# *Discrete Event Models: Getting the Semantics Right*


Edward A. Lee

Robert S. Pepper Distinguished Professor

Chair of EECS

UC Berkeley



With thanks to Xioajun Liu, Eleftherios Matsikoudis, and Haiyang Zheng


Invited Keynote Talk

Winter Simulation Conference

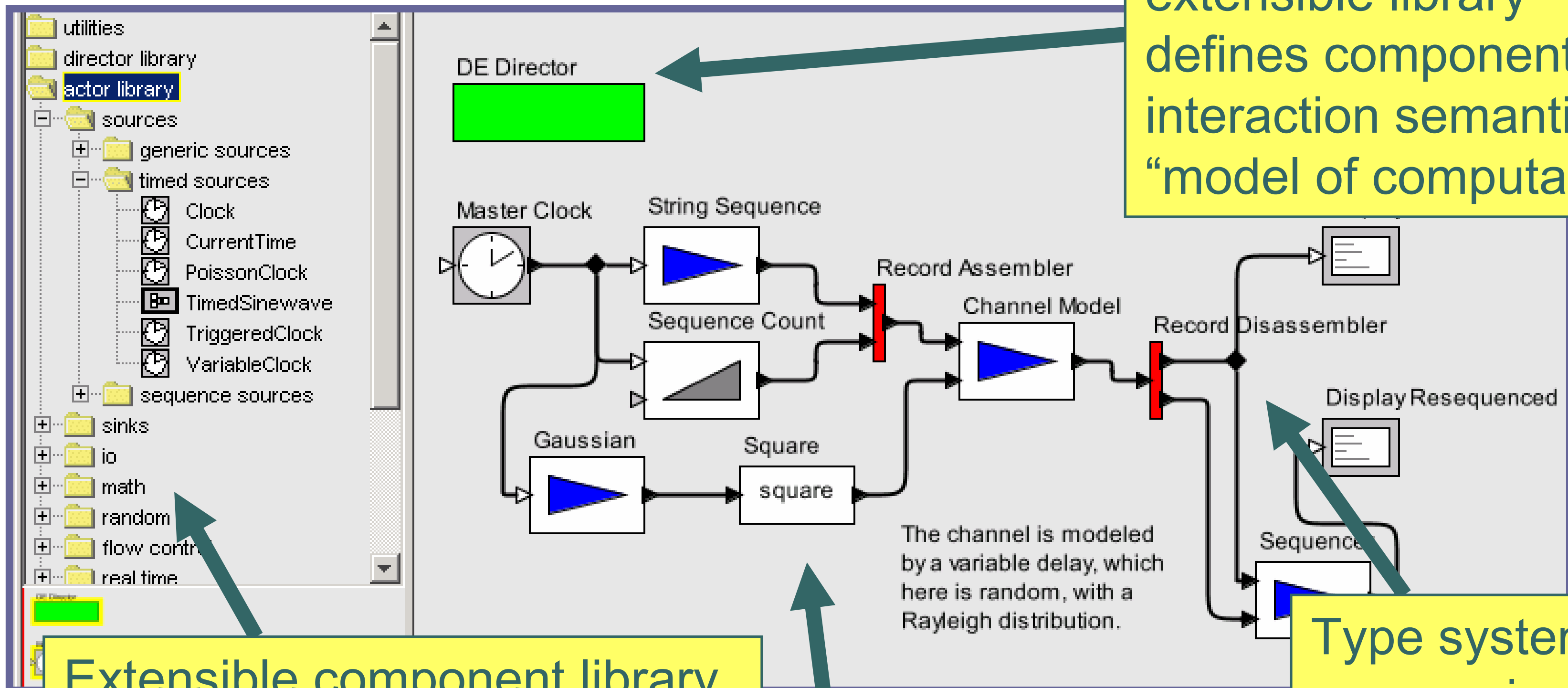
December 4, 2006

Monterey, CA

# Ptolemy II: Our Laboratory for Studying Concurrent Models of Computation

Concurrency management supporting dynamic model structure.

Director from an extensible library defines component interaction semantics or "model of computation."




Extensible component library.

Type system for communicated data

Visual editor for defining models

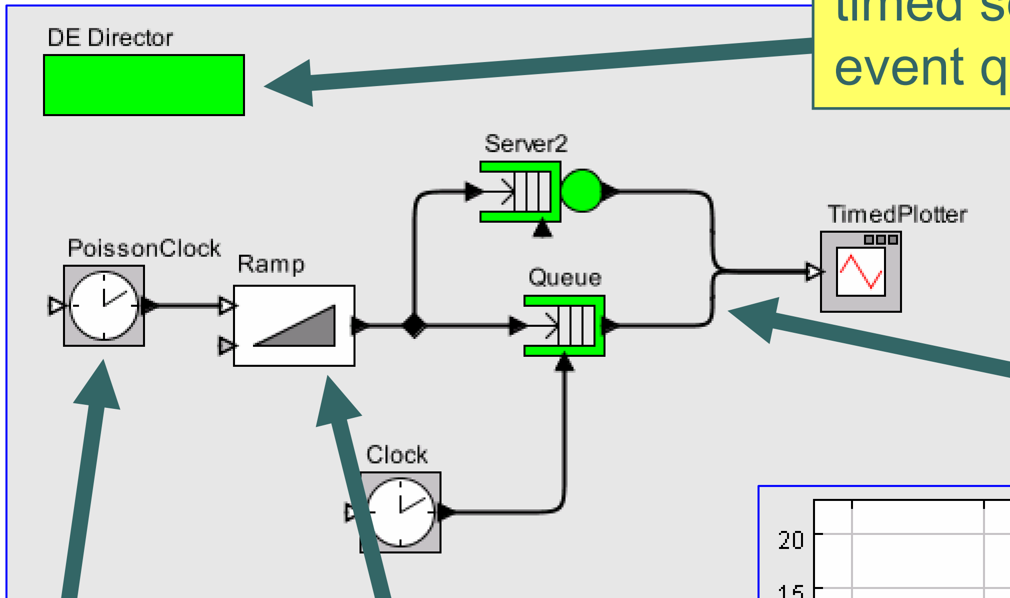
# Some Models of Computation Implemented in Ptolemy II

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DDF – dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

This talk will focus on Discrete Events (DE)

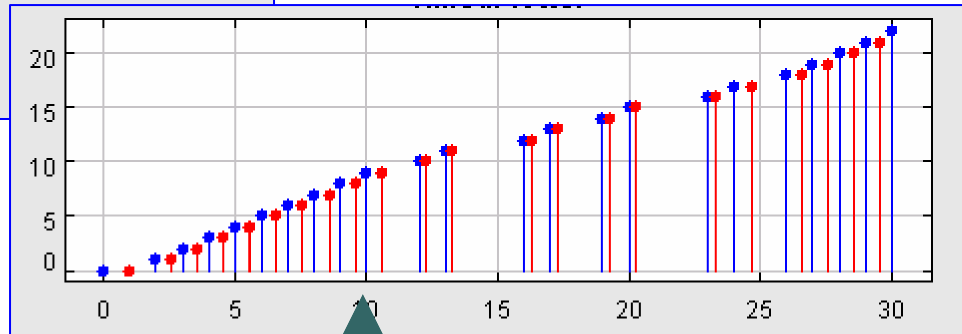
But will also establish connections with Continuous Time (CT), Synchronous Reactive (SR) and hybrid systems (CT + FSM)

# Discrete Events (DE): A Timed Concurrent Model of Computation




DE Director implements timed semantics using an event queue

Actors communicate via "signals" that are marked point processes (discrete, valued, events in time).




Actor (in this case, source of events)

Reactive actors produce output events in response to input events

Plot of the value (marking) of a signal as function of time.

# Our Applications of DE

- Modeling and simulation of
  - Communication networks (mostly wireless)
  - Hardware architectures
  - Systems of systems
- Design and software synthesis for
  - Sensor networks (TinyOS/nesC)
  - Distributed real-time software
  - Hardware/software systems

# First Attempt at a Model for Signals

Let $\mathbb{R}_+$ be the non-negative real numbers. Let $V$ be an arbitrary family of values (a data type, or alphabet). Let

$$V_\varepsilon = V \cup \{\varepsilon\}$$

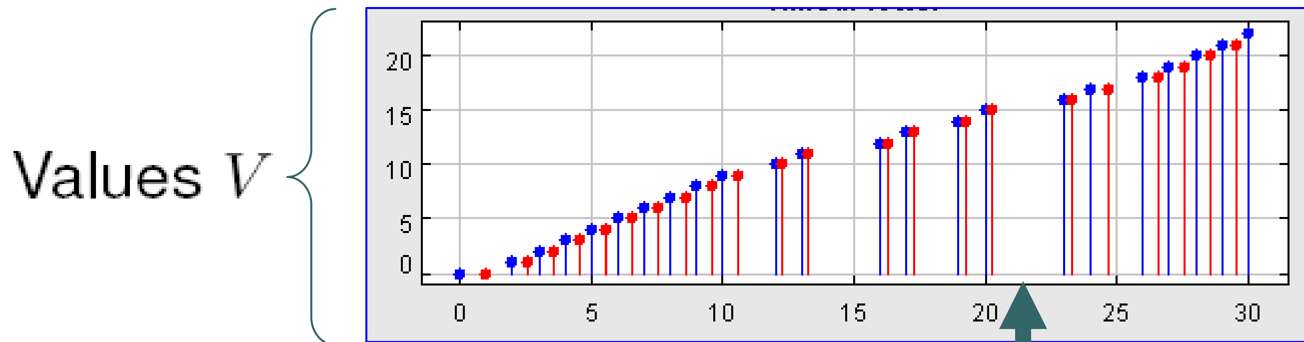
be the set of values plus "absent." Let $s$ be a signal, given as a partial function:

$$s : \mathbb{R}_+ \rightharpoonup V_\varepsilon$$

defined on an initial segment of $\mathbb{R}_+$

# First Attempt at a Model for Signals

$$s : \mathbb{R}_+ \rightharpoonup V_\varepsilon$$

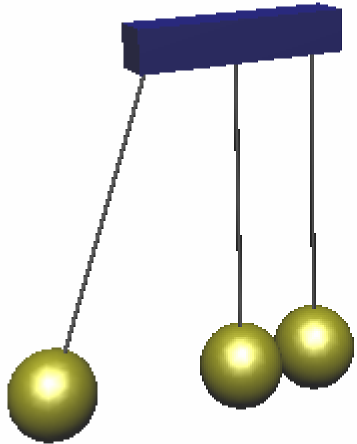



Initial segment $I \subseteq \mathbb{R}_+$ where the signal is defined.

