Cyber-Physical Systems

A Rehash or A New Intellectual Challenge?

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Talk in the Distinguished Speaker Series
Sponsored by the IEEE Council on Electronic Design Automation (CEDA)
Held at the Design Automation Conference (DAC)

June 4, 2013.
Austin, Texas

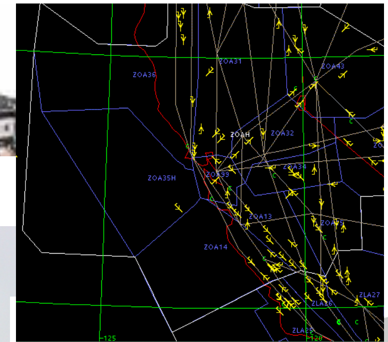
Abstract

The term cyber-physical systems (CPS) refers to the integration of computation and networking with physical processes. CPS is firmly established as a buzzword du jour. Yet many of its elements are familiar and not altogether new. Is CPS just a rehash of old problems designed to attract new funding? In this talk, I will argue that quite to the contrary, CPS is pushing hard at the frontiers of engineering knowledge, putting severe stress on the abstractions and techniques that have proven so effective in the separate spaces of cyber systems (information and computing technology) and physical systems (the rest of engineering). My argument will center on the role of models, and I will show that questions about semantics of models become extremely challenging when the models are required to conjoin the cyber and the physical worlds.

Cyber-Physical Systems (CPS):

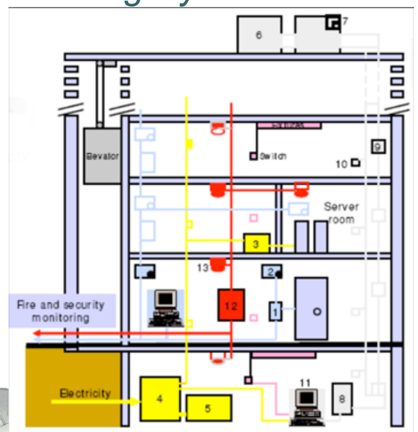
Orchestrating networked computational resources with physical systems

Avionics

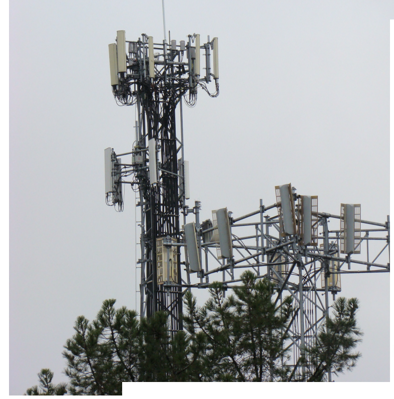


Transportation (Air traffic control at SFO)

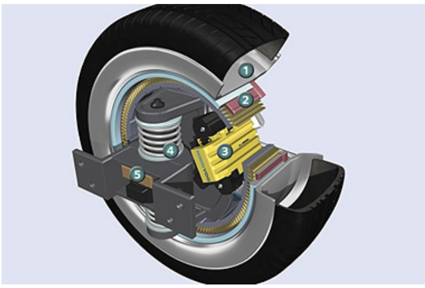
Building Systems



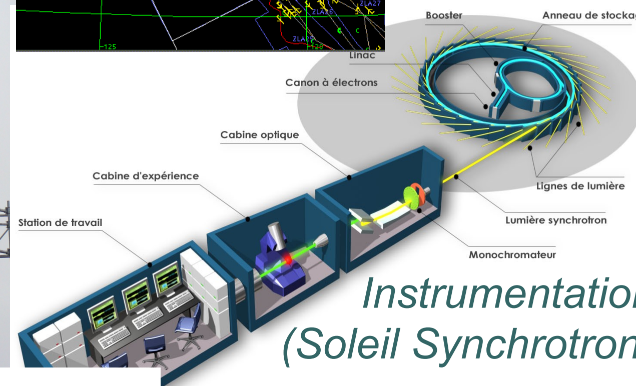
Telecommunications



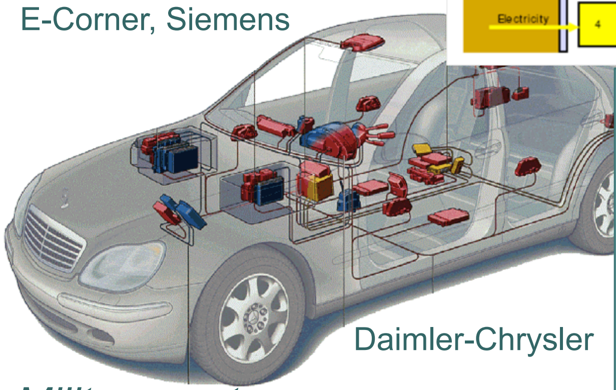
Automotive



E-Corner, Siemens



Instrumentation (Soleil Synchrotron)

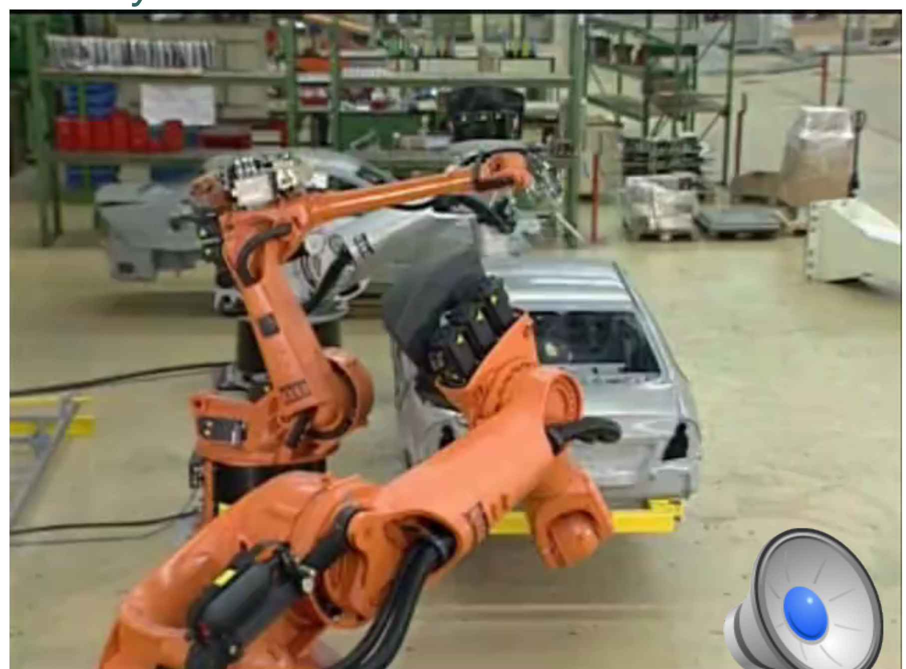


Daimler-Chrysler

Power generation and distribution

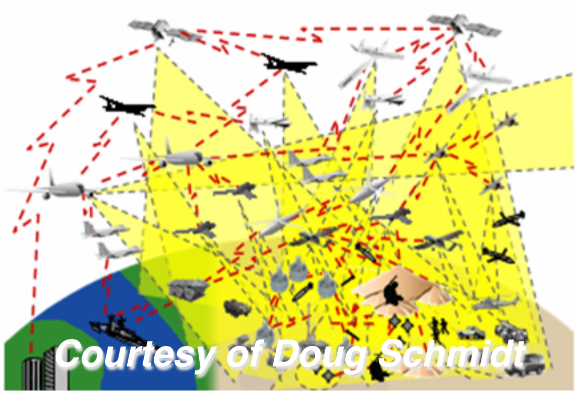
Courtesy of General Electric

Factory automation



Courtesy of Kuka Robotics Corp.

Military systems:



Courtesy of Doug Schmidt

Part 1

Engineering Models for CPS

Models vs. Reality

Solomon Golomb: Mathematical models – Uses and limitations. Aeronautical Journal 1968

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.

But this does not, in any way, diminish the value of a map!

The Kopetz Principle

Prof. Dr. Hermann Kopetz

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

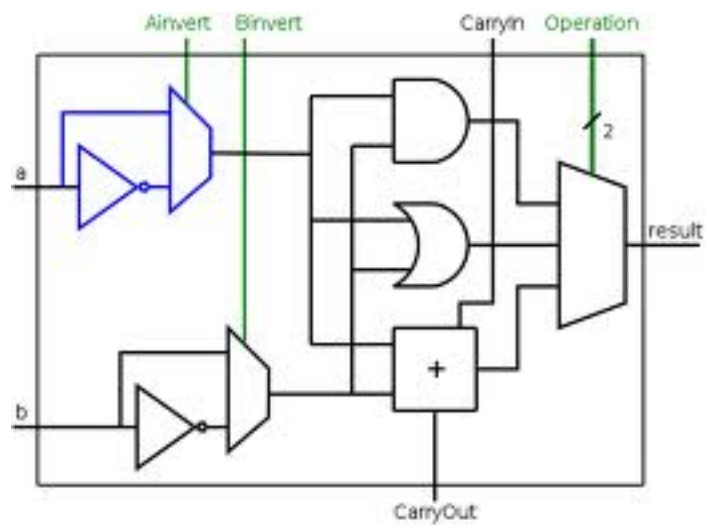
We can make definitive statements about *models*, from which we can *infer* properties of system realizations. The validity of this inference depends on *model fidelity*, which is always approximate.

