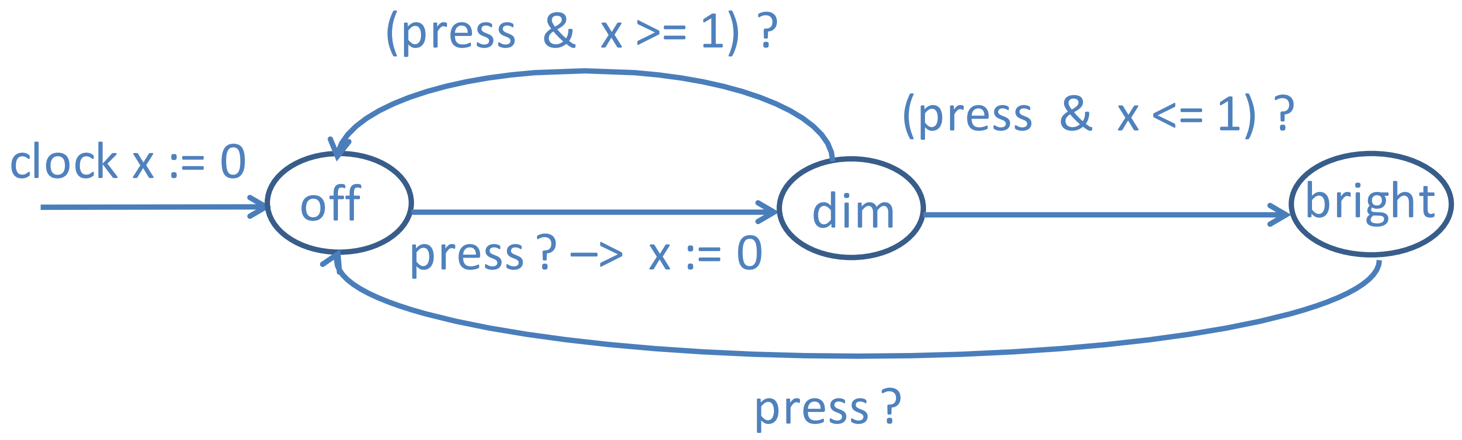
CS:4980
Foundations of Embedded Systems

Timed Model

Models of Reactive Computation

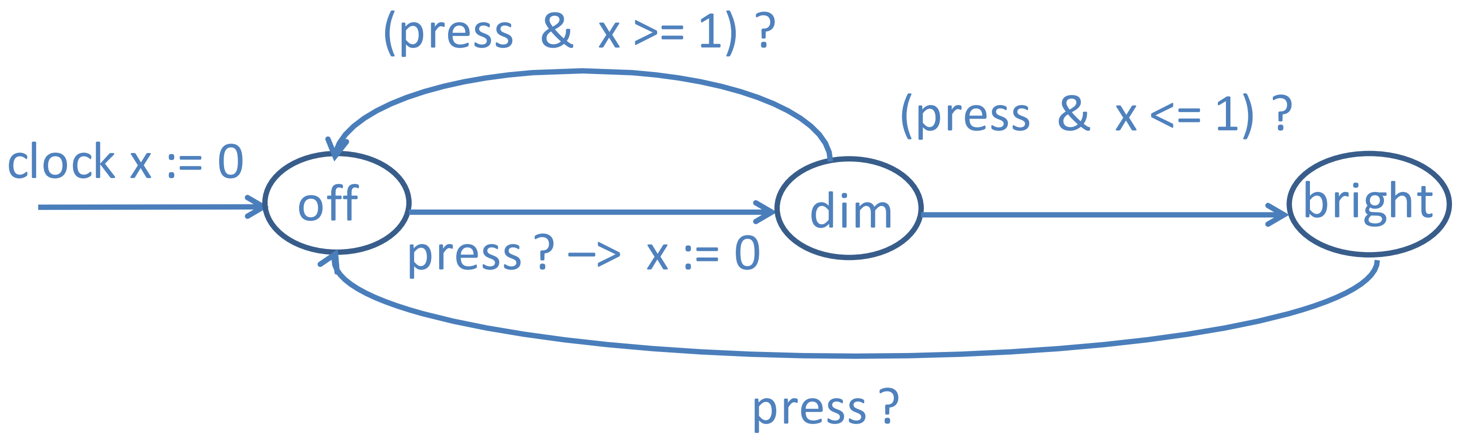
- Synchronous model
  - Components execute in a sequence of discrete rounds in lock-step
  - Computation within a round: Execute all tasks in an order consistent with precedence constraints
- Asynchronous model
  - Speeds at which different components execute are independent
  - Computation within a step: Execute a single task that is enabled
- Continuous-time model for dynamical system
  - Synchronous, but now time evolves continuously
  - Execution of system: Solution to differential equations
- Timed model
  - Like asynchronous for communication of information
  - Can rely on global time for coordination

Example Timed Model



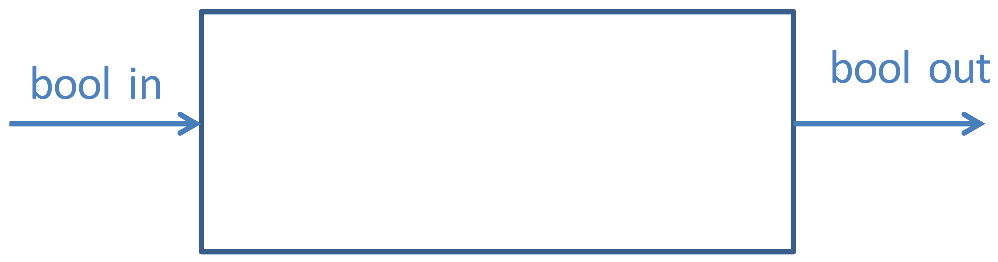
Initial state:	(mode = off, x = 0)
Timed transition:	(off, 0) –0.5–> (off, 0.5)
Input transition:	(off, 0.5) –press?–> (dim, 0)
Timed transition:	(dim, 0) –0.8–> (dim, 0.8)
Input transition:	(dim, 0.8) –press?–> (bright, 0.8)
Timed transition:	(dim, 0.8) –1–> (dim, 1.8)
Input transition:	(dim, 1.8) –press?–> (off, 1.8)

Example Timed Model



- Clock variables
  - Tests and updates in mode-switches like other variables
  - New: During a timed transition of duration d, the value of clock variables increases by an amount equal to d
- Timing constraint: Setting x to 0 for off –> dim and guard x <= 1 for dim –> bright specifies that timing of these two transitions is <= 1 apart

Example: Timed Buffer

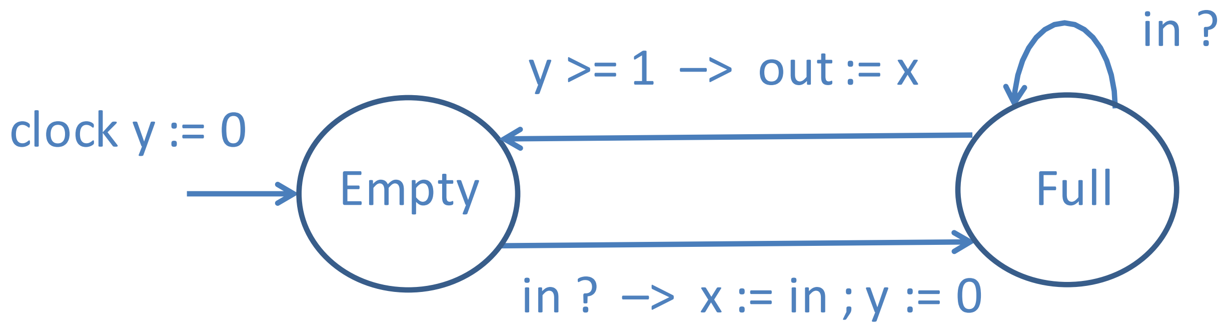


- Buffer with a bounded delay

- Behavior: Input received on channel in is transmitted on output channel out after a delay of d, with LB <= d <= UB

  (i.e. we know lower and upper bounds on this delay)

