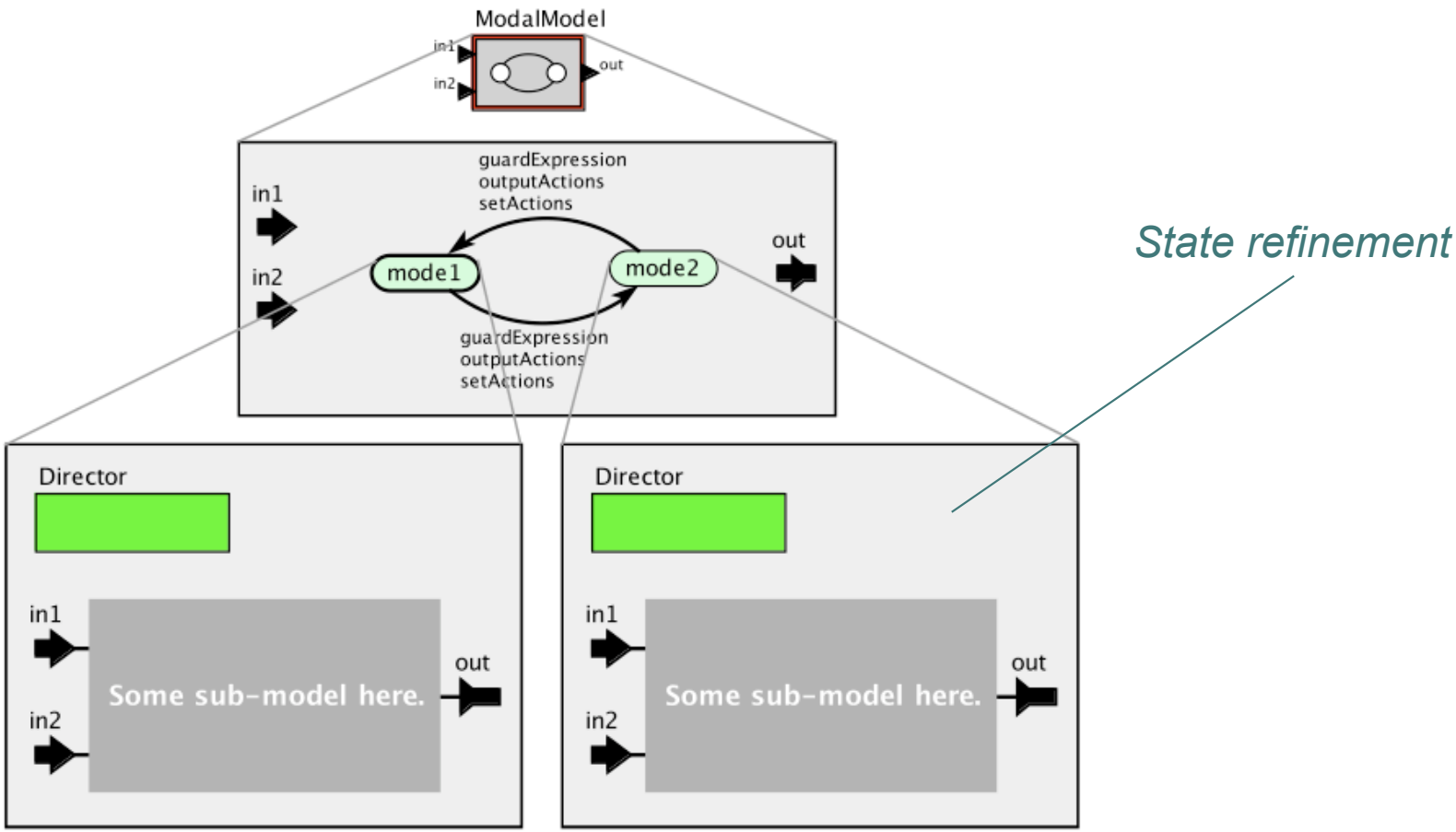
# Semantics of Modal Models in Ptolemy II

*Edward A. Lee*
*Stavros Tripakis*
*UC Berkeley*

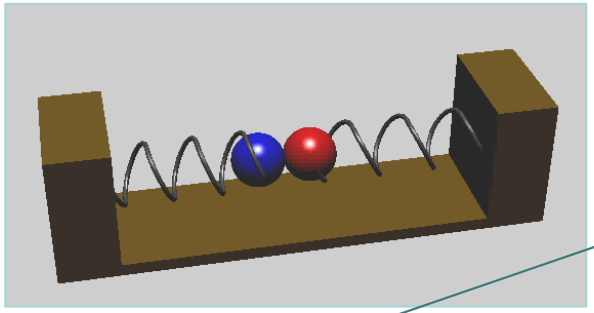
**Ninth Biennial Ptolemy Miniconference**
**February 16, 2011**

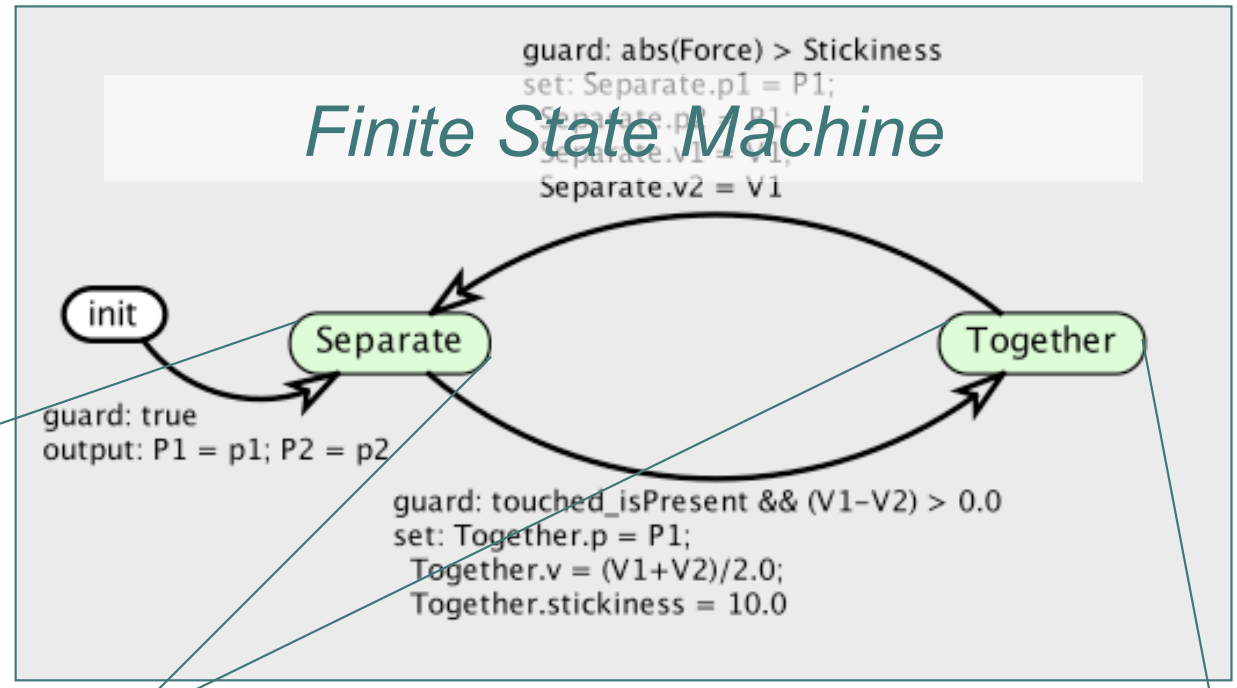
# What are modal models?

