Modeling with UML

Lecture 4

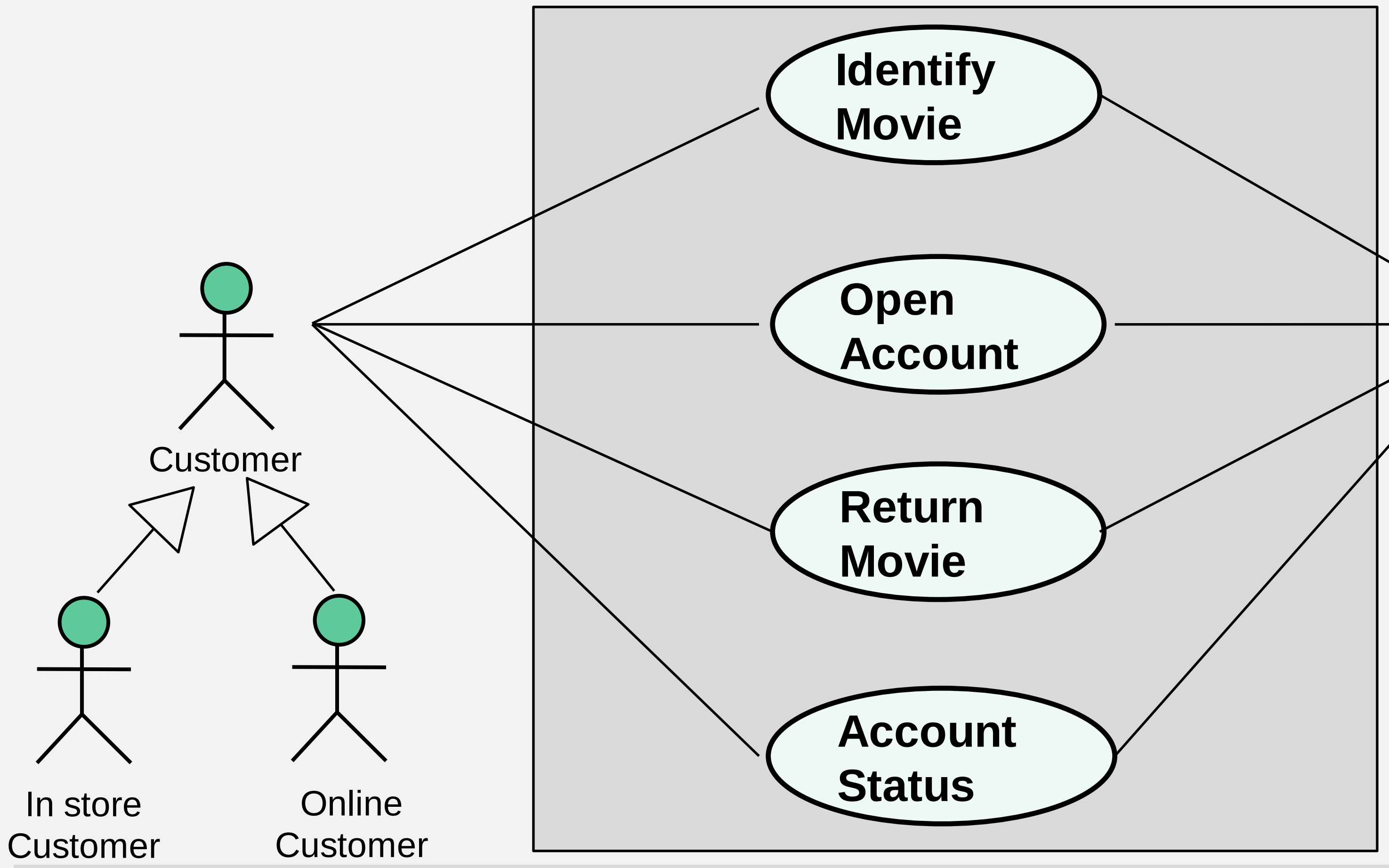
UML Basic Notation Summary

UML provides a wide variety of notations for modeling many aspects of software systems

we concentrated on a few notations:

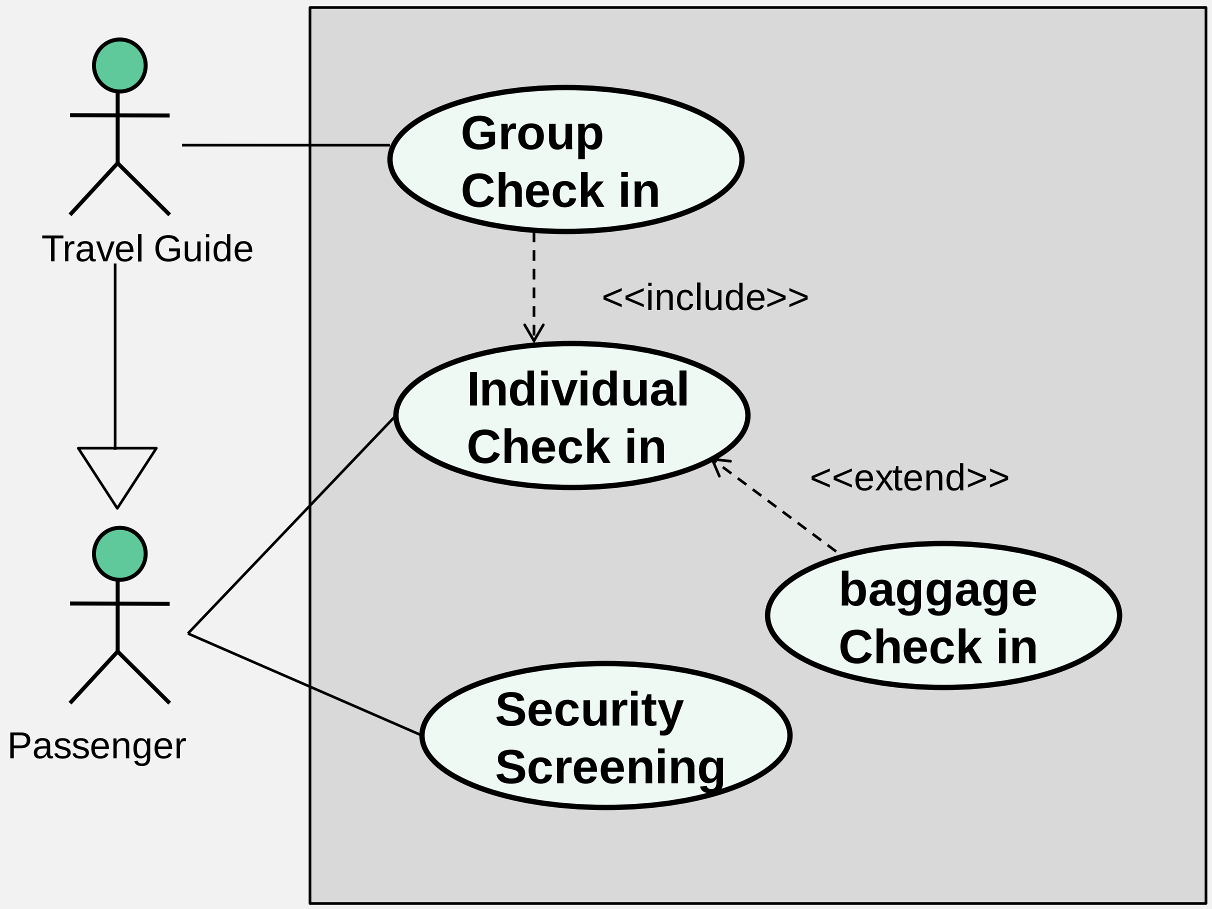
- Functional model: Use case diagram
- Object model: Class diagram
- Dynamic model: Sequence diagrams, activity diagram, statechart

