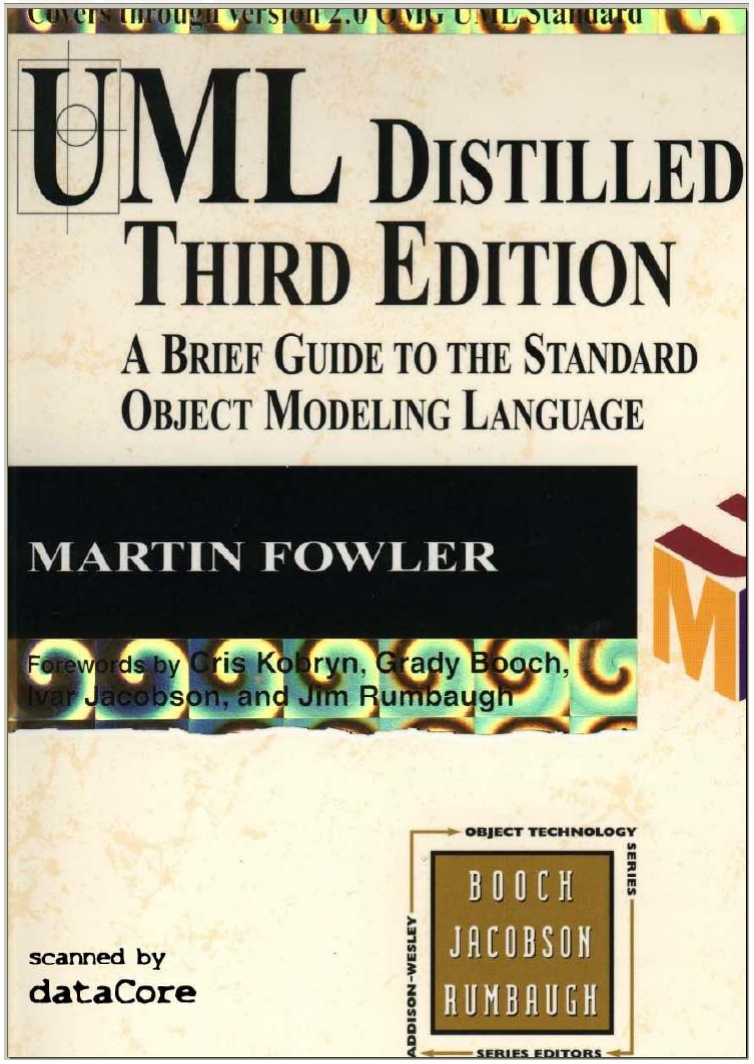


Introduction to UML 2.0

(slides from ‘06 CS550 by Prof.Bae)

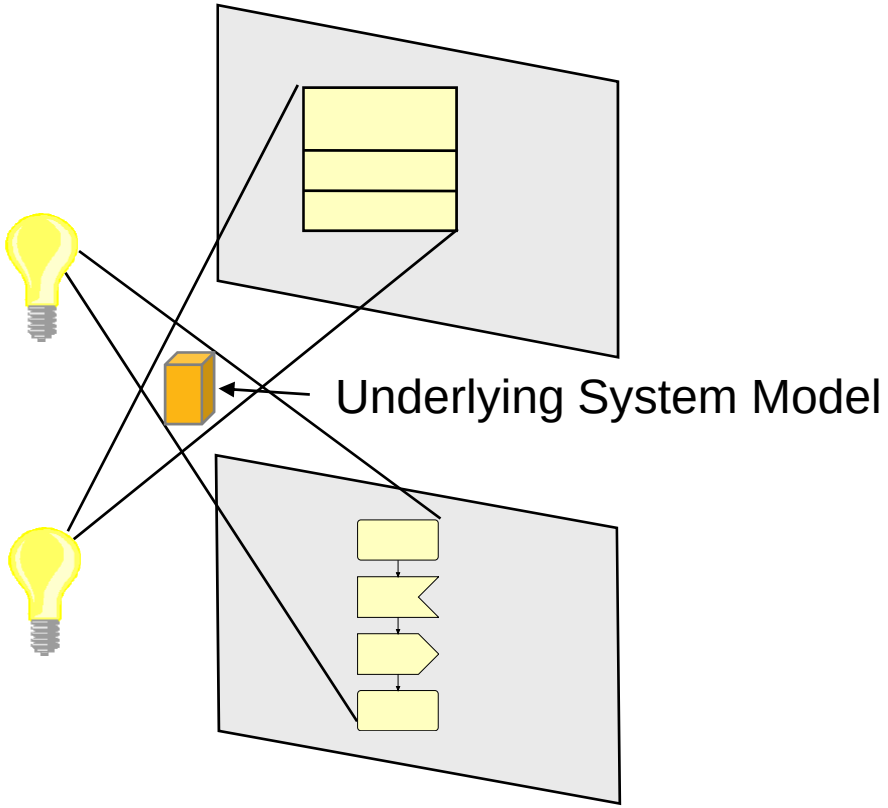
UML Introduction

What is UML?

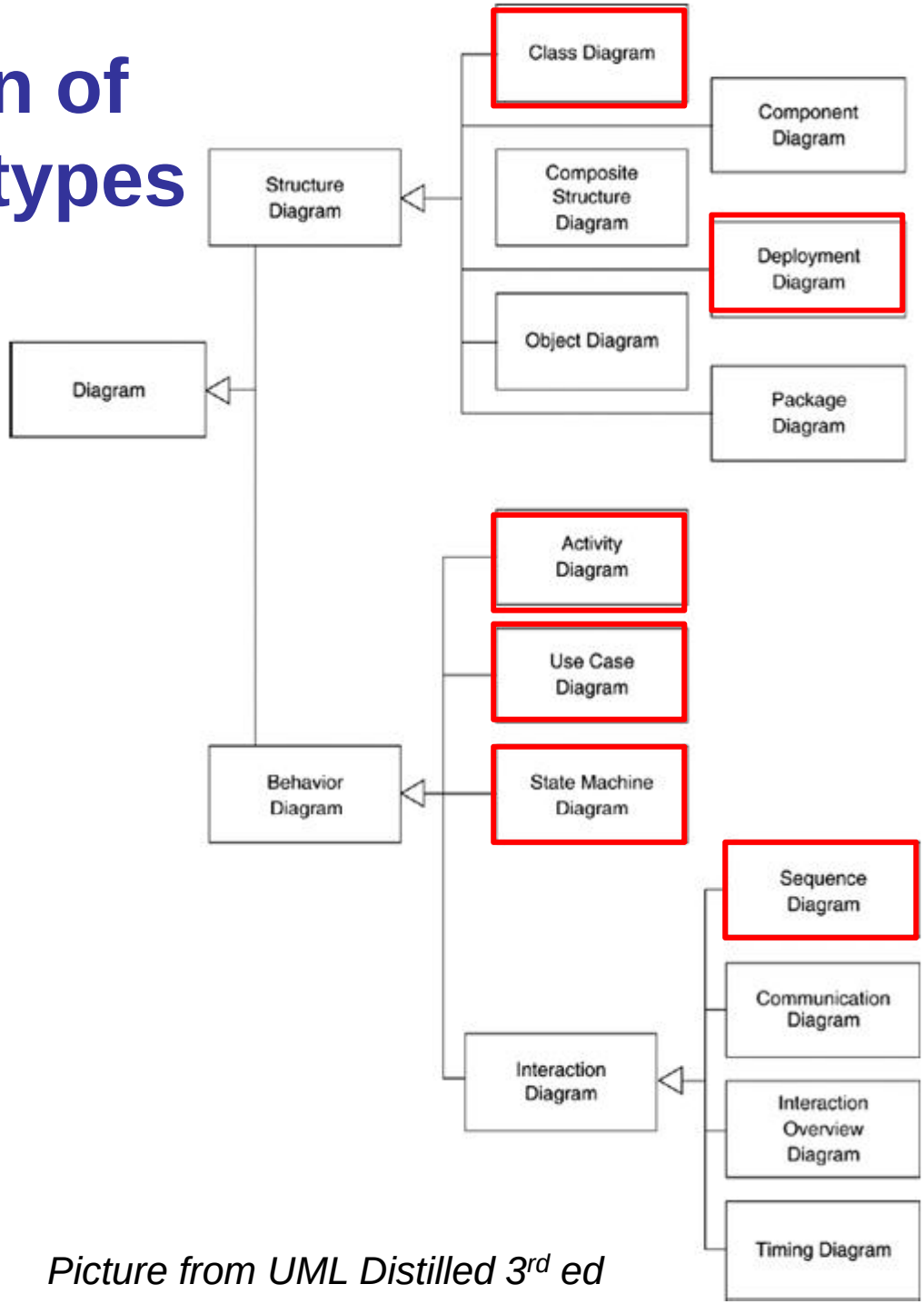
- **U**nified **M**odeling **L**anguage
  - Visual language for specifying, constructing and documenting
- Maintained by the OMG (Object Management Group)
  - Website: http://www.omg.org
- Object-oriented
- Model / view paradigm
- Target language independent

Model / View Paradigm

- Each diagram is just a view of part of the system
- Together, all diagrams provides a complete picture



Classification of UML diagram types



Picture from UML Distilled 3rd ed

Usage of UML

- UML as sketch
  - Selectivity (abstraction) is the key
  - No formal semantics are given
- UML as blueprint
  - Completeness is the key
- UML as a programming language
  - To generate C/Java code from UML diagrams
  - No formal definition exists of how the UML maps to any particular programming language

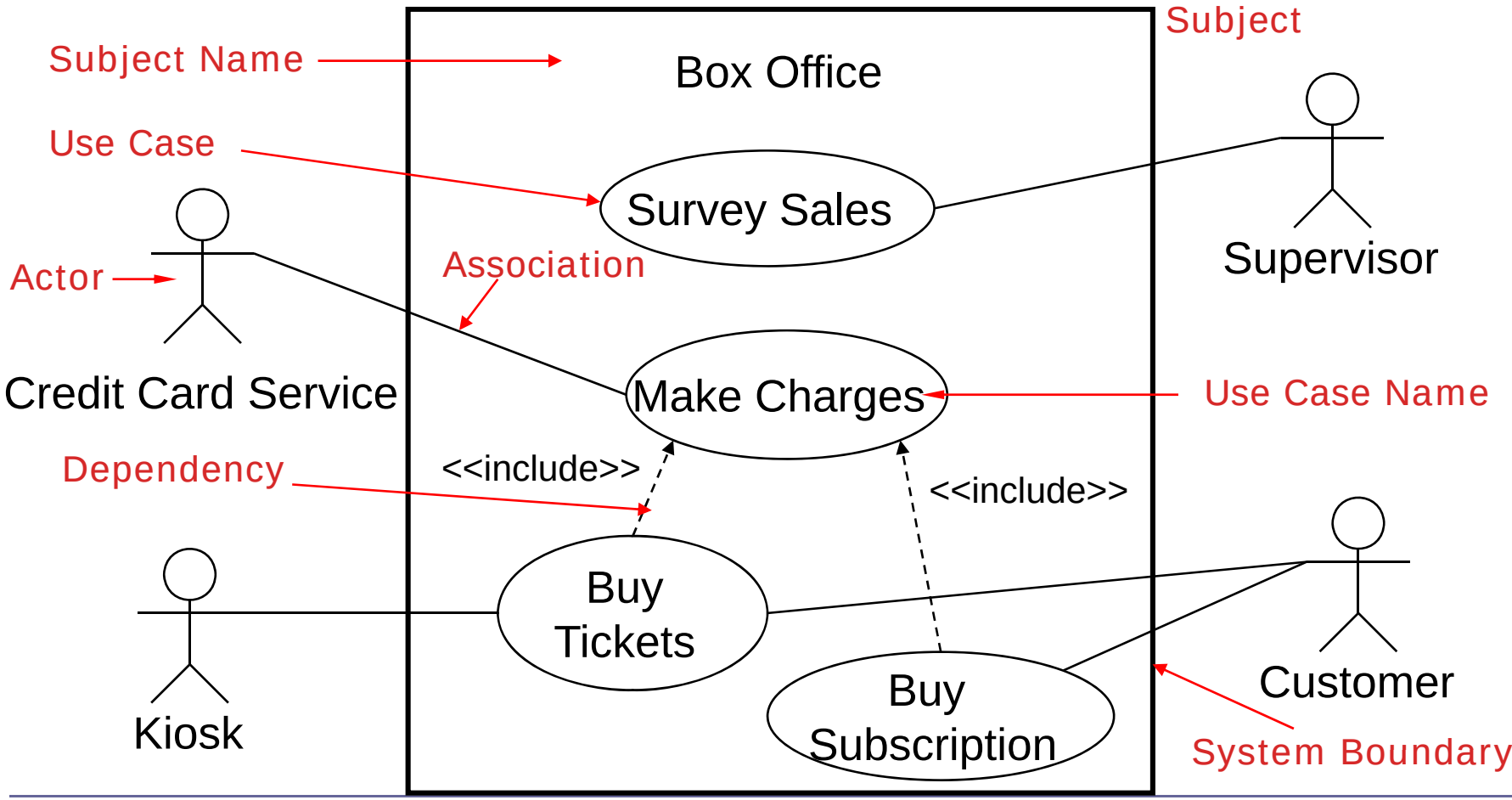
Use Case Diagrams

What is a Use Case?