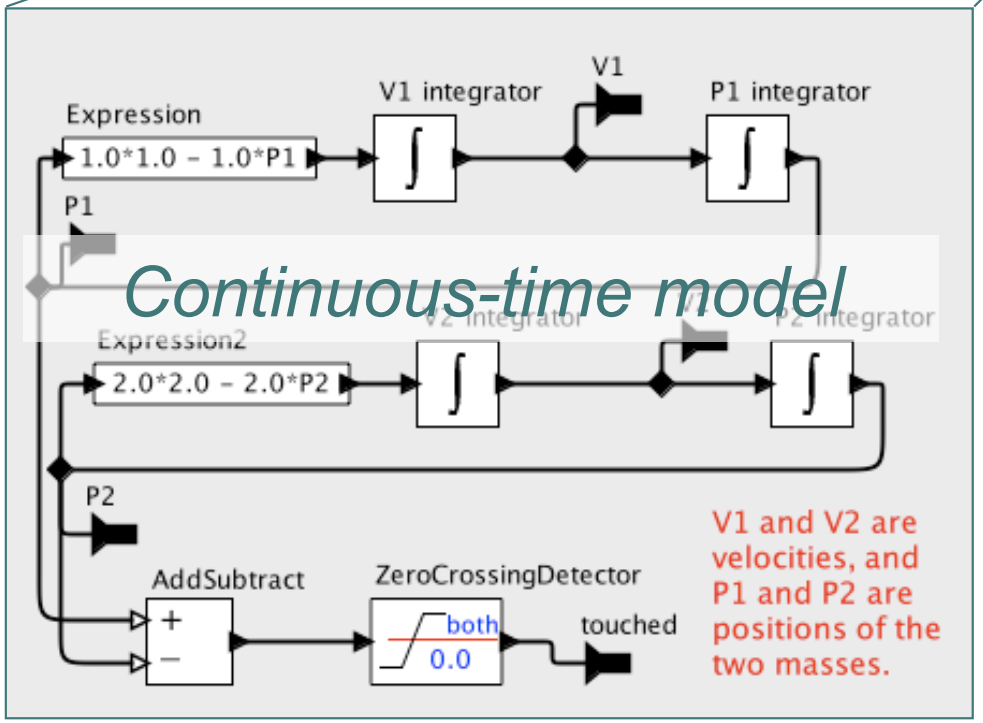
- Modal models = hierarchical models mixing FSMs (Finite State Machines) and other models