Absent: $s(\tau) = \varepsilon$ for almost all $\tau \in I$.

*This model is not rich enough because it does not allow a signal to have multiple events at the same time.*

# Example Motivating the Need for Simultaneous Events Within a Signal



*Newton's Cradle*:

- Steel balls on strings
- Collisions are events
- Momentum of the middle ball has three values at the time of collision.

This example has continuous dynamics as well (I will return to this)

*Other examples*:

- Batch arrivals at a queue.
- Software sequences abstracted as instantaneous.
- Transient states.

# A Better Model for Signals: *Super-Dense Time*

Let $T = \mathbb{R}_+ \times \mathbb{N}$ be a set of "tags" where $\mathbb{N}$ is the natural numbers, and give a signal $s$ as a partial function:

$$s : T \rightharpoonup V_\varepsilon$$

defined on an initial segment of $T$, assuming a lexical ordering on $T$:

$$(t_1, n_1) \leq (t_2, n_2) \iff t_1 < t_2, \text{ or } t_1 = t_2 \text{ and } n_1 \leq n_2 \,.$$

*This allows signals to have a sequence of values at any real time t.*

# Super Dense Time



Initial segment $I \subseteq \mathbb{R}_+ \times \mathbb{N}$ where the signal is defined

Absent: $s(\tau) = \varepsilon$ for almost all $\tau \in I$.

# Events and Firings

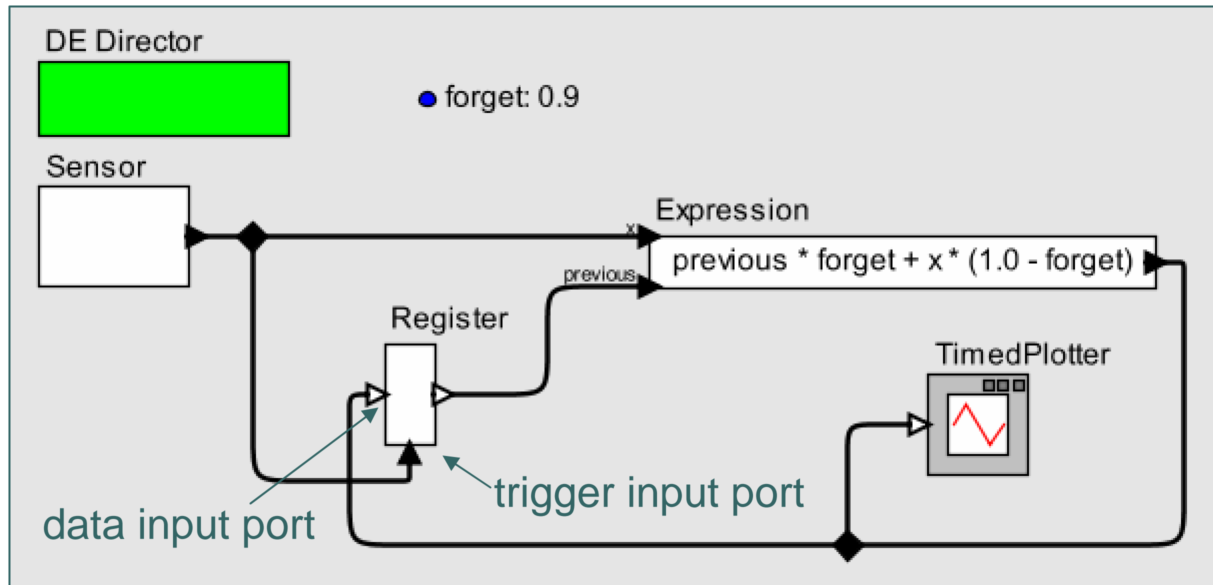
$$s : T \rightharpoonup V_\varepsilon$$

- A *tag* is a time-index pair, $\tau = (t, n) \in T = \mathbb{R}_+ \times \mathbb{N}$.
- An *event* is a tag-value pair, $e = (\tau, v) \in T \times V$.
- $s(\tau)$ is an event if $s(\tau) \neq \varepsilon$.

Operationally, events are processed by presenting all input events at a tag to an actor and then *firing* it.

However, this is not always possible!

# A Feedback Design Pattern




In this model, a sensor produces measurements that are combined with previous measurements using an exponential forgetting function.

The feedback loop makes it impossible to present the Register actor with all its inputs at any tag before firing it.

# Solving Feedback Loops

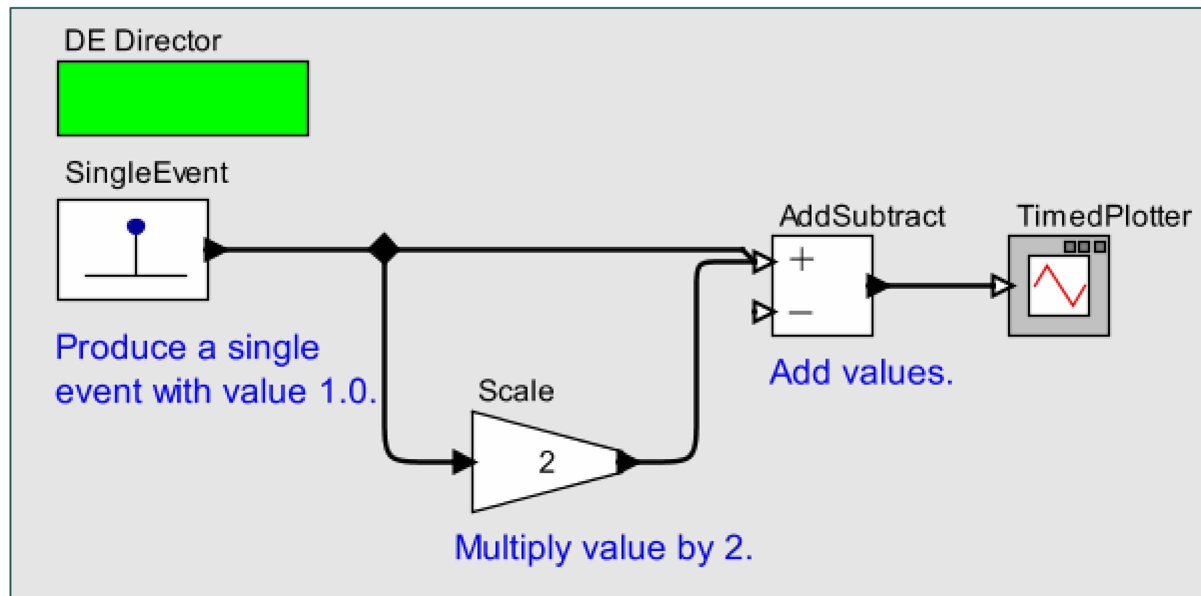
Possible solutions:

- All actors have time delay
- Some actors have time delay,
  and every directed loop must have an actor with time delay.
- All actors have delta delay
- Some actors have delta delay
  and every directed loop must have an actor with delta delay.

Although each of these solutions is used, all are problematic.

The root of the problem is simultaneous events.

# Consider "All Actors Have Time Delay"



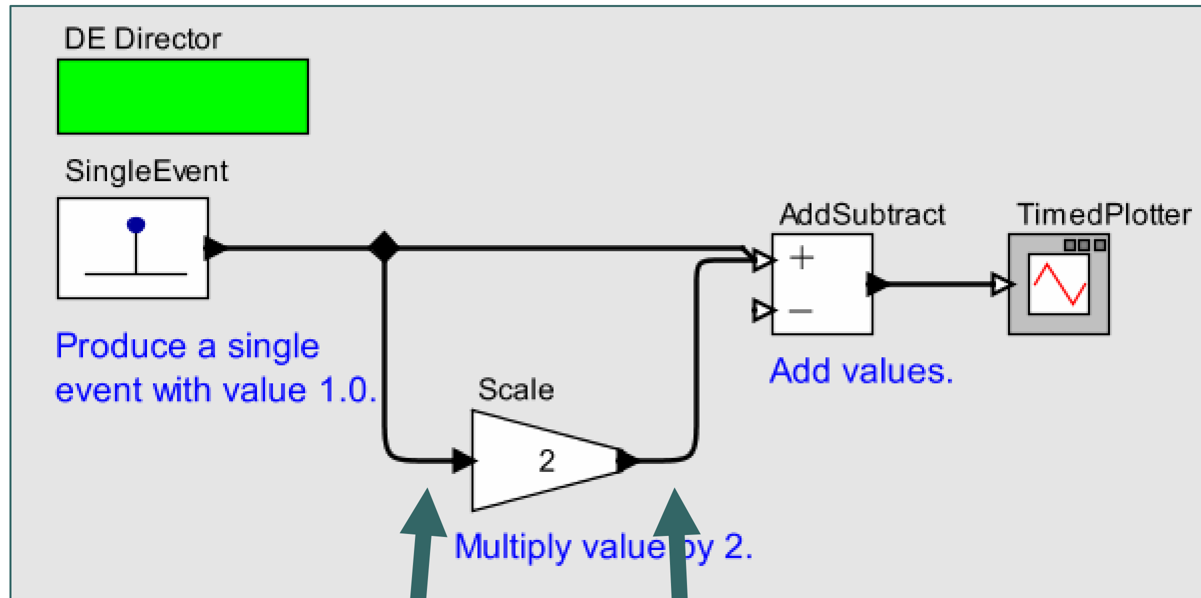

If all actors have time delay, this produces either:

- Event with value 1 followed by event with value 2, or
- Event with value 1 followed by event with value 3.

(the latter if signal values are persistent).

