

# Component Models

Component-Based Software Engineering

[www.win.tue.nl/~mchaudro/cbse2005](http://www.win.tue.nl/~mchaudro/cbse2005)

M.R.V. Chaudron

Technische Universiteit Eindhoven

# Lecture Scheme

1. Introduction, Motivation, Concepts, Terms & Defs
2. Component Models & Architecture
3. Reuse & Economics of Reuse
4. Design & Specification of Components
5. Component Testing & Certification
6. Industrial Example of CBD - Koala
7. Generative Programming
8. Sw Composition & Predictable Assembly
9. t.b.d. / aAdvanced topics

Selection of topics from the CBSE area  
Some more stable than others.

# Today's Lecture

- ▶ Concepts of CBSE (continued)
- ▶ Component Models

Feedback on lecture notes are welcome.

# Why Component-based SW development

CBSE aims to improve

- Quality (Flexibility, Maintainability, Reliability) of software systems
- Productivity & Time-to-market of software development

through enabling the easy assembly of software from existing building blocks

**Table 2-2. Comparison of Conventional Approach to CBSD**

Characteristics	Conventional	CBSD
Architecture—how the system is set up	Monolithic	Modular
Components—the pieces of the system	Implementation and white box <sup>a</sup>	Interface and black box <sup>b</sup>
Process—how the system is put together	Big bang and waterfall (analysis to design to implementation to testing)	Evolution and concurrent engineering (component development to component integration)
Methodology—how the system changes through time	Build from scratch	Composition
Organization—market for buying and selling components	None	Specialized—component vendors, brokers, and integrators

<sup>a</sup>When a "white box" component is used, the programmer is given access to the source code from the component. The programmer then can change the code, and adaptation is relatively easy.

<sup>b</sup>"Black box" components cannot be adapted or changed.

Source: Aoyama, 1998.

# What is a Component?

A software component is a unit of composition with contractually specified *interfaces* and *explicit context dependencies* only

A software component is *independently deployable* and subject to *composition* by third parties.

Clemens Szyperski, 1997

# What is a Component?

A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction.

Grady Booch, *Software Components with Ada*, 1987

A component is a piece of software small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability.

Jed Harris, President of CI Labs (Jan. 1995).

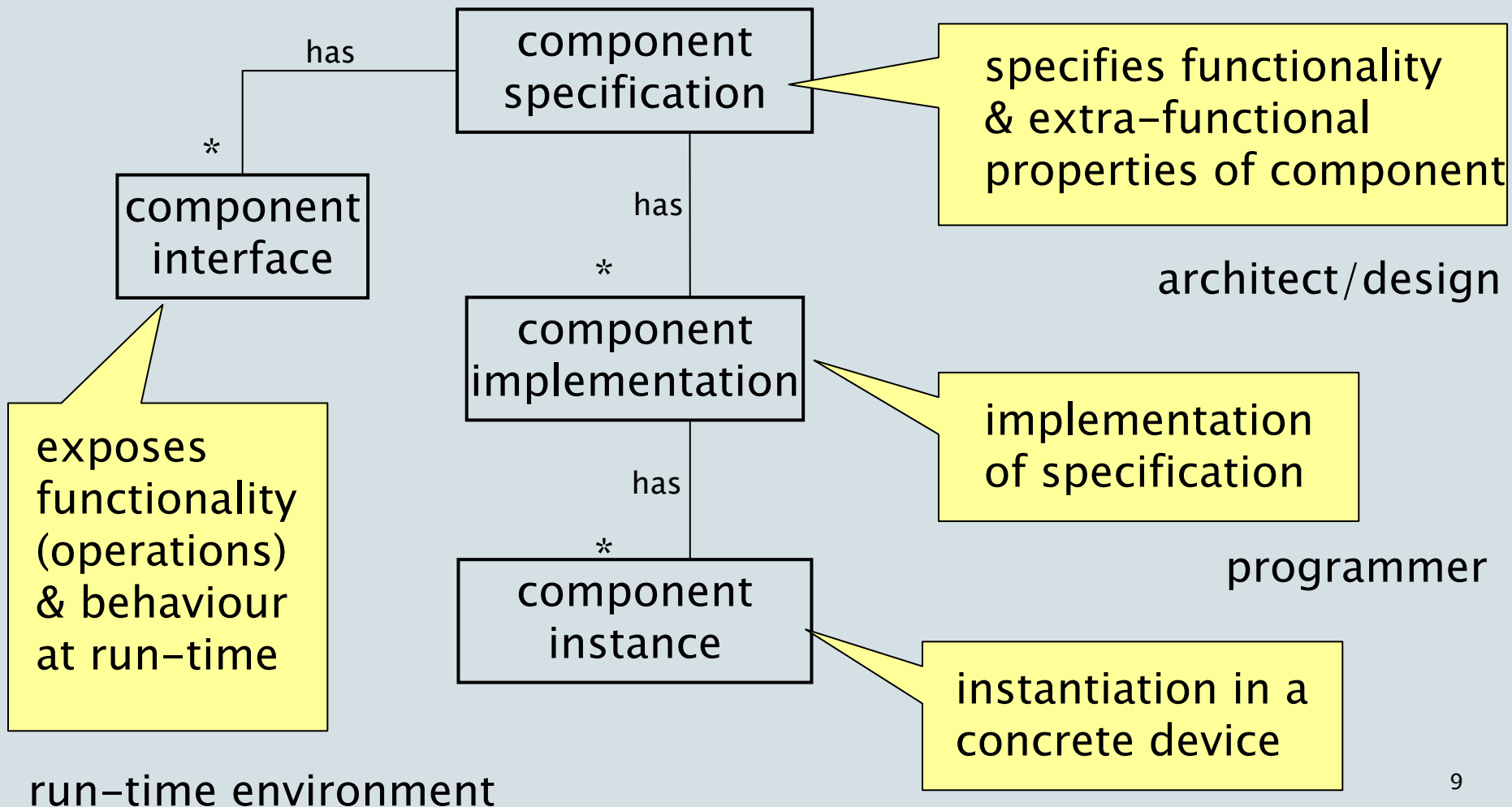
# What is a Component?

- a unit of
  - independent production, acquisition, deployment, and maintenance
  - replacement
  - reuse
  - composition

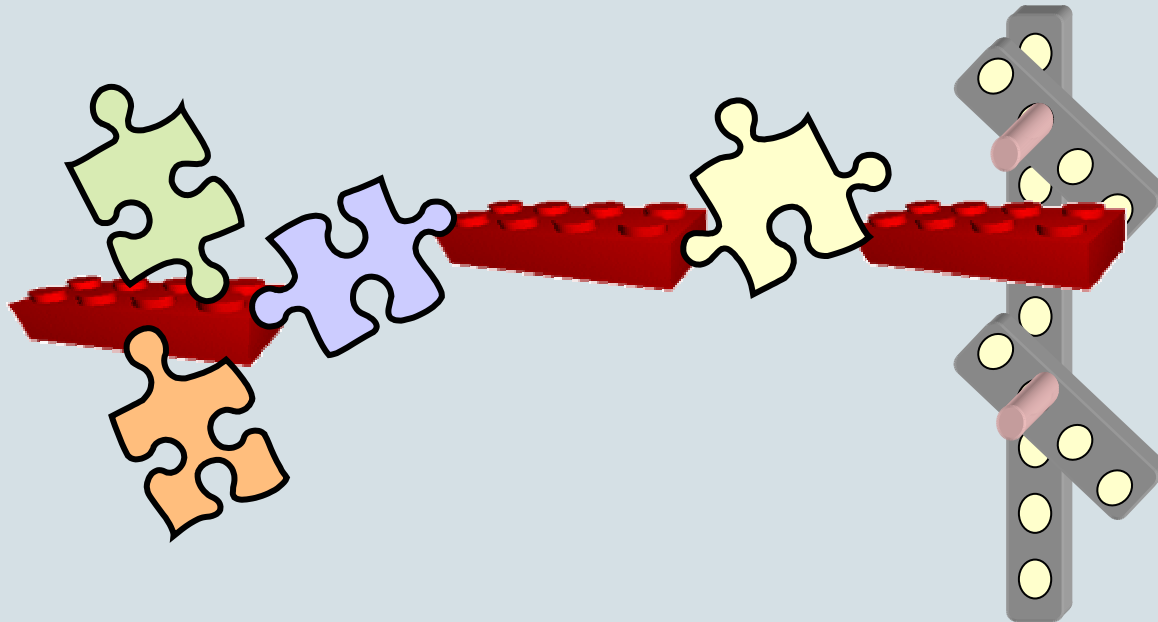
} properties depend on how they are used
- a package of cohesive services
  - self-contained
  - encapsulates design decisions
  - explicit dependencies
  - denotes a single abstraction
  - generic (application independent)
    - configurable
  - loosely coupled

