SOFTWARE ENGINEERING (Week-3)

USAMA MUSHARAF

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

CONTENTS OF WEEK # 3

Intro to Software Architecture

- Conceptual Model of Architecture Representation
- Architectural Views
- Views and View Point
- 4+1 View Model
- Discussion on Uber Case Study (System Design)

Architectural Styles

Categories of Architectural Style

- Hierarchical Software Architecture
  - Layered
- Data Flow Software Architecture
  - Pipe and Filter
  - Batch Sequential

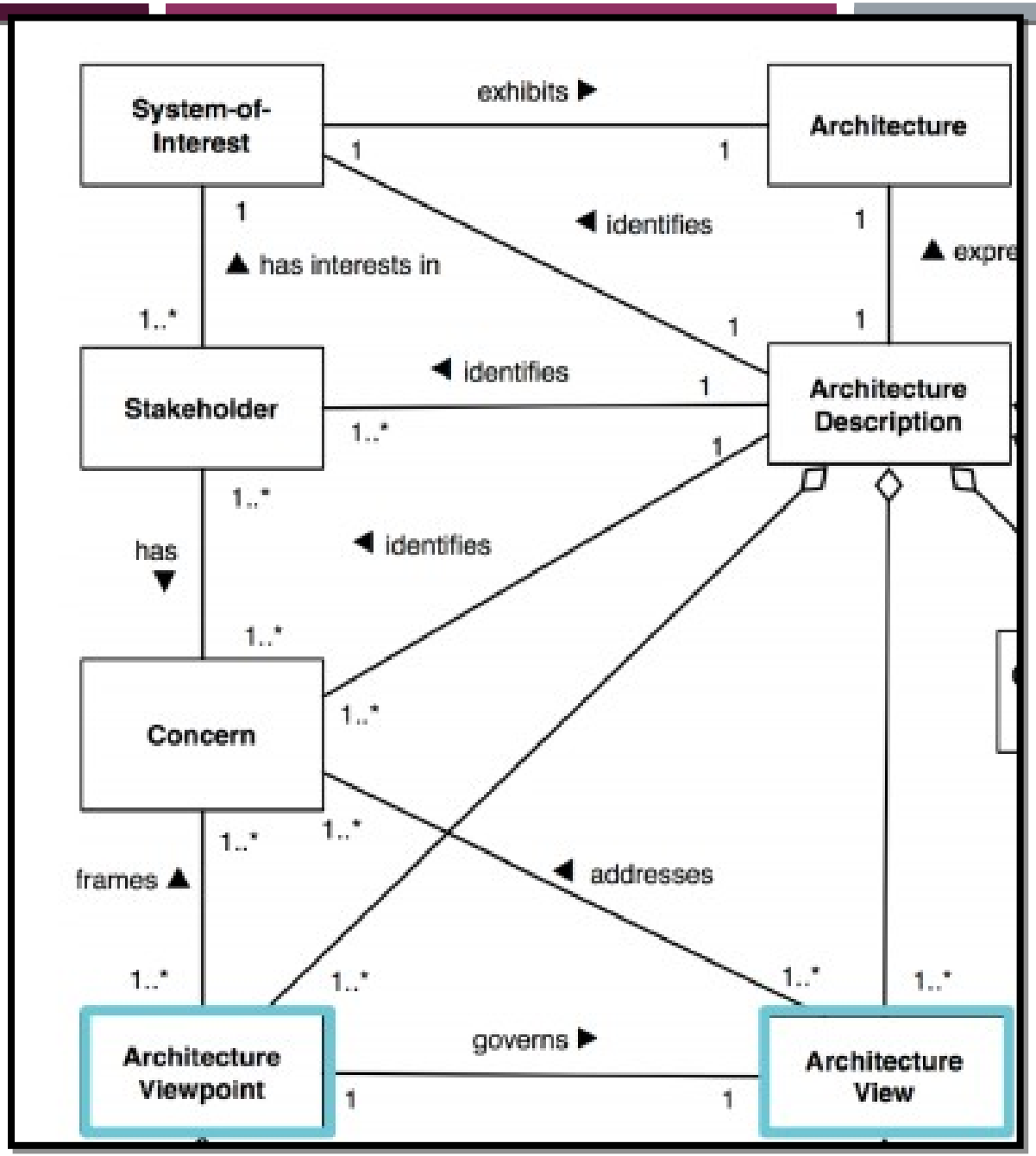
SOFTWARE ARCHITECTURE

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them.

Conceptual Model of an Architecture Description

ISO/IEC/IEEE 42010


System-of-Interest
exhibits
Architecture
1
1
1
identifies
1
has interests in
1..*
1
1
Stakeholder
identifies
1
Architecture Description
1..*
1
1..*
identifies
has
1..*
Concern
1..*
1..*
1..*
frames
addresses
1..*
1..*
1..*
1..*
Architecture Viewpoint
governs
Architecture View
1
1

INTRODUCTION TO VIEWS

Dictionary Meaning

Manner of looking at something

Why (multiple) view ?

For better understanding and managing.

Multi dimensional view must be taken for any complex entity because of its complex nature ,

It can’t be described in 1 dimensional view.

INTRODUCTION TO VIEWS

For example, In civil what are the views of a building…

- *Room layout*
- *3D view of building / room*
- *Electrical diagram*
- *Plumbing diagram*
- *Security alarm diagram*
- *AC duct diagram etc…etc…*

- Which of the above view is Architecture?
- **In Software, What are views ? ……..**

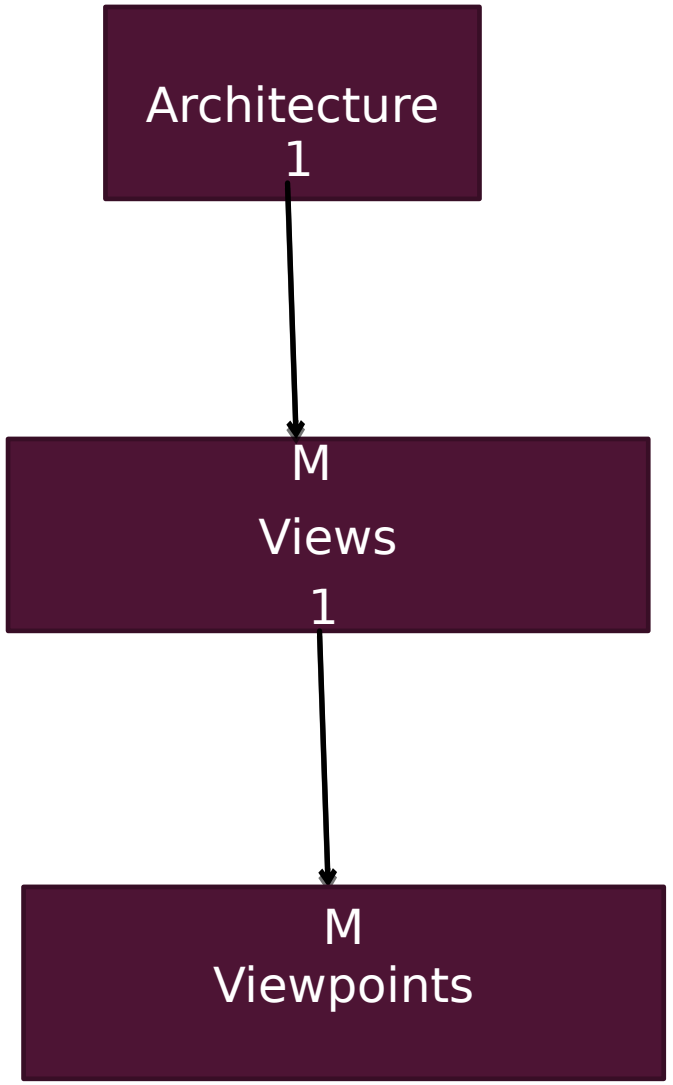
DEFINITION OF SW VIEW

As per IEEE definition,

- Software architecture descriptions are commonly organized into views,
- Each view addresses a set of system concerns, following the conventions of its *viewpoint*.

