Software
Design & Architecture

Introduction to Architectural Styles

SESD-2222

Spring 2025

M Khizar Hayat

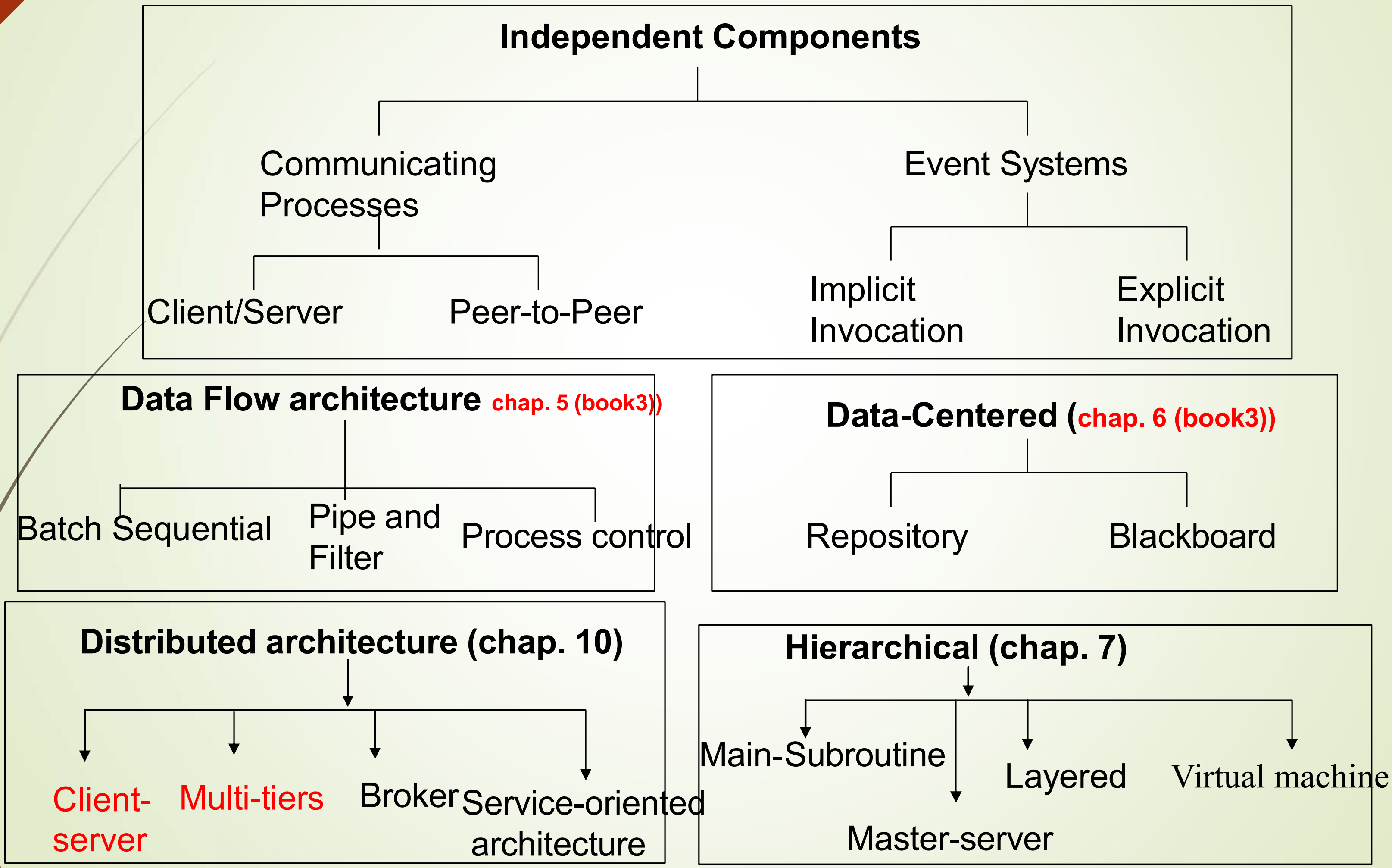
Khizer.hayat@ucp.edu.pk

Office: Building D, 2nd flor

OFFICE HOURS: Wednesday: 1:00 to 3:00

Thursday: 11:00 to 1:00

A Taxonomy of Architectural styles



Independent Components

- Communicating Processes
  - Client/Server
  - Peer-to-Peer
- Event Systems
  - Implicit Invocation
  - Explicit Invocation

Data Flow architecture chap. 5 (book3))

- Batch Sequential
- Pipe and Filter
- Process control

Data-Centered (chap. 6 (book3))