} *intrinsic* properties

# Useful Distinction

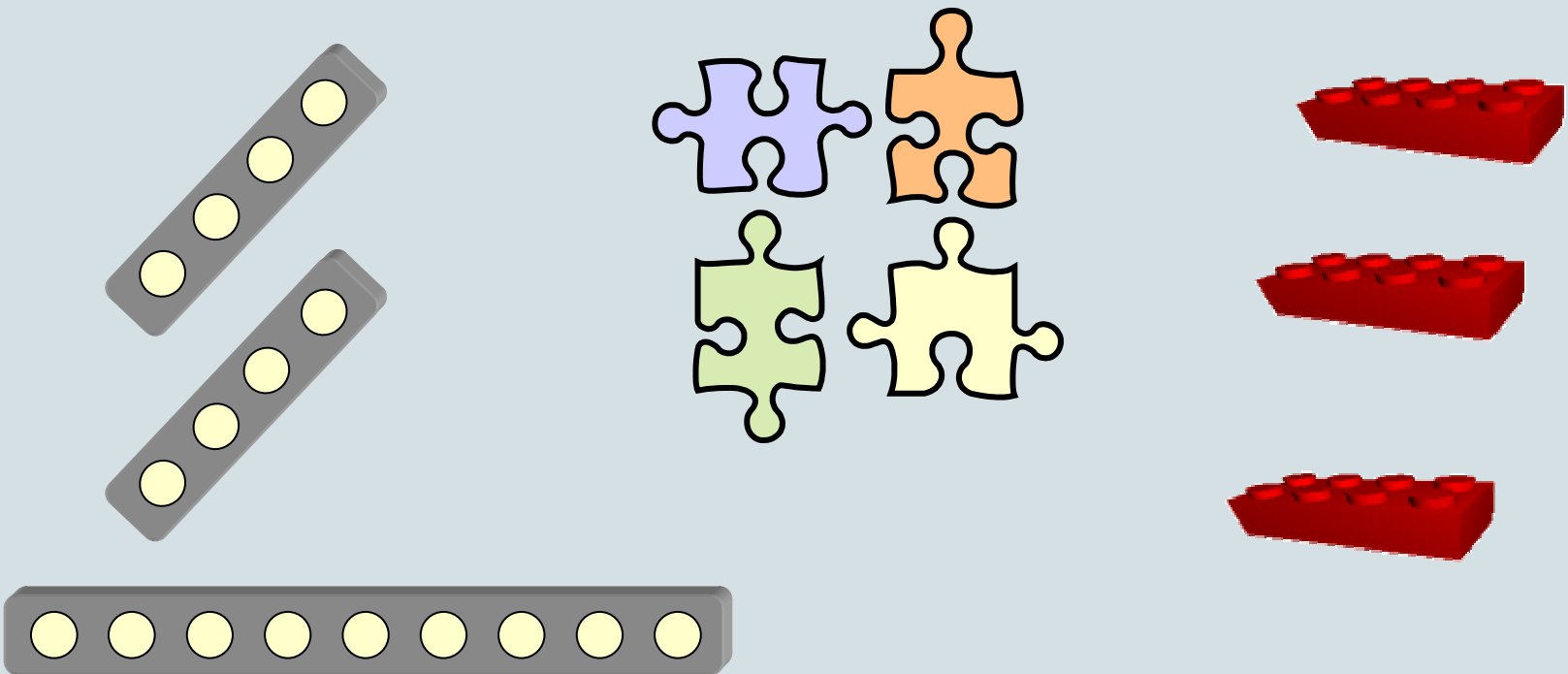


# CBD in Practice



Lego + Fisher Technik + Meccanno + Ministek + ...

A component can be used within the scope of a component-model



.Net, Enterprise Java Beans, Corba Components + ...

# Component Model

Definition A *component model* specifies the standards and conventions that are needed to enable the composition of independently developed components.

Includes/implies standards for

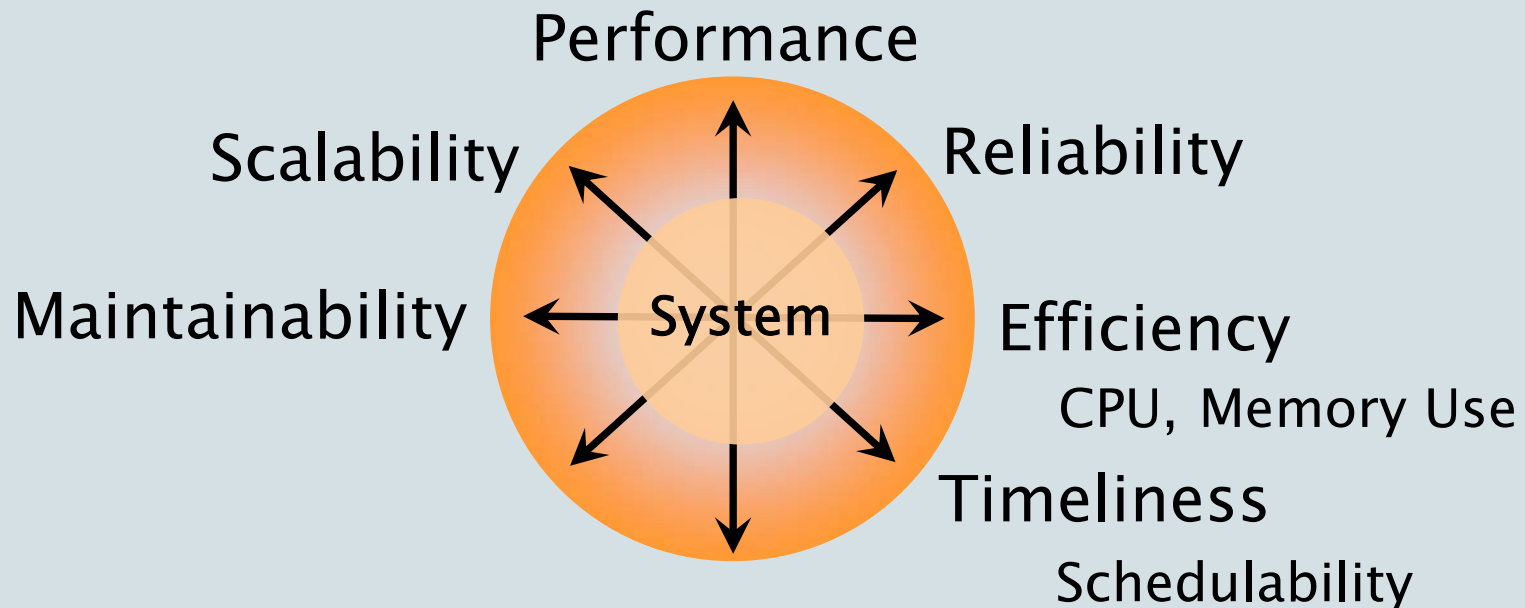
- Implementation technology
- Standard interfaces
- Specification/documentation/meta-data

# Multitude of Component Models

- Q: Why are there so many component models?
- A: because there are many different requirements
  - Technically: on component-systems
  - Organizationally: development process



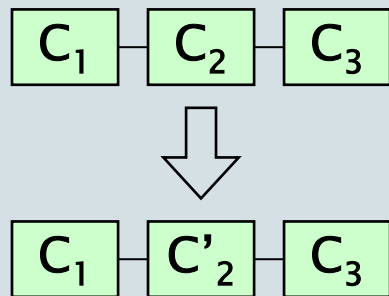
# Extra Functional System Properties



Different systems have different extra-functional requirements

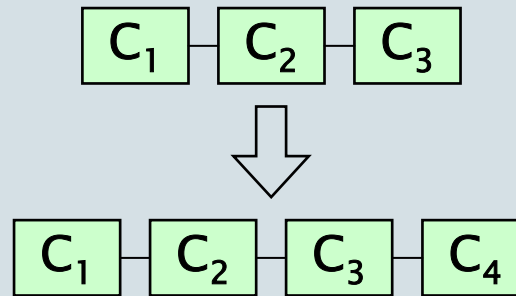
# Different usage requirements

## Substitutability



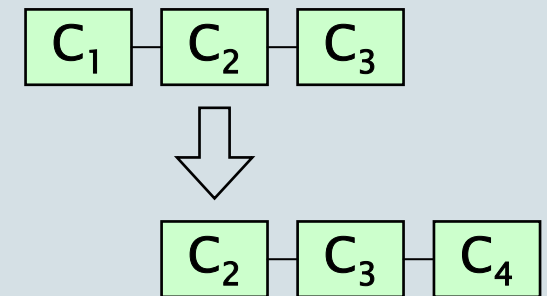
complete  
specification

## Extensibility



extensible  
architecture

## Decomposability



generic components,  
flexible architecture

At what stage is flexibility needed?

