Introduction to Software Engineering

Software design

Philippe Lalanda

Philippe.lalanda@imag.fr

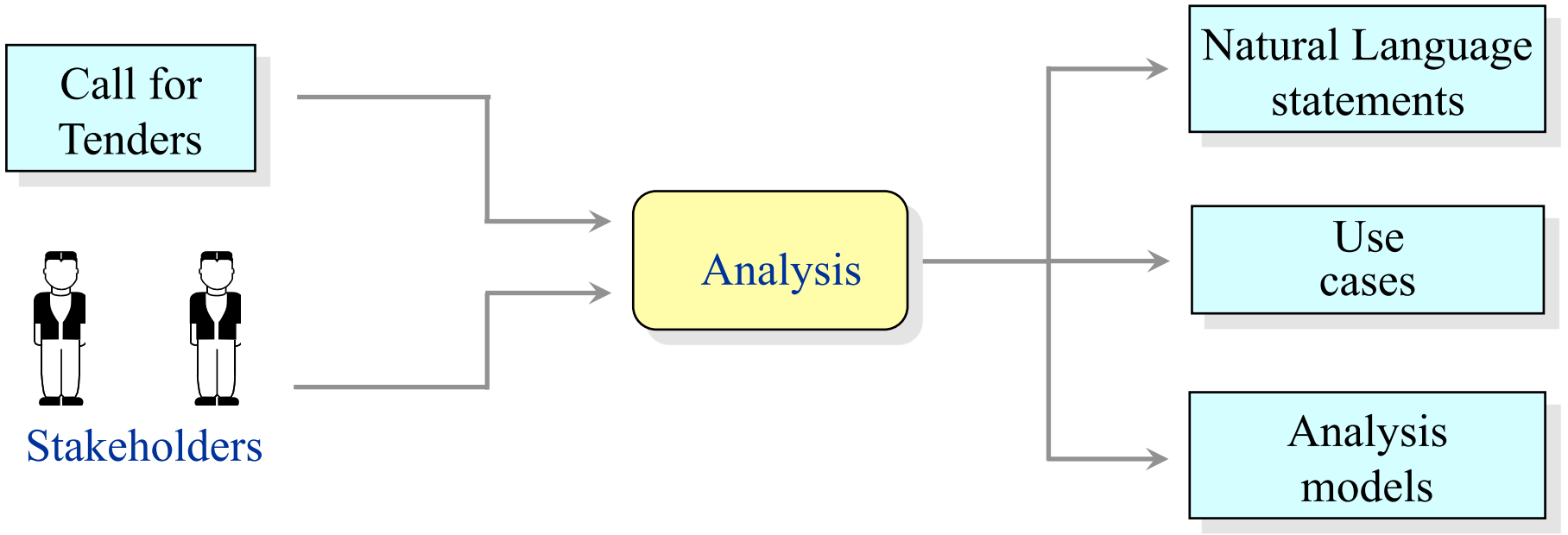
http://membres-liglab.imag.fr/lalanda/

Outline

- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

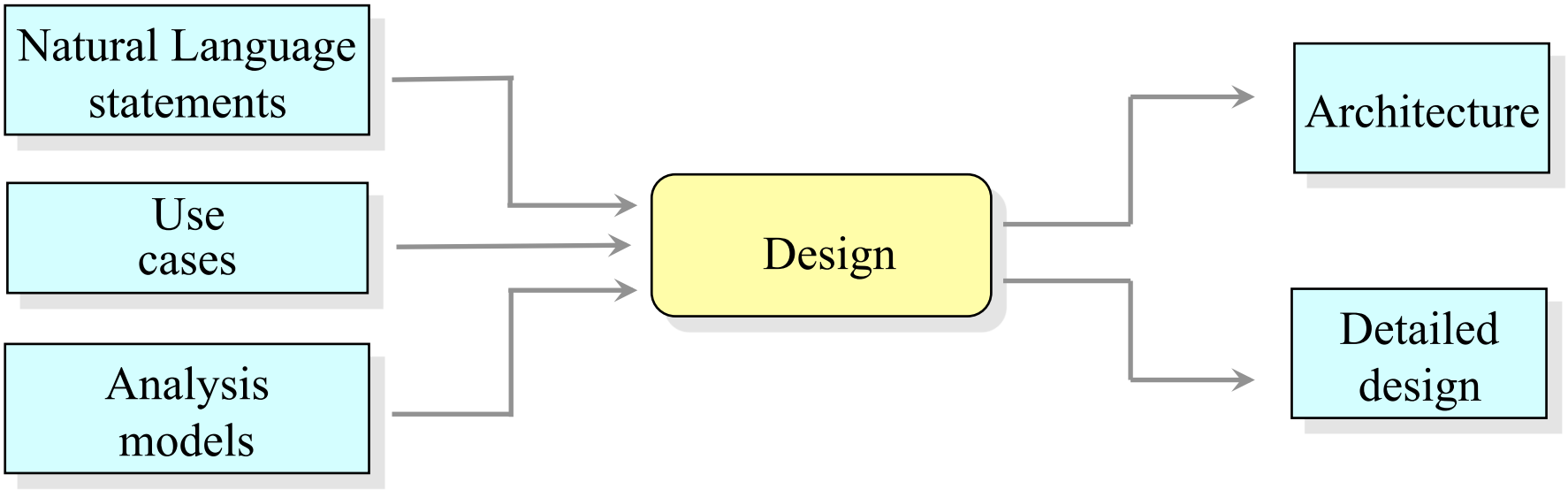
Reminder - requirements

- Requirements express what customers need/want
  - Goals, functions, qualities, constraints



Design

- A creative process transforming the problem into a solution
  - *i.e.* define a logical organization for the code

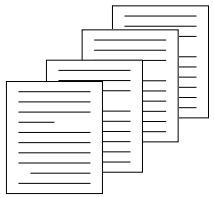


Issue

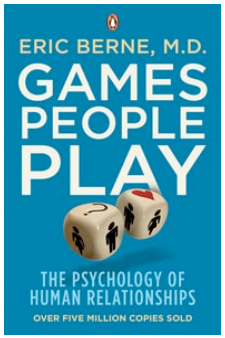
- There is a gap between analysis and design
  - Requirement phase: identify stakeholders, understand goals and needs, identify and resolve conflicts, …
  - Design phase: define components (or procedures, classes, …), connections, configurations, …
- Non cascading projects make this problem even harder
  - Design has to be integrated in an iterative approach

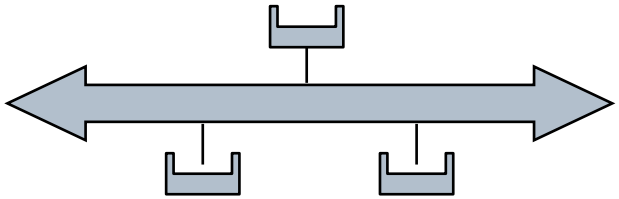
Context

- No such thing as “pure design”
- Design is done in a context
  - Enterprise rules
  - COTS to be used
  - Politics ...



Enterprise rules

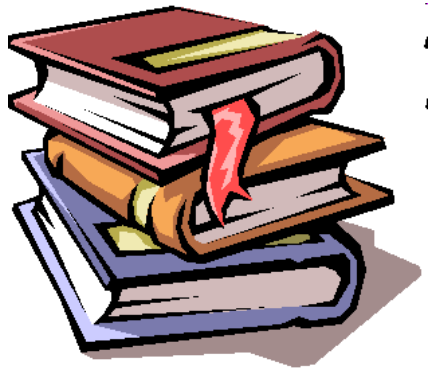




COTS, tools, …

Enterprise rules

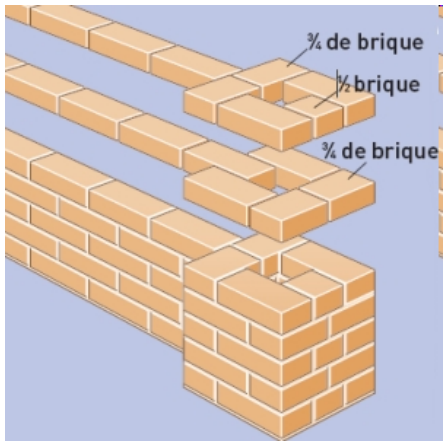
- In order to better manage software developments, most companies define their own good practices
  - This includes design directives
- Examples
  - Architectural styles
  - Design patterns
  - Technologies to be used
  - Consultants to be employed



COTS and tools

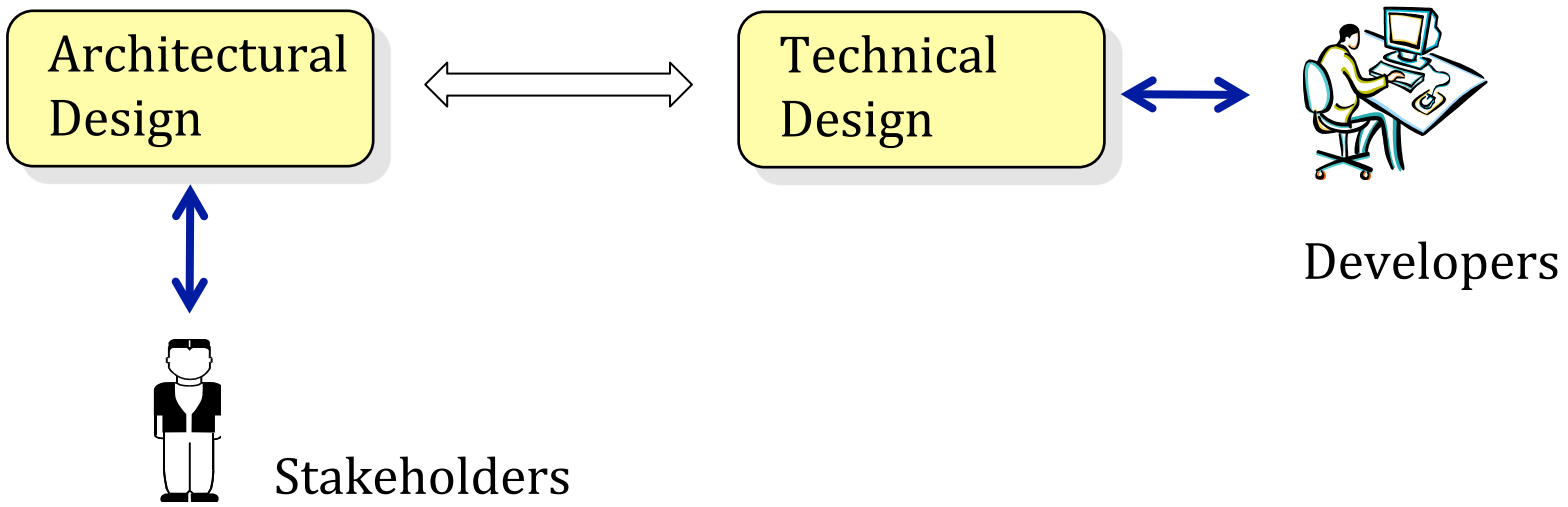
- Component Off The Shelves
  - Favor reuse
  - Avoid “Not Invented Here” syndrome
  - But uncontrolled lifecycle

- *Mega-programming* is a reality today
  - DBMS
  - Middleware
  - Browsers
  - HMI framework



