

UML Modeling

Modeling

- Describing a system at a high level of abstraction
 - A model of the system
 - Used for requirements and specifications
- Is it necessary to model software systems?

What is UML?

- UML stands for “Unified Modeling Language”
- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- The UML uses mostly graphical notations to express the OO analysis and design of software projects.
- Simplifies the complex process of software design

Why UML for Modeling

- Use graphical notation to communicate more clearly than natural language (imprecise) and code (too detailed).
- Help acquire an overall view of a system.
- UML is *not* dependent on any one language or technology.
- UML moves us from fragmentation to standardization.

Types of UML Diagrams

- **Use Case Diagram**
- **Class Diagram**
- **Sequence Diagram and Activity Diagrams**
- **Collaboration Diagram**
- **Deployment Diagram**
- **State Diagram**

Introduction to Use Case

- Getting started is the most difficulty part of any new process.
- In software modelling, the first thing you need to do is understand what are you going to model and ultimately develop.
- Creating a highest form details about a system--use case diagram--is an almost natural point of origin for the software design.
- A use case diagram is an excellent way to communicate to management, customers, and other non-development people what a system will do when it is completed.

Use Case Diagram

- Used for describing a set of user **scenarios**
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers

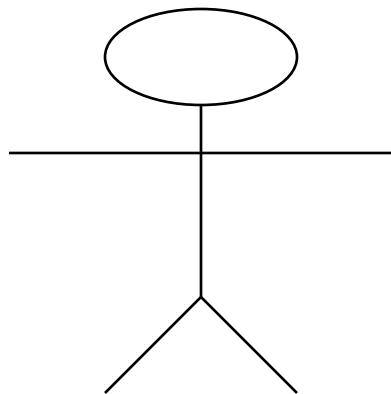
Use Case Diagrams

- A set of *ACTORS*: roles users can play in interacting with the system.
 - An actor is used to represent something that users our system.
- A set of *USE CASES*: each describes a possible kind of interaction between an actor and the system.
 - Uses cases are actions that a user takes on a system

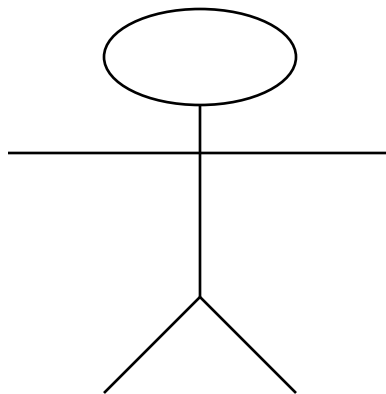
System boundary: rectangle diagram representing the boundary between the actors and the system.

Use Case Diagrams - Actors

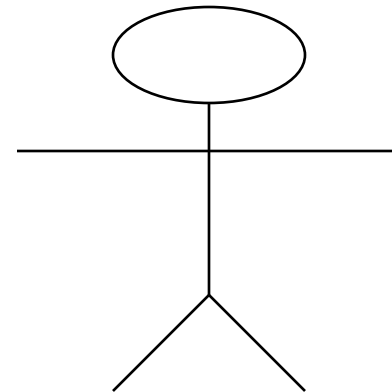
- An *actor* is a user of the system playing a particular role.
- Actor is shown with a stick figure.



employer



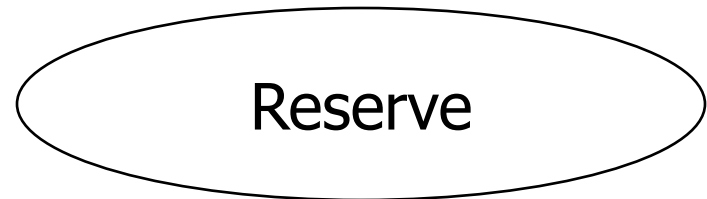
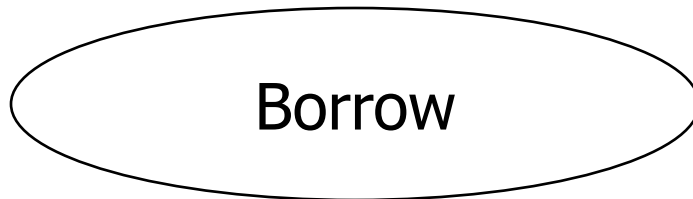
employee



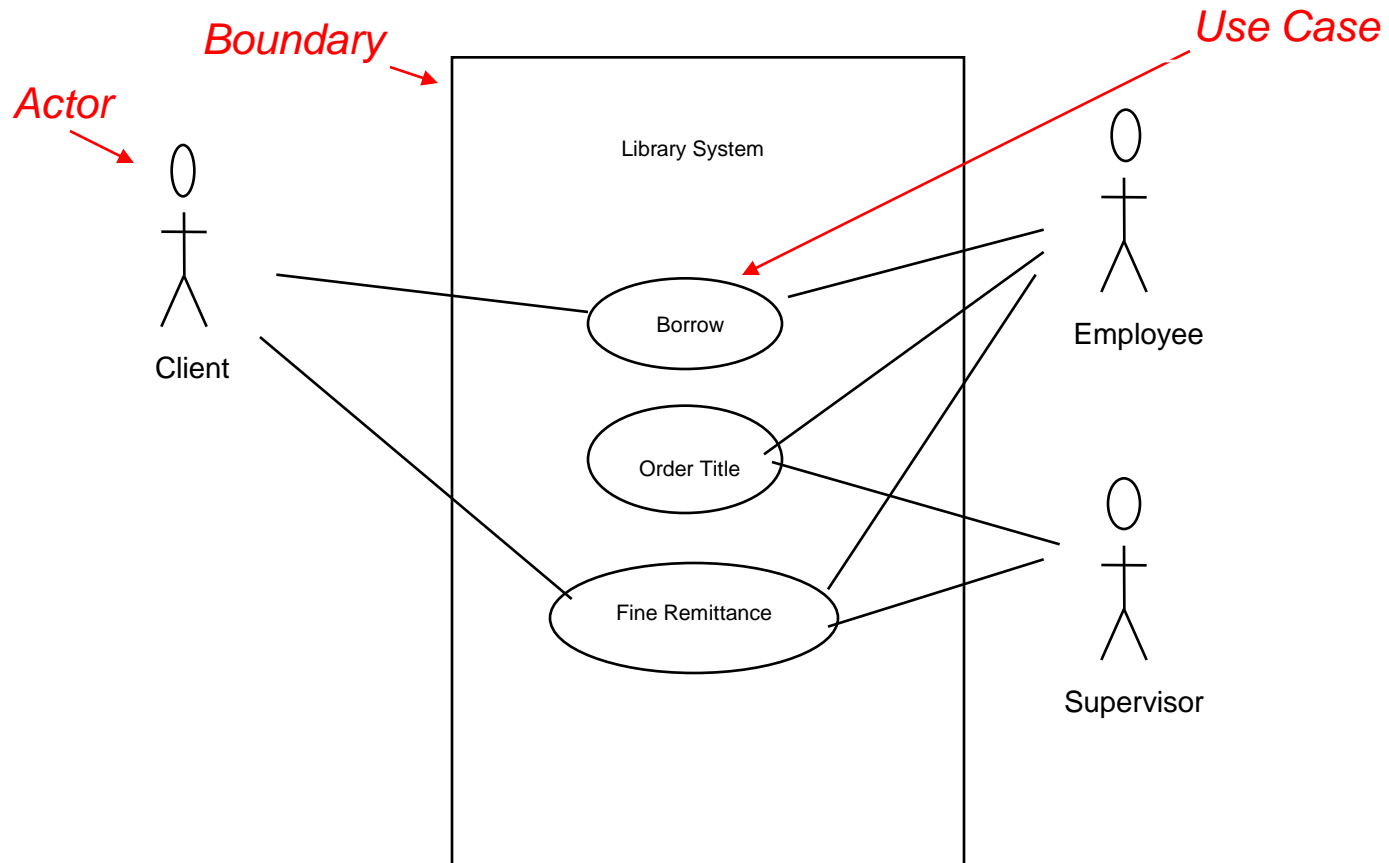
client

Use Case Diagrams – Use Cases

- Use case is a particular activity a user can do on the system.
- Is represented by an ellipse.
- Following are two use cases for a library system.



Use Case Diagrams



- A generalized description of how a system will be used.
- Provides an overview of the intended functionality of the system

Finding Actors

- Who or what will use the main functionality of the system?
- Who or what will provide input to this system?
- Who or what will use output from this system?
- Who will need support from the system to do their work?
- Are there any other software systems with which this one needs to interact
- Are there any hardware devices used or controlled by this system?

Answer these questions to find actors for a use cases of a cash dispenser system

Customer
Withdraw money
Get account balance

University Record System (URS)

- A University record system should keep information about its students and academic staff.
- Records for all university members are to include their id number, surname, given name, email, address, date of birth, and telephone number.
 - Students and academic staff each have their own unique ID number: studN (students), acadN (academic employee), where N is an integer ($N > 0$).
- In addition to the attributes mentioned above:
 - Students will also have a list of subjects they are enrolled in. A student cannot be enrolled in any more than 10 subjects.
 - Academic employees will have a salary, and a list of subjects they teach. An academic can teach no more than 3 subjects.

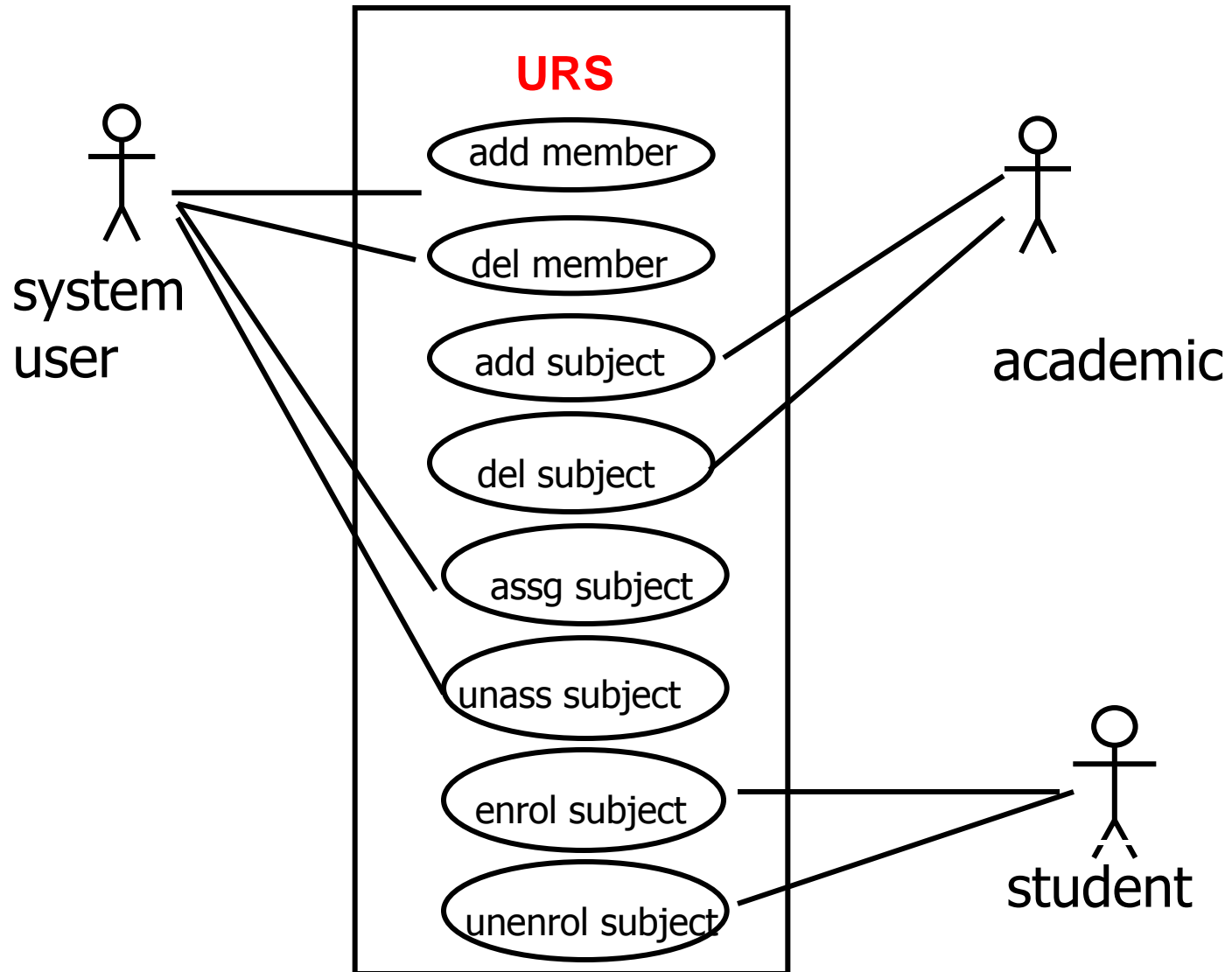
Some Actions Supported by URS

- The system should be able to handle the following commands.
 - Add and remove university members (students, and academic staff)
 - Add and Delete subjects
 - Assign and Un-assign subjects to students
 - Assign and Un-assign subjects to academic staff.

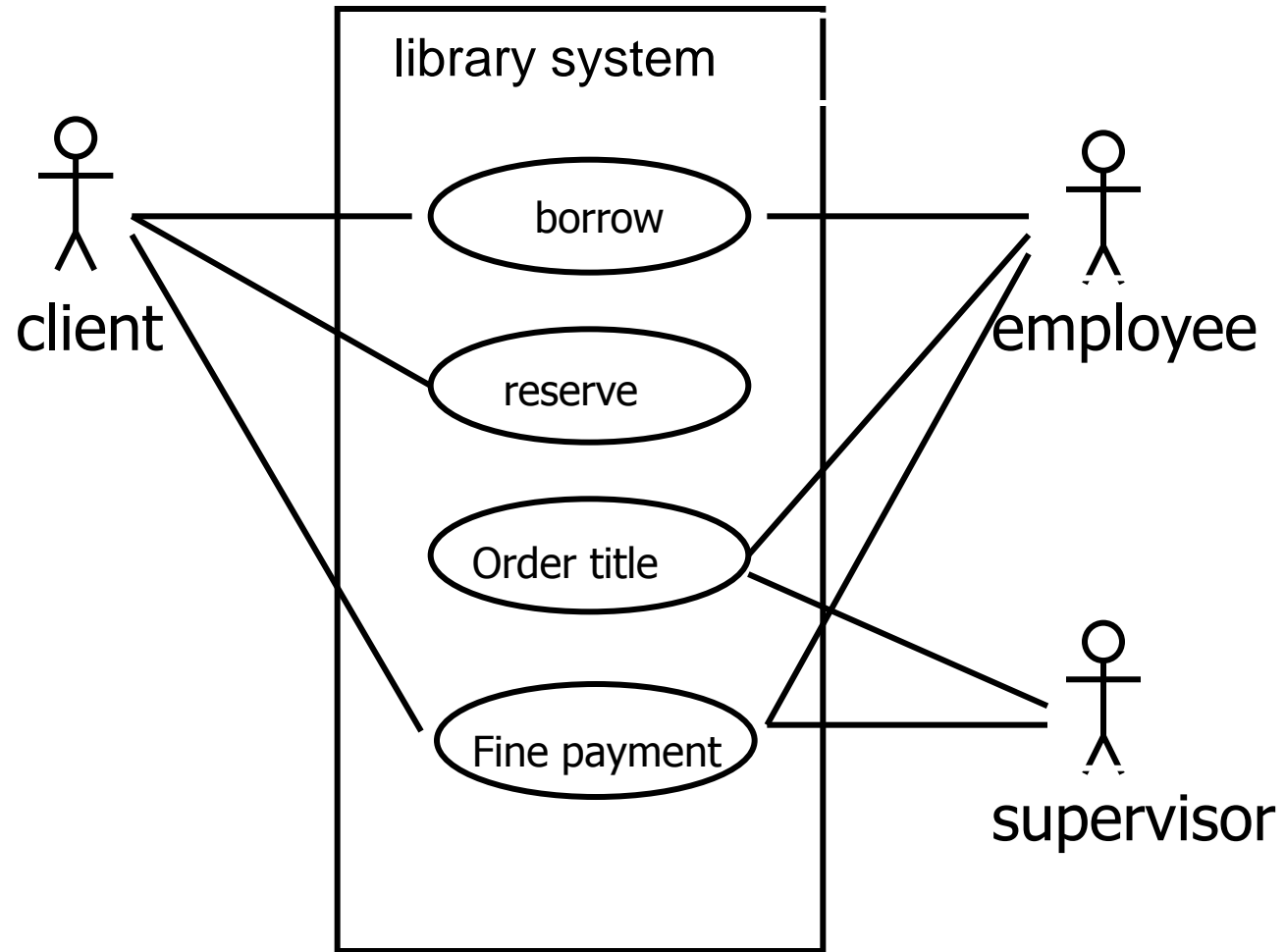
Use Case Diagrams

- Use Case diagrams show the various activities the users can perform on the system.
 - System is something that performs a function.
- They model the dynamic aspects of the system.
- Provides a *user's* perspective of the system.

