

---

**ITI**

**Component-based Software Engineering – New  
Challenges in Software Development**

**Ivica Crnkovic**

**[ivica.crnkovic@mdh.se](mailto:ivica.crnkovic@mdh.se)**

**<http://www.idt.mdh.se/~icc>**

**Mälardalen University**

Ivica Crnkovic • Magnus Larsson  
editors



building  
reliable  
component-based  
Software  
Systems

---

**The talk is based on:  
Building reliable component-based  
systems**

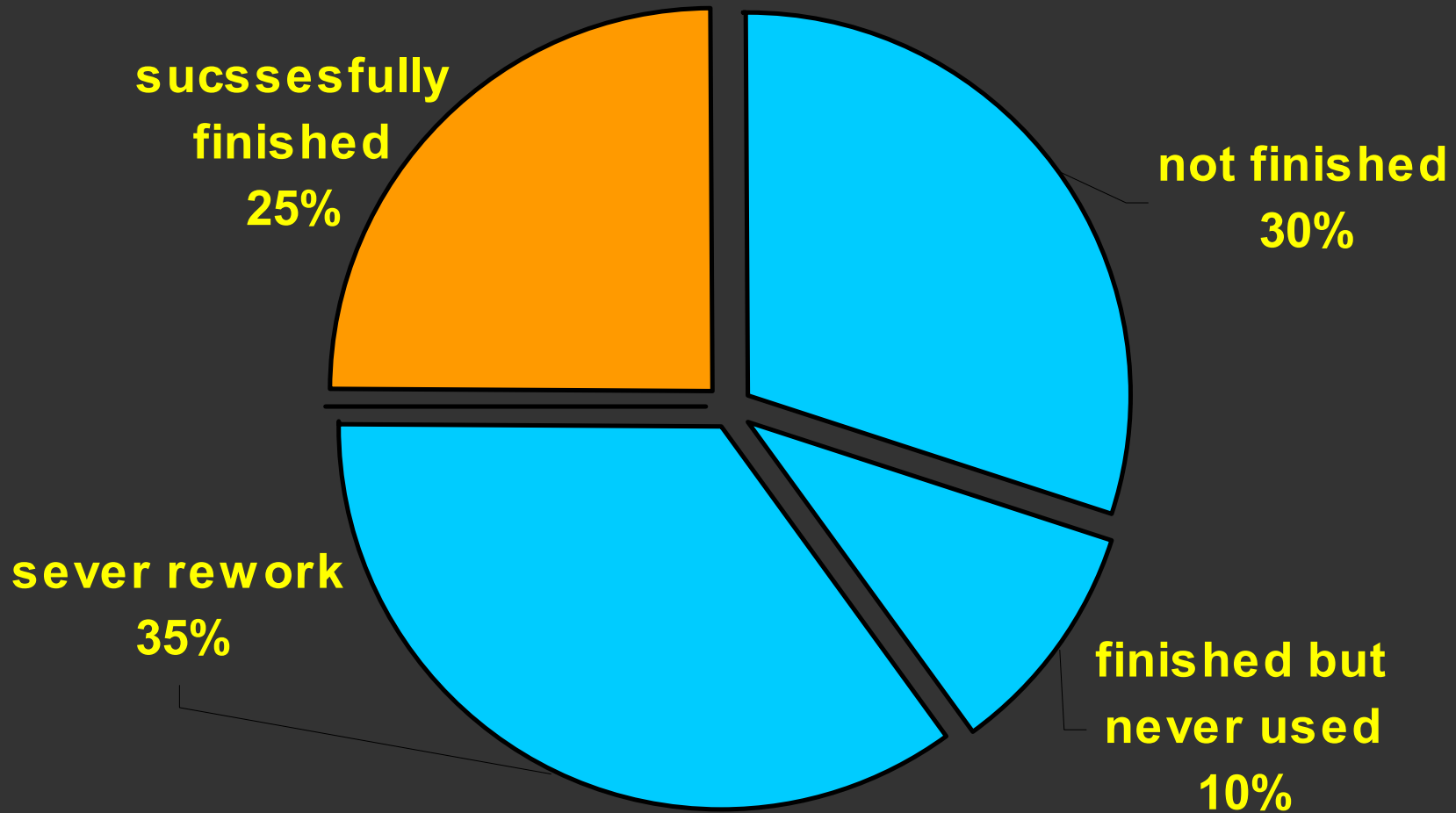
**Several distinguished researchers in  
Component-based Software  
Engineering participated**

---

## **A PESIMISTIC VIEW**

Software development is in a permanent crise

# Software development is in a permanent crises



SOFTWARE PROJECTS 1990

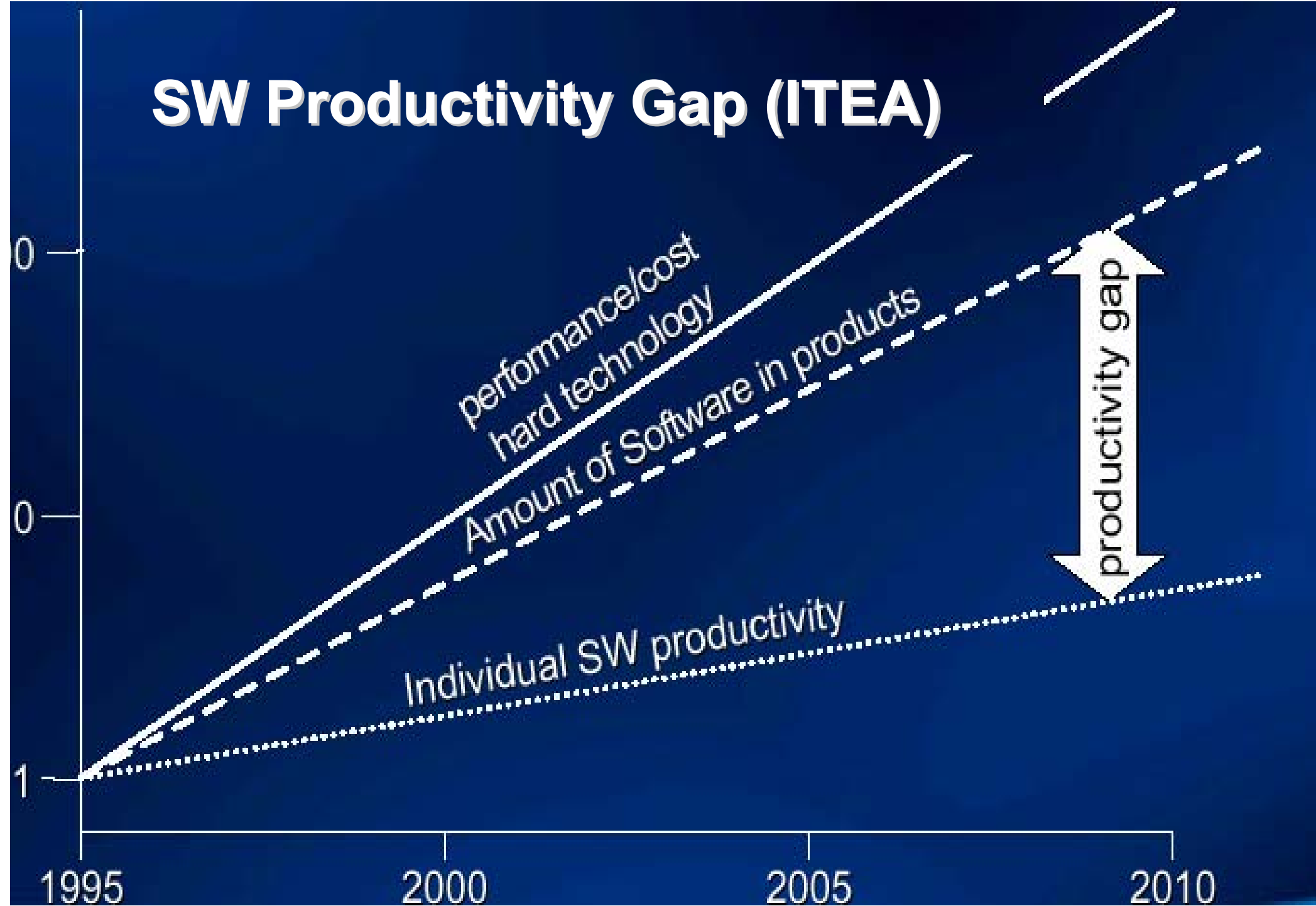
# Software development is in a permanent crises

---

## CHAOS 2001 study (The Standish Group)

- 28% of IT software development projects were successful
  - ☞ i.e. completed on time and on budget, with all features and functions originally specified
- 23% failed completely
- Among challenged projects;
  - ☞ cost overrun were 45%
  - ☞ time overrun 63%
  - ☞ required features delivered 67%.