  - Viewpoint - A position or direction from which something is observed or considered;
  - View – Details or full specification considered from that viewpoint

- **So, a view of a system is a representation of the system from the perspective of a viewpoint.**


Architecture
1
M
Views
1
M
Viewpoints

VIEW MODEL

Software designers can organize the description of their architecture decisions in different views.

4+1 VIEW MODEL

The 4+1 view is an architecture verification technique for studying and documenting software architecture design.

THE 4 +1 VIEW MODEL

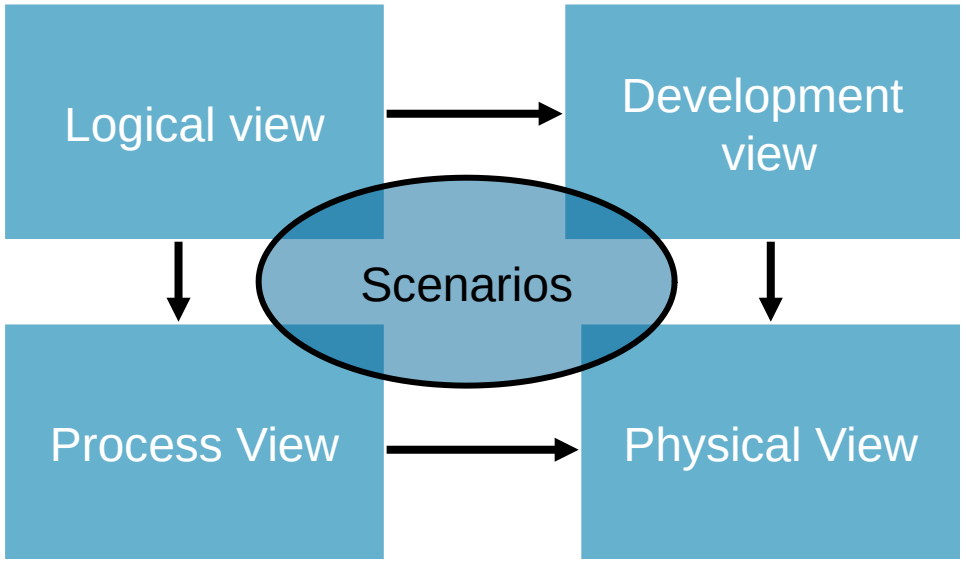
The 4+1 view model was originally introduced by Philippe Kruchten (Kruchten, 1995).

The model provides four essential views:

- the logical view,
- the process view,
- the physical view,
- the development view

- and fifth is the scenario view

4+1 VIEW MODEL OF ARCHITECTURE



THE 4+1 VIEW MODEL

- Multiple-view model that addresses different aspects and concerns of the system.
- Standardizes the software design documents and makes the design easy to understand by all stakeholders.

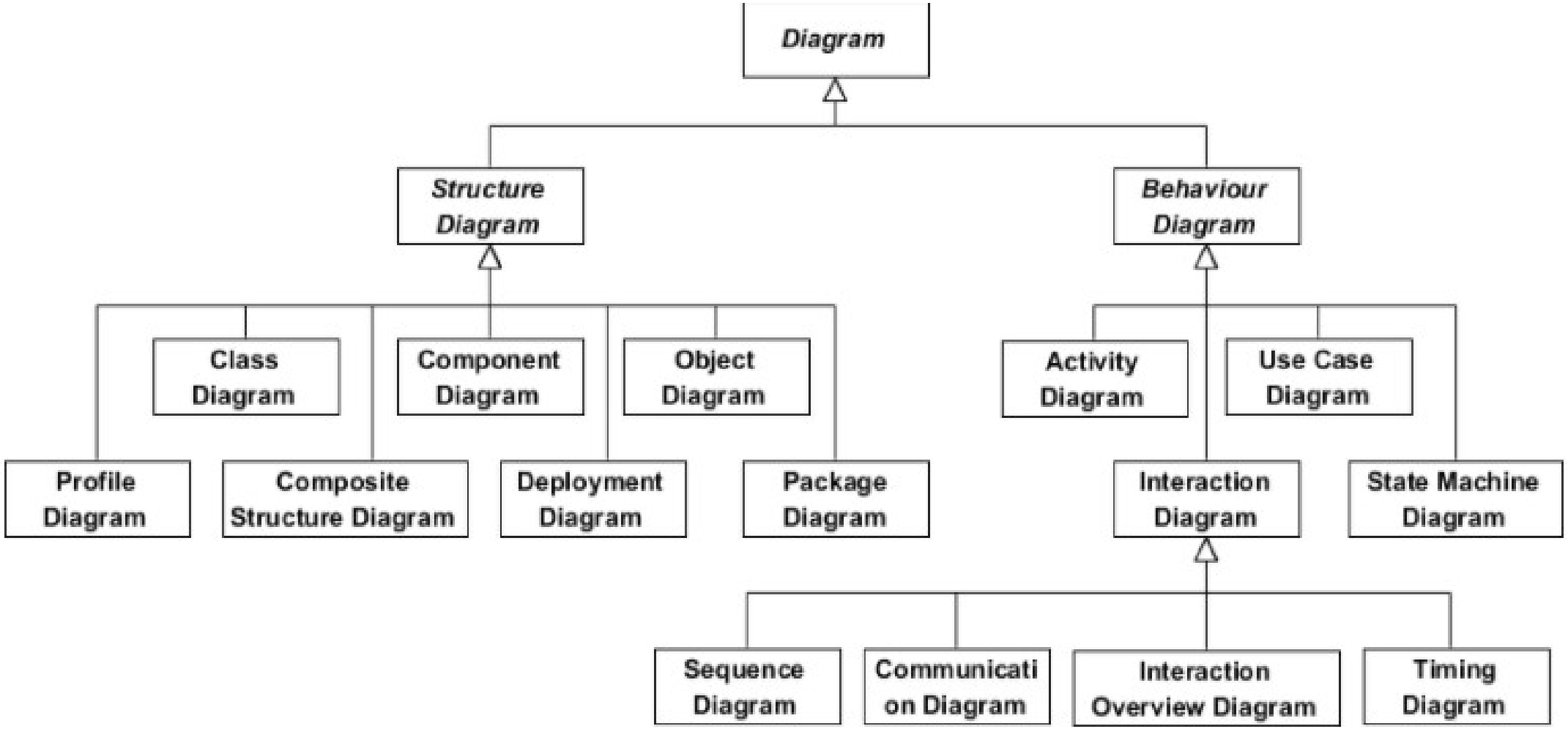


Diagram
Structure Diagram
Behaviour Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communicati on Diagram
Interaction Overview Diagram
Timing Diagram

THE SCENARIO VIEW- USE CASE VIEW

- The scenario view describes the functionality of the system, i.e., how the user employs the system and how the system provides services to the users.

