

CptS 487

Software Design and Architecture

Lesson 9

Sequence Diagrams

Outline

- Sequence Diagrams
- Optional reading on other UMLs

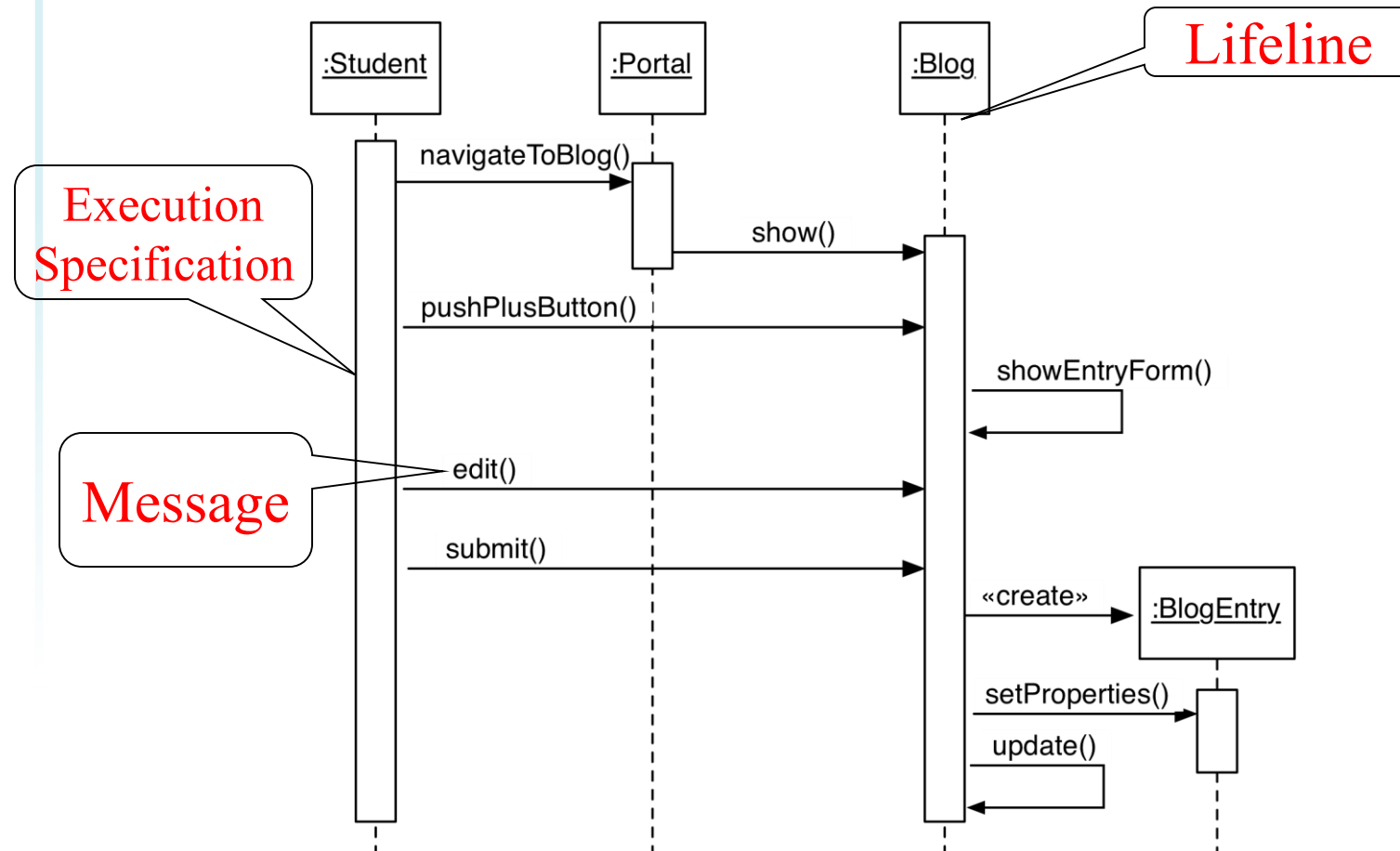
UML Interaction Diagrams

- Interaction diagrams describe patterns of communication among a set of interacting objects
- An interaction diagram typically captures the behavior of a single use case
- Interaction diagrams usually do not capture the complete behavior of a use case, only typical scenarios

UML Interaction Diagrams

- Two types of interaction diagrams:
 - Sequence Diagram:
 - Describes the dynamic behavior *between* several objects over time
 - Good for real-time specifications.
 - Communication Diagram:
 - Shows the temporal relationship among objects
 - Position of objects is identical to the position of the classes in the corresponding UML class diagram
 - Good for identifying the protocol between objects
 - Does not show time

Sequence diagram: Basic Notation



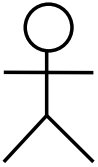
Sequence diagrams represent the behavior of a system as messages (“interactions”) between *different objects*.

Sequence Diagrams

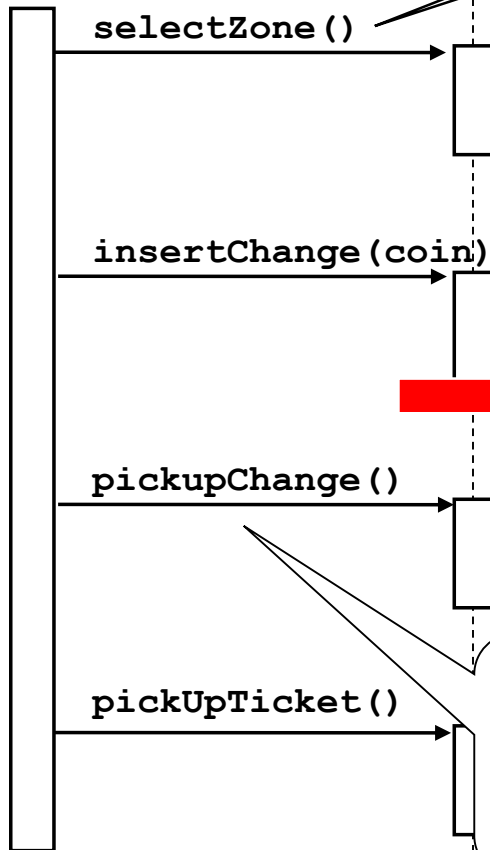
- A **lifeline** represents an individual participant (or object) in the interaction
- An **execution specification** specifies a behavior or interaction within the lifeline
- An execution specification is represented as a thin rectangle on the lifeline.
- **Messages** define a particular communication between lifelines of an interaction
- Examples of communication
 - raising a signal
 - invoking an operation
 - creating or destroying an instance

Sequence Diagrams

Focus on
control flow


Passenger

TicketMachine



Messages ->
Operations on
participating Object

Used during analysis

- to refine use case descriptions
- to find additional objects (“participating objects”)

Used during system design

to refine subsystem interfaces

Processes are represented by rectangles. **Actors** by sticky

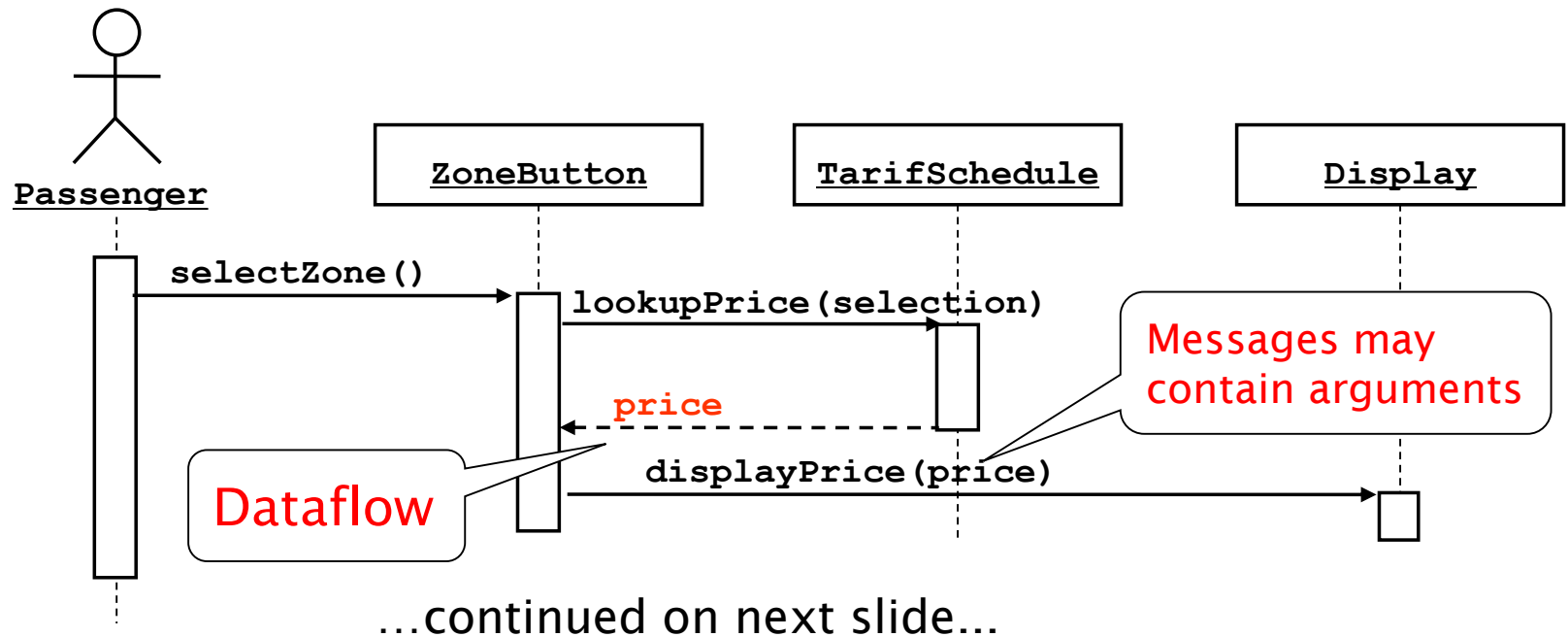
Objects are represented by

dashed lines

Messages are represented by
arrows

Activations are represented by
narrow rectangles.

