An Introduction to Workflow Modeling using Activity Models

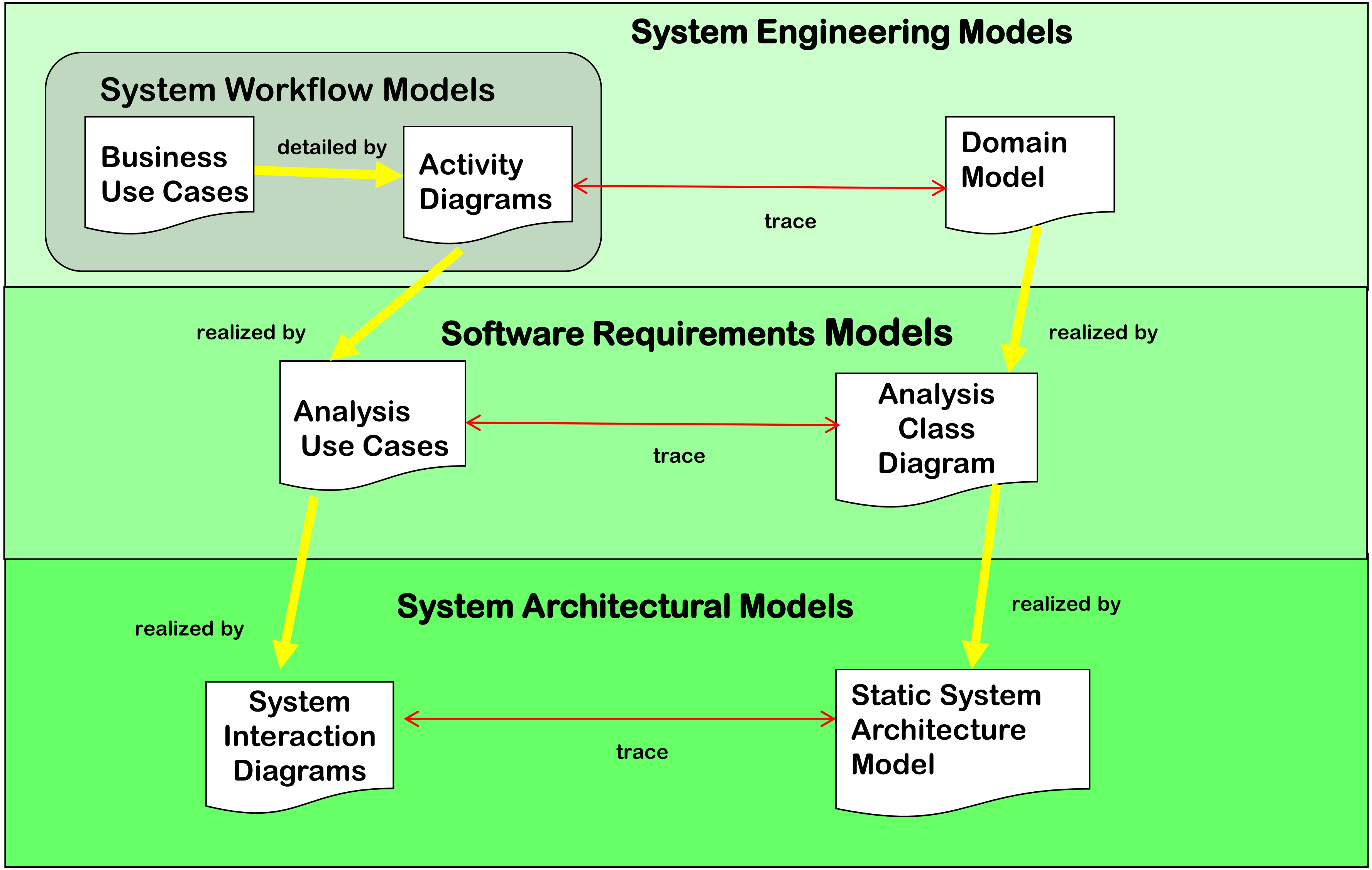
Robert B. France

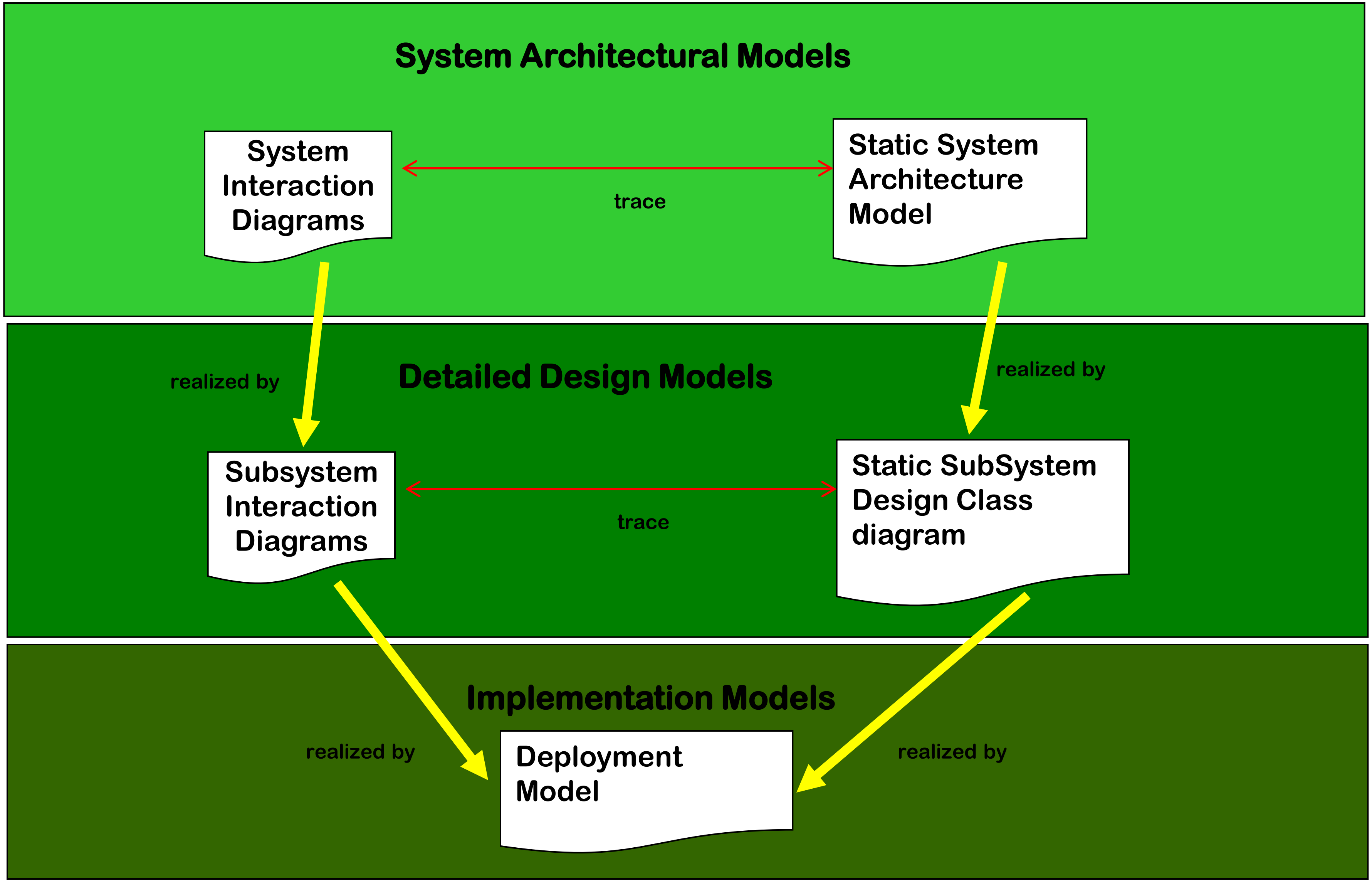
Colorado State University

Software Development Phases

- System Engineering (Business Process Engineering)
  - Focus on understanding the context in which software will operate
- Software Requirements Analysis
  - Focus on understanding specific parts of system problem targeted by software
- Design
  - Focus on developing a solution that satisfies the requirements
  - Two sub-phases: Architectural design; Detailed design
- Implementation
  - Focus on developing an executable and deployable form of the design

Models in a UML process




System Architectural Models
System Interaction Diagrams
trace
Static System Architecture Model
realized by
realized by
Detailed Design Models
Subsystem Interaction Diagrams
trace
Static SubSystem Design Class diagram
Implementation Models
realized by
realized by
Deployment Model

System/Business Process Engineering

- Software exists within some larger system
  - Encompassing system must be understood if software is to work properly within system
- The process by which a software engineer learns about the domain to better understand the problem is called domain analysis:
  - The *domain* is the general field of business or technology in which the clients will use the software
  - A *domain expert* is a person who has a deep knowledge of the domain
- System engineering is concerned with modeling the system encompassing software.
  - If the system exists within a business organization system engineering is referred to as *business process engineering*

Modeling systems

- Two types of models
- Domain model: describe system entities and their static relationships
  – Described using class diagrams
- Workflow/process model: describes how work is accomplished in system
  – Described using activity diagrams

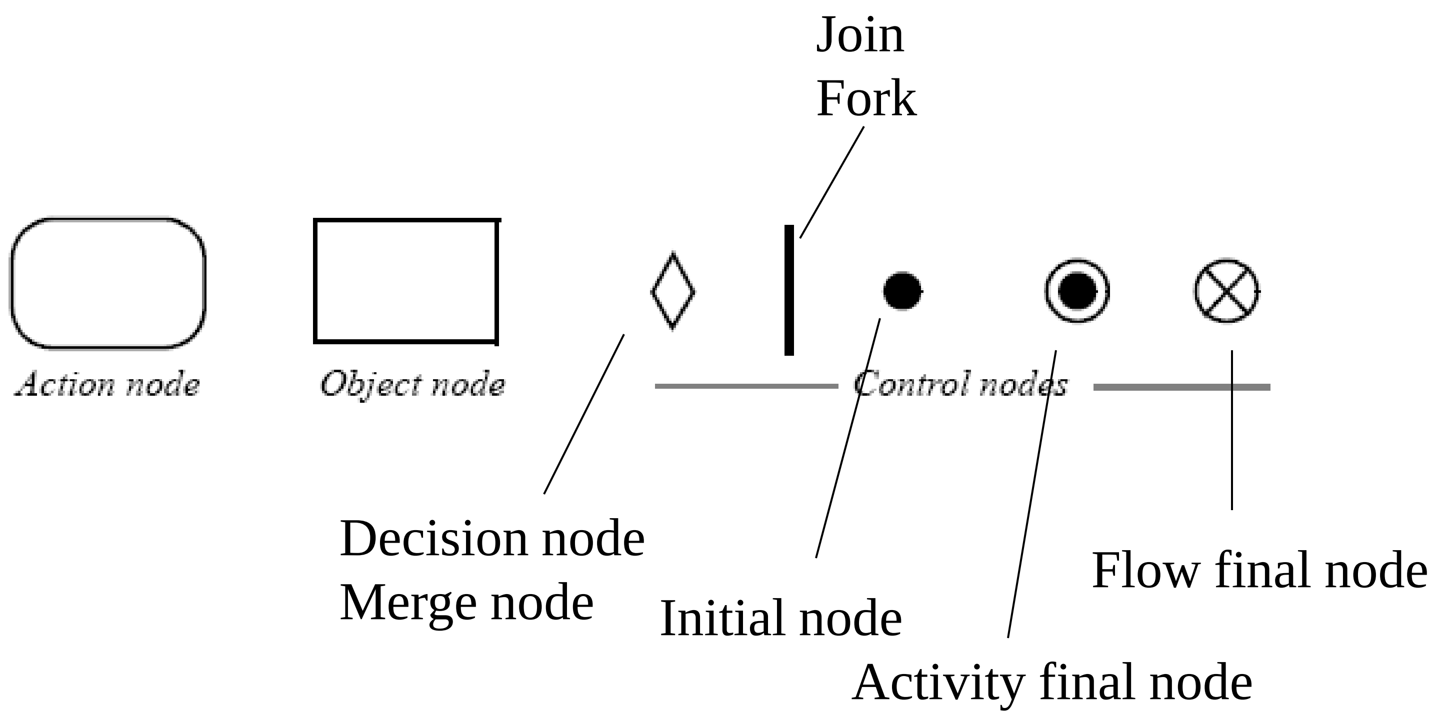
Modeling system workflows using activity diagrams

- Activity diagrams are used to model a process as a collection of nodes and edges between those nodes
- Use activity diagrams to model the behavior of:
  - √ workflows/business processes
  - – use cases
  - – operations and methods in classes

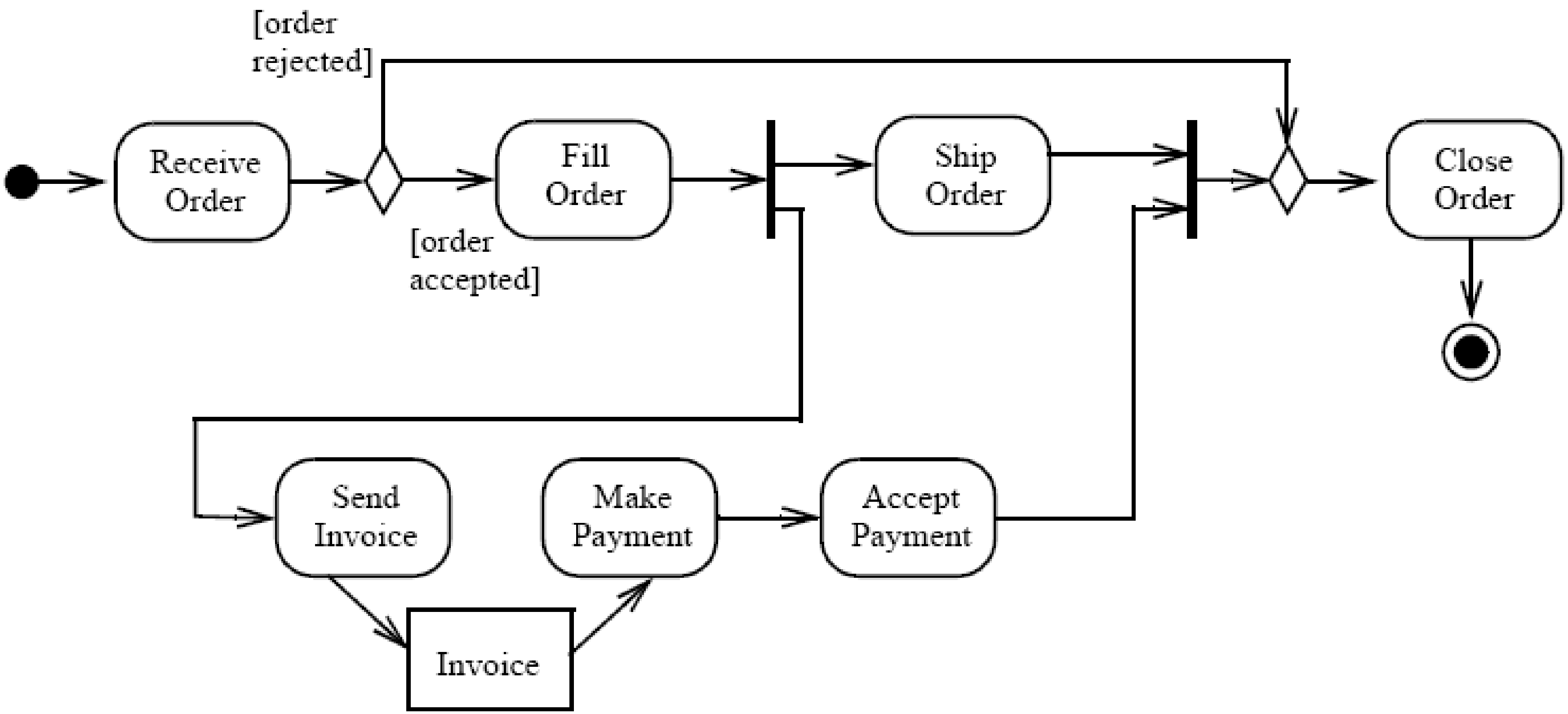
Activities

- 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

Key Activity Model symbols

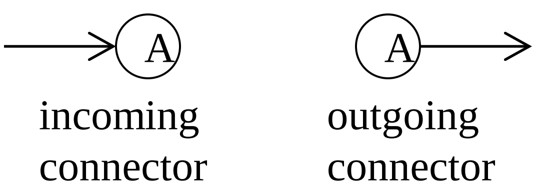


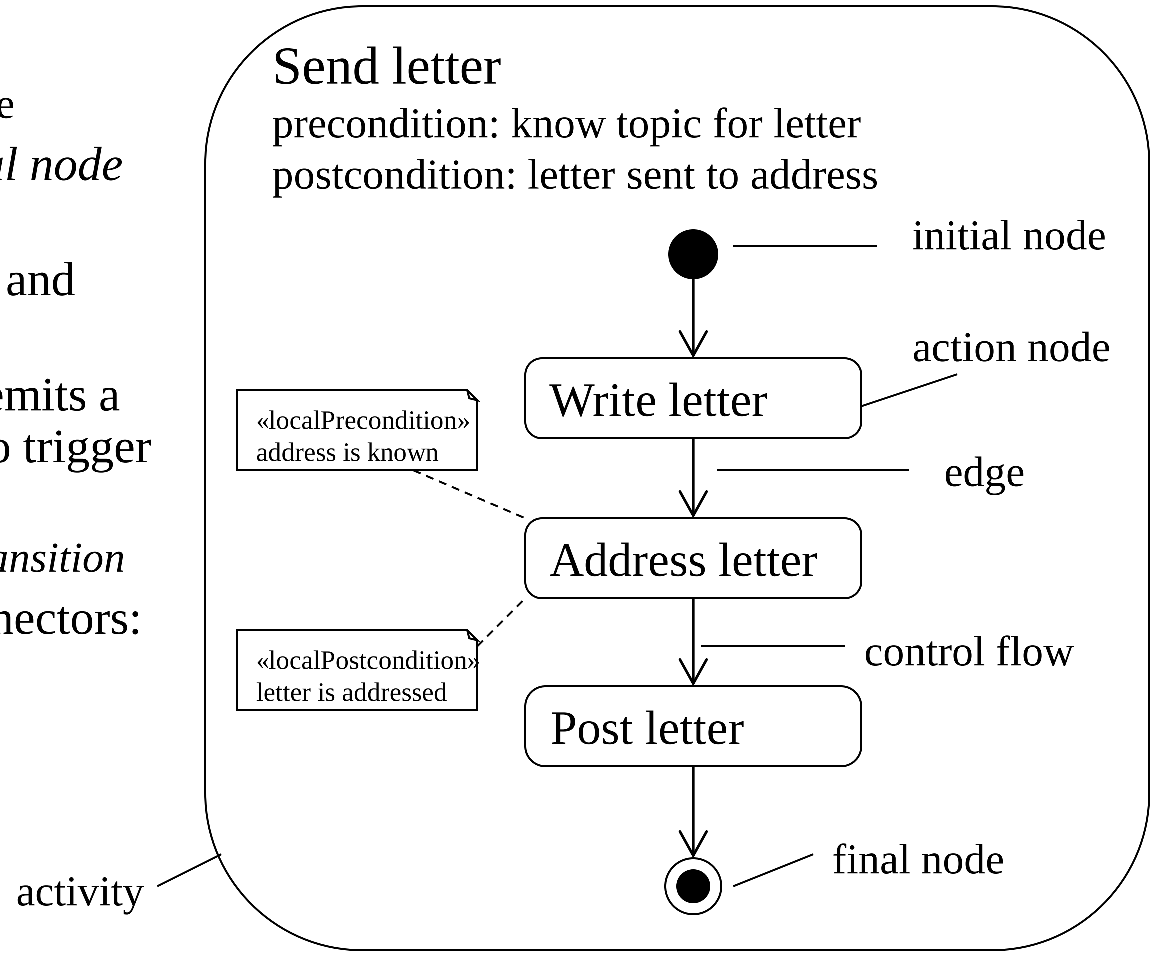
Simple example



Activity diagram syntax

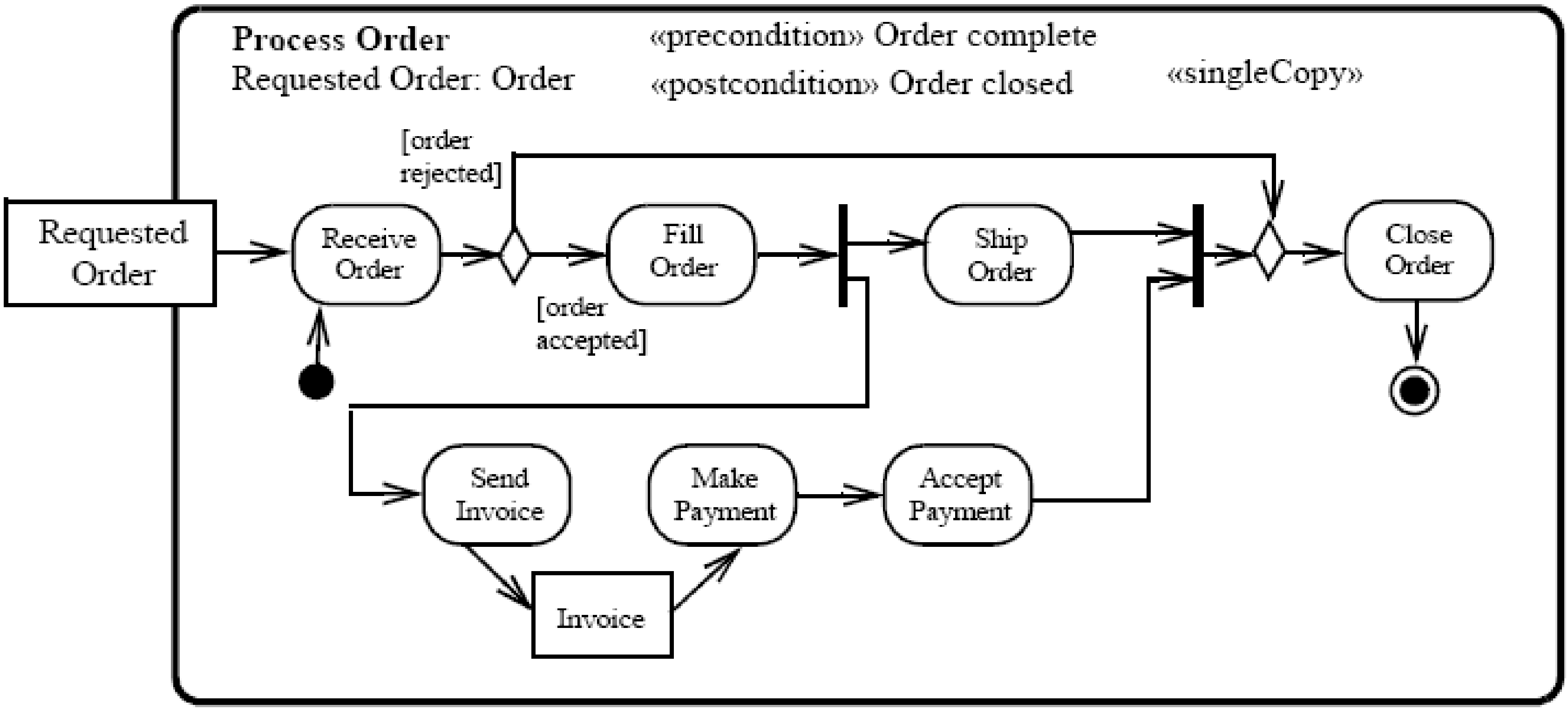
- Activities are networks of *nodes* connected by *edges*
  - The control flow is a type of edge
- Activities usually start in an *initial node* and terminate in a *final node*
- Activities can have preconditions and postconditions
- When an action node finishes, it emits a token that may traverse an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



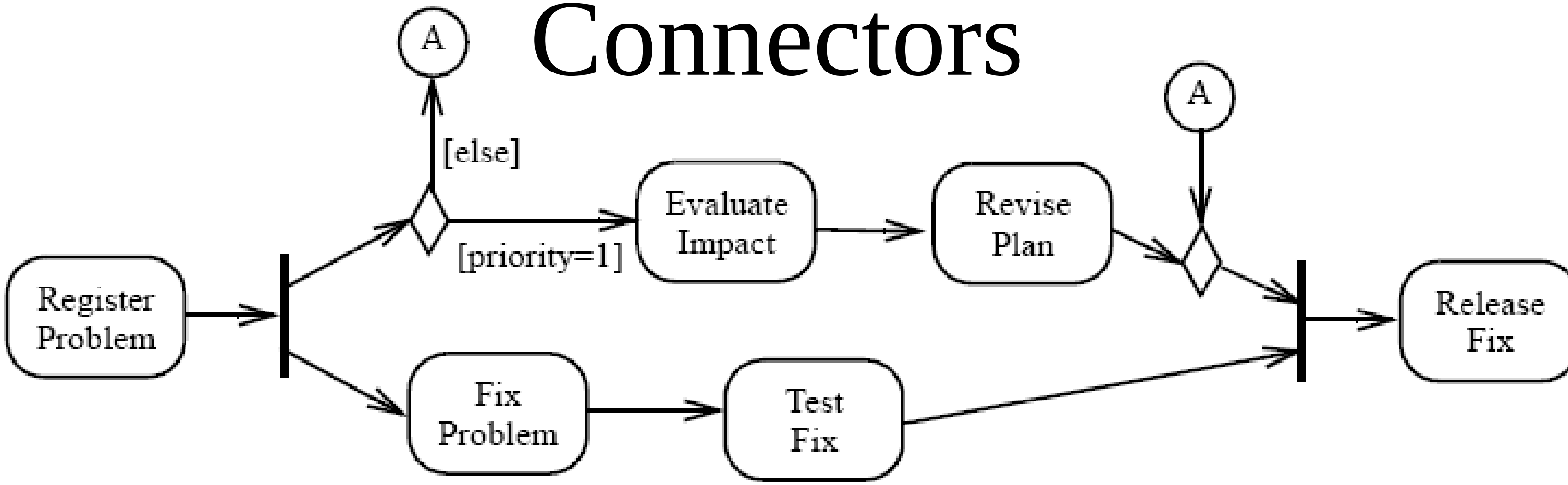


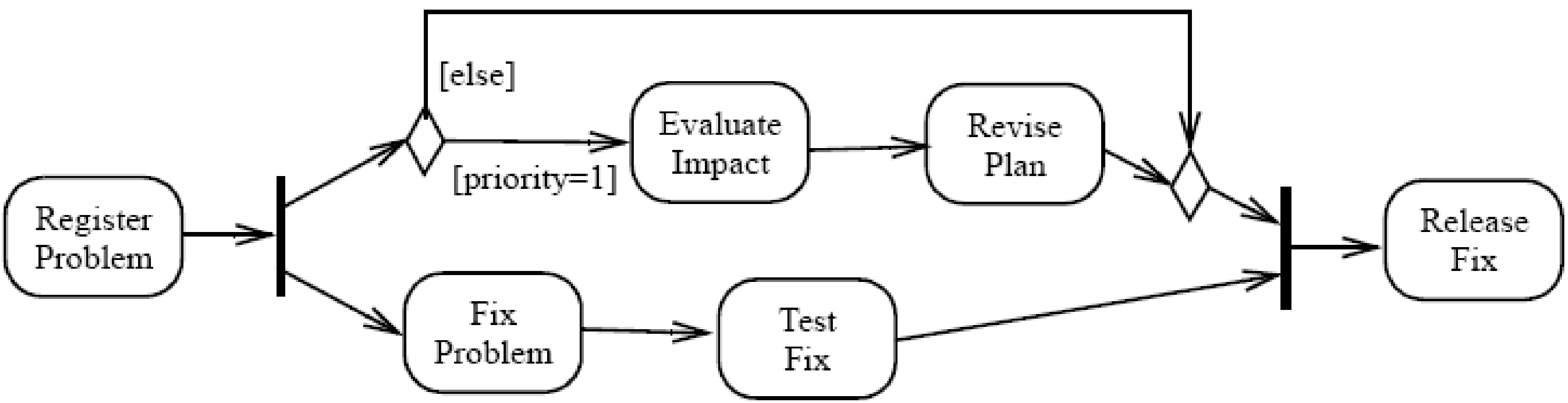
Modeling activities

An activity is a structure of actions

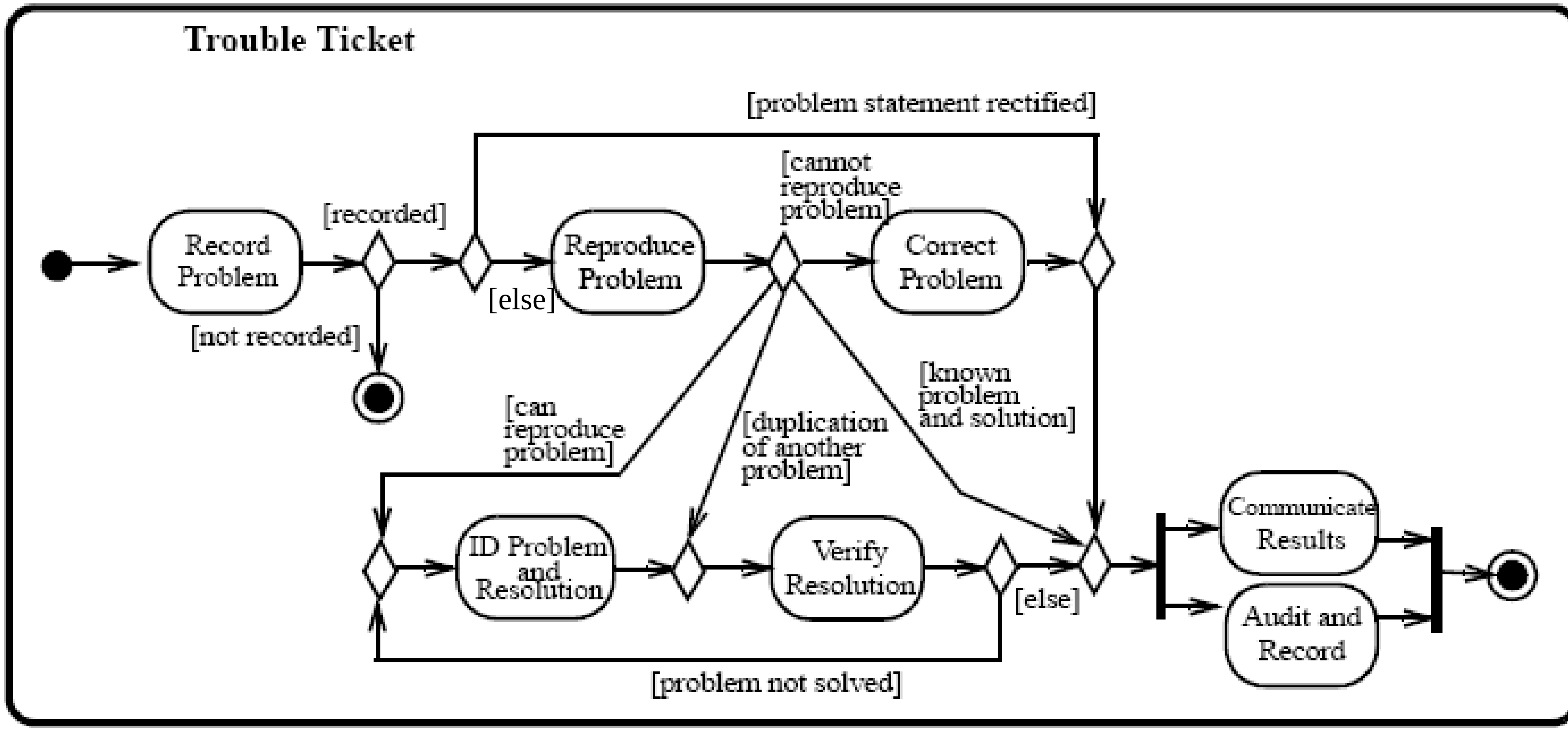


Connectors


A
A
[else]
[priority=1]
Register Problem
Evaluate Impact
Revise Plan
Release Fix
Fix Problem
Test Fix

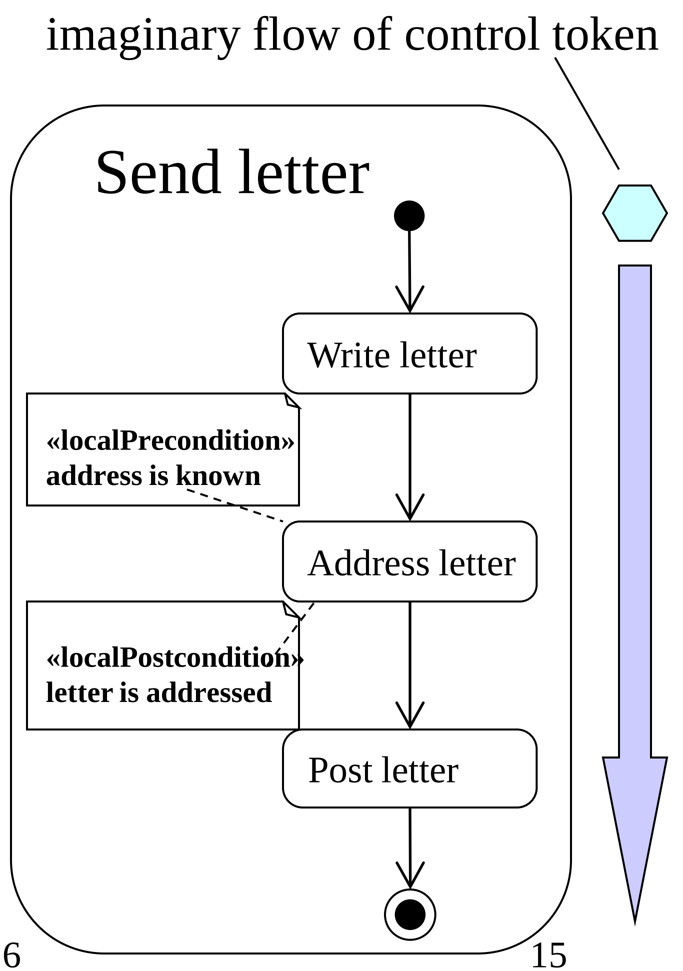

[else]
[priority=1]
Register Problem
Evaluate Impact
Revise Plan
Release Fix
Fix Problem
Test Fix

Another example



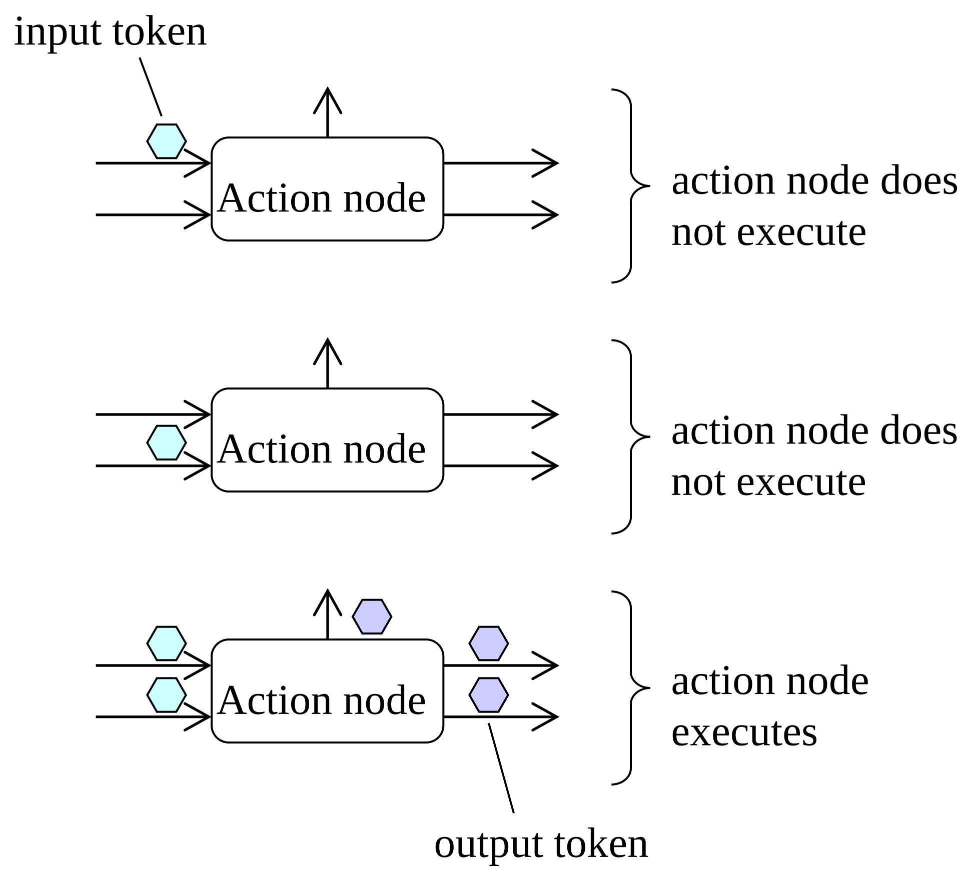
Activity diagram semantics

- 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

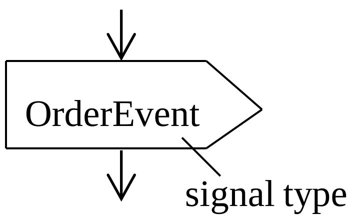
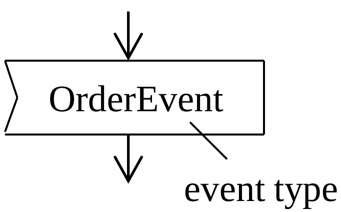
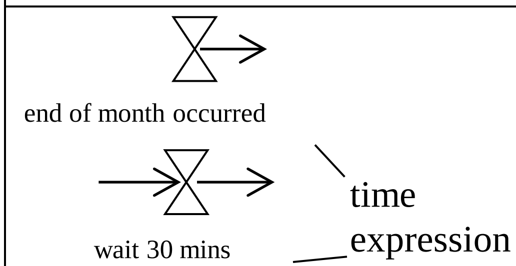


Action nodes

- 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

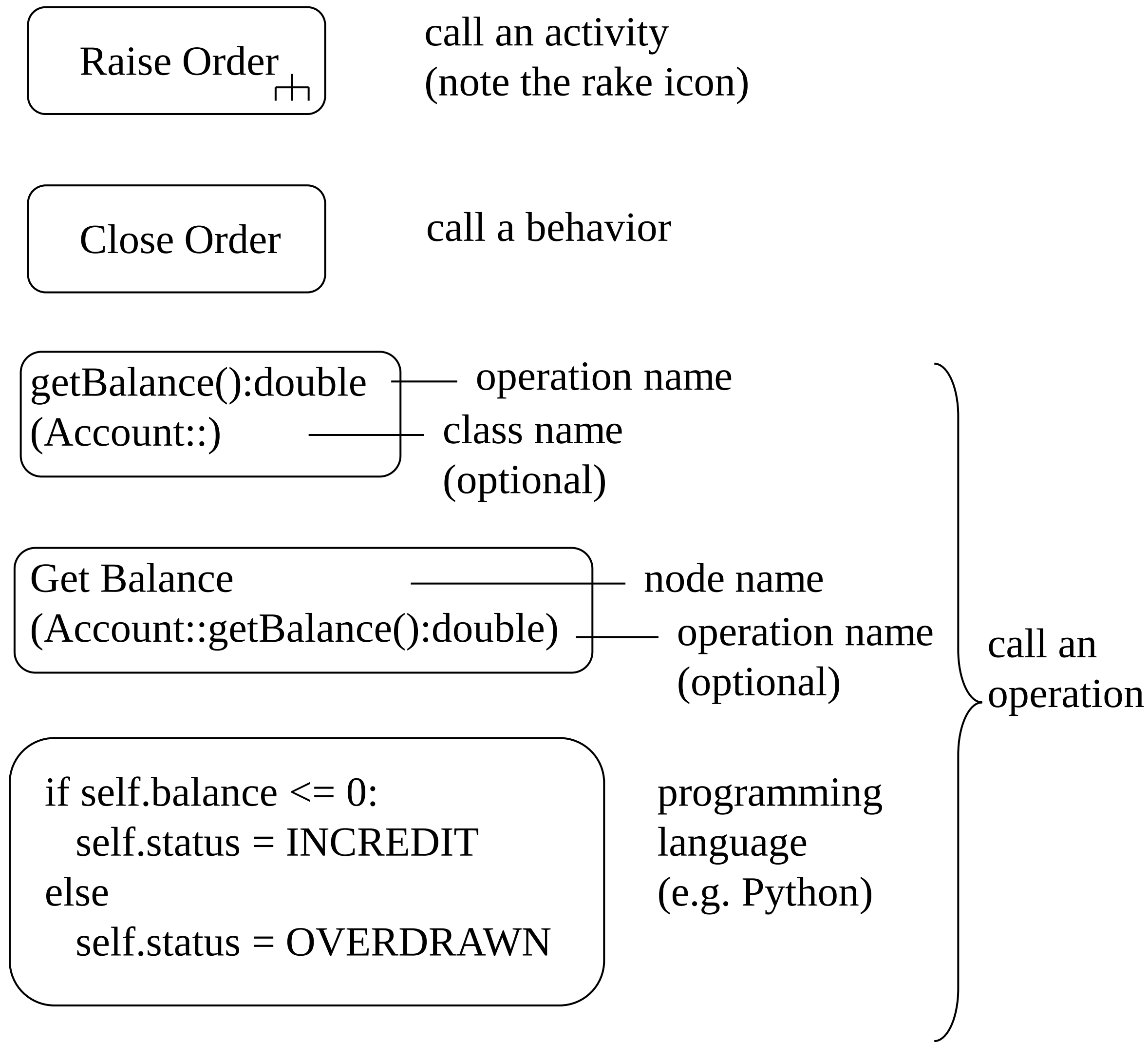


Types of action node

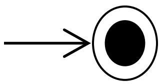
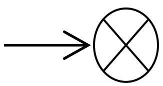
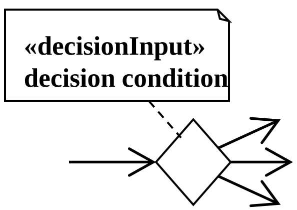
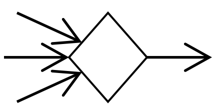
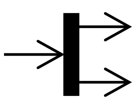
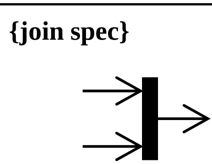
action node syntax	action node semantics
	Call action - invokes an activity, a behavior or an operation. The most common type of action node. See next slide for details.
	Send signal action - sends a signal asynchronously. The sender *does not* wait for confirmation of signal receipt. It may accept input parameters to create the signal
	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 *no* input edge it starts when its containing activity starts and is *always* enabled.
	Accept time event action - waits for a set amount of time. Generates time events according to it's time expression.

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

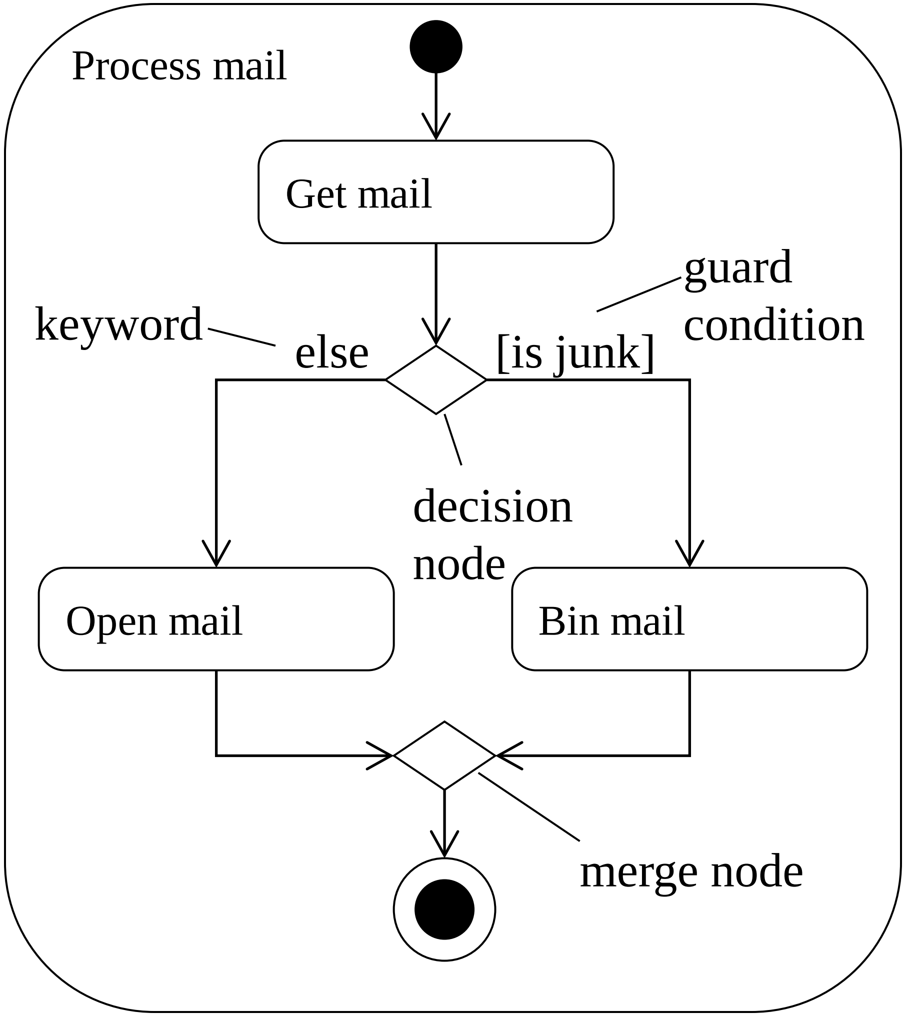


Control nodes

control node syntax	control node semantics	
	**Initial node – indicates where the flow starts when an activity is invoked**	
	**Activity final node – terminates an activity**	**Final nodes**
	**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 them for traversa** **May optionally have inputs defined by a «decisionInput»**	**See examples on next two slides**
	**Merge node – selects *one* 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**	

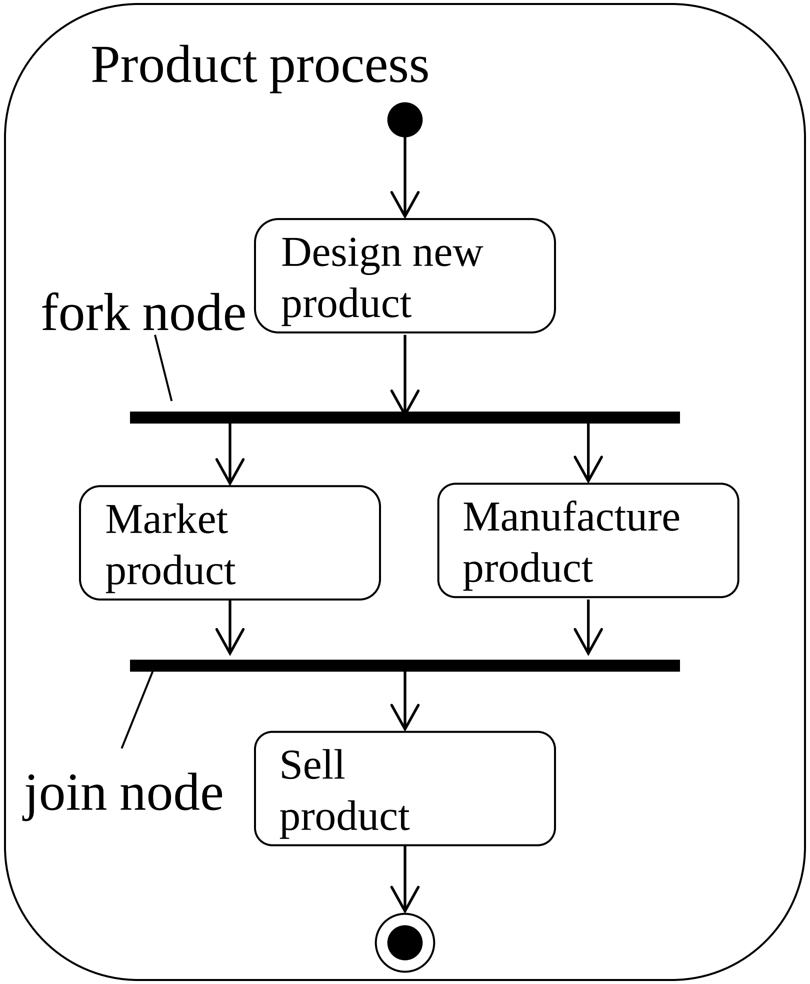
Decision and merge nodes

- 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

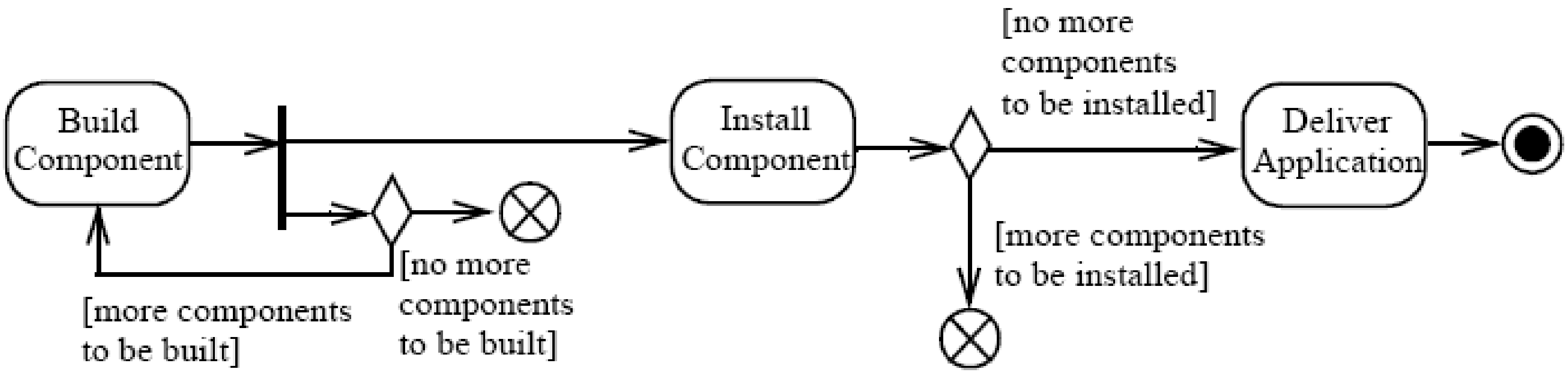


Fork and join nodes - concurrency

- 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

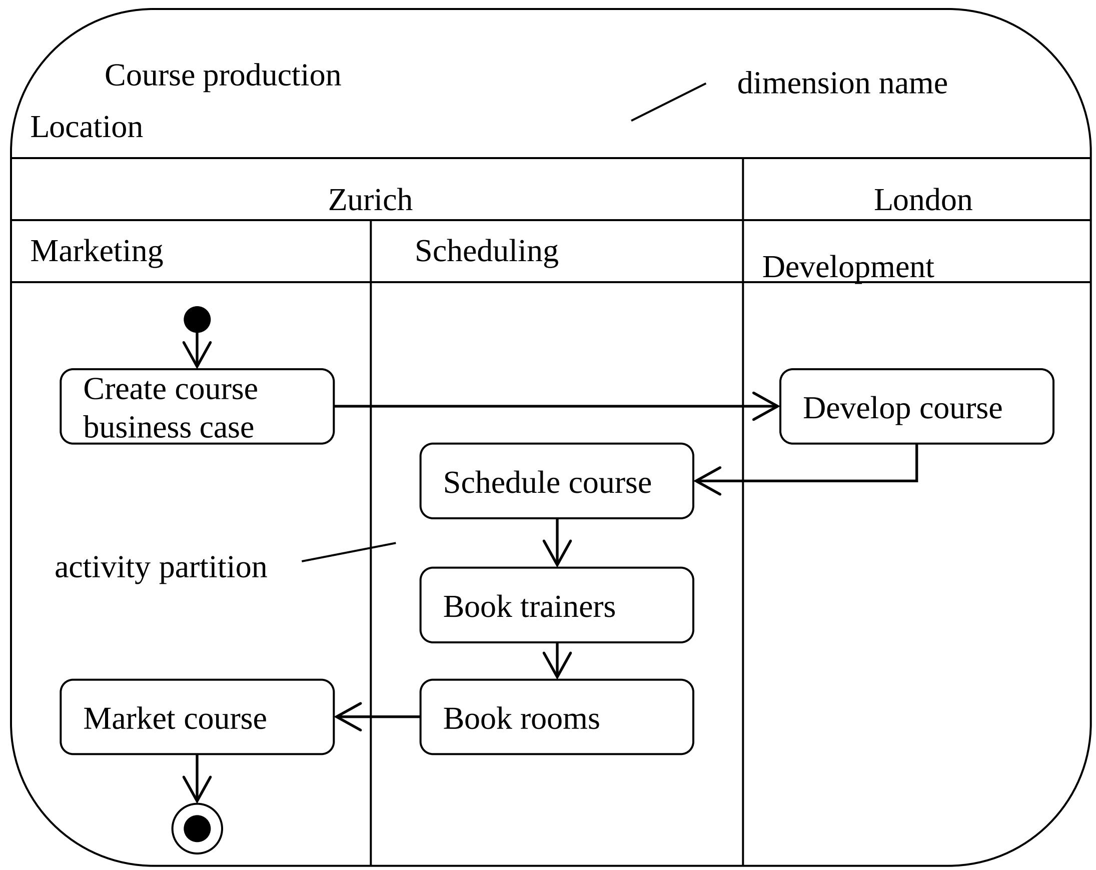


Activity Final Nodes vs. Flow Final Nodes

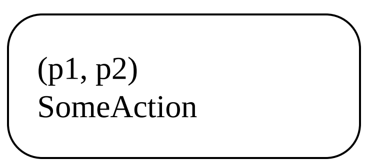


Activity partitions

- 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

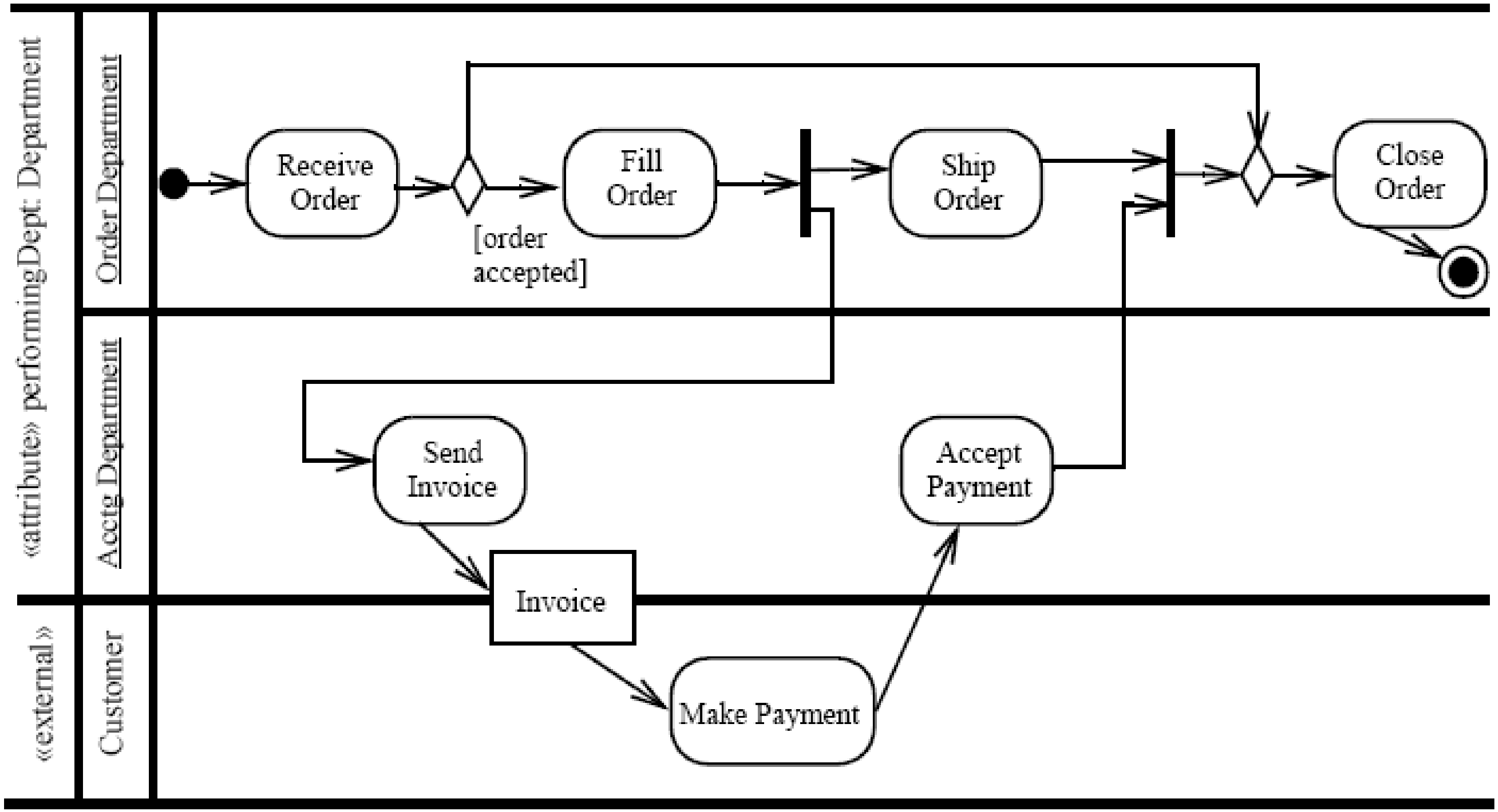


nested partitions

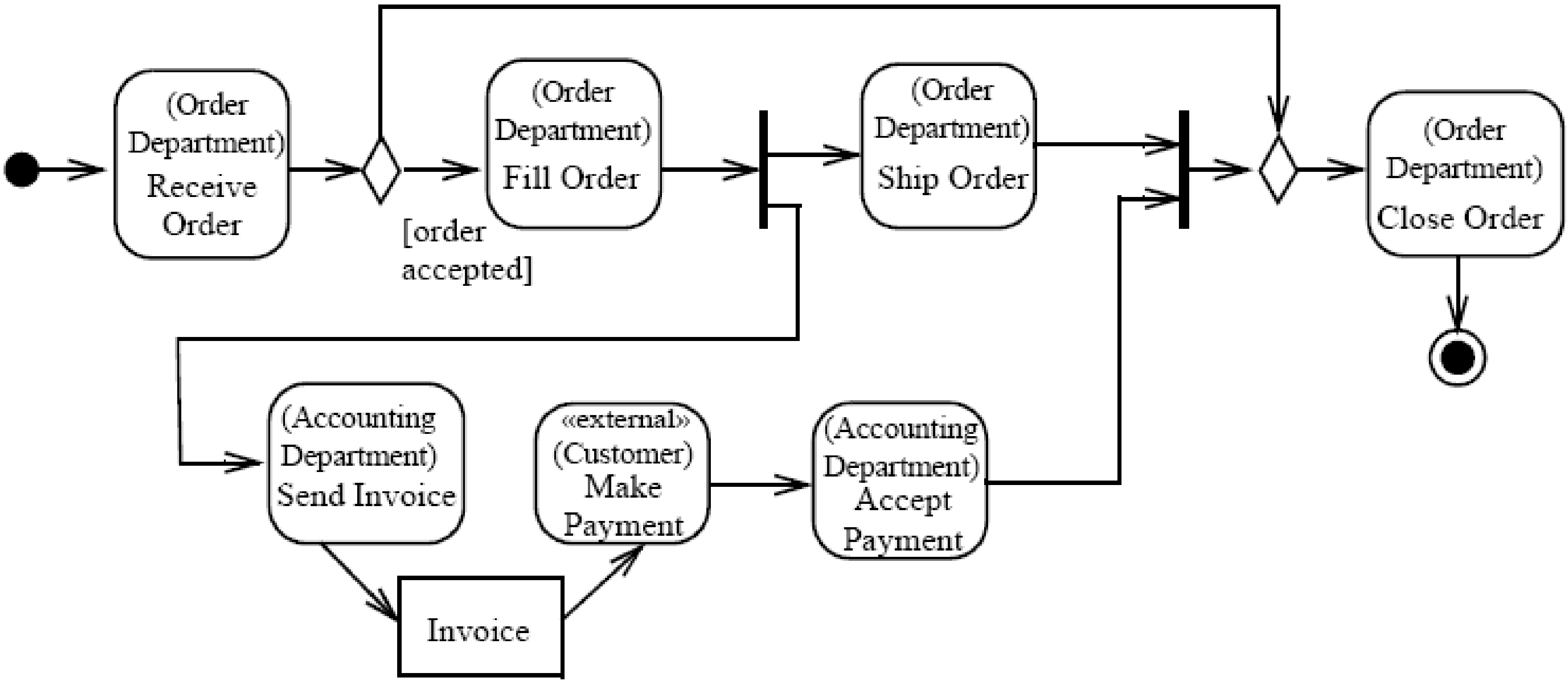


multiple partitions

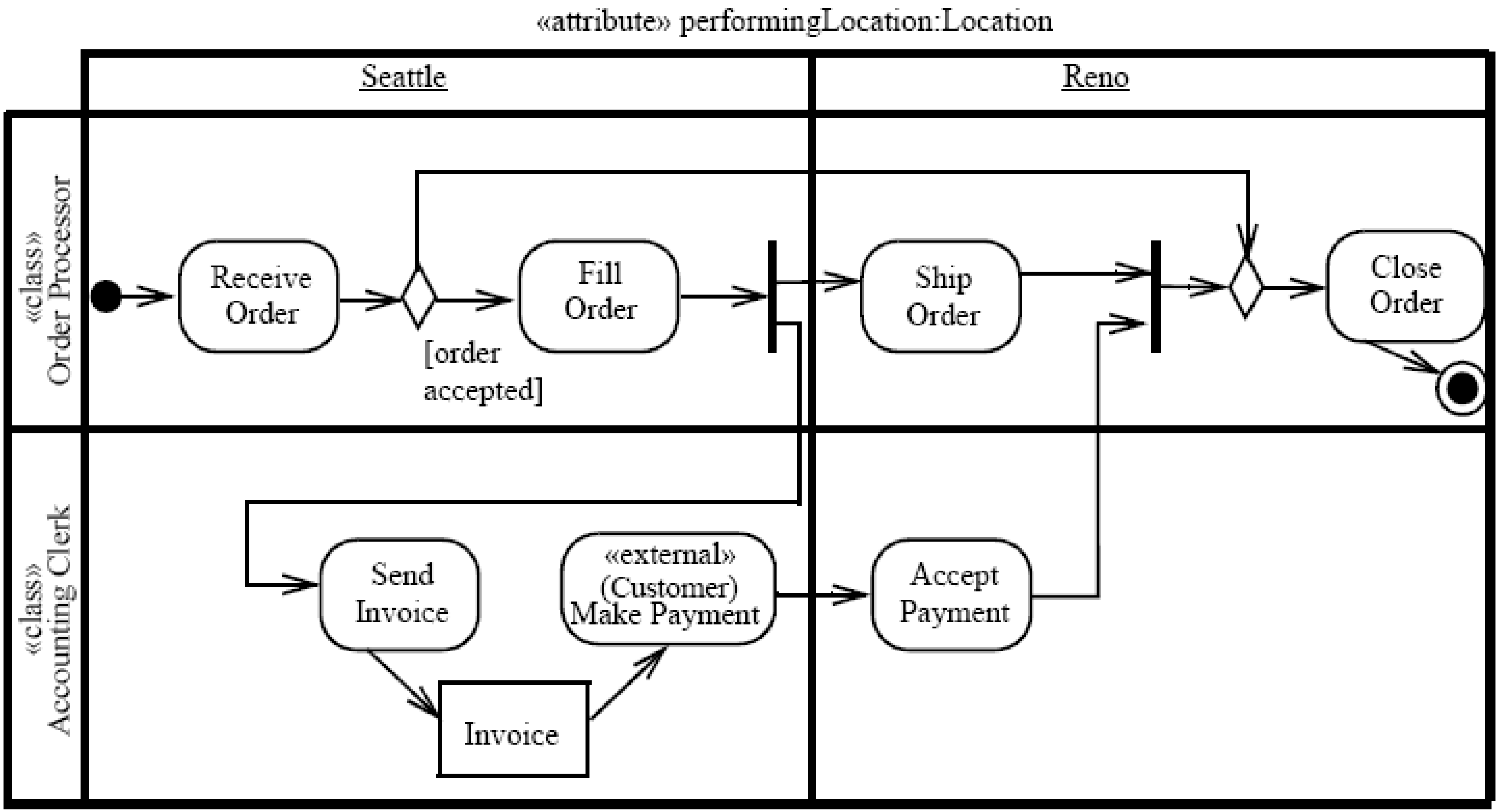
Partitions/Swimlanes



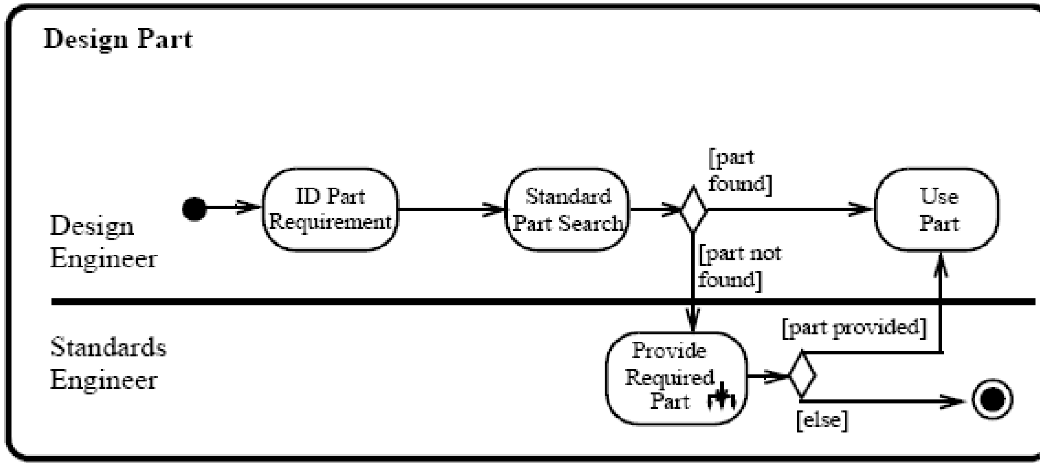
Partitions using annotations

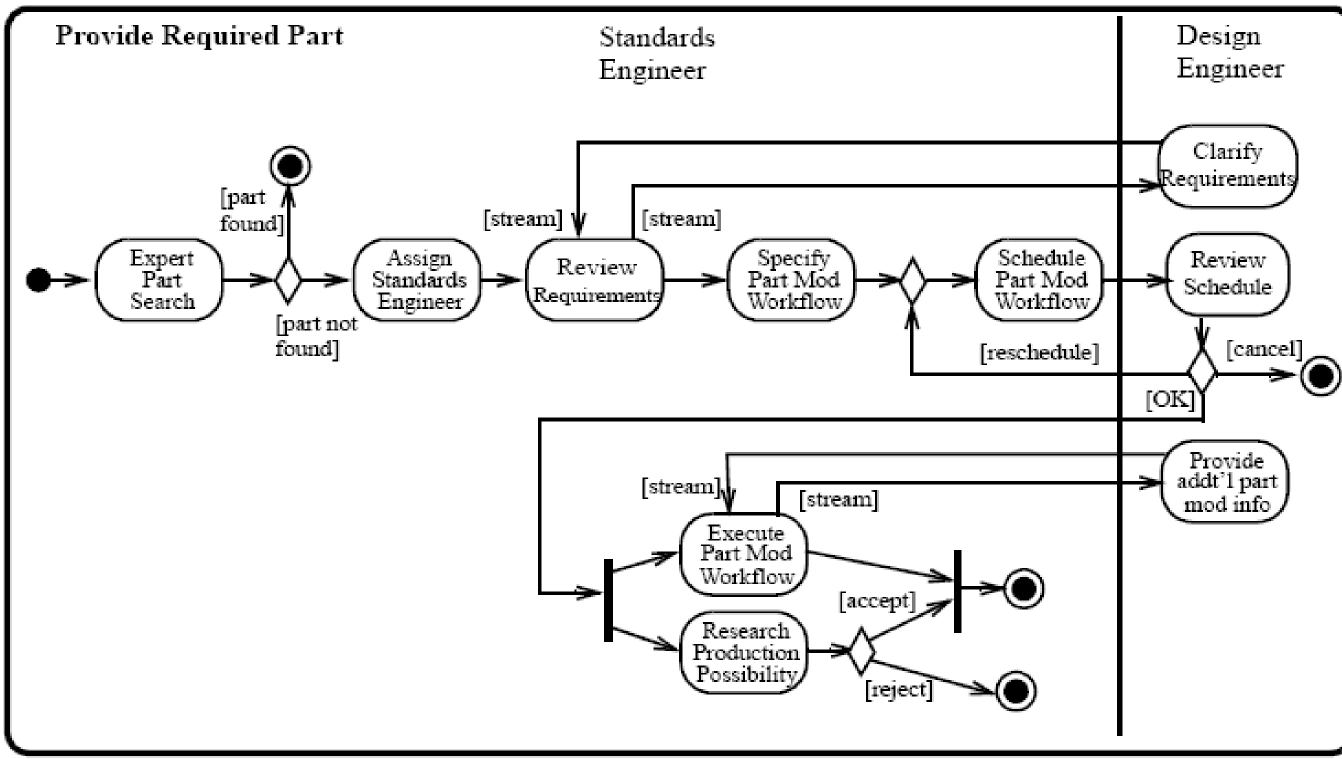


Dimensional partitions



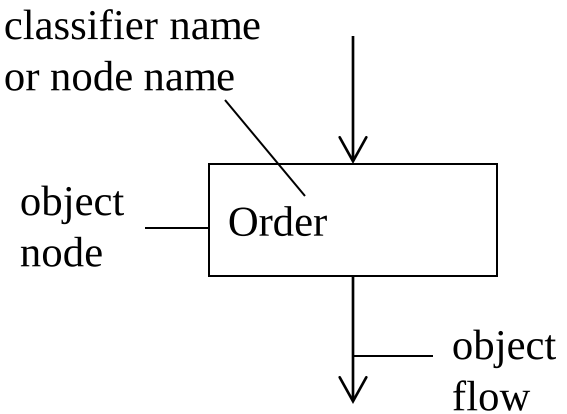
Expanding activities

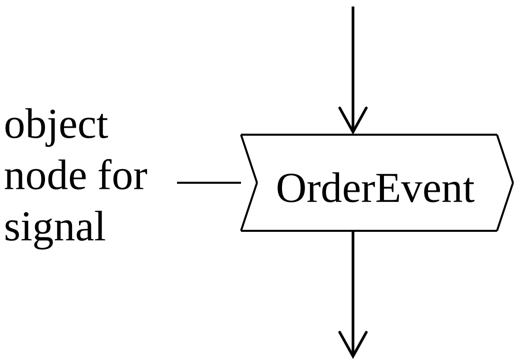




Object nodes

- 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)
  - LIFI – last in, first out
  - Modeler defined – a selection criterion is specified for the object node



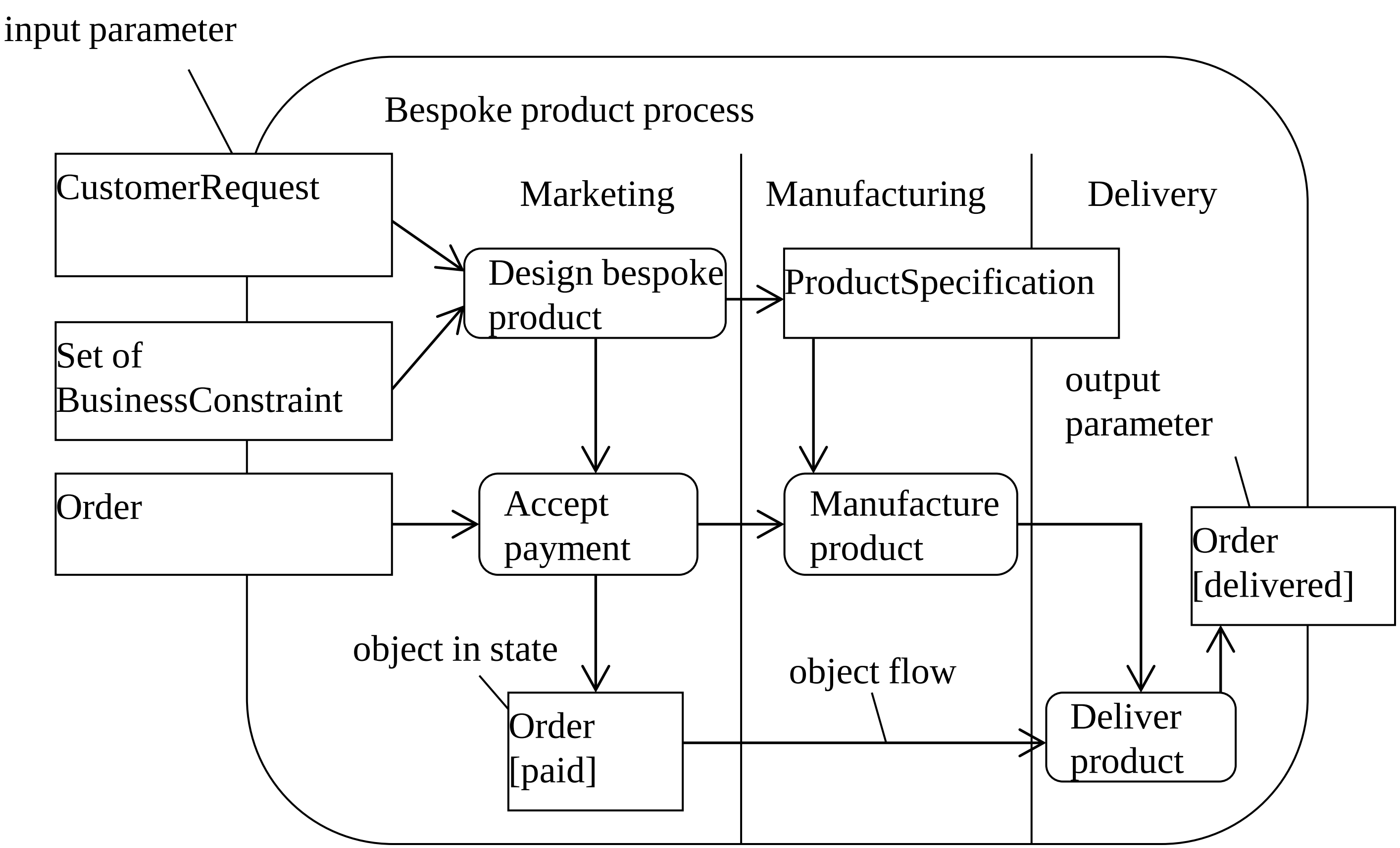


Object node syntax

- Object nodes have a flexible syntax. You may show:
  - upper bounds
  - ordering
  - sets of objects
  - selection criteria
  - object in state

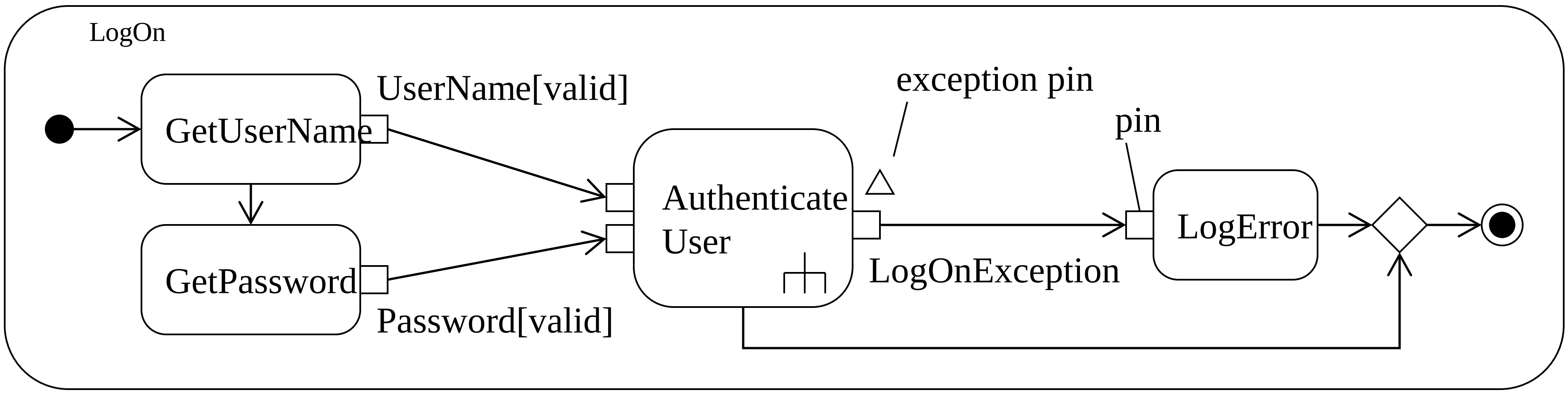
	Order	order objects may be available
	Order {upperBound = 12}	zero to 12 Order objects may be available
	Order {ordering = LIFO}	last Order object in is the first out (FIFO is the default)
	Set of Order	sets of Order objects may be available
«selection» monthRaised = "Dec"	Order	Order objects raised in December may be available
	Order [open]	select Order objects in the open state

Activity parameters

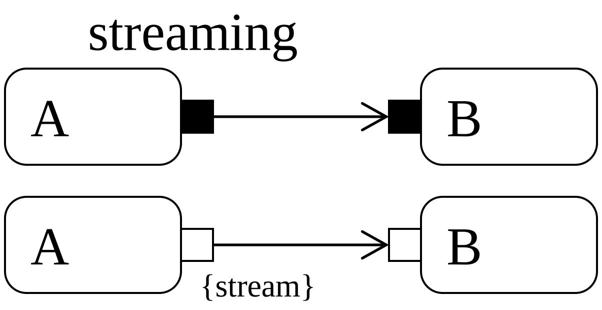


- Object nodes can provide input and output parameters to activities
  - Input parameters have one or more output object flows into the activity
  - Output parameters have one or more input object flows out of the activity
- Draw the object node overlapping the activity boundary

Pins

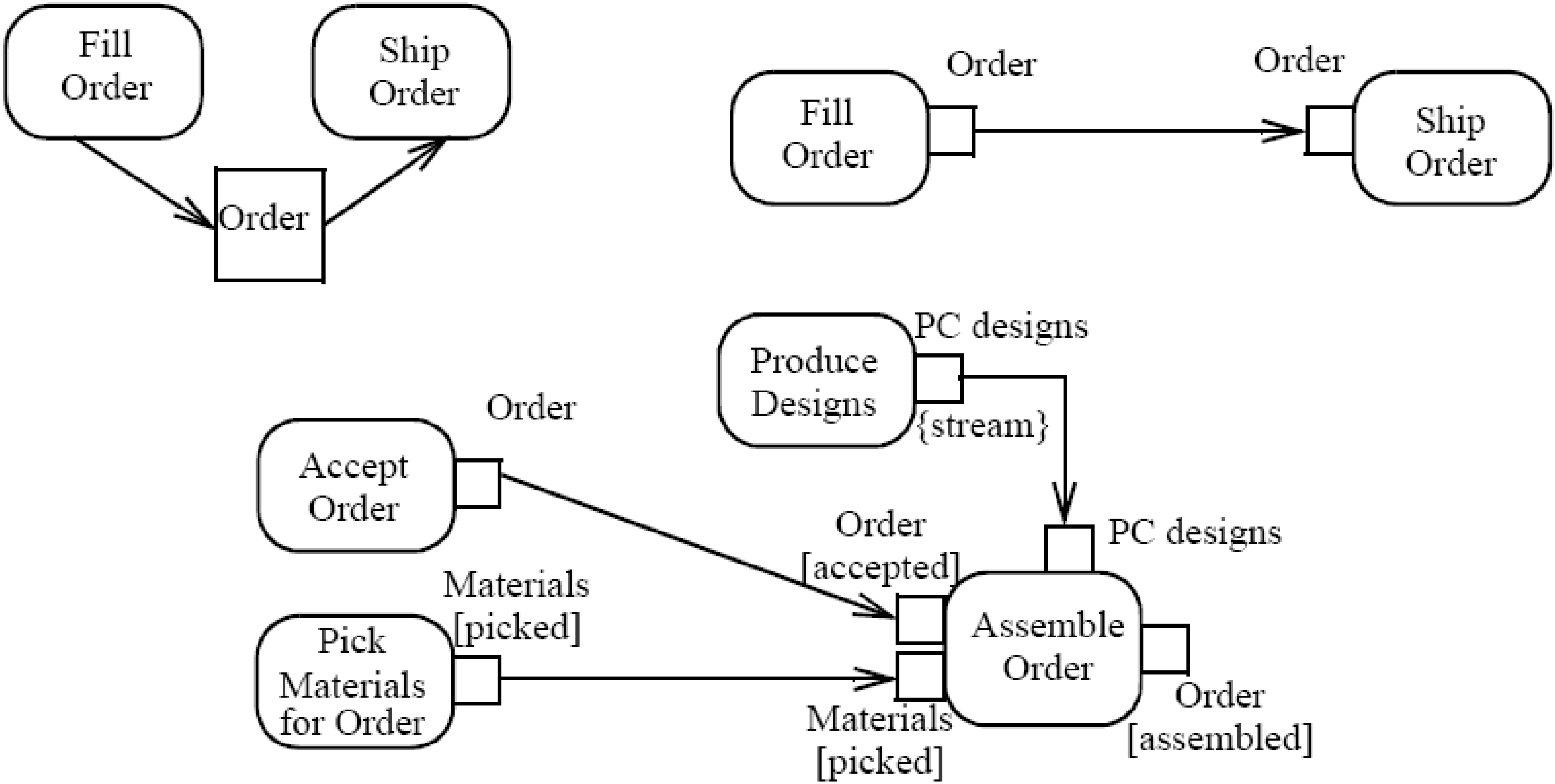


- Pins are object nodes for inputs to, and outputs from, actions
  - Same syntax as object nodes
  - Input pins have exactly one input edge
  - Output pins have exactly one output edge
  - Exception pins are marked with an equilateral triangle
  - Streaming pins are filled in black or marked with {stream}

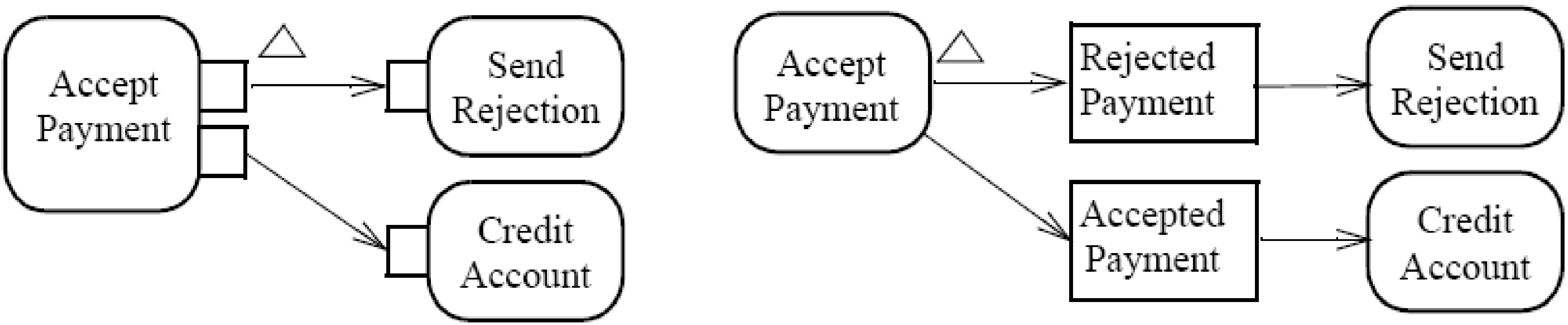


Input/Output pins

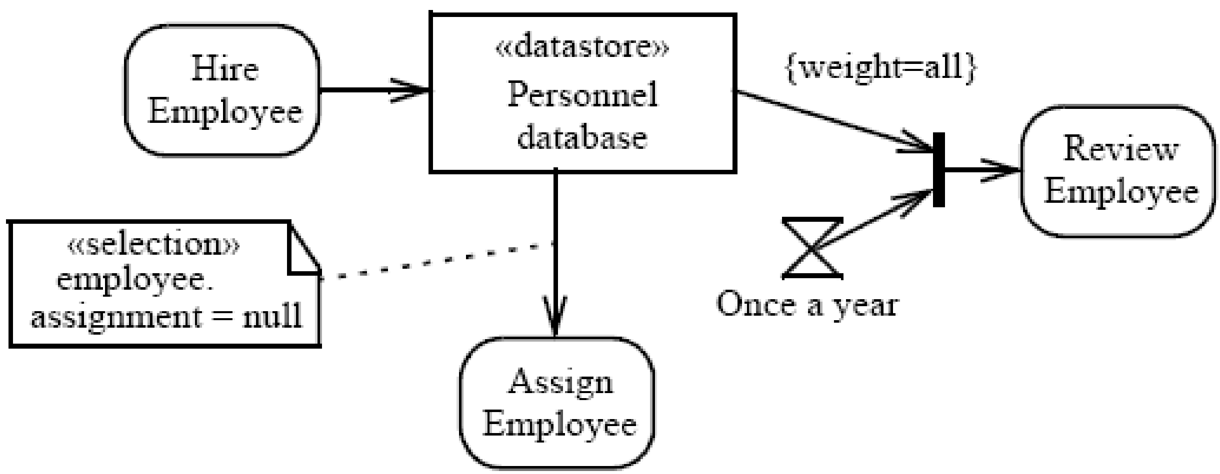
A pin represent an input or output data node



Exceptions

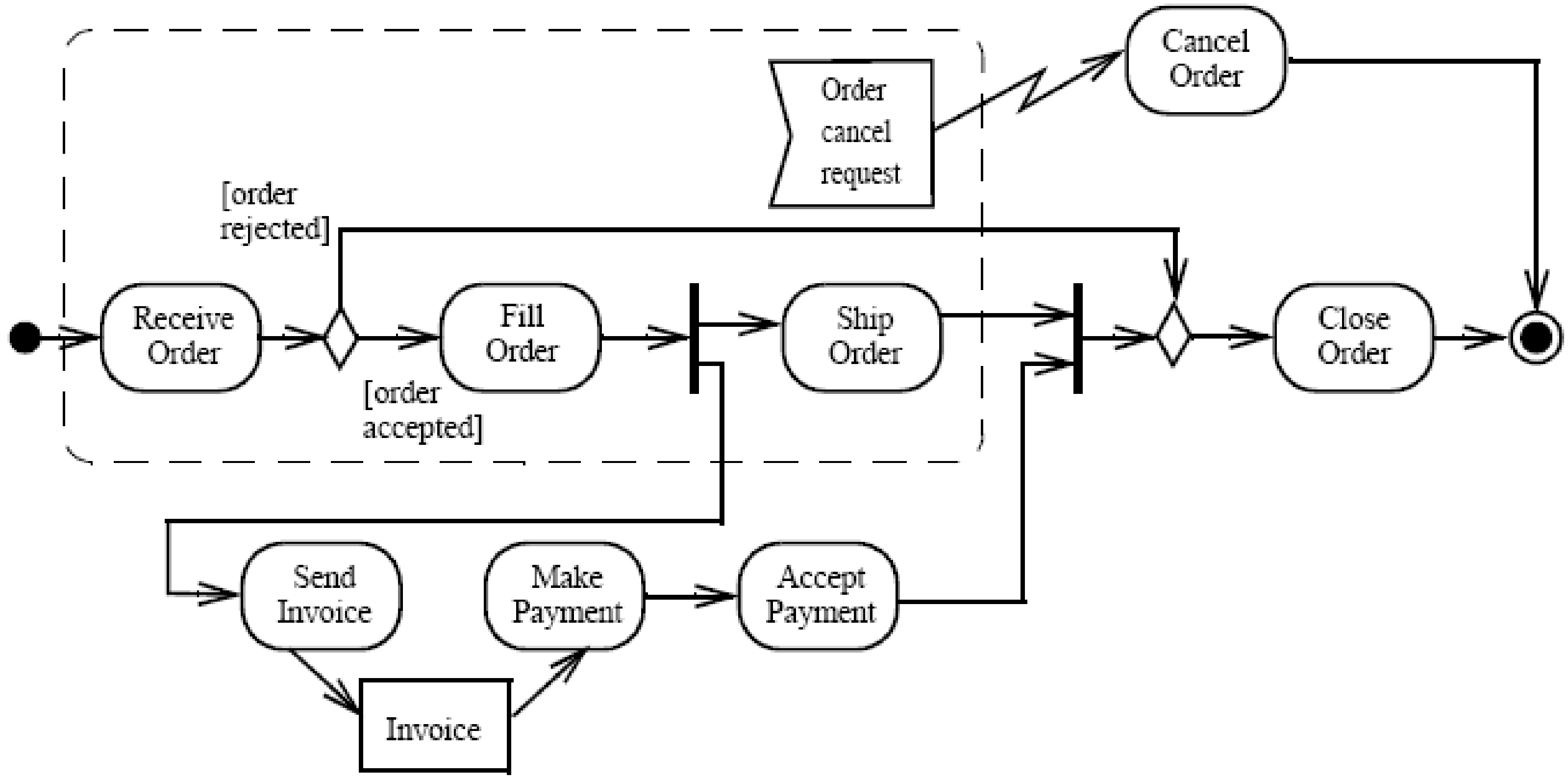

Accept
Payment
Send
Rejection
Credit
Account
Accept
Payment
Rejected
Payment
Accepted
Payment
Send
Rejection
Credit
Account

Timers



Hire
Employee
«datastore»
Personnel
database
{weight=all}
Review
Employee
«selection»
employee.
assignment = null
Once a year
Assign
Employee

Interrupts


Cancel Order
Order cancel request
[order rejected]
Receive Order
Fill Order
Ship Order
Close Order
[order accepted]
Send Invoice
Make Payment
Accept Payment
Invoice

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