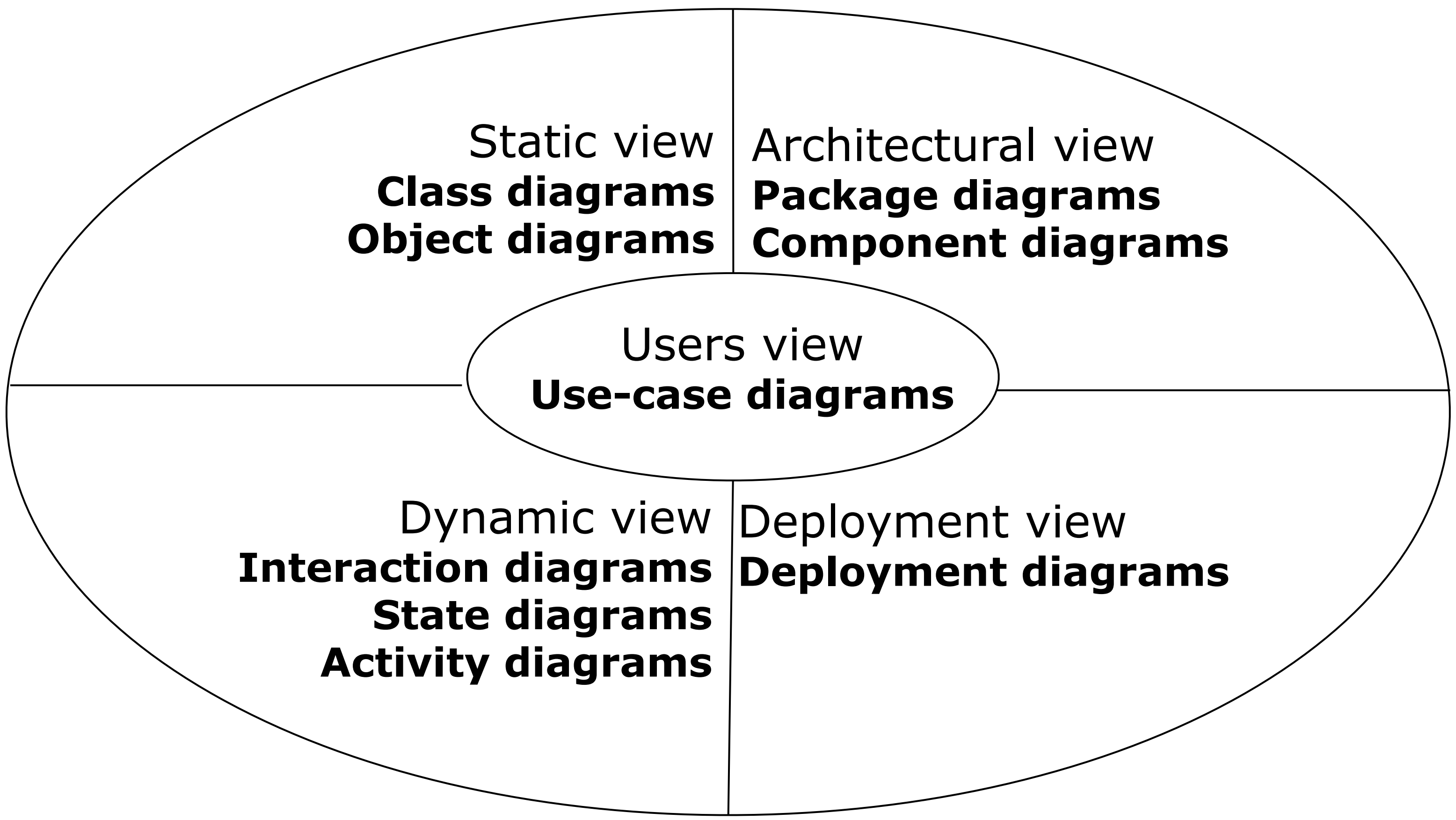
Modelling dynamic behaviour

- Activity diagrams
- State diagrams
- Interaction diagrams



Main Activities of Software Development

Requirements Gathering

Define requirement specification

Analysis

Define the conceptual model

Design

Design the solution / software plan

Implementation

Code the system based on the design

Integration and Test

Prove that the system meets the requirements

Deployment

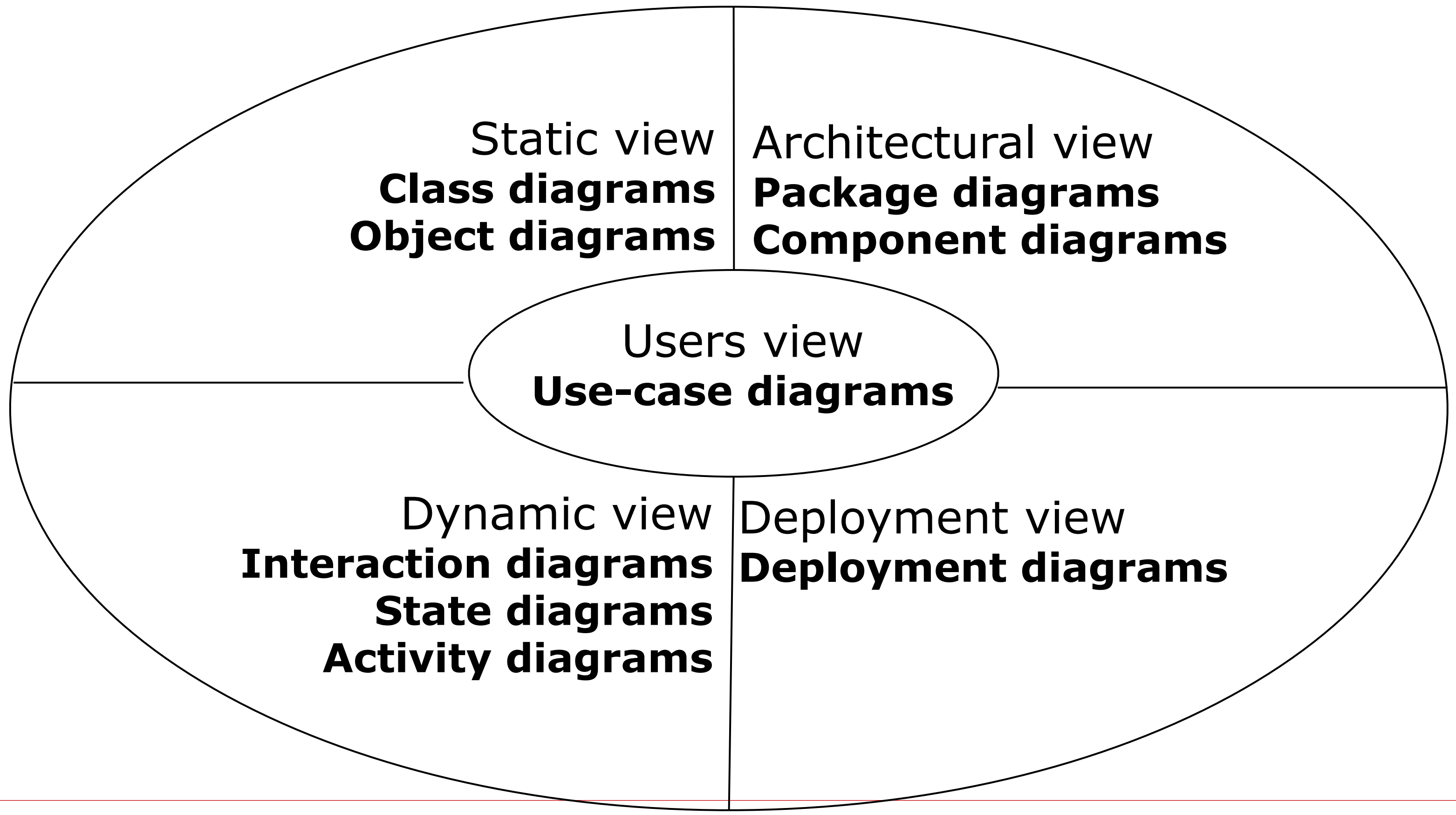
Installation and training

Maintenance

Post-install review
Support docs
Active support

Modelling dynamic behaviour

- Describing what happens during the execution of the system
  - The behaviour of the objects
- Dynamic behaviour modelling allows to complete the information in the static diagram
  - How the elements in the static diagrams
    - provide the functionality of the system
    - change their states
    - communicate with each other
    - cooperate to perform their tasks



