

Software Engineering

(303105254)



UNIT-4

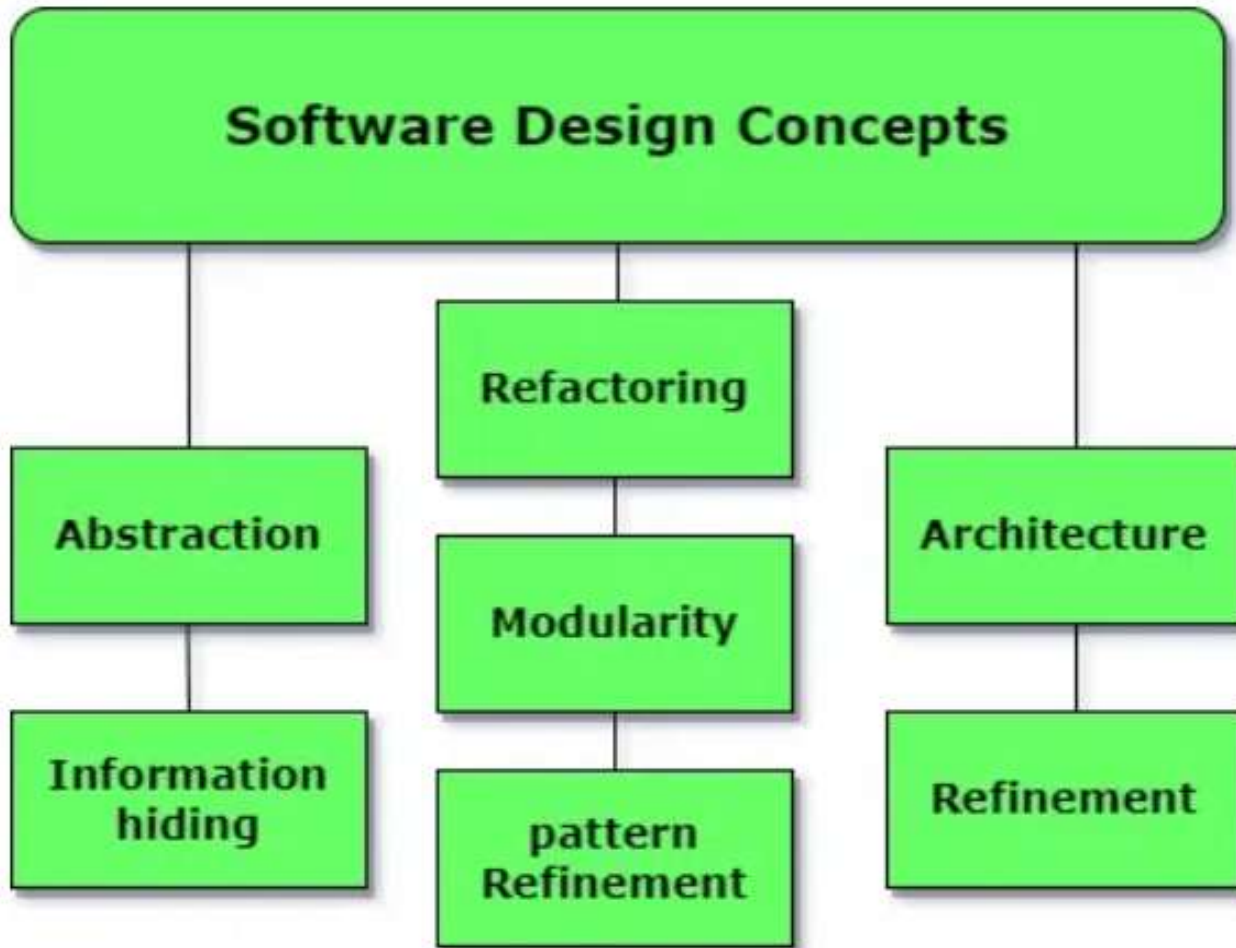
Structured System Design

Design Concepts

- ❖ Software design is the **process of transforming user requirements into a suitable form**, which helps the programmer in software coding and implementation.
- ❖ During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, **this phase aims to transform the SRS document into a design document.**



Design Concepts



Software Design Concepts



Design Concepts

❖ **Abstraction (Hide Irrelevant data):** Abstraction simply means to **hide the details** to reduce complexity and increase efficiency or quality.

❖ **Modularity (subdivide the system):** Modularity simply means **dividing the system or project into smaller parts** to reduce the complexity of the system or project. In the same way, modularity in design means **subdividing a system into smaller parts so that these parts can be created independently** and then use these parts in different systems to perform different functions.

❖ **Architecture (design a structure of something):** Architecture simply means **a technique to design a structure of something**. Architecture in designing software is a **concept that focuses on various elements and the data of the structure**. These components interact with each other and use the data of the structure in architecture.

Design Concepts

❖ **Refinement (removes impurities):** Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner which means elaborating a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

❖ **Pattern (a Repeated form):** A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.



Design Concepts

❖ **Information Hiding (Hide the Information):** Information hiding simply means to **hide the information** so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by **designing the modules in a manner that the information gathered or contained in one module is hidden** and can't be accessed by any other modules.

❖ **Refactoring (Reconstruct something):** Refactoring simply means **reconstructing something** in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behavior or its functions. Fowler has defined refactoring as “**the process of changing a software system in a way that it won't impact the behavior of the design and improves the internal structure**”.



Design Models

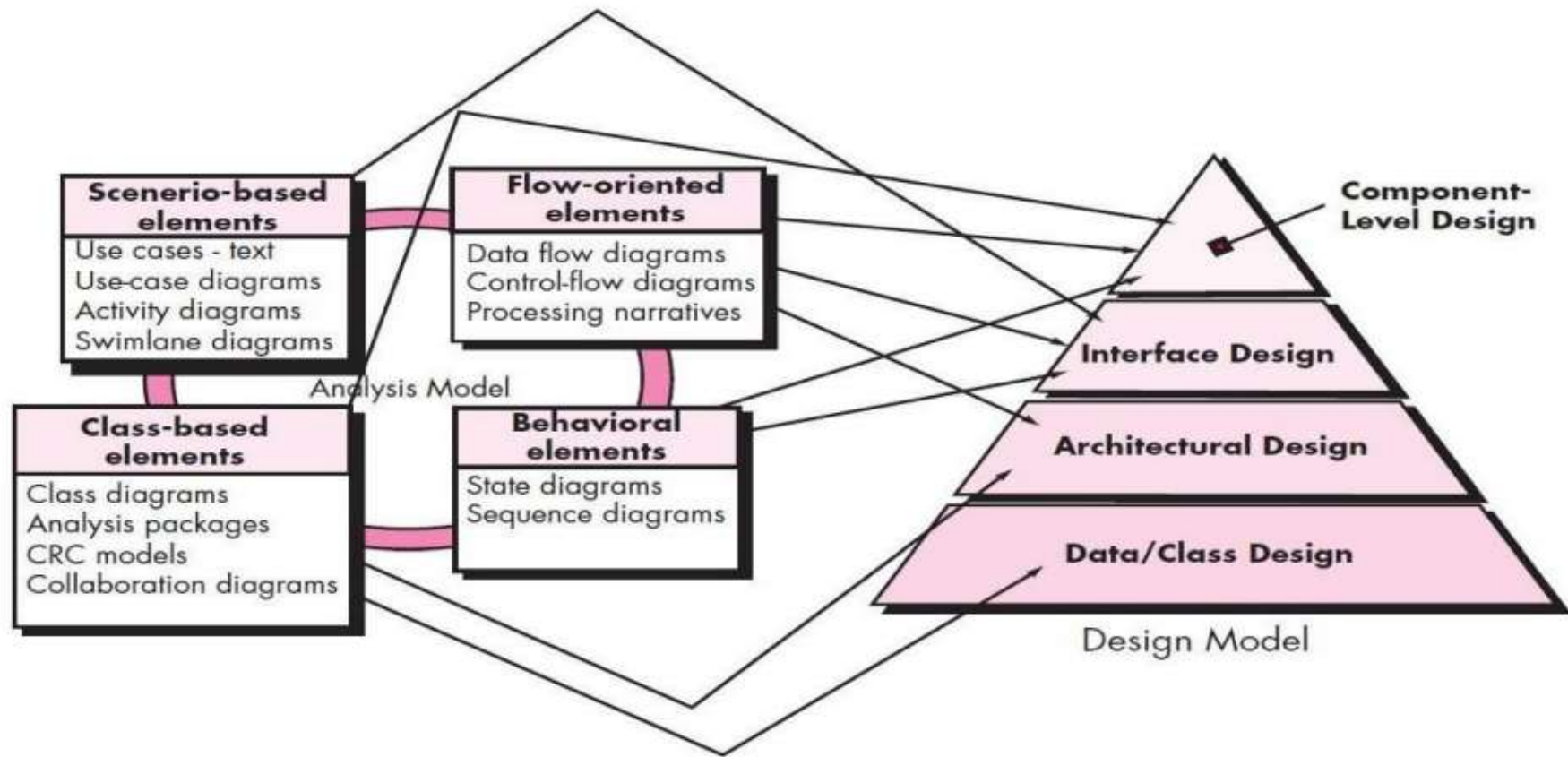


Fig : Translating the requirements model into the design model



Cont....

- ❖ The design model can be viewed in two different dimensions.
- ❖ The **process dimension** indicates the evolution of the design model as design tasks are executed as part of the software process.
- ❖ The **abstraction dimension** represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.
- ❖ The design model has four major elements: **data, architecture, components, and interface.**



Cont....

- ❖ **Data design** (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data).
- ❖ This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system.
- ❖ The structure of data has always been an important part of software design.



Cont....

- ❖ At the program **component level**, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high- quality applications.
- ❖ At the **application level**, the translation of a data model into a database is pivotal to achieving the business objectives of a system.
- ❖ At the **business level**, the collection of information stored in disparate databases and reorganized into a “data warehouse” enables data mining or knowledge discovery that can have an impact on the success of the business itself.



