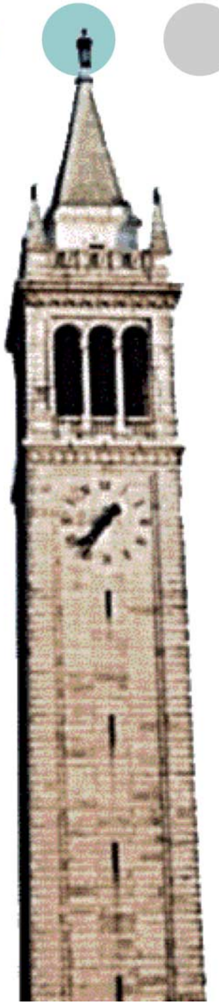


Synthesis of Reliable Distributed Real-Time Software

Edward A. Lee
Slobodan Matic
Jia Zou

UC Berkeley

Invited Keynote Talk

Workshop on Software Synthesis

ESWEEK 2010
Scottsdale, AZ, USA, October 29, 2010

Focus of this Talk: Distributed CPS Example – Printing Press

Bosch-Rexroth

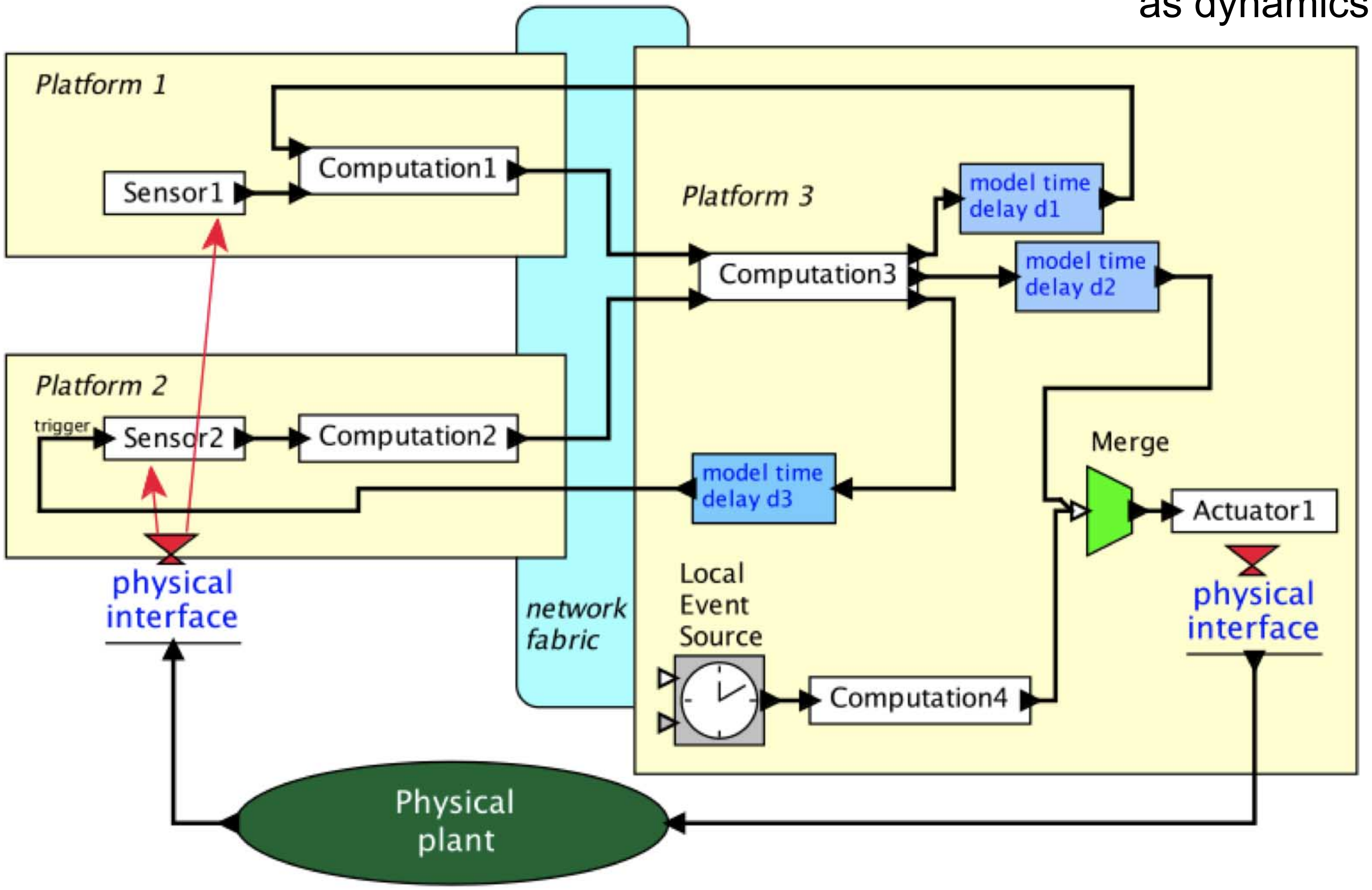
- *Distributed*
  - *100s of microcontrollers.*
  - *Ethernet with time synchronization (IEEE 1588).*
  - *Requires distributed fault handling.*

- *High-speed, high precision*
  - *Speed: 1 inch/ms.*
  - *Precision: 0.01 inch*

  --> Time accuracy: 10us.

Approaching the CPS Challenge

Physicalizing the cyber (PtC): to endow software and network components with abstractions and interfaces that represent their physical properties, such as dynamics in time.



Cyberizing the Physical (CtP): to endow physical subsystems with cyber-like abstractions and interfaces

For distributed cyber-physical systems,

Timing needs to be a part of the network *semantics*, not a side effect of the implementation.

Technologies needed:

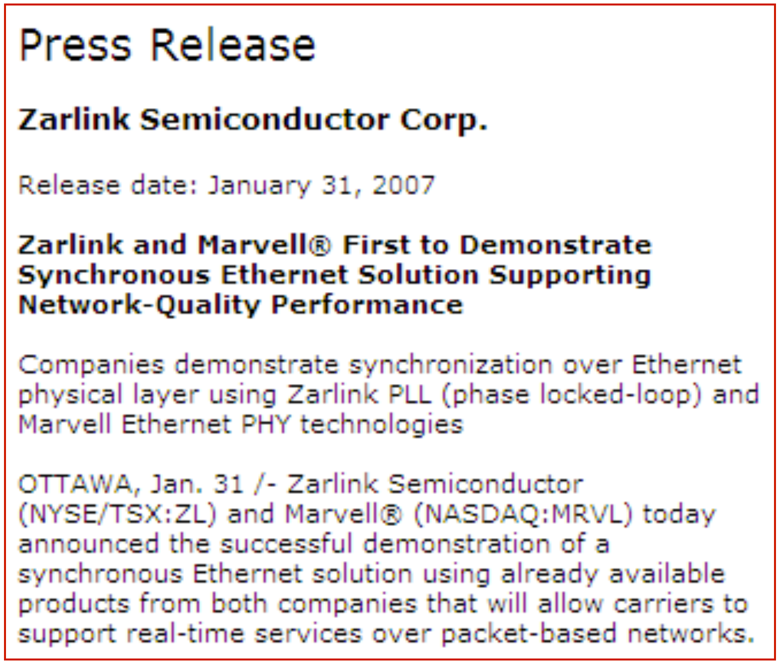
- Time synchronization
- Bounds on latency
- Time-aware fault isolation and recovery
- Time-aware robustness

Background - Domain-Specific Networks with Timed Semantics

- **WorldFIP** (Factory Instrumentation Protocol)
  - Created in France, 1980s, used in train systems
- **CAN**: Controller Area Network
  - Created by Bosch, 1980s/90s, ISO standard
- Various **ethernet** variants
  - PROFInet, EtherCAT, Powerlink, …
- **TTP/C**: Time-Triggered Protocol
  - Created around 1990, Univ. of Vienna, supported by TTTech
- **MOST**: Media Oriented Systems Transport
  - Created by a consortium of automotive & electronics companies
  - Under active development today
- **FlexRay**: Time triggered bus for automotive applications
  - Created by a consortium of automotive & electronics companies
  - Under active development today

Services in Time-Aware Networks

- Frequency locking
  - E.g., **synchronous ethernet**: ITU-T G.8261, May 2006
  - Enables integrating circuit-switched services on packet-switched networks
  - Can deliver performance independent of network loading.



Press Release

Zarlink Semiconductor Corp.

Release date: January 31, 2007

Zarlink and Marvell® First to Demonstrate Synchronous Ethernet Solution Supporting Network-Quality Performance

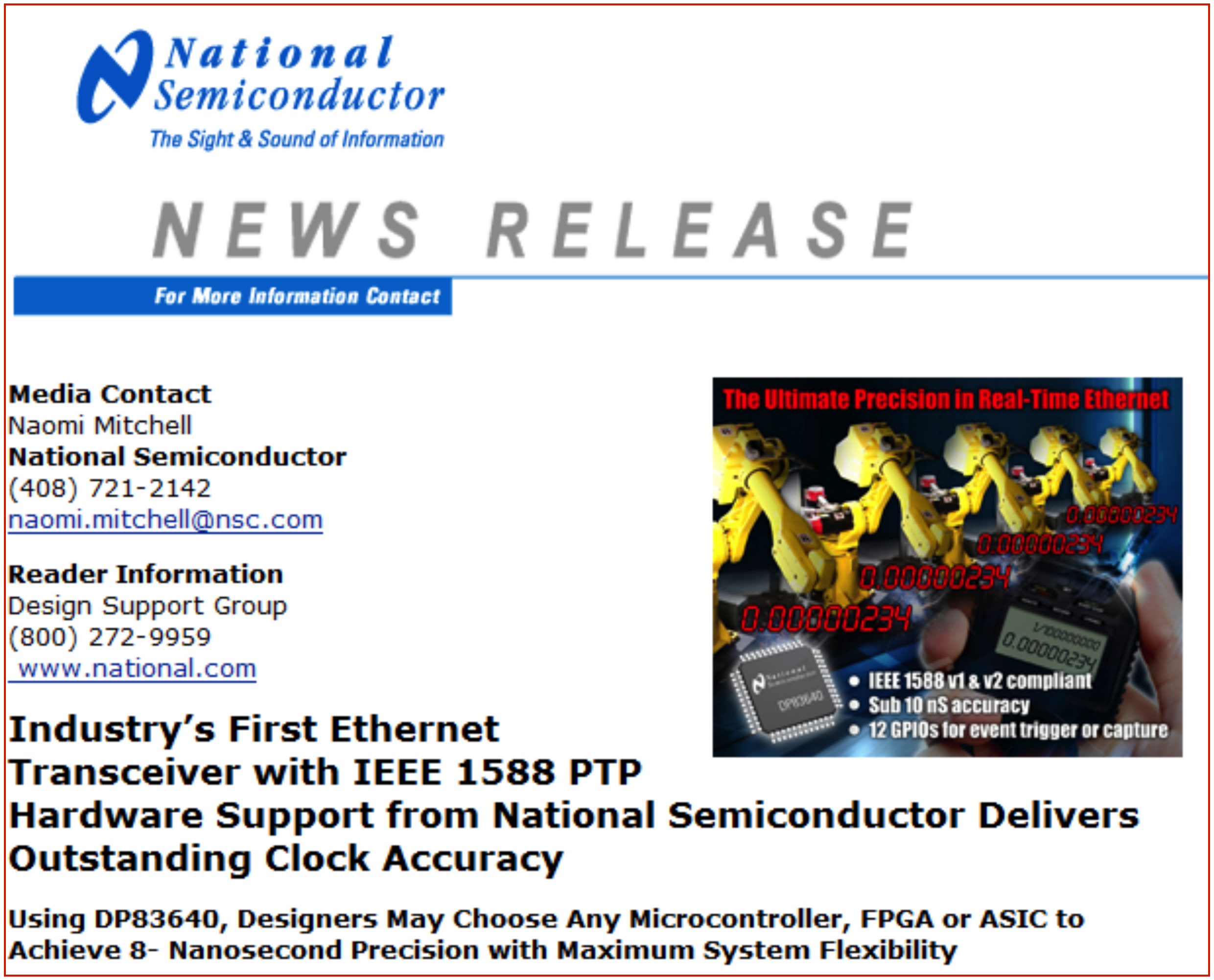
Companies demonstrate synchronization over Ethernet physical layer using Zarlink PLL (phase locked-loop) and Marvell Ethernet PHY technologies

OTTAWA, Jan. 31 /- Zarlink Semiconductor (NYSE/TSX:ZL) and Marvell® (NASDAQ:MRVL) today announced the successful demonstration of a synchronous Ethernet solution using already available products from both companies that will allow carriers to support real-time services over packet-based networks.

- Time synchronization
  - E.g., **IEEE 1588** standard set in 2002.
  - Synchronized time-of-day across a network.

Time Synchronization on Ethernet with TCP/IP: IEEE 1588 PTP

Press Release October 1, 2007



National Semiconductor
The Sight & Sound of Information

NEWS RELEASE

For More Information Contact

Media Contact
Naomi Mitchell
National Semiconductor
(408) 721-2142
naomi.mitchell@nsc.com

Reader Information
Design Support Group
(800) 272-9959
www.national.com

The Ultimate Precision in Real-Time Ethernet
0.00000234
0.00000234
0.00000234
0.00000234
1/100000000
0.00000234
DP83640
• IEEE 1588 v1 & v2 compliant
• Sub 10 nS accuracy
• 12 GPIOs for event trigger or capture

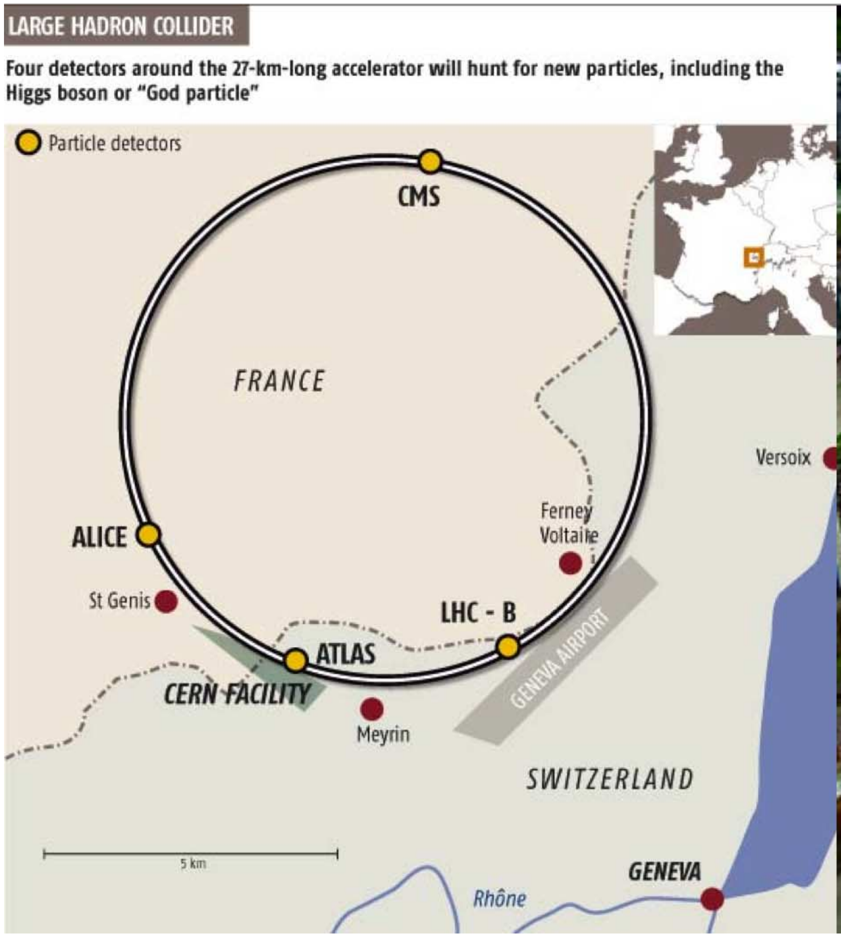
Industry's First Ethernet Transceiver with IEEE 1588 PTP Hardware Support from National Semiconductor Delivers Outstanding Clock Accuracy