Design-, compile-, run-time

# Component Development Requirement: Black, Glass, Gray or White?

Black box: besides its interface and specification, no details are known

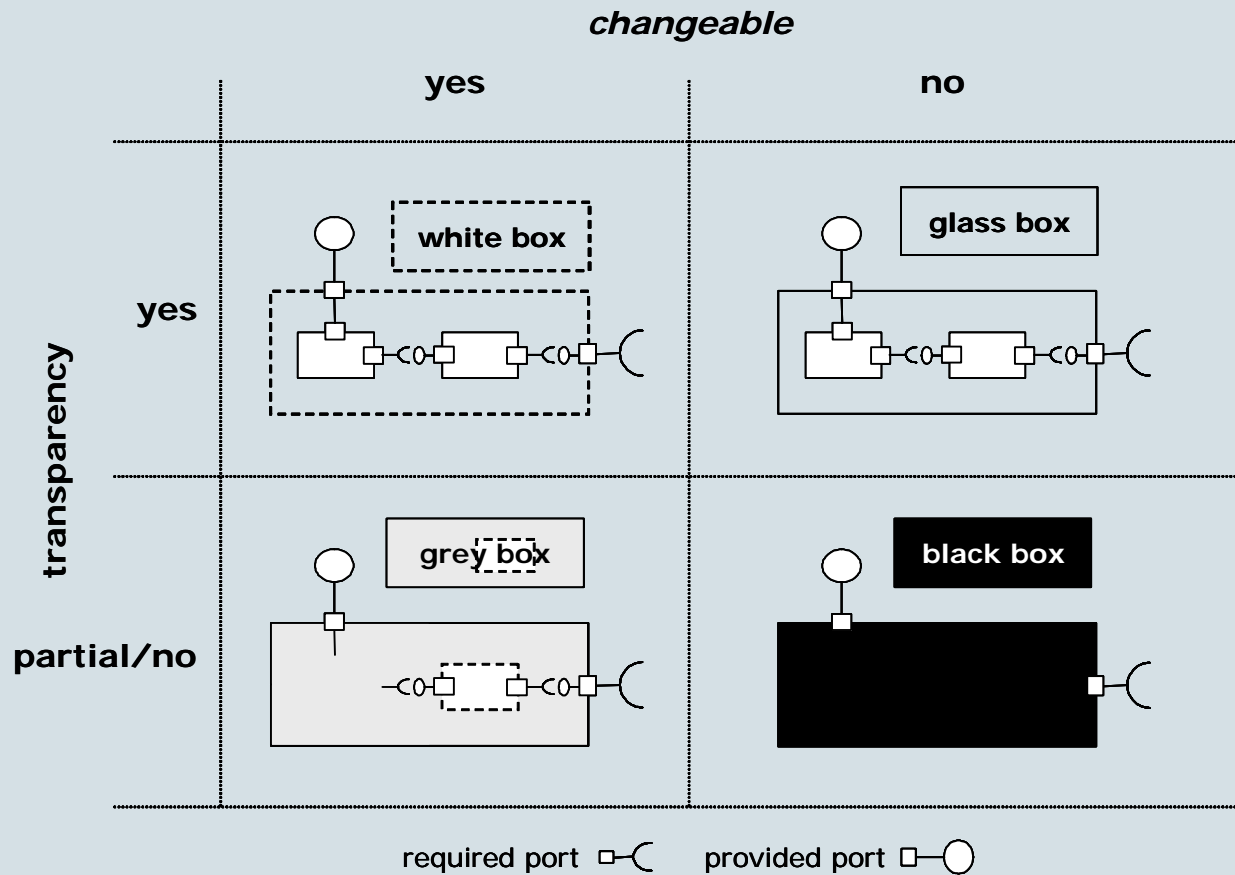
Glass box: inside of component can be seen, but not changed

White box: implementation of a component can be studied and manipulated

Gray box: part of the implementation can be studied and manipulated

Composition should be possible without reference to the internals.  
What needs to be specified in order to enable (predictable) composition?

# White, Glass, Grey & Black box



# Characteristics of Components: Black Box

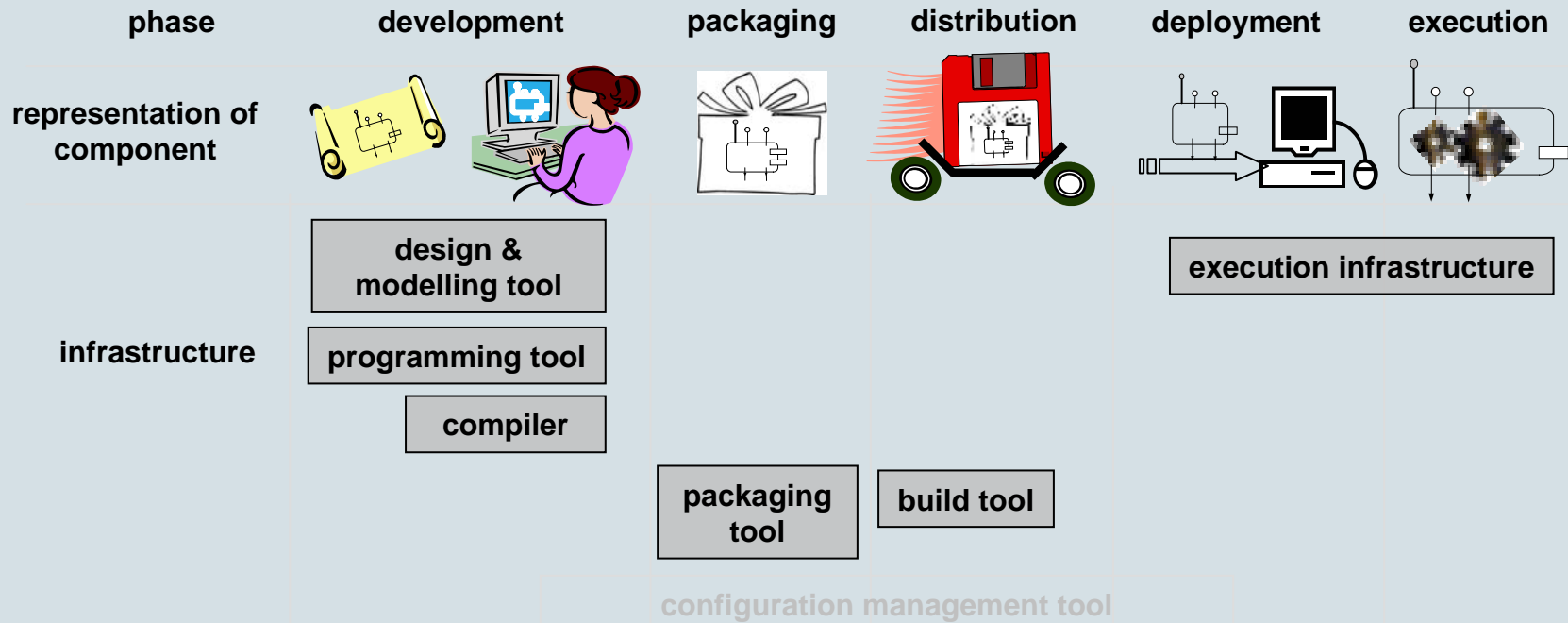
Black-box: has well defined interfaces that specify *what* it does, not *how* it does this.

