Indian Institute of
Information Technology
Kottayam

IOE 321 Software Design Patterns

Chapter II

Software Architectural Patterns

Presented by
Dr. Koppala Guravaiah
Assistant Professor
IIIT Kottayam

Syllabus

Architectural Patterns

- Introduction
- Layered architecture
- Pipers and Filters,
- Blackboard
- Broker
- MVC
- MVVM
- Micro-Kernel
- Master-Slave
- PAC
- others

Reference:

- Frank Buschmann, Kevlin Henney, Douglas C. Schmidt, Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Wiley, 2007.
- https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013
- https://www.simform.com/blog/software-architecture-patterns/#sectiond

Introduction

- Flaws in any software have a significant impact on the business of an organization.
- One of the main reason for any software failure can be the selection of wrong software architecture patterns
- If companies start the process of application development without a formal architecture in place.
- However, they tend to miss that the absence of an architectural pattern can force the developing team to adopt a traditional pattern with no guidelines.

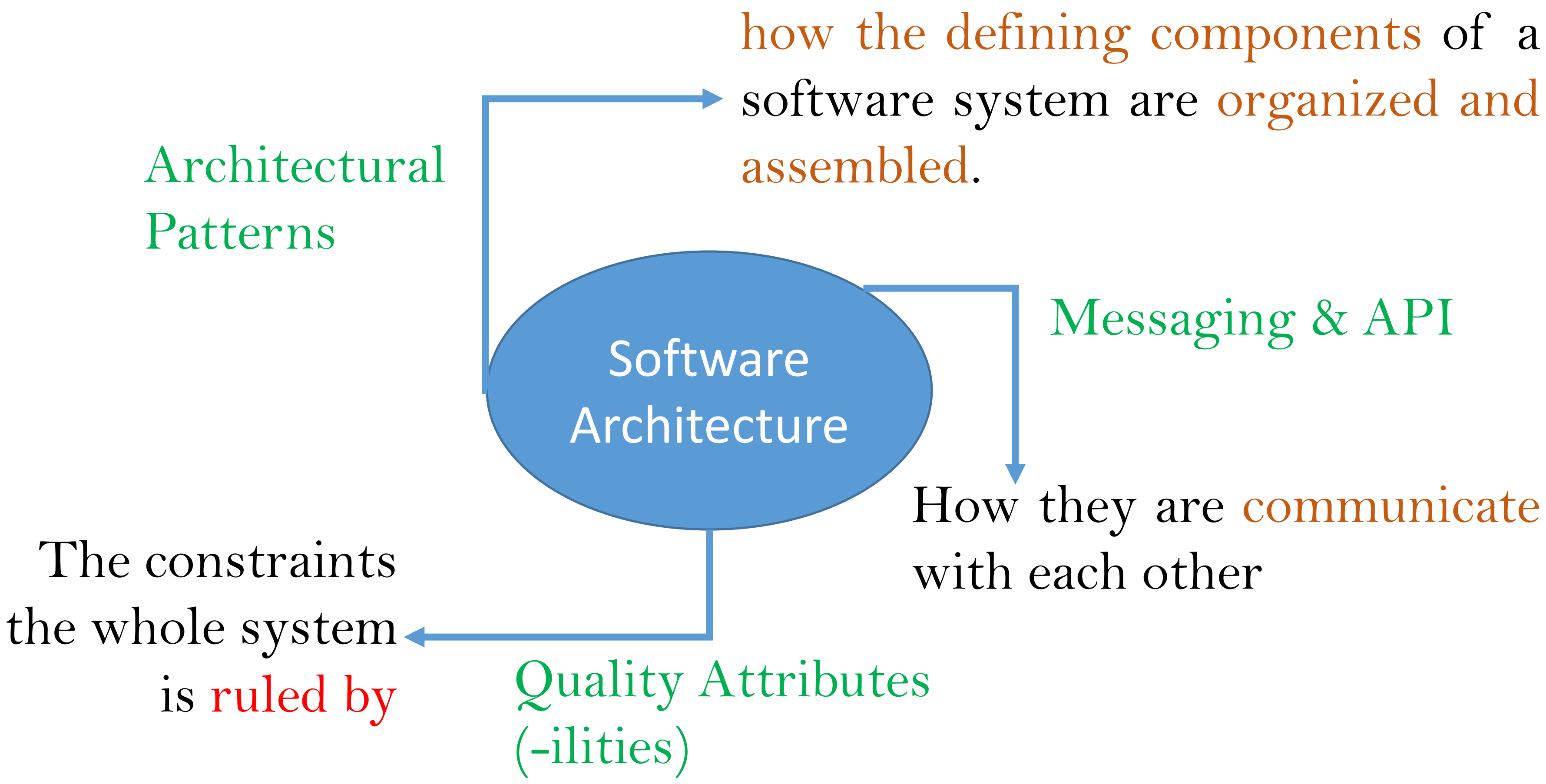
Introduction … Contd.

- Eventually, they end up with codes that lack clear roles, responsibilities, and relationships with one another.
- online banking application not require a complex architecture like a microservices pattern, It can be developed using a client-server architecture for fetching requests.

Architectural Pattern

- Software architecture is how the defining components of a software system are organized and assembled. How they are communicate with each other and the constraints the whole system is ruled by.
- An architectural pattern can be called an outline that allows you to express and define a structural schema for all kinds of software systems.
- Deals with Overall structure of the system.
- Defines the granularity of the components

Architectural Pattern … Contd.



Architectural Patterns

how the defining components of a software system are organized and assembled.

Software Architecture

Messaging & API

How they are communicate with each other

Quality Attributes (-ilities)

The constraints the whole system is ruled by

Architectural Pattern … Contd.

- It's a reusable solution that provides
  - a predefined set of subsystems,
  - roles, and responsibilities, including the rules and roadmap for defining relationships among them.
- It helps you address various software engineering concerns such as
  - Performance limitations
  - High availability
  - Minimizing business risk

Architectural Pattern … Contd.

- Patterns are known as "strictly described and commonly utilized".
- The success of the system depends on software architecture selection.
- These patterns hold significant importance for it can solve various problems within different domains
  - For instance, instead of depending on a single server, complex user requests can be easily segmented into smaller chunks and distributed across multiple servers.
  - In another example, testing protocols can be simplified by dividing various segments of the software rather than testing the whole thing at once.

