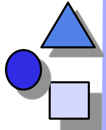
Component-Based Software Engineering Introduction

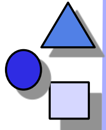
Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de

10-0.1, Apr 12, 2010



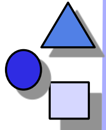
Goals

- Understand what a component-based system is
- Understand the difference of component-based and composition-based systems
- Understand 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



Contents

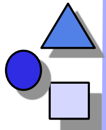
- A little history of software composition
  - Comparison criteria for composition
- How it is realized for Invasive Software Composition
- Future software composition systems



Obligatory Reading

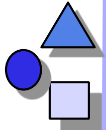
- [ISC], Chapter 1, Chapter 2
- Douglas McIlroy's home page
  http://cm.bell-labs.com/who/doug/
- [McIlroy] Douglas McIlroy. Mass Produced Software Components. In P. Naur and B. Randell, "Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th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

A Little History



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

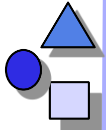
In the phrase `mass production techniques,' my emphasis is on `techniques' and not on mass production plain. Of course mass production, in the sense of limitless replication of a prototype, is trivial for software.

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.

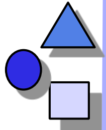
Yet this fragile analogy is belied when we seek for analogues of other tangible symbols of mass production.

- 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?



Definitions of 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.

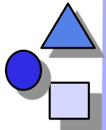
(ECOOP Workshop WCOP 1997 Szyperski)

A reusable software component is a

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

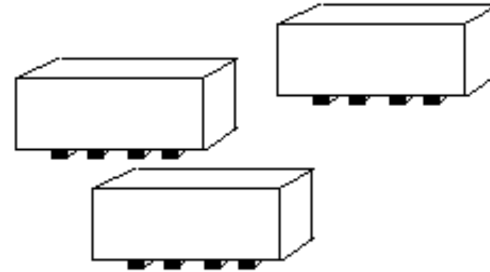
A software component is a static abstraction with plugs.

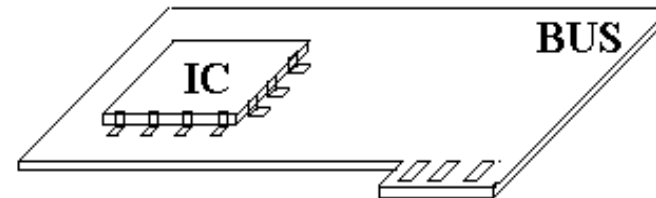
(Nierstrasz/Dami)

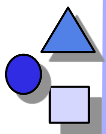


Real Component Systems

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



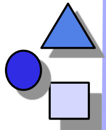




What is a Software Component?

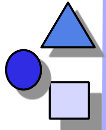
- A component is a *container with*
  - *variation points*
  - *extension points*
  - that are adapted during composition
- A component is a reusable *unit for composition*

- 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?

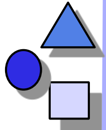
- A component-based system has the following divide-and-conquer feature:
  - A component-based system is a system in which a major relationship between the components is **tree-shaped or reducible**.
- Consequence: the entire system can be reduced to one abstract node
  - at least along the structuring relationship
- Systems with layered relations (dag-like relations) are not necessarily component-based.
  - Because they cannot be reduced



What Is A Component-Based System?

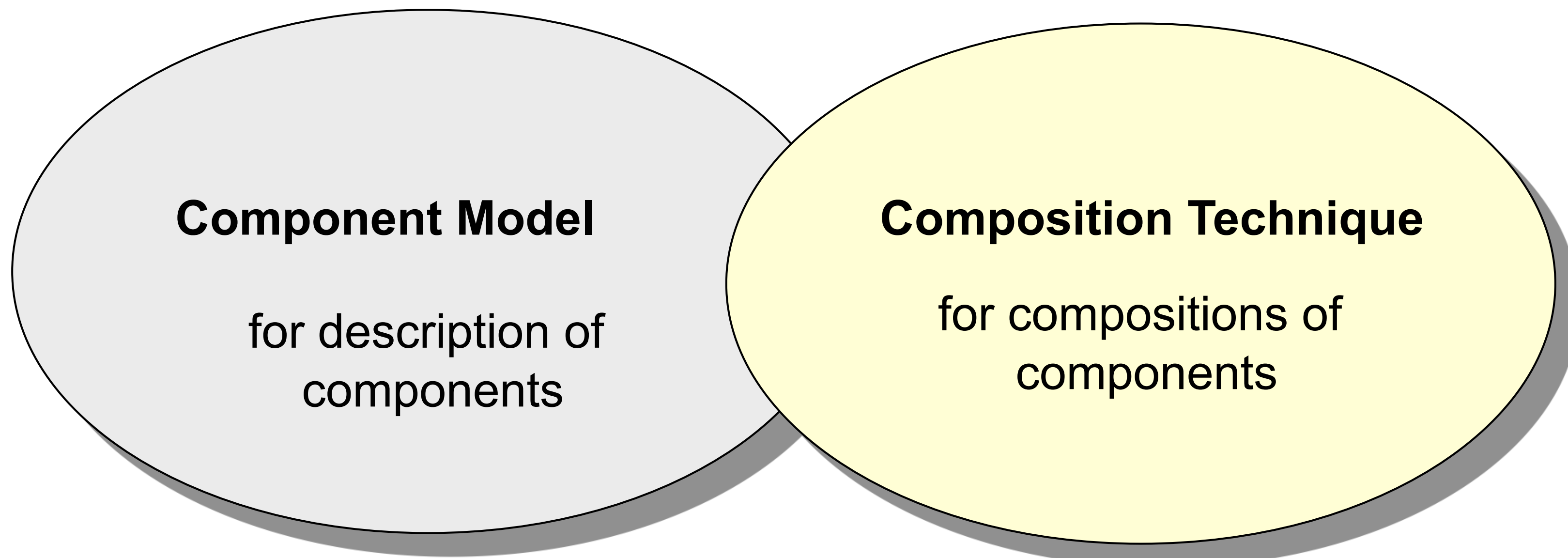
- Because of the divide-and-conquer property, component-based development is attractive.
- However, we have to choose the structuring relation
- And, we have to choose the composition model

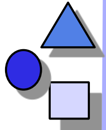
- Mainly, 2 types of models are known
  - Modular decomposition (blackbox)
  - Separation of concerns (graybox)



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





Composition Systems

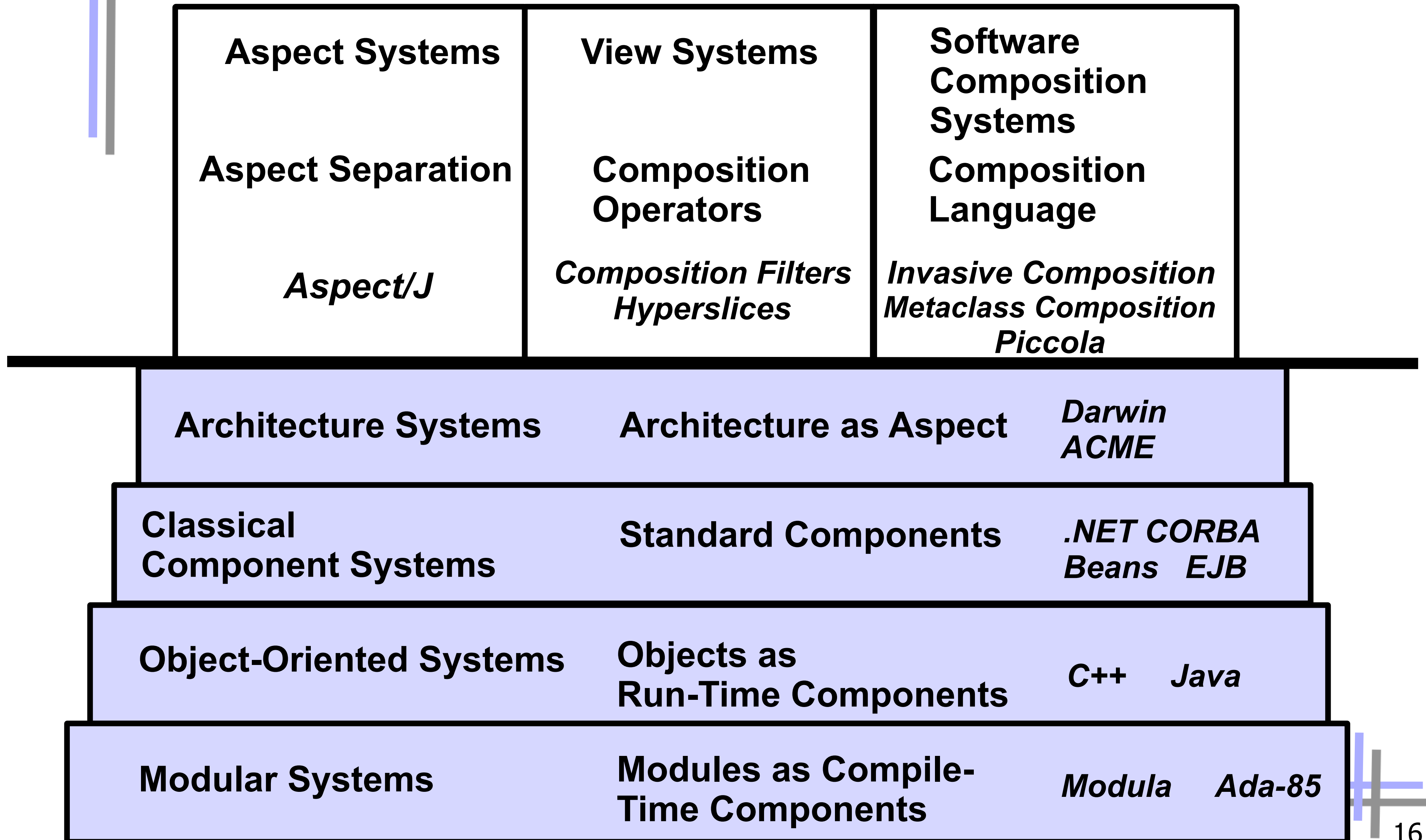
- A composition system has

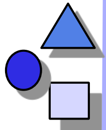
Component Model

Composition Technique

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

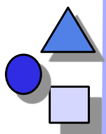
The Ladder of Component and Composition Systems





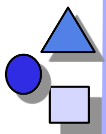
Desiderata for Flexible Software Composition

- 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!



