UNIVERSITÄT
BERN

Introduction to Software Engineering

10. Software Architecture

Andrea Caracciolo

Adapted from slides by **Oscar Nierstrasz** and **Mircea Lungu**

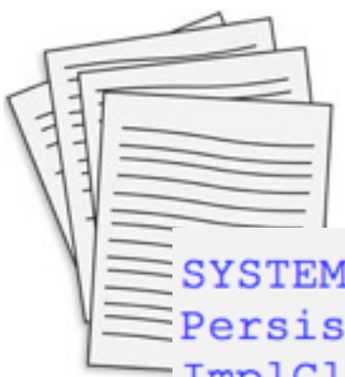
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
- UML diagrams for architectures

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
- UML diagrams for architectures

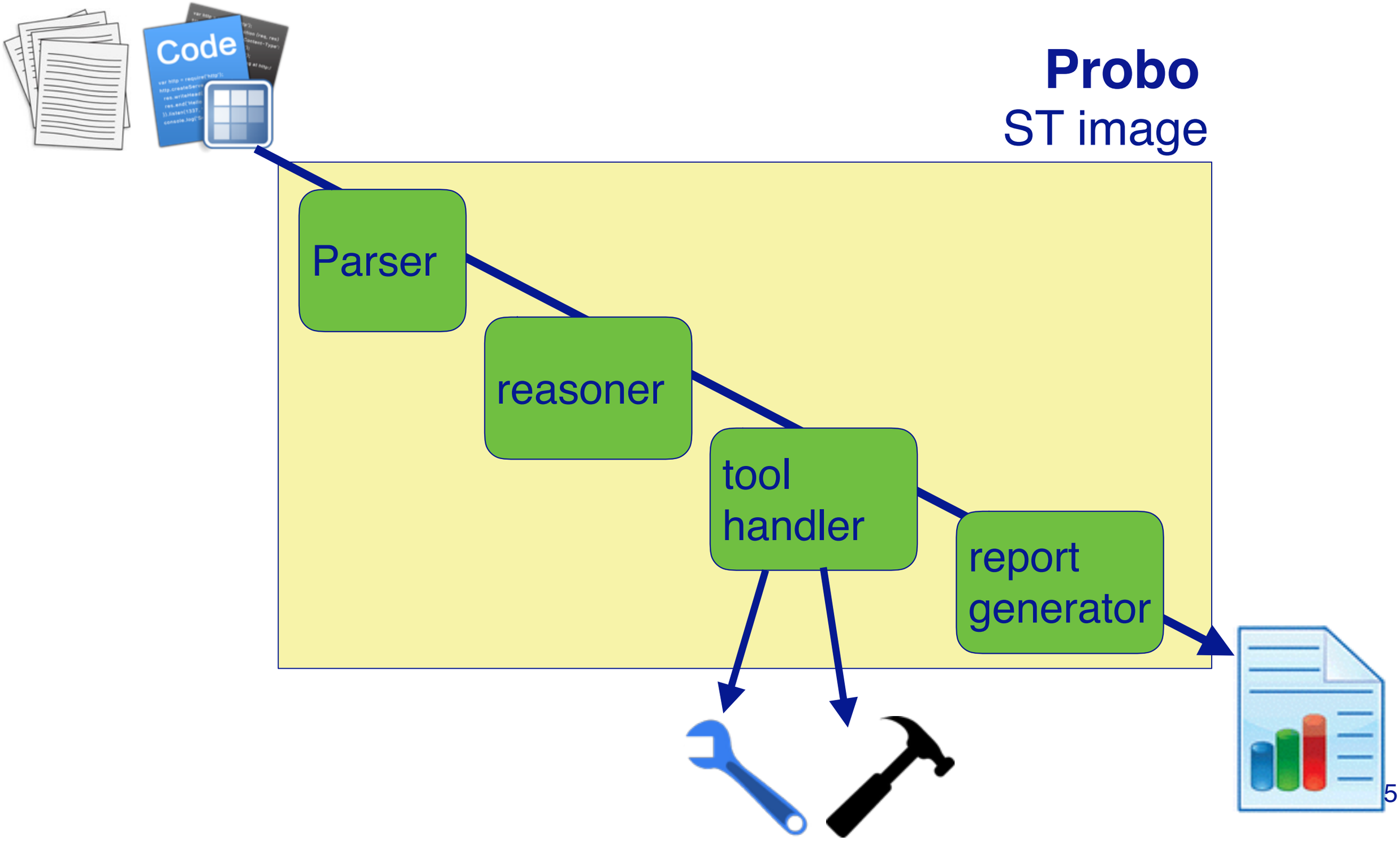
Example Architecture



```
SYSTEM cannot contain cycles
PersistencePackage cannot depend on ServicePackage
ImplClass must have annotation "@πService"
```

Probo

Batch Process



Client/Server

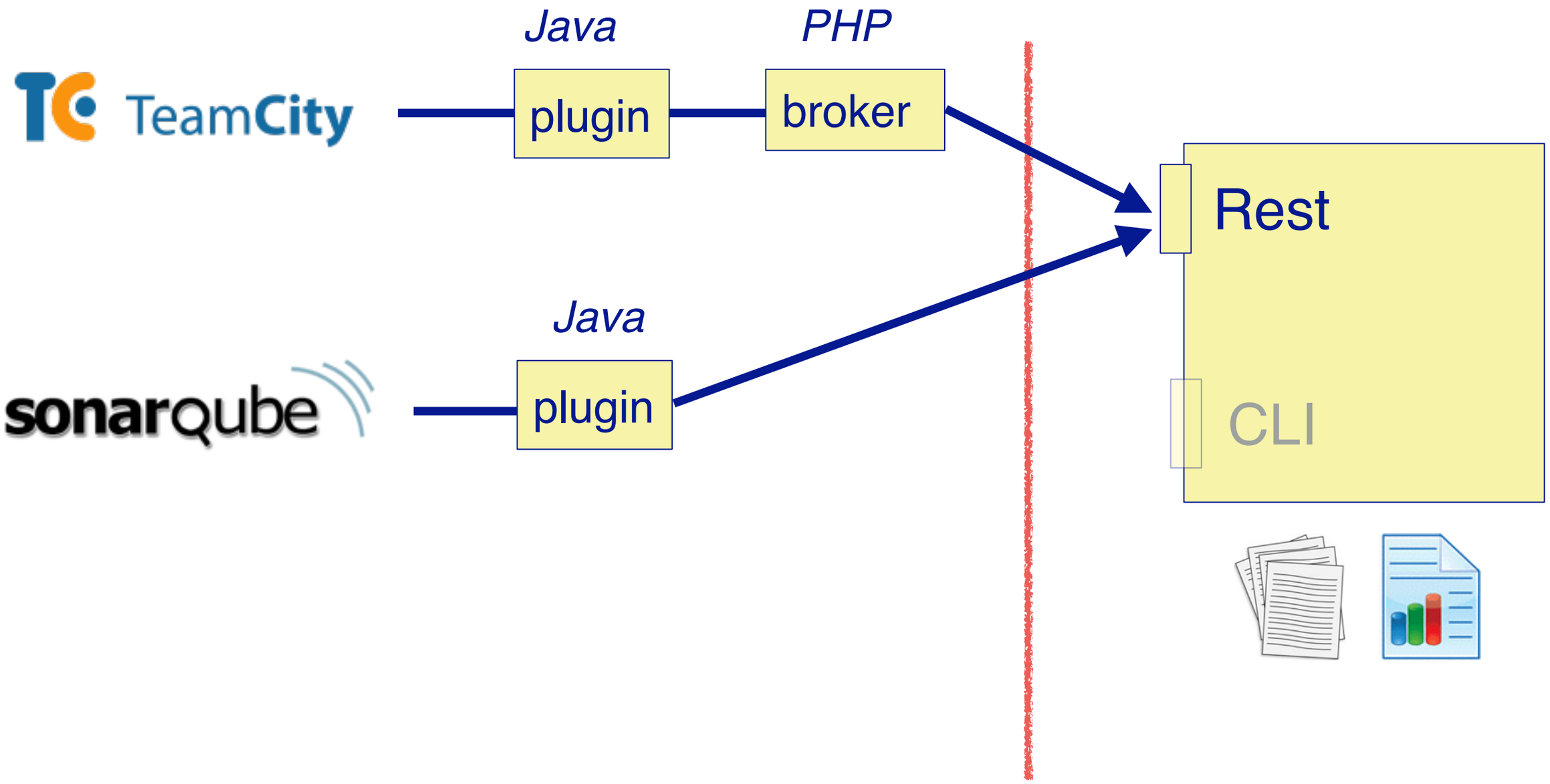
Client

Server

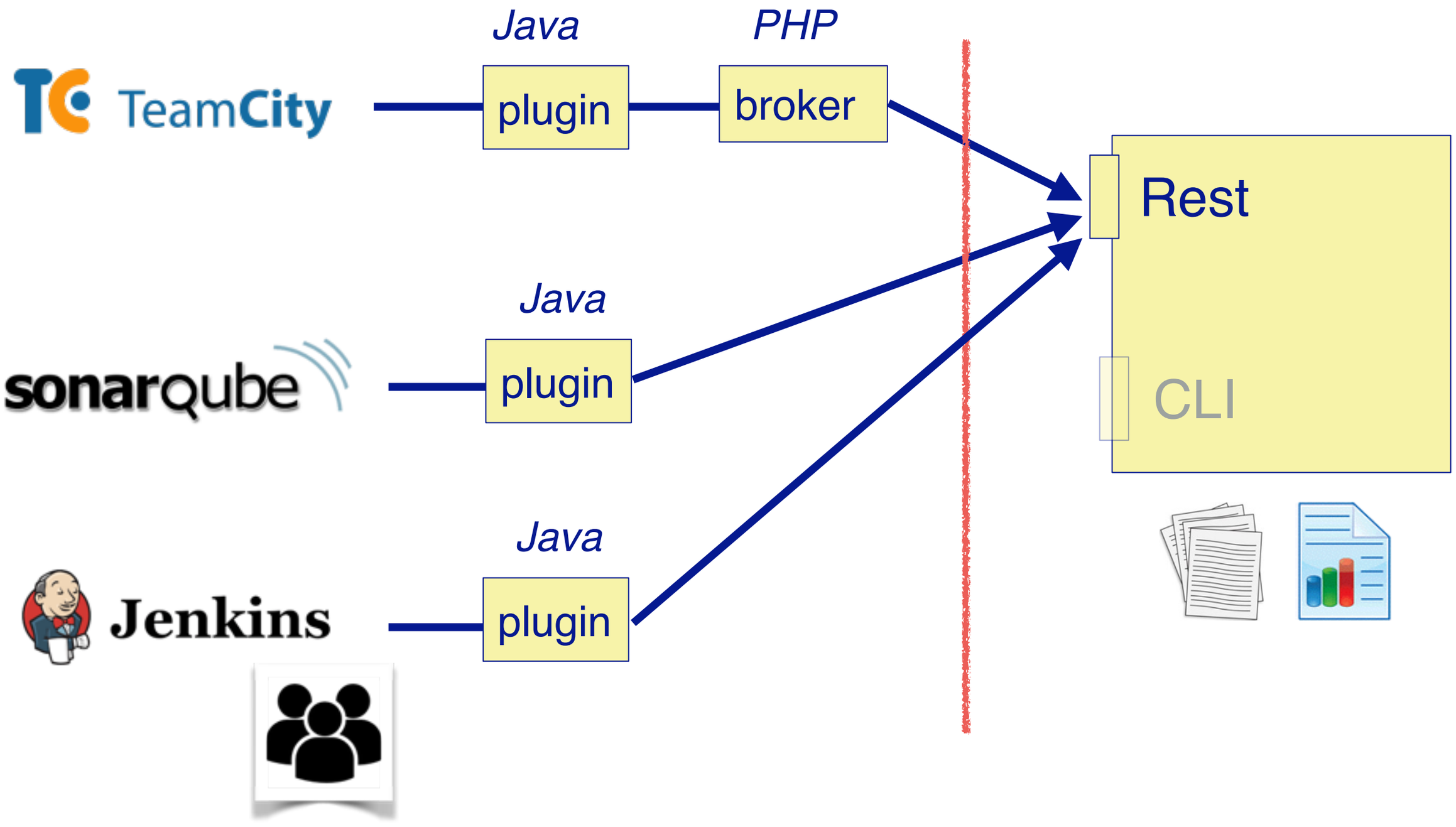
Rest

CLI

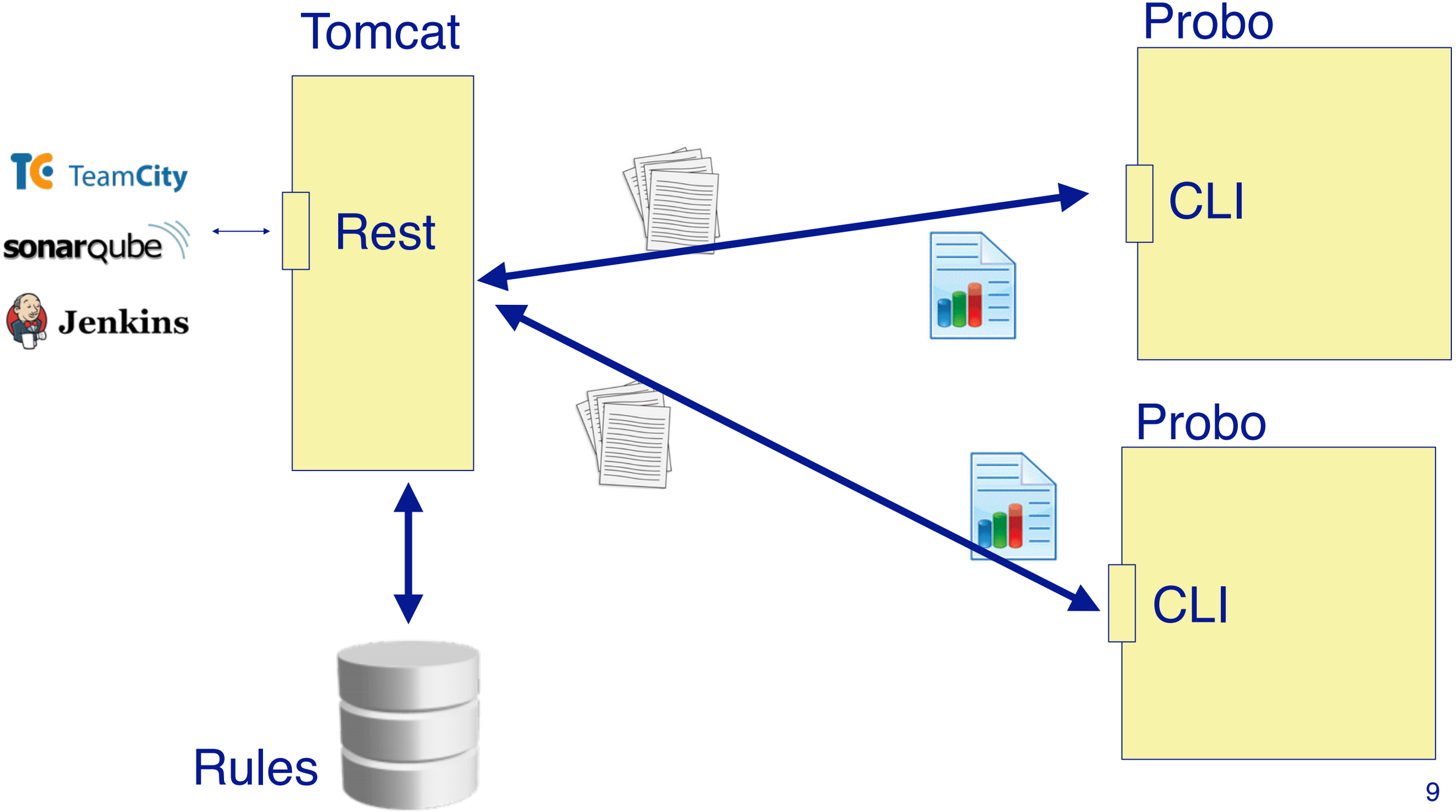
Integration - Analysis as service



Integration - Analysis as service



Scalability



What is Software Architecture?

