



# Time for High-Confidence Distributed Embedded Systems

**Edward Ashford Lee**

*Robert S. Pepper Distinguished Professor  
UC Berkeley*

*Key collaborators:*

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

Invited keynote talk

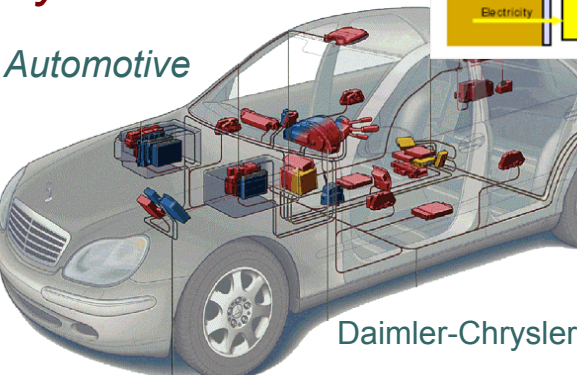
**IEEE Real-Time Systems Symposium (RTSS)**

*San Juan, Puerto Rico  
Dec. 4-7, 2012*

# The Context Cyber-Physical Systems (CPS)

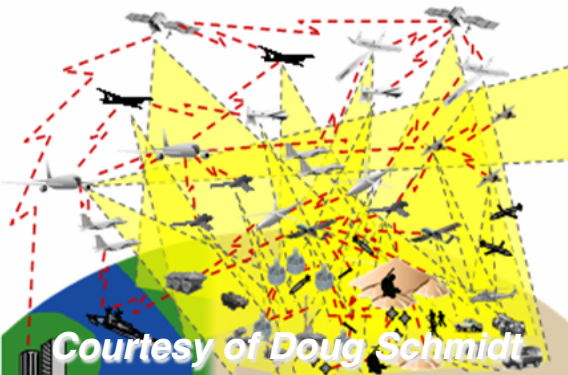
*Orchestrating  
networked  
computational  
resources with  
physical  
systems*

Automotive



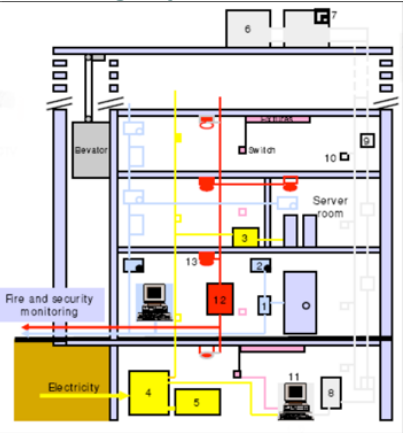
Daimler-Chrysler

Military systems:



Courtesy of Doug Schmidt

Building Systems



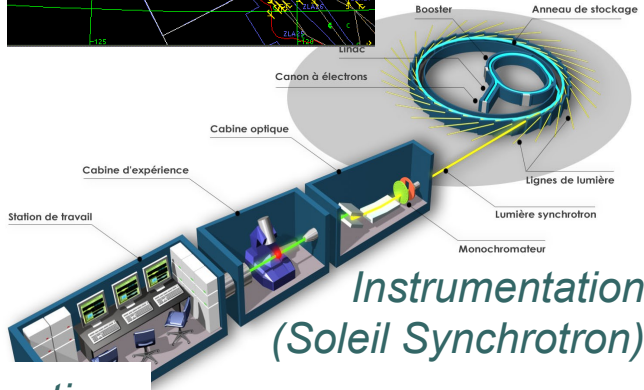
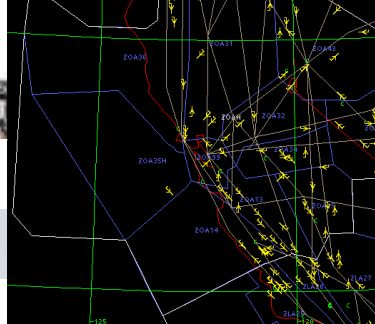
Avionics



Telecommunications

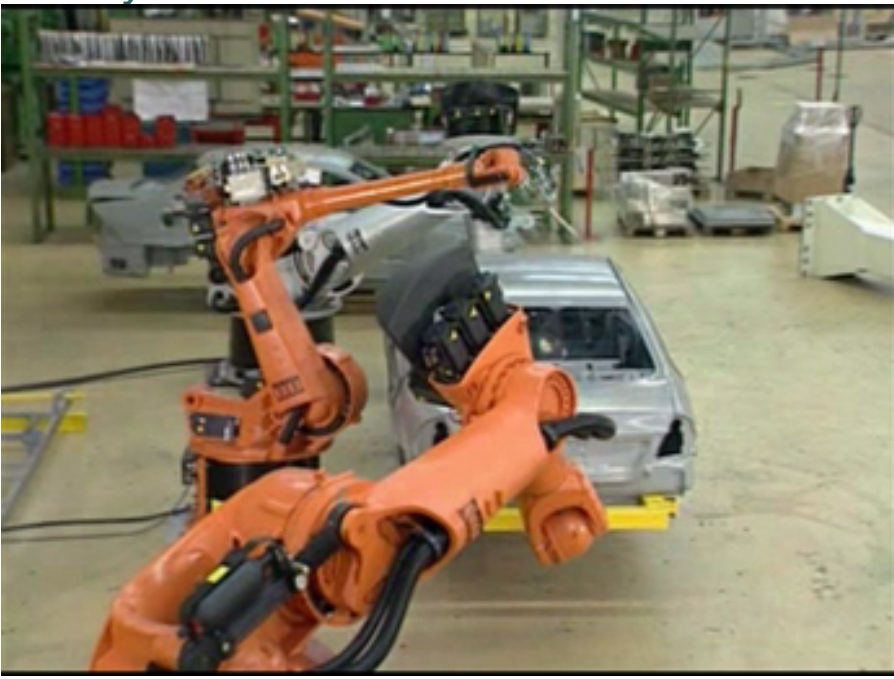


Transportation  
(Air traffic  
control at  
SFO)



Instrumentation  
(Soleil Synchrotron)

Factory automation



Courtesy of Kuka Robotics Corp.

Power  
generation and  
distribution



Courtesy of  
General Electric

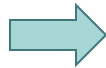
# A Prediction

Time synchronization is going to change the world  
(again)



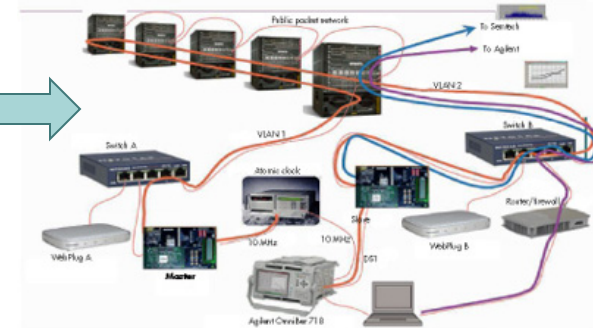
*Gregorian Calendar (BBC history)*

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.”

# Precision Time Protocols (PTP) IEEE 1588 on Ethernet

*Press Release October 1, 2007*



## NEWS RELEASE

For More Information Contact

### Media Contact

Naomi Mitchell  
National Semiconductor  
(408) 721-2142  
[naomi.mitchell@nsc.com](mailto:naomi.mitchell@nsc.com)

### Reader Information

Design Support Group  
(800) 272-9959  
[www.national.com](http://www.national.com)

**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

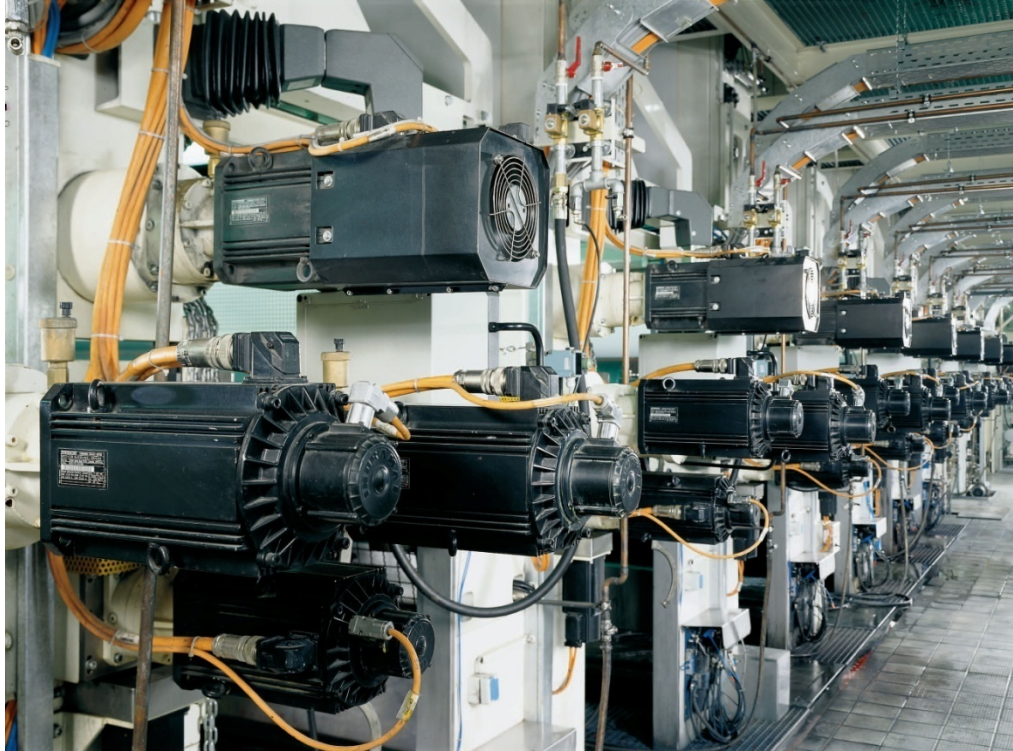


**This is becoming  
routine!**

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

A question we are  
addressing at  
Berkeley: How does  
this change how we  
develop distributed  
CPS software?

# A Cyber-Physical System Printing Press



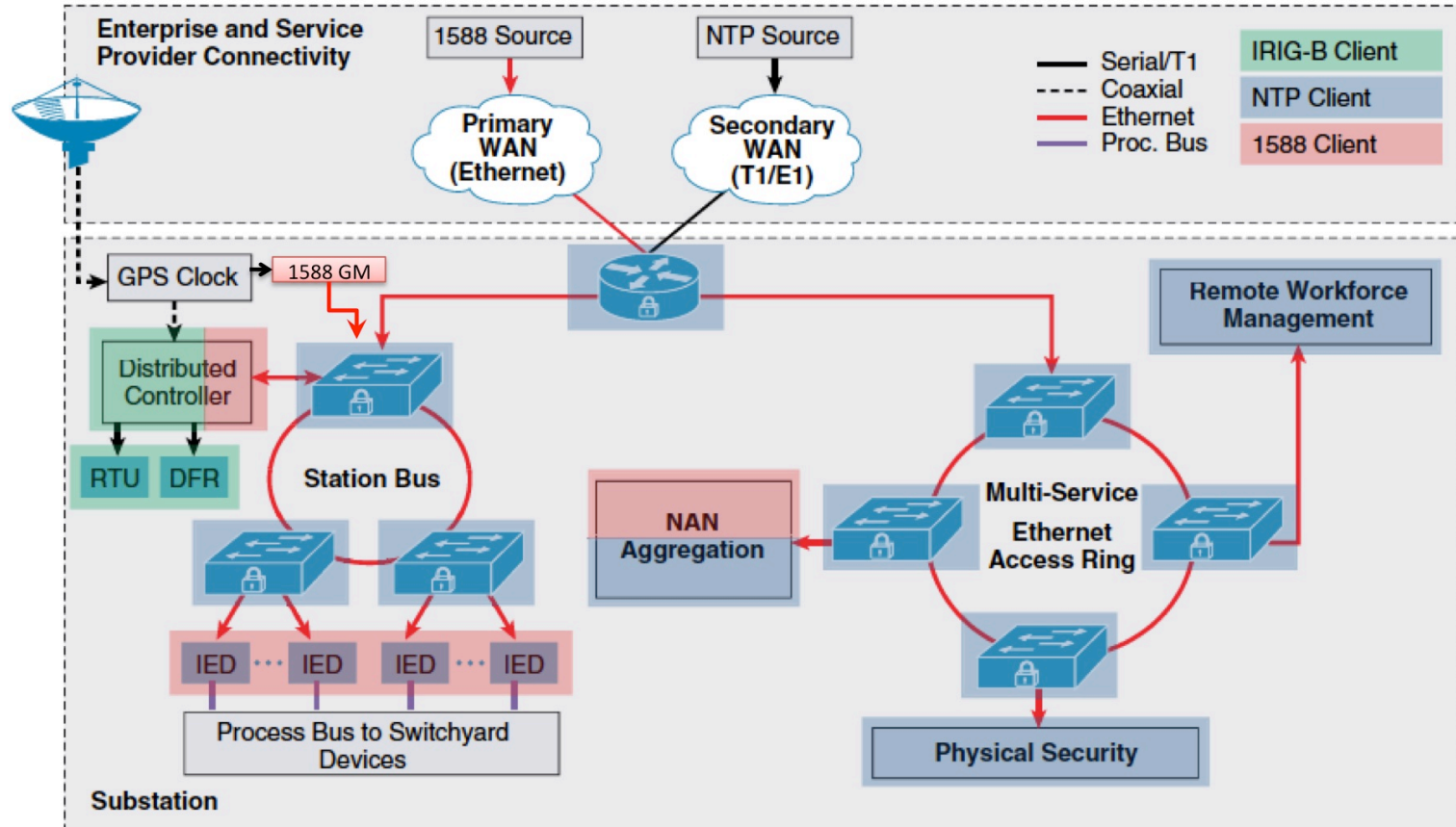
Hundreds of microcontrollers and an Ethernet network are orchestrated with precisions on the order of microseconds.

