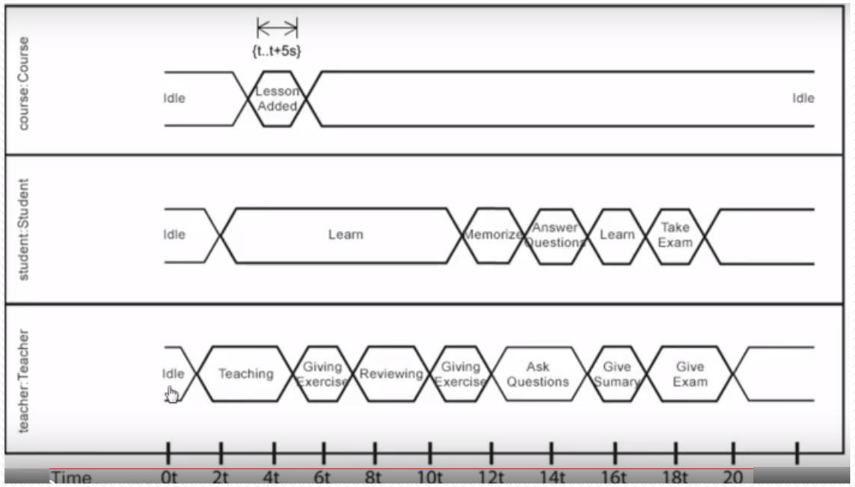
Interaction diagrams

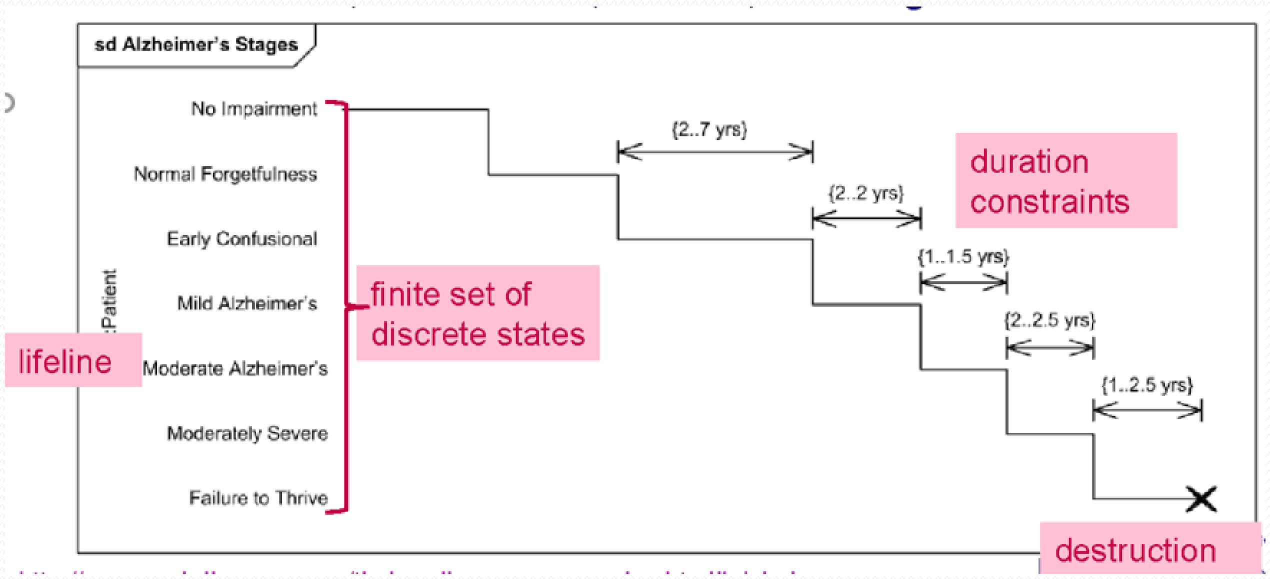
Timing Diagram

- Timing Diagram list different actions that are going to happen for the piece of project that we're going to draw the diagram for.
- It shows how long it takes each action to be processed and when each of them will be triggered.
- Use Case Description and its step of execution(main flow) and its extension help us in drawing activities (states) of the timing diagram.

Alternate Notations



Concise Notation



Robust Notation

Purpose of Timing Diagram

- Define the timing between different states in the system.
- Describe when events occur, how long it takes for participants to react and how long it take for each individual interaction to get complete.

