CS445/ECE 451/CS645

Software Requirements Specifications & Analysis

Behavioural Modelling

Glossary

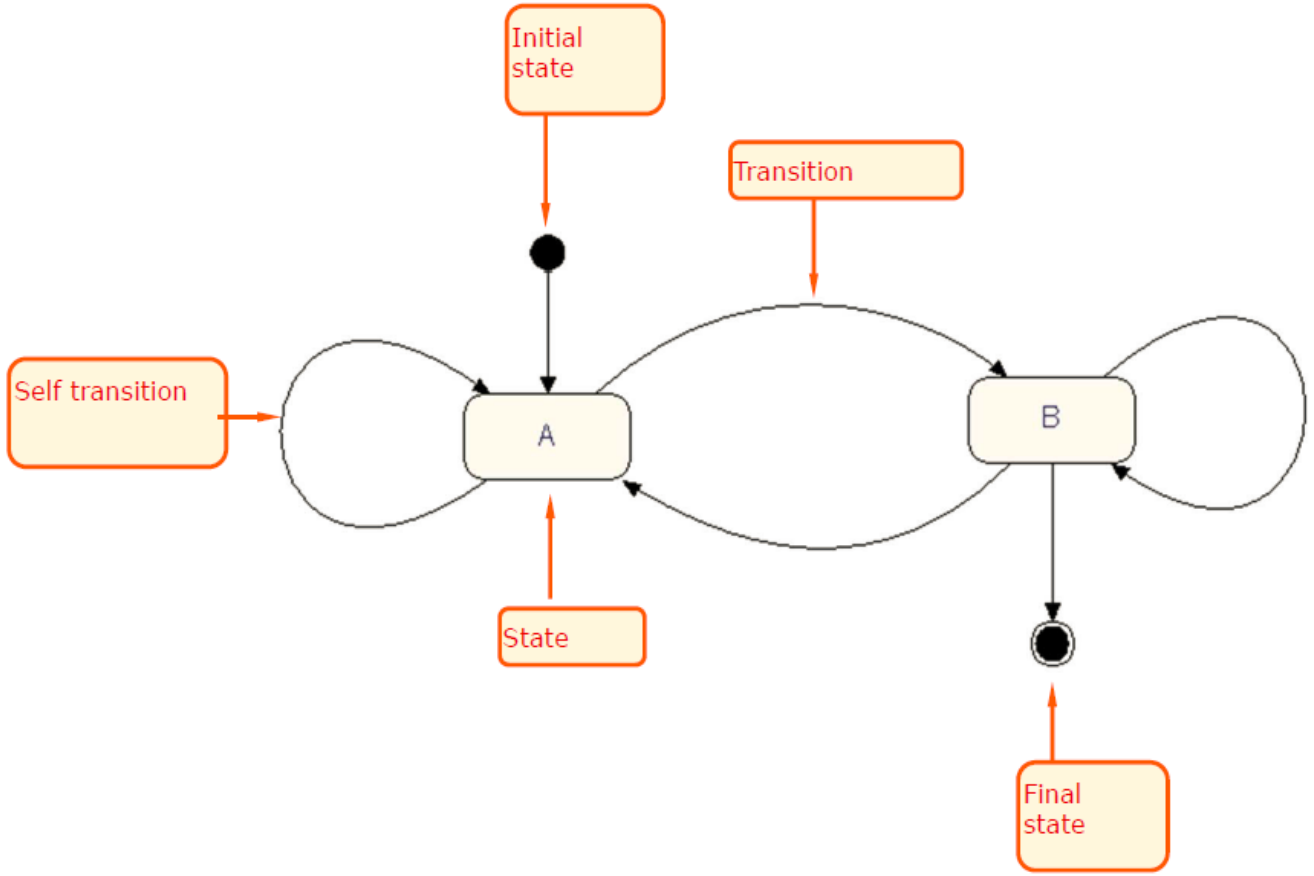
- System behavior: is how a system acts and reacts.
- Behavior model: a view of a system that emphasizes the behavior of the system as a whole (as it appears to outside users).
- State-driven behavior: means that the object's behavior can be divided into disjoint sets.

UML state diagrams

- **State diagram**: Shows data and behavior of a single object throughout its lifetime.
  - set of states (including an initial start state)
  - transitions between states
  - entire diagram is drawn from that object's perspective
- What objects are best used with state diagrams?
  - large, complex objects with a long lifespan
  - domain ("model") objects
  - not useful to do state diagrams for every class in the system!
- Commonly used in design to describe object's behavior as a guide to implementation
- Used in RE to model interface specs (e.g. UI)
- Specify each object's contribution to all scenarios of all use cases.

UML state diagrams

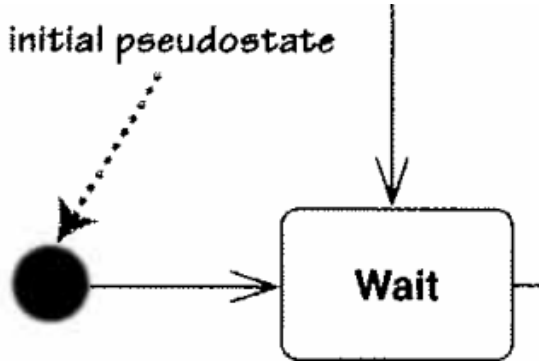
- Represented by Finite State Machine (FSM)
  - FSA – Finite State Automaton- is another term for FSM



States

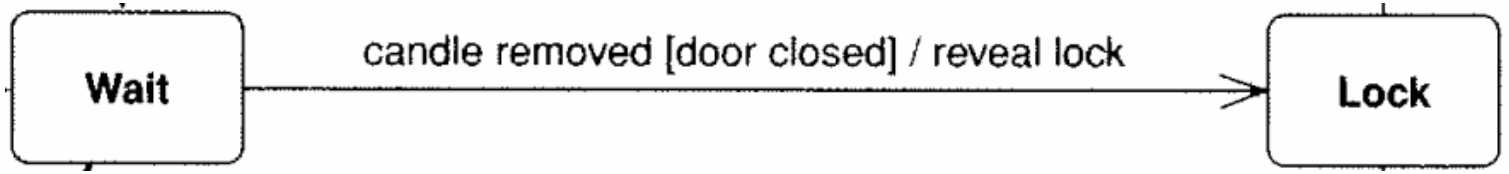
- **State**: conceptual description of the data in the object
  - represented by object's field values

- Entire diagram is drawn from the central object's perspective
  - only include states / concepts that this object can see and influence
  - don't include every possible value for the fields; only ones that are conceptually different

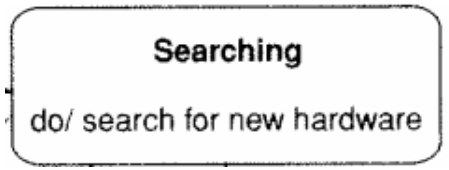


Transitions

- **Transition**: movement from one state to another



- *Event* [*condition*] */ action*
  - event: triggers (potential) state change
  - condition: boolean condition that must be true
  - action: any behavior executed during transition *(optional)*



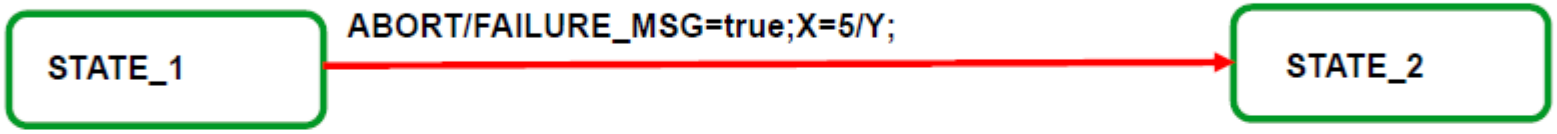
- Transitions must be mutually exclusive (deterministic)
  - must be clear what transition to take for an event
  - most transitions are instantaneous (existing or measured at a particular instant.) , except "do" activities

Note:

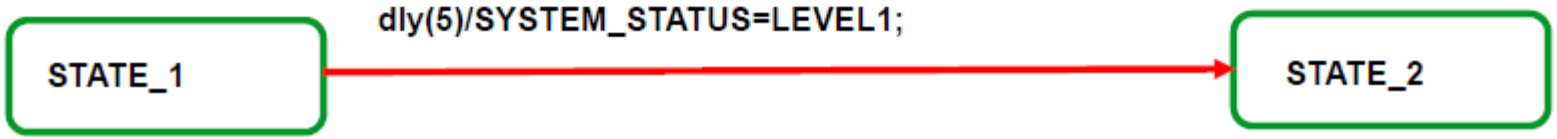
- Event is a noteworthy or significant occurrence in the environment.
  - input message from the env. (login request)
  - change in the env. (coin inserted, elevator button pressed)
  - passage of time
  - multiple events on a transition label are alternative triggers
- Condition is a Boolean expression:
  - over domain model phenomena
  - over state-machine variables
- Action is the system's response to an event, it is non-interruptible.
  - output message
  - change to env. phen. (Turnstile.locked := true. AddLoan(m:LibraryMember, p:Publication, today:Date)
  - multiple actions are separated by ";" and execute sequentially

Examples

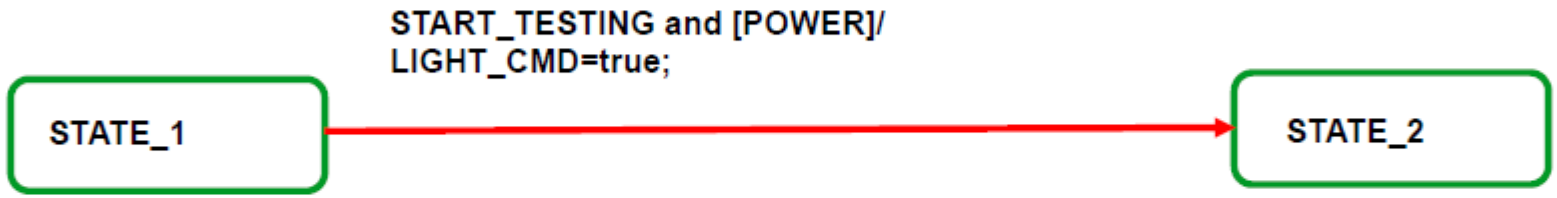
1. The variable *failure_msg* is set to true and *x* is set to 5/y when the event abort occurs.



