SOFTWARE ENGINEERING
(Week-8)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 8

- Architectural Styles
- Categories of Architectural Style
- Hierarchical Software Architecture
  - Layered
- Data Flow Software Architecture
  - Pipe n Filter
  - Batch Sequential

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 n Filter**
- **Batch Sequential**

Distributed Software Architecture

- Client Server
- Peer to Peer
- REST
- SOA
- Microservices

Event Based Software Architecture

Data Centered Software Architecture

- Black board
- Shared Repository

Component-Based Software Architecture

HIERARCHICAL SOFTWARE ARCHITECTURE

HIERARCHICAL STYLE

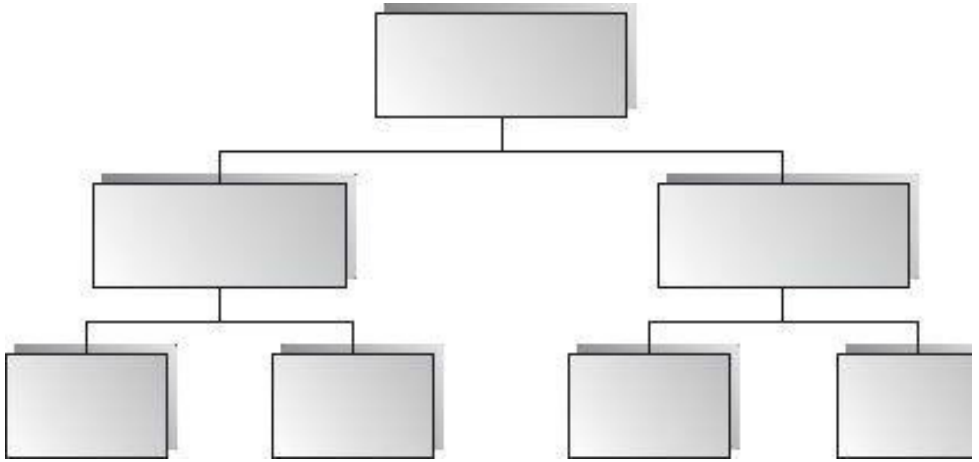
The hierarchical software architecture is characterized by viewing the entire system as a hierarchy structure.

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

HIERARCHICAL STYLE

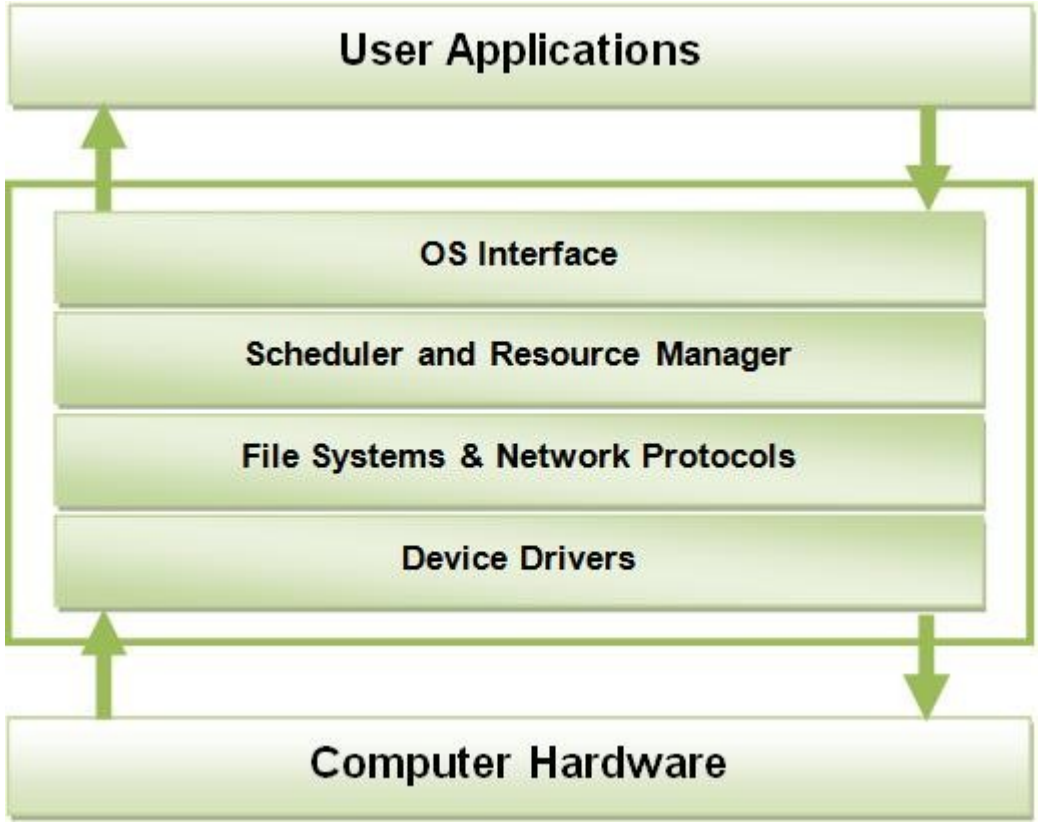
Modules at different levels are connected by method invocations.