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

Bosch-Rexroth

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

# PTP in the Network for Critical Infrastructure

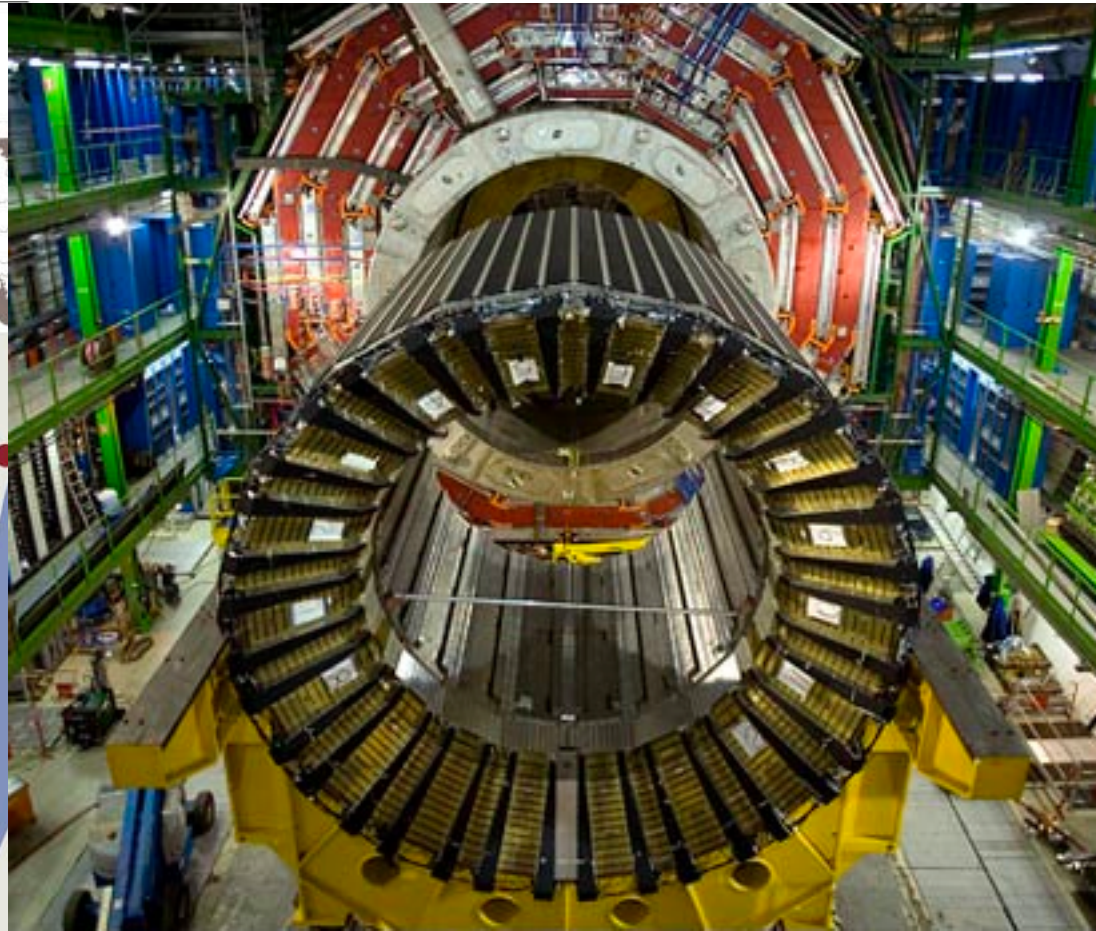
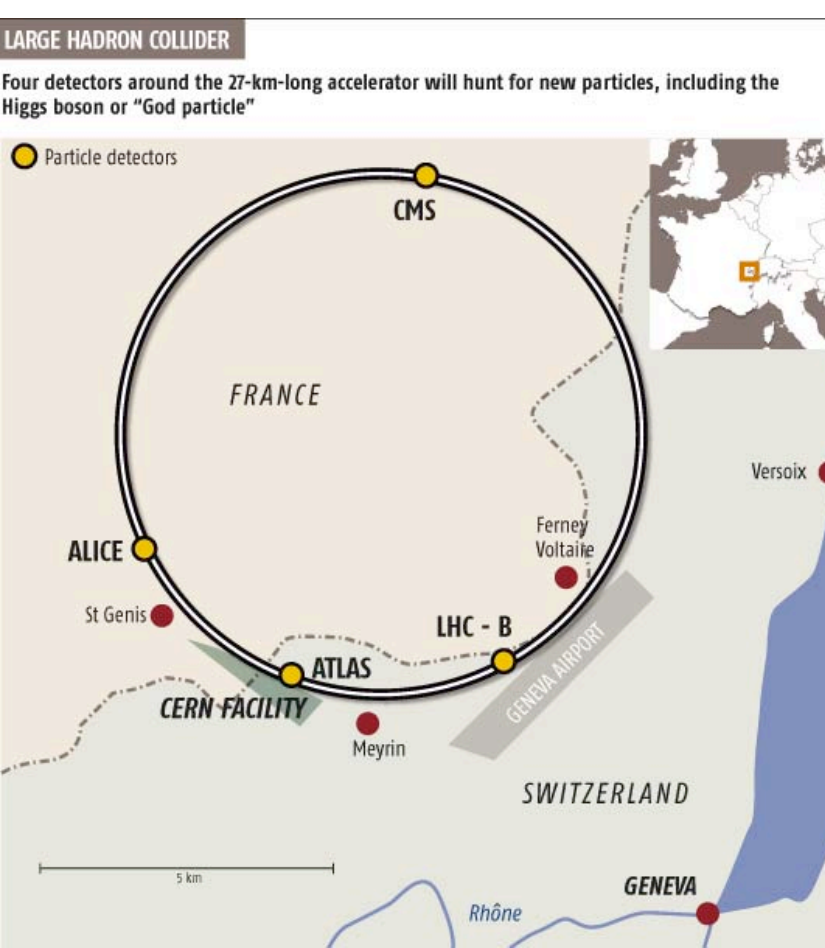


Substation Timing Synchronization Using IEEE-1588 Power Profile (Cisco)

<http://developer.cisco.com/web/tad/sample-solutions-2>

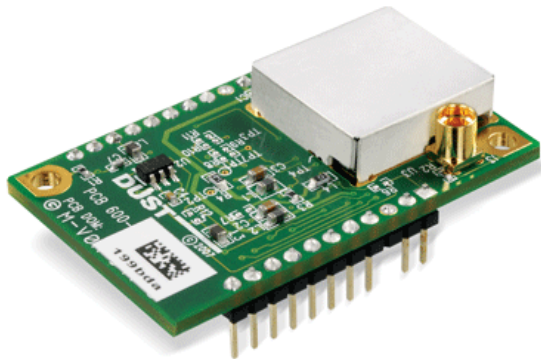
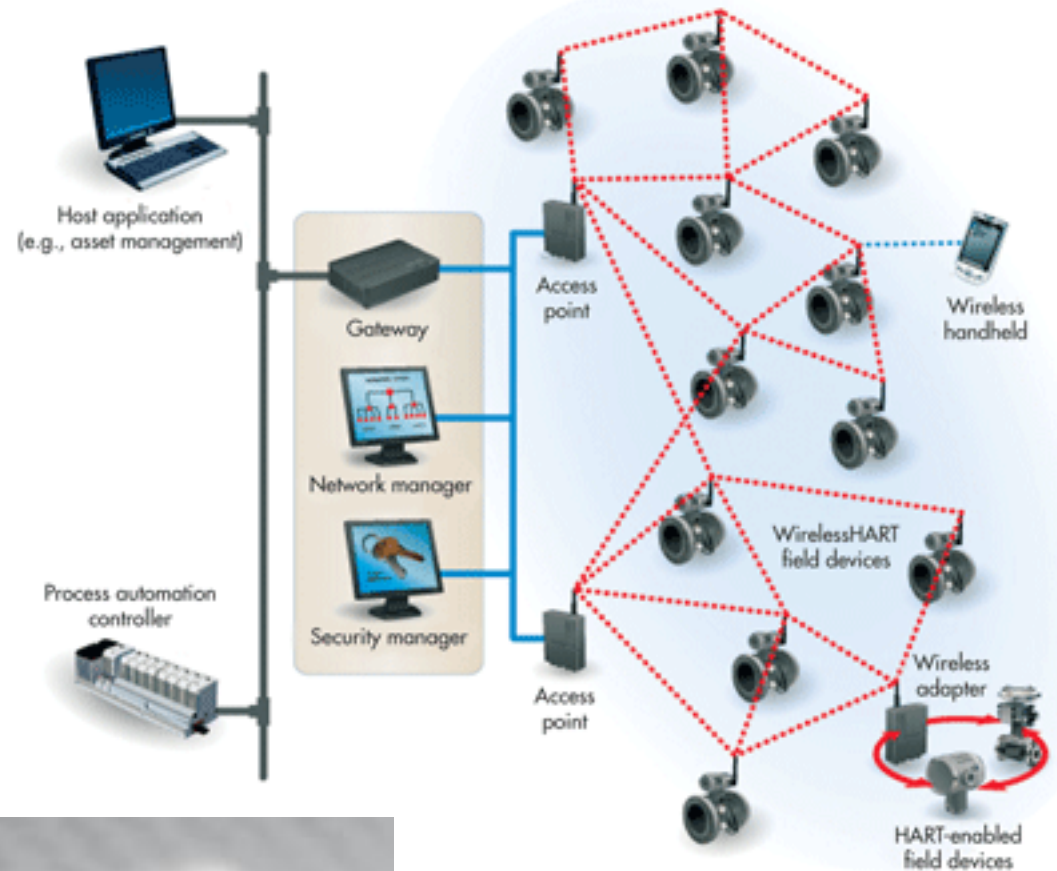
# An Extreme Example: The Large Hadron Collider

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.



# Wireless

Wireless HART uses  
Time Synchronized  
Mesh Protocol (TSMP)  
in a Mote-on-Chip  
(MoC), from Dust  
Networks Inc.



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

For cyber-physical systems,  
*programs* do not adequately control timing.

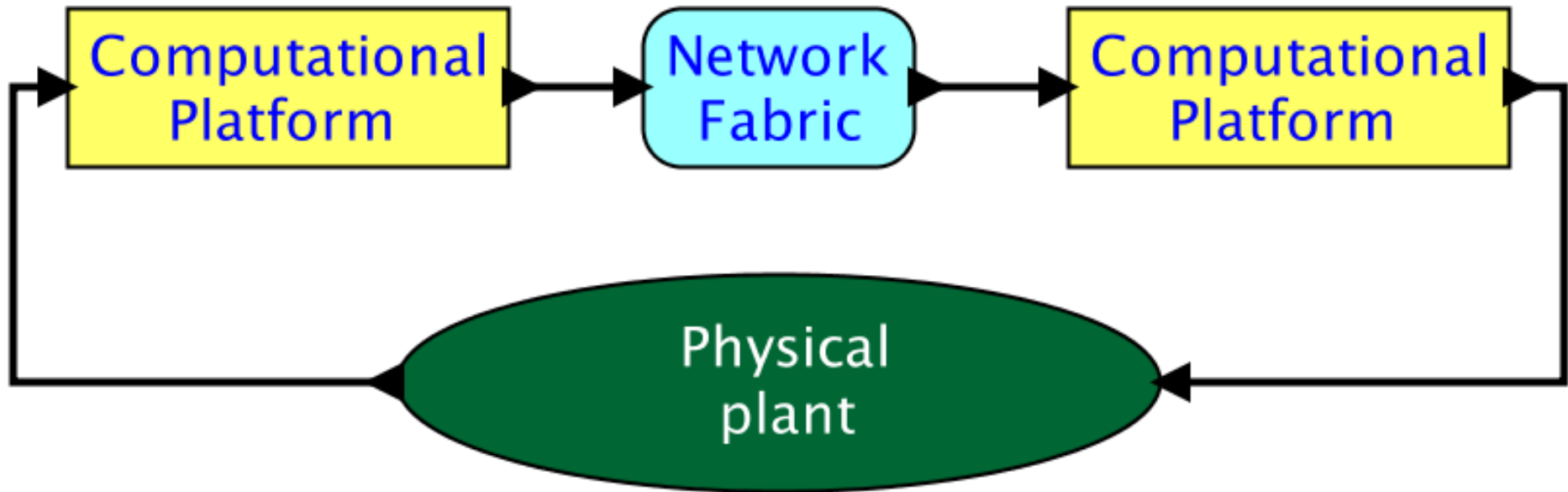
The core notions of “computation” today ignore time.

The core notions tomorrow will not...