*Neither of these is likely what we want.*

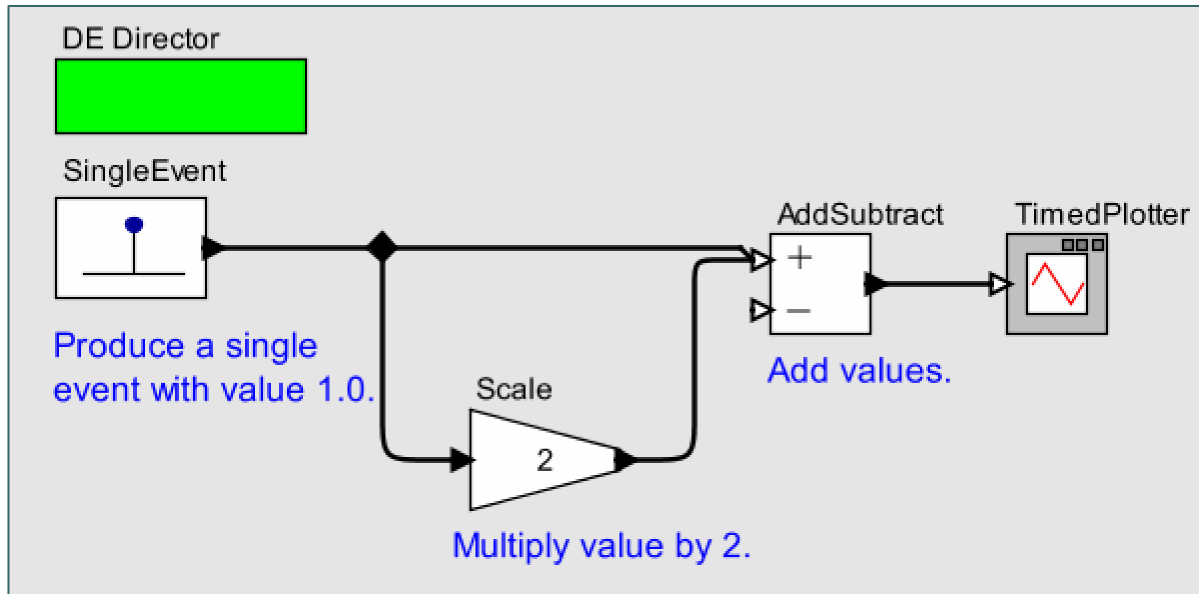
# Consider "All Actors Have Delta Delay"




With delta delays, if an input event is ((t, n), v), the corresponding output event is ((t, n+1), v'). Every actor is assumed to give a delta delay.

This style of solution is used in VHDL.

# Consider "All Actors Have Delta Delay"



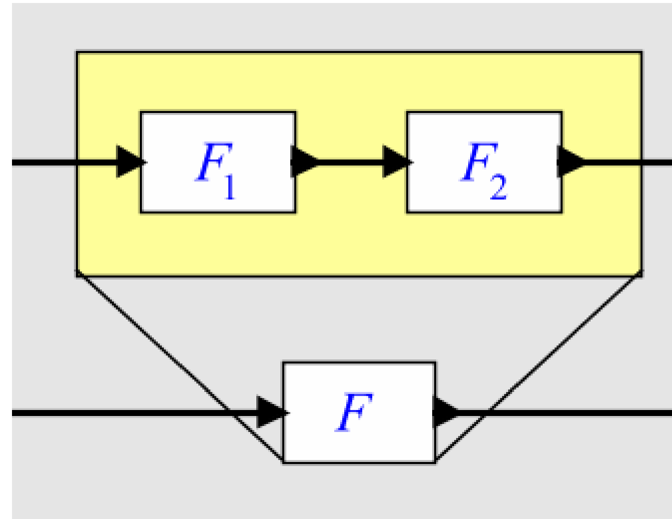

If all actors have a delta delay, this produces either:

- Event with value 1 followed by event with value 2, or
- Event with value 1 followed by event with value 3

(the latter if signal values are persistent, as in VHDL).

*Again, neither of these is likely what we want.*

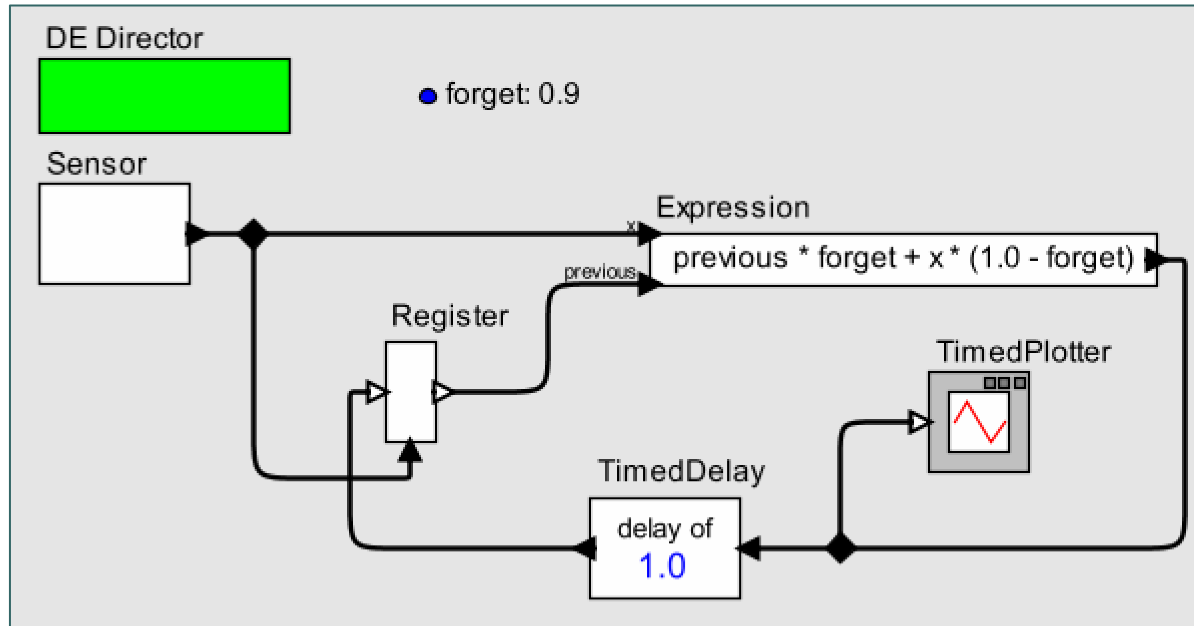
# More Fundamental Problem: Delta Delay Semantics is Not Compositional



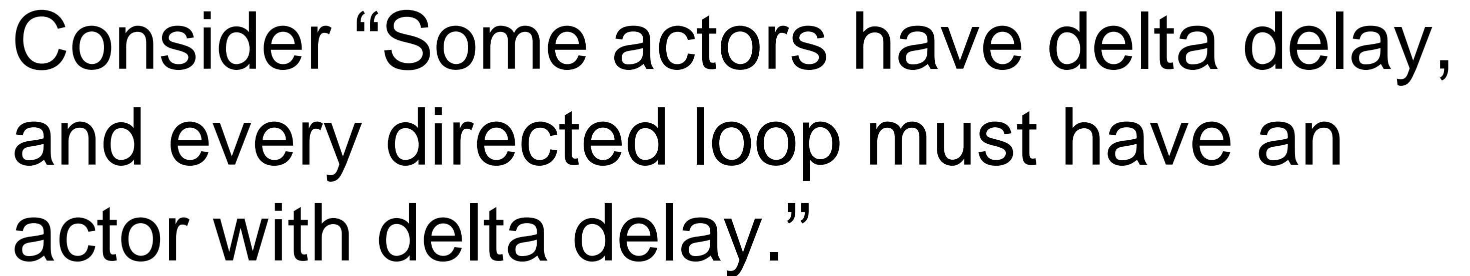

The top composition of two actors will have a two delta delays, whereas the bottom abstraction has only a single delta delay.

Under delta delay semantics, a composition of two actors cannot have the semantics as a single actor.

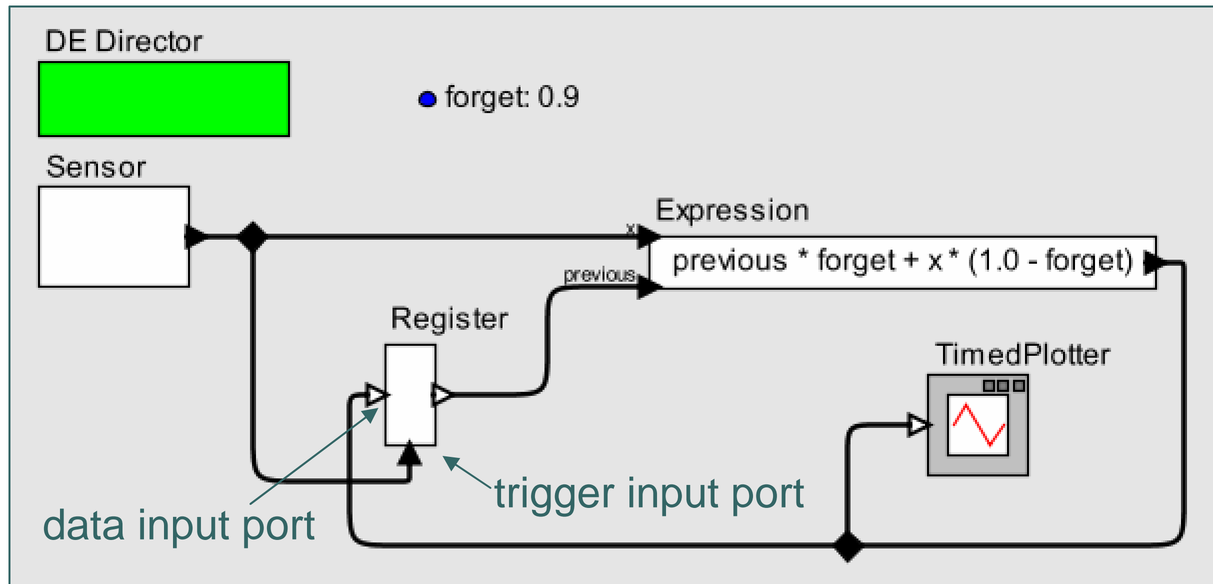
# Consider "Some actors have time delay, and every directed loop must have an actor with time delay."




Any non-zero time delay imposes an upper bound on the rate at which sensor data can be accepted. Exceeding this rate will produce erroneous results.



# Consider "Some actors have delta delay, and every directed loop must have an actor with delta delay."

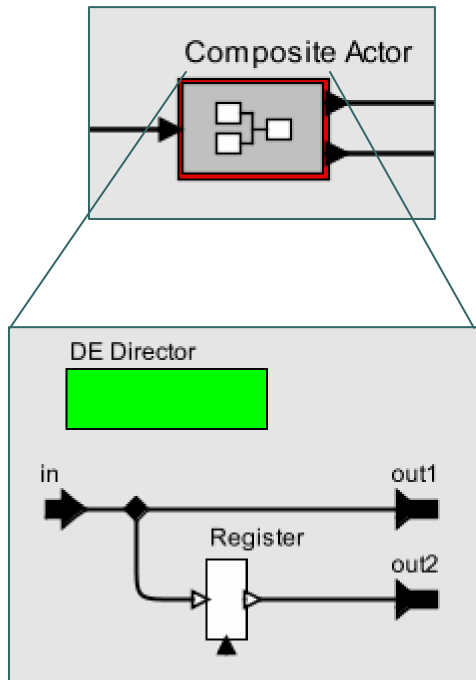



The output of the Register actor must be at least one index later than the data input, hence this actor has at least a delta delay.

