SOFTWARE ENGINEERING

Module 3

Module-III Syllabus

Requirements, design and solution Considerations for architecture, design, Data, modules, interfaces – application architectures.

System design: modular design – cohesion and coupling, Structural Design - Top down, Bottom up approaches; data models, User Interface guidelines; UML Activity, Sequence, Component, Collaboration, Deployment diagrams

Architectural design:when to happen

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

Software Architectural design

It is concerned with understanding:

- how a software system should be organized
- Designing the overall structure of system.
- The design process for identifying the sub-systems making up a system
- framework for sub-system control and communication

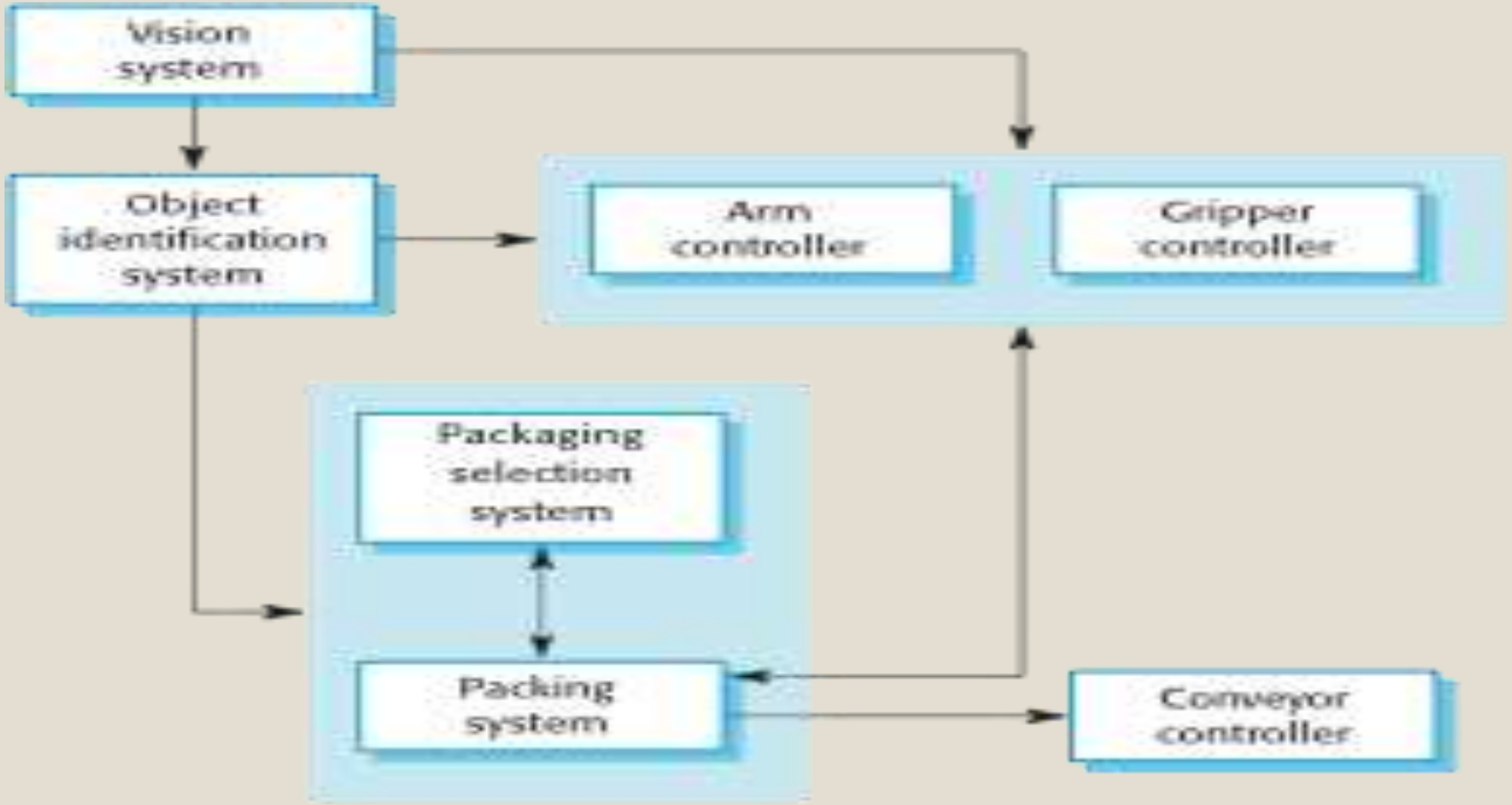
software architecture diagram elements

- **Components:** Building blocks of the software system representing the various modules, classes, services, or layers.
- **Relationships:** Lines or arrows that indicate how the components interact or communicate with each other.
- **External systems:** Entities outside the software that interact with the system, such as APIs, databases, or third-party services.
- **Interfaces:** how one component will provide or consume information from another.
- **Boundaries:** Sort related components into groups and distinguish between the subsystems within the architecture (for example, presentation layer, application layer, communication layer, etc.)

Architectural abstraction

- Abstraction in architecture is the process of simplifying a complex structure by removing non-essential details.
- It's a fundamental concept in architecture that helps designers focus on the most important aspects of a design.
- **Architecture in the small** is concerned with the architecture of individual programs.
- At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large** is concerned with the architecture of complex enterprise systems
- These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

The architecture of a packing robot control system



Box-and-Line Diagram Heuristics

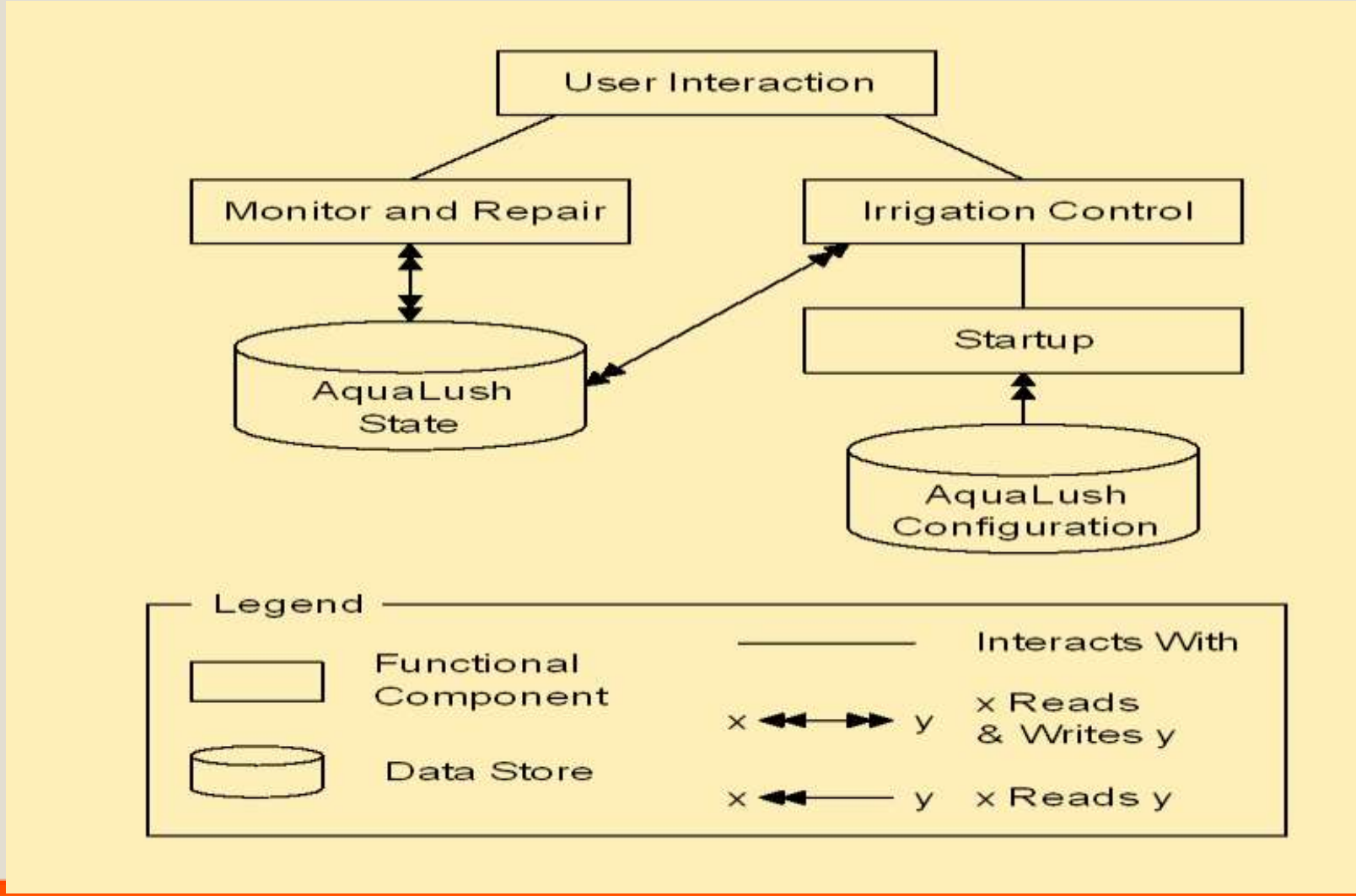
- Make box-and-line diagrams only when no standard notation is adequate.
- Keep the boxes and lines simple.
- Make symbols for different things look different.
- Use symbols consistently in different diagrams.
- Use grammatical conventions to name elements.
- Don’t mix static and dynamic elements.

8

Box and line diagrams

- Icons (boxes) connected with lines
- No rules governing formation
- Used for both static and dynamic modeling
- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

Box and line diagrams Example



Architectural design decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architecture characteristics

- **Configurability**: How easy it is for the end user to change the application's configuration
- **Extensibility**: How easy it is to extend the application's functionality
- **Reliability**: How reliable the system is, especially if it's critical to human safety
- **Robustness**: How well the system can handle errors and boundary conditions, like power outages or hardware failures
- **Scalability**: How well the system can handle increasing workloads and expanding functionality
- **Performance**: How well the system performs

4 + 1 view model of software architecture

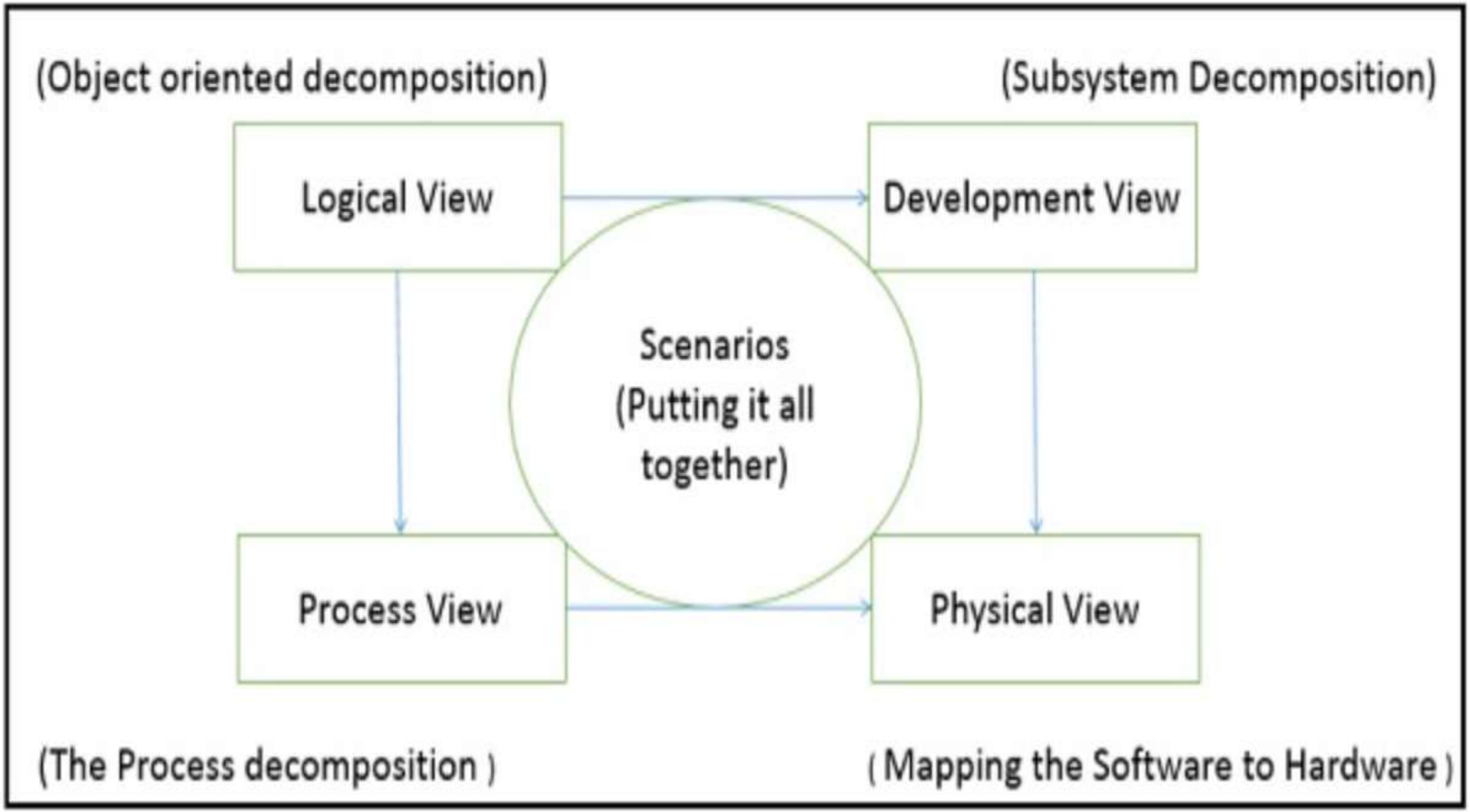
- A model is a complete, basic, and simplified description of software architecture
- Model is composed of multiple views from a particular perspective
- A view is a representation of an entire system
- It describe the system from the viewpoint of different stakeholders such as end-users, developers, project managers, and testers.
- The 4+1 View Model was designed by Philippe Kruchten to describe the architecture of a software–intensive system based on the use of multiple and concurrent views.

4 + 1 view model of software architecture

- It provides four essential views −
- **The logical view or conceptual view** − It describes the object model of the design.
- **The process view** − It describes the activities of the system, captures the concurrency and synchronization aspects of the design.
- **The physical view** − It describes the mapping of software onto hardware and reflects its distributed aspect.
- **The development view** − It describes the static organization or structure of the software in its development of environment.

4 + 1 view model of software architecture

- This view model can be extended by adding one more view called **scenario view** or **use case view** for end-users or customers of software systems.
- The **use case view** has a special significance as it details the high level requirement of a system while other views details


(Object oriented decomposition)
(Subsystem Decomposition)
Logical View
Development View
Scenarios
(Putting it all
together)
Process View
Physical View
(The Process decomposition)
(Mapping the Software to Hardware)

	Logical	Process	Development	Physical	Scenario
Description	Shows the component (Object) of system as well as their interaction	Shows the processes / Workflow rules of system and how those processes communicate, focuses on dynamic view of system	Gives building block views of system and describe static organization of the system modules	Shows the installation, configuration and deployment of software application	Shows the design is complete by performing validation and illustration
Viewer / Stake holder	End-User, Analysts and Designer	Integrators & developers	Programmer and software project managers	System engineer, operators, system administrators and system installers	All the views of their views and evaluators
Consider	Functional requirements	Non Functional Requirements	Software Module organization (Software management reuse, constraint of tools)	Nonfunctional requirement regarding to underlying hardware	System Consistency and validity
UML – Diagram	Class, State, Object, sequence, Communication Diagram	Activity Diagram	Component, Package diagram	Deployment diagram	Use case diagram

Architectural patterns

- This is a high-level structure of the entire system
- Scope is Broad, covers entire system.
- Establish entire system layout.
- Focus on System stability, structural organization.
- Patterns are a means of representing, sharing and reusing knowledge.

Types of Architectural Pattern

1. Layered Architecture Pattern
2. Client-Server Architecture Pattern
3. Pipe-Filter Architecture Pattern
4. Repository Architecture
5. Application Architecture

Layered Architecture Pattern

- The pattern is separated into layers of subtasks
- They are arranged one above another.
- Each layer has unique tasks to do
- 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.

Layered Architecture Pattern

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

Layered Architecture Pattern 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.

Layered Architecture Pattern Disadvantages:

1. **Complexity:** Adding more layers to the architecture can make the system more complex and difficult to manage.

2. **Performance Overhead:** Multiple layers can introduce latency due to additional communication between the layers.

3. **Strict Layer Separation:** Strict layer separation can sometimes lead to inefficiencies and increased development effort.

Layered Architecture Pattern

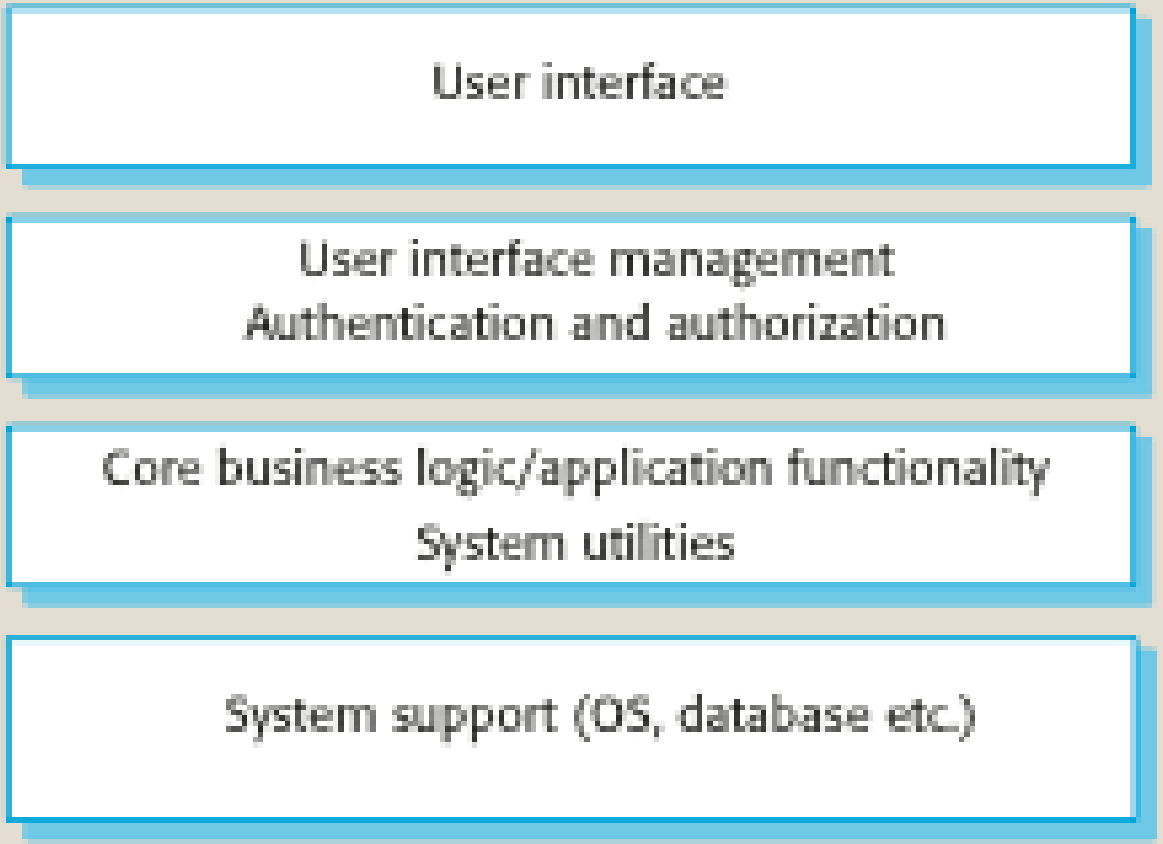
1.Enterprise Applications like Customer Relationship Management (CRM).

2.Web Applications like E-commerce platforms.

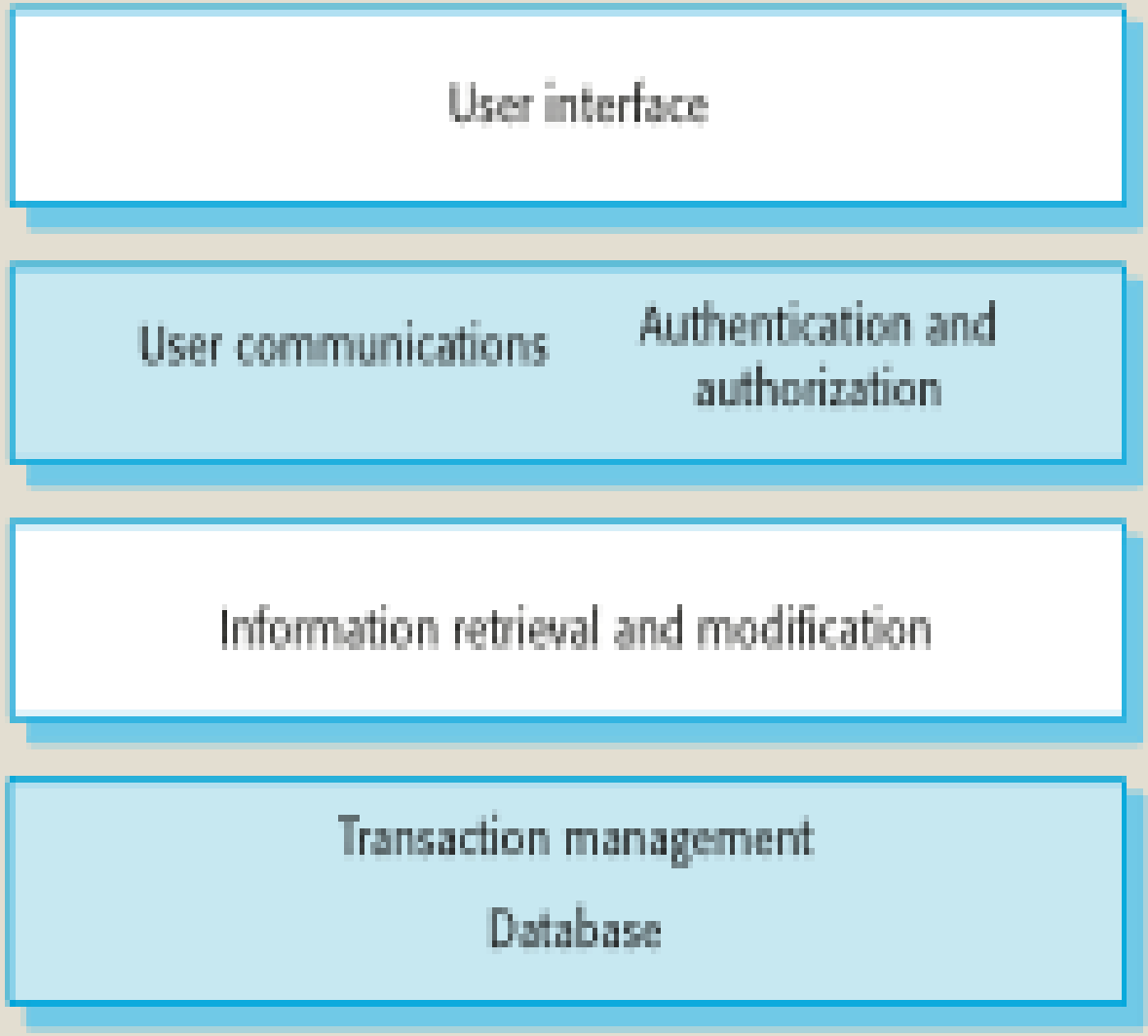
3.Desktop Applications such as Financial Software.

4.Mobile Applications like Banking applications.