(paraphrased)

Determinate Models

Physical System

Model



Synchronous digital logic

Determinate Models

Physical System

Model

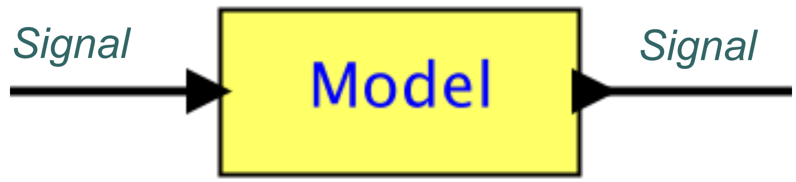

```
/** Reset the output receivers, which are the inside receivers of
 *  the output ports of the container.
 *  @exception IllegalActionException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalActionException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    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

Determinate Models

Physical System

Model

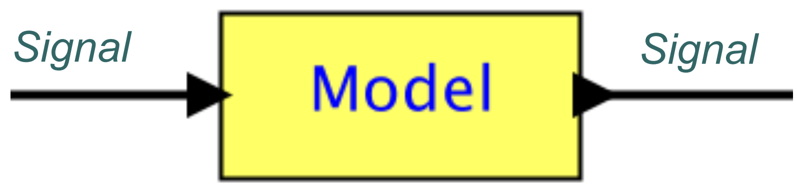


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

Differential Equations

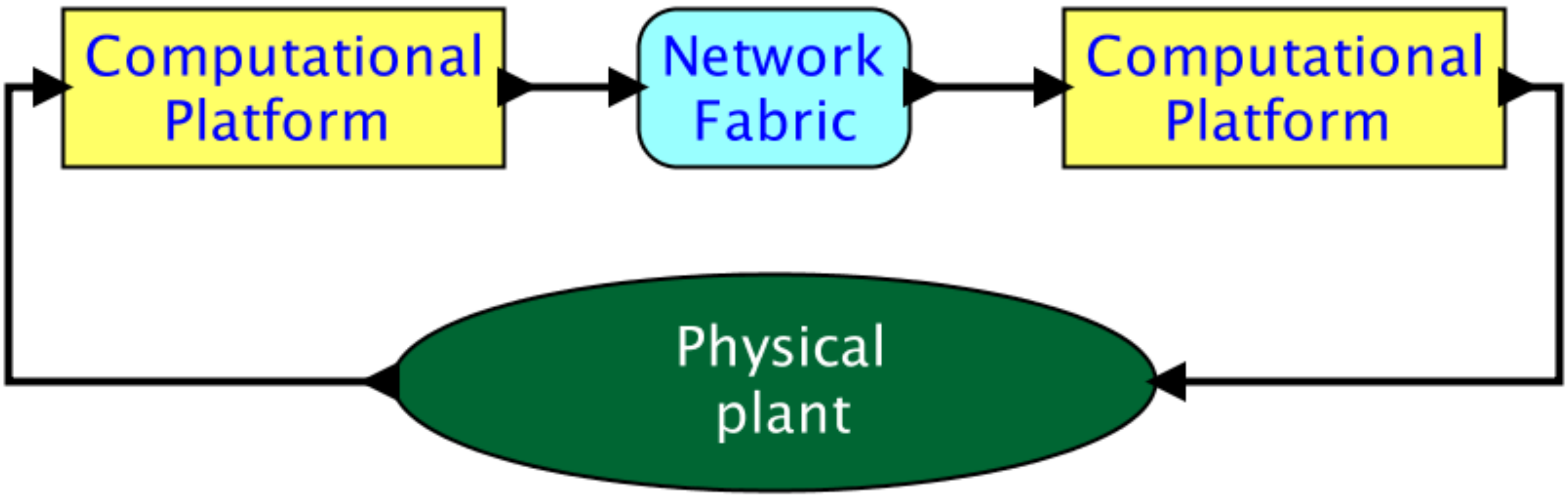
Combinations are Nondeterminate


```java
/** Reset the output receivers, which are the inside receivers of
 *  the output ports of the container.
 *  @exception IllegalActionException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalActionException {
    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();
                        }
                    }
                }
            }
        }
    }
}
```

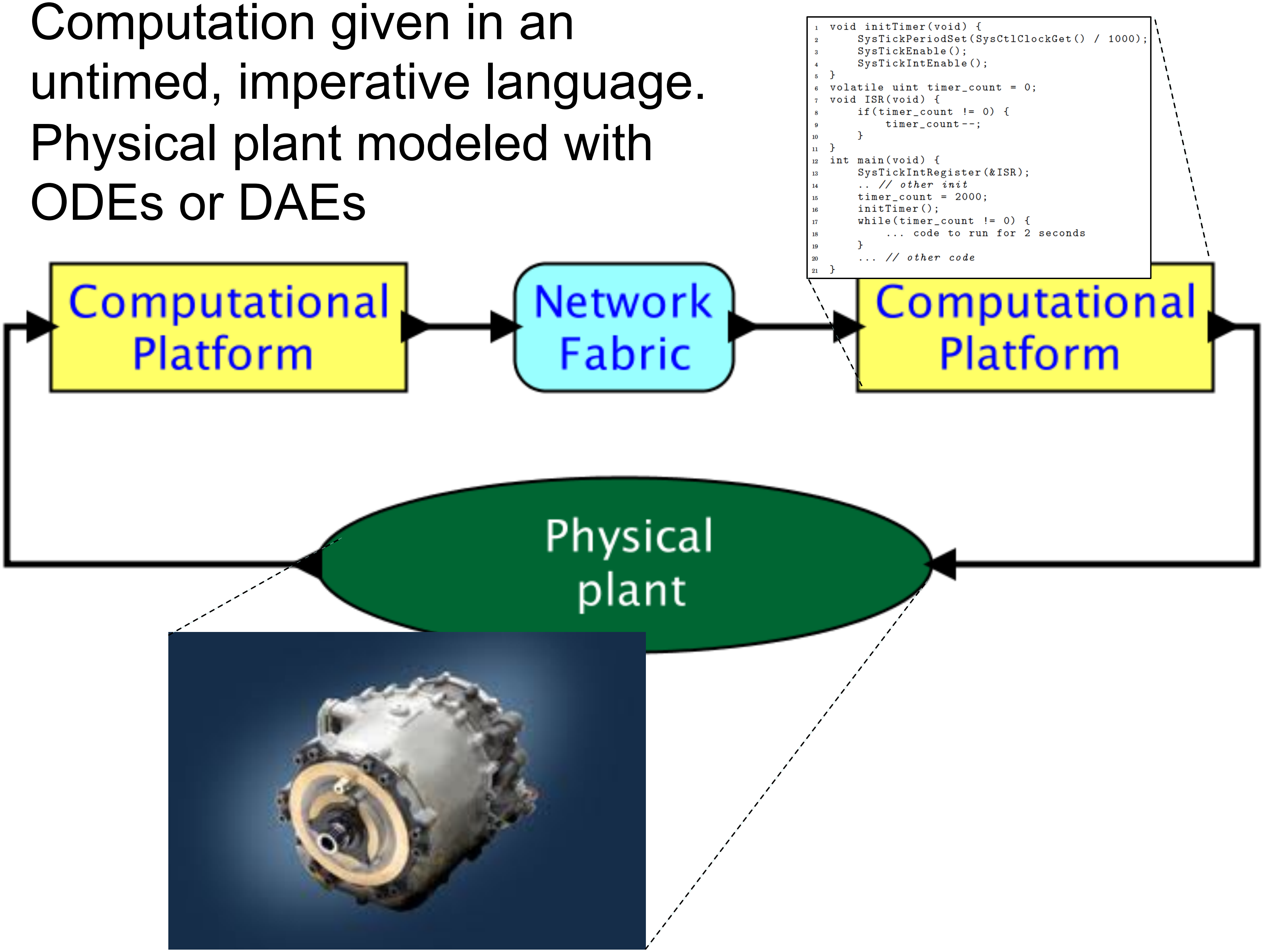


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

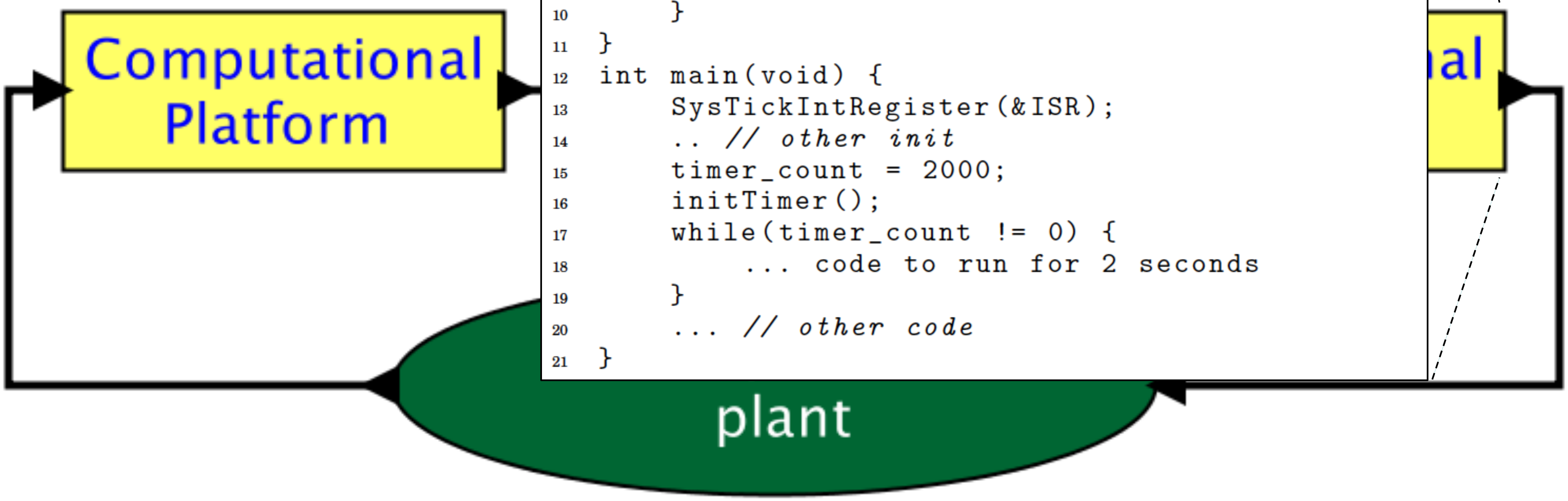
Schematic of a simple CPS:



Computation given in an untimed, imperative language. Physical plant modeled with ODEs or DAEs



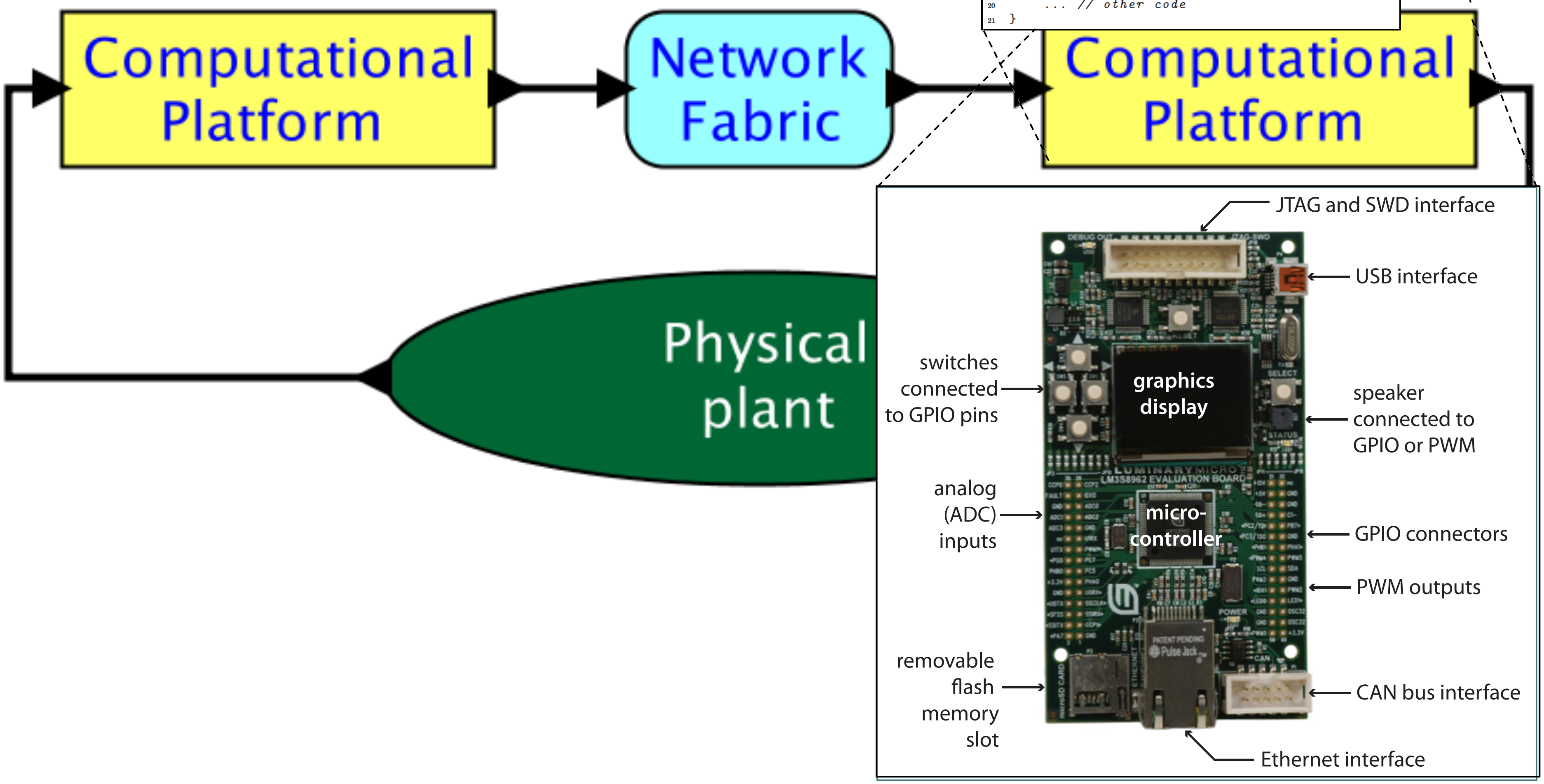
This code is attempting to control timing. But will it really?