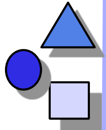
Desiderata Component Model

- **CM-M: Modularity**
  - M1 Component secrets (information hiding):
    - Location, way of deployment
    - Component lifetime
    - Component language
    - Explicit specification of interfaces (contact points, exchange points, binding points)
    - Provided and required interfaces
  - M2 Semantic substitutability (conformance, contracts)
    - Syntactic substitutability (typing)
  - 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



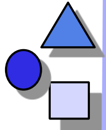
Desiderata Composition Technique

- **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?



Desiderata Composition Language

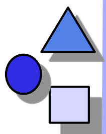
- **CL1: Product Consistency**
  - Variant cleanness: consistent configurations
  - Robustness: absence of run-time exceptions
- **CL2: Software Process Support**
  - Build management automation
- **CL3: Meta-composition**
  - Is the composition language component-based, i.e., can it be composed itself?
  - Reuse of architectures
- **CL4: Architectural styles** (composition styles)
  - Constraints for the composition



Service Components

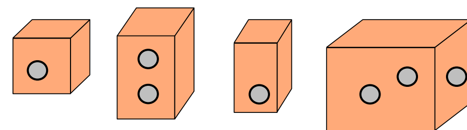
- A *service component* is a software component whose location, style of deployment, and name is not known.
  - It is described by metadata (attributes)
  - [from Greenfield/Short, Software Factories, AWL]

Historical Approaches to Components

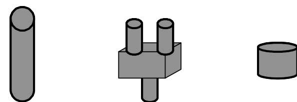


The Essence of the 60s-90s: LEGO Software

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

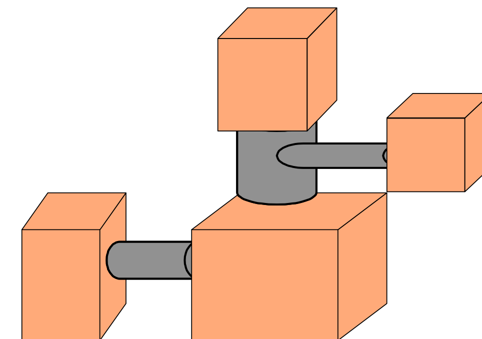


Components

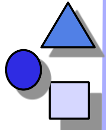


Connectors

Composition recipe

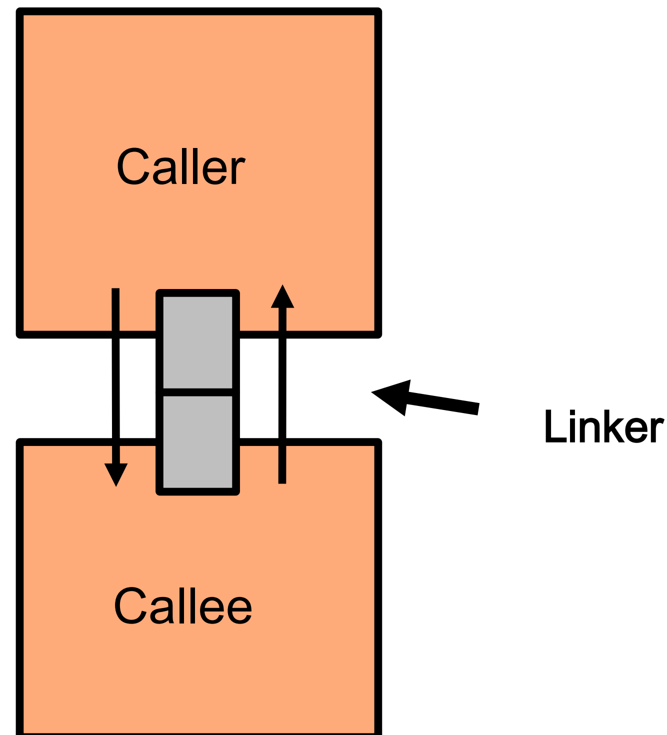


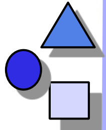
Component-based applications



Procedure Systems

- Fortran, Algol
- 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

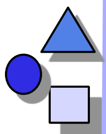
Composition Technique

Connection by linking object files

Program transformation on object files

Composition time: link-time, static

Composition Language

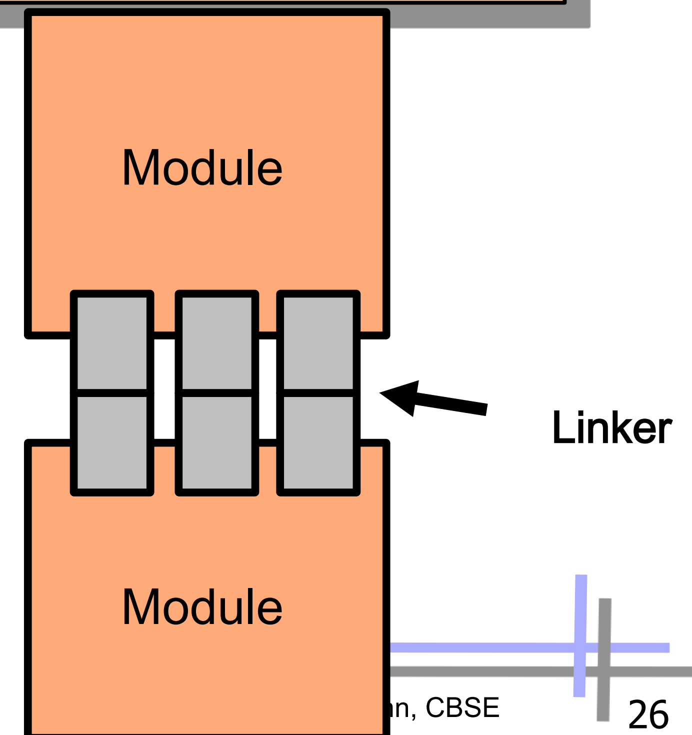


Modules (Information-Hiding-Based Design a la Parnas)

- 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.

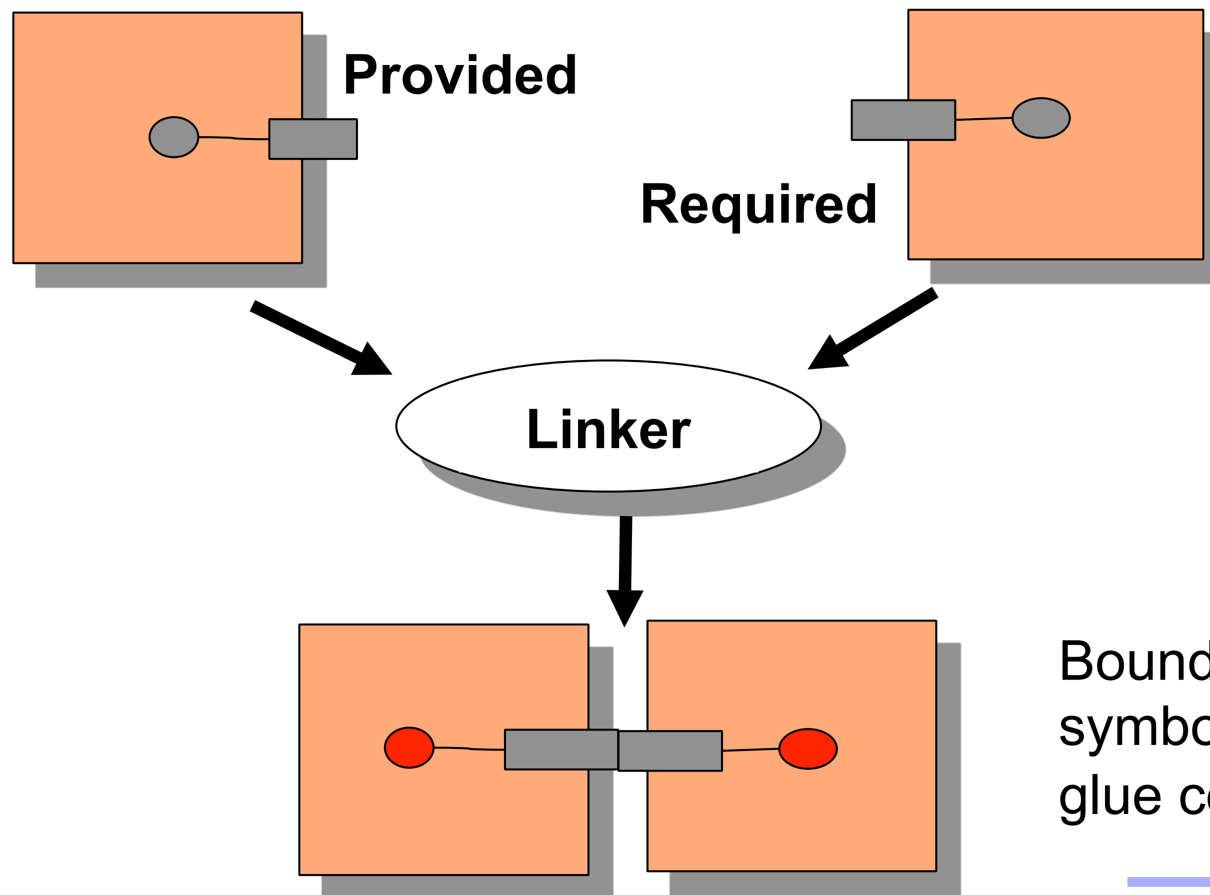
Such modules have rather abstract interfaces which are relatively unlikely to change.