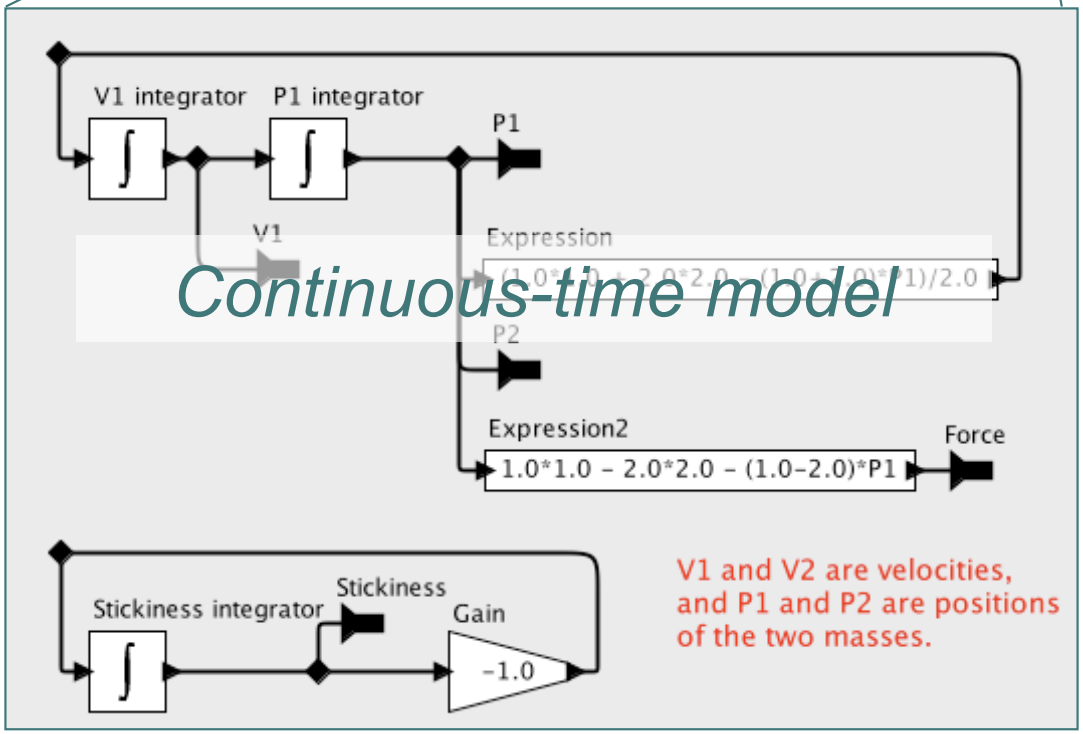


# Example: Hybrid System





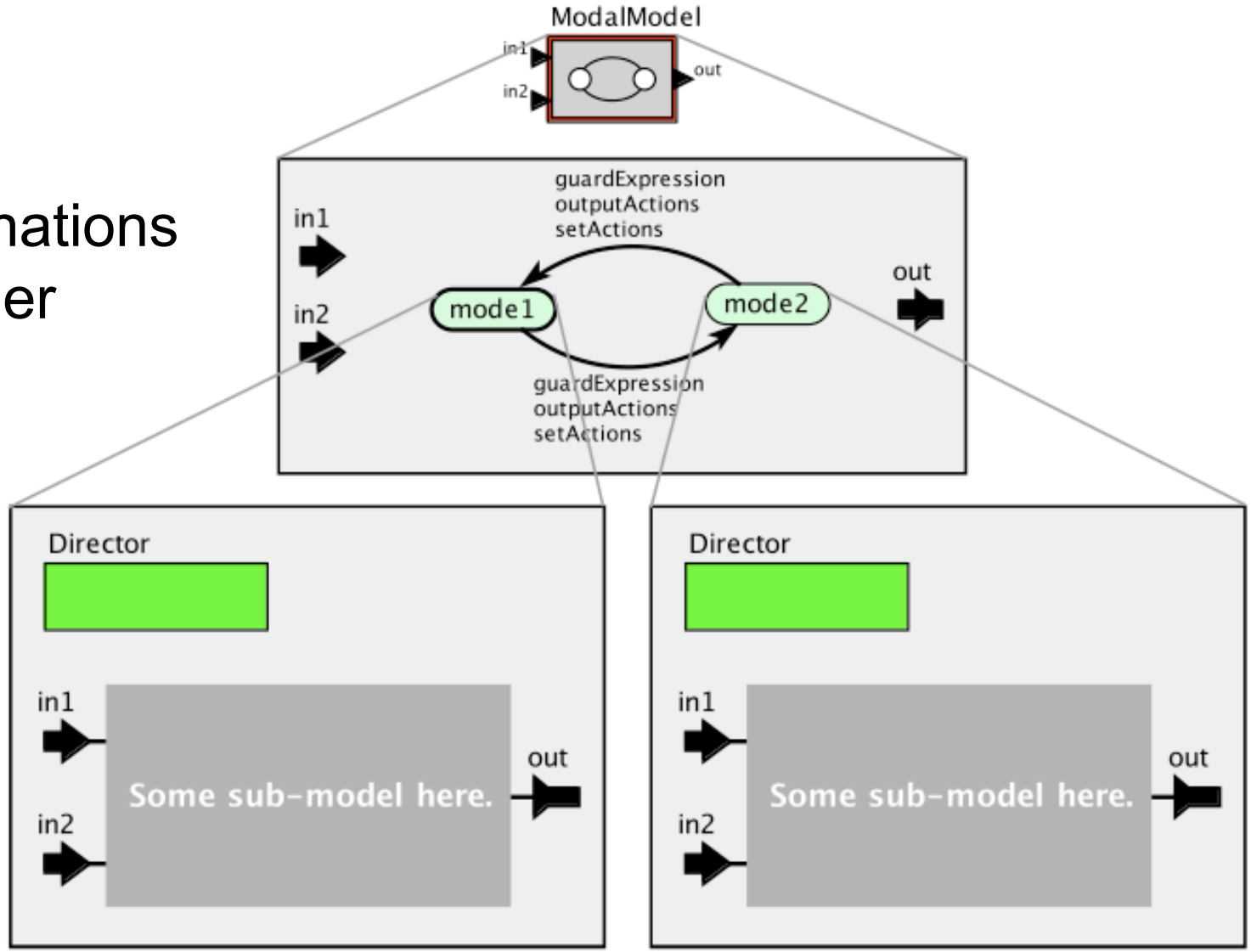



# Influences for this work

- Statecharts [Harel 87]
- Argos [Maraninchi 91]
- Esterel [Berry & Gonthier 92]
- Abstract state machines [Gurevich 93]
- Hybrid systems [Puri & Varaiya 94, Henzinger 99]
- Timed automata [Alur & Dill 94]
- SyncCharts [Andre 96]
- I/O Automata [Lynch 96]
- *Charts [Girault, Lee, & Lee 99]
- UML State machines