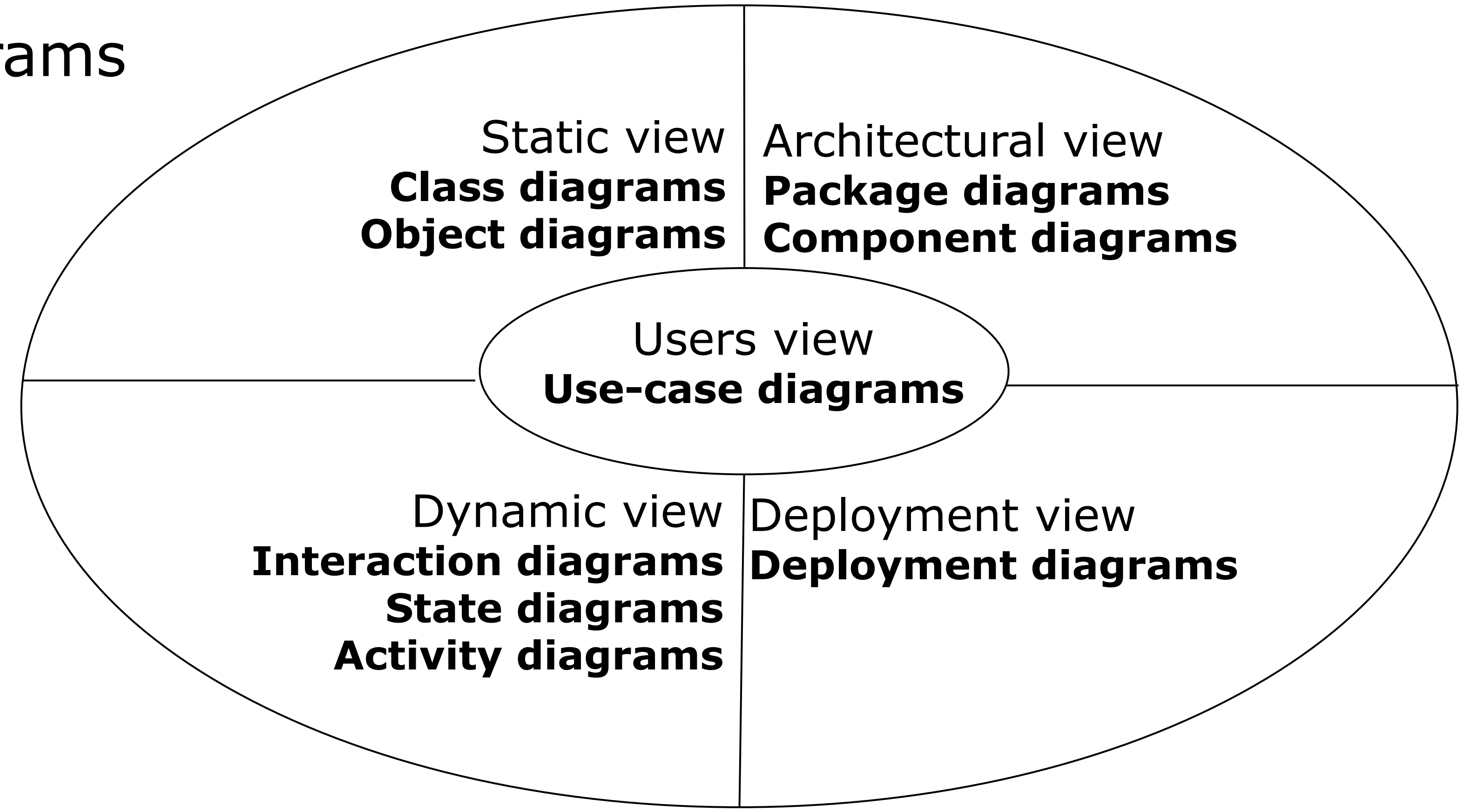
Modelling dynamic behaviour

- Difficulties
  - Modelling of the dynamic behaviours of a complex system is always difficult
    - Too many features
    - Too many paths
  - The collaborations between objects are complicated
  - It is not easy to allocate and carry out responsibilities

- Suggestions
  - Focusing on the communication of one dynamic aspect of the system to better master the complexity
  - Modelling only the essential elements
  - Providing a detail suitable for each level of abstraction

Modelling dynamic behaviour

- Diagrams
  - Activity diagrams
    - High level dynamic behaviour
    - Performing objectives of the system
  - State diagrams
    - Internal behaviour of the system
  - Interaction diagrams
    - Communication between objects
      - Sequence diagrams
      - Collaboration diagrams



Activity diagrams

- Allowing to determine the dynamic behaviour of the system from **one or several use-cases**
- Being useful to model the flows of treatment in the system
- Modelling the control flow and also the data stream

- An activity diagram includes
  - the activities carried out by the system and the actors
  - the order in which these activities are carried out
  - the possible dependencies between activities.

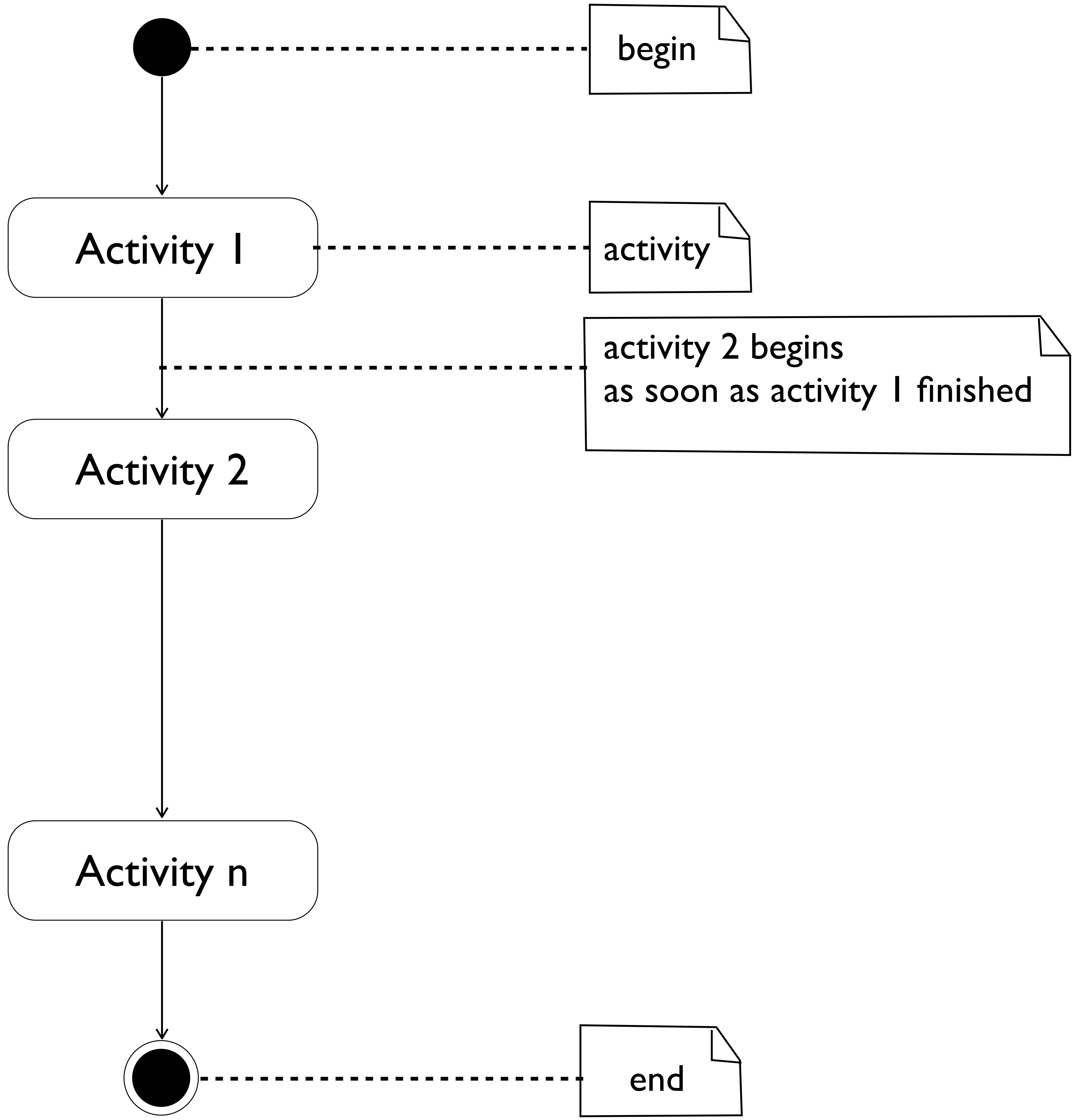
Activity diagrams

- An activity corresponds to a high-level task in the system
- Distinction between the activities and operations in the static structure
  - Activities are carried out by the system or the actors
  - Operations are related to classes
  - In general, activities do not correspond to operations

- Activity diagrams are generally built before (design) class diagrams
  - Activity diagrams are used to determine which operations to add to class diagrams

