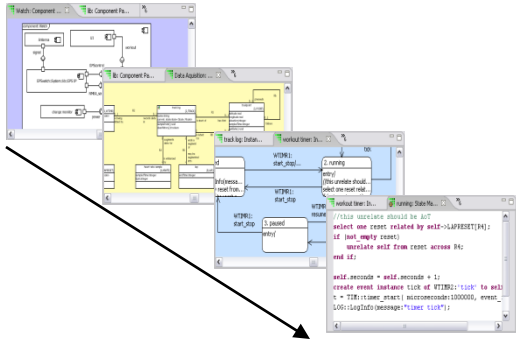
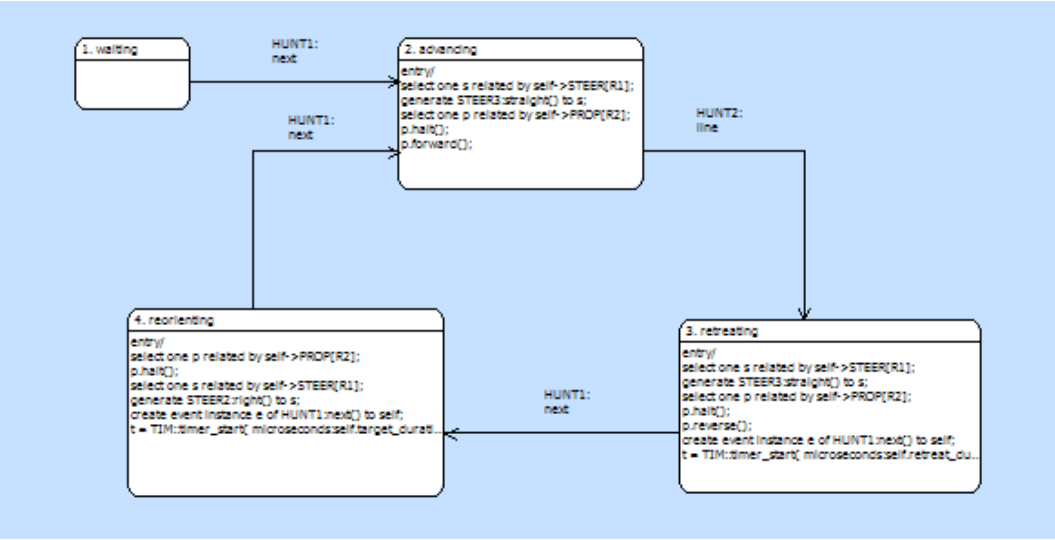
The xtUML method - Building State Machines

- **Analysis** – questioning, thinking, sketching...
  - Descriptive UML diagrams
    - use case, sequence, ...
- **Executable Modeling** – formalizing the analysis:
  - Component Diagrams (partitioning/interfaces)
  - Class Diagrams (data)
  - **State Machines (control)**
  - Activities (processing)
- **Verification**
  - Interpretive Model Execution
- **Code generation**
  - Template and Rule-Based Translation