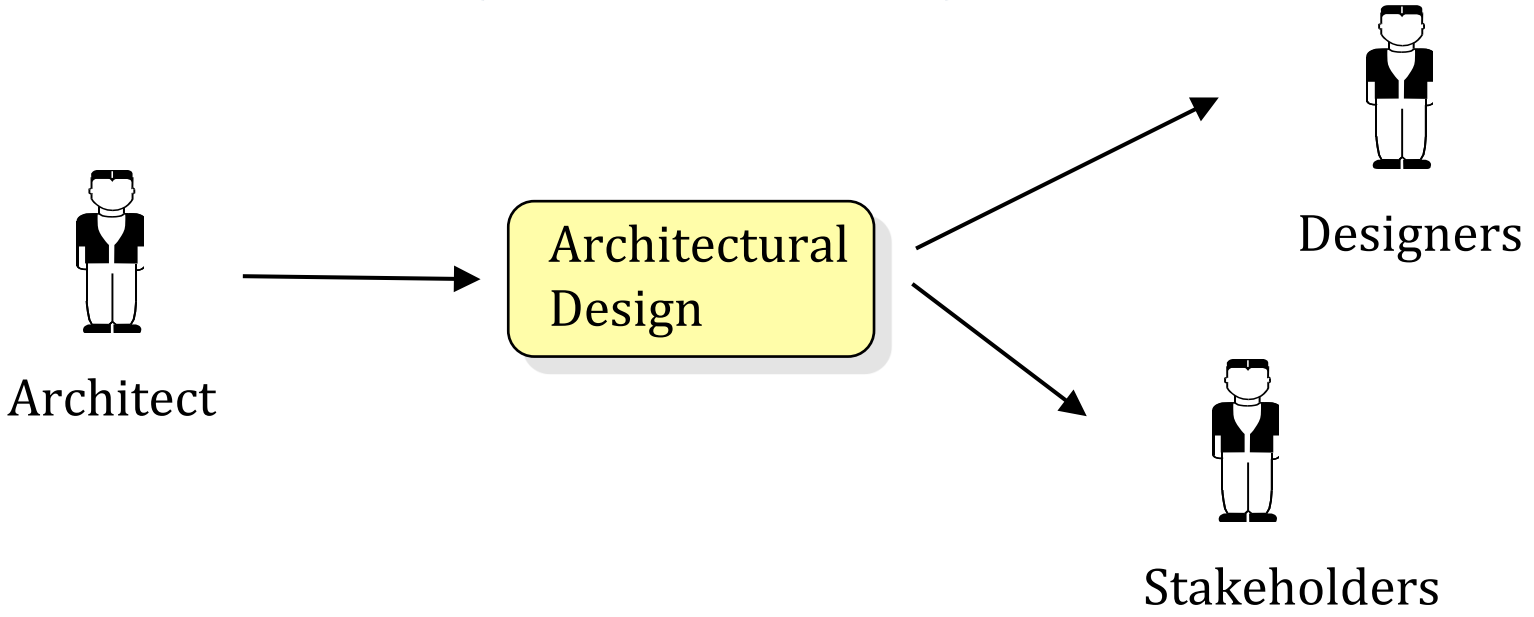
Architectural and technical design

- Design is a two-step iterative process
  - Architectural design: high-level description of the system
  - Technical design: much more detailed description



Architectural design

- Absolutely necessary
  - To discuss solutions between designers
  - To convince stakeholders (and get an agreement)
  - Must remain (and be updated)



Technical design

- Important
  - To specify to developers what they have to do
  - Very hard to write and maintain
    - Should it be complete?
    - Should it be updated?

Outline

- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

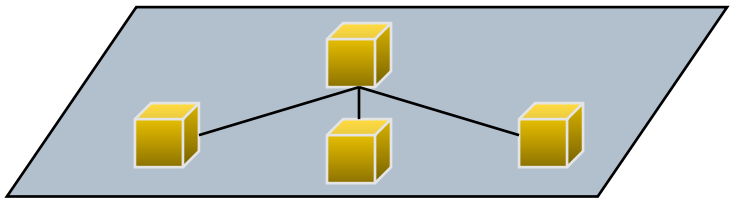
Models

Designing is modelling

What is a model ?

- A model is a simplified, biased representation of a (software) system

M_1
(model)



Is represented by

M_0
(software under design)

Files HMI Computer

What is a model ?

- A model is simpler and cheaper that reality
  - Less concepts
  - Less relationships
  - Focalization

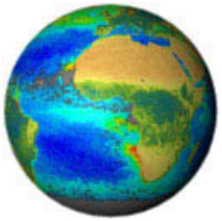
- A model is built for a specific purpose
  - Models are seldom reusable

Models are biased

- Models allow to better reason and communicate on some aspects

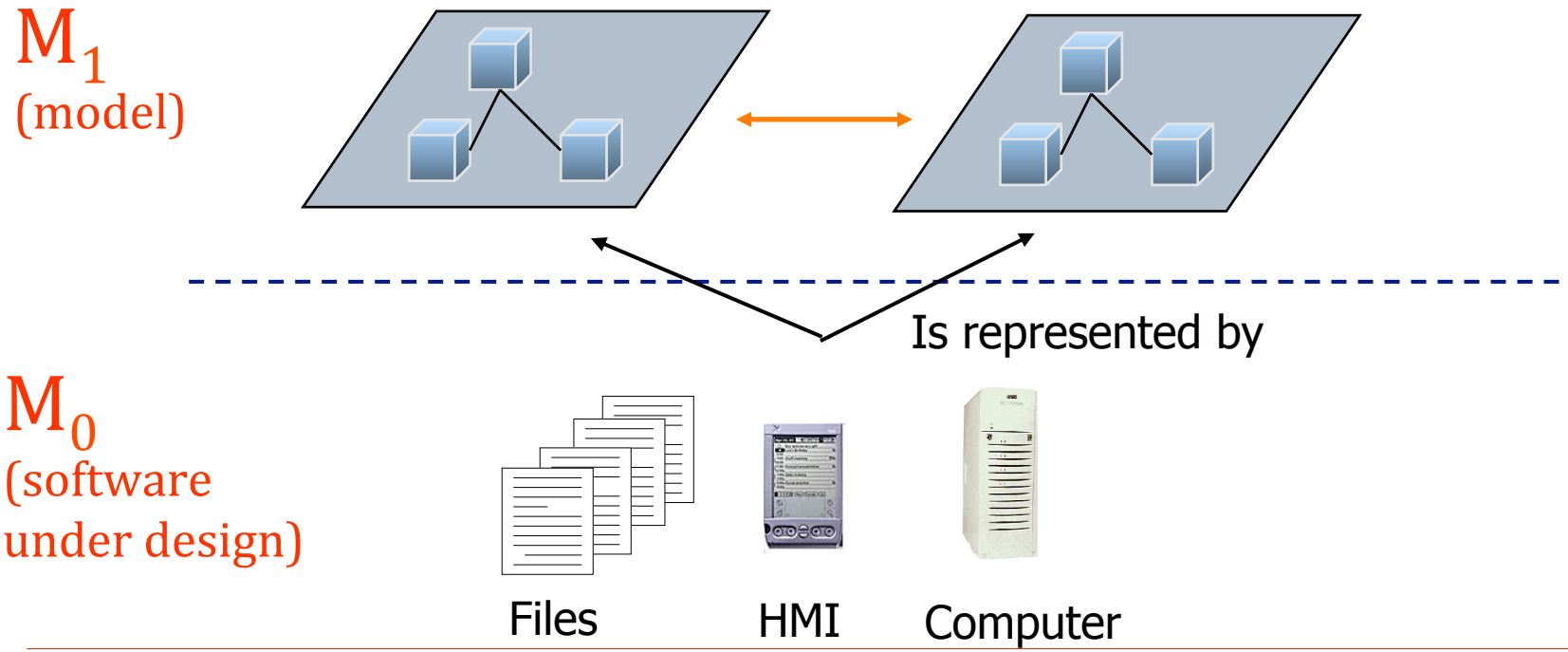
- But
  - Not on every aspects
  - Not completely (some details are missing)

Is represented by



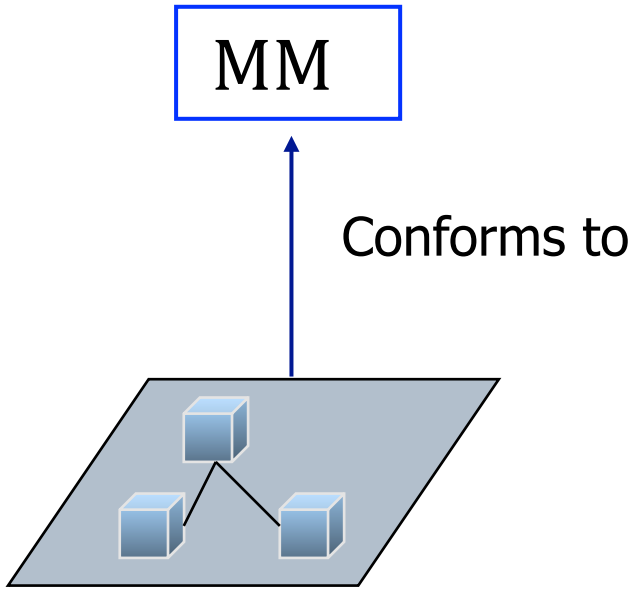
Many models

- Several models can be used for a given system
  - Separation of concern

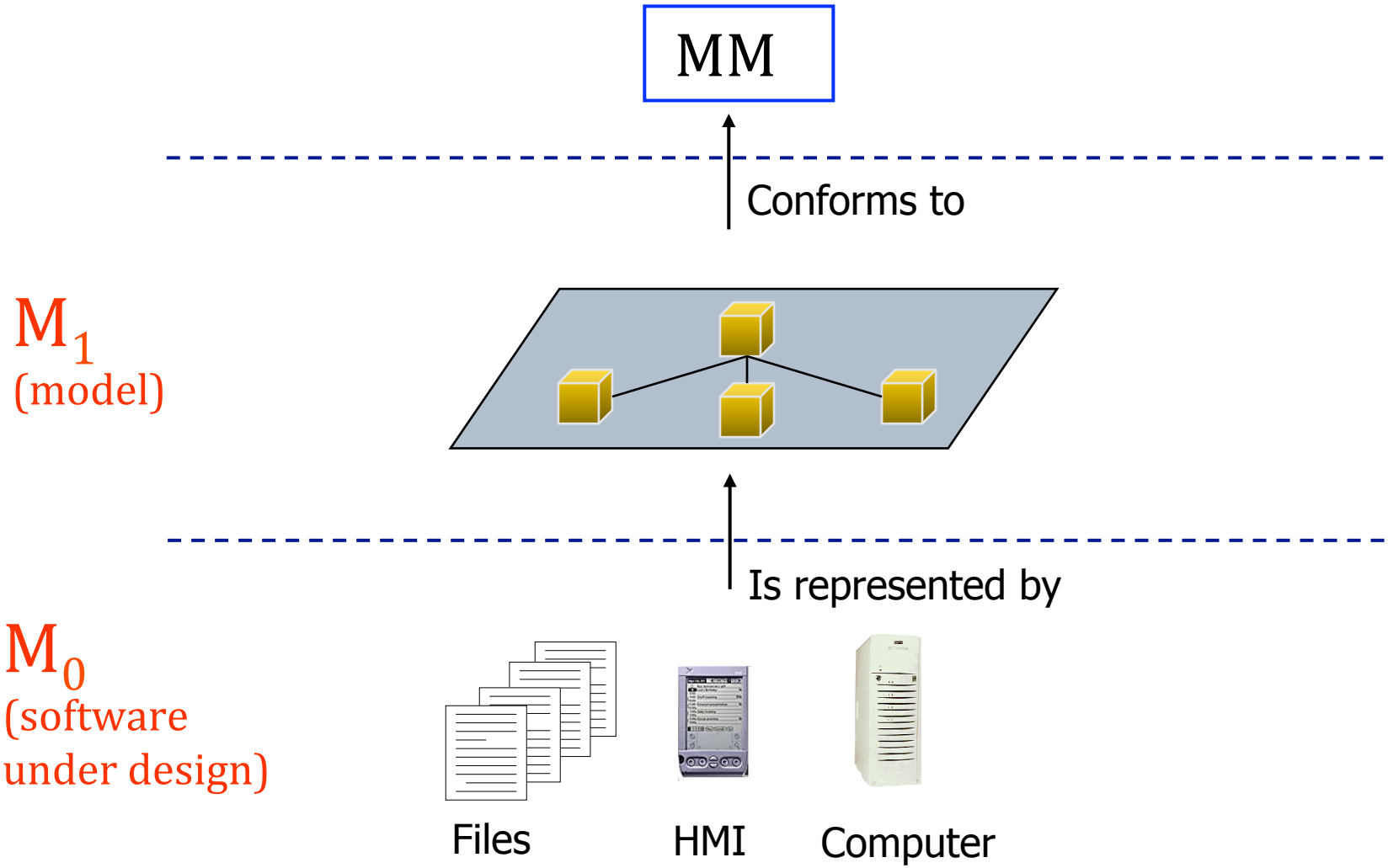


Meta models

- A meta model defines a language to define models
  - Simple form of ontology
- A meta-model defines
  - A vocabulary
  - A grammar
- UML provides a language to build models

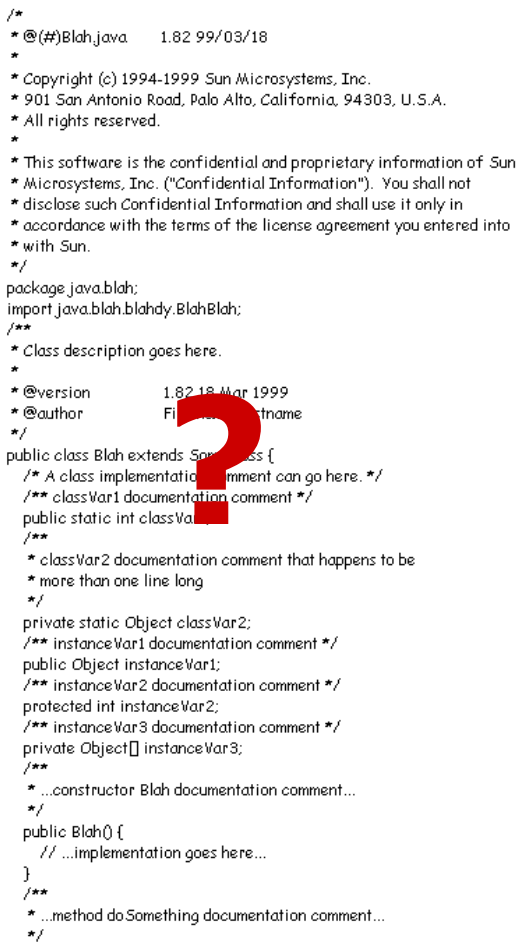


Big picture


MM
Conforms to
M1
(model)
Is represented by
M0
(software
under design)
Files
HMI
Computer

Design models - 1

- Usage
  - reasoning
  - communication
  - code generation ?
- Provided languages
  - UML
  - DFD, ...