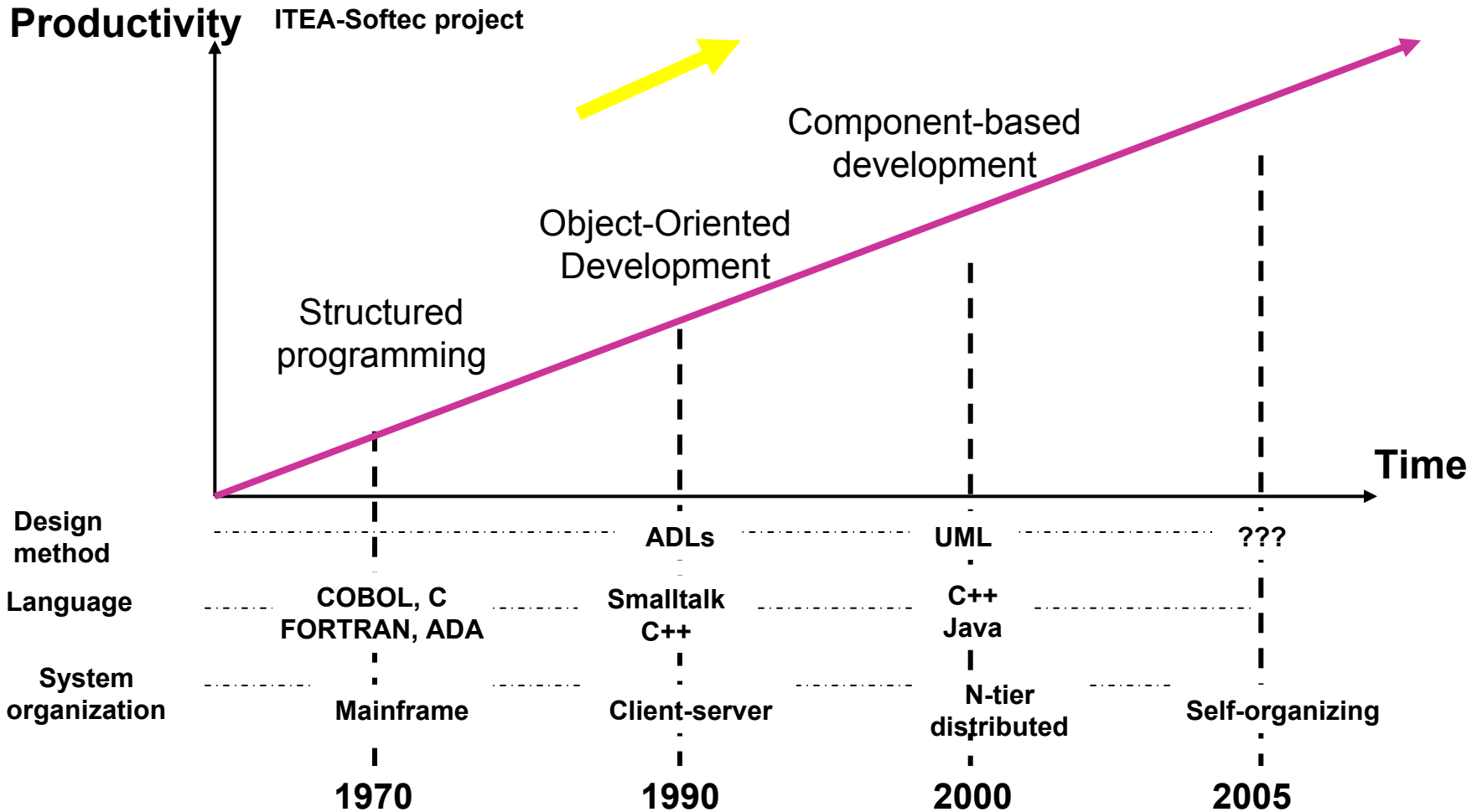
# SW Productivity Gap (ITEA)



