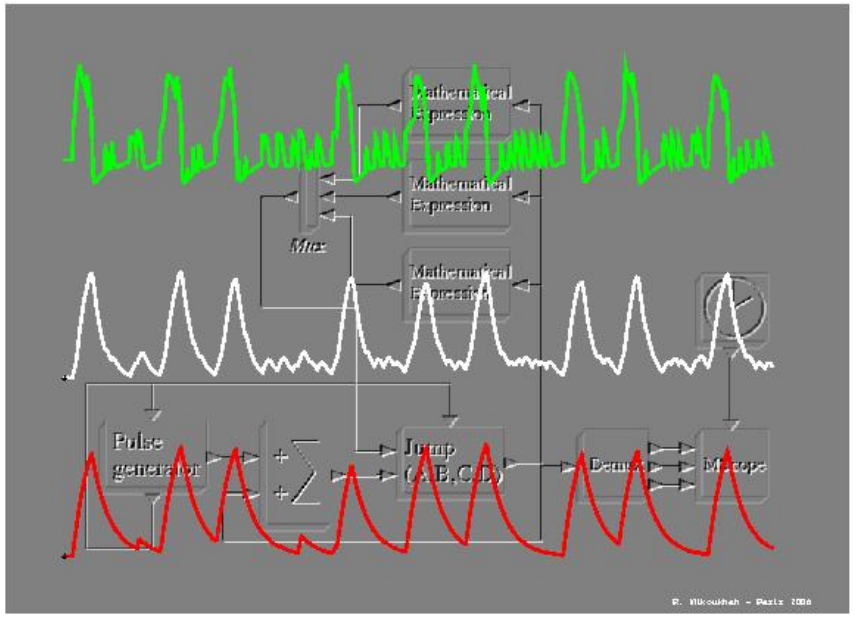
Activation Inheritance in Modelica

Ramine Nikoukhah

Scicos Project

July 8, 2008



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche PARIS - ROCQUENCOURT

Should all events be considered synchronous

Consider multi-rate systems

Different system components run at different frequencies

Parts of the system can run on conditional bases

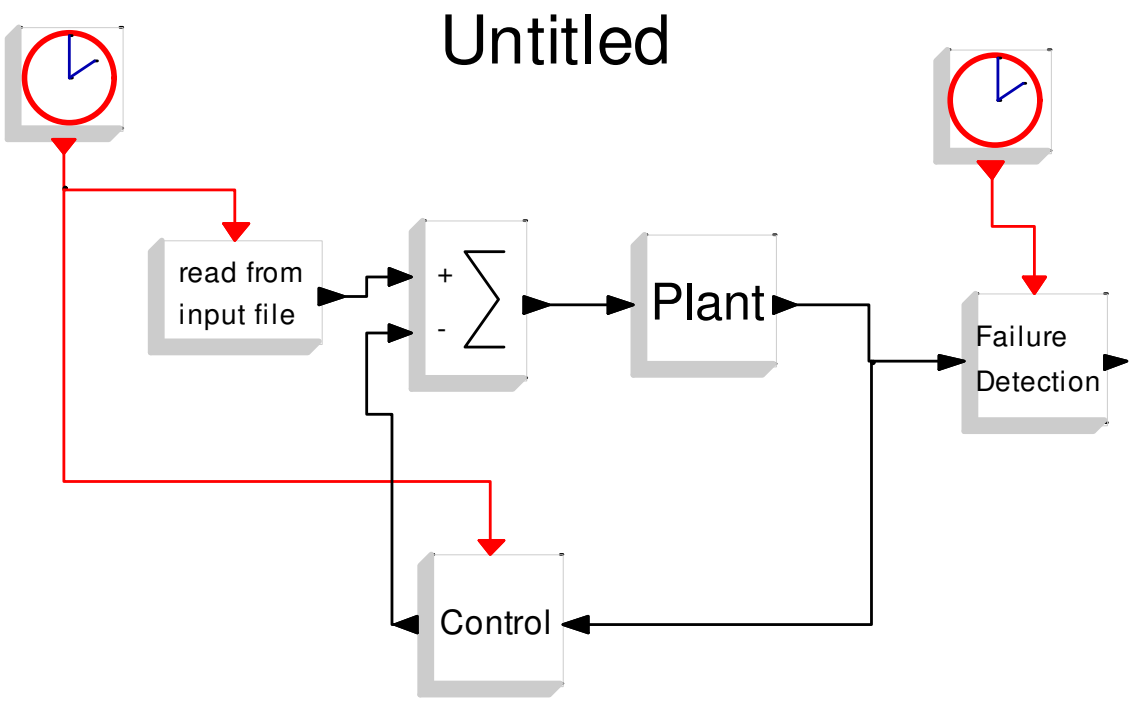
Full or partial synchronization may or may not be needed