Cont....

- ❖ The **architectural design** for software is the equivalent to the floor plan of a house.
- ❖ The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms.
- ❖ **Architectural design elements give us an overall view of the software.**



Cont....

- ❖ The architectural model is derived from three sources:
 - (1) **information about the application** domain for the software to be built;
 - (2) **specific requirements model elements** such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand; and
 - (3) **the availability of architectural styles and patterns.**



Cont....

- ❖ The **interface design** for software is analogous to a set of detailed drawings for the doors, windows, and external utilities of a house.
- ❖ There are three important elements of interface design:
 - (1) the user interface (UI);
 - (2) external interfaces to other systems, devices, networks, or other producers or consumers of information; and
 - (3) internal interfaces between various design components.
- ❖ These interface design elements allow the software to communicate externally and enable internal communication and collaboration among the components that populate the software architecture.



Cont....

- ❖ The **component-level design** for software is the equivalent to a **set of detailed drawings for each room in a house**.
- ❖ These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, sinks, showers, tubs, drains, cabinets, and closets.
- ❖ The component-level design for **software fully describes the internal detail of each software component**.
- ❖ To accomplish this, the component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations.



Cont....

❖ **Deployment-level design** elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

❖ Deployment diagrams begin in descriptor form, where the **deployment environment is described in general terms**. Later, instance form is used and elements of the configuration are explicitly described.



Software Architecture

- ❖ Architecture serves as a **blueprint for a system**.
- ❖ It provides an **abstraction** to manage the system complexity and establish a communication and coordination mechanism among components.
- ❖ It defines a **structured solution to meet all the technical and operational requirements**, while optimizing the common quality attributes like performance and security.
- ❖ **Bass, Clements, and Kazman** define this elusive term in the following way: “**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 among them.**”



Software Architecture

- ❖ The architecture is not the operational software. Rather, it is a representation that enables you to
 - (1) **analyze the effectiveness of the design in meeting its stated requirements,**
 - (2) **consider architectural alternatives at a stage when making design changes is still relatively easy, and**
 - (3) **reduce the risks associated with the construction of the software.**
- ❖ **Bass and his colleagues identify three key reasons** that software architecture is important:
 - Representations of software architecture are an enabler for **communication** between all parties (stakeholders) interested in the development of a computer-based system.



Software Architecture

- The architecture **highlights early design decisions** that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture “**constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together**” The architectural design model and the architectural patterns contained within it are transferable



ARCHITECTURAL STYLES

❖ An architectural style as a **descriptive mechanism** to differentiate the house from other styles.

❖ The software that is built for computer-based systems also exhibits one of many architectural styles. Each style describes a system category that encompasses

- (1) **a set of components** (e.g., a database, computational modules) that perform a function required by a system;
- (2) **a set of connectors** that enable “communication, coordination and cooperation” among components;
- (3) **constraints** that define how components can be integrated to form the system; and
- (4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.



Data Centered Architectures

- ❖ A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store
- ❖ Client software accesses a central repository. In some cases the data repository is passive.

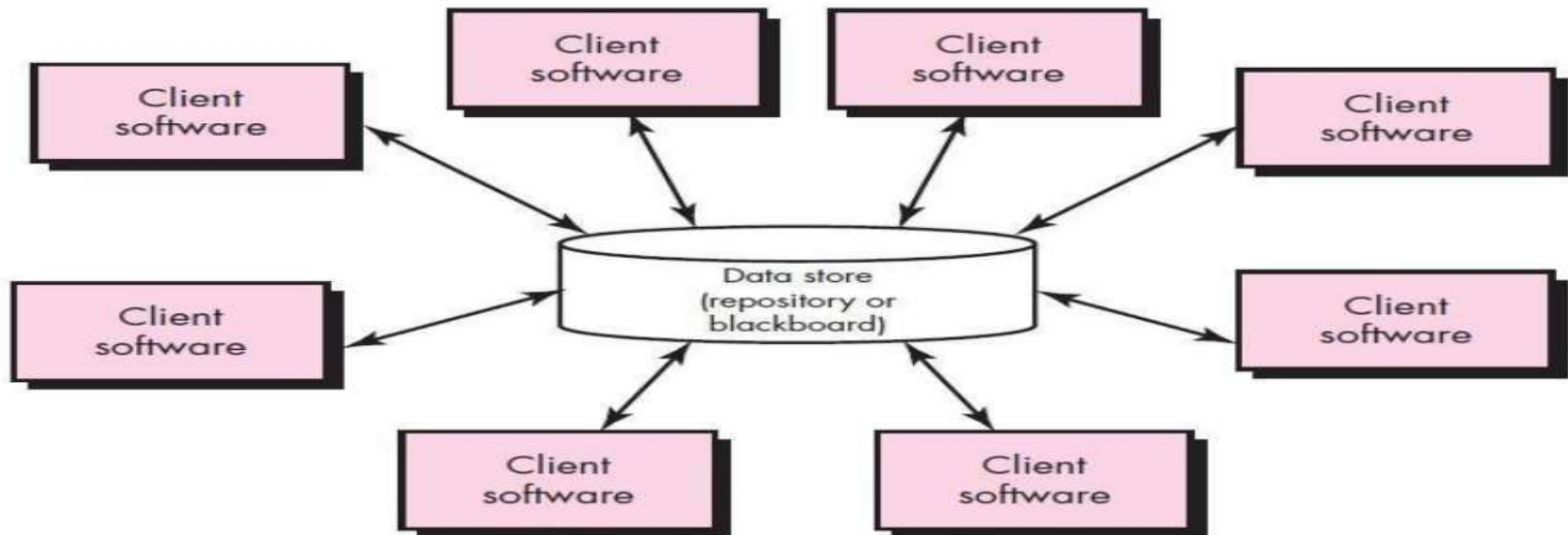


Fig : Data-centered architecture

Data Flow Architectures

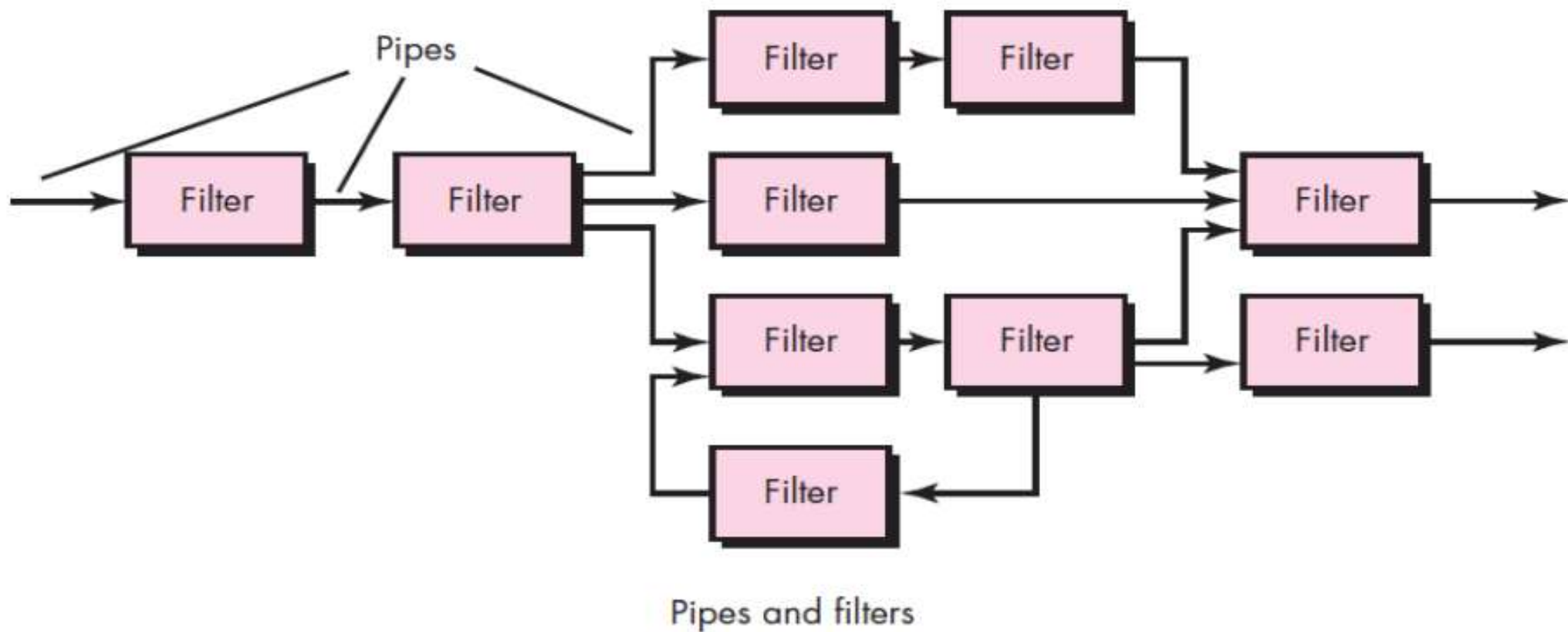


Fig : Data-flow architecture

Cont....

- ❖ This architecture is applied when **input data are to be transformed through a series of computational or manipulative components into output data**
- ❖ It has a set of components, called **filters**, connected by **pipes** that transmit data from one component to the next.
- ❖ Each **filter works independently** of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output of a specified form.
- ❖ However, **the filter does not require knowledge of the Workings of its neighboring filters.**



Call and Return Architecture

- ❖ This architectural style enables you to achieve a program structure that is relatively easy to modify and scale.
- ❖ A number of sub styles exist within this category:
 - **Main program/subprogram architectures:** This classic program structure decomposes function into a control hierarchy where a “main” program invokes a number of program components that in turn may invoke still other components.
 - **Remote procedure call architectures:** The components of a main program/subprogram architecture are distributed across multiple computers on a network.



