



Systems of Systems Modeling

Edward A. Lee

*Robert S. Pepper Distinguished Professor
UC Berkeley*

Invited Talk

International Conference on Complex Systems Design & Management

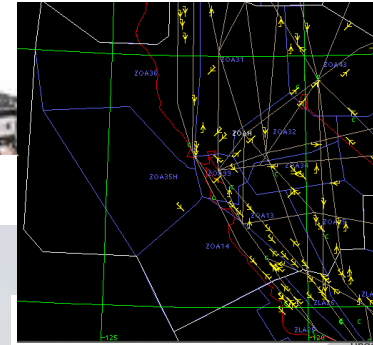
December 4-6, 2013

Paris, France

Cyber-Physical Systems (CPS): Orchestrating networked computational resources with physical systems

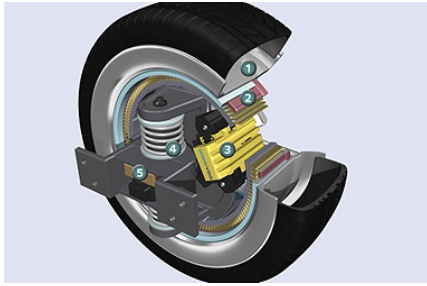


Avionics



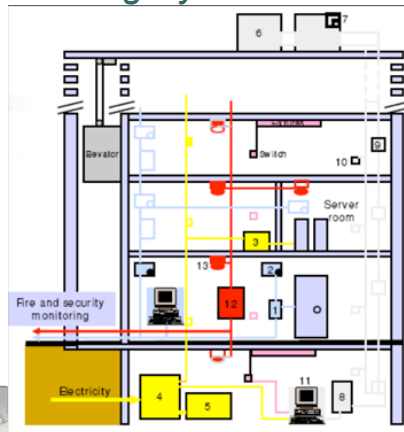
Transportation
(Air traffic control at SFO)

Automotive

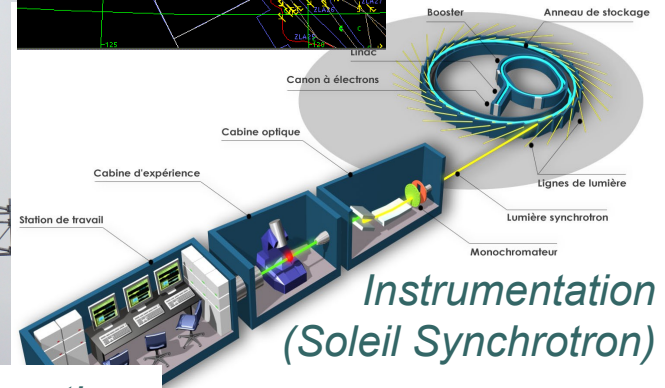


E-Corner, Siemens

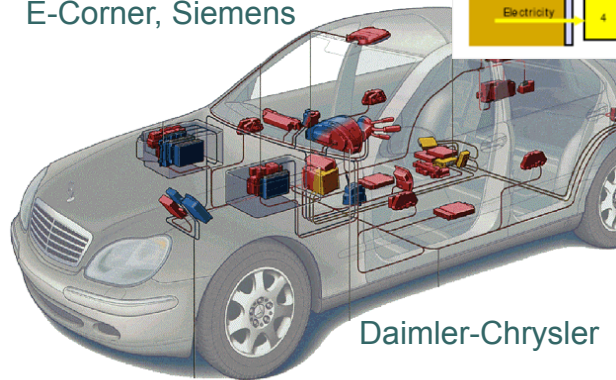
Building Systems



Telecommunications



Instrumentation
(Soleil Synchrotron)



Daimler-Chrysler

Military systems:



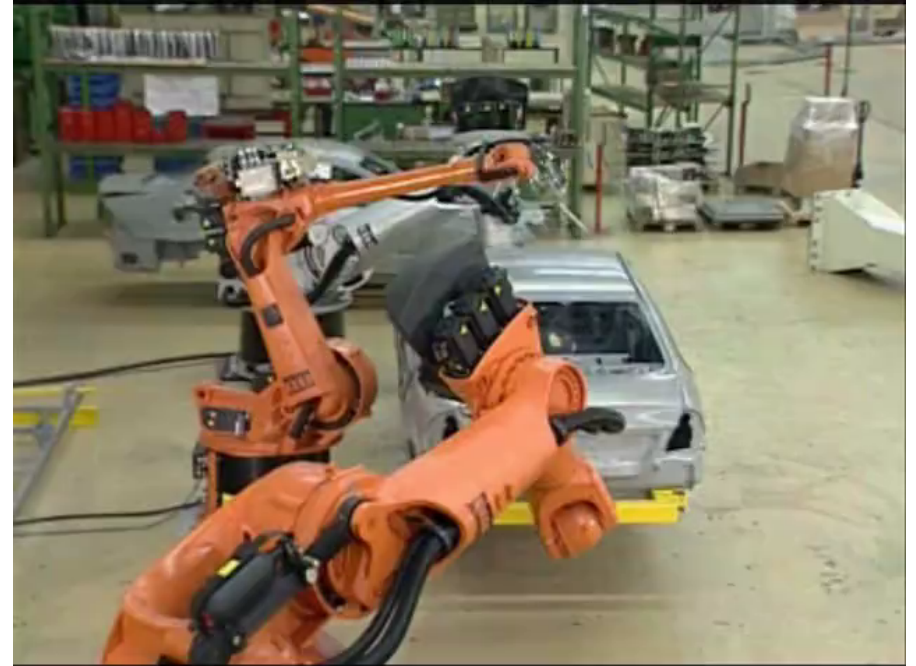
Courtesy of Doug Schmidt

Power generation and distribution

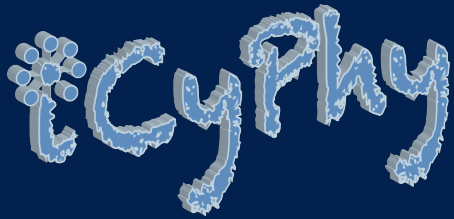


Courtesy of General Electric

Factory automation



Courtesy of Kuka Robotics Corp.



A New Research Center: iCyPhy Industrial Cyber-Physical Systems

Berkeley:

Ilge Akkaya
Nikunj Bajaj
David Broman
Christopher Brooks
Dai Bui
Patricia Derler
John Eidson, PhD
John B. Finn
Liangpeng Guo
Barb Hoversten
Antonio Iannopolo
Hokeun Kim



Edward A. Lee
Chung-Wei Lin
Marten Lohstroh
Mehdi Maasoumy
Pierluigi Nuzzo
Alberto Puggelli
Alberto Sangiovanni-Vincentelli
Chris Shaver
Christos Stergiou
Stavros Tripakis
Ben Zhang

Other Participants:

Johan Akesson (Modelon)
Torsten Blochwitz (Itisim)
Magnus Gafvert (Modelon)
Oscar Mickelin (KTH)
Necmiye Ozay (U. Michigan)
Hubertus Tummescheit (Modelon)
Michael Wetter (LBL)

Caltech:

Scott Livingston
Quentin Malliet
Yilin Mo
Richard Murray
Necmiye Ozay
Mumu Xu



UTC:

Joshua Berg
Gloria F. Byar
Massimiliano Chiodo
Don Chritz
Marco di Natale
Jeff Ernst
Alberto Ferrari
Clas Jacobson
Scott Kaslusky
Al Markunas
Brian T. Murray
Satish Narayanan
Claudio Pinello
Alessandro Pinto
Eelco Scholte

