Time for High-Confidence Distributed Embedded Systems

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Keynote Talk

International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication

ISPCS

September 26, 2012

Key collaborators:
- *Patricia Derler*
- *John Eidson*
- *Slobodan Matic*
- *Sanjit Seshia*
- *Yang Zhao*
- *Michael Zimmer*
- *Jia Zou*

The Short Version of My Talk

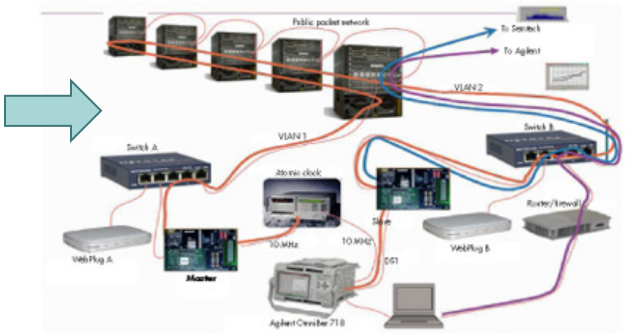
Time synchronization is going to change the world (again)

Gregorian Calendar (BBC history)

1500s
days

Lackawanna Railroad Station, 1907, Hoboken. Photograph by Alicia Dudek

1800s
seconds



2005: first IEEE 1588 plugfest

2000s
nanoseconds

Today’s networks

“On August 12, 1853, two trains on the Providence & Worcester Railroad were headed toward each other on a single track. The conductor of one train thought there was time to reach the switch to a track to Boston before the approaching train was scheduled to pass through. But the conductor's watch was slow. As his speeding train rounded a blind curve, it collided head-on with the other train—fourteen people were killed. The public was outraged. All over New England, railroads ordered more reliable watches for their conductors and issued stricter rules for running on time.”

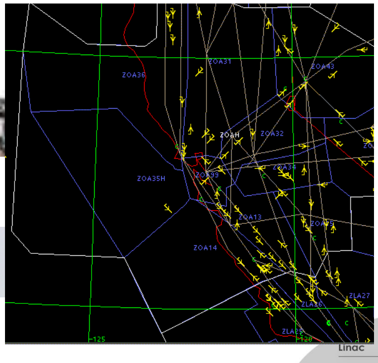
Source: National Museum of American History

What is going to change? Design of Cyber-Physical Systems (CPS)

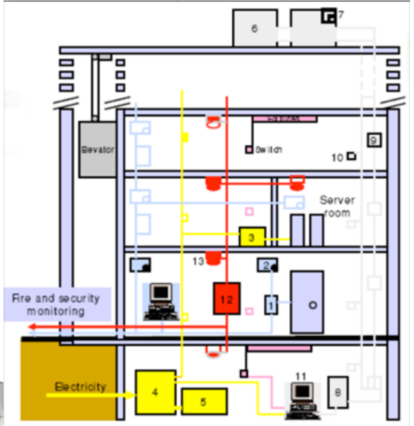
Orchestrating networked computational resources with physical systems

Avionics

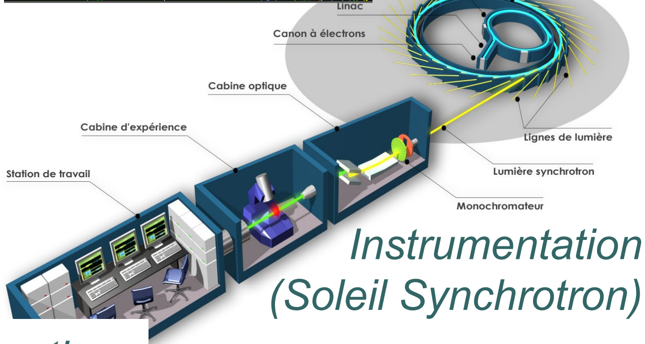
Transportation (Air traffic control at SFO)



Building Systems

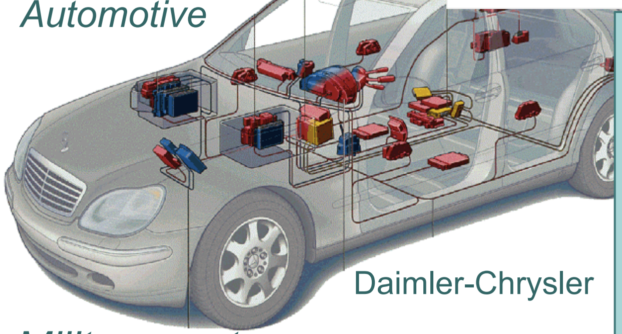


Telecommunications



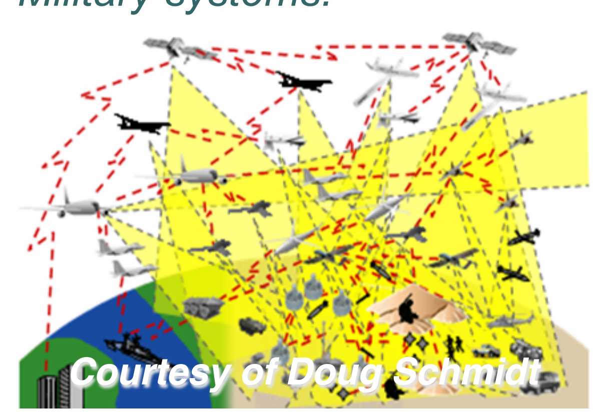
Instrumentation (Soleil Synchrotron)

Automotive



Daimler-Chrysler

Military systems:

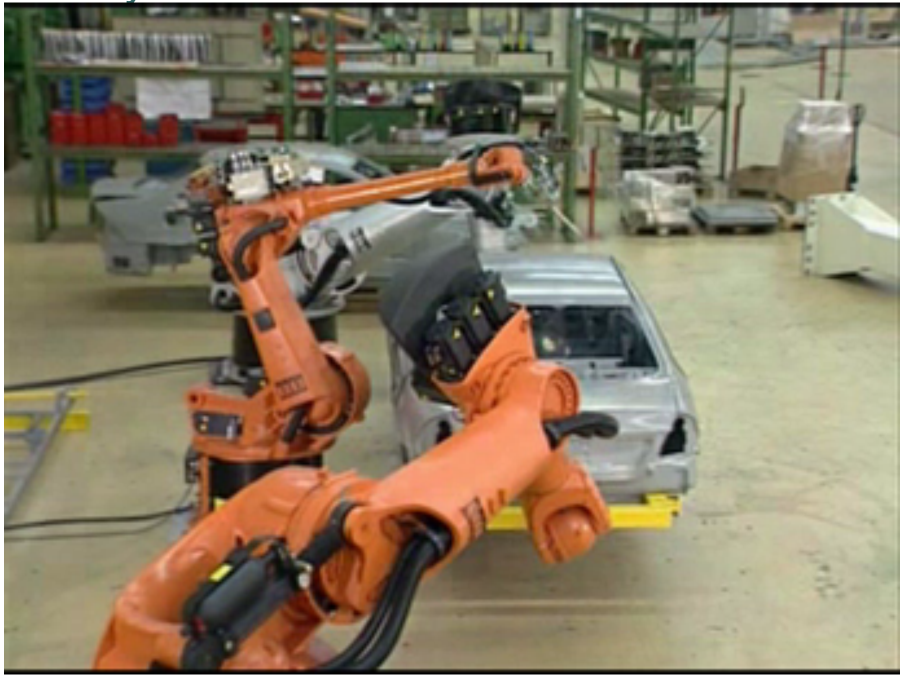


Courtesy of Doug Schmidt

Power generation and distribution

Courtesy of General Electric

Factory automation



Courtesy of Kuka Robotics Corp.

A Cyber-Physical System
Printing Press

Bosch-Rexroth

Hundreds of microcontrollers and an Ethernet network are orchestrated with precisions on the order of microseconds.

Software for such systems can be developed in a completely new way.

Time synchronization enables tightly coordinated actions and reliable networking with bounded latency.

But software technology will need to adapt to take advantage of this revolution…

For cyber-physical systems,
programs do not adequately specify *behavior*.

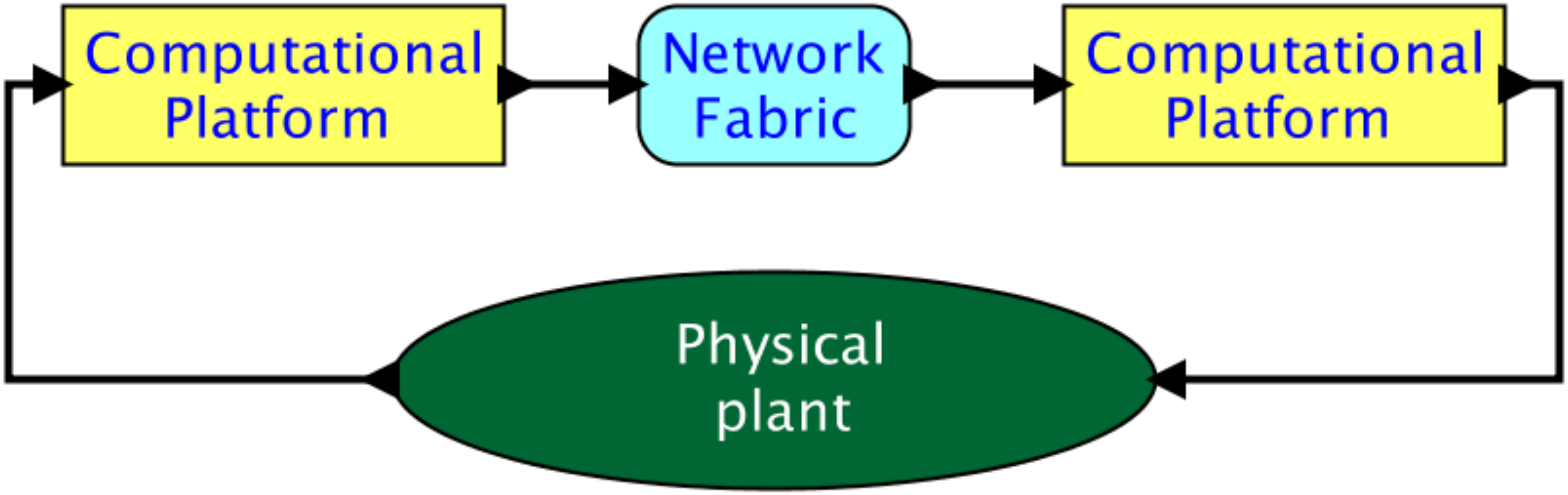
The core notions of "computation" today ignore time.

The core notions tomorrow will not…

Challenges that We are Addressing

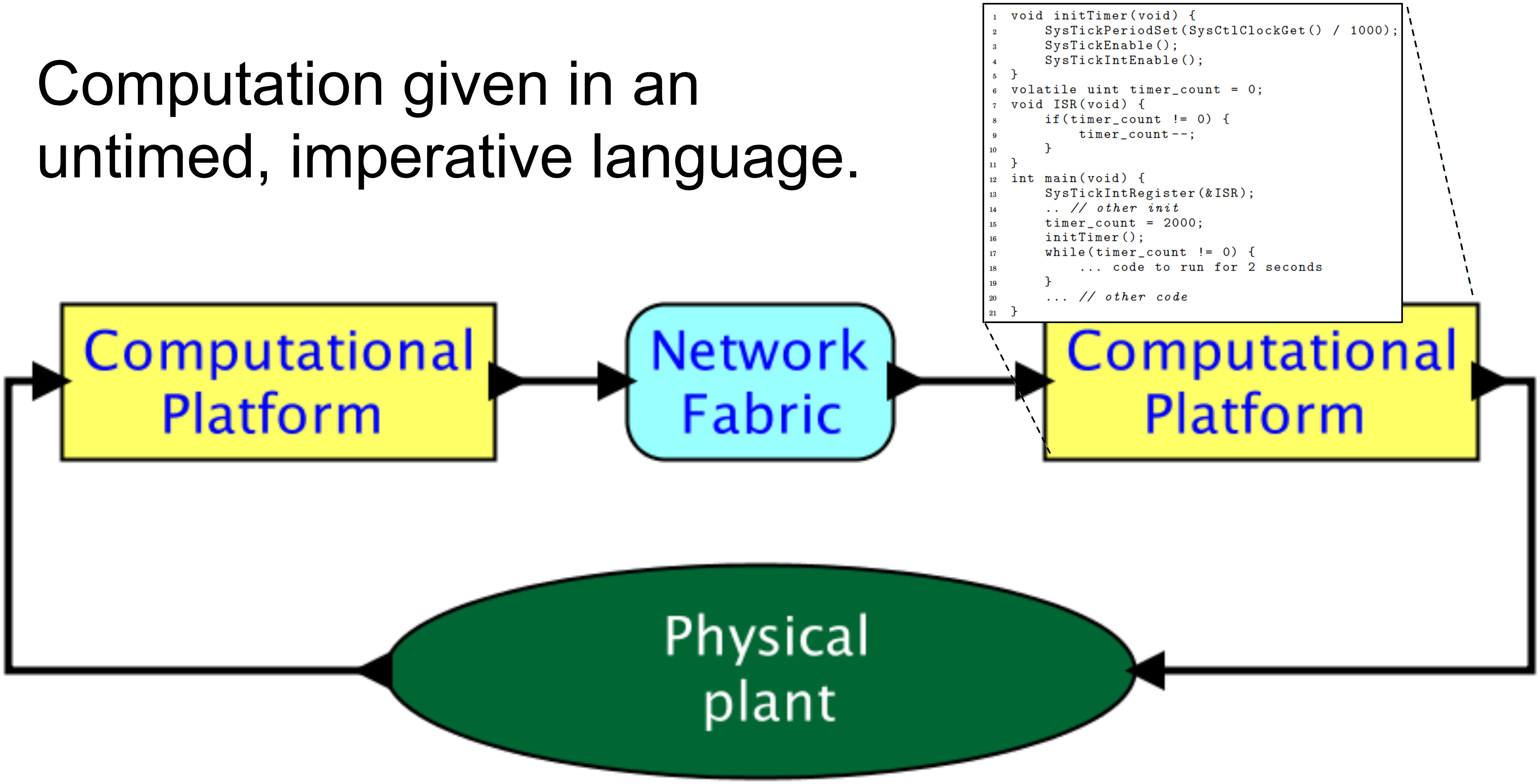
- Representation of time
  - Data types, superdense time, operations on time, etc.
- Gaining control over timing in software
  - PRET – Precision-timed computer architecture
- Multiform time
  - Hierarchical clocks proceeding at different rates.
- Joint modeling of functionality and implementation
  - Timing emerges from the implementation
- Programming models that specify timed behavior
  - PTIDES – A programming model for distributed systems

Schematic of a simple CPS:



Assume synchronized clocks.
How can the software take
advantage of this?

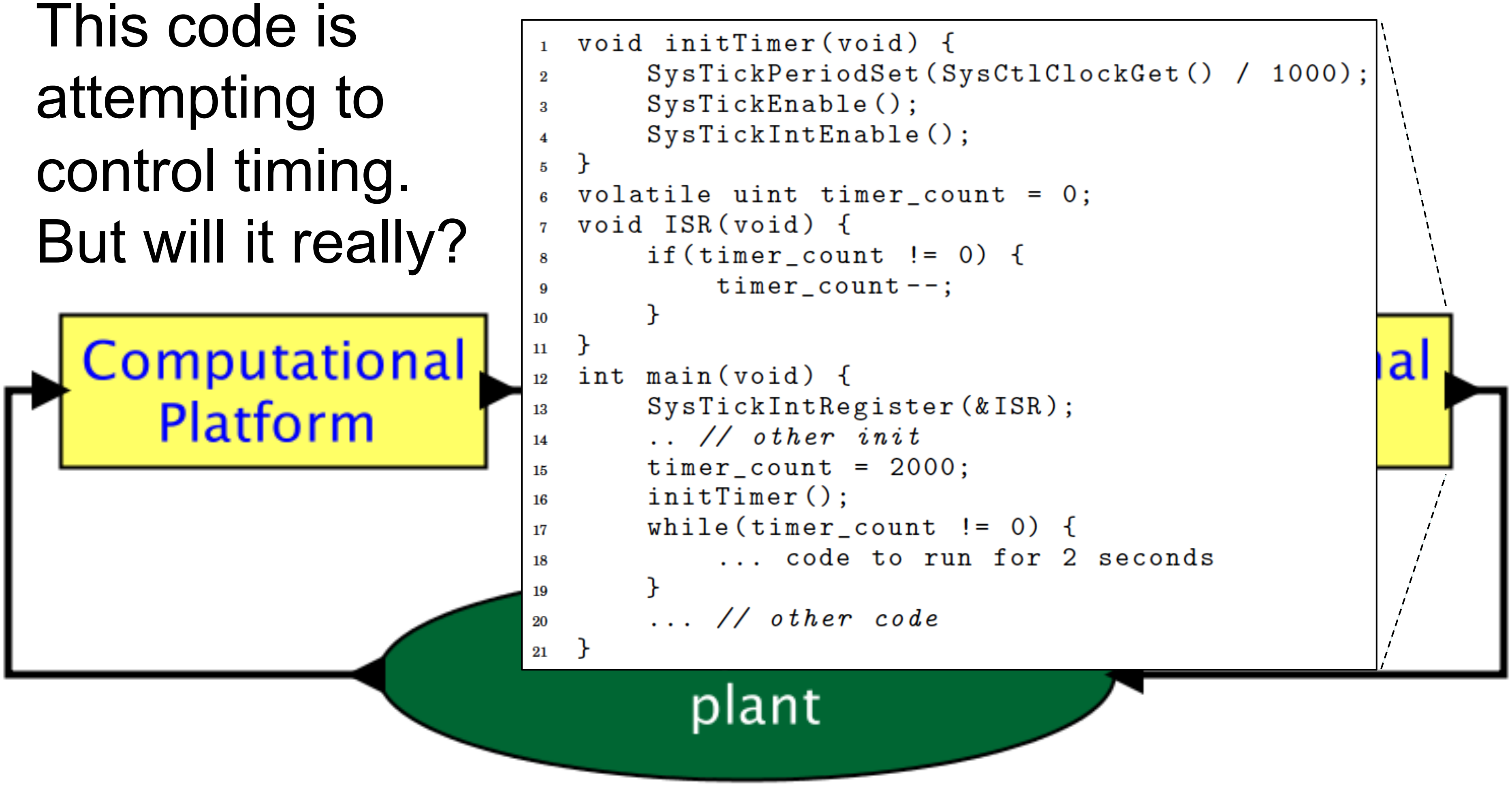
Computation given in an untimed, imperative language.



```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```

Assume synchronized clocks. How can the software take advantage of this?

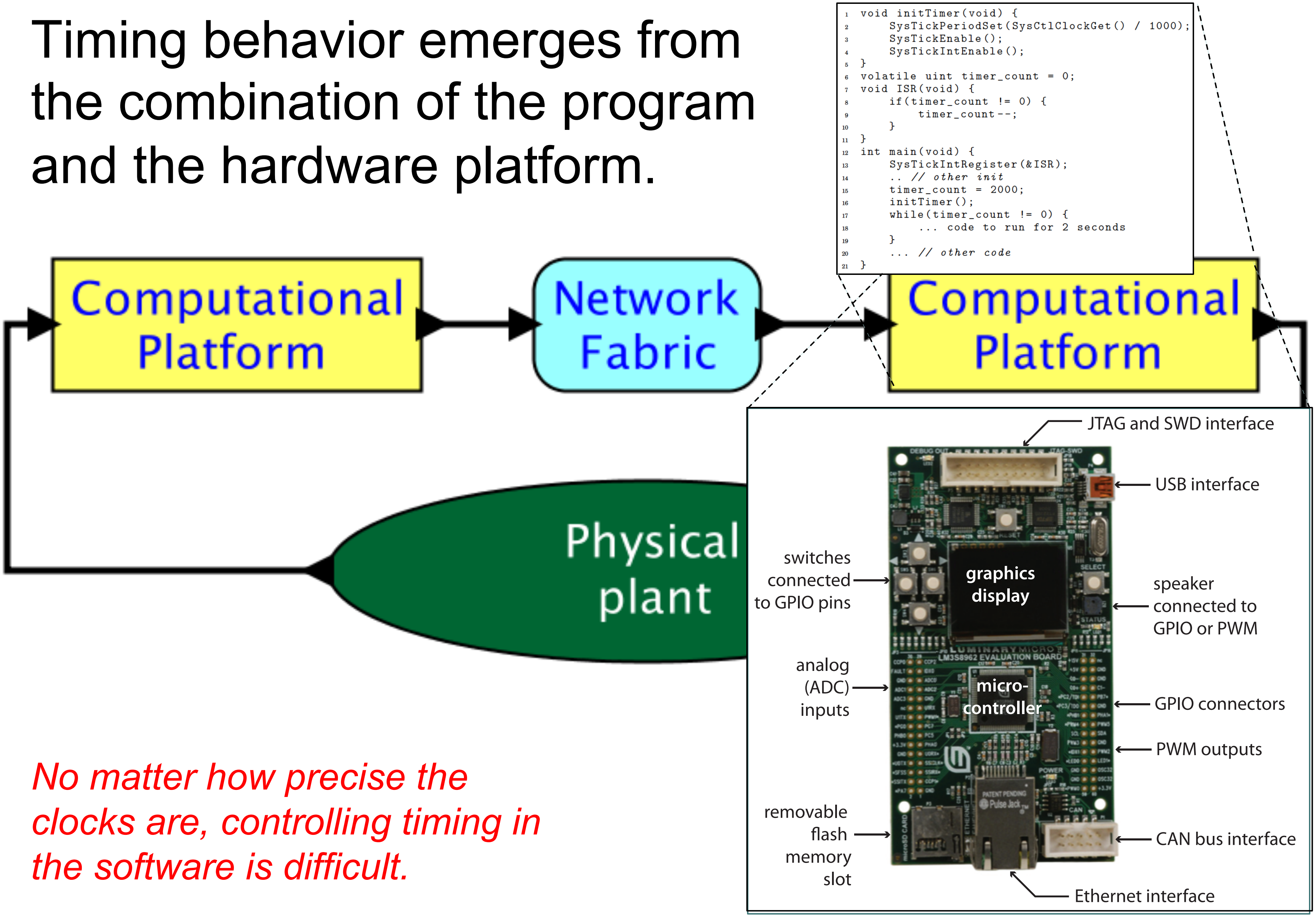
This code is attempting to control timing. But will it really?



```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```

Assume synchronized clocks. How can the software take advantage of this?