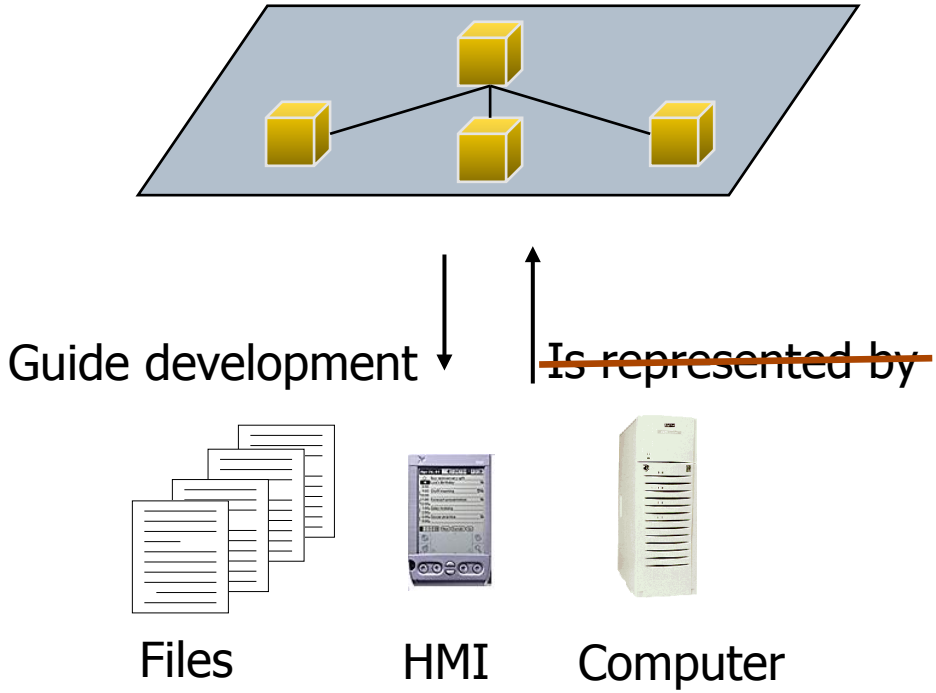


```
/*
 * @(#)Blah.java      1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 */
package java.blah;
import java.blah.blahdy.BlahBlah;
/**
 * Class description goes here.
 *
 * @version          1.82 18 Mar 1999
 * @author           Fi          tname
 */
public class Blah extends So          ss {
    /* A class implementatio          mment can go here. */
    /** classVar1 documentation comment */
    public static int classVa
    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;
    /** instanceVar1 documentation comment */
    public Object instanceVar1;
    /** instanceVar2 documentation comment */
    protected int instanceVar2;
    /** instanceVar3 documentation comment */
    private Object[] instanceVar3;
    /**
     * ...constructor Blah documentation comment...
     */
    public Blah() {
        // ...implementation goes here...
    }
    /**
     * ...method doSomething documentation comment...
     */
```

Design models - 2

- Most of the time, models and reality (code) diverge
  - Keeping track is hard
  - Tools are needed
  - A lot of rigor too ..

- Programmers have the last word !



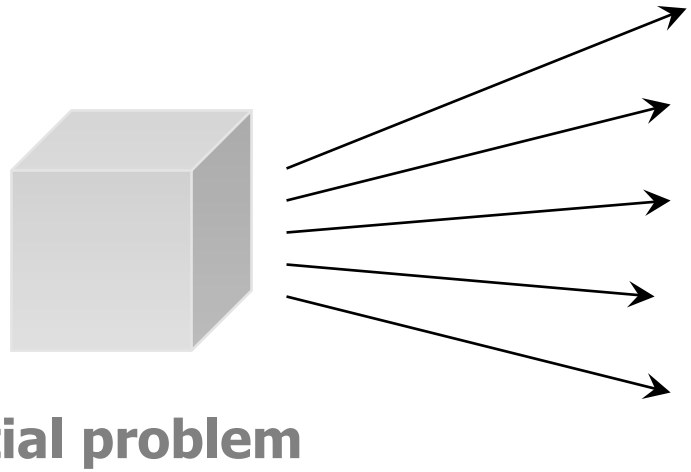
Outline

- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

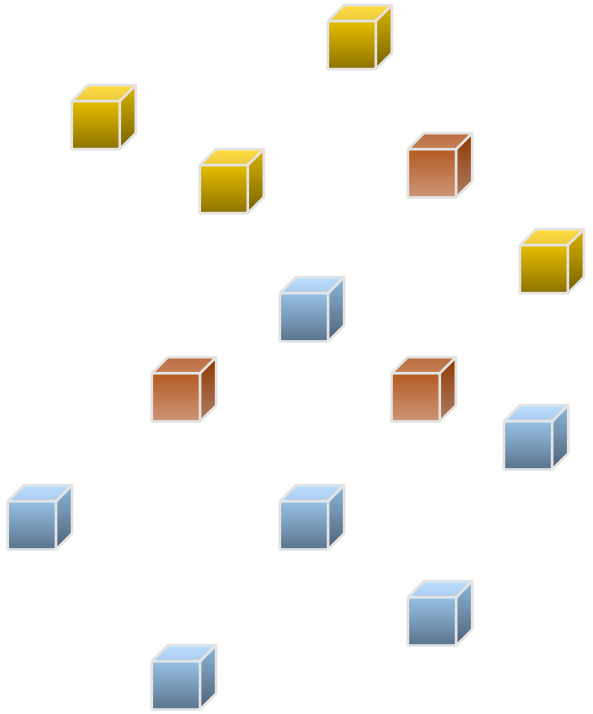
Design principles

- Decomposition
  - Modularity
  - Abstraction
- Coupling
- Cohesion
- Encapsulation
- Separation of concerns
- Simplicity
- Technology agnostic

Modularization (« Divide and conquer »)