Review of Use Case Diagrams



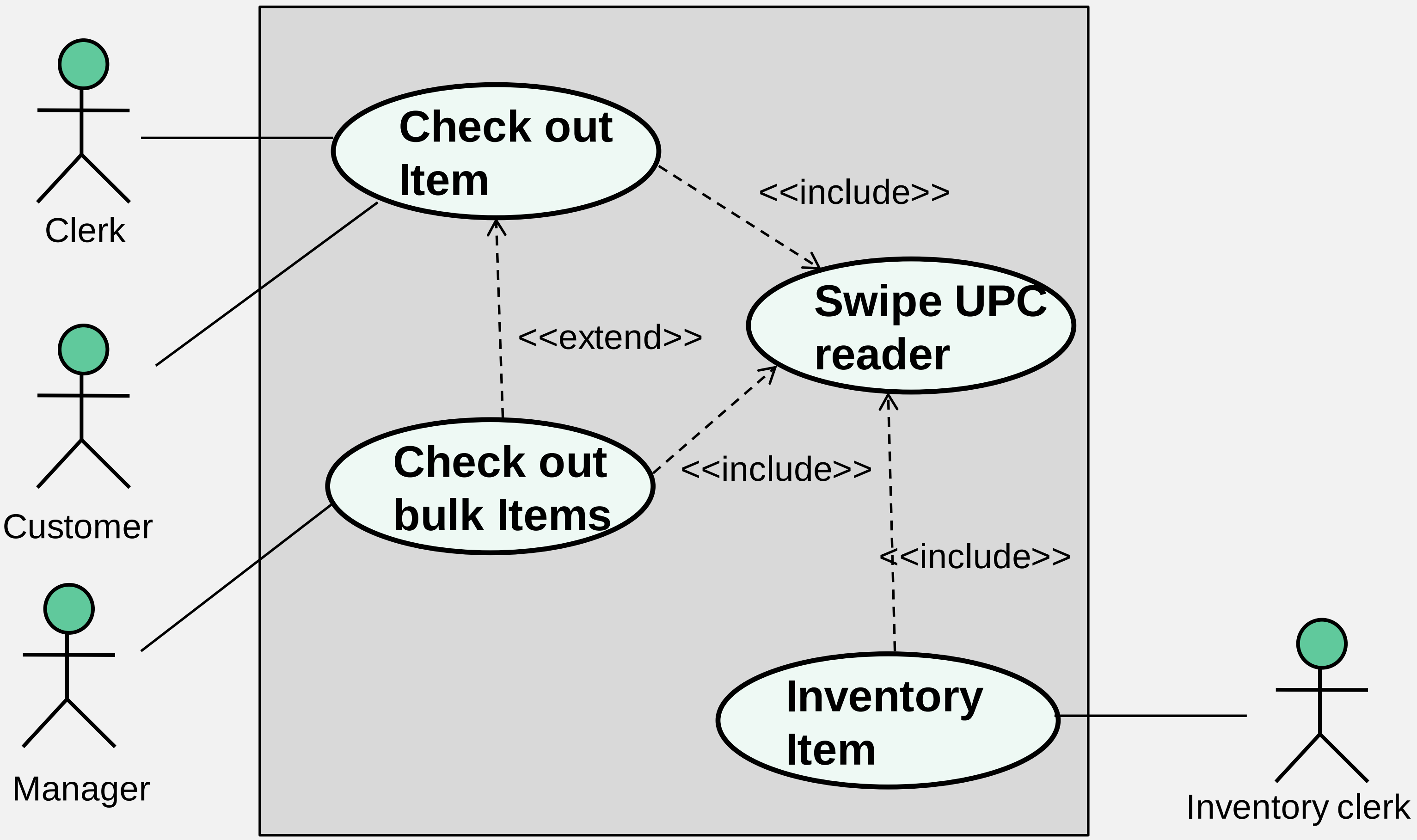
Use case diagrams represent the functionality of the system from user's point of view

Review of Use Case Diagrams



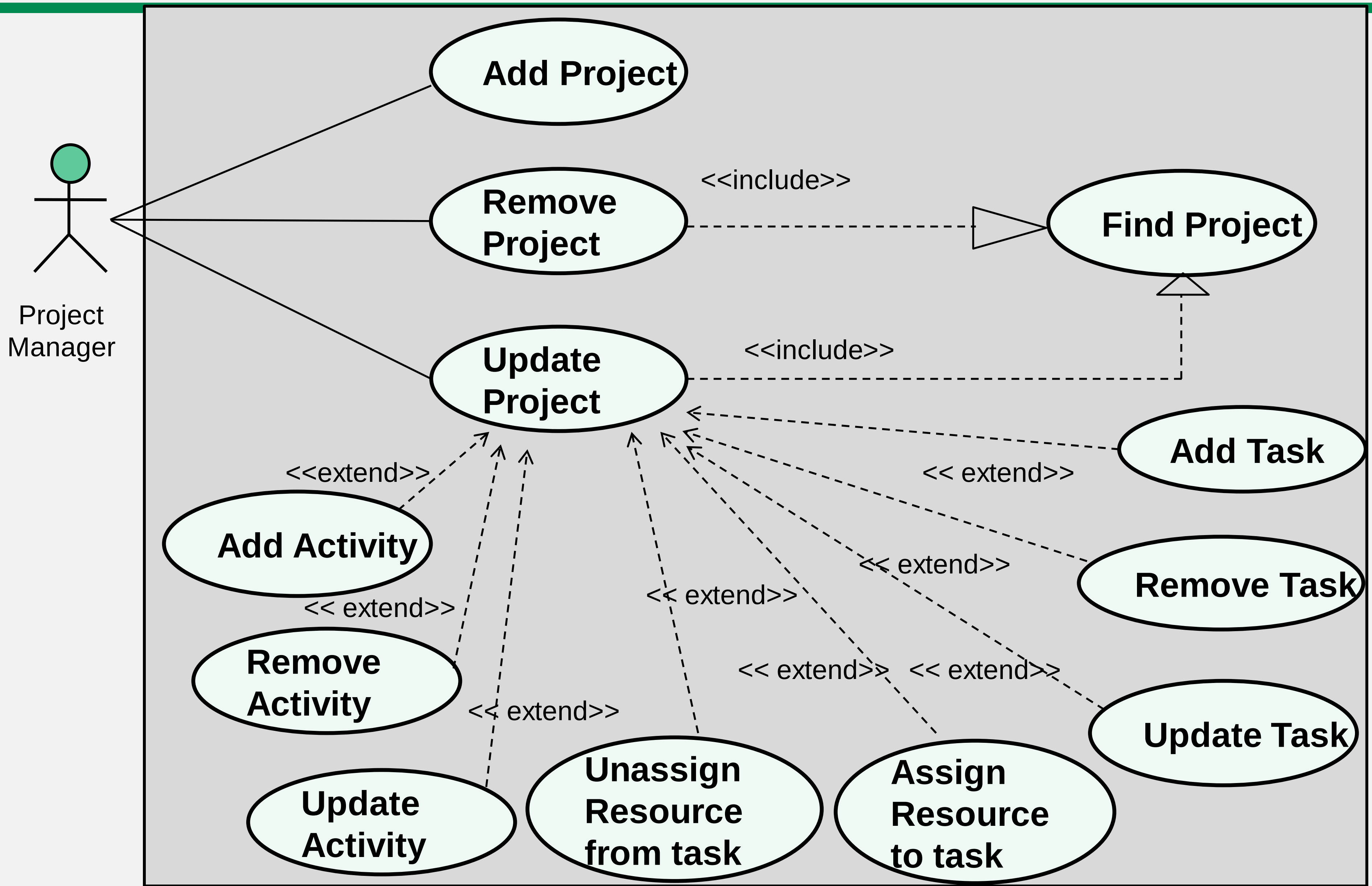
Use case diagrams represent the functionality of the system from user's point of view

