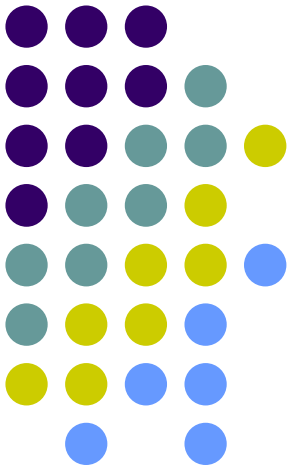
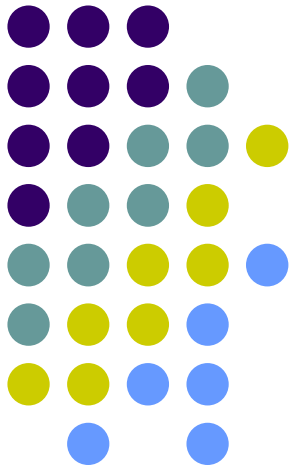
Unit V:
Activity Diagram, State Diagram, Timing Diagram

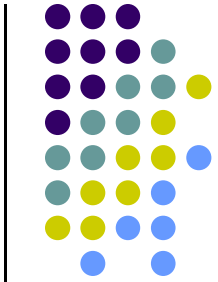
Deepali Londhe



Activity diagrams in UML 2.0



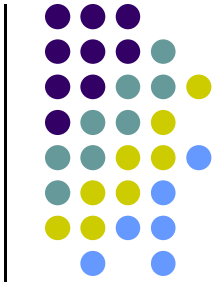
Activity Diagrams



- to model the dynamic aspects of a system
- Model control and information flow of a procedure or process. This involves modeling the sequential (and possibly concurrent) steps in a computational process.
- you can also model the flow of an object as it moves from state to state at different points in the flow of control.
- to visualize, specify, construct, and document the dynamics of a society of objects, or they may be used to model the **flow of control** of an operation.
- Can also be used to model control flow within a task method (knowledge model)

Activity diagrams in UML 2.0

Contents:

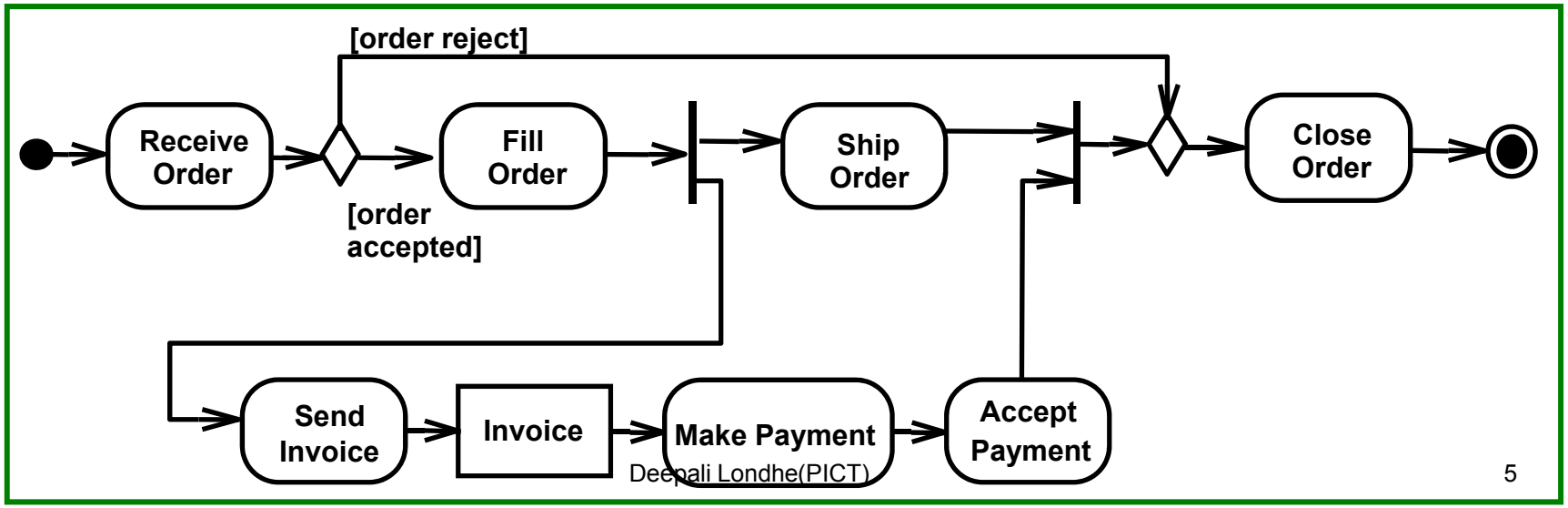


- Introduction to UML 2.0 *Activity Diagrams*
- Activities, Sub activities
- Concepts of Action, Pins and Activity
- Description of activity nodes and activity edges
- New notations
  - ActivityPartition
  - Pre & post condition
  - SendSignalAction
  - Time triggers and Time events
  - AcceptEventAction
  - InterruptibleActivityRegion
  - Exception
  - ExpansionRegion

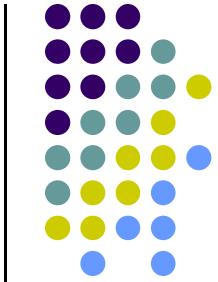
Activity diagrams

- Useful to specify software or hardware system behaviour
- Based on data flow models – a graphical representation (with a Directed Graph) of how data move around an information system

Activity Diagram elements UML 1.x



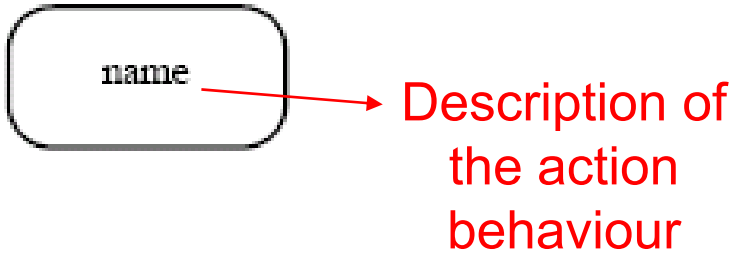
Some definitions

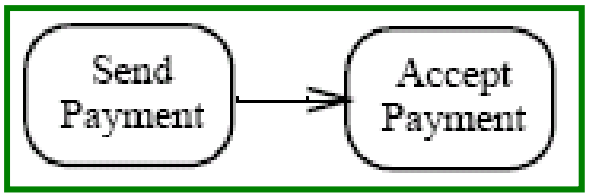


- ***Flow***: permits the interaction between two nodes of the activity diagram (represented by edges in activity diagram)
- ***State***: a condition or situation in the life of an object during which it satisfies some conditions, performs some activities, or waits for some events
- ***Type***: specifies a domain of objects together with the operations applicable to the objects (also none); includes primitive built-in types (such as integer and string) and enumeration types
- ***Token***: contains an object, datum, or locus of control, and is present in the activity diagram at a particular node; each token is distinct from any other, even if it contains the same value as another
- ***Value***: an element of a type domain

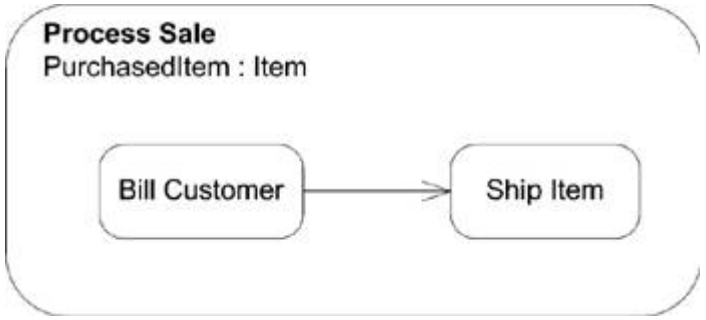
Actions

- The fundamental unit of executable functionality in an activity
- The execution of an action represents some transformations or processes in the modeled system (creating objects, setting attribute values, linking objects together, invoking user-defined behaviours, etc.)

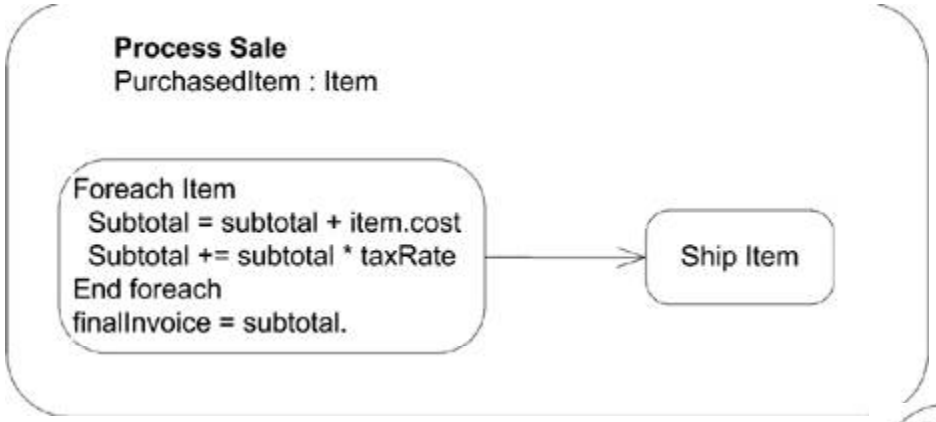




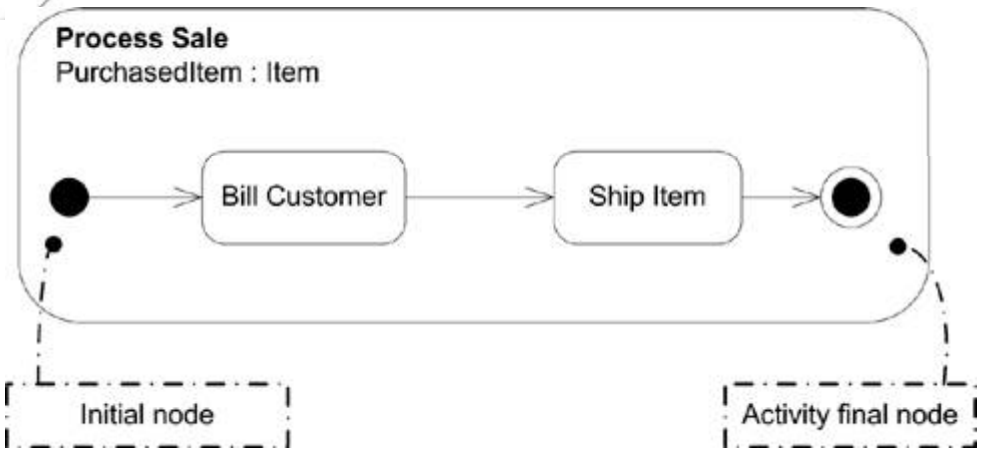
A simple activity with no details



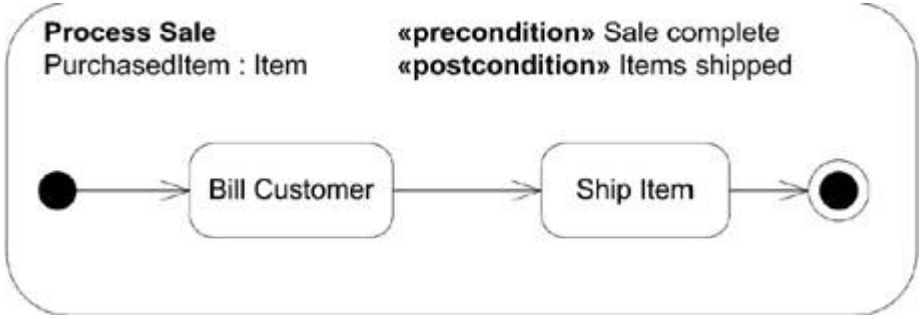
Activity with simple details



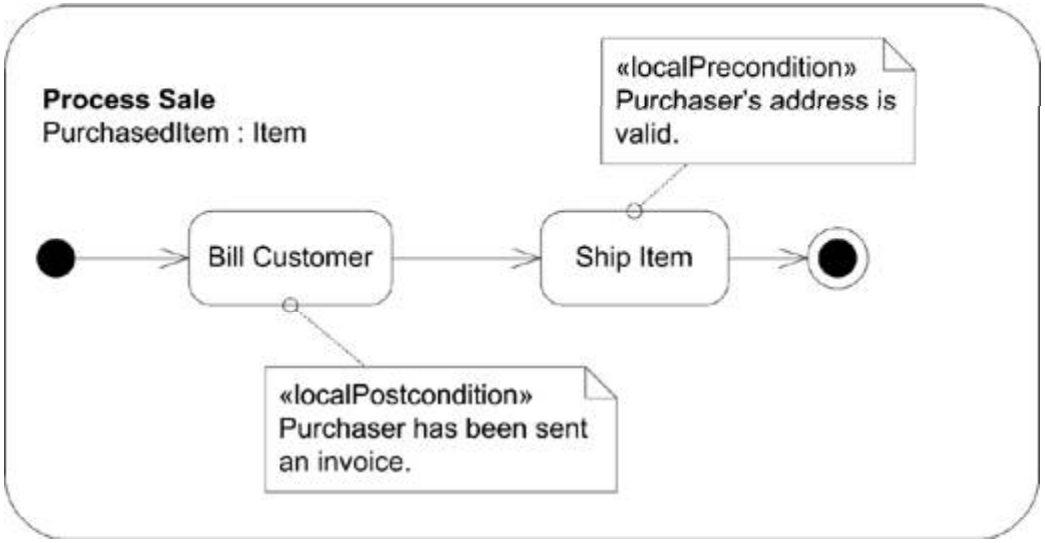
An action with pseudocode



An activity diagram with initial and final nodes



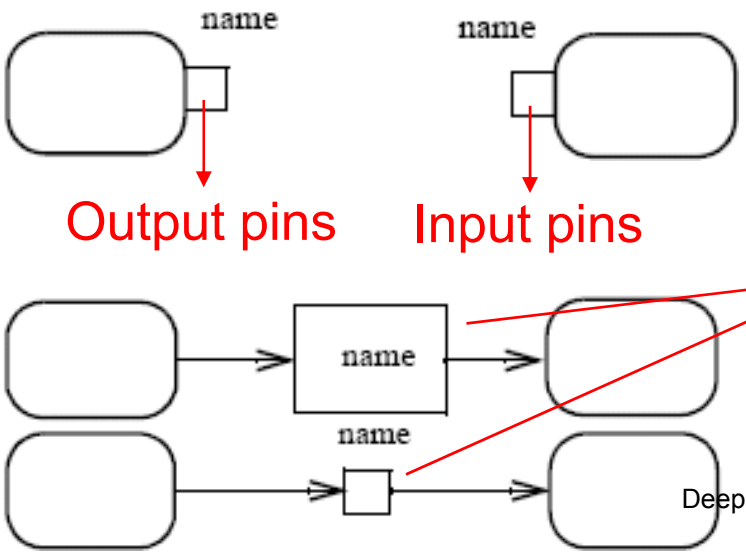
Activity diagram with pre- and post conditions

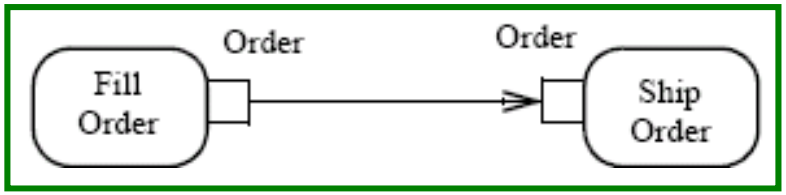


An activity diagram with local pre- and post conditions for actions

Pins

- Actions can have inputs and outputs, through the pins
- Hold inputs to actions until the action starts, and hold the outputs of actions before the values move downstream
- The name of a pin is not restricted: generally recalls the type of objects or data that flow through the pin



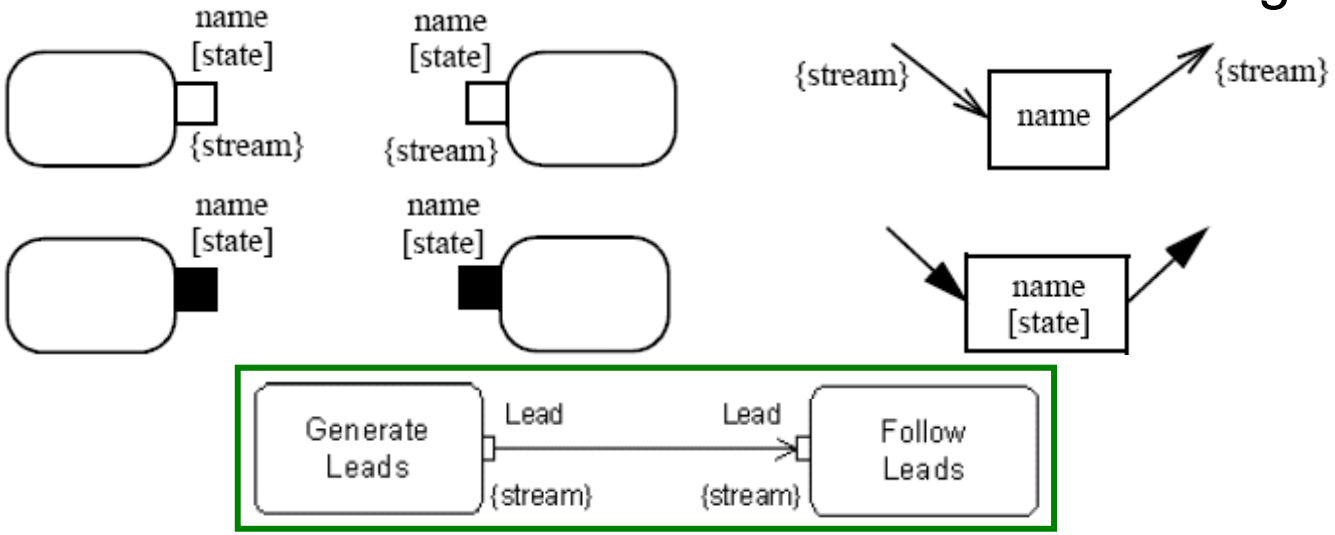


Standalone pin notations: the output pin and the input pin have the same name and same type

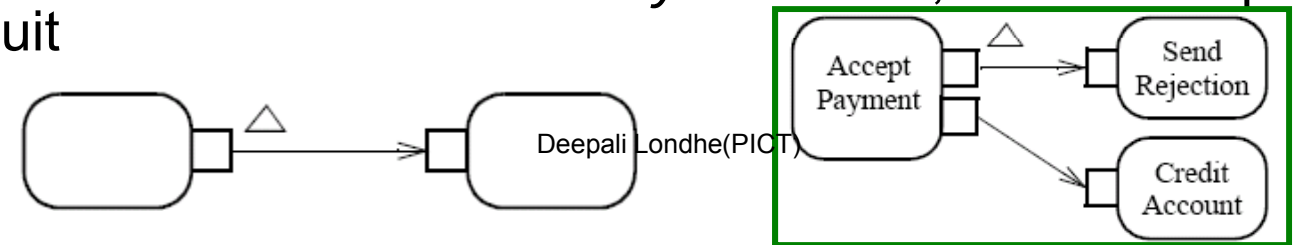
Special kind of pins (1)