Using DP83640, Designers May Choose Any Microcontroller, FPGA or ASIC to Achieve 8- Nanosecond Precision with Maximum System Flexibility

Clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

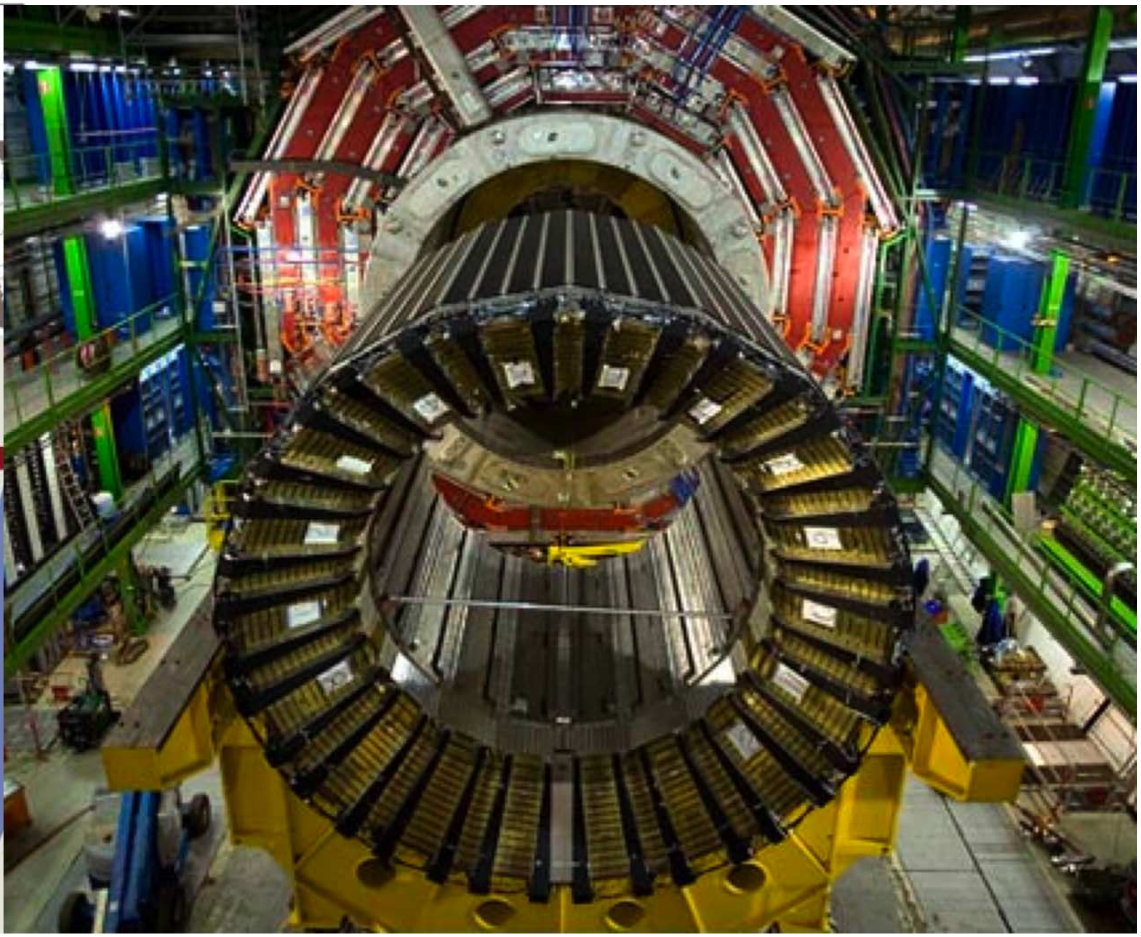
An Extreme Example:
The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous Ethernet.



LARGE HADRON COLLIDER

Four detectors around the 27-km-long accelerator will hunt for new particles, including the Higgs boson or "God particle"



The question we address:

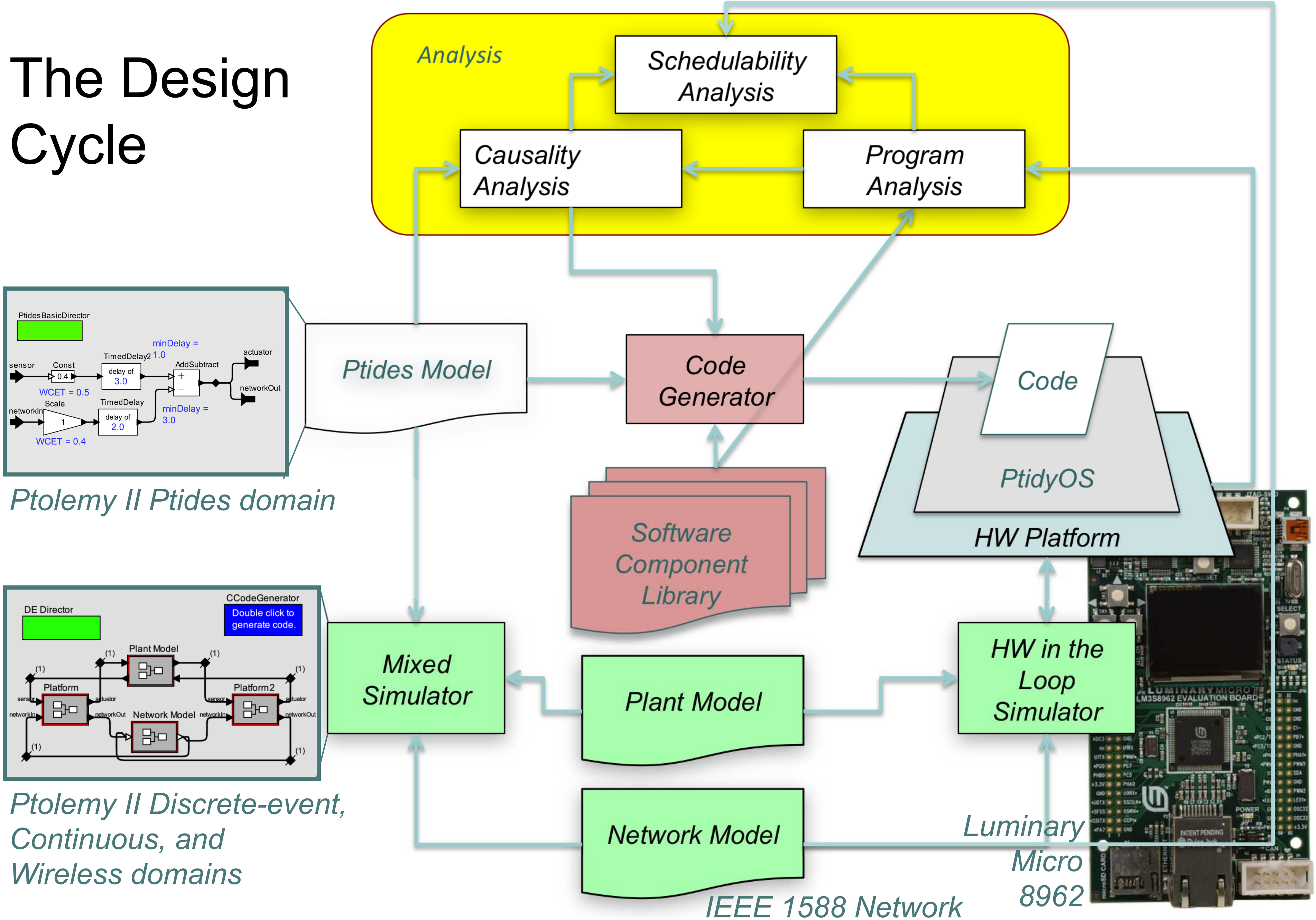
If you assume that computers on a network can agree on the current time of day within some bounded error,

how does this change how we develop distributed real-time software?

Our answer: It changes everything!

Our approach: Model-based design based on distributed discrete-event (DE) models with synthesis of embedded software.

The Design Cycle



Analysis

Schedulability Analysis

Causality Analysis

Program Analysis

Ptides Model

Code Generator

Code

PtidyOS

HW Platform

Software Component Library

Mixed Simulator

Plant Model

HW in the Loop Simulator

Network Model

Ptolemy II Ptides domain

Ptolemy II Discrete-event, Continuous, and Wireless domains

Luminary Micro 8962

IEEE 1588 Network time protocol

Lee, Matic, Zou, Berkeley 10

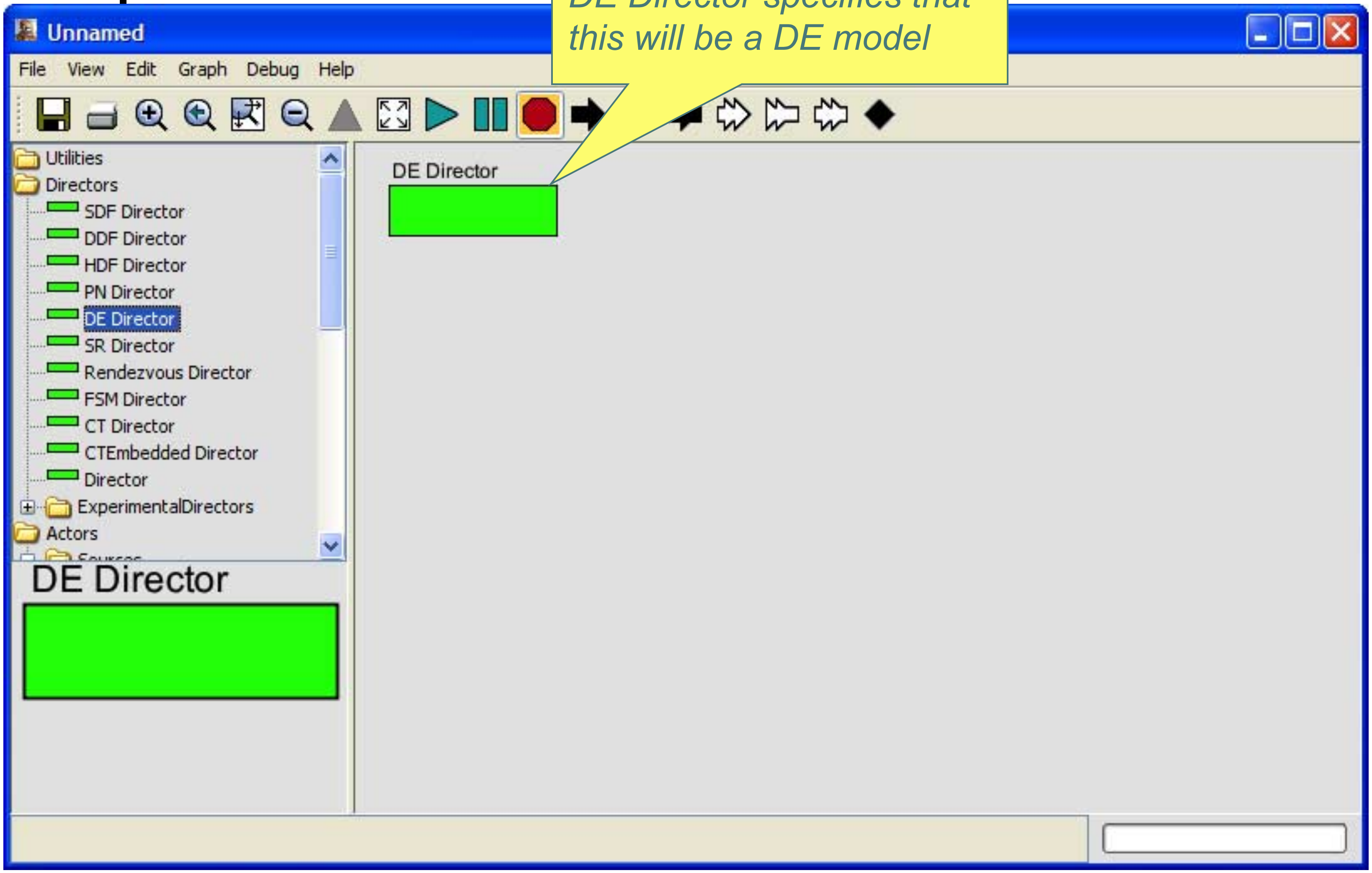
Our Approach is based on Discrete Events (DE)

- Concurrent actors
- Exchange time-stamped messages (“events”)

A correct execution is one where every actor reacts to input events in time-stamp order.

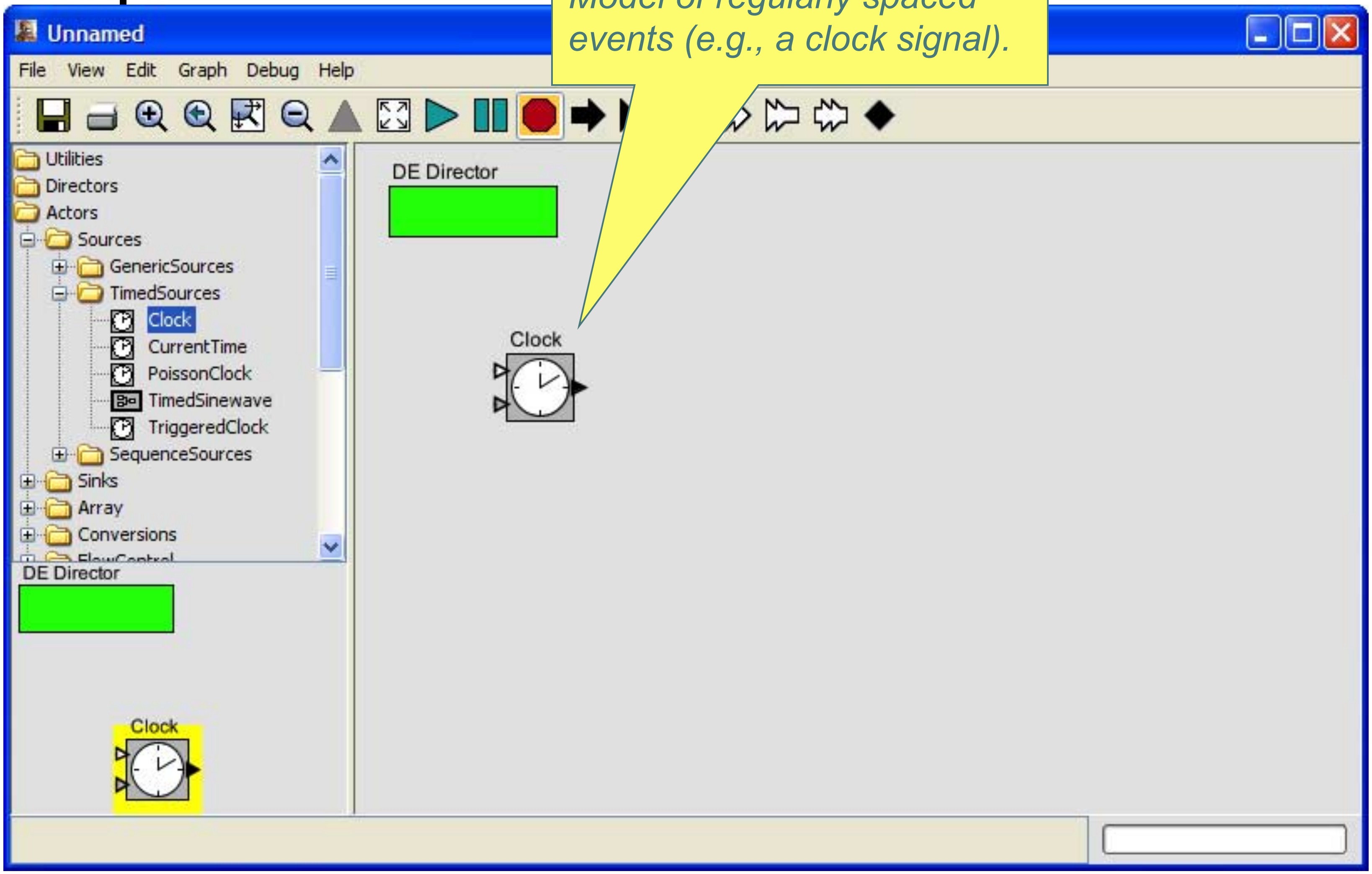
Time stamps are in “**model time**,” which typically bears no relationship to “**real time**” (wall-clock time). We use *superdense time* for the time stamps.