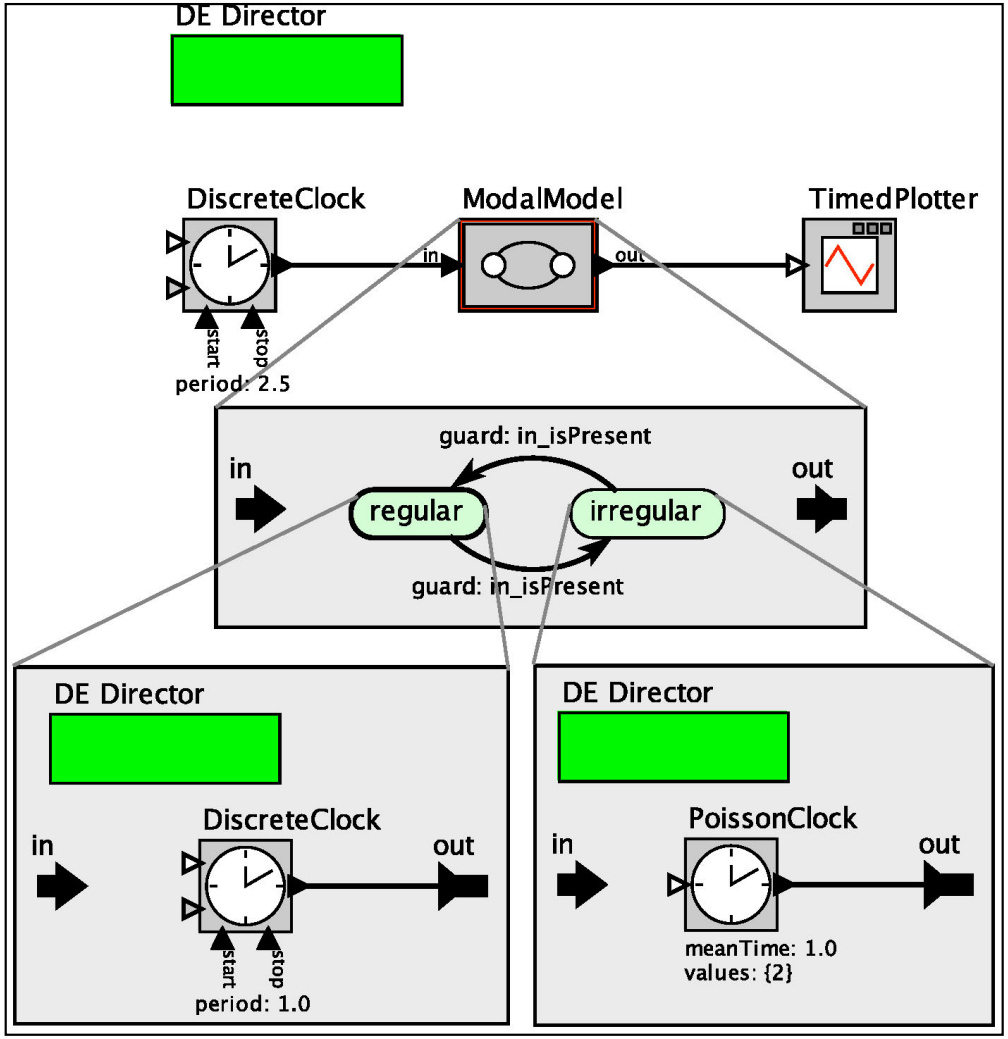
# Ptolemy II goes one step further

**Arbitrary** combinations of FSMs with other domains



# How to give meaning to modal models?

- Not always trivial:




*What happens to the events produced by the discrete clock while system is at mode "irregular"?*

# How to give meaning to modal models?

- Approach 1:
  - Give a meaning to every possible combination of models:
    - Hierarchical state machines (Statecharts, UML, …)
    - Timed automata (timed models within state machines)
    - Hybrid automata (continuous models within state machines)
    - Mode automata (synchronous/reactive within state machines)
    - …
  - Scalable?

# How to give meaning to modal models?

- Approach 2 [Ptolemy]:
  - Modular semantics
    - semantics of composite blocks = function of semantics of sub-blocks
  - Compositionality
  - Heterogeneity

# This talk

- A formal semantics for Ptolemy
  - operational semantics
    - close enough to the Java implementation to be faithful
    - but not too close (fits in a few pages)
- A formal semantics for modal models

# A FORMAL SEMANTICS FOR PTOLEMY

# Abstract semantics

- Actor = State Machine
- Actor = Inputs + Outputs + States + Initial state
  + Fire + Postfire

- Fire = output function: produces outputs given current inputs + state

$$F : S \times I \rightarrow O$$

- Postfire = transition function: updates state given current inputs + state

$$P : S \times I \rightarrow S$$

# Implemented as Java interfaces

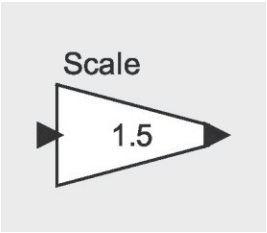
*Interface "Initializable"*

| Method Summary | |
|---|---|
| void | addInitializable(Initializable initializable)<br>Add the specified object to the list of objects whose preinitialize(), intialize(), and wrapup() methods should be invoked upon invocation of the corresponding methods of this object. |
| void | initialize()<br>Begin execution of the actor. |
| void | preinitialize() |

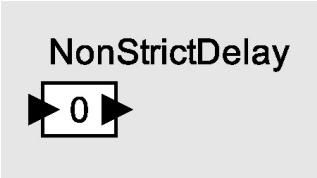
*Interface "Executable"*

| Method Summary | |
|---|---|
| void | fire()<br>Fire the actor. |
| boolean | isFireFunctional()<br>Return true if this executable does not change state in either the prefire() or the fire() method. |
| boolean | isStrict()<br>Return true if this executable is strict, meaning all inputs must be known before iteration. |
| int | iterate(int count)<br>Invoke a specified number of iterations of the actor. |
| boolean | postfire()<br>This method should be invoked once per iteration, after the last invocation of fire() in that iteration. |
| boolean | prefire()<br>This method should be invoked prior to each invocation of fire(). |
| void | stop()<br>Request that execution of this Executable stop as soon as possible. |
| void | stopFire()<br>Request that execution of the current iteration complete. |
| void | terminate()<br>Terminate any currently executing model with extreme prejudice. |

# Examples




*Single state ("stateless").*
*P : trivial (state never changes).*
*F : out := in*1.5*




*State: the current value*
*F : out := state*
*P: state := input*

# Behaviors – untimed

$$F : S \times I \rightarrow O$$
$$P : S \times I \rightarrow S$$

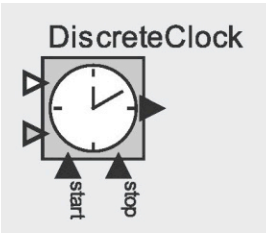
- Set of untimed traces:

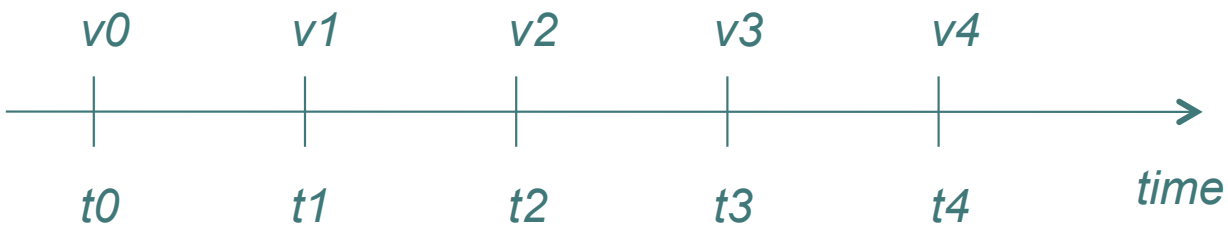
$$s_0 \xrightarrow{x_0, y_0} s_1 \xrightarrow{x_1, y_1} s_2 \xrightarrow{x_2, y_2} \cdots$$

- such that for all i :

$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i, x_i)$$

# What about “timed” actors?




- States include special ***timer*** variables:
  - Set to some positive value, “expire” when they reach 0, can be “frozen” and “resumed”, …