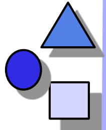
- Static binding of functional interfaces to each other
- Concept has penetrated almost all programming languages (Modula, Ada, Java, C++, Standard ML, C#)

A Linker is a 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

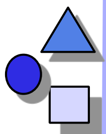
Composition Technique

Connection by linking object files

Program transformation on object files

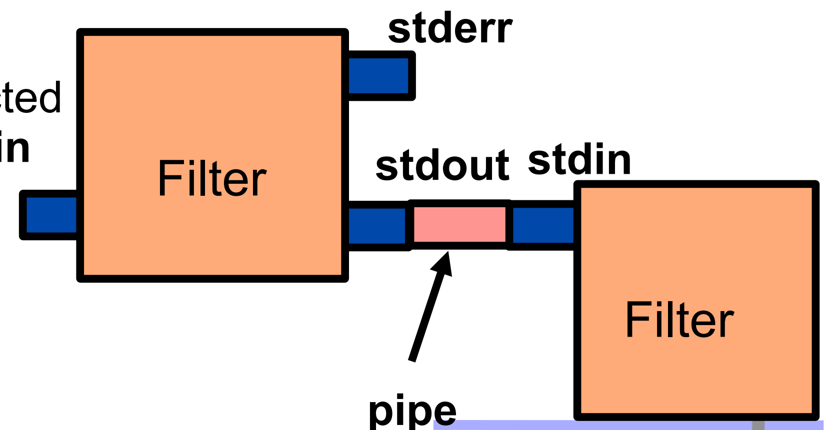
Composition time: link-time, static

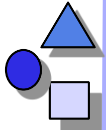
Composition Language



UNIX Shells and Pipes (McIlroy)

- UNIX shells still offers the most used component paradigm
  - 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

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

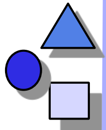
Caller Object

dispatch

Callee

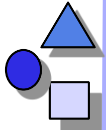
Callee

Callee



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

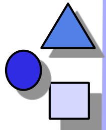
Binding points: static and polymorphic calls (dynamically dispatched calls)

Composition Technique

Adaptation by inheritance or delegation

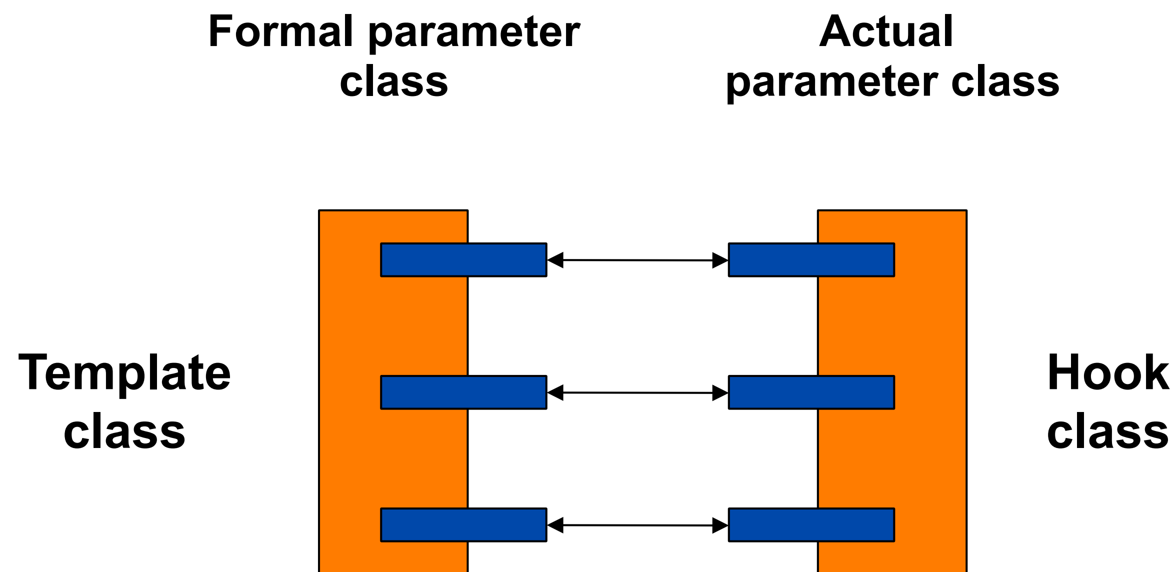
Extensibility by subclassing

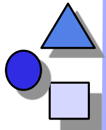
Composition Language



O-O Frameworks

- [Pree] A framework consists of a set of template classes which can be parameterized by *hook classes (parameter classes)*