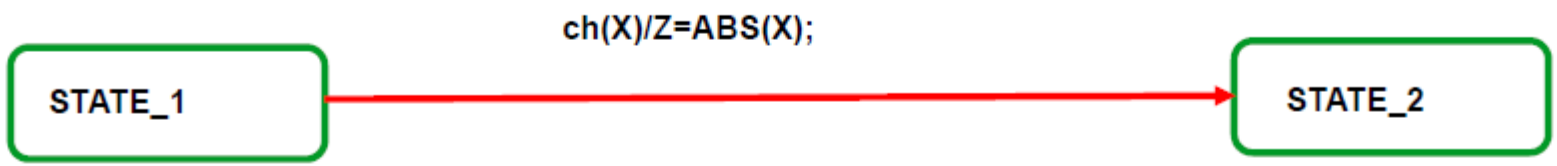
2. Five time units after STATE_1 is entered, *system_status* is set to *level1*.



3. If the event start_testing occurs and power is true, the variable light_cmd is set to true.



4. When the value of X changes, then set Z to the absolute value of X.



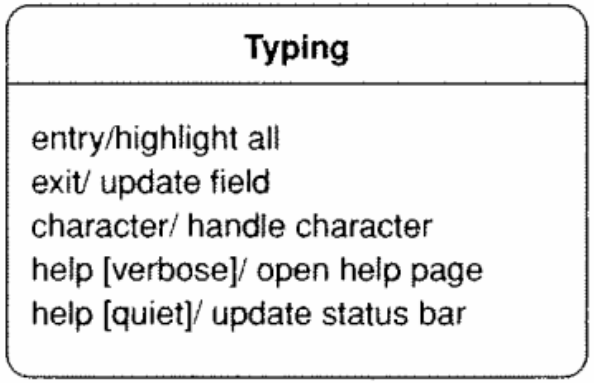
How are transactions handled?

- If an object is in a state S that responds to an event evX, then it acts upon that event.
  - It transitions to the specified state, if the event triggers a transition and the condition (if any) on that transition evaluates to TRUE.
  - It executes any actions associated with that transition.
- Events are quietly discarded if:
  - A transition is triggered, but the transition's condition evaluates to FALSE.
  - The event does not explicitly trigger a transition or reaction.

Internal activities

Interruptable

- **Internal activity**: actions that the central object takes on itself
  - sometimes drawn as self-transitions (events that stay in same state)



- entry/exit activities
  - reasons to start/stop being in that state

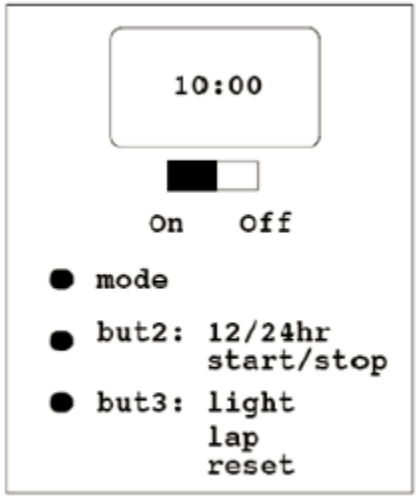
- Take time, interruptible, may require computation.

UML terminology

- Actions are known as “regular activities” **Non-interruptable**
- Activities are known as “do-activities” **Interruptable**

Example: Stopwatch

1. Watch should be able to toggle between time and stopwatch mode
2. Watch should be able to toggle between 12h and 24h display
3. User should be able to start/stop timer; beep for 0.25 seconds when in stopwatch mode
4. User should be able to turn light on for 3 seconds when watch in time mode
5. Watch should record laptime; display laptime, and turn light on for 3 seconds when watch is in stopwatch mode, the timer is running and displayed.
6. Watch should reset timer, and turn light on for 3 seconds when watch is in stopwatch mode, the timer stopped and displayed.
7. Watch should display timer, and turn light on for 3 seconds when watch is in stopwatch mode, laptime is displayed.



- Stopwatch starts in off state
- When “off”,
  - Display is turned off
  - Battery continues to power an internal “wall clock”
  - Last value of timer is kept in memory, but timer is turned off (if it was running)
- When powered on,
  - Display is turned on
  - The timer is off (but not reset)
  - Default initial display is 12 hour wall clock time
- Hardware has built-in timer mechanism
  - Can start/stop/reset/get value
- Starting/stopping the timer should cause an audible beep for 0.25s
  - Hw supports “beep”, but is not tied to start/stop by default

State diagram for Stopwatch

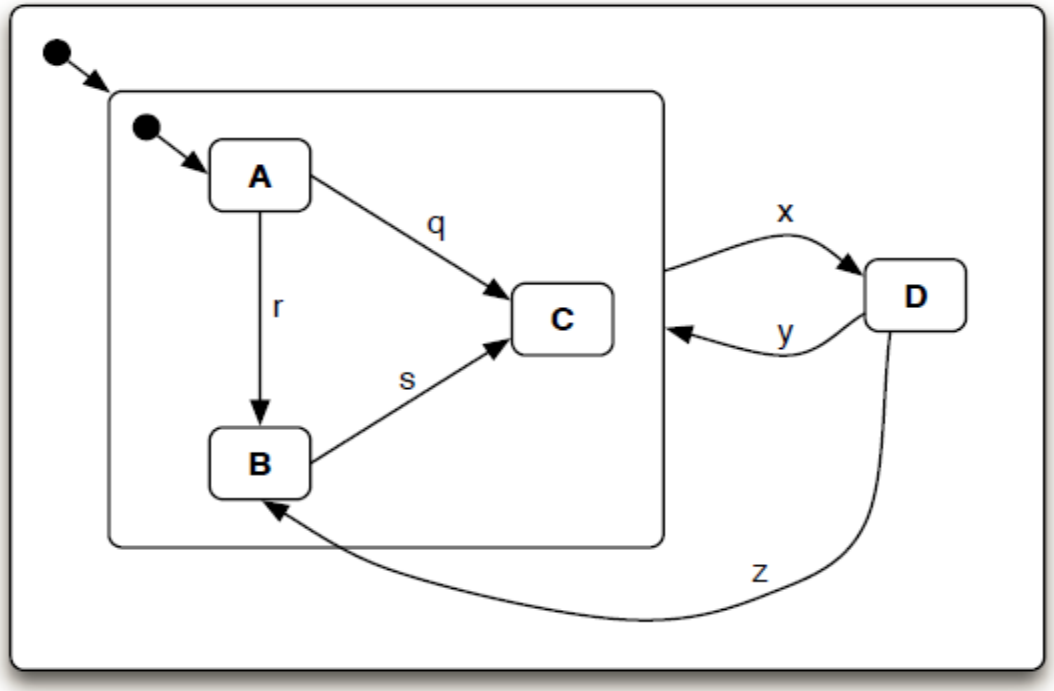
Composite state

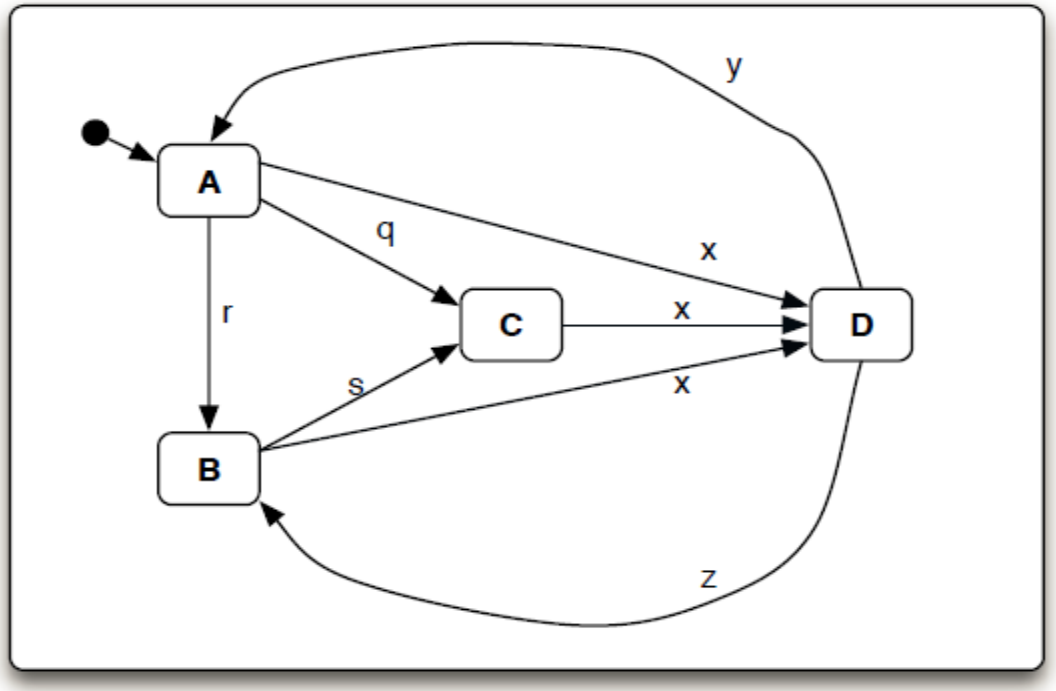
- Combines states and transitions that work together towards a common goal. There are two kinds:
  1. Hierarchical (simple / or-states)
  2. Concurrent (orthogonal / and-states)

Hierarchical states

Hierarchy is used to cluster states that have some similar behaviors / exiting transitions.

