
Development of Component- and Service-Oriented Software Architectures with UML and UML-RT — Use Cases and Activity Diagrams —

Ingolf H. Krueger
ikrueger@ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA



California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA



Overview

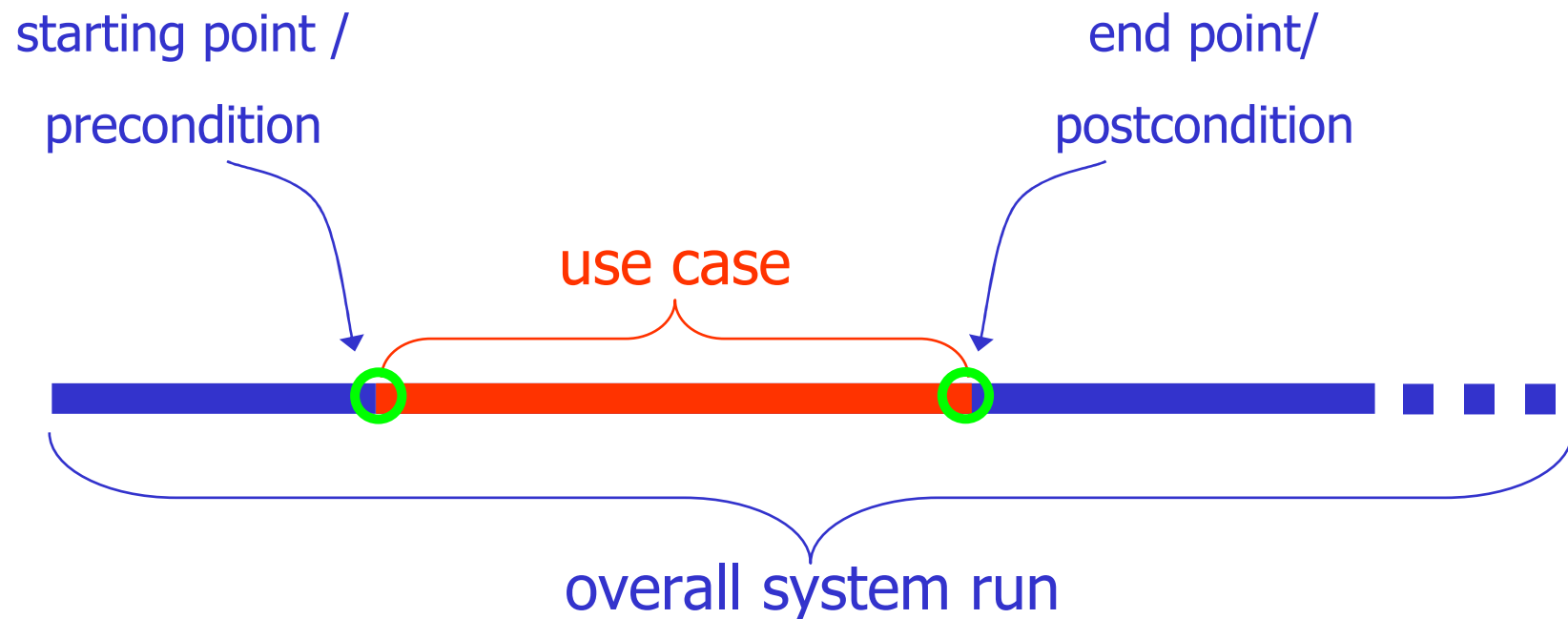
- Use Cases
- Activities
- From Use Cases and Activities to Classes and Objects

Use Cases

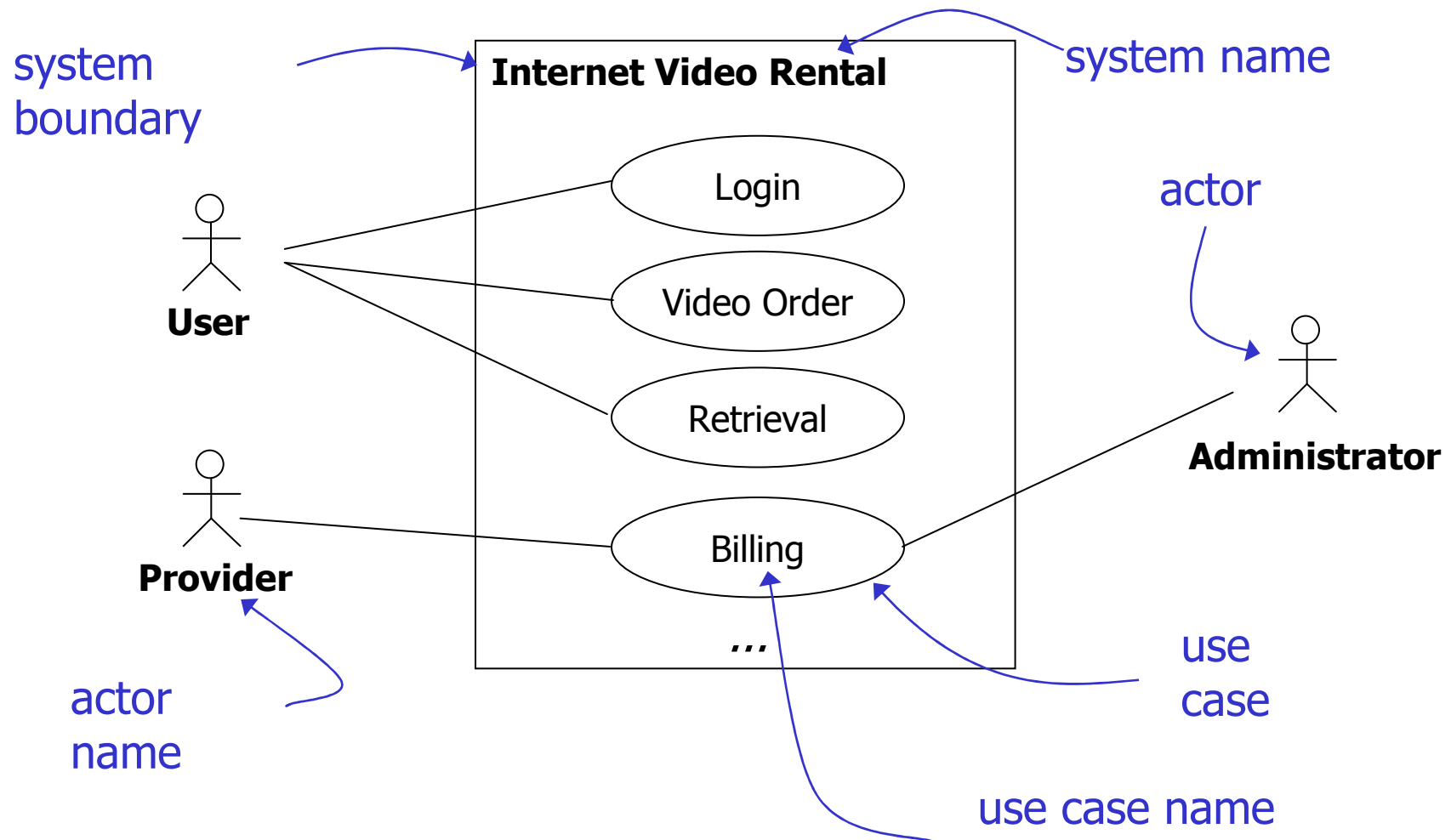
- Determine system boundary
- Determine external actors
- Goal orientation:
 - What services does the system offer?
 - What problem does the system solve?
 - What subtasks/-problems are there?
 - What (sub-)steps are necessary to solve the problem(s)?
 - How are these steps connected?
- Interactive and iterative process!

What's in a Use Case?

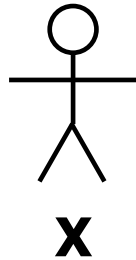
A **use case** is a **coherent sequence of actions** of the system under consideration, describing an externally observable functionality / system service; typically, this includes listing alternative scenarios and exceptions.