```
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
}
```

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
}
```



Consequences

When timing affects system behavior, 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.

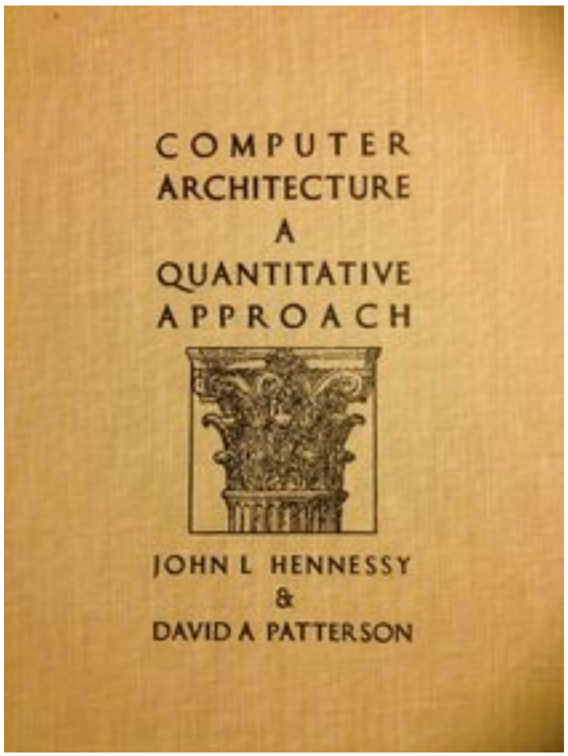
A Key Challenge:
Timing is not Part of Software Semantics

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.

Programmers have no map!

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.)

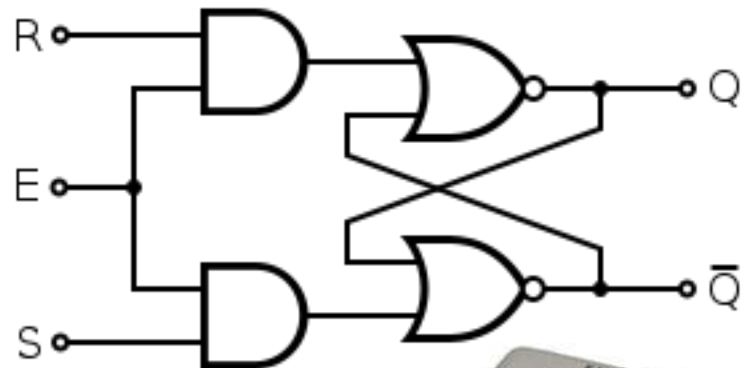
```
1  void foo(int32_t x) {
2          if (x > 1000) {
3                  x = 1000;
4          }
5          if (x > 0) {
6                  x = x + 1000;
7                  if (x < 0) {
8                          panic();
9                  }
10         }
11 }
```

*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]);
}
```

Challenge # 1

Can we change programming models so that a *correct* execution of a program always delivers the same temporal behavior (up to some precision) at the subsystem I/O?

i.e. we need determinate CPS models

Challenge # 2

How can we overcome the powerful inertia created by existing languages, tools, and methodologies to allow innovation that may change key abstractions?

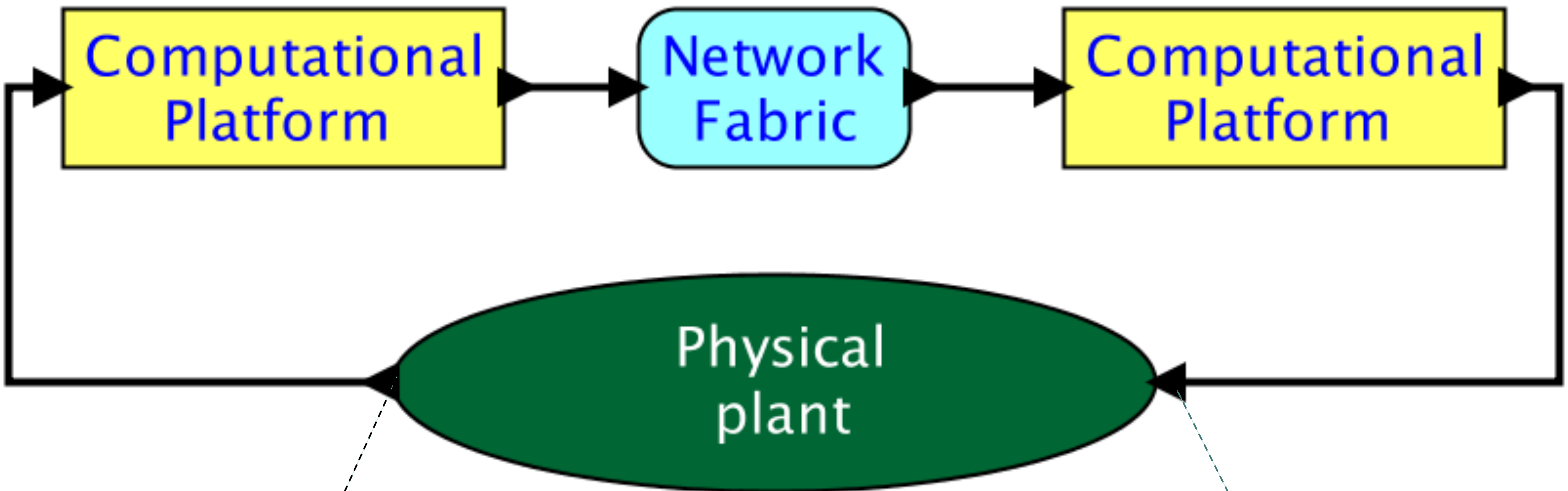
i.e. we need open minds

But Wait… There are Subtleties…

Part 2

Time

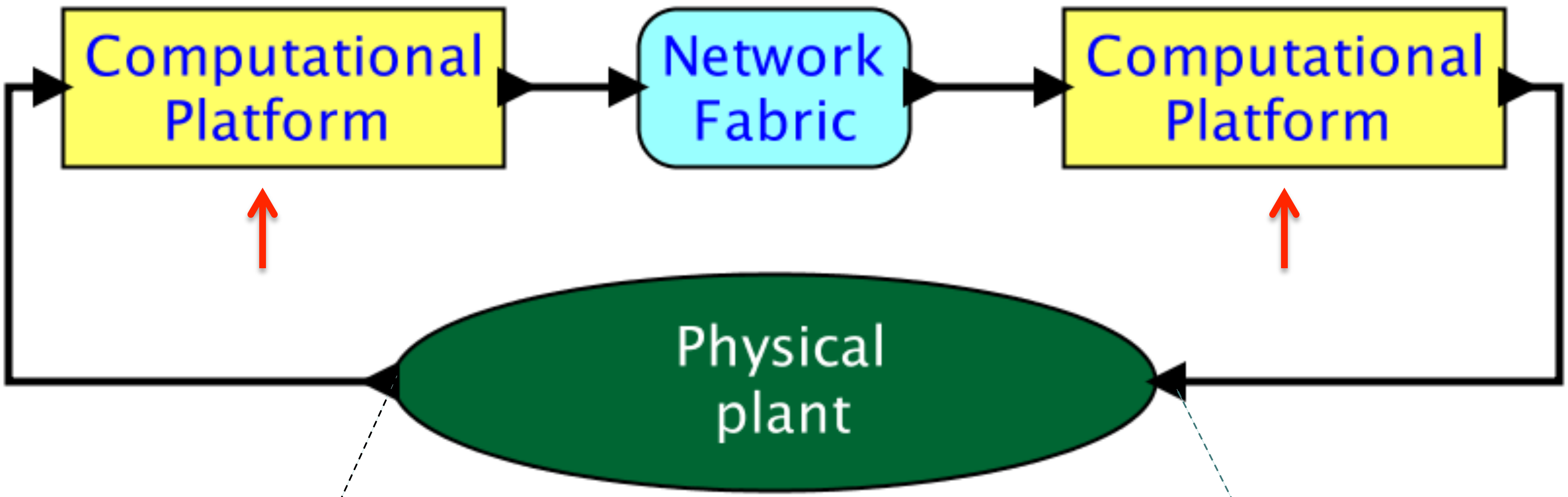
For CPS the very notion of *time* is subtle.



Idealized Newtonian notion of time.

$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_{0}^{t} \mathbf{T}(\tau)\,d\tau$$

Computational platforms have no access to *t*.
Instead, local measurements of time are used.



$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_{0}^{t} \mathbf{T}(\tau)\, d\tau$$

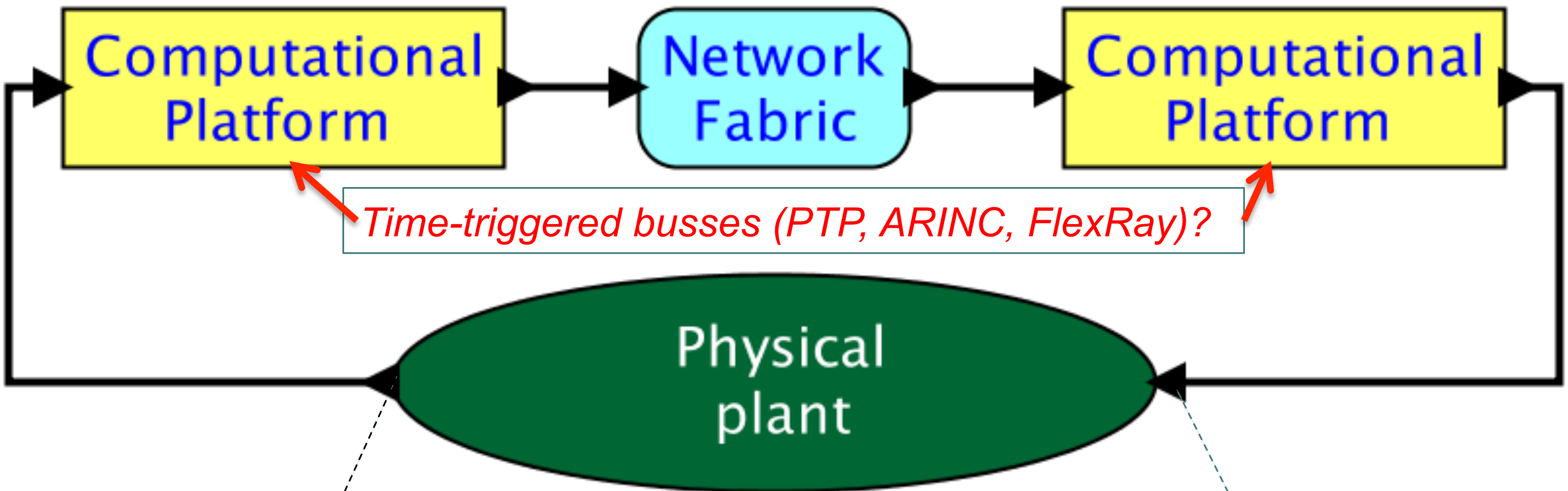
There are many naïve answers out there

E.g.,

$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_0^t \mathbf{T}(\tau)\,d\tau$$

