CHRISTIAN HUEMER
MARION SCHOLZ

Object-Oriented Modeling with UML
Part IV – Activity Diagram

Activity Diagram

The Activity Diagram

Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

Introduction

- Focus on **procedural processing aspects**
- Specification of **control** and/or **data flow** between work steps (actions) to realize an activity
- **Activity Diagram in UML2:**
  - process-oriented language concepts
  - based on Petri nets and BPEL, among others
- Language concepts and notation variants cover a **wide range of applications**
  - Modeling of object-oriented and non-object-oriented systems
  - In addition to the suggested graphical notation, any other notation (e.g. pseudocode) is also permitted

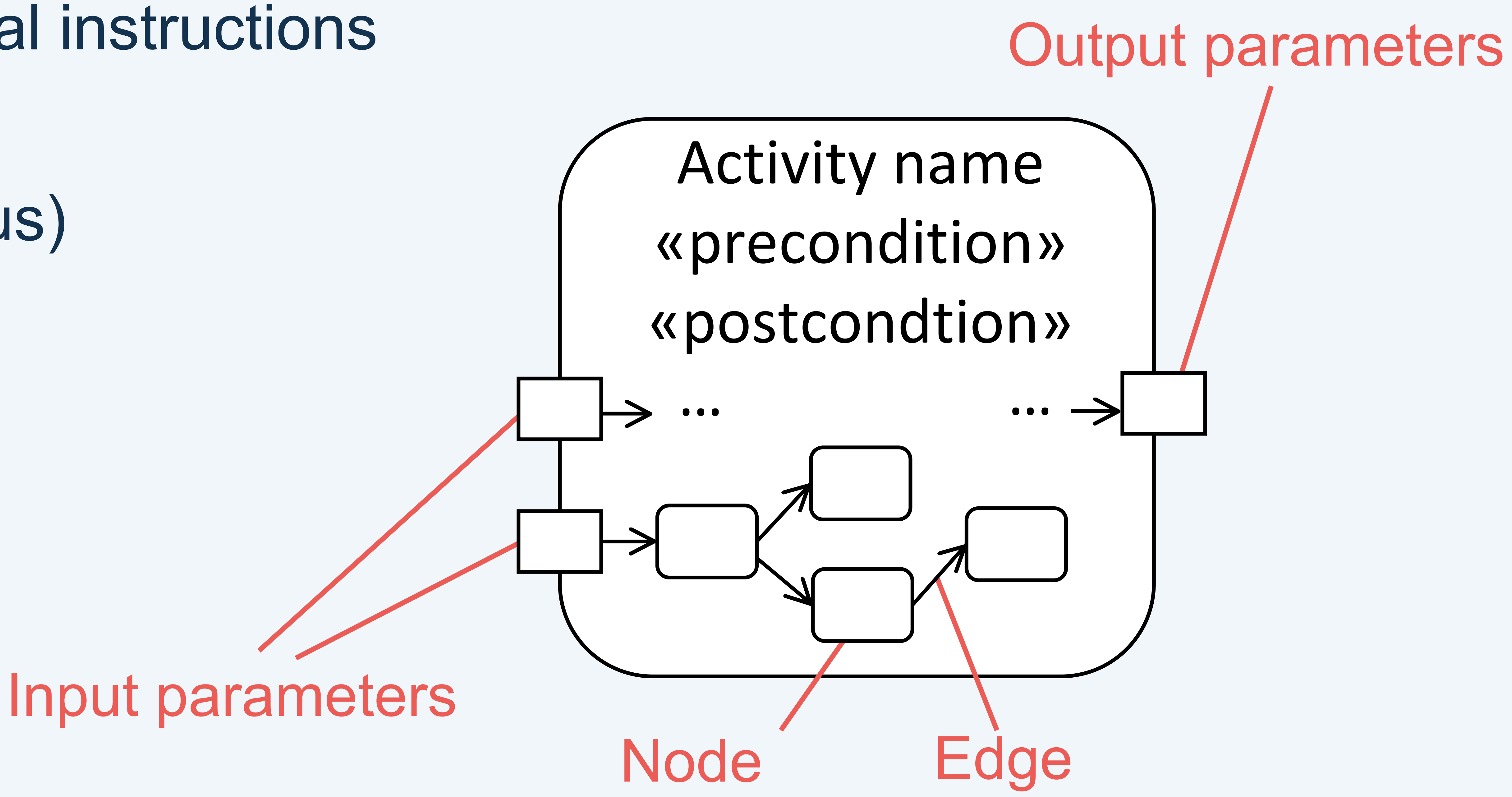
Activity Diagram
Activities, Actions and Their Transitions

Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

Activity

- An activity is a directed graph
  - Nodes: Actions (or activities) and objects
  - Edges: Control and data flows
- Control and data flows define potential “processes”
- Specification of user-defined behavior at different levels of granularity
  Examples:
  - Definition of an operation in the form of individual instructions
  - Sequence of a use case
  - Specification of a business process (autonomous)
- optional:
  - Parameters (e.g. for operations)
  - Pre- and post-conditions that must apply at the start and end of the activity

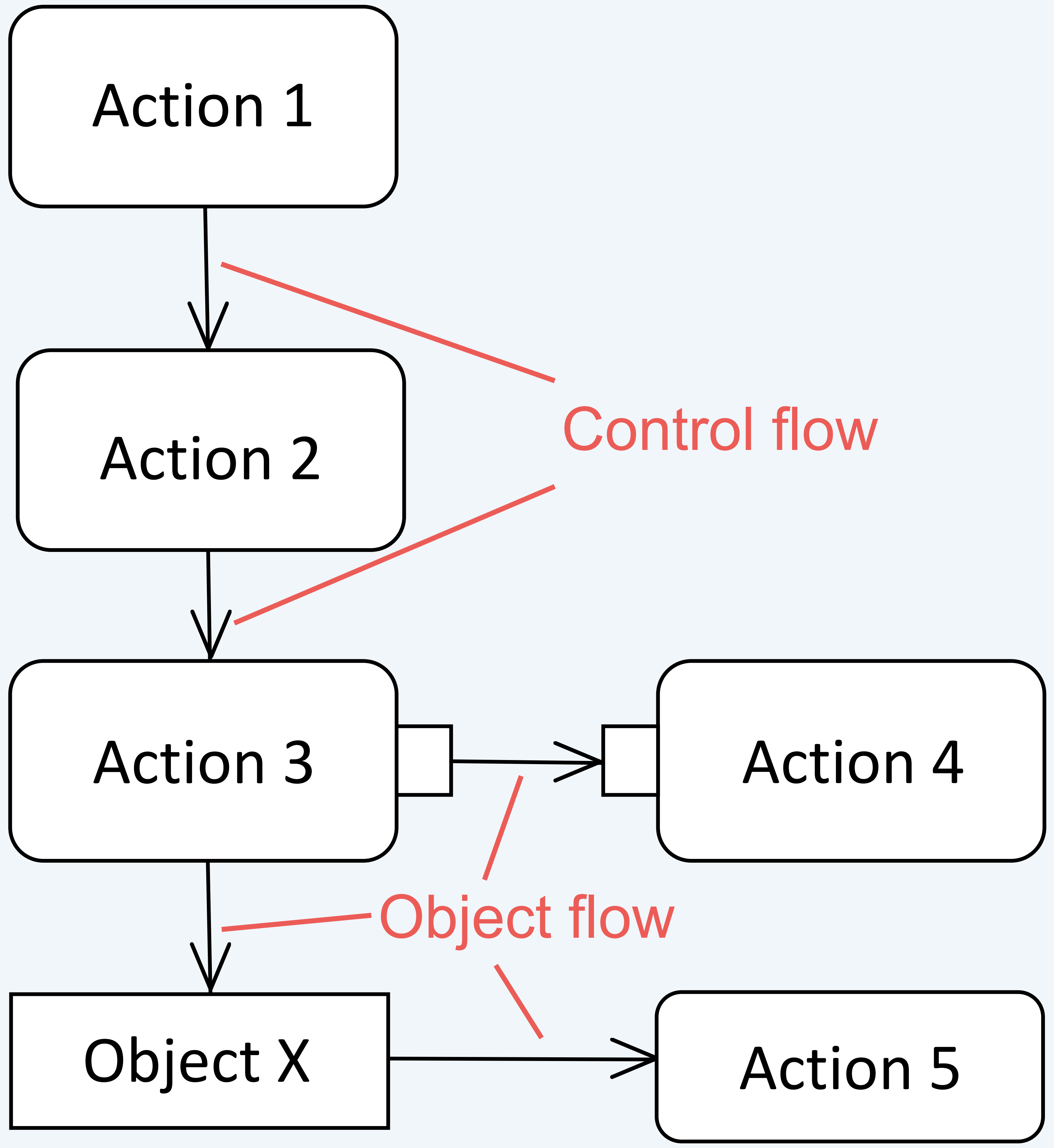


Actions

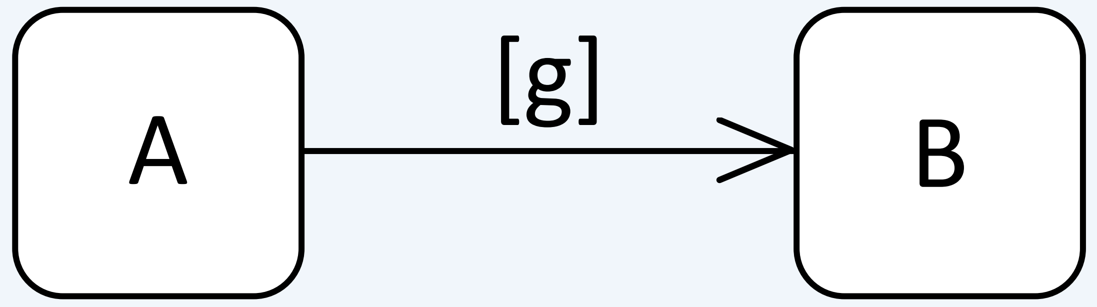
- **Elementary building blocks**
- **Atomic**, but can be aborted
- **Language-independent**, but definition in any programming language possible
- Actions can process input values into output values
- Special **notation** for 44 different action types
- **Categorization** of predefined actions:
  - Communication-related actions (e.g. signals and events)
  - Object-related actions (e.g. creating and deleting objects)
  - Actions related to structural characteristics and variables (e.g. setting and deleting individual values of variables)
  - Link-related actions (e.g. creating and deleting links between objects as well as navigation)

Edges

- Edges connect nodes and define **possible sequences** of an activity
  - Control flow edges
    - Express a **pure control dependency** between predecessor and successor nodes
  - Object flow edges
    - Transport additional data and thus also express a **data dependency** between predecessor and successor nodes



- Guard condition
  - Determines whether control and object flow continues or not



Activity Diagram
The Start and End of Processes

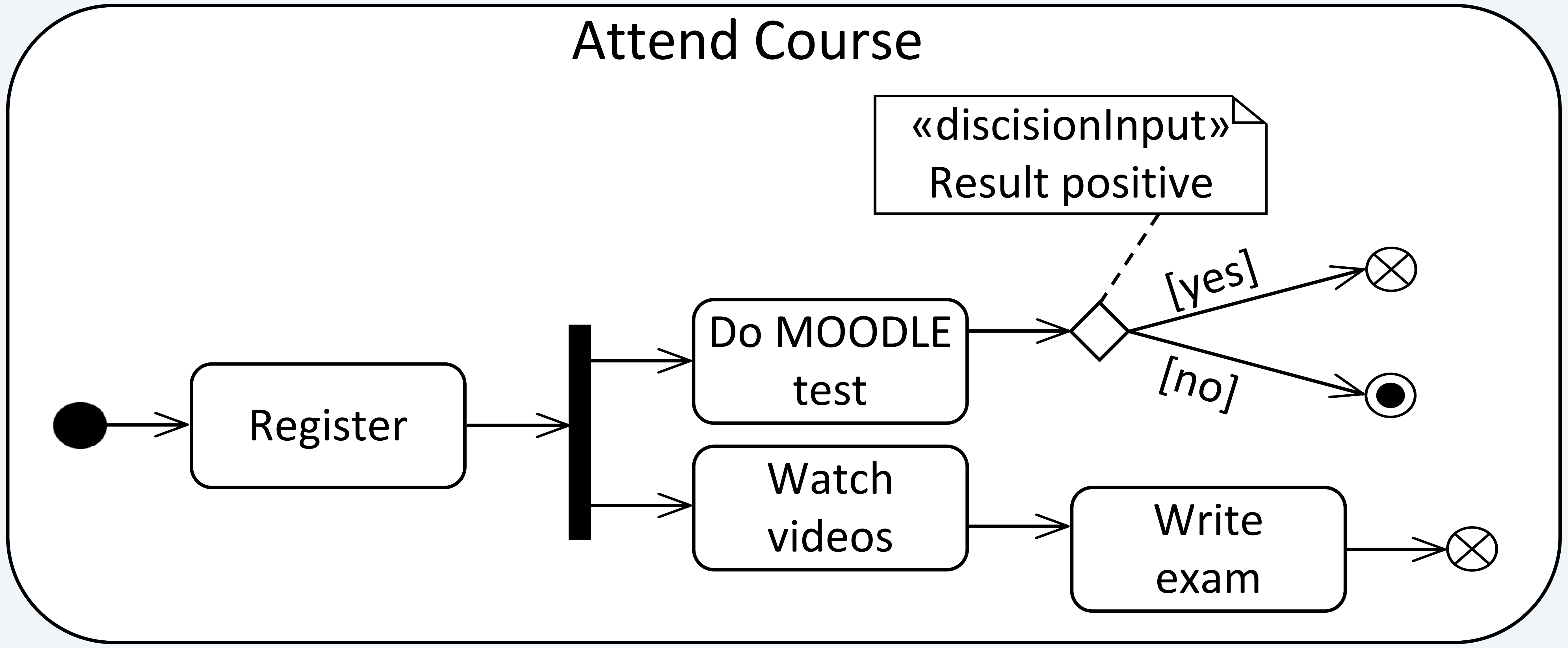
Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

Start and End of Activities and Processes

- ● Initial node
  - Start of an activity sequence
  - Supplies all outgoing edges with control tokens
  - Storage of tokens permitted, but guard conditions may block transfer
  - None or several initial nodes allowed per activity
- ◉ Activity final node
  - Ends all processes of an activity and the life cycle of an object
  - The first token that reaches a final node ends the activity
  - No further actions are executed
  - Control tokens are deleted, but data tokens on the activity's output pins are not
  - Several activity final nodes allowed per activity
- ⊗ Flow final node
  - Ends an activity sequence