Use Case ~ A behavior or coherent set of behaviors triggered by events sent to the system by human user(s), other systems, hardware components, or an internal clock

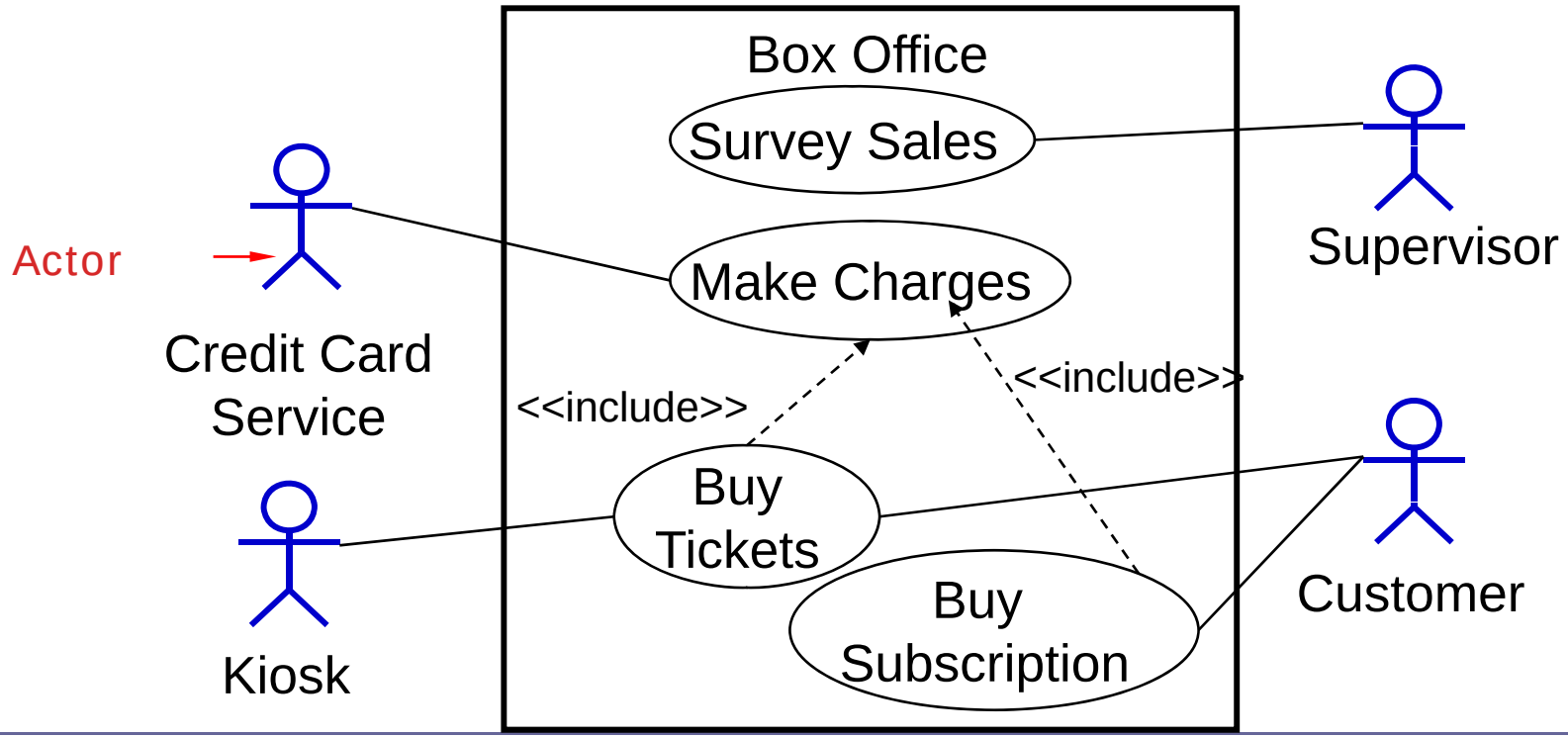
Use Case Diagrams

- Describe WHAT the system will do at a high-level



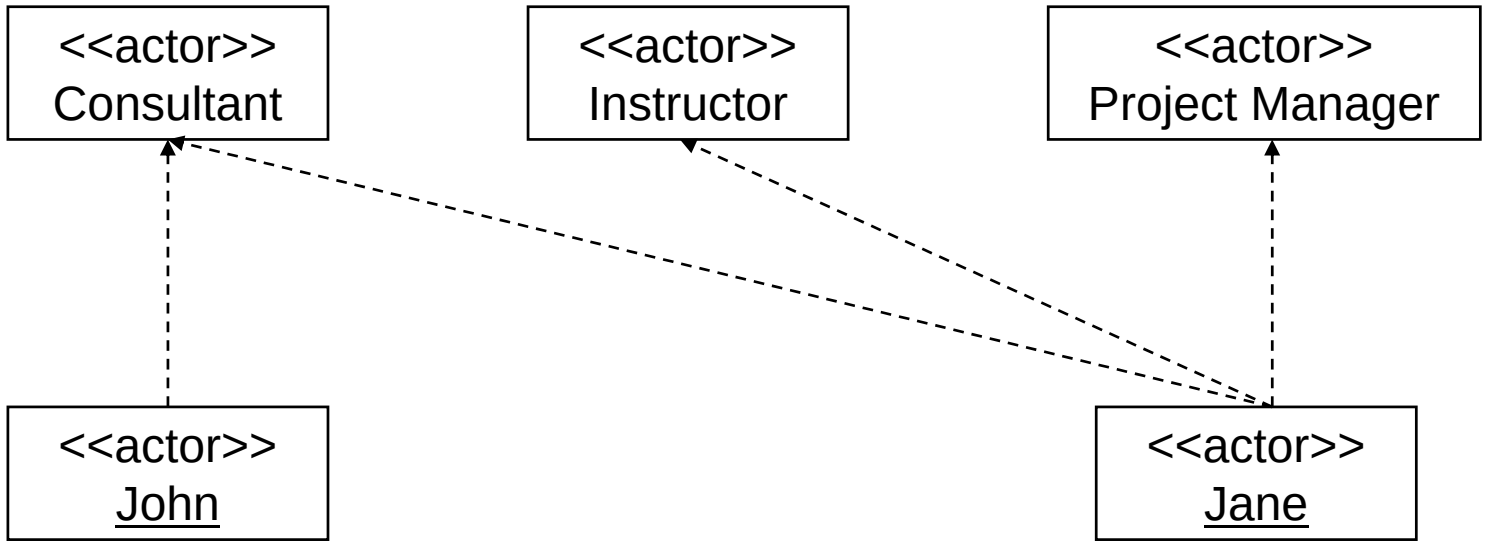
Actor

- Someone or some thing that must interact with the system
  - Users, external systems, devices



An Actor is a Role

- An actor defines a single role played by users in their interactions with the system:
  - Multiple users can play a single role
  - A single user may play multiple roles



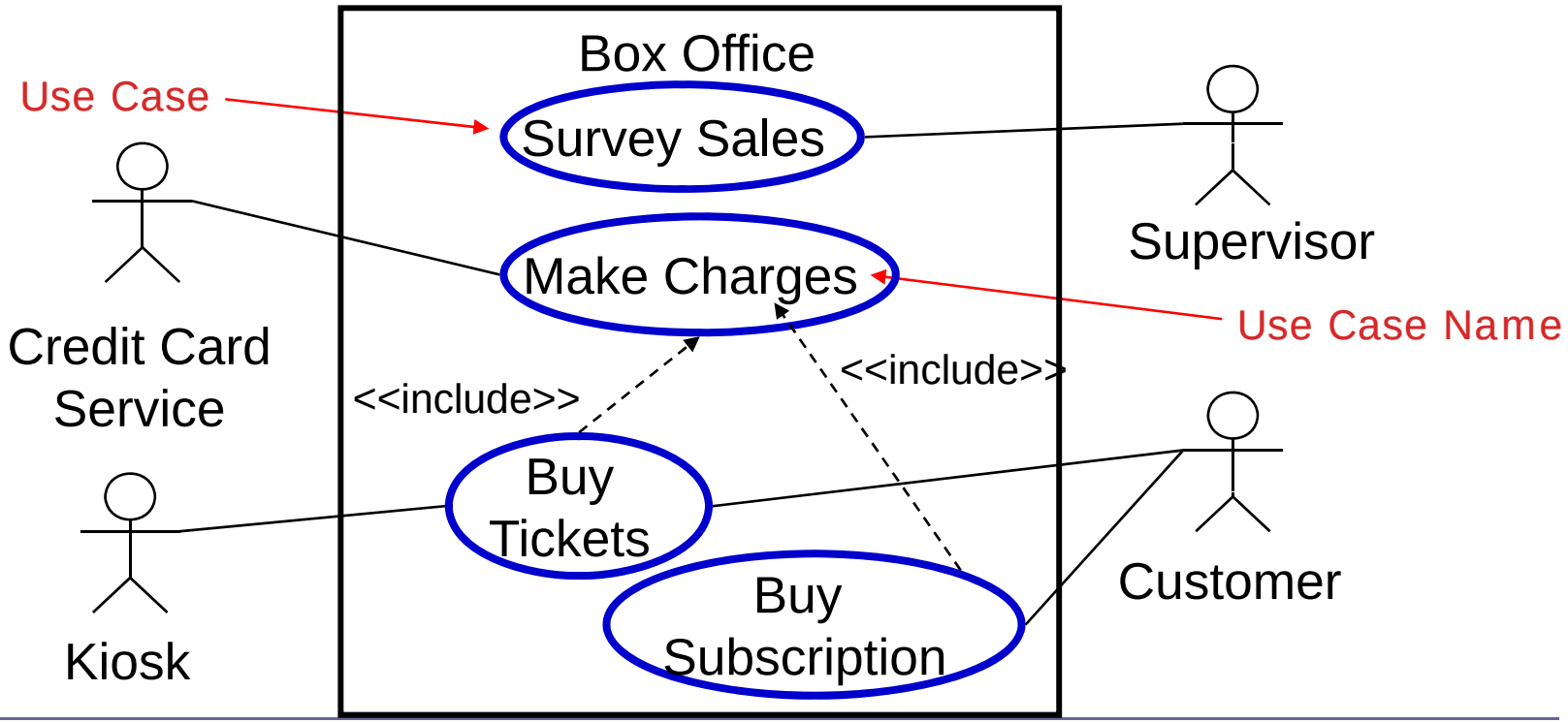
Identifying Actors

- Useful questions
  - Who will use the main functionality of the system (primary actors)?
  - Who will need support from the system to their daily tasks?
  - Who will need to maintain, administrate, and keep the system working (secondary actors)?
  - Which hardware devices does the system need to handle?
  - With which other systems does the system need to interact?
  - Who or what has an interest in the results (the value) that the system produces?