Call and Return Architecture

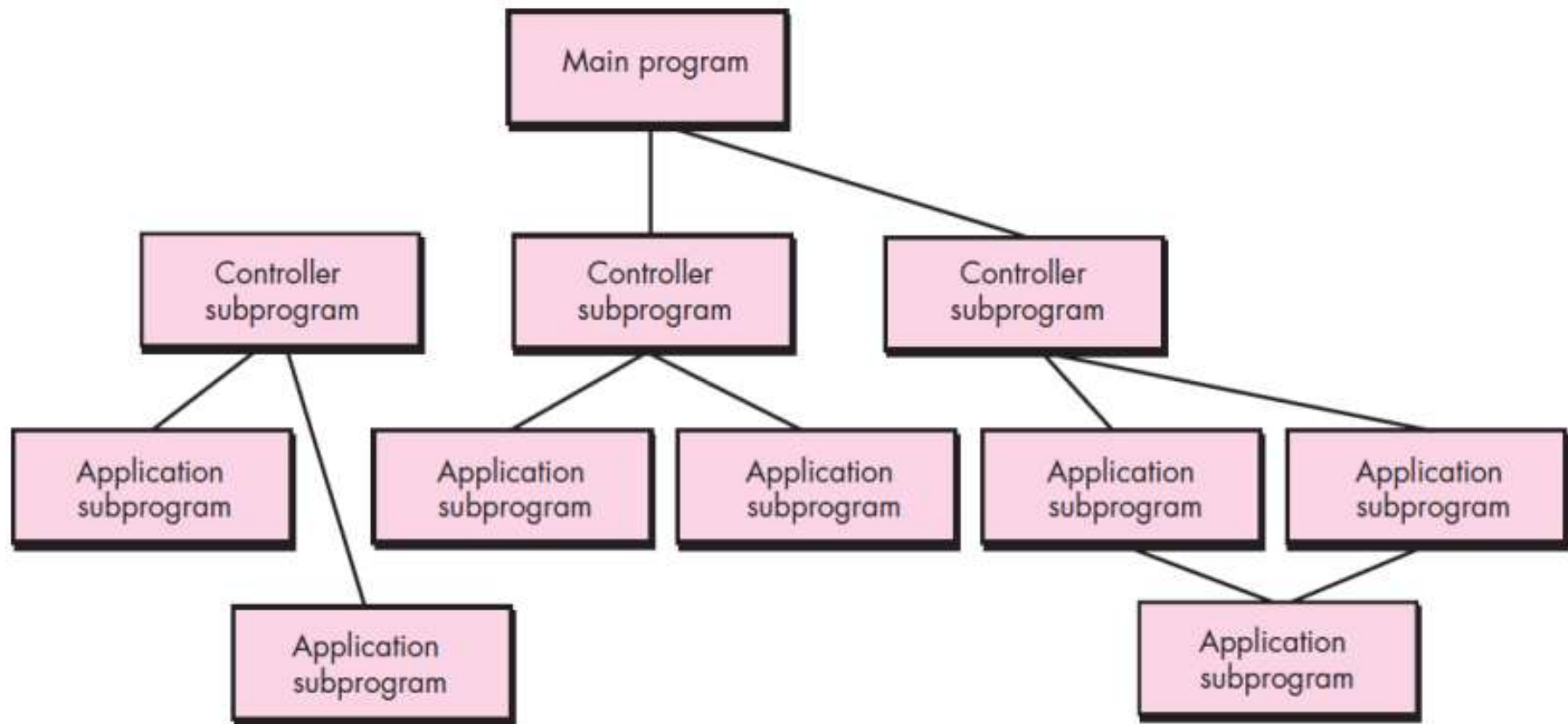


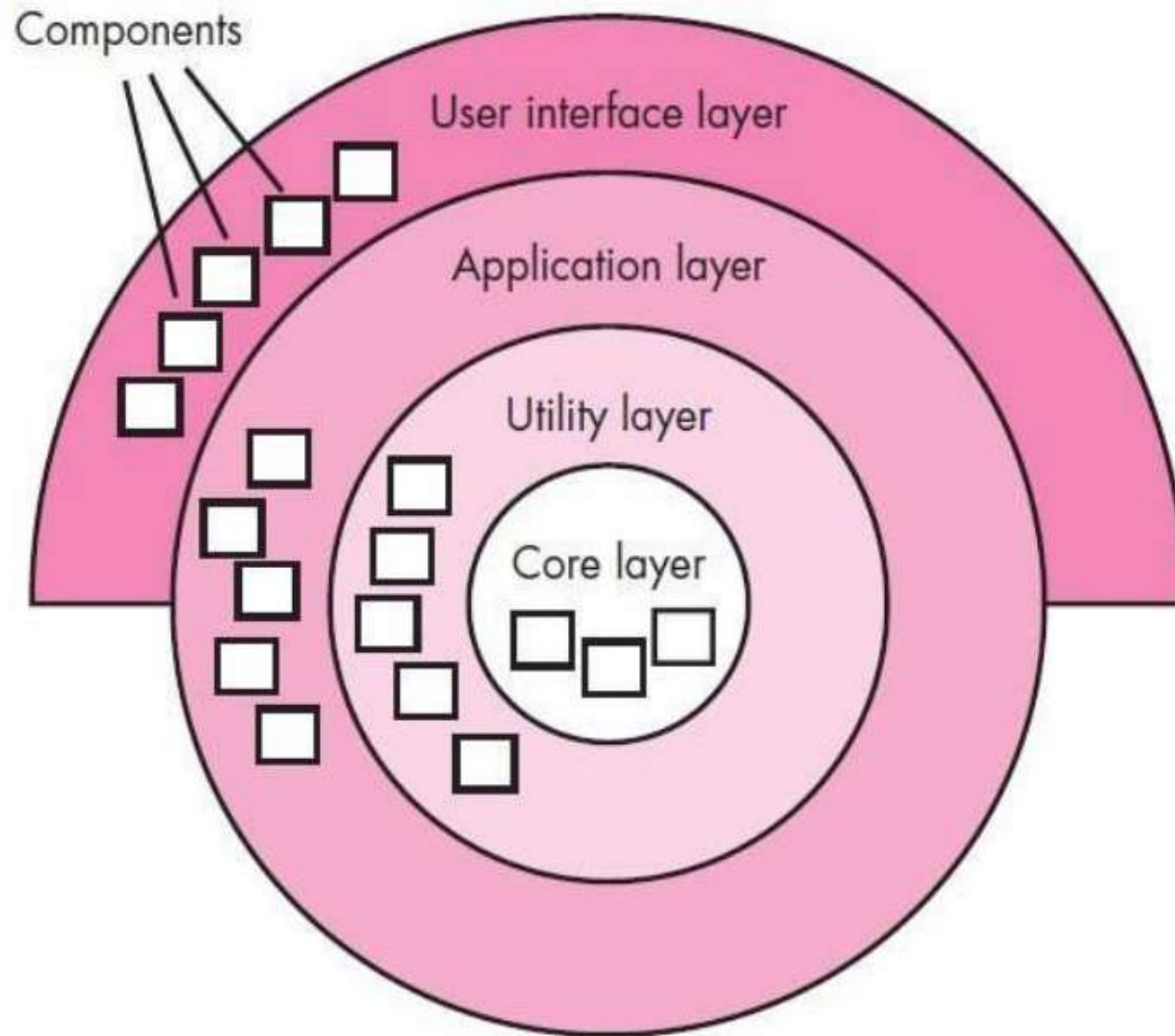
Fig : Main program/subprogram architecture



Architectural styles

- ❖ **Object-oriented architectures:** The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- ❖ **Communication and coordination between components are accomplished via message passing**
- ❖ **Layered architectures:** A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
 - ❖ At the outer layer, components service user interface operations.
 - ❖ At the inner layer, components perform operating system interfacing.
 - ❖ Intermediate layers provide utility services and application software functions.

Architectural styles



Architectural Patterns

Types of Software Architecture Patterns

1. Layered Architecture Pattern
2. Client-Server Architecture Pattern
3. Event-Driven Architecture Pattern
4. Microkernel Architecture Pattern
5. Microservices Architecture Pattern
6. Space-Based Architecture Pattern
7. Master-Slave Architecture Pattern
8. Pipe-Filter Architecture Pattern
9. Broker Architecture Pattern
10. Peer-to-Peer Architecture Pattern



Architectural Patterns

Layered Architectural Pattern

- ❖ As the name suggests, components(code) in this **pattern are separated into layers of subtasks** and they are arranged one above another.
- ❖ Each layer has unique tasks to do and all the layers are independent of one another.
- ❖ Since **each layer is independent, one can modify the code inside a layer without affecting others**. It is the most commonly used pattern for designing the majority of software.
- ❖ This layer is also known as **‘N-tier architecture’**. Basically, this pattern has 4 layers.

Presentation layer: The user interface layer where we see and enter data into an application.)



Architectural Patterns

Layered Architectural Pattern

❖ **Business layer:** This layer is responsible for executing business logic as per the request.)

❖ **Application layer:** This layer acts as a medium for communication between the 'presentation layer' and 'data layer'.

❖ **Data layer:** This layer has a database for managing data.

Advantages:

Scalability: Individual layers in the architecture can be scaled independently to meet performance needs.

Flexibility: Different technologies can be used within each layer without affecting others.

Maintainability: Changes in one layer do not necessarily impact other layers, thus simplifying the maintenance.



Architectural Patterns

Layered Architectural Pattern

Disadvantages:

- ❖ **Complexity:** Adding more layers to the architecture can make the system more complex and difficult to manage.
- ❖ **Performance Overhead:** Multiple layers can introduce latency due to additional communication between the layers.
- ❖ **Strict Layer Separation:** Strict layer separation can sometimes lead to inefficiencies and increased development effort.



Architectural Patterns

Client-Server Architectural Pattern

❖ The client-server pattern has two major entities. They are a server and multiple clients.

❖ Here the server has resources(data, files or services) and a client requests the server for a particular resource. Then the server processes the request and responds back accordingly.