- Repository
- Blackboard

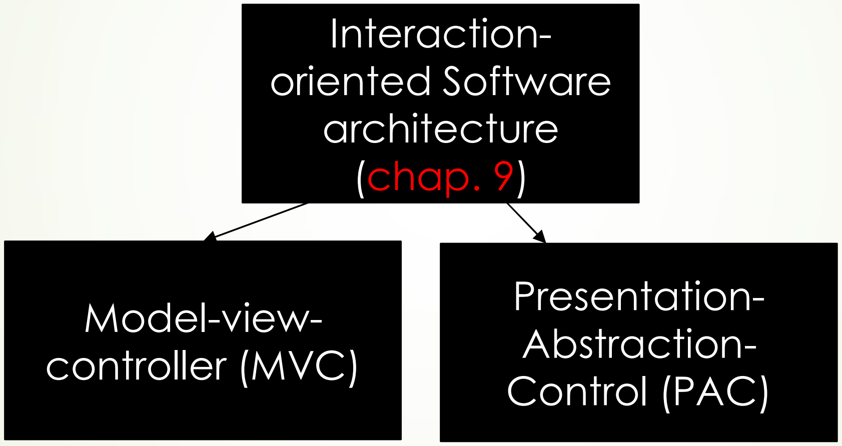
Distributed architecture (chap. 10)

- Client-server
- Multi-tiers
- Broker
- Service-oriented architecture

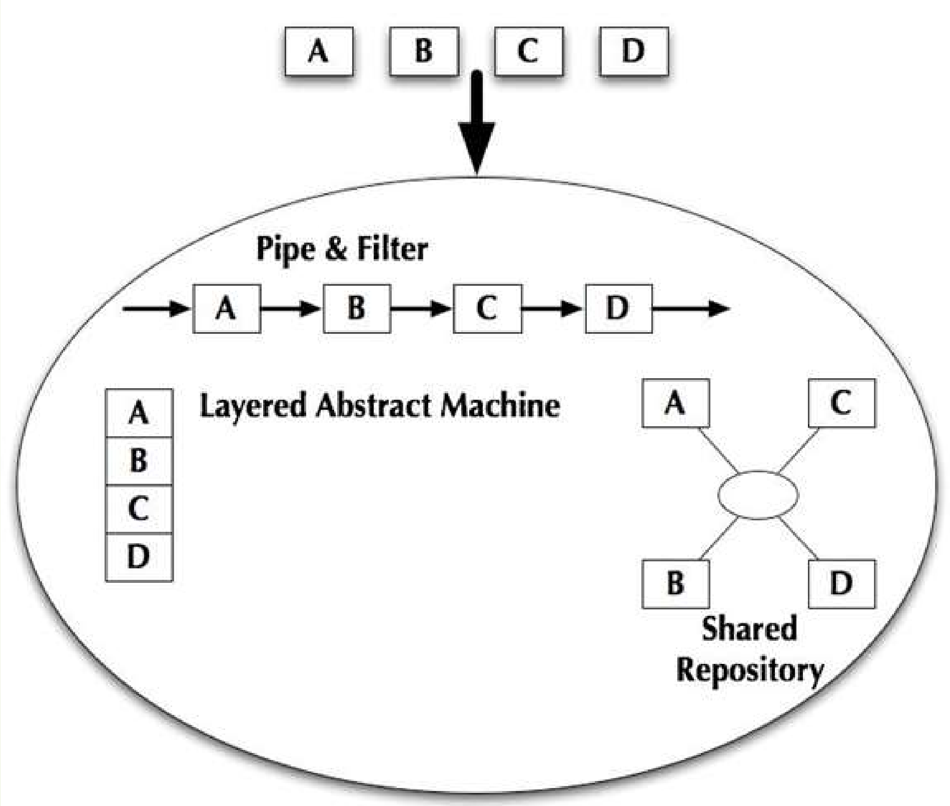
Hierarchical (chap. 7)

- Main-Subroutine
- Master-server
- Layered
- Virtual machine

A Taxonomy of Architectural styles


Interaction-oriented Software architecture (chap. 9)
Model-view-controller (MVC)
Presentation-Abstraction-Control (PAC)

Architectural variants



Software Architecture –Architecture Styles

- An architecture style (also known as an "architecture pattern") abstracts the common properties of a family of similar designs → contains a set of rules, constraints, and patterns of how to structure a system into a set of elements and connectors.

- Shows constituent element types and their runtime interaction of flow control and data transfer.

