Software Architectures Introduction

Software Engineering Seminar

Author: Eng. Carlos Andrés Sierra, M.Sc.
cavirguezs@udistrital.edu.co

Full-time Adjunct Professor
Computer Engineering Program
School of Engineering
Universidad Distrital Francisco José de Caldas

2025-III

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

1. Systems Thinking
2. Reference Architectures

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Outline

1 Systems Thinking

2 Reference Architectures

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

What is a System?

- A **system** is a set of interacting components that *work* together to achieve a common goal.
- A **system** is a collection of elements that are *organized* in a specific way.
- A **system** is a structure that is designed to *perform* a specific function.

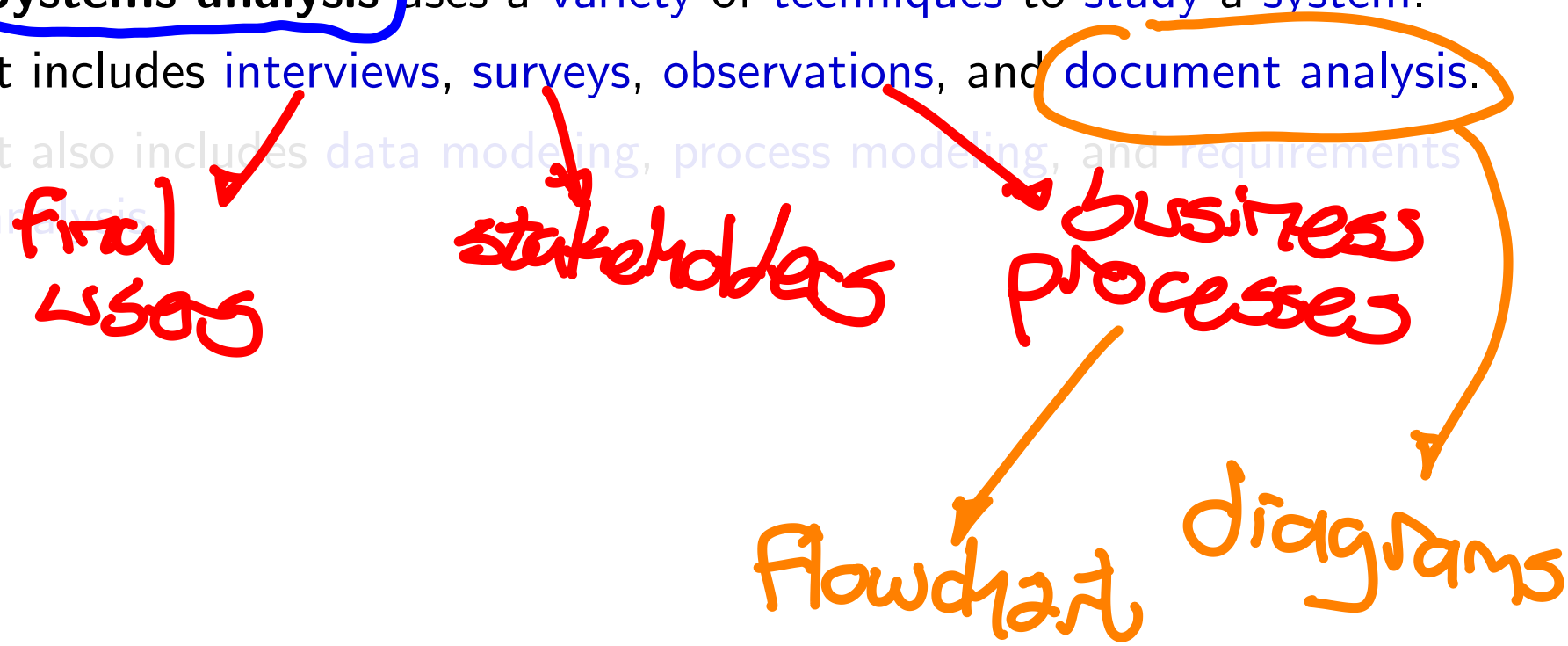
Systems Analysis Process

- **Systems analysis** is the *process* of studying a system in order to identify its components, interactions, and goals.
- **Systems analysis** is the *process* of understanding how a system works and how it can be improved.
- **Systems analysis** is the *first step* in the systems development lifecycle.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Systems Analysis Techniques

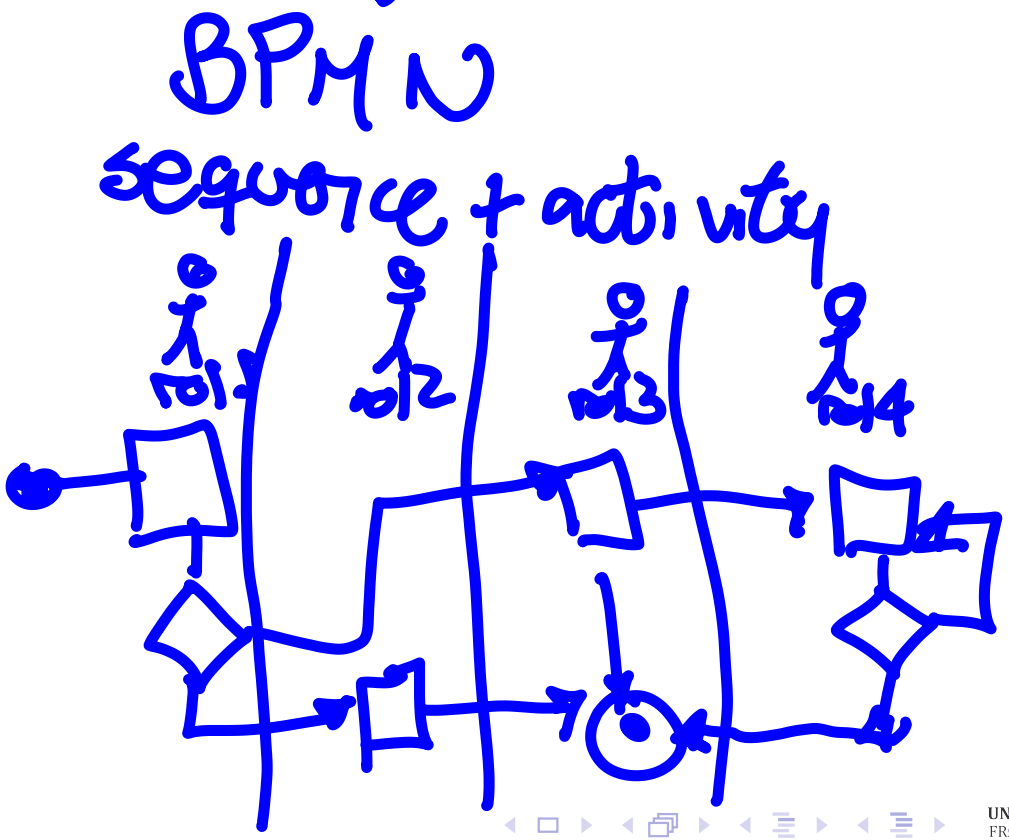
- **Systems analysis** uses a variety of techniques to study a system.
- It includes interviews, surveys, observations, and document analysis.
- It also includes data modeling, process modeling, and requirements analysis.



Systems Analysis Techniques

- **Systems analysis** uses a variety of techniques to study a system.
- It includes interviews, surveys, observations, and document analysis.
- It also includes data modeling, process modeling, and requirements analysis.

- sources
- format + structure
- understanding



Complexity and Emergence

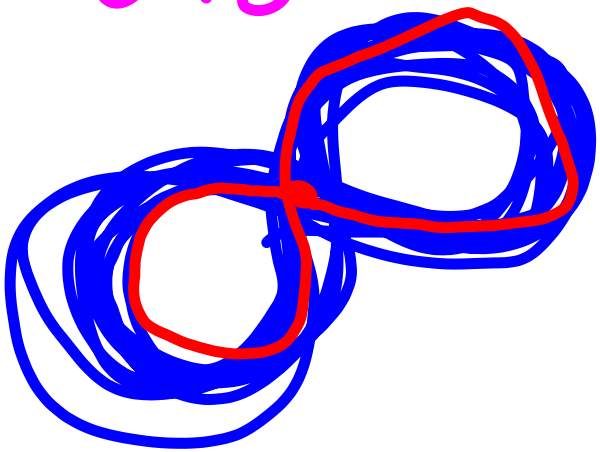
- **Complexity** is the *degree* to which a system is difficult to understand.
- **Emergence** is the *appearance* of unexpected properties in a system that arise from the interactions of its components.
- **Complexity** and **emergence** are common in dynamic systems that are non-linear and chaotic.
- They can be *studied* and understood through systems analysis and modeling.

elements + relations

chaos theory

randomness (stochastic) → non-linear → chaotic attractors

Lorenz



Complexity and Emergence

- **Complexity** is the *degree* to which a system is difficult to understand.
- **Emergence** is the *appearance* of unexpected properties in a system that arise from the interactions of its components.
- Complexity and emergence are common in dynamic systems that are nonlinear and chaotic.
- They can be *studied* and understood through systems analysis and modeling.