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.



HIERARCHICAL STYLE

Lower levels provide more specific functionality down to fundamental utility services

such as I/O services, transaction, scheduling, and security services, etc.

Middle layers, in an application setting, provide more domain- dependent functions

such as business logic or core processing services.

Upper layers provide more abstract functionality in the form of user interfaces

such as command line interpreters, GUIs, etc.

LAYERED ARCHITECTURE

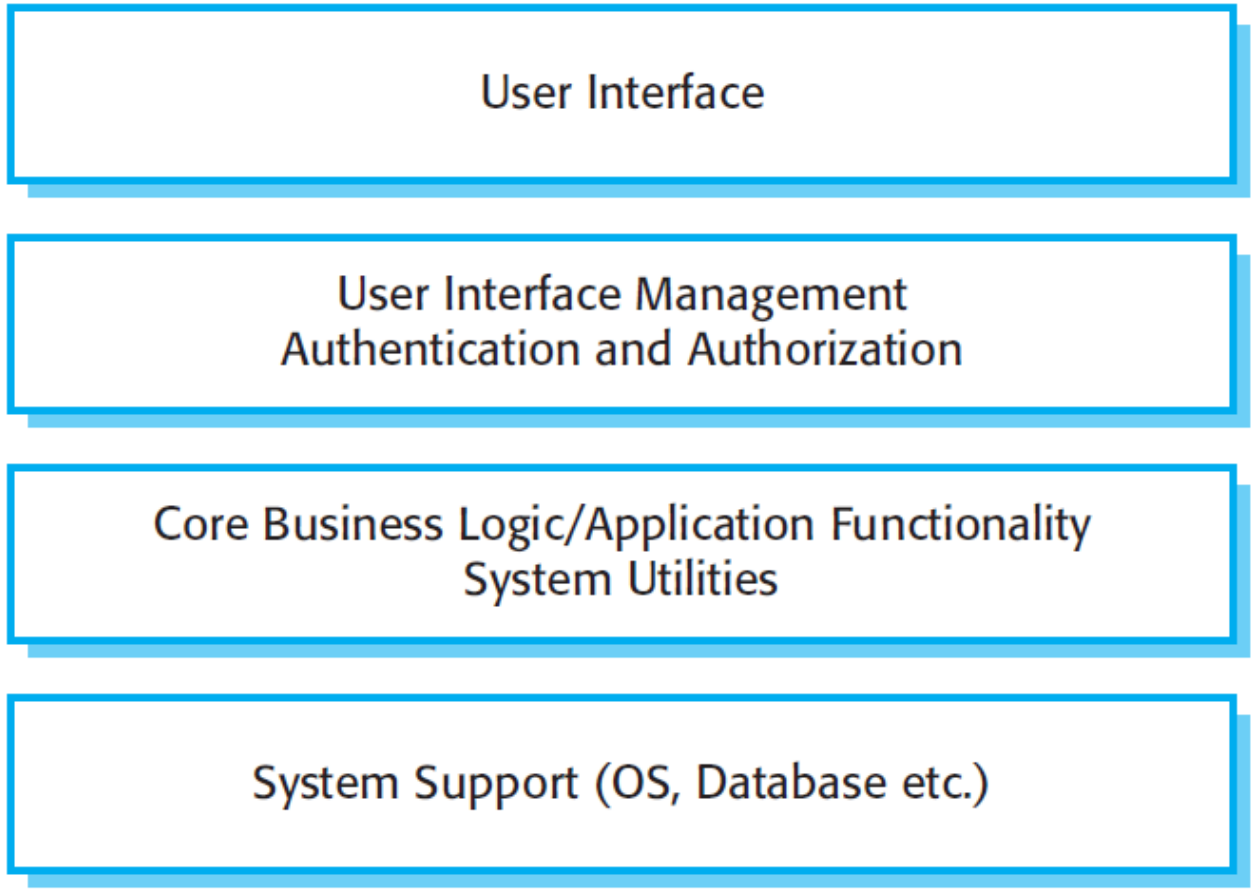
LAYERED STYLE

Organized hierarchically into layers.

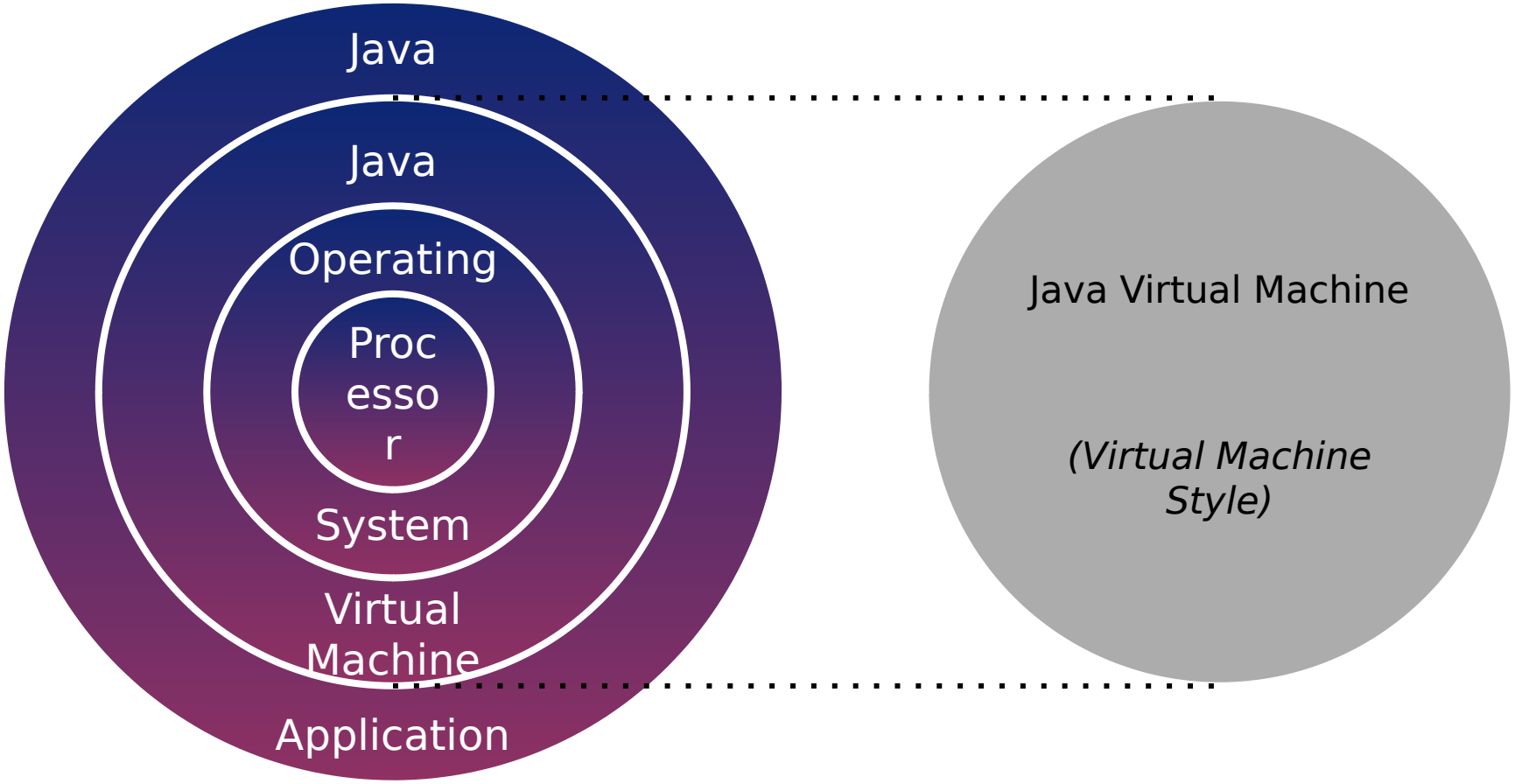
Each layer provides service to the layer above it and serves as a client to the layer below.

