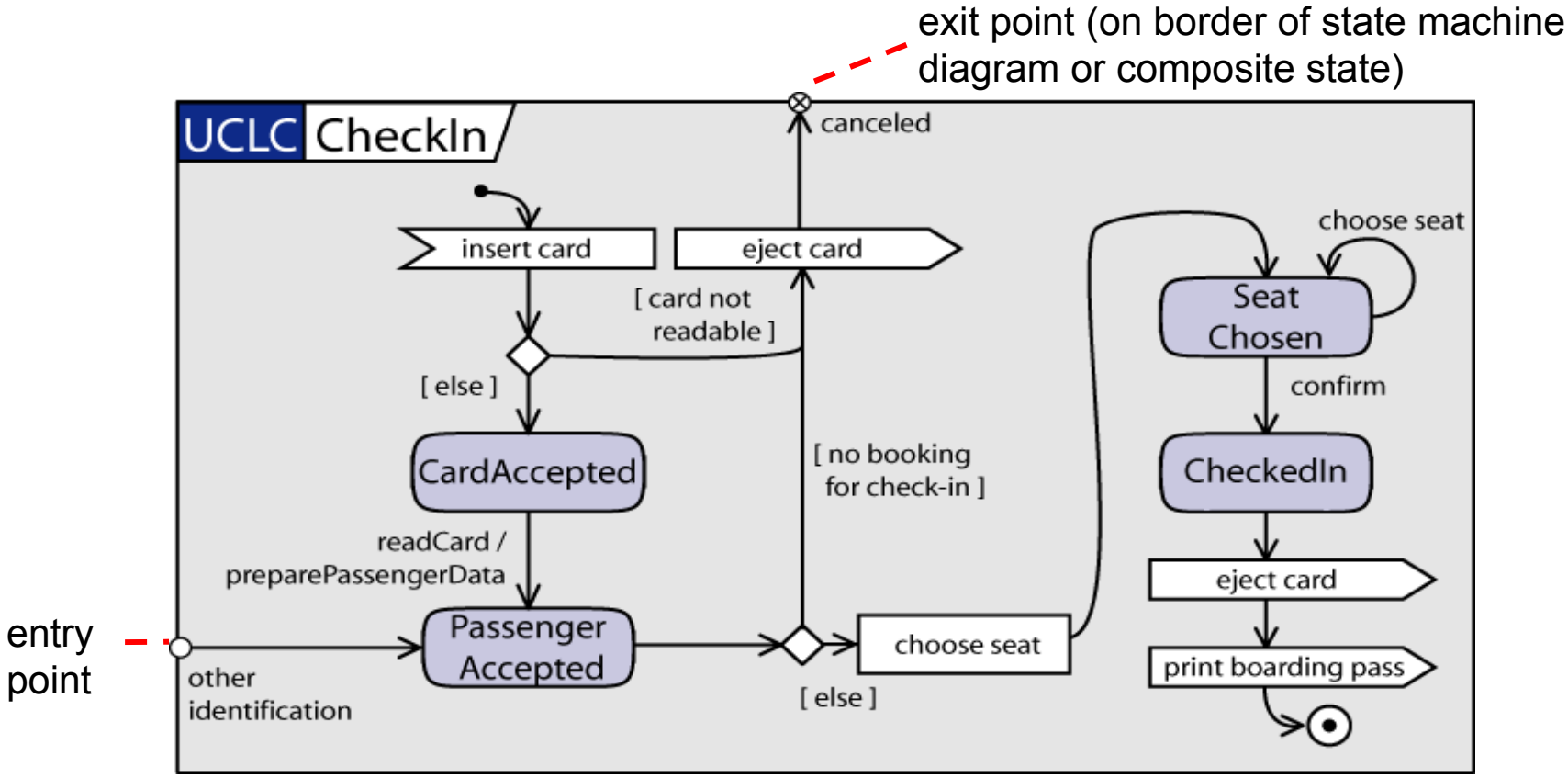
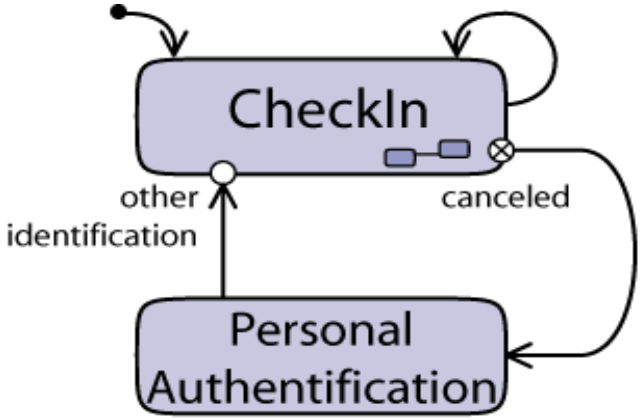
Entry and exit points (1)

