CSE 403: Software Engineering, Winter 2016

courses.cs.washington.edu/courses/cse403/16wi/

Software Architecture

Emina Torlak
emina@cs.washington.edu

Outline

- What is a software architecture?
- What does an architecture look like?
- What is a good architecture?
- Properties of architectures
- Example architectures

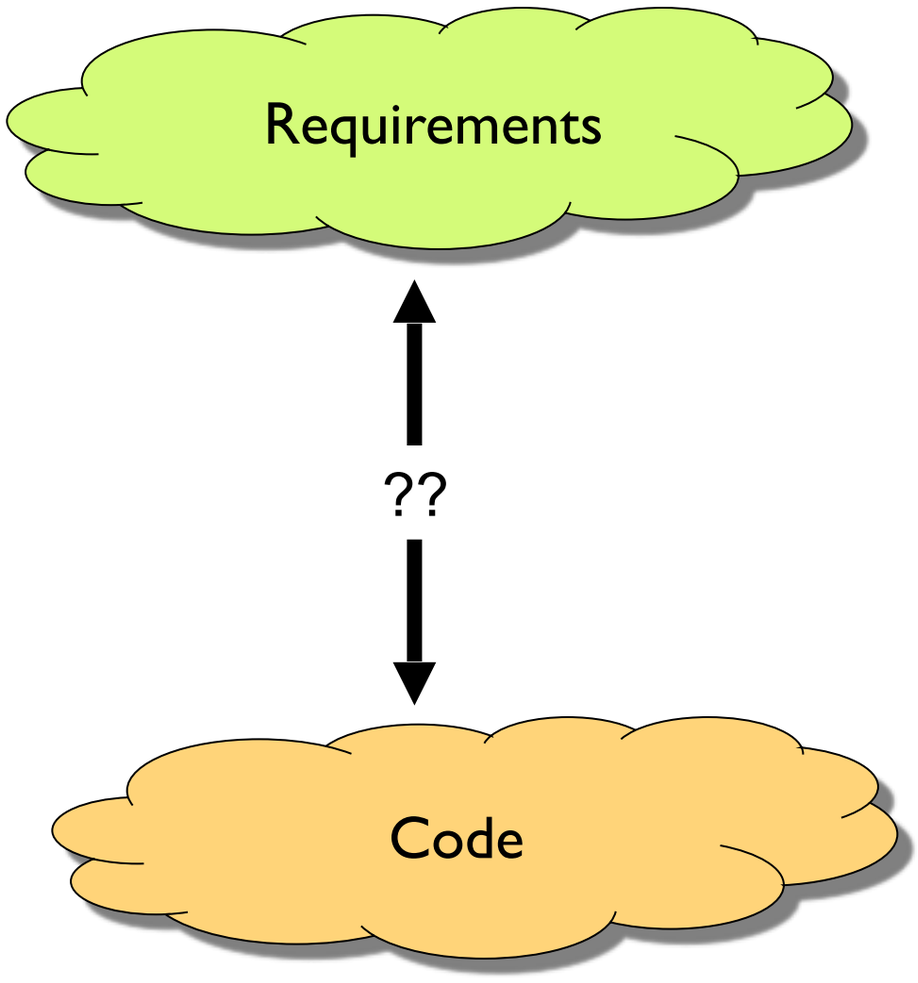
basics

software architecture: motivation & definition

Why software architecture?

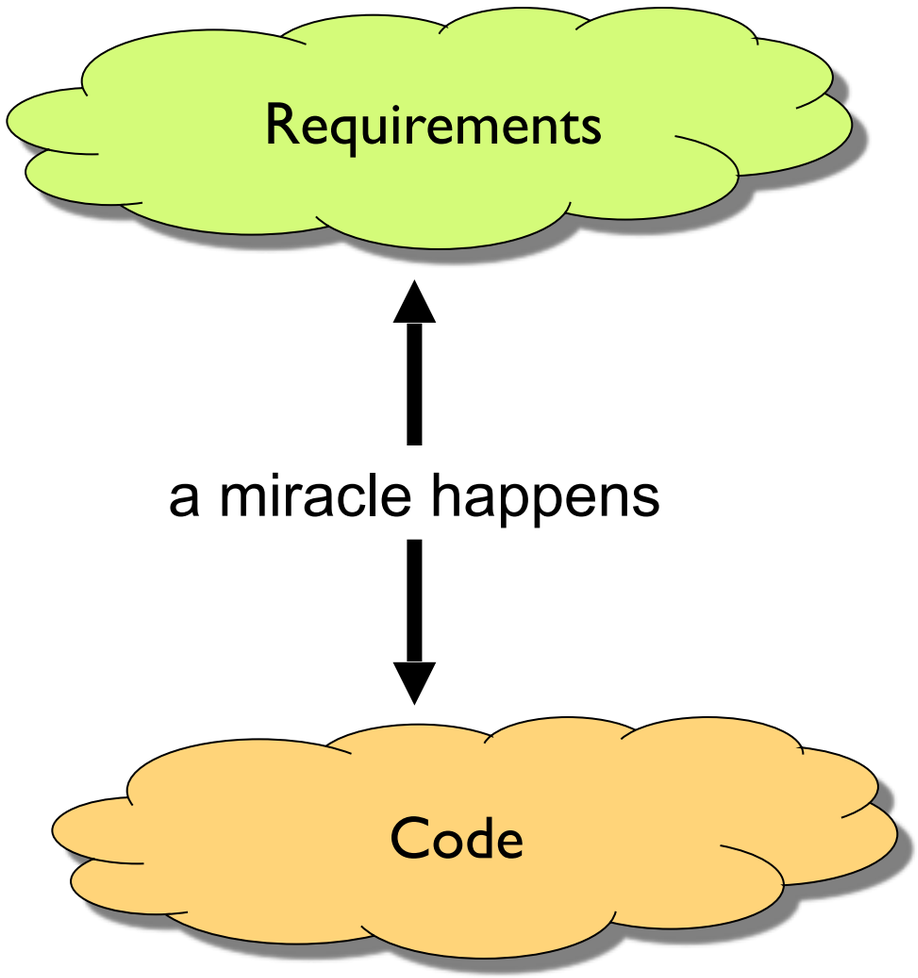
> "There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other is to make it so complicated that there are no obvious deficiencies."
>
> C.A.R. Hoare (1985)

