Requirements Engineering

Instructor: Mehroze Khan

Requirements Engineering

- The **requirements** for a system are the descriptions of the **services** that a system should provide and the **constraints** on its operation.
- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.
- The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering (RE).**

Requirements Engineering Tasks

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Management

Requirements Engineering Tasks

- **Inception**: understand problem, people, nature of solution, effectiveness of communication.
- **Elicitation**: Ask questions about objectives, targets, detailed requirements
- **Elaboration**: Identify software function, behavior, information. Develop Requirements Model!!! Analysis???
- **Negotiation**: Resolve conflicts. Prioritize Requirements!!!
- **Specification**: Write document containing requirements models, scenarios etc.
- **Validation**: Ensure that all requirements have been stated, unambiguously!
- **Management**: Identify, control, track requirements and changes

Elaboration

- Analyze, model, specify…
- Some Analysis Techniques
    - Data Flow Diagrams (DFD)
    - Use case Diagram
    - Object Models (ER Diagrams)
    - State Diagrams
    - Sequence Diagrams
    - Activity Diagrams

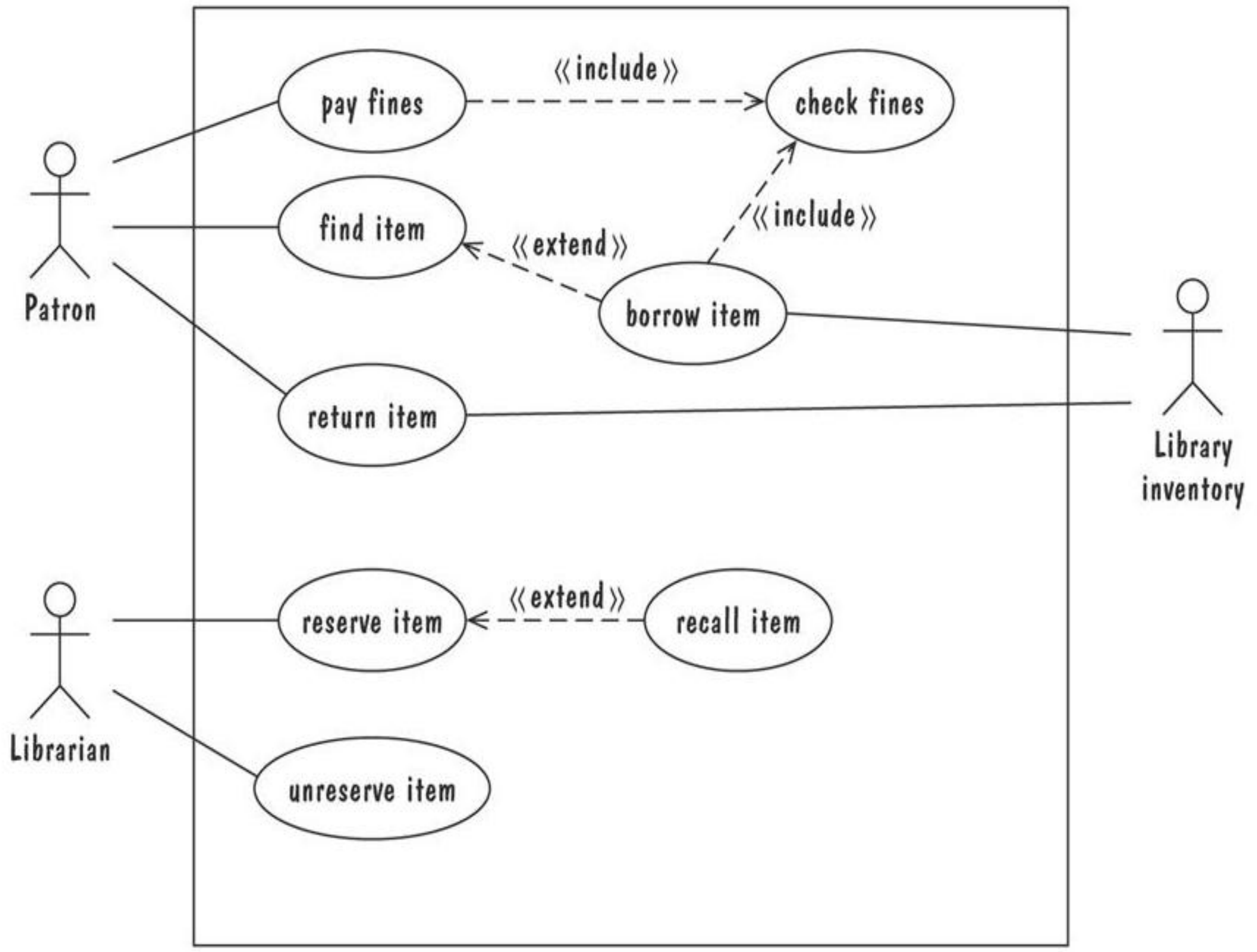
Requirements Modeling

1. Scenario-based Models
   - User Stories
   - Use Case Diagram
2. Class-oriented Models
   - Class Diagram
   - CRC Cards
3. Behavioral Models
   - State Diagram
   - Sequence Diagram
4. Flow-oriented Models
   - Data Flow Diagram

Use Case Diagram

- Components
    - **A large box:** *system boundary*
    - **Stick figures** outside the box: *actors*, both human and systems
    - Each **oval** inside the box: a use case that represents some major required functionality and its variant
    - A **line** between an actor and use case: the actor participates in the use case
- Use cases do not necessarily model all the tasks, instead they are used to specify user views of essential system behaviour

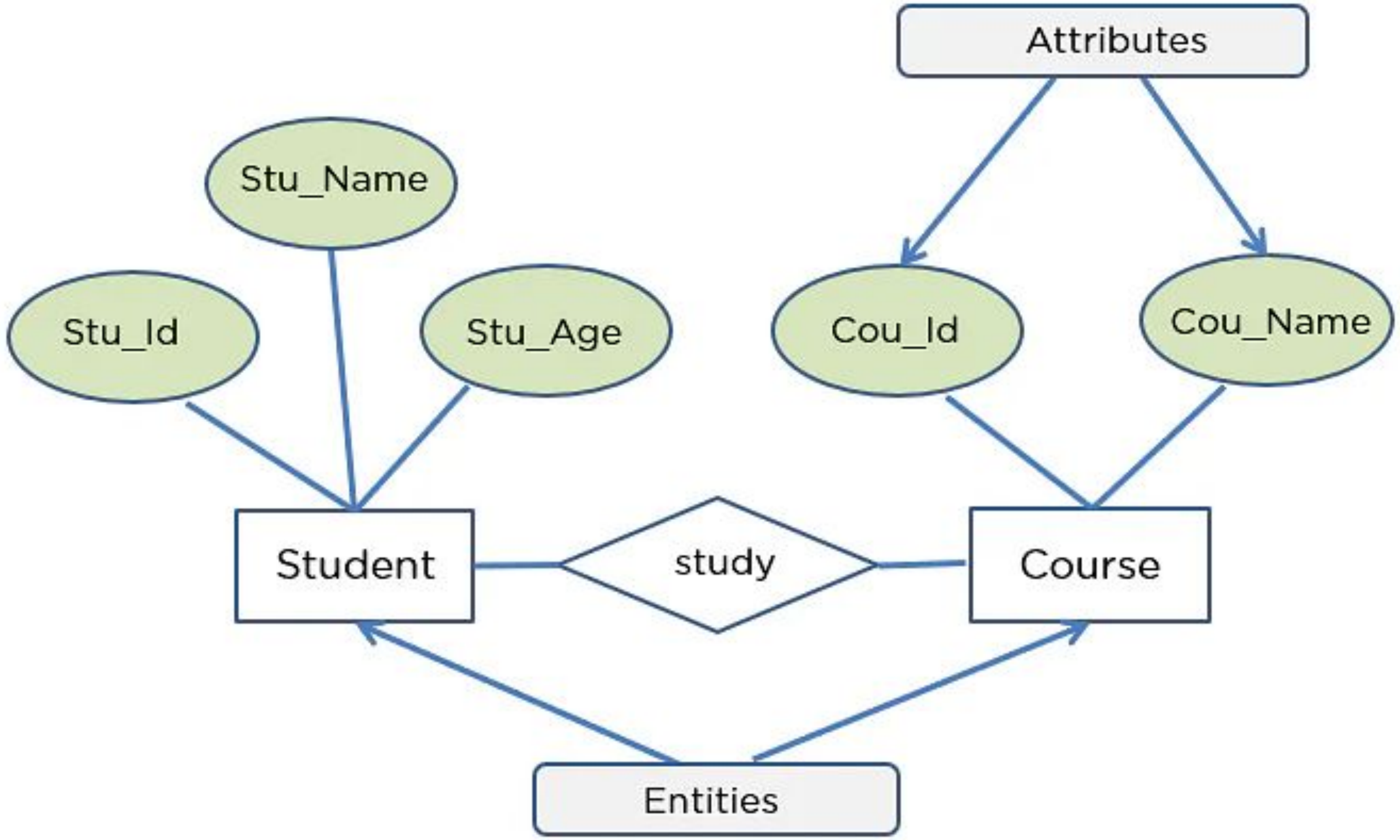
Use Case Diagram



Entity Relationship Diagram

- ER Model is used to model the **logical view** of the system from a data perspective.
- It consists of following components:
  - **An *entity***: depicted as a **rectangle**, represents a collection of real-world objects that have common properties and behaviours.
  - **A *relationship***: depicted as an **edge** between two entities, **with diamond** in the middle of the edge specifying the type of relationship.
  - **An *attribute***: represented as an **oval**, describes data or properties associated with the entity.

Entity Relationship Diagram



Entity Relationship Diagram

- ER diagrams are popular because
  - they provide an **overview of the problem** to be addressed.
  - the **view is relatively stable** when changes are made to the problem's requirements.
- ER diagram is likely to be used to model a problem early in the requirements process.

Class Diagram

- Class Diagram is a type of **static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- **Components:**
    - ***Objects***: akin to entities, organized in classes.
    - ***Attributes***: object's variables or characteristics.
    - ***Behaviours***: actions on the object's variables.