```
double time;
```

Time synchronization? Precision? Representation?
Superdense time? Hyperdense time?
Multiform time? Semantics of simultaneity?



Idealized Newtonian notion of time.

$$\dot{\theta}(t) = \dot{\theta}(0) + \frac{1}{I}\int_{0}^{t} \mathbf{T}(\tau)d\tau$$

Challenge # 3

Can we develop a model of time that is consistent with the realities of time measurement and time synchronization and also with the engineering models used for physical systems?

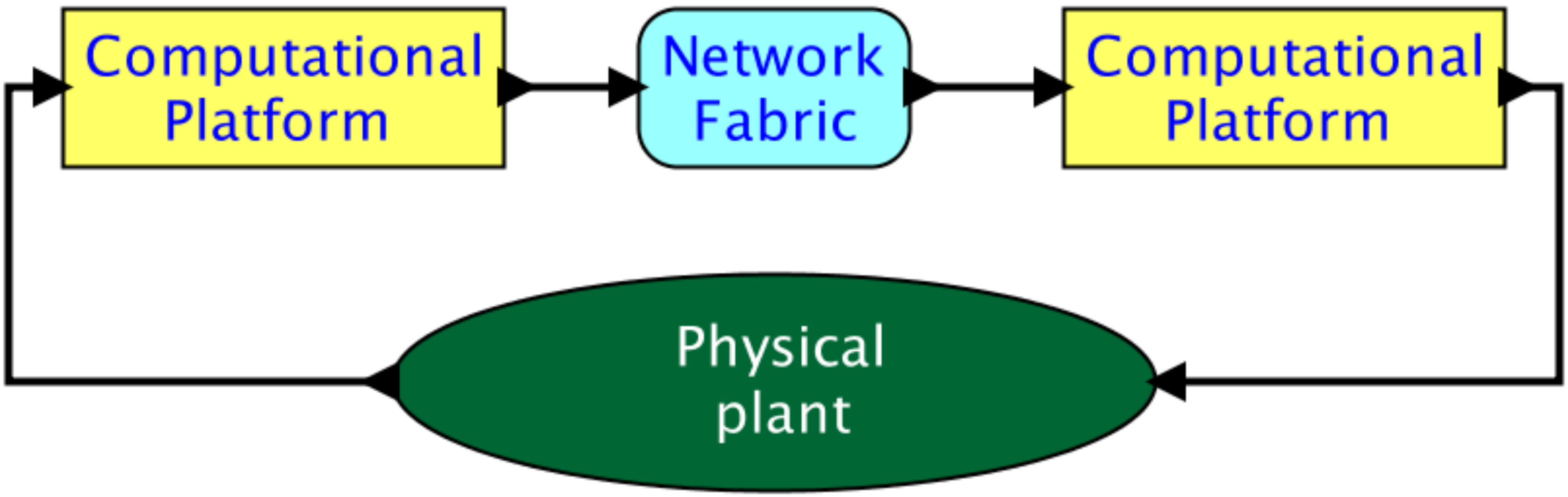
i.e. we need a semantics of time

But Wait… There is more…

Part 3

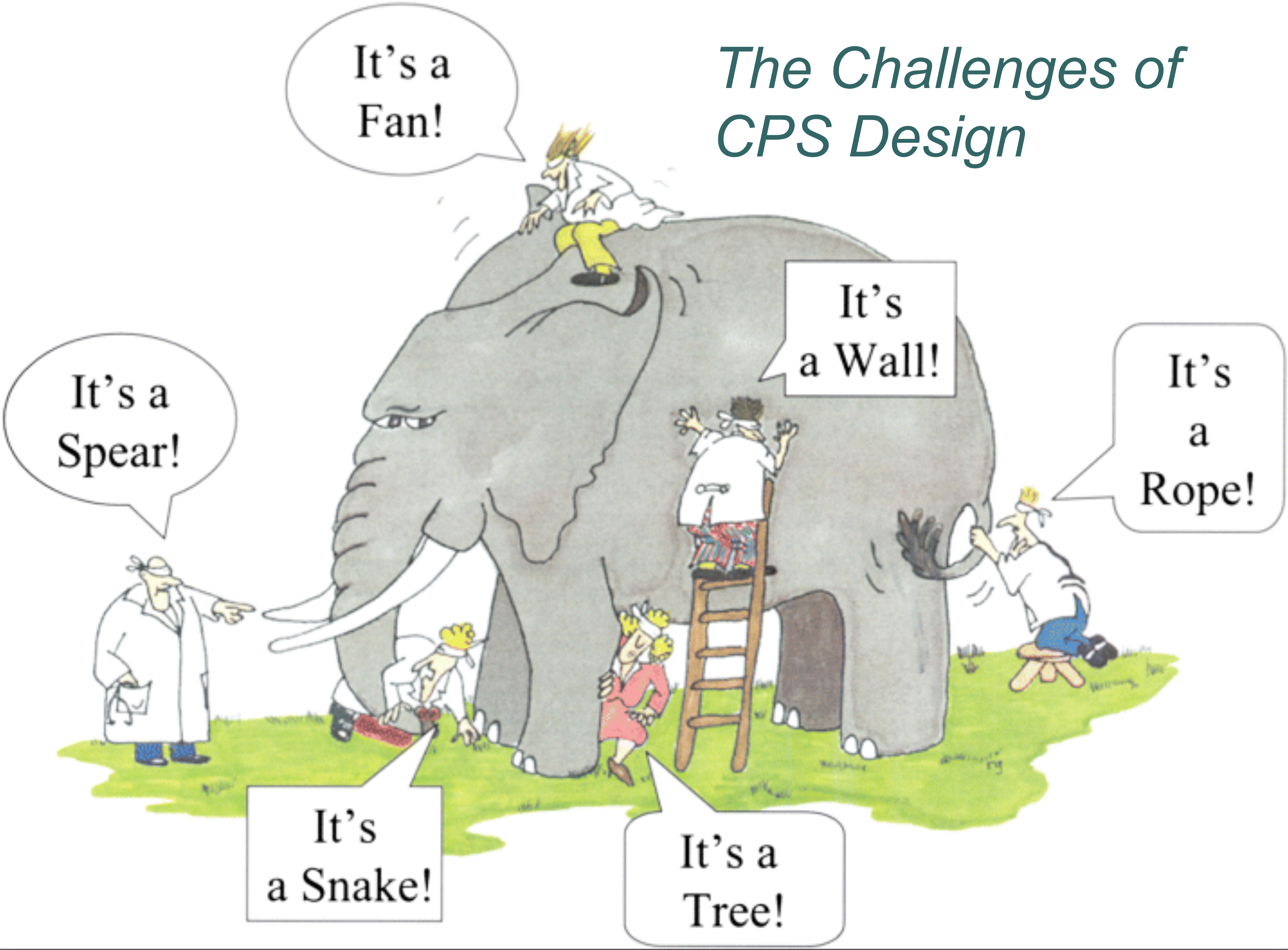
Model Engineering

Engineering Abstractions *and* Engineering Methodology



Components in such a system come from multiple vendors in diverse engineering disciplines with distinct domain expertise.

The Challenges of CPS Design



E.g. Electric Power Systems (EPS) for Aircraft

Physically:

- Generators
- Contactors
- Busses
- Loads

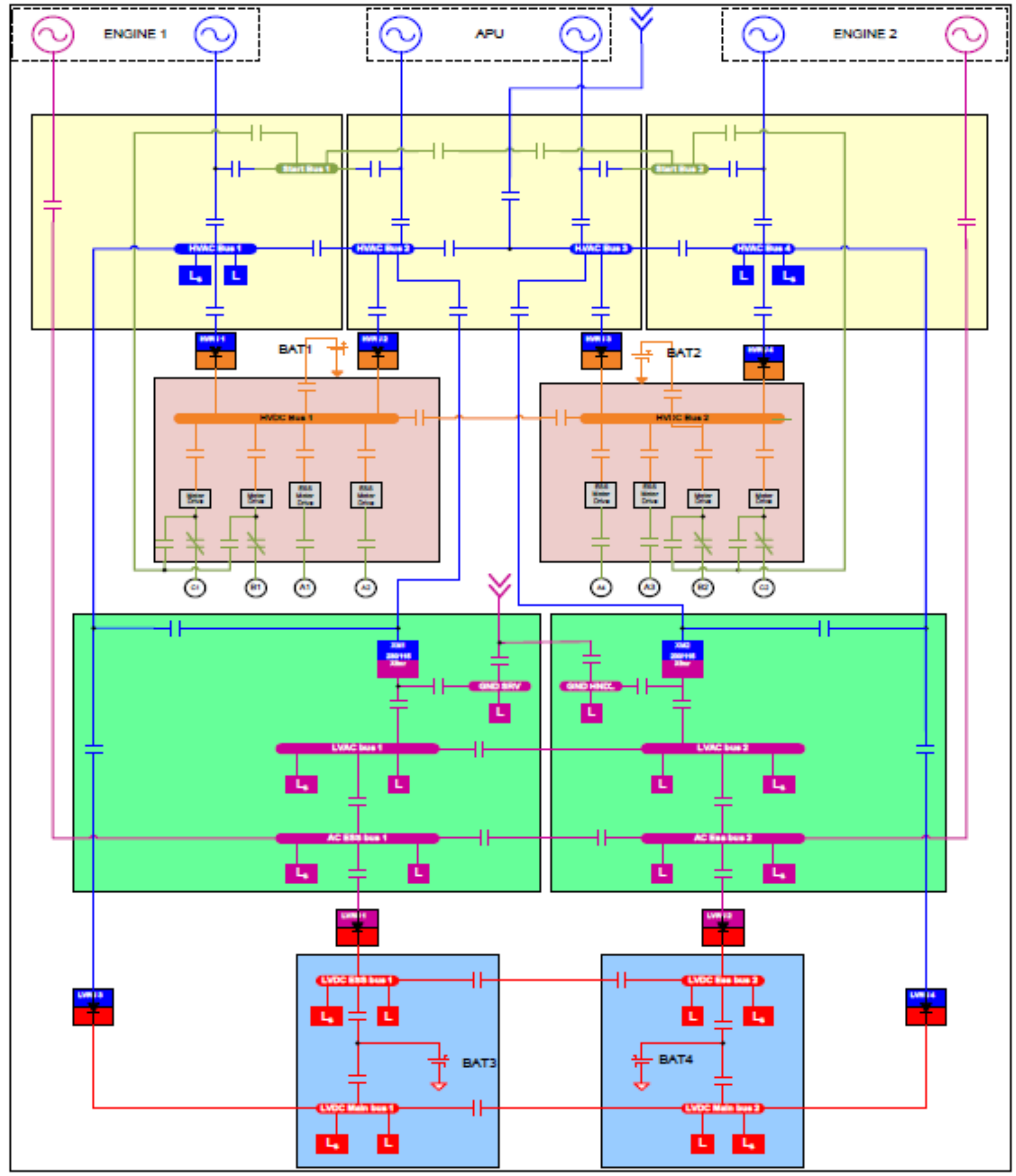
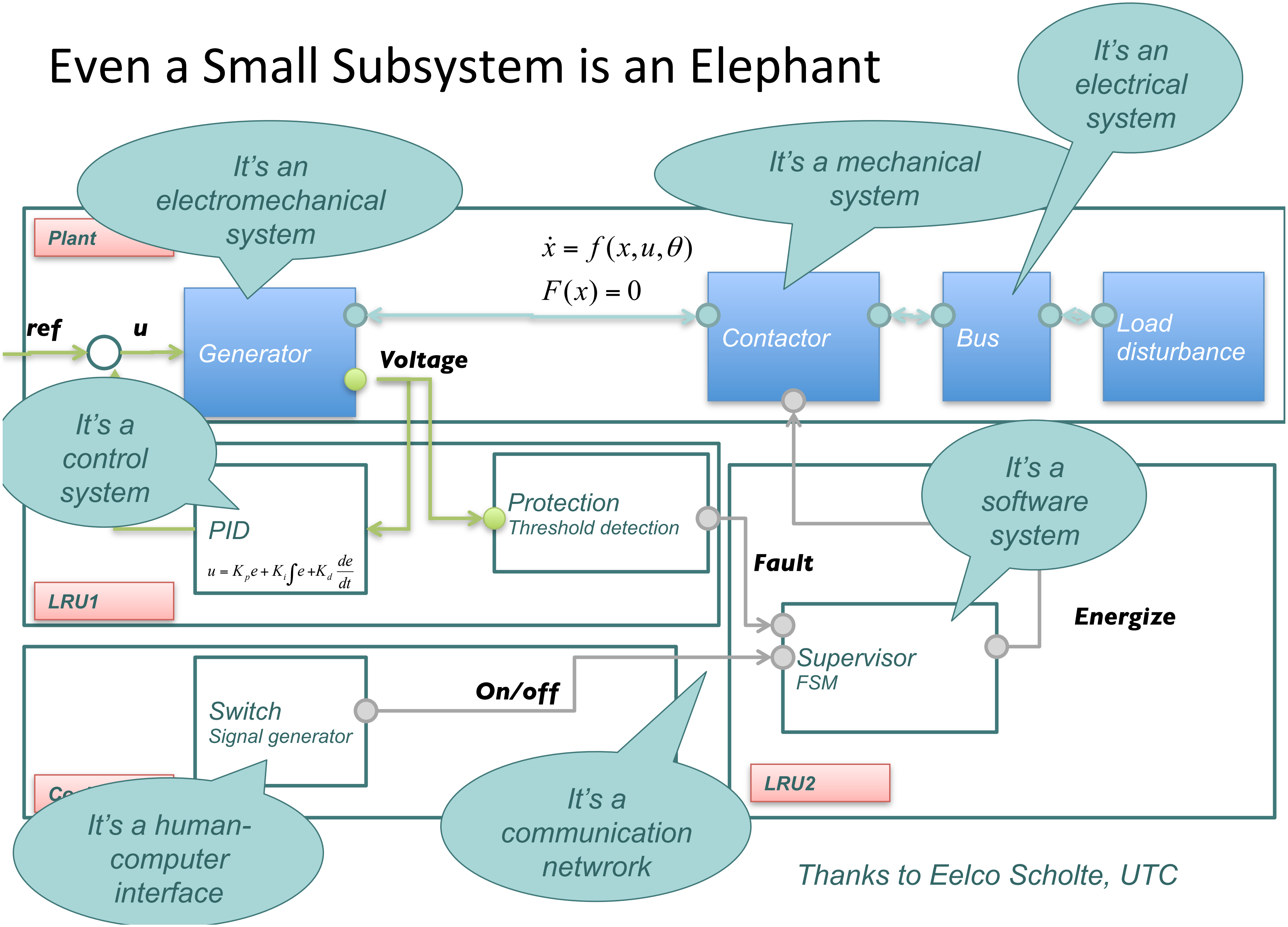


Figure 1: Single line diagram of an electric power system adapted from Honeywell Patent US 7,439,634 B2. Figure courtesy of Rich Poisson, Hamilton-Sundstrand.

Even a Small Subsystem is an Elephant



Thanks to Eelco Scholte, UTC

Challenge # 4

How can we define interfaces between components that bridge engineering disciplines and clarify requirements and expectations?

i.e. we need a discipline of "model engineering"

Part 4

Some Promising Approaches

Some Promising Approaches

- PRET machines
- Network time synchronization (PTP, IEEE 1588)
- Model-based design
  - with timed, determinate, concurrent MoCs, like PTIDES
- Temporal logics
  - with methods for synthesis and verification.
- Platform-based design
- Contract-based design
- Abstract semantics
  - for heterogeneous modeling