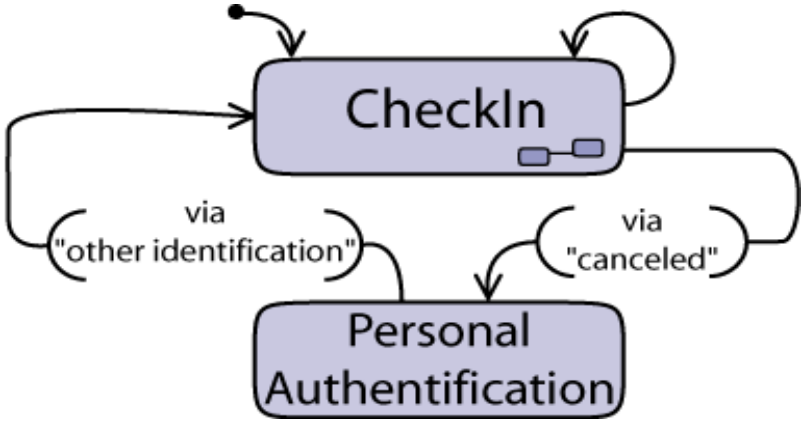
- Entry and exit points (Pseudostates)
  - provide better encapsulation of composite states
  - help avoid “unstructured” transitions



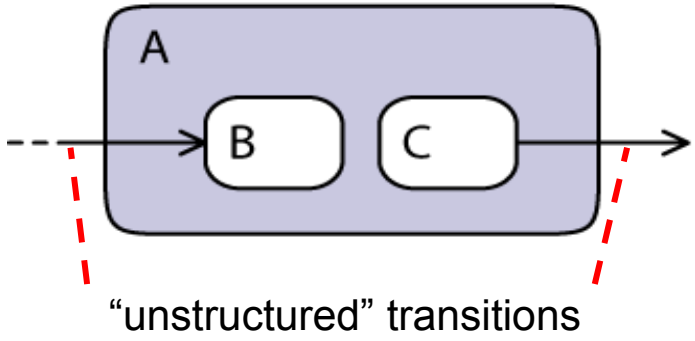
Entry and exit points (2)