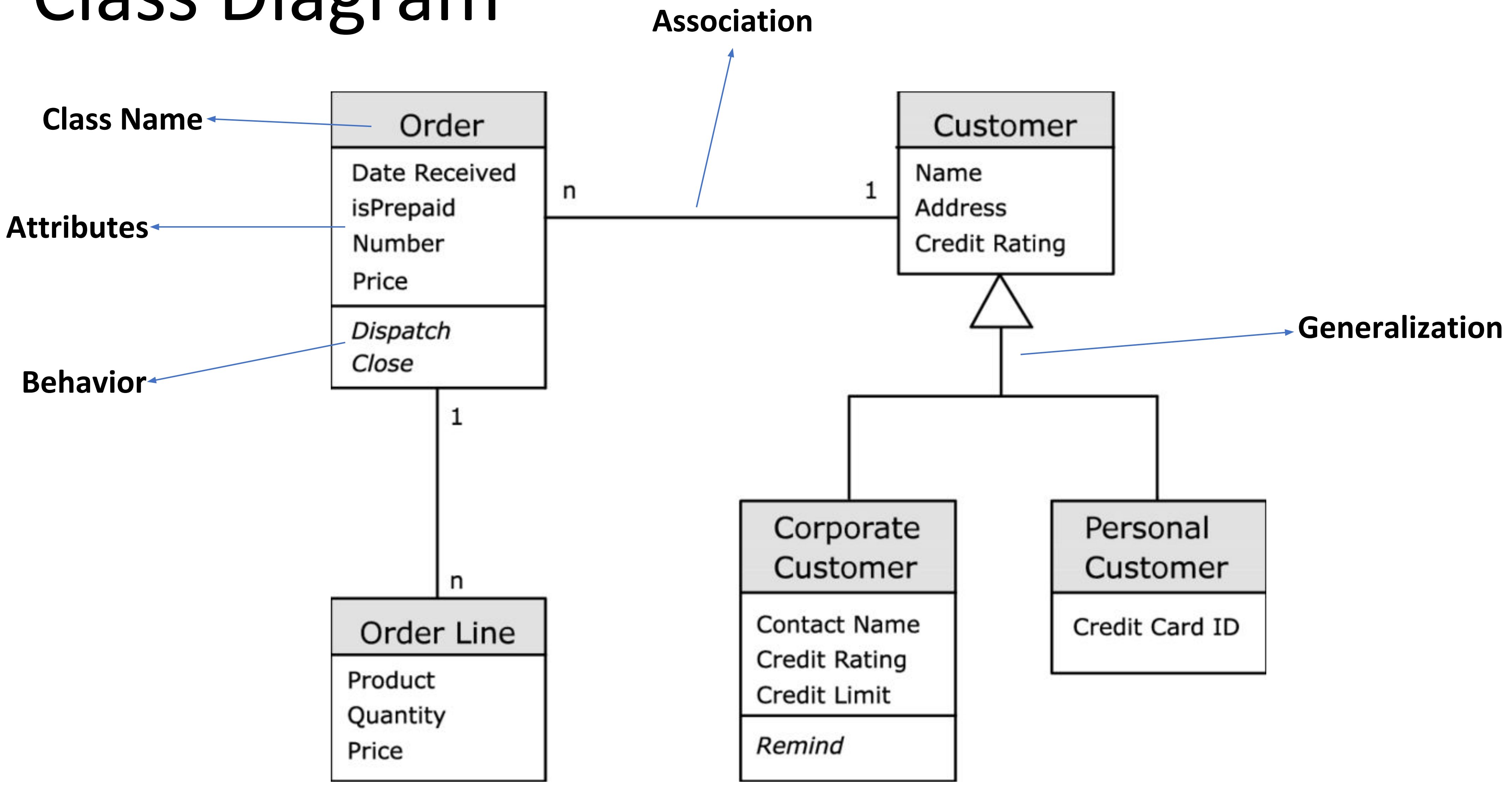
Types of Relationship

1. **Association:** Represents static relationship between classes.
2. **Dependency:** Represents dependency of one class on another class.
3. **Aggregation:** Represents has-a relationship. Contained class does not have strong dependency on container class.
4. **Composition:** Contained class has a strong dependency on container class.
5. **Generalization:** Represents is-a relationship. It has superclass and subclass relationships

Identifying Classes

- External Entities (producers, consumers)
- Things
- Occurrences or events
- Roles
- Organizational Units
- Places
- Structures

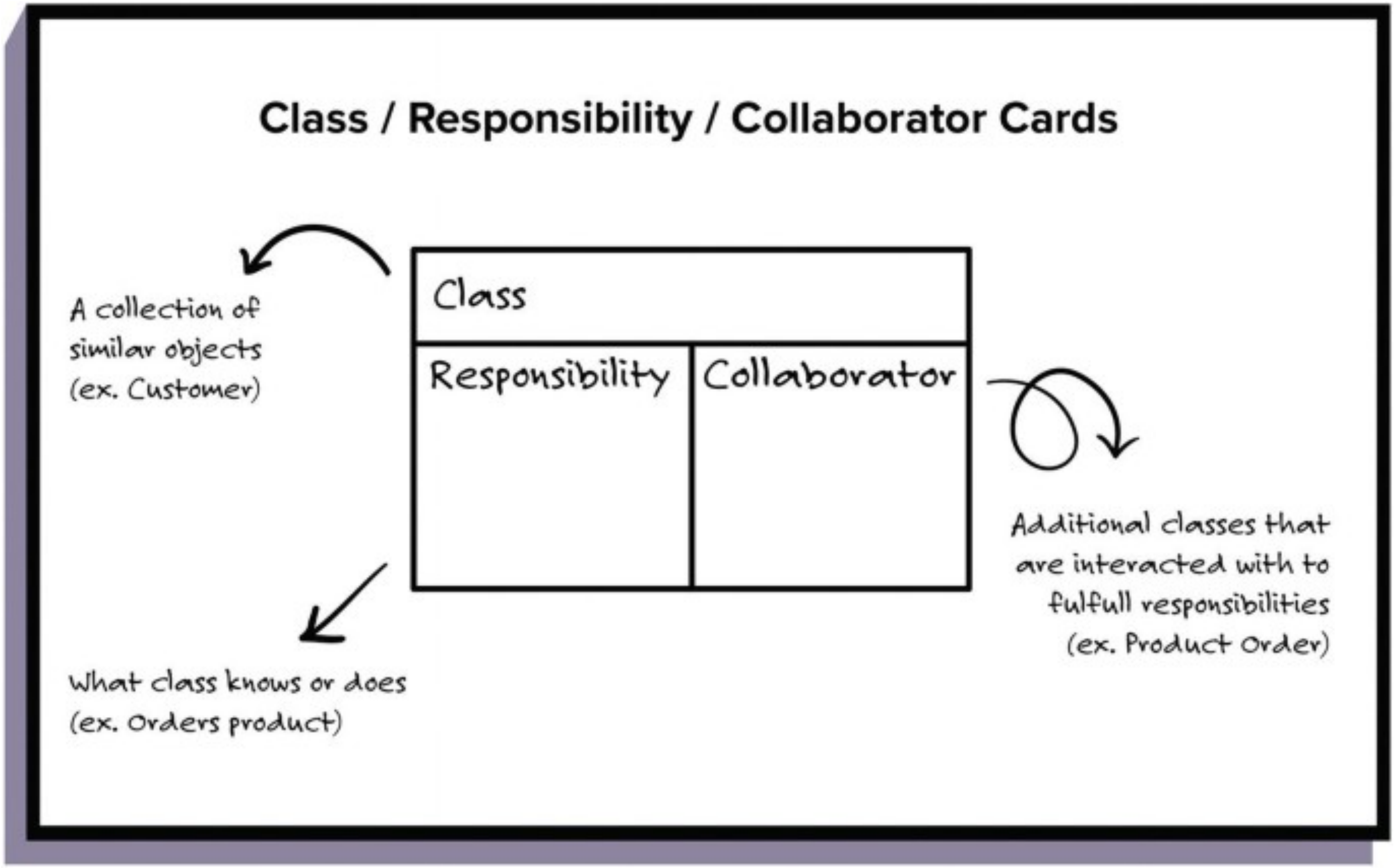
Class Diagram



Class-Responsibility-Collaborator Modeling

- Class-responsibility-collaborator (CRC) modeling provides a simple means for **identifying and organizing the classes** that are relevant to system or product requirements.
- A CRC model can be viewed as a **collection of index cards**.
- Each index card contains a list of **responsibilities** on the left side and the corresponding **collaborations** that enable the responsibilities to be fulfilled on the right side.
- **Responsibilities** are the attributes and operations that are relevant for the class.
- **Collaborators** are those classes that provide a class with the information needed or action required to complete a responsibility.

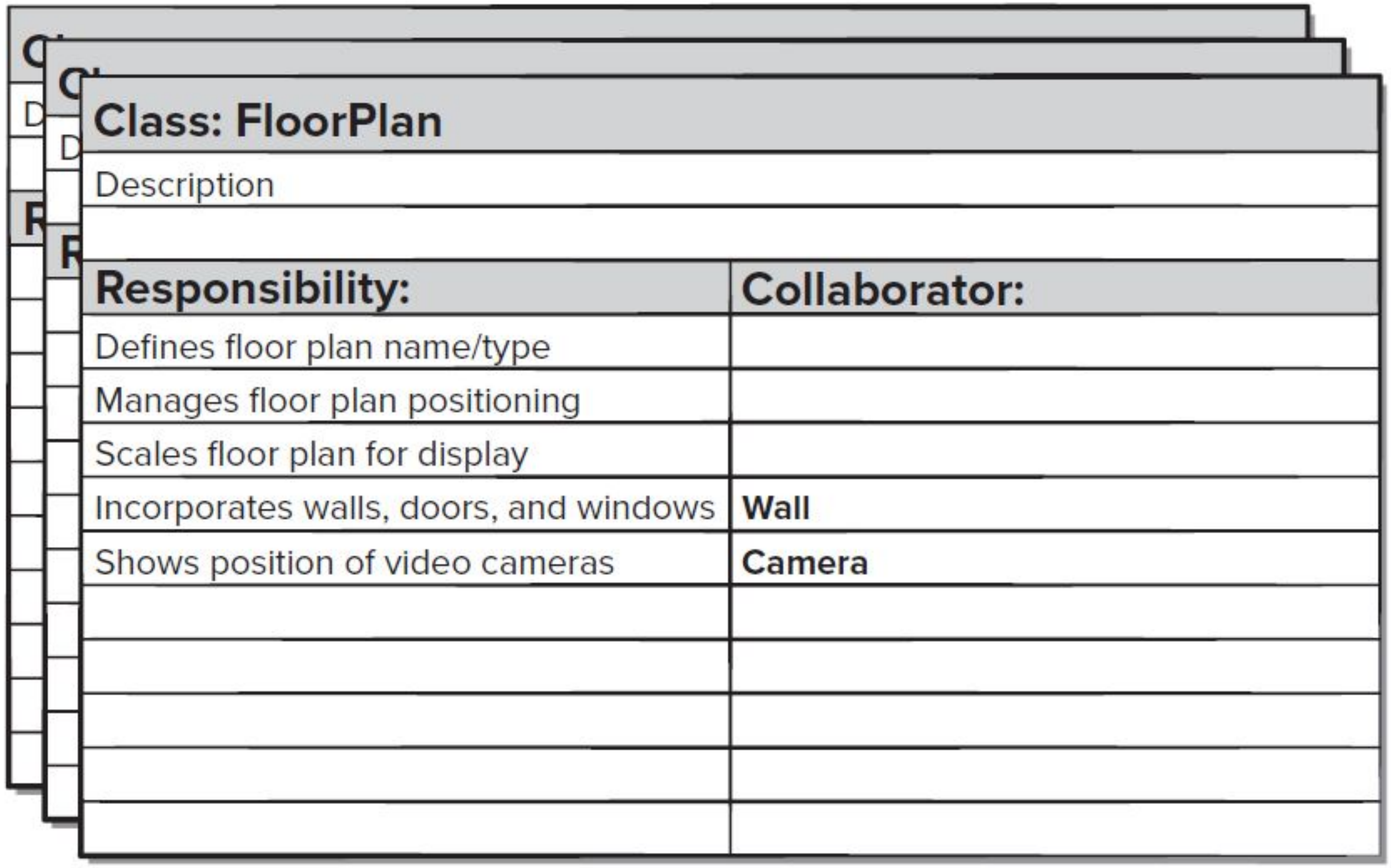
Class-Responsibility-Collaborator Modeling