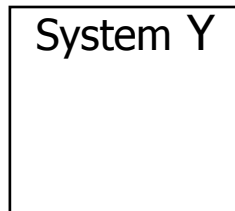
Use Case-Diagrams: Basic Notation



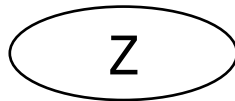
Use Case-Diagrams: Basic Notation



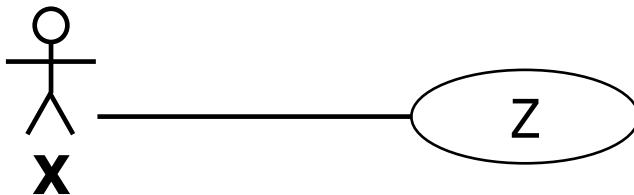
actor X



system boundary of system Y

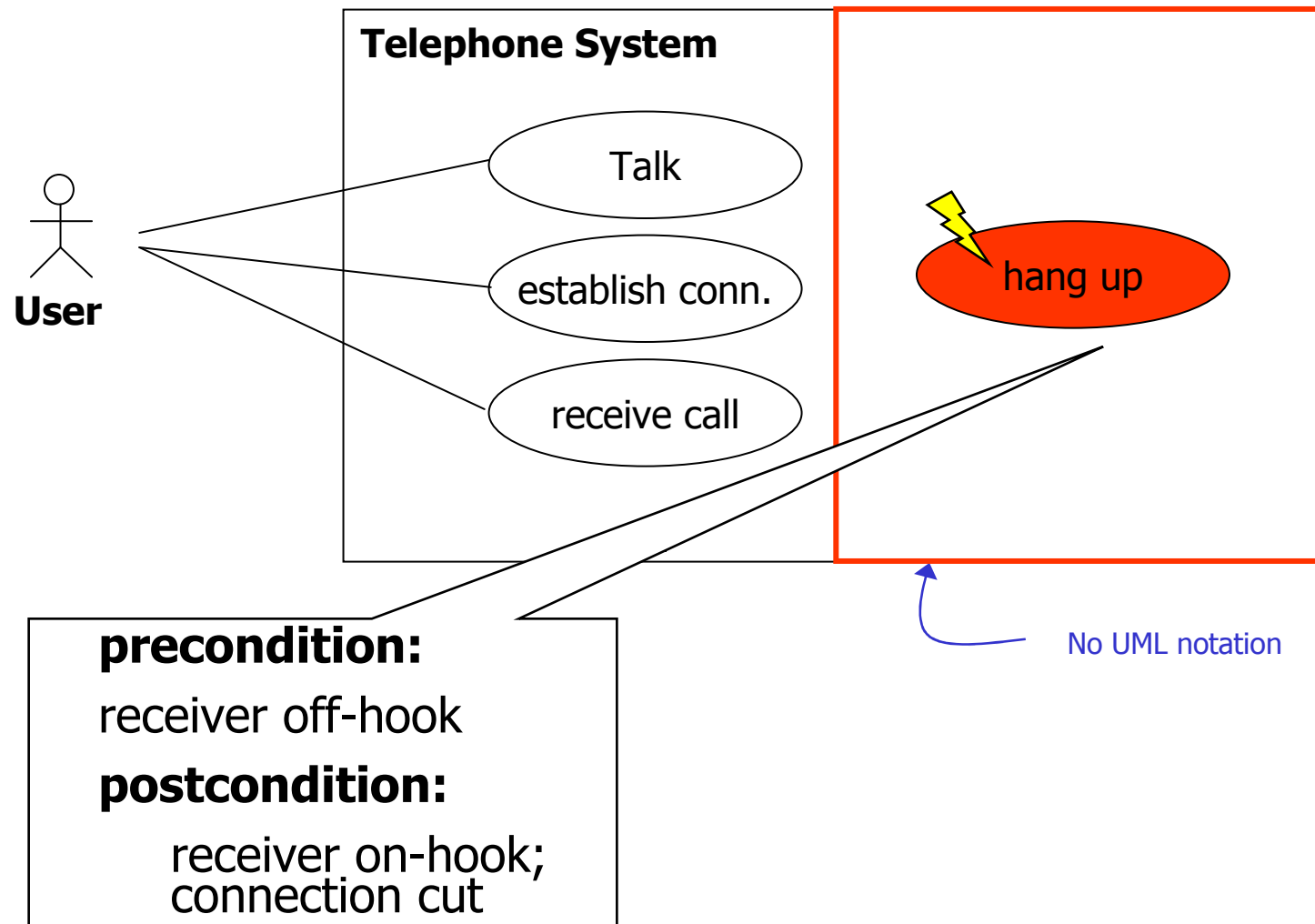


use case Z



actor X participates in use case Z

Use Cases and Exceptions



Relationships between Use Cases

- Inclusion
- Extension
- Generalization

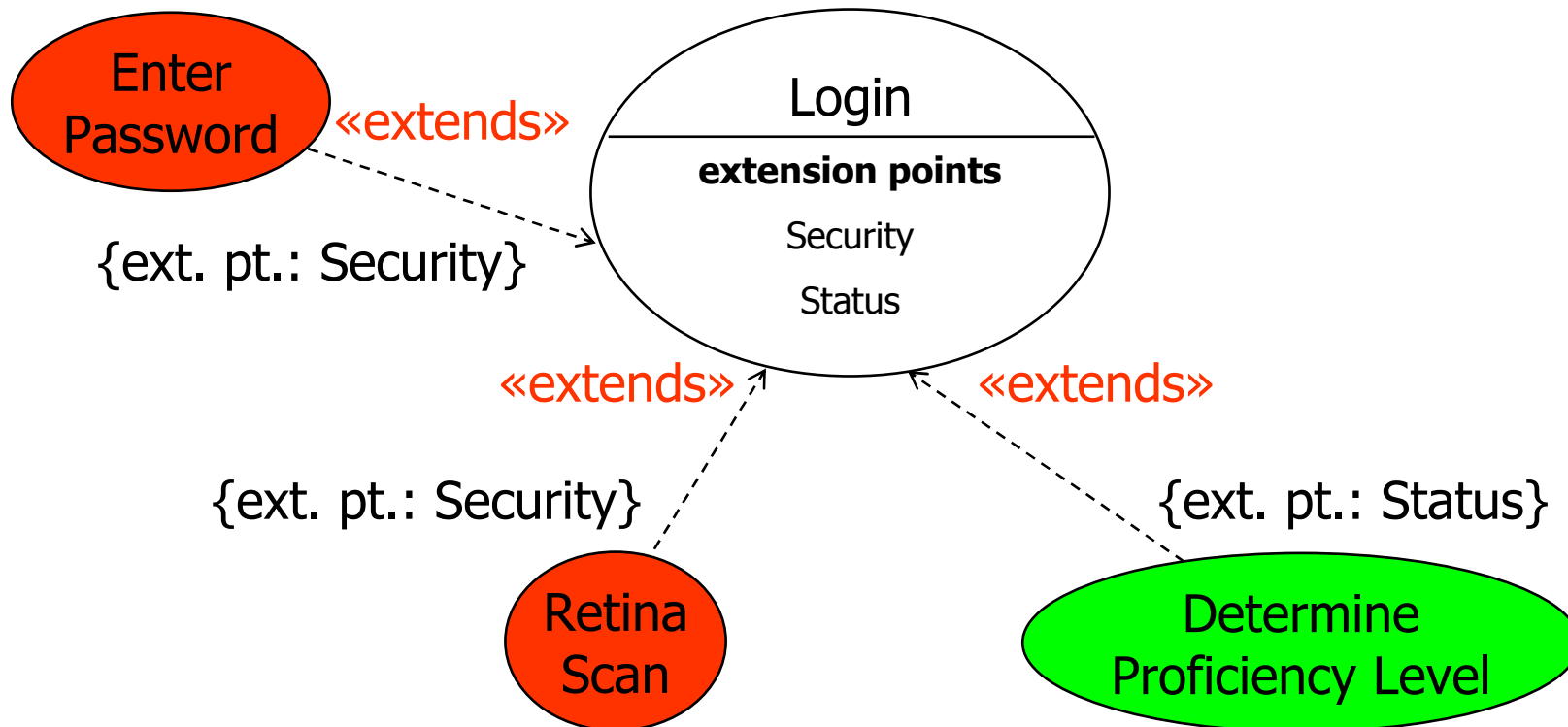
Relationships between Use Cases: Inclusion



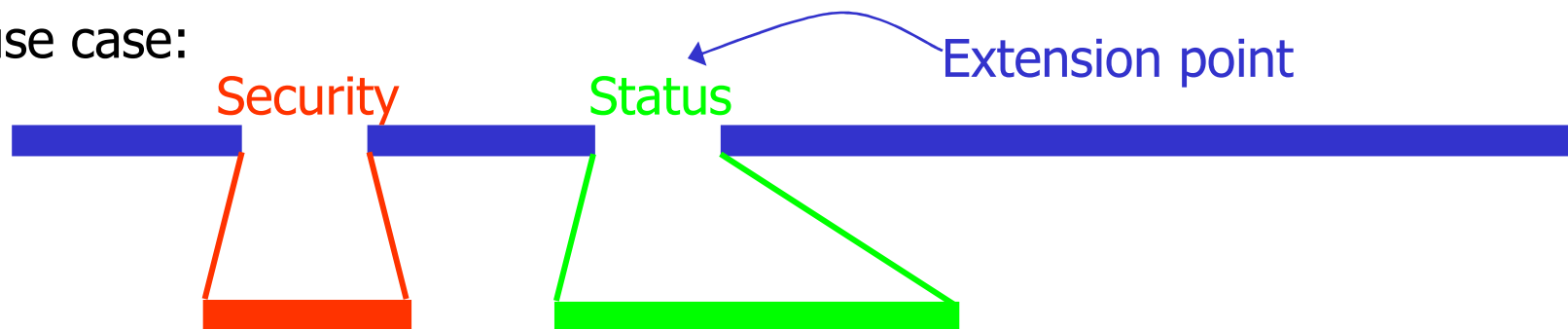
use case:



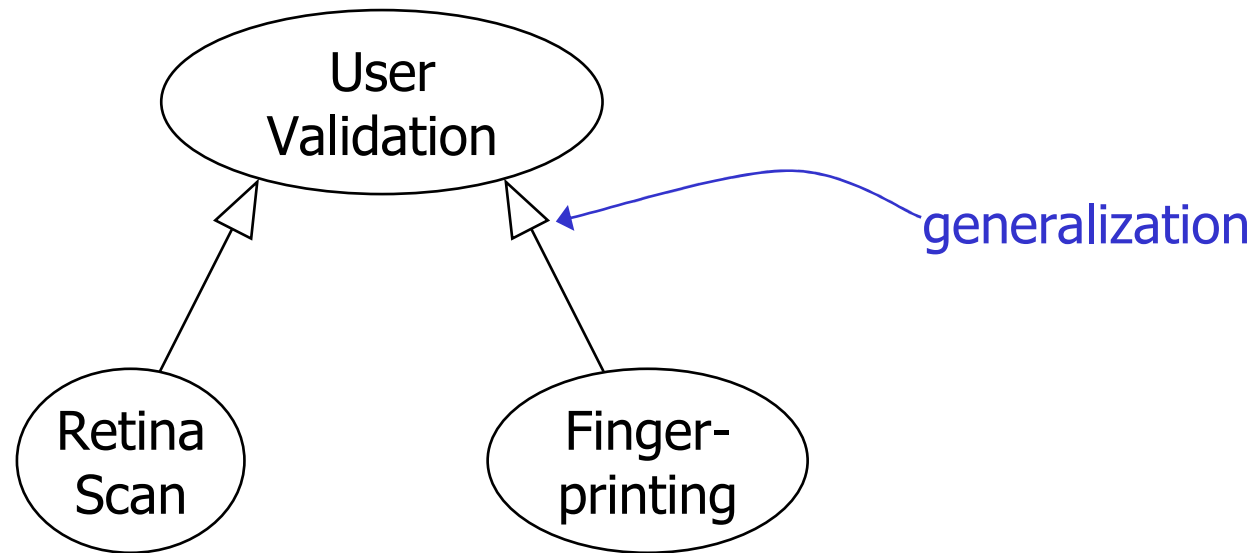
Relationships between Use Cases: Extension



use case:

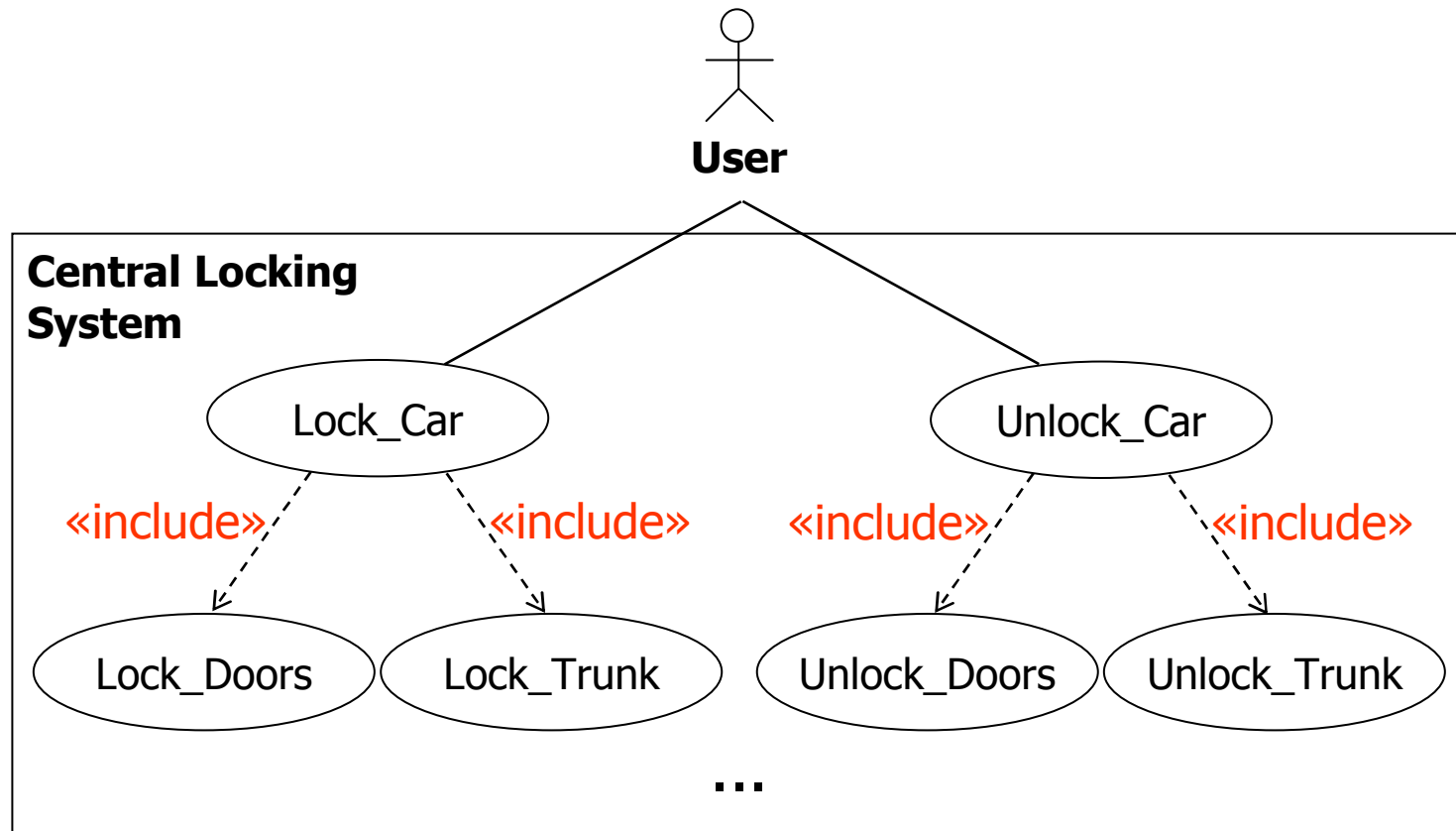


Relationships between Use Cases: Generalization



use case "Retina Scan" inherits the properties of "User Validation"

Use Cases: Example



Use Cases - Summary

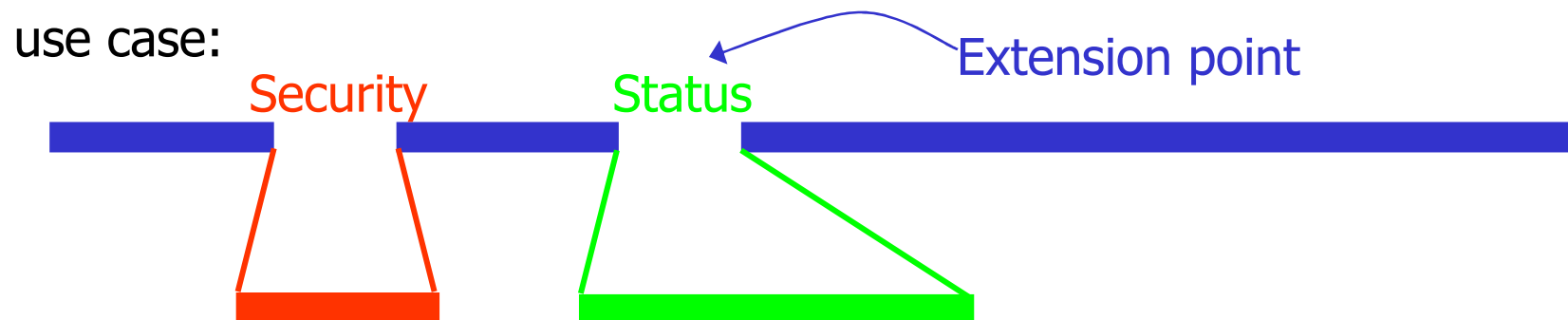
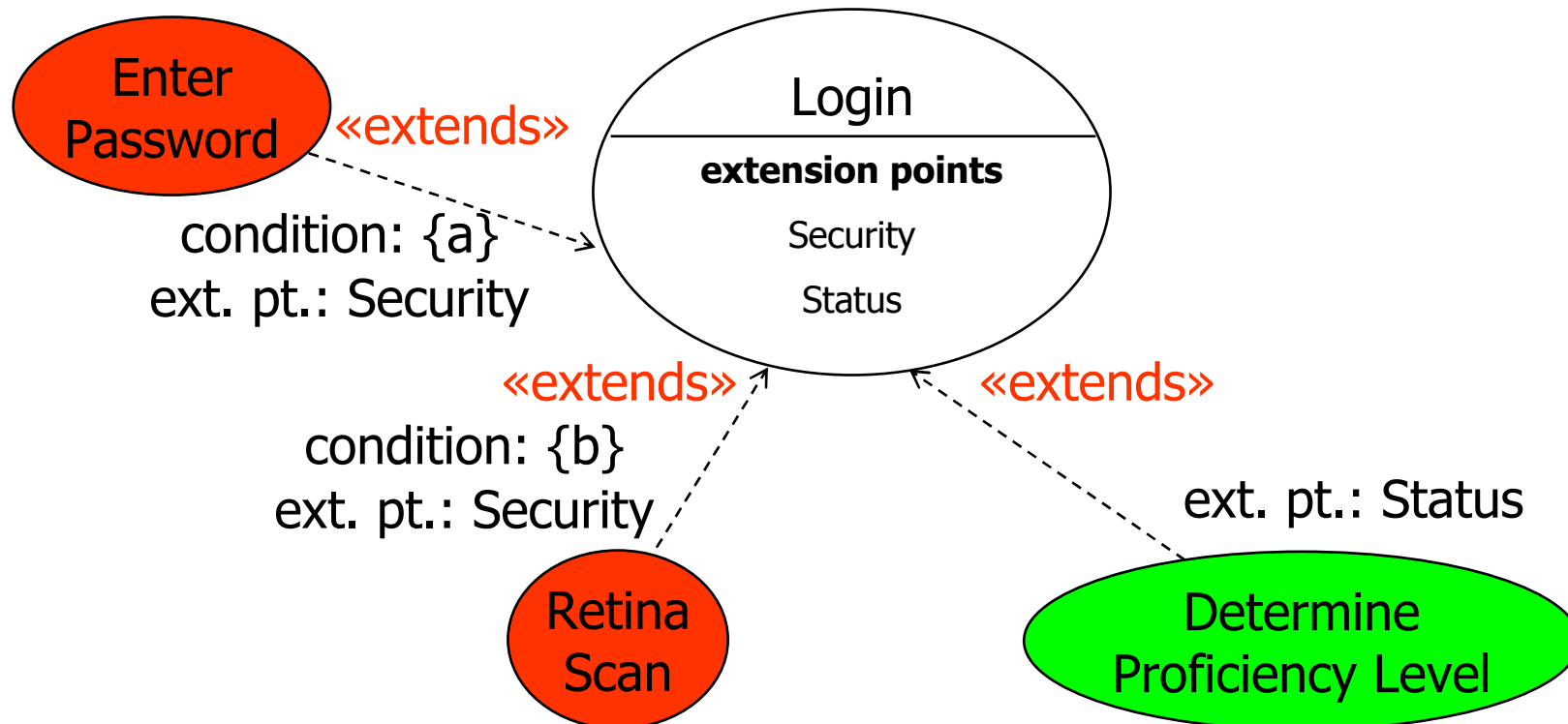
- Goal: identification of key system services
- Determine system boundary
- Determine key actors
- Goal-oriented structuring of the externally visible behavior
- Elicitation of relationships between system services
- Contrast with features/user stories