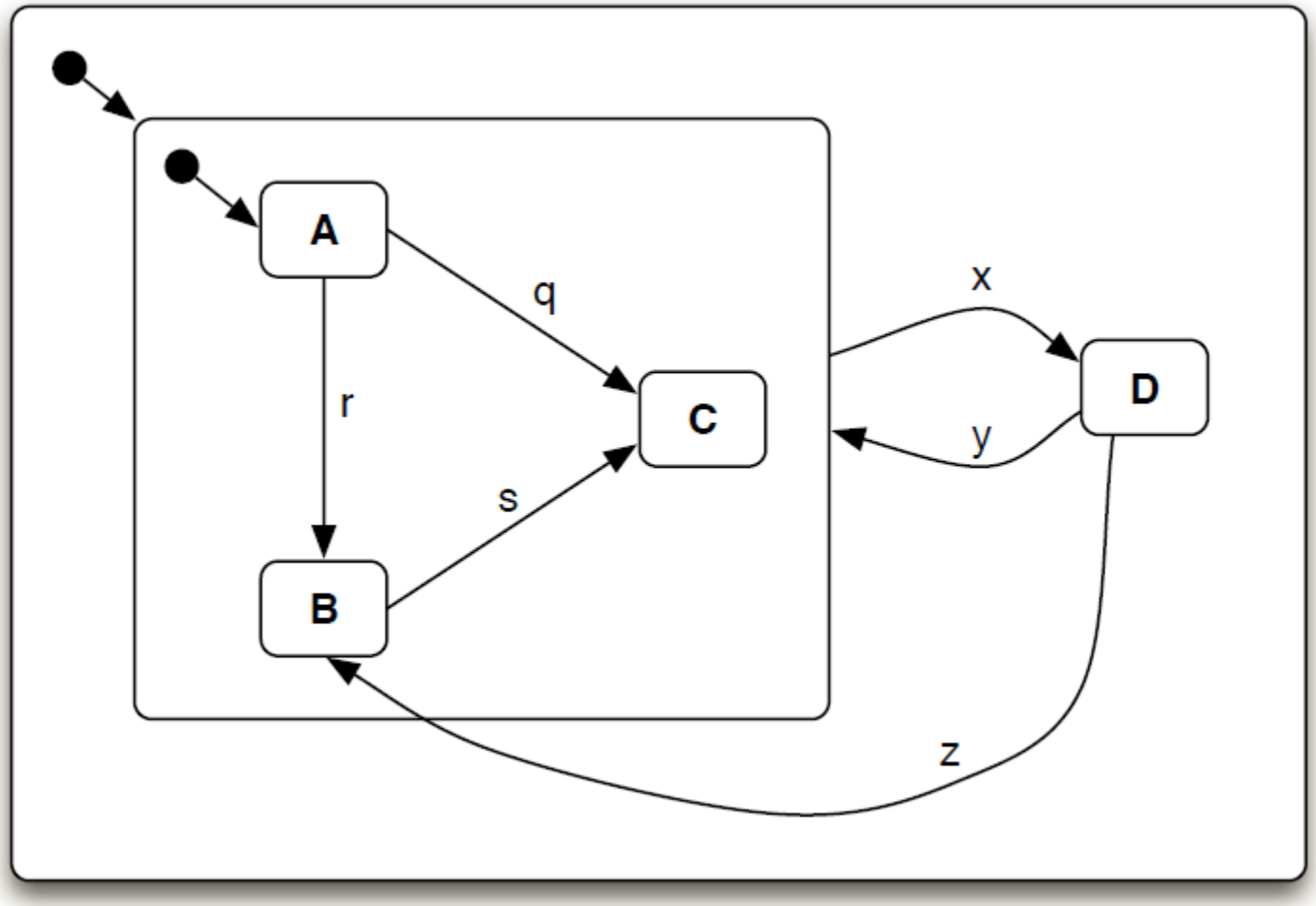
- One transition leaving a superstate represents a transition from each of the superstate's descendent states.





Test your self

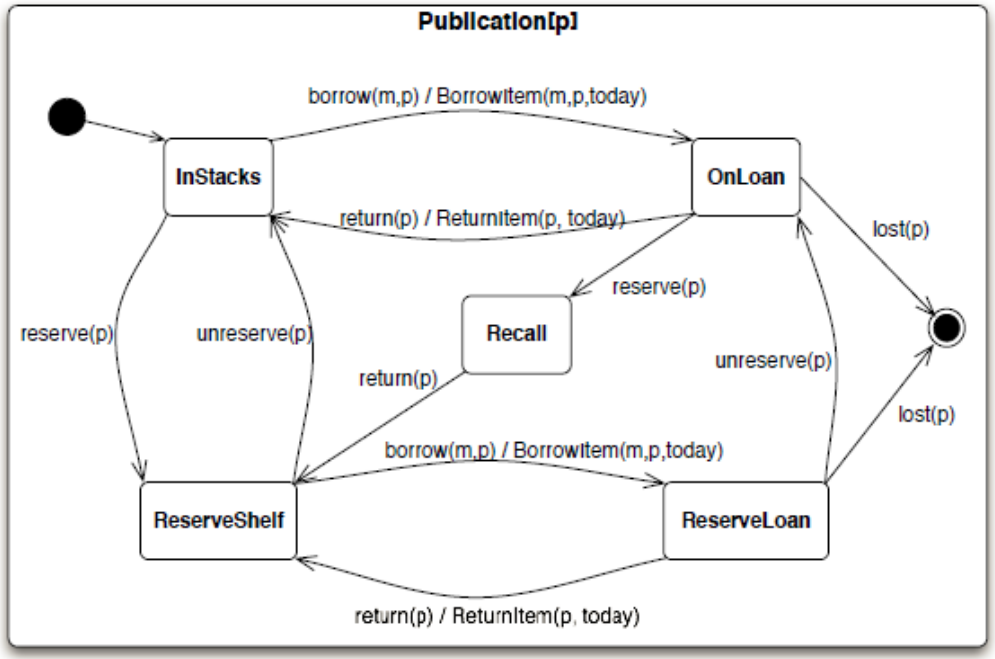
1. What happens if event z occurs when in state D?
2. What happens if event y occurs when in state D?
3. Can the execution ever leave state C?

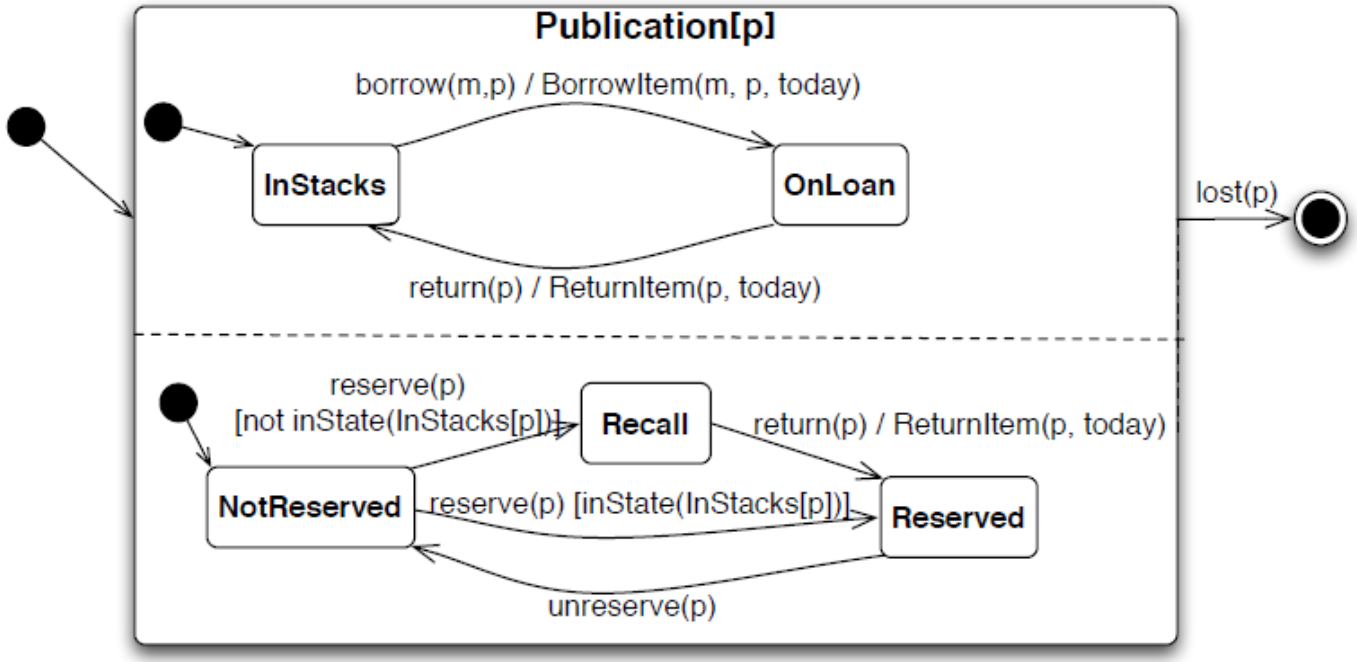


Concurrent regions

Some systems have orthogonal behaviors that are best modelled as concurrent state machines