Complexity and Emergence

- **Complexity** is the *degree* to which a system is difficult to understand.
- **Emergence** is the *appearance* of unexpected properties in a system that arise from the interactions of its components.
- **Complexity** and **emergence** are `common` in dynamic systems that are non-linear and chaotic.
- They can be *studied* and `understood` through systems analysis and modeling.

Strategies to Solve Problems

- **Top-Down**: Start from the big picture and break it down into smaller parts.
- **Bottom-Up**: Start from small, well-defined components and integrate them into a complete system.
- Both strategies are useful and often *combined* in `software design`.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Strategies to Solve Problems

- **Top-Down**: Start from the big picture and break it down into smaller parts.
- **Bottom-Up**: Start from small, well-defined components and integrate them into a complete system.
- Both strategies are useful and often *combined* in `software design`.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Strategies to Solve Problems

- **Top-Down**: Start from the big picture and break it down into smaller parts.
- **Bottom-Up**: Start from small, well-defined components and integrate them into a complete system.
- Both strategies are useful and often *combined* in `software design`.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Process Definition

- A **Process** is a series of `steps or actions` taken to achieve a particular end.
- Processes are used to **organize** and **manage** work.

Workflows

- A **Workflow** is a series of `tasks` that are performed in a specific order to *achieve a goal*.
- **Workflows** are used to *automate* and *optimize* business processes.
- **Workflows** can be sequential, parallel, conditional, or repetitive.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Process Models

- A **Process Model** is a representation of a process that shows the sequence of steps and the relationships between them.
- **Process models** are used to analyze, design, and improve processes.
- Examples of process models include `flowcharts`, `data flow diagrams`, `activity diagrams`, `business process model and notation (BPMN)`, `petri nets`, `state diagrams`, among others.

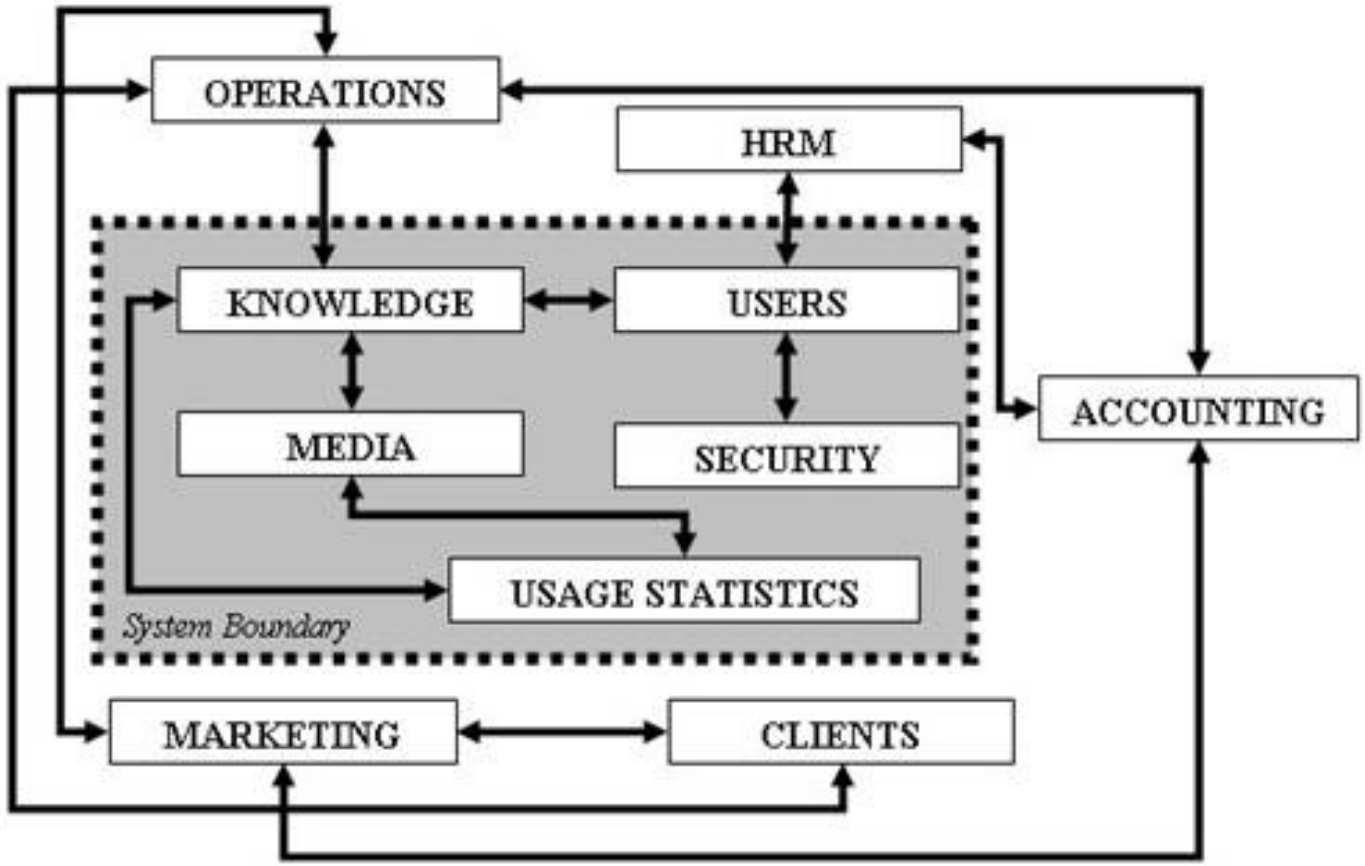
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Causal Loops

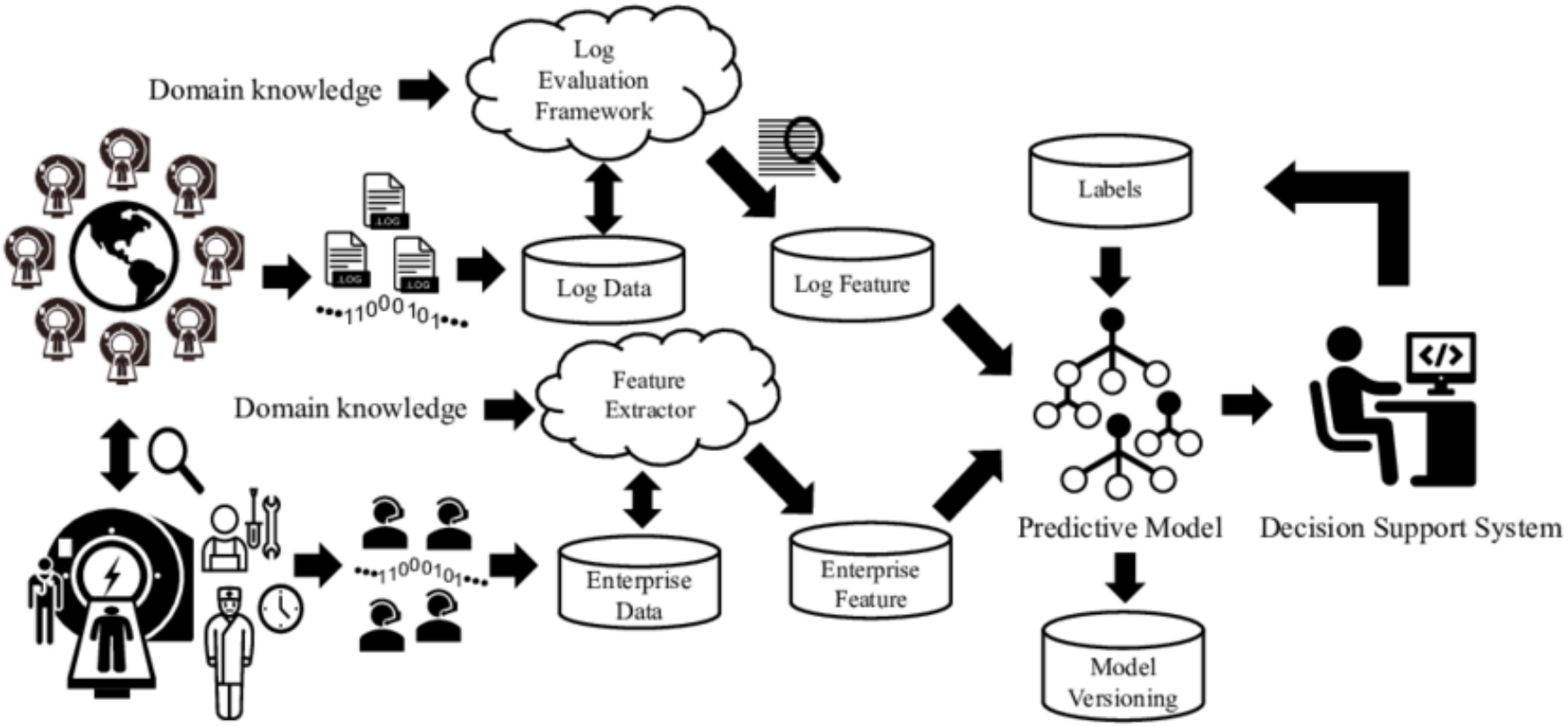
- A **Causal Loop** is a diagram that shows the relationships between different variables in a system.
- Causal loops are used to analyze and understand the dynamics of a system.
- Causal loops can be positive or negative.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