❖ Advantages:

Centralized Management: Servers can centrally manage resources, data, and security policies, thus simplifying the maintenance.

Scalability: Servers can be scaled up to handle increased client requests.

Security: Security measures such as access controls, data encryption can be implemented in a better way due to centralized controls.

Architectural Patterns

Client-Server Architectural Pattern

❖ Disadvantages:

Single Point of Failure: Due to centralized server, if server fails clients lose access to services, leading loss of productivity.

Costly: Setting up and maintaining servers can be expensive due to hardware, software, and administrative costs.

Complexity: Designing and managing a client-server architecture can be complex.



Architectural Patterns

Event-Driven Architectural Pattern

- ❖ Event-Driven Architecture is an agile approach in which services (operations) of the **software are triggered by events**.
- ❖ When a user takes action in the application built using the EDA approach, a **state change happens and a reaction is generated that is called an event**.

Advantages:

Scalability: System can scale horizontally by adding more consumers.

Real-time Processing: This enables real-time processing and immediate response to events.

Flexibility: New event consumers can be added without modifying existing components.



Architectural Patterns

Event-Driven Architectural Pattern

Disadvantages:

Complexity: The architecture can be complex to design, implement, and debug.

Complex Testing: Testing event-driven systems can be complicated compared to synchronous systems.

Reliability: Ensuring reliability requires additional mechanisms to handle failed events.



Architectural Patterns

Microkernel Architectural Pattern

- ❖ Microkernel pattern has two major components. They are a **core system and plug-in modules**.
- ❖ The **core system handles the fundamental and minimal operations** of the application.
- ❖ The **plug-in modules handle the extended functionalities** (like extra features) and customized processing.

Advantages:

Flexibility: New functionalities can be added easily through plug-ins.

Scalability: The system can scale by adding more plug-ins to handle more tasks.

Maintainability: Plug-ins are developed and tested independently which makes maintenance easier.



Architectural Patterns

Microkernel Architectural Pattern

Disadvantages:

Complex Communication: Managing communication between the core systems and plug-ins can be complex.

Lack Built-in Functionalities: Due to minimalistic design the basic functionalities are absent that are common in monolithic architectures.

Complex Design: Designing the microkernel and its communication mechanisms can be challenging.



Architectural Patterns

Microservices Architectural Pattern

- ❖ The **collection of small services that are combined to form the actual application** is the concept of microservices pattern.
- ❖ Instead of building a bigger application, **small programs are built for every service** (function) of an application independently.
- ❖ And those **small programs are bundled together** to be a full-fledged application. So adding new features and modifying existing microservices without affecting other microservices are no longer a challenge when an application is built in a microservices pattern.
- ❖ **Modules in the application of microservices patterns are loosely coupled.** So they are easily understandable, modifiable and scalable.



Architectural Patterns

Microservices Architectural Pattern

Advantages:

Scalability: Each service can be scaled independently based on demand.

Faster Delivery: Independent services allows teams to develop, test, and deploy features faster.

Easier Maintenance: Services can be updated and maintained independently.

Disadvantages:

Complex Management: Managing multiple services requires robust monitoring and management tools.

Network Congestion: Increased network traffic between services can lead to congestion and overhead.

Security: Securing multiple services and their communication increases the probability of attack.



Architectural Patterns

Space based Architectural Pattern

❖ Space-Based Architecture Pattern is also known as **Cloud-Based or Grid-Based Architecture Pattern**.

❖ It is designed to **address the scalability issues** associated with large-scale and high-traffic applications.

❖ This pattern is built around the concept of **shared memory space that is accessed by multiple nodes**.

Advantages:

Scalability: The system can be easily scaled horizontally by adding more processing units.

Performance: In-memory data grids reduces the data access latency.

Flexibility: Modular components allow for flexible deployment.



Architectural Patterns

Space based Architectural Pattern

Disadvantages:

Complexity: Designing and managing distributed system is complex.

Cost: The infrastructure for space-based architecture pattern requires multiple servers and advanced middleware, which can be expensive.

Network Latency: Communication between distributed components can introduce network latency.



Architectural Patterns

Master-Slave Architectural Pattern

- ❖ The Master-Slave Architecture Pattern is also known as Primary-Secondary Architecture.
- ❖ It involves a **single master component** and that **controls multiple slave components**.
- ❖ The **master components assign tasks to slave components** and the **slave components report the results of task execution back to the master**.
- ❖ This is often used for parallel processing and load distribution.

Advantages:

Scalability: The system can scale horizontally by adding more slave units to handle increased load.

Fault Tolerance: If slave fails, then the master can reassign the tasks to some another slave. thus enhancing the fault tolerance.



Architectural Patterns

Master-Slave Architectural Pattern

Performance: Parallel execution of tasks can improve the performance of the system.

Disadvantages:

Single Point of Failure: The master component is a single point of failure. If the master fails then the entire system can collapse.

Complex Communication: The communication overhead between master and slave can be significant especially in large systems.

Latency: Systems' responsiveness can be affected by the latency introduced by master-slave communication.



Architectural Patterns

Pipe-Filter Architectural Pattern

❖ Pipe-Filter Architecture Pattern structures a system around a series of processing elements called filters that are connected by pipes.

❖ Each filter processes data and passes it to the next filter via pipe.

Advantages:

Reusability: Filters can be reused in different pipelines or applications.

Scalability: Additional filters can be added to extend the functionality to the pipeline.

Parallelism: Filters can be executed in parallel if they are stateless, thus improving performance.



Architectural Patterns

Pipe-Filter Architectural Pattern

Disadvantages:

Debugging Difficulty: Identifying and debugging issues are difficult in long pipelines.

Data Format constraints: Filters must agree on the data format, requiring careful design and standardization.

Latency: Data must be passed through multiple filters, which can introduce latency.



Architectural Patterns

Broker Architectural Pattern

- ❖ The **Broker** architecture pattern is designed to manage and facilitate communication between decoupled components in a distributed system.
- ❖ It involves a **broker** that acts as a intermediary to route the requests to the appropriate server.

Advantages:

Scalability: Brokers support horizontal scaling by adding more servers to handle increased load.

Flexibility: New servers can be added and the existing ones can be removed or modified without impacting the entire system.

Fault Tolerance: If a server fails, then broker can route the request to another server.



Architectural Patterns

Broker Architectural Pattern

Disadvantages:

Complex Implementation: Implementing a broker requires robust management of routing and load balancing, thus make system more complex.

Single Point of Failure: If broker is not designed with failover mechanisms then it can become a single point of failure.

Security Risks: Securing broker component is important to prevent potential vulnerabilities.



Architectural Patterns

Peer to Peer Architectural Pattern

- ❖ The Peer-to-Peer (P2P) architecture pattern is a **decentralized network model** where each node, known as a peer, acts as both a client and a server.
- ❖ There is no central authority or single point of control in P2P architecture pattern. Peers can share resources, data, and services directly with each other.

Advantages:

Scalability: The network can scale easily as more peers join.

Fault Tolerance: As data is replicated across multiple peers, this results in system being resilient to failures.

Cost Efficiency: There is no need for centralized servers, thus reducing the infrastructure cost.



Architectural Patterns

Peer to Peer Architectural Pattern

Disadvantages:

Security Risks: Decentralized nature of the architecture makes it difficult to enforce security policies.

Data Consistency: Ensuring data consistency across peers can be challenging.

Complex Management: Managing a decentralized network with numerous independent peers can be complex.



Comparison

Features	Software Architecture Pattern	Design Pattern
Definition	This is a high-level structure of the entire system.	This is a low-level solution for common software design problems within components.
Scope	Broad, covers entire system.	Narrow, focuses on individual components.
Purpose	Establish entire system layout.	Provide reusable solutions for the recurring problems within a system's implementation.
Focus	System stability, structural organization.	Behavioral and structural aspects within components.
Documentation	It involves architectural diagrams and high-level design documents.	It includes UML diagrams, detailed design specifications.
Examples	Layered Architecture, Microservices, Client-Server.	Singleton, Factory, Strategy, Observer.

Component level design

❖ Component-level design focuses on **defining software architecture through modular components**.

❖ Each component provides a specific functionality and interacts with other components to build the complete system.

❖ **Key Aspects:**

- **Modularity:** Breaking down the system into manageable and reusable components.
- **Interfaces:** Defining clear interfaces for components to interact with each other.
- **Encapsulation:** Hiding the internal workings of a component from the outside.



Component level design

❖ Modeling Techniques:

- **UML Component Diagrams:** These diagrams show the organization and relationships between components. They depict components, their interfaces, and dependencies.
- **Deployment Diagrams:** These illustrate the physical deployment of artifacts on nodes.
- **Sequence Diagrams:** Used to model interactions between components over time.



Procedural Design

❖ Procedural design focuses on defining procedures or functions to perform specific tasks. It is typically used in structured programming paradigms where the flow of control is critical.

❖ **Key Aspects:**

- **Functions/Procedures:** Breaking down tasks into smaller, manageable functions.
- **Control Flow:** Using loops, conditionals, and sequences to control the execution of procedures.
- **Data Flow:** Managing the flow of data through the system



Procedural Design

❖ Modeling Techniques:

- **Flowcharts:** Visual representations of the control flow within a program, using various symbols to denote different types of actions and decisions.
- **Data Flow Diagrams (DFD):** Illustrate how data moves through the system, showing inputs, processes, and outputs.
- **Structured Charts:** Depict the hierarchical structure of procedures and modules



Object Oriented Design

- ❖ Object-oriented design (OOD) focuses on creating systems using objects, which encapsulate data and behavior.
- ❖ OOD is built around the principles of abstraction, encapsulation, inheritance, and polymorphism.

❖ Key Aspects:

- **Classes and Objects:** Defining classes as blueprints for objects that encapsulate data and behavior.
- **Inheritance:** Allowing classes to inherit properties and methods from other classes.
- **Polymorphism:** Enabling objects to be treated as instances of their parent class.