- Regions within a concurrent state execute in parallel.
- Each has its own thread of control.
- Each can “see” and react to events /conditions in the world

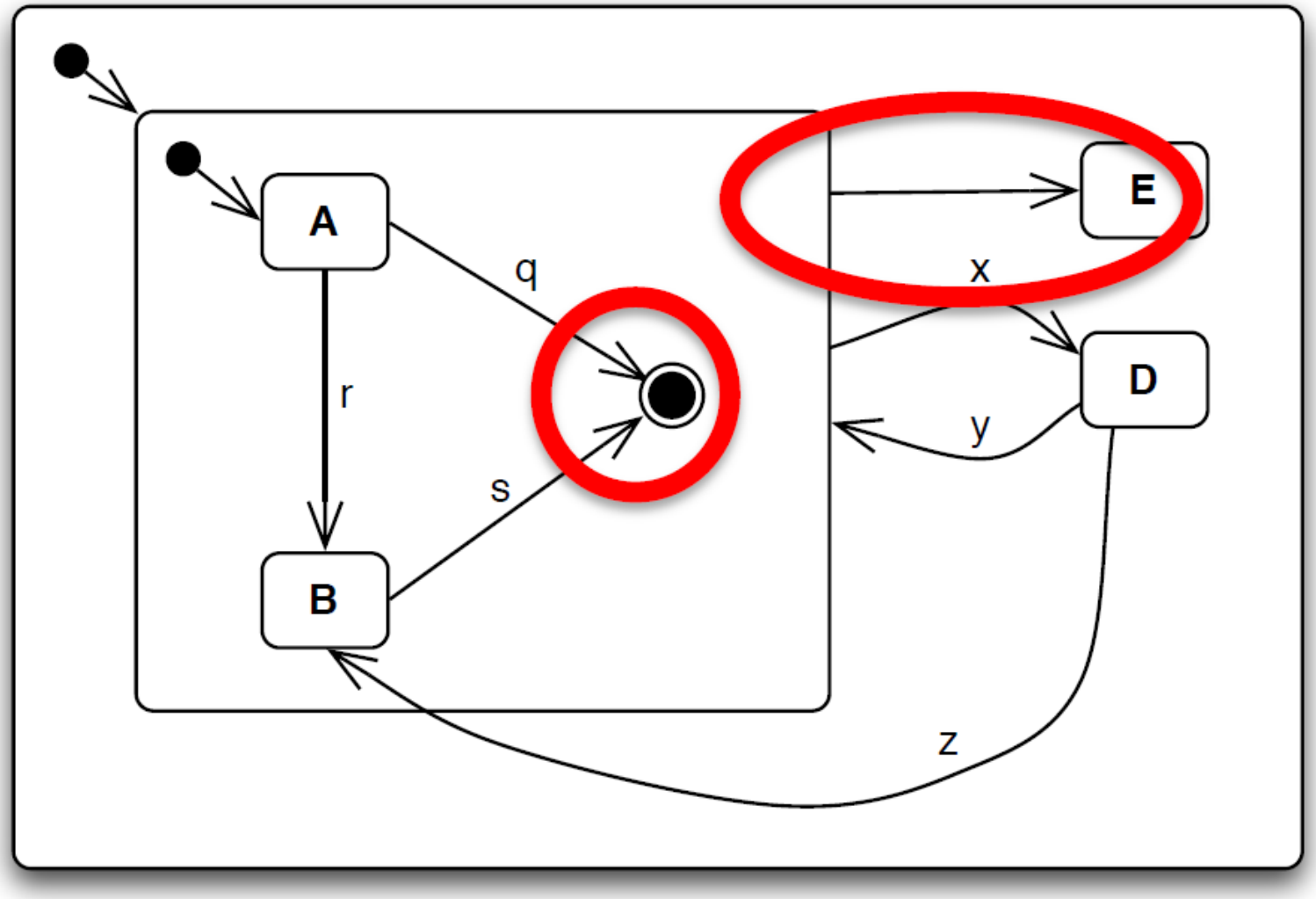

Publication[p]
borrow(m,p) / BorrowItem(m,p,today)
InStacks
OnLoan
return(p) / ReturnItem(p, today)
lost(p)
reserve(p)
Recall
reserve(p)
unreserve(p)
return(p)
unreserve(p)
lost(p)
borrow(m,p) / BorrowItem(m,p,today)
ReserveShelf
ReserveLoan
return(p) / ReturnItem(p, today)


Publication[p]
borrow(m,p) / BorrowItem(m, p, today)
InStacks
OnLoan
lost(p)
return(p) / ReturnItem(p, today)
reserve(p)
[not inState(InStacks[p])]
Recall
return(p) / ReturnItem(p, today)
NotReserved
reserve(p) [inState(InStacks[p])]
Reserved
unreserve(p)