To schedule this, could break the feedback loop at actors with delta delay, then do a topological sort.

# Naïve Topological Sort is not Compositional

Breaking loops where an actor has a delta delay and performing a topological sort is not a compositional solution:




Does this composite actor have a delta delay or not?

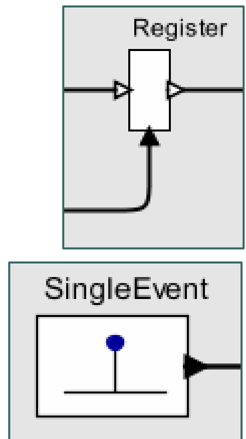
# Our Answer: No Required Delay, and Feedback Loops Have Fixed Points Semantics



Given an input event ((t, n), v), the corresponding output event is ((t, n), v'). The actor has no delay.

The challenge now is to establish a determinate semantics and a scheduling policy for execution.

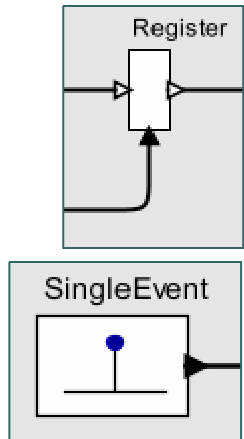
# How Does This Work? Execution of Ptolemy II Actors




Flow of control:

- Initialization
- Execution
- Finalization

# How Does This Work? Execution of Ptolemy II Actors

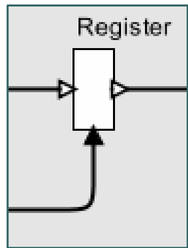


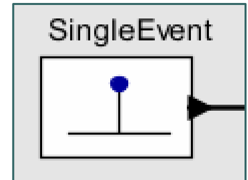

Flow of control:

- Initialization
- Execution
- Finalization

**Post tags on the event queue corresponding to any initial events the actor wants to produce.**

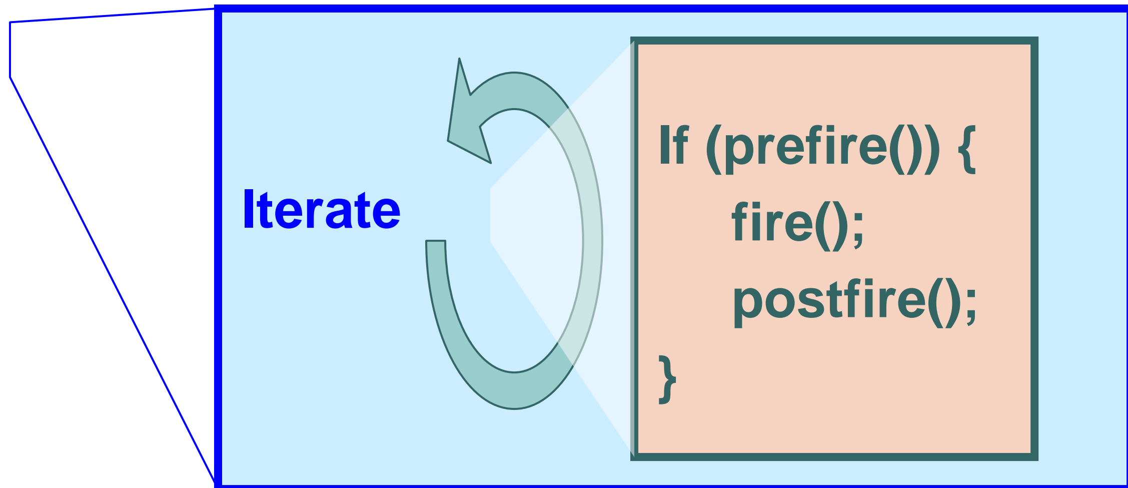
# How Does This Work? Execution of Ptolemy II Actors



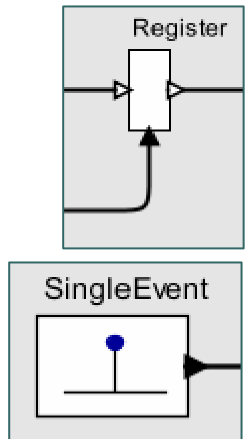

Flow of control:

- Initialization
- Execution
- Finalization




**Only the postfire() method can change the state of the actor.**

# How Does This Work? Execution of Ptolemy II Actors




Flow of control:

- Initialization
- Execution
- Finalization

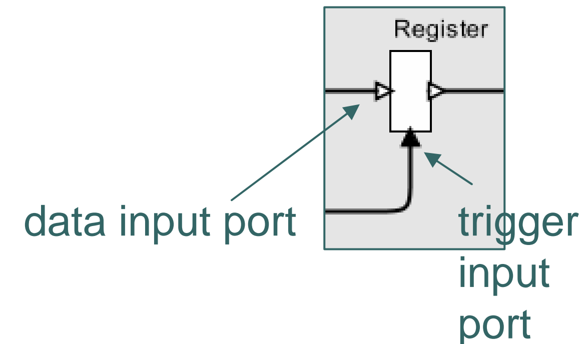
# Definition of the Register Actor (Sketch)

Can the actor fire?

React to trigger input.

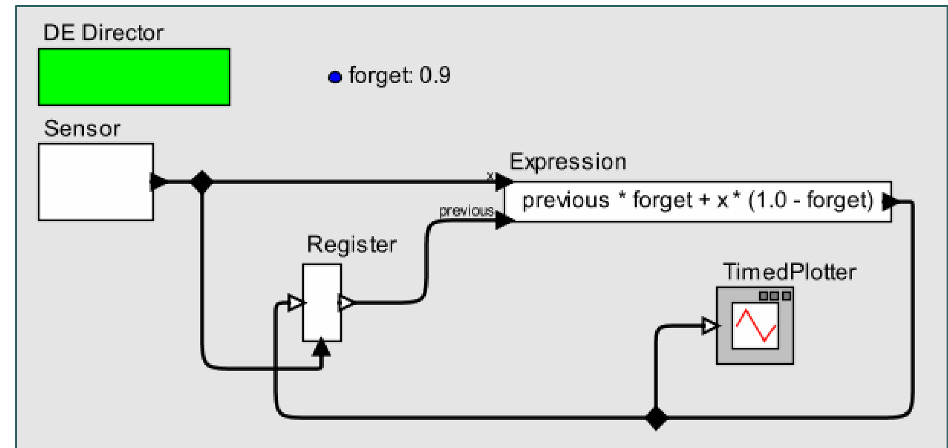
Read the data input and update the state.

```
class Register {
  private Object state;
  boolean prefire() {
    if (trigger is known) { return true; }
  }
  void fire() {
    if (trigger is present) {
      send state to output;
    } else {
      assert output is absent;
    }
  }
  void postfire() {
    if (trigger is present) {
      state = value read from data input;
    }
  }
```



# Execution of a DE Model (Conceptually)

- Start with all signals empty. Initialize all actors (some will post tags on the event queue).
- Take all earliest tag (t, n) from the event queue.
- Mark all signals unknown at tag (t, n).
- Prefire and fire the actors in any order. If enough is known about the inputs to an actor, it may make outputs known at (t, n).
- Keep firing actors in any order until all signals are known at (t, n).
- When all signals are known, postfire all actors (to commit state changes). Any actor may now post a tag (t', n') > (t, n) on the event queue.




Key questions:

- Is this right?
- Can this be made efficient?

The answer, of course, is yes to both.

This scheme underlies synchronous/reactive languages (Esterel, Lustre, Signal, etc.)

# Where We Are

Proposed:

- Superdense time
- Zero delay actors
- Execution policy

Now: Show that it's right.

- Conditions for uniqueness (Scott continuity)
- Conditions for liveness (causality)

# Observation: Any Composition is a Feedback Composition

p1 $F_1$ p5 p2 $F_2$ p4 p3 $F_3$ p6

(a)

p1 $F_1$ p5 p2 $F_2$ p4 p3 $F_3$ p6

(b)

p1 p2 p3 $F$ p4 p5 p6

(c)

$\mathbf{s} \in S^N$

$F$

(d)

- Signal: $s \colon \mathbb{R}_+ \times \mathbb{N} \rightharpoonup V_\varepsilon$
- Set of signals: $S$
- Tuples of signals: $\mathbf{s} \in S^N$
- Actor: $F \colon S^N \to S^M$

If every actor is a function, then the semantics of the overall system is the least $\mathbf{s} \in S^N$ such that $F(\mathbf{s}) = \mathbf{s}$.

We have a *least fixed point semantics.*

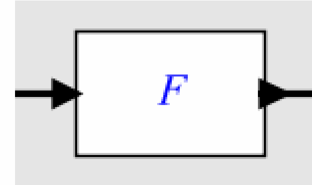
# Prefix Order**

- Recall that a signal $s$ is a partial function: defined on an initial segment of $T$. Such a function can be given by its *graph*, $s \subset T \times V_\varepsilon$.
- A signal $s_1$ is a *prefix* of a signal $s_2$ if $s_1 \subseteq s_2$. The prefix relation is a partial order on the set $S$ of signals.
- Fact: $S$ with the prefix order is a complete semilattice (and hence also a CPO).
- Generalizes easily to tuples of signals $S^N$.

# Monotonic and Continuous Functions**

A function $F\colon S \to S$ is *monotonic* if it is order-preserving,

$$\forall s_1, s_2 \in S, \quad s_1 \subseteq s_2 \implies F(s_1) \subseteq F(s_2).$$




The same function is (Scott) *continuous* if for all directed sets $S' \subseteq S$, $F(S')$ is a directed set and

$$F(\bigvee S') = \bigvee F(S').$$

Here, $F(S')$ is defined in the natural way as $\{F(s) \mid s \in S'\}$, and $\vee X$ denotes the least upper bound of the set $X$.

Every continuous function is monotonic, and behaves as follows: Extending the input (in time or tags) can only extend the output.