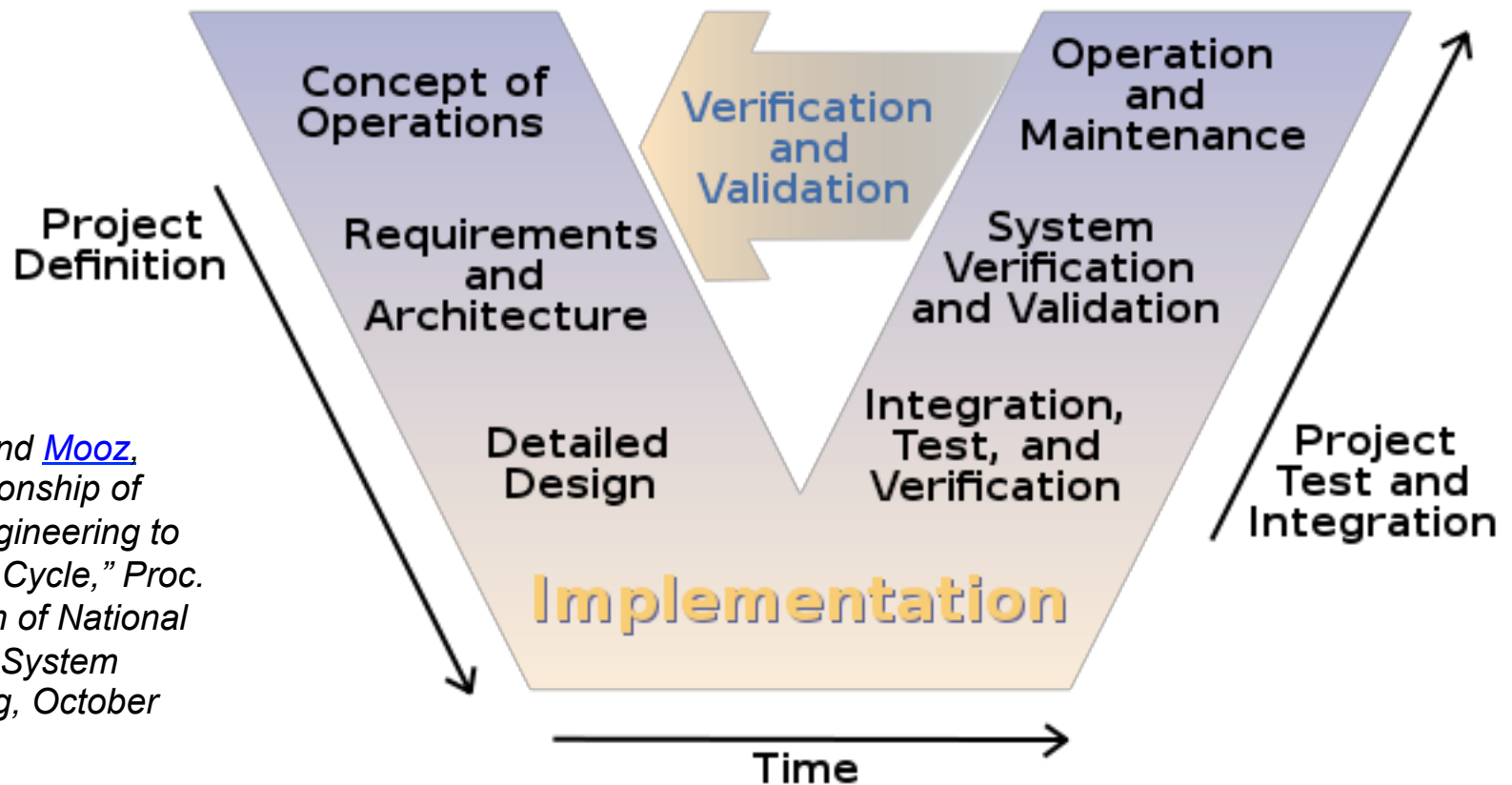
IBM:

John F. Arnold
Henry Broodney
Yishai Feldman
Amit Fisher
Lev Greenberg
Itai Jaeger
Michael Masin
Eldad Palachi
Gabi Zodik