How to have flexibility to:

Leave different components asynchronous

Impose easily synchronism when needed

Example



Synchronism is not needed in this situation
Imposing it creates unneeded constraints

Why not impose synchronization?

Due to numerical errors in solvers, zero-crossing times are never exact

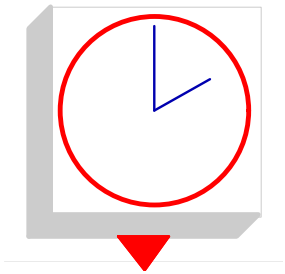
Counting on simultaneous zero-crossing detections only increases non-determinism

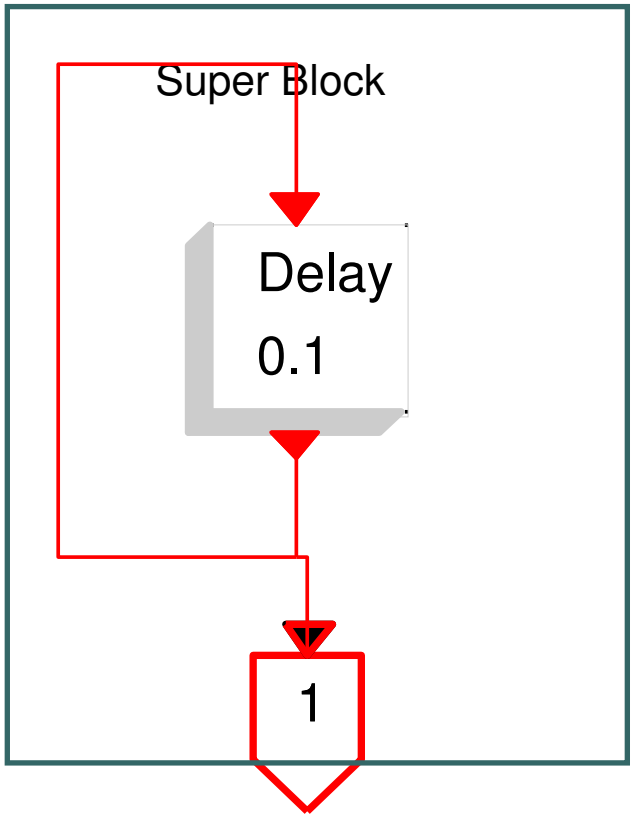
In most cases, such synchronisms are unwanted; user does (should) not count on them

They lead to an exponentially growing number of event scenarios: virtually impossible to generate efficient static code