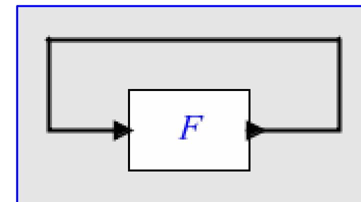
# Knaster-Tarski Fixed-Point Theorem**

A classic fixed point theorem states that if $F$ is continuous, then it has a least fixed point, and that least fixed point is

$$\bigvee \{F^n(\perp_S) \mid n \in \mathbb{N}\},$$

where $\perp_S$ is the least element of $S$ (the empty signal) and $\mathbb{N}$ is the natural numbers.

- Start with empty signals.
- Iteratively apply function *F*.
- Converge to the unique solution.



# Summary: Existence and Uniqueness of the Least Fixed Point Solution.

p1 p5 p2 p4 p3 p6 $F_1$ $F_2$ $F_3$

(a)

p1 p5 p2 p4 p3 p6 $F_1$ $F_2$ $F_3$

(b)

p1 p2 p3 p4 p5 p6 $F$

(c)

$\mathbf{s} \in S^N$

$F$

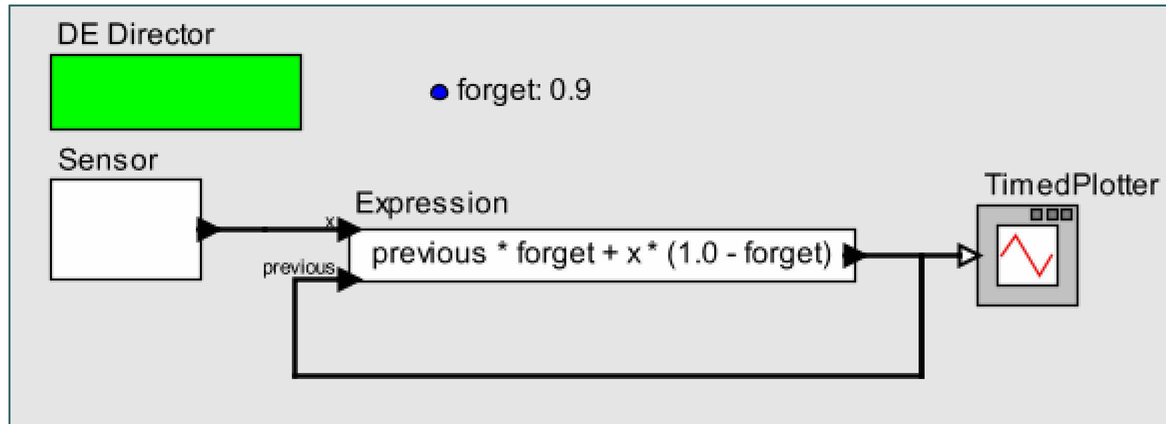
(d)

- Signal: $s\colon \mathbb{R}_+ \times \mathbb{N} \rightharpoonup V_\varepsilon$
- Set of signals: $S$
- Tuples of signals: $\mathbf{s} \in S^N$
- Actor: $F\colon S^N \to S^M$

A unique least fixed point, $\mathbf{s} \in S^N$ such that $F(\mathbf{s}) = \mathbf{s}$, exists and be constructively found if $S^N$ is a CPO and $F$ is (Scott) continuous.

Under our execution policy, actors are usually (Scott) continuous.

# But: Need to Worry About Liveness: Deadlocked Systems

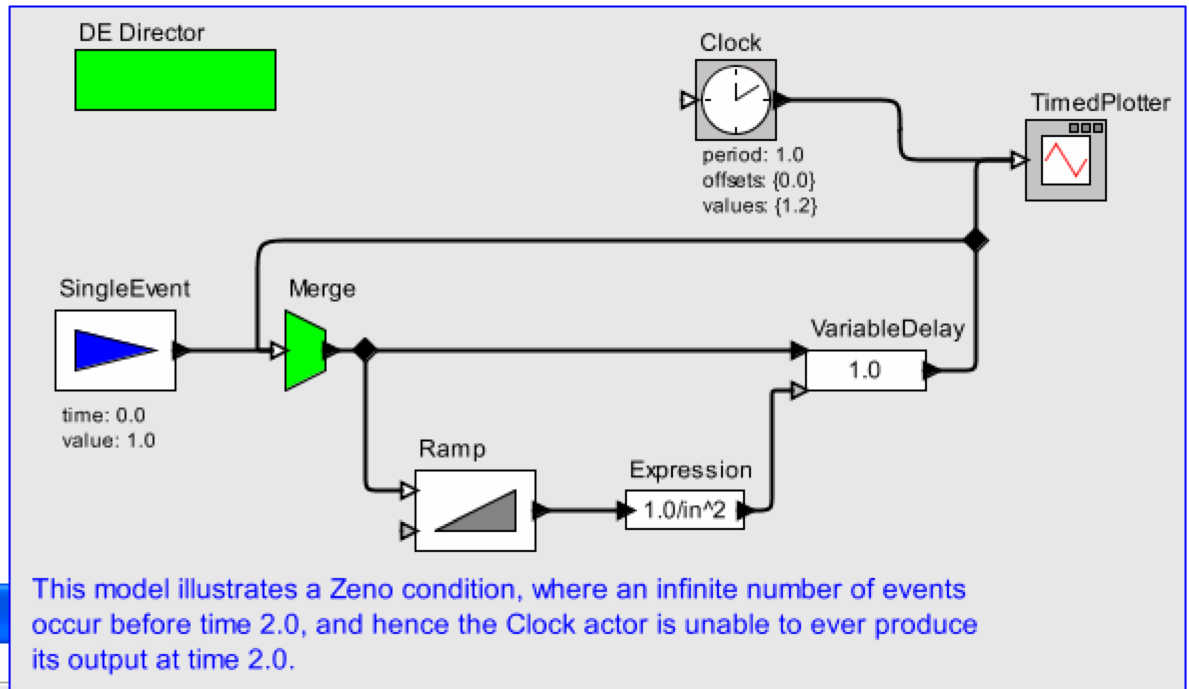


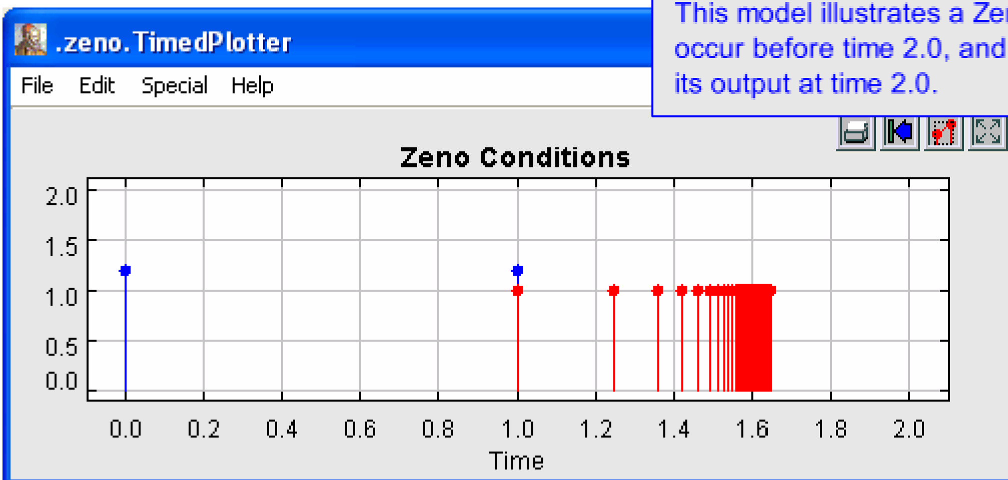

Existence and uniqueness of a solution is not enough.

The least fixed point of this system consists of empty signals. It is deadlocked!

# Another Liveness Concern: Zeno Systems

DE systems may have an infinite number of events in a finite amount of time. These "Zeno systems" can prevent time from advancing.




In this case, our execution policy fails to implement the Knaster-Tarski constructive procedure because some of the signals are not total.

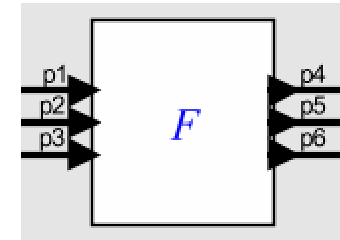
# Liveness

- A signal is *total* if it is defined for all tags in $T$.
- A model with no inputs is *live* if all signals are total.
- A model with inputs is *live* if all input signals are total implies all signals are total.

*Liveness ensures freedom from deadlock and Zeno.*

- Whether a model is live is, in general, undecidable.
- We have developed a useful sufficient condition based on *causality* that ensures liveness.

# Causality Ensures Liveness of an Actor




A monotonic actor $F$ is *causal* if for all sets of input signals $S_i$, the corresponding set of output signals $S_o = F(S_i)$ satisfy

$$\bigcap_{s \in S_i} \mathrm{dom}(s) \subseteq \bigcap_{s \in S_o} \mathrm{dom}(s) .$$

An immediate consequence of this definition is that a causal actor is live. Thus, whether a composition of actors is causal will tell us whether it is live.

Causality does not imply continuity and continuity does not imply causality. Continuity ensure existence and uniqueness of a least fixed point, whereas causality ensures liveness.

# Strict Causality Ensures Liveness of a Feedback Composition

A composition of causal actors without directed cycles is itself a causal actor. With cycles, we need:

- A monotonic actor $F$ is *strictly causal* if for all sets of input signals $S_i$, the corresponding set of output signals $S_o = F(S_i)$ either consists only of total signals (defined over all $T$) or

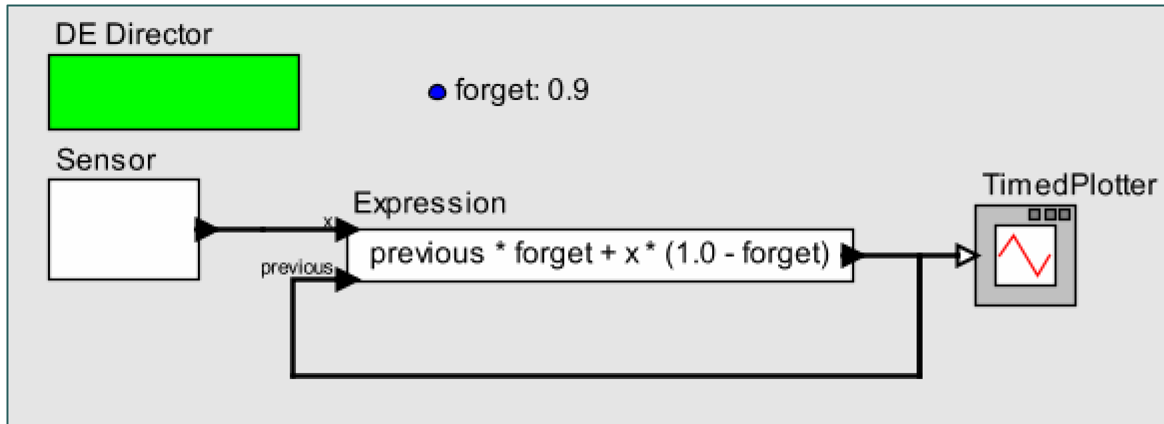
$$\bigcap_{s \in S_i} \mathrm{dom}(s) \subset \bigcap_{s \in S_o} \mathrm{dom}(s) .$$

($\subset$ denotes strict subset). If $F$ is a strictly causal actor with one input and one output, then $F(s_\perp) \neq s_\perp$. $F$ must "come up with something from nothing."