# Challenges that We are Addressing

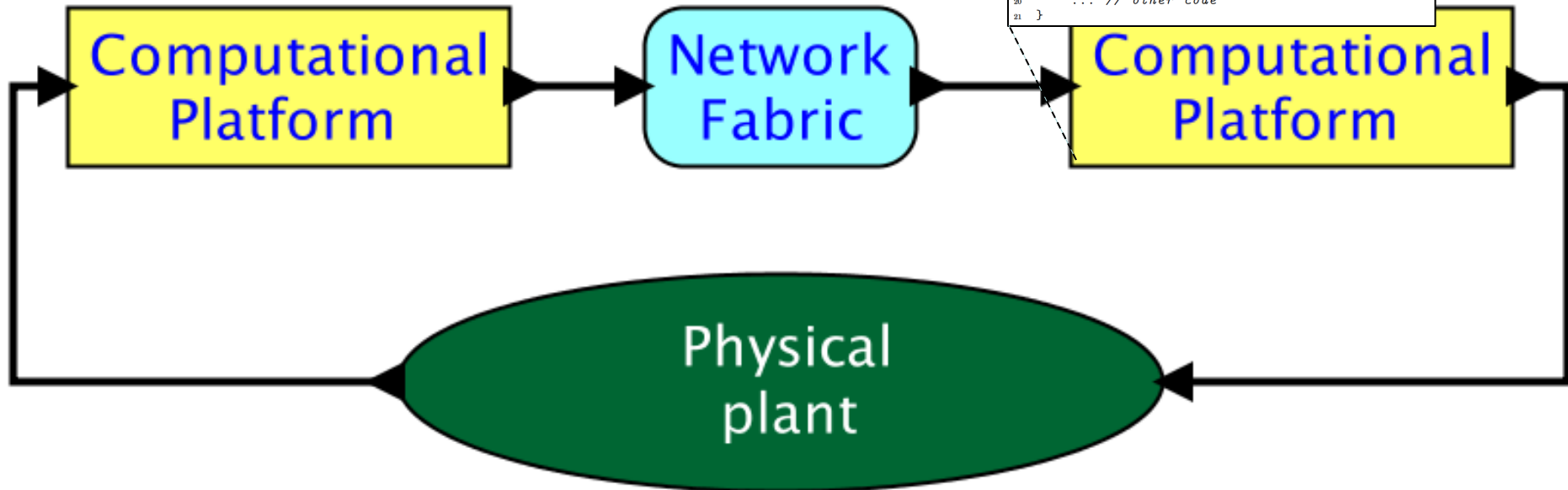
- Representation of time
  - Data types, superdense time, operations on time, etc.
- Gaining control over timing in software
  - PRET – Precision-timed computer architecture
- Multiform time
  - Hierarchical clocks proceeding at different rates.
- Joint modeling of functionality and implementation
 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

## Schematic of a simple CPS:



Computation given in an  
untimed, imperative language.

```
1 void initTimer(void) {  
2     SysTickPeriodSet(SysCtlClockGet() / 1000);  
3     SysTickEnable();  
4     SysTickIntEnable();  
5 }  
6 volatile uint timer_count = 0;  
7 void ISR(void) {  
8     if(timer_count != 0) {  
9         timer_count--;  
10    }  
11 }  
12 int main(void) {  
13     SysTickIntRegister(&ISR);  
14     .. // other init  
15     timer_count = 2000;  
16     initTimer();  
17     while(timer_count != 0) {  
18         ... code to run for 2 seconds  
19     }  
20     ... // other code  
21 }
```



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

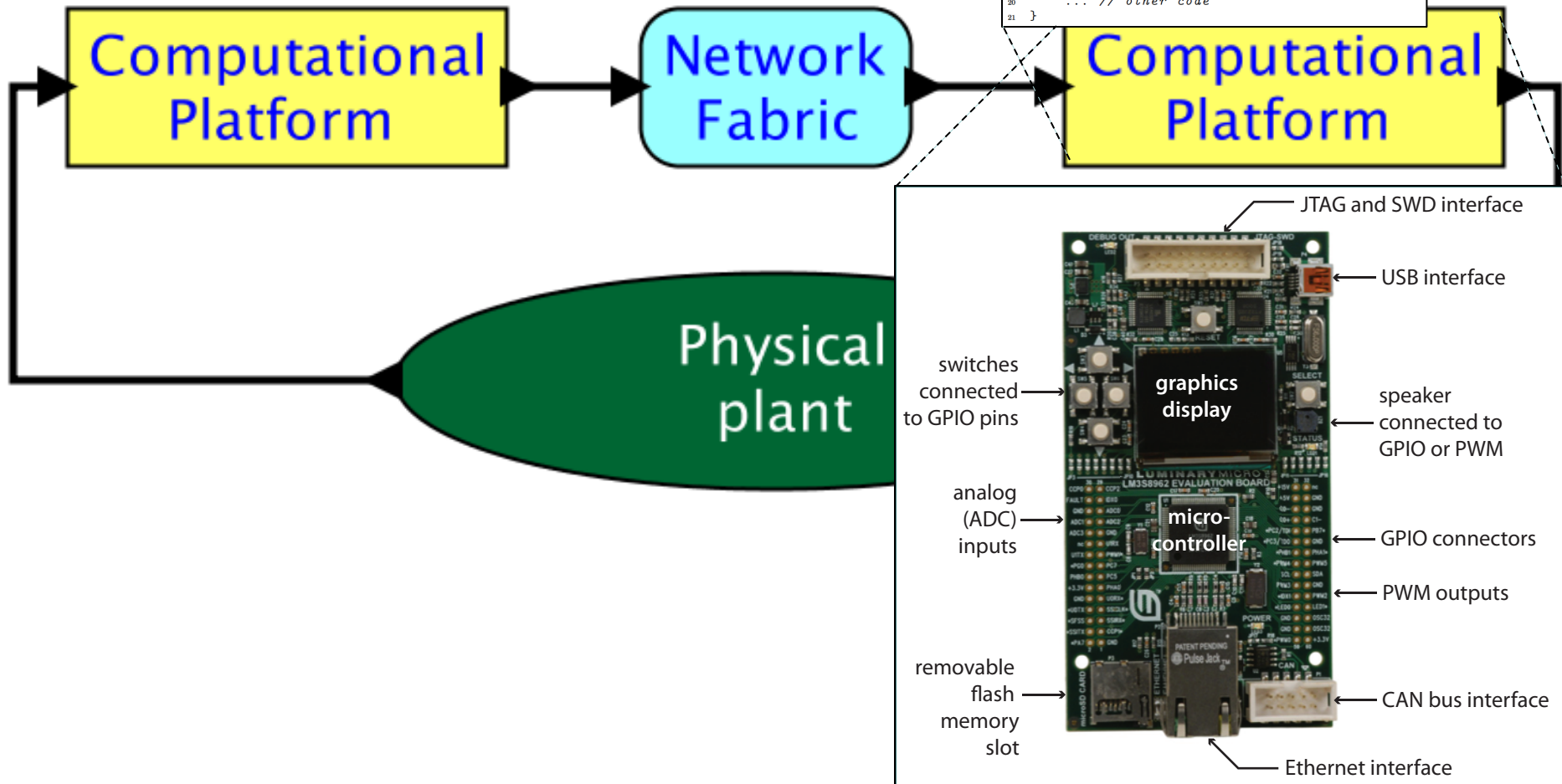
Computational  
Platform

```
1 void initTimer(void) {
2     SysTickPeriodSet(SysCtlClockGet() / 1000);
3     SysTickEnable();
4     SysTickIntEnable();
5 }
6 volatile uint timer_count = 0;
7 void ISR(void) {
8     if(timer_count != 0) {
9         timer_count--;
10    }
11 }
12 int main(void) {
13     SysTickIntRegister(&ISR);
14     .. // other init
15     timer_count = 2000;
16     initTimer();
17     while(timer_count != 0) {
18         ... code to run for 2 seconds
19     }
20     ... // other code
21 }
```

plant

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

```
1 void initTimer(void) {  
2     SysTickPeriodSet(SysCtlClockGet() / 1000);  
3     SysTickEnable();  
4     SysTickIntEnable();  
5 }  
6 volatile uint timer_count = 0;  
7 void ISR(void) {  
8     if(timer_count != 0) {  
9         timer_count--;  
10    }  
11 }  
12 int main(void) {  
13     SysTickIntRegister(&ISR);  
14     .. // other init  
15     timer_count = 2000;  
16     initTimer();  
17     while(timer_count != 0) {  
18         ... code to run for 2 seconds  
19     }  
20     ... // other code  
21 }
```



# 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.*



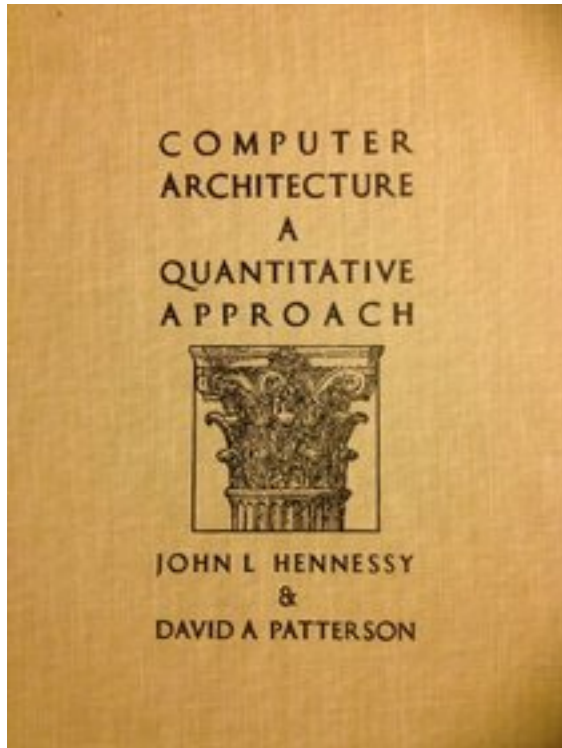
Programmers have to step *outside* the programming abstractions to specify timing behavior.

# 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. Results:

- System behavior emerges only at system integration.
- Manufacturers stockpile parts to suffice for the complete production run of a product.
- Manufacturers cannot leverage improvements in the hardware (e.g. weight, power).
- Any change forces re-testing and re-certifying.
- Designs are over provisioned, increasing cost, weight, and energy usage.

# 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

```
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 }
```



(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.)

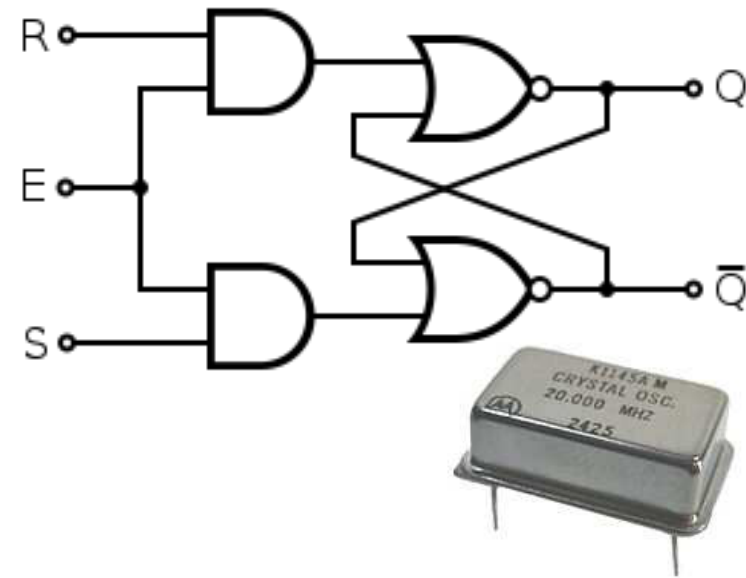
*We can develop **absolute confidence** in the software, in that only a **hardware failure** is an excuse.*

*But not with regards to timing!!*

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

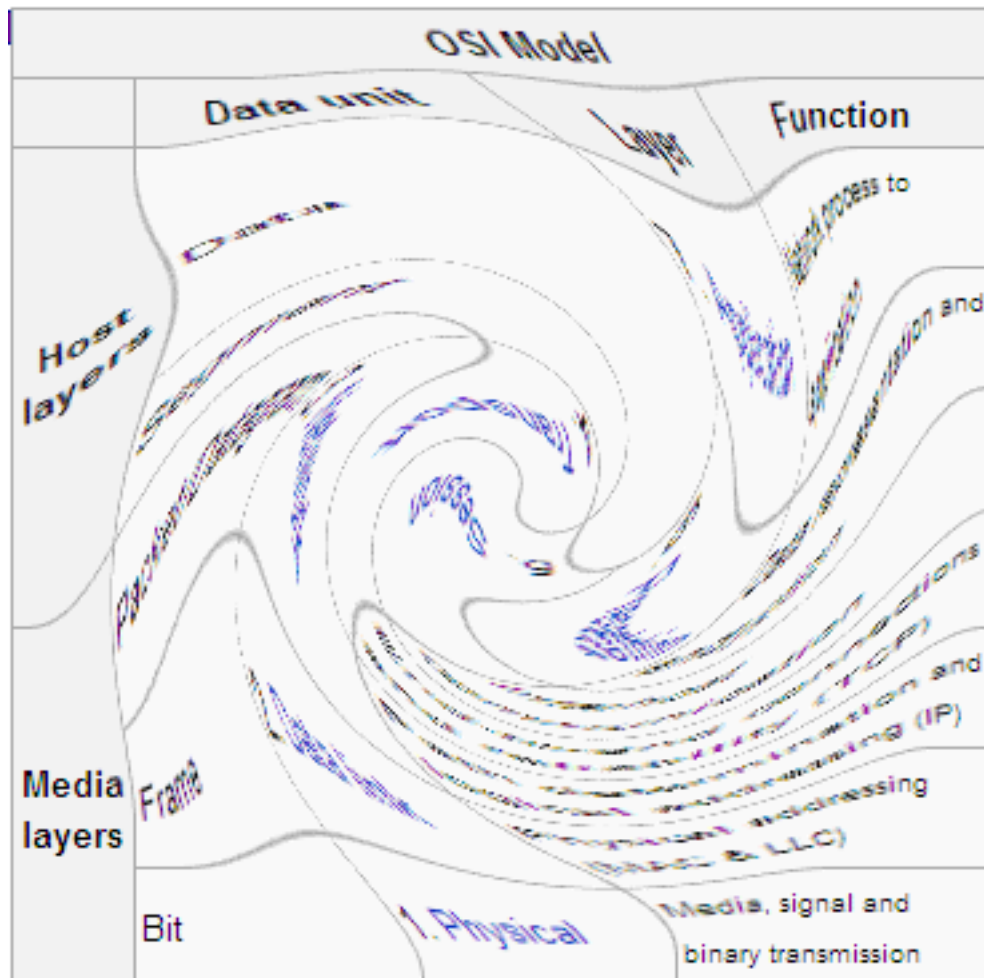
The synchronous digital logic abstraction removes the messiness of transistors.