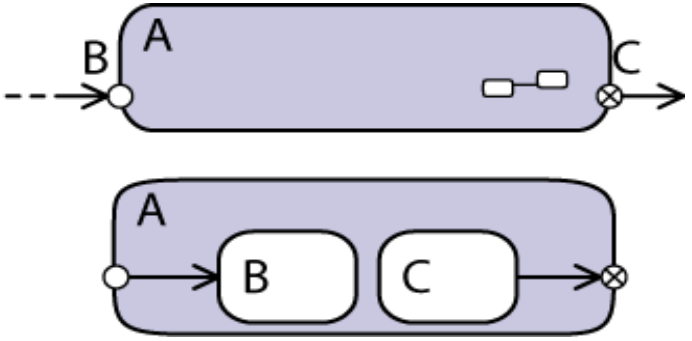
Notational alternatives





Semantically equivalent

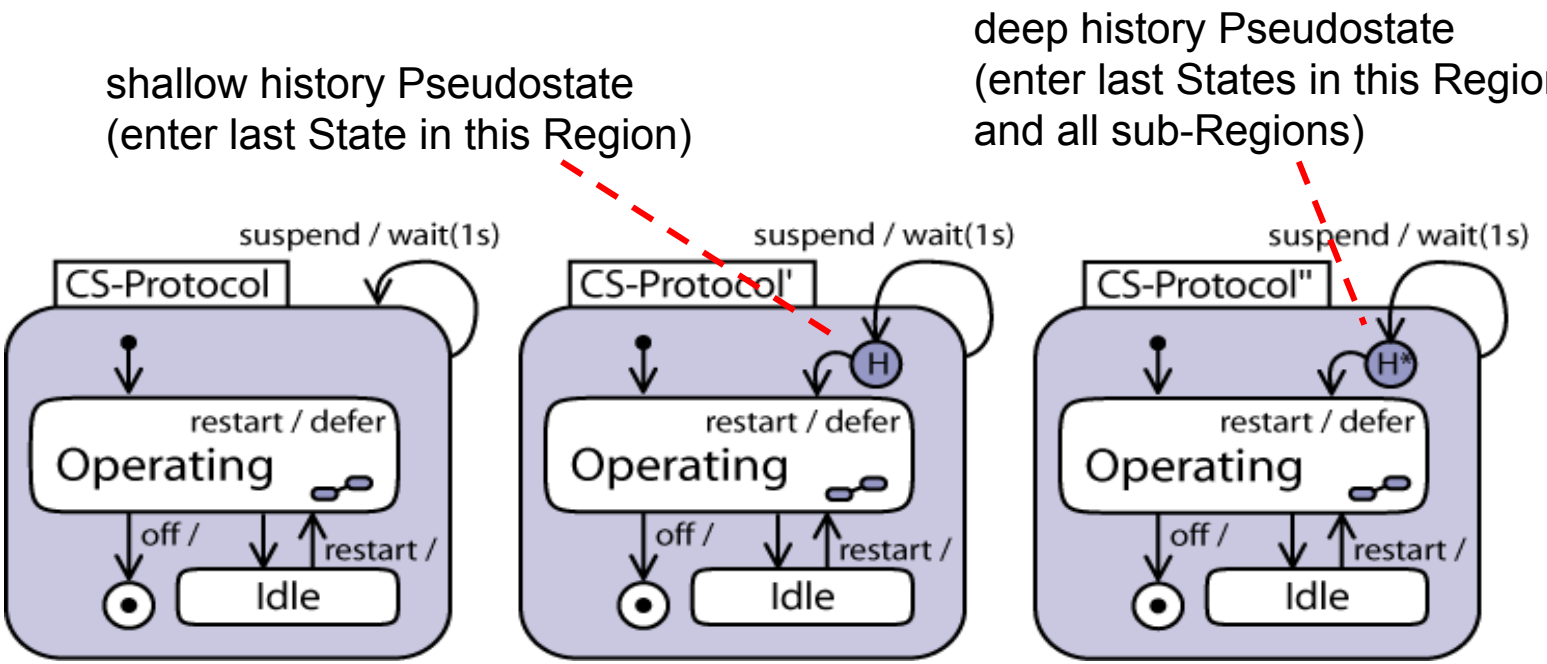