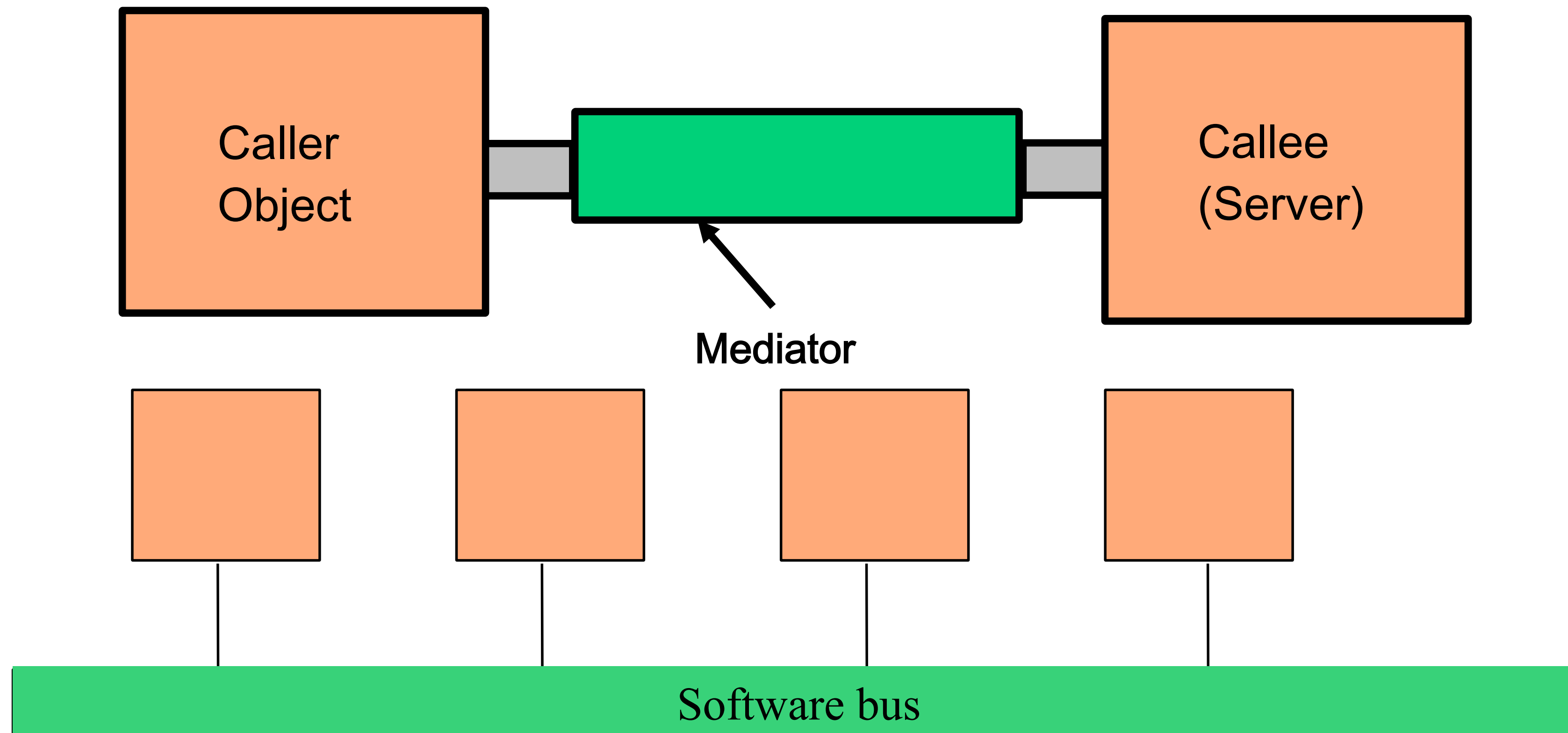


O-O Frameworks

- 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

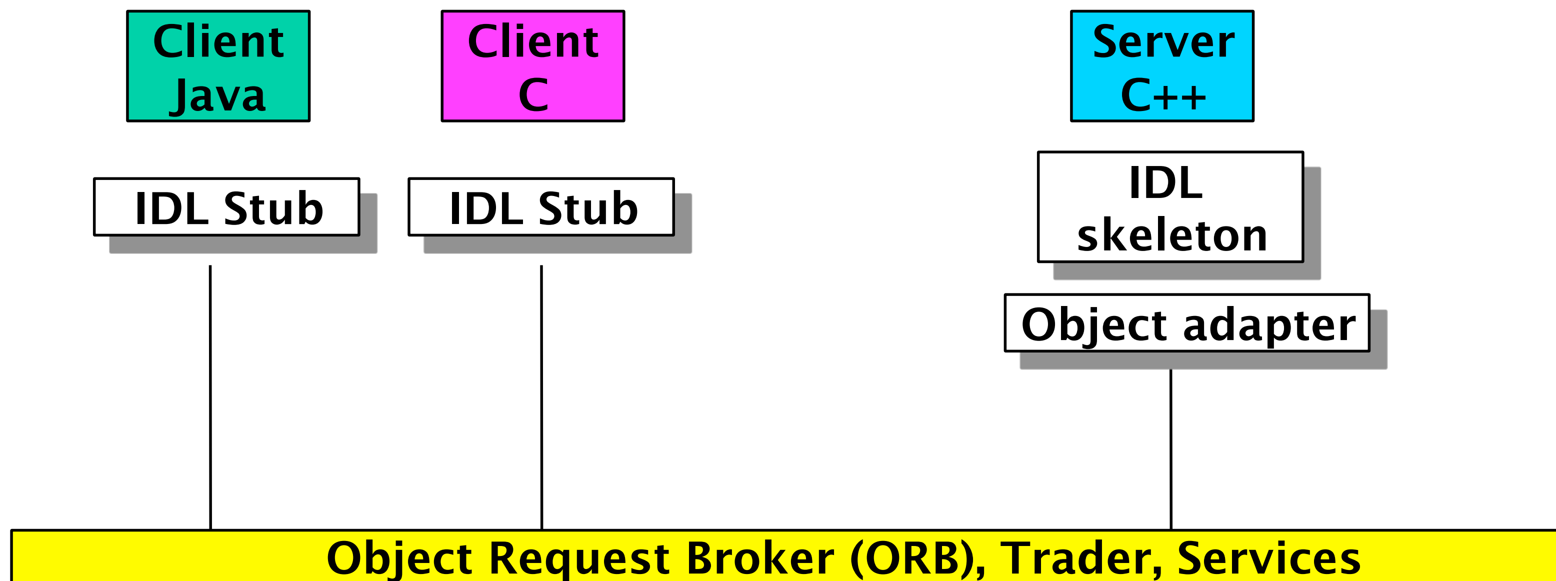
Commercial Component Systems (COTS, Components off the Shelf)

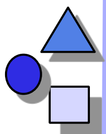
- CORBA/DCOM/.NET/JavaBeans/EJB
- Although different on the first sight, turn out to be rather similar



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

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

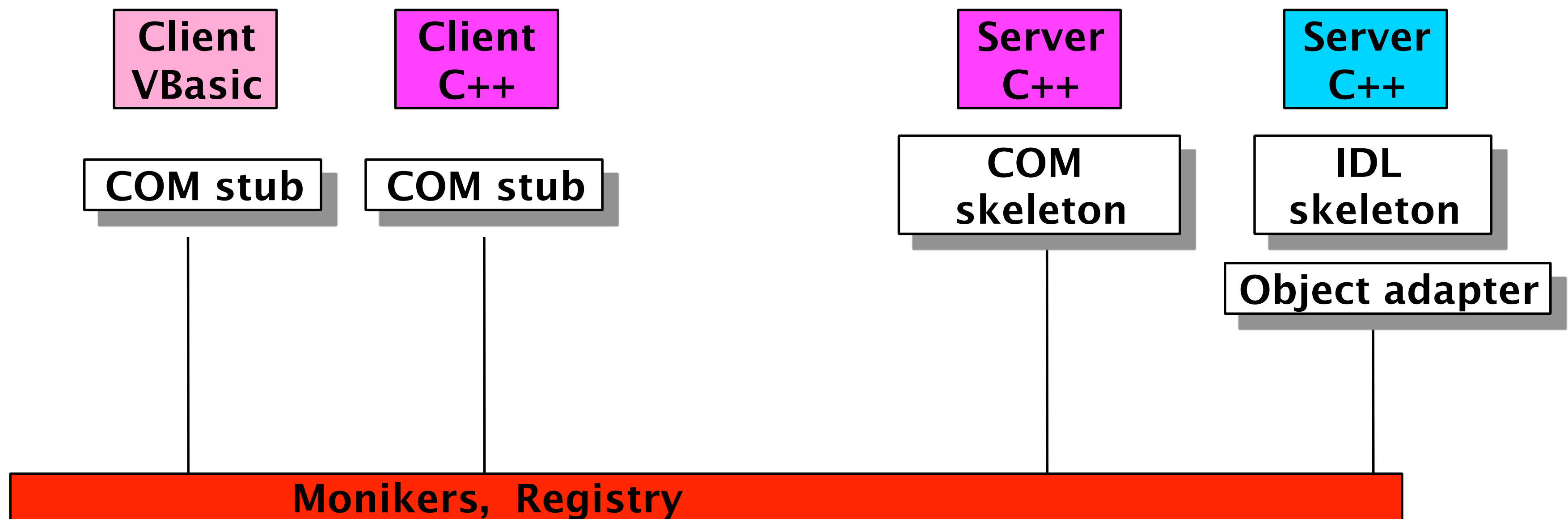


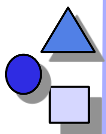


(D)COM(+), ActiveX

http://www.activex.org

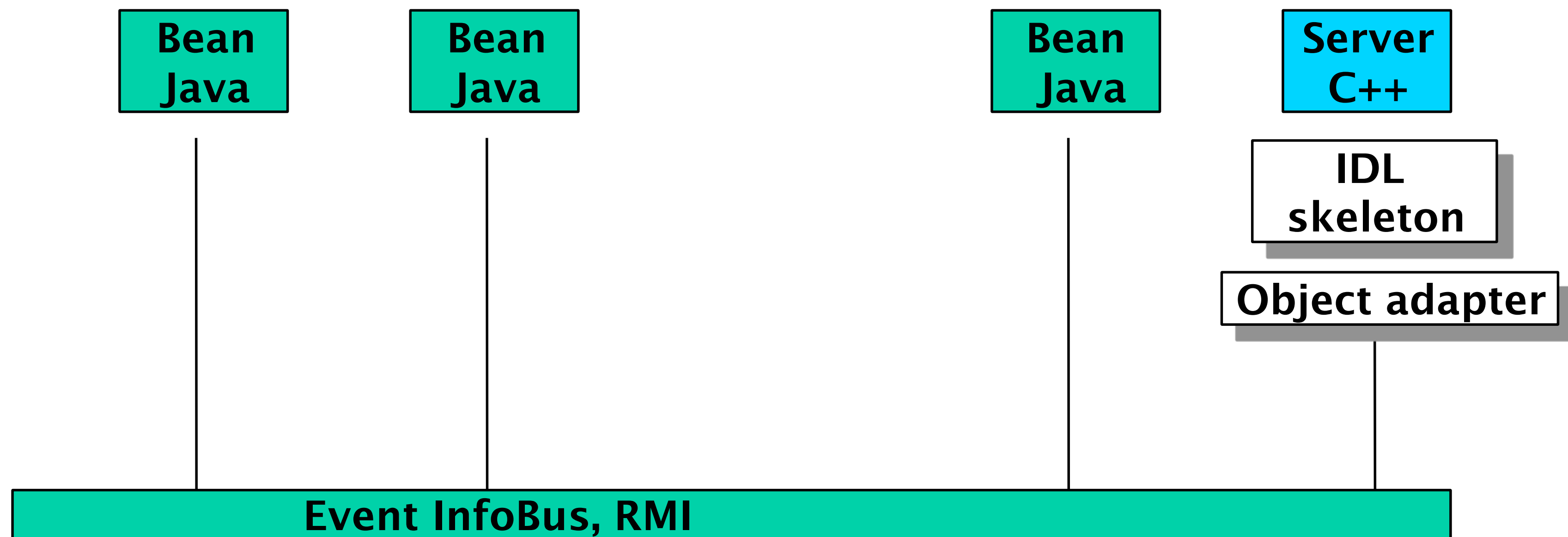
- Microsoft's model is similar to CORBA. Proprietary
- DCOM is a binary standard

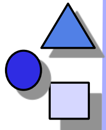




Java Beans
http://www.javasoft.com

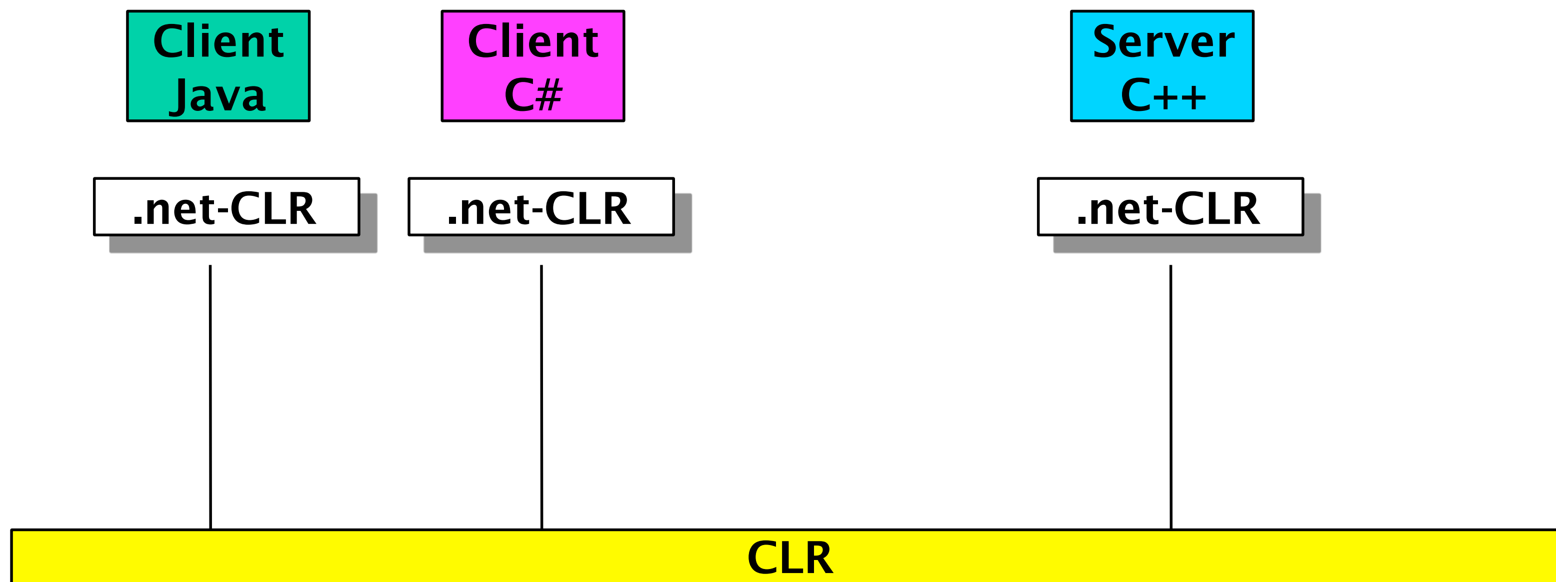
- Java only, event-based, transparent distribution by remote method invocation (RMI)
- source code/bytecode-based

