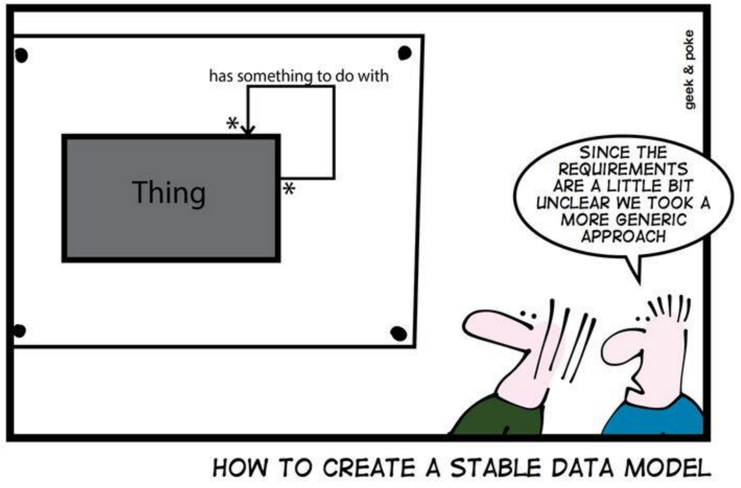
University
Mohammed VI
Polytechnic


has something to do with
*
*
Thing
geek & poke
SINCE THE REQUIREMENTS ARE A LITTLE BIT UNCLEAR WE TOOK A MORE GENERIC APPROACH
HOW TO CREATE A STABLE DATA MODEL

Introduction To UML – Part 2

Pr. Imane Fouad

Introduction

Classes represent the **static structure** of a system.

Interactions show **how instances** of these classes work together to perform a function.

Interactions illustrate the **dynamic behavior** of the system.

Objectives of Interactions

Identify **which classes** interact for a given **use case**.

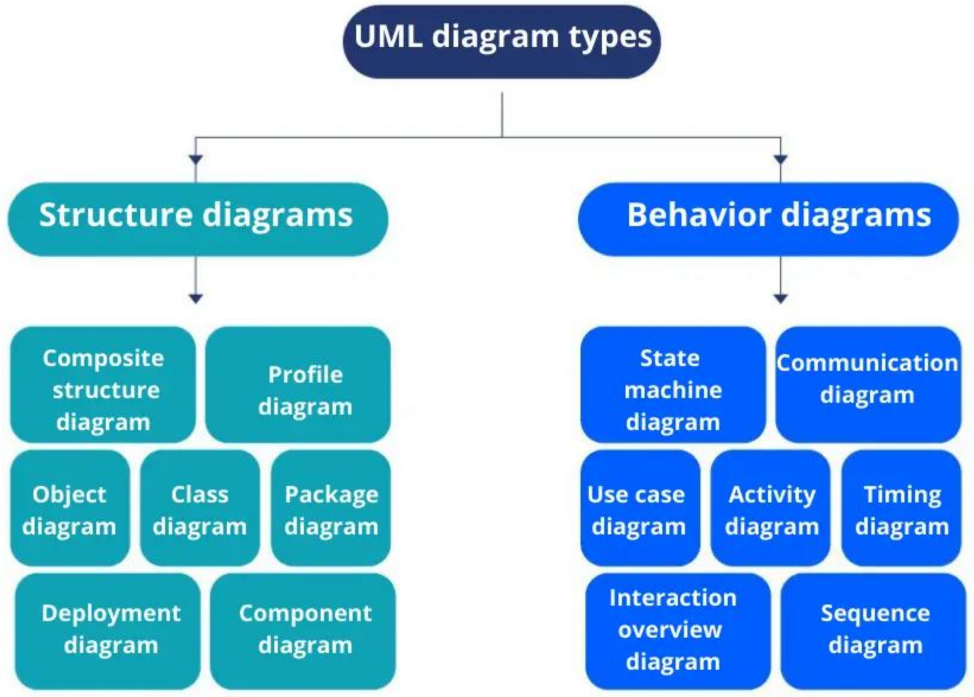
Determine the **messages** exchanged between classes to achieve a specific behavior.

Update the requirements and analysis models if needed.

Do **not** create interactions for every use case, focus only on the **most important** and **complex** ones.

Behavioral Diagrams

- **Behavioral diagrams** are the best way to represent **behaviors and interactions** within a system.
- A **sequence diagram** shows an **ordered sequence of messages** over time.
- An **activity diagram** describes the **flow of actions** that lead to the completion of a function.



Sequence Diagrams

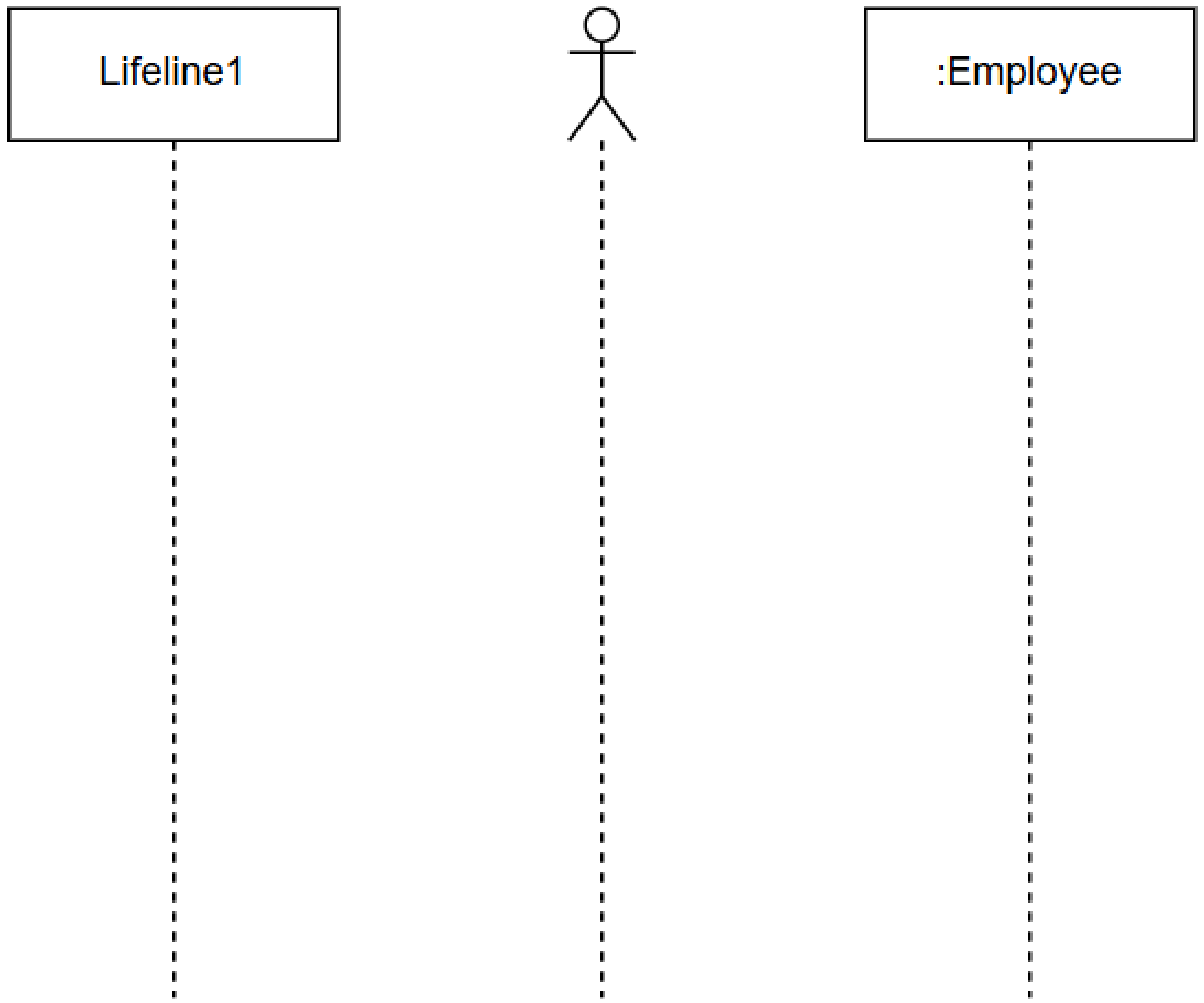
Sequence Diagrams

- **Sequence diagrams (SDs)** describe an action that unfolds over time.
- SDs document use cases and are part of the analysis model.
- SDs are made up of **three main elements**:
  - **Lifelines**: represent objects or actors involved.
  - **Messages**: show communication between lifelines.
  - **Fragments**: illustrate conditions or control structures.

Lifelines

- A **lifeline** represents a **single participant** in an interaction.
- A lifeline can represent an **instance of a class**, or an **actor**.
- To represent the interaction, **messages connect** the different lifelines.

