WHITESTEIN
Technologies

Unified Modeling Language

Activities

Radovan Cervenka

Activity Model

→ **Specification of an algorithmic behavior.**

- Used to represent control flow and object flow models.
- *Executing activity* (of on object) is usually attached to a class or an operation.
- *Emergent activity* (of several objects) is usually attached to a package or a use-case.

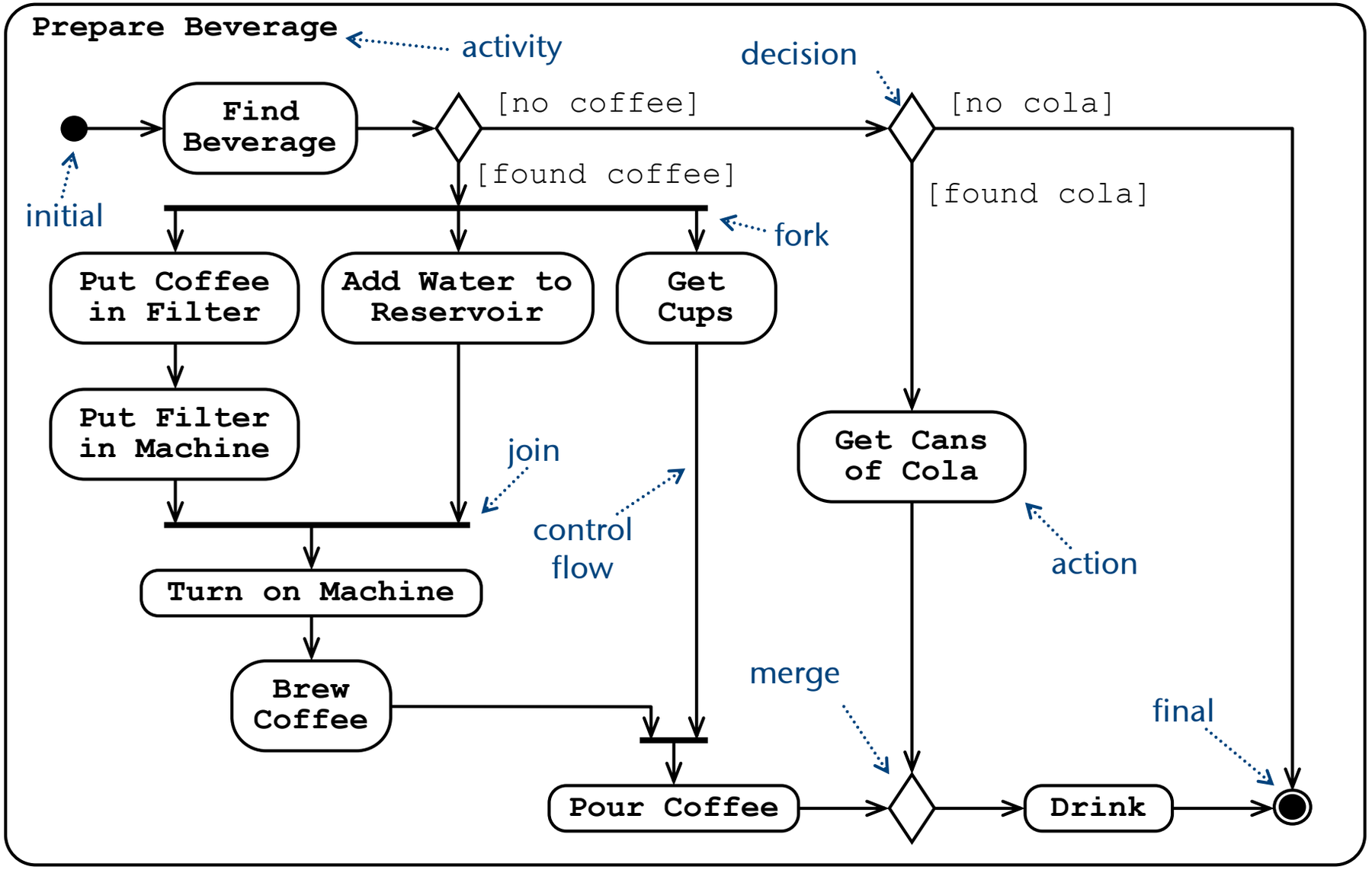
Consists of:

- Activity diagrams.
- Element descriptions.

Used (mainly) in:

- Requirements ⇒ algorithms of use cases.
- Analysis and design ⇒ behavior of objects (their operations), subsystems, and components.

Example of Activity Diagram



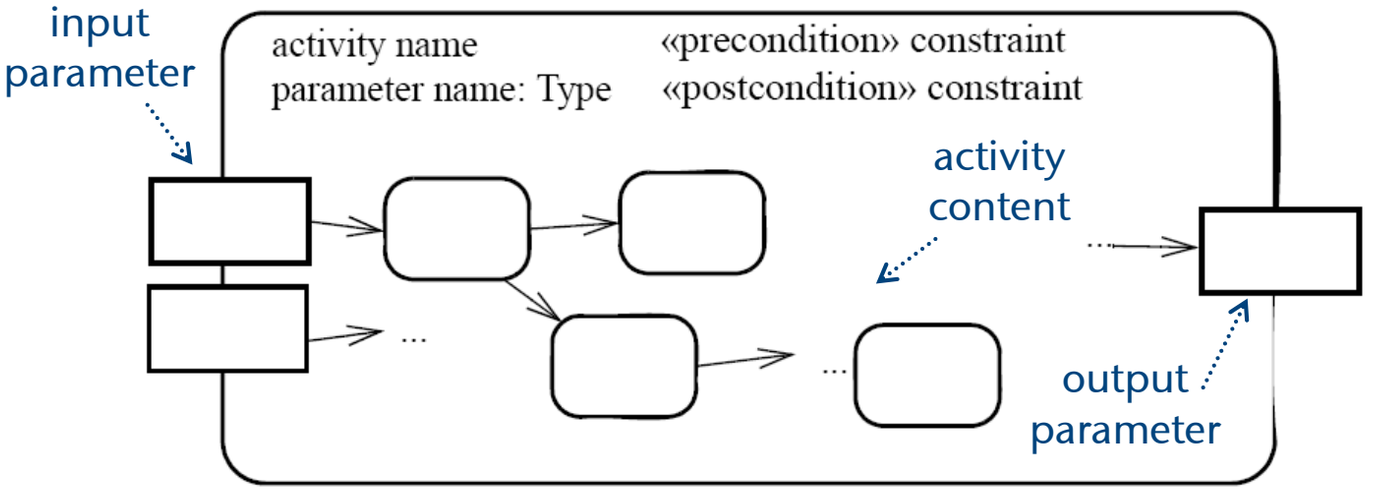
Activity

→ The specification of parameterized behavior as the coordinated sequencing of subordinate units whose individual elements are actions.

- Activities may describe procedural computation.
- The flow of execution is modeled as activity nodes connected by activity edges.
  - A node can be the execution of a subordinate behavior.
  - Activity nodes also include flow-of-control constructs, such as synchronization, decision, and concurrency control.
- Activities may be applied to organizational modeling for business process engineering and workflow.
- Formally, the semantics of activities is based on token flow.
  - Tokens represent locus of control which execute the activity nodes and traverse along to the activity edges.
  - There can be several distinct tokens in one execution of an activity.
  - When a node completes execution, a token is removed from the node and tokens are offered to some or all of its output edges.

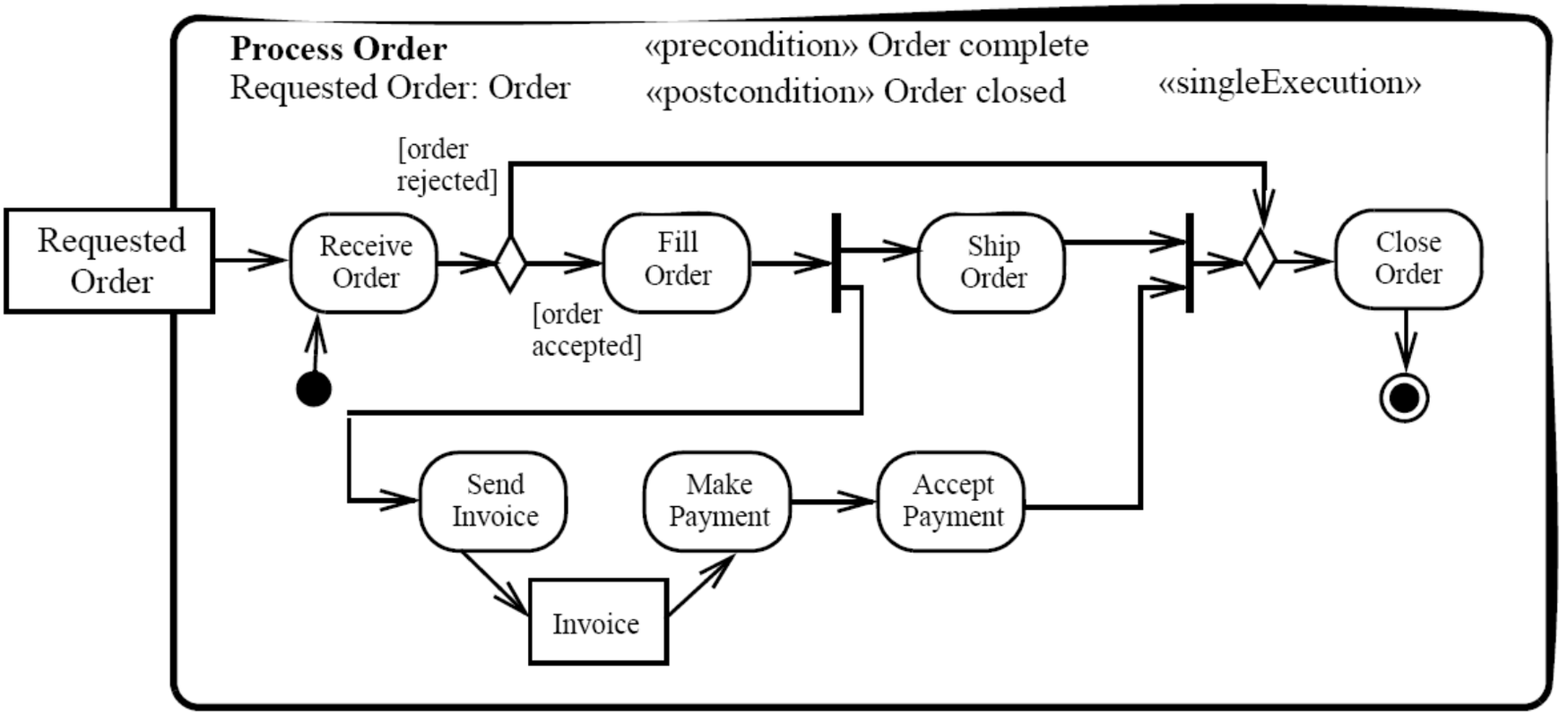
Activity (cont.)

- Activities can have inputs and outputs modeled by means of *activity parameters*.
  - An input/output represents either as a single object or an object set.
- An activity has access to the attributes and operations of its context object and any objects linked to the context object transitively.
- An activity that is also a method of a behavioral feature has access to the parameters of the behavioral feature.