Participants of timing Diagram

- Participants (it’s an object)
- States (actions)

Points to Consider

- If an object doesn't have any major interaction, then there's no reason to draw that object in a Timing Diagram.
- Events and message will be shown in the same way as it is shown on Sequence Diagram. They will be shown to trigger the state changes from one participants to another.

Steps of Timing Diagram

- Find states.
- Find the participants that these states belong to.
- Match the states to participants.

Drawing Timing Diagram

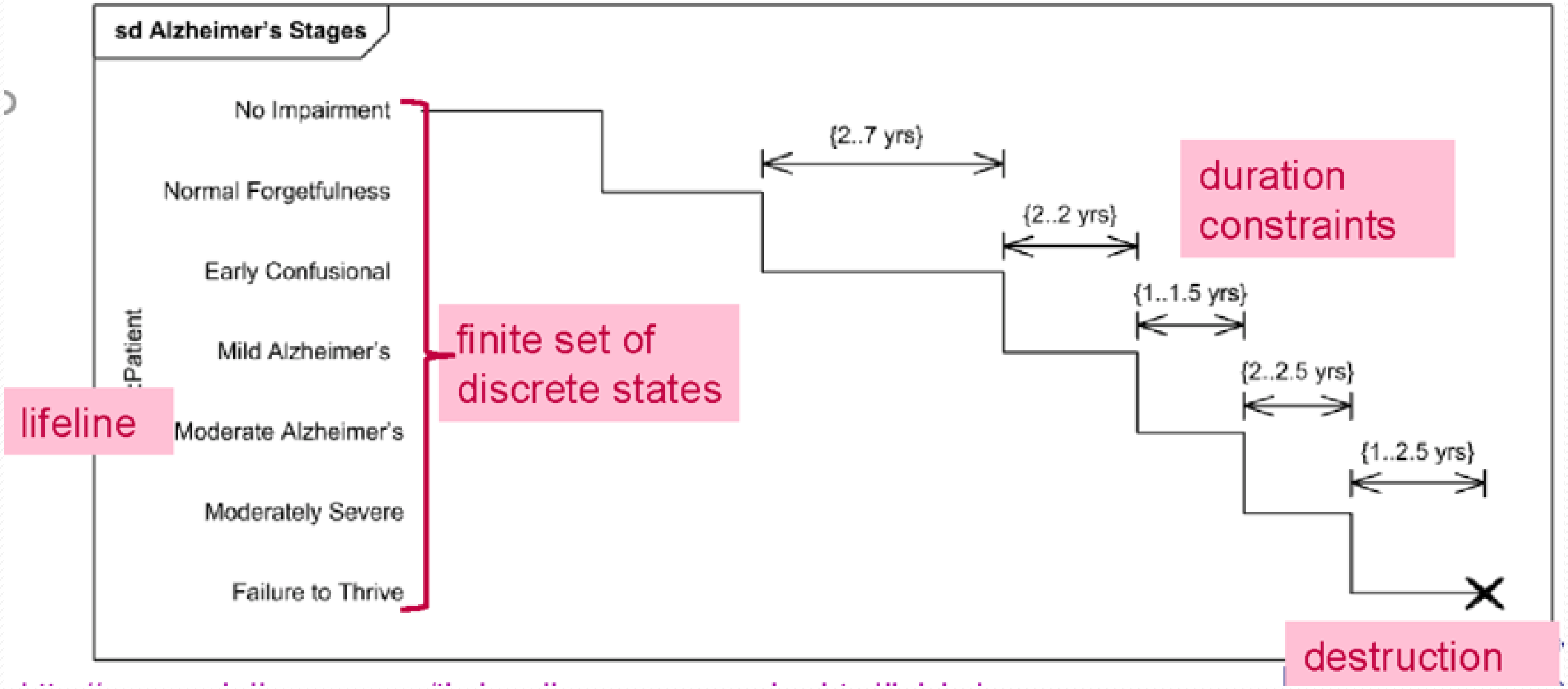
- We draw a table which contains participants in each row.
- Now we start the lifetime of the first action of the first participants that starts a system. As soon as we reach an event or a message we draw it and connect to next participants lifetime line and again continue.
- Then we move on and show how long it takes state of a participant to change to another state for all of the participants

Timing Diagrams

- A timing diagram allows you to show the interaction of objects and changes in state for those objects along a time axis.
- A timing diagram provides a convenient way to show active objects and their state changes during their interactions with other active objects and system resources.
- The X-axis of the timing diagram has the time units, while the Y-axis shows the objects and their states.
- Timing diagrams describe behavior of both individual **classifiers** & **interactions of classifiers**, focusing attention on time of events causing changes in the modeled conditions of the lifelines.