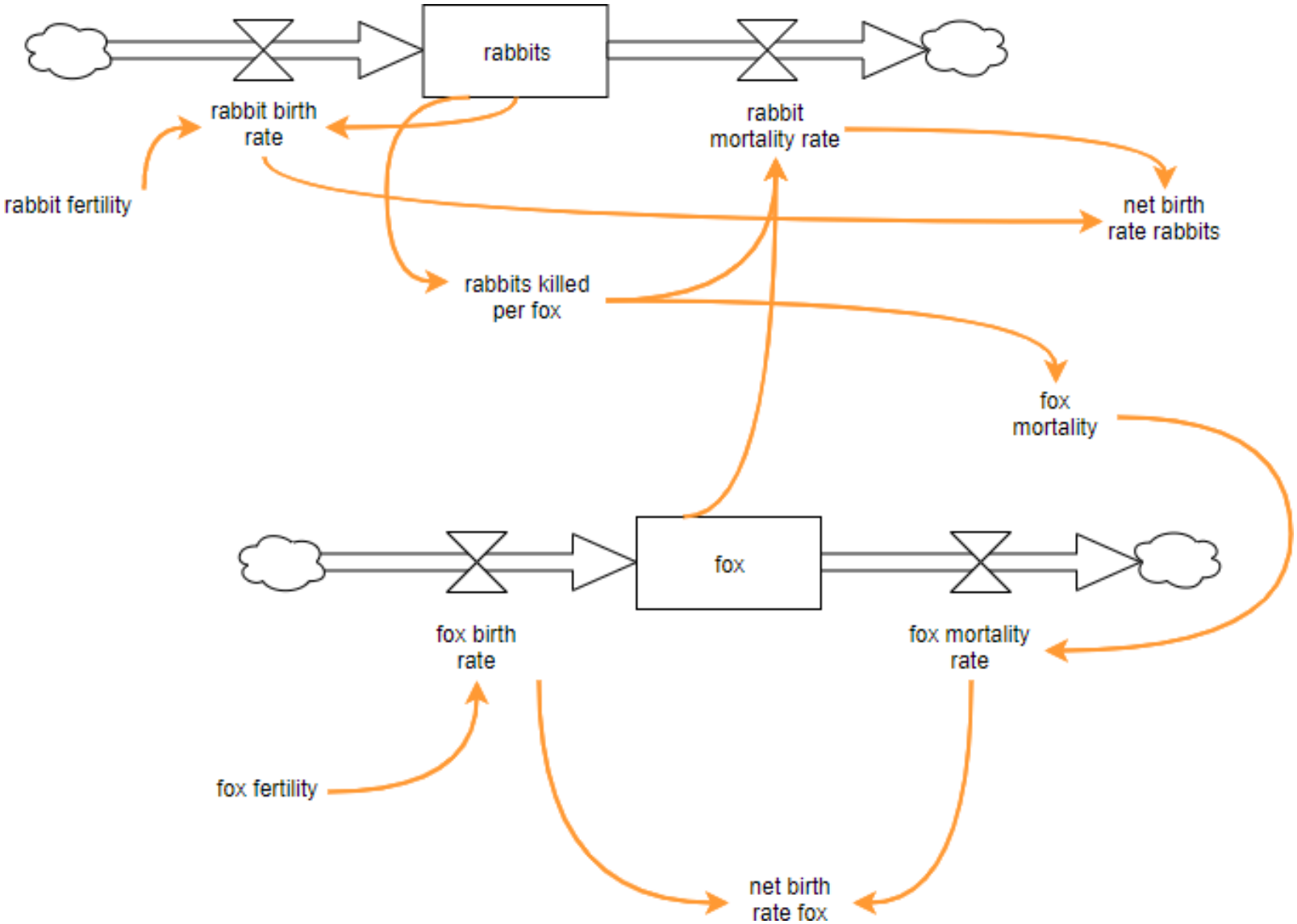
System Schema Example: Company Structure



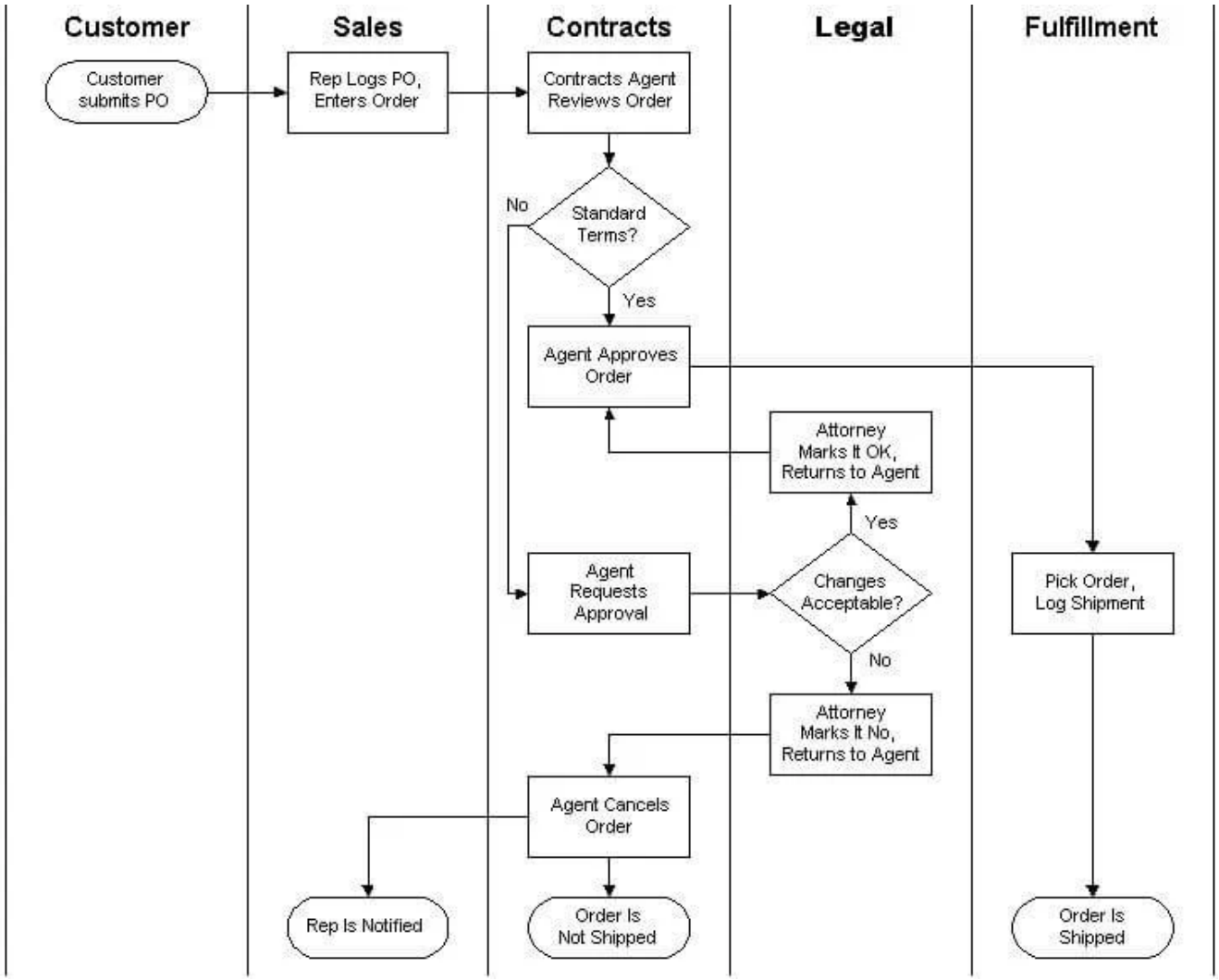
System Schema Example: Processing Pipeline



Stock and Flow Diagram



Business Process Model and Notation (BPMN)



Outline

1 Systems Thinking

2 Reference Architectures

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Design Before Code

- **Design** should come before coding.
- Jumping into *code without a plan* leads to confusion and rework.
- **Good design** clarifies the `problem` and guides the `solution`.

