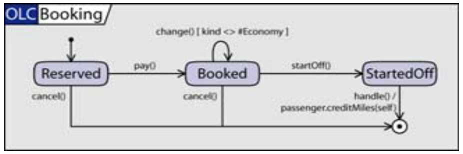
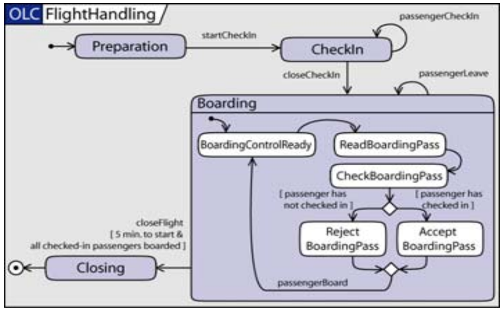
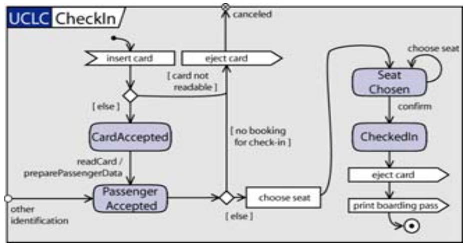
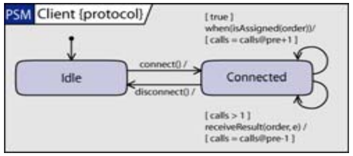
Unified Modeling Language 2

State machines









History and predecessors

- 1950’s: Finite State Machines
  - Huffmann, Mealy, Moore

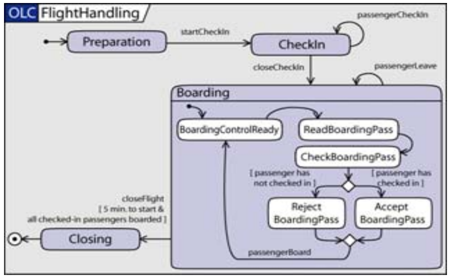
- 1987: **Harel Statecharts**
  - conditions
  - hierarchical (and/or) states
  - history states

- 1990’s: Objectcharts
  - adaptation to object orientation

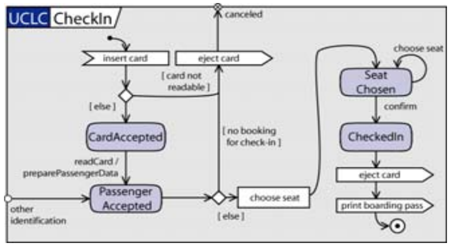
- 1994: **ROOM Charts**
  - run-to-completion (RTC) step

Usage scenarios

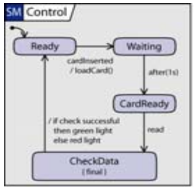
- **Object life cycle**
  - Behaviour of objects according to business rules
  - in particular for active classes

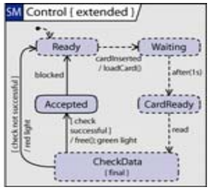


- Use case life cycle
  - Integration of use case scenarios
  - Alternative: activity diagrams

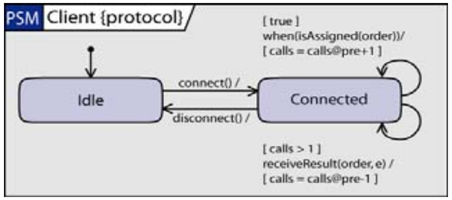


- Control automata
  - Embedded systems


SM Control
Ready
cardInserted / loadCard()
Waiting
after(1s)
CardReady
read
/ if check successful then green light else red light
CheckData { final }

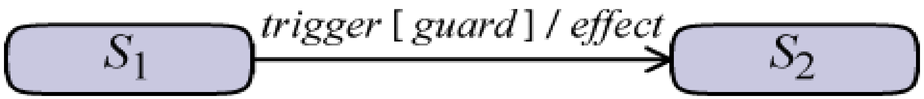

SM Control { extended }
Ready
cardInserted / loadCard()
Waiting
after(1s)
blocked
Accepted
CardReady
read
[check successful] / free(); green light
[check not successful] / red light
CheckData { final }