The basic problem: from requirements to code

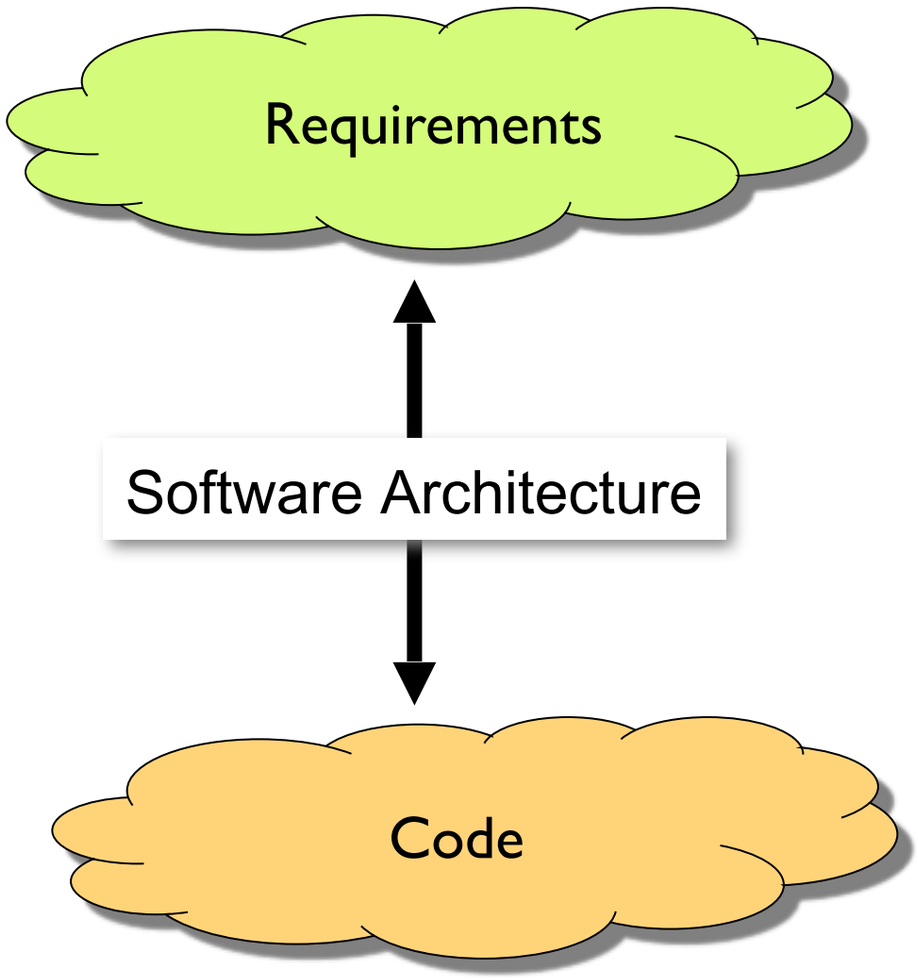


How to bridge the gap between requirements and code?

The basic problem: solve with inspiration



The basic problem: solve with engineering



Provides a high-level framework to build and evolve a software system.

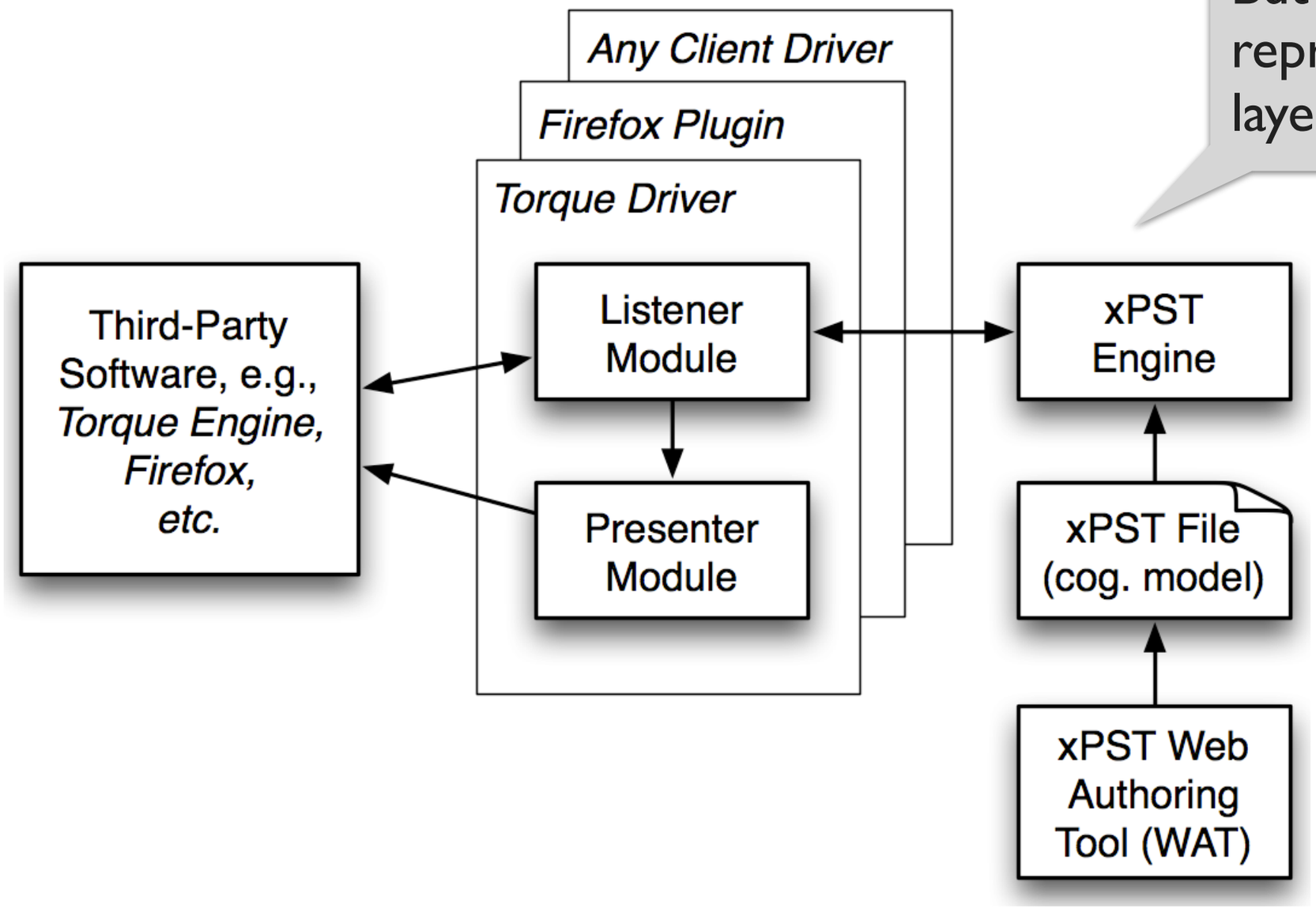
depict

what does an architecture look like?

Box and arrow diagrams

Very common and highly valuable.

But what does a box represent? An arrow? A layer? Adjacent boxes?



An architecture: components and connectors

- **Components** define the basic computations comprising the system and their behaviors
    - abstract data types, filters, etc.
- **Connectors** define the interconnections between components
    - procedure call, event announcement, asynchronous message sends, etc.
- The line between them may be fuzzy at times
    - A connector might (de)serialize data, but can it perform other, richer computations?

A standard notation for architecture: UML

- UML = unified modeling language
- A standardized way to describe (draw) architecture
    - Also implementation details such as subclassing, uses (dependences), and much more
- Widely used in industry
- Topic of next lecture

what is a good architecture?

A good architecture ...

- Satisfies functional and performance requirements
- Manages complexity
- Accommodates future change
- Is concerned with
    - reliability, safety, understandability, compatibility, robustness

A good architecture ...

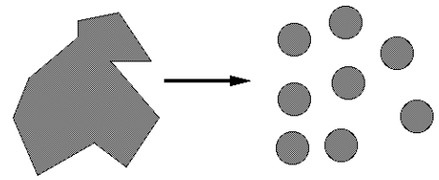
- Satisfies functional and performance requirements
- Manages complexity
- Accommodates future change
- Is concerned with
    - reliability, safety, understandability, compatibility, robustness

Leads to modularity and separation of concerns.

A modular architecture helps with ...

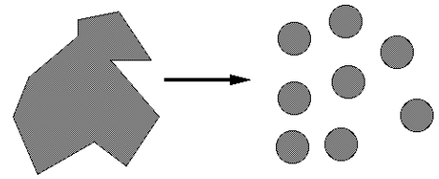
- **System understanding**: interactions between modules
- **Reuse**: high-level view shows opportunity for reuse
- **Construction**: breaks development down into work items
- **Evolution**: high-level view shows evolution path
- **Management**: helps understand work items and track progress
- **Communication**: provides vocabulary; a picture says 1000 words

You know your software is modular when it is ...



