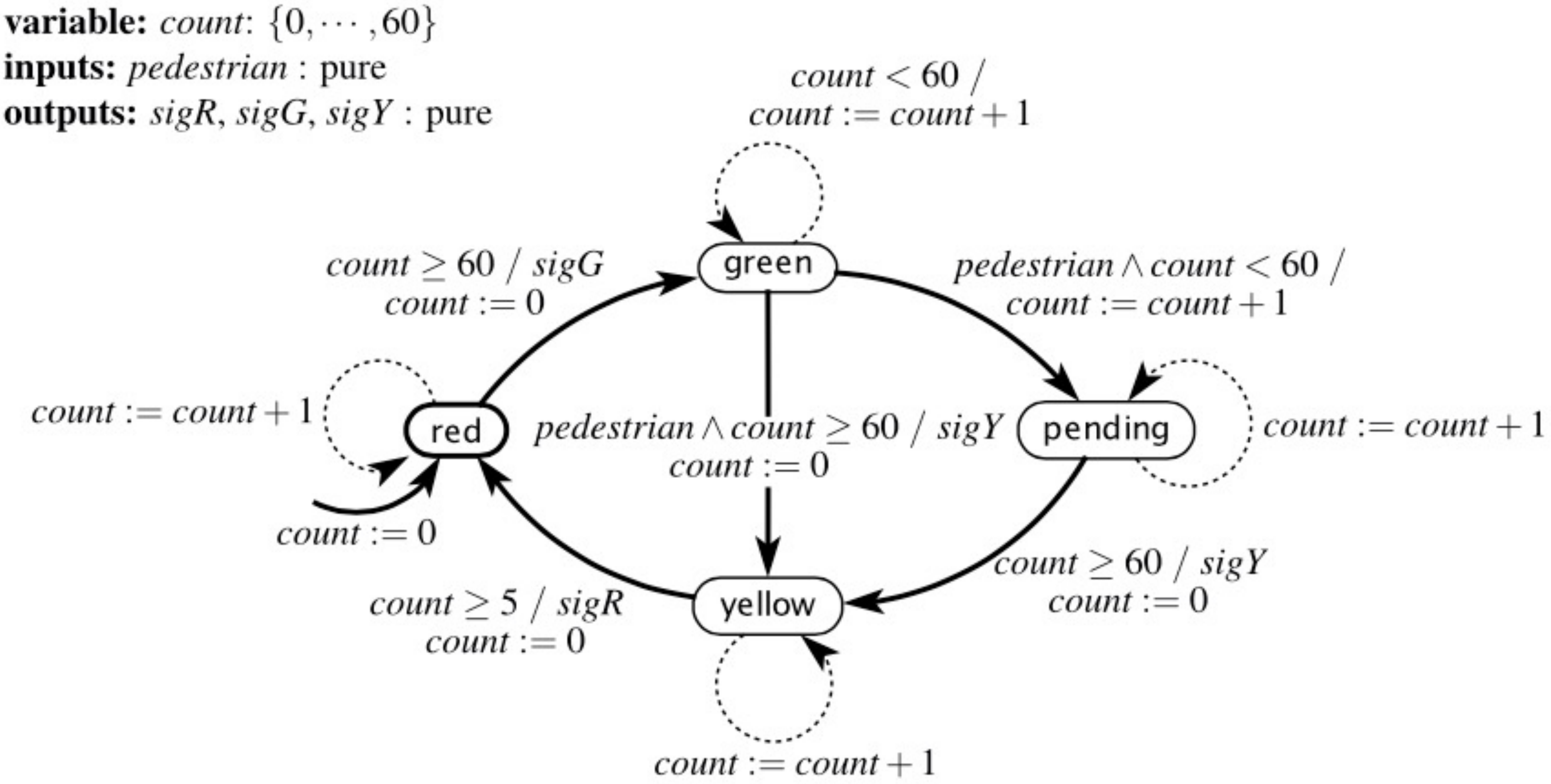
Cyber-Physical Systems

Laura Nenzi

Università degli Studi di Trieste
I Semestre 2023

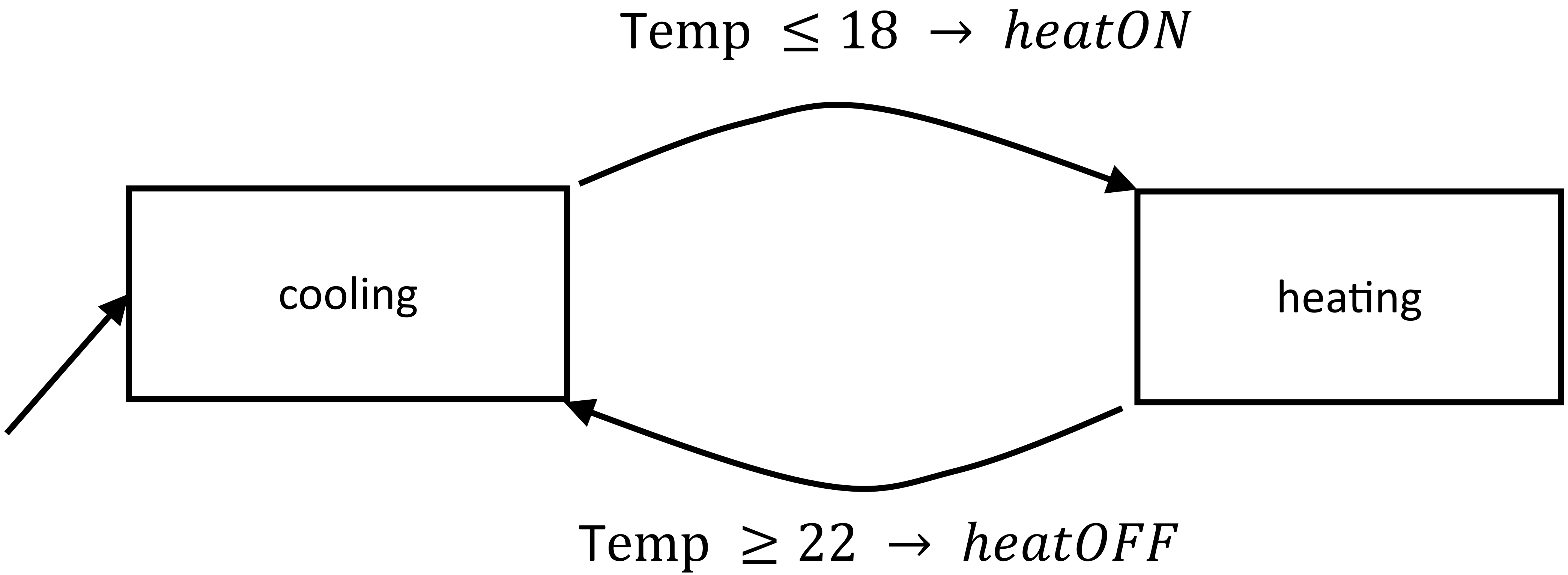
Lecture 4: Timed Models

Time Trigger Machine



variable: *count*: $\{0, \cdots, 60\}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure

Thermostat FSM



$\text{Temp} \leq 18 \rightarrow \mathit{heatON}$

cooling

heating

$\text{Temp} \geq 22 \rightarrow \mathit{heatOFF}$

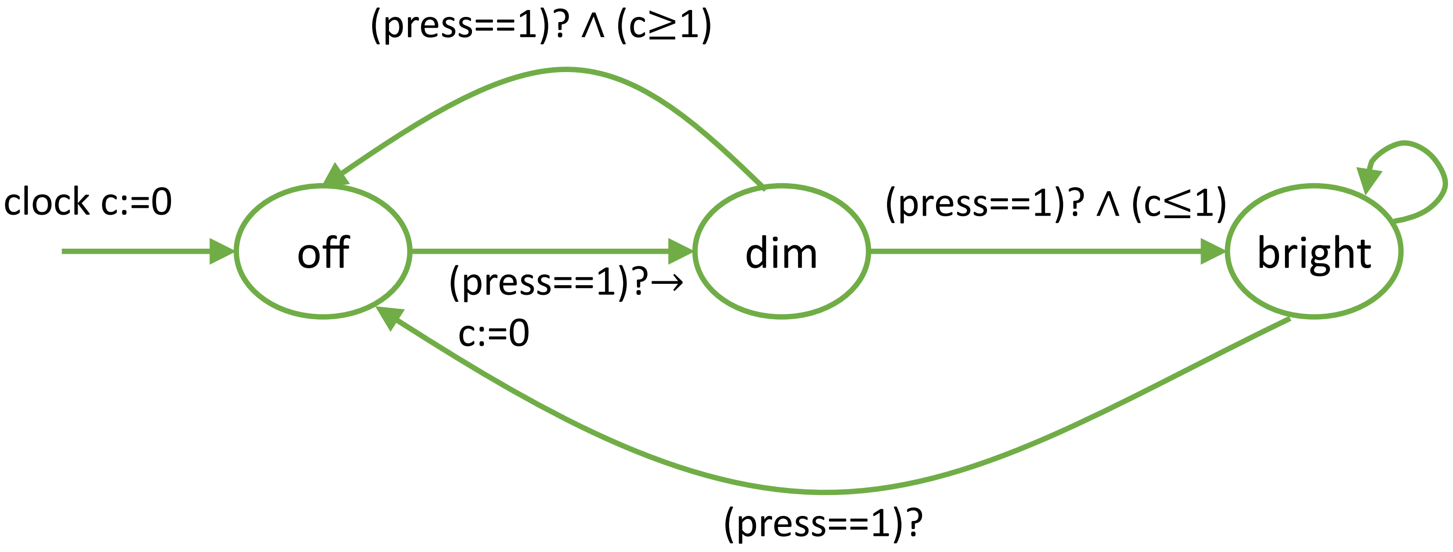
It could be event triggered, like the garage counter, in which case it will react whenever a *temperature* input is provided. Alternatively, it could be time triggered, meaning that it reacts at regular time intervals