- **Streaming Parameters** (notated with {STREAM}): accept or provide one or more values while an action is executing

Stand-alone style

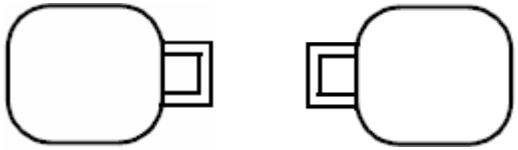


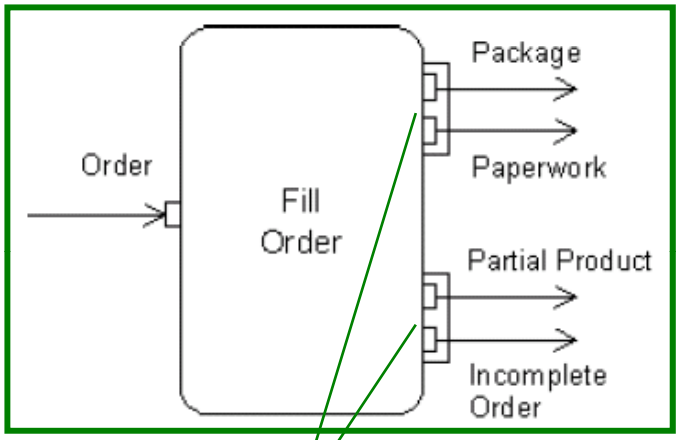
- **Exception Output Parameters** (notated with a triangle)
  - Provide values to the exclusion of any other output parameter or outgoing control of the action
  - The action *must immediately terminate*, and the output cannot quit



Special kind of pins (2)

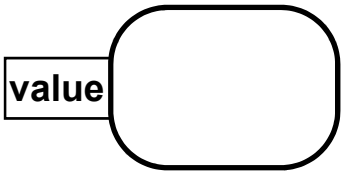
- **Parameter Sets**
  - group of parameters
  - the action can only accept inputs from the pins in one of the sets
  - the action can only provide outputs to the pins in one of the sets

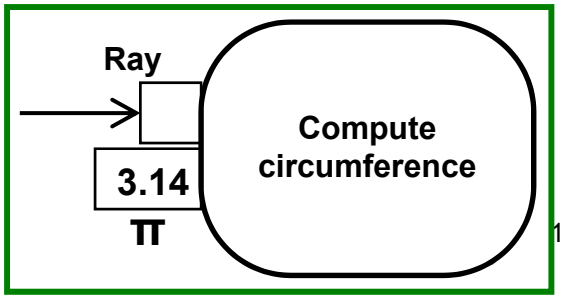




Output is provided only to the first or to the second set

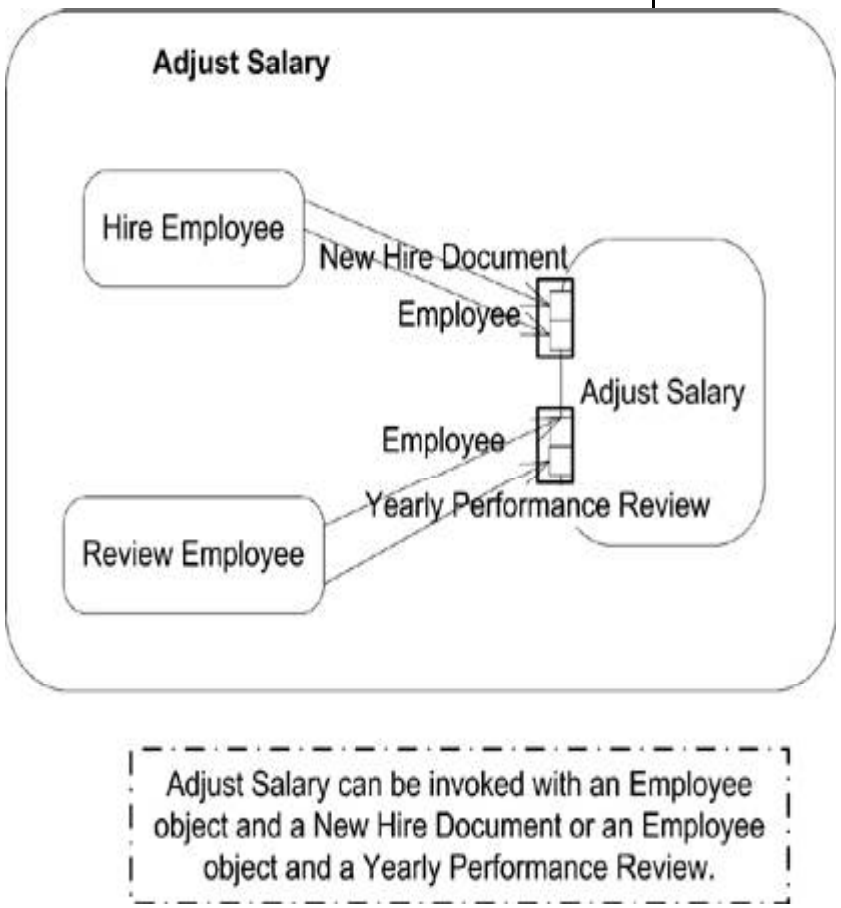
- **ValuePin**: special kind of input pin defined to provide constant values (the type of specificated value must be compatible with the type of the value pin)



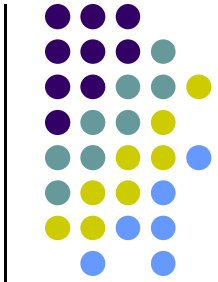


Parameter set

- There are times when an action may accept more than one type of valid input to begin execution.
- For example, a human resources action named Adjust Salary may require an Employee object and either a New Hire Document or Yearly Performance Review Document, but not both. You can show groups and alternatives for input pins (see "Pins" under "Object Nodes") by using *parameter sets.*
- A parameter set groups one or more pins and indicates that they are all that are needed for starting or finishing an action.



Conditions to start and end actions

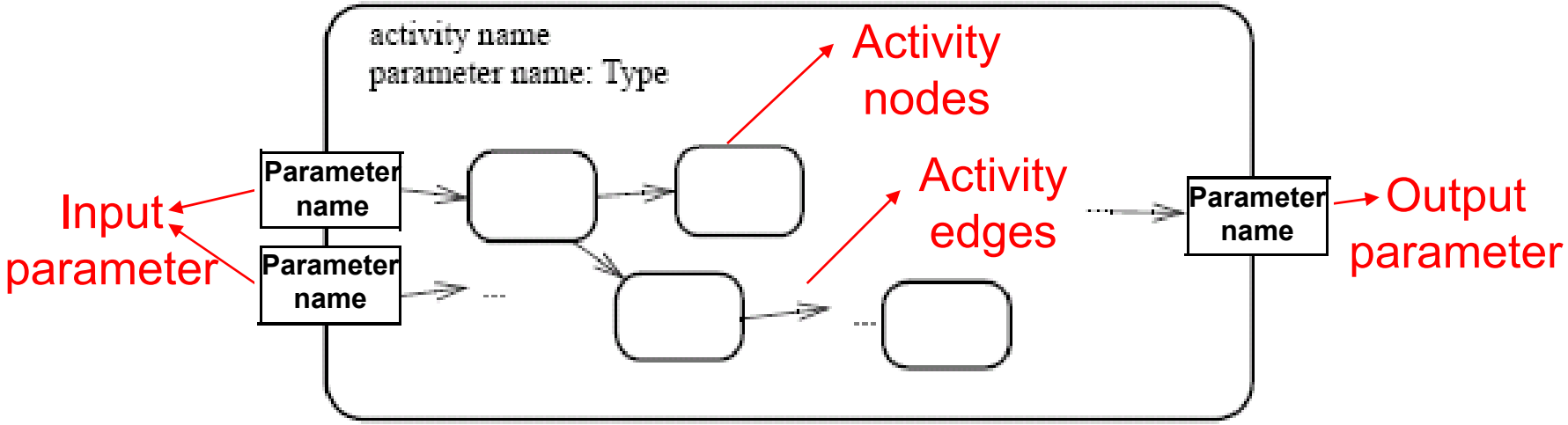


- An action can start only if:
  - all non-stream inputs have arrived (or at least one stream input if there are only stream inputs)
- The action can finish only if:
  - all inputs have arrived (streaming inputs included)
  - all non-stream and non-exception outputs (or an exception outputs) have been provided

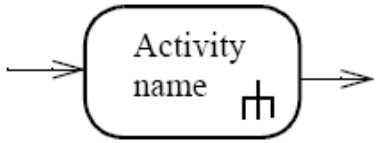
- ***Prevent deadlock***: an input pin of an action cannot accept tokens until all the input pins of the action can accept them

Activities

- An activity is the specification of parameterized behaviour as the coordinated sequencing of subordinate units whose individual elements are actions
- Uses parameters to receive and provide data to the invoker



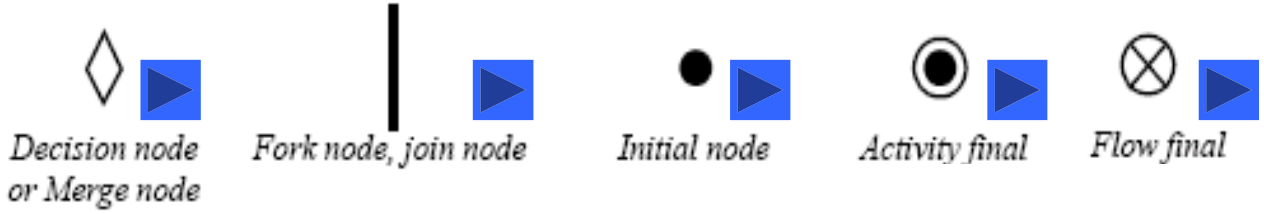
- An action can invoke an activity to describe its action more finely



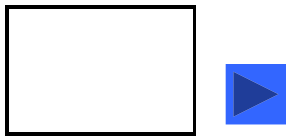
Activity nodes

- Three type of activity nodes:
  - **Action nodes**: executable activity nodes; the execution of an action represents some transformations or processes in the modeled system (already seen)

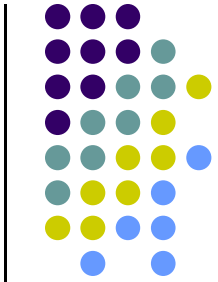
  - **Control nodes**: coordinate flows in an activity diagram between other nodes



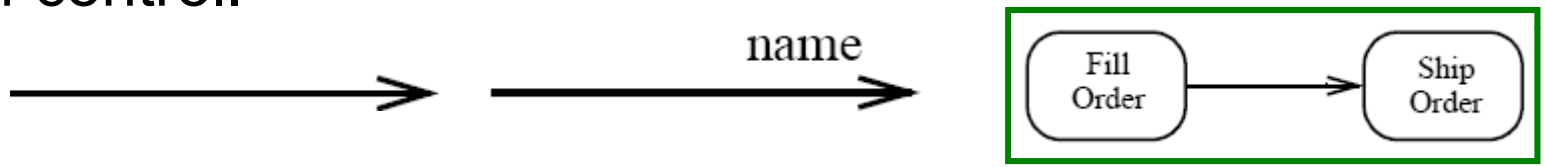
  - **Object nodes**: indicate an instance of a particular object, may be available at a particular point in the activity (i.e Pins are object nodes)



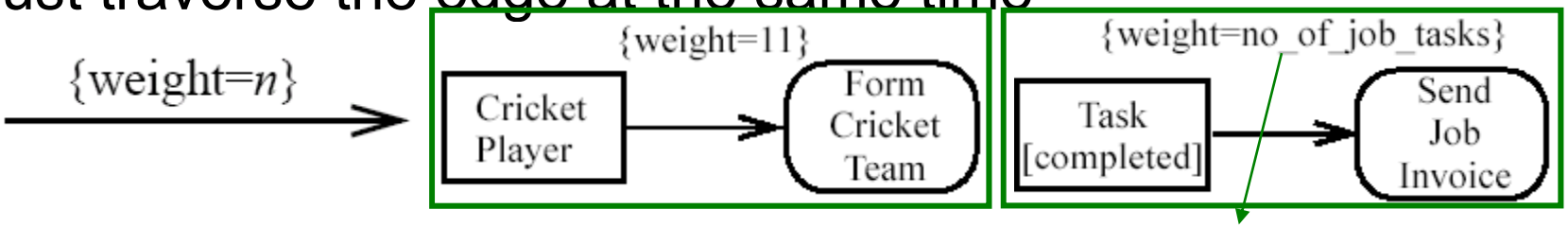
Activity edges (1)



- Are directed connections
- They have a source and a target, along which tokens may flow
- A token may represent real data, an object, or the focus of control.

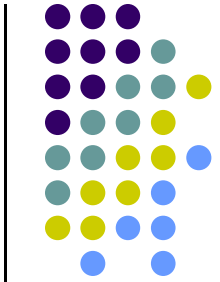


- Any number of tokens can pass along the edge, in groups at one time, or individually at different times
- ***Weight***: determines the minimum number of tokens that must traverse the edge at the same time



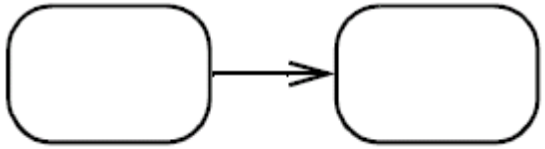
In this example we use a non-constant weight: an invoice for a particular job can only be sent when all of its tasks have been completed

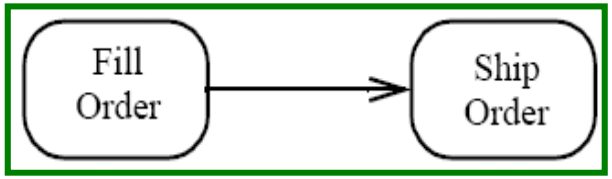
Activity edges (2)



- Two kinds of edges:
  - **Control flow edge** - is an edge which starts an activity node after the completion of the previous one by passing a control token

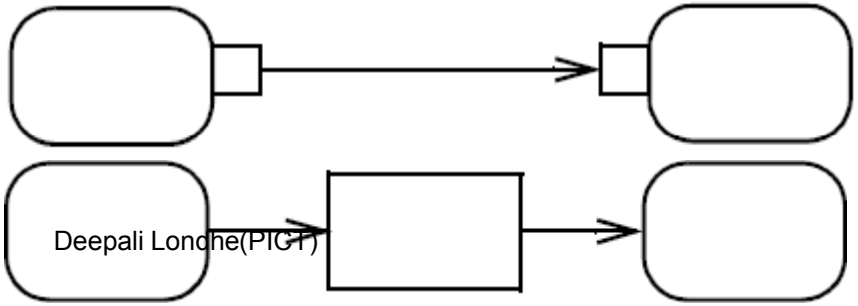
Notation without activity nodes



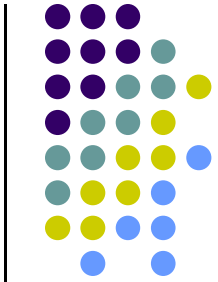


  - **Object flow edge** - models the flow of values to or from object nodes by passing object or data tokens

Notation without activity nodes

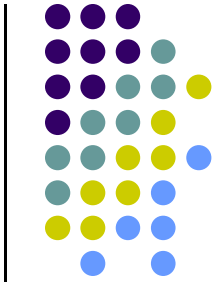


Options



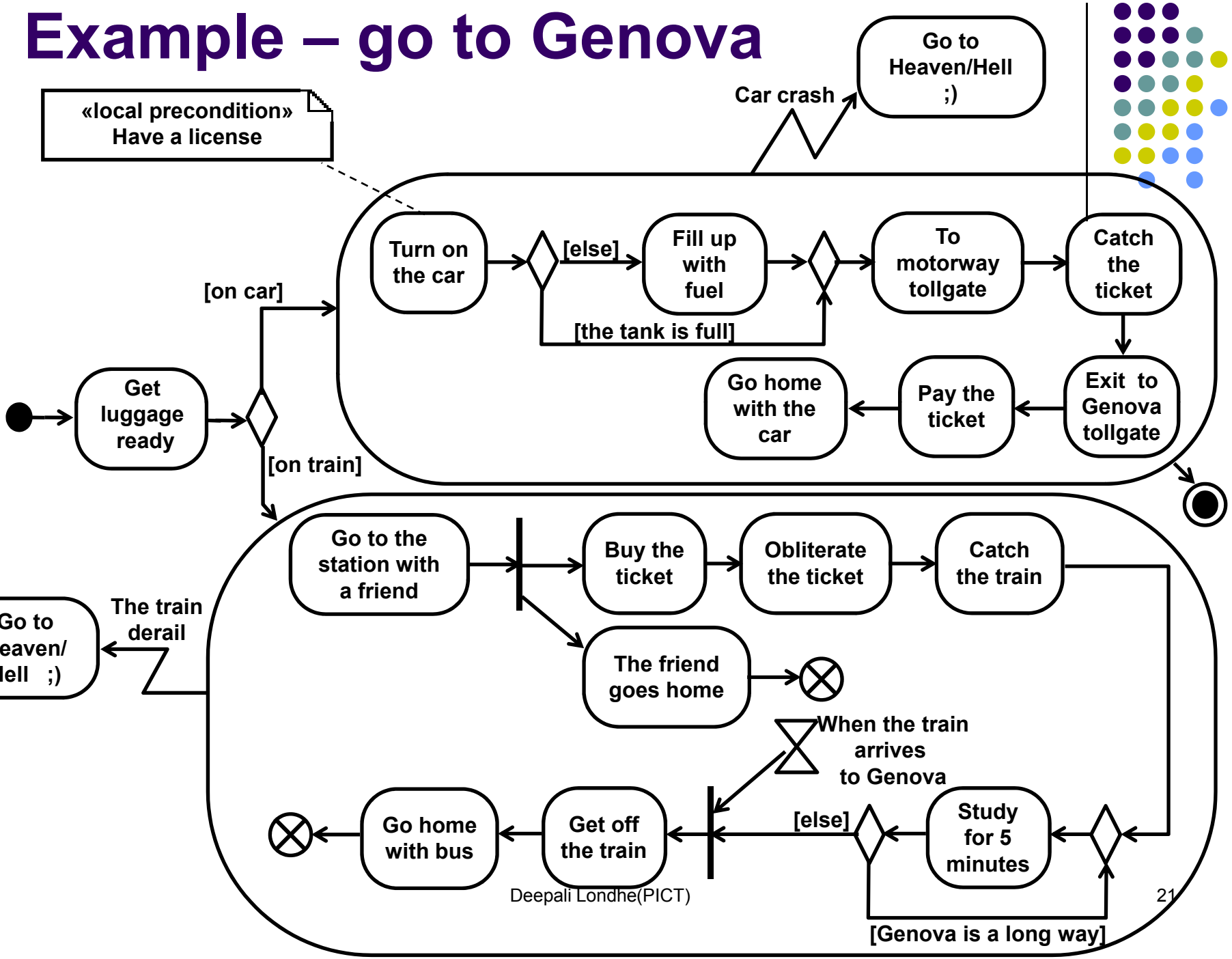
- Object nodes
  - multiplicities and upperBound
  - effect and ordering
- Activity nodes
  - presentation options
  - transformation
- Selection
- Token competion

