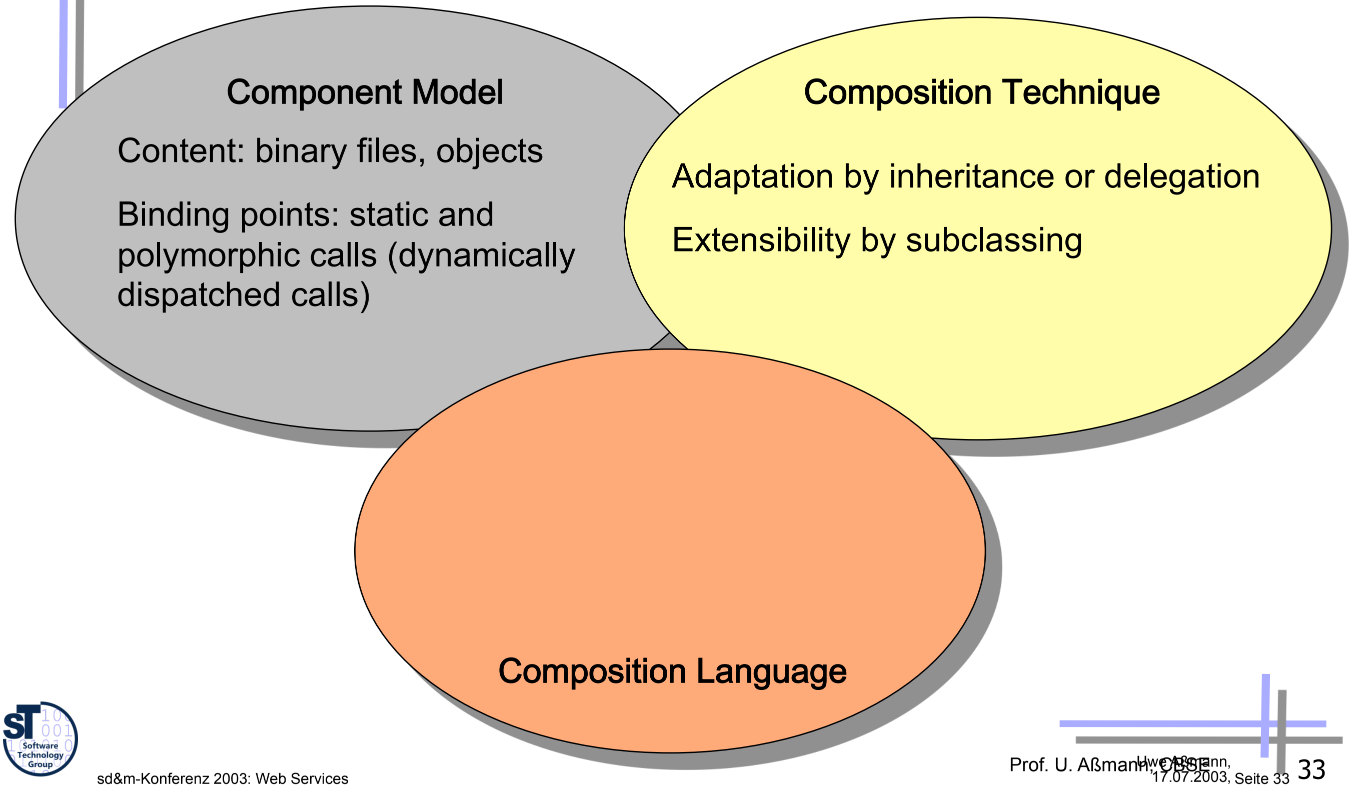
- Components: objects (runtime) and classes (compile time)
  - Objects are instances of classes (modules) with unique identity
  - Objects have runtime state
  - Late binding of calls by search at runtime



# Object-Oriented Systems

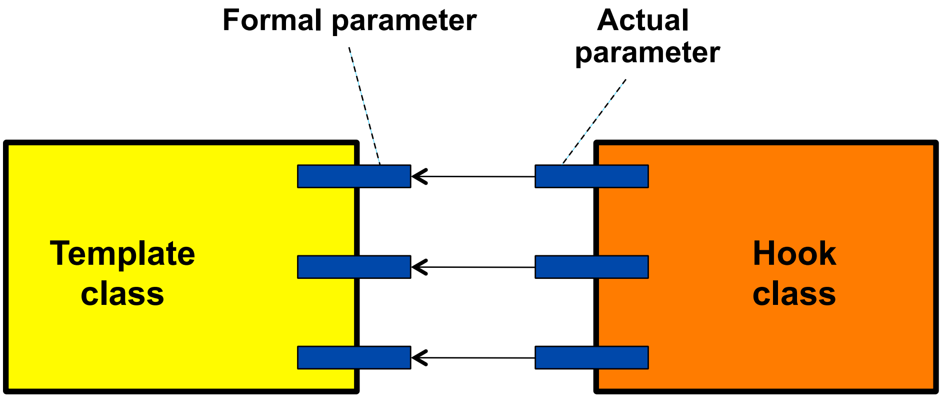
- Component Model
  - Content: code (static) and values (dynamic)
  - Binding points:
    - monomorphic calls (static calls)
    - polymorpic calls (dynamically dispatched calls)
- Composition Technique
  - Adaptation by inheritance or delegation
  - Extensibility by subclassing
- Composition Language: 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

# Object-Oriented Systems: Frameworks

- [Pree] An object-oriented framework consists of a set of template classes which can be parameterized by *hook classes (parameter classes)*
- This principle can be transferred to many other composition systems

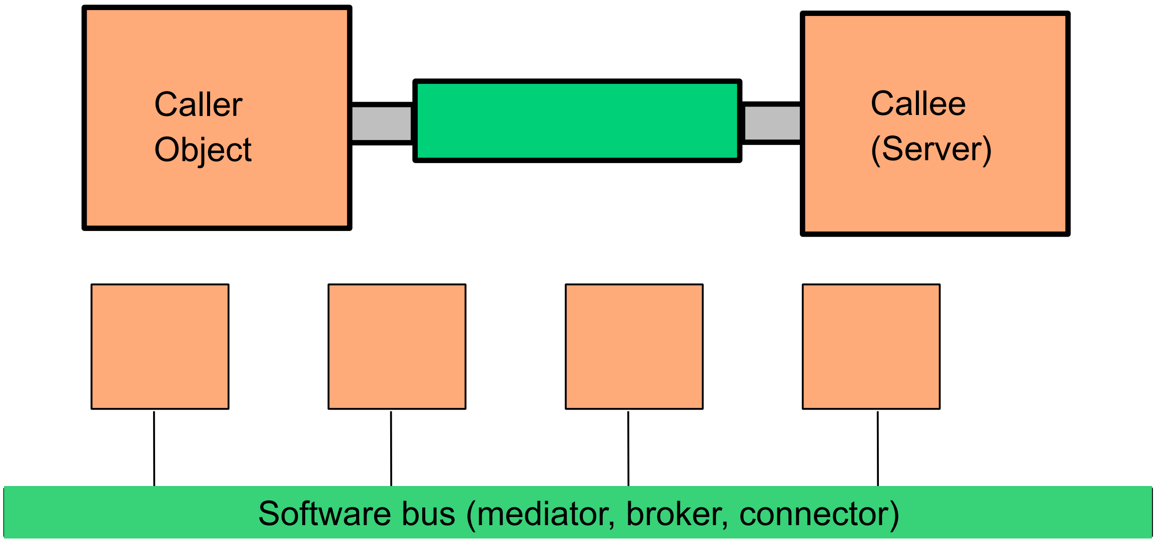


# O-O Frameworks

- Component Model
  - Binding points: Hot spots to exchange the parameter classes (sets of polymorphic methods)
    - Variation points: 1 out-of n choice
    - Extension points: arbitrarily many extensions
- Composition Technique
  - Same as OO
- Compostion language
  - Same as OO

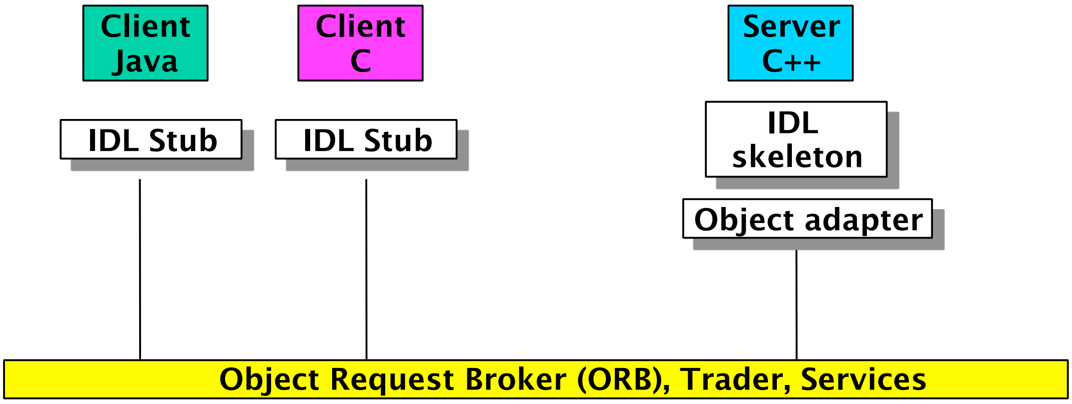
# Commercial Component Systems (COTS, Components off the Shelf)

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



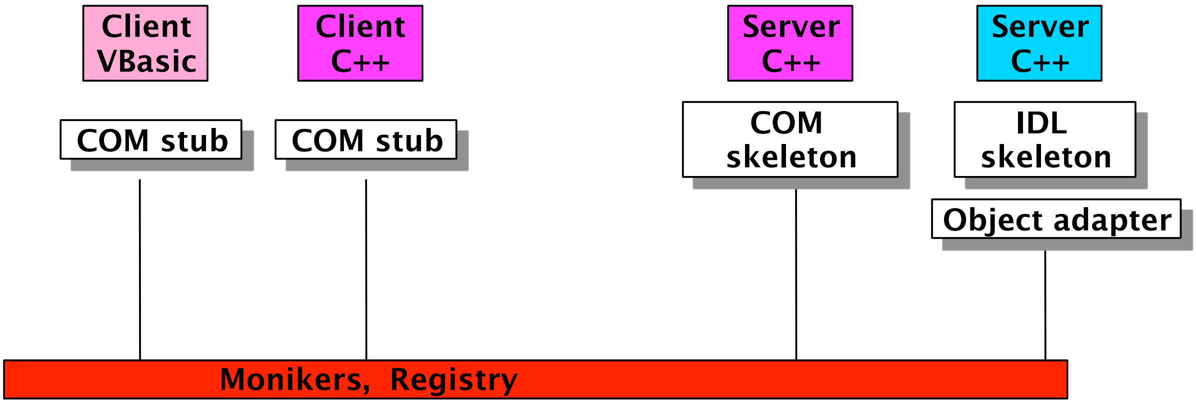
## CORBA
## http://www.omg.org/corba

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



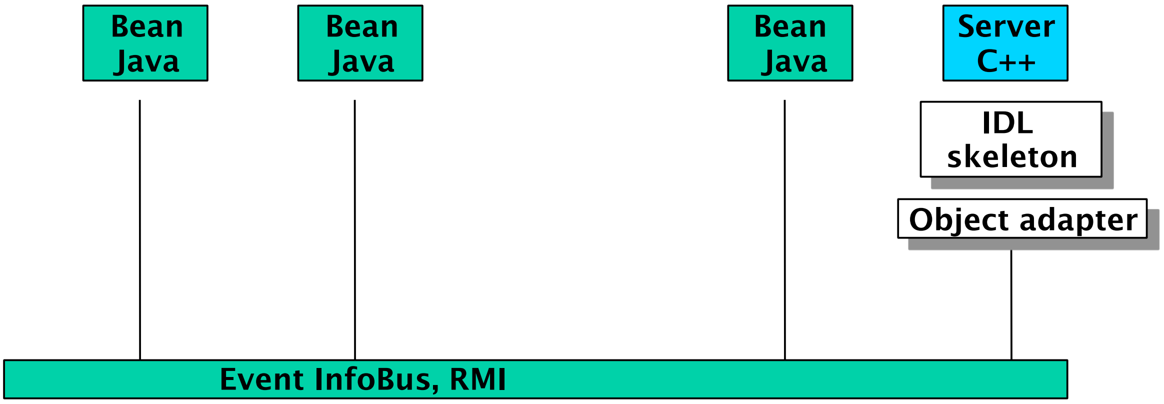
## (D)COM(+), ActiveX
### http://www.activex.org