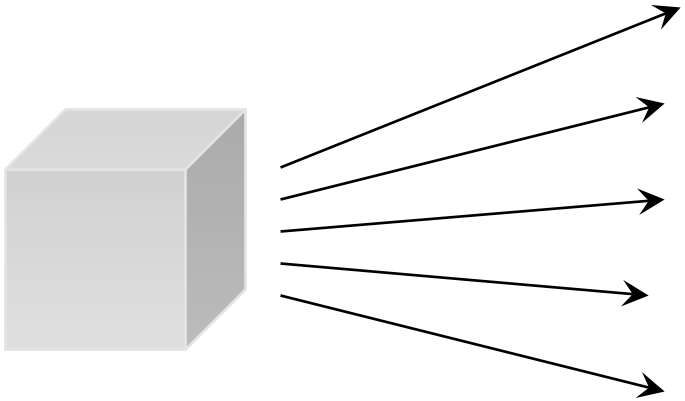


Initial problem

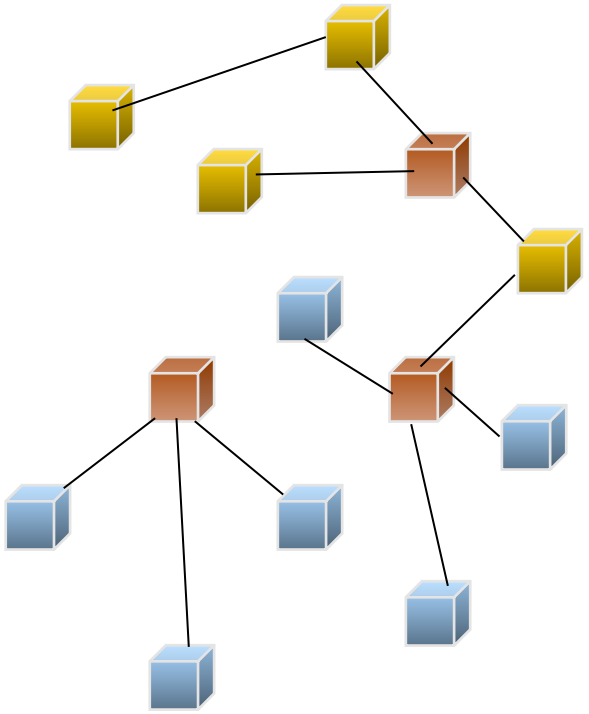


Sub problems

Coupling

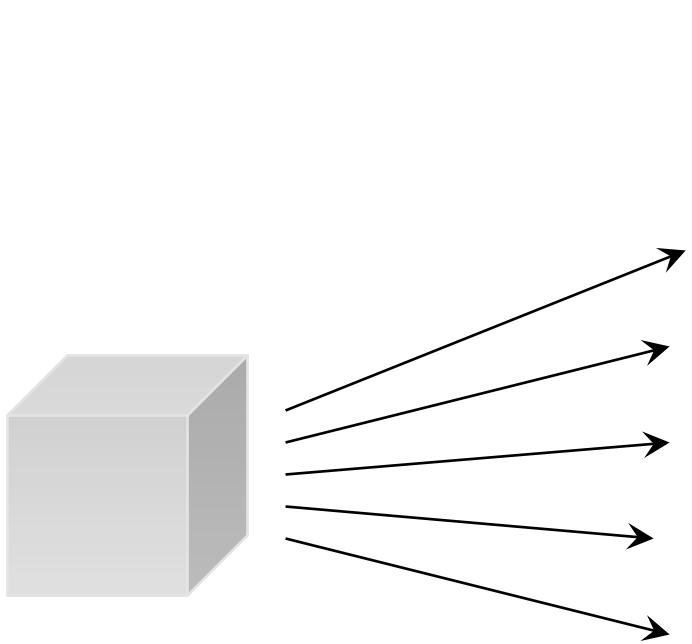


Initial problem

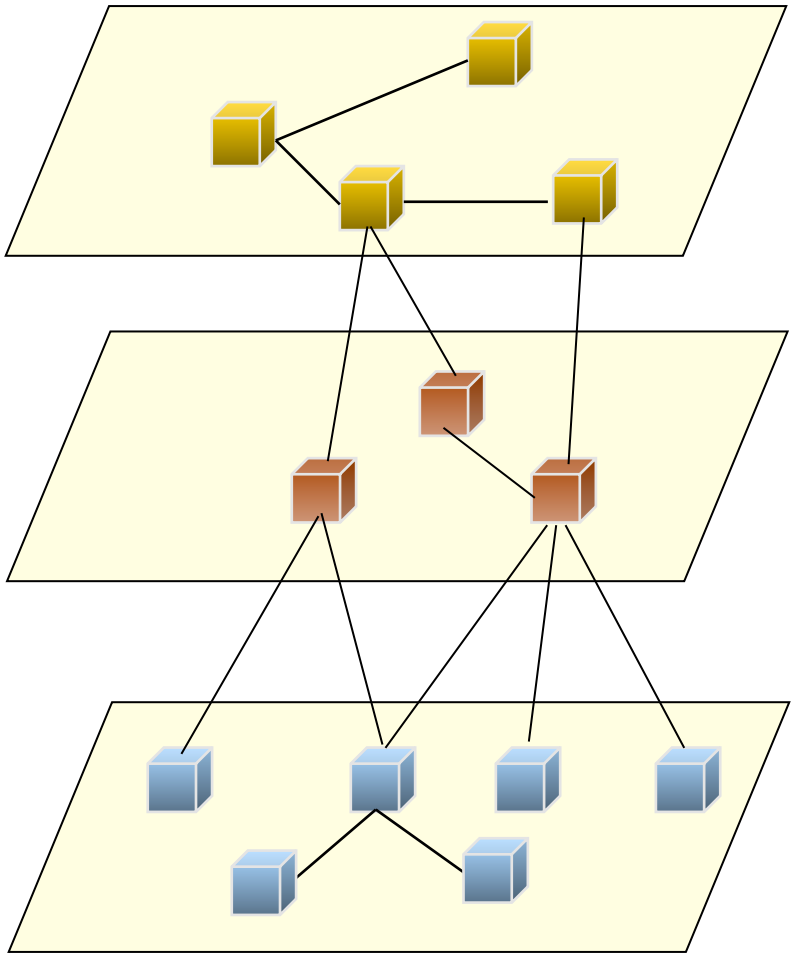


Sub problems

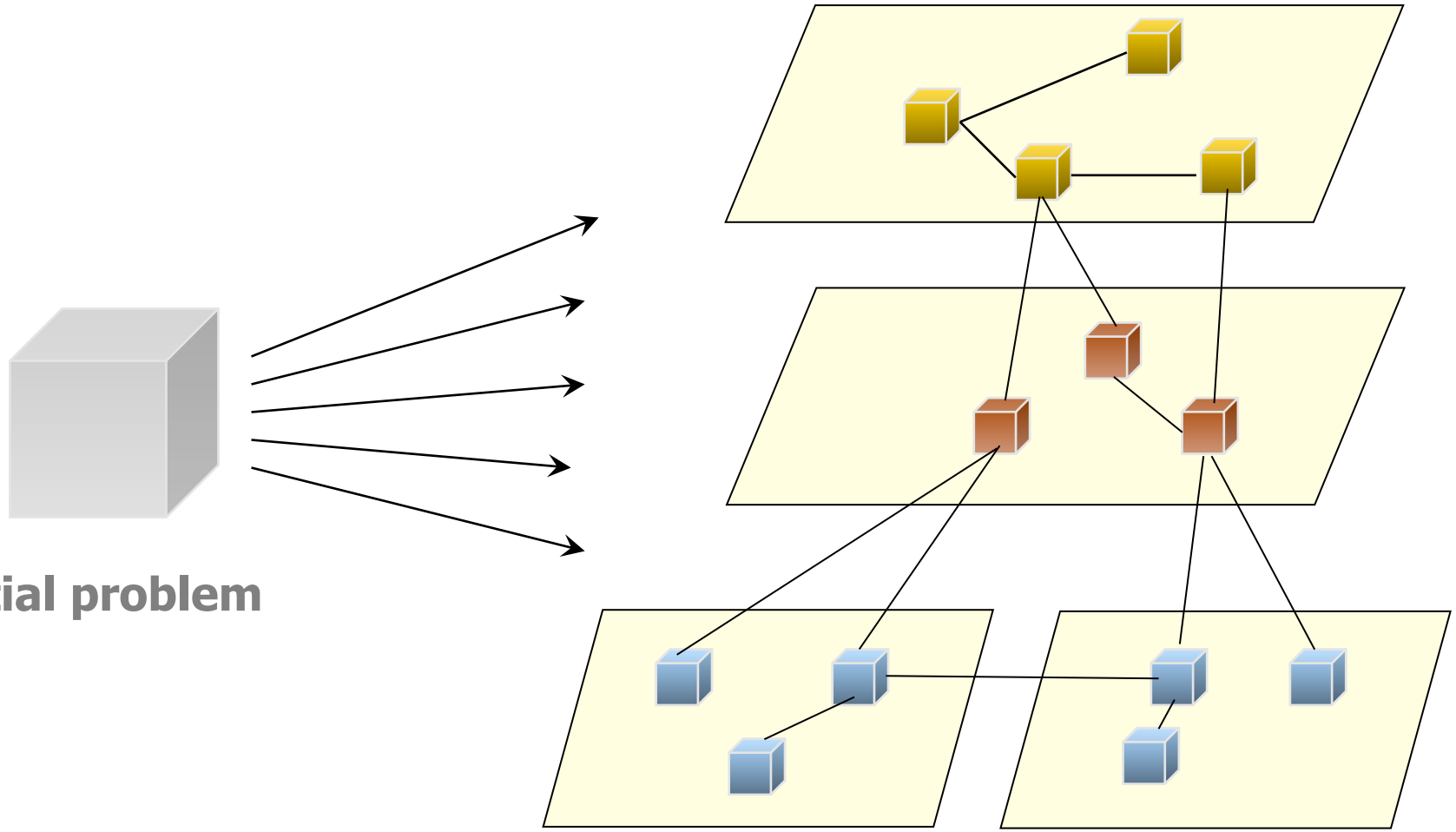
Abstraction



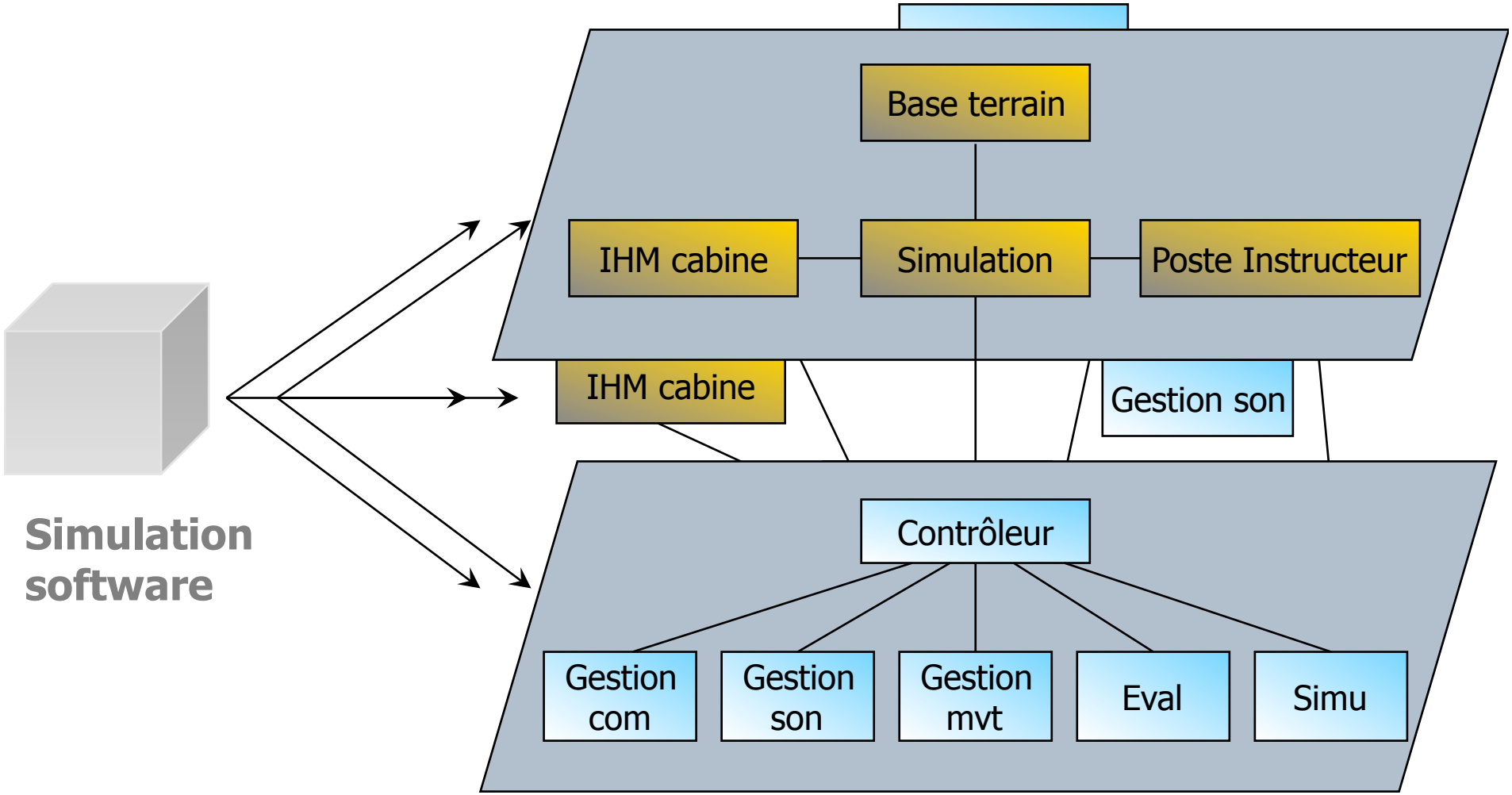
Initial problem



Separation of concerns



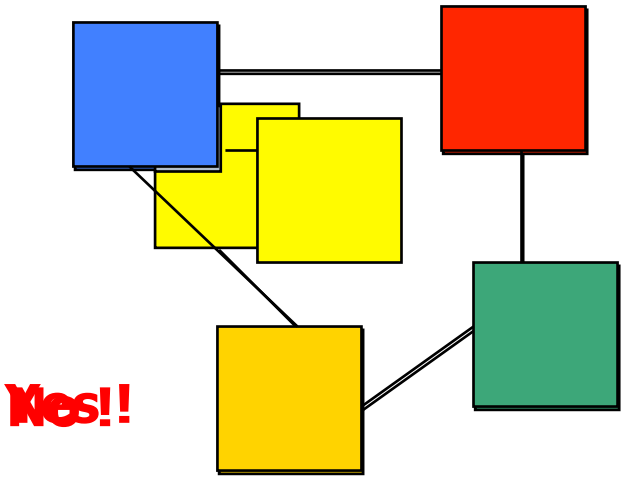
Example


Base terrain
IHM cabine
Simulation
Poste Instructeur
IHM cabine
Gestion son
Contrôleur
Gestion com
Gestion son
Gestion mvt
Eval
Simu
Simulation software

Modularity

- Various kinds of modules have been proposed
  - Data structures
  - Functions
  - Classes
  - Components

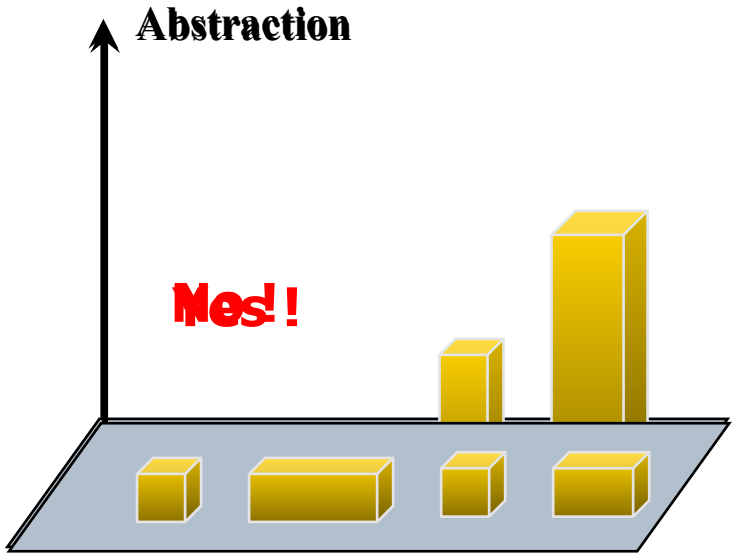
- Modules have to be
  - Solution oriented
  - Comprehensible
  - Homogeneous
  - Non overlapping



Abstraction

- The goal of abstraction is to work at a given level of generalization
  - Limiting the number of concepts
  - Ignoring details

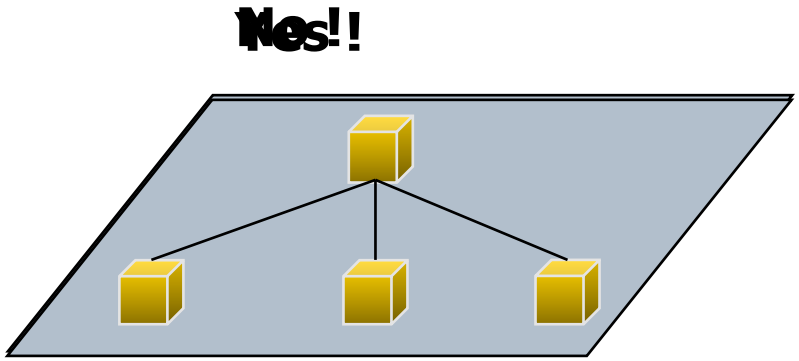
- An abstraction level
  - Solution oriented
  - Comprehensible
  - Homogeneous
  - Complete



Coupling

- Various kinds of relations depending on methods and abstraction levels
  - Function calls
  - Methods calls (and inheritance)
  - Dependencies