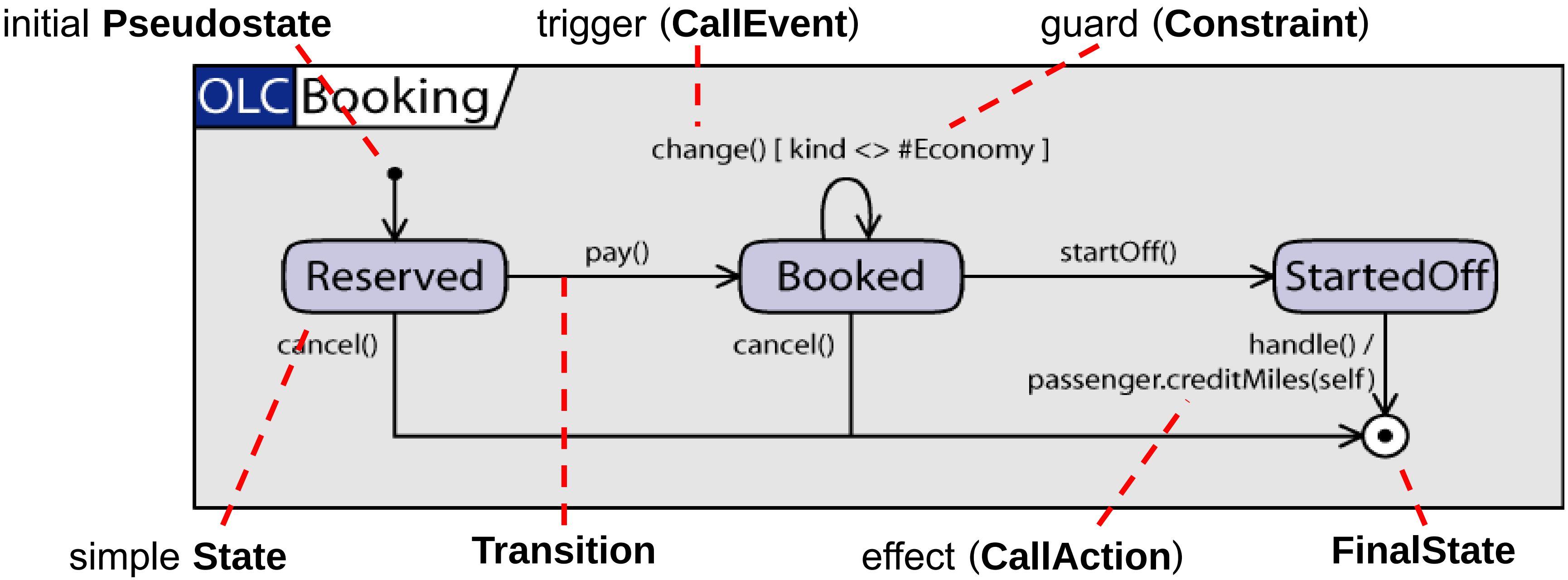
- **Protocol specification**
  - Communication interfaces


PSM Client {protocol}
Idle
connect() /
disconnect() /
Connected
[true] when(isAssigned(order))/ [calls = calls@pre+1]
[calls > 1] receiveResult(order, e) / [calls = calls@pre-1]

States and transitions

- State machines model behaviour
  - using **states** interconnected …
  - with **transitions** triggered …
  - by **event** occurrences.





initial **Pseudostate**

trigger (**CallEvent**)

guard (**Constraint**)

simple **State**

Transition

effect (**CallAction**)

FinalState

Relation to class diagrams

- State machines are defined in the context of a BehavioredClassifier.

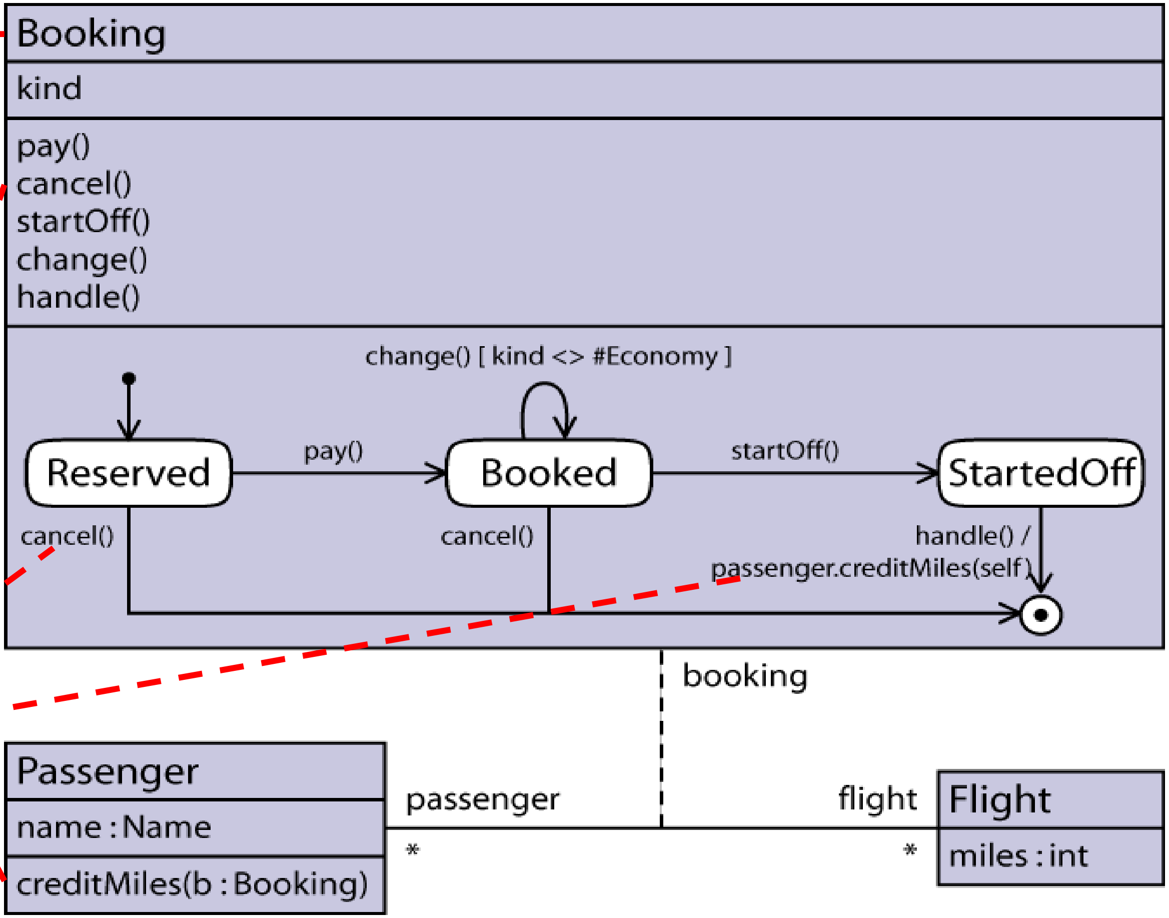
- **Context** defines which
  - events can occur
  - features are available