Timed Models

- Like Asynchronous models, but with explicit time information

- Can make use of global time for coordination

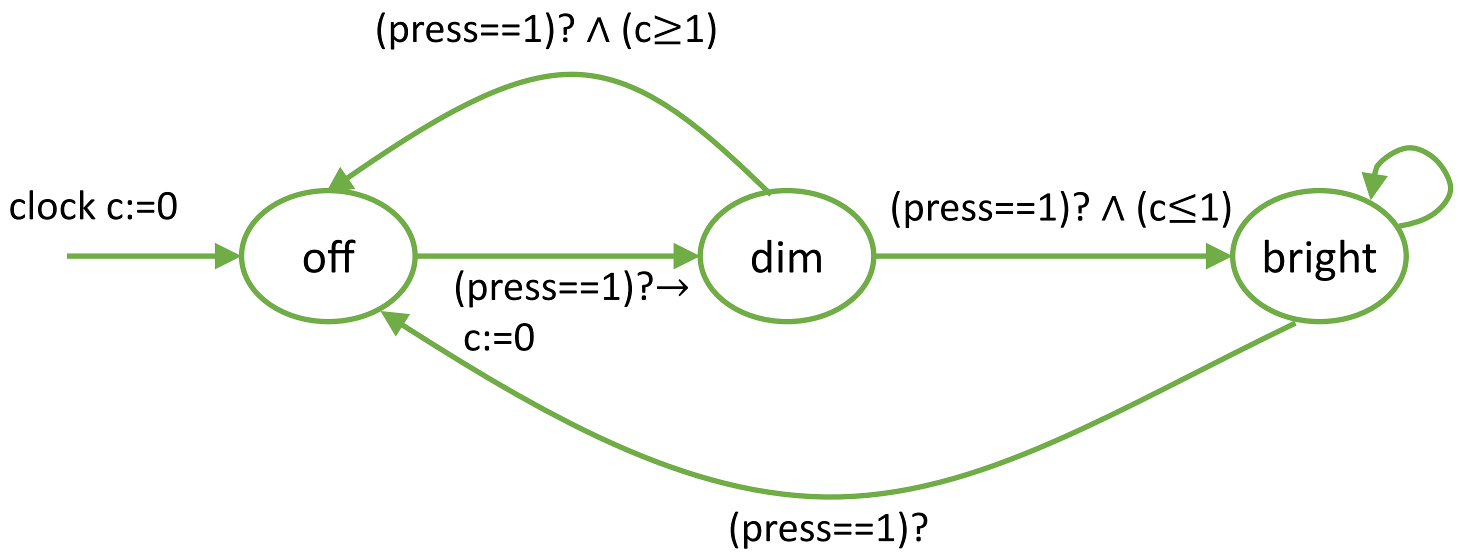
Timed ESMs: a Light Switch



Like asynchronous ESMs, have input, output channels, state variables

- Special type of state variable called “clock”
- Clock variables evolve continuously in time
- ESM can “stay” in a mode with clock increasing monotonically from the start value

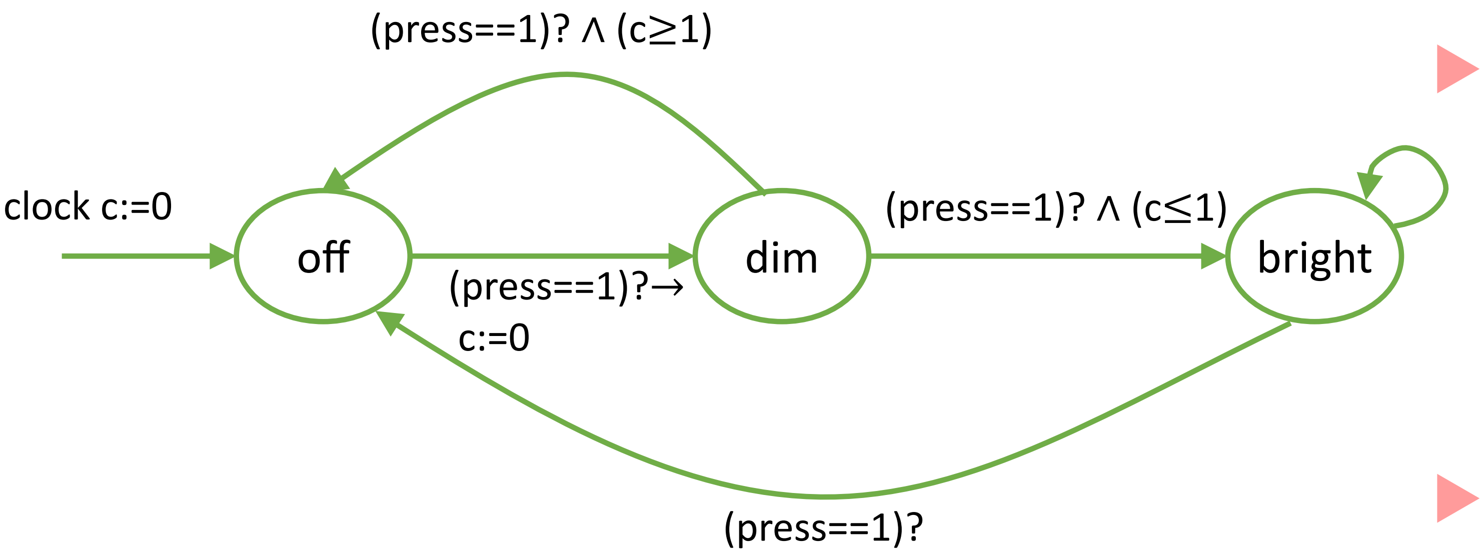
Transitions of a timed ESM



- Mode switch: discrete action
  - machine moves from one mode to another
  - guard on the transition must be true for mode switch to occur
  - update specified by the transition will update/reset clock variables