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



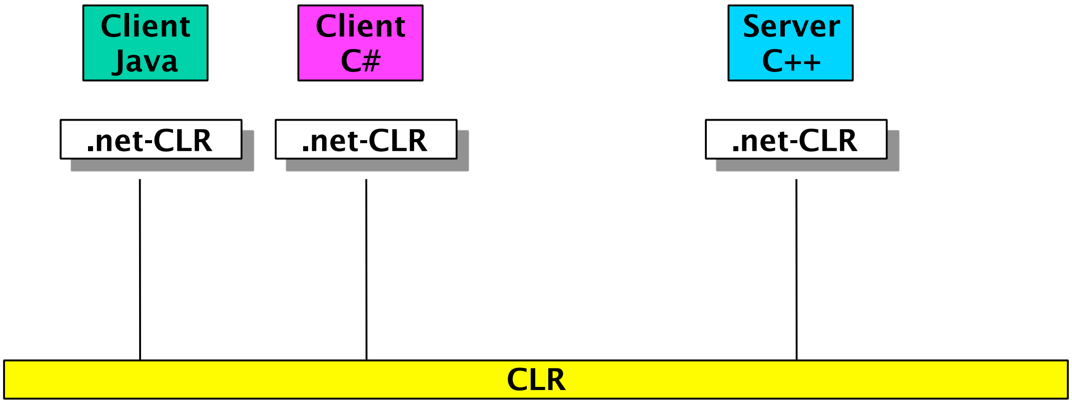
## Java Enterprise Beans

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



## .NET
## http://www.microsoft.com

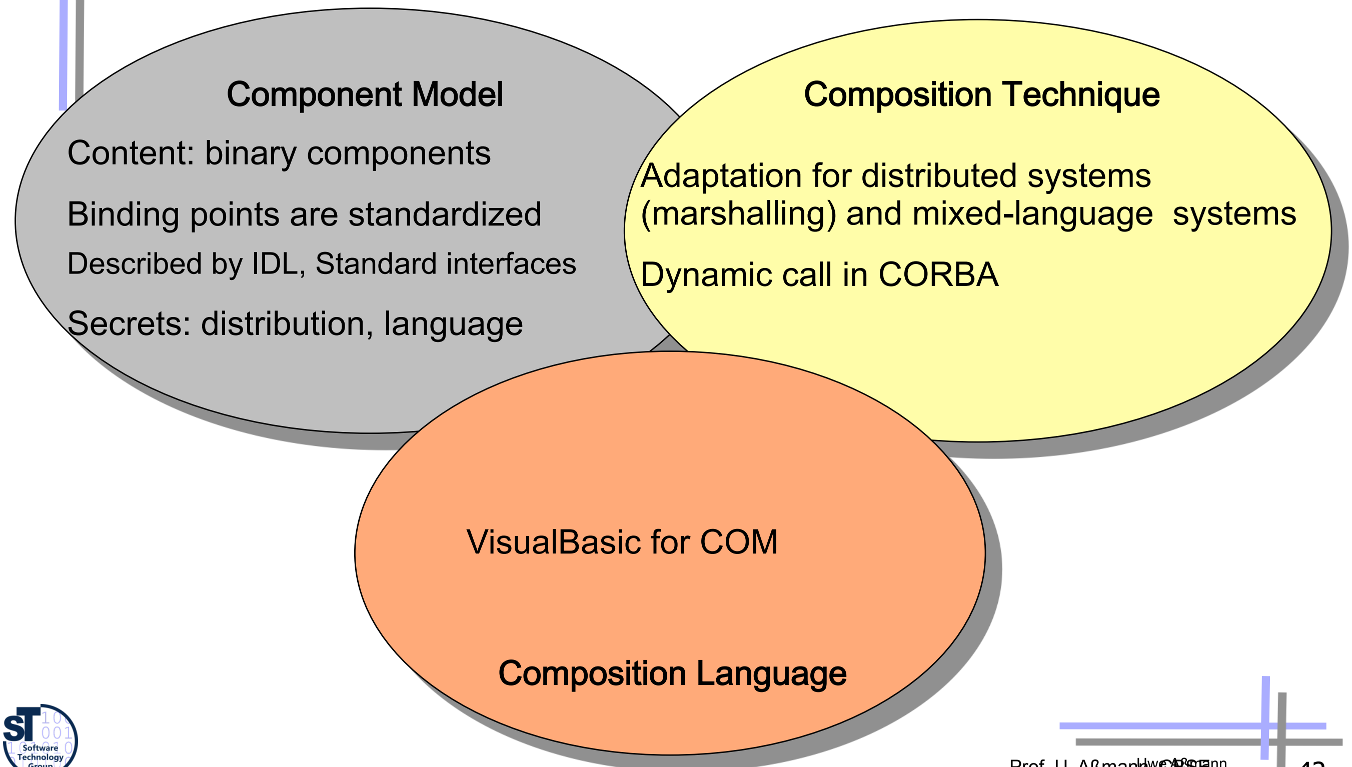
- Language independent, distribution transparent
- NO interface definition language IDL (at least for C#)
- source code or bytecode MSIL
- Common Language Runtime CLR



# 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

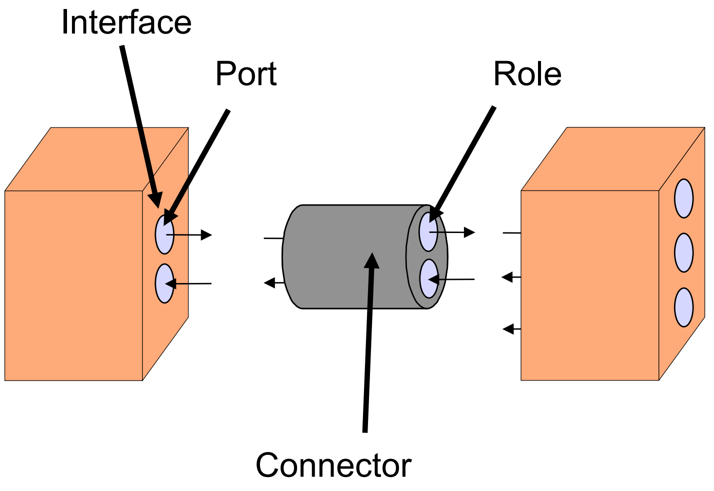
**Composition Language**

# Architecture Systems

- Unicon, ACME, Darwin, Reo
  - feature an Architecture Description Language (ADL)
- 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

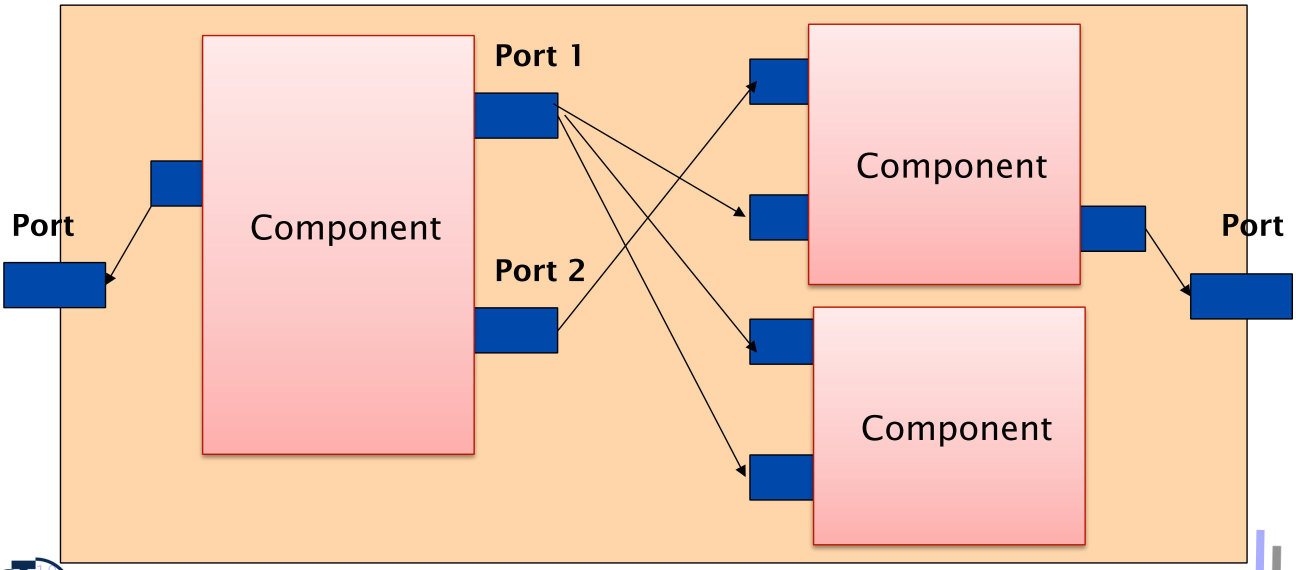
# Component Model in Architecture Systems