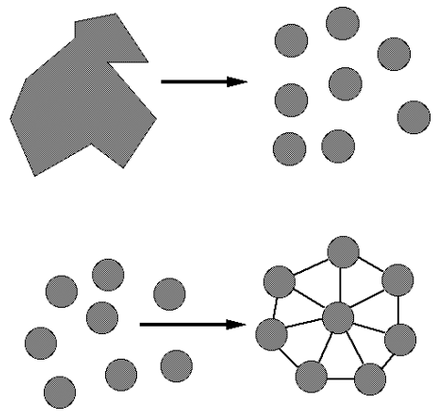
You know your software is modular when it is ...

- Decomposable
    - can be broken down into pieces



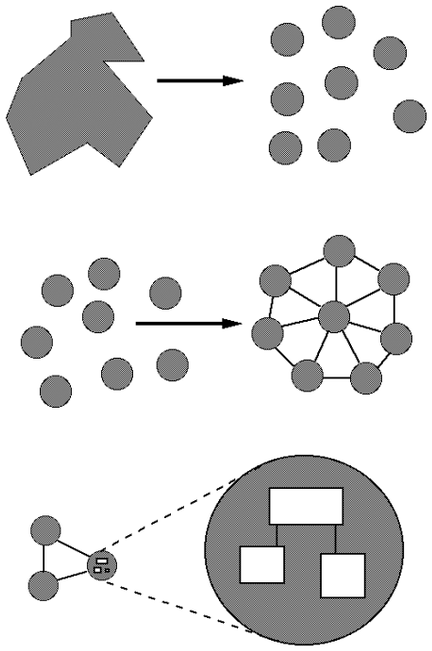
You know your software is modular when it is ...

- Decomposable
  - can be broken down into pieces
- Composable
  - pieces are useful and can be combined



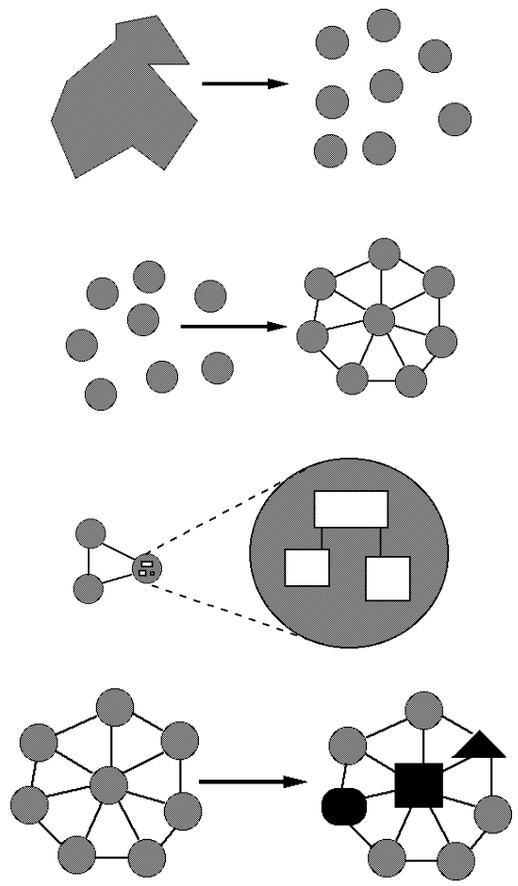
You know your software is modular when it is ...

- Decomposable
  - can be broken down into pieces
- Composable
  - pieces are useful and can be combined
- Understandable
  - one piece can be examined in isolation



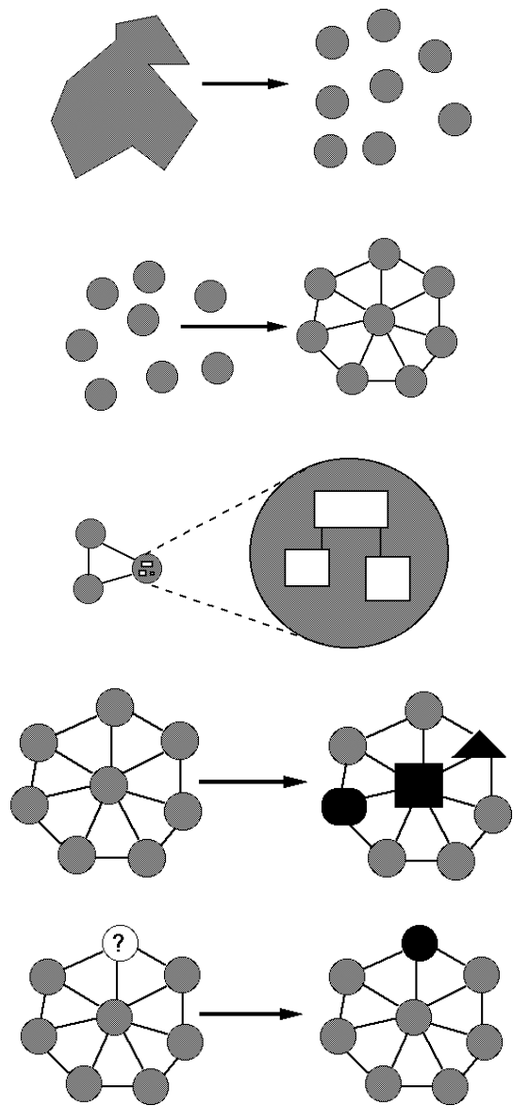
You know your software is modular when it is ...

- Decomposable
  - can be broken down into pieces
- Composable
  - pieces are useful and can be combined
- Understandable
  - one piece can be examined in isolation
- Adaptable
  - change in requirements affects few modules



You know your software is modular when it is ...

- Decomposable
  - can be broken down into pieces
- Composable
  - pieces are useful and can be combined
- Understandable
  - one piece can be examined in isolation
- Adaptable
  - change in requirements affects few modules
- Safe
  - an error affects few other modules



Achieving modularity: think about interfaces

- **Public interface**: data and behavior of the object that can be seen and executed externally by "client" code.
- **Private implementation**: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed.
- **Client**: code that uses your module.

Achieving modularity: think about interfaces

- **Public interface**: data and behavior of the object that can be seen and executed externally by "client" code.
- **Private implementation**: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed.
- **Client**: code that uses your module.

Achieving modularity: think about interfaces

- **Public interface**: data and behavior of the object that can be seen and executed externally by "client" code.
- **Private implementation**: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed.
- **Client**: code that uses your module.

Public interface is the speaker, volume buttons, station dial.

Private implementation is the guts of the radio (transistors, capacitors, voltage readings, etc.) that user should not see.