Example: Completion of a course



Activity Diagram

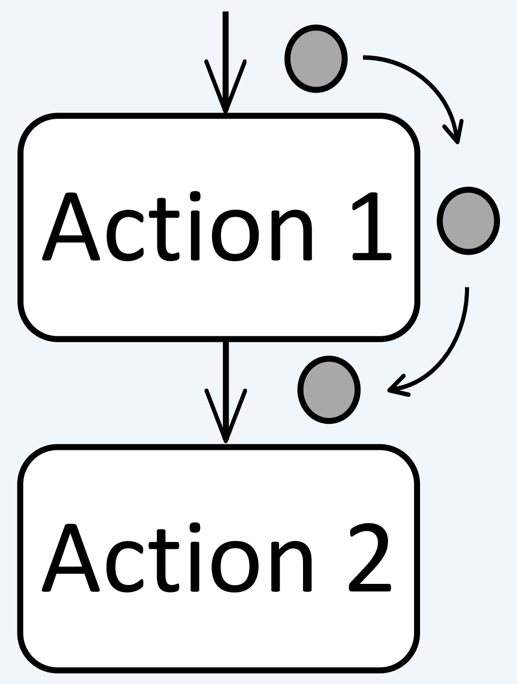
The Token and Alternative Processes

Christian Huemer und Marion Scholz

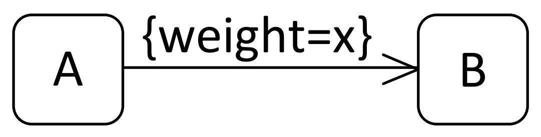
Presented by Nicholas Bzowski

Token

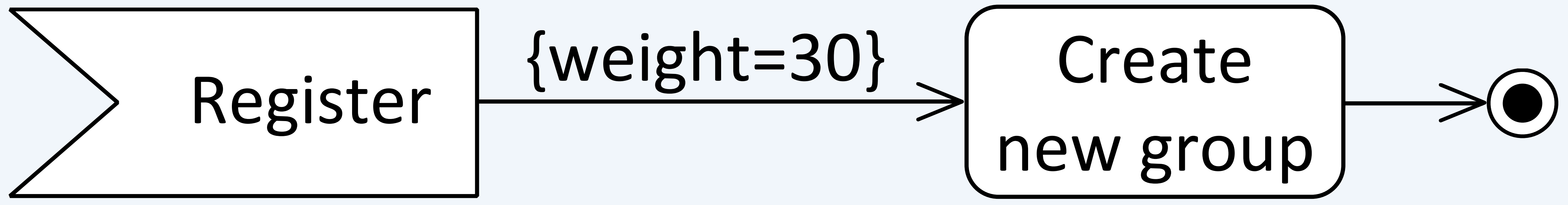
- "**Virtual coordination mechanism**"
  for the description of activity sequences
- Token describes possible sequence of an activity
- Tokens flow along the edges from predecessor to successor nodes
- Action starts when there is a token on all incoming edges
- After the action has been carried out, a token is assigned to all outgoing edges
- Guard condition can prevent tokens from being transferred
- Distinction between control and data tokens
  - **Control token**:
    "Execution permission" for the successor node
  - **Data token**:
    Transfer of data value or reference to object



The Weight of an Edge

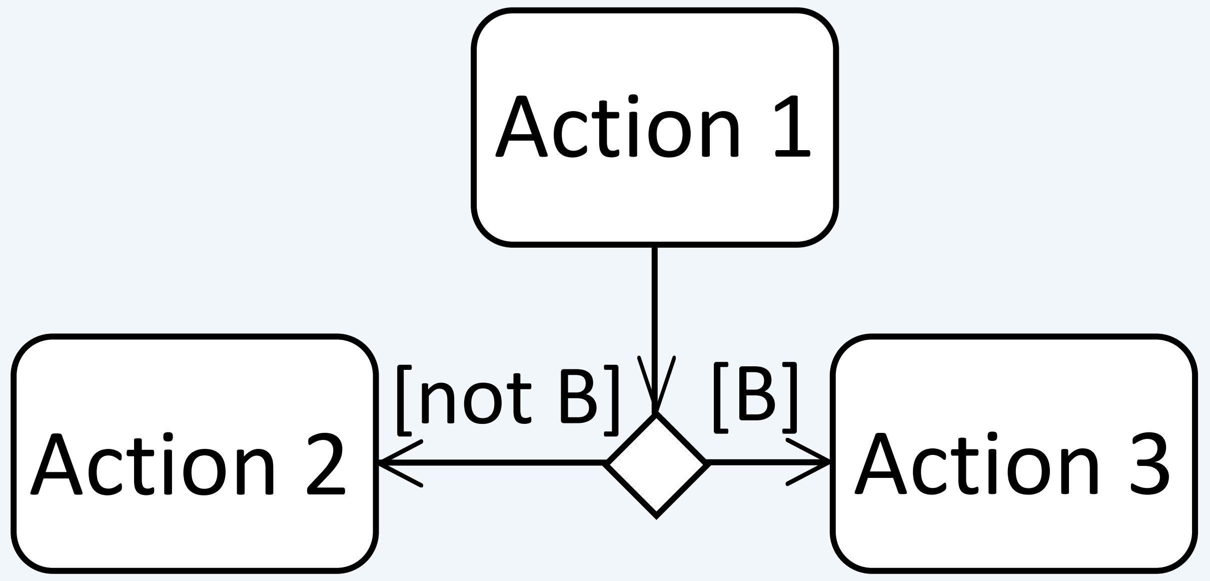


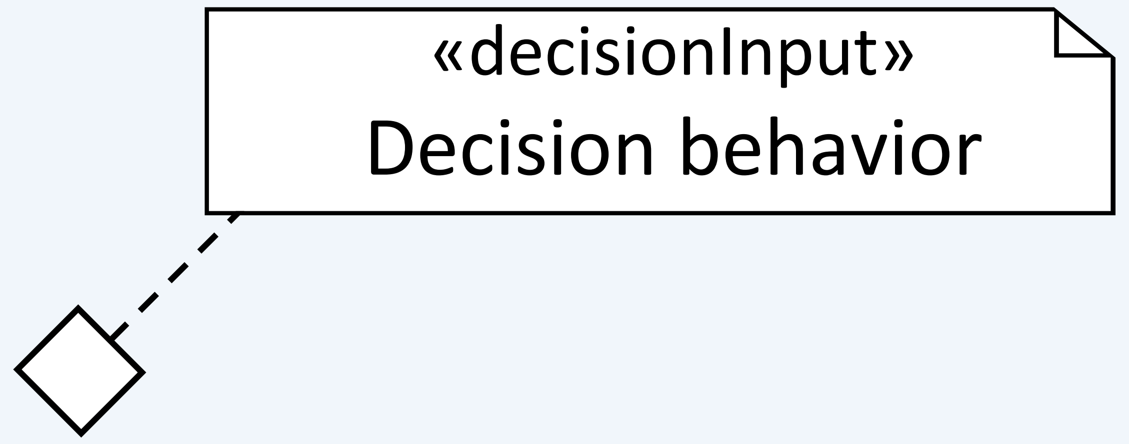
- Minimum number of tokens that must be present for an action to be executed
- Default: **1**



Alternative Processes - Decision Nodes

- Defines alternative branches and represents a “switch” for the token flow
  - Can also be used for modeling loops
- Guard condition
  - Selects the branch
  - Mutually exclusive
  - `[else]` is pre-defined
- Decision behavior
  - Enables more detailed specification of the decision at a central point
  - Arrival of tokens starts the decision behavior - data tokens act as parameters

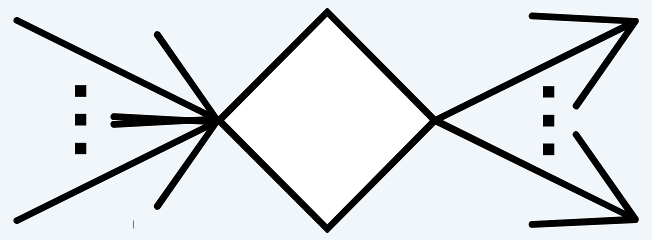




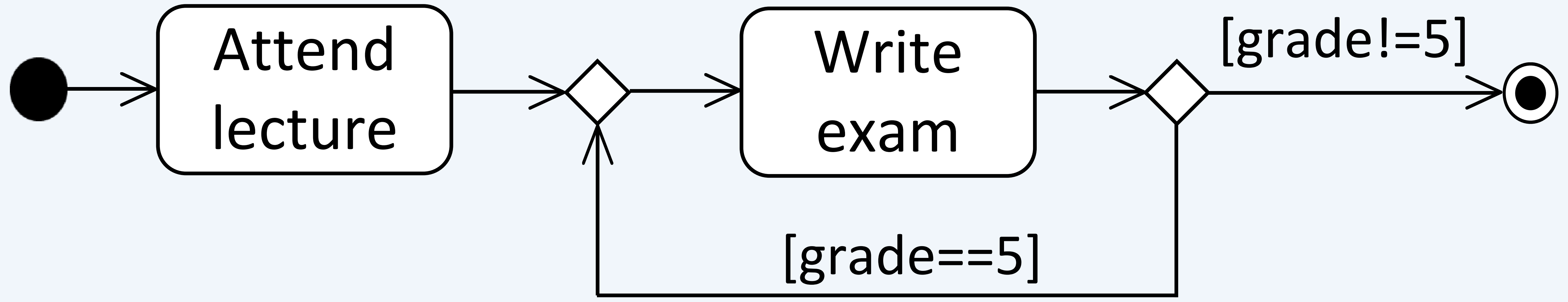
Alternative Processes - Merge Nodes

- A merge node brings alternative processes together again
- Tokens are passed on to the successor node as soon as possible

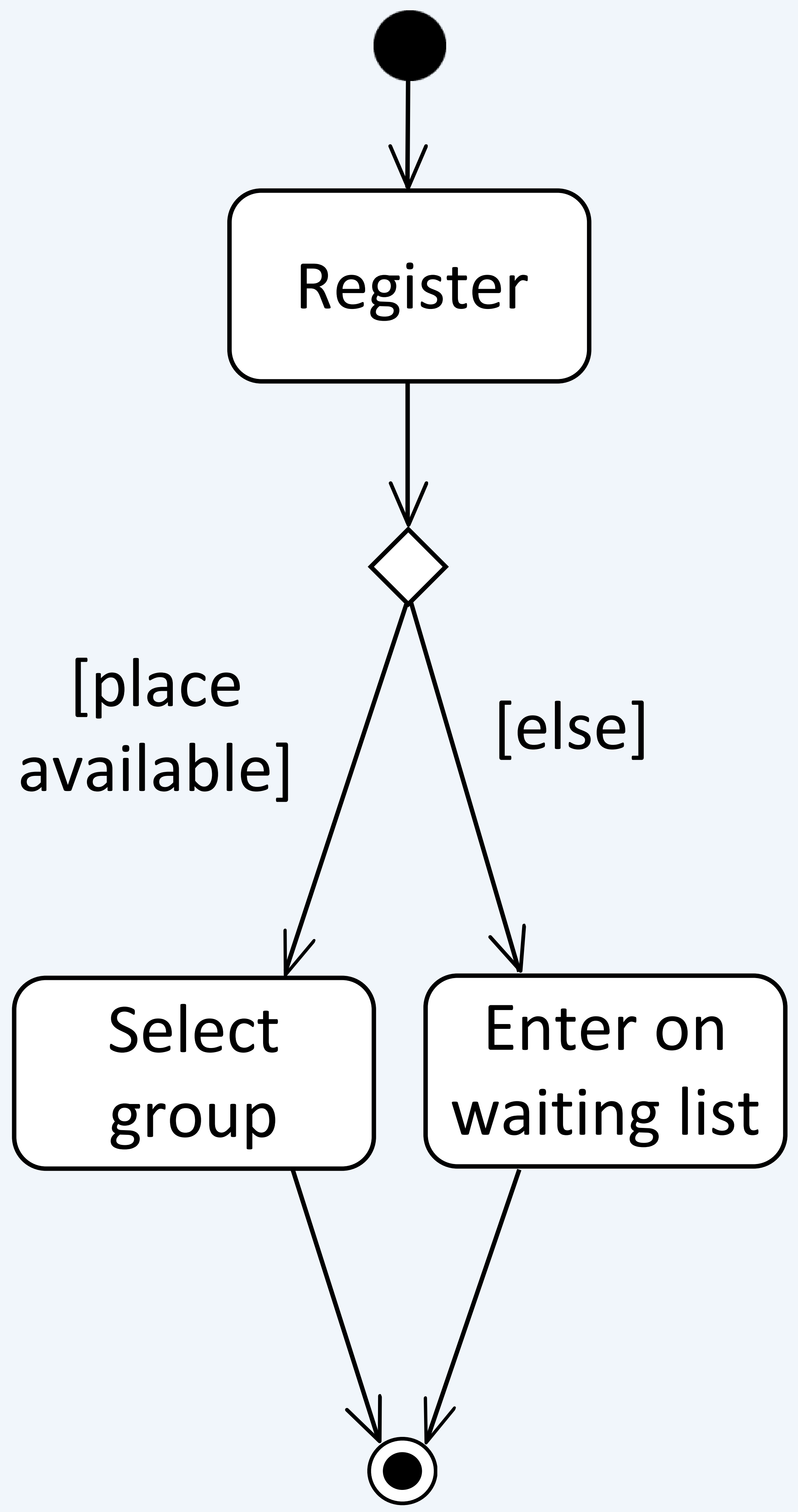
- Combined decision and merge node



- Ex.:



Alternative Processes - Example



Activity Diagram

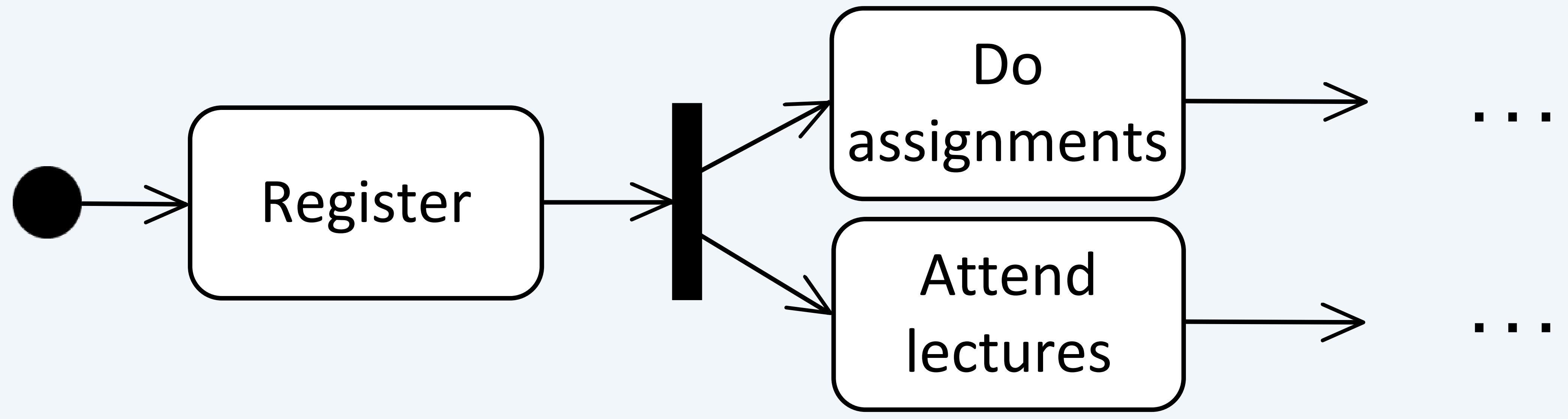
The Token and Concurrent Processes

Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

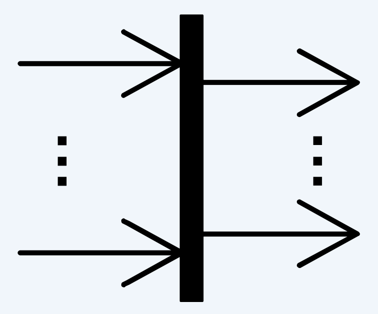
Concurrent Processes – The Parallelization Node

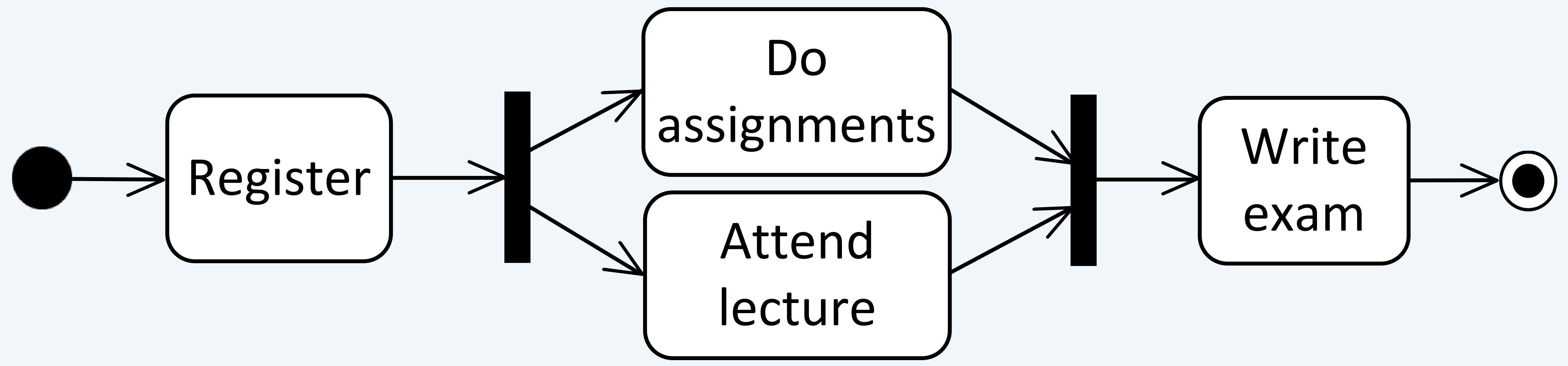
- For modeling the splitting of processes
- Incoming tokens are duplicated for all outgoing edges as soon as at least one guard condition accepts them
- Non-accepted tokens are stored
- Ex.:



Concurrent Processes – The Synchronization Node

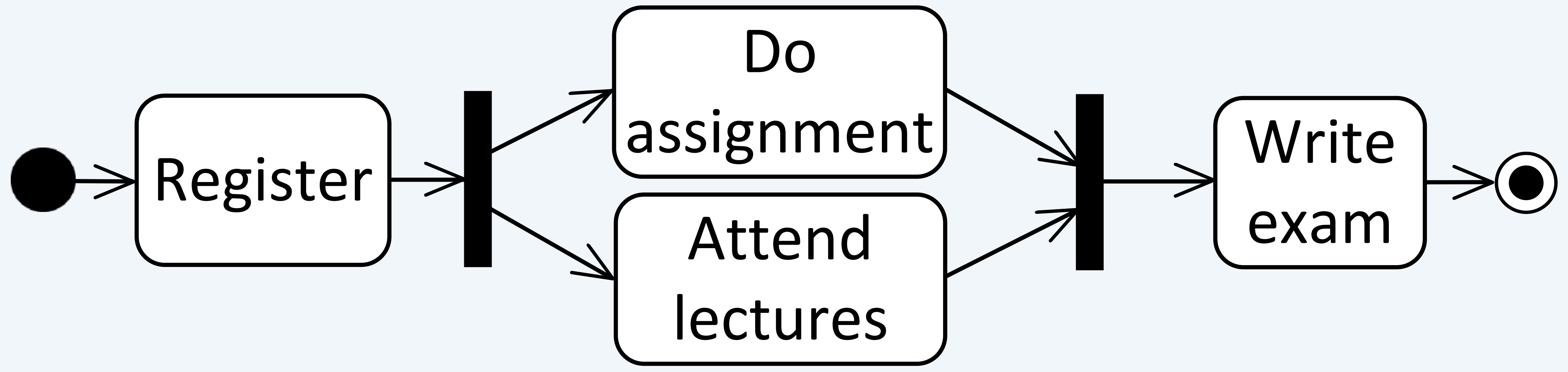
- Merges concurrent processes
- Tokenverarbeitung
  - Unification of the tokens as soon as all edges are reached
  - Control tokens of different edges are combined and only a single token is passed on
  - All data tokens are passed on
  - Only data tokens are passed on for control and data tokens
- Combined parallelization and synchronization node:

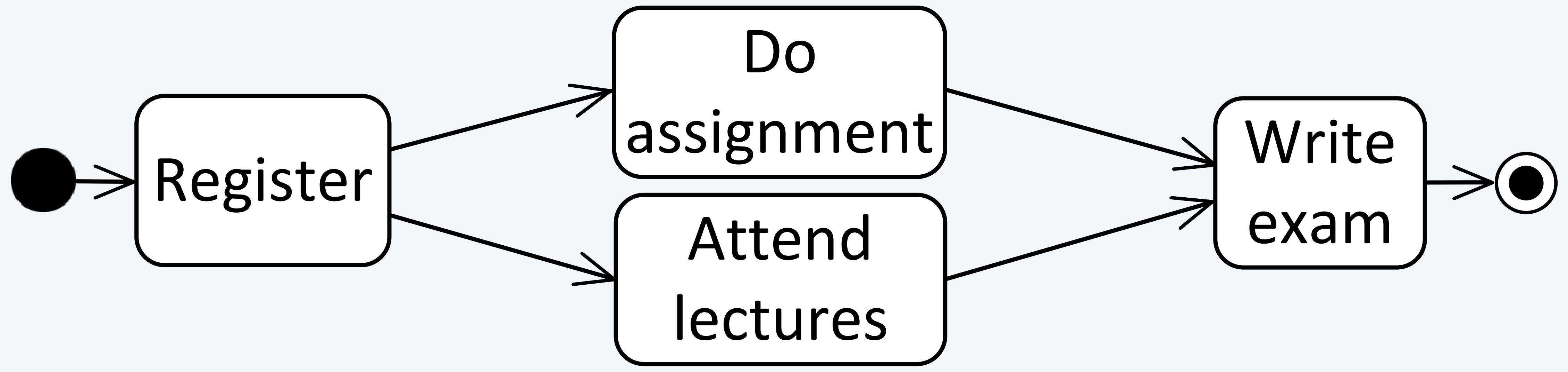




Example: Alternative Modeling

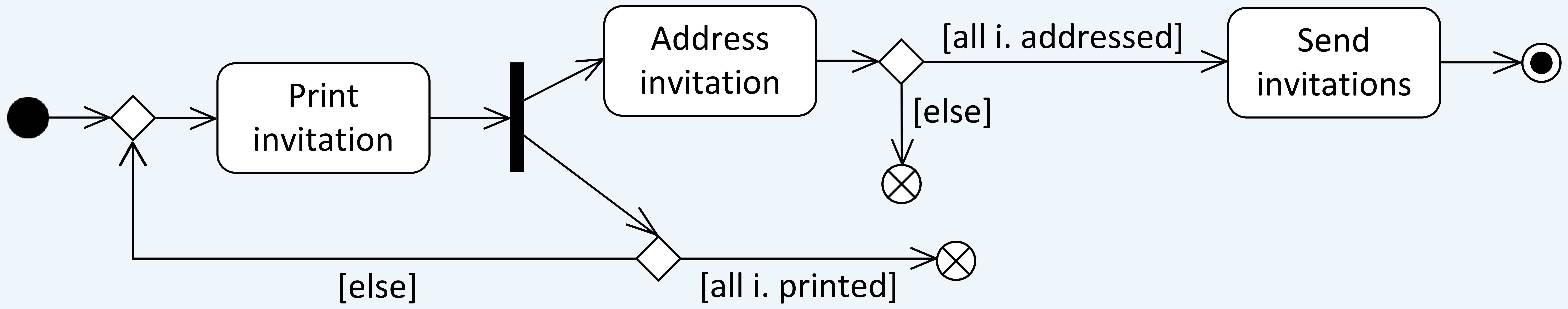
- Equivalent control flow



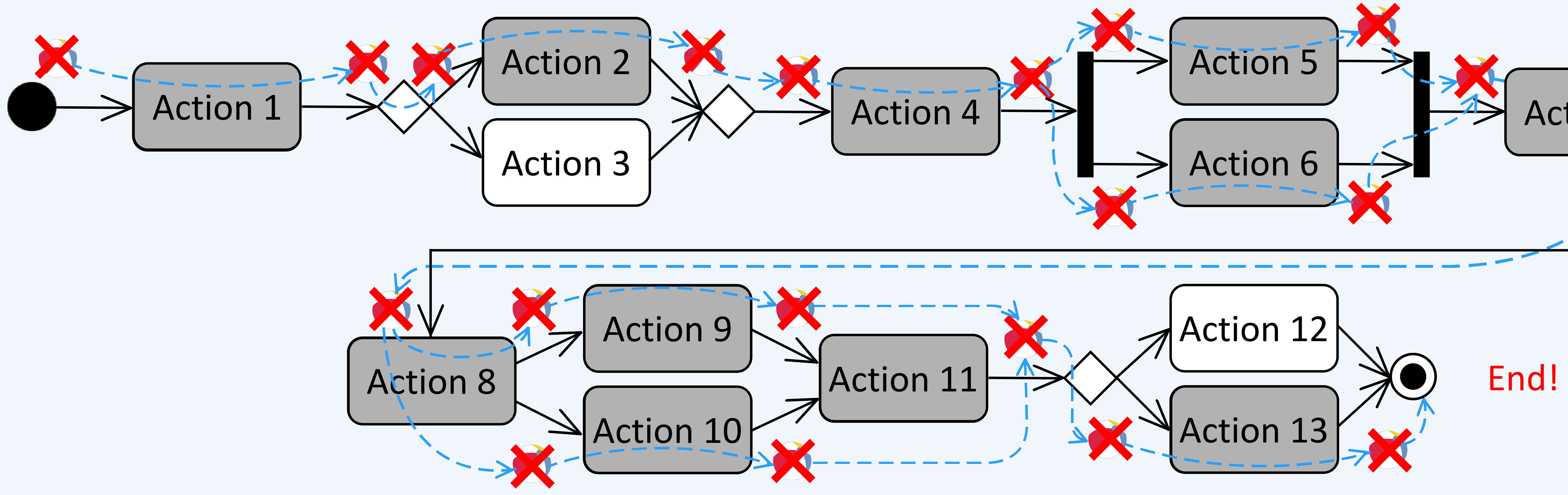


Example: Creating and sending invitations to a meeting

- As new invitations are printed, invitations that have already been printed are addressed.
- As soon as all invitations have been addressed, they will be sent out.



Token – Example (Control Flow)



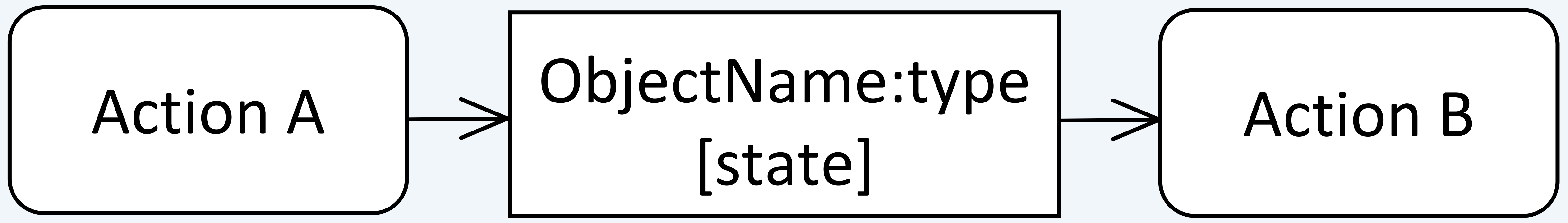
Activity Diagram
The Object Node

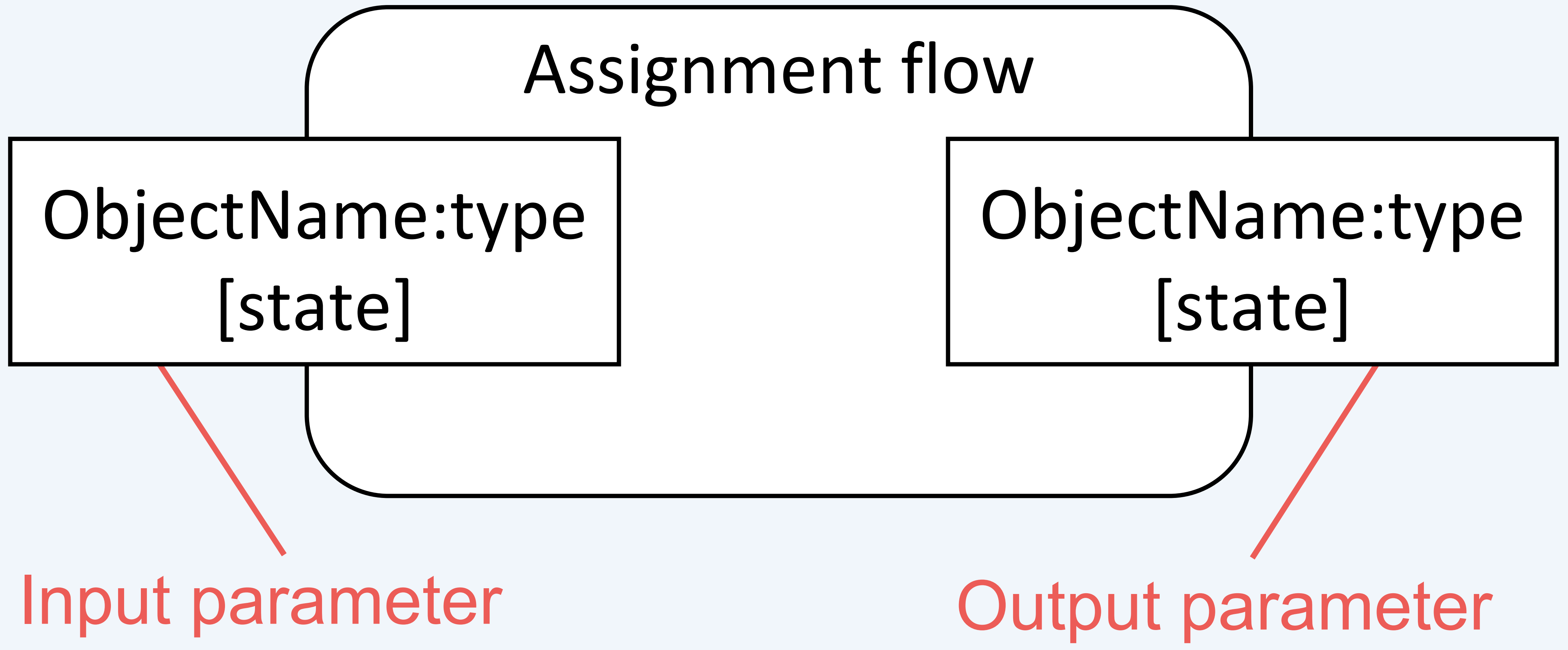
Christian Huemer und Marion Scholz

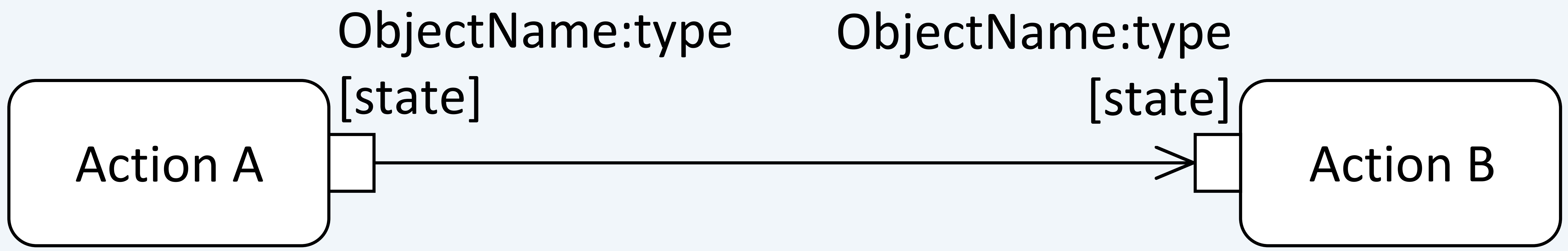
Presented by Nicholas Bzowski

Object Nodes

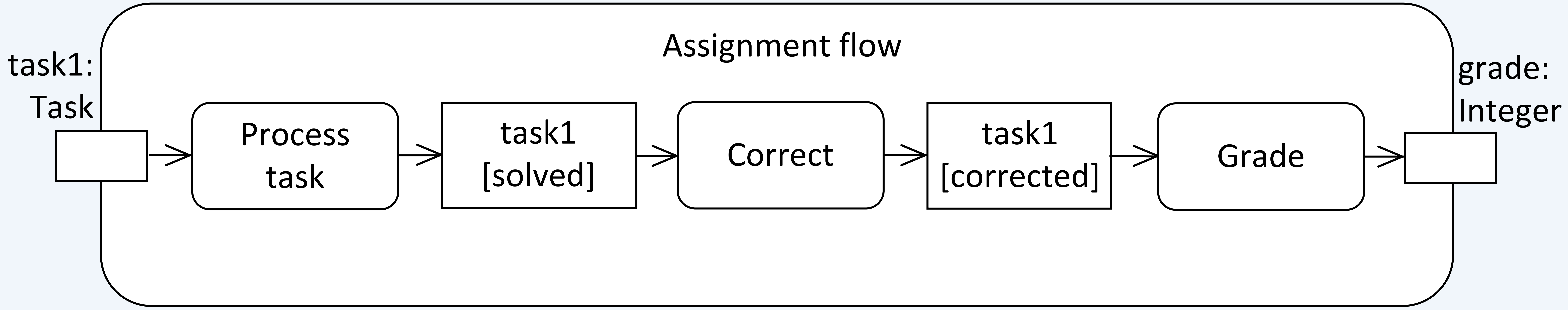
- Contents: **Data tokens**
- Object nodes connect actions via object flows
- Content is the **result of an action** and **input** for another action
- Type specification and condition restriction are optional
- Object nodes as **input/output parameters**
  - for activities (activity parameter node)
  - for actions (pins)