Grady Booch @Grady_Booch · Nov 14

All architecture is design, not all design is architecture; architecture is most significant design decisions

Architecture: The set of ***design decisions*** about any system (or subsystem) that keeps its implementors and maintainers from exercising ***needless creativity***.

What is Software Architecture?

Grady Booch @Grady_Booch · Nov 14

All architecture is design, not all design is architecture; architecture is most significant design decisions

design decisions resulting in element properties that are **not visible** (make no difference outside the element) are **non-architectural**.

What is Software Architecture?

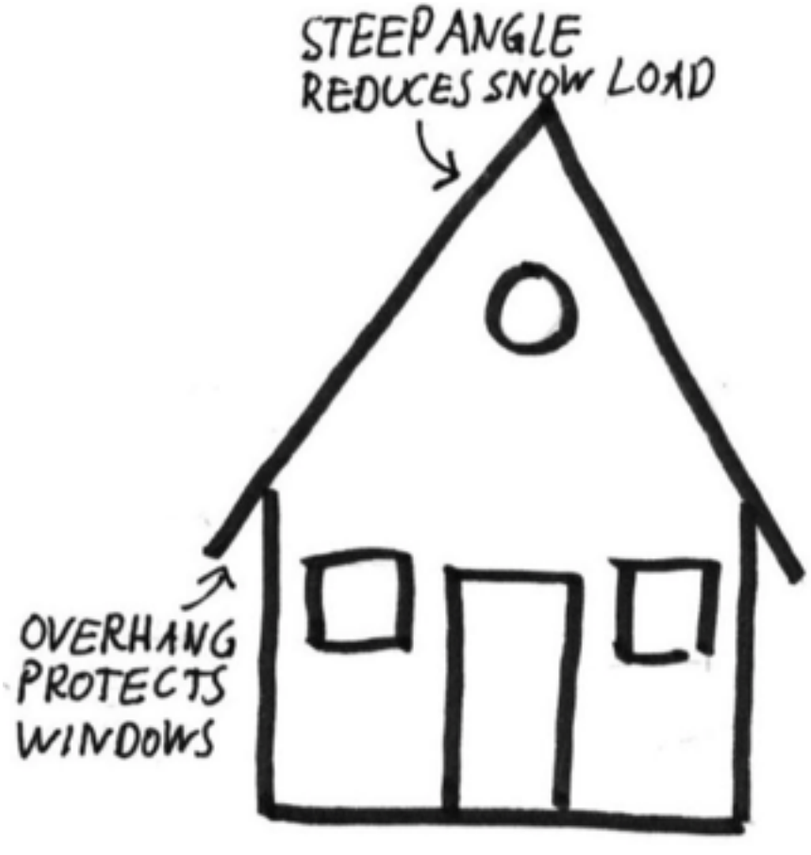
The architecture of a system consists of:

1. the *structure(s) of its parts*
   e.g. design-time, test-time, and run-time software and hardware parts
2. the *externally visible properties* of those parts
   e.g. provided services, performance, fault handling, shared resource usage
3. the *relationships and constraints* between them

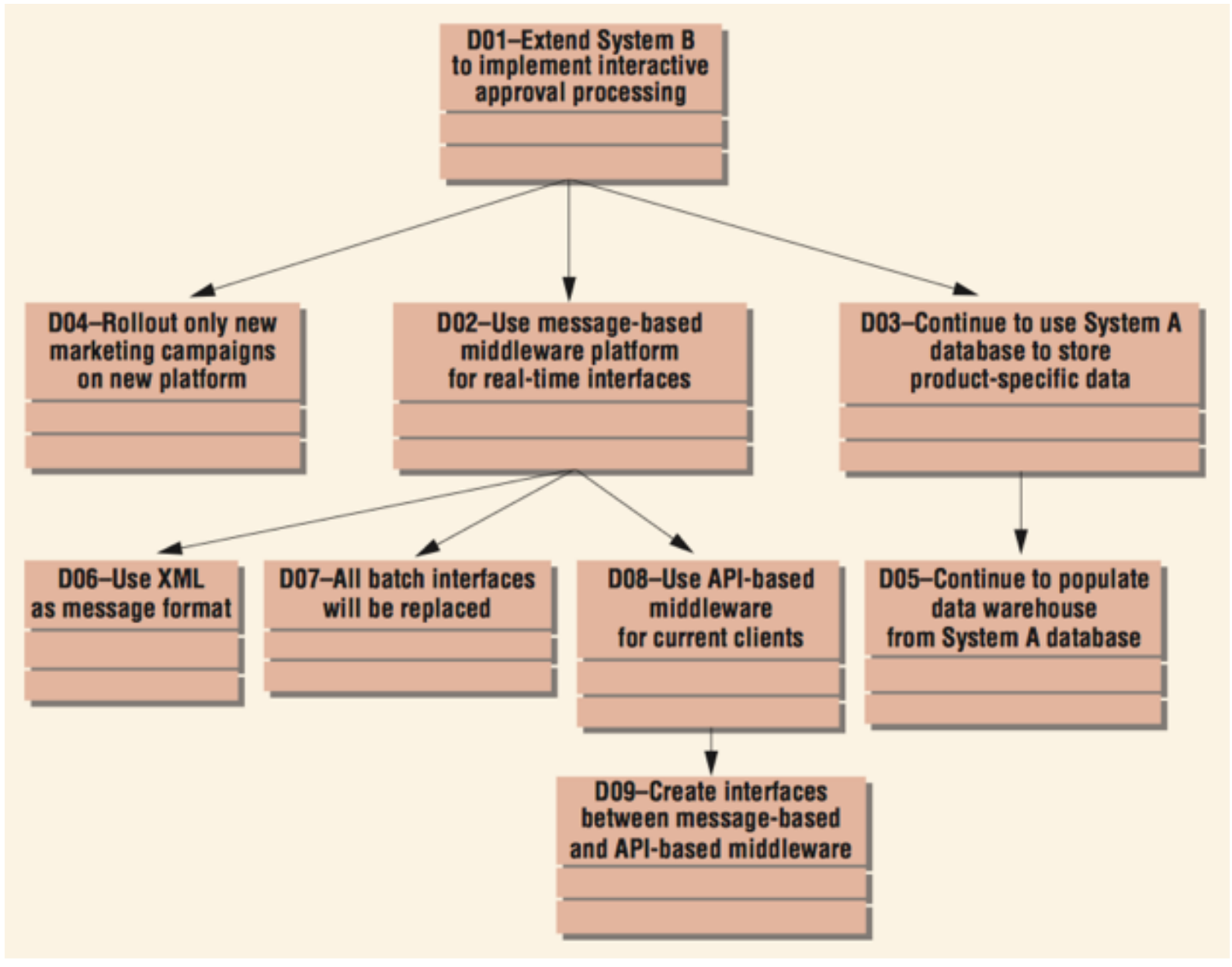
— *Bass & Clements, IEEE 1471*

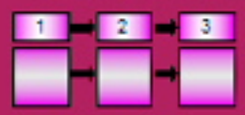
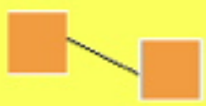
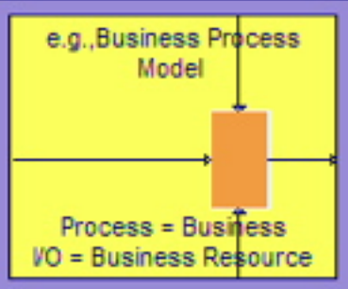
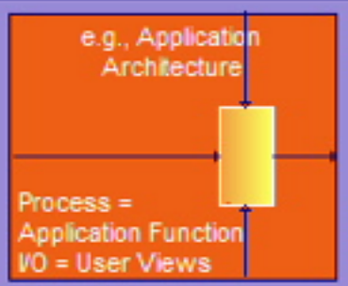
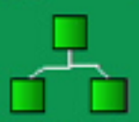
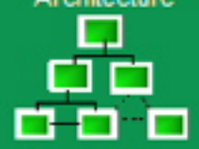
Rationale: Design Decisions

architectural decisions are ones that permit a system to meet its **quality attribute** and **behavioral requirements**.



Rationale: Design Decisions



	WHAT	HOW	WHERE	WHO	WHEN	WHY
	DATA	**FUNCTION**	**NETWORK**	**PEOPLE**	**TIME**	**MOTIVATION**
SCOPE {contextual} Planner	List of things important to the business  Entity = Class of business things	List of processes the business performs  Process = Class of business process	List of locations in which the business operates  Node = Major business locations	List of organisations important to the business  People = Major business unit	List of event cycles signifcant to the business  Time = Major Business Event Cycle	List of business goals/ strategies  End/Means = Major Business Goal/Strategy
BUSINESS MODEL {Conceptual} Owner	e.g., Semantic Model  Entity = Business Entity Relationship = Business	e.g.,Business Process Model  Process = Business I/O = Business Resource	e.g.,Business Logistics System  Node = Business Location Link = Business Linkage	e.g.,Workflow Model  People = Organisation unit Work = Work Product	e.g.,Master Schedule  Time = Business Event Cycle = Business Cycle	Business Plan  End = Business Objective Means = Business Strategy
SYSTEM MODEL {Logical} Designer	e.g., Logical Data Model  Entity = Data Entity Relationship = Data Relationship	e.g., Application Architecture  Process = Application Function I/O = User Views	e.g., Distributed System Model  Node = I/S Function Relationship = Line Characteristics	e.g., Human Interface Architecture  People = Role Work = Deliverable	e.g., Processing Structure  Time = System Event Cycle = Processing Cycle	e.g., Business Rule Model  End = Structural Assertion Means = Action Assertion
TECHNOLOGY MODEL {Physical} Builder	e.g., Physical Data Model  Entity = Segment/Table Relationship = Pointer/key	e.g., System Design  Process = Computer Function I/O = Data Elements/sets	e.g., Technology Architecture  Node = H/w /System s/w Relationship = Line Specifications	e.g., Presentation Architecture  People = User Work = Screen Formats	e.g., Control Structure  Time = Execute Cycle = Component Cycle	e.g., Rule Design  End = Condition Means = Action
DETAILED REPRESENTATIONS {Out-of-context} Subcontractor	e.g., Data Definition  Entity = Field Relationship = Address	e.g., Program  Process = Language Statement I/O = Control Block	e.g., Network Architecture  Node = Address Link = Protocol	e.g., Security Architecture  People = Identity Work = Job	e.g., Timing Definition  Time = Interrupt Cycle = Machine Cycle	e.g., Rule Specification  End = Sub-condition Means = step
FUNCTIONING ENTERPRISE	e.g DATA	e.g FUNCTION	e.g NETWORK	e.g ORGANISATION	e.g SCHEDULE	e.g STATEGY

http://en.wikipedia.org/wiki/Zachman_Framework

BUSINESS MODEL {Conceptual} Owner	e.g., Semantic Model Entity = Business Entity Relationship = Business	e.g.,Business Process Model Process = Business I/O = Business Resource	e.g.,Business Logistics System Node = Business Location Link = Business Linkage	e.g.,Workflow Model People = Organisation unit Work = Work Product
SYSTEM MODEL {Logical} Designer	e.g., Logical Data Model Entity = Data Relationship = Data Relationship	e.g., Application Architecture Process = Application Function I/O = User Views	e.g., Distributed System Model Node = I/S Function Relationship = Line Characteristics	e.g., Human Interface Architecture People = Role Work = Deliverable
TECHNOLOGY MODEL {Physical} Builder	e.g., Physical Data Model Entity = Segment/Table Relationship = Pointer/key	e.g., System Design Process = Computer Function I/O = Data Elements/sets	e.g., Technology Architecture Node = H/w /System s/w Relationship = Line Specifications	e.g., Presentation Architecture People = User Work = Screen Formats
DETAILED REPRESENTATIONS {Out-of-context} Subcontractor	e.g., Data Definition Entity = Field Relationship = Address	e.g., Program Process = Language Statement I/O = Control Block	e.g., Network Architecture Node = Address Link = Protocol	e.g., Security Architecture People = Identity Work = Job