The Conventional V-Model of System Design



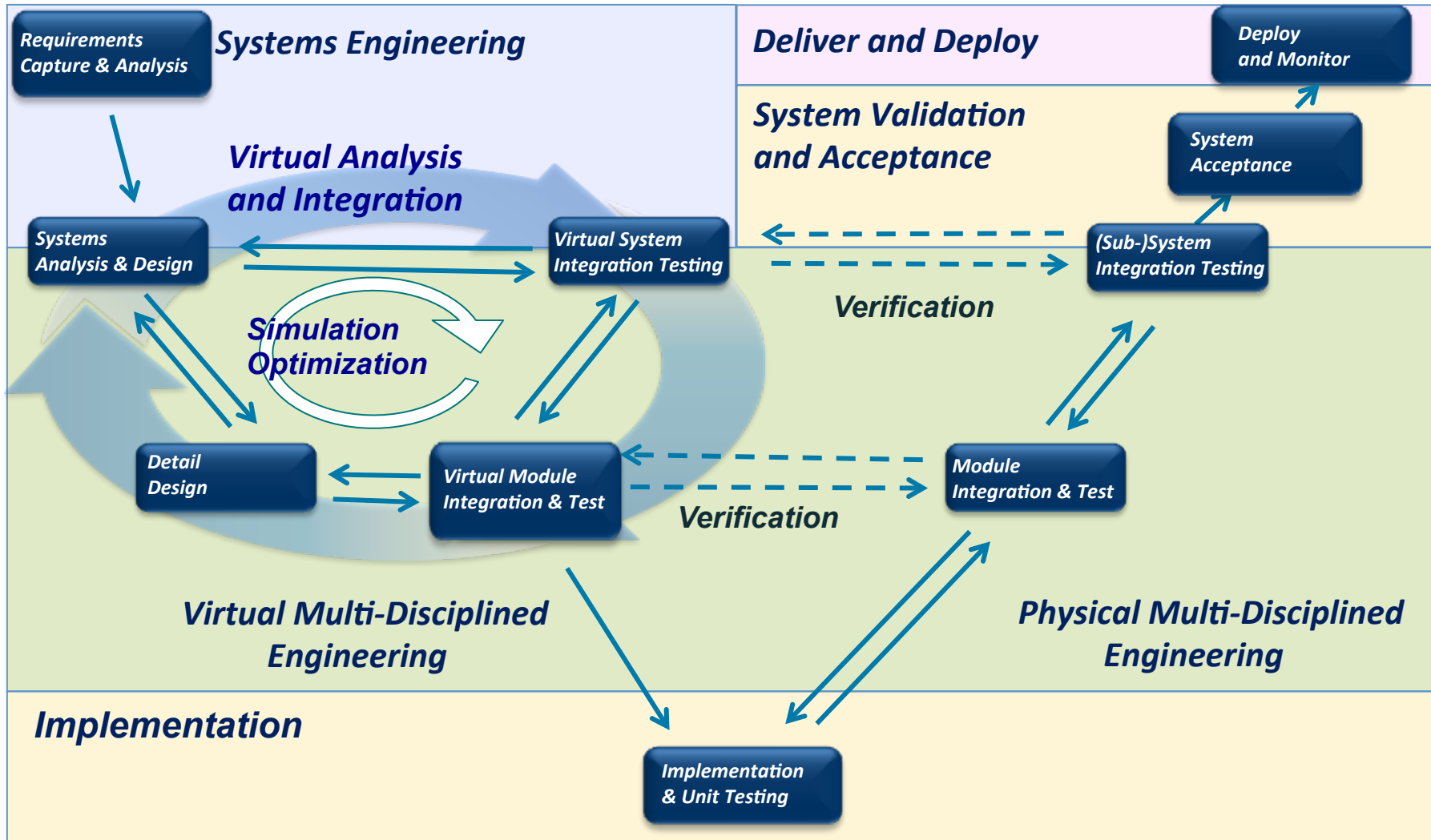
[Forsberg and Mooz](#),
"The Relationship of
System Engineering to
the Project Cycle," Proc.
Symposium of National
Council on System
Engineering, October
1991.

**This leads to late
validation (at system
integration time).**

Image source: [Clarus Concept of Operations](#). Publication No. FHWA-JPO-05-072, Federal Highway Administration (FHWA), 2005.



The iCyPhy Model of System Design



Virtual Integration Requires Working with Models. 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.

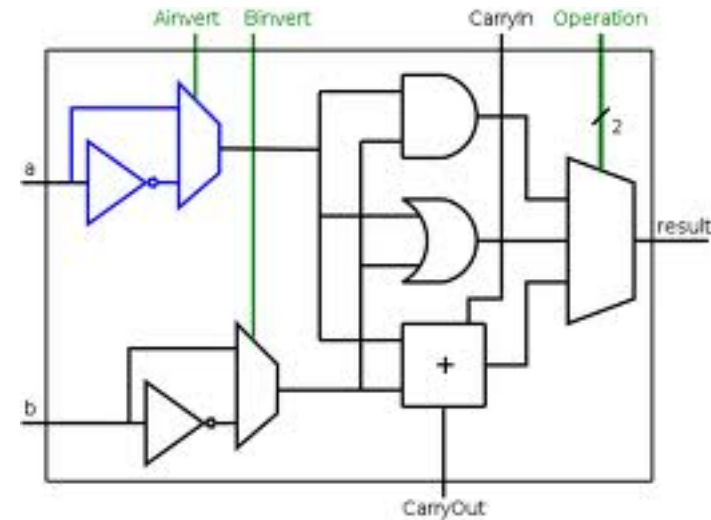
We can make definitive statements about *models*, from which we can *infer* properties of system realizations. The validity of this inference depends on *model fidelity*, which is always approximate.