*... 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*

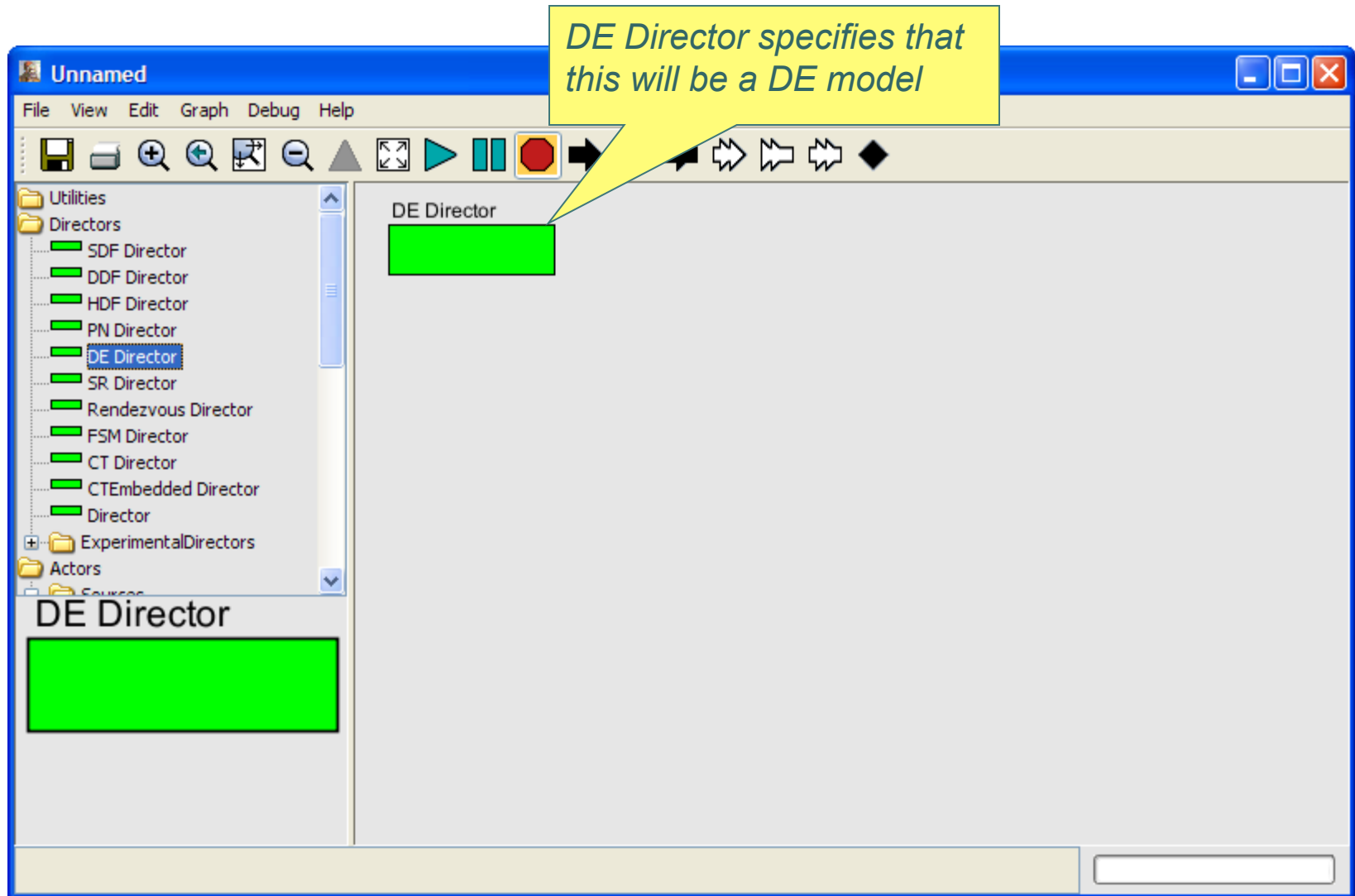
*A Programming Model for Distributed Cyber-Physical Systems  
Based on **Discrete Events (DE)***

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

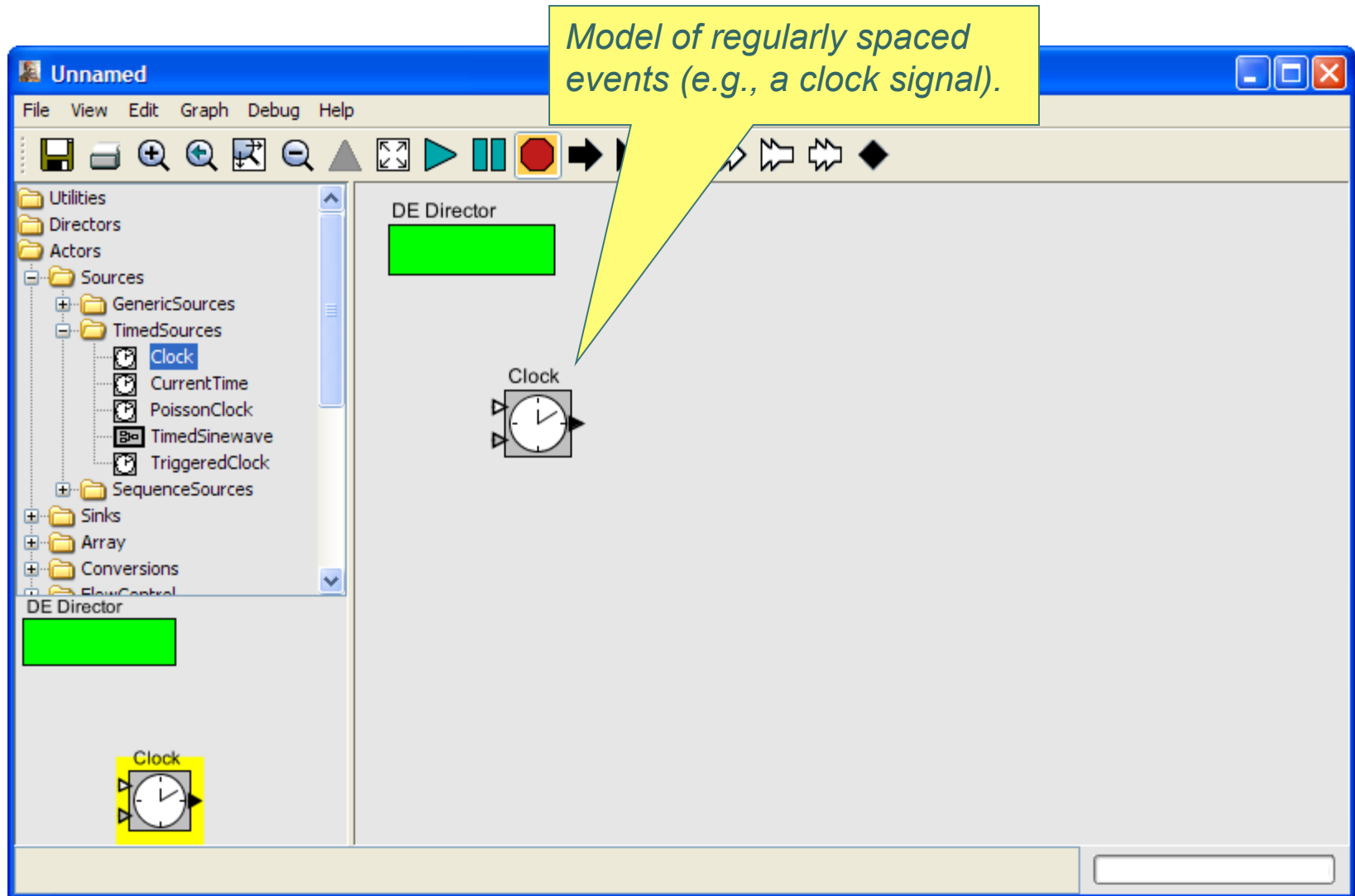
A “correct” execution is one where every actor reacts to input events in time-stamp order.

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

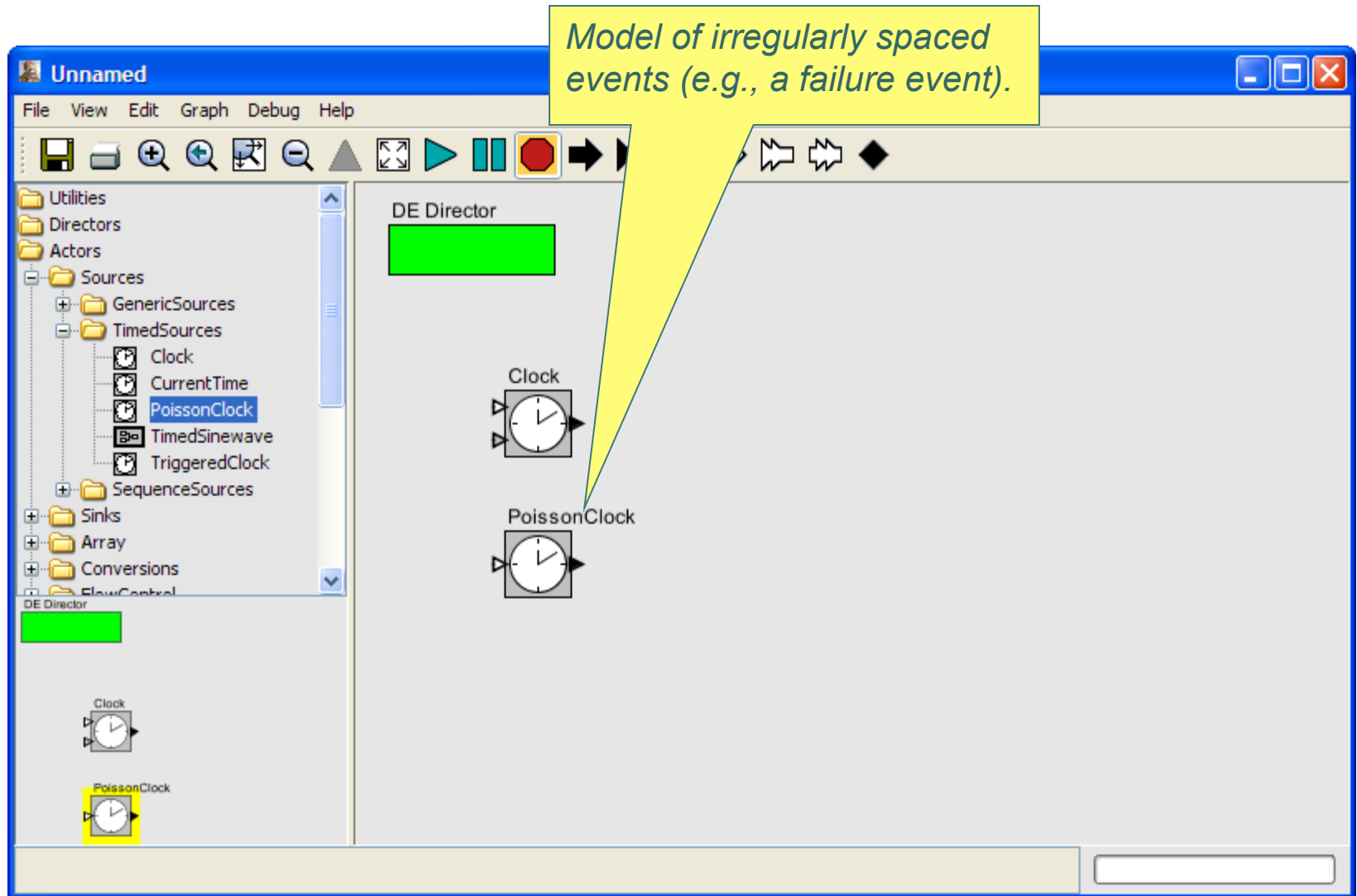
# Discrete-Event Models (in Ptolemy II)