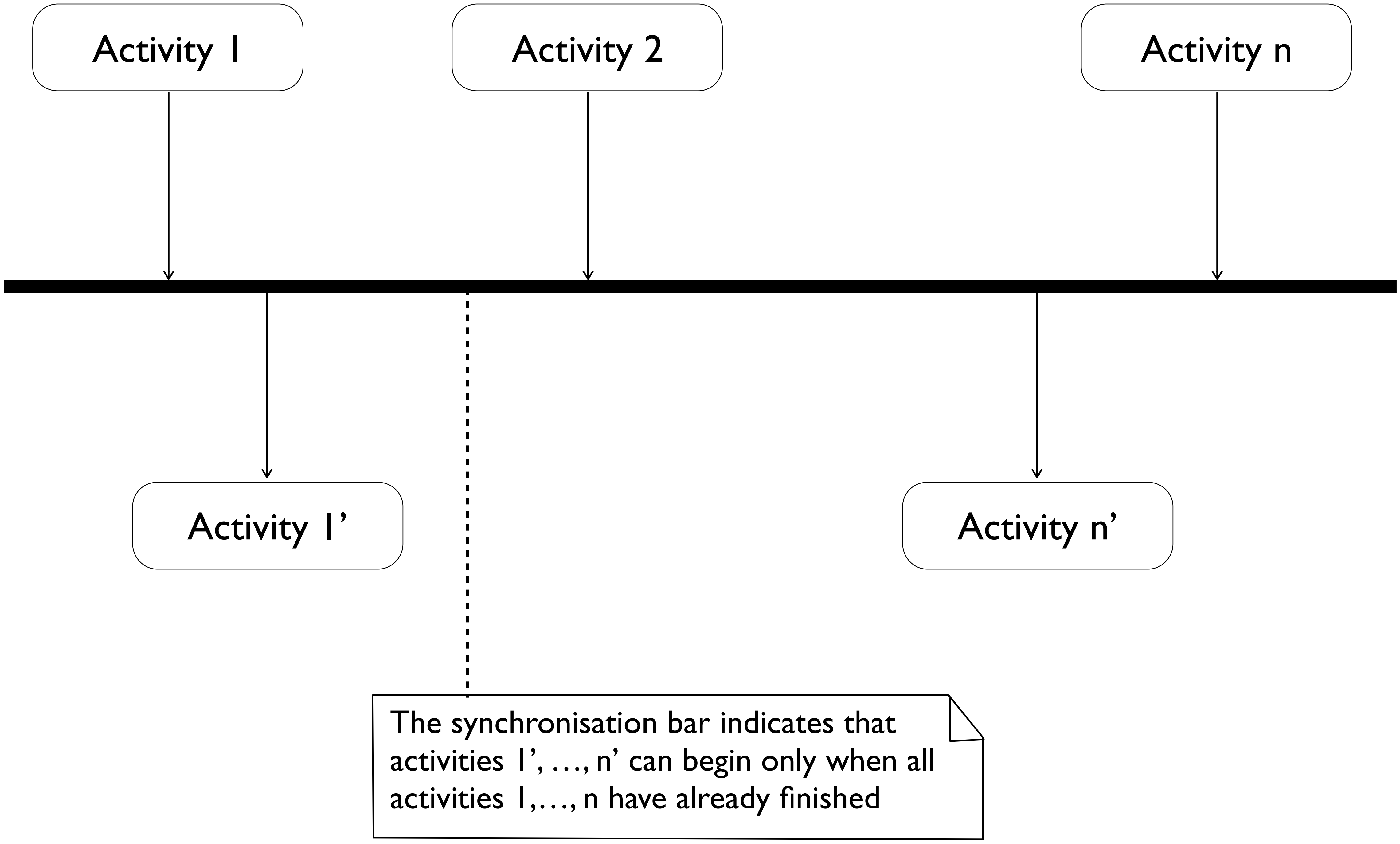
Activity diagrams

- Notation



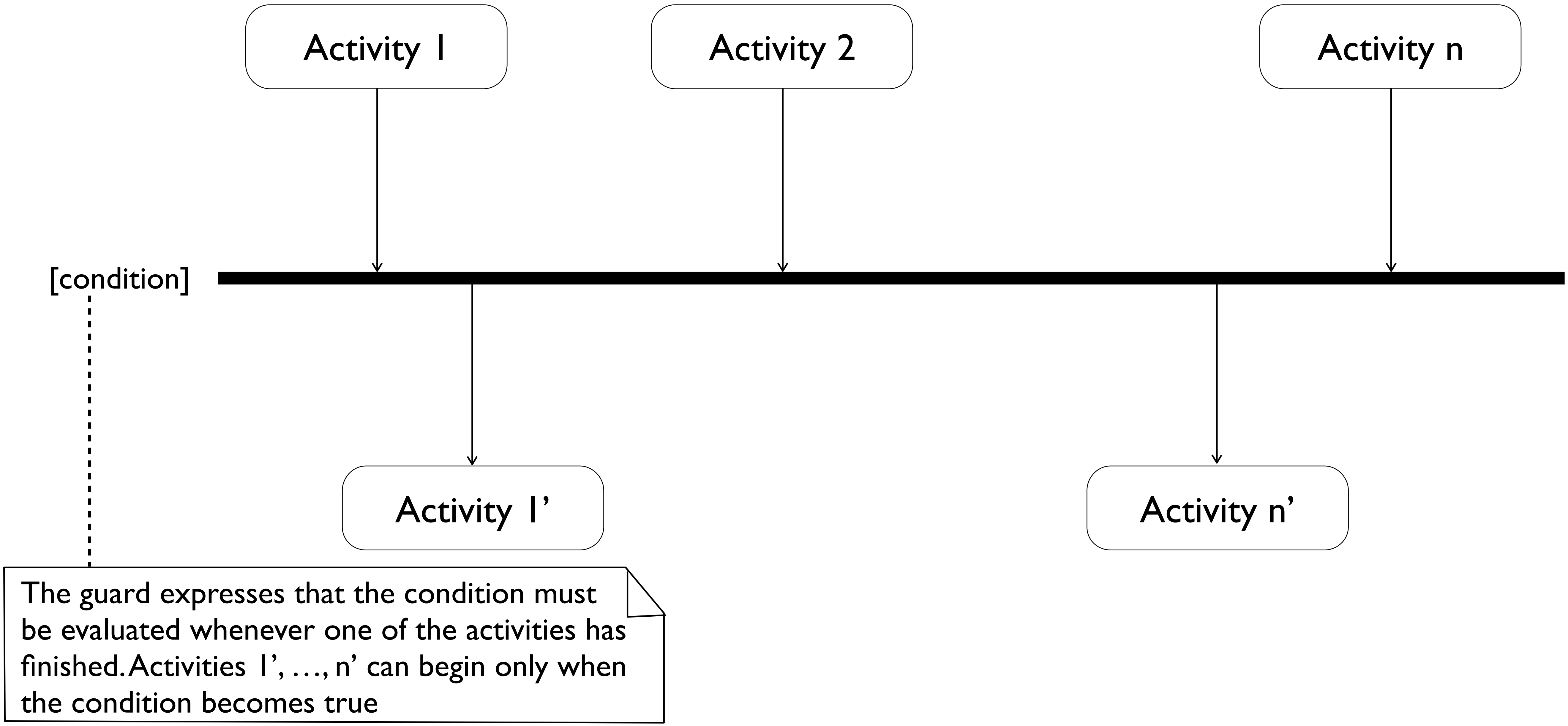
Activity diagrams

- Synchronisation of activities



Activity diagram

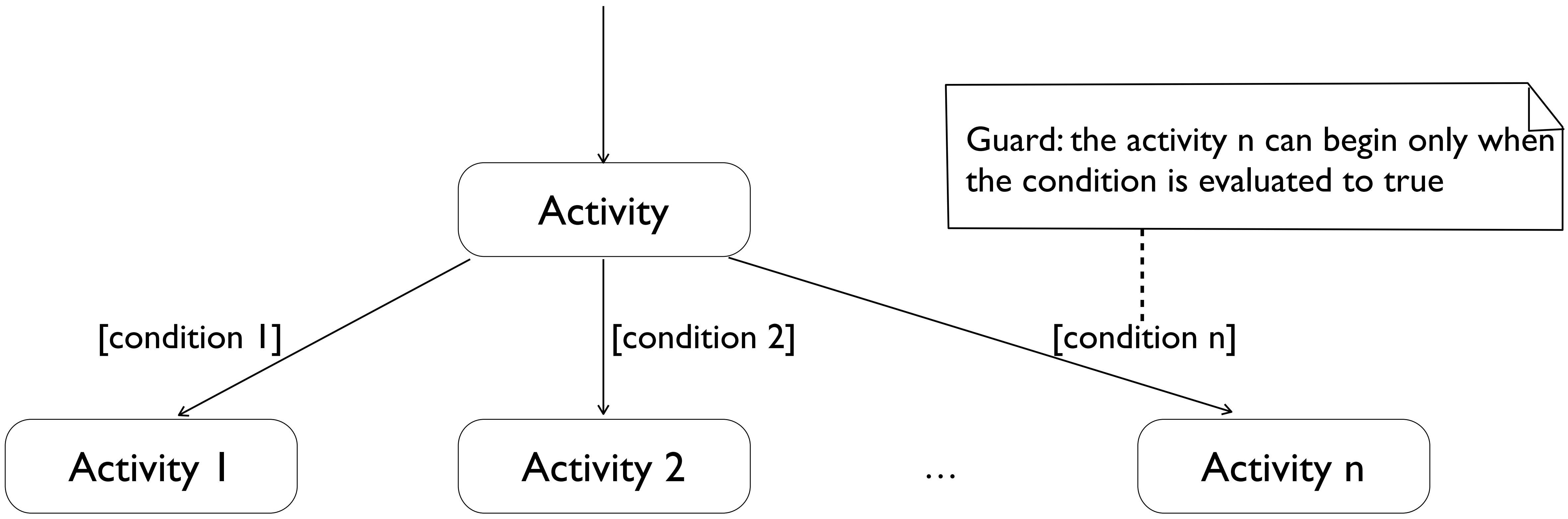
- Synchronisation guarded by a condition



- The absence of guard can be considered as a special guard. This one becomes true when all the activities at the entrance to the synchronisation bar have finished