- Co-simulation and model-exchange standards
  - particularly FMI
- Aspect-oriented modeling

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

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

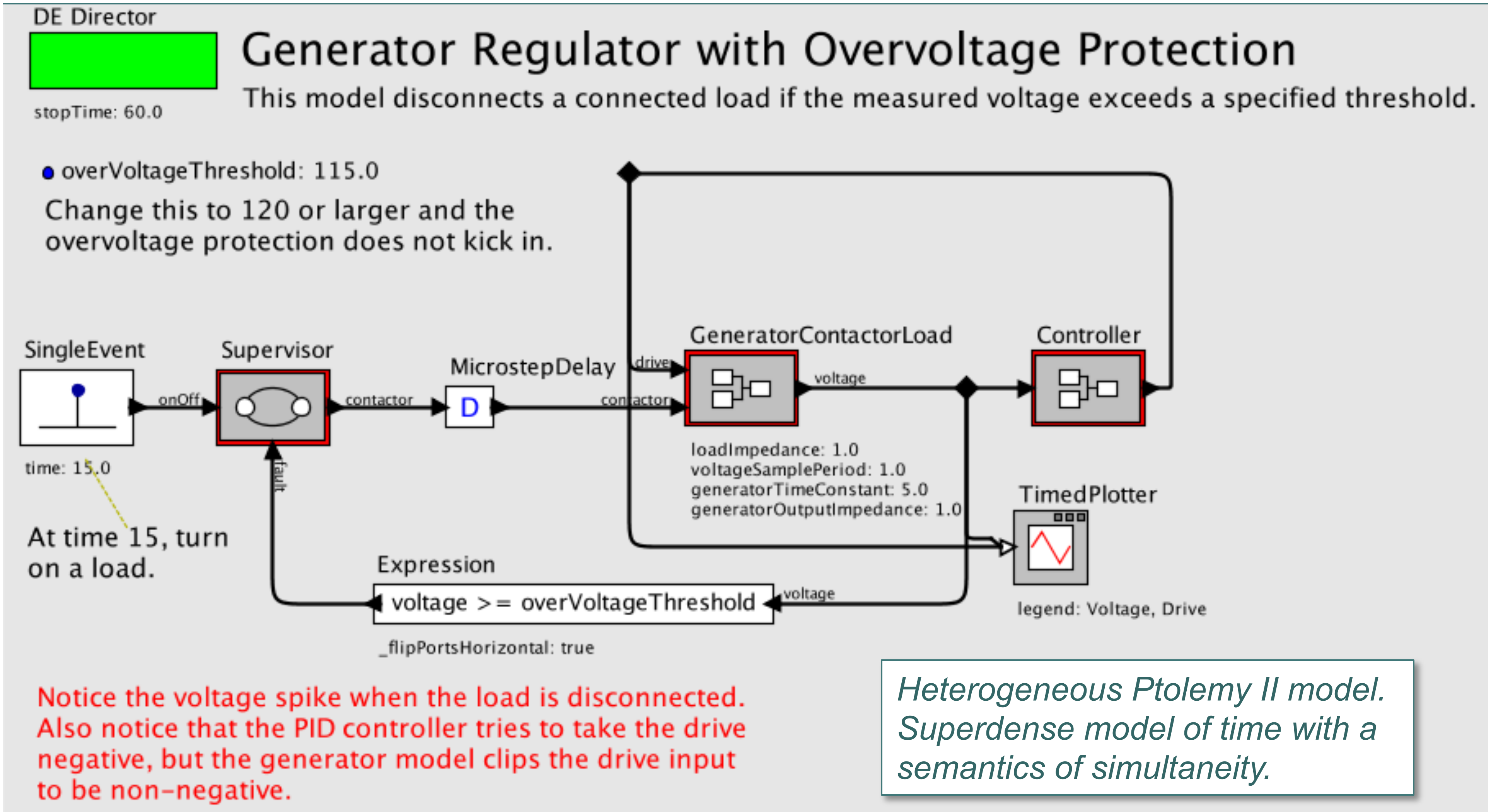
Computing

With time

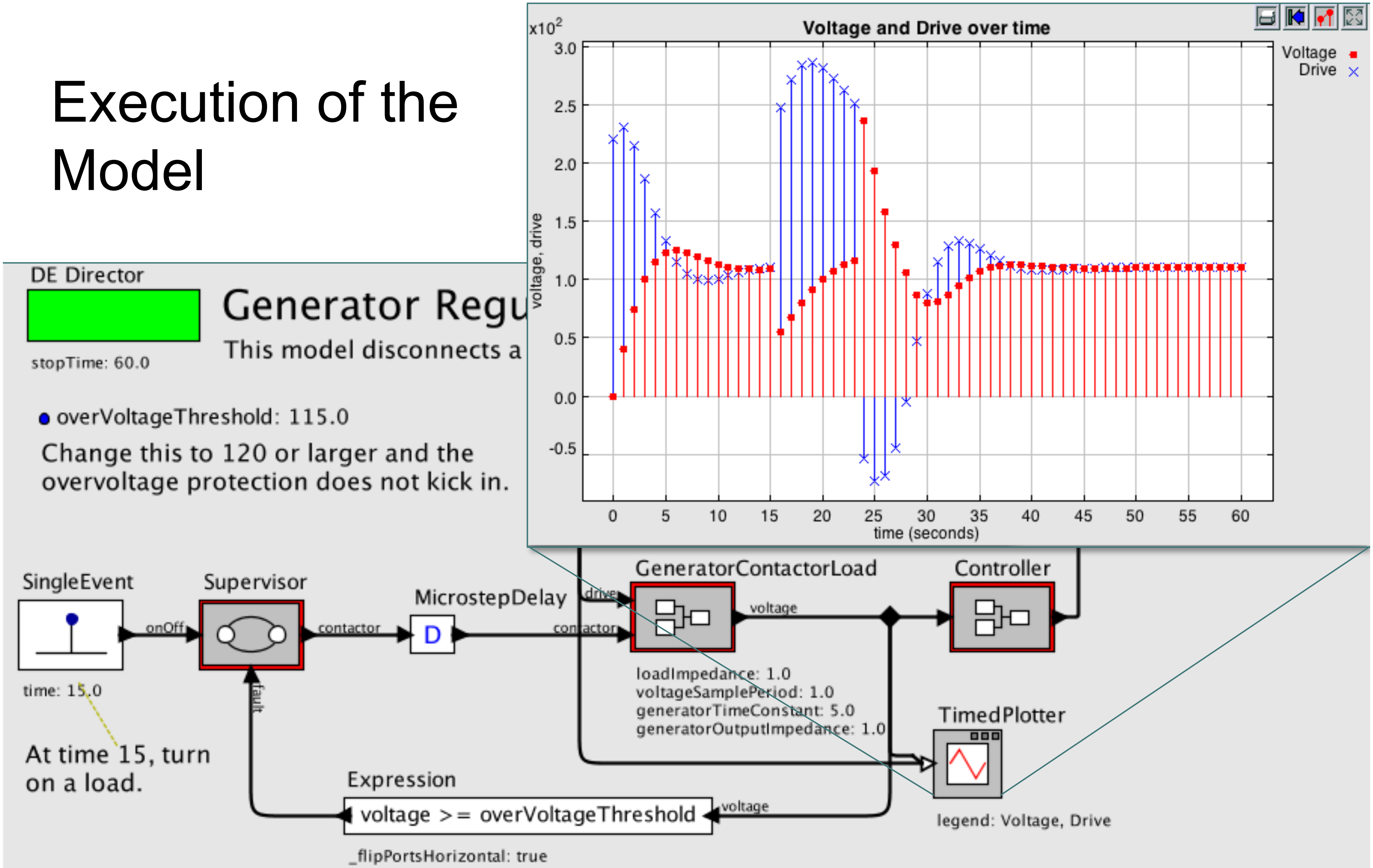
Some Promising Approaches

- PRET machines
- Network time synchronization (PTP, IEEE 1588)
- Model-based design
  - with timed, determinate, concurrent MoCs, like PTIDES
- Temporal logics
  - with methods for synthesis and verification.
- Platform-based design
- Contract-based design
- Abstract semantics
  - for heterogeneous modeling
- Co-simulation and model-exchange standards
  - particularly FMI
- Aspect-oriented modeling

Heterogeneous Models: Discrete-Event (DE) Model of a Generator-Regulator-Protector

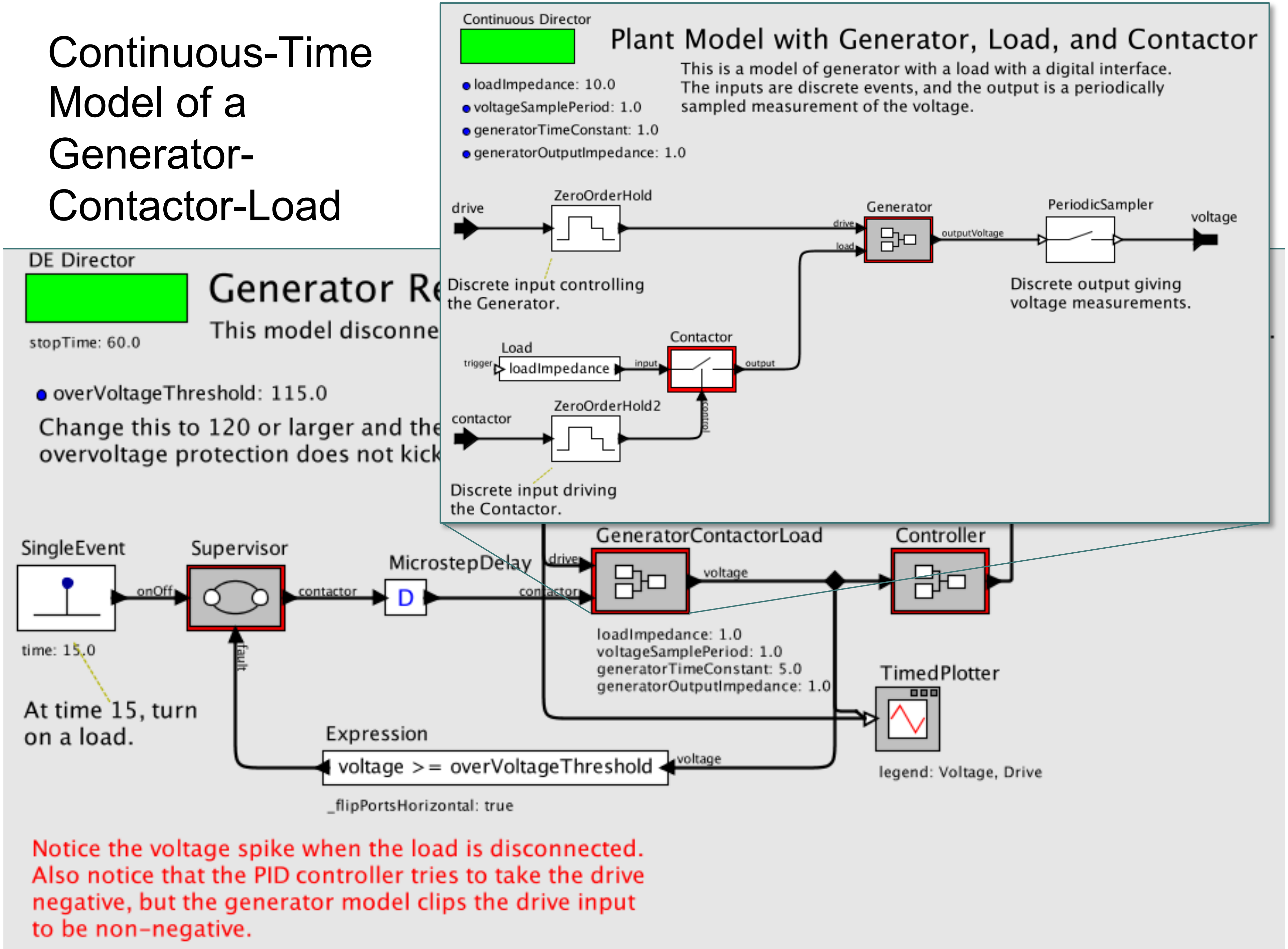


Execution of the Model