- The key components of an architecture style are:
  - Elements
  - Connectors
  - Constraints
  - Attributes

Architecture styles contd..

- Each style has a set of quality attributes that it promotes

- domain-independent. → It is also true that an architecture style maybe applied to many application domains

- The choice of a particular software architecture is made on the basis of an overall system organization

- There is no single-fit, perfect architecture. Over time, several different software architectural styles have been created - each having strengths and weaknesses.

Architectural styles

- For each style, we will identify:
  - Components
  - Connectors
  - Advantages
  - Dis-advantages

Data Flow Architectural Style

- Has the goal of modifiability
- Series of transformations apply on successive pieces of input data
- Data enters the system and then flows through the components one at a time until they are assigned to output or a data store
- Subsystems are independent → high modifiability and reusability
- Examples: banking, business batch processing etc.
- Direction of Data flow ?

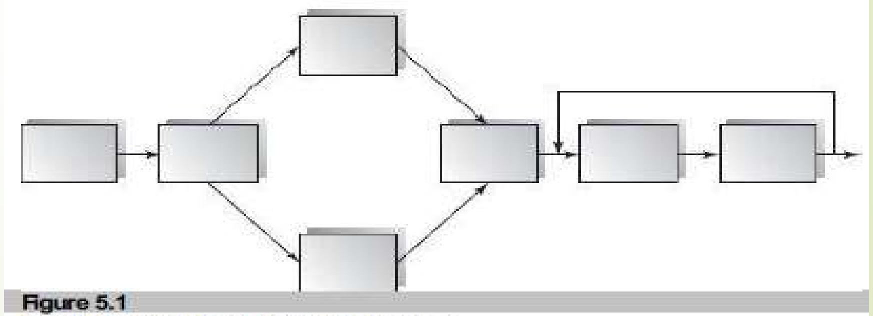


Figure 5.1
Block diagram of data flow architecture

Data Flow Architectural Style

- Three subcategories in the data flow architecture styles are:

i. Batch sequential

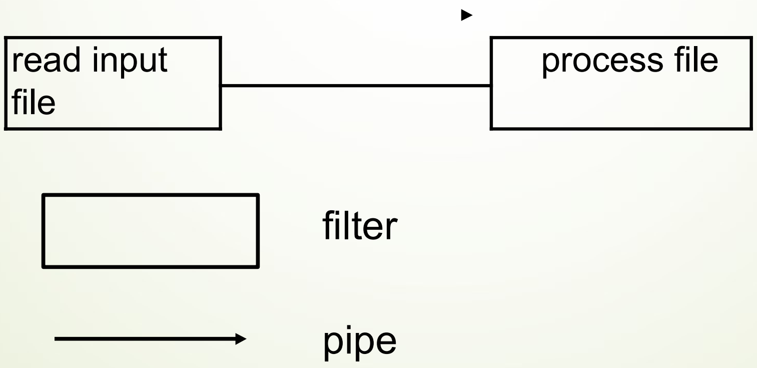
ii. Pipe and filter

iii. Process control

Pipe and Filter Architecture

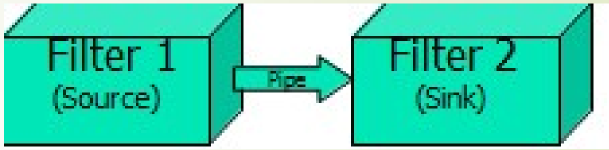
- Concurrent and incremented execution.
- Main components:
  - Filter: computing component that process the stream of input data to some output data
  - Pipe: communication channel that allows the flow of data

- Data stream is a first-in/first-out buffer



Pipes and filters

- Filter 1 may only send data to Filter 2
- Filter 2 may only receive data from Filter 1
- Data Source may not receive data
- Data Sink may not send data
- Pipe is the data transport mechanism



Example: pipe and filter architecture → concurrent execution.

- Convert document characters to uppercase and then count the occurrences of A, E character