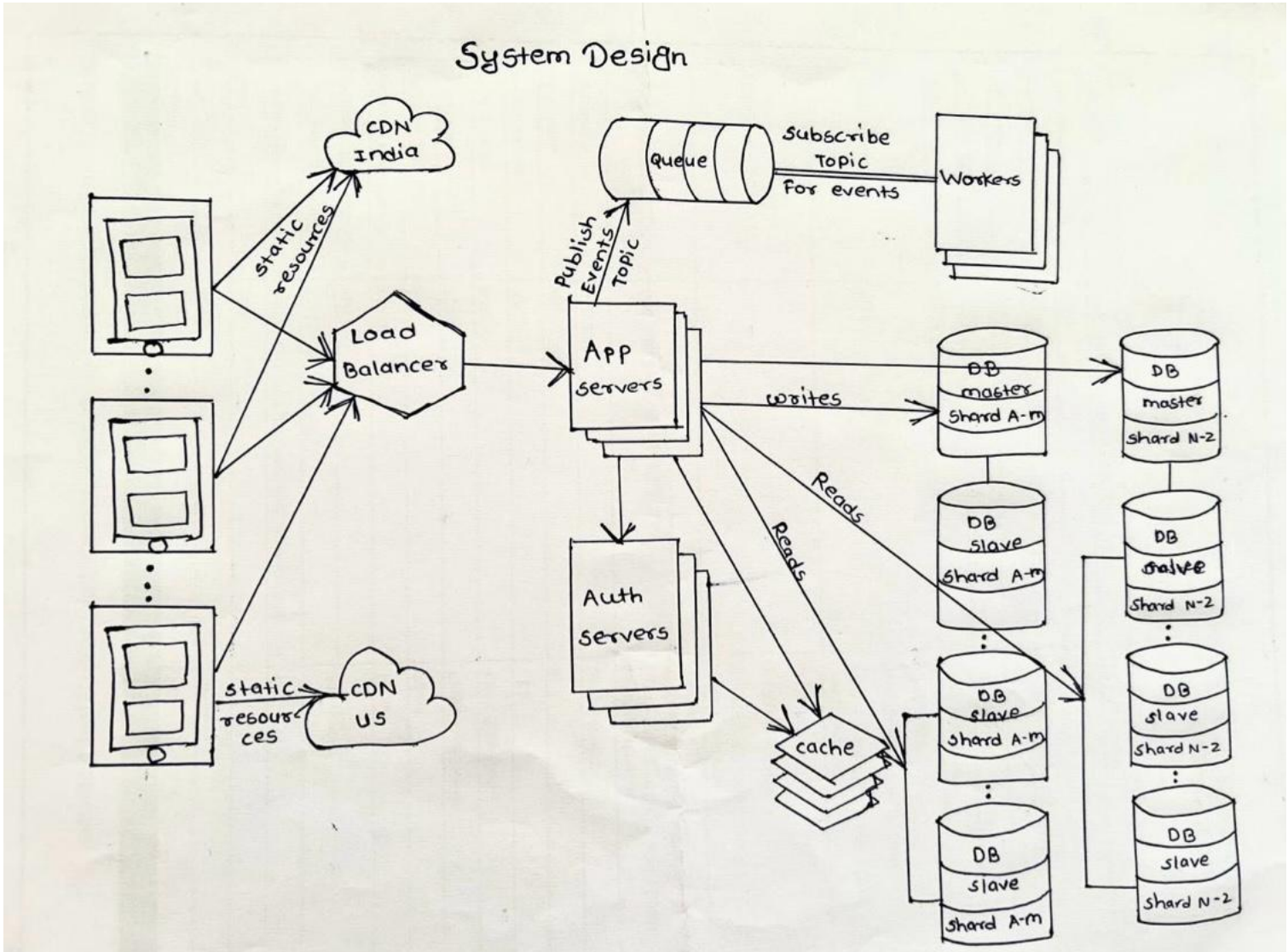
Understanding the Requirements

- **Requirements** must be well understood before design.
- `Ask questions, clarify ambiguities, and document all requirements.`
- Requirements define the scope and direction of the `design`.

Design Based on the Problem

- **Design** should be driven by the problem, not by `technology`.
- Focus on what needs to be solved, *not just* how to implement it.
- Use the **problem statement** to identify key objects and their *relationships*.

Systems Design applied to Software Architectures



Conceptual Design and Technical Design

- **Conceptual Design**: What the system should do, using `high-level models`.
- **Technical Design**: How the system will be implemented, using `detailed diagrams` and specifications.
- Both are essential for a successful software project.

Conceptual Design and Technical Design

- **Conceptual Design**: What the system should do, using `high-level models`.
- **Technical Design**: How the system will be implemented, using `detailed diagrams` and specifications.
- Both are essential for a successful software project.

Conceptual Design and Technical Design

- **Conceptual Design**: What the system should do, using `high-level models`.
- **Technical Design**: How the system will be implemented, using `detailed diagrams` and specifications.
- Both are essential for a successful software project.

What is a System Architecture?

- A **system architecture** is the *structure* of a `system` that *defines* its components, interactions, and relationships.
- A **system architecture** is the *blueprint* of a `system` that *guides* its development and implementation.
- A **system architecture** is the `foundation` of a system that ensures that it meets the needs of its users.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

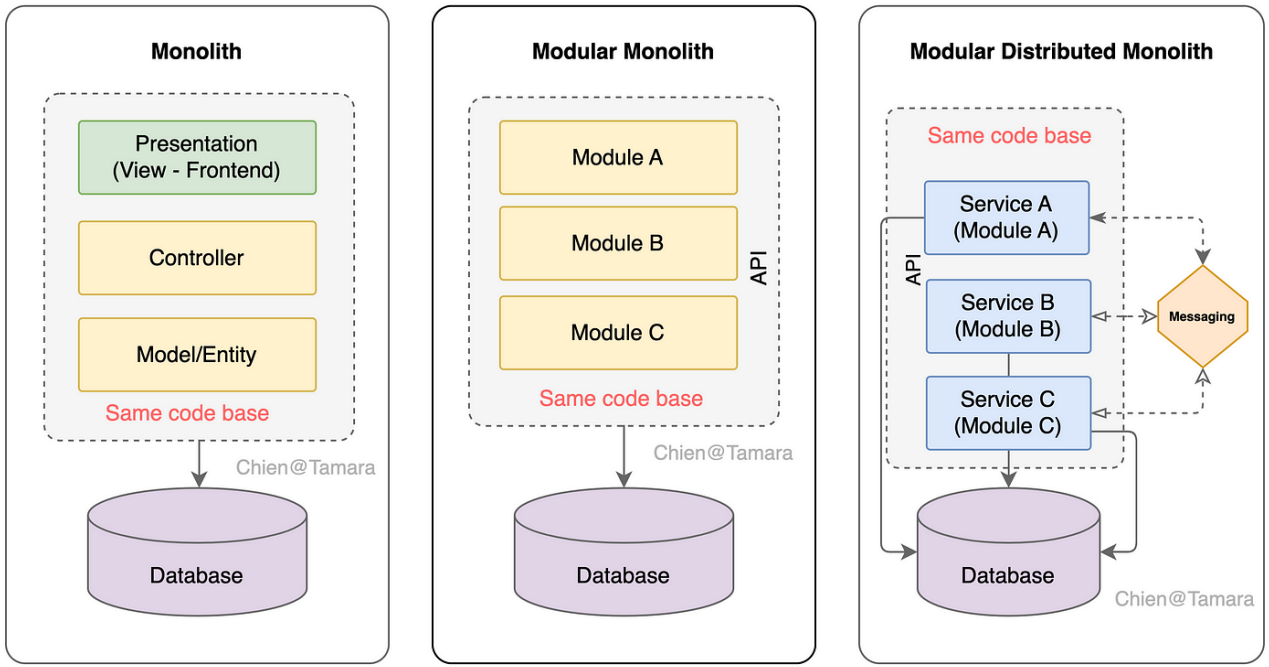
Types of System Architectures

- There are *several types* of system architectures that are *used* in `systems development`.
- They include monolithic, client-server, peer-to-peer, and distributed architectures.
- Each *type* of architecture has its own advantages and disadvantages that *depend* on the `specific requirements` of the system.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Monolithic System Architecture