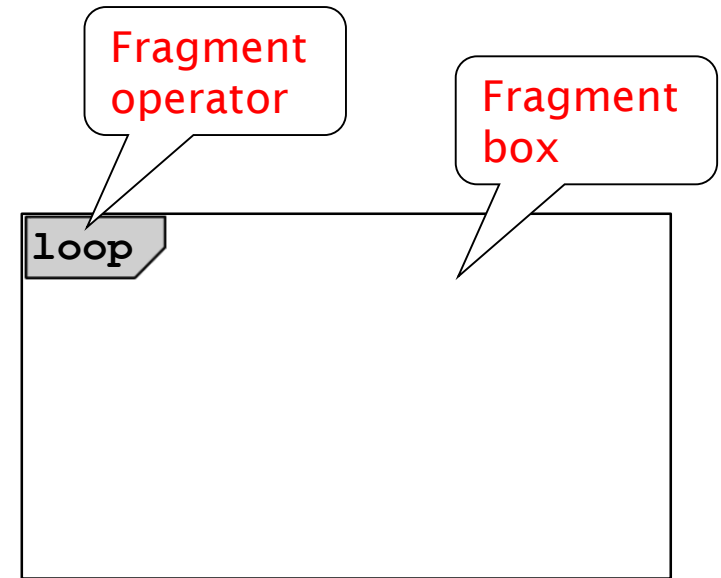
Sequence Diagrams can also model the Flow of Data



- The source of an arrow indicates the activation which sent the message
- **Arguments** may be passed along with a message and are bound to the parameters of the executing method in the receiving object.
- **Horizontal dashed arrows indicate data flow**, for example return results from a message

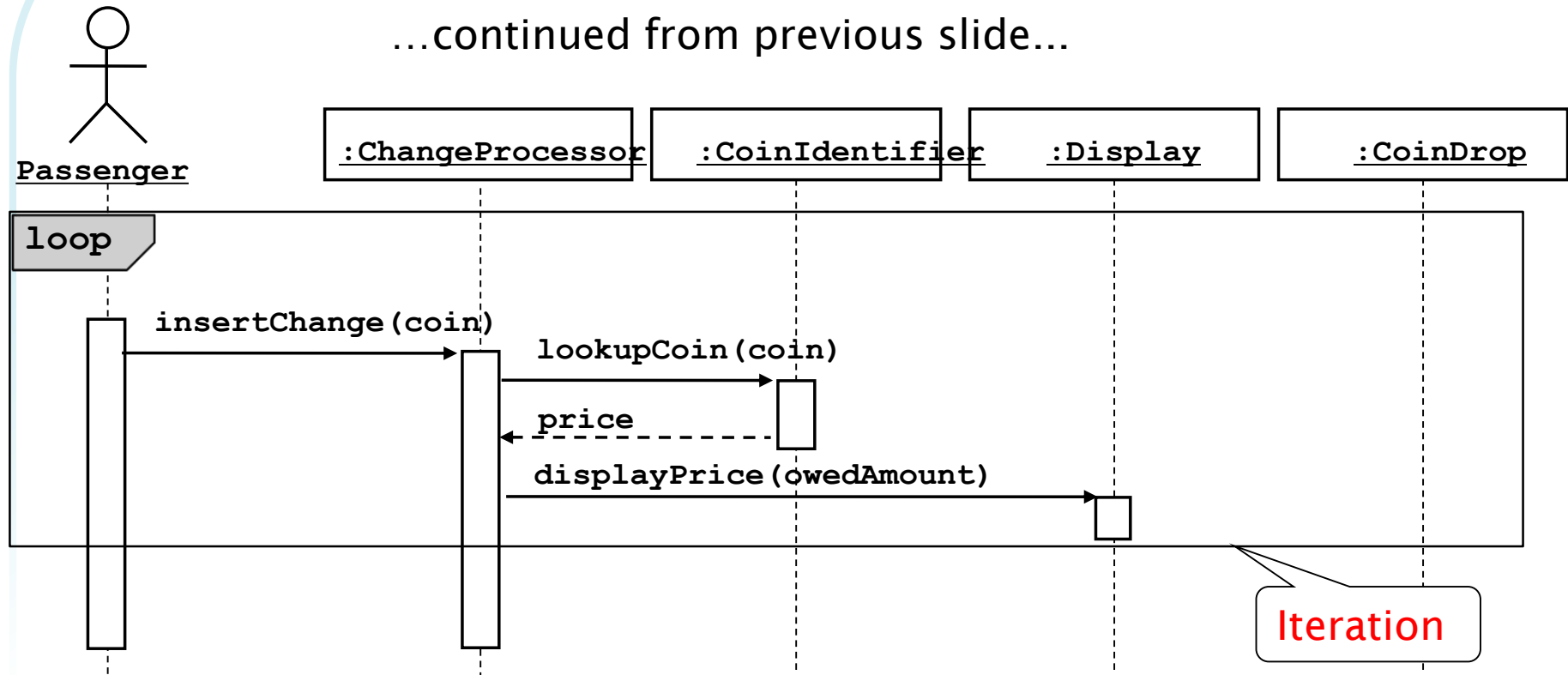
Sequence Diagrams: Combined Fragments

- UML 2.0 introduces combined fragments
- A combined fragment is used to group sets of messages together to show conditional flow in a sequence diagram
- The fragment operator (in the top left corner) indicates the type of fragment
- **Fragment types:** `ref`, `assert`, `loop`, `break`, `alt`, `opt`, `neg`



Sequence Diagrams: Combined Fragments

...continued from previous slide...

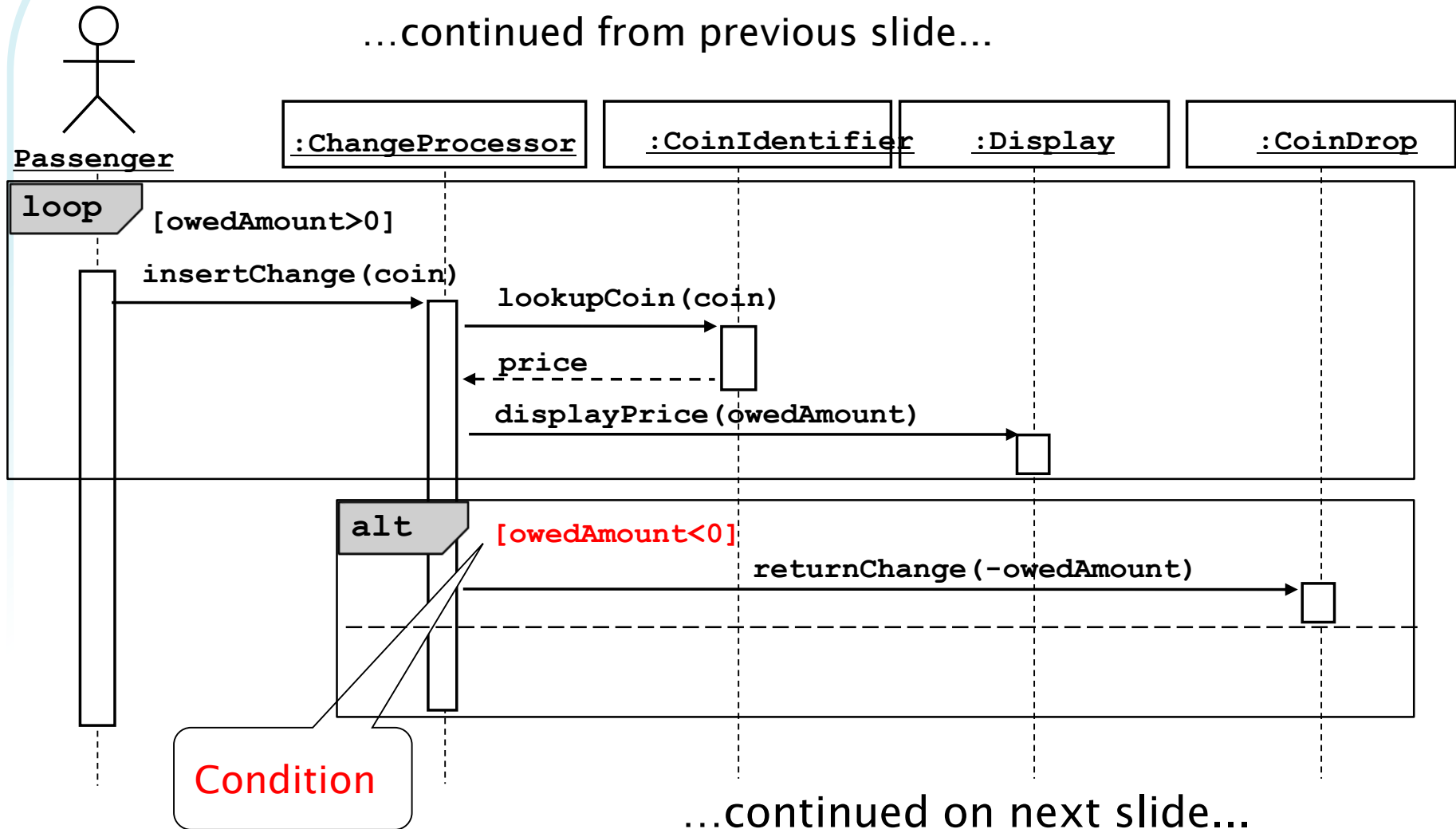


...continued on next slide...

- The “loop” combined fragment models the iterations, i.e., repeating sequences.