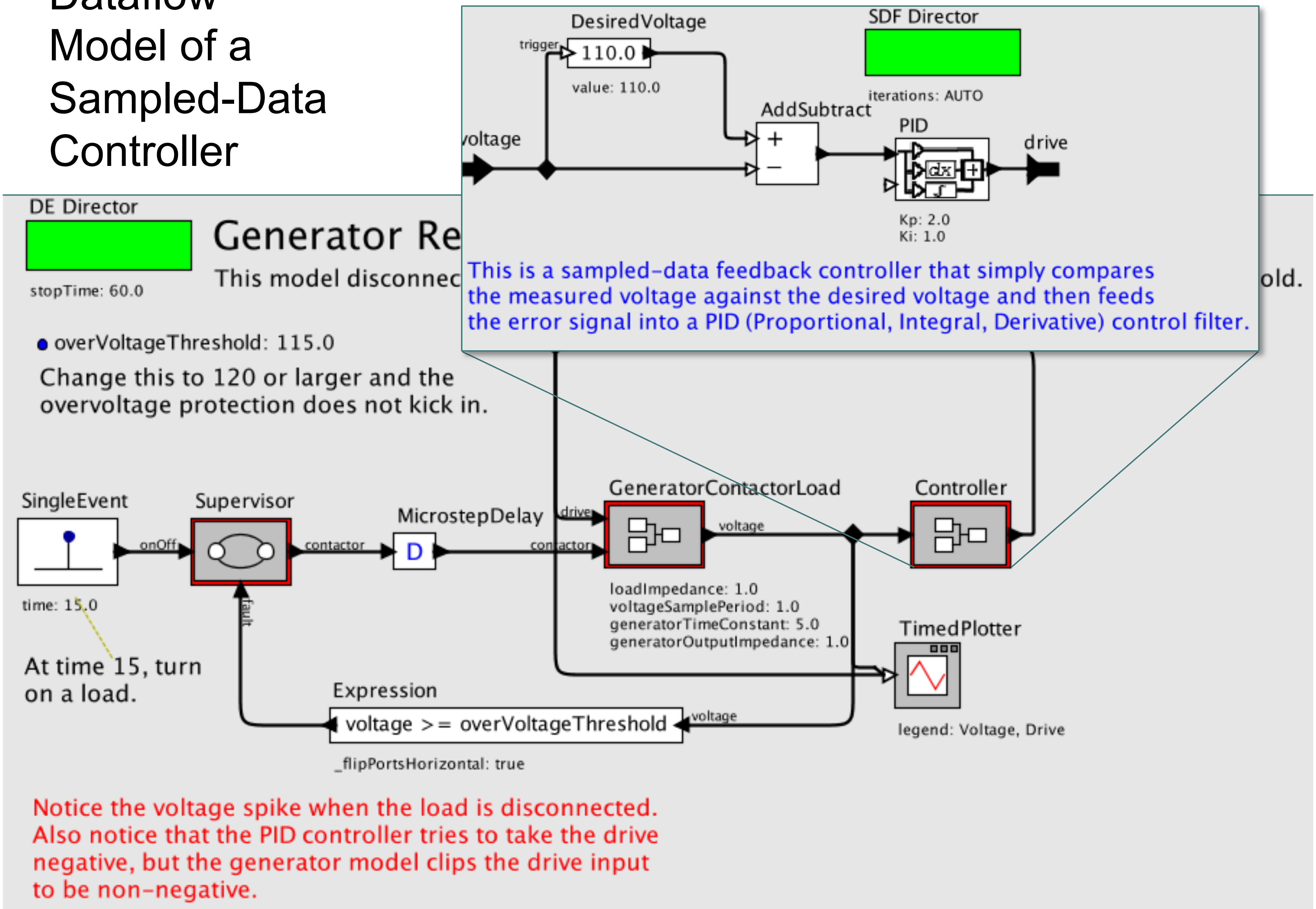


Notice the voltage spike when the load is disconnected. Also notice that the PID controller tries to take the drive negative, but the generator model clips the drive input to be non-negative.

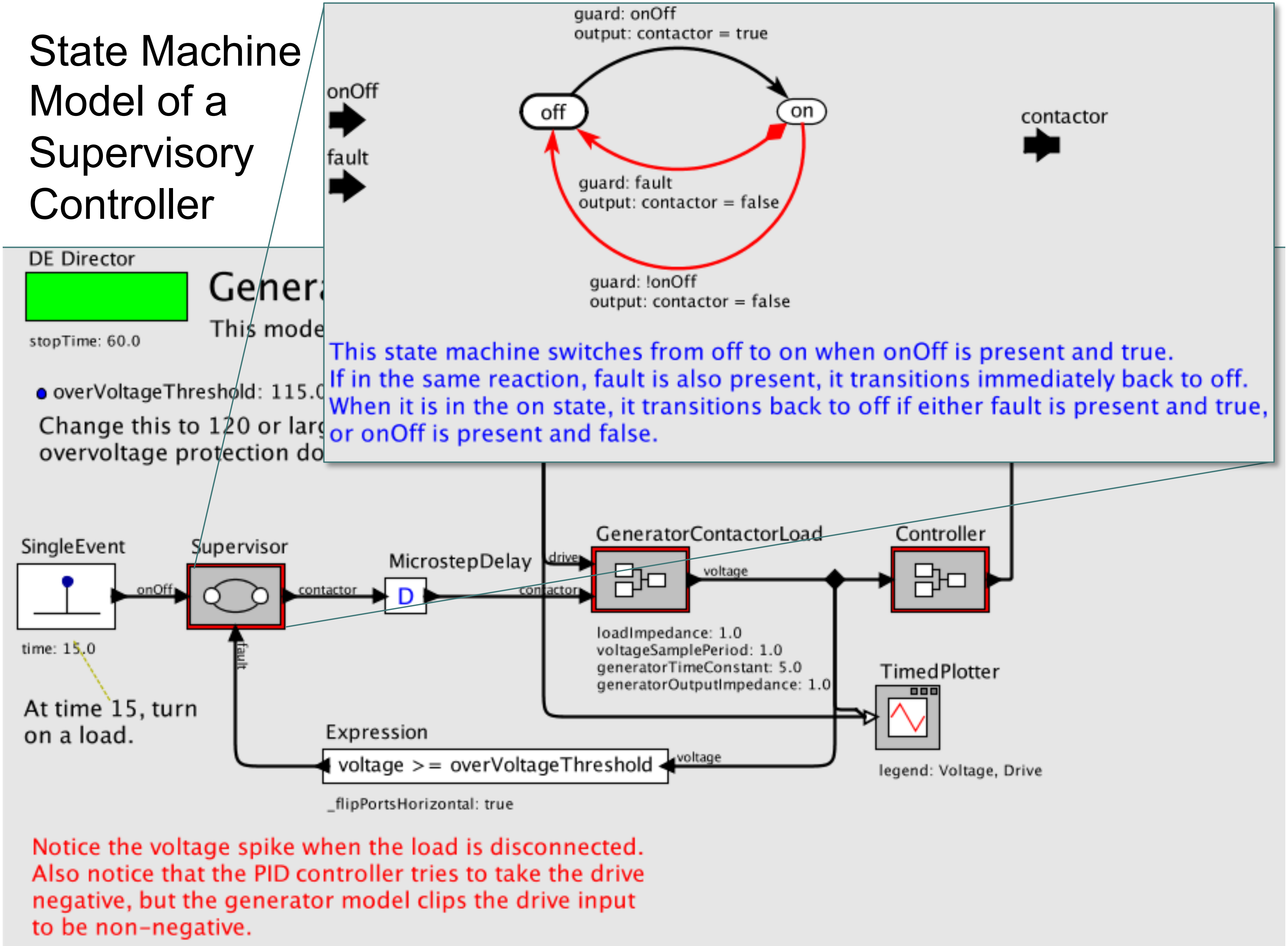
Continuous-Time Model of a Generator-Contactor-Load



Dataflow Model of a Sampled-Data Controller



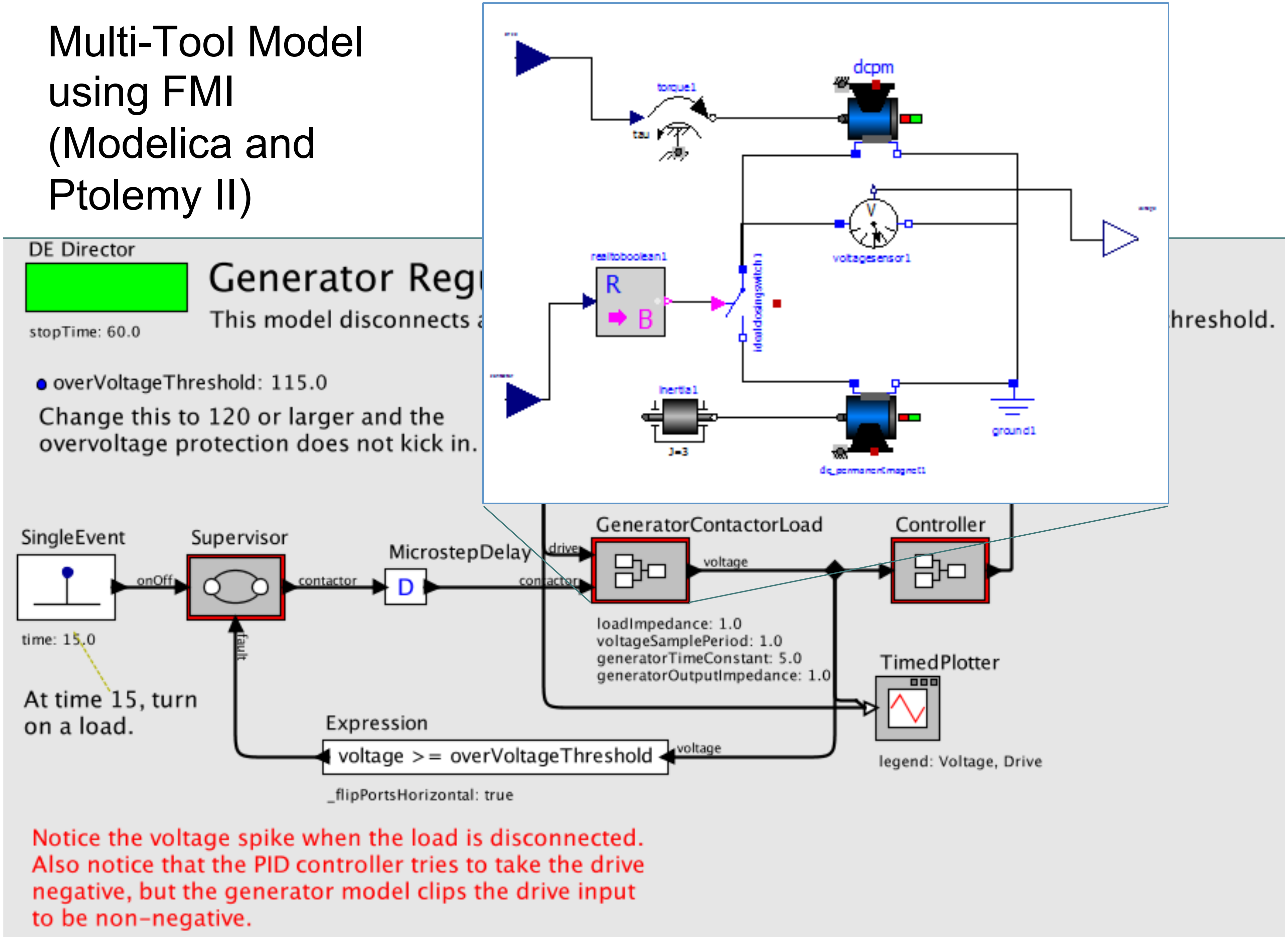
State Machine Model of a Supervisory Controller



Some Promising Approaches

- PRET machines
- Network time synchronization (PTP, IEEE 1588)
- Model-based design
  - with timed, determinate, concurrent MoCs, like PTIDES
- Temporal logics
  - with methods for synthesis and verification.
- Platform-based design
- Contract-based design
- Abstract semantics
  - for heterogeneous modeling
- Co-simulation and model-exchange standards
  - particularly FMI
- Aspect-oriented modeling