# Software paradigm shift

© 2001 ITEA Office Assoc

ITEA-Softec project

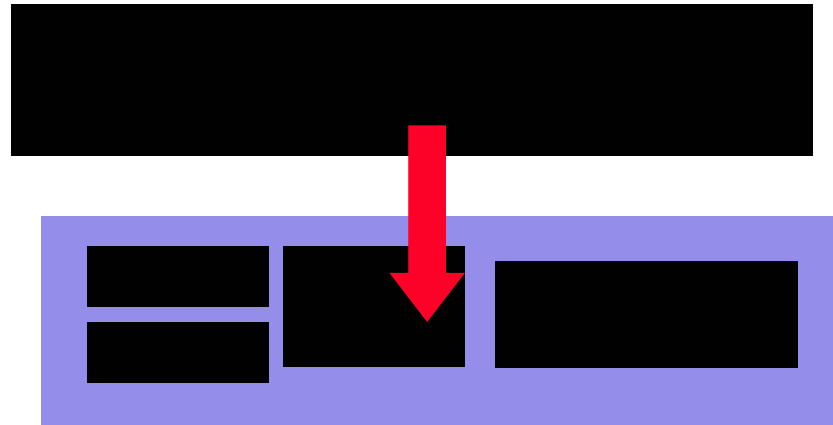


# How to solve the problems?

---

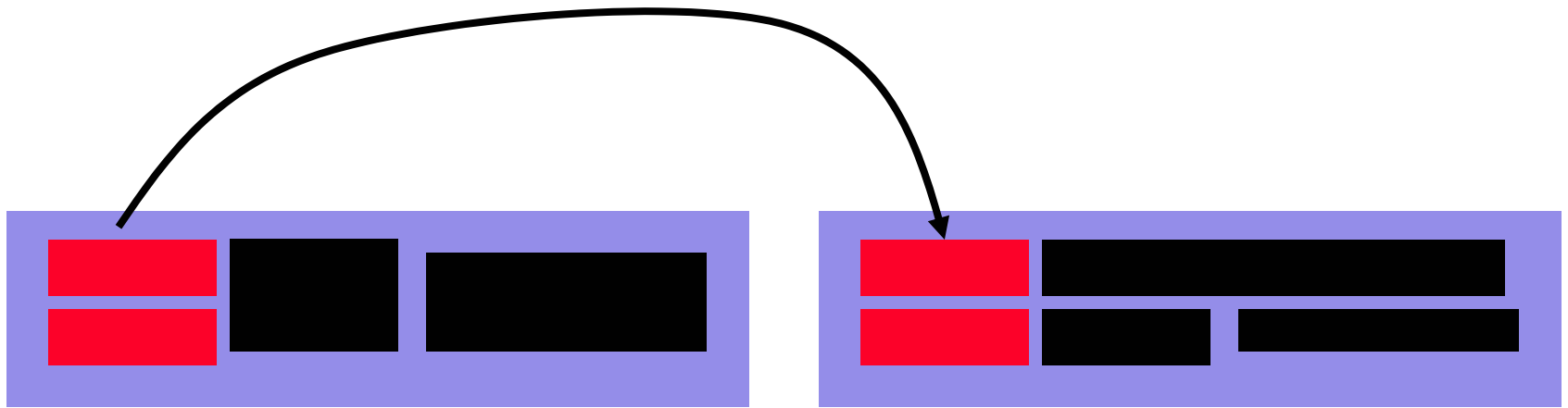
## □ Principle A

- Divide and conquer



## □ Principle B

- Reuse solutions

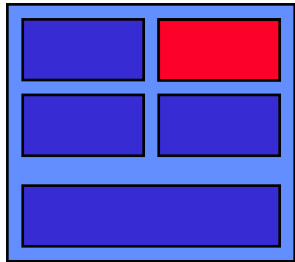


# Application development – granularity, reuse, modularity

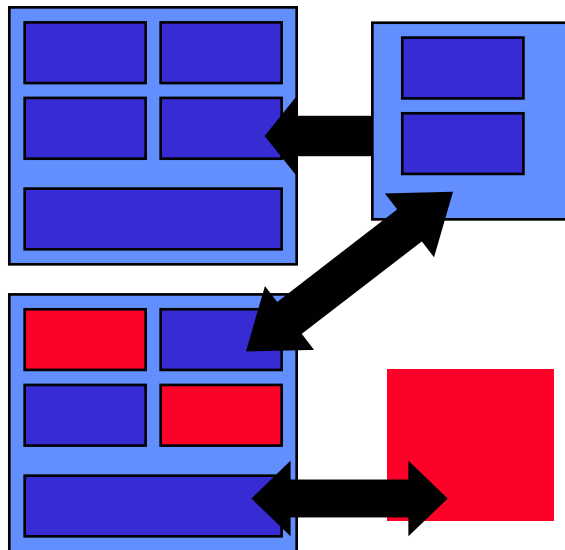
---



**1970 – Programming in small**  
**One application, one module (assembler)**



**1980 – Programming in large**  
**One application, several modules**  
**Separate compilation, final linking**  
**Common data**



**1980 – Client/server**  
**Even at run time separated (distributed applications)**

**Possibility to dynamically update**

# Reusability, modularity, evolution

## Object oriented approach

Encapsulation (information hiding), abstraction,

### □ Classes

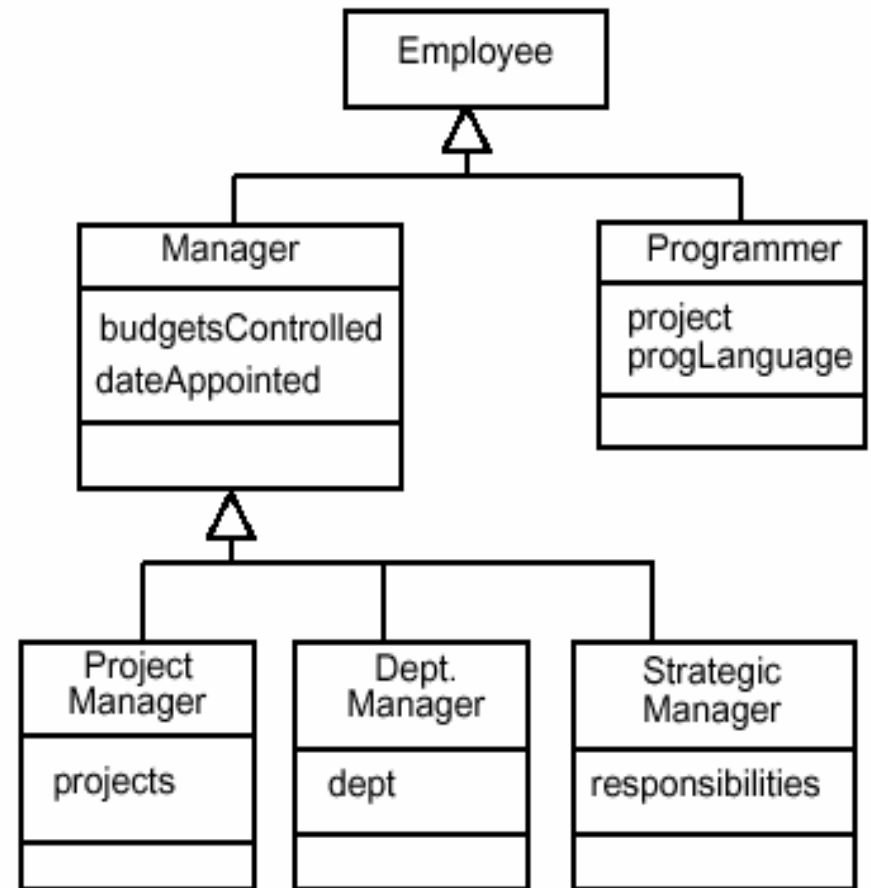
- Properties
- Operations
- Subclasses (inheritance)
- Intercommunication

Inheritance Advantages

Evolution

Inheritance Disadvantages

Dependencies

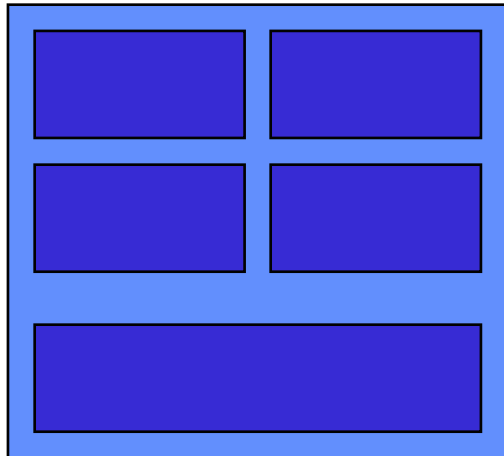


# OO approach

---

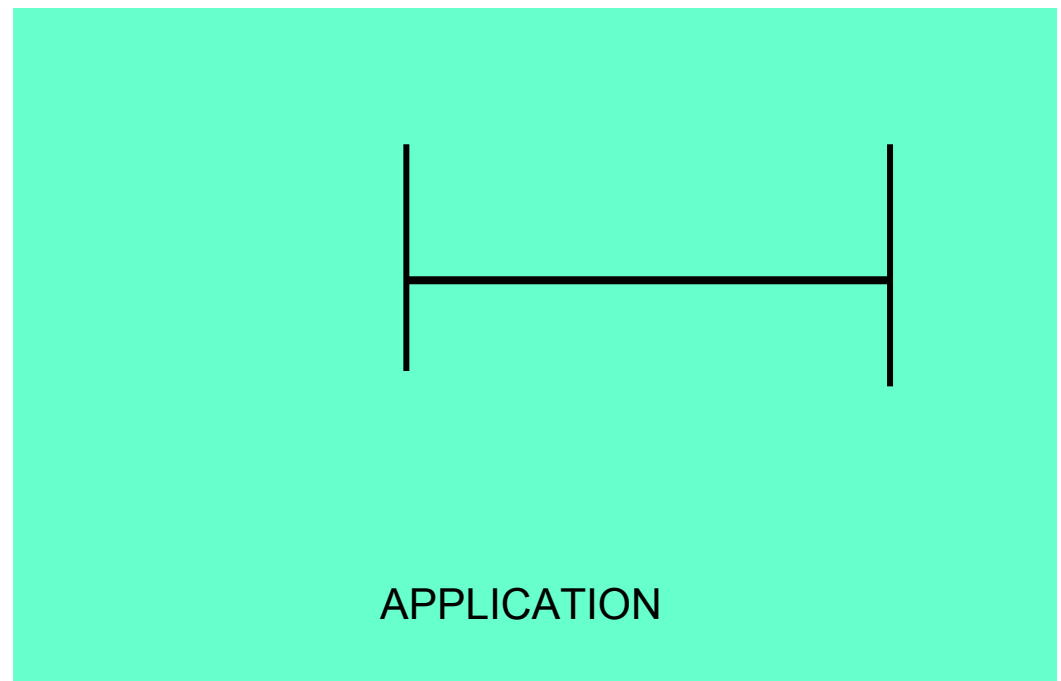
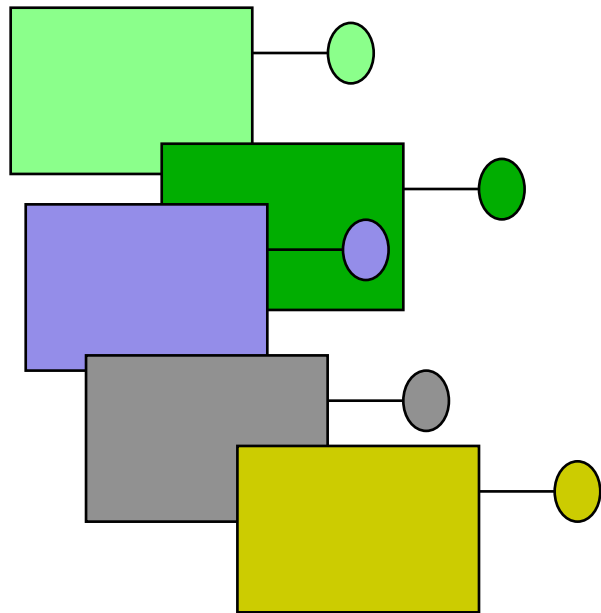
## □ Still

