SOFTWARE ENGINEERING

Chapter 3.5: System Design Architectures

MOTIVATION…

“Order and simplification are the first steps towards mastery of a subject”

Thomas Mann

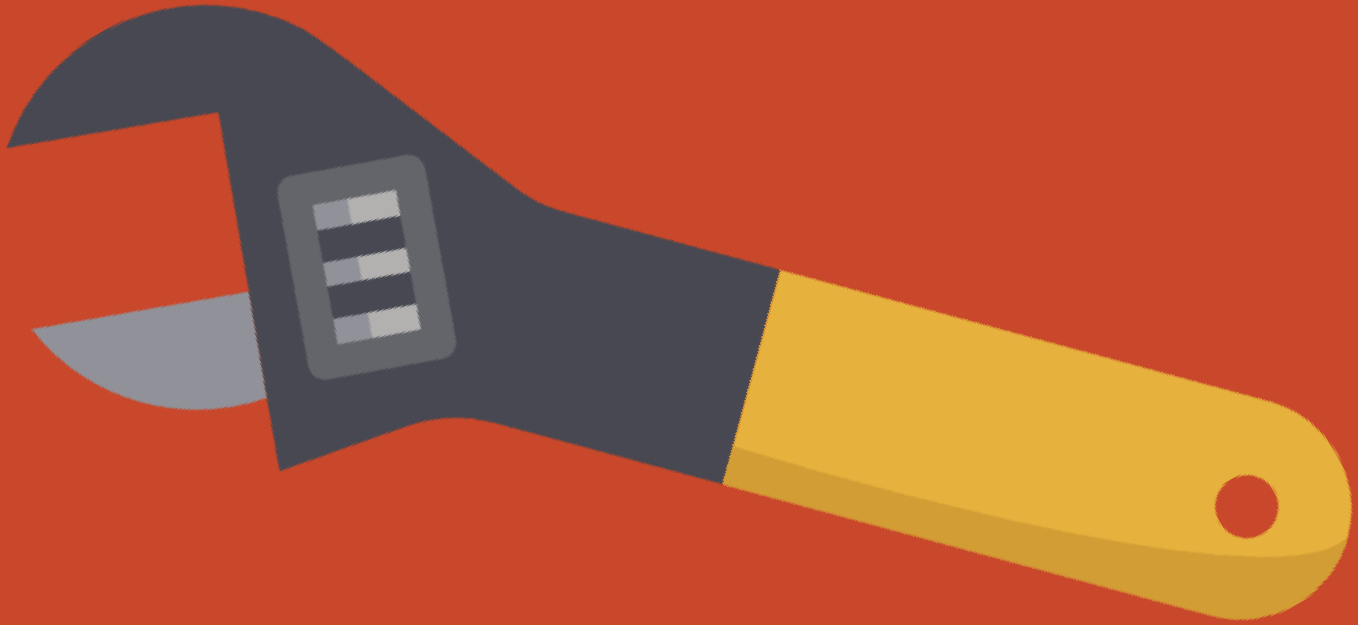
Remember!...

Requirements Engineering

- What is the problem?
- What are the requirements?

Design

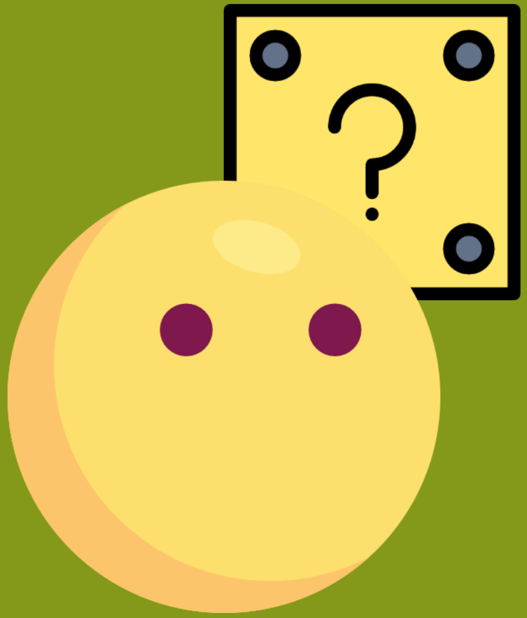
- How to build a solution?
- How to solve the problem taking into account the analysis?



Note!