Sequence Diagrams: Combined Fragments

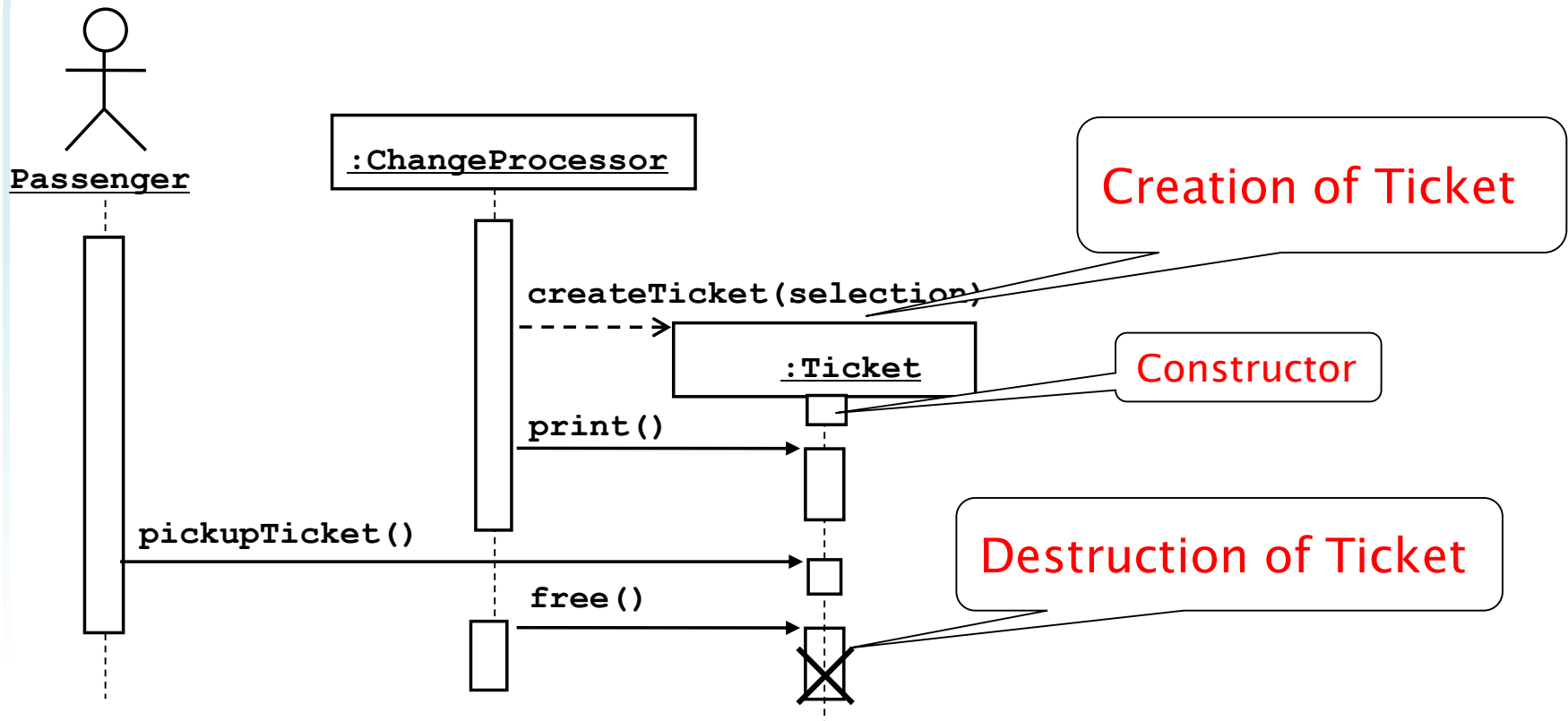
...continued from previous slide...



- Alternatives (“**alt**”) are used to designate a mutually exclusive choice between two or more message sequences

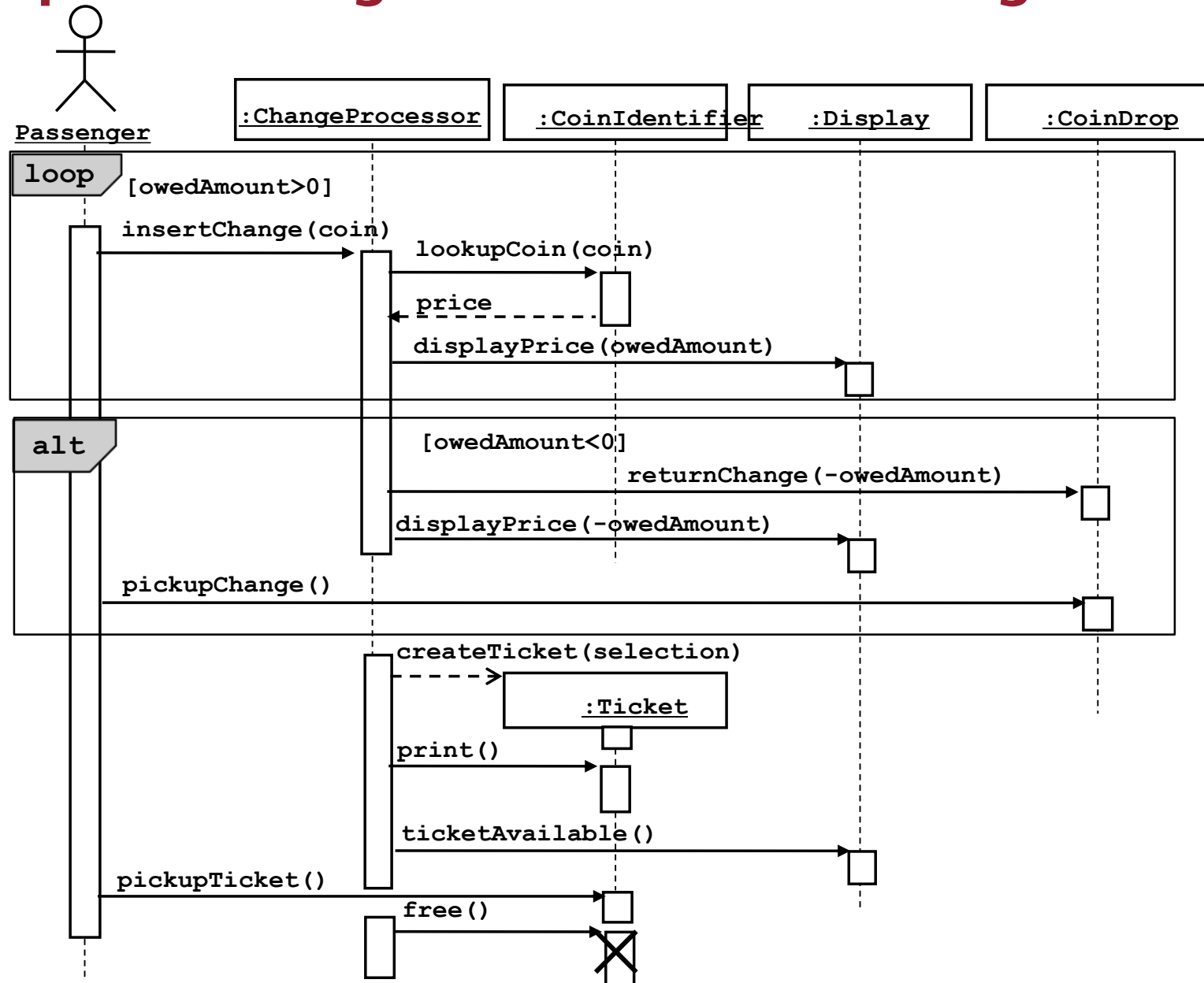
Sequence Diagrams: Creation and Destruction

...continued from previous slide...







- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
- In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Sequence Diagrams: Combined Fragments

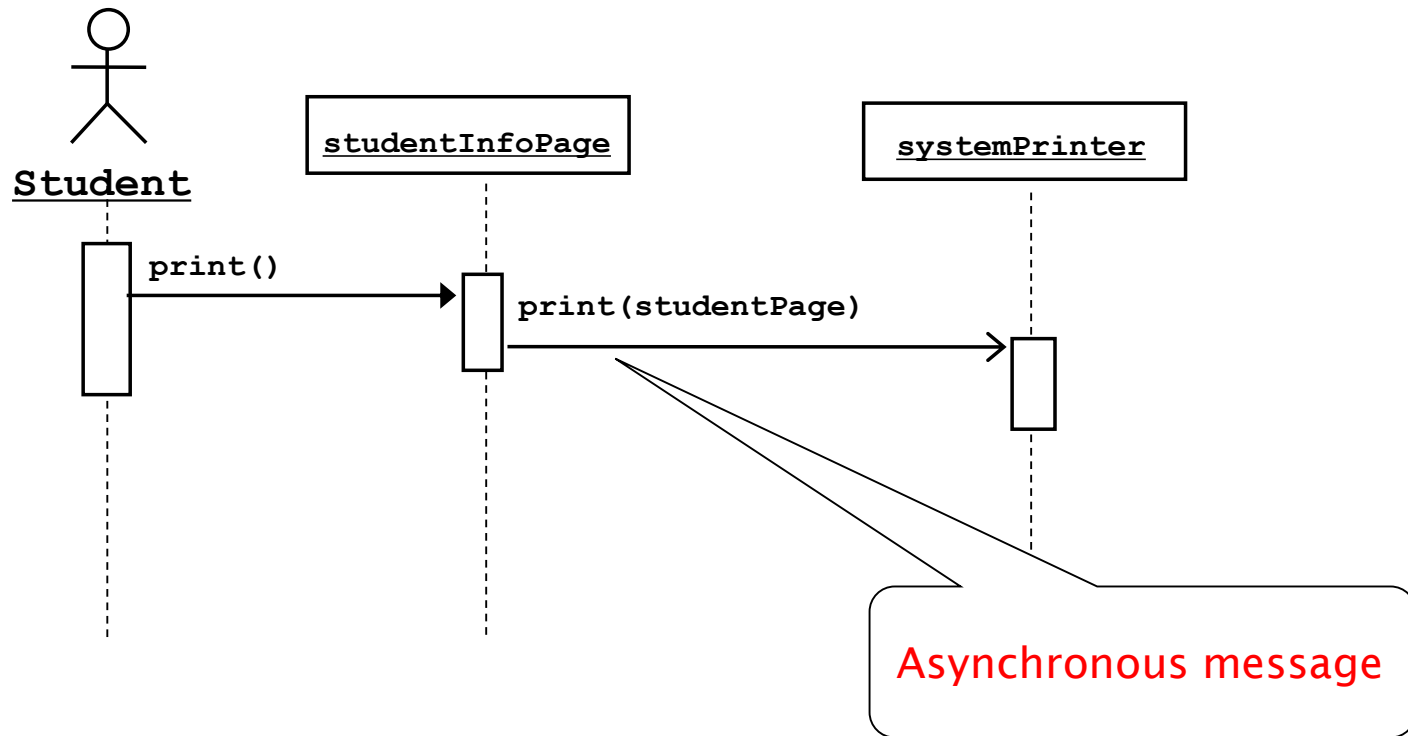


Sequence Diagrams: Message Type Notation

			
Synchronous or Call	Asynchronous	Creation	Reply (Return) Data Flow

Sequence Diagrams: Message Type Notation

- Asynchronous messages:



How to Produce Sequence Diagrams

- Decide on Context: Identify behavior (or use case) to be specified
- Identify structural elements:
 1. Model objects (classes)
 2. Model lifelines
 3. Model activations
 4. Model messages
 5. Model Timing constraints
- Refine and elaborate as required