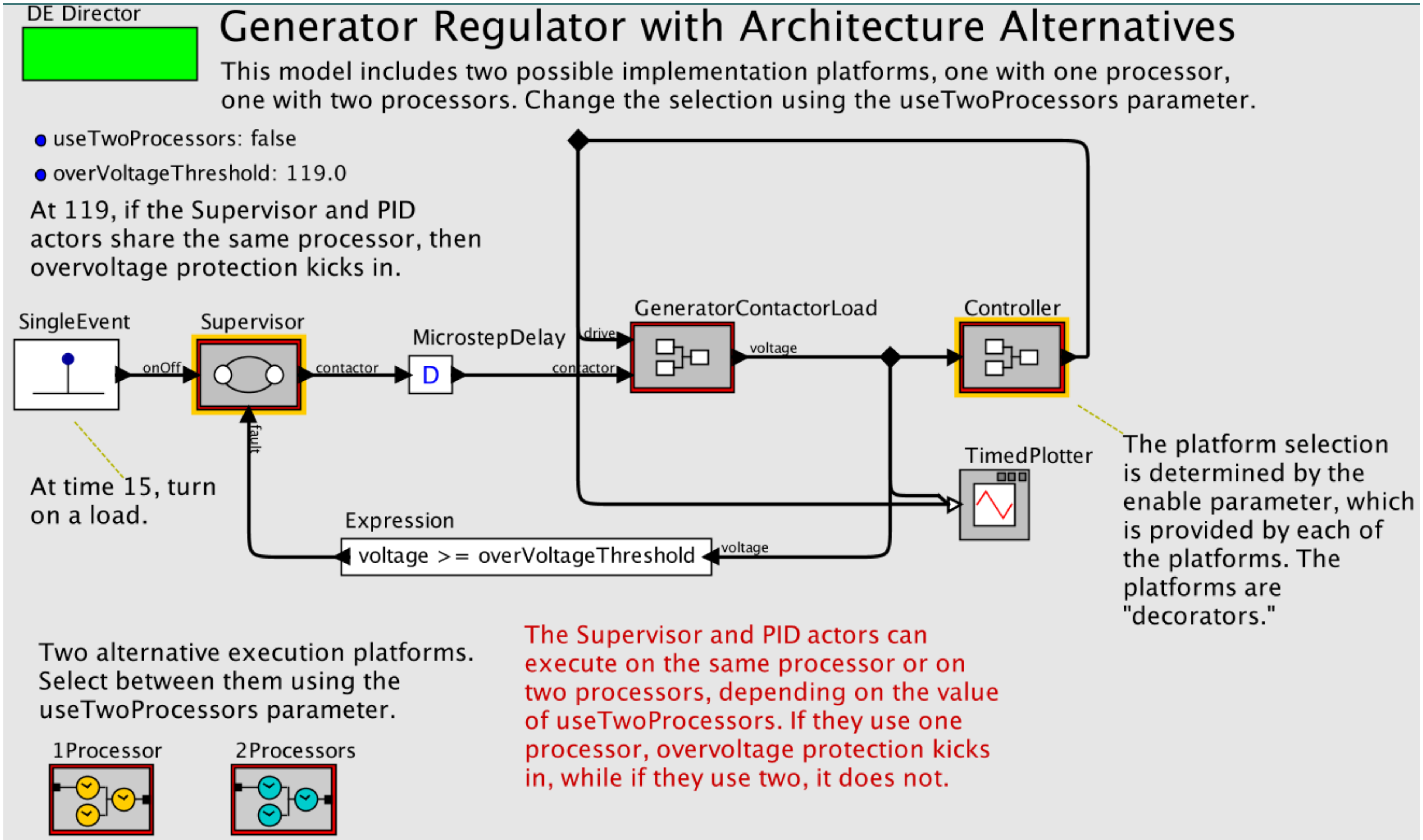
Multi-Tool Model using FMI (Modelica and Ptolemy II)



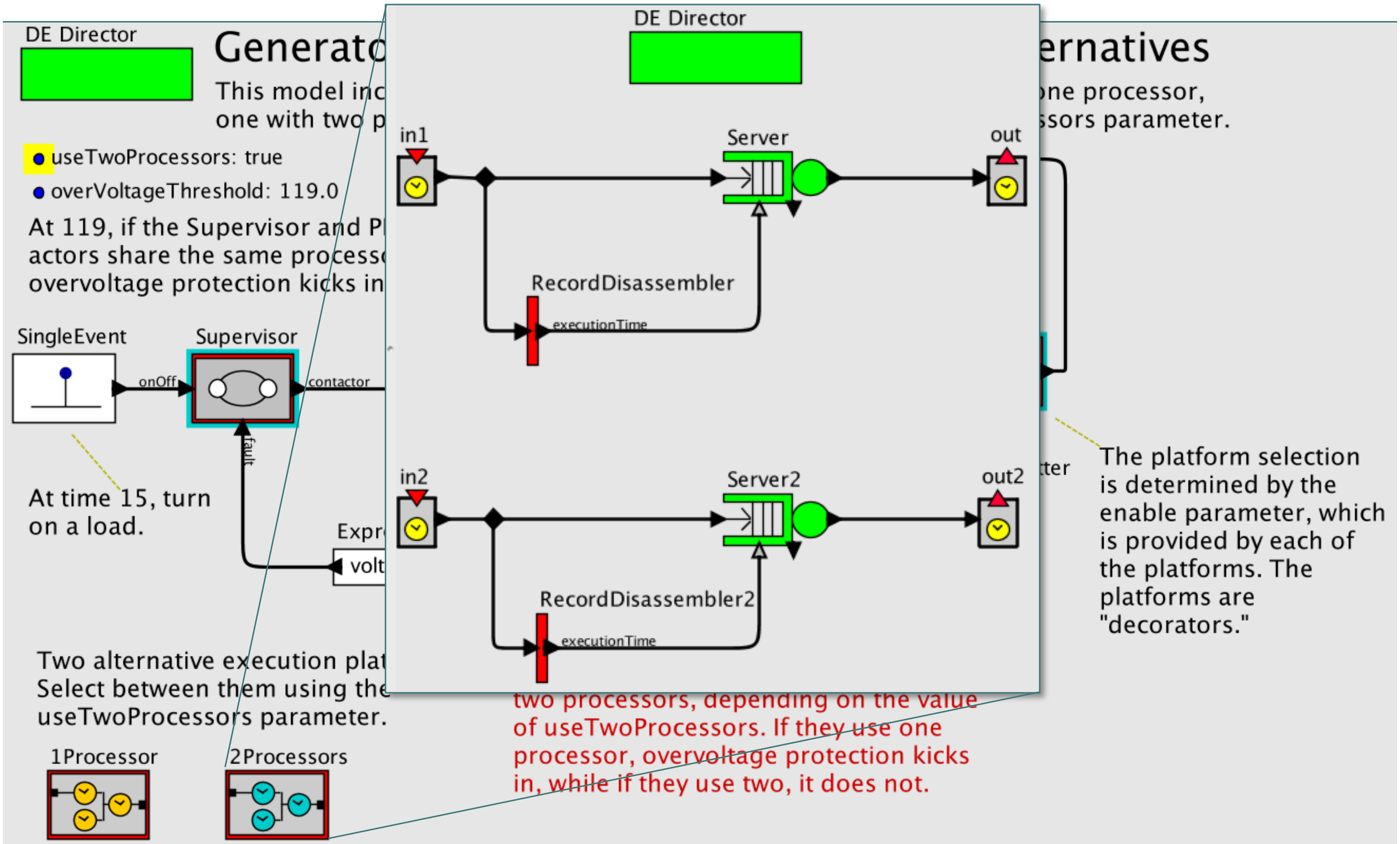
Some Promising Approaches

- PRET machines
- Network time synchronization (PTP, IEEE 1588)
- Model-based design
  - with timed, determinate, concurrent MoCs, like PTIDES
- Temporal logics
  - with methods for synthesis and verification.
- Platform-based design
- Contract-based design
- Abstract semantics
  - for heterogeneous modeling
- Co-simulation and model-exchange standards
  - particularly FMI
- Aspect-oriented modeling

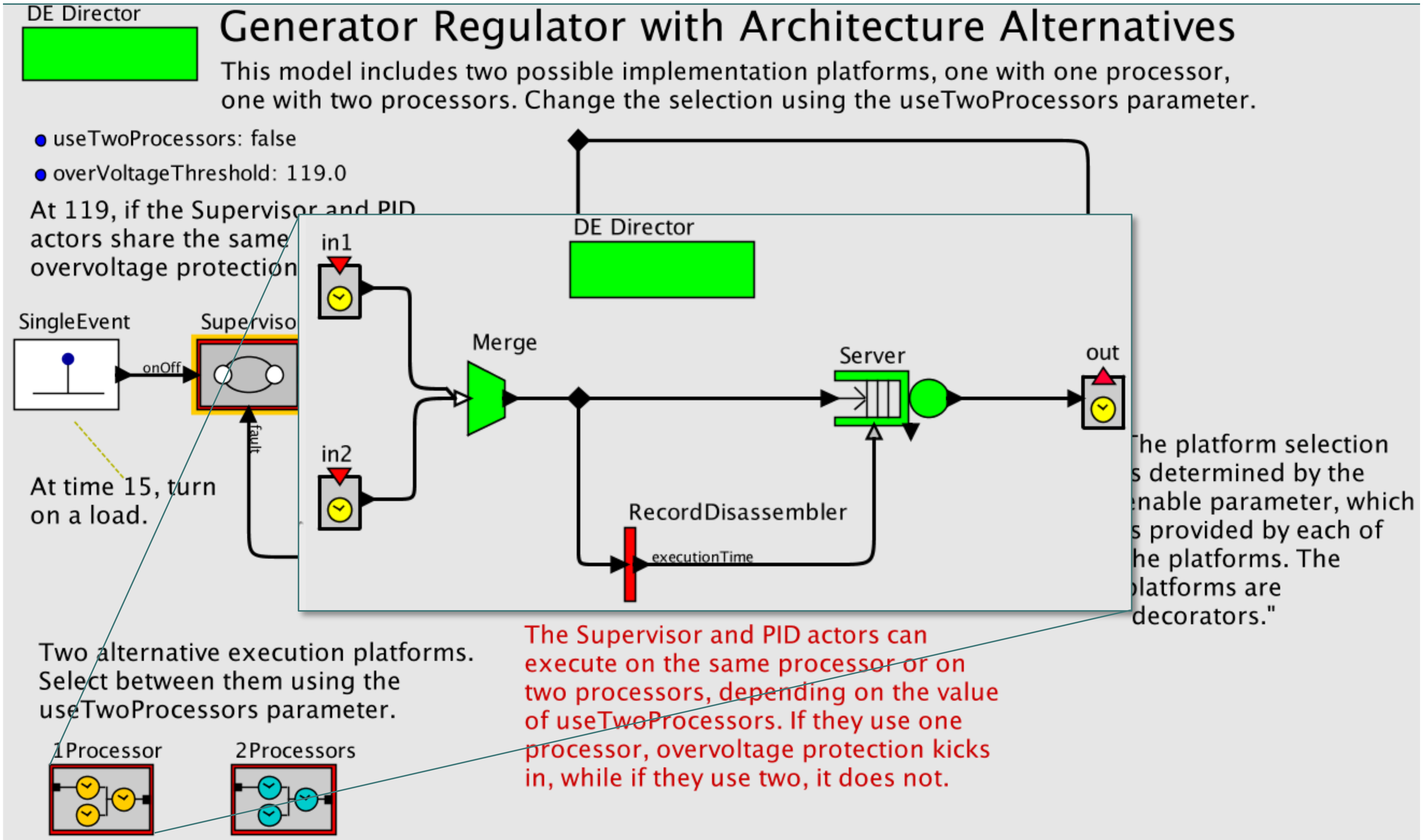
Aspect-Oriented Modeling
Decorators and Metronomy



Two Processor Architecture Model



One Processor Architecture Model



Four Big Challenges

1. Determinate CPS models
2. Open minds about languages and tools
3. A semantics of time
4. A discipline of "model engineering"

Raffaello Sanzio da Urbino – The Athens School