Changes in UML 2.0

Use cases

- Conditional extensions:
 - conditions show the actual logic necessary for one use case to extend another.
 - they also show the exact extension point that is used between the two use cases.

Relationships between Use Cases: Extension



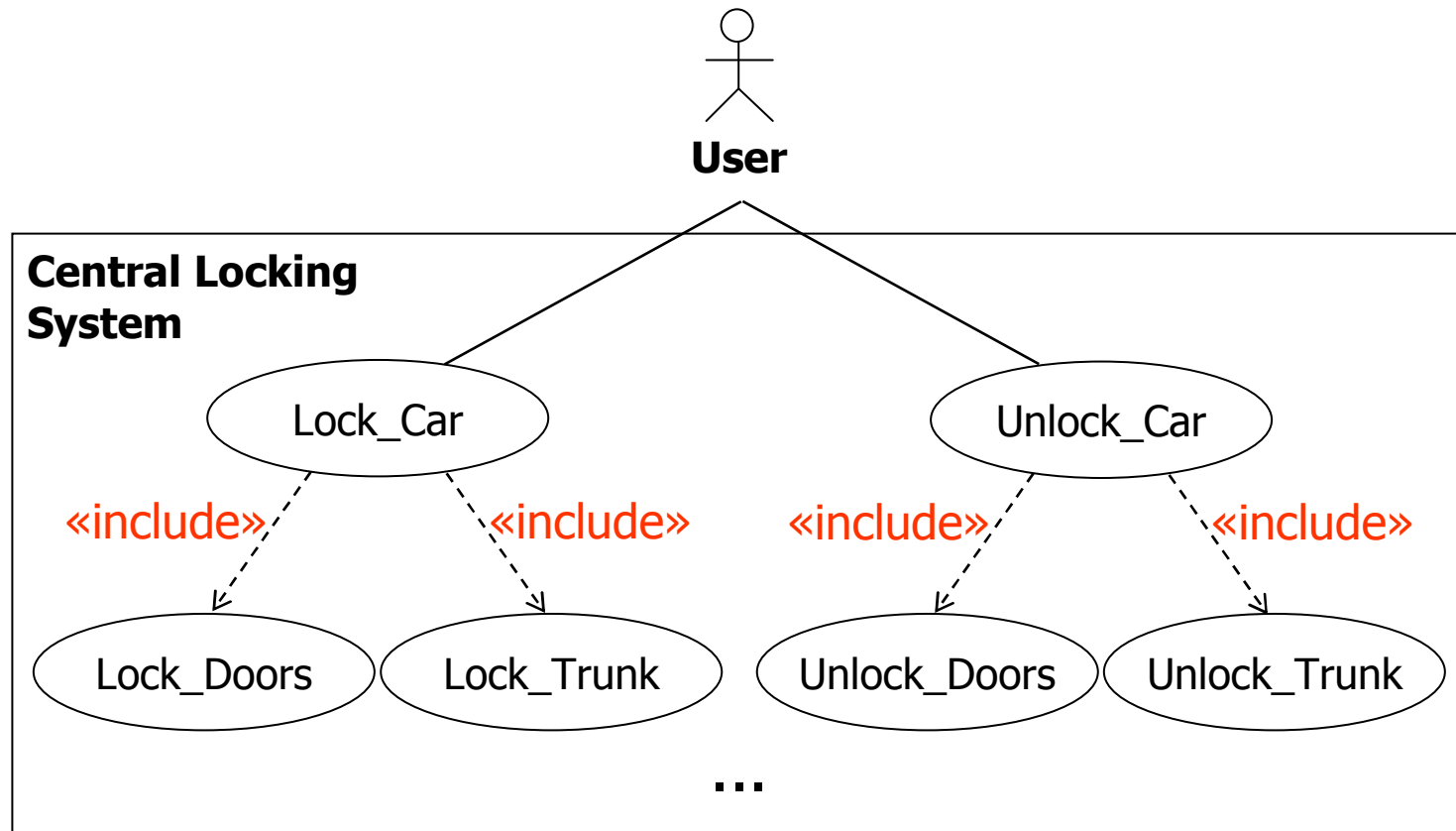
Overview

- Use Cases
- Activities
- From Use Cases and Activities to Classes and Objects

Use Cases vs. Activities

- Use Cases:
 - First Overview of
 - key system functions / services
 - participating actors
 - **no** detailed behavior
 - **no** assignment of responsibilities
- Next Step:
More detailed description of scenarios for use cases
- Notational tool: activity diagrams

Use Cases: Example



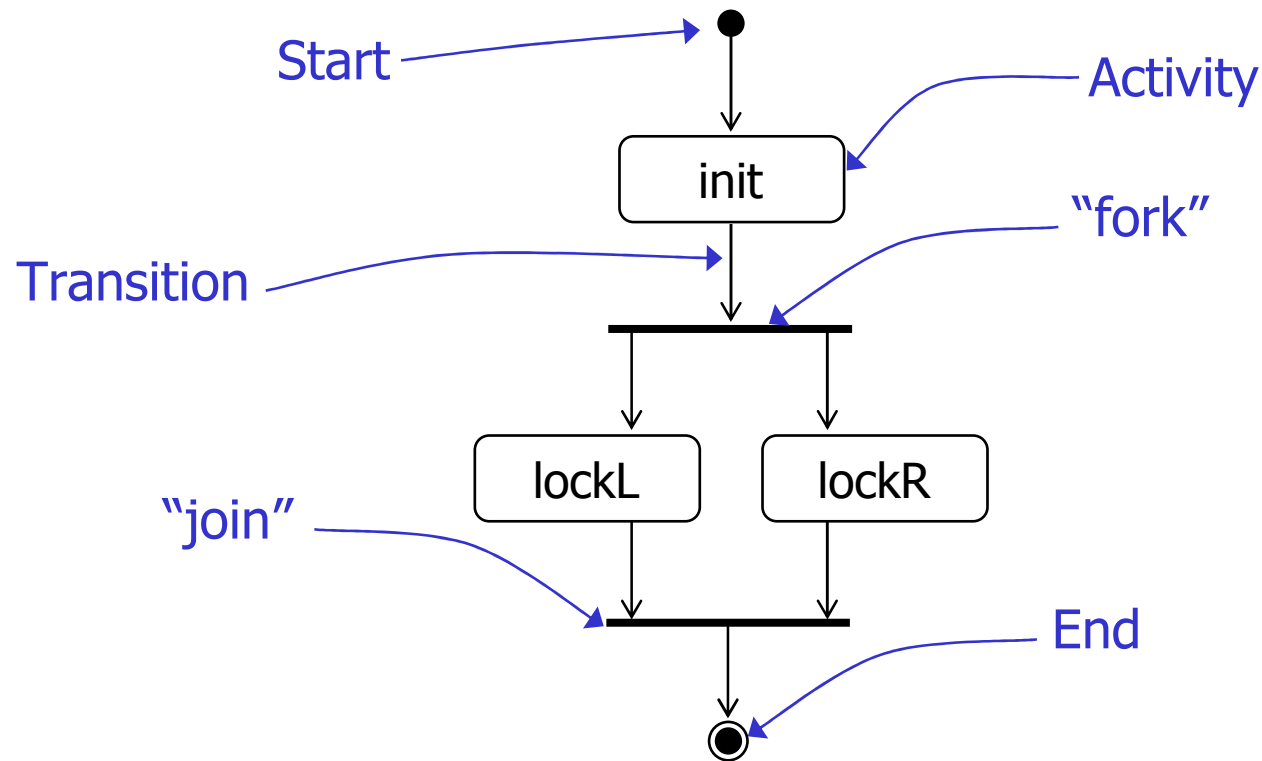
Activity Diagrams

What happens in Lock_Doors ?