Object Oriented Design

❖ Modeling Techniques:

- **UML Class Diagrams:** Represent the classes, attributes, methods, and relationships (such as inheritance and associations) between them.
- **UML Sequence Diagrams:** Illustrate how objects interact with each other through message passing over time.
- **UML Use Case Diagrams:** Capture functional requirements and user interactions with the system.



Comparison

Aspect	Component-Level Design	Procedural Design	Object-Oriented Design
Focus	Modularity and interfaces	Functions and control flow	Objects and interactions
Abstraction Level	Medium	Low	High
Modeling Techniques	Component Diagrams, Deployment Diagrams	Flowcharts, DFDs, Structured Charts	Class Diagrams, Sequence Diagrams
Design Principles	Encapsulation, Interface definition	Step-by-step procedures, Data manipulation	Encapsulation, Inheritance, Polymorphism



Data

- ❖ Data is a **raw and unorganized fact that is required to be processed** to make it meaningful.
- ❖ It can be considered as **facts and statistics collected together for reference or analysis**
- ❖ Data are individual units of information.
- ❖ In analytical processes, data are represented by variables.
- ❖ Data is always interpreted, by a human or machine, to derive meaning. So, data is meaningless.
- ❖ Data contains **numbers, statements, and characters** in a raw form.



Data

❖ Types of Data

1. **Quantitative:** Quantitative data refers to **numerical information** like weight, height, etc.
2. **Qualitative:** Qualitative data refers to **non-numeric information** like opinions, perceptions, etc



Information

- ❖ Information is defined as **structured, organized, and processed data, presented within a context** that makes it relevant and useful to the person who needs it.
- ❖ Information is the **knowledge that is remodeled and classified into an intelligible type**, which may be utilized in the method of deciding.
- ❖ In short, once **knowledge ends up being purposeful when conversing, it's referred to as info**. It's one thing that informs, in essence, it provides a solution to a specific question. It may be obtained from numerous sources like newspapers, the internet, television, people, books, etc.



Data vs Information

S.NO	DATA	INFORMATION
Definition	Data is defined as unstructured information such as text, observations, images, symbols, and descriptions. In other words, data provides no specific function and has no meaning on its own.	Information refers to processed, organized, and structured data. It gives context for the facts and facilitates decision making. In other words, information is processed data that makes sense to us.
Purpose	Data are the variables that help to develop ideas/conclusions.	Information is meaningful data.
Nature	Data are text and numerical values.	Information is refined form of actual data.
Dependence	Data doesn't rely on Information.	While Information relies on Data.
Measurement	Bits and Bytes are the measuring unit of data.	Information is measured in meaningful units like time, quantity, etc.
Structure	As tabular data, graphs, and data trees can be easily structured.	Information can also be structured as language, ideas, and thoughts.
Purposefulness	Data does not have any specific purpose	Information carries a meaning that has been assigned by interpreting data.
Knowledge Level	It is low-level knowledge.	It is the second level of knowledge.



Data vs Information

Decision Making	Data does not directly help in decision making.	Information directly helps in decision making.
Meaning	Data is a collection of facts, which itself has no meaning.	Information puts those facts into context.
Example	Example of data is student test scores.	Example of information is average score of class that is derived from given data.



E-R Diagram

- ❖ The Entity Relationship Diagram **explains the relationship among the entities present** in the database.
- ❖ ER models are **used to model real-world objects** like a person, a car, or a company and the relation between these real-world objects.
- ❖ In short, the ER Diagram is the structural format of the database.








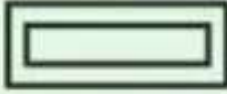
E-R Diagram

ER diagrams are **used to represent the E-R model in a database**, which makes them easy to convert into relations (tables).

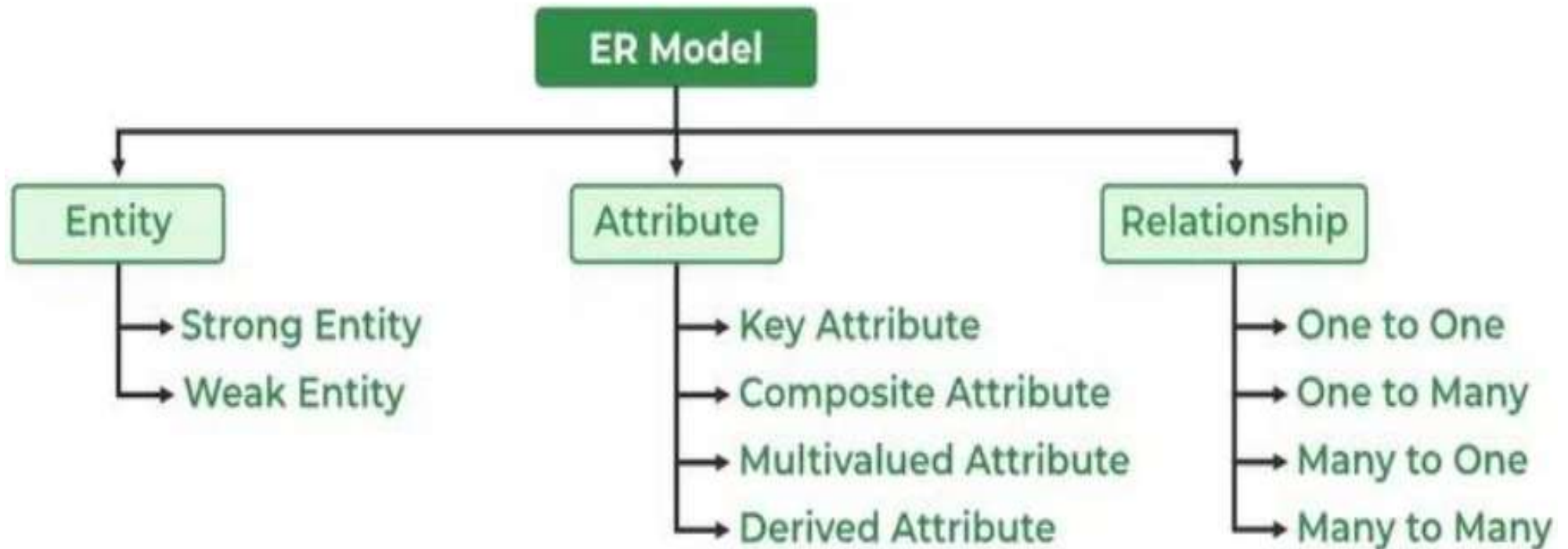
- ER diagrams provide the **purpose of real-world modeling of objects** which makes them intently useful.
- ER diagrams **require no technical knowledge** and no hardware support.
- These diagrams are **very easy to understand and easy to create even for a naive user**.
- It gives a standard solution for **visualizing the data logically**.



E-R Diagram

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

E-R Diagram- Components



Components of ER Diagram



E-R Diagram

❖ **Entity:** An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

❖ **Entity are of two types**

1.Tangible Entity – Which can be touched like car , person etc.

2.Non – tangible Entity – Which can't be touched like air , bank account etc.



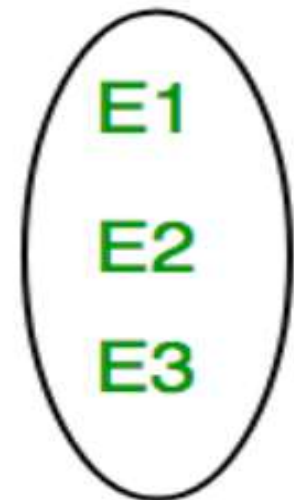
E-R Diagram

❖ Entity Set: An Entity is an object of Entity Type and a **set of all entities is called an entity set.**

❖ For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity Type



Entity Set

E-R Diagram

❖ We can represent the entity set in ER Diagram but can't represent entity in ER Diagram because entity is row and column in the relation and ER Diagram is graphical representation of data.

1. Strong Entity :

- A Strong Entity is a type of **entity that has a key Attribute**. Strong Entity **does not depend** on other Entity in the Schema.
- It has a **primary key**, that helps in identifying it uniquely, and it is **represented by a rectangle**.



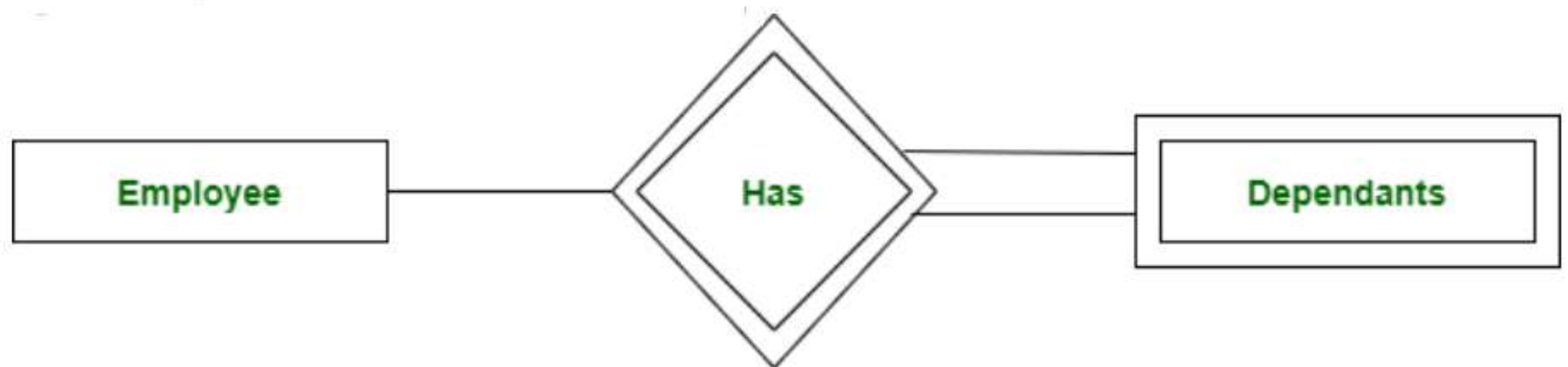
E-R Diagram

1. Weak Entity :

- An Entity **type** has a **key attribute** that **uniquely identifies each entity in the entity set**. But some entity type exists for which key attributes can't be defined.
- A weak entity type is **represented by a Double Rectangle**. The participation of weak entity types is always total.
- The relationship between the weak entity type and its identifying strong entity type is called **identifying relationship** and it is **represented by a double diamond**.