# Behaviors – timed

$$F : S \times I \rightarrow O$$
$$P : S \times I \rightarrow S$$

- Set of timed traces:

Delays

$$s_0 \xrightarrow{x_0, y_0, d_0} s_1 \xrightarrow{x_1, y_1, d_1} s_2 \xrightarrow{x_2, y_2, d_2} \cdots$$

- such that for all i :

Decrement timers by $d_i$

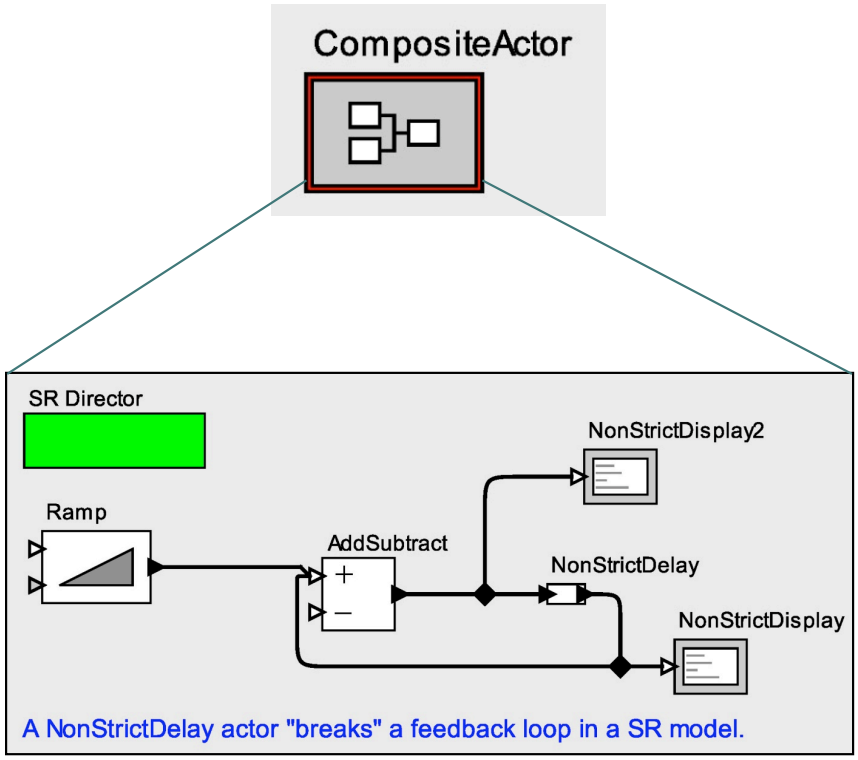
$$y_i = F(s_i, x_i)$$
$$s_{i+1} = P(s_i - d_i, x_i)$$
$$d_i \leq \min\{v \mid v \text{ is the value of a timer in } s_i\}$$

# What about hierarchy?

How to give semantics to a hierarchical model?

i.e.,

How to give semantics to a composite actor?



# Directors = composition operators

- Given a composite actor with a set of subactors
  *A1, A2, …,* with fire & postfire functions
  *F1/P1, F2/P2, …*

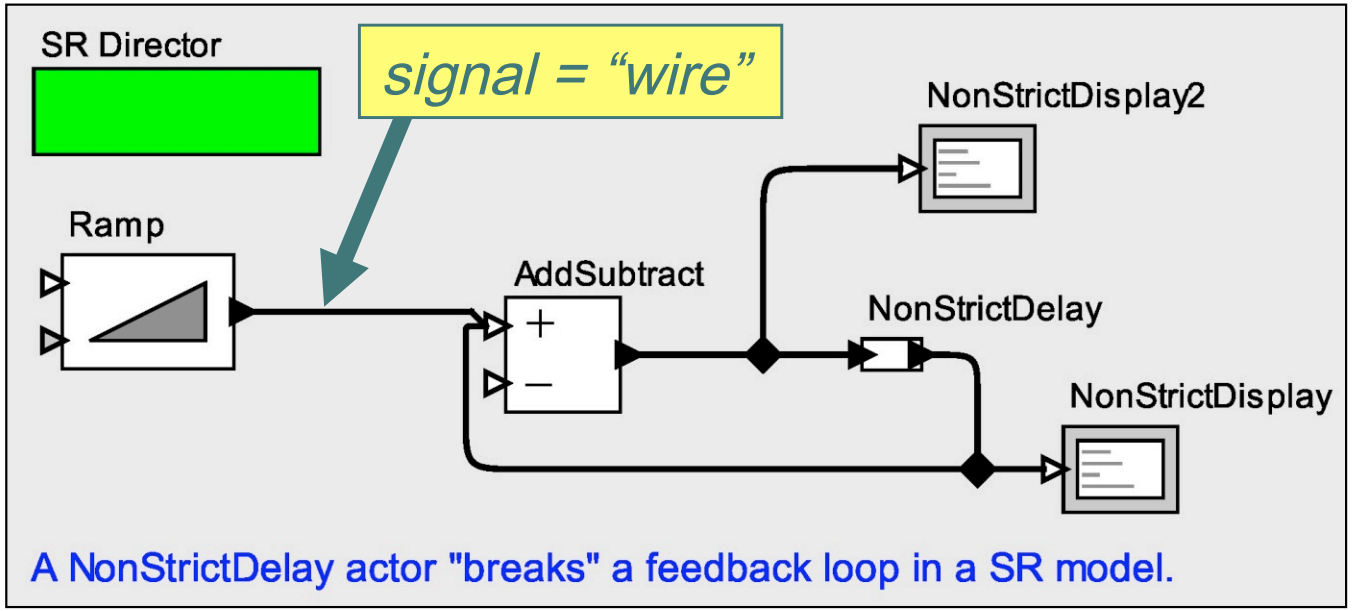
- … its director defines a new pair of fire & postfire functions *F* and *P.*

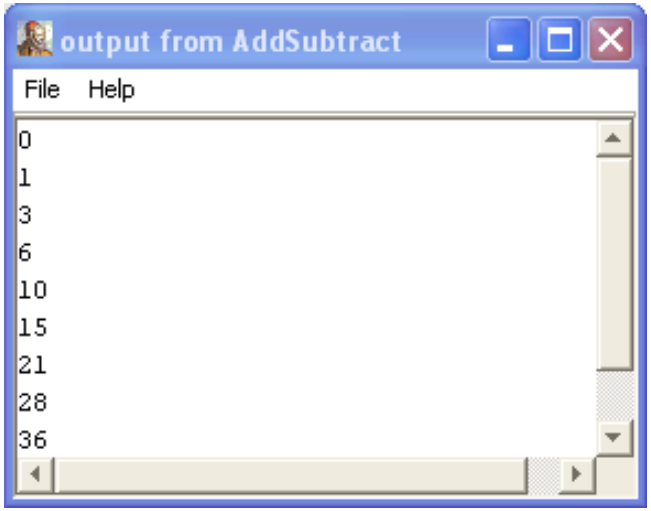
- *F* and *P* define a new, composite actor *A.*

- *A* can be used like an atomic actor (black-box).