Class / Responsibility / Collaborator Cards
Class
Responsibility
Collaborator
A collection of similar objects (ex. Customer)
What class knows or does (ex. Orders product)
Additional classes that are interacted with to fulfull responsibilities (ex. Product Order)

Class-Responsibility-Collaborator Modeling

- **Collaborations:** Classes fulfill their responsibilities in one of two ways:
  1. A class can use its own operations to manipulate its own attributes, thereby fulfilling a responsibility itself
  2. A class can collaborate with other classes.
- Collaborations are identified by determining whether a class can fulfill each responsibility itself. If it cannot, then it needs to interact with another class.

Class-Responsibility-Collaborator Modeling



Class: FloorPlan

Description

Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**

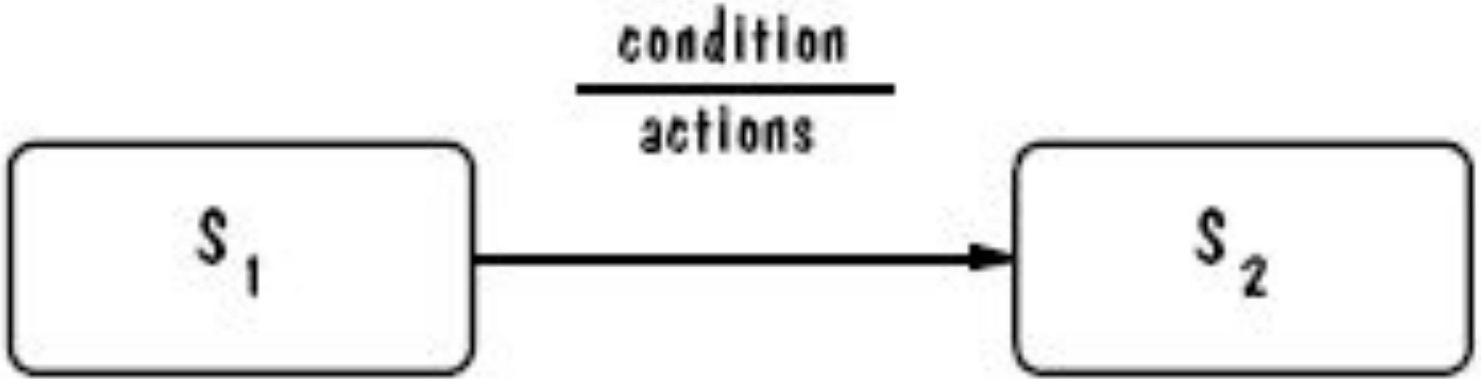
Class-Responsibility-Collaborator Modeling

- Once a complete CRC model has been developed, the representatives from the stakeholders can review the model using the following approach [Amb95]:
  1. All participants in the review (of the CRC model) are given a **subset of the CRC** model index cards. No reviewer should have two cards that collaborate.
  2. The review leader **reads the use case** deliberately. As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.
  3. When the token is passed, the holder of the class card is asked to **describe the responsibilities** noted on the card. The group determines whether one (or more) of the responsibilities satisfies the use case requirement.
  4. If an error is found, **modifications are made** to the cards. This may include the definition of **new classes** (and corresponding CRC index cards) or **revising lists** of responsibilities or collaborations on existing cards.

State Diagram

- A graphical description of all dialog between the system and its environment.
  - **Node** (*state*) represents a stable set of conditions that exists between event occurrences
  - **Edge** (*transition*) represents a change in behaviour or condition due to the occurrence of an event
- Useful both for **specifying dynamic behaviour** and for **describing how behaviour should change** in response to the history of events that have already occurred.

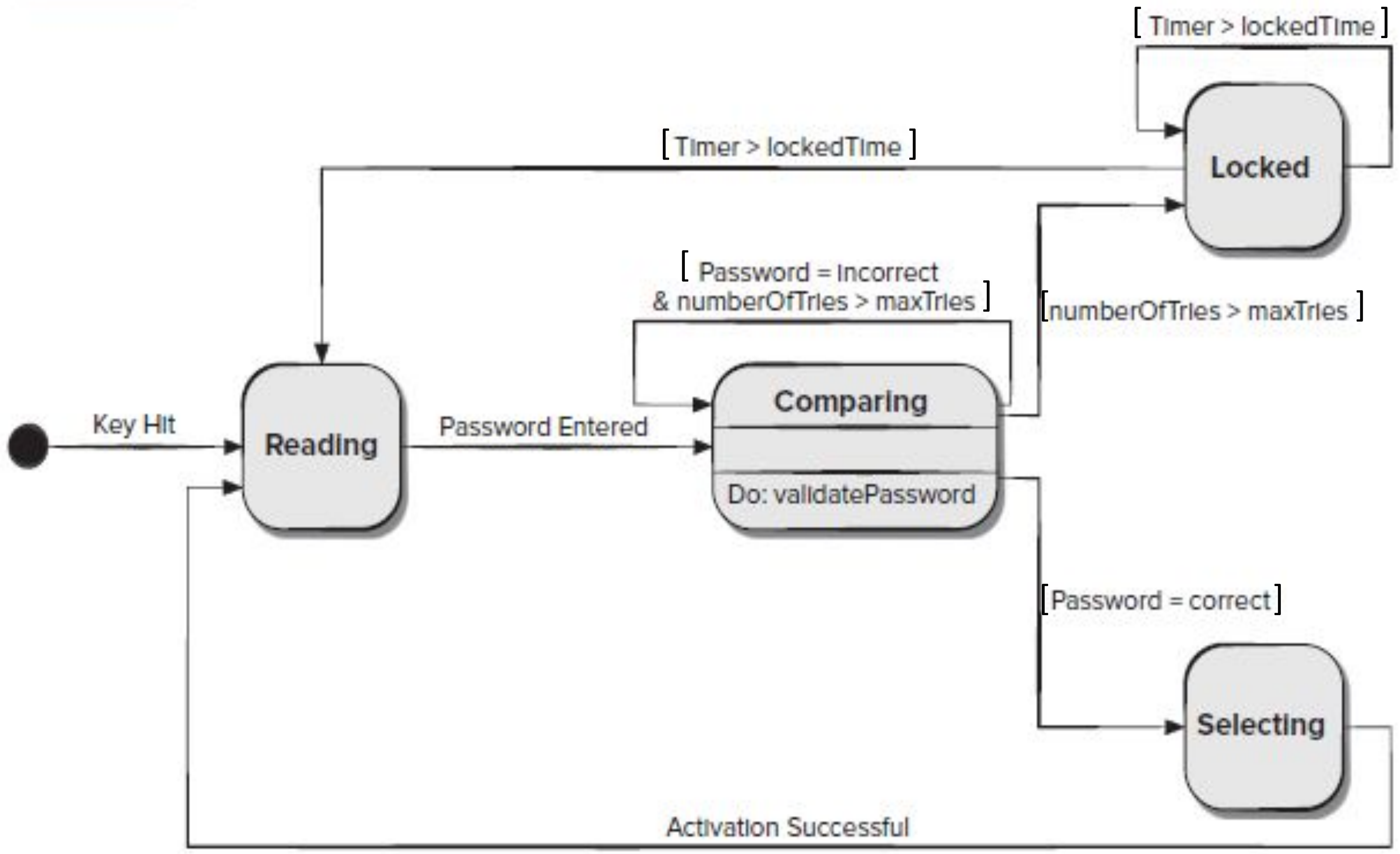
State Diagram: UML Notation



State Diagram: UML Statechart

- Statechart diagrams provide us an efficient way to **model the interactions** or **communication** that occur within the external entities and a system.
- These diagrams are used to **model the event-based system**.
- A UML model is a **collection of concurrently executing statecharts**.
- UML statechart diagram have a rich syntax, including **state hierarchy, concurrency, and inter-machine communication**.

State Diagram(Control Panel)

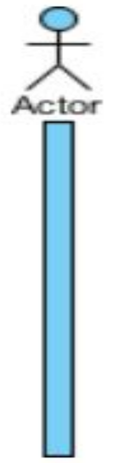


Sequence Diagram

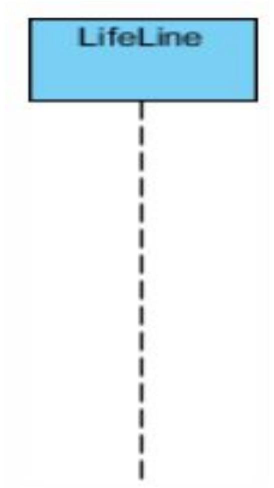
- A sequence diagram simply **depicts interaction between objects in a sequential order**, the order in which these interactions take place.
- Sequence diagrams describe **how and in what order the objects in a system function**.