Operation

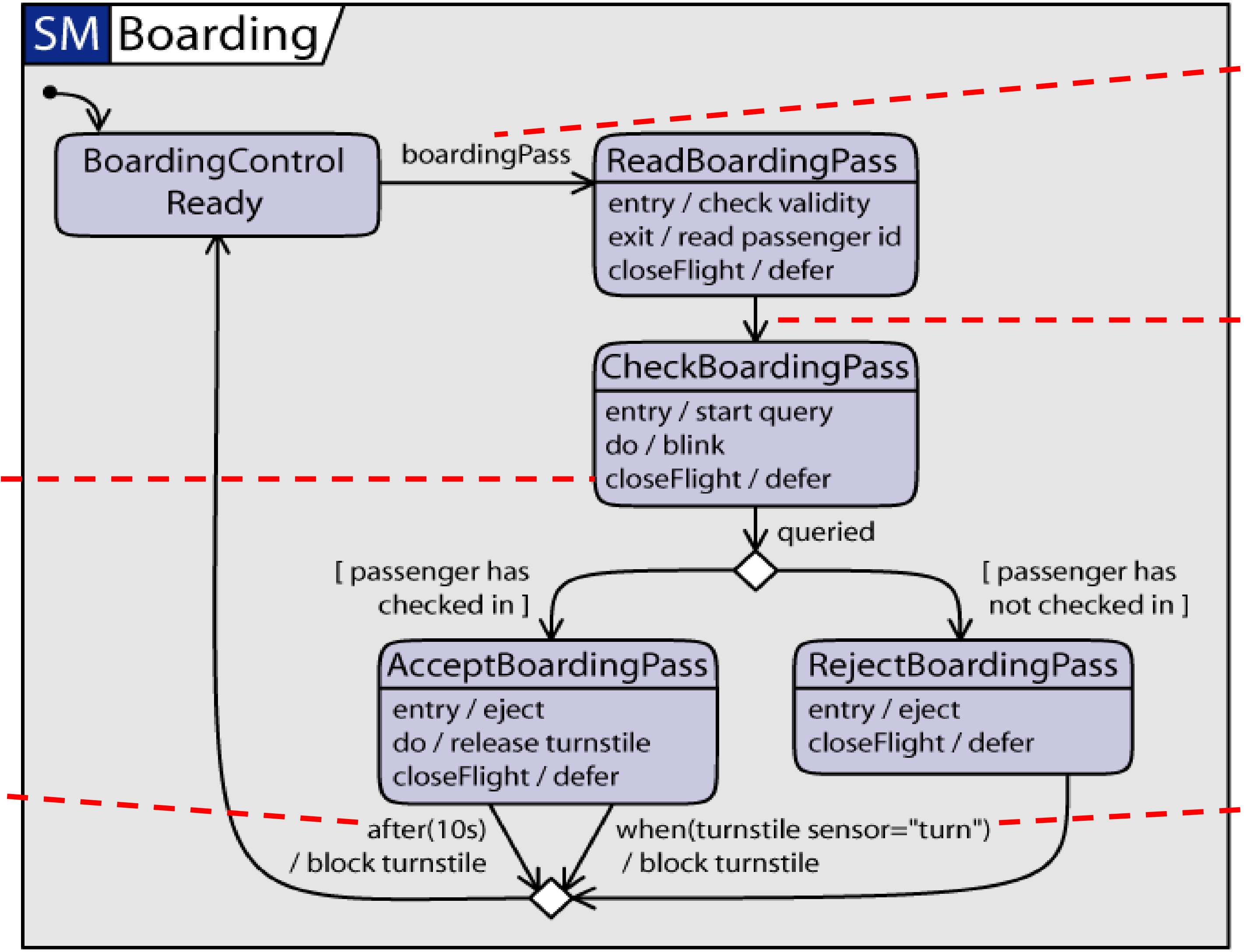
corresponding **CallEvent**

CallAction

called **Operation**



Triggers and events (1)



SignalEvent

completion event (no explicit trigger)

deferred event

TimeEvent (relative)