Activity diagrams

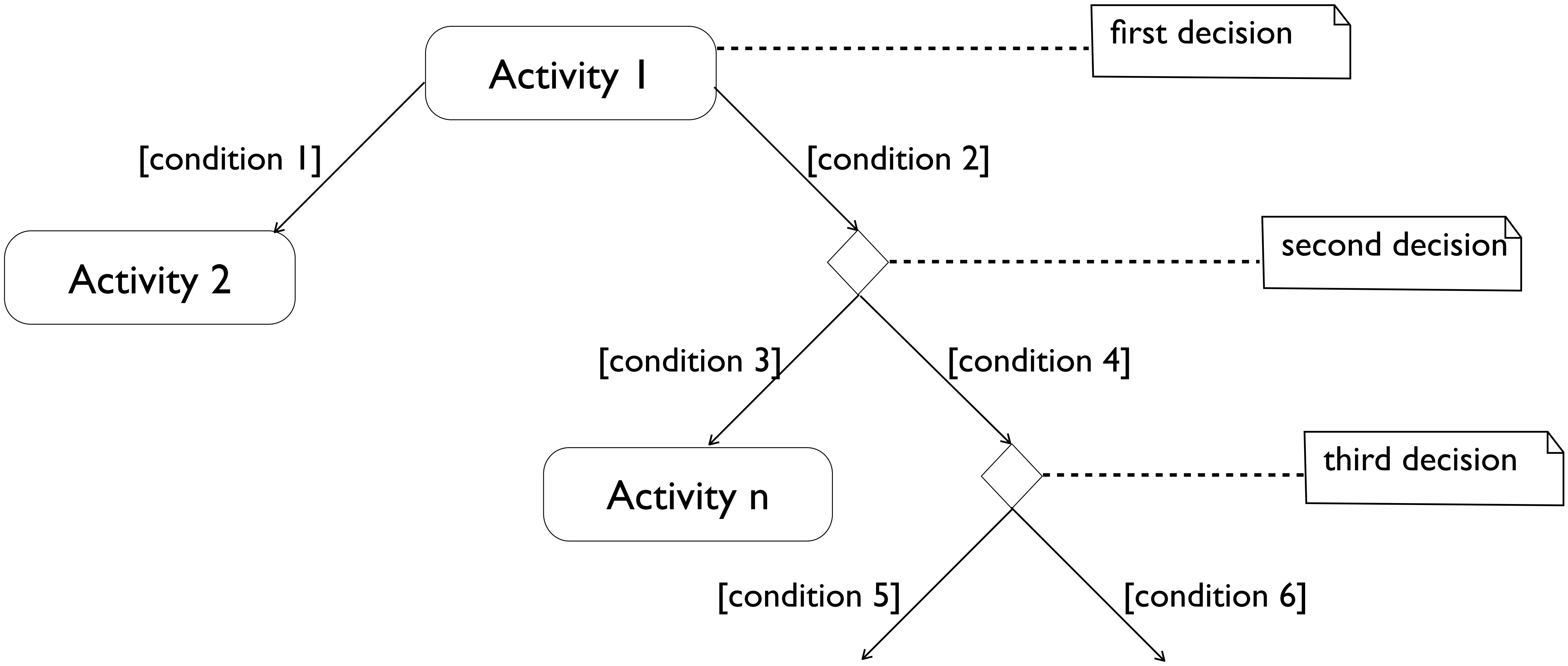
- Decision



- The transition guards coming out of the same activity should be mutually exclusive