Building a DE Model (in Ptolemy II)



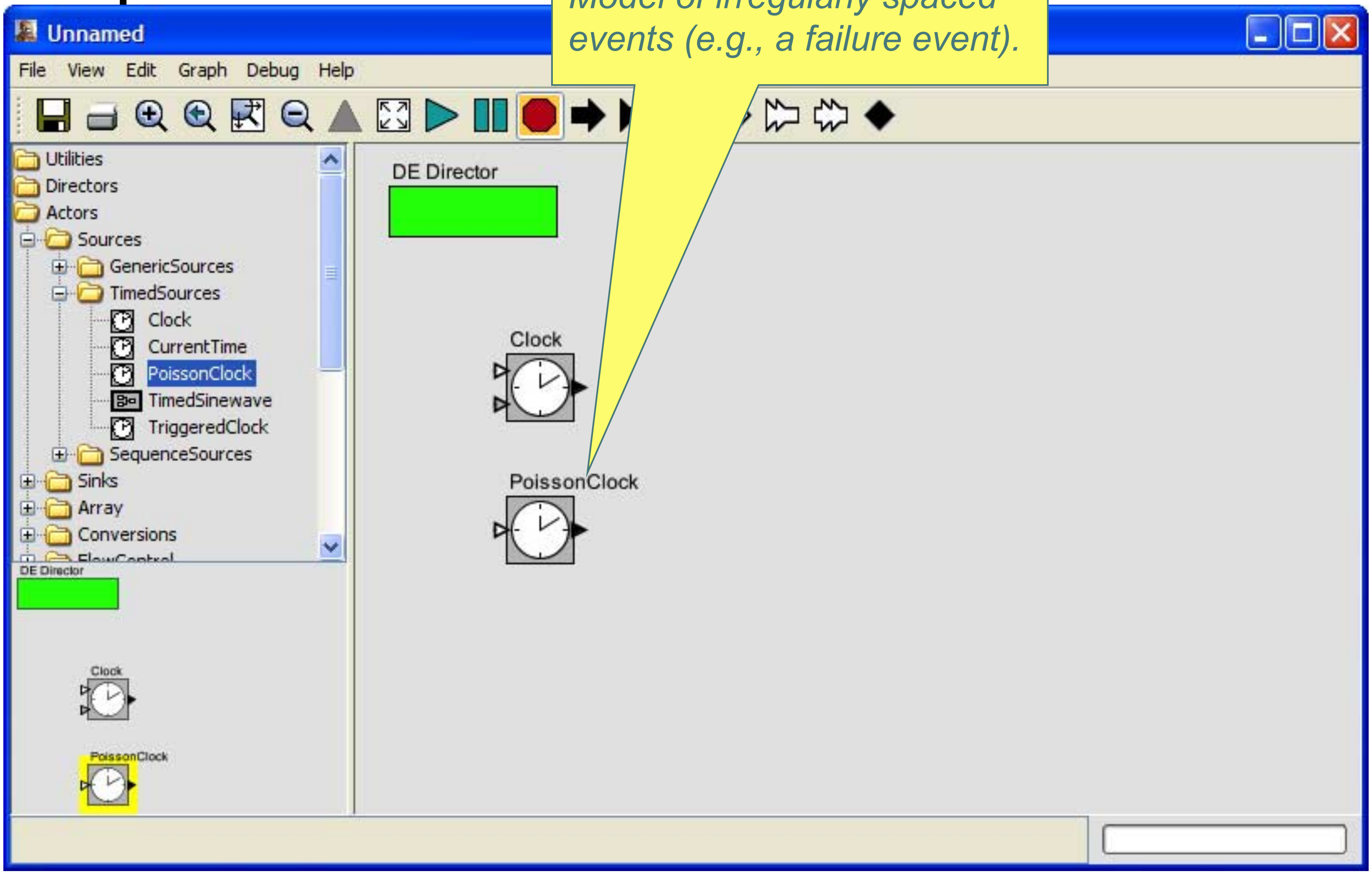
DE Director specifies that this will be a DE model

Building a DE Model (in Ptolemy II)



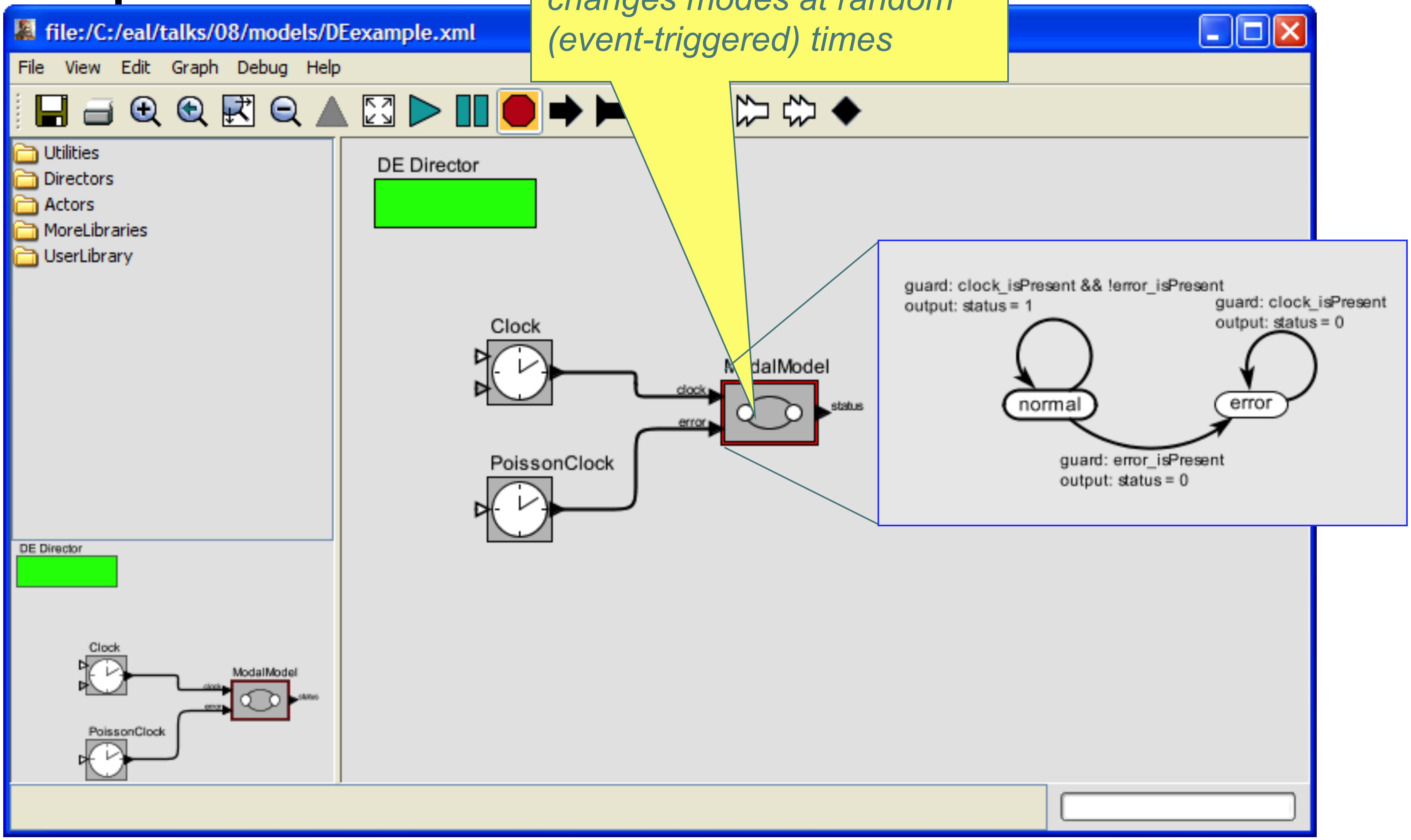
Model of regularly spaced events (e.g., a clock signal).

Building a DE Model (in Ptolemy II)



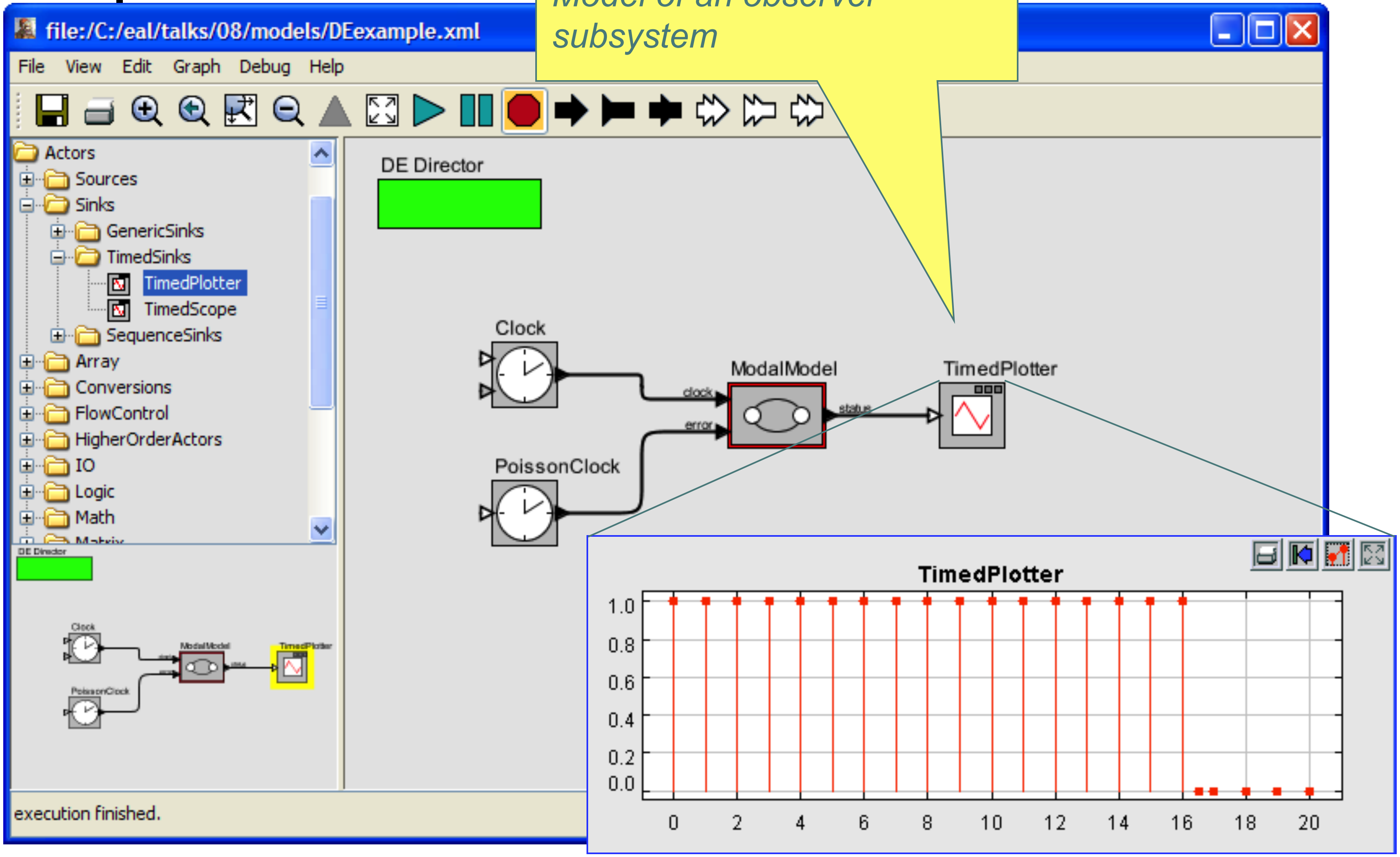
Model of irregularly spaced events (e.g., a failure event).

Building a DE Model (in Ptolemy II)



Model of a subsystem that changes modes at random (event-triggered) times

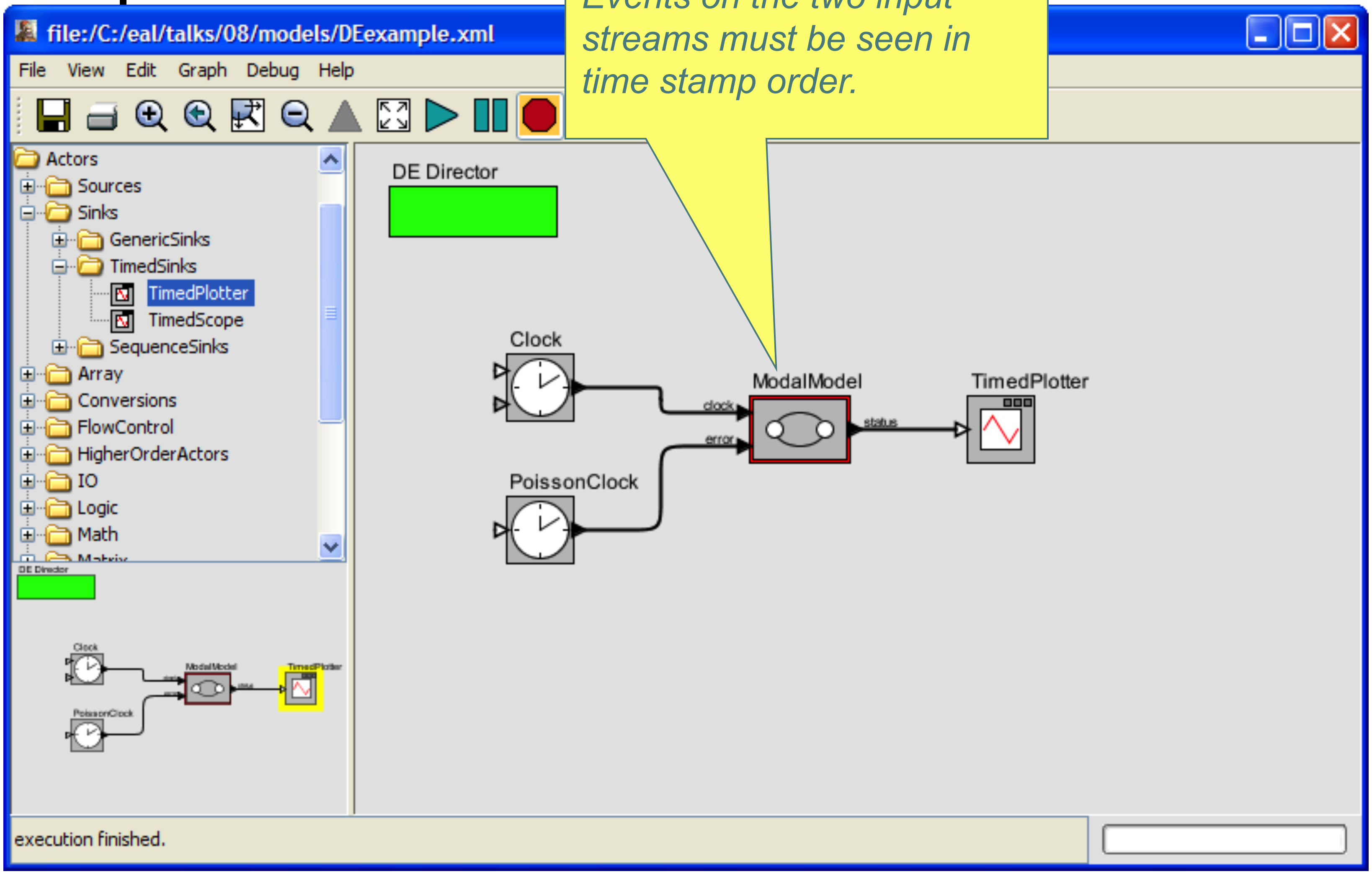
Building a DE Model (in Ptolemy II)



execution finished.