- You ,must recompile/relink your application if you want to change it
- There can be many relations between classes/Objects
- (Objects play different roles in different context)



# Component-based approach

- ❑ **Applications/systems are built of independent parts**
  - Parts are developed independently of the application
  - Parts may be deployed/replaced independently of the rest of the system



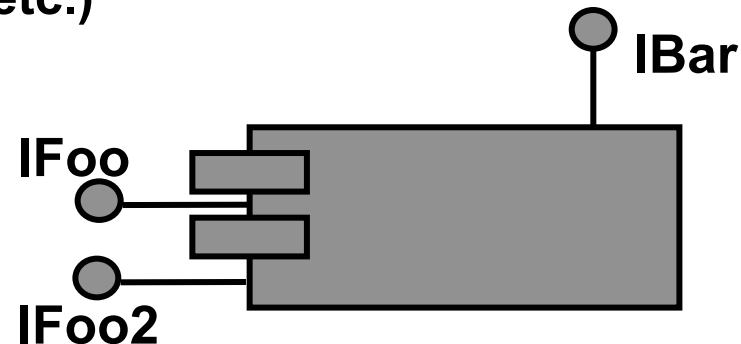
- 
- ❑ **What is a component?**  
**What is Component-based Software Engineering?**

- 
- ❑ **A component is a part of a system**
  - ❑ **A software component is a part of a (software) system**
  
  - **Recognizable part of a system at**
    - ☞ Requirement specification phase
    - ☞ Design phase
    - ☞ Implementation /code
    - ☞ Execution phase
    - ☞ Maintenance phase

# Software Component definition (Szyperski)

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 party.

- ❑ **Contract** - A specification attached to an interface that mutually binds the clients and providers of the components.
  - **Functional Aspects (API)**
  - **Pre- and post-conditions for the operations specified by API.**
  - **Non functional aspects (different constrains, environment requirements, performance, etc.)**



# Component-based software engineering

---

## □ Goals – provide support for:

- Build systems from reusable units (components)
- Build units (components) that are reusable
- Provide methods/processes/technologies for building components and for their “composition”

# Component technologies – what do they provide?

---

- Component specification – interfaces (component model)**
- Development Environment (for building components)**
- Intercommunication (middleware)**
- Deployment support**
- Set of standard components**
- .....**

# Component technologies

---

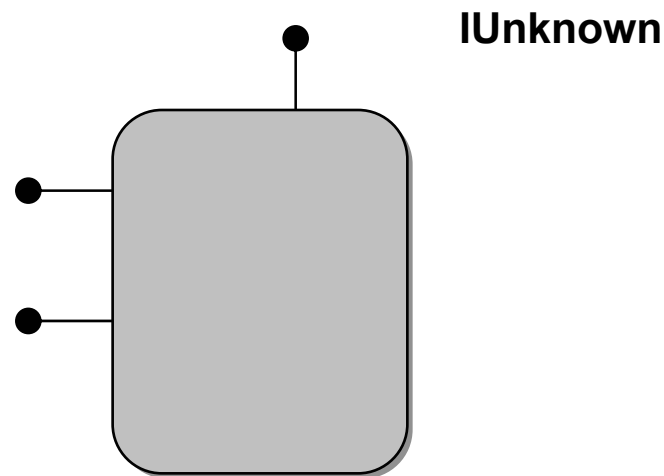
**Examples: COM/DCOM, COM+  
.NET  
JavaBeans, EJB  
CORBA component model**

**Other component models**

# Example 1: COM Component Object Model

---

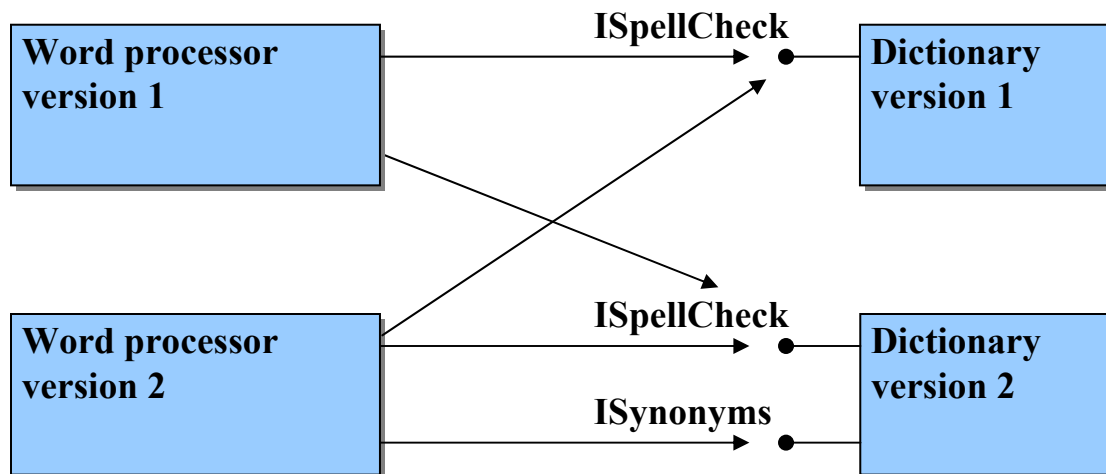
- ❑ It defines a binary standard of the interface
  - Language independent
- ❑ All Access is through the interface
  - Supports multiple interfaces



# COM Interface

---

- ❑ Multiple interfaces
- ❑ A new interface for new functionality
- ❑ Example:



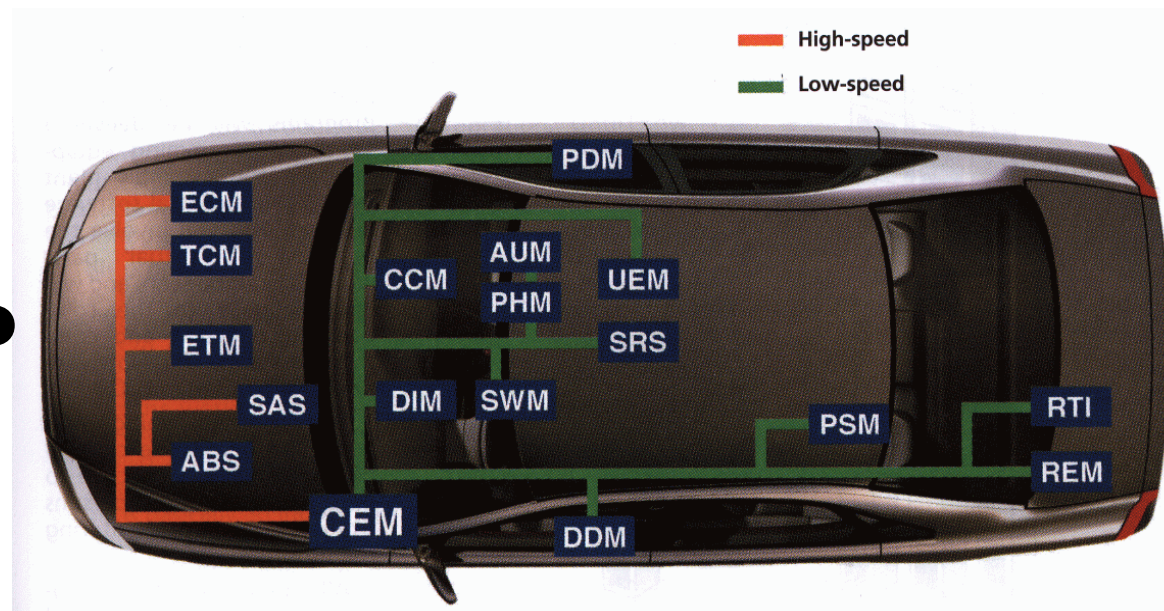
# Component models for Distributed Real-time System