Business Process

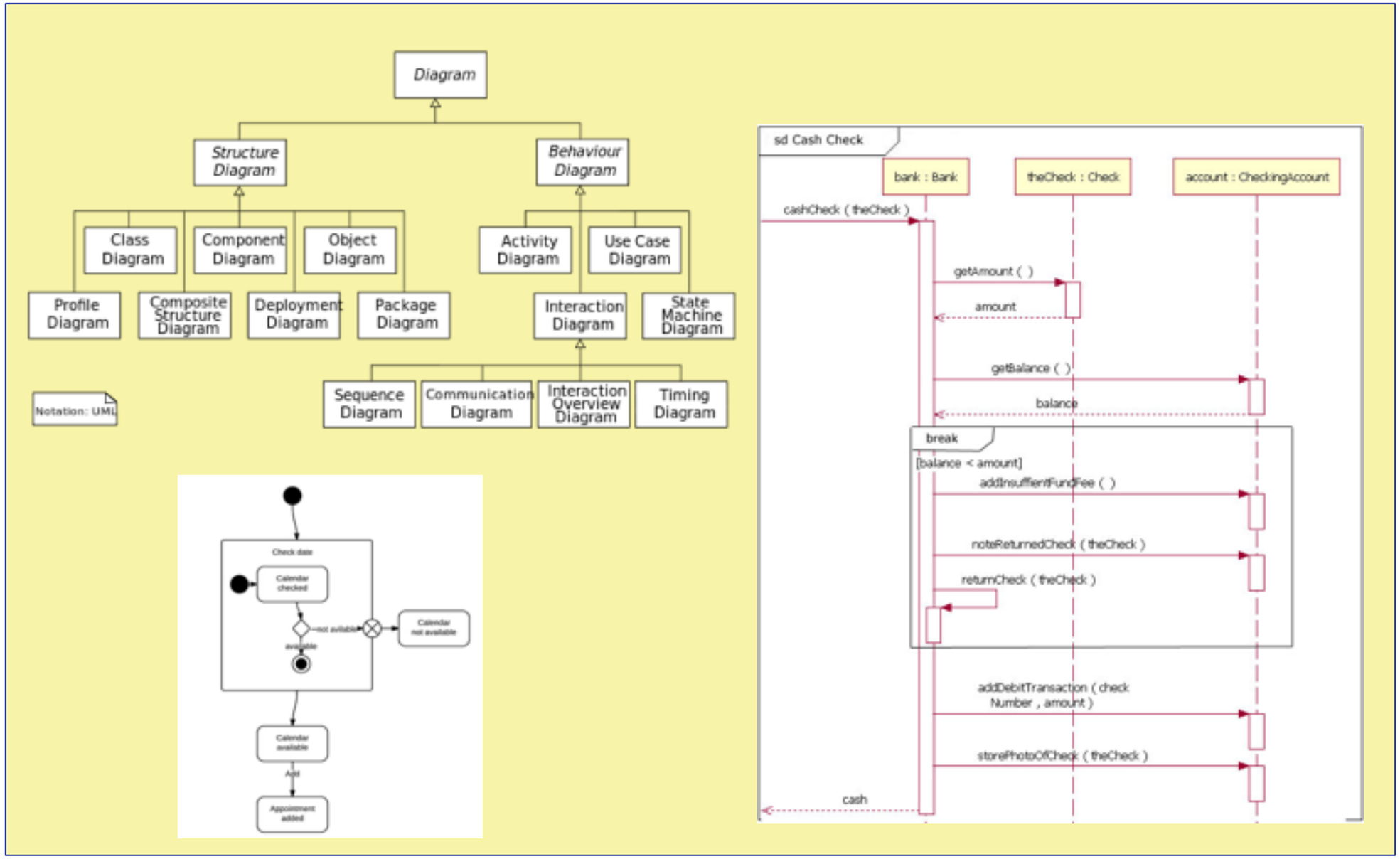
Architectural Design

Non-Architectural Design

Code

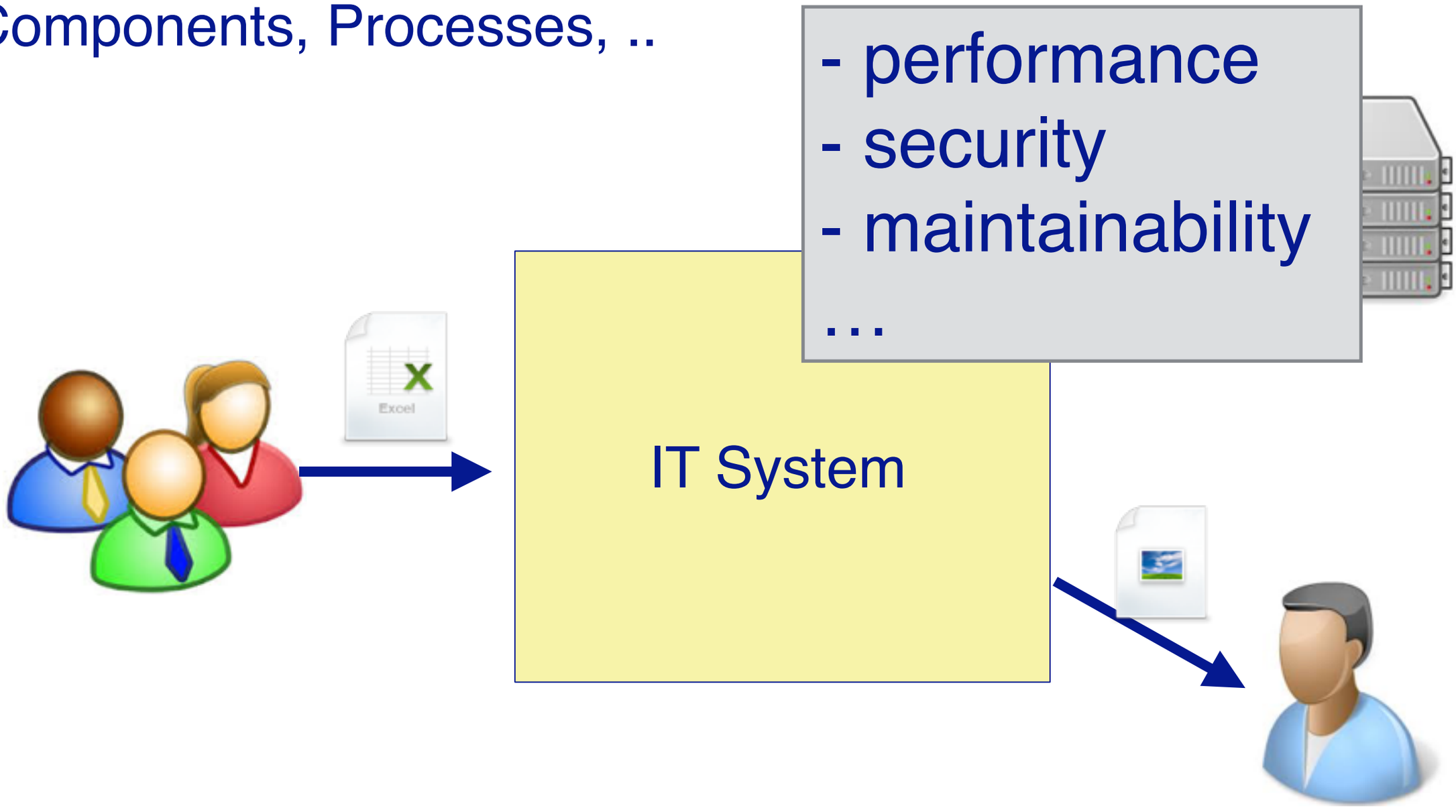
Non-Architectural Design

Objects, Functions, DB tables, ..



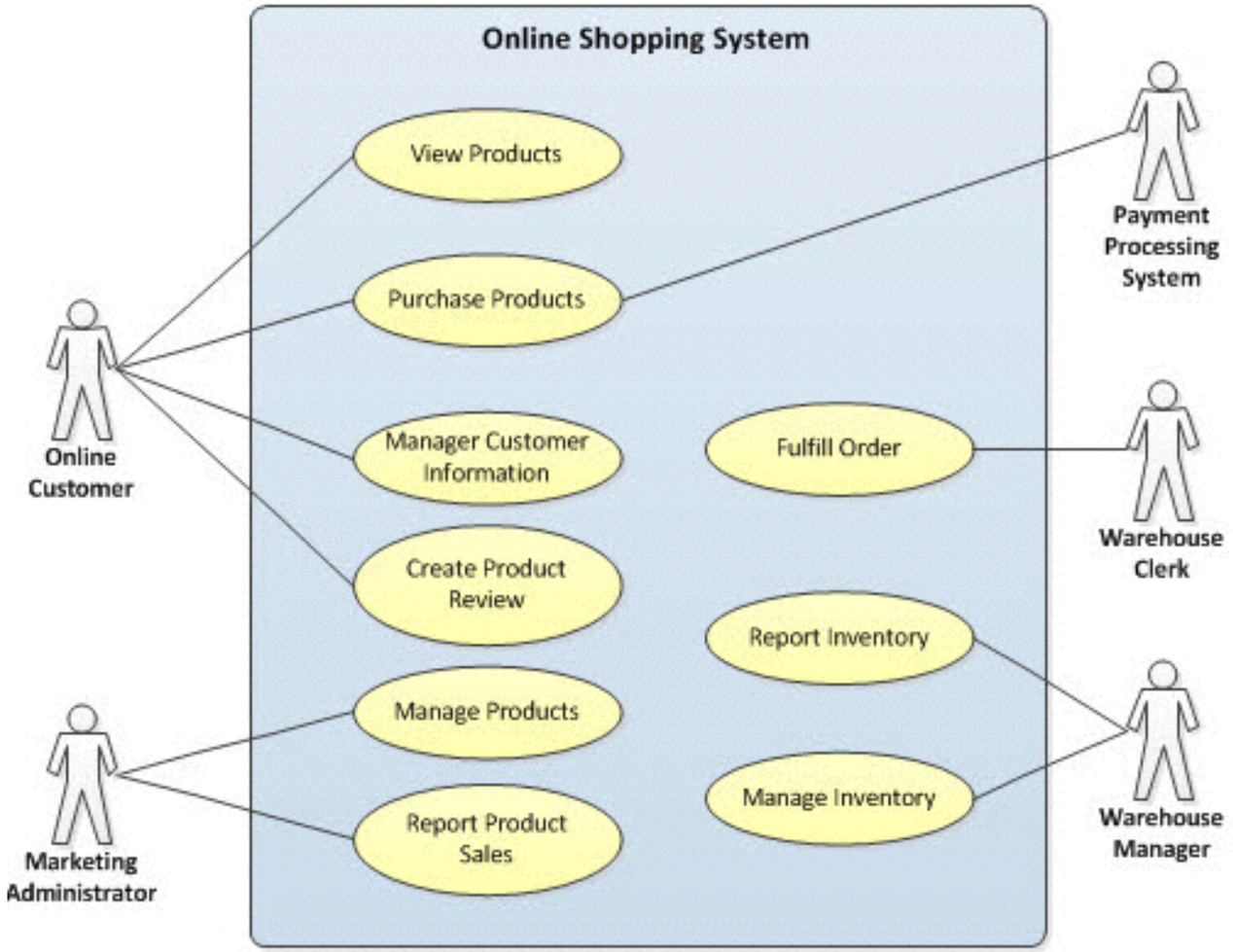
Architectural Design

Components, Processes, ..

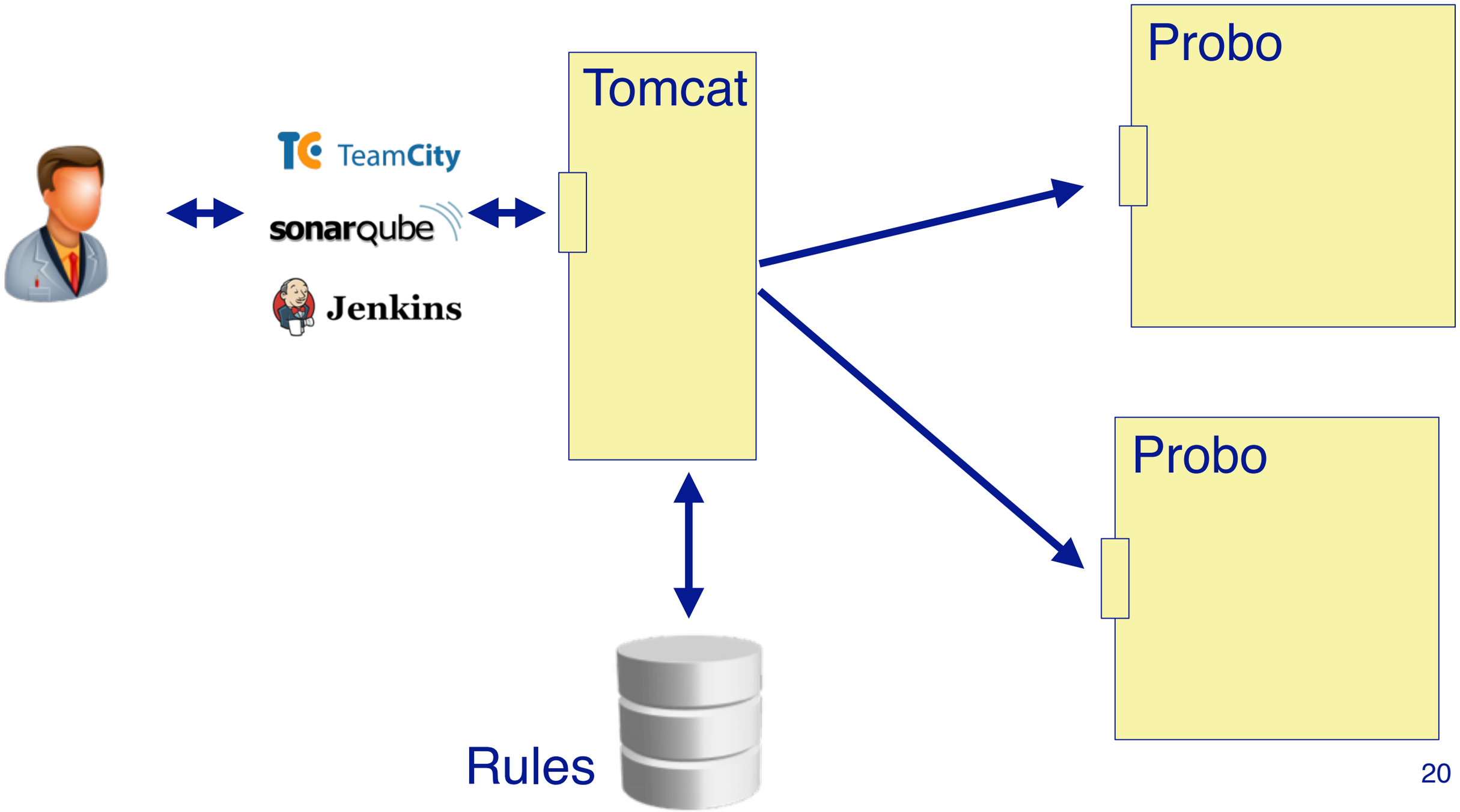


Architectural design: WHO

- identify actors (human/not human)
- what kind of information do they need/produce?