*Modular, compositional semantics*

# Example: Synchronous/Reactive (SR)

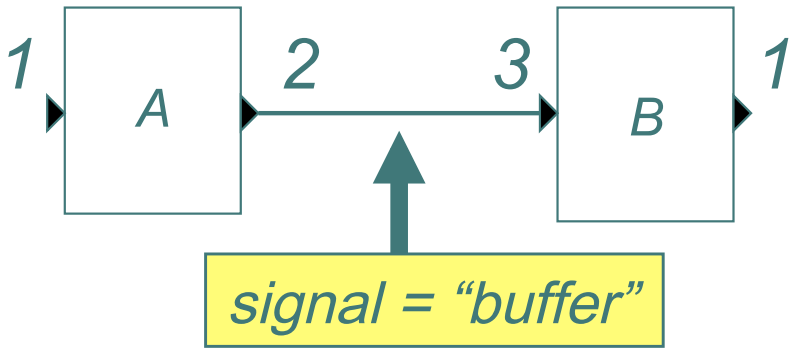


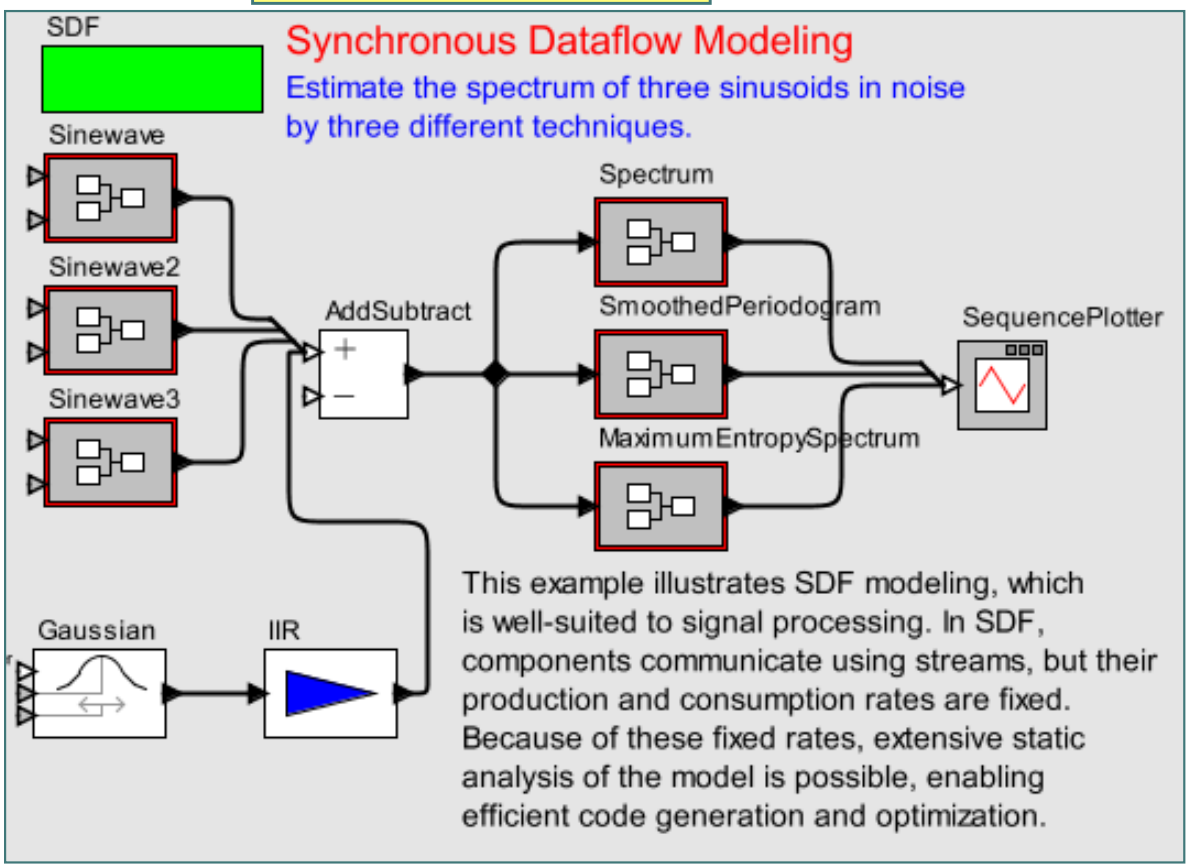

- *To compute the composite F, director solves a **fixpoint**:*
  - *Keep on evaluating Fi’s until the values of all signals stabilize (this includes output signals in particular)*
  - *State remains unchanged during computation of the fixpoint!*

    *c.f. separation of fire and postfire*
- *To compute the composite P, just execute all Pi’s*

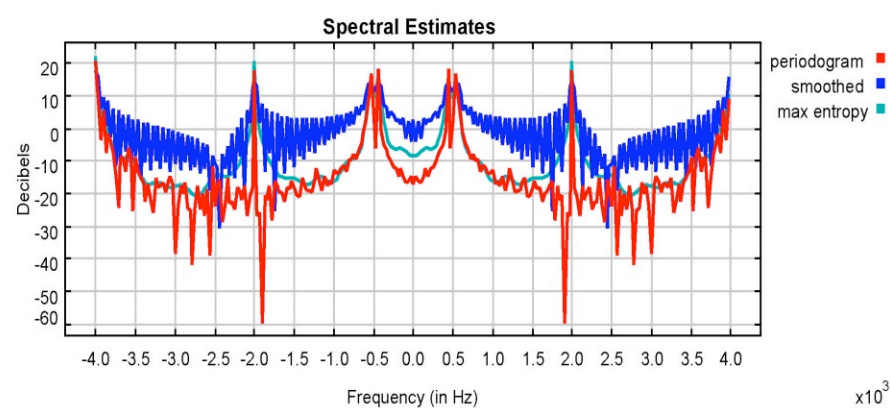
# Example: Synchronous Data Flow (SDF)




In each firing, actors consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.