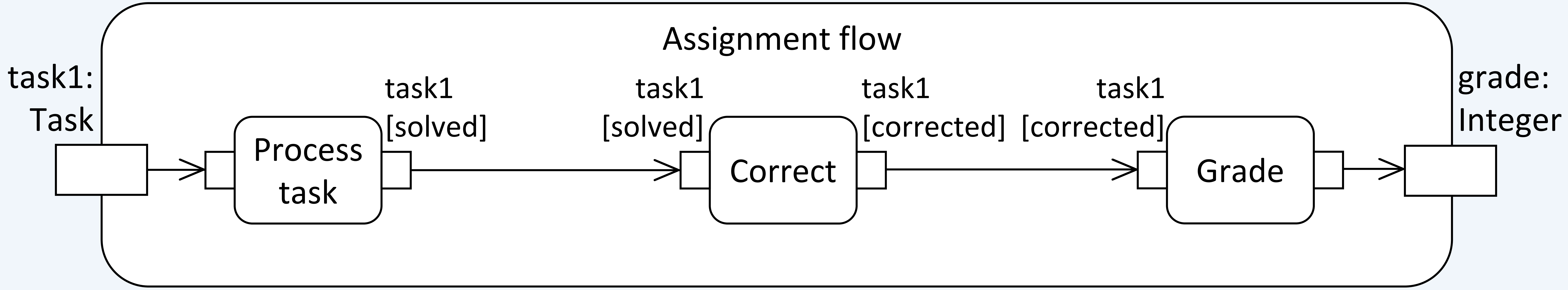




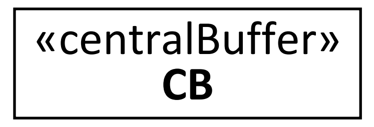


Object nodes for actions: 2 notation variants

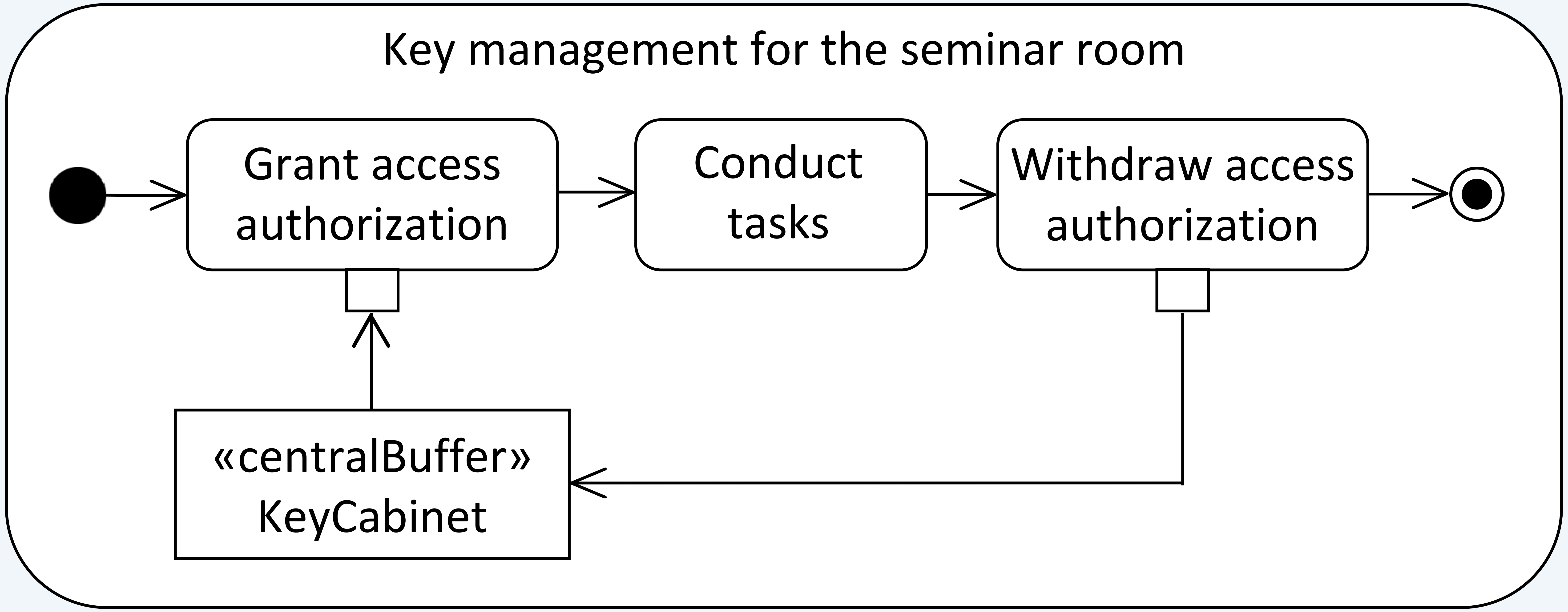




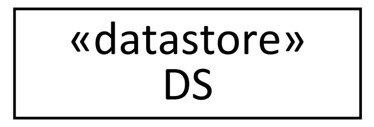
Central Buffer



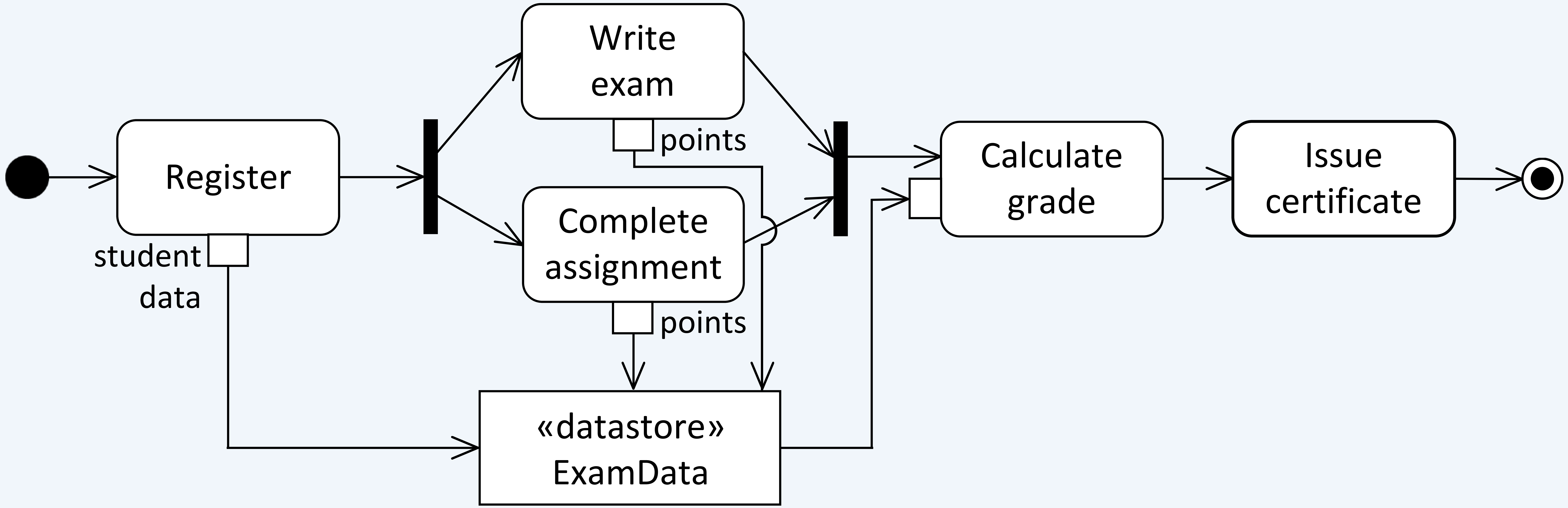
- Central buffering of data tokens
- For storing and transferring object tokens
- Accepts object tokens from object nodes and passes them on to other object nodes
- Transient buffer node
  - Deletes data tokens as soon as it has passed them on


Key management for the seminar room
Grant access authorization
Conduct tasks
Withdraw access authorization
«centralBuffer» KeyCabinet

Data Store



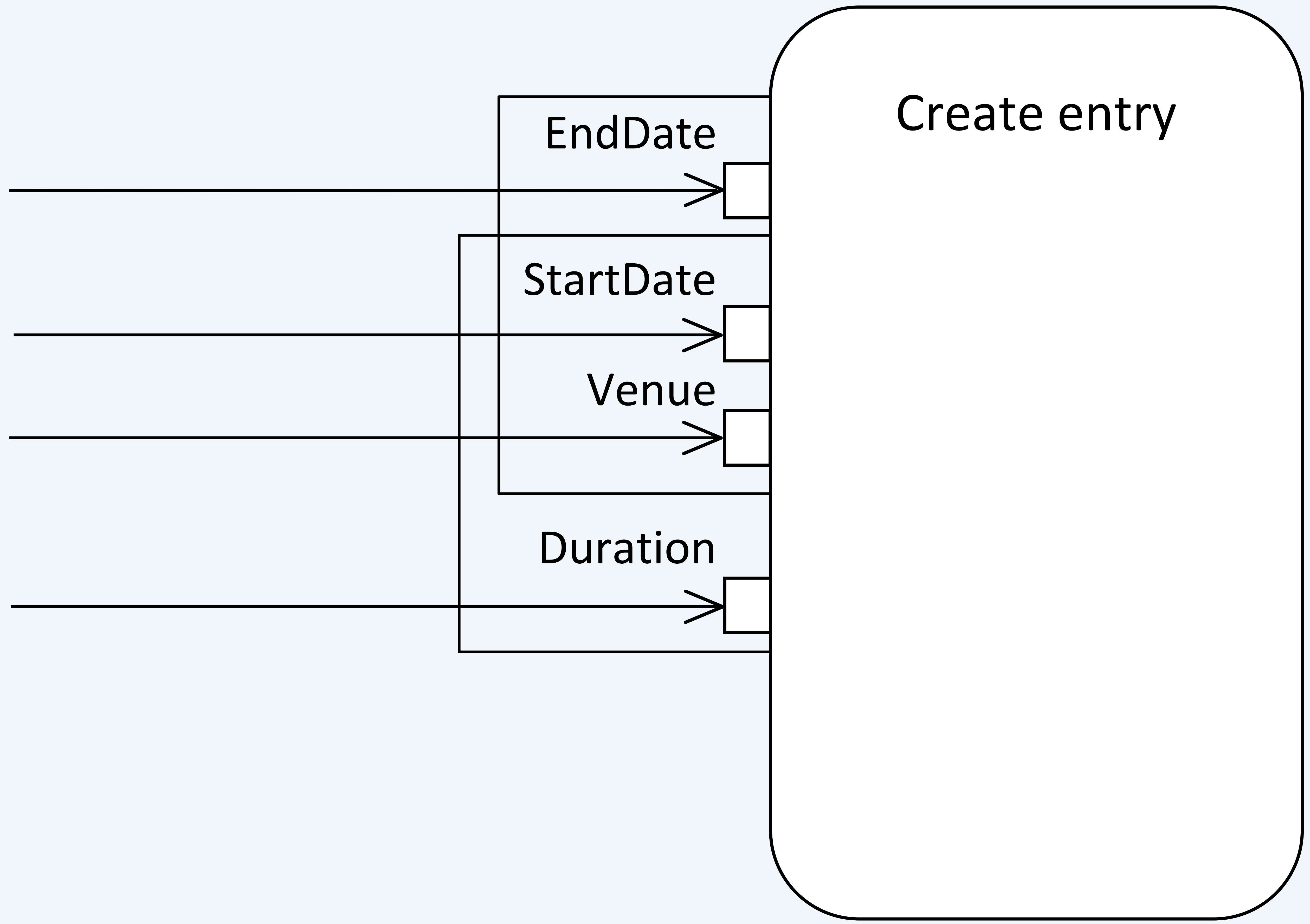
- For storing and transferring object tokens
- Permanent storage
  - Retains data tokens and passes on duplicates
- Does not store identical objects more than once
- Explicit "fetching" of data tokens possible



Parameter Set

- Grouping of parameters
- Alternative groups of input and output values

- Ex: 2 types of appointments



Activity Diagram
The Object Flow and The Partition and Signals and Events

Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

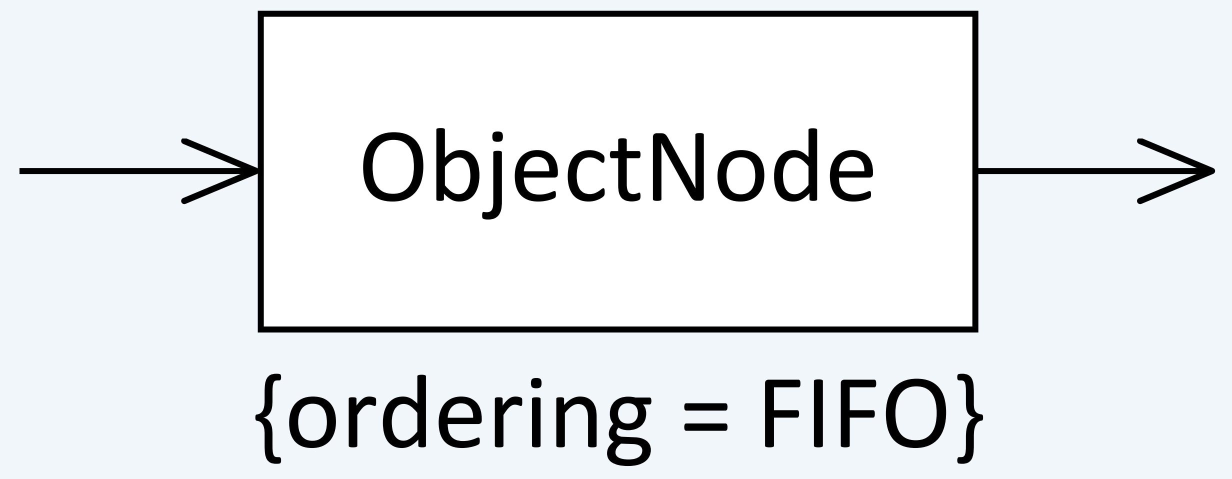
Object Flow (1/4)

- **Transfer** and **control function**
- **Links** actions not directly, but instead **via object nodes**
- Object nodes determine the type of objects to be transferred

- **Control options** for passing on data tokens:
  - Order
  - Upper bound and weight
  - Selection behavior
  - Transformation behavior