A generic layered architecture

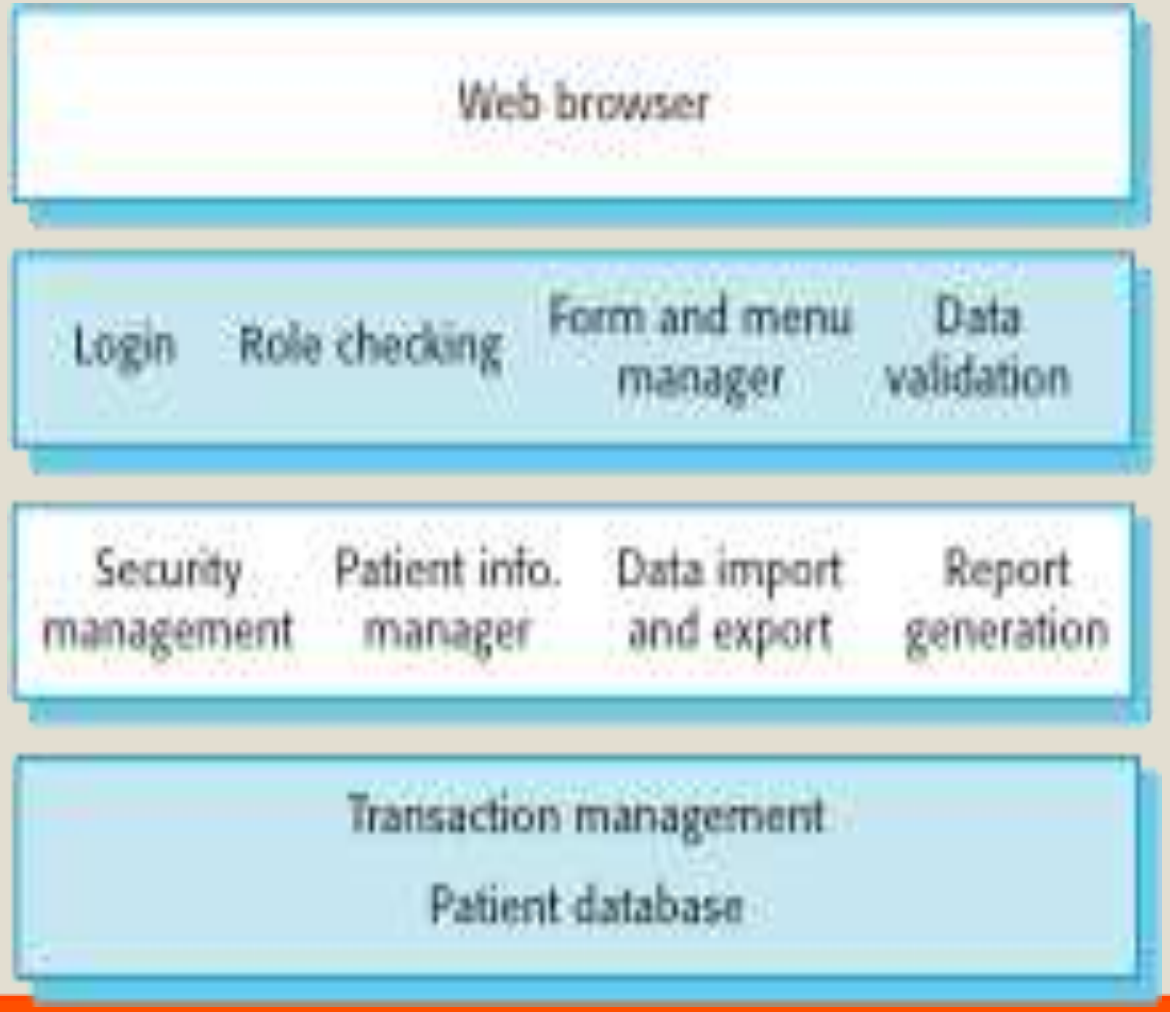


Layered information system architecture : generic architecture that can be organised as a layered architecture.

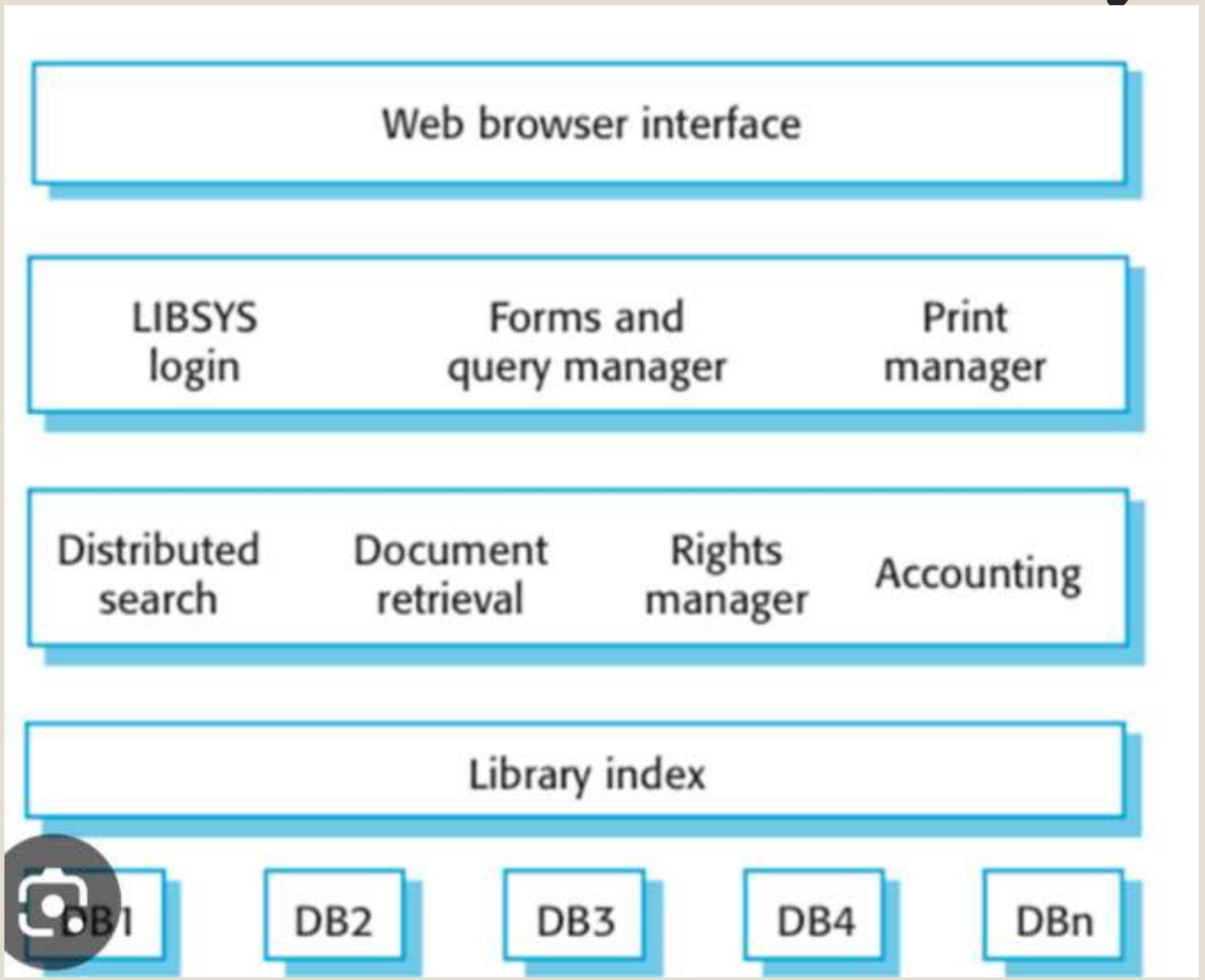


The user interface
User communications
Information retrieval
System database

The architecture of the MHC-PMS



The architecture of the LIBSYS system



Client-Server Architecture 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)
- client requests the server for a particular resource.
- Then the server processes the request and responds back accordingly.

Client-Server Architecture Pattern Advantages:

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

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

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

Client-Server Architecture Pattern Disadvantages

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

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

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

Client-Server Architecture Pattern

1.Web Applications like Amazon.

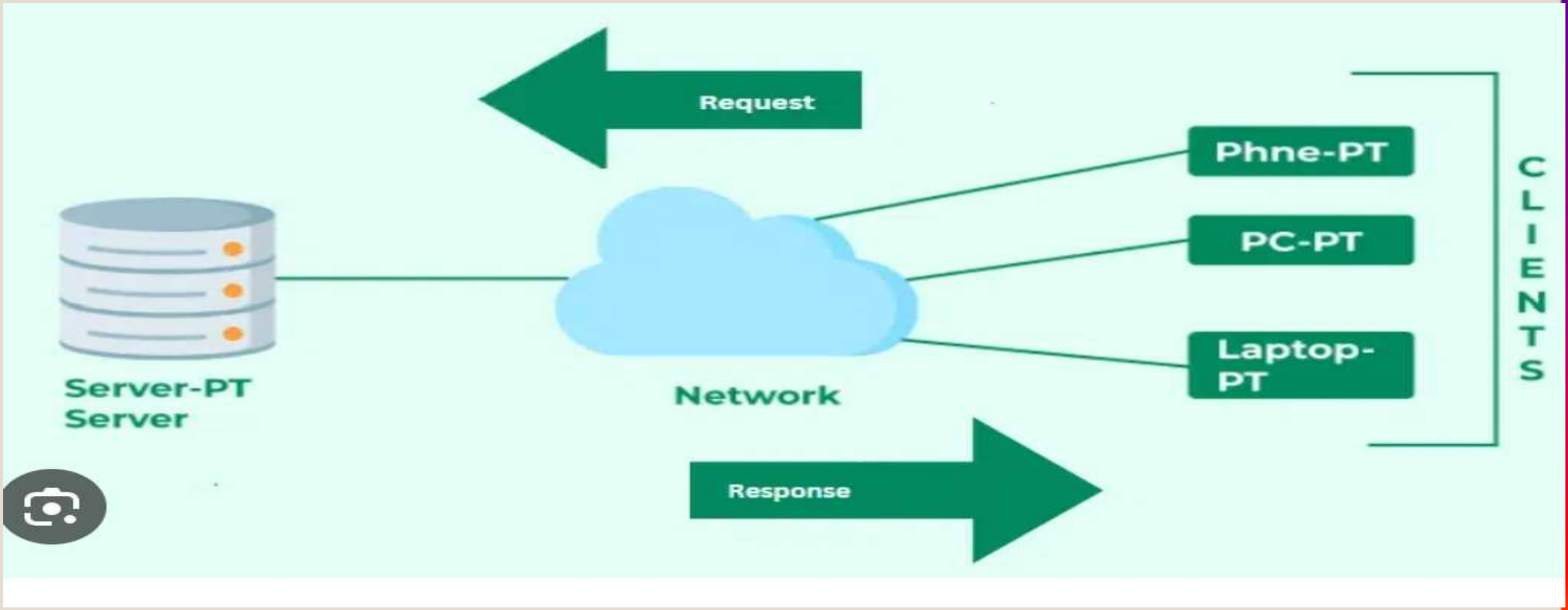
2.Email Services like Gmail, Outlook.

3.File Sharing Services like Dropbox, Google Drive.

4.Media Streaming Services like Netflix.

5.Education Platforms like Moodle.

Client-Server Architecture Pattern



Pipe-Filter Architecture Pattern

- It is a process into a series of distinct steps, called filters, linked by channels known as pipes.
- It structures a system around a series of processing elements called filters
- The filters are connected by pipes.
- Each filter processes data and passes it to the next filter via pipe.

Advantages:

1.Reusability: Filters can be reused in different pipelines

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

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

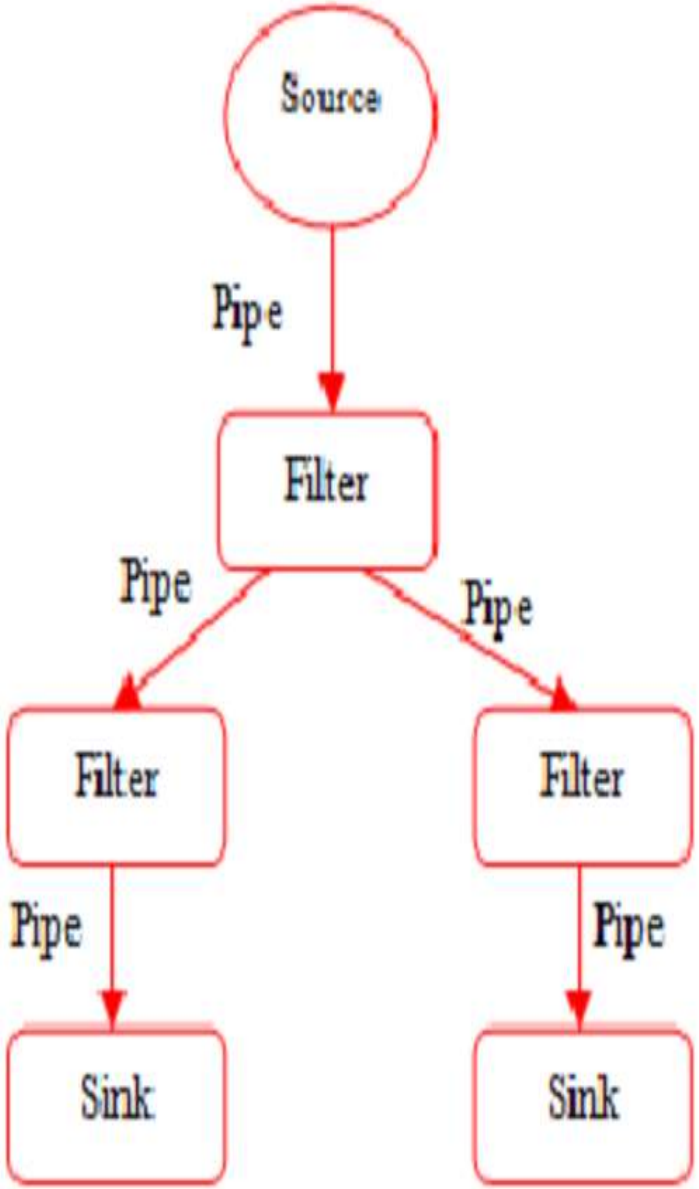
Pipe-Filter Architecture Pattern

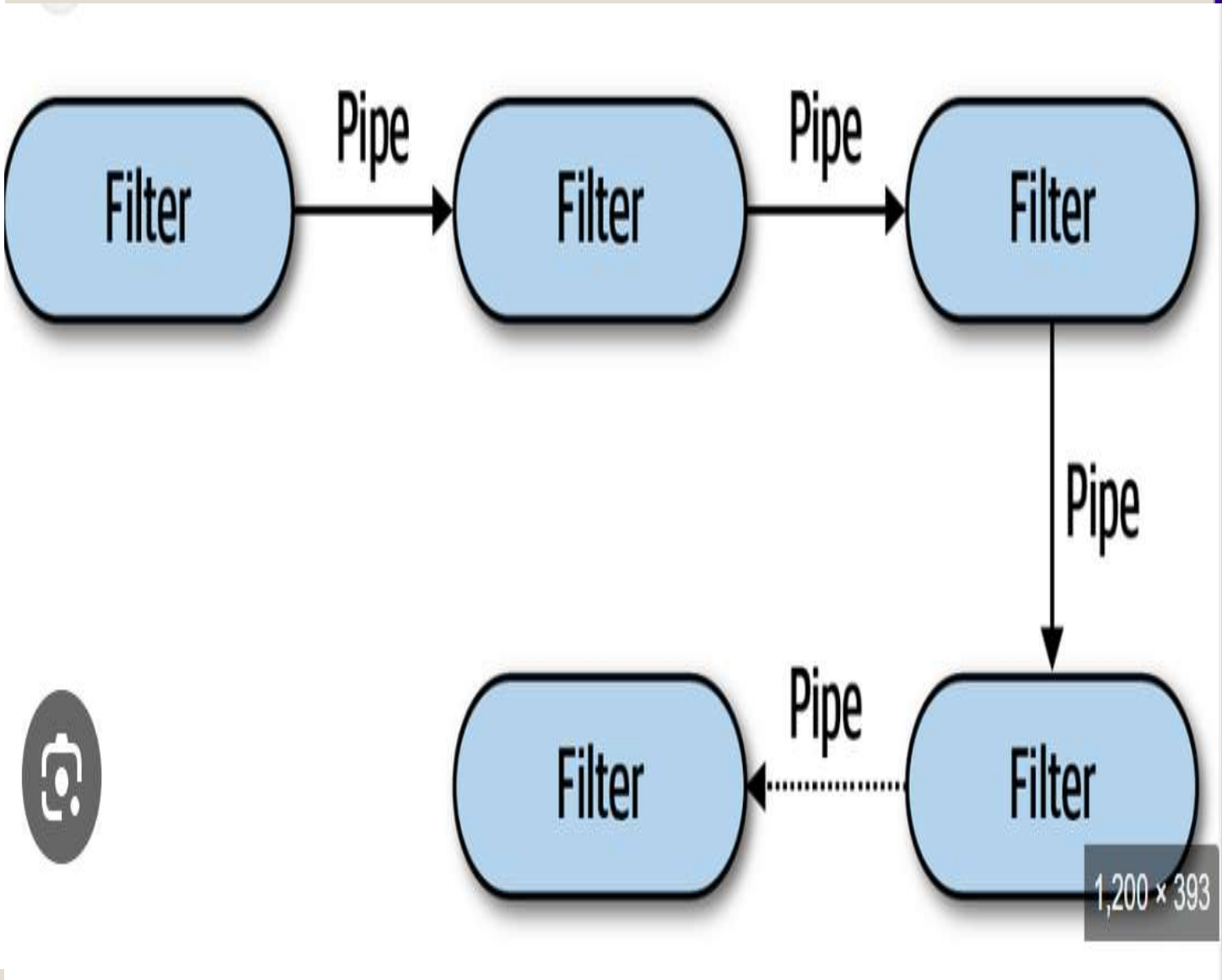
Disadvantages:

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

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

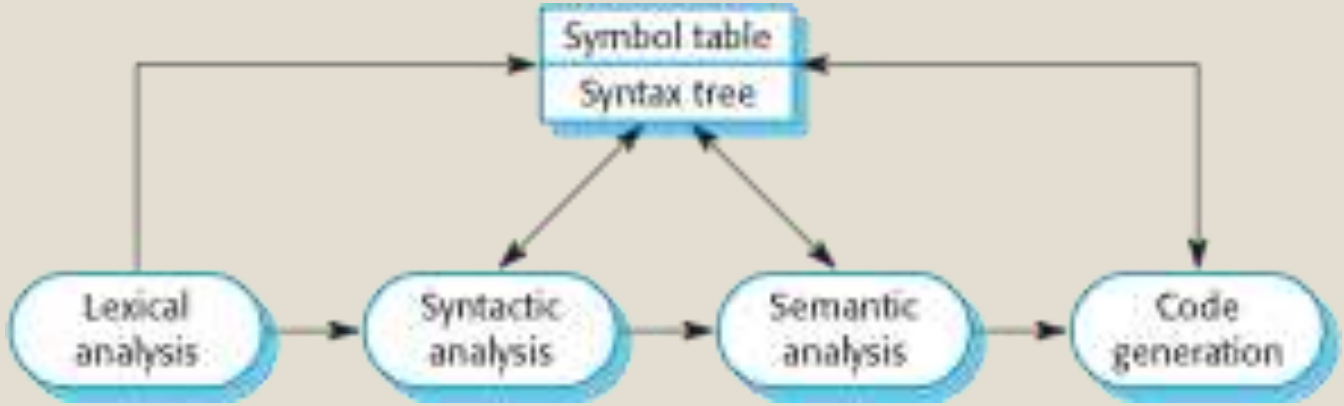
3. **Latency:** Data must be passed through multiple filters, which can introduce latency.


Source
Pipe
Filter
Pipe
Pipe
Filter
Filter
Pipe
Pipe
Sink
Sink


Filter
Pipe
Filter
Pipe
Filter
Pipe
Filter
Pipe
Filter
1,200 × 393

An example of the pipe and filter architecture

A pipe and filter compiler architecture



Repository architecture

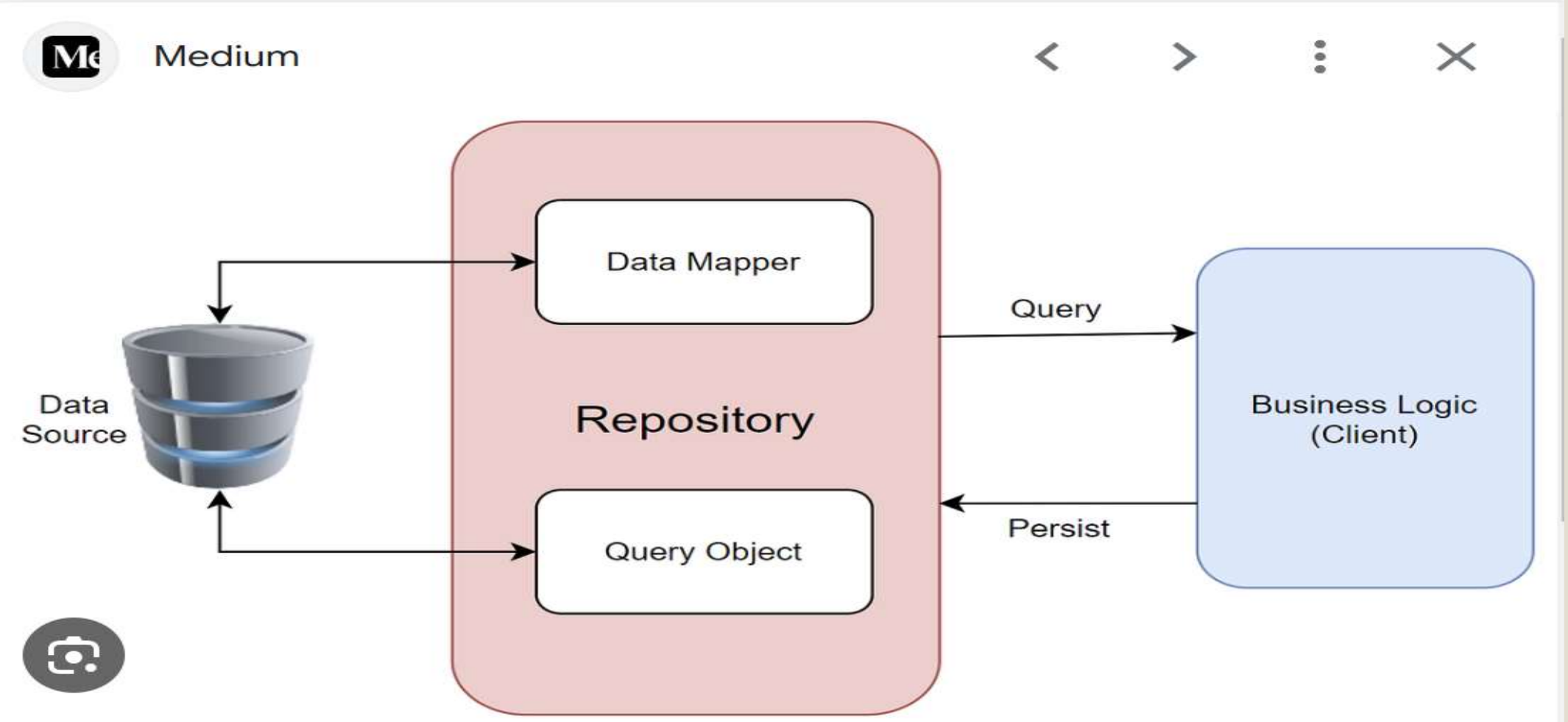
- Sub-systems must exchange data.