- It helps designers to discover architecture elements during the design process and to validate the architecture design afterward.

USE CASE

- They use case view illustrates the functionality of the system.
- Using use case we can capture the goals of the user or what the user expects from the system.
- In UML, Use Cases can be created through use case diagrams or use case descriptions
- Use cases can be created by analysts' architects or even by the users.

THE LOGICAL OR CONCEPTUAL VIEW

- The logical view is based on application domain entities necessary to implement the functional requirements.

- The logical view specifies system decomposition into conceptual entities (such as objects) and connections between them (such as associations).

LOGICAL VIEW

- The logical view shows the parts that make up the system and how they interact with each other.
- It represents the abstractions that are used in the problem domain
  - These abstractions are classes and objects
- Different UML diagrams show the logical way such as class diagram state diagram sequence, diagram communication diagram and object diagram.

Logical view

- class diagram
- state diagram
- Sequence diagram
- communication diagram
- object diagram

THE DEVELOPMENT OR MODULE VIEW

- The development view derives from the logical view and describes the static organization of the system modules.

- UML diagrams such as package diagrams and component diagrams are often used to support this view.

DEVELOPMENT VIEW

- The development view describes the modules are the components of the system.
- This might include packages or libraries.
- It gives a high-level view of the architecture of the system and helps in managing the layers of the system.
- UML provides two diagrams for development view.
  - component Diagram
  - package Diagrams

Development View

- Component Diagram
- Packages Diagram

THE PROCESS VIEW

- The process view focuses on the dynamic aspects of the system, i.e., its execution time behavior.
- This view maps functions, activities, and interactions onto runtime implementation.

PROCESS VIEW

- Then we have the process view
- Through this view, we can describe the processes of the system and how they communicate with each other using process
- Using process view, we can find out what needs to happen to the system
- So using process view we can understand the overall functioning of the system
- Activity diagram in UML represents the process view

Process view

- Activity Diagram

THE PHYSICAL VIEW

- The physical view describes installation, configuration, and deployment of the software application.
- It concerns itself with how to deliver the deploy-able system.
- The physical view shows the mapping of software onto hardware.

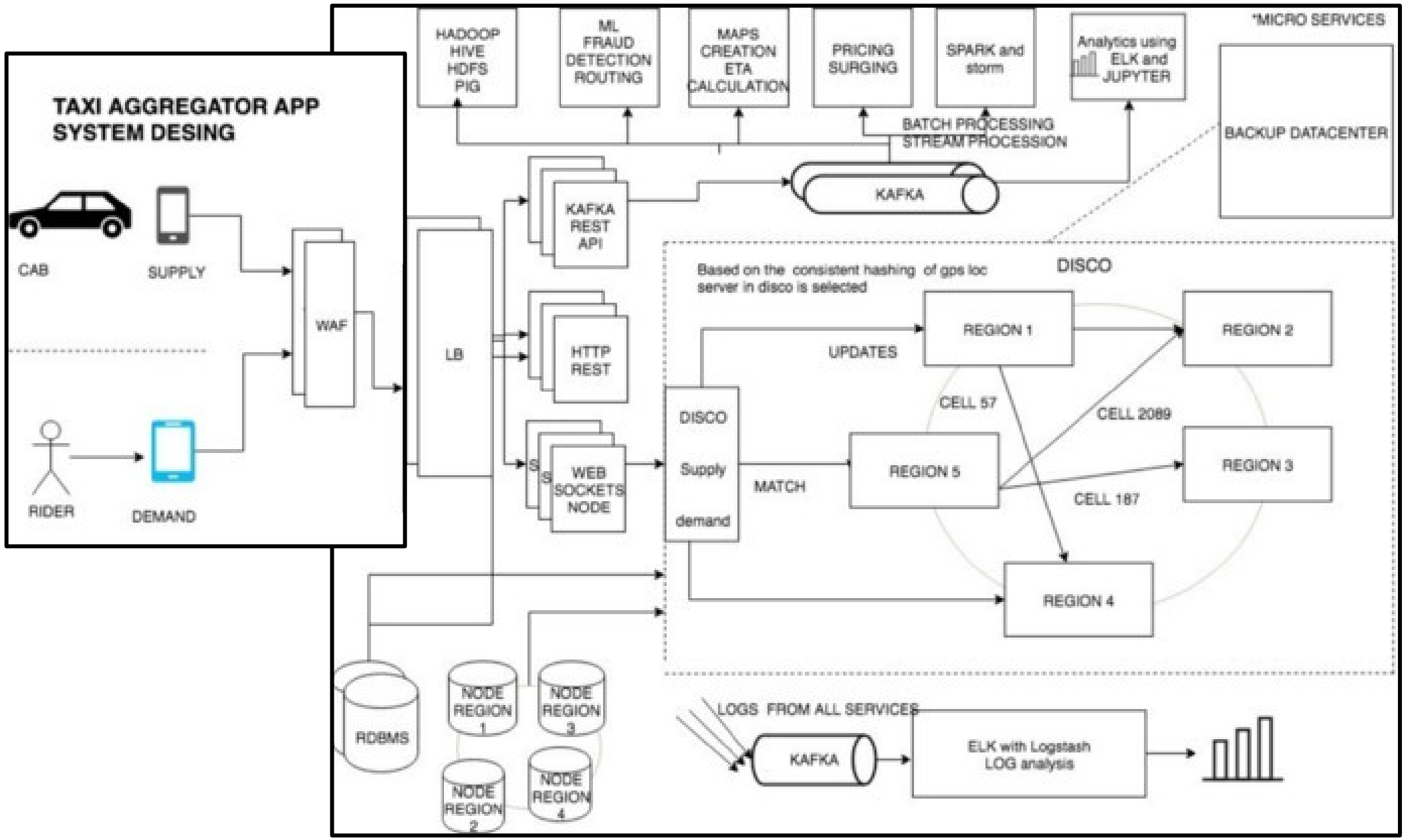
PHYSICAL VIEW

- The physical view is the view that models the execution environment of the system
- Using this view, we can model the software entities onto the hardware that will host and run the entities
- The physical view in UML is represented through deployment diagrams

Physical view

- Deployment Diagrams

UBER CASE STUDY


TAXI AGGREGATOR APP
SYSTEM DESING
CAB
SUPPLY
RIDER
DEMAND
WAF
LB
KAFKA REST API
HTTP REST
WEB SOCKETS NODE
HADOOP HIVE HDFS PIG
ML FRAUD DETECTION ROUTING
MAPS CREATION ETA CALCULATION
PRICING SURGING
SPARK and storm
Analytics using ELK and JUPYTER
*MICRO SERVICES
BACKUP DATACENTER
BATCH PROCESSING
STREAM PROCESSION
KAFKA
DISCO
Based on the consistent hashing of gps loc server in disco is selected
DISCO
Supply
demand
MATCH
UPDATES
REGION 1
REGION 2
REGION 3
REGION 4
REGION 5
CELL 57
CELL 2089
CELL 187
RDBMS
NODE REGION 1
NODE REGION 2
NODE REGION 3
NODE REGION 4
LOGS FROM ALL SERVICES
KAFKA
ELK with Logstash
LOG analysis

UBERS CASE STUDY

- Uber’s technology may look simple but when A user requests a ride from the app, and a driver arrives to take them to their destination.