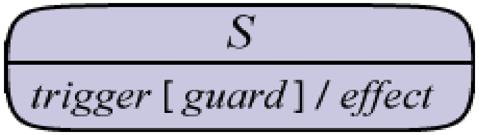
ChangeEvent

Triggers and events (2)

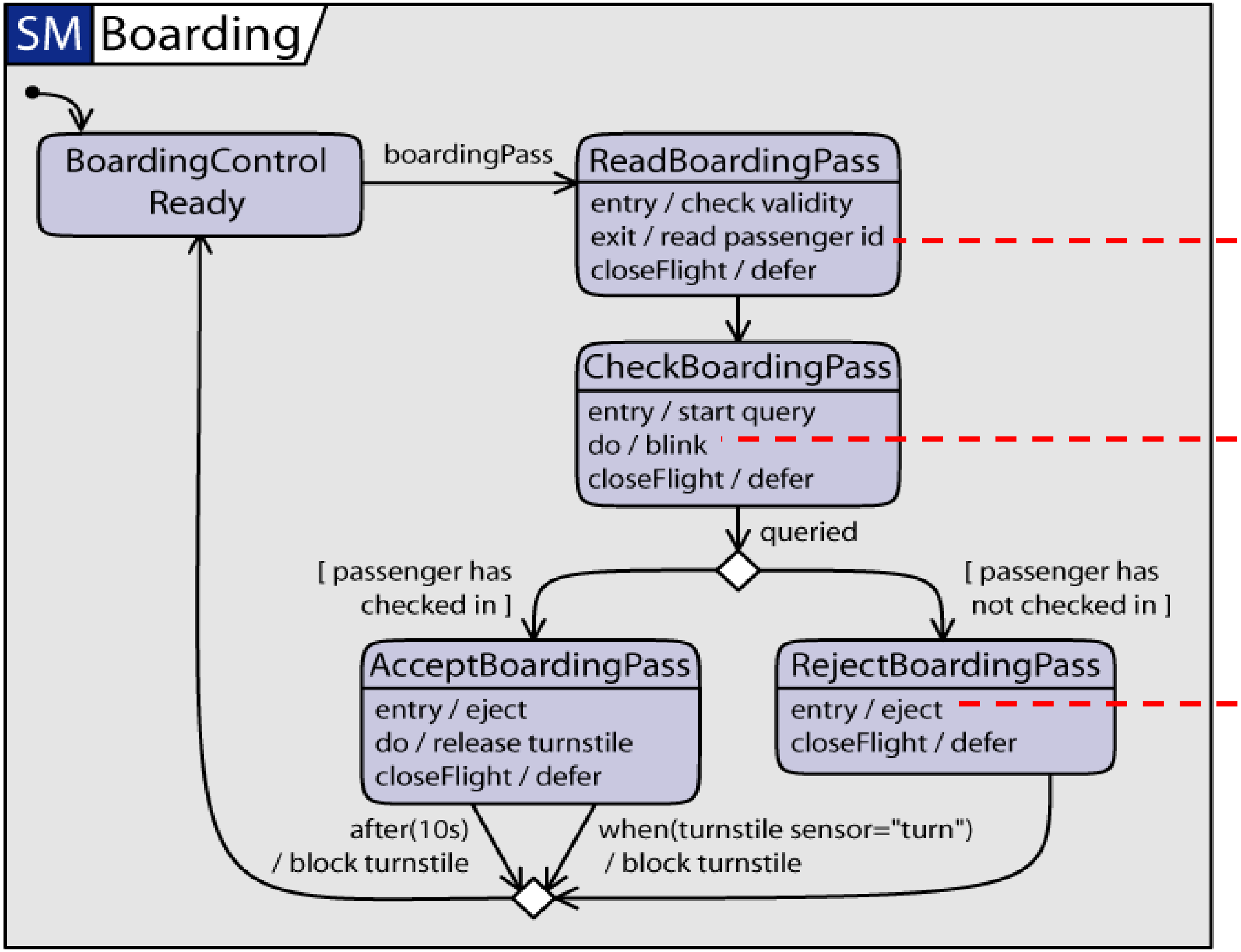
- **CallEvent**
  - receipt of a (a)synchronous Operation call
  - triggering after Behavior of Operation executed
- **SignalEvent**
  - receipt of an asynchronous Signal instance
  - reaction declared by a Reception for the Signal
- TimeEvent
  - absolute reference to a time point (at *t*)
  - relative reference to trigger becoming active (after *t*)
    - presumably meaning relative to state entry
- ChangeEvent
  - raised each time condition becomes true
    - may be raised at some point after condition changes to true
    - could be revoked if condition changes to false

Triggers and events (3)

- **Completion event**
  - raised when all internal activities of a state are finished
    - do activity, subregion
    - no metamodel element for completion events
  - dispatched before all other events in the event pool
- Deferred events
  - events that cannot be handled in a state but should be kept in the event pool
    - reconsidered when state is changed
    - no predefined deferring policy
- Internal transitions
  - … are executed without leaving and entering their containing state
    - normally, on transition execution states are left and entered