This may be done in two ways:

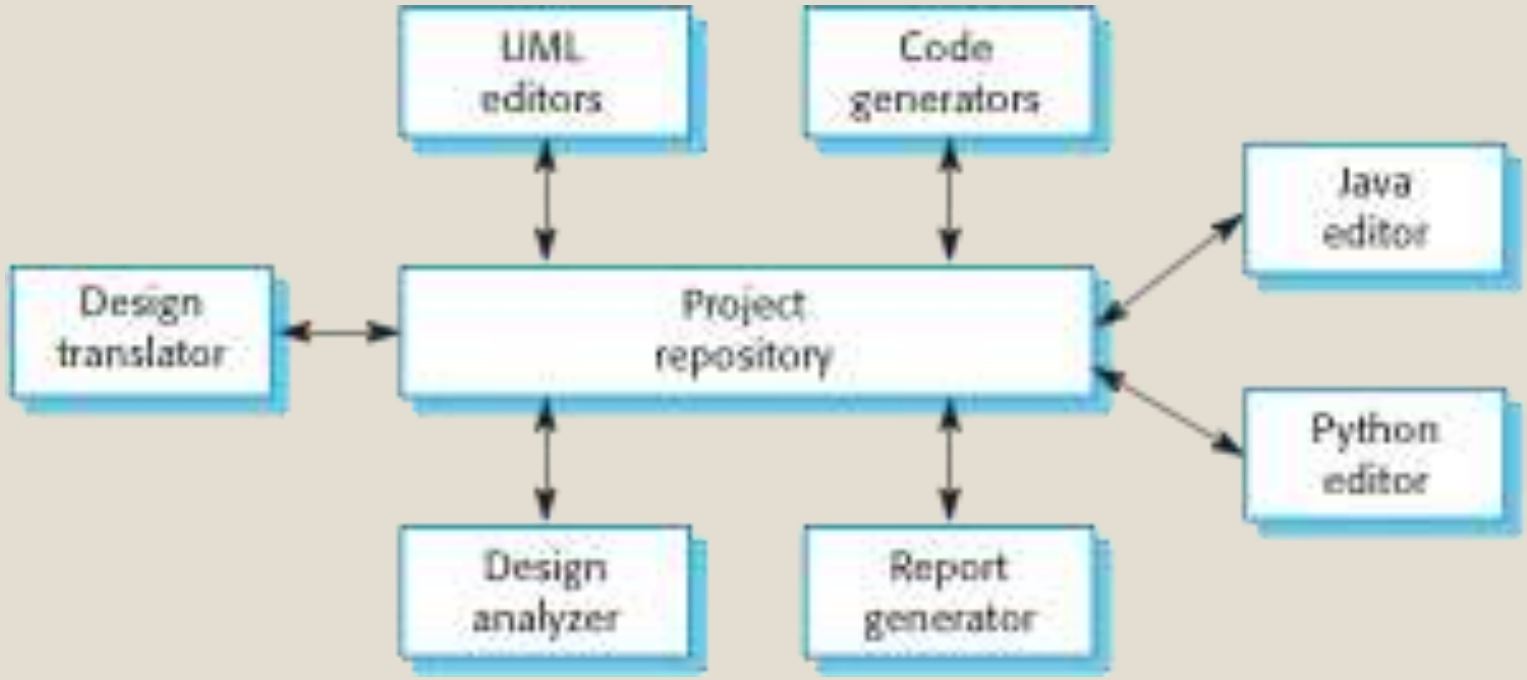
  - Shared data is held in a central database
  - Each sub-system maintains its own database
  - Each sub-system passes data explicitly to other sub-systems

- When large amounts of data are to be shared
- large volumes of information are generated that has to be stored for a long time.
- the repository model of sharing is an efficient data sharing mechanism.
- repository is a single point of failure so problems in the repository affect the whole system

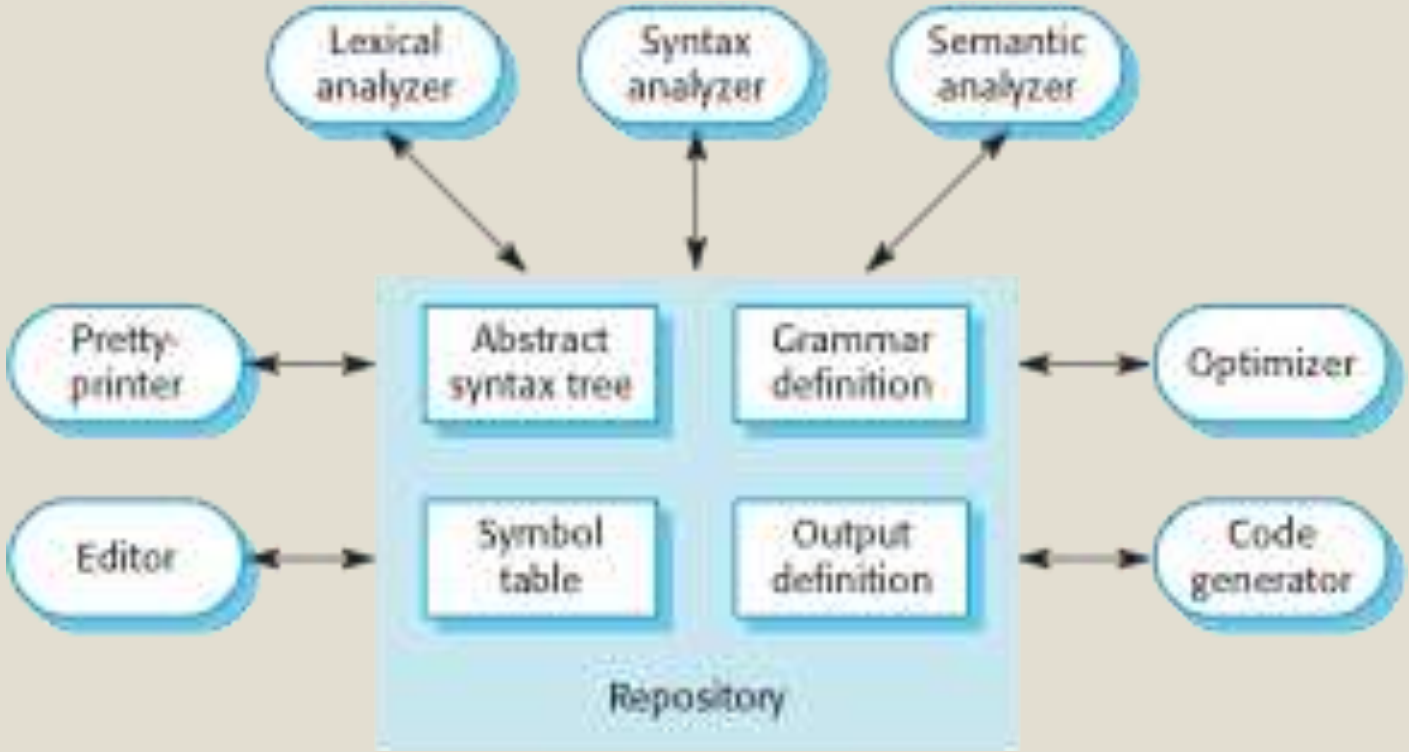
Repository architecture



Repository architecture



A repository architecture for a language processing system



Application architectures

- It is designed to meet an organisational need.
- It is the process of defining a structure that meets all the technical and operational requirements of an application.
- They tend to have a common architecture that reflects the application requirements
- They are configured and adapted to create a system that meets specific requirements

Examples of application types

- Data processing applications
- Transaction processing applications
- Event processing systems
- Language processing systems

Examples of application types

❖**Data processing applications**

❖Data processing applications are tools that help businesses analyze and process data to make informed decisions.

Transaction processing applications

❖A transaction processing application is a collection of transaction programs designed to automate a given business activity.

- **E-commerce systems;**
- **Reservation systems.**

Examples of application types

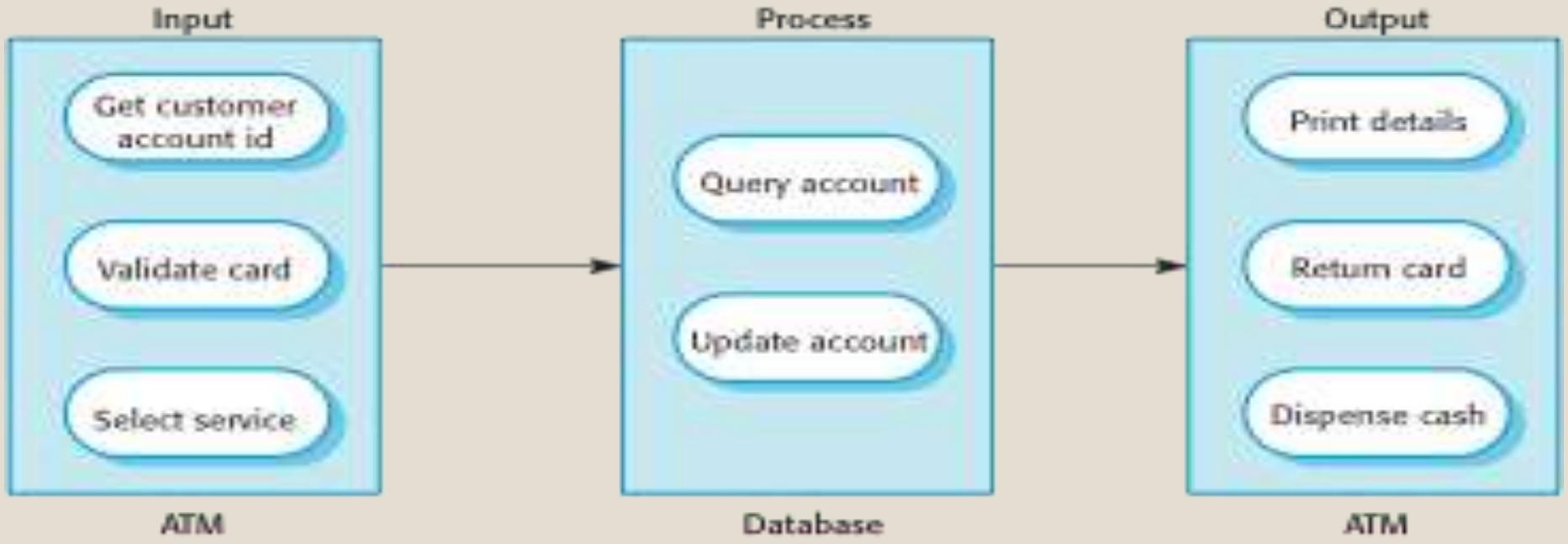
- **Event processing systems**
  - Event processing systems analyze events in real-time and perform actions based on the analysis. Event processing systems can be used to detect patterns, identify trends, and trigger actions

Language processing systems

- It is a subfield of computer science and artificial intelligence (AI) that uses machine learning to enable computers to understand and communicate with human language..
  - **Compilers;**
  - **Command interpreters.**

The structure of transaction processing applications

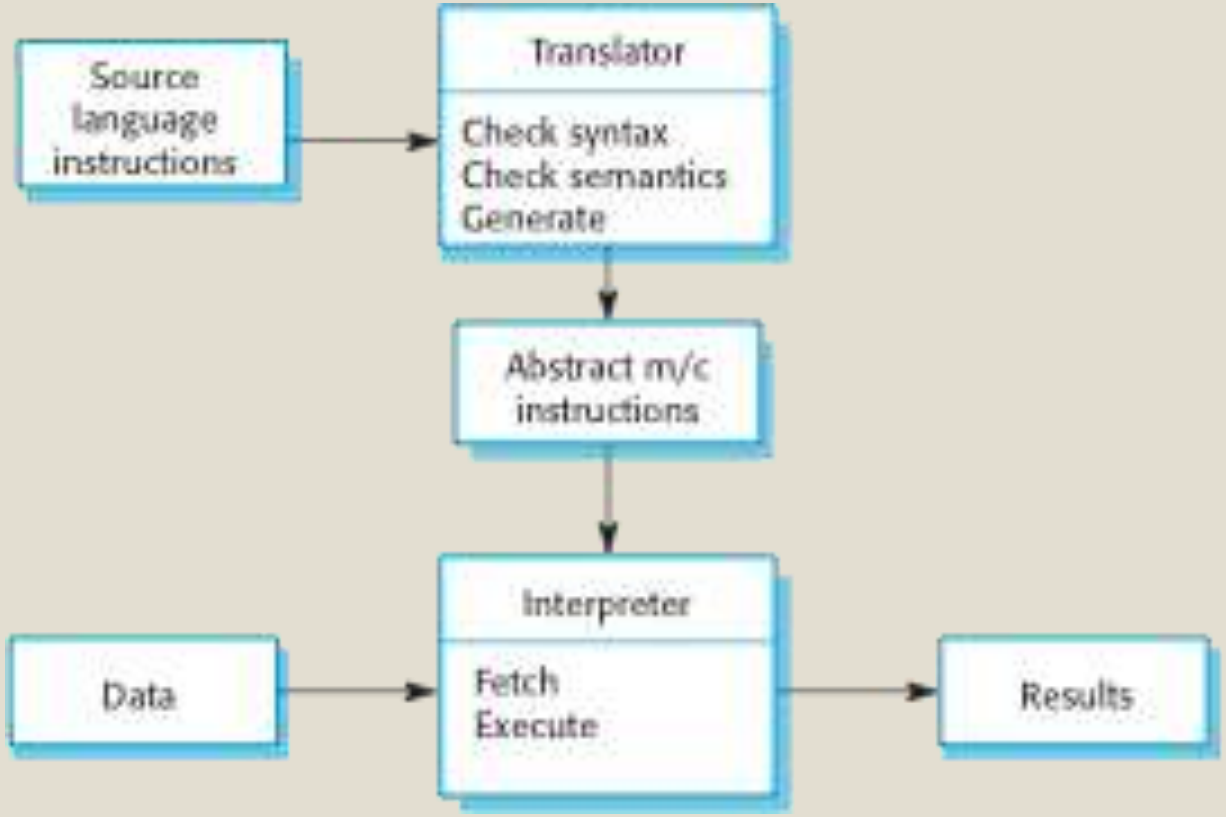
The software architecture of an ATM system transaction processing applications



Language processing systems

- Accept a natural or artificial language as input
- generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.

The architecture of a language processing system

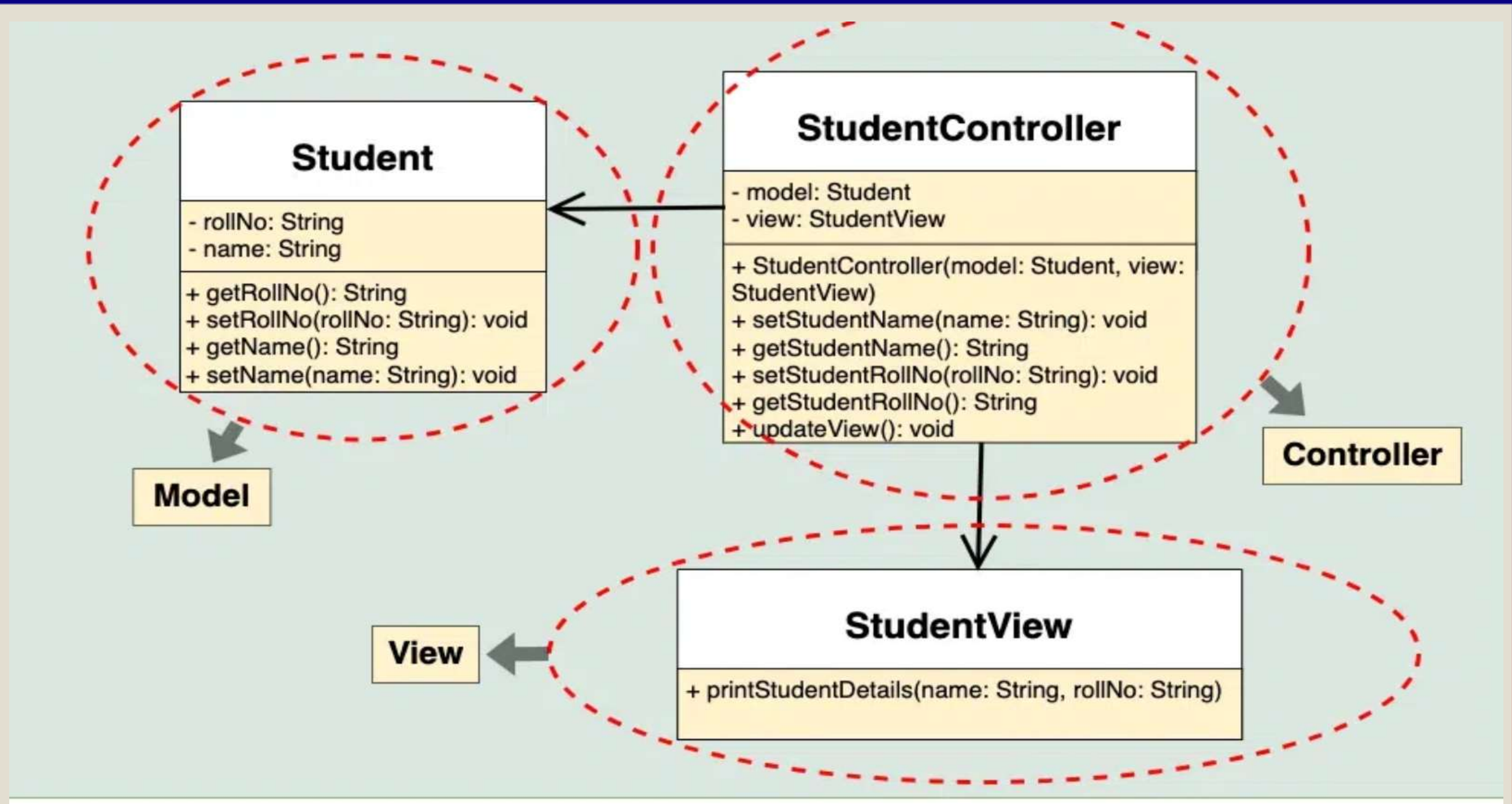


Model/View/Controller

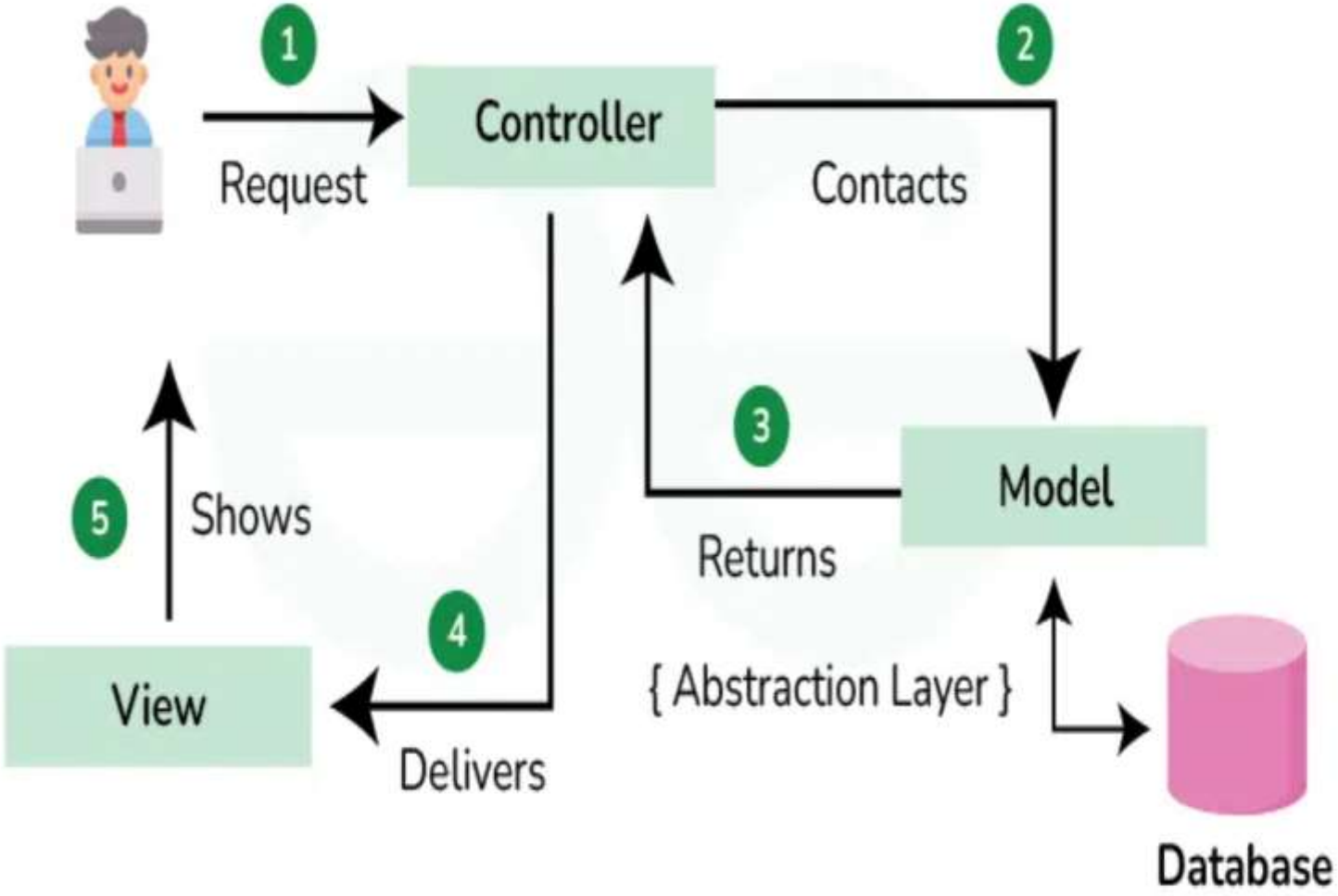
- In the **Model/View/Controller** (MVC) architectural style subsystems are classified into three different types:
- **model** subsystems maintain domain knowledge,
- **View** subsystems display it to the user
- **controller** subsystems manage the sequence of interactions with the user

Model/View/Controller architectural style

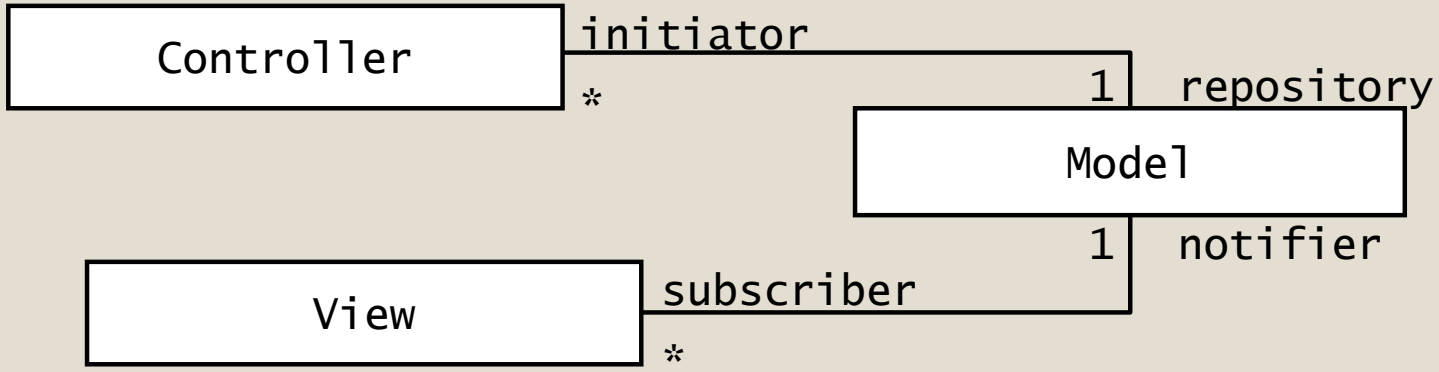
- MVC is a software architecture pattern
- It separates an application into three main components: Model, View, and Controller
- It allows for the reusability of components
- It promotes a more modular approach to software development.