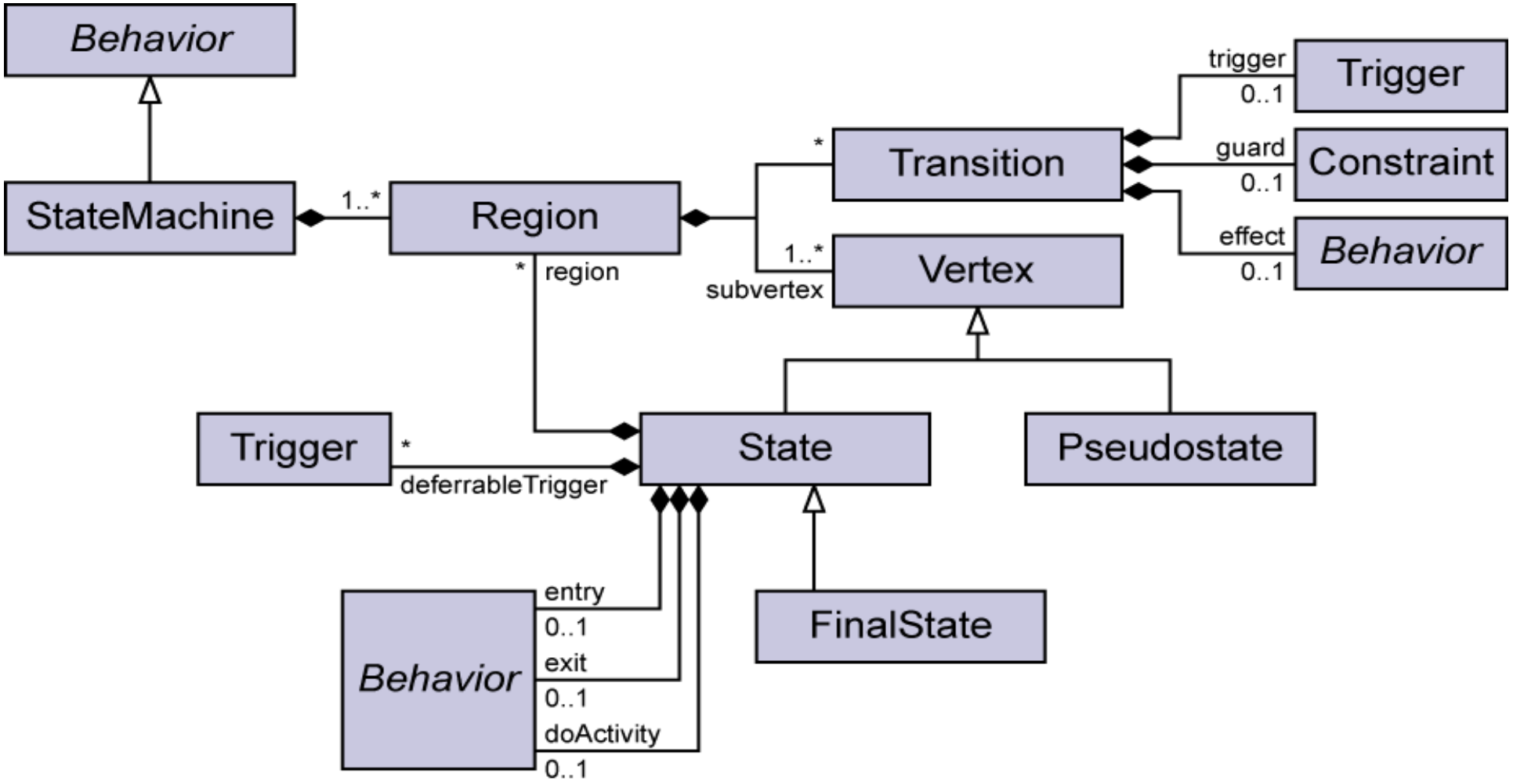
History states

- History states represent the last active
  - substate (shallow history), or
  - configuration (deep history)

