



# OO Analysis and Design with UML 2 and UP

---

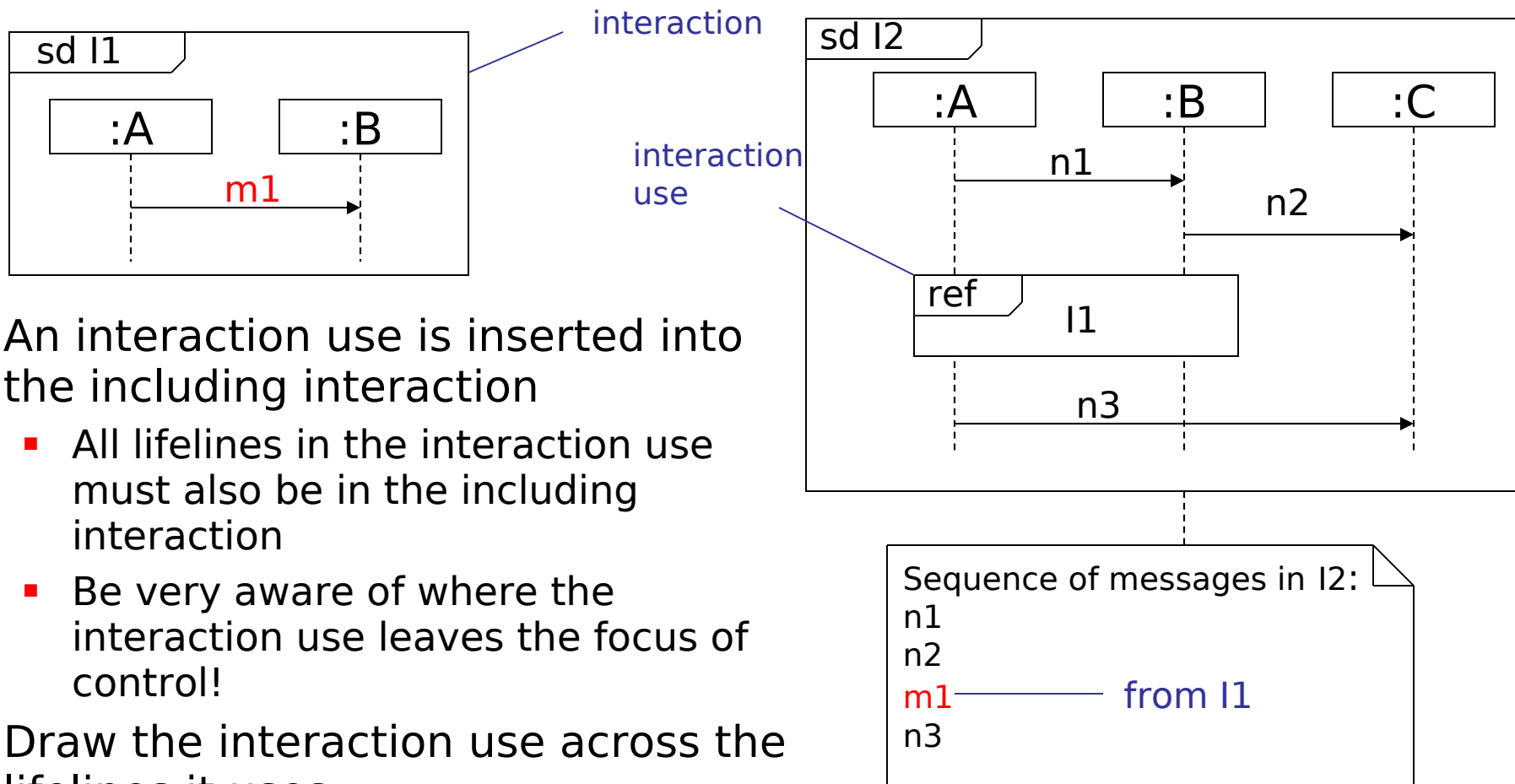
Dr. Jim Arlow,  
Zuhlke Engineering Limited

# Analysis - advanced use case realization



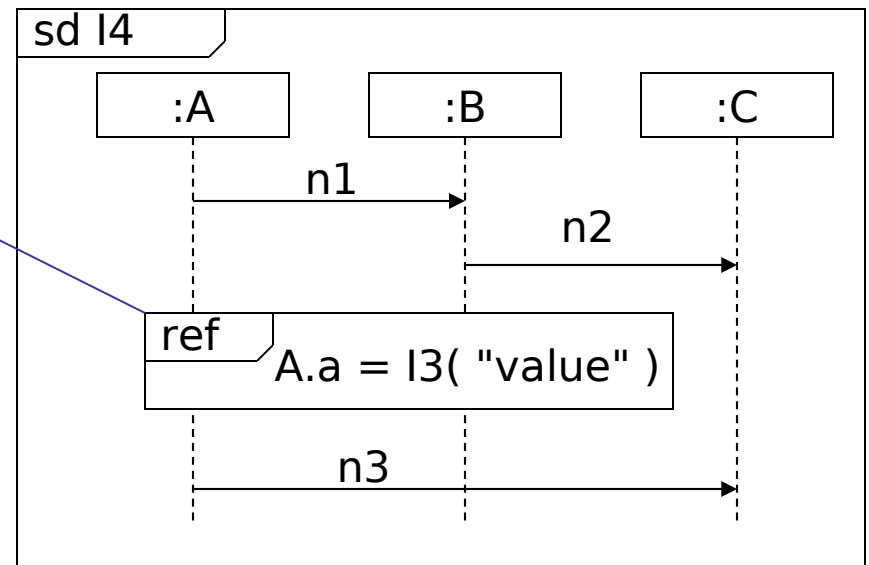
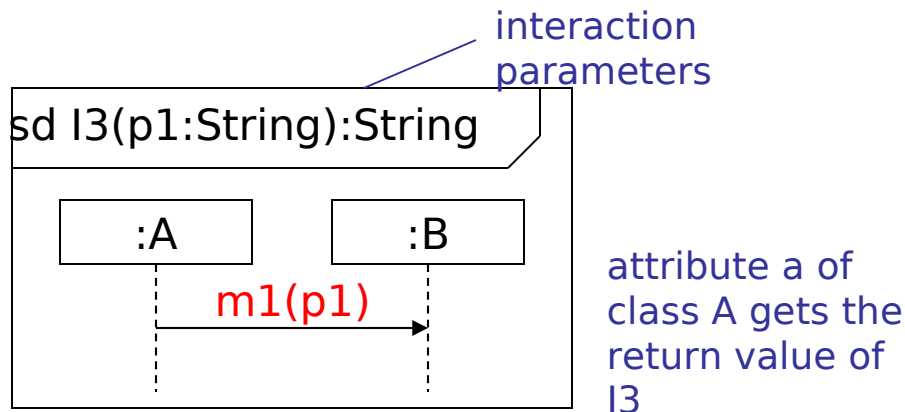
---

# Interaction occurrences



- An interaction use is inserted into the including interaction
  - All lifelines in the interaction use must also be in the including interaction
  - Be very aware of where the interaction use leaves the focus of control!
- Draw the interaction use across the lifelines it uses

# Parameters



Sequence of messages in I4:

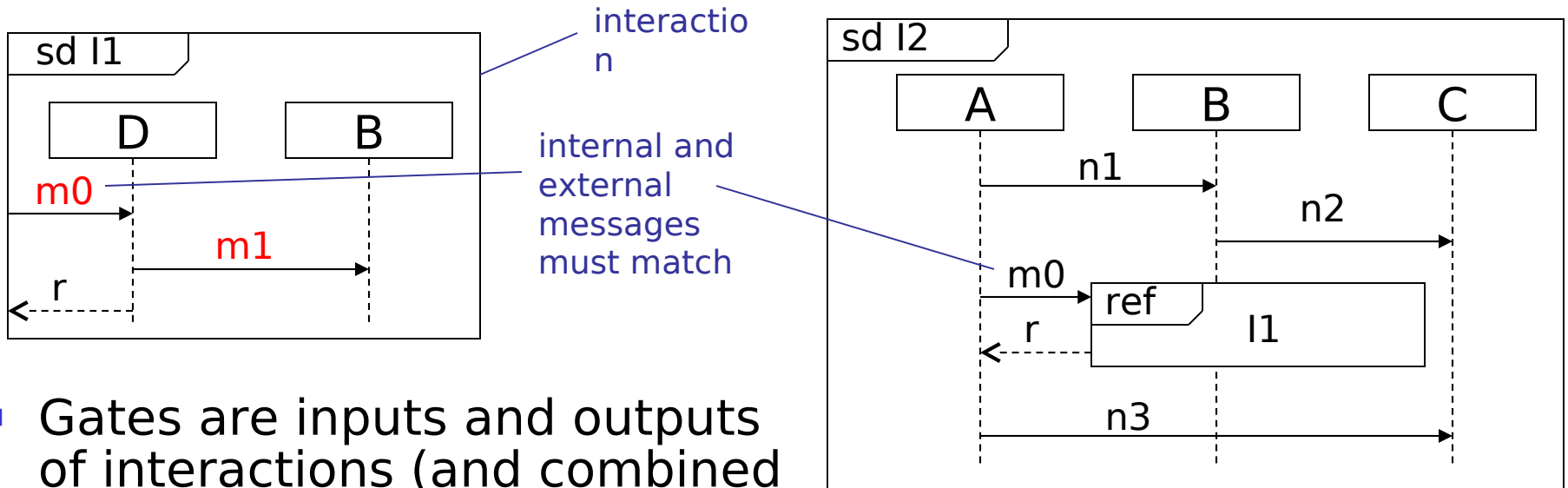
n1

n2

m1( "someValue" ) (from I1)

n3

# Gates



- Gates are inputs and outputs of interactions (and combined fragments – see next slide)
  - Provide connection points that relate messages inside an occurrence or fragment to messages outside it