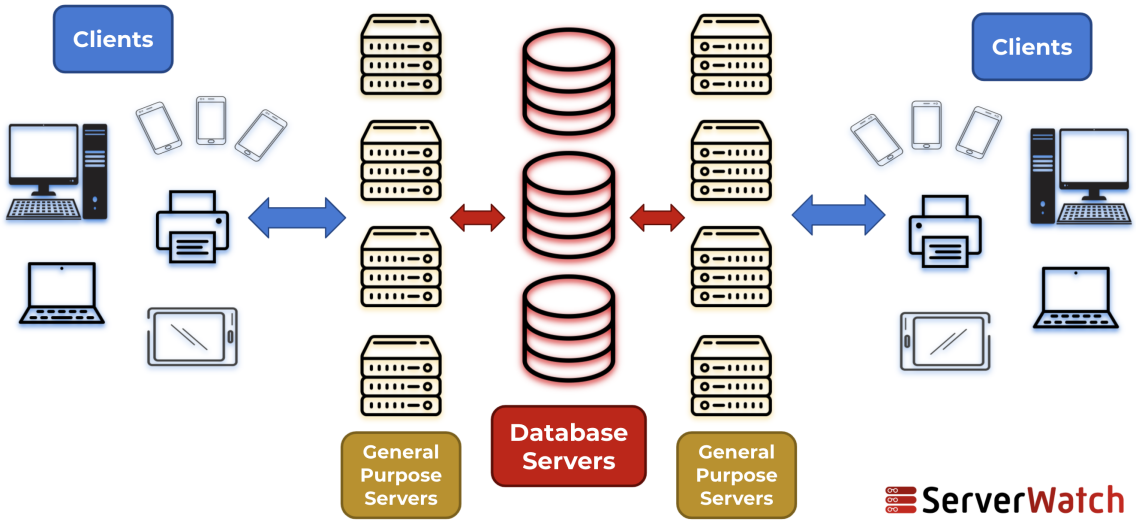
- A **monolithic system architecture** is a single-tier architecture that consists of a single unit that `performs` all the `functions` of the system.
- It is simple, easy to develop, and maintain, but it is **not** scalable and flexible. It is typically used for small systems that do not require high performance or reliability.



Client-Server System Architecture

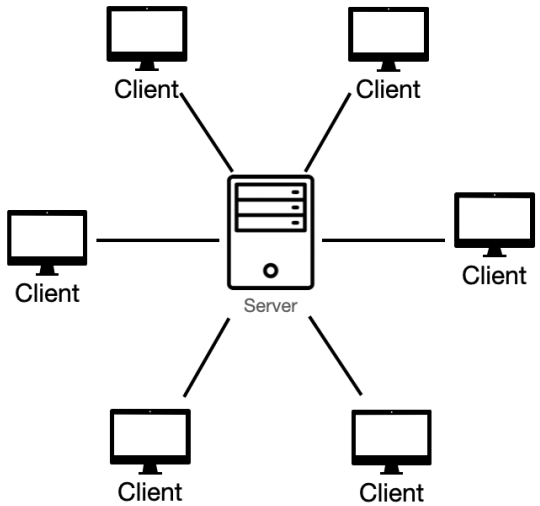
- A **client-server system architecture** is a two-tier architecture that consists of a client and a server that communicate with each other over a network.
- It is scalable, flexible, and efficient, but it is complex and difficult to develop and maintain. It is used for medium to large systems that require high performance and reliability.

The Client-Server Model

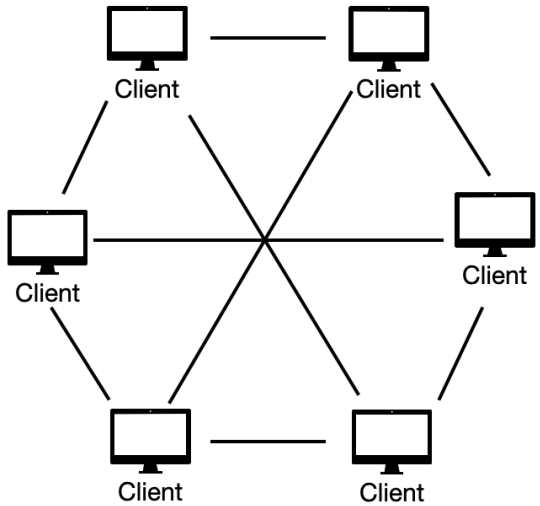


Peer-to-Peer System Architecture

- A **peer-to-peer system architecture** is a two-tier architecture that consists of a network of peers that communicate with each other directly.
- It is scalable, flexible, and efficient, but it is complex and difficult to develop and maintain.It is used for medium to large systems that require high performance and reliability.



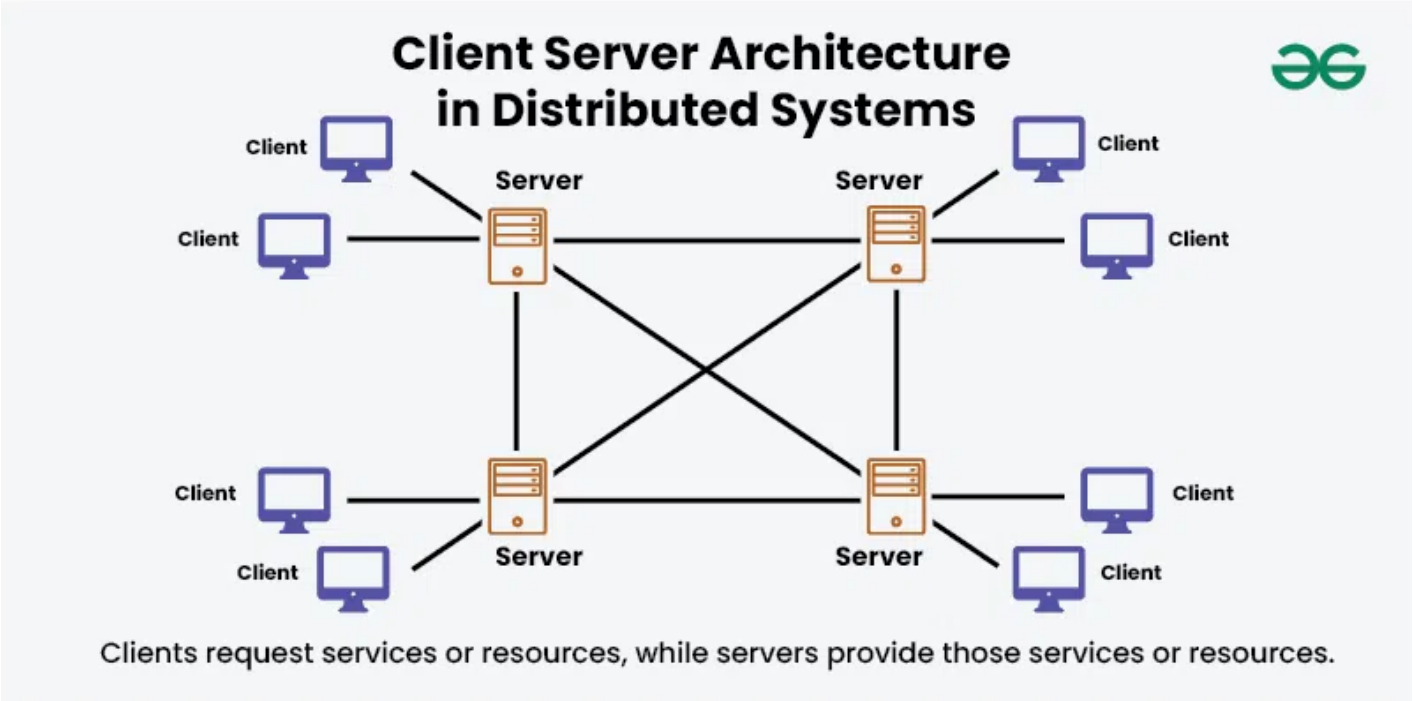
Client Server Architecture



P2P Architecture

Distributed System Architecture

- A **distributed system architecture** is a multi-tier architecture that consists of a network of nodes that communicate with each other over a network.
- It is scalable, flexible, and efficient, but it is complex and difficult to develop and maintain.It is used for large systems that require high performance and reliability.