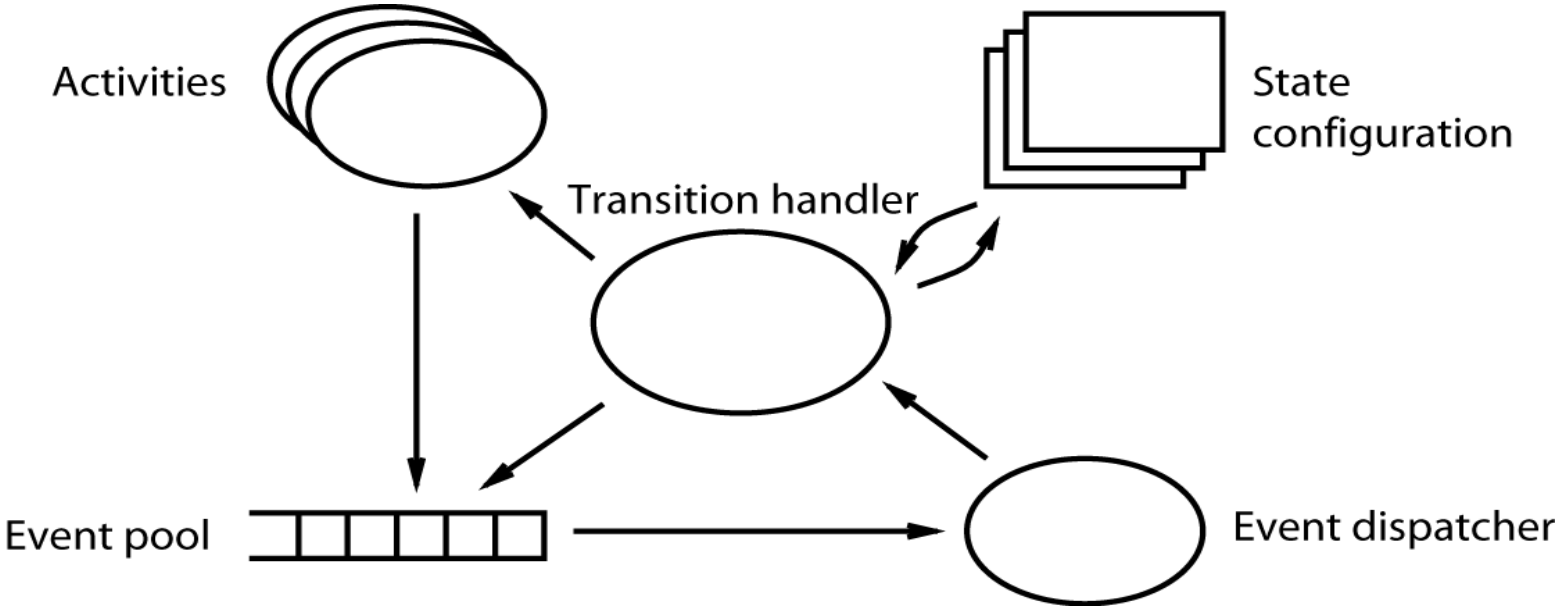
  of a region.



Metamodel


Behavior
StateMachine
1..*
Region
*
region
*
Transition
1..*
subvertex
Vertex
trigger
0..1
Trigger
guard
0..1
Constraint
effect
0..1
Behavior
Trigger
*
deferrableTrigger
State
Pseudostate
FinalState
Behavior
entry
0..1
exit
0..1
doActivity
0..1

Run-to-Completion Step: Overview

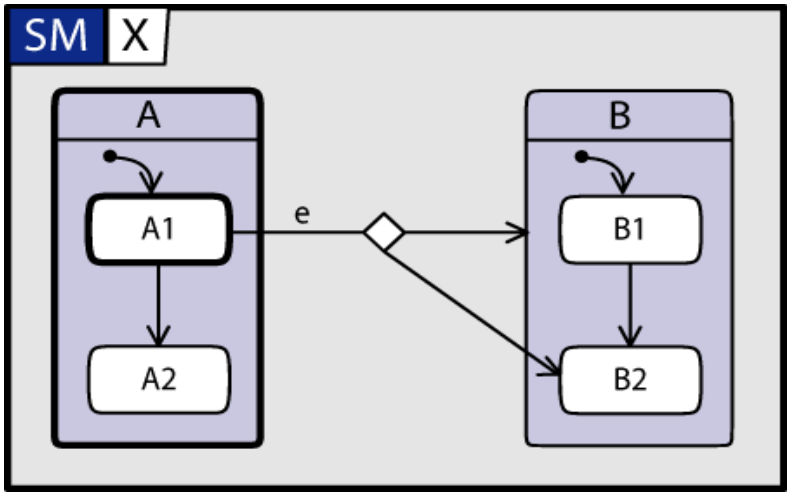


- Choose an **event** from the event pool (queue)
- Choose a **maximal**, **conflict-free**, **prioritized**, set of transitions enabled by the event
- Execute set of transitions
  - exit source states (inside-out)
  - execute transition effects
  - enter target states (outside-in)