Behaviours

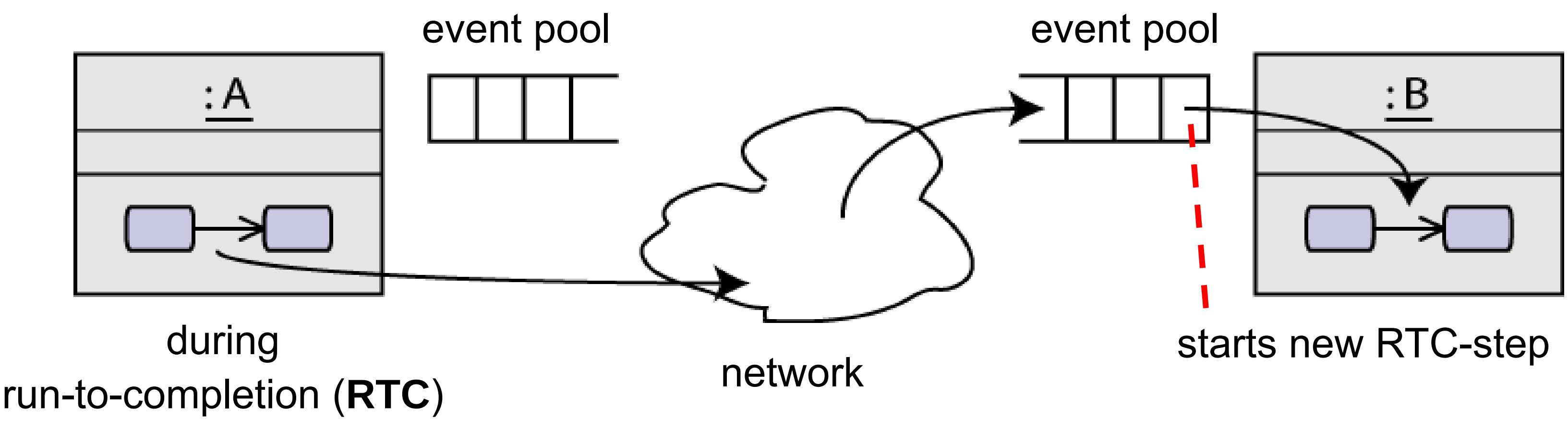


exit **Behavior**
(on exiting a state)

do activity **Behavior**
(concurrently while in state, may be interrupted)

entry **Behavior**
(on entering a state)

How state machines communicate



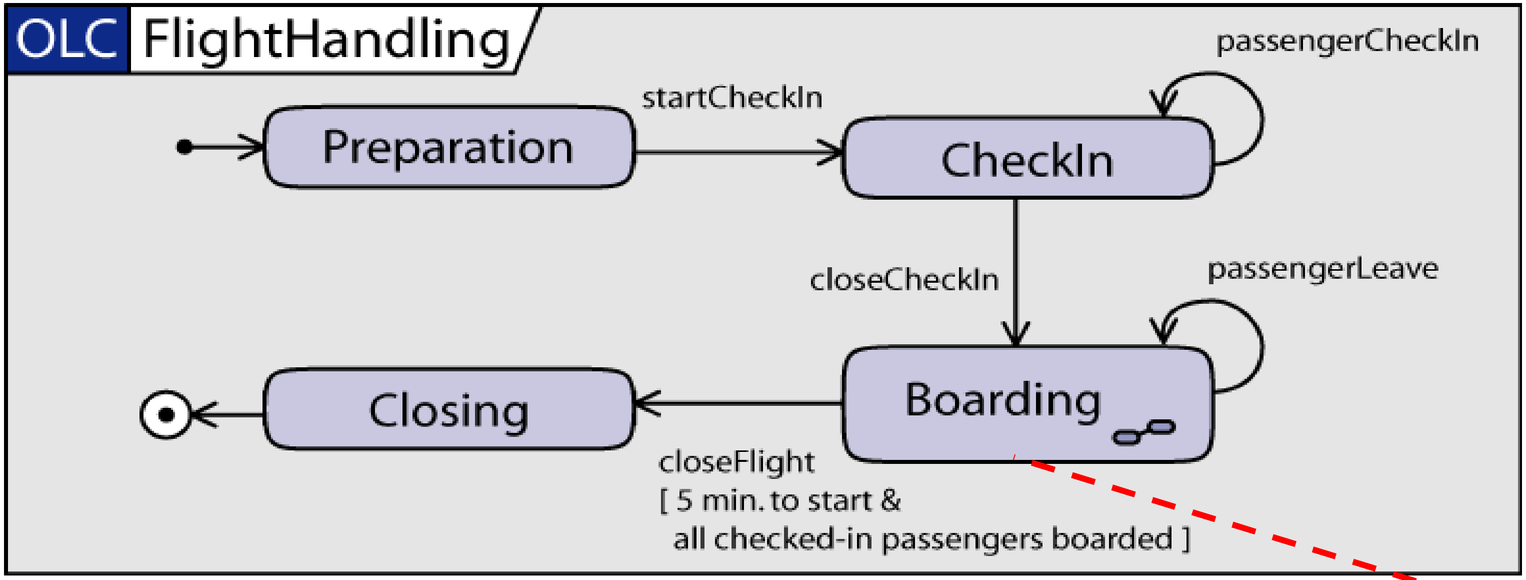
signals: *asynchronous* (no waiting)
calls: *asynchronous* or *synchronous* (waiting for RTC of callee)

No assumptions are made on timing between
event occurrence, event dispatching, and event consumption.