Use Case Diagram - URS System

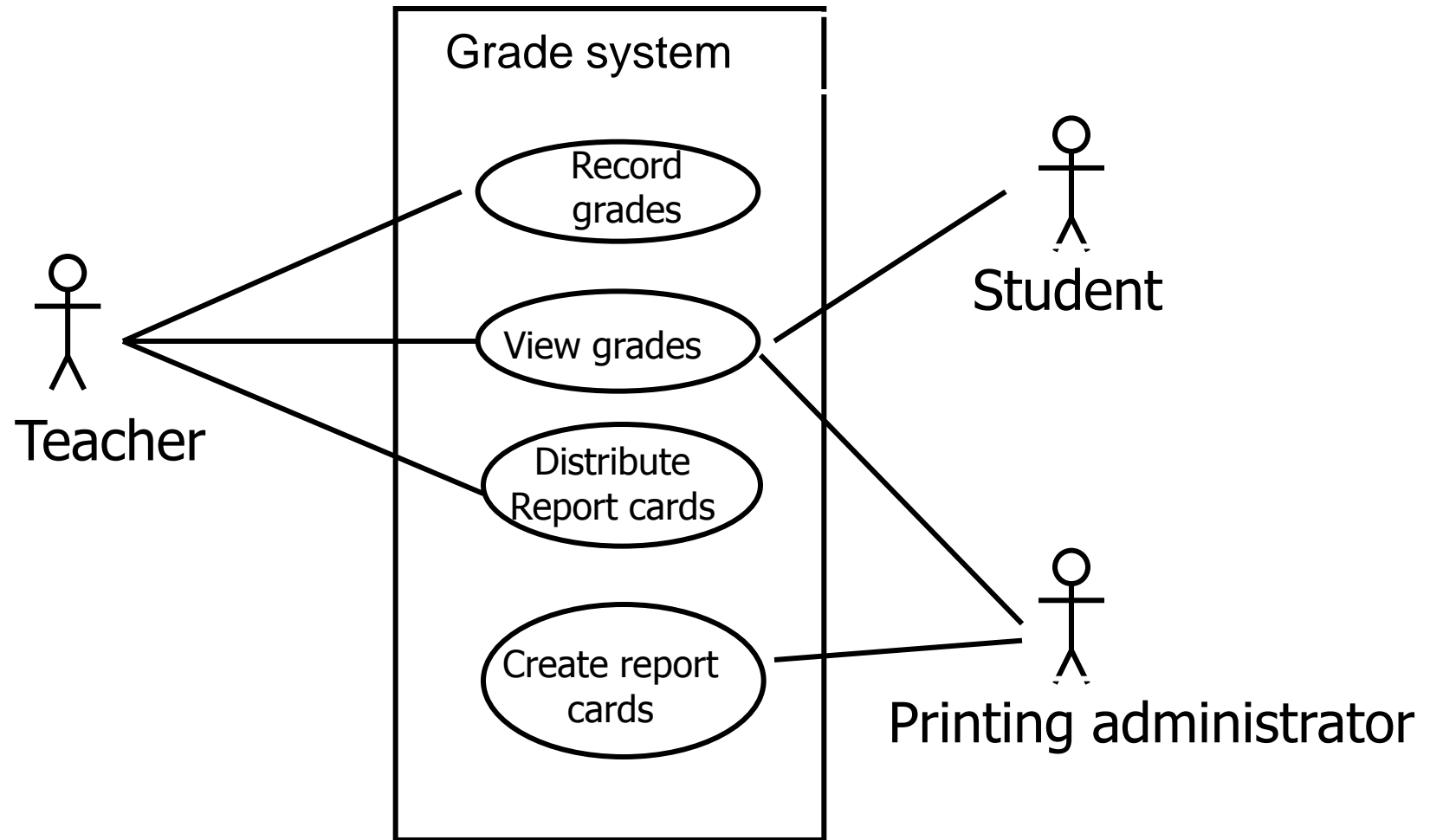


Use Case Diagram – Example1 (Library)

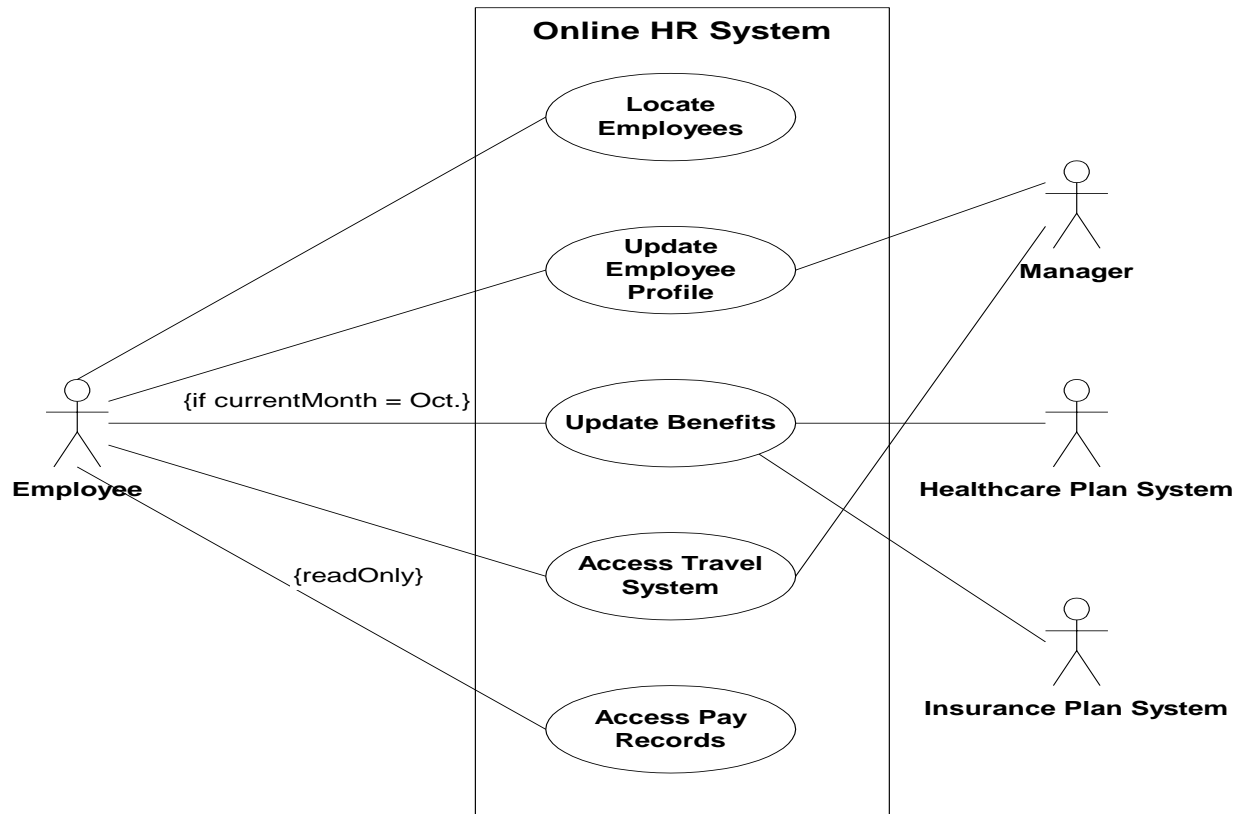


A Library System.

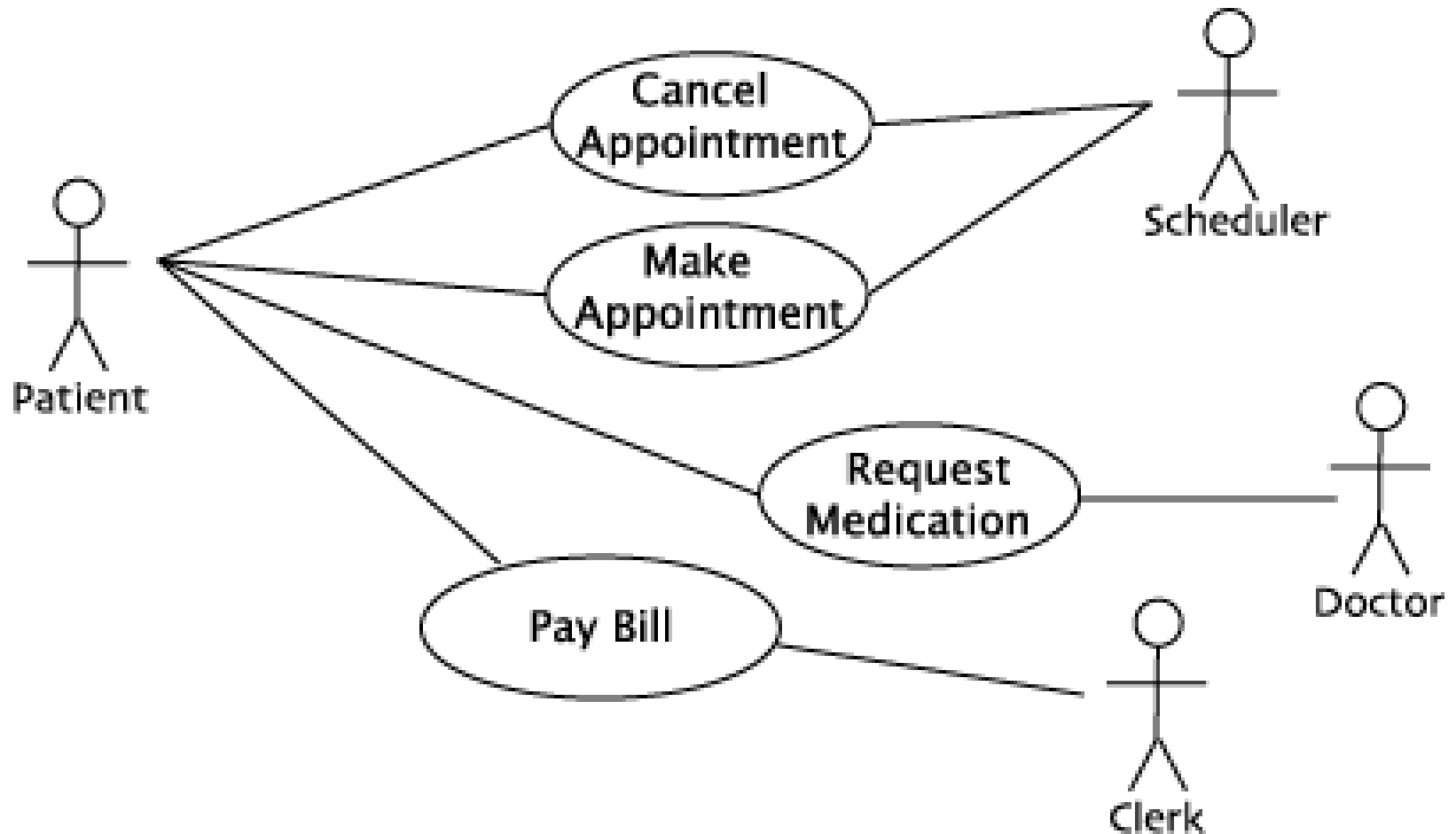
Use Case Diagram for Student Assessment Management System



Example: Online HR System



Example: Make Appointment



Use Case Vs Scenarios

- Each use case is one or more scenarios.
 - Add Subject Use Case :
 - Scenario 1 : Subject gets added successfully.
 - Scenario 2 : Adding the subject fails since the subject is already in the database.
 - Enroll Subject Use Case:
 - Scenario 1 : Student is enrolled for the subject.
 - Scenario 2 : Enrollment fails since the student is already enrolled in the subject.
- Each scenario has a sequence of steps.

Scenarios

- Each scenario has a sequence of steps.
 - Scenario 1 : Student is enrolled for the subject.
 - Student chooses the “enroll subject” action.
 - Check the student has enrolled in less than 10 subjects.
 - Check if the subject is valid.
 - Assign the subject to the student.

Scenarios

- Each scenario has a sequence of steps.
 - Scenario 2 : Enrolling fails since the student is already enrolled in 10 subjects.
 - Student chooses the “enroll subject” action.
 - Check the student has enrolled in less than 10 subjects.
 - Return an error message to the student.

Use case and Scenario Example

- A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of Rupees 100. Approval must be obtained from the bank before cash is dispensed.

Withdrawal Transaction Use Case

- A withdrawal transaction asks the customer to choose a type of account to withdraw from. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to the bank.
- Pre-conditions: The customer must have a valid ATM card and PIN.
- Post-conditions: The customer receives the cash amount that he wanted to withdraw, with a receipt, if indicated. The customer's account balance is updated in the system.

- Specifications:
- Primary Actor: Customer
- Stakeholders:
 - Customer: Wants quick, accurate withdrawal of cash
 - Bank: Wants to give fast, accurate and reliable service to the customer
 - Bank that owns ATM: (If not the same as the customer's bank): Wants to charge the customer the correct amount of surcharge on the withdrawal.
 - ATM Administrator: Wants to ensure that the ATM always has sufficient cash for a predicted number of withdrawals per day.
- **Normal flow of events:**
 1. The customer inserts ATM card into the ATM machine and enters PIN.
 2. The system validates the ATM card and PIN.
 3. The customer selects the 'Cash Withdrawal' option from the Options Menu.
 4. The system prompts the customer to enter the amount of cash that he or she wants to withdraw.
 5. The customer enters a cash amount and selects the 'Submit' option on the Cash Withdrawal Screen.
 6. The system validates the amount entered; checks account balance and that the machine has enough cash for the transaction, and asks the customer if he or she wants a receipt for the transaction.
 7. The customer selects 'Yes' on the Receipts Screen.
 8. The system ejects the ATM card, provides the cash, prints the receipt and updates the account balance of the customer in the system.

■ **Alternate flow of events:**

1. The customer has entered invalid PIN - The system prompts the customer to enter a valid PIN.
2. If ATM card is not compatible - The system rejects the ATM card and displays an error message.
3. The customer has entered an amount that exceeds the withdrawal limit.
4. The system rejects the transaction & displays an error message.