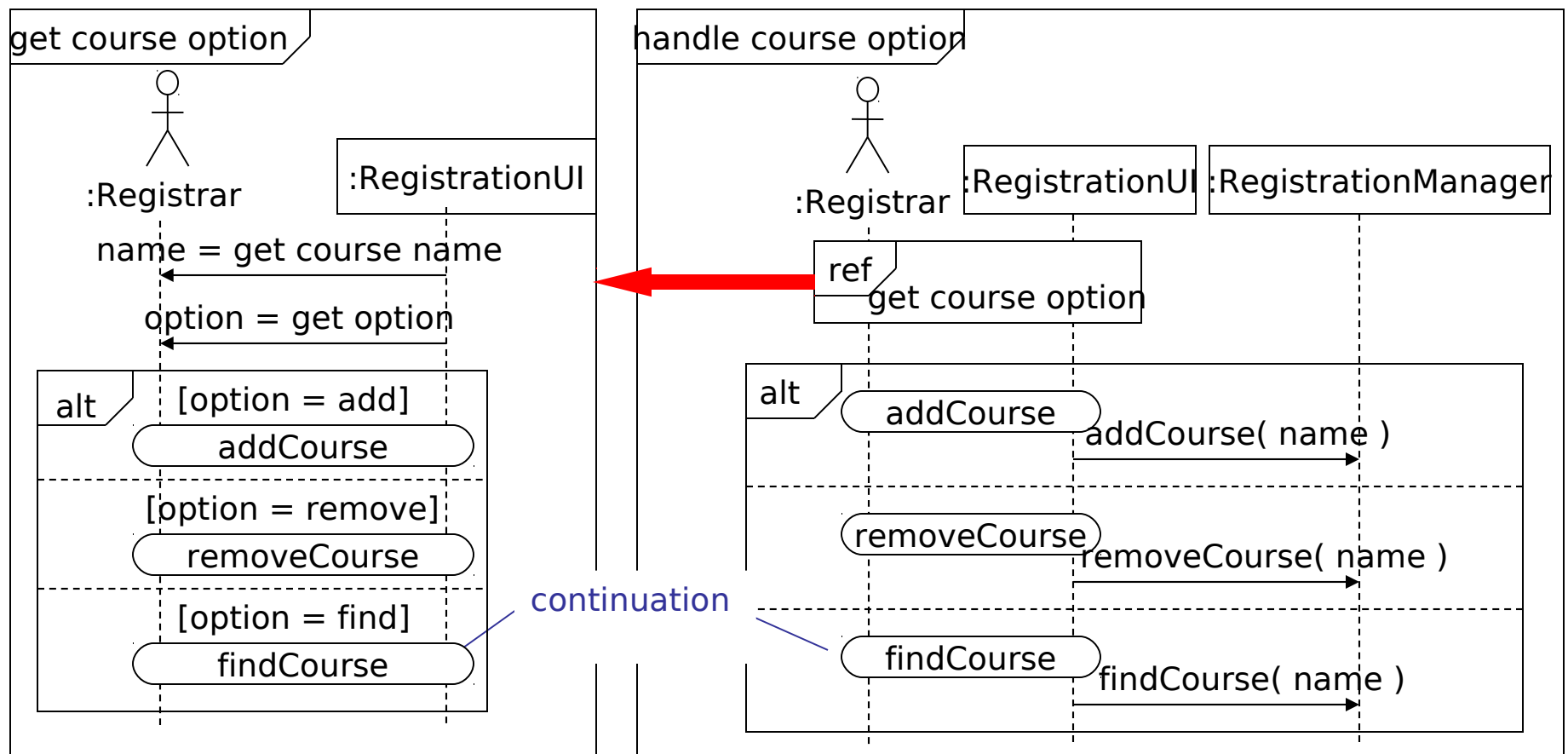
Sequence of messages in I2:

```
n1
n2
m0
m1
n3
```

from I1

# Continuations

- Continuations allow an interaction fragment to terminate in such a way that it can be continued by another fragment





# Summary

---

- In this section we have looked at:
  - Interaction occurrences
  - Parameters
  - Gates
  - Continuations

# Analysis - activity diagrams



---



# What are activity diagrams?

---

- Activity diagrams are "OO flowcharts"!
- They allow us to model a process as a collection of nodes and edges between those nodes
- Use activity diagrams to model the behavior of:
  - use cases
  - classes
  - interfaces
  - components
  - collaborations
  - operations and methods
  - business processes



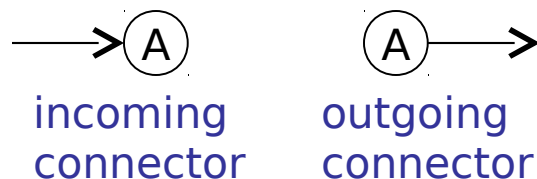
# Activities

---

- Activities are networks of nodes connected by edges
- There are three categories of node:
  - Action nodes - represent discrete units of work that are atomic within the activity
  - Control nodes - control the flow through the activity
  - Object nodes - represent the flow of objects around the activity
- Edges represent flow through the activity
- There are two categories of edge:
  - Control flows - represent the flow of control through the activity
  - Object flows - represent the flow of objects through the activity

# Activity diagram syntax

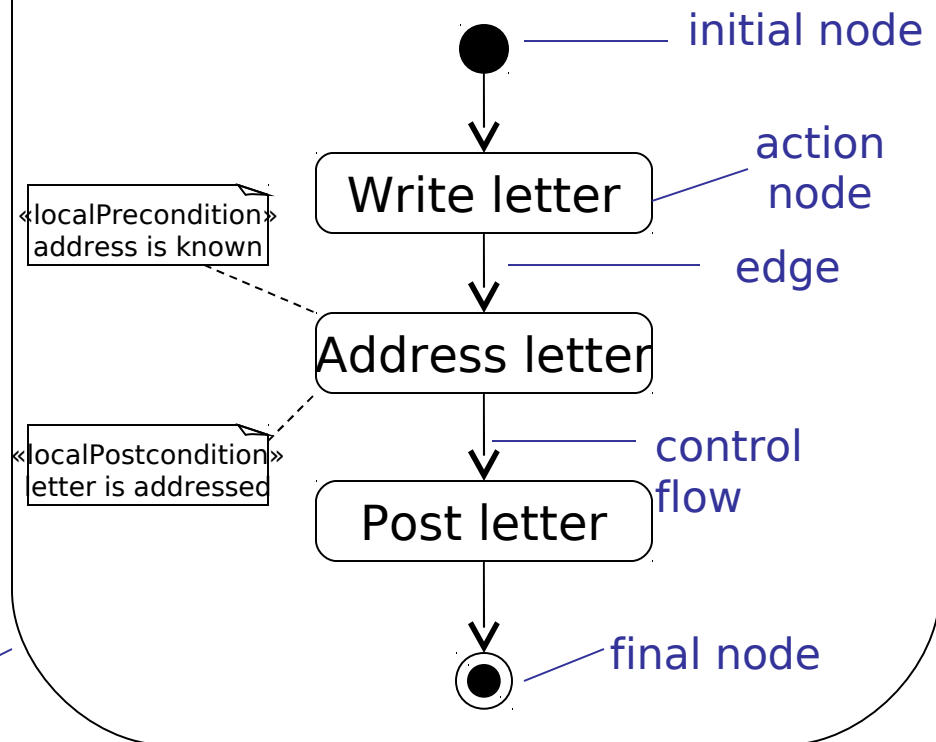
- Activities are networks of *nodes* connected by *edges*
  - The control flow is a type of edge
- Activities usually start in an *initial node* and terminate in a *final node*
- Activities can have preconditions and postconditions
- When an action node finishes, it emits a token that may traverse an edge to trigger the next action
  - This is sometimes known as a *transition*
- You can break an edge using connectors:



activity

## Send letter

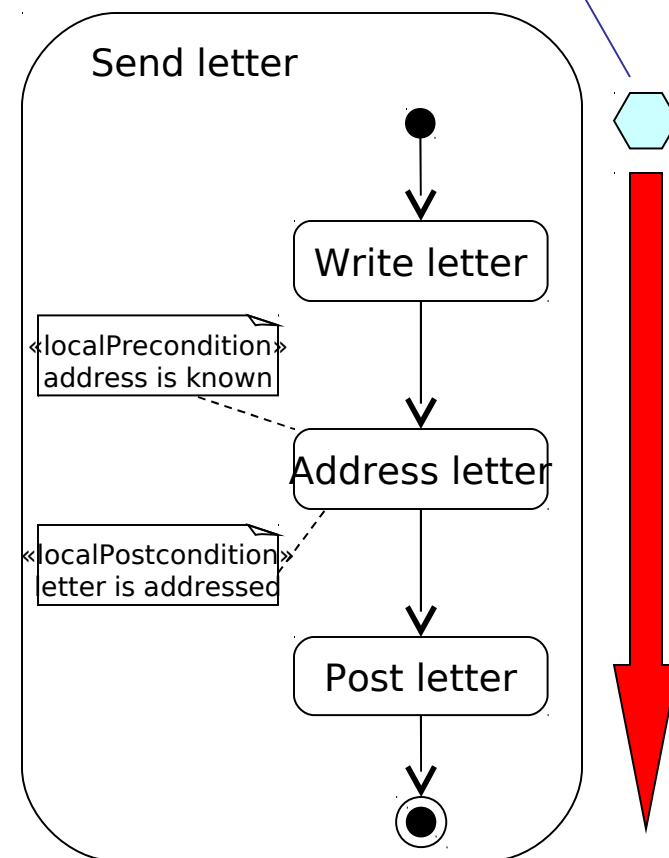
precondition: know topic for letter  
postcondition: letter sent to address



# Activity diagram semantics

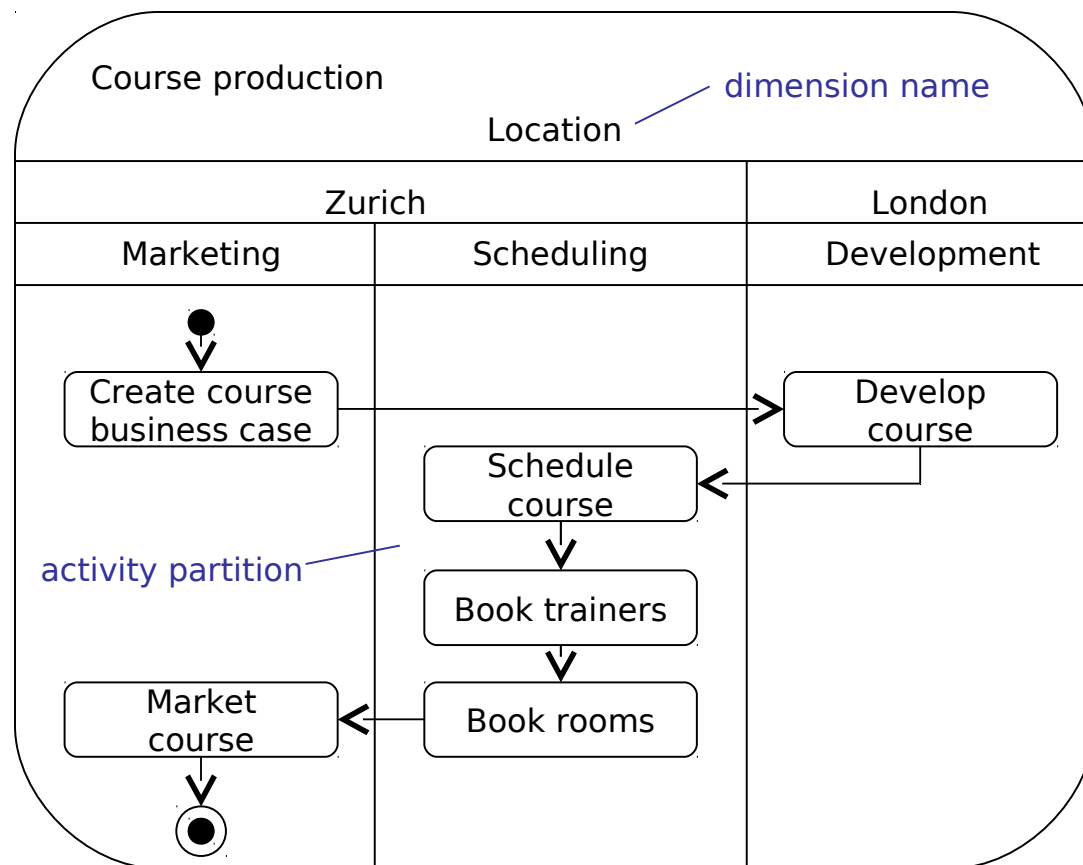
- The *token game*
  - Token - an object, some data or a focus of control
  - Imagine tokens flowing around the activity diagram
- Tokens traverse from a source node to a target node via an edge
  - The source node, edge and target node may all have constraints controlling the movement of tokens
  - All constraints *must* be satisfied before the token can make the traversal
- A node executes when:
  - It has tokens on all of its input edges AND these tokens satisfy predefined conditions (see later)
- When a node starts to execute it takes tokens off its input edges
- When a node has finished executing it offers tokens on its output edges

imaginary flow of control token



# Activity partitions

- Each activity partition represents a high-level grouping of a set of related actions
  - Partitions can be hierarchical
  - Partitions can be vertical, horizontal or both
- Partitions can refer to many different things e.g. business organisations, classes, components and so on
- If partitions can't be shown clearly using parallel lines, put their name in brackets directly above the name of the activities



(London::Marketing)  
Market product

(p1, p2)  
SomeAction

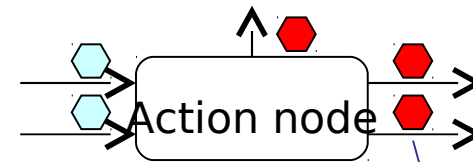
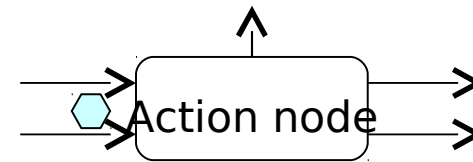
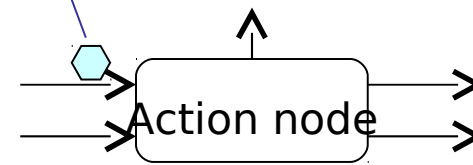
nested partitions

multiple partitions

# Action nodes


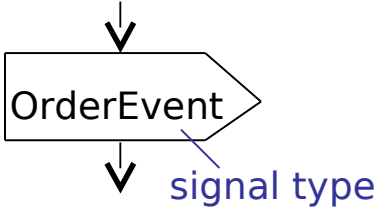
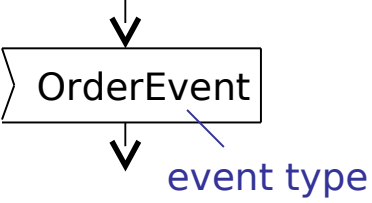
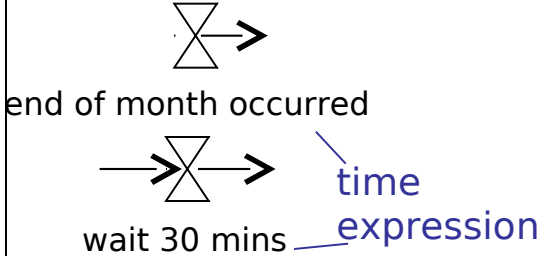
- Action nodes offer a token on *all* of their output edges when:
  - There is a token *simultaneously* on each input edge
  - The input tokens satisfy all preconditions specified by the node
- Action nodes:
  - Perform a logical AND on their input edges when they begin to execute
  - Perform an implicit fork on their output edges when they have finished executing

input token



output token

# Types of action node

action node syntax	action node semantics
	<p>Call action - invokes an activity, a behavior or an operation. The most common type of action node.</p> <p>See next slide for details.</p>
	<p>Send signal action - sends a signal asynchronously. The sender <i>does not</i> wait for confirmation of signal receipt.</p> <p>It may accept input parameters to create the signal</p>
	<p>Accept event action - waits for events detected by its owning object and offers the event on its output edge. Is enabled when it gets a token on its input edge. If there is <i>no</i> input edge it starts when its containing activity starts and is <i>always</i> enabled.</p>
	<p>Accept time event action - waits for a set amount of time. Generates time events according to it's time expression.</p>

# Call action node syntax

- The most common type of node
- Call action nodes may invoke:
  - an activity
  - a behavior
  - an operation
- They may contain code fragments in a specific programming language
  - The keyword 'self' refers to the context of the activity that owns the action

Raise Order 

call an activity  
(note the rake icon)

Close Order

call a behavior

getBalance():double  
(Account::)

operation name  
class name  
(optional)

Get Balance  
(Account::getBalance():double)

node name  
operation name  
(optional)