Event clocks in Scicos

Event clocks are not basic blocks

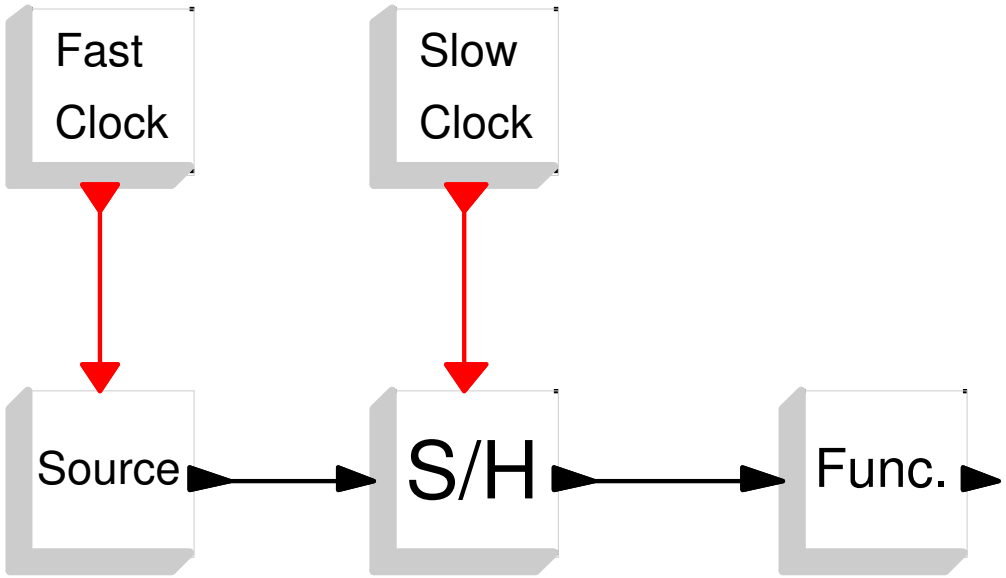




Two Event clocks do not generate synchronous events

Synchronization problems to avoid

Incorrect way of implementing decimation



Activation sources generate asynchronous events => order of block execution is not predictable.

Synchronization problems to avoid

Use frequency division:

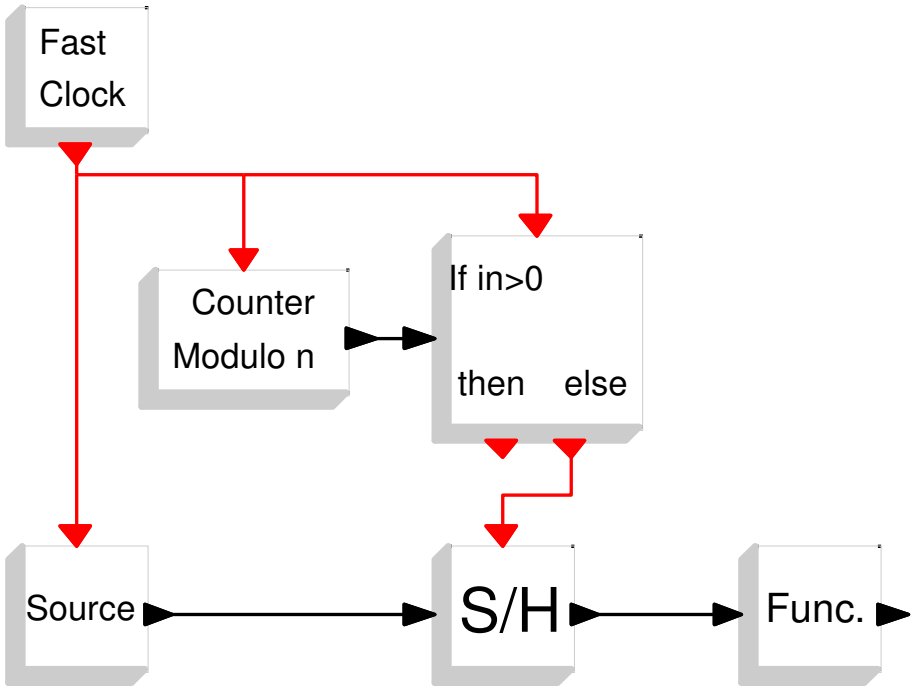
Combination Counter Modulo and If-then-else => frequency division.

Division factor set by fixing the value of n, and the phase by the initial state of the counter.

The freq_div (Super) block, available in the Events palette, is constructed this way.