Internals are hidden to

- prevent other components to build dependencies on internals
- allow fast understanding
- allow replacement of an implementation
- protect intellectual capital from disclosure

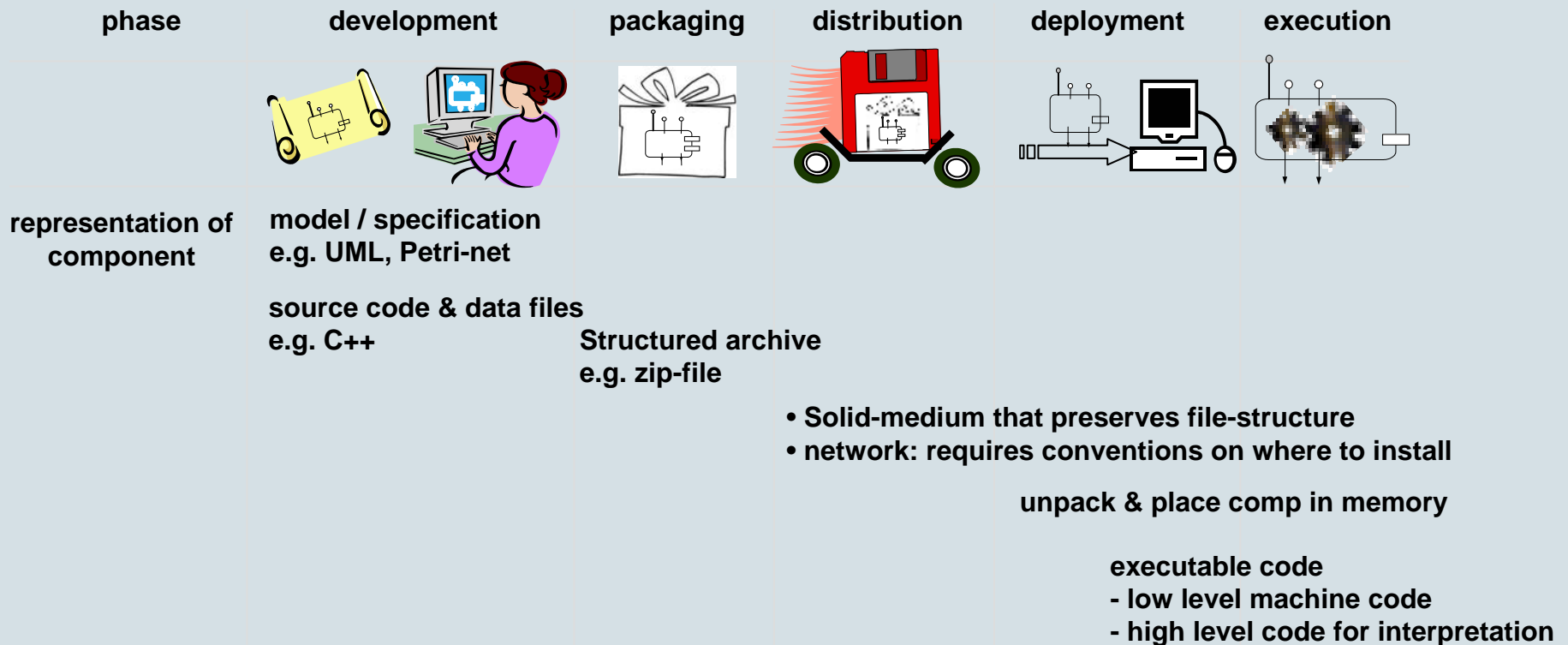
This complicates diagnosing system failures