- But Behind the scenes, however, a giant infrastructure consisting of thousands of services and terabytes of data supports each and every trip on the platform.

- Like most web-based services, the Uber backend system started out as a “monolithic” software architecture with a bunch of app servers and a single database.

UBERS SERVICES

The challenging thing is to supply demand.

So we need two services

- Supply service
- Demand service

UBERS SERVICES

Supply service

The Supply Service tracks cars using geolocation (lat and lang).

Every cab which is active keep on sending lat-long to the server every 5 sec once.

Demand service

The Demand Service tracks the GPS location of the user when requested.

Now we have supply and demand. all we need a service which matches they demand to a supply and that service in UBER is called as *Dispatch Optimization.*

HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

GPS/ location data is what drive dispatch system, that means we have to model our maps and location data.

The earth is a sphere. It’s hard to do summarization and approximation based purely on longitude and latitude.

HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

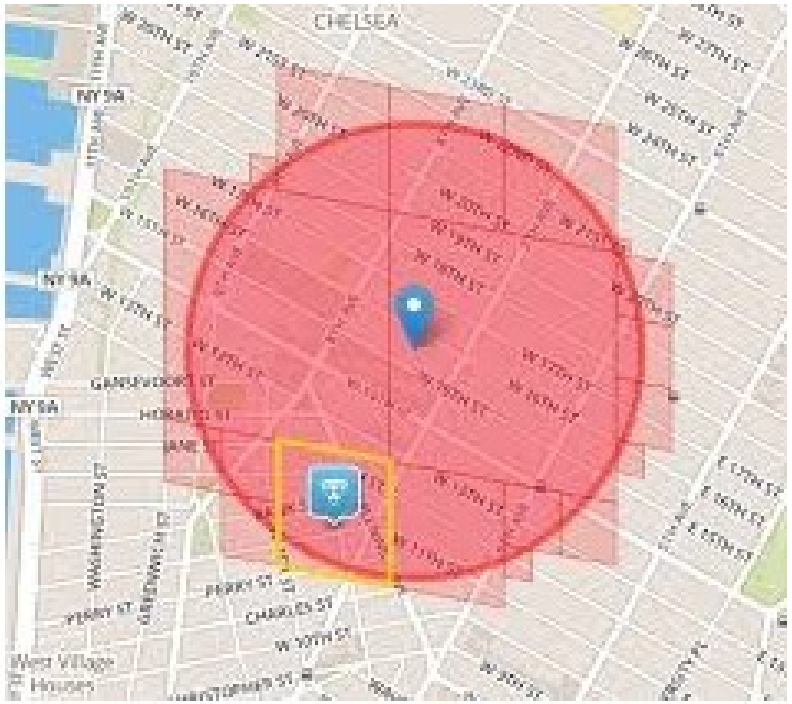
So Uber divides the earth into tiny cells using the Google S2 library. Each cell has a unique cell ID.

S2 can give the coverage for a shape. If you want to draw a circle with a 1km radius centered on London, S2 can tell what cells are needed to completely cover the shape.

HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

ETA (Estimated Time to Arrival)

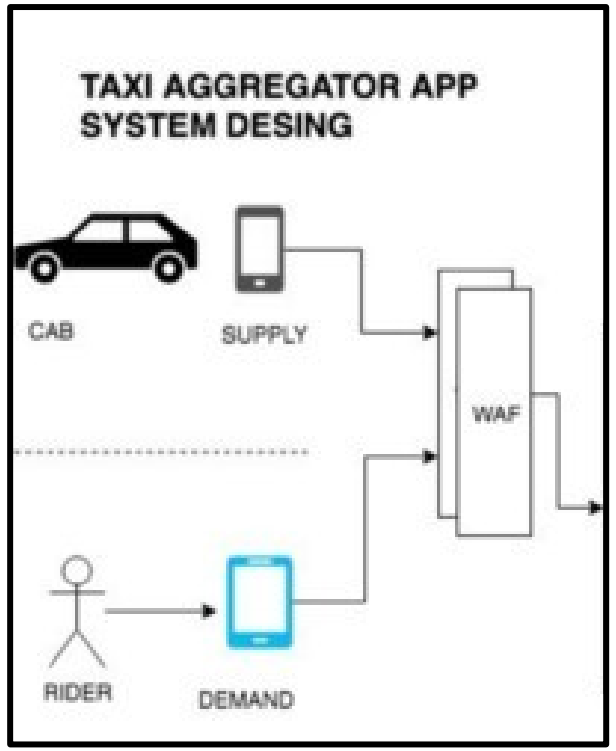
To compute the ETA of how nearby they are not geographically, but by the road system.



WEB APPLICATION FIREWALL

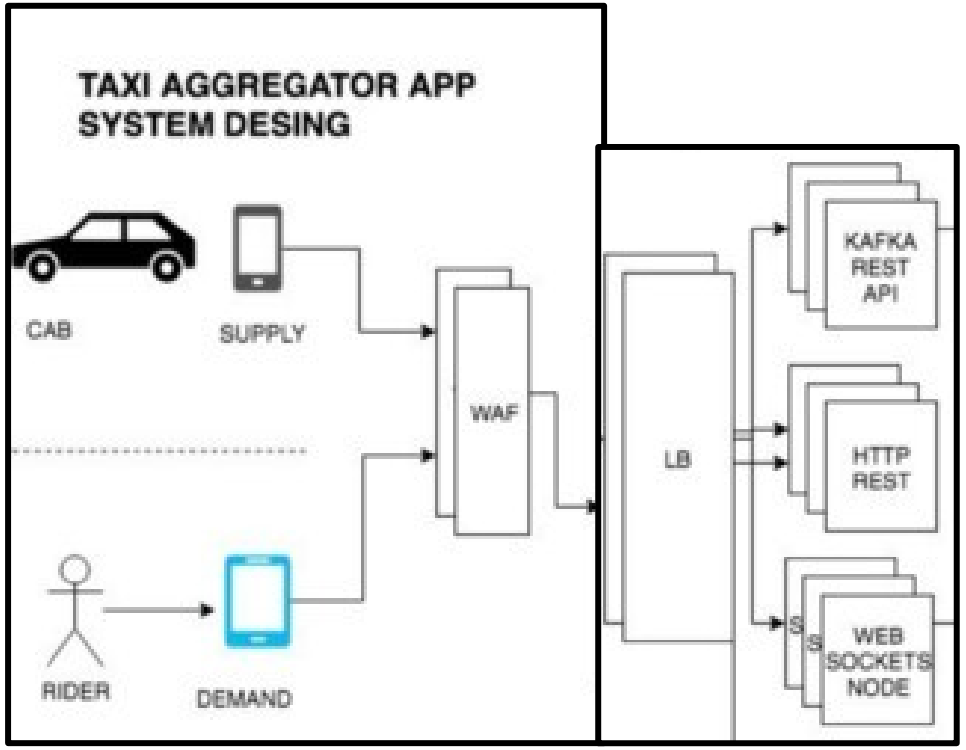
A web application firewall (WAF) is a firewall that monitors, filters and blocks requests from