Event occurrences for which no trigger exists may be discarded
(if they are not deferred).

Hierarchical states (1)

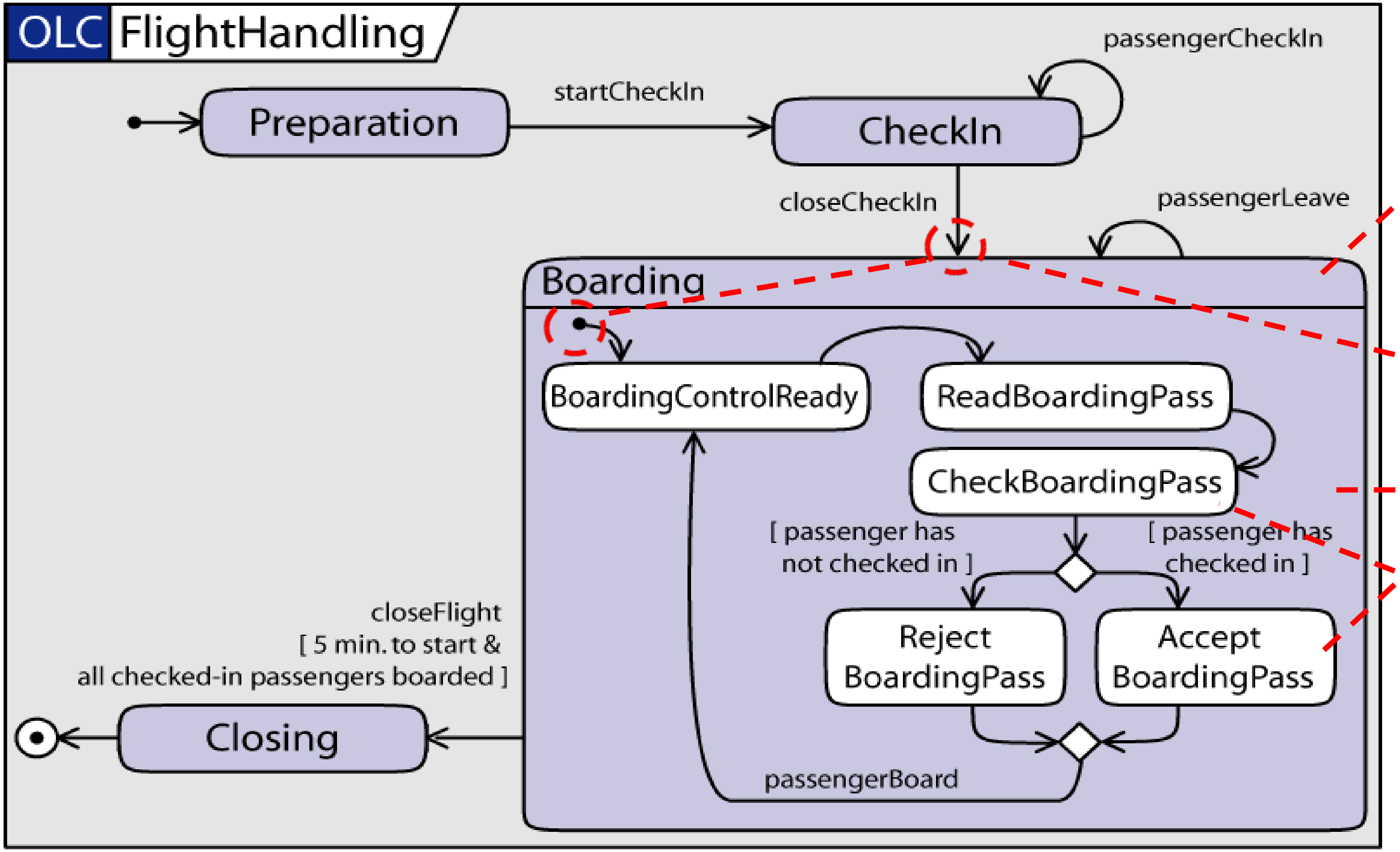
- Hierarchical states allow to **encapsulate** behaviour and facilitate reuse.
- However, they are rarely used this way.
- UML 2.0 provides concepts supporting this usage.
  - entry and exit points



composite **State**

Transition triggering is **prioritized** inside-out, i.e., transitions deeper in the hierarchy are considered first.