Lifelines – UML representation



instanceName : ClassName

Messages

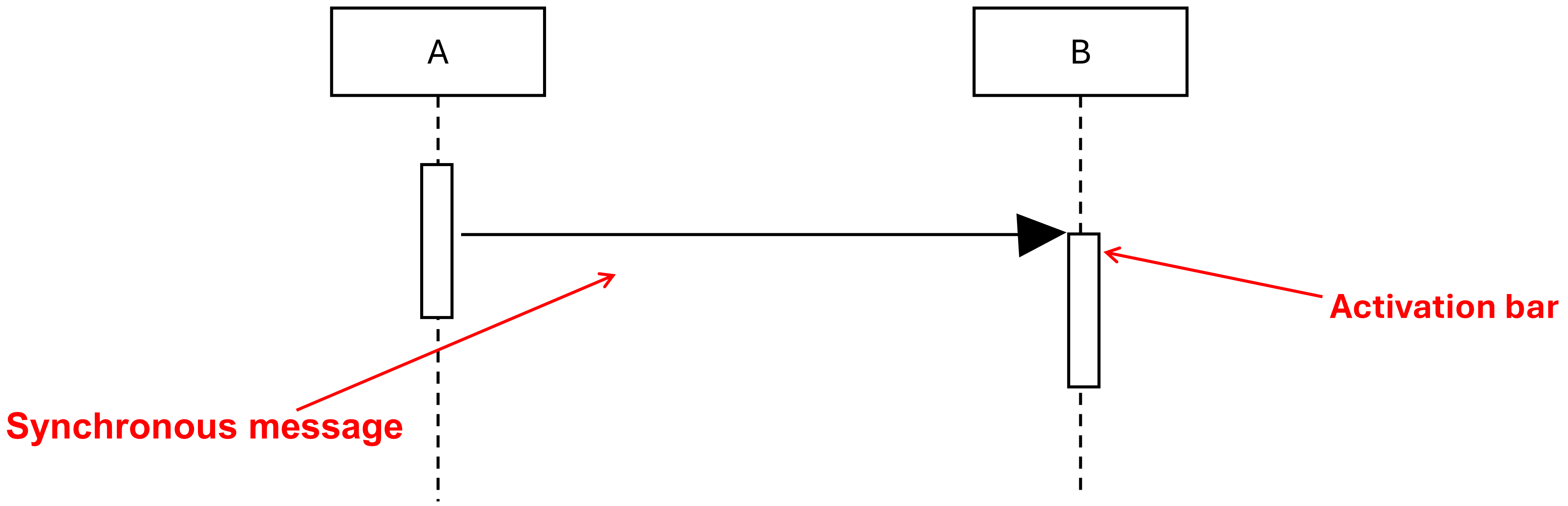
- A message represents **communication** between two lifelines during an interaction.
- A message can represent:
  - A **call to an operation**
  - The **creation** or **destruction** of an instance
- When a lifeline receives a message, it usually corresponds to calling an operation with the same signature.
- When a lifeline receives a message, it becomes active, this is shown by an **activation bar.**

Types of Messages

Message Type	Description
Synchronous Message	The sender **waits** for the receiver to finish the operation before continuing.
Asynchronous Message	The sender **sends the message** and continues execution **without waiting** for the receiver to finish.
Return Message	The sender **regains activation** after having passed it to the receiver.
Create Message	The message **creates** the receiver instance.
Destroy Message	The message **destroys** the receiver instance.
Found Message	The **sender is outside** the current interaction. (The origin is unknown or external.)
Lost Message	The **receiver never gets** the message — often used to represent **error or communication loss**.

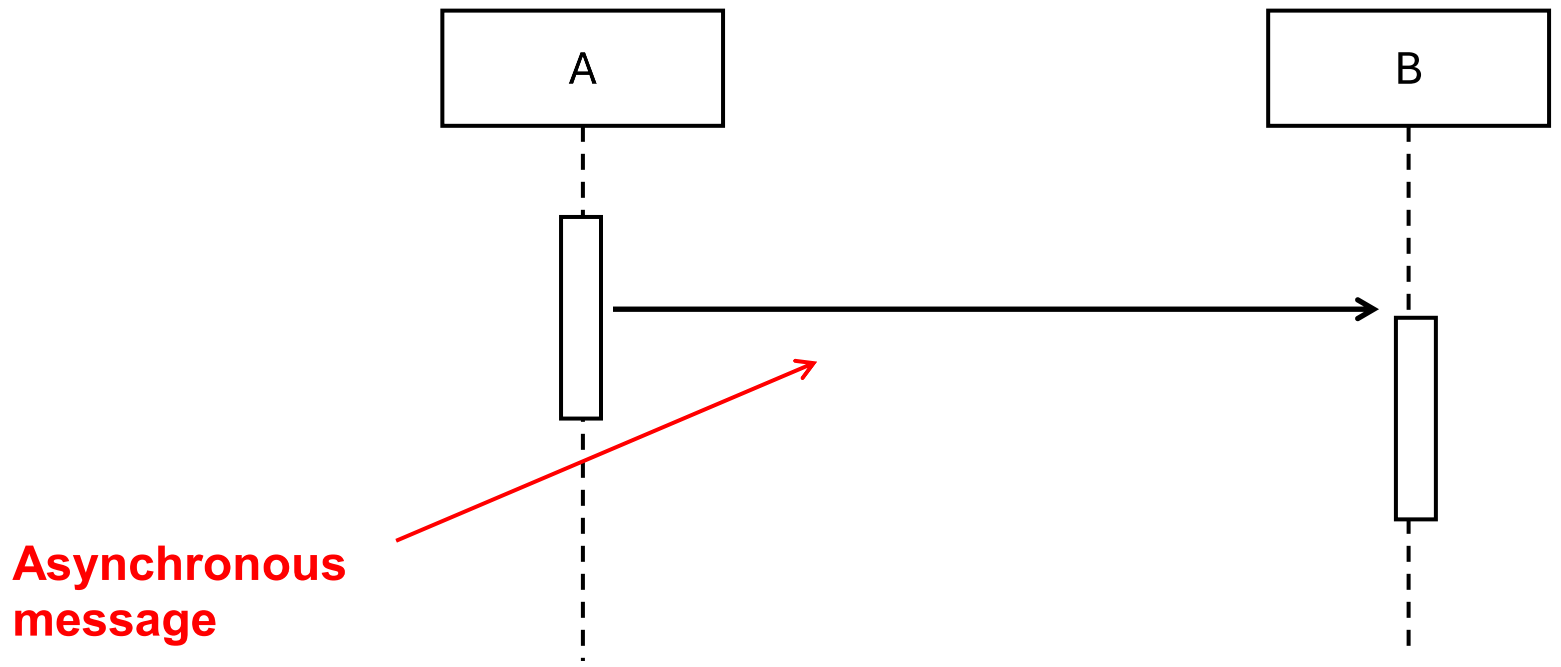
Synchronous Messages

- A **synchronous message blocks the sender** until the receiver completes its operation.
- The control flow passes from the sender to the receiver.
  - Example: If object **A** calls a method on object **B**, **A waits** until **B finishes** executing the method.



Asynchronous Messages

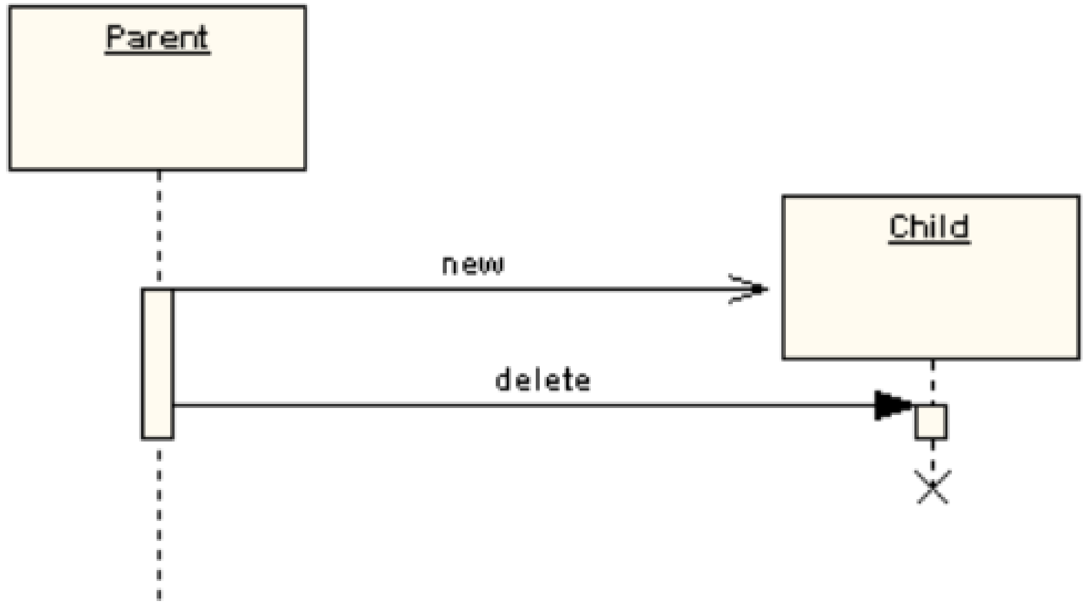
- An **asynchronous message** does **not block the sender**.
- The sender can continue executing immediately after sending the message.
- The receiver may **process the message later** or even **ignore it**.
- Typically used for events, or messages where the result is not needed immediately.