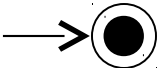
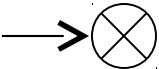
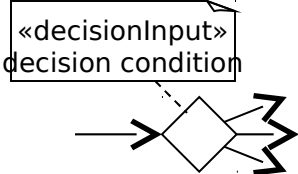
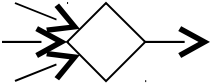
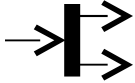
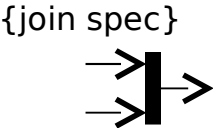
call an  
operation

```
if self.balance <= 0:
    self.status = INCREMENT
else
    self.status = OVERDRAWN
```

programming  
language  
(e.g.  
Python)



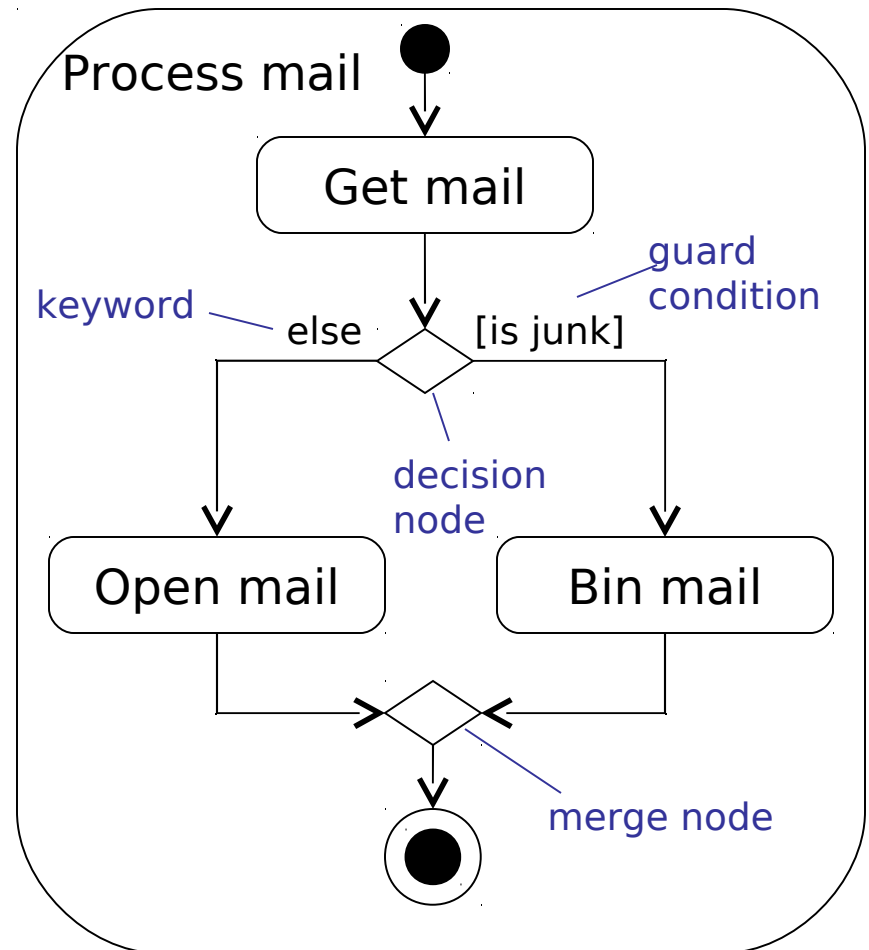
# Control nodes

control node syntax	control node semantics	
	Initial node - indicates where the flow starts when an activity is invoked	
	Activity final node - terminates an activity	
	Flow final node - terminates a specific flow within an activity. The other flows are unaffected	
	Decision node- guard conditions on the output edges select one of the for traversal May optionally have inputs defined by a «decisionInput»	
	Merge node - selects <i>one</i> of its input edges	
	Fork node - splits the flow into multiple concurrent flows	
	Join node - synchronizes multiple concurrent flows May optionally have a join specification to modify its semantics	

 Final nodes  
See examples on next two slides

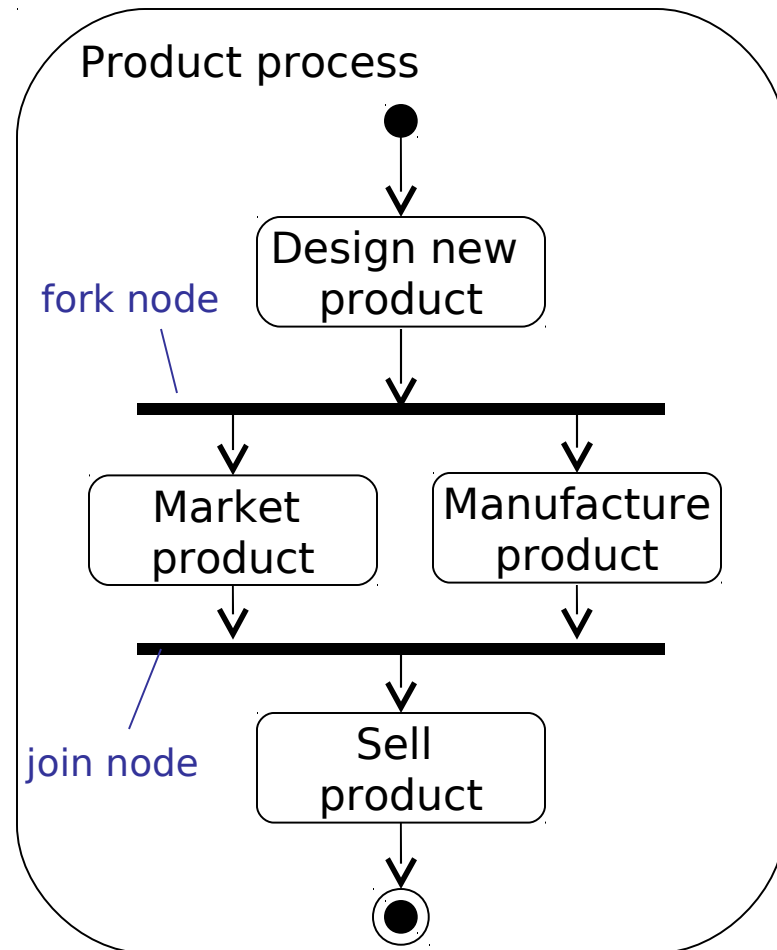
# Decision and merge nodes

- A decision node is a control node that has one input edge and two or more alternate output edges
  - Each edge out of the decision is protected by a *guard condition*
  - guard conditions must be mutually exclusive
  - The edge can be taken if and only if the guard condition evaluates to true
  - The keyword *else* specifies the path that is taken if *none* of the guard conditions are true
- A merge node accepts one of several alternate flows
  - It has two or more input edges and exactly one output edge



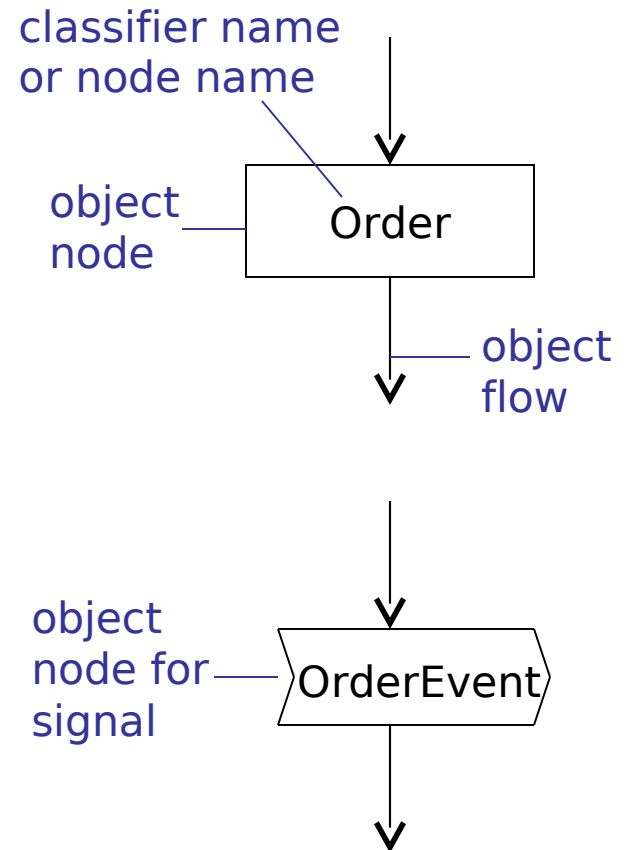
# Fork and join nodes - concurrency

- Forks nodes model concurrent flows of work
  - Tokens on the single input edge are replicated at the multiple output edges
- Join nodes synchronize two or more concurrent flows
  - Joins have two or more incoming edges and exactly one outgoing edge
  - A token is offered on the outgoing edge when there are tokens on *all* the incoming edges i.e. when the concurrent flows of work have all finished



# Object nodes

- Object nodes indicate that instances of a particular classifier may be available
  - If no classifier is specified, then the object node can hold any type of instance
- Multiple tokens can reside in an object node *at the same time*
  - The upper bound defines the maximum number of tokens (infinity is the default)
- Tokens are presented to the single output edge according to an ordering:
  - FIFO – first in, first out (the default)
  - LIFO – last in, first out
  - Modeler defined – a selection criterion is specified for the object node