State Machines = control flow

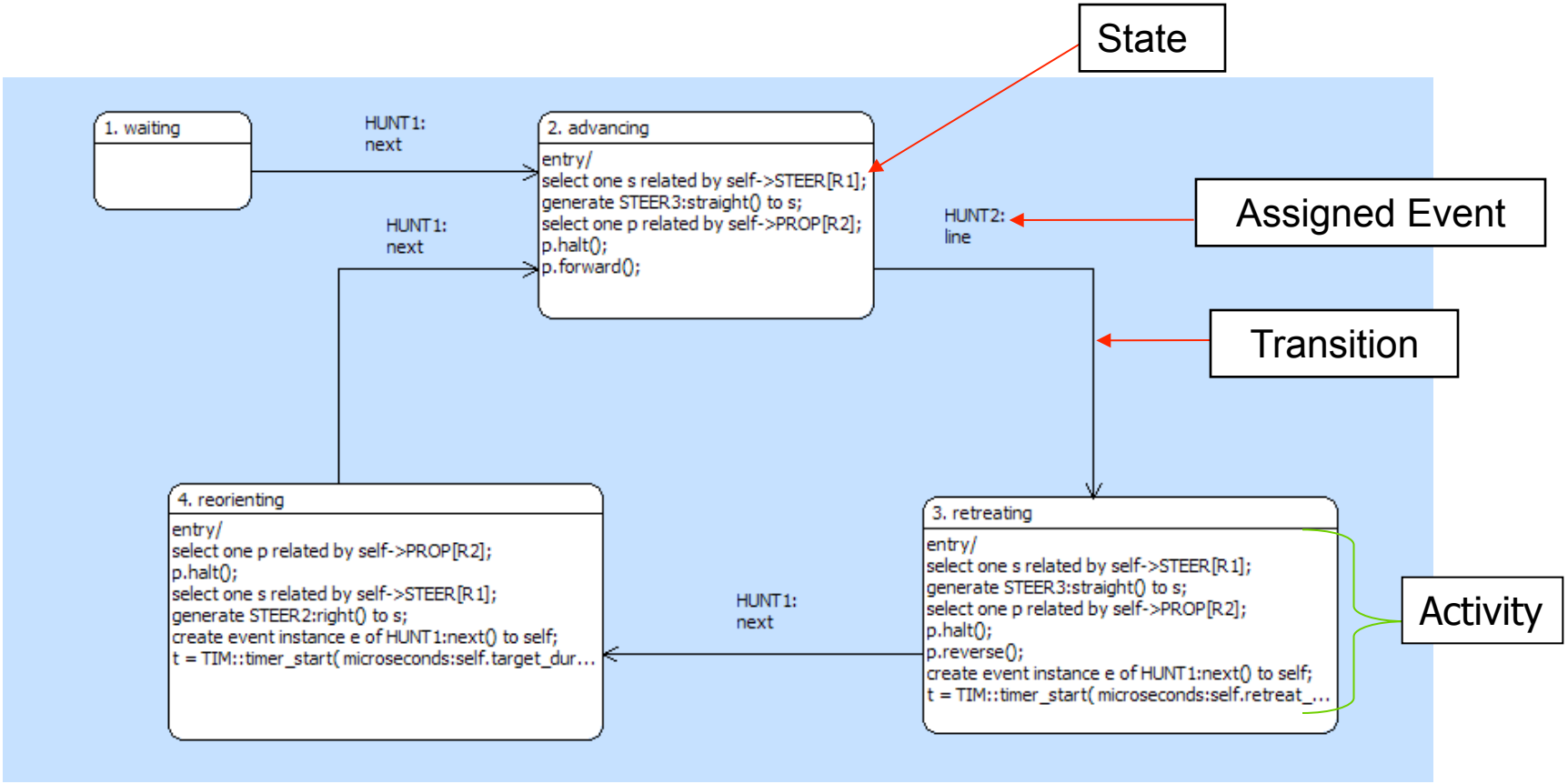
- **Some classes have dynamic behavior; they progress through stages during their lifetime.**
- **The collection of stages and the order of progression constitutes the lifecycle, represented as a state machine.**