1. Initialization
2. Locking of left door
3. Locking of right door

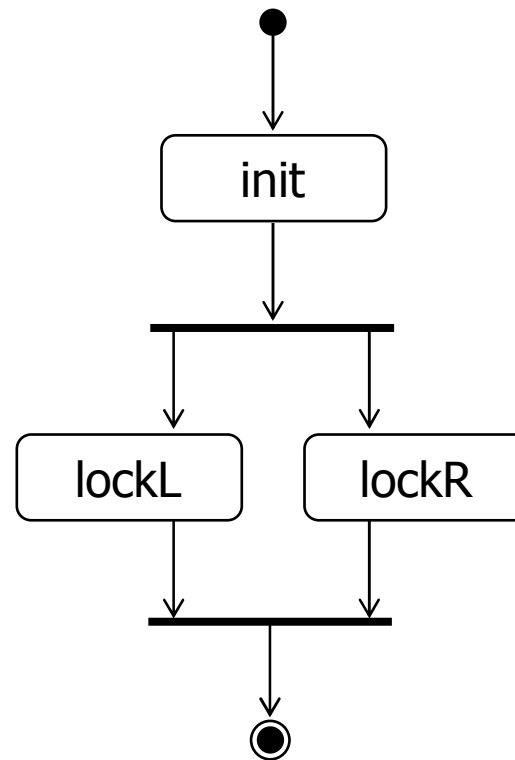
Activity diagrams: More detailed representation of use cases

Activity Diagrams

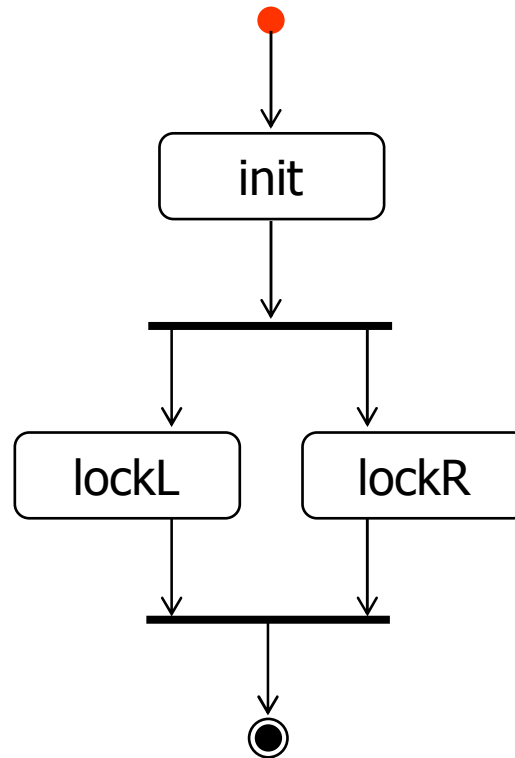


- Execution view on system activities
- Causality between activities
- Assignment of activities to components not mandatory (but possible!)