New notations

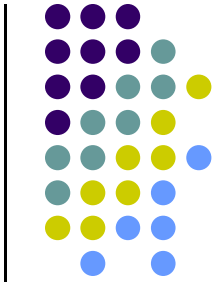


- ActivityPartition
- Pre & post condition
- SendSignalAction
- Time triggers and Time events
- AcceptEventAction
- InterruptibleActivityRegion
- Exception
- ExpansionRegion

Example – go to Genova



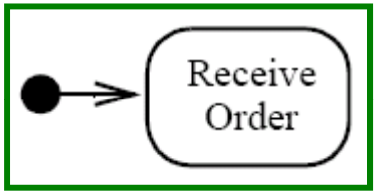
Bibliography



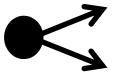
- Object Management Group, “UML 2.0 Superstructure Specification,” http://www.omg.org/cgi-bin/doc?ptc/03-08-02, August 2003
- Conrad Bock (U.S. NIST), “UML 2 Activity and Action Models,” in *Journal of Object Technology*, vol. 2, no. 4, July-August 2003, pp. 43-53, http://www.jot.fm/issues/issue_2003_07/column3
- Conrad Bock (U.S. NIST), “UML 2 Activity and Action Models, Part 2: Actions,” in *Journal of Object Technology*, vol. 2, no. 5, September-October 2003, pp. 41-56, http://www.jot.fm/issues/issue_2003_09/column4
- Conrad Bock (U.S. NIST), “UML 2 Activity and Action Models, Part 3: Control Nodes,” in *Journal of Object Technology*, vol. 2, no. 6, November-December 2003, pp. 7-23, http://www.jot.fm/issues/issue_2003_11/column1
- Conrad Bock (U.S. NIST), “UML 2 Activity and Action Models, Part 4: Object Nodes,” in *Journal of Object Technology*, vol. 3, no. 1, January-February 2004, pp. 27-41, http://www.jot.fm/issues/issue_2003_11/column1
- Gruppo Yahoo su UML, http://groups.yahoo.com/group/uml-forum/

Control nodes – initial nodes

- In an *activity* the flow starts in initial nodes, that return the control immediately along their outgoing edges



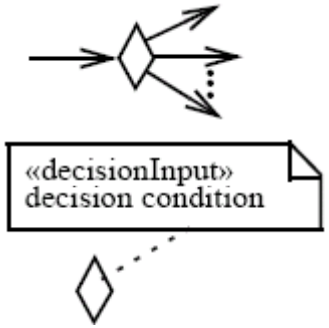
- If there are more than one initial node, a control token is placed in each initial node when the activity is started, initiating multiple flows
- If an initial node has more than one outgoing edge, only one of these edges will receive control, because initial nodes cannot duplicate tokens

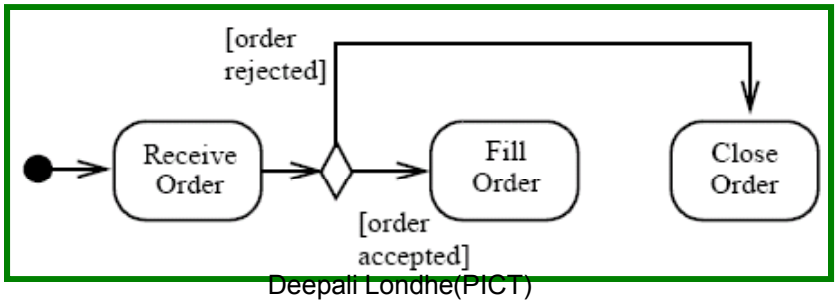


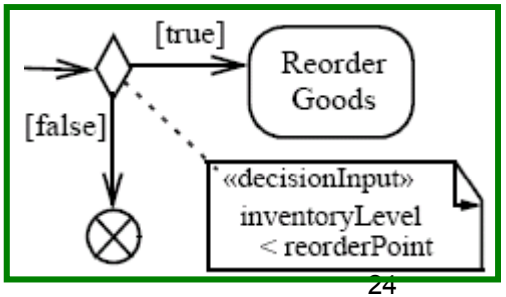
A or B ?

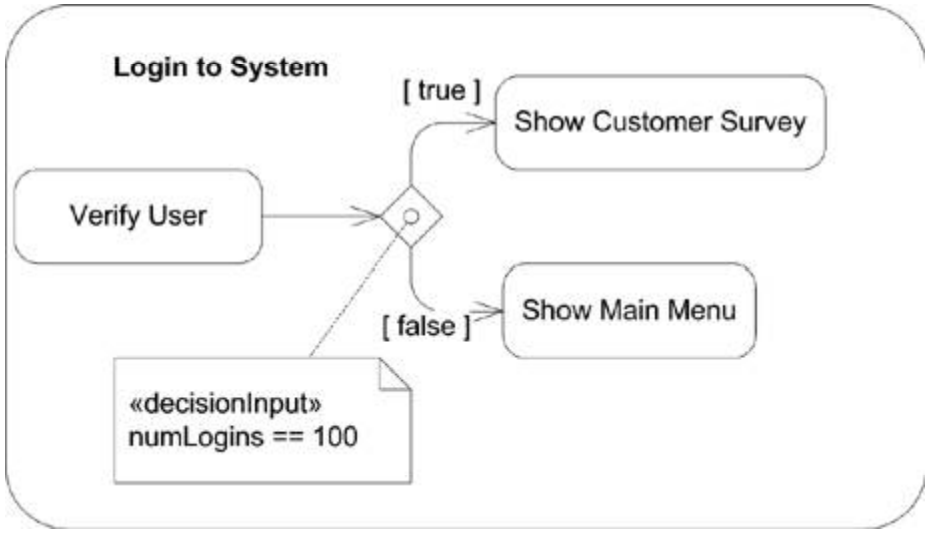
Control nodes – decision nodes

- Route the flow to one of the outgoing edges (tokens are not duplicated)
- Guards are specified on the outgoing edges or with the stereotype «decisionInput»
- There is also the predefined guard **[else]**, chosen only if the token is not accepted by all the other edges
- If all the guards fail, the token remains at the source object node until one of the guards accept it





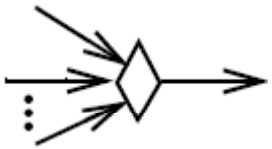


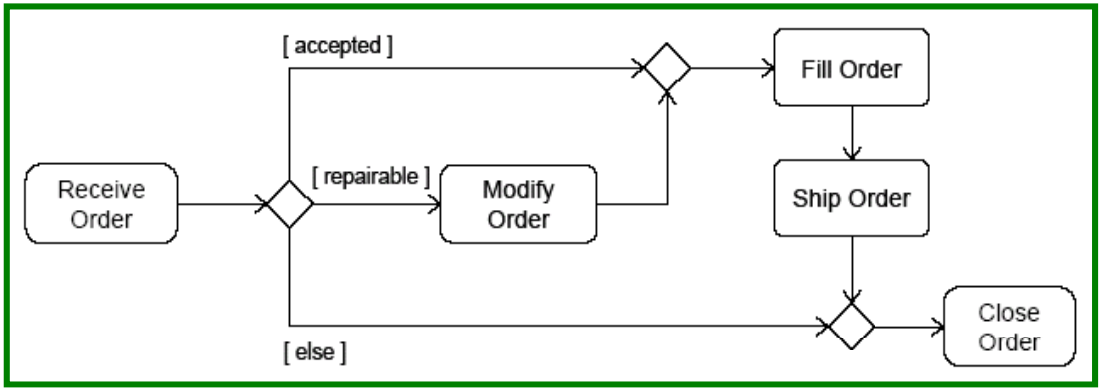


Decision node with an input behavior

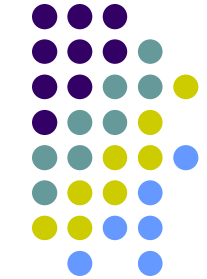
Control nodes – merge nodes

- Bring together multiple alternate flows
- All controls and data arriving at a merge node are immediately passed to the outgoing edge
- There is no synchronization of flows or joining of tokens

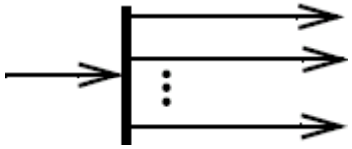


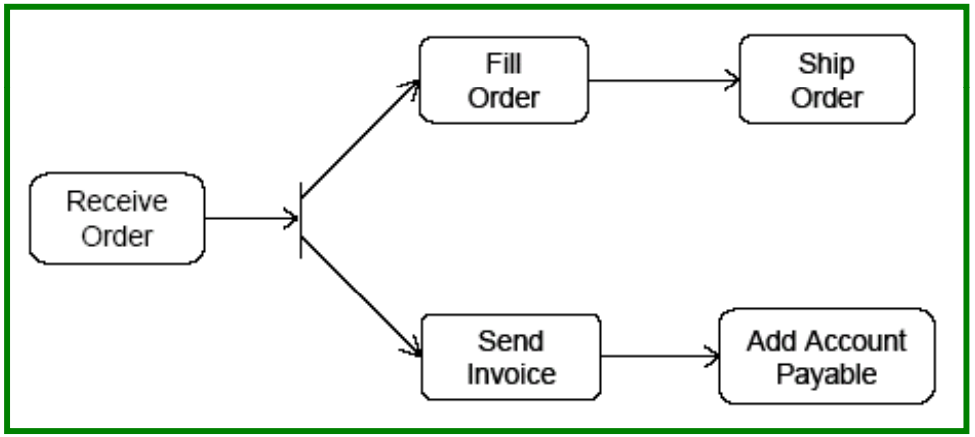


Control nodes – fork nodes



- Fork nodes split flows into multiple concurrent flows (tokens are duplicated)

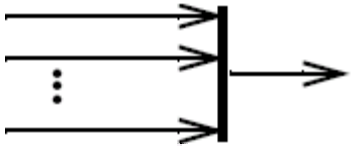


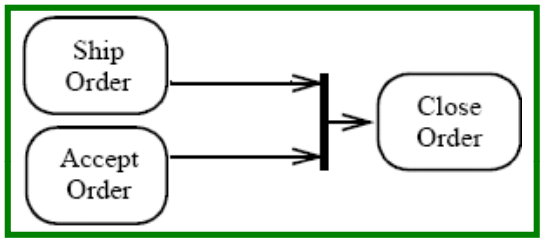


- State machine forks in UML 1.5 required synchronization between parallel flows through the state machine RTC step (it means that the first state in each branch is executed, then the second one, etc.)
- UML 2.0 activity forks model unrestricted parallelism

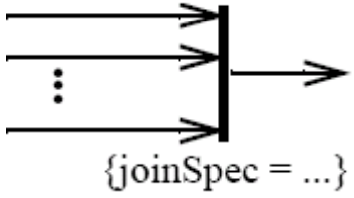
Control nodes – join nodes

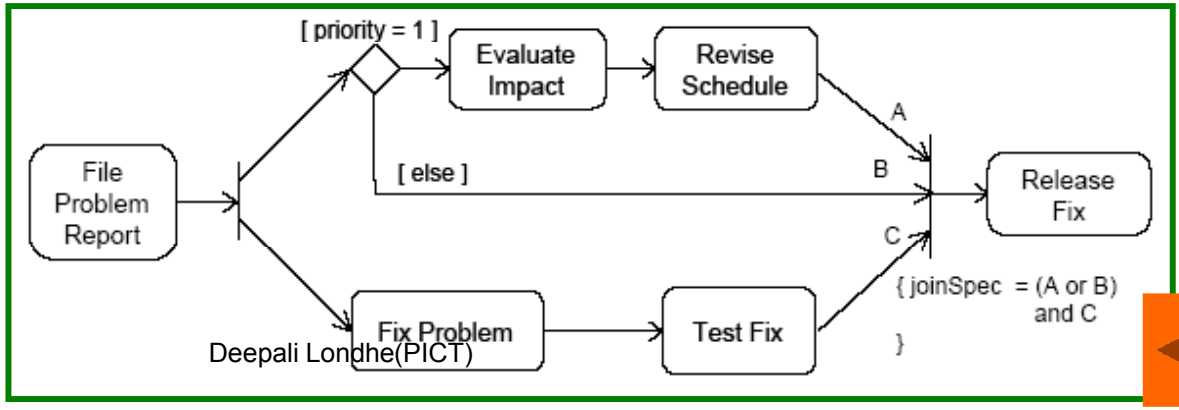
- Join nodes synchronize multiple flows





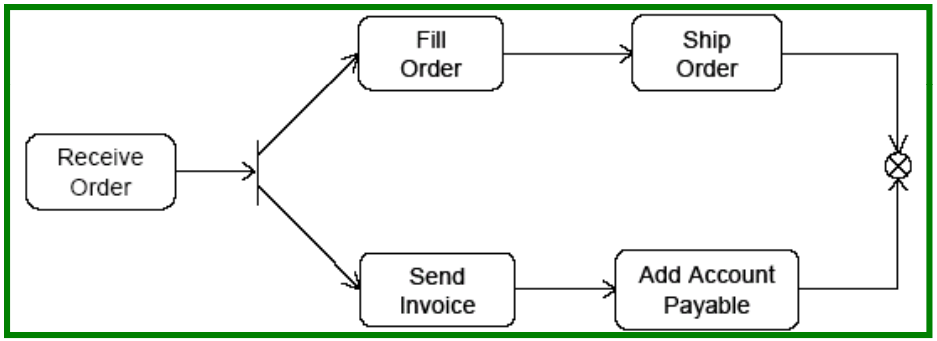
- Generally, controls or data must be available on every incoming edge in order to be passed to the outgoing edge, but user can specify different conditions under which a join accepts incoming controls and data using a *join specification*



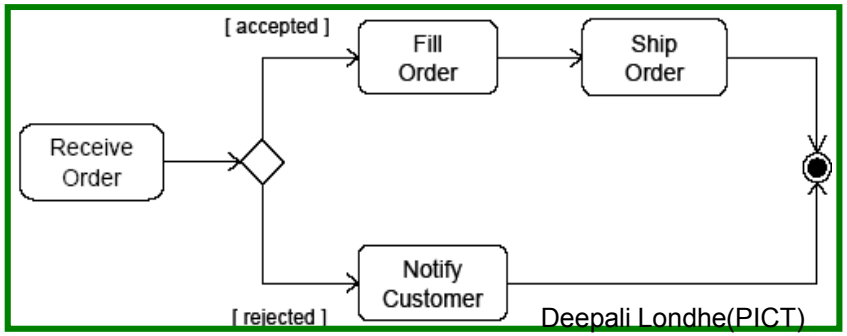


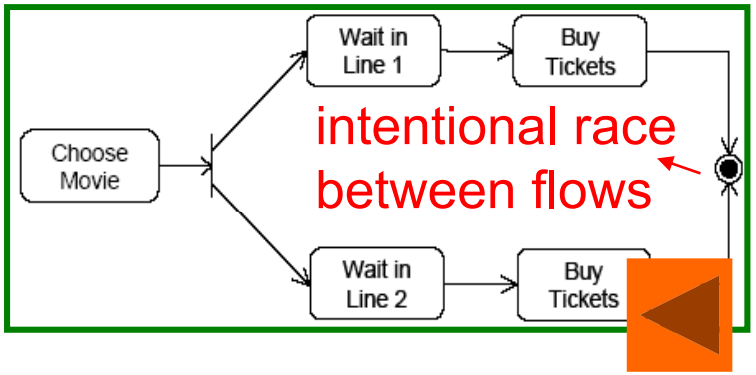
Control nodes – final nodes

- **Flow final**:
  - destroys the tokens that arrive into it
  - the activity is terminated when all tokens in the graph are destroyed



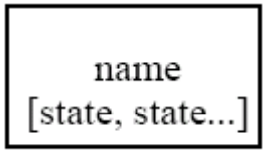
- **Final node**:
  - the activity is terminated when the first token arrives



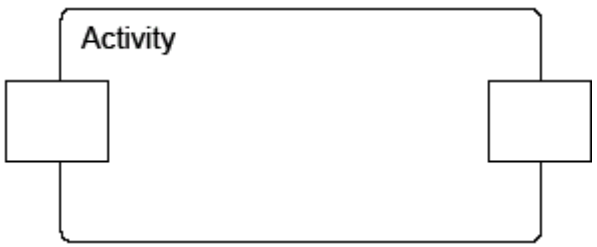


Object nodes

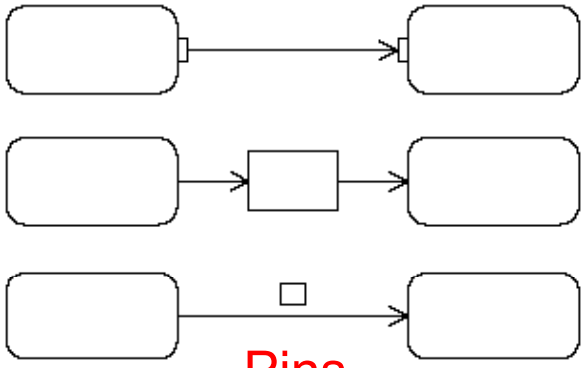
- Hold data temporarily while they wait to move through the graph
- Specify the type of values they can hold (if no type is specified, they can hold values of any type)
- Can also specify the state of the held objects



- There are four kinds of object nodes:

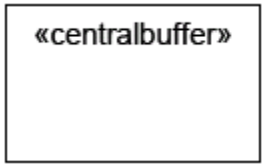


Activity Parameter Nodes

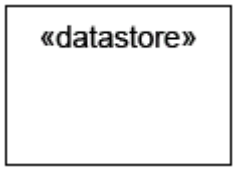


Pins

(three differents notations)