Creation & Destruction Messages

- **Create Message**: Creates a new instance of a class. Represented by a **message arrow pointing to the top of the lifeline** of the new object

- **Destroy Message**: Destroys an existing instance. Represented by a **message arrow pointing to the lifeline**, which ends with an **X**.

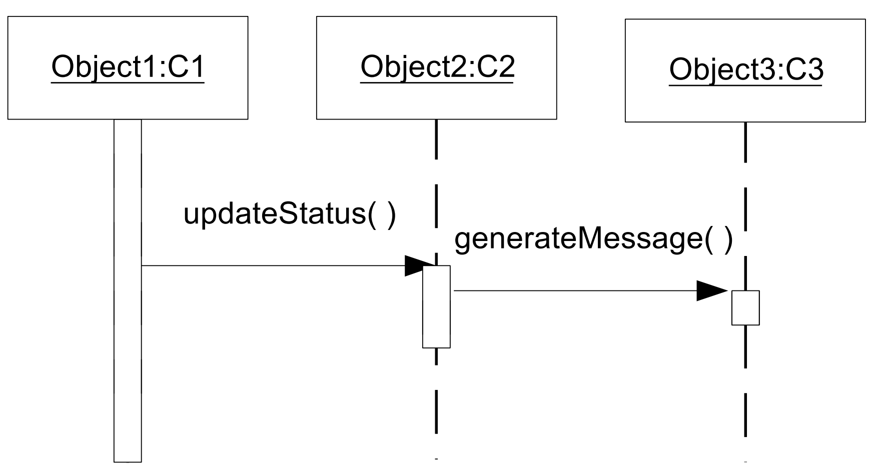


Return Messages

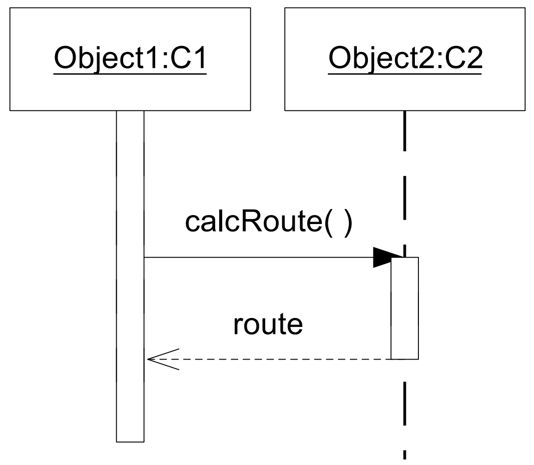
- The **receiver of a synchronous message** returns control to the sender by sending a **return message**.
- **Return messages are optional**: the **end of the activation period** also indicates the end of method execution.
- Return messages are typically used to **specify the result** of the invoked method.
- For **asynchronous messages**, the return is done by sending **new asynchronous messages**, not an immediate return.

Return Messages

- Return messages are optional: the end of the activation period also indicates the end of method execution.



Implicit Returns

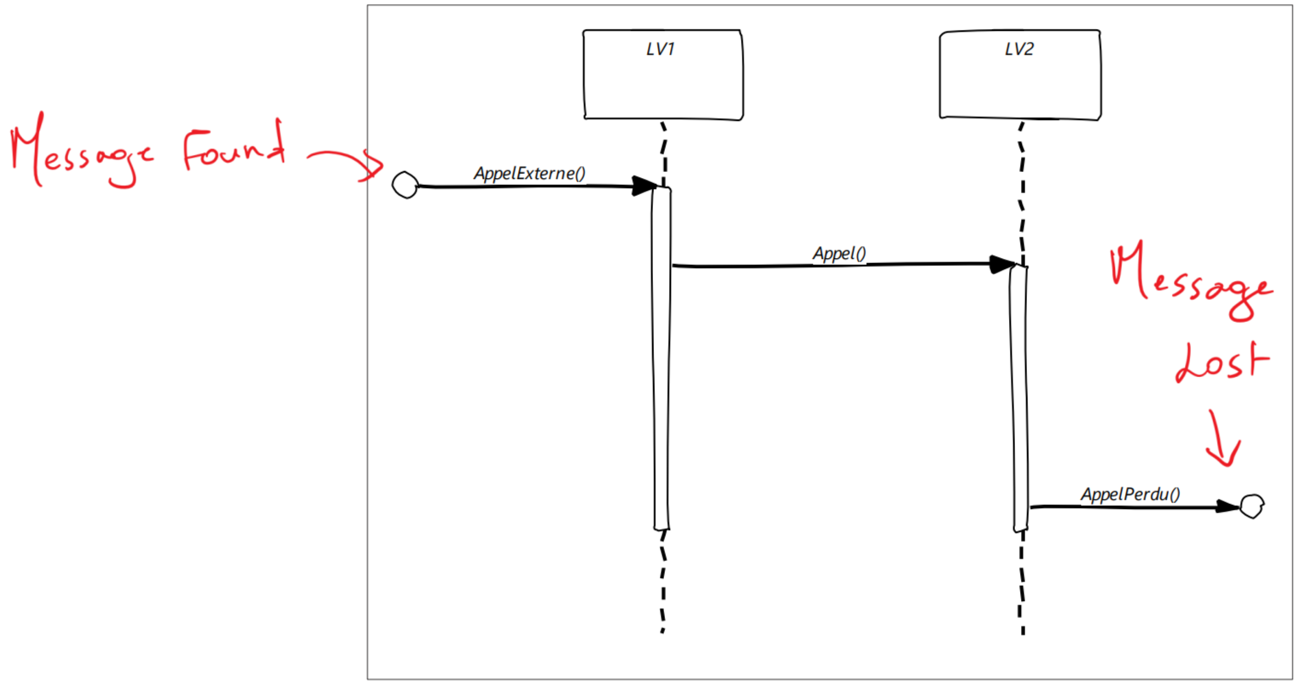


Explicit Return

Synchronous call

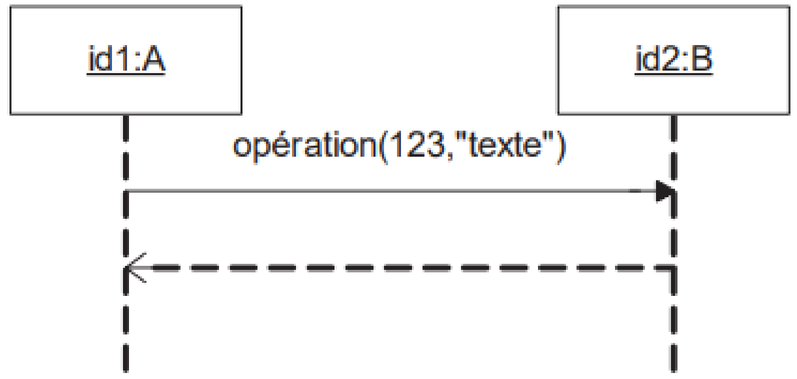
Complete, Lost, and Found Messages

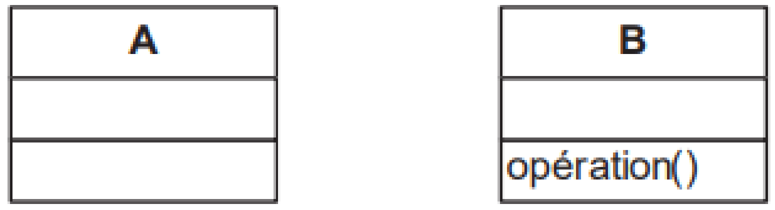
- **Complete Message**: Both **send and receive events are known**
    - Represented by an **arrow from one lifeline to another**
- **Lost Message**: **Send event is known**, receiver is unknown
    - Arrow ends on a **small circle**
- **Found Message**: **Receive event is known**, sender is unknown
    - Originates from **outside the interaction**



Links with the Structural View of the Model

- Every object participating in an interaction must have its **type defined as a class** in the structural view.
- Every **operation-call message** must target an **operation defined in the structural view**.
- The operation must belong to the **class of the object receiving the message**.
- Every **operation-call message** include **values for the operation's parameters**.