Sequence Diagram Notations

- Actor:
  - Represent roles played by human users, external hardware, or other subjects.

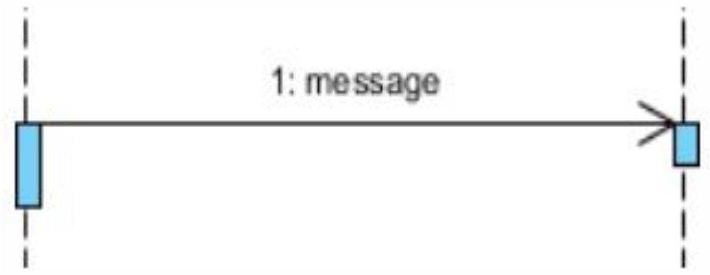


- Lifeline:
  - A lifeline represents an individual participant in the interaction.

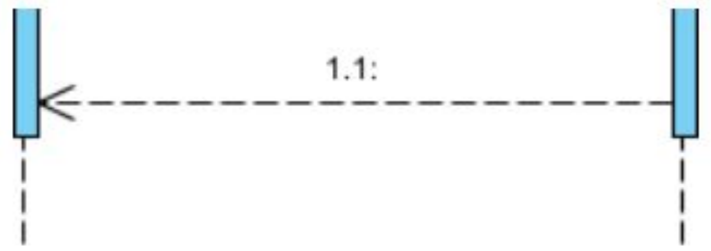


Sequence Diagram Notations

- Call Message:
  - It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

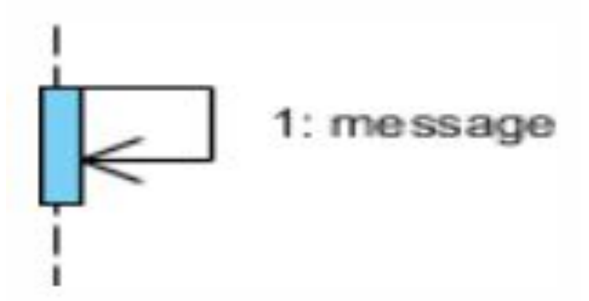


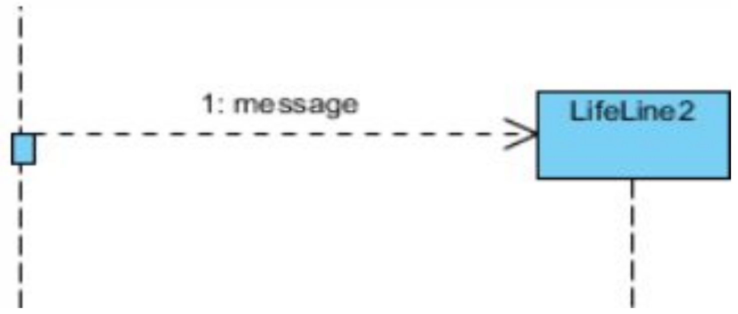
- Return Message:
  - It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.

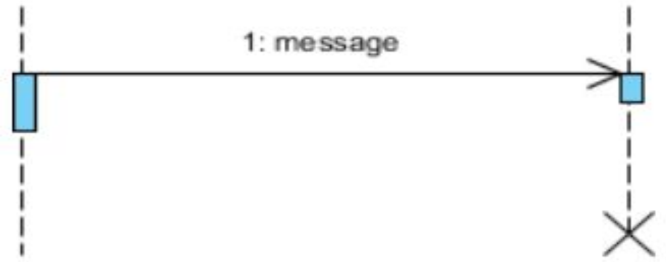


Sequence Diagram Notations

- Self Message:
  - It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.
- Create Message:
  - It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.
- Destroy Message:
  - It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.







Search Book : Use Case

- Main scenario -

The Customer specifies an author on the Search Page and then presses the Search button.

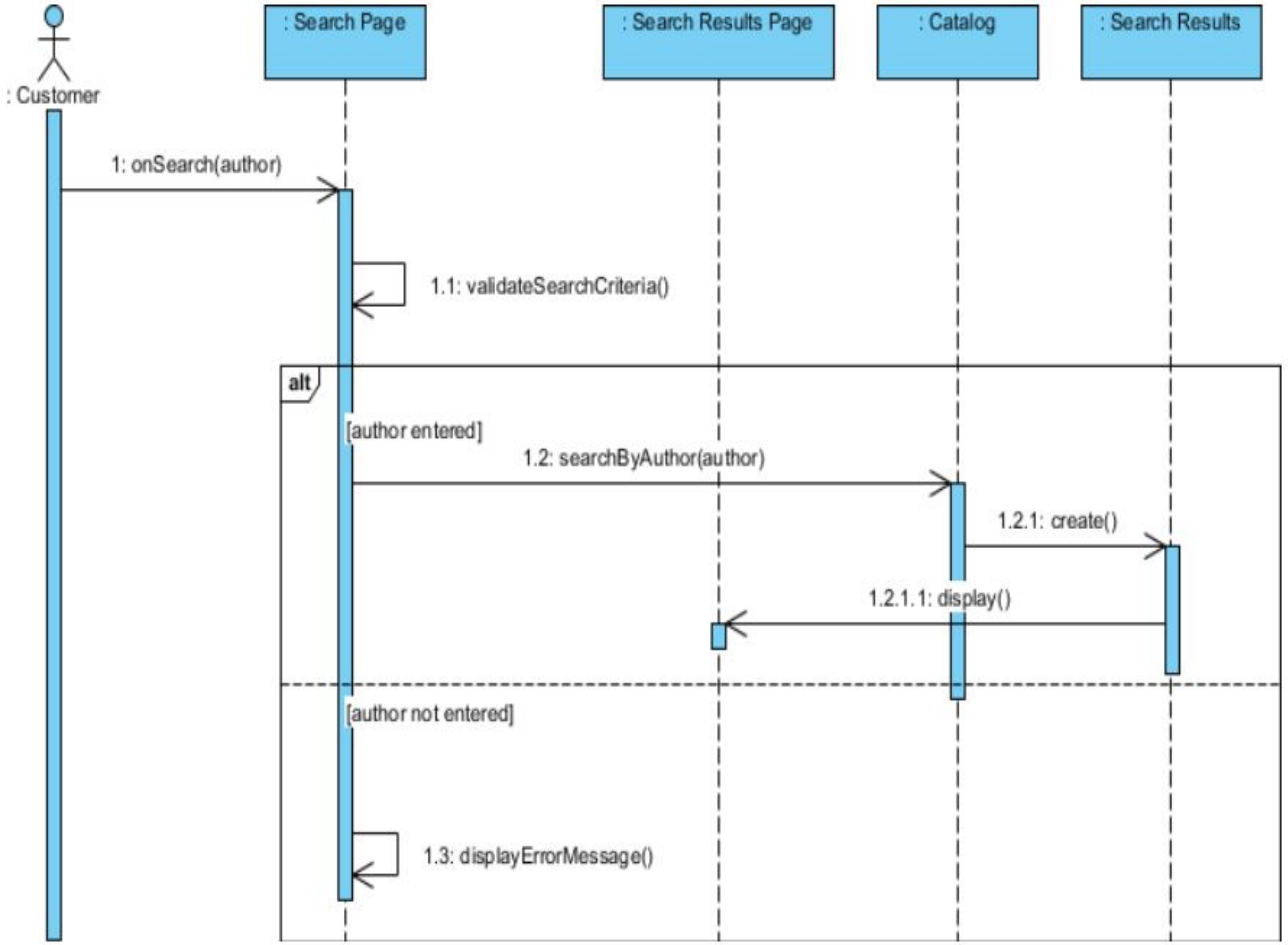
The system validates the Customer's search criteria.

If author is entered, the System searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results page.

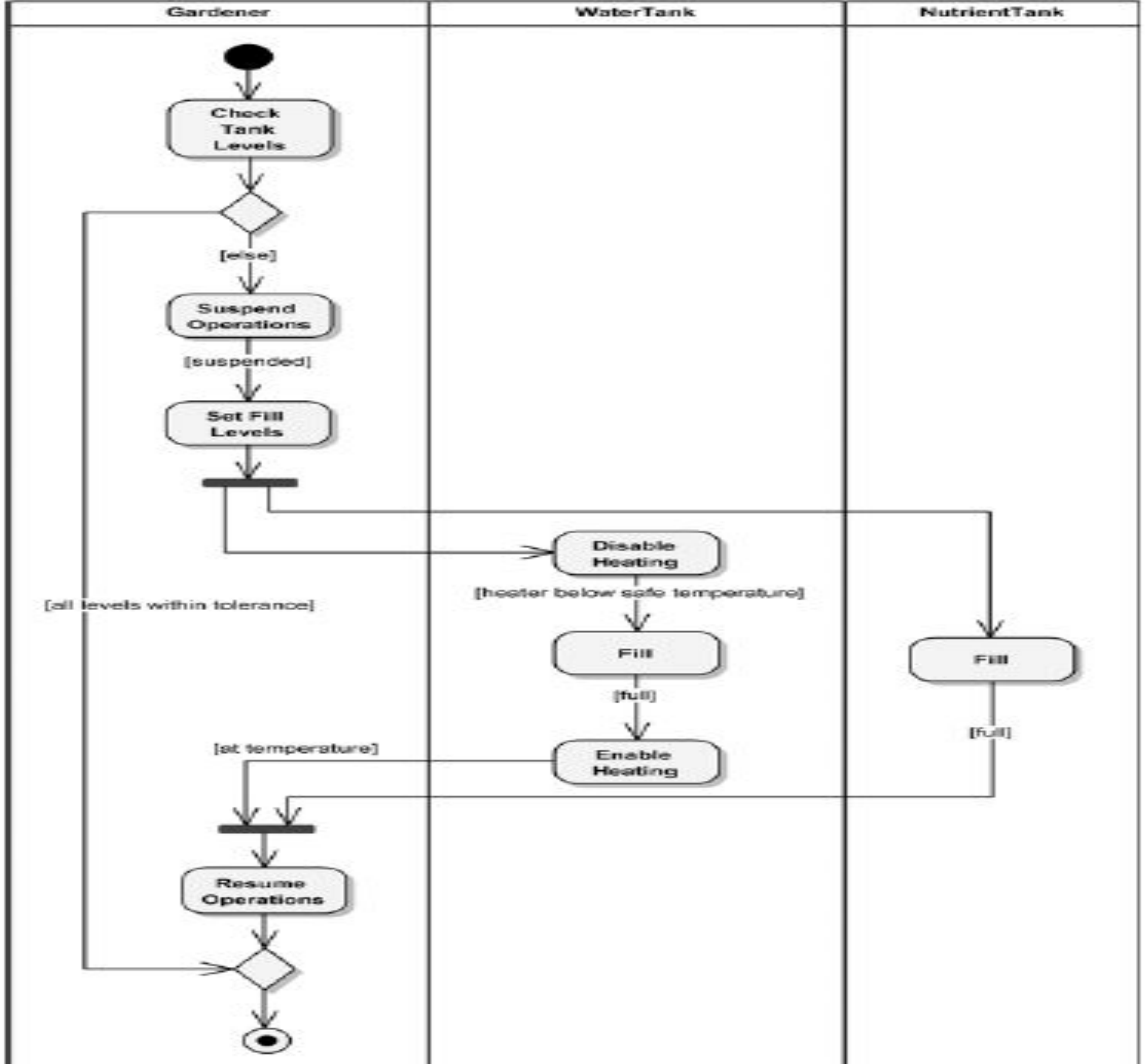
- Alternate path -

If the Customer did not enter the name of an author before pressing the Search button, the System displays an error message