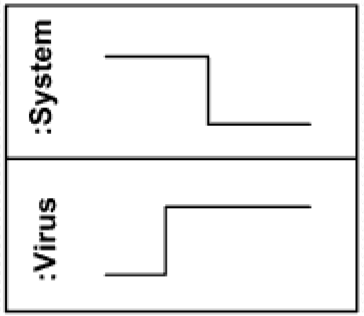
Timing Diagrams

- Timing Diagrams are Interaction diagram for reasoning about time.
- **Basic elements**: lifelines, states, duration/time constraints, destruction, events, messages



Lifeline

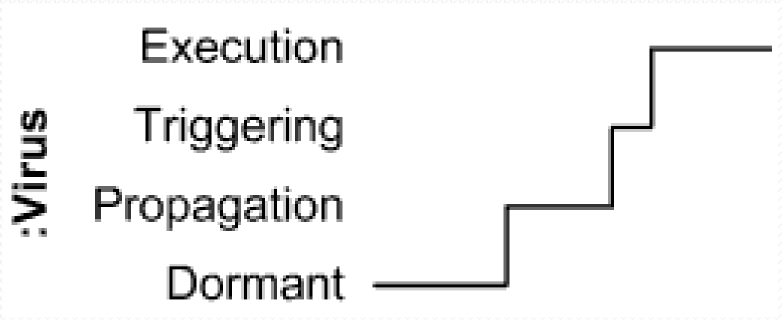
- **Lifeline** is a **named element** which represents an **individual participant** in the interaction. While **parts** and structural features may have multiplicity greater than 1, lifelines represent **only one** interacting entity.
- Lifeline on the timing diagrams is represented by the **name** of classifier or the instance it represents. It could be placed inside diagram frame or a "swimlane".



***Lifelines** representing instances of System and Virus*

State or Condition Timeline

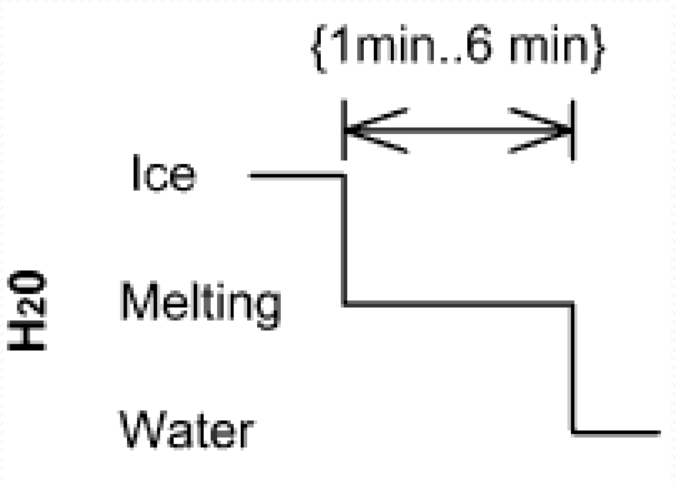
- Timing diagram could show **states** of the participating **classifier** or attribute, or some testable **conditions**, such as a discrete or enumerable value of an attribute.



- *Timeline shows Virus changing its state between Dormant, Propagation, Triggering and Execution state*

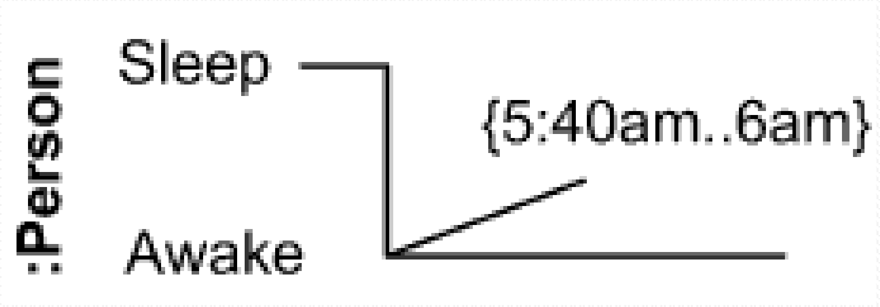
Duration Constraint

- **Duration constraint** is an **interval constraint** that refers to a **duration interval**. The duration interval is duration used to determine whether the constraint is satisfied.
- E.g., *Ice should melt into water in 1 to 6 minutes*