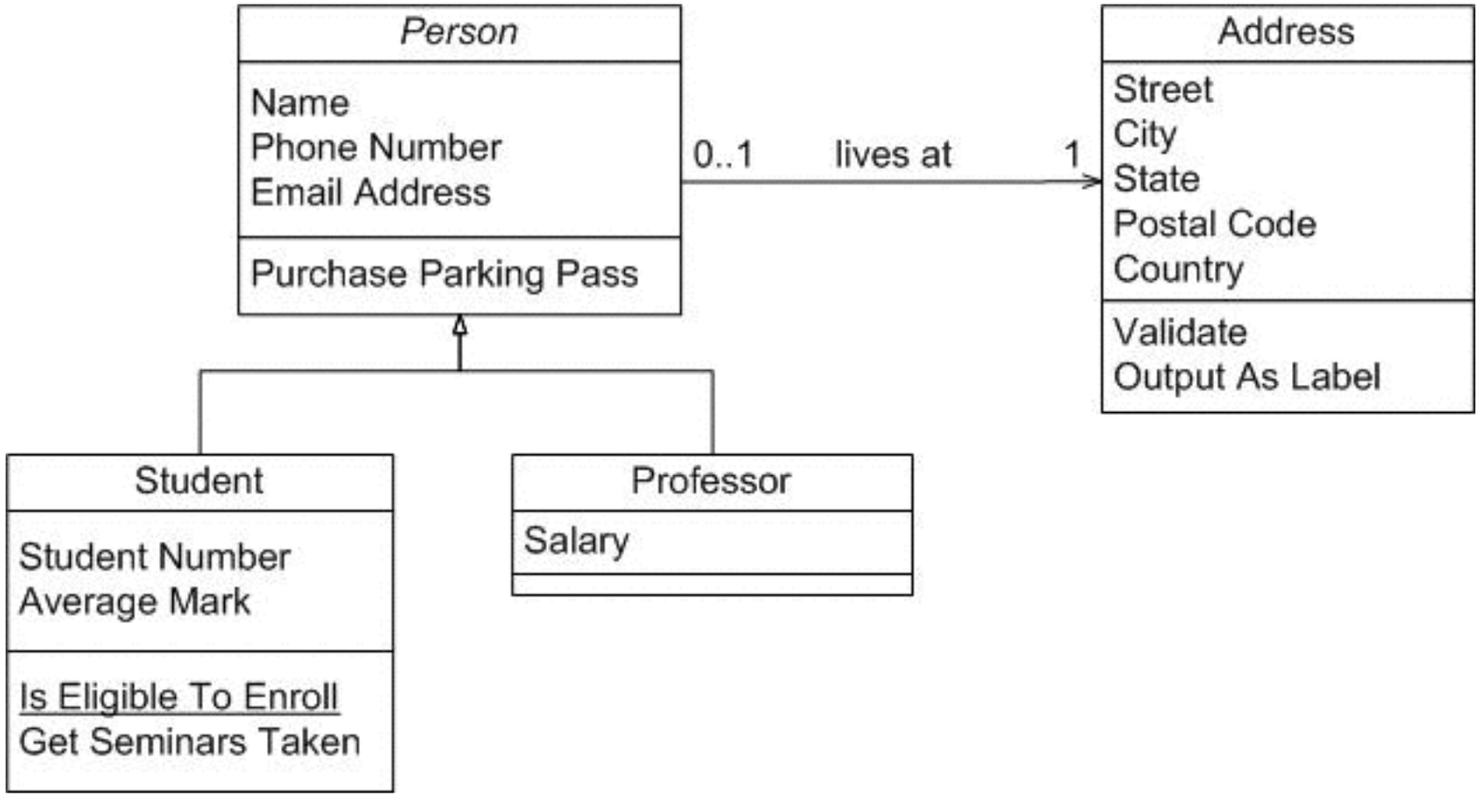


Architectural design: WHO

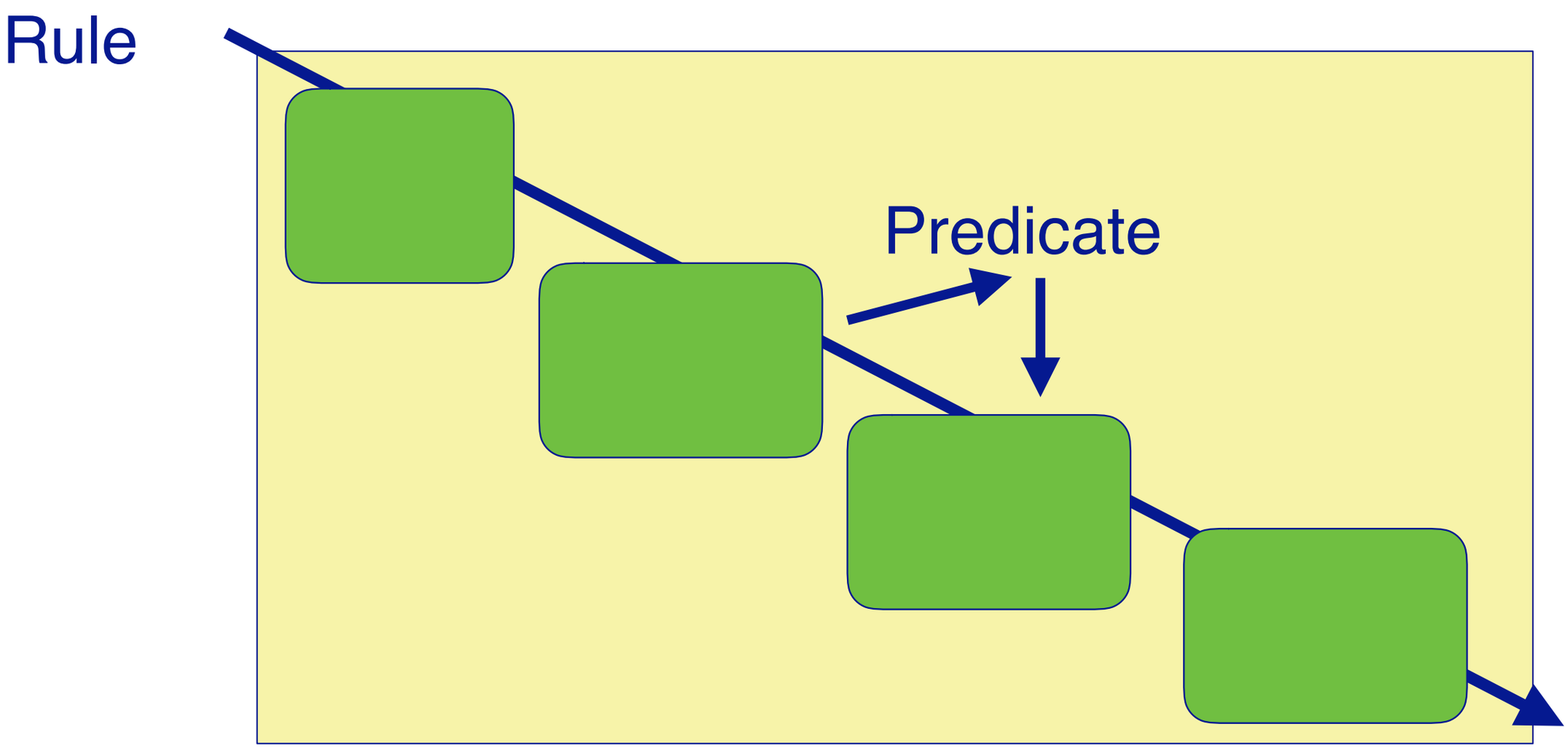


Architectural design: WHAT

- domain abstraction model

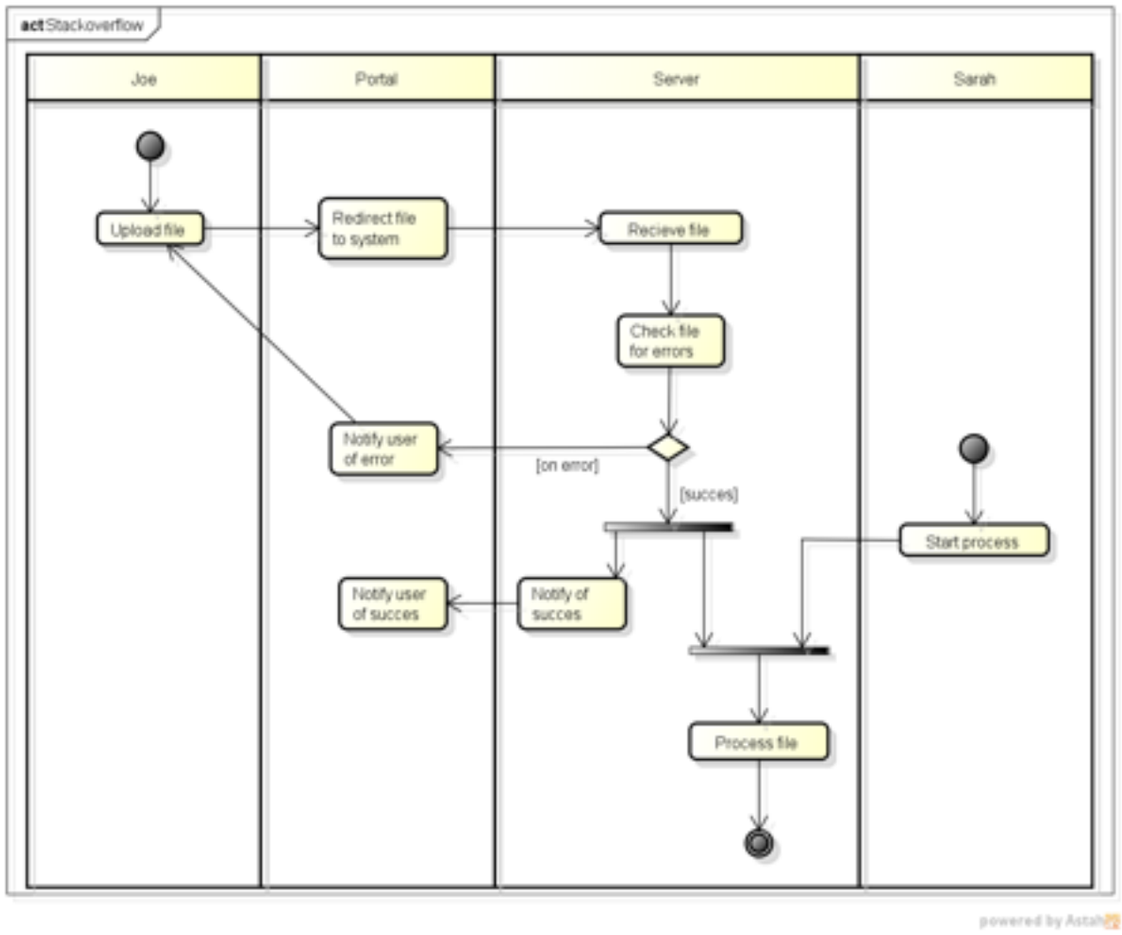


Architectural design: WHAT

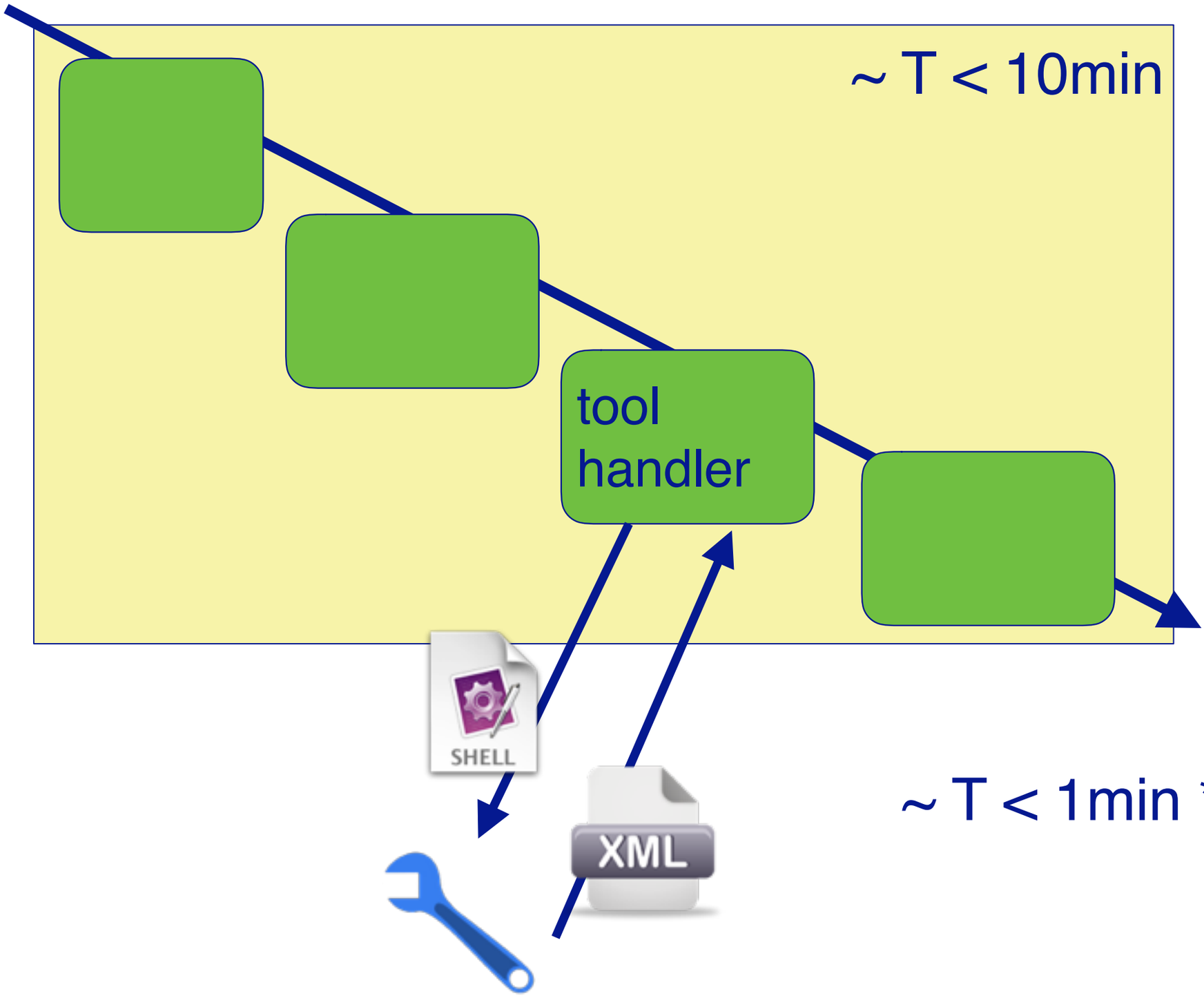


Architectural design: HOW / WHEN

- How/When is information generated, processed and transmitted (activities and information flows)



Architectural design: HOW / WHEN



Architectural design: WHERE

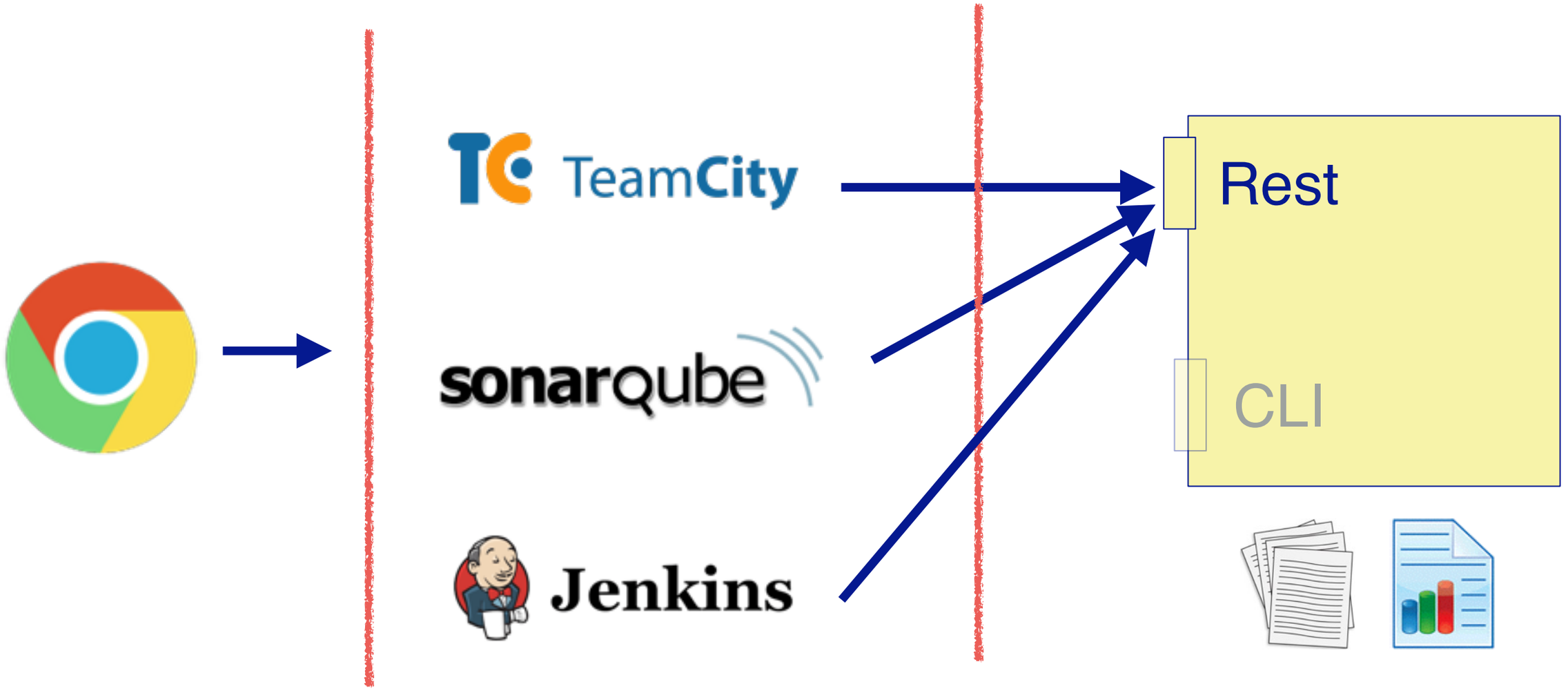
- Where actors, sources and sinks are physically and logically located