Why Architectural Pattern

- **Defining Basic Characteristics of an Application**
- **Maintaining Quality and Efficiency**
- **Providing Agility**
- **Problem Solving**
- **Enhancing Productivity**

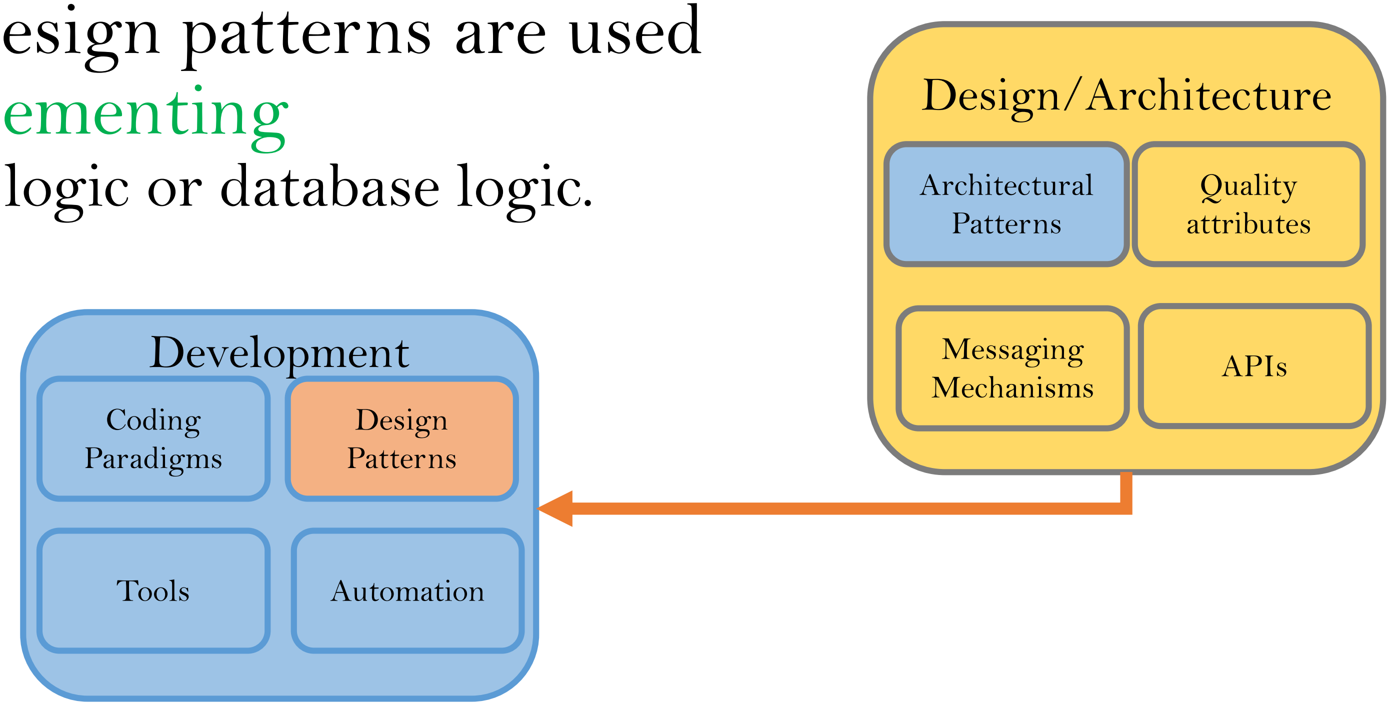
Software architectural pattern vs. design pattern

- A thin line between an architecture pattern and a design pattern to differentiate
- For basics, let's imagine your team given a task to build a house and live in it
  - First have to plan it out before placing bricks and cement on an empty ground
  - After a house is planned, there is more to making it worth living – they would need basic amenities like kitchen appliances, beddings, toiletries, and many more.
  - In this analogy, how the house should look represents architectural patterns, whereas the interior design of the house represents the design patterns.

Architectural pattern vs. design pattern … Contd.

In a software system,

- Software Architecture is considered when you have to create
  - business logic, database logic, UI, etc.,
- Software design patterns are used while implementing
  - business logic or database logic.



Architectural pattern vs. design pattern … Contd.

	Architecture Patterns	Design Patterns
Definition	Fundamental structural organization for software systems	Specification that could help in implementation of a software
Role	Conversion of software characteristics to a high level structure	Description of all the units of software system to support coding
Level	Large level tool – concerns large scale components, global properties, and mechanisms of the system	Small level tool: concerns schemes for refining and building smaller sub systems- structure behavior of entities and their relation ships
Problem Addressed	Distributed functionality, system partitioning, protocols, interfaces, scalability, reliability, security	Problems in software construction

Architectural pattern vs. design pattern … Contd.

	Architecture Patterns	Design Patterns
Scope	High Level, Universal Scope	Low level Scope
designed	How components are organized and assembled	How Components are built
Example	Microservices, Server-less, and Event-driven	Creational, Structural, Behavioral

Architectural patterns

- Layered Architecture
- MVC
- PAC Presentation
- Microkernel Architecture
- Pipe-Filter Architecture
- Blackboard
- Broker Architecture
- Master-Slave Architecture
- MVVM

- Others
  - Client-Server Architecture
  - Interpreter
  - Microservices Architecture
  - Peer-to-Peer Architecture
  - Space-Based Architecture

Domain Model

- The DOMAIN MODEL pattern defines a precise model for the structure and workflow of an application domain including their variations.
- Model elements are abstractions meaningful in the application domain; their roles and interactions reflect domain workflow and map to system requirements

Domain Model … Contd.