The connectors are defined by the protocols that determine how the layers will interact.

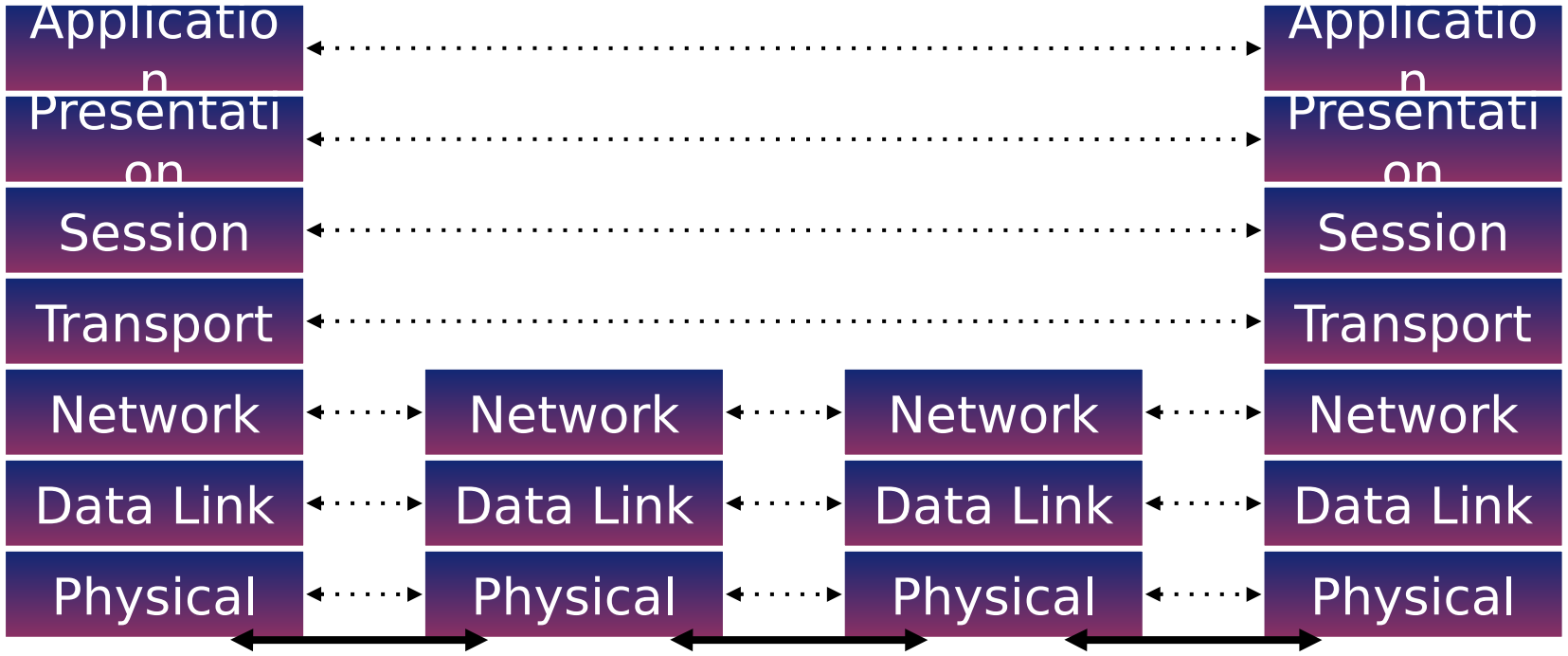
A GENERIC LAYERED ARCHITECTURE



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.