Model of an observer subsystem

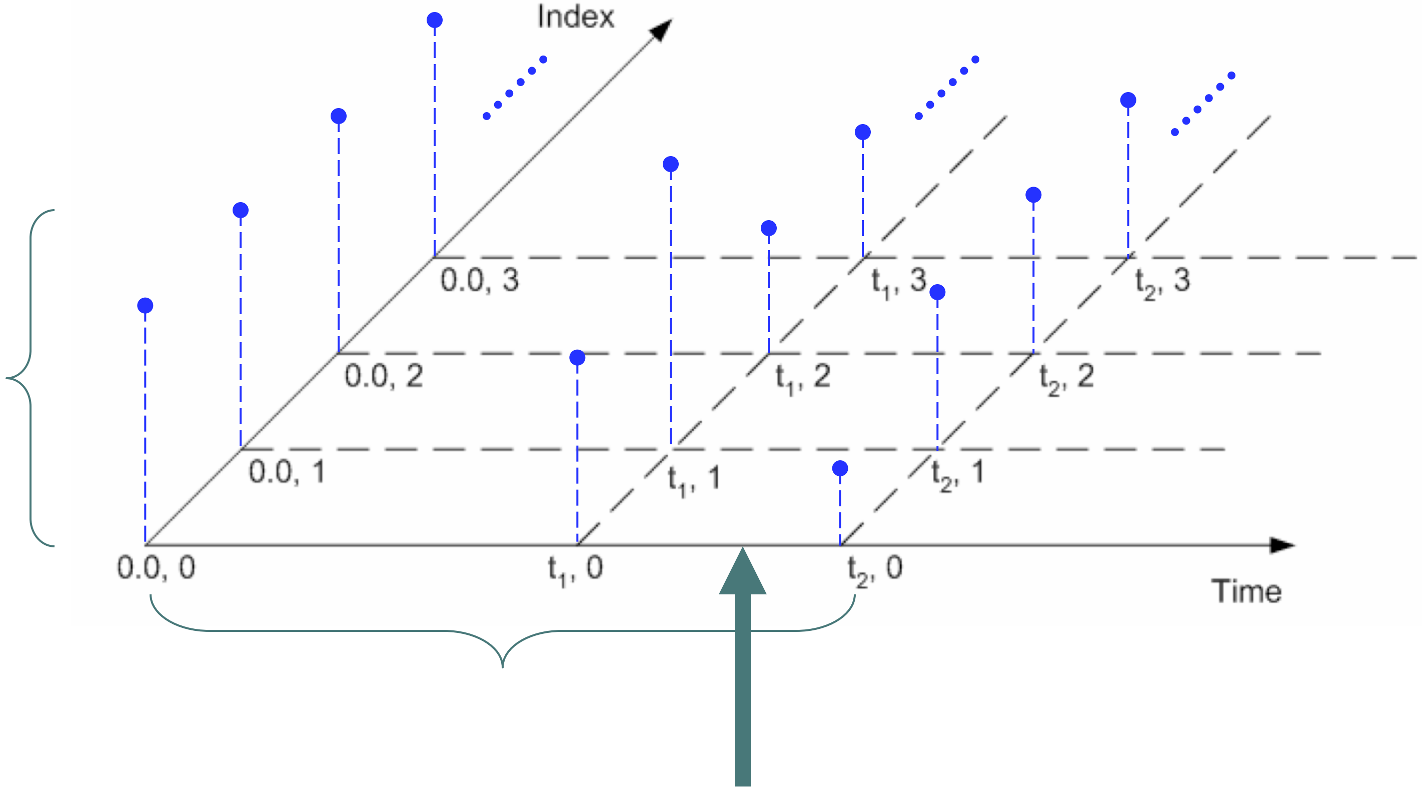
Building a DE Model (in Ptolemy II)



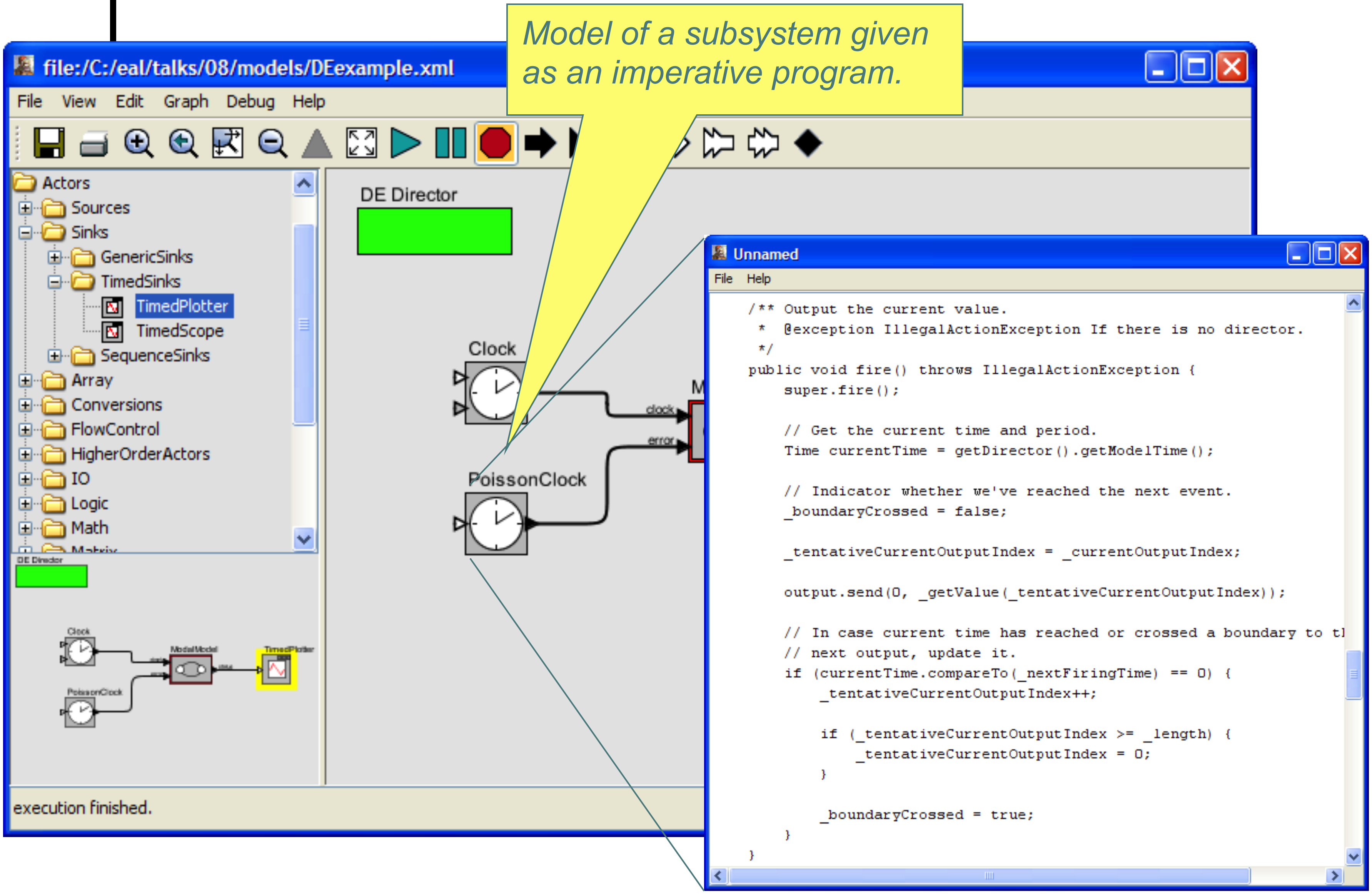
execution finished.

Events on the two input streams must be seen in time stamp order.

Aside:
Superdense Time Enables Better Conjunction of Computation and Physical Processes



This is a Component Technology

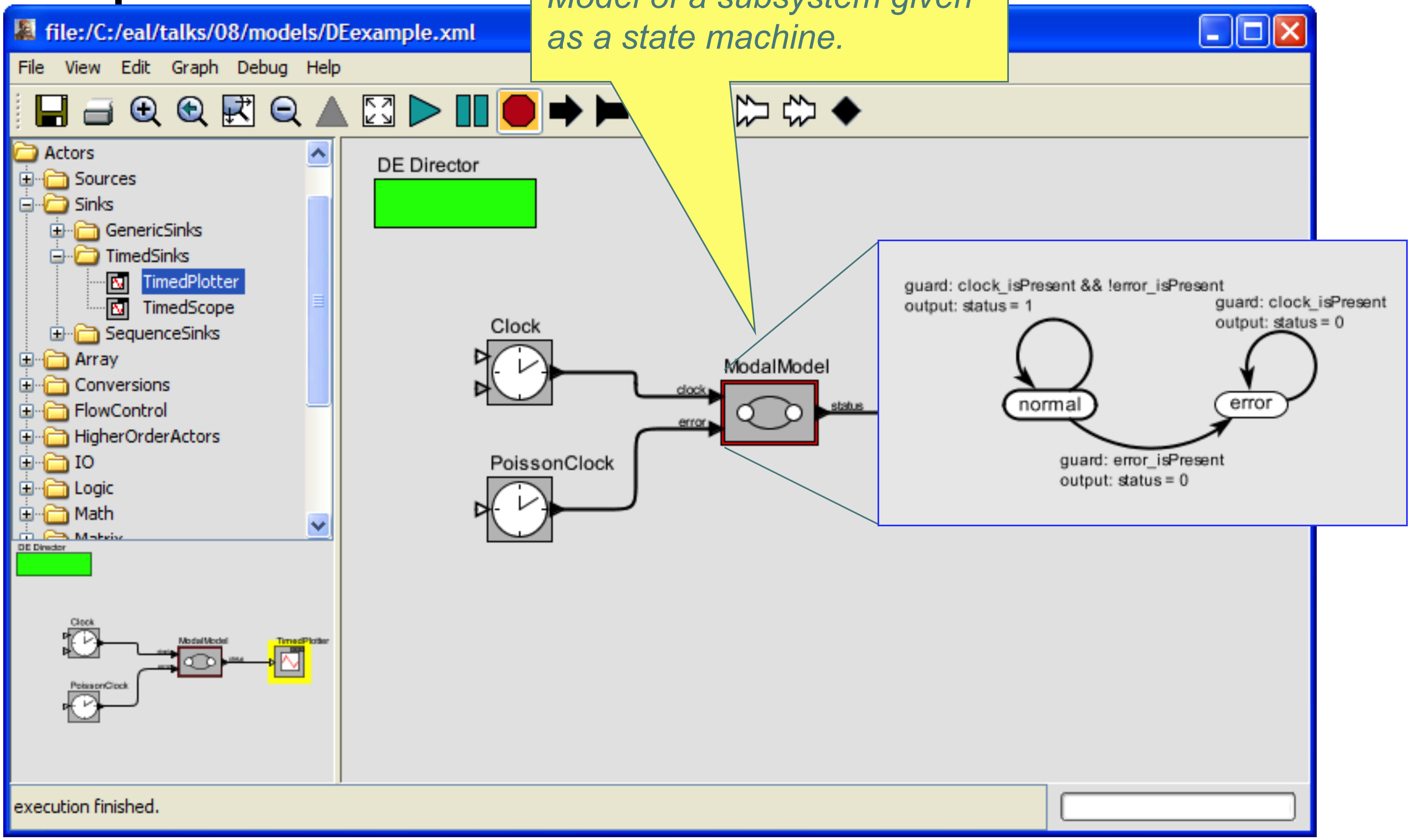


Model of a subsystem given as an imperative program.

```
/** Output the current value.
 *  @exception IllegalActionException If there is no director.
 */
public void fire() throws IllegalActionException {
    super.fire();

    // Get the current time and period.
    Time currentTime = getDirector().getModelTime();

    // Indicator whether we've reached the next event.
    _boundaryCrossed = false;

    _tentativeCurrentOutputIndex = _currentOutputIndex;

    output.send(0, _getValue(_tentativeCurrentOutputIndex));

    // In case current time has reached or crossed a boundary to t
    // next output, update it.
    if (currentTime.compareTo(_nextFiringTime) == 0) {
        _tentativeCurrentOutputIndex++;

        if (_tentativeCurrentOutputIndex >= _length) {
            _tentativeCurrentOutputIndex = 0;
        }

        _boundaryCrossed = true;
    }
}
```

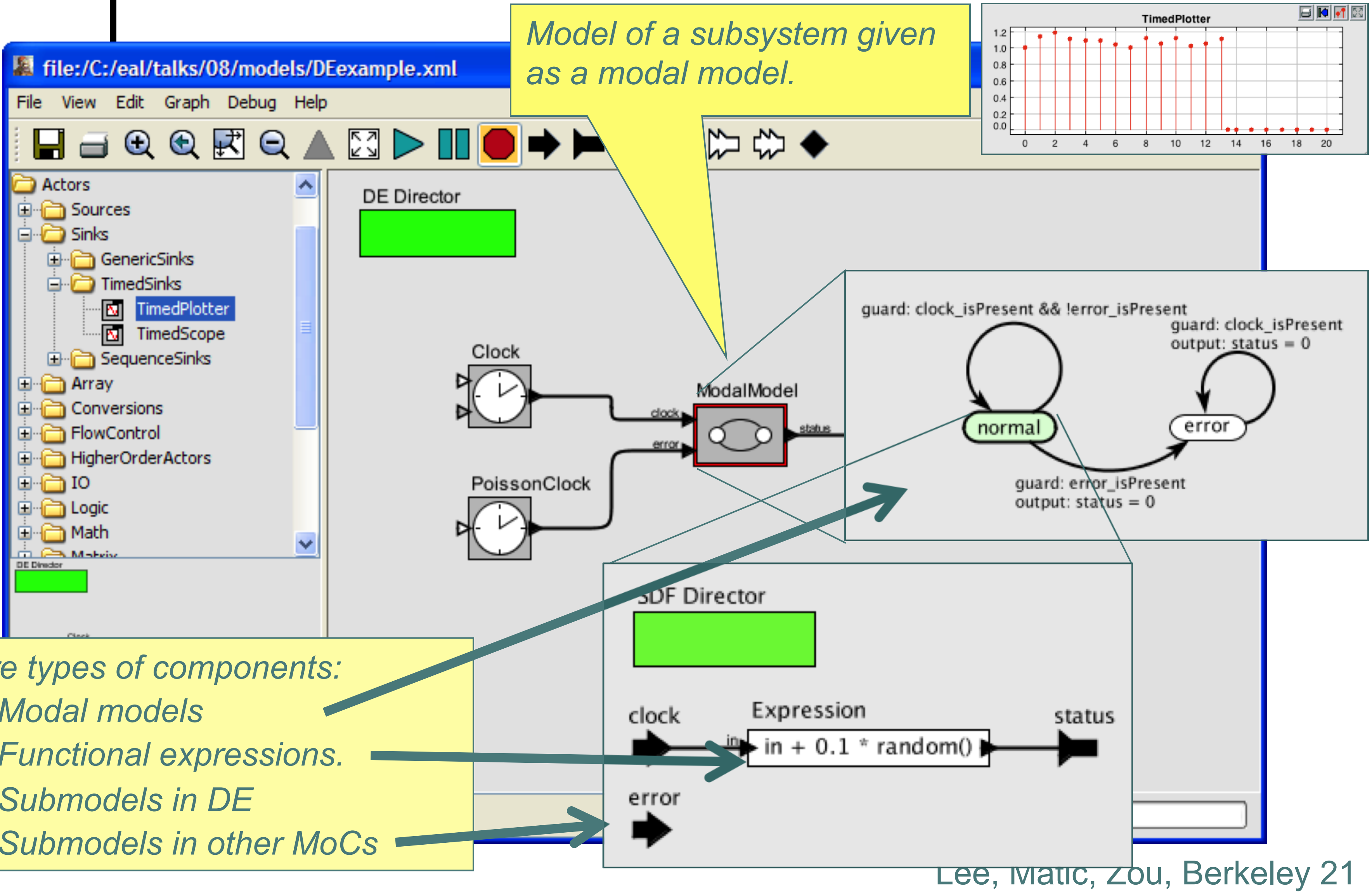
This is a Component Technology

Model of a subsystem given as a state machine.



execution finished.