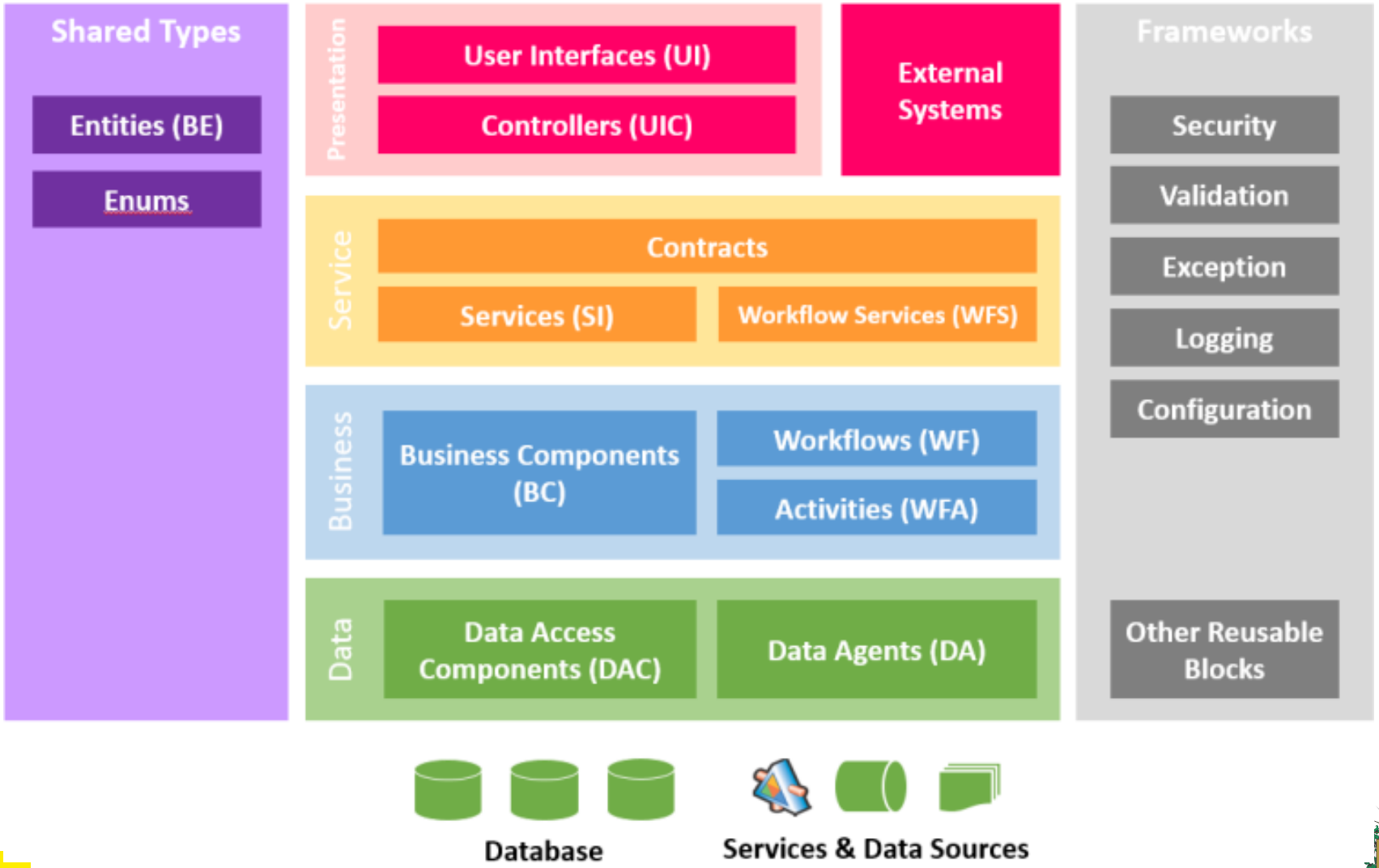


Software Architectures

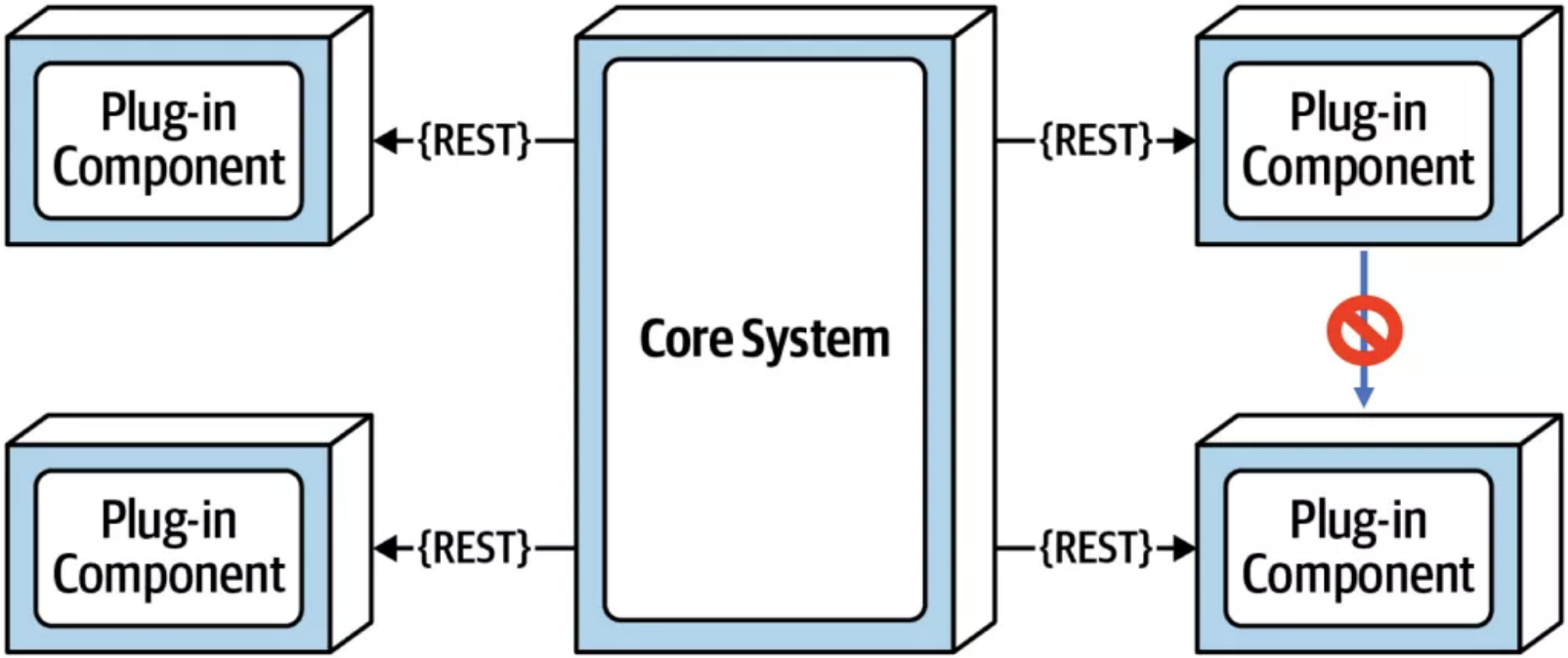
- A **software architecture** is a high-level structure of a software system that defines its components, interactions, and relationships.
- A **software architecture** is the *blueprint* of a software system that guides its development and implementation.
- A **software architecture** is the foundation of a software system that ensures it meets the `needs of its users`.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

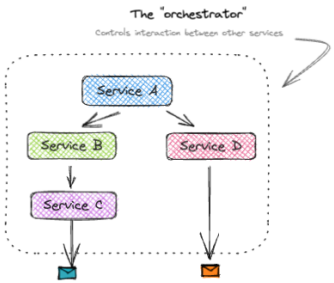
Layered Architecture Pattern



Microkernel Architecture Pattern

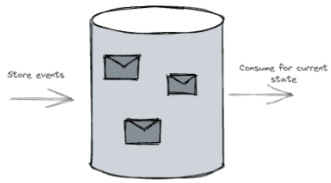


Event-based Architecture Pattern



Process Manager

Central unit that orchestrates communication
Process a workflow in a single unit



Event sourcing

Store events and consume for state
Build current state using events

Event streaming

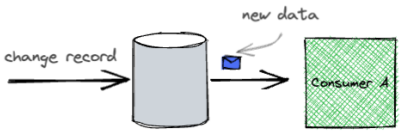
Stream of events usually used for real-time processing

Inside event-driven architectures

What patterns will you come across when building event-driven architectures?