Synchronization problems to avoid

Correct way of implementing decimation

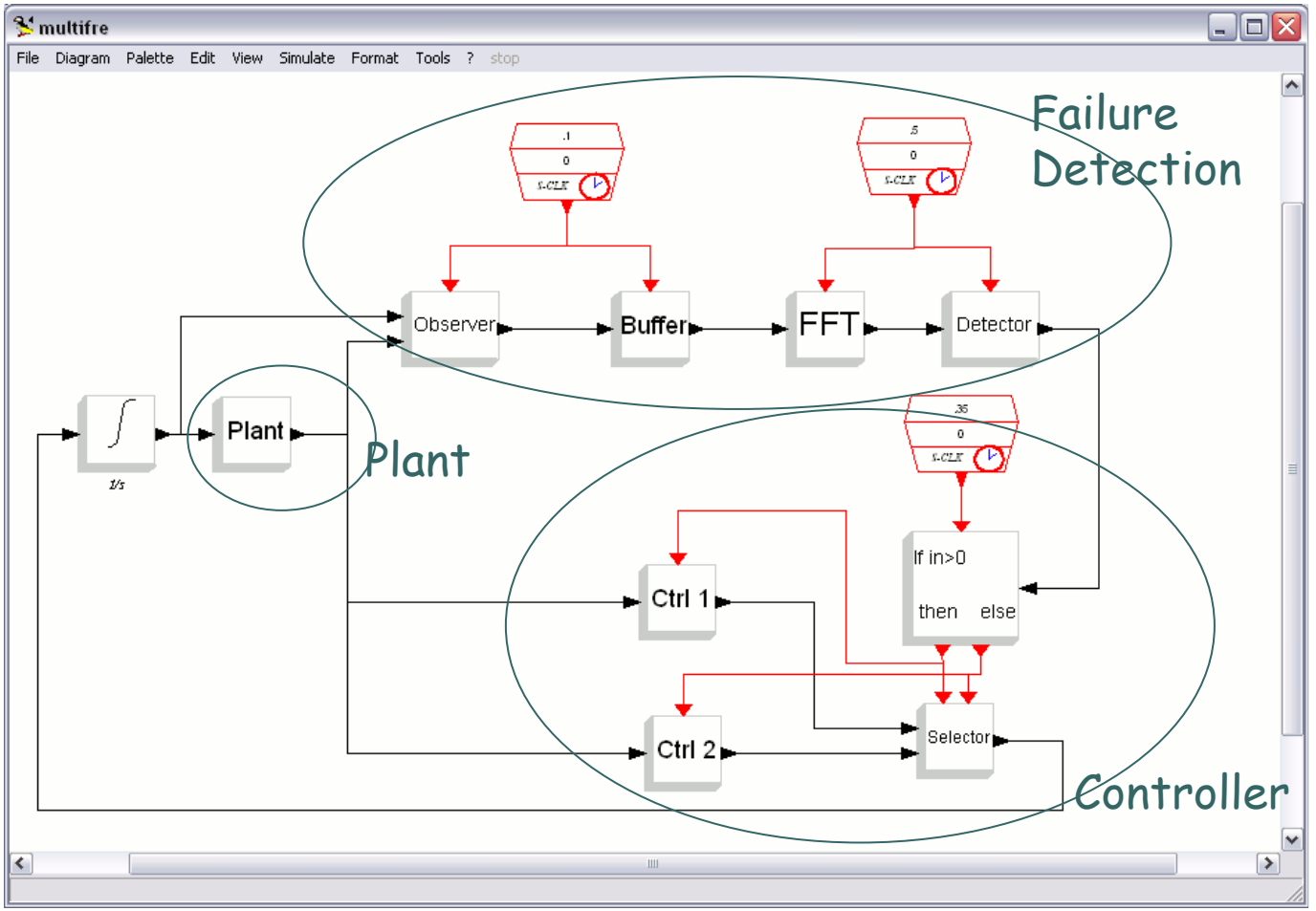


Synchronized events at different frequencies can be implemented by sub-sampling the fast clock