## Example:

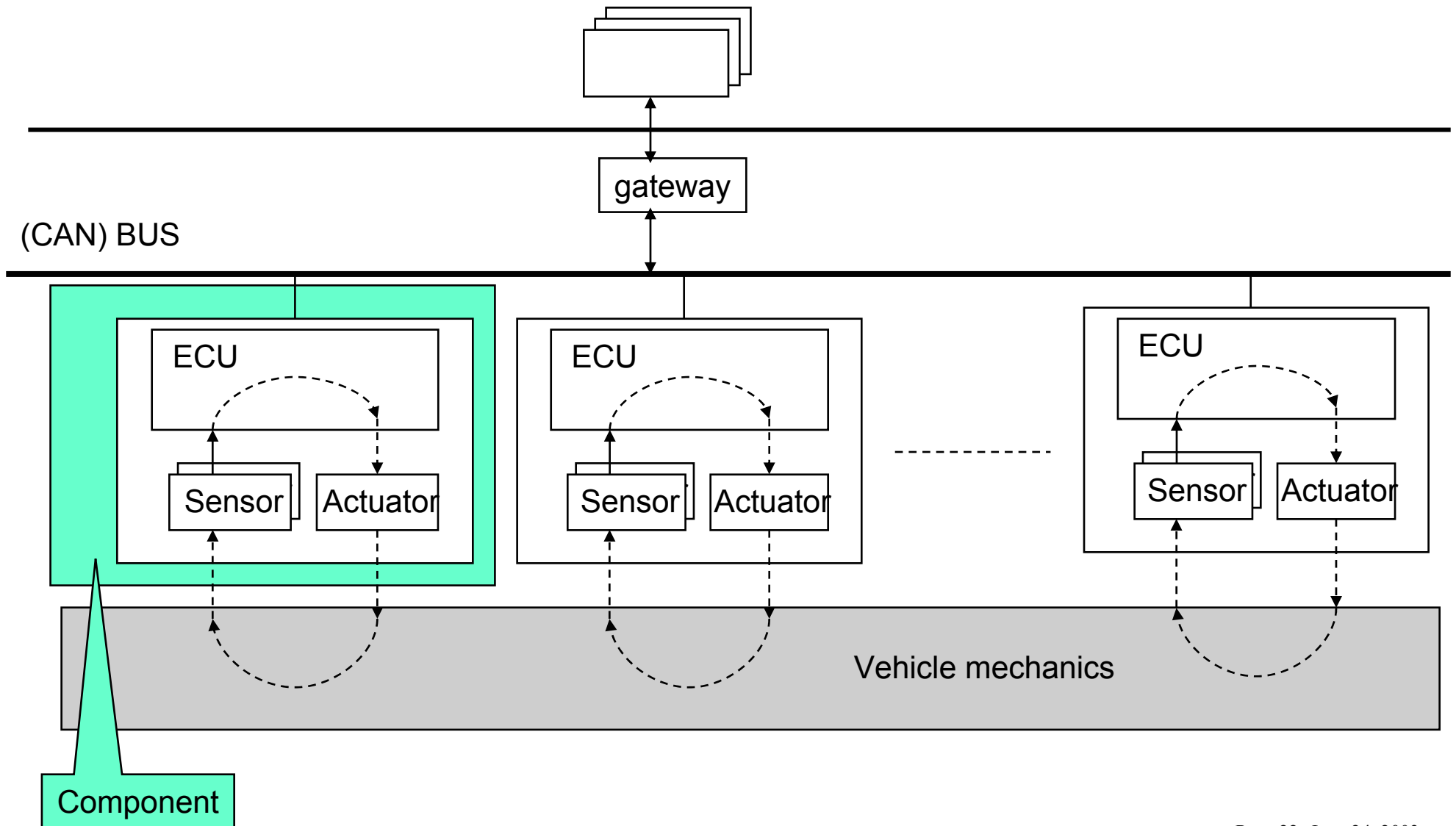


- Networks instead of cables
- Electronic control units replace electromechanical components