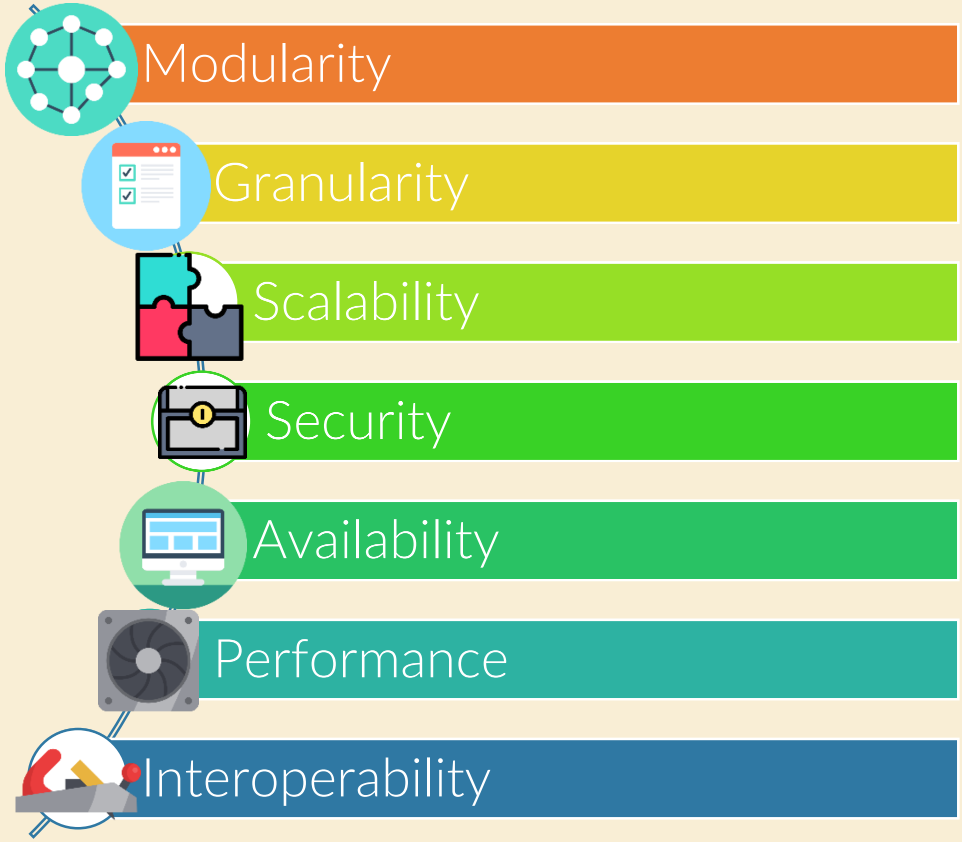
Probably you will want to create a product that fulfills and accomplishes the expected requirements

Remember non functional requirements?



Key Concepts

(They’re totally related with non functional requirements)



Modularity

Granularity

Scalability

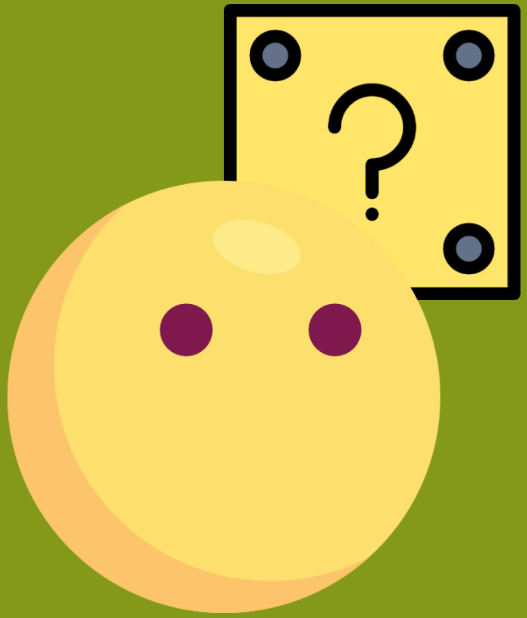
Security

Availability

Performance

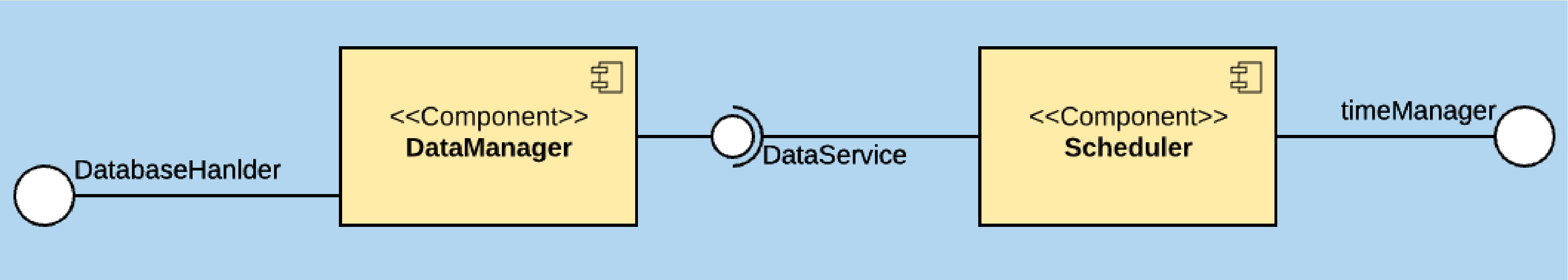
Interoperability

Could you define all of them?



Modularity

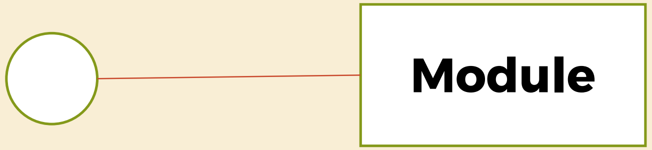
This is the property of dividing software into different modules that can blend together



Component diagram for the Circus example

Software Modules

- Has all the code for a given functionality (Encapsulation)
- Has an interface that allows the access for *clients*
- This interface also allows the module to be plugged in with another one
- Is easily packaged and can easily be deployed



Granularity

Defines how your product will tackle the fine details

How detailed will your solutions be?