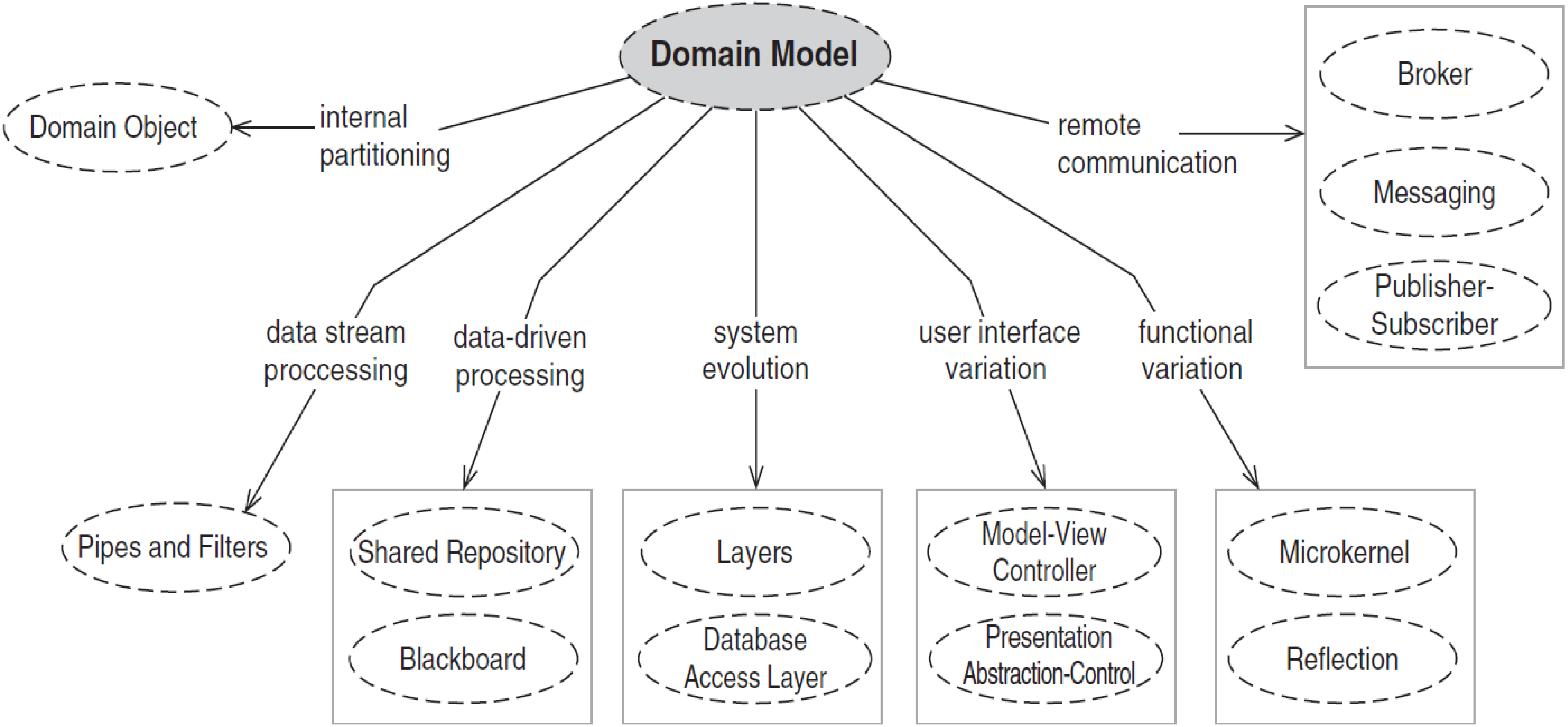


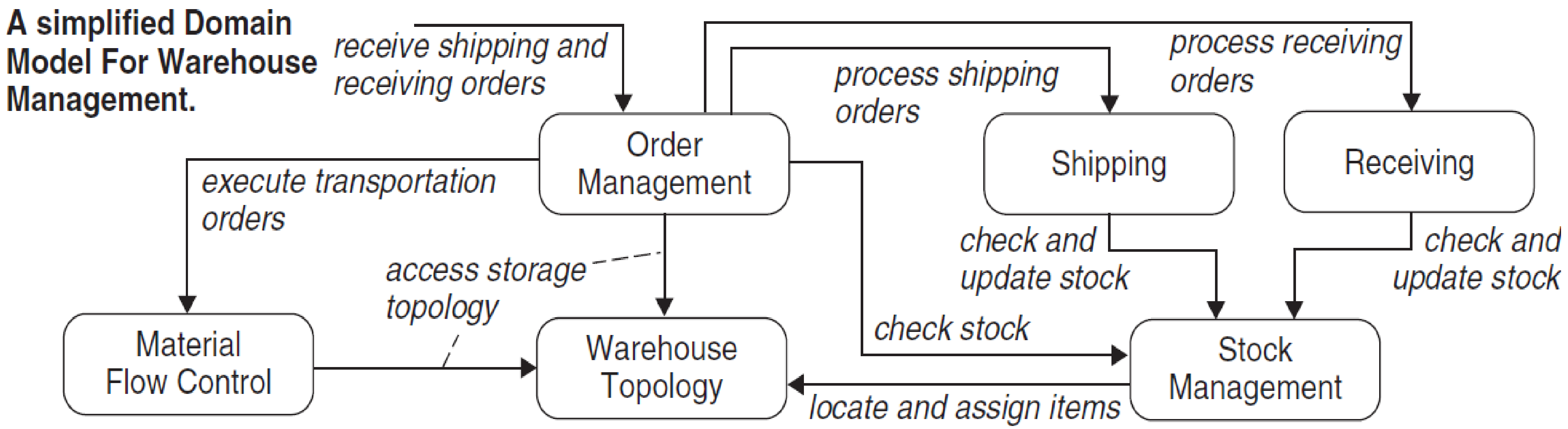
Figure: Domain Model connects to the body of our pattern language

Domain Model … Cond.

- Finding a suitable application partitioning depends on framing answers to several key questions and challenges:
  - *How does the application interact with its environment?*
  - *How is application processing organized?*
  - *What variations must the application support?*
  - *What is the life expectancy of the application?*
- When starting to build a (distributed) application, need an initial structure for the software being developed
- **Requirements and constraints** inform the functionality, quality of service, and deployment aspects of a software system, but do not themselves suggest a concrete structure to guide development.

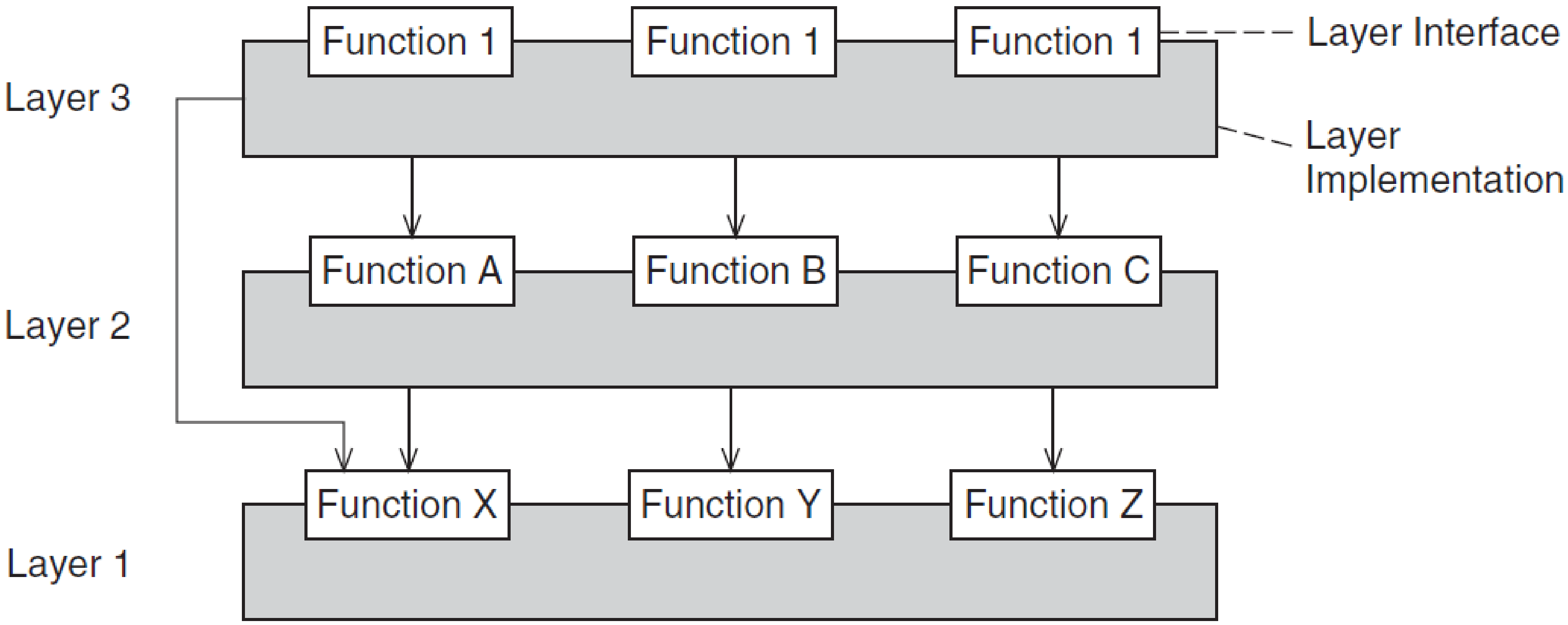
Domain Model … Cond.