# Continuity, Liveness, and Causality

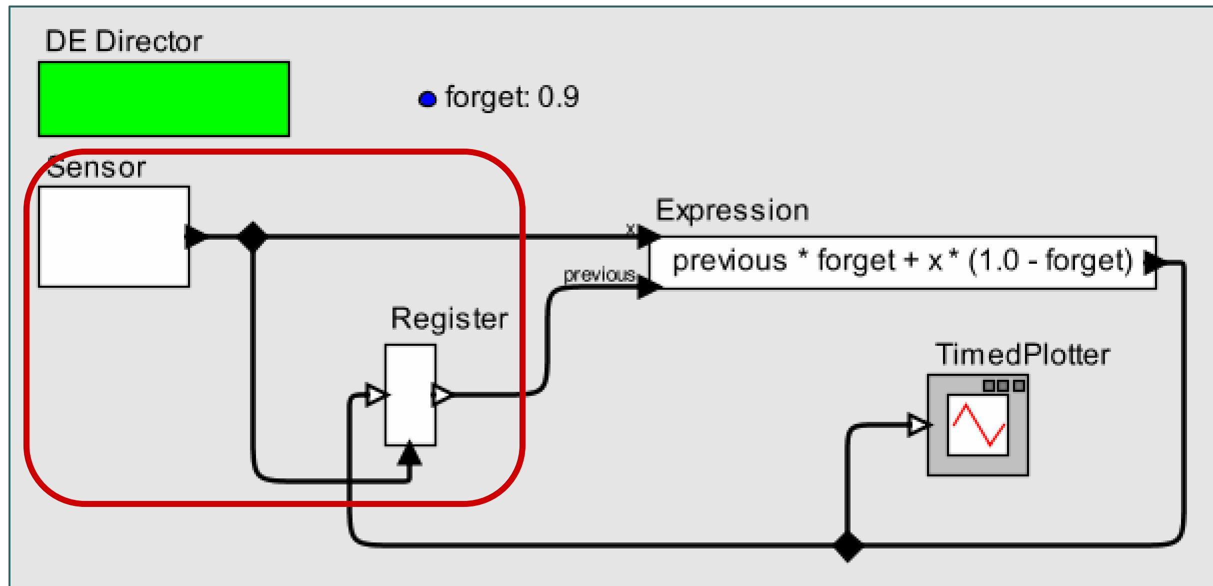
**Theorem**: Given a totally ordered tag set and a network of causal and continuous actors where in every dependency loop in the network there is at least one strictly causal actor, then the network is a causal and continuous actor.

This gives us sufficient, but not necessary condition for freedom deadlock and Zeno.




The feedback loop has no strictly causal actor.

# Feedback Loop that is Not Deadlocked




This feedback loop also has no strictly causal actor, unless…

We aggregate the two actors as shown into one.

# Causality Interfaces Make Scheduling of Execution and Analysis for Liveness Efficient

A causality interface exposes just enough information about an actor to make scheduling and liveness analysis efficient.

An algebra of interfaces enables inference of the causality interface of a composition.

(a)

(b)

(c)

# Models of Computation Implemented in Ptolemy II

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DDF – dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Done

But will also establish connections with Continuous Time (CT) and hybrid systems (CT + FSM)

SR is a special case of DE where time has no metric.

# Standard Model for Continuous-Time Signals

In ODEs, the usual formulation of the signals of interest is a function from the time line (a connected subset of the reals) to the reals:

$$p\colon \mathbb{R}_+ \to \mathbb{R}^n$$

$$\dot{p}\colon \mathbb{R}_+ \to \mathbb{R}^n$$

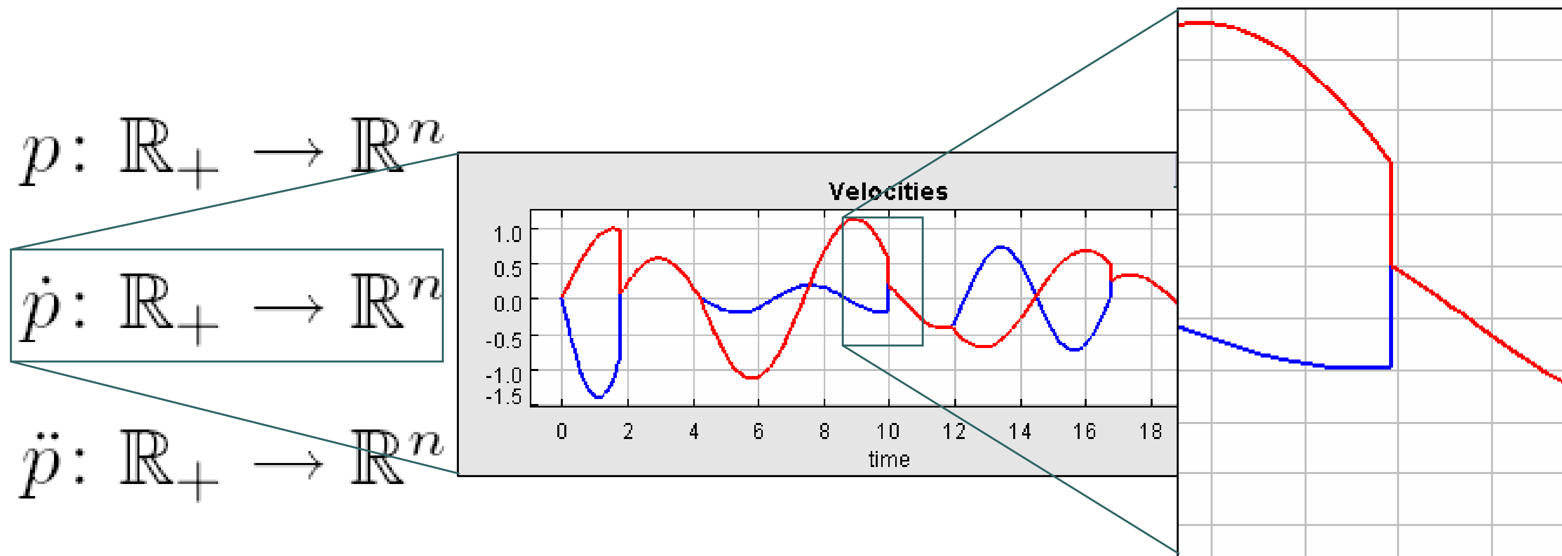
$$\ddot{p}\colon \mathbb{R}_+ \to \mathbb{R}^n$$

Such signals are continuous at $t \in \mathbb{R}_+$ if (e.g.):

$$\forall\, \epsilon > 0, \exists\, \delta > 0, \text{s.t.} \forall \tau \in (t-\delta, t+\delta), \quad ||\dot{p}(t) - \dot{p}(\tau)|| < \epsilon$$

# Piecewise Continuous Signals

In hybrid systems of interest, signals have discontinuities.




*Piecewise continuous signals* are continuous at all $t \in \mathbb{R}_+ \setminus D$ where $D \subset \mathbb{R}_+$ is a *discrete set.*[1]

[1]A set $D$ with an order relation is a *discrete set* if there exists an order embedding to the integers.

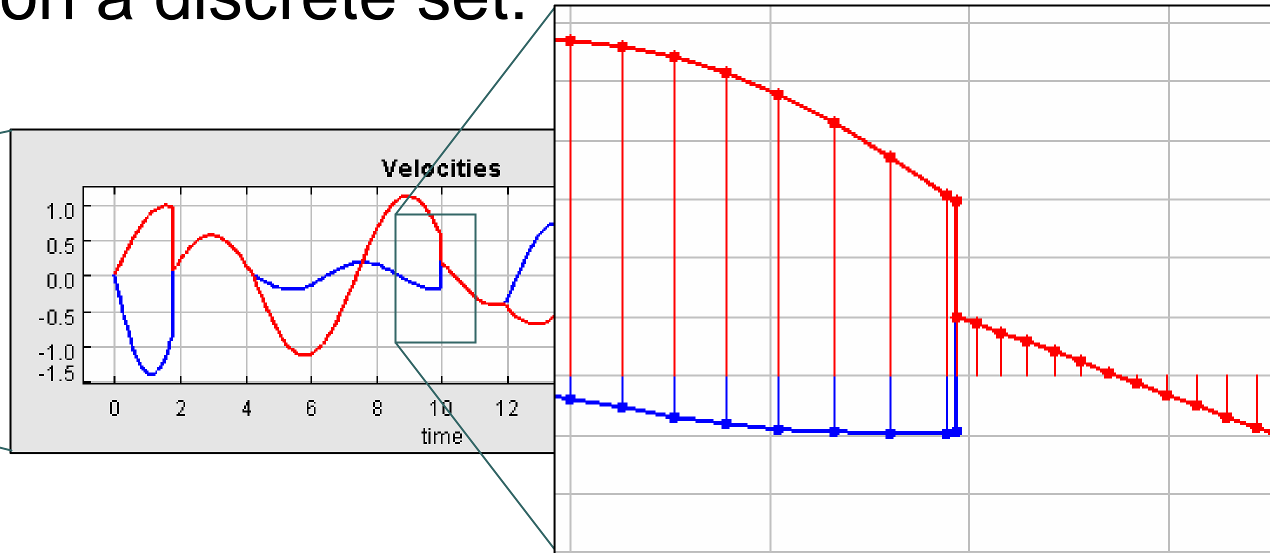
# Operational Semantics of Hybrid Systems