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



# Architecture can be exchanged independently of components

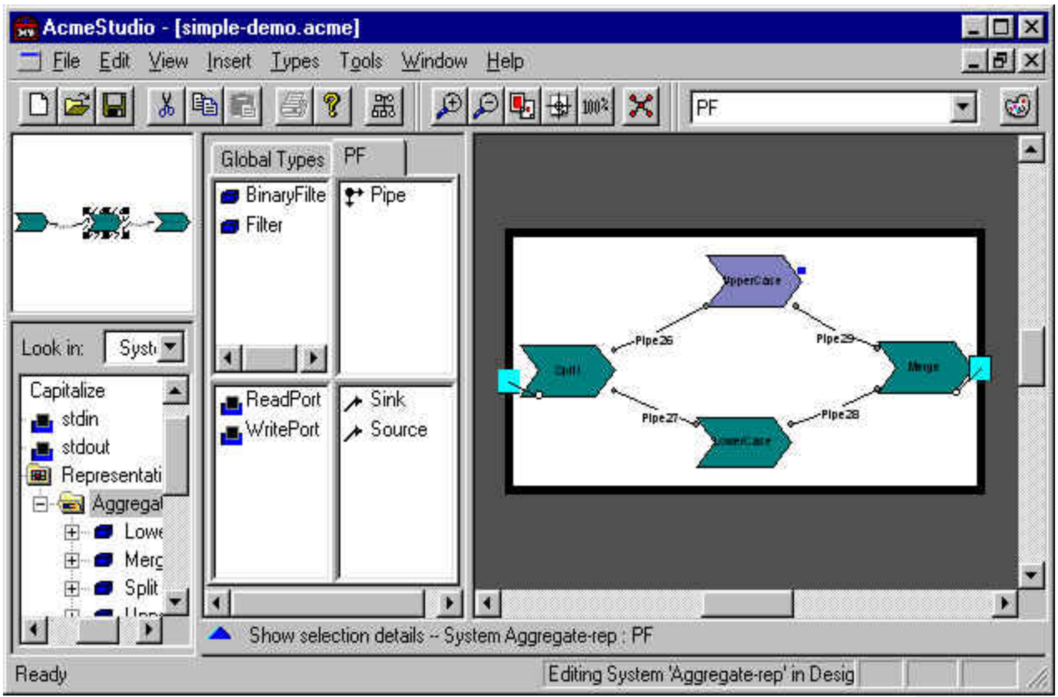
- Reuse of components and architectures is fundamentally improved



# The Composition Language: ADL

- Architecture language (architectural description language, ADL)
  - ADL-compiler
  - XML-Readers/Writers for ADL. XADL is a new standard exchange language for ADL based on XML
- Graphic editing of systems
- Checking, analysing, simulating systems
  - Dummy tests
  - Deadlock checkers
  - Liveness checking