Swimlane Diagrams

- A swim lane diagram is a type of flowchart that **defines who does what in a process**.
- Using the **metaphor of lanes in a pool**, a swim lane diagram provides clarity and accountability by placing process steps within the horizontal or vertical “swimlanes” of a particular employee, work group or department.
- It shows connections, communication and handoffs between these lanes, and it can serve to **highlight waste, redundancy and inefficiency in a process**.

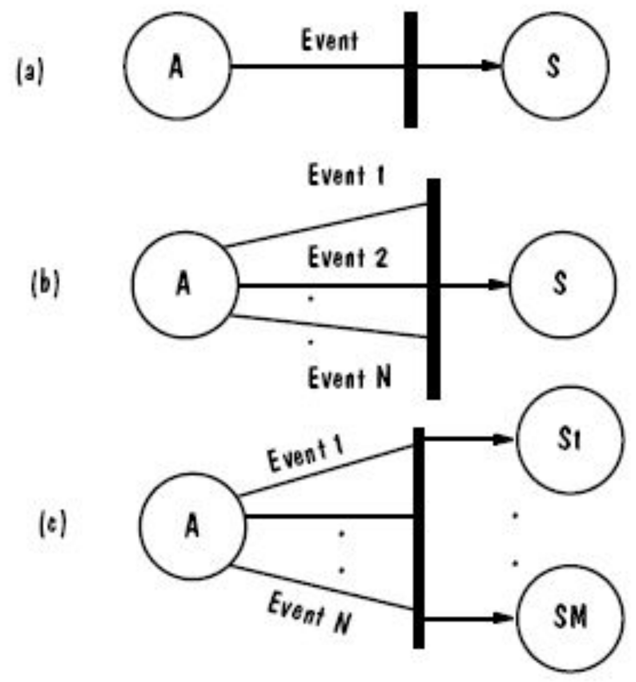

Gardener
WaterTank
NutrientTank
Check Tank Levels
[else]
Suspend Operations
[suspended]
Set Fill Levels
Disable Heating
[heater below safe temperature]
Fill
[full]
Enable Heating
Fill
[full]
[all levels within tolerance]
[at temperature]
Resume Operations

Petri Nets

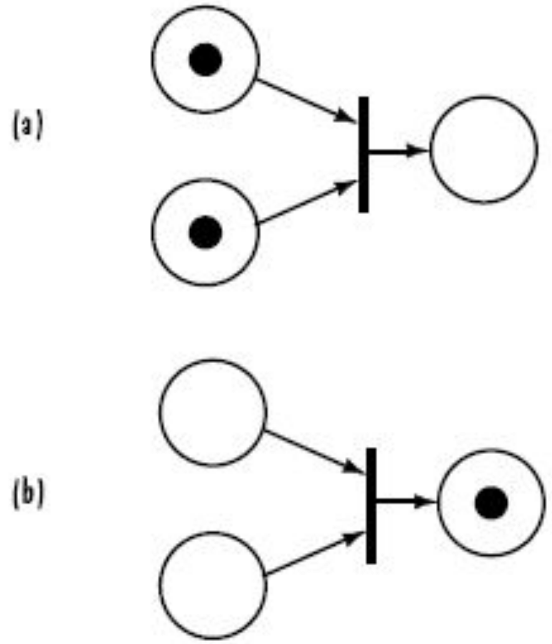
- A form of state-transition notation that is used to model concurrent activities and their interaction
  - Circles (***places***) represent activities or conditions
  - Bars represents ***transitions***
  - ***Arcs*** connect a transition with its input places and its output places
  - The places are populated with ***tokens**,* which act as enabling conditions for the transitions
  - Each arc can be assigned *a* ***weight*** that specifies how many tokens are removed from arc's input place, when the transition fires

Petri Nets

- A transition is *enabled* if each of its input places contains enough tokens to contribute its arc's weight's worth of tokens, should the enabled transition actually fire.



Three types of transitions

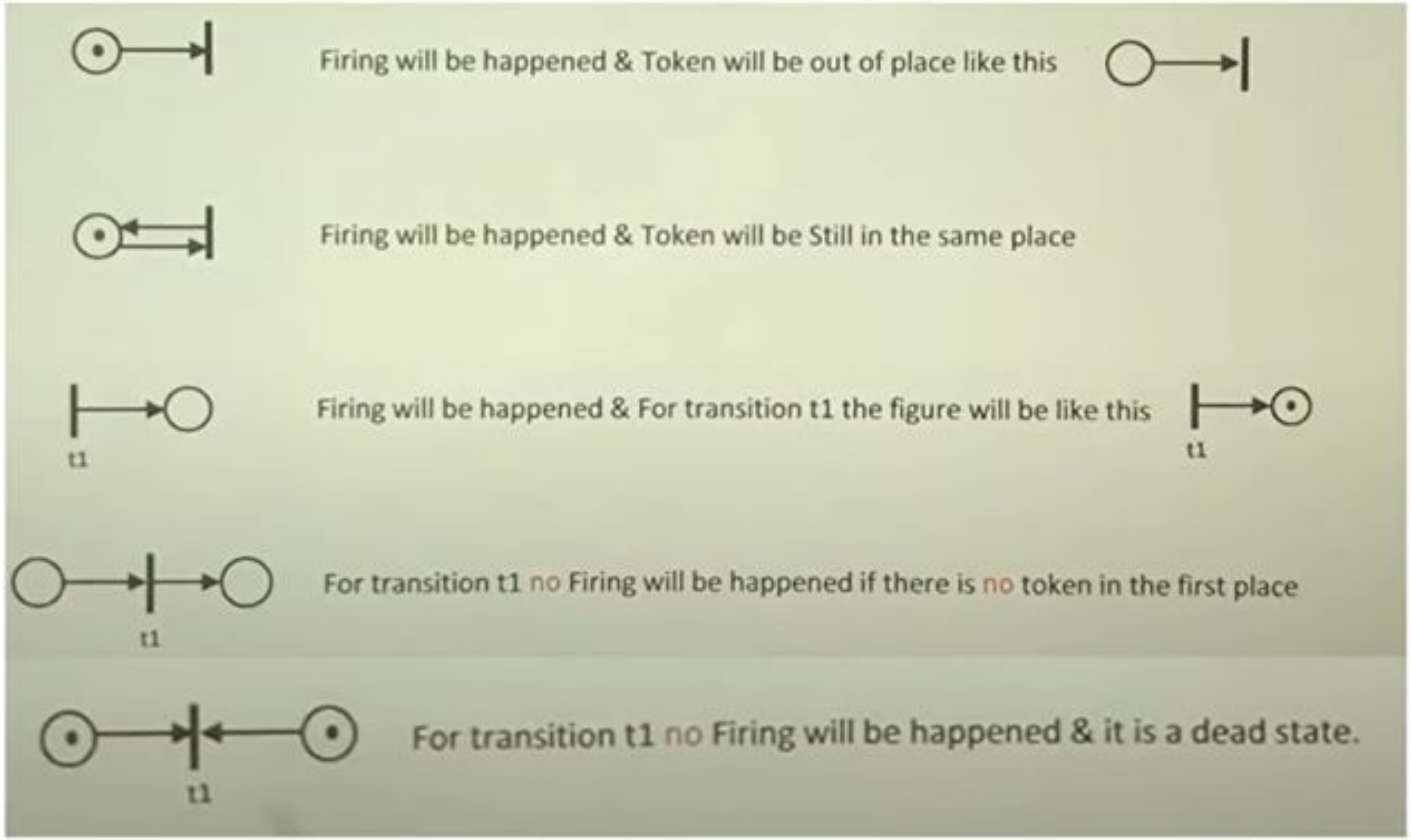


Tokens

Petri Nets

- Concurrent entities are **synchronized** whenever their activities, or places, act as input places to the same transition.
- This synchronization ensures that all of the pre-transition activities occur before the transition fires but does not constrain the **order** in which these activities occur.
- These features of concurrency and synchronization are especially useful for modeling events whose order of occurrence is not important.