Timing behavior emerges from the combination of the program and the hardware platform.



```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```

No matter how precise the clocks are, controlling timing in the software is difficult.

Consequences

When precise control over timing is needed, designs are brittle. Small changes in the hardware, software, or environment can cause big, unexpected changes in timing. Testing has to be redone. Results:

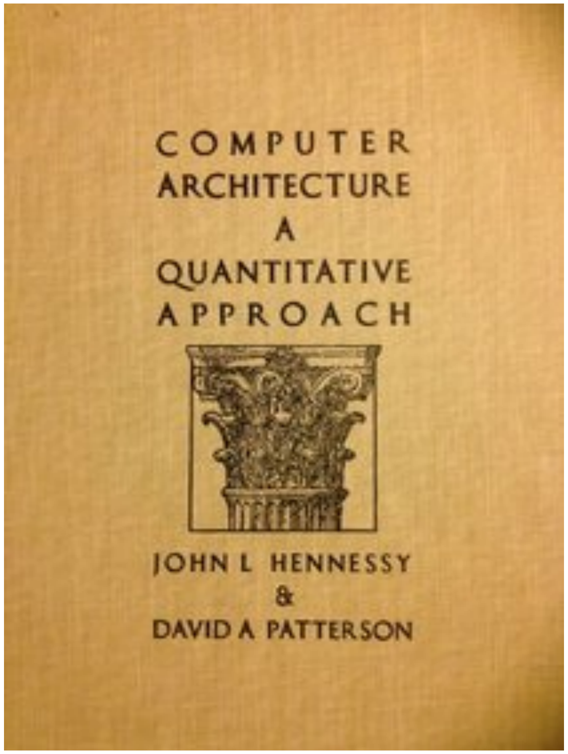
- Manufacturers frequently stockpile parts to suffice for the complete production run of a product.
- Manufacturers cannot take advantage of improvements in the hardware (e.g. weight, power). The cost of re-testing and re-certifying is too high.
- Designs are over provisioned, increasing cost, weight, and energy usage.

A Key Challenge:
Timing is not Part of Software Semantics

***Correct** execution of a program in C, C#, Java, Haskell, OCaml, etc. has nothing to do with how long it takes to do anything. Nearly all our computation and networking abstractions are built on this premise.*

Programmers have to step *outside* the programming abstractions to specify timing behavior.

Computer Science has not *ignored* timing…



The first edition of Hennessy and Patterson (1990) revolutionized the field of computer architecture by making performance metrics the dominant criterion for design.

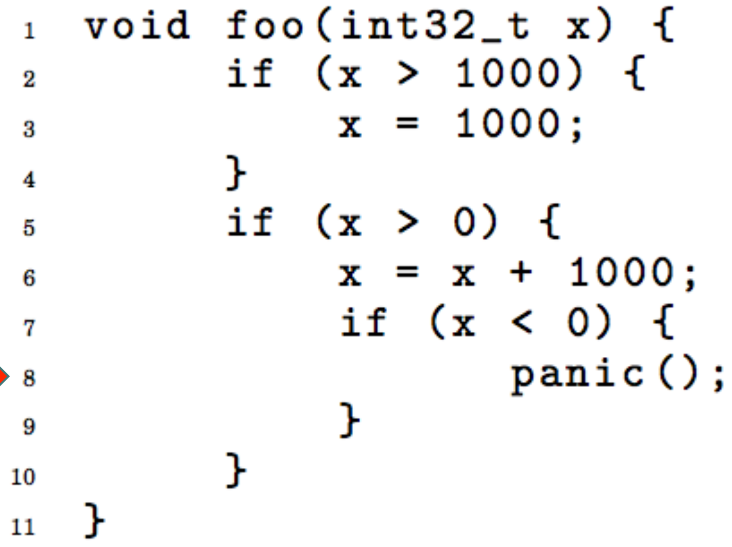
Today, for computers, timing is merely a performance metric.

It needs to be a correctness criterion.

Correctness criteria

We can safely assert that line 8 does not execute

(In C, we need to separately ensure that no other thread or ISR can overwrite the stack, but in more modern languages, such assurance is provided by construction.)



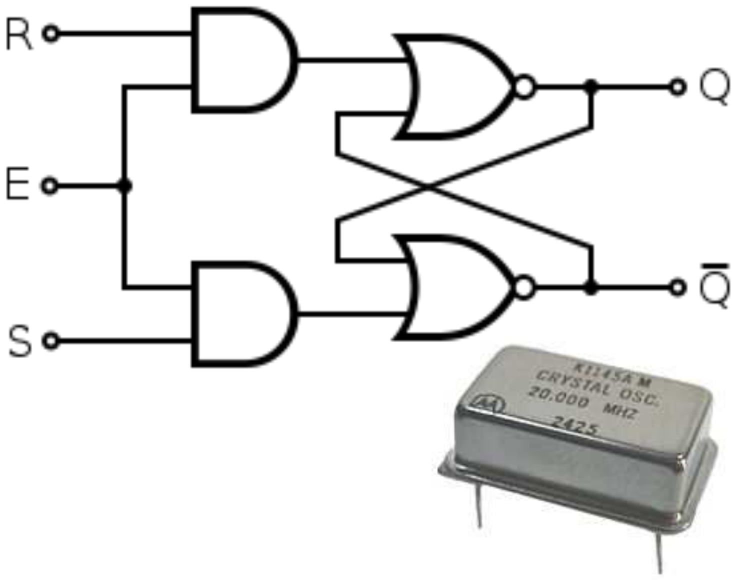
```
 1  void foo(int32_t x) {
 2          if (x > 1000) {
 3                  x = 1000;
 4          }
 5          if (x > 0) {
 6                  x = x + 1000;
 7                  if (x < 0) {
 8                          panic();
 9                  }
10          }
11  }
```

*We can develop **absolute confidence** in the software, in that only a **hardware failure** is an excuse.*

But not with regards to timing!!

The hardware out of which we build computers is capable of delivering “correct” computations and precise timing…

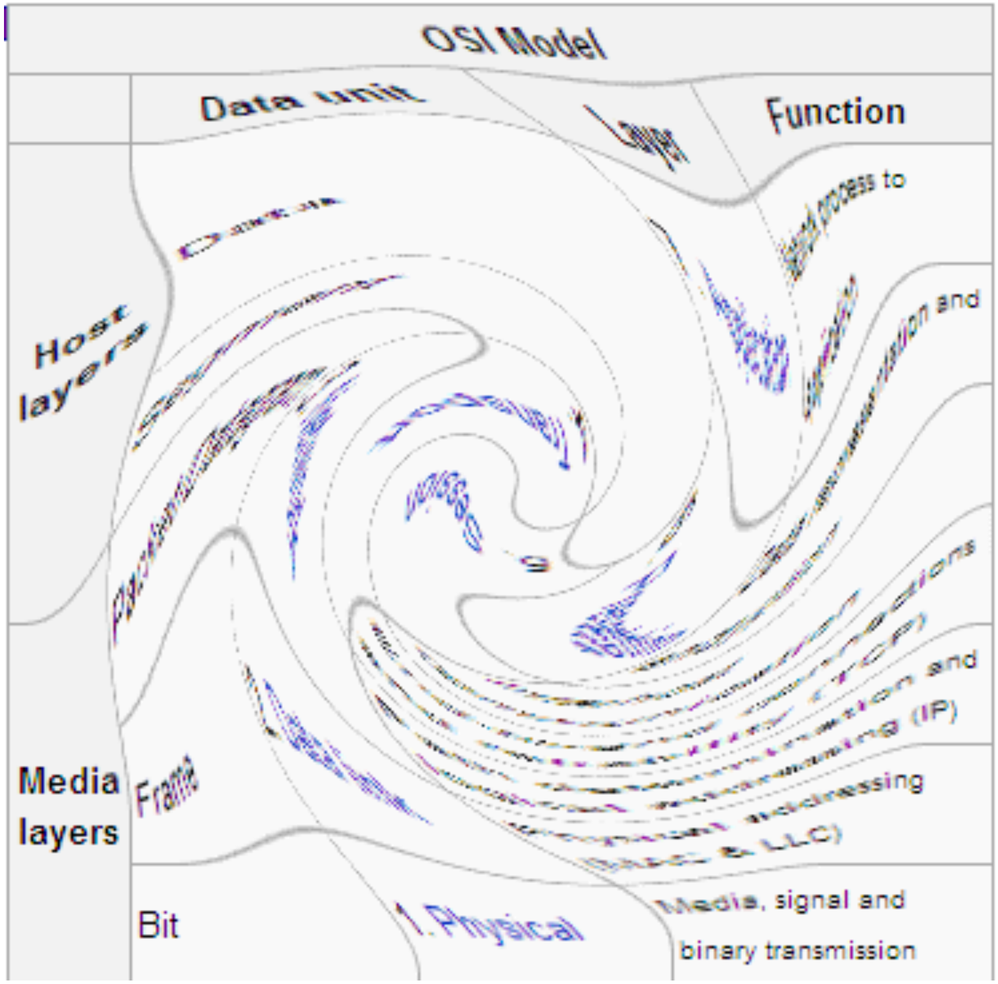
The synchronous digital logic abstraction removes the messiness of transistors.