# Object node syntax

Object nodes have a flexible syntax. You may show:

- upper bounds
- ordering
- sets of objects
- selection criteria
- object in state

Order

order objects may be available

Order

zero to 12 Order objects may be available

{upperBound = 12}

Order

last Order object in is the first out (FIFO is the default)

{ordering = LIFO}

Set of Order

sets of Order objects may be available

«selection»  
monthRaised = "Dec"

Order

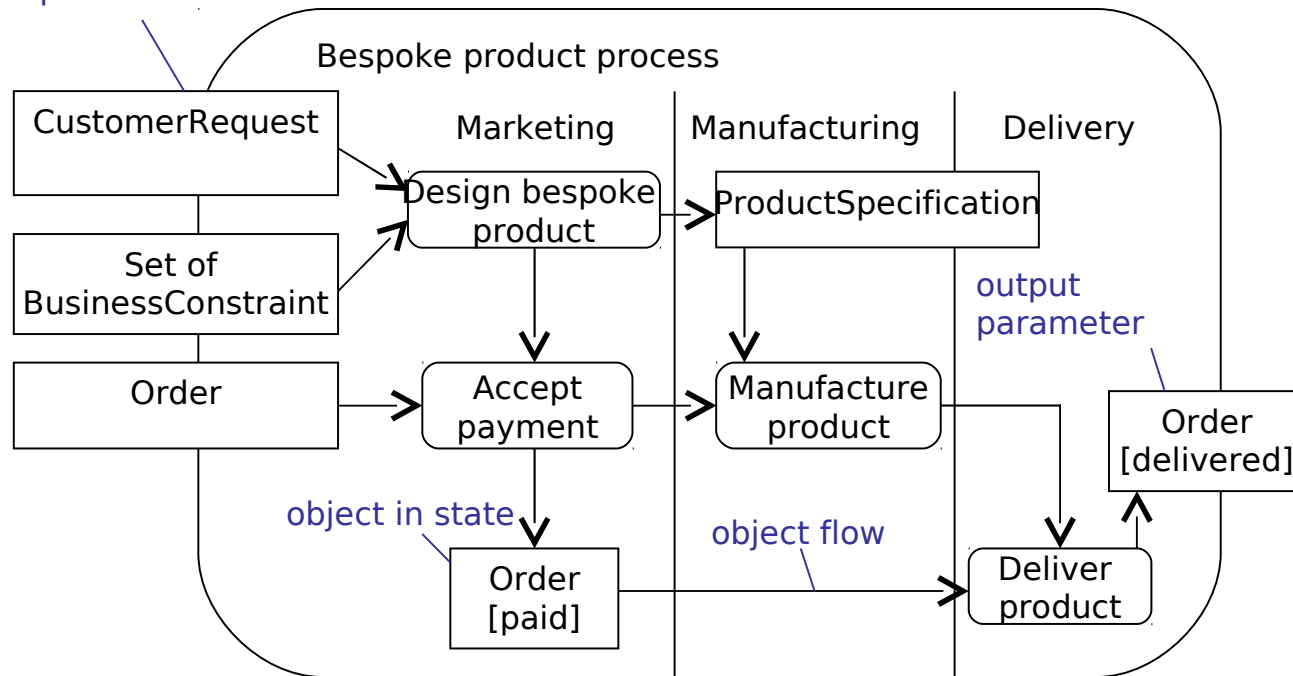
Order objects raised in December may be available

Order  
[open]

select Order objects in the open state

# Activity parameters

input parameter



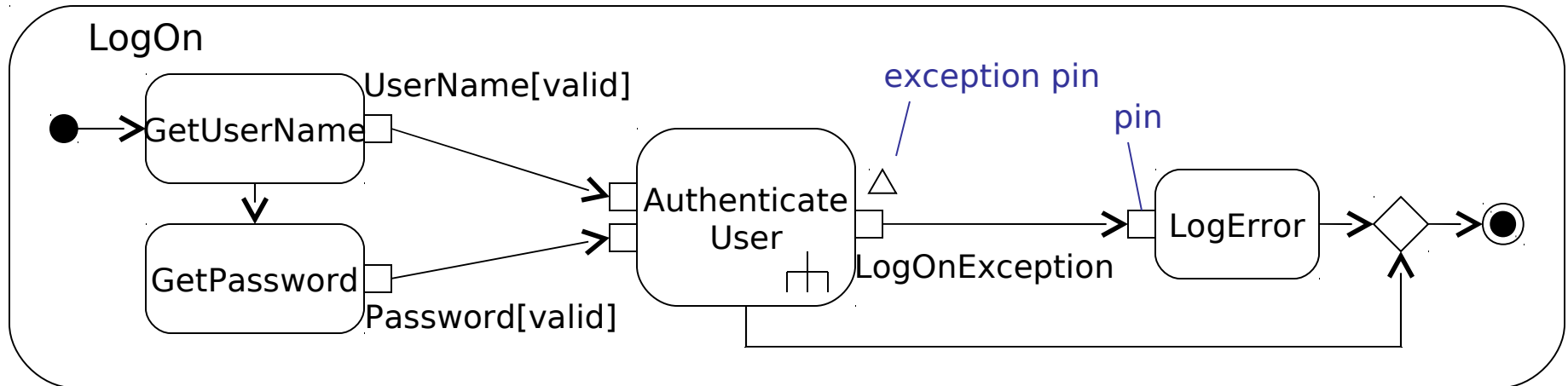
output parameter

object in state

object flow

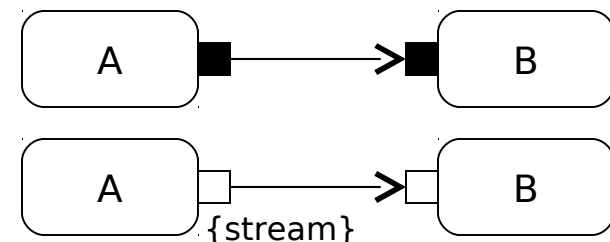
- Object nodes can provide input and output parameters to activities
  - Input parameters have one or more output object flows into the activity
  - Output parameters have one or more input object flows out of the activity
- Draw the object node overlapping the activity boundary

# Pins



- Pins are object nodes for inputs to, and outputs from, actions
  - Same syntax as object nodes
  - Input pins have exactly one input edge
  - Output pins have exactly one output edge
  - Exception pins are marked with an equilateral triangle
  - Streaming pins are filled in black or marked with {stream}

streaming - see notes





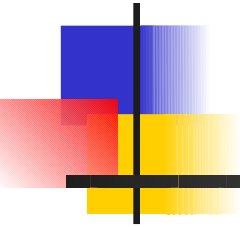
# Summary

---

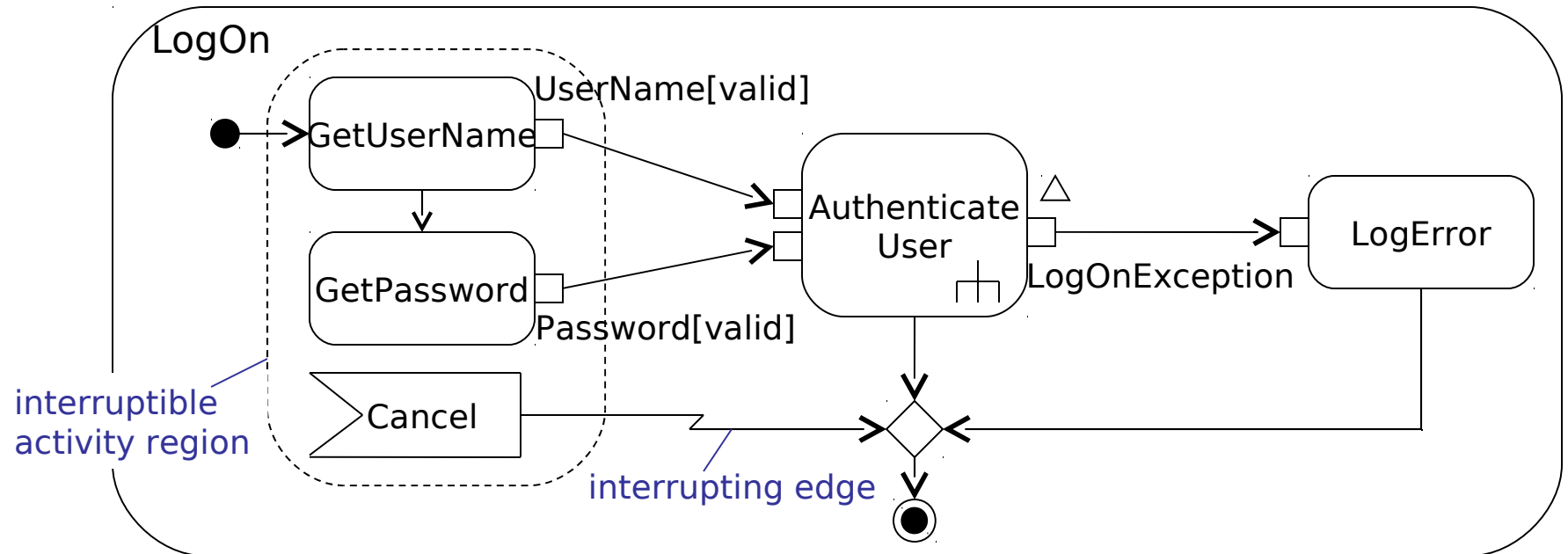
- We have seen how we can use activity diagrams to model flows of activities using:
  - Activities
    - Connectors
  - Activity partitions
  - Action nodes
    - Call action node
    - Send signal/accept event action node
    - Accept time event action node
  - Control nodes
    - decision and merge
    - fork and join
  - Object nodes
    - input and output parameters
    - pins