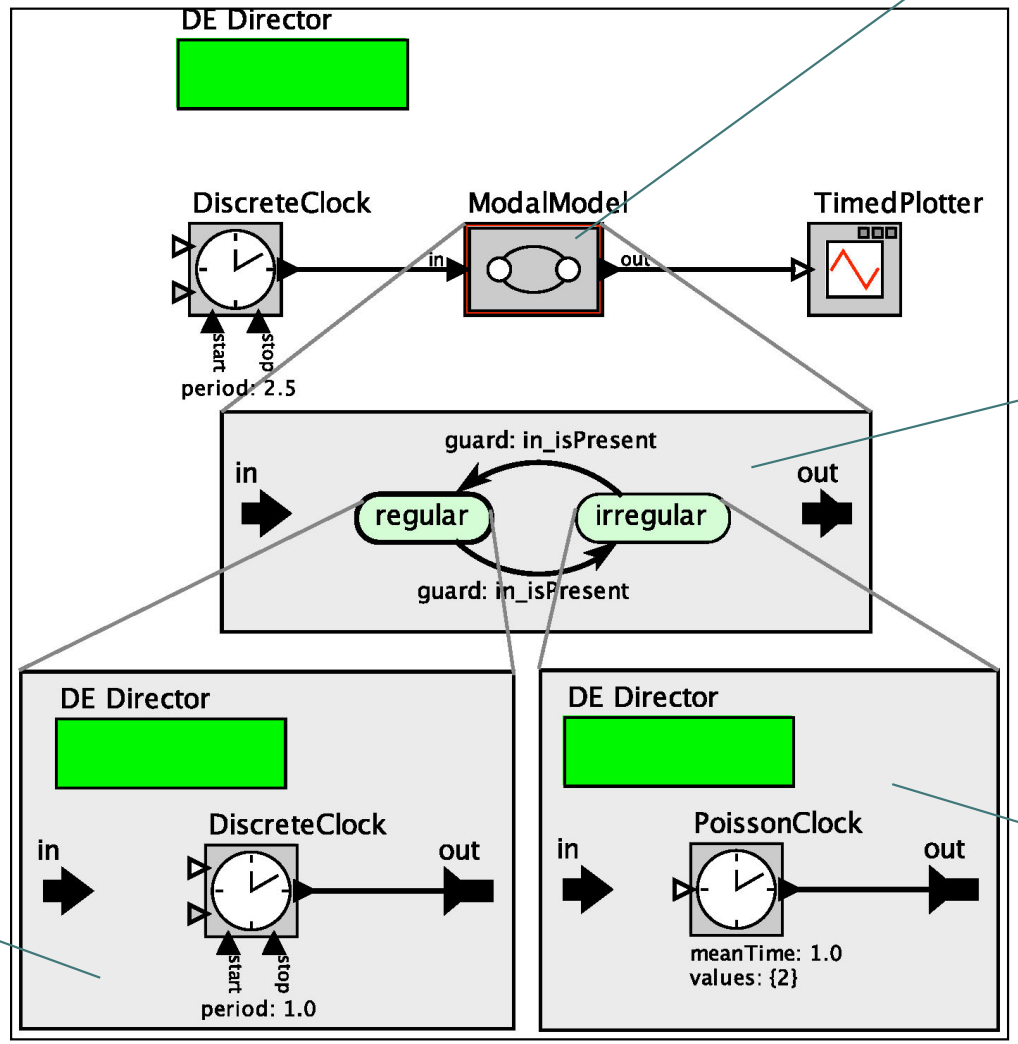

- *SDF Director computes periodic schedule, e.g., A,A,A,B,B*
- *Composite fire() fires all internal actors according to schedule*