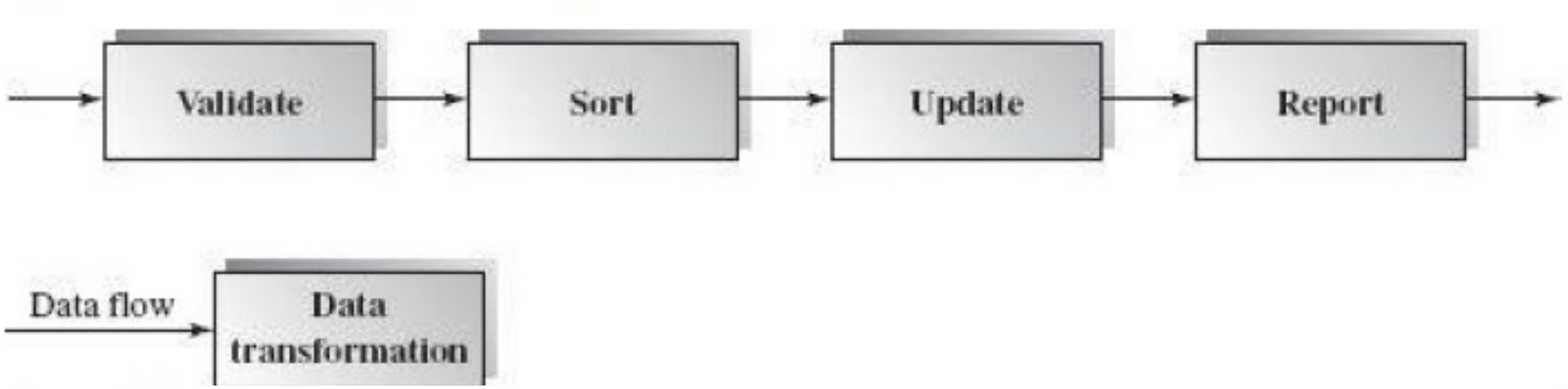
Data flow carries a batch of data as a whole from one subsystem to another.

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.

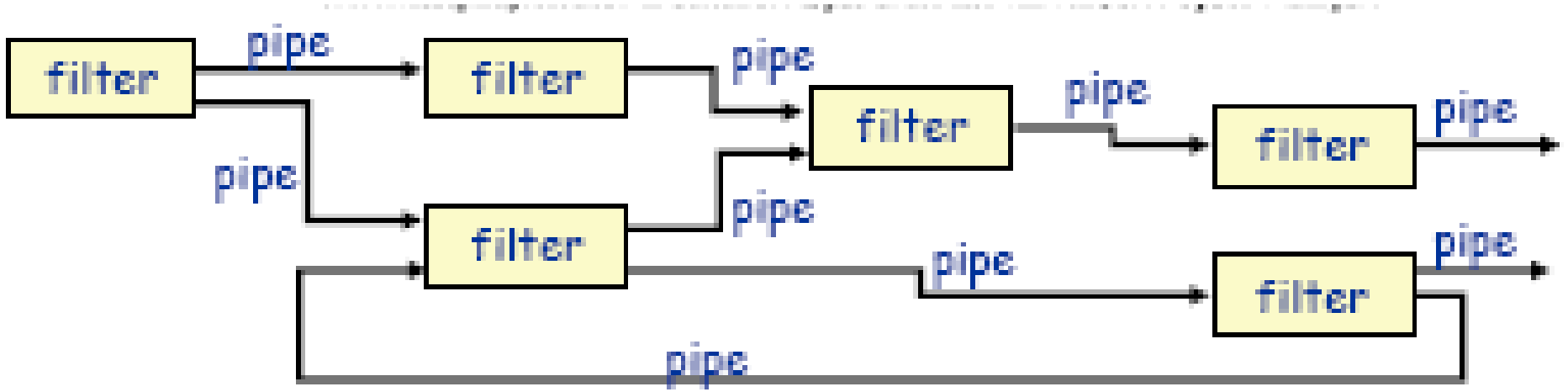
This makes connectors act as **Pipes**.

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 GOO DAY!