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

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

A Major Problem for CPS: Combinations are Nondeterminate

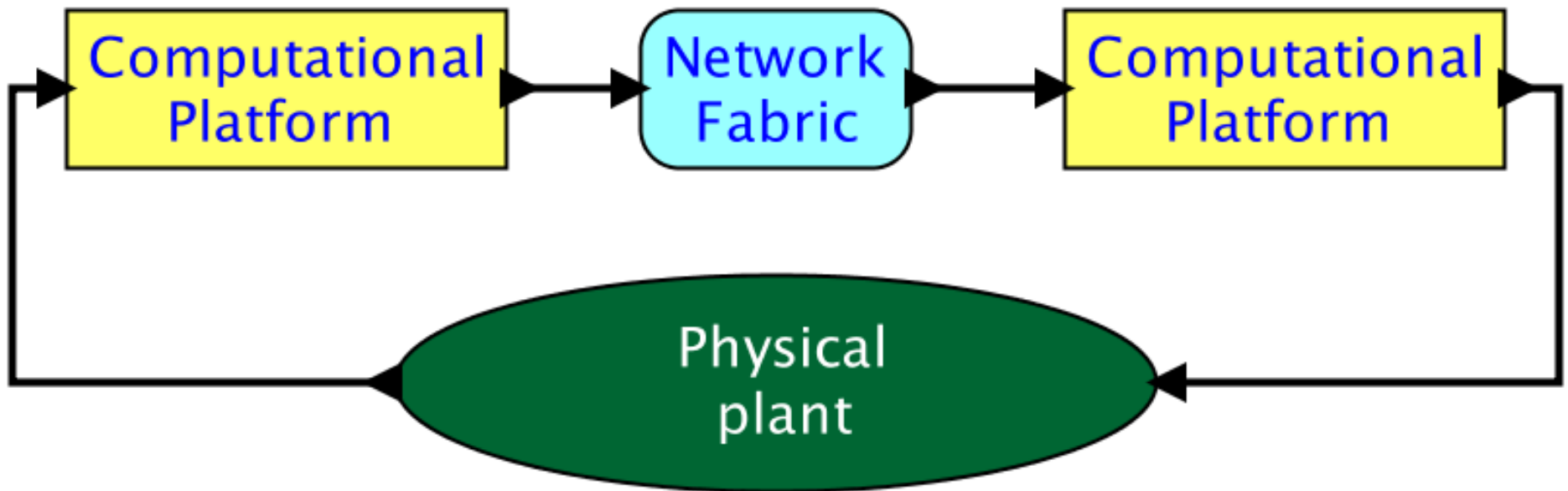


```
/** Reset the output receivers, which are the inside receivers of
 * the output ports of the container.