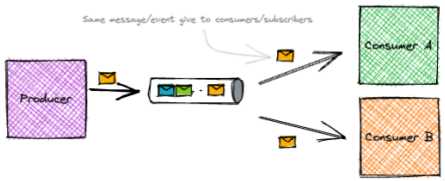
Point-to-point messaging (pull)

Messages put onto queue by sender
Receiver consumes messages from queue



Change data capture

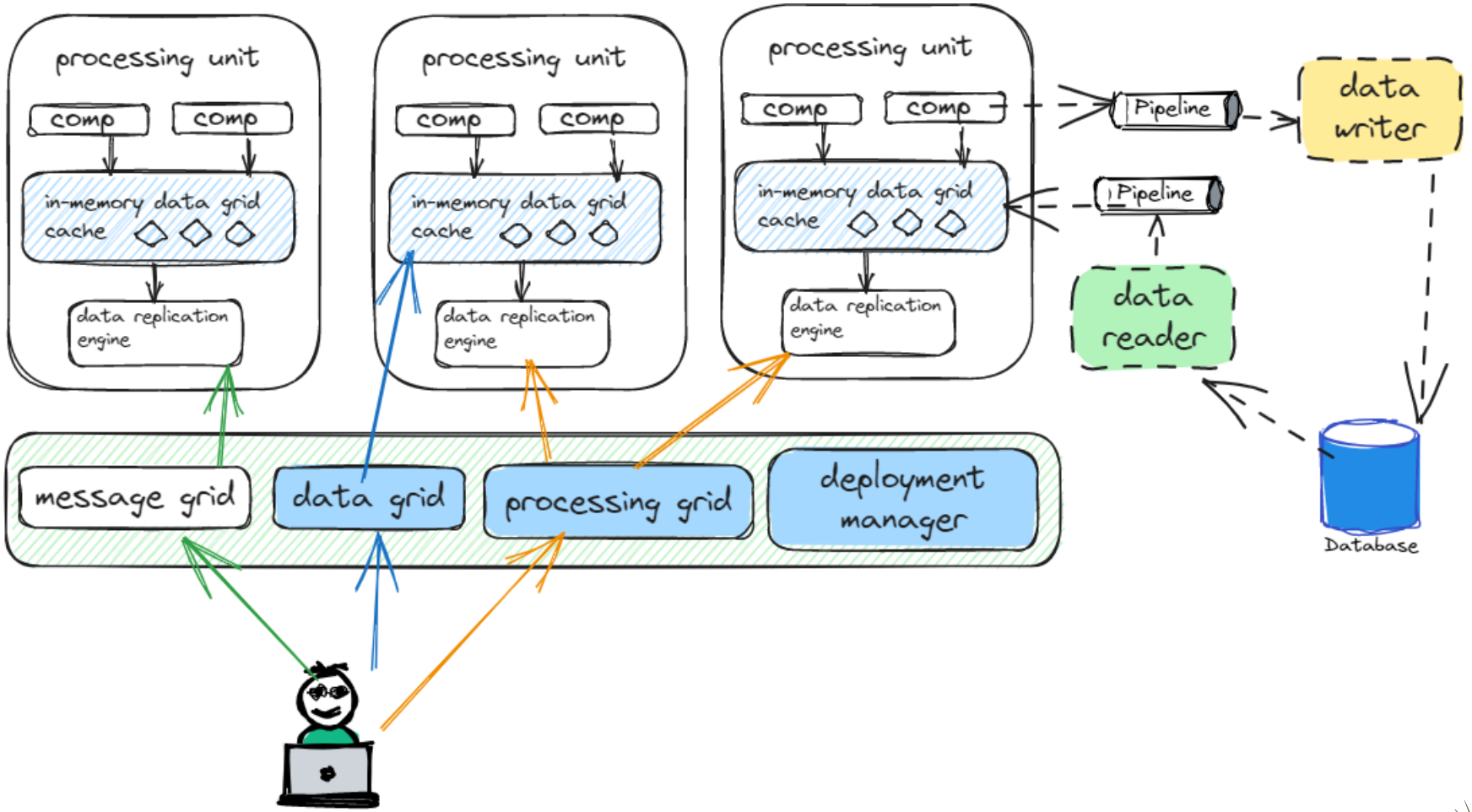
Listening to events from changes in data
Consume events directly from DB



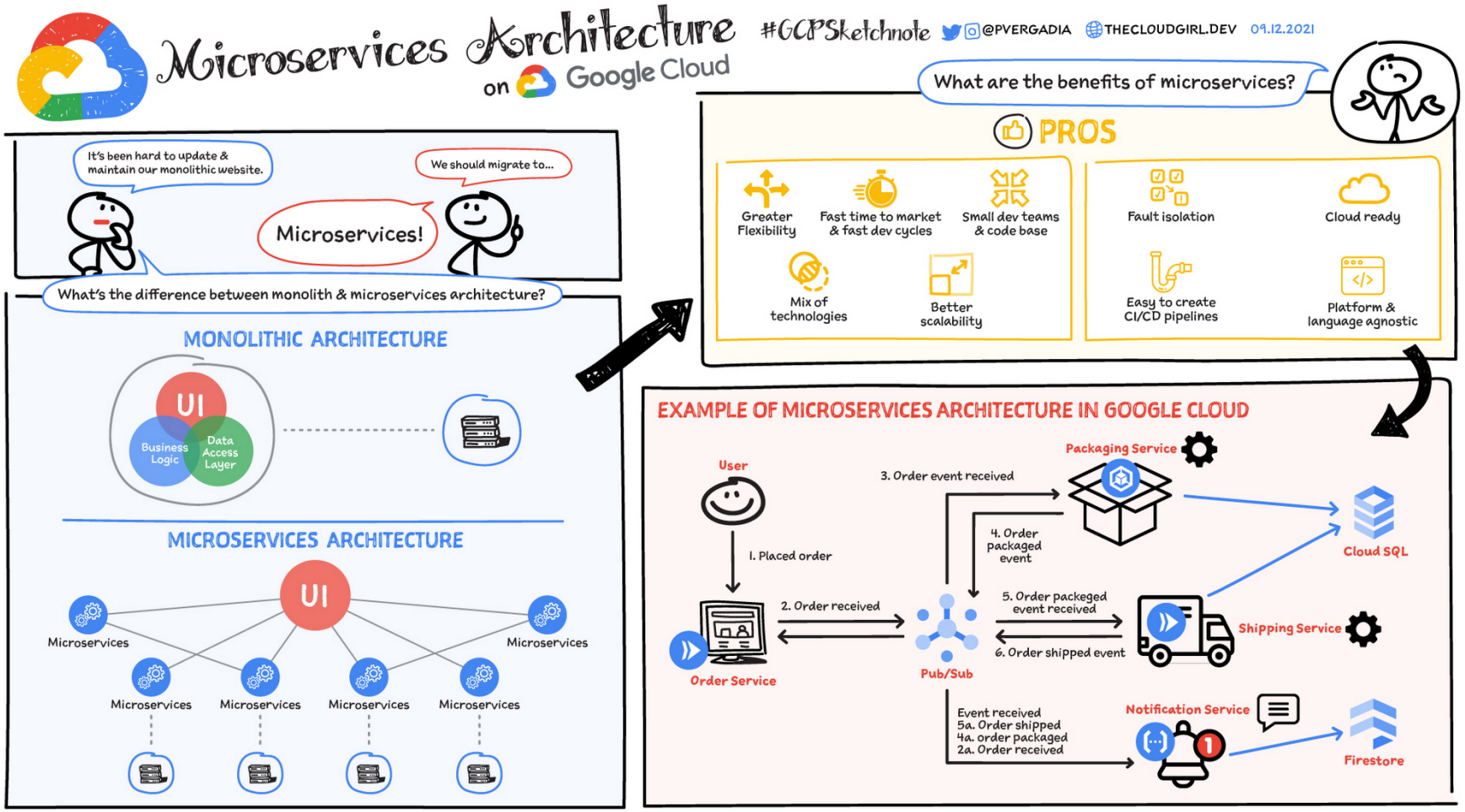
Pub/Sub (push)