# ACME Studio

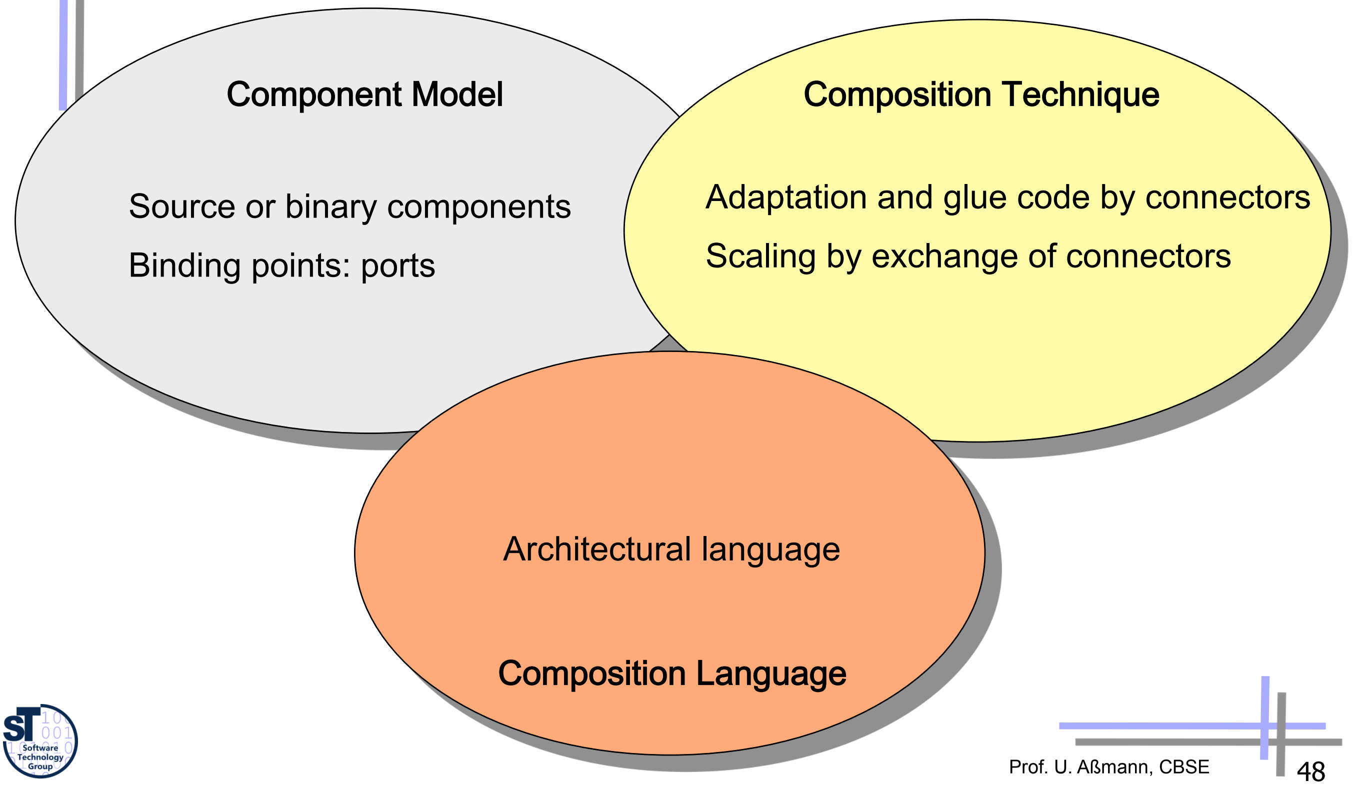


Show selection details -- System Aggregate-rep : PF

Ready

Editing System 'Aggregate-rep' in Desig

# Architecture Systems as Composition Systems



Component Model

Source or binary components

Binding points: ports

Composition Technique

Adaptation and glue code by connectors

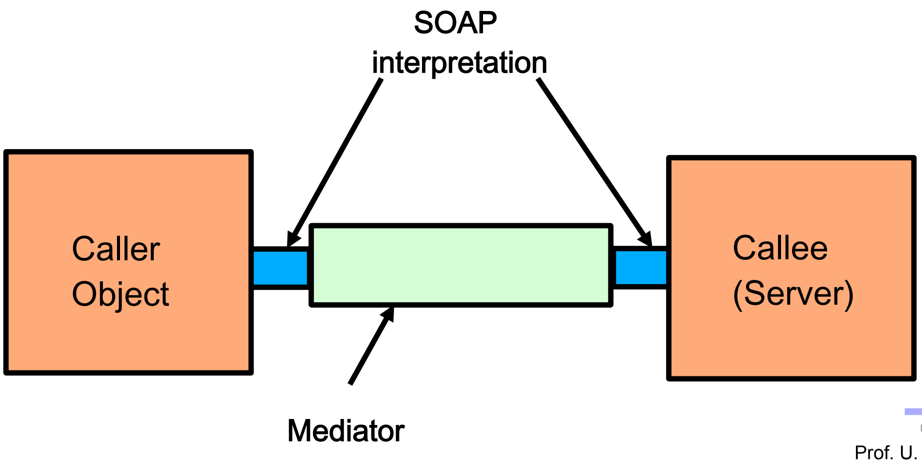
Scaling by exchange of connectors

Architectural language

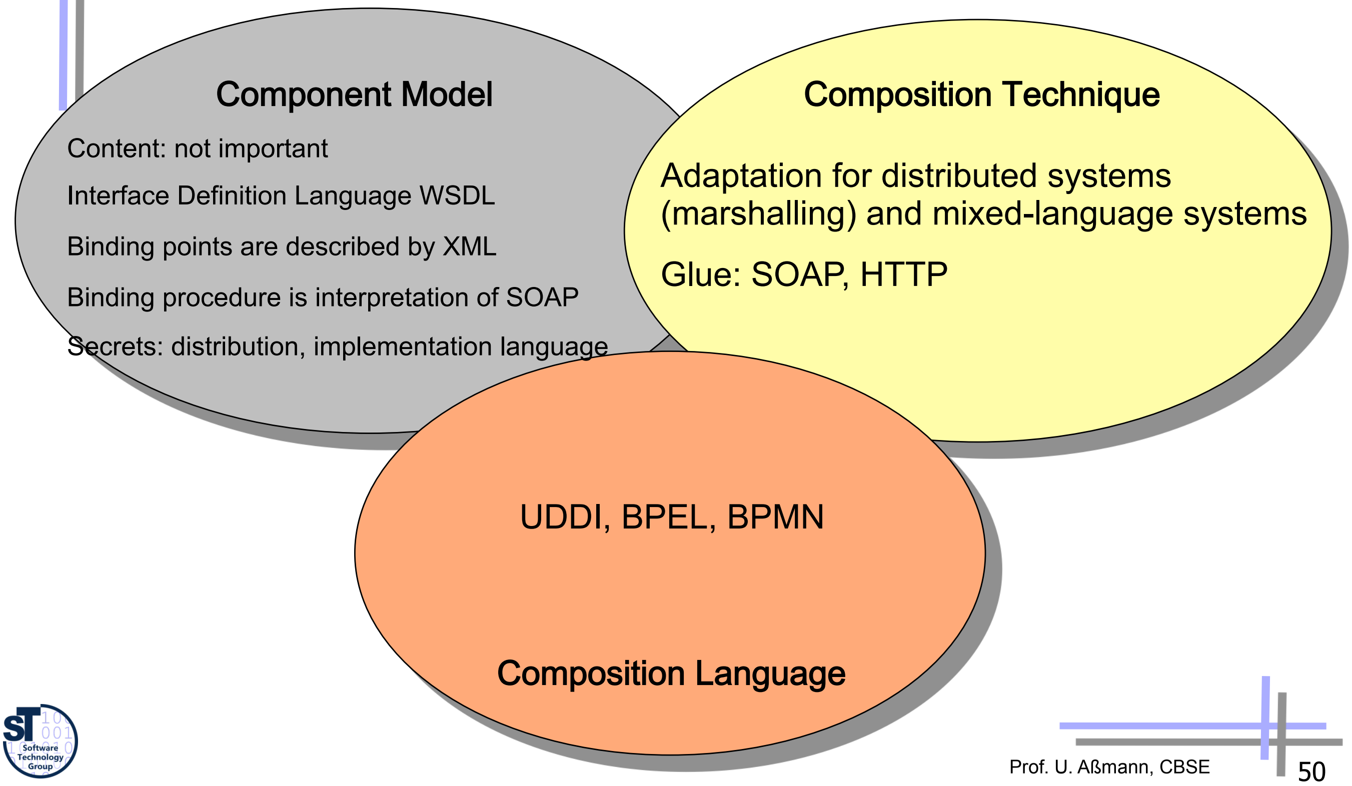
Composition Language

# Web Services and their Languages as Specific ADL

- Languages: BPEL, BPNM
- 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

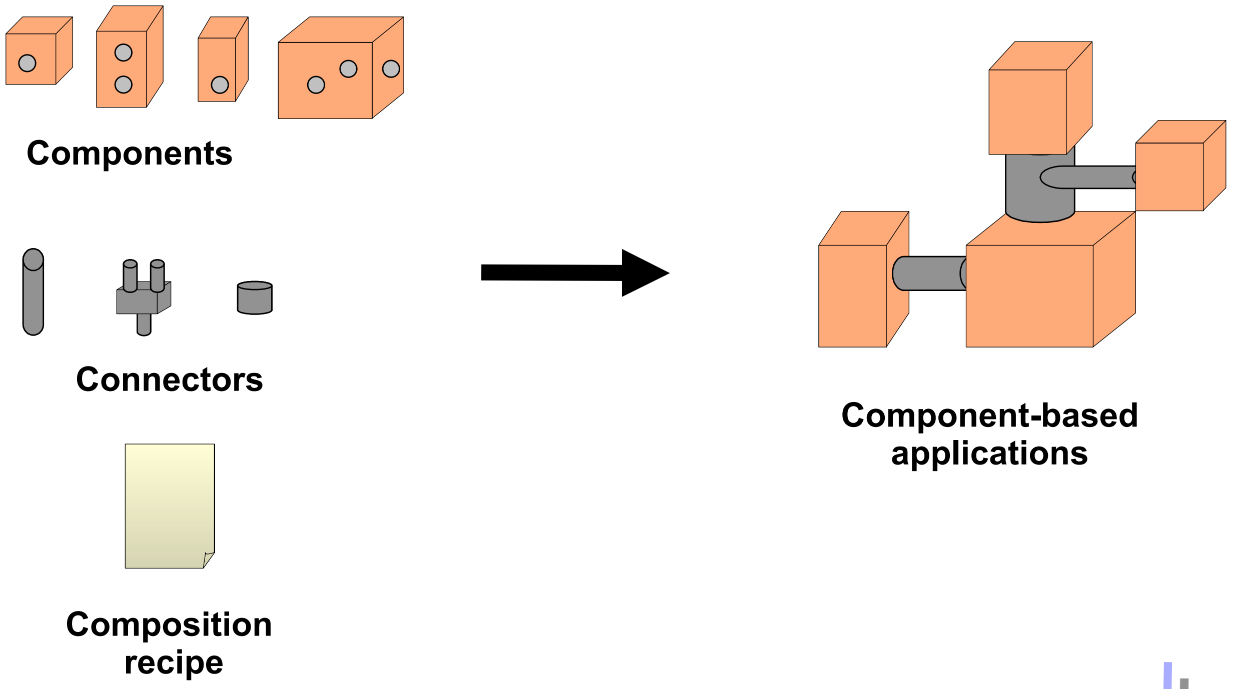
UDDI, BPEL, BPMN

**Composition Language**

# What the Composition Language Offers for the Software Process

- Communication
  - Client can understand the architecture graphics well
  - Architecture styles classify the nature of a system in simple terms (similar to design patterns)
- Design support
  - Refinement of architectures (stepwise design, design to several levels)
  - Visual and textual views to the software resp. the design
- Validation: Tools for consistency of architectures
  - Are all ports bound? Do all protocols fit?
  - Does the architecture corresponds to a certain style? Or to a model architecture?
  - Parallelism features as deadlocks, fairness, liveness,
  - Dead parts of the systems
- Implementation: Generation of large parts of the communications and architecture

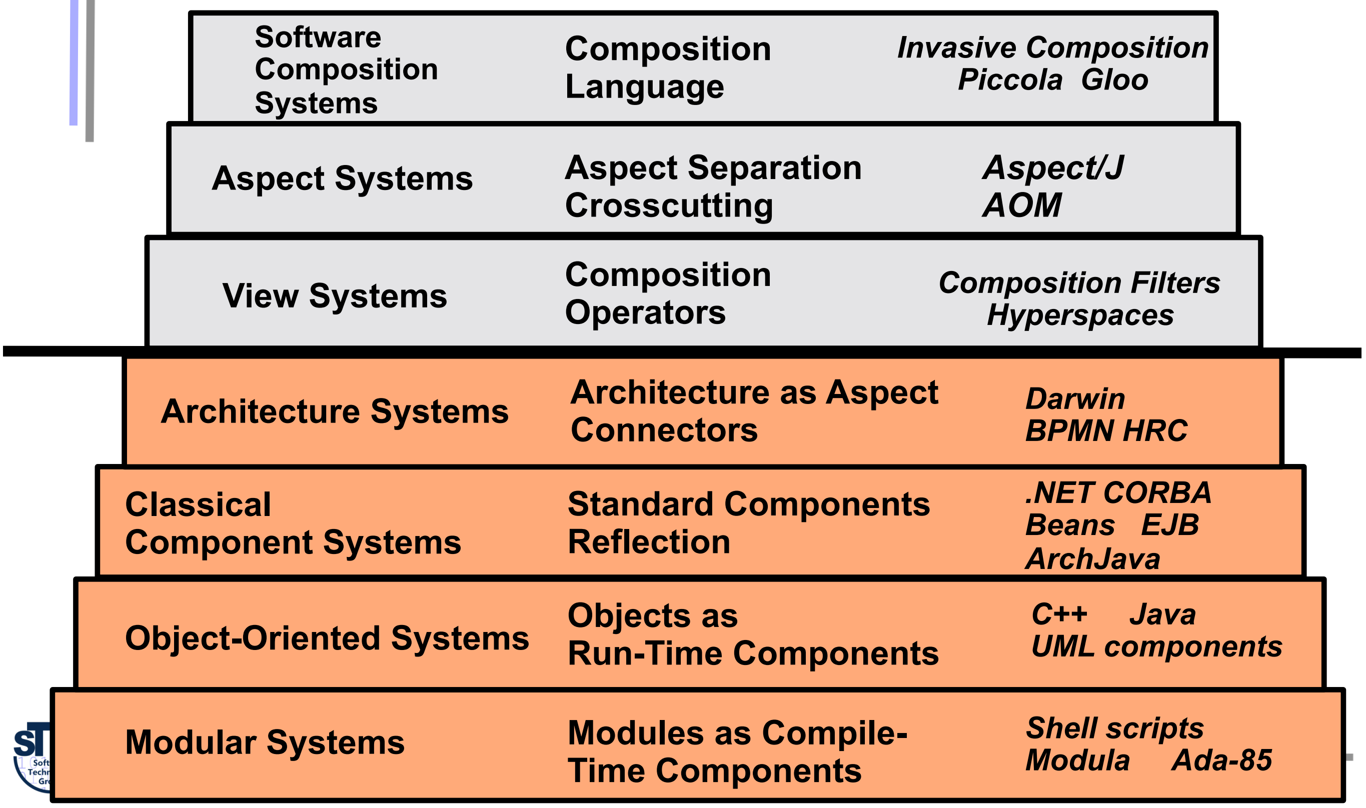
# Black-Box Composition



## 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* |

# 1.3 Gray-box Component Models

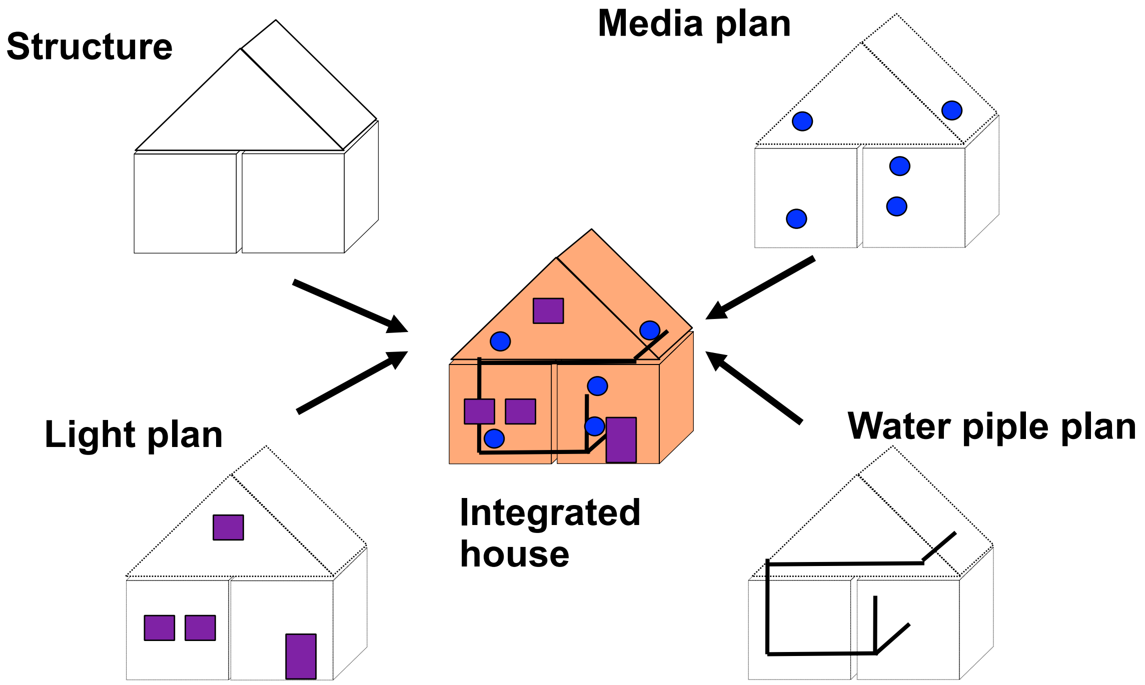
## *Grey-Box Component Models:*
## *The Development of the Last Years*

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

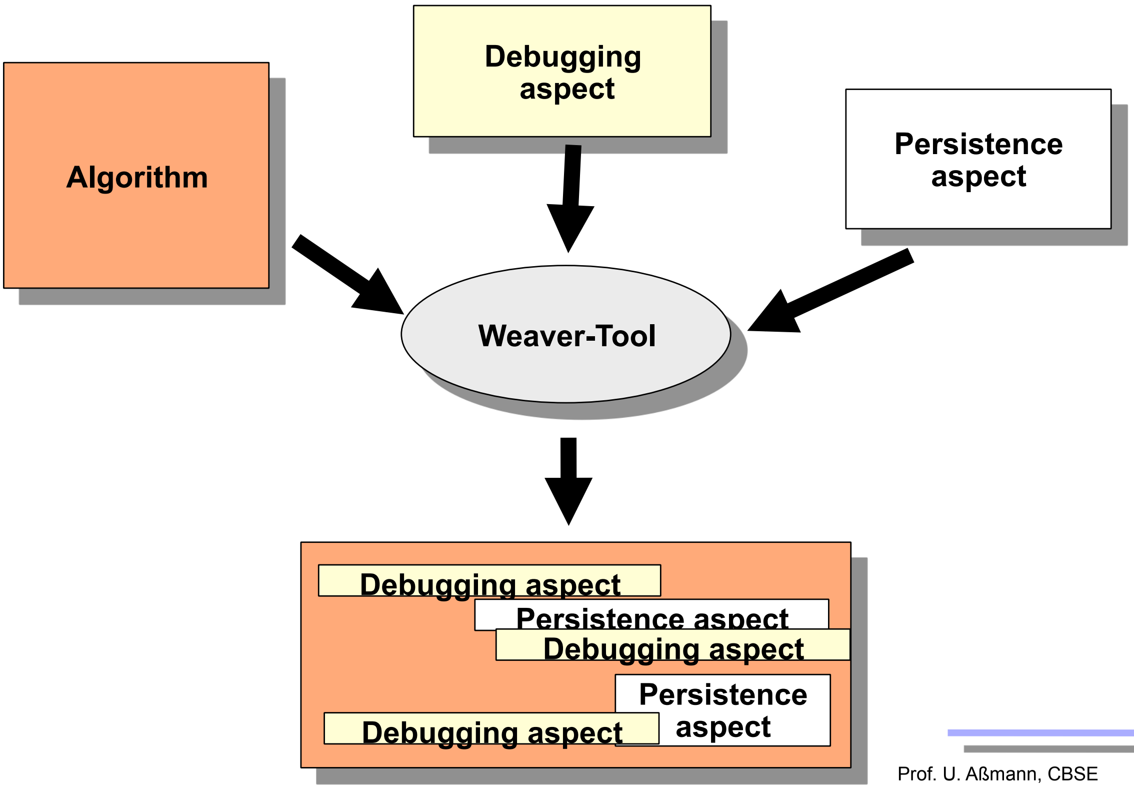
Gray-box composition merges design-time components to run-time components