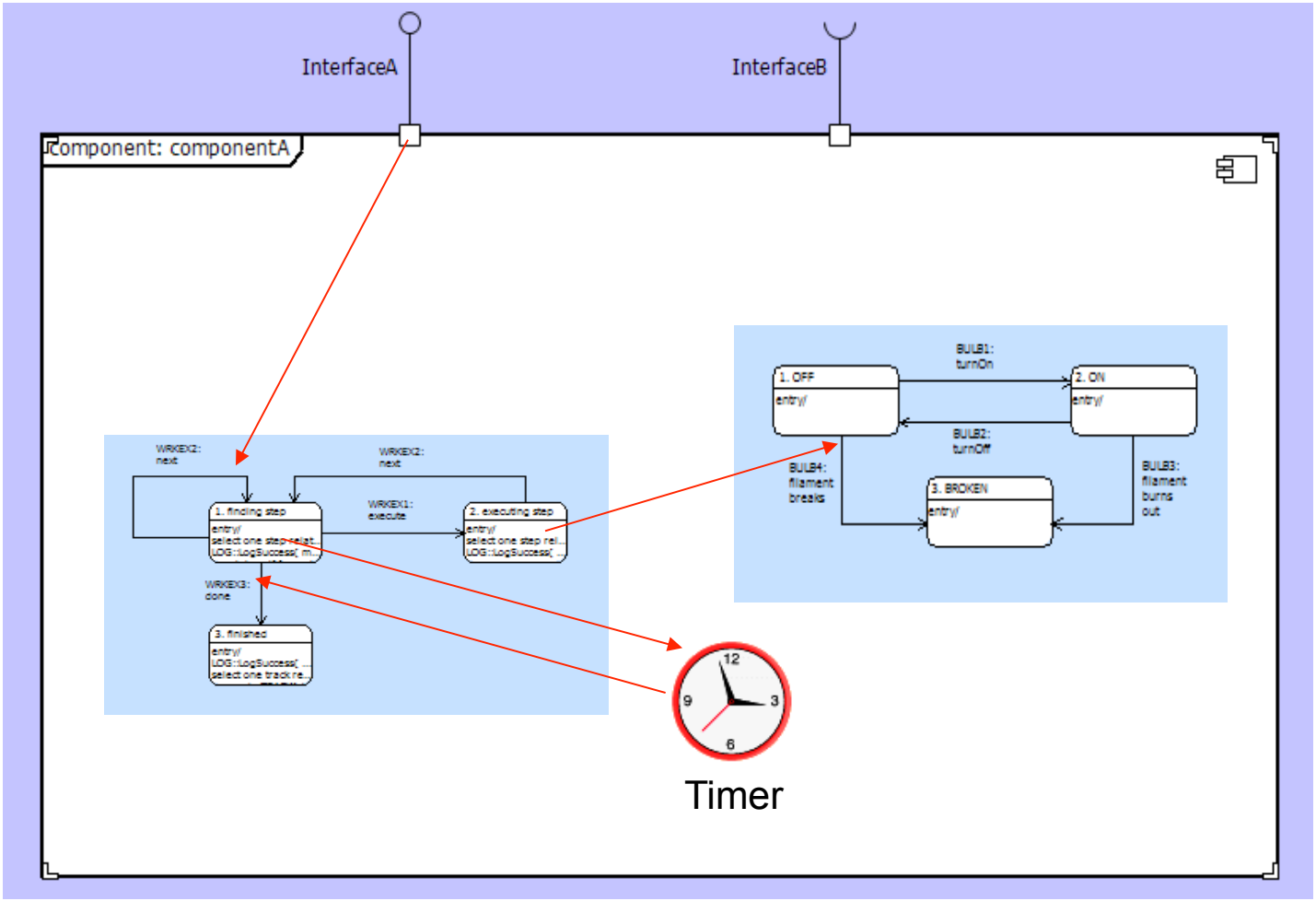
State Machines

- Each existing instance of a class is in exactly one state of its lifecycle at any instant.
- Transitions between states are driven by received signals, called events.
- Events arise from incidents in the real world appearing as signals on component interfaces.
- Internally, state machines may generate new events to propagate change to other state machines.
- Internally generated events may be delayed by timers.