$$(\text{off},0.5)\xrightarrow{(\text{press}==1)?}(\text{dim},0)$$

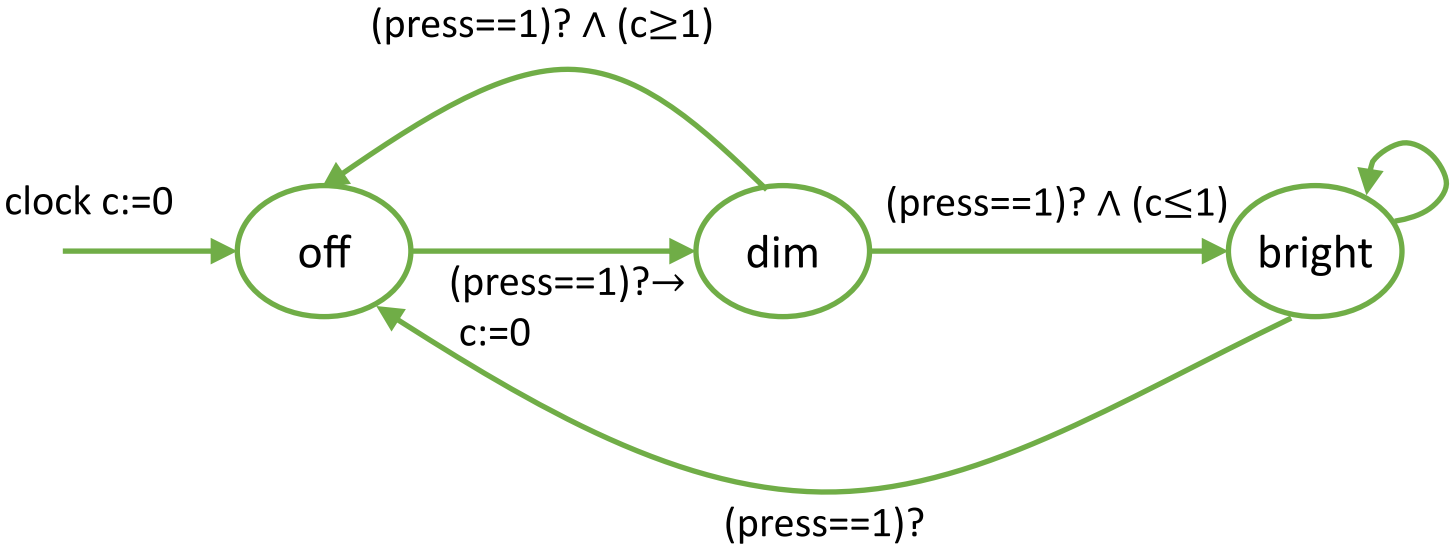
Transitions of a timed ESM



In a mode: Timed action

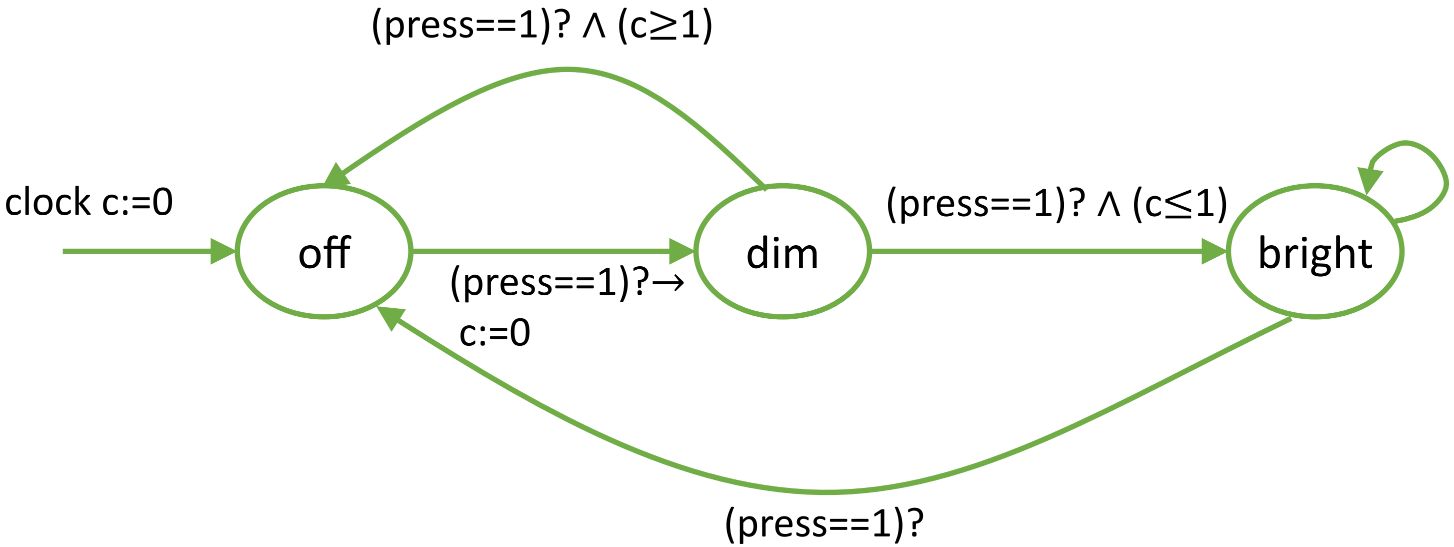
- When machine stays in any given mode for time δ, each clock variable increases by δ and all other state variables remain unchanged
- Captures timing constraints
  - Resetting c to 0 from off→dim and guard c≥1 from dim→off specifies that these mode switches are ≥1 second apart

Timed Processes: explicit clock variables

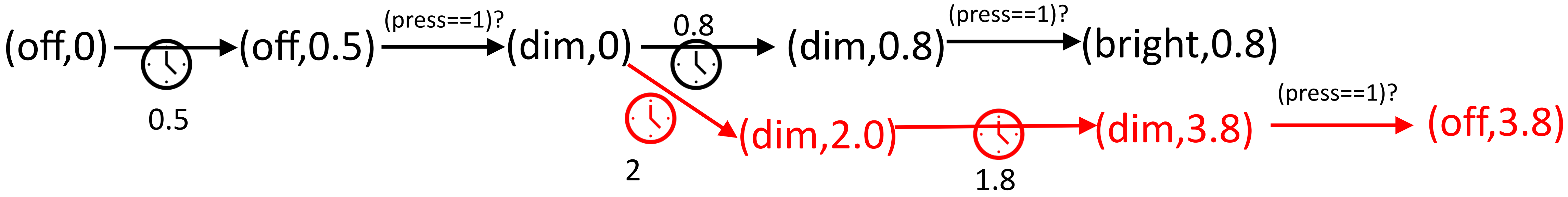


- Clock variables
  - Like other state variables, can be used in guards
  - Can be reset to 0 during mode switches
  - When the machine is in a given mode for duration δ, the clock variable increases by δ