E-R Diagram

- ❖ **For Example**, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents can't exist without the employee.
- ❖ So Dependent will be a Weak Entity Type and Employee will be Identifying Entity type for Dependent, which means it is Strong Entity Type.



E-R Diagram

- ❖ **Attribute:** The properties that define the entity type. **For example**, Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student

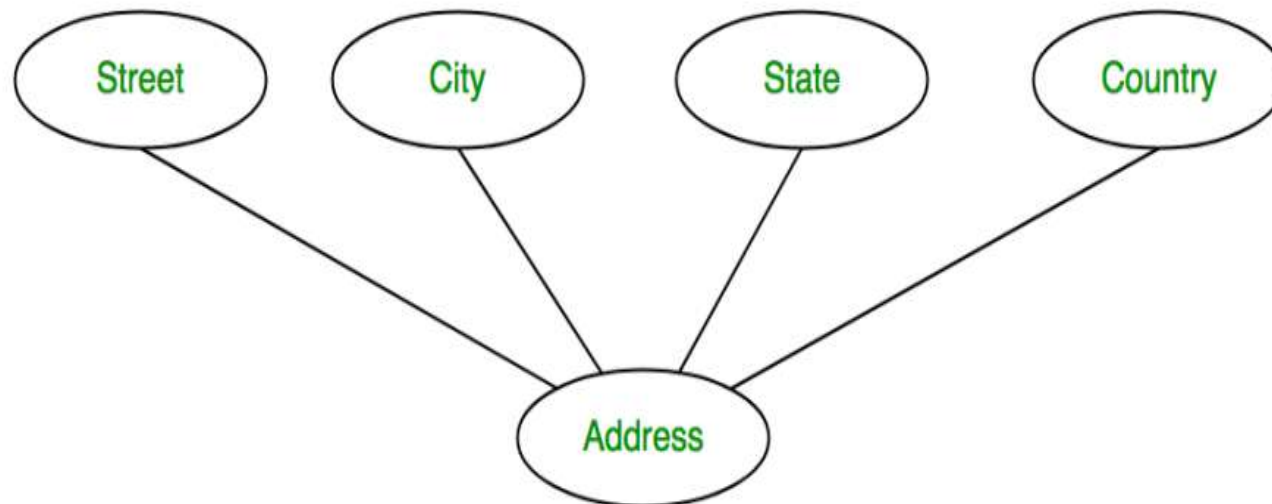


- ❖ **Key Attribute:** The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. **For example**, Roll_No will be unique for each student



E-R Diagram

- ❖ **Composite Attribute:** An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country



E-R Diagram

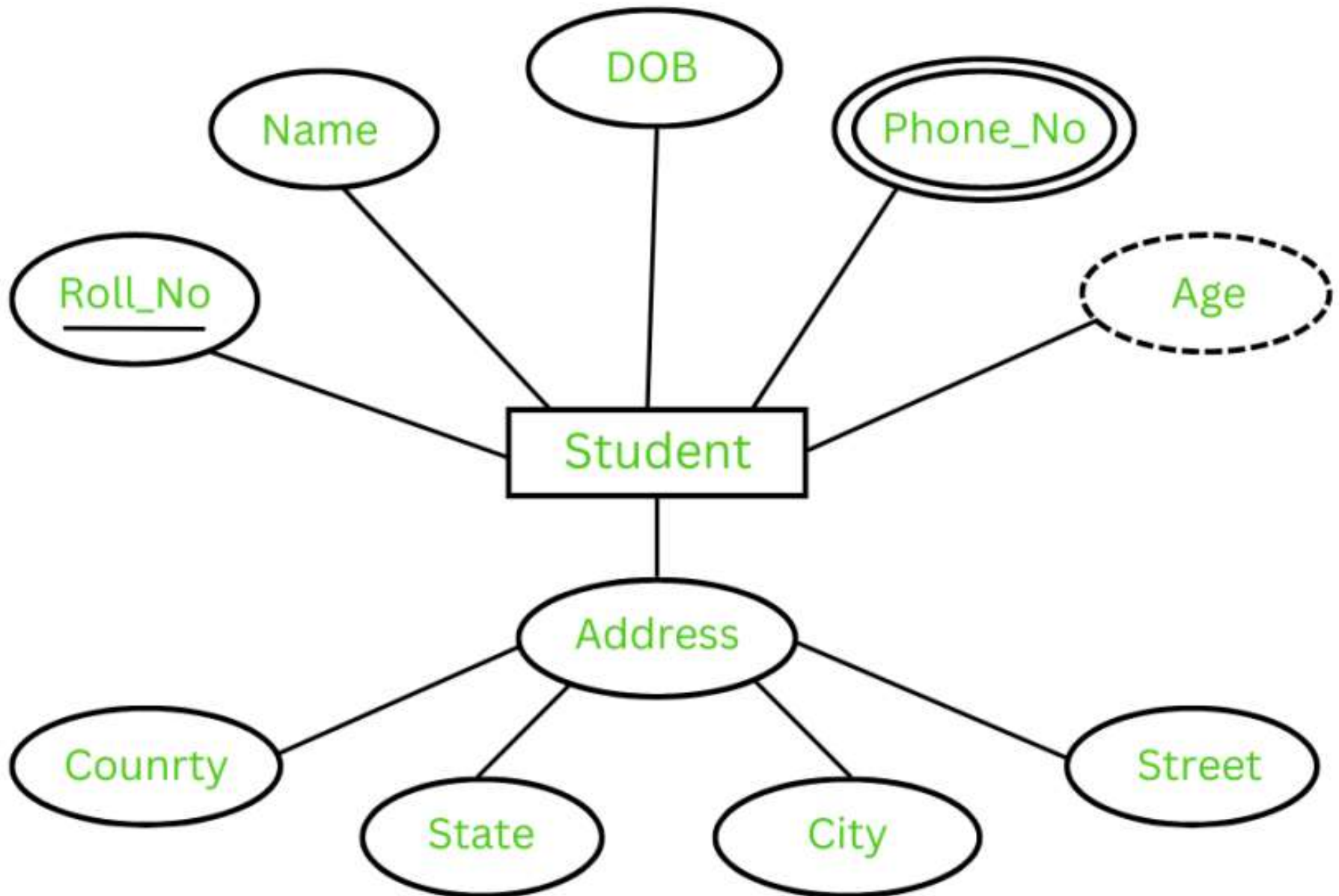
- ❖ **Multivalued Attribute:** An attribute consisting of **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student)



- ❖ **Derived Attribute** An attribute that can be derived from other attributes of the entity type is known as a derived attribute.
e.g.; Age (can be derived from DOB)