# MODAL MODEL SEMANTICS

# Giving semantics to modal models

Goal:
define $F, P$ functions
for the modal model

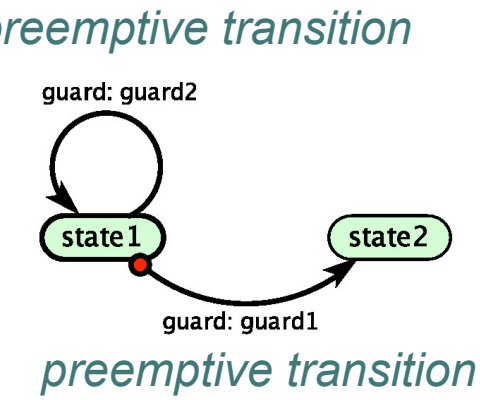



$F_c, P_c$ functions
already defined
for the "controller"
automaton

$F_1, P_1$ functions
already defined
for this refinement

$F_2, P_2$ functions
already defined
for this refinement

# Rough description of semantics



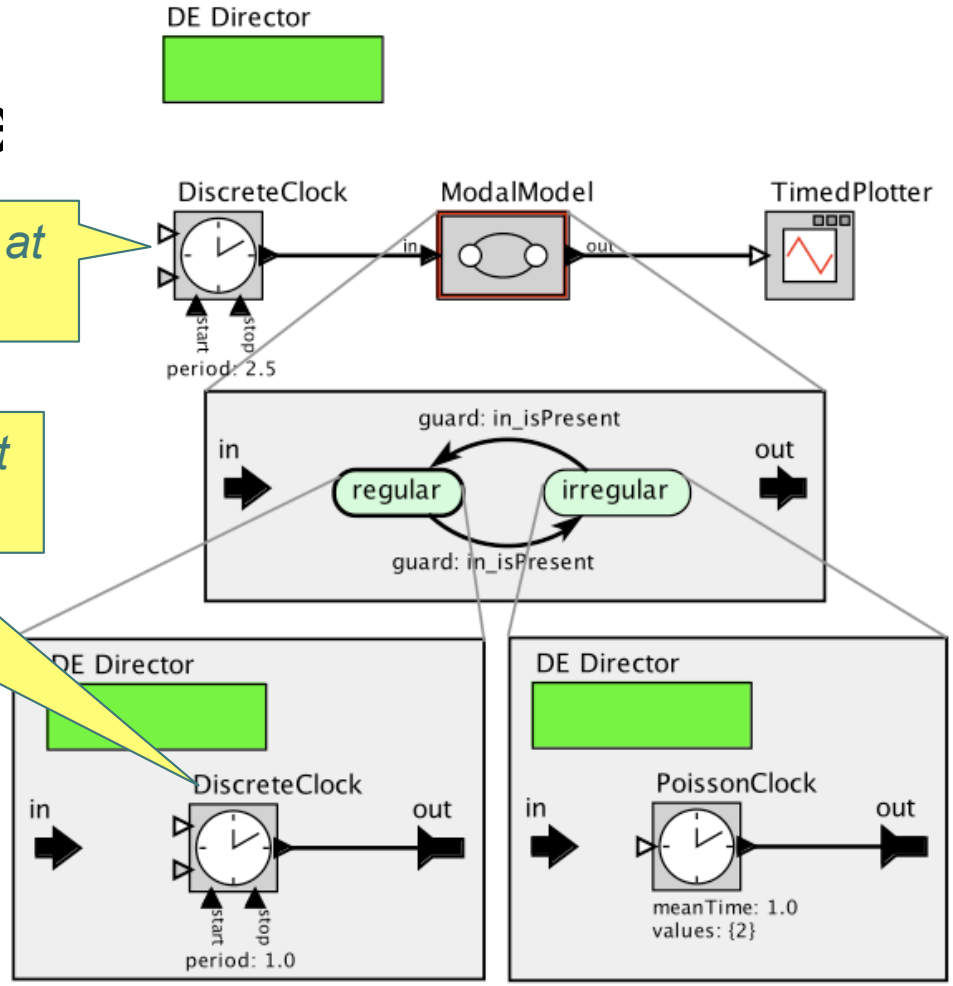

- Given current controller state *si* :
- If no outgoing transitions from *si* are enabled:
  - Use *Fi* and *Pi* to compute *F* and *P*
- If preemptive outgoing transitions from *si* are enabled:
  - Use the actions of these transitions to compute *F* and *P*
- If only non-preemptive outgoing transitions from *si* are enabled:
  - First fire refinement, then transition, i.e.:
    - *F* is the composition of *Fi* and the output action of a transition
    - *P* is the composition of *Pi* and the state update action of a transition
- Timers of refinements suspended and resumed when exiting/entering states
- Details in paper “Modal Models in Ptolemy” [EOOLT 2010]

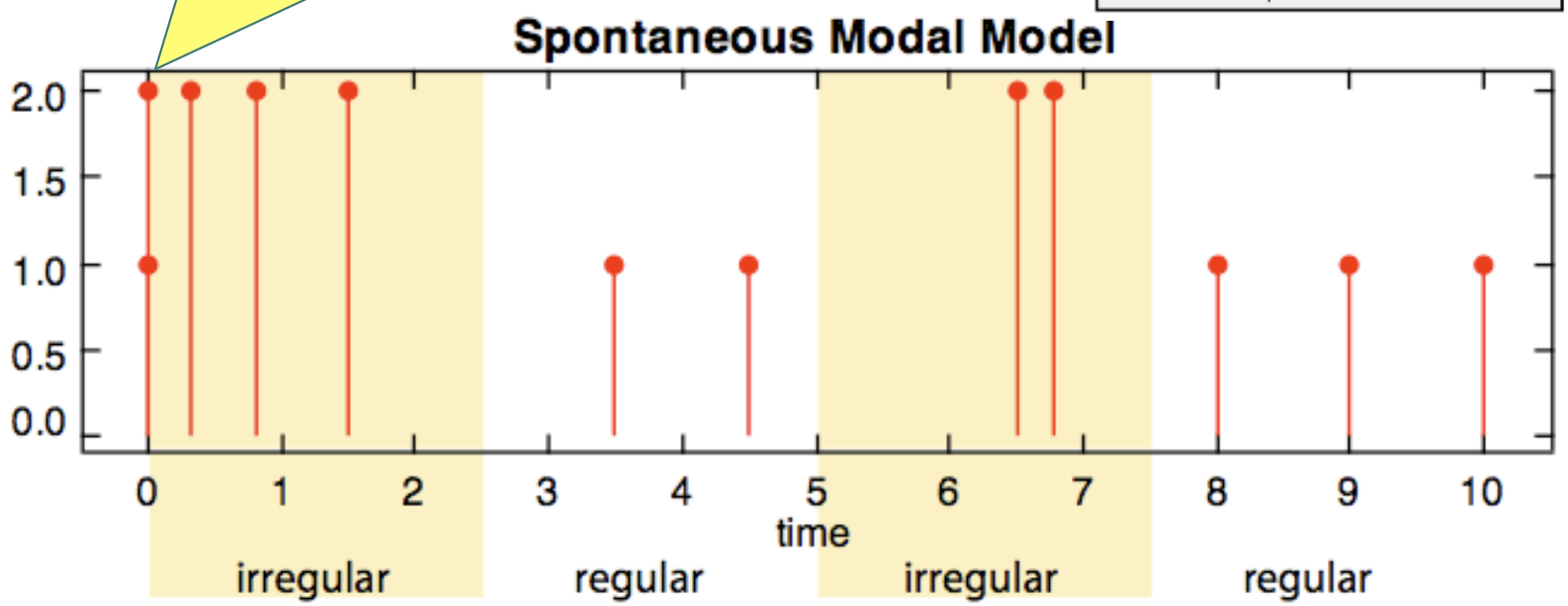
# Timed modal model example

*Mode transitions triggered at times 0, 2.5, 5, 7.5, etc.*

*Events with value 1 produced at (local times) 0, 1, 2, 3, etc.*

*First regular event generated at (global time) 0, then transition is immediately taken.*



# Conclusions, ongoing work and future challenges

- Modular formal semantics for Ptolemy II
  - Directors = composition operators over state machines
- Semantics worked out for modal models [EOOLT 2010]
  - Currently extending it to other domains: Synchronous-Reactive, SDF, Discrete-Event, Continuous-Time, …
- Meta-model to describe semantics?

# Thank you

- Questions?