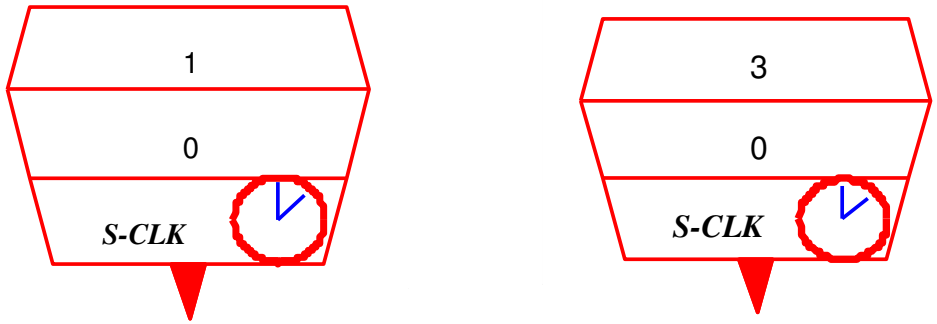
Example of a multirate system

Systems diagnosis with controller reconfiguration

Synchronization is required



Sample clocks



Virtual blocks, replaced by one Event clock and sub-sampling

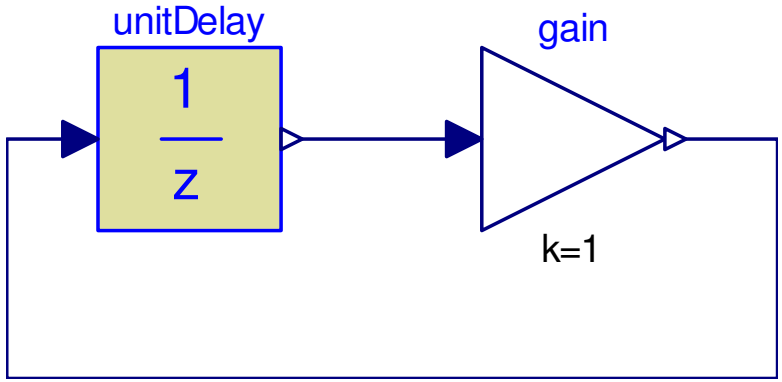
Uses slowest clock generating all events using sub-sampling: modulo counter and conditional blocks