- tech. infrastructure
- network topology
- …

Architectural design: WHO / WHERE

Reporting clients

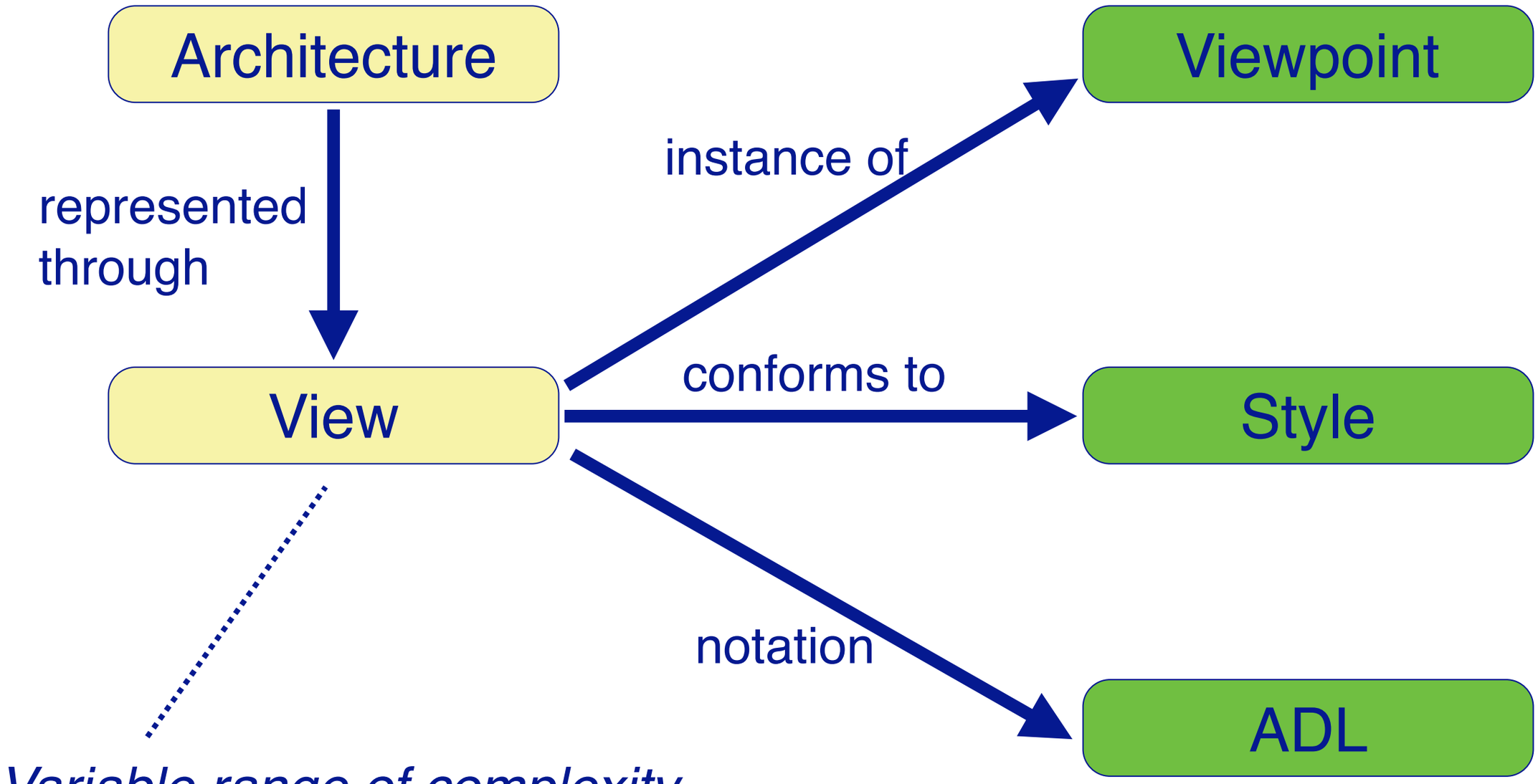
Analysis server



HTTP
HTML

HTTP
Json/XML

Describing Software Architecture

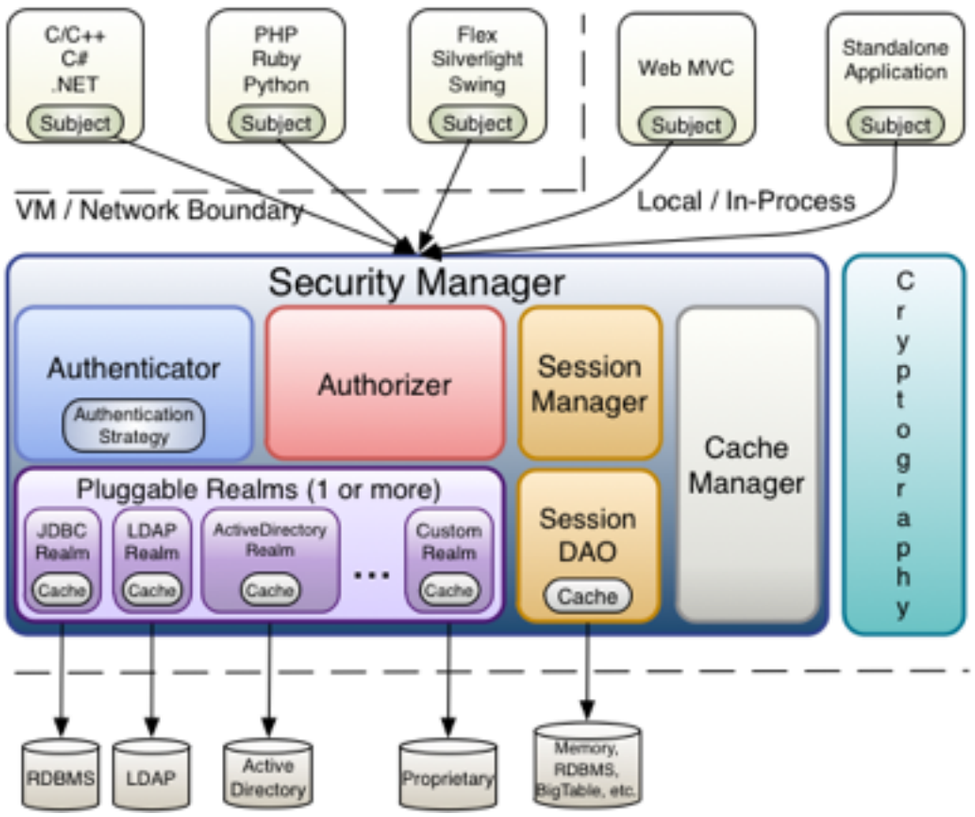


Architectural Viewpoints

Run-time	How are responsibilities distributed amongst run-time entities?
Process	How do processes communicate and synchronize?
Dataflow	How do data and tasks flow through the system?
Deployment	How are components physically distributed?
Module	How is the software partitioned into modules?
Build	What dependencies exist between modules?

How Architecture Is Usually Specified

> "Use a *3-tier client-server architecture*: all business logic must be in the middle tier, presentation and dialogue on the client, and data services on the server; that way you can scale the application server processing independently of persistent store."



Jeff Bezos - 2002 Email

All teams will henceforth expose their data and functionality through **service interfaces**
Teams must **communicate exclusively through these interfaces** with each other.
It doesn't matter what technology they use.
There will be **no other form of inter-process communication** allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever.
Anyone who doesn't do this will be fired.
Thank you; have a nice day!

Architectural Description Languages
or how architecture could be specified...

Formal languages for representing and reasoning about software architecture.

Provide a **conceptual framework** and a concrete syntax for characterizing architectures.

Some are **executable**, or implemented in a general-purpose programming language.

Wright	underlying model is CSP, focuses on connectivity of concurrent components
Darwin	focuses on supporting distributed applications. Components are single-threaded active objects

ADL example

```
process implementation process1.basic
   subcomponents
      A: thread t1.basic; B: thread t2.basic; C: thread t2.basic;
   connections
      cn1: data port signal -> A.p1;
      cn2: data port A.p2 -> B.p1;
      cn3: data port B.p2 -> result1;
      cn4: data port A.p2 -> C.p1;
      cn5: data port C.p2 -> result2;
      cn6: data port A.p3 -> status;
      cn7: event port init -> C.reset;
   flows
      f1: flow path signal->cn1->A.fs1->cn2->B.fs1->cn3->result1;
      f2: flow path signal->cn1->A.fs1->cn4->C.fs1->cn5->result2;
      f3: flow sink init->cn7->C.fs2;
      f4: flow source A.fs2->cn6->status;
end process1.basic;
```

```
system implementation Software.Basic
subcomponents
   Sampler_A : process Collect_Samples {
      Source_Text => ("collect_samples.ads", "collect_samples.adb") ;
      Period => 50 ms ;
} ;
end Software.Basic ;
```

Roadmap

- What is Software Architecture?
- **Cohesion and Coupling**
- Architectural styles
- UML diagrams for architectures

Sub-systems, Modules and Components

- A sub-system is a system in its own right whose operation is *independent* of the services provided by other sub-systems.

- A module is a system component that *provides services* to other modules but would not normally be considered as a separate system.

- A component is an *independently deliverable unit* of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.

Cohesion

Cohesion is a measure of *how well the parts of a component "belong together".*

> Cohesion is weak if elements are bundled simply because they perform similar or related functions (e.g., `java.lang.Math`).

> Cohesion is strong if all parts are needed for the functioning of other parts (e.g. `java.lang.String`).

—Strong cohesion *promotes maintainability* and adaptability by *limiting the scope of changes* to small numbers of components.

There are many definitions and interpretations of cohesion.

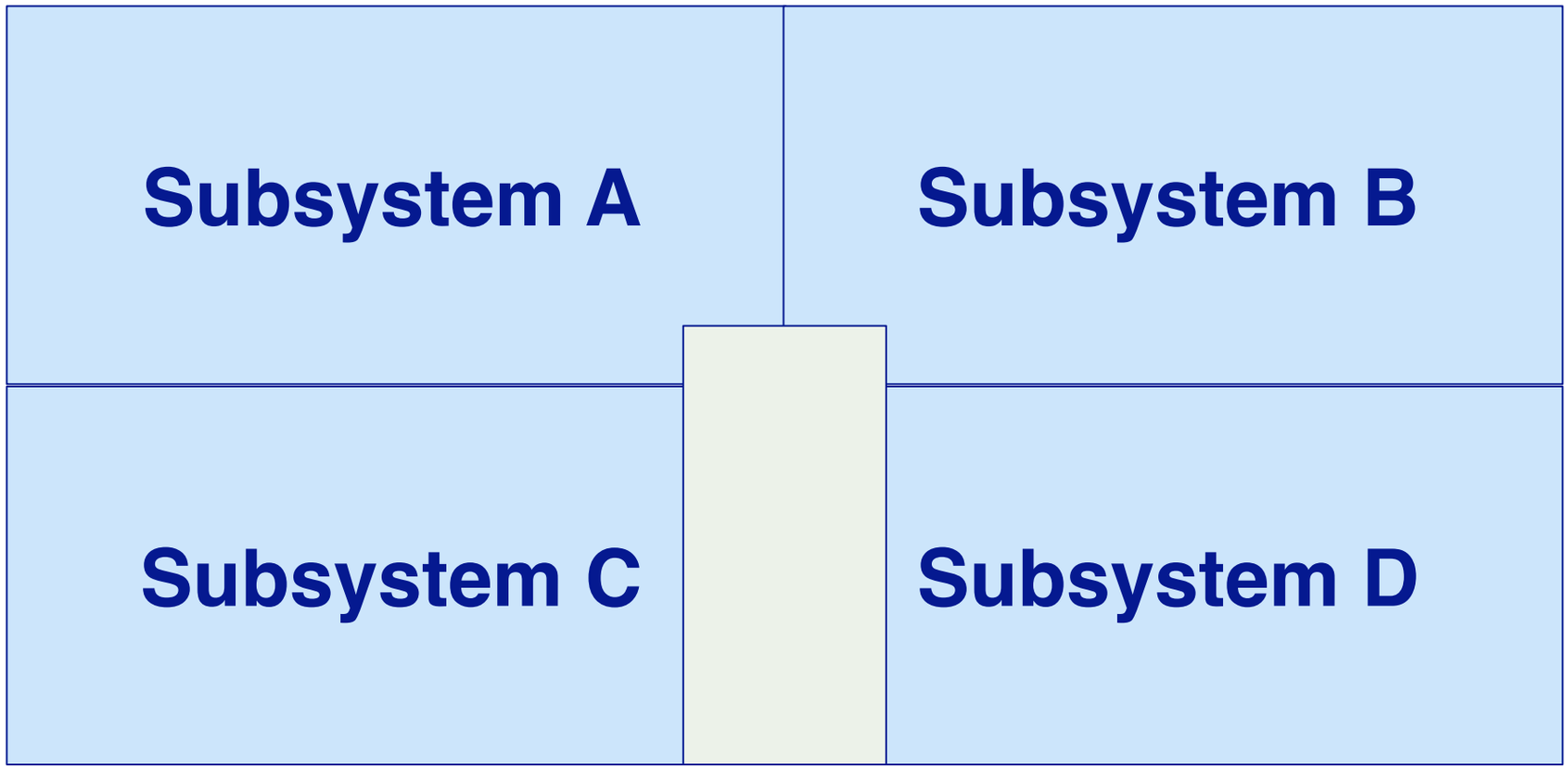
Most attempts to formally define it are inadequate!

Coupling

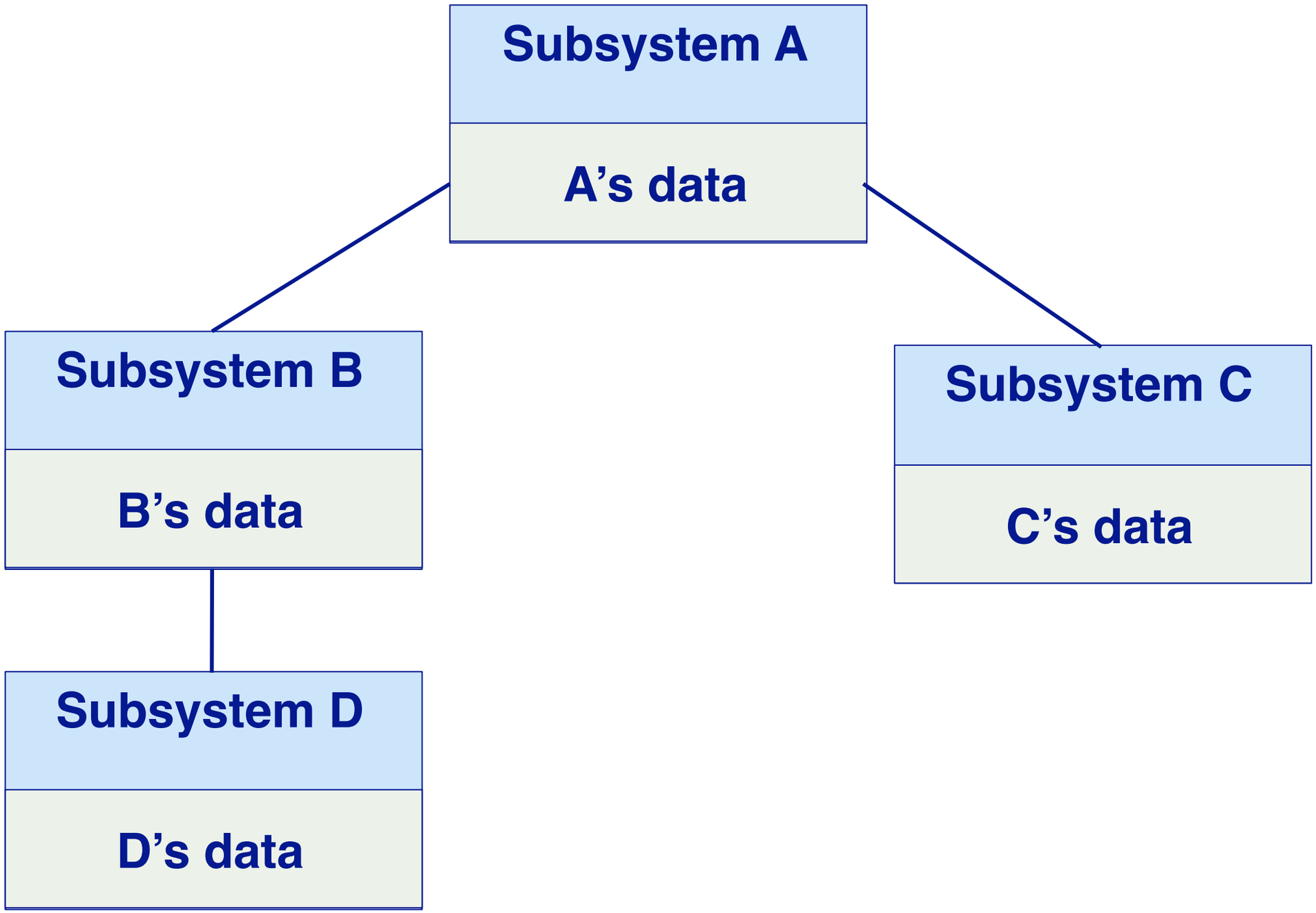
Coupling is a measure of the *strength of the interconnections* between system components.