State Machine Diagram Elements



Event Sources

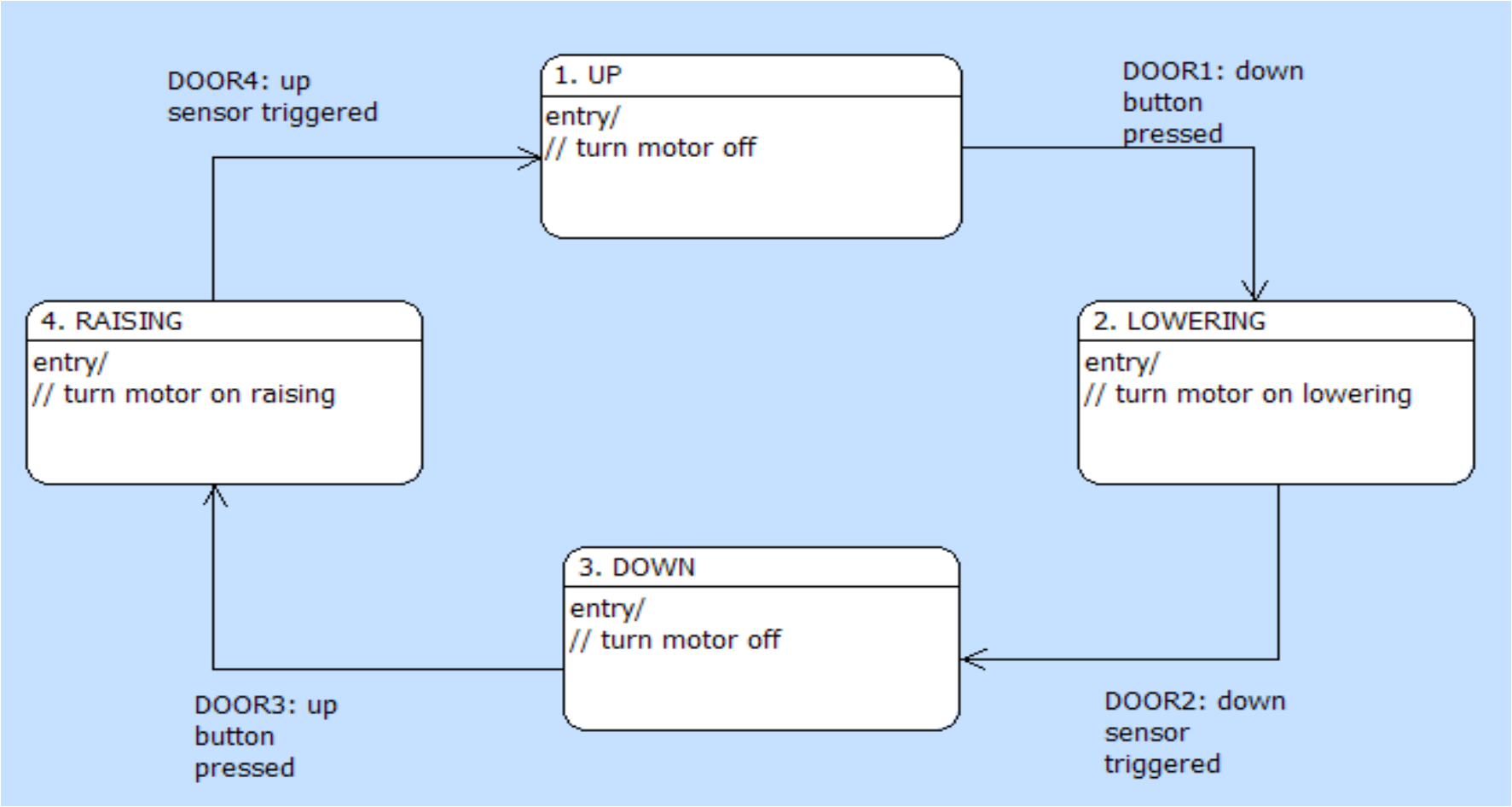


Timers

- A timer allows a pre-created event to be delivered at some future time
- The event is placed in the event queue when the timer expires
- One shot or recurring
- Delayed events define *minimum* delay

Checking for Completeness

- **An automatic garage door: Two Buttons and Position Sensors**



Filling the State Event Matrix

Events

Entry Action

	Down button	Down sensed	Up button	Up sensed	Action
UP	LOWERING				Stop motor
LOWERING		DOWN			Motor down
DOWN			RAISING		Stop motor
RAISING				UP	Motor up