Hierarchical states (2)



detailed (non-orthogonal) composite **State**

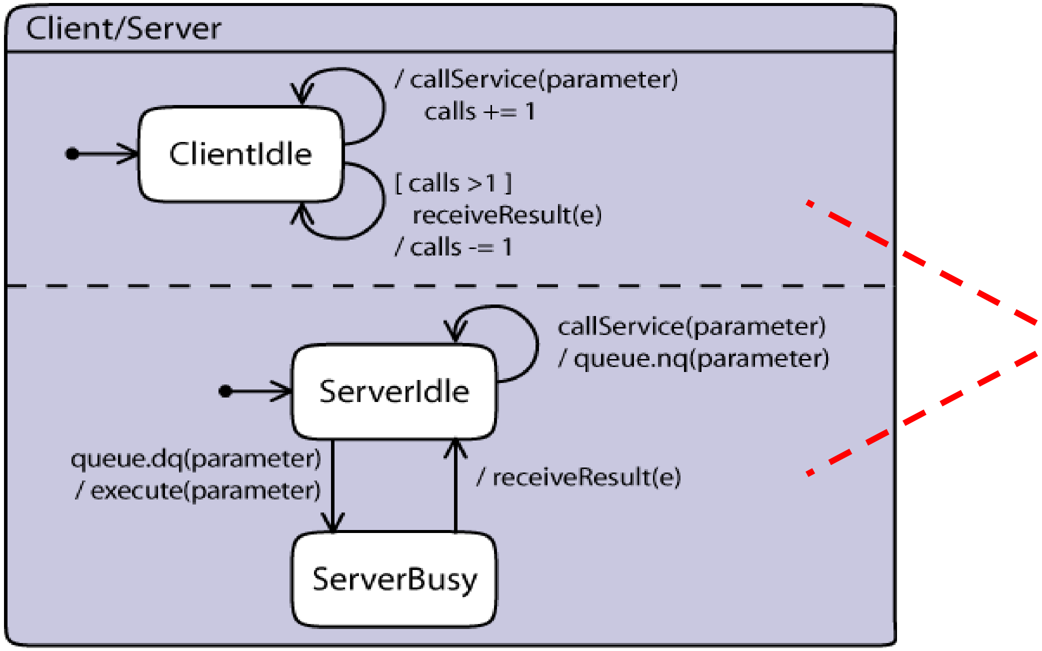
default entry

Region

substates

Orthogonal regions

- **Simple State**: containing no Region
- **Composite State**: containing at least one Region
  - simple composite State: exactly one
  - orthogonal composite State: at least two



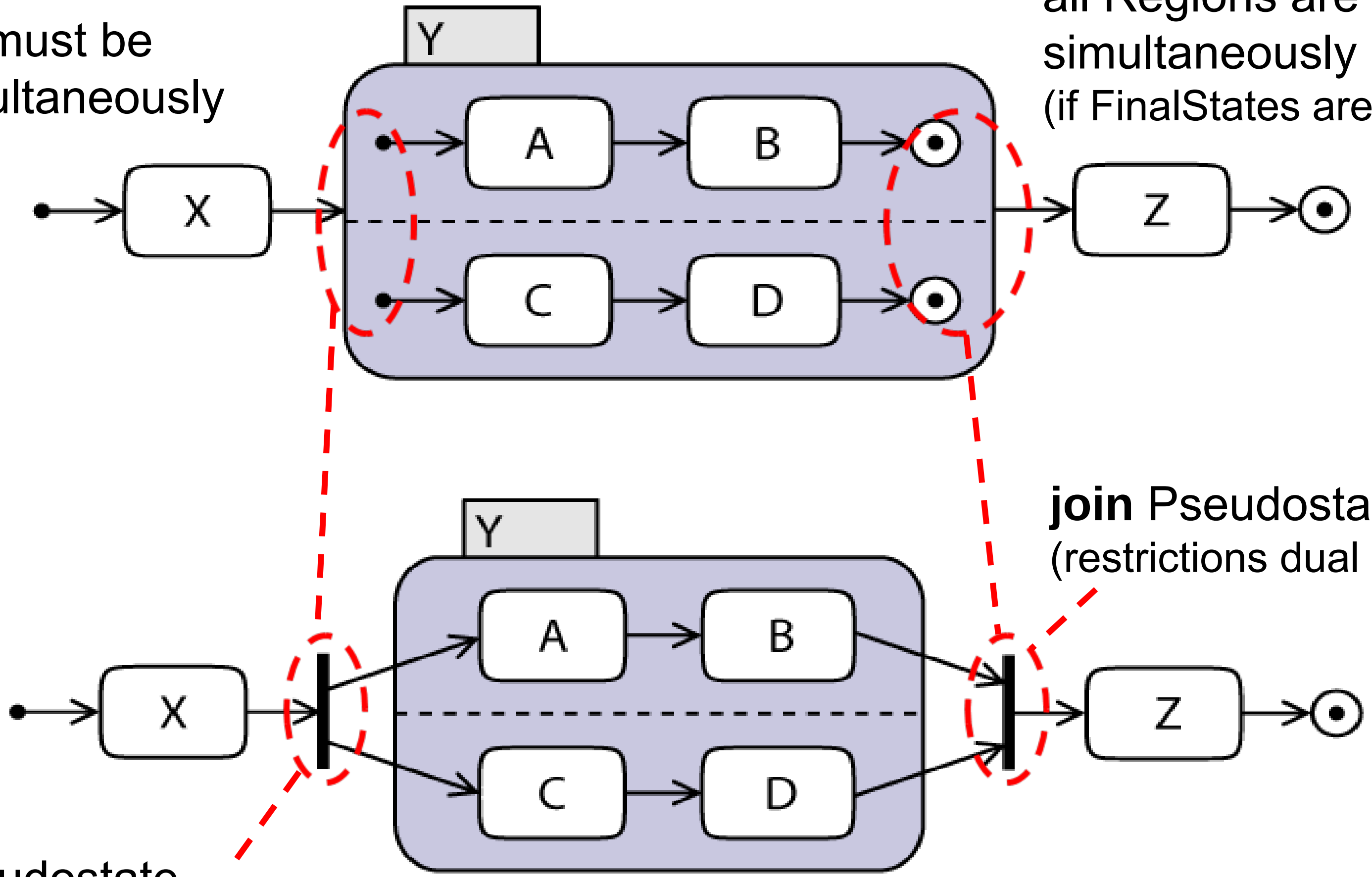
orthogonal **Regions**, both active if Client/Server active