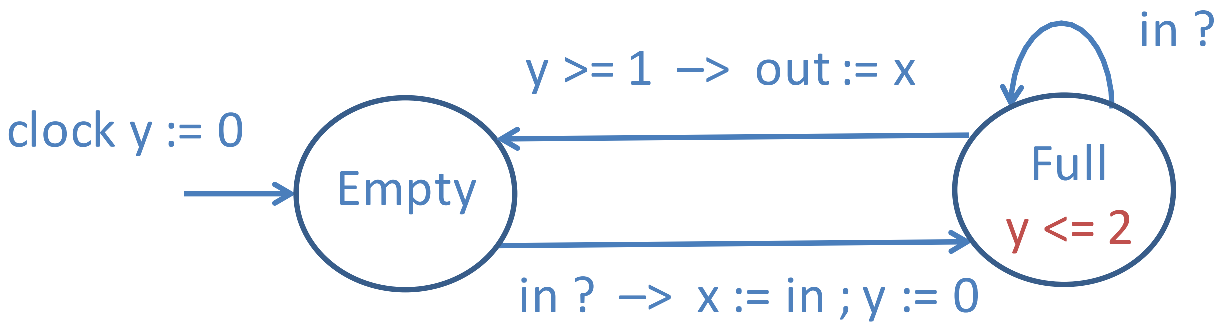
Modeling Timed Buffer



- ❑ Mode indicates whether the buffer is full or not
- ❑ State variable x remembers the last input value when buffer is full
- ❑ Clock variable y tracks the time elapsed since buffer filled up
- ❑ When buffer is full, input events are ignored
- ❑ Guard y >= 1 ensures that at least 1 time unit elapses in mode Full

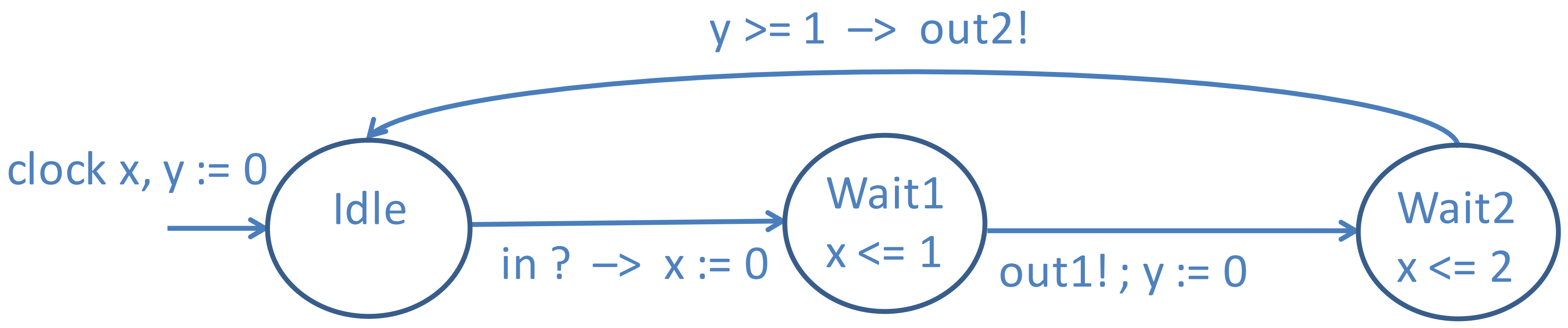
How to ensure that mode-switch from Full to Empty is executed before clock y exceeds the upper bound 1?

Clock Invariants



- The constraint y <= 1 associated with mode Full is a *clock invariant*
- A timed transition of duration d is allowed only if the clock invariant is satisfied for the entire duration of the transition
  - (Full, x, 0.8) –0.7–> (Full, x, 1.5)   allowed
  - (Full, x, 0.8) –1.4–> (Full, x, 2.2)   disallowed
- Clock invariants to limit how long a process stays in a mode

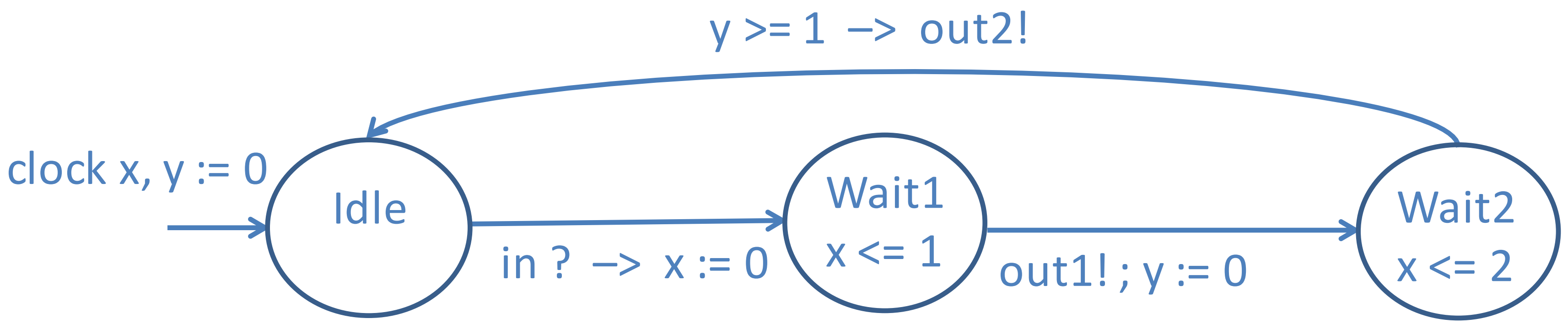
Example with Two Clocks



- Input event: in
- Output events: out1, out2
- Two clock variables: x, y
- As time passes, both clocks increase (and at the same rate)
- Sample timed transitions from state (mode, x, y) = (Wait2, 0.8, 0) :

  (Wait2, 0.8, 0) –0.3–> (Wait2, 1.1, 0.3) –0.72–> (Wait2, 1.82, 1.02)

Two Clock Example



- Clock x tracks time elapsed since the last input event
- Clock y tracks time elapsed since the output event
- What is the behavior of this model?
- If input event occurs at time t, the process issues an output event on channel out1 at time t' within the interval [t, t+1], and then on channel out2 at time t'' within the interval [t'+1, t+2]

Example Specification

- Consider a timed process with

  Input: event x Output: event y, event z

- Desired behavior
  - For each input, produce both output events
  - Time delay between x? and y! is in the interval [2, 4]
  - Time delay between x? and z! is in the interval [3,5]
  - Ignore later inputs received in these intervals

Definition of Timed Process

- A timed process TP consists of
  1. An asynchronous process P, where some of the state variables can be of type clock (ranging non-negative reals)
  2. A *clock invariant* CI, a Boolean expression over P's state variables

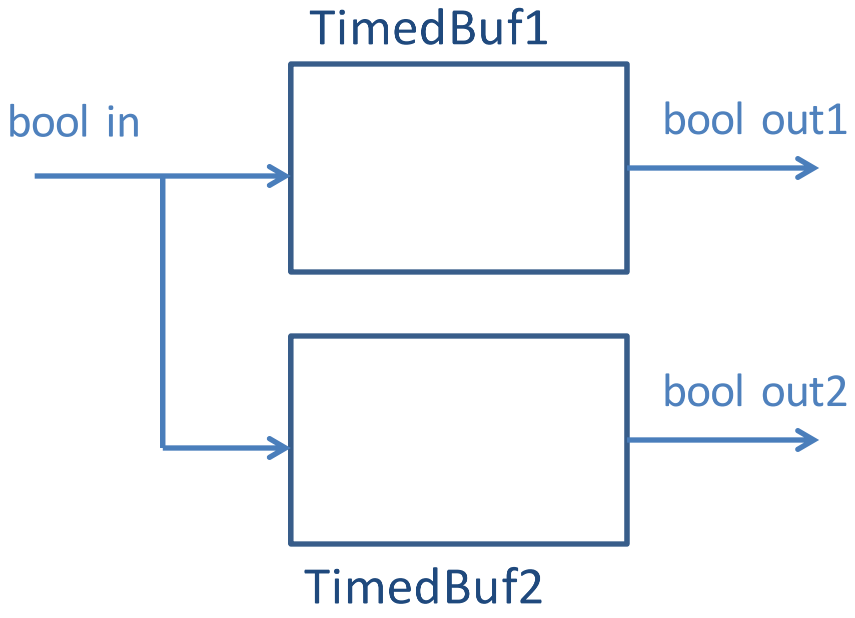
- Inputs, outputs, states, initial states, internal actions, input actions, and output actions exactly as in the asynchronous model