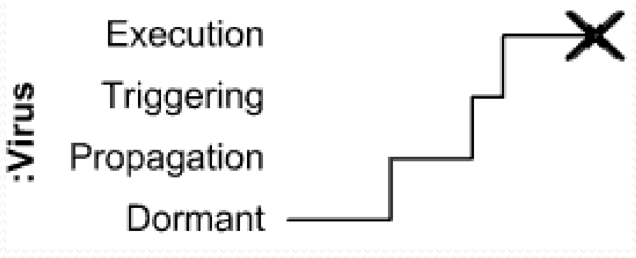
Time Constraint

- **Time constraint** is an **interval constraint** that refers to a **time interval**. The time interval is time expression used to determine whether the constraint is satisfied.
- Typically this graphical association is a small line, e.g., between an occurrence specification and a time interval.
- E.g., *Person should wake up between 5:40 am and 6 am*



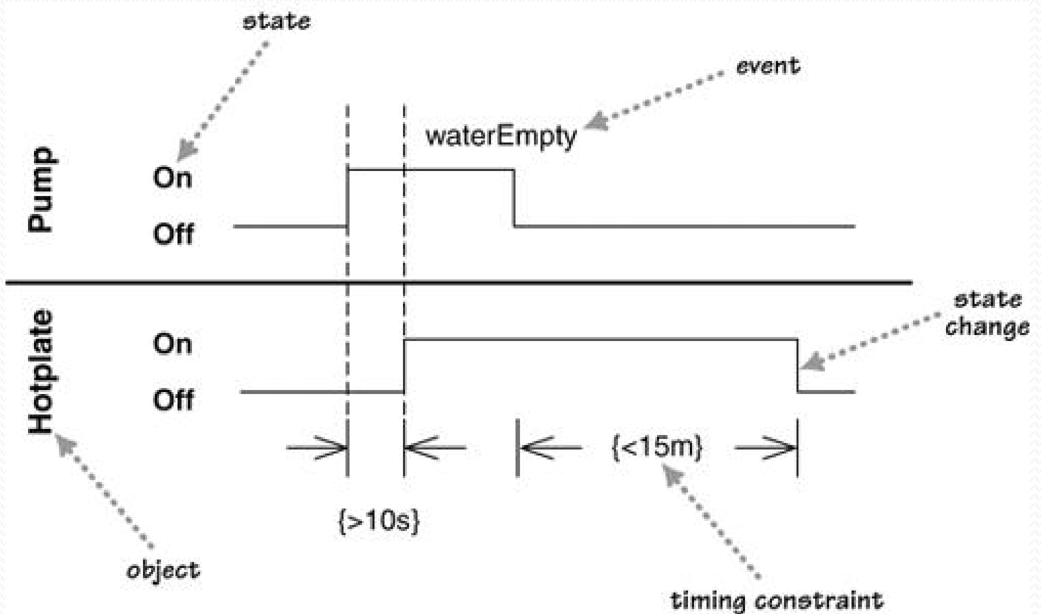
Destruction Occurrence