Student
- rollNo: String
- name: String
+ getRollNo(): String
+ setRollNo(rollNo: String): void
+ getName(): String
+ setName(name: String): void
StudentController
- model: Student
- view: StudentView
+ StudentController(model: Student, view: StudentView)
+ setStudentName(name: String): void
+ getStudentName(): String
+ setStudentRollNo(rollNo: String): void
+ getStudentRollNo(): String
+ updateView(): void
StudentView
+ printStudentDetails(name: String, rollNo: String)
Model
Controller
View

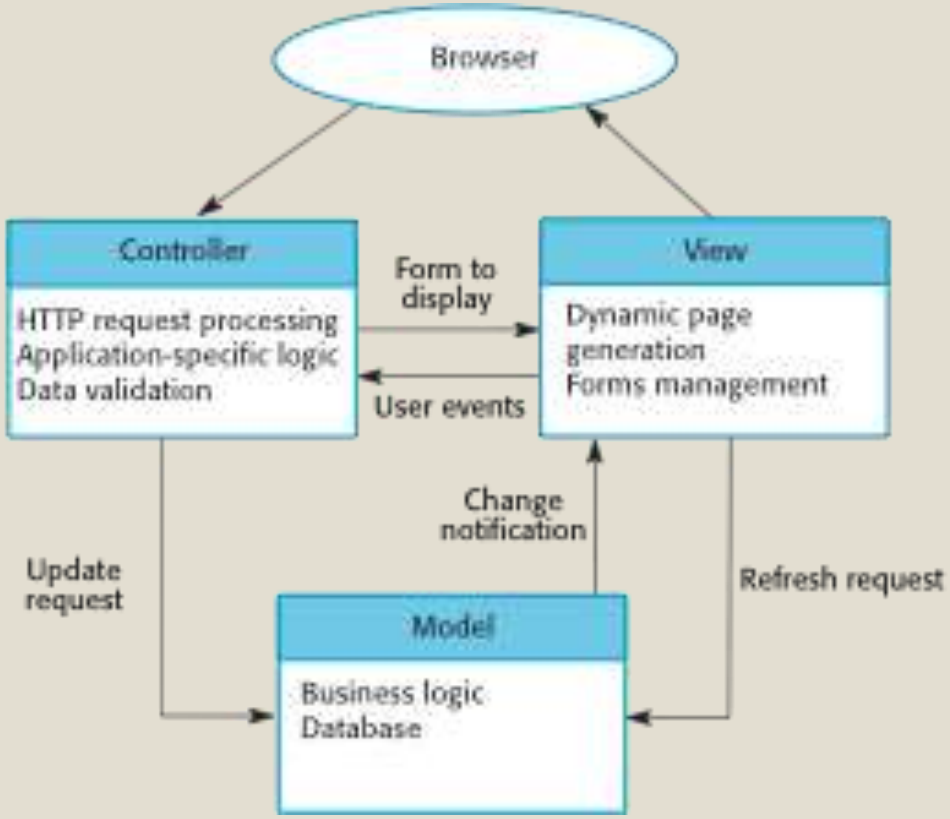
Components of the MVC Design Pattern



Model/View/Controller architectural style.



Web application architecture using the MVC pattern



Top-down Approach

- In software engineering, both top-down and bottom-up approaches are used for designing and developing software systems.
- Each approach has its own advantages and is suitable for different scenarios
- **Top-down Approach**: In the top-down approach, the system is designed by starting with a high-level overview and breaking it down into smaller, more detailed components.

Top-down Approach

Example Scenario: Consider the **development of a website for an e-commerce platform**.

In the top-down approach:

1. Start with defining the main features of the website, such as user authentication, product browsing, and checkout.
2. Break down each feature into sub-features, such as user registration, login, product listing, adding items to the cart, and payment processing.
3. Further break down each sub-feature into smaller components, such as user interface design, database interactions, and business logic.
4. Develop each component iteratively, starting from the high-level components and gradually implementing the lower-level details.

Top-down Approach Advantages

1. Provides a clear roadmap for development.
2. Helps in understanding the system architecture from a high-level perspective.
3. Facilitates early identification of major components and their interactions.

Bottom-up Approach

- **Bottom-up Approach**: In the bottom-up approach, the system is designed by starting with the smallest, most basic components and gradually combining them to form larger, more complex components.
- Example Scenario: Continuing with the e-commerce website example, in the bottom-up approach:
- Start by developing basic reusable components such as form validation, database access functions, and session management.

Bottom-up Approach

- Combine these basic components to build higher-level components such as user authentication, product management, and shopping cart functionality.
- Integrate these components to create the overall website structure, including navigation, layout, and user interface.

Bottom-up Approach

Advantages:

- Encourages reusability of components.
- Facilitates early testing and validation of individual components.
- Allows for incremental development, where new features can be added by integrating existing components.

Differences between Cohesion and Coupling

	Cohesion	Coupling
1	Cohesion is the degree to which the elements inside a module belong together.	Coupling is the degree of interdependence between the modules.
2	A module with high cohesion contains elements that are tightly related to each other and united in their purpose.	Two modules have high coupling (or tight coupling) if they are closely connected and dependent on each other.
3	A module is said to have low cohesion if it contains unrelated elements.	Modules with low coupling among them work mostly independently of each other.
4	Highly cohesive modules reflect higher quality of software design	Loose coupling reflects the higher quality of software design

Cohesion

- *cohesion* implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself!
- Levels of cohesion!
- Functional!
- Layer!
- Communicational!

Cohesion

1.Functional Cohesion:

1. Functional cohesion occurs when elements of a module (such as functions or methods) are grouped together because they all contribute to a single, well-defined task or functionality.
2. Example: A module that calculates the area of a circle, computes its circumference, and checks if a point lies within the circle exhibits functional cohesion.

2.Sequential Cohesion:

1. Sequential cohesion occurs when elements of a module are grouped together because they are executed in a specific sequence, with the output of one element serving as the input to the next.
2. Example: A module that reads data from a file, processes it, and then writes the results to another file exhibits sequential cohesion.

Cohesion

3. Communicational Cohesion:

1. Communicational cohesion occurs when elements of a module are grouped together because they operate on the same data or share the same input/output parameters.
2. Example: A module that performs various operations (such as validation, formatting, and storage) on a user's personal information exhibits communicational cohesion.

4. Procedural Cohesion:

1. Procedural cohesion occurs when elements of a module are grouped together because they are all part of a specific procedural or algorithmic sequence.
2. Example: A module that implements a sorting algorithm, including steps for comparing elements, swapping positions, and iterating through the array, exhibits procedural cohesion.

Cohesion

5. Temporal Cohesion:

1. Temporal cohesion occurs when elements of a module are grouped together because they are all executed at the same time, such as during a specific phase or event.
2. Example: A module that initializes system resources, performs setup operations, and starts background tasks during system startup exhibits temporal cohesion.

6. Coincidental Cohesion:

1. Coincidental cohesion occurs when elements of a module are grouped together arbitrarily, without any logical relationship between them.
2. Example: A utility module that contains unrelated functions for performing various unrelated tasks, such as file I/O, string manipulation, and mathematical calculations, exhibits coincidental cohesion.

Coupling

Coupling is a qualitative measure of the degree to which classes are connected to one another.

As classes (and components) become more interdependent, coupling increases.

An important objective in component-level design is to keep coupling as low as is possible.

Content Coupling:

1. Definition: Occurs when one component directly modifies data that belongs to another component

   Example: A function within Module A directly modifies a variable that is supposed to be private to Module B.

Coupling

Common Coupling:

1. Definition: Multiple components rely on a global variable, potentially leading to unintended consequences when changes are made.
2. Example: Several functions in different modules all use a global configuration variable to determine behavior.

Data Coupling:

1. Definition: Operations passing large amounts of data as arguments, increasing interface complexity and making testing and maintenance more difficult.
2. Example: A function passes a complex data structure (e.g., an array or object) to another function for processing.

Routine Call Coupling:

1. Definition: One operation invokes another, increasing system connectedness, which is common but can contribute to complexity.
2. Example: Function A calls Function B to perform a specific task, relying on its functionality to achieve a larger goal.

Inclusion or Import Coupling:

1. Definition: A component imports content from another component, potentially leading to dependencies between them.
2. Example: Module A includes the header file of Module B, creating a dependency on its contents.

External Coupling:

1. Definition: A component communicates or collaborates with external infrastructure components, which should be limited to a small number within a system.
2. Example: An application interacts with the operating system's file system functions to read and write files.

PERFORMING USER INTERFACE DESIGN

Introduction

- Software blue print is done -→ architecture is ready
- But it is **not sufficient** we have to do interface design

Three areas of concern: Design of interfaces Between

- software components
- software components & external entities
- Human computer interaction

Introduction

Definition of interface: that creates effective communication between human & system

Why it is required: software is difficult

Steps in interface design

- ❖Identify user tasks
- ❖User scenarios are created
- ❖Analyse scenarios
- ❖Define set of user interface objects & actions

Introduction

- Software blue print is done -→ architecture is ready
- But it is **not sufficient** we have to do interface design

Three areas of concern: Design of interfaces Between

1. software components
2. software components & external entities
3. Human computer interaction

Introduction

Definition of interface: that creates effective communication between human & system

Why it is required: software is difficult

Steps in interface design

- Identify user tasks
- User scenarios are created
- Analyse scenarios
- Define set of user interface objects & actions

Steps In Interface Design

- Identify User,task And Environmental Techniques
- Based On User Tasks User Scenario Are Prepared
- User Scenario Help To Design User Interface And Its Corresponding Action
- This Set Of Actions And Objects Help In Deciding Screen Layout
- Once Layout Is Prepared Appropriate Icons,screen Texts And Menu Items Are Placed At Respective Position

Golden Rules

- Theo Mandel Proposed 3 Golden Rules For good User Interface Design
- **Place The User In Control(6)**
- **Reduce Users Memory Load (5)**
- **Make Interface Consistent (3)**

GOLDEN RULES

- **Place The User In Control**
- While Analysing Any Requirements During Requirement Analysis The **User Often Demands For System Which Will Satisfy User Requirements And Help Him To Get Things Done**

GOLDEN RULES

- Design Principles That Allow User To Control System
- **Define Interaction Modes In Such A Way That User Will Be Restricted From Doing Unnecessary Actions**
- Interaction Mode Means The Current State In Which User Is Working And Being In Such Mode User Is Suppposed To Do the Related Task Only If User Has To Perform Unnecessary Actions At Such Time Gui Becomes Frustating

The Interaction Should Be Flexible

- User Interactions Should Be Flexible
- **Example:** micrososft Power Point Slide Show The Slide Transition Is Possible Using Mouse Clicks As Well As By Using Keyboard. Such Flexibility Allows Any User To Operate The System As Per His Comfort.

Provide The Facility Of Undo Or Interruption In User Interaction

- This Is The Feature That Allows User To Correct Himself Whenever Necessary Without Loosing His Previous Work..
- Example: in Software Like Paint One Can Draw Some Objects And Perform Redo /Undo Actions For Performing His Desired Drawing

Golden Rules

Allow User To Customize Interaction:

- It Is Observed That In While Handling User Interface Certain Actions Need To Be Done Repeatedly. It Saves Time If These Actions Are Collected In Macros

Hide Technical Details From User

- User Should Not Be Aware Of System Commands, operating System Functions, file Management Functions. Example: While Printing A Document If User Gives **Command For Printing**. Instead Of That If We Have Symbol Print Then It Is Very Useful For Casual User

Golden Rules

The Objects Appearing On Screen Should Be Interactive User Should Be In Position To Adjust The Object Appearing On Screen.Example:in Paint One Should Be Able To Stretch The Oval /Edit The Text Written In The Object.

Reduce The Users Memory Load:

- **If User Interface Is Good Then User Should Has To Remember Very Less.**
- Design Should Be In Such That System Remembers More For User And Ultimately Assist The User To Handle Computer Based System.

Reduce The Users Memory Load:

- **Do Not Force The User To Have long Term Memory:**
- When User Is Involved In Complex Tasks Then He Has To Remember More. The User Interface Should Be Such That He Need Not Remember Past Actions And Result. And This Is Achieved By Providing Proper Visual Interface

Reduce The Users Memory Load:

Establish Meaningful Defaults

- Meaningful Defaults Option Should Be Available To User.

  Example::in Ms Word User Can Set Default Font Size As Per His Choice. There Should Be Some Reset Options Available To User In order To Get Back To Original Position

Use Institutive Shortcuts

- Example::ctrl+s Key =Saving File;ctrl+o=opening File. Hence These To Be Designed.
- (D)**the Visual Layout Of The Interface Should Be Realistic**
- When **Certain Aspect/Feature Of System Needs To Be Highlighted** Then Use Of Proper Visual Layout Is Helpful To Casual User To Handle System.Example: In Online Purchase System If Pictures Of Master Card/Visa Are Given It Guides The User To Understand The Payment Mode Of Purchasing.

Use Institutive Shortcuts

- **Disclose The Information Gradually: there** Should Not Be **Bombarding Of Info** On User. It Should Be Presented To User In Systematic Way. Example::one Can Organize Info In Hierarchical Manner, top Down/Bottom Up Way.

Make The Interface Consistent

- The User Interface Should Be Consistent.
- All visual information should be organized
- Task to task navigating mechanisms should be defined properly

The Consistency Can Be Maintained At Three Levels Such As::

1 allow user to put current task in meaning full way

2. maintain consistency among family of applications

3. if certain standards are maintained in previous mode do not change it untill necessary

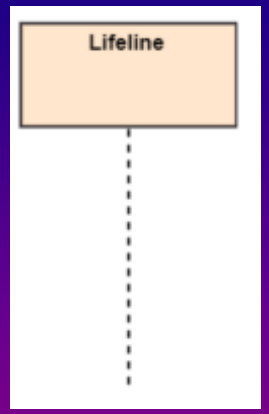
Interaction diagram

- capture dynamic behavior of a system.
- To describe the message flow in the system.
- To describe structural organization of the objects.
- To describe interaction among objects.

Sequence Diagram

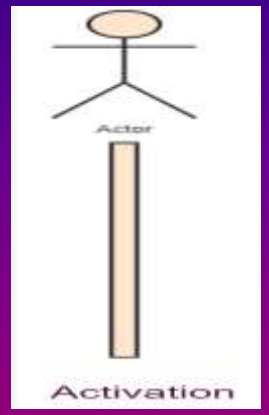
- sequence diagram is the most commonly used interaction diagram. Interaction diagram – It is used to show the interactive behavior of a system
- The sequence diagram represents the flow of messages in the system.
- It portrays time-ordered sequence of events.
- It models high-level interaction among active objects within a system.
- lifeline is represented by a vertical bar
- message flow is represented by a vertical dotted line

Sequence Diagram



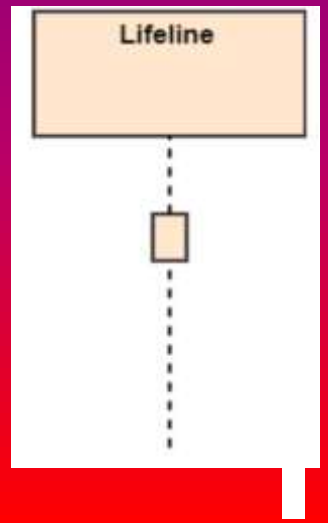
Lifeline

An individual participant in the sequence diagram is represented by a lifeline.



Actor

It represents the role, which involves human users and external hardware or subjects.



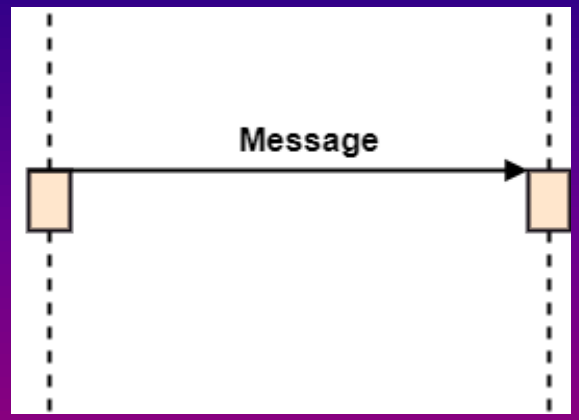
Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element

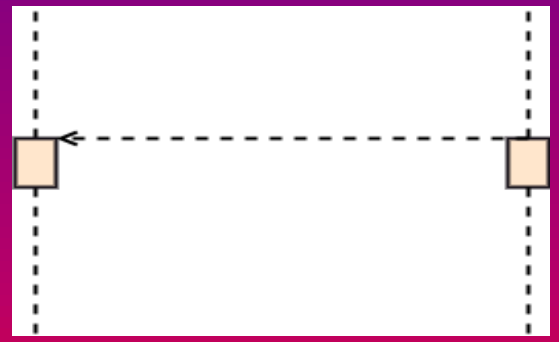
Messages

The messages depict the interaction between the objects and are represented by arrows.

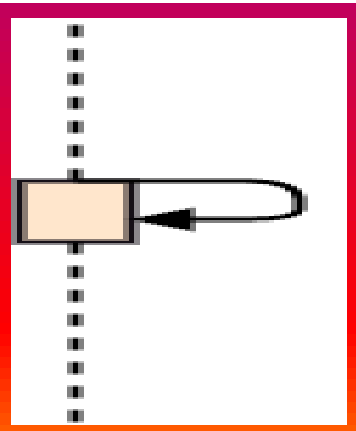
Sequence Diagram



•**Call Message:** represents that the target lifeline has invoked an operation.

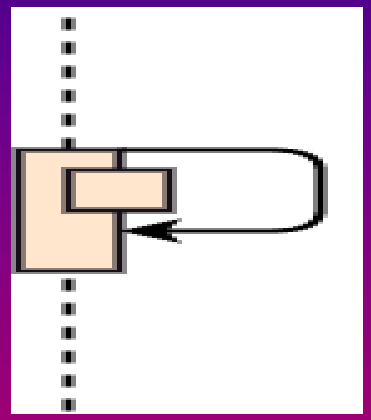


•**Return Message:** represent the flow of information from the receiver of the corresponding caller message.

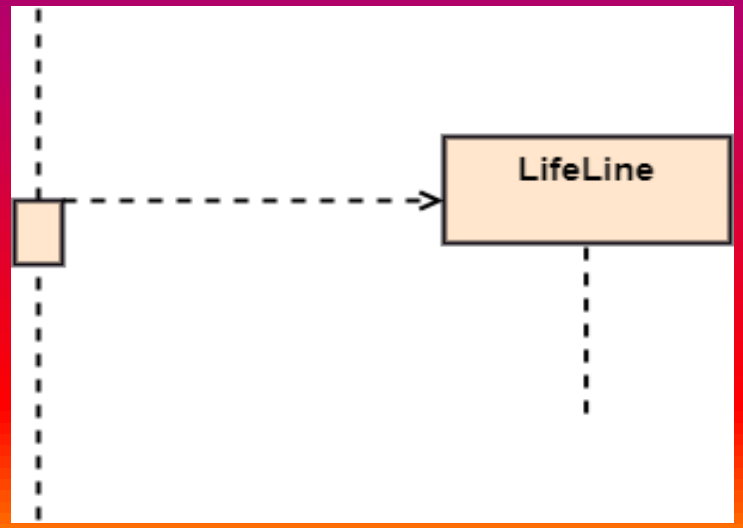


•**Self Message:** a message of the same lifeline, has been invoked.

SEQUENCE DIAGRAM



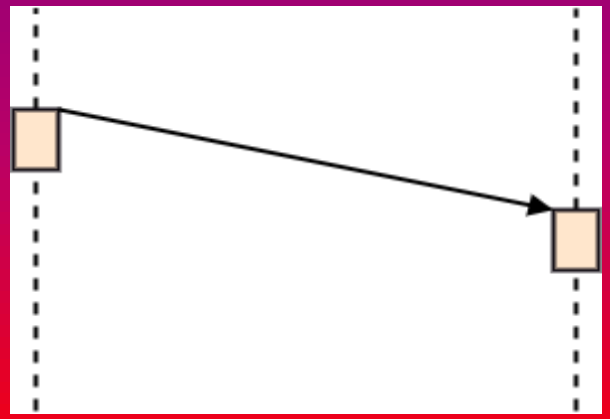
•**Recursive Message:** It is a special case of the self message as it represents the recursive calls.



•**Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.

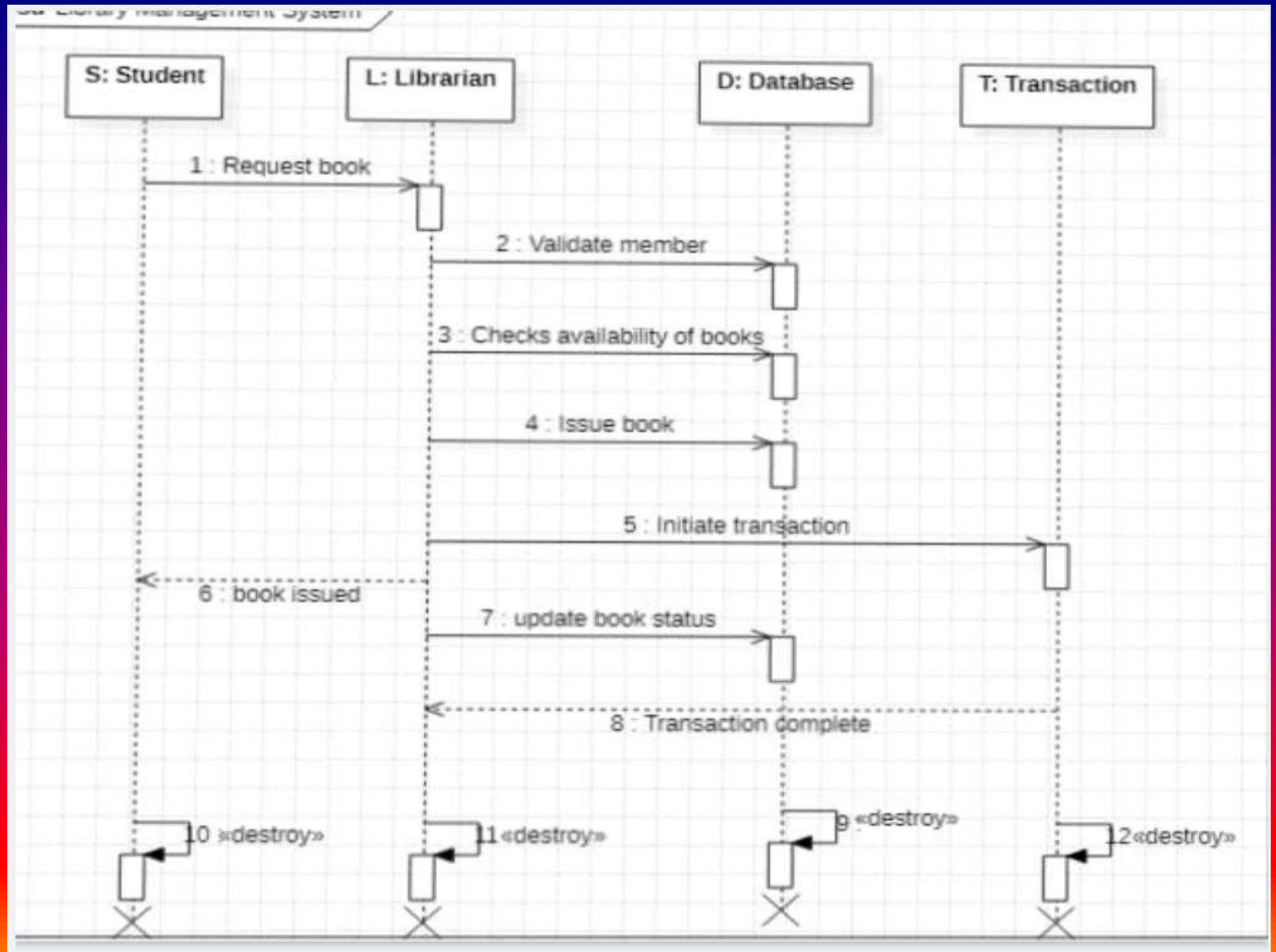
SEQUENCE DIAGRAM

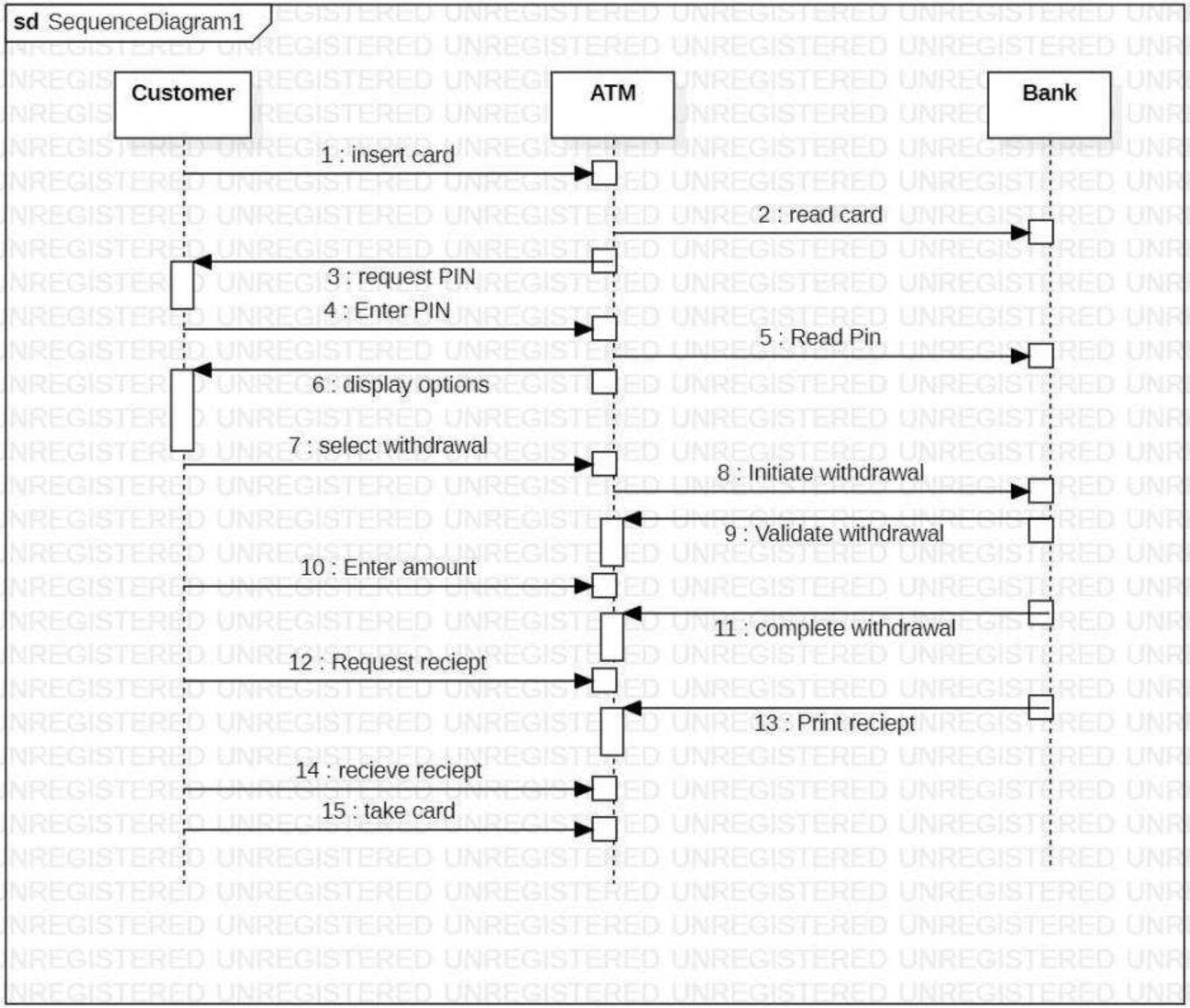
•**Destroy Message:** depicts a request to destroy the lifecycle of the target.