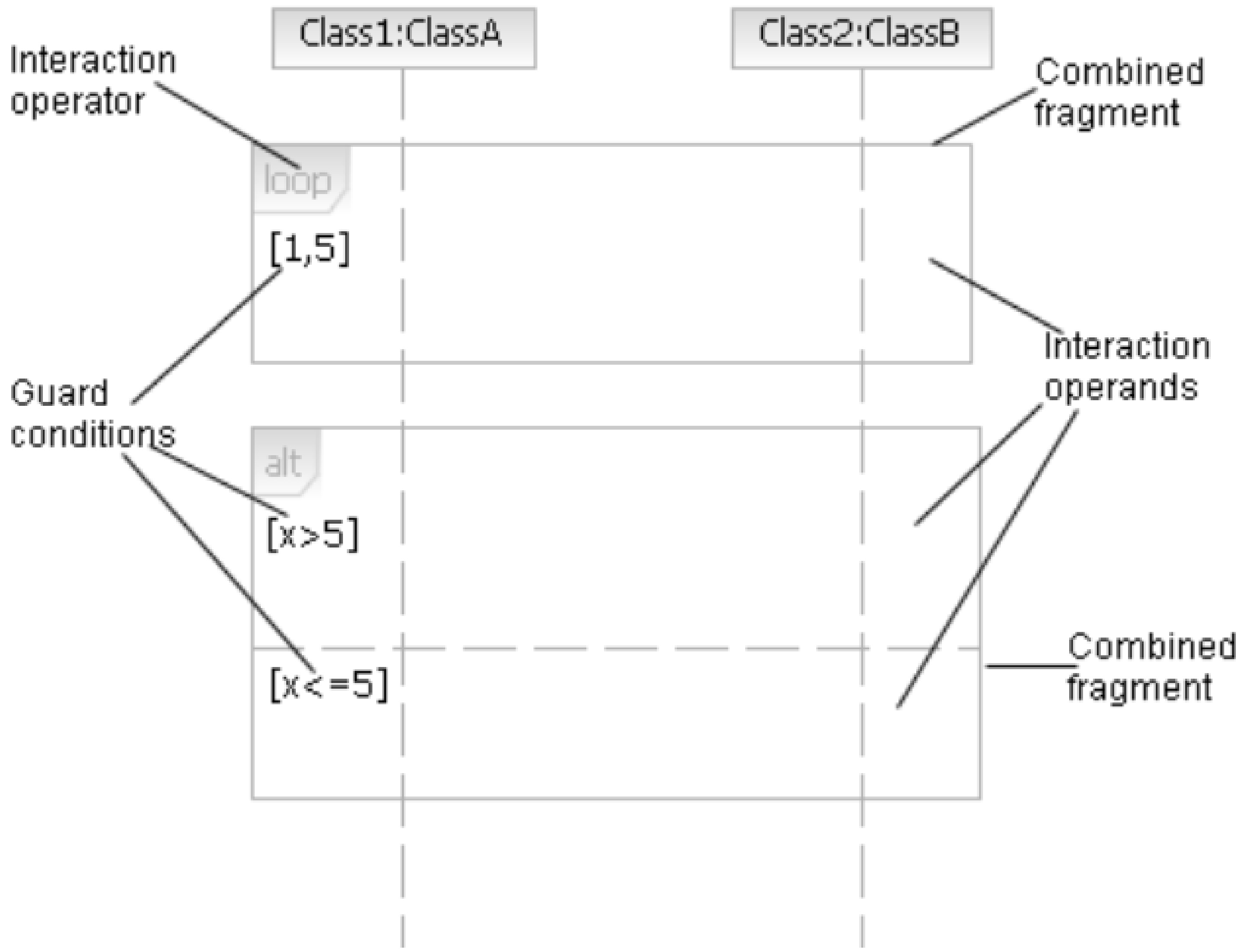




Fragments in Sequence Diagrams

- A sequence diagram can contain multiple fragments
- A fragment has a **name**
- Can contain one or **more** messages
- Composed of an operator
- Operator determines how the fragments are executed
- Can have one or more conditions
- Fragments can be nested

Fragments in Sequence Diagrams

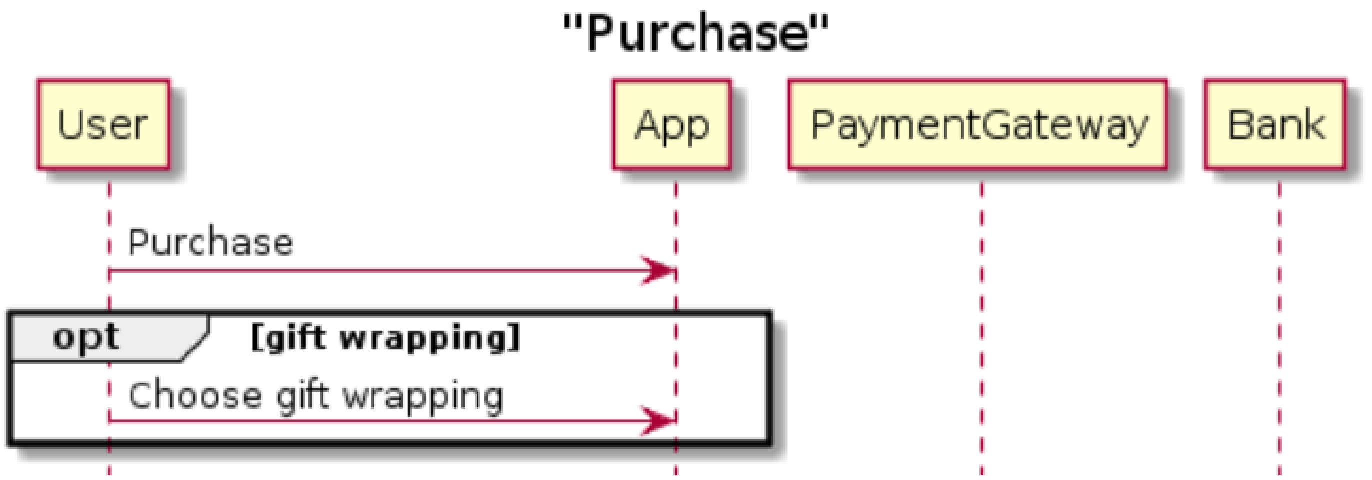


Fragment Operators

Operator	Name	Description
opt	Optional	The operand executes **only if the condition is true**
alt	Alternatives	Multiple alternatives; **only the operand whose condition is true executes**
loop	Iteration	Repeats execution **as long as the condition is true**
ref	Reference	Refers to **another interaction** (can reuse a sequence diagram)

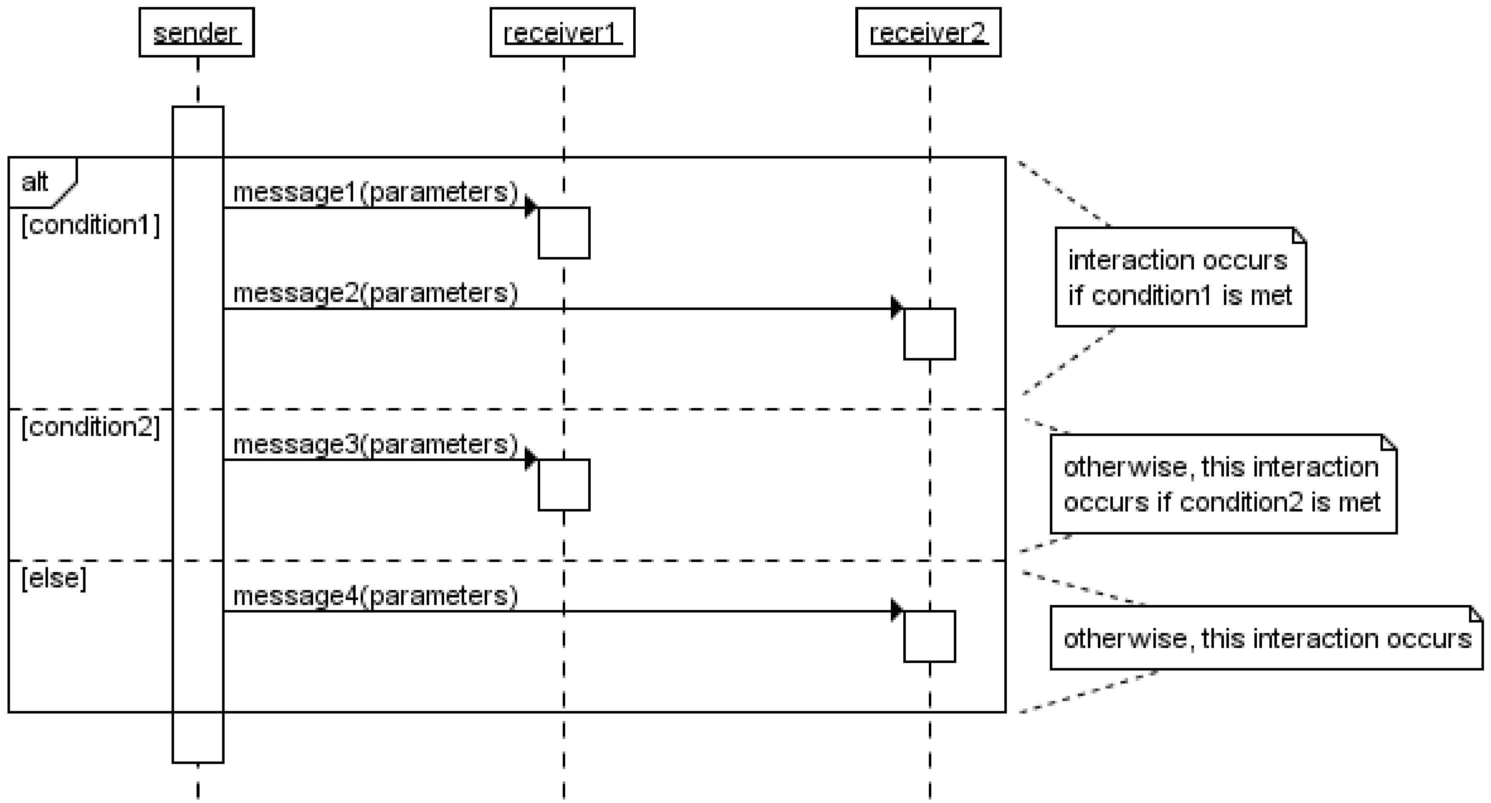
Optional Fragment

- Execute the operand **only if a condition is true**
- Contains Messages that run conditionally