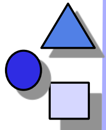




.NET
http://www.microsoft.com

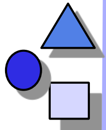
- Language independent, distribution transparent
- NO interface definition language IDL (at least for C#)
- source code or bytecode MSIL
- Common Language Runtime CLR





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

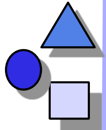
Composition Technique

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

Dynamic call in CORBA

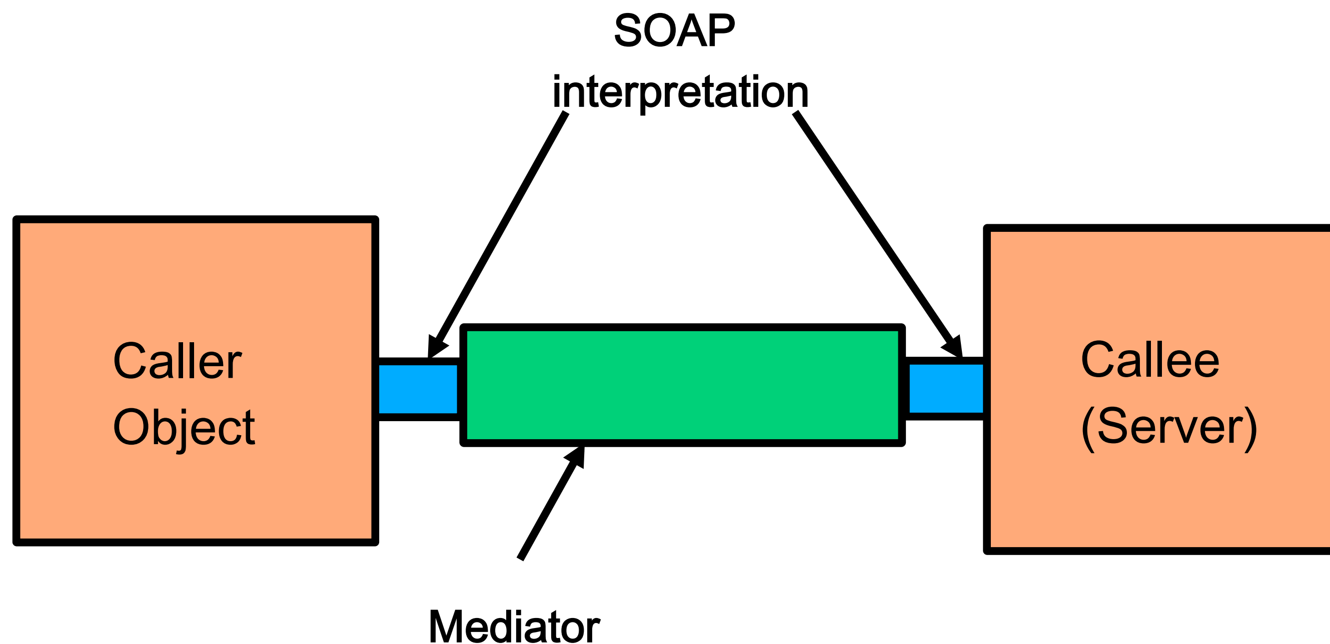
Composition Language

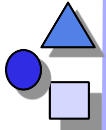
VisualBasic for COM



Web Services

- Binding procedure is interpreted, not compiled
- More flexible:
  - When interface changes, no recompilation and rebinding
  - Ubiquitous protocol HTTP





Web Services as Composition System

Component Model

Content: not important

Binding points are described by XML

Binding procedure is interpretation of SOAP

Secrets: distribution, implementation language

Composition Technique

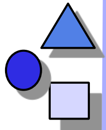
Adaptation for distributed systems (marshalling) and mixed-language systems

Glue: SOAP, HTTP

WSDL, JAX-WS,

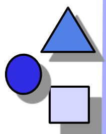
UDDI, BPEL

Composition Language



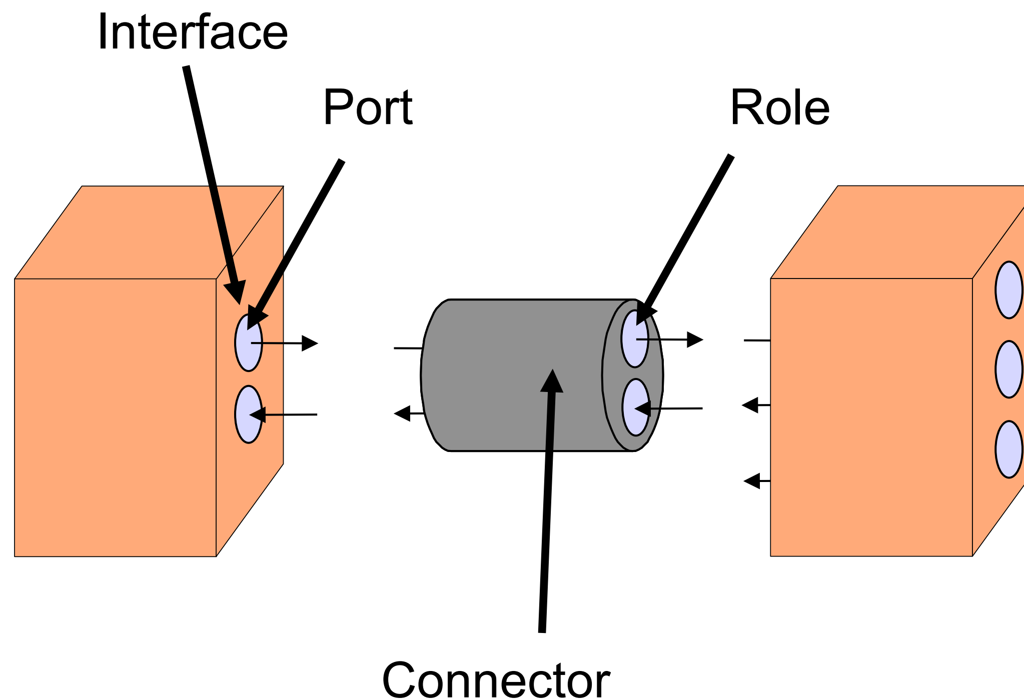
Architecture Systems

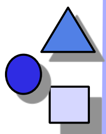
- Unicon, ACME, Darwin
  - feature an Architecture Description Language (ADL)
- Split an application into:
  - Application-specific part (encapsulated in components)
  - Architecture and communication (in architectural description in ADL)
  - Better reuse since both dimensions can be varied independently



Component Model in Architecture Systems

- Ports abstract interface 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

Port 1

Port

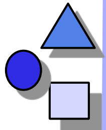
Component

Port 2

Component

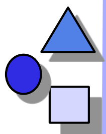
Port

Component



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

AcmeStudio - [simple-demo.acme]

File Edit View Insert Types Tools Window Help

PF

Global Types | PF

BinaryFilte
Filter

Pipe

ReadPort
WritePort

Sink
Source

Look in: Syst

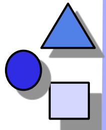
Capitalize
stdin
stdout
Representati
Aggregat
Lowe
Merg
Split

UpperCase
Pipe26
Pipe29
Split
Merge
Pipe27
Pipe28
LowerCase

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

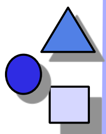
Composition Technique

Adaptation and glue code by connectors

Scaling by exchange of connectors

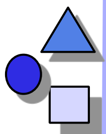
Architectural language

Composition Language

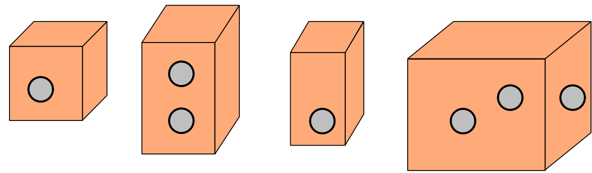


What the Composition Language Offers for the Software Process

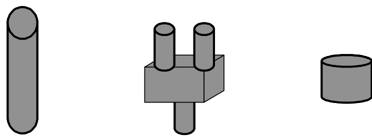
- Communication
  - Client can understand the architecture graphics well
  - Architecture styles classify the nature of a system in simple terms (similar to design patterns)
- Design support
  - Refinement of architectures (stepwise design, design to several levels)
  - Visual and textual views to the 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



Blackbox Composition

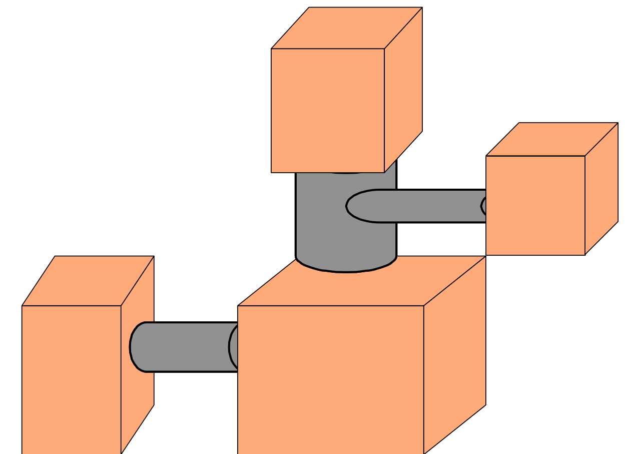


Components

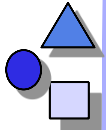


Connectors

Composition recipe

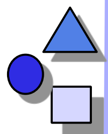


Component-based applications



The Essence of Blackbox Composition

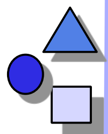
- 3 Problems in System construction
  - Variability
  - Extensibility
  - Adaptation
- In “Design Patterns and Frameworks”, we learned about design patterns to tackle these problems
- Blackbox composition supports variability and adaptation
  - not extensibility



The Ladder of Composition Systems

Aspect Systems	View Systems	Software Composition Systems
Aspect Separation	Composition Operators	Composition Language
Aspect/J	*Composition Filters* *Hyperslices*	*Invasive Composition* *Metaclass Composition* *Piccola*

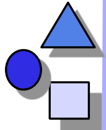
Architecture Systems	Architecture as Aspect	*Darwin* *ACME*
Classical Component Systems	Standard Components	*.NET CORBA* *Beans EJB*
Object-Oriented Systems	Objects as Run-Time Components	*C++ Java*
Modular Systems	Modules as Compile-Time Components	*Modula Ada-85*



The Ladder of Composition Systems (rev.)