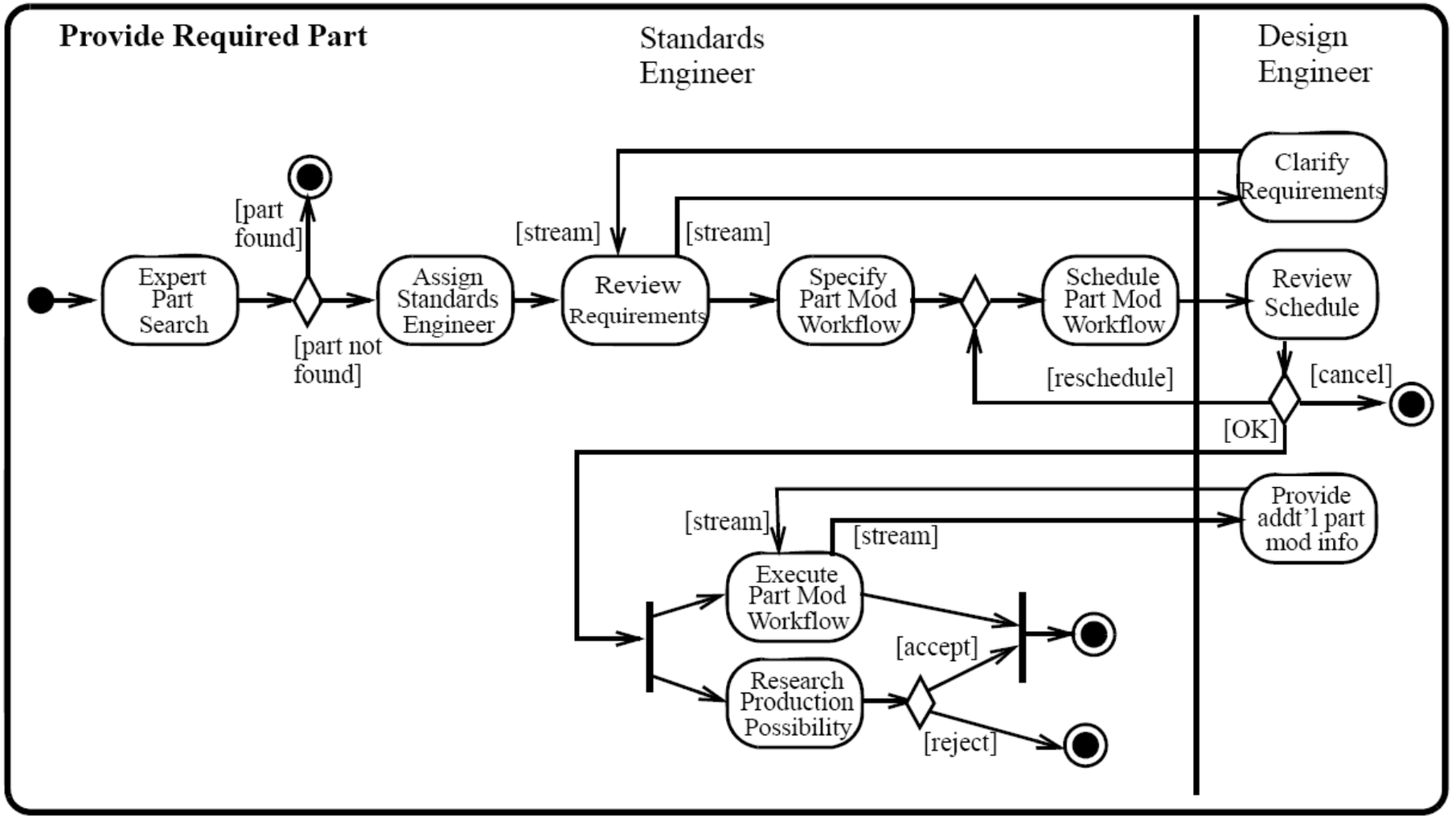
«activity» Activity Name
attribute : type attribute : type
operation (parameters) operation (parameters)

Examples of Activities (1)



«activity» Fill Order
costSoFar : USD timeToComplete : Integer
suspend () resume ()

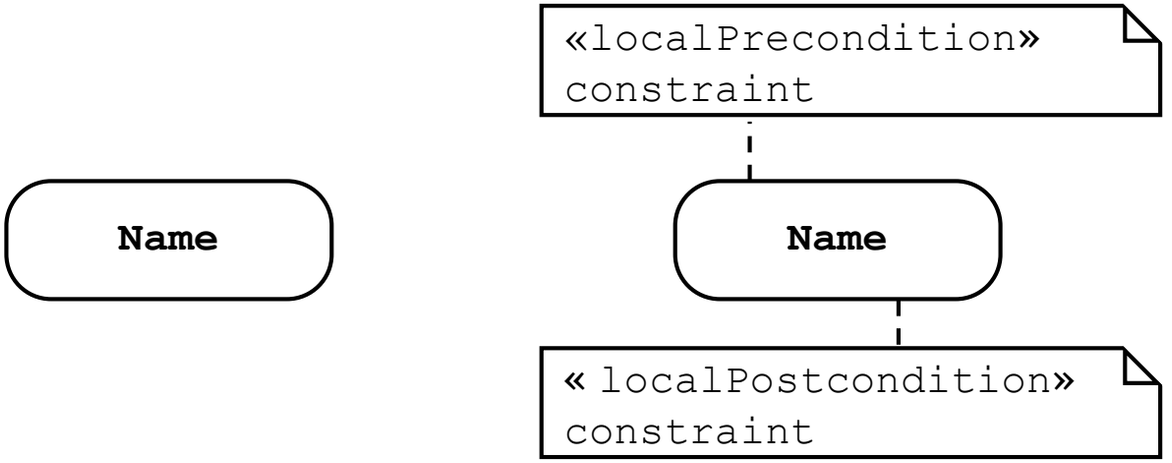
Examples of Activities (2)



Action

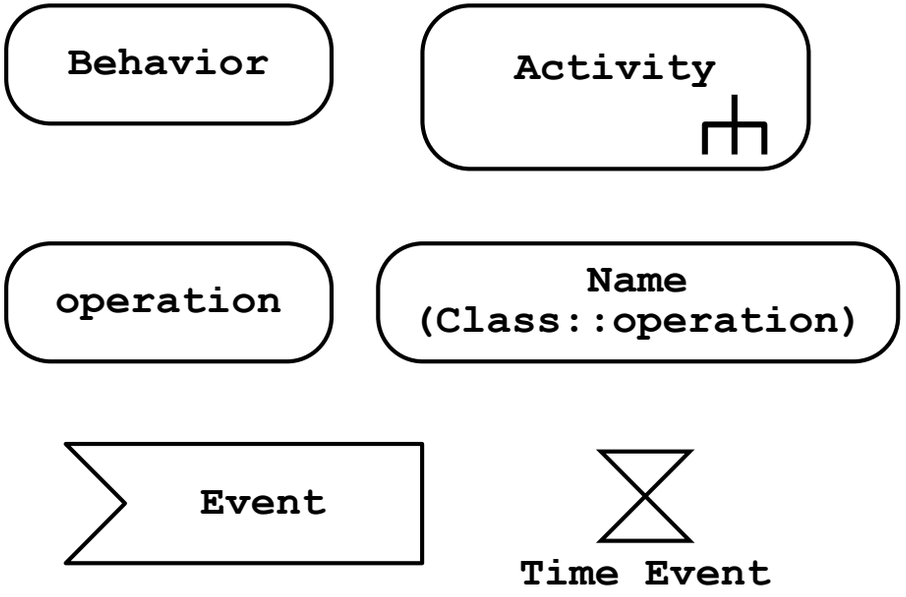
→ The fundamental unit of executable functionality.

- The execution of an action represents some transformation or processing in the modeled system.
- Action can have inputs and outputs modeled by ***pins***.
- Action can specify:
  - ***Local pre condition***–constraint that must be satisfied when execution is started.
  - ***Local post condition***–constraint that must be satisfied when execution is completed.



Some Special Kinds of Actions

- Call Behavior Action
  - → Invokes a behavior directly.
- Call Operation Action
  - → Transmits an operation call request to the target object.
- Accept Event Action
  - → Waits for the occurrence of an event meeting the specified condition.
- Send Signal Action
  - → creates a signal instance from its inputs, and transmits it to the target object

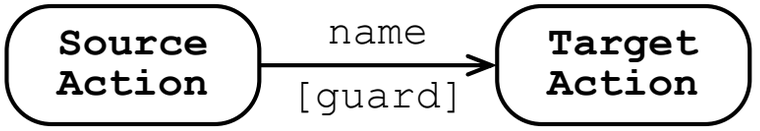


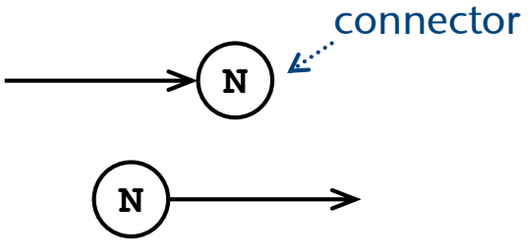
- UML provides much more special action kinds for manipulating structural features, relations, communication, etc.

Control Flow

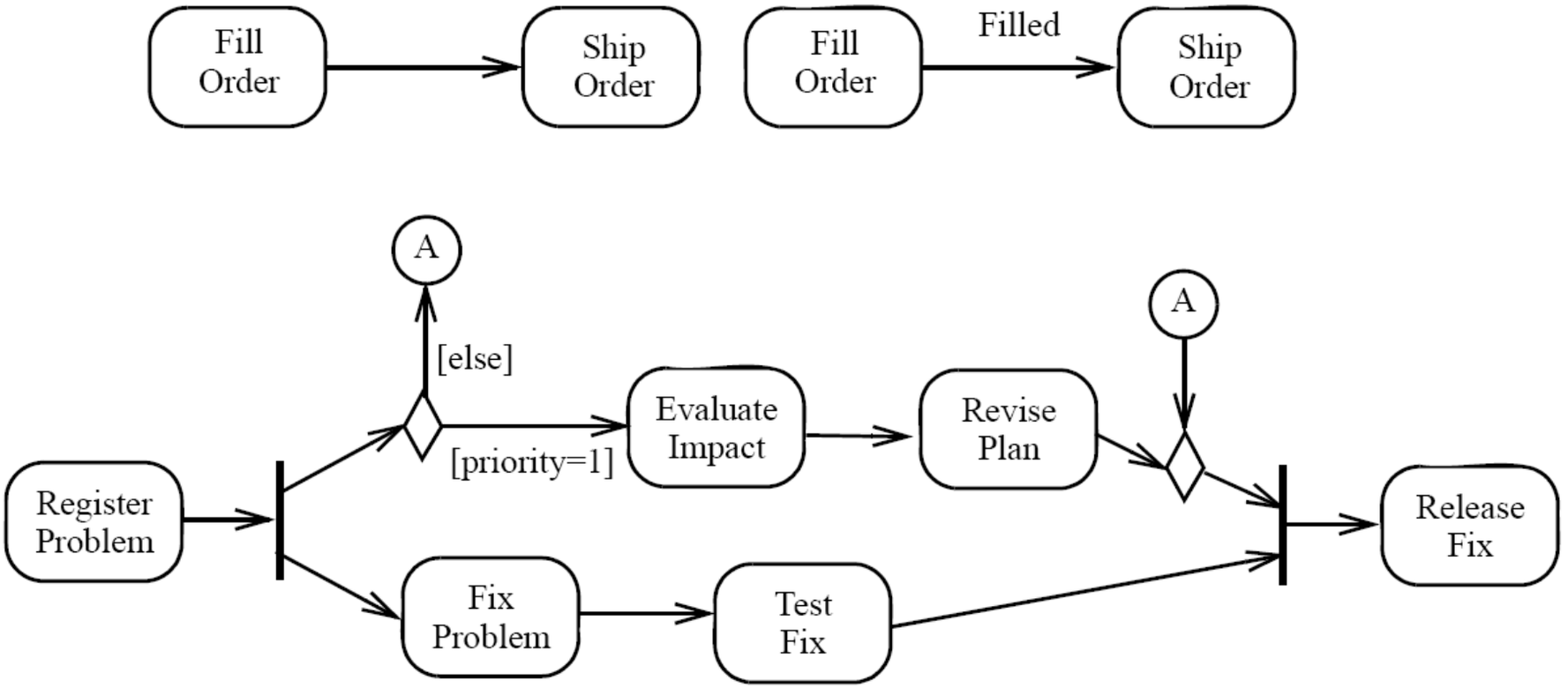
→ An activity edge that starts an activity node after the previous one is finished.

- An activity edge that only passes control tokens.
  - Tokens offered by the source node are all offered to the target node.
- Can specify a guard condition which must evaluate to true for every token that is offered to pass along the edge.
- A connector (a small circle with the name of the edge in it) can also be used to simplify diagrams with many activity edges.
  - Purely notational mechanism, it does not affect the model.
  - Every connector with a given label must be paired with exactly one other with the same label on the same activity diagram.





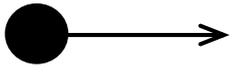
Examples of Control Flows



Initial Node

→ A control node at which flow starts when the activity is invoked.

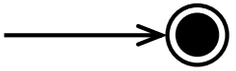
- An initial node has no incoming edges.
- Only control edges can have initial nodes as source.
- A control token is placed at the initial node when the activity starts, but not in initial nodes in structured nodes contained by the activity.
  - Tokens in an initial node are offered to all outgoing edges.
  - If an activity has more than one initial node, then invoking the activity starts multiple flows, one at each initial node.



Activity Final Node

→ A final node that stops all flows in an activity.

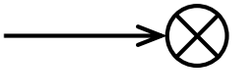
- A token reaching an activity final node terminates the activity.
  - It stops all executing actions in the activity, and destroys all tokens in object nodes, except in the output activity parameter nodes.
  - Terminating the execution of synchronous invocation actions also terminates whatever behaviors they are waiting on for return.
  - Any behaviors invoked asynchronously by the activity are not affected.
- Has no outgoing edges.
- An activity may have more than one activity final node.
- If there is more than one final node in an activity, the first one reached terminates the activity.