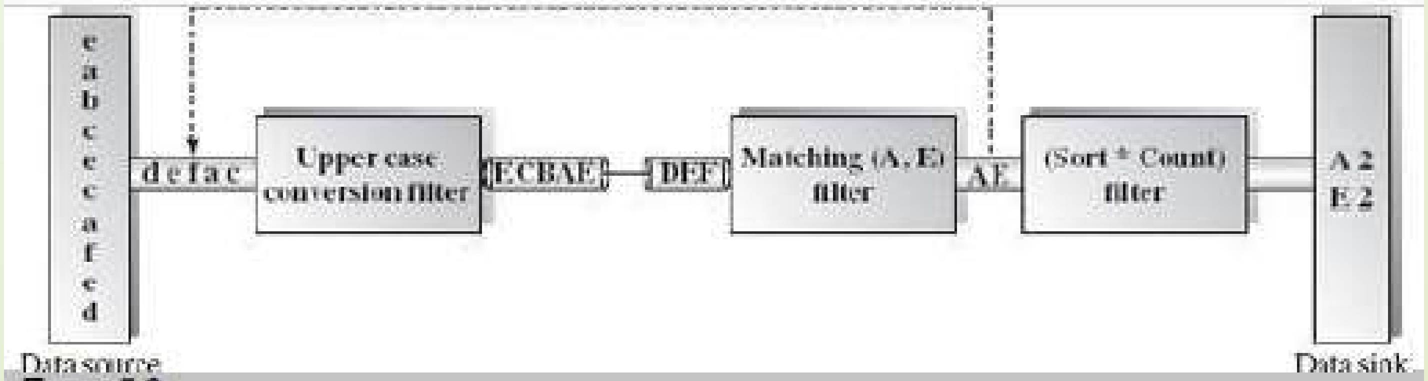


Figure 5.6
Pipelined pipe and filter

2. Pipe and Filter Architecture .. Data flow types

- A pipe is placed between two filters → filters can run in separate threads of the same process as Java I/O streams.

- three ways to make the data flow are:
  - **Push only (Write only)**
    - A data source may pushes data in a downstream
    - A filter may push data in a downstream
  - **Pull only (Read only)**
    - A data sink may pull data from an upstream
    - A filter may pull data from an upstream
  - **Pull/Push (Read/Write)**
    - A filter may pull data from an upstream and push transformed data in a downstream

Push-Only (Write-Only)

•**What It Means:**

•Data is actively sent downstream without the receiver requesting it.

•A data source or filter **pushes** data to the next stage.

•**Examples:**

1.**Sensor Systems**: A temperature sensor continuously pushes readings to a monitoring system.

Pull-Only (Read-Only)

•**What It Means:**

•Data is requested or "pulled" by a downstream component from an upstream source.

•A data sink or filter **pulls** data as needed.

•**Examples:**

1.**Web Browsing**: A browser pulls a webpage from a server only when requested by a user.

2.**File Processing**: A program reads chunks of a file only when processing is required.

Pull/Push (Read/Write)

- **What It Means:**
  - A filter reads (pulls) data from an upstream source, processes or transforms it, and then sends (pushes) the processed data downstream.
  - This mode combines both actions, making it versatile for transforming pipelines.
- **Examples:**
  - **Data Processing Pipeline**:
    - A program pulls raw data from a database, processes it (e.g., filtering, aggregating), and pushes the results to a visualization tool.
  - **ETL Pipelines (Extract, Transform, Load)**:
    - Data is pulled from a source (e.g., API or database), transformed (e.g., cleaned or enriched), and pushed to a data warehouse.

Pipe and Filter Architecture .. Active or passive filters

- An **active filter** pulls in data and push out the transformed data (pull/push) → works with a passive pipe
  - pipe & filter mechanism in Unix adopts this mode.
  - The PipedWriter and PipedReader classes in Java are also passive pipes

- A **passive filter** lets connected pipes to push data in and pull data out → works with active pipes
  - The filter must provide the read/write mechanisms in this case.
  - This is very similar to the data flow hardware architecture

Pipes and Filters: pro and cons

- **Advantages:**
  - Cohesive style
  - Low coupling
  - High reusability and modifiability
- **Disadvantages:**
  - The architecture is static (no dynamic reconfiguration)
  - Filter processes which send streams of data over pipes is a solution that fits well with heavy batch processing, but may not do well with any kind of user-interaction.
  - Anything that requires quick and short error processing is still restricted to sending data through the pipes, possibly making it difficult to interactively react to error-events.

Example: Text File Data Processing System

🎯 Scenario / Use Case:

- A company receives large text-based log files daily. These logs need to be processed to extract useful information such as error messages, filter out noise, convert formats, and finally store results in a report file. The system needs to process this data in a **step-by-step (pipelined)** manner.