Software Composition Systems	Composition Language	*Invasive Composition Piccola*
Aspect Systems	Aspect Separation Crosscutting	*Aspect/J*
View Systems	Merge Operator	*Composition Filters Hyperslices*
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

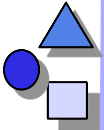


The Essence of the Last Years

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

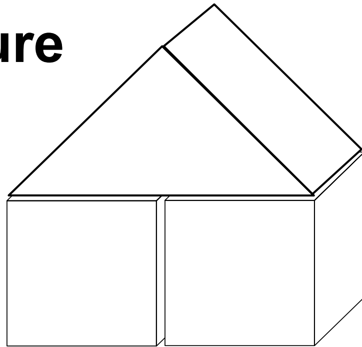
Component Integration

Component Extension

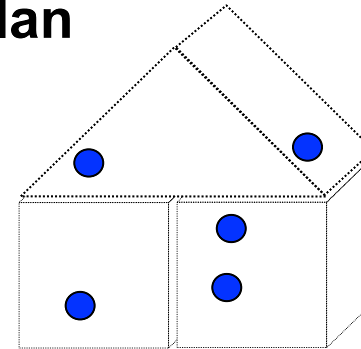


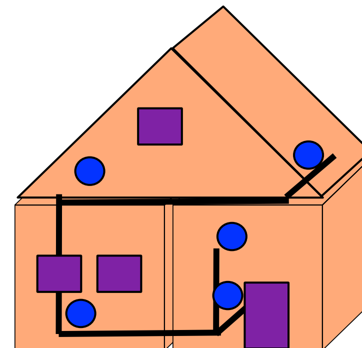
Aspects in Architecture

Structure



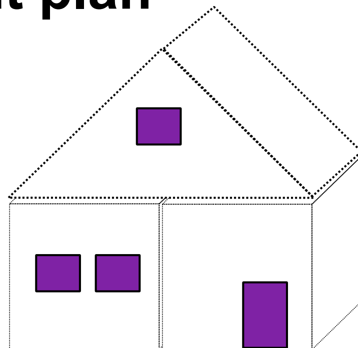
Media plan



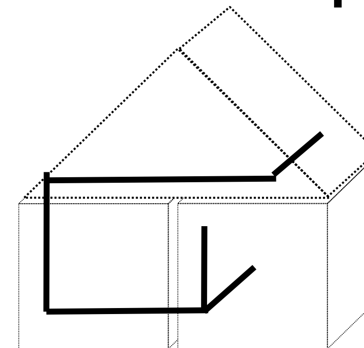


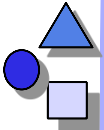
Integrated house

Light plan



Water piple plan





Aspects in Software

Algorithm

Debugging aspect

Persistence aspect

Weaver-Tool

Debugging aspect

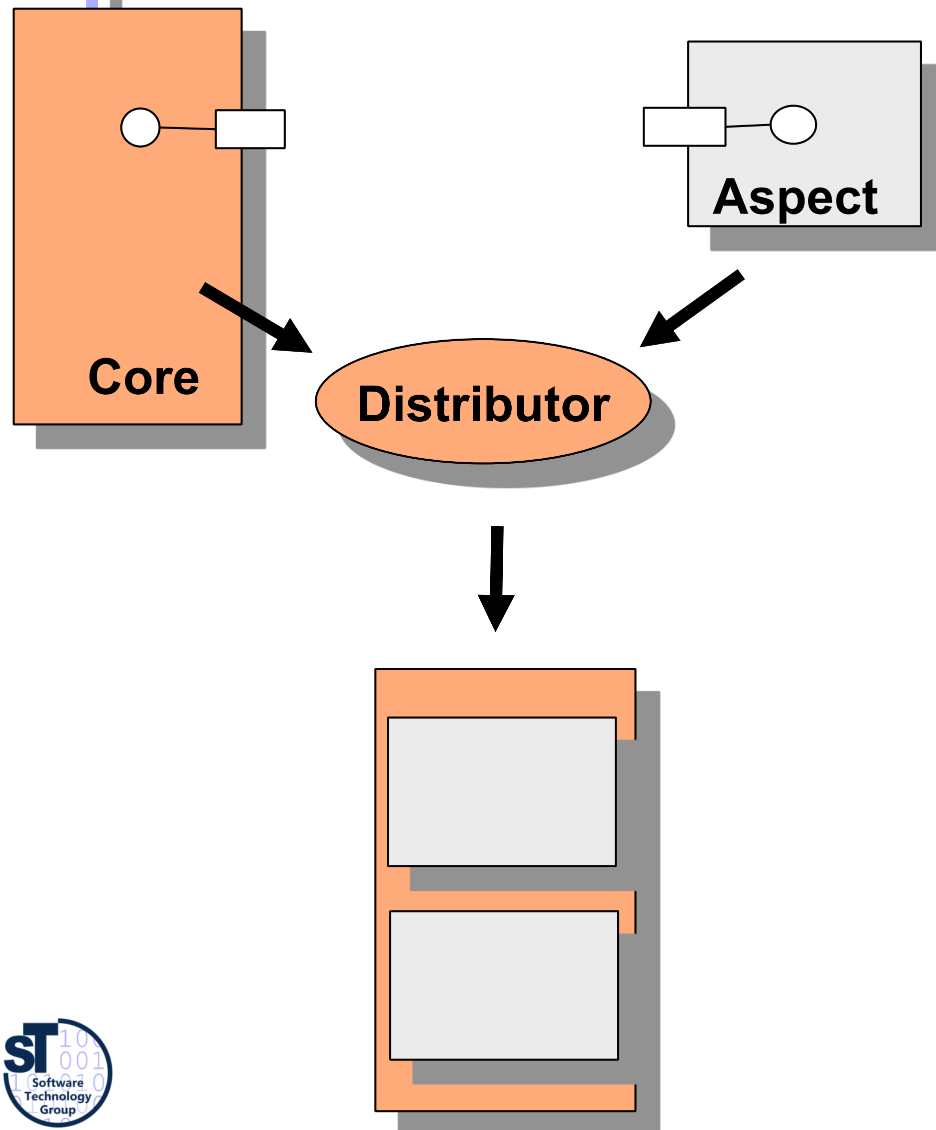
Persistence aspect

Debugging aspect

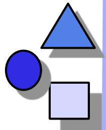
Persistence aspect

Debugging aspect

Aspect Weavers Distribute Advice Components over Core Components



- Aspects are *crosscutting*
- Hence, aspect functionality must be *distributed* over the core



Aspect Systems As Composition Systems

Component Model

Core- and aspect components

Aspects are relative and crosscutting

Binding points: join points

Composition Technique

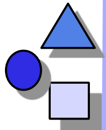
Adaptation and glue code by weaving

Weaving is distribution

Weaving Language

Composition Language

Full-Fledged Composition Systems



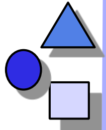
Composition Systems

Component Model

Composition Technique
Composition Operators
Black and Gray-Box

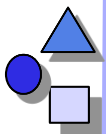
Composition
Expressions

Composition Language



Composition Systems

- Hyperspace Programming [Ossher et al., IBM]
- Piccola [Nierstrasz et al., Berne]
- Metaclass composition [Forman/Danforth, Cointe]
- Invasive software composition (ISC) [Aßmann]
- Formal calculi
  - Lambda-N calculus [Dami]
  - Pi-L calculus [Lumpe]