A computer execution of a hybrid system is constrained to provide values on a discrete set:

$$p\colon \mathbb{R}_+ \to \mathbb{R}^n$$

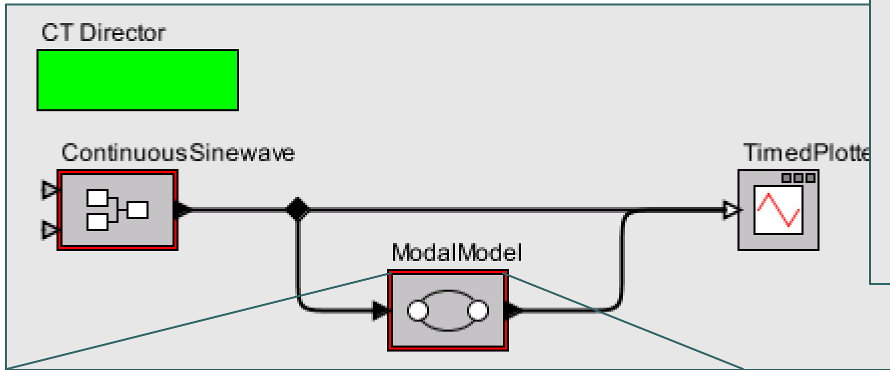
$$\dot{p}\colon \mathbb{R}_+ \to \mathbb{R}^n$$

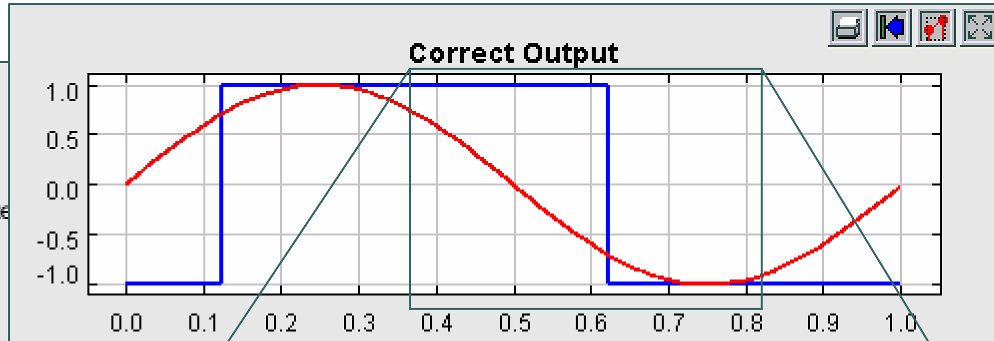
$$\ddot{p}\colon \mathbb{R}_+ \to \mathbb{R}^n$$

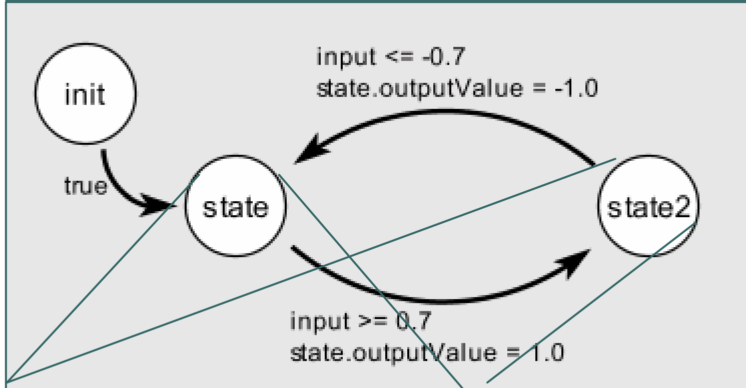



Given this constraint, choosing $T \subset \mathbb{R}$ as the domain of these functions is an unfortunate choice. It makes it impossible to unambiguously represent discontinuities.

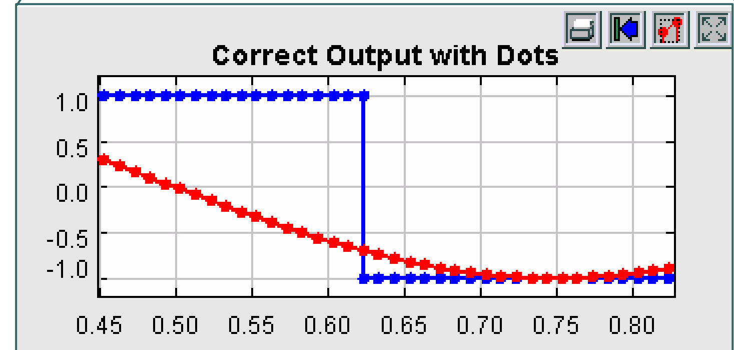
# Discontinuities Are Not Just Rapid Changes

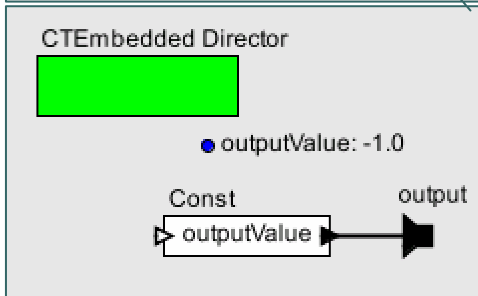


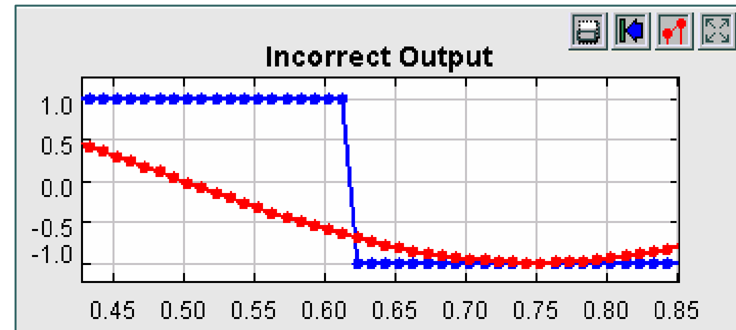




*Discontinuities must be semantically distinguishable from rapid continuous changes.*



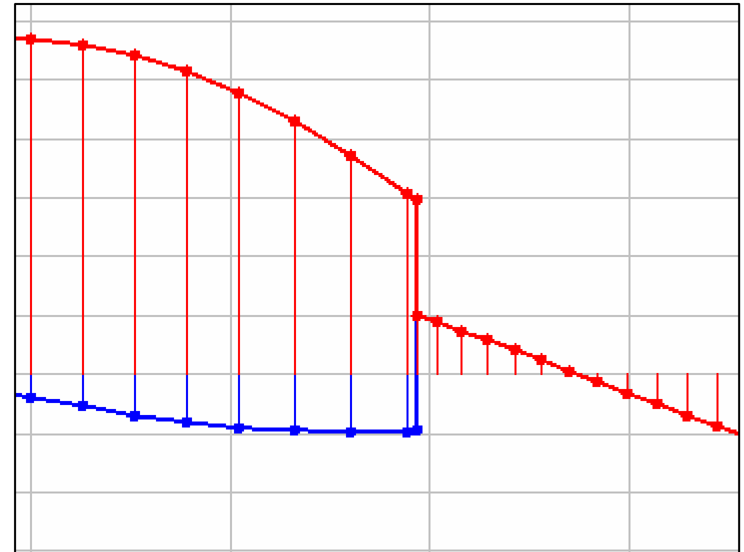



# Solution is the Same: Superdense Time

$$p\colon \mathbb{R}_+ \times \mathbb{N} \to \mathbb{R}^n$$

$$\dot{p}\colon \mathbb{R}_+ \times \mathbb{N} \to \mathbb{R}^n$$

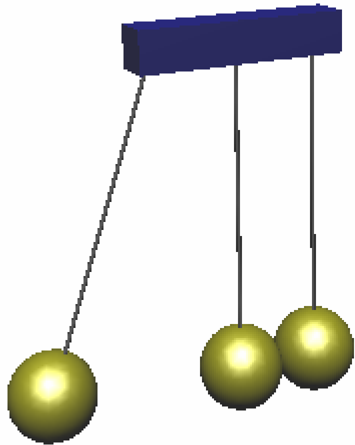
$$\ddot{p}\colon \mathbb{R}_+ \times \mathbb{N} \to \mathbb{R}^n$$



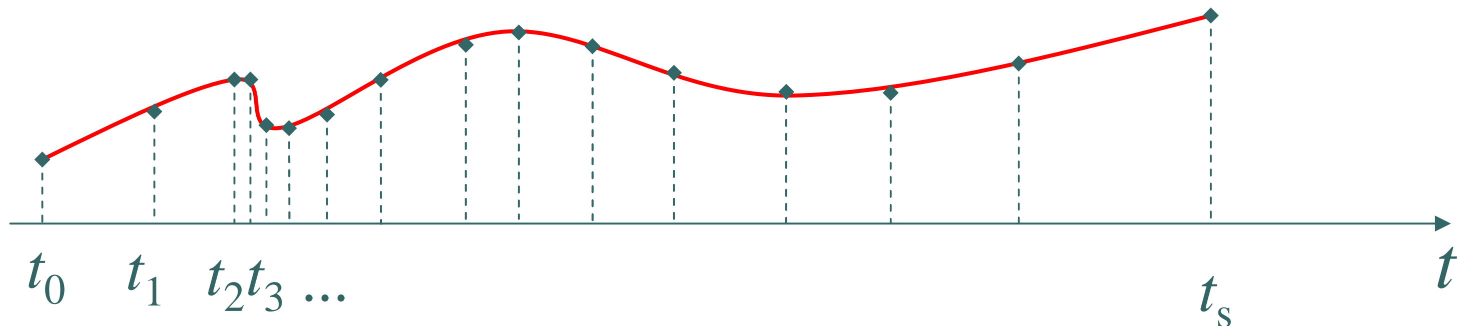
This makes it quite easy to construct models that combine continuous dynamics with discrete dynamics.

# Ideal Solver Semantics

[Liu and Lee, HSCC 2003]



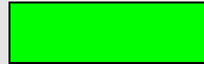
In the *ideal solver semantics,* an ODE governing the hybrid system has a unique solution for intervals $[t_i, t_{i+1})$, the interval between discrete time points. A discrete trace loses nothing by not representing values within these intervals.




Common fixed point semantics enables hybrid discrete/continuous models.