**Volvo  
S80**



# The architectural design

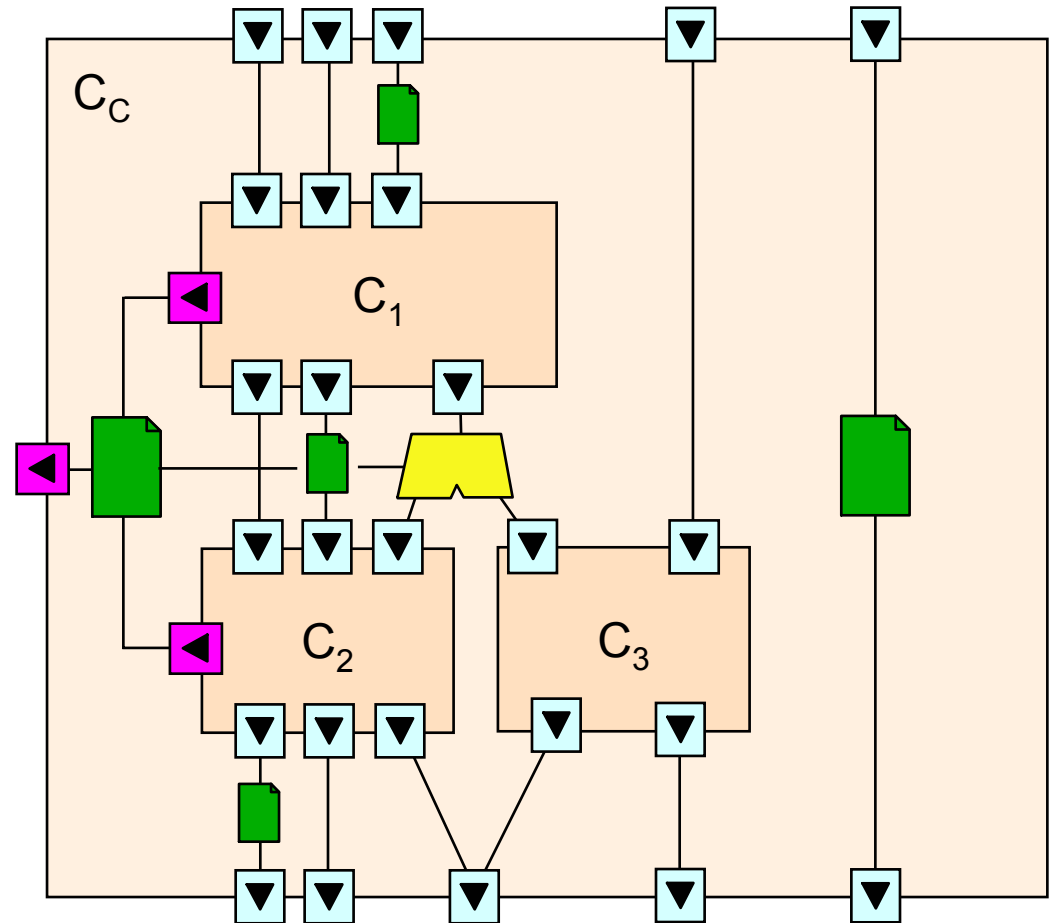


# The Philips Koala Component Model

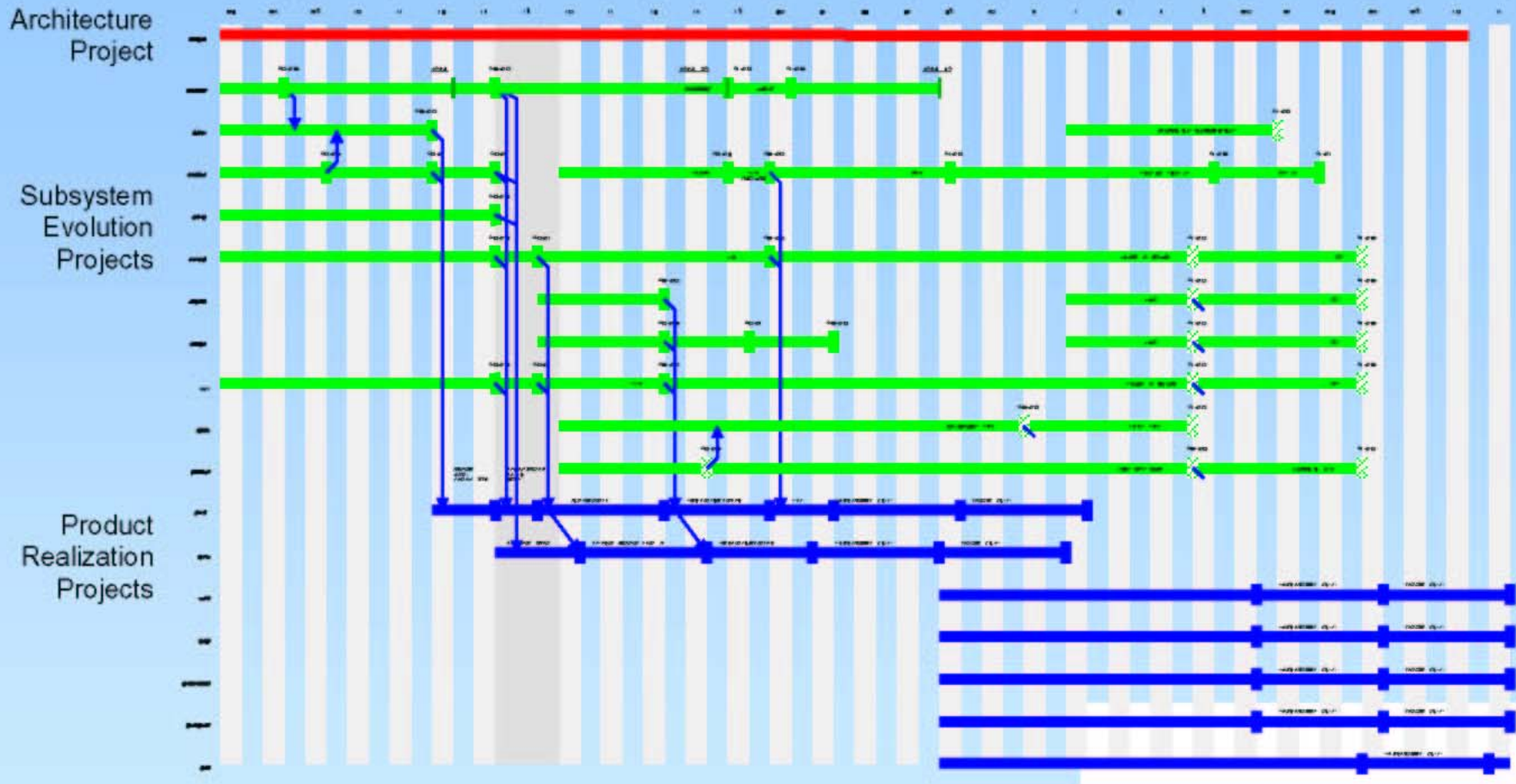
Koala is:

- a Software Component Model
- with an ADL

Used for TV sets product family



Product and subsystem releases are carefully planned:



# The main benefits of component-based approach

---

## □ Support:

- Reuse
- Shorter development/production time
- Better quality
- Decreased development costs
- Encouraging business



**What are the challenges???**

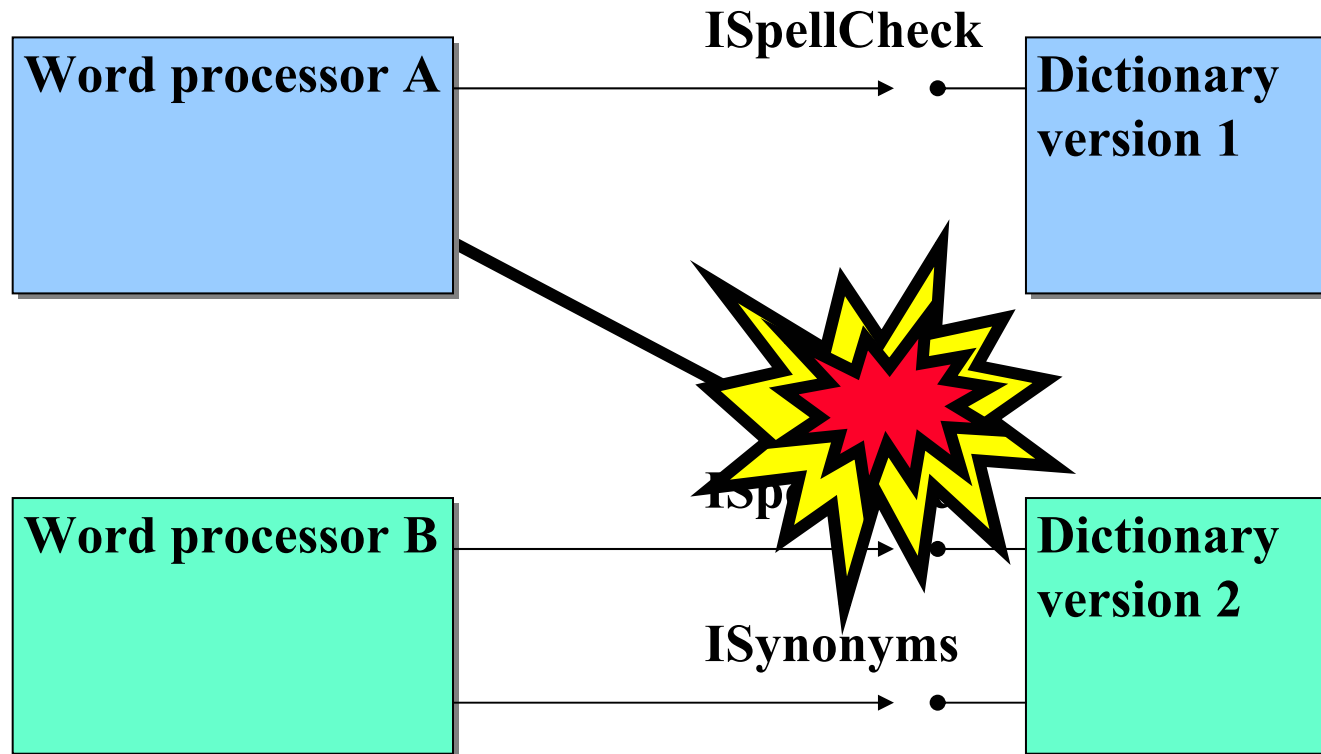
---

## **Problems & Challenges**

- Technical nature
- Business and marketing nature
- Organizational nature
- Legislative nature