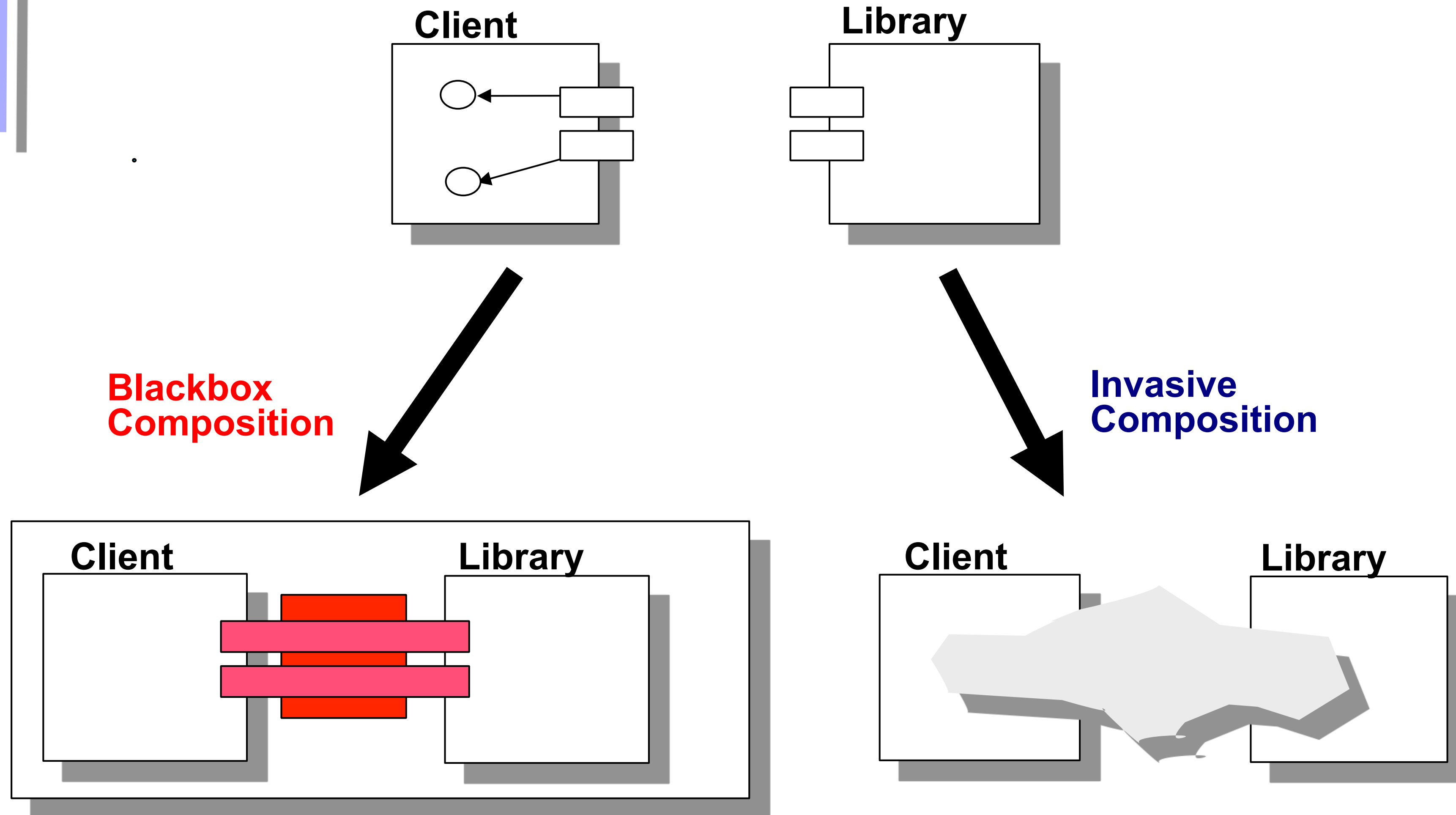


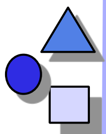
Composers Generalize Connectors (ADL Component Model)

components + composers + variation points

components + *connectors* + *ports*

Connectors are Composition Operators



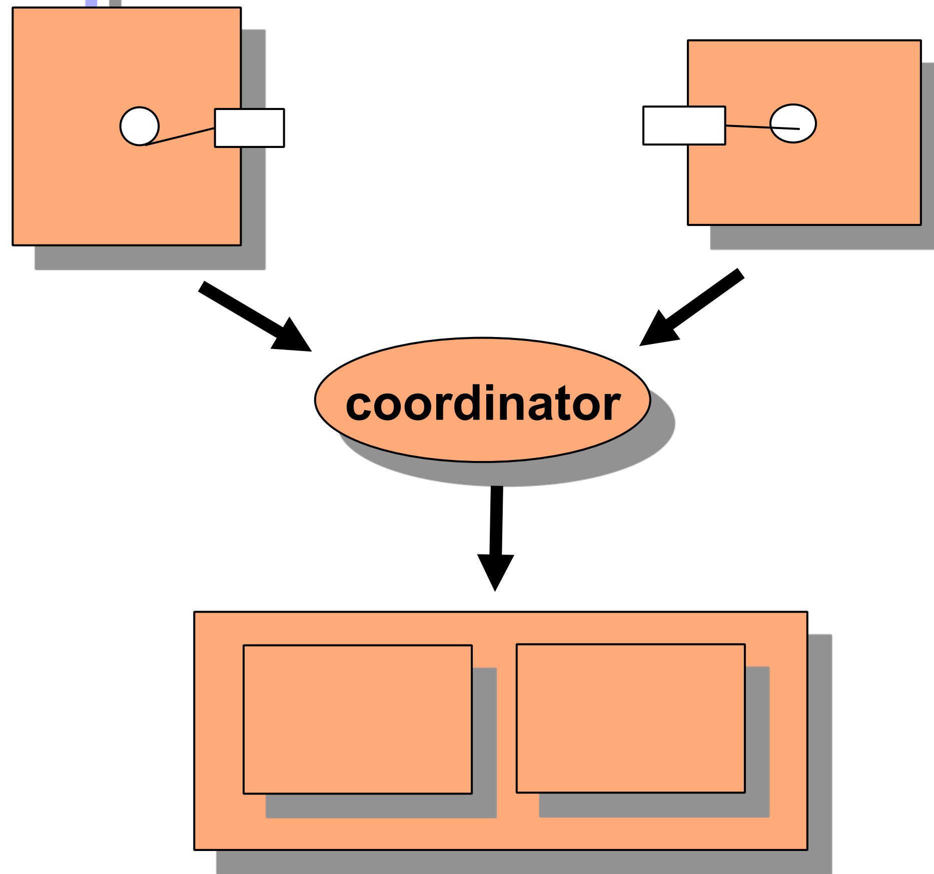


Composers Generalize Skeletons (Coordinators)

components + composers + variation points

components + *skeletons* + *ports*

Composers Can Be Used For Skeletons (Coordinators)



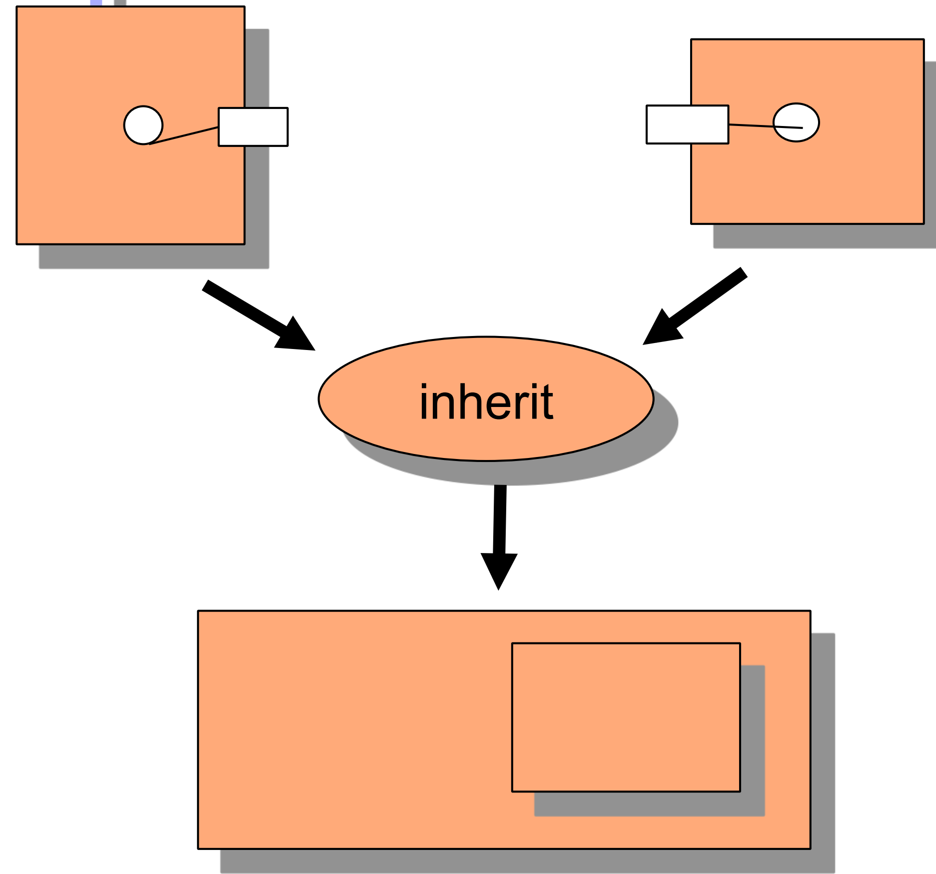
- Instead of functions or modules, skeletons can be defined over fragment components
- CoSy coordination schemes (ACE compiler component framework www.ace.nl)
  - Compose basic components with coordinating operators

Composers Generalize Inheritance Operators (Classes as Components)

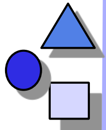
components + composers + extension points

components + *mixin* + *feature lists*

Composers Can Be Used For Inheritance



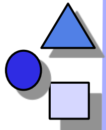
- Extension can be used for inheritance (mixins)
- inheritance :=
  - copy first super document;
  - extend with second super document;



Composers Generalize View Extensions

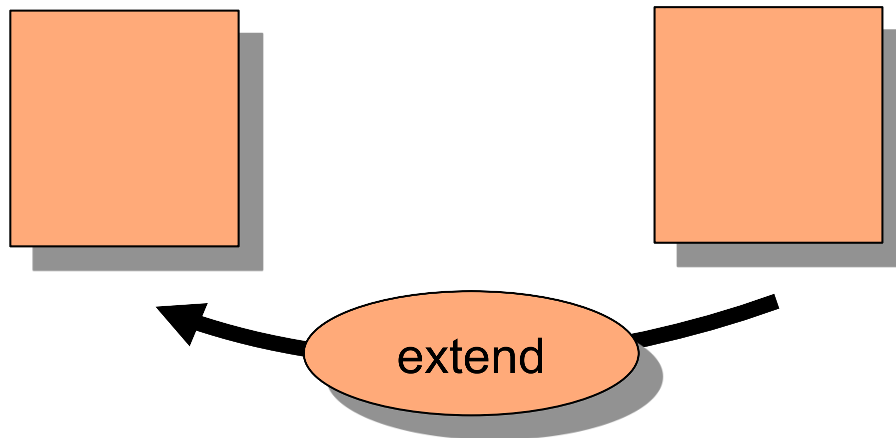
components + composers + extension points