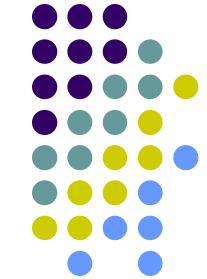


Central Buffer Nodes

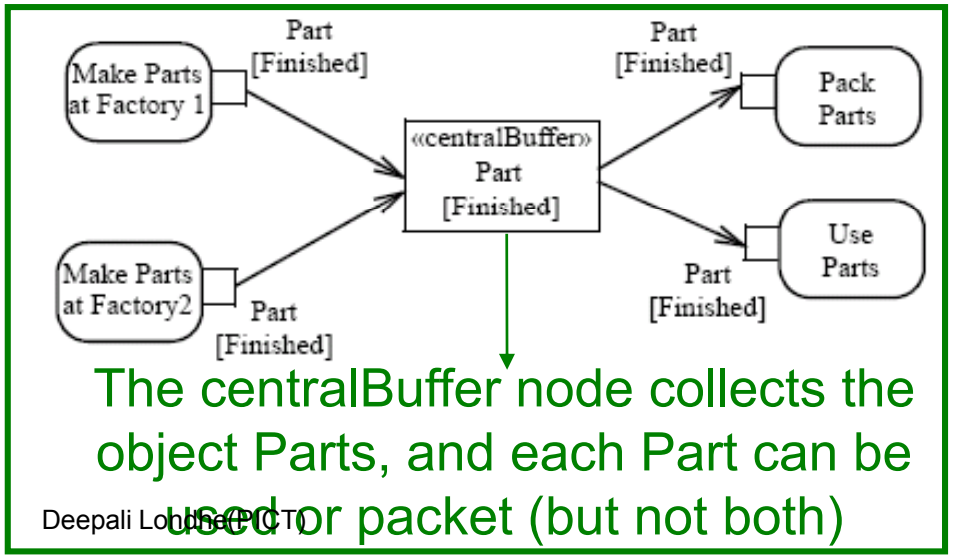


Data Store Nodes

Object nodes – centralBuffer

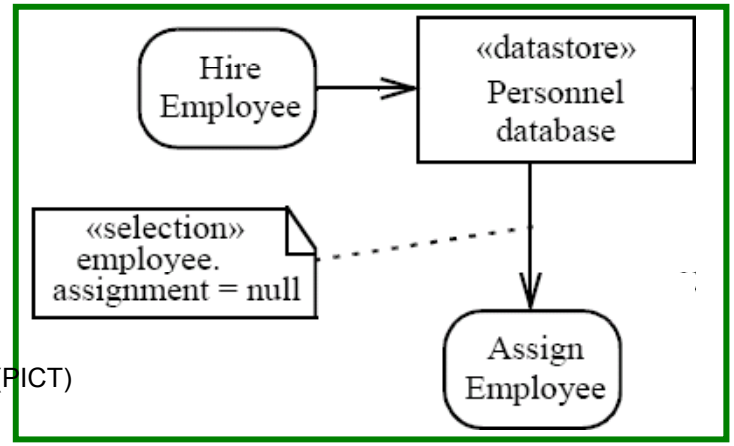


- A central buffer node is an object node that manages flows from multiple sources and destinations (as opposed to pins and parameters)
- Acts as a buffer for multiple input flows and output flows
- Is not tied to an action like pins, or to an activity like activity parameter nodes



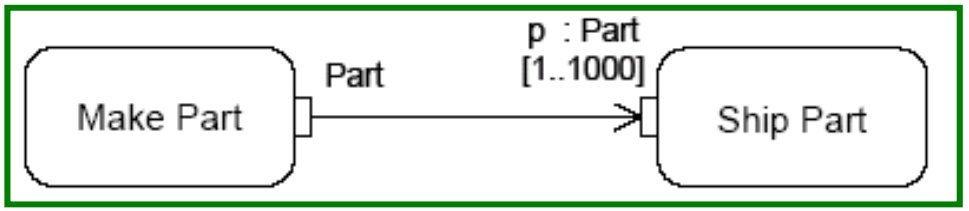
Object nodes – datastore

- Is a specific central buffer node which stores objects persistently
- Keeps all tokens that enter into it
- Tokens chosen to move downstream are copied so that tokens never leave the data store
- If arrives a token containing an object already present in the data store, this replaces the old one
- Tokens in a data store node cannot be removed (they are removed when the activity is terminated)

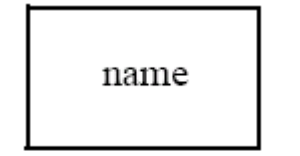


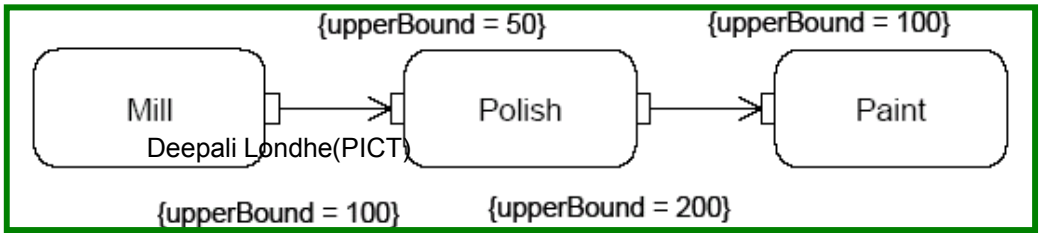
Object nodes - multiplicities and upperBound

- ***Multiplicities***: specify the minimum (≥0) and maximum number of values each pin accepts or provides at each invocation of the action:
  - when is available the minimum number of values, the action can start
  - if there is more values than the maximum, the action takes only the first maximum value

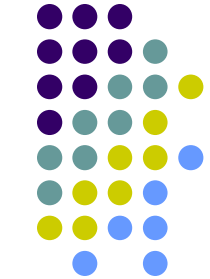


- ***UpperBound***: shows the maximum number of values that an object node can hold: at runtime, when the upper bound has been reached, the flow is stopped (buffering)

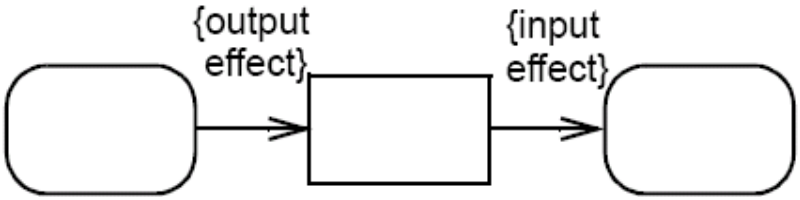


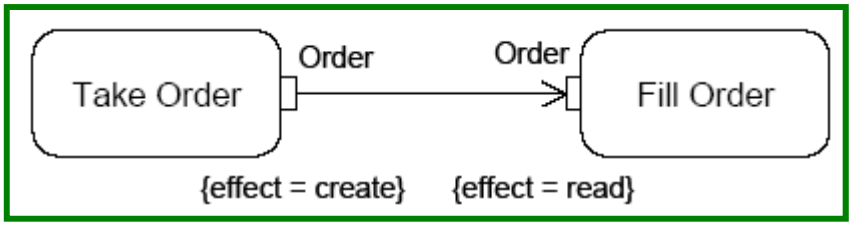


Object nodes – effect and ordering

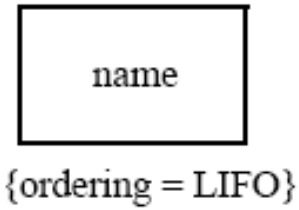


- ***Effect***: pins can be notated with the effect that their actions have on objects that move through the pin
- The effects can be: ‘create’ (only on output pins), ‘read’, ‘update’ or ‘delete’ (only on input pins)





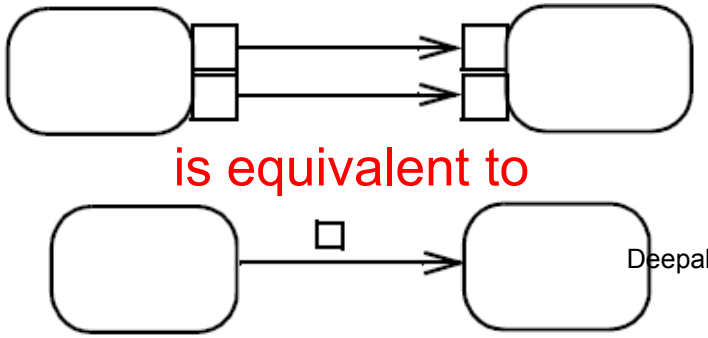
- ***Ordering***: specifies the order in which the tokens of an object node are offered to the outgoing edges (FIFO, LIFO or modeler-defined ordering)



Activity edges – presentation options

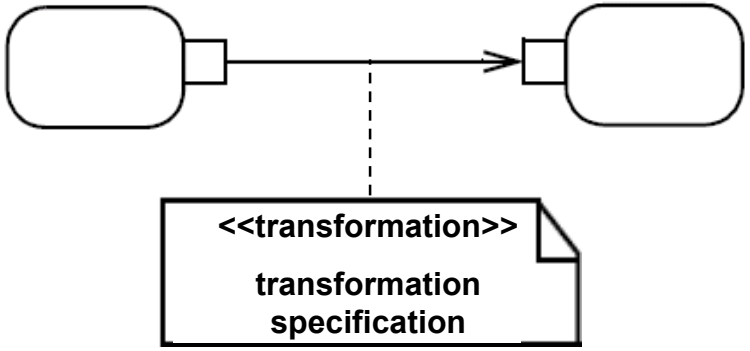
- An edge can also be notated using a connector
- Every connector with a given label must be paired with exactly one other with the same label on the same activity diagram

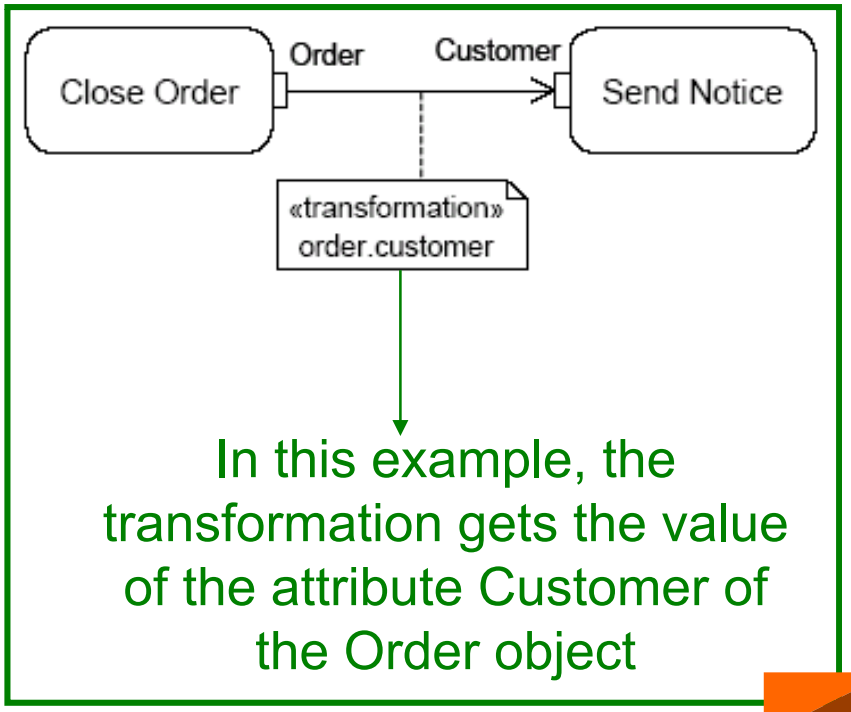
- To reduce clutter in complex diagrams, object nodes may be elided



Activity edges - transformation

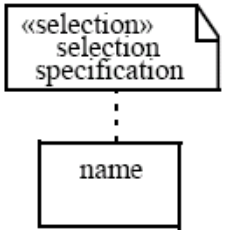
- It is possible to apply a transformation of tokens as they move across an object flow edge (each order is passed to the transformation behaviour and replaced with the result)

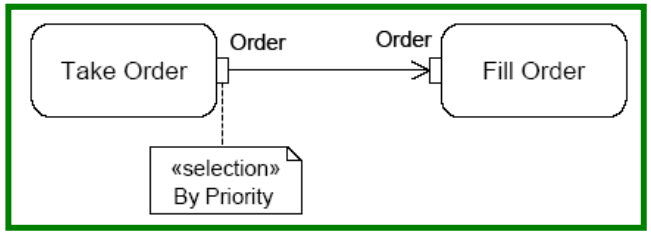




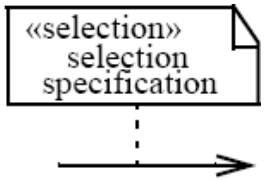
Selection

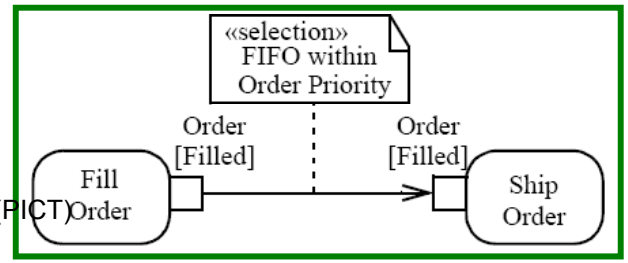
- Specifies the order (FIFO, LIFO or modeler-defined ordering) in which tokens in the node are offered to the outgoing edges
- Can be applied to:
  - ***Object node*** - specifies the object node ordering, choosing what token offers to the outgoing edge whenever it asks a token





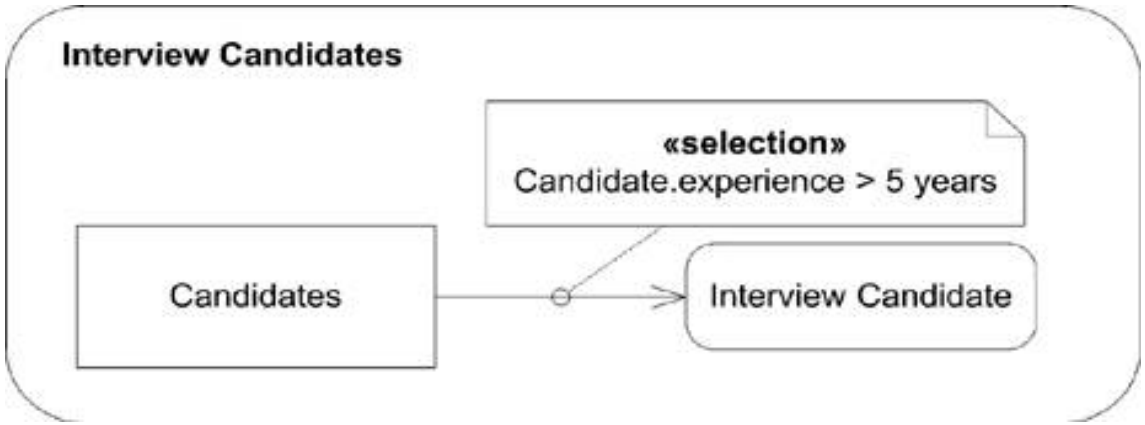
  - ***Edge*** - chooses the order on which tokens are offered from the source object node to the edge (overrides any selection present on the object node, that is object node ordering)





Selection Contd..

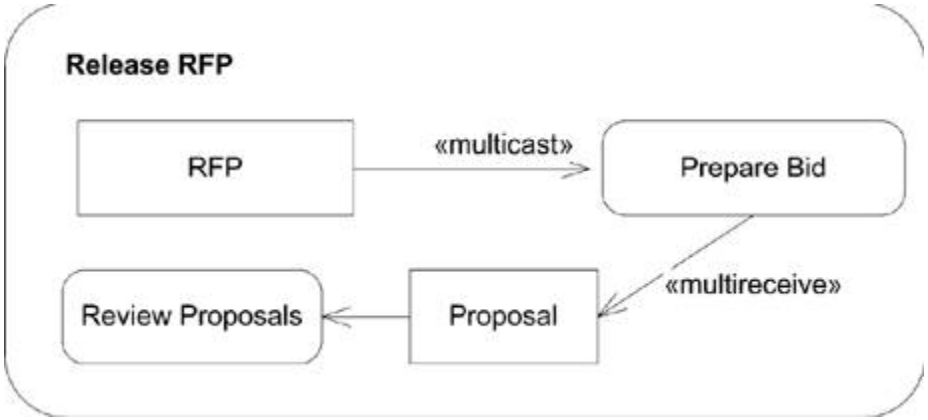
UML offers a data-only activity edge, called *object flows. Object flows add support for multicasting* data, token selection, and transformation of tokens.



Activity diagram where objects are selected based on a selection behavior

«multicast»

- Object flows allow you to specify sending data to multiple instances of a receiver using *multicasting.*
- *For example, if you are modeling a bidding process, you can show a Request for* Proposal (RFP) being sent to multiple vendors and their responses being received by your activity.

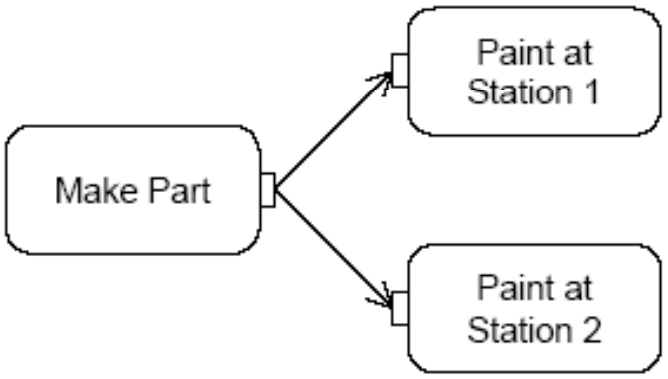


Multicast object flows

you show data being received from multiple senders by labeling an object flow with the keyword «multireceive»

Token competition

- A parameter node or pin may have multiple edges coming out of it, whereupon there will be competition for its tokens, because object nodes cannot duplicate tokens while forks can
- Then there is *indeterminacy* in the movement of data in the graph



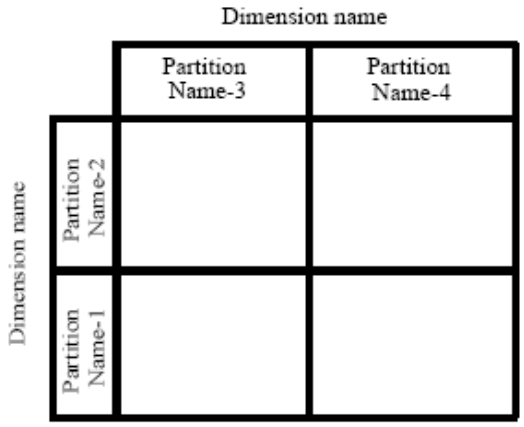
If the input pin of *Paint at Station 1* is full, the token remains at the output of *Make Part* until the traversal can be completed to one of the input pins

ActivityPartition (1)

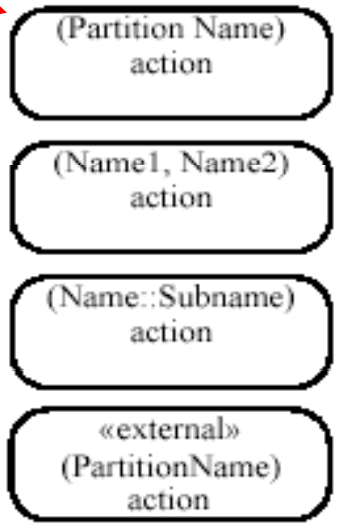
- Partitions divide the nodes and edges for identifying actions that have some characteristics in common
- They often correspond to organizational units in a business model
- Partitions can be hierarchical and multidimensional
- Additional notation is provided: placing the partition name in parenthesis above the activity name

a) Partition using a swimlane notation

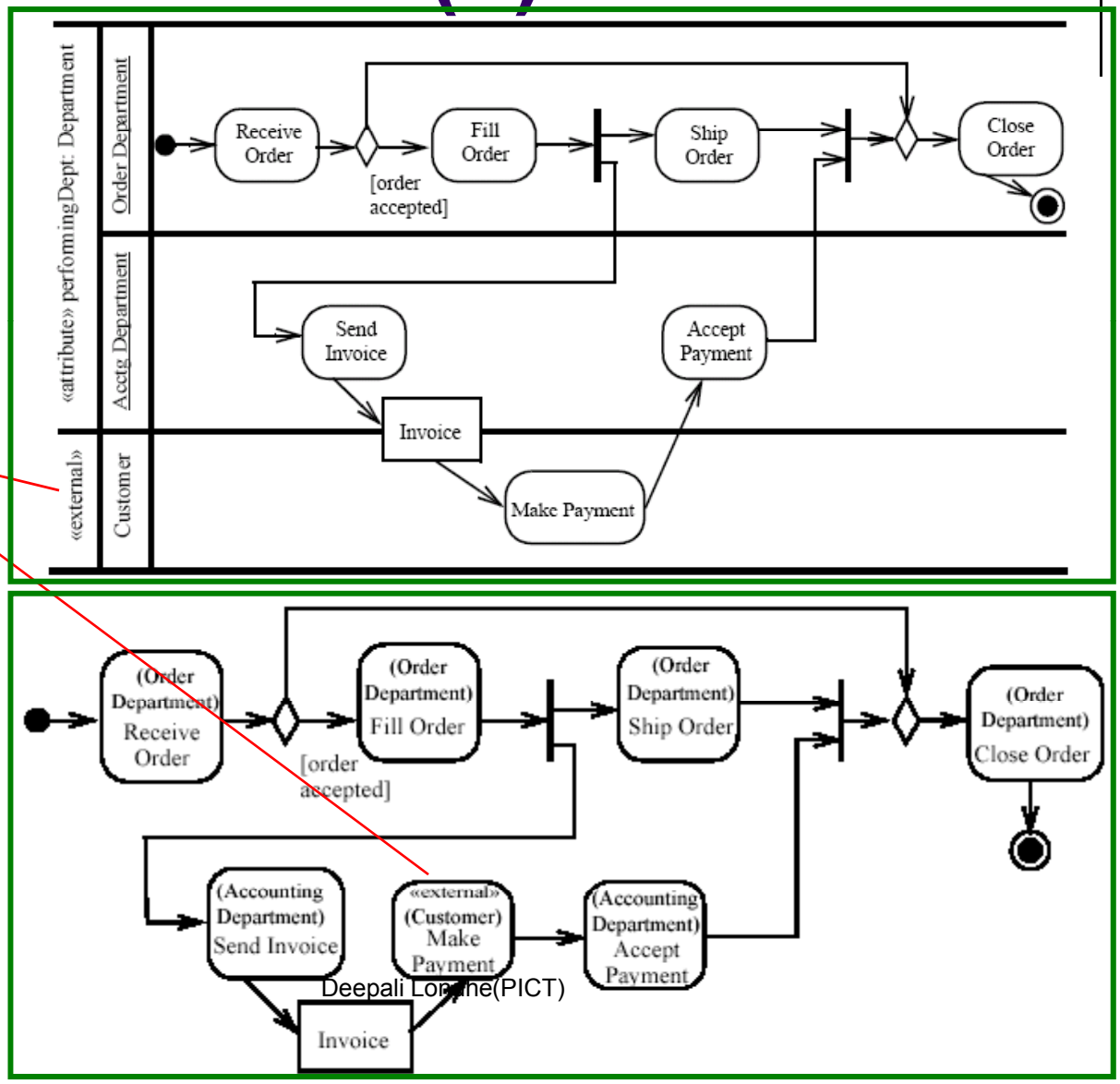
b) Partition using a hierarchical swimlane notation



c) Partition using a multidimensional hierarchical swimlane notation



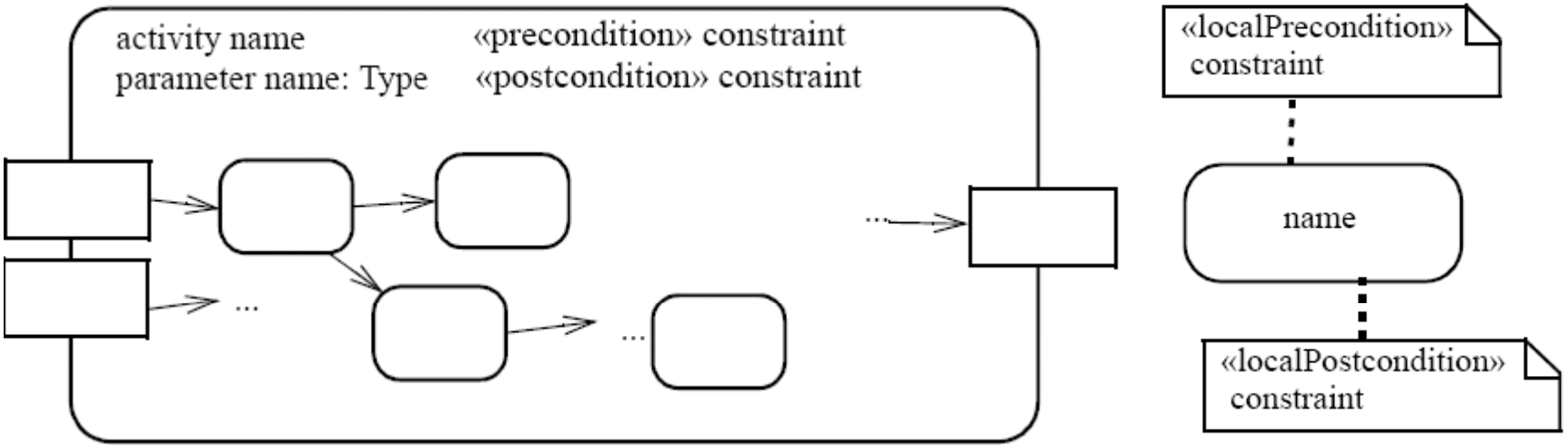
ActivityPartition (2)



Partition notated to occur outside the primary concern of the model

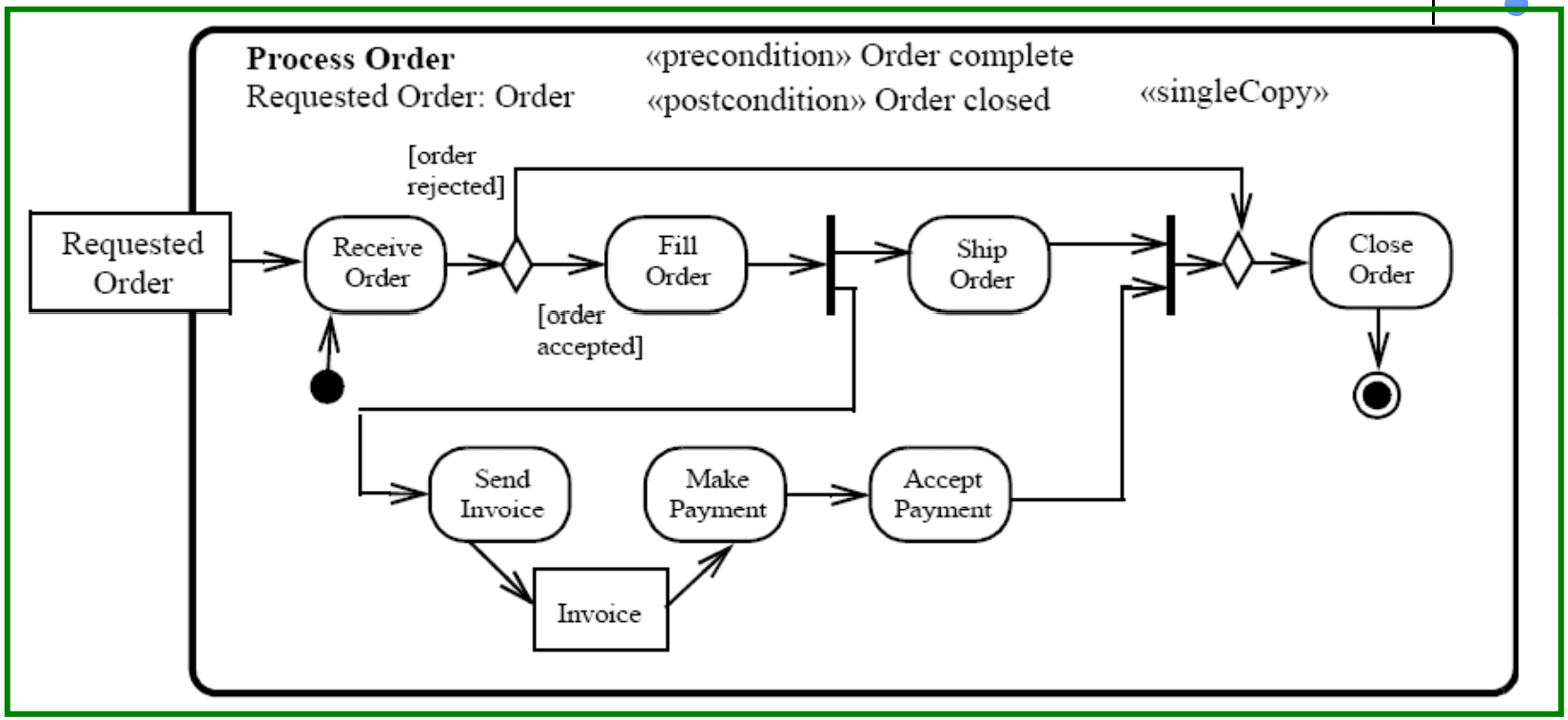
Pre & post condition (1)

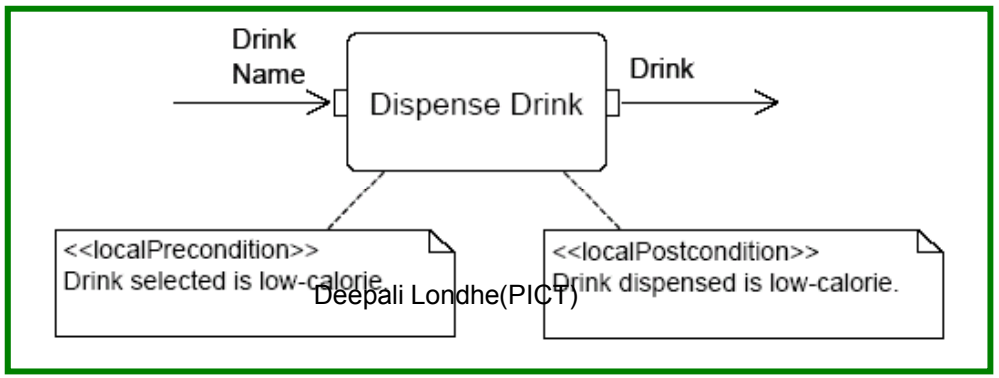
- Can be referred to an activity or to an action (local condition)



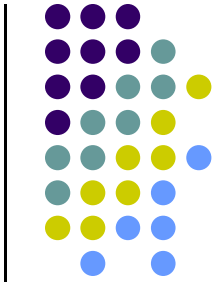
- UML intentionally does not specify when or whether pre/post conditions are tested (design time, runtime, etc.)
- UML also does not define what the runtime effect of a failed pre/post condition should be (error that stops execution, warning, no action)

Pre & post condition (2)



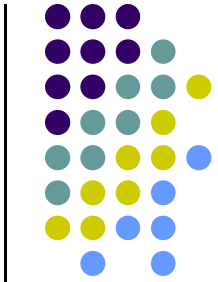


SendSignalAction

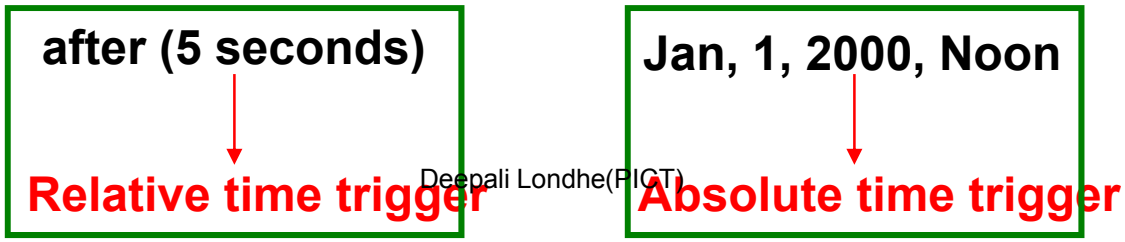


- Creates a signal instance from its inputs, and transmits it to the target object (local or remote)
- A signal is an asynchronous stimulus that triggers a reaction in the receiver in an asynchronous way and without a reply
- Any reply message is ignored

Time triggers and Time events

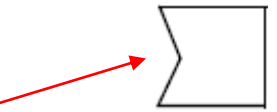


- A *Time trigger* is a trigger that specifies when a time event will be generated
- *Time events* occur at the instant when a specified point in time has transpired
- This time may be relative or absolute
  - ***Relative time trigger***: is specified with the keyword 'after' followed by an expression that evaluates to a time value
  - ***Absolute time trigger***: is specified as an expression that evaluates to a time value

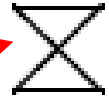


AcceptEventAction

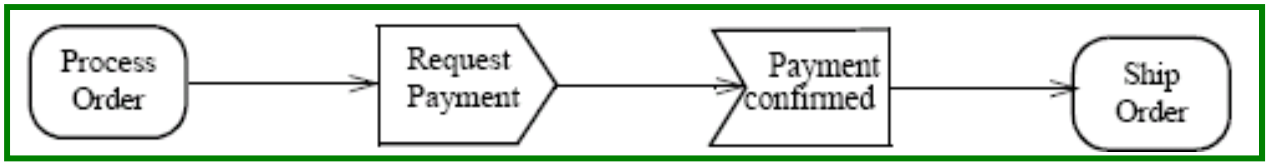
- Waits for the occurrence of an event meeting specified conditions
- Two kinds of AcceptEventAction:
  - **Accept event action** – accepts signal events generated by a SendSignalAction
  - **Wait time action** – accepts time events

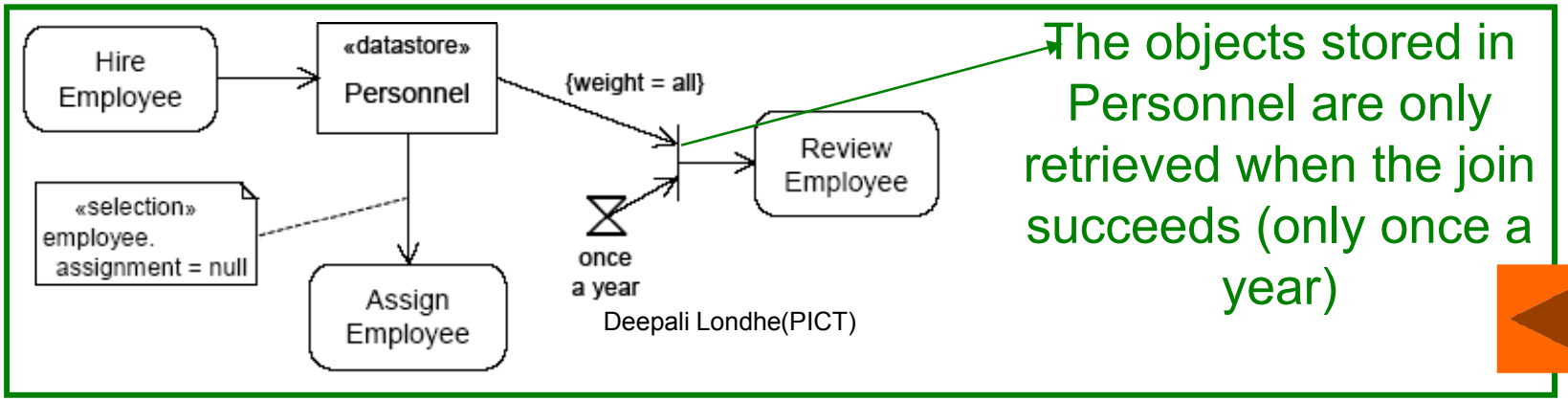


Accept event actior



Wait time action

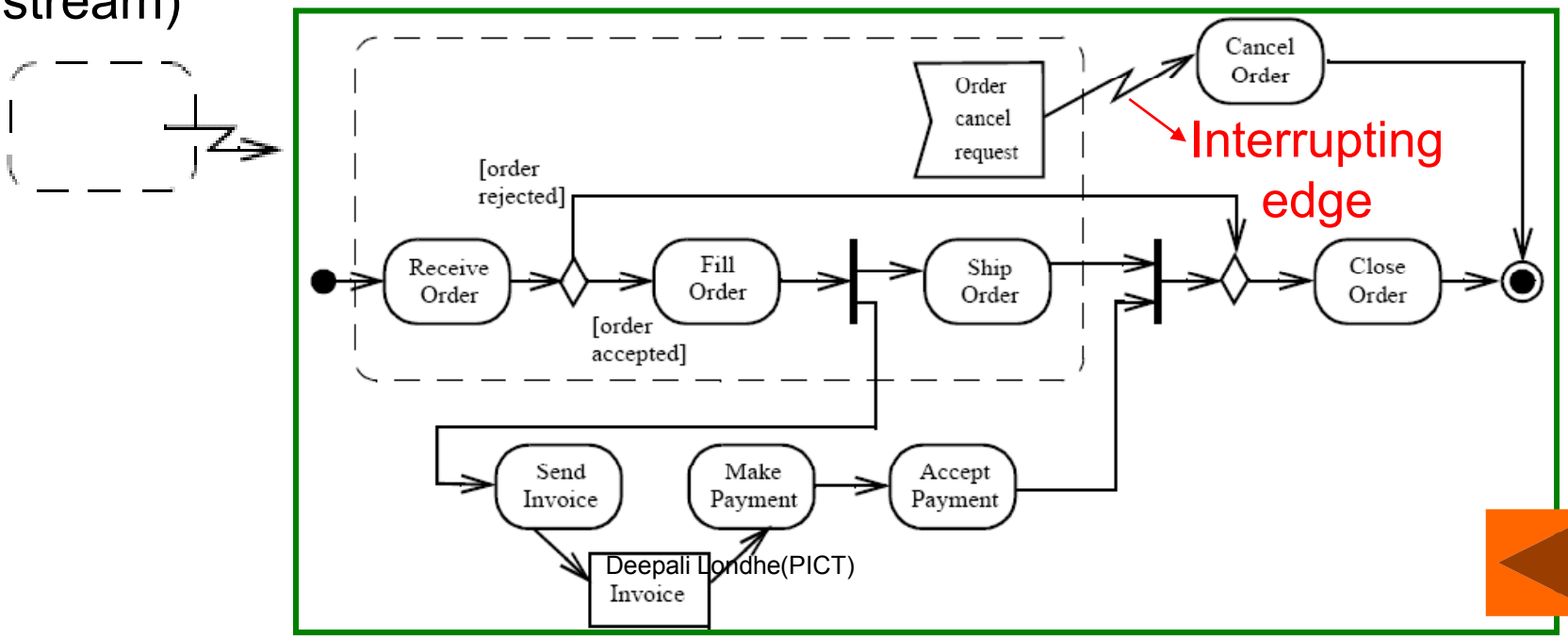




The objects stored in Personnel are only retrieved when the join succeeds (only once a year)

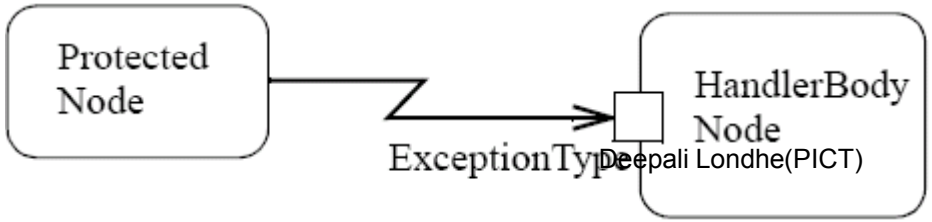
InterruptibleActivityRegion

- Is an activity group (sets of nodes and edges) that supports termination of tokens flowing into it
- When a token leaves an interruptible region via interrupting edges, all tokens and behaviours in the region are terminated
- Token transfer is still atomic: a token transition is never partial; it is either complete or it does not happen at all (also for internal stream)



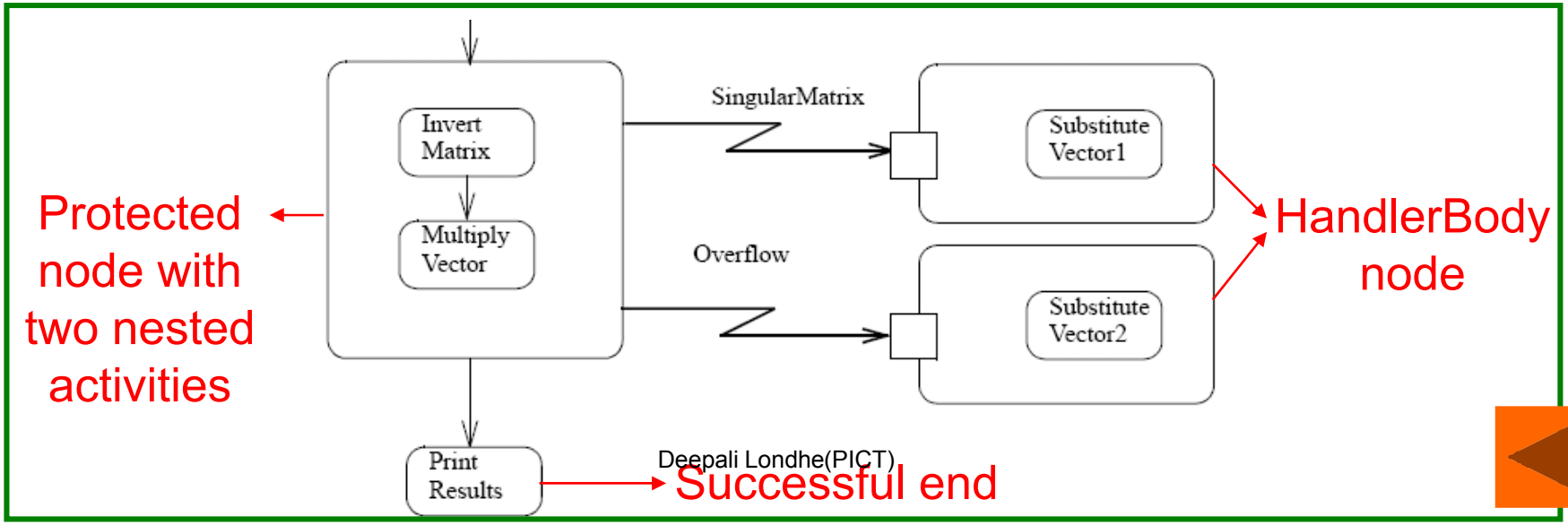
Exceptions (1)

- Exception handler - specifies the code to be executed when the specified exception occurs during the execution of the protected node
- When an exception occurs the set of execution handlers on the action is examined to look for a handler that matches (*catches*) the exception
- If the exception is not caught, it is propagated to the enclosing protected node, if one exists
- If the exception propagates to the topmost level of the system and is not caught, the behaviour of the system is unspecified; profiles may specify what happens in such cases



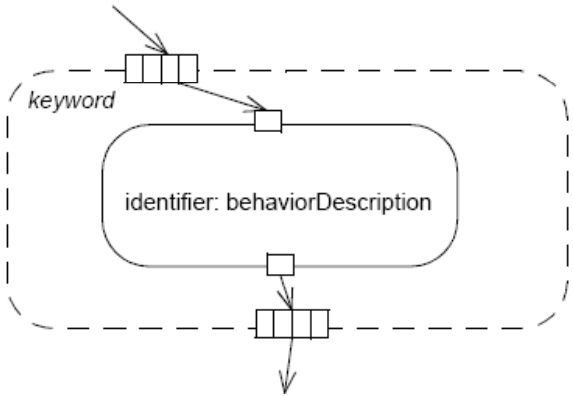
Exceptions (2)

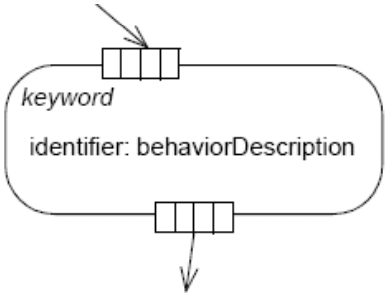
- When an exception is caught, is executed the exception body instead of the protected node, and then the token is passed to all the edges that go out from that protected node
- The exception body has no explicit input or output edges
- Exception body can resolve the problems that have caused the exception or can abort the program
- We can put any activities nested in a protected node (in UML 2.0, nesting activities is allowed)

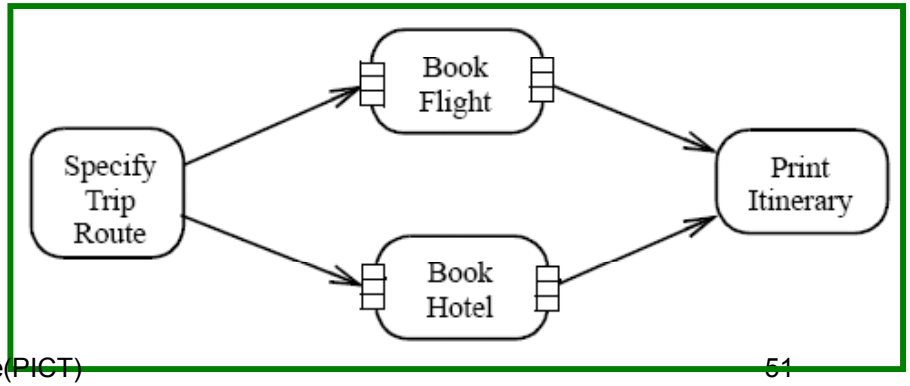


ExpansionRegion (1)

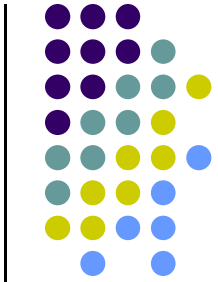
- Nested region of an activity in which each input is a collection of values
- The expansion region is executed once for each element (or position) in the input collection
- On each execution of the region, an output value from the region is inserted into an output collection at the same position as the input elements







ExpansionRegion (2)



- There are three ways of interaction between the executions:
  - **Parallel** (concurrent): all the interactions are independent
  - **Iterative**: the interactions occur in the order of the elements (the executions of the region must happen in sequence, with one finishing before another can begin)
  - **Stream** (streaming): there is a single execution of the region, where the values in the input collection are extracted and placed into the execution of the expansion region as a stream (in order if the collection is ordered)

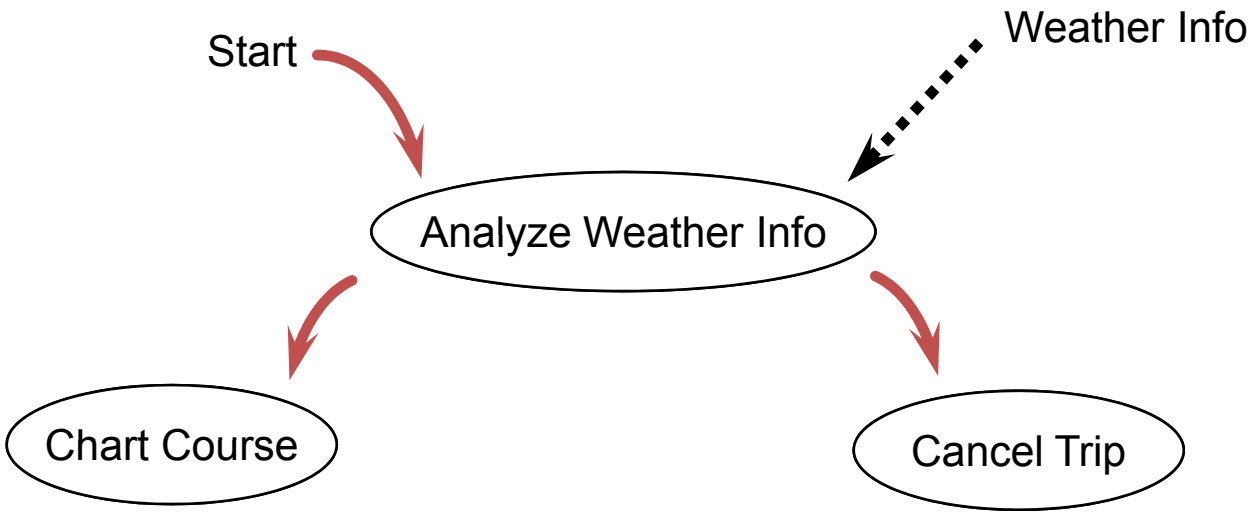
Activity Diagram

Activity Diagram Applications

- Intended for applications that need control flow or object/data flow models …
- ... rather than event-driven models like state machines.
- For example: business process modeling and workflow.
- The difference in the three models is how step in a process is initiated, especially with respect to how the step gets its inputs.

Control Flow

- Each step is taken when the previous one finishes …
- …regardless of whether inputs are available, accurate, or complete (“pull”).
- Emphasis is on order in which steps are taken.

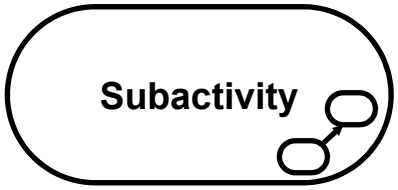


Not UML Notation!

Kinds of Steps in Activity Diagrams

Action (State)

Subactivity (State)

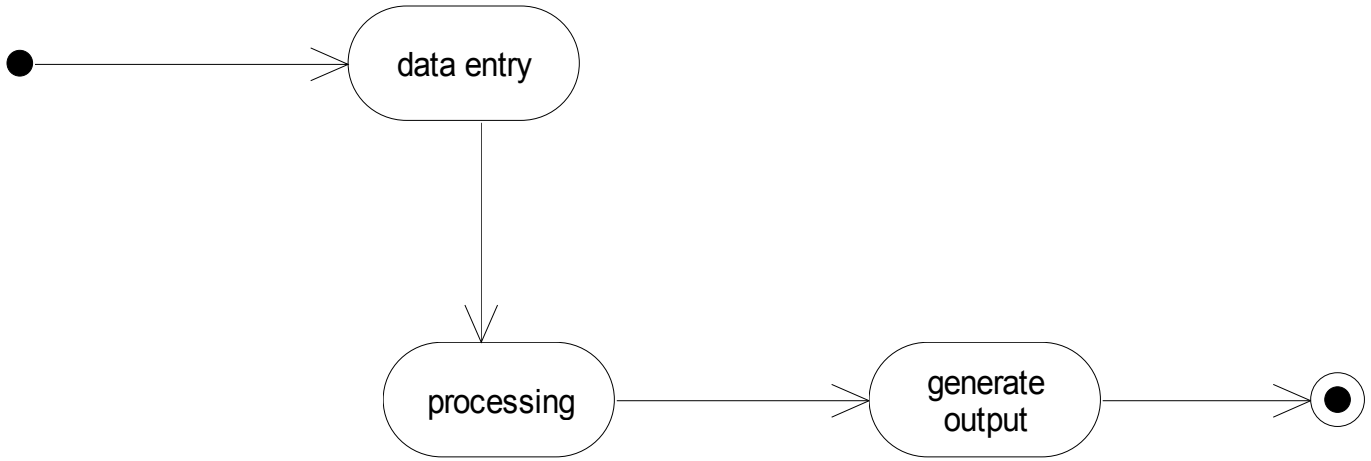


- An activity is an ongoing nonatomic execution within a state machine.
- Activities ultimately result in some action, which is made up of executable atomic computations that results in a change in state of the system or the return of a value.

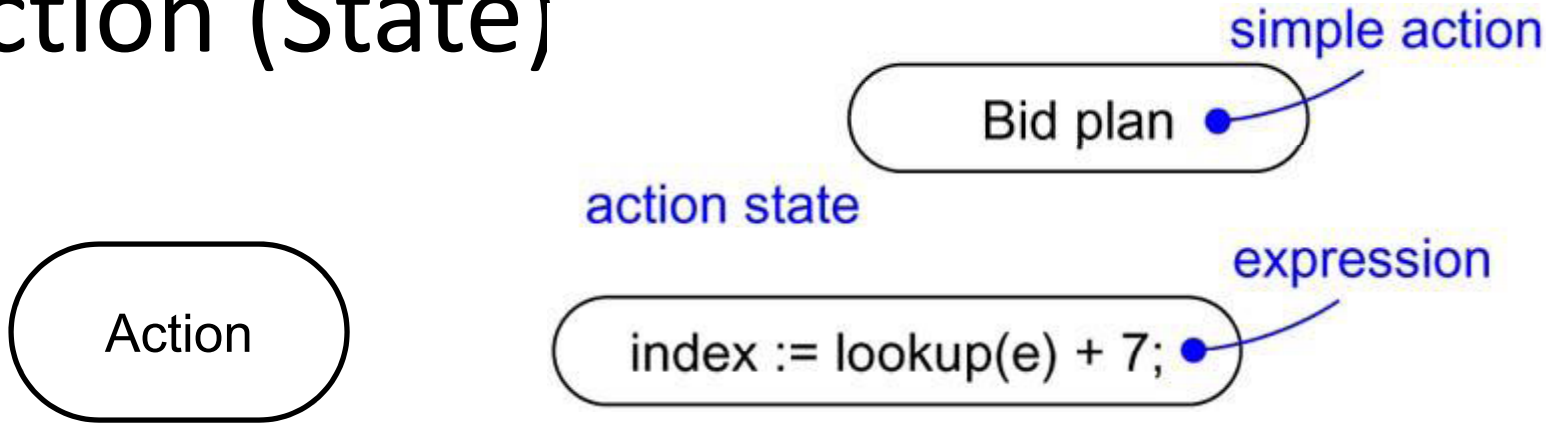
Action state

- State in which some work is being done
  - activity, task
- State terminates when the work is finished
  - difference with state diagrams
- After termination the action state can lead to another action state
  - “state transition”
- Special symbols for being and end of a procedure or process

Basic notation for activity diagram



Action (State)

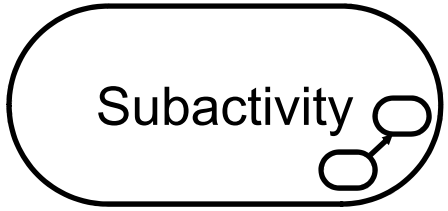


- An action is used for anything that does not directly start another activity graph, like invoking an operation on an object, or running a user-specified action.
- While modeling flow of control, You might evaluate some expression that sets the value of an attribute or that returns some value. Alternately, you might call an operation on an object, send a signal to an object, or even create or destroy an object.
- These executable, atomic computations are called action states because they are states of the system, each representing the execution of an action.

Action (State)

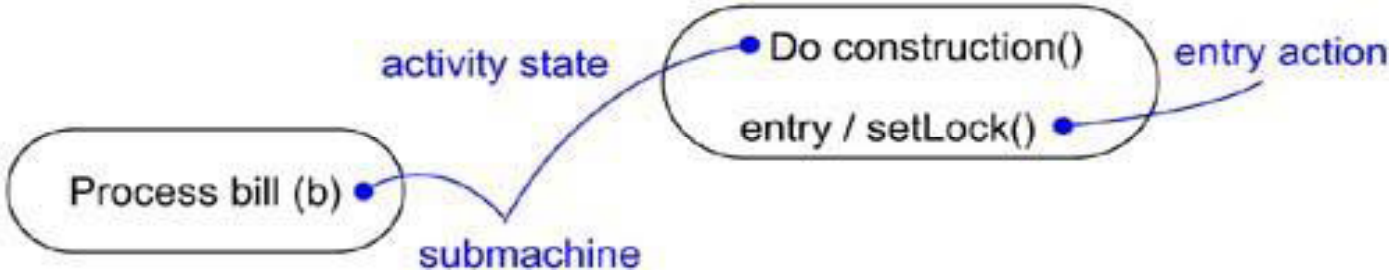
- Action states can't be decomposed. Furthermore, action states are atomic, meaning that events may occur, but the work of the action state is not interrupted. Finally, the work of an action state is generally considered to take insignificant execution time.
- However, an action can invoke an operation that has another activity graph as a method (possible polymorphism).

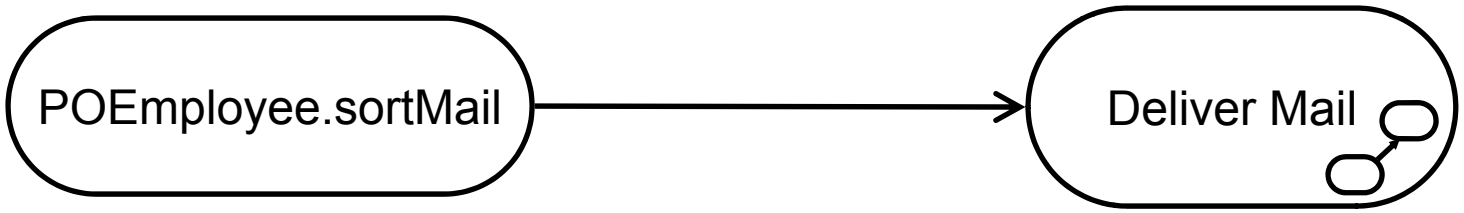
Subactivity (State)

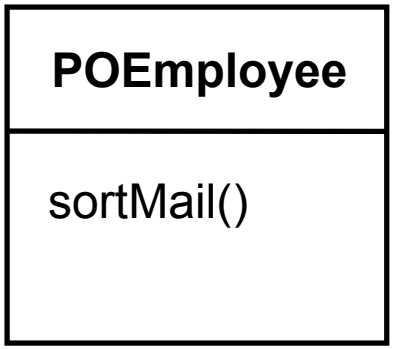


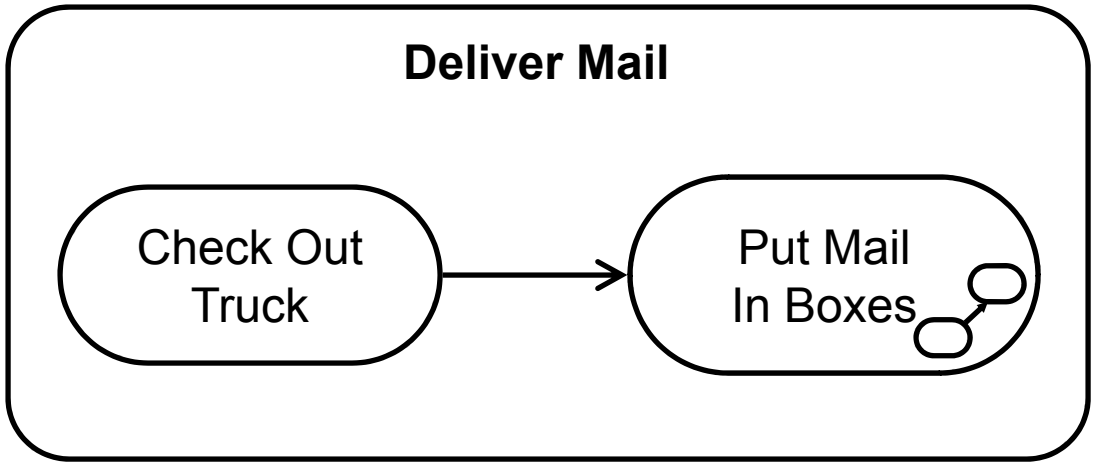
- Activity states can be further decomposed, their activity being represented by other activity diagrams.
- Activity states are not atomic, meaning that they may be interrupted and, in general, are considered to take some duration to complete.
- An action state is a special case of an activity state. An action state is an activity state that cannot be further decomposed.
- Activity state is a composite, whose flow of control is made up of other activity states and action states.
- The invoked activity graph can be used by many subactivity states.
- an activity state may have additional parts, such as entry and exit actions

Example

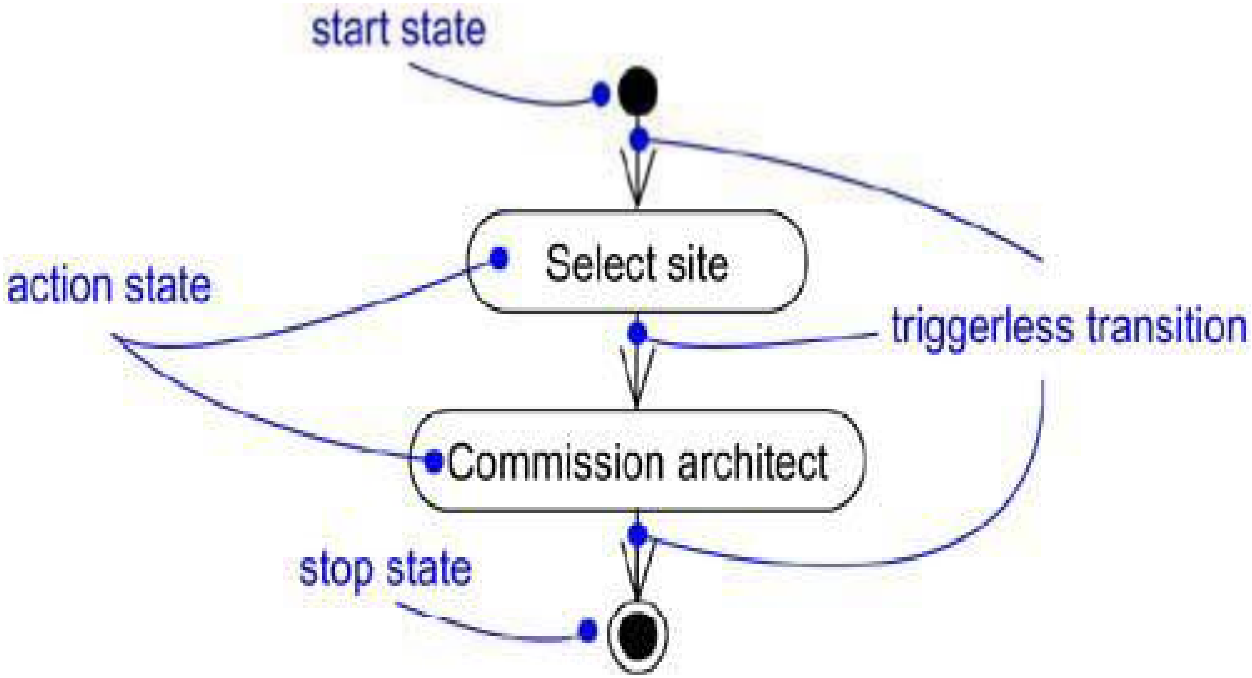

activity state
Do construction()
entry action
entry / setLock()
Process bill (b)
submachine


POEmployee.sortMail
Deliver Mail


POEmployee
sortMail()


Deliver Mail
Check Out Truck
Put Mail In Boxes

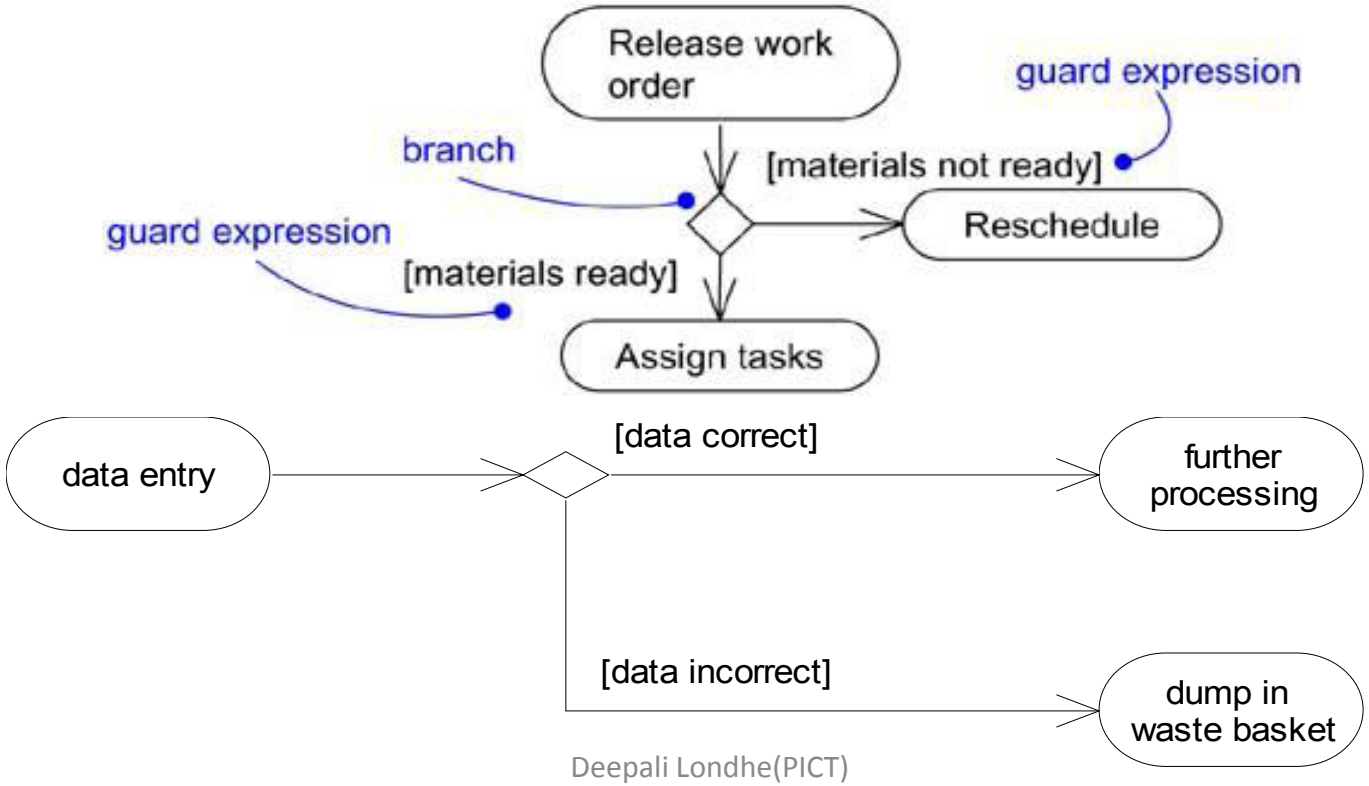
Transition



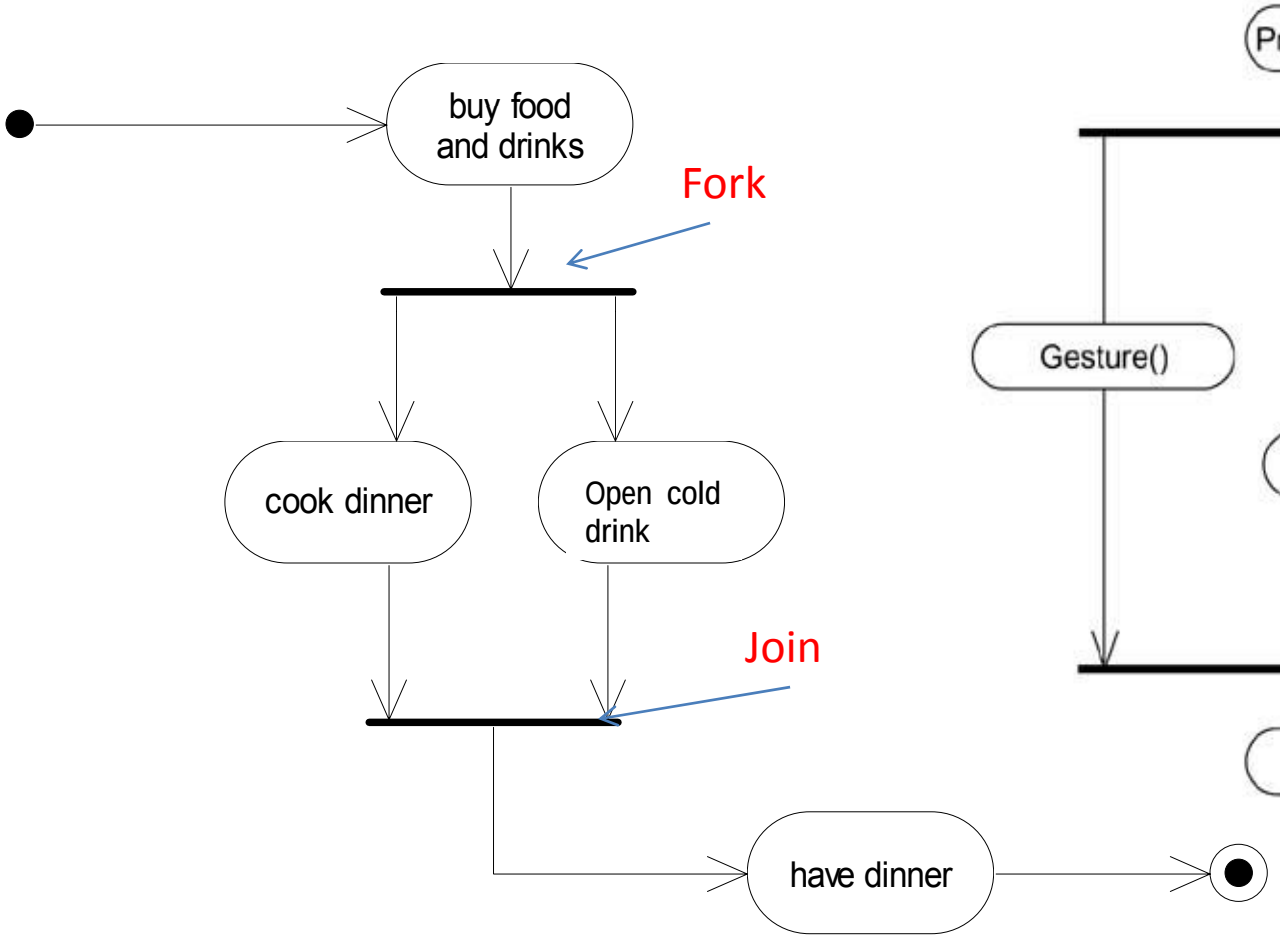
Semantically, these are called triggerless, or completion, transitions because control passes immediately once the work of the source state is done. Once the action of a given source state completes, you execute that state's exit action (if any). Next, and without delay, control follows the transition and passes on to the next action or activity state

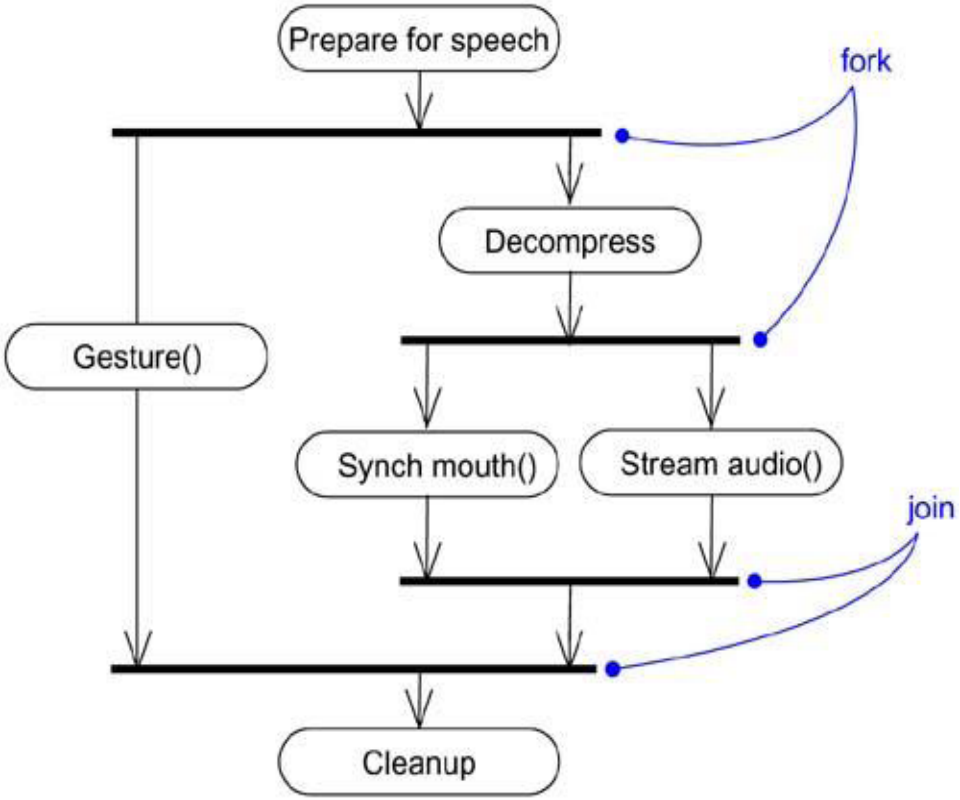
Decision- branching

- Sate transition is deterministic
- If transition depends on outcome of the work, then introduce a decision



Introducing concurrency





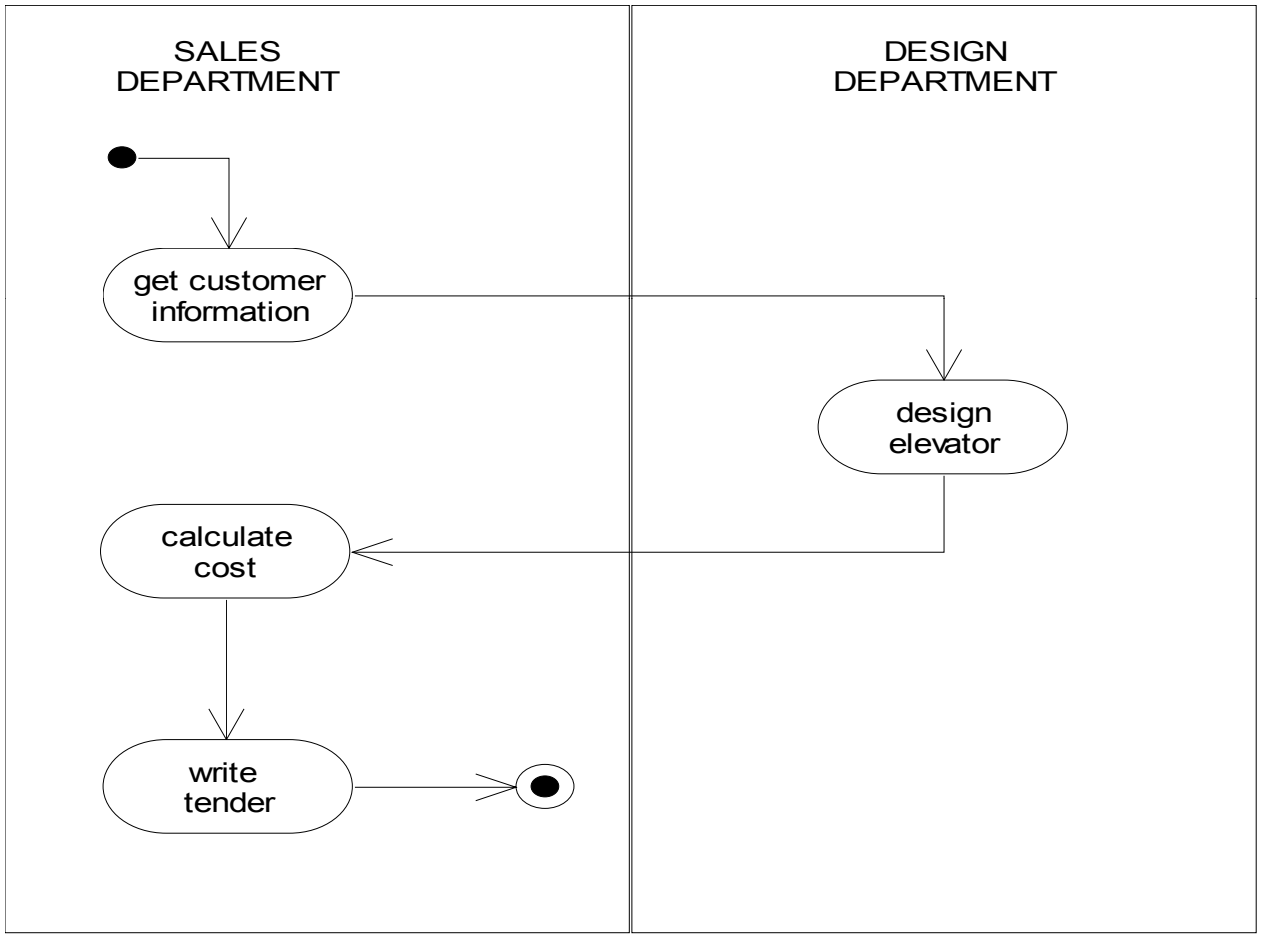
Concurrency

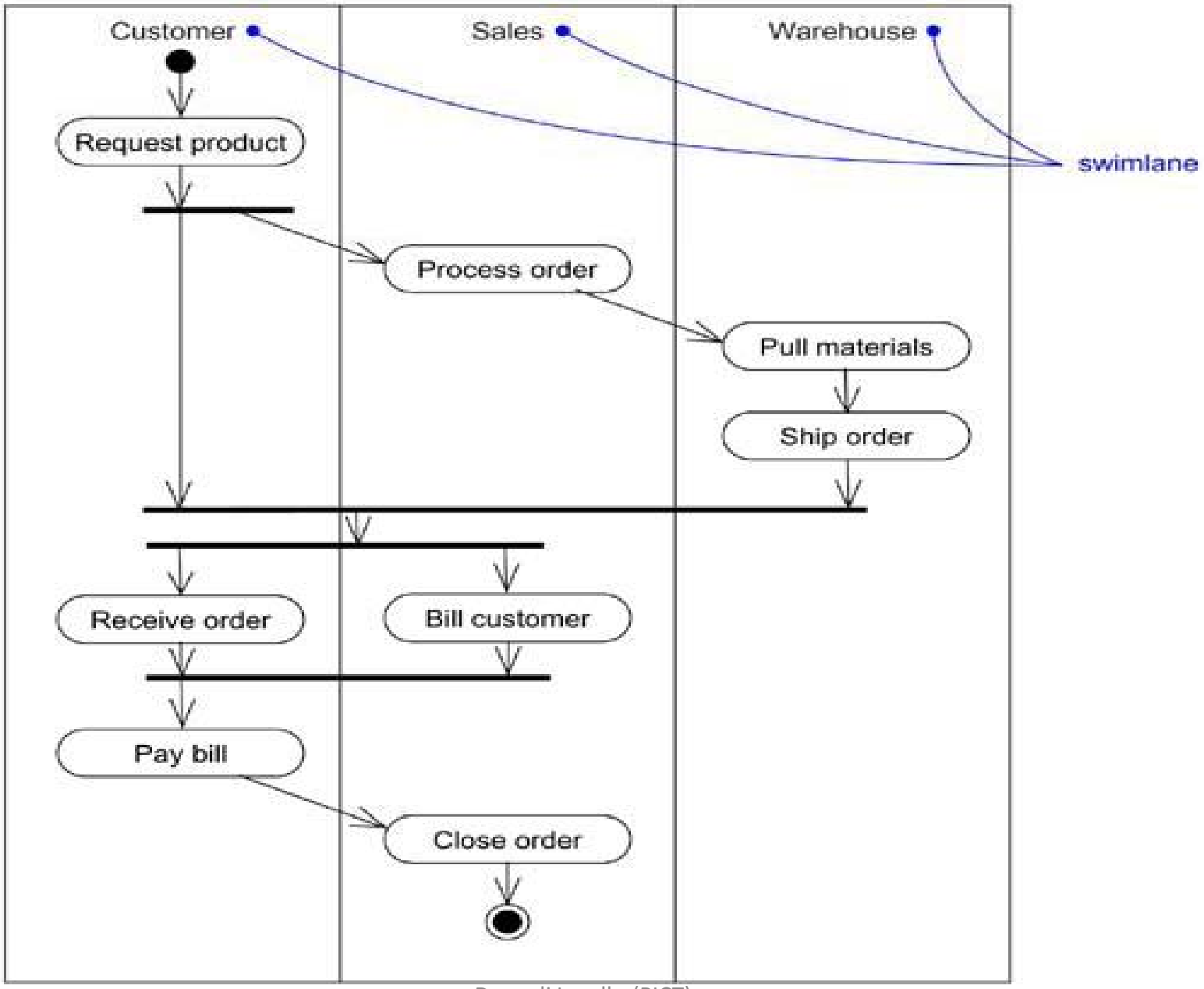
- Joins and forks should balance, meaning that the number of flows that leave a fork should match the number of flows that enter its corresponding join.
- Also, activities that are in parallel flows of control may communicate with one another by sending signals.
- This style of communicating sequential processes is called a coroutine. Most of the time, you model this style of communication using active objects.
- You can also model the sending of and response to these signals in the submachines associated with each communicating activity state

Swim lanes

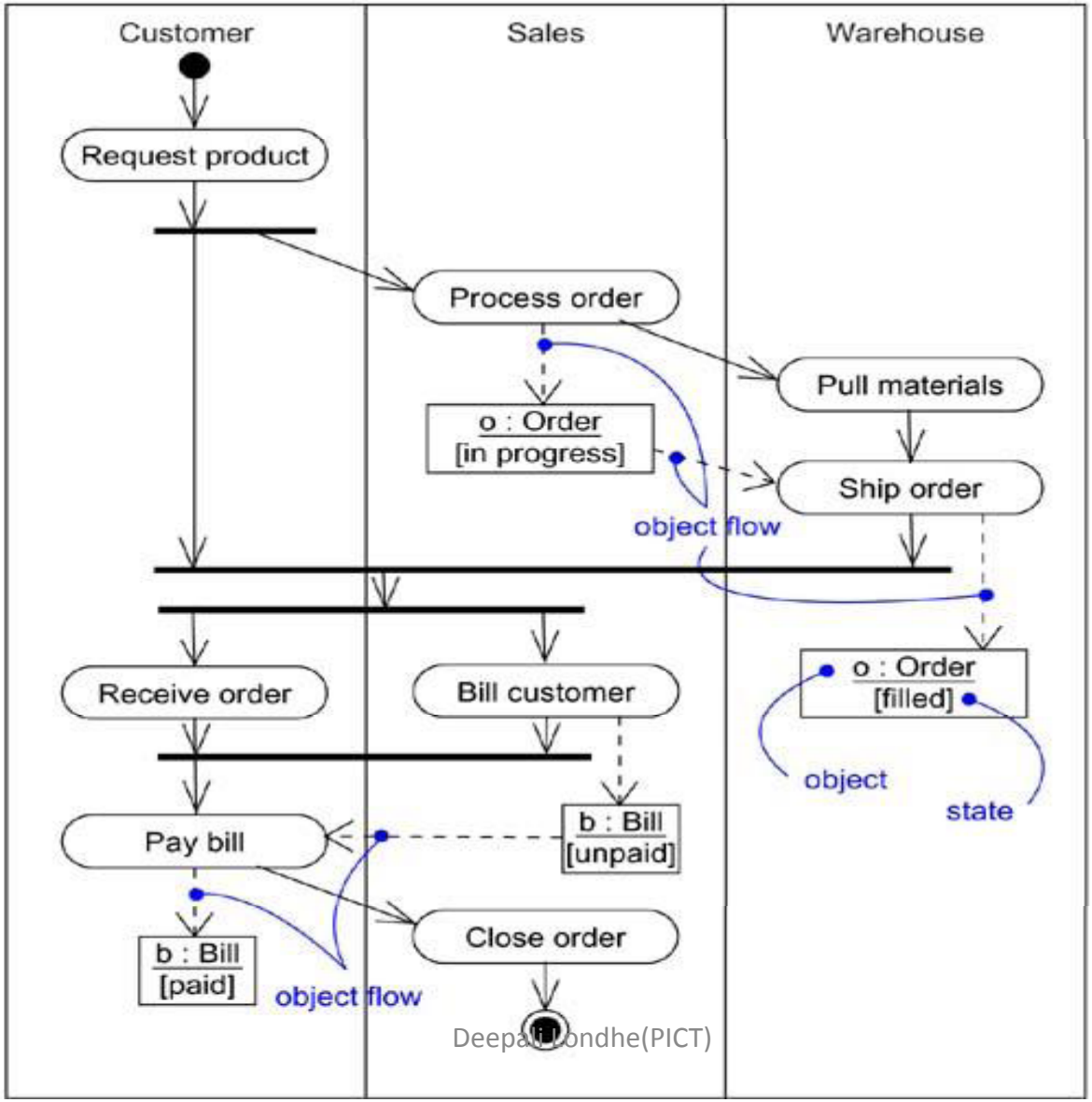
- when you are modeling workflows of business processes, to partition the activity states on an activity diagram into groups, each group representing the business organization responsible for those activities. In the UML, each group is called a swimlane
- Process can sometimes be distributed over several agents or organizational units
- Notation: use compartments
- In particular useful when modeling a business process (e.g. in organization model)
- *A swimlane is a kind of package;*

Notation for swim lanes




Customer
Sales
Warehouse
swimlane
Request product
Process order
Pull materials
Ship order
Receive order
Bill customer
Pay bill
Close order

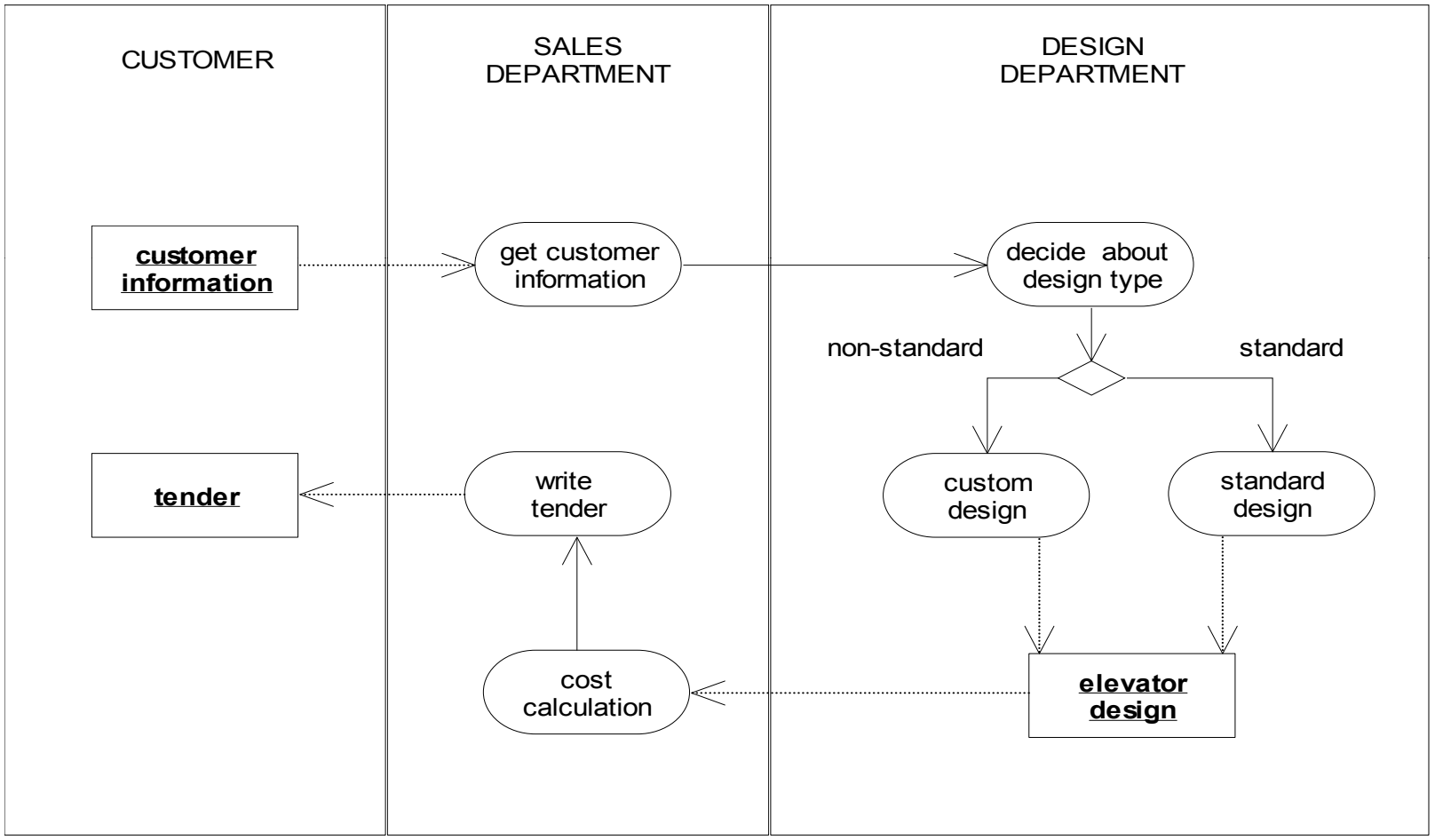
Object flow


Customer
Sales
Warehouse
Request product
Process order
Pull materials
o : Order
[in progress]
Ship order
object flow
Receive order
Bill customer
o : Order
[filled]
object
state
Pay bill
b : Bill
[unpaid]
Close order
b : Bill
[paid]
object flow

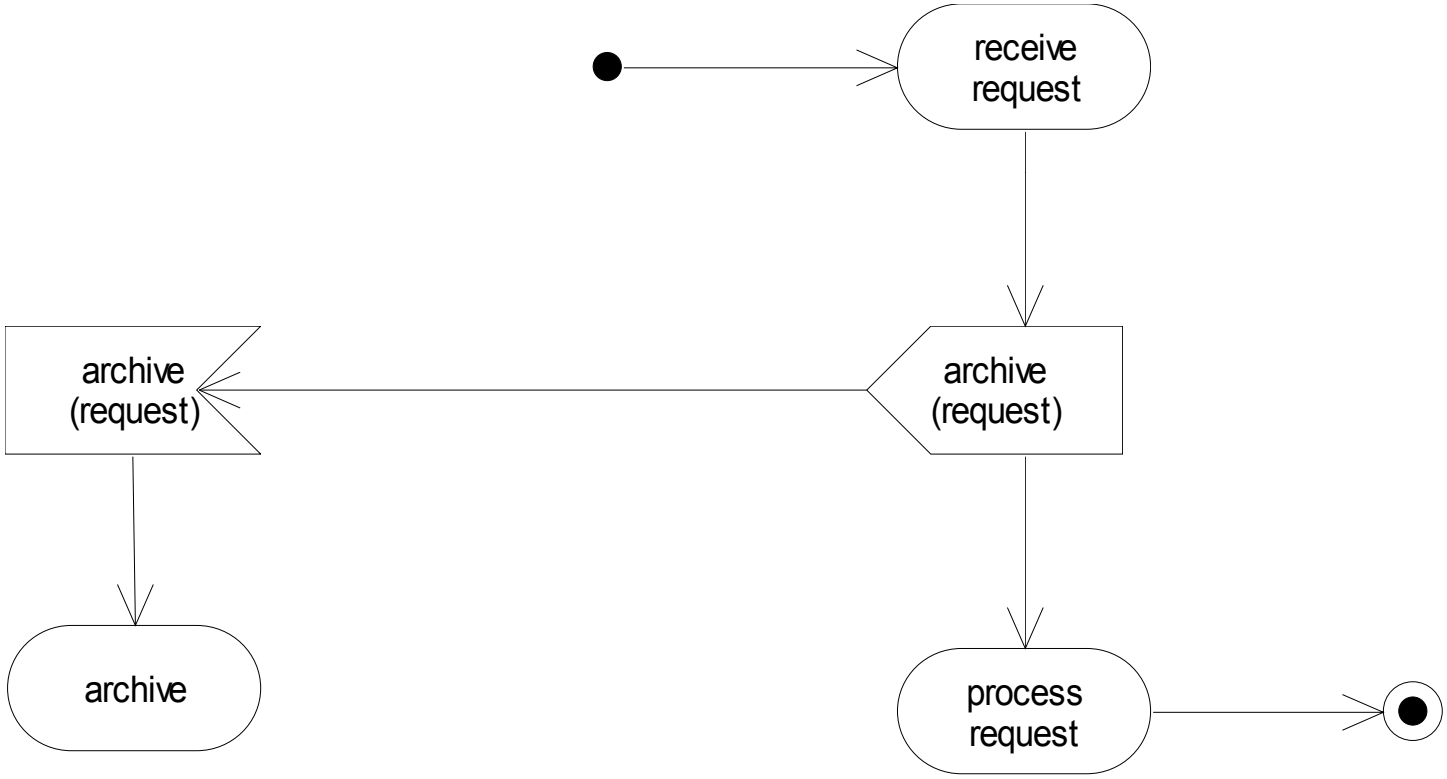
Object flow

- Objects may be involved in the flow of control associated with an activity diagram.
- For example, in the workflow of processing an order as in the previous figure, the vocabulary of your problem
- space will also include such classes as Order and Bill. Instances of these two classes will be produced by certain activities (Process order will create an Order object, for example); other activities may modify these objects (for example, Ship order will change the state of the Order object to filled).

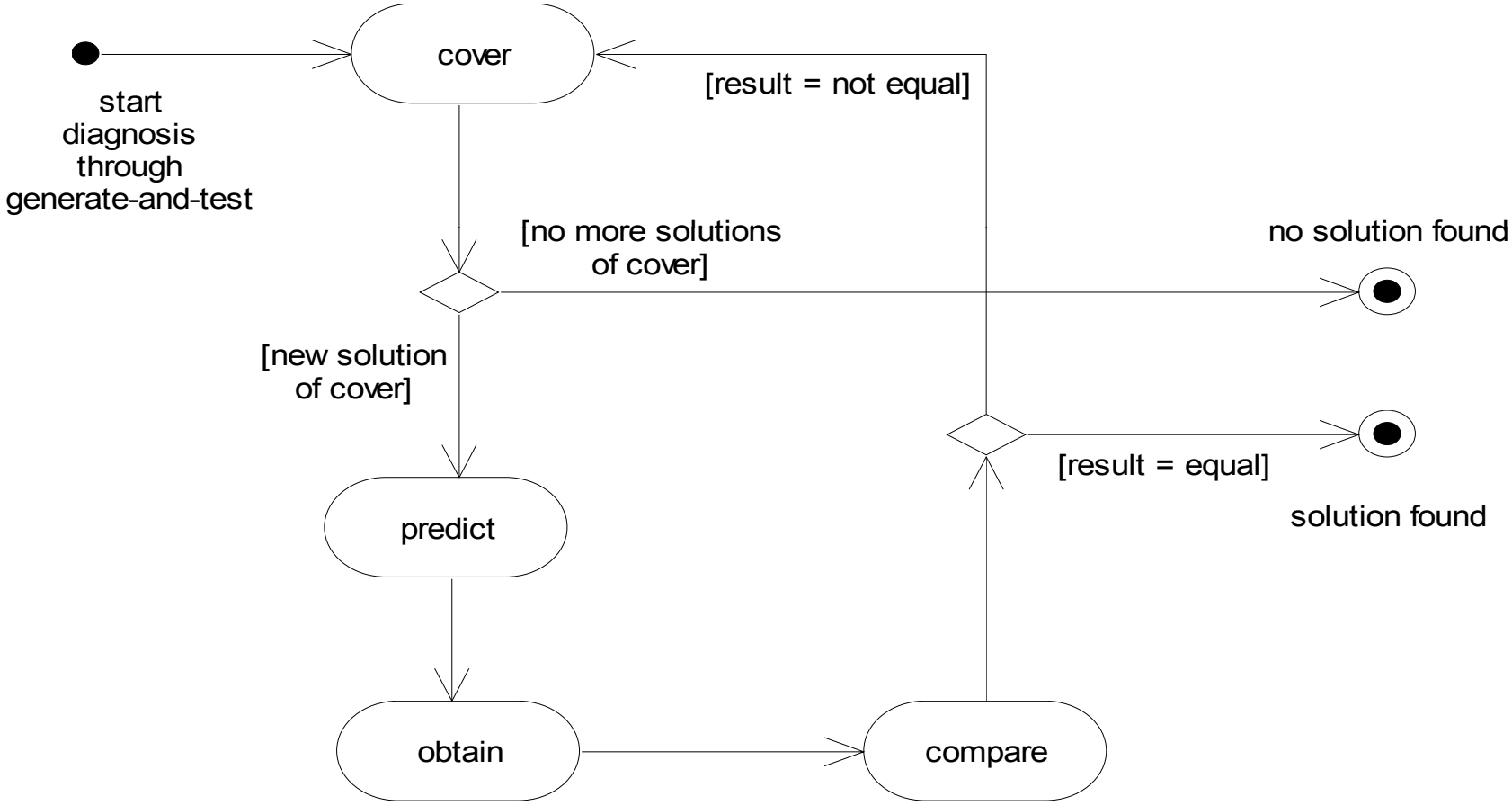
Object flow


CUSTOMER
SALES DEPARTMENT
DESIGN DEPARTMENT
customer information
get customer information
decide about design type
non-standard
standard
tender
write tender
custom design
standard design
cost calculation
elevator design

Signals


receive
request
archive
(request)
archive
(request)
archive
process
request

Activity diagram of method control



Thank You