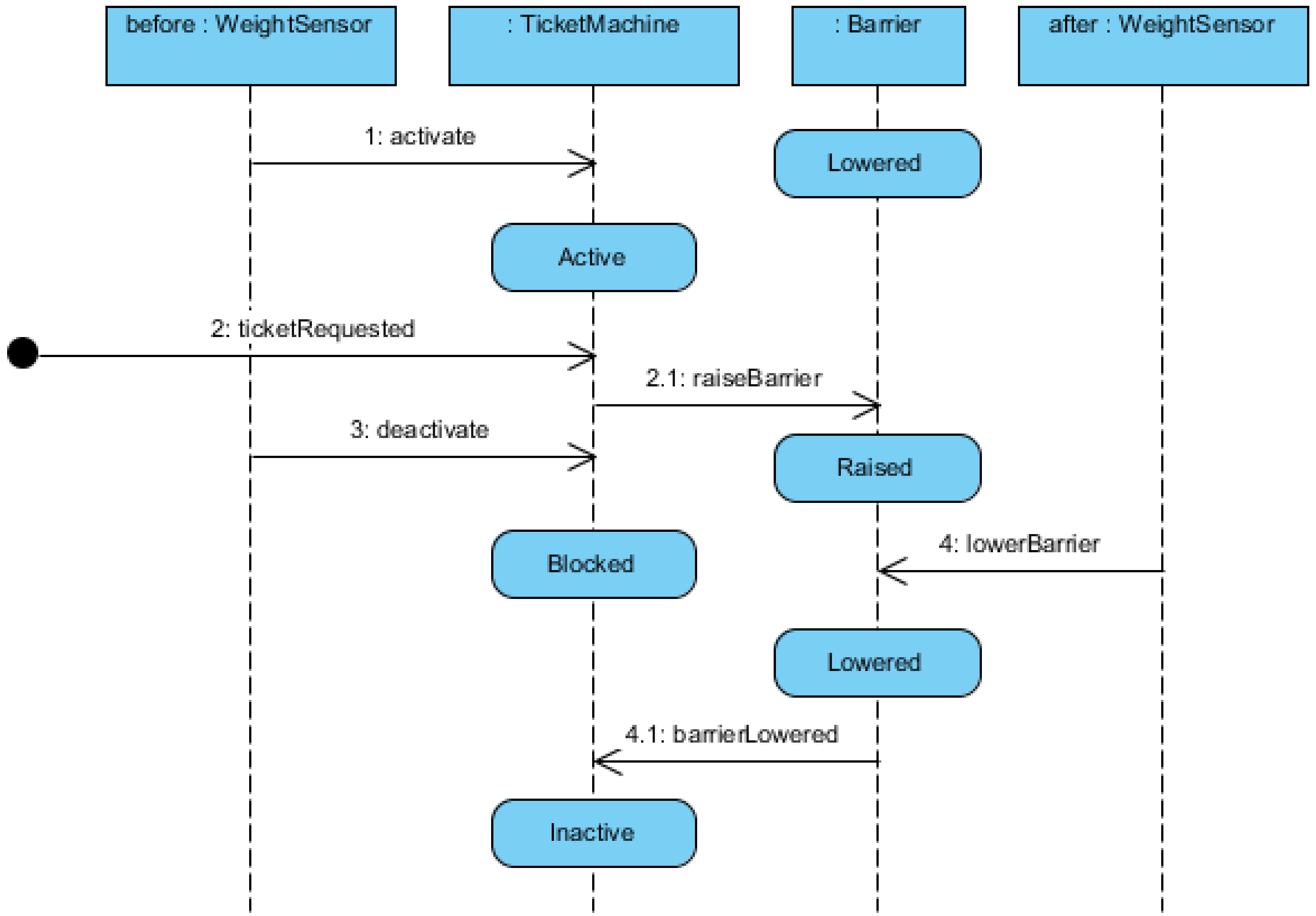
- **Destruction occurrence** is a **message occurrence** which represents the destruction of the instance described by the **lifeline**.
- It may result in the subsequent destruction of other objects that this object owns by **composition**.
- No other occurrence may appear after the destruction event on a given lifeline.
- ***Notation***
- The destruction event is depicted by a cross in the form of an **X** at the end of a timeline.