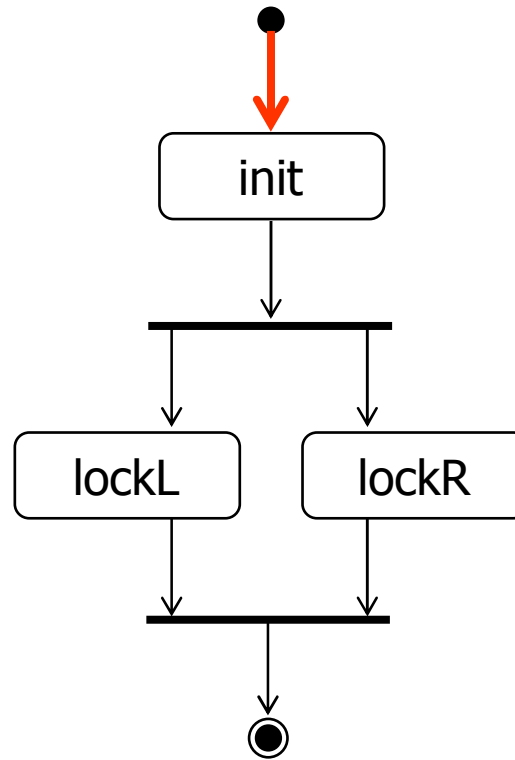
Activity Diagrams: Control Flow



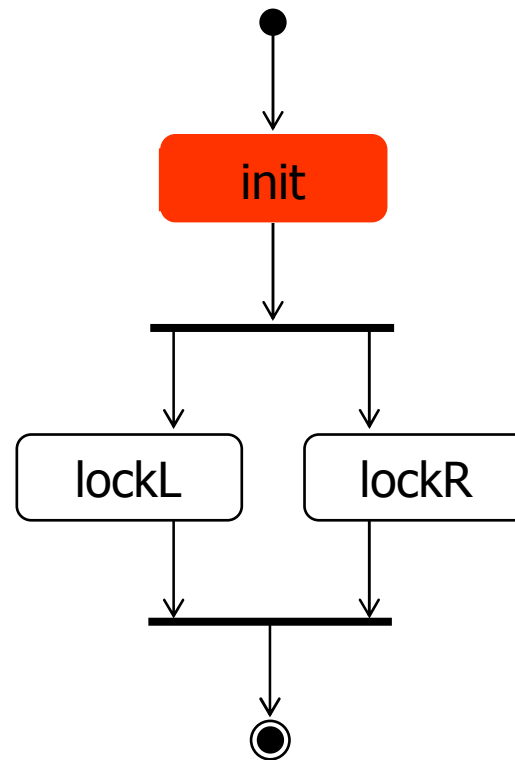
Activity Diagrams: Control Flow



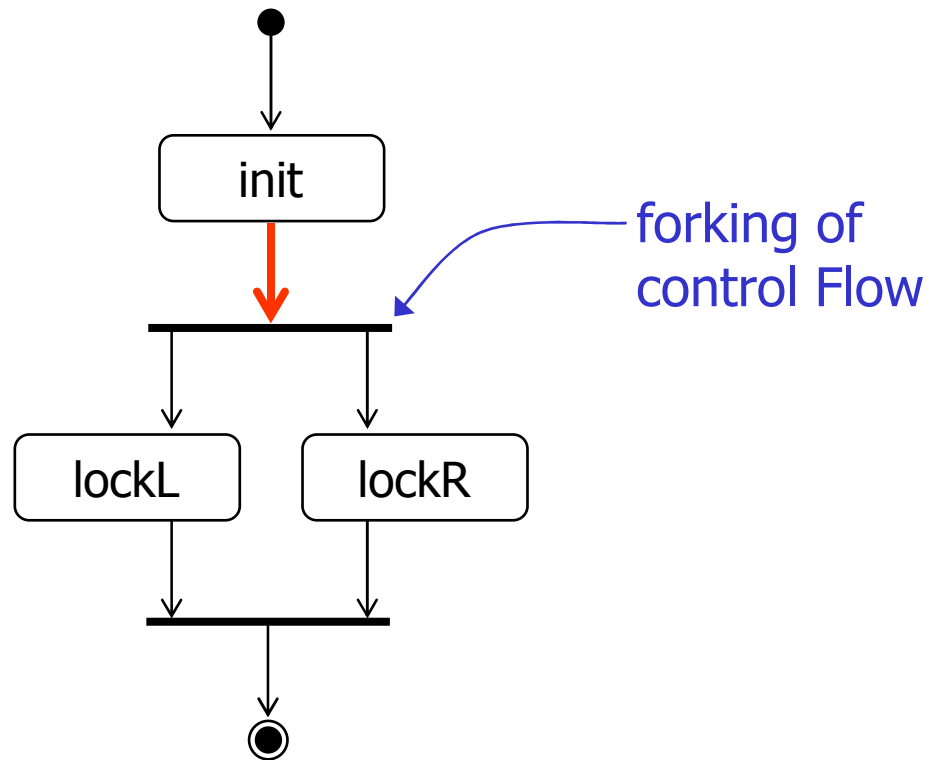
Activity Diagrams: Control Flow



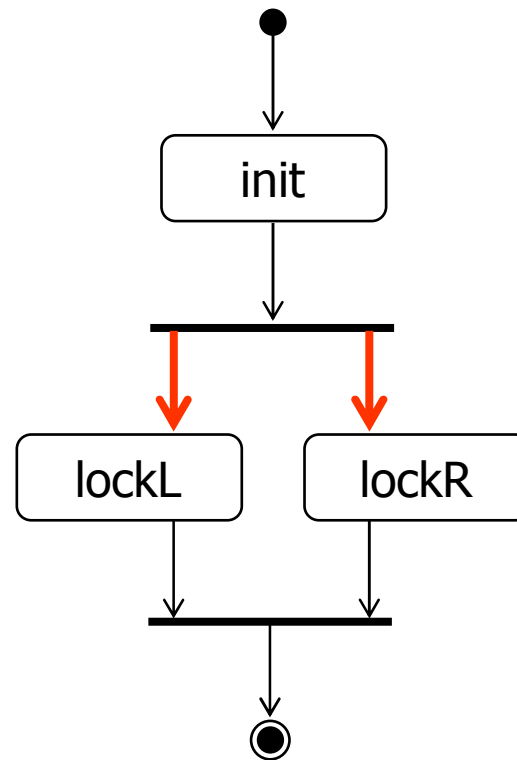
Activity Diagrams: Control Flow



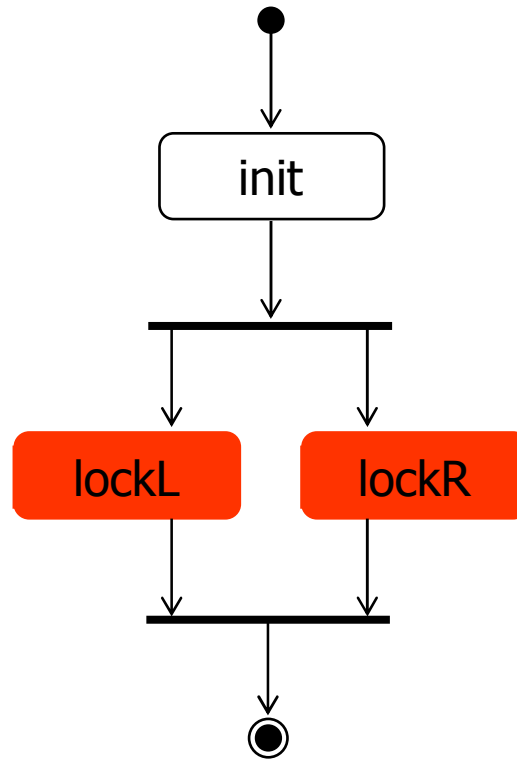
Activity Diagrams: Control Flow



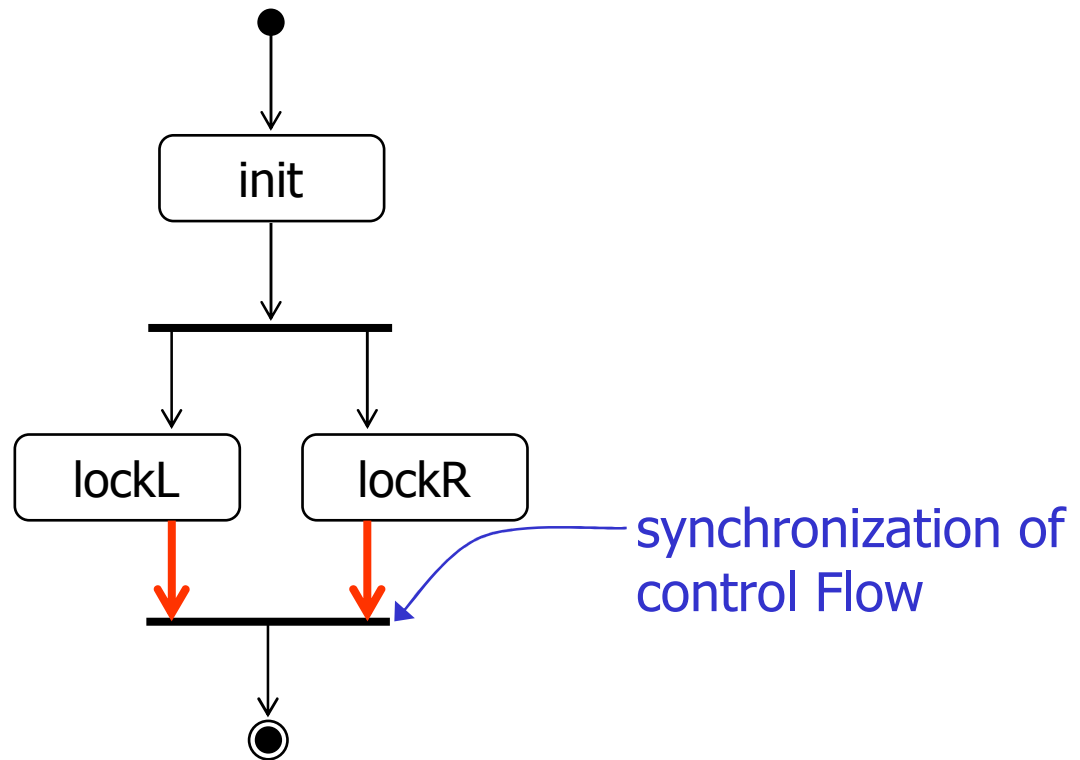
Activity Diagrams: Control Flow