know

properties of architectures

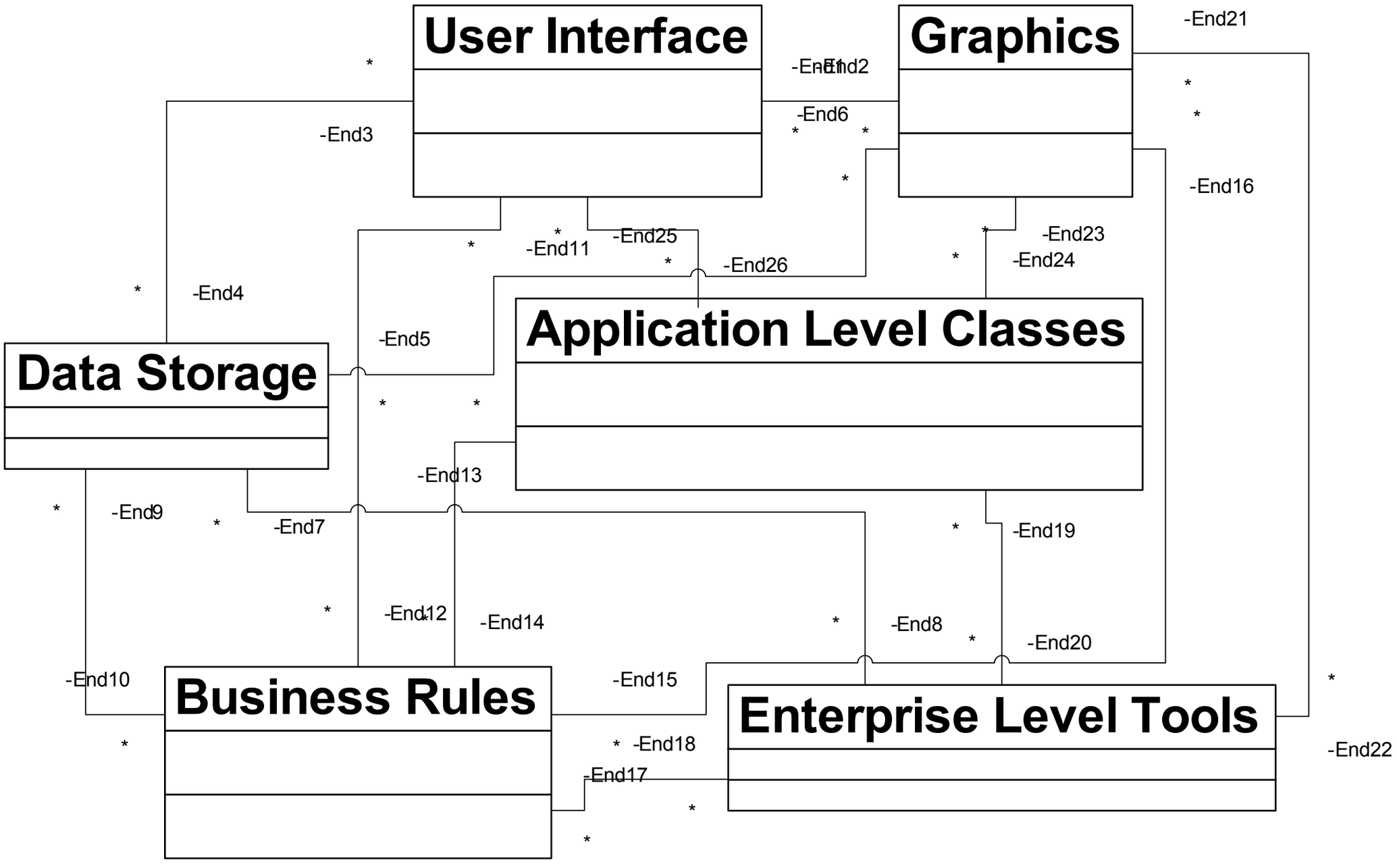
Key properties of an architecture

- Coupling
- Cohesion
- Style conformity
- Matching

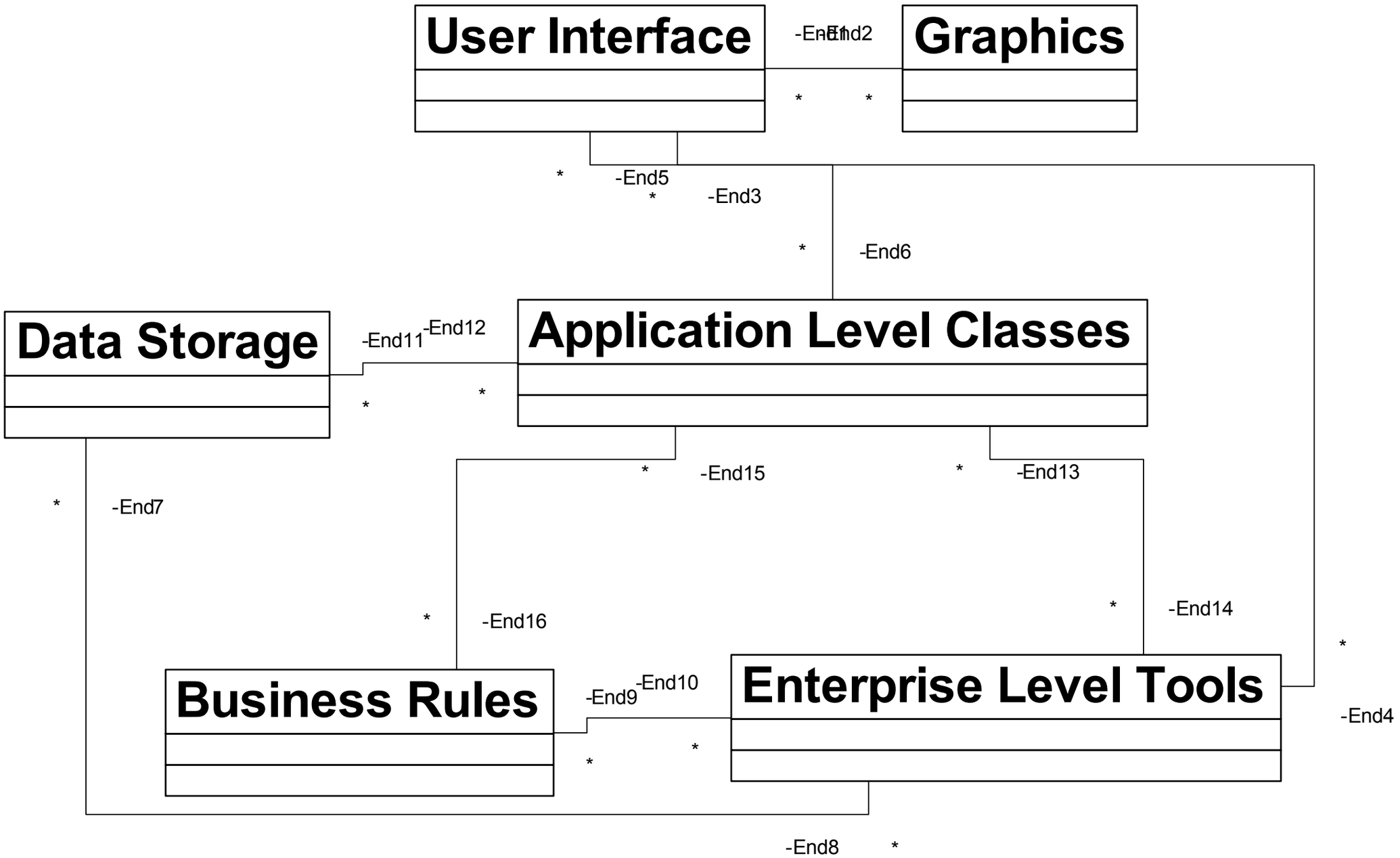
Coupling (loose vs tight)

- **Coupling**: the kind and quantity of interconnections among modules
- Modules that are **loosely coupled** (or uncoupled) are better than those that are tightly coupled
- The more tightly coupled two modules are, the harder it is to work with them separately

Tightly or loosely coupled?



Tightly or loosely coupled?



Cohesion (strong vs weak)

- **Cohesion**: how closely the operations in a module are related
- Tight relationships improve clarity and understanding
- Classes with **good abstraction** usually have **strong cohesion**
- No schizophrenic classes!

Strong or weak cohesion?

```
class Employee {

 public:

   FullName GetName() const;
   Address GetAddress() const;
   PhoneNumber GetWorkPhone() const;

   bool IsJobClassificationValid(JobClassification jobClass);
   bool IsZipCodeValid (Address address);
   bool IsPhoneNumberValid (PhoneNumber phoneNumber);

   SqlQuery GetQueryToCreateNewEmployee() const;
   SqlQuery GetQueryToModifyEmployee() const;
   SqlQuery GetQueryToRetrieveEmployee() const;

   …
}
```

Style conformity: what is a style?