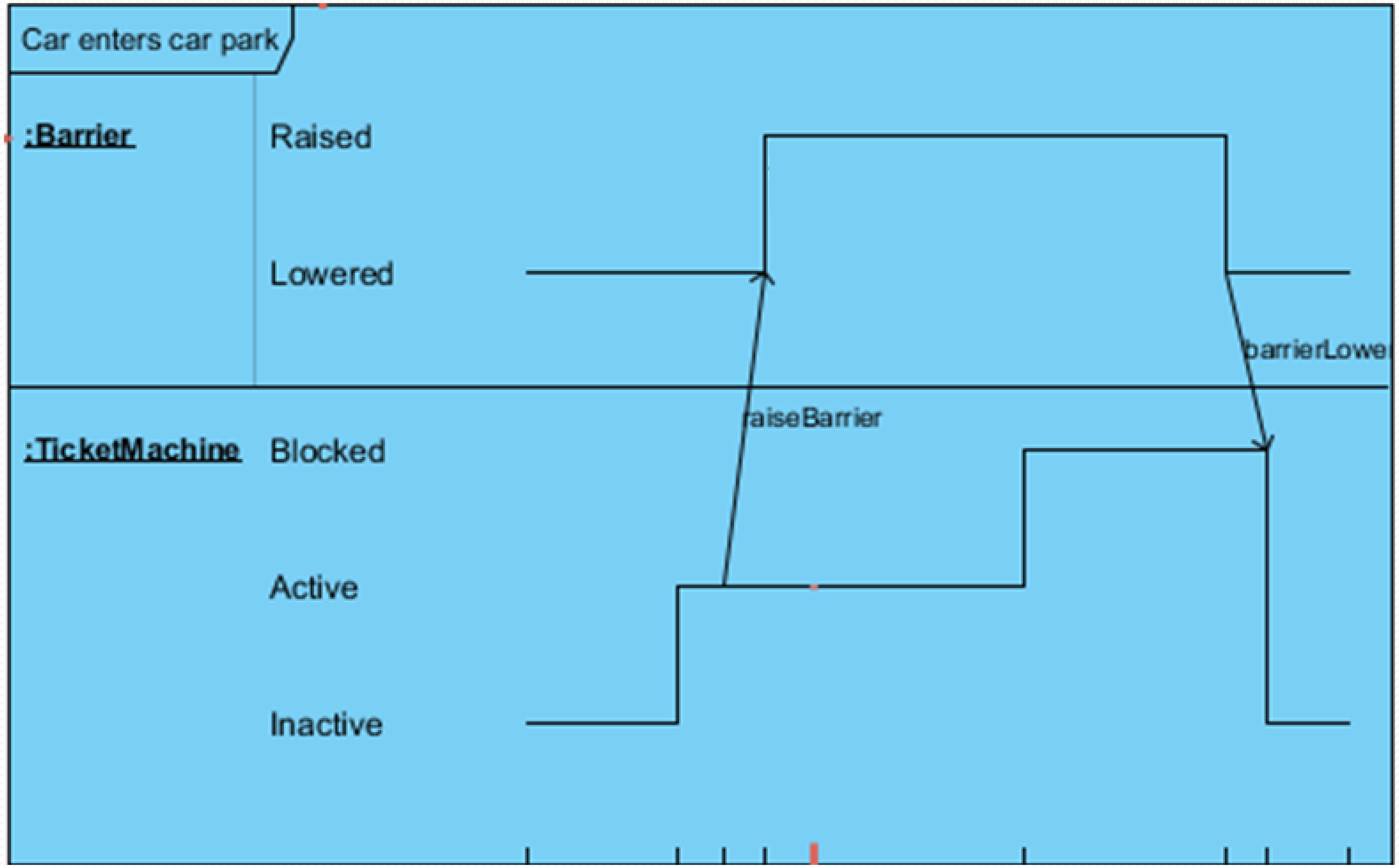


Coffee pot

- Let's take a simple scenario based on the pump and hotplate for a coffee pot. Let's imagine a rule that says that at least 10 seconds must pass between the pump coming on and the hotplate coming on. The water reservoir becomes empty after 6 minutes and the pump switches off, and the hotplate cannot stay on for more than 15 minutes.




before : WeightSensor
: TicketMachine
: Barrier
after : WeightSensor
1: activate
Lowered
Active
2: ticketRequested
2.1: raiseBarrier
3: deactivate
Raised
4: lowerBarrier
Blocked
Lowered
4.1: barrierLowered
Inactive


Car enters car park
:Barrier
Raised
Lowered
barrierLowe
raiseBarrier
:TicketMachine
Blocked
Active
Inactive

Example 1

- Two active objects share a common resource. In this case, both objects show a trade that requires the execution engine for some time. One object is for a "platinum" user who is guaranteed a trade within 10 time units, and the other is a "gold" user who has no performance guarantee. The user objects have a waiting and executing state along with an idle state. The execution engine has a processing state that takes five time units for each trade and an executing state that takes two time units.

Ex 2: Website Latency

- A fabricated website evaluates how long web user should wait to see something rendered on his/her display.
- After web user enters web page URL, the URL should be resolved to some IP address. DNS resolution can add some tangible waiting time to the response latency as perceived by user. Latency delays related to DNS resolution could range from 1 ms (local DNS cache) to several seconds.

Ex 2: Website Latency

- With simple Model–View–Control (MVC) implementation, Java servlet gets control and requests some data from "model". Communication with data sources usually takes some discernible time. After data is received and processed, servlet forwards request processing to JSP ("view"). Buffered HTTP response is sent back to the browser.
- Web browser takes some time to process HTTP response and HTML page to start rendering the page view to the web client. (Note, that after that it could take even more time for the web browser to request other resources like CSS, JavaScript, images, which is not shown on the diagram.)