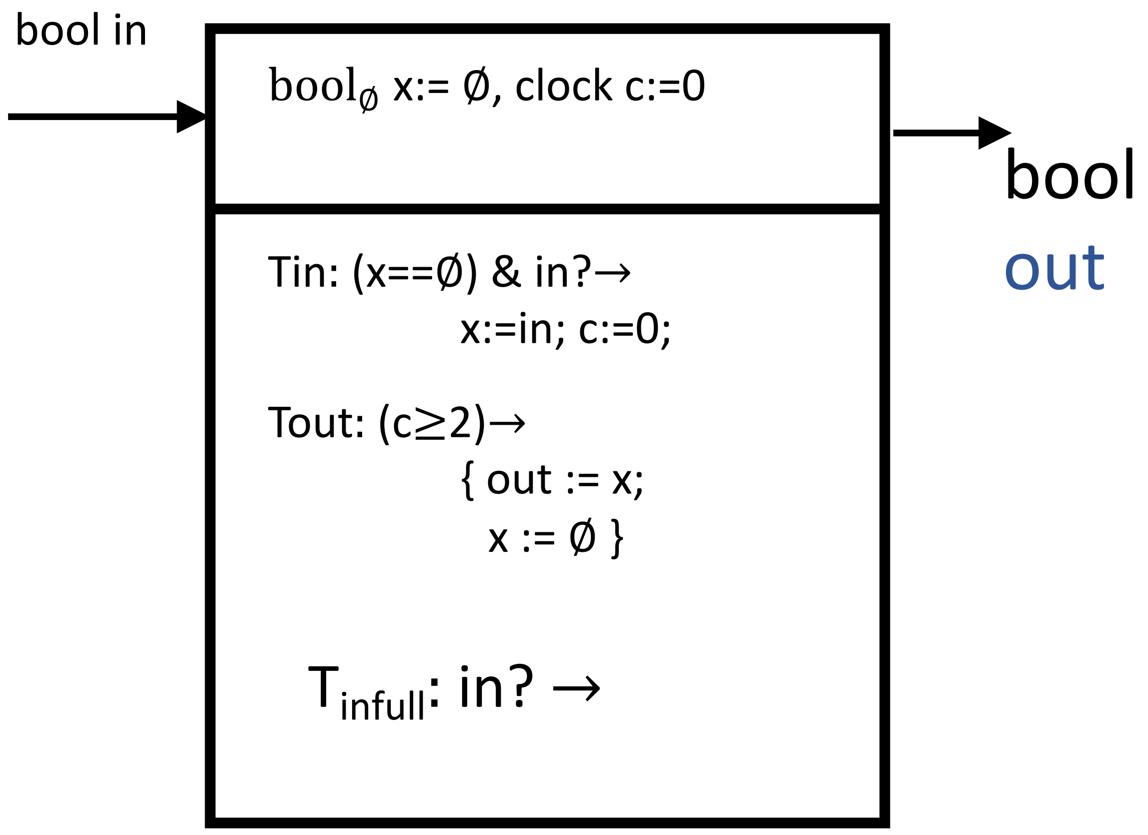
Timed Process Execution



- Machine execution is through alternating timed transitions and mode switches



Timed Buffer



bool in

$\text{bool}_{\emptyset}$ x:= ∅, clock c:=0

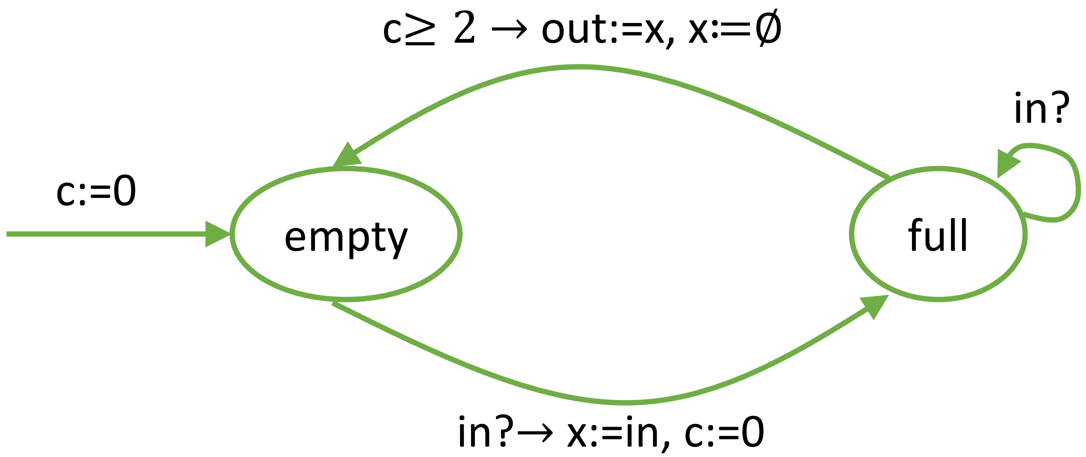
```
Tin: (x==∅) & in?→
          x:=in; c:=0;

Tout: (c≥2)→
          { out := x;
            x := ∅ }
```

T_{infull}: in? →

bool out

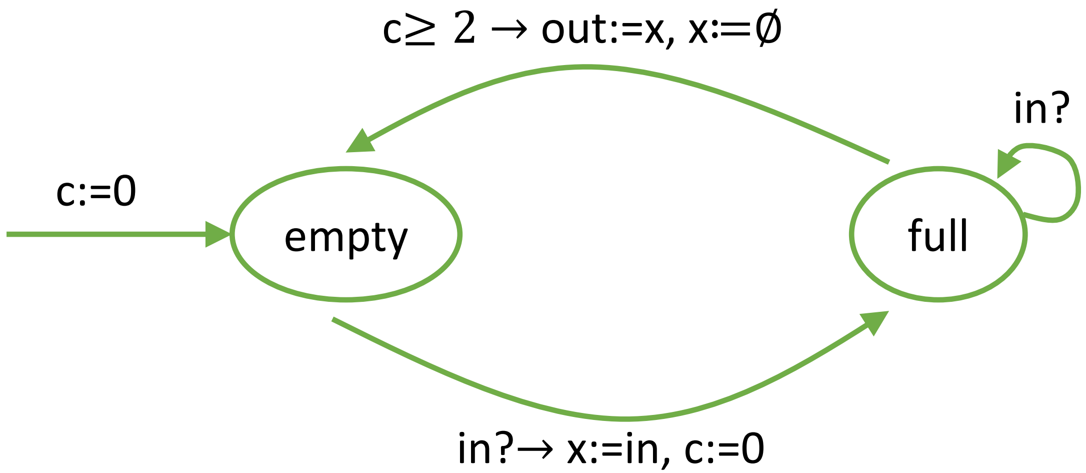
- Input channel in of type bool
- Output channel out of type bool
- State variable x of type bool+∅. The value ∅ indicates empty
- If x is ∅, then read new value into x, and set clock to 0
- If clock value is ≥ 2 seconds, output value of x, and set x to ∅

Timed State Machine representation



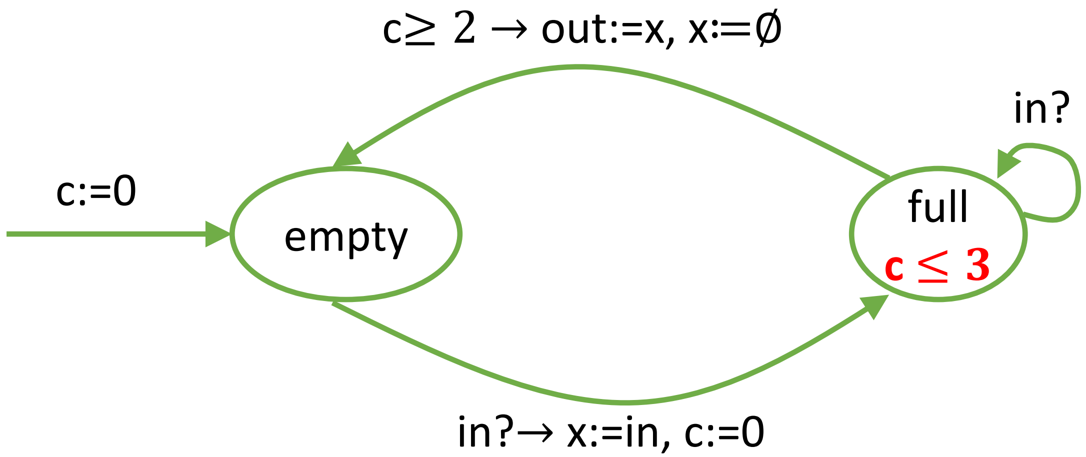
- Mode captures whether x==∅
- Clock variable tracks the time that elapsed since x received a value
- Guard ensures that at least 2 seconds pass before the value of x is output
- Guard ***does not force*** transitions
  - c can keep increasing while process remains in mode full
- How do we make sure that process does not remain in full mode for at most 3 seconds?

Clock invariants



- Attempt 1: we could make the guard $2 \leq c \leq 3$
- Attempt 1 fails because:
  - You could keep getting new input (self-loop executes) till $c \geq 3$
- Larger problem: Guards are non-forcing: nothing requires the guard to be executed
- We can fix this by introducing **clock invariants**
- Clock invariant of any mode: symbolic expression that must evaluate to true at all times, and if not, the process must exit that mode

Clock invariants



- Add clock invariant:

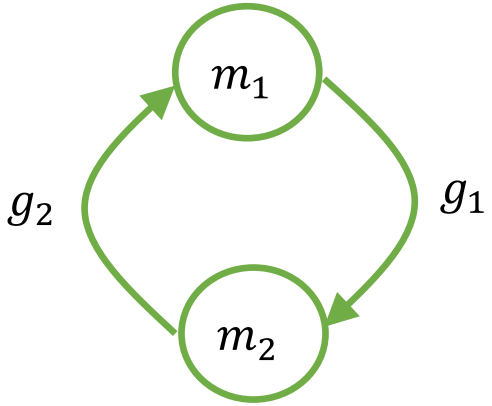
  (mode==full) ⇒ (c ≤ 3)
- Forces process to leave mode full if c becomes greater than 3
- Staying in mode full when c≥ 3 would violate the clock invariant
- Useful construct to limit how long a process stays in a certain mode

Why model using invariants and guards?

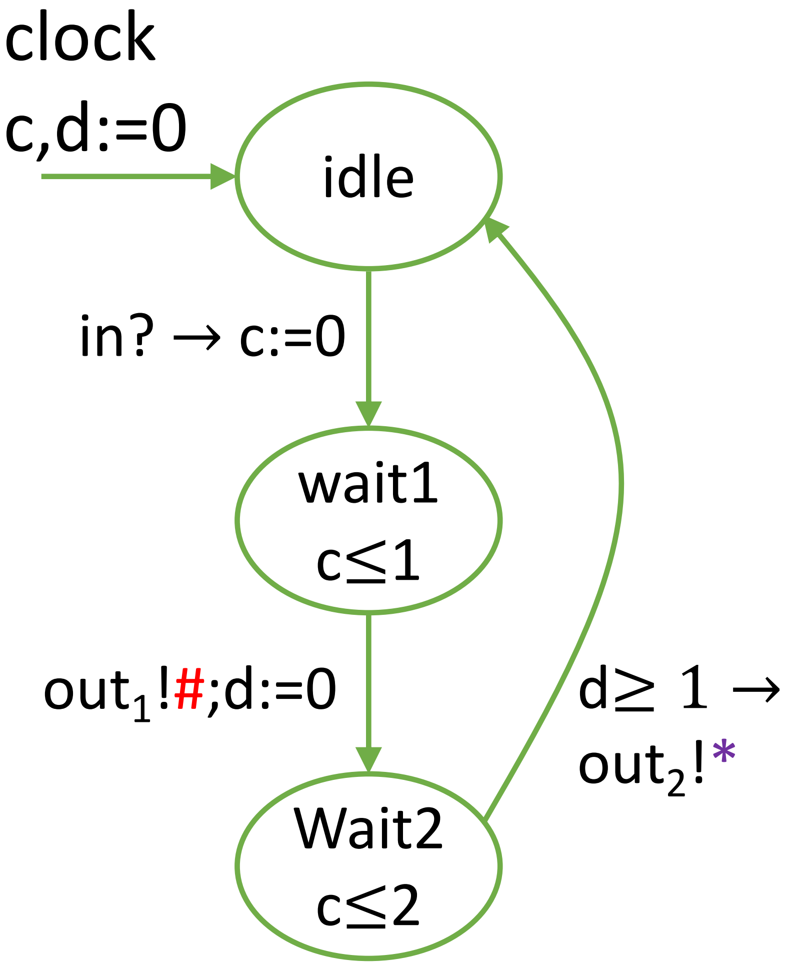
- Each mode is a guard-enabled task; if guard is true, task is executed
  - Going from one mode to another is a task switch
- Checking if process leaves mode m_1 and goes to $m_2$depends on if incoming *guard of m_2 is true*
- Staying in $m_1$does not/should not depend on the *guard* of m_1
- So, we use invariants:
  - a condition always checked in a given mode