# Discrete-Event Models (in Ptolemy II)

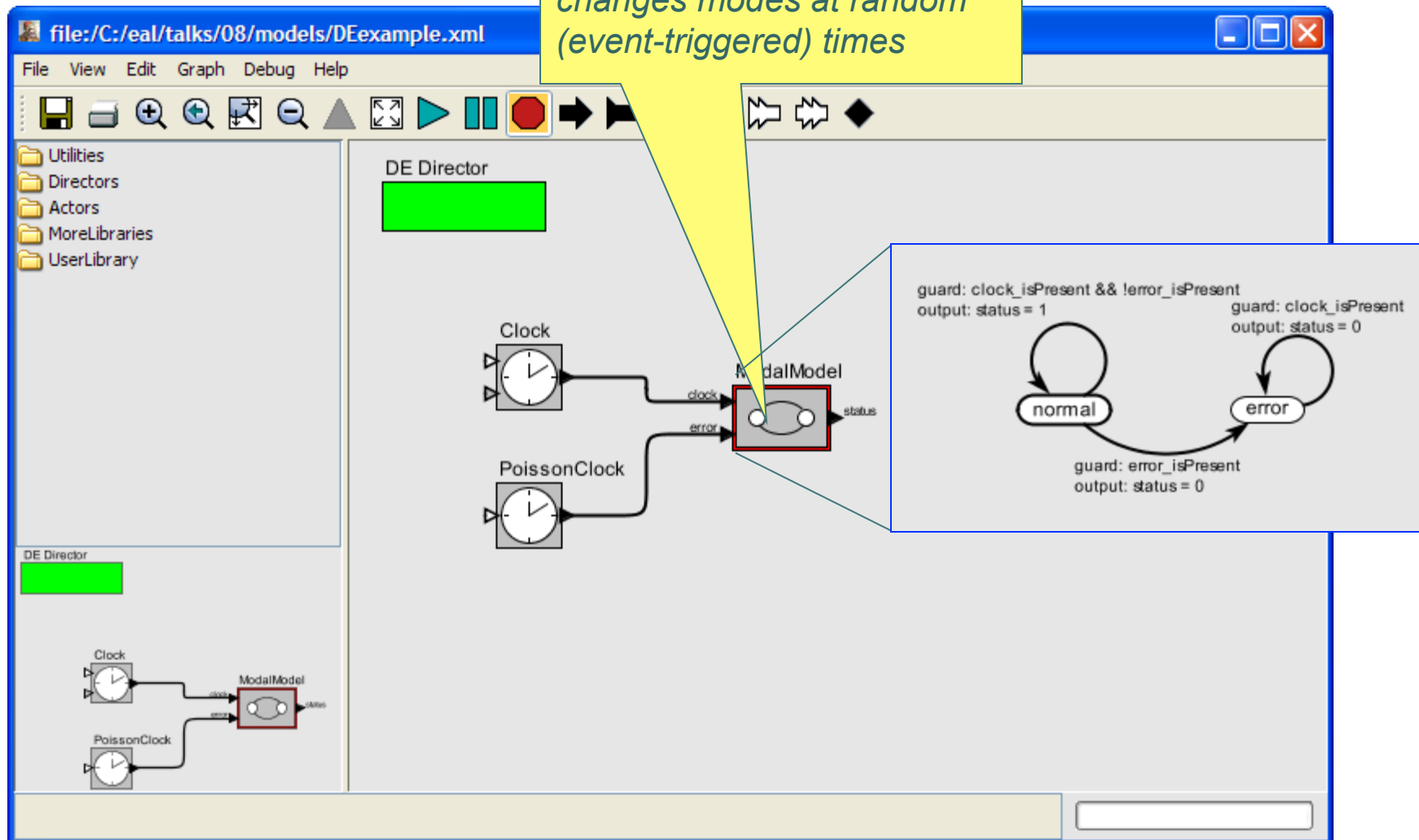


# Discrete-Event Models (in Ptolemy II)

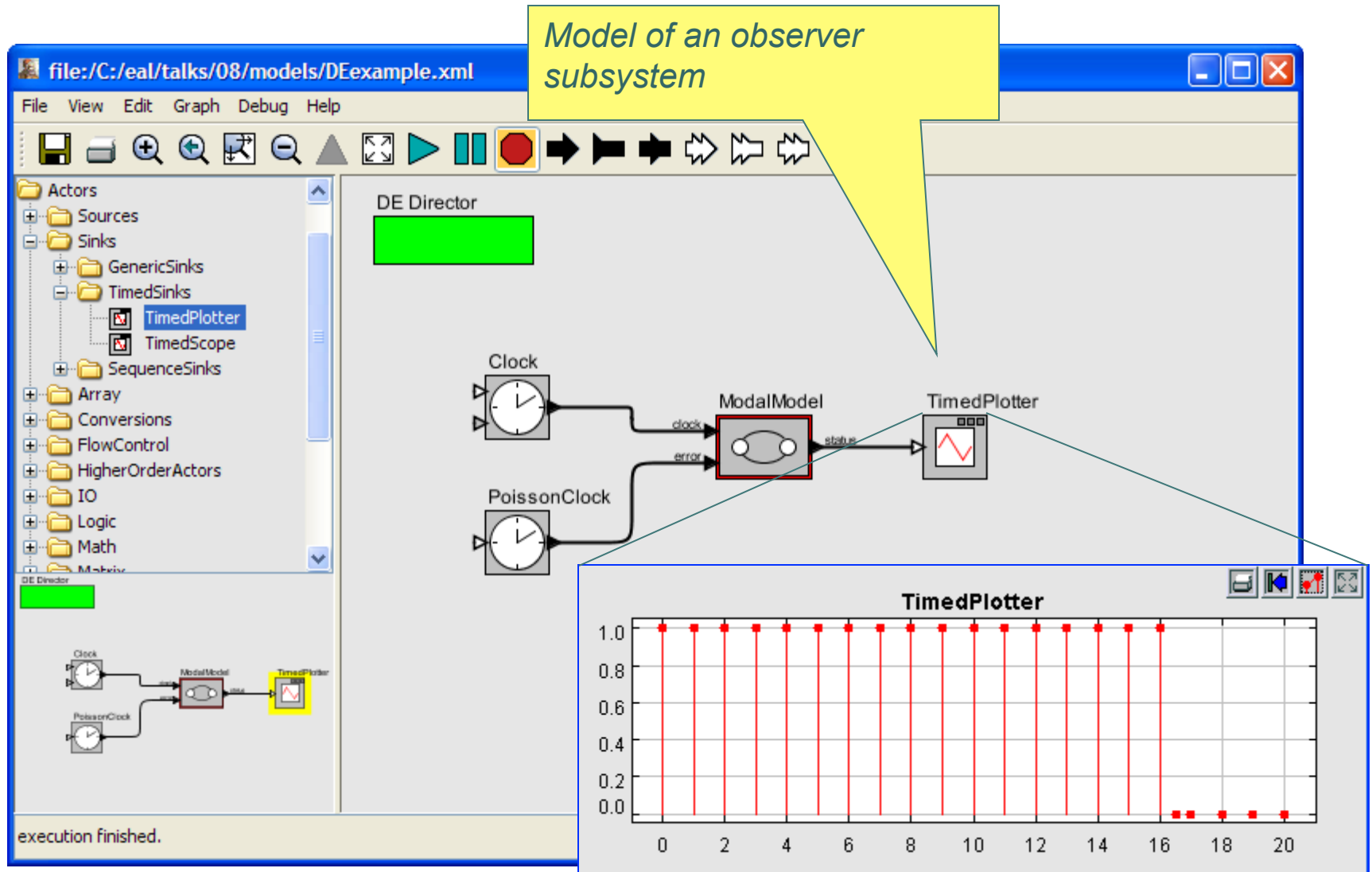


# Discrete-Event Models (in Ptolemy II)

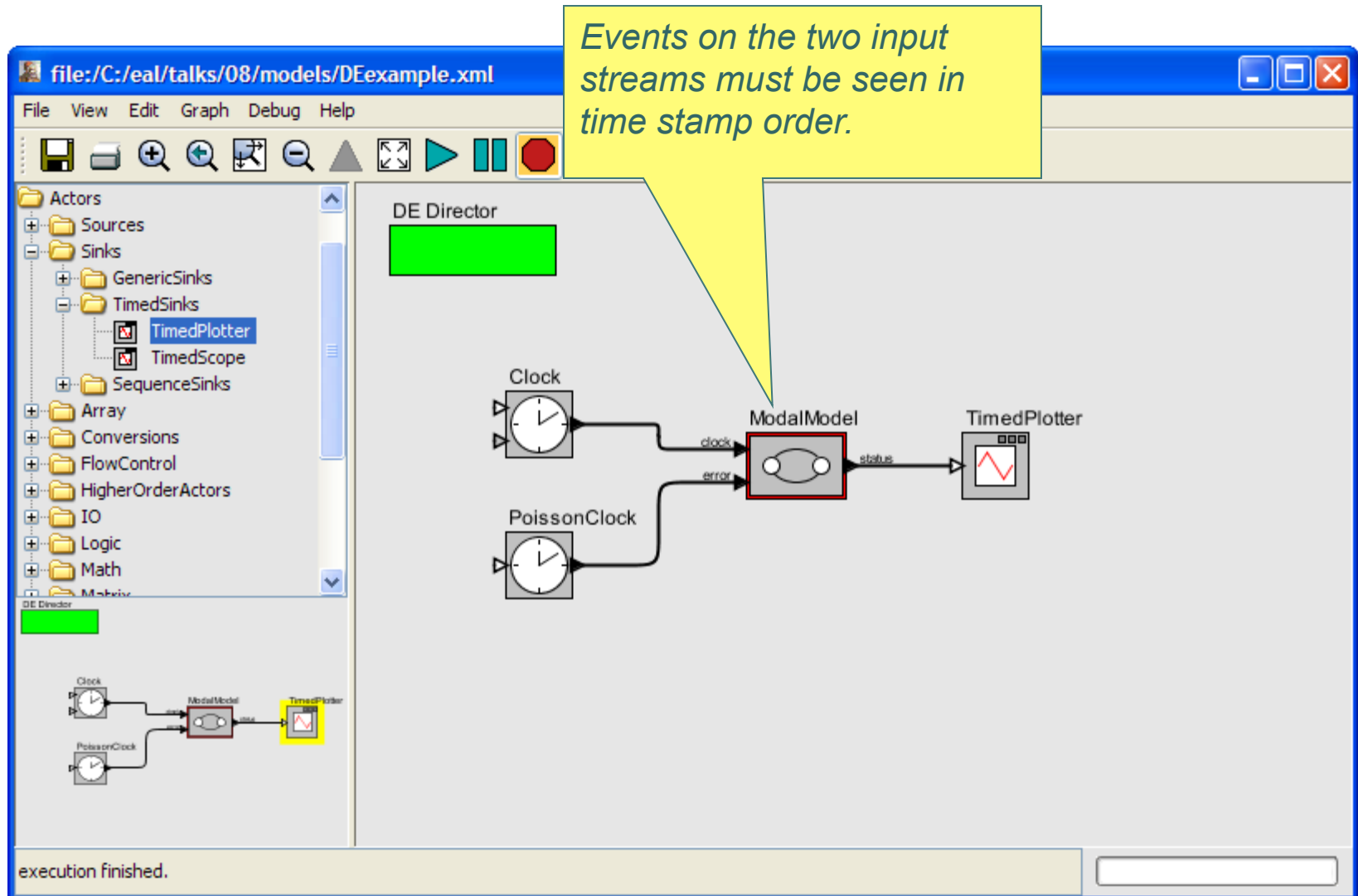
*Model of a subsystem that changes modes at random (event-triggered) times*



# Discrete-Event Models (in Ptolemy II)



# Discrete-Event Models (in Ptolemy II)



# This is a Component Technology

*Model of a subsystem given as an imperative program.*

The image displays a software development environment for a component technology. The main window, titled "file:/C:/eal/talks/08/models/DEexample.xml", shows a hierarchical tree of components on the left, including "Actors", "Sources", "Sinks", "GenericSinks", "TimedSinks", "SequenceSinks", "Array", "Conversions", "FlowControl", "HigherOrderActors", "IO", "Logic", "Math", and "Matrix". The central workspace, labeled "DE Director", contains a diagram of a subsystem model. This model consists of two clock components, "Clock" and "PoissonClock", connected to a "ModelModel" component, which is in turn connected to a "TimedPlotter" component. A yellow callout box points to the "Clock" component with the text "Model of a subsystem given as an imperative program." Below the main workspace, a status bar indicates "execution finished." To the right, a separate window titled "Unnamed" displays the imperative code for the "Clock" component. The code is in Java and includes comments and a "fire()" method that interacts with the "DE Director" to get the current time and period, and to send output values.

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

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

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

    _tentativeCurrentOutputIndex = _currentOutputIndex;

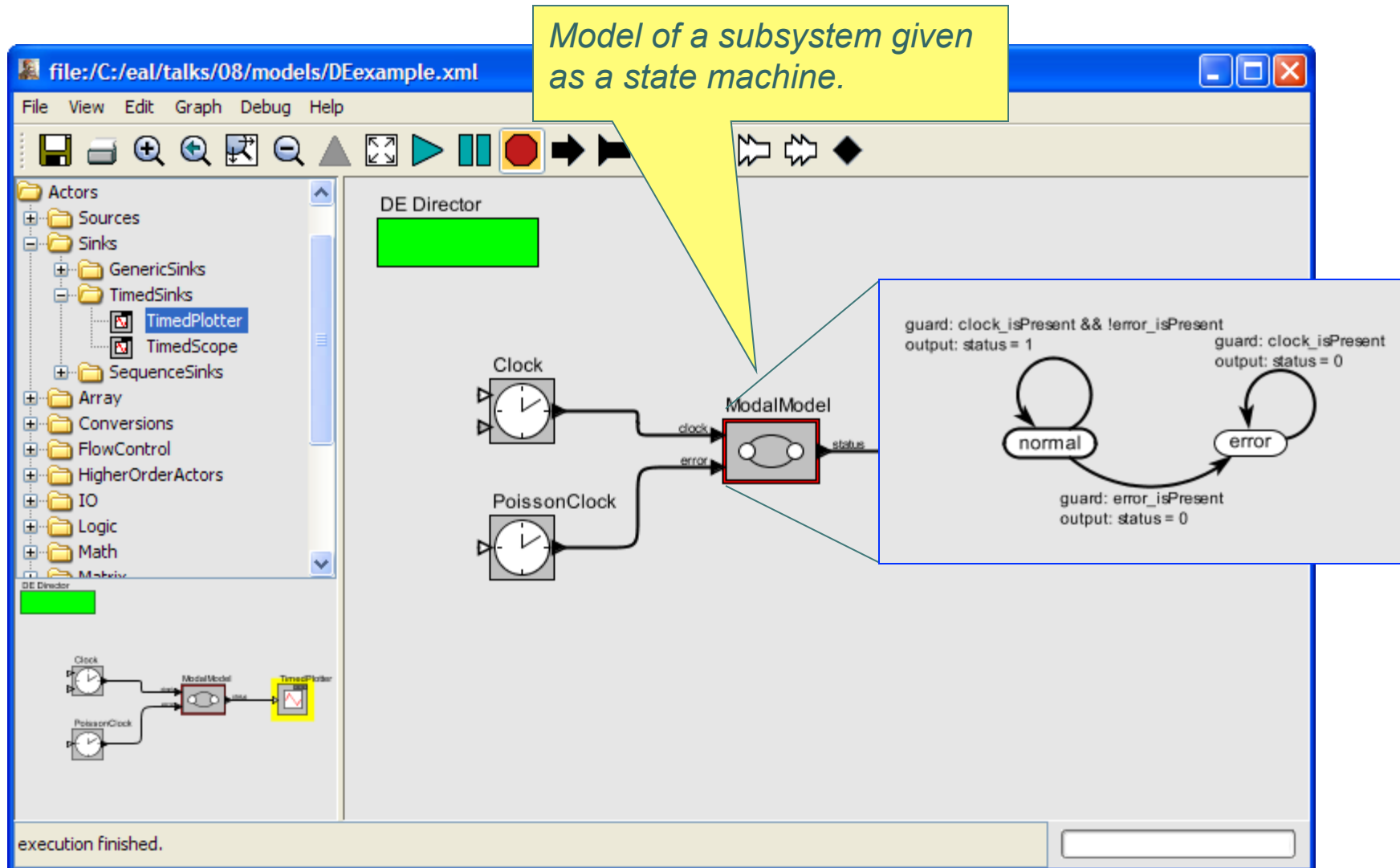
    output.send(0, _getValue(_tentativeCurrentOutputIndex));