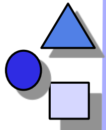
components + *extend* + *views*



Composers Generalize View-based Extensions

- A *core component* is extended by a *view component*



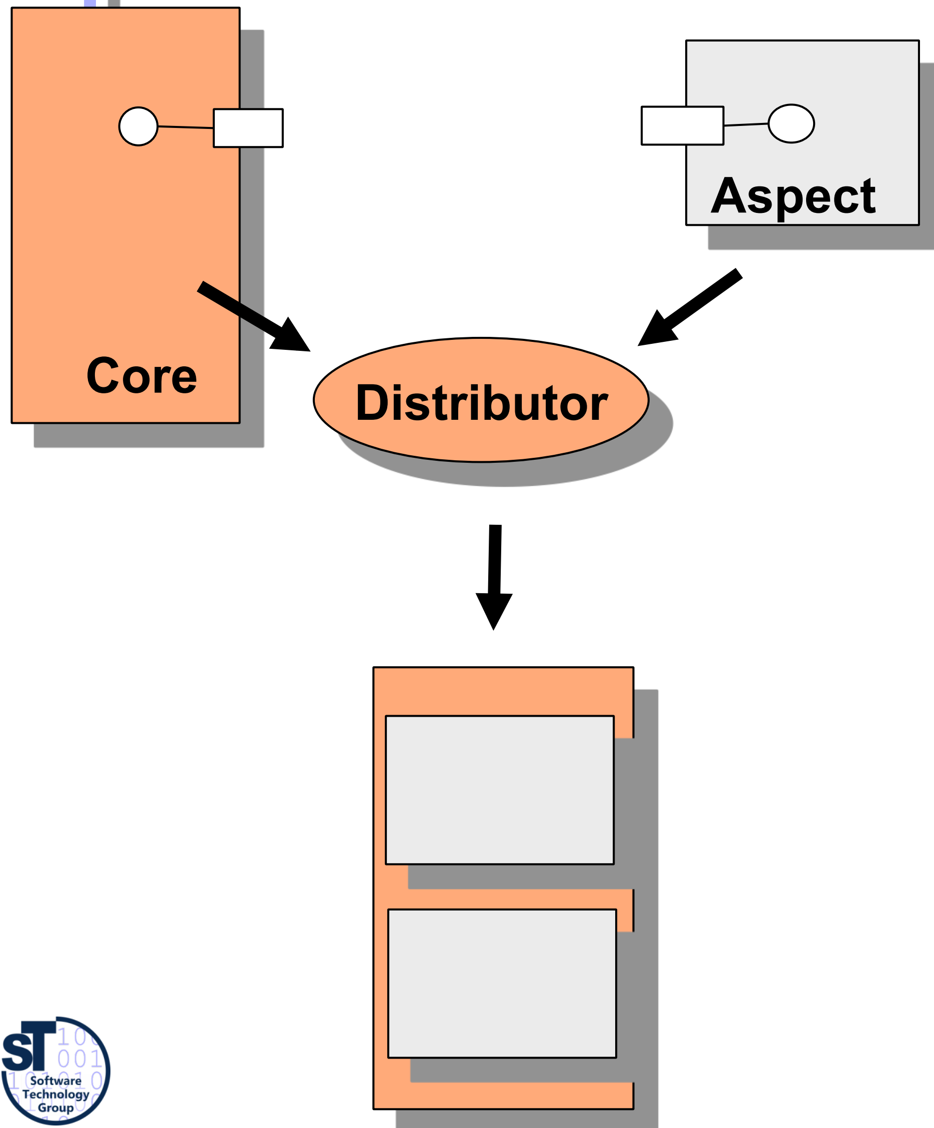


Composers Generalize Aspect Weavers

components + composers + extension points

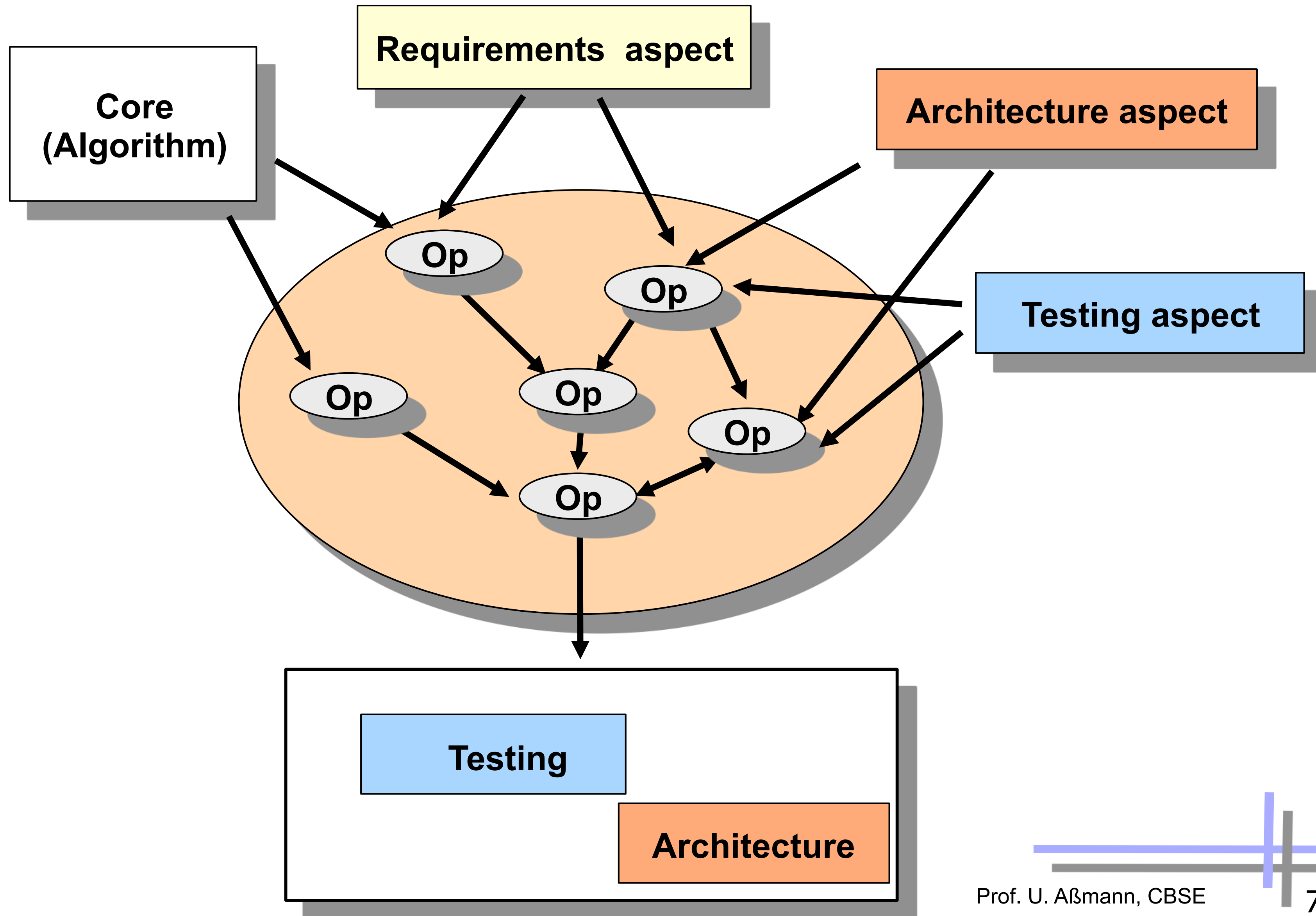
components + *weaver* + *join points*

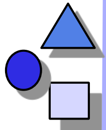
Composers Generalize Aspect Weavers



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

Weavers As Distributors



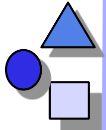


Composition Languages

- Composition Languages describe the structure of the systen in-the-large (“programming in the large”)
- Composition programs combine the basic composition operations of the composition language

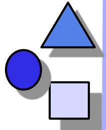
- Composition languages can look quite different
  - Standard languages, such as Java
  - Makefiles
- Enables us to describe large systems

Composition program size	1
System size	10



Conclusions for Composition Systems

- Components have *composition interface*
  - 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



Steps in System Construction

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

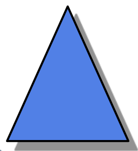
System composition
(System generation)

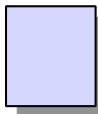
System compilation

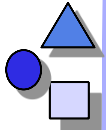
System deployment

System execution

What Have We Learned?

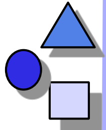






Component-based Systems

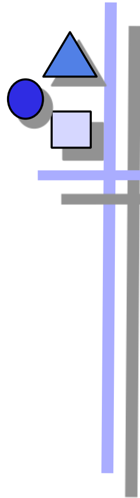
- ... are produced by component systems or composition systems
- ... have a central relationship that is tree-like or reducible
- ... support a component model
- ... allow for component composition with composition operators
  - ... and – in the large – with composition languages
- Historically, component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



The Ladder of Composition Systems

Aspect Systems	**View Systems**	**Software Composition Systems**
Aspect Separation	**Composition Operators**	**Composition Language**
Aspect/J	***Composition Filters*** ***Hyperslices***	***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