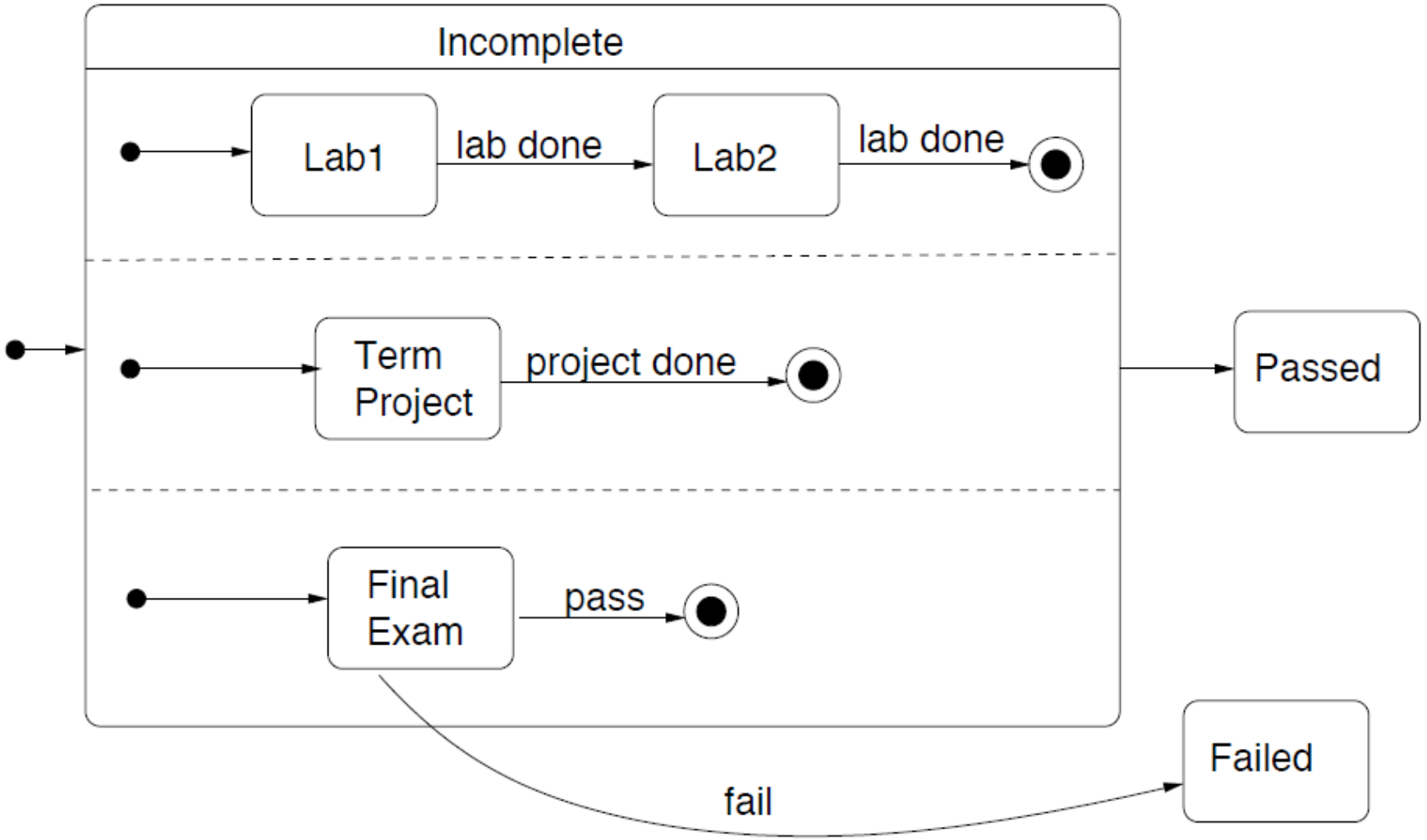
Final State

A transition that has no event or condition in its label is enabled when its

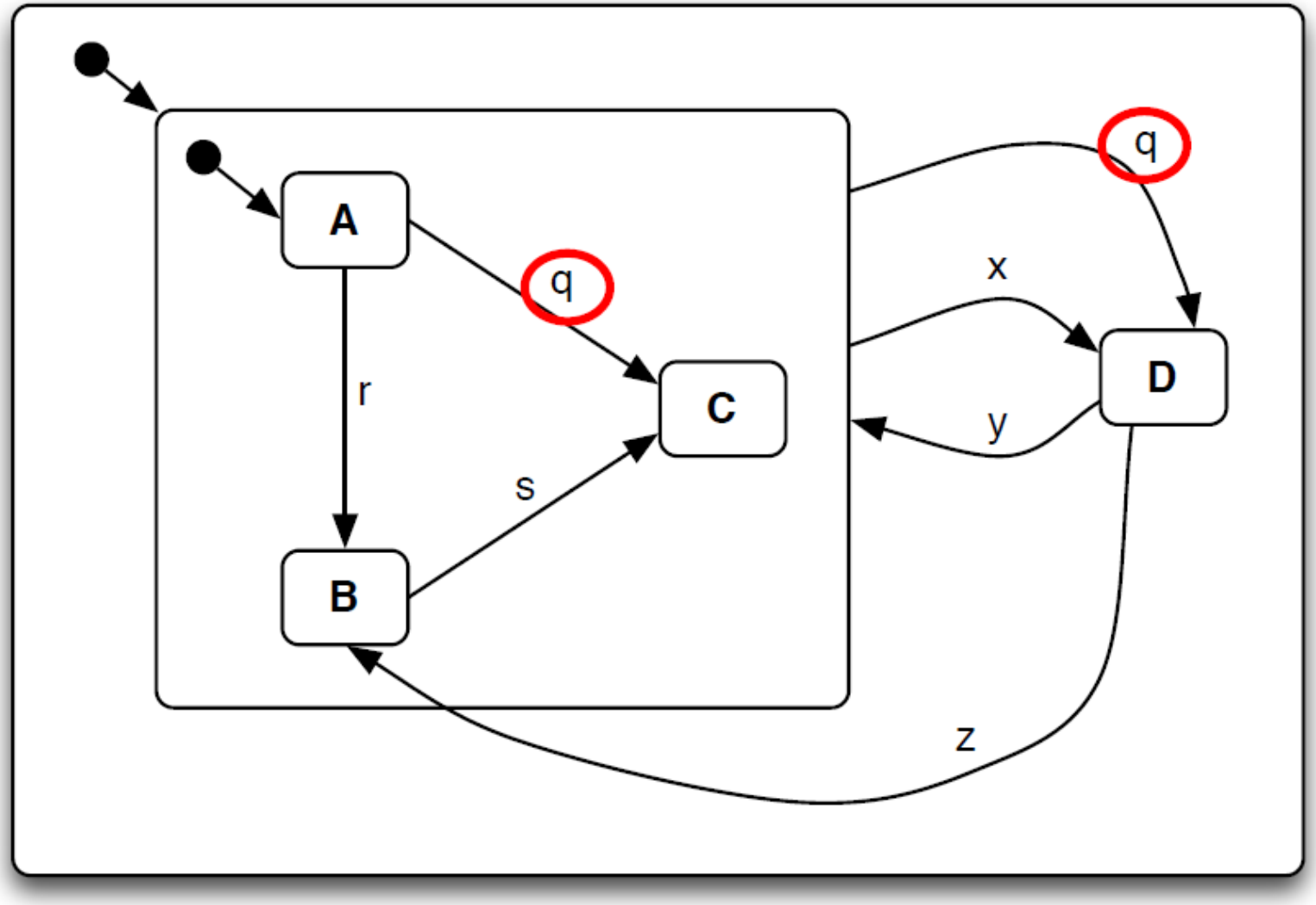
- source state is basic and idle, or
- source superstate entered its final state, or
- source basic state has finished internal activity