(From :oopsla.snu.ac.kr/research/UML/)

Use Case

- Unit of functionality expressed as a transaction among actors and the subject
- Interaction between one or more actors and the system



Use Case

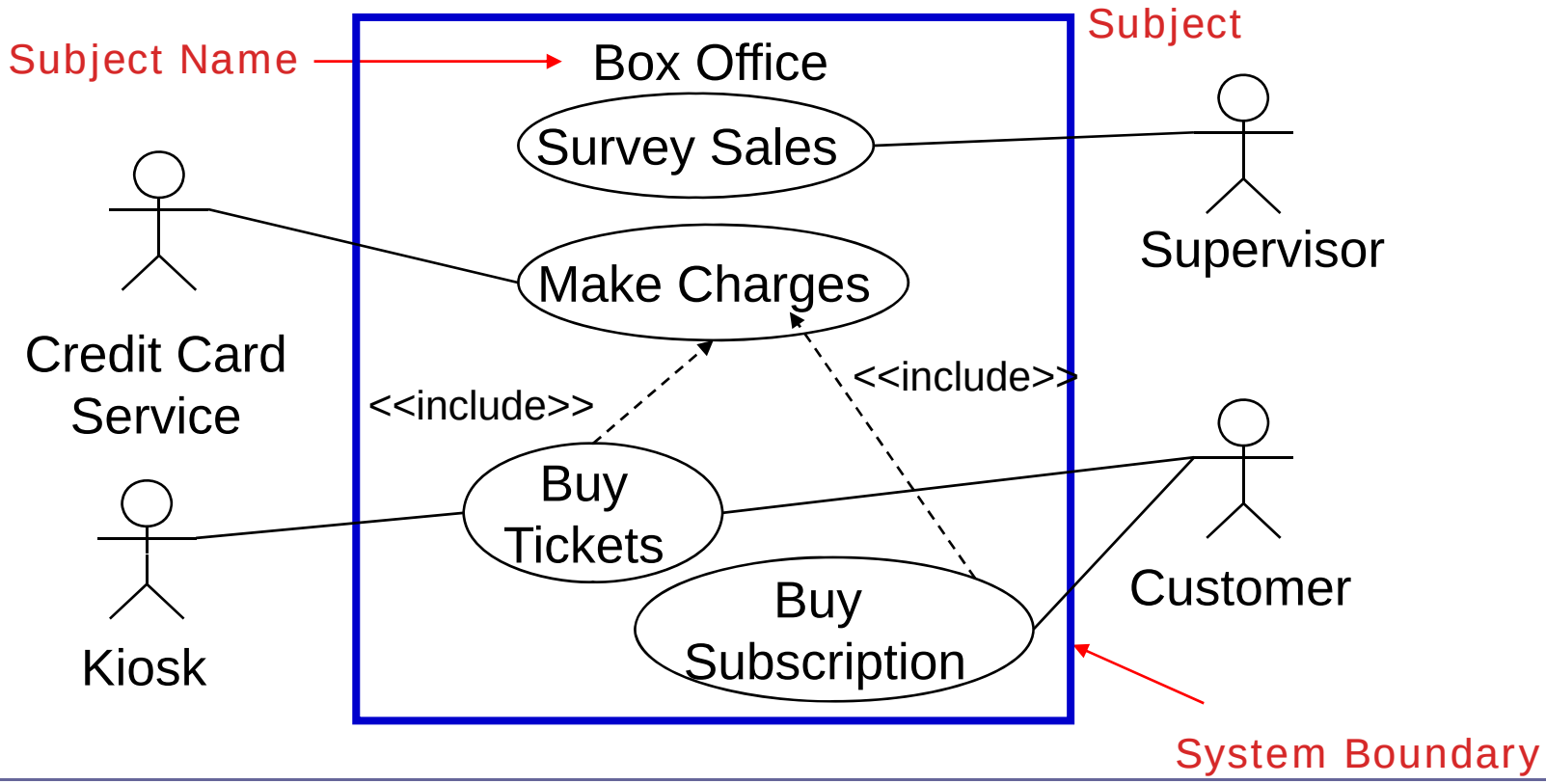
- Identifying Use Cases
  - Which functions does the actor require from system?
  - Does the actor need to read, create, destroy, modify, or store some kind of information in the system?
  - Does the actor have to be notified about events in the system
  - Could the actor's daily work be simplified or made more efficient through new functions in the system

An Example of Use Case Text

- Buy a Product
  - Main Success Scenario :
    - **1 . Customer browses catalog and selects items to buy**
    - **2 . Customer goes to check out**
    - **3. Customer fills in shipping information (address ; next-day or 3-day delivery)**
    - **4. System presents full pricing information, including shipping**
    - **5 . Customer fills in credit card information**
    - **6 . System authorizes purchase**
    - **7 . System confirms sale immediately**
    - **8 . System sends confirming e-mail to customer**
  - Extensions :
    - **3a : Customer is regular customer**
      - .1 : System displays current shipping, pricing, and billing information
      - .2 : Customer may accept or override these defaults, returns to MSS at step 6
    - **6a : System fails to authorize credit purchase**
      - .1 : Customer may reenter credit card information or may cancel

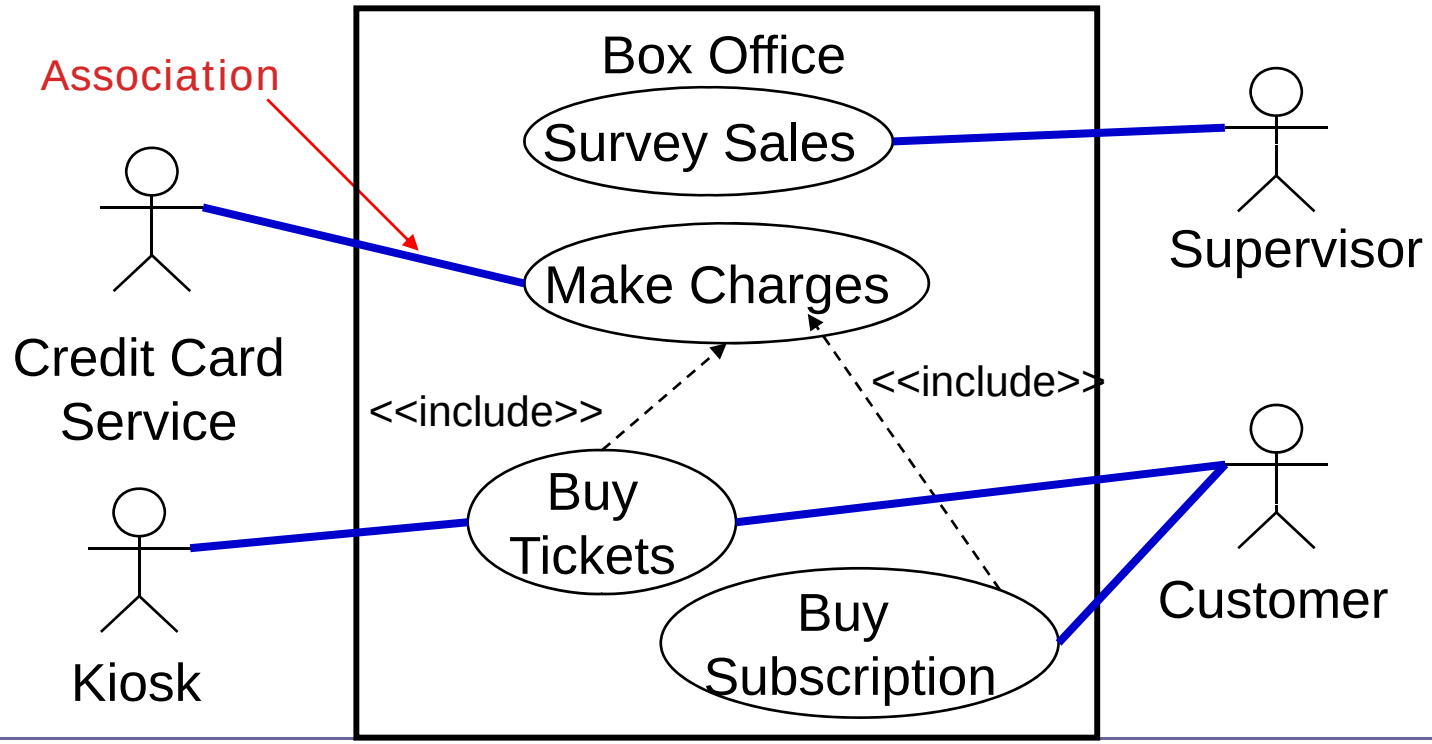
Subject Symbol

- Indicate system boundary
  - Classifier that realizes behavior defined by a use case



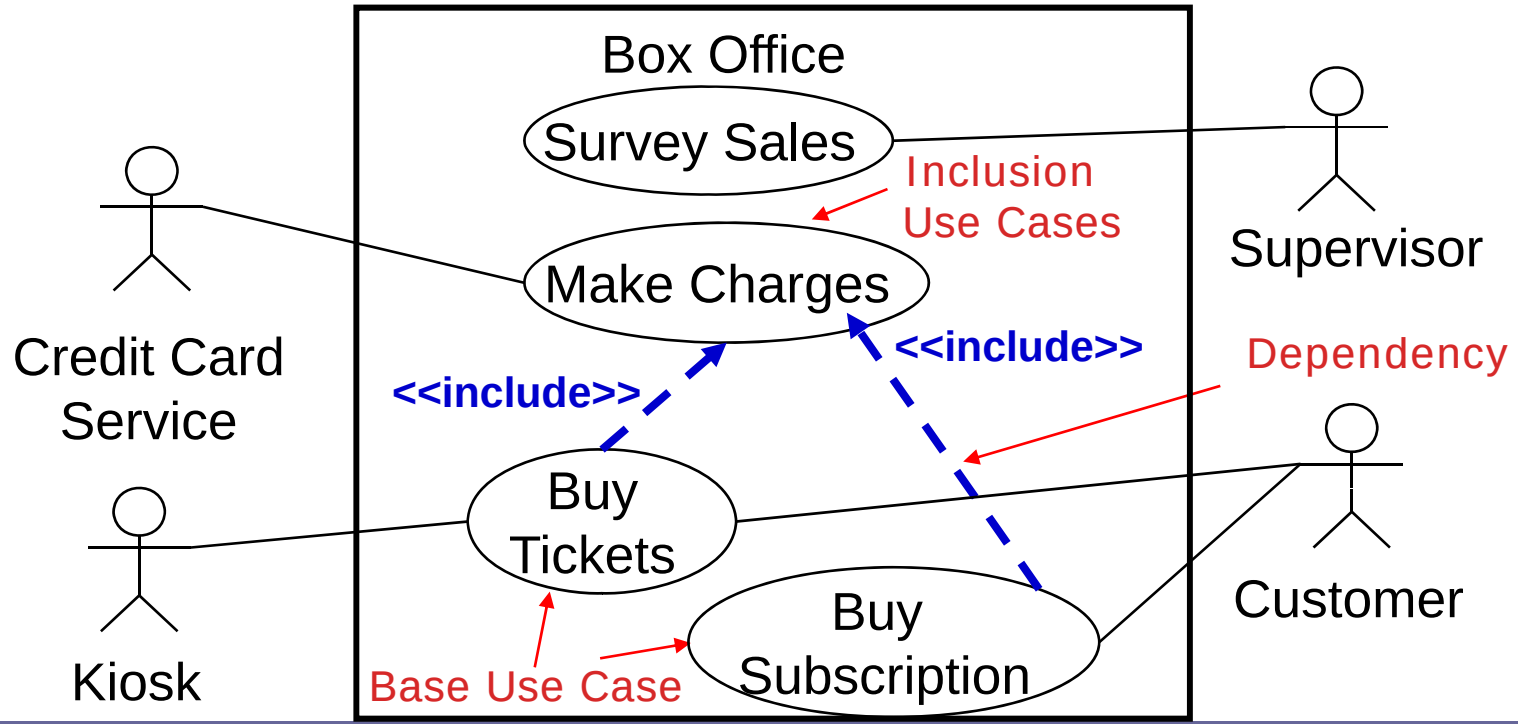
Association

- Represent bi-directional communication between the actor and the system
- Drawn between an actor and a use case



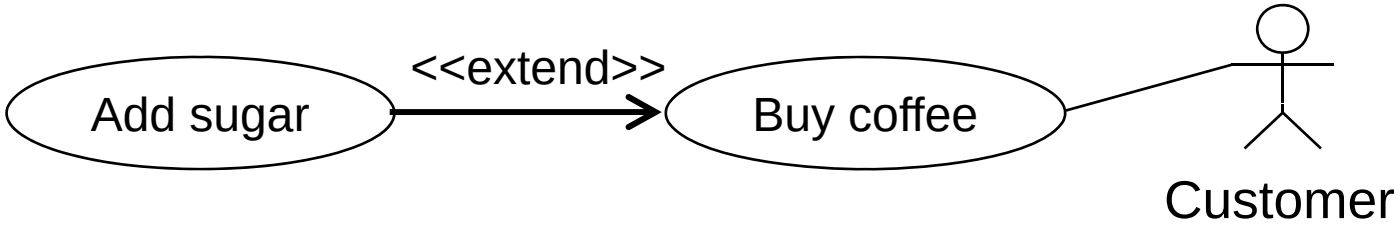
Dependency – Include

- Represent relationship from a *base* to an *inclusion* use case
- Imply a Use Case calls another Use Case
- Primarily used to reuse behavior common to several Use Cases



Dependency – Extend

- Used when some additional behavior should be added
  - Models optional or conditional behavior
  - Show infrequent events