- block IPs
- Bots
- Or where UBER service is not launched yet

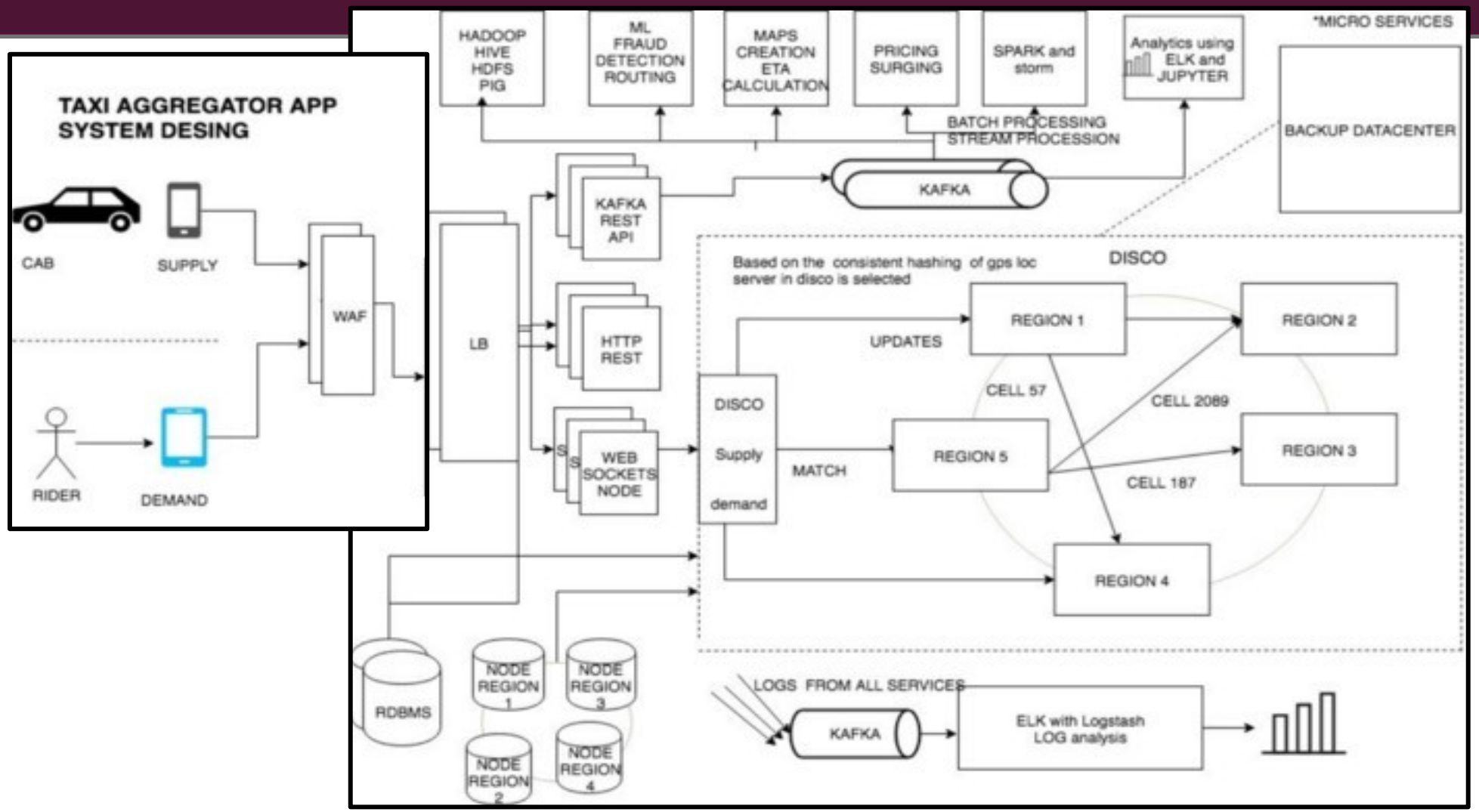


LOAD BALANCING

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool.*



SUPPLY/ DEMAND



ARCHITECTURAL STYLES

“A set of design rules that identify the kinds of components and connectors that may be used to compose a system.

The architectural style is a very specific solution to a particular software system, which typically focuses on how to organize the components created for the software.

COMPONENTS OF A STYLE

The key components of an architecture style are:

- **Elements/components**

  that perform functions required by a system

- **connectors**

  that enable communication, coordination, and cooperation among elements

- **constraints**

  that define how elements can be integrated to form the system

- **attributes**

  that describe the advantages and disadvantages of the chosen structure

CATEGORIES OF ARCHITECTURAL STYLES

- **Hierarchical Software Architecture**
  - Layered
- **Data Flow Software Architecture**
  - Pipe and Filter
  - Batch Sequential
- **Data Centered Software Architecture**
  - Black board
  - Shared Repository
- **Component-Based Software Architecture**
- **Distributed Software Architecture**
  - Client Server
  - Peer to Peer
  - REST
  - SOA
  - Microservices
  - Cloud Architecture
- **Event Based Software Architecture**

HIERARCHICAL SOFTWARE ARCHITECTURE

HIERARCHICAL STYLE

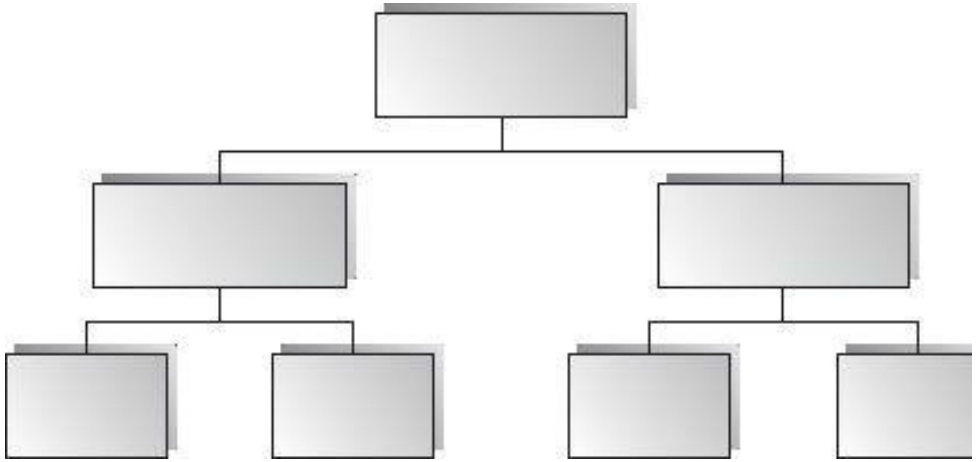
- The hierarchical software architecture is characterized by viewing the entire system as a hierarchy structure.

- The software system is decomposed into logical modules (subsystems) at different levels in the hierarchy.