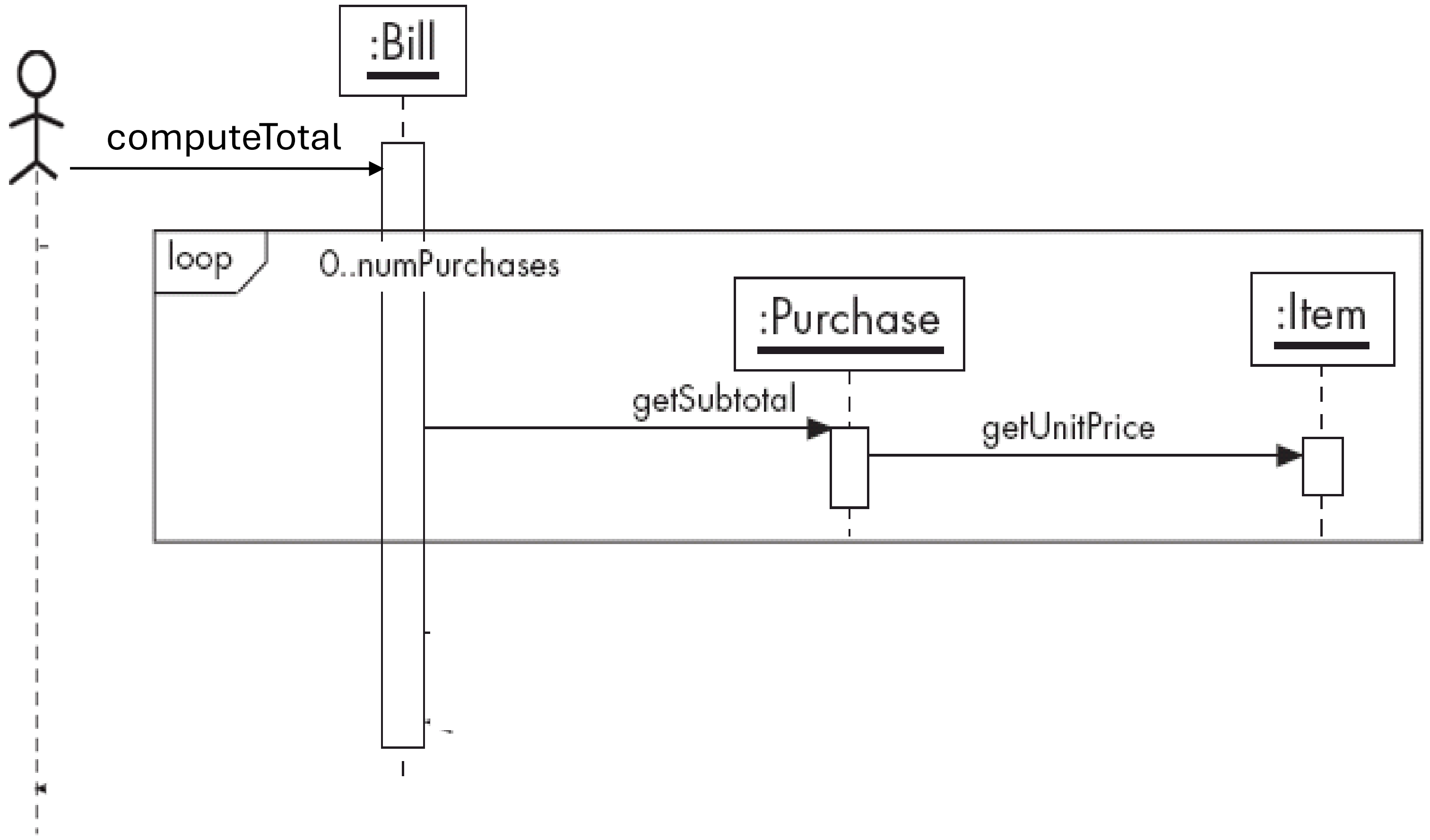
Alternative, optional interactions

- Represent **multiple alternative flows**
- Only the **operand whose condition is true** executes
- Notation: alt with operands for each condition



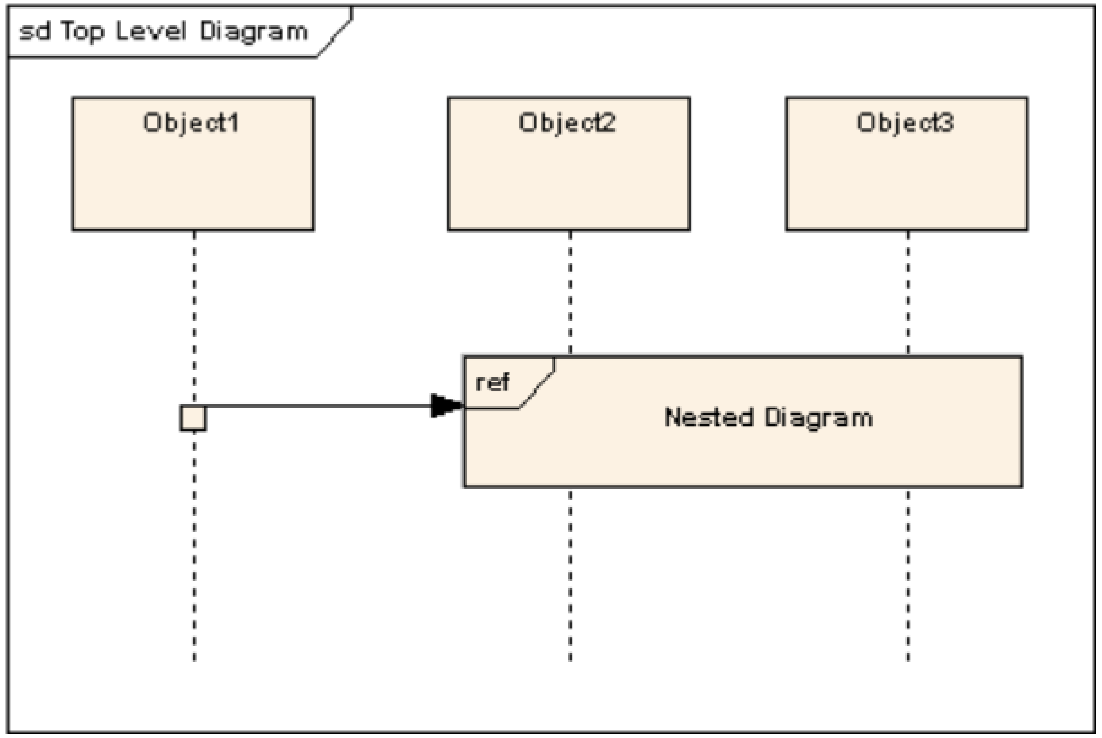
Iterative interactions

- loop (minNbIterations, maxNbIterations)
- The loop continues **while the condition is true**, up to **maxNbIterations** times.