- Coupling is tight between components if they depend heavily on one another, (e.g., there is a lot of communication between them).
- Coupling is loose if there are few dependencies between components.
  - —Loose coupling *promotes maintainability* and adaptability since *changes in one component are less likely to affect others.*
  - —*Loose coupling increases the chances of reusability.*

Tight Coupling



Loose Coupling



Roadmap

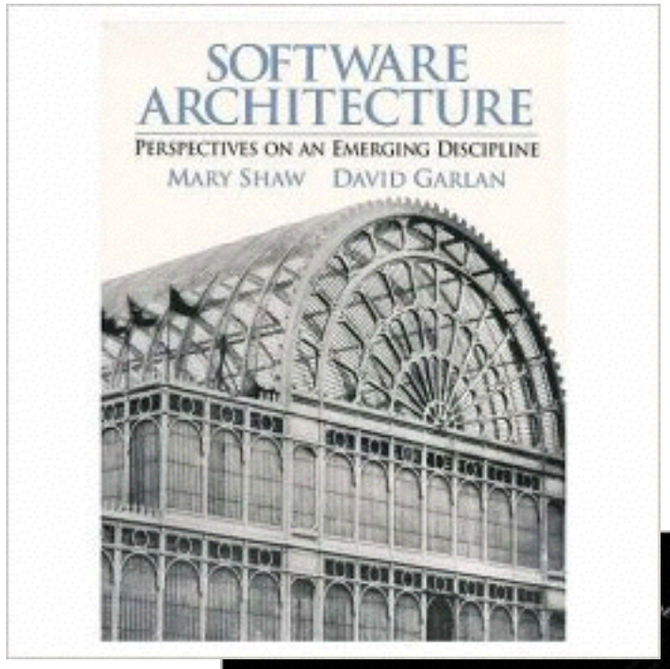
- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - —Structure
  - —Shared Data
  - —Communication
  - —Distribution
- UML diagrams for architectures

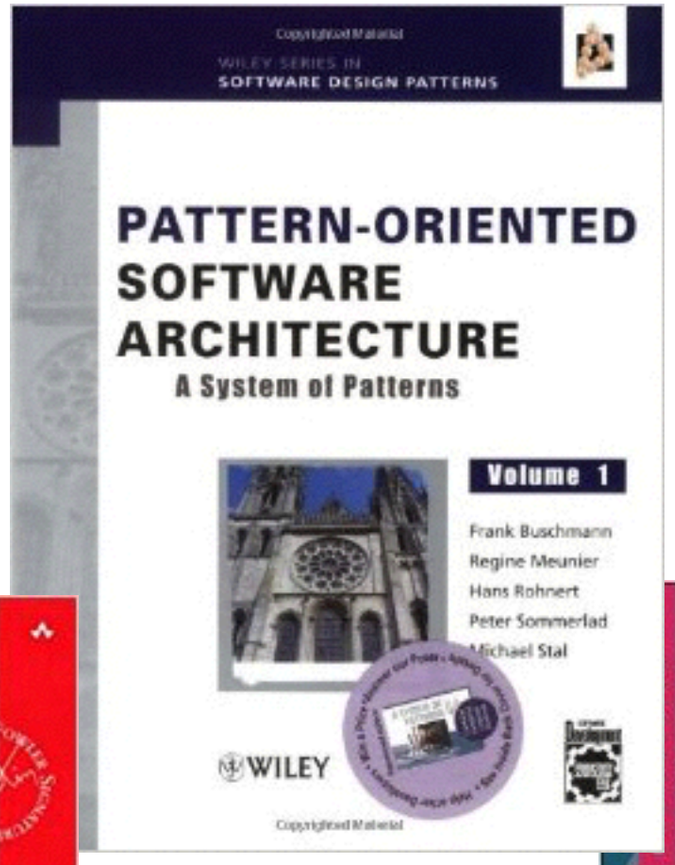
Architectural Styles

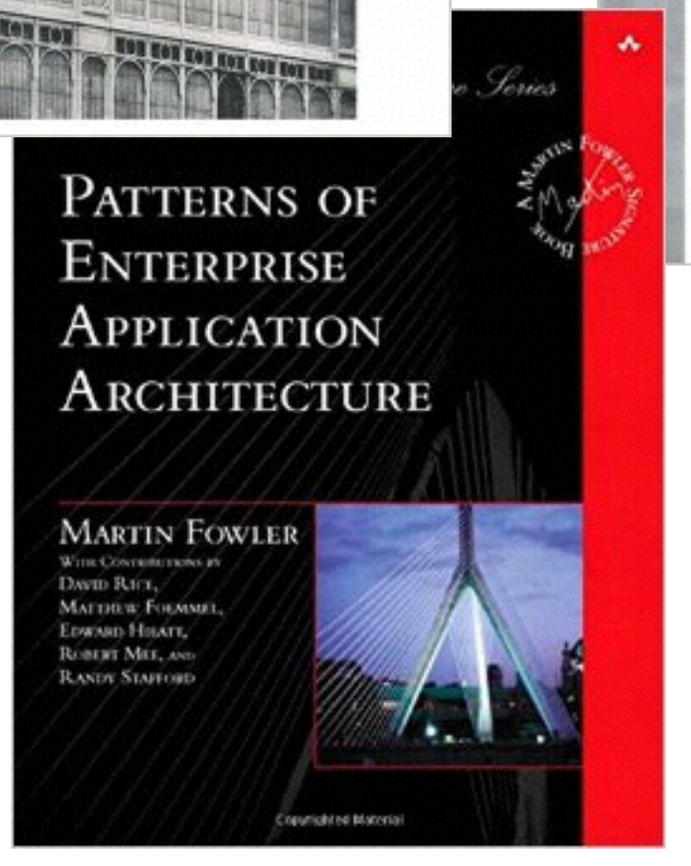
*An architectural style defines a **family of systems** in terms of a pattern of **structural organization**. More specifically, an architectural style defines a vocabulary of components and connector **types**, and a set of constraints on how they can be combined.*

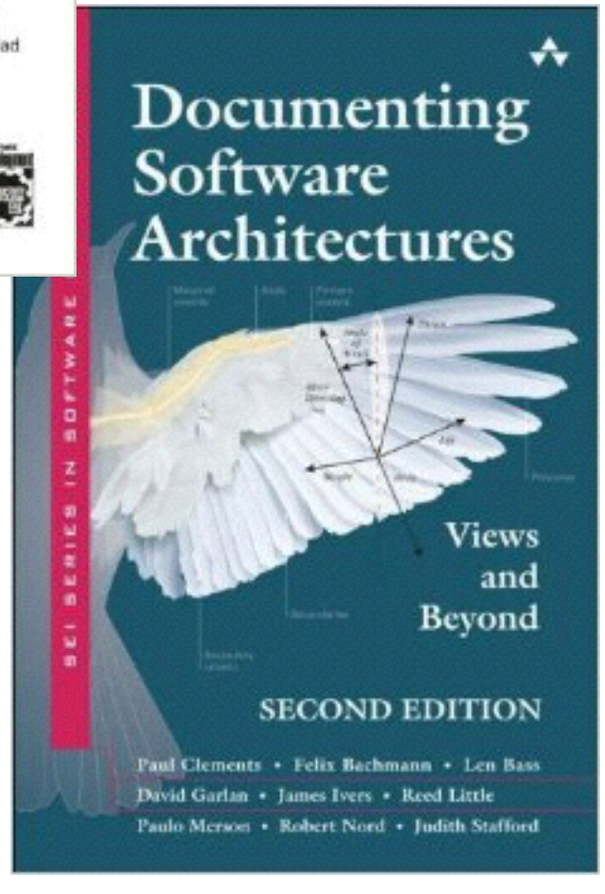
— Shaw and Garlan

Architectural Style “Catalogues”









Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - **Structure**
  - Data flow
  - Call-return
  - Event-driven
- UML diagrams for architectures

"Big Ball of Mud"

The system is organized as a single element. No modularity. No constraints.

Example:

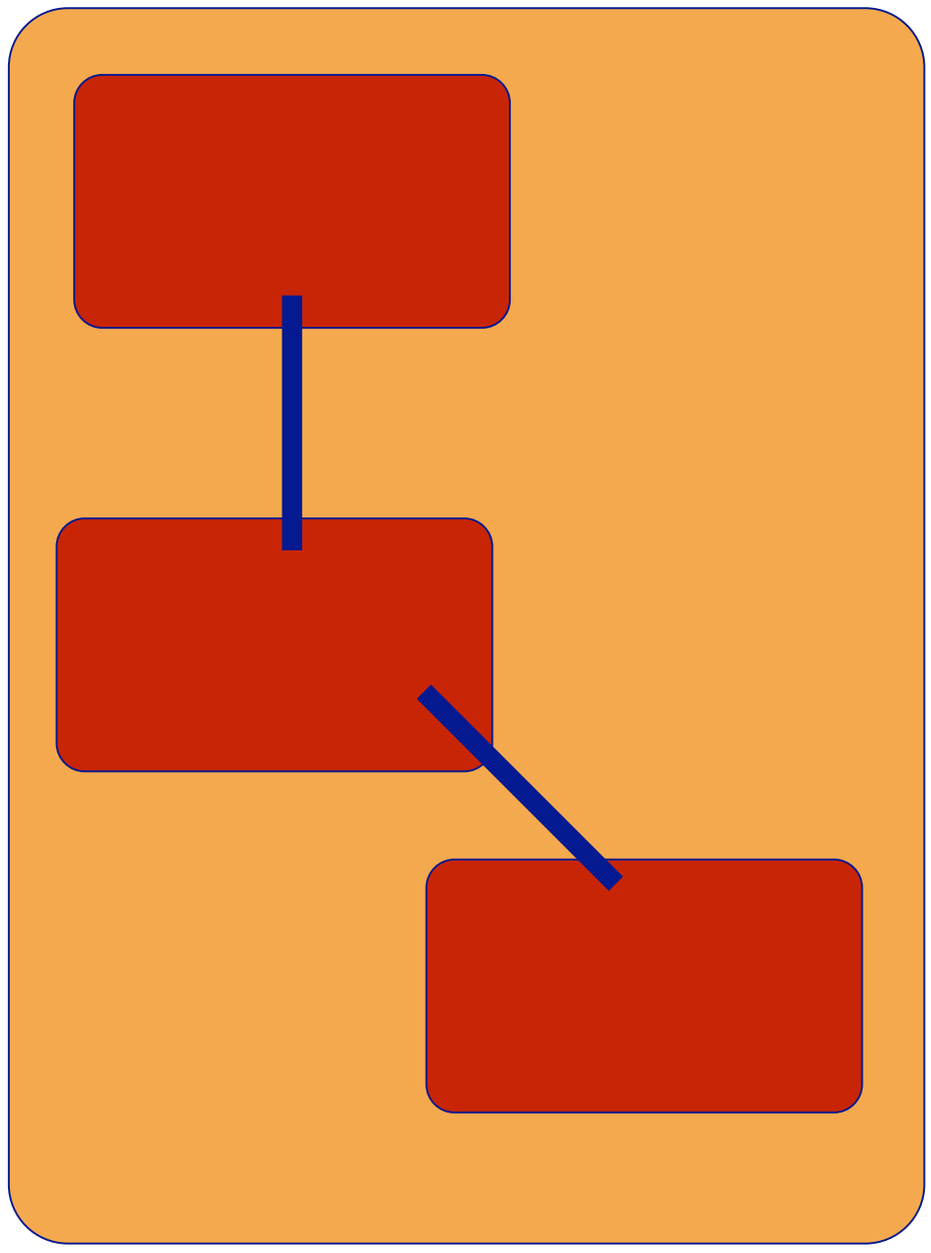
Mainframe application

Qualities:

- Poor Extensibility
- Poor Maintainability

Component-based

Components have well defined interfaces and communicate via connectors linking their interfaces



Example:

Modules, WebServices, ..

Qualities:

+ Separation of concerns
+ Reuse

Layered

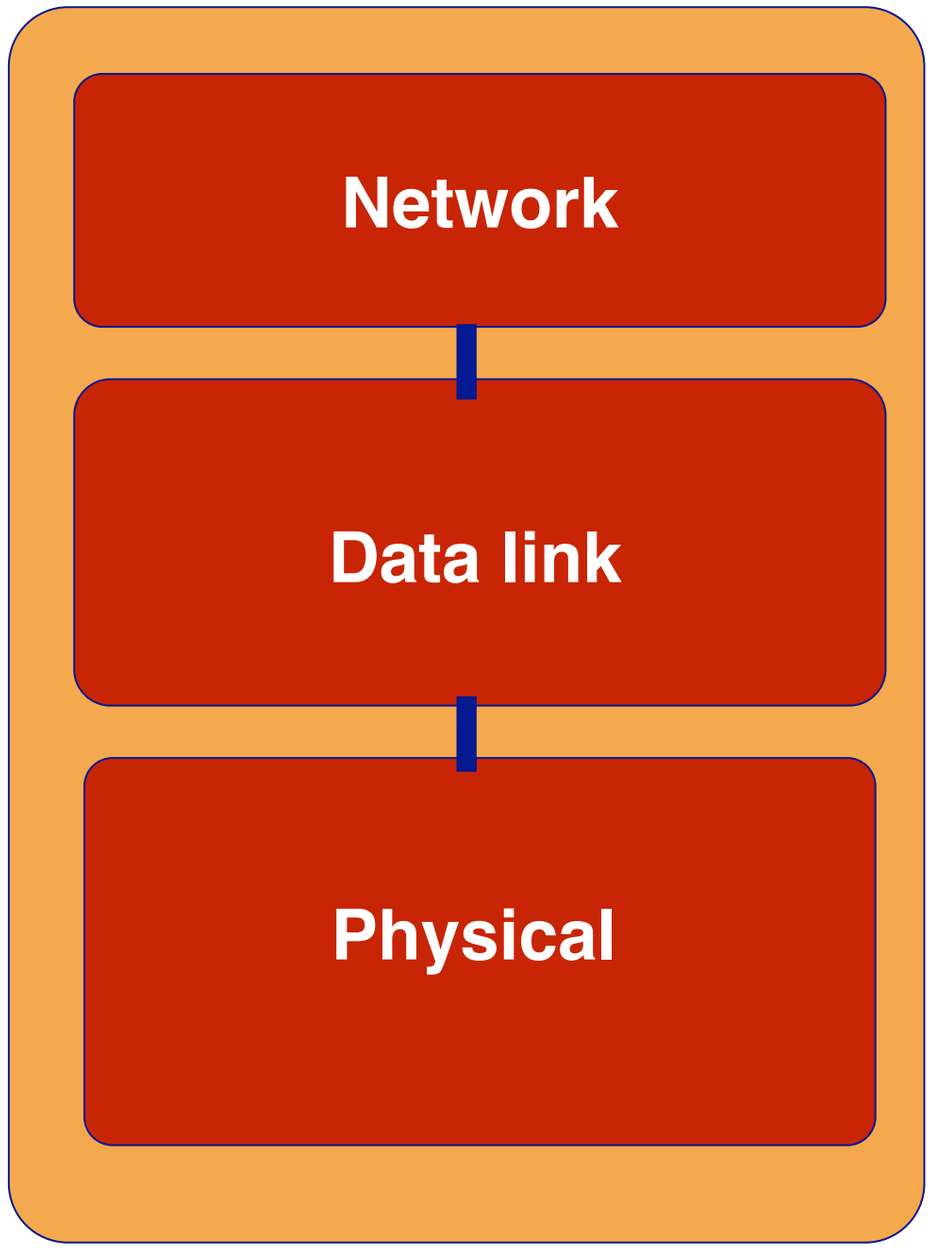
The elements in each layer communicate only with entities that are in the layers above and below