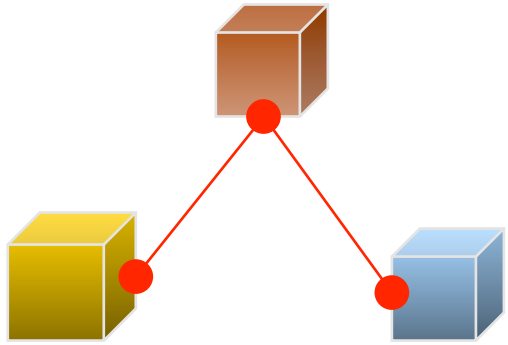
- Relations have to be
  - Comprehensible
  - Limited and simple
  - Non redundant



Encapsulation

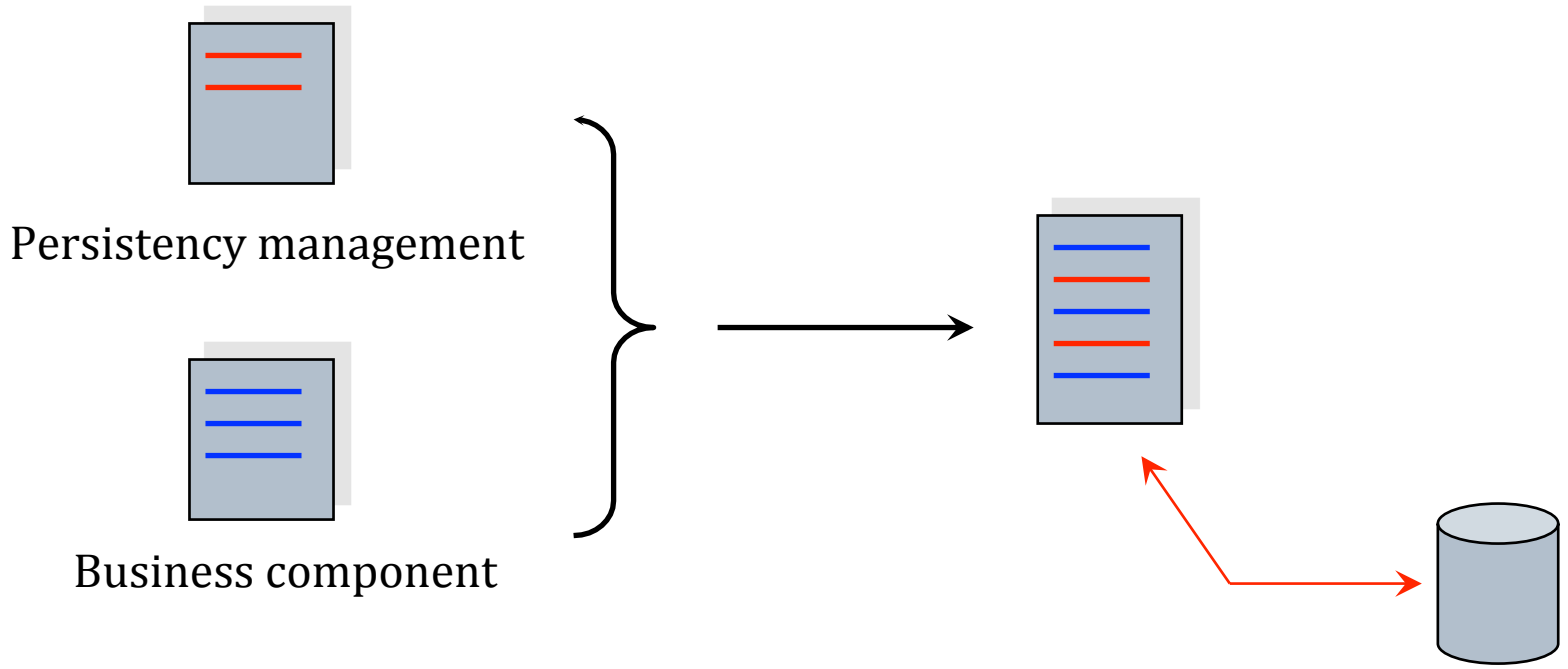
- Information masking
  - Modules structures and internal data have to be private
  - It relates to low level design decisions – not the concern of the other modules

- Modules communicate through high level information



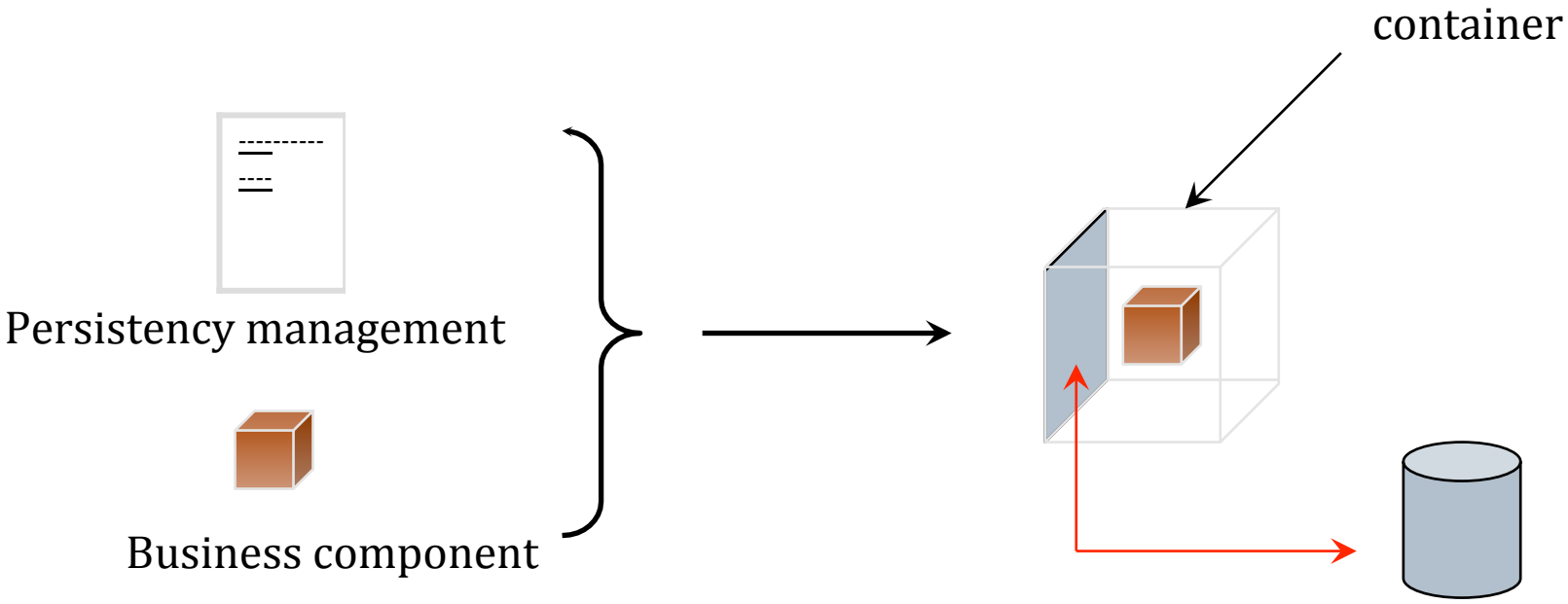
Separation of concerns - 1

- Aspect oriented programming (AOP)
  - Concerns are added through code injection

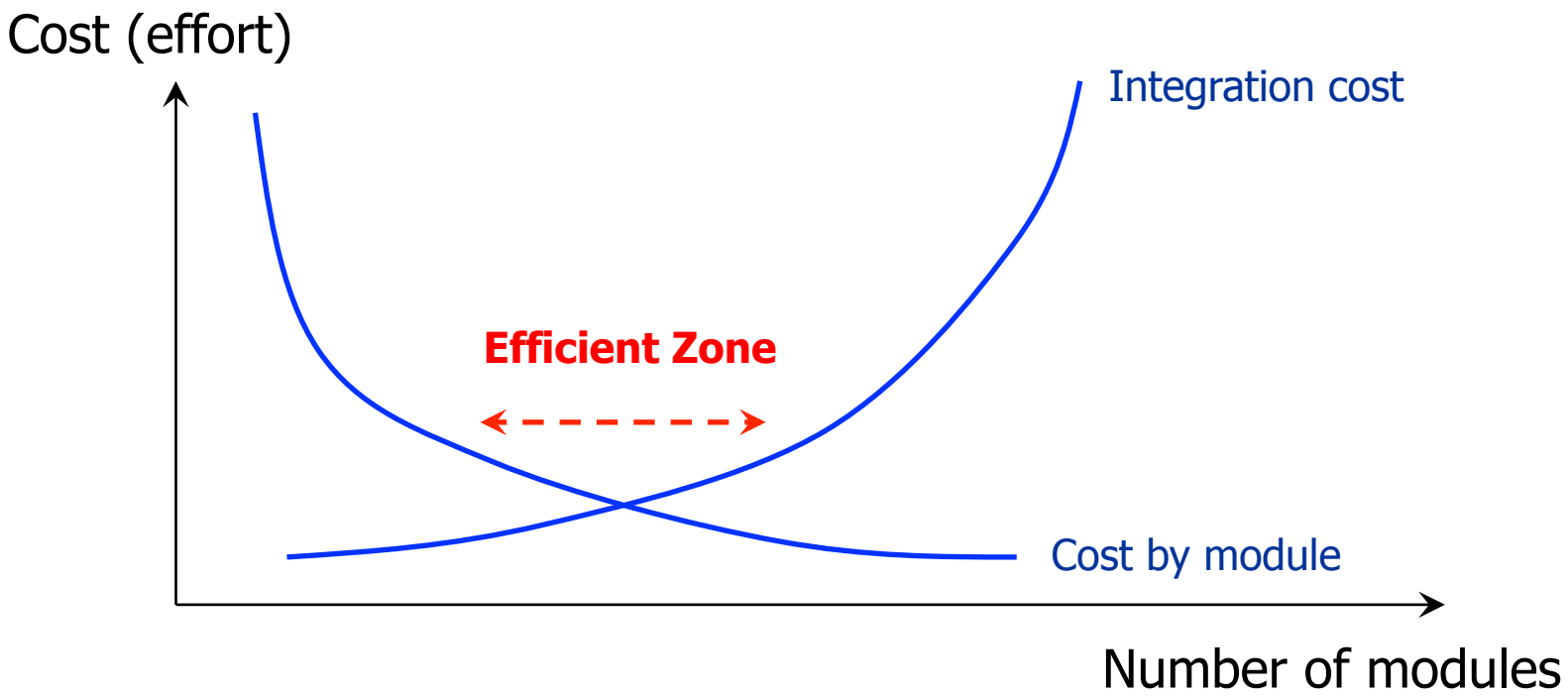


Separation of concerns - 2

- Components are executed in a container
  - Concerns are added in various ways (static or dynamic proxies, code injection, …)



How many modules



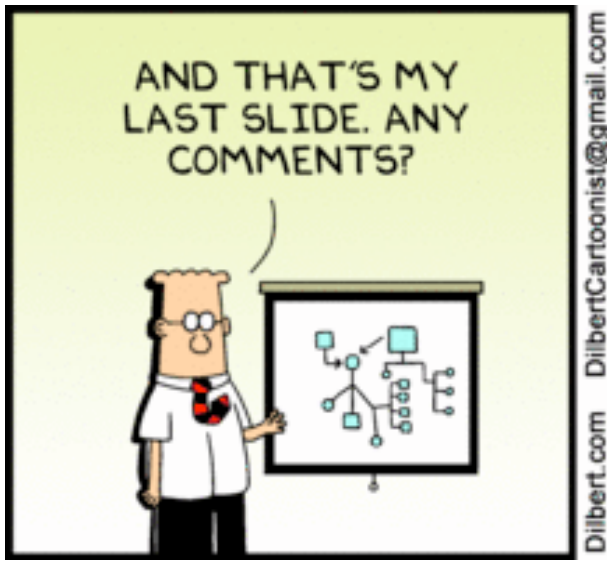
Adapted from de « Software Engineering: a practitioner approach » de R. Pressman

Simplicity

- Reducing complexity
  - Simplify structure as much as possible without changing the nature of solution
  - Simplicity is good for understanding and evolution

Side note

- Complicate designs are often done on purpose
  - Games people play!



Technology agnostic

- A design should not include technologies
  - It is premature
  - It has to be business oriented
  - It has to be logical

- Clean separation between design and code is not always possible
  - It has to be looked for however

Outline

- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

Cohesion and coupling

- Two major criteria when evaluating a design
  - Within development team
  - In a formal review
- Cohesion
  - Why some elements are grouped together ?
- Coupling
  - Why some elements communicate ?