    // 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

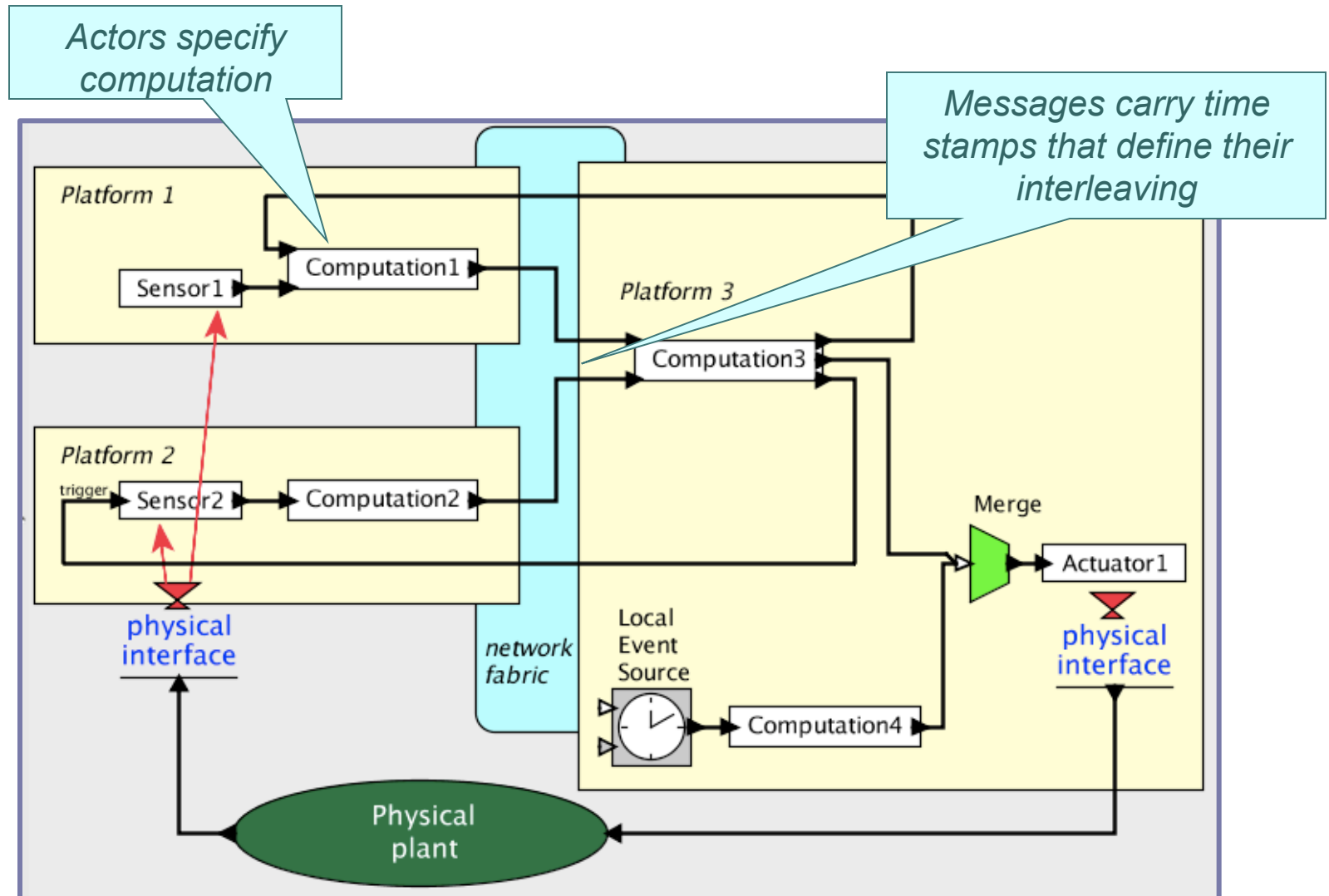


# Using Discrete Event Semantics in Distributed Real-Time Systems

- 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

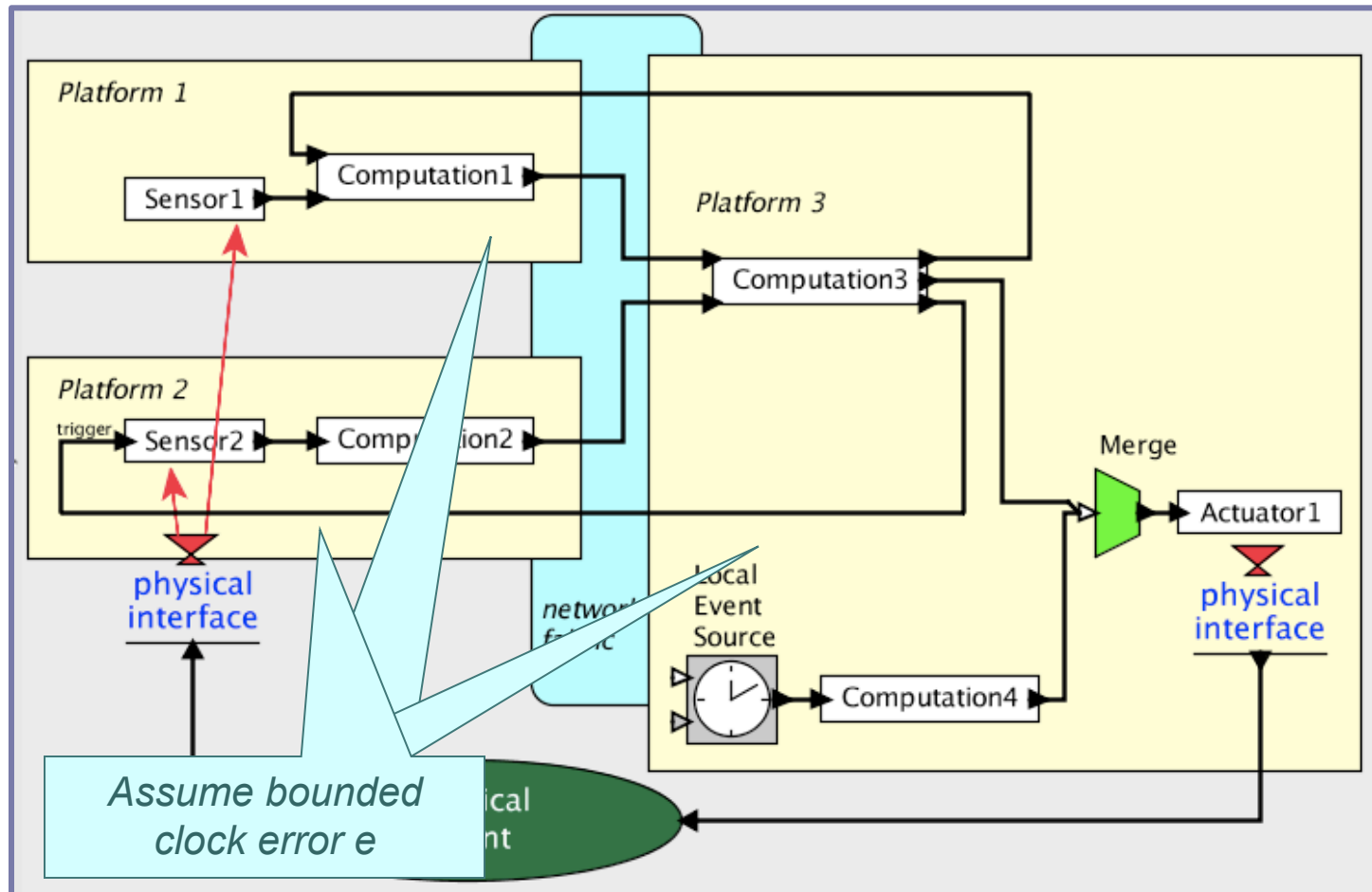
# Ptides: Programming Temporally Integrated Distributed Embedded Systems