Example:

OSI, web-apps (MVC)

Qualities:

+ Exchangeability
+ Limited error propagation
- Performance overhead



Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - —Structure
  - —**Data flow**
  - —Call-return
  - —Event-driven
- UML diagrams for architectures

Pipes & Filters

```
ls -l |
grep "\.txt$" |
sort -d
```

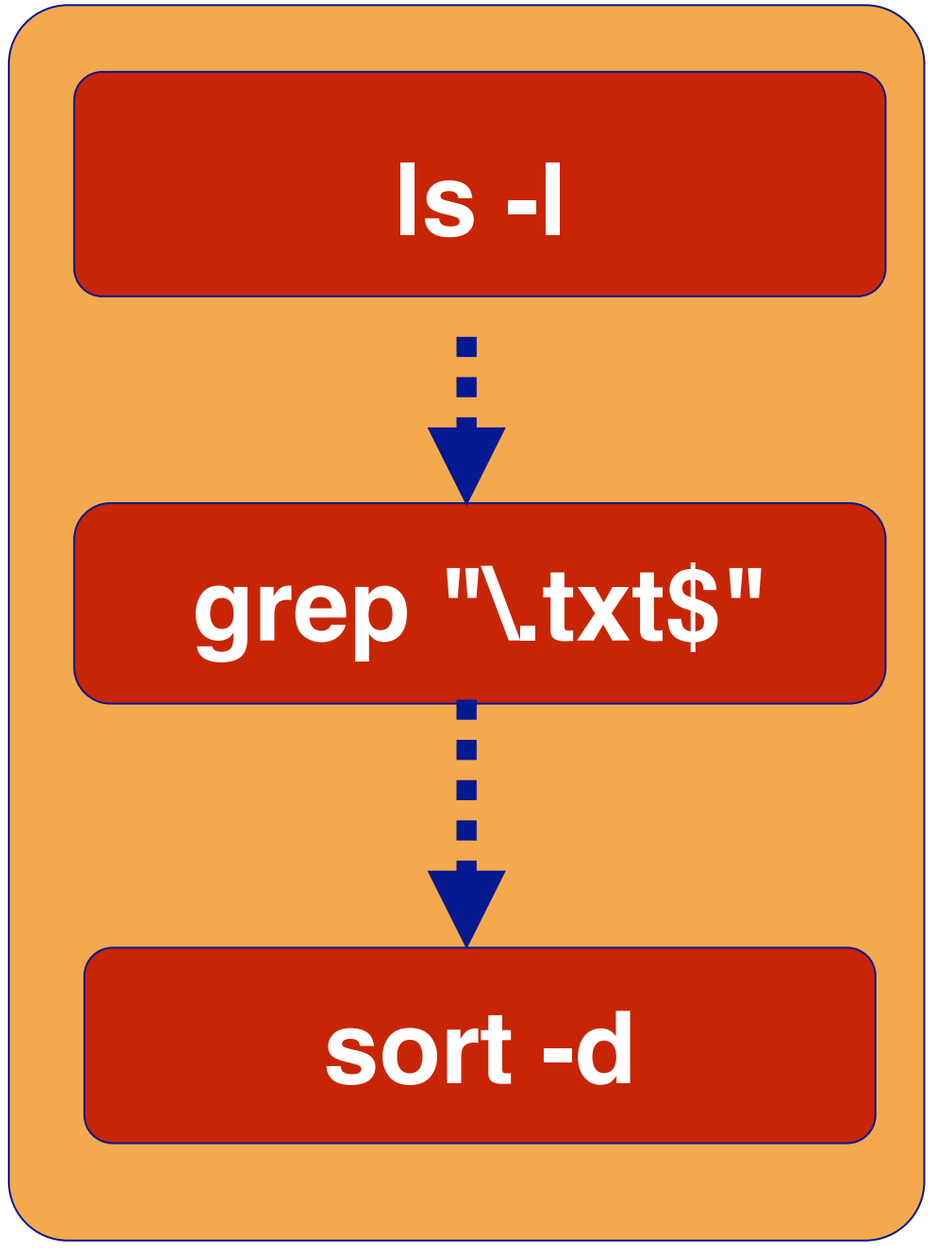
One element reading data at one end and writing it at the other end. Pipelines do not have to be linear.

Example:

Image processing, Compilers

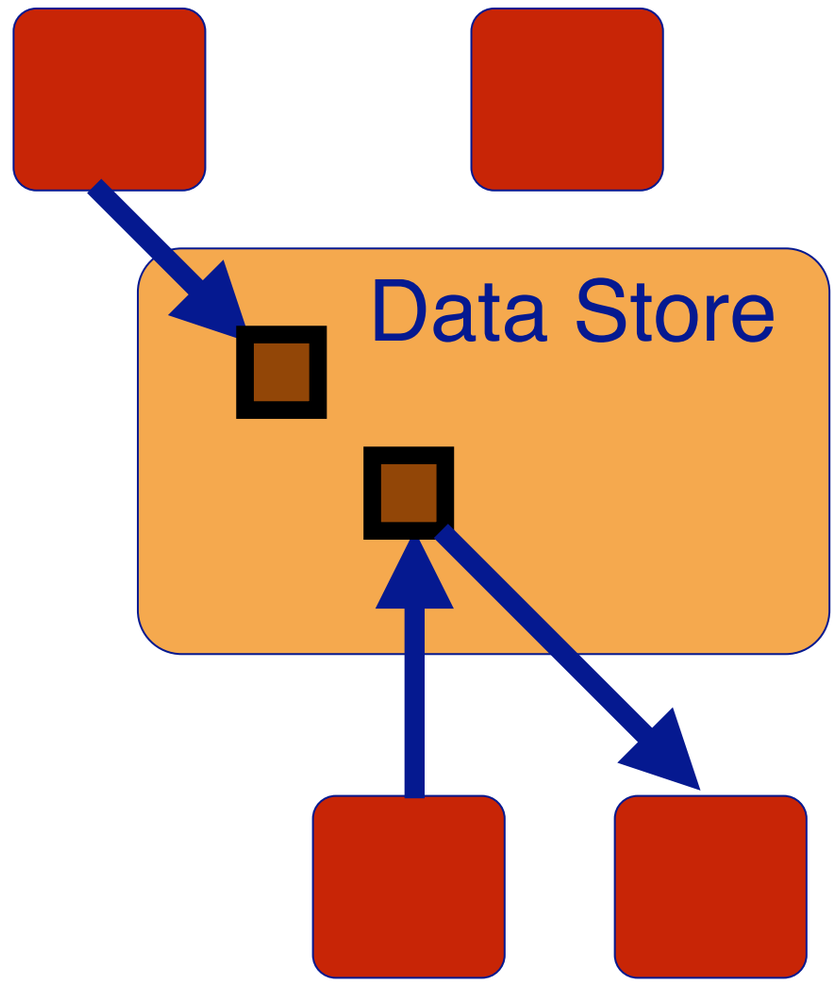
Qualities:

+ Flexibility by recombination
- Performance (state/data sharing)
- Error handling



Blackboard

Elements share, post, update data written on the blackboard in order to collectively work on a solution to the problem.



Example:

Sensor network, distributed computing

Qualities:

- Difficult to test / Lack of control
- Semantic coupling

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - —Structure
  - —Data flow
  - —**Call-return**
  - —Event-driven
- UML diagrams for architectures

Client-server

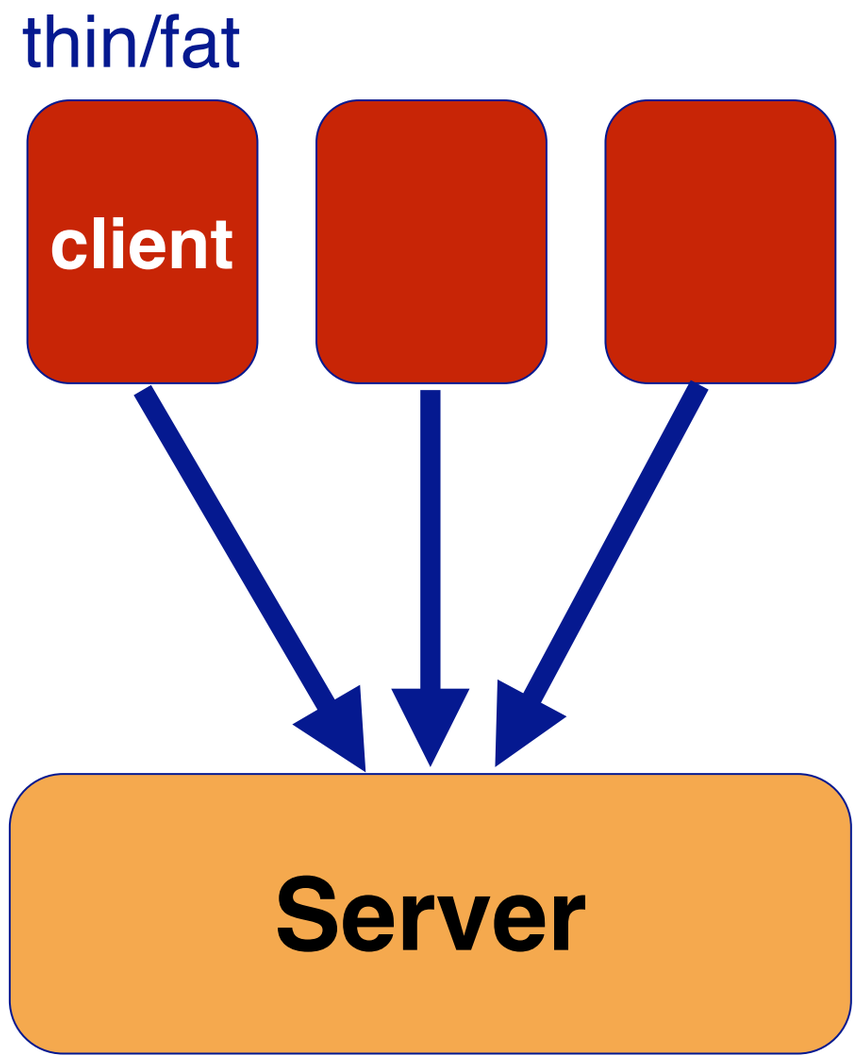
One or more clients send requests to the server, which processes them before sending them back a response

Example:

Web browser, email reader, DB-app

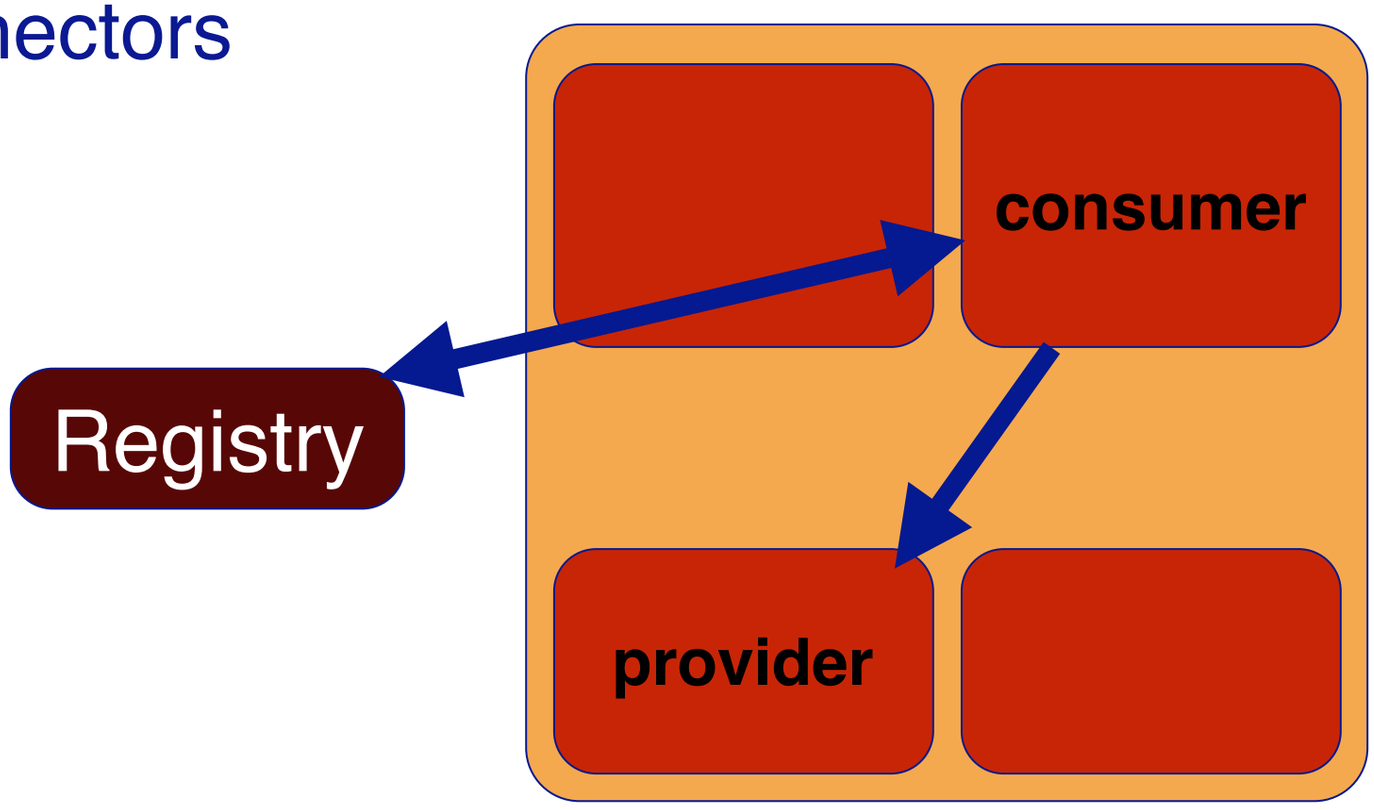
Qualities:

- Communication overhead
+ Cheap infrastructure
- Single point of failure



Service oriented

Distributed components
have well defined interfaces and
communicate via specific connectors
linking their interfaces.