- Notation: For a state s and time t, let s+t denote the state such that
  - (s+t)(x) = s(x)+t for every clock variable x, and
  - (s+t)(y) = s(y) for every non-clock variable y

- Timed actions: Given a state s and a time d > 0, s –d–> s+d is a transition of duration d as long as the state s+t satisfies invariant CI for all t in [0, d]

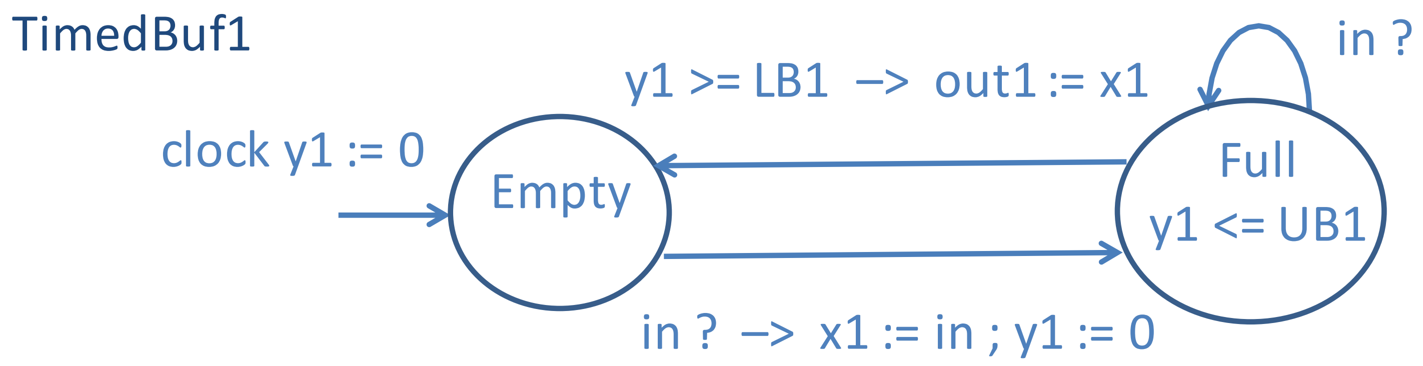
  Note: If a clock-invariant is a convex constraint then it is sufficient to check that the end-states s and s+d satisfy CI

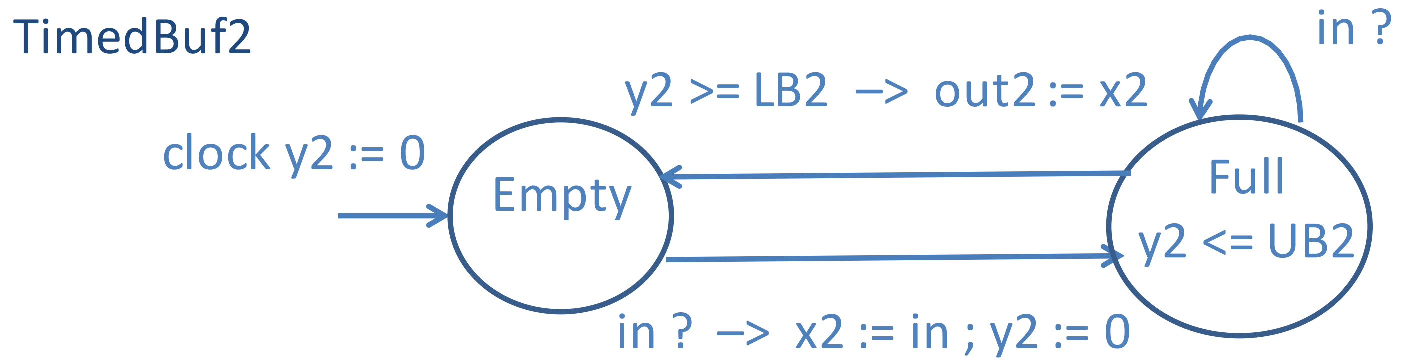
Composition of Processes



- How to construct timed process corresponding to the composition of the two processes?
- What are the possible behaviors of the composite process?