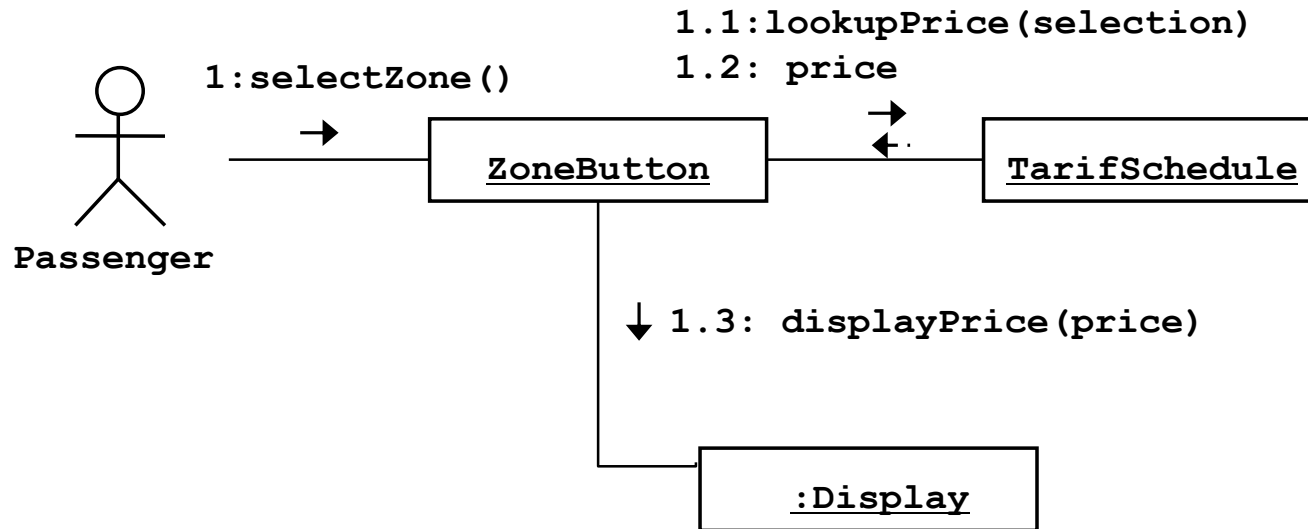
Rules of Thumb for Sequence Diagram

- Typically, one sequence diagram for each use case.
 - Start with an actor (on the left)
 - Actor would interact with the boundary classes (GUI elements)
 - Boundary classes pass information along to (newly created) control classes.
 - Control classes dictate the entity classes involved.
 - (Optional) Return messages are passed back all the way to the boundary classes for display.
- Note: sequence diagrams help uncover new/necessary operations for the classes.

Reference

- <https://msdn.microsoft.com/en-us/library/dd409377.aspx>

Communication Diagrams



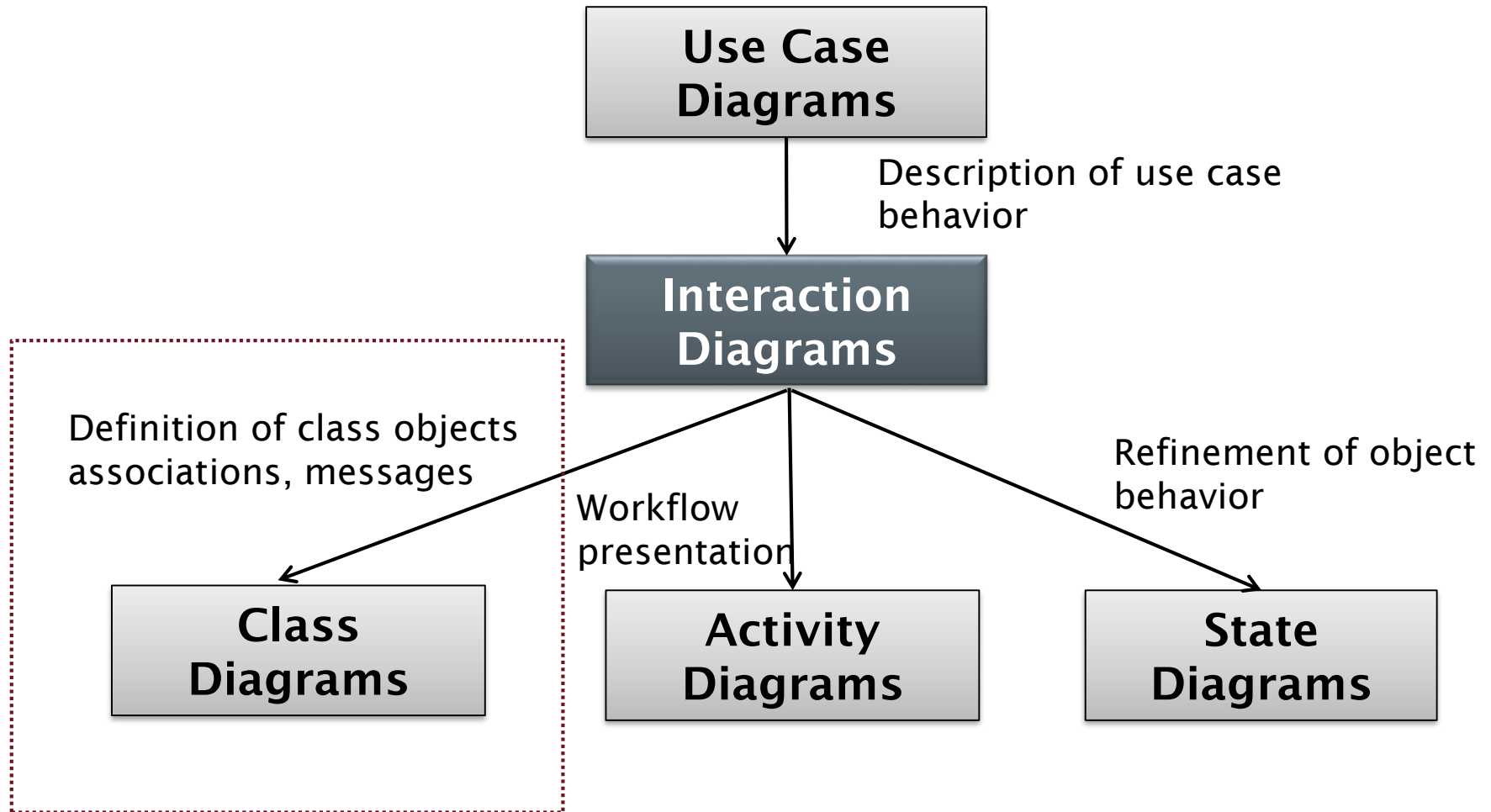
- **Communication diagrams** depict the same information as sequence diagrams .
- Emphasize the flow of messages among objects, rather than timing and ordering of messages
- Represent sequence of messages by numbering the interactions.
- More compact diagram, however more difficult to follow

How do interaction diagrams help?

- Check use cases: Useful to identify or find missing objects in use cases
- Decide which objects require particular operations
- Complement the class diagrams (which represent structure).
- Time consuming to build, but worth the investment

Optional Contents

How do interaction diagrams help?

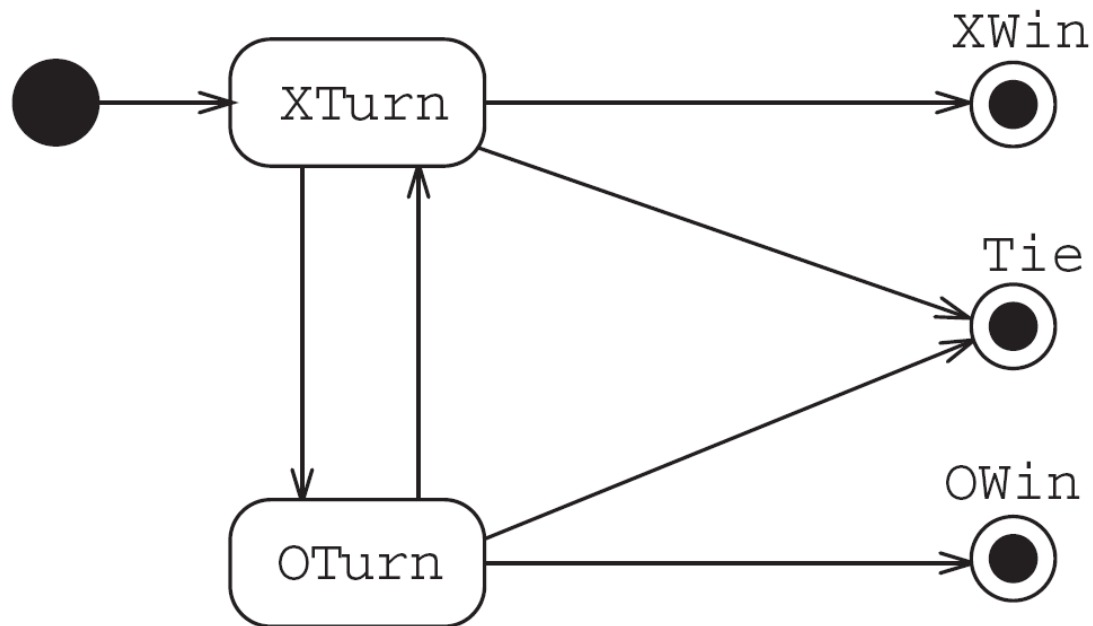


UML State Machine Diagrams

- A state diagram describes the sequence of states an object goes through in response to external events.
- At any given point in time, the system or object is in a certain state.
 - Being in a state means that it will behave in a specific way in response to any events that occur.
- Some events will cause the system to change state.
 - In the new state, the system will behave in a different way to events.
- A state diagram is a directed graph where the nodes are states and the arcs are transitions.