Tips for Use Case Modeling

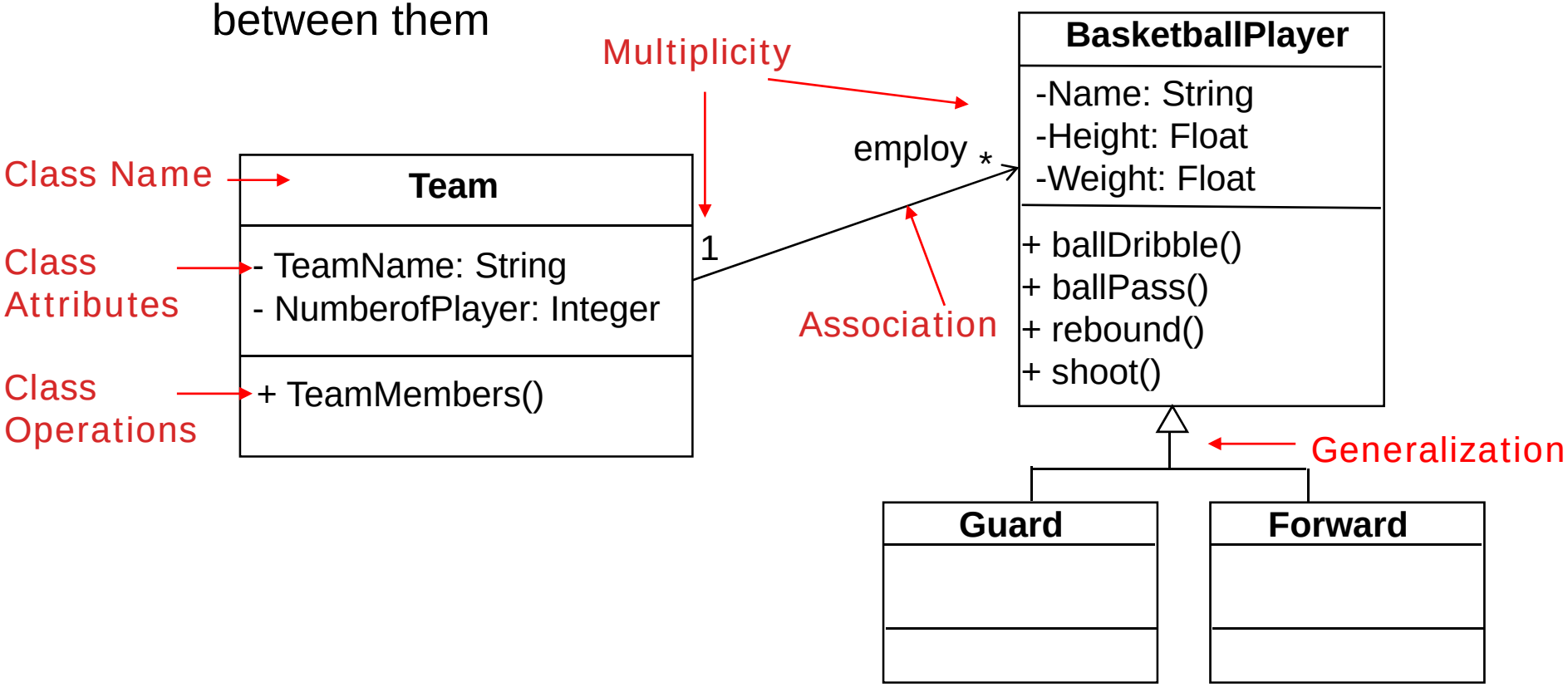
- Make sure that each use case describes a significant chunk of system usage that is understandable by both domain experts and programmers
- When defining use cases in text, use nouns and verbs accurately and consistently to help derive objects and messages for interaction diagrams
- Factor out common usages that are required by multiple use cases
  - If the usage is required use <<include>>
  - If the base use case is complete and the usage may be optional, consider use <<extend>>
- A use case diagram should
  - contain only use cases at the same level of abstraction
  - include only actors required
- Large numbers of use cases should be organized into packages

(From :oopsla.snu.ac.kr/research/UML/)

Class Diagrams

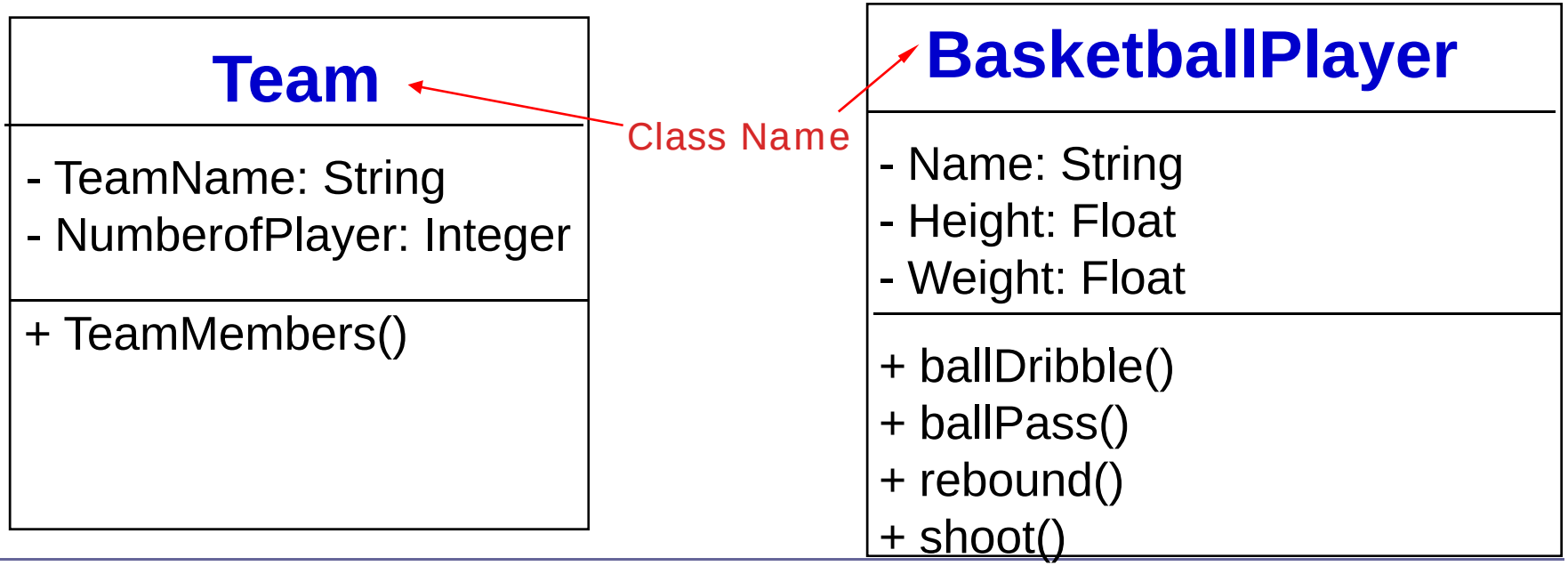
Class Diagrams

- Description of static structure
  - Showing the types of objects in a system and the relationships between them



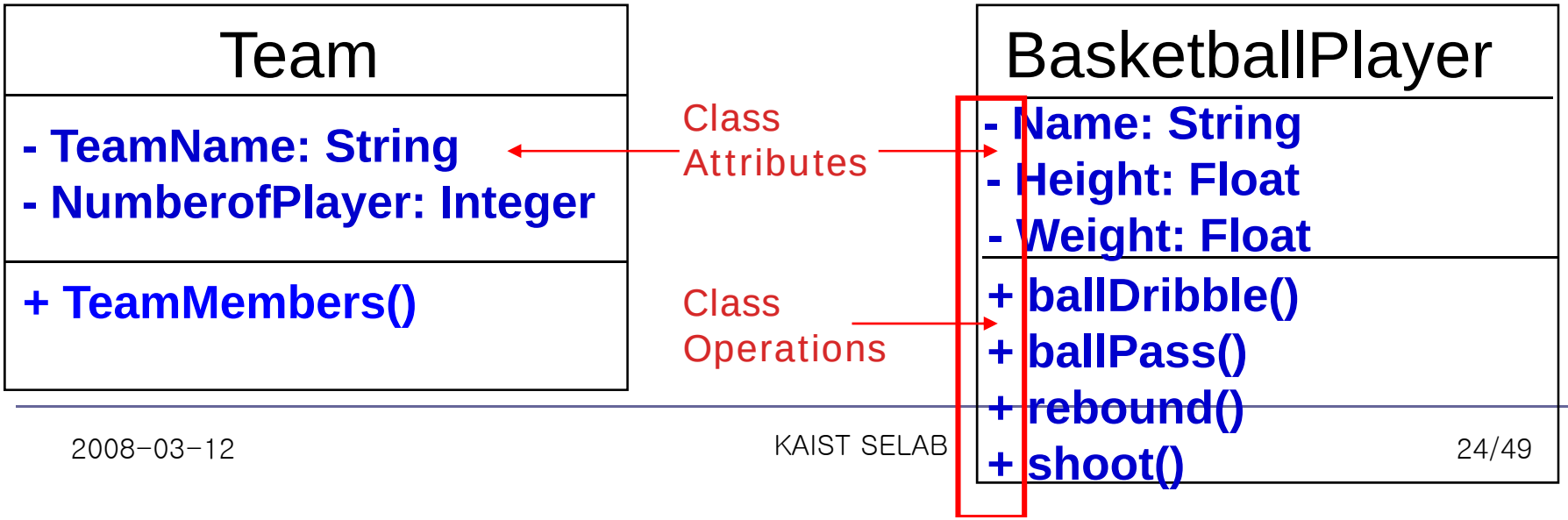
Classes

- Most important building block of any object-oriented system
- Description of a set of objects
- Abstraction of the entities
  - Existing in the problem/solution domain



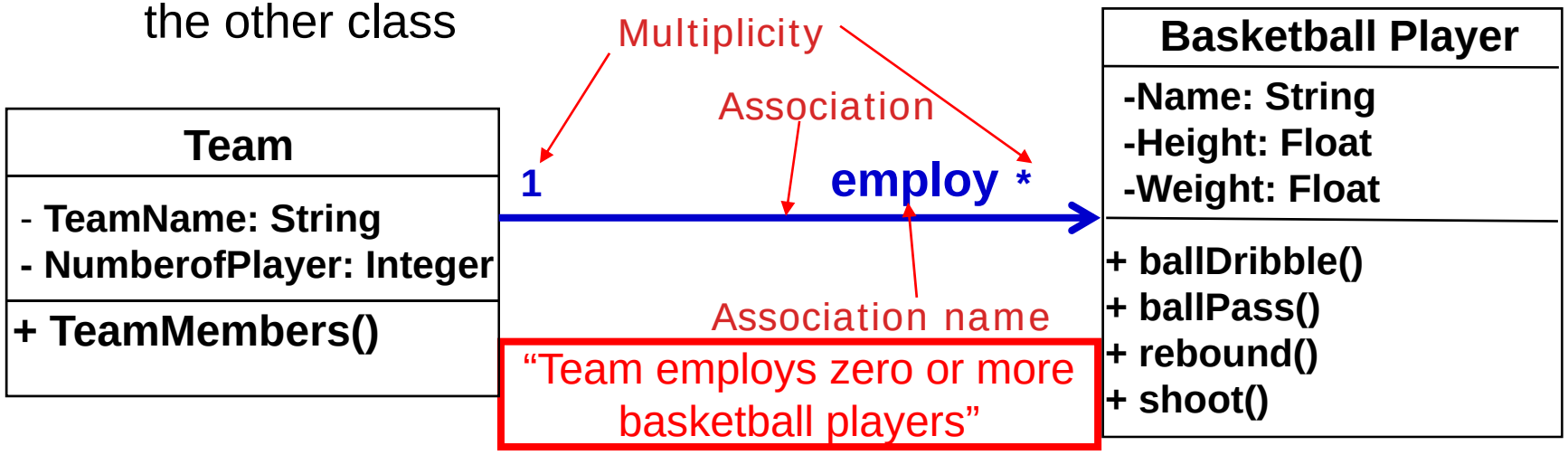
Attributes and Operations

- Attributes
  - Represent some property of the thing being modeled
  - Syntax: attributeName : Type
- Operations
  - Implement of a service requested from any object of the class
  - Syntax: operationName(param1:type, param2:type, ...) : Result



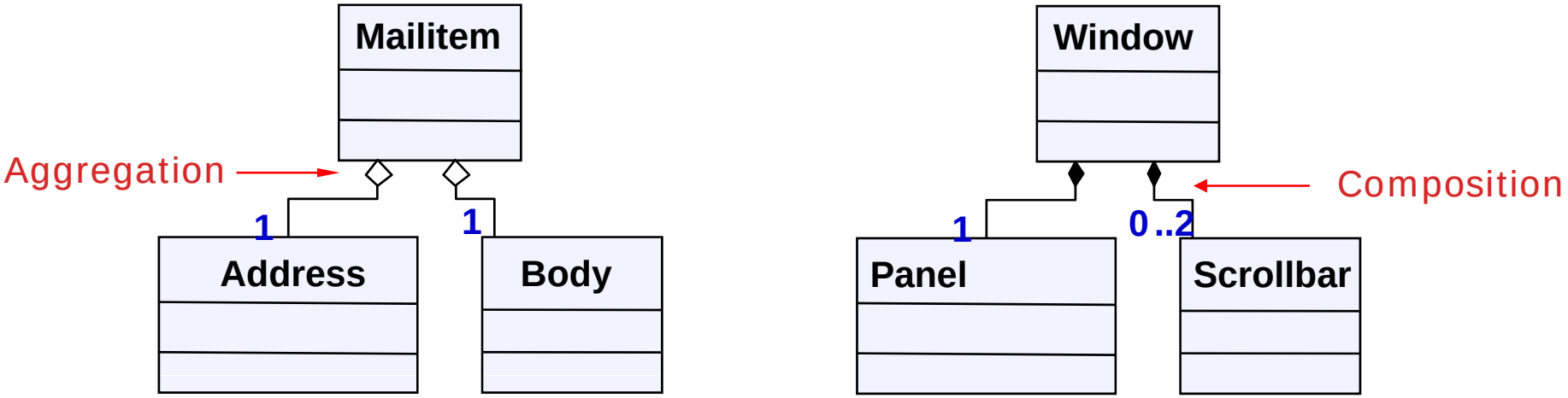
Association and Multiplicity

- Association
  - Relationship between classes that specifies connections among their instances
- Multiplicity
  - Number of instances of one class related to ONE instance of the other class