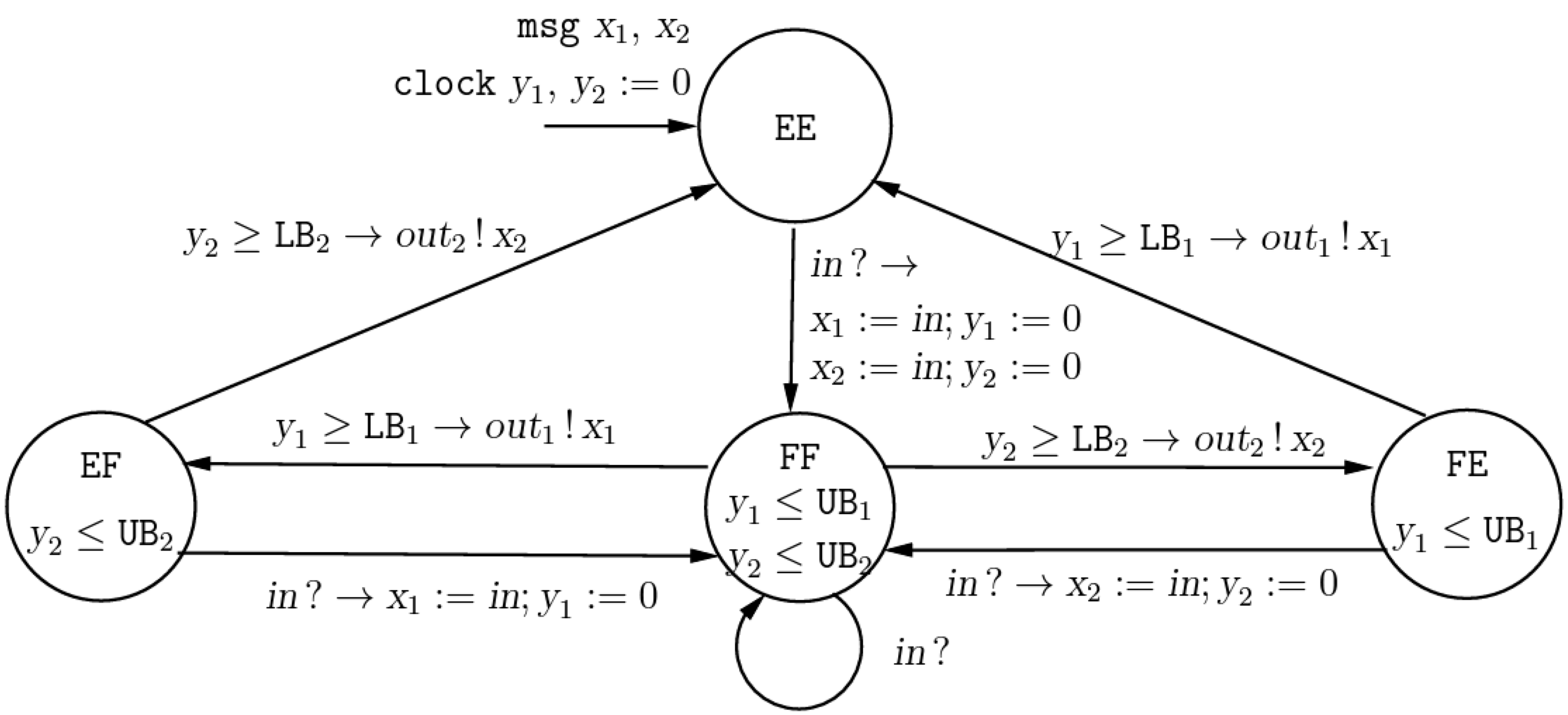
Composition of Timed Processes





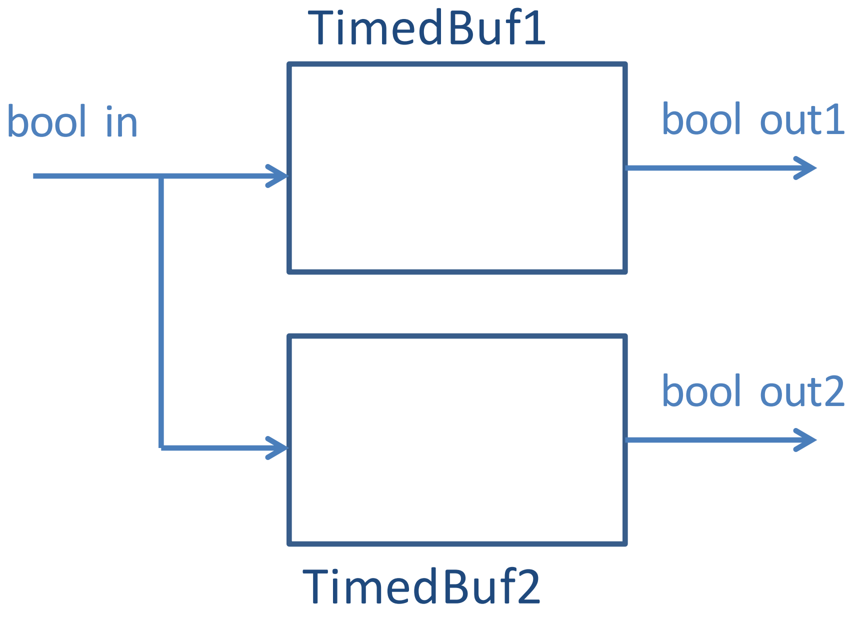
The composite process has four modes: (Empty, Empty), (Empty, Full), (Full, Empty), (Full, Full),

Composition of Timed Processes


msg x_1, x_2
clock $y_1, y_2 := 0$
EE
$y_2 \geq \mathrm{LB}_2 \rightarrow out_2 ! x_2$
$in? \rightarrow$
$x_1 := in; y_1 := 0$
$x_2 := in; y_2 := 0$
$y_1 \geq \mathrm{LB}_1 \rightarrow out_1 ! x_1$
EF
$y_2 \leq \mathrm{UB}_2$
$y_1 \geq \mathrm{LB}_1 \rightarrow out_1 ! x_1$
FF
$y_1 \leq \mathrm{UB}_1$
$y_2 \leq \mathrm{UB}_2$
$y_2 \geq \mathrm{LB}_2 \rightarrow out_2 ! x_2$
FE
$y_1 \leq \mathrm{UB}_1$
$in? \rightarrow x_1 := in; y_1 := 0$
$in? \rightarrow x_2 := in; y_2 := 0$
$in?$

(mode = EF => y2 <= UB2) & (mode = FF => y1 <= UB1 & y2 <= UB2) &
(mode = FE => y1 <= UB1)

Composition of Processes

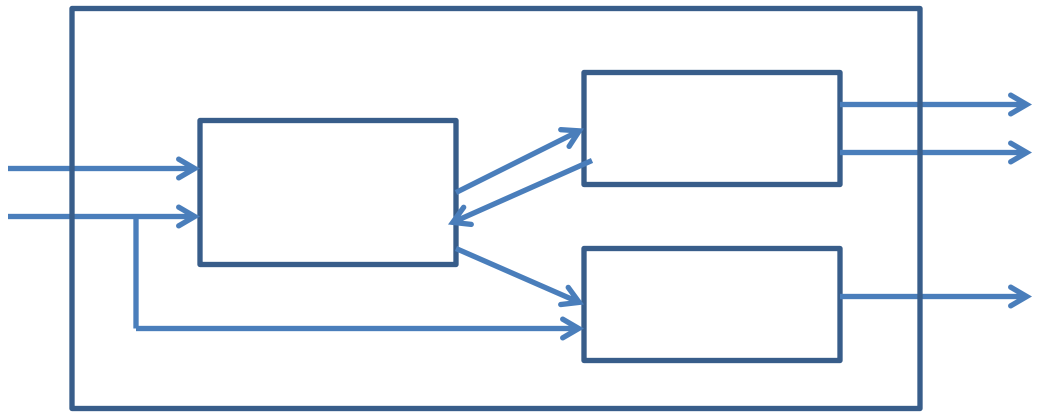


- If UB1 < LB2 then out1 guaranteed to occur before out2
  - Implicit coordination based on bounds on delays
- Is it possible to observe two out1 events without an intervening out2 event?
  - Depends on relative magnitudes of bounds (need timing analysis!)

Definition of Parallel Composition

- Consider timed processes $TP_1 = (P_1, CI_1)$ and $TP_2 = (P_2, CI_2)$
- When is the parallel composition $TP_1 \mid TP_2$ defined?
  - Exactly when the asynchronous parallel composition $P_1 \mid P_2$ is defined (that is, when the outputs of the two are disjoint)
- $TP_1 \mid TP_2 = (P_1 \mid P_2, CI_1 \,\&\, CI_2)$
  - Asynchronous composition of P_1 and P_2 defines the internal, input and output actions of the composite
  - Conjunction of the clock-invariants defines the clock-invariant of the composite
- Consequence: The composite process can allow a timed action of duration d exactly when both TP_1 and TP_2 can wait for time d

Block Diagrams



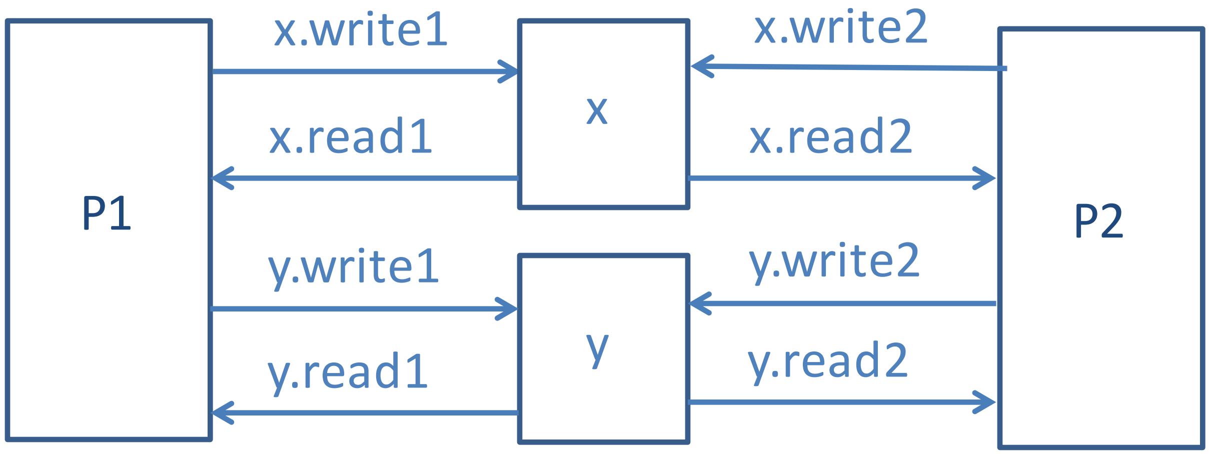
- Components can be timed processes now
  - Operation: instantiation (input/output variable renaming), parallel composition, and variable hiding

- A step of the composite system is either
  1. An internal step of one of components
  2. A communication (input/output) step involving relevant sender and receivers
  3. A timed step involving all the components