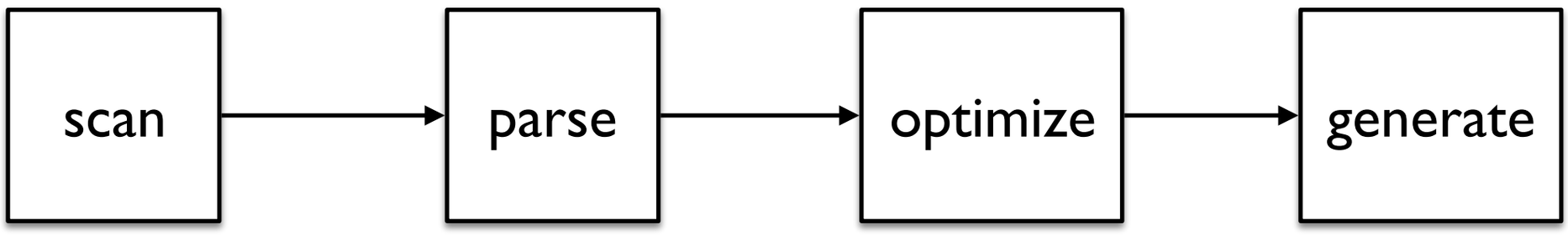
- An **architectural style** defines
    - The vocabulary of components and connectors for a family of architectures
    - Constraints on the elements and their combination
        - Topological constraints (no cycles, etc.)
        - Execution constraints (timing, etc.)
- By choosing a style, one gets all the known properties of that style (for any architecture in that style)
    - For example: performance, lack of deadlock, ease of making particular classes of changes, etc.

Style conformity: more than boxes and arrows

- Consider pipes & filters (Garlan and Shaw)
    - Pipes must compute local transformations
    - Filters must not share state with other filters
    - There must be no cycles
- If these constraints are violated, it's not a pipe & filter system
    - One can't tell generally this from a picture
    - One can **formalize** these constraints

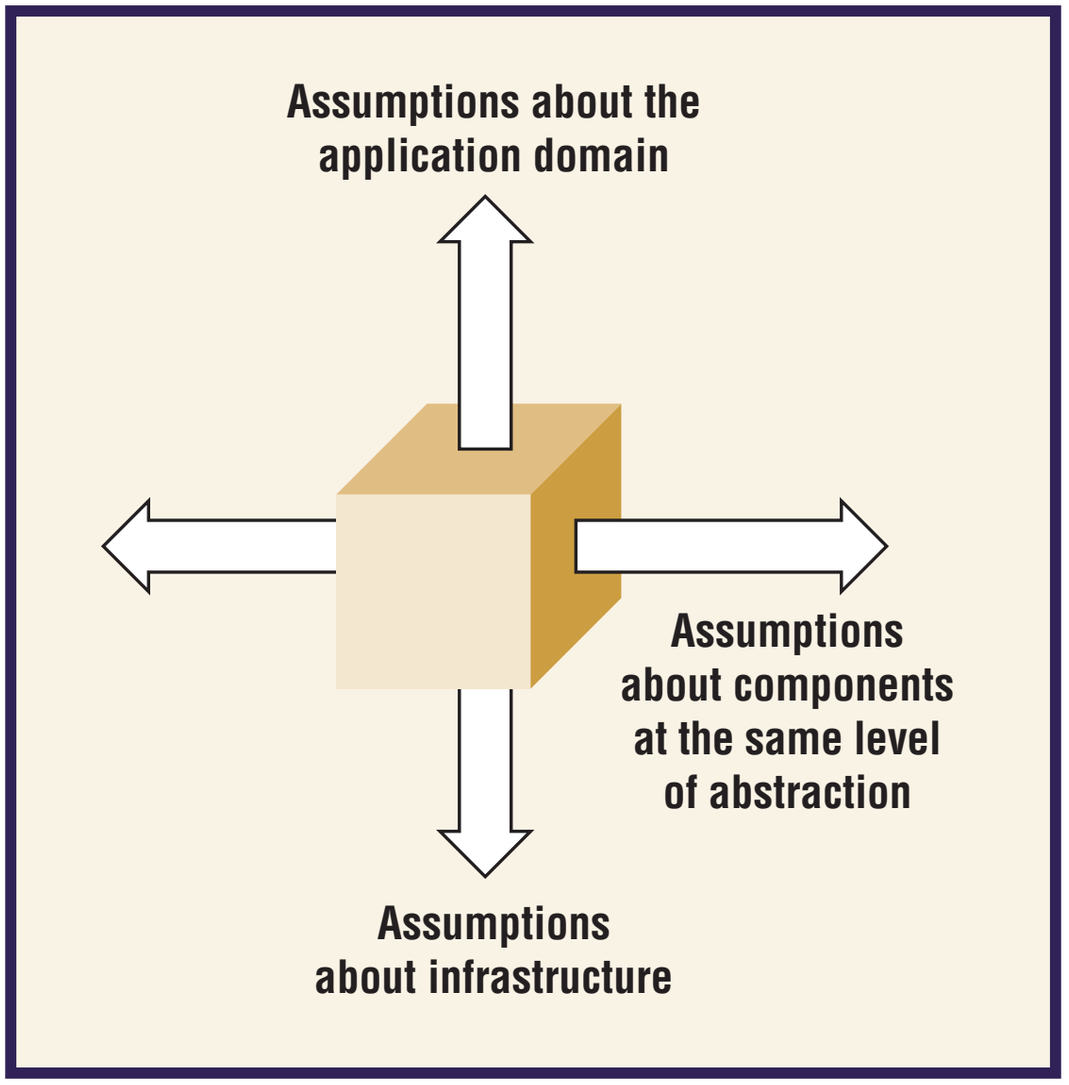
Style conformity: more than boxes and arrows

- Consider pipes & filters (Garlan and Shaw)
    - Pipes must compute local transformations
    - Filters must not share state with other filters
    - There must be no cycles
- If these constraints are violated, it's not a pipe & filter system
    - One can't tell generally this from a picture
    - One can **formalize** these constraints



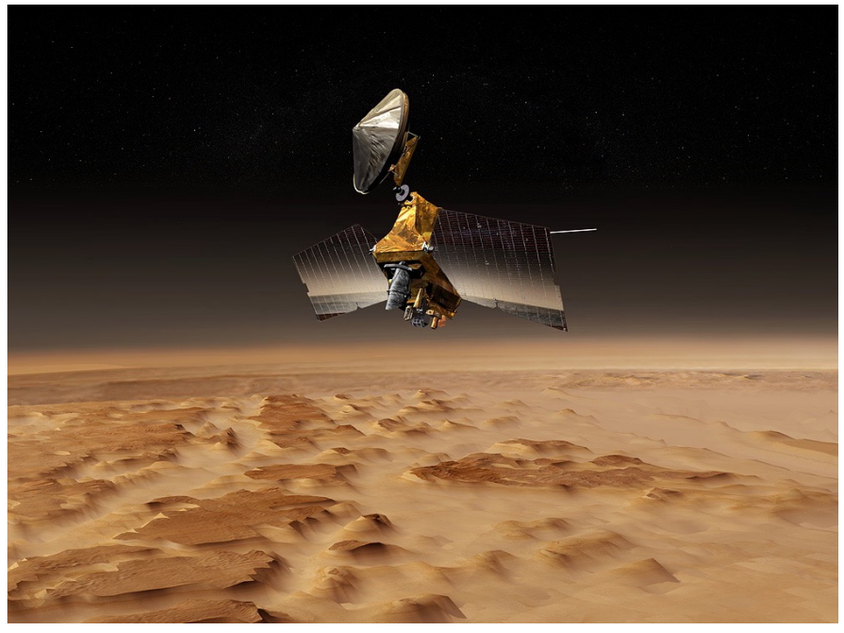
Component matching

- Components in an architecture **match** if they make compatible assumptions about their operating environment (Garlan, Allen, Ockerbloom).
- Mismatches lead to
  - Excessive code size
  - Poor performance
  - Error-prone construction
  - Having to modify off-the-shelf components