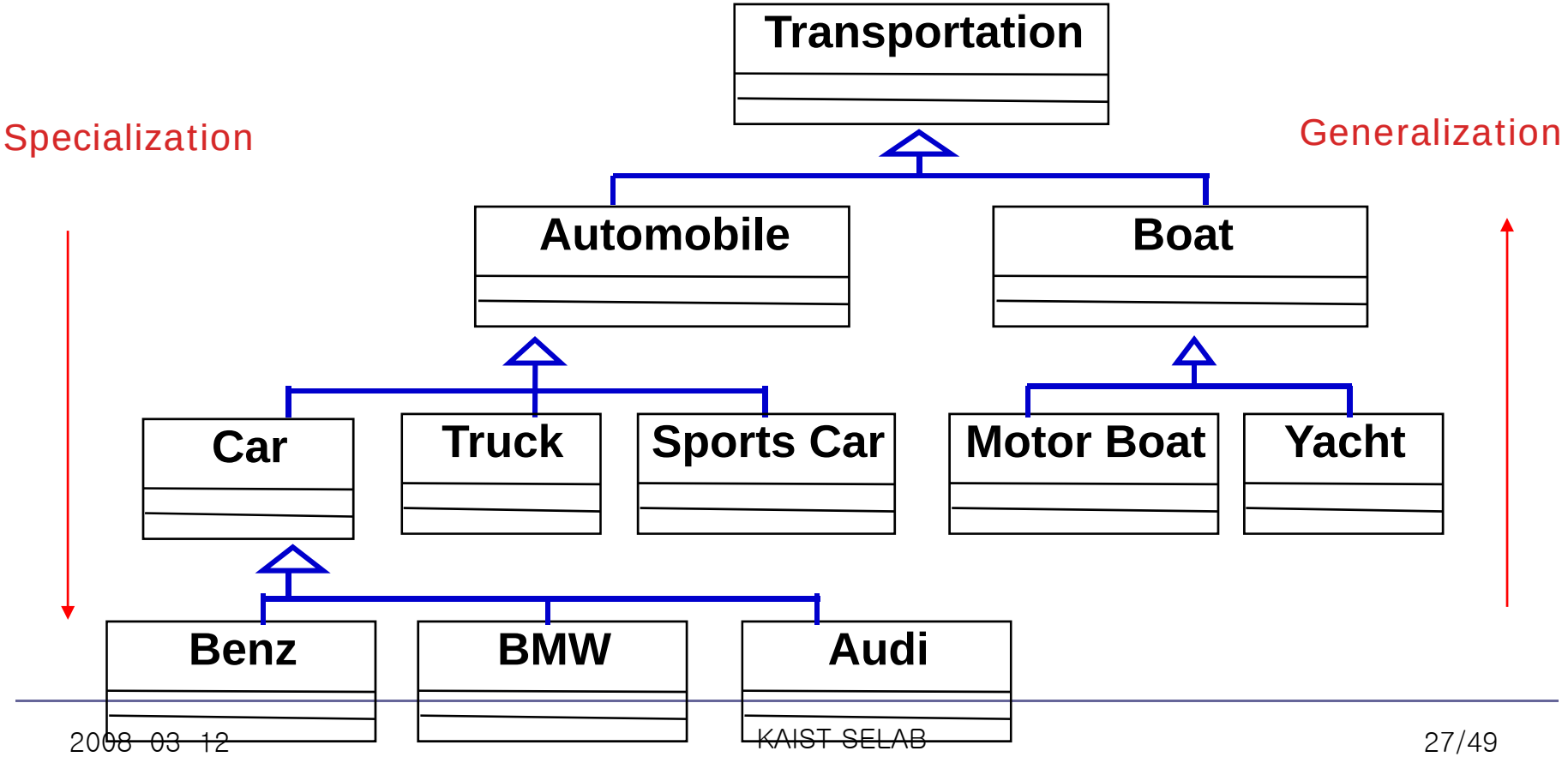
Aggregations and Compositions

- Aggregation
  - Weak “whole-part” relationship between elements
    - **Mailitem ‘has a’ address**
- Composition
  - Strong “whole-part” relationship between elements
    - **Window ‘contains a’ scrollbar**



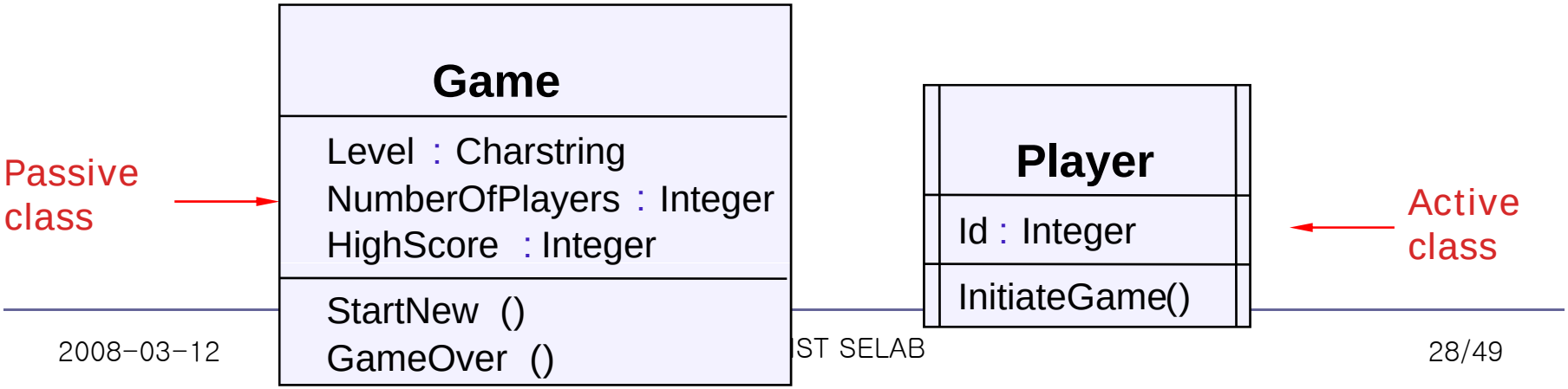
Inheritance

- Relationship between superclass and subclasses
  - All attributes and operations of the superclass are part of the subclasses



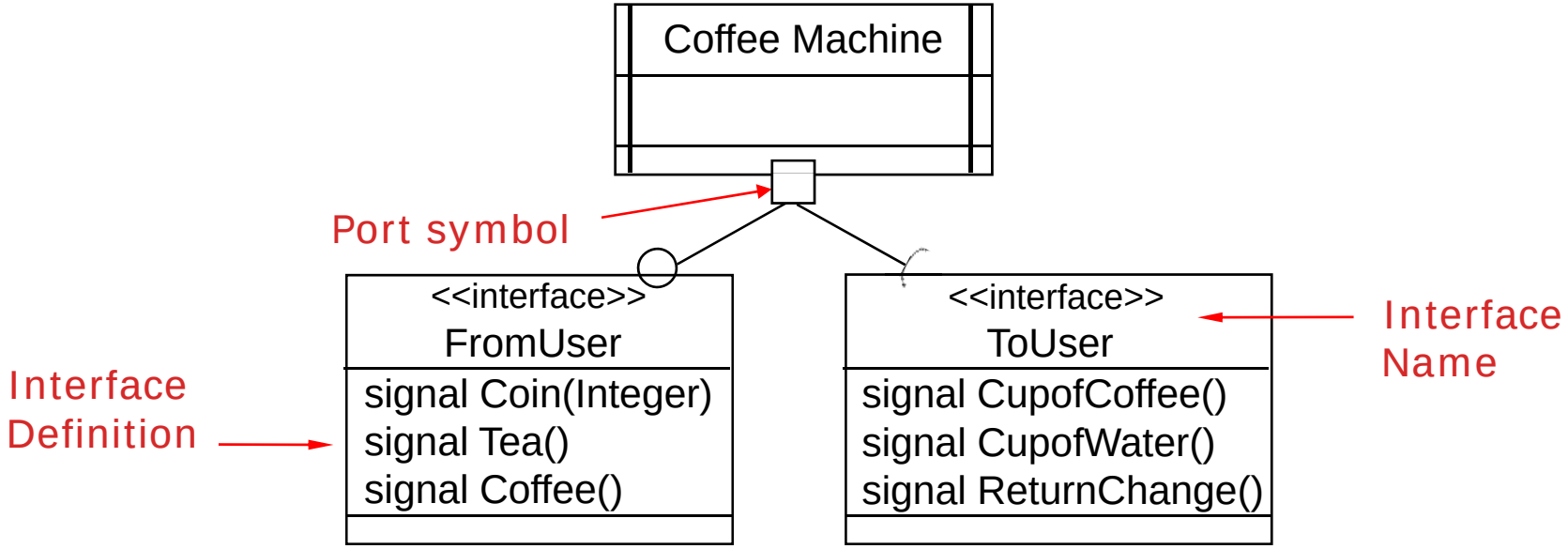
Active vs. Passive Class

- Active class
  - Own a thread control and can initiate control activity
    - **Used when asynchronous communication is necessary**
    - **Typically modeled with a state machine of its behavior**
    - **Encapsulated with ports and interfaces**
- Passive class
  - Own address space, but not thread of control
    - **Executed under a control thread anchored in an active object**



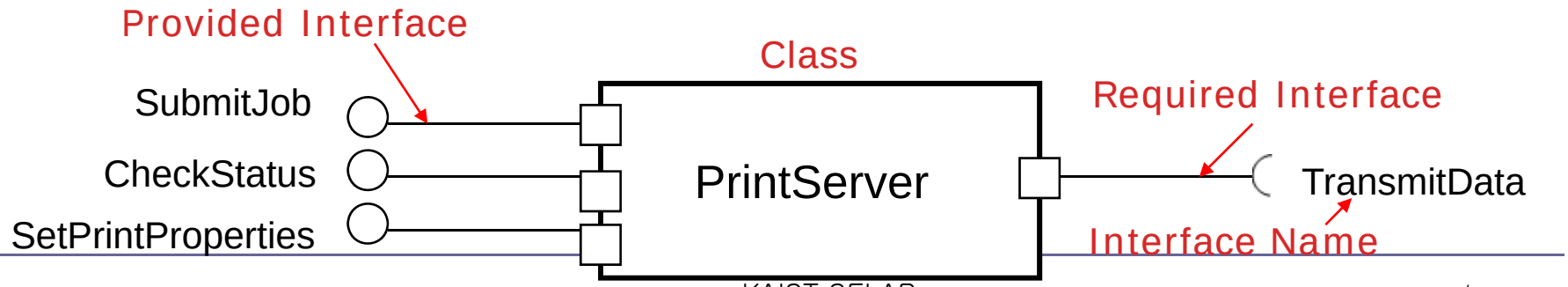
Ports and Interfaces

- Ports
  - Define an interaction point on a classifier with external environment
- Interfaces
  - Describe behavior of objects without giving their implementations
    - **Each class implements the operations found in the interface**