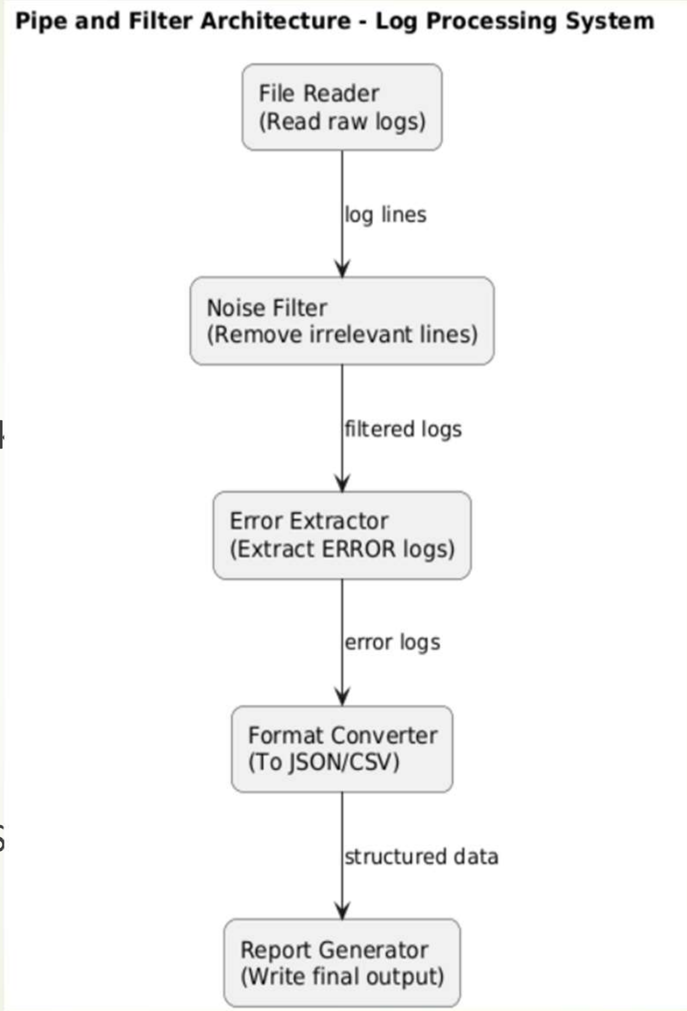
🏗️ Architecture Type: Pipe and Filter

◈ **Filters (Independent Processing Units):**
Each filter performs a single transformation on the data and passes it to the next.

◈ **Pipes (Connectors):**
Used to pass data from one filter to the next.

🛠 Filters in the System:

1. **File Reader Filter**
   - Reads raw log data from a file
   - Sends each line to the next stage
2. **Noise Filter**
   - Removes unnecessary lines (e.g., debug logs, routine chec
   - Passes only meaningful entries forward
3. **Error Extractor Filter**
   - Extracts only lines marked as "ERROR"
   - Filters out all other severities like INFO or WARNING
4. **Format Converter Filter**
   - Converts the filtered lines into a structured format (e.g., JS
5. **Report Generator Filter**
   - Writes the structured output to a final report file