- A list of requirements shows the problem domain of an application, but not its solution domain
- Example: Domain Model for warehouse management



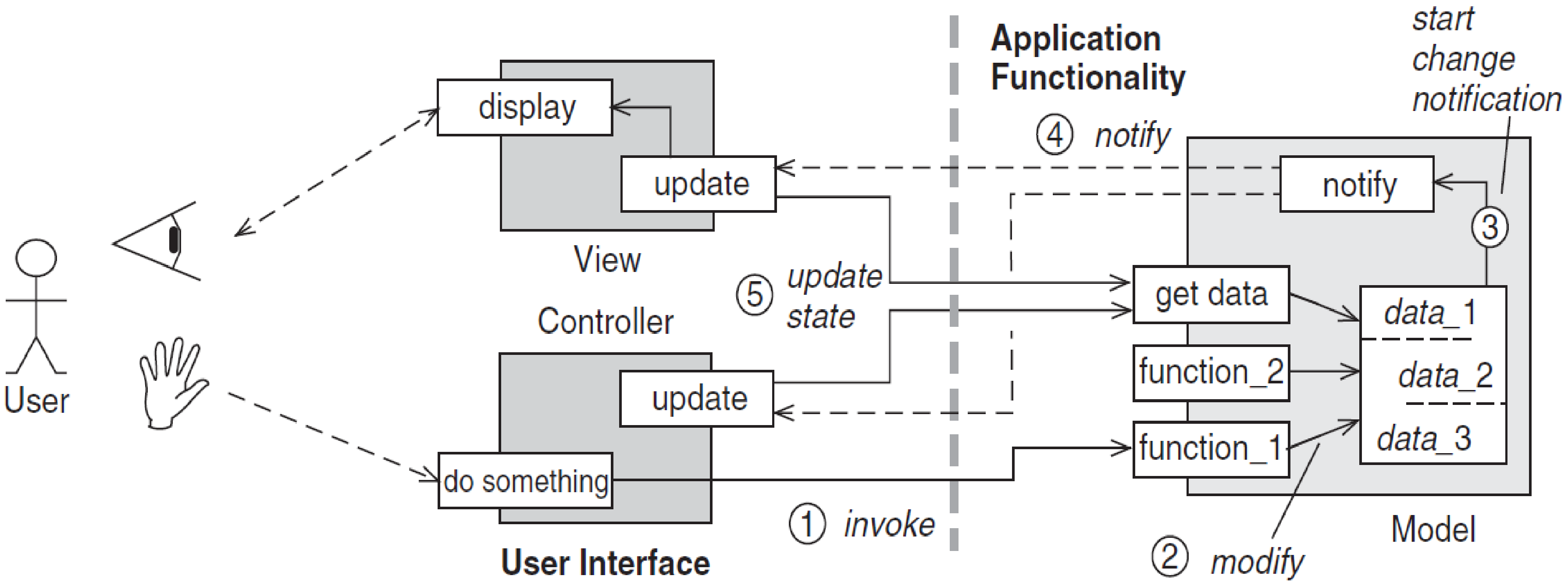
Layered Architecture

- Define one or more layers for the software under development, with each layer having a distinct and specific responsibility



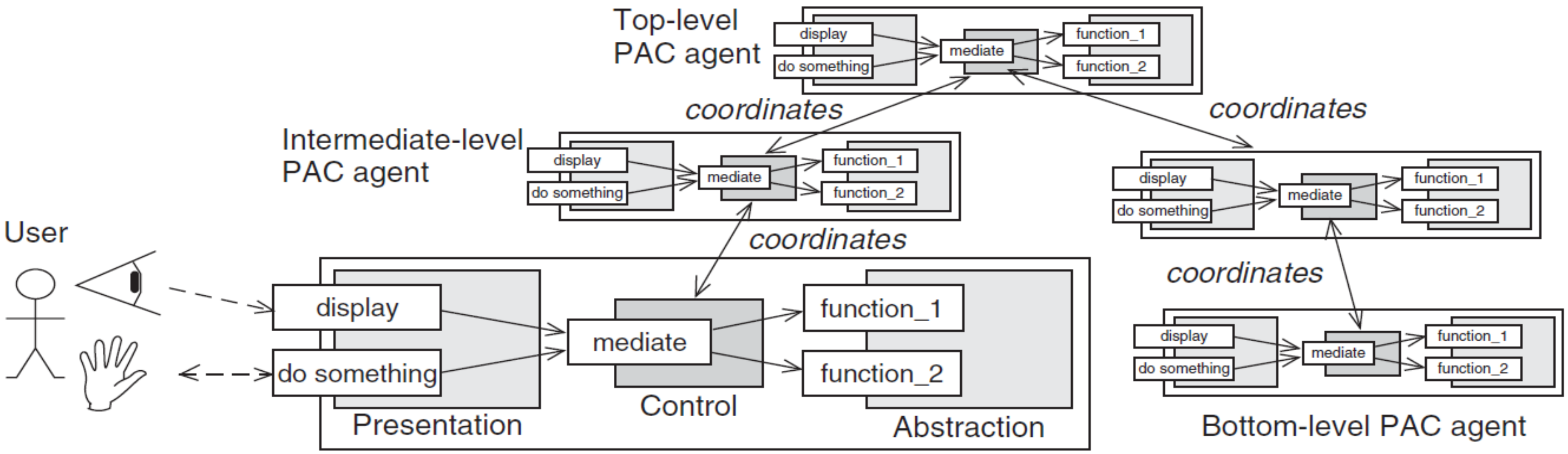
Model-View-Controller

- Divide the interactive application into three decoupled parts: **processing, input, and output**. Ensure the consistency of the three parts with the help of a change propagation mechanism