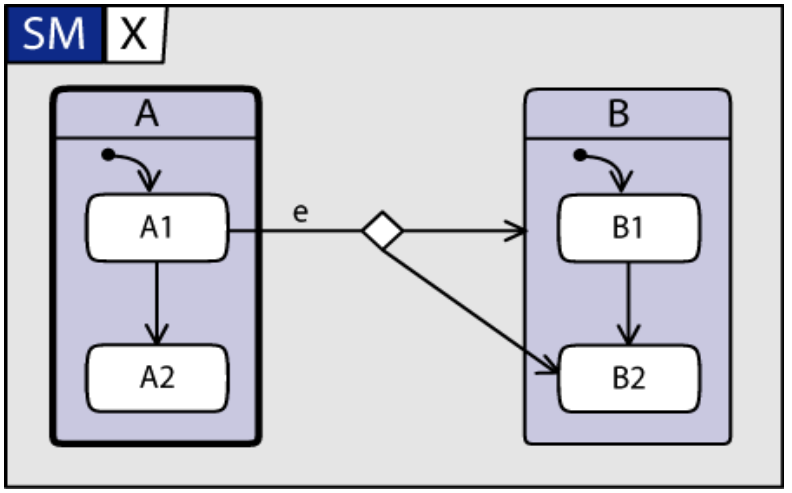
  thereby generating new events and activities

Run-to-Completion Step: Preliminaries (1)

- **Active state configuration**
  - the states the state machine currently is in
  - forms a tree
    - if a composite state is active, all its regions are active
- **Least-common-ancestor** (LCA) of states s_1 and s_2
  - the least region or orthogonal state (upwards) containing s_1 and s_2