Use Case Diagrams - Relationships

- Inclusion

- *Inclusion enables to reuse one use case's steps inside another use case.*

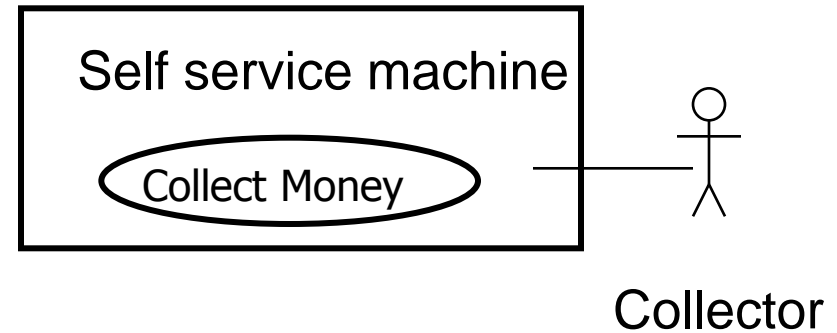
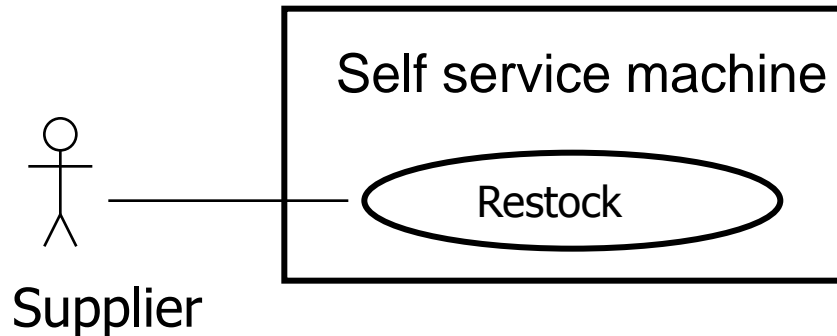
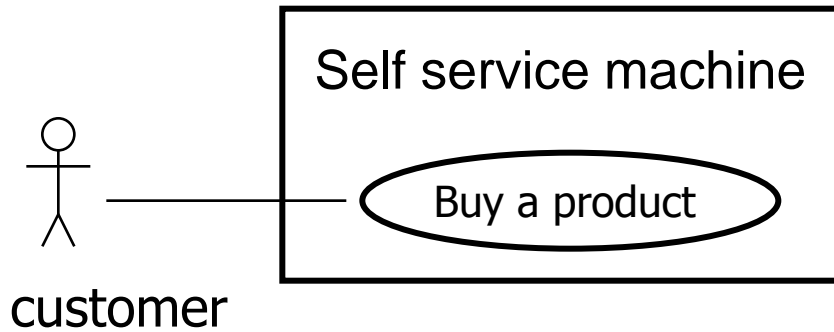
- Extension

- *Allows creating a new use case by adding steps to existing use cases*

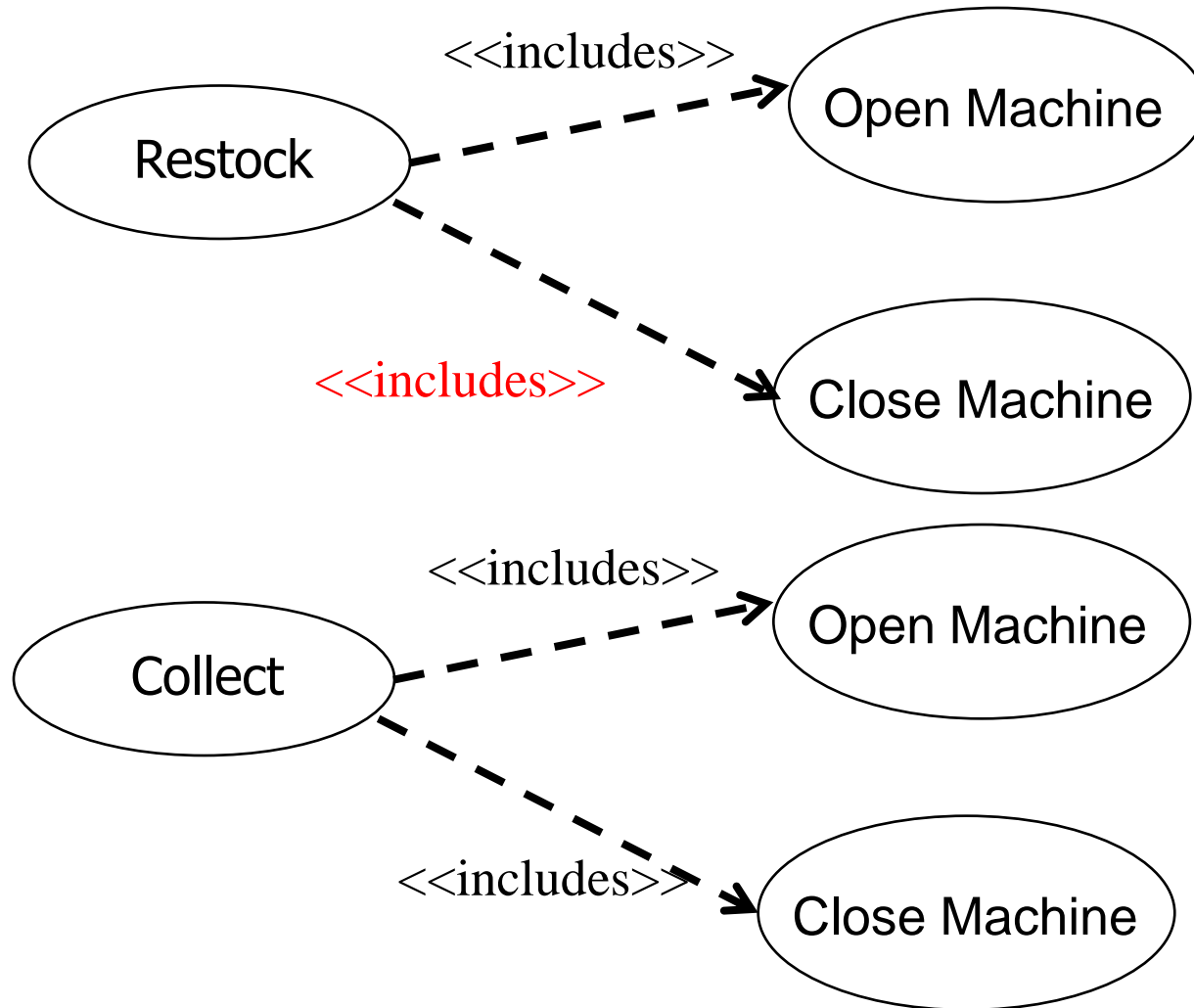
- Generalization

- *Allows child use cases to inherit behavior from parent use cases*

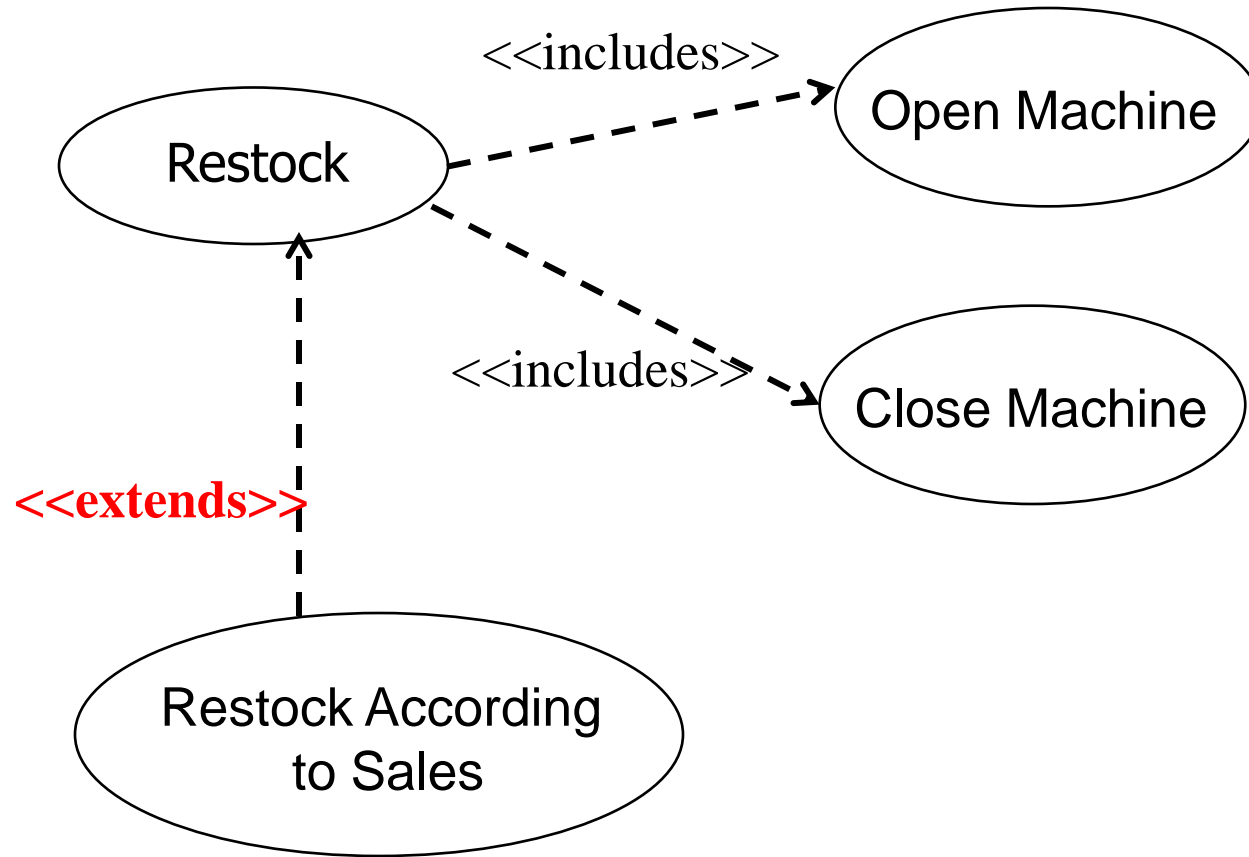
Use Case – Example 1



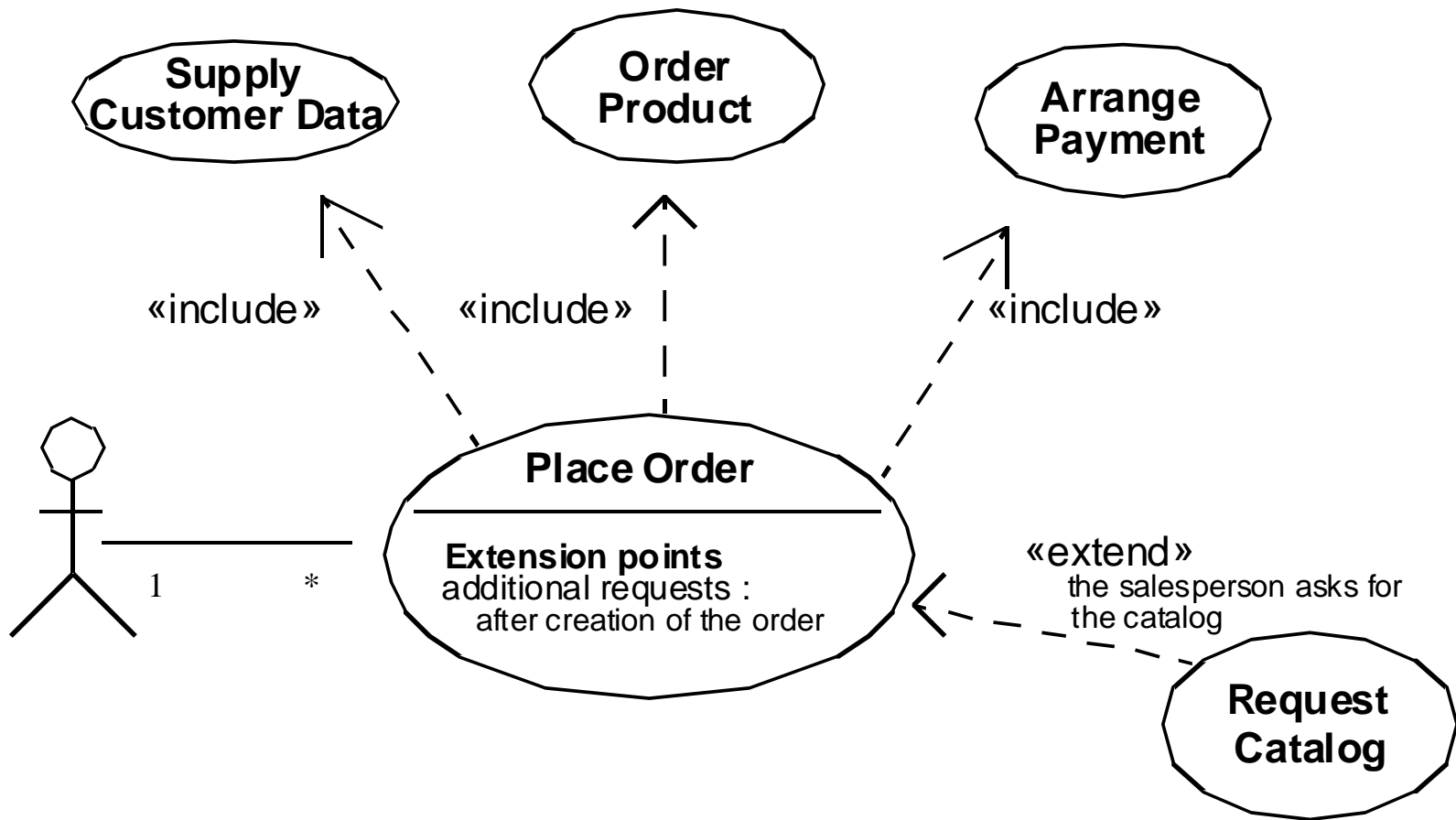
Use Case – Example (self service machine – includes relationship)



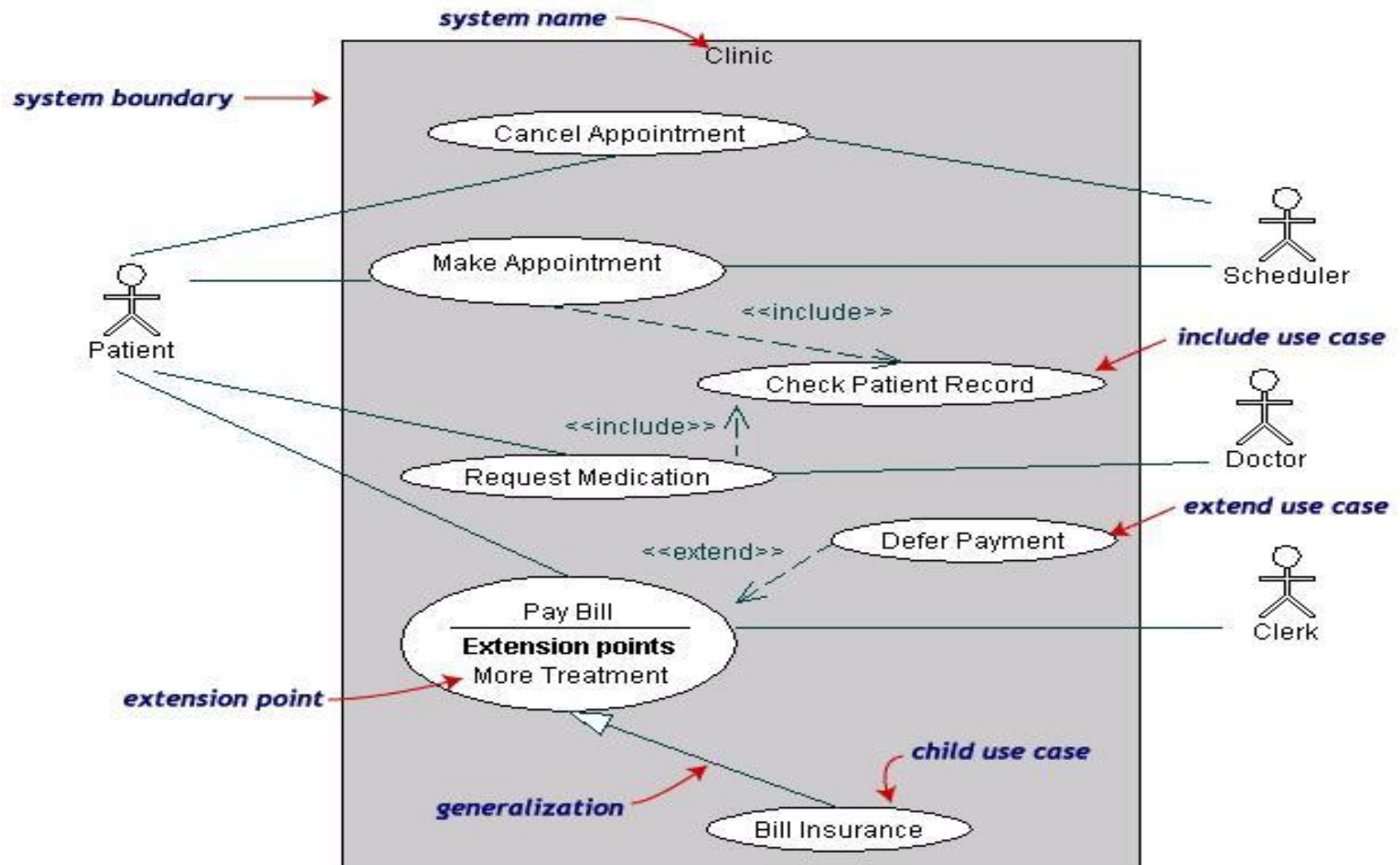
Use Case – Example (self service machine – extends relationship)



Use Case Relationship Example-2



Use Case Diagrams Example 3

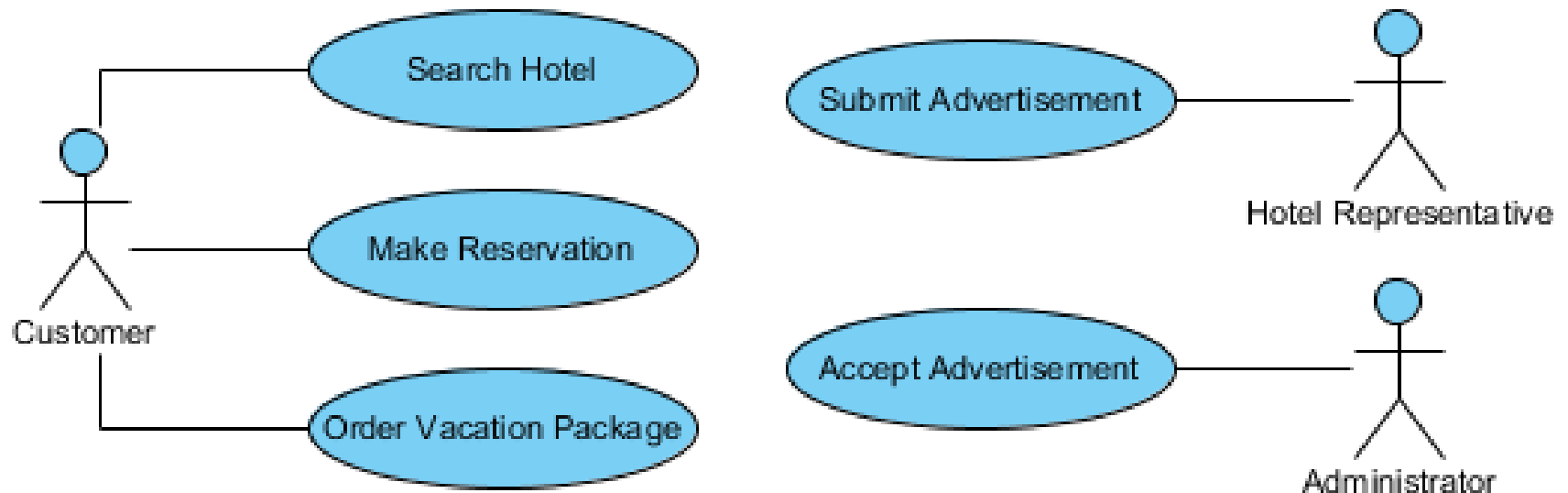


Use Case Diagrams(cont.)