UML State Machine Diagrams – An Example

- tic-tac-toe game

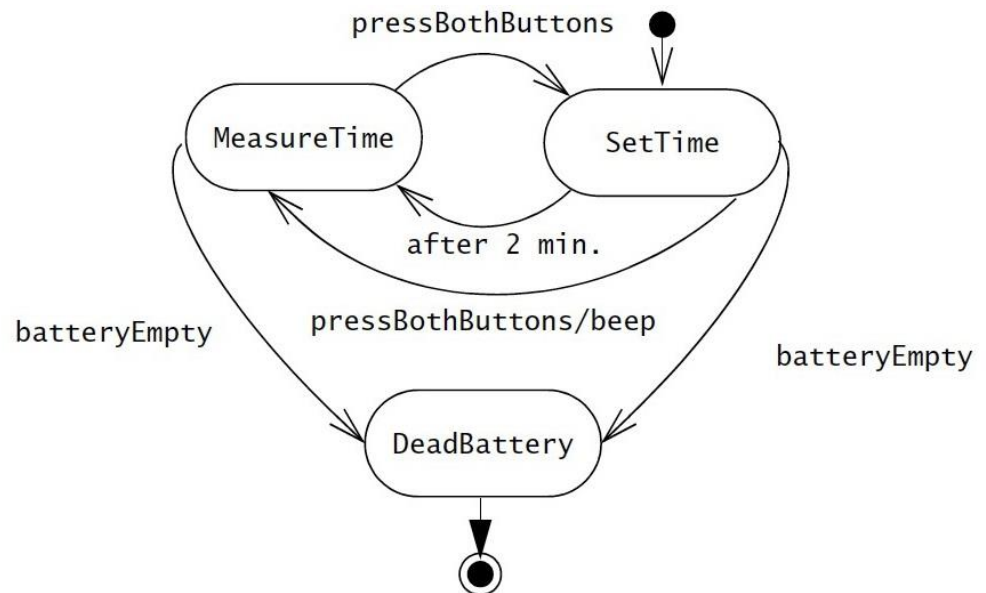


States

- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.
- A state is represented by a rounded rectangle containing the name of the state.
- Special states:
 - A black circle represents the start state
 - A circle with a ring around it represents an end state

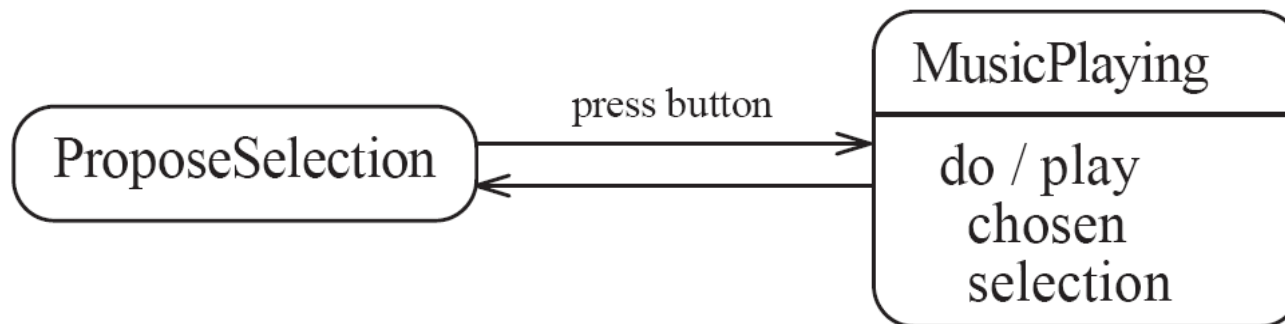
Transitions

- A transition represents a change of state in response to an event.
- It is considered to occur instantaneously.
- The label on each transition is the event that causes the change of state
 - Event, conditional and time transitions
- State machine diagram for “set time” function of the Watch class



Activities in State Machine Diagrams

- An activity is something that takes place while the system is in a state.
- It takes a period of time.
- The system may take a transition out of the state in response to completion of the activity,
- Some other outgoing transition may result in:
 - The interruption of the activity, and
 - An early exit from the state.

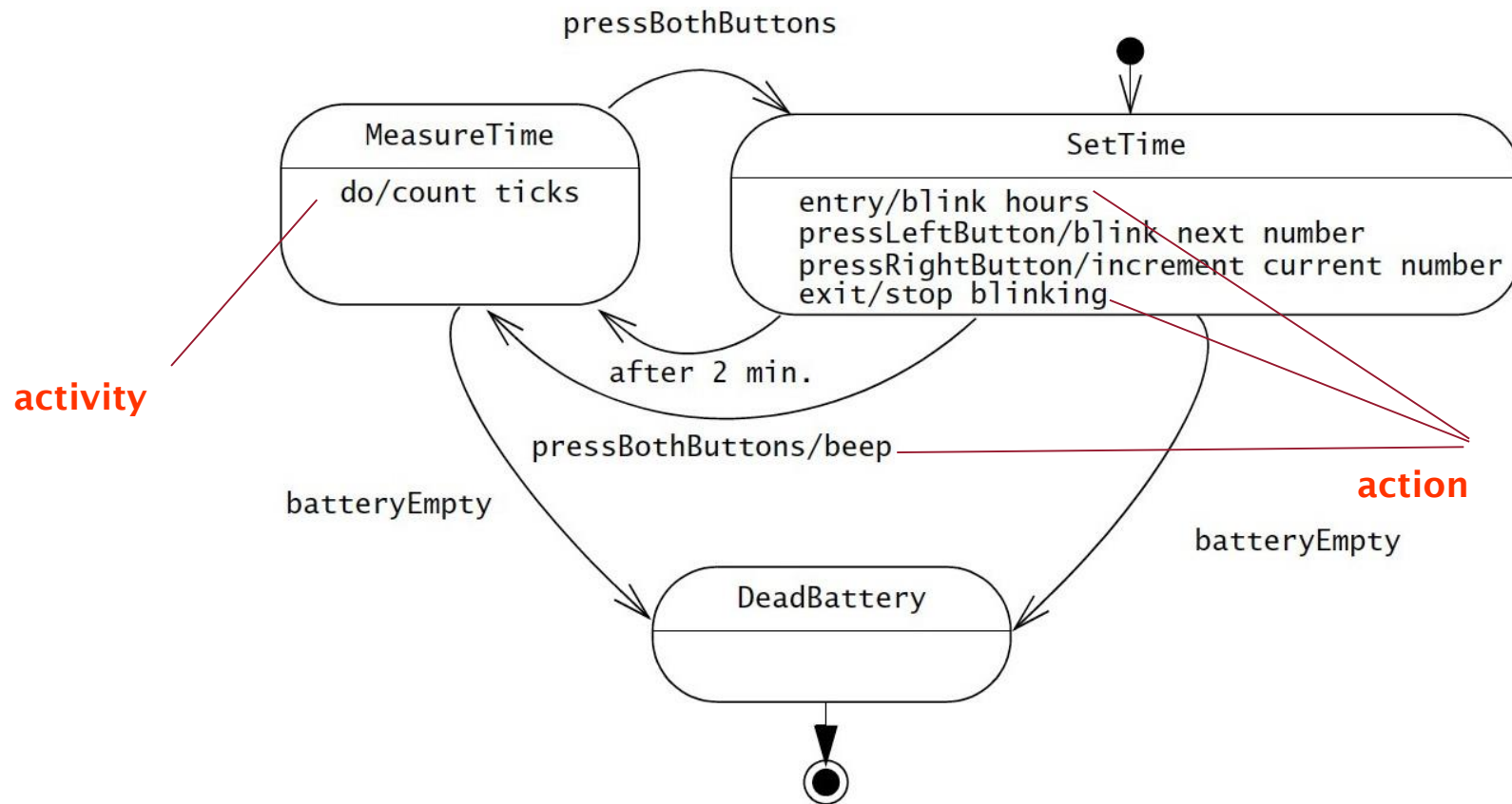


Actions in State Machine Diagrams

- Actions denote the behavior within the states or during transitions between states
- Actions take short amount of time to execute and are not interruptable
- Actions can occur in three places:
 - When a particular transition is taken,
 - Upon entry into a particular state, or
 - Upon exit from a particular state

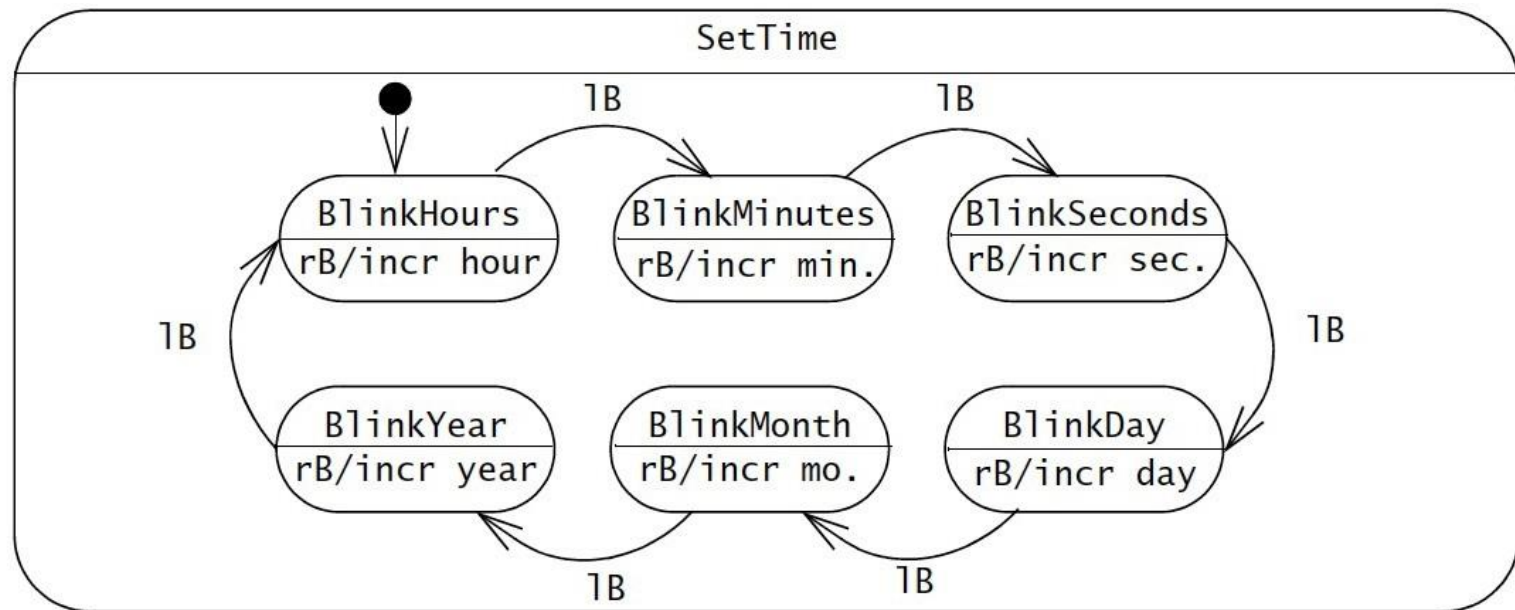
State Machine Diagram – example with actions