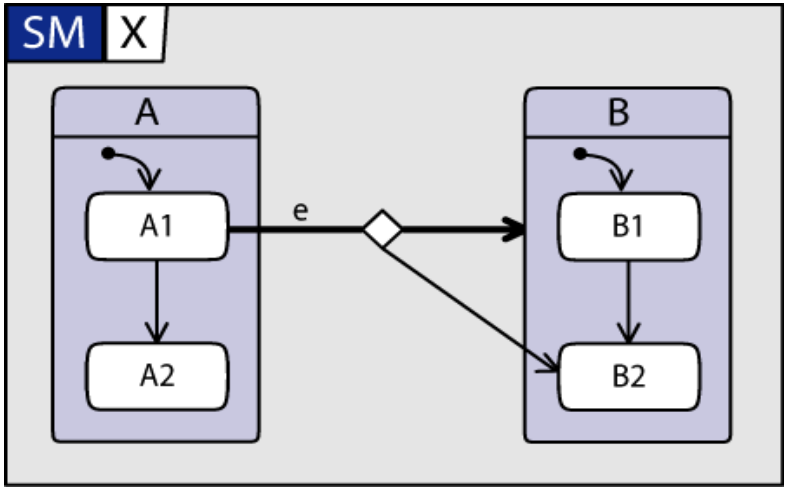
bold: active state configuration

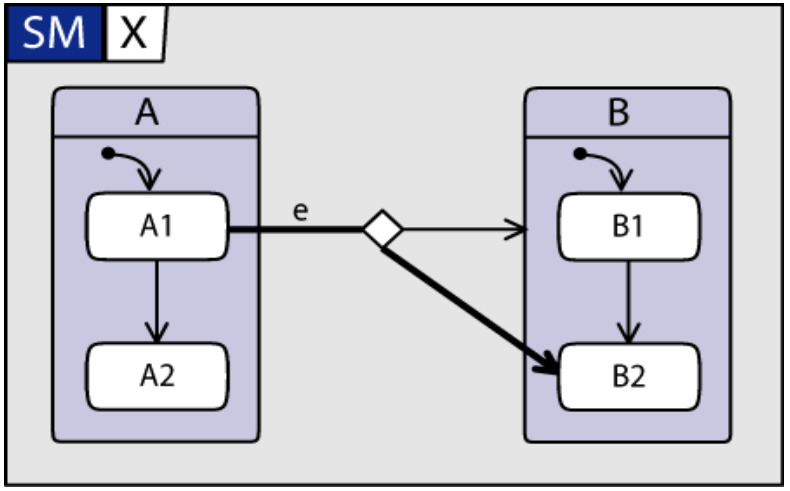


bold: LCA of states A1 and A2

Run-to-Completion Step: Preliminaries (2)

- **Compound** transitions
  - transitions for an event are “chained” into compound transitions
    - eliminating pseudostates like junction, fork, join, entry, exit
    - this is not possible for choice pseudostates where the guard of outgoing transitions are evaluated dynamically (in contrast to junctions)
  - several source and target states





Run-to-Completion Step: Preliminaries (3)

- **Main source** / **target state** m of compound transition t
  - Let s be LCA of all source and target states of t
  - If s region: m = direct subvertex of s containing all source states of t
  - If s orthogonal state: $m = s$
  - Similarly for main target state
  - All states between main source and explicit source states are exited, all state between main target and explicit target states are entered.

- **Conflict** of compound transitions t_1 and t_2
  - intersection of states exited by t_1 and t_2 not empty

- **Priority** of compound transition t_1 over t_2
  - s_i “deepest” source state of transition t_i
  - s_1 (direct or transitive) substate of s_2

Run-to-Completion Step (1)

RTC(*env*, *conf*) ≡
⌈*event* ← `fetch`()
step ← choose steps(*conf*, *event*)
if *step* = ∅ ∧ *event* ∈ *deferred*(*conf*)
then `defer`(*event*)
fi
for *transition* ∈ *step* do
conf ← handleTransition(*env*, *conf*, *transition*)
od
if *isCall* (*event*) ∧ *event* ∉ *deferred*(*conf*)
then `acknowledge`(*event*)
fi
conf ⌋

Run-to-Completion Step (2)

steps(*env*, *conf*, *event*) ≡
  ⌈ *transitions* ← *enabled*(*env*, *conf*, *event*)
    {*step* | (*guard*, *step*) ∈ steps(*conf*, *transitions*) ∧ *env* ⊨ *guard* } ⌋

steps(*conf*, *transitions*) ≡
  ⌈ *steps* ← {(true, ∅)}
    for *transition* ∈ *transitions* do
        for (*guard*, *step*) ∈ steps(*conf*, *transitions* \ {*transition*}) do
          if *inConflict*(*conf*, *transition*, *step*)
          then if *higherPriority*(*conf*, *transition*, *step*)
                then *guard* ← *guard* ∧ ¬*guard*(*transition*) fi
          else *step* ← *step* ∪ {*transition*}
                guard ← *guard* ∧ *guard*(*transition*) fi
          steps ← *steps* ∪ {(*guard*, *step*)} od od
    steps ⌋

Run-to-Completion Step (3)

handleTransition(*conf*, *transition*) ≡
⌈ for *state* ∈ *insideOut*(*exited*(*transition*)) do
uncomplete(*state*)
for *timer* ∈ *timers*(*state*) do stopTimer(*timer*) od
execute(*exit*(*state*))
conf ← *conf* \ {*state*}
od
execute(*effect*(*transition*))
for *state* ∈ *outsideIn*(*entered*(*transition*)) do
execute(*entry*(*state*))
for *timer* ∈ *timers*(*state*) do startTimer(*timer*) od
conf ← *conf* ∪ {*state*}
complete(*conf*, *state*)
od
conf ⌋