Example

- Imagine your software is a firewall:
  - Block IP addresses
    - Detect where the IP addresses come from to block them
      - Detect related IP addresses and block them
        - Detect packets and select them
          - Learn… 😵‍💫

How many of this fine details will your software be able to provide?

Scalability

- What happens when the load of work grows?

Scale Up:	• Stronger Hardware
Scale Out:	• More Hardware

Scalability

Vertical

- Add more hardware and create a load balancer that spreads the load amount into different servers.

Horizontal

Improve the behaviour in threaded tasks and parallel traffic.

Availability

- Availability can be seen in different contexts:

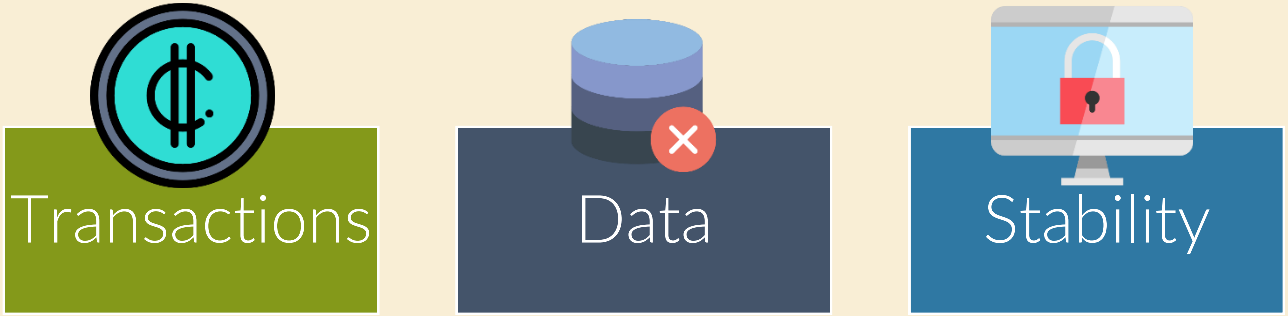
Services

System

Data

Security

- This measures how safe will the information of the system be and if the system is able to respond to any attack.



Transcriptions

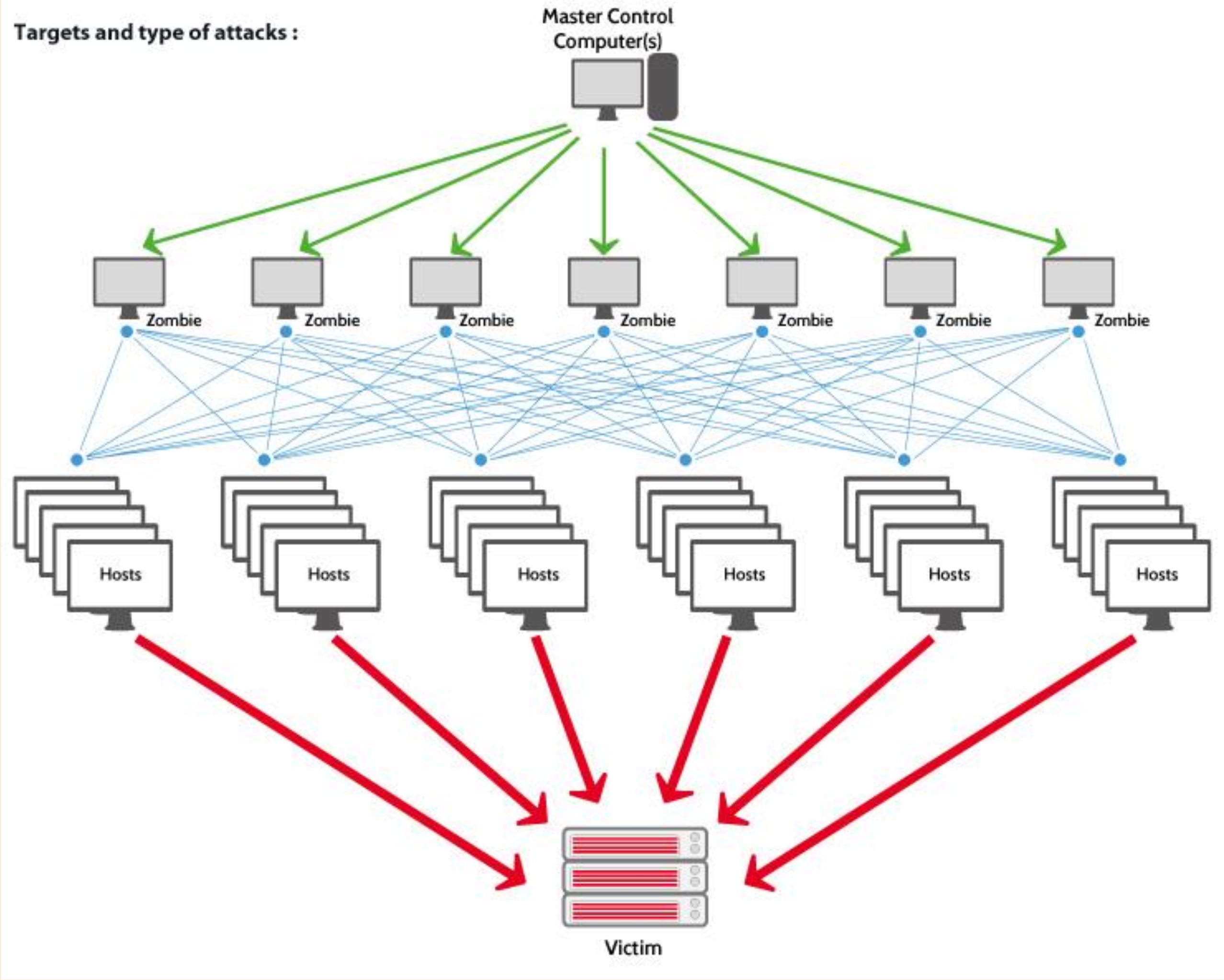
Data

Stability

Security

Example of denial of service DDoS attack!