Review of Use Case Diagrams

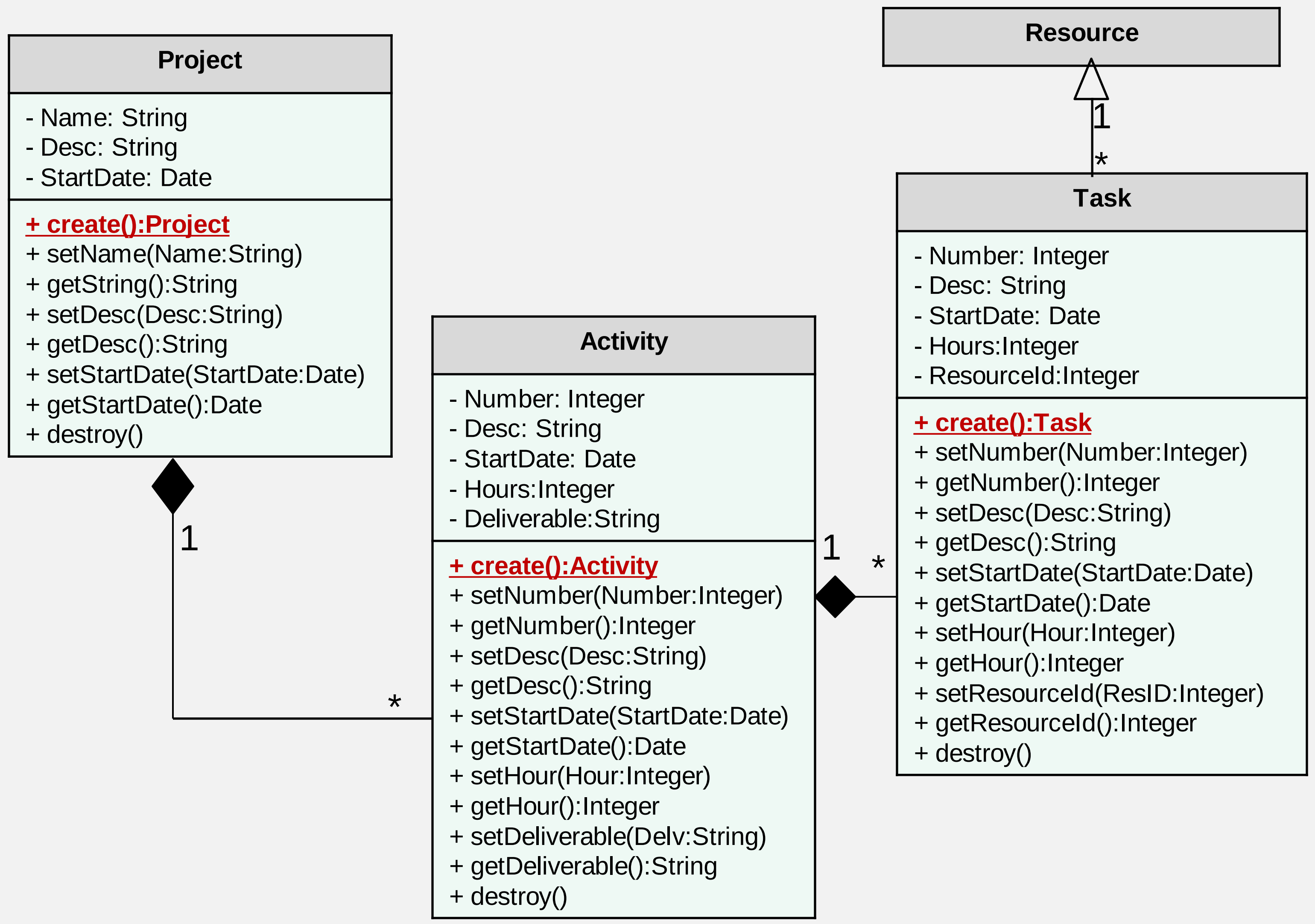


Use case diagrams represent the functionality of the system from user's point of view