Timed Model

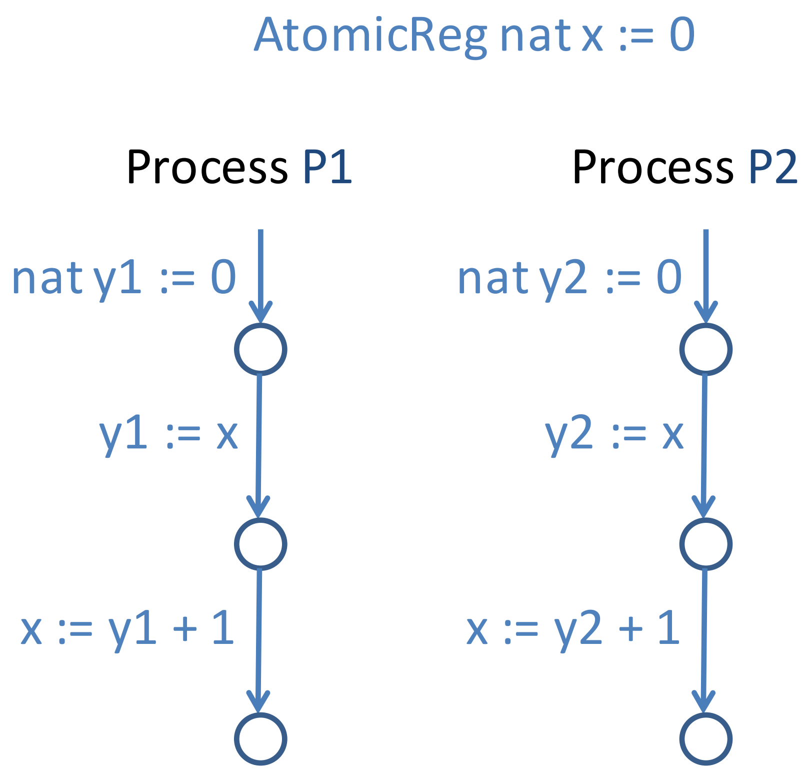
- Timed model is sometimes called the *semi-synchronous* model (mix of asynchronous and synchronous)
- Definitions/concepts that carry over naturally from those models:
  - Executions of a timed process
  - Transition system associated with a timed process
  - Safety/liveness requirements
- Distributed coordination problems: how can we exploit the knowledge of timing delays to design protocols?

Recall: Shared Memory Asynchronous Processes



- Processes P1 and P2 communicate by reading/writing shared variables
- Each shared variable can be modeled as an asynchronous process
  - State of each such process is the value of corresponding variable
  - In implementation, shared memory can be a separate subsystem
- Read and write channel between each process and each shared variable
  - To write x, P1 synchronizes with x on x.write1 channel
  - To read x, P2 synchronizes with x on x.read2 channel

Shared Memory Programs with Atomic Registers



Declaration of shared variables

+ code for each process

Key restriction: Each statement of a process either

changes local variables,

reads a single shared var, or

writes a single shared var

Execution model: execute one step of one of the processes

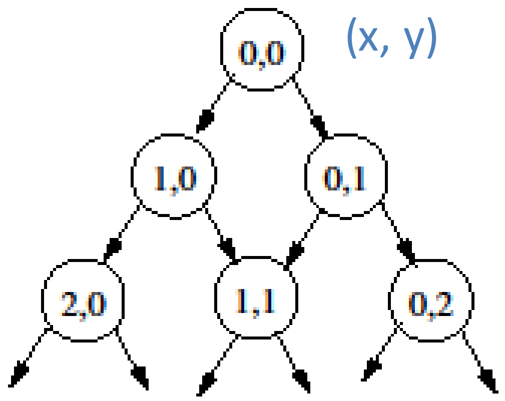
What if we knew lower and upper bounds on how long a read or a write takes? Could we solve coordination problems better?

Asynchronous Execution Model

nat x := 0 ; y := 0

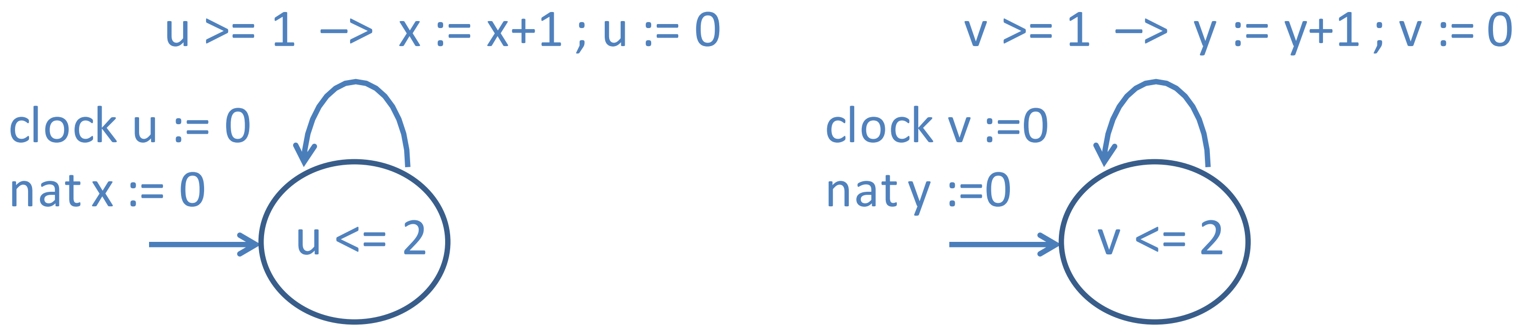
A_x : x := x + 1

A_y : y := y + 1



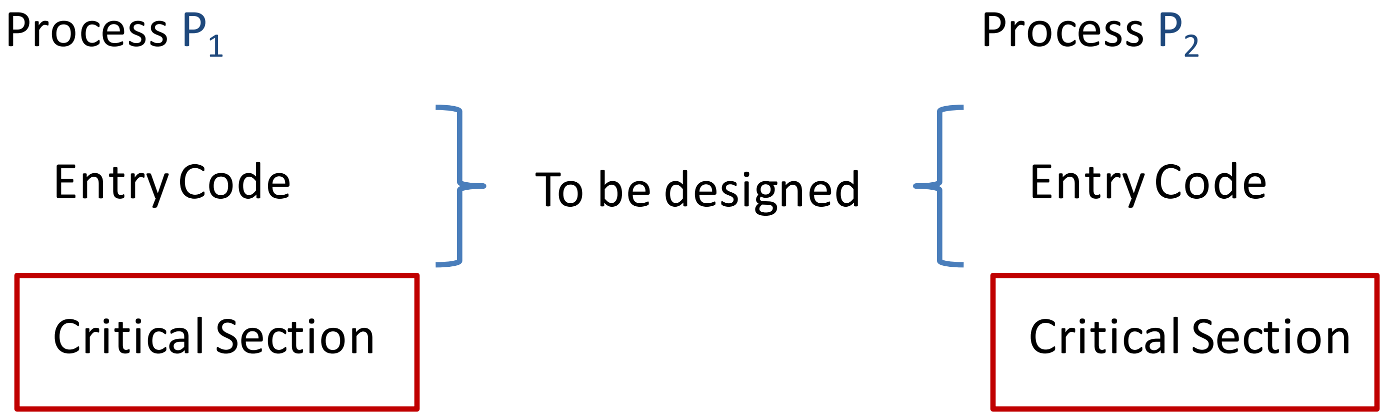
- Tasks A_x and A_y execute in an arbitrary order
- For every possible choice of numbers m and n, the state (m, n) is reachable
- Recall: Fairness assumptions can be used to rule out executions where one of the tasks is ignored forever (although this does not affect the set of reachable states)
- What if we know how long each of these increments take?

Timed Increments



- Task A_x increments x, and this takes between 1 to 2 time units
- Task A_y increments y, and this also takes between 1 to 2 time units
- Two tasks execute in parallel, asynchronously, but timing introduces loose coordination
- Which states are reachable? What is the relationship between m and n so that the state (m, n) is reachable?