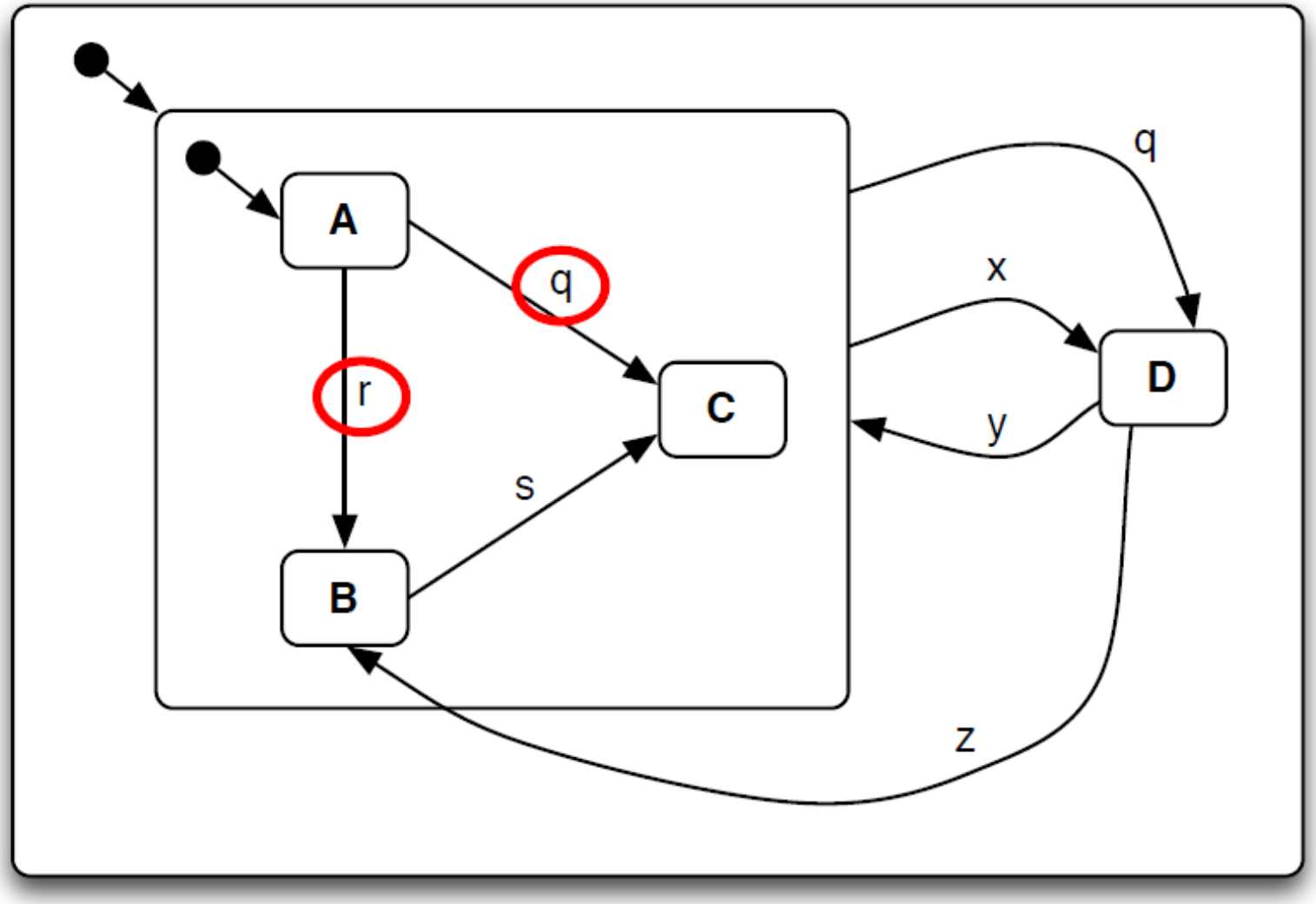
Concurrency and Final States



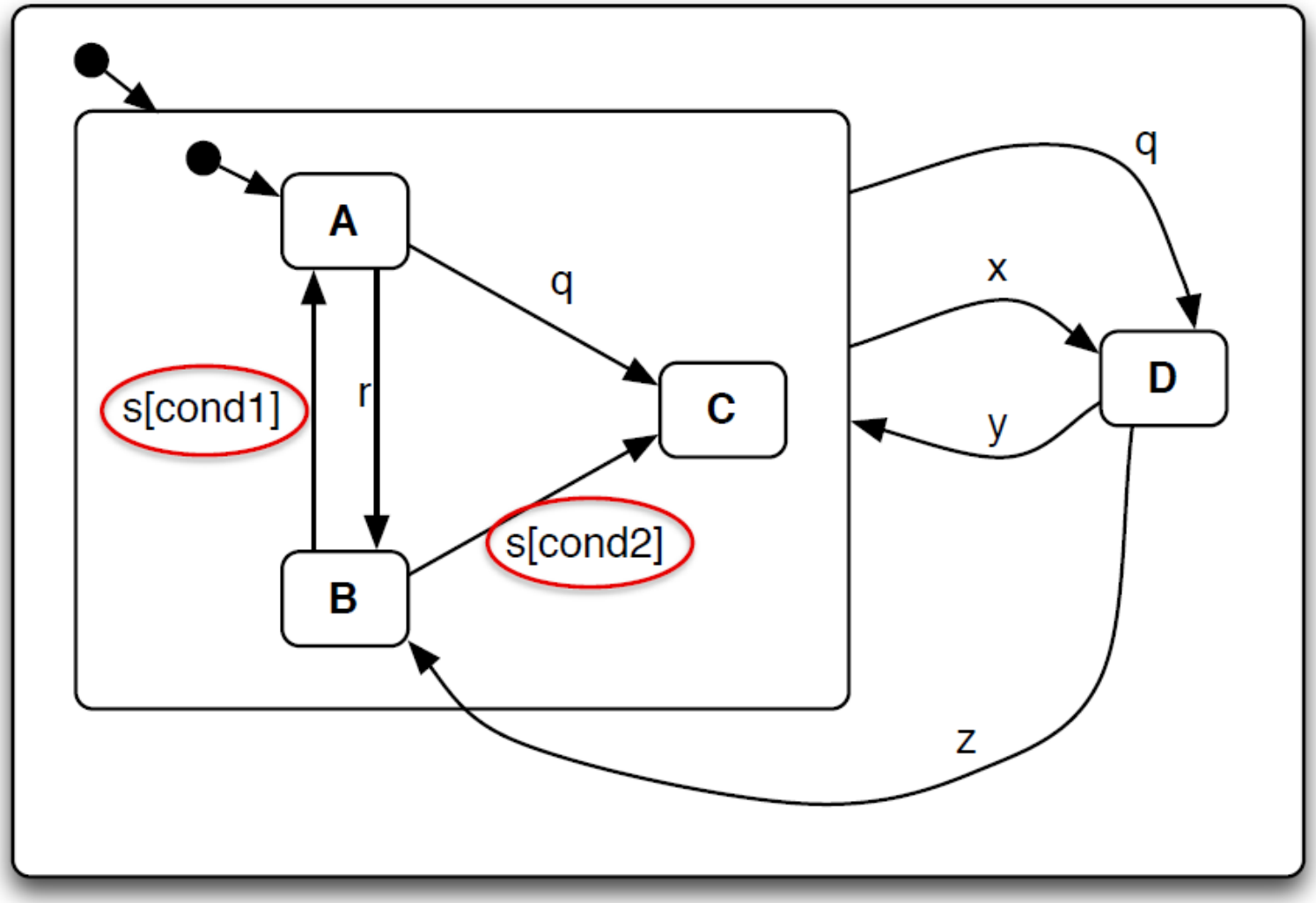
Priority



Priority

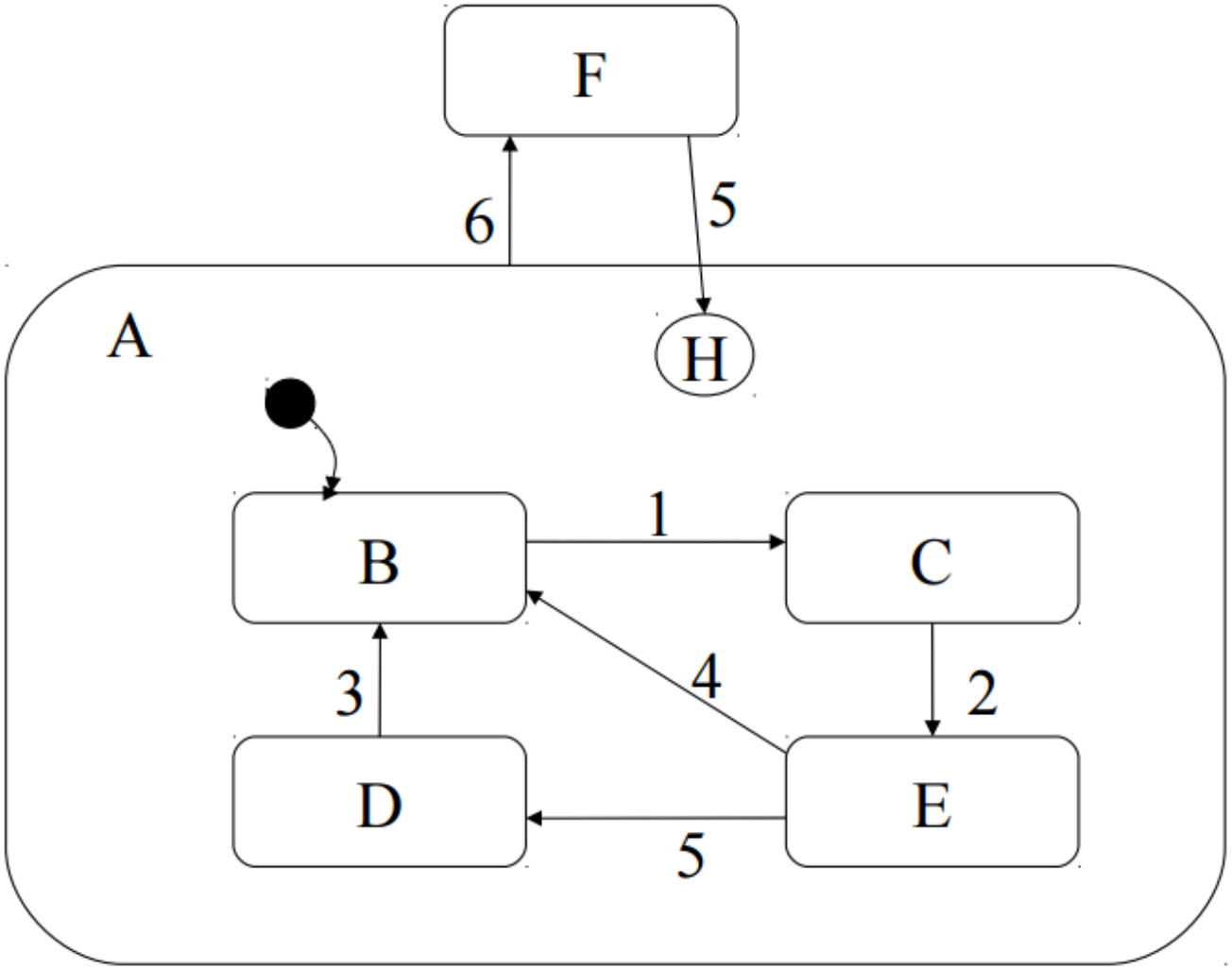


Determinism



History

- Provides a way of entering a group of states based on the system's history in that group.
  - That is, the state entered is the most recently visited state in that group.
  - In the next slide when event 5 occurs and state A is entered the history mechanism is used to determine the next state within A.
    - This is read as 'enter the most recently visited state in the group (B, C, D, E) or enter state B if this is the first visit to the state'.