# Analysis - advanced activity diagrams



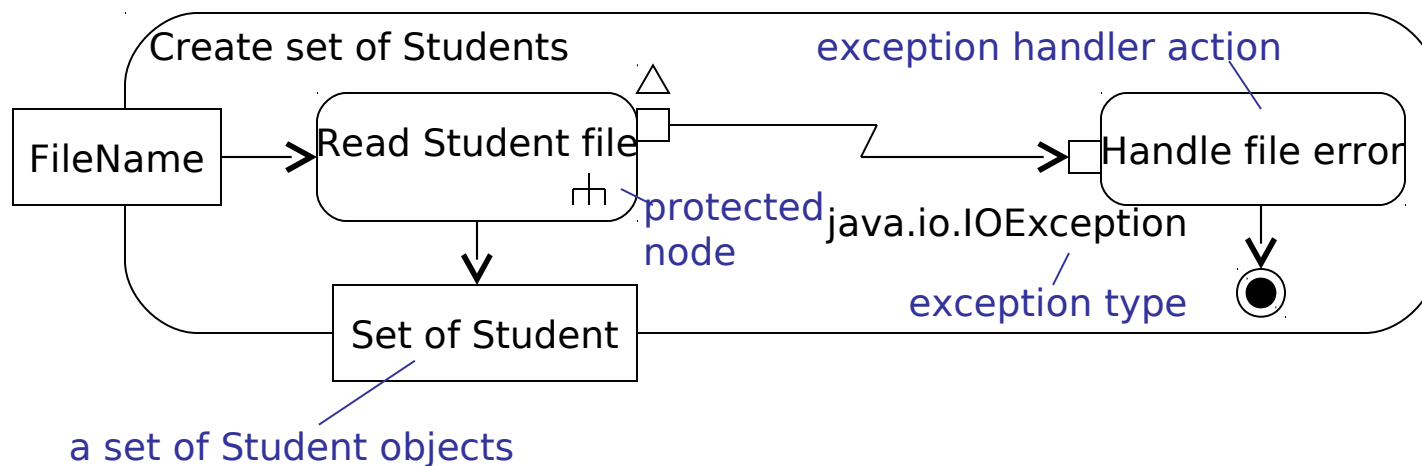
# Interruptible activity regions



- Interruptible activity regions may be interrupted when a token traverses an interrupting edge
  - All flows in the region are aborted
- Interrupting edges *must* cross the region boundary

→  
alternative notation

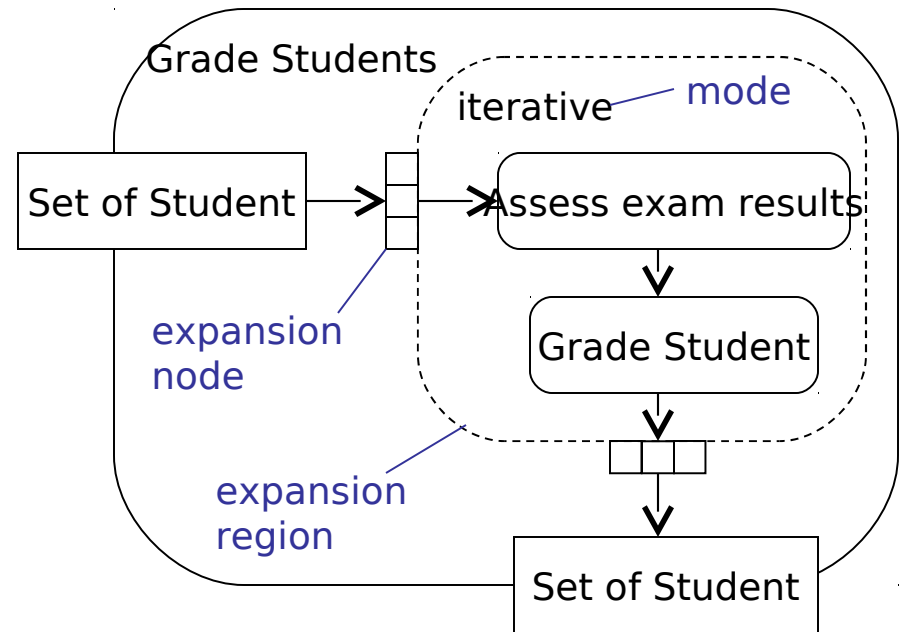
# Exception handling



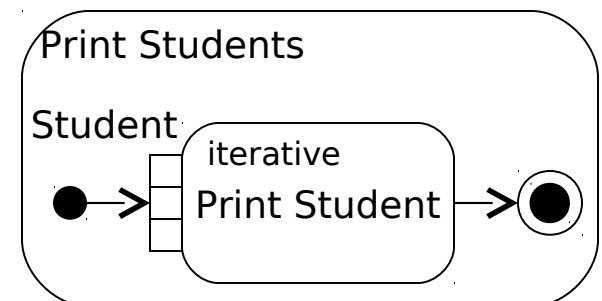
- Protected nodes have exception handlers:
  - When the exception object is raised in the protected node, flow is directed along an interrupting edge to the exception handler body

# Expansion nodes

- Expansion node - an object node that represents a collection of objects flowing into or out of an *expansion region*
  - Output collections *must* correspond to input collections in collection type and object type!
- The expansion region is executed once per input element according to the keyword:
  - iterative - process sequentially
  - parallel - process in parallel
  - stream - process a stream of input objects

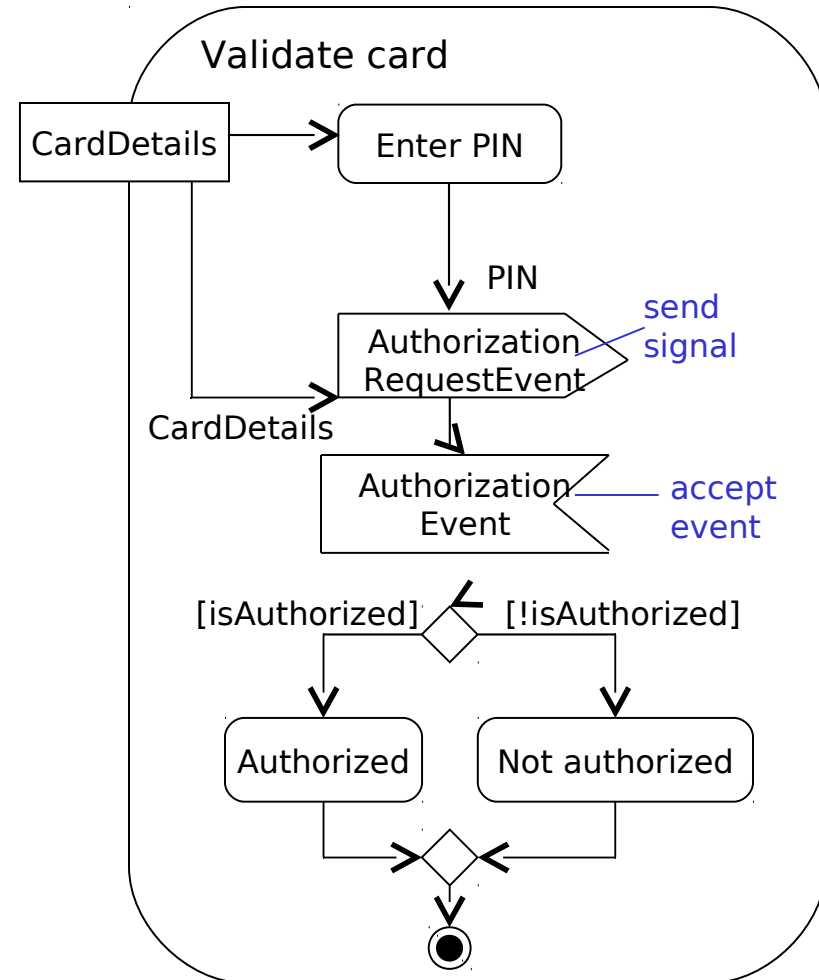
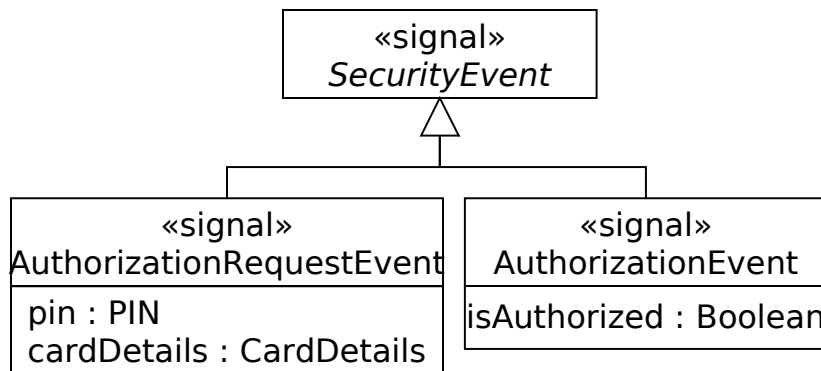