Interface mismatch

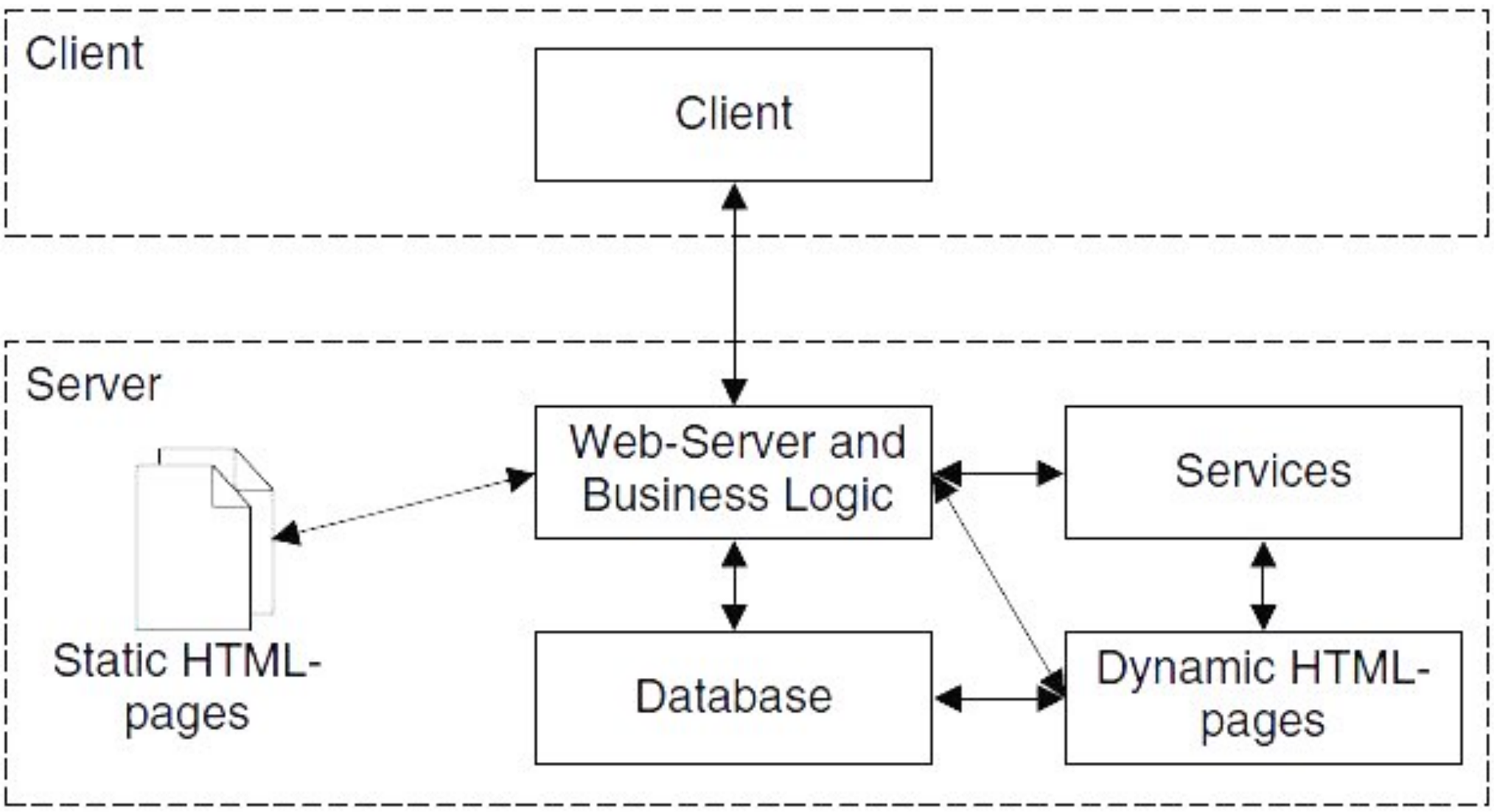
NASA lost a $125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation.