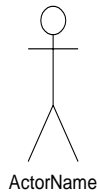
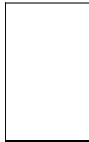
- **Pay Bill** is a parent use case and **Bill Insurance** is the child use case. (generalization)
- Both **Make Appointment** and **Request Medication** include **Check Patient Record** as a subtask.(include)
- The **extension point** is written inside the base case **Pay bill**; the extending class **Defer payment** adds the behavior of this extension point. (extend)

Exercise




- Create a Use case for booking a hotel
- Create a use case for advertising hotel package
- Create a use case for accepting a advertisement by an administrator



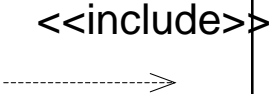
Use Case Modeling: Core Elements (Summary 1)

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

Use Case Modeling: Core Elements (Summary 2)

Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
extend	A relationship from an <i>extension</i> use case to a <i>base</i> use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	

Use Case Modeling: Core Elements (Summary 3)

Construct	Description	Syntax
include	An relationship from a <i>base</i> use case to an <i>inclusion</i> use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	 <p>The diagram shows the UML syntax for an include relationship. It consists of the text '<<include>>' followed by a dashed arrow pointing to the right. The arrow is horizontal and has a solid arrowhead.</p>

Activity Diagrams

Definition

- Activity diagrams represent the dynamics of the system.
- They are flow charts that are used to show the workflow of a system.
- They show.
 - The flow of control from activity to activity in the system,
 - What activities can be done in parallel.
 - Alternate paths through the flow.
- They can show the flow across use cases or within a use case.
- Helpful in developing system sequence diagrams (SSD)

Activity Diagram

Supplements the use-case by providing a diagrammatic representation of procedural flow

Details:

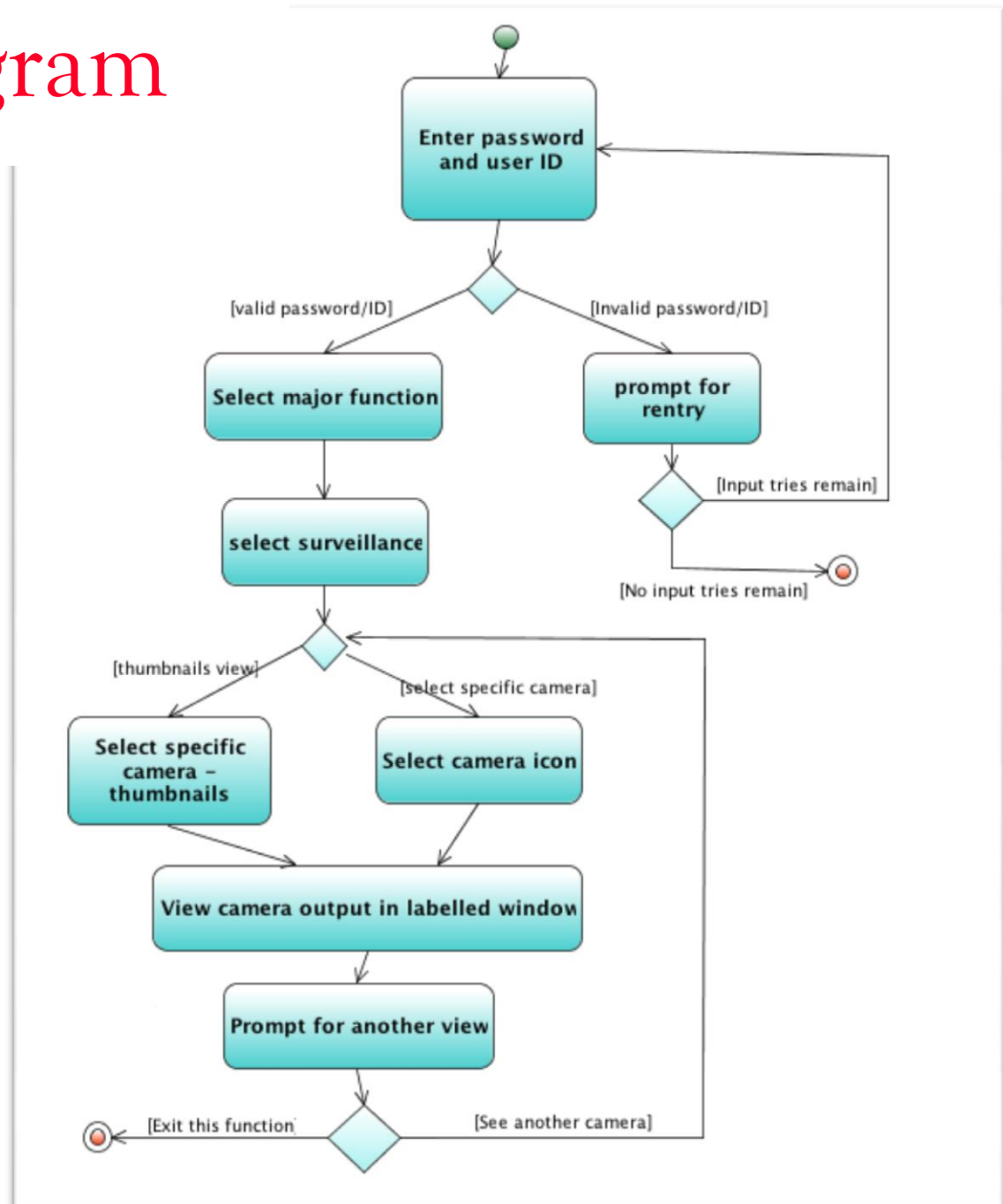
- Start is a single circle
- End is a bulls-eye
- Decisions are diamonds (guards must be on both branches of the diamond!)



Start Marker



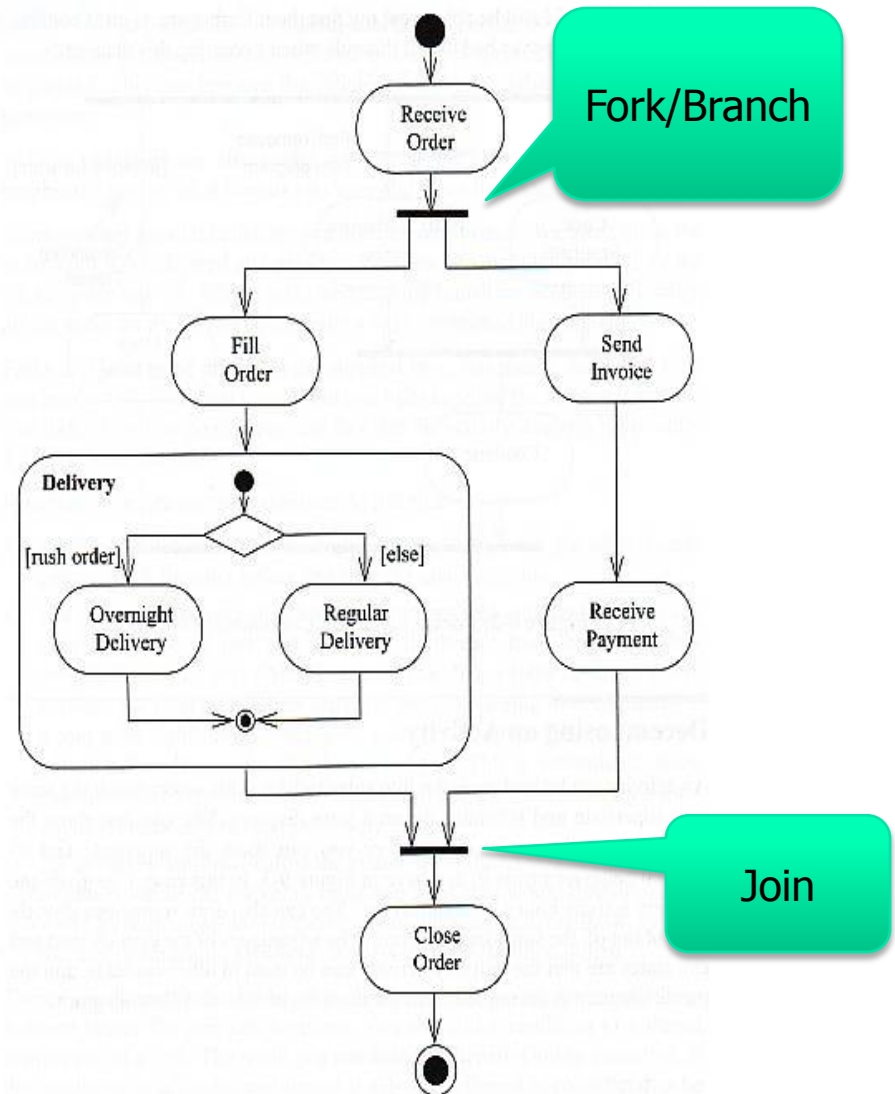
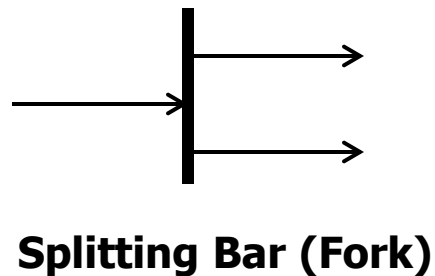
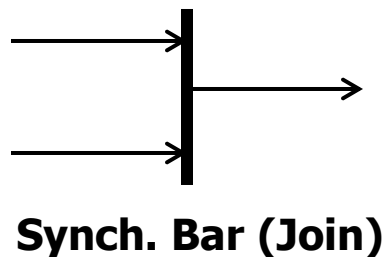
Stop Marker

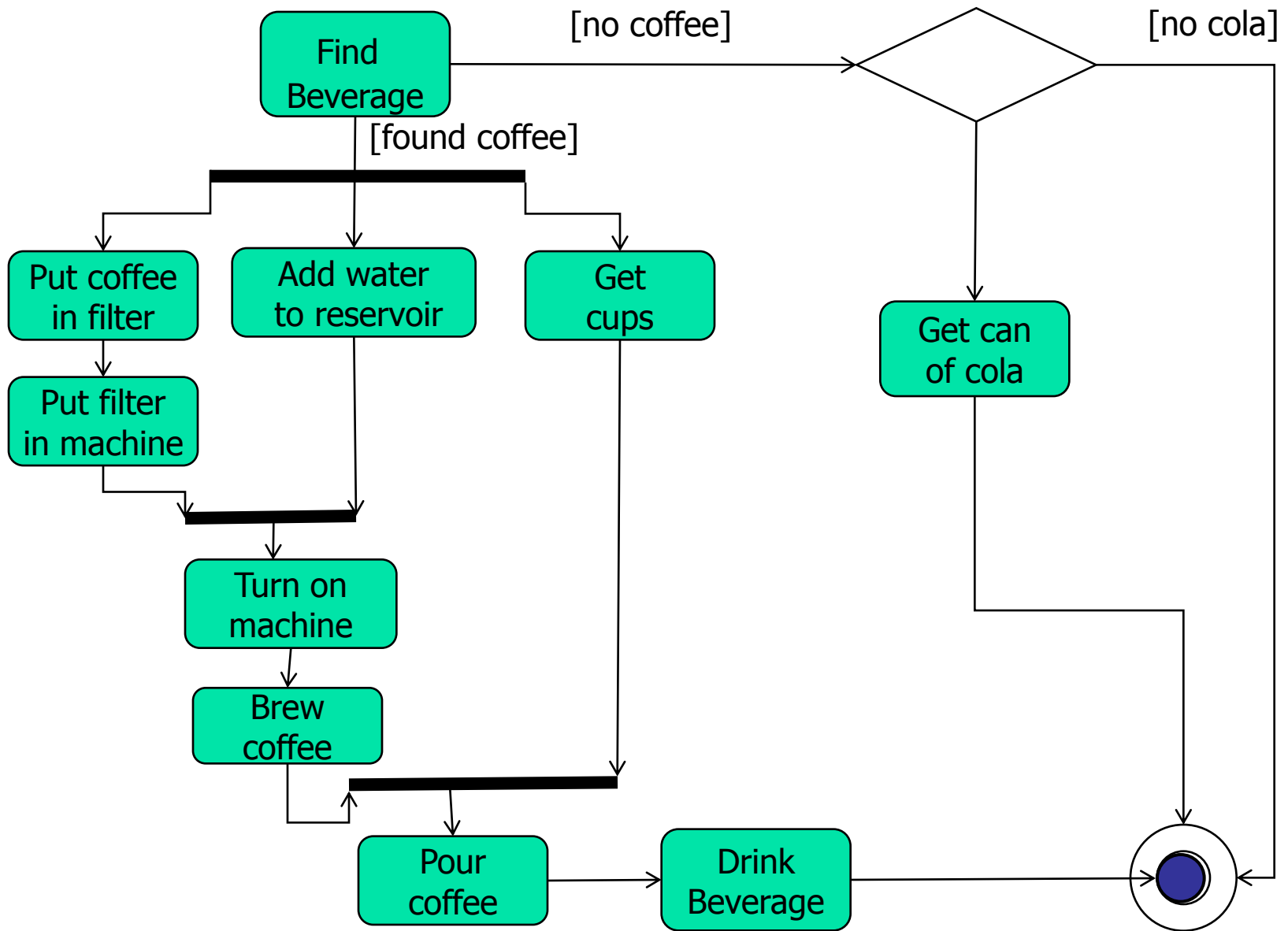


Activity Diagram Example

To show concurrent activity, activity diagrams allow branches and joins.