… but the overlaying software abstractions discard the timing precision.

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

As with processors, for networks, timing is a *performance metric*, not a *correctness criterion*



The point of these abstraction layers is to isolate a system designer from the details of the implementation below.

In today's networks, timing emerges from the details of the implementation.

Even QoS-aware networks (e.g. AVB) derive timing properties from packet priorities & network topology.

Project 1: (which I will not talk about today) PRET Machines

- **PRE**cision-**T**imed processors = **PRET**
- **P**redictable, **RE**peatable **T**iming = **PRET**
- **P**erformance *with* **RE**peatable **T**iming = **PRET**

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

+

= PRET

Computing

With time

Project #2: *PTIDES*

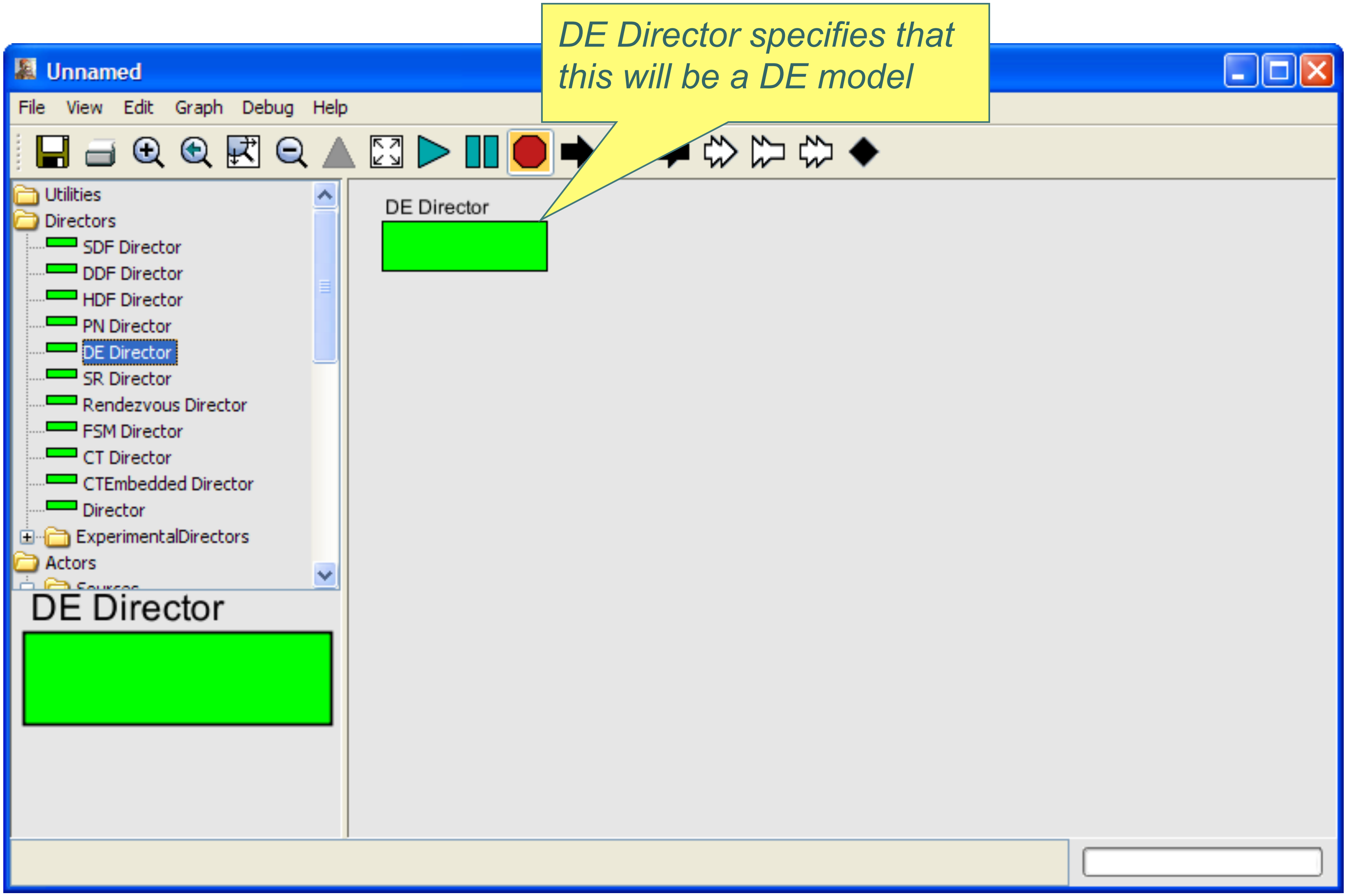
A Programming Model for Distributed Cyber-Physical Systems Based on Discrete Events (DE)

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

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

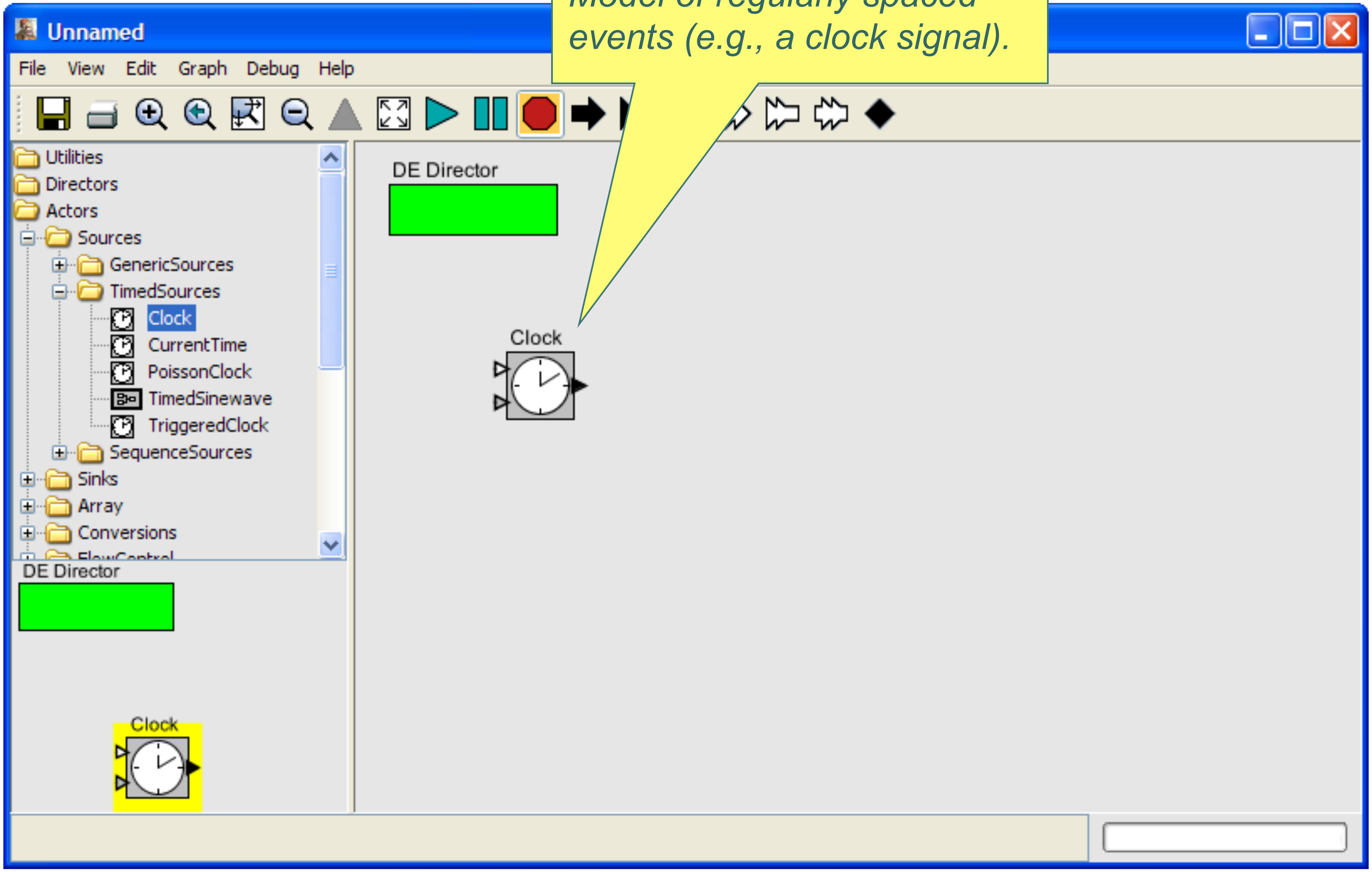
PTIDES leverages network time synchronization to deliver determinate distributed real-time computation.

Discrete-Event Models (in Ptolemy II)



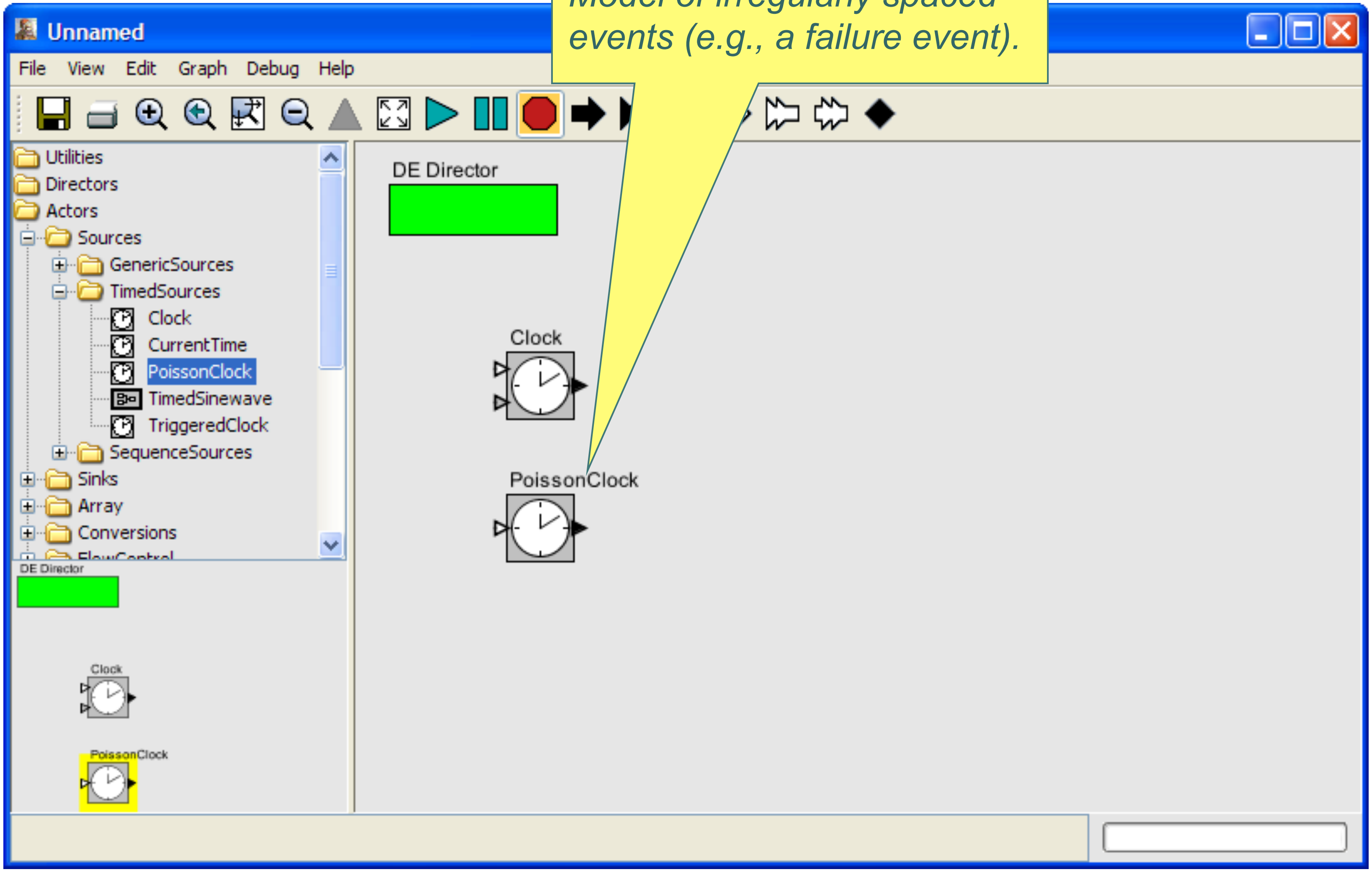
DE Director specifies that this will be a DE model

Discrete-Event Models (in Ptolemy II)



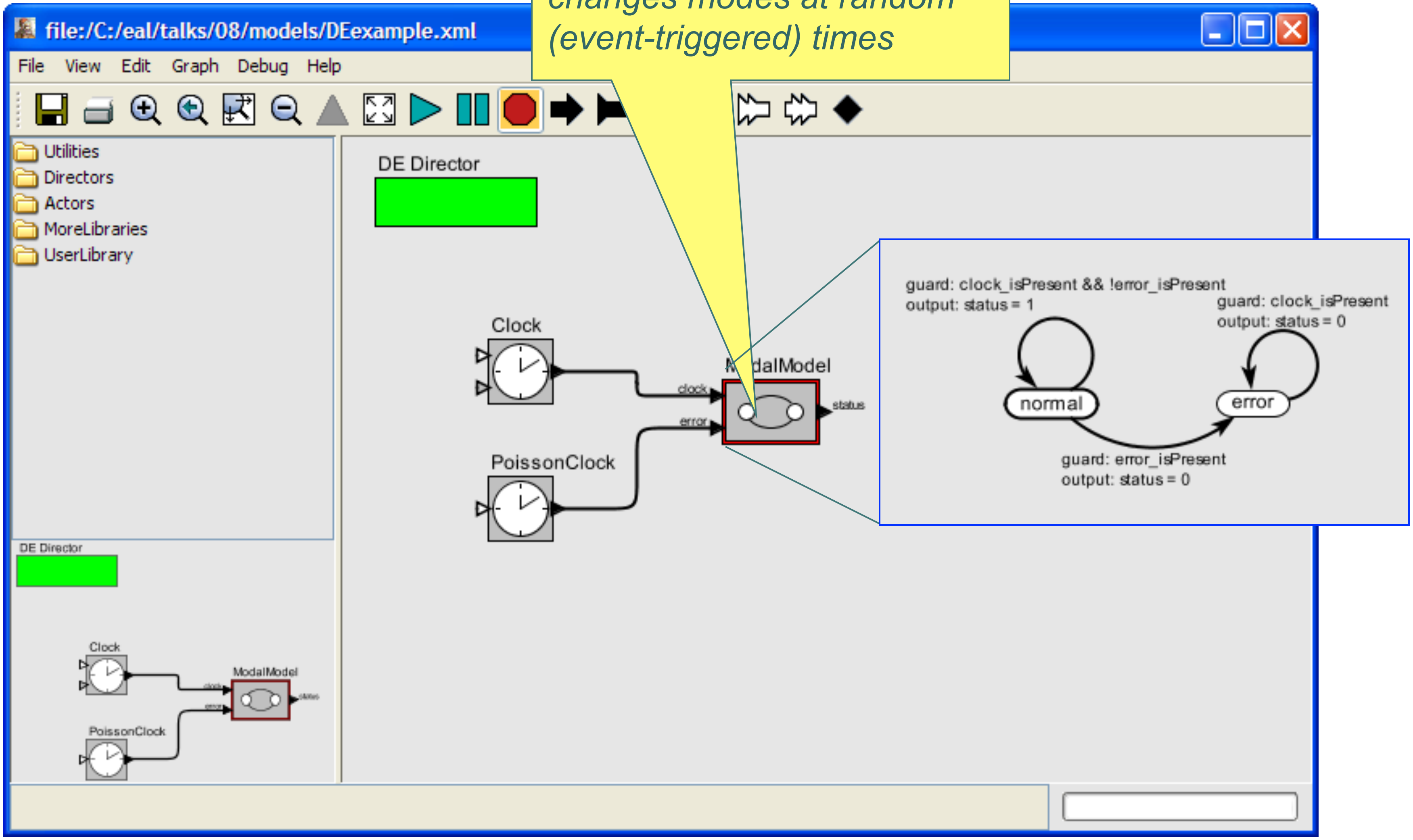
Model of regularly spaced events (e.g., a clock signal).

Discrete-Event Models (in Ptolemy II)



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

Discrete-Event Models (in Ptolemy II)



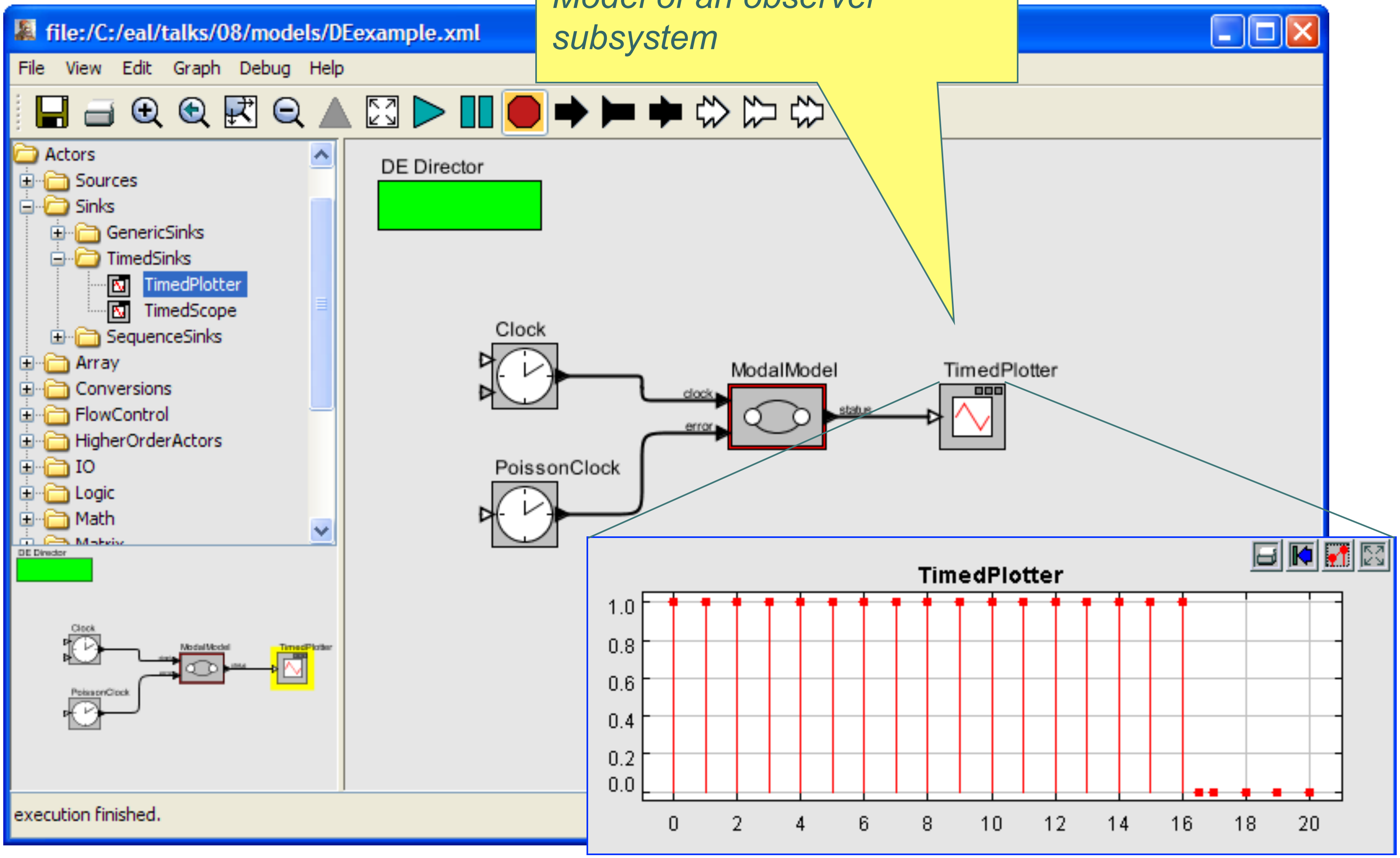
guard: clock_isPresent && !error_isPresent
output: status = 1

guard: clock_isPresent
output: status = 0

guard: error_isPresent
output: status = 0

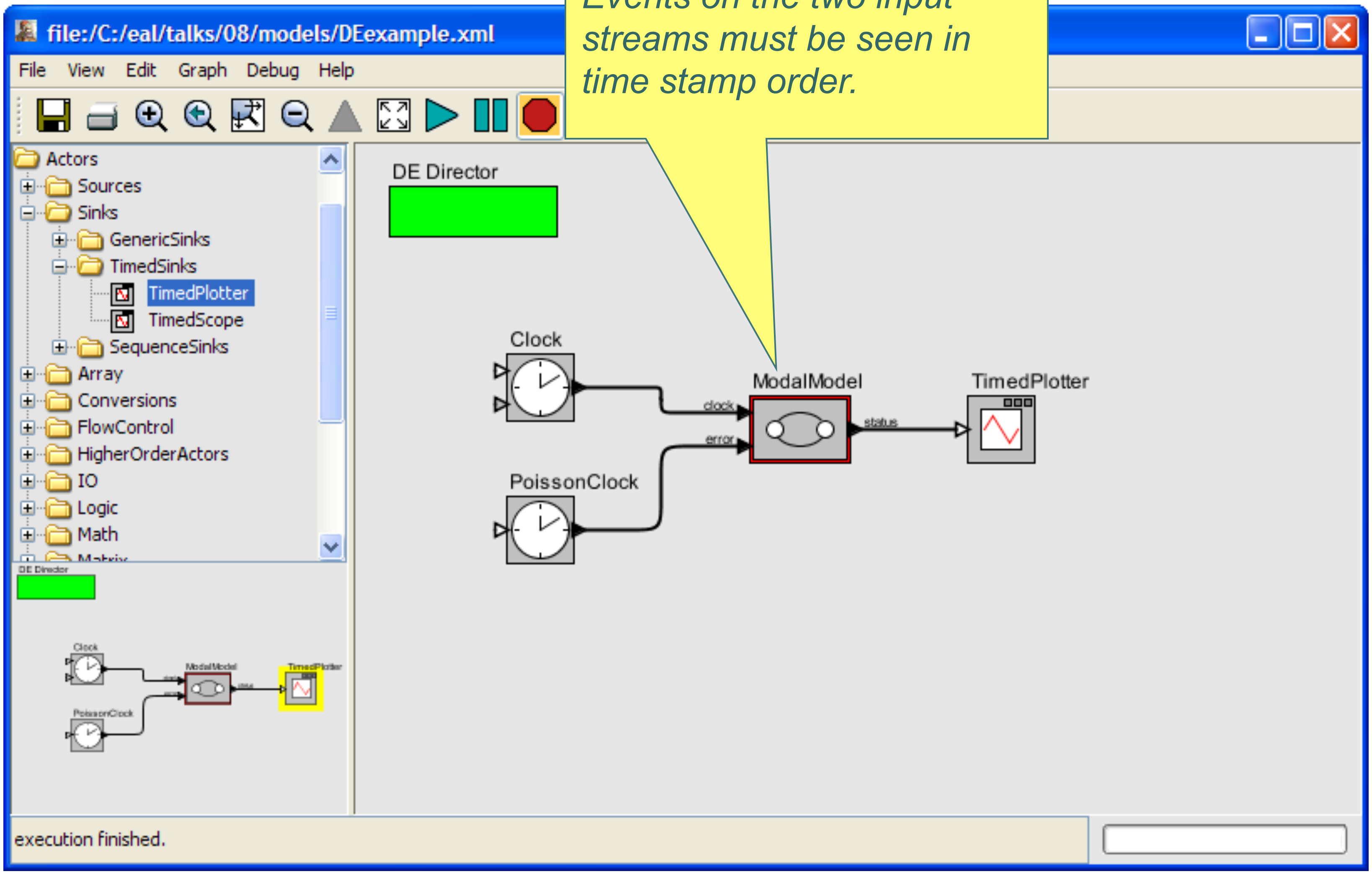
Model of a subsystem that changes modes at random (event-triggered) times

Discrete-Event Models (in Ptolemy II)



Model of an observer subsystem

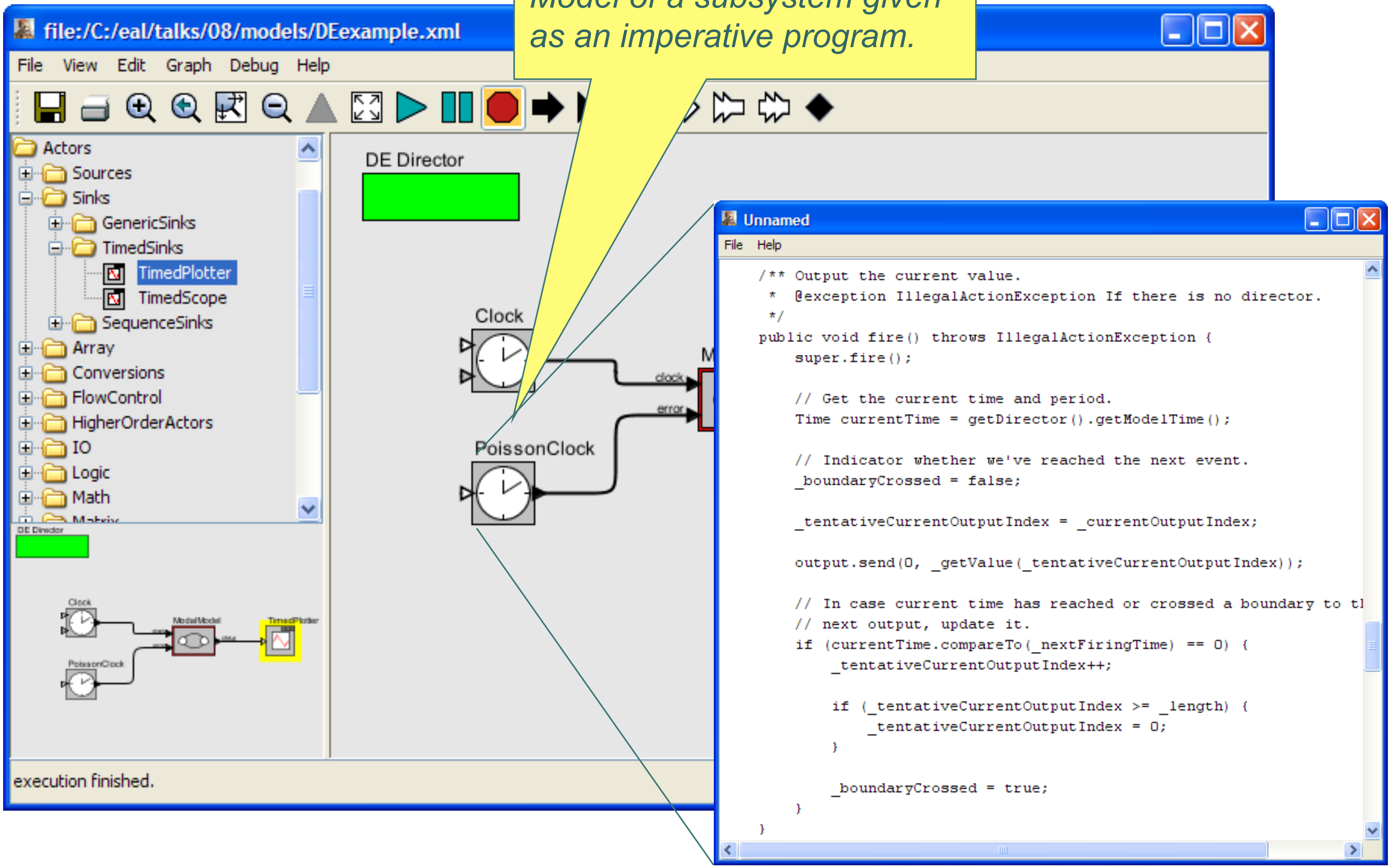
Discrete-Event Models (in Ptolemy II)



execution finished.

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

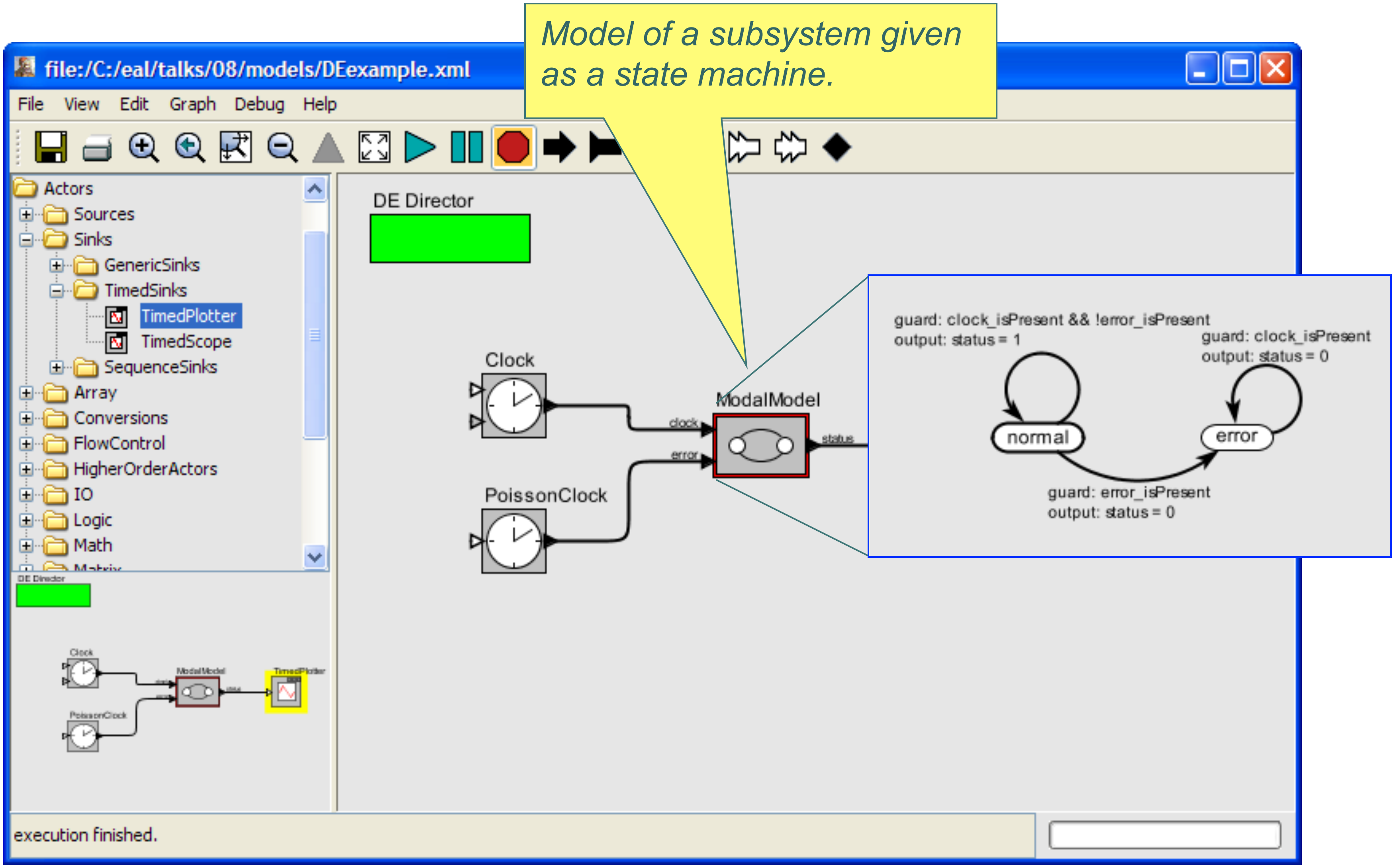
This is a Component Technology



Model of a subsystem given as an imperative program.

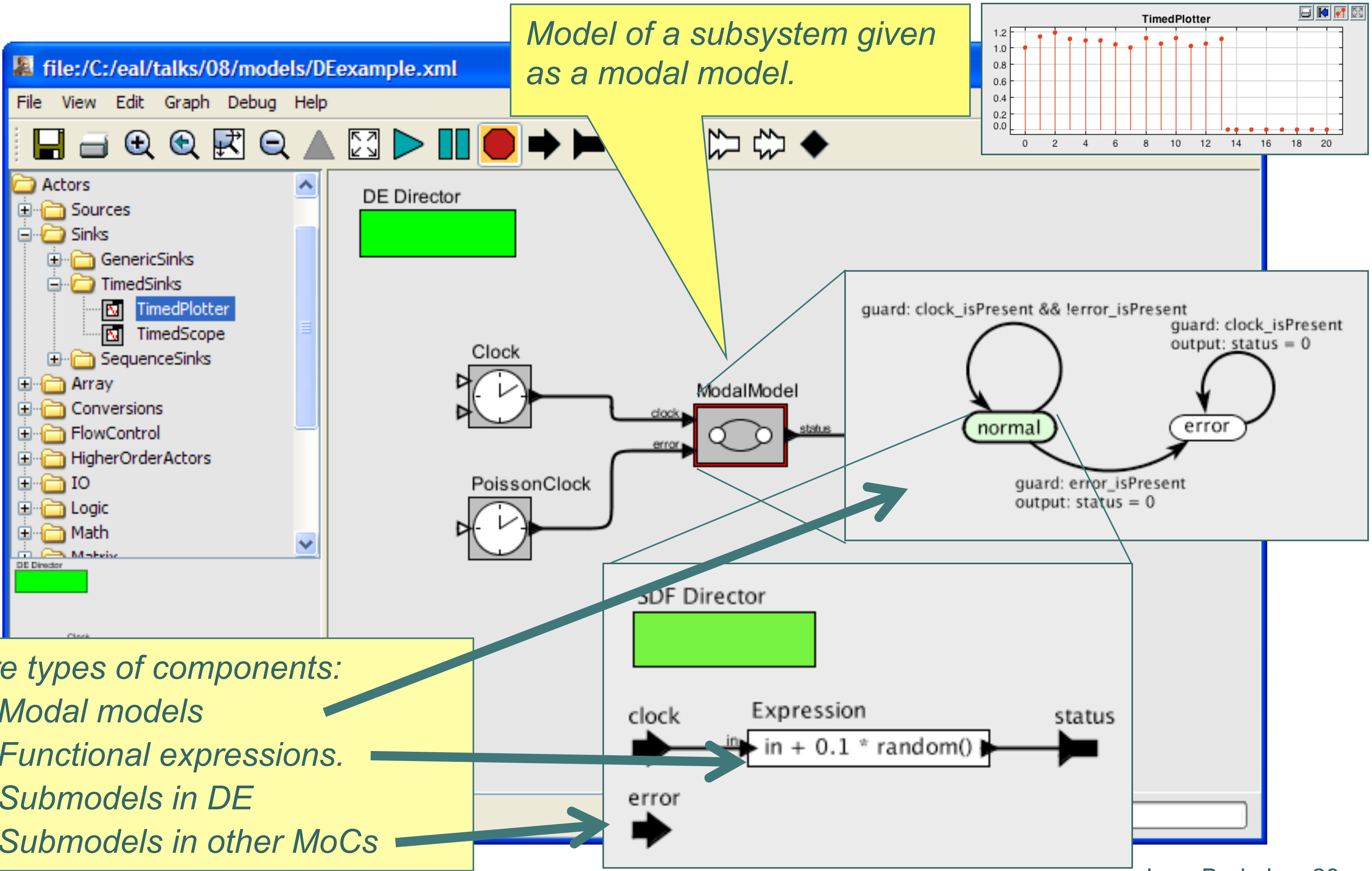
```
/** Output the current value.
 *  @exception IllegalActionException If there is no director.
 */
public void fire() throws IllegalActionException {
    super.fire();

    // 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.

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 Discrete Event Semantics in Distributed Real-Time Systems

- DE is usually used for simulation (HDLs, network simulators, …)
- Distributing DE is done to accelerate simulation.

- We are using DE for distributed real-time software, binding time stamps to real time only where necessary.

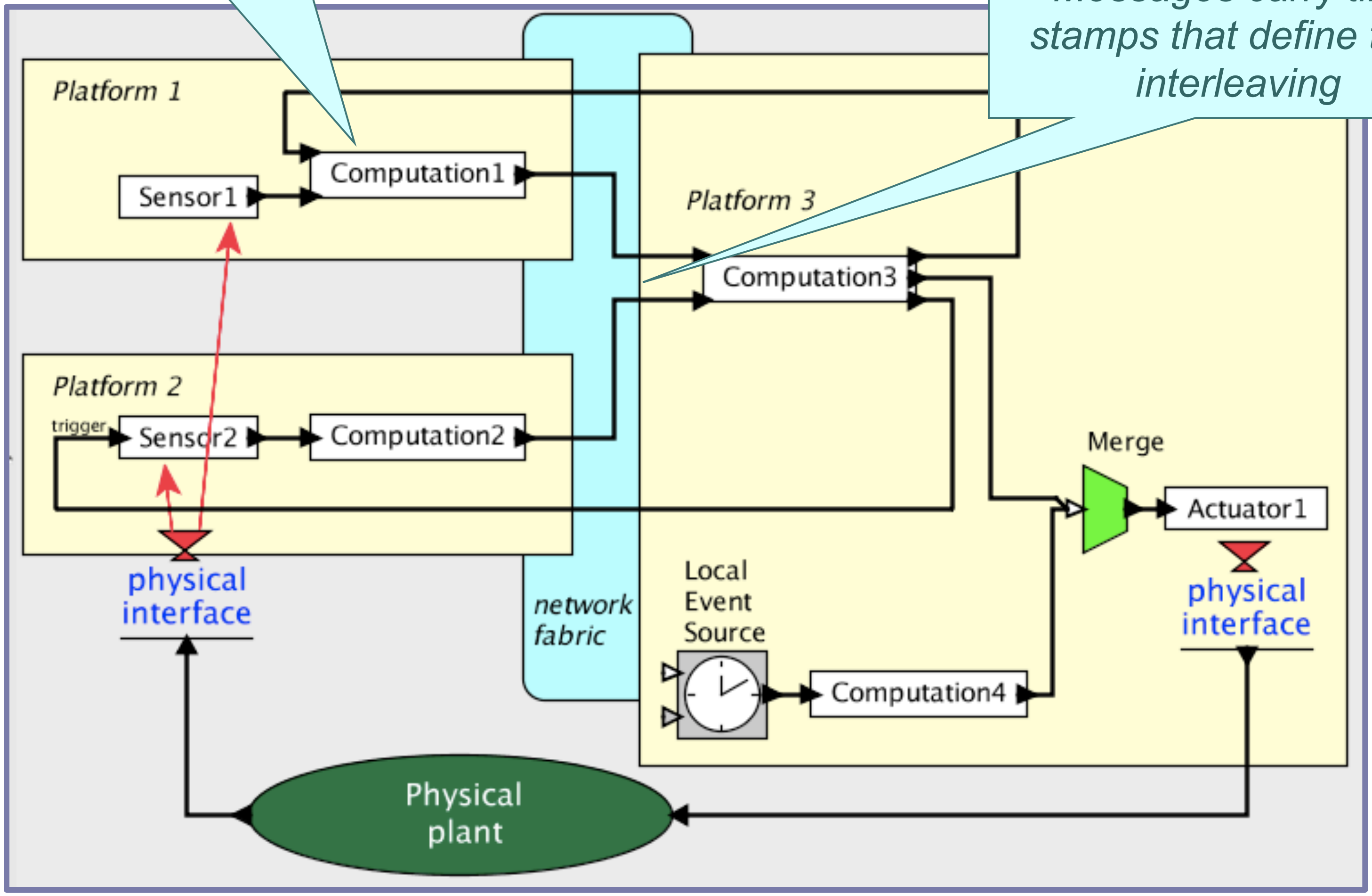
- ***PTIDES***: Programming Temporally Integrated Distributed Embedded Systems

Ptides: Programming Temporally Integrated Distributed Embedded Systems

First step: Time-stamped messages.

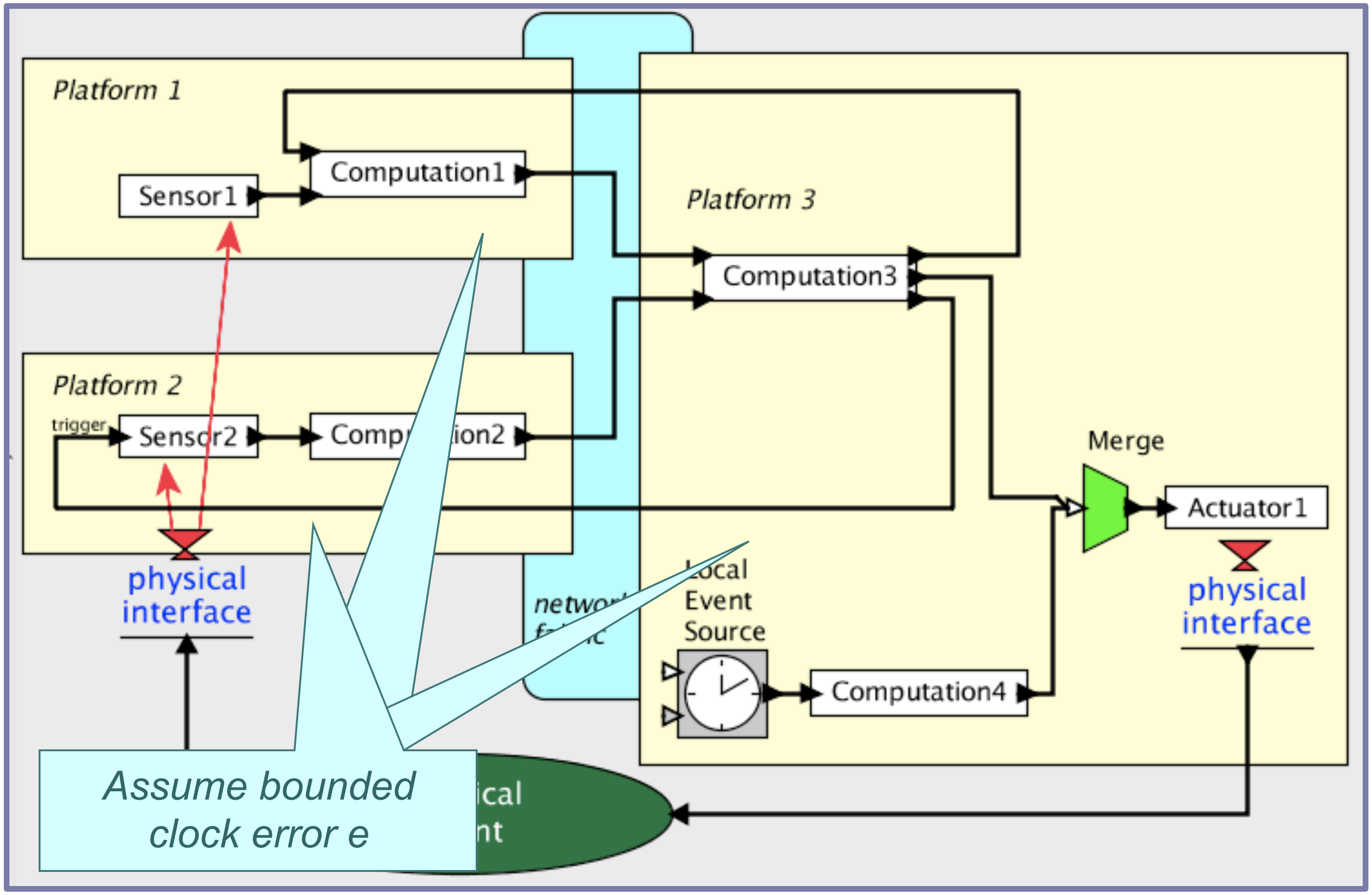
Actors specify computation

Messages carry time stamps that define their interleaving



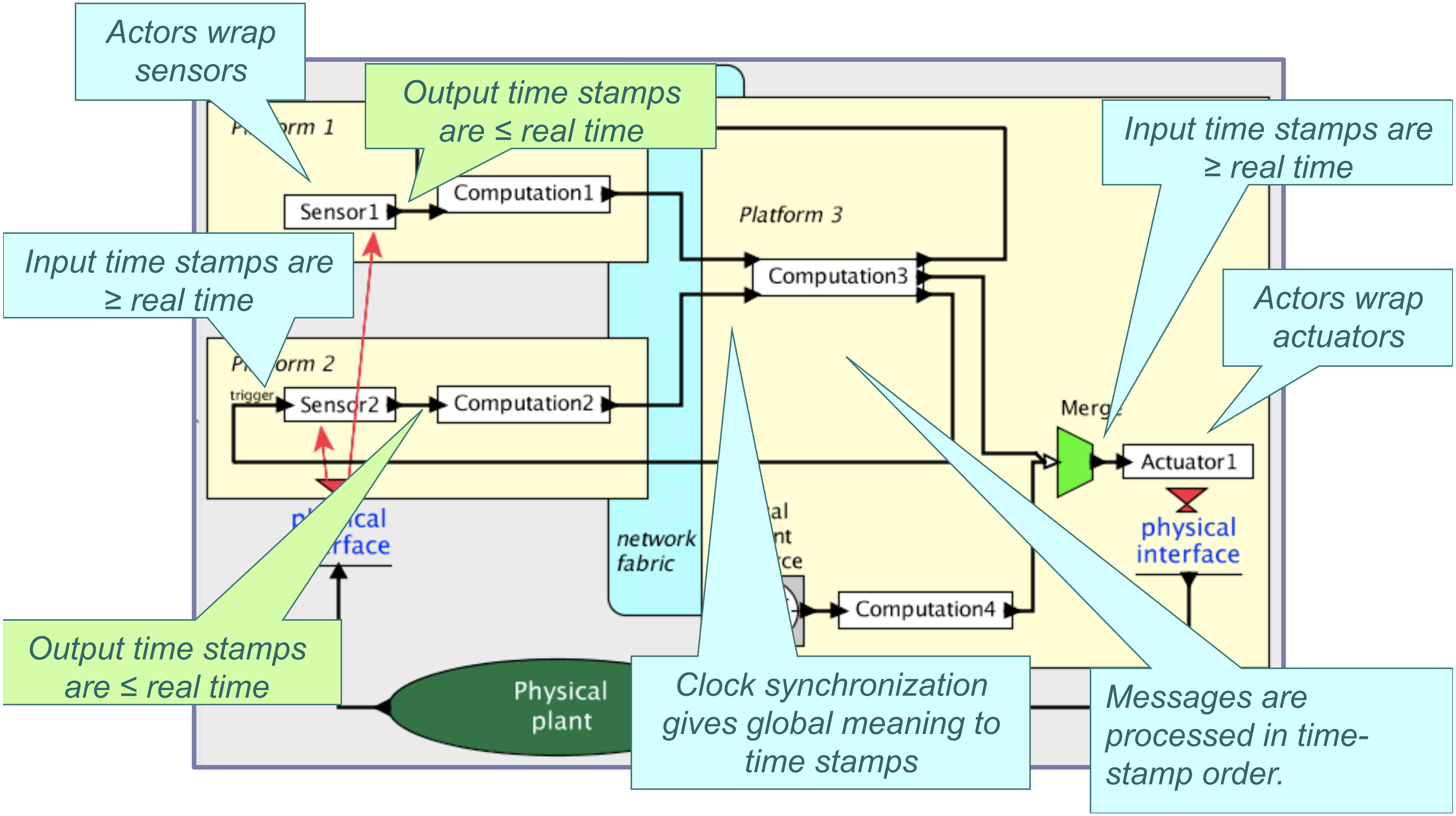
Ptides: Second step: Network time synchronization

GPS, NTP, IEEE 1588, time-triggered busses, … they all work. We just need to bound the clock synchronization error.



Assume bounded clock error e

Ptides: Third step:
Bind time stamps to real time at sensors and actuators



Actors wrap sensors

Output time stamps are ≤ real time

Input time stamps are ≥ real time

Actors wrap actuators

Input time stamps are ≥ real time

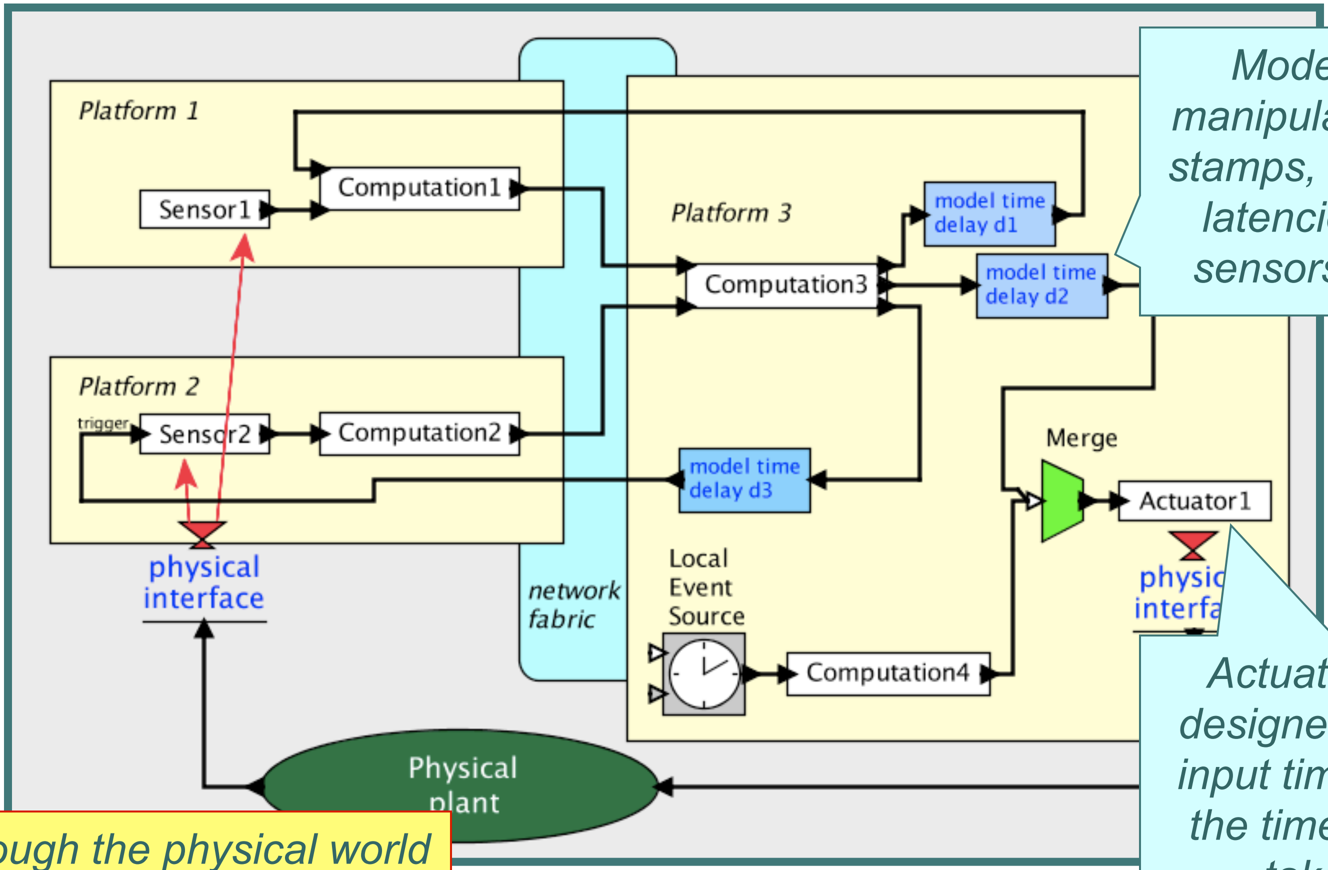
Output time stamps are ≤ real time

Clock synchronization gives global meaning to time stamps

Messages are processed in time-stamp order.

Ptides: Fourth step:
Specify latencies in the model

Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.



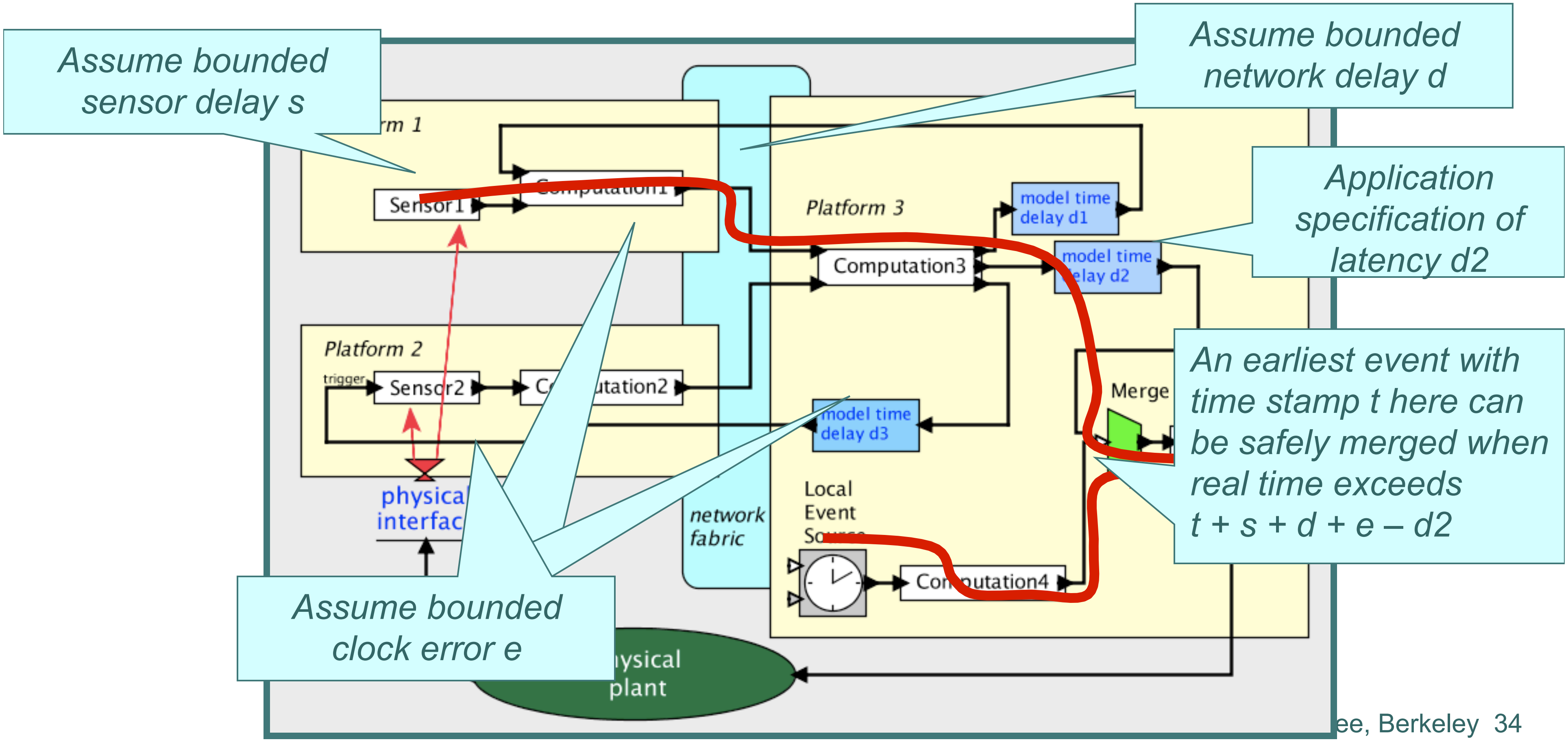
Model includes manipulations of time stamps, which control latencies between sensors and actors

Actuators may be designed to interpret input time stamps as the time at which to take action.

Feedback through the physical world

Ptides: Fifth step
Safe-to-process analysis (ensures determinacy)

Safe-to-process analysis guarantees that the generated code obeys time-stamp semantics (events are processed in time-stamp order), given some assumptions.



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

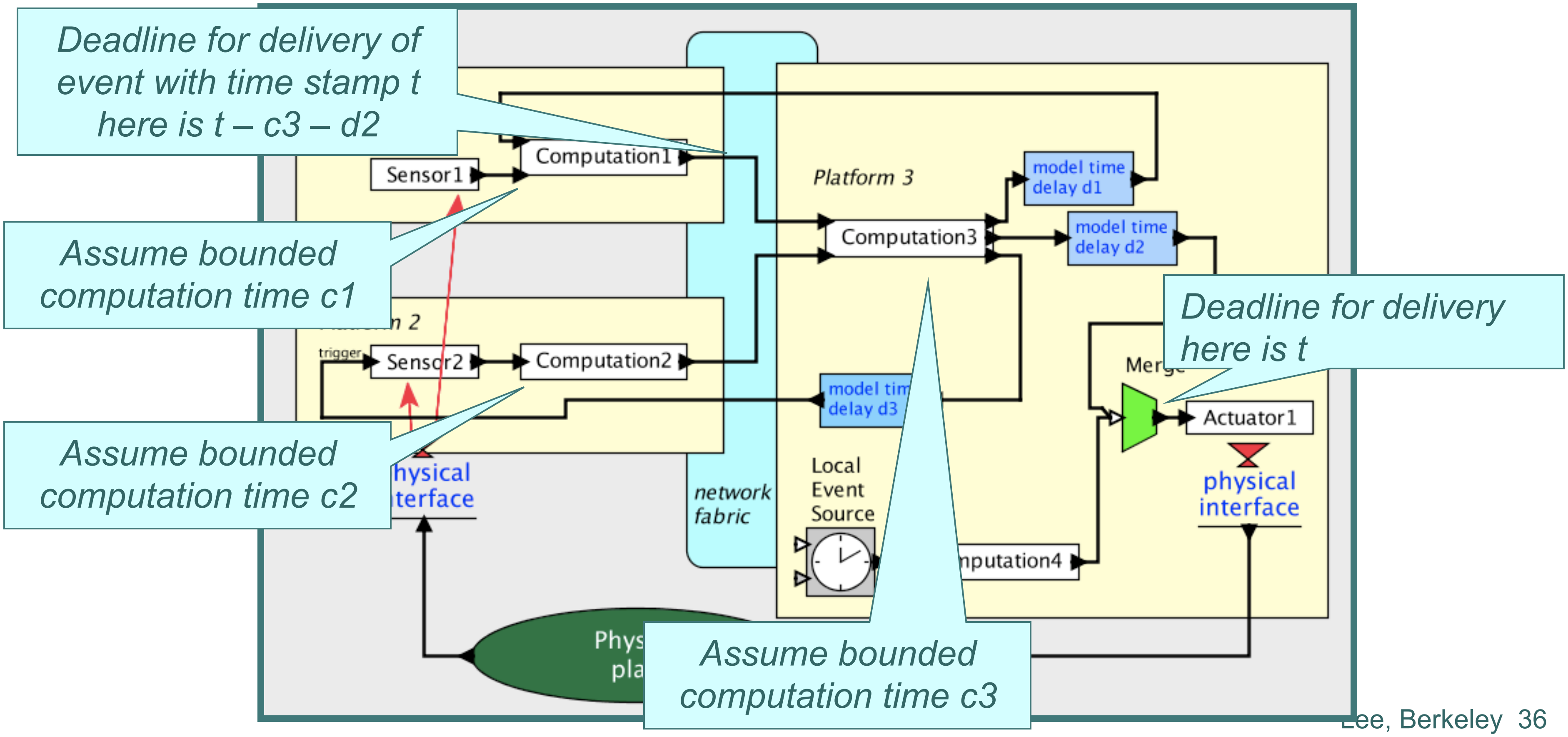
Bounded network delay
is enabled by time synchronization…

- **ARINC 429**
  - Local area network used in avionics systems.
- **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, TU 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

Ptides Schedulability Analysis

Determine *whether* deadlines can be met

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

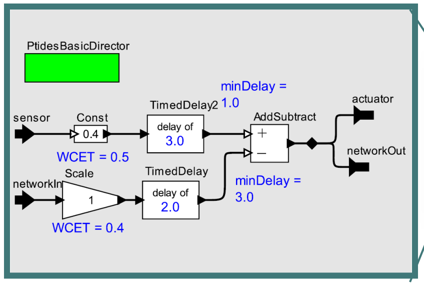
Assume bounded computation time c1

Assume bounded computation time c2

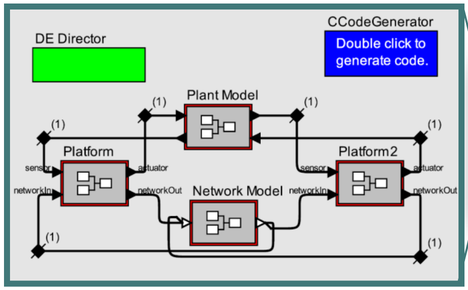
Assume bounded computation time c3

Deadline for delivery here is t

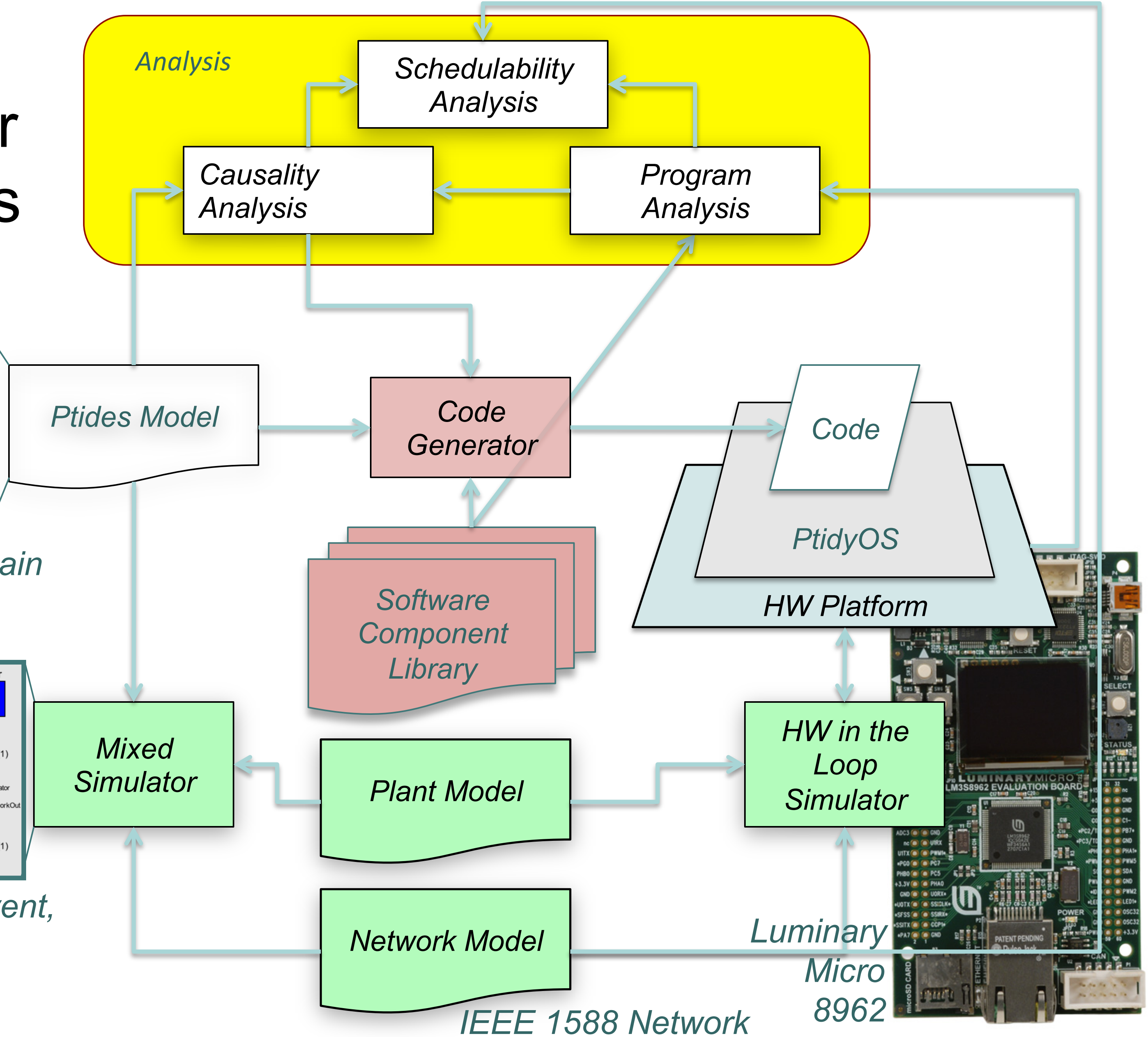
Workflow Structure for Experiments



Ptolemy II Ptides domain



Ptolemy II Discrete-event, Continuous, and Wireless domains



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

Luminary Micro 8962

IEEE 1588 Network time protocol

Designing & Evaluating PTIDES-based Systems

To meet real-time constraints,
the implementation platform matters.

Conventional approach: Specify functionality and implementation. Then measure temporal properties.

Our approach: Specify temporal requirements. Then verify that they are met by a candidate implementation.

Topics for further discussion

- How to represent time?
  - Need superdense time for a clean semantics of simultaneity.
- How to advance time?
  - Need multiform time to model inhomogeneity and imperfect sync.
- How to determine the required accuracy of time sync?
  - PTIDES offers a tradeoff between latency and time sync accuracy.
- How to handle faults?
  - PTIDES can detect violations of assumptions (bounded clock error, bounded network latency, and bounded sensor delay).
- Security?
  - Does time synchronization create a point of vulnerability?

Overview References:

- *Lee.* ***Computing needs time****. CACM, 52(5):70–79, 2009*
- *Eidson et. al, Distributed Real-Time Software for Cyber-Physical Systems, Proc. of the IEEE January, 2012.*

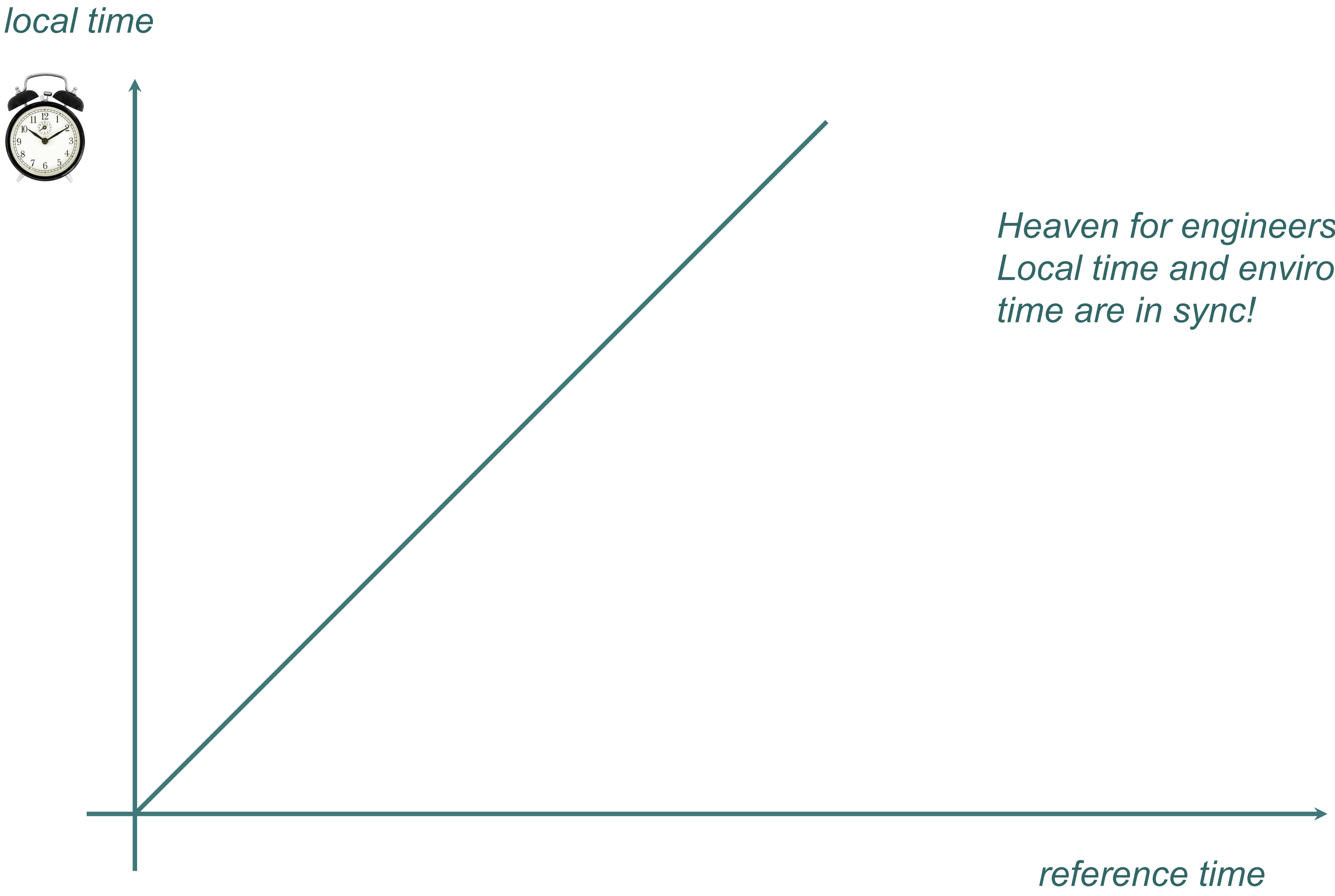
Conclusions

Today, timing emerges from *realizations* of systems.

Tomorrow, timing behavior will be a *semantic* property of networks, programs, and models.

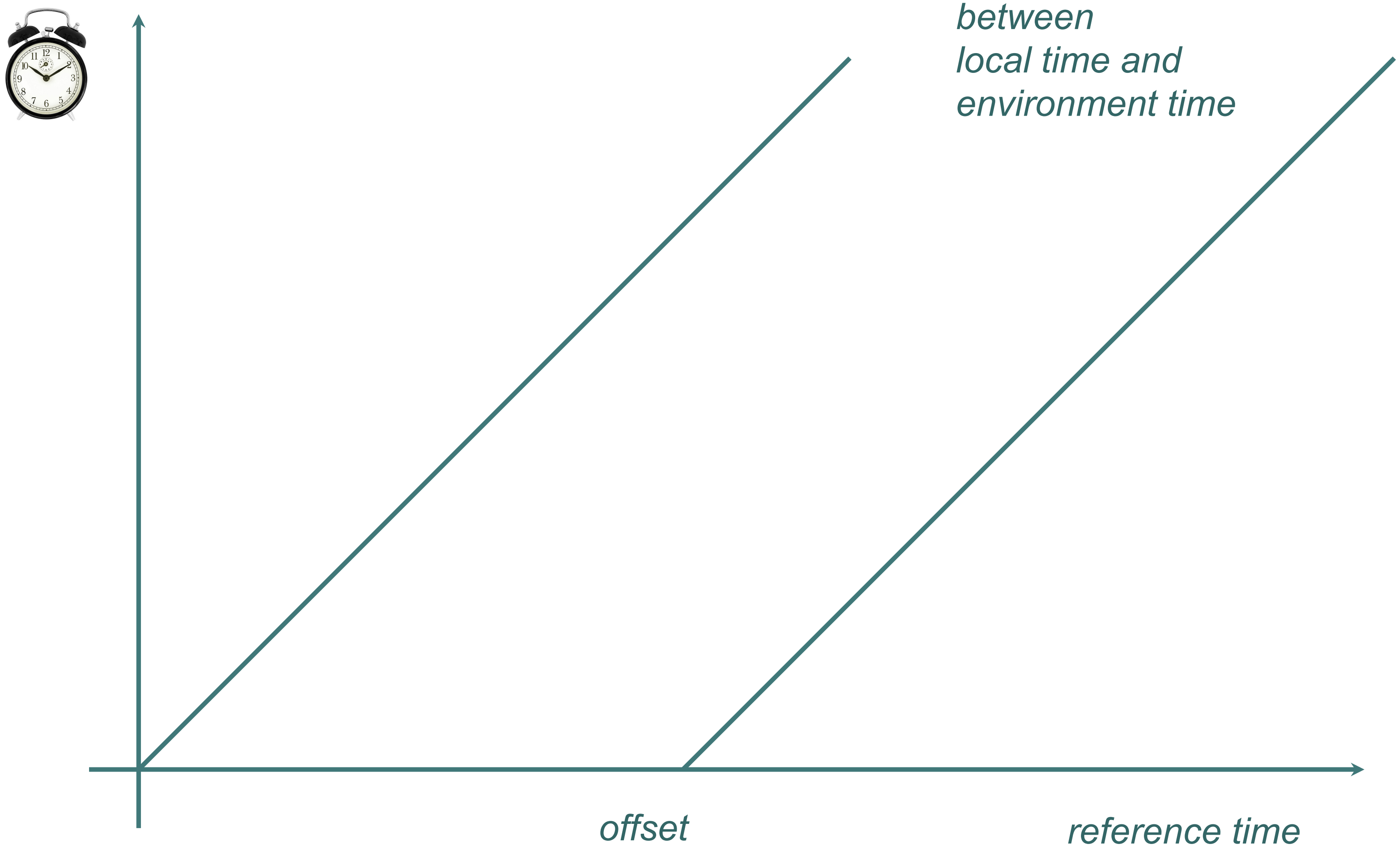
Raffaello Sanzio da Urbino – The Athens School

Multiform Time



Heaven for engineers.
Local time and environment time are in sync!

Multiform Time in the Real World



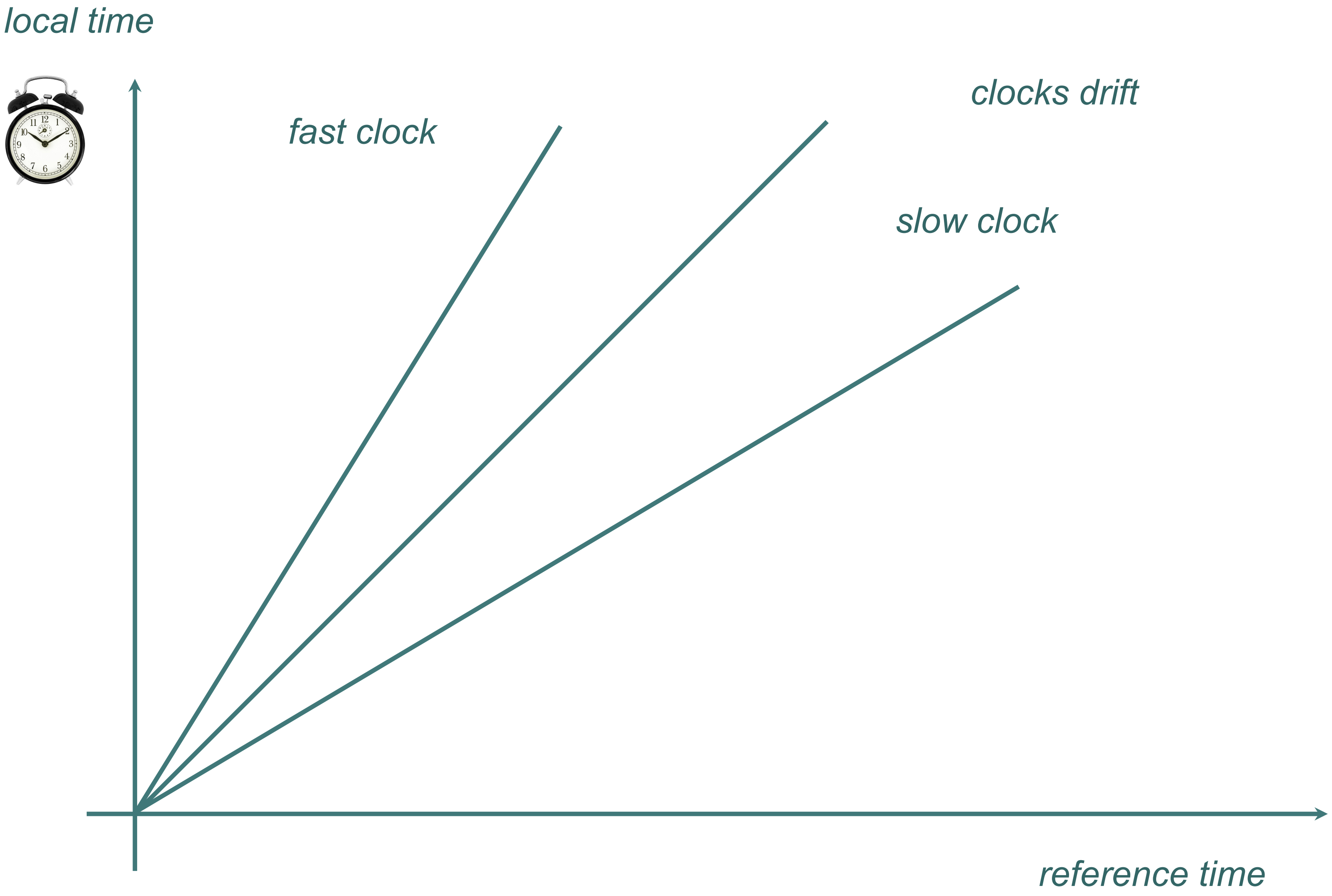
local time

There is an offset between
local time and
environment time

offset

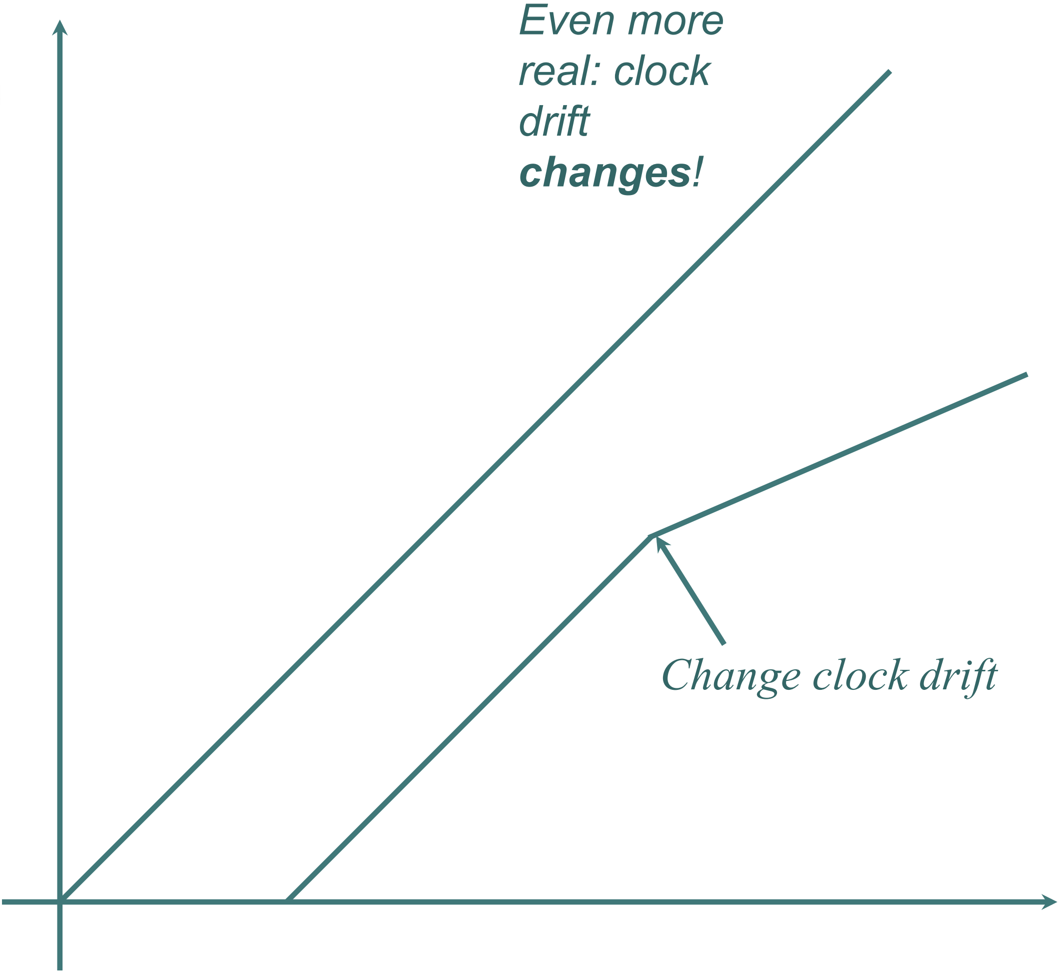
reference time

Multiform Time in the Real World



Multiform Time in the Real World

local time

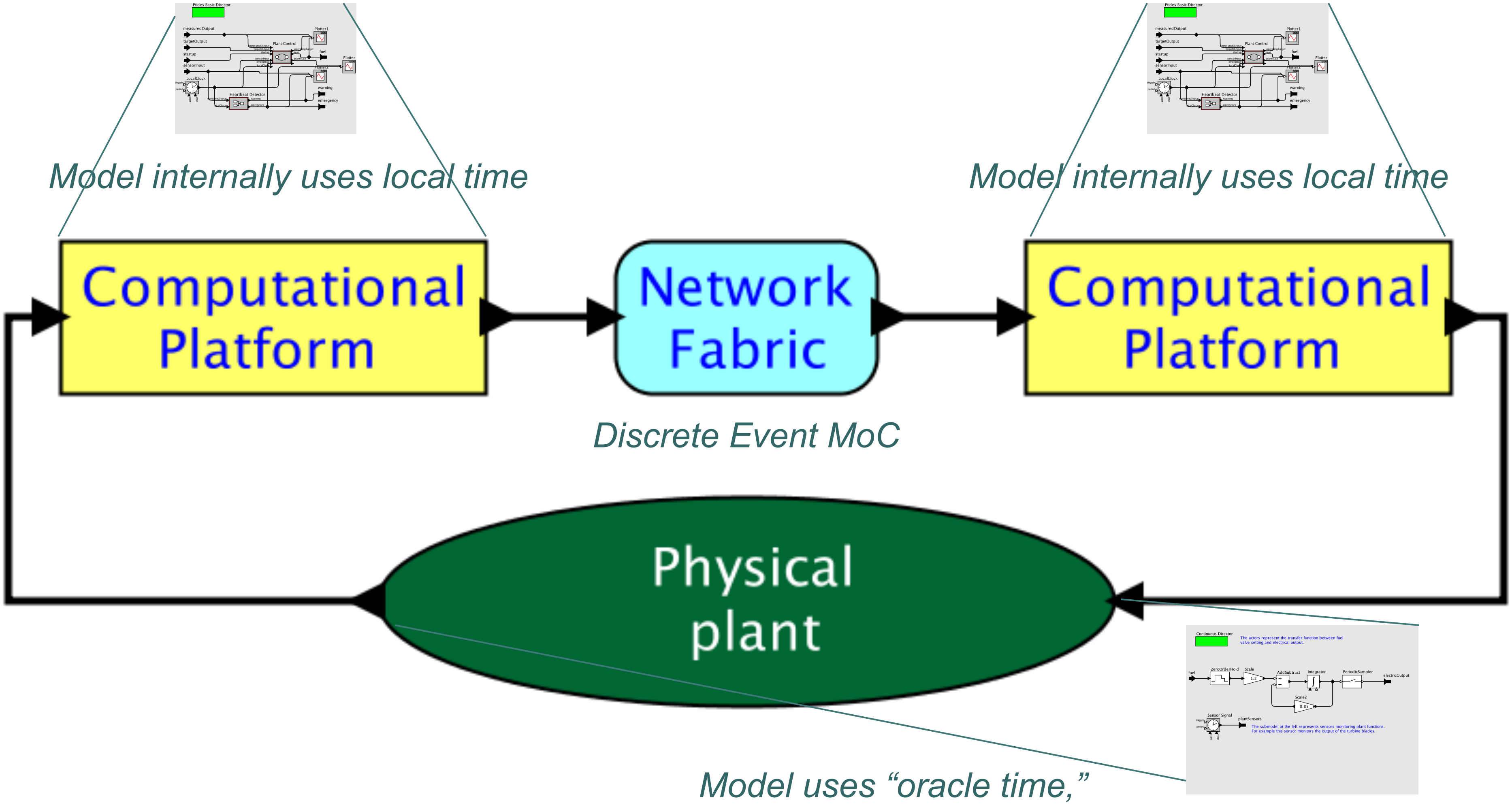


Even more real: clock drift ***changes***!

Change clock drift

reference time

Ptolemy II model: Local time within a hierarchy advances at different rates.



Model internally uses local time

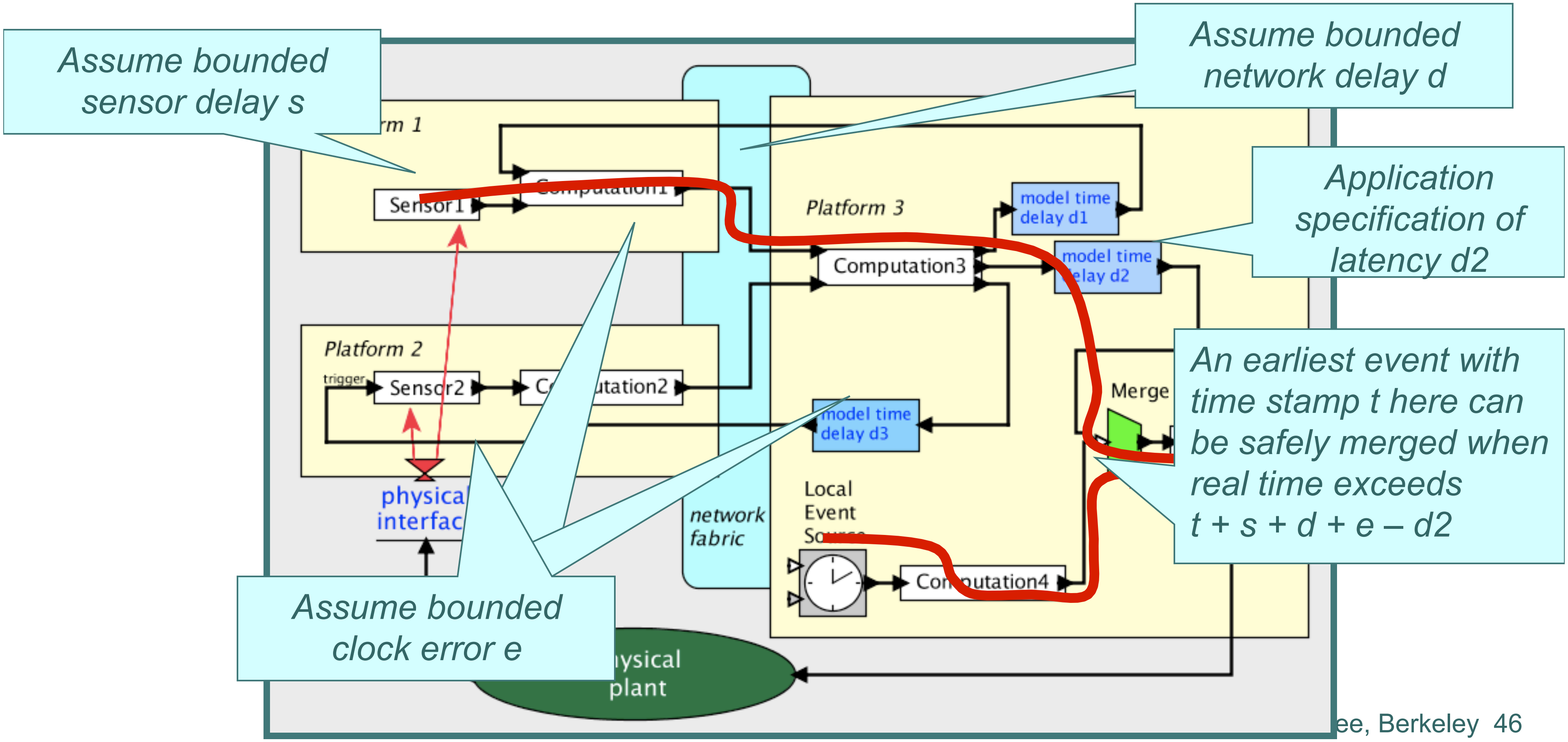
Model internally uses local time

Discrete Event MoC

Model uses "oracle time," which becomes "environment time" for the subsystems.

Ptides: Fifth step
Safe-to-process analysis (ensures determinacy)

Safe-to-process analysis guarantees that the generated code obeys time-stamp semantics (events are processed in time-stamp order), given some assumptions.



Assume bounded sensor delay s

Assume bounded network delay d

Application specification of latency d2

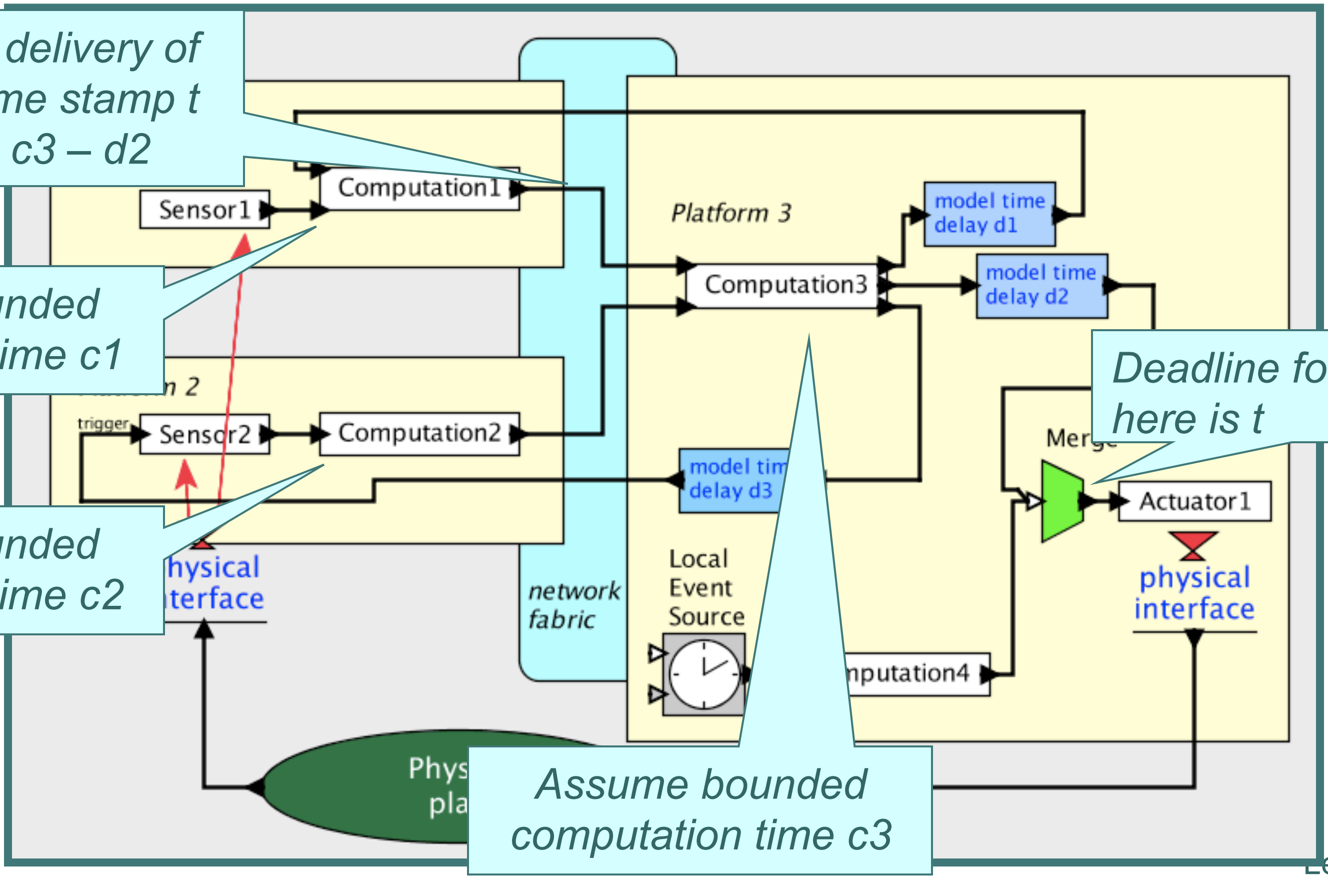
An earliest event with time stamp t here can be safely merged when real time exceeds t + s + d + e – d2

Assume bounded clock error e

Ptides Schedulability Analysis

Determine whether *deadlines* can be met

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

PtidyOS: A lightweight microkernel supporting Ptides semantics

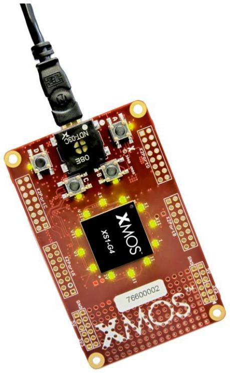
An interesting property of PtidyOS is that despite being highly concurrent, preemptive, and EDF-based, it does not require threads.

A single stack is sufficient!

PtidyOS runs on

- Arm (Luminary Micro)
- Renesas
- XMOS

Occupies about 16 kbytes of memory.



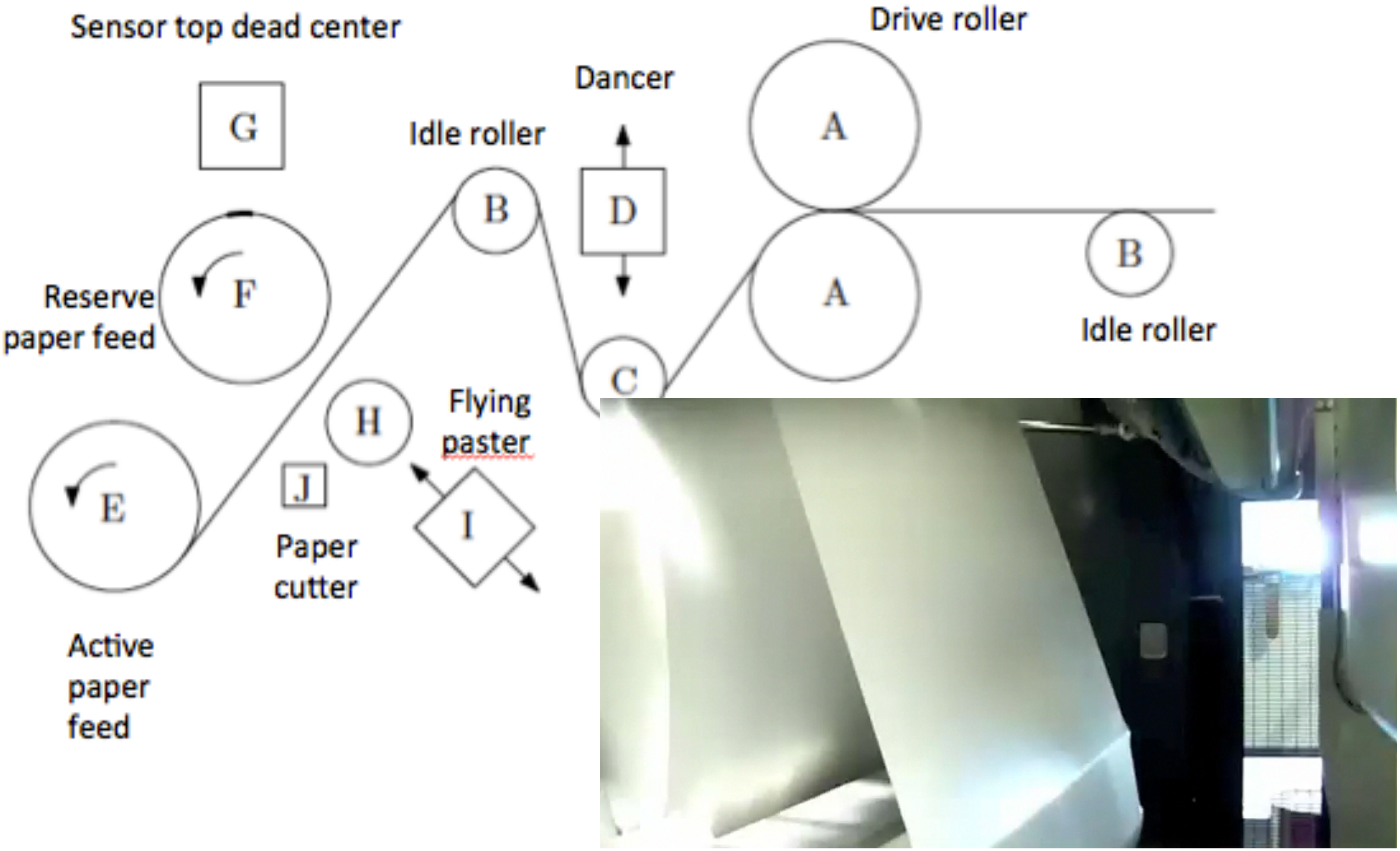
XMOS development board with 4 XCores.

Renesas 7216 Demonstration Kit

Luminary Micro 8962

The name "PtidyOS" is a bow to TinyOS, which is a similar style of runtime kernel.

Example – Flying Paster



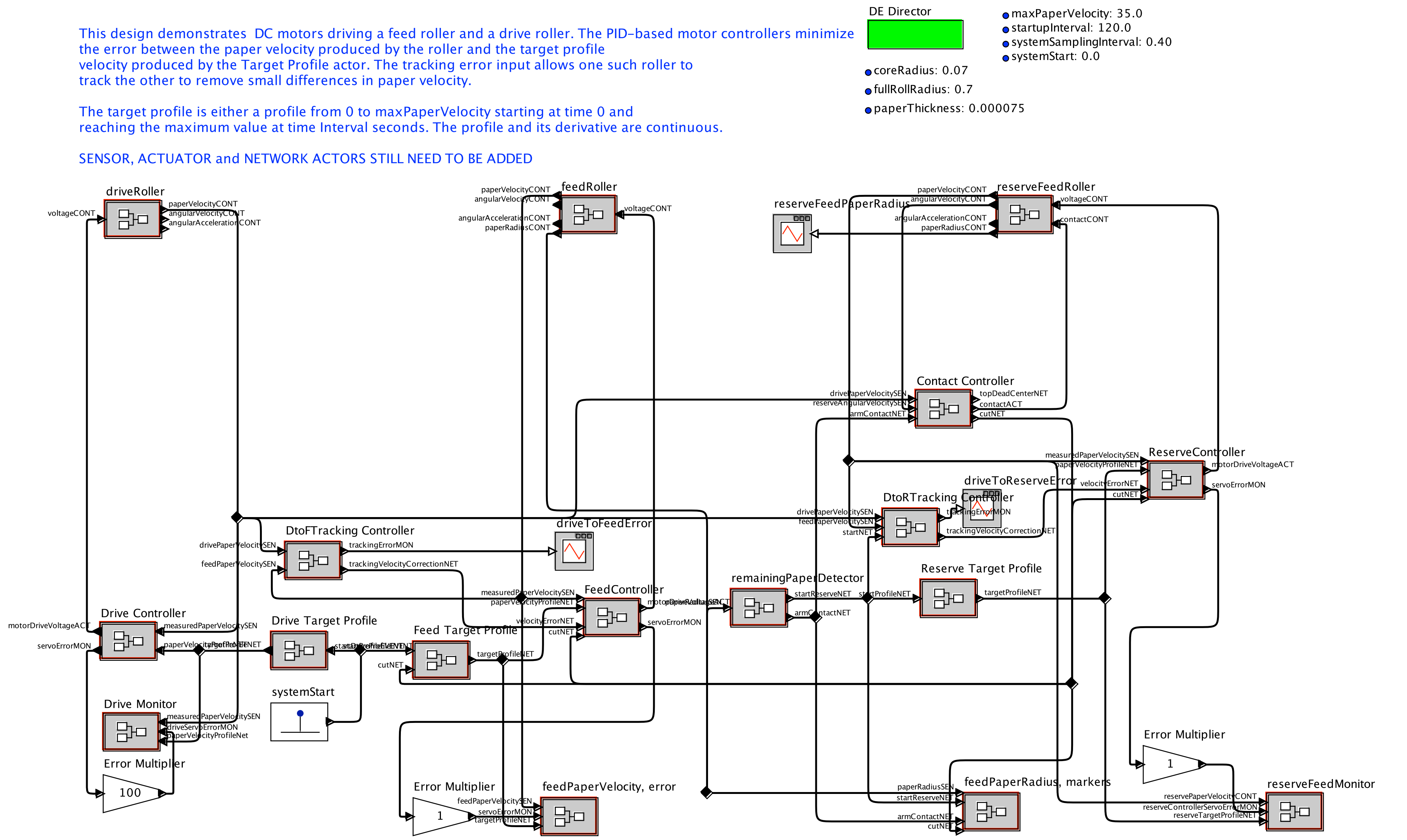
Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.htı

Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.html

Flying Paster

Model by Patricia Derler

Printing Press – Model in Ptolemy II



This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

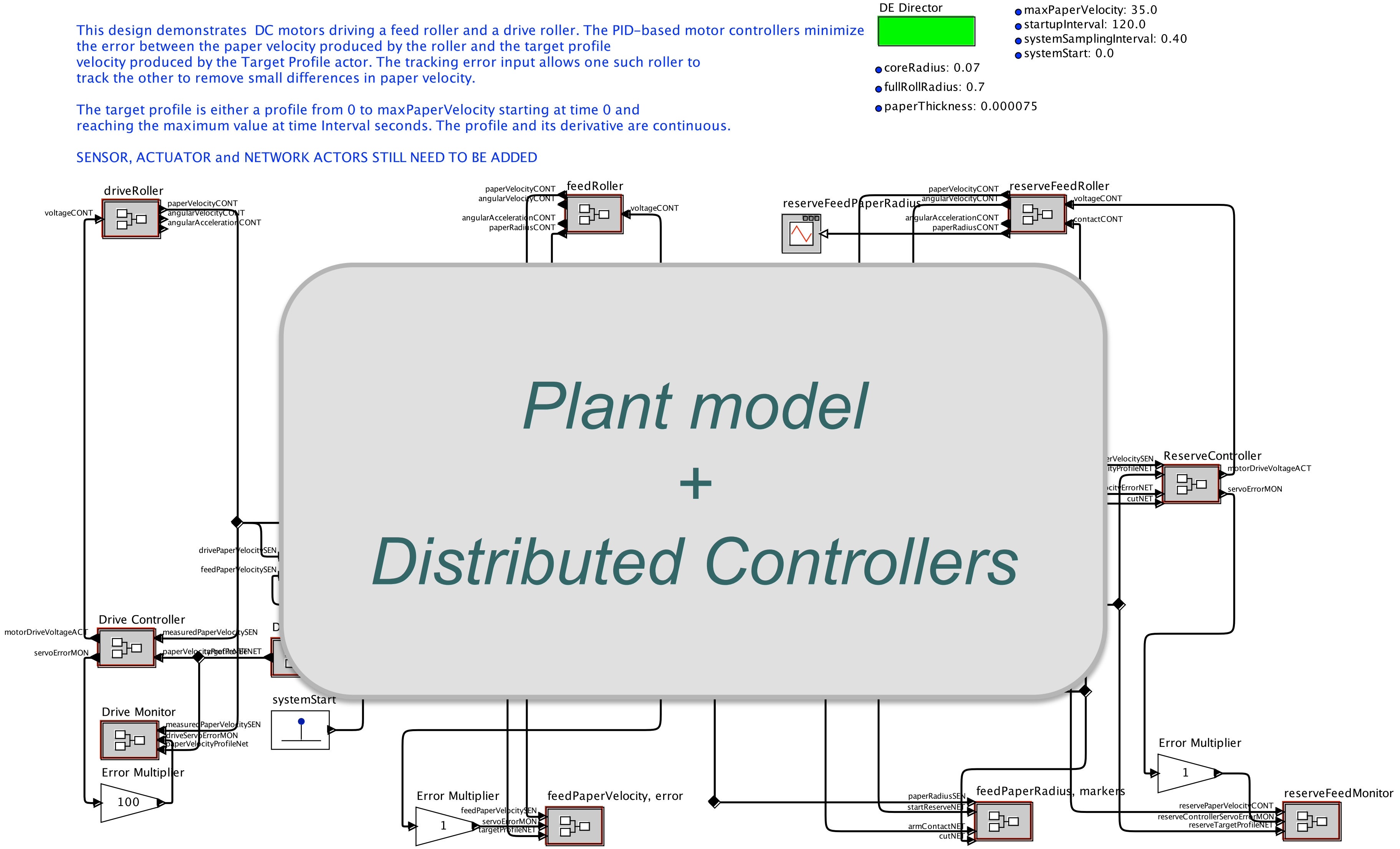
SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

DE Director

- maxPaperVelocity: 35.0
- startupInterval: 120.0
- systemSamplingInterval: 0.40
- systemStart: 0.0
- coreRadius: 0.07
- fullRollRadius: 0.7
- paperThickness: 0.000075

Model by Patricia Derler

Printing Press – Model in Ptolemy II



This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

DE Director

- maxPaperVelocity: 35.0
- startupInterval: 120.0
- systemSamplingInterval: 0.40
- systemStart: 0.0
- coreRadius: 0.07
- fullRollRadius: 0.7
- paperThickness: 0.000075

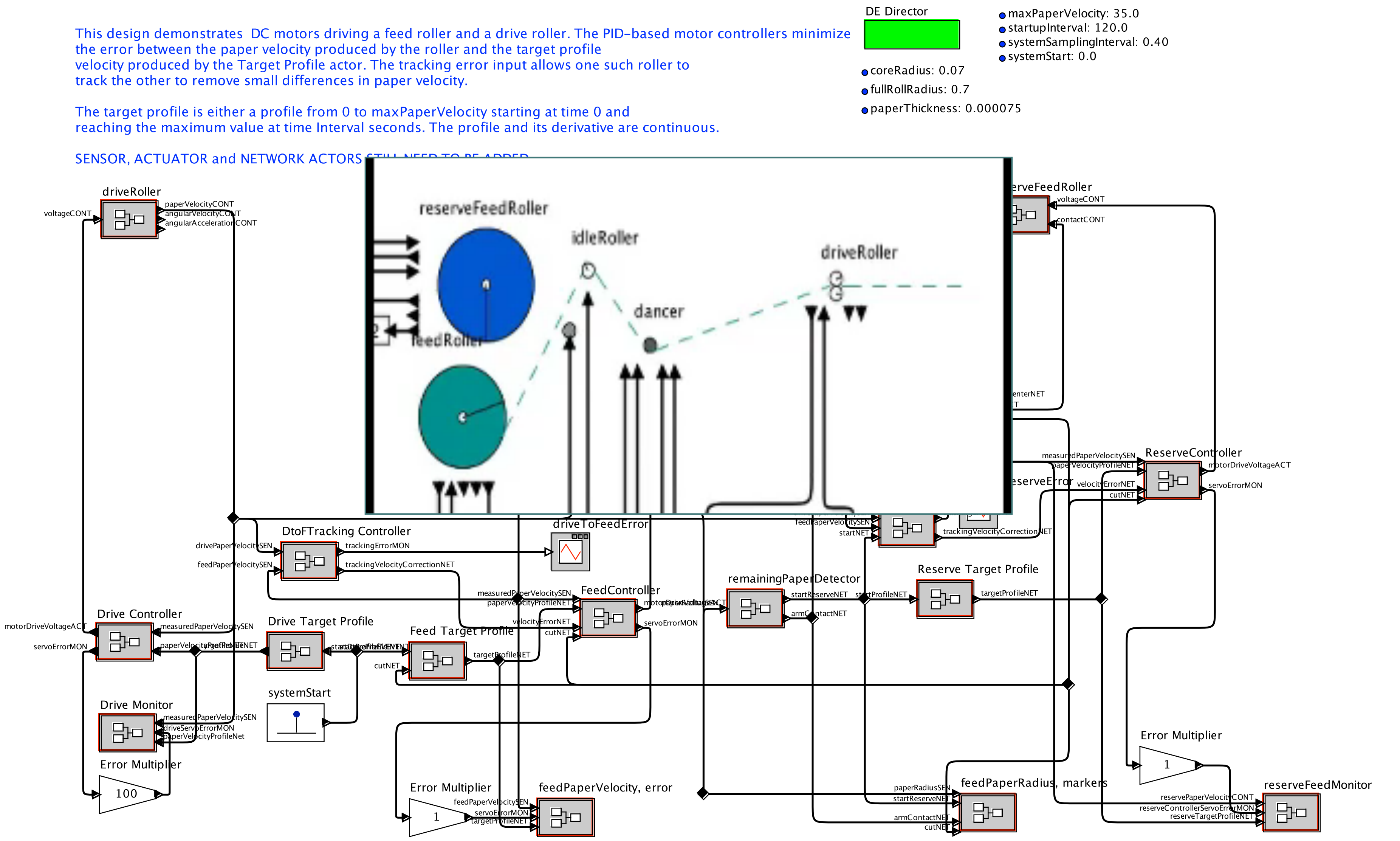
Plant model

+

Distributed Controllers

Model by Patricia Derler

Printing Press – Model in Ptolemy II



This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

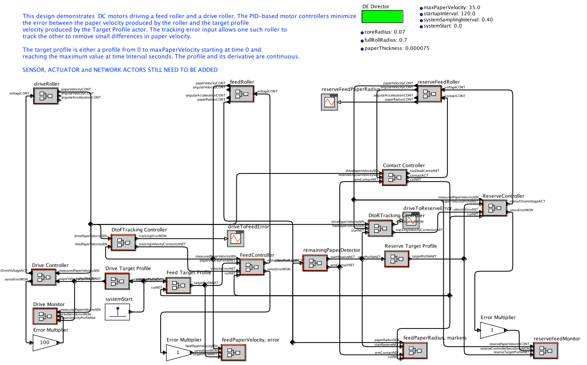
SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

DE Director

- maxPaperVelocity: 35.0
- startupInterval: 120.0
- systemSamplingInterval: 0.40
- systemStart: 0.0
- coreRadius: 0.07
- fullRollRadius: 0.7
- paperThickness: 0.000075

Determinate timing at sensors and actuators

Platform independent model of functional and timing behavior



This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

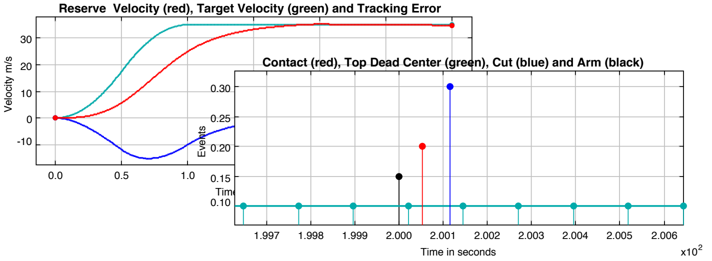
The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

- maxPaperVelocity: 35.0
- startupInterval: 120.0
- systemSamplingInterval: 0.40
- systemStart: 0.0
- coreRadius: 0.07
- fullRollRadius: 0.7
- paperThickness: 0.000075

Code Generation to multiple target platforms

Simulation

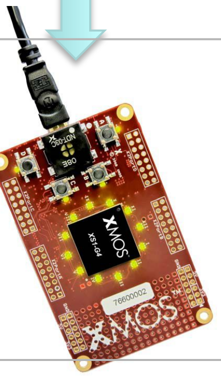


```
while(1) {
    // Check channels for any event queue requests.
    checkChannels(schedulerChannels, numChan);
    // Start at front of event queue.
    event = eventQueueHead;
    // No reason to get timestamp if no events in event queue.
    // But what about update?
    //if(event == NULL) {
    //    continue;
    //}
    // Get current platform time for safe-to-process.
    getTimestamp(&platformTime, platformClockChannel);
    // Search queue for highest priority event which is safe to proces:
    while(event != NULL) {
        // Check channels for any event requests.
        // If request results in event queue modification, need to
        // restart search.
        //storeTestTime(7);
        if(checkChannels(schedulerChannels, numChan) == RESTART) {
            break;
        }
        //storeTestTime(8);
        // If the event has a higher priority than the currently
        // processing event, then safe-to-process can be checked.
        if((!currProcessing() ||
        compareEvents(event, peekProcessing()) < 0)) {
            if(safeToProcess(event, &platformTime)) {
                //storeTestTime(9);
                // Store event as being currently processed.
                pushProcessing(event);
                // Process the event(s).
                processEvent(event, eventProcessingChannel);
                // No events past this will have a higher priority, so
                // restart search.
                //storeTestTime(10);
                break;
            }
            // Check next event in event queue.
            event = event->nextEvent;
            // No events past this will have a higher priority, so
            // restart search.
        } else {
            break;
        }
    }
}
```

```
edef struct {
 int32 secs;
 int32 nsecs;
ignedTime;
edef struct {
 Time timestamp;
 uint16 microstep;
ag;
t32 countUntilTimestamp(const Time *timestamp,
eaming chanend platformClockChannel);
d getTimestamp(Time *timestamp,
eaming chanend platformClockChannel);
d getTimestampFromCount(Time *timestamp,
eaming chanend platformClockChannel, uint32 count);
- Scheduler.h --//
edef struct Event {
 union {
     char boolean_Value;
     char charValue;
     int16 shortValue;
     uint16 ushortValue;
     int32 intValue;
     uint32 uintValue;
     float floatValue;
     double double_Value;
     struct {
         uint32 u;
         uint32 l;
     } dataValue;
 } Val;
 Tag tag;
 uint32 depth;
 Time deadline;
 SignedTime offset;
 void (*fire)();
 struct Event** sinkEvent;
 struct Event* nextEvent;
 struct Event* prevEvent;
vent;
fine NO_RESTART 0
fine RESTART 1
TODO: figure out max
Maximum number of events. */
fine MAX_EVENTS 20
Maximum number of processing events. */
fine MAX_PROCESSING 10
m {
```

Same I/O behavior w.r.t. value and timing

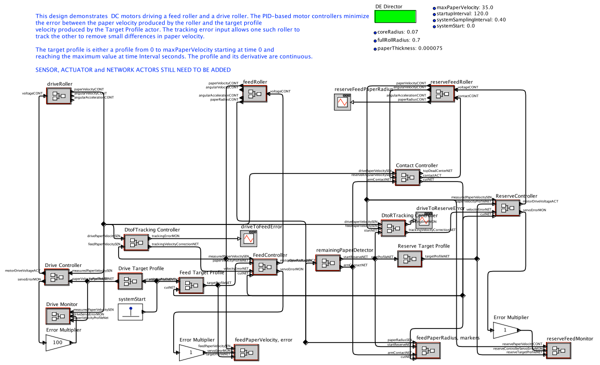
e.g.: XMOS development board with 4 XCores.



e.g.: Renesas 7216 Demonstration Kit

Determinate timing at sensors and actuators

Platform independent model of functional and timing behavior



This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

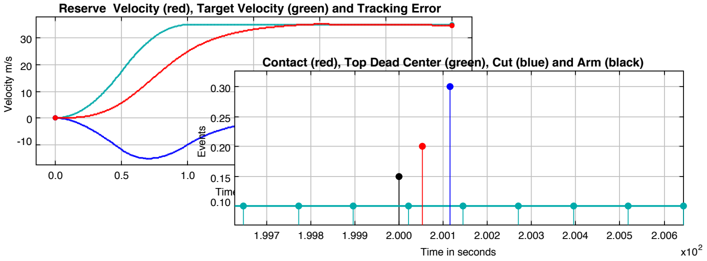
SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

DE Director

- maxPaperVelocity: 35.0
- startupInterval: 120.0
- systemSamplingInterval: 0.40
- systemStart: 0.0
- coreRadius: 0.07
- fullRollRadius: 0.7
- paperThickness: 0.000075

Code Generation to multiple target platforms

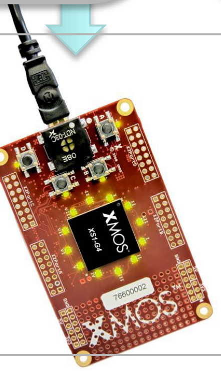
Simulation

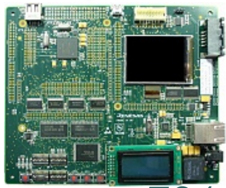


XMOS
Predictable timing
Multiple cores
No analog I/O
No FPU
No hardware clock

Renesas
PHY chip for accurate timestamping of inputs,
Analog I/O

e.g.: XMOS development board with 4 XCores.



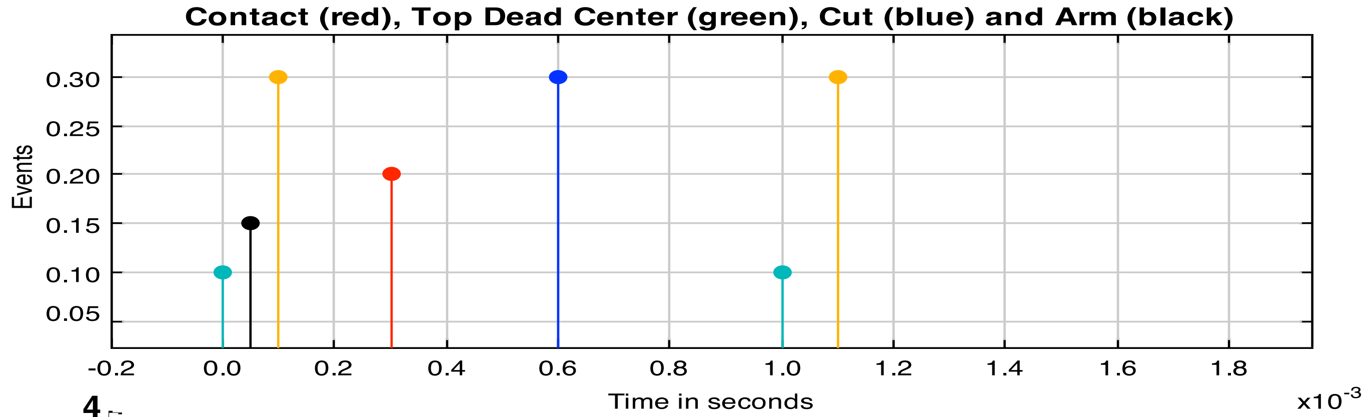


e.g.: Renesas 7216 Demonstration Kit

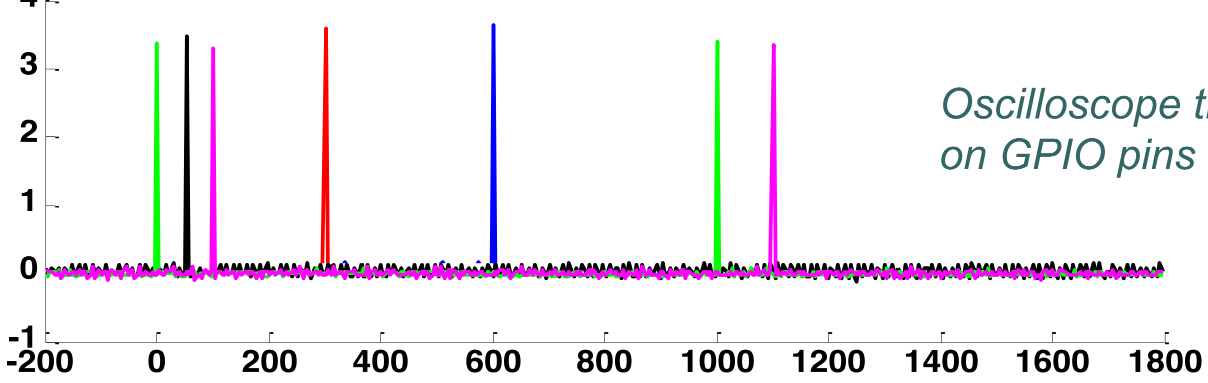
Same I/O behavior w.r.t. value and timing

Simple test case validates platform-independent timing.

Simulation

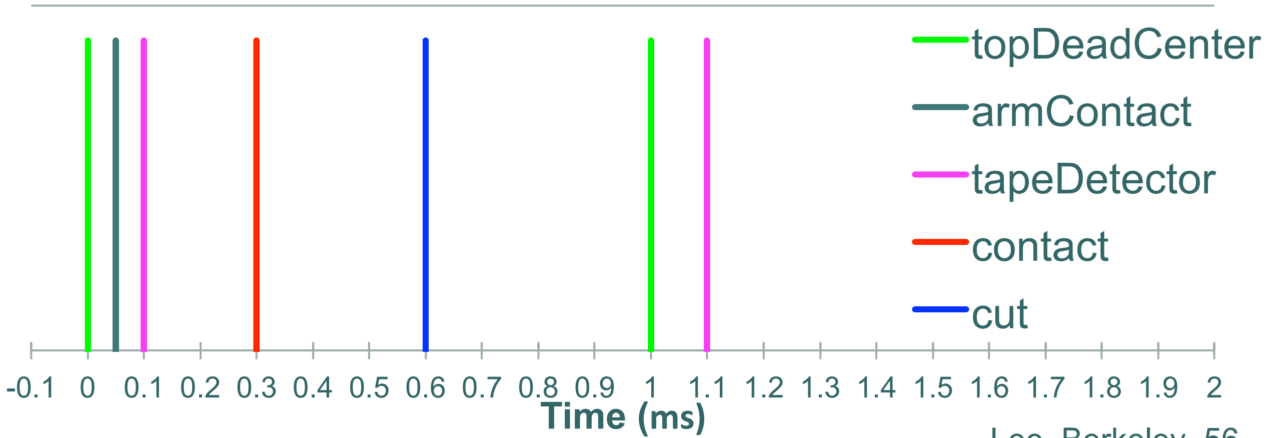


Renesas



Oscilloscope traces on GPIO pins

XMOS

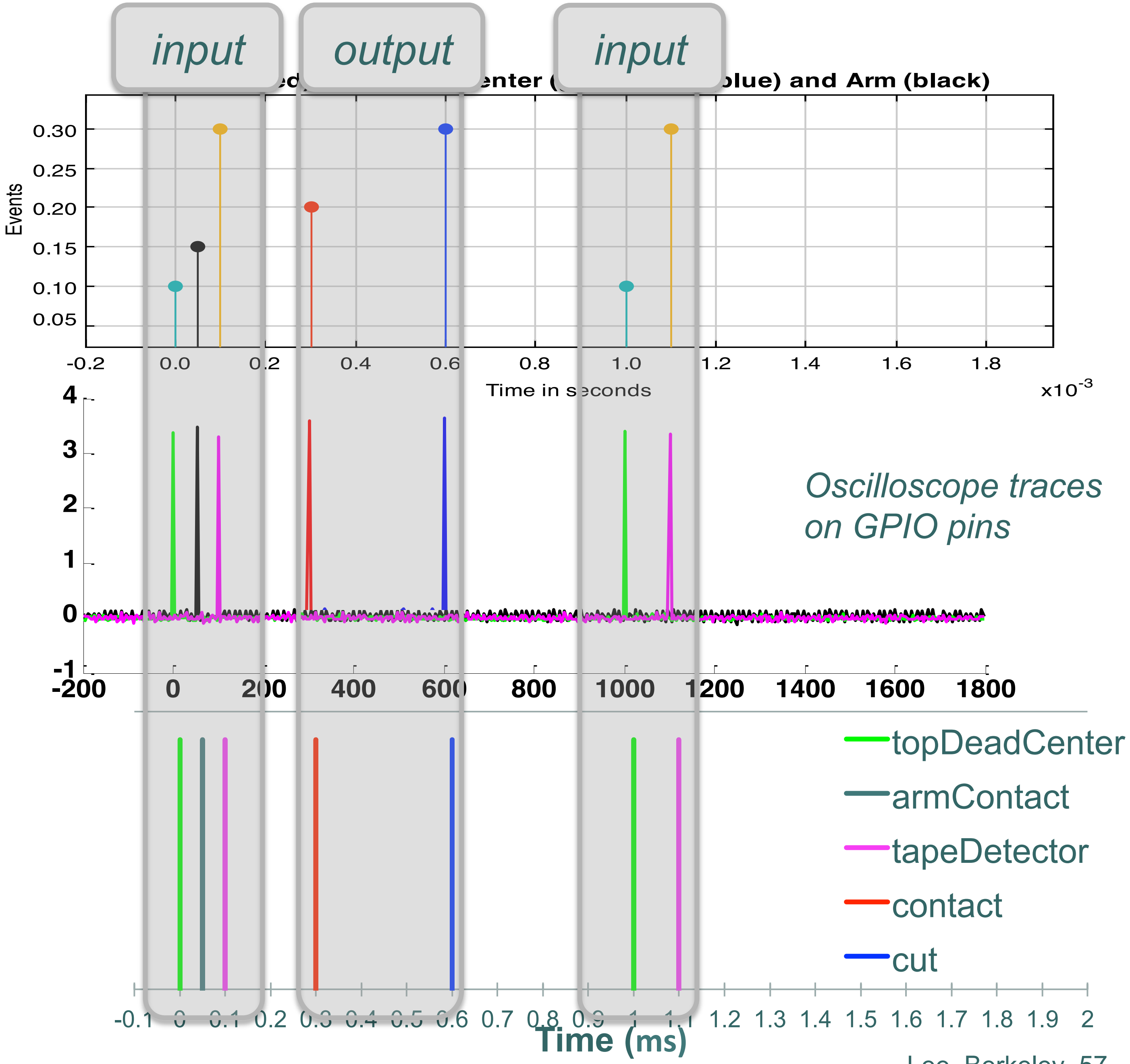


Renesas vs. XMOS: I/O timing

Simulation

Renesas

XMOS



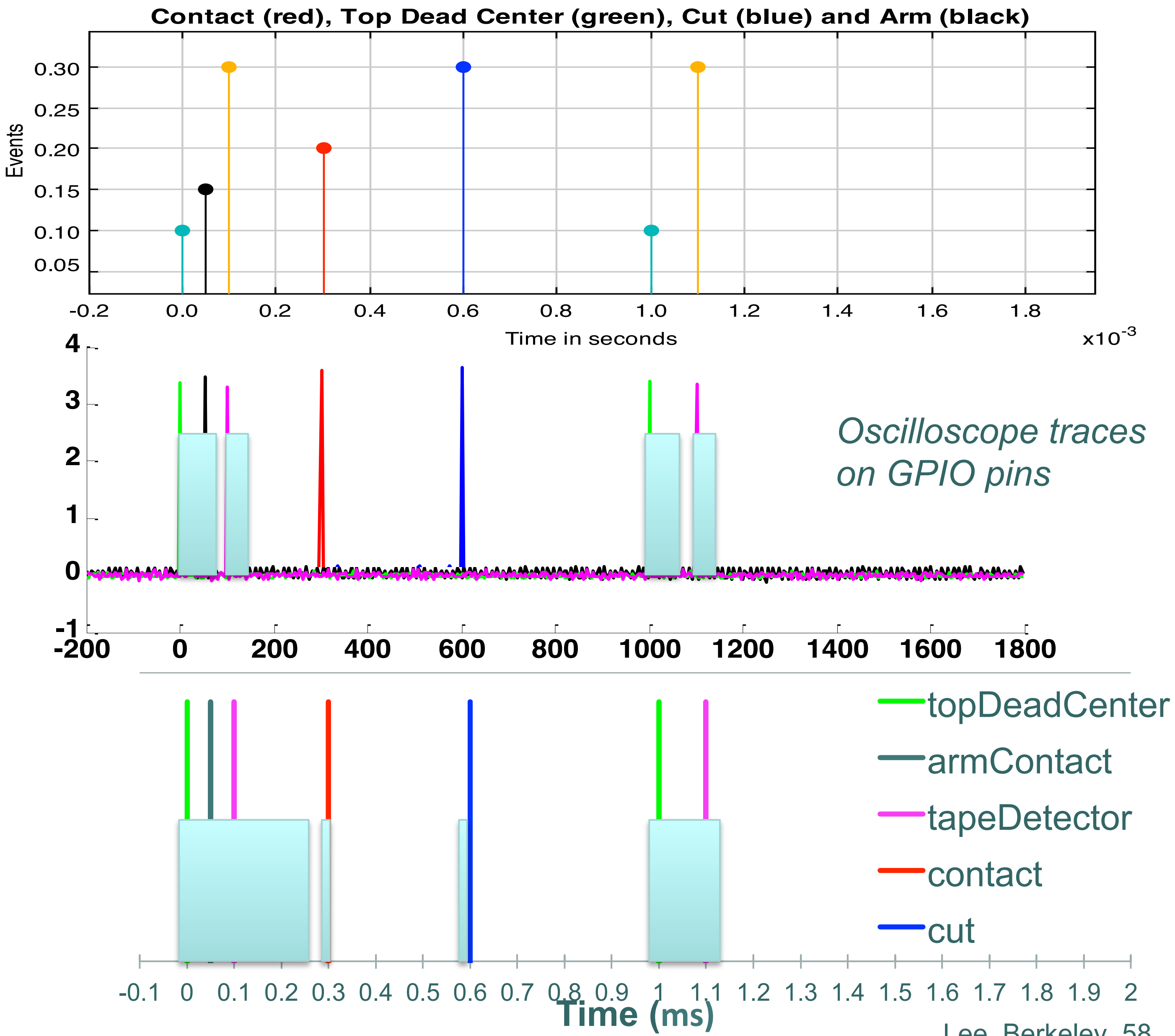
Oscilloscope traces on GPIO pins

Renesas vs. XMOS: Busy vs. Idle Time

Simulation

Renesas

XMOS



Oscilloscope traces on GPIO pins

Execution-time analysis, by itself, does not solve the problem!

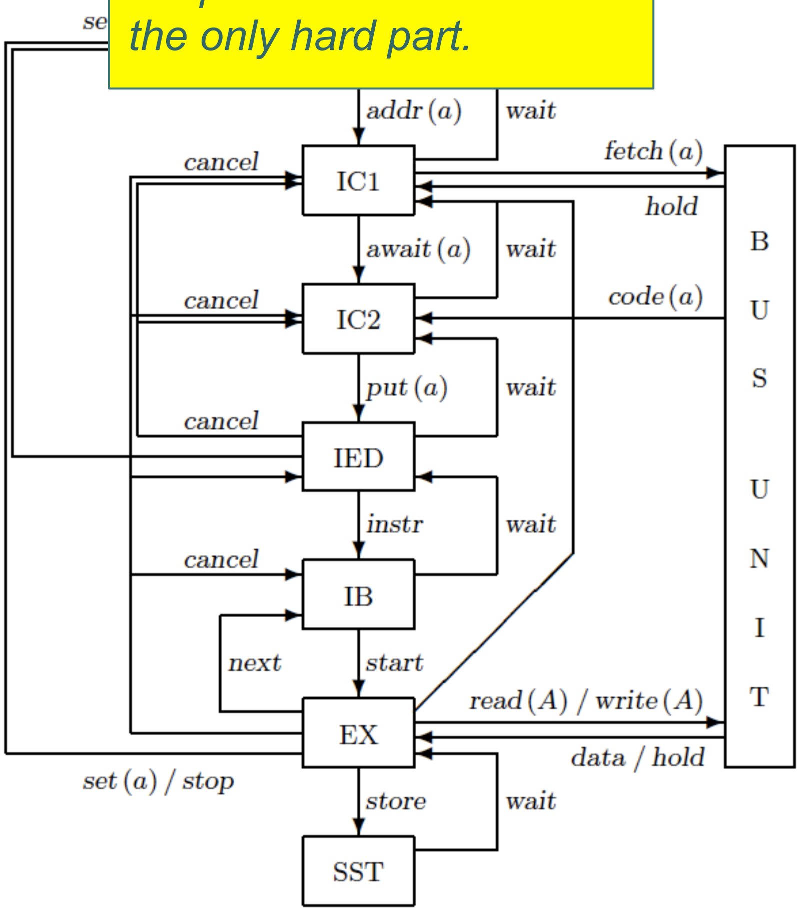
Our first goal is to reduce the problem so that this is the only hard part.

Analyzing software for timing behavior requires:

- Paths through the program (undecidable)
- Detailed model of microarchitecture
- Detailed model of the memory system
- Complete knowledge of execution context
- Many constraints on preemption/concurrency
- Lots of time and effort

And the result is valid only for that exact hardware and software!

Fundamentally, the ISA of the processor has failed to provide an adequate abstraction.



Wilhelm, et al. (2008). "The worst-case execution-time problem - overview of methods and survey of tools." ACM TECS 7(3): p1-53.

Dual Approach

- Rethink the ISA
  - Timing has to be a *correctness* property not a *performance* property.

- Implementation has to allow for multiple realizations and efficient realizations of the ISA
  - Repeatable execution times
  - Repeatable memory access times

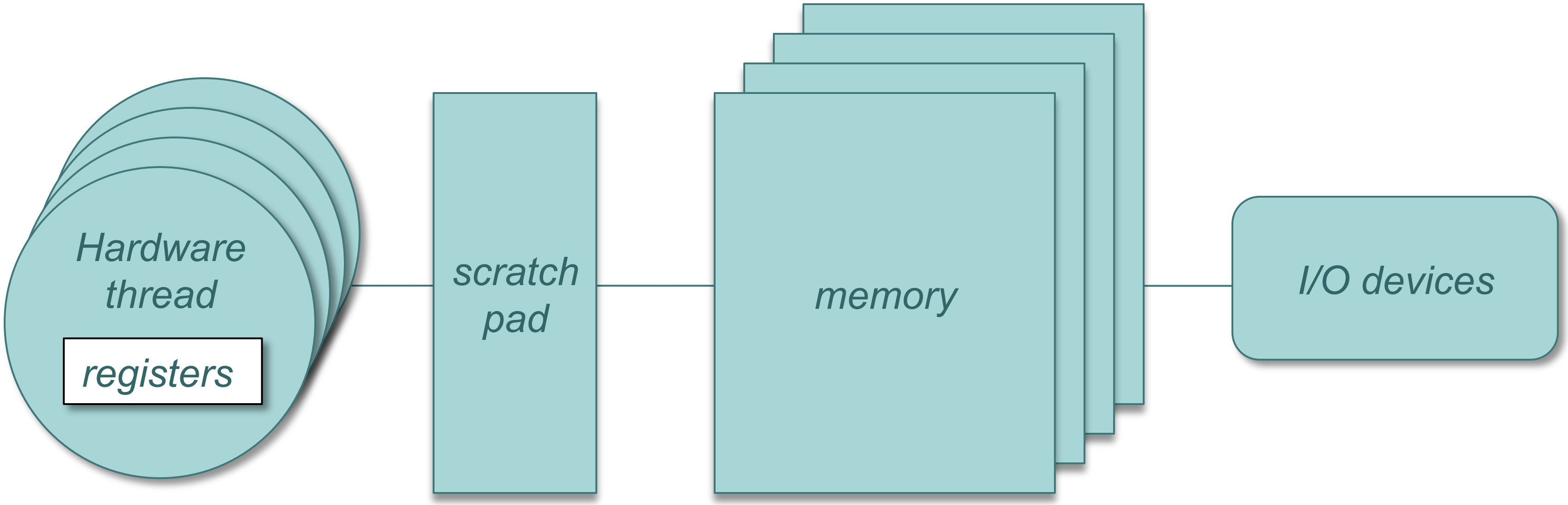
To deliver repeatable timing, we have to rethink the microarchitecture

Challenges:

- Pipelining
- Memory hierarchy
- I/O (DMA, interrupts)
- Power management (clock and voltage scaling)
- On-chip communication
- Resource sharing (e.g. in multicore)

Our Current PRET Architecture

PTArm, a soft core on a Xilinx Virtex 5 FPGA



Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

Performance Cost?

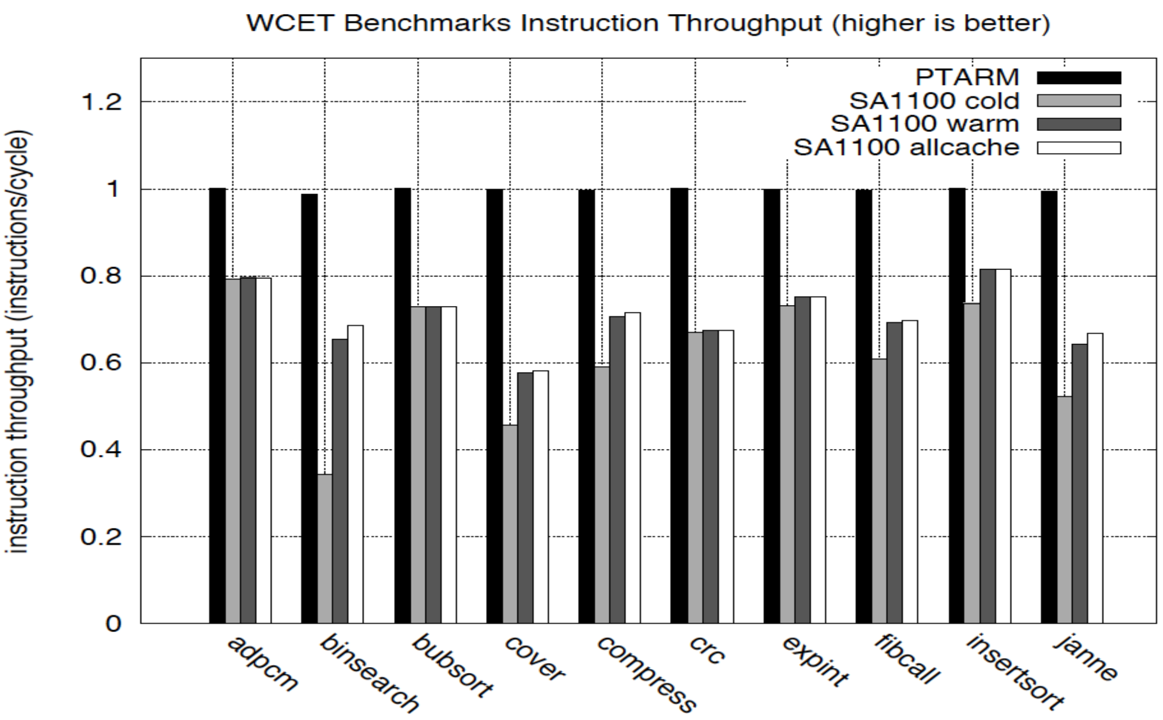
No!

Comparing PTARM against SimIT-ARM simulator (StrongARM 1100) [Qin & Malik] over Malardalen WCET benchmarks [Gustafsson…].

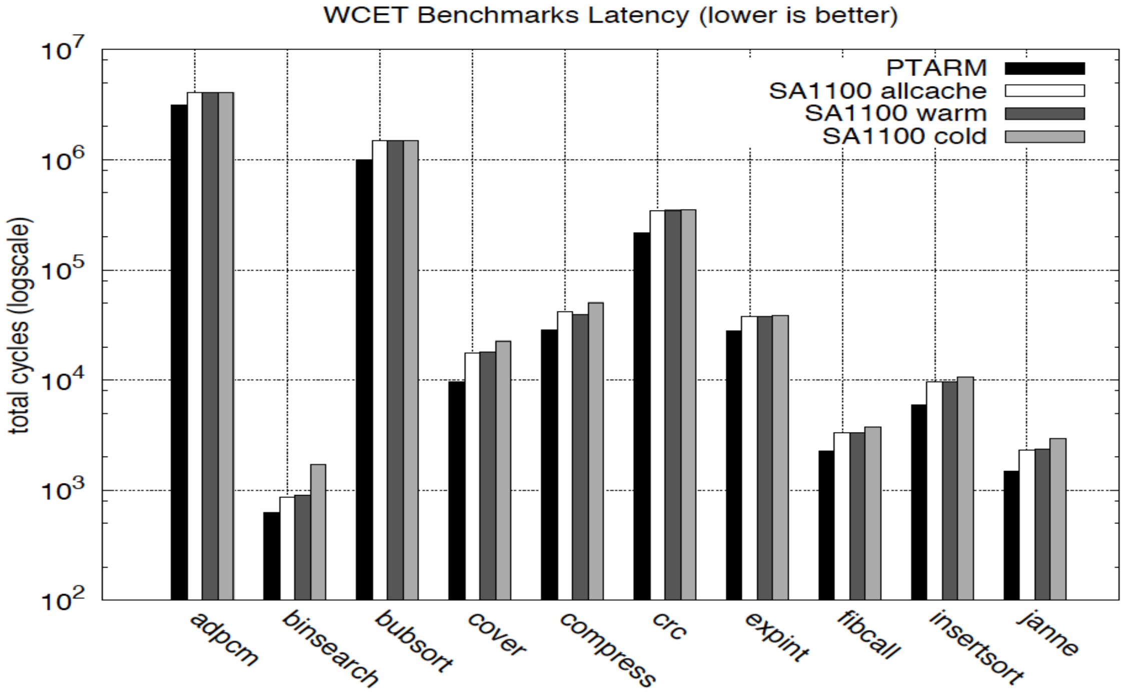
Given enough concurrency, the PTARM beats the StrongARM on every benchmark!

Moreover, our simpler pipeline can probably be clocked faster.

[Isaac Liu, PhD Thesis, May, 2012]



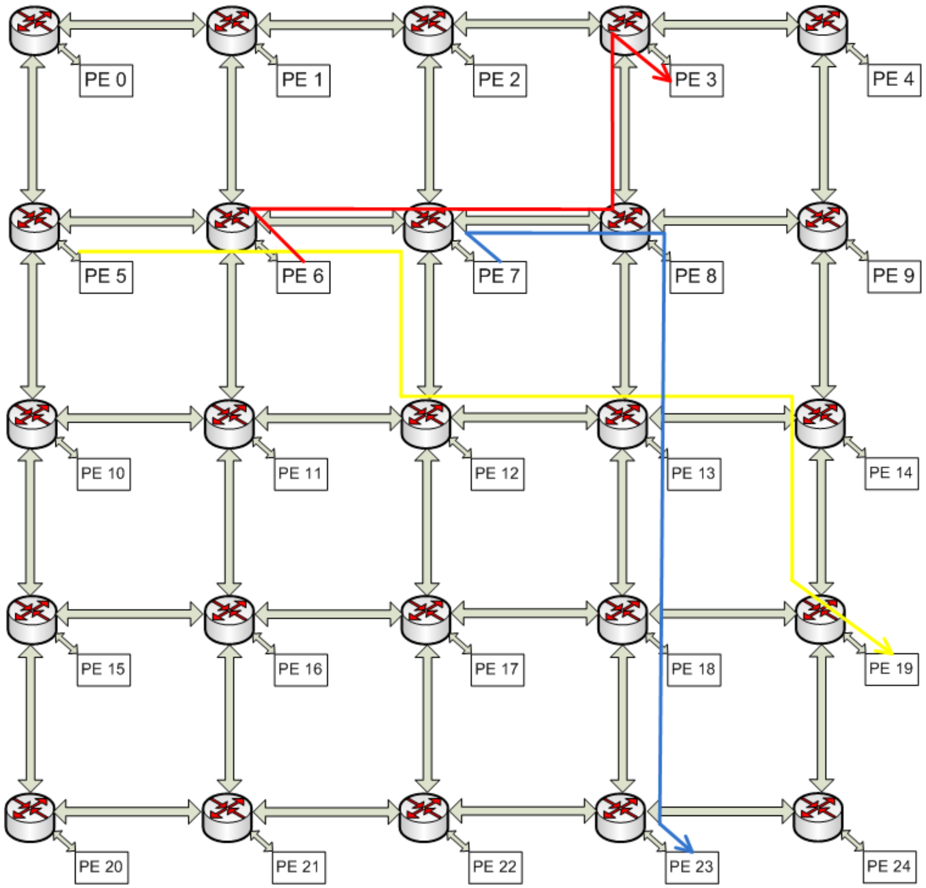
WCET Benchmarks Instruction Throughput (higher is better)



WCET Benchmarks Latency (lower is better)

Multicore PRET

In today's multicore architectures, one thread can disrupt the timing of another thread *even if they are running on different cores and are not communicating*!



Our preliminary work shows that control over timing enables conflict-free routing of messages in a network on chip, making it possible to have non-interfering programs on a multicore PRET.

Status of the PRET project

- Results:
  - PTArm implemented on Xilinx Virtex 5 & 6 FPGA.
  - Multicore PRET demonstration on real-time CFD app.
  - UNISIM simulator of the PTArm facilitates experimentation.
  - DRAM controller with repeatable timing and DMA support.
  - PRET-like utilities implemented on COTS Arm.
- Much still to be done:
  - Realize MTFD, interrupt I/O, compiler toolchain, scratchpad management, etc.

A Key Next Step:
Parametric PRET Architectures

```
set_time r1, 1s
// Code block
MTFD r1
```

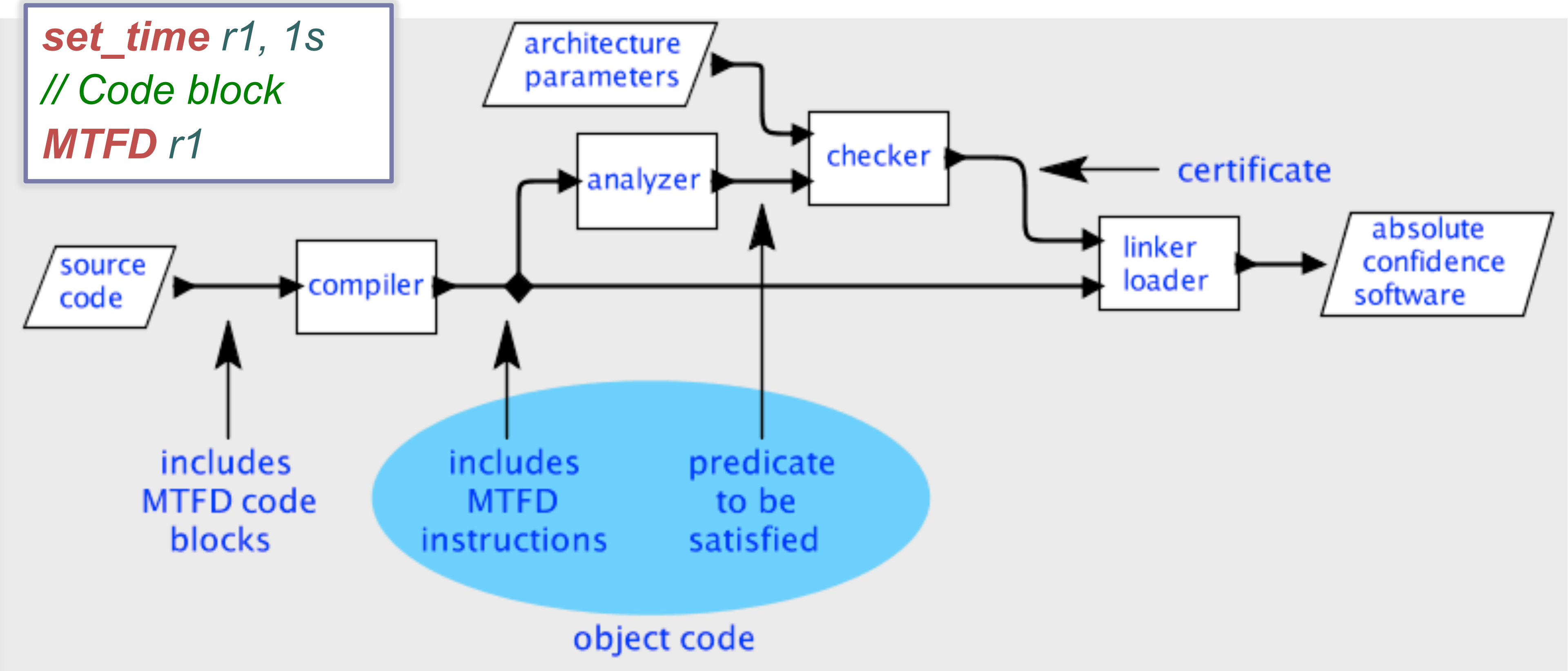
ISA that admits a variety of implementations:

- Variable clock rates and energy profiles
- Variable number of cycles per instruction
- Latency of memory access varying by address
- Varying sizes of memory regions
- …

A given program may meet deadlines on only some realizations of the same parametric PRET ISA.

Realizing the MTFD instruction on a parametric PRET machine

```
set_time r1, 1s
// Code block
MTFD r1
```


architecture parameters
source code
compiler
analyzer
checker
certificate
linker loader
absolute confidence software
includes MTFD code blocks
includes MTFD instructions
predicate to be satisfied
object code

The goal is to make software that will run correctly on a variety of implementations of the ISA, and that correctness can be checked for each implementation.

Potential Uses of PRET Machines

- Deeply embedded applications (CPS)
- Safety-critical systems
- High-precision systems
- Extremely low power applications
- Real-time coprocessor (OpenRT?)
  - High performance I/O
  - Memory controllers
  - High precision I/O
- Mixed hardware/software designs

PRET Publications

http://chess.eecs.berkeley.edu/pret/

- S. Edwards and E. A. Lee, "**The Case for the Precision Timed (PRET) Machine**," in the *Wild and Crazy Ideas* Track of DAC, June 2007.
- B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards and E. A. Lee, "**Predictable programming on a precision timed architecture**," CASES 2008.
- S. Edwards, S. Kim, E. A. Lee, I. Liu, H. Patel and M. Schoeberl, "**A Disruptive Computer Design Idea: Architectures with Repeatable Timing**," ICCD 2009.
- D. Bui, H. Patel, and E. Lee, "**Deploying hard real-time control software on chip-multiprocessors**," RTCSA 2010.
- Bui, E. A. Lee, I. Liu, H. D. Patel and J. Reineke, "**Temporal Isolation on Multiprocessing Architectures**," DAC 2011.
- J. Reineke, I. Liu, H. D. Patel, S. Kim, E. A. Lee, **PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation** (to appear), CODES+ISSS, Taiwan, October, 2011.
- S. Bensalem, K. Goossens, C. M. Kirsch, R. Obermaisser, E. A. Lee, J. Sifakis, **Time-Predictable and Composable Architectures for Dependable Embedded Systems**, Tutorial Abstract (to appear), EMSOFT, Taiwan, October, 2011