This is a Component Technology



Model of a subsystem given as a modal model.

More types of components:

- *Modal models*
- *Functional expressions.*
- *Submodels in DE*
- *Submodels in other MoCs*

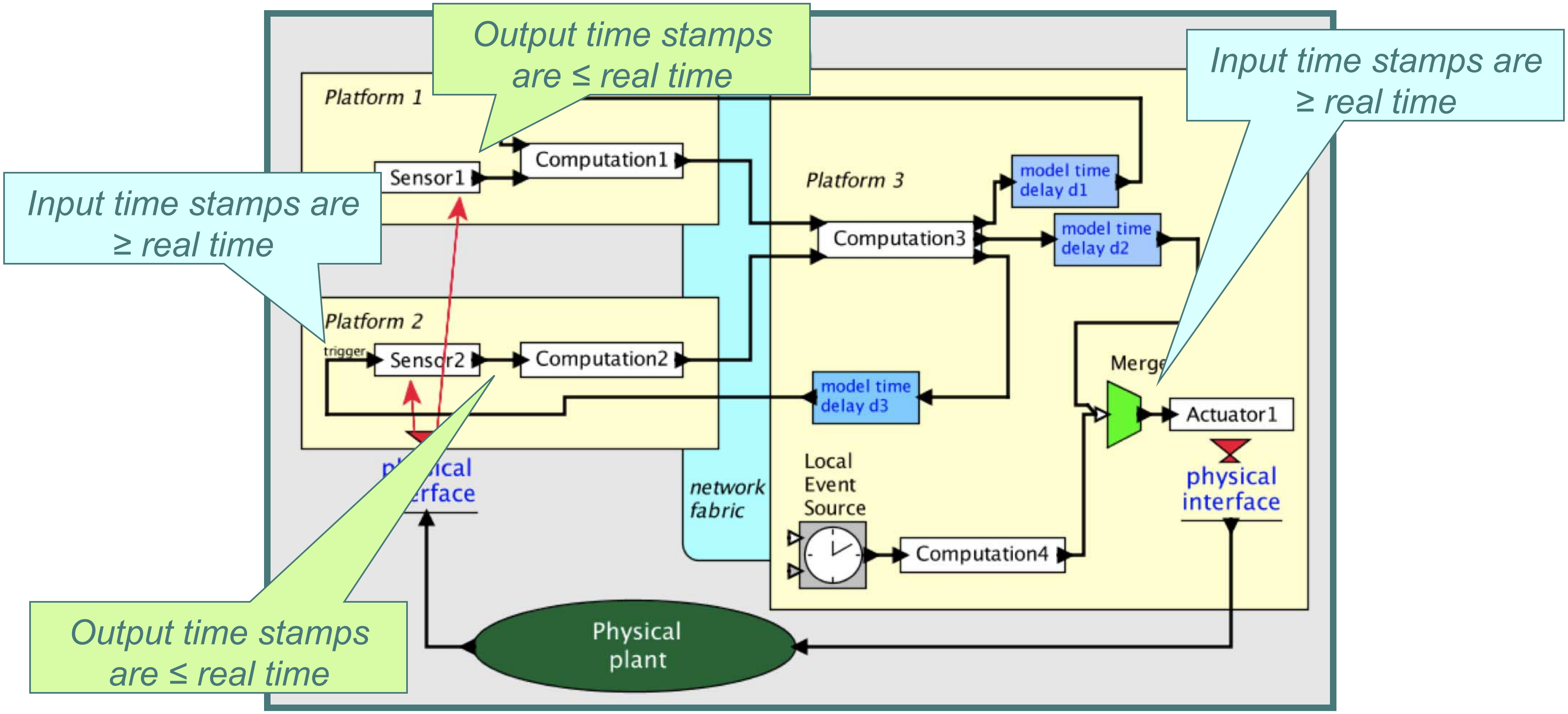
Using DE Semantics in Distributed Real-Time Systems

- DE is usually a simulation technology.
- Distributing DE is traditionally done for acceleration.
- Hardware design languages (e.g. VHDL) use DE where time stamps are literally interpreted as real time, or abstractly as ticks of a physical clock.

- We are using DE for distributed real-time software, binding time stamps to real time only where necessary.
- *PTIDES*: Programming Temporally Integrated Distributed Embedded Systems

PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Distributed execution under discrete-event semantics, with "model time" and "real time" bound at sensors and actuators.



Output time stamps are ≤ real time

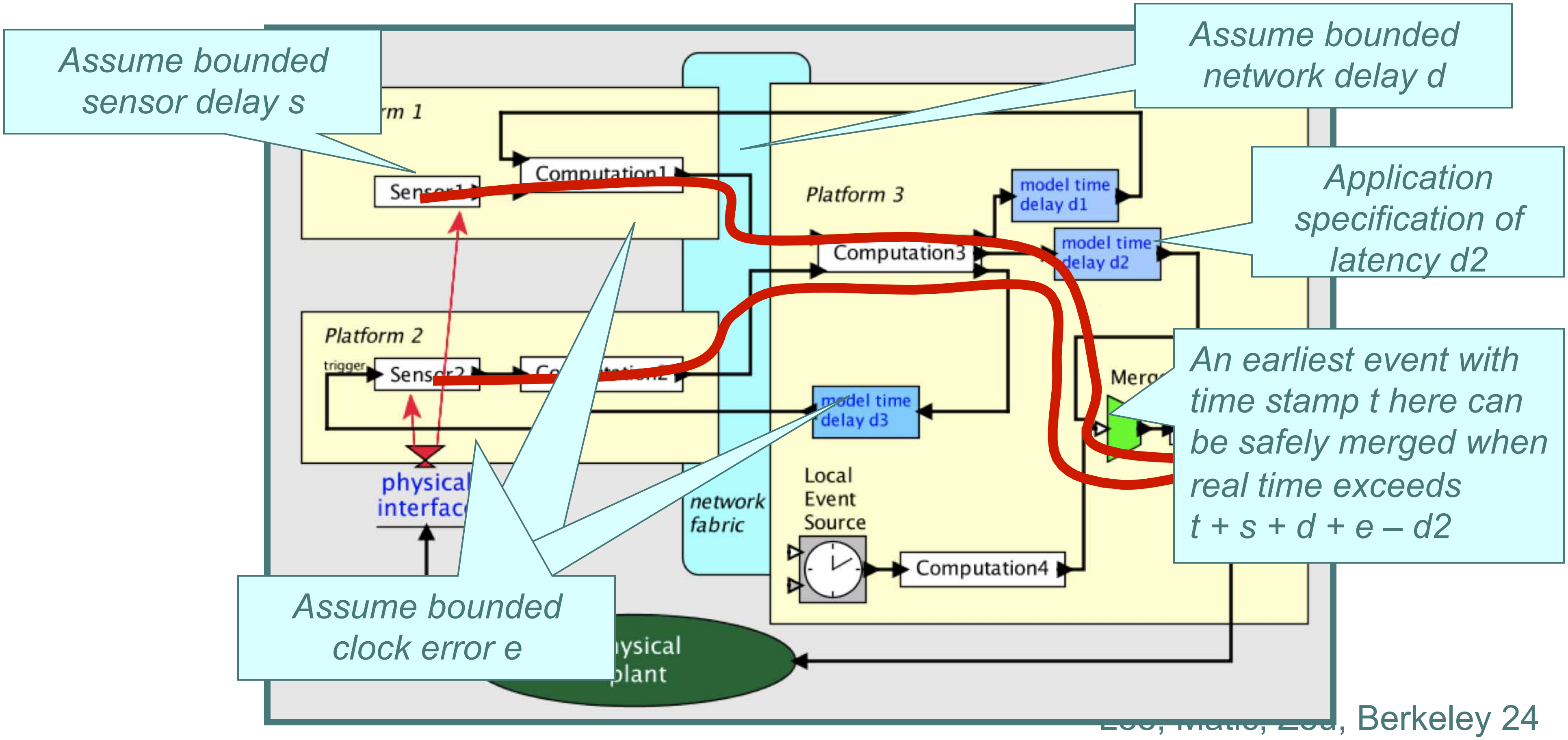
Input time stamps are ≥ real time

Input time stamps are ≥ real time

Output time stamps are ≤ real time

PTIDES: Programming Temporally Integrated Distributed Embedded Systems

PTIDES uses static causality analysis to determine when events can be safely processed (preserving DE semantics).



Assume bounded sensor delay s

Assume bounded network delay d

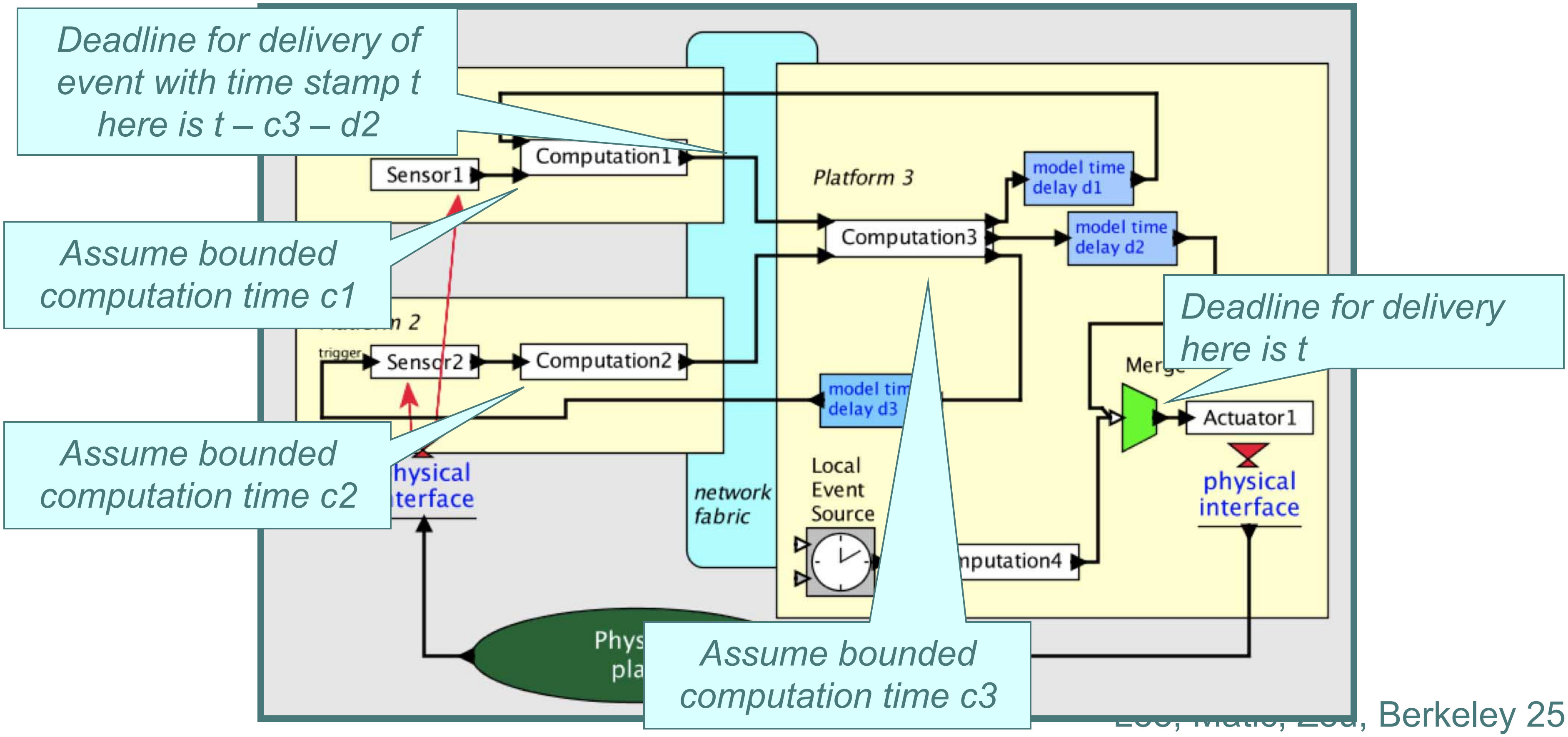
Application specification of latency d2

An earliest event with time stamp t here can be safely merged when real time exceeds t + s + d + e – d2

Assume bounded clock error e

PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Schedulability analysis incorporates computation times to determine whether we can guarantee that deadlines are met.