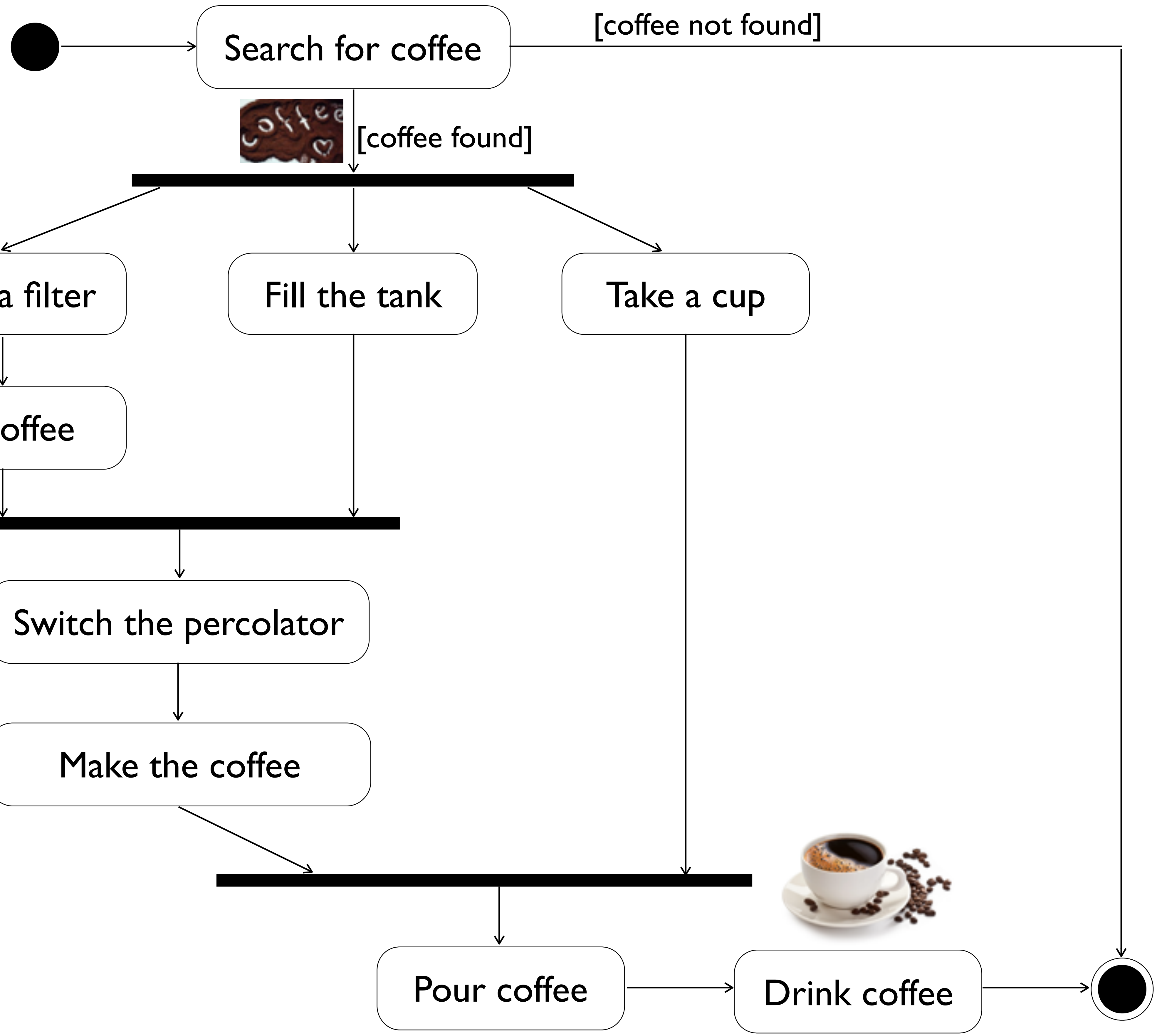
Activity diagrams

- Multiple decisions



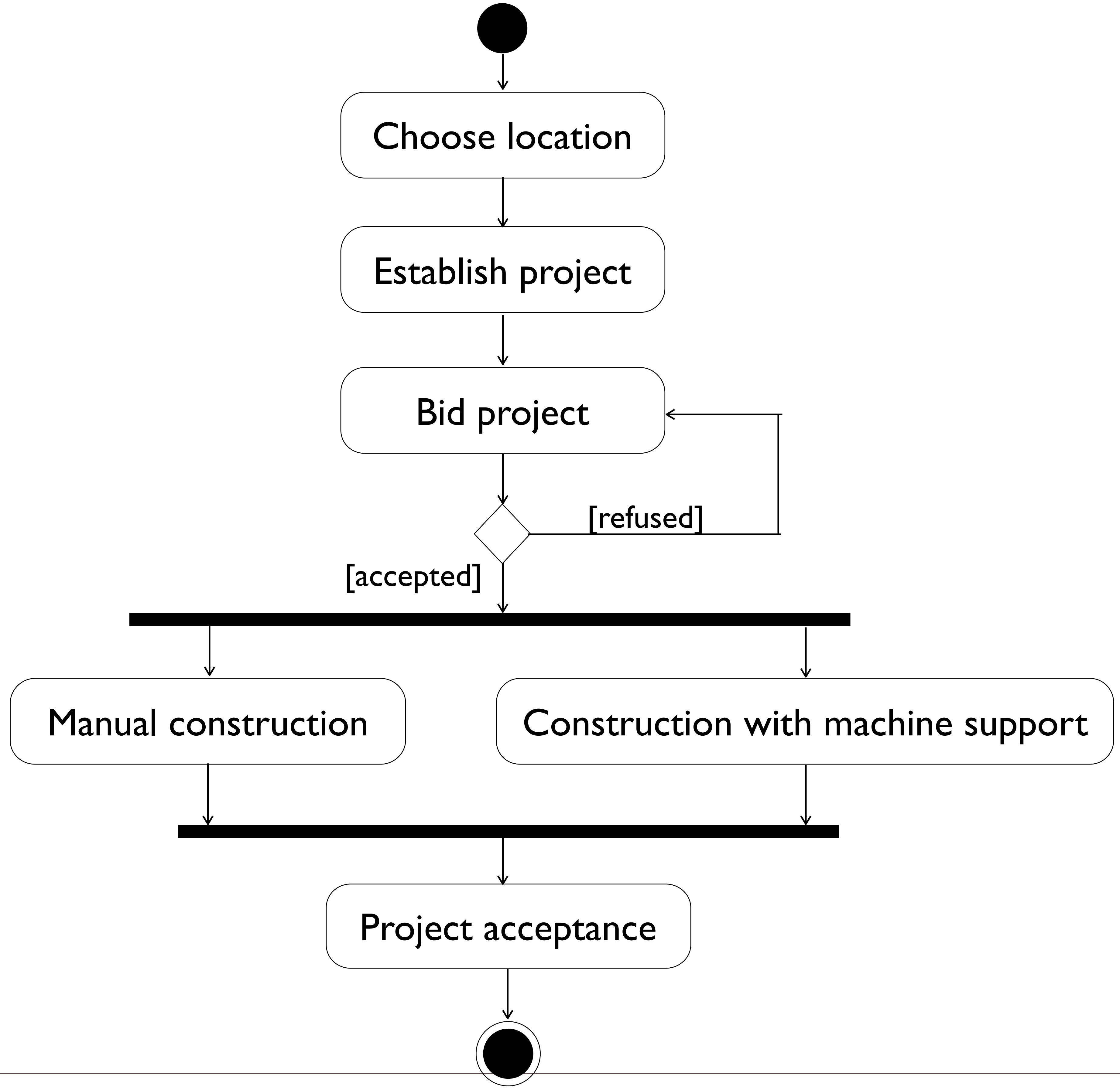
Activity diagrams

- Example



Activity diagrams

- Example

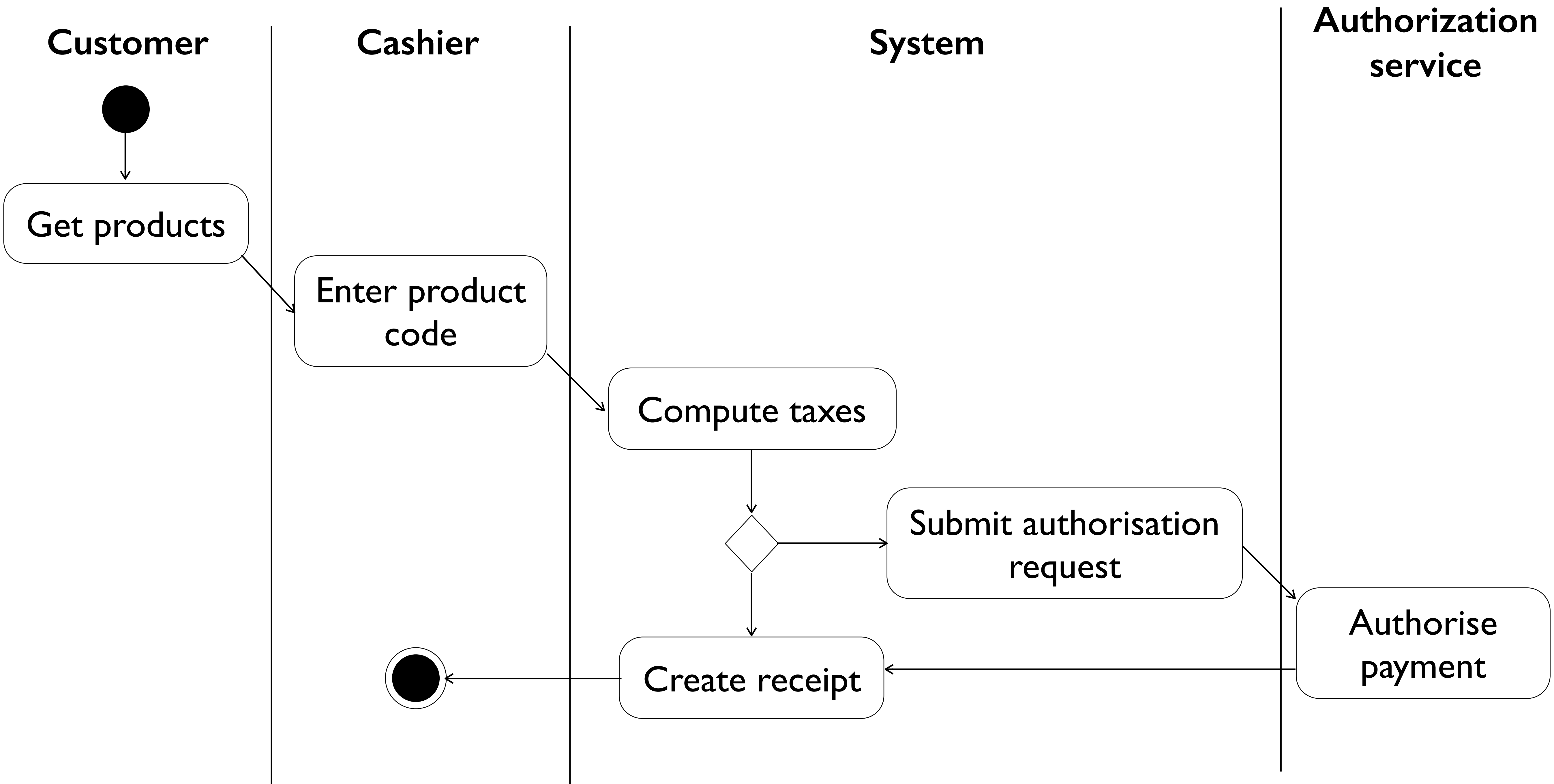


Activity diagram

- **Swimlane** helps to clarify on the activity diagrams the actors or components of the system that perform different activities
- Example: Activity diagram for "**sale**" use-case