•**Duration Message:** portrays the time passage of the message while modeling a system.

Note
A note is the capability of attaching several remarks to the element.


S: Student
L: Librarian
D: Database
T: Transaction
1 : Request book
2 : Validate member
3 : Checks availability of books
4 : Issue book
5 : Initiate transaction
6 : book issued
7 : update book status
8 : Transaction complete
10 «destroy»
11 «destroy»
«destroy»
12 «destroy»


sd SequenceDiagram1
Customer
ATM
Bank
1 : insert card
2 : read card
3 : request PIN
4 : Enter PIN
5 : Read Pin
6 : display options
7 : select withdrawal
8 : Initiate withdrawal
9 : Validate withdrawal
10 : Enter amount
11 : complete withdrawal
12 : Request reciept
13 : Print reciept
14 : recieve reciept
15 : take card

Collaboration Diagram

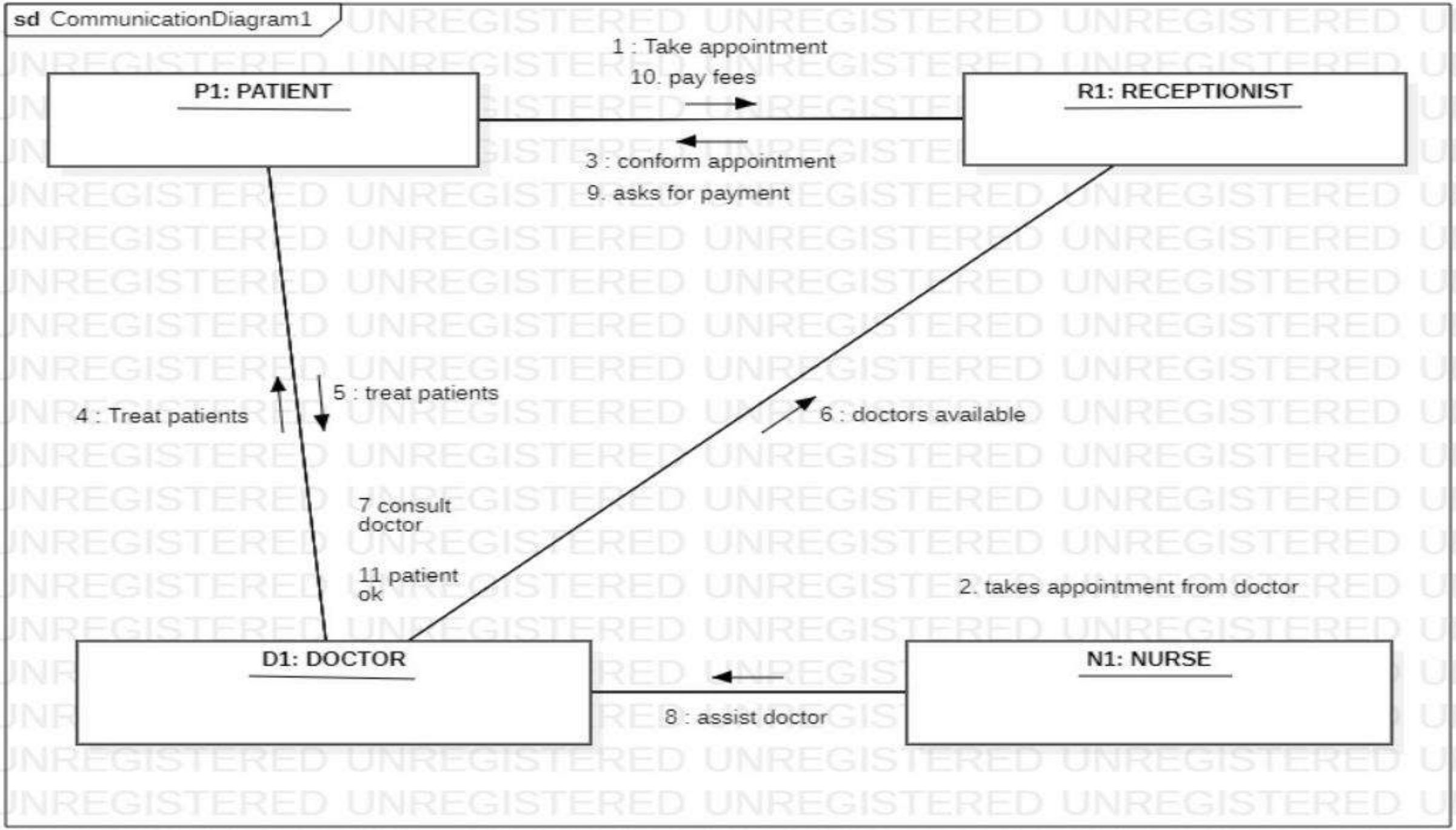
- The collaboration diagram is used to show the relationship between the objects in a system.
- sequence and the collaboration diagrams represent the same information but differently.
- it depicts the architecture of the object residing in the system
- it is essential to depict the relationship between the object.

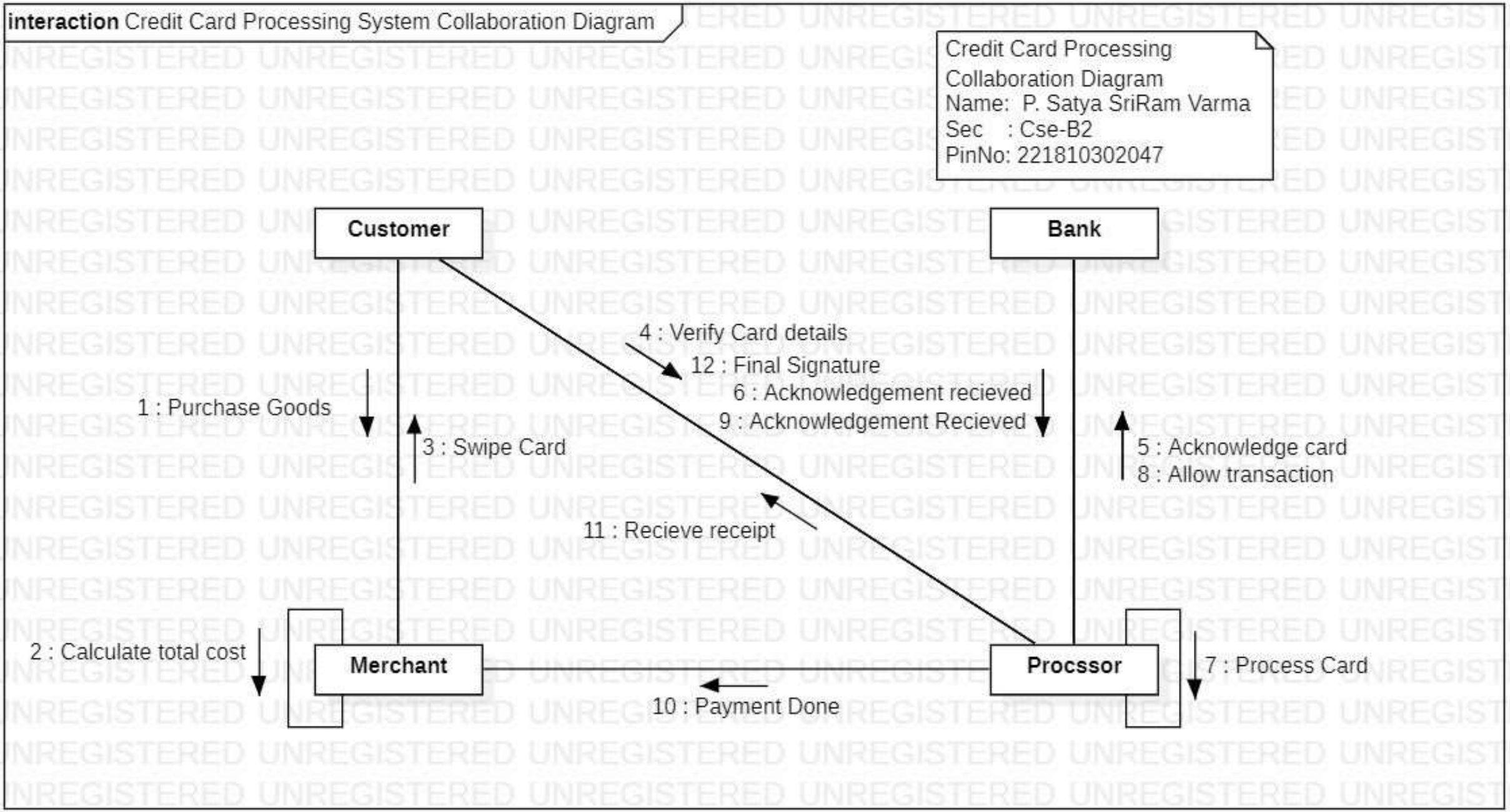
Collaboration Diagram

- Objects: The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.
- Actors: the actor plays the main role as it invokes the interaction. Each actor has its respective role and name.
- Links: It portrays a relationship between the objects through which the messages are sent.
- Messages: It is a communication between objects which carries information and includes a sequence number

Collaboration diagram

- It is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.
- The purpose of collaboration diagram is similar to sequence diagram. But the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.


sd CommunicationDiagram1
P1: PATIENT
R1: RECEPTIONIST
D1: DOCTOR
N1: NURSE
1 : Take appointment
10. pay fees
3 : conform appointment
9. asks for payment
4 : Treat patients
5 : treat patients
6 : doctors available
7 consult doctor
11 patient ok
2. takes appointment from doctor
8 : assist doctor


interaction Credit Card Processing System Collaboration Diagram
Credit Card Processing
Collaboration Diagram
Name: P. Satya SriRam Varma
Sec : Cse-B2
PinNo: 221810302047
Customer
Bank
4 : Verify Card details
12 : Final Signature
6 : Acknowledgement recieved
9 : Acknowledgement Recieved
1 : Purchase Goods
3 : Swipe Card
5 : Acknowledge card
8 : Allow transaction
11 : Recieve receipt
2 : Calculate total cost
Merchant
Procssor
7 : Process Card
10 : Payment Done

State chart diagram

- Any real time system reacts to internal/external events.
- These events are responsible for state change of the system.
- It is used to represent the event driven state change of a system.
- It visualize the reaction of a system by internal/external factors.
- It describes the flow of control from one state to another state.
- States are defined as a condition in which an object exists
- It changes when some event is triggered.
- It is used to model life time of an object from creation to termination.

State chart diagrams

The main purposes of using State chart diagrams:

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

STATE CHART DIAGRAM

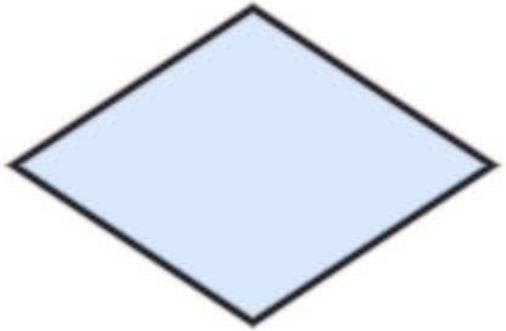
- state machine diagram is also called the Statechart or State Transition diagram
- It captures the software system's behavior
- It models event-based systems to handle the state of an object.
- defines several distinct states of a component within the system
- It records the dynamic view of a system

Notation of a State Machine Diagram

Following are the notations of a state machine diagram enlisted below:

Initial state

State-box



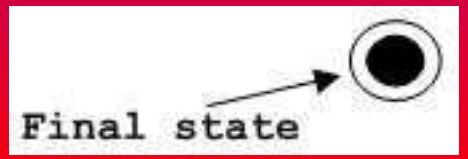
Decision-box

Final State

Structural Things – Initial state and Final state notation

- Initial state is defined show the start of a process. This notation is used in almost all diagrams.

- Final state is used to show the end of a process. This notation is also used in almost all diagrams to describe the end.

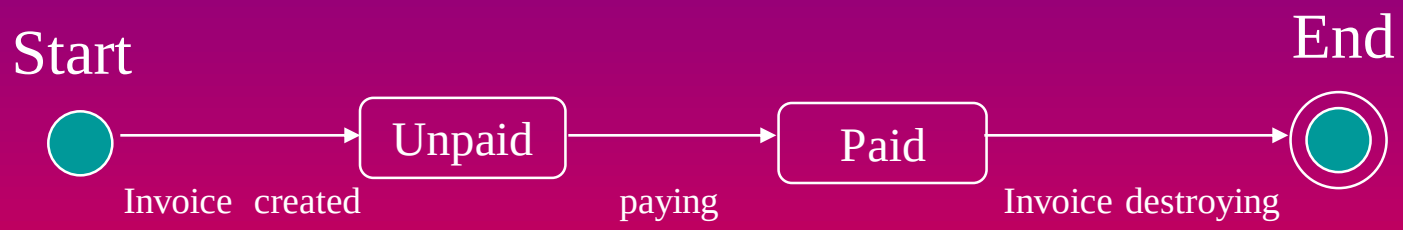


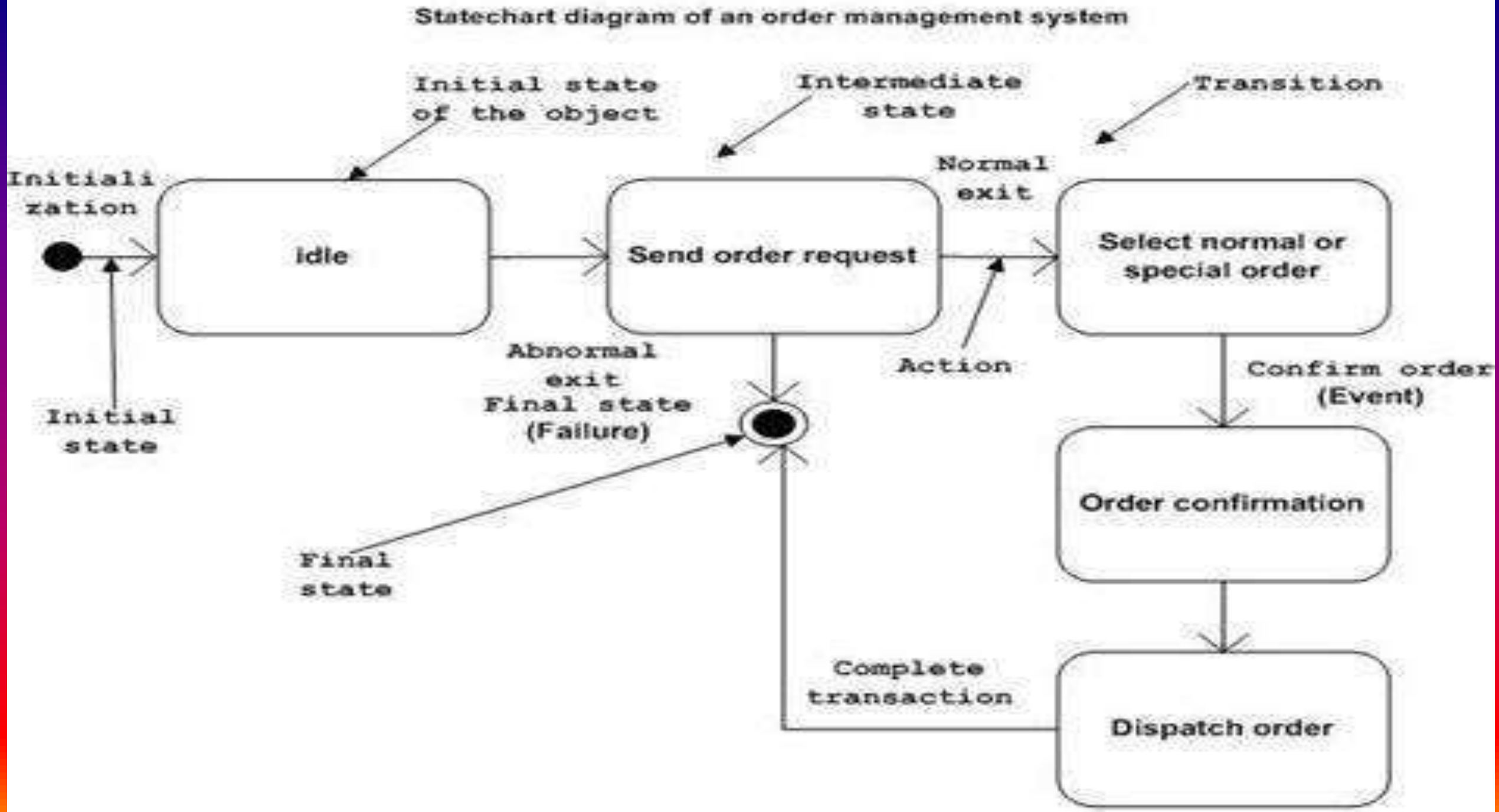
STATE CHART DIAGRAM

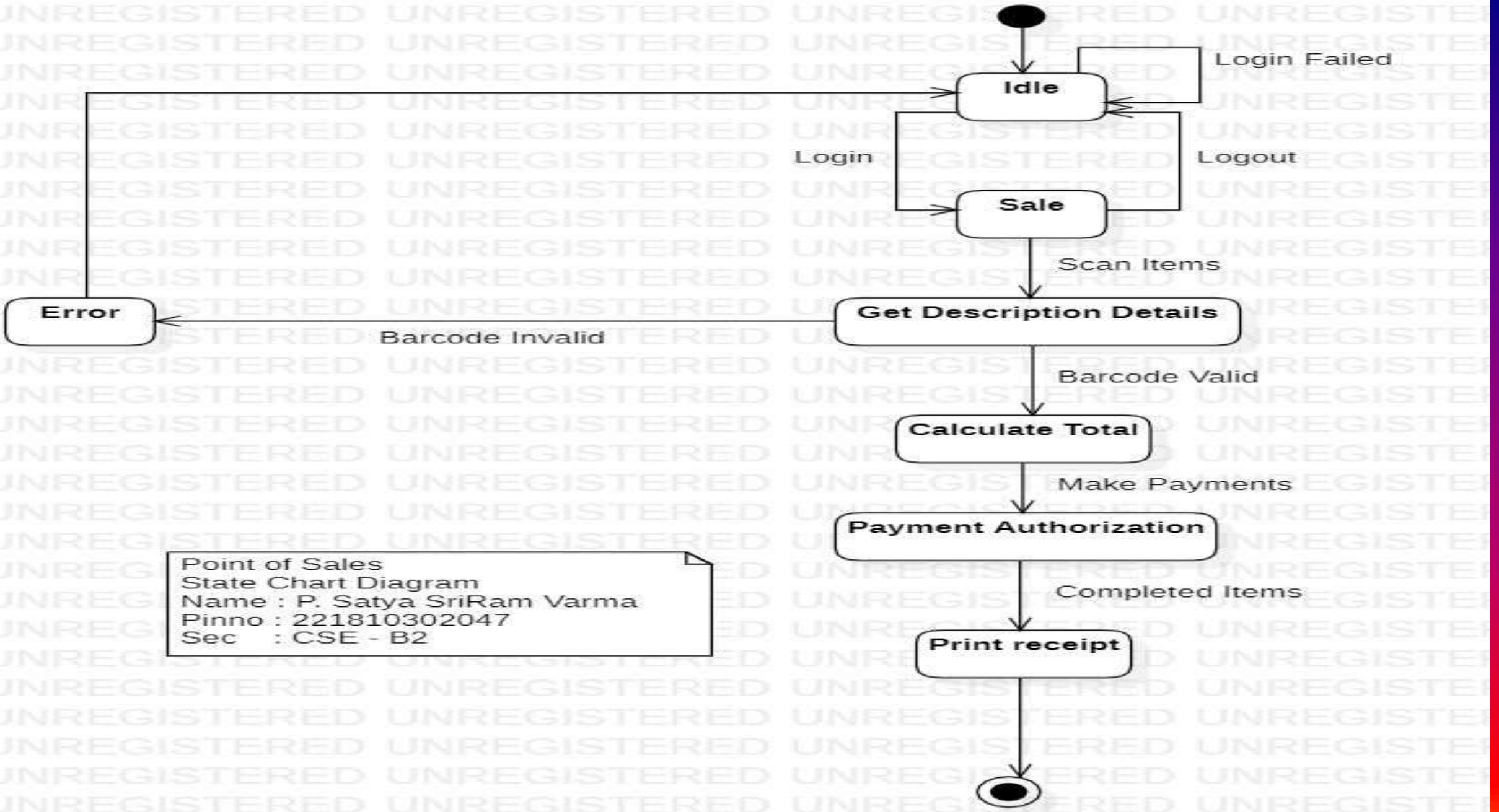
1. **Initial state:** It defines (beginning) of a system
2. **Final state:** It represents (end) of a system.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time.