Deadline for delivery of event with time stamp t here is t – c3 – d2

Assume bounded computation time c1

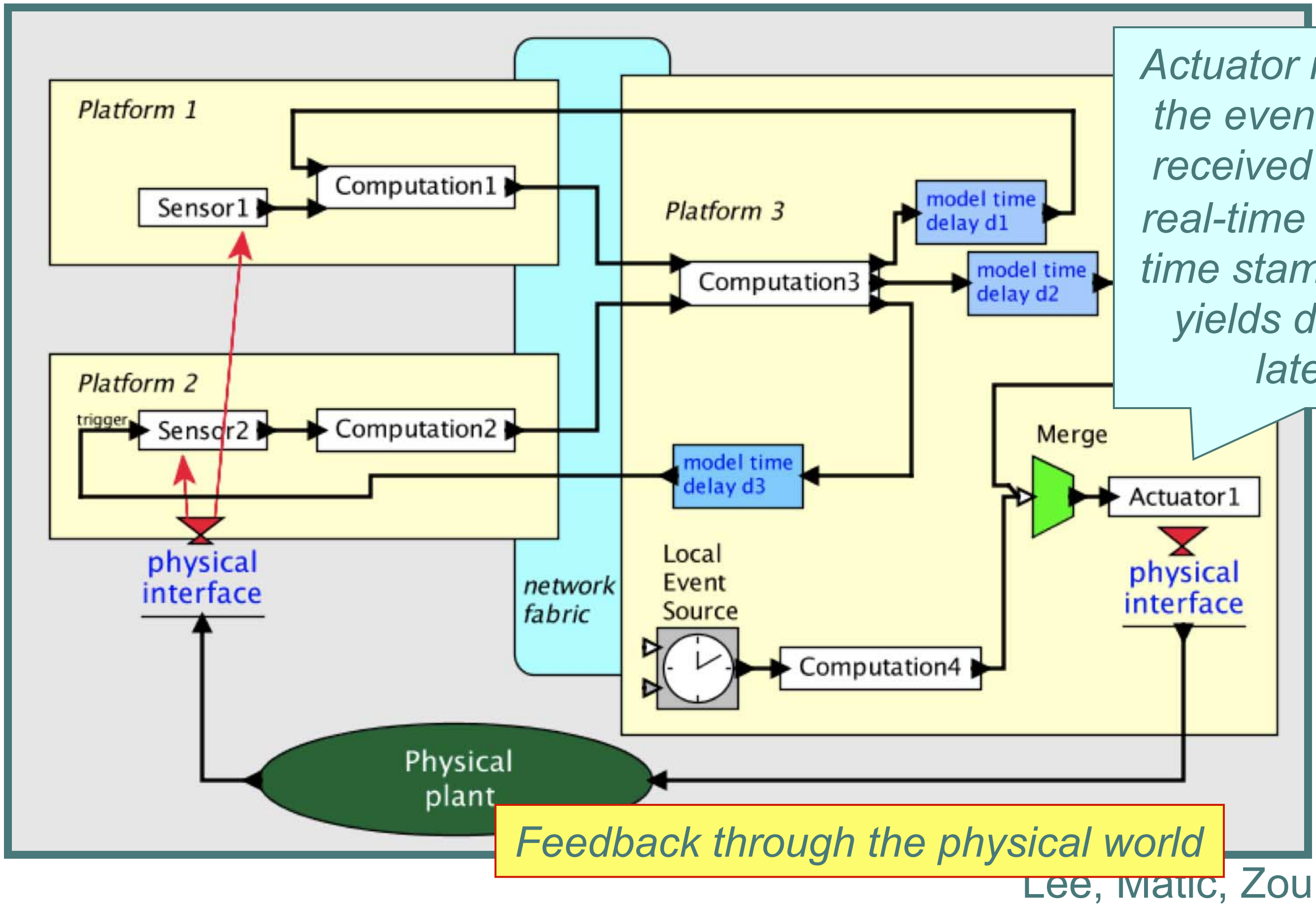
Assume bounded computation time c2

Deadline for delivery here is t

Assume bounded computation time c3

PTIDES: Programming Temporally Integrated Distributed Embedded Systems

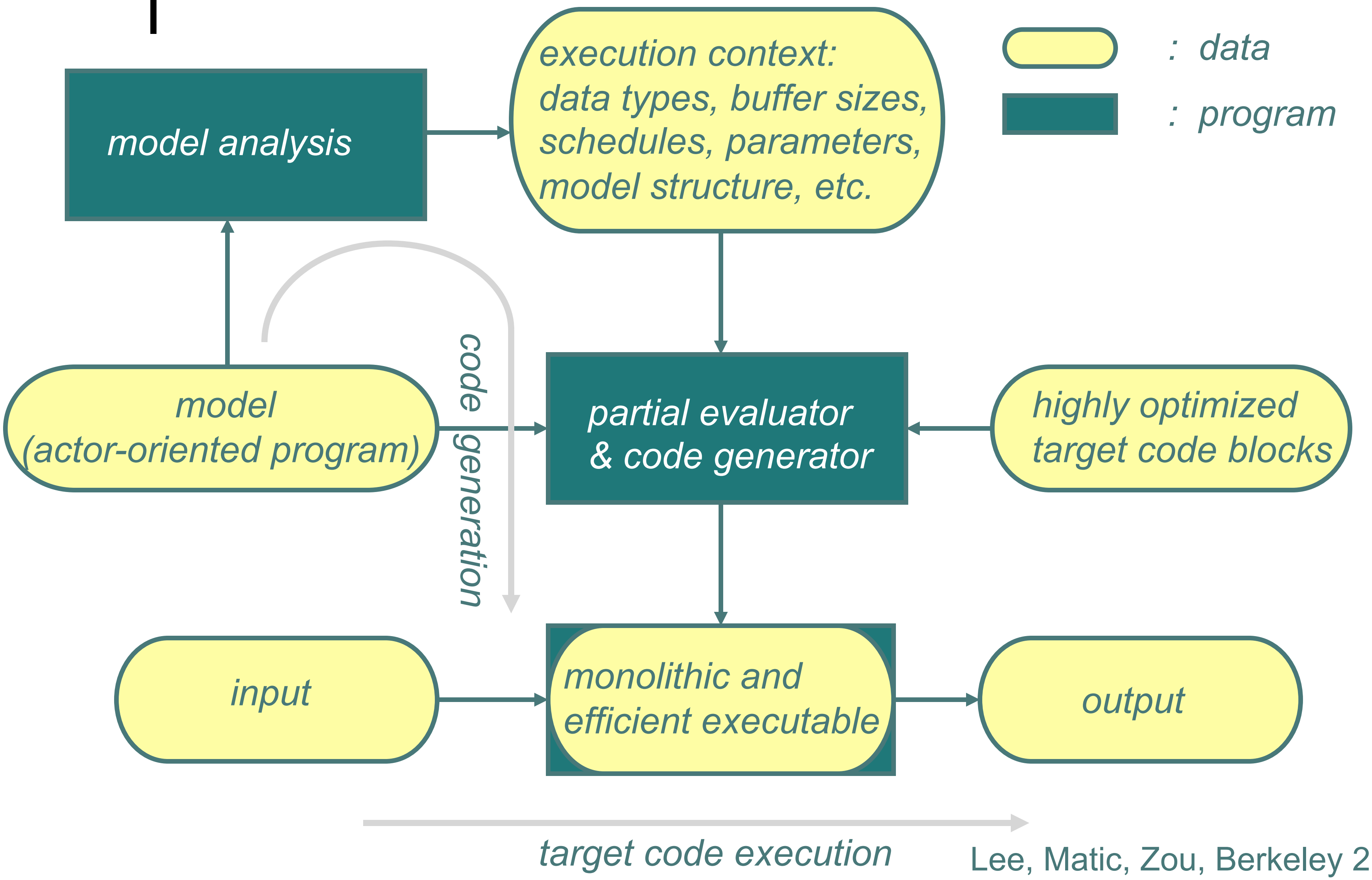
… and being explicit about time delays means that we can analyze control system dynamics…



Actuator may process the event at the time received or wait until real-time matches the time stamp. The latter yields determinate latencies.

Feedback through the physical world

Code Generation Approach: Run-time Kernel + Partial Evaluation + Generator Libraries



PtidyOS
The Run-Time Kernel

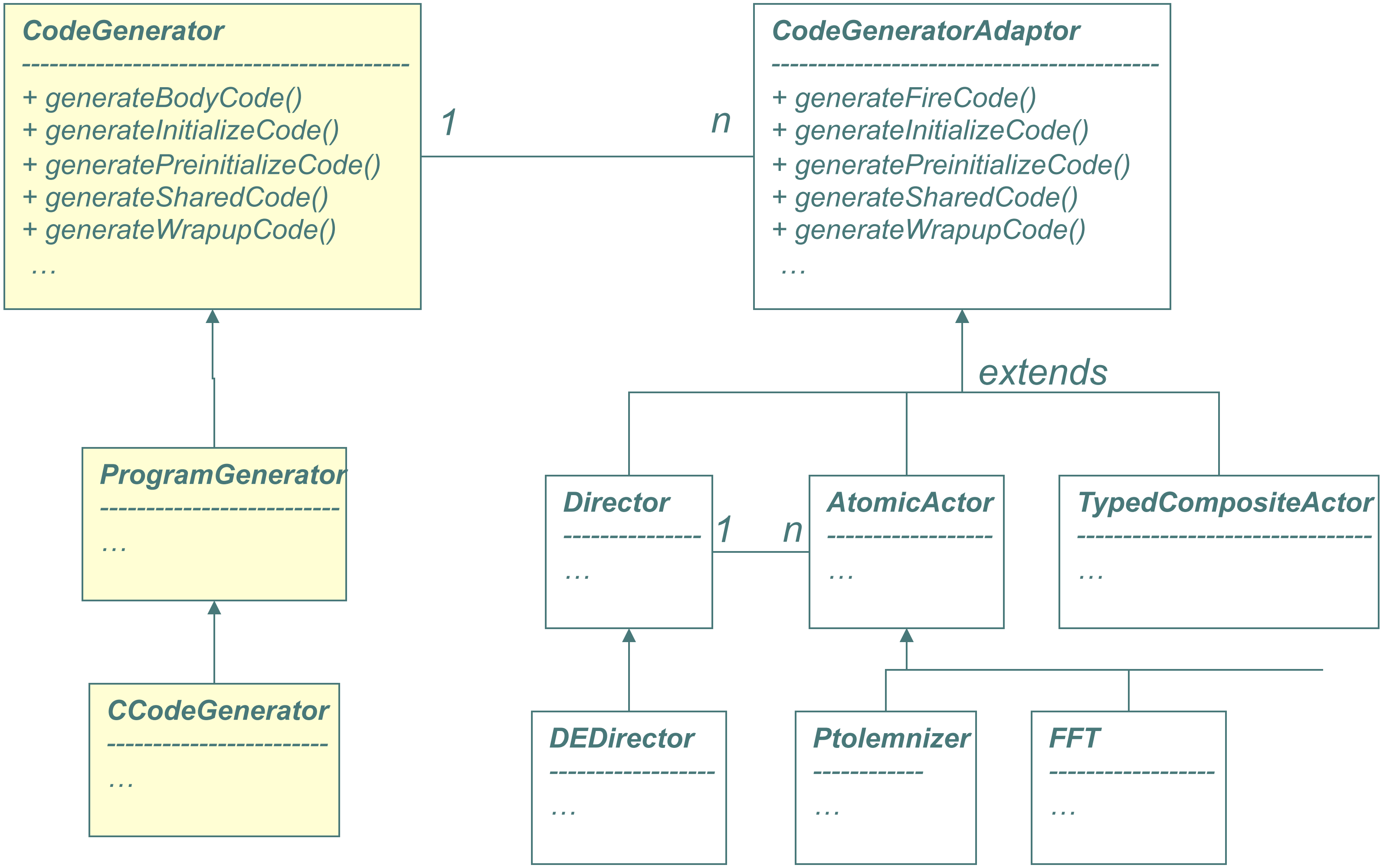
PtidyOS is a C library that gets linked with application code. Services:

- Synchronized time service (IEEE 1588)
- Sorted event queue (EQ)
- Scheduler dispatching event from EQ (safe-to-process analysis + EDF)
- Single-stack operation (preemption is strictly nested, caused by interrupts)
- Device driver services (time stamping of events, delayed actuation, etc.)

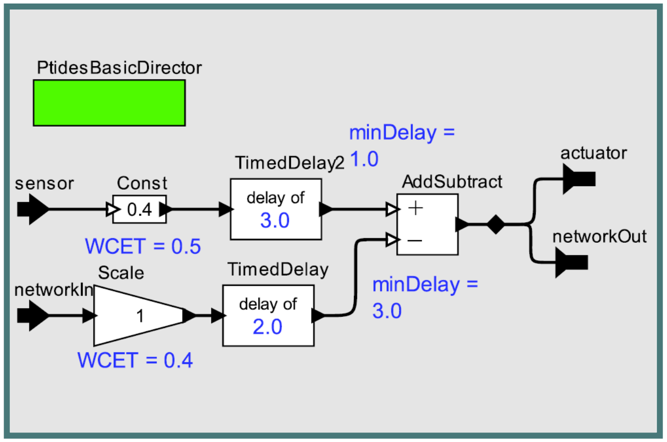
Partial Evaluation

- Type inference in Ptolemy II reduces polymorphic components to type-specific components.
- Dependencies among parameters reveal which can be statically evaluated, becoming constants in the generated code.
- Small primitive operations can be inlined rather than dispatched from the event queue.

Application code given by code generators called Adapters



Adapters are discovered

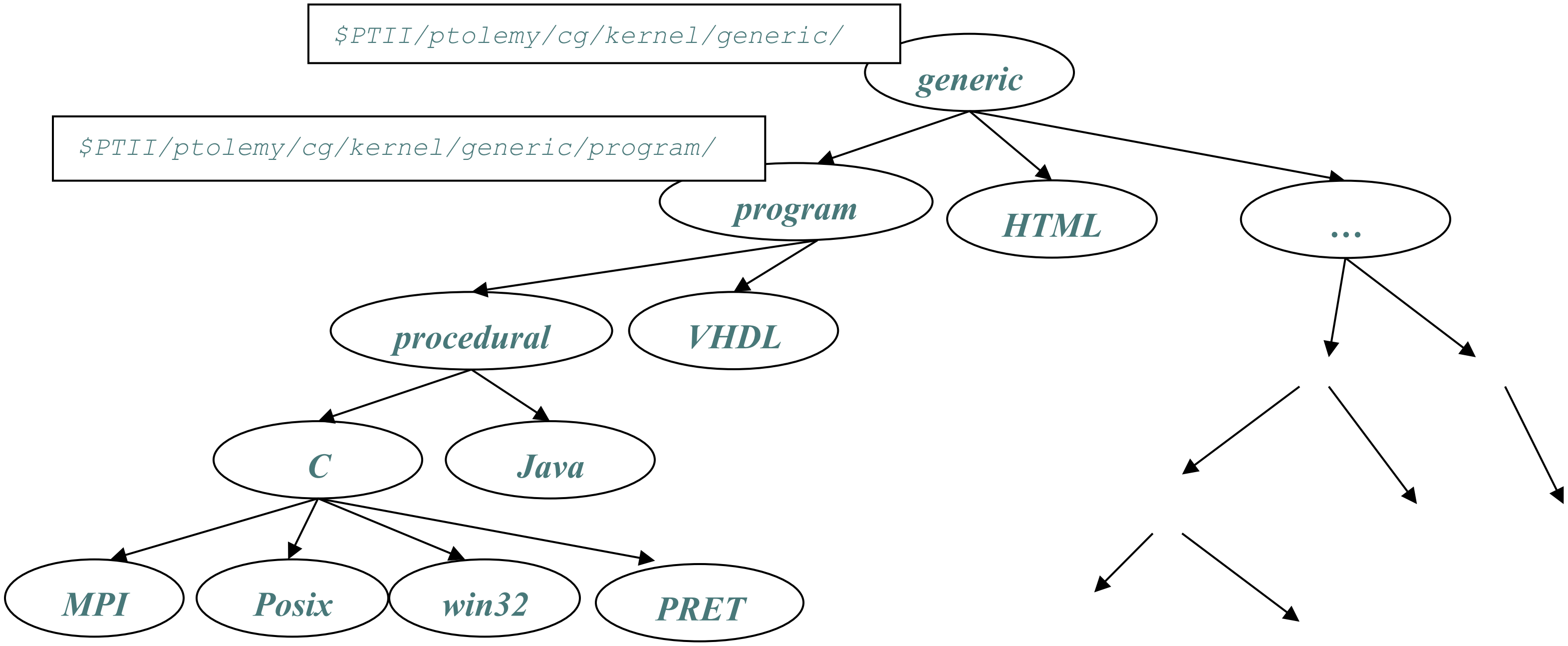


Ptides Model

Target

- Search for target-specific adapter
- If none found, search for language-specific adapter
- If none found, search for generic adapter (e.g. to generate documentation)
- Do this first for Directors, then for Actors

Adapter Library Hierarchy


$PTII/ptolemy/cg/kernel/generic/
generic
$PTII/ptolemy/cg/kernel/generic/program/
program
HTML
…
procedural
VHDL
C
Java
MPI
Posix
win32
PRET

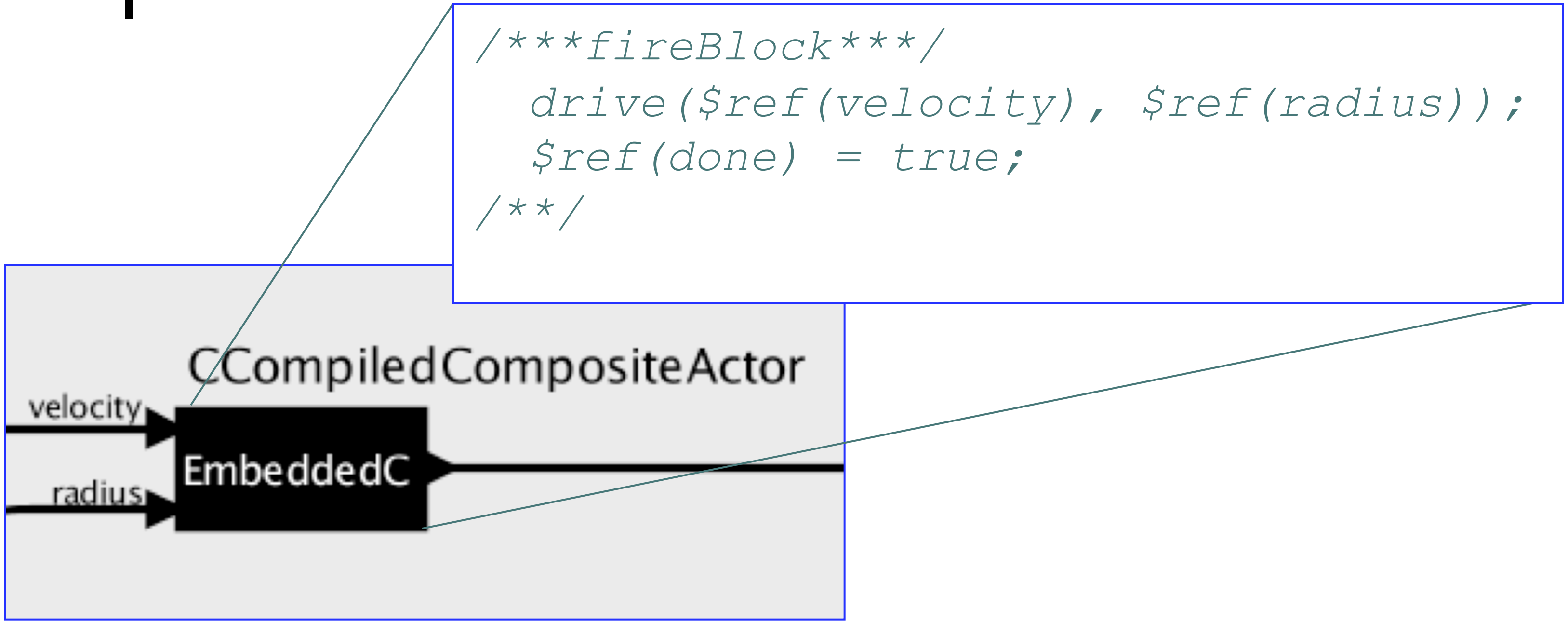
Within each library, adapters provide either code generators or template code to be customized by a generic code generator.

Sections of the Generated Content:

```
           Include Files
------------------------------------------
      Variable Declarations
------------------------------------------
     Procedure Declarations
------------------------------------------
          Initialize Code
------------------------------------------
             Body Code
------------------------------------------
            Wrapup Code
```

Adapters for directors and actors provide each of these sections either as a template or as a code generator.

Example of a Template-Based Adapter

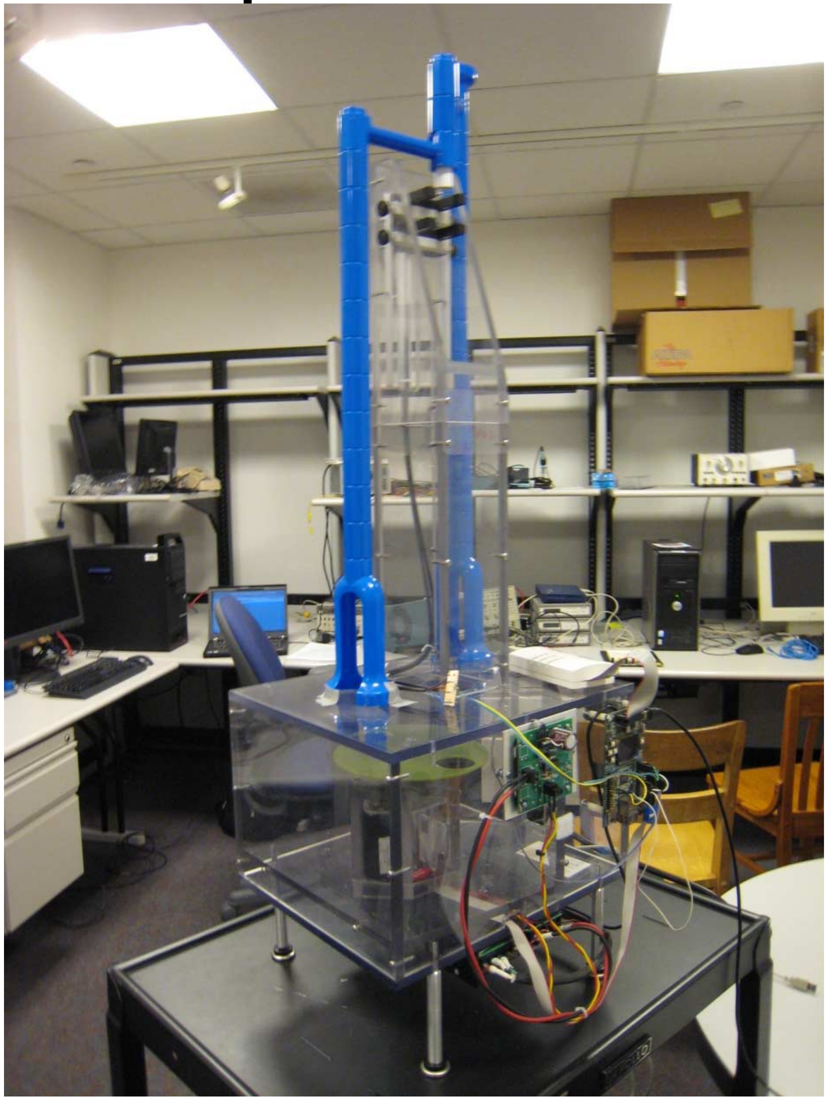


```
/***fireBlock***/
  drive($ref(velocity), $ref(radius));
  $ref(done) = true;
/**/
```

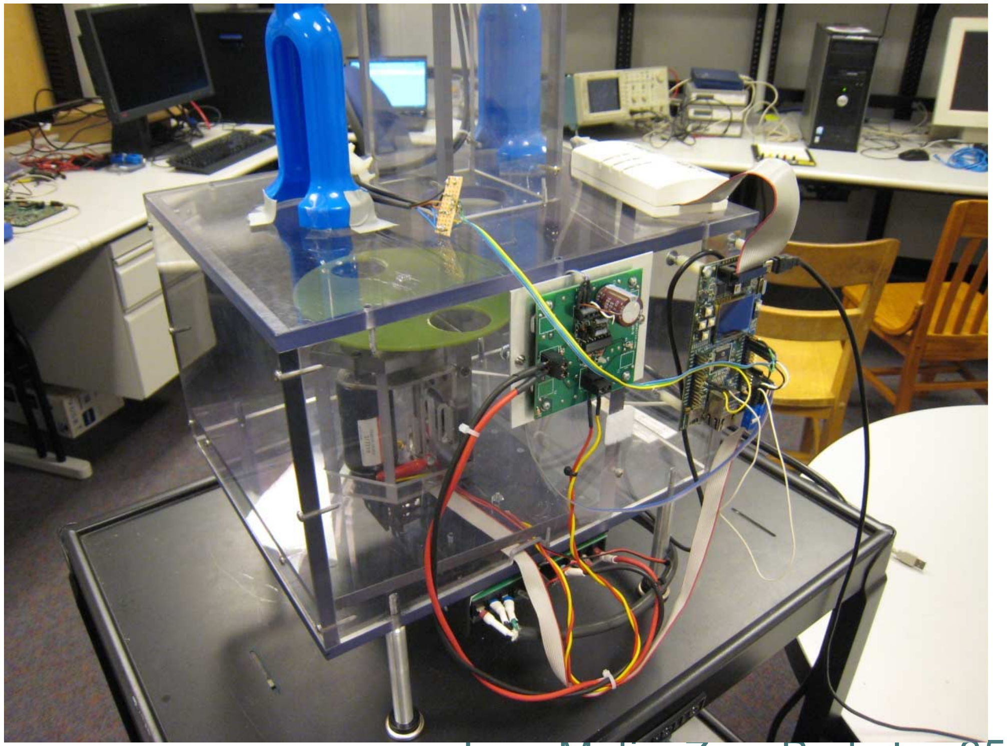
Templates allow actor functionality to be designed in low-level, target-specific code. This facilitates using PTIDES as *component architecture* rather than a *programming language.*

This device was designed by Jeff Jensen, now at National Instruments.

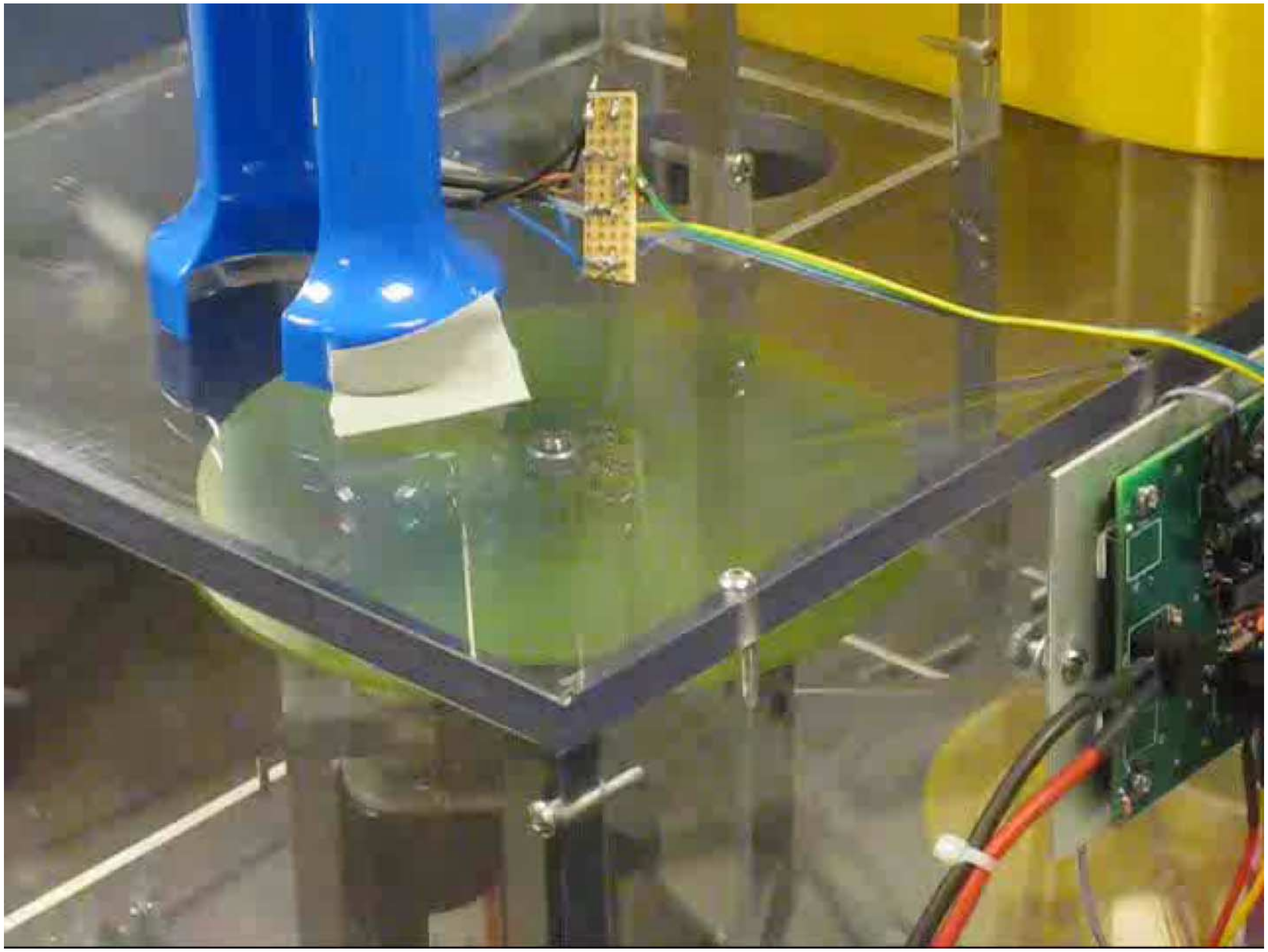
First Test Case

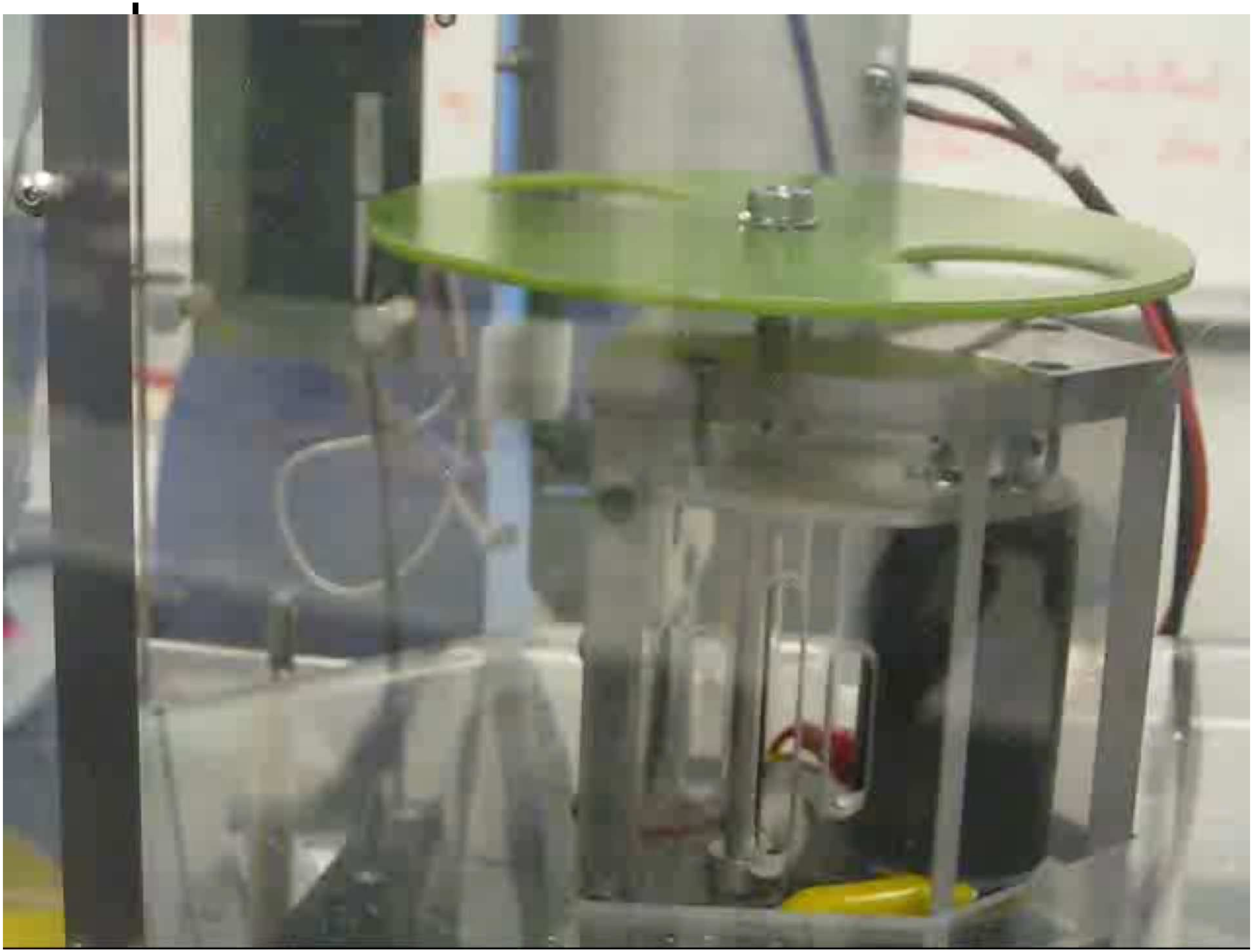


- *Tunneling Ball Device*
  - *sense ball*
  - *track disk*
  - *adjust trajectory*