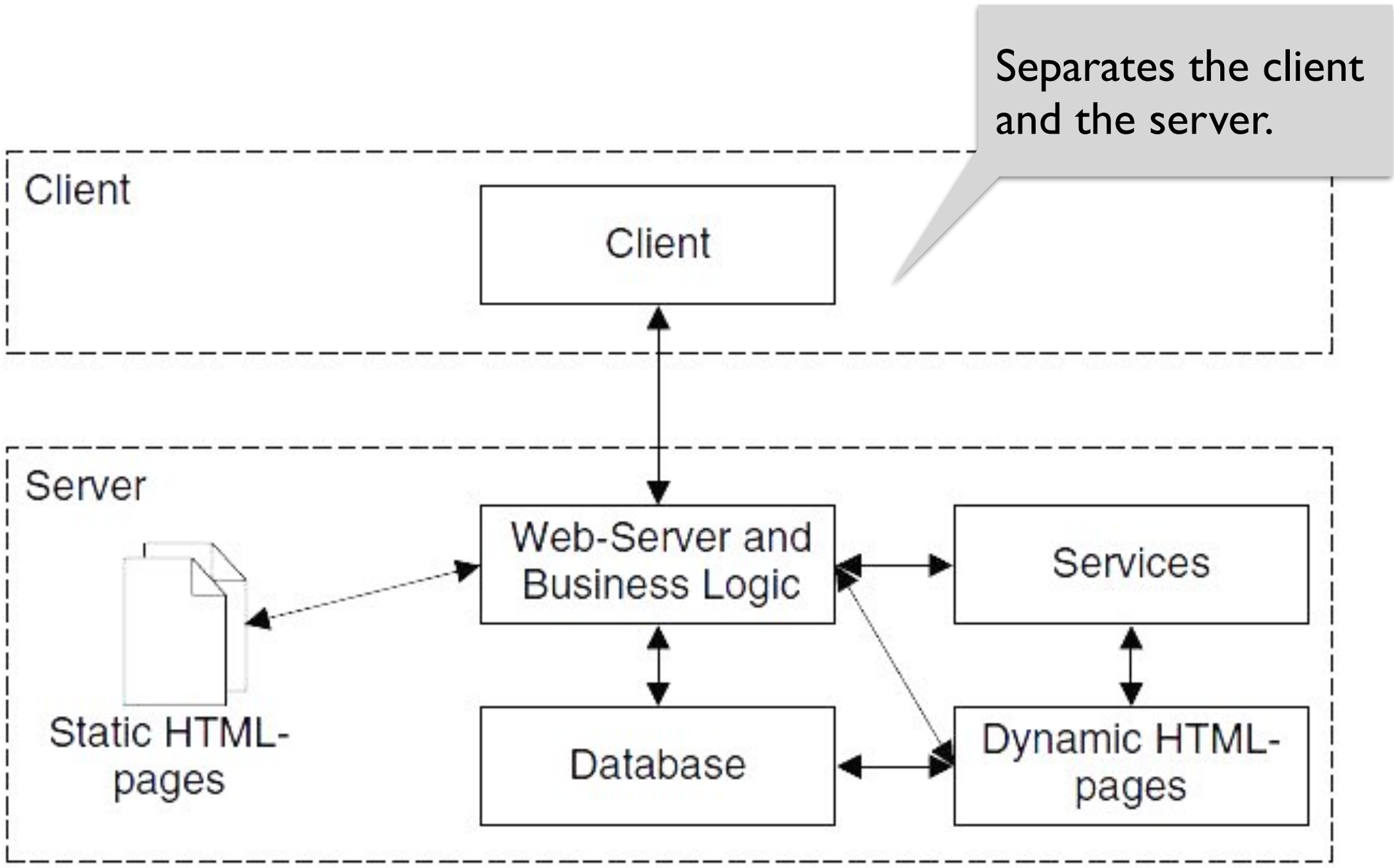
example architectures

study

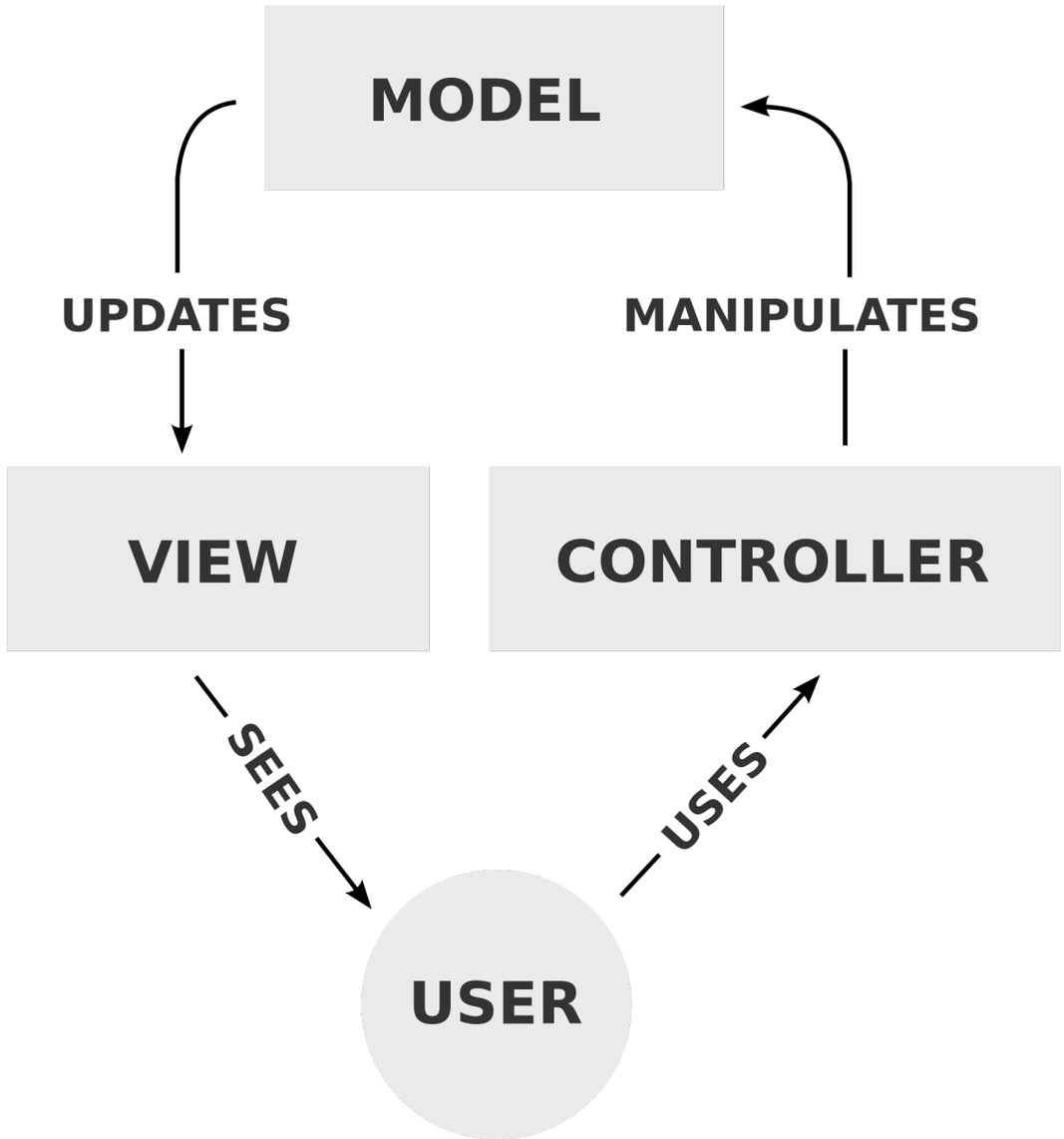
Client-server architecture



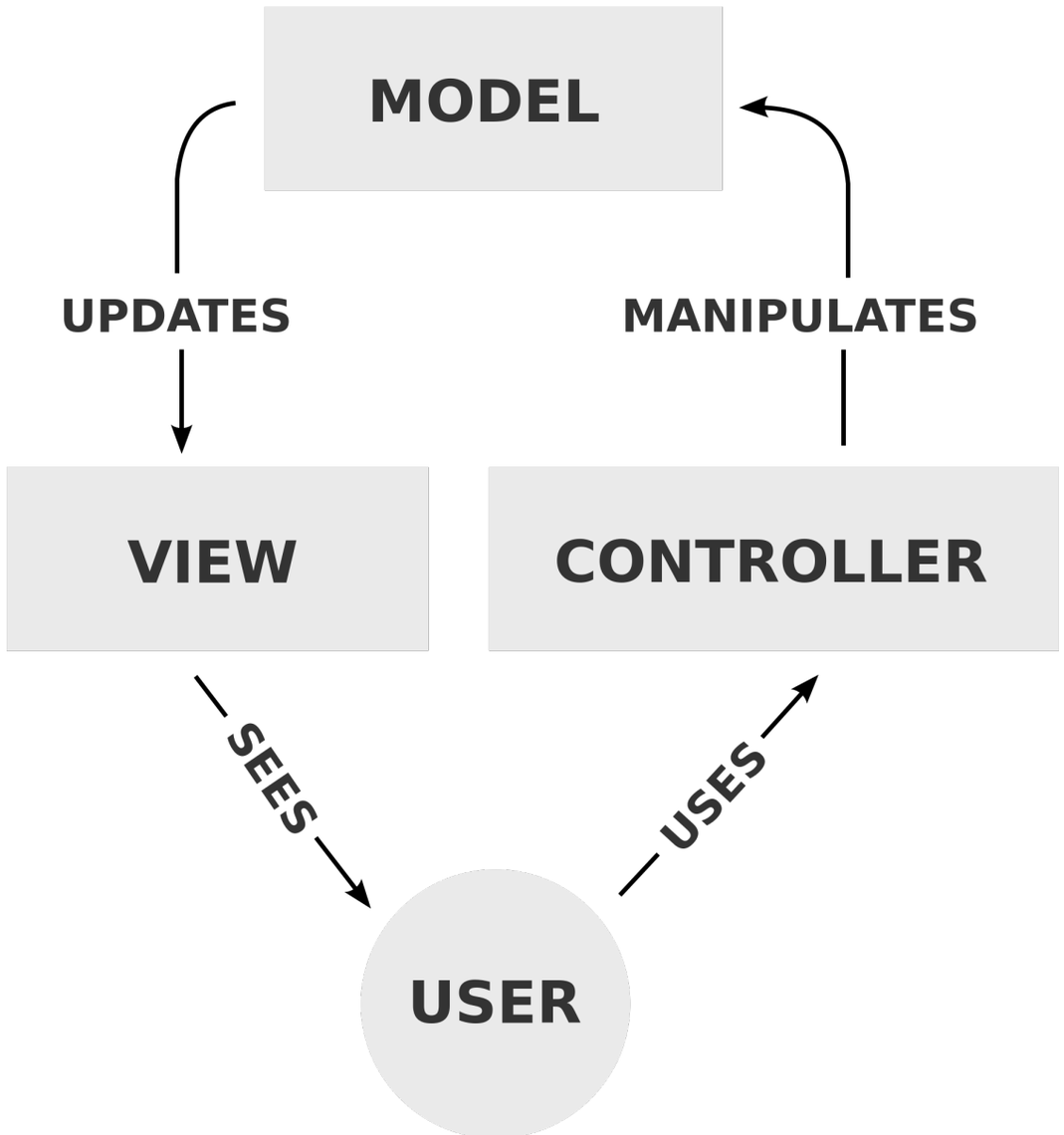
Client-server architecture



Model, view, controller (MVC)