# Challenge – Example 1

---



**Interface mismatch (architectural mismatch)  
DLL Hell**

## Challenge 2: Implementation of complex functions

---

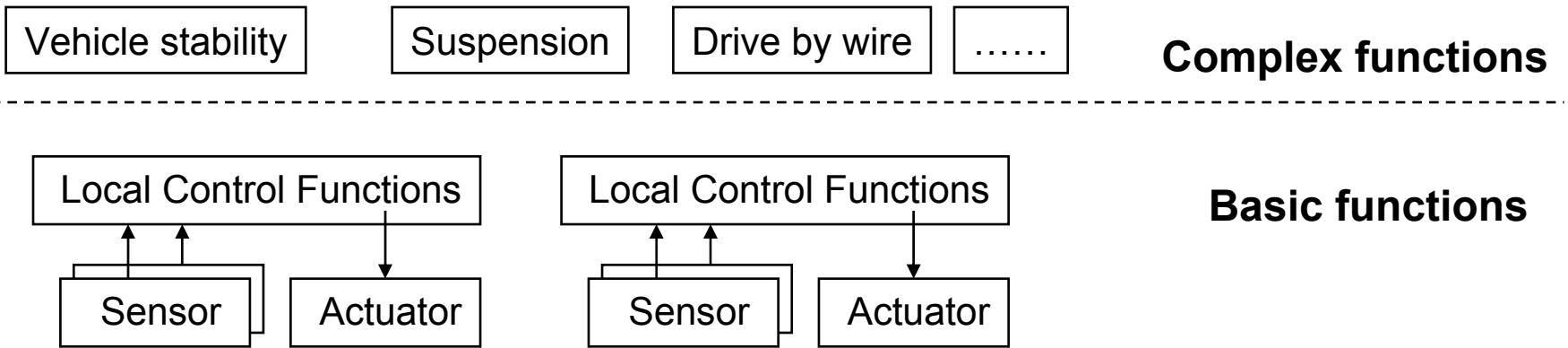


**Volvo  
S80**

**How to open a back door?**

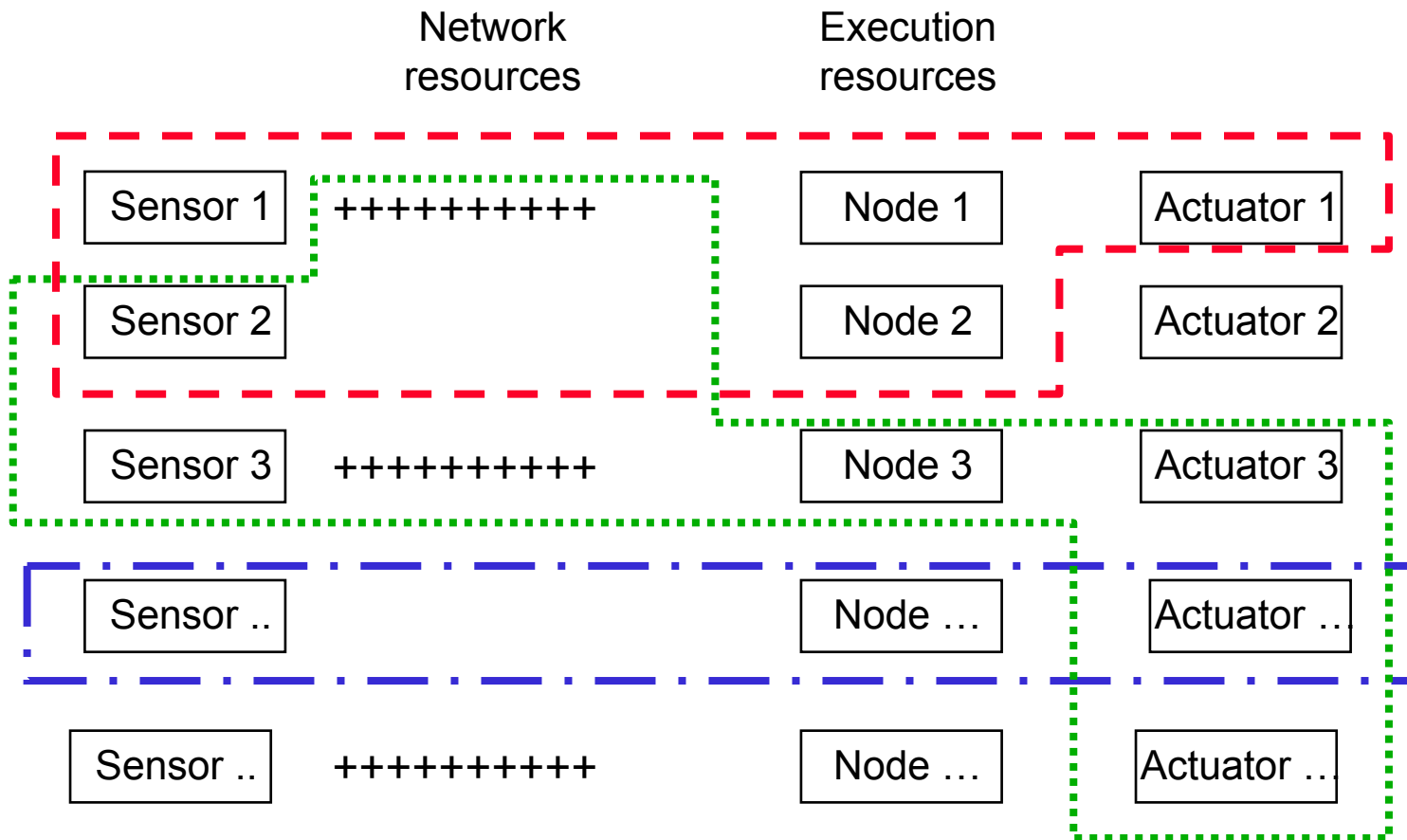
# The architectural design challenge

---



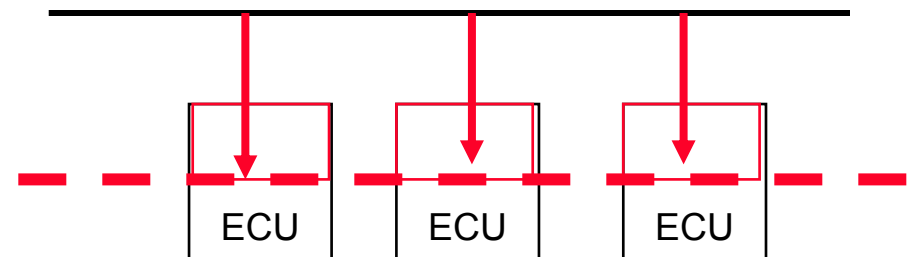
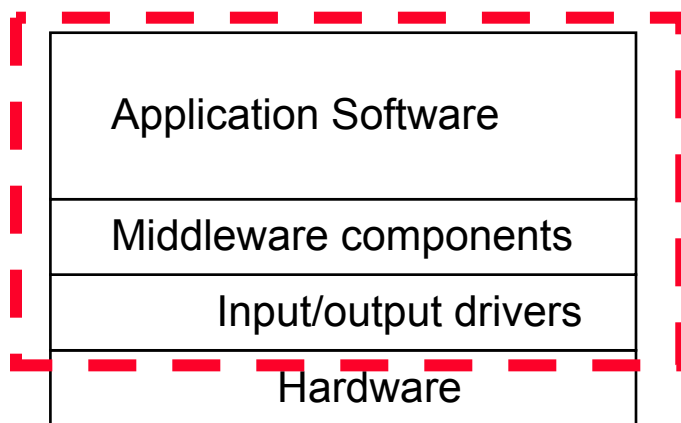
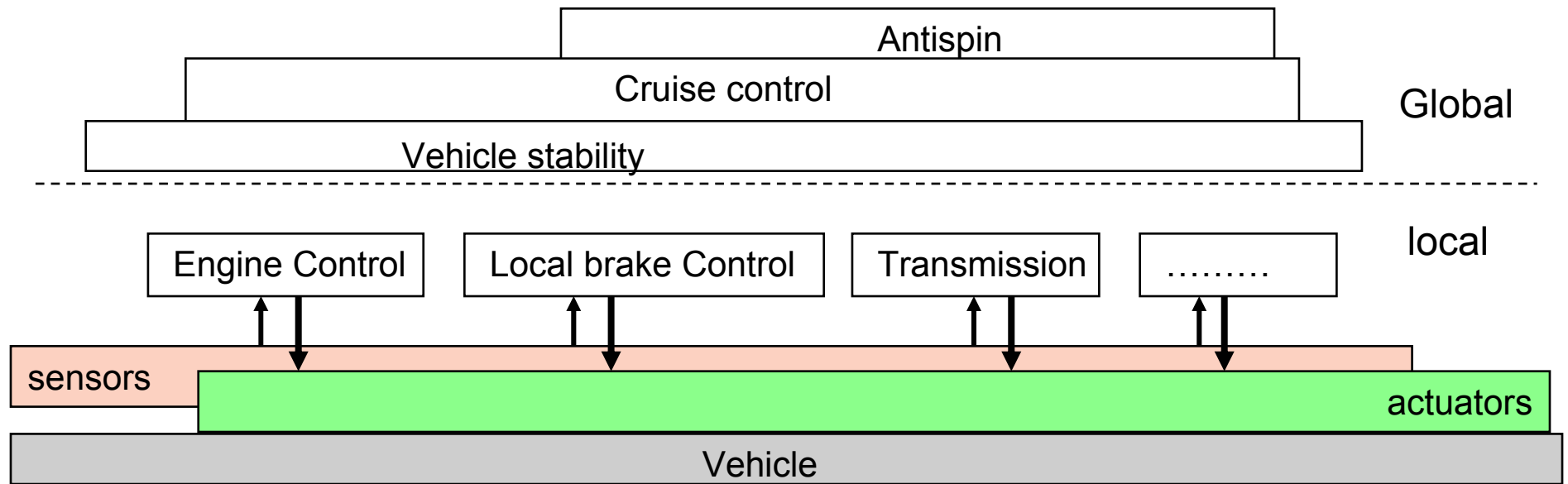
**How to implement complex functions base on local control functions?**