 * @exception IllegalArgumentException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalArgumentException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();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

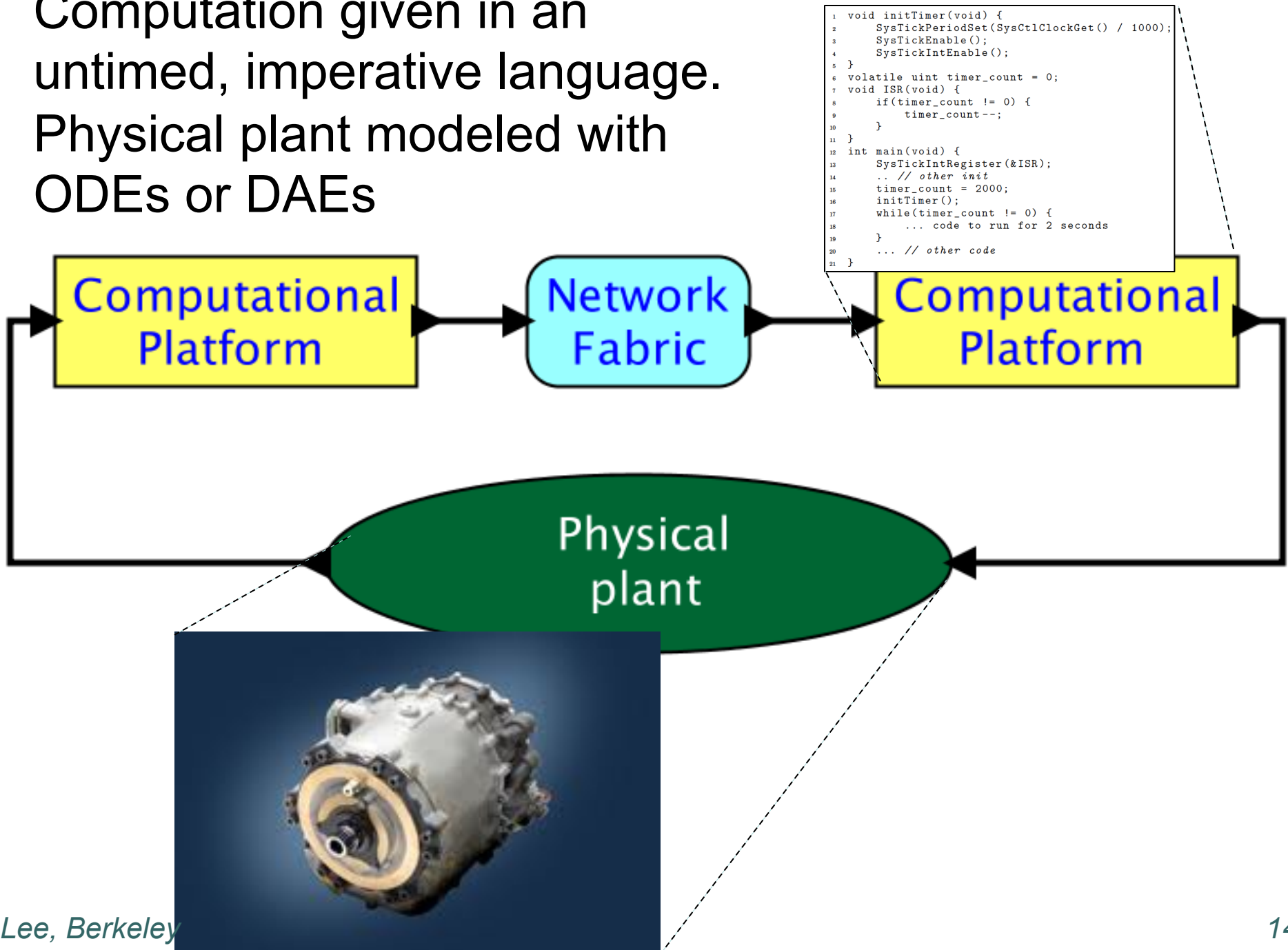