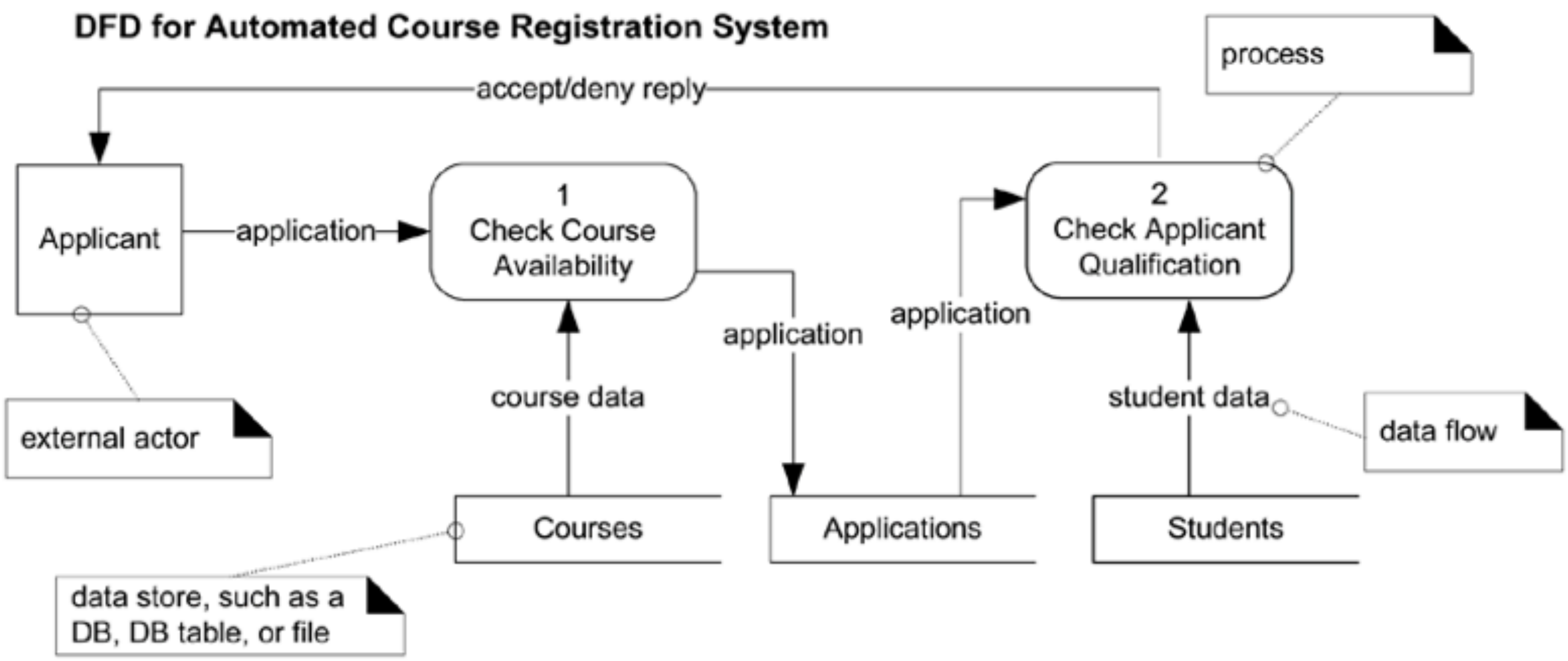
Activity diagram

- Example: Activity diagram for "**purchase products**" use-case



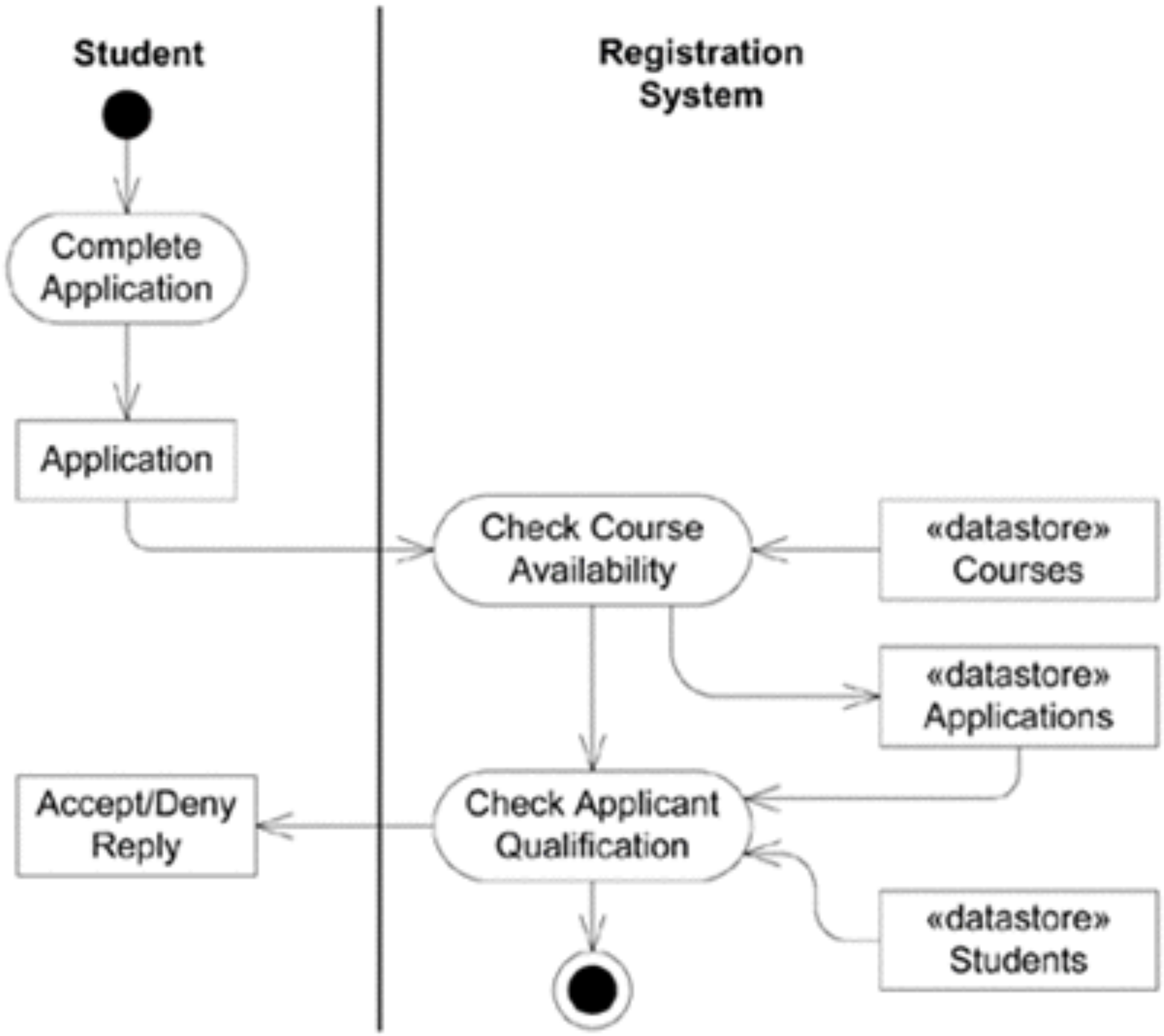
Activity diagram

- Activity Diagrams v.s. Data Flow Diagams
  - Data Flow Diagram (DFD)



Activity diagrams

- Activity Diagrams v.s. Data Flow Diagams
  - Activity Diagram



State Diagrams

- State diagrams
  - are **finite state automata**
  - allow to model the dynamic behaviour of a collaboration or a class
  - focus on the behaviour of objects, ordered by events
  - are especially used for modelling reactive systems

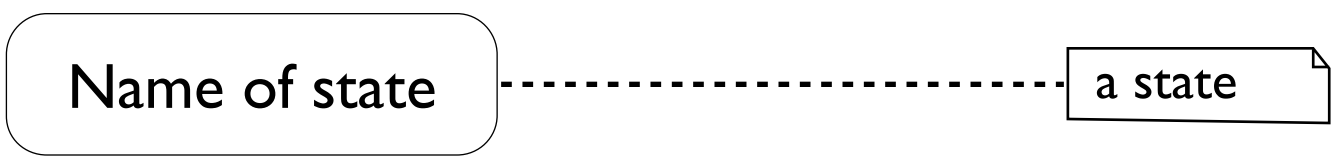
State diagrams

- State diagrams describe the behaviour of a system, part of a system or an object in the system
  - Each system or object has a **state** at a given time
  - In a given state, the system behaves in a specific manner to respond to the coming events
  - The **events** trigger state changes
- Specifically, a state diagram models the changes of states of a system/object in response to events
- A state diagram includes
  - **State**: state of a system/object at a given time
  - **Transition**: allows to switch a state to another
  - **Event**: activates the transition

State diagrams

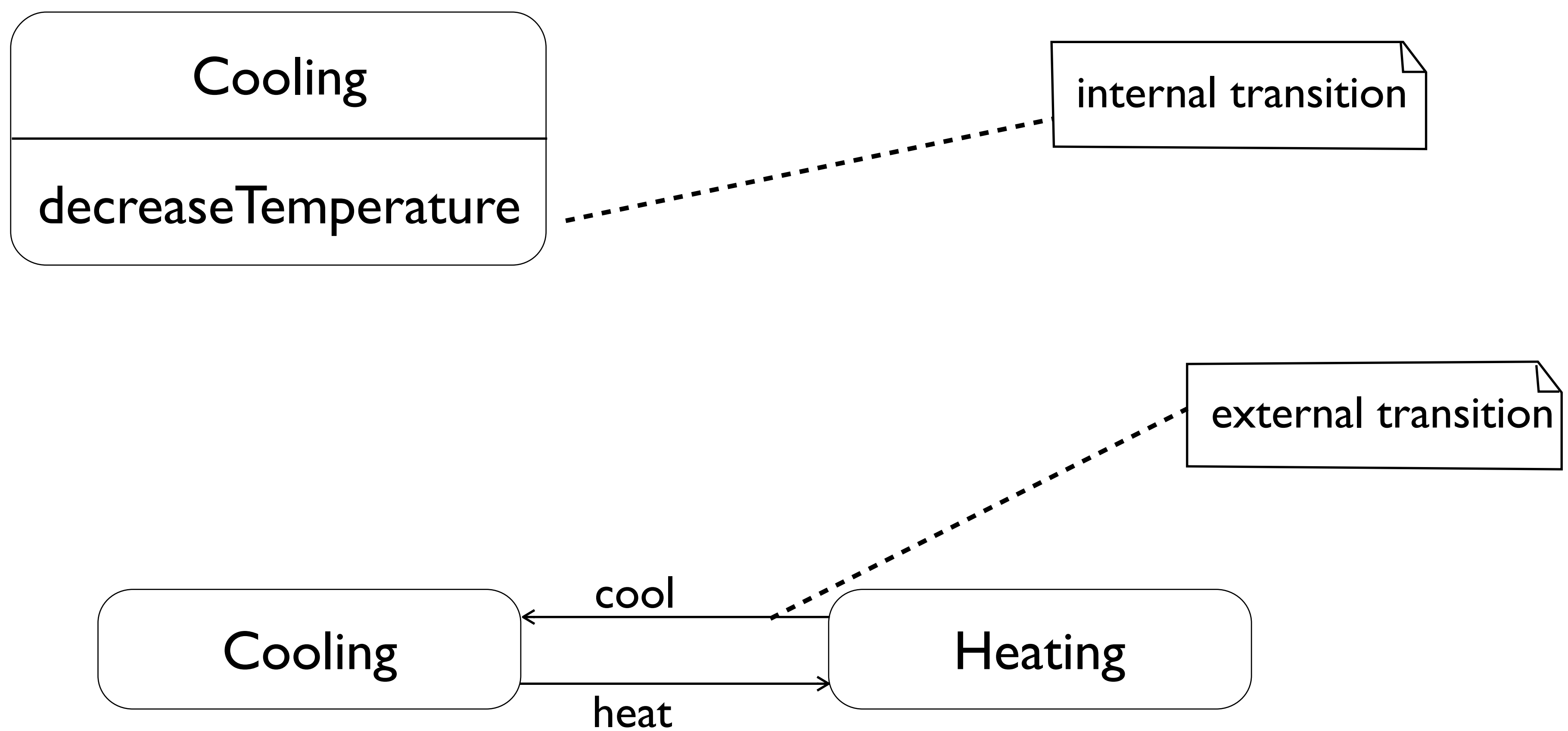
- State
  - Represents a situation of a system/object at an instance
  - System/object remains in a state for a while. Meanwhile, it can
    - perform certain **activities**
    - wait until an **event** occurs