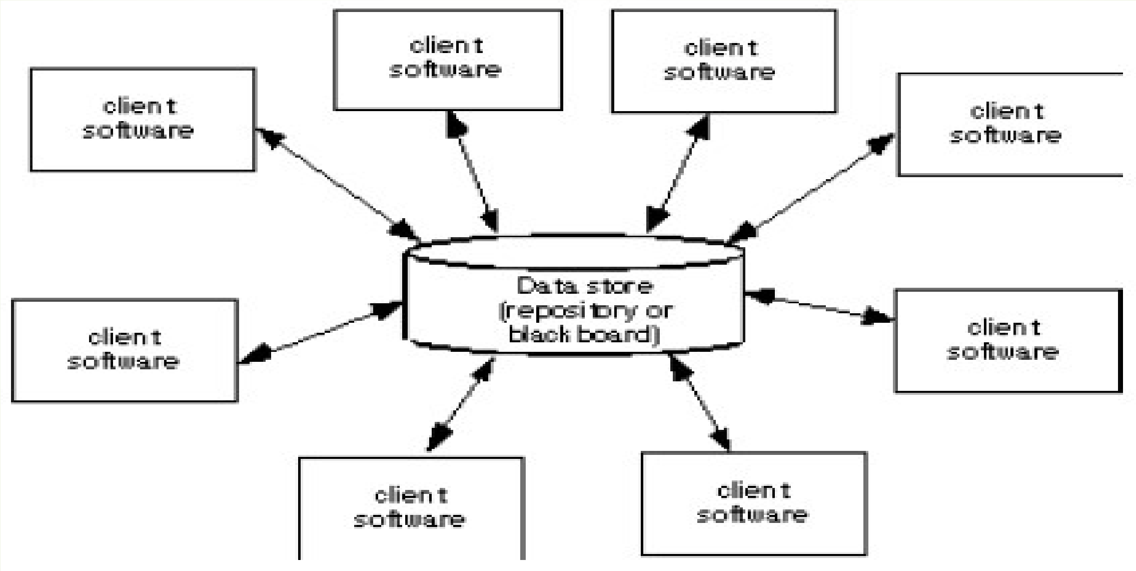
Data-Centered Software Architecture

Data-Centered Software Architecture

- Centralized data store that is shared by all surrounding software components
- Software system is decomposed into two major partitions:
  - Data store
  - Independent software components or agents.
- Connections between the data module and the software components are implemented either by explicit method invocation or by implicit method invocation.
- Pure data-centered software architecture: all communication is through repository
- Two categories of data-centered architecture:

i. Repository : passive data store

ii. Blackboard : active data store


client software
client software
client software
client software
Data store (repository or black board)
client software
client software
client software
client software

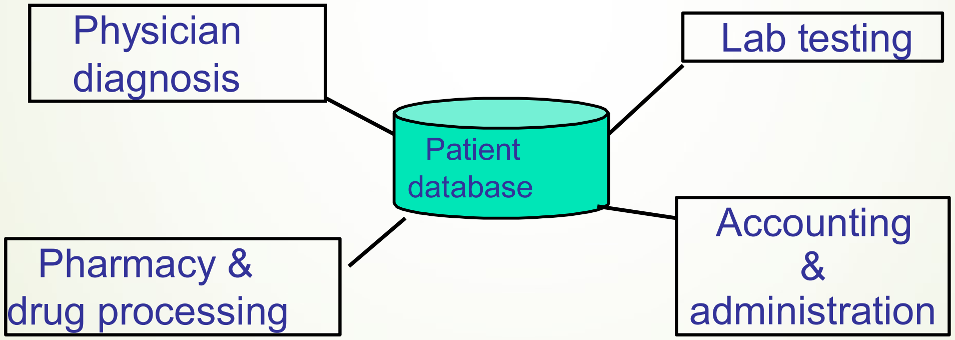
Repository Architecture Style

- A repository is a *shared data-store and* supports user interaction
- Clients can get data from the data store and put data in the data store → different access privileges
- Very common in information systems where data is shared among different functions
- All clients are not necessarily completely independent. There may still be some communication between individual agents e.g compiler

Repository Architecture Style

- Most commonly used when large amounts of data are to be shared

- Problems that fit this style have the following properties:
  1. All the functionalities work off a shared data-store
  2. Any change to the data-store may affect all or some of the functions
  3. All the functionalities need the information from the data-store



Repository Architecture Style

- **Passive repository** or repository based architecture: simply a central place to store application source code and information which is used ONLY to compile application programs.
  - Allow information to be reused when recompiling the application
  - has limited benefits to the application developer.

- **ACTIVE repository** or blackboard architecture : used in both the building and execution of the applications.
  - dynamic and it is used when the application executes.
  - changes made to the data store can be immediately reflected in the application without having to rebuild coded application programs.

- Example: "Employee Age" field for hiring process

Application domain of repository architecture

1. **Relational database management system**

- Database system with its data repository:

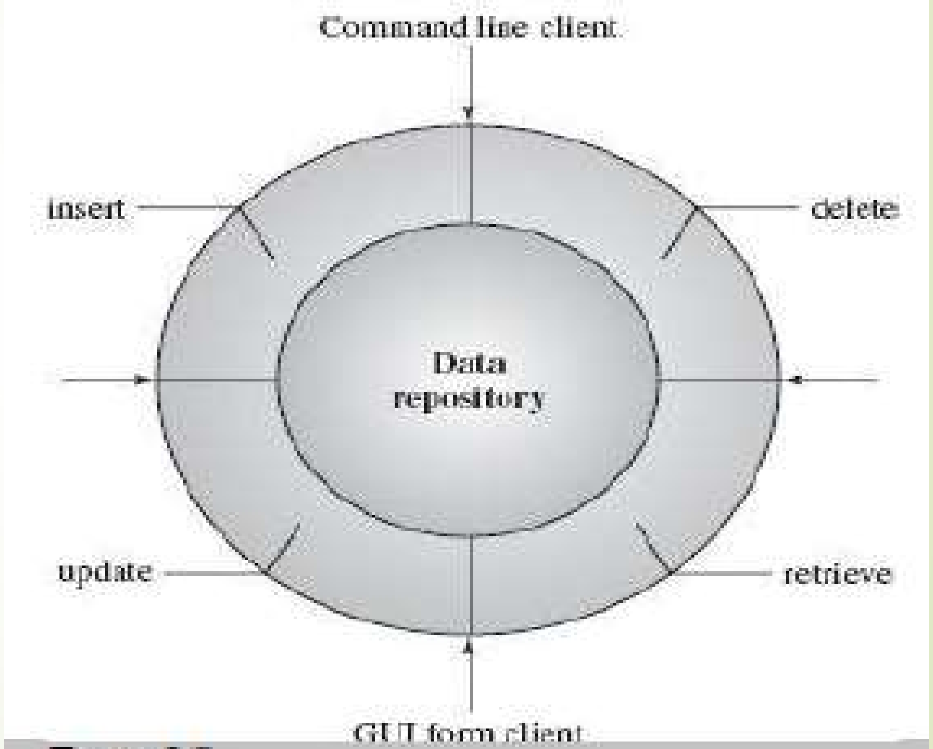


Figure 6.5
Database system

Application domain of repository architecture

2. **Computer Aided Software Engineering (CASE) system:**

- Many CASE tools surrounding the data store can generate different products for different purposes based on the same set of data.
- User of CASE tools can draw a UML diagram and store the design blueprints in the repository. These UML diagrams can then be converted to other diagrams
- Java or C++ skeleton code generation from diagrams and vice versa by reverse engineering tool.

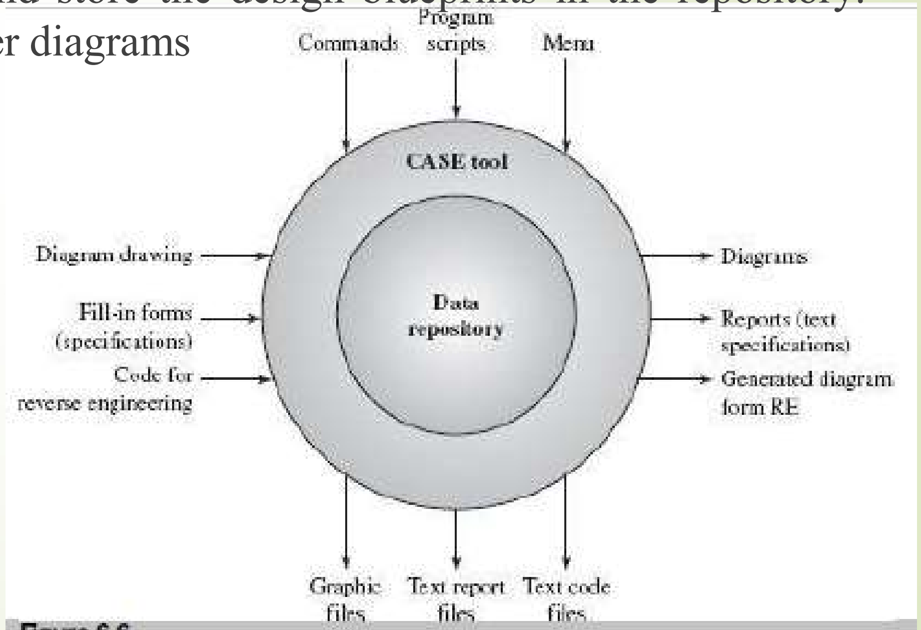


Figure 6.6
CASE system

Application domain of repository architecture

3. **Compiler construction**: Every compiler system has its own reserved keyword table, identifier symbol table, constant table generated after lexical analysis, and syntax and semantics trees generated by syntax and semantics analysis → all these stored in memory and shared by all phases of the compilation

- Each phase will generate new data or update the existing data in the data repository.

- The flow control is controlled by a program

Repository Architecture Style

Benefits:

- Data integrity: easy to back up and restore
- System scalability and reusability of agents
- Reduces the overhead of transient data between software components

Limitations:

- Data store reliability and availability are important issues. Centralized repository is vulnerable to failure compared to distributed repository with data replication.
- High dependency between data structure of data store and its agents.
- Cost of moving data on network if data is distributed

Practical Application Examples:

1. Version Control Systems:

•**How it works:**

A central repository (e.g., Git) stores code changes. Developers interact with the repository to commit, push, pull, and merge changes.

•**Use Case:** Software development projects using tools like GitHub, GitLab, or Bitbucket.

2. Content Management Systems (CMS):

•**How it works:**

A repository stores all content (e.g., articles, media files) and metadata. Components like editors, viewers, and publishing modules interact with the repository.