Example:

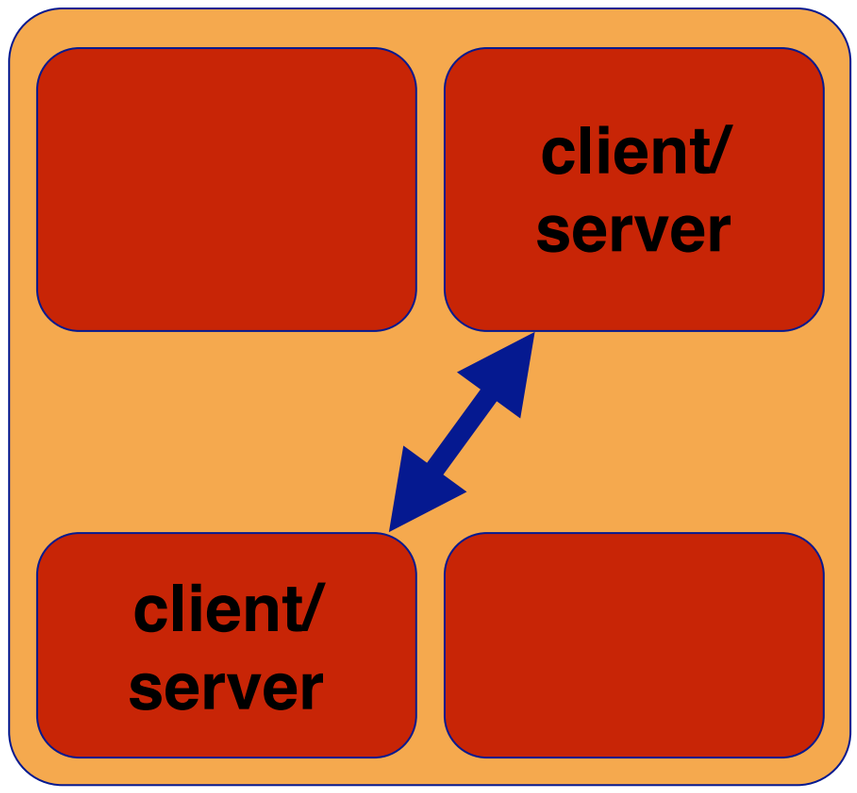
REST, SOAP

Qualities:

+ Loose structural coupling
+ Technology independent

Peer to peer

There is no central server as all elements can both act as client and as server and send one another requests and response messages



Example:

Torrent

Qualities:

+ Adaptability, Scalability

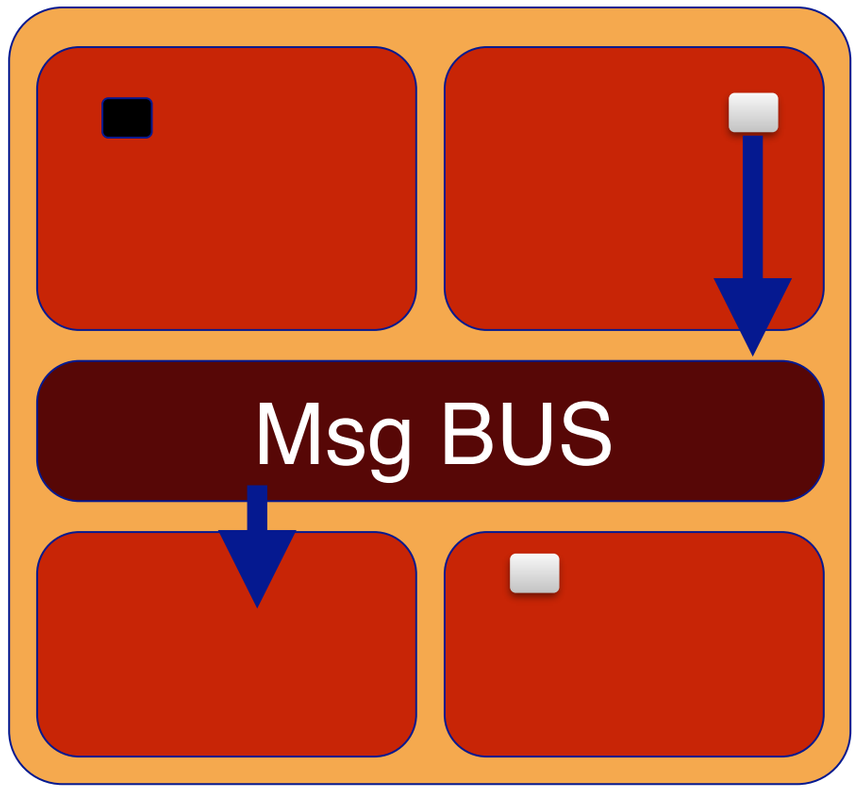
- Lack of control

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- **Architectural styles**
  - —Structure
  - —Data flow
  - —Call-return
  - —**Event-driven**
- UML diagrams for architectures

Publish-subscribe

Event Driven system where elements are coupled by subscriptions and receive notifications when some interesting event happens



Example:

Message broadcasting, GUI

Qualities:

- Semantic coupling

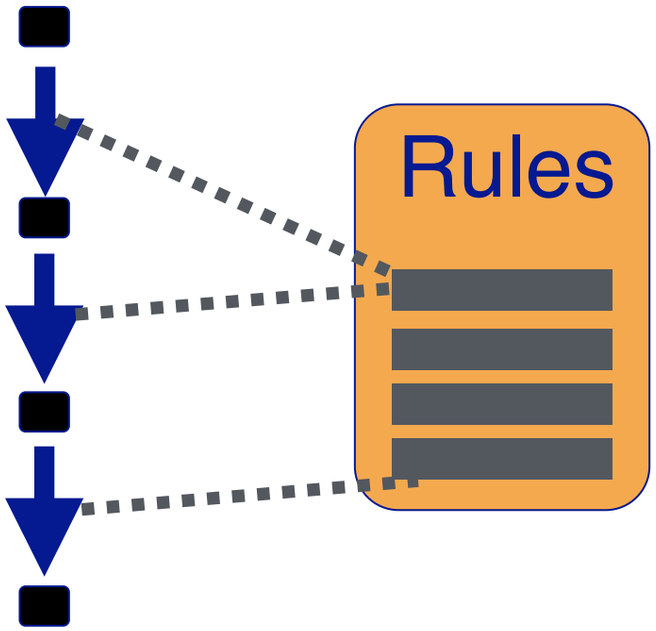
+ Loose structural coupling

Rule-based

attempts to derive execution instructions
from a starting set of data and rules

Example:

Financial system, Natural language



Qualities:

- Difficult to test / Lack of control

+ Convenient for certain domains

Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles
- **UML diagrams for architectures**

UML support: Package Diagram

Decompose system into *packages* (containing any other UML element, incl. packages)

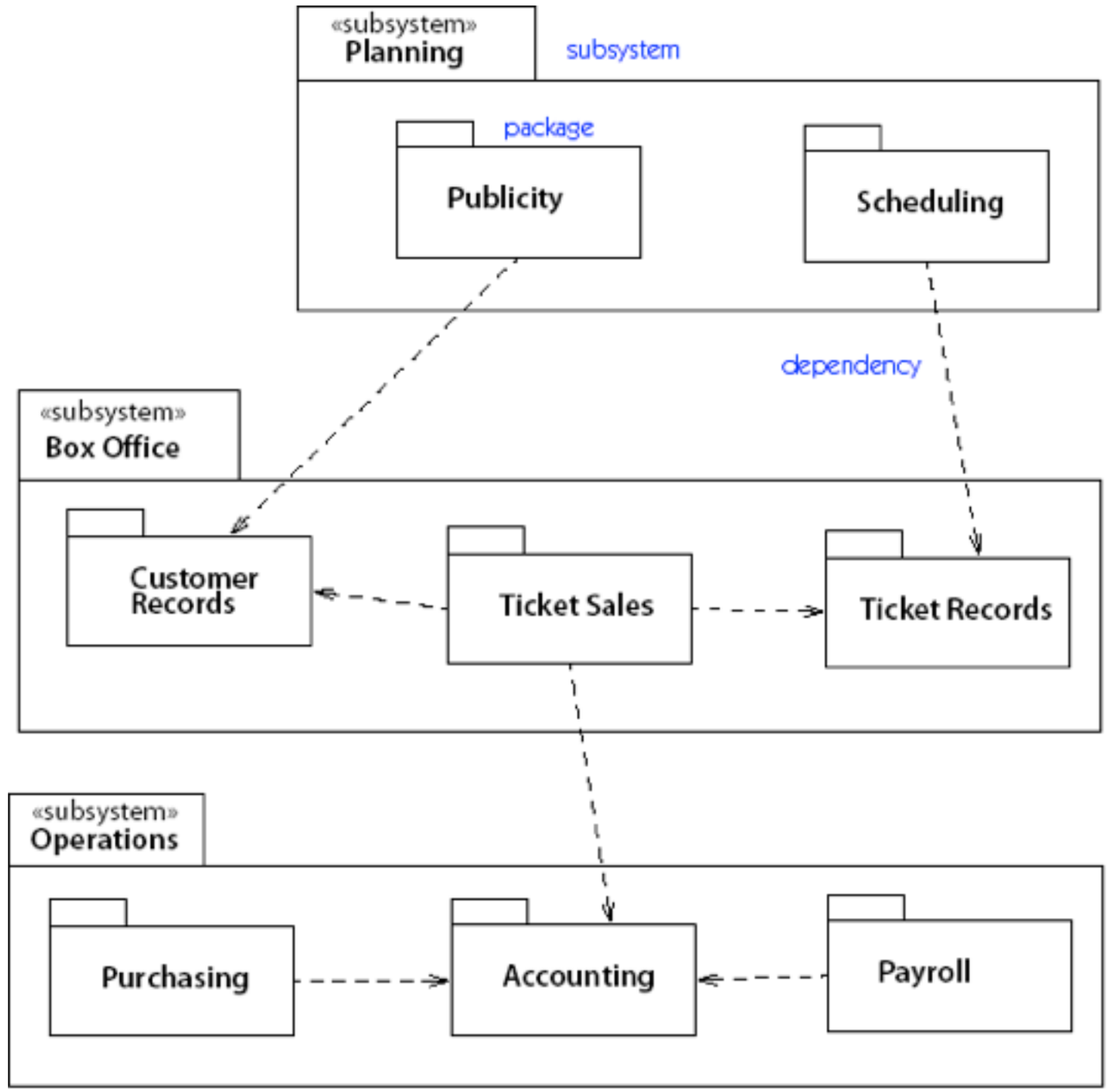


Figure 3-10. *Packages*

UML support: Deployment Diagram

Physical layout of run-time components on hardware nodes.

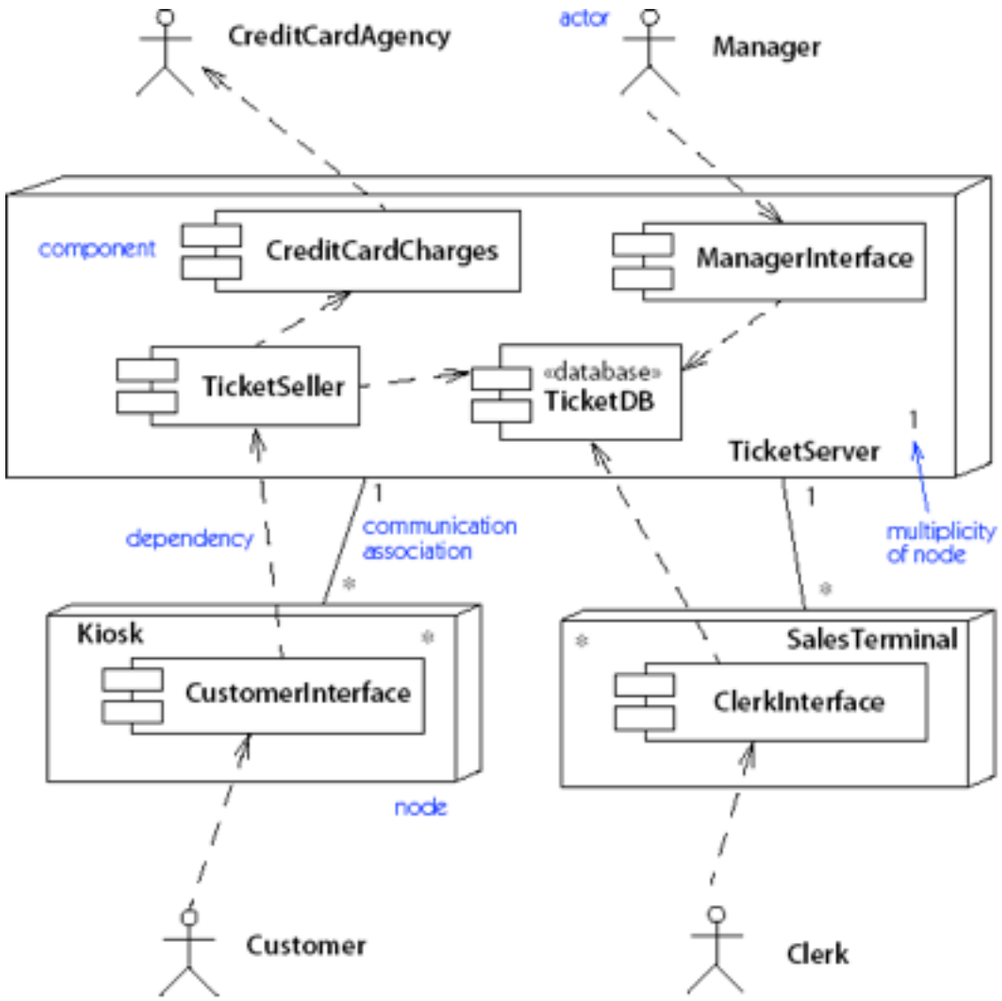


Figure 3-8. *Deployment diagram (descriptor level)*

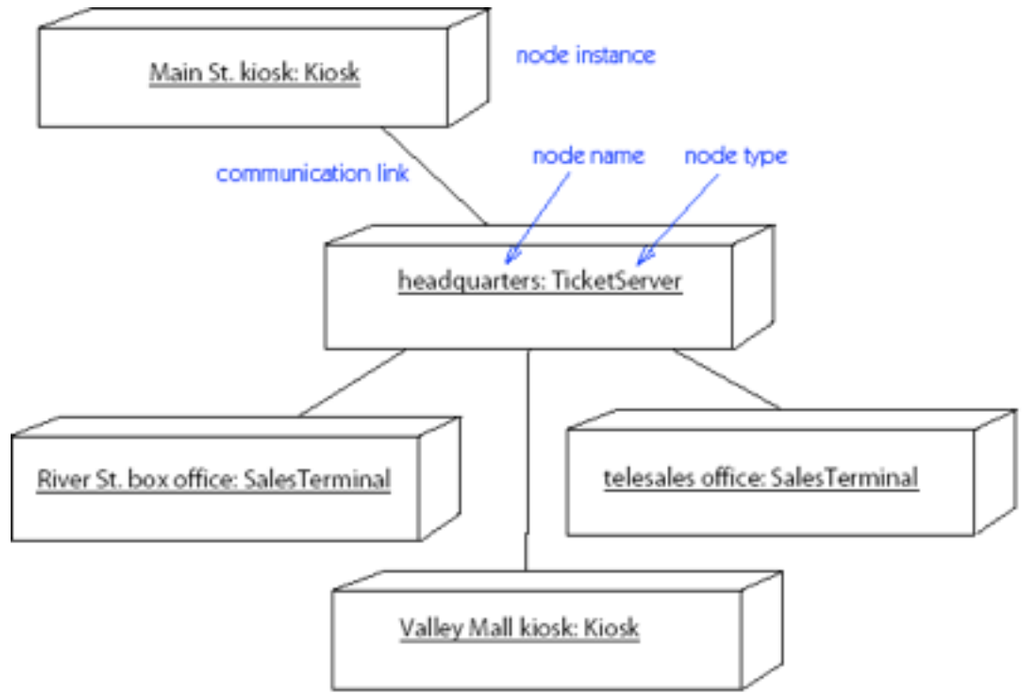


Figure 3-9. *Deployment diagram (instance level)*

Sources

- *Software Engineering*, I. Sommerville, 7th Edn., 2004.
- *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999
- *Pattern-Oriented Software Architecture — A System of Patterns*, F. Buschmann, et al., John Wiley, 1996
- *Software Architecture: Perspectives on an Emerging Discipline*, M. Shaw, D. Garlan, Prentice-Hall, 1996

What you should know!

- What is software architecture
- What is the difference between non-architectural and architectural design
- What are architectural viewpoints and architectural styles
- What are ADLs, components and connectors
- Advantages and disadvantages of classical architectural styles

Can you answer the following questions?

- What kind of architectural styles are in your project?
- What are the characteristics of a multi tier architecture?
- How can you reduce coupling between software layers?
- How would you implement a dataflow architecture in Java?

Exercise

- Customers can use the ATM from any bank to withdraw cash from their bank account.
- Each bank has its own system to deal with accounts (checking access rights, balance, etc…)
- Each ATM keeps a list of the transactions performed, so that banks can keep track of the amount of money they owe each other
- At the end of each day, each ATM sends a report to the banks involved in each transaction.

- Bank A customer goes to an ATM of a bank different from his/her own bank to withdraw cash. The ATM machine (locally) verifies the correspondence between customer's card and PIN. The customer asks for cash, the ATM connect the bank system, check the availability on customer's account, log the operation and give cash.