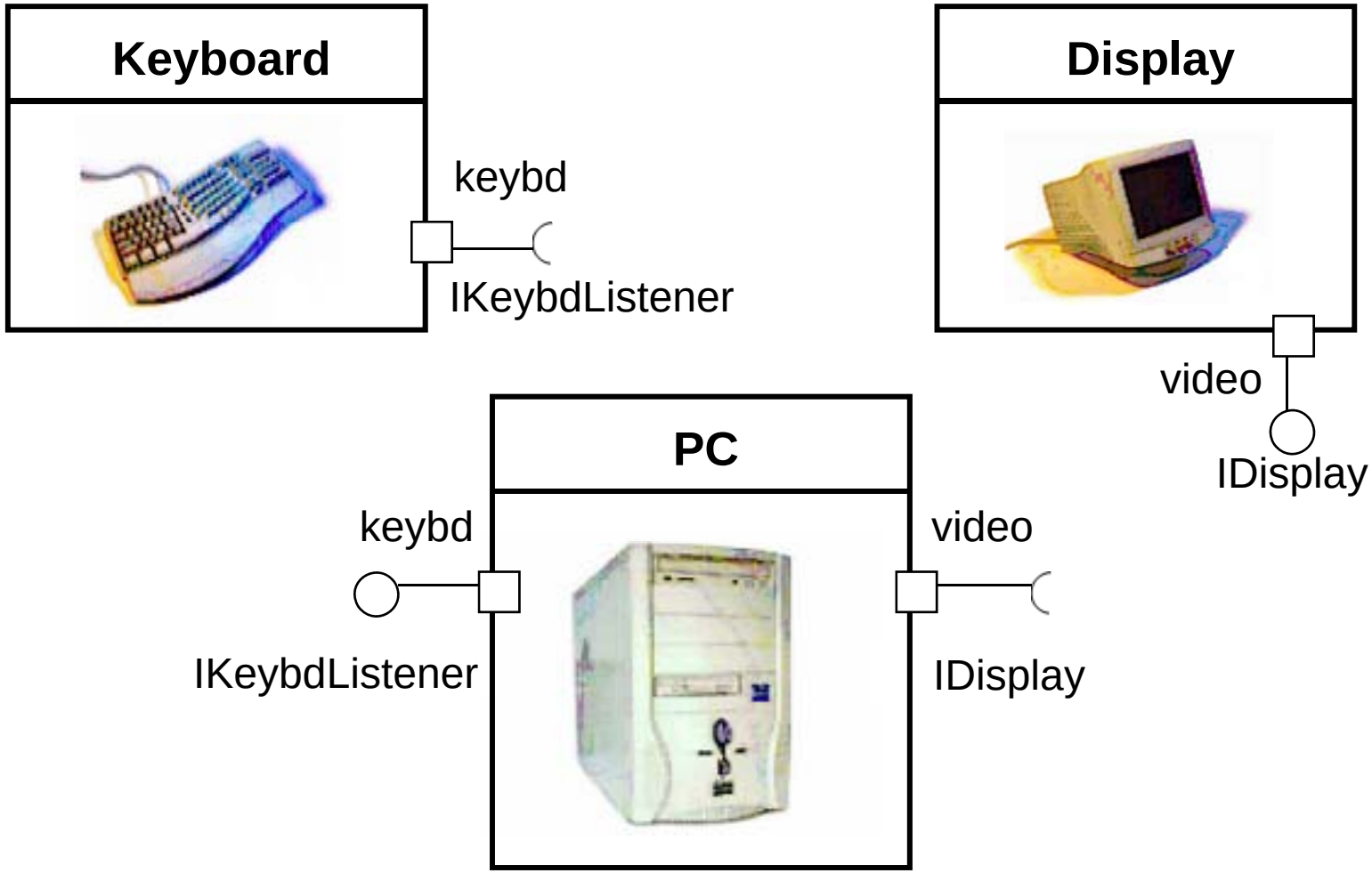


Provided/ Required Interface

- Provided interface
  - Class provides the services of the interface to outside callers
  - What the object can do
  - Services that a message to the port may request (incoming)
- Required interface
  - Class uses to implement its internal behavior
  - What the object needs to do
  - Services that a message from the port may require from external environment (outgoing)



Computer Device Example



Tips for Class Modeling

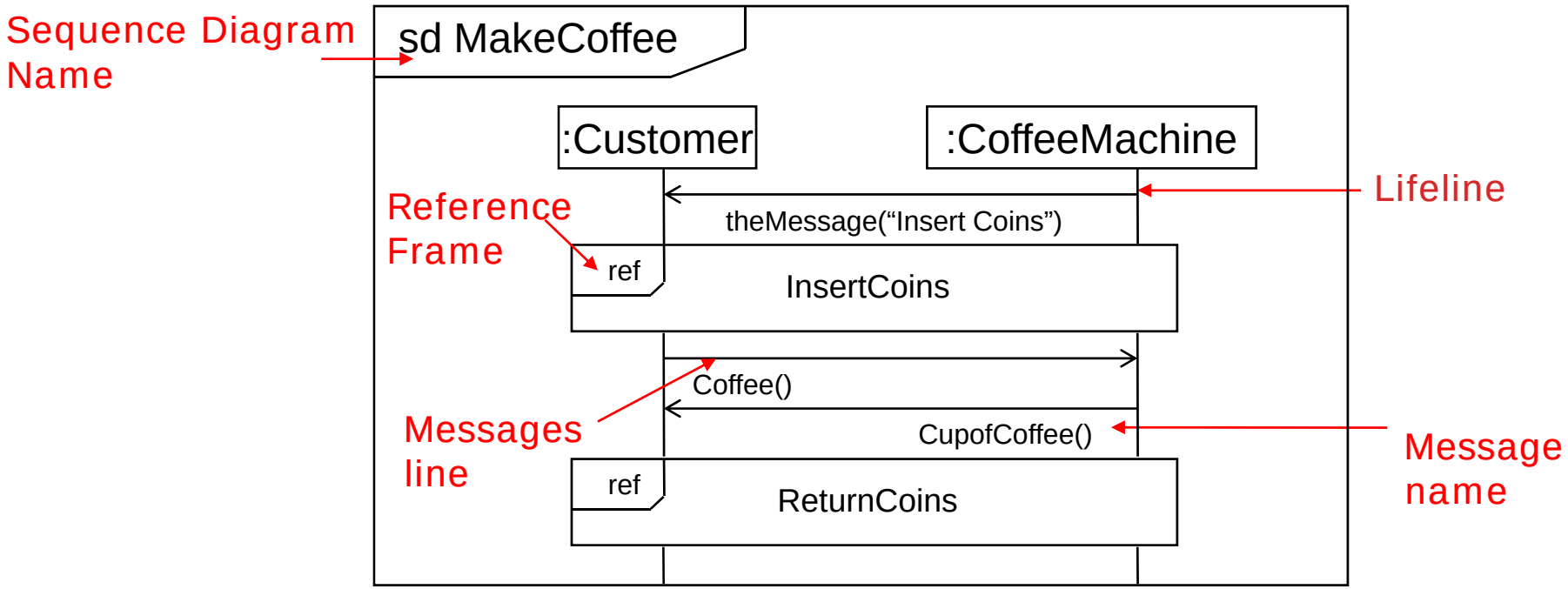
- Finding Classes
  - Do we have that should be stored or analyzed ?
  - Do we have external system ?
    - **External system is modeled as class**
  - Do we have any patterns, class libraries, components, and so on ?
  - Are there devices that the system must handle ?

(From :oopsla.snu.ac.kr/research/UML/)

Sequence Diagrams

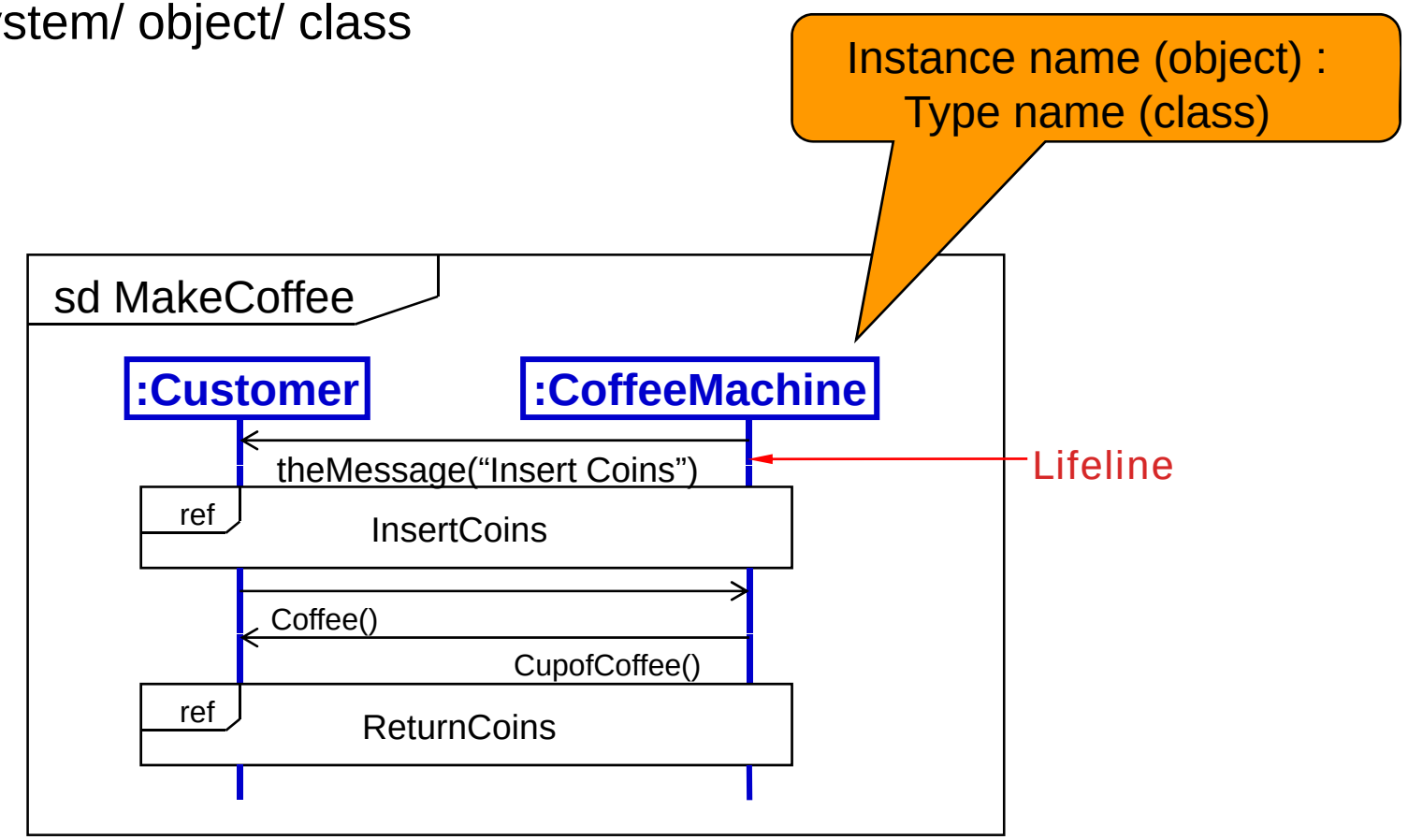
Sequence Diagrams

- Show sequences of messages (“interactions”) between instances in the system
- Emphasize time ordering



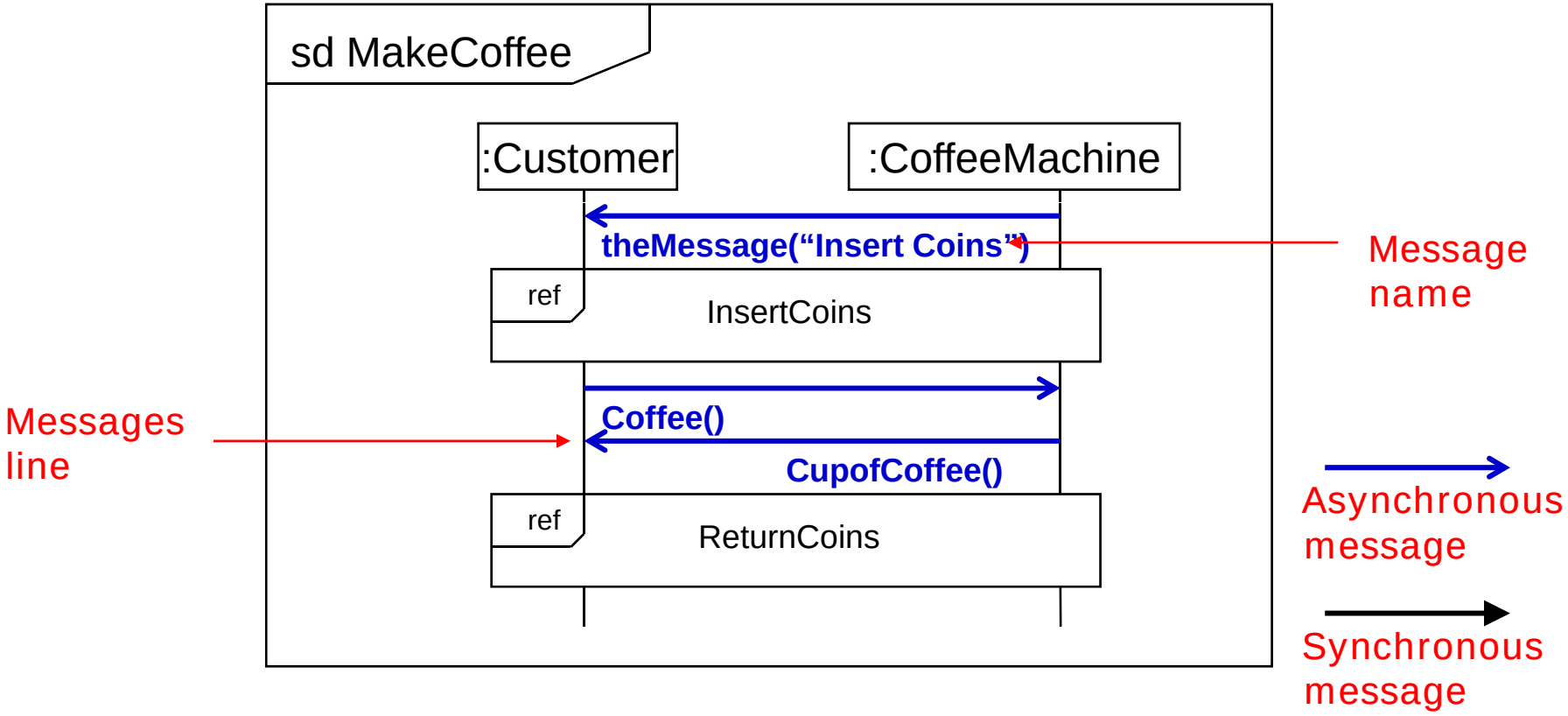
Lifelines

- Individual participant in the interaction over period time
  - Subsystem/ object/ class
  - Actor