Petri Nets



Firing will be happened & Token will be out of place like this

Firing will be happened & Token will be Still in the same place

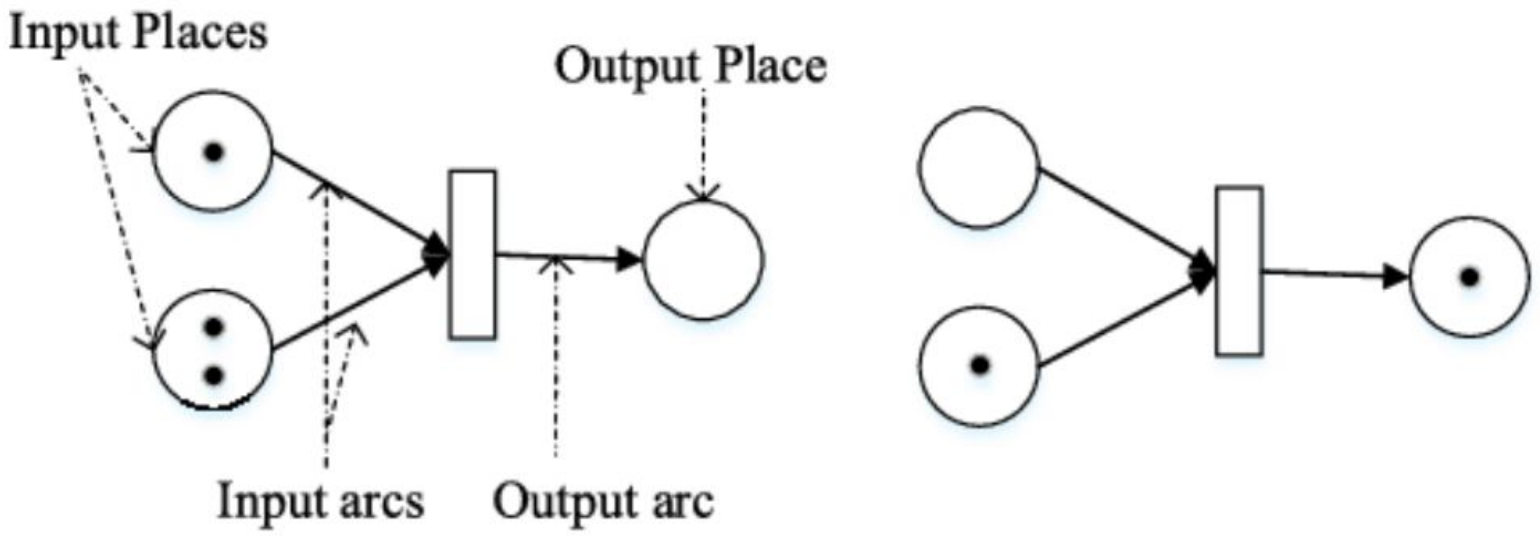
Firing will be happened & For transition t1 the figure will be like this

For transition t1 no Firing will be happened if there is no token in the first place

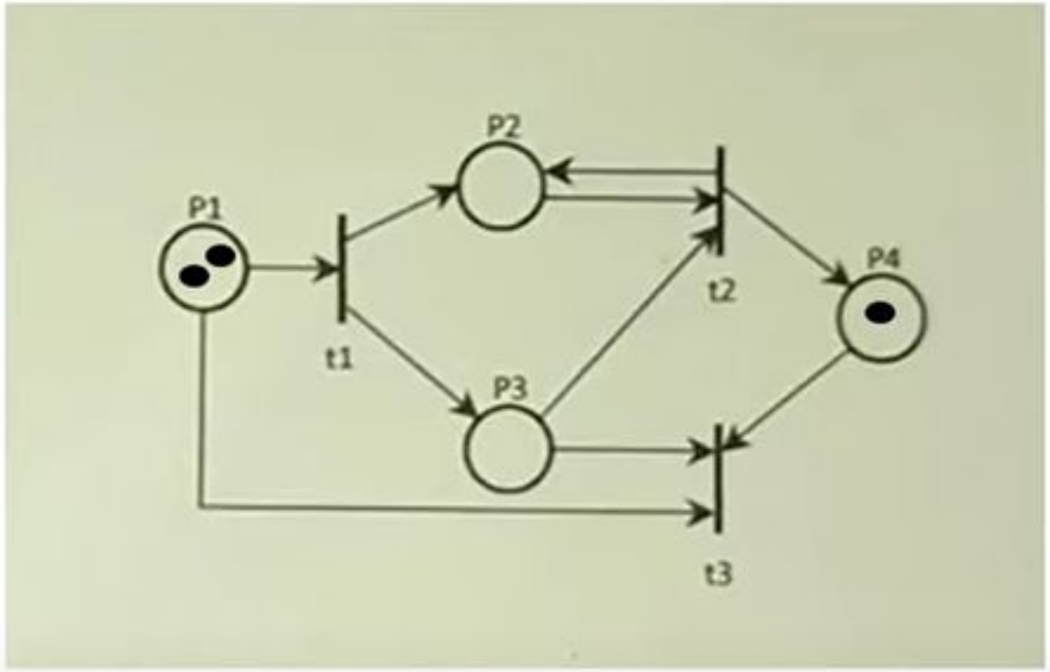
For transition t1 no Firing will be happened & it is a dead state.

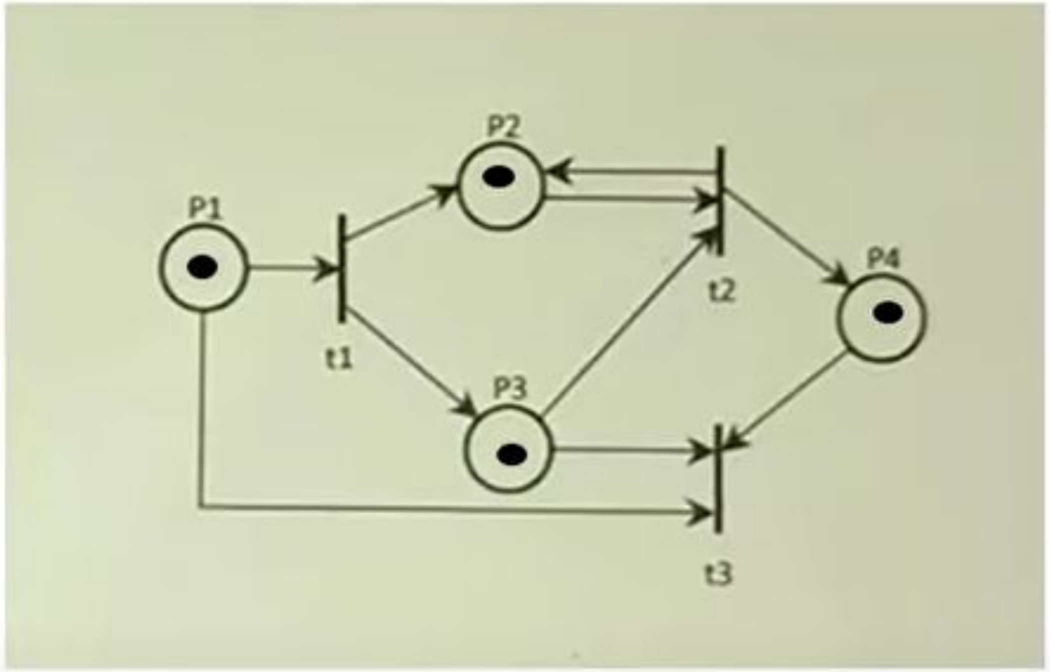
Petri Nets

- A transition with multiple input places requires tokens in all of its input places before it can fire and produce a token in an output place.



Petri Nets (Example)

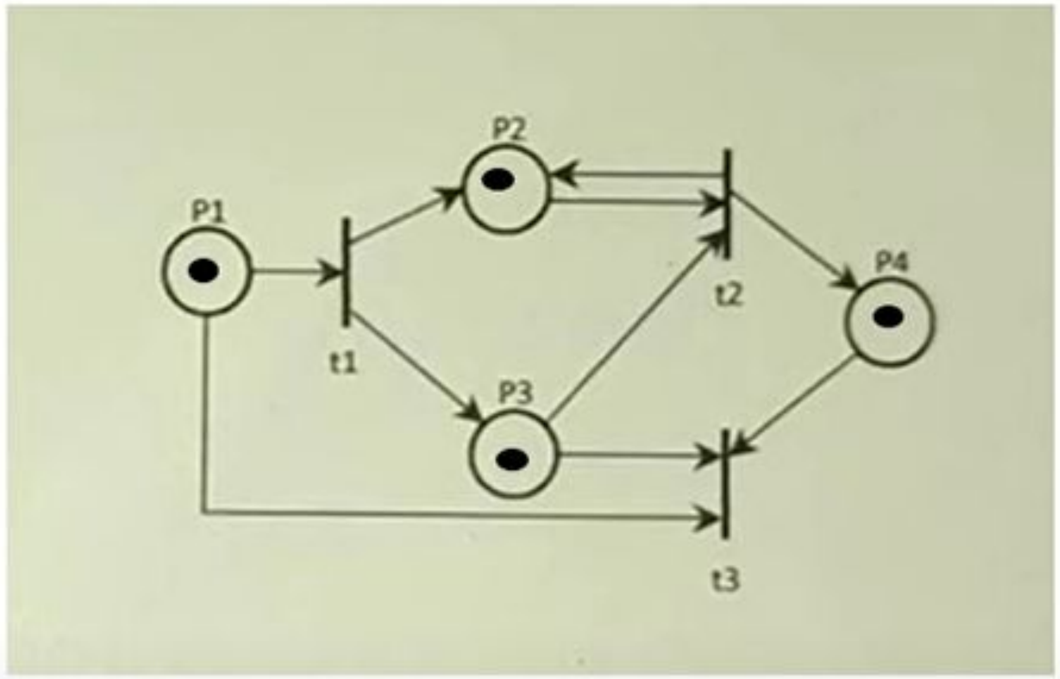


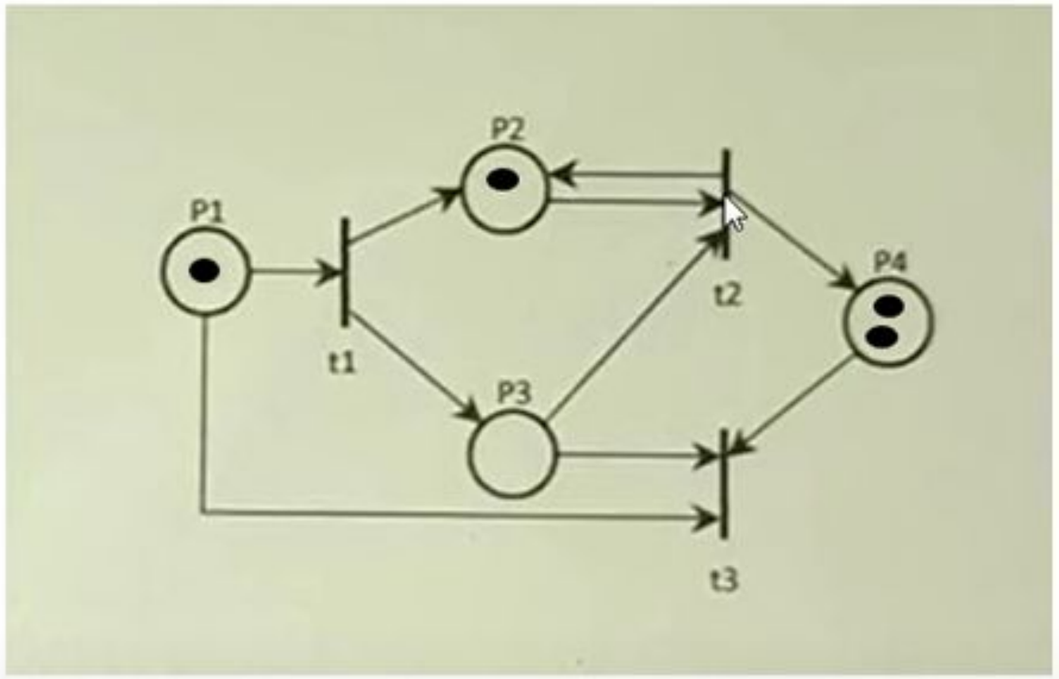


X0 = [2 , 0 , 0 , 1]