HIERARCHICAL STYLE

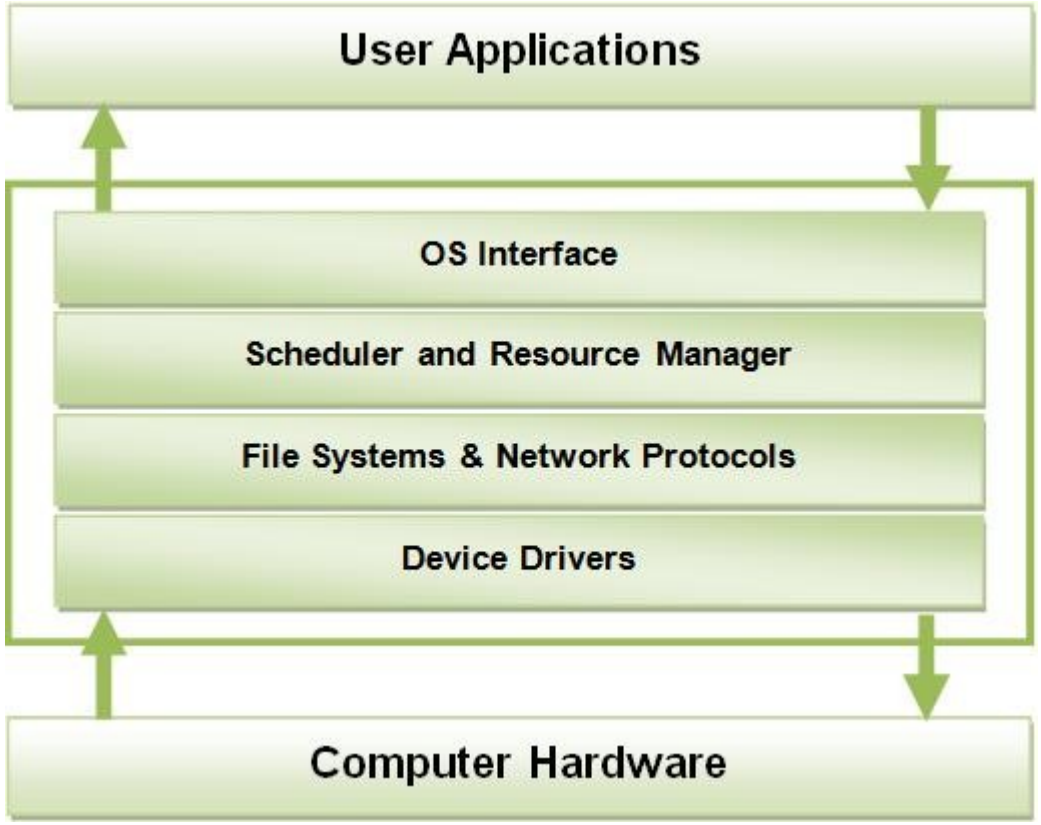
- Modules at different levels are connected by method invocations.

  a lower-level module provides services to its adjacent upper-level modules, which invokes the methods or procedures in the lower level.



HIERARCHICAL STYLE

System software is typically designed using the hierarchical architecture style.