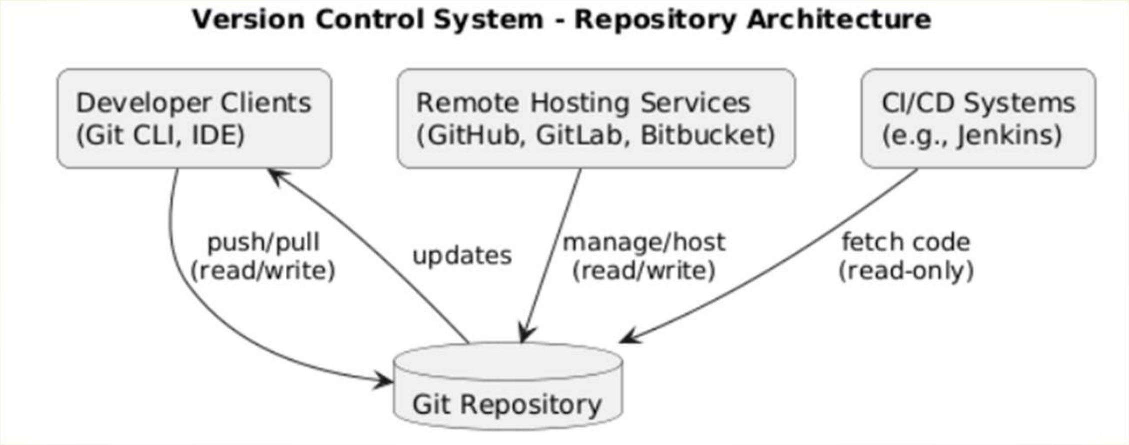
•**Use Case:** Websites built on WordPress, Drupal, or Joomla.

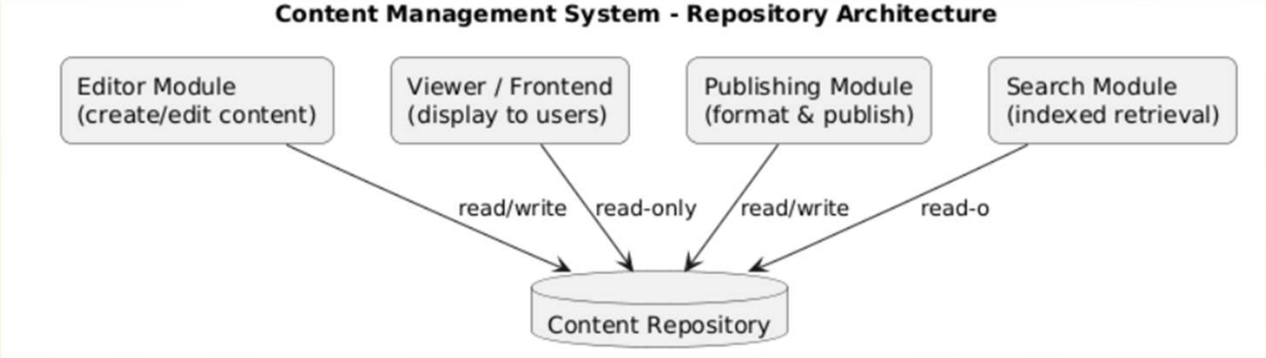
3. Database Management Systems (DBMS):

•**How it works:**

A central database acts as the repository for storing, retrieving, and managing data, with multiple applications interacting with it.

•**Use Case:** Enterprise applications like ERP (SAP, Oracle) or CRM (Salesforce).


Version Control System - Repository Architecture
Developer Clients
(Git CLI, IDE)
Remote Hosting Services
(GitHub, GitLab, Bitbucket)
CI/CD Systems
(e.g., Jenkins)
push/pull
(read/write)
updates
manage/host
(read/write)
fetch code
(read-only)
Git Repository


Content Management System - Repository Architecture
Editor Module
(create/edit content)
Viewer / Frontend
(display to users)
Publishing Module
(format & publish)
Search Module
(indexed retrieval)
read/write
read-only
read/write
read-o
Content Repository

Cont….

4. Healthcare Systems:

•**How it works:**

A central repository maintains patient records, lab results, and imaging data. Various hospital departments access and update this repository.

•**Use Case:** Electronic Health Records (EHR) systems like Epic or Cerner.

5. E-Commerce Platforms:

•**How it works:**

The repository stores product catalogs, user data, and transaction details. Components like recommendation engines, payment gateways, and inventory management interact with it.

•**Use Case:** Amazon, Shopify, or eBay.

References

- Chapter 5; book 3: “Software Architecture Design illuminated”