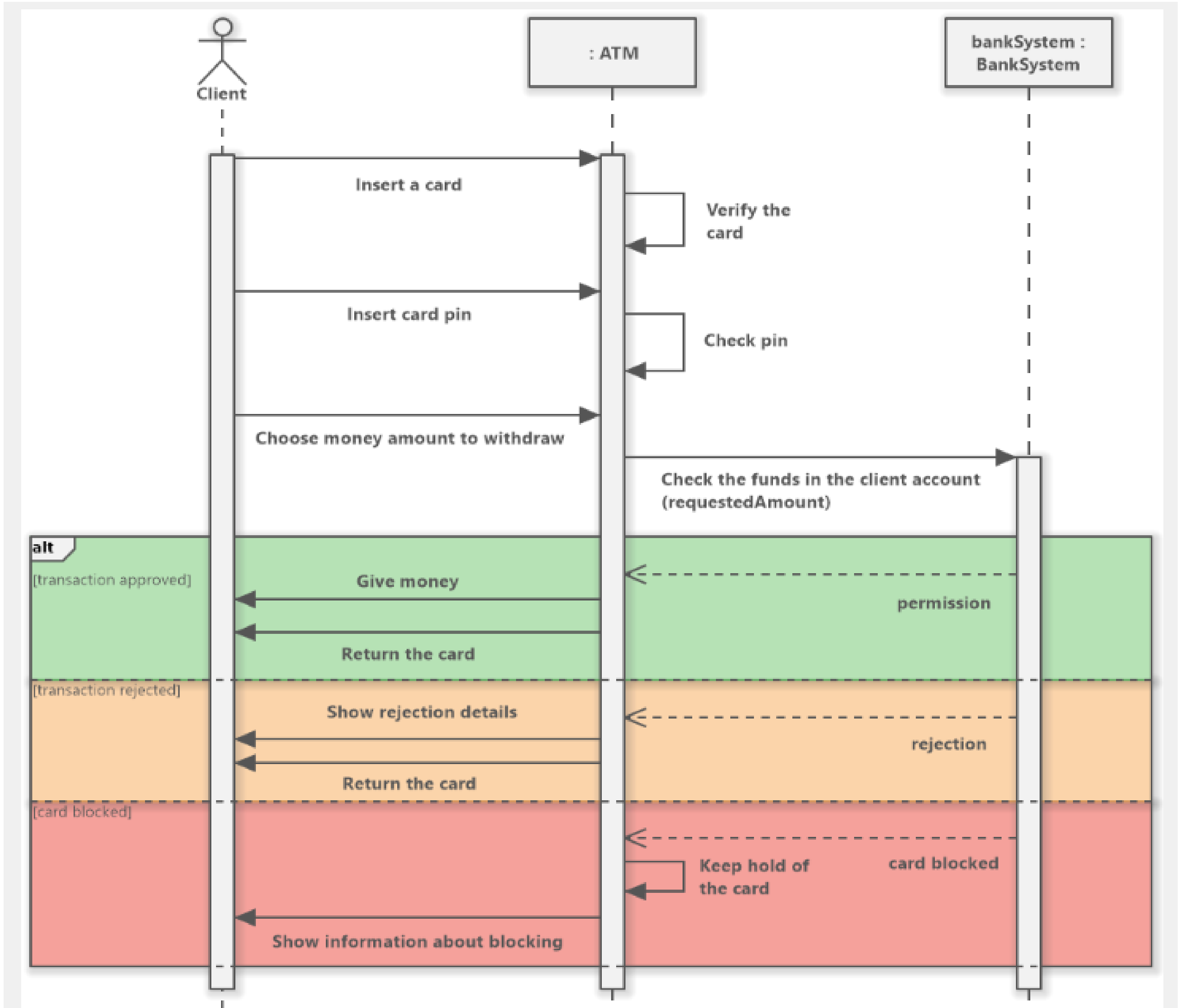


Reference Fragment

- Refers to **another interaction diagram** instead of duplicating messages
- Simplifying diagrams by modularizing complex interactions

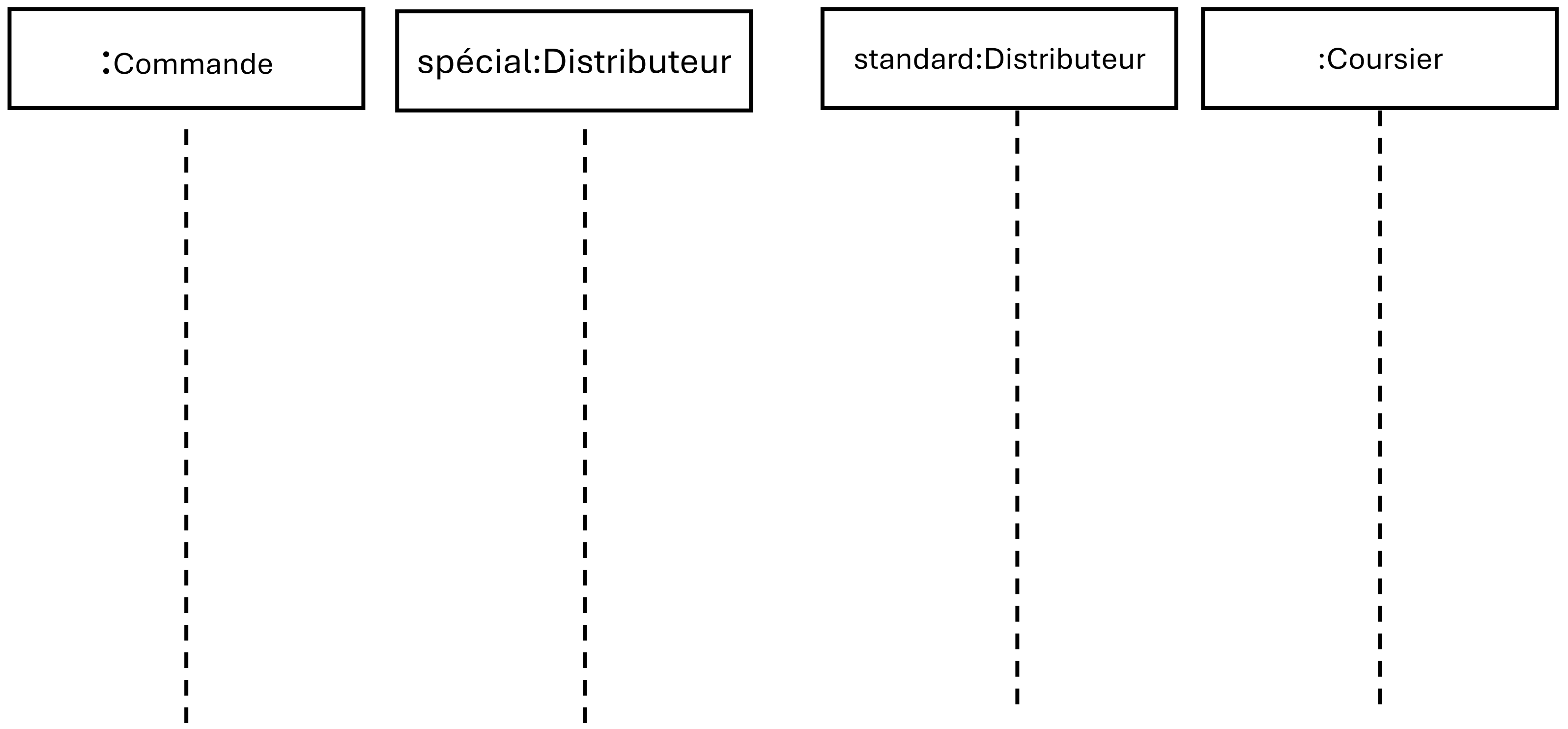


Exp


Client
: ATM
bankSystem : BankSystem
Insert a card
Verify the card
Insert card pin
Check pin
Choose money amount to withdraw
Check the funds in the client account (requestedAmount)
alt
[transaction approved]
Give money
permission
Return the card
[transaction rejected]
Show rejection details
rejection
Return the card
[card blocked]
Keep hold of the card
card blocked
Show information about blocking

Algorithm2SequenceDiagram

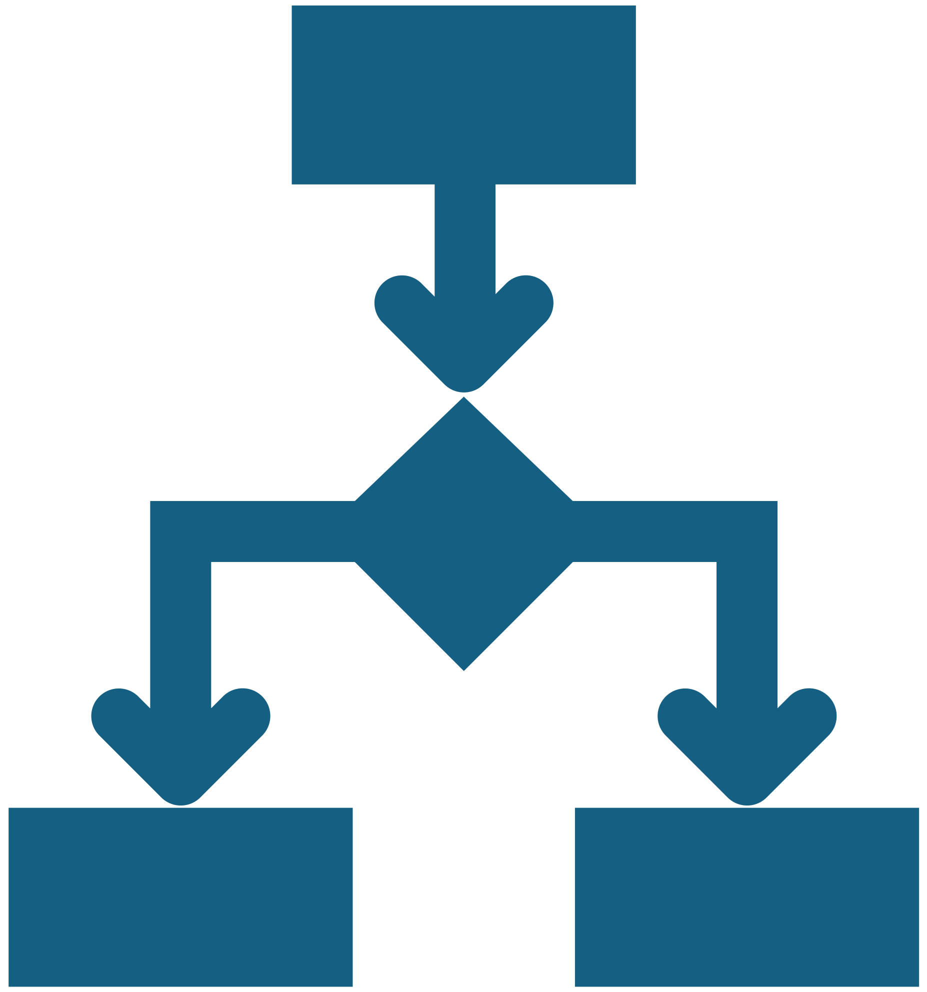
```
foreach (ligne)
  if (produit.valeur
               > $10000
    spécial.distribuer
  else
    standard.distribuer
  endif
end for
if (nécessiteConfirmation)
  coursier.confirmer
end procedure
```



Activity Diagrams

Activity Diagram

- Behavior Diagram
- Describes the behavior of a system or some components under the form of a stream/sequence of activities.
- Unlike sequence diagrams, which focus on object interactions, **ADs show the flow of actions**.