Black-box composition leaves design-time components untouched (1:1 relationship)

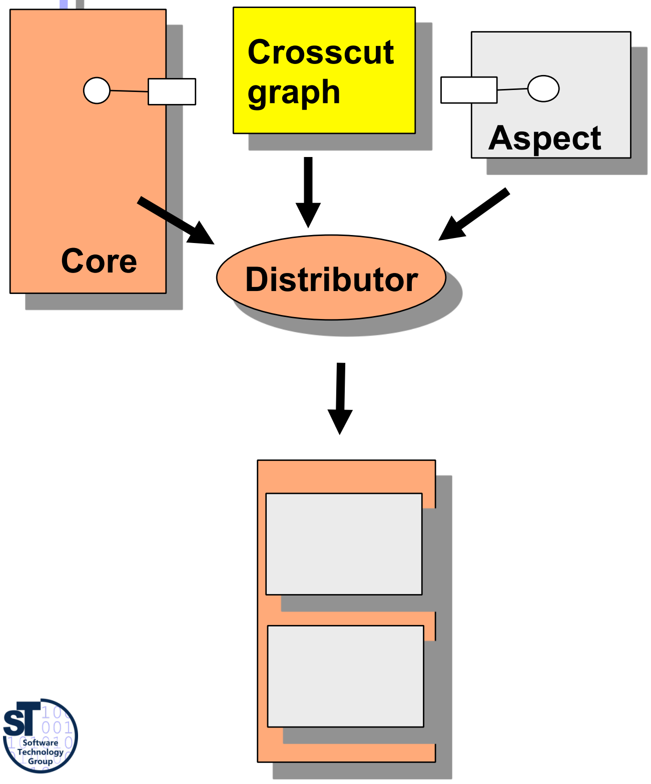
## Aspects in Architecture



## Aspects in Software

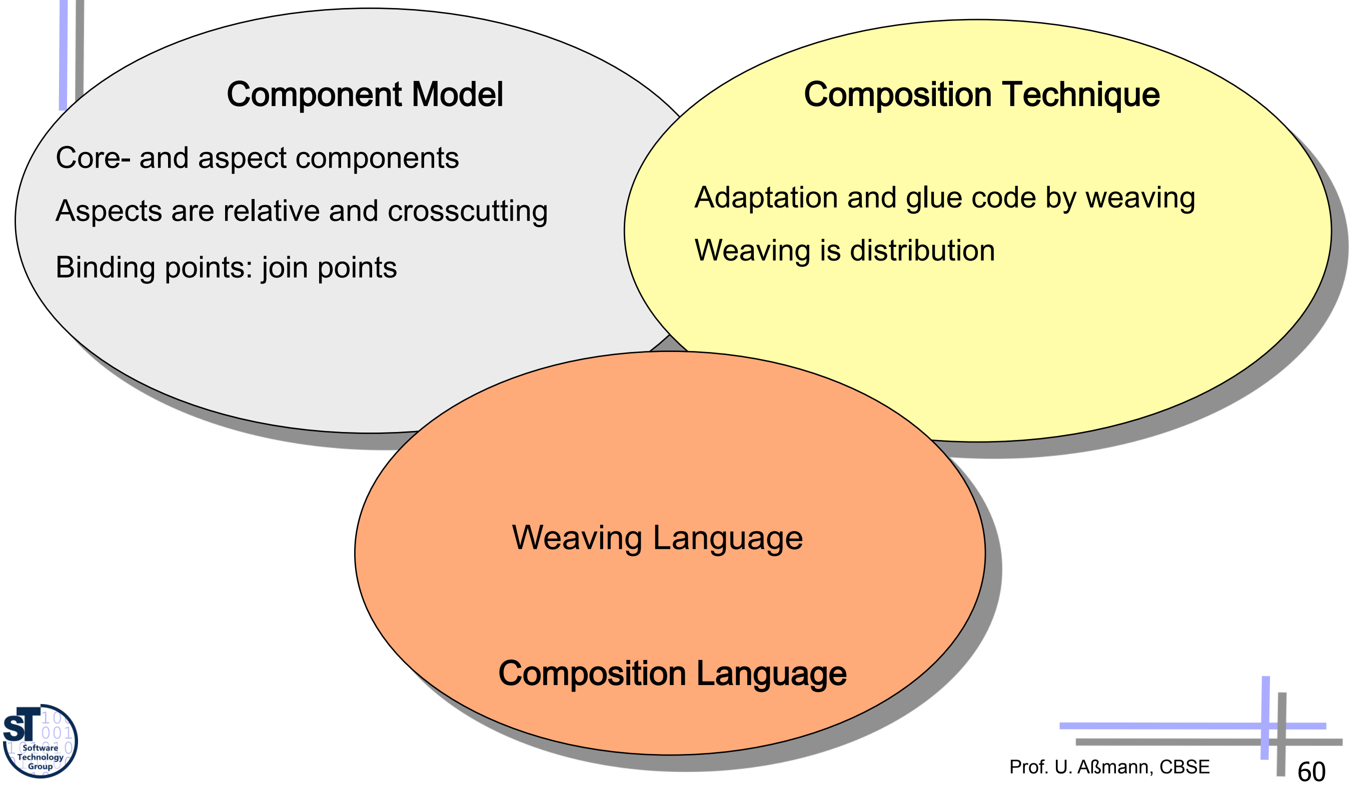


## Aspect Weavers Distribute Advice Components over Core Components



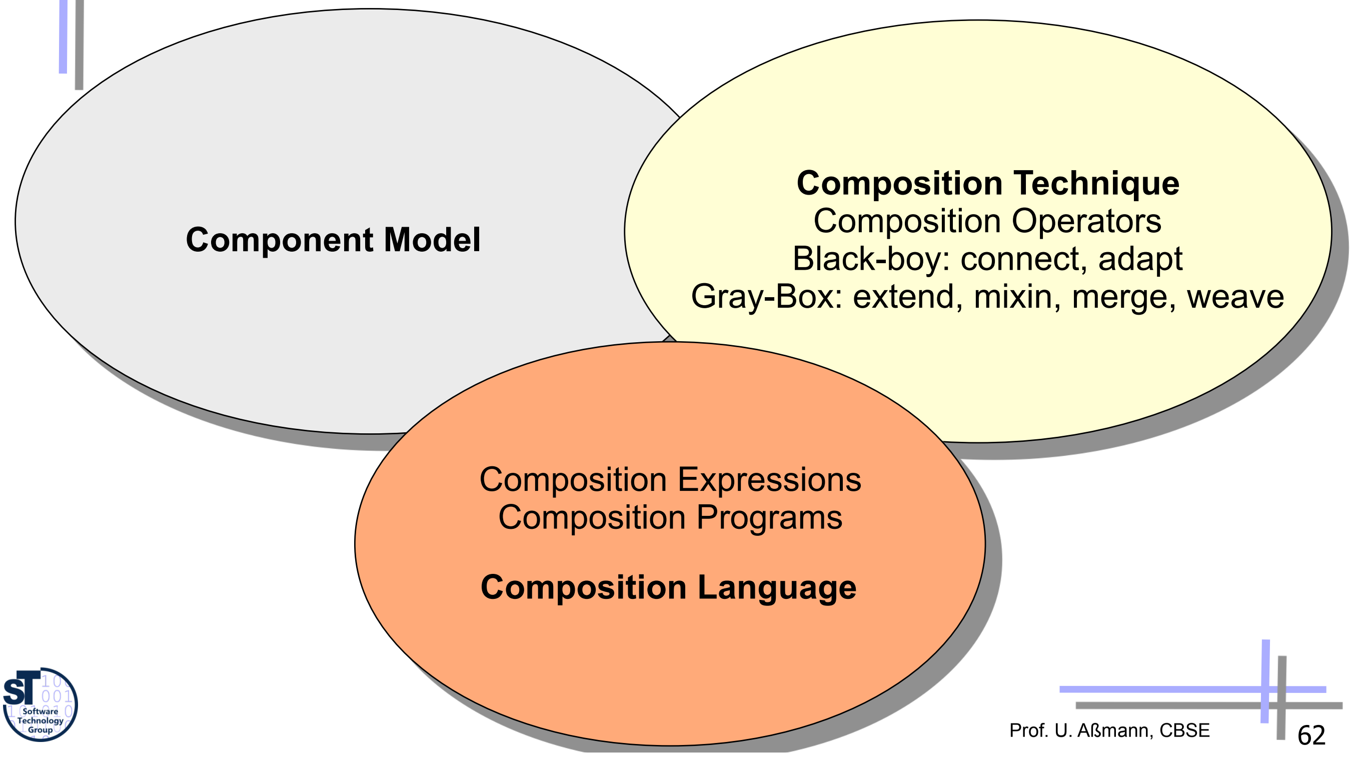
- Aspects are *crosscutting*
- Hence, aspect functionality must be *distributed* over the core
- The distribution is controlled by a crosscut graph

## 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

# 1.3.1 Full-Fledged Composition Systems

## Composition Systems

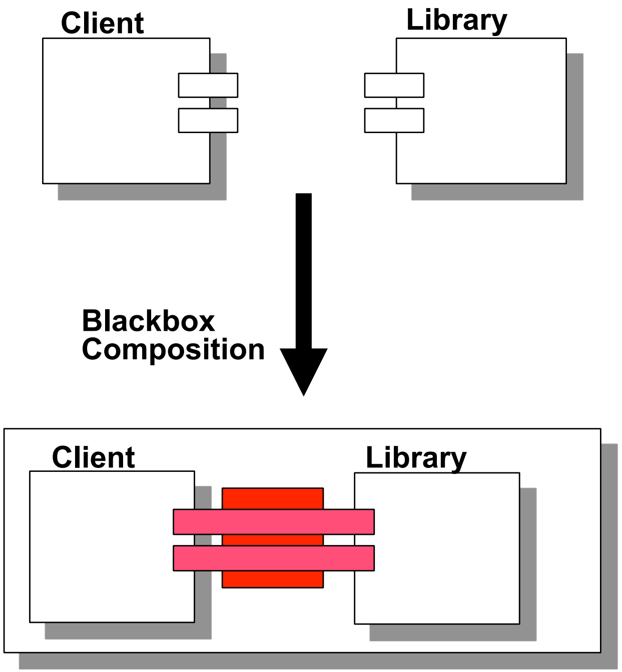


# Composition Systems

- 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]
  - Piccola [Nierstrasz et al., Berne]
  - Invasive software composition (ISC) [Aßmann]
  - Formal calculi
    - Lambda-N calculus [Dami]
    - Lambda-F calculus [Lumpe]