X1 = [1 , 1 , 1 , 1]

Petri Nets (Example)

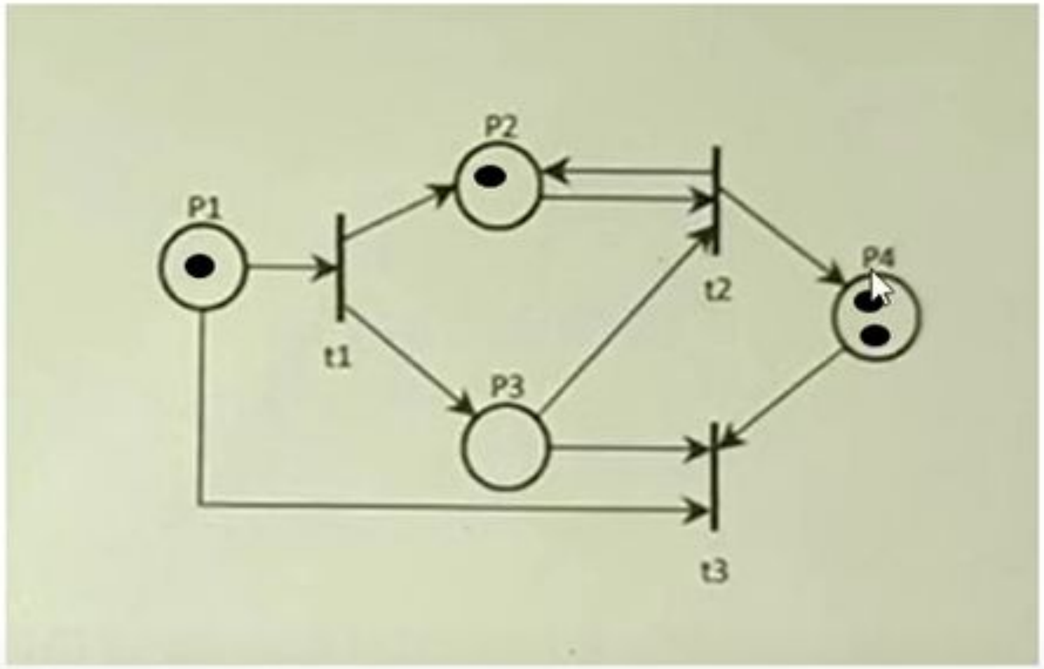


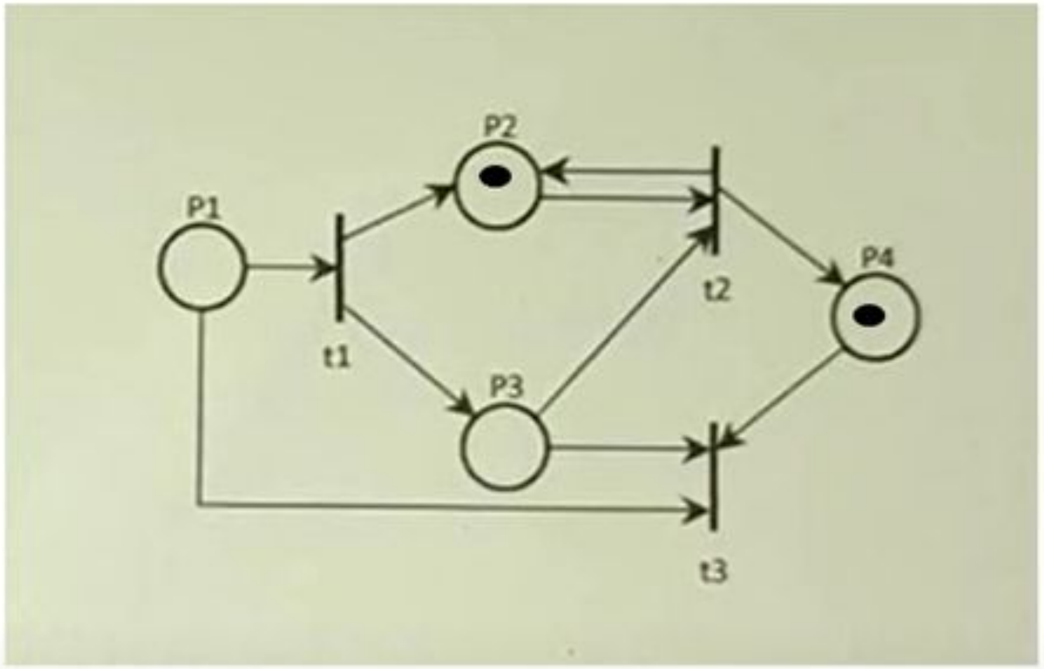


X1 = [1 , 1 , 1 , 1]

X2 = [1 , 1 , 0 , 2]

Petri Nets (Example)