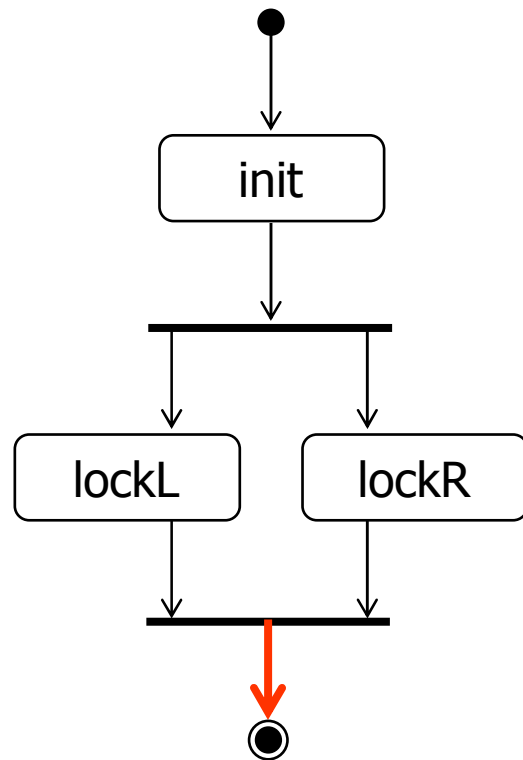
Activity Diagrams: Control Flow



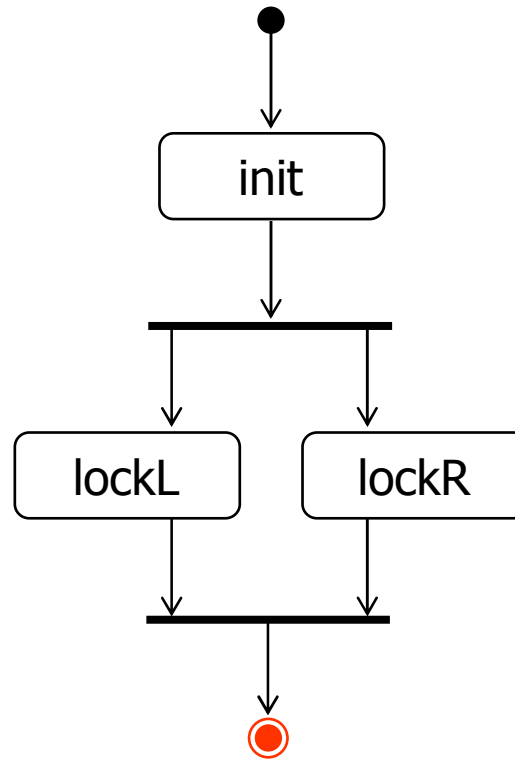
Activity Diagrams: Control Flow



Activity Diagrams: Control Flow



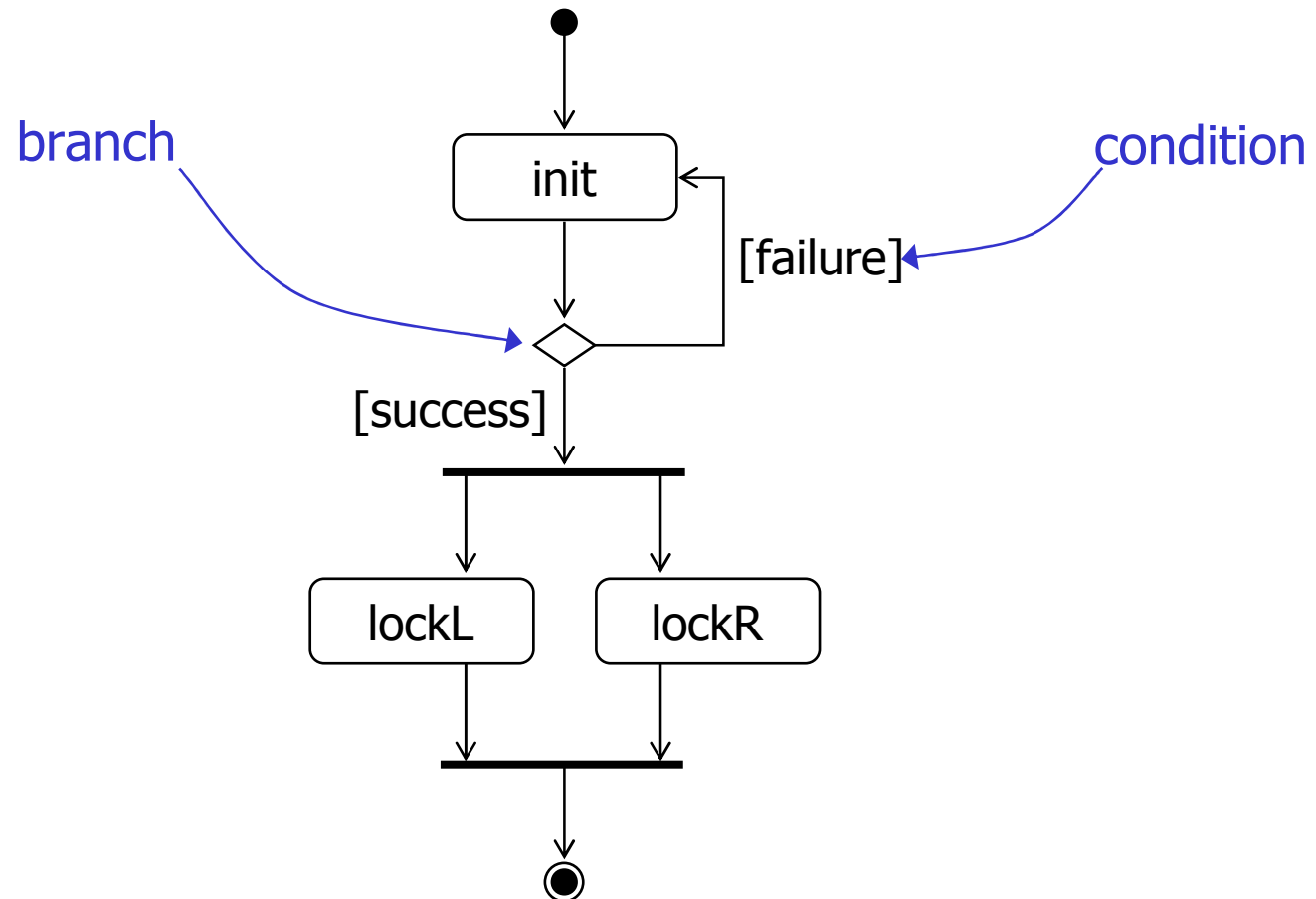
Activity Diagrams: Control Flow



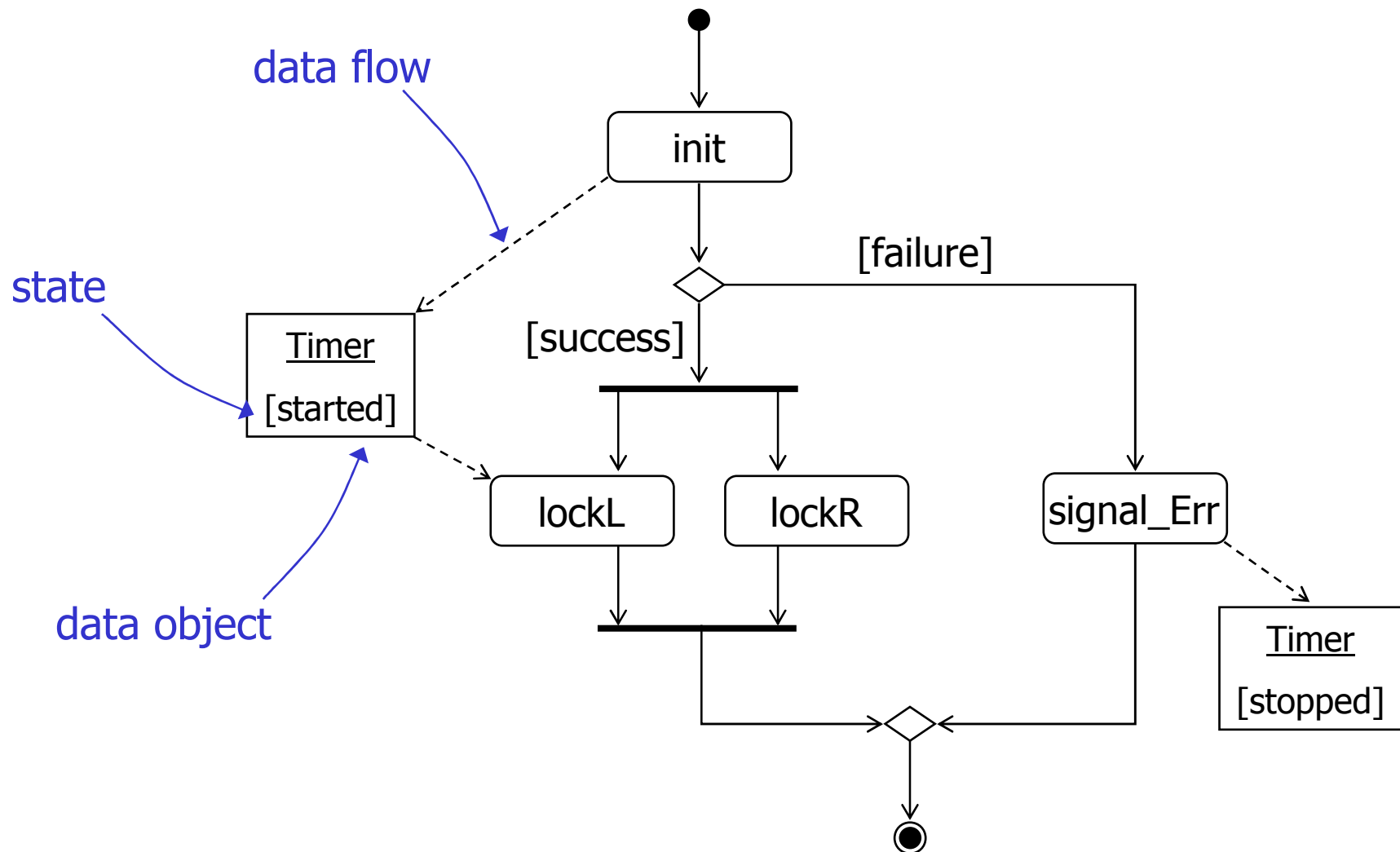
What's in an Activity?

- Coherent subsequence of actions
- “Interruptible”
(non-atomic – compare with *action*)
- Control-Flow-oriented