- An example state diagram with actions and activities



Nested Substates

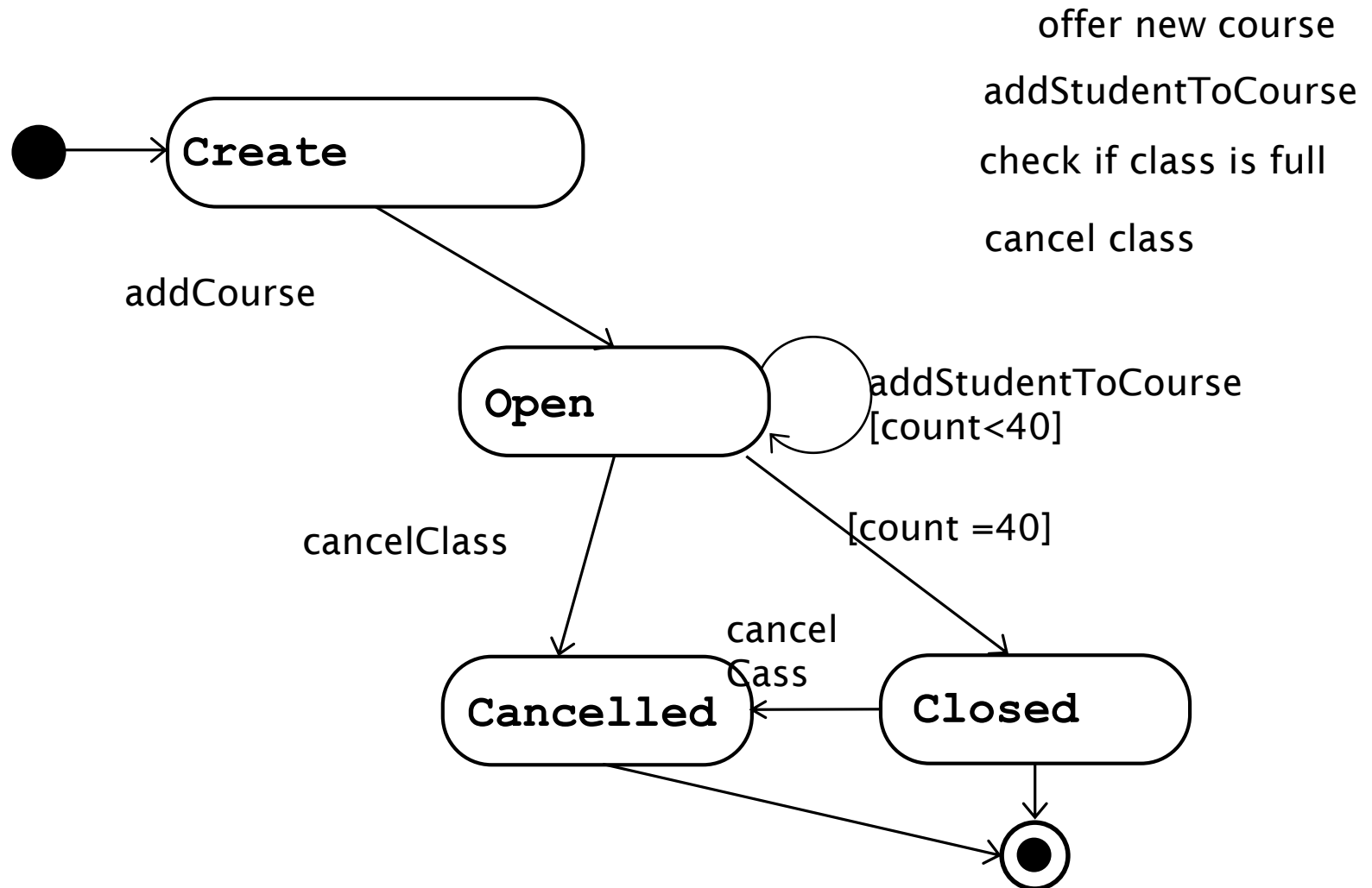
- A state diagram can be nested inside a state.
 - The states of the inner diagram are called substates.
- Nested state machine diagrams reduce complexity



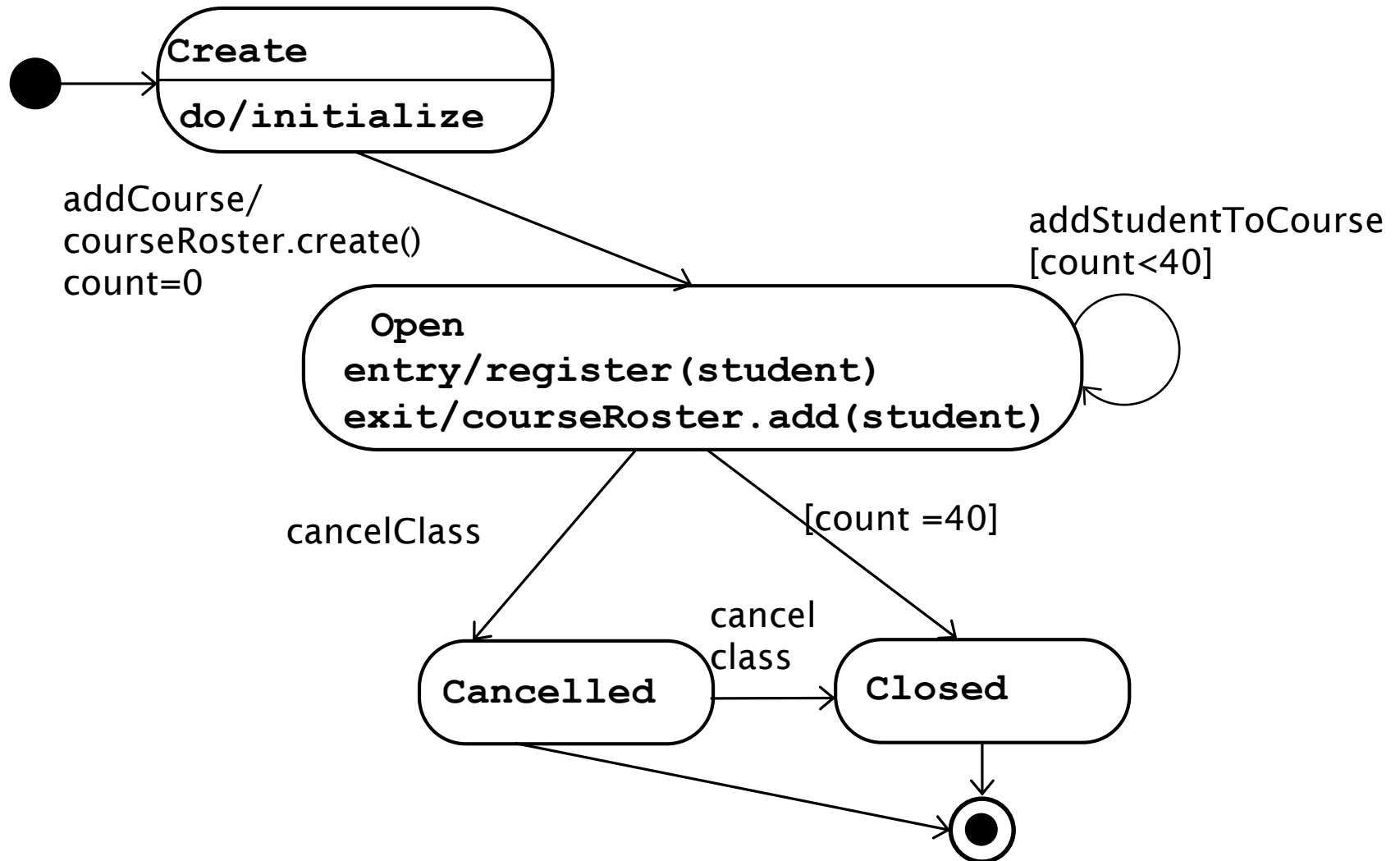
Building State Machine Diagrams

- Identify **entities** that have **complex behavior**, i.e., identify a **class** whose lifecycle is to be specified
- Model **states**-Determine the **initial** and **final** states of the entity
- Model **transitions**
- Model **events**-Identify the events that affect the entity
- Working from the initial state, trace the impact of events and identify intermediate states
- Identify any **entry** and **exit** actions on the states
- Expand states using substates where necessary
- If the entity is a class, check that the actions in the state are supported by the operations and relationships of the class, and if not extend the class
- Refine and elaborate as required

Building State Machine Diagrams-example



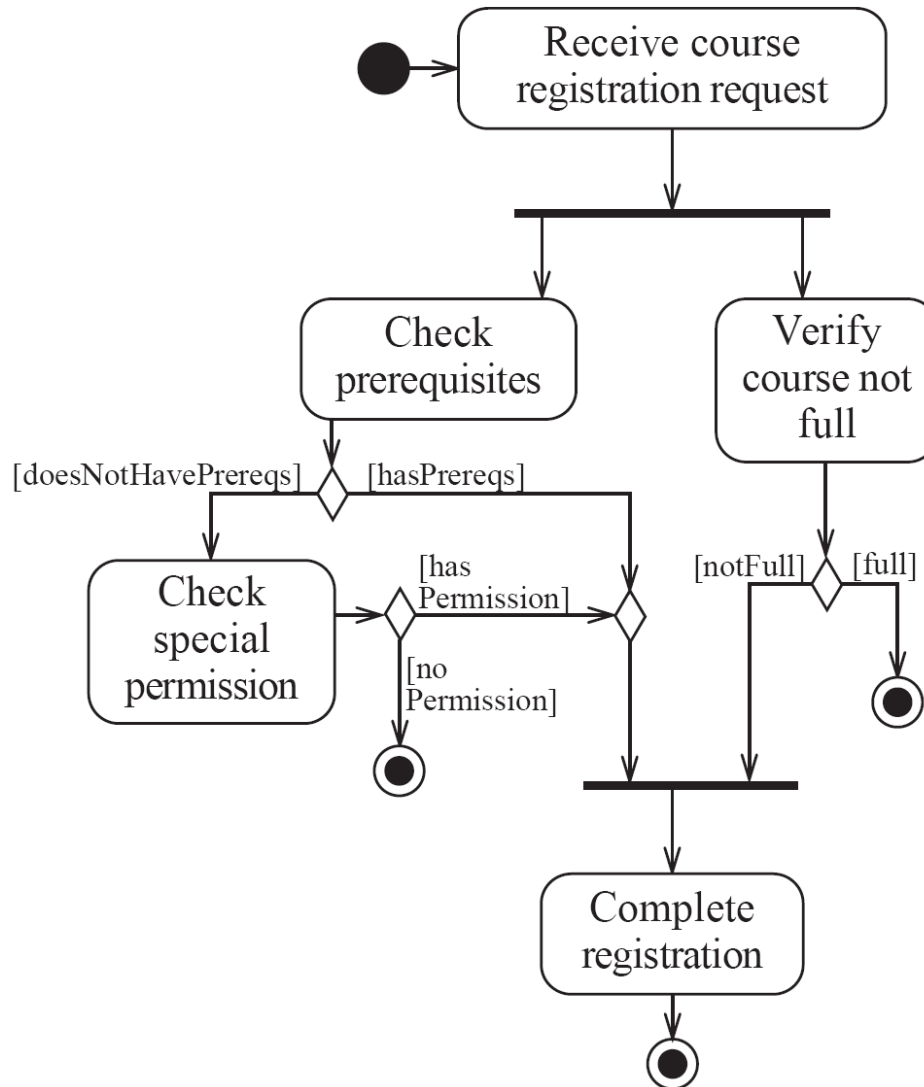
Building State Machine Diagrams-example



Activity Diagrams

- An activity diagram is like a state machine diagram.
 - Except most transitions are caused by internal events, such as the completion of a computation.
- An activity diagram
 - Can be used to understand the flow of work that an object or component performs.
 - Can also be used to visualize the interrelation and interaction between different use cases.
 - Is most often associated with several classes.
- One of the strengths of activity diagrams is the representation of concurrent activities.