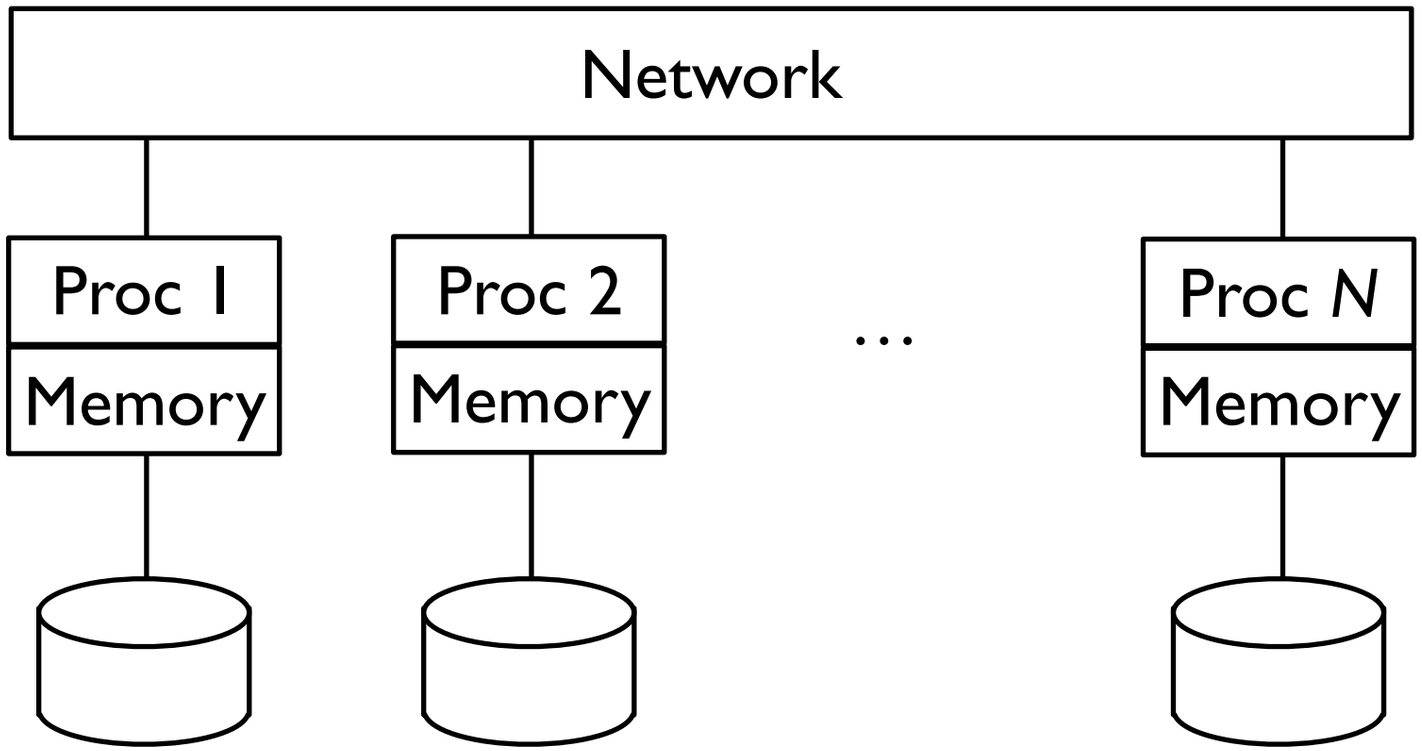
Model, view, controller (MVC)



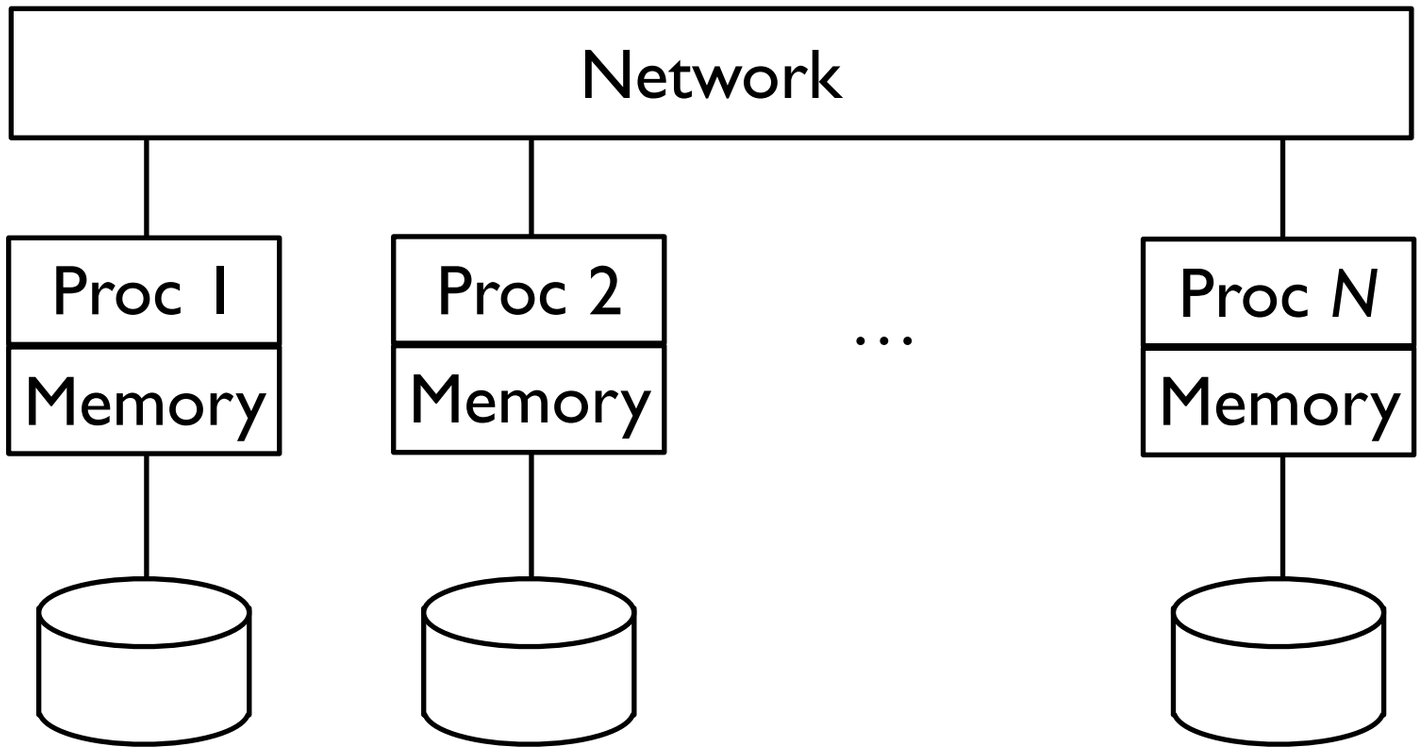
Separates:

- the application object (model)
- the way it is represented to the user (view)
- the way in which the user controls it (controller)

Shared nothing (SN) architecture



Shared nothing (SN) architecture



Separates individual components (nodes) from each other.

Summary

- An architecture provides a high-level framework to build and evolve a software system.
- Strive for modularity: strong cohesion and loose coupling.
- Consider using existing architectural styles or patterns.