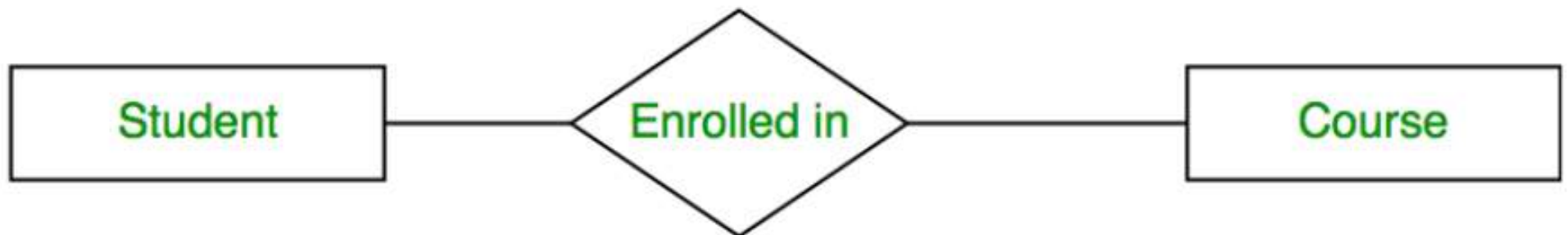


E-R Diagram



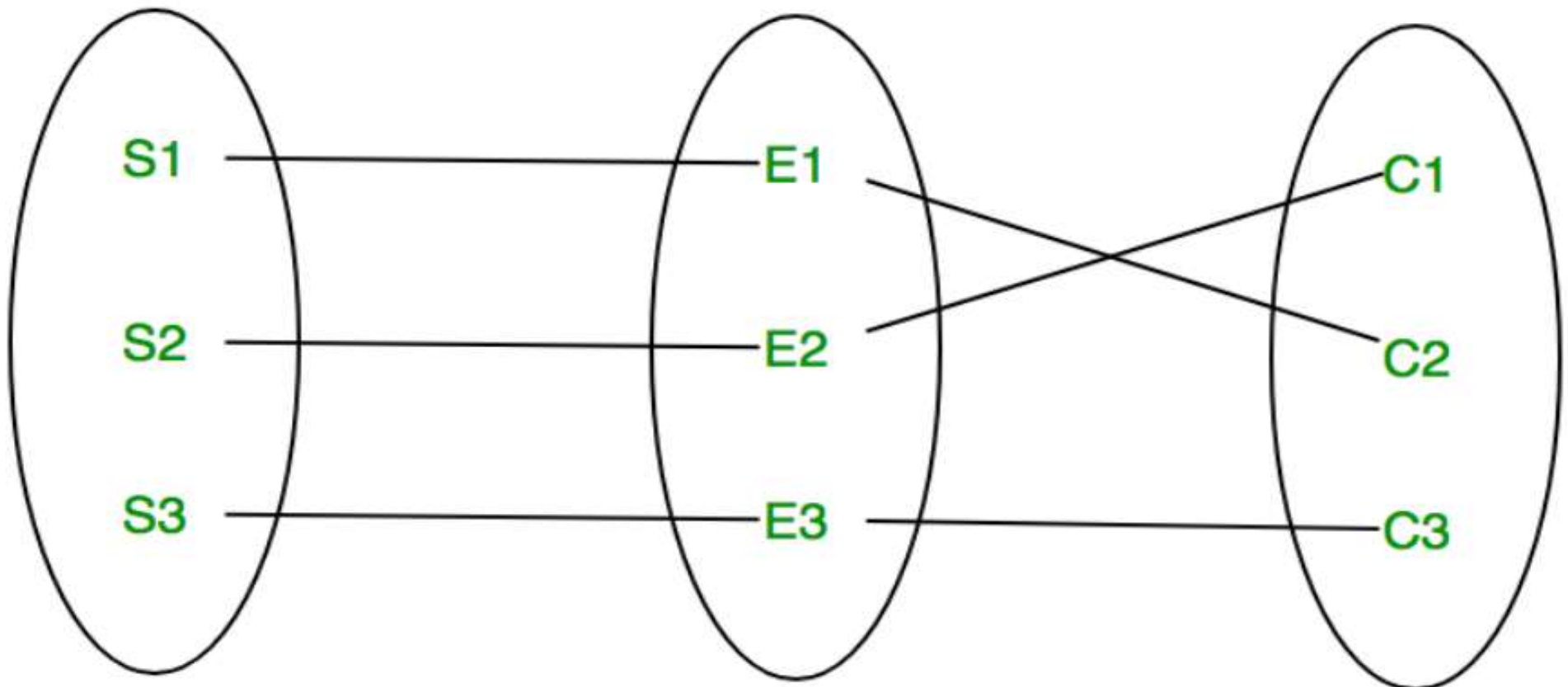
E-R Diagram

- ❖ **Relationship Type and Relationship Set:** A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course.



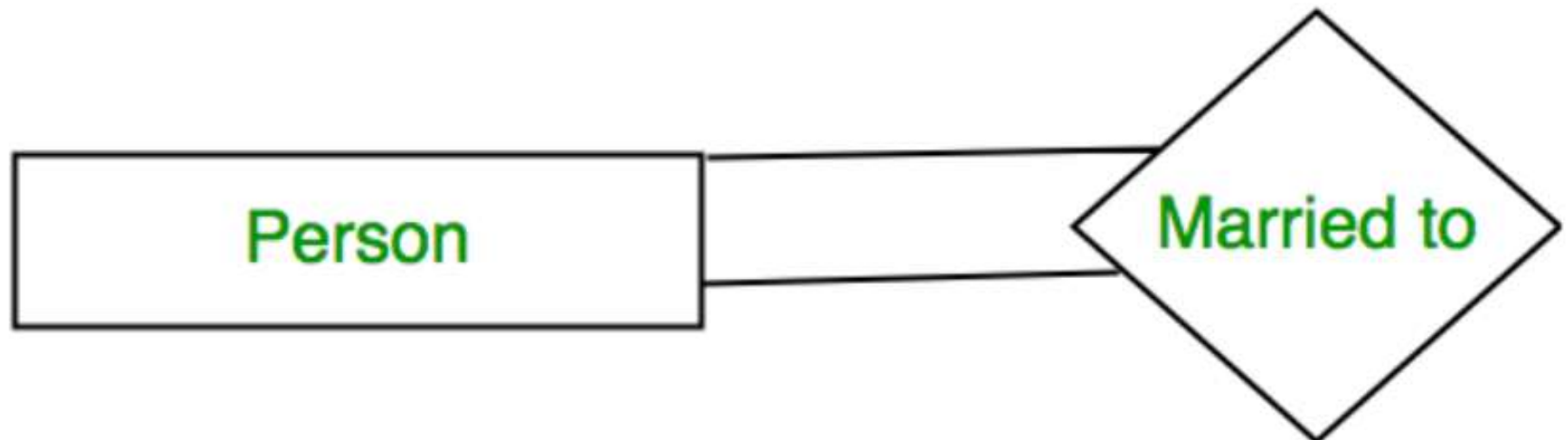
E-R Diagram

- ❖ **Entity-Relationship Set:** A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.



E-R Diagram

- ❖ **Degree of a Relationship Set:** The number of different entity sets participating in a relationship set is called the degree of a relationship set.
- ❖ **Unary Relationship:** When there is only **ONE** entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



E-R Diagram

- ❖ **Binary Relationship:** When there are **TWO entities set participating in a relationship**, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



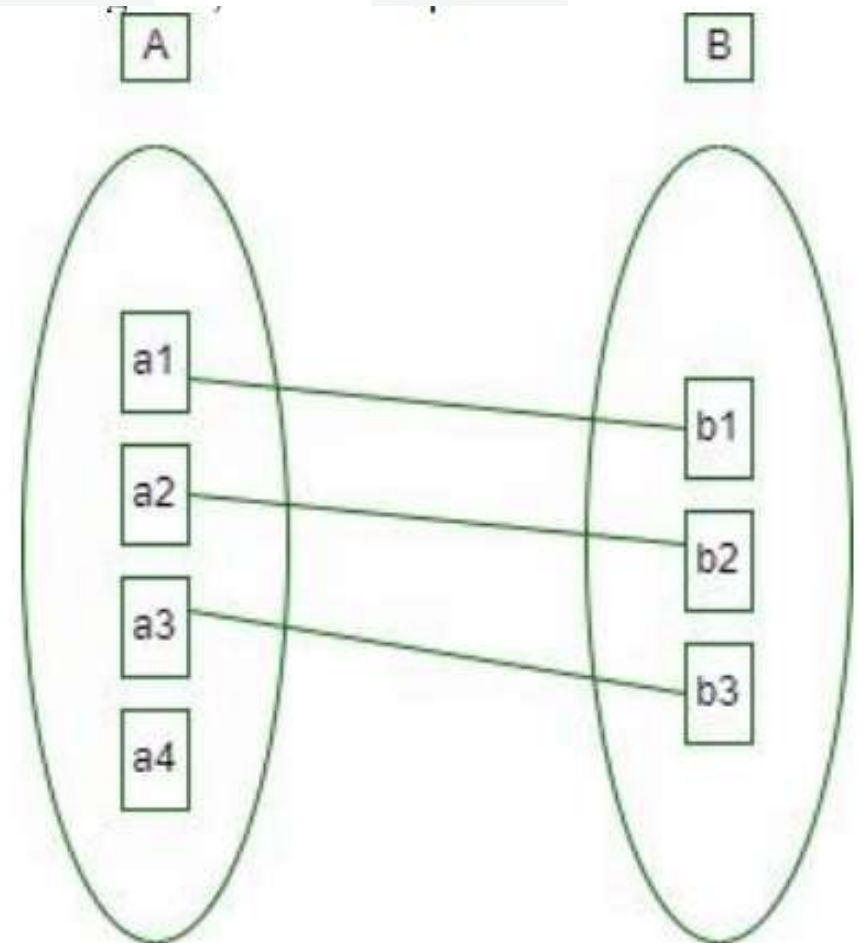
E-R Diagram

- ❖ Ternary Relationship: When there are **n entities set participating in a relation**, the relationship is called an n-ary relationship.
- ❖ **Cardinality**: The **number of times an entity of an entity set participates in a relationship set** is known as cardinality



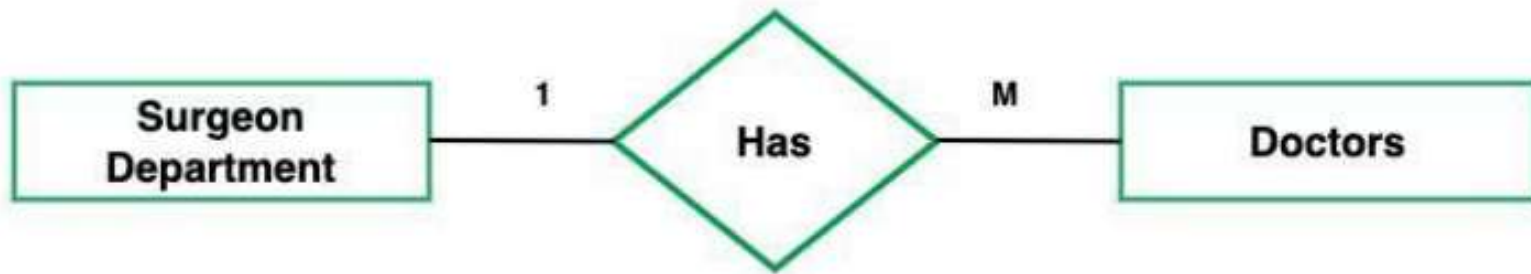
E-R Diagram

- ❖ **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one



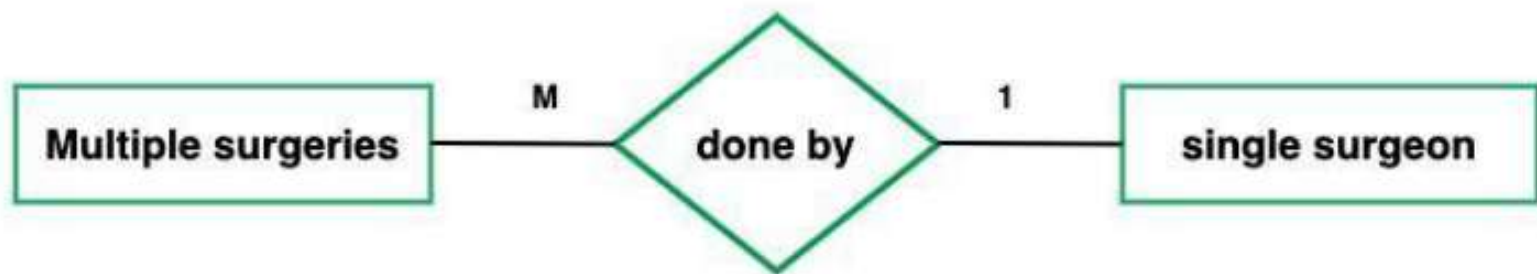
E-R Diagram

- ❖ **One-to-Many:** In one-to-many mapping as well where **each entity can be related to more than one entity**