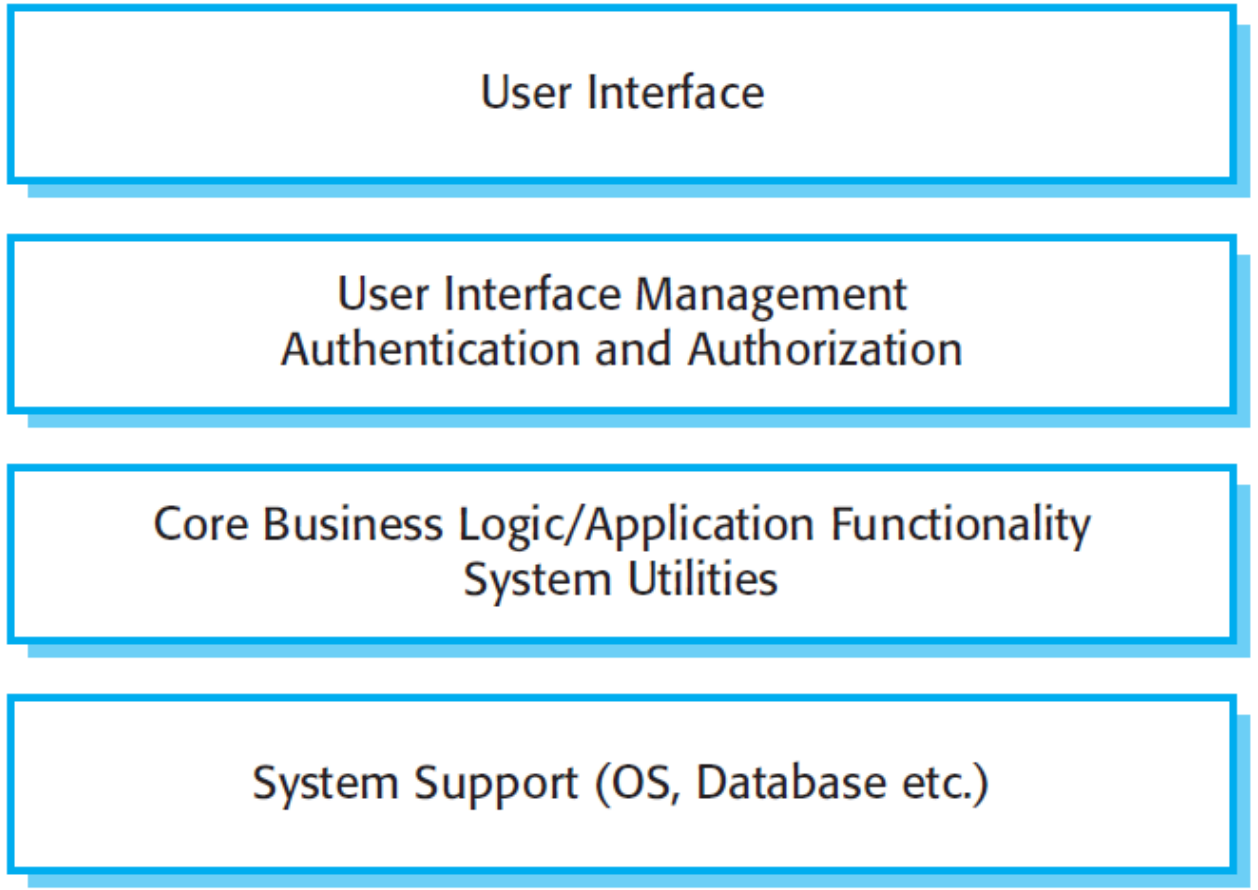


LAYERED ARCHITECTURE

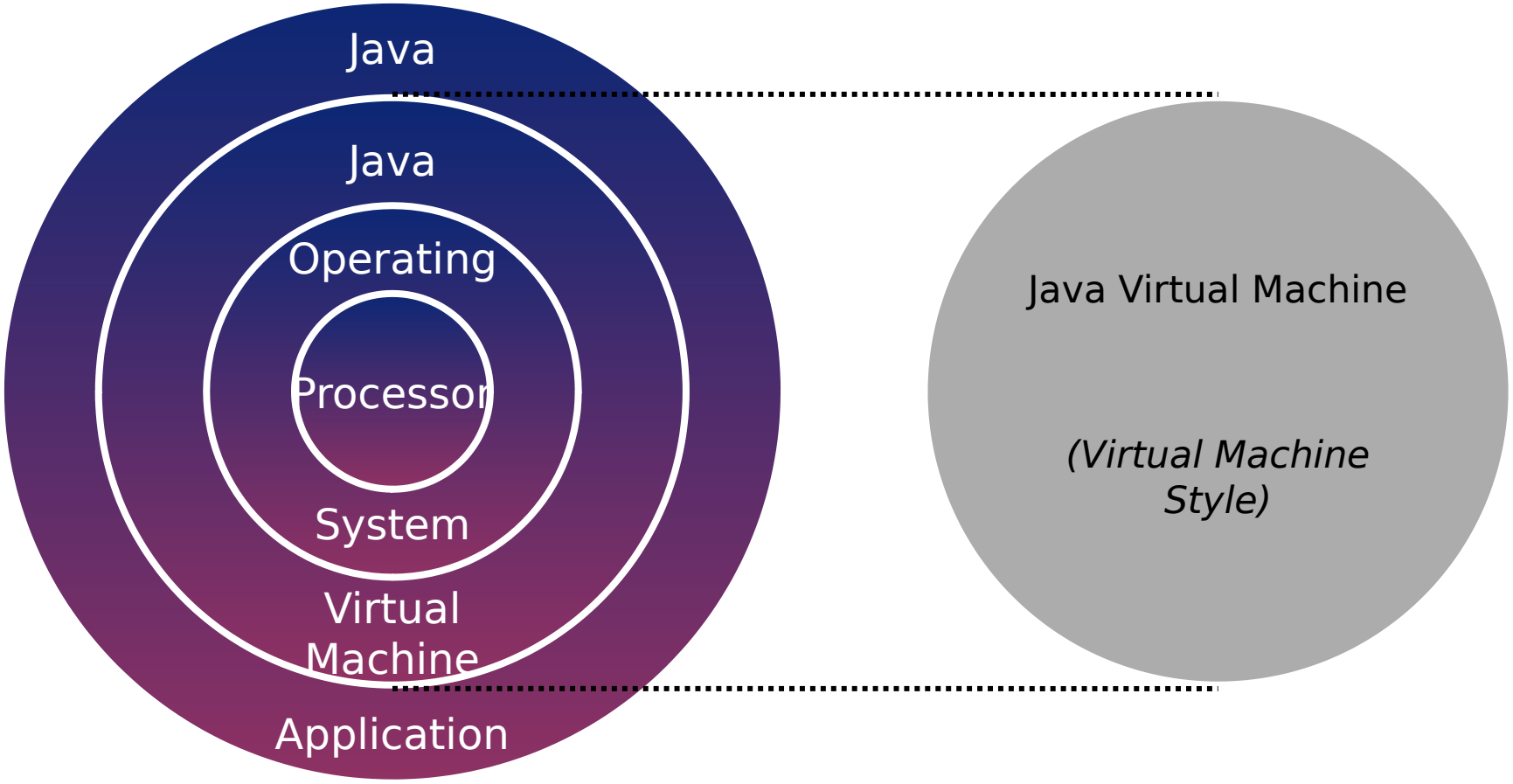
LAYERED STYLE

- Organized hierarchically into layers.
- Each layer provides service to the layer above it and serves as a client to the layer below.
- The connectors are defined by the protocols that determine how the layers will interact.

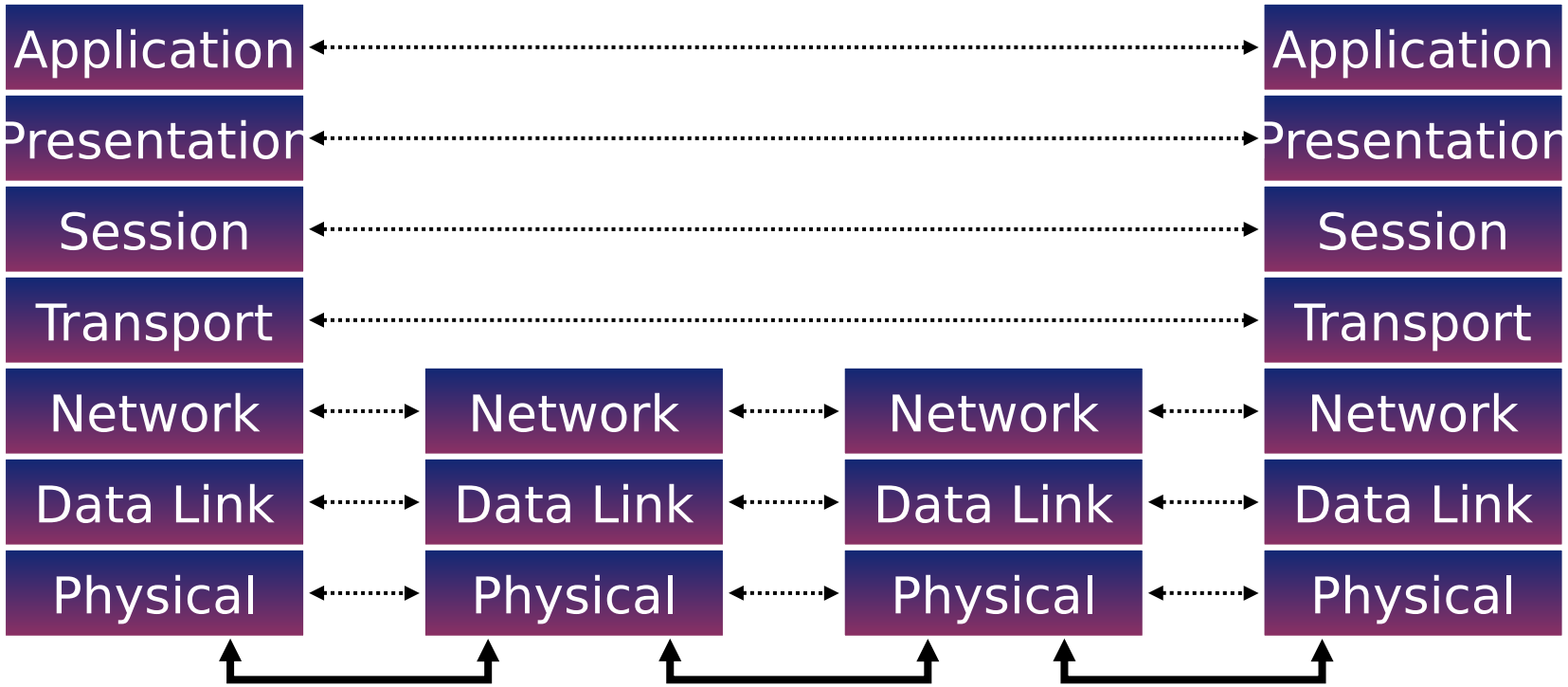
A GENERIC LAYERED ARCHITECTURE



LAYERED VIRTUAL MACHINE EXAMPLE: JAVA



LAYERED SYSTEM EXAMPLE: OSI PROTOCOL STACK



APPLICABLE DOMAINS OF LAYERED ARCHITECTURE:

- Any system that can be divided between the application-specific portions and platform-specific portions which provide generic services to the application of the system.
- Applications that have clean divisions between core services, critical services, user interface services, etc.
- Applications that have a number of classes that are closely related to each other so that they can be grouped together into a package to provide the services to others.

BENEFITS:

- Incremental software development based on increasing levels of abstraction.
- Enhanced independence of upper layer to lower layer since there is no impact from the changes of lower layer services as long as their interfaces remain unchanged.

BENEFITS (CONT..)

- Enhanced flexibility: interchangeability and reusability are enhanced due to the separation of the standard interface and its implementation.
- Promotion of portability: each layer can be an abstract machine deployed independently.

LIMITATIONS:

- Lower runtime performance since a client's request or a response to a client must go through potentially several layers.
- There are also performance concerns of overhead on the data processing and buffering by each layer.

LIMITATIONS (CONT..)

- Breach of interlayer communication may cause deadlocks, and “bridging” may cause tight coupling.
- Exceptions and error handling are issues in the layered architecture, since faults in one layer must propagate upward to all calling layers.