Other security flaws such as phishing, data corruption... are important requirements!



This gets worse when we consider multiple interfaces

A software system's architecture is the set of principal design decisions made about the system..

N. Taylor et al

Architectural Models

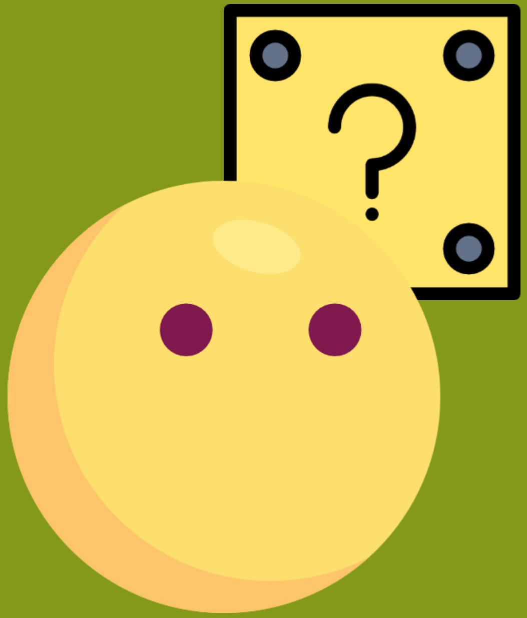
- Focus discussion about software requirements
- They also help to document a design

Architectural Models
Principle 4+1

- Logical View
  Structure
- Process View
  Runtime
- Development View
  Components
- Physical View
  Hardware

Conceptual View

What about patterns?



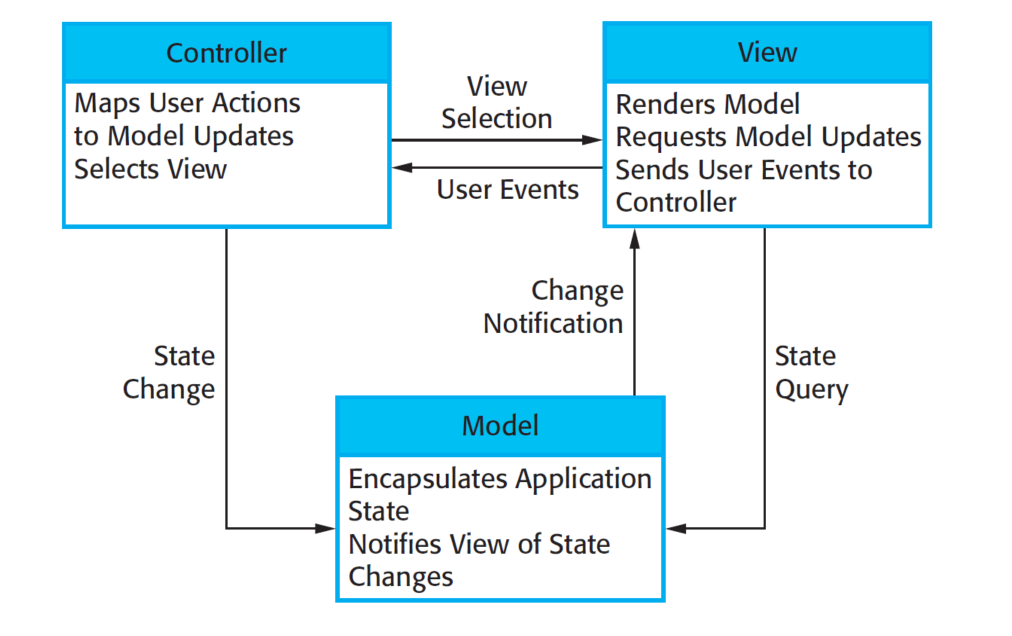
Architectural conceptual Patterns

- They're generated by some architectural design successful cases.
- They normally suit the best some kind of problems and are generic, an information architect is the one who proposes a system architect based on system's specification.