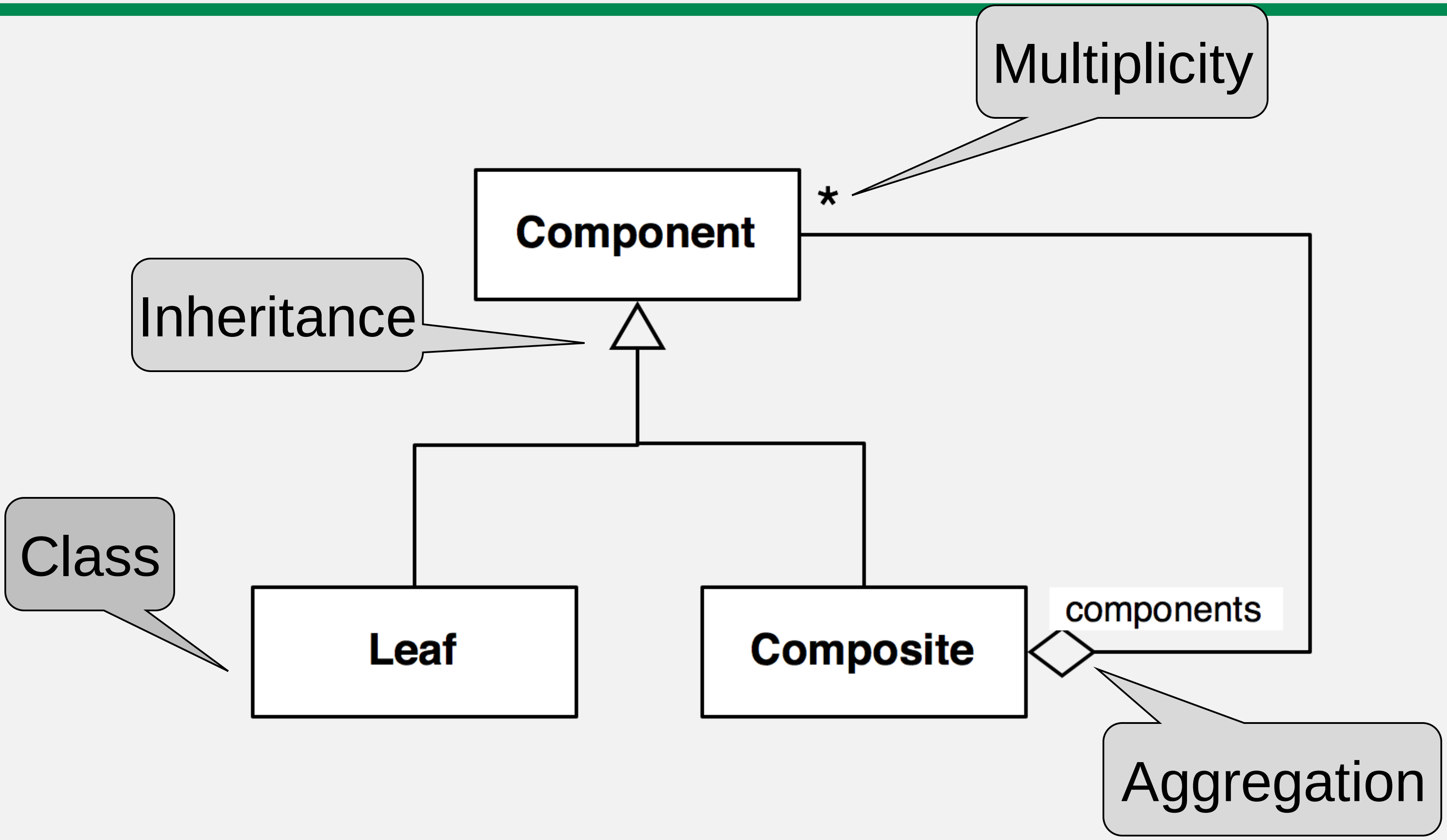
Review of Use Case Diagrams



Review of Class Diagrams



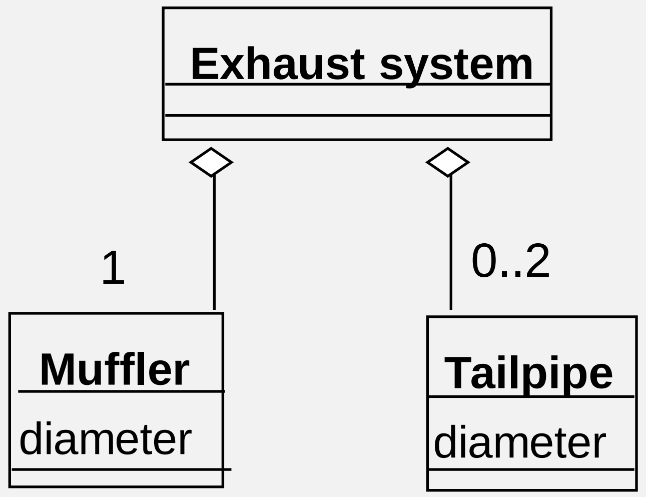
Review of Class Diagrams



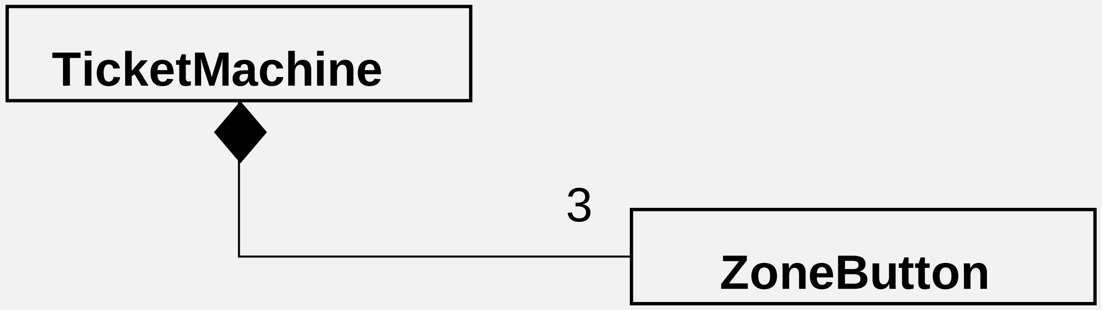
Class diagrams represent the structure of the system

Aggregation

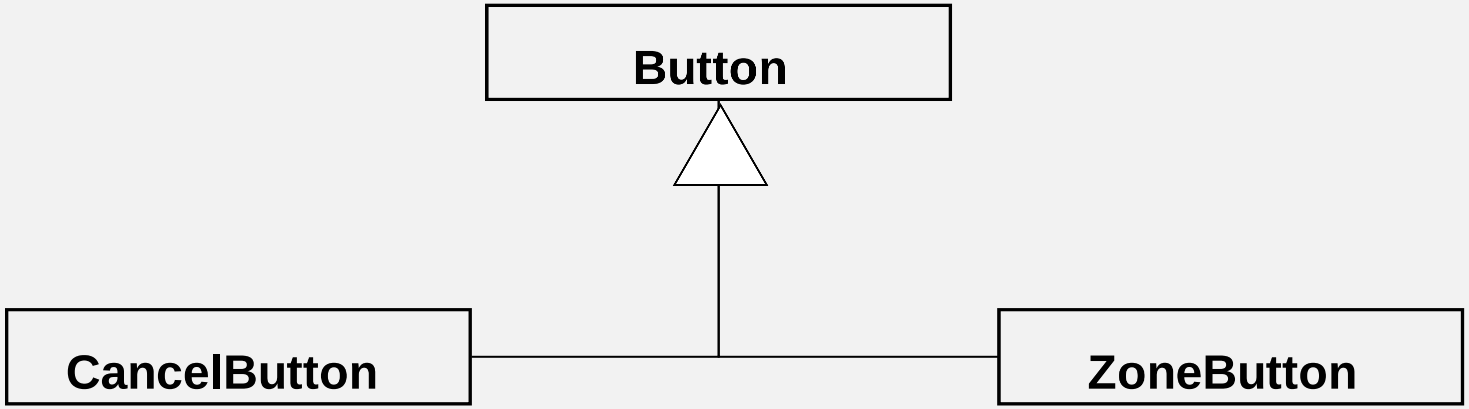
Aggregation



Composition

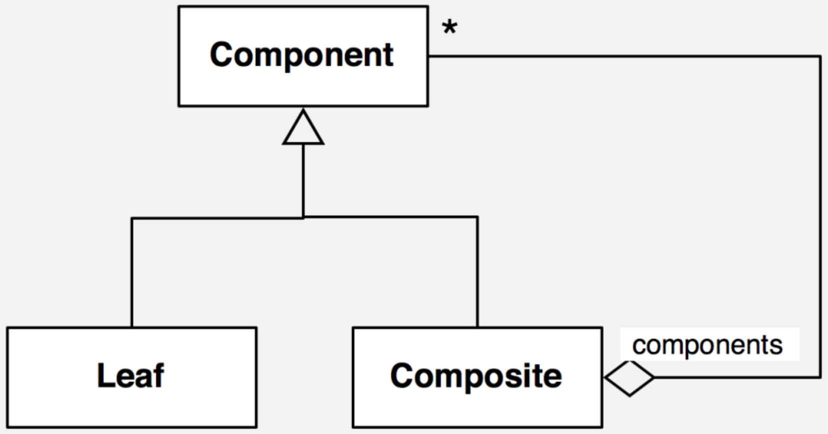


Inheritance



Inheritance is another special case of an association denoting a "kind-of" hierarchy for describing taxonomies

Code Generation from UML to Java I

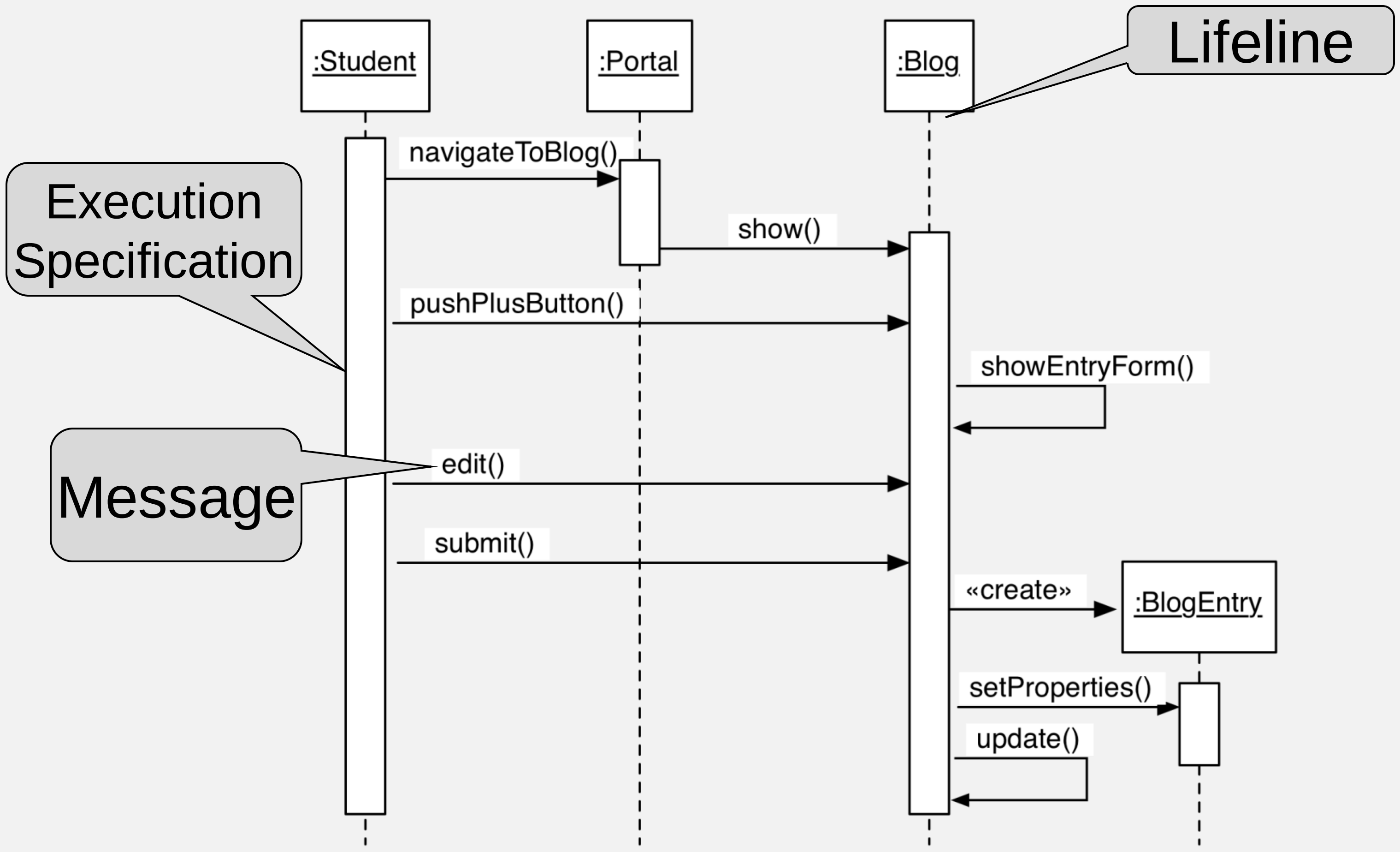


```
public class Component{   }

public class Leaf extends Component{   }

public class Composite extends Component{
   private Collection<Component> components;
   …
}
```