# Problem: resource sharing



**Can functions of different criticality be allowed to share resources?**

# Challenge – open and dependable platform



# Component specification challenges

---

**Components are not only functional interfaces**

**There exists a number of other component attributes**

- Called:
- Non-functional attributes
- Extra-functional Attributes
- Quality of services

# Some extra functional properties...

---

## References:

- ❑ Kazman, R., L. Bass, G. Abowd, M. Webb, "SAAM: A method for analyzing properties of software architectures," Proceedings of the 16th International Conference on Software Engineering, 1994.
- ❑ Kazman et al, Toward Deriving Software Architectures from Quality Attributes, Technical Report CMU/SEI-94-TR-10, 1994.
- ❑ McCall J., Richards P., Walters G., Factors in Software Quality, Vols I,II,III', US Rome Air Development Center Reports, 1977.
- ❑ Bosch, J., P. Molin, "Software Architecture Design: Evaluation and Transformation," Proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999.

Accuracy; Audibility; Availability; Completeness; Conciseness; Consistency; Correctness; Ease of creation; Efficiency; Error (fault) tolerance; Execution; efficiency; Expandability; Flexibility; Generality; Hardware independence; Integrability; Integrity; Interoperability; Maintainability; Modifiability; Modularity; Operability; Performance; Portability; Reliability; Reliability; Reusability; Scalability; Security; Simplicity; Software system independence; Testability; Traceability; Usability;

# Example: Dependability

---

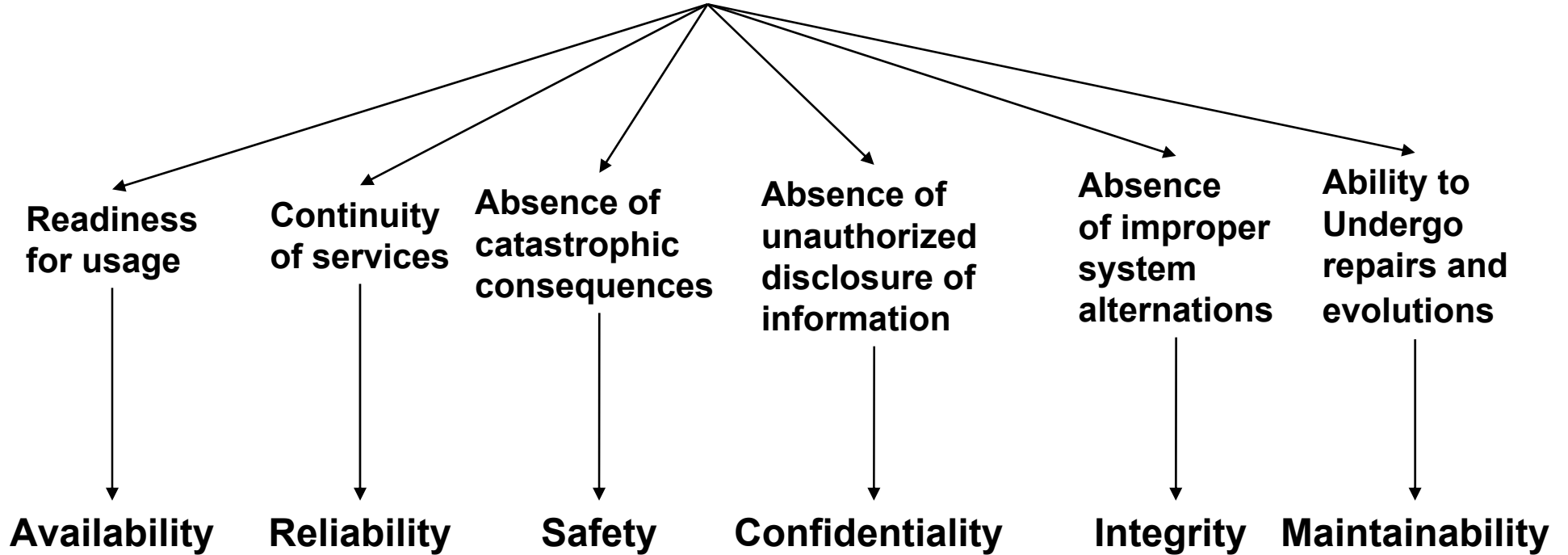
1. **Ability of a system to deliver service that can justifiably be trusted**
2. **Ability of a system to avoid failures that are more frequent or more severe than is acceptable to user(s)**

## Related to

- Trustworthiness (assurance that a system will perform as expected)**

---

## Dependability



# Example...

---

## □ Safety-critical systems

- Safety - “Absence of catastrophic consequences on the users and the environment”
- Property depends on
  - ☞ Environment
  - ☞ Use of the system
  - ☞ Consequently, software is only hazardous in a context
  - ☞ **It is a system behavior**
  - ☞ **THERE ARE NO SAFETY-CRITICAL COMPONENTS**
  - ☞ **(Safety is a system emerging property)**

# Questions – dependability & Components?

---

## DEPENDABILITY ATTRIBUTES

=====

<b>Property</b>	<b>System</b>	<b>Component</b>
<b>Availability</b>	<b>Yes</b>	<b>No (???)</b>
<b>Reliability</b>	<b>Yes</b>	<b>Yes</b>
<b>Safety</b>	<b>Yes</b>	<b>No</b>
<b>Confidentiality</b>	<b>Yes</b>	<b>??</b>
<b>Integrity</b>	<b>Yes</b>	<b>No</b>
<b>Maintainability</b>	<b>Yes</b>	<b>Yes</b>

**However no system property can be derived directly from component property**

# The question

---

**If we want to have a dependable systems  
(reliable, available,...)**

- Which properties should the components have?
- How should we specify/ measure/ verify these properties for the components?

# Challenges - predictability

---

- Given the system extra-functional properties, which properties of involved components are required?**
- Given a set of properties of components, which system properties we can predict?**
- To which, and to which extent, and under which constraints the emerging properties (i.e. the system properties non-existing on the component level) are determined by the component properties?**
- How to achieve predictable system behavior (i.e. predictable extra-functional properties) from components which extra-functional properties are determined to certain accuracy.**

# More questions

---

- Extra-functional properties:**
  - Which properties are emerging system properties
  - Which are properties of both systems and components?
- How these properties in a component-based system are related to component properties?**
- To which extent (and how) these properties can be determined from component properties?**
- To which extent can uncertainty in predictability of these properties be minimized and how much is that related to the uncertainty of the component properties?**
- In which phase of the development process are these properties addressed mostly?**

# Conclusion 1: Predictability

---

## □ Niels Bohr

- Prediction is difficult, especially if it is about the future

# CBSE research and the SW life-cycle