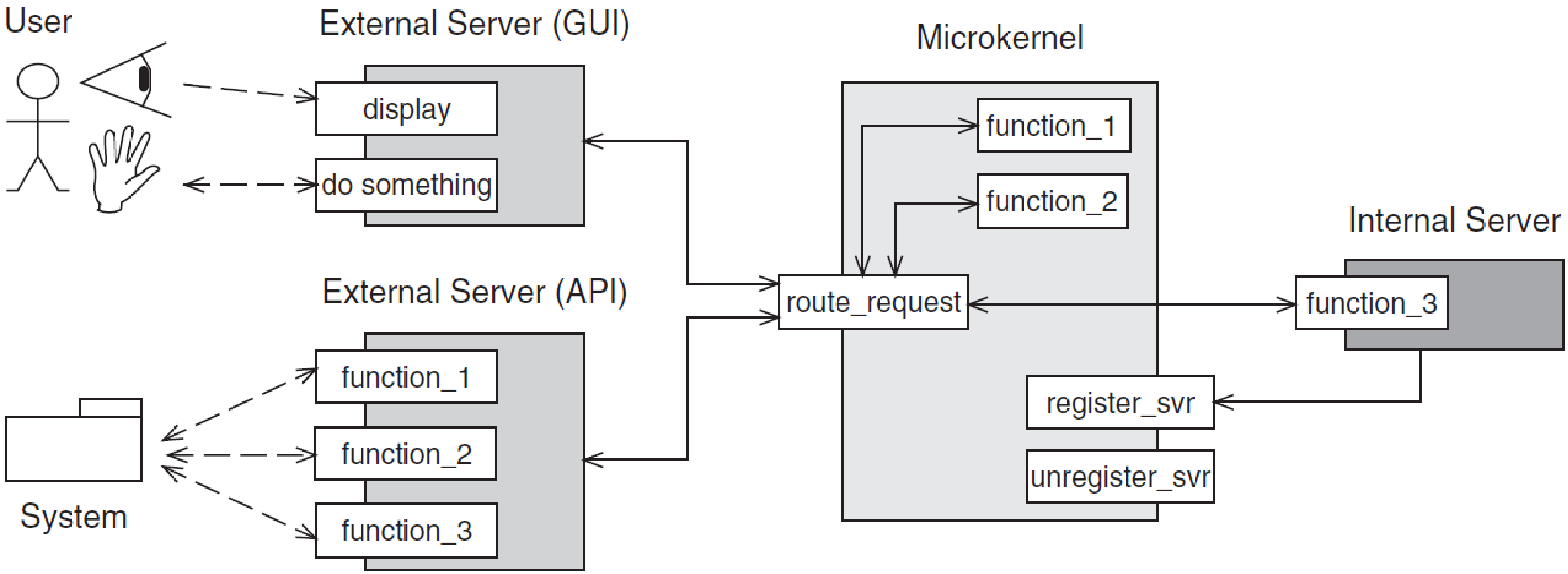
Presentation-Abstraction-Control

- Structure the interactive application as a hierarchy of decoupled agents: **one top-level agent, several intermediate-level agents, and many bottom-level agents**. Each agent is responsible for a specific functionality of the application and provides a specialized user interface for it.



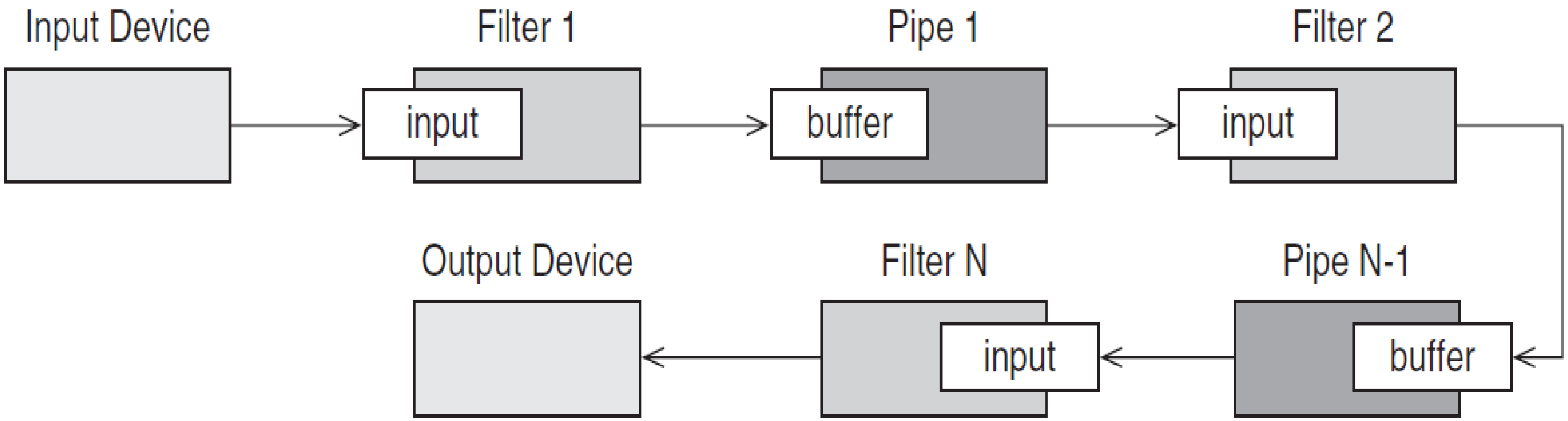
Microkernel

- Compose different versions of the application by extending a common but minimal core via a ‘plug-and-play’ infrastructure



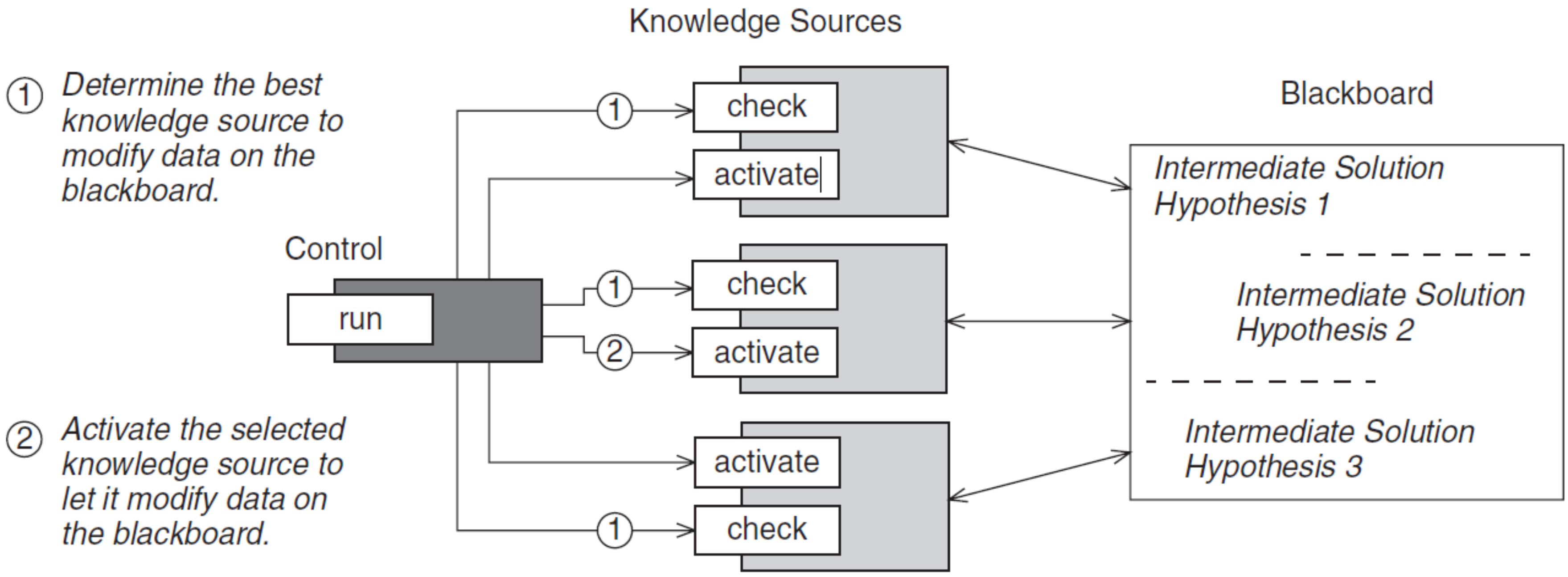
Pipes and Filters

- Divide the application's task into several self-contained data processing steps and connect these steps to a data processing pipeline via intermediate data buffers