$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M} \int_0^t \mathbf{F}(\tau) d\tau$$

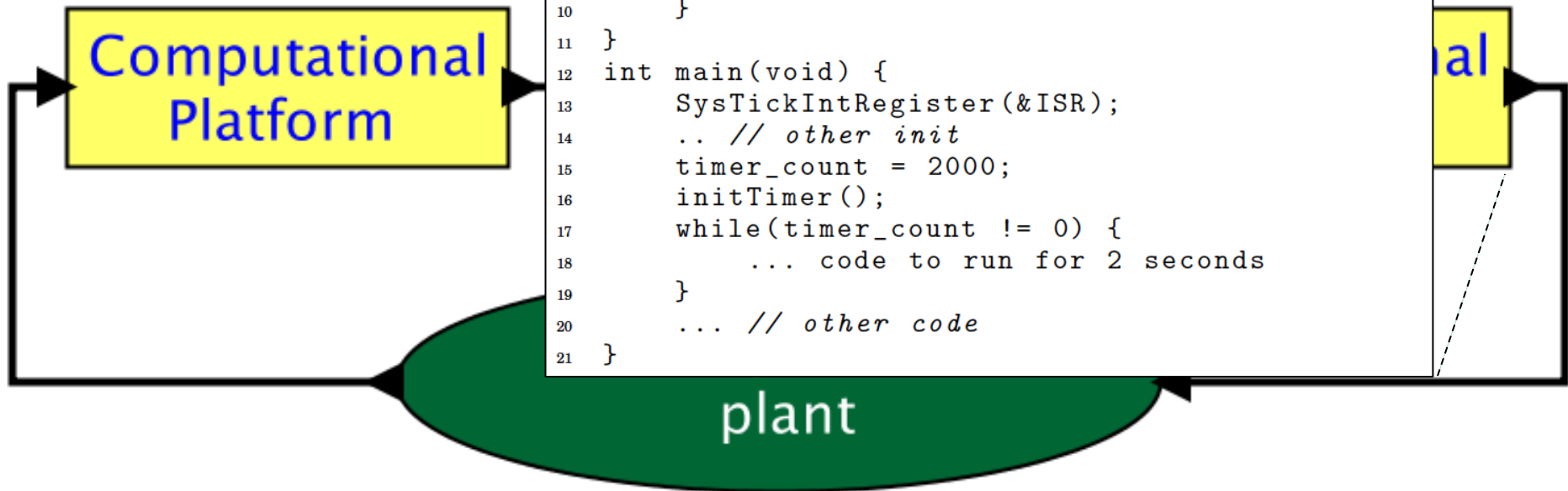
Schematic of a simple CPS:



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?

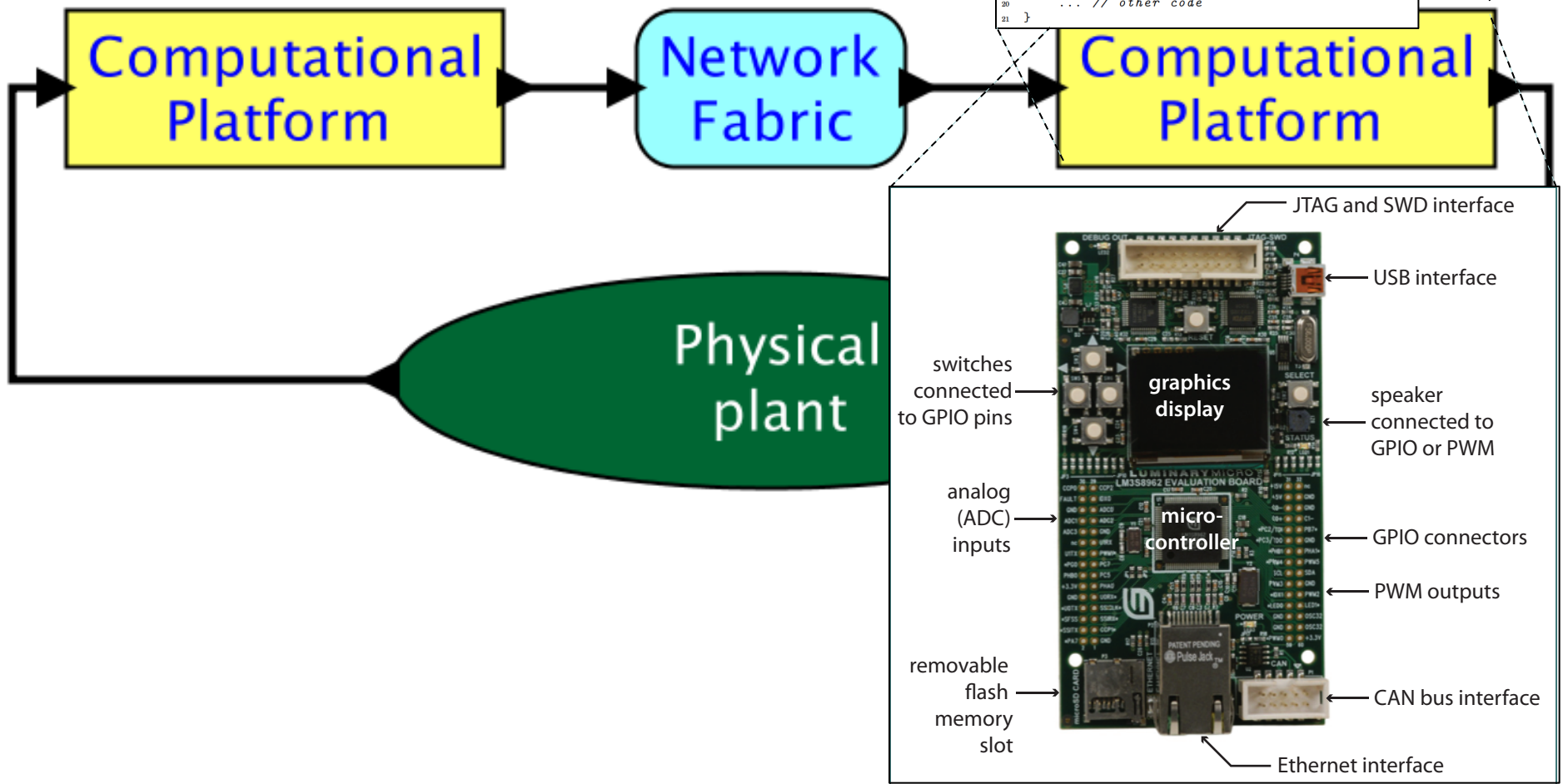


Timing behavior emerges from the combination of the program and the hardware platform.