Mutual Exclusion Problem



- Safety requirement: processes should not both be in critical section simultaneously (can be formalized using invariants)
- Absence of deadlocks: if any process is trying to enter, then some process should be able to enter

Mutual Exclusion: Incorrect Solution

AtomicReg {0, 1, 2} Turn := 0

Process P1

$$\begin{tikzcd}
{} \arrow[r] & \text{Idle} \arrow[loop above] \arrow[r] & \text{Try1} \arrow[loop above, "\text{else}"] \arrow[r, "\text{Turn = 0 ?}"] & \text{Try2} \arrow[r, "\text{Turn : = 1}"] & \text{Crit} \arrow[loop above] \arrow[lll, bend left, "\text{Turn := 0}"]
\end{tikzcd}$$

Process P2

What is the problem?

$$\begin{tikzcd}
{} \arrow[r] & \text{Idle} \arrow[loop above] \arrow[r] & \text{Try1} \arrow[loop above, "\text{else}"] \arrow[r, "\text{Turn = 0 ?}"] & \text{Try2} \arrow[r, "\text{Turn : = 2}"] & \text{Crit} \arrow[loop above] \arrow[lll, bend left, "\text{Turn := 0}"]
\end{tikzcd}$$

Timing-based Mutual Exclusion

1. Before entering critical section, read the shared variable Turn
2. If Turn != 0 then go to step 1 and try again
3. If Turn = 0 then set Turn to your ID

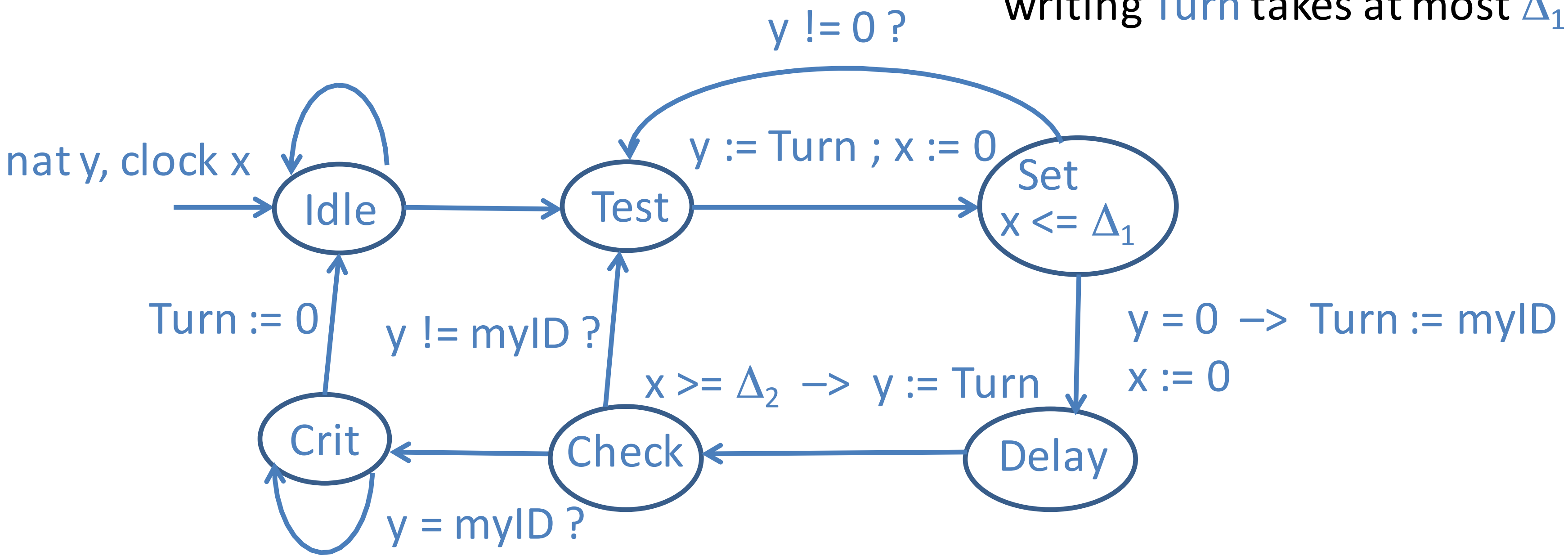
Proceeding directly to critical section is a problem (since the other process may also have concurrently read Turn to be 0, and updating Turn to its own ID). Solution:

4. Delay and wait till you are sure that concurrent writes are finished
5. Read Turn again: if Turn equals your own ID then proceed to critical section; otherwise, go to Step 1 and try again
6. When done with critical section, set Turn back to 0

Fisher's Mutual Exclusion Protocol

AtomicReg Turn := 0

Timing assumption:
writing Turn takes at most Δ_1



Wait for at least Δ_2 time units,
and read Turn again

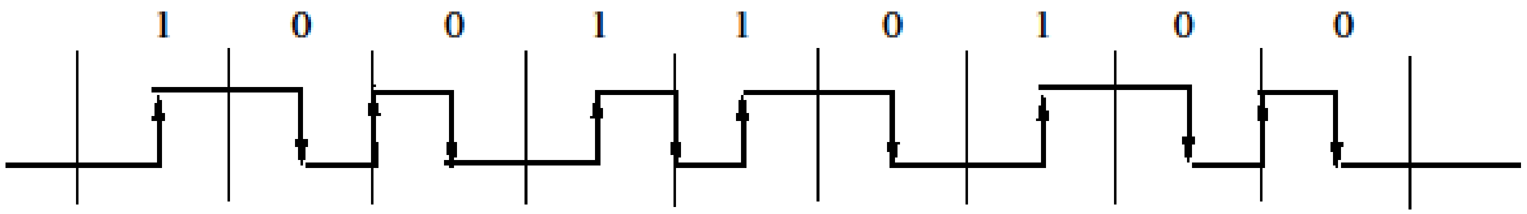
Why does this work ?

Properties of Timed Fisher's Protocol

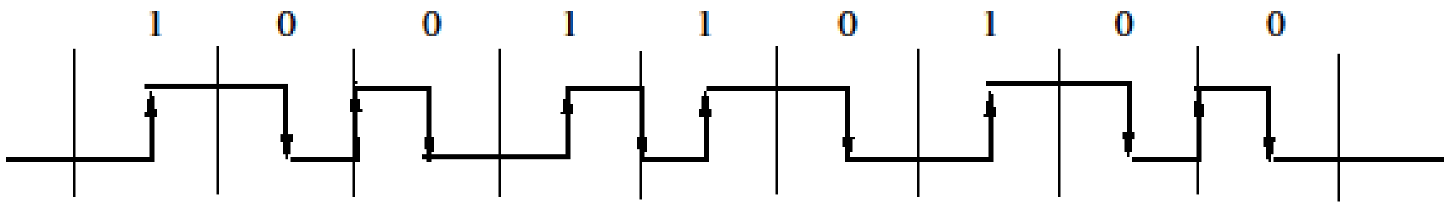
- Assuming $\Delta_2 > \Delta_1$, the algorithm satisfies:
  - Mutual exclusion: Two processes cannot be in critical section simultaneously
  - Deadlock freedom: If a process wants to enter critical section then some process will enter critical section
- Protocol works for arbitrarily many processes (not just 2)
  - In contrast, in the asynchronous model, mutual exclusion protocol for N processes is lot more complex than Peterson's algorithm
- Exercise: Does the protocol satisfy the stronger property of starvation freedom (if a process wants to enter critical section then it eventually will)?
- Exercise: If $\Delta_2 <= \Delta_1$ does mutual exclusion hold? Deadlock freedom?

Timed Communication

- Suppose a sender wants to transmit a sequence of bits to a receiver connected by a communication bus
- Natural strategy: Divide time into slots, and in each slot transmit a bit using high/low voltage values to encode 0/1
- *Manchester encoding*: 0 encoded as a falling edge, and 1 encoded as a rising edge