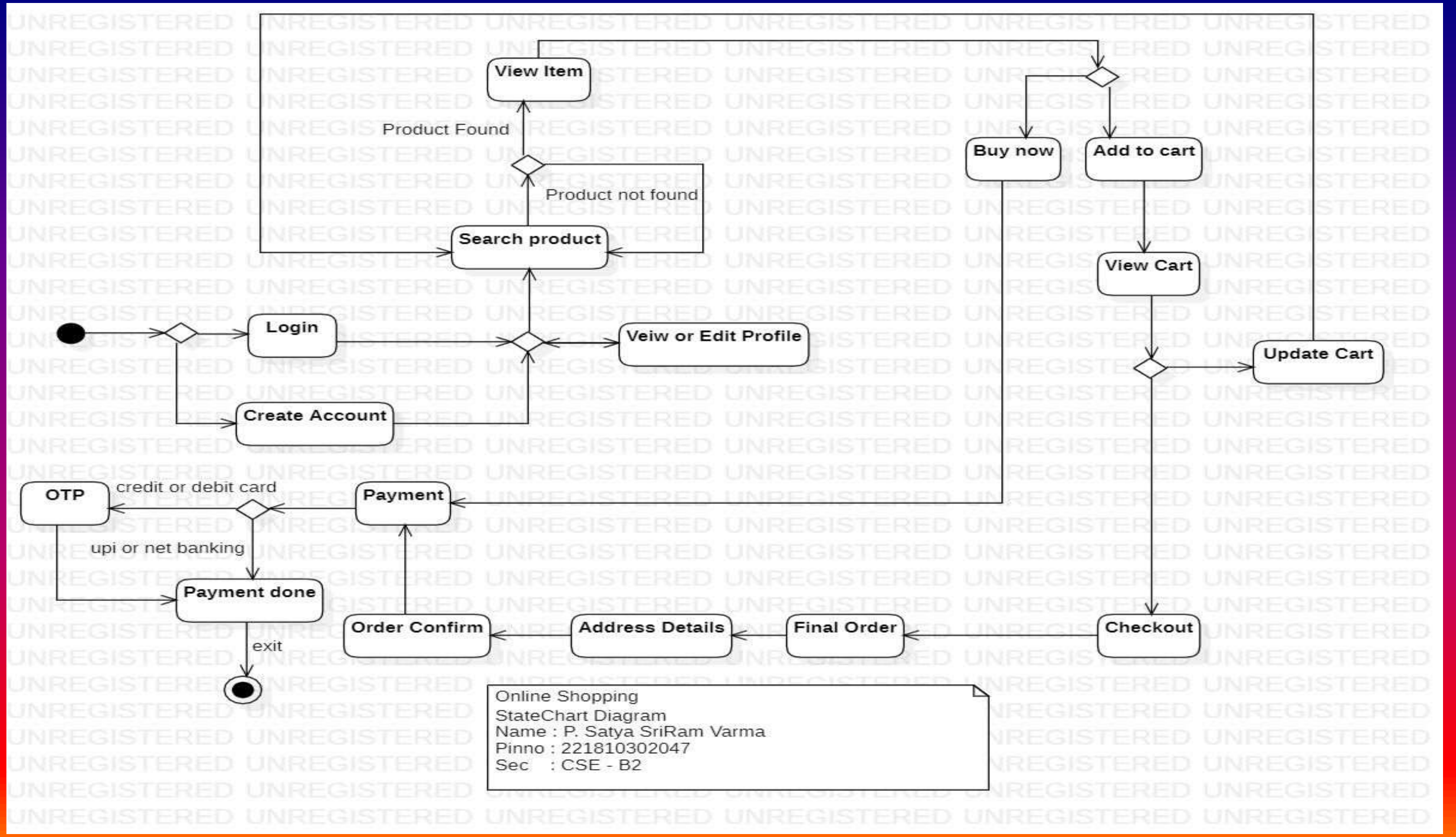
State Diagrams
(Billing Example)

State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



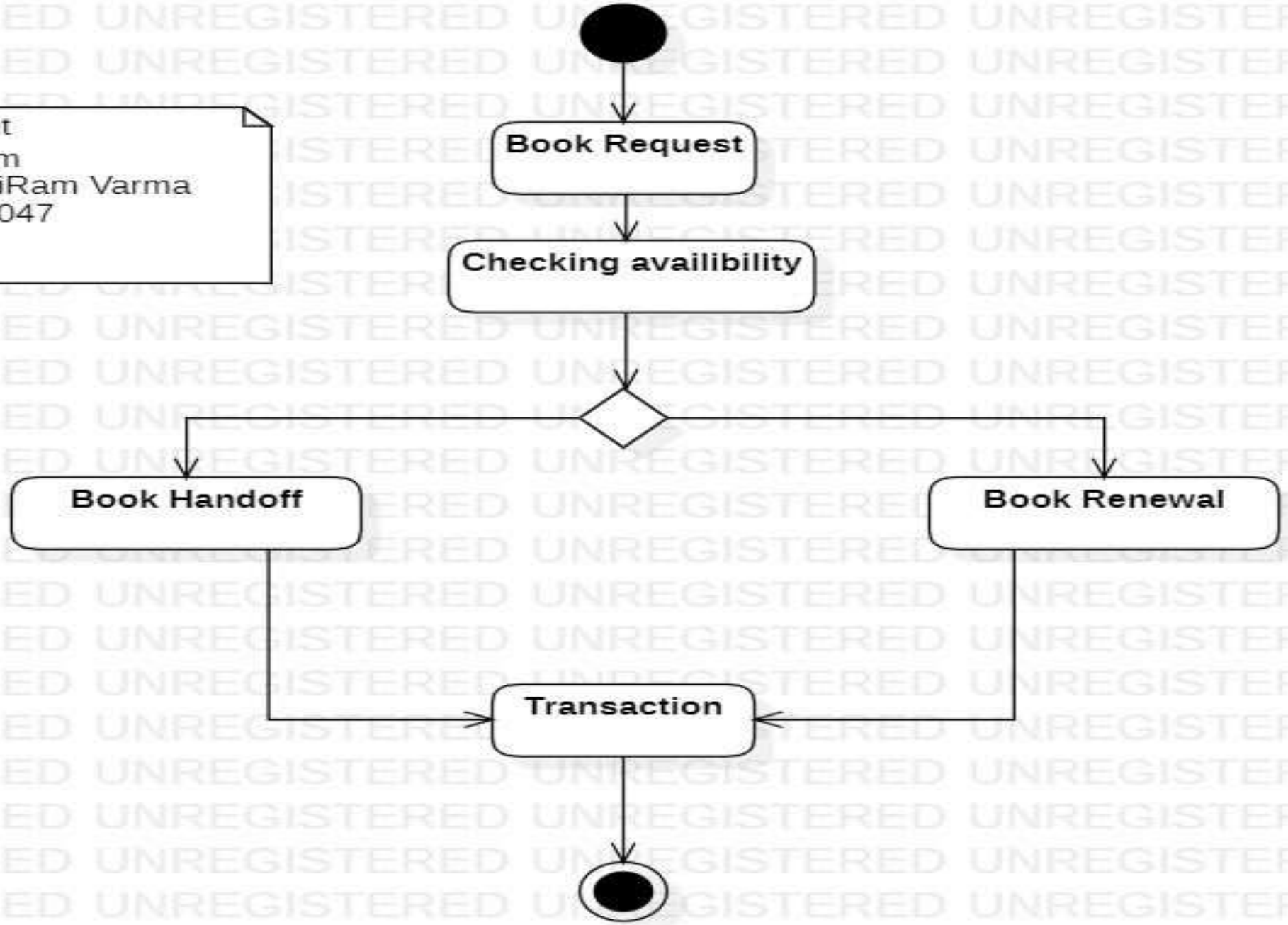

Statechart diagram of an order management system
Initial state of the object
Intermediate state
Transition
Initiali zation
Normal exit
idle
Send order request
Select normal or special order
Initial state
Abnormal exit Final state (Failure)
Action
Confirm order (Event)
Order confirmation
Final state
Complete transaction
Dispatch order


Idle
Login Failed
Login
Logout
Sale
Scan Items
Error
Get Description Details
Barcode Invalid
Barcode Valid
Calculate Total
Make Payments
Payment Authorization
Completed Items
Print receipt
Point of Sales
State Chart Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2

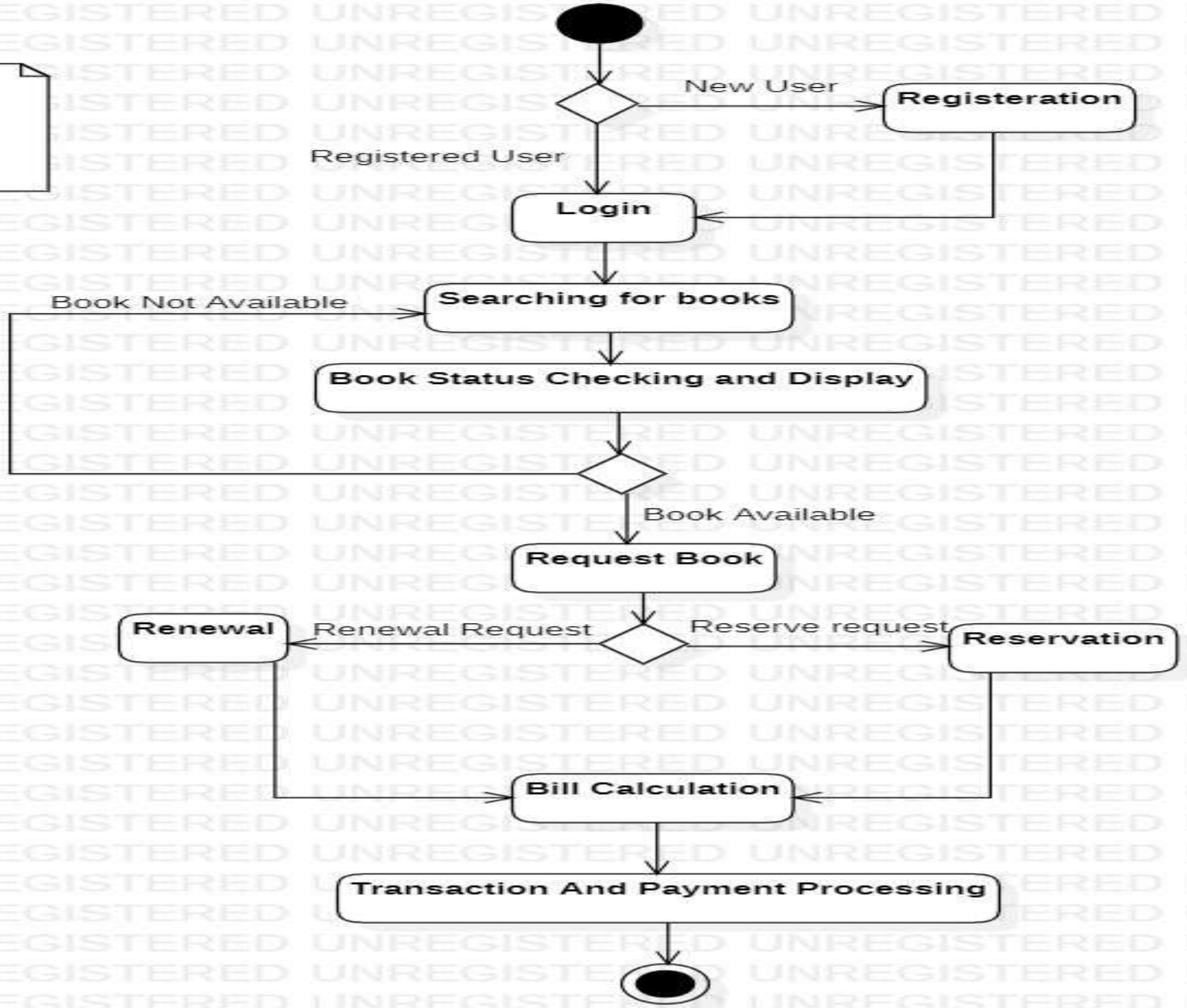

View Item
Product Found
Product not found
Search product
Buy now
Add to cart
View Cart
Update Cart
Login
Veiw or Edit Profile
Create Account
OTP
credit or debit card
Payment
upi or net banking
Payment done
exit
Order Confirm
Address Details
Final Order
Checkout
Online Shopping
StateChart Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2

UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTERED

library management
State Chart Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2

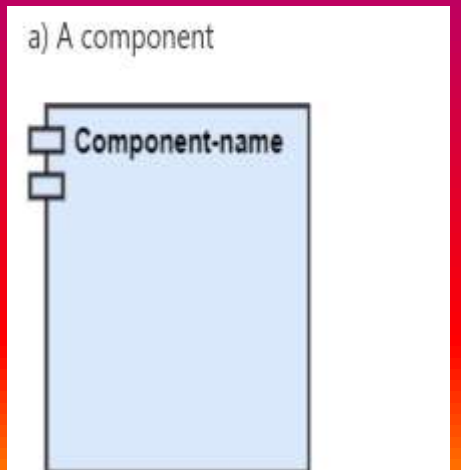


E-library
State Chart Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2



COMPONENT DIAGRAM

- It models the view of a system such as executables, files, libraries, etc. that resides withi
- A component is a single unit of the system, which is replaceable and executable.
- The implementation details of a component are hidden
- components are libraries, files, executables etc.

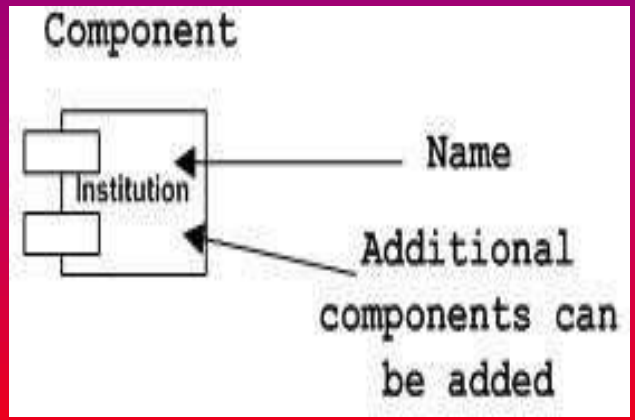


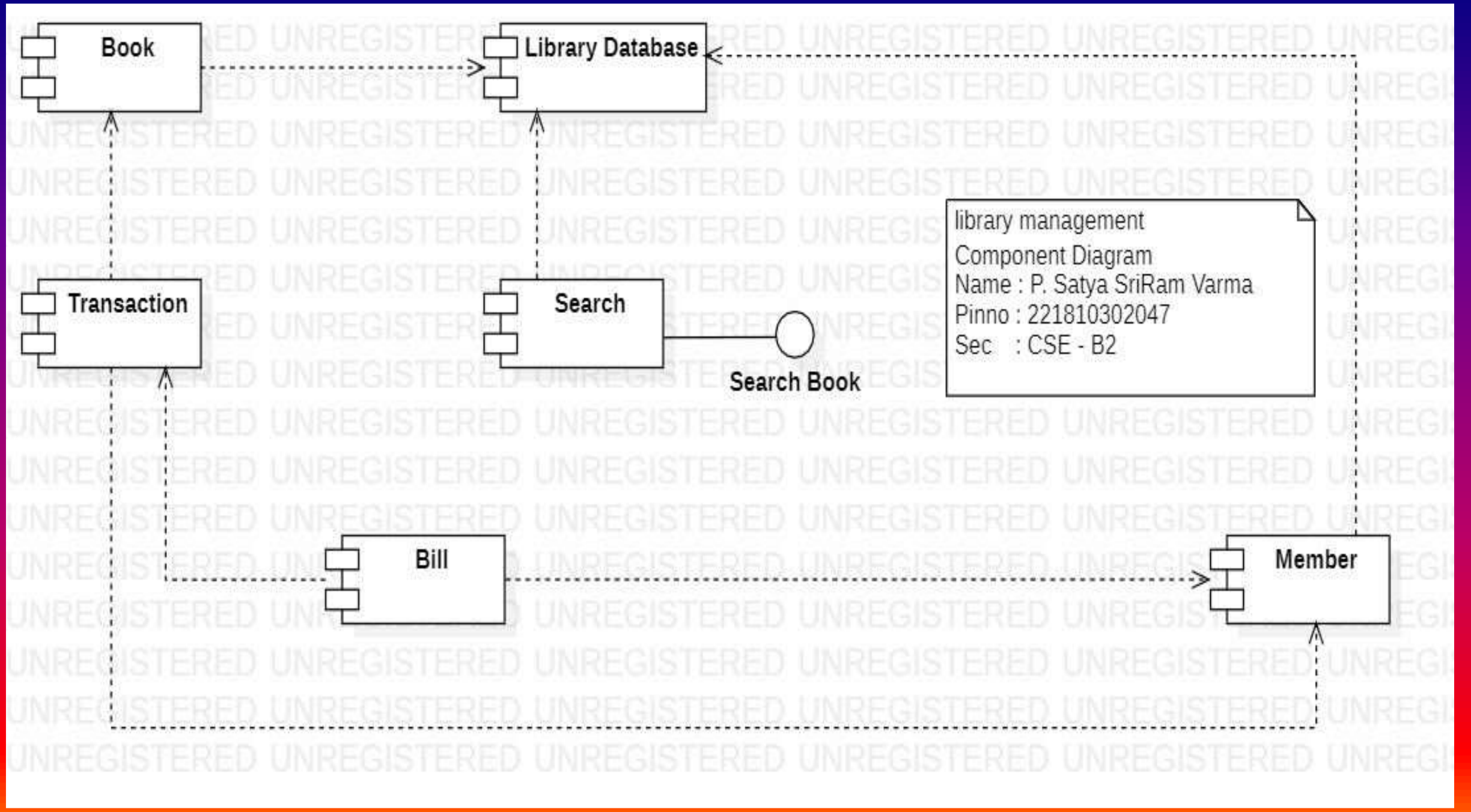
COMPONENT DIAGRAM

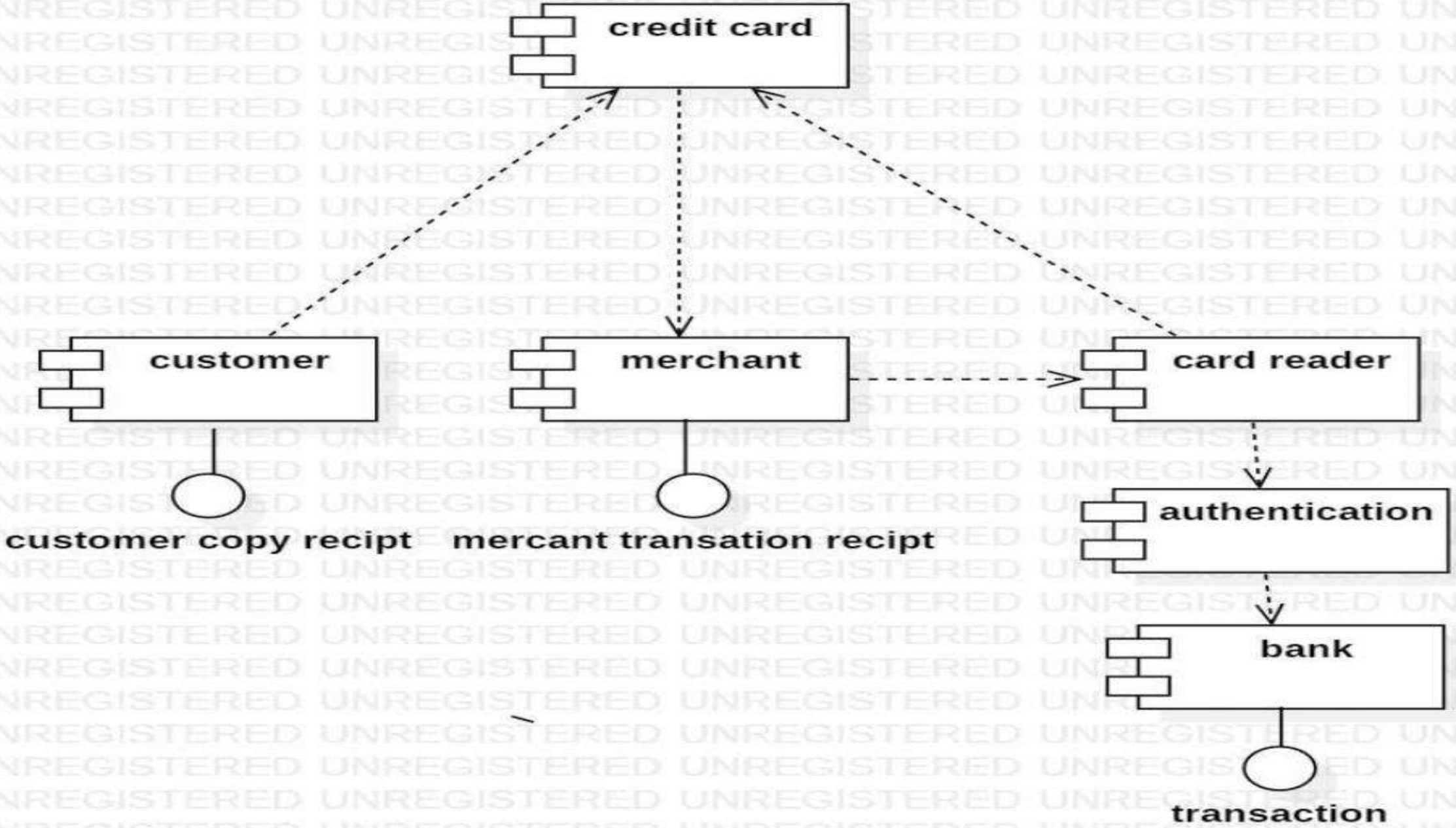
- To model the components of the system.
- To model the schemas of a database.
- To model the applications of an application.
- To model the system's source code.
- It depicts the relationships and organization of components.
- They represent a set of components and their relationships.
- They represent the implementation view of a system.

Structural Things – Component Notation

- A component in UML is shown as below with a name inside. Additional elements can be added wherever required.




Book
Library Database
Transaction
Search
Search Book
library management
Component Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2
Bill
Member


credit card
customer
merchant
card reader
authentication
bank
customer copy recipt
mercant transation recipt
transaction

DEPLOYMENT DIAGRAM

- visualizes the physical hardware on which the software will be deployed.
- It portrays the static deployment view of a system.
- It involves the nodes and their relationships.
- hardware topology of the system.
- It represent the hardware components on which the software components are installed.
- To describe the processing of nodes at the runtime.

Deployment diagrams

- Deployment diagrams visualize the topology of the physical components of a system
- They are a set of nodes and their relationships.
- These nodes are physical entities
- This is generally used by the deployment team.
- They describe the hardware components where software components are deployed.
- Component diagrams and deployment diagrams are closely related.

Deployment diagrams

- The purpose of deployment diagrams can be described as:
- Visualize hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe runtime processing nodes.