mode $m \coloneqq m_1$, clock c

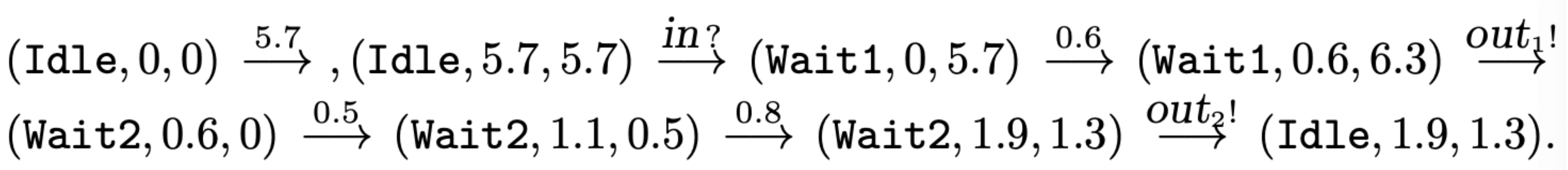
$T_{1\to 2}$: $\mathrm{g}_1(\mathrm{c}) \wedge (m = m_1) \to$
$m \coloneqq m_2$
$T_{2\to 1}$: $\mathrm{g}_2(\mathrm{c}) \wedge (m = m_2) \to$
$m \coloneqq m_1$



Example with two clocks



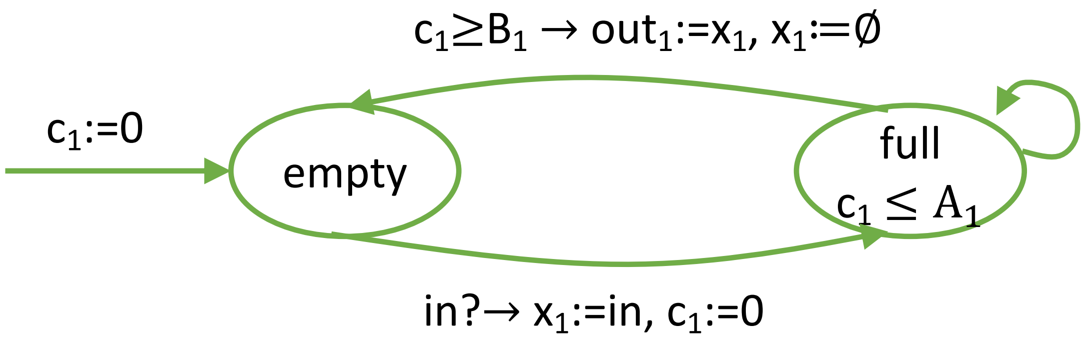
- Model with one input channel and two output channels: out_1 and out_2
- Clock c tracks time elapsed since occurrence of the input task execution
- Clock d tracks time elapsed since occurrence of output task for out_1
- Behavior of process: If input event occurs at some time t, then process issues output # on out_1 some time t' ∈ [t,t+1] and then issues output * on out_2 at time t'' ∈ [t'+1, t+2]

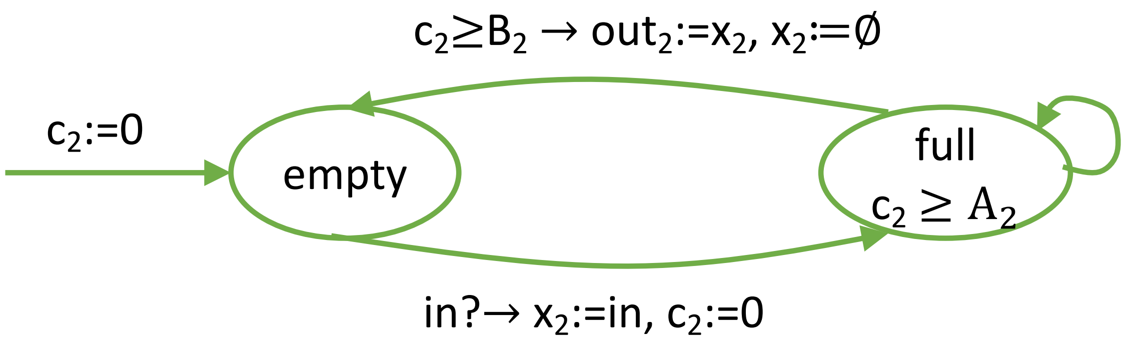


$$(\texttt{Idle},0,0) \xrightarrow{5.7} ,(\texttt{Idle},5.7,5.7) \xrightarrow{in?} (\texttt{Wait1},0,5.7) \xrightarrow{0.6} (\texttt{Wait1},0.6,6.3) \xrightarrow{out_1!}$$

$$(\texttt{Wait2},0.6,0) \xrightarrow{0.5} (\texttt{Wait2},1.1,0.5) \xrightarrow{0.8} (\texttt{Wait2},1.9,1.3) \xrightarrow{out_2!} (\texttt{Idle},1.9,1.3).$$

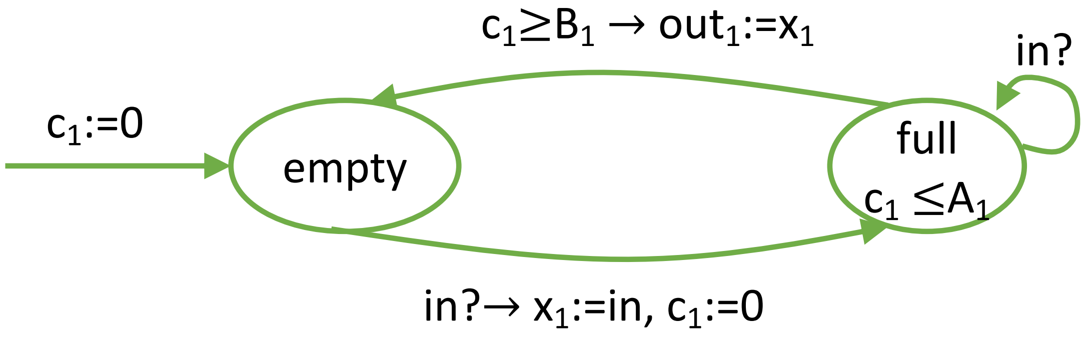
Composing Timed Processes

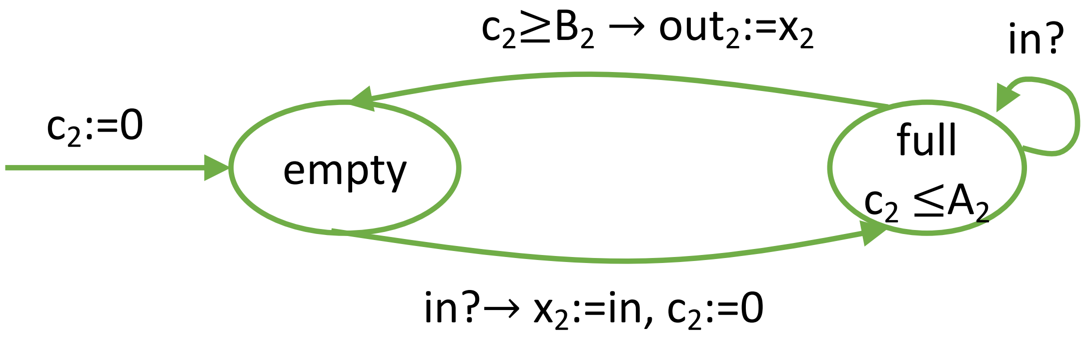

$c_1 \geq B_1 \rightarrow out_1 := x_1, x_1 := \emptyset$
$c_1 := 0$
empty
full
$c_1 \leq A_1$
in?→ $x_1 := in, c_1 := 0$