E-R Diagram

- ❖ **Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one



E-R Diagram

- ❖ **Many-to-Many:** When entities in all entity sets can take part more than once in the relationship cardinality is many to many

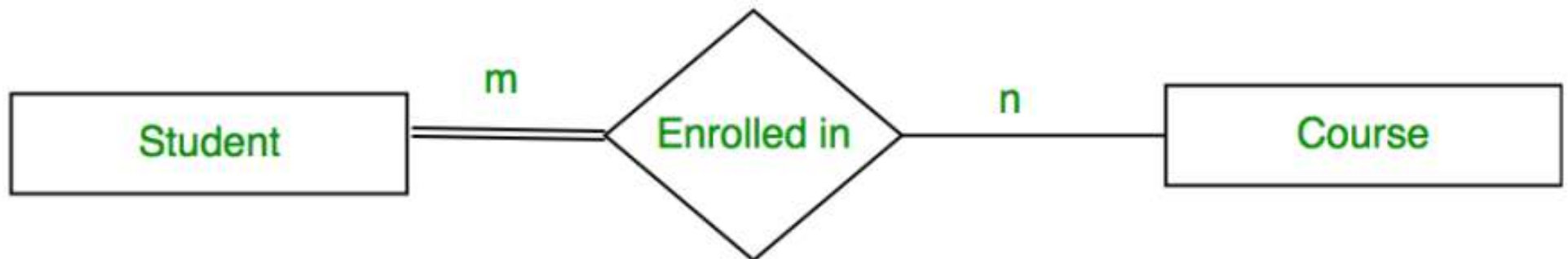


E-R Diagram

- ❖ **Participation Constraint:** It is applied to the entity participating in the relationship set.
- ❖ **Total Participation** – Each entity in the **entity set must participate in the relationship**. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a **double line** in the ER diagram.
- ❖ **Partial Participation** – The **entity in the entity set may or may NOT participate in the relationship**. If some courses are not enrolled by any of the students, the participation in the course will be partial



E-R Diagram



E-R Diagram

How to Draw ER Diagram?

- The very **first step** is **Identifying all the Entities**, and place them in a Rectangle, and labeling them accordingly.
- The **next step** is to **identify the relationship** between them and place them accordingly using the Diamond, and make sure that, Relationships are not connected to each other.
- **Attach attributes** to the entities properly.
- **Remove redundant entities** and relationships.
- **Add proper colors to highlight** the data present in the database



Data Flow Model

- ❖ DFD is the abbreviation for Data Flow Diagram.
- ❖ The **flow of data in a system or process** is represented by a Data Flow Diagram (DFD). It also gives **insight into the inputs and outputs** of each entity and the process itself.
- ❖ Data Flow Diagram (DFD) **does not have a control flow and no loops or decision rules** are present.
- ❖ Specific operations, depending on the type of data, can be explained by a flowchart. It is a graphical tool, useful for communicating with users, managers and other personnel. it is useful for analyzing existing as well as proposed systems.



Data Flow Model

❖ It provides an overview of

- What data is system processes.
- What transformation are performed.
- What data are stored.
- What results are produced , etc

❖ Data Flow diagrams are very popular because they help us **to visualize the major steps and data involved in software-system processes.**



Data Flow Model

❖ **Characteristics of Data Flow Diagram (DFD)**

❖ **Graphical Representation:** Data Flow Diagram (DFD) use **different symbols and notation to represent data flow** within system. That simplify the complex model.

❖ **Problem Analysis:** Data Flow Diagram (DFDs) are very useful in **understanding a system** and can be effectively used during analysis. Data Flow Diagram (DFDs) are quite general and are not limited to problem analysis for software requirements specification.



Data Flow Model

❖ **Abstraction:** Data Flow Diagram (DFD) provides a abstraction to complex model i.e. DFD **hides unnecessary implementation details and show only the flow of data and processes** within information system.

❖ **Hierarchy:** Data Flow Diagram (DFD) provides a hierarchy of a system. High- level diagram i.e. **0-level diagram provides an overview of entire system while lower-level diagram like 1-level DFD and beyond provides a detailed data flow of individual process.**



Data Flow Model

- ❖ **Data Flow:** The primary objective of Data Flow Diagram (DFD) is to **visualize the data flow between external entity, processes and data store**. Data Flow is represented by **an arrow** Symbol.
- ❖ **Ease of Understanding:** Data Flow Diagram (DFD) can be **easily understand** by both technical and nontechnical stakeholders.
- ❖ **Modularity:** Modularity can be achieved using Data Flow Diagram (DFD) as **it breaks the complex system into smaller module or processes**. This provides easily analysis and design of a system



Data Flow Model

❖ Types of Data Flow Diagram (DFD)

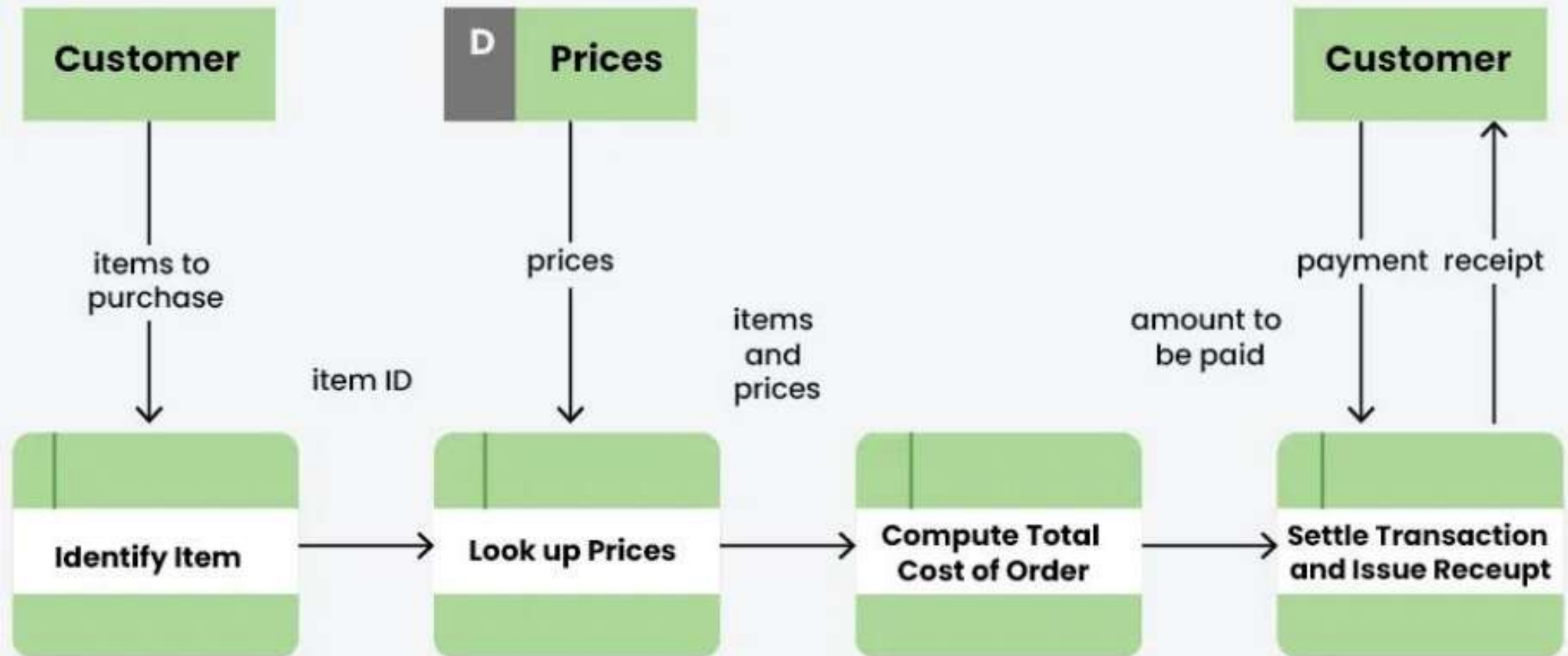
❖ Logical Data Flow Diagram (DFD) :

- ❖ mainly focuses on **the system process**
- ❖ illustrates **how data flows in the system**
- ❖ mainly focuses on **high level processes and data flow** without diving deep into technical implementation details
- ❖ used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another



Data Flow Model(Online Grocery Store)

Logical Data Flow Diagram (DFD)



Data Flow Model

❖ Types of Data Flow Diagram (DFD)

❖ Physical Data Flow Diagram (DFD) :

- ❖ Physical data flow diagram shows how the **data flow is actually implemented** in the system.
- ❖ We include additional details such as data storage, data transmission, and specific technology or system components.
- ❖ Physical DFD is more specific and close to implementation



Data Dictionary

- The data dictionary is a reference **work of data about data** (that is, *metadata*), one that is compiled by systems analysts to guide them through analysis and design.
- Data flow diagrams are an **excellent starting point for collecting data dictionary entries**.



Data Dictionary

Data Dictionary may be used to also:

- ❖ **Validate the data flow diagram** for completeness and accuracy.
- ❖ **Provide a stating point** for developing screens and reports.
- ❖ **Determine the contents of data stored** in files.
- ❖ **Develop the logic for data-flow diagram** processes.



Data Dictionary

FIELDNAME	DATATYPE	SIZE	CONSTRAINT	DESCRIPTION
Schemeid	Varchar2	20	Primary key	Stores the id of scheme
Time	Number	10	Not null	Stores the total time provided by scheme (in hours)
Days	Number	10	Not null	Stores the total days for expiring the scheme
Rupees	Number	4	Not null	Stores the amount for the schemes



Any Query?