Expansion regions containing a single action - place the expansion node directly on the action



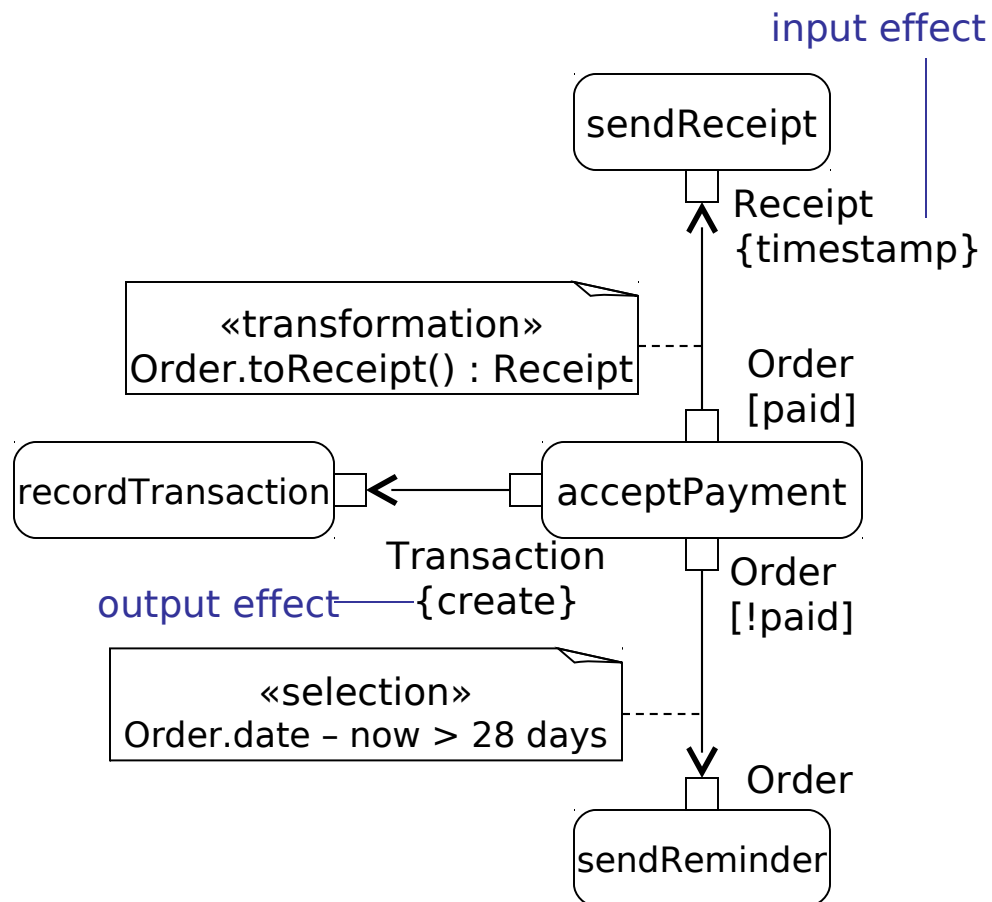
# Sending signals and accepting events

- Signals represent information passed asynchronously between objects
  - This information is modelled as attributes of a signal
  - A signal is a classifier stereotyped «signal»
- The accept event action asynchronously accepts event triggers which may be signals or other objects



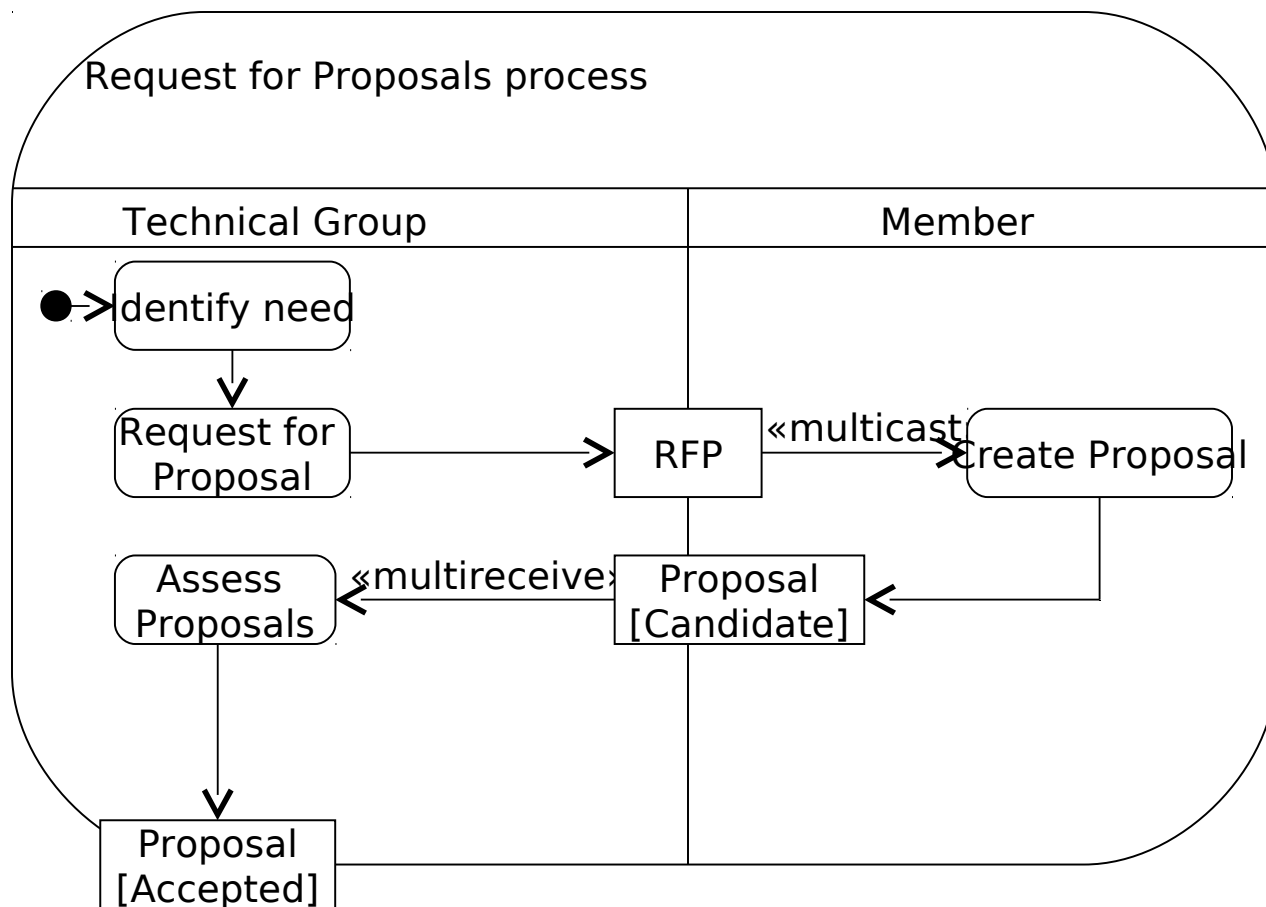
# Advanced object flow

- Input effect
  - Specifies the effect of the action on objects flowing into it
- Output effect
  - Specifies the effect of the action on objects flowing out of it
- «selection»
  - the flow to selects objects that meet a specific criterion
- «transformation»
  - An object is transformed by the object flow