Model - View – Controller
MVC

- Decouples the interactions with the data management and event manipulation

- This architecture works best when view requirements tend to be unknown

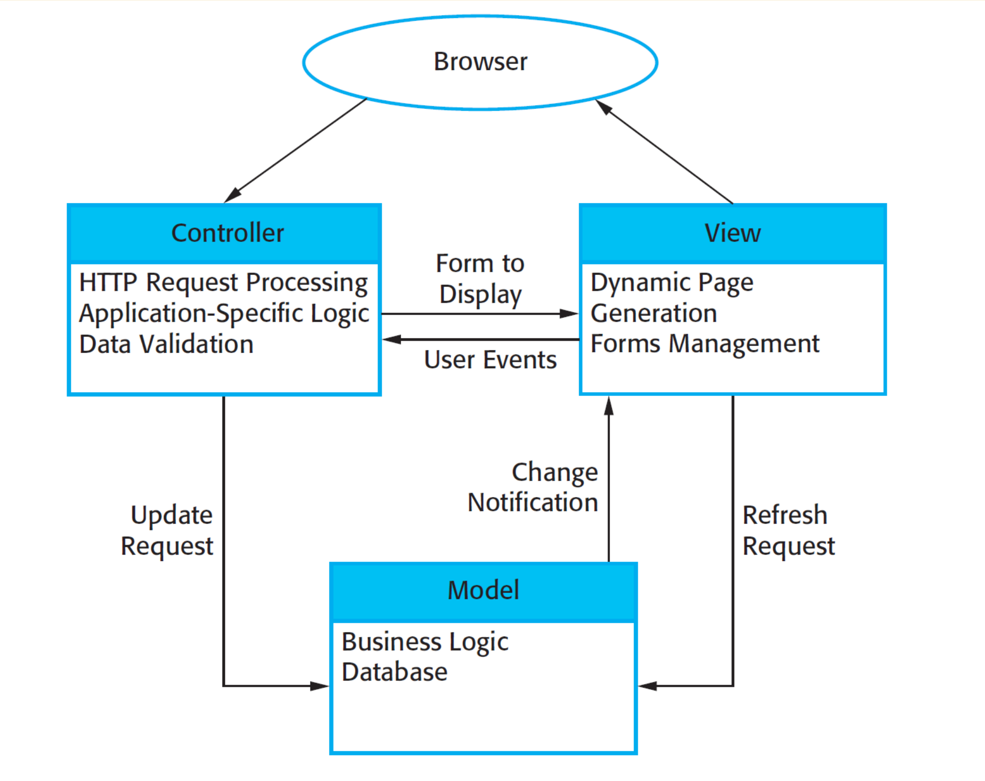


Taken from Somerville

Model - View – Controller MVC

- Allows the free interchange of data independent of its representation.

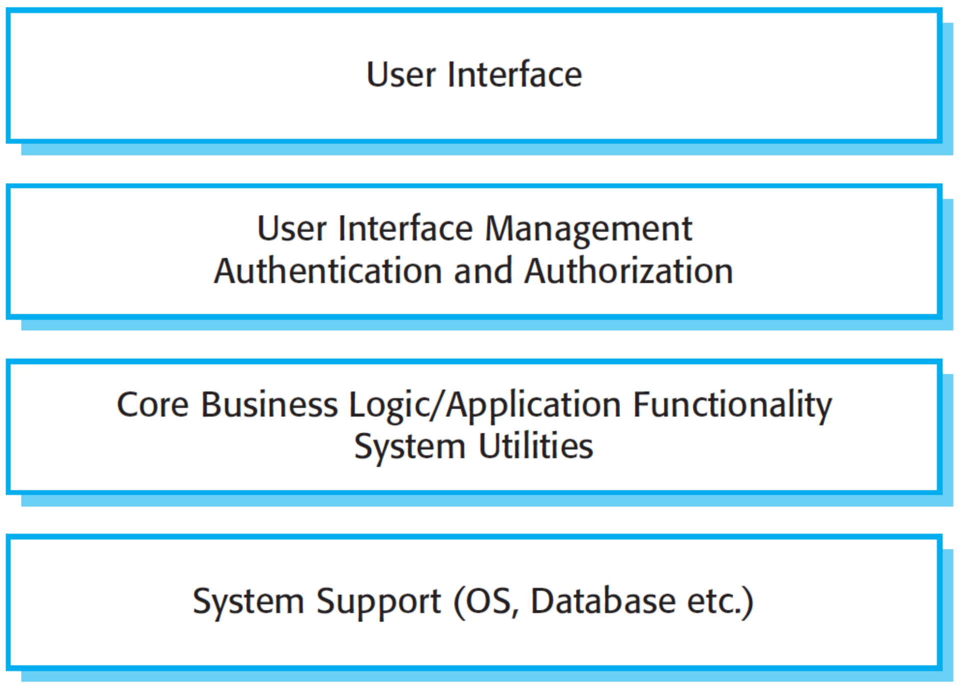
- Can involve additional code when the view is simple.



Taken from Somerville the product is a website

Layered Architecture

- Organizes the system in different layers of functionality
- Each layer provides functionality from high to low levels.