Cohesion

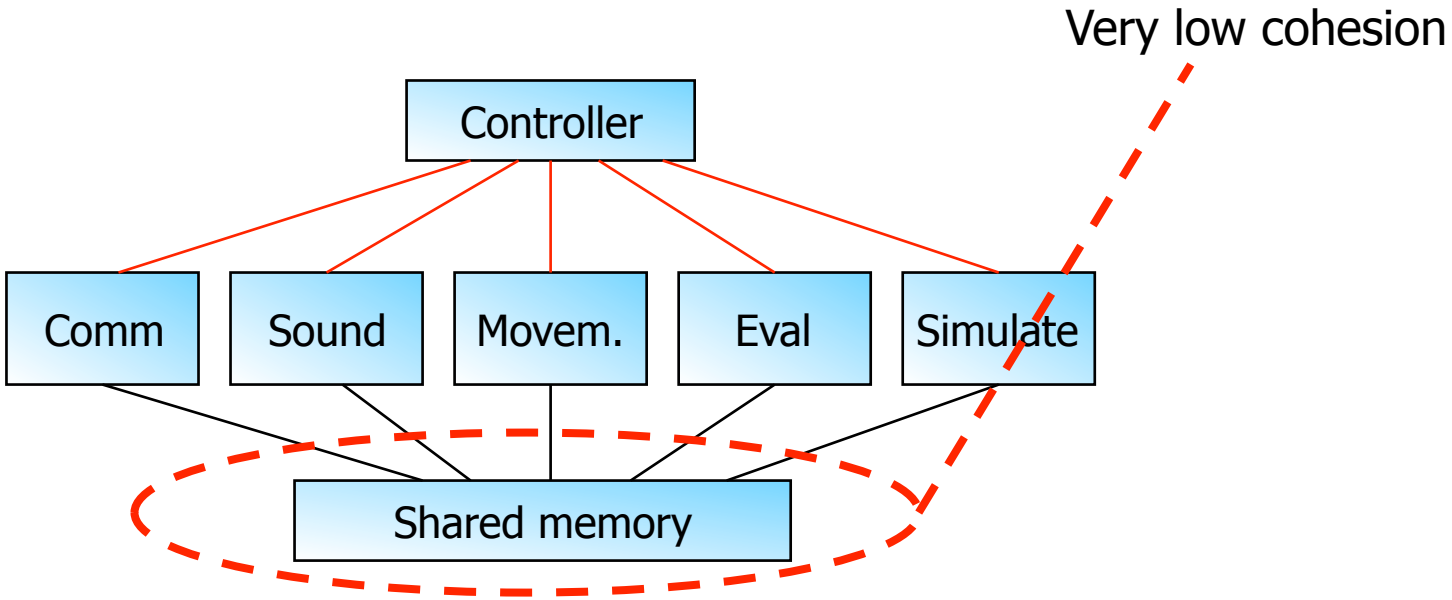
- Cohesion is about the functional scope of a module. It is related to:
  - Coherency
  - Largeness
- Elements that are grouped together are to be there for a good reason
- Cohesion is good for understandability and evolution
  - Elements to be changed are likely to be together

Cohesion levels

- Four levels of cohesion
  - Accidental
  - Syntactic
  - Temporal
  - Functional
- Qualitative measurement

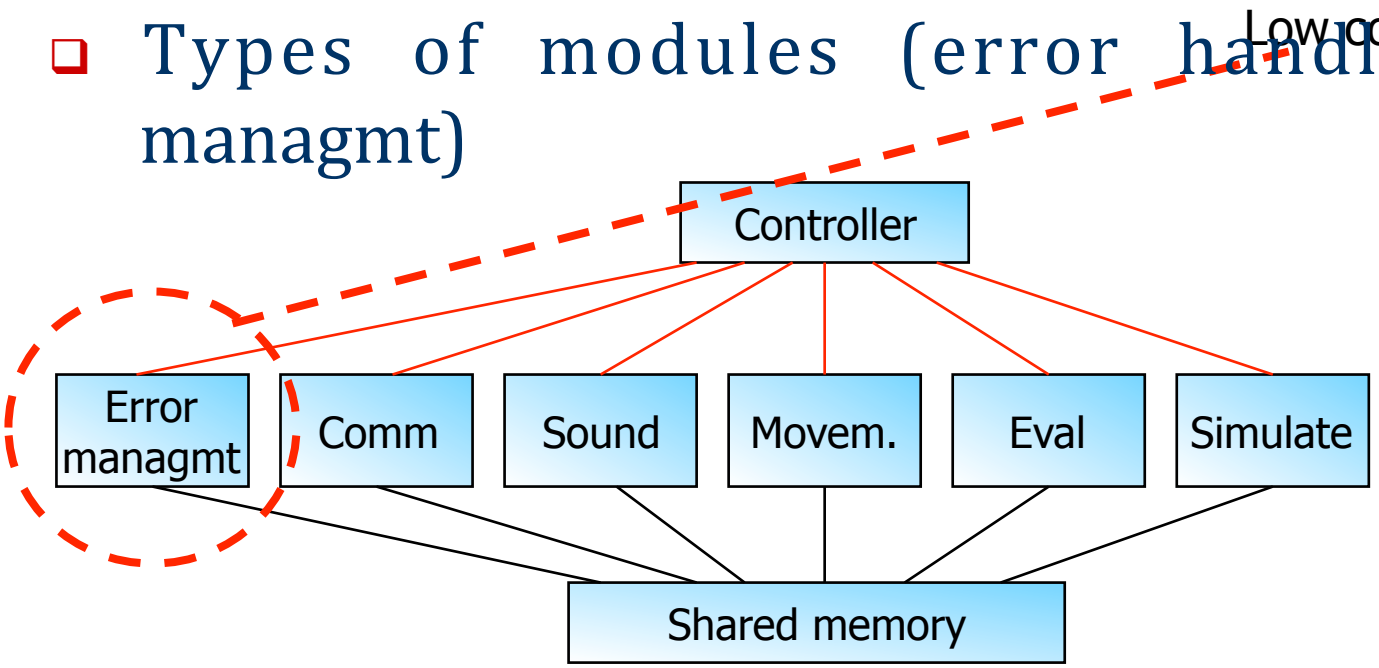
Accidental cohesion

- Elements have been grouped without apparent logic
  - Miscellaneous modules
  - Data stores used by any components, …



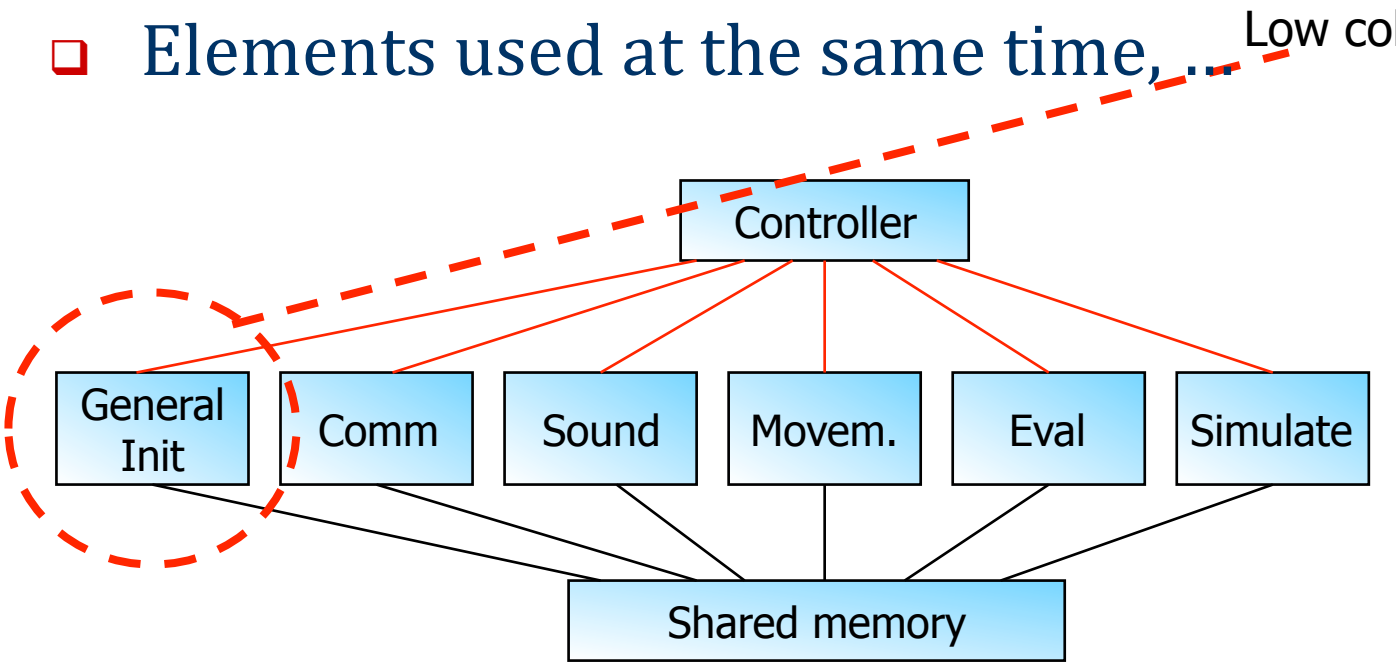
Syntactic cohesion

- Elements have been grouped on syntactic basis
  - Names of modules / classes / components
  - Types of modules (error handling, com managmt)



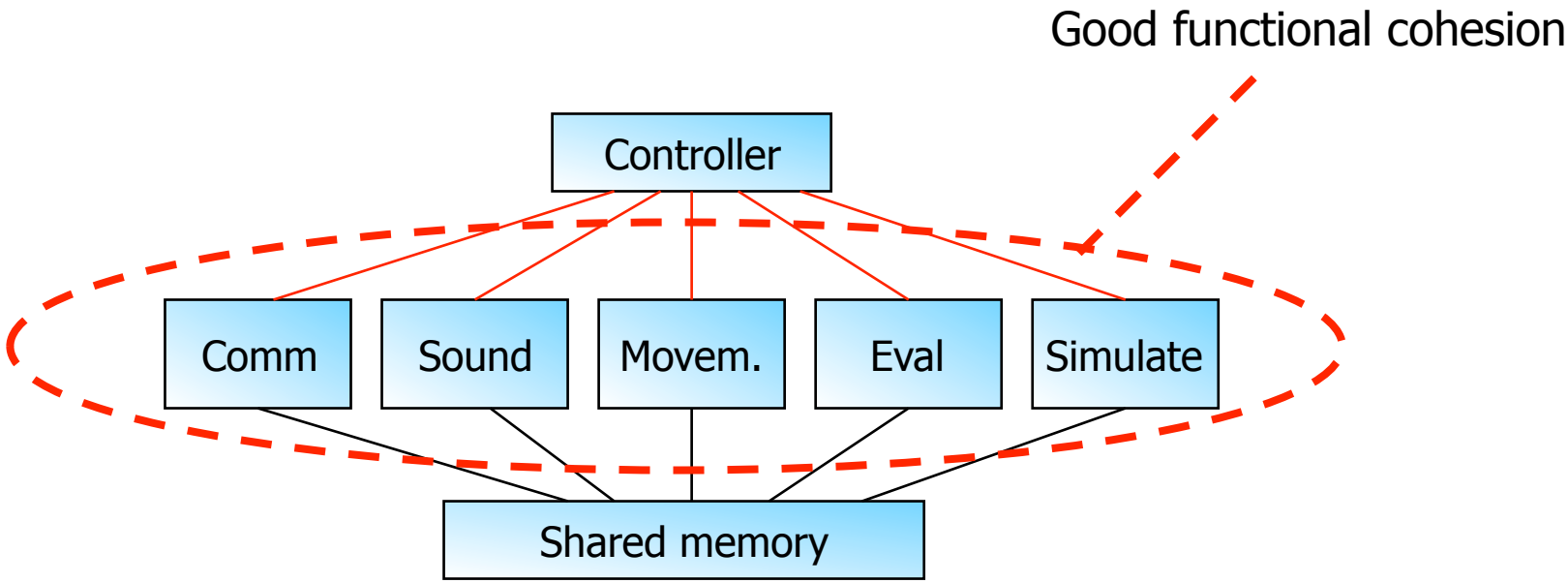
Temporal cohesion

- Elements have been grouped on temporal basis
  - Elements activated at the same time
  - Elements used at the same time, ...