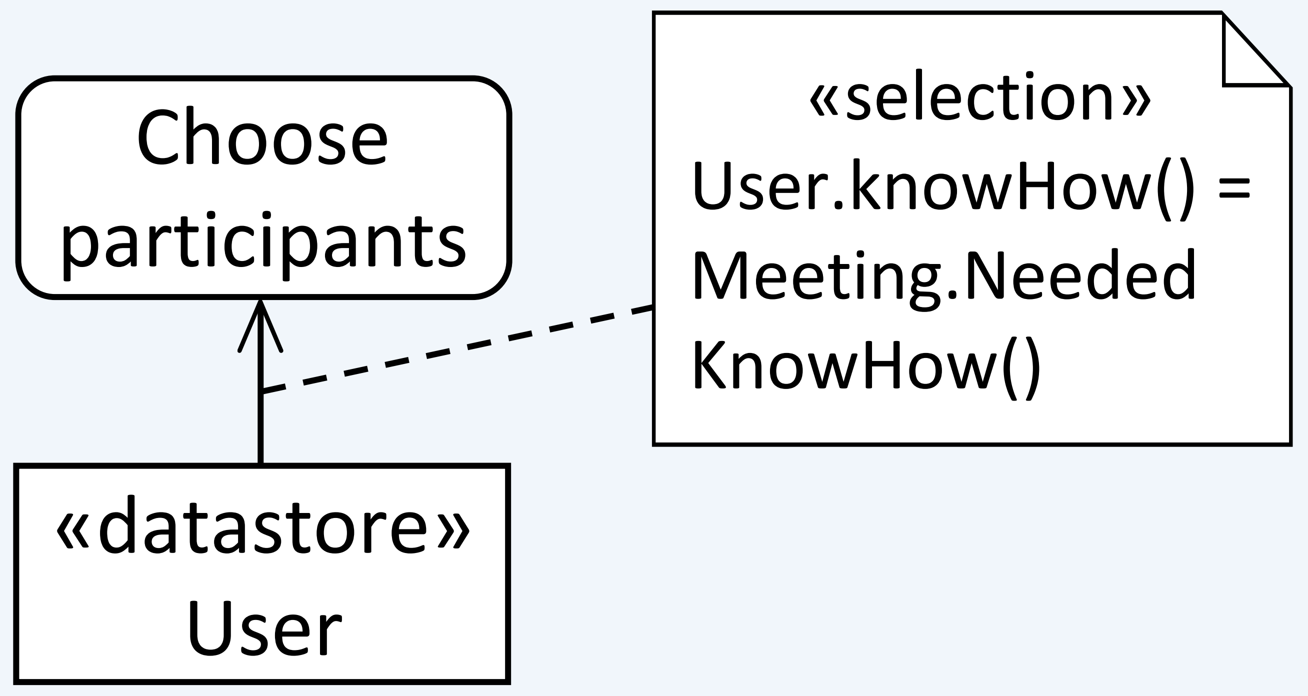
Object Flow (2/4) – Order of token transfer

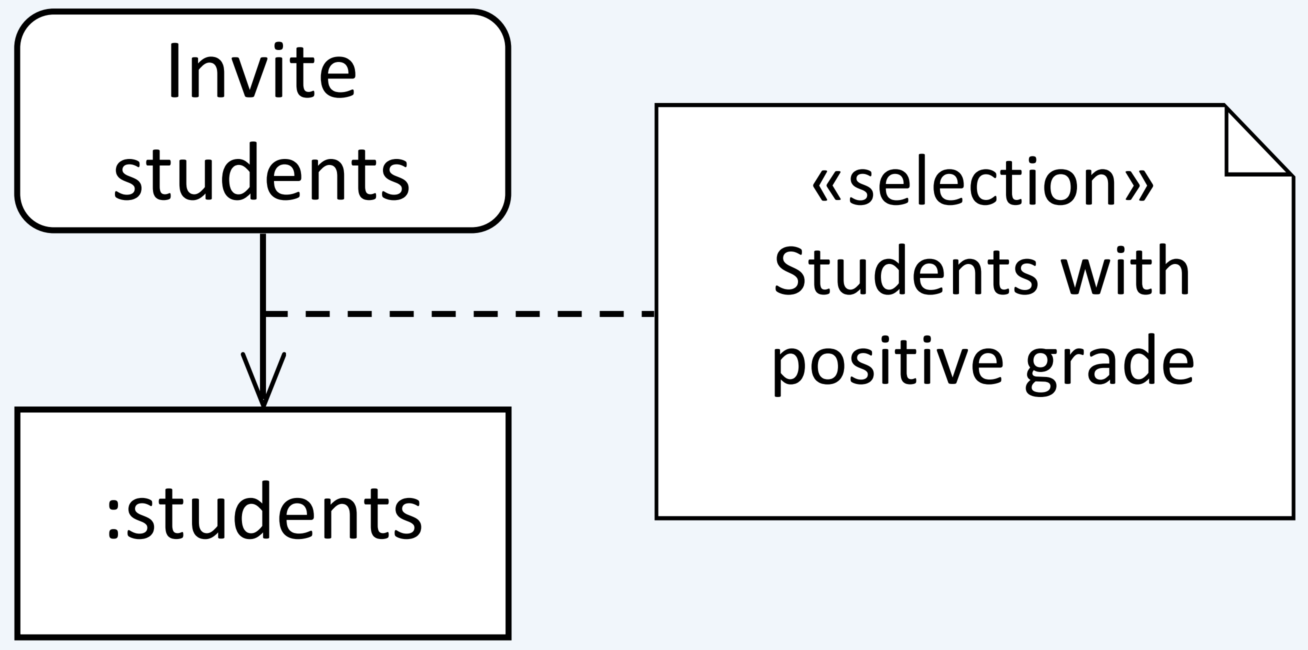
- Explicit definition of the order in which a data token is passed
  - **FIFO** (first in, first out) - {ordering = FIFO}
  - **LIFO** (last in, first out) - {ordering = LIFO}
  - **Ordered** - {ordering = ordered}
    - User-defined sequence (specification of selection behavior)
  - **Unordered** - {ordering = unordered}
    - The order in which the tokens are received has no influence on the order in which they are passed on



Object Fluss (3/4) - Selection behavior

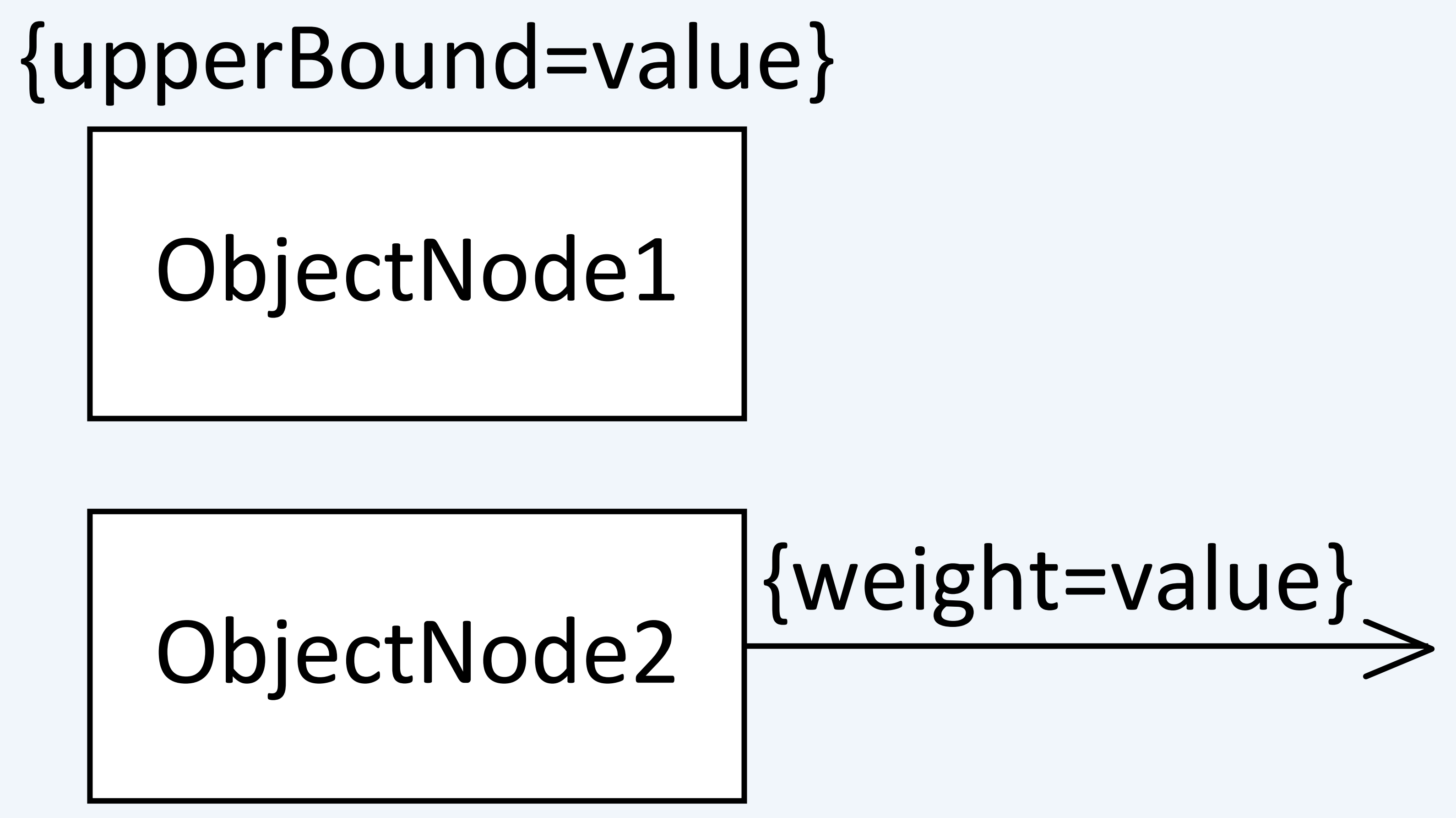
- Selects certain tokens for forwarding
- Object nodes and object flow edges can exhibit selection behavior
- Example:



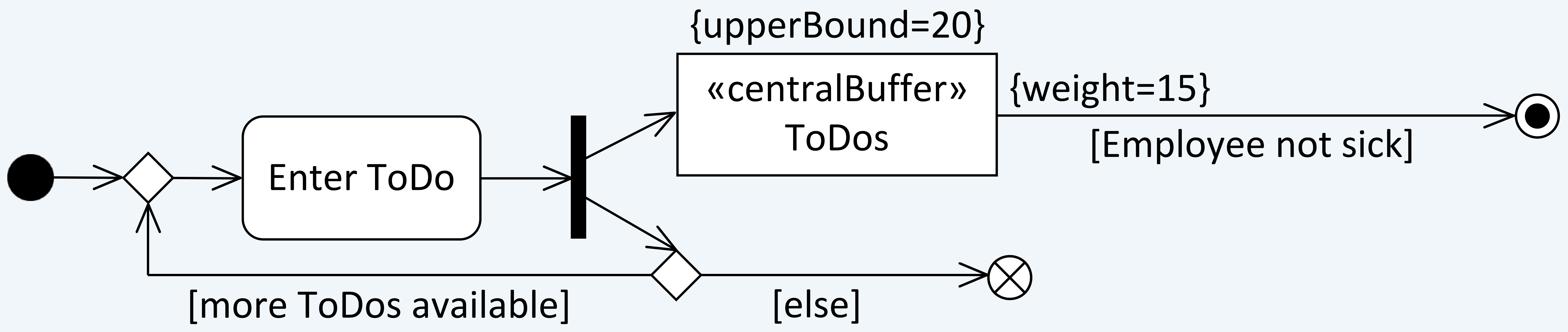


Object Flow (4/4)

- **Upper bound** of an object node
  - Max. number of tokens that may be in this node at any one time
- **Weight** of an object flow edge:
  - Number of tokens that must be present before they are passed on to successor nodes

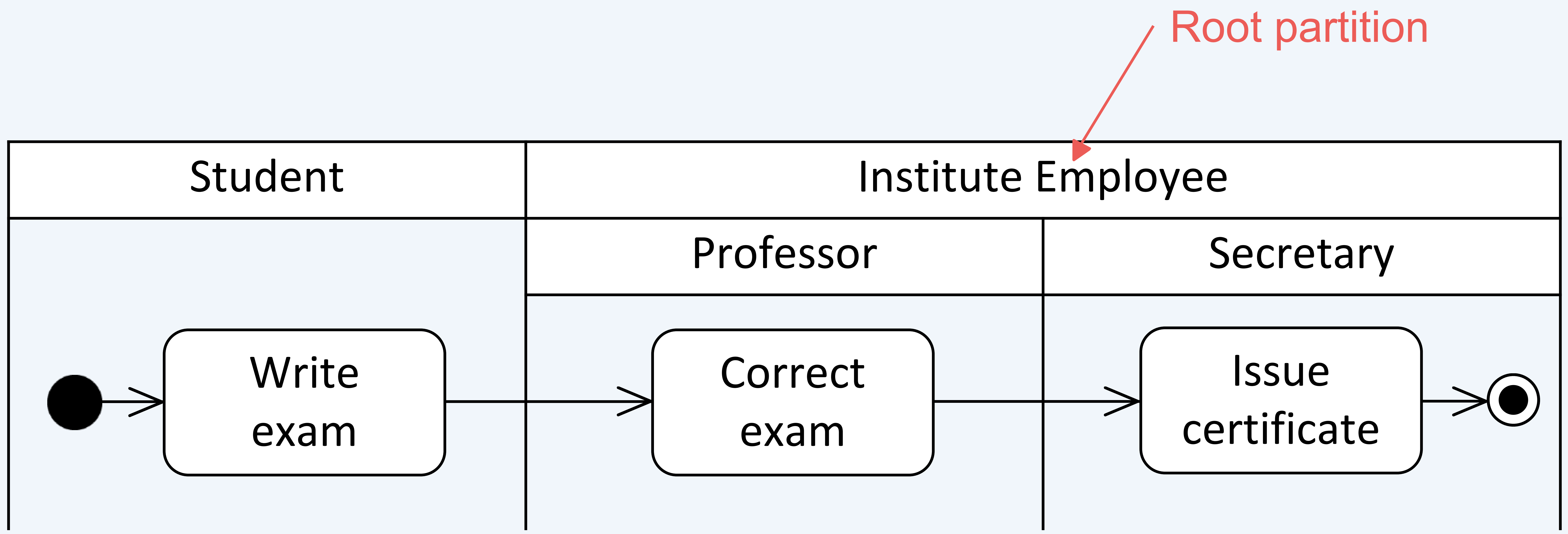


- **Example:** Buffer node can hold a maximum of 20 ToDos.