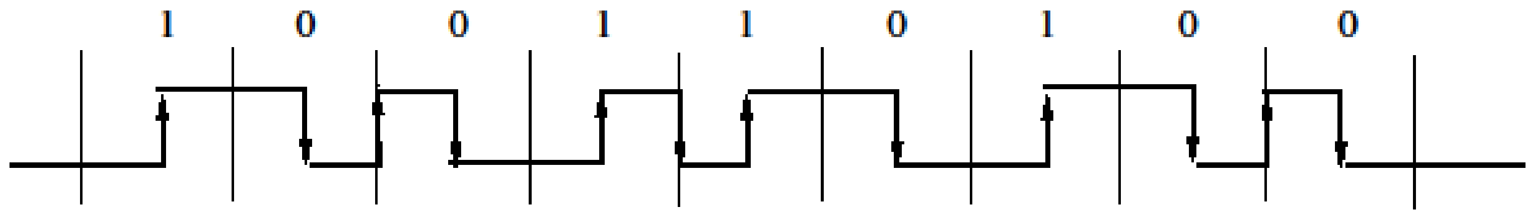


Timed Communication Challenges



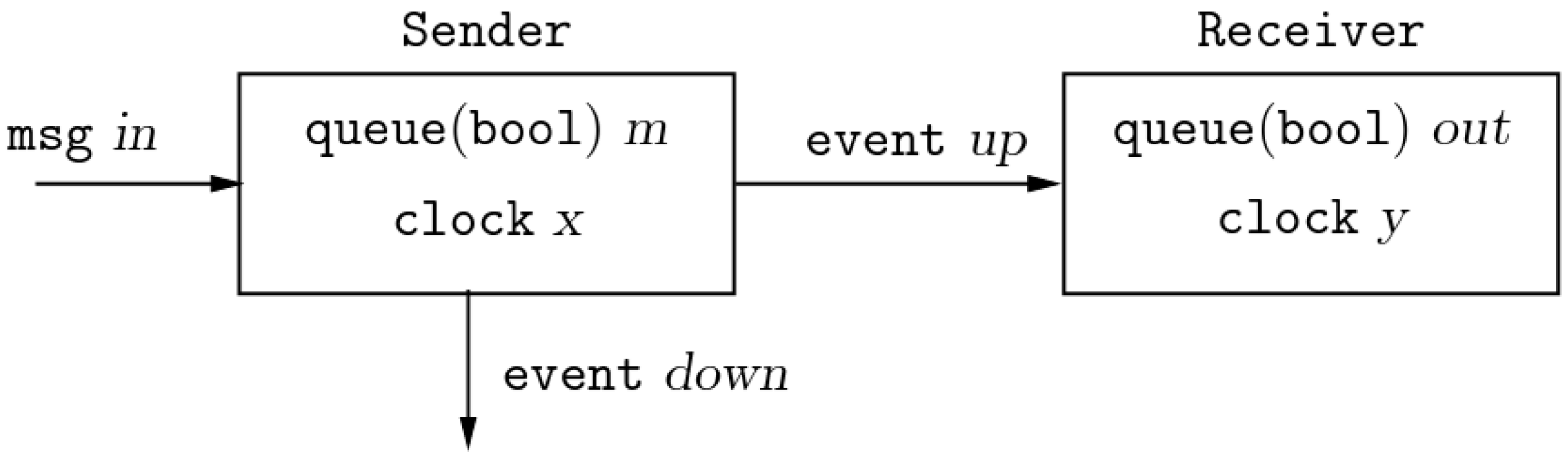
- Sender and receiver know the duration of each time slot, but …
- Receiver does not know when the communication begins
  - When idle, the voltage is set to low
- Receiver cannot reliably detect falling edges
- Sender and receiver clocks are synchronized imperfectly due to drift
  - When a clock x is 1, actual elapsed time is in interval $[1-\varepsilon, 1+\varepsilon]$
  - Since in the timed model clocks are considered to be perfect, we can capture this error by using $x <= 1+\varepsilon$ instead of $x <= 1$, and $1-\varepsilon <= x$ instead of $1 <= x$
- Addressing the challenges:
  - All messages start with 1 and end with 00
  - Processes use timing information to transmit 0s

Audio Control Protocol

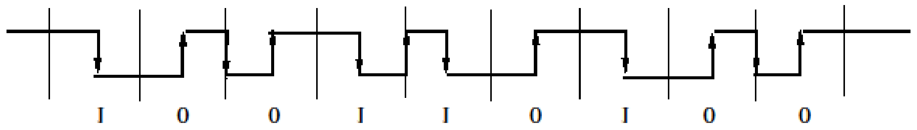


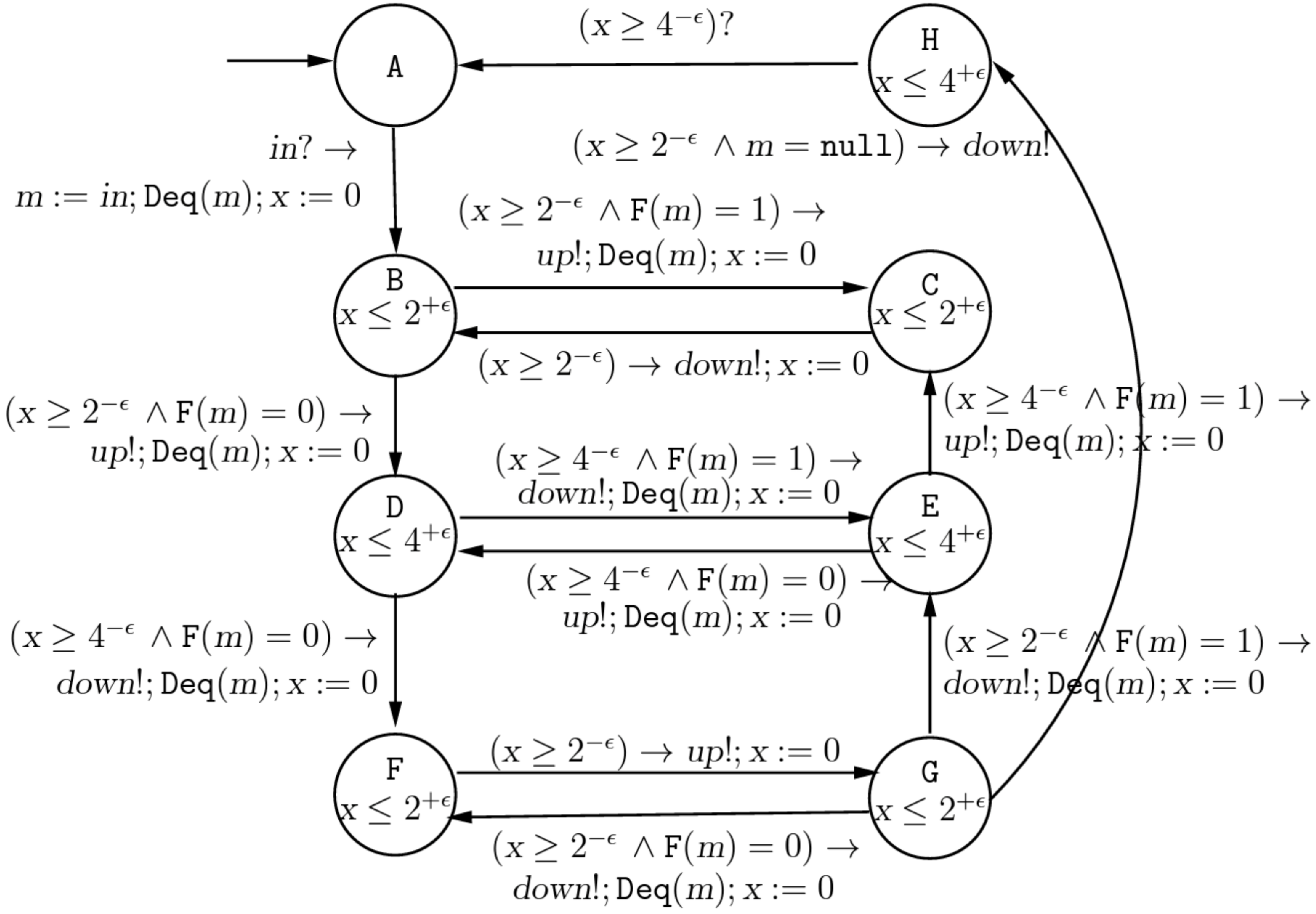
- Protocol developed by Philips to reliably transmit messages in presence of imperfect clocks
- Design logic for receiver to map measured delays between successive raising edges to sequence of bits
- Verification: Prove that message transmission is reliable for a given drift rate ε
- Optimization: Find the largest skew value that the protocol tolerates