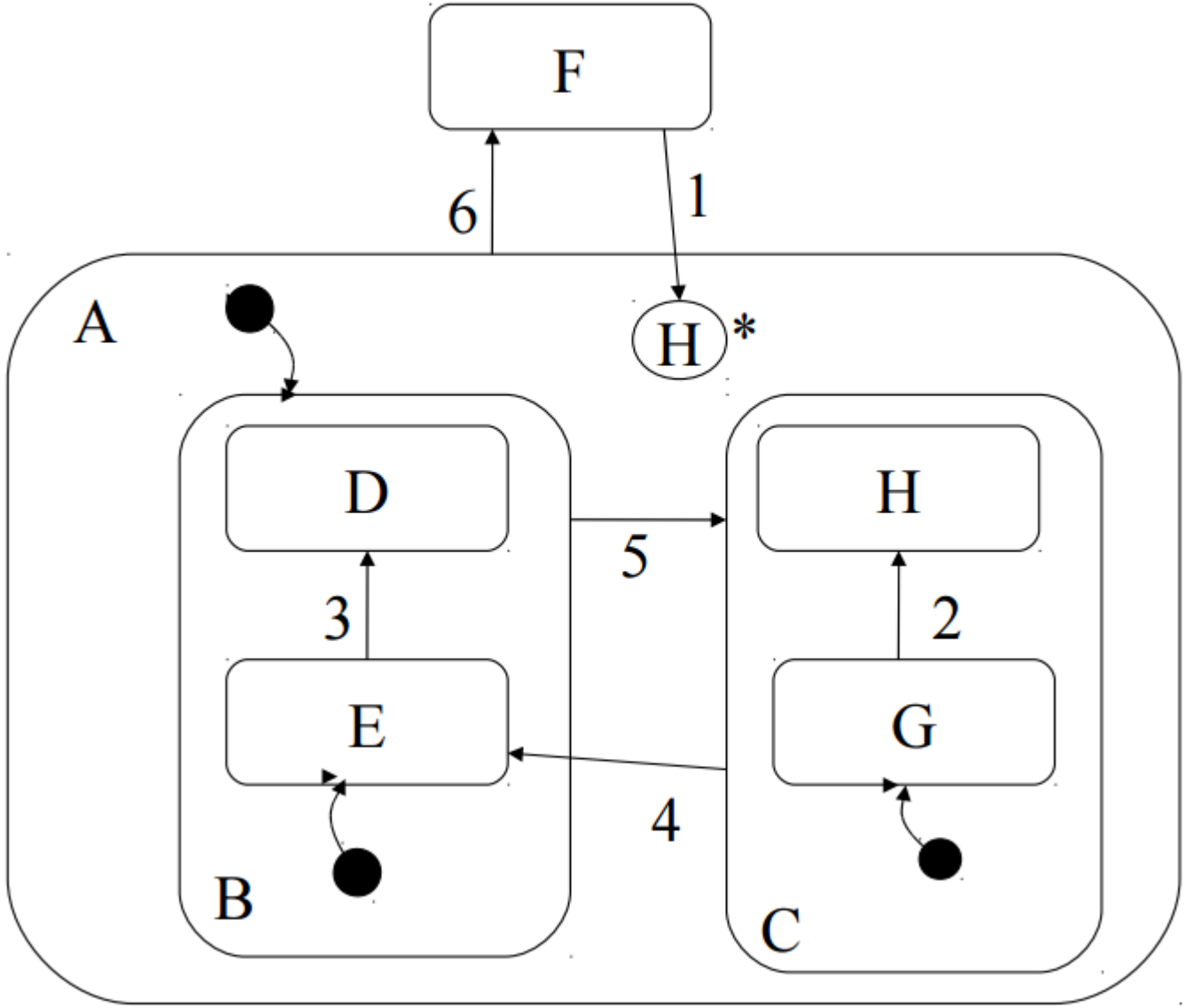

F
6
5
A
H
B
1
C
3
4
2
D
E
5

History mechanism usage

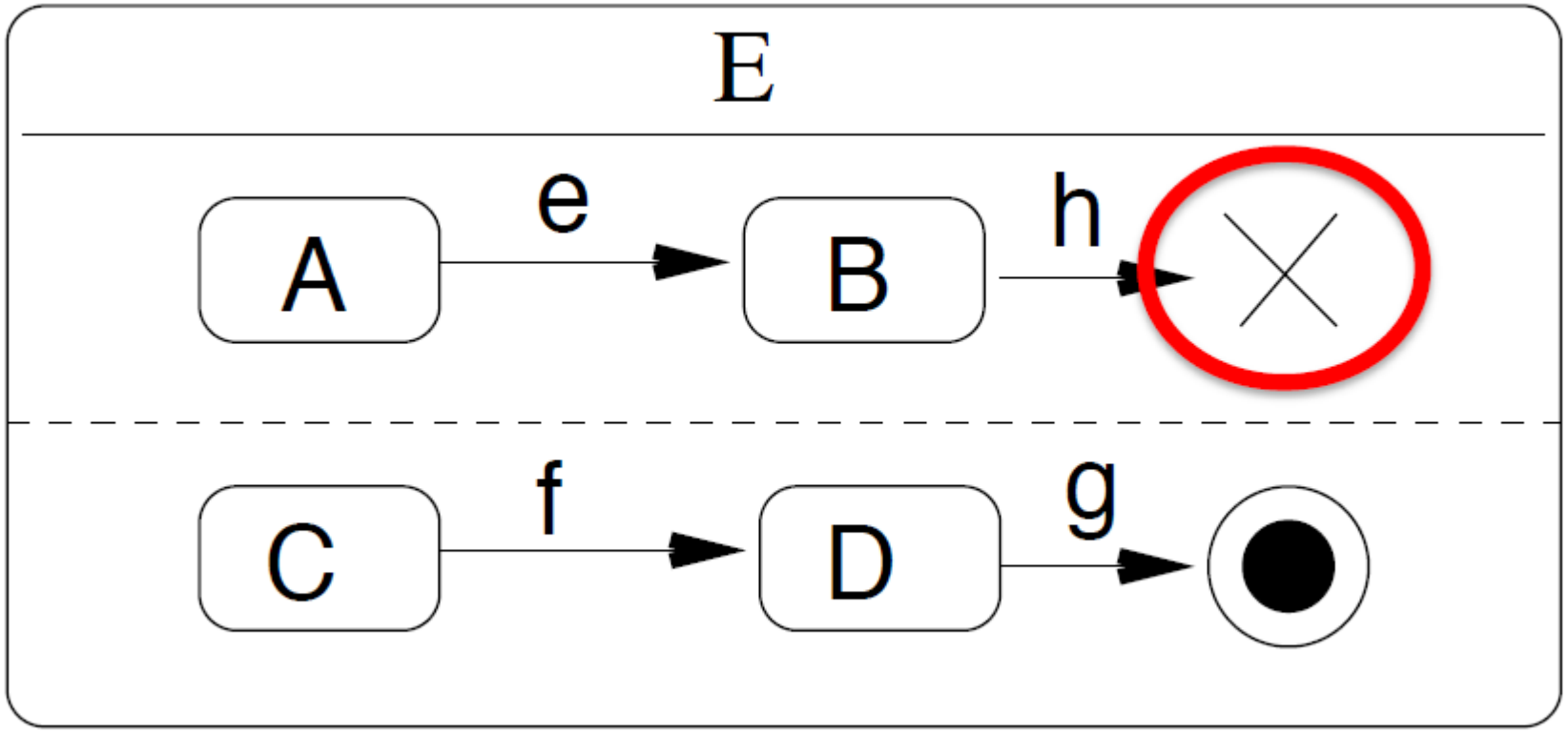
- The history of a system overrides the default start state.
- A default start state must be specified for a group that uses the history mechanism for when the group is entered for the first time.
- The history of a system is applied only to the level in the hierarchy in which it appears.
- To apply the history mechanism at a lower level in the state hierarchy it is necessary to use a history symbol in the lower levels.

Deep history

- An asterisk can be attached to the history symbol to indicate that the history of the system should be applied all the way down to the lowest level in the state hierarchy.



Termination



Time event

A time event is the occurrence of a specific date/time or the passage of time.

- Absolute time:
  - at (12:12 pm, 12 Dec 2012)
- Relative time:
  - after (10 seconds since exit from state A)
  - after (10 seconds since x)
  - after (20 minutes) // since the transition's source state was entered

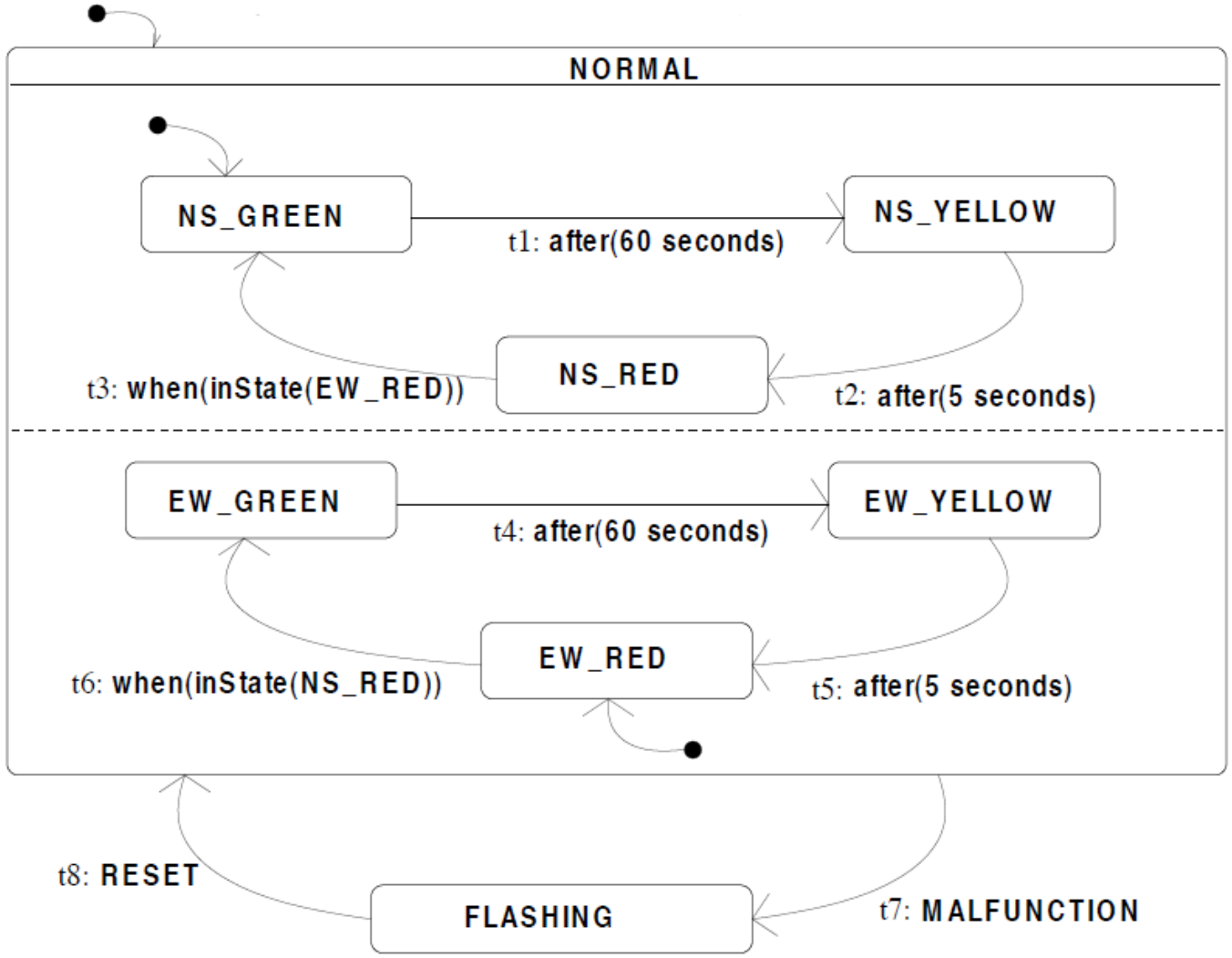
Change events

A change event is the event of a condition becoming true.

- The event “occurs” when the condition changes value from *false* to *true*.
  - when (temperature > 100 degrees)
  - when (on)
- The event does not reoccur unless the value of the condition becomes *false* and then returns to *true*.