Functional cohesion

- Elements necessary to execute a function are grouped together

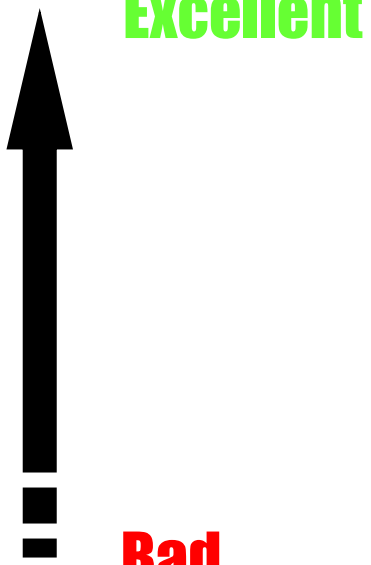


Coupling

- Coupling is about relationships between modules
- Two kinds of dependencies
  - Number of relations
  - Nature of relations
- Coupling is good for understandability and evolution
  - Side effects are limited

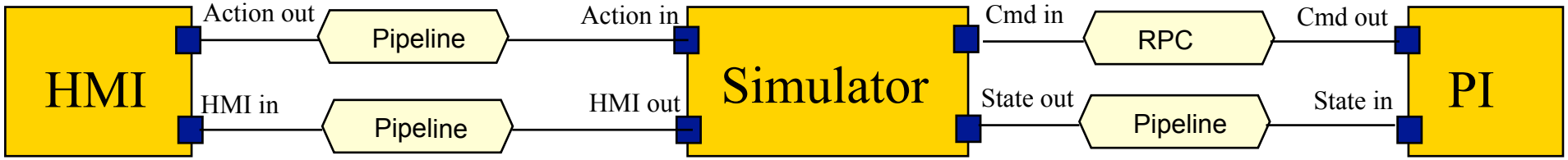
Coupling levels

- Six levels of coupling
  - No direct coupling
  - Data coupling
  - Coupling by reference
  - Control coupling
  - External coupling
  - Content coupling
- Qualitative measurement



No direct coupling

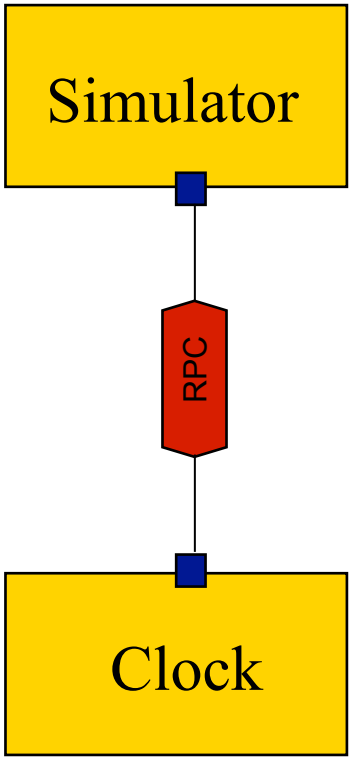
- Modules with no relations and no shared data
  - For instance, HMI and PI are not closed and have no shared information
- Evolution
  - No impact



Data coupling

- Modules exchanging data by value through their interfaces

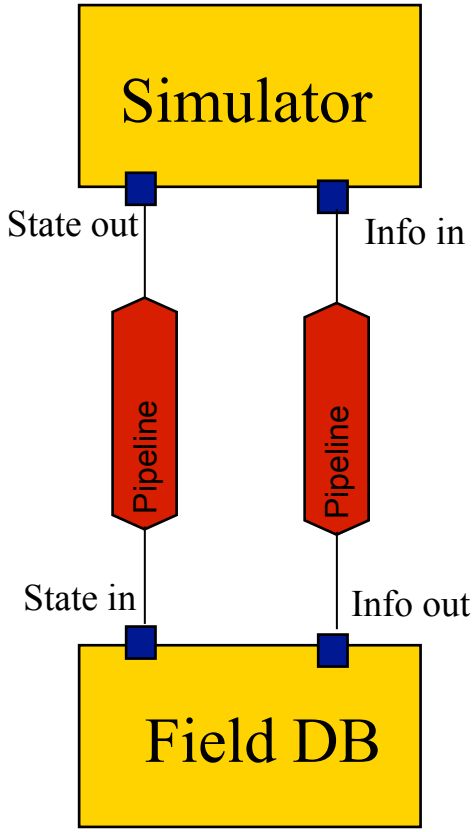
- Evolutions
  - Modules are sensible to interface evolution



Coupling by reference

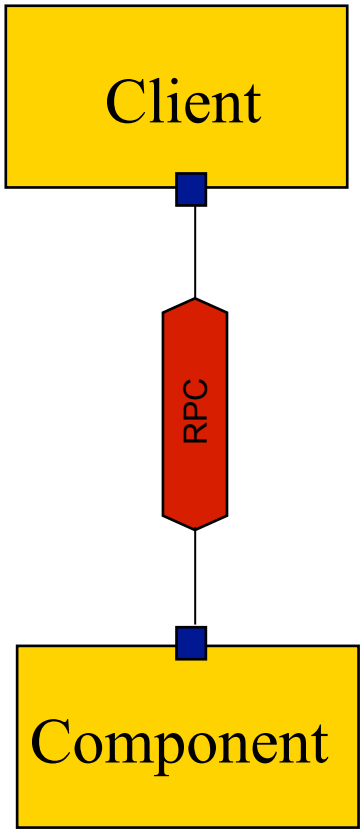
- Modules exchanging data by address through their interfaces

- Evolution
  - Modules are sensible to interface evolution and to data structure evolution



Control coupling

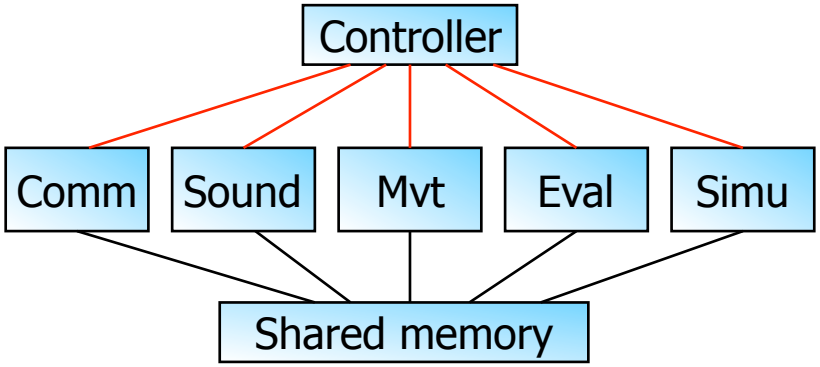
- The interface of a module allows to influence its behavior
- Evolution
  - Modules are sensible to interface evolution and to internal functions evolution



External coupling

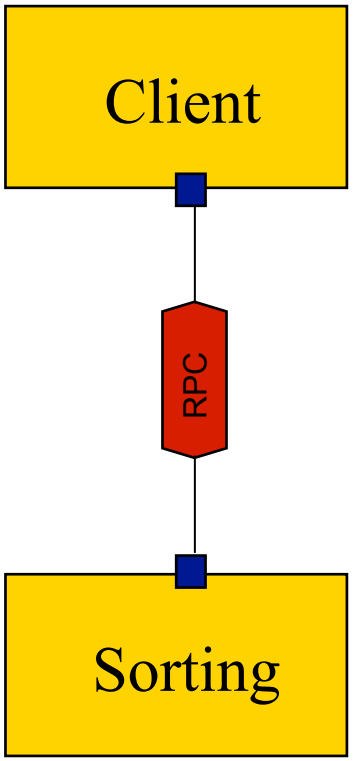
- Components communicate through an intermediary tier
  - sort of global variable
  - Problem: the communication canal can be forgotten

- Evolution
  - Sensible to any change in the tier



Content coupling

- A module exploits the content of another one, not through interfaces
  - Private variables, constants, logical structures
- Evolution
  - Once the content is known and used, it cannot evolve



Outline

- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

The challenge

- Find out the right decomposition
  - The modules and their relationships

- Ensure that
  - Requirements are met
  - Expected COTS are used
  - Evolutions are possible
  - Traceability is managed

How to design ?

- Mine/examine requirements
- Define or identify modules
  - Top down
  - bottom up
- Organize modules
  - Design patterns
- Evaluate the result
  - Cohesion and coupling
  - Review

Requirements mining

Libellé	Numéro	Catégorie
Restitution sensation de mouvements, secousses	E 3 SYS	MVT
Restitution des sensations de vibrations du char	E 4 SYS	MVT
Vision partie visible du canon	E 6 SYS	VIS
Symbologie pilote pour épiscope central	E 7 SYS	VIS
Rétroviseurs dans les épiscopes latéraux	E 8 VIS	VIS
Visuel : restitue conditions climatiques (EAU)	E 9 VIS	VIS
Visuel : restitue type et conditions d'observation	E 10 VIS	VIS
Visuel : Feux de signalisation	E 12 VIS	VIS
Restitue l'environnement sonore du poste de pilotage	E 14 SYS	SON
Simulation bruits de roulement	E 15 SIM	SON
Simulation bruits de châssis (moteur, trans, venti)	E 16 SIM	SON
Simulation bruits tourelle et de tirs 120 mm	E 17 SIM	SON
Emet les alertes vocales	E 19 PHO	SON
PCA : Suivre et contrôler le travail des élèves	E 21 PCA	INS
PCA : Créer et modifier des exercices	E 22 PCA	INS
PCA : Répétition état pupitres et commandes pilote	E 23 PCA	INS
Piloter à l'aide d'un mini-manche	E 27 SYS	INS

Top down approach

- Principle
  - Decompose into sub problems
  - Implement or reuse the designed components

- Issues
  - Find out the right granularity level
  - Ensure interoperability with existing components ?
  - How to be sure not to divagate (ignoring low level details)?

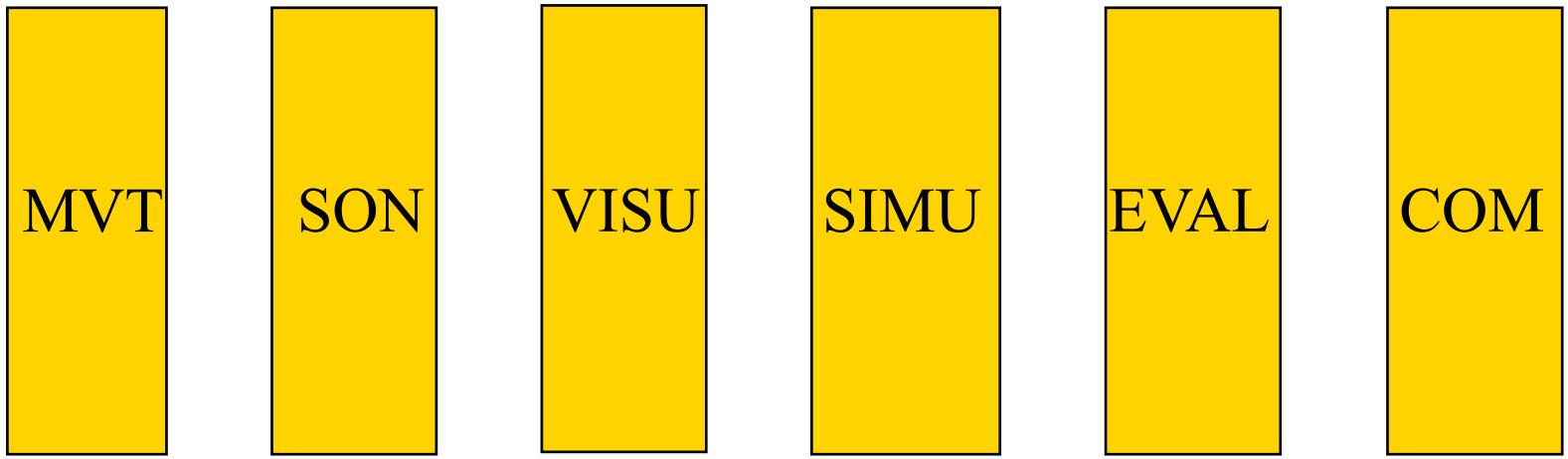
Bottom up approach

- Principle
  - Define/reuse components – based on the existing base
  - Assemble these components to make up the system

- Issues
  - No global view
  - Lost of time on details
  - Ensure transverse properties
  - Ensure good decomposition (coming from existing assets)