Activity Diagrams – an example – Course Registration



UML Activity Diagrams

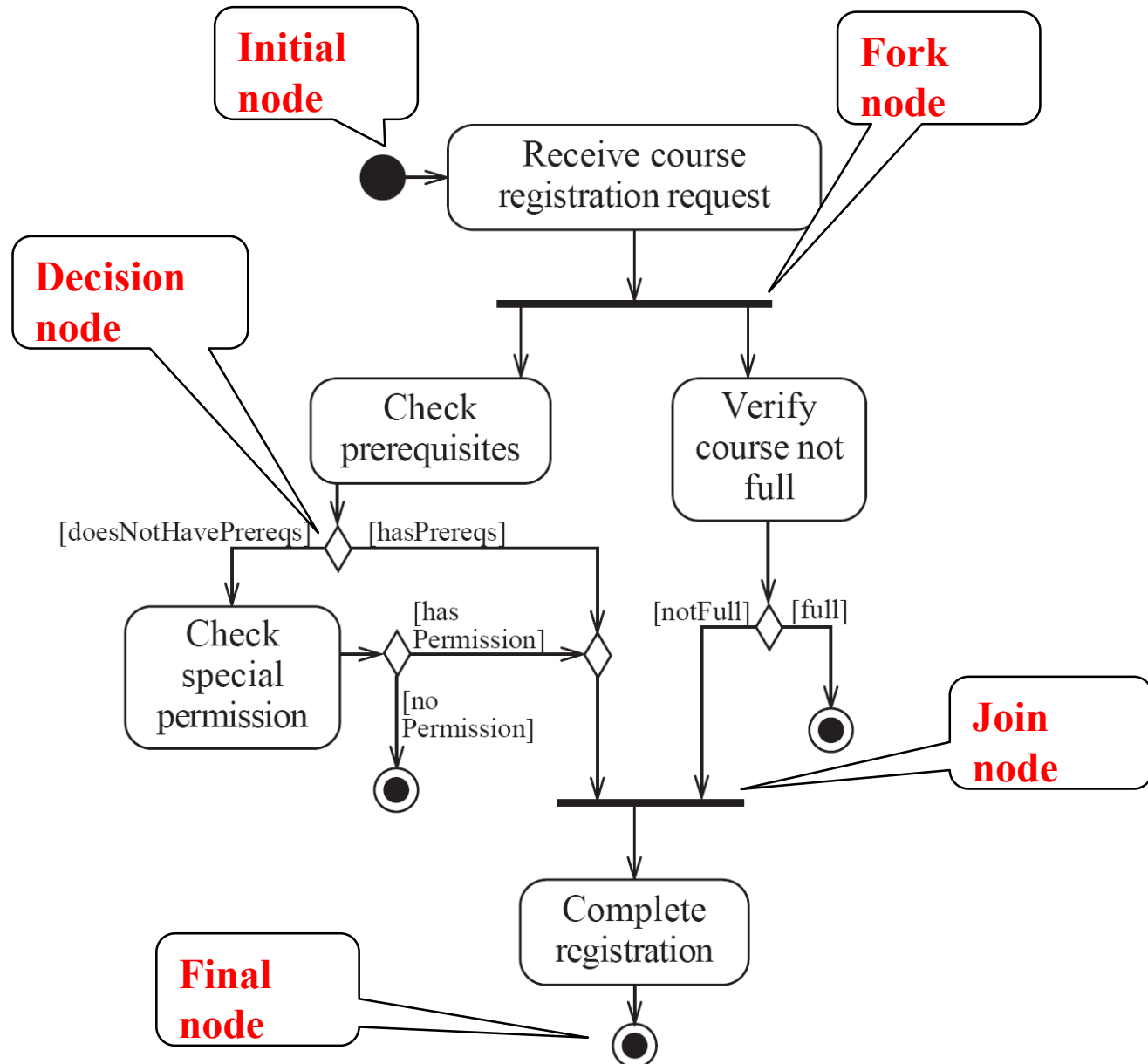
An activity diagram consists of nodes and edges

- **Nodes describe activities and objects**
 - Control nodes
 - **decision** nodes, **fork** nodes, **join** nodes
 - Executable nodes (**Actions**)
 - Object nodes
 - E.g. a document
- **Edge** is a directed connection between nodes

Representing Concurrency

- **Concurrency** is shown using forks and joins
- A **fork** has one incoming transition and multiple outgoing transitions.
 - The execution splits into multiple concurrent threads.
- A **join** has multiple incoming transitions and one outgoing transition.
 - The flow of control is merged into a single thread
 - Join nodes denote the synchronization of multiple threads