When to Use Activity Diagrams

- The main reason to use activity diagrams is to model the workflow behind the system being designed.

- Activity Diagrams are also useful for:
  - analyzing a use case by describing what actions need to take place and when they should occur
  - describing a complicated sequential algorithm

- Activity Diagrams should not take the place of interaction diagrams.

- Activity diagrams do not give detail about how objects behave or how objects collaborate.

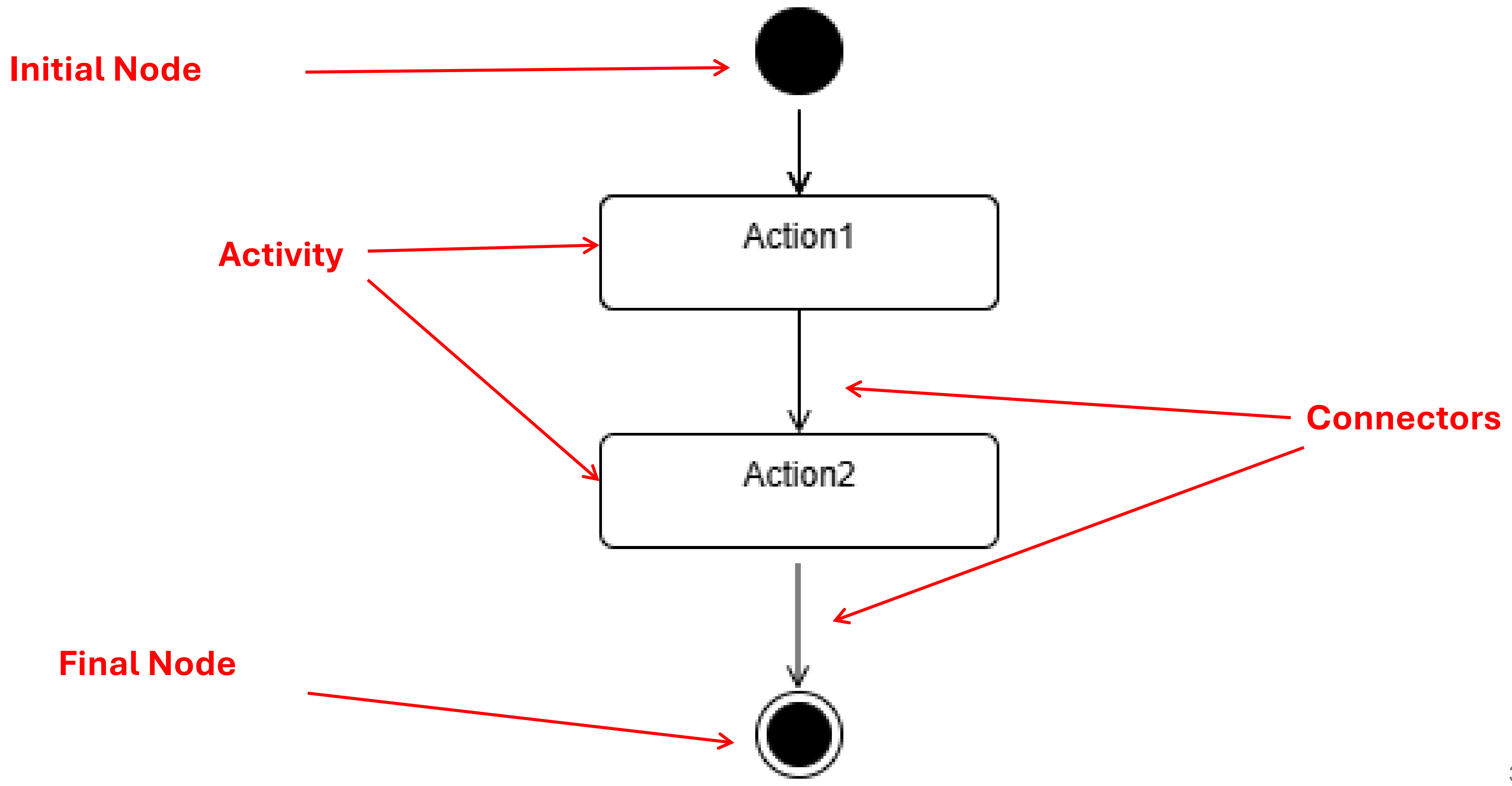
How Activity Diagrams Work

- The diagram follows a **sequence determined by a token**
- The **token moves** from the **initial node** to the **final node**
- A token can represent **the current flow, an object, or data**
- The **state of an activity diagram** is determined by its tokens
- Tokens move from one node to another **via connectors**

Components

- An **activity** is an execution of a step in a workflow (such as an operation or transaction)
    - Represented with a rounded rectangle.
    - Text in the activity box should represent an activity (verb phrase in present tense).
- An activity starts with a special node called the **Initial Node**.
- An activity ends with a special node called the **Final Node**.
- **Connectors** represent the flow between node

Activity Diagram



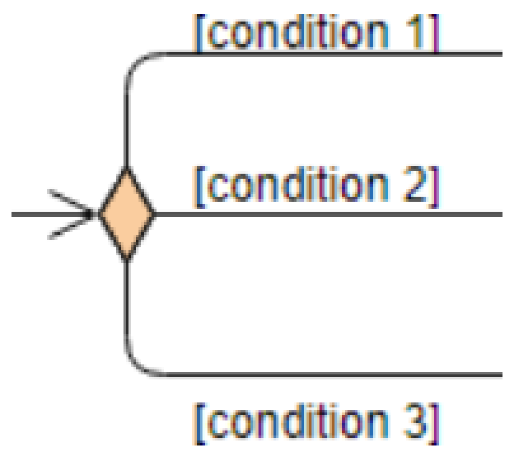
How to Draw an Activity Diagram

- Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.
  - A **fork** is used when multiple activities are occurring at the same time.
  - A **branch** describes what activities will take place based on a set of conditions.
  - All branches at some point are followed by a **merge** to indicate the end of the conditional behavior started by that branch.
  - After the merge all of the parallel activities must be combined by a **join** before transitioning into the final activity state.

Branch and Merge Nodes

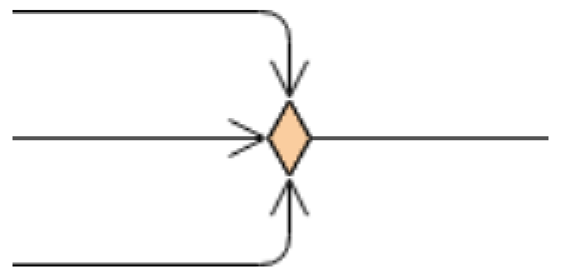
Decision Node

- Has one incoming connector and two or more outgoing connectors
- The token follows the path whose condition is true
- Represents branching in the process



Merge Node

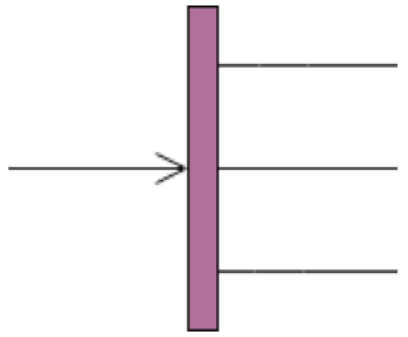
- A merge node brings together multiple alternate flows.
- A merge node has multiple incoming edges and a single outgoing edge.