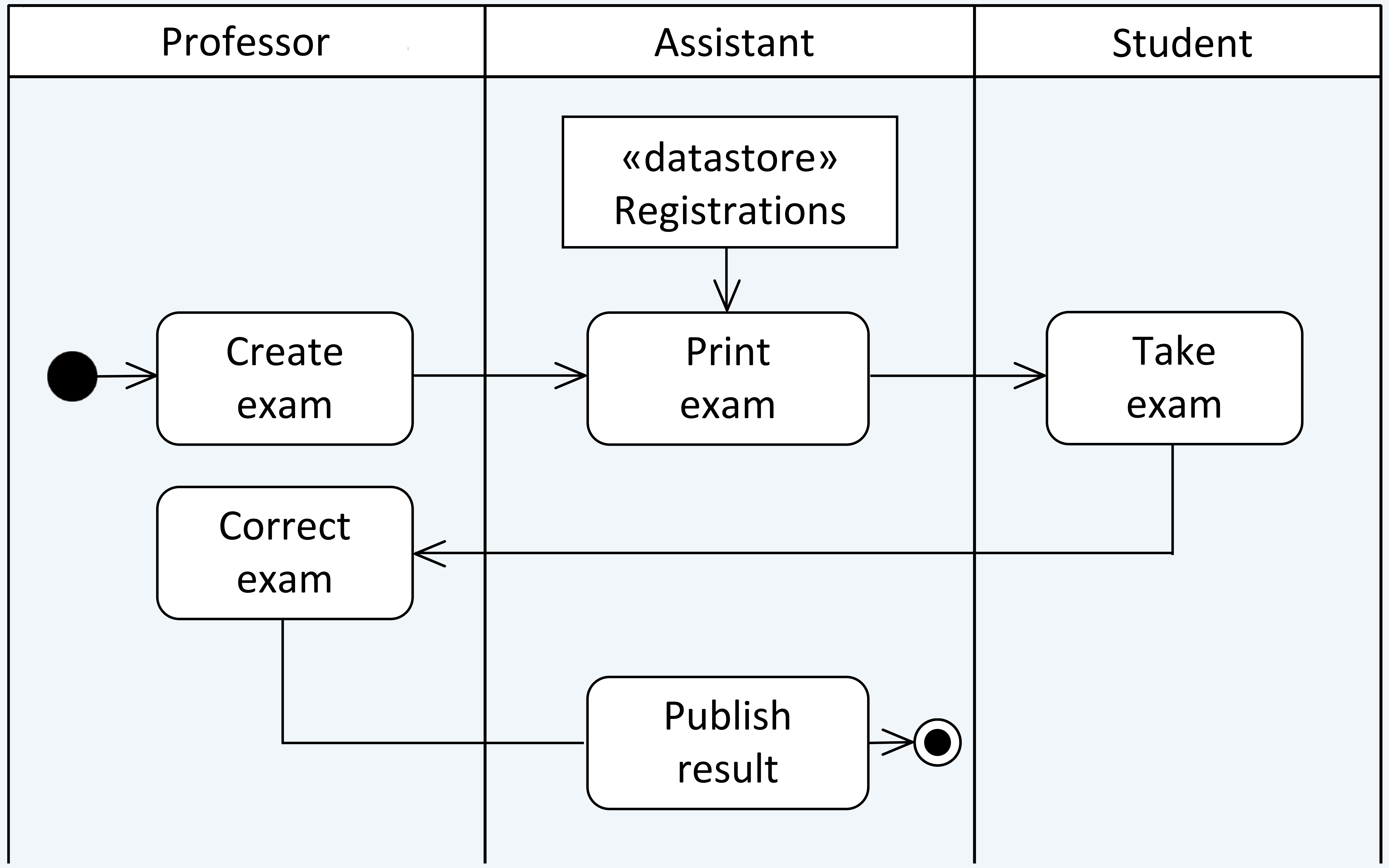


Partitions

- Allow the **grouping of nodes and edges** of an activity according to certain criteria
- **Logical view of an activity** to increase the clarity and semantics of the model
- Hierarchical partitions
  - For nesting at different hierarchy levels

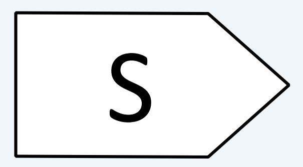


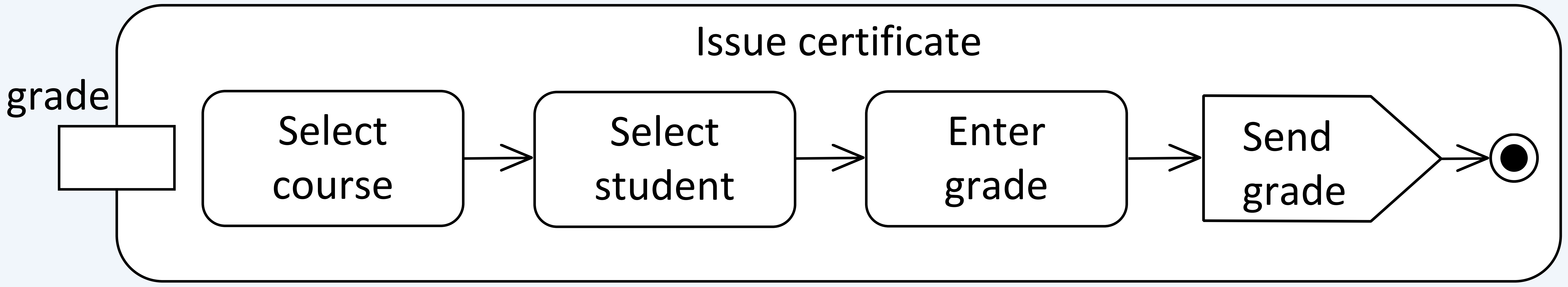
Partitions – Example



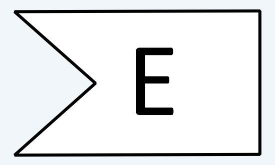
Event-based Actions

- Send signal action

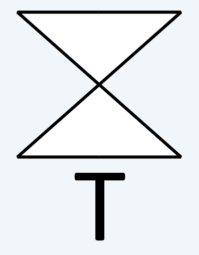


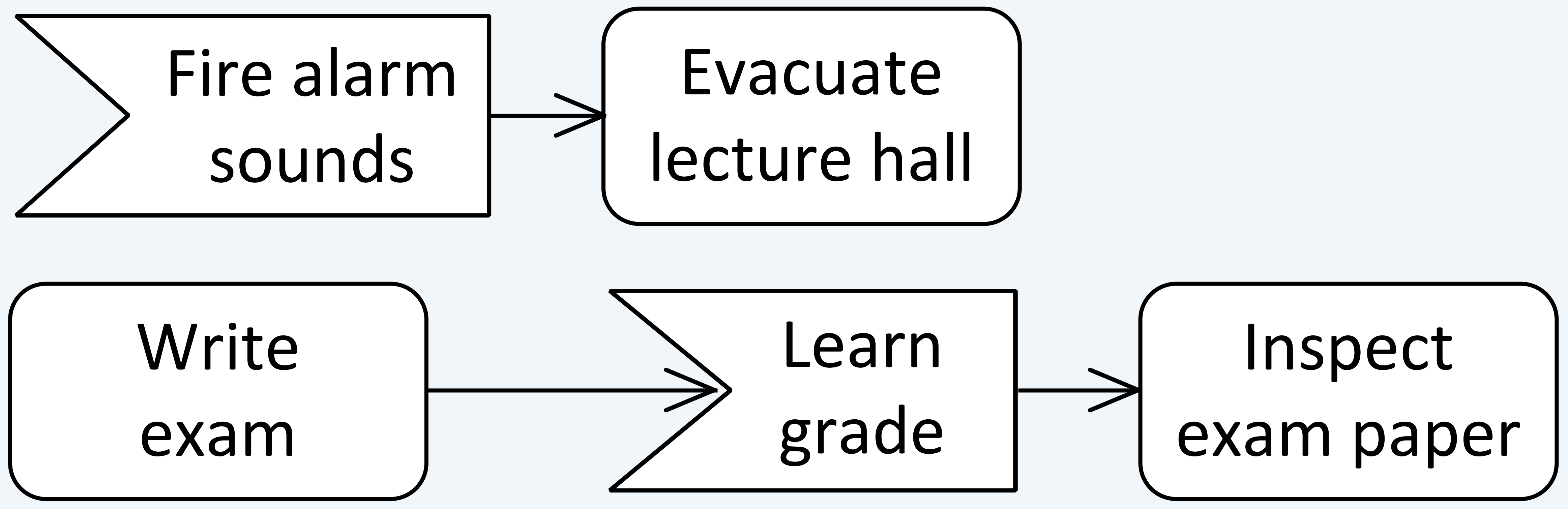


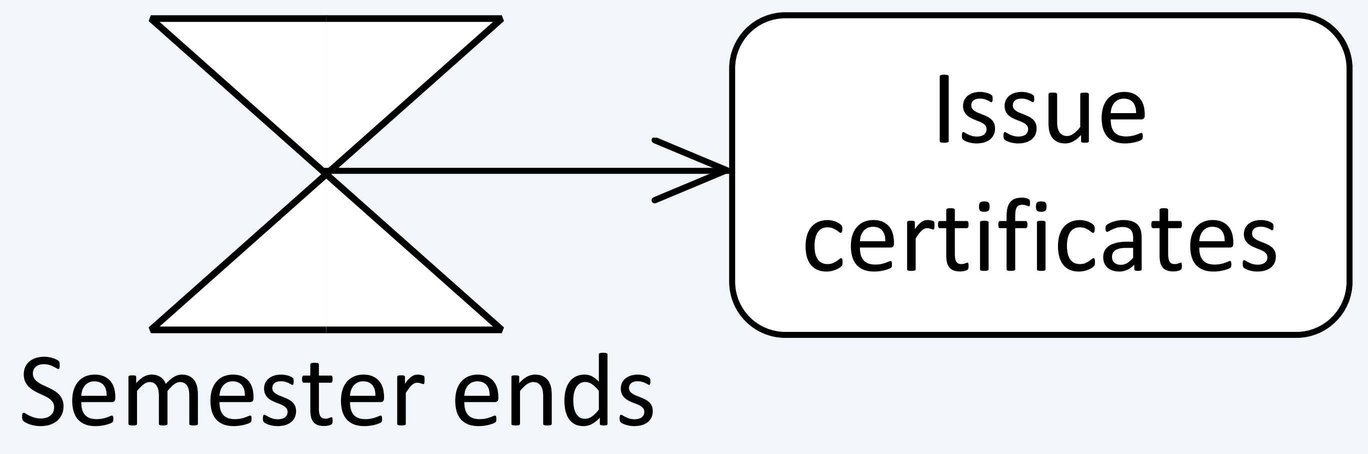
- Accept event actions
  - Asynchronous event