User Interface

User Interface Management
Authentication and Authorization

Core Business Logic/Application Functionality
System Utilities

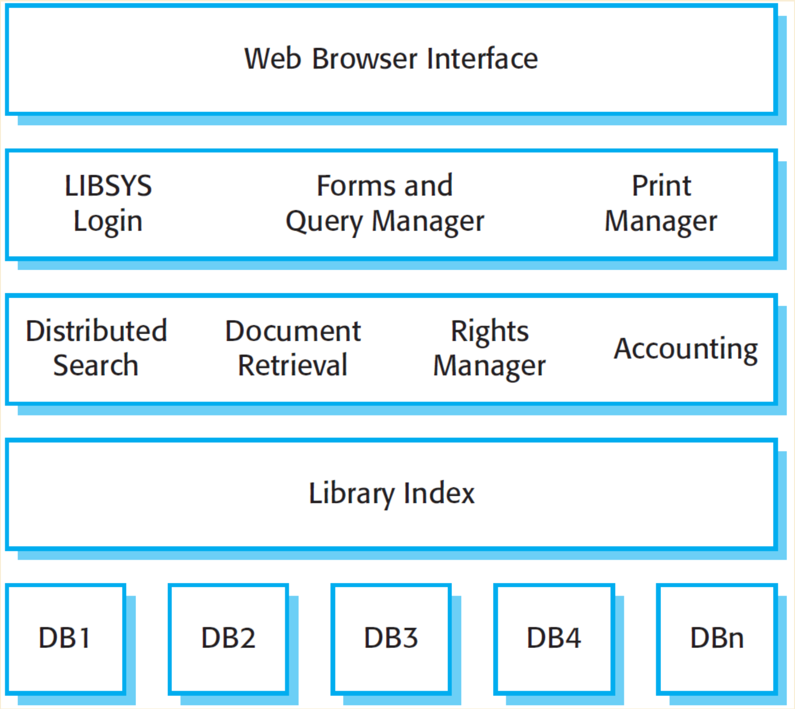
System Support (OS, Database etc.)

Taken from Somerville

Layered Architecture

- Development can be separated in the different layers. (but connections between them can become difficult)

- Each layer can fulfill the five requirements in different ways if required

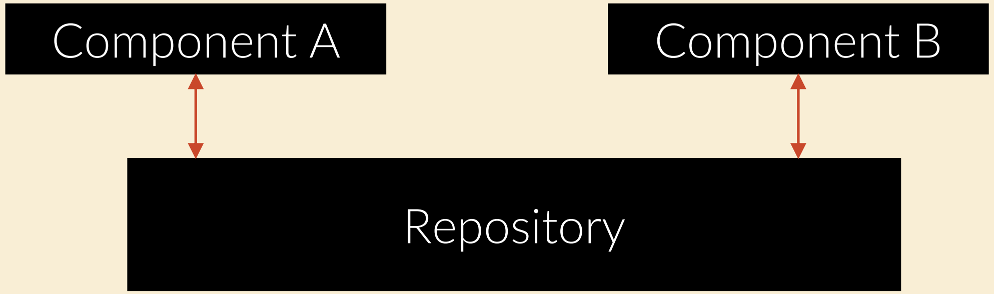


Taken from Somerville the product Is a copyright detector

If layers are too deep, reaching the low level features due to a service request of the system can become too complex and performance may be harmed by that!

Repository Based Architecture

- Suits best data/driven systems
- This pattern works best when information needs to be stored at large amounts of time
- Components are decoupled!

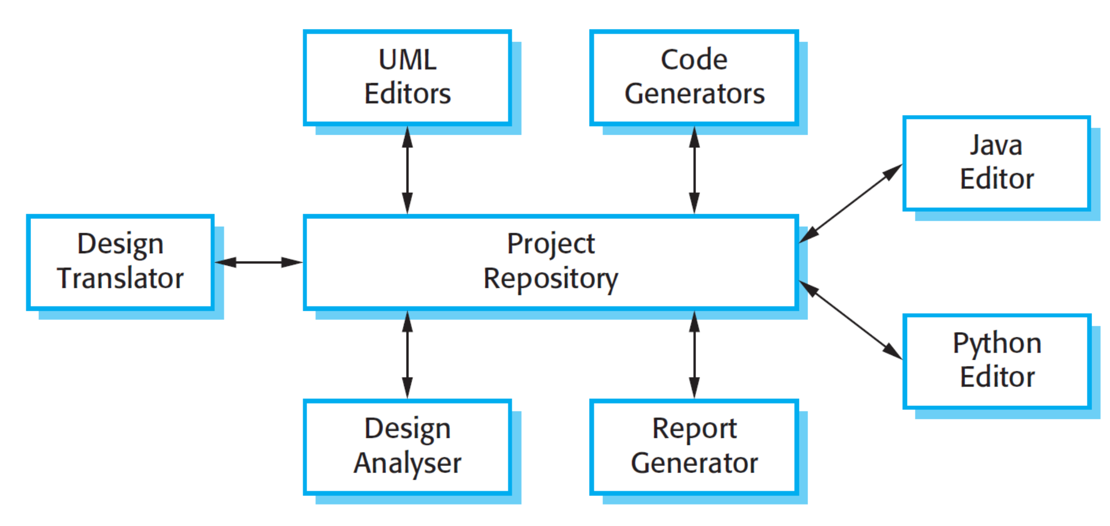


Taken from Somerville

Repository Based Architecture

- The repository is a critical feature, everything is stored there.