Deployment diagram

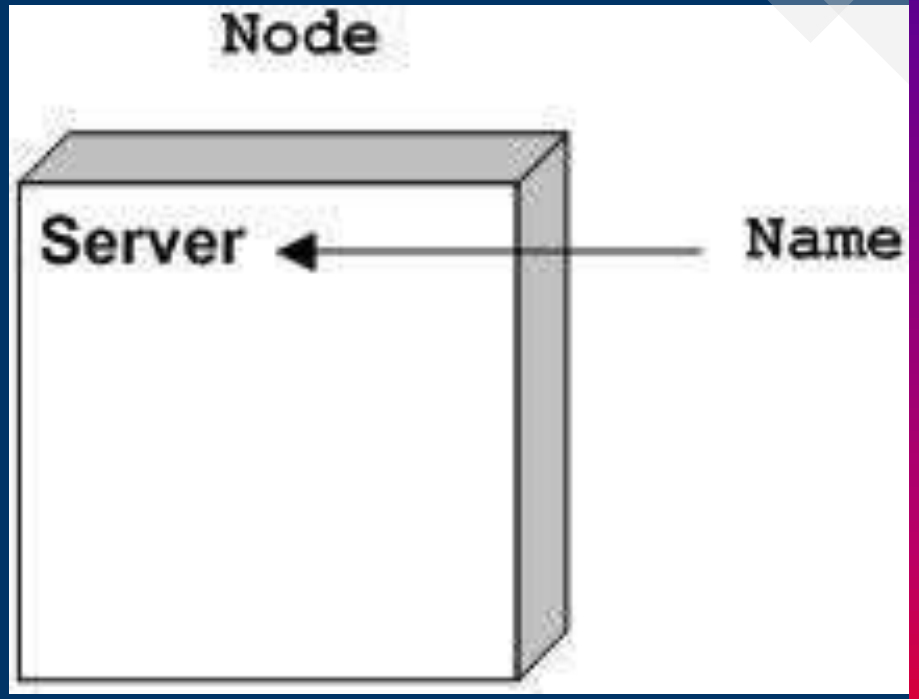
- An efficient deployment diagram is very important because it controls the following parameters
- Performance
- Scalability
- Maintainability
- Portability

So before drawing a deployment diagram the following should be identified:

- Nodes
- Relationships among nodes

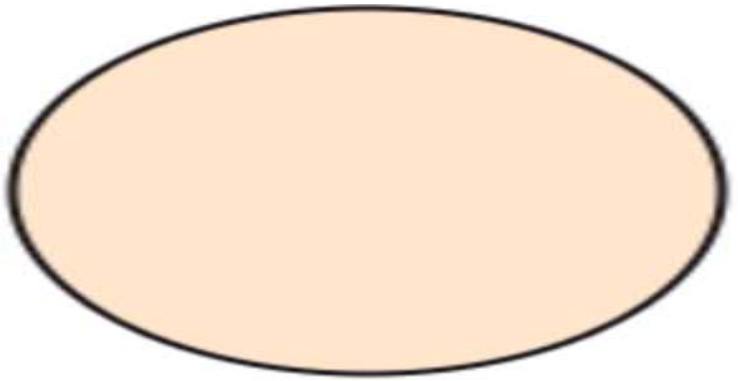
Structural Things – Node Notation

- A node in UML is represented by a square box as shown below with a name. A node represents a physical component of the system
- Node is used to represent physical part of a system like server, network etc.

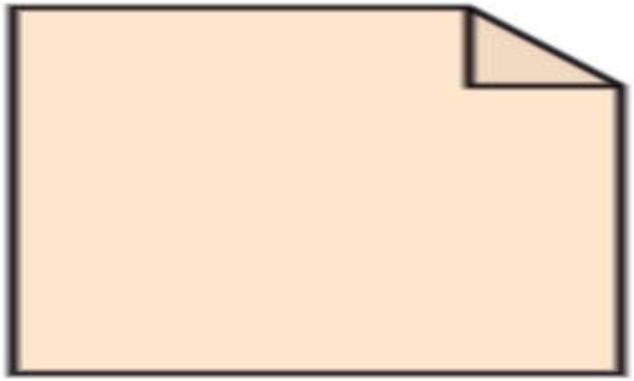


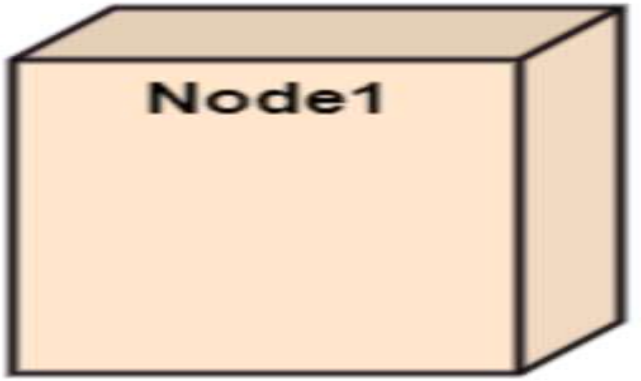
The deployment diagram consist of the following notations:

1. A component
2. An artifact
3. An interface
4. A node

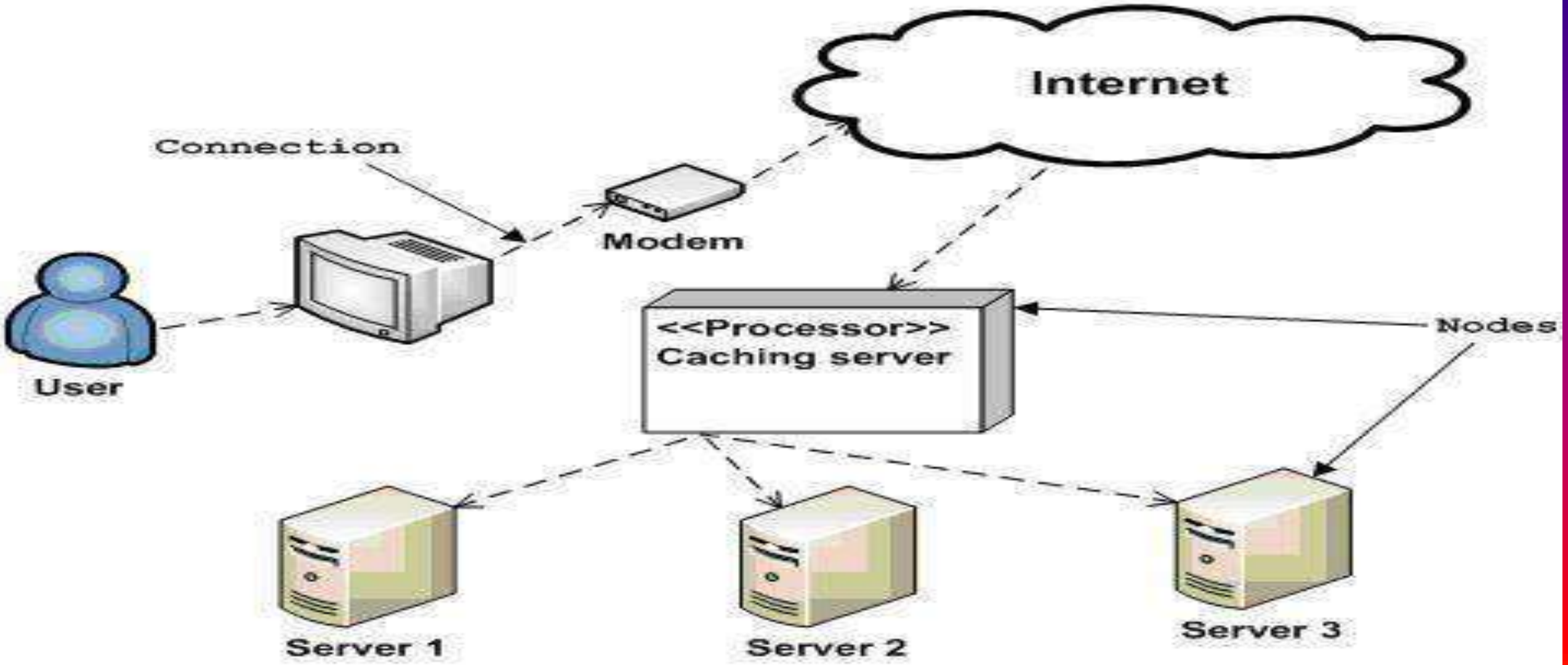


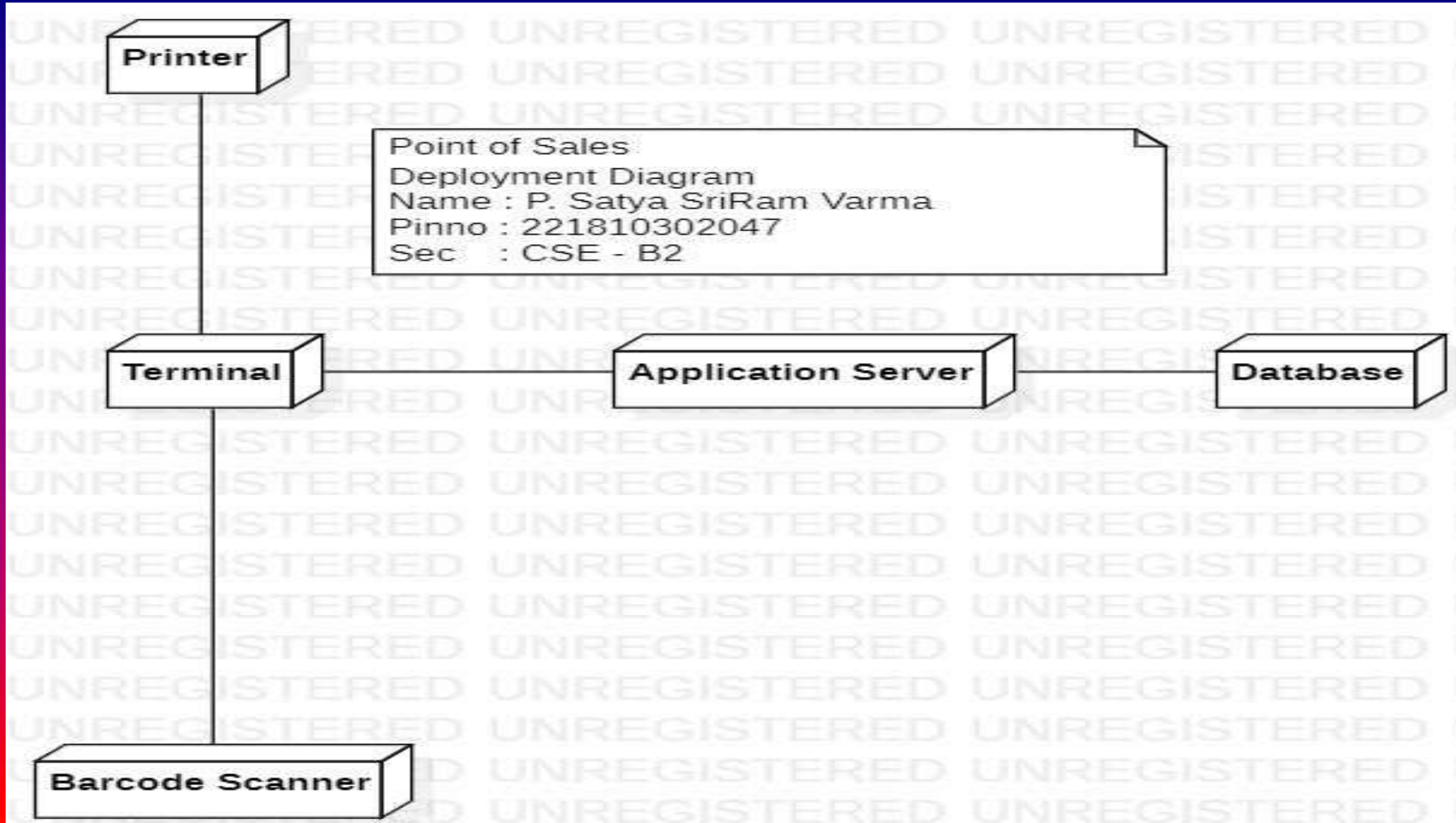
Interface1

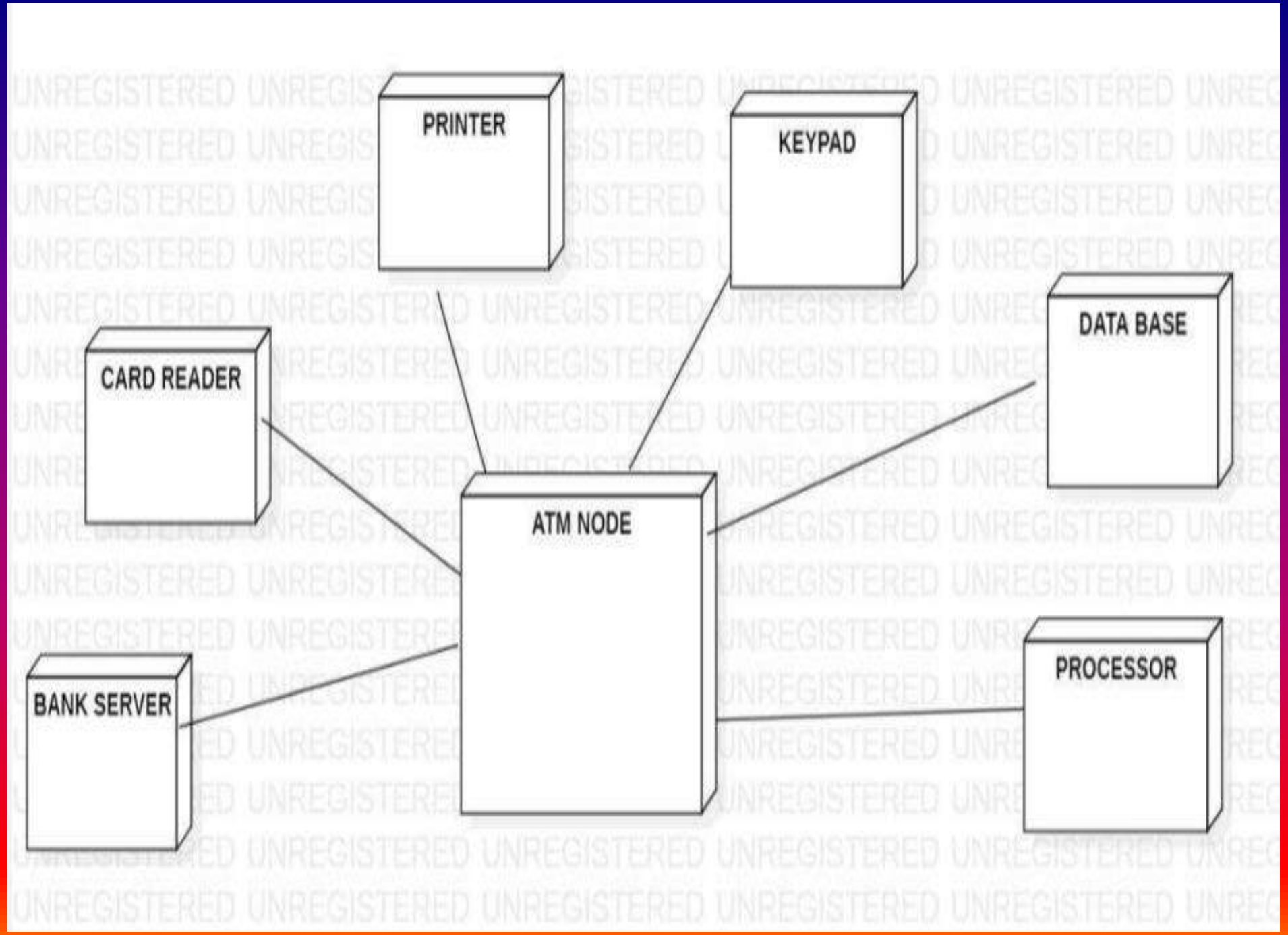


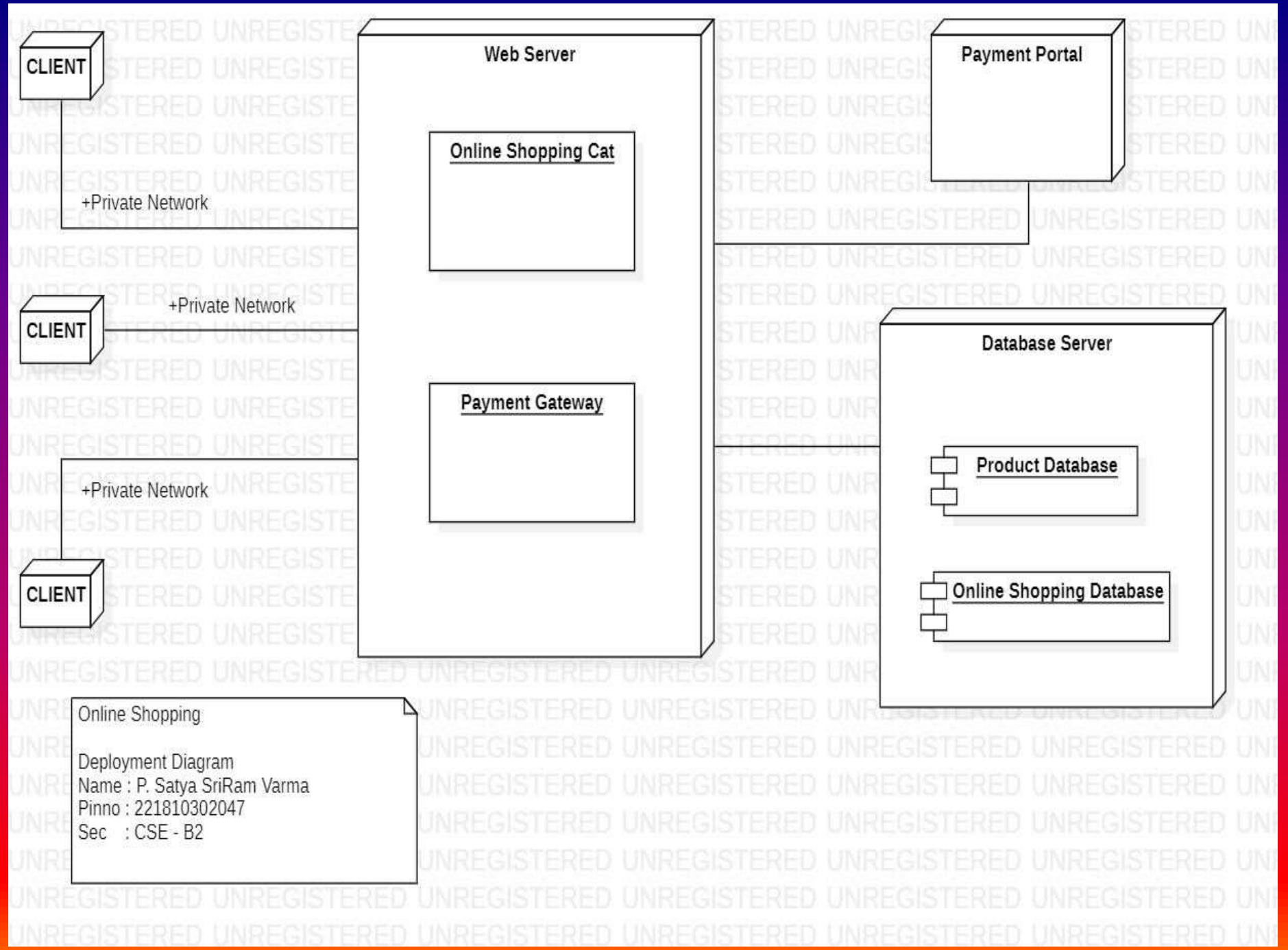


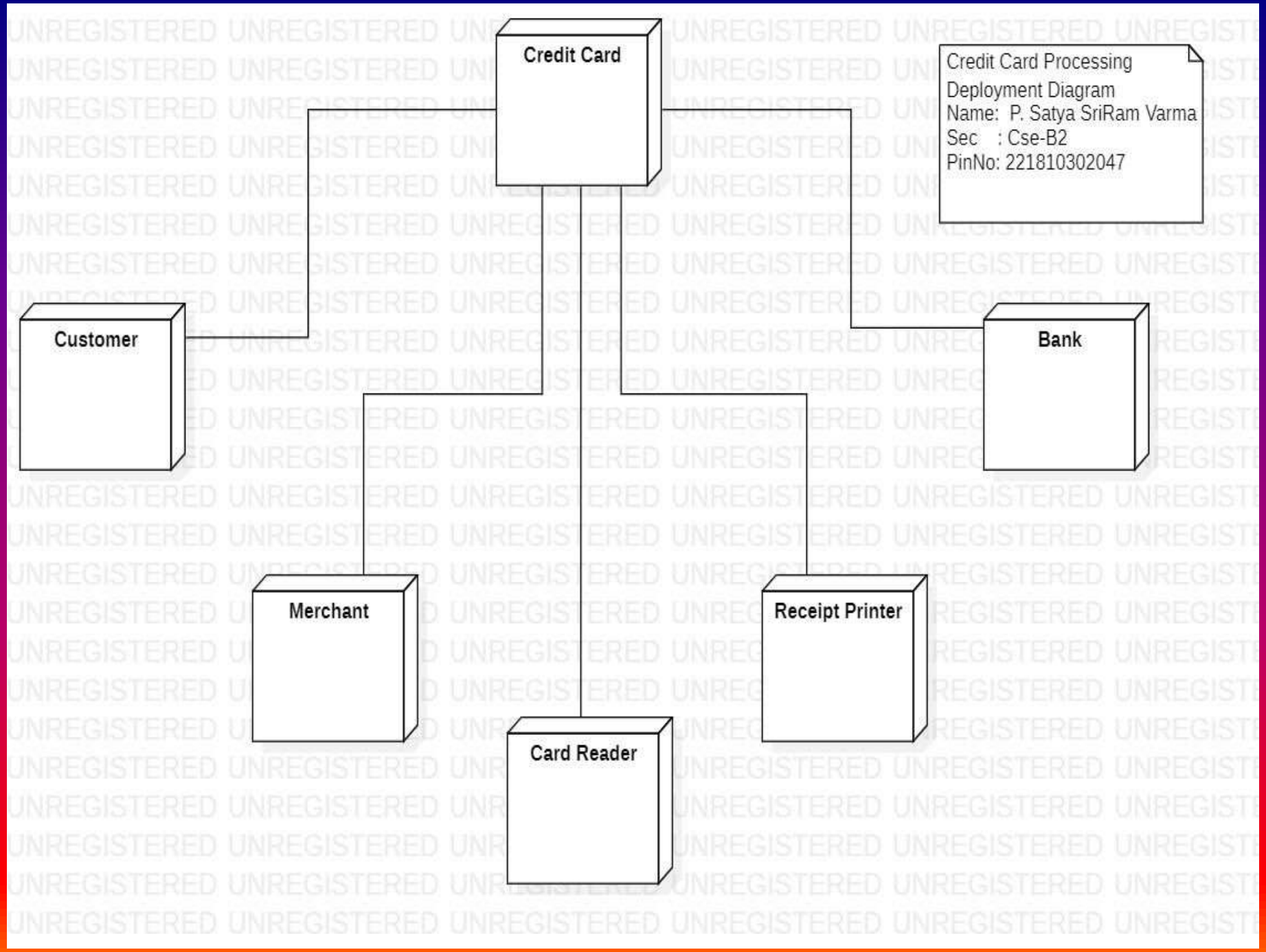
Deployment diagram of an order management system




Printer
Point of Sales
Deployment Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2
Terminal
Application Server
Database
Barcode Scanner


PRINTER
KEYPAD
CARD READER
DATA BASE
ATM NODE
BANK SERVER
PROCESSOR


CLIENT
CLIENT
CLIENT
+Private Network
+Private Network
+Private Network
Web Server
Online Shopping Cat
Payment Gateway
Payment Portal
Database Server
Product Database
Online Shopping Database
Online Shopping
Deployment Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2


Credit Card
Credit Card Processing
Deployment Diagram
Name: P. Satya SriRam Varma
Sec : Cse-B2
PinNo: 221810302047
Customer
Bank
Merchant
Receipt Printer
Card Reader

ACTIVITY DIAGRAM

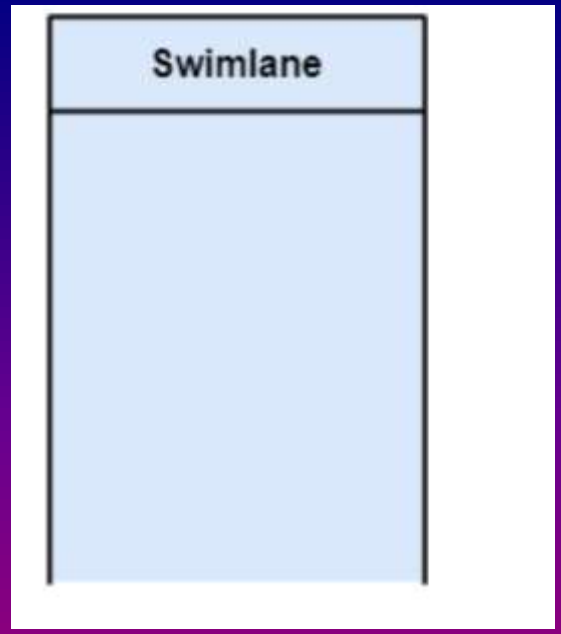
- It models the concurrent and sequential activities.
- flow can be sequential, branched, or concurrent
- It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram

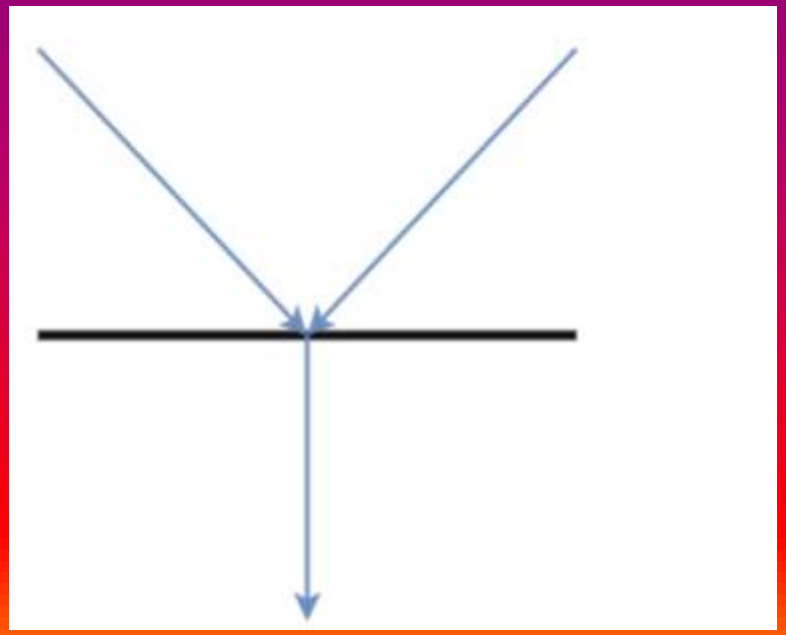
ACTIVITY DIAGRAM

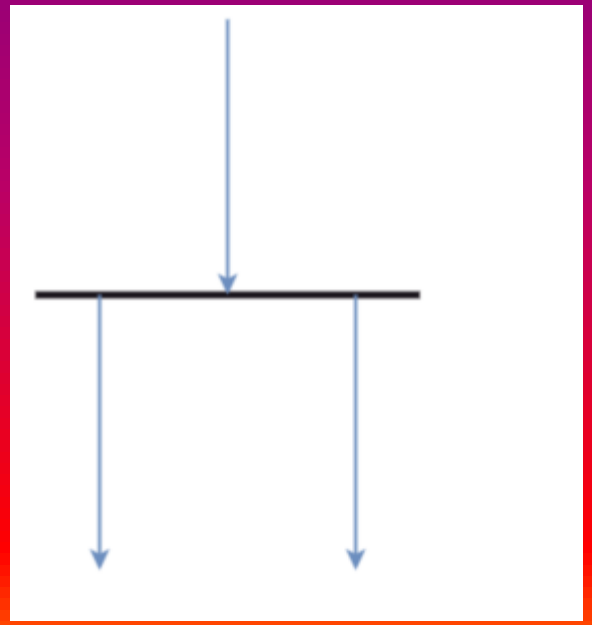
- **Swimlane:** It is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal.
- **Forks:** Forks and join nodes generate the concurrent flow inside the activity.
- A fork node consists of one inward edge and several outward edges.
- Join nodes are the opposite of fork nodes.