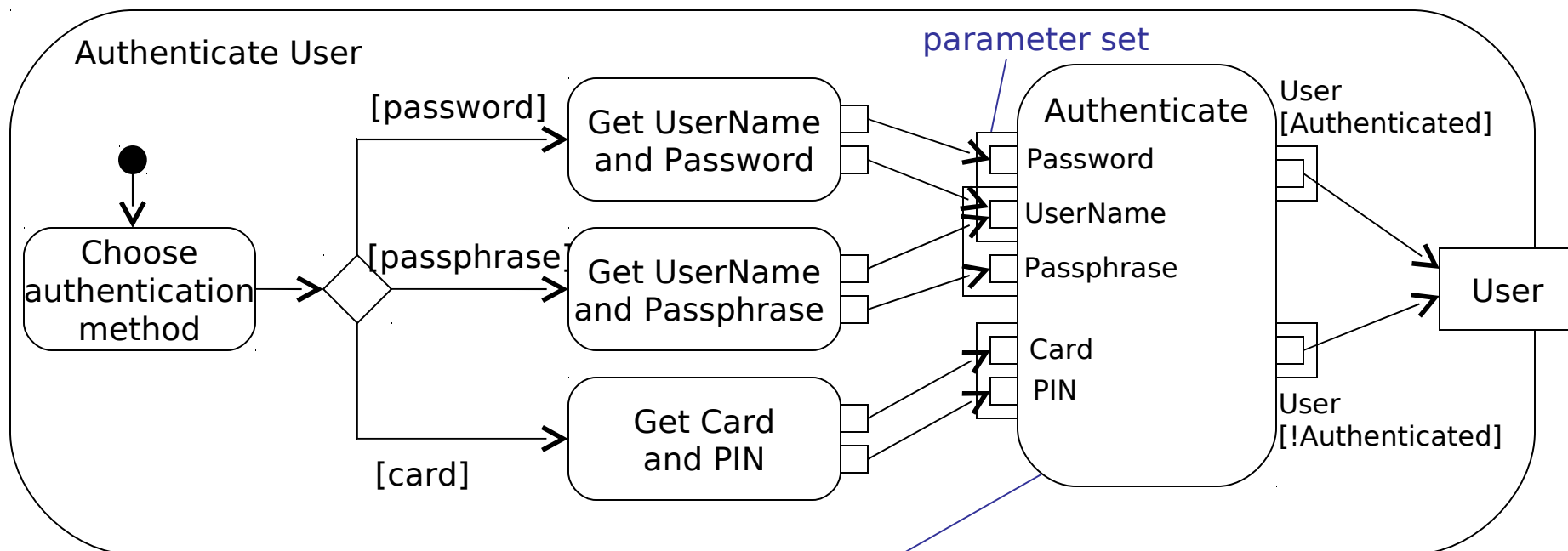


# Multicast and multireceive

- A «multicast» object flow sends an object to multiple receivers
- A «multireceive» object flow receives an object from multiple receivers



# Parameter sets



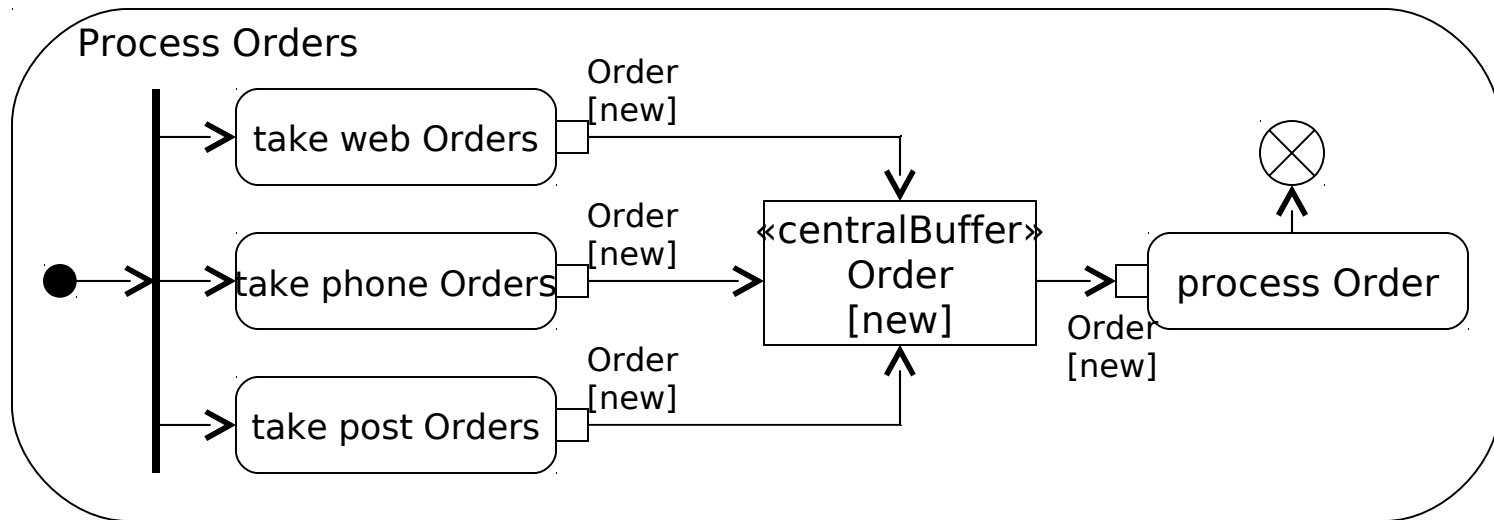
input condition: ( UserName AND Password ) XOR ( UserName AND Passphrase ) XOR ( Card AND PIN )

output: ( User [Authenticated] ) XOR ( User [!Authenticated] )

- Parameter sets provide alternative sets of input pins and output pins to an action
  - Only one input set and one output set may be chosen (XOR)



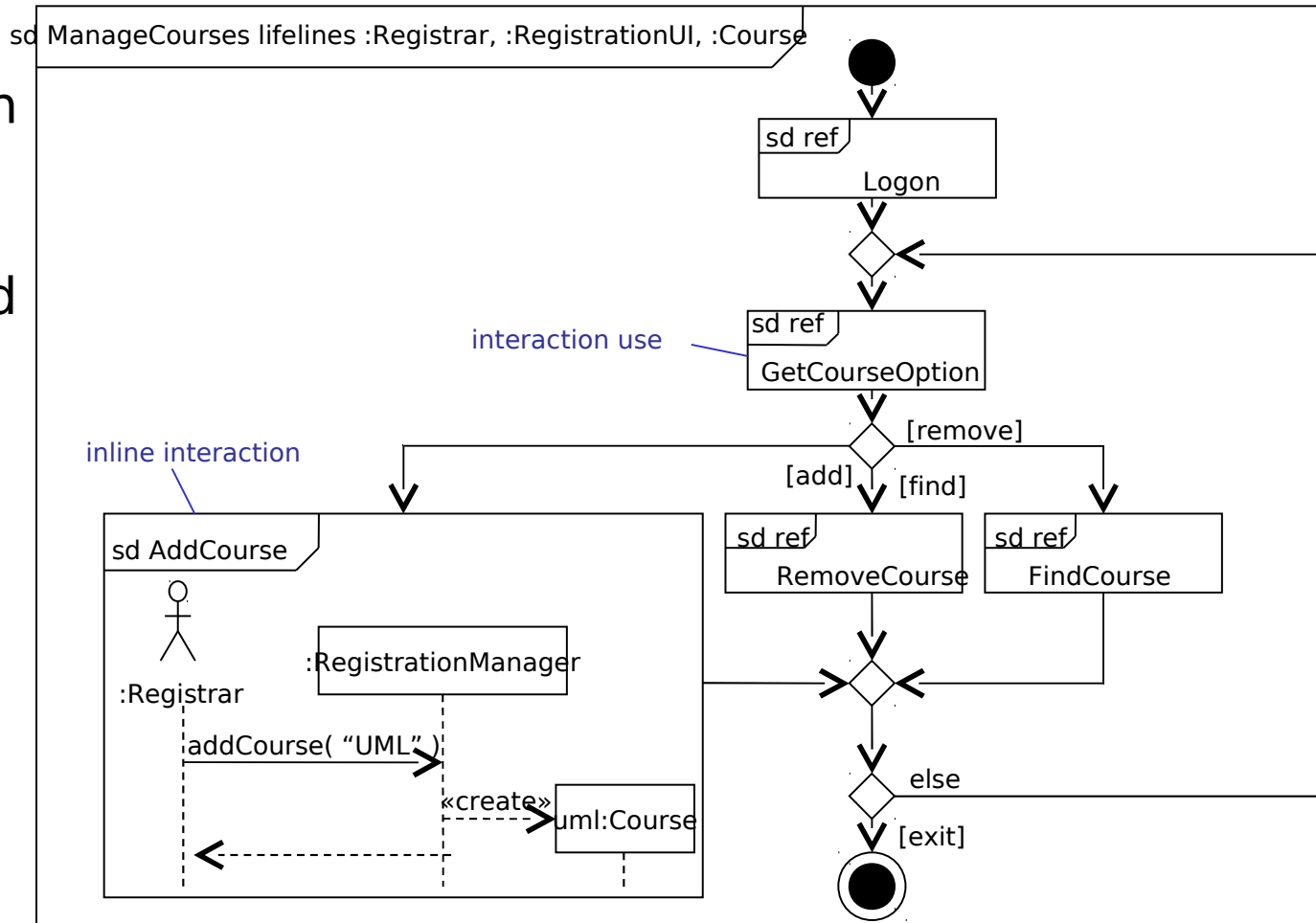
# «centralBuffer» node



- Central buffer nodes accept multiple upstream object flows
- They hold the objects until downstream nodes are ready for them

# Interaction overview diagrams

- Model the high level flow of control between interactions
- Show interactions and interaction occurrences
- Have activity diagram syntax





# Summary

---

- In this section we have looked at some of the more advanced features of activity diagrams:
  - Interruptible activity regions
  - Exception handlers
  - Expansion nodes
  - Advanced object flow
  - Multicast and multireceive
  - Parameter sets
  - Central buffer nodes
  - Interaction overview diagrams