You can also reference or include other activity diagrams





When to Use Activity Diagrams

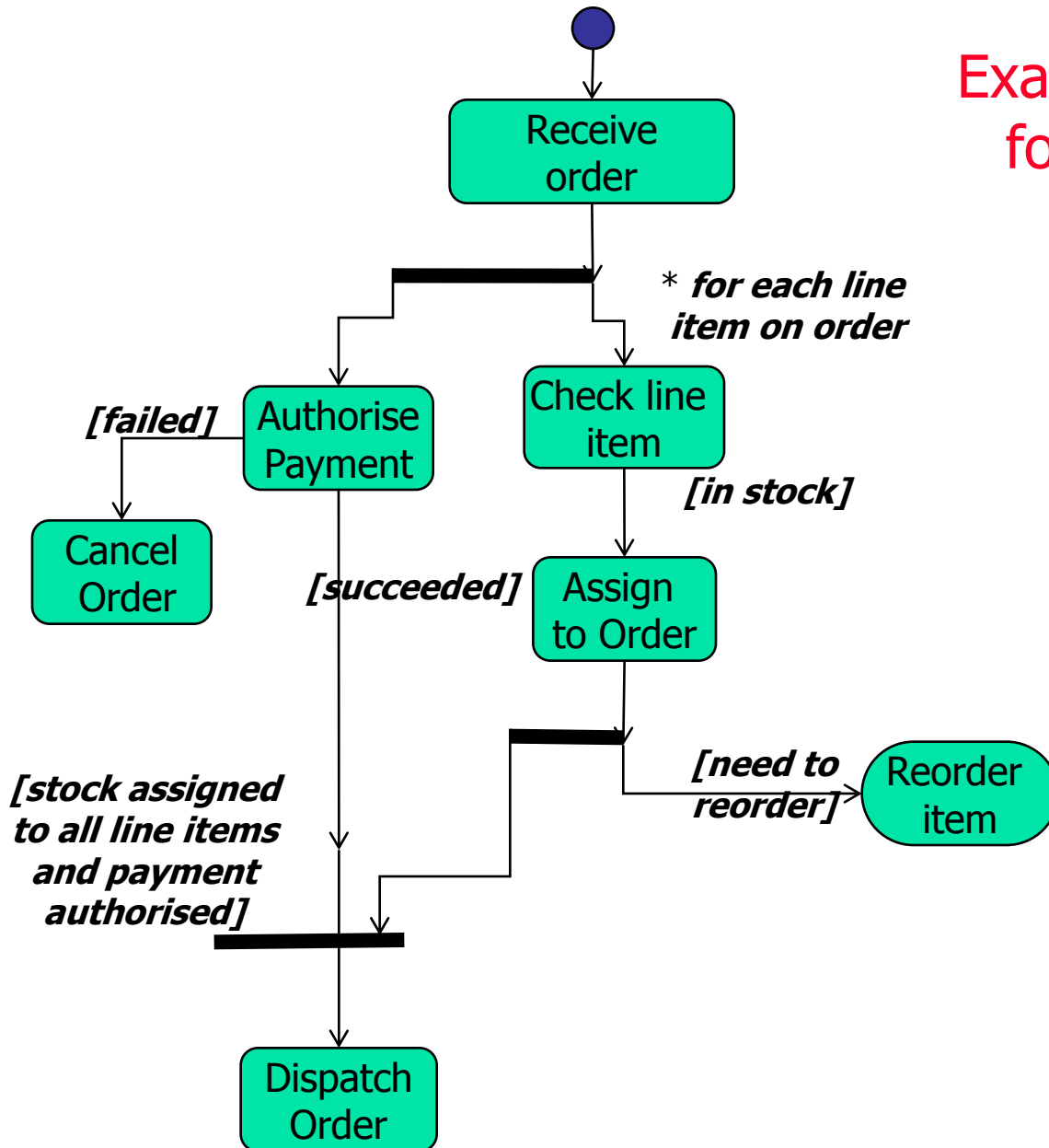
- Do use them for

- Analysing Use Cases.
- Understanding workflow across many Use Cases.
- Dealing with multi-threaded applications.

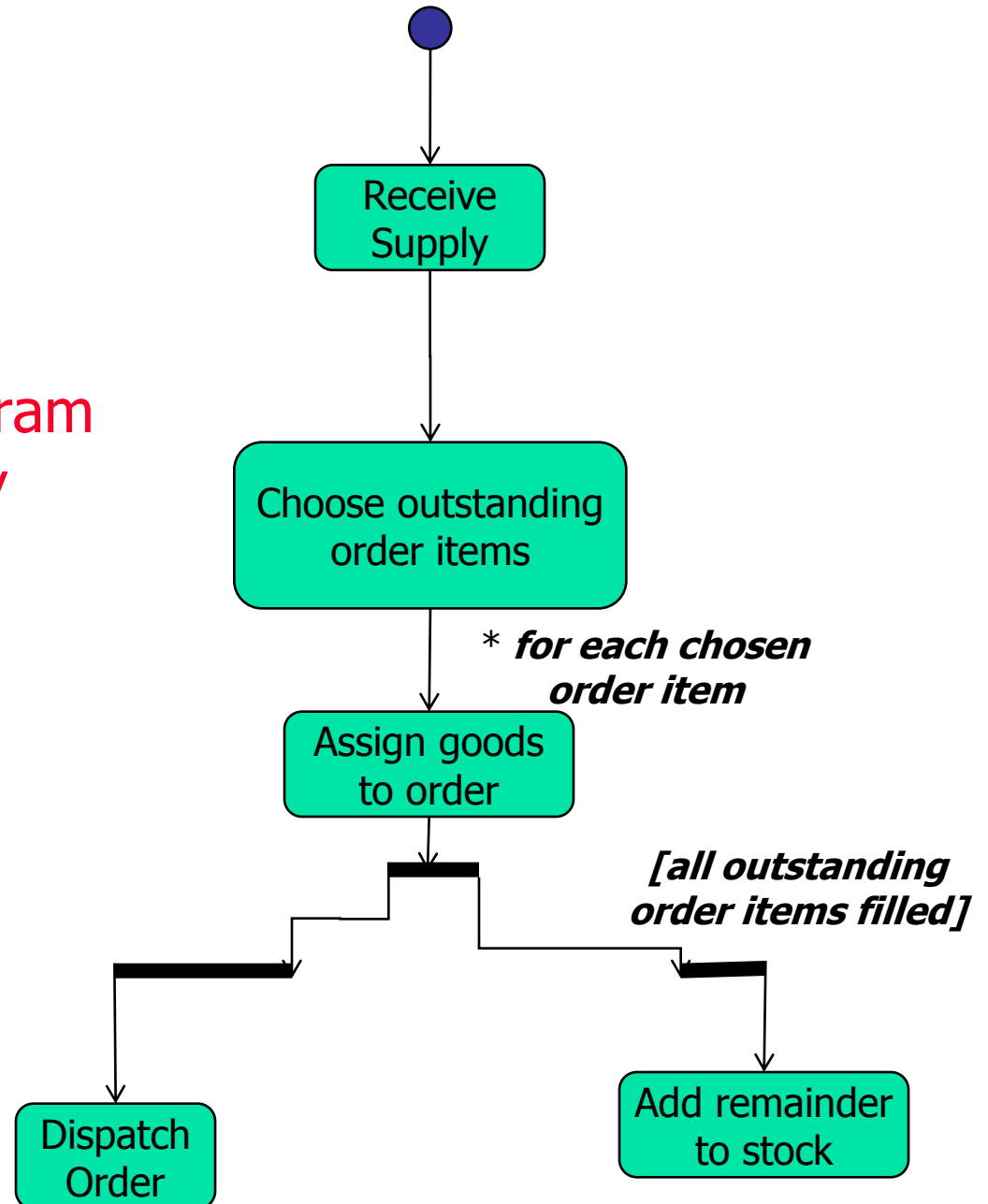
- Don't use them

- to see how objects collaborate.
- to see how an object behaves over its lifetime.

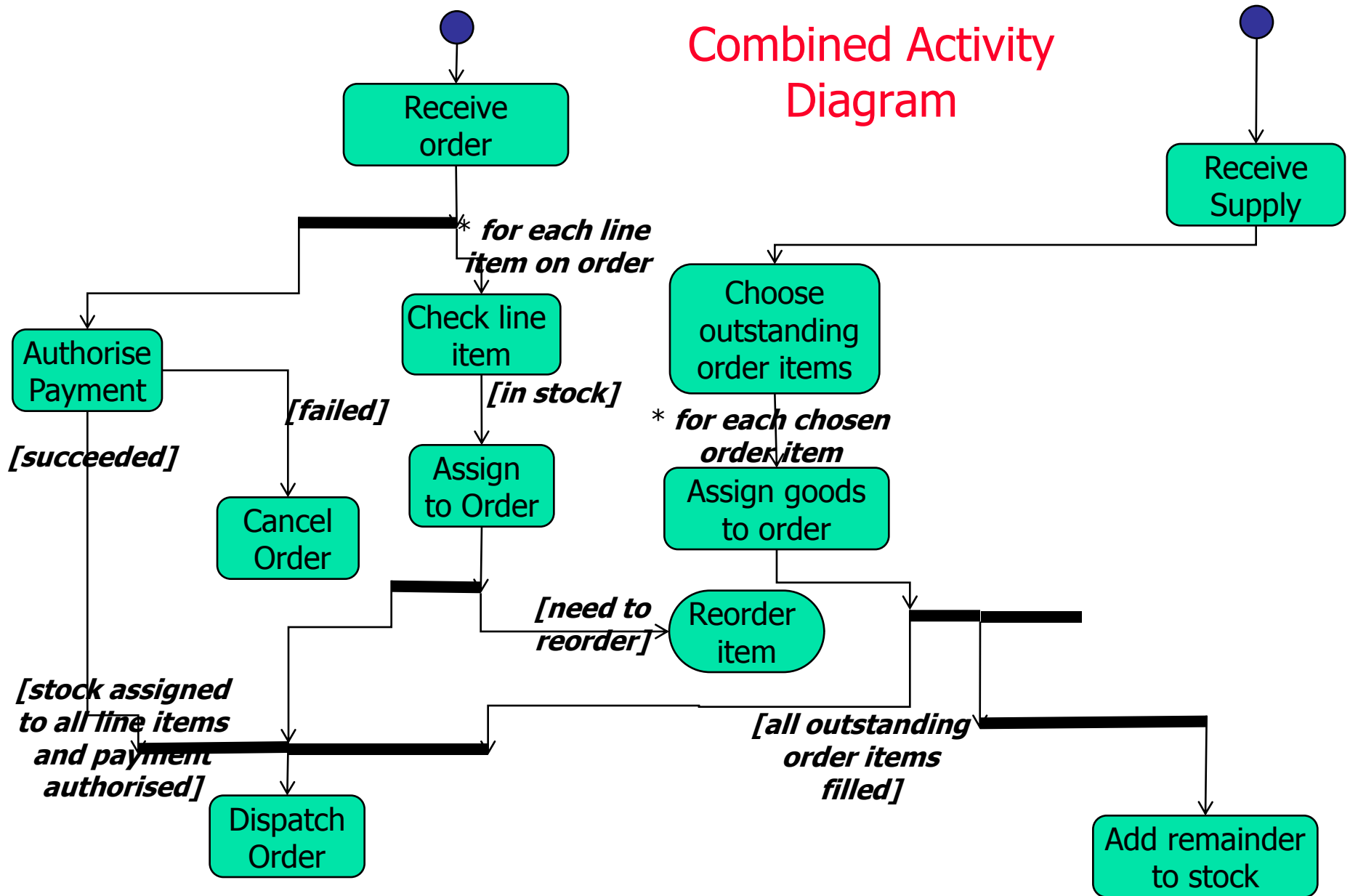
Example: Activity Diagram for Receiving an Order

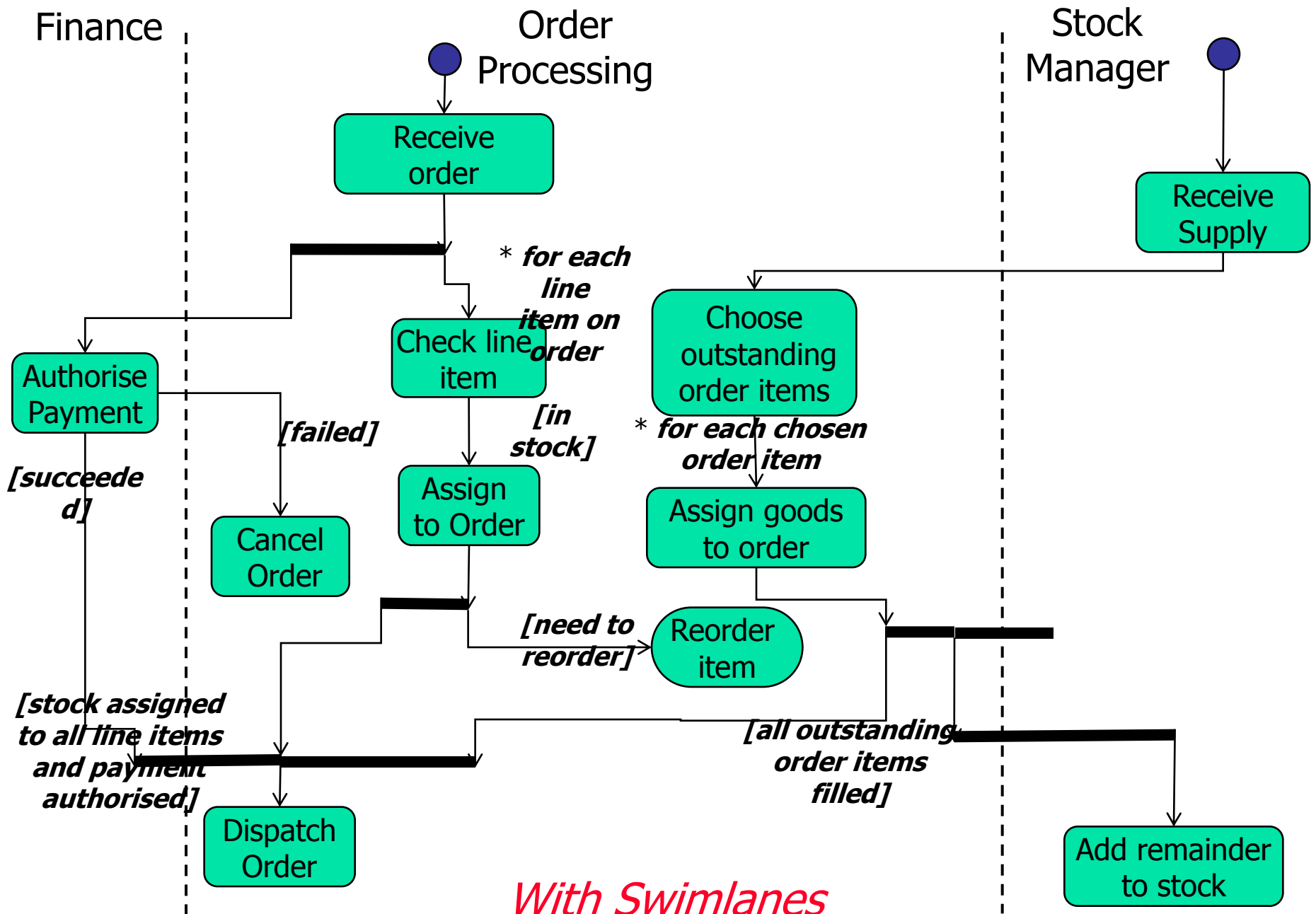


Example: Activity Diagram
for receiving Supply



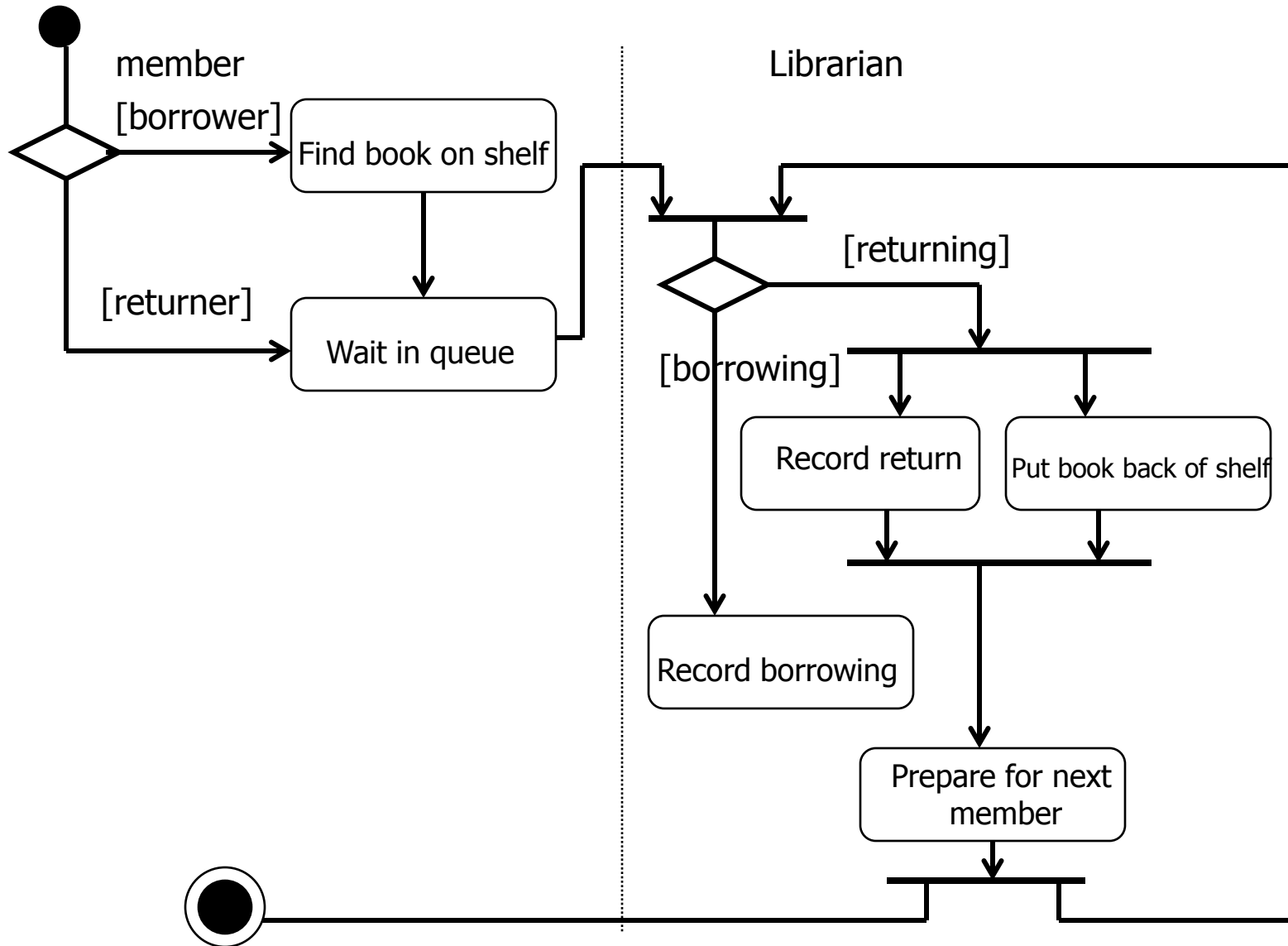
Combined Activity Diagram





Example: Business Level Activity Diagram of the Library

With Swimlanes



Interaction Diagram

What is an Interaction Diagram?