Activity diagram

- It describes the flow of control in a system.
- It consists of activities and links.
- The flow can be sequential, concurrent or branched.

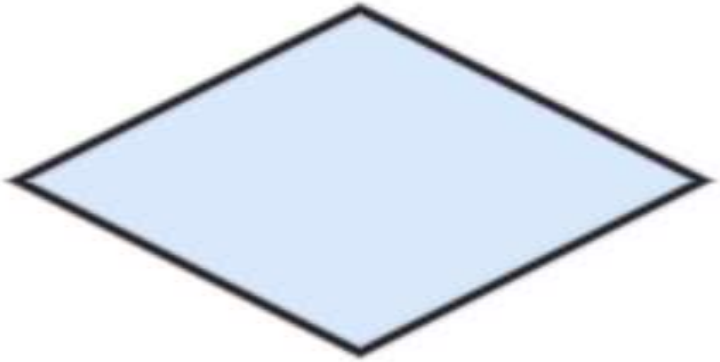

Swimlane





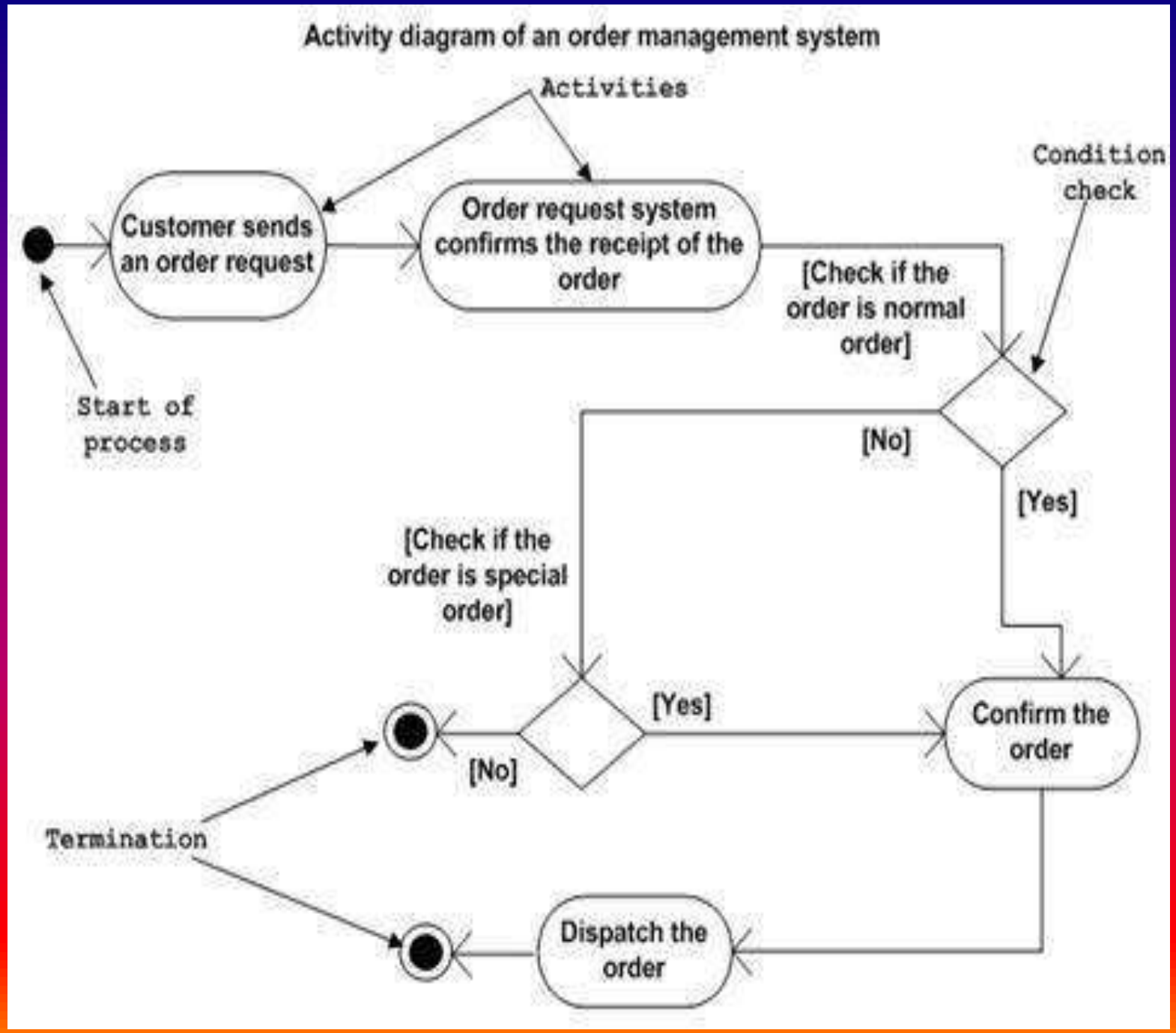
Initial State

Action box

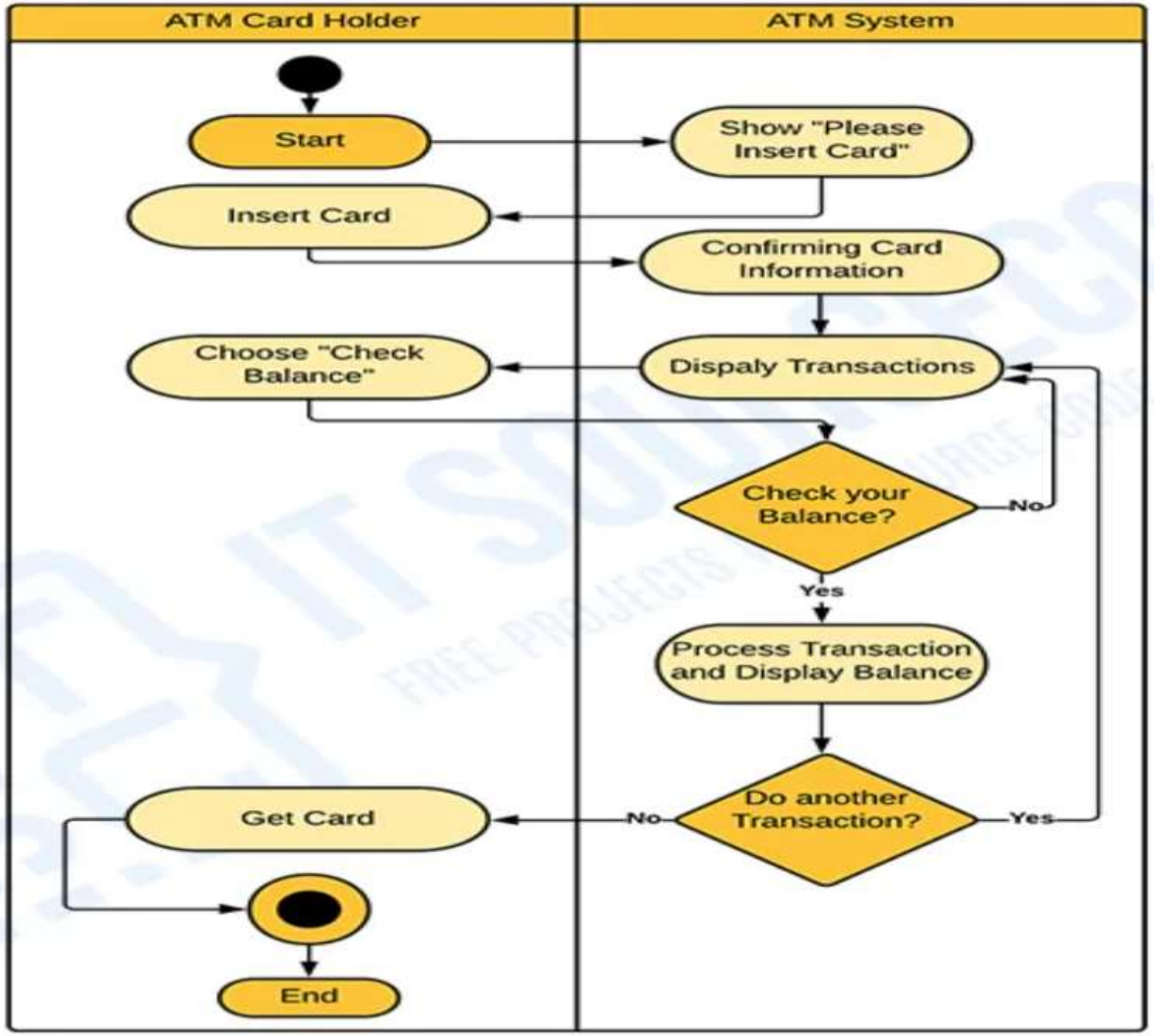


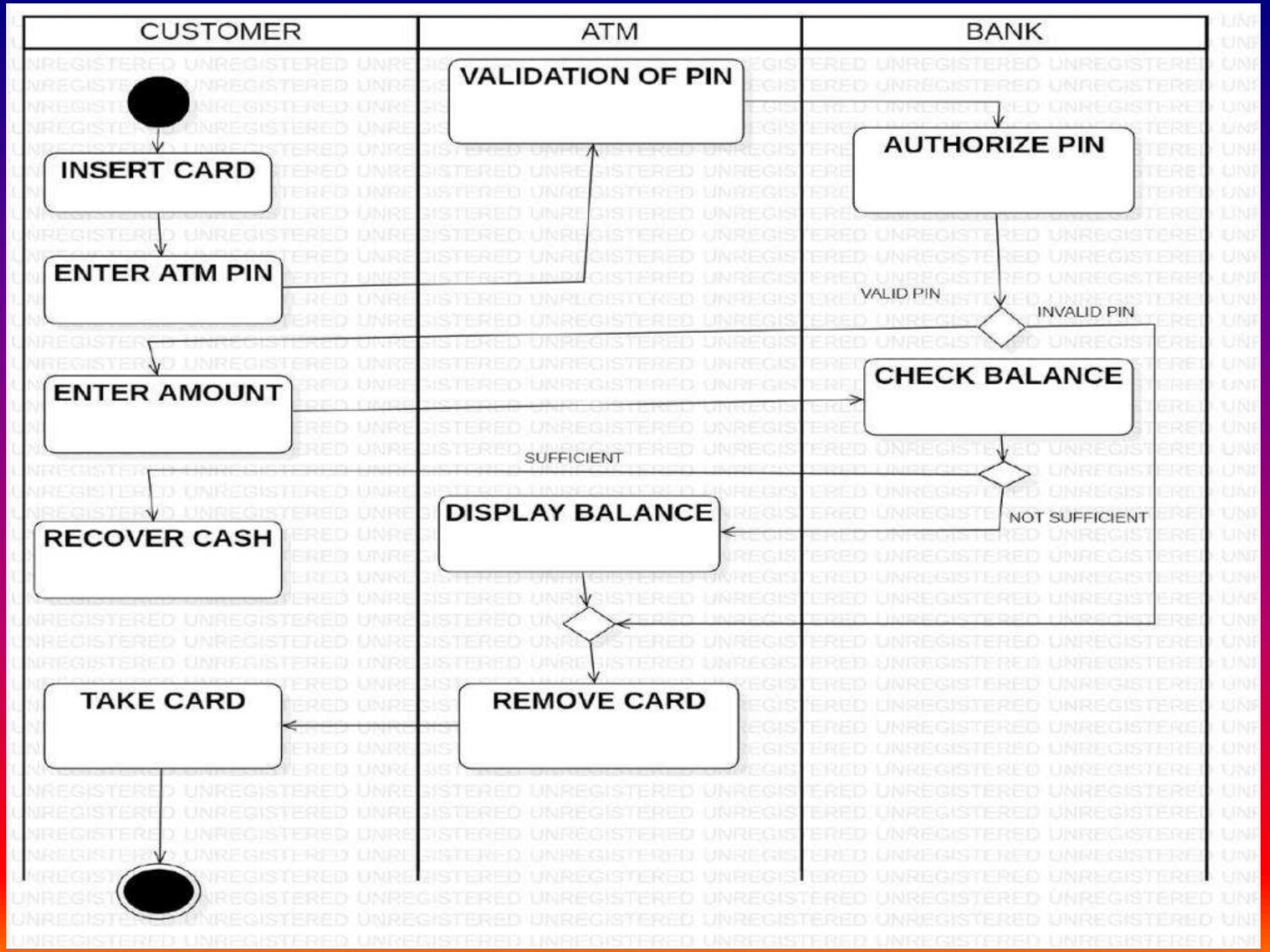
Decision-box

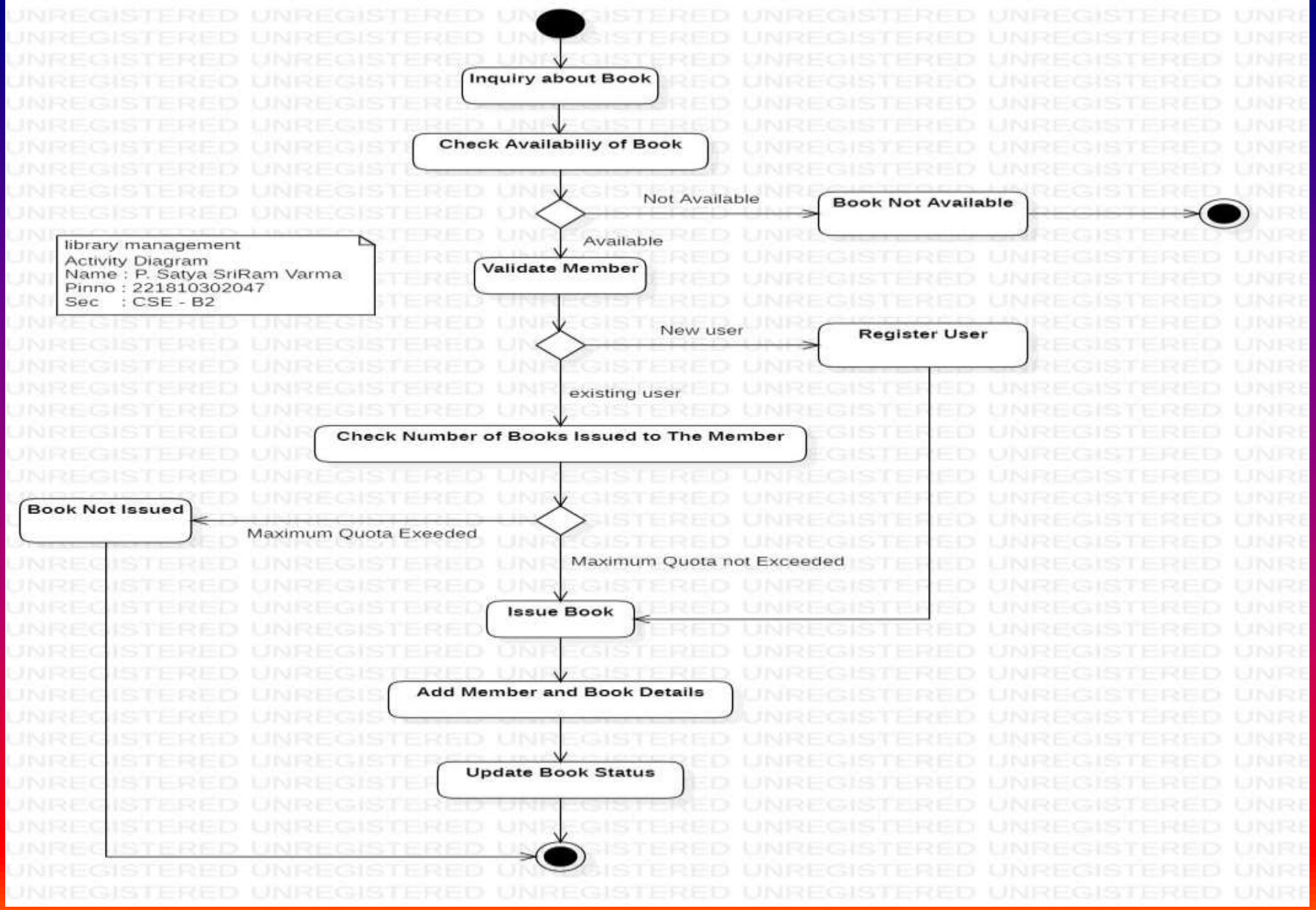
Final State

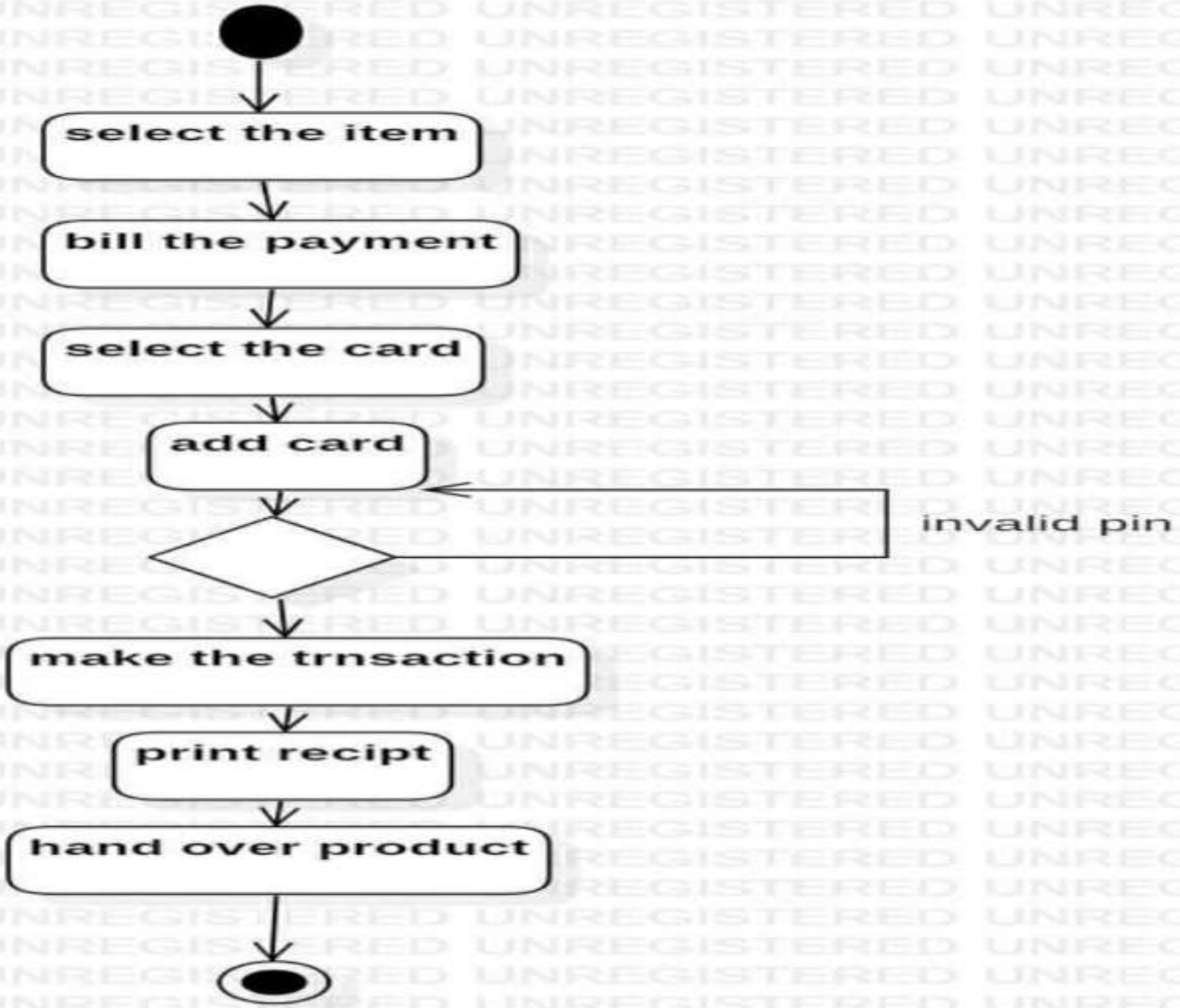

Activity diagram of an order management system
Activities
Condition check
Customer sends an order request
Order request system confirms the receipt of the order
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

ATM MANAGEMENT SYSTEM




CUSTOMER
ATM
BANK
VALIDATION OF PIN
AUTHORIZE PIN
INSERT CARD
ENTER ATM PIN
VALID PIN
INVALID PIN
ENTER AMOUNT
CHECK BALANCE
SUFFICIENT
DISPLAY BALANCE
NOT SUFFICIENT
RECOVER CASH
TAKE CARD
REMOVE CARD


Inquiry about Book
Check Availabiliy of Book
Not Available
Book Not Available
Available
library management
Activity Diagram
Name : P. Satya SriRam Varma
Pinno : 221810302047
Sec : CSE - B2
Validate Member
New user
Register User
existing user
Check Number of Books Issued to The Member
Book Not Issued
Maximum Quota Exeeded
Maximum Quota not Exceeded
Issue Book
Add Member and Book Details
Update Book Status


select the item
bill the payment
select the card
add card
invalid pin
make the trnsaction
print recipt
hand over product