Where are we?

- ✓ What is UML?
- ✓ Review functional modeling
  - ✓ Use case diagram
- ✓ Review object modeling
  - ✓ Class diagram
- ➢ Review dynamic modeling
  - ➢ Sequence diagram
  - State chart diagram
  - Activity diagram

Sequence diagram: Basic Notation



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

Lifeline and Execution Specification

A **lifeline** represents an individual participant (or object) in the interaction

A lifeline is shown using a symbol that consists of a rectangle forming its “head” followed by a vertical line (which may be dashed) that represents the lifetime of the participant

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

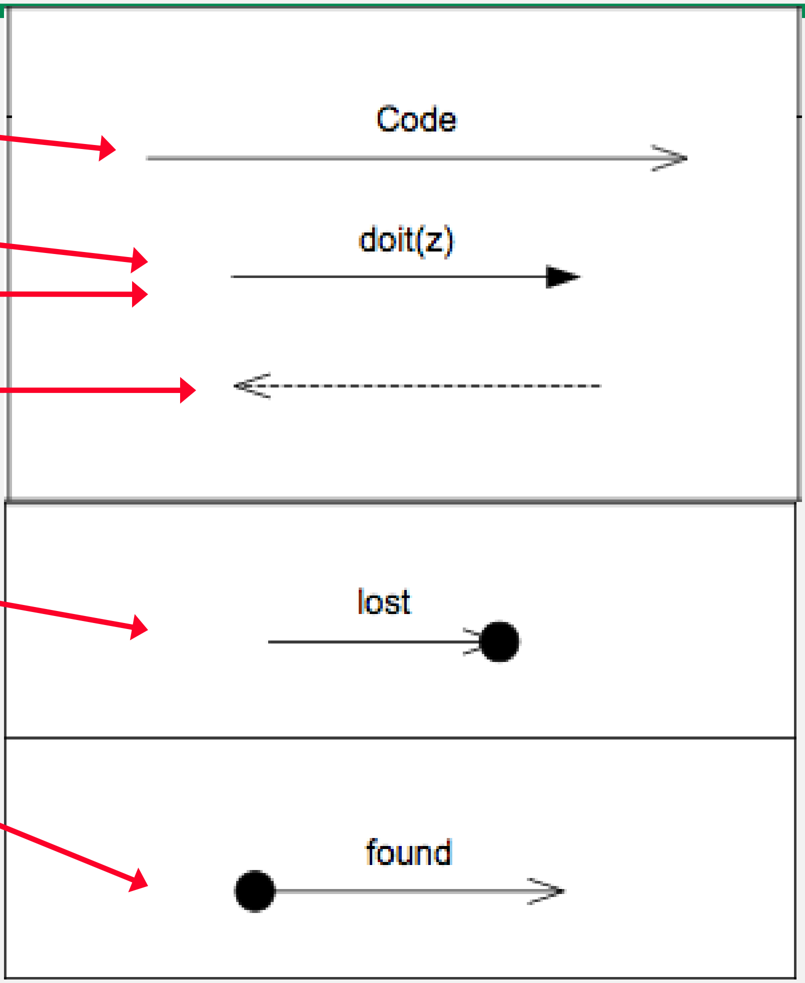
Specify (implicitly) sender and receiver

are shown as a line from the sender to the receiver

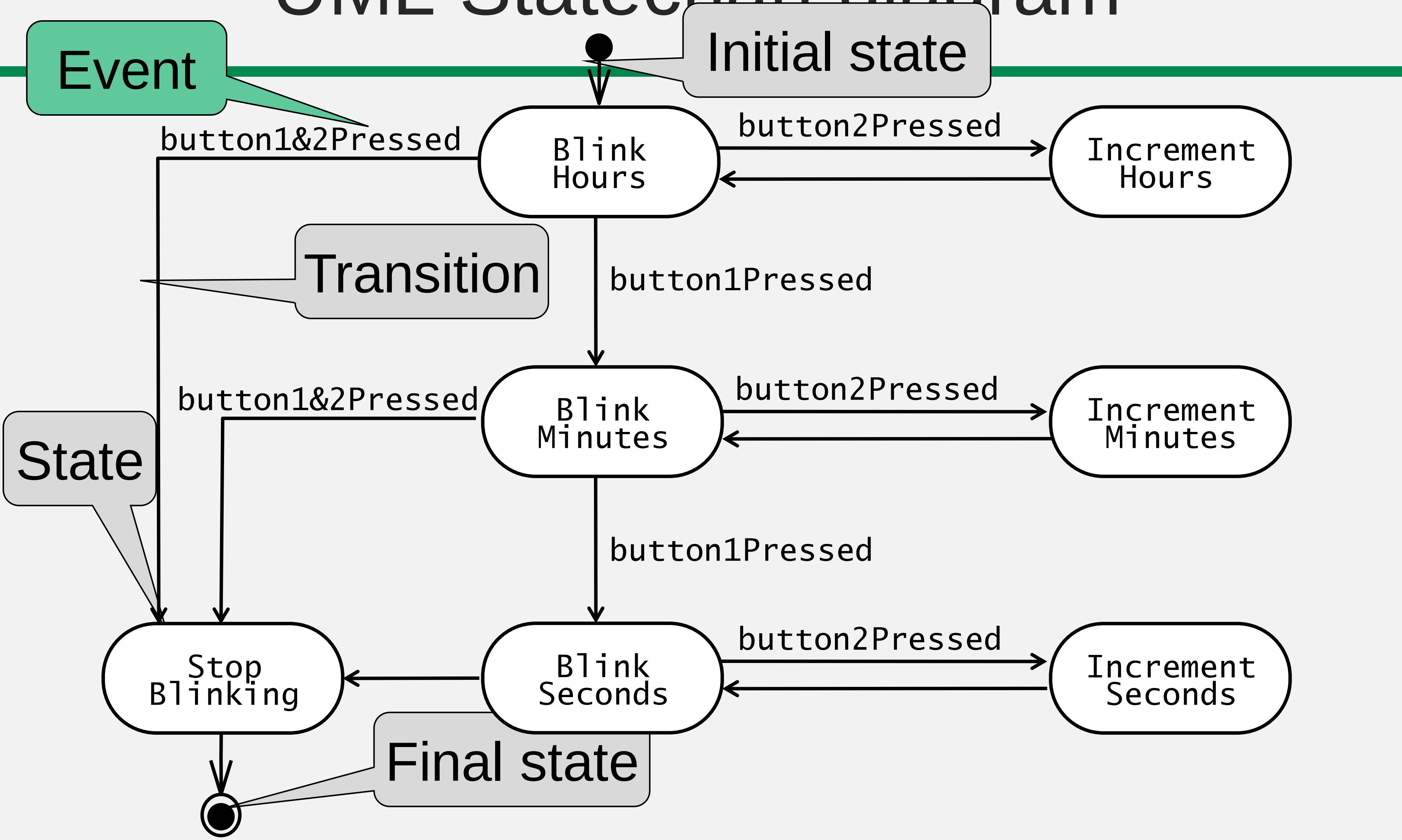
Form of line and arrowhead reflect message properties