Example

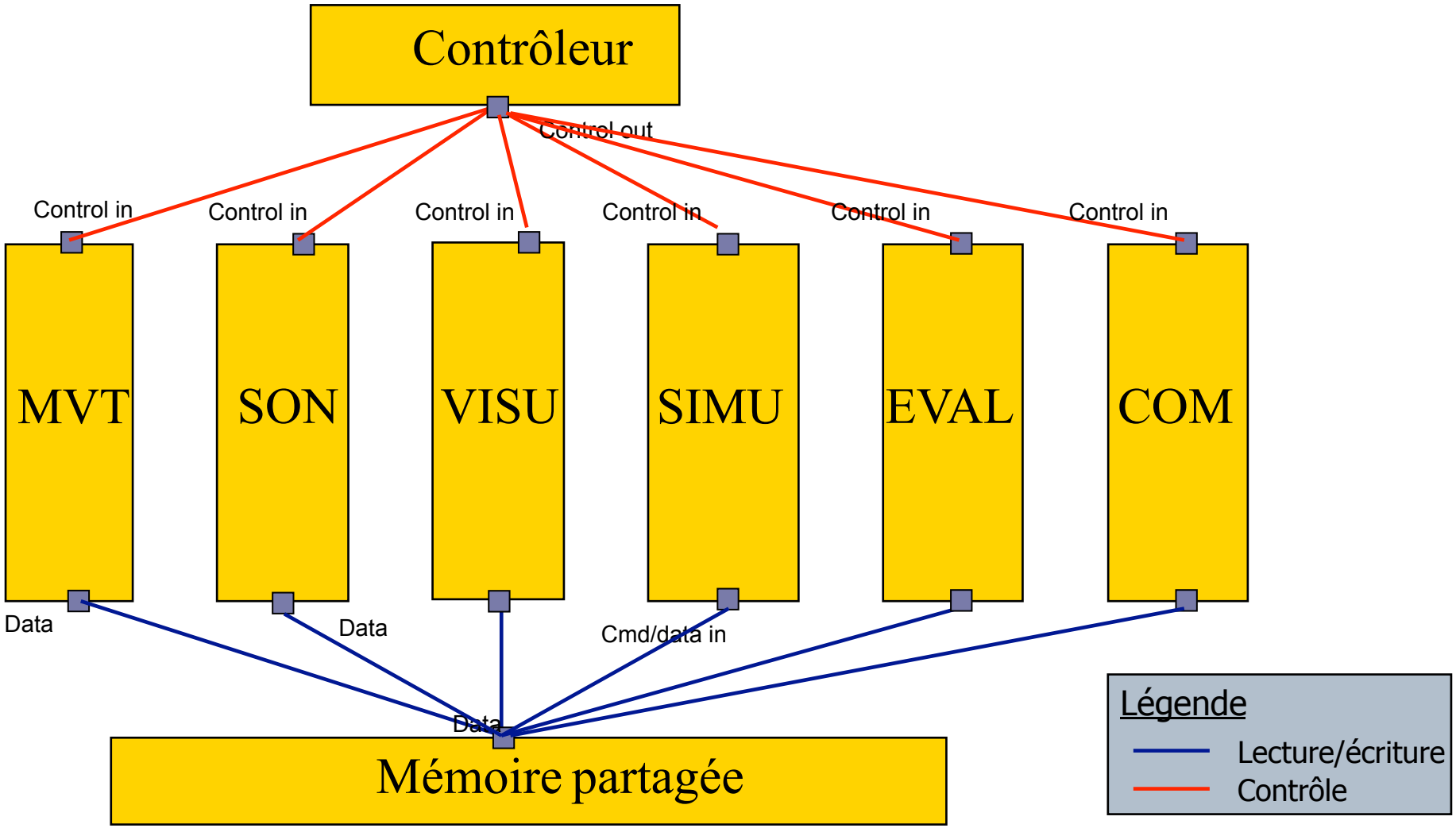
- Identification of functional components



Organization

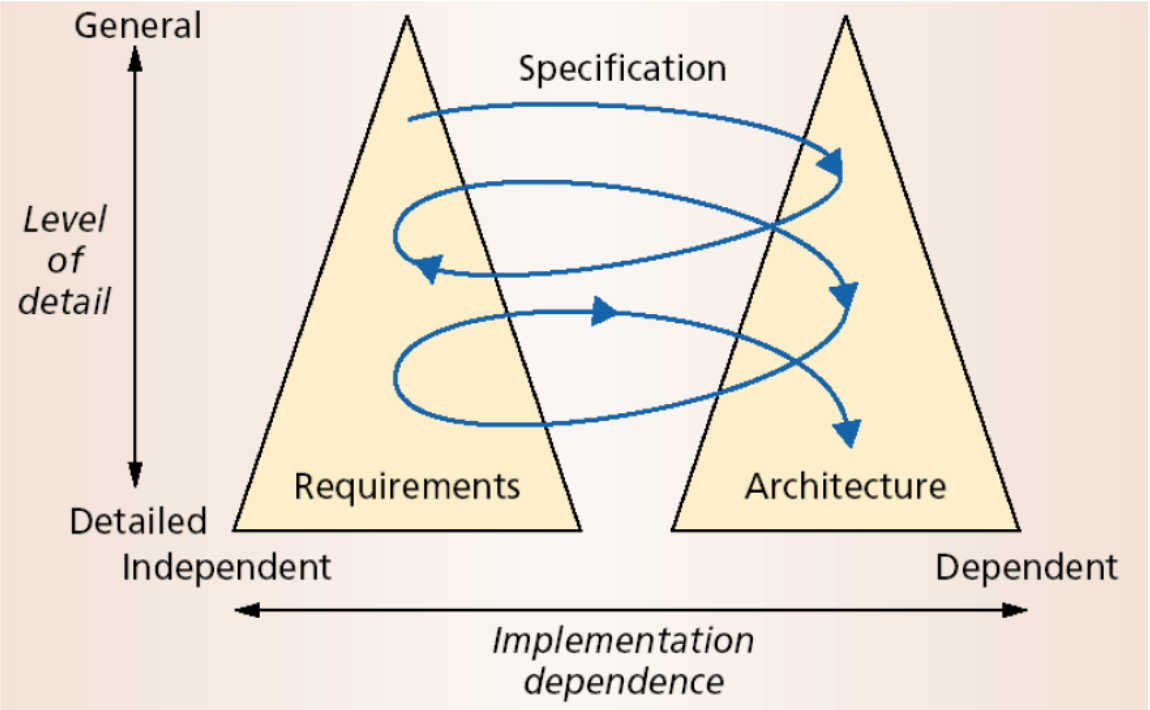
- Topology comes from technical reasons
  - Non functional properties meeting
  - Performance, availability, security, …
- Use of patterns
  - Reusable patterns that can be adapted to meet the project requirements

Example


Contrôleur
Control out
Control in
Control in
Control in
Control in
Control in
Control in
MVT
SON
VISU
SIMU
EVAL
COM
Data
Data
Cmd/data in
Data
Mémoire partagée
Légende
Lecture/écriture
Contrôle

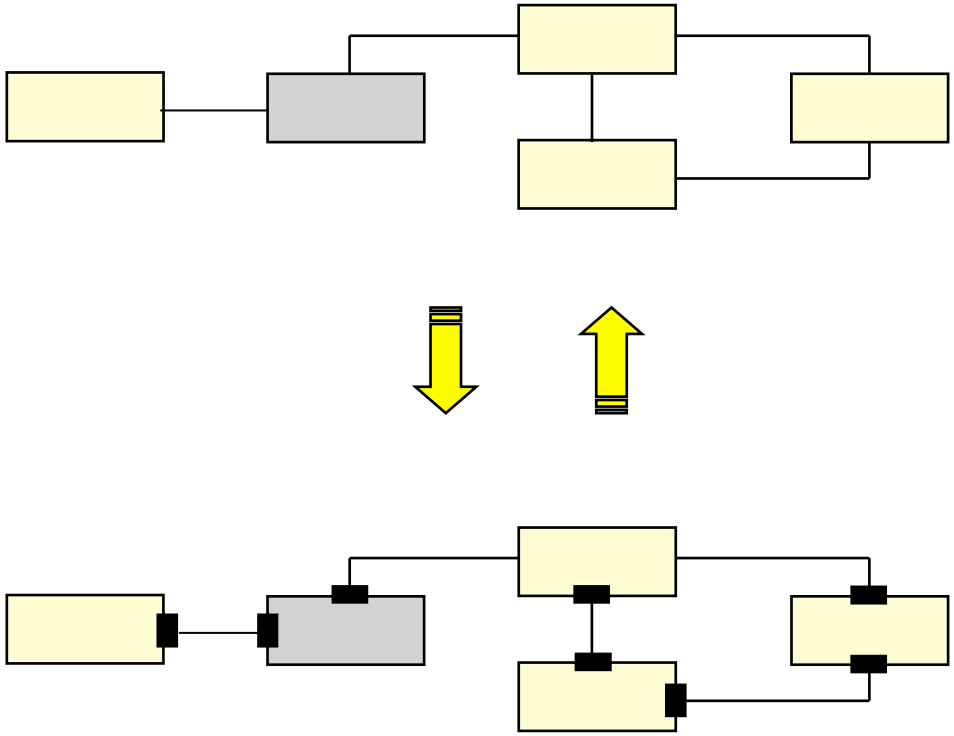
Incremental activity - 1

- Nuseibeh : Twin peak model



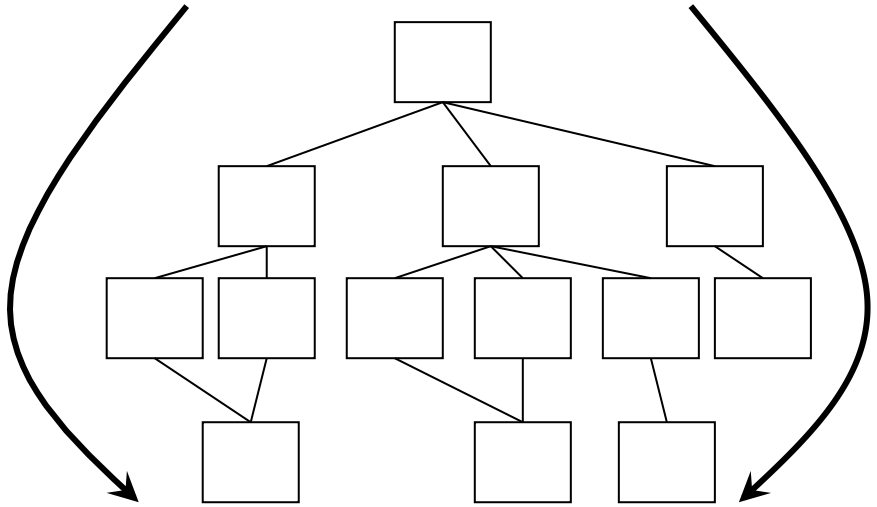
Incremental activity - 2

- Incremental interface definition



Incremental activity – fan in / fan out

- Fan in: introduce all concepts
- Fan out: reduce the architecture
  - Concepts grouping
- Encourages reuse of the lower levels



Outline

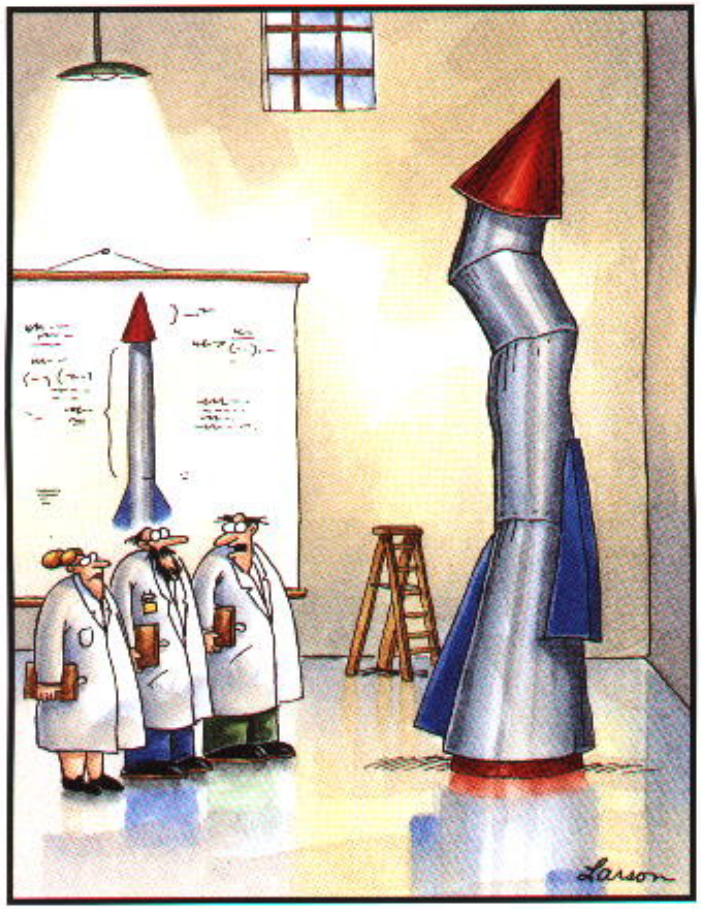
- Introduction
- Models
- Design principles
- Cohesion and coupling
- An iterative process
- Conclusion

Conclusion

Avoiding shaky constructions

is of major importance

Good design is key !



"It's time we face reality, my friends. ...
We're not exactly rocket scientists."