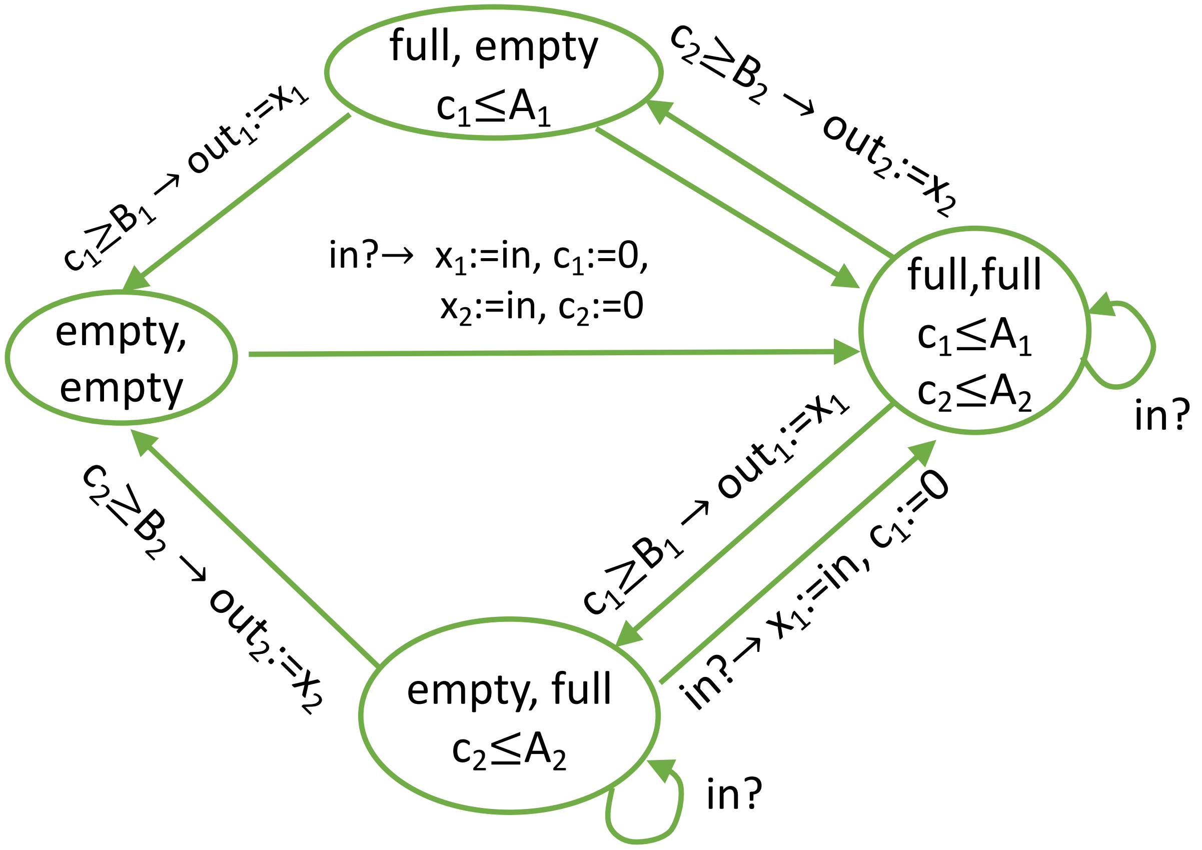

$c_2 \geq B_2 \rightarrow out_2 := x_2, x_2 := \emptyset$
$c_2 := 0$
empty
full
$c_2 \geq A_2$
in?→ $x_2 := in, c_2 := 0$

- Each process stays in mode full for $t \in [B_i, A_i]$
- Need to construct a new process with 4 new modes
- Each new mode is a pair consisting of modes from process 1 and 2
- Mode switches in the new machine correspond to mode switches in the old machine
- Interesting timing behavior can arise!

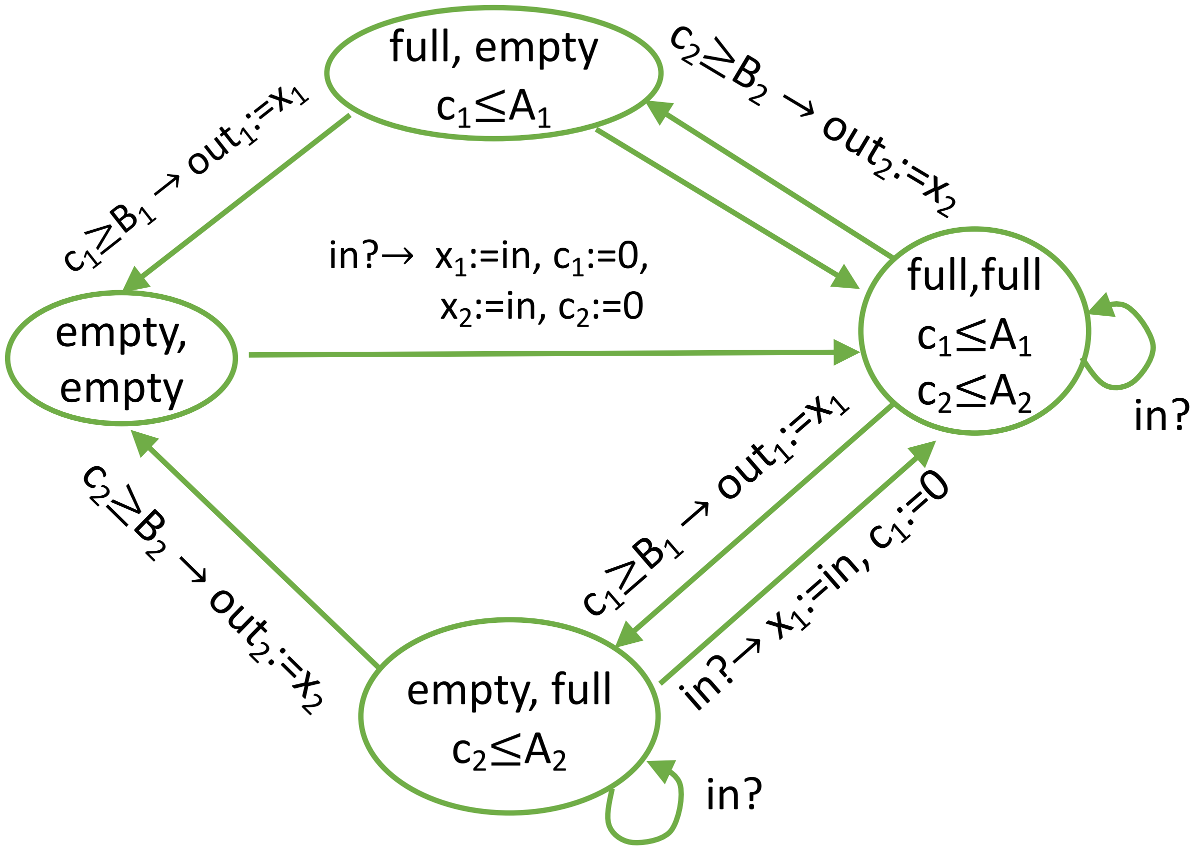
Composing Timed Processes


$c_1 \geq B_1 \rightarrow out_1 := x_1$
in?
$c_1 := 0$
empty
full
$c_1 \leq A_1$
in?→ $x_1 := in,\ c_1 := 0$


$c_2 \geq B_2 \rightarrow out_2 := x_2$
in?
$c_2 := 0$
empty
full
$c_2 \leq A_2$
in?→ $x_2 := in,\ c_2 := 0$


full, empty
$c_1 \leq A_1$
$c_2 \geq B_2 \rightarrow out_2 := x_2$
$c_1 \geq B_1 \rightarrow out_1 := x_1$
in?→ $x_1 := in,\ c_1 := 0,$
$x_2 := in,\ c_2 := 0$
empty,
empty
full,full
$c_1 \leq A_1$
$c_2 \leq A_2$
in?
$c_2 \geq B_2 \rightarrow out_2 := x_2$
$c_1 \geq B_1 \rightarrow out_1 := x_1$
in?→ $x_1 := in,\ c_1 := 0$
empty, full
$c_2 \leq A_2$
in?

Semi-synchrony


full, empty
$c_1 \leq A_1$
$c_1 \geq B_1 \rightarrow out_1 := x_1$
$c_2 \geq B_2 \rightarrow out_2 := x_2$
in?→ $x_1 := in$, $c_1 := 0$,
$x_2 := in$, $c_2 := 0$
empty,
empty
full,full
$c_1 \leq A_1$
$c_2 \leq A_2$
in?
$c_1 \geq B_1 \rightarrow out_1 := x_1$
in?→ $x_1 := in$, $c_1 := 0$
$c_2 \geq B_2 \rightarrow out_2 := x_2$
empty, full
$c_2 \leq A_2$
in?