orthogonal states are "concurrent" as a single event may trigger a transition in each orthogonal region "simultaneously"

Forks and joins

all Regions must be entered simultaneously

all Regions are left simultaneously
(if FinalStates are reached)



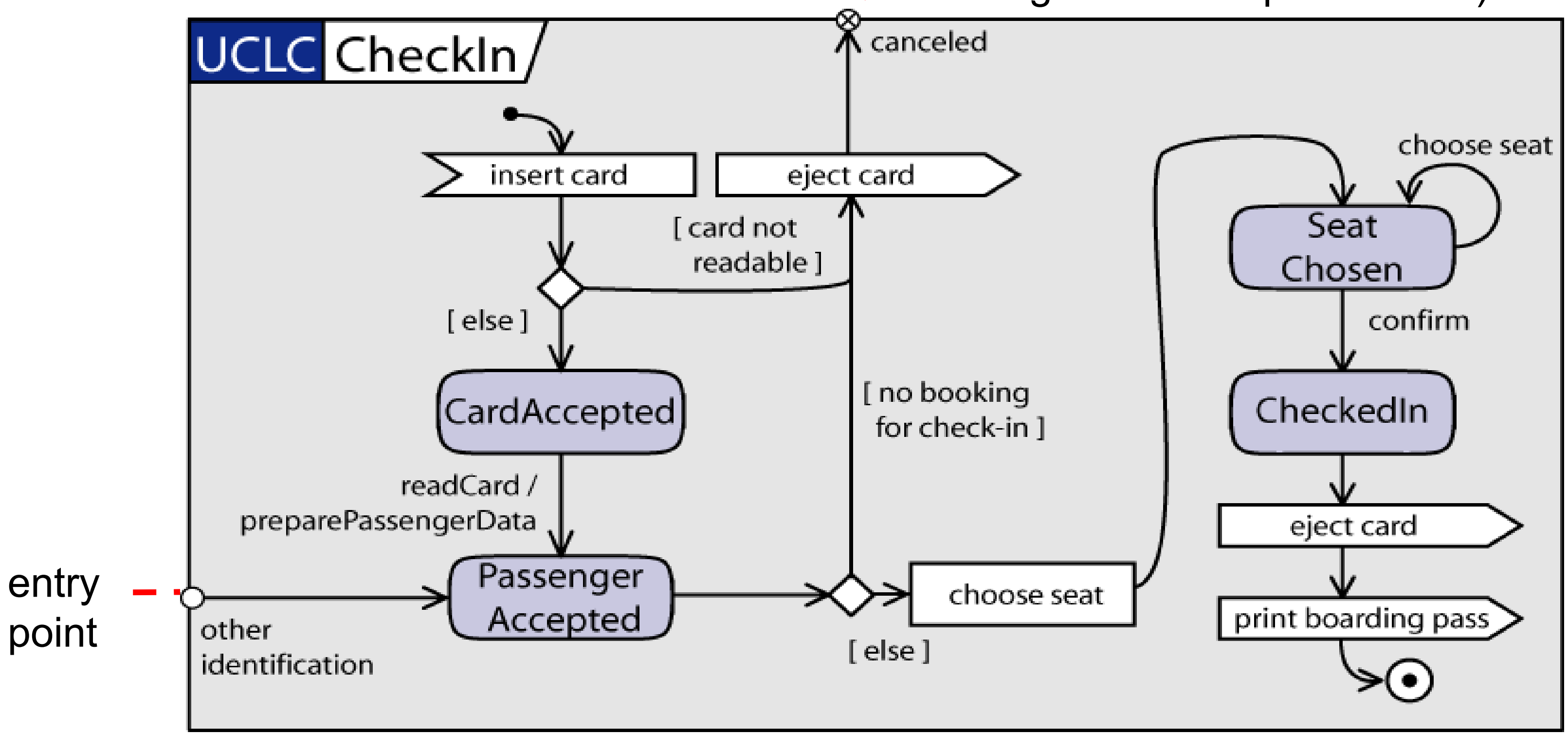
join Pseudostate
(restrictions dual to forks)

fork Pseudostate
(one incoming, at least two outgoing Transitions;
outgoing Transitions must target States in different Regions of an orthogonal State)

Entry and exit points (1)

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

exit point (on border of state machine diagram or composite state)



entry point