## First step: Time-stamped messages.

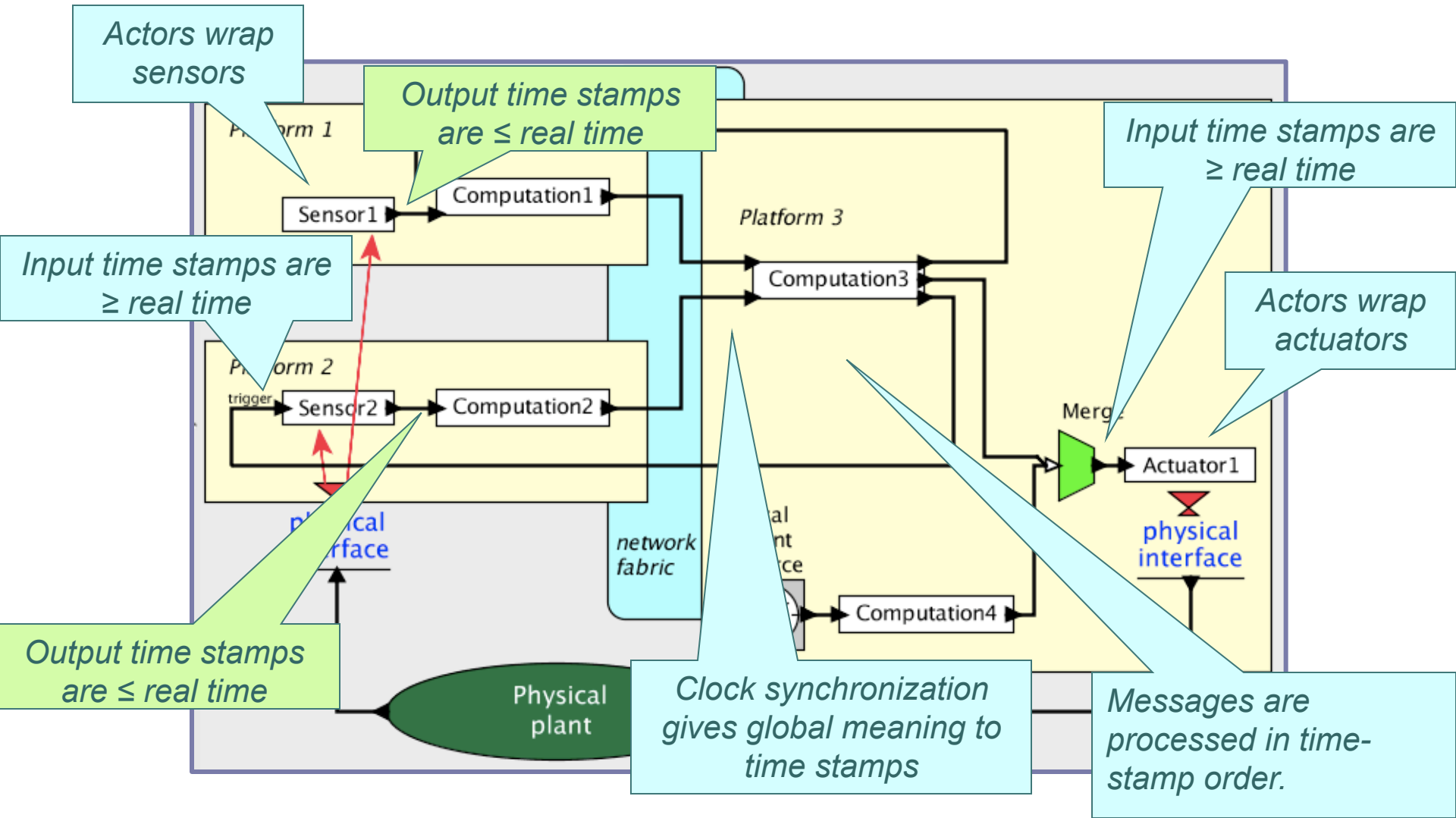


# Ptides: Second step: Network time synchronization

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

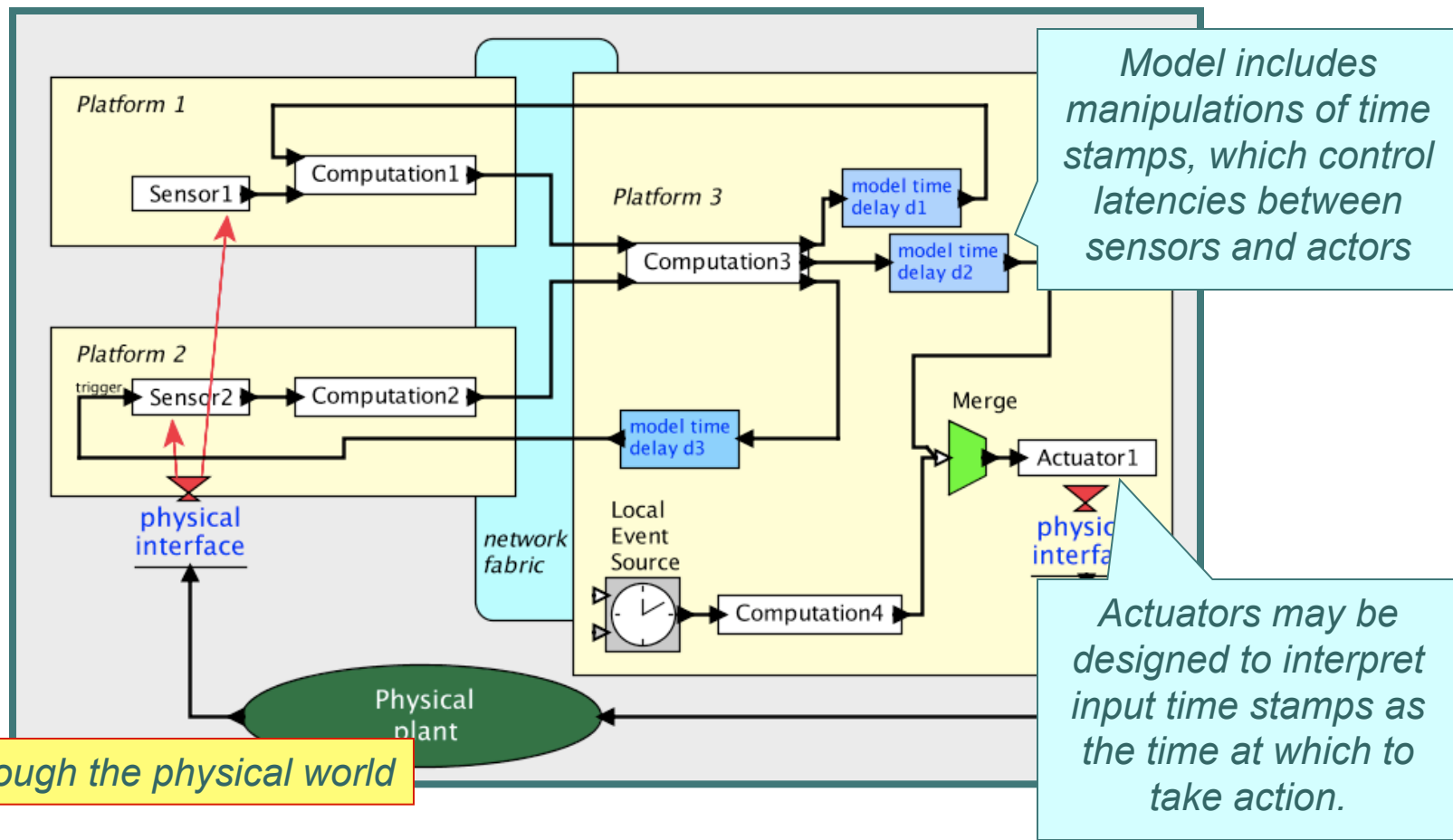


# Ptides: Third step: Bind time stamps to real time at sensors and actuators



# Ptides: Fourth step: Specify latencies in the model

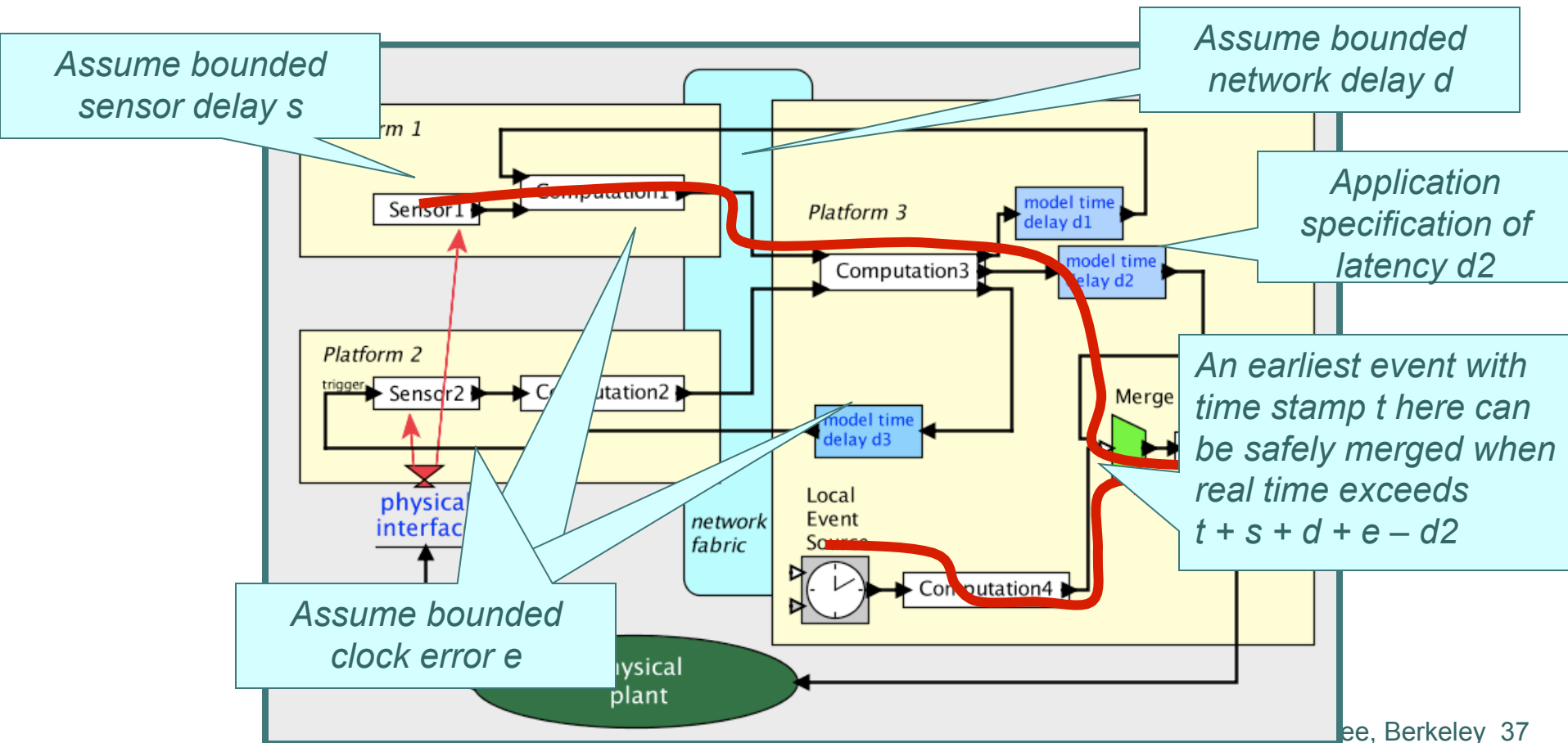
*Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.*



# 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.*



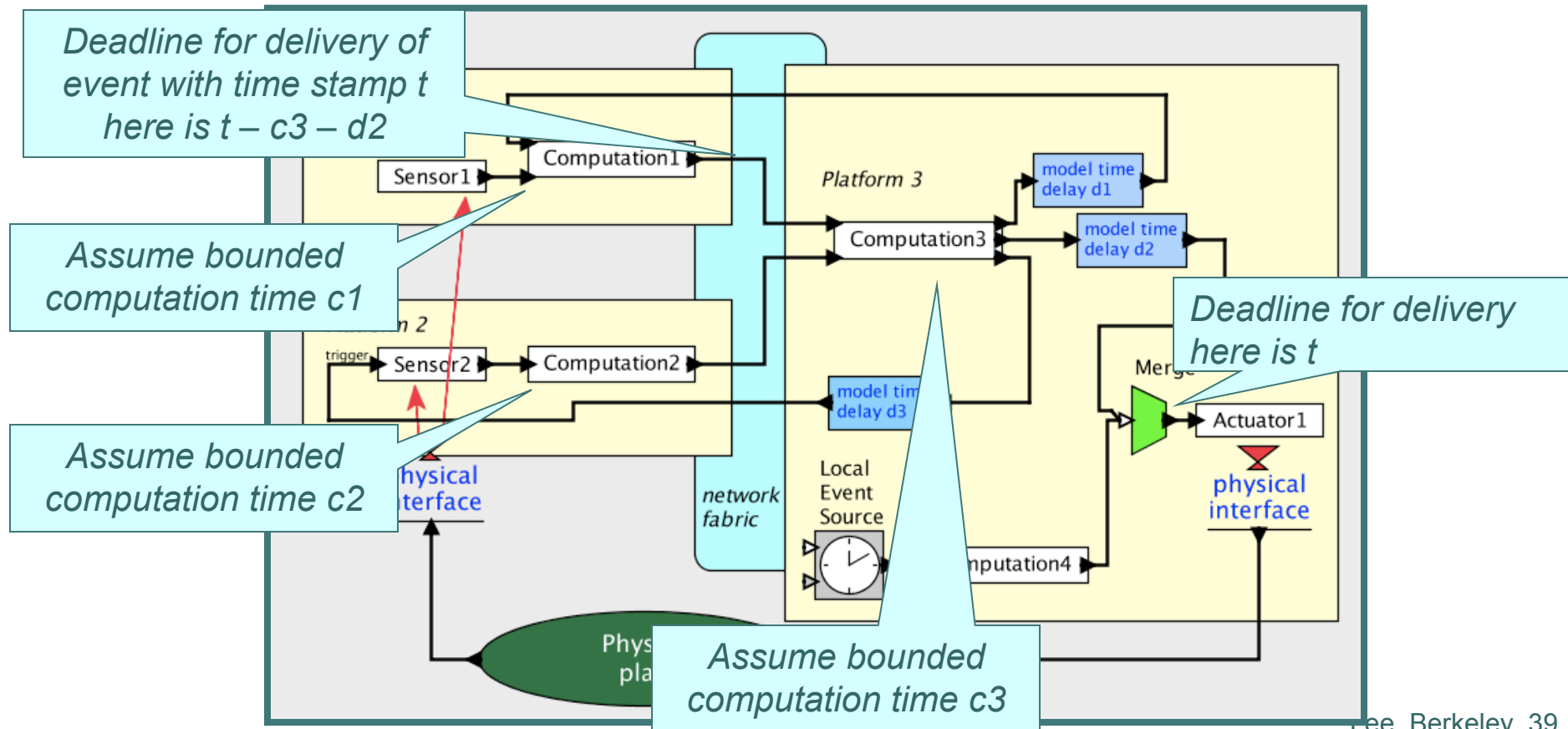
# Bounded network delay is enabled by time synchronization...

- **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

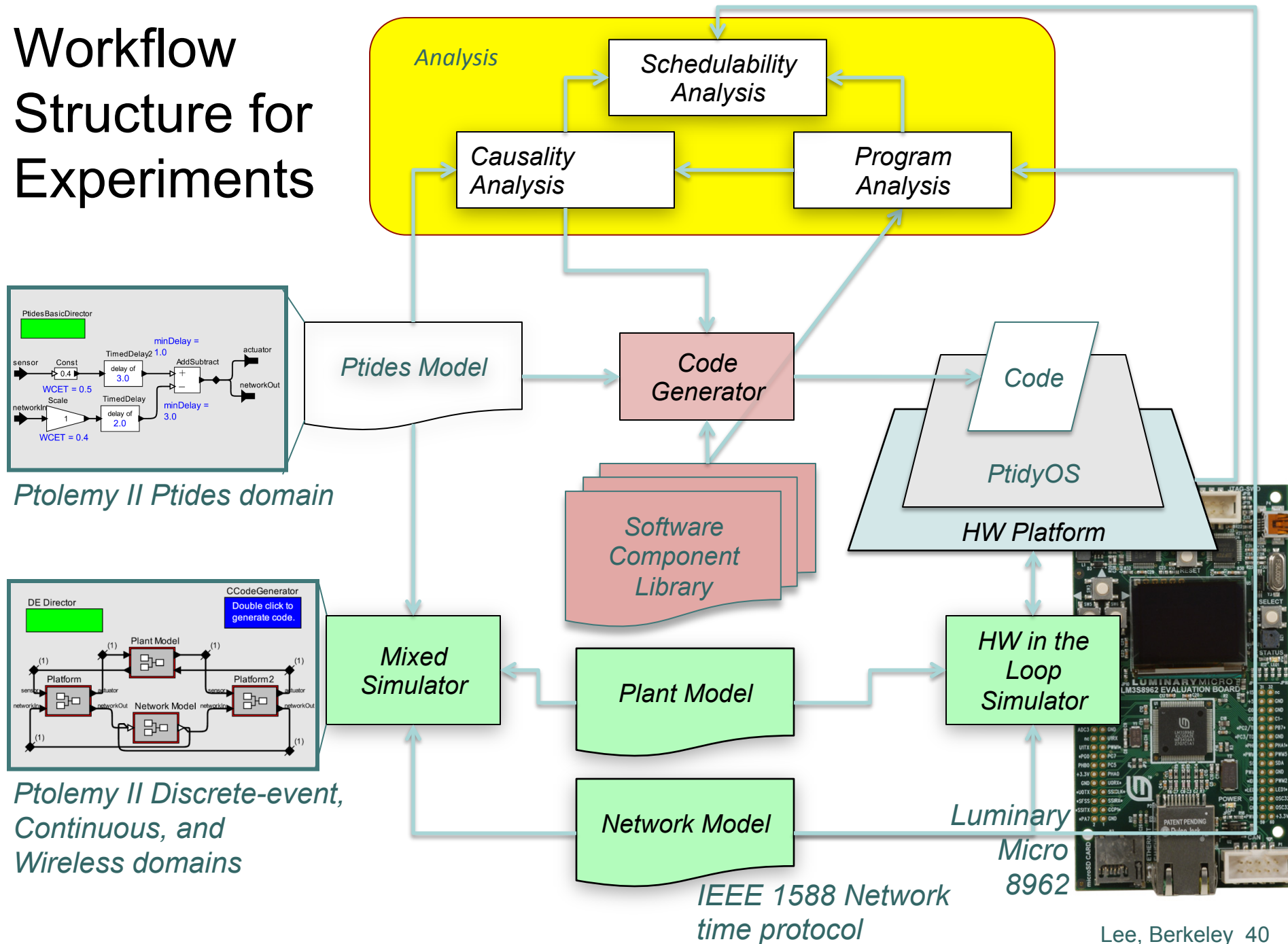
# Ptides Schedulability Analysis

Determine *whether* deadlines can be met

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



# Workflow Structure for Experiments



# Designing & Evaluating PTIDES-based Systems

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

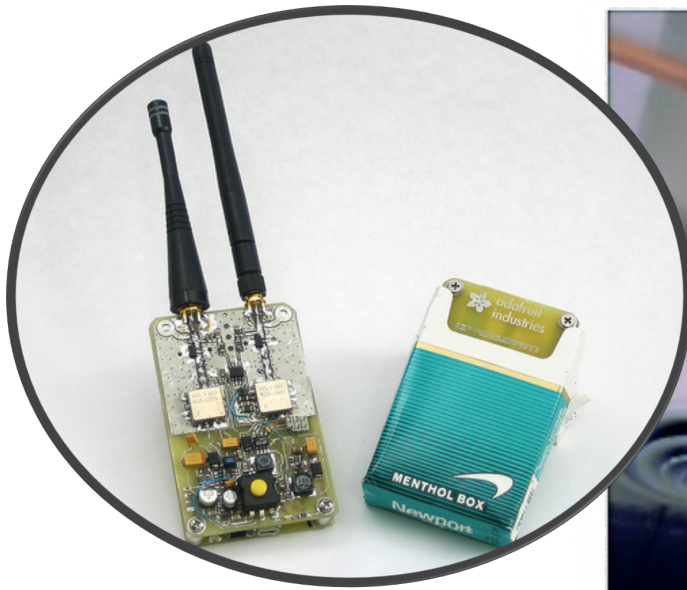
**Conventional approach:** Specify functionality and implementation. Then measure temporal properties.