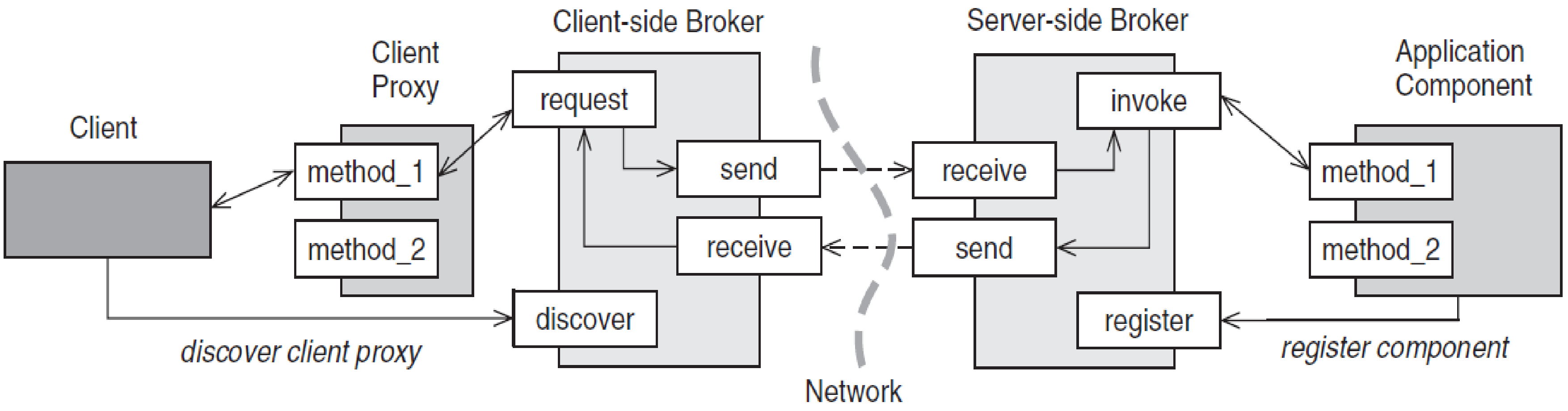
Blackboard

- Use heuristic computation to resolve the task via multiple smaller components with deterministic solution algorithms that gradually improve an intermediate solution hypothesis.


Knowledge Sources
① Determine the best knowledge source to modify data on the blackboard.
② Activate the selected knowledge source to let it modify data on the blackboard.
Control
run
1
2
check
activate
check
activate
activate
check
Blackboard
Intermediate Solution Hypothesis 1
Intermediate Solution Hypothesis 2
Intermediate Solution Hypothesis 3

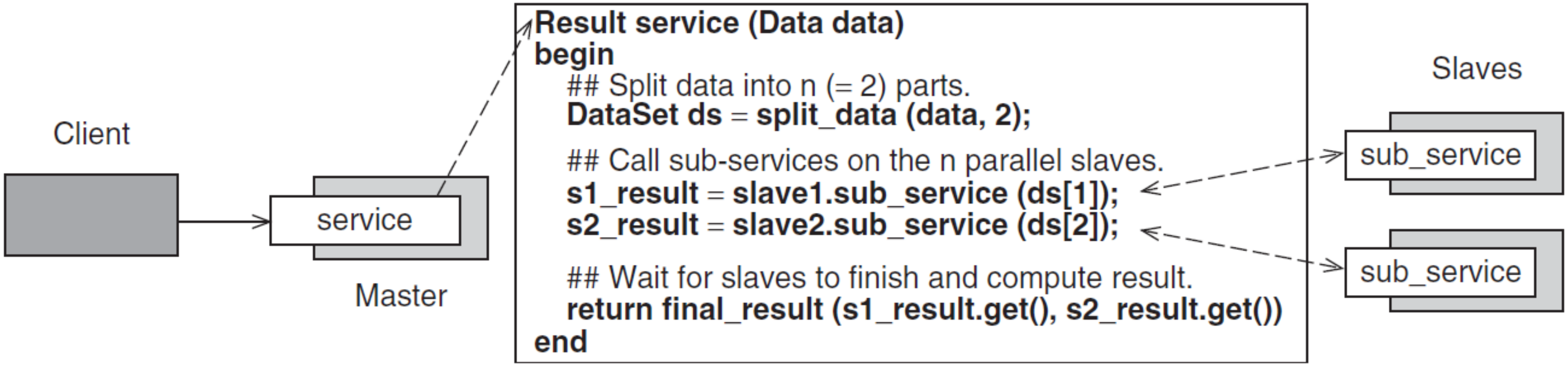
Broker

- Use a federation of brokers to separate and encapsulate the details of the communication infrastructure in a distributed system from its application functionality.
- Define a component based programming model so that clients can invoke methods on remote services as if they were local



Master-Slave

- Meet the performance, fault-tolerance, or accuracy requirements of the component via a ‘**divide and conquer**’ strategy.
- Split its services into independent subtasks that can be executed in parallel, and combine the partial results returned by these subtasks to provide the service’s final result.



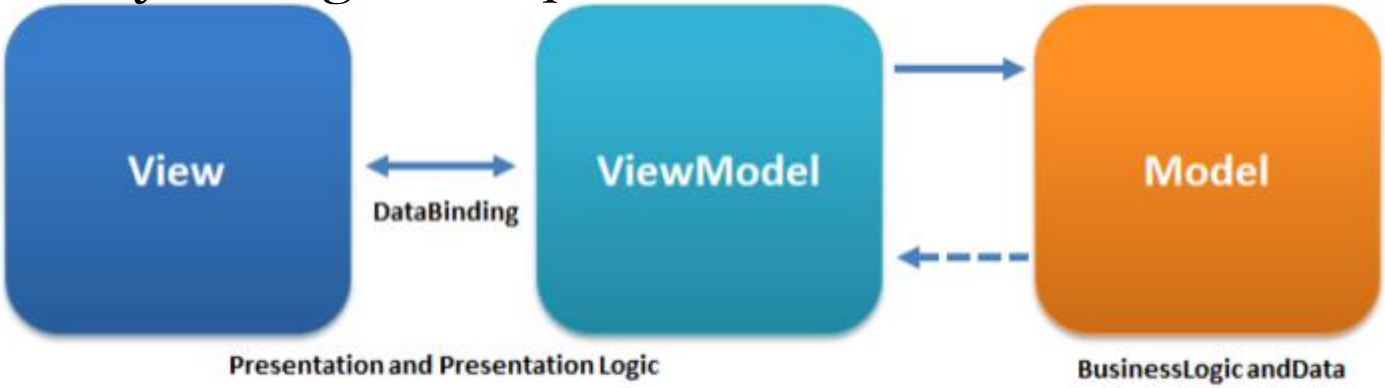
```
Result service (Data data)
begin
   ## Split data into n (= 2) parts.
   DataSet ds = split_data (data, 2);

   ## Call sub-services on the n parallel slaves.
   s1_result = slave1.sub_service (ds[1]);
   s2_result = slave2.sub_service (ds[2]);

   ## Wait for slaves to finish and compute result.
   return final_result (s1_result.get(), s2_result.get())
end
```