  - Notation



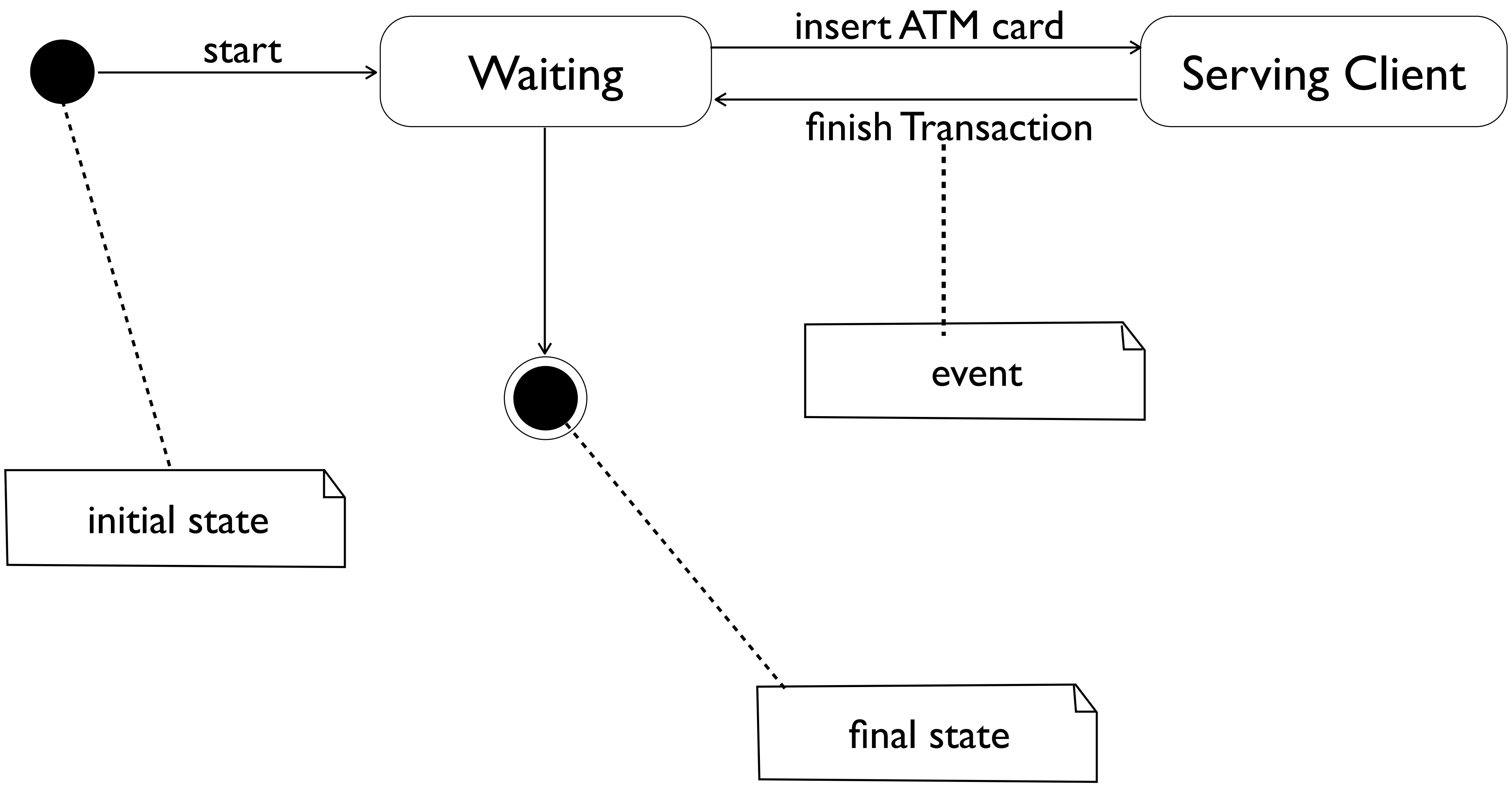
State diagrams

- Transitions
  - Transitions are related to actions that can be performed by the system/object associated with the diagram
  - Two transition types
    - **Internal transitions** to a state that react to an event without changing the current state of the system or object
    - The **transitions between states** or **external transitions**, which express a change in state
  - Example: States of an air conditioner



State diagrams

- Example: Describing the states of an ATM machine



State diagrams

- Event
  - Events of a transition have the following general form

Event [guard] / action

- Event: the event name leading to the transition
- Guard: the condition must be satisfied in order to overcome the transition
- Action: the operation performed when crossing the transition

- Remark: some of these elements may be omitted

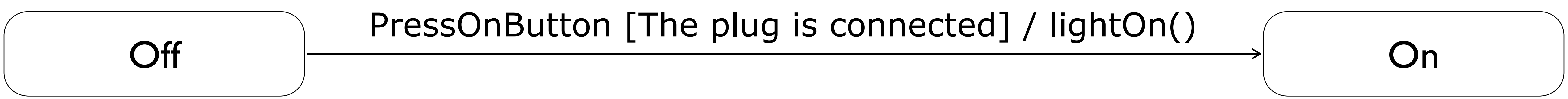
State diagrams

- Event
  - Example: states of a heater

InActive —PressButton [The plug is connected] / heat()→ Active

State diagrams

- Example: State of a lightbulb



State diagrams

- Three **special events** associated with state transition
  - **entry**: allows to specify an action to be performed when entering the state
  - **exit**: allows to specify an action to be performed when going out of a state
  - **do**: allows to specify an action to be performed while the system/ object is in the state

  - Example

TypingPassword

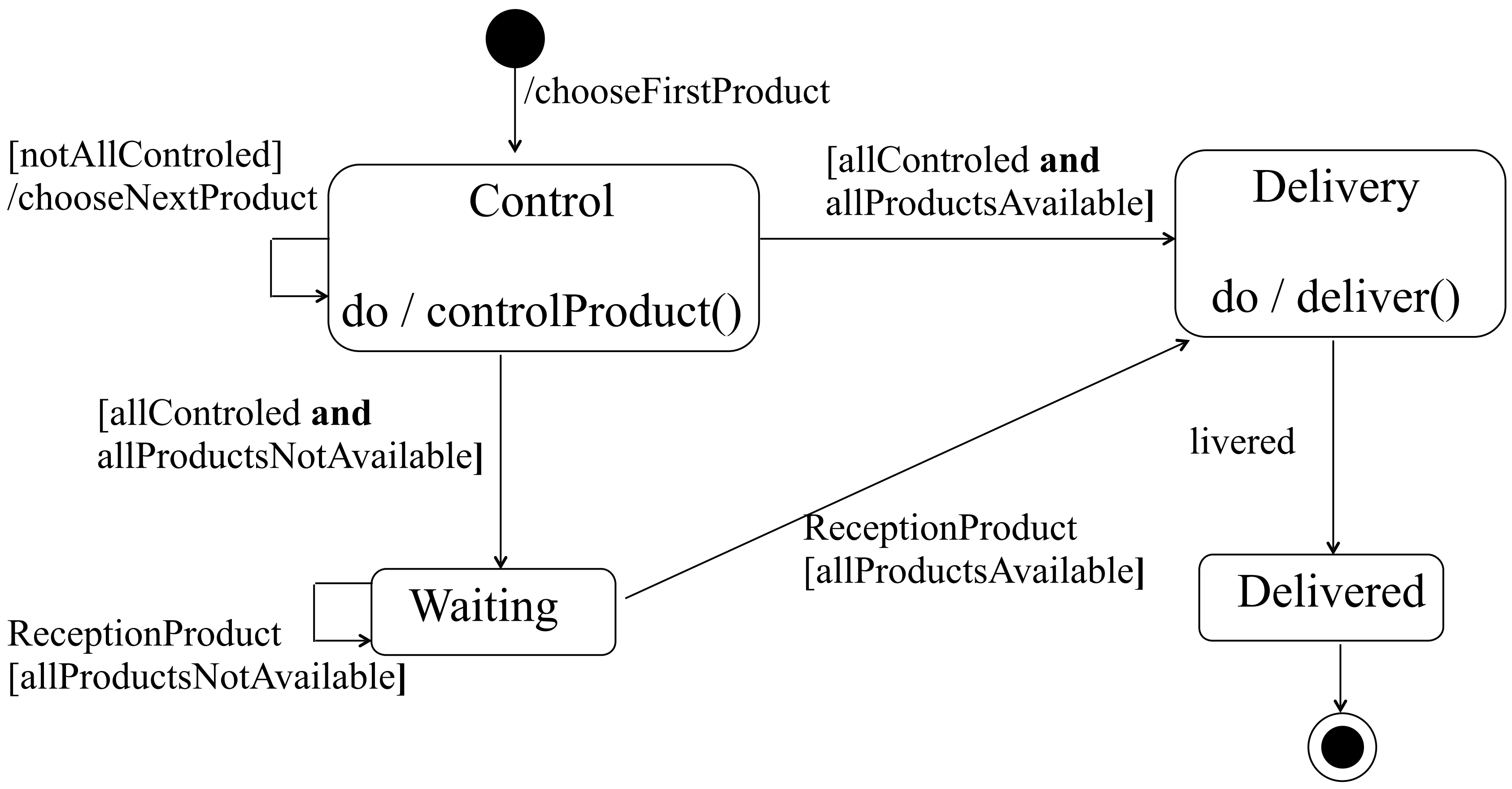
entry / setEchoInvisible
exit / setEchoNormal
do / handleCharacter

ReceivingPhoneCall

entry / pickup
exit / disconnect

State diagrams

- Example
  - Describe the behaviour an “Order”