- Interaction Diagrams – models that describe how groups of objects collaborate in some behavior
 - Typically, an Interaction Diagram captures the behavior of a single use case
 - The Interaction Diagram shows a number of objects and the messages that are passed between these objects within the use case
- 2 kinds of Interaction Diagrams
 - Sequence Diagrams
 - Collaboration Diagrams

Sequence Diagram

Sequence Diagrams Purpose

- Used to show the interactions between objects (that participate in a use case) in the sequential order
- Conveys this information along the horizontal and vertical dimensions:
 - Vertical dimension
 - Top down, the time sequence of messages/calls as they occur
 - Horizontal dimension
 - Left to right, the object instances that the messages are sent to.

Emphasis on time ordering!

Elements

- Actor
- Object
- Lifeline
- Execution Occurrence
- Message
- Frame (scope)

Actor

- Person or system that delivers benefit
 - External to the system
- Participates by sending and/or receiving messages
- Shown as a stick figure (default)
- Non-human actor as a rectangle with “<<actor>>” inside (alternative)

Object

- Placed across the top of the diagram
- Participates in a sequence by sending and/or receiving messages
- Drawn as a box
 - Name is placed inside the box
 - The UML standard for naming a object follows the format of:
Object Name : Class Name

Lifeline

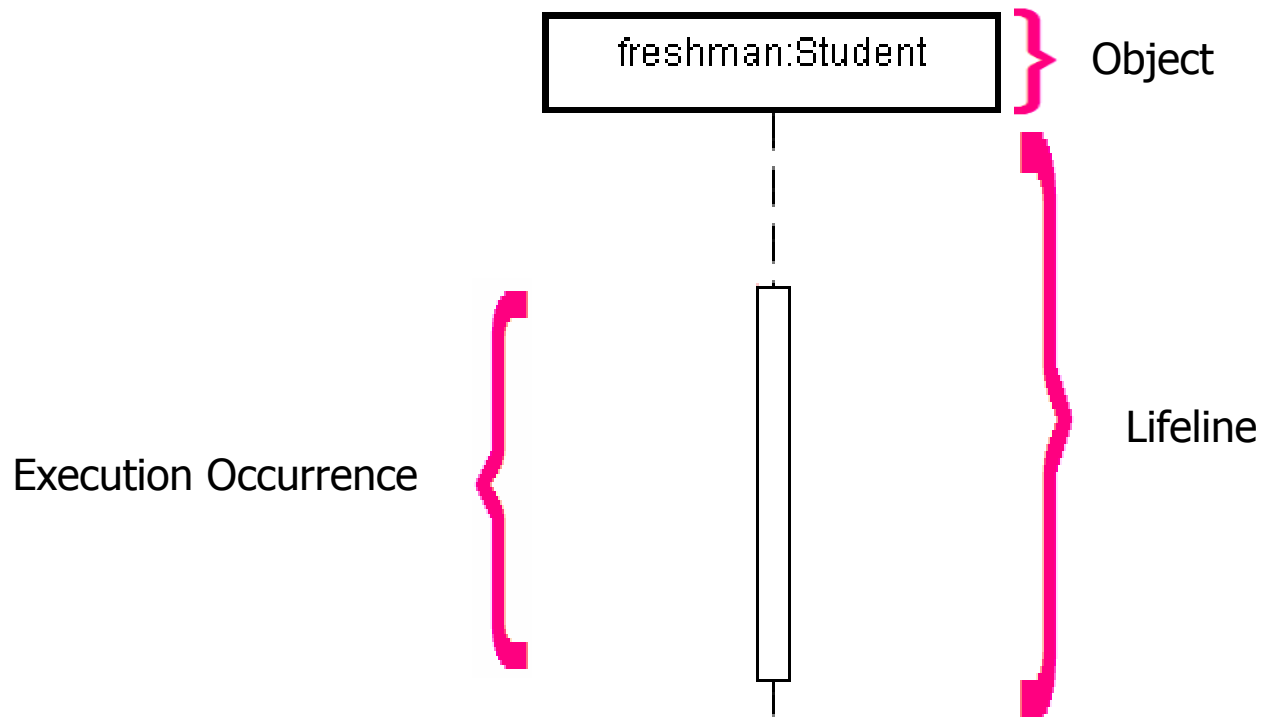
- Denotes the life of an object during a sequence
- Lifeline continues to the bottom of the diagram as the object continues to exist
- Contains an "X" at the point at which the class no longer interacts

Execution Occurrence

- Denotes when an object is sending or receiving messages
- Is a long narrow rectangle overlaid on to the lifeline

Example

- Student class used in an object whose instance name is "freshman"



A Message

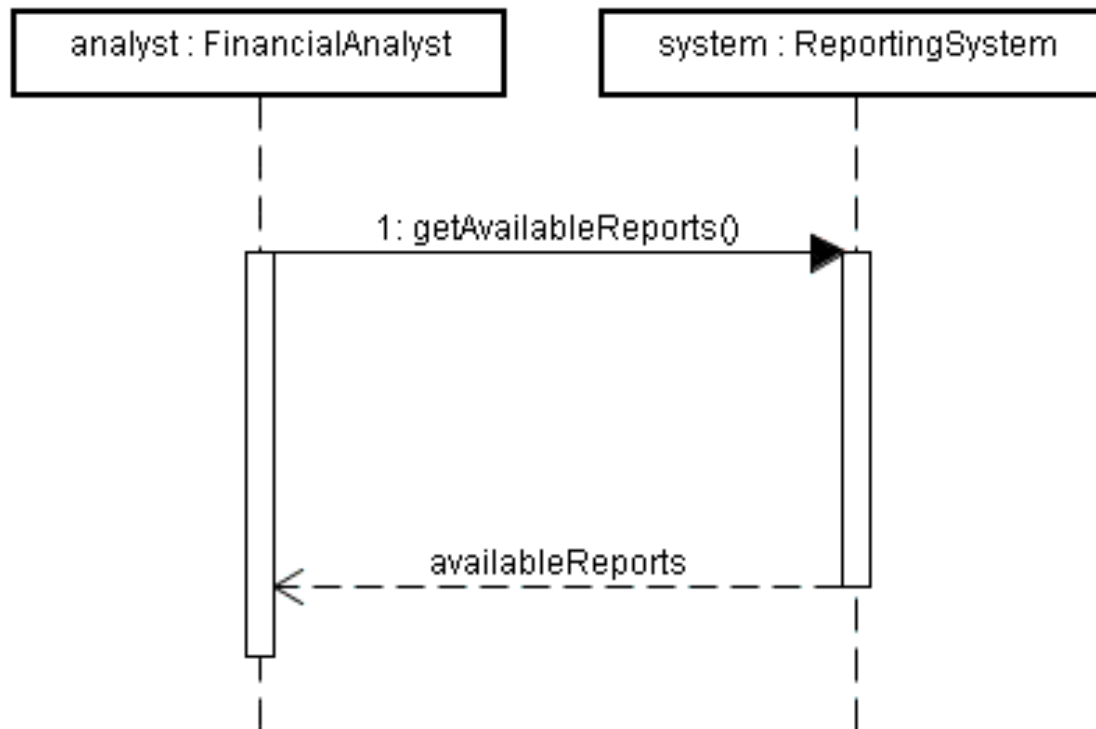
- A communication between objects that convey information with the expectation that activity will ensure
 - Represent operation/method
- First message always starts at the top
- Following messages are placed slightly lower than the previous message

Messages, continued

- Two types of messages are typically used
 - Operation Call – shown using solid lines connecting two objects with an arrow on the line showing which way the message is being passed
 - Return – depicted as a dashed line with an arrow on the end of the line portraying the direction of the return
- Object can send message to itself
 - self-delegation

Example 1

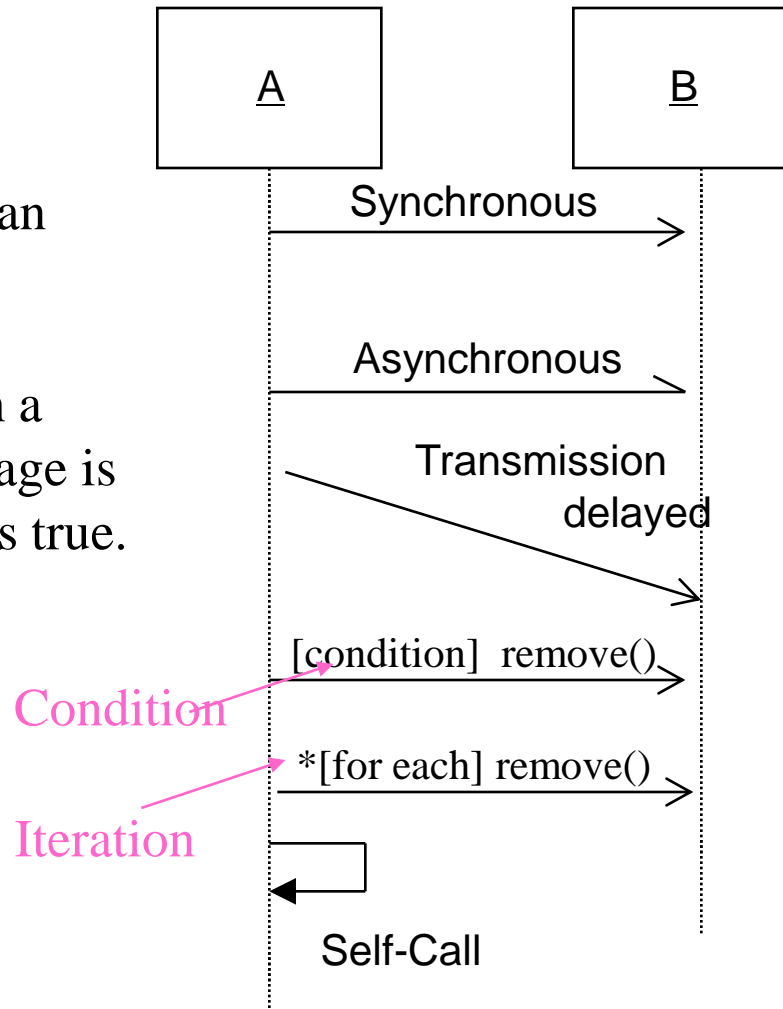
- Sending and returning of messages between objects



Object interaction Example 2

Self-Call: A message that an Object sends to itself.

Condition: indicates when a message is sent. The message is sent only if the condition is true.



Example 3: Object Life Spans Summary

■ Creation

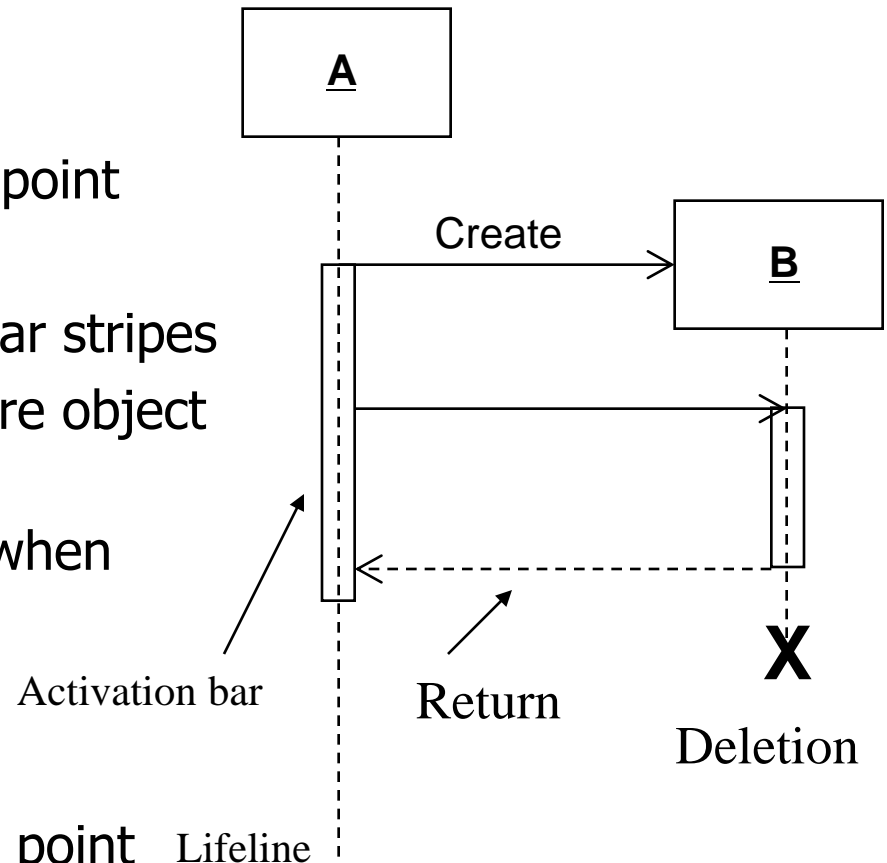
- Create message
- Object life starts at that point

■ Activation

- Symbolized by rectangular stripes
- Place on the lifeline where object is activated.
- Rectangle also denotes when object is deactivated.

■ Deletion

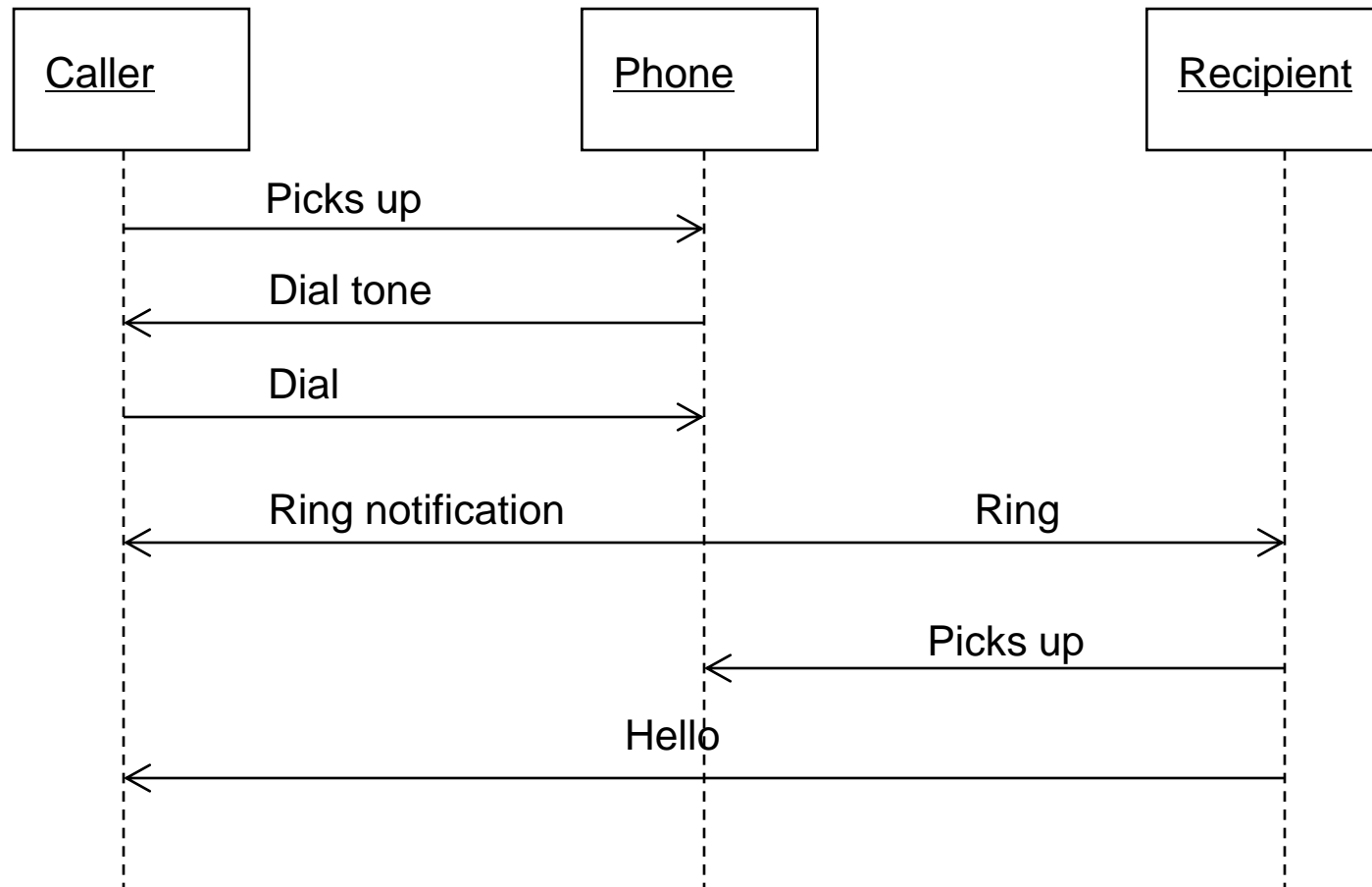
- Placing an 'X' on lifeline
- Object's life ends at that point