# Connectors are Composition Operators

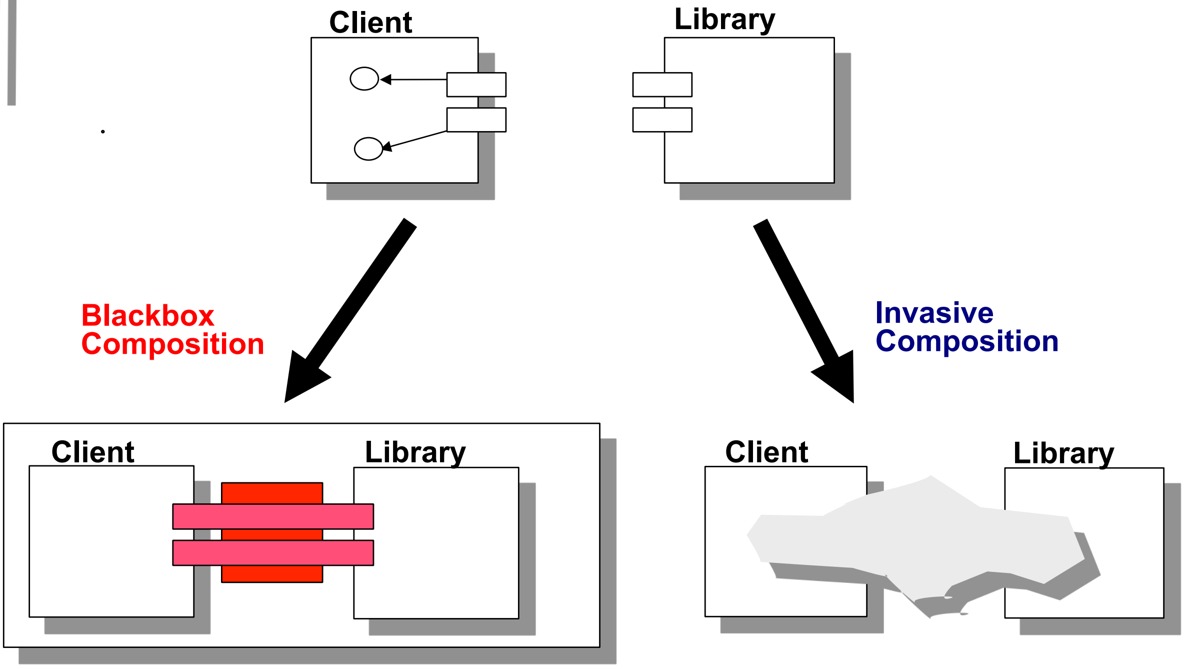
Usually, connectors connect (glue) black-box components for communication



Blackbox connection with glue code

# Connectors can be Grey-Box Composition Operators

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

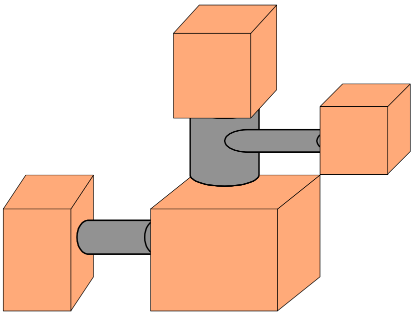


Blackbox connection with glue code

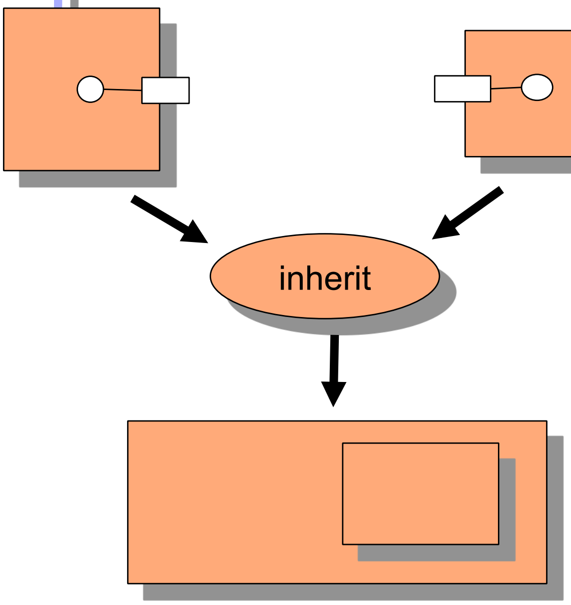
Grey-box (Invasive) Connection

# Composers Generalize Connectors (ADL Component Model)

| Components | Composers | Variation points |
|---|---|---|
| *Black-Box Components* | *Connectors,*<br>*Invasive connectors*<br>*Encapsulation operators* | *Ports* |



# Composers Can Be Used For Inheritance



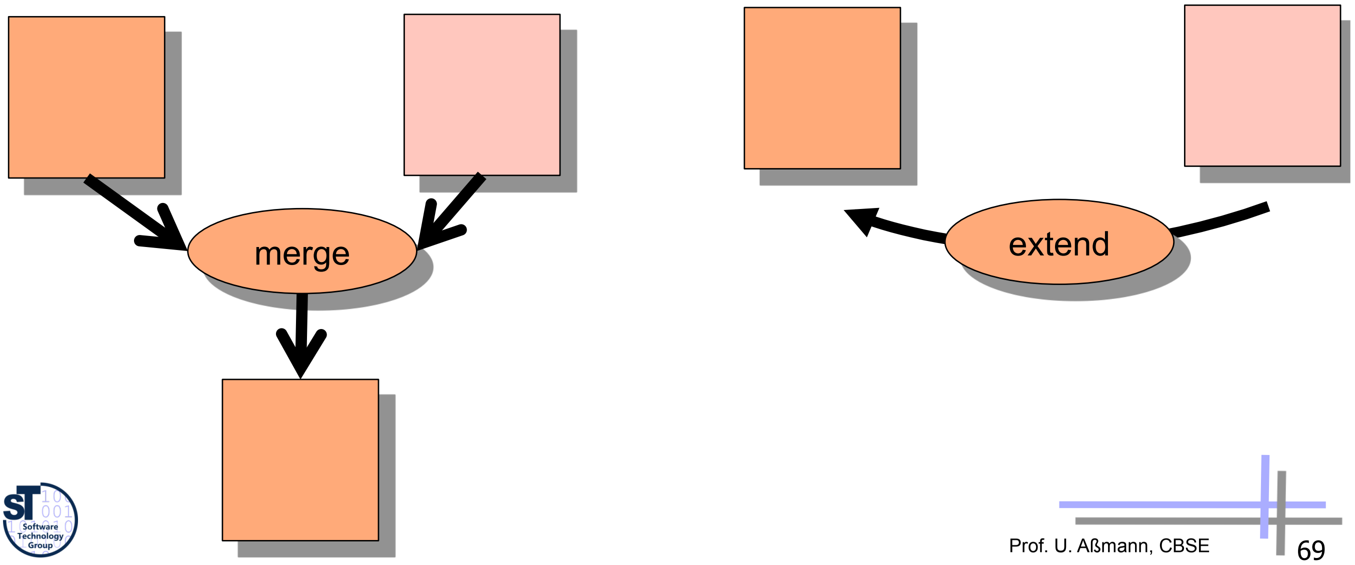
- Extension can be used for inheritance (mixins)
- inheritance :=
  - copy first super document;
  - extend with second super document;
- Be aware: The composition system of object-oriented frameworks (course DPF) is only one of the possible ones

# Composers Generalize Inheritance Operators (Classes as Components)

| Components | Composers | Extension points |
|---|---|---|
| *Classes* | *Mixin operators, inheritance operators* | *Class member lists* |

# Composers Generalize View-based Extensions

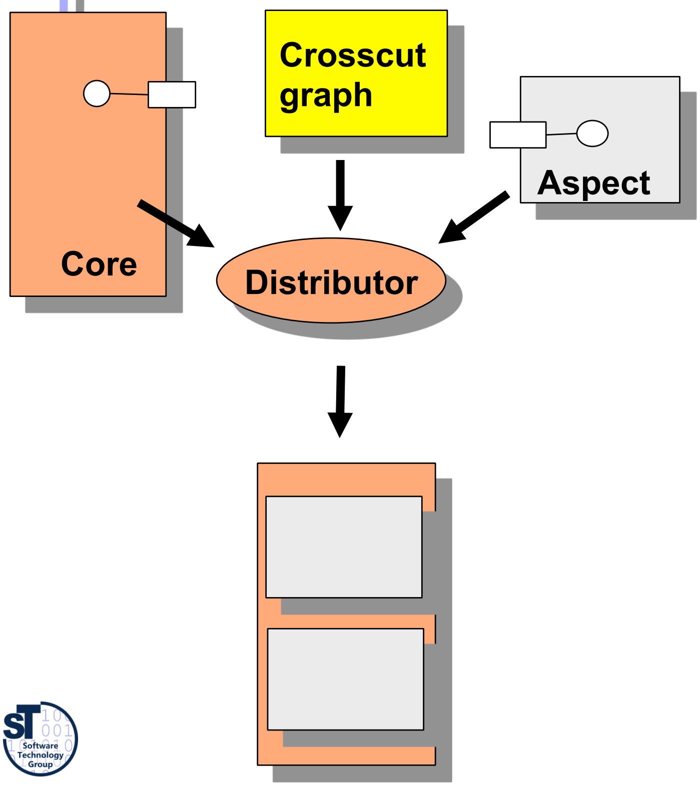
- **Symmetric view**: Two components are *merged*
- **Asymmetric view**: A *core component* is extended by a *view component*