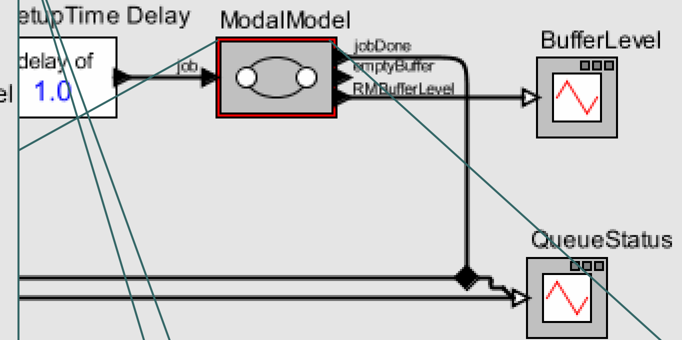
# The Hybrid Plant Model

DE Director

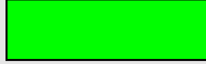


Hybrid model of a plant where a continuous flow of raw material is directed into bottles as the bottles (jobs) arrive. The top level is a discrete-event model with a modal continuous-time model ... right.

- jobArrivalRate: 1.0/3.0 jobs/minute
- setupTime: 1.0 minutes
- rawMaterialInflowRate: 1.5 liters/minute
- maxInflowRate: 2.0 liters/minute
- targetContainerLevel: 10.0 liters
- initialRawMaterialLevel: 3.0 liters

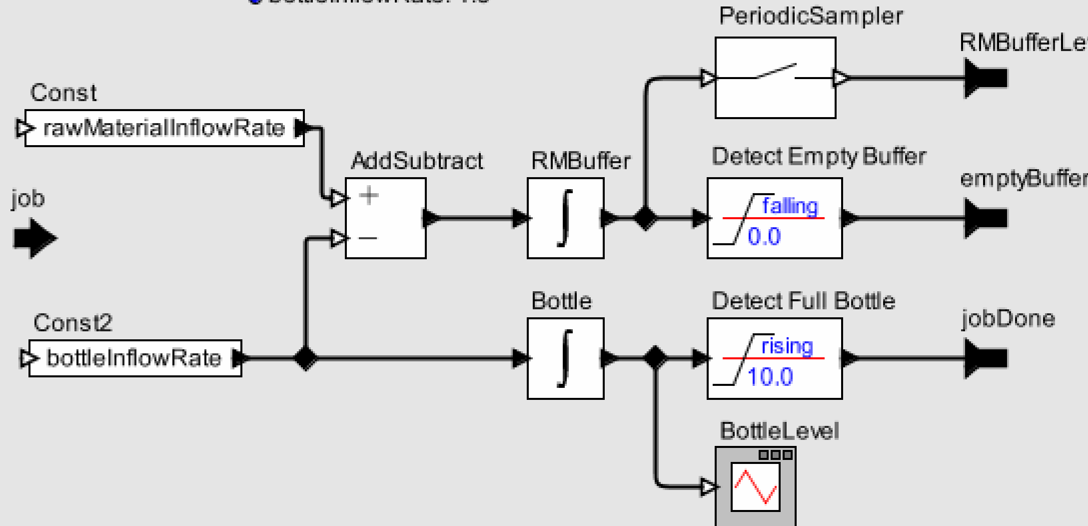



CT Director



Model of raw material buffer and bottle being filled.

- bottleInflowRate: 1.5




true
processing.RMBuffer.initialState = initialRawMaterialLevel;
processing.Bottle.initialState = 0.0;
processing.bottleInflowRate = 0.0

processing.Bottle.initialState = 0.0

Not filling a bottle. Collecting raw material.

idle

job_isPresent
processing.bottleInflowRate = maxInflowRate

jobDone_isPresent
processing.bottleInflowRate = 0.0;
processing.Bottle.initialState = 0.0

inflowRate

maxRate

Filling the bottle at the in-flow rate (raw material buffer is empty).

Filling the bottle at the maximum rate.

emptyBuffer_isPresent && !jobDone_isPresent
processing.bottleInflowRate = rawMaterialInflowRate

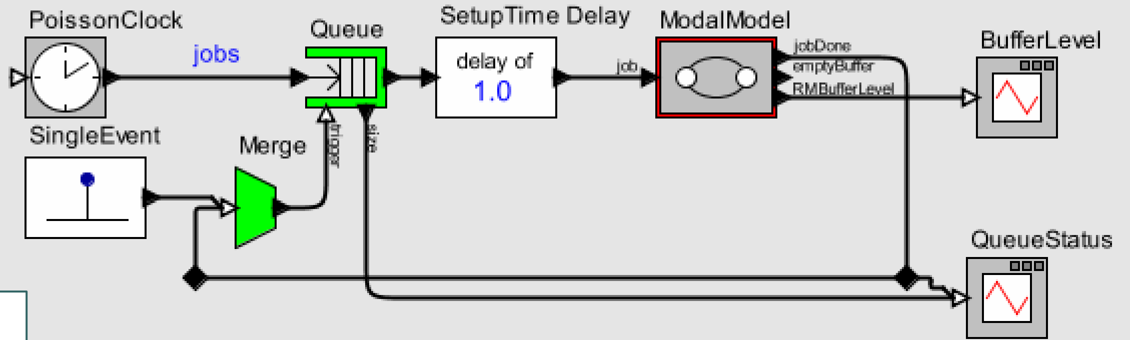
This model is a hierarchical mixture of three models of computation (MoCs): DE, FSM, and CT.

# The Hybrid Plant Model

DE Director

Hybrid model of a plant where a continuous flow of raw material is directed into bottles as the bottles (jobs) arrive. The top level is a discrete-event model with a modal continuous-time model inside. Model parameters are to the right.

- jobArrivalRate: 1.0/3.0 jobs/minute
- setupTime: 1.0 minutes
- rawMaterialInflowRate: 1.5 liters/minute
- maxInflowRate: 2.0 liters/minute
- targetContainerLevel: 10.0 liters
- initialRawMaterialLevel: 3.0 liters




init

true
processing.RMBuffer.initialState = initialRawMaterialLevel;
processing.Bottle.initialState = 0.0;
processing.bottleInflowRate = 0.0

jobDone_isPresent
processing.bottleInflowRate = 0.0;

Not filling a bottle.
Collecting raw material.

idle

job_isPresent
processing.bottleInflowRate = maxInflowRate

jobDone_isPresent
processing.bottleInflowRate = 0.0;
processing.Bottle.initialState = 0.0

inflowRate

maxRate

Filling the bottle at the maximum rate.

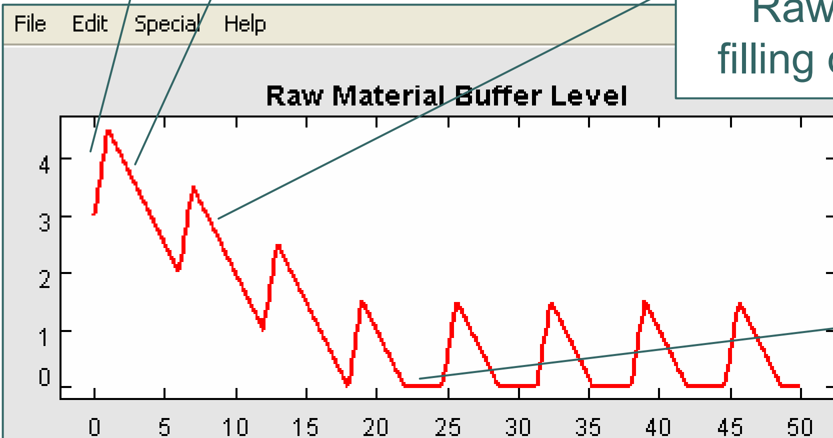
Done_isPresent
awMaterialInflowRate

Raw material buffer filling during setup time

Bottle filling at maximum rate

Raw material buffer filling during setup time

Bottle filling at limited rate



# Conclusions

*We have given a rigorous semantics to discrete-event systems that leverages principles from synchronous/reactive languages and admits interoperability with both SR and continuous-time models.*

# Further Reading


[1] X. Liu and E. A. Lee, "CPO Semantics of Timed Interactive Actor Networks," UC Berkeley, Berkeley, CA, Technical Report EECS-2006-67, May 18 2006.

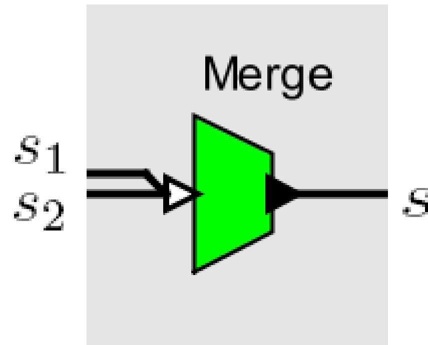
[2] X. Liu, E. Matsikoudis, and E. A. Lee, "Modeling Timed Concurrent Systems," in CONCUR 2006 - Concurrency Theory, Bonn, Germany, 2006.

[3] A. Cataldo, E. A. Lee, X. Liu, E. Matsikoudis, and H. Zheng, "A Constructive Fixed-Point Theorem and the Feedback Semantics of Timed Systems," in Workshop on Discrete Event Systems (WODES), Ann Arbor, Michigan, 2006.

[4] E. A. Lee, "Modeling Concurrent Real-time Processes Using Discrete Events," Annals of Software Engineering, vol. 7, pp. 25-45, March 4th 1998 1999.

[5] E. A. Lee, H. Zheng, and Y. Zhou, "Causality Interfaces and Compositional Causality Analysis," in *Foundations of Interface Technologies (FIT), Satellite to CONCUR*, San Francisco, CA, 2005.

# Semantics of Merge



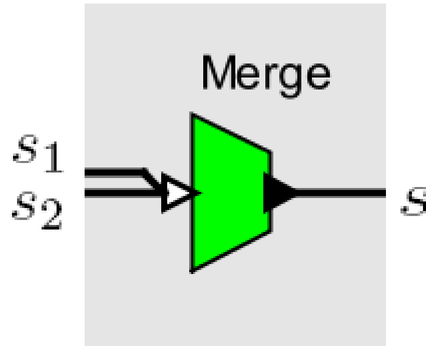

At time $t$, input sequences are interleaved. That is, if the inputs are $s_1$ and $s_2$ and

$$s_1(t,0) = v_1,$$
$$s_2(t,0) = w_1, \quad s_1(t,1) = w_2$$

(otherwise absent) then the output $s$ is

$$s(t,0) = v_1, \quad s(t,1) = w_1, \quad s(t,2) = w_2.$$

# Implementation of Merge




```
private List pendingEvents;
fire() {
  foreach input s {
    if (s is present) {
      pendingEvents.append(event from s);
    }
  }
  if (pendingEvents has events) {
    send to output (pendingEvents.first);
    pendingEvents.removeFirst();
  }
  if (pendingEvents has events) {
    post event at the next index on the event queue;
  }
}
```