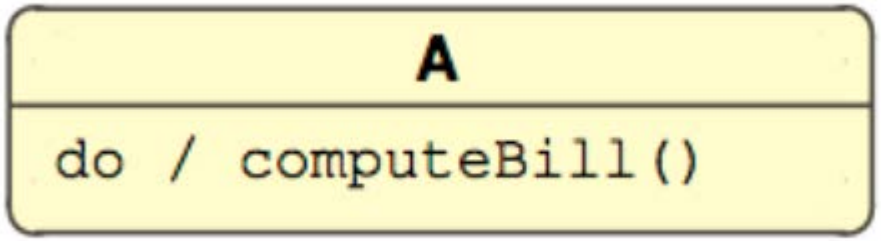
- when(X) vs. [X]

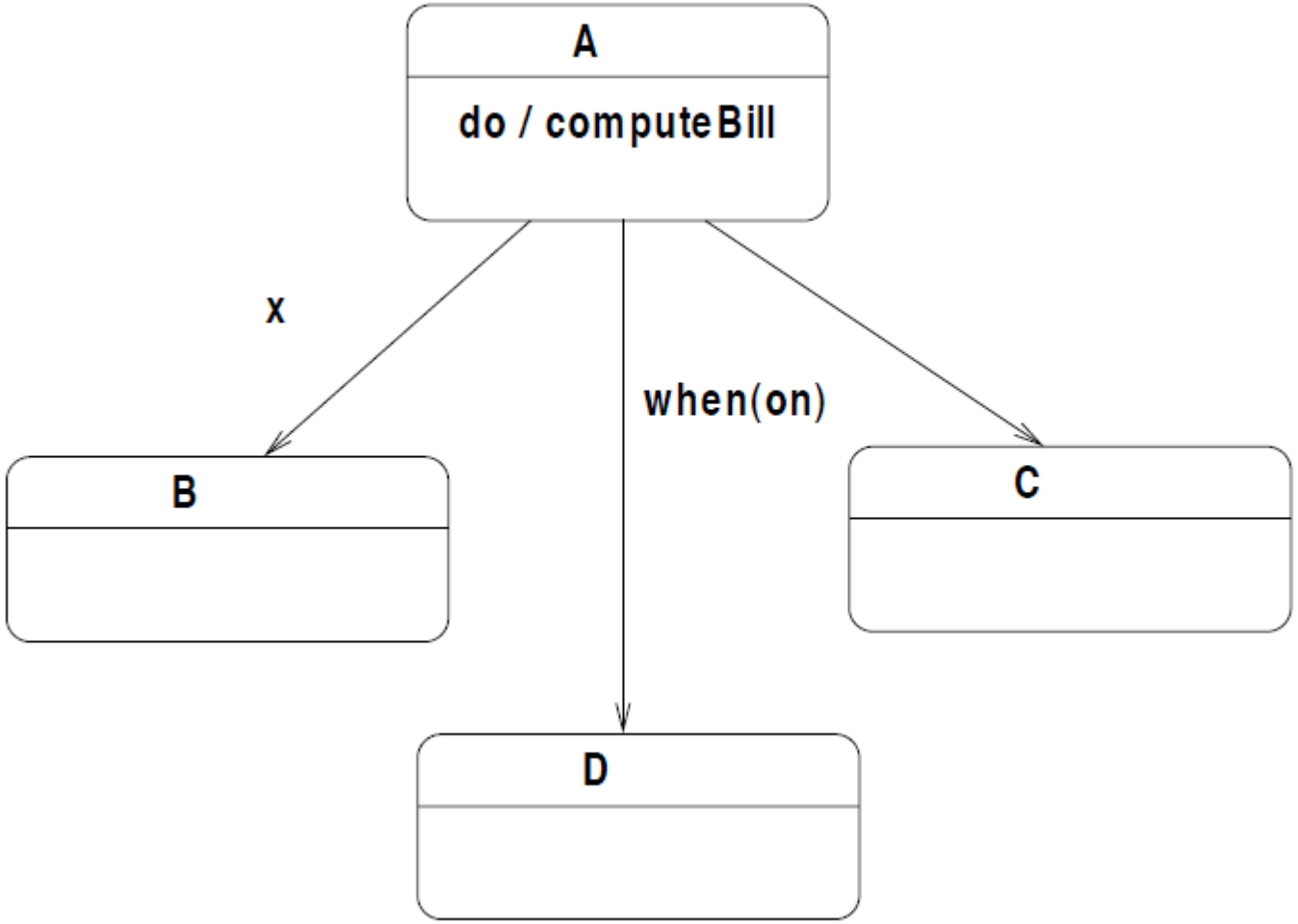
Traffic light example



State Activities

- An activity is computation of the system that takes time, and can be interrupted.
  - c.f., an action, which is uninterruptible
  - An activity may be associated with a state.
  - States with activities are called *activity states*.




A
do / computeBill
x
when(on)
B
C
D

Creating a Behavior Model

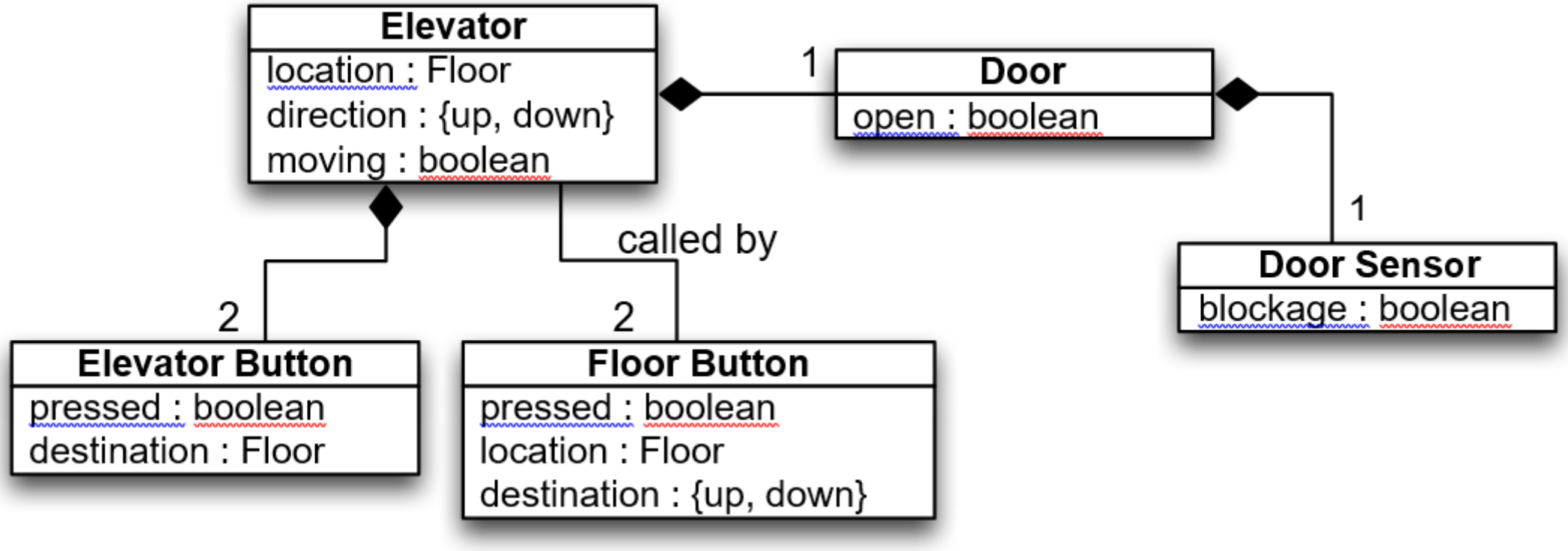
1. Identify input and output events
2. Think of a natural partitioning into states
   - *Activity states* – system performs activity or operation
   - *Idle states* – system waits for input
   - *System modes* – use different states to distinguish between different reactions to an event
3. Consider the behavior of the system for each input at each state.
4. Revise (using hierarchy, concurrency, state events)
   - Use concurrency to separate orthogonal behavior
   - Use hierarchy, and entry/exit actions, to abbreviate common behavior

Behavioral Models Validation

- Avoid inconsistency: multiple transitions that leave the same state under the same event/conditions.
- Ensure completeness: specify a reaction for every possible input at a state.
  - If there are transitions triggered by an event conditioned on some guard, what happens if the guard is false?
- Walkthrough: compare the behaviour of your state diagrams with the use-case scenarios.
  - All paths through the scenarios should be pathed in the state machines.

Improve the State diagram for Stopwatch

Practice exercise:
You are to create a state-machine model for an elevator that has the following domain model



- There is one button on each floor just outside of the elevator that passengers can press to summon the elevator to come pick them up. There two buttons inside of the elevator, one button for each floor that passengers can press to instruct the elevator to deliver them to the associated floor.

- When idle, the elevator sits stationary at the last floor serviced, with its doors closed. When there is a request to service a particular floor (i.e., to deliver a passenger to that floor, or to pick up a passenger from that floor), the elevator moves to that floor (if necessary) and opens its doors.

- A request to service a floor remains outstanding until the elevator reaches the requested floor and opens its doors; at which point, all outstanding requests for that floor are considered serviced. If the elevator is stationary, then requests to service the floor that the elevator is currently on have priority over requests to service the other floor.

- For simplicity, your model should have only one door for the elevator (i.e., don't model the outside doors on each floor). When the elevator door is opened, it remains open for 5 seconds. If the door sensors detect some blockage on closing the doors, the doors will reopen and again try to close after 5 seconds.

The End!