# Composers Generalize View Extensions

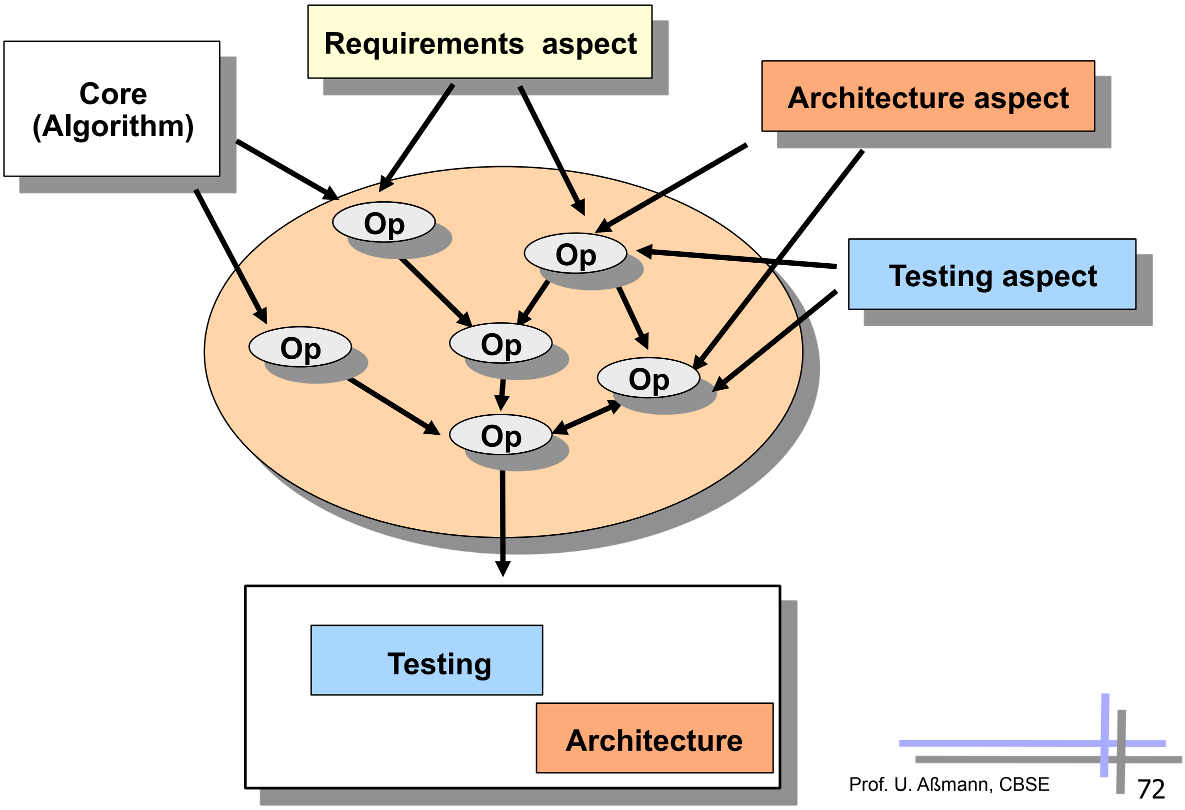
| **Components** | **Composers** | **Extension points** |
|---|---|---|
| *Views* | *Merge operators, extend operators* | *Open definitions* |

# Composers Generalize Aspect Weavers



- Complex composers *distribute* aspect fragments over core fragments
- *Distributors* extend the core
  - Distributors are more complex operators, defined from basic ones
  - Distribution is steered by a *crosscut graph*

# Weavers Are Complex Distributors



# Composers Generalize Aspect Weavers

| Components | Composers | Extension points |
|---|---|---|
| *Core, advice groups* | *Weaver* | *Join points* |

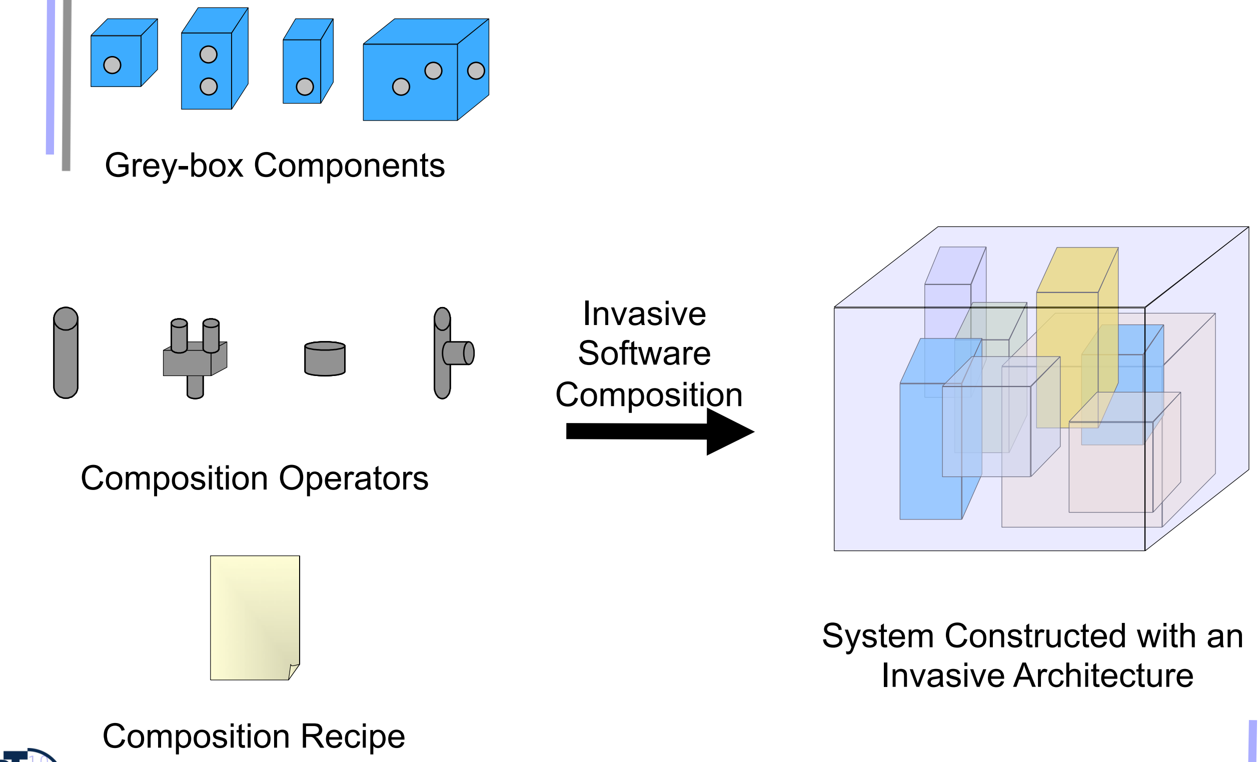
# Comparison Table

| Approach | Components | Composers | Variation/ Extension points |
|---|---|---|---|
| Modular systems | Modules | Static linking<br>Dynamic linking | Linker symbols |
| Object-oriented systems | Classes | Mixin inheritance operator,<br>mixin layer operator,<br>other inheritance operators | Class member lists |
| | Objects | Polymorphic dispatch<br>Dynamic invocation<br>Trading | |
| Architecture systems | Black-Box Components | Connectors,<br>Invasive connectors<br>Encapsulation operators | Ports |
| Generic systems | Generic Fragments | Binding | Slots |
| View systems | Views (fragments) | Merge operators, extend operators | Open definitions |
| Aspect systems | Core, advice groups | Weaver | Join points |
| Full composition systems | All of the above | Explicit crosscut specifications | Slots and join points |

# 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

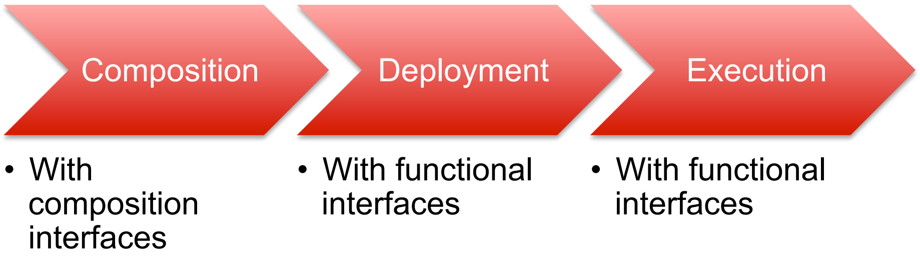
| Composition program size | 1 |
|---|---|
| System size | 10 |


Grey-box Components
Composition Operators
Composition Recipe
Invasive Software Composition
System Constructed with an Invasive Architecture

# Conclusions for Composition Systems

- Components have *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

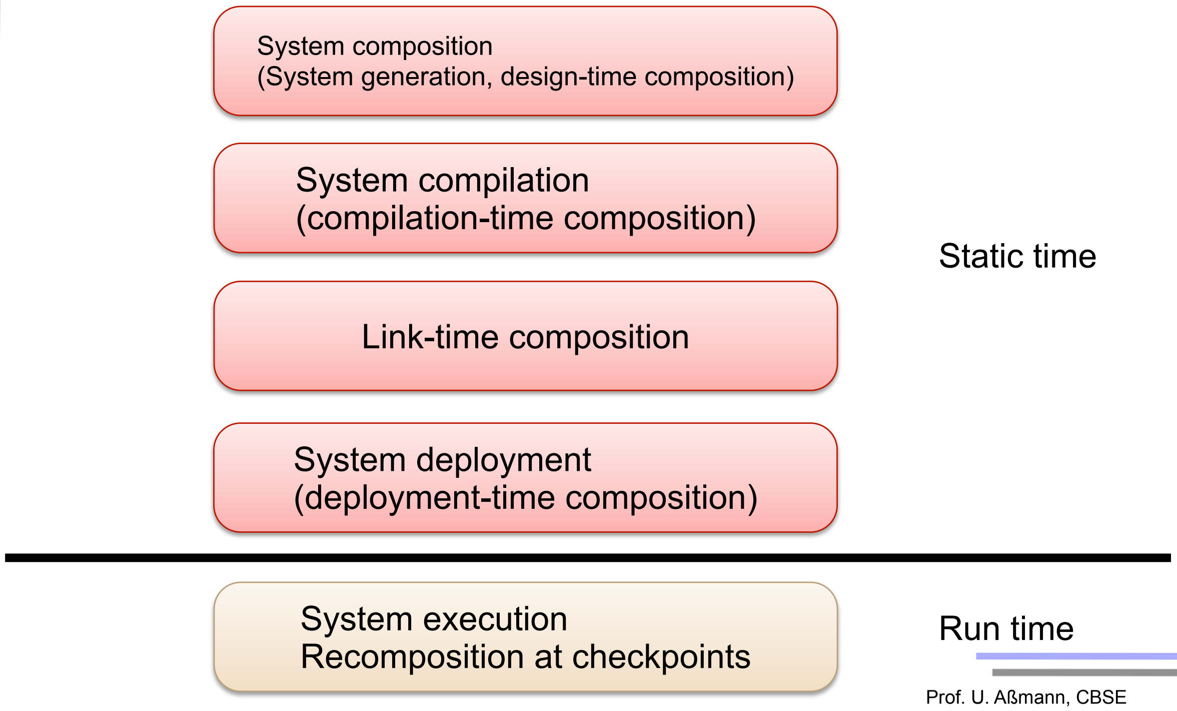
- System composition becomes a new step in system build


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

# 1.4 UBIQUITUOUS COMPONENT MODELS

## Steps in System Construction

- We need *different* component models and composition systems on all levels of system construction



System composition
(System generation, design-time composition)

System compilation
(compilation-time composition)

Link-time composition

System deployment
(deployment-time composition)