Publish messages/events to many subscribers
Each subscriber gets copy of event to process

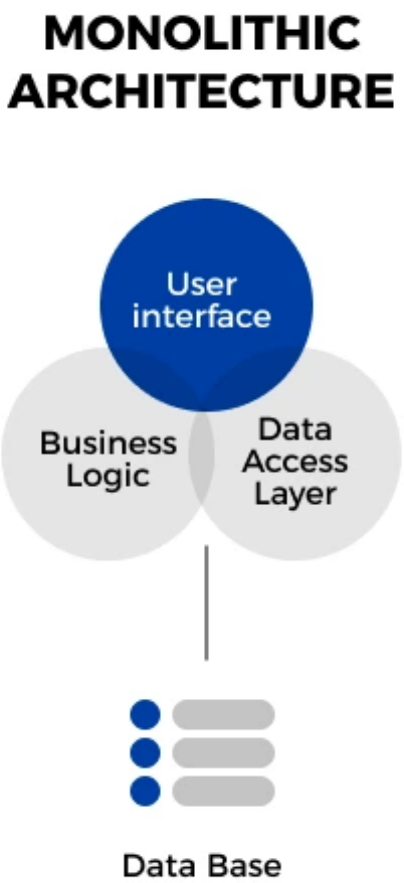
Space-based Architecture Pattern

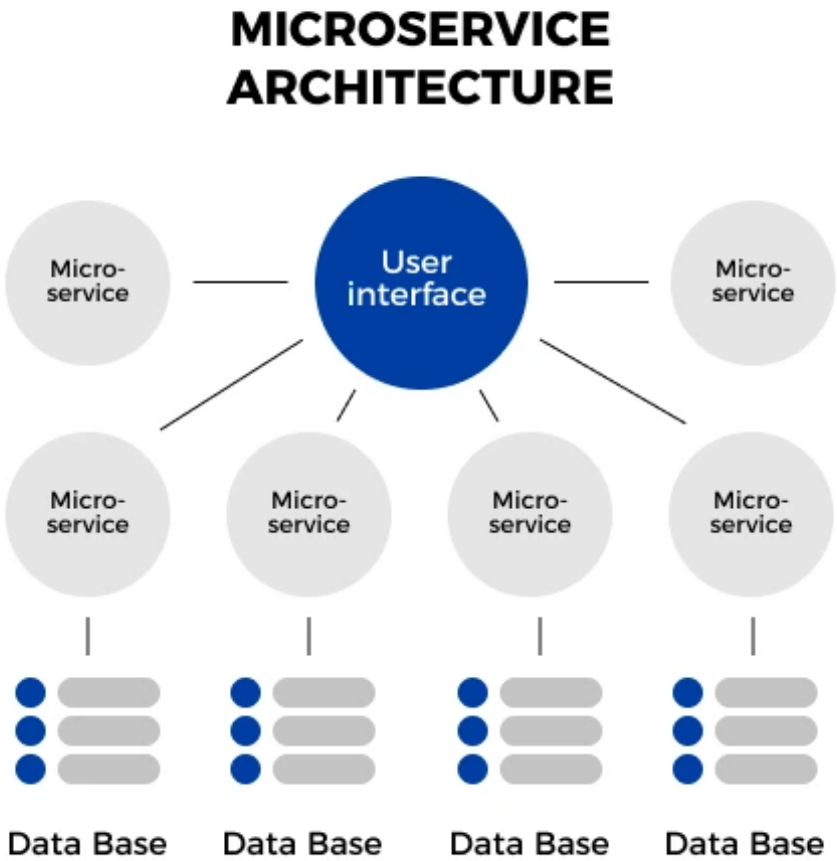


Microservices Architecture Pattern

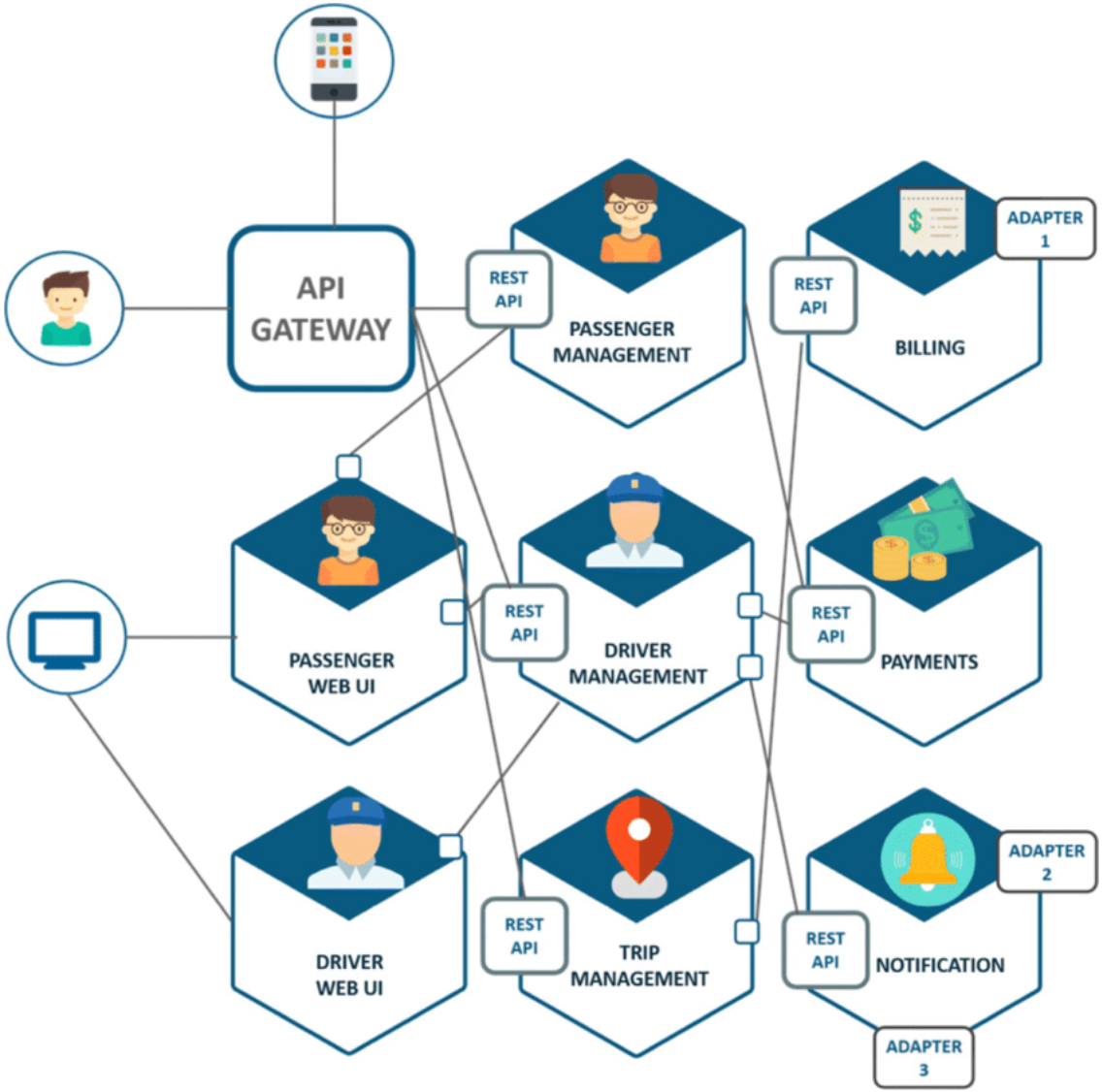


Monolithic vs. Microservices Architectures





Case Study: Uber Microservices



Source: Kappagantula 2018

Outline

1 Systems Thinking

2 Reference Architectures

Thanks!

Questions?

Repo: *www.github.com/EngAndres/ud-public/tree/main/courses/software_engineering_seminar*

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS