# State Transition Diagram

**Lecture # 37, 38**
**1, 2 Dec**

Rubab Jaffar
rubab.jaffar@nu.edu.pk

# Software Design and Analysis
CS-3004

# Today's Outline

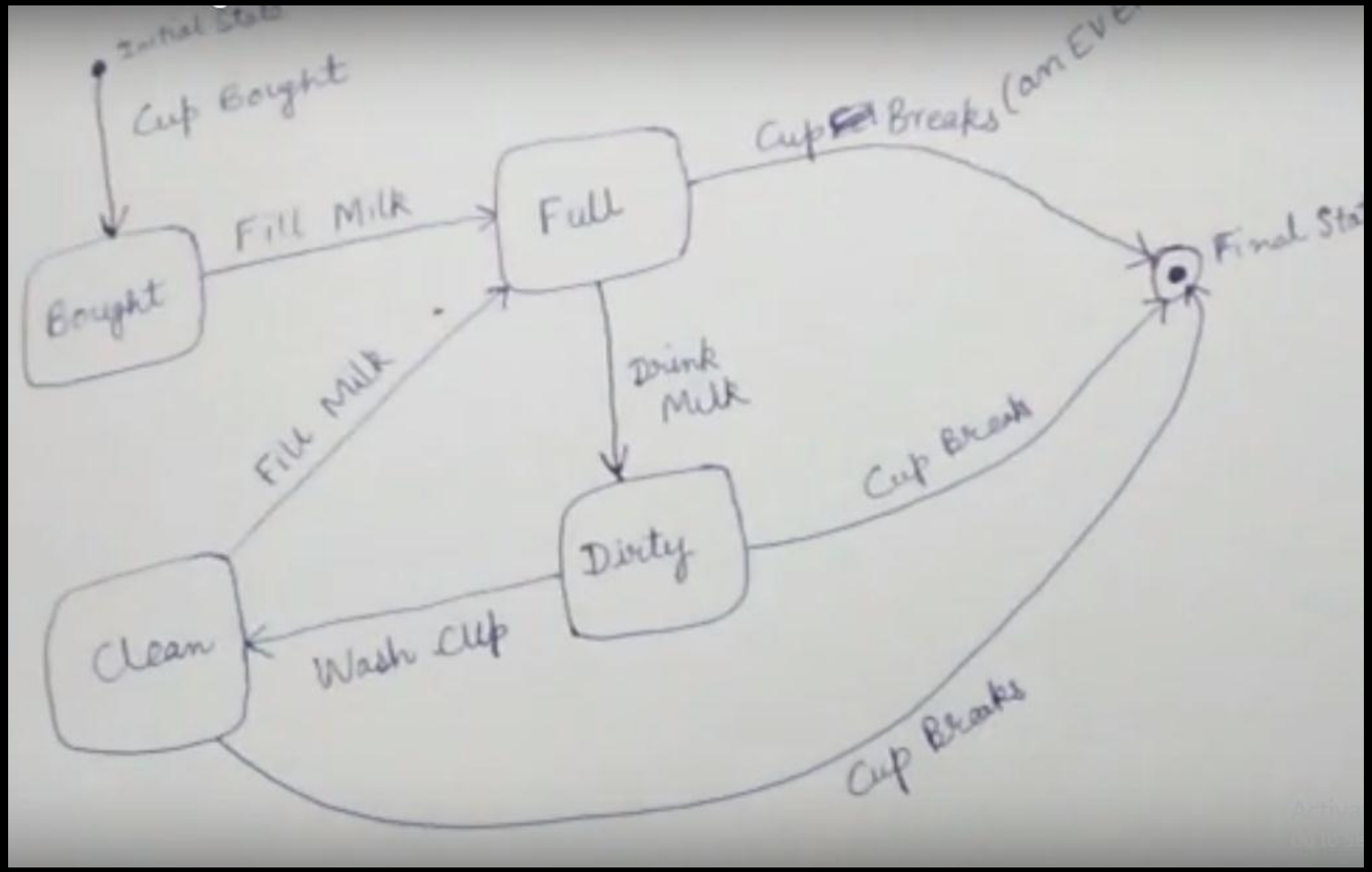
- State chart Diagrams
- Elements of State chart Diagrams
- Notations of State chart Diagrams
- Why State chart Diagrams?
- States
  - Composite States
  - Concurrent States
- Transitions
- Events

i. Static
    a. Use case diagram
    b. Class diagram
ii. Dynamic
    a. Activity diagram
    b. Sequence diagram
    c. Object diagram
    d. State diagram
    e. Collaboration diagram
iii. Implementation
    a. Component diagram
    b. Deployment diagram

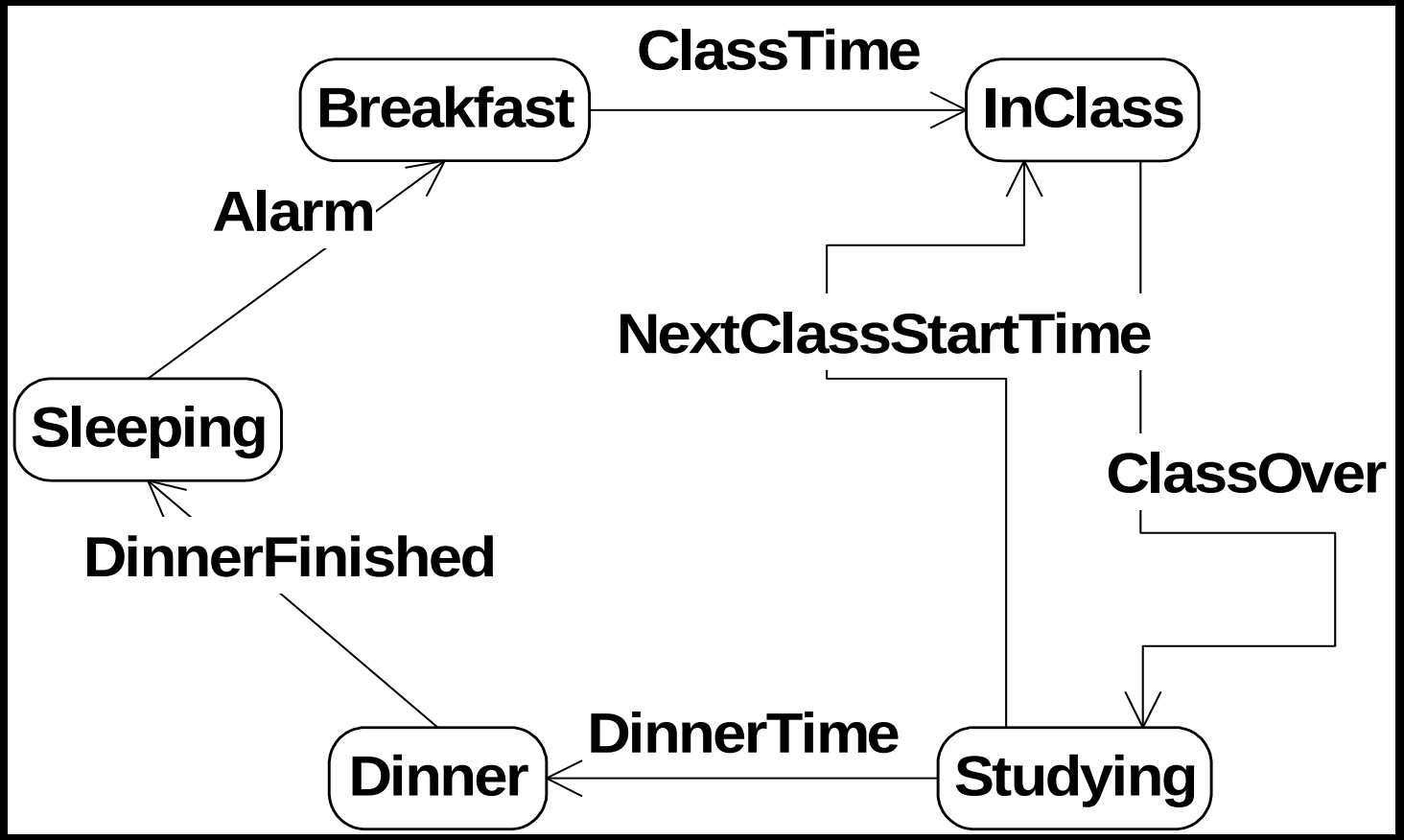
# State Chart Diagrams

- A state machine diagram models the behavior of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.
- A state chart diagram is normally used to model how the state of an object changes in its lifetime.
- State chart diagrams are good at describing how the behavior of an object changes across several use case executions. However, if we are interested in modeling some behavior that involves several objects collaborating with each other, state chart diagram is not appropriate.

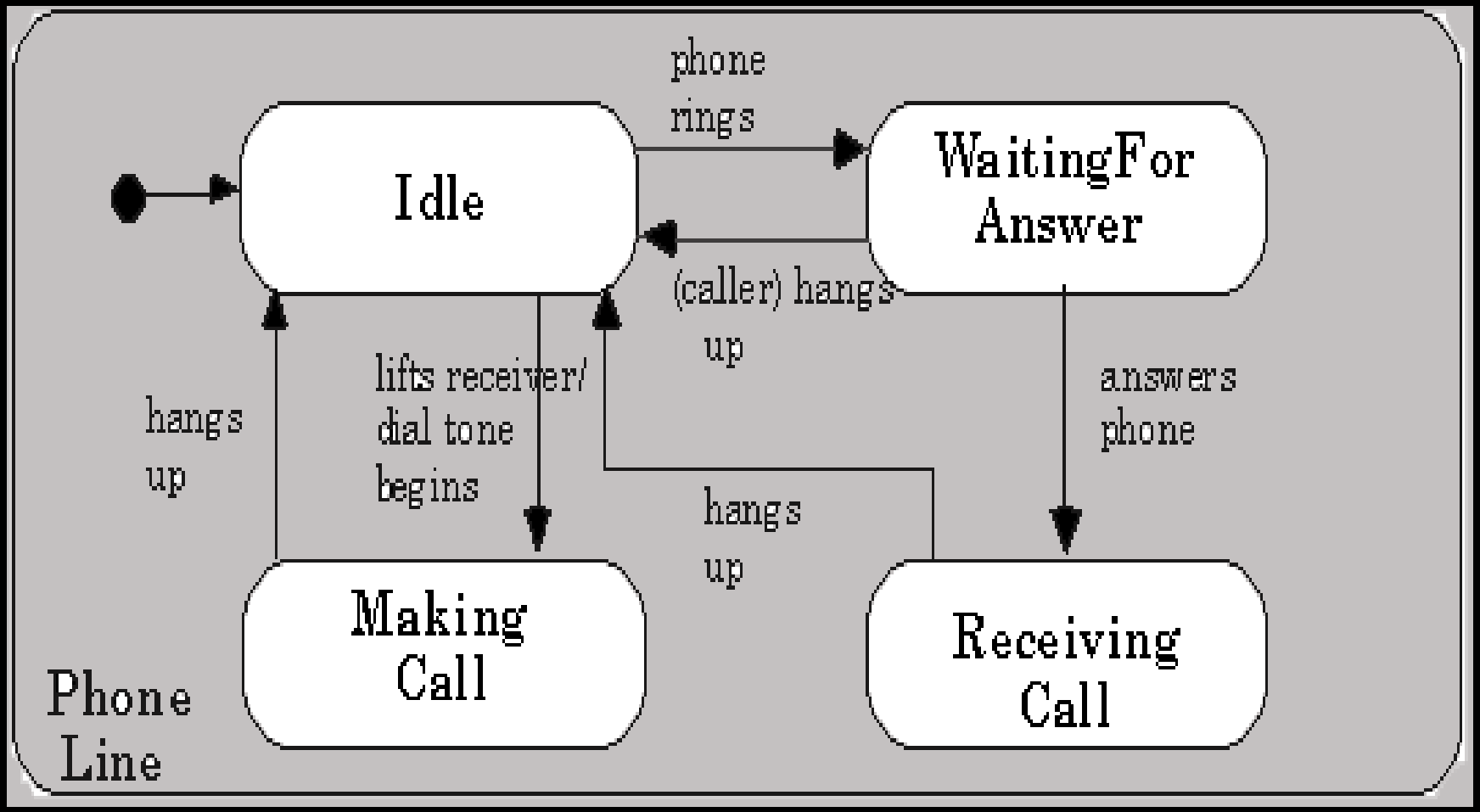
# State Diagram-Example 1



# State Diagram-Example 2



# State Chart Diagram – Phone Line



# Why Use State chart Diagrams?

- State charts typically are used to describe state-dependent behavior for an object
  - An object responds differently to the same event depending on what state it is in.
  - Usually applied to objects but can be applied to any element that has behavior
    - Actors, use cases, methods, subsystems, systems
- Statecharts are typically used in conjunction with interaction diagrams (usually sequence diagrams)
  - A statechart describes all events (and states and transitions for a single object)
  - A sequence diagram describes the events for a single interaction across all objects involved

# State Chart Diagrams

- State diagrams describe the life of an object using three main elements:
  - States of an object
  - Transitions between states
  - Events that trigger the transitions

# States

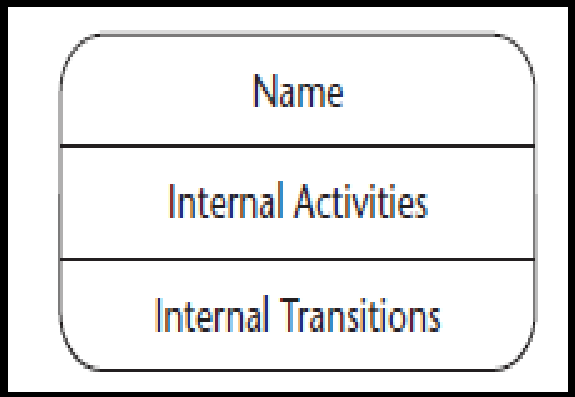

- State is a condition or situation during the life of an object within which it performs some activity, or waits for some events .
- State may also label activities, e.g., do/check item
- Examples of object states are:
  - The invoice (object) is paid (state).
  - The car (object) is standing still (state).
  - The engine (object) is running (state).
  - Jim (object) is playing the role of a salesman (state).

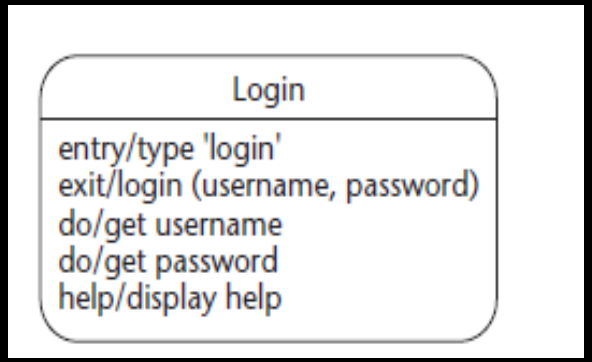
# State Compartments

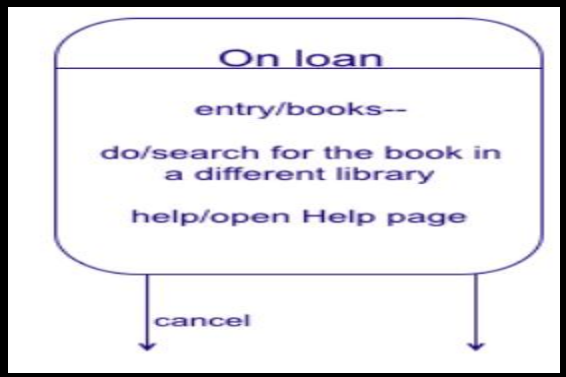
- A state may contain three kinds of compartments.
- The first compartment shows the name of the state, for example, idle, paid, and moving.
- The second compartment is the optional **activity compartment**, which lists behavior in response to events. You can define your own event, such as selecting a Help button, as well as the activity to respond to that event.
- Three standard events names are reserved in UML: entry, exit, and do.
- The entry event can be used to specify actions on the entry of a state; for example, assigning an attribute or sending a message.
- The exit event can be used to specify actions on exit from a state.
- The do event can be used to specify an action performed while in the state; for example, sending a message, waiting, or calculating.

# State Compartments

- The third compartment is the optional internal transition compartment. This compartment contains a list of internal transitions. A transition can be listed more than once if it has different guard conditions.



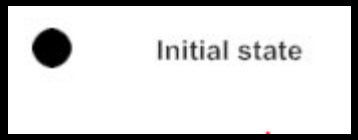



# Initial/Final States

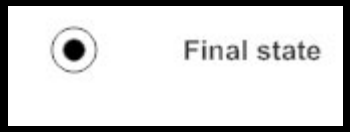
- **Initial State**

  A filled circle followed by an arrow represents the object's initial state.

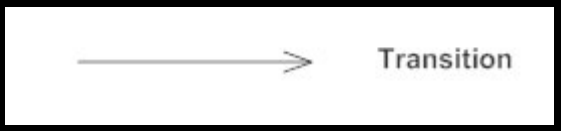



- **Final State**

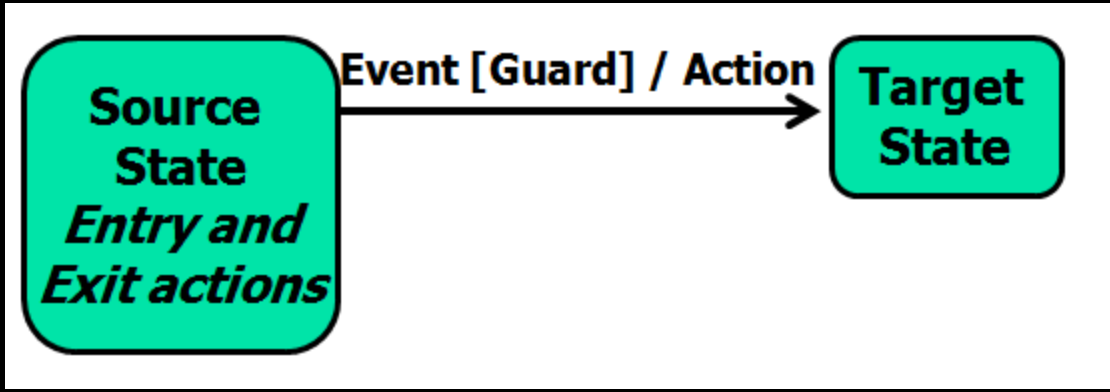
  An arrow pointing to a filled circle nested inside another circle represents the object's final state.



# Transition

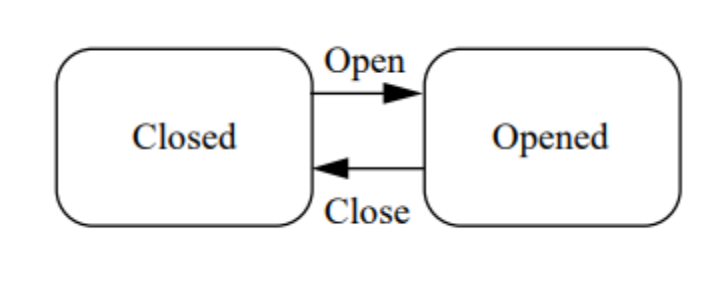



- Show what the dependencies are between the state of an object and its reactions to messages or other events
- A solid arrow represents the path between different states of an object.
- Label the transition with the event that triggered it and the action that results from it.
- A state can have a transition that points back to itself.



# Examples

- If a Bank Account was Closed and it saw an Open event, it would end up in the Opened state - If the account was Opened and it saw a Close event it would end up in the Closed state.



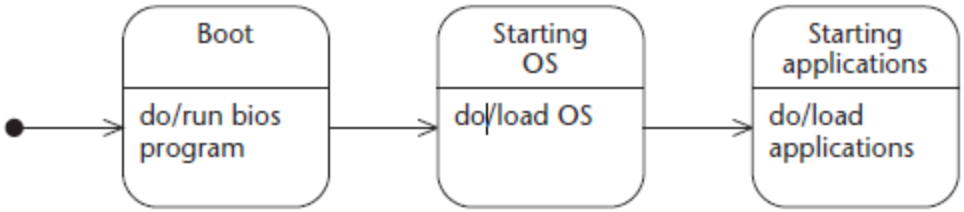
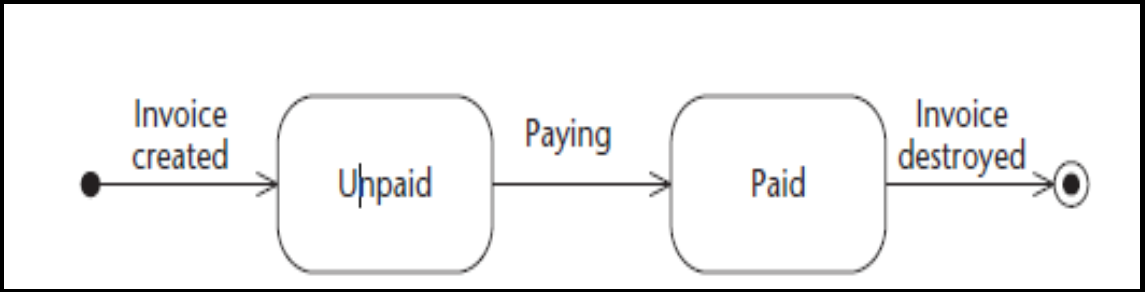




**Figure 5.6** State transitions without explicit events. The transitions occur when the activities in each state have been performed.

# Events

- An object transitions (changes) from one state to another state when something happens, which is called an event; for example, someone pays an invoice, starts driving the car,
- An event is an instant in time that may be significant to the behavior of the objects in a class - Events can have associated arguments
- Events are written simply as text strings
  - Open
  - Deposit(Amount)
  - Withdraw(Amount)
  - Close



# Types of Events

- UML has four types of events, also referenced as triggers because the event triggers behavior that depends on how you build the system.
- A condition becoming true. This is shown as a guard-condition on a state transition.
- Receipt of an explicit signal from another object. The signal is itself an object. This type of event is called a message.
- Receipt of a call on an operation by another object (or by the object itself). This type of event is also called a message.
- Passage of a designated period of time.

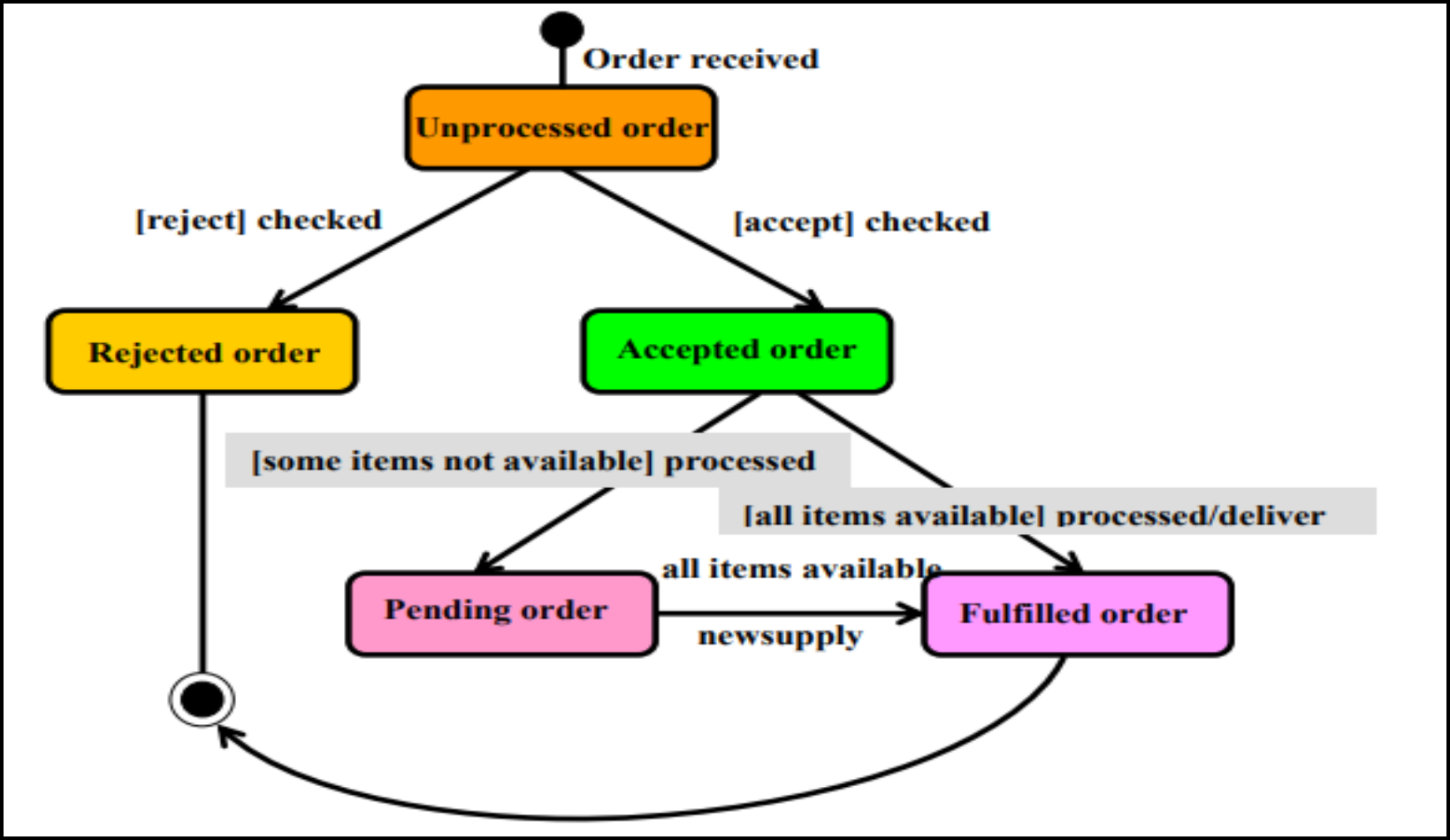
# Guard Condition

- Guard-condition is a Boolean expression placed on a state transition. If the guard-condition is combined with an event-signature, the event must occur, *and the guard-condition must be true for the transition to fire*.
- *If only a guardcondition* is attached to a state transition, the transition fires when the condition becomes true Examples of state transitions with a guard-condition are as follows:
  - [t = 15sec]
  - [number of invoices > n]
  - withdrawal (amount) [balance >= amount]

# Example

- Here is just an example of how an online ordering system might look like :
- On the event of an order being received, we transit from our initial state to Unprocessed order state.
- The unprocessed order is then checked.
- If the order is rejected, we transit to the Rejected Order state.
- If the order is accepted and we have the items available we transit to the fulfilled order state.
- However if the items are not available we transit to the Pending Order state.
- After the order is fulfilled, we transit to the final state. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.

# State chart diagram for an order object



# Internal transitions

- An event causes only some internal actions to execute but does not lead to a change of state (state transition). In this case, all actions executed comprise the **internal transition**.

- For example, when one types on a keyboard, it responds by generating different character codes. However, unless the Caps Lock key is pressed, the state of the keyboard does not change (no state transition occurs).

# Internal transitions

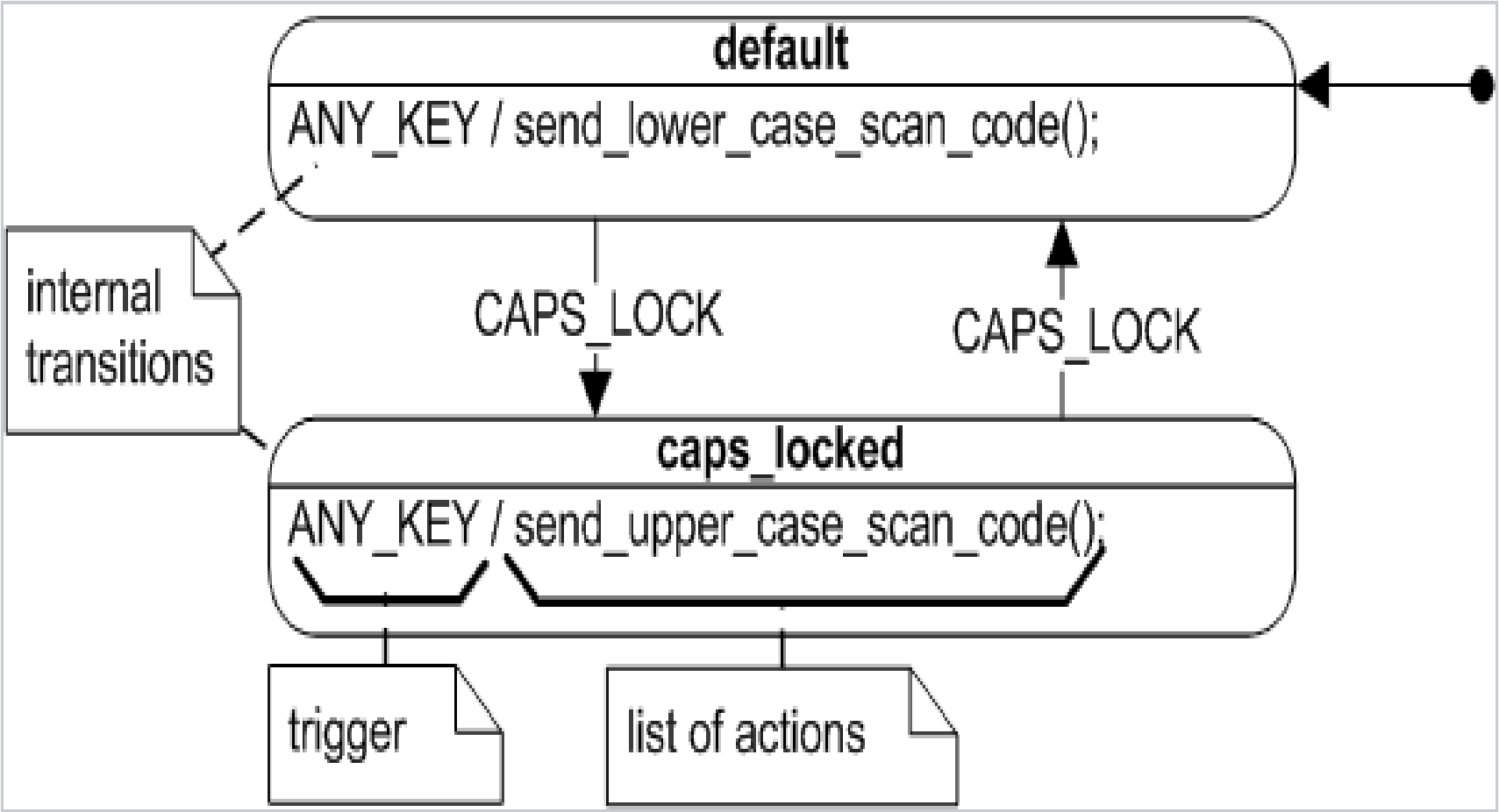
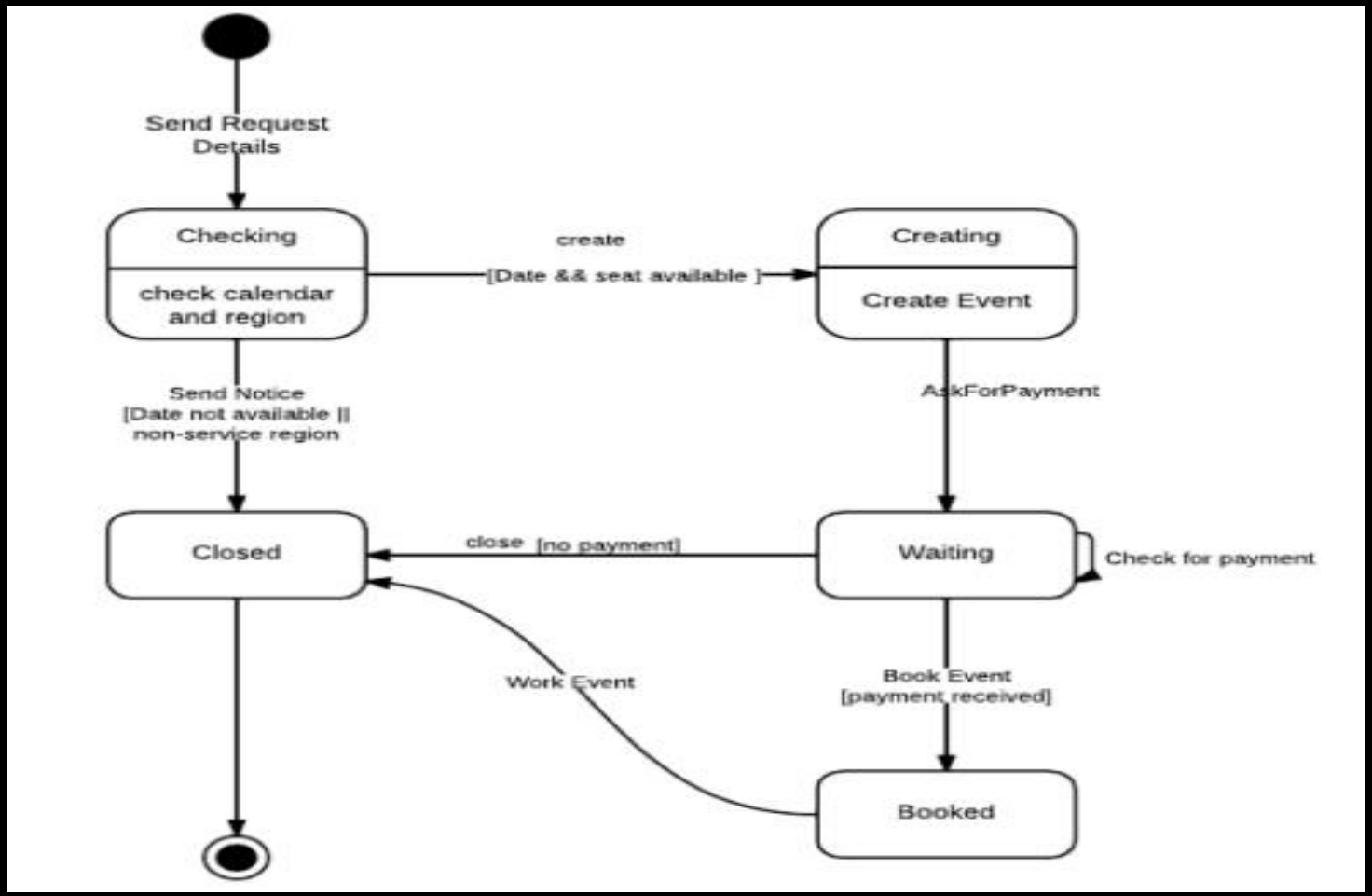



Figure 6: UML state diagram of the keyboard state machine with internal transitions

# Draw a statechart to describe the process of booking a seat for some special event.

- Customer checks the event schedule for a particular event on a date and in a specific region.
- If the seat is available on that date then event is created.
- Customer is asked for payment. Event booking process is in waiting state until the customer makes the payment. If the customer doesn't make payment, event is closed.
- When the customer makes the payment seat is booked for the event and then closed.
- Event is also closed if on specific date and region, no event is available.

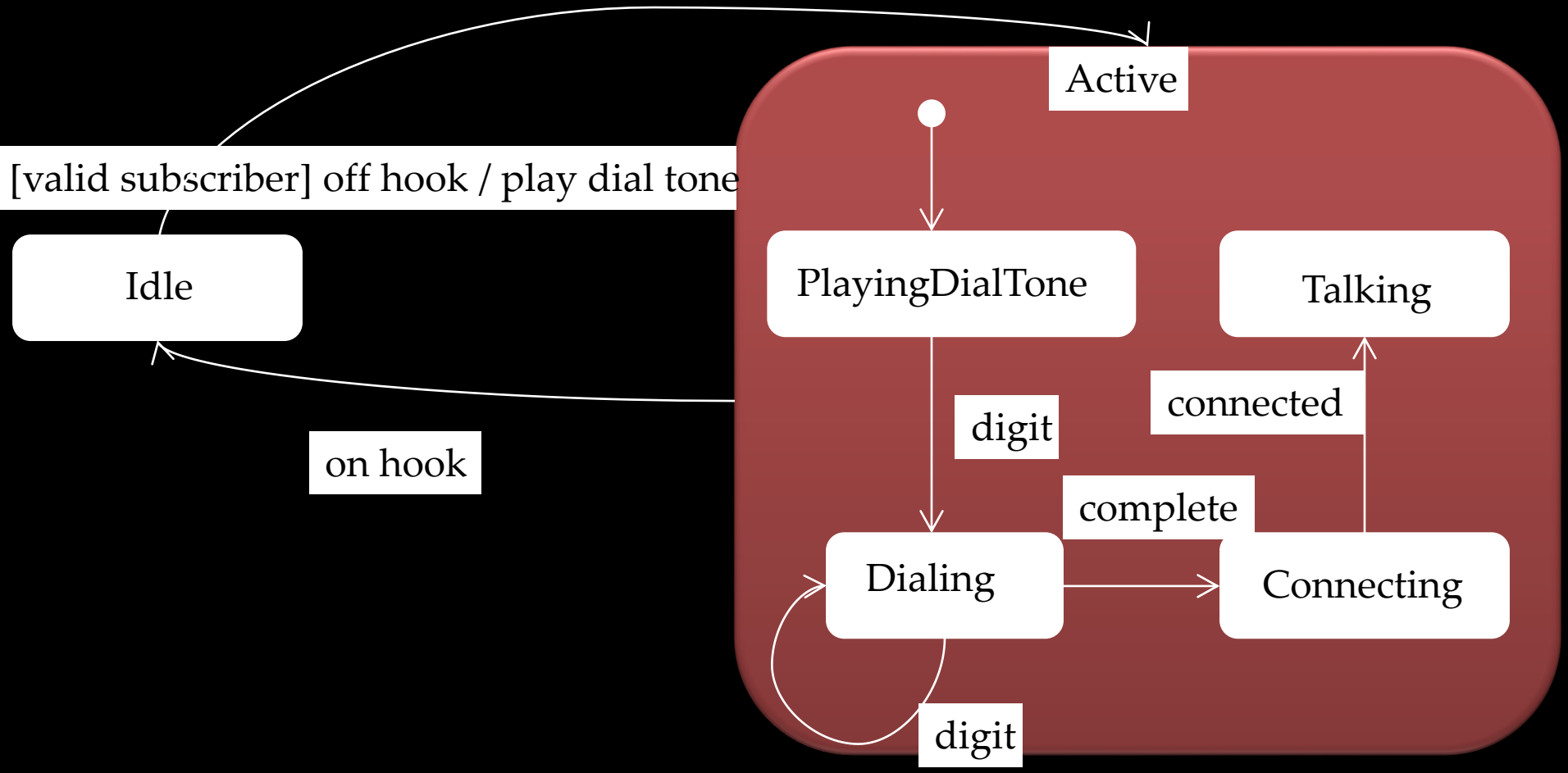
# Draw a statechart to describe the process of booking a seat for some special event.



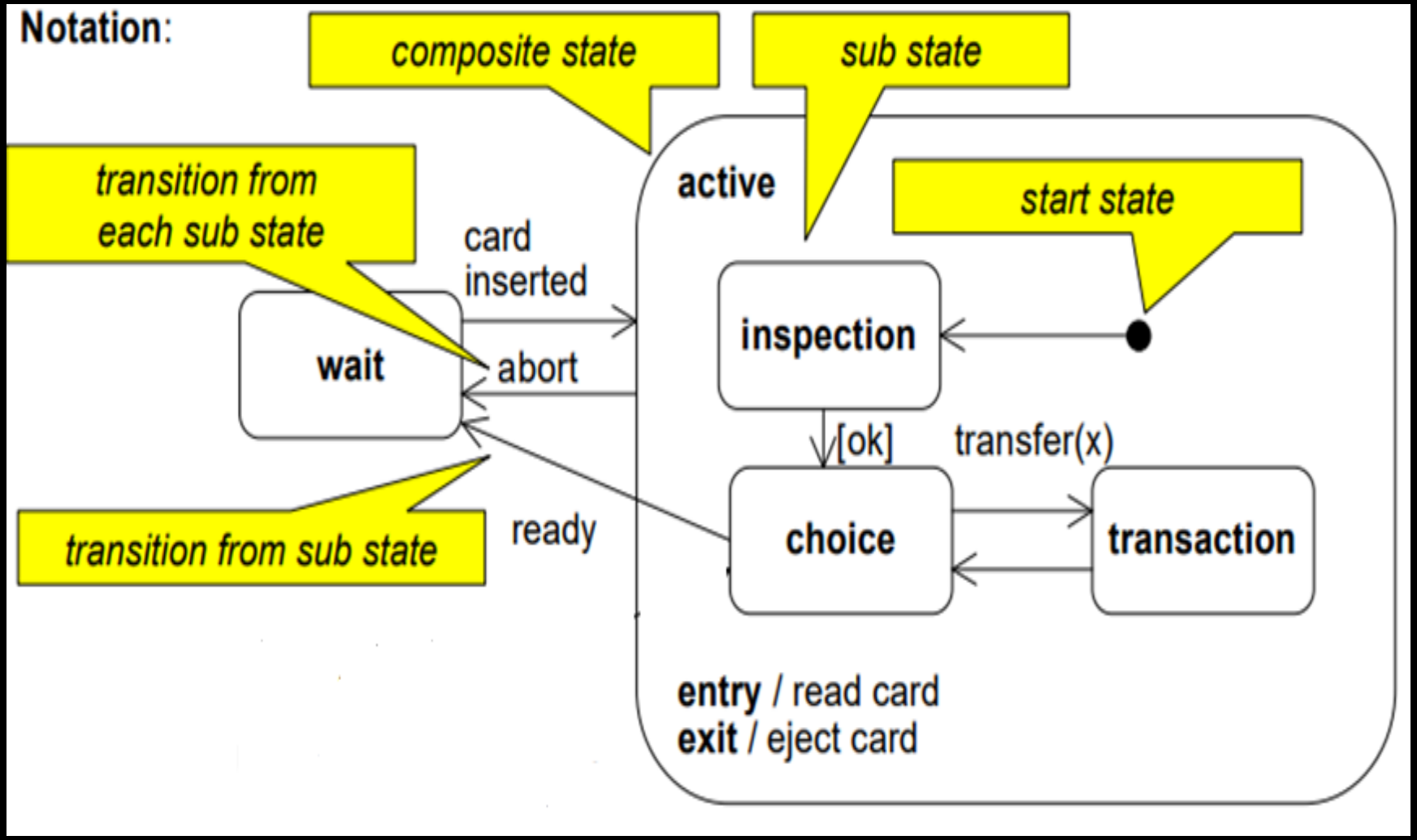
# Composite State

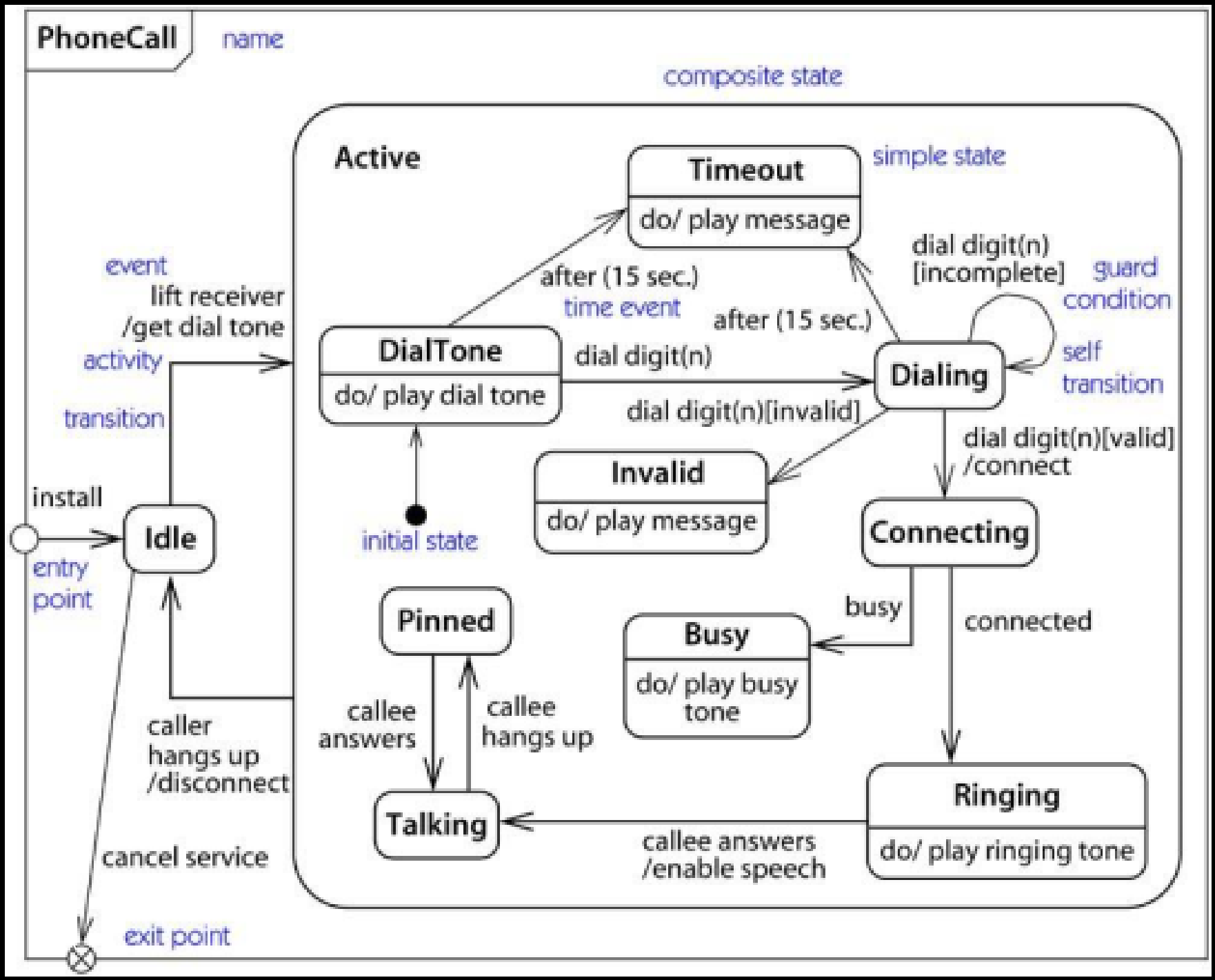
- A state can be refined hierarchically by composite states.
- Generally, **composite state** is defined as **state** that has substates (nested **states**). Substates could be sequential (disjoint) or concurrent (orthogonal).
- A state allows nesting to contain substates. A substate inherits the transitions of its superstate (the enclosing state).
  - Within the *Active* state, and no matter what substate the object is in, if the *on hook* event occurs, a transition to the *idle* state occurs.

# Composite States/Nested State

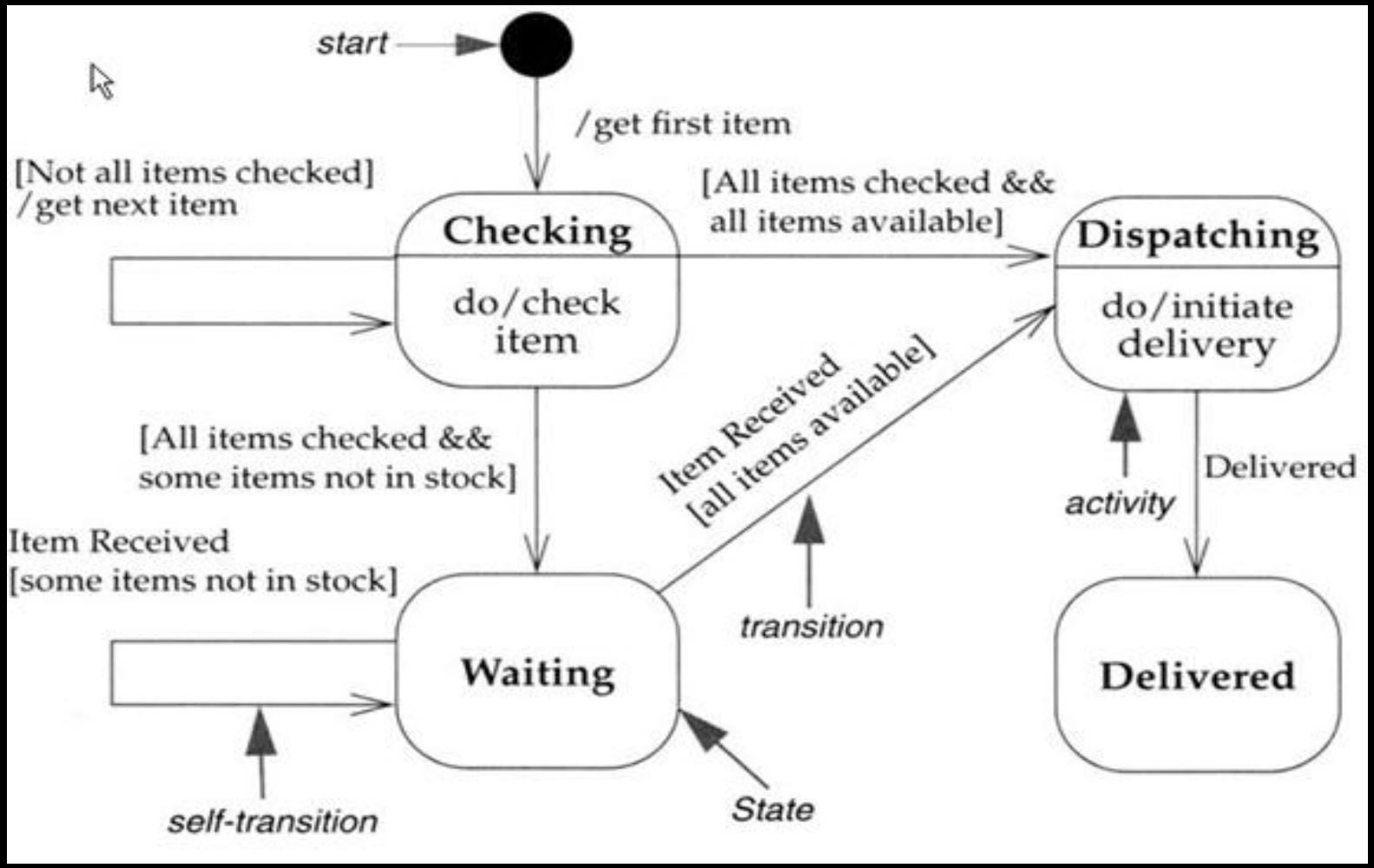


# Composite State

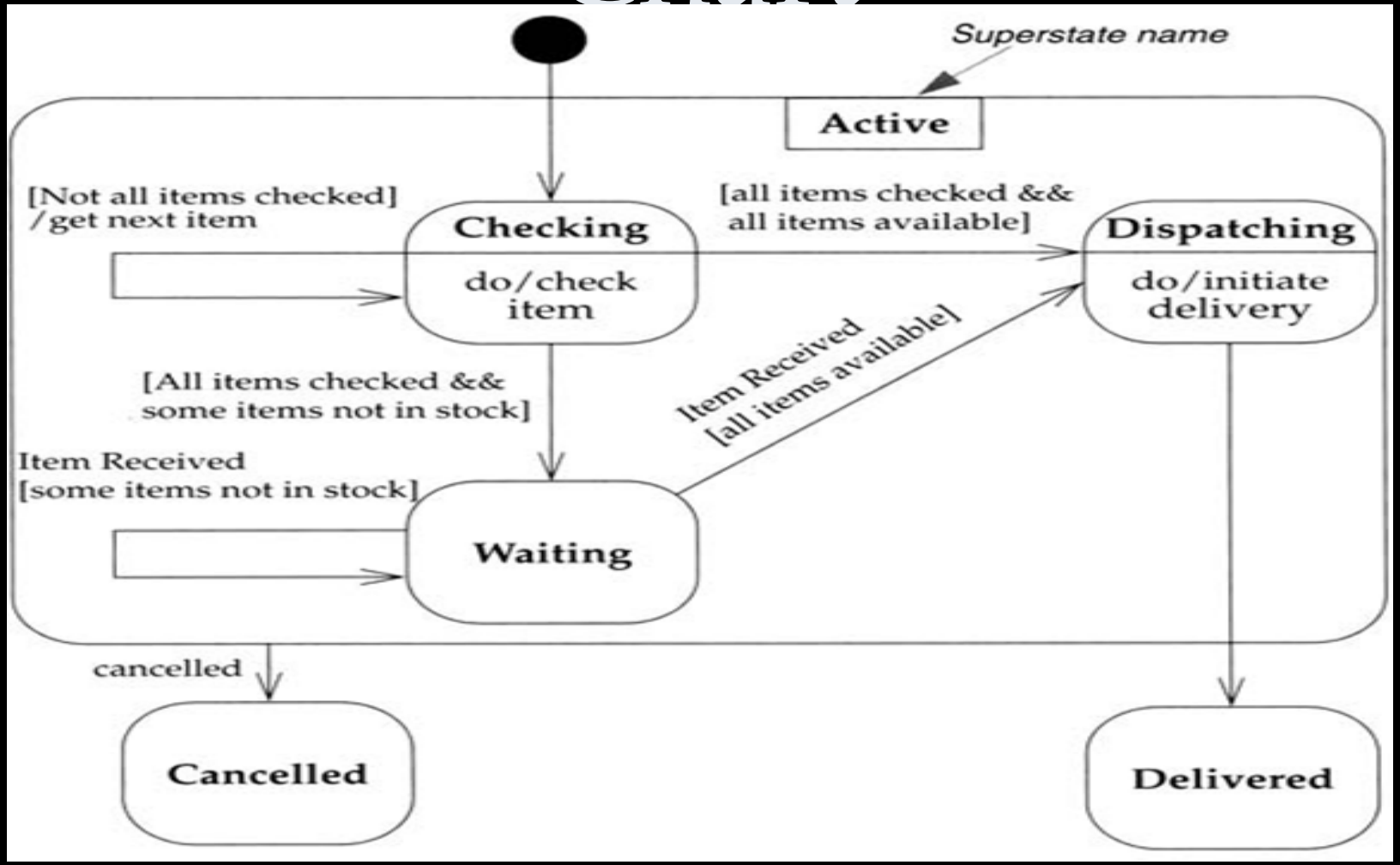



PhoneCall
name
composite state
Active
simple state
Timeout
do/ play message
event
lift receiver
/get dial tone
activity
transition
after (15 sec.)
time event
after (15 sec.)
dial digit(n)
[incomplete]
guard
condition
self
transition
DialTone
do/ play dial tone
dial digit(n)
Dialing
dial digit(n)[invalid]
dial digit(n)[valid]
/connect
Invalid
do/ play message
install
Idle
entry
point
initial state
Connecting
Pinned
busy
Busy
do/ play busy
tone
connected
callee
answers
callee
hangs up
caller
hangs up
/disconnect
Ringing
Talking
callee answers
/enable speech
do/ play ringing tone
cancel service
exit point

# Order Management State Chart



# Order Management State Chart



# Concurrency

- Concurrency on a state machine diagram can be expressed by an orthogonal state (a composite state with multiple regions).
- If an entering transition terminates on the edge of the orthogonal state, then all of its regions are entered.
- When exiting from an orthogonal state, each of its regions is exited

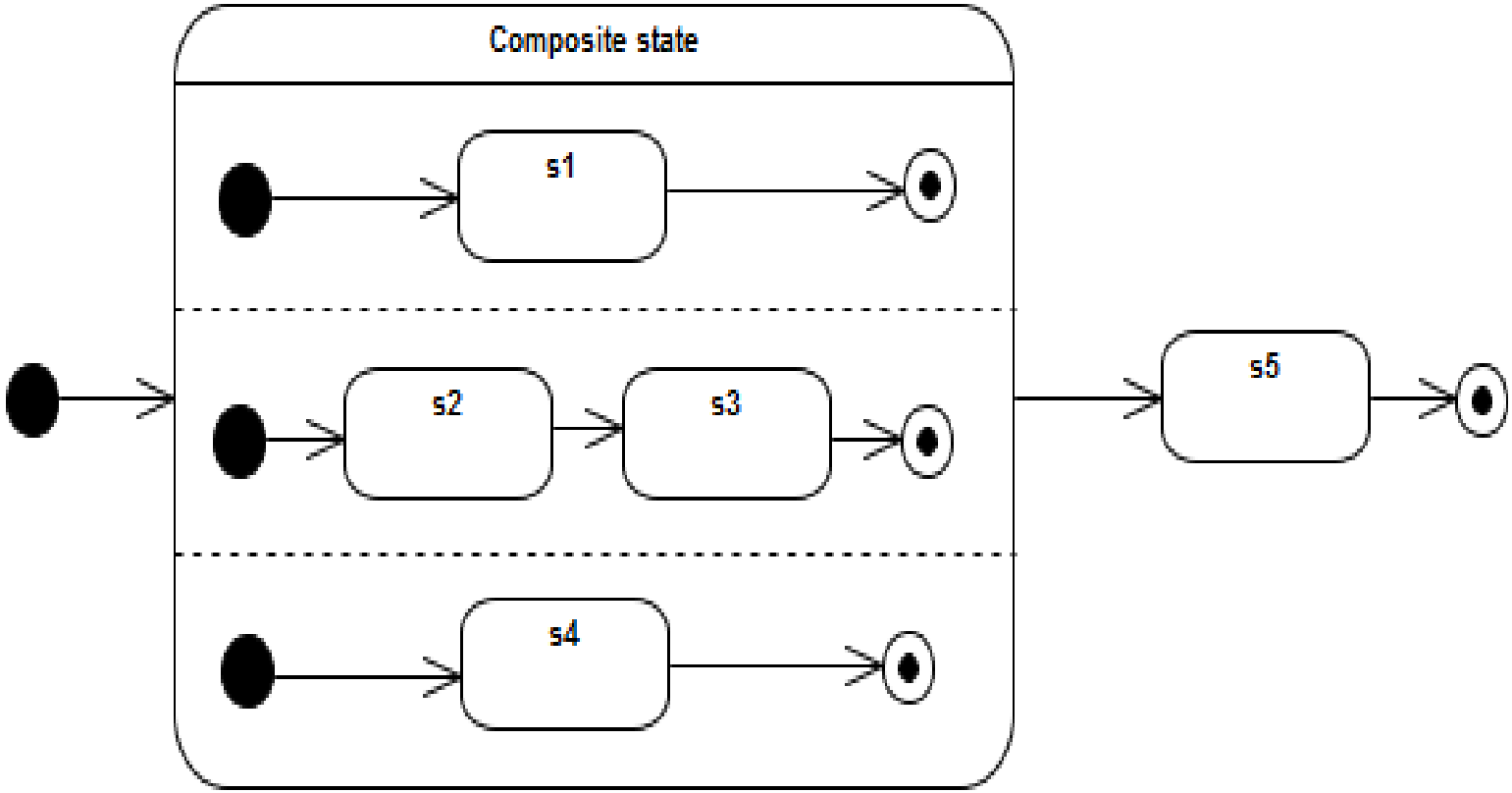



Figure 8. Orthogonal state

# Concurrency

- Concurrency can be shown explicitly using fork and join pseudostates.
- A fork is represented by a bar with one or more outgoing arrows terminating on orthogonal regions (i.e. states in different regions);
- A join merges one or more transitions.
- Concurrent substates specify two or more state machines that execute in parallel in the context of the enclosing object
- Execution of these concurrent substates continues in parallel. These substates waits for each other to finish to joins back into one flow

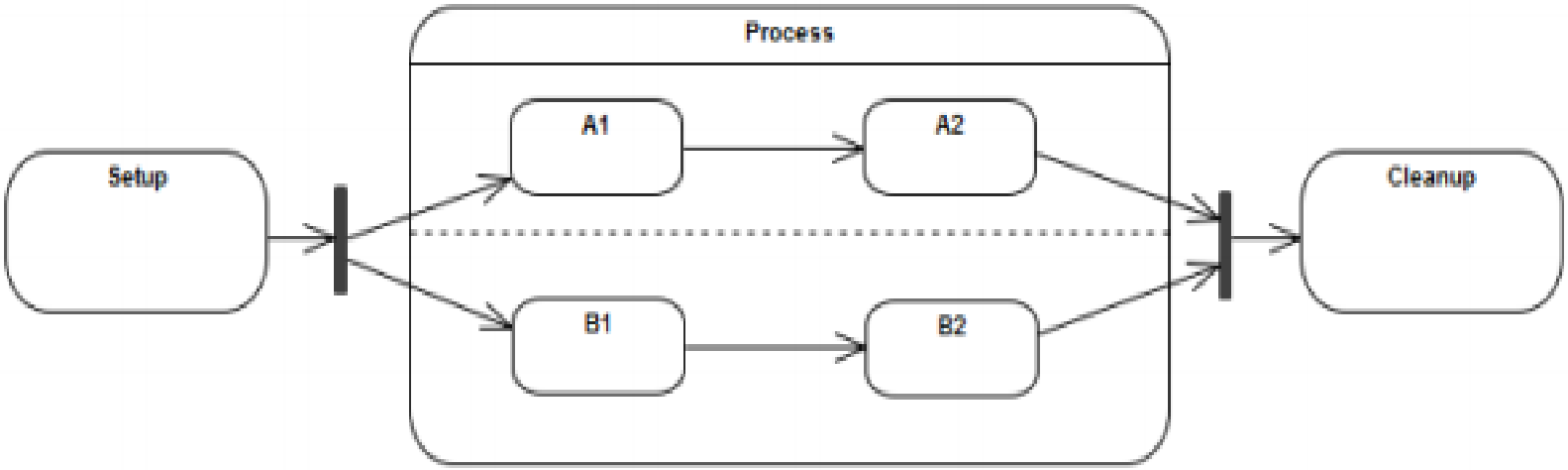
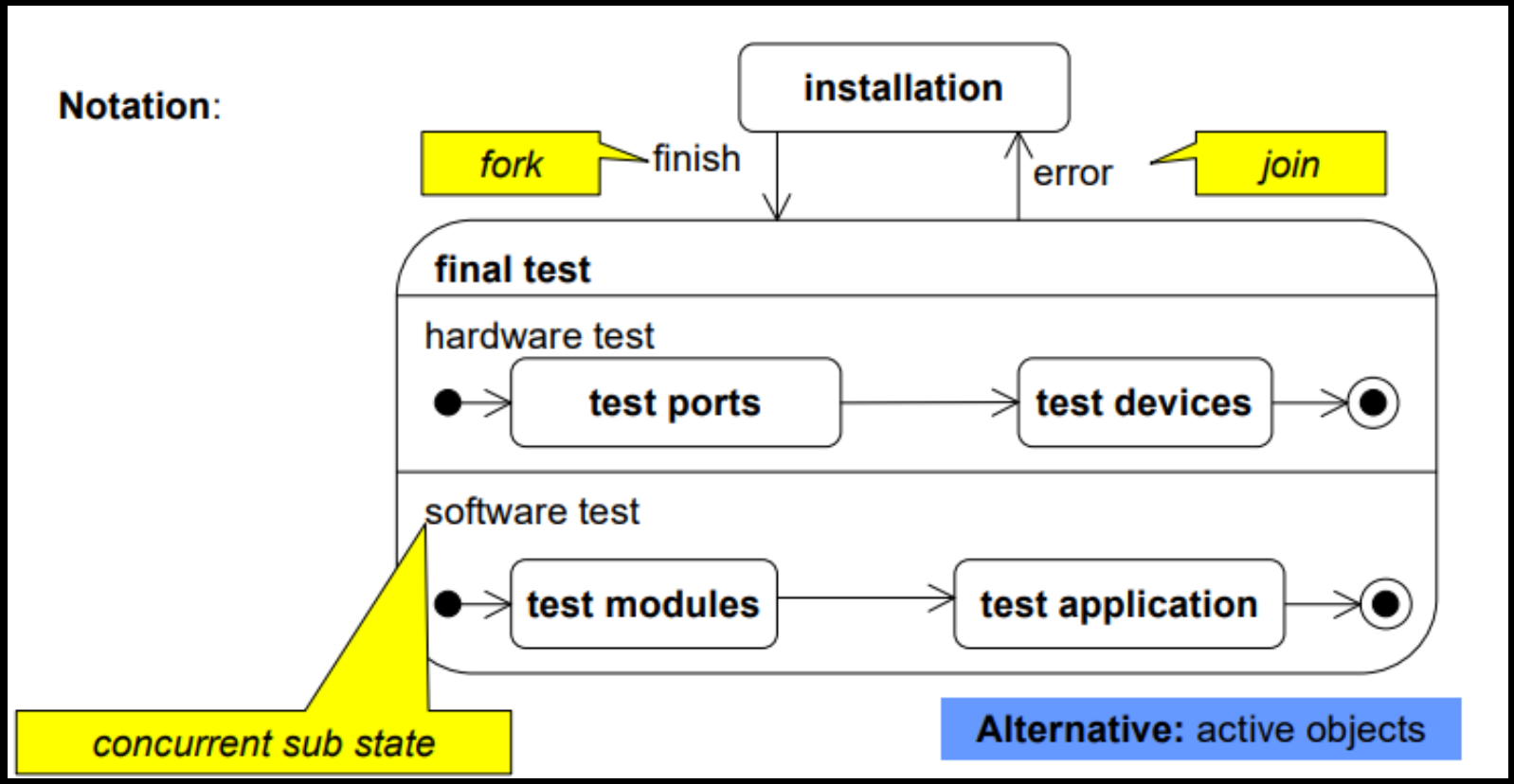


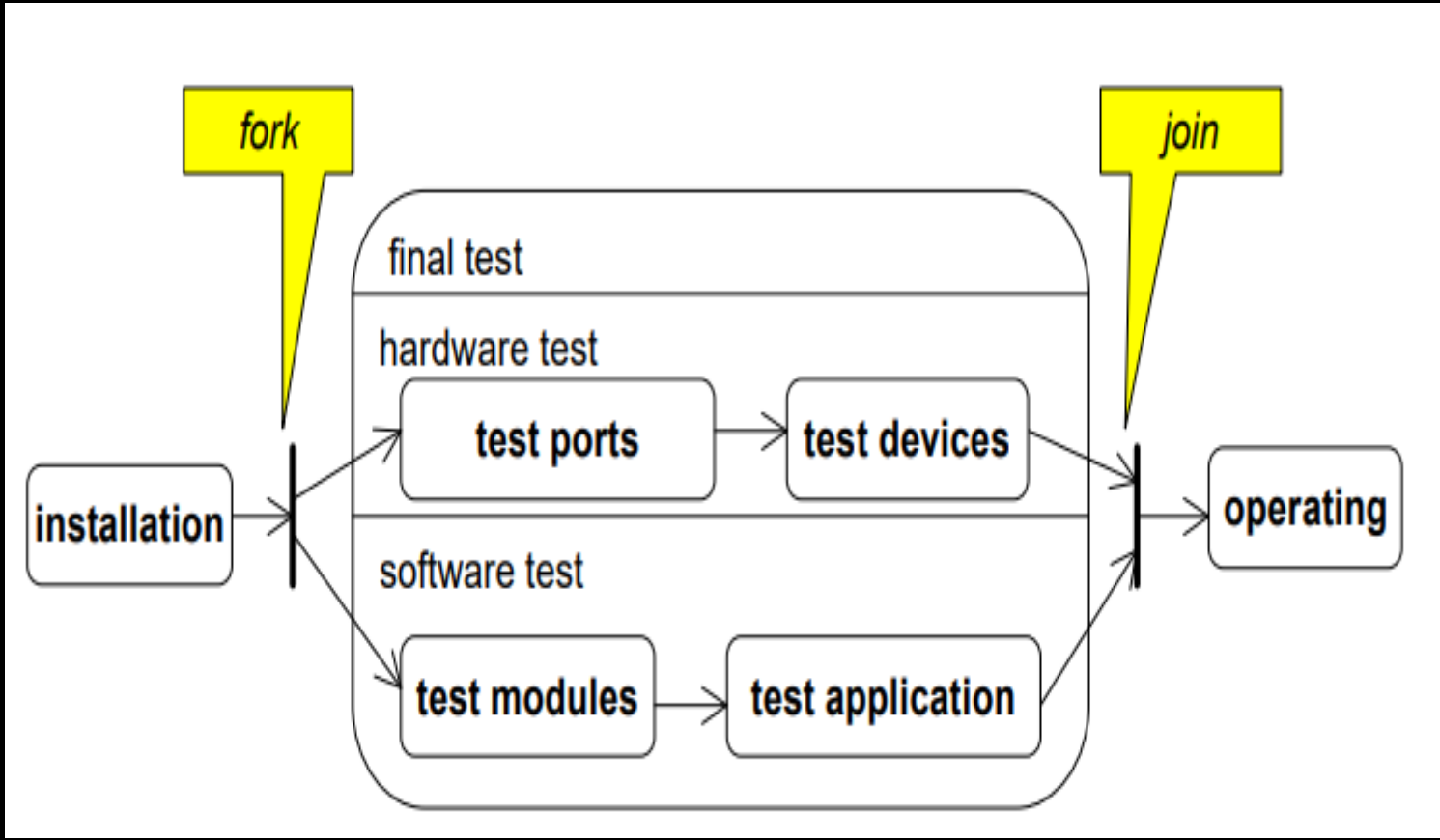

Figure 9. Fork and Join Pseudostates

# Concurrent Sub States

- In a state several sequences of sub states described by own state machines can be performed concurrently.

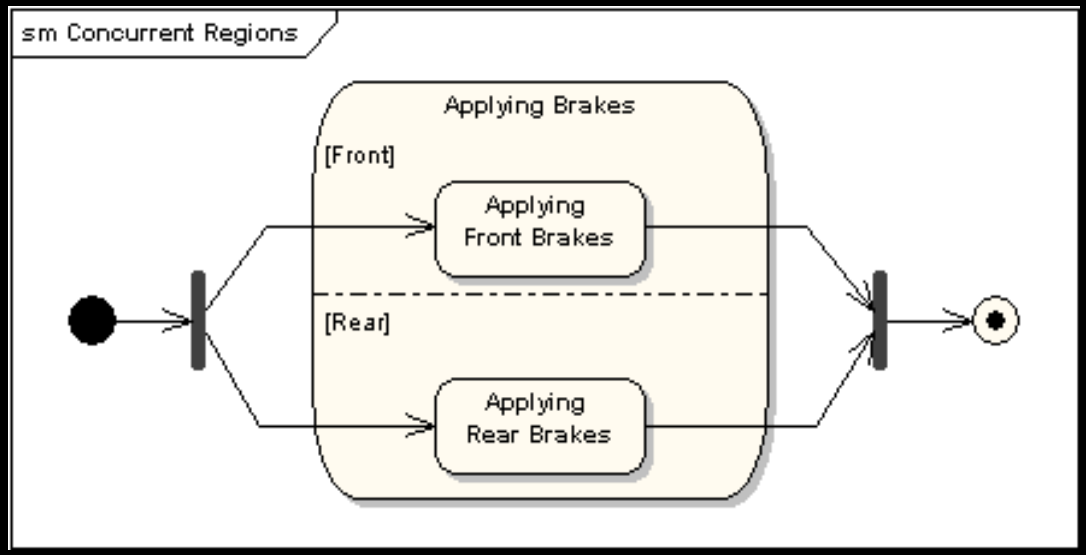


# Concurrent Sub States: Alternative



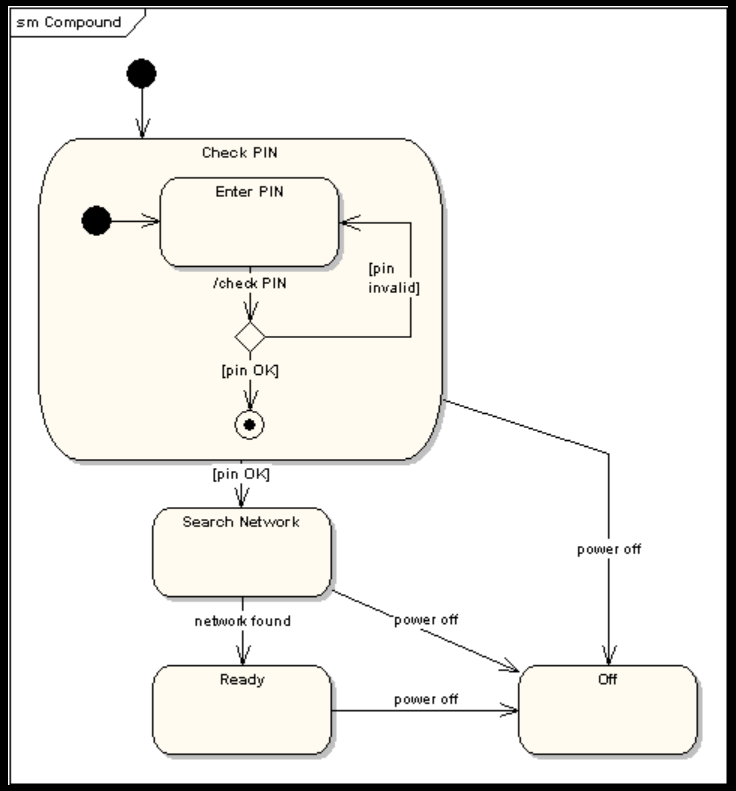
# State Machine Diagrams-Concurrent

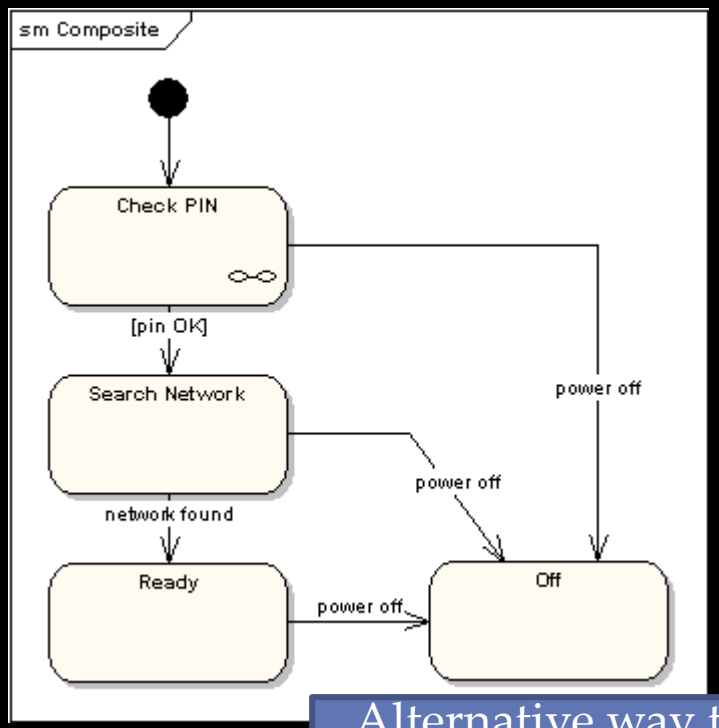
- ***Concurrent Regions*** - A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently.



# State Machine Diagrams

- ***Compound States*** - A state machine diagram may include sub-machine diagrams, as in the example below.



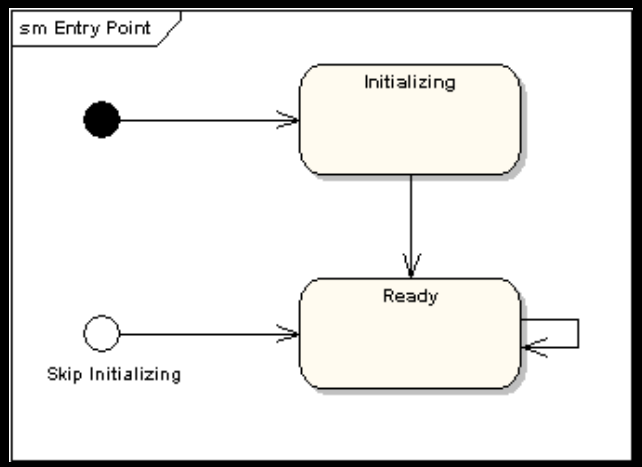



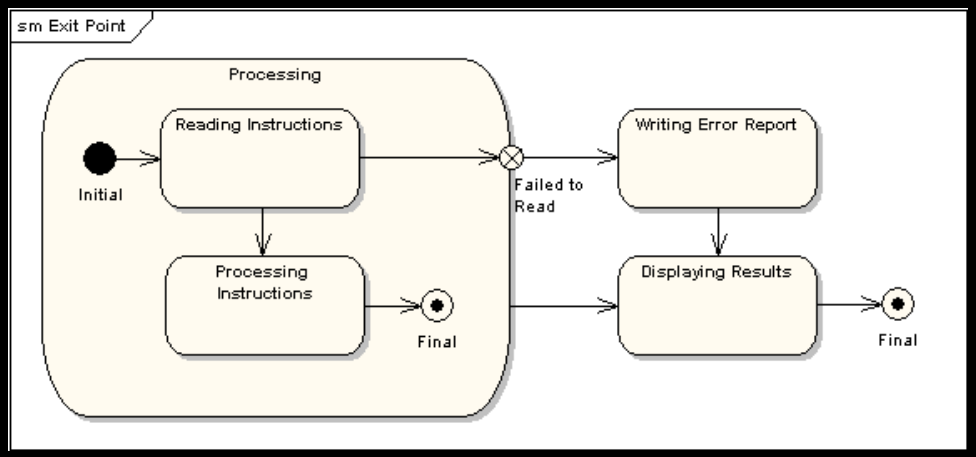

Alternative way to show the same information

**•The ∞ symbol indicates that details of the Check PIN sub-machine are shown in a separate diagram.**

# State Machine Diagrams

- ***Entry Point*** - Sometimes you won't want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.

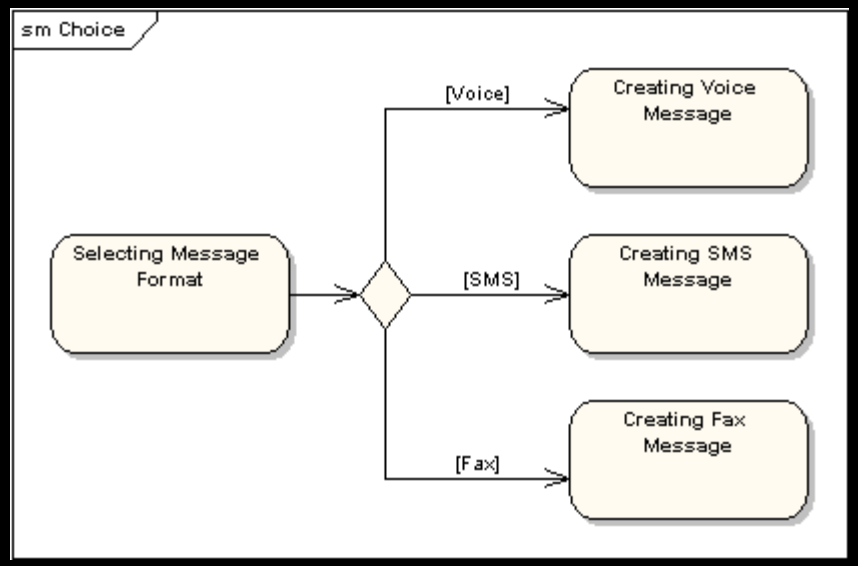



- ***Exit Point*** - In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.

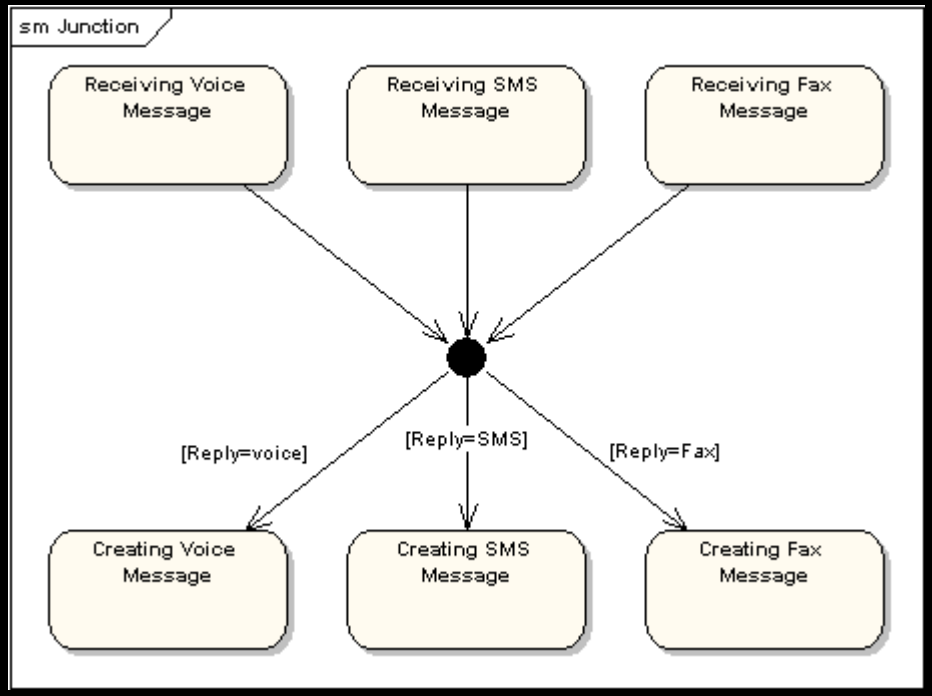
# State Machine Diagrams

- ***Choice Pseudo-State*** - A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



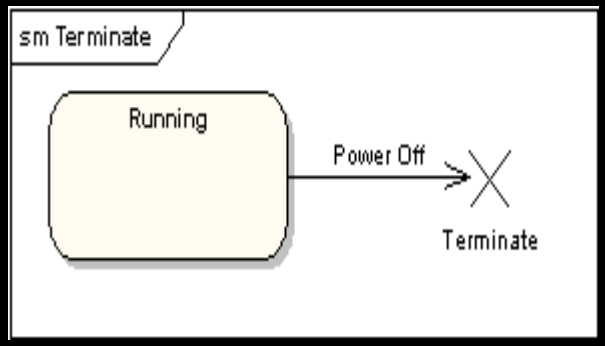
# State Machine Diagrams

- ***Junction Pseudo-State*** - Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free.



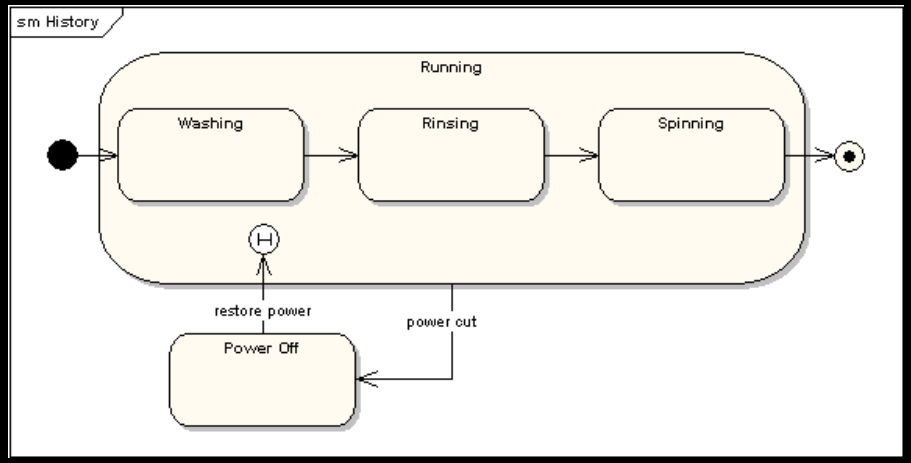
# State Machine Diagrams

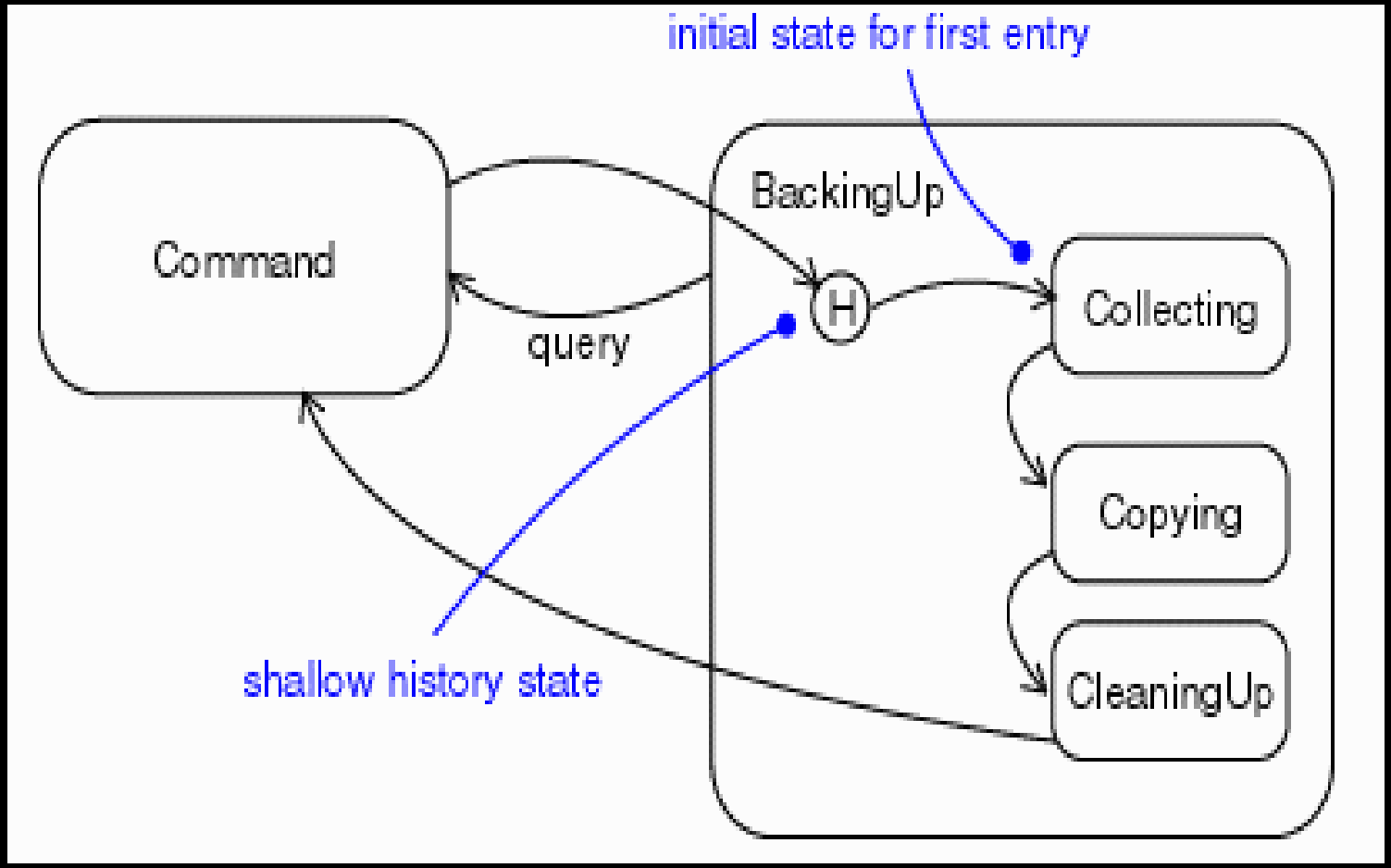
- ***Terminate Pseudo-State*** - Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.



# State Machine Diagrams

- ***History States*** - A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.




initial state for first entry
Command
BackingUp
H
query
Collecting
Copying
CleaningUp
shallow history state

# State Model Pitfalls

- Too detailed
- Badly drawn/organized
  - need for state model tool
- Not really a state model, but a flow chart
  - states are actions and transitions are conditions on existing values, like a control flow diagram
- Transitions should be made on the occurrence of events

# State Transition Diagram Activity

- **Case Study – State Chart Diagram**

1. There is a control button available for the user of the oven. If the oven is idle with the door is closed and you push the button, the oven will start cooking (that is, energize the power-tube for one minute).
2. Also while cooking, another button is pushed, it will cause the oven to cook for extended time.
3. Pushing the button when the door is open has no effect (because it is disabled).
4. Whenever the oven is cooking or the door is open the light in the oven will be on.
5. Opening the door stops the cooking.
6. Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
7. If the oven times-out the light and the power-tube are turned off and then a beeper emits a sound to indicate that the cooking is finished.

# Activity diagram vs. State chart diagram

- Both activity and state chart diagrams model the dynamic behavior of the system. Activity diagram is essentially a flowchart showing flow of control from activity to activity. A state chart diagram shows a state machine emphasizing the flow of control from state to state.
- An activity diagram is a special case of a state chart diagram in which all or most of the states are activity states and all or most of the transitions are triggered by completion of activities in the source state (An activity is an ongoing non-atomic execution within a state machine).
- Activity diagrams may stand alone to visualize, specify, and document the dynamics of a society of objects or they may be used to model the flow of control of an operation. State chart diagrams may be attached to classes, use cases, or entire systems in order to visualize, specify, and document the dynamics of an individual object.


# That is all