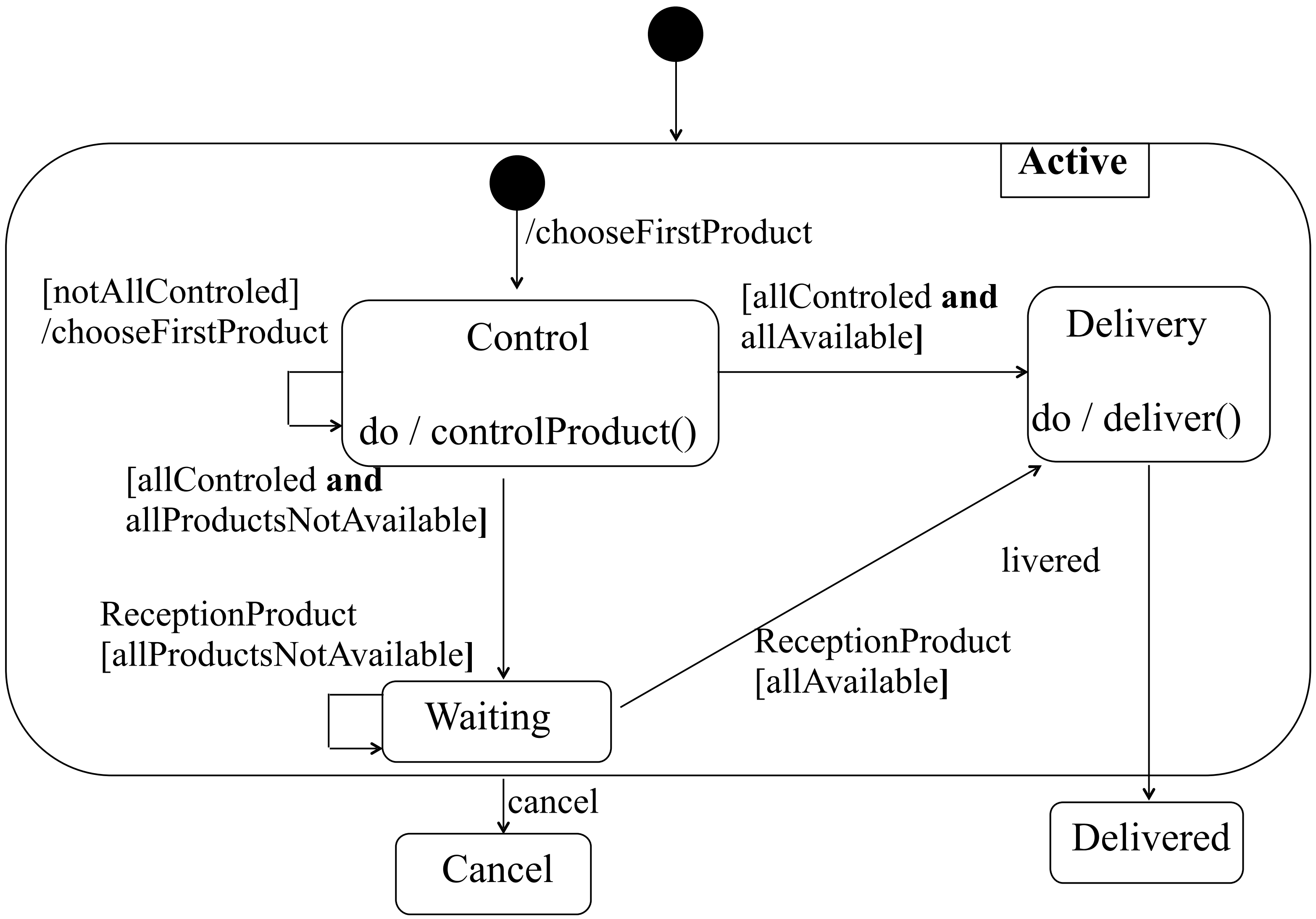
State diagrams

- Composite state
  - Several states and the transitions between these states can be combined into a composite state

  - Principles
    - The composite state has an initial state
    - The **transition to the composite state** is immediately followed by its initial state
    - The **transition from the composite state** may be originated from any of its belonging states

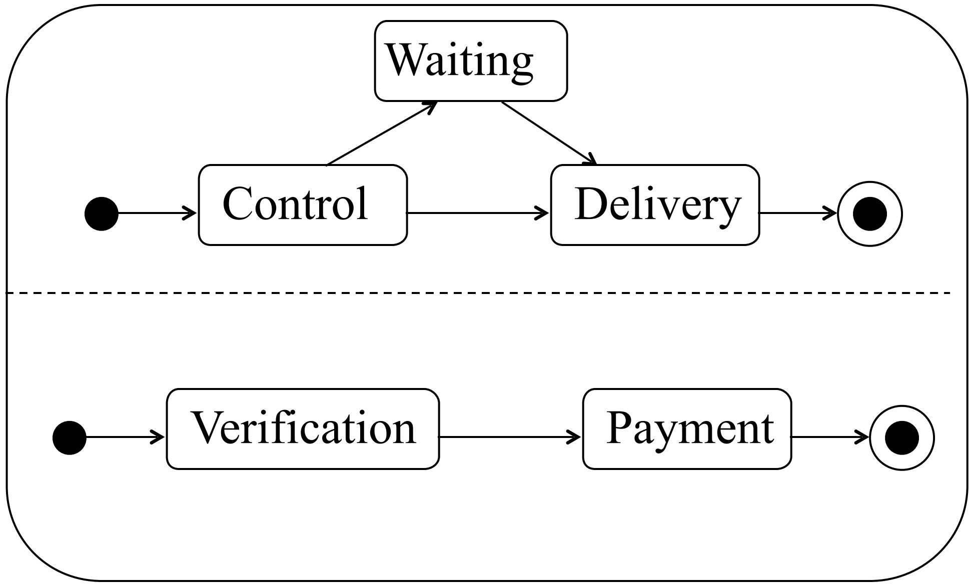
State diagrams

- Example of composite state



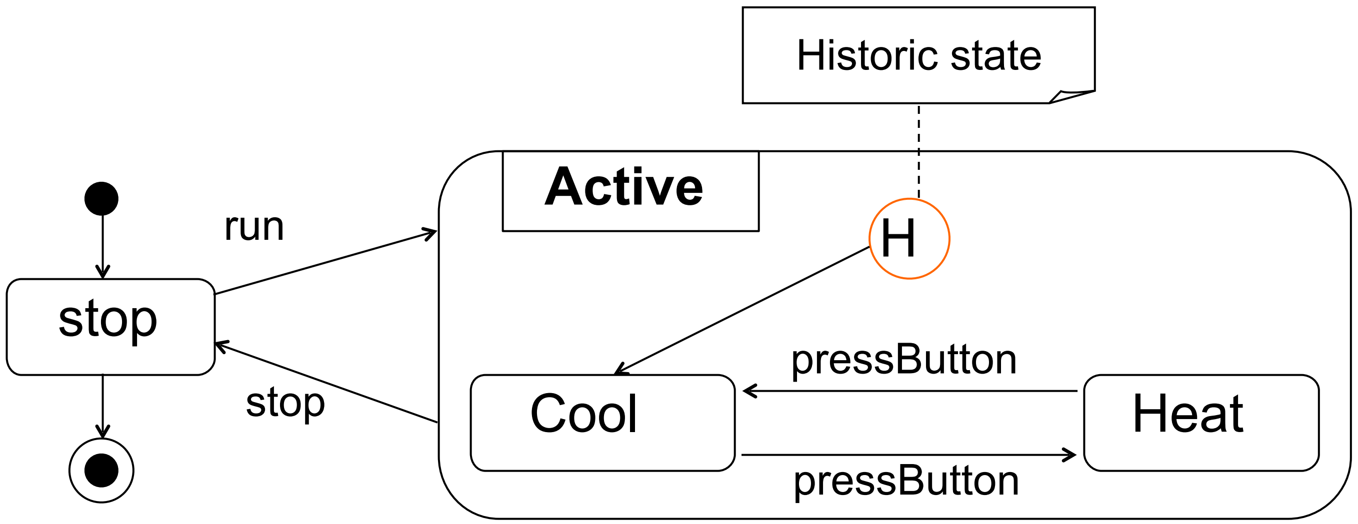
State diagrams

- Parallelism
  - Defining concurrent state within a composite state
  - Several states may exist simultaneously within a composite state
  - Example
    - Simultaneous processing of an order and its payment



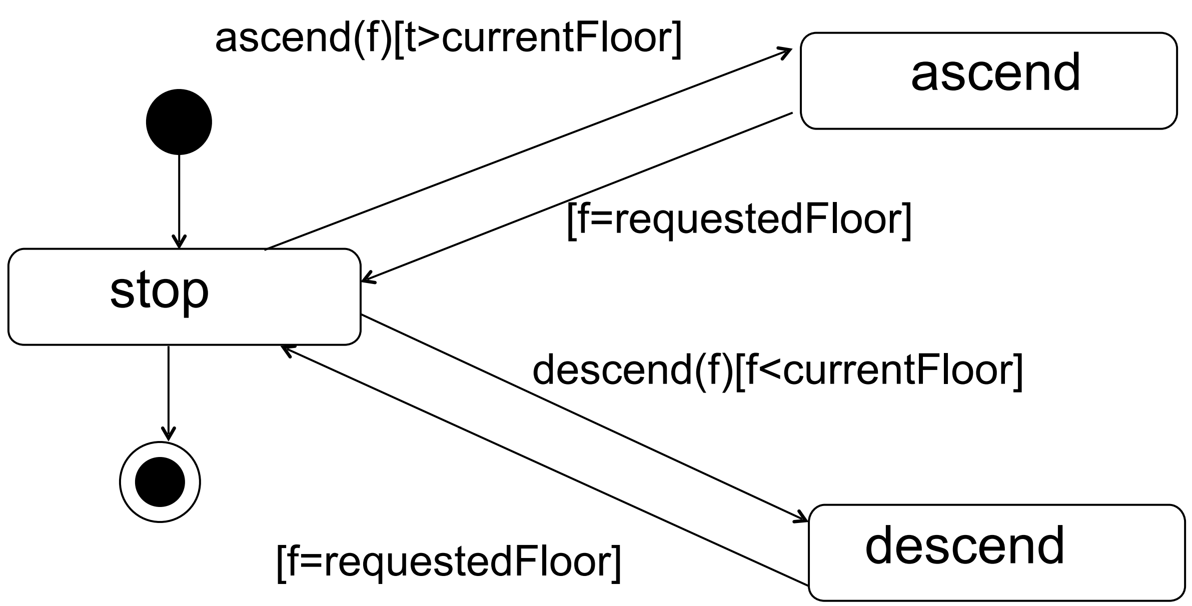
State diagrams

- Historic state
  - Allowing to memorize the current state when exiting a composite state



State diagrams

- Example
  - Modelling an elevator's states



State diagram

- Example
  - Modelling the cash register system