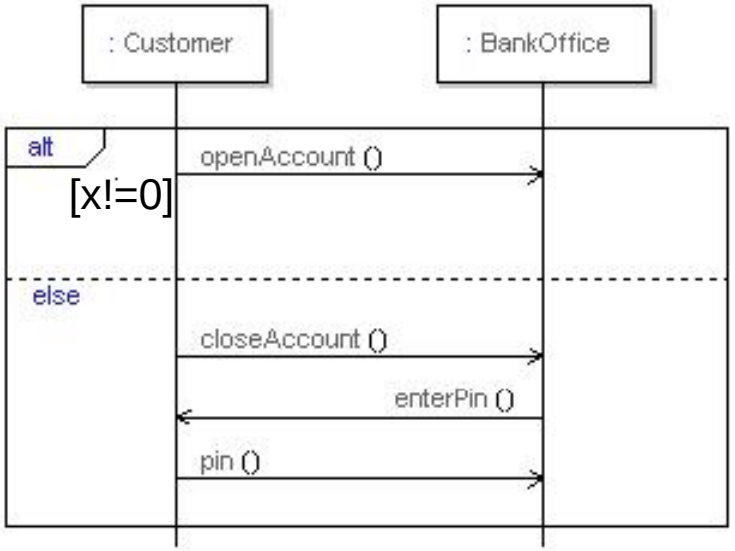
Messages

- One-way communication between two objects
- May have parameters that convey values



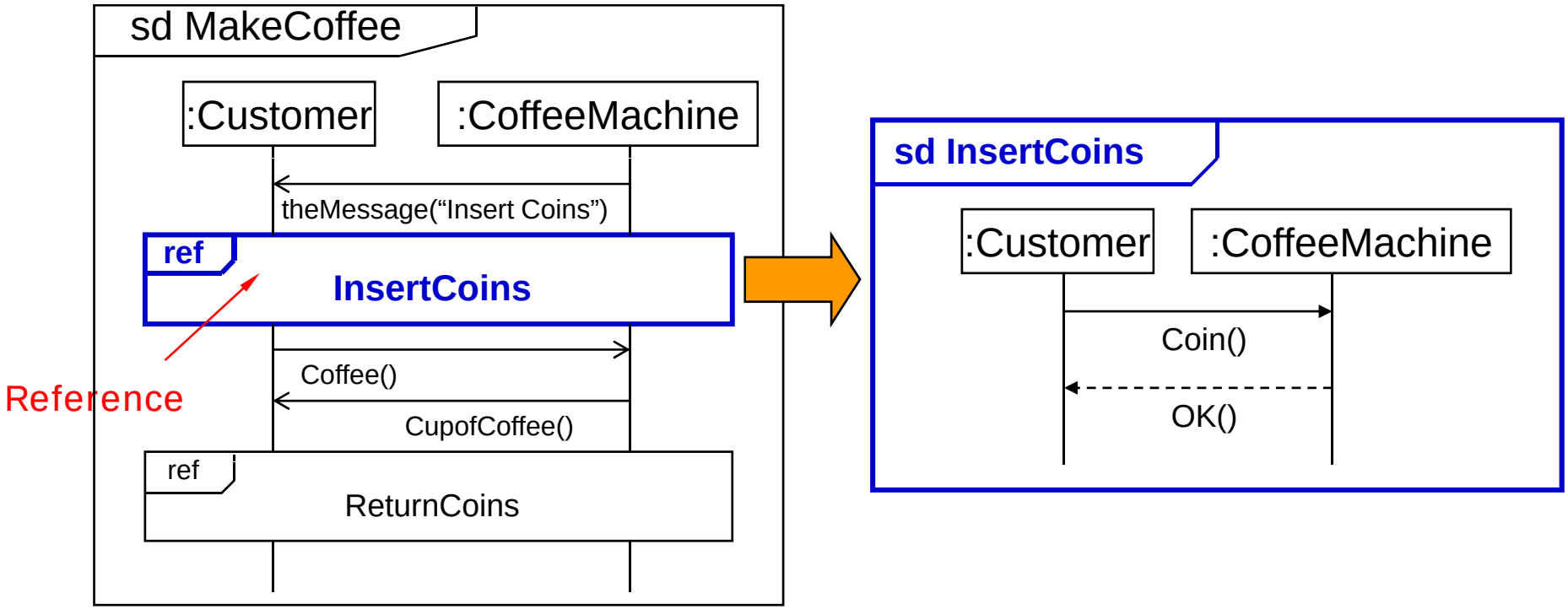
Combined Fragment Frame

- Defines an expression of interaction fragments
- Interaction operators define how the contents describe behavior
  - Alt: each section is one alternative
    - **E.g. alt [a>0]**
  - Ref: reference to another Use Case
  - Loop: specifies a repeated sequence of behavior
    - **E.g. 'loop [1,5]', 'loop [6]'**



Referencing

- Reuse already existing sequence diagrams
  - Avoid unnecessary duplication



Tips for Sequence Diagram

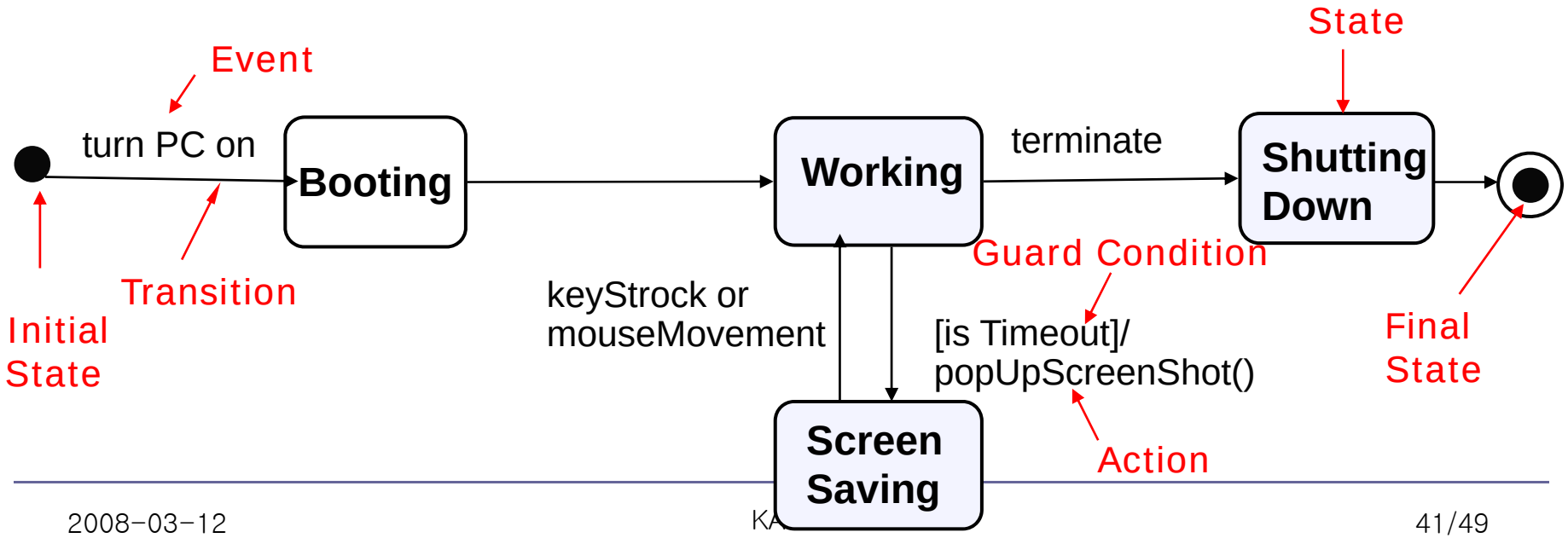
- Set the context for the interaction.
  - E.g. one use case
- Express the flow from left to right and from top to bottom.
- Put active instances to the left/top and passive ones to the right/bottom.

(From :oopsla.snu.ac.kr/research/UML/)

State Machine Diagrams

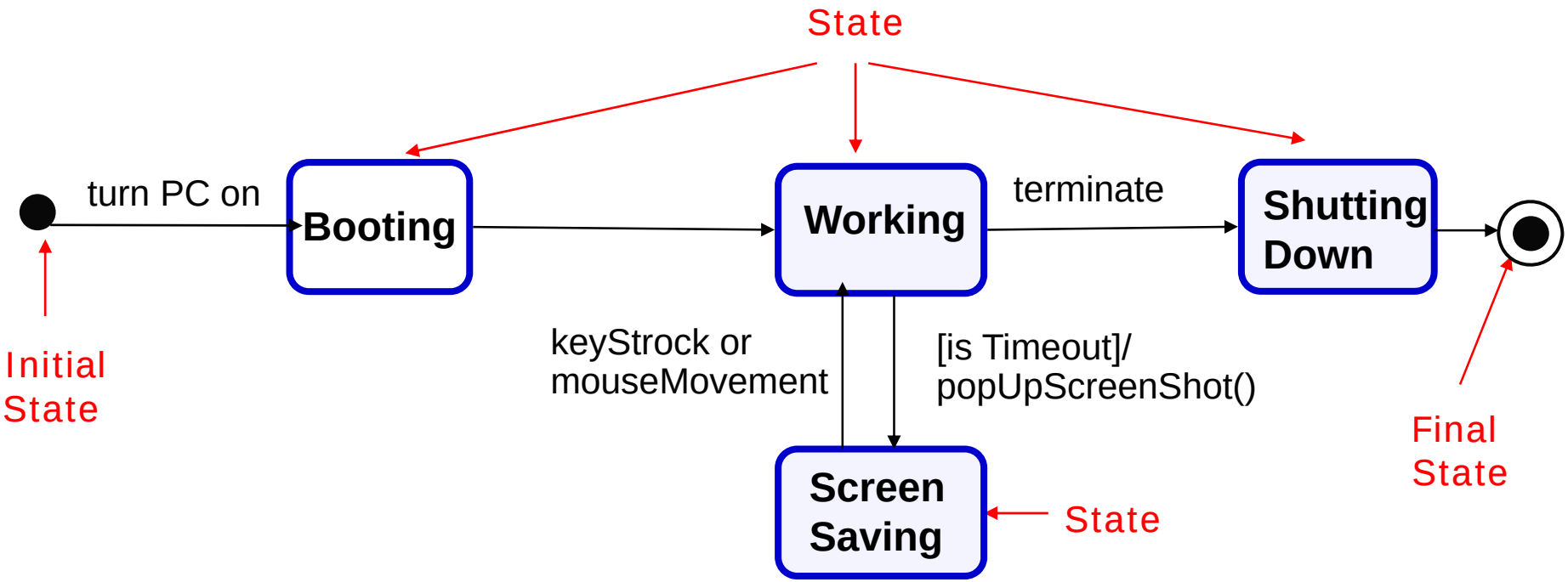
State Machine Diagrams

- Describe the dynamic behavior of objects over time by modeling the lifecycles of objects of each class
- Show
  - The event that cause a transition from one state to another
  - The actions that result from a state change



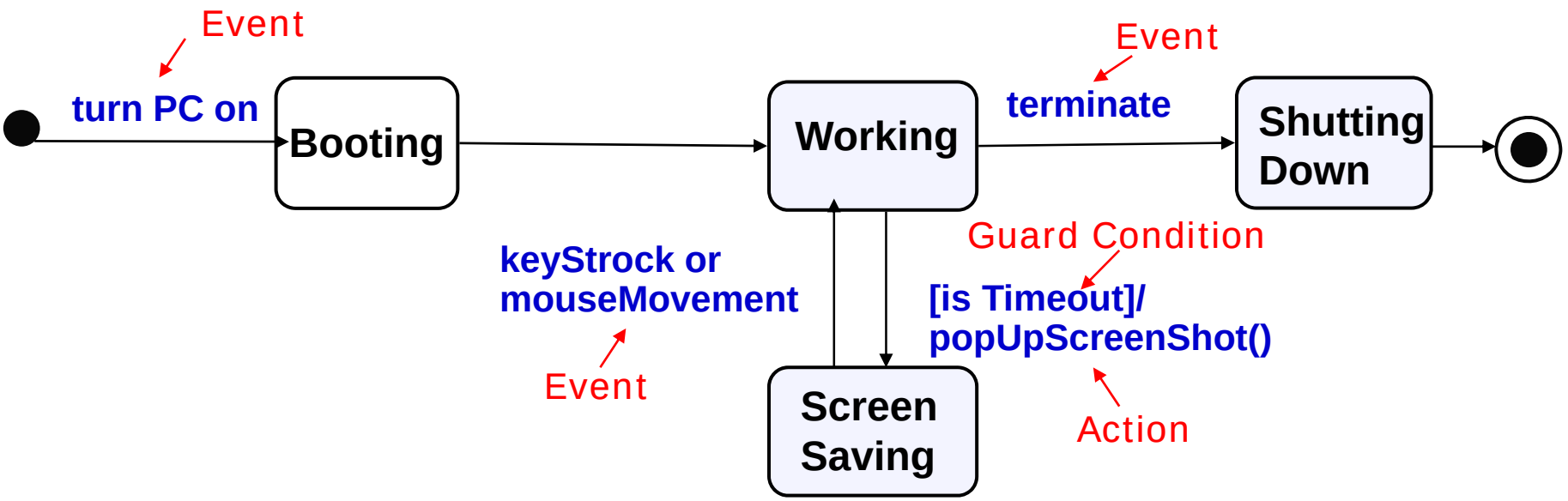
States

- State
  - Condition or situation during the life of an object
    - **Satisfies some condition, performs some activity or waits for some event**