Message Types

- Asynchronous
- Synchronous
- Call and Object creation
- Reply
- Lost
- Found



UML Statechart diagram



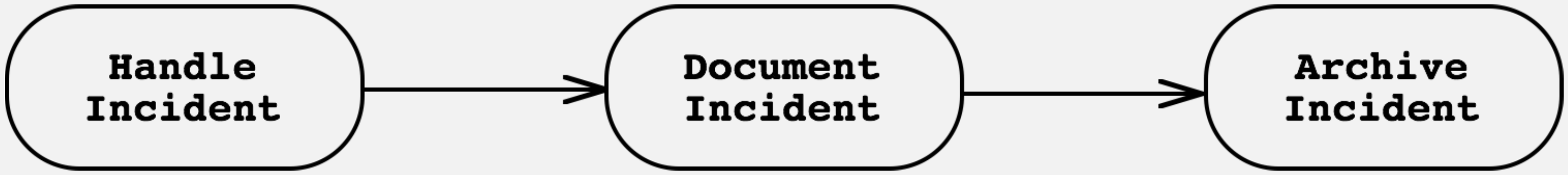
Represents behavior of *a single object* with interesting dynamic behavior.

UML Activity Diagrams

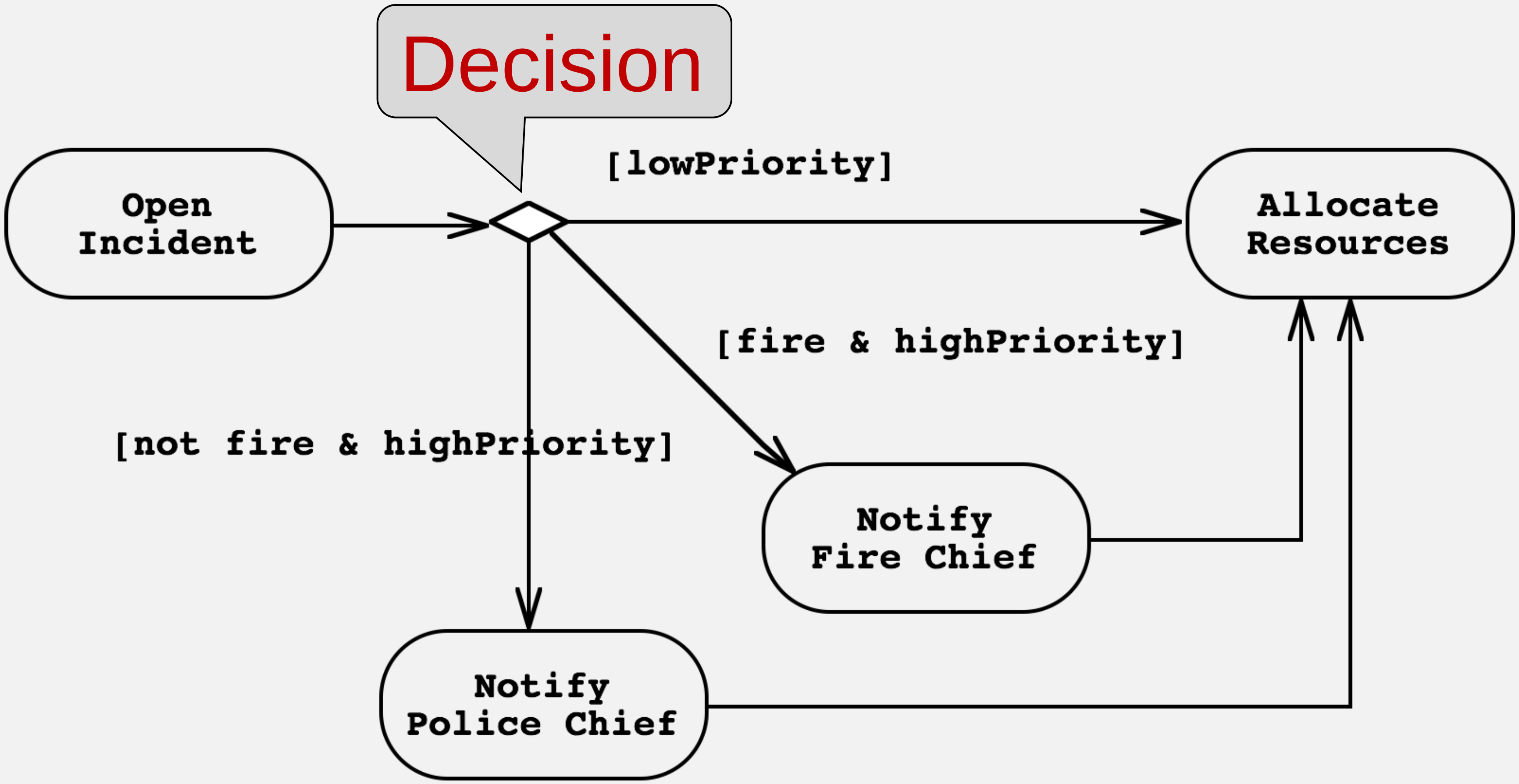
An activity diagram is a special case of a state chart diagram

The states are activities ("functions")

An activity diagram is useful to depict the workflow in a system.



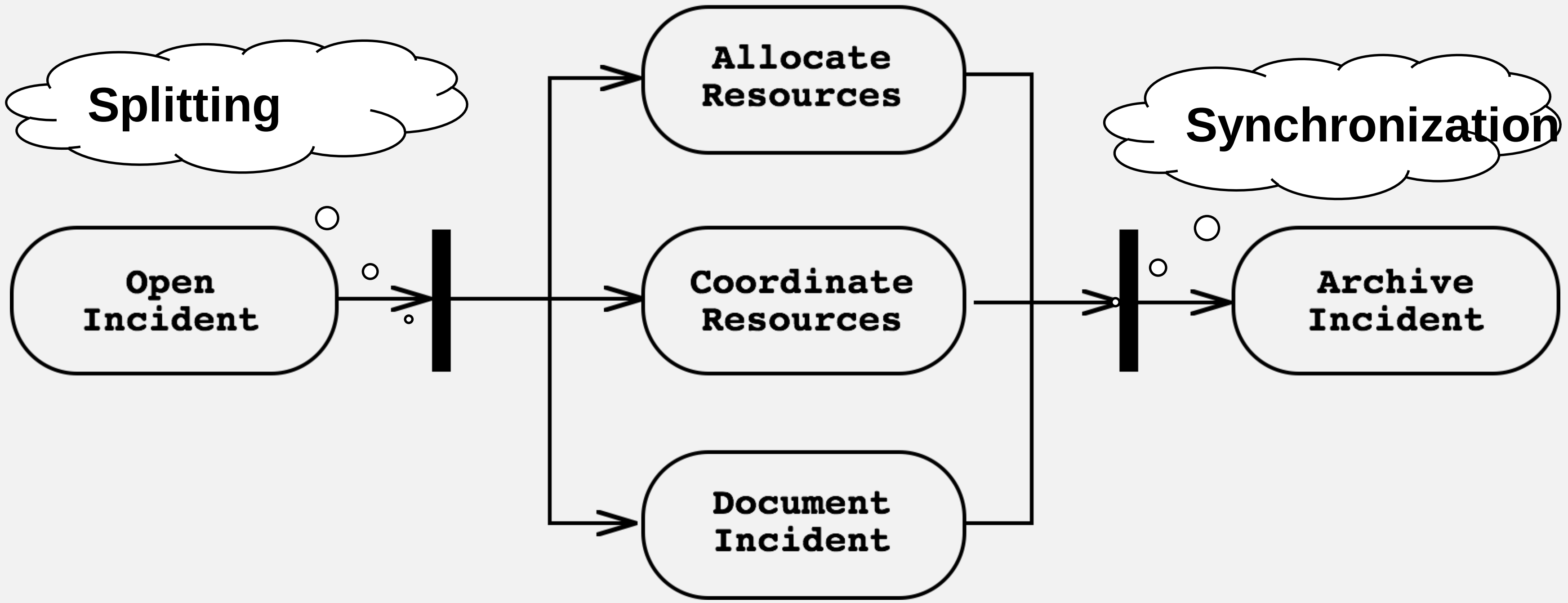
Activity Diagrams allow to model Decisions