```

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 }

```



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.



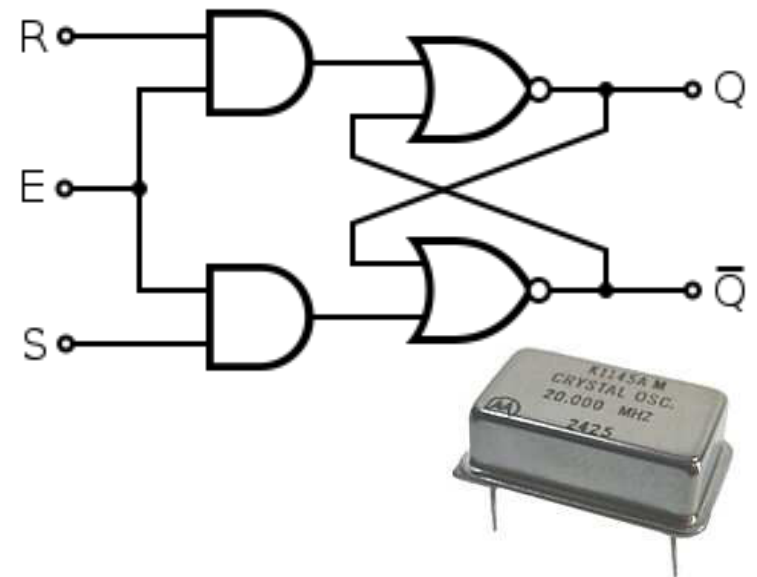
Programmers have to step *outside* the programming abstractions to specify timing behavior.

Programmers have no map!

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.



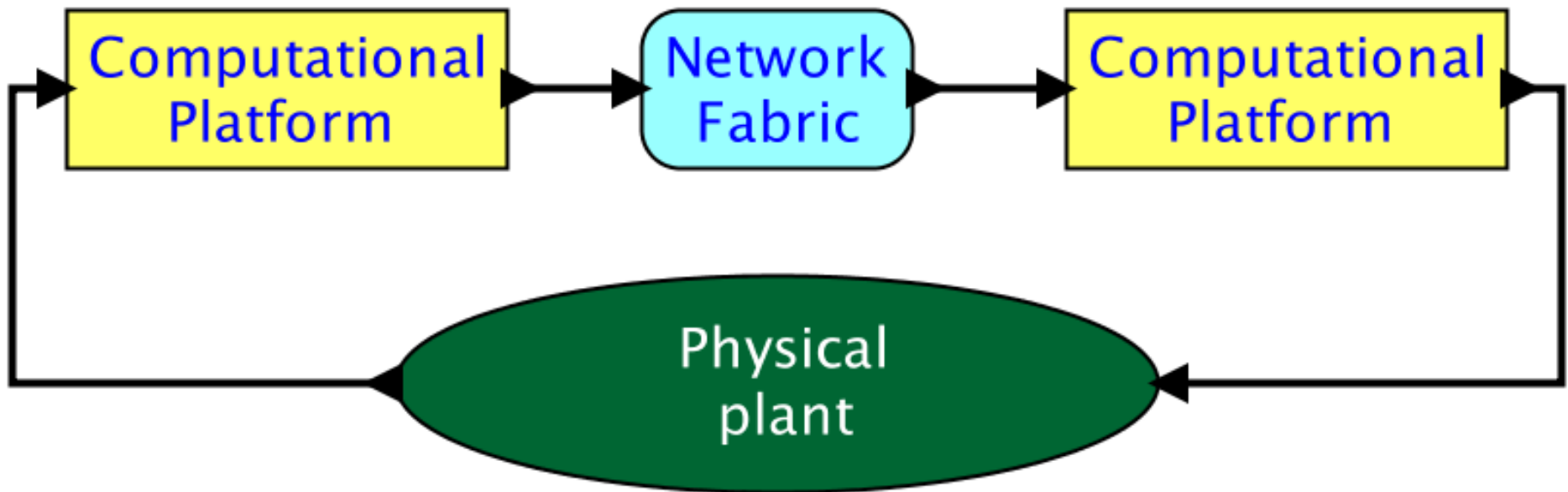
```
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
```

Challenge

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?