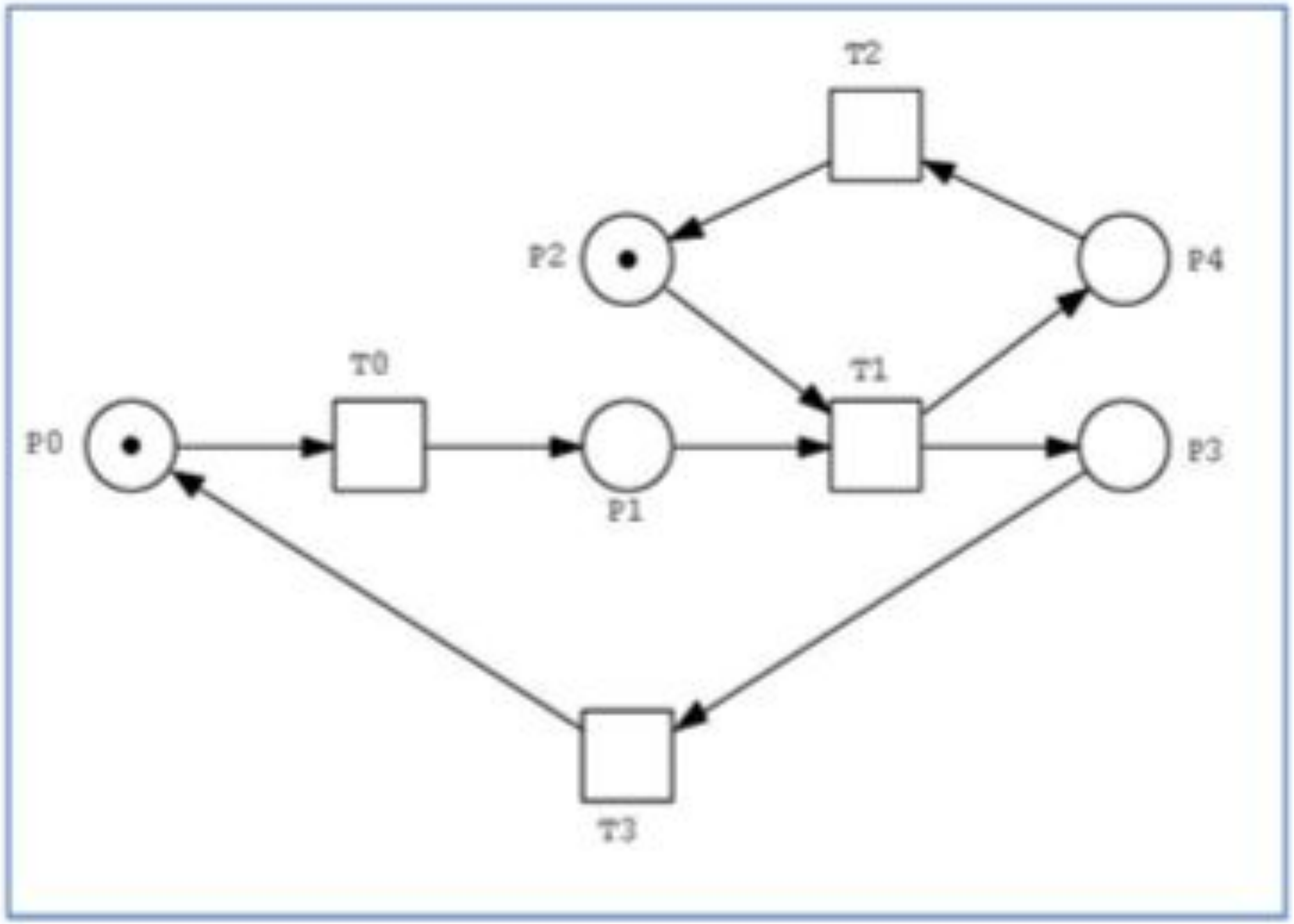




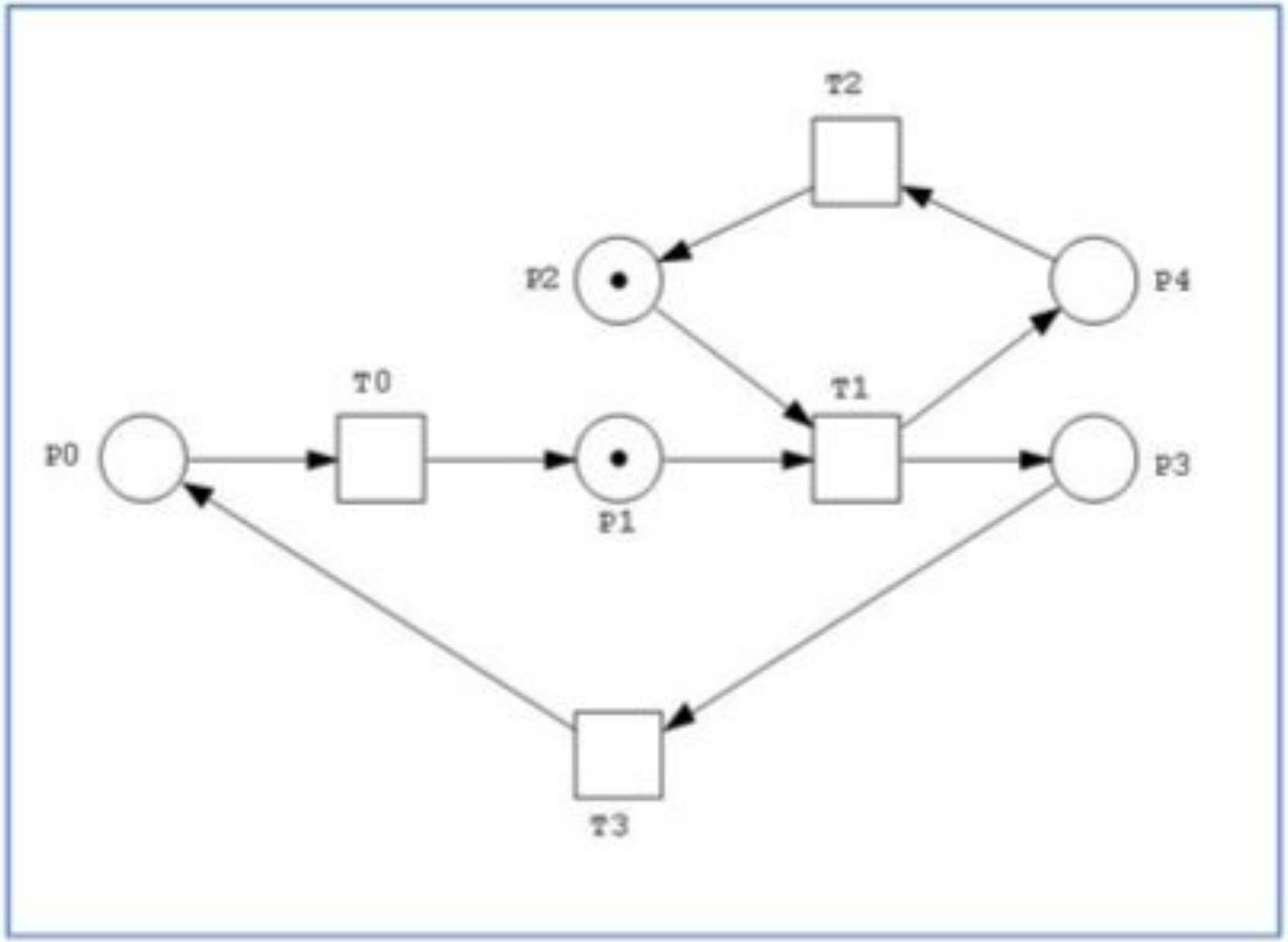
X2 = [1 , 1 , 0 , 2]

X3 = [0 , 1 , 0 , 1]

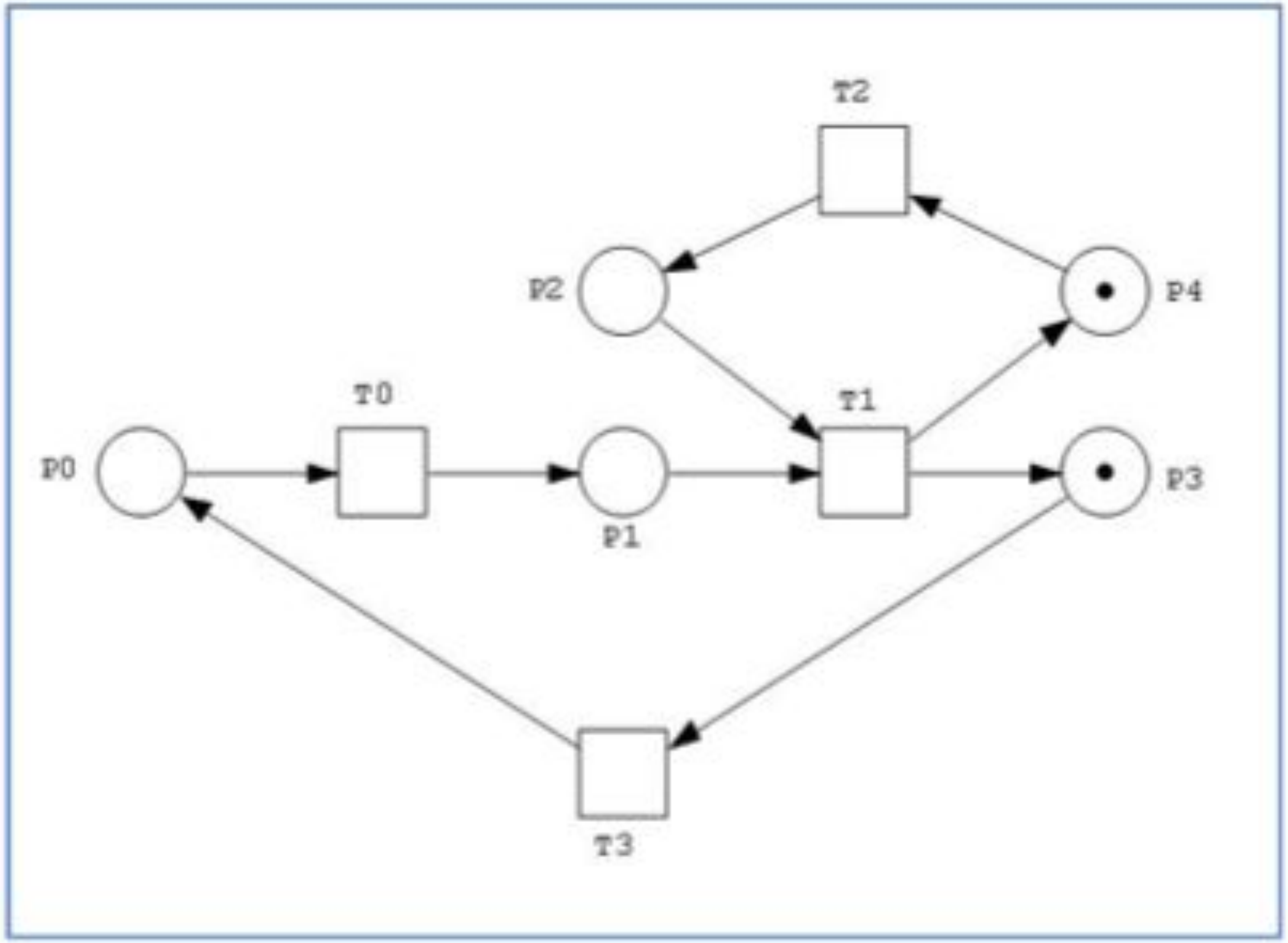
Example



Example

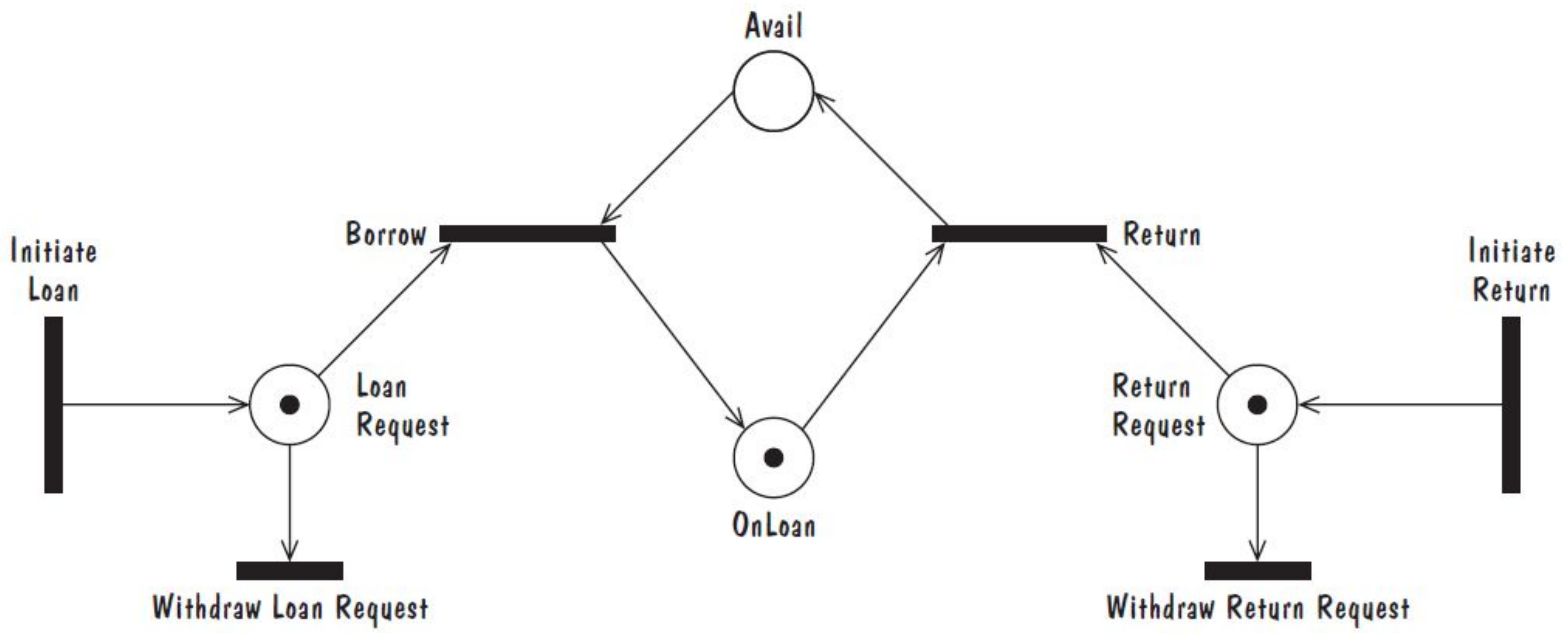


Example



Petri Nets

Petri net of book loan



References

1. Roger S. Pressman, Software Engineering A Practitioner's Approach, 9th Edition. McGrawHill
2. Shari PFleeger, Joanne Atlee, Software Engineering: Theory and Practice, 4th Edition
3. Roger S. Pressman, Software Engineering A Practitioner's Approach, 5th Edition. McGrawHill