Fork and Join Nodes

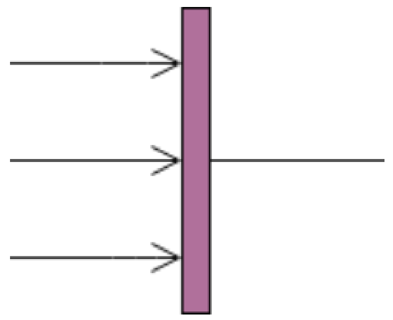
Fork Node

- Creates **parallel flows** in an activity diagram

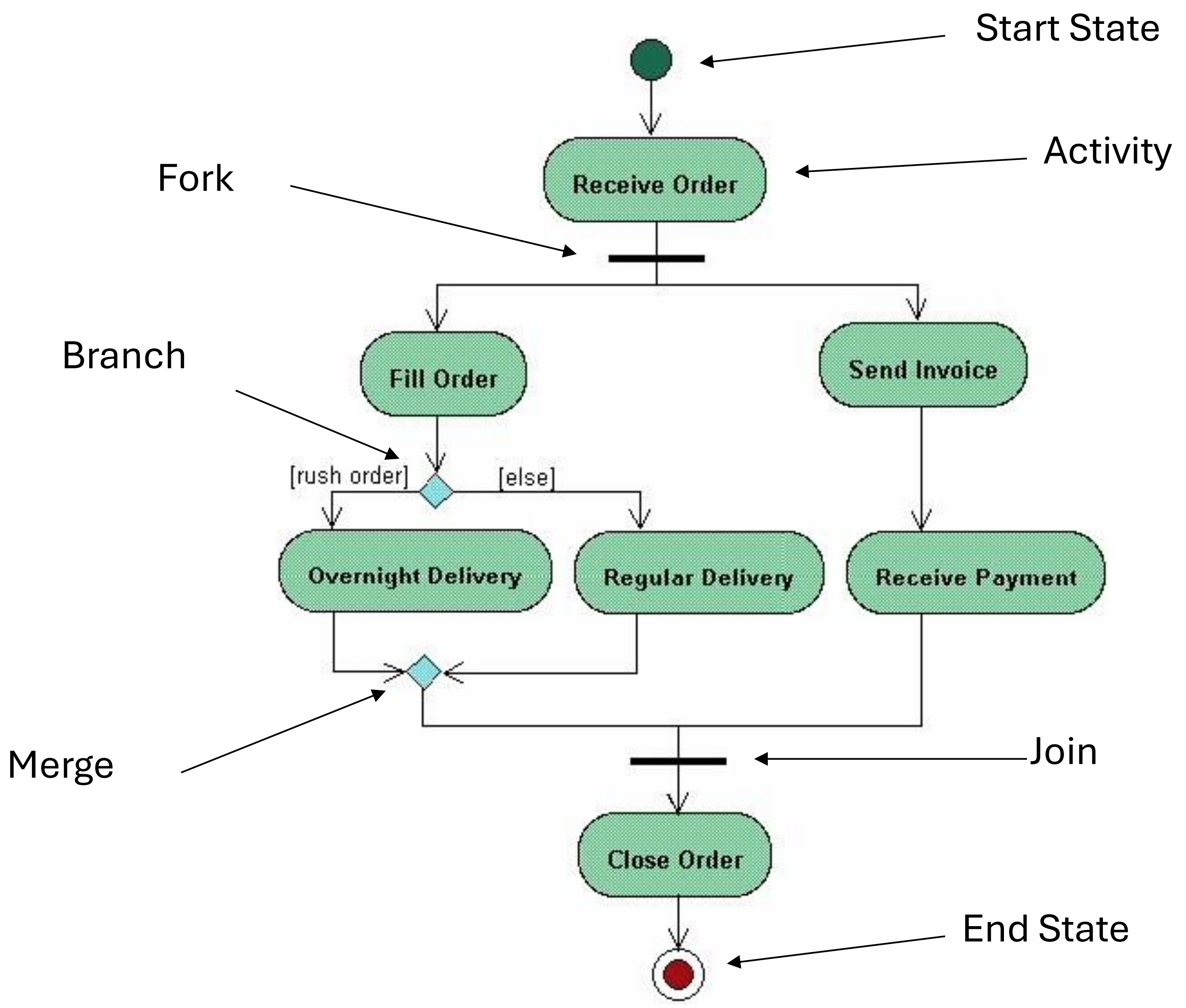


Join Node

- Merges **parallel flows** back into a single flow
- Waits until **all incoming flows complete** before proceeding

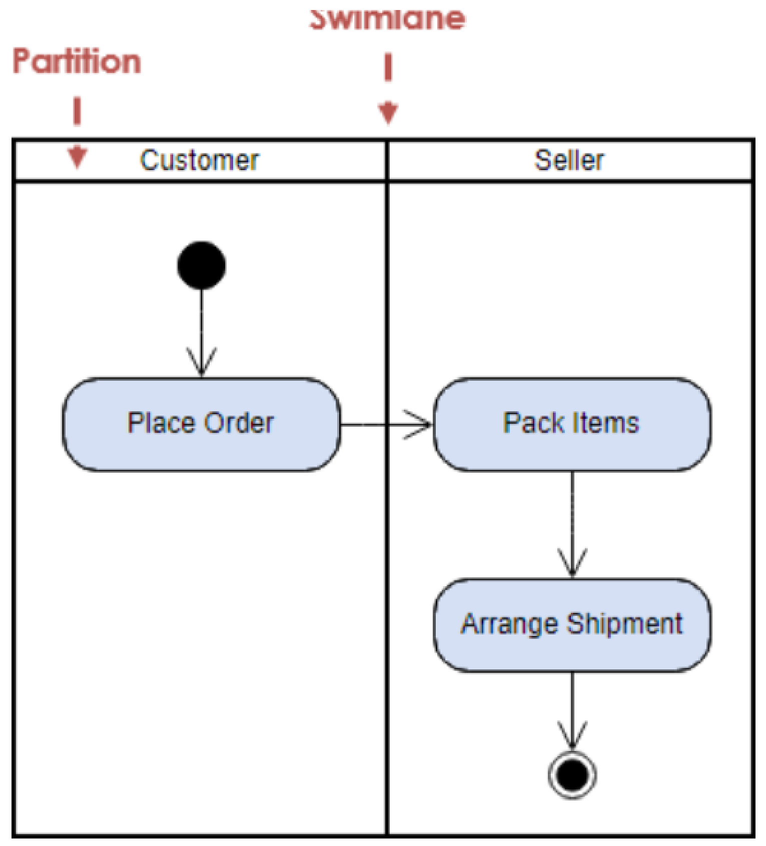


Activity Diagram Example

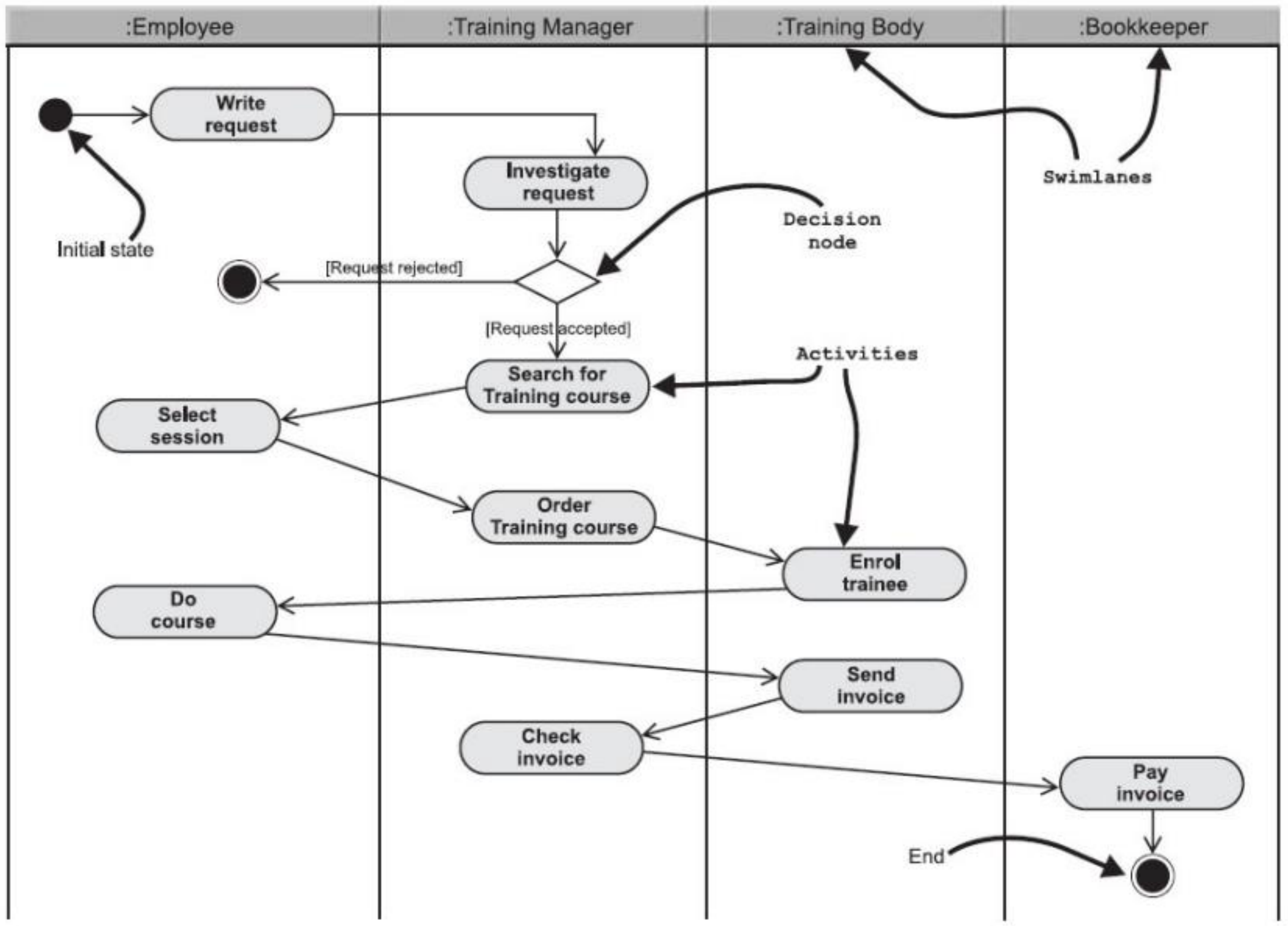


Partitions in Activity Diagrams

- **Semantically related activities** can be grouped into **partitions** (also called swimlanes)
- A **partition** usually represents the **role or actor performing the action**
- Partitions make activity diagrams **easier to read and more expressive**
- Partitions can be drawn **horizontally or vertically**



Activity diagram - Training process



Use Case

- Withdraw money from a bank account through an ATM