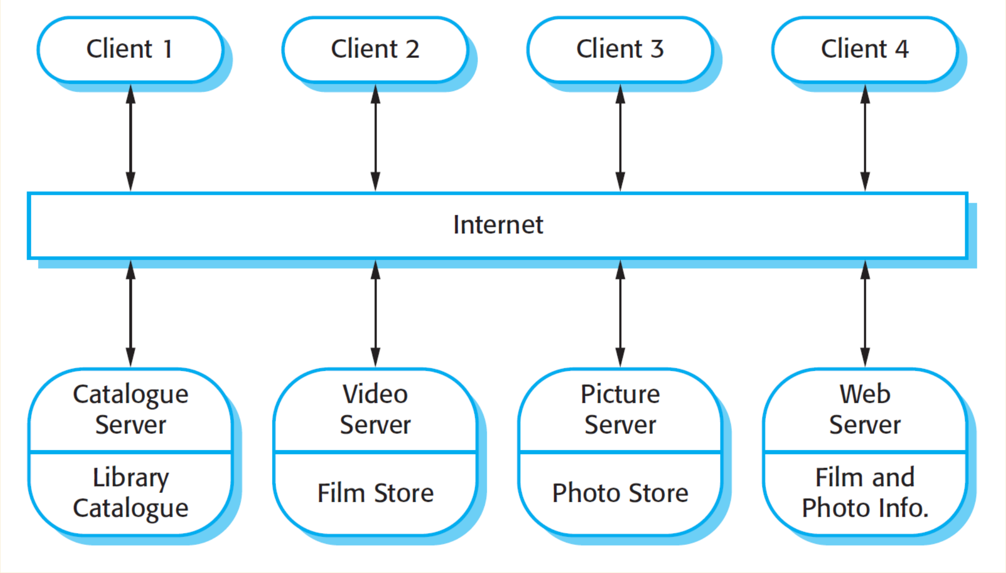
- Any problem in the repository affects the whole system!



Taken from Somerville the product is an IDE

Client-Server Architecture

- Each client asks for services
- The network connects the clients with servers that offer the services
- This architecture is an introduction to the services one!



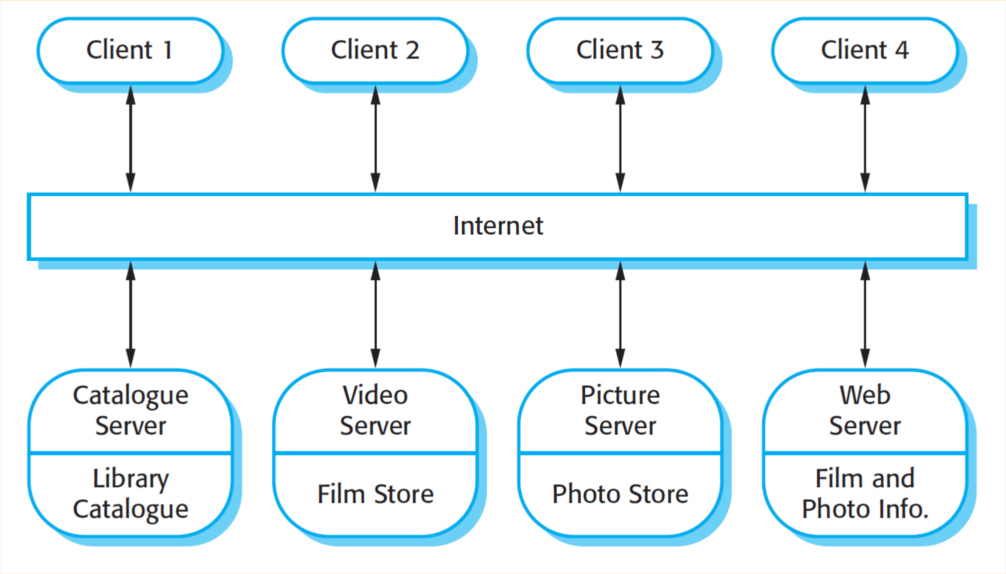
Taken from Somerville the product is an store of photography and video contents

Client-Server Architecture

- Susceptible to denial of service attacks!

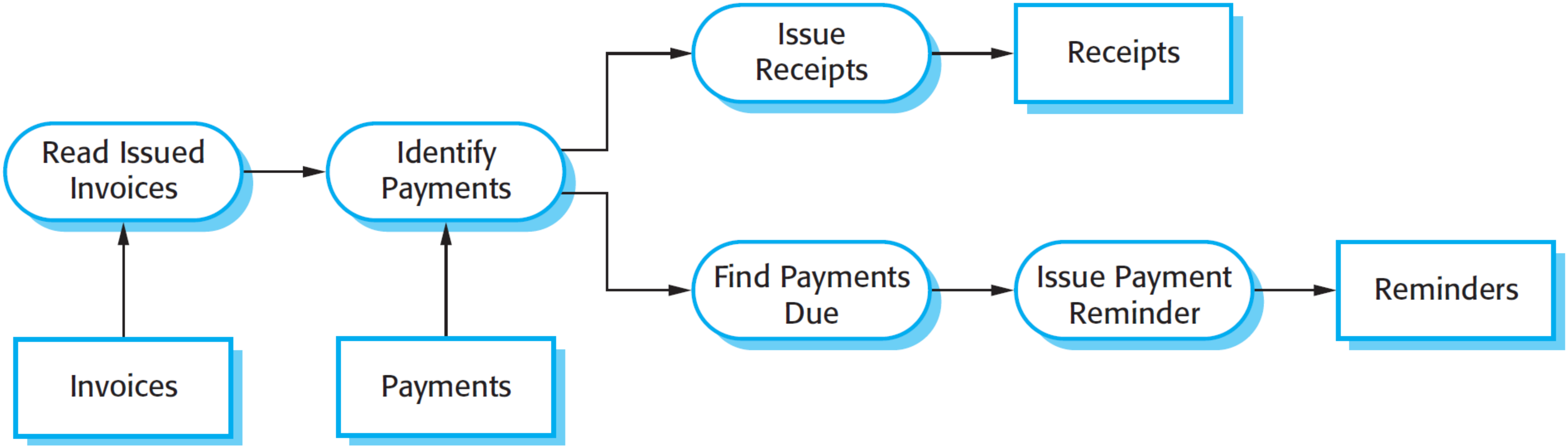
- Decoupled but at the same time each server can represent a critical part of the system

- Easily replicable, data is totally accessible.