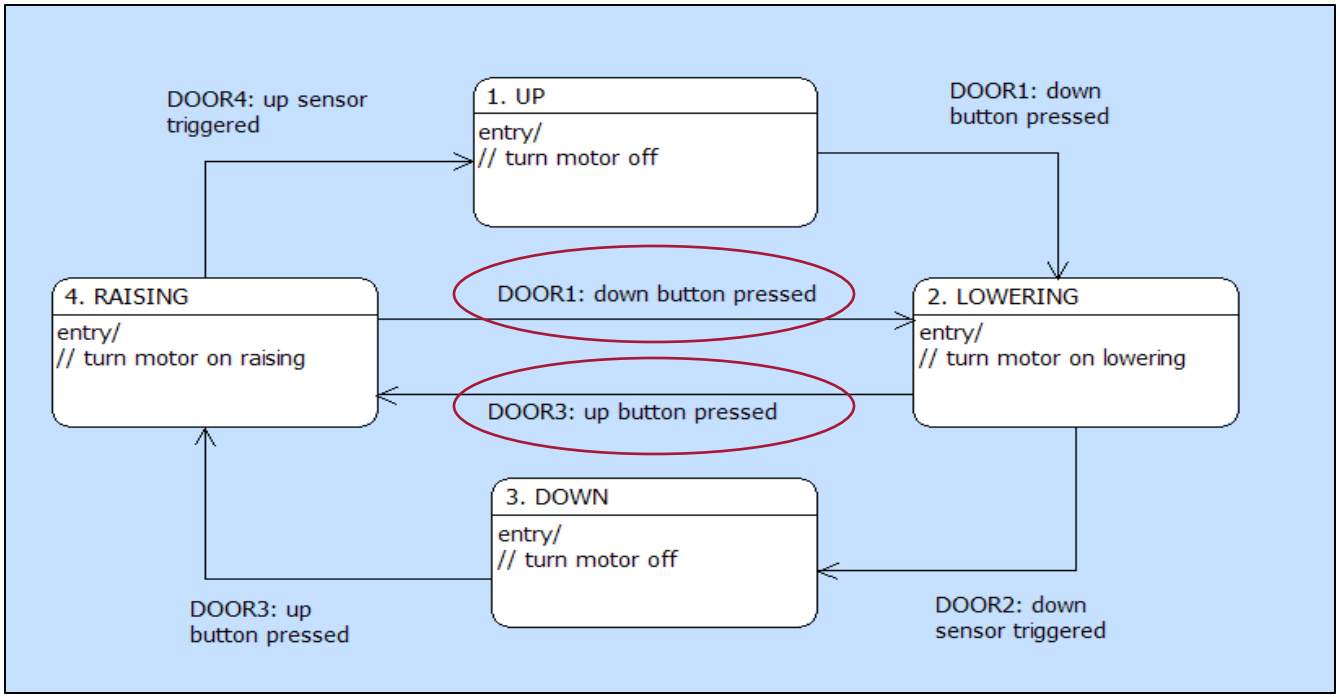
States

What do the empty cells mean?

Filling the State Event Matrix (cont)

	Down button	Down sensed	Up button	Up sensed	Action
UP	LOWERING	Can't happen	Event Ignored	Can't happen	Stop motor
LOWERING	Event Ignored	DOWN	RAISING	Can't happen	Motor down
DOWN	Event Ignored	Can't happen	RAISING	Can't happen	Stop motor
RAISING	LOWERING	Can't happen	Event Ignored	UP	Motor up

Completed Diagram



Initial and Final States

- When an instance of a class is synchronously created, it is placed in its initial state
  - It has not 'transitioned' into that state – so the 'entry' state action is not executed
  - An event assigned to a 'reflexive' transition can trigger execution of the action
  - The initial state is the lowest numbered state

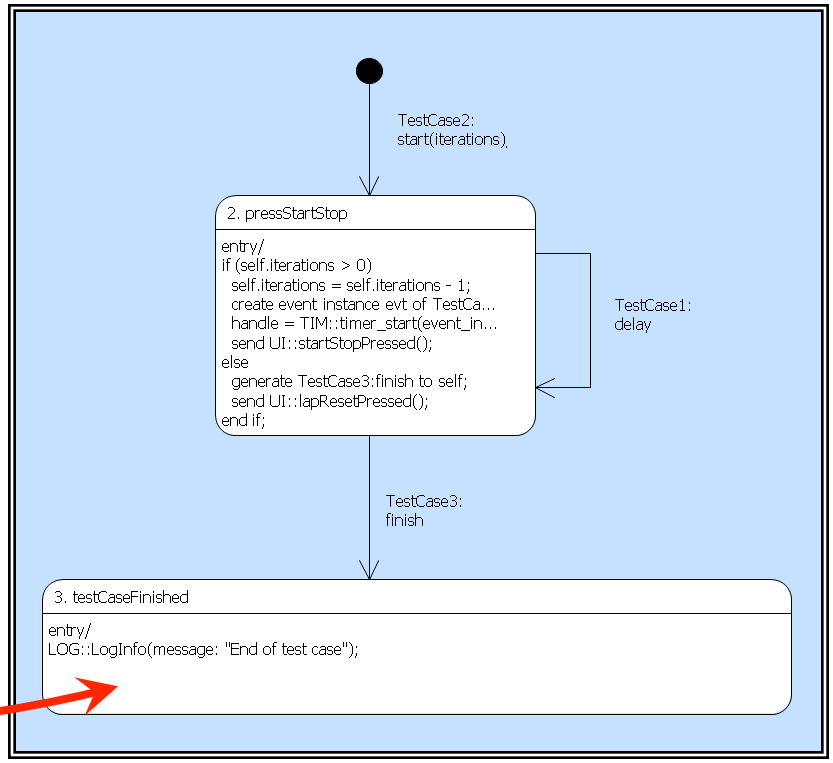
- A state machine may have one or more states with no outgoing transitions
  - Obviously, when such a state is entered, the lifecycle can never proceed further
  - The state can be marked as 'final'; after completing the state action, the instance will delete itself.