Building a Sequence Diagram

- 1) Set the context
- 2) Identify which object will participate
- 3) Set the lifeline for each object
- 4) Layout the messages from the top to the bottom of the diagram based on the order in which they are sent
- 5) Add the execution occurrence
- 6) Validate the sequence diagram

Sequence Diagram(make a phone call)



Example

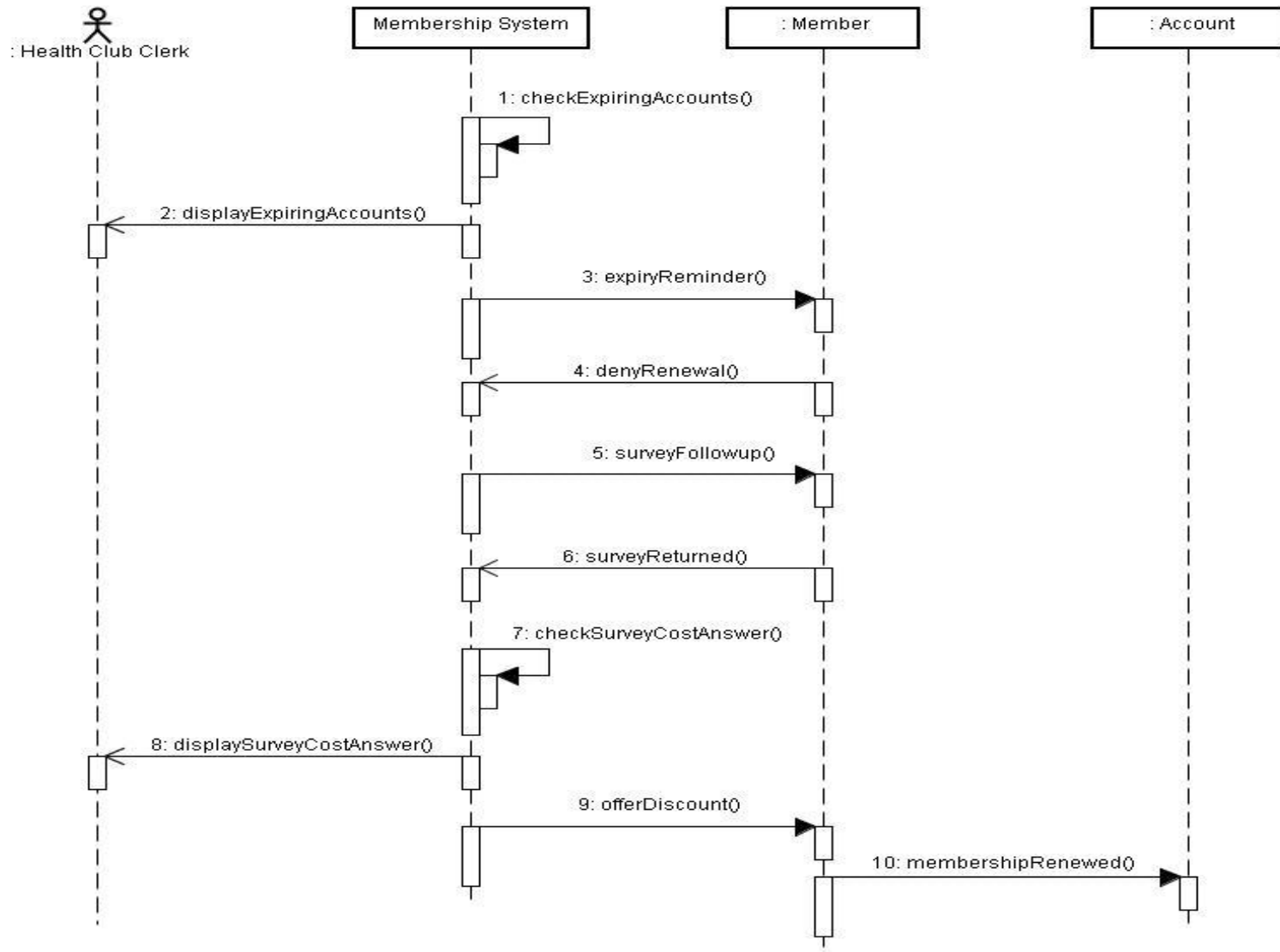
Sequence Diagrams

Create a sequence diagram for the following scenario description for a health club membership system. When members join the health club, they pay a fee for a certain length of time. The club wants to mail out reminder letters to members asking them to renew their memberships one month before they expire. About half of the members do not renew their memberships. These members are sent follow-up surveys to complete about why they decided not to renew so that the club can learn how to increase retention. If the member did not renew because of cost, a special discount is offered to that customer. Typically, 25 % of accounts are reactivated because of this offer.

Assumptions

- Member declined initial membership renewal
- Member declined renewal because of cost, allowing Health Club to offer special discount
- System automates checking of expiring accounts
- System automates sending of online account expiry reminder
- System automates sending of an online follow-up survey

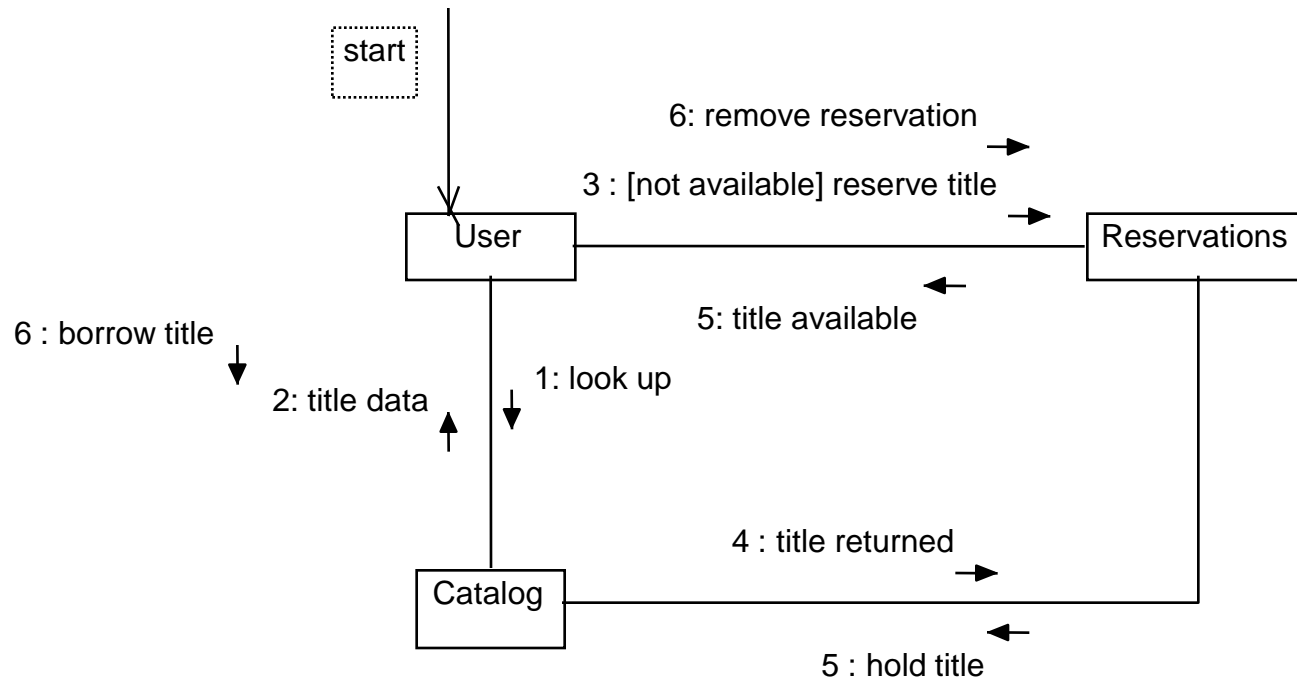
Example



Conclusion

- Use sequence diagrams when you want to show:
 - Interaction of logic between objects
 - Time progression of the system in a use case

Interaction Diagrams: Collaboration diagrams



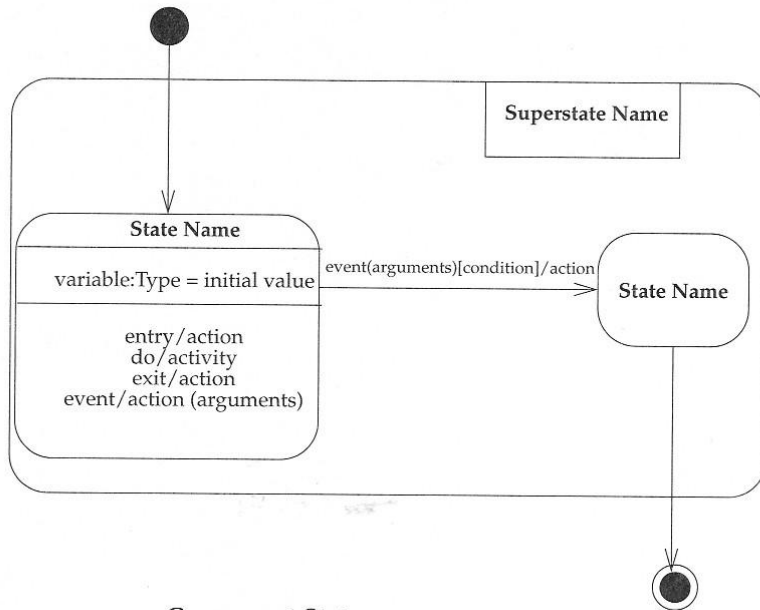
1. Shows the relationship between objects and the order of messages passed between them.
2. The objects are listed as rectangles and arrows indicate the messages being passed
3. The numbers next to the messages are called sequence numbers. They show the sequence of the messages as they are passed between the objects.
4. convey the same information as sequence diagrams, but focus on object roles instead of the time sequence.

What are State Diagrams?

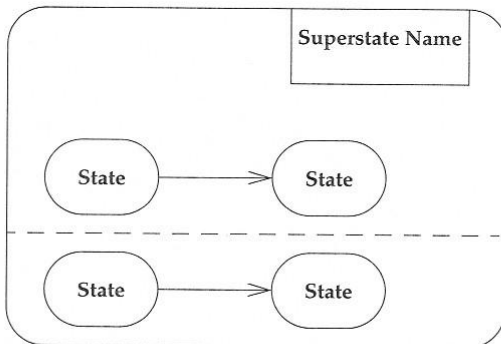
- State Diagrams – describe all the possible states an object can assume and how the object's state changes as a result of events that affect the object
 - State Diagrams are drawn for a single class to show the lifetime behavior of a single object
 - State Diagrams are good for describing the behavior of an object across several use cases
- In UML, State Diagrams
 - Support superstates (states which contain other states)
 - Support concurrency