MVVM Architecture

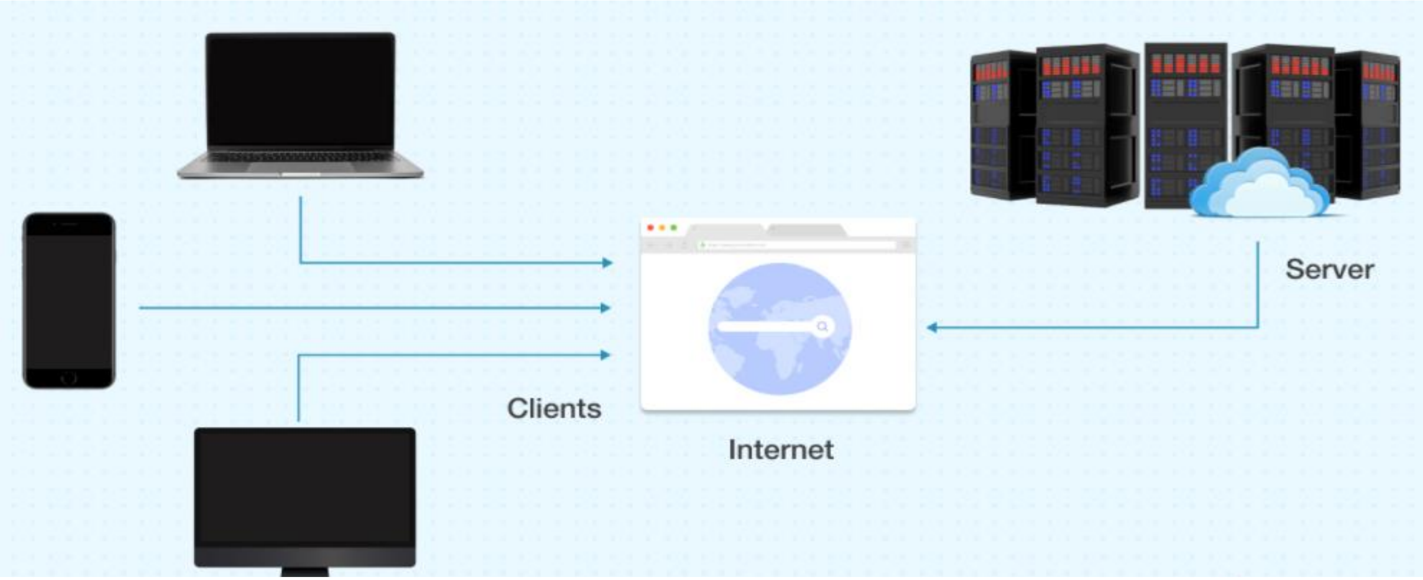
- **Model–view–viewmodel** (**MVVM**) is the separation of the development of the graphical user interface (the *view*) – be it via a markup language or GUI code – from the development of the business logic or back-end logic (the *model*) so that the view is not dependent on any specific model platform.
- The *viewmodel* is a value converter, responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented



Ref: https://en.wikipedia.org/wiki/Model-view-viewmodel

Client-Server Architecture

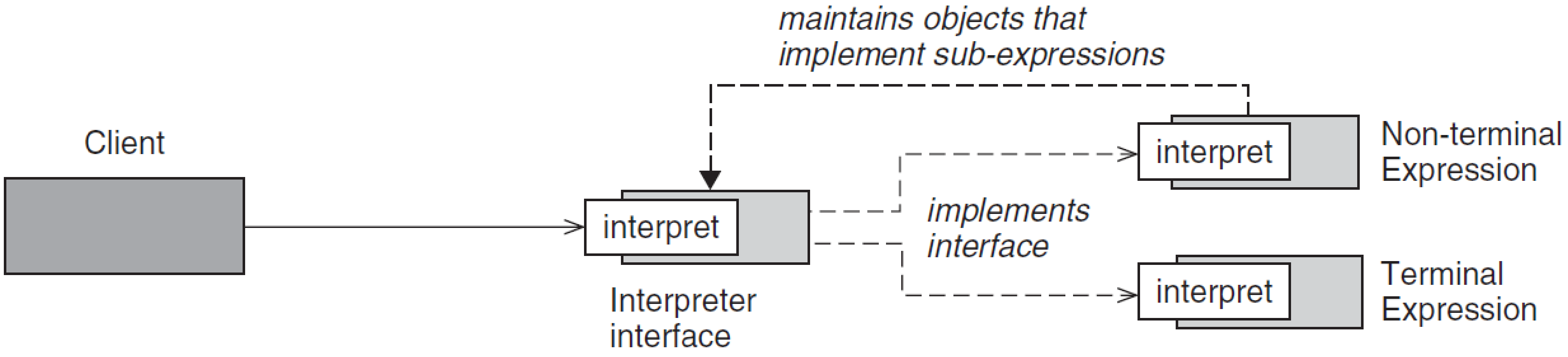
- A client-server architecture pattern is described as a distributed application structure having two main components – a client and a server.
- This architecture facilitates the communication between the client and the server, which may or may not be under the same network