# Component Development Lifecycle

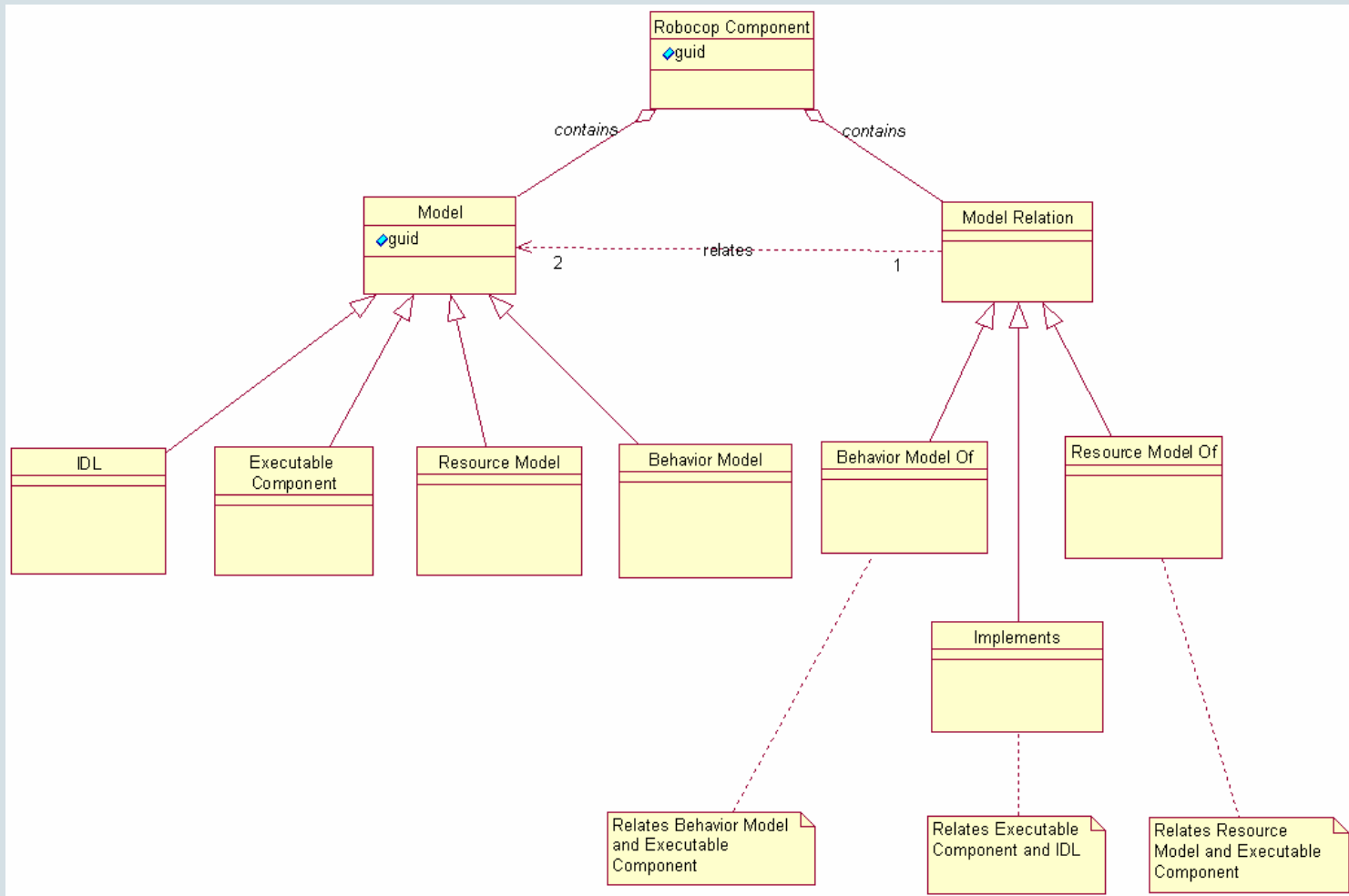


At any stage, components may be exchanged

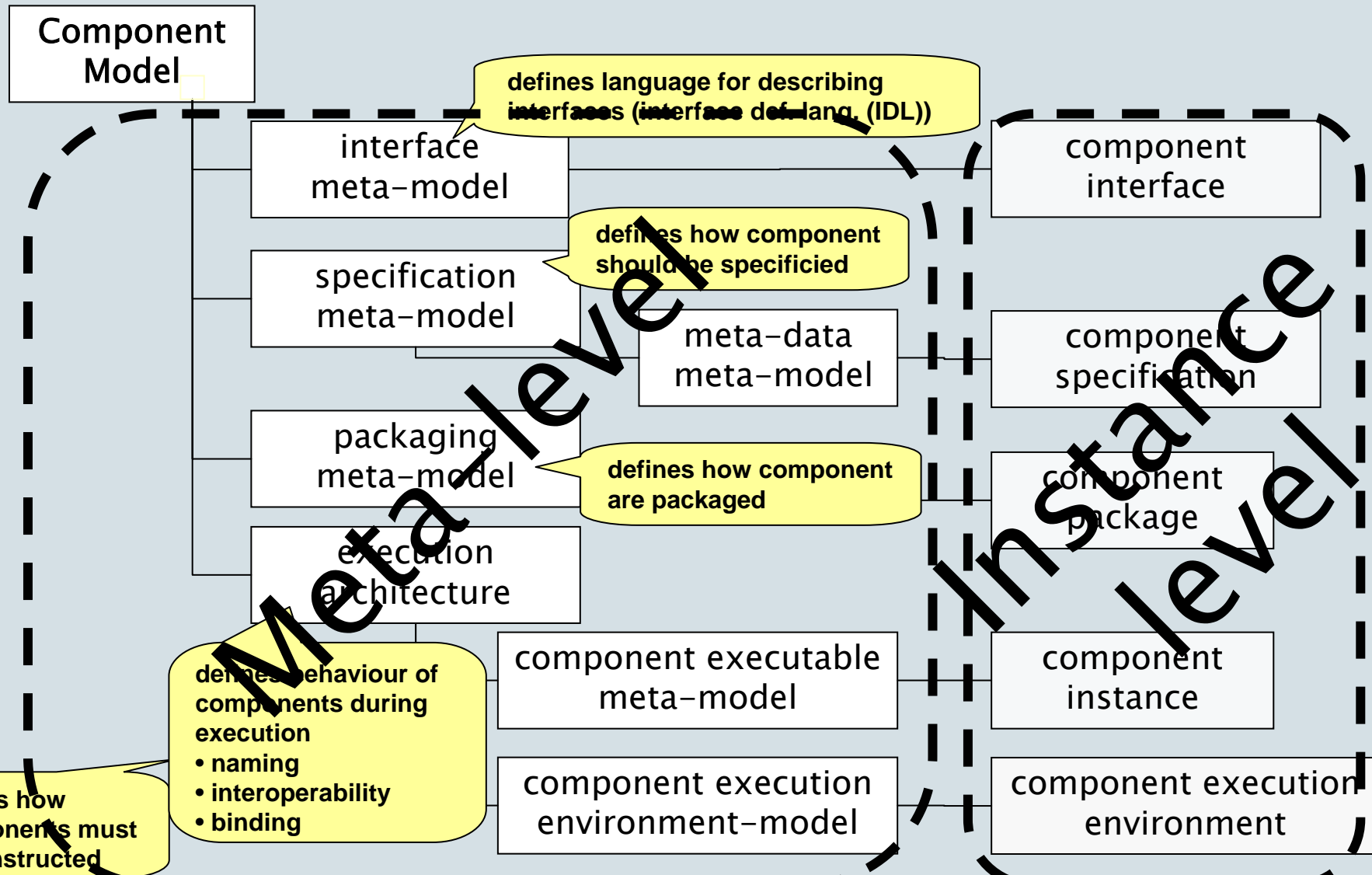
# Component Development Lifecycle



# Component Packaging (UML)

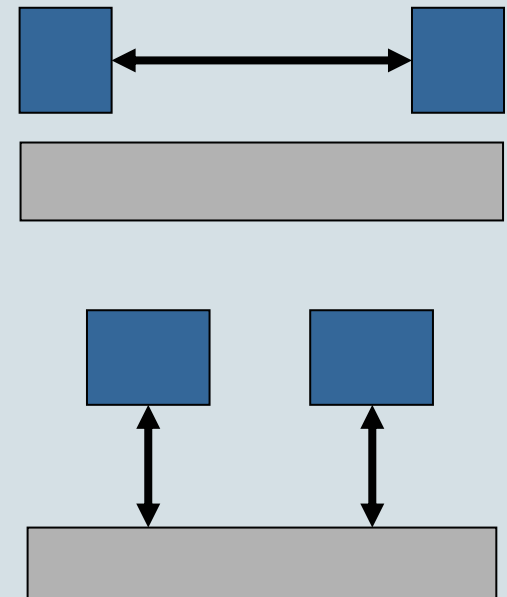


# Anatomy of Component Models



# Two types of Interfaces

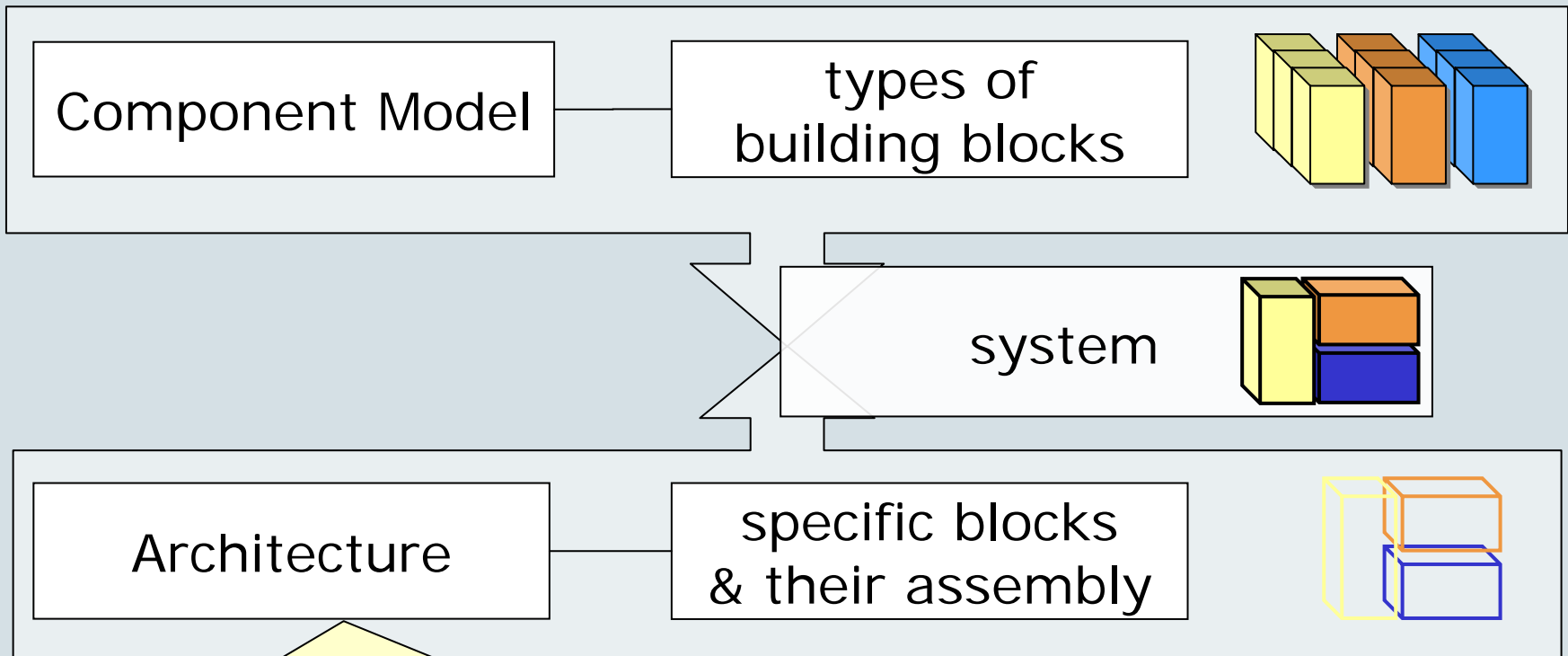
- Vertical standards:  
Finance, Healthcare, Automotive, ...
- Horizontal standards:  
Internet, OS 's, UI's, ...



# Two types of Interfaces

- Vertical standards:  
Finance, Healthcare, Automotive, ...
  
- Horizontal standards:  
Internet, OS 's, UI's, ...

# Component Model & Architecture

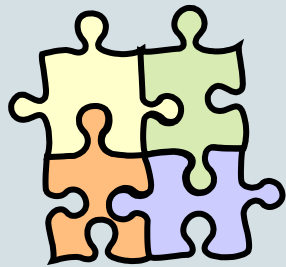


defines

- which specific components form a system
- which specific interfaces these components have
- which specific patterns of execution are used

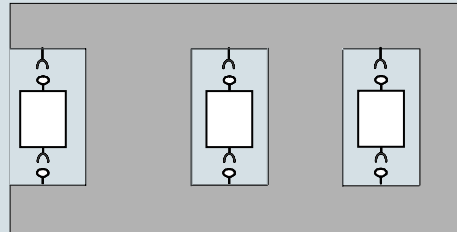
# Architecture vs. Generic Components

Architectural component



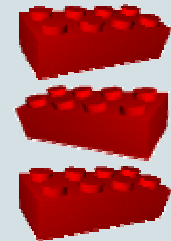
there is exactly one component that fits each place in the system

Framework component 'plug-in'



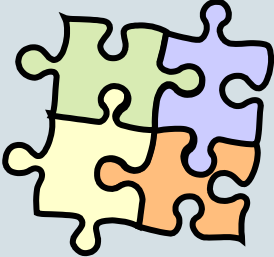
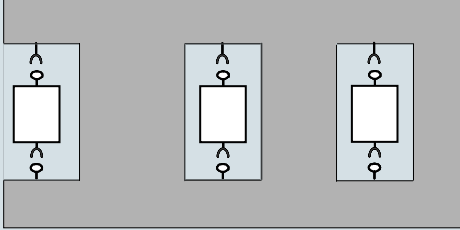
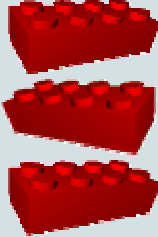
several components fit at a particular place in the system

Generic component

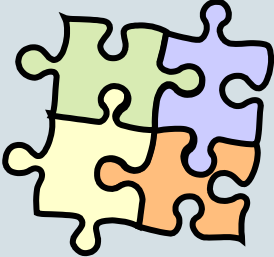
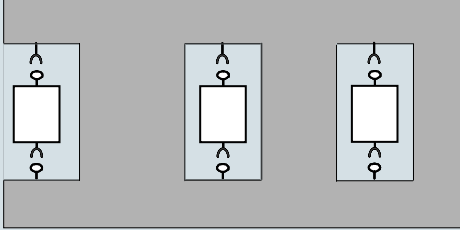
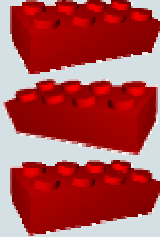
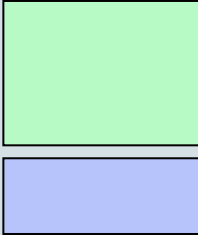
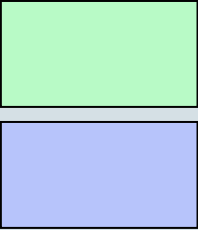
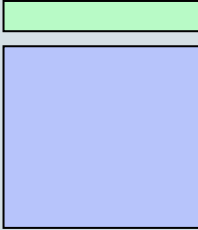


every component fits at every place

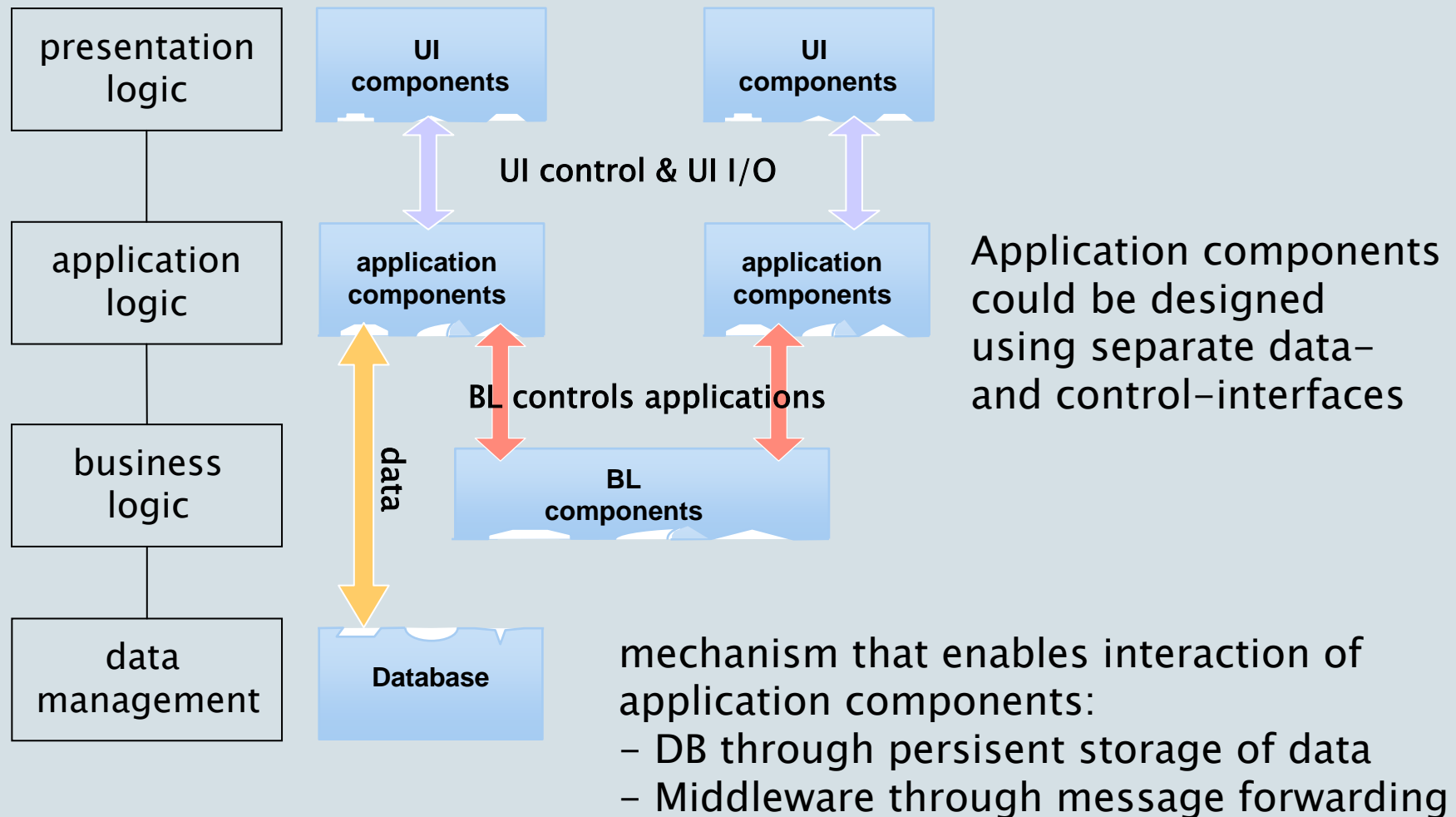
# Architecture vs. Generic Components

Architectural component	Framework component 'plug-in'	Generic component
 <p>there is exactly one type of functionality that fits each place in the system</p>	 <p>specific types of function fit at a particular place in the system</p>	 <p>functionality can be combined arbitrarily</p>
<b>independence of component design</b>		
Design of components is coordinated	Design of components is scoped	Design of components is independent
<b>Extensibility of component</b>		
Extension by architecture only	Internet-browsers	
<b>example</b>		
Architecture XYZ	Internet-browsers	Unix' Pipe & Filters

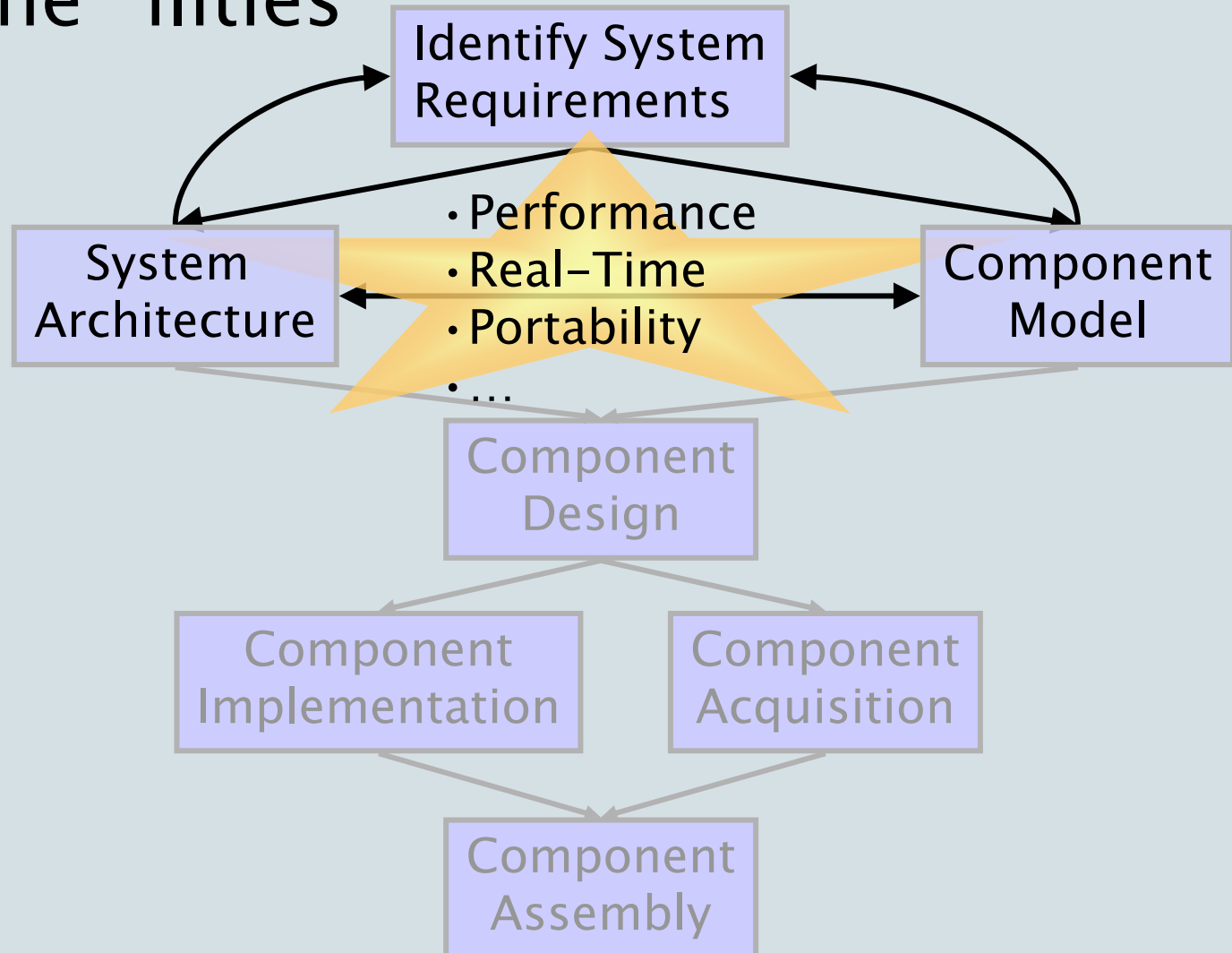
# Architecture vs. Generic Components

Architectural component	Framework component 'plug-in'	Generic component
		
relative contribution to component model		
<div data-bbox="47 1011 312 1153" style="background-color: #e0ffe0; padding: 5px;">architecture driven features</div> <div data-bbox="47 1182 293 1282" style="background-color: #e0e0ff; padding: 5px;">basic interoperability</div> 		

# Example: Client / Server Systems



# Architecture & Component Model determine \*ilities



# Aspects of Component Models

A component model is a set of agreements that is needed to enable the *combination* of components.

A component model typically addresses

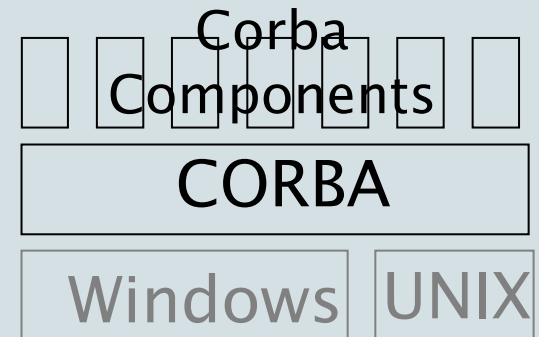
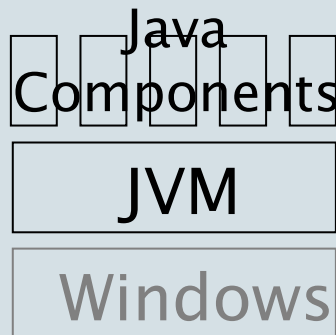
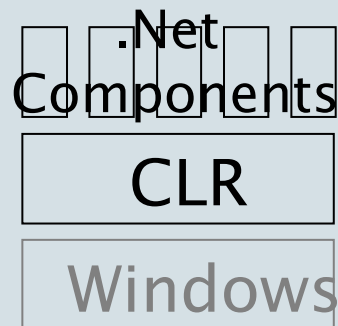
- Life-cycle management: instantiation, (de)activation, removal
- Binding mechanisms
- Interaction style
- Data exchange format
- Process model

Related: Packaging Model

# Component Platform

A component platform is run-time infrastructure of the component model.

For example



# Component Platform

Component Platforms typically provide support for

## Inter-component services

- binding
- interaction

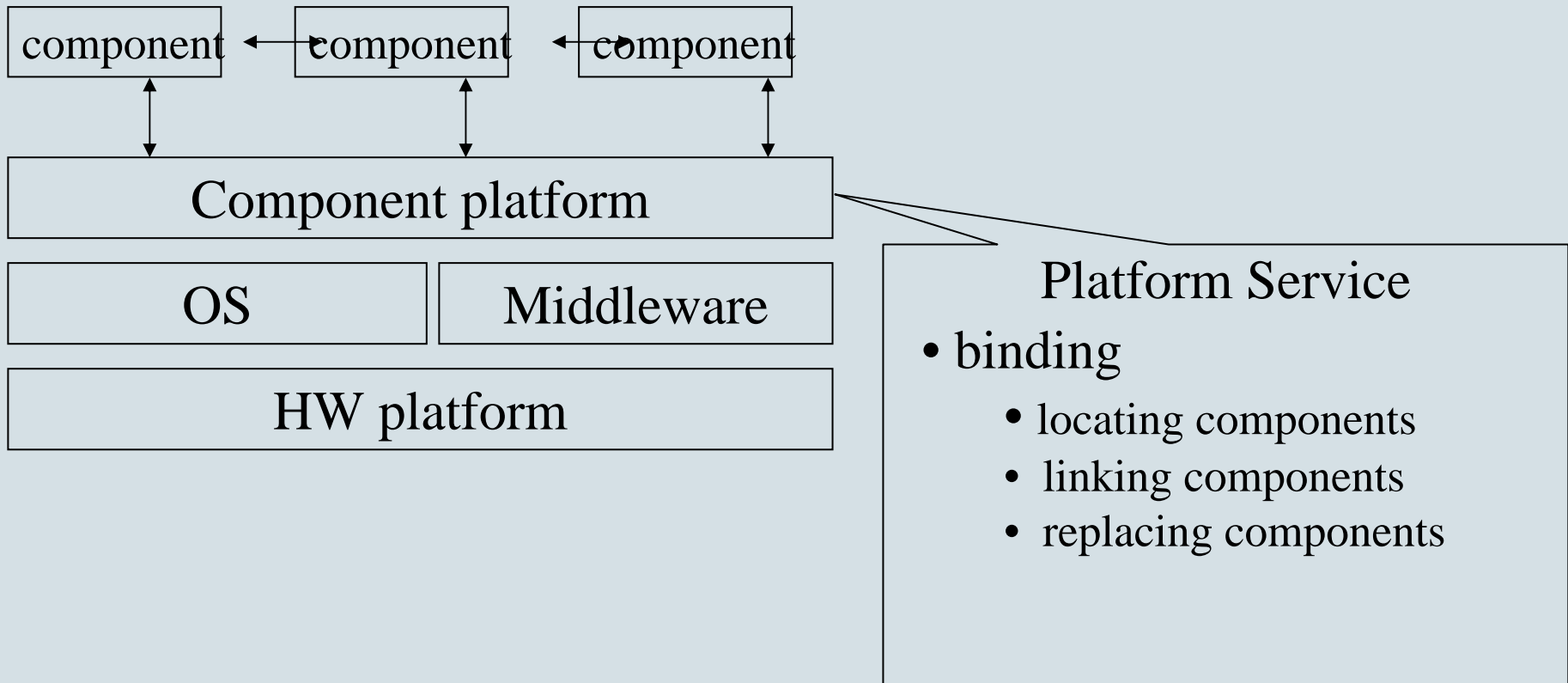
## Component lifecycles

- install, create, replace, start, stop, remove

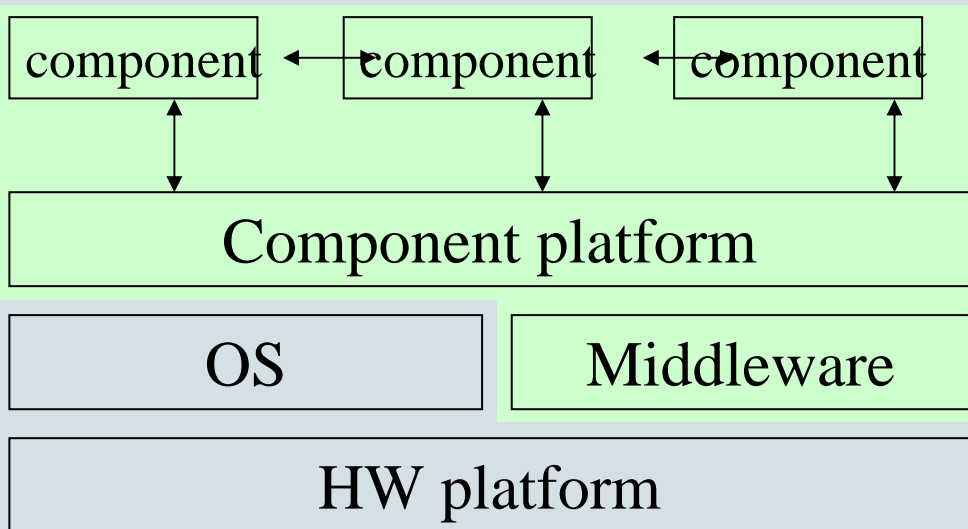
## Extra-functional / resource aspects

- scheduling
- quality of service management
  - (dynamic) load balancing
  - (re)negotiation
- security
- fault tolerance (replication)
- interoperability (language/OS)

# Architecture of Component Models



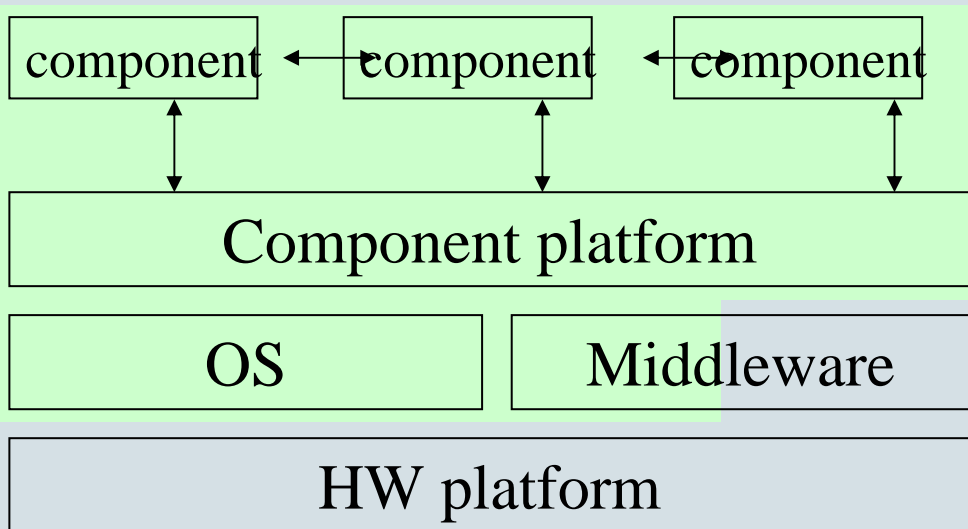
# Component Models: EJB



- EJB includes e.g.
- network services
  - transaction services
  - Java VM

better: EJB is tied to RMI  
which provides Middleware  
services

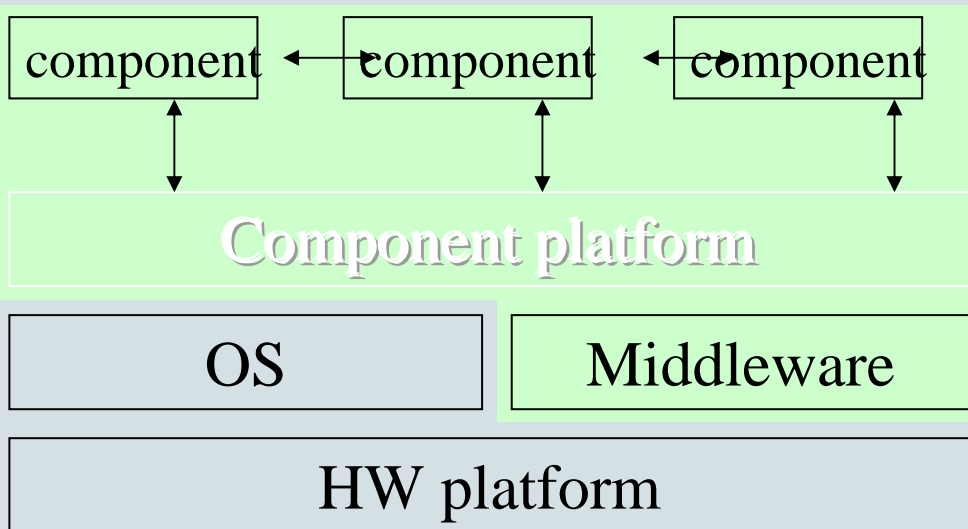
# Component Models: DCOM



## DCOM

- component support is built into OS
- includes some networking services

# Component Models: CORBA

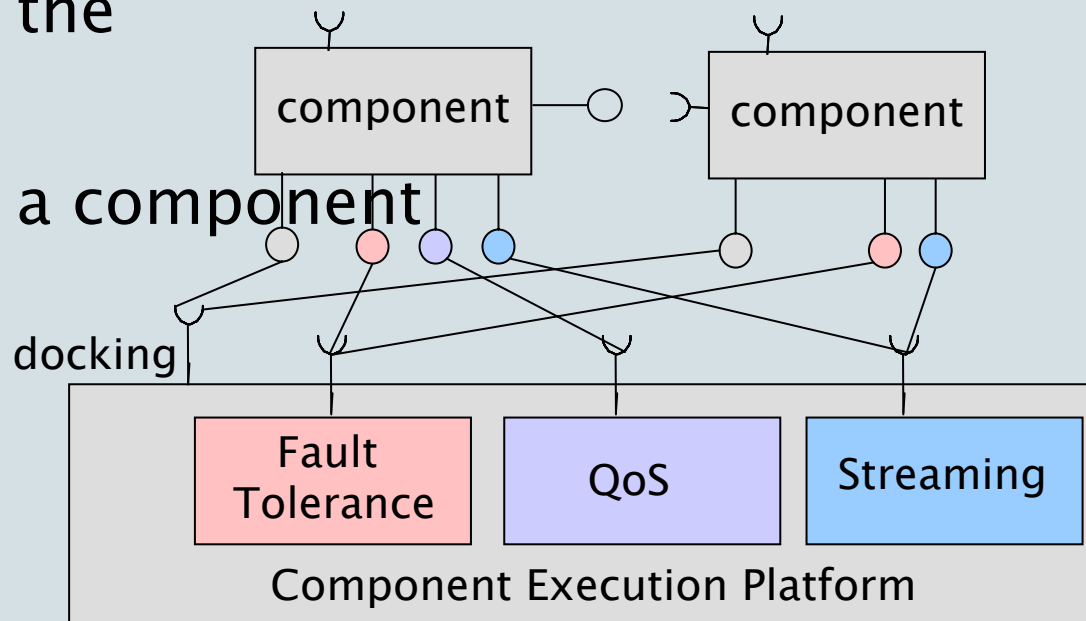


CORBA's original goal was a technology for enabling interoperation between OO applications; i.e. mainly a middleware.  
A component model is derived from that.

# Multiple Interfaces

A component model may provide multiple (optional) frameworks

- 'Docking' a.k.a Binding
  - how to connect to the infrastructure
  - how to connect to a component
- Fault Tolerance
- Quality of Service
- Streaming
- Transactions ...




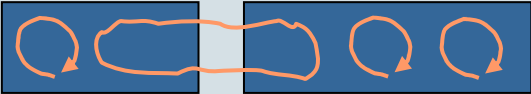


# Design Choices in Component Models

- control flow
- distribution
- mobility
- topology
- multiplicity
- binding time
- binding type
- interaction style
- platform features
- auxiliary

# Control Flow

## Multiplicity & Locus of control:

	within component	across components
$\leq 1$ / component		
$\geq 1$ / component		

# Distribution

A collection of component resides on:

- a single location
  - cpu, board
- network
  - (in)homogeneous
- open/closed network
  - internet vs. private network

Can a single component be distributed over multiple locations?

# Mobility

Can components move from one location to another?

- unlimited or do they need access-rights?
- do components move themselves or are they moved by a (resource) manager?

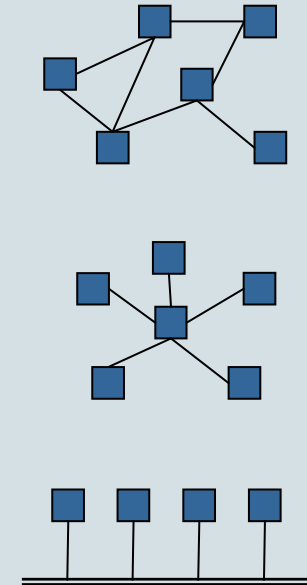
# Topology

How are components connected?

- network (node to node)
- star (central node)
- bus (central interconnection)

Changeability of the topology:

- static
- dynamic
  - changes in links
  - component creation/deletion
  - changes in component location



# Multiplicity

May multiple instances of a component exist concurrently?

# Interaction Style

What mechanism may components use to exchange data?

- request/response (rpc), asynchronous or synchronous
- datagram
- broadcast (multicast)
- blackboard
- streaming?

Multiplicity of interaction styles:

- single style vs. multiple styles

# Concluding Remarks

- A component is a building block that conforms to a component model
- Different component models aim to achieve different system quality properties
- A component model may be biased to specific types of architectures