Activity Diagrams can model Concurrency

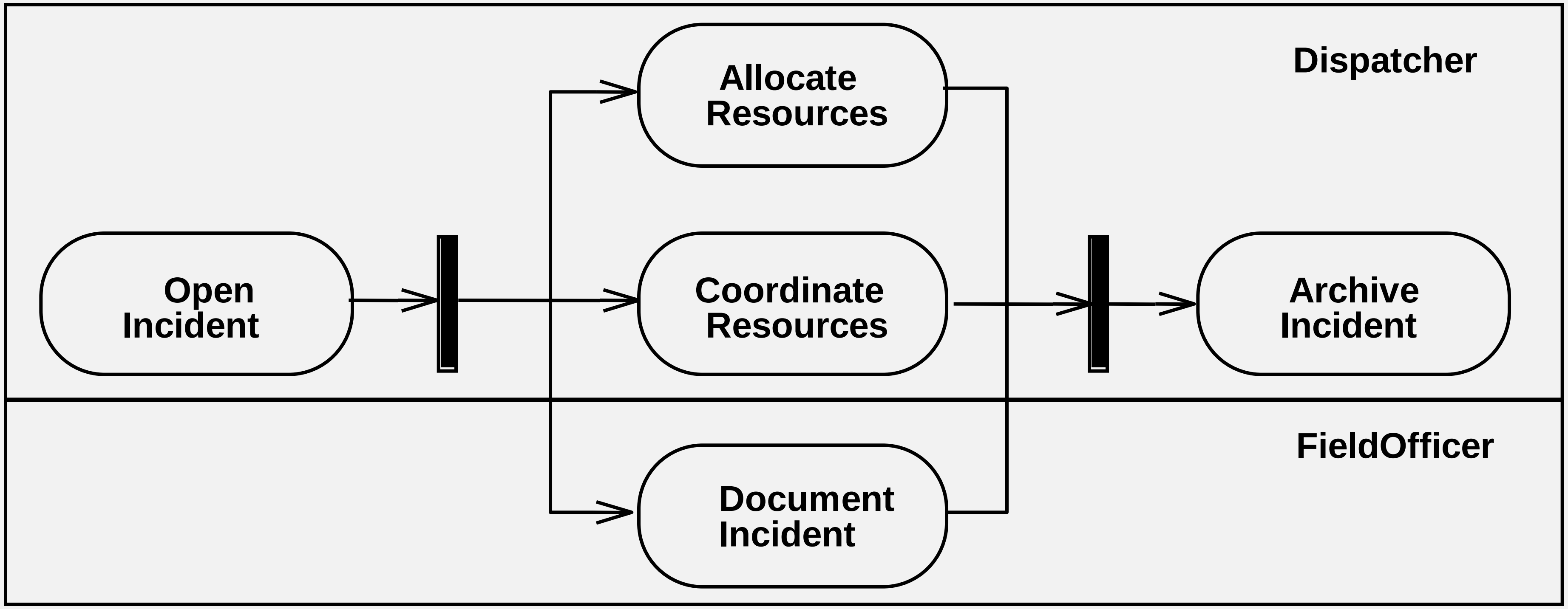
Synchronization of multiple activities

Splitting the flow of control into multiple threads



Activity Diagrams: Grouping of Activities

Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.



Domain Modeling

Why? —The goal of domain modeling is to understand how **system-to-be** will work

Requirements analysis determined how **users** will interact with system-to-be (external behavior)

Domain modeling determines how **elements** of system-to-be interact (internal behavior) to produce the external behavior

Domain Modeling

How? —We do domain modeling based on sources:

Knowledge of how system-to-be is supposed to behave (*from requirements analysis, e.g., use cases*)

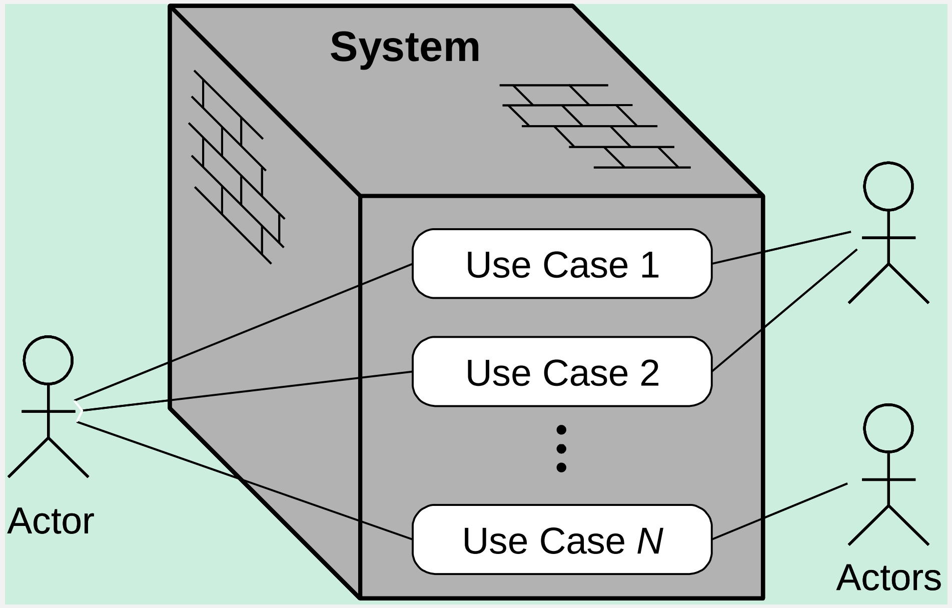
- Studying the work domain (or, problem domain)
- Knowledge base of software designs
- Developer's past experience with software design

Use Cases vs. Domain Model

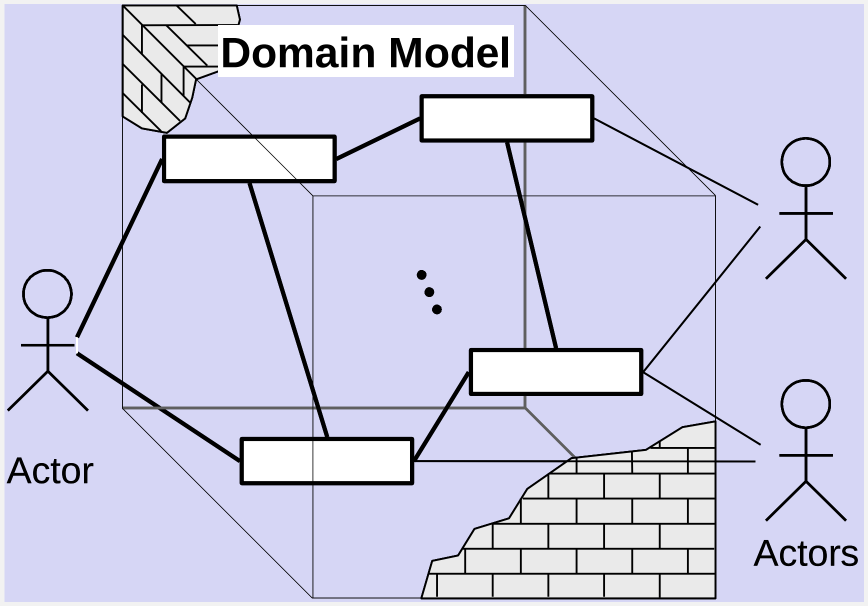
In **use case analysis**, we consider the system as a "**black box**"