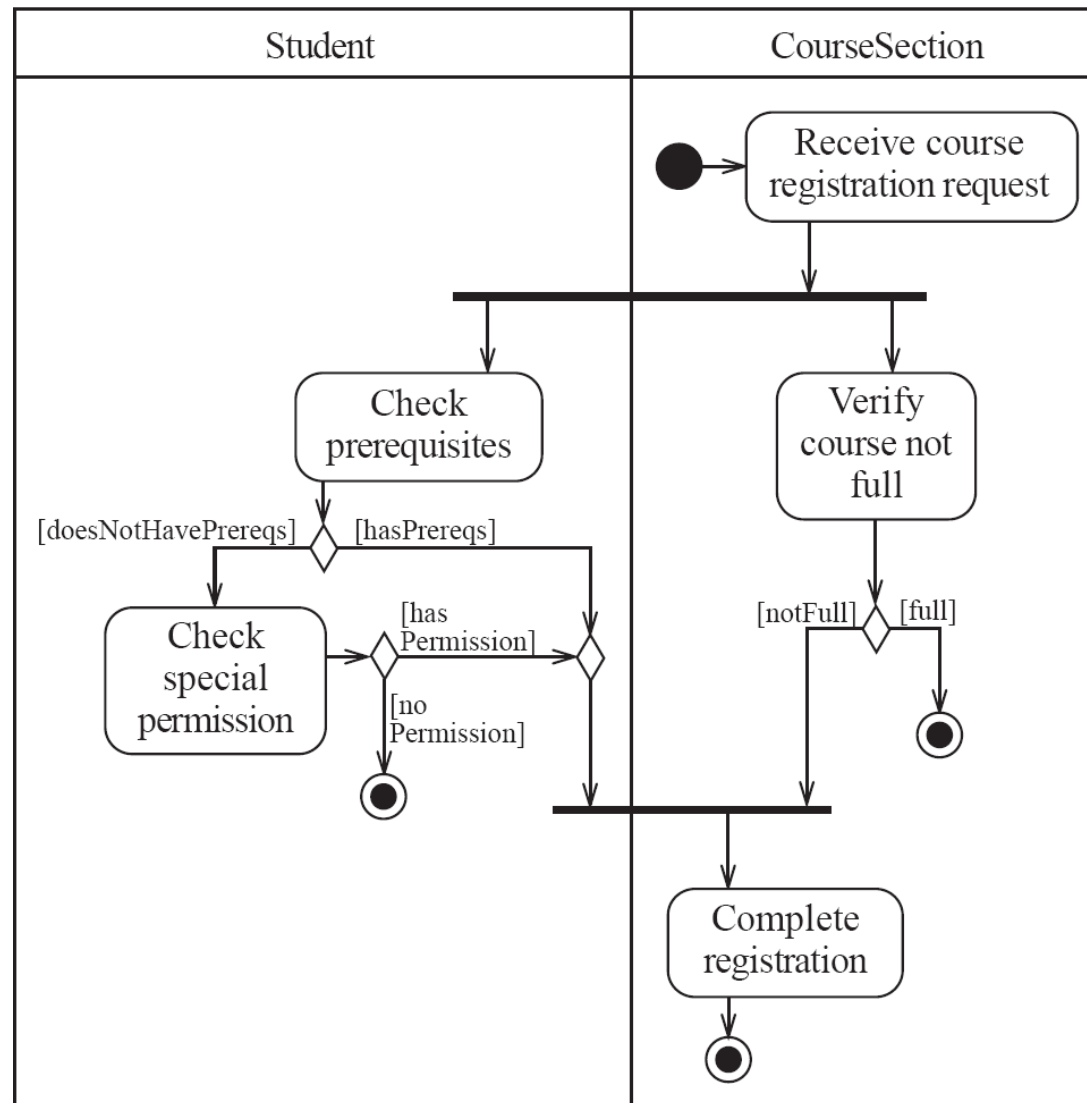
Activity Diagrams - example



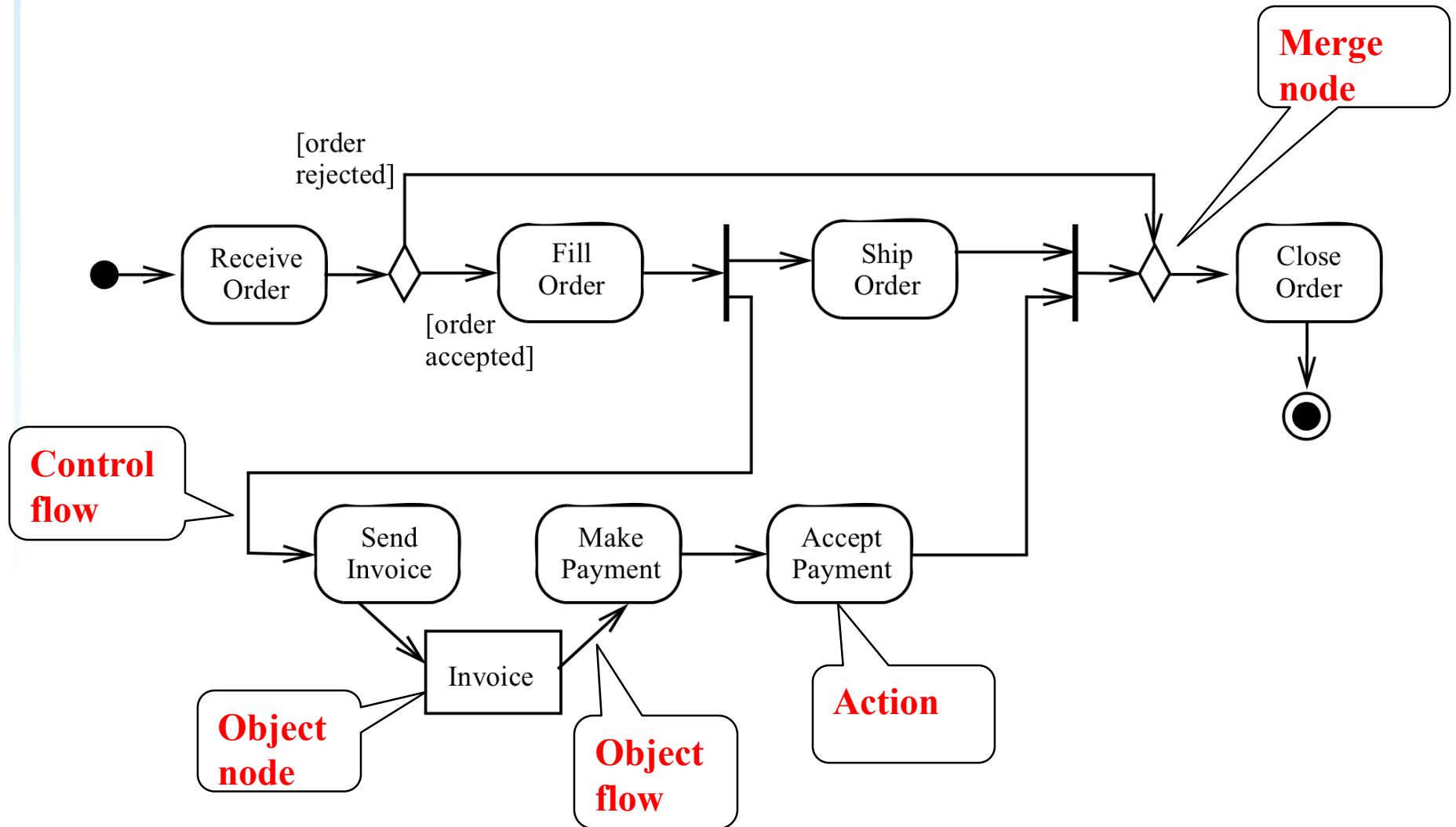
Activity Diagrams: Grouping of Activities

- Activity diagrams are most often associated with several classes.
 - The partition of activities among the existing classes can be explicitly shown using **swimlanes**

Activity diagrams – an example with swimlanes

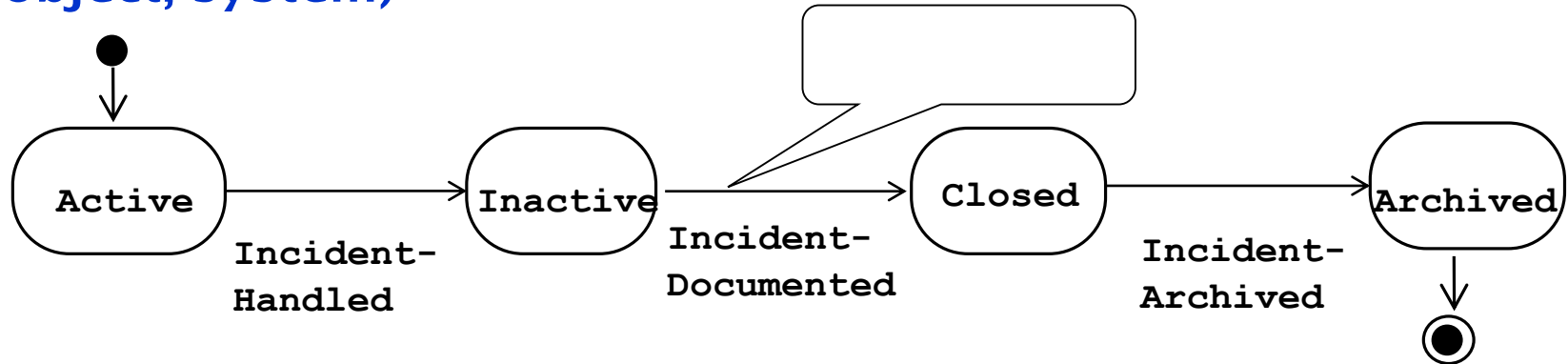


Activity Diagrams – another example

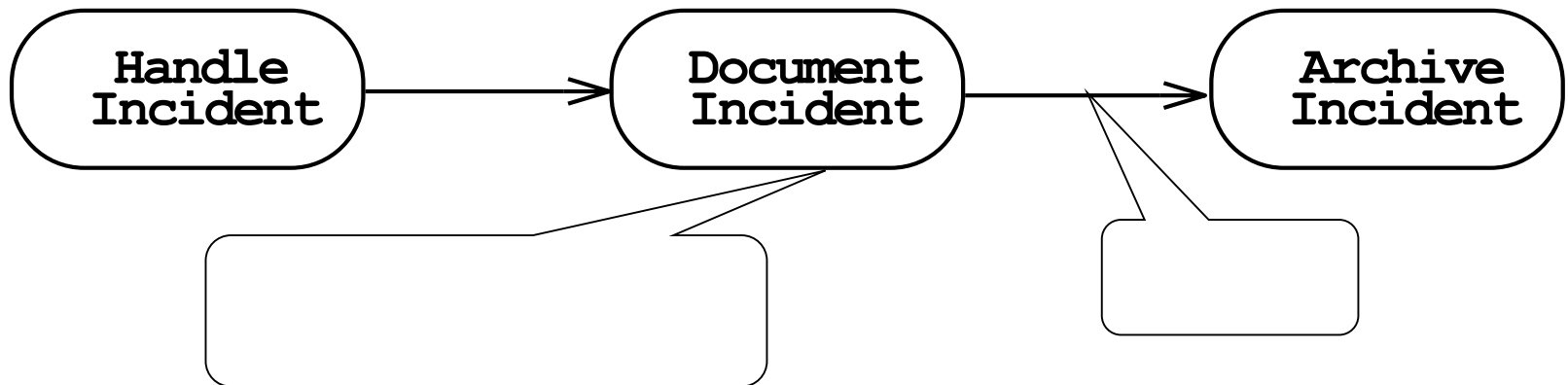


Statechart Diagram vs Activity Diagram

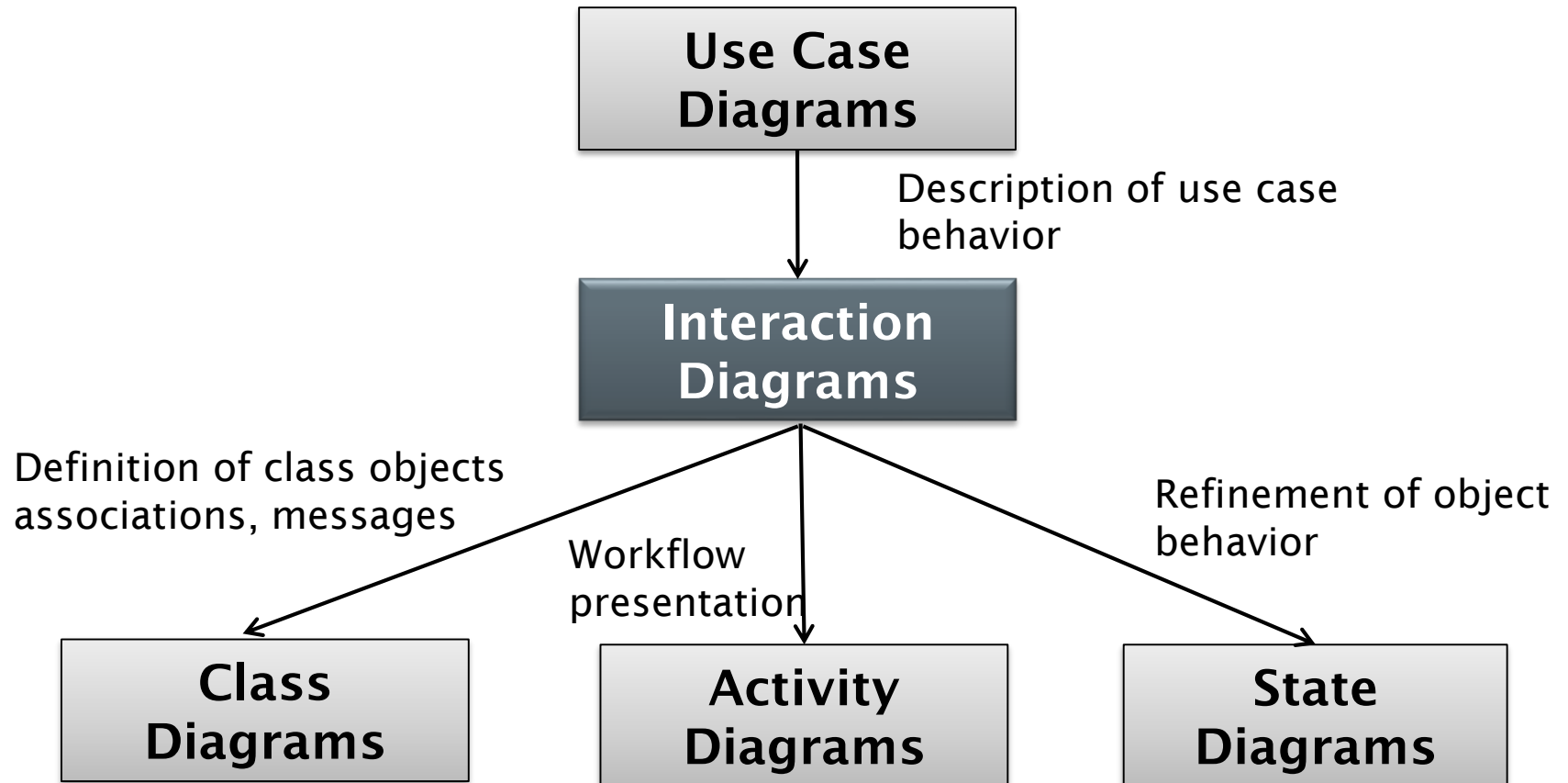
Focus on the set of attributes of a single abstraction
(object, system)



Focus on dataflow in a system



How do interaction diagrams help?



UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
 - Powerful, but complex
- UML is a programming language
 - Can be misused to generate unreadable models
 - Can be misunderstood when using too many exotic features
- We concentrated on a few notations:
 - Functional model: Use case diagram
 - Object model: class diagram
 - Dynamic model: sequence diagrams, statechart and activity diagrams.