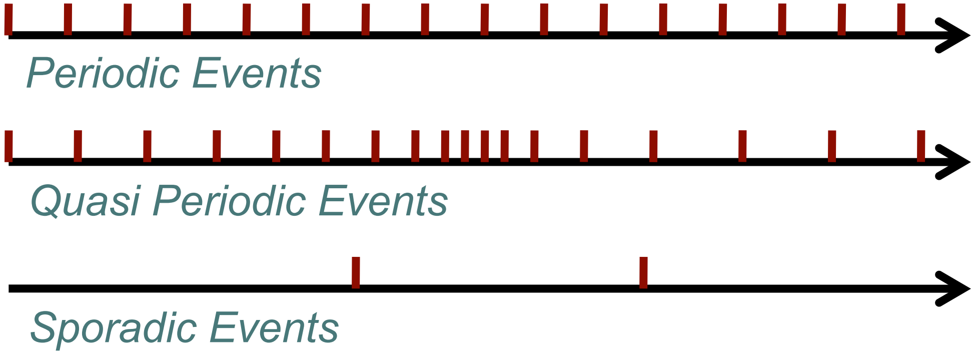
Tunneling Ball Device in Action





Tunneling Ball Device

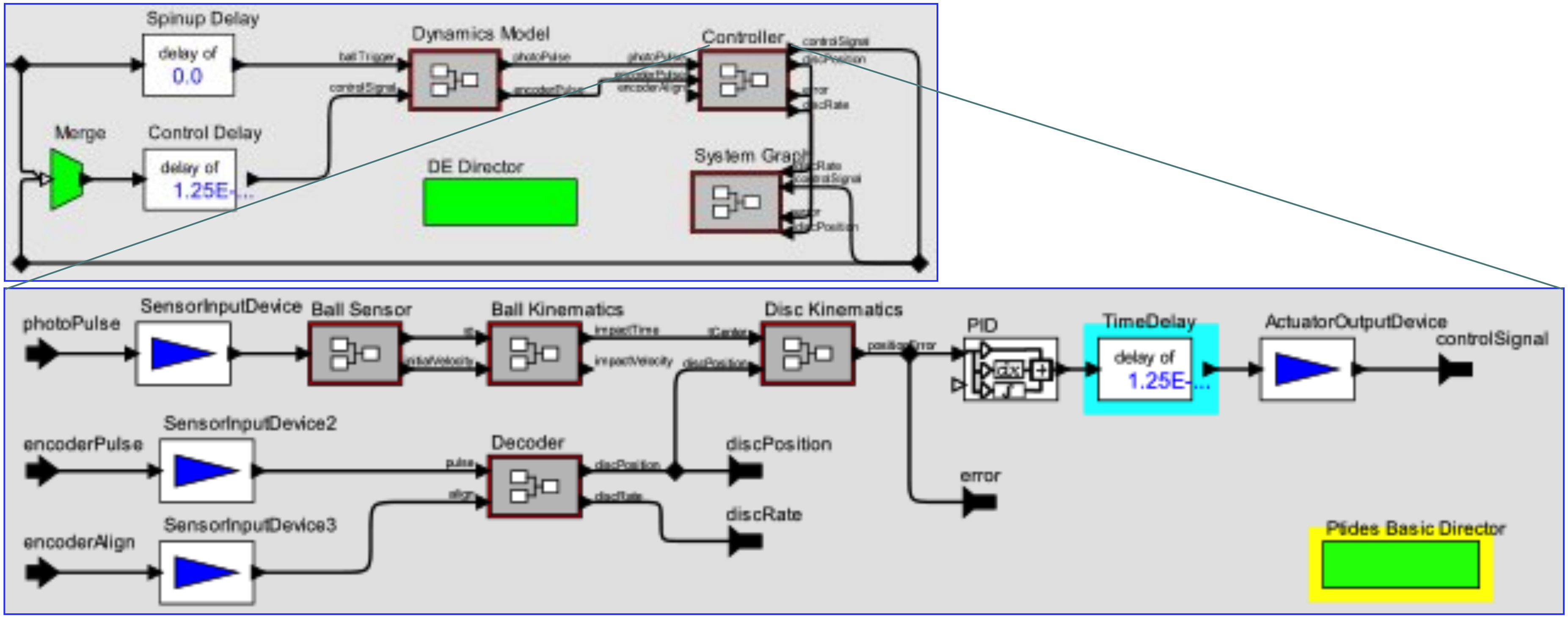
Mixed event sequences



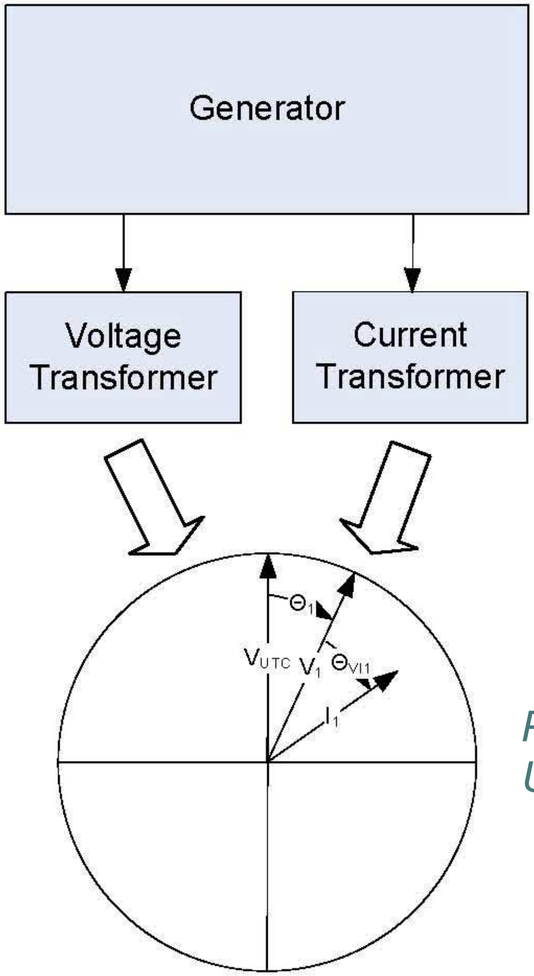
Periodic Events

Quasi Periodic Events

Sporadic Events

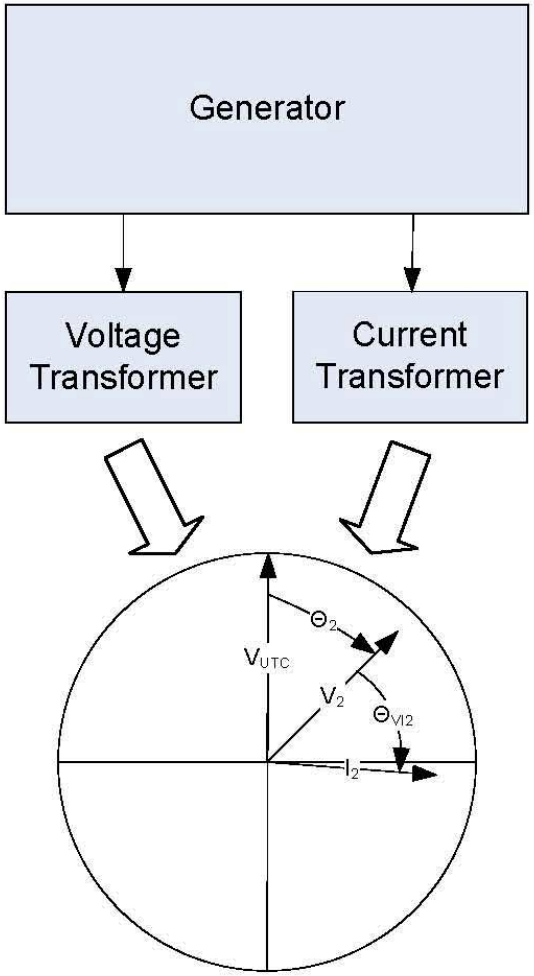


Second Test Case: Distributed Synchrophasor Measurement & Control



Synchrophasor data: Θ_1, V_1 ,I_1 ,Θ_{VI1}
Power factor: $\cos(\Theta_{VI1})$

Power swing and Unstability detection



Synchrophasor data: Θ_2, V_2 ,I_2 ,Θ_{VI2}
Power factor: $\cos(\Theta_{VI2})$

Thanks to Vaselin Skendzic, Schweitzer Engineering

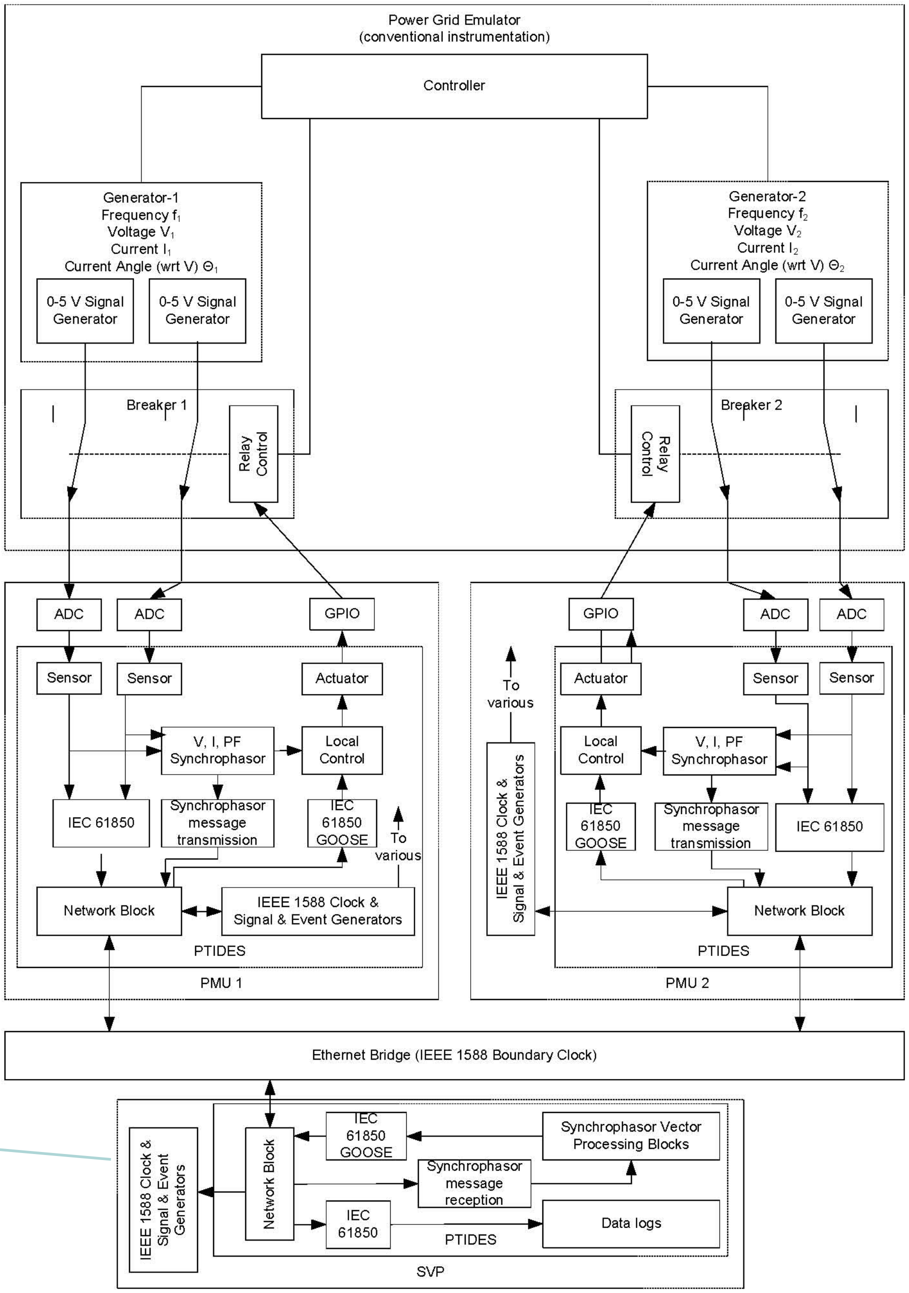
Experiment Diagram

Grid emulator built with National Instruments PXI

'Primary Measurement Unit (PMU) built with Renesas demo boards with DP83640

Ethernet bridge or 1588 boundary/transparent clock

Synchrophasor Vector Processing unit (SVP) built with Renesas demo board with DP83640



Thanks to Vaselin Skendzic, Schweitzer Engineering

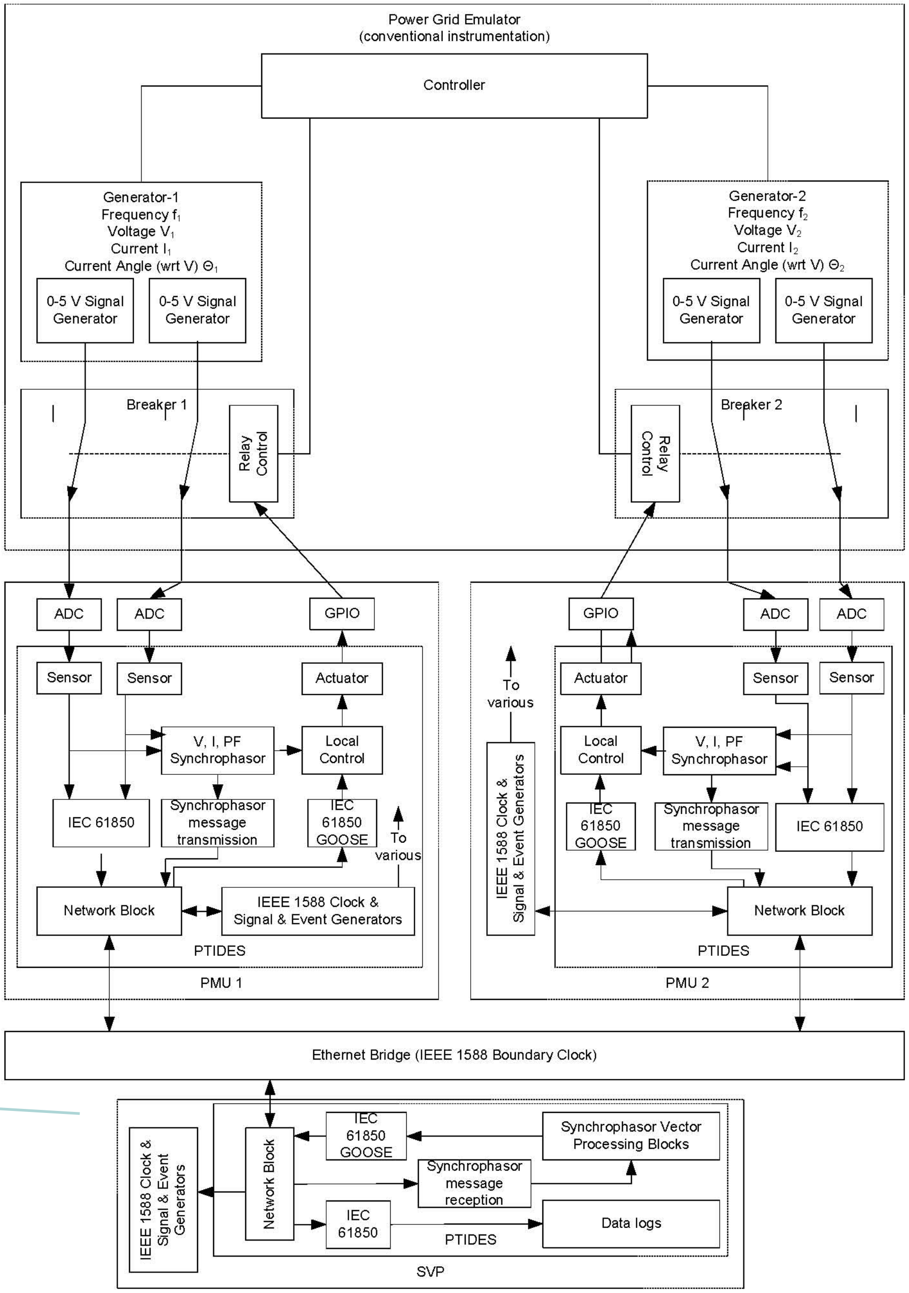
Basic PTIDES Timing Testing

Vary phasor data independently
Freq. and phases w.r.t. global time

Sample voltage and current
Signal processing
Send phasor data
Local control: on/off breaker

Wireshark monitoring of network events

Detect discrepancies
If unstable region send on/off command



Thanks to Vaselin Skendzic, Schweitzer Engineering

Current Status (as of Oct. 2010)

- Prototype of PtidyOS executes on single Luminary Micro (ARM platform)
- Overhead of event processing is still too high in this prototype. We are working on optimizations (e.g. dispatching certain events without putting them on the EQ).
- Realizing IEEE 1588 synchronized time service on Renesas board.
- Porting PtidyOS to Renesas board.

Summary

- Network time synchronization is a potentially game-changing advance for distributed embedded systems.

- The PTIDES model of computation offers an attractive possible programming model for distributed cyber-physical systems.

- Synthesis of embedded software from PTIDES models seems feasible, though performance improvements are still needed.

Future Work

- Schedulability analysis to statically determine whether deadlines at actuators will always be met (the question is undecidable in general, but decidable for some cases).

- Improving code generator to use more sophisticated metaprogramming techniques (such as EMF & openarchitectureware).