State Diagrams - Notation

State Diagram

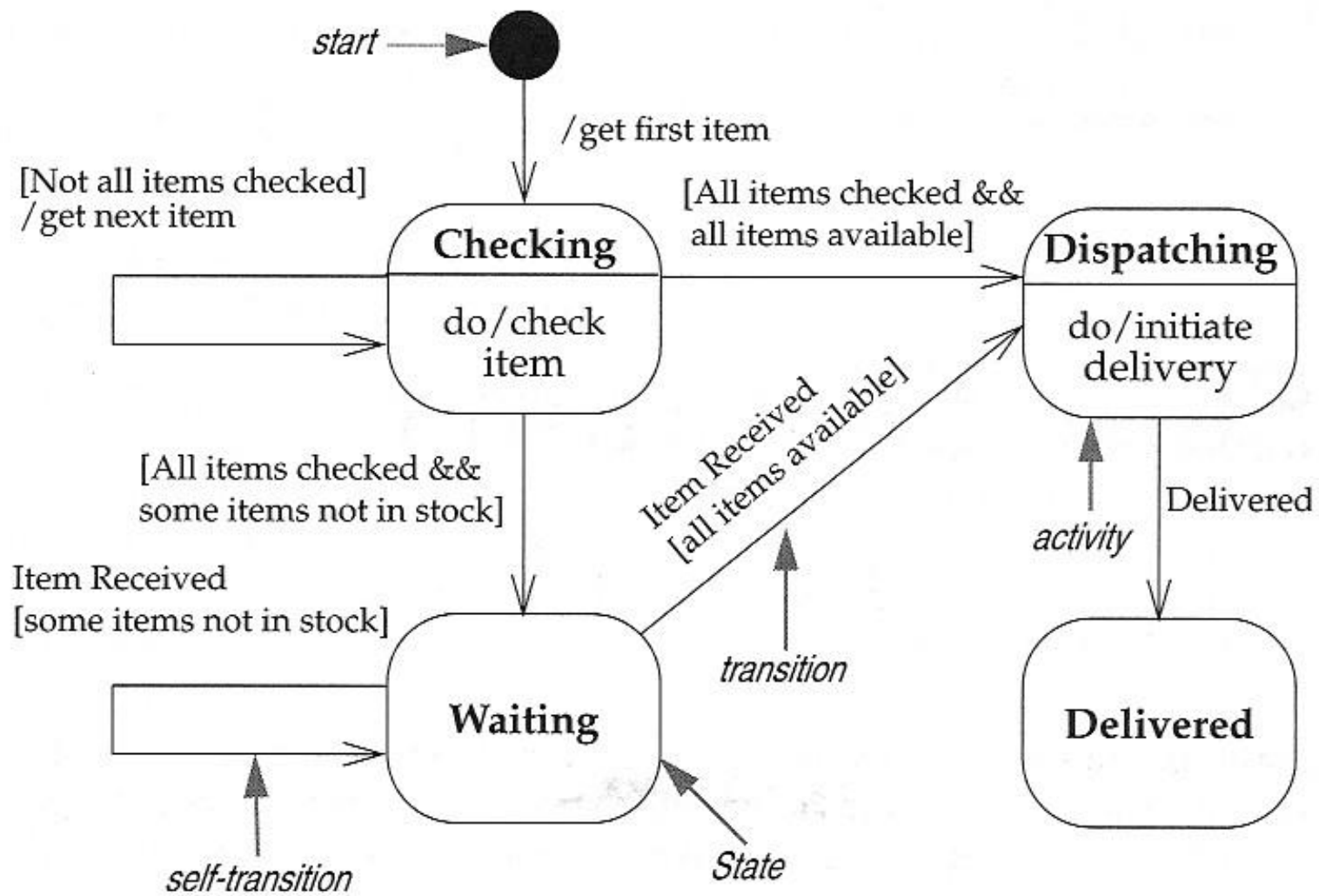


Concurrent States

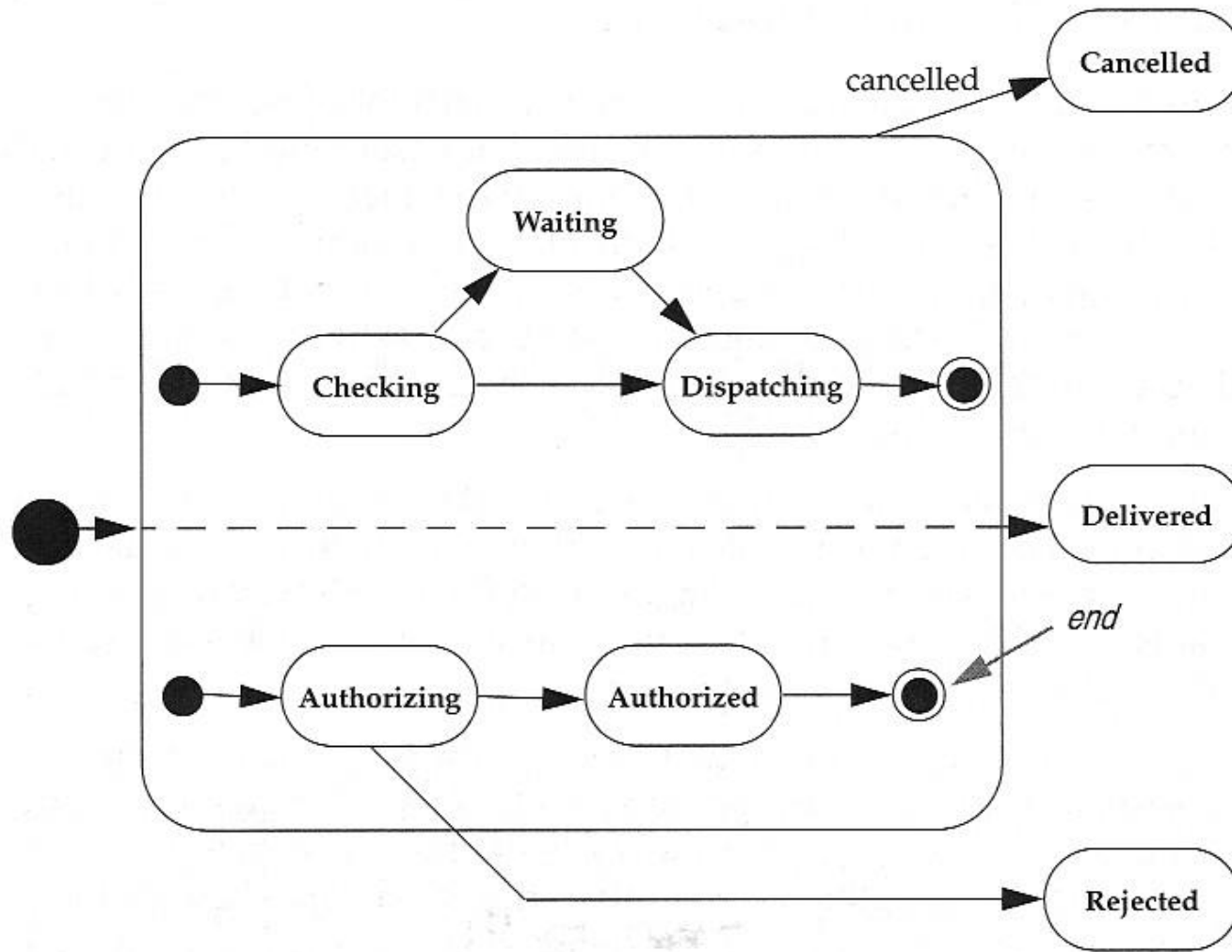


- Action – processes associated with transitions that occur quickly and are not interruptible
- Activity – processes associated with states that may take a while and may be interrupted by events
- Event – a stimulus that causes a transition or a self-transition to take place from one state to another
- Guard – a logical condition that returns "true" or "false"
- Superstate – a state that is itself a collection of states

Example: State Diagrams



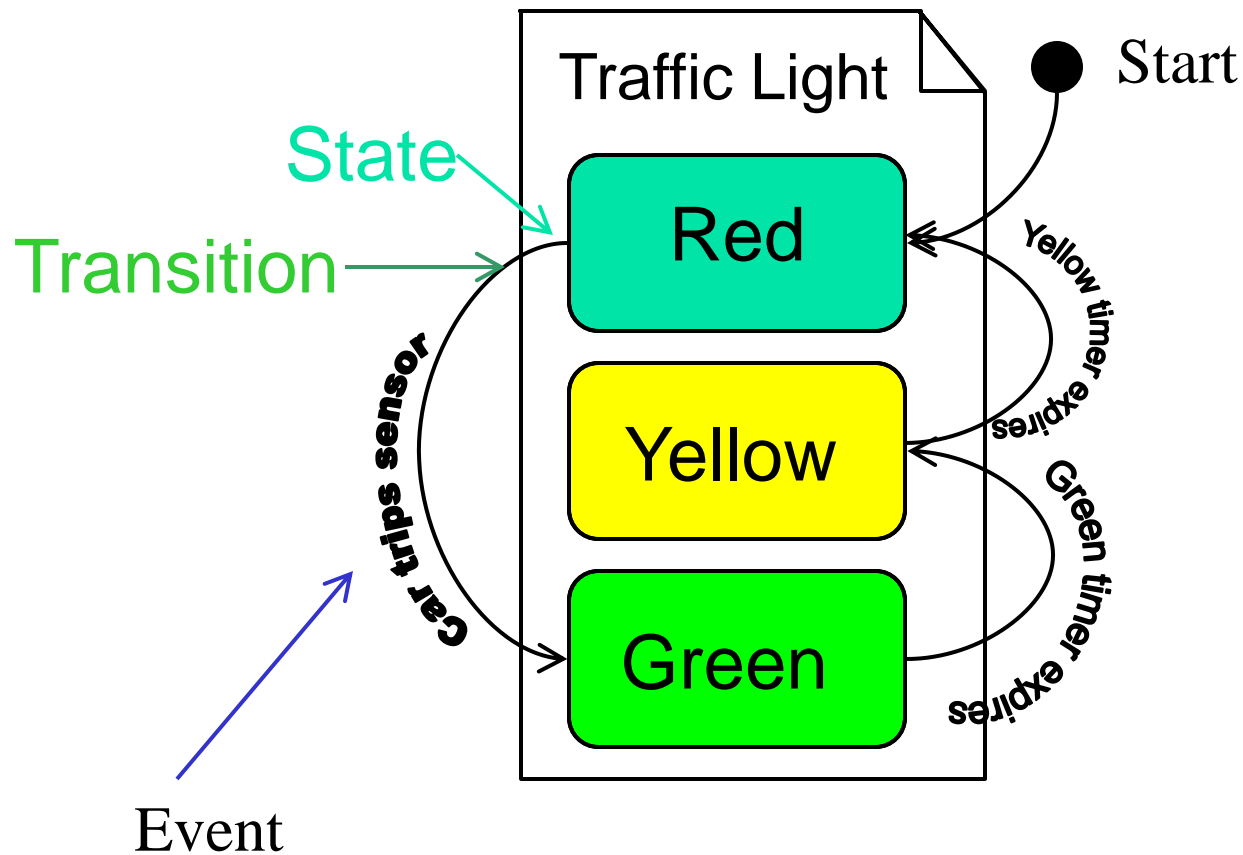
State Diagrams – Example 2



*Concurrent
State Diagram*

State Diagrams

(Traffic light example)



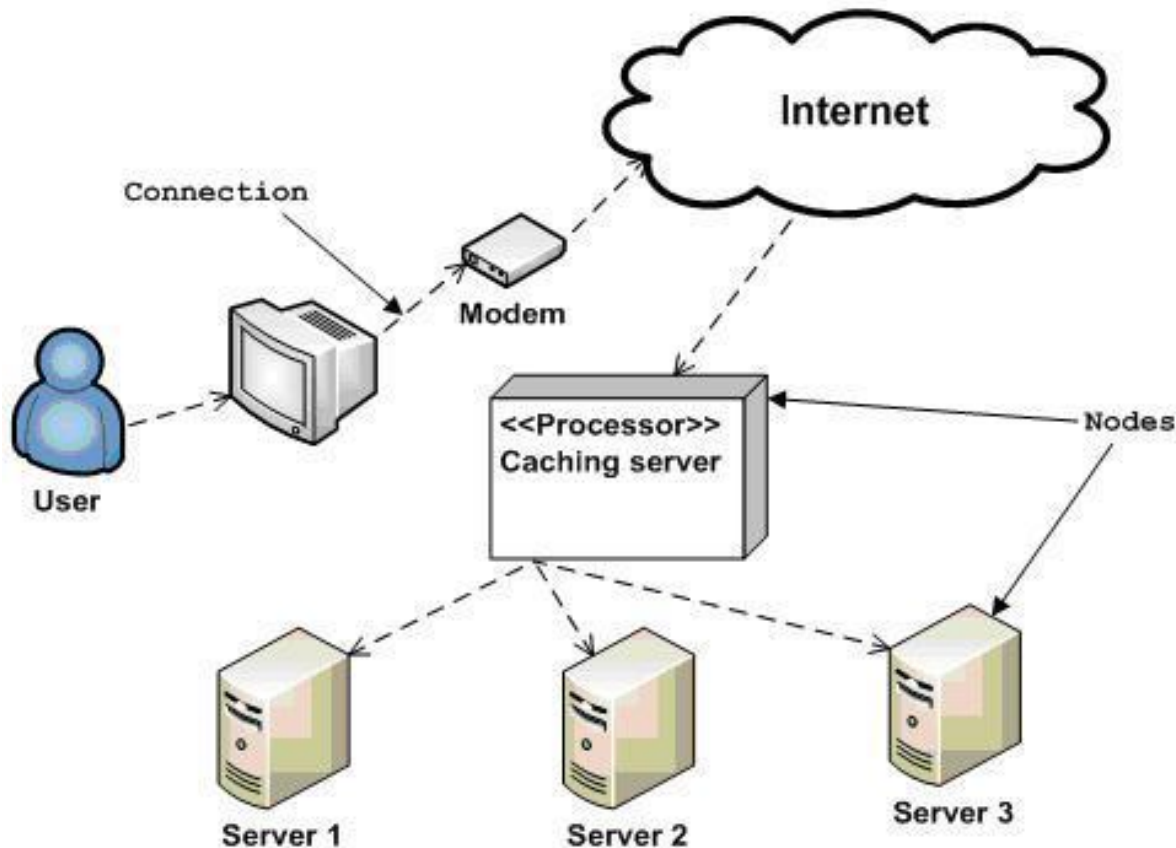
Deployment Diagram

- Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related.
- Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.
- 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.

- 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 artifacts should be identified:
 - Nodes
 - Relationships among nodes

Example Deployment Diagram

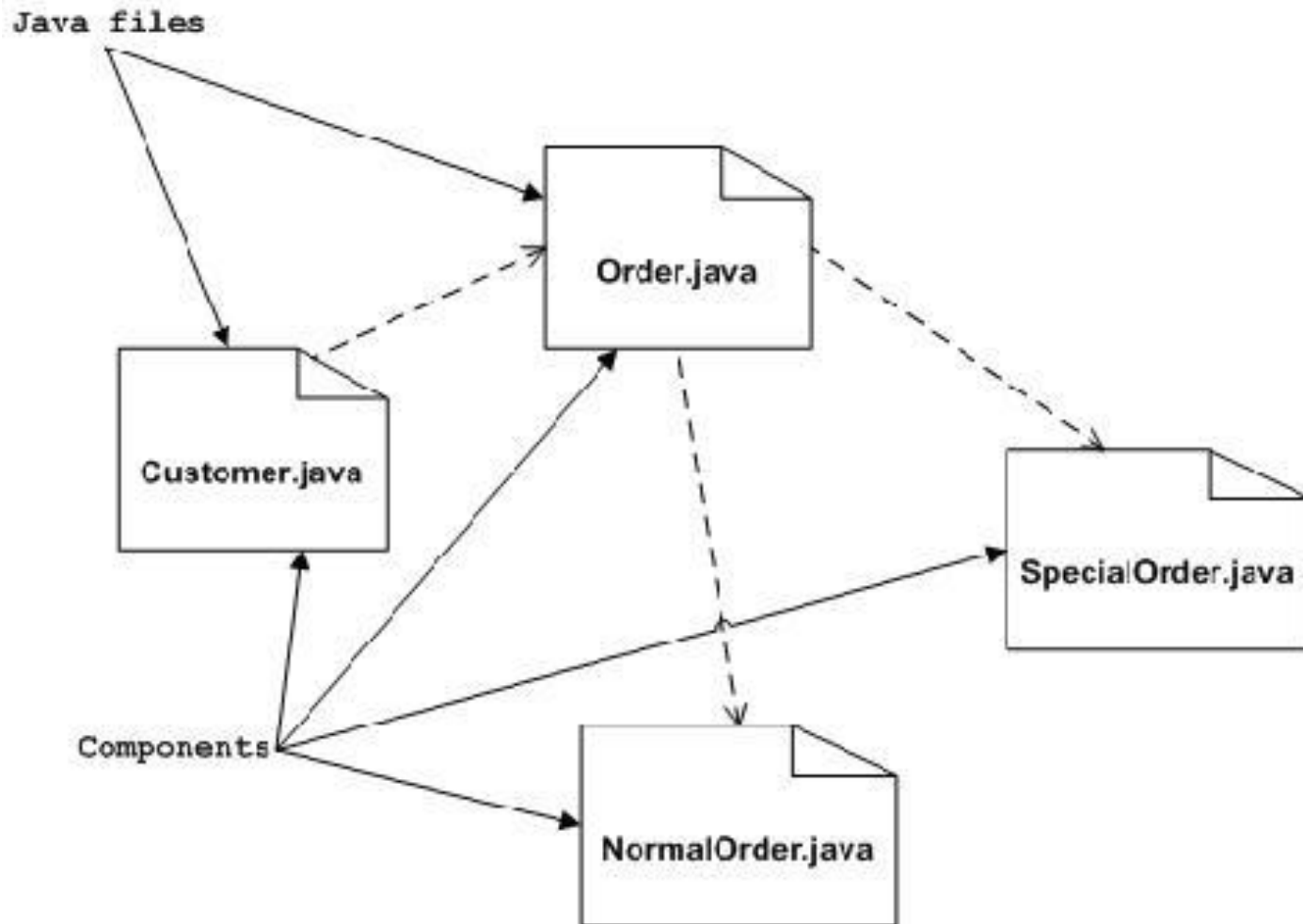
Deployment diagram of an order management system



The application is assumed to be a web based application which is deployed in a clustered environment using server 1, server 2 and server 3. The user is connecting to the application using internet. The control is flowing from the caching server to the clustered environment

Component Diagram

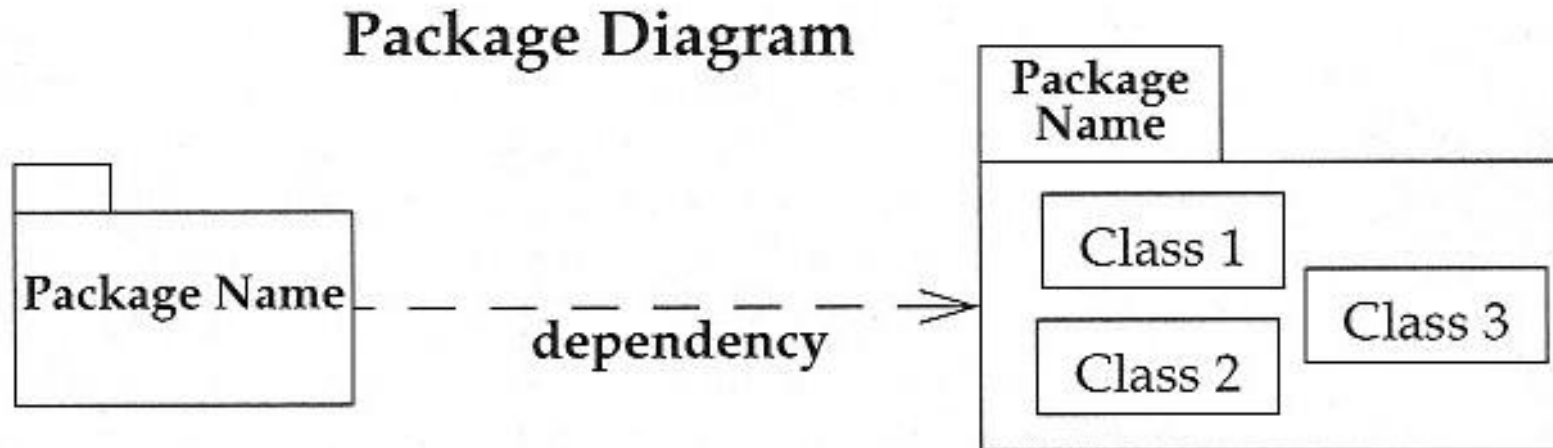
Component diagram of an order management system



What is a Package?

- Package – a grouping of classes (a conventional Package - a unit above a class in the abstraction hierarchy) and other packages (a Domain Package)
- Package Diagram – a UML diagram that shows packages of classes and the dependencies among them
 - A dependency exists between two elements if changes to the definition of one element may cause changes to the other
- Classes have dependencies for several reasons, including:
 - One class sends a message to another
 - One class has another as part of its data
 - One class mentions another as a parameter to an operation

Package Diagrams - Notation



- Package – contains classes
- Dependency – changes to the definition (interface) of one package may cause changes in the other package

Package Diagrams – Example