Creation Transition

- **It is possible to create instances asynchronously.**
- **New instance is created by the architecture and the event is enqueued to it**
- **Target state entry action is executed and event data is available to it.**

```
generate TestCase:start(iterations:5)
    to TestCase class;
```

This state is ‘final’



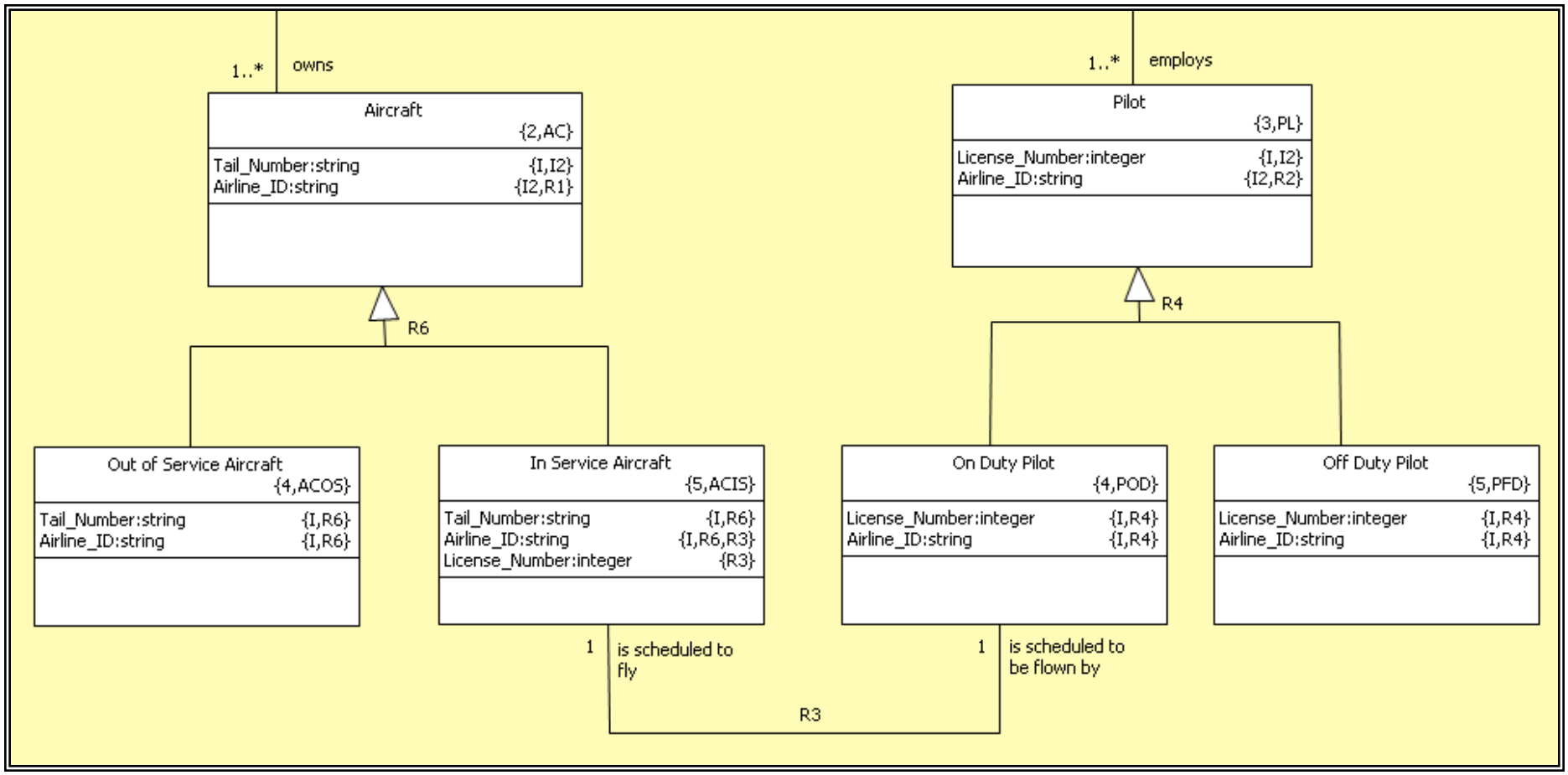
Common State Machine Patterns

- **One shot**
  - Manage an action that takes time to complete
  - No record of action is required (Born and Die)
  - Record of action is required (Born and Quiescent)
- **Cyclic**
  - Reusable resource such as equipment, link etc.
  - Usually returns to an ‘Idle’ state
- **Managing Contention for resources**
  - Can be modeled using a singleton class or as a Class State Machine
  - Class State Machine can co-exist with an Instance State Machine if required

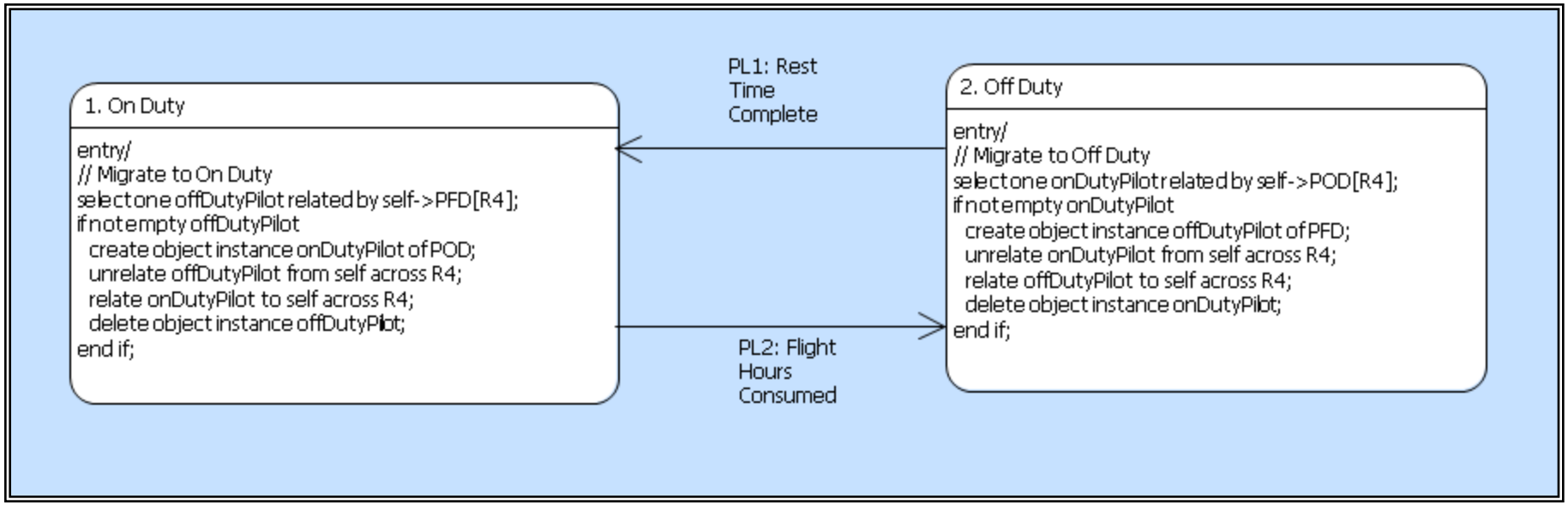
Constructing State Machines

- Draw and name the states you know.
- Write a comment: what does this state mean?
- Draw the transitions you know, into or out of each state.
- Do incomplete transitions suggest missing states?
- Define and name the known events.
- Assign an event to each transition; any missing events?
- Do events need to carry event data?
- Check for completeness; add discovered states/transitions.

Remember this?



Where Data and State Modeling meet



This is the state model for Pilot (PL).
Note the creation of a new subtype to reflect the change of state (and the destruction of the old subtype).
This technique is known as ‘Type Migration’.

Descriptions for Everything

- **State Machines**
  - Describe the scenarios the state machine is required to support
  - Before beginning to identify states and events
- **States**
  - Describe the state in which the instance is found
  - Before beginning to identify transitions
- **Events**
  - Describe the real world occurrence that the events abstract
  - Before beginning to assign to transitions

You know it makes sense