DATA FLOW
SOFTWARE ARCHITECTURE

DATA FLOW ARCHITECTURES

- The data flow software architecture style views the entire software system as a series of transformations on successive sets of data.

- The software system is decomposed into data processing elements where data directs and controls the order of data computation processing.

DATA FLOW ARCHITECTURES

- Each component in this architecture transforms its input data into corresponding output data.

- In general, there is no interaction between the modules except for the output and the input data connections between subsystems.

DATA FLOW ARCHITECTURES

- One subsystem can be substituted by another without affecting the rest of the system
- Since each subsystem does not need to know the identity of any other subsystem, modifiability and reusability are important property attributes of the data flow architecture.
- Example: Image Processing

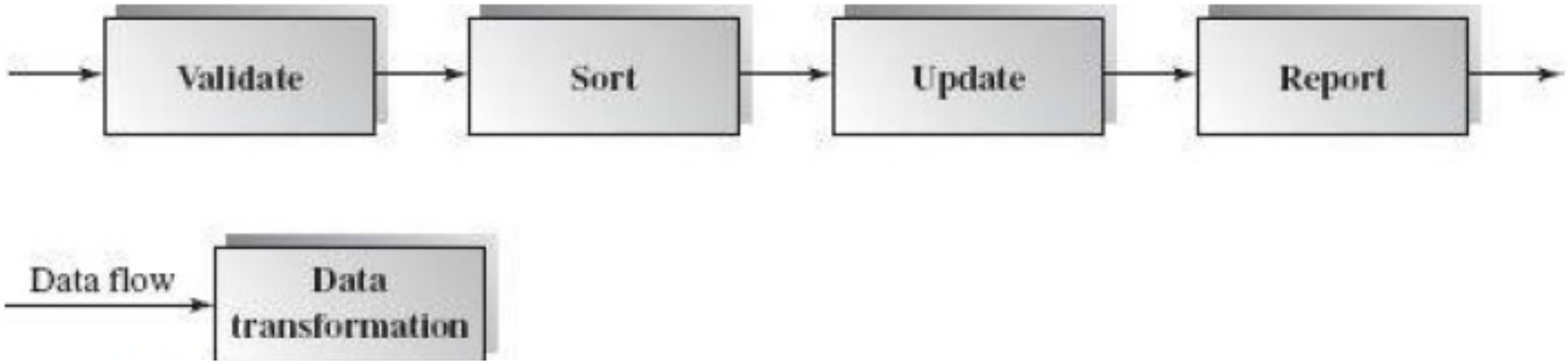
BATCH SEQUENTIAL

BATCH – SEQUENTIAL

- In batch sequential architecture, each data transformation subsystem or module cannot start its process until its previous subsystem completes its computation.

- Data flow carries a batch of data as a whole from one subsystem to another.

- First subsystem validates the transaction requests (insert, delete, and update) in their totality.
- Next, the second subsystem sorts all transaction records in an ascending order on the primary key of data records
- The transaction update module updates the master file with the sorted transaction requests, and then
- the report module generates a new list.



- The architecture is in a linear data flow order.

BENEFITS:

- Simple divisions on subsystems.
- Each subsystem can be a stand-alone program working on input data and producing output data.

LIMITATIONS:

- It does not provide interactive interface.
- Concurrency is not supported and hence throughput remains low.
- High latency.

PIPES-AND-FILTERS STYLE

PIPE AND FILTER

- This architecture decomposes the whole system into components of data source, filters, pipes, and data sinks.
- The connections between components are data streams.
- The particular property attribute of the pipe and filter architecture is its concurrent and incremented execution.

FILTER

- Each filter is an independent data stream transformer;
  - it reads data from its input data stream, transforms and processes it, and then
  - writes the transformed data stream over a pipe for the next filter to process.

FILTER

A filter does not need to wait for batched data as a whole.

- As soon as the data arrives through the connected pipe, the filter can start working right away.

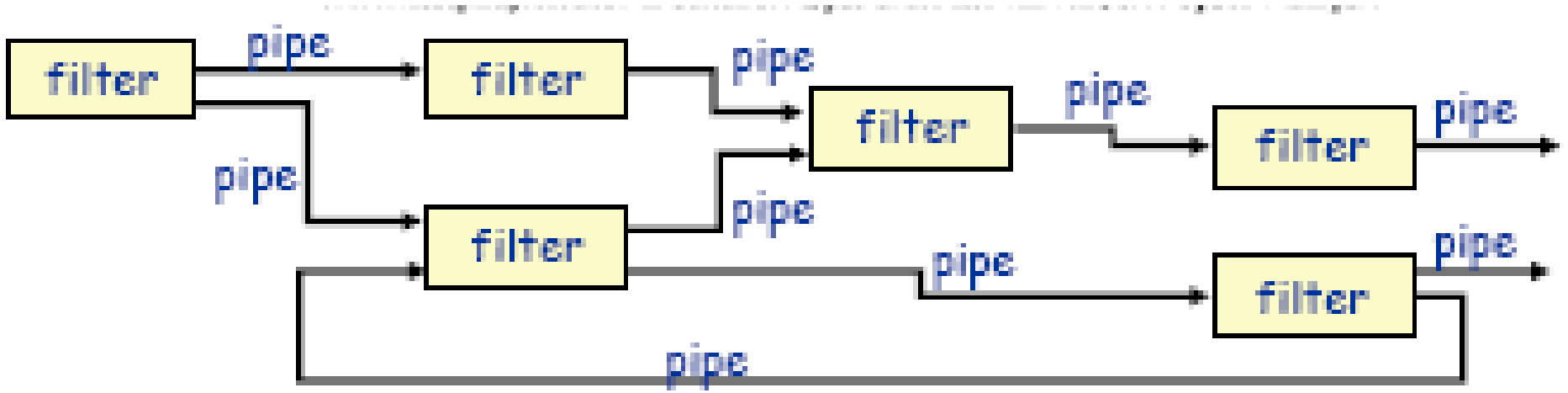
PIPES

- The connectors serve as channels for the streams, transmitting outputs of one filter to inputs of the other.
  - This makes connectors act as **Pipes**.

PIPES

- A pipe moves a data stream from one filter to another.
- A pipe is placed between two filters; these filters can run in separate threads of the same process.

STRUCTURE


filter
pipe
filter
pipe
filter
pipe
filter
pipe
filter
pipe
filter
pipe
pipe
pipe
pipe

EXAMPLES

Traditional Compilers:

- Compilation phases are pipelined, though the phases are not always incremental. The phases in the pipeline include:
  - *lexical analysis + syntax analysis (parsing) + semantic analysis + code optimization + code generation*

EXAMPLE: ARCHITECTURE OF A COMPILER

- Compilation is regarded as a sequential (pipeline) process.
- Every phase is dependent on some data on the preceding phase.

BENEFITS:

- Concurrency: It provides high overall throughput for excessive data processing.
- Reusability: Encapsulation of filters makes it easy to plug and play, and to substitute.
- Flexibility: It supports both sequential and parallel execution.

BENEFITS:

- Modifiability: It features low coupling between filters, less impact from adding new filters, and modifying the implementation of any existing filters as long as the I/O interfaces are unchanged.

- Simplicity: It offers clear division between any two filters connected by a pipe.

DISADVANTAGES

- Not good choice for **interactive systems**, because of their transformational characteristic.

HAVE A GOOD DAY!