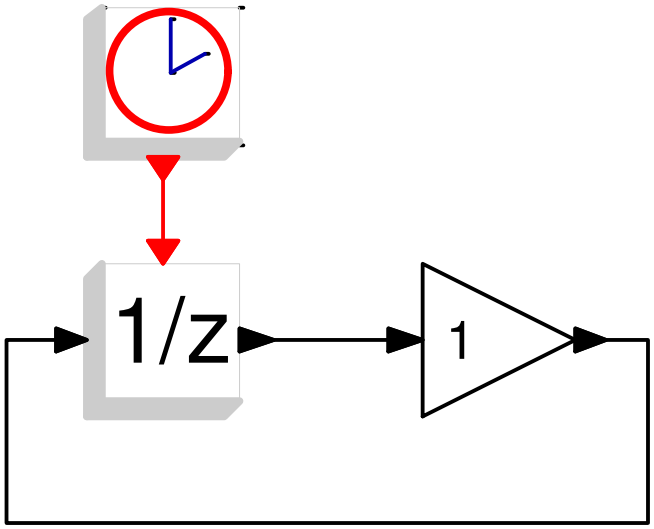
Clock algebra similar to Simulink

Resulting events are synchronized

Transparent to the user

Modelica vs Scicos

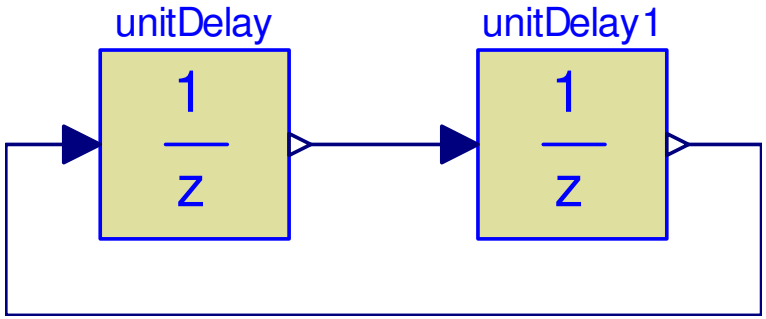


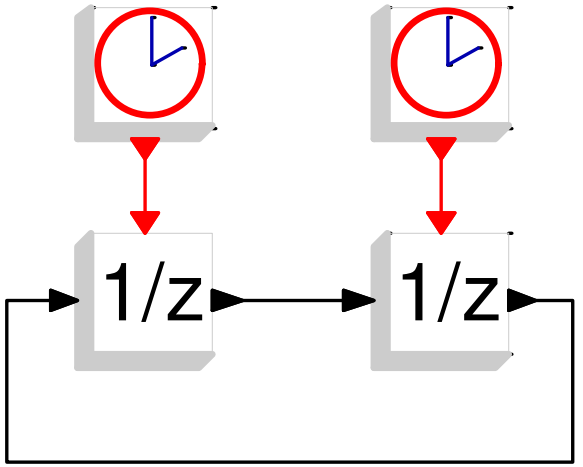


In this case Event Clock or Sample Clock can be used in Scicos

No synchronization problem

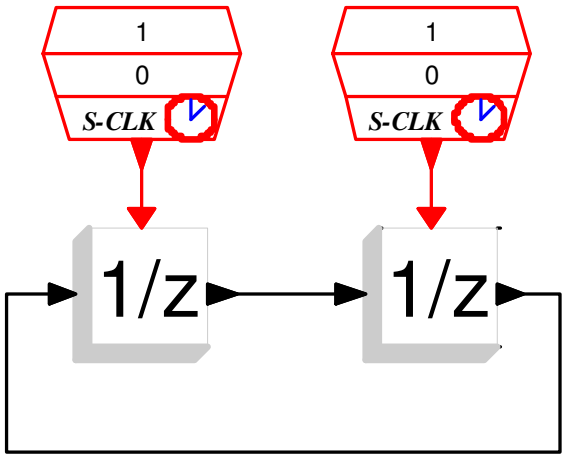
Modelica vs Scicos



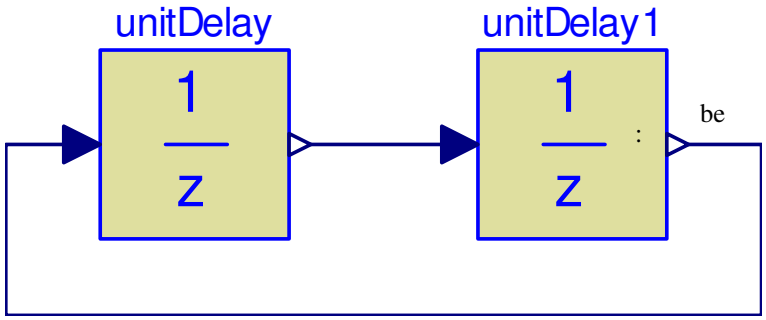


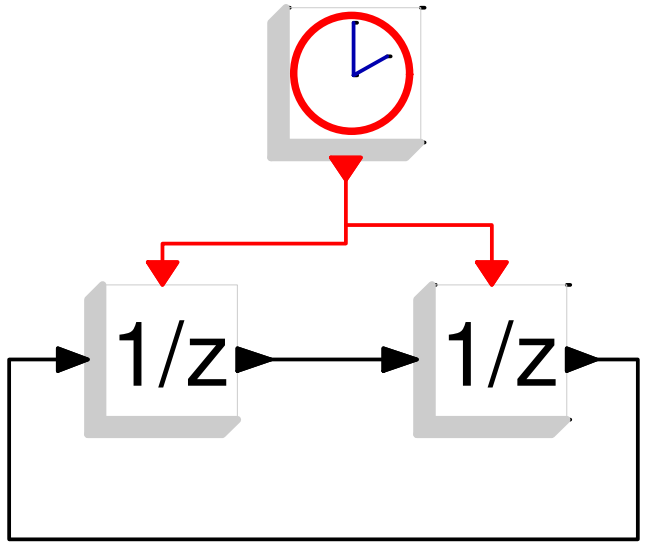
The two diagrams are not equivalent
Scicos diagram is not synchronous

The correct formulation is:



Other solution: explicit event signals

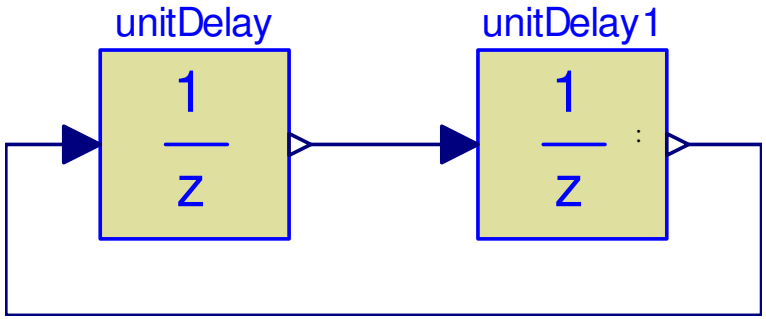


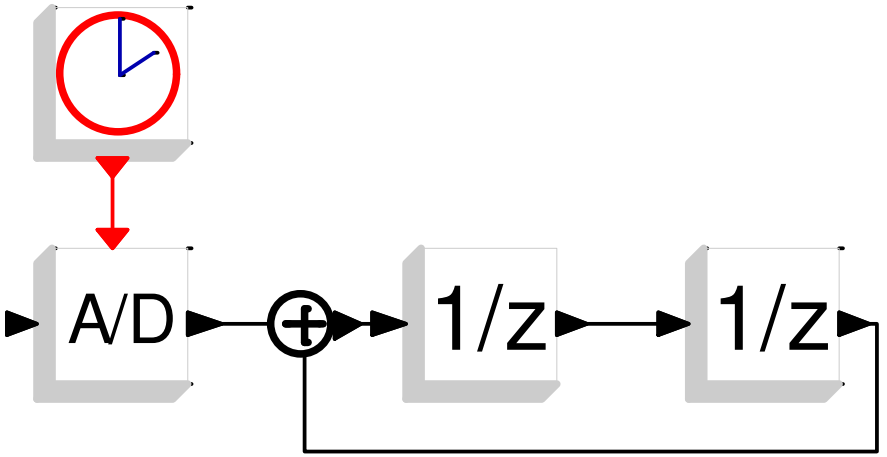


Other solution to synchronize blocks is to drive them explicitly with synchronized event signals

Events need not be periodic

Other solution: activation inheritance





Blocks, in the absence of activation, inherit their activations through regular inputs

Events need not be periodic

Activation inheritance

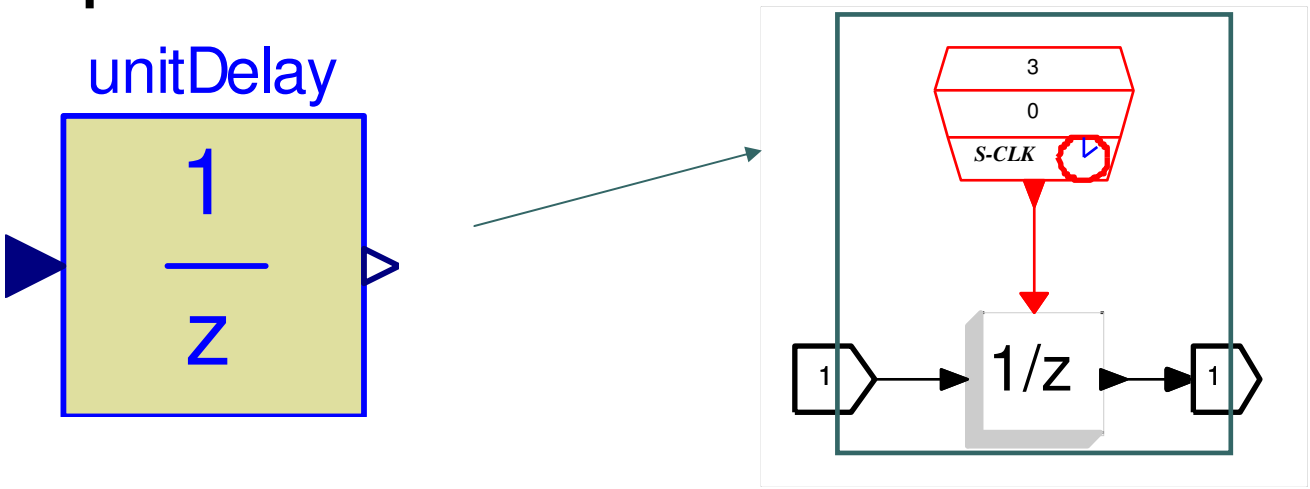
Simple and non-ambiguous rules

Provides a data-flow like behavior

Inheritance mechanism does not considerably modify the compiler: missing activation signals added at a pre-compilation phase