In **domain analysis**, we consider the system as a "**transparent box**"

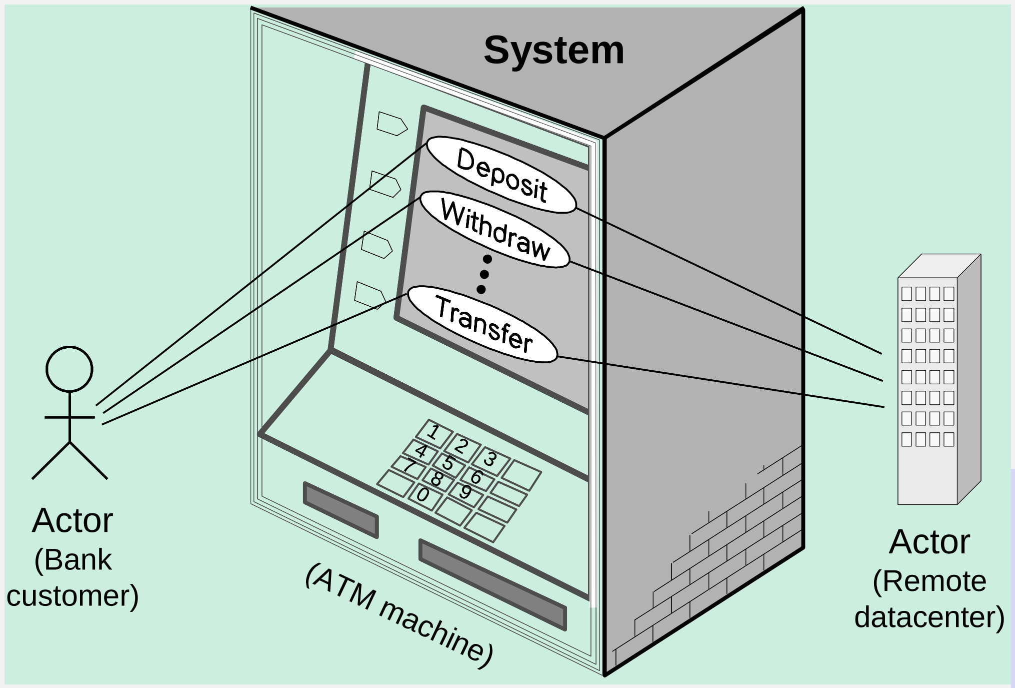
(a)



(b)

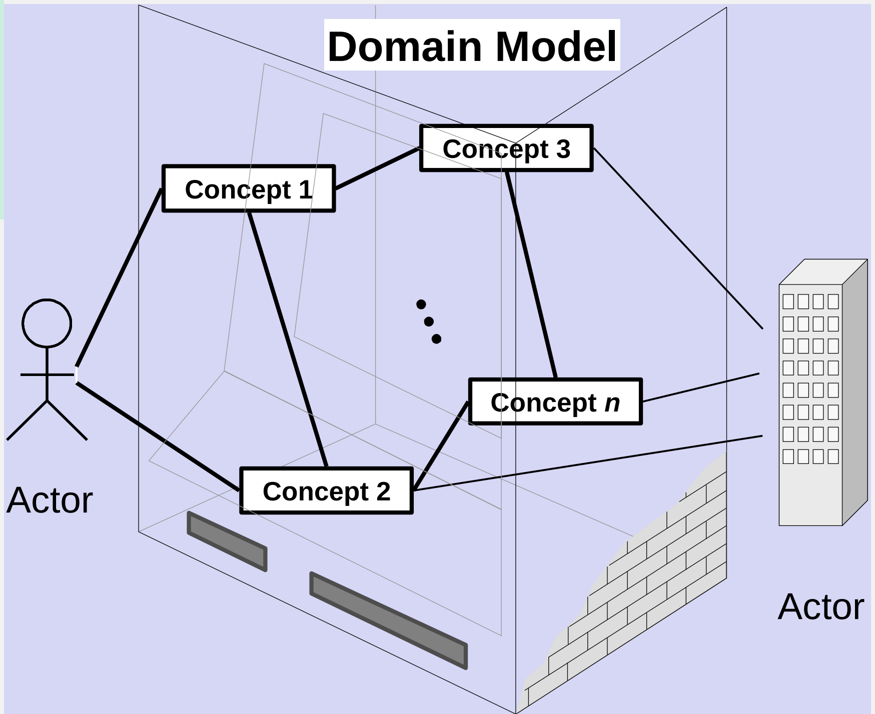


Example: ATM Machine



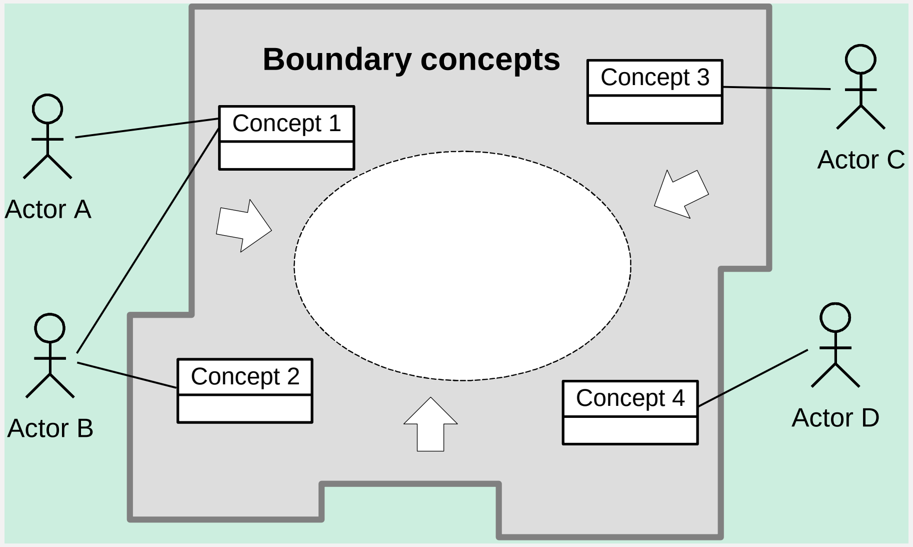
(a)

(b)

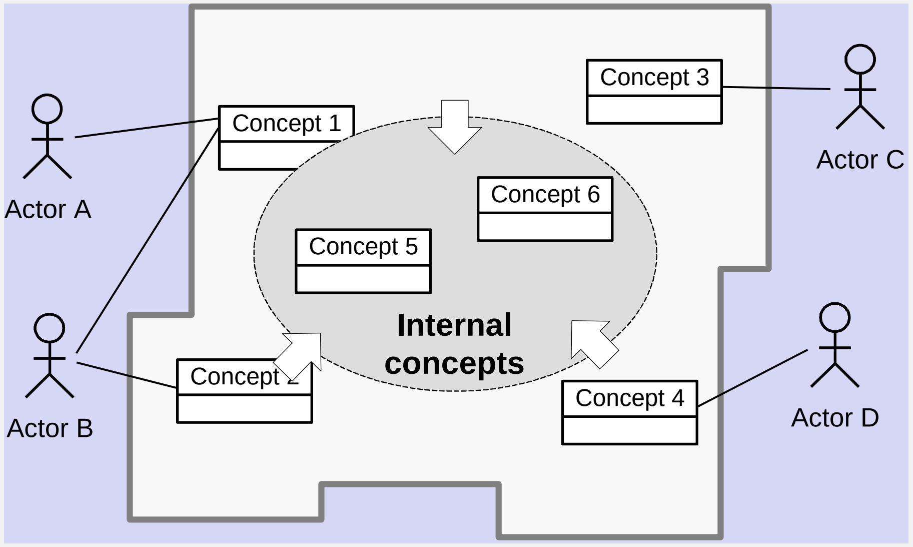


Building Domain Model from Use Cases

Step 1: Identifying the boundary concepts



Step 2: Identifying the internal concepts



Thank You