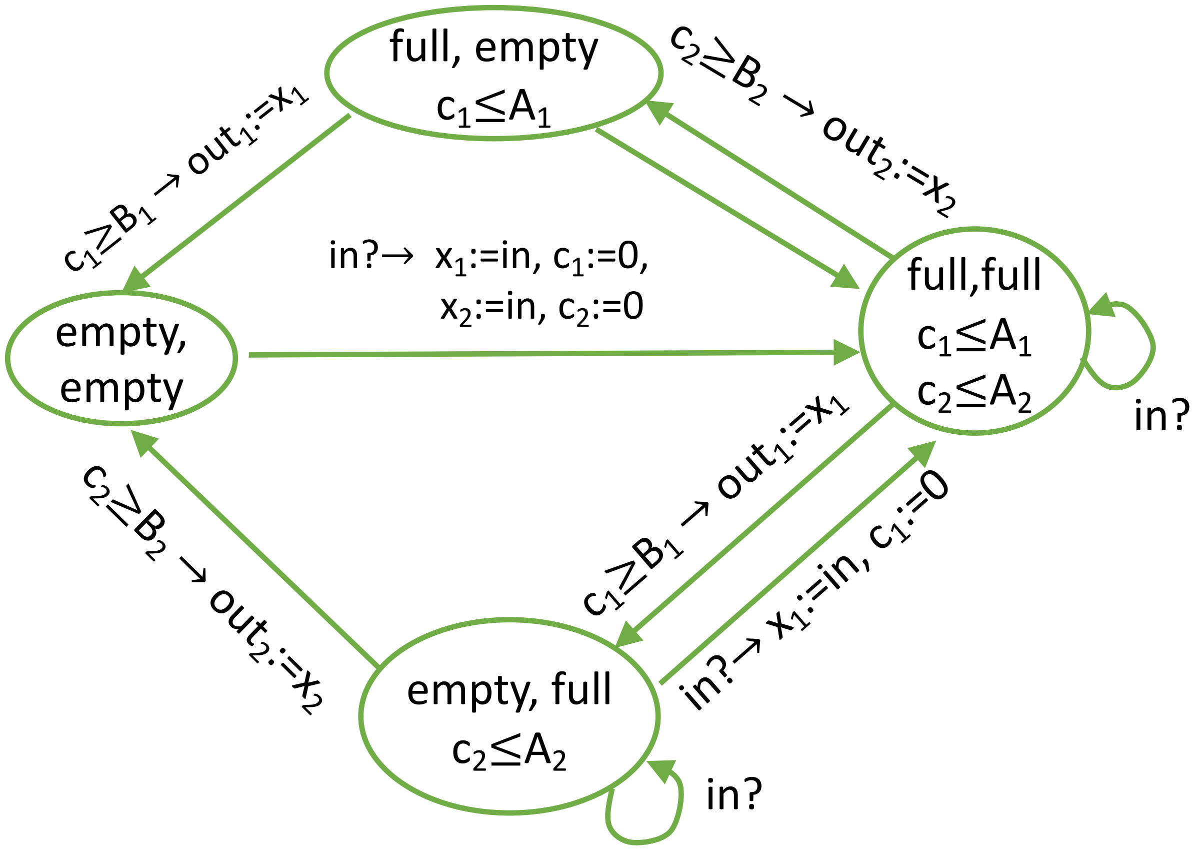
If $B_1 < A_1 < B_2$:

- (full,full) →(full,empty) can never be enabled!

Why?

- c_1 reaches A_1 and the process gets kicked out of state (full,full)
- But c_1 cannot be greater than B_2 so, guard from (full,full) to (full,empty) is not enabled!

Semi-synchrony


full, empty
$c_1 \leq A_1$
empty, empty
full,full
$c_1 \leq A_1$
$c_2 \leq A_2$
empty, full
$c_2 \leq A_2$
$c_1 \geq B_1 \to out_1 := x_1$
$c_2 \geq B_2 \to out_2 := x_2$
in?→ x_1:=in, c_1:=0,
x_2:=in, c_2:=0
in?
$c_1 \geq B_1 \to out_1 := x_1$
in?→ x_1:=in, c_1:=0
$c_2 \geq B_2 \to out_2 := x_2$
in?

- If $B_1 < A_1 < B_2$:
  - (full,full) → (full,empty) cannot happen
- If $B_1 < A_1 < B_2$:
  - (full, full) → (empty,full) will happen eventually
- **out_1 guaranteed to happen before out_2**

- Implicit coordination based on delays
  - Both process clocks increase in tandem
  - Global clock-based synchronization
- Reason why timed models are called semi-synchronous or partially synchronous

Formal recap of a timed process

- Timed process consists of:
  - An asynchronous process, where some of the state variables are of type clock (ranging over non-negative reals)
  - A clock invariant I which is a Boolean expression over the state variables
- Inputs, Outputs, States, Initial states, Actions: Internal, Input and Output: same as for asynchronous processes
- Timed Action: Given a state q and time $\delta > 0$, action q $\xrightarrow{\delta}$q' specifies a transition of duration δ if:
  - q' represents a state where the non-clock variables have the same value as in q, i.e. q'(x) = q(x)
  - q' represents a state where the clock variables in q are incremented by δ, i.e. q'(c) = q(c) + δ, and
  - For all times t ∈ [q(c), q(c)+δ], the clock invariant I is satisfied
  - If clock invariant is *convex*, enough to check clock invariant at q(c) and q(c)+δ

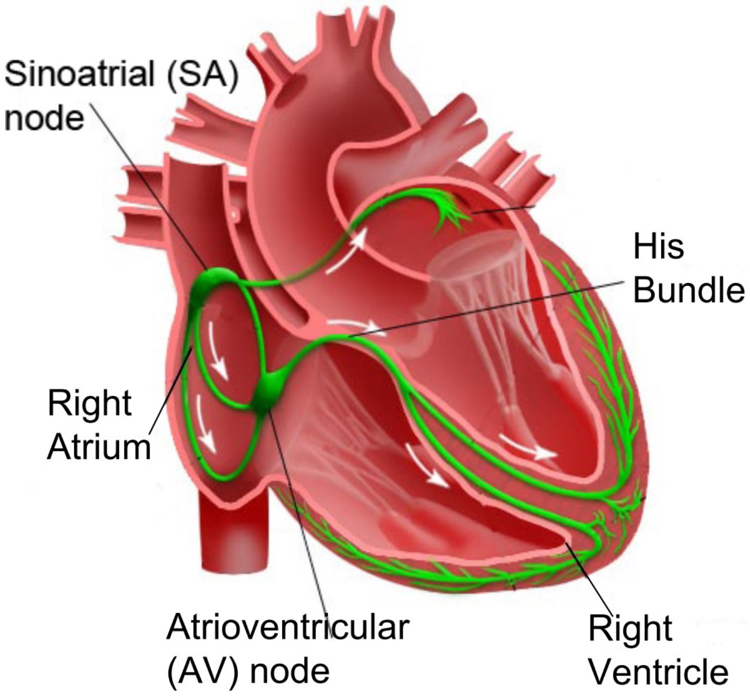
Pacemaker Modeling as a Timed Process

- Most material that follows is from this paper:

Z. Jiang, M. Pajic, S. Moarref, R. Alur, R. Mangharam, *Modeling and Verification of a Dual Chamber Implantable Pacemaker*, In Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2012.

- The textbook has detailed descriptions of some other pacemaker components

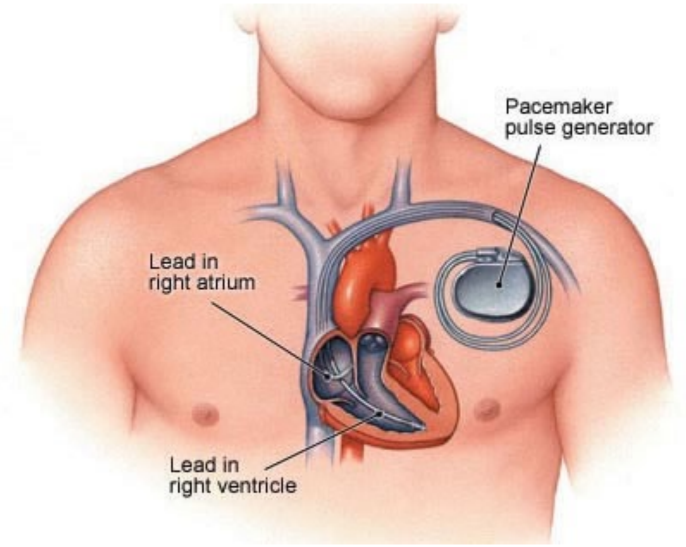
How does a healthy heart work?