Taken from Somerville the product is an store of photography and video contents

Pipe and Filter architecture



Taken from Somerville the product is an invoice processing tool to detect payments, receipts and reminders!

- Totally susceptible of data corruption!
- The format for each data stream passing through the system must be agreed for every node (filter in this case).

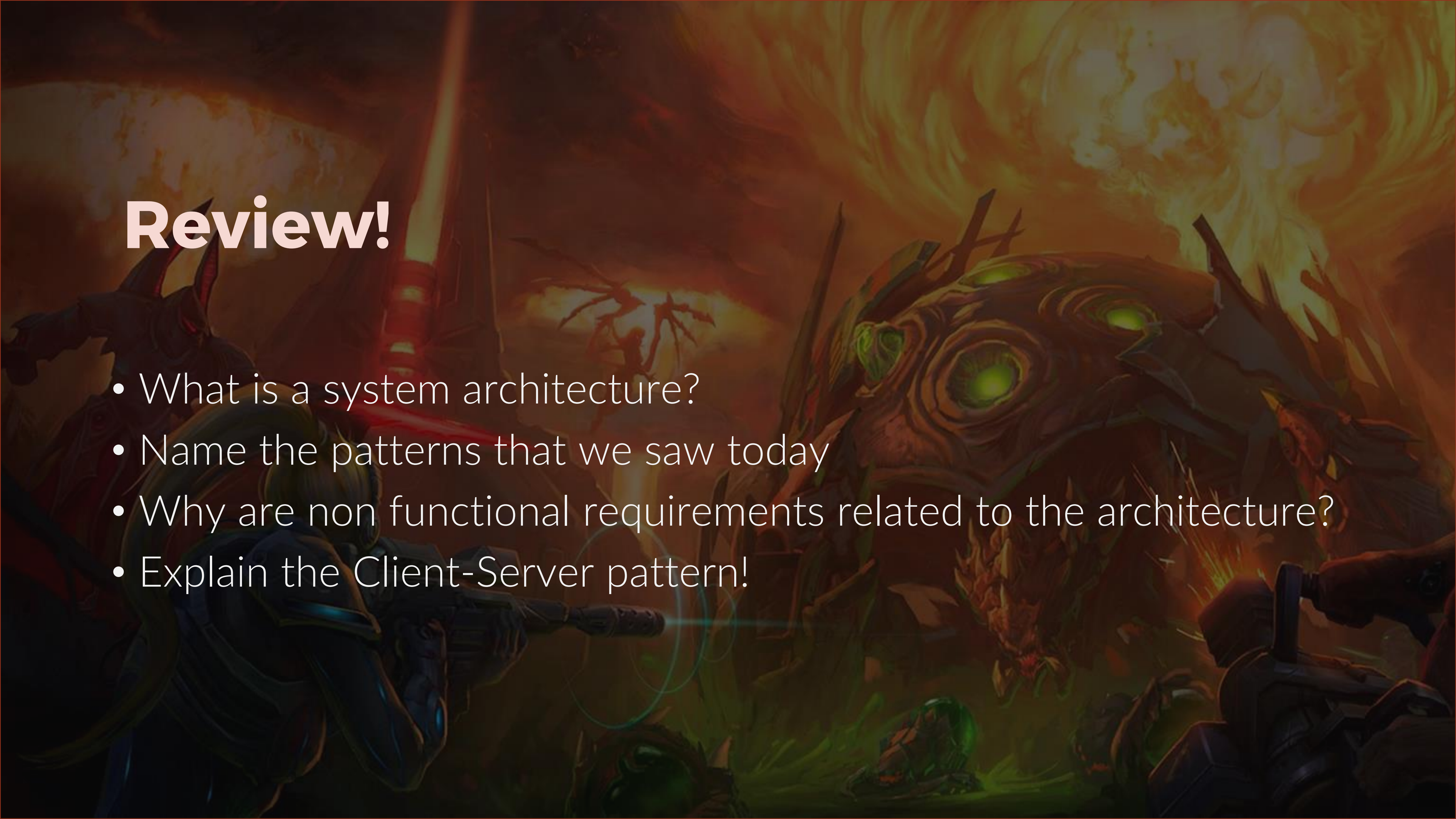
Question!

How do you think that our attributes (security, availability…) may work in this pattern?

Help!!!

Help me to draw the architecture of the circus project!

One Shot Review



Review!

- What is a system architecture?
- Name the patterns that we saw today
- Why are non functional requirements related to the architecture?
- Explain the Client-Server pattern!

References

- [SOMMERVILLE] Ian Sommervile. *Software Engineering 9th Edition*
- [SCHMIDT] Richard Schmidt. *Software Development Architecture-Driven Software Development*
- [STEPHENS] Beginning Software Engineering. 2015
- [CROOKSHANKS] Software Development Techniques. 2015

Class has died... for today!