Ref: https://www.simform.com/blog/software-architecture-patterns/#sectiond

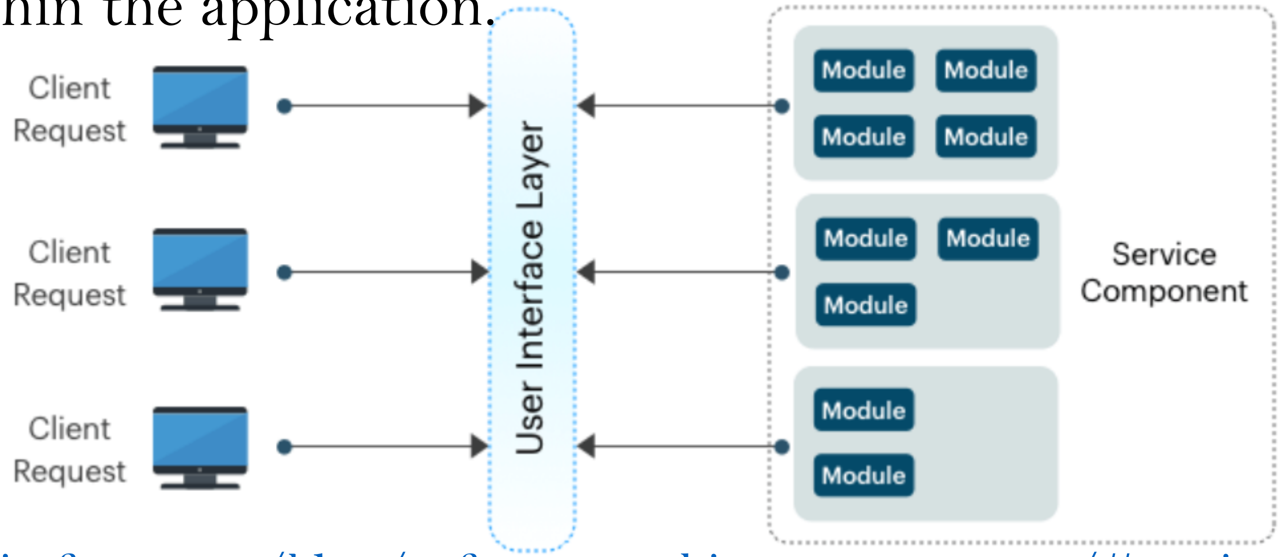
Interpreter

- Introduce an interpreter that represents the grammar of the language and its execution.
- The interpreter is a whole-part hierarchy of classes, typically with one class per grammar rule



Micro-Services

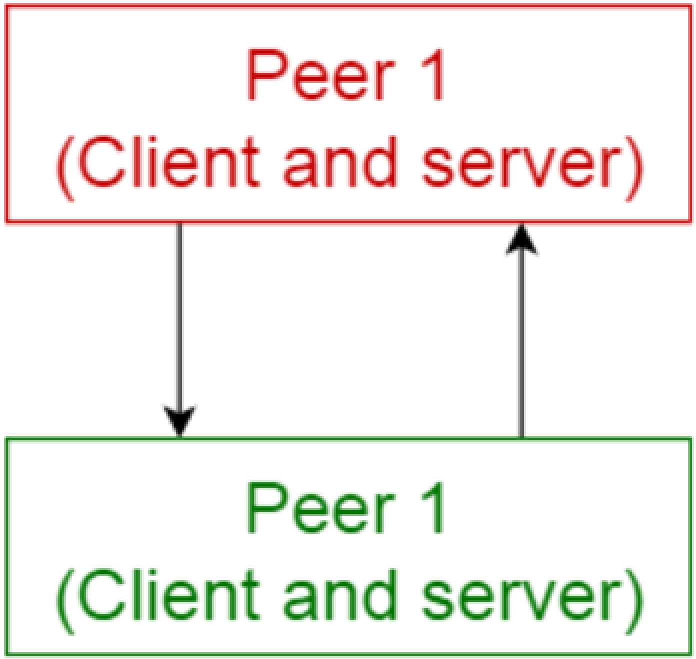
- Microservices architecture pattern is seen as a viable alternative to monolithic applications and service-oriented architectures.
- The components are deployed as separate units through an effective, streamlined delivery pipeline.
- The pattern's benefits are enhanced scalability and a high degree of decoupling within the application.



Ref: https://www.simform.com/blog/software-architecture-patterns/#sectiond

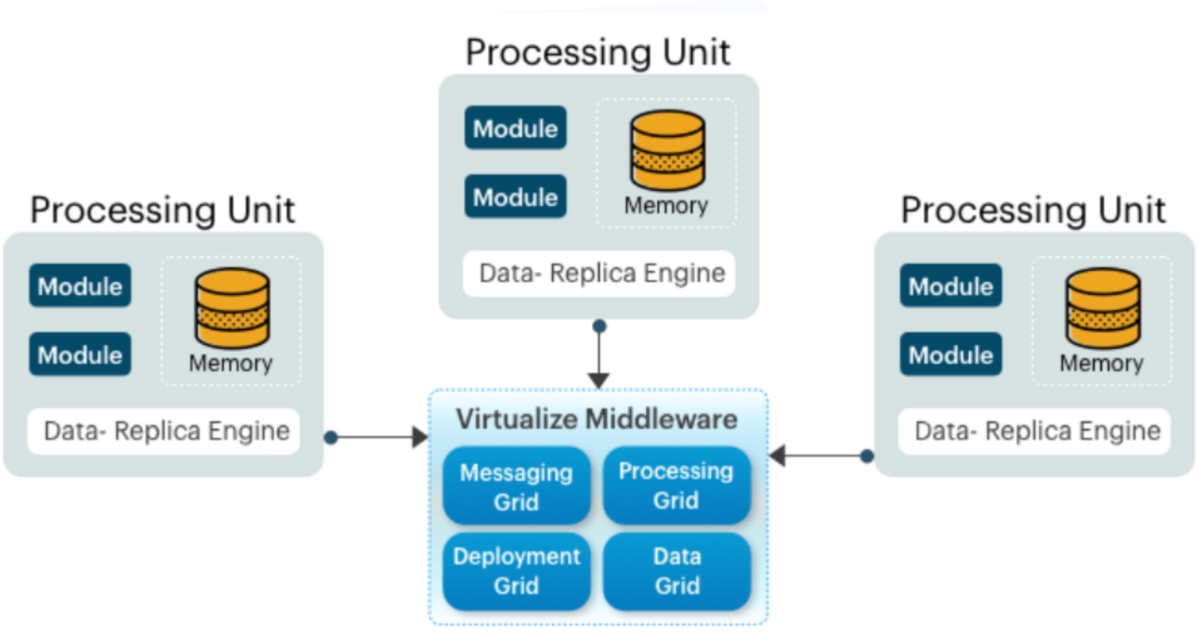
Peer-to-Peer Architecture

- Individual components are called peers. A peer can act as a client, a server, or both and change its role dynamically over time.
- As a client, a peer can request service from other peers, and as a server, a peer can provide services to other peers.
- The significant difference between peer-to-peer and client-server architecture is that each computer on the network has considerable authority and the absence of a centralized server.
- **Example:** file-sharing networks like Skype, BitTorrent, and Napster.



Space-Based Architecture

- The concept of tuple space – the idea of distributed shared memory is the basis of the name of this architecture.
- The space-based pattern comprises two primary components – a processing unit and a virtualized middleware



Ref: https://www.simform.com/blog/software-architecture-patterns/#sectiond

Architectural patterns

- Architectural Pattern
- Architectural vs design patterns
- Different Architectural patterns
    - Layered Architecture
    - MVC
    - PAC Presentation
    - Microkernel Architecture
    - Pipe-Filter Architecture
    - Blackboard
    - Broker Architecture
    - Master-Slave Architecture
    - MVVM
- Others
    - Client-Server Architecture
    - Interpreter
    - Microservices Architecture
    - Peer-to-Peer Architecture
    - Space-Based Architecture

Thank you