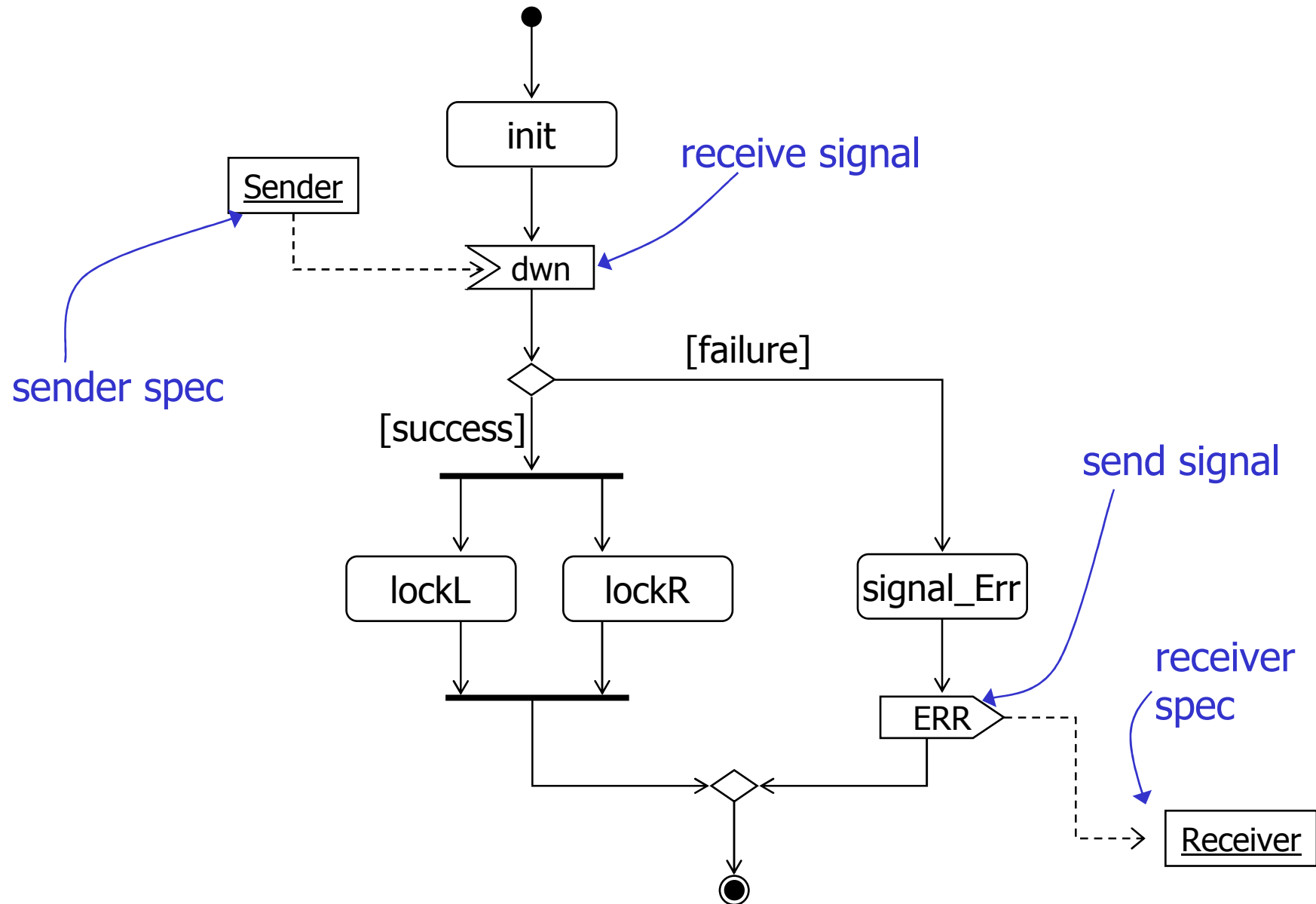
Activity Diagrams: Branches



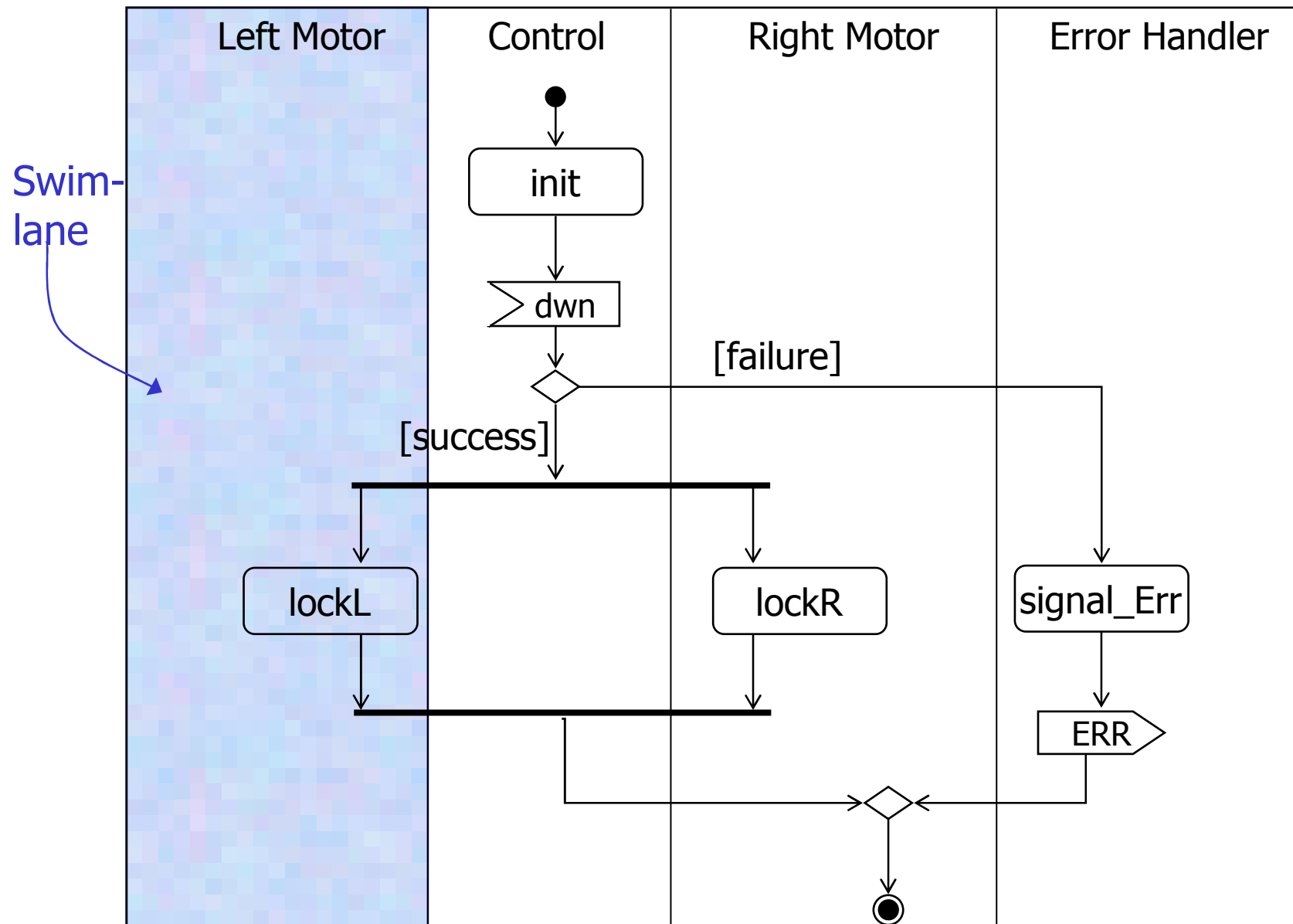
Activity Diagrams: Data Flow



Activity Diagrams: Signal Flow



Activity Diagrams: Swimlanes



Usage of Swimlanes

- Identify “organizational entities”
- Structure activity diagrams
- Initial assignment of responsibilities
- (unique) Assignment of activities to “components”

Activity Diagrams - Summary

- Goal: Identify key system traces
- Relationship to use cases
 - ⇒ activity diagrams *refine* use cases
- Representation of control-, data- and signal Flow
 - ⇒ Causality
 - ⇒ Alternatives
 - ⇒ Repetition
 - ⇒ Concurrency
- Clear notation for concurrency
- Initial assignment of responsibilities

Changes in UML 2.0

Activity diagrams

- the nodes in activity diagrams are no longer called activities, but actions instead.
- An action has local preconditions and local postconditions.
- An action can have multiple incoming flows:
 - all of the incoming flows must reach the node before the action starts.
- Introduces a special kind of signal: the time signal.

Activity diagrams

- Introduces the *flow final* node: 
 - it is a final node like the activity final node
 - indicates that a single flow within an activity is complete. Other flows within the activity may still be proceeding.
- Introduces connectors:
 - A connector allows the flow to move from one activity diagram to another

Activity diagrams - Grouping

Ways of grouping actions:

1. An action can be implemented by a subactivity diagram
2. Activity partitions - group actions by a certain set of characteristics
3. Expansion regions
 - a collection of values as input
 - is executed once for each of the elements in the input collection => behaves like a “for loop” over the input collection
 - can execute the interactions in 3 ways: in parallel, iterative, or stream.

Activity diagrams - Exceptions

- “try” block -> a protected node
- “catch” block -> handler body node
- If an exception occurs, the set of handlers is examined for a possible match.
 - If a match is found, the handler body is invoked.
 - Otherwise, the exception is propagated to the enclosing protected node if one exists
- Allows nesting activities in a protected node

Overview

- Use Cases
- Activities
- From Use Cases and Activities to Classes and Objects



Finding Objects

Nouns in the problem description

⇒ candidates for objects

Example: Elevator Controller

Your job is to develop the controller for an elevator, consisting of six cabins. Each cabin has a user console by which users select their destination. On each floor there is a call-button for each of the directions “up” and “down”. On each floor and for every cabin, as well as inside of each cabin there is a position indicator ...

Separate relevant/irrelevant objects

Finding Objects

- Play through use cases
 - who/what participates?
 - what data/information is needed?
 - does the execution of the use case need a dedicated controller (transactions)?
- What are the system's active/passive entities?
- What are the system's physical components?
 - particularly interesting for embedded systems
 - provides interface view on external components
- What are the fundamental concepts of the application domain?
 - Examples: day plan (production), call (telephony), ...

Finding Objects

- What belongs to the user interface
 - Starting point for finding domain objects
- Organization/Structuring/Saving of groups of objects required?
 - Container objects
 - Persistent storage

Finding Classes

- Classes represent common “features” of objects
- Concrete, meaningful names for sets of objects
 - ⇒ candidates for classes
- Categories for classes:
 - Physical Entities
 - People
 - Locations
 - Organizations
 - Interfaces
 - Association manager
 - Container
 - Controllers for use cases

Finding Associations

- How are classes/objects related?
- What classes/objects exchange information?
- How durable are the relationships between classes/objects?
- What roles do participating classes/objects play?
- Does the association require attributes/operations?
⇒ Transform association into class

Finding Associations

- Does class A control the creation/destruction of class B's objects?
 - Aggregation
 - Composition
- What cardinalities does the association have?
- Check:
What communication paths do you find while playing through the use cases?

Finding Attributes

- What data does an object need to fulfill its task?
- What external information does an object need?
- What characterizes the object?
- How complex is an attribute?
 - ⇒ transform into an object if too complex
- Does the attribute belong to the class or to its instances?
 - ⇒ class versus instance attributes

Finding Operations/Methods

- What are the use cases in which the object partakes?
- What roles do the objects play in the respective use cases?
- What messages does the object receive?
- What parameters/results does the operation need/provide?
- Concentrate on externally observable behavior!
- What are the pre-/postconditions of the operation?