Electrical Conduction System of the Heart

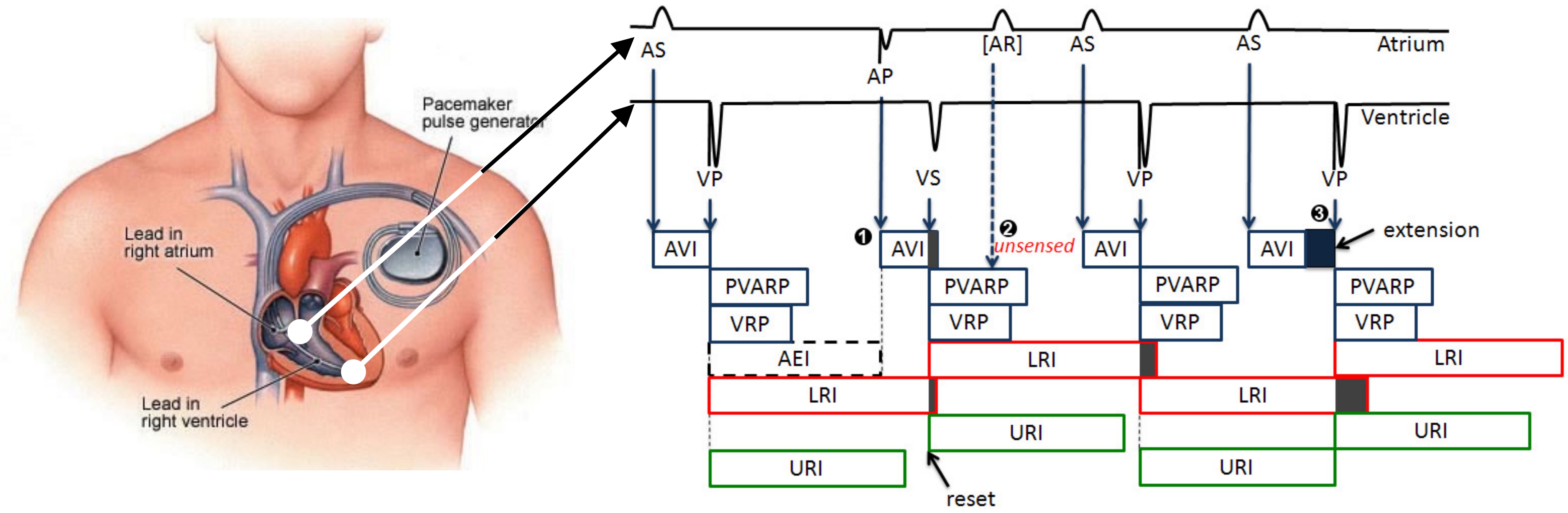
- SA node (controlled by nervous system) periodically generates an electric pulse
- This pulse causes both atria to contract pushing blood into the ventricles
- Conduction is delayed at the AV node allowing ventricles to fill
- Finally the His-Pukinje system spreads electric activation through ventricles causing them both to contract, pumping blood out of the heart

What do pacemakers do?



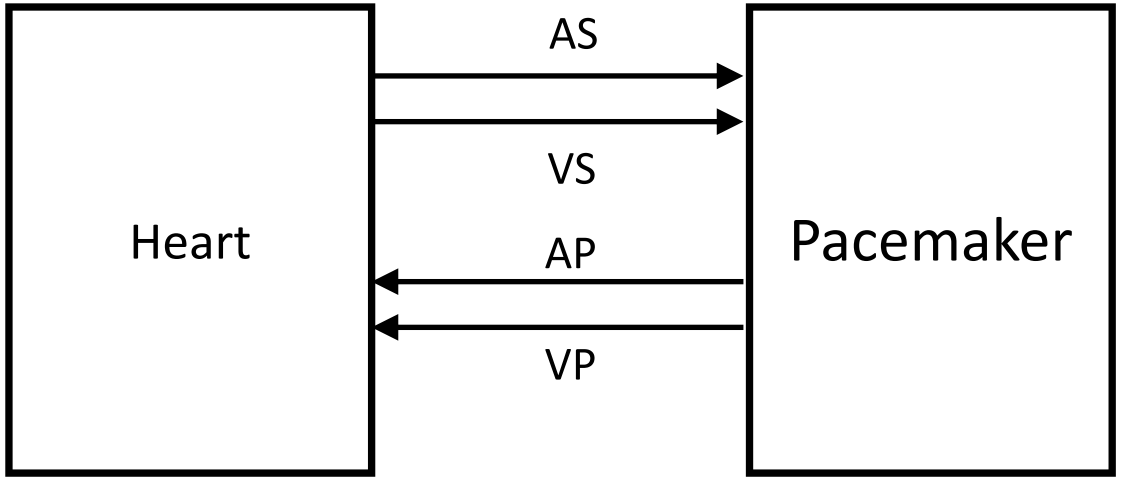
- Aging and/or diseases cause conduction properties of heart tissue to change leading to changes in heart rhythm
- Tachycardia: faster than desirable heart rate impairing hemo-dynamics (blood flow dynamics)
- Bradycardia: slower heart rate leading to insufficient blood supply
- Pacemakers can be used to treat bradycardia by providing pulses when heart rate is low

Implantable Pacemaker modeling



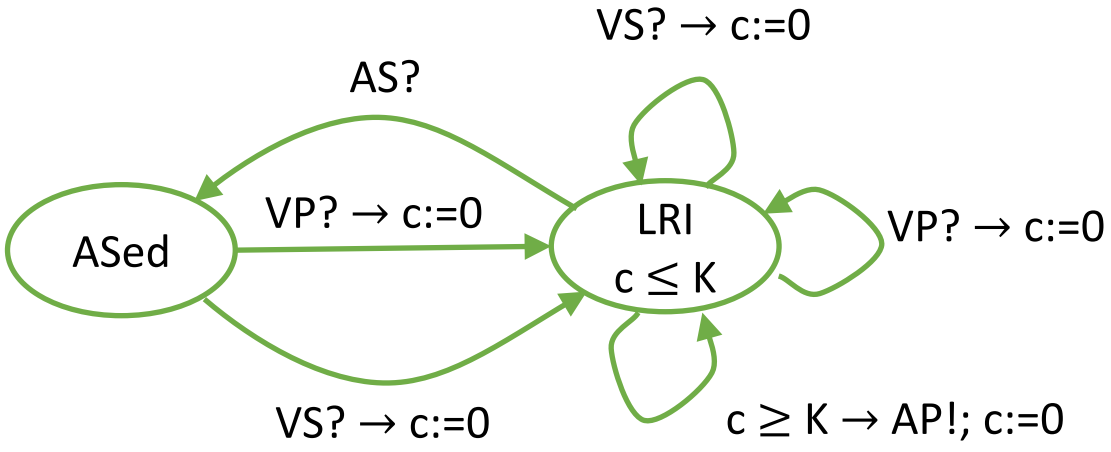
How dual-chamber pacemakers work

- Two fixed leads on wall of right atrium and ventricle respectively
- Activation of local tissue sensed by the leads (giving rise to events Atrial Sense (AS) and Ventricular Sense (VS))
- Atrial Pacing (AP) or Ventricular Pacing (VP) are delivered if no sensed events occur within deadlines



The Lower Rate Interval (LRI) mode

LRI component keeps heart rate above minimum level



K= 850ms

- LRI = lower rate interval
- LRI component keeps heart rate above minimum level
- One of the pacemaker modes of operation that models the basic timing cycle
- Measures the longest interval between ventricular events
- Clock reset when VS or VP received
- No AS received ⇒ LRI outputs AP after K time units

FSM Software Tools

- Statecharts (Harel, 1987), a notation for concurrent composition of hierarchical FSMs, has influenced many of these tools.
- One of the first tools supporting the Statecharts notation is STATEMATE (Harel et al., 1990), which subsequently evolved into Rational Rhapsody, sold by IBM.
- Almost every software engineering tool that provides UML (unified modeling language) capabilities (Booch et al., 1998).
- SyncCharts (André́ , 1996) is a particularly nice variant in that it borrows the rigorous semantics of Esterel (Berry and Gonthier, 1992) for composition of concurrent FSMs.
- LabVIEW supports a variant of Statecharts that can operate within dataflow diagrams
- Simulink with its Stateflow extension supports a variant that can operate within continuous-time models.
- UPPAAL (Yi, Pettersson, Larseń, mid-1990s) is is a tool for modeling, simulation, and verification of real-time systems. It was jointly developed by Uppsala University in Sweden and Aalborg University in Denmark.