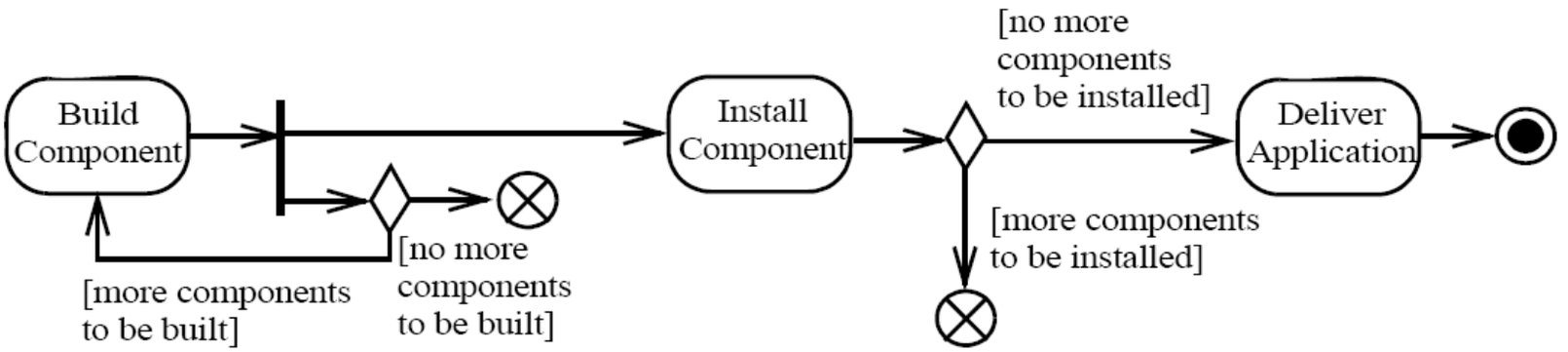
Flow Final Node

→ A final node that terminates a flow.

- Destroys all tokens that arrive at it.
- It has no effect on other flows in the activity.
- Has no outgoing edges.



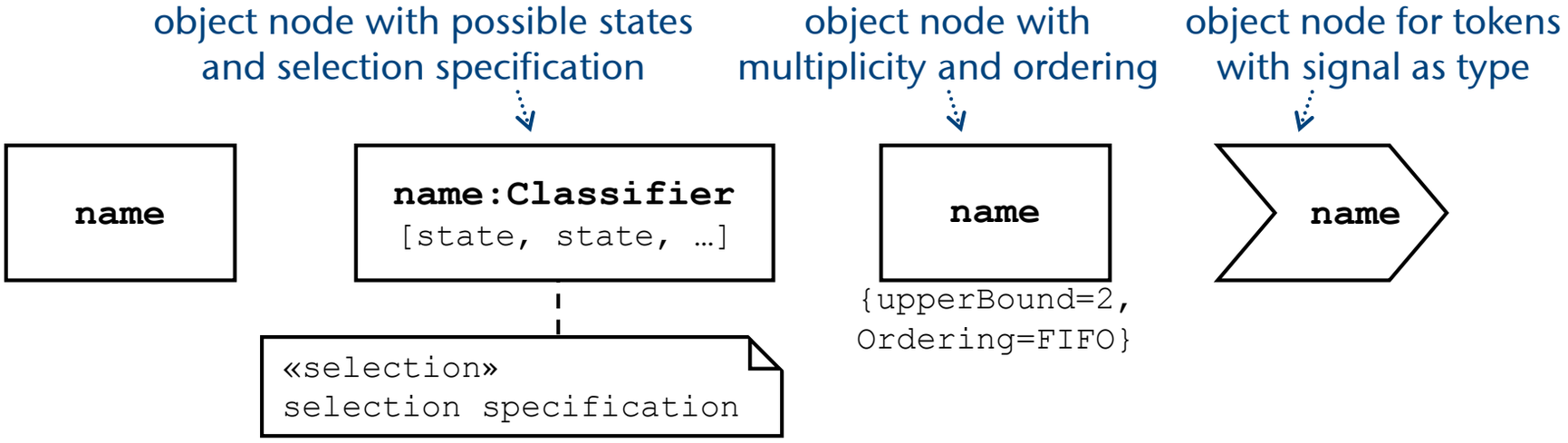
Examples of Final Nodes



Object Node

→ An ***abstract*** activity node that indicates an instance of a particular *classifier*, possibly in a particular *state*, may be available at a particular point in the activity. Its concrete sub-elements are, e.g., Pin and Activity Parameter.

- All edges coming into or going out of object nodes must be object flow edges.
- Can specify multiplicity, ordering and selection criteria of the represented values (tokens).
- Multiple tokens containing the same value may reside in the object node at the same time. This includes data values.
  - A token in an object node can traverse only one of the outgoing edges.

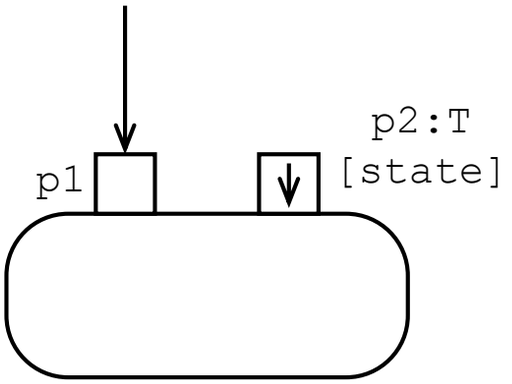


Pins

Input Pin

→ A specialized pin and object node that holds input values to be consumed by an action.

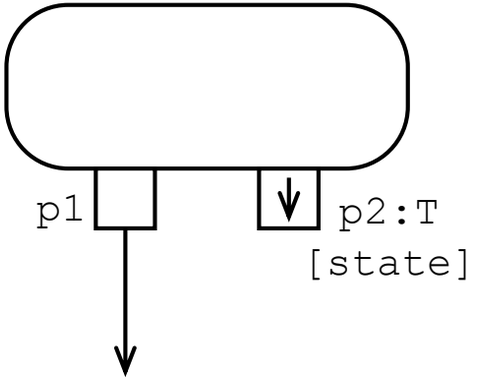
- Can specify name, type status and multiplicity.
  - An action cannot start execution if an input pin has fewer values than the lower multiplicity.
  - The upper multiplicity determines how many values are consumed by a single execution of the action.



Output Pin

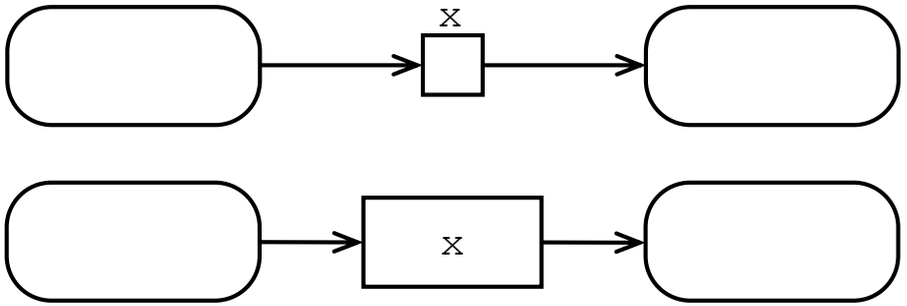
→ A specialized pin and object node that holds output values produced by an action.

- Can specify name, type and multiplicity.
  - An action cannot terminate itself if an output pin has fewer values than the lower multiplicity.
  - An action may not put more values in an output pin in a single execution than the upper multiplicity of the pin.

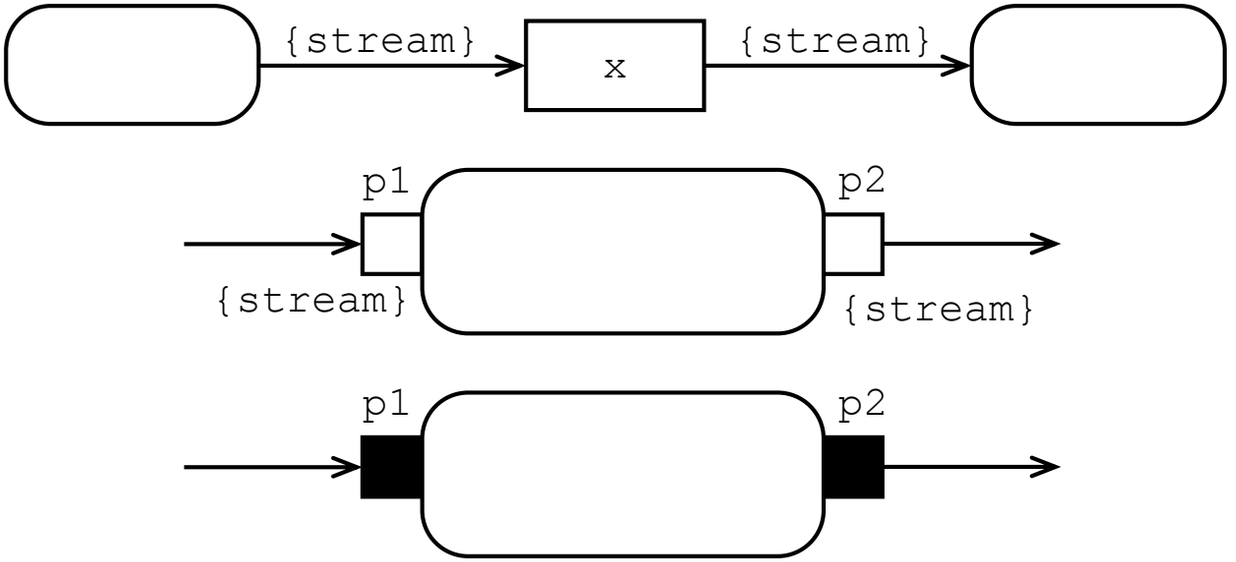


Pins (cont.)

Standalone pin notation representing input and output pins together:



- Pins can specify streaming, i.e., tells whether an input pin may accept values while its behavior is executing, or whether an output pin post values while the behavior is executing.

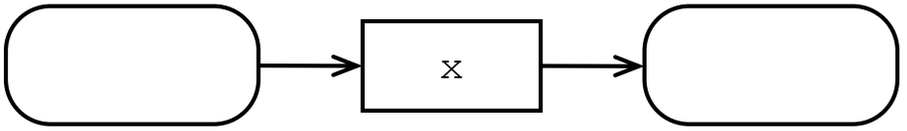


Object Flow

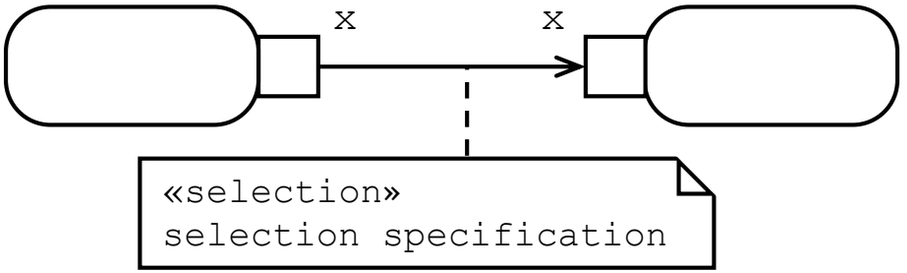
→ An activity edge that can have objects or data passing along it.

- Models the flow of values to or from object nodes.
- Object flows add support for multicast/receive, token selection from object nodes, and transformation of tokens.
- Object flows may not have actions at either end.
- Object nodes connected by an object flow must have compatible types and upper bounds.

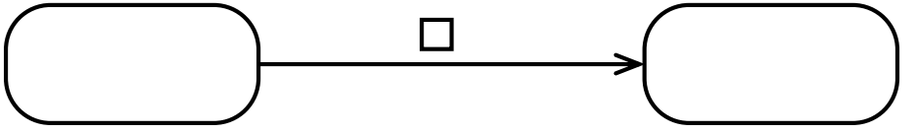
Two object flows linking an object node and actions:



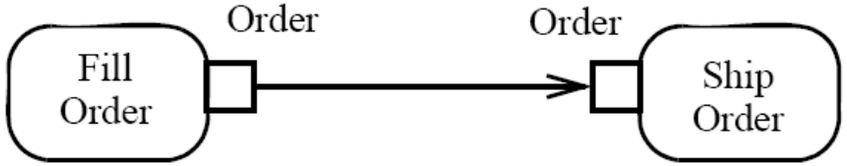
An object flow with selection linking two pins:

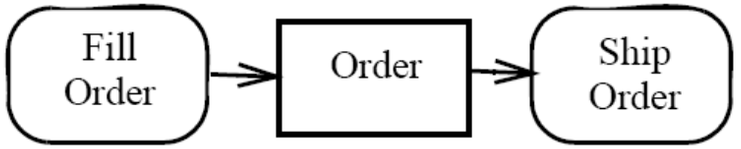


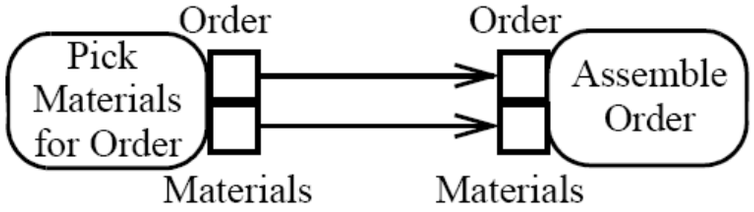
An object flow with pins elided:

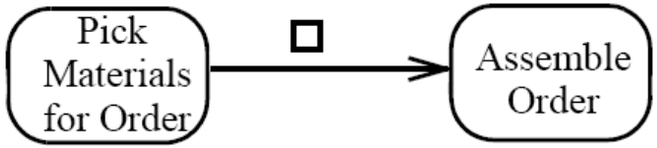


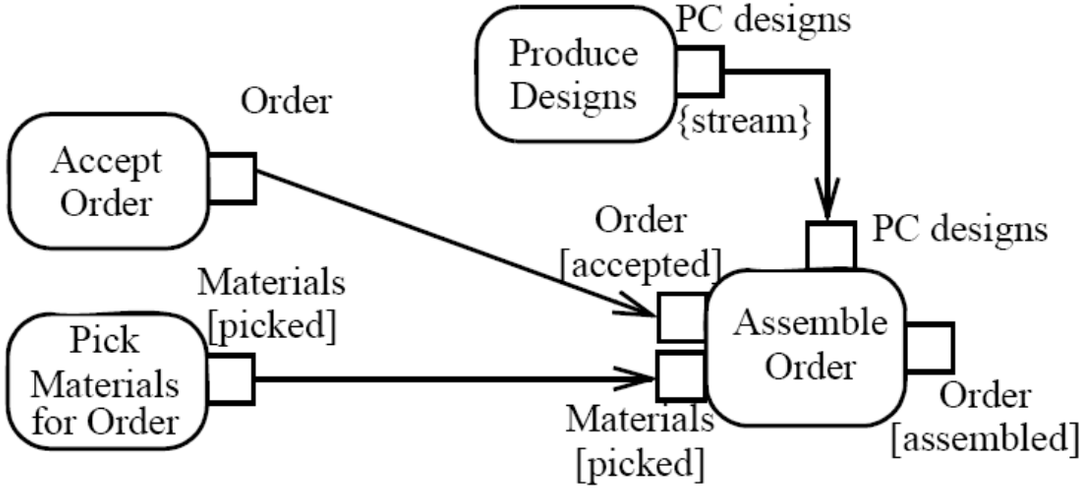
Examples of Pins and Object Flows

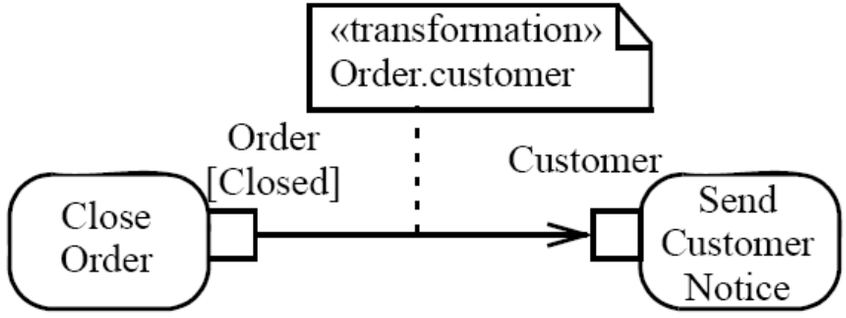


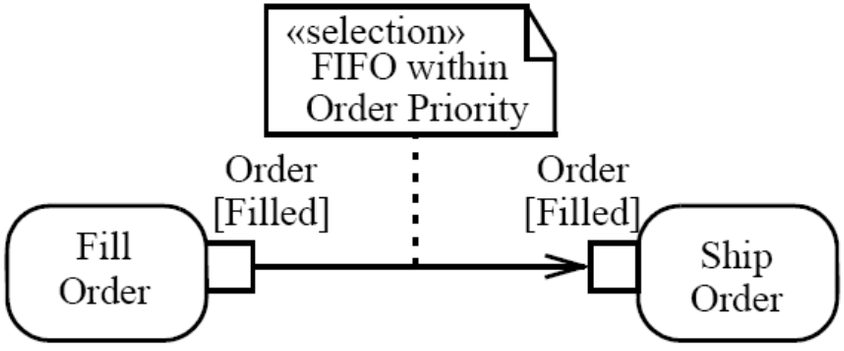








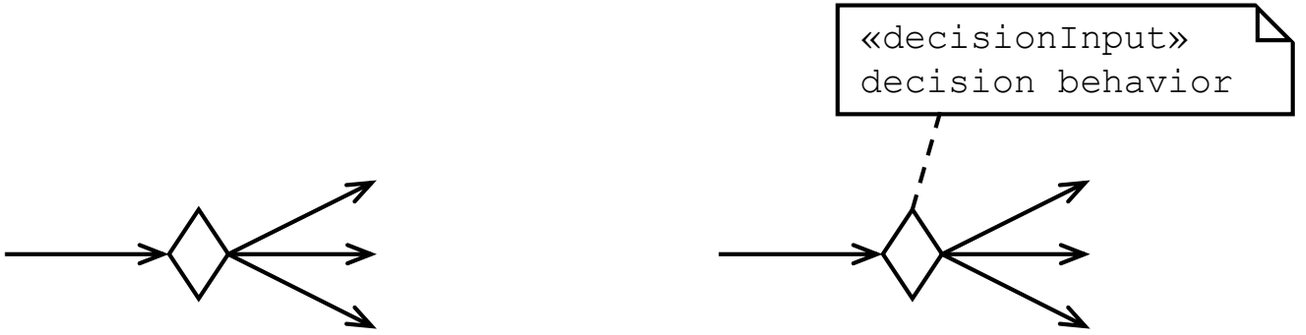




Decision Node

→ A control node that chooses between outgoing flows.

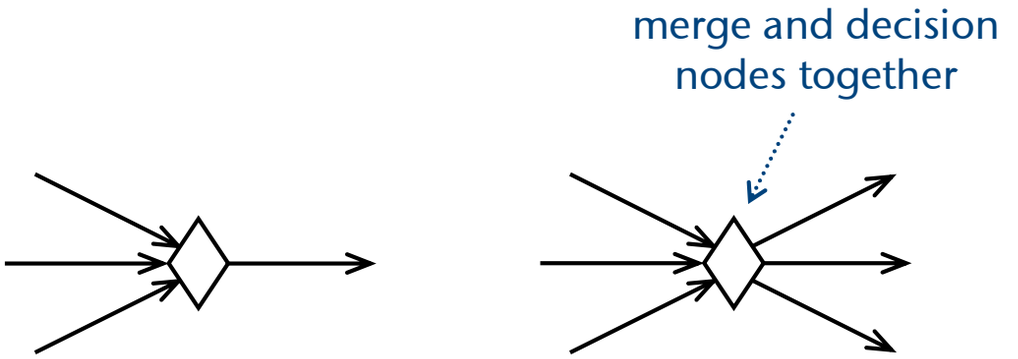
- Has one incoming edge and multiple outgoing activity edges.
- Each token arriving at a decision node can traverse only one outgoing edge.
- Guards of the outgoing edges are evaluated to determine which edge should be traversed.
  - The evaluation order is not defined.
  - The predefined ‘else’ guard can be used.
- A decision behavior/condition applied for each token before it is offered to the outgoing edges can also be specified.



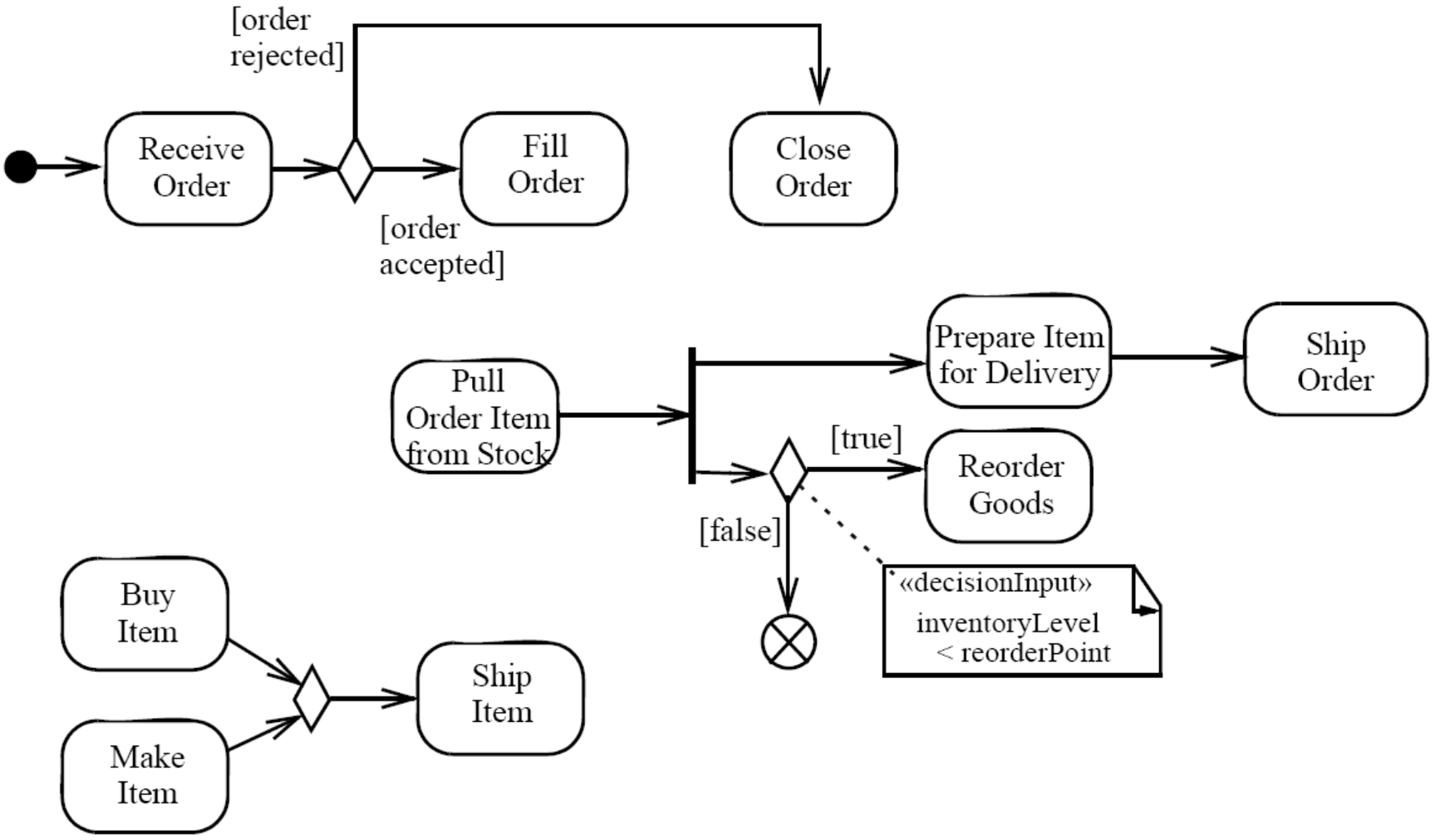
Merge Node

→ A control node that brings together multiple alternate flows.

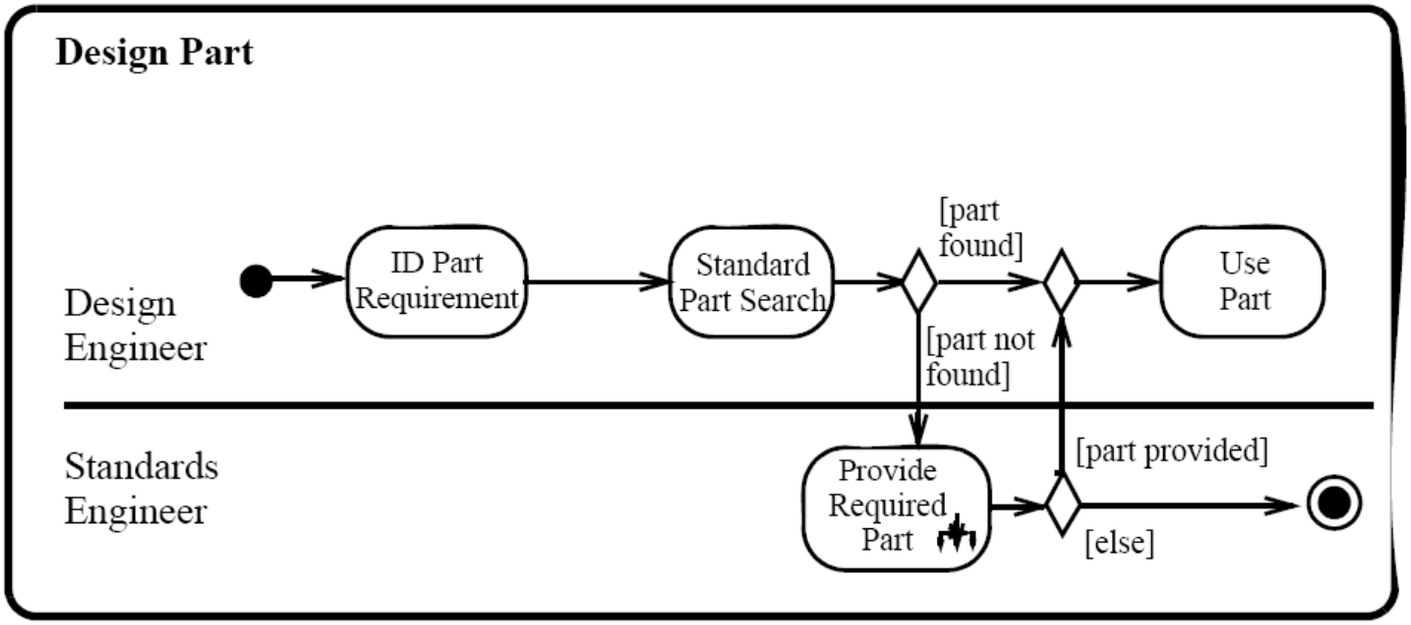
- Has multiple incoming edges and a single outgoing edge.
- All tokens offered on incoming edges are offered to the outgoing edge.
  - There is no synchronization of flows or joining of tokens.



Examples of Decisions and Merges (1)



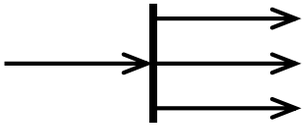
Examples of Decisions and Merges (2)



Fork Node

→ A control node that splits a flow into multiple concurrent flows.

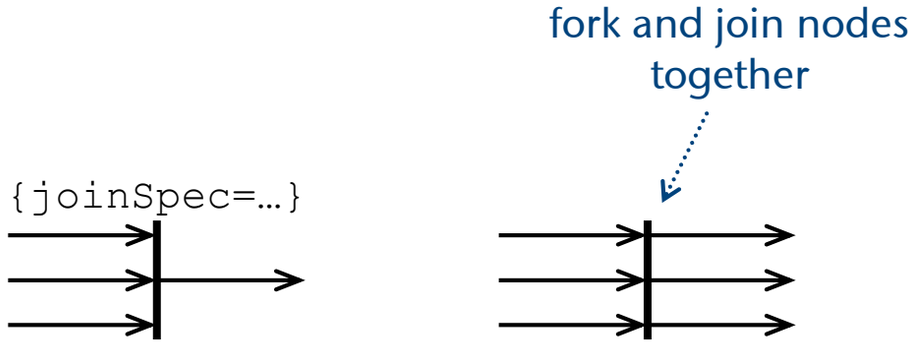
- Has one incoming edge and multiple outgoing edges.
- Tokens arriving at a fork are duplicated across the outgoing edges.
  - If at least one outgoing edge accepts the token, duplicates of the token are made and one copy traverses each edge that accepts the token.
  - The outgoing edges that did not accept the token due to failure of their targets to accept it, keep its copy in an implicit FIFO queue until it can be accepted by the target.
  - The rest of the outgoing edges do not receive a token (these are the ones with failing guards).



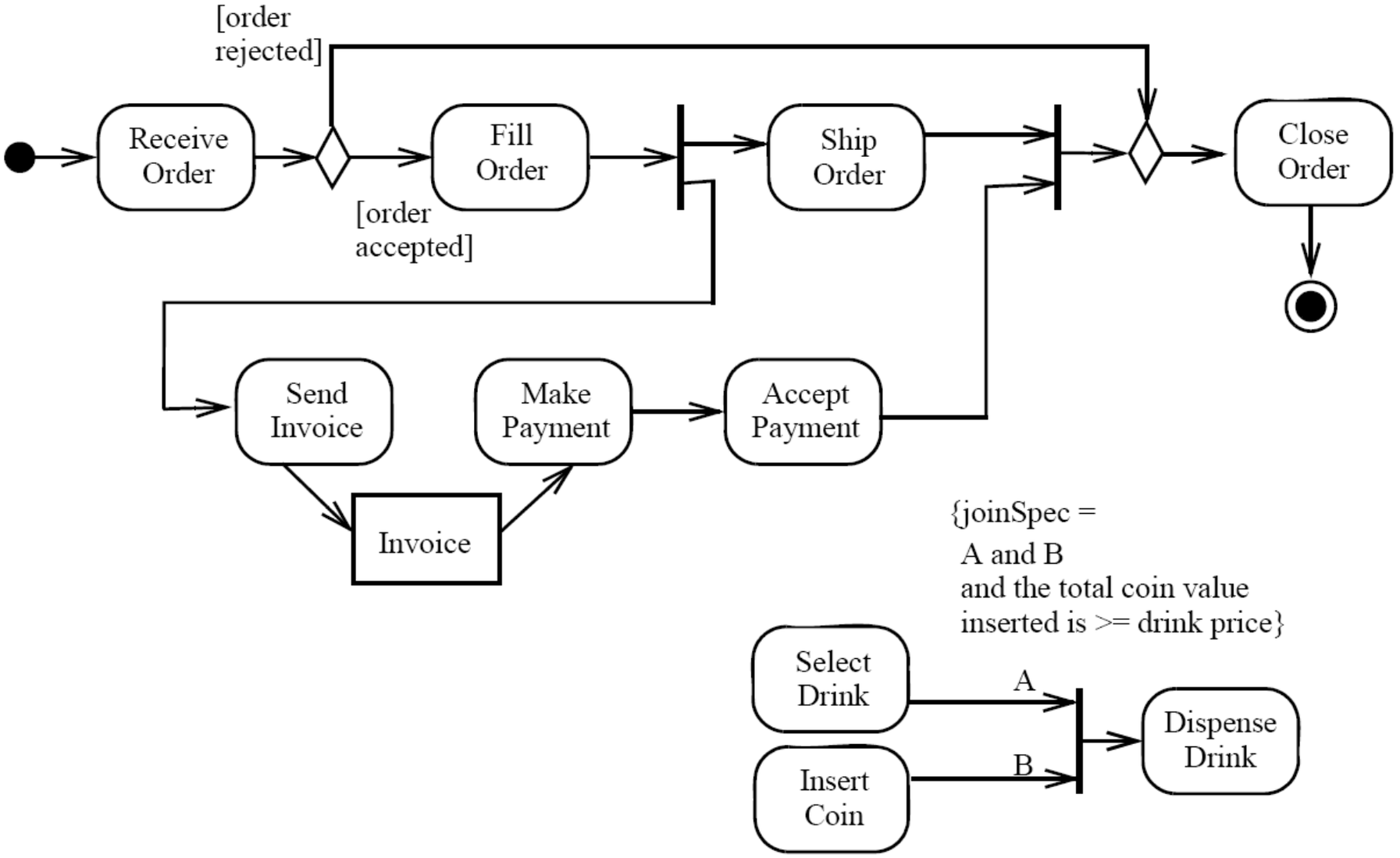
Join Node

→ A control node that synchronizes multiple flows.

- Has multiple incoming edges and one outgoing edge.
- Can have a boolean value specification (modeled by the *joinSpec* tag) using the names of the incoming edges to specify the conditions under which the join will emit a token. If the joinSpec is not given, then:
- If all the tokens offered on the incoming edges are control tokens, then one control token is offered on the outgoing edge.
- If some of the tokens offered on the incoming edges are control tokens and others are data tokens, then only the data tokens are offered on the outgoing edge. Tokens are offered on the outgoing edge in the same order they were offered to the join.



Examples of Fork and Join Nodes

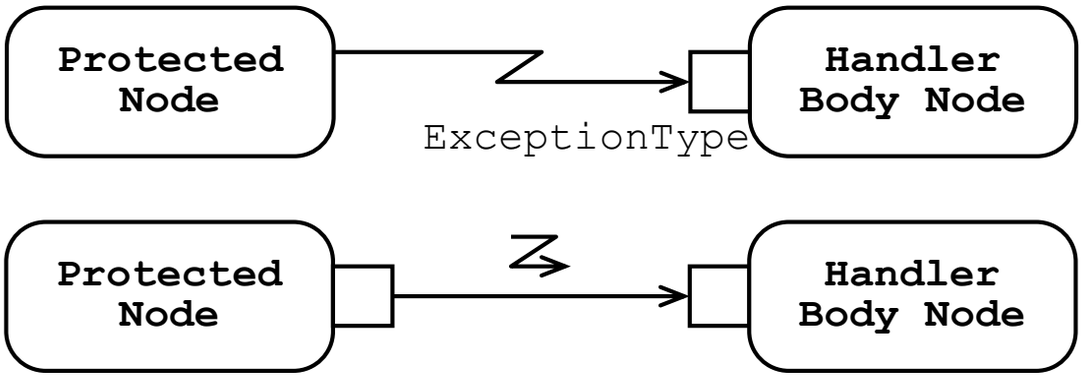


Exception Handler

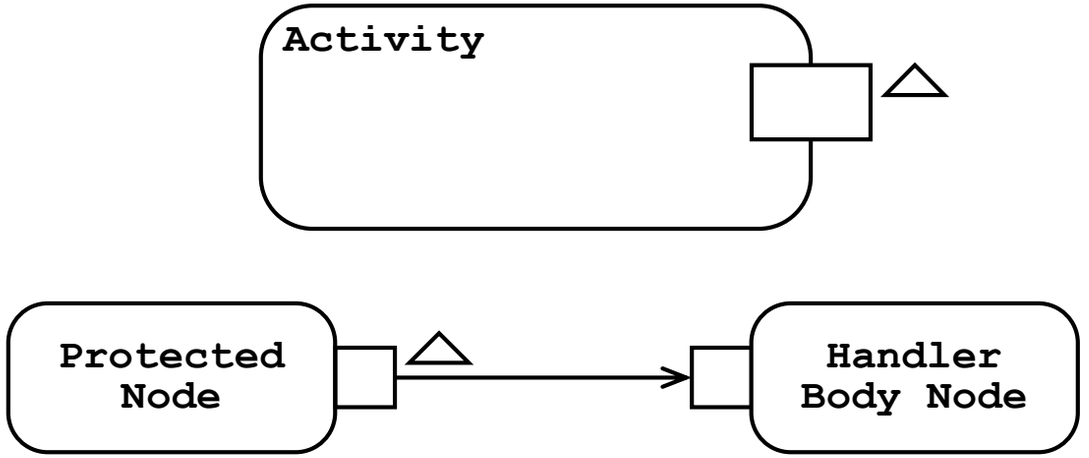
→ Specifies a *body* to execute in case the specified *exception* occurs during the execution of the *protected node*.

- If an exception occurs (a Raise Exception Action is executed) in the protected node, all the tokens in the protected node are terminated. Then, the exception handlers are examined for matching the exception type, and the handler body of any matching exception handler is used to handle the exception which arrives via the exception input pin.
- If the exception is not caught at the level of the protected node, the exception handling process repeats at the level of the enclosing structured node or activity.
- If the exception is not caught at the top-most level of asynchronously invoked activity, the exception is lost.
- If the action that invoked the activity is synchronous, the exception propagates up to that action. The process of exception propagation recurs until the exception is caught, or reaches the topmost level of the system, where the behavior for the uncaught exceptions is unspecified.
- The result tokens of the handler body become the result tokens of the protected node.

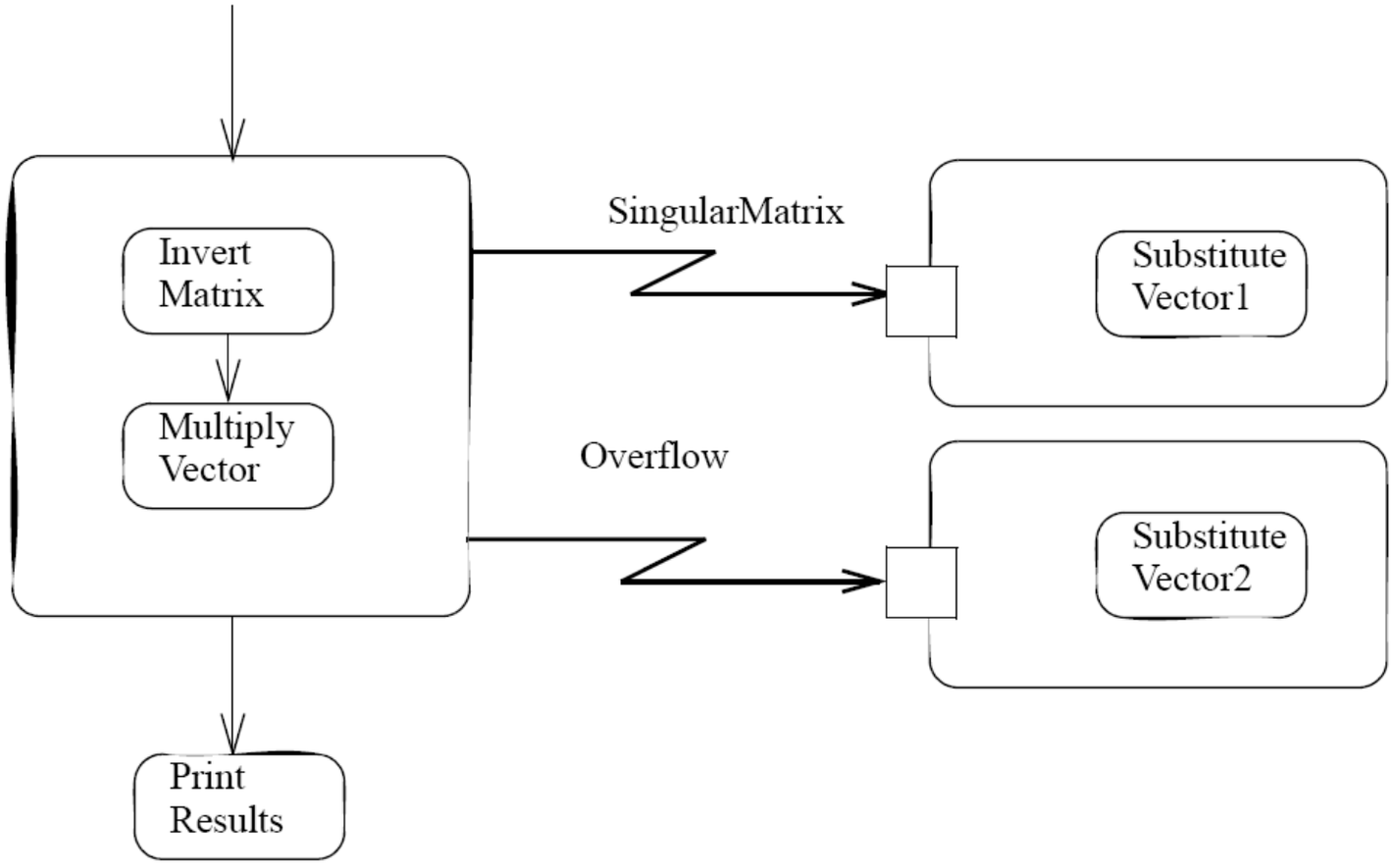
Exception Handler (cont.)



- An alternative notation for exception flows and exception object nodes (activity parameters and pins) with a small triangle icon:



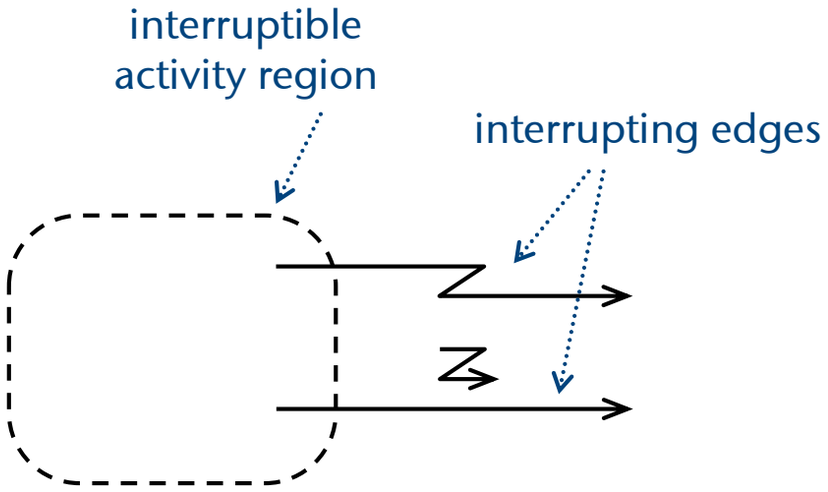
Example of Exception Handler



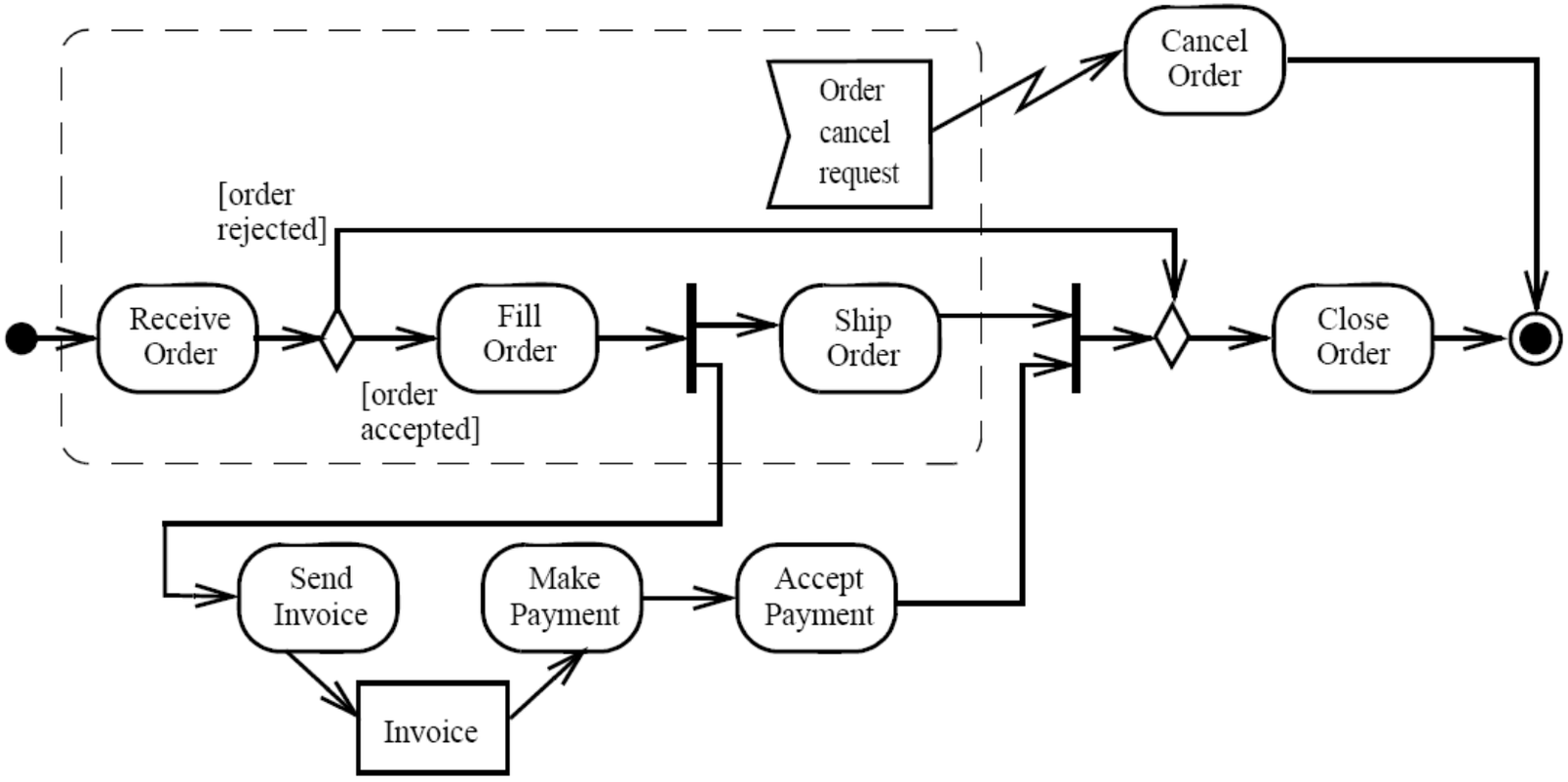
Interruptible Activity Region

→ An activity group that supports termination of tokens flowing in the portions of an activity.

- Contains other activity nodes.
- When a token leaves an interruptible region via edges designated by the region as interrupting edges, all tokens and behaviors in the region are terminated.
- Interrupting edges of a region must have their source node in the region and their target node outside the region in the same activity containing the region.



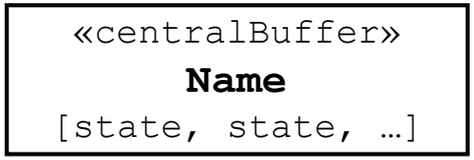
Example of Interrupting Activity Region



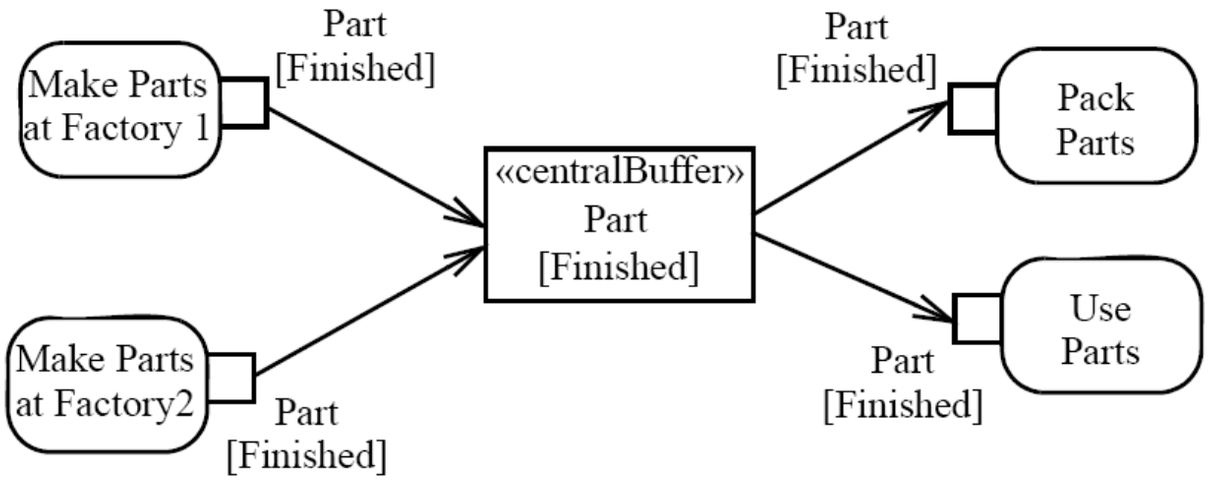
Central Buffer Node

→ An object node for managing flows from multiple sources and destinations.

- Accepts tokens from upstream object nodes and passes them along to downstream object nodes.



Example of Central Buffer Node

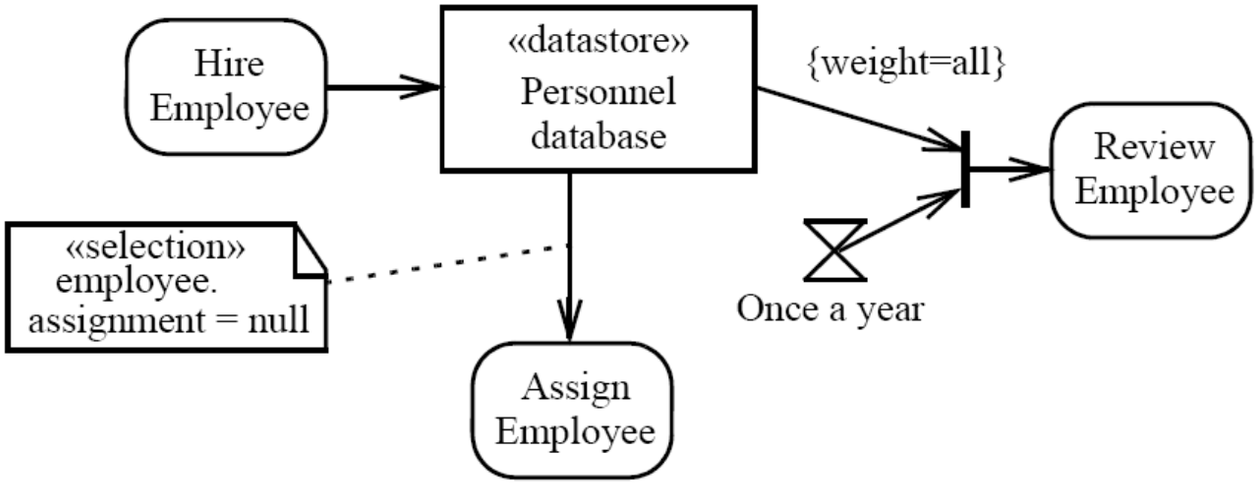


Data Store Node

→ A central buffer node for non-transient information.

- Keeps all tokens that enter it and copies them when they are chosen to move downstream.
- Selection and transformation behavior on outgoing edges can be designed to get information out of the data store, as if a query were being performed.
- Incoming tokens containing a particular object replace any tokens in the object node containing that object.

Example of Data Store Node



Expansion Region

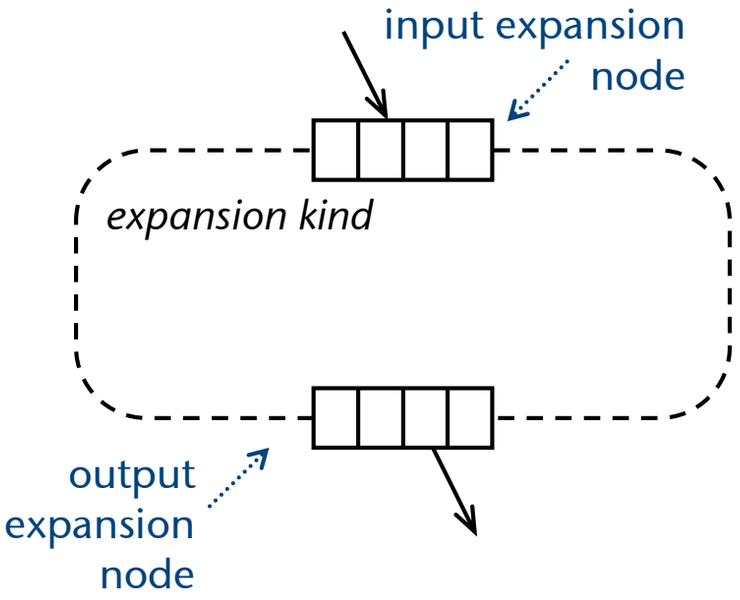
Expansion Region

→ A structured activity region that executes multiple times corresponding to elements of an *input collection*.

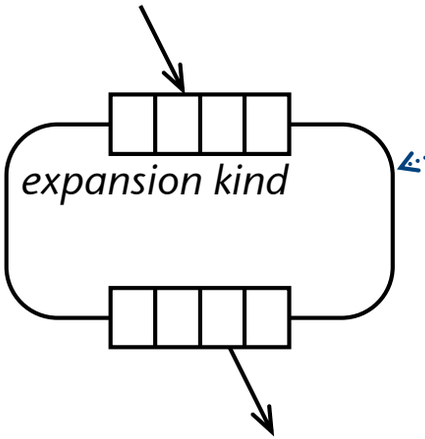
- Each input is a collection of values modeled as an ***expansion node***.
  - If there are multiple inputs to one expansion node, each of them must hold the same kind of collection.
  - Each input flow edge produces elements of the input collection.
- The expansion region is executed once for each element (or position) in the input collection.
- On each execution of the region, an output value from the region is inserted into an *output collection*, modeled also as an expansion node, at the same position as the input elements.
  - If the region execution ends with no output, then nothing is added to the output collection.
- From the inside of the region, expansion nodes are visible as individual values.

Expansion Region (cont.)

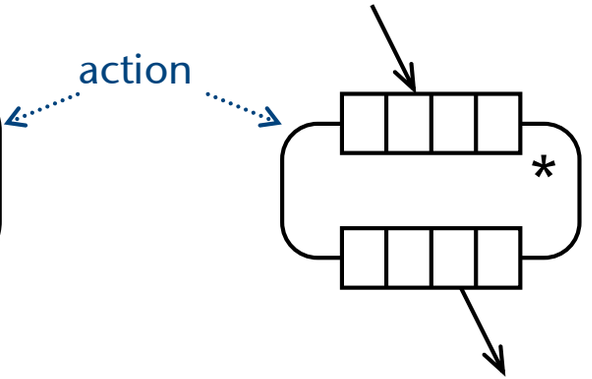
- Any object flow edges that cross the boundary of the region, without passing through expansion nodes, provide values that are fixed within the different executions of the region input pins.
- The ***expansion kind*** specifies the way in which the executions interact:
  - *parallel*–all interactions are independent
  - *iterative* (default)–the interactions occur in order of the elements
  - *stream*–a stream of values flows into a single execution



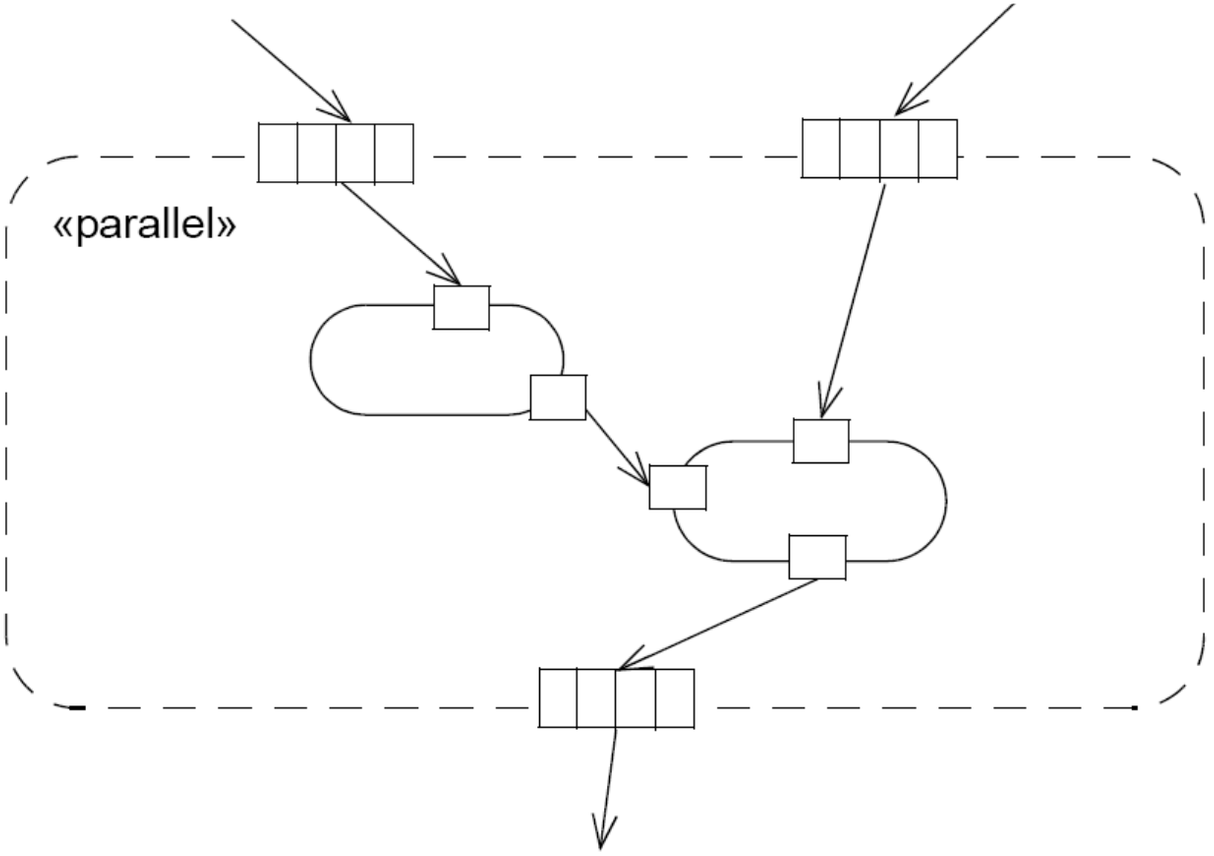
Shorthand notation for expansion region containing a single action:



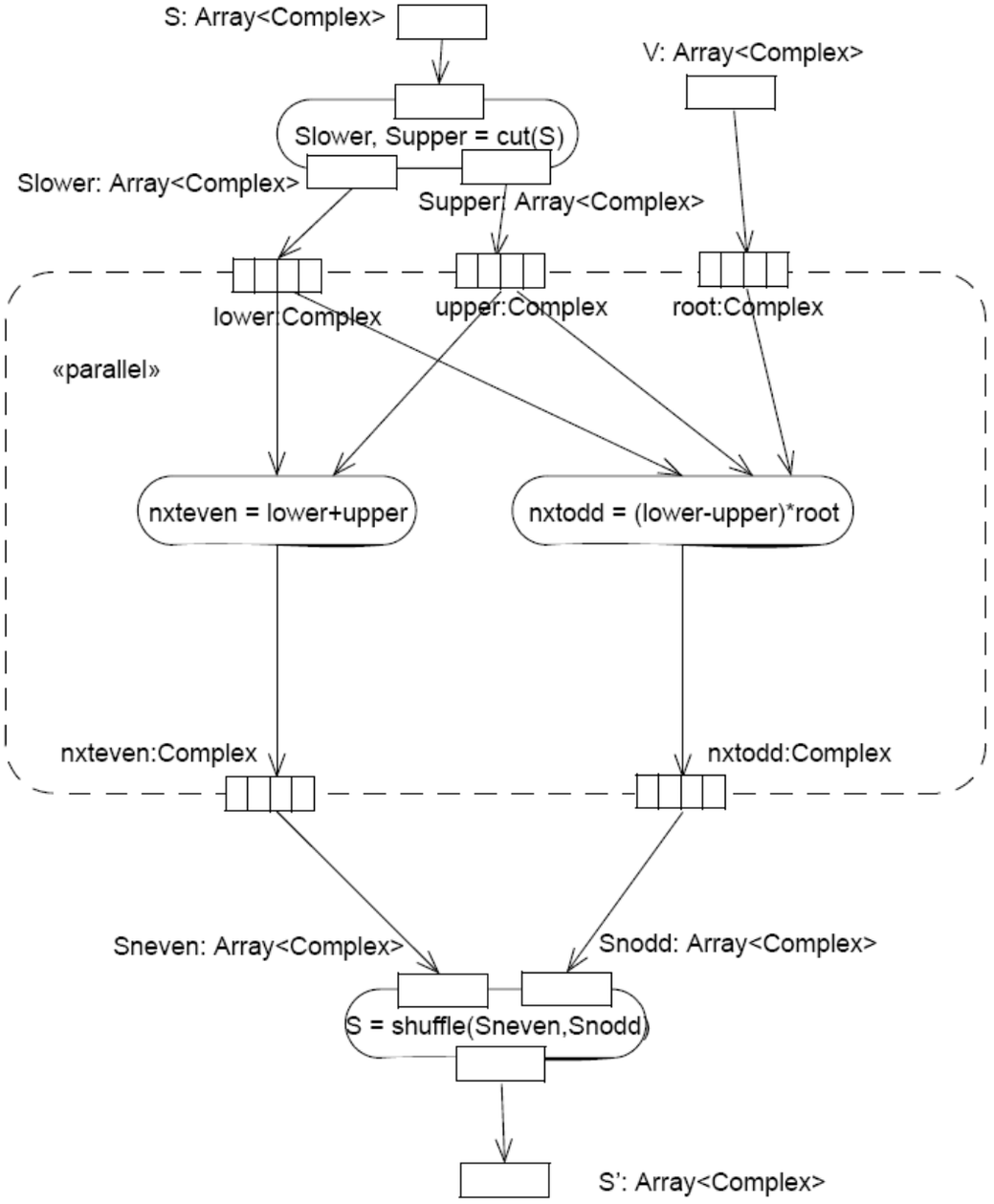
Shorthand notation for parallel expansion region containing a single action:

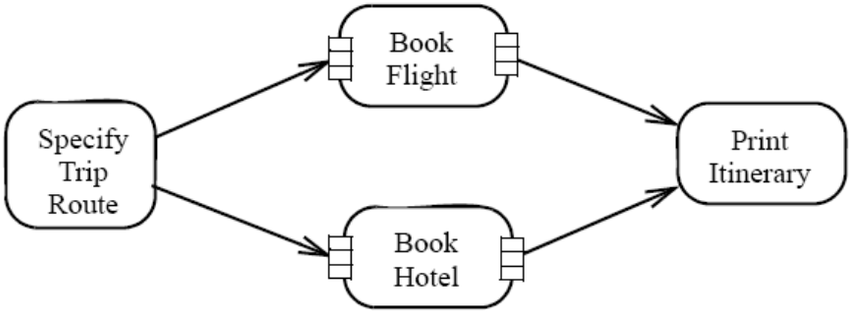


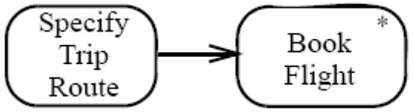
Examples of Expansion Regions (1)



Examples of Expansion Regions (2)







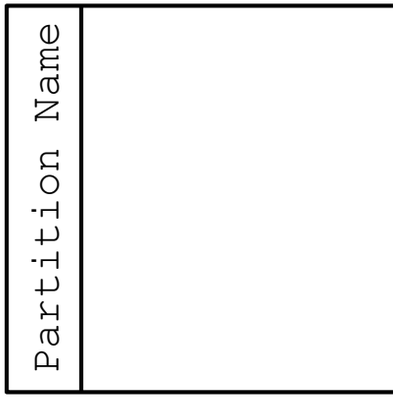
Activity Partition

→ A kind of activity group for identifying actions that have some characteristic in common.

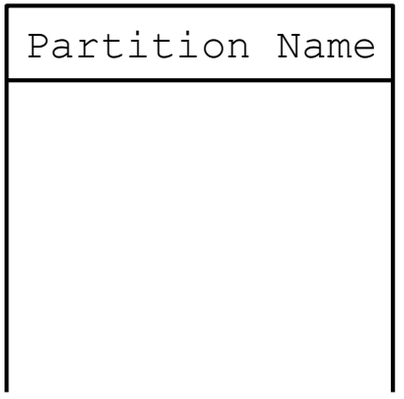
- A partition can represent a classifier, instance, part, attribute or value which is responsible for execution of the contained activity part.
- An ***external partition*** (marked by *«external»*) represents an entity to which the partitioning structure does not apply.
- Partitions can share contents.
- Partitions can be hierarchical and multi-dimensional.
- Partitions do not affect the token flow of the model.

Activity Partition (cont.)

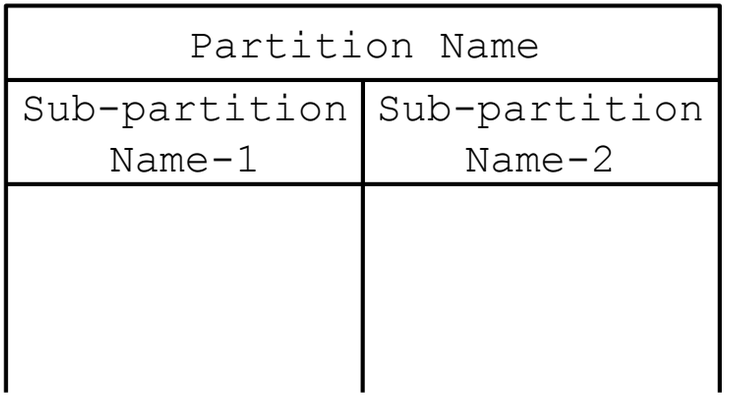
Horizontal partition:



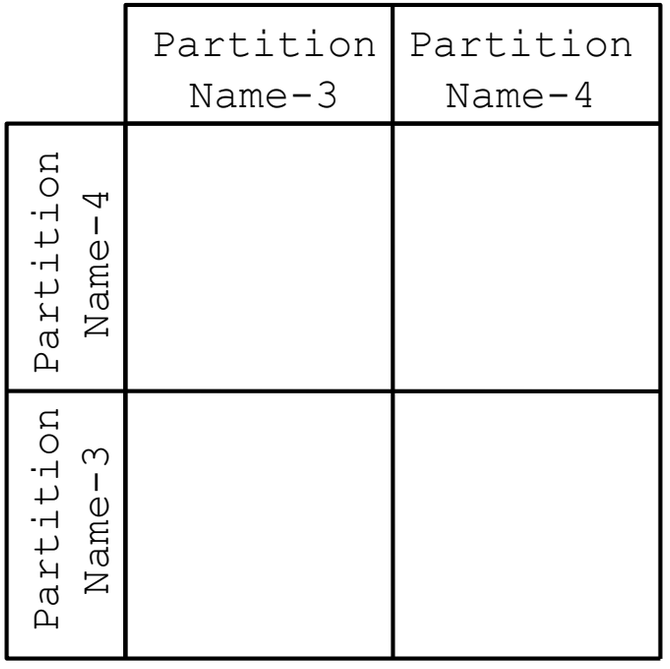
Vertical partition:



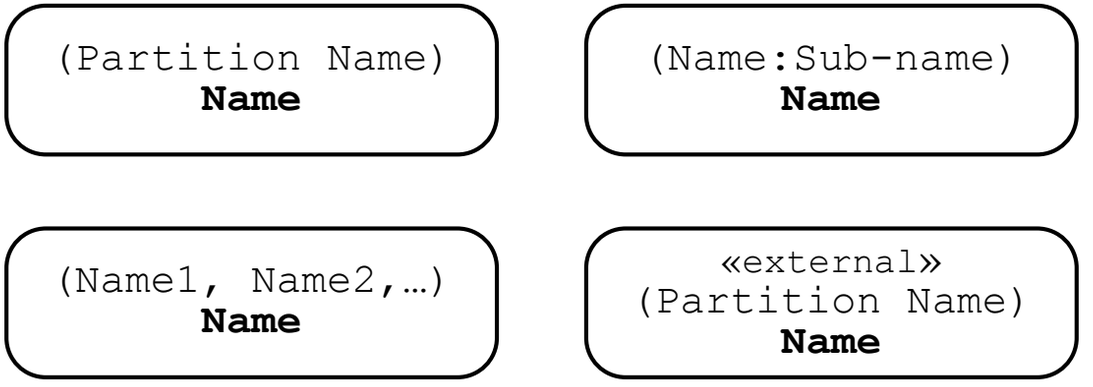
Hierarchical partitions:



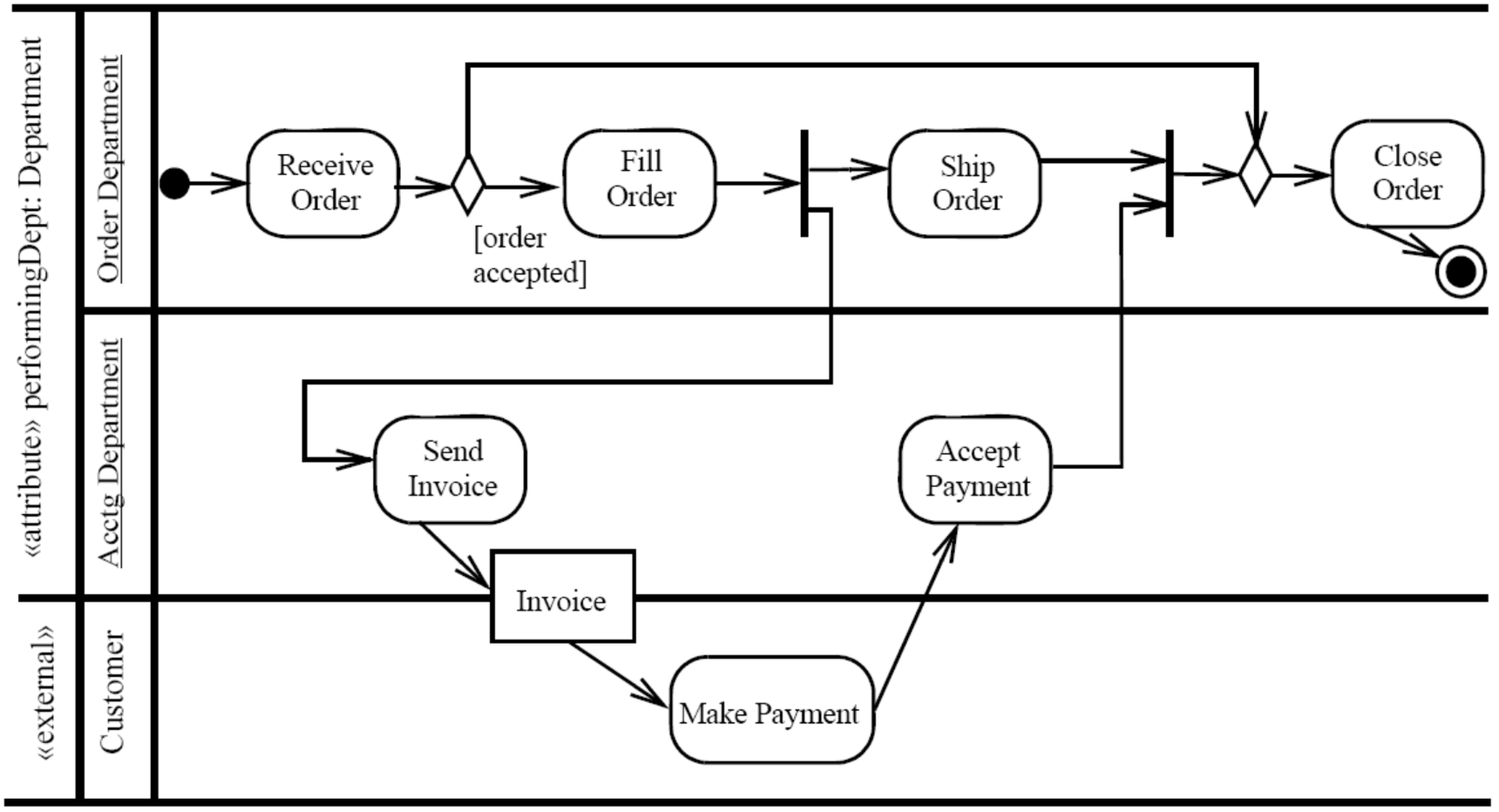
Multi-dimensional partitions:



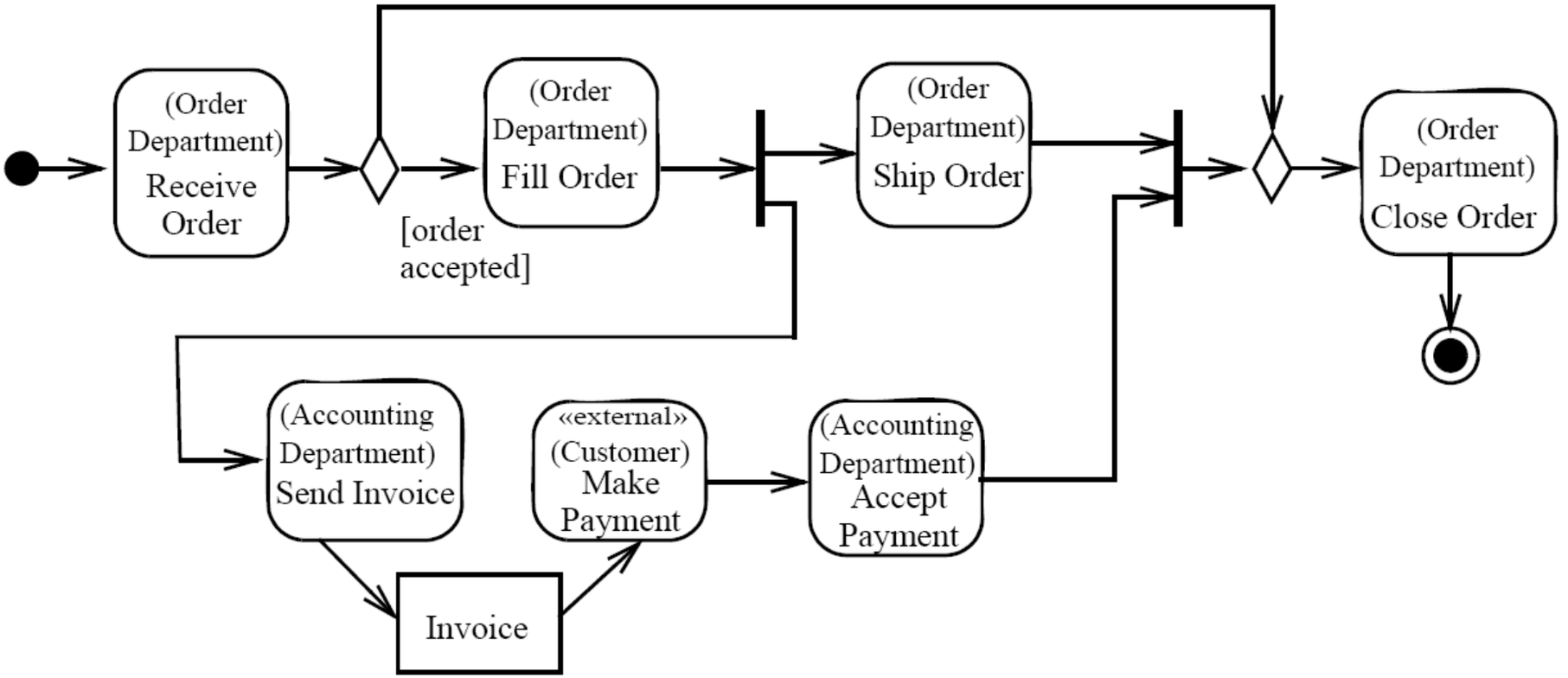
Partition(s) notated on a specific activity:



Examples of Activity Partitions (1)



Examples of Activity Partitions (2)



Examples of Activity Partitions (3)