Audio Control System



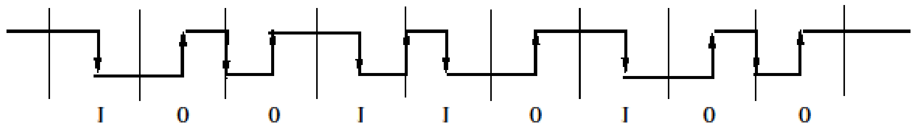
Sender Process

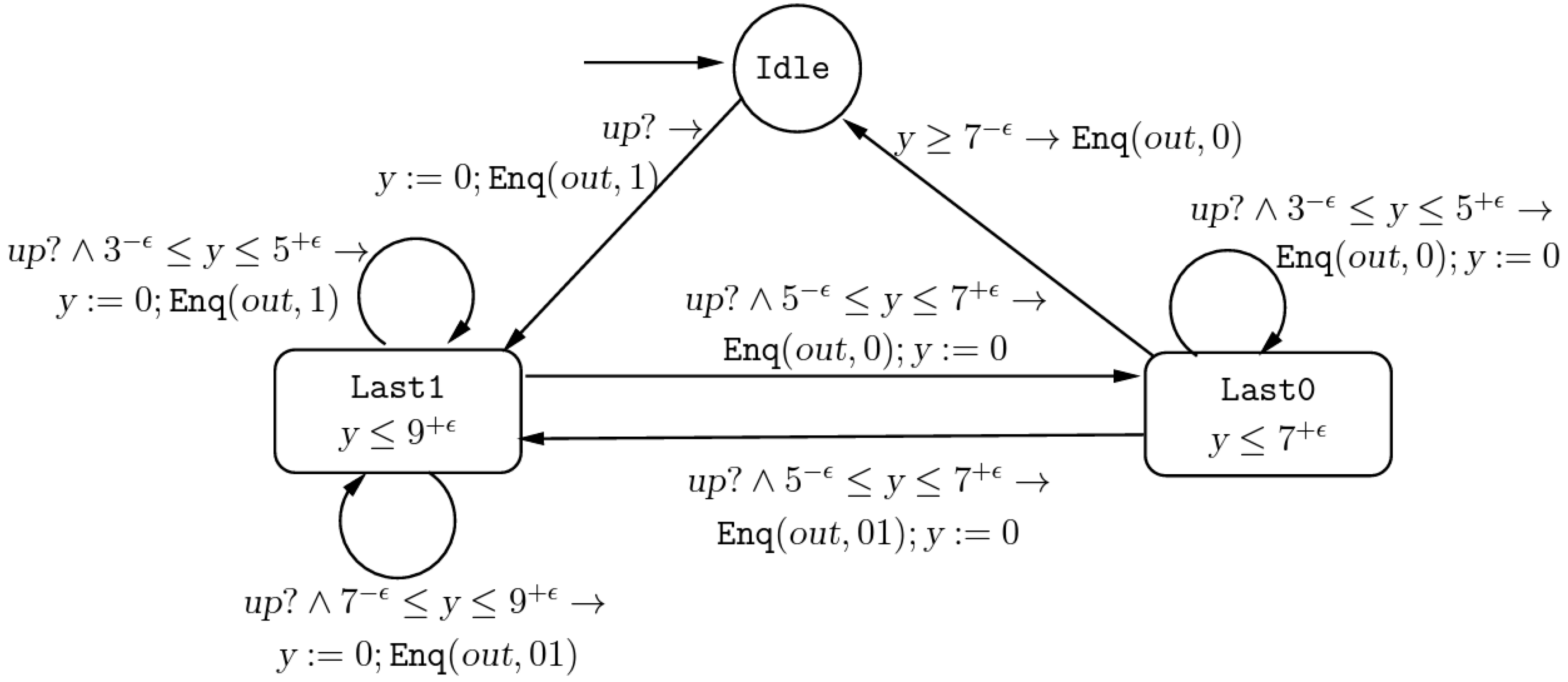




A
H $x \le 4^{+\epsilon}$
B $x \le 2^{+\epsilon}$
C $x \le 2^{+\epsilon}$
D $x \le 4^{+\epsilon}$
E $x \le 4^{+\epsilon}$
F $x \le 2^{+\epsilon}$
G $x \le 2^{+\epsilon}$
$(x \ge 4^{-\epsilon})?$
$in? \to$ $m := in; \mathtt{Deq}(m); x := 0$
$(x \ge 2^{-\epsilon} \wedge m = \mathtt{null}) \to down!$
$(x \ge 2^{-\epsilon} \wedge \mathtt{F}(m) = 1) \to$ $up!; \mathtt{Deq}(m); x := 0$
$(x \ge 2^{-\epsilon}) \to down!; x := 0$
$(x \ge 2^{-\epsilon} \wedge \mathtt{F}(m) = 0) \to$ $up!; \mathtt{Deq}(m); x := 0$
$(x \ge 4^{-\epsilon} \wedge \mathtt{F}(m) = 1) \to$ $up!; \mathtt{Deq}(m); x := 0$
$(x \ge 4^{-\epsilon} \wedge \mathtt{F}(m) = 1) \to$ $down!; \mathtt{Deq}(m); x := 0$
$(x \ge 4^{-\epsilon} \wedge \mathtt{F}(m) = 0) \to$ $up!; \mathtt{Deq}(m); x := 0$
$(x \ge 4^{-\epsilon} \wedge \mathtt{F}(m) = 0) \to$ $down!; \mathtt{Deq}(m); x := 0$
$(x \ge 2^{-\epsilon} \wedge \mathtt{F}(m) = 1) \to$ $down!; \mathtt{Deq}(m); x := 0$
$(x \ge 2^{-\epsilon}) \to up!; x := 0$
$(x \ge 2^{-\epsilon} \wedge \mathtt{F}(m) = 0) \to$ $down!; \mathtt{Deq}(m); x := 0$

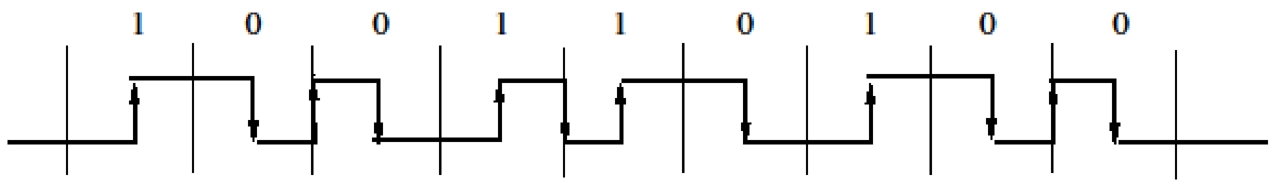
Receiver Process





Idle
Last1
$y \leq 9^{+\epsilon}$
Last0
$y \leq 7^{+\epsilon}$
$up? \rightarrow$ $y := 0; \texttt{Enq}(out, 1)$
$y \geq 7^{-\epsilon} \rightarrow \texttt{Enq}(out, 0)$
$up? \wedge 3^{-\epsilon} \leq y \leq 5^{+\epsilon} \rightarrow$ $y := 0; \texttt{Enq}(out, 1)$
$up? \wedge 3^{-\epsilon} \leq y \leq 5^{+\epsilon} \rightarrow$ $\texttt{Enq}(out, 0); y := 0$
$up? \wedge 5^{-\epsilon} \leq y \leq 7^{+\epsilon} \rightarrow$ $\texttt{Enq}(out, 0); y := 0$
$up? \wedge 5^{-\epsilon} \leq y \leq 7^{+\epsilon} \rightarrow$ $\texttt{Enq}(out, 01); y := 0$
$up? \wedge 7^{-\epsilon} \leq y \leq 9^{+\epsilon} \rightarrow$ $y := 0; \texttt{Enq}(out, 01)$

Execution Example



Time	Event	x	Sender	Queue m	y	Receiver	Queue out
0			B	00110100		Idle	null
2.07	*up*	2.07	D	0110100		Last1	1
5.97	*down*	3.9	F	110100	3.9	Last1	1
7.97	*up*	2	G	110100	5.9	Last0	10
9.92	*down*	1.95	E	10100	1.95	Last0	10
14.08	*up*	4.16	C	0100	6.11	Last1	1001
16.1	*down*	2.02	B	0100	2.02	Last1	1001
18	*up*	1.9	D	100	3.92	Last1	10011
22.05	*down*	4.05	E	00	4.05	Last1	10011
25.91	*up*	3.86	D	0	7.91	Last1	1001101
30.01	*down*	4.1	F	null	4.1	Last1	1001101
32.11	*up*	2.1	G	null	6.2	Last0	10011010
34.16	*down*	2.05	H	null	2.05	Last0	10011010
38.29		4.13	A	null	6.18	Last0	10011010
39.39		1.1	A	null	7.28	Idle	100110100

Credits

Notes based on Chapter 7 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015