**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?

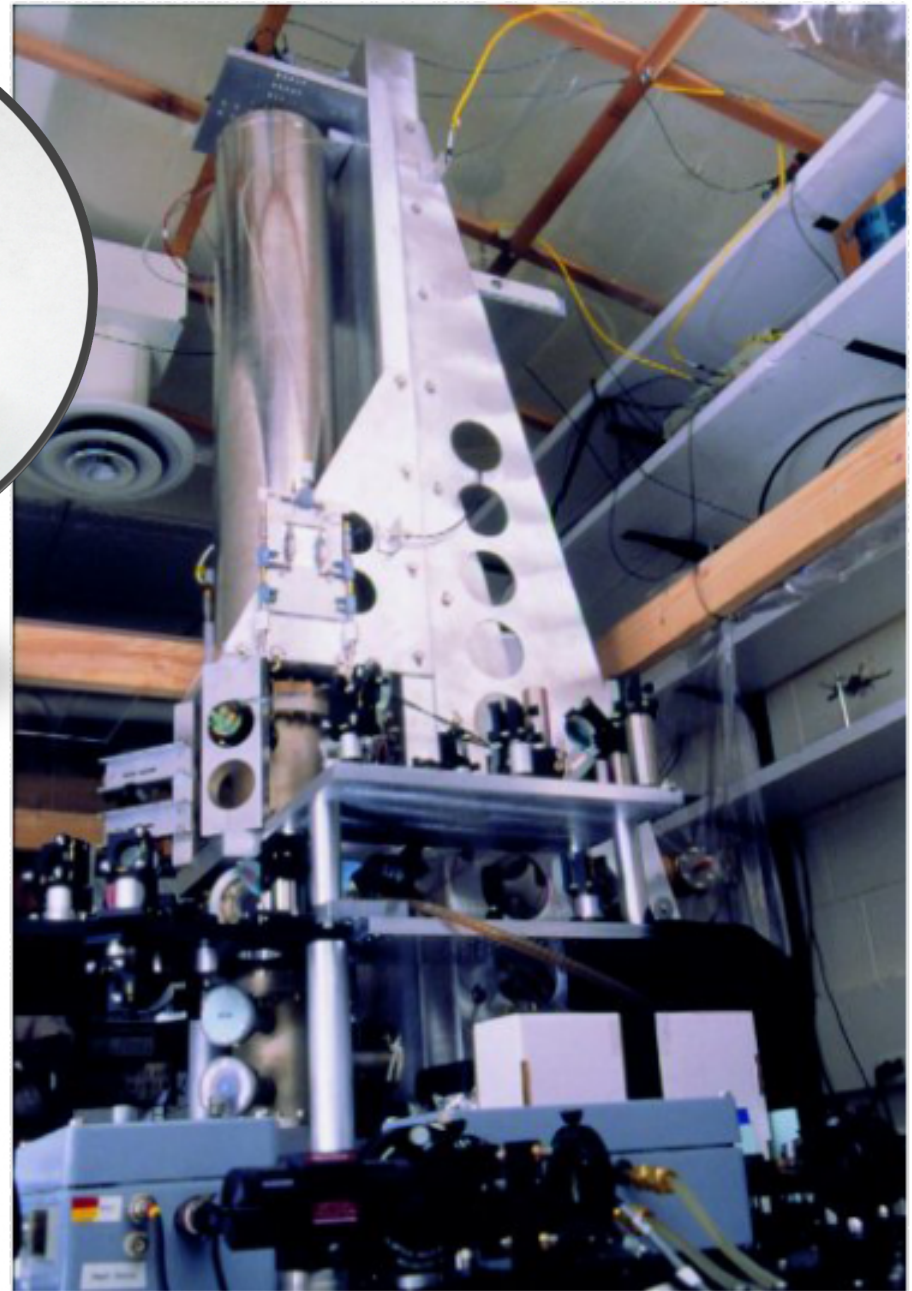
# Security



*GPS Jammer, courtesy of  
Kyle D. Wesson, UT Austin*

With stable local clocks you can:

- Prevent packet losses.
- Detect hardware failures.
- Detect denial of service.
- Detect GPS and PTP spoofing.
- Coordinate w/out communication.



NIST-F1 Atomic Clock

Public Doman Photo

# Synchronized Clocks

In my view, synchronized clocks offer as big a change as:

- Synchronous digital circuit design.
- Imperative programming languages.

both of which are *models* with strong formal properties.

# Modeling vs. Reality

*Solomon Golomb: Mathematical models – Uses and limitations.  
IEEE Transactions on Reliability, 1971*

*You will never strike oil by  
drilling through the map!*



*Solomon Wolf Golomb (1932) mathematician and engineer and a professor of electrical engineering at the University of Southern California. Best known to the general public and fans of mathematical games as the inventor of polyominoes, the inspiration for the computer game Tetris. He has specialized in problems of combinatorial analysis, number theory, coding theory and communications.*

# Don't confuse models with reality!

## *The Kopetz Principle*



*Prof. Dr. Hermann Kopetz*

Many (predictive) properties that we assert about systems (determinism, timeliness, reliability) are in fact not properties of an *implemented* system, but rather properties of a *model* of the system.

We can make definitive statements about *models*, from which we can *infer* properties of system realizations. The validity of this inference depends on *model fidelity*, which is always approximate.

(paraphrased)

*Synchronized clocks enable determinate models for distributed real-time software that have high-fidelity realizations.*

# High-fidelity determinate models for CPS

## *Our Dual Solution:*

1. Regain control over timing at the microarchitecture level (the **PRET** project, Precision Timed Machines).
2. Regain control over timing at the network level (the **PTIDES** project, Programming Timed Distributed Embedded Systems).

- 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 digital systems.

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

Raffaello Sanzio da Urbino – *The Athens School*

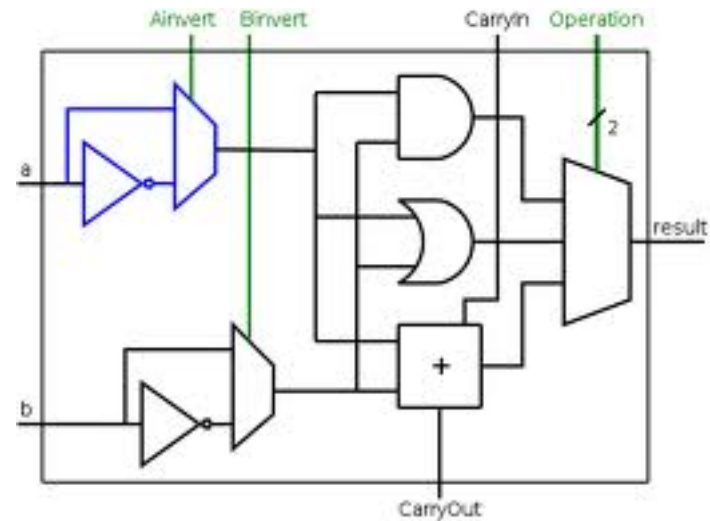


# Determinate Models

Physical System



*Model*



*Synchronous digital logic*

# Determinate Models

## Physical System



## Model

```
/** Reset the output receivers, which are the inside receivers of
 * the output ports of the container.
 * @exception IllegalArgumentException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalArgumentException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    for (IOPort output : outputs) {
        if (_debugging) {
            _debug("Resetting inside receivers of output port: "
                + output.getName());
        }
        Receiver[] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```

*Single-threaded imperative programs*

# Modeling Time

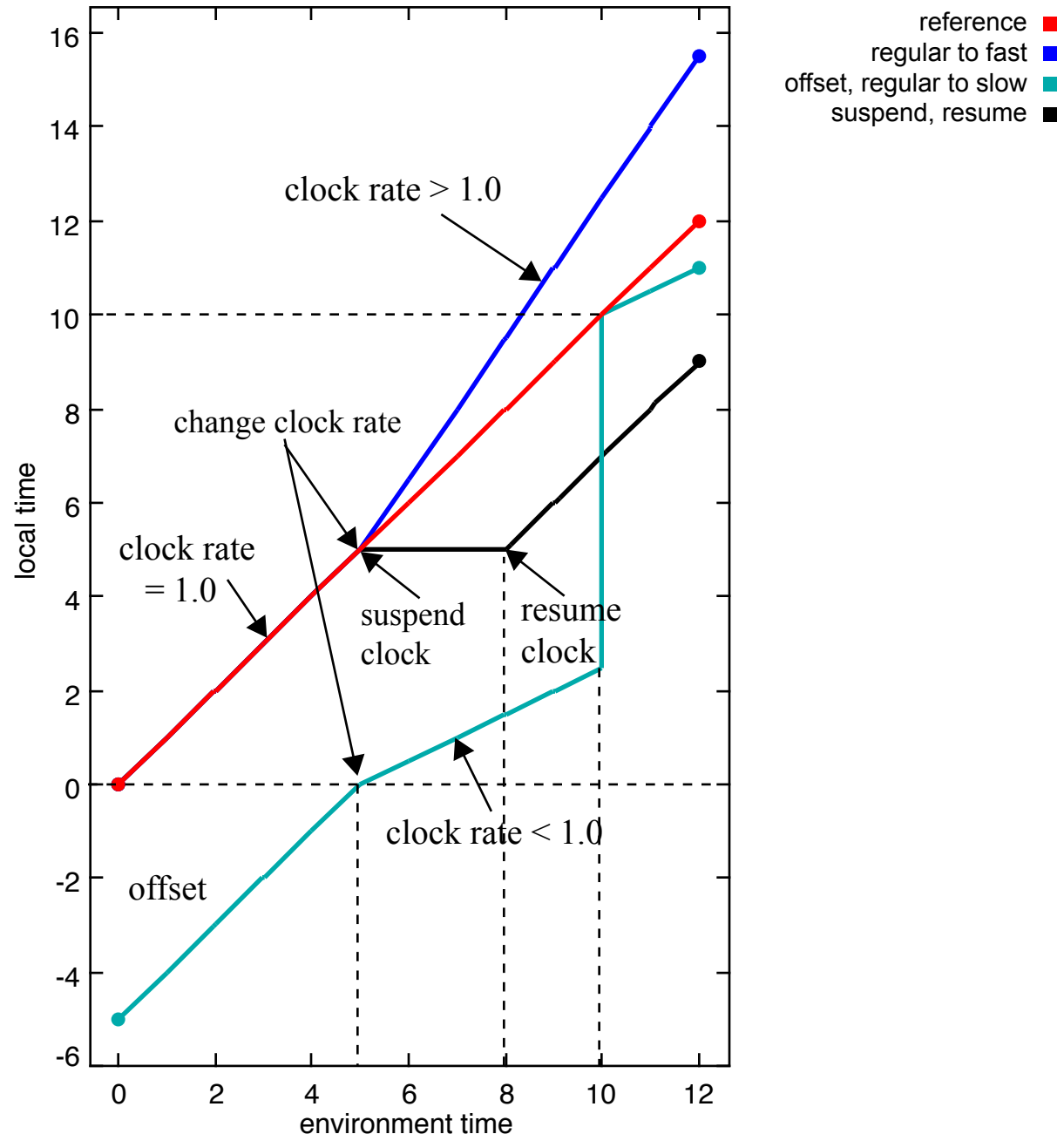
Variable time passage  
across distributed systems.

*Clocks drift*

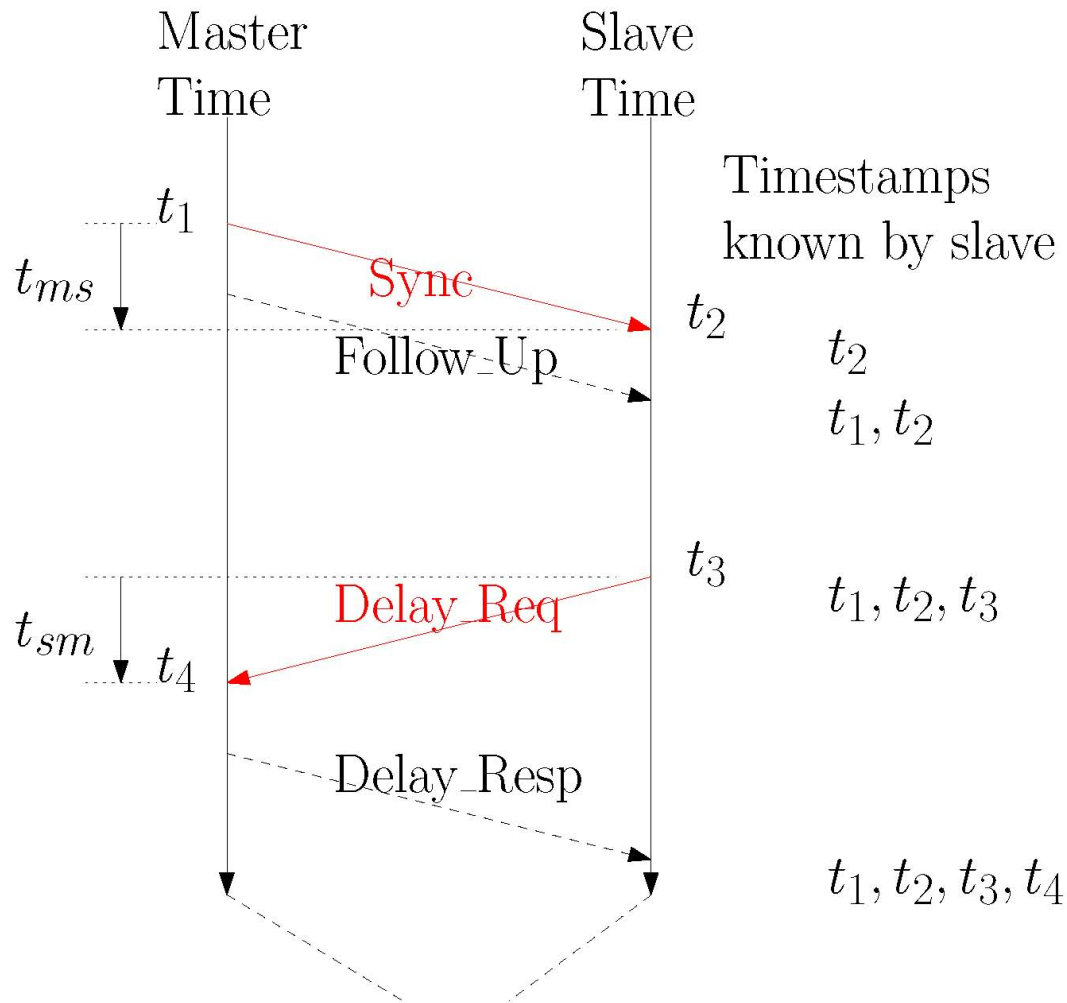
*Clock synchronization*

Next steps:

- notation and theory that enables designers to easily converse about multiform time
- *Multiform notion of time in abstract actor semantics*
- *Test on models of real systems*



# Backup: How PTP Synchronization works



If link is symmetric:

*Offset* =

$$t_{slave} - t_{master} = [(t_2 - t_1) - (t_4 - t_3)]/2 = [t_{ms} - t_{sm}]/2$$

*Propagation time* =

$$[(t_2 - t_1) + (t_4 - t_3)]/2 = [t_{ms} + t_{sm}]/2$$