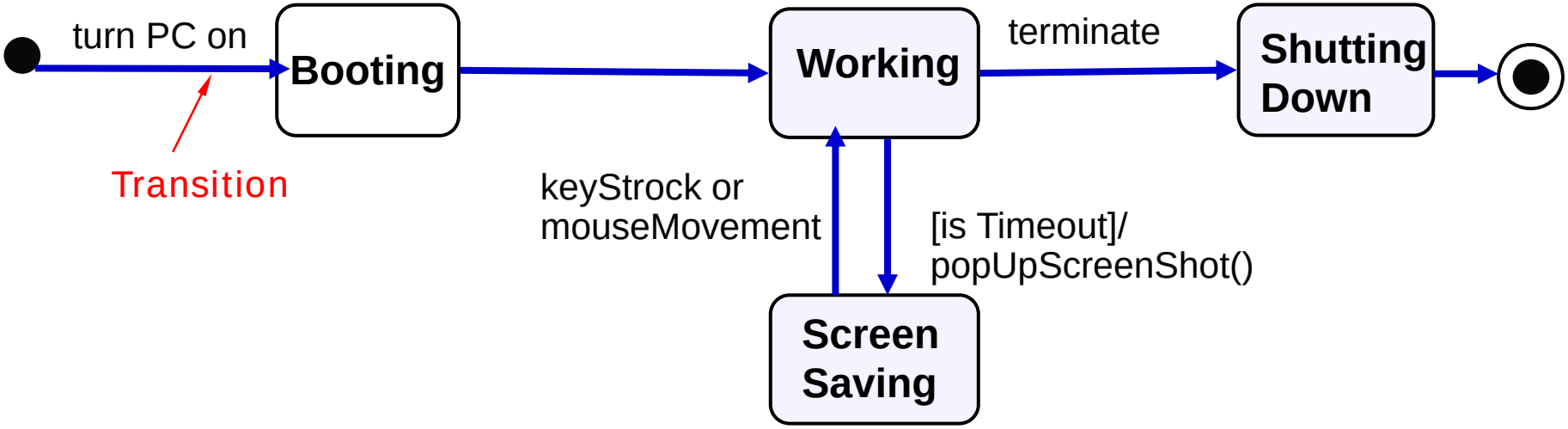
Event and Action

- Event
  - Stimulus which causes the object to change state
- Action
  - Output of a signal or an operation call



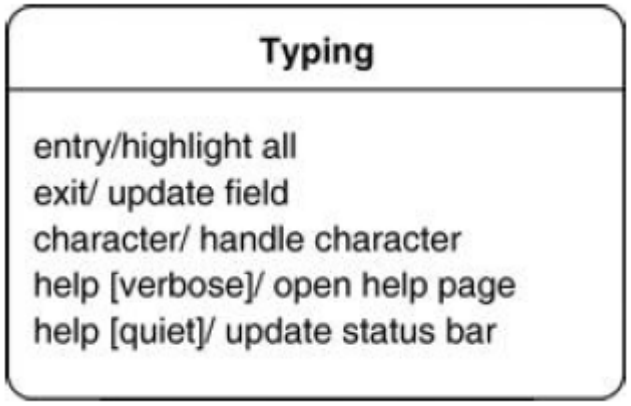
Transition

- Change state from one to another triggered by an event
- Occur only when guard condition is true
- Syntax: event(arguments)[condition]/action



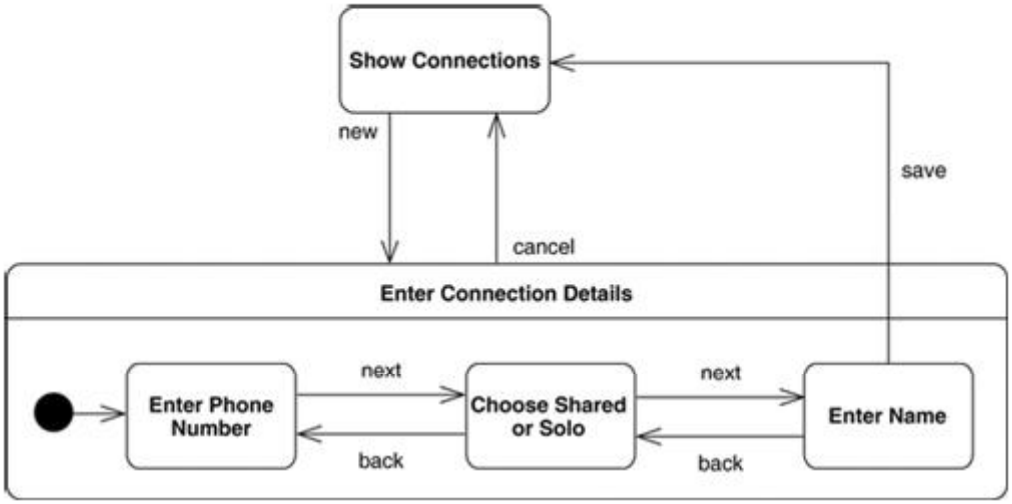
Internal Activities

- States can react to events without transition
  - Putting the event, guard, and activity inside the state box
  - Two special activities
    - **The entry and exit activities**
- Internal activities do not trigger the entry and exit activities



Superstates

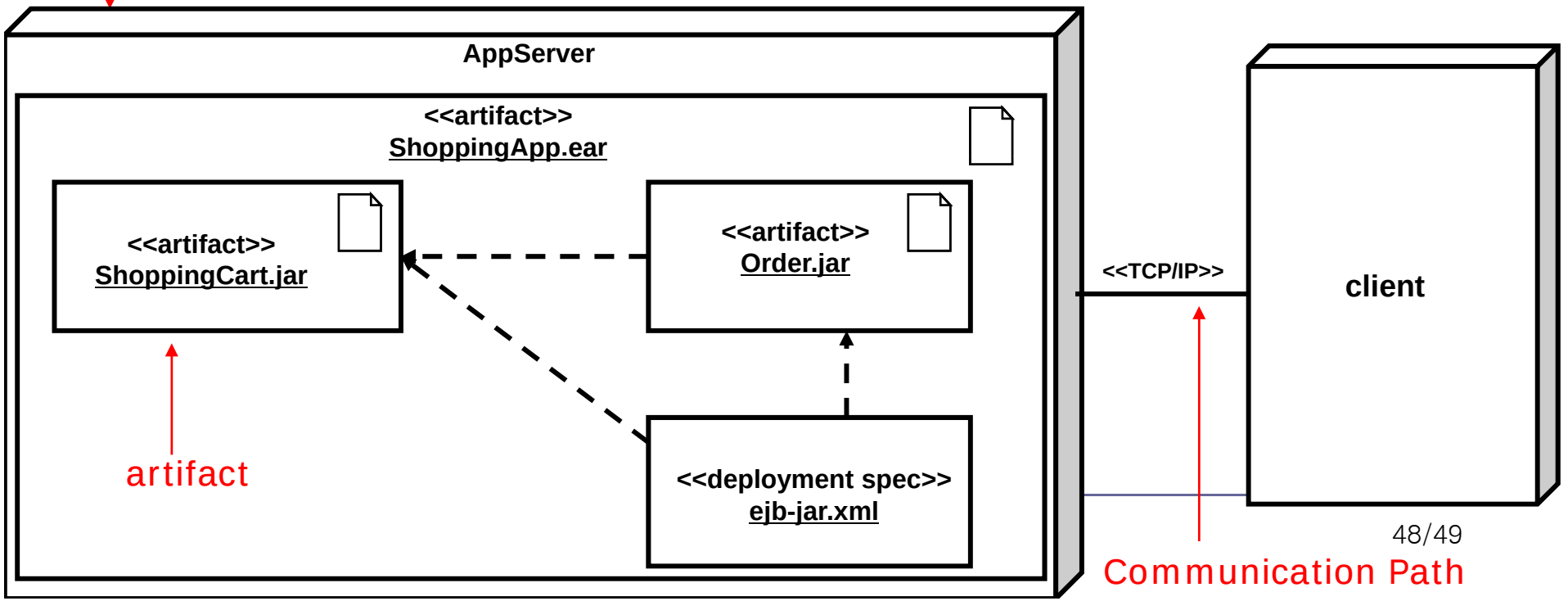
- Several states share common transitions and internal activities
  - Move the shared behavior into a superstate
  - A behavior can be expressed in a modular/hierarchical way



Deployment Diagrams

Deployment Diagrams

- Show runtime architecture of devices, execution environments, and artifacts in architecture
  - Physical description of system topology
    - **Describe structure of hardware units and software executing on each unit**

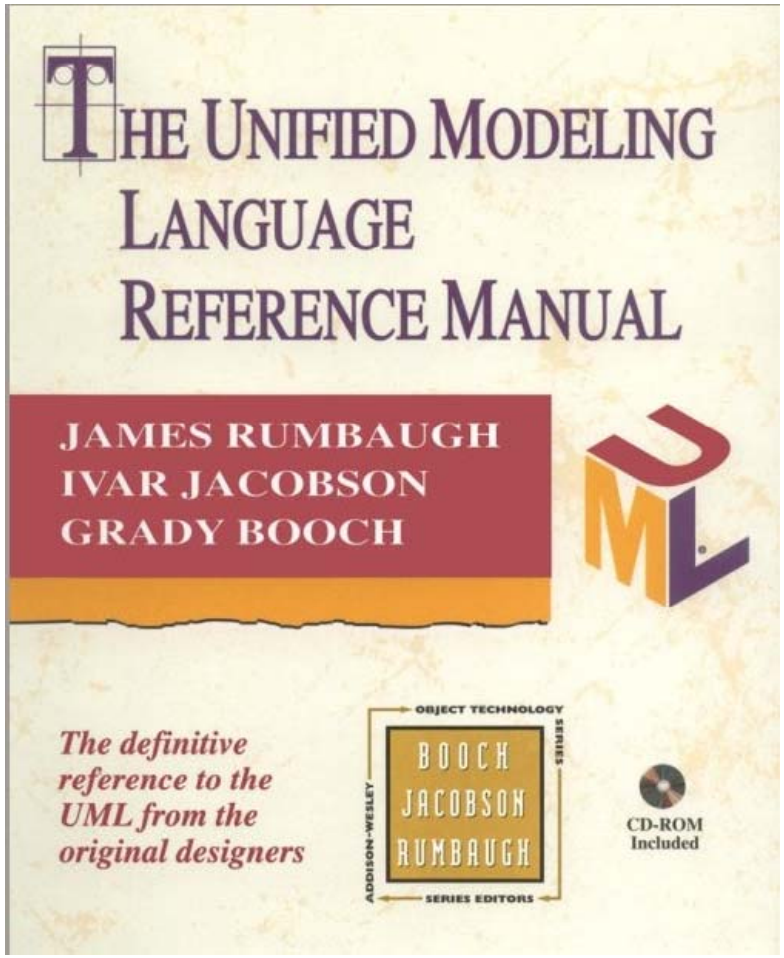


Deployment Diagrams

- Node
  - Computational resource upon which artifacts may be deployed for execution
- Communication path
  - Show connection between nodes
    - **Stereotype can be used for communication protocol or network used**
- Artifact
  - Specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system.
    - **Examples of artifacts include model files, source files, scripts, and binary executable files, a table in a database system, a development deliverable, or a word-processing document, a mail message.**

Summary

- UML can be used as
  - Sketch level
  - Blue print level
  - Programming language level
- Use appropriate UML diagrams for different goals
  - If you just starts your SE projects, start with
    - **Use-case diagrams with use-case texts**
  - If you want to look at behavior across many use cases or many threads,
    - **Activity diagram**
  - If you want to look at the behavior of several objects within a single use case,
    - **Sequence diagrams**
  - If you want to look at the behavior of a single object across many use cases,
    - **State diagrams**



Concurrent states