Can also be used in applications where the activations are not periodic

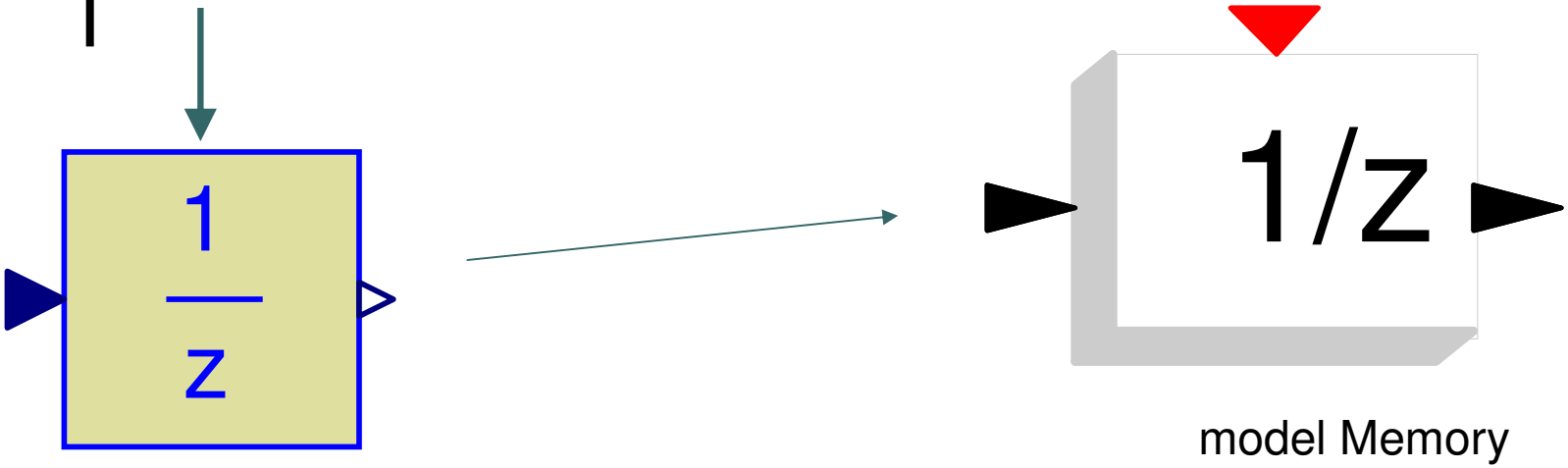
Modelica: method 1



Consider a masked Scicos block to include period information in the block

Compatible with current Modelica Discrete library if special interpretation used for the keyword sample

Modelica: method 2

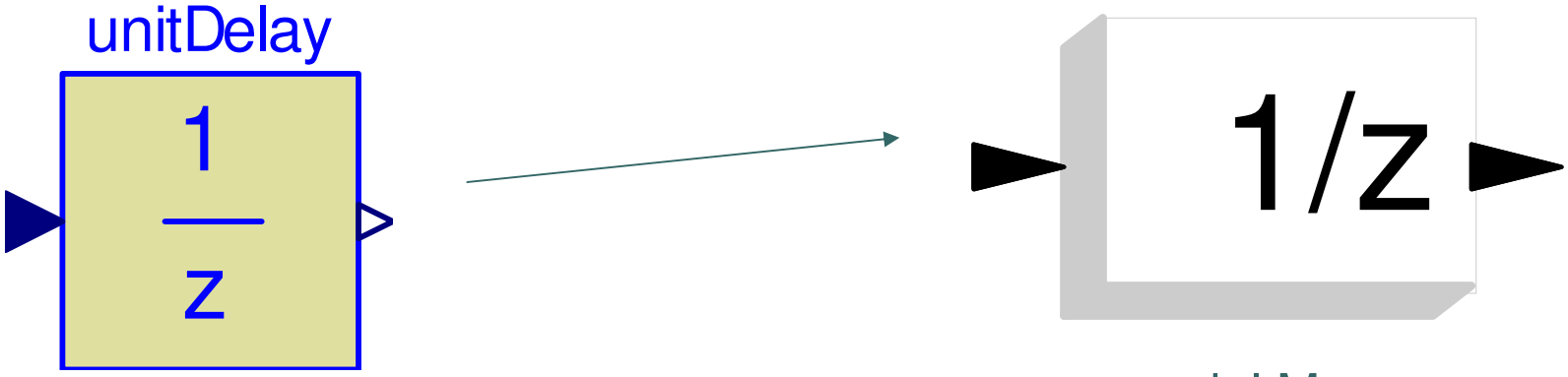


Introduce a new type “Event” in Modelica and include an Event input port for this block

Defining events as new types in Modelica has other advantages

```
model Memory
  input Event e1;
  output Real y;
  input Real u;
  discrete Real z;
  equation
  when e1 then
    z=u;
    y=pre(z);
  end when;
end Memory;
```

Modelica: method 3



Use activation inheritance mechanism

Allow “discrete” equations in “equation section where the activation in inherited

```
model Memory
   output Real y;
   input Real u;
   discrete Real z;
 equation
    z=u;
    y=pre(z);
end Memory;
```

Conclusion

Before considering to rewrite the discrete block library in Modelica, synchronization issue need to be clarified

All three mechanisms can be used but result in different libraries

Very relevant to current discussions about the controller specification and real-time code generation