System execution
Recomposition at checkpoints

Static time

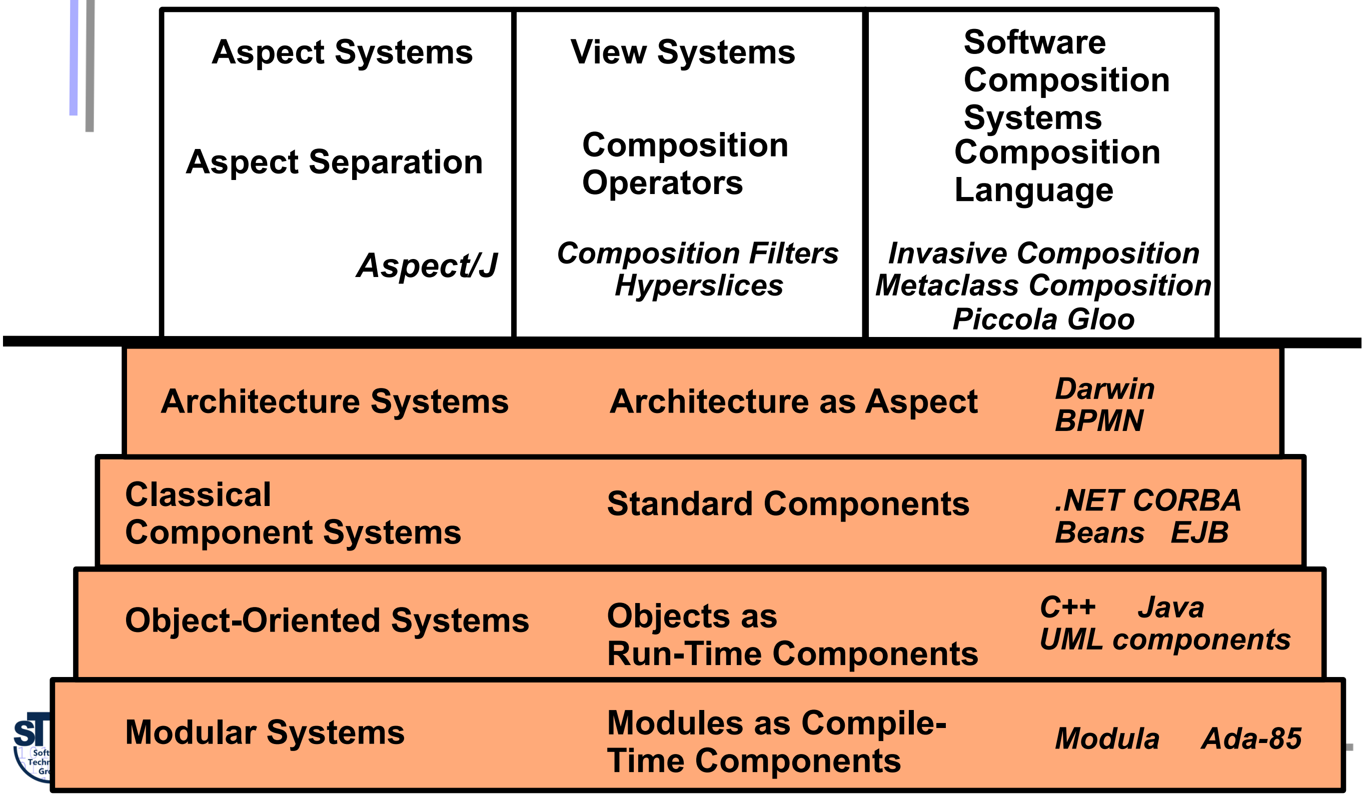
Run time

# 1.5 What Have We Learned?

# Component-based Systems

- ... are produced by component systems or composition systems
- ... have a central relationship that is tree-like or reducible
- ... support a component model
- ... allow for component composition with composition operators
  - ... and – in the large – with composition languages
- Historically, component models and composition techniques have been pretty different
  - from compile time to run time
- Blackbox composition supports variability and glueing
- Graybox composition supports extensibility, views, aspects
- Object-orientation is just one of the many composition systems which have been defined

# The Ladder of 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 Gloo*

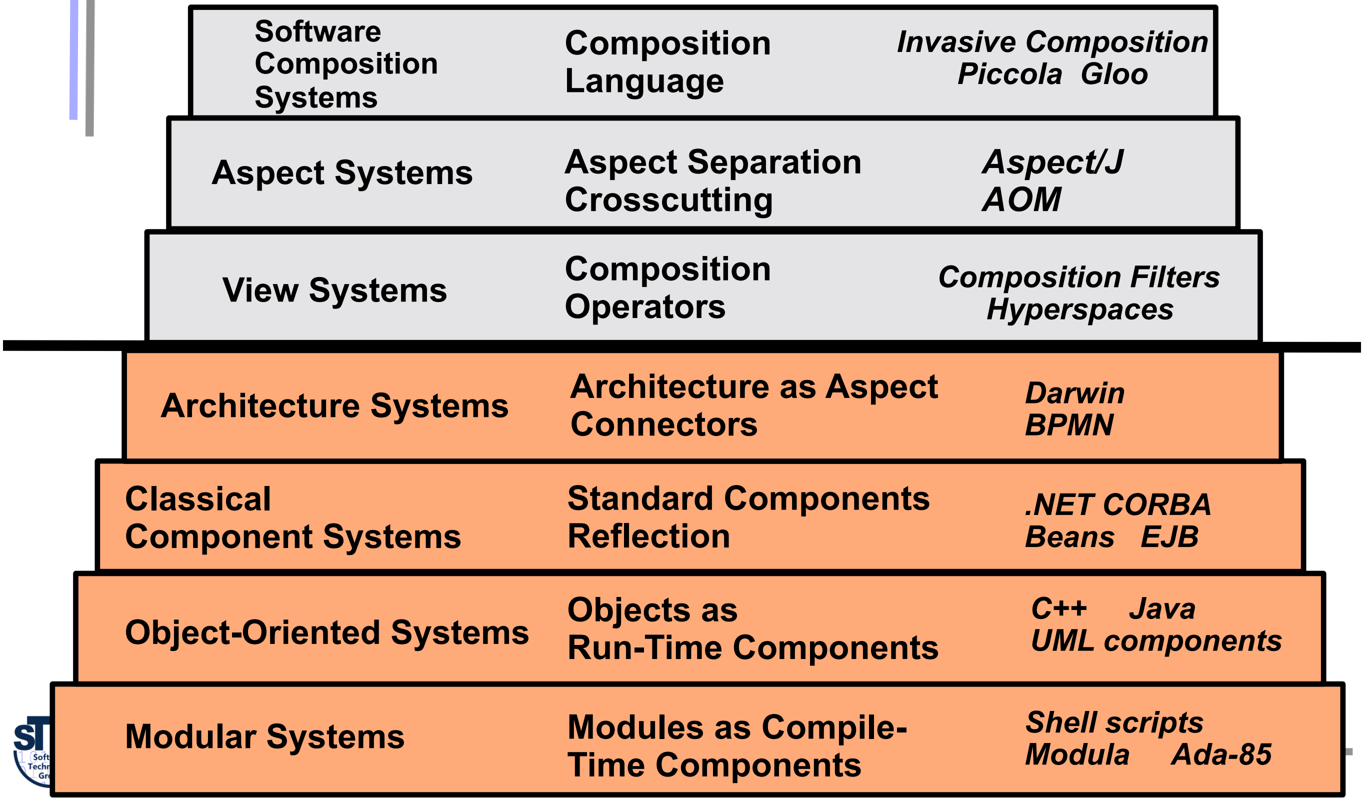
Architecture Systems — Architecture as Aspect — *Darwin BPMN*

Classical Component Systems — Standard Components — *.NET CORBA Beans EJB*

Object-Oriented Systems — Objects as Run-Time Components — *C++ Java UML components*

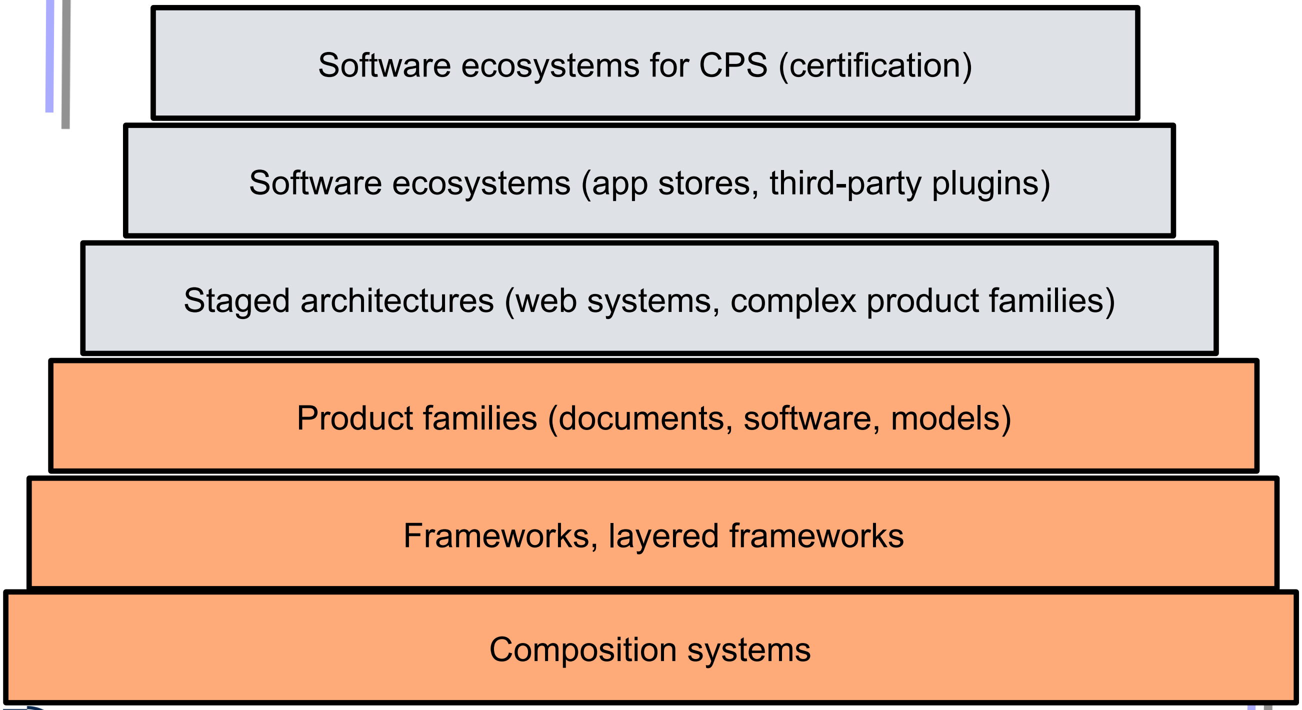
Modular Systems — Modules as Compile-Time Components — *Modula Ada-85*

# The Ladder of Composition Systems



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

# What Can Be Done with Composition Systems?



Software ecosystems for CPS (certification)

Software ecosystems (app stores, third-party plugins)

Staged architectures (web systems, complex product families)

Product families (documents, software, models)

Frameworks, layered frameworks

Composition systems

# The End