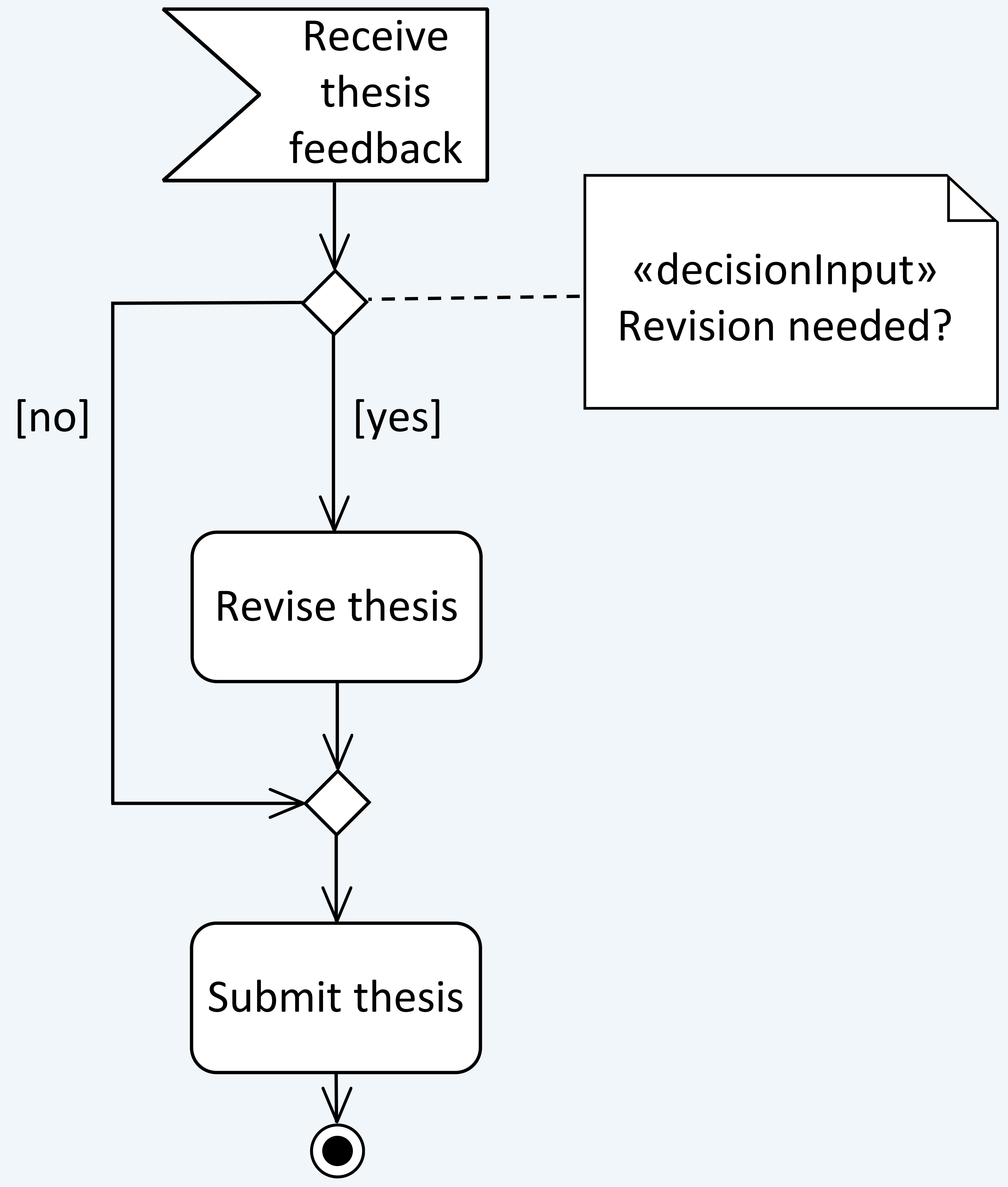
  - Asynchronous time event



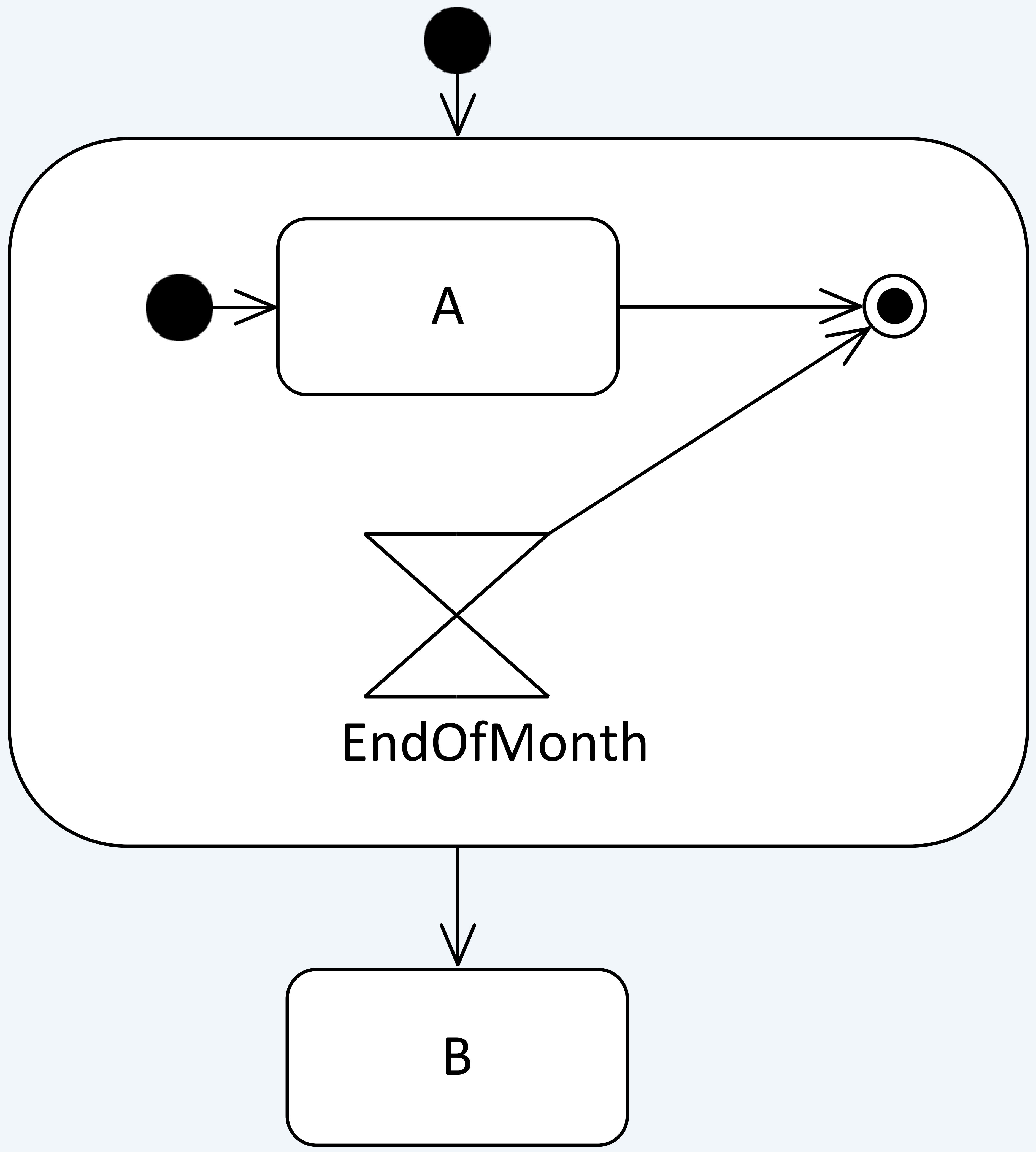




Example: Asynchronous event



Example: Asynchronous time event



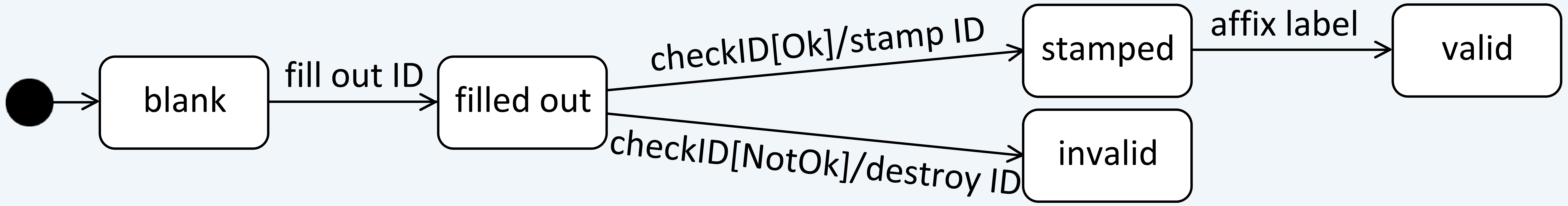
Activity Diagram
An Example

Christian Huemer und Marion Scholz

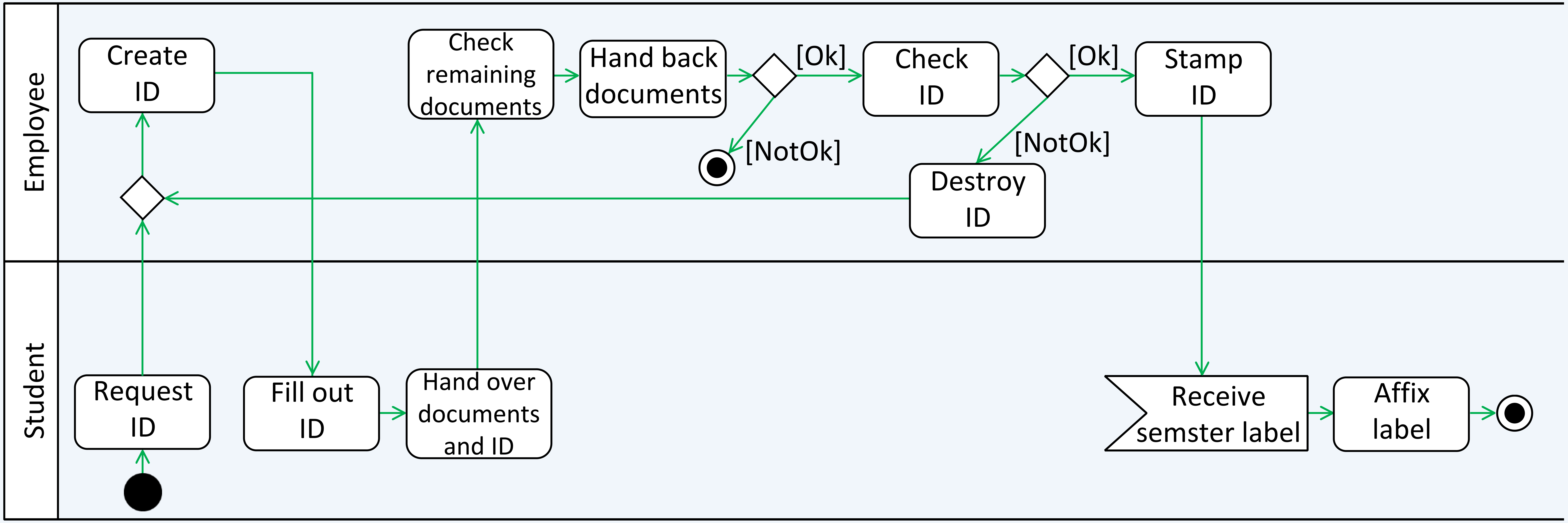
Presented by Nicholas Bzowski

Student ID Card (1/2)

- State diagram:

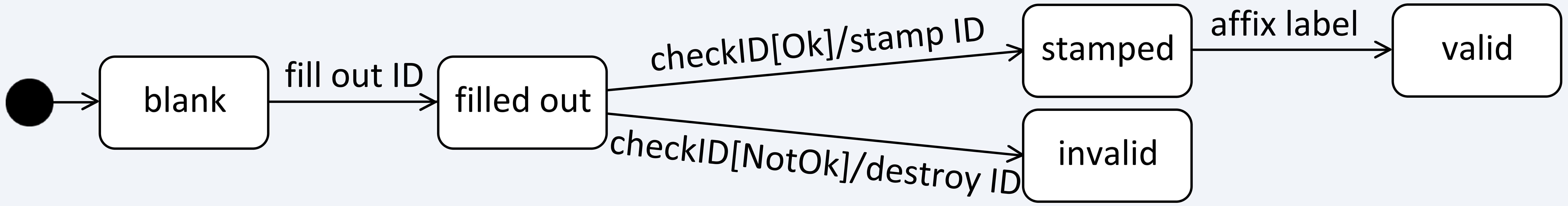


- Activity diagram - control flow:

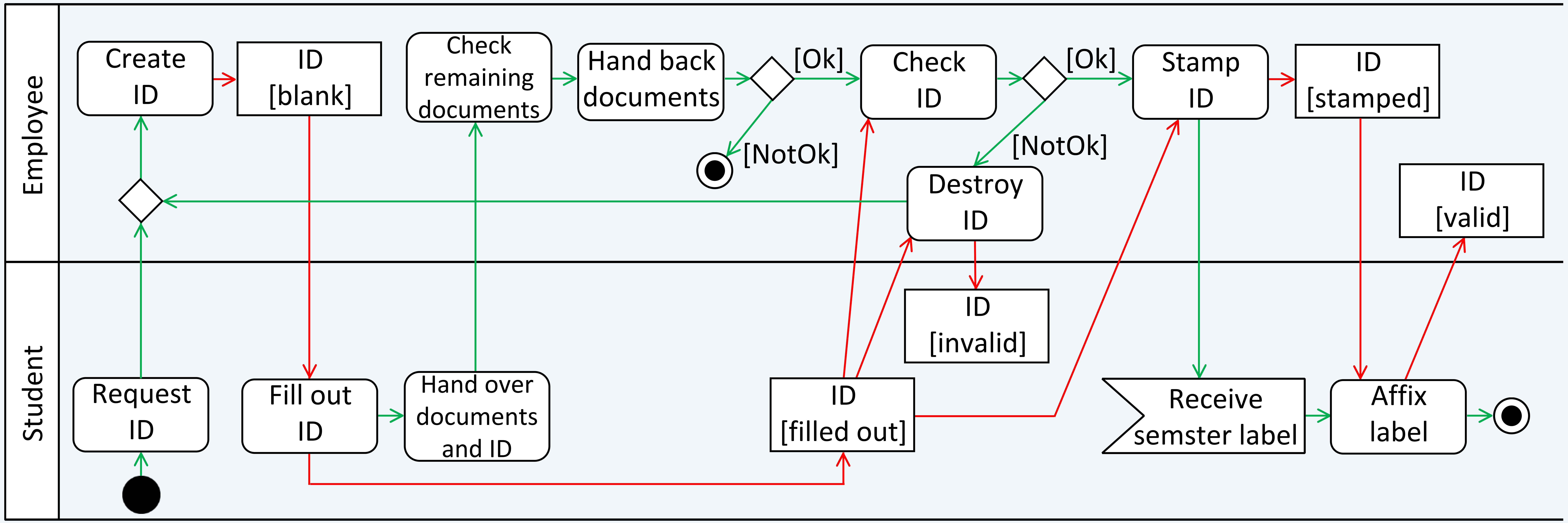


Student ID Card (2/2)

- State diagram:



- Control flow (green) and object flow (red) in one diagram



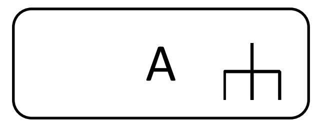
Activity Diagram

The Nesting of Activities and The Handling of Exceptions

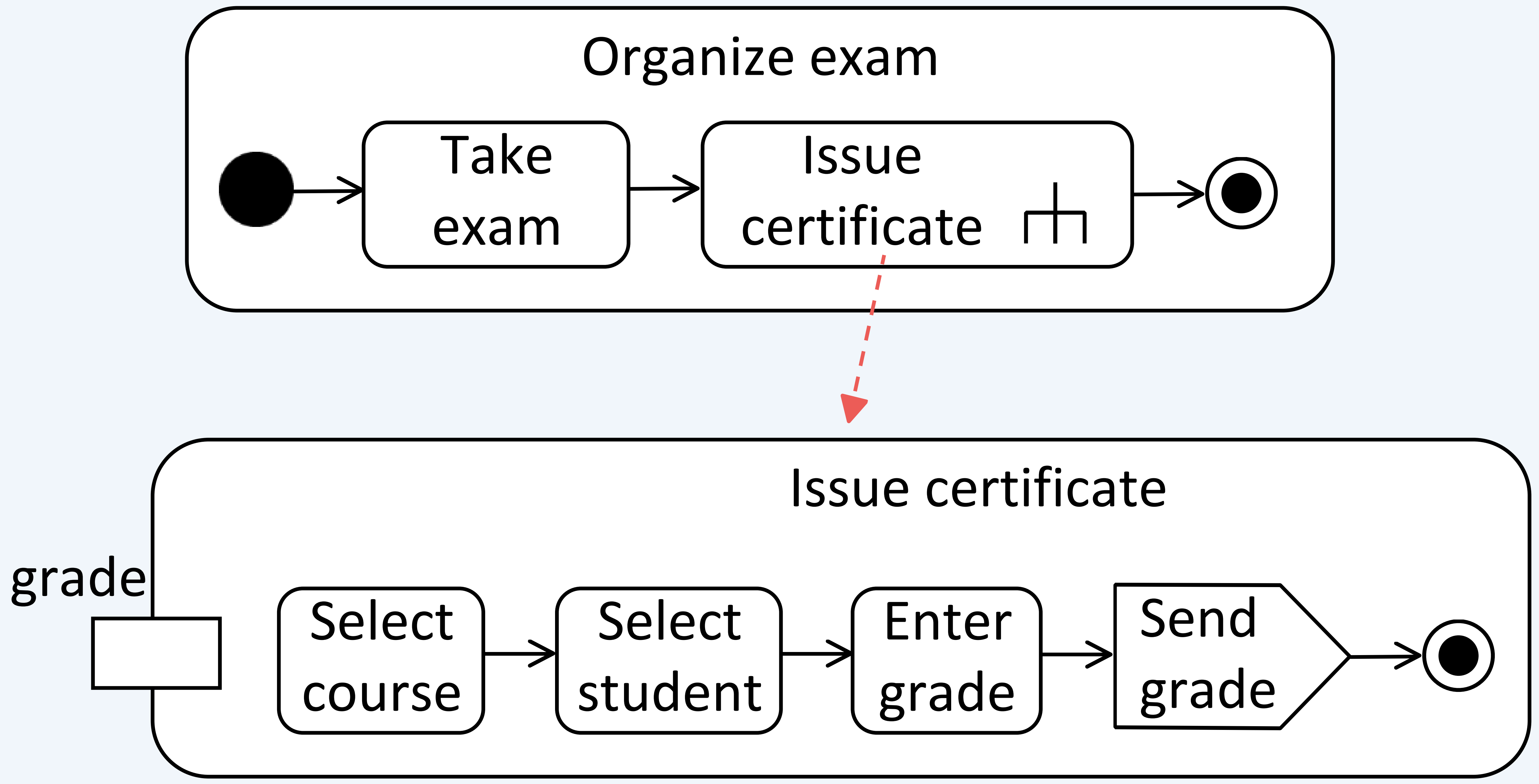
Christian Huemer und Marion Scholz

Presented by Nicholas Bzowski

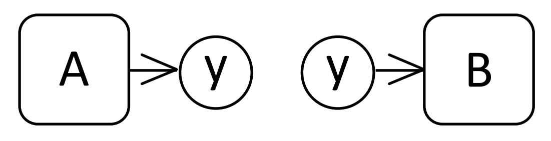
Activity Call



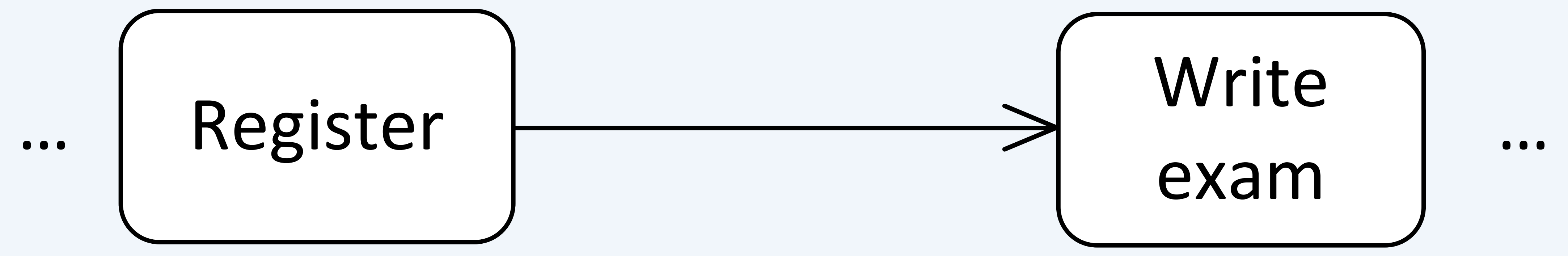
- Activities can also call activities
- Details are outsourced to a lower level
- Advantages :
  - Better readability
  - Reuse
- Notation:
  - An activity is called in an action



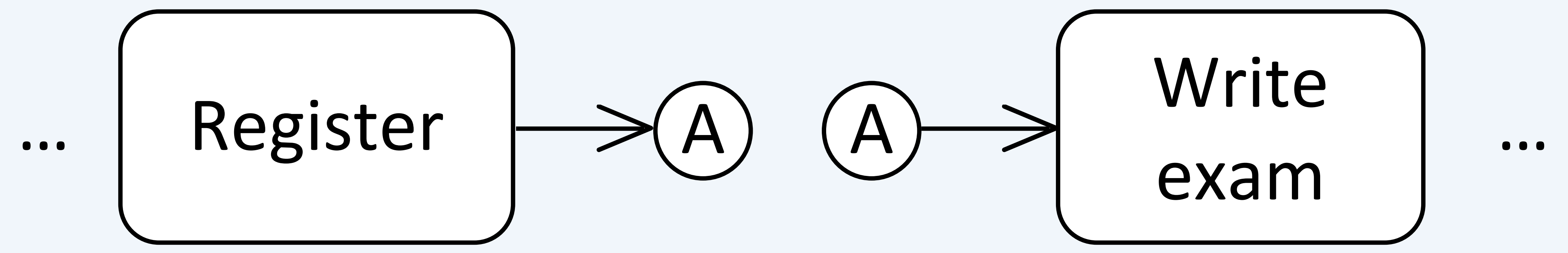
Connector



- Useful if two connected actions are far apart in the diagram
- Without connector:

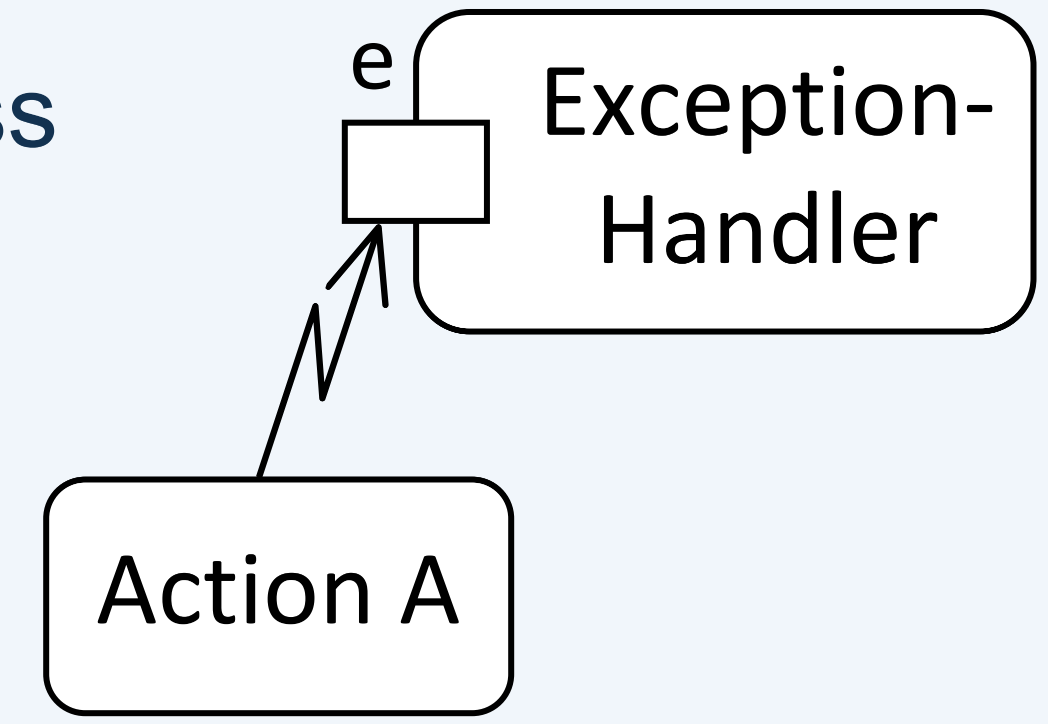


- With connector:

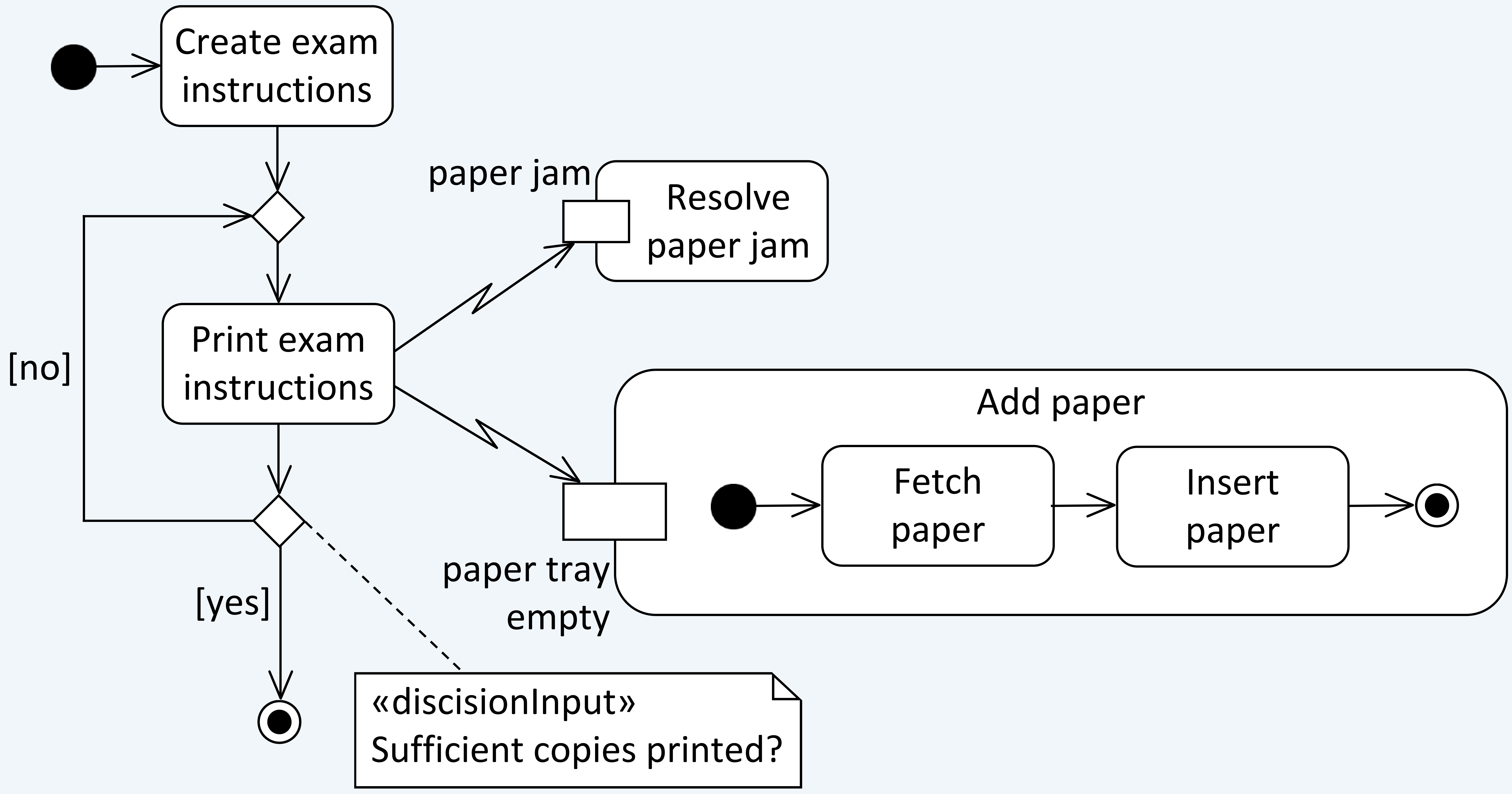


Exception Handler (1/3)

- **Predefined exceptions** (e.g. division by 0)
- Defines how the system should react to a specific error
- The exception handling node **substitutes** the “protected” node and has no outgoing control or object flows
- If error **e** occurs...
  - All tokens in **`Action A`** are deleted
  - The **`Exception-Handler`** is activated
  - The **`Exception-Handler`** is executed instead of **`Action A`**
  - Afterwards, the process continues as normal



Exception Handling – Exception Handler (2/3) Ex.

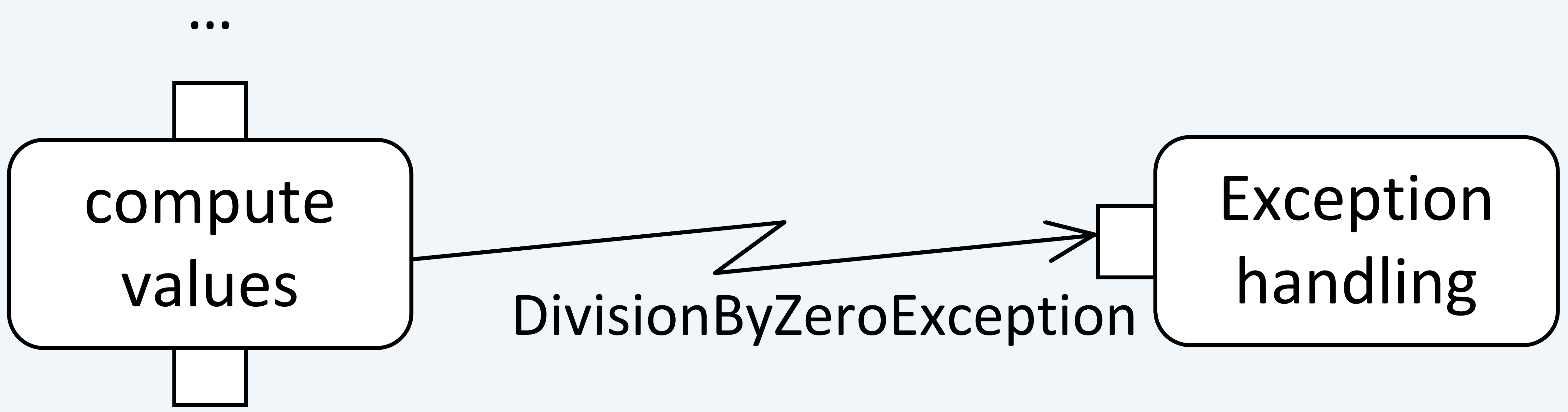


Exception Handling – Exception Handler (3/3)

- If there is no exception handling for an exception type, the affected action is terminated and the exception is propagated to the outward

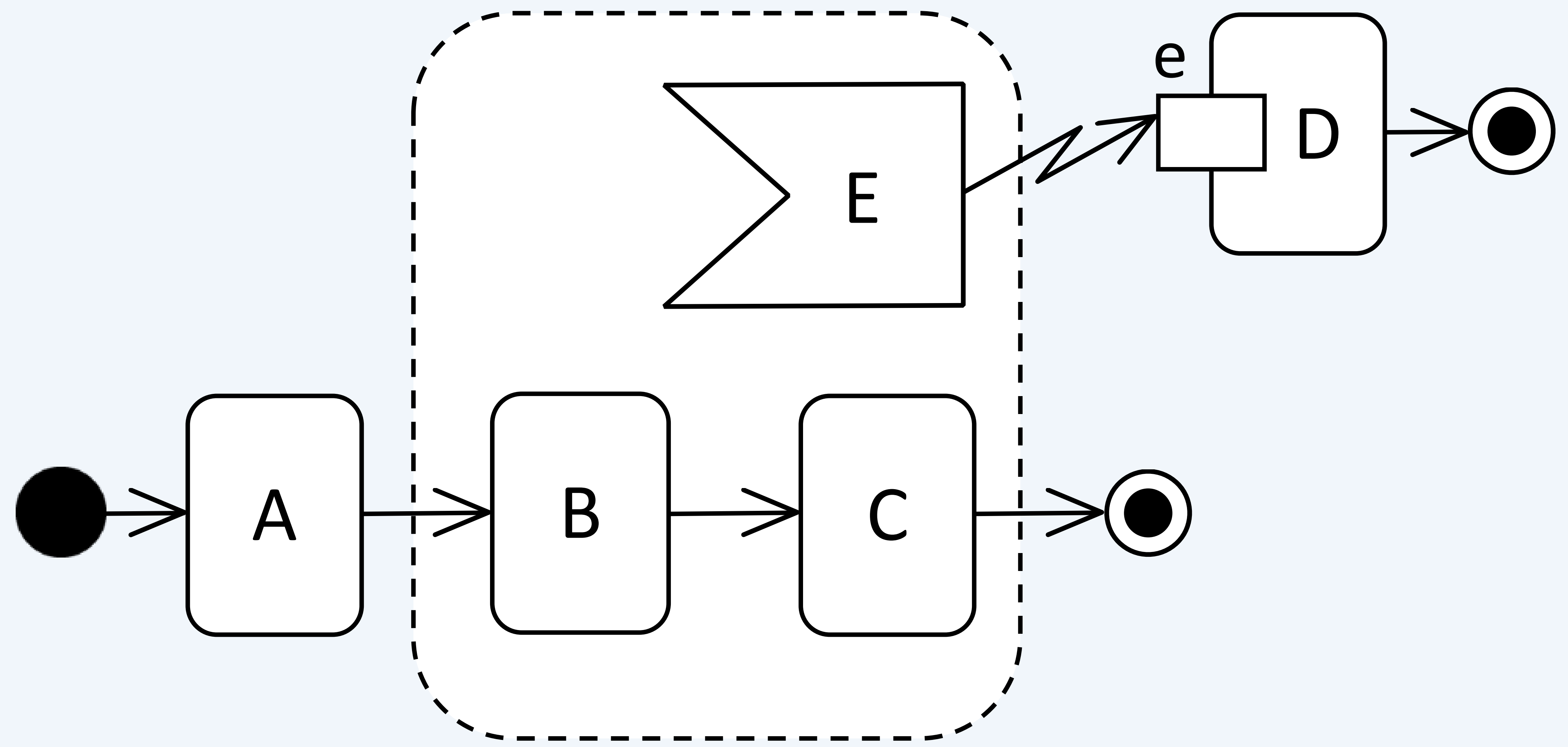
- Ex.:

```
try {
    // compute values
} catch (DivisionByZeroException) {
    // exception handling
}
```

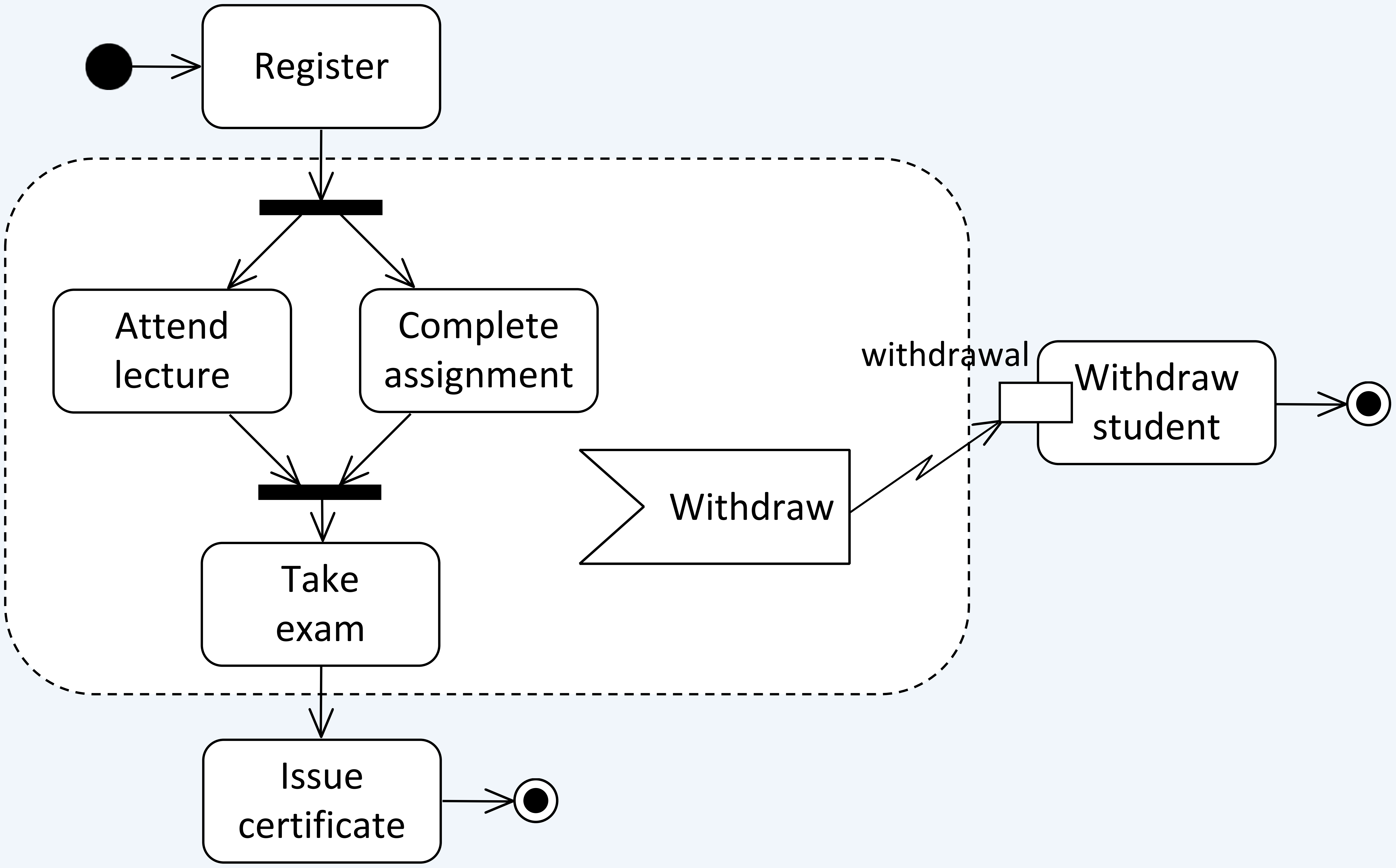


Exception Handling – Interruptible activity region (1/2)

- Includes 1-n actions whose execution is terminated immediately if a certain event occurs
- If the event **E** occurs during the execution of action **B** or action **C**
  - Exception handling is activated
  - All control tokens within the interruption area (i.e. in B and C) are deleted
  - **D** is activated and executed



Exception Handling – Interruptible activity region(2/2)


Register
Attend lecture
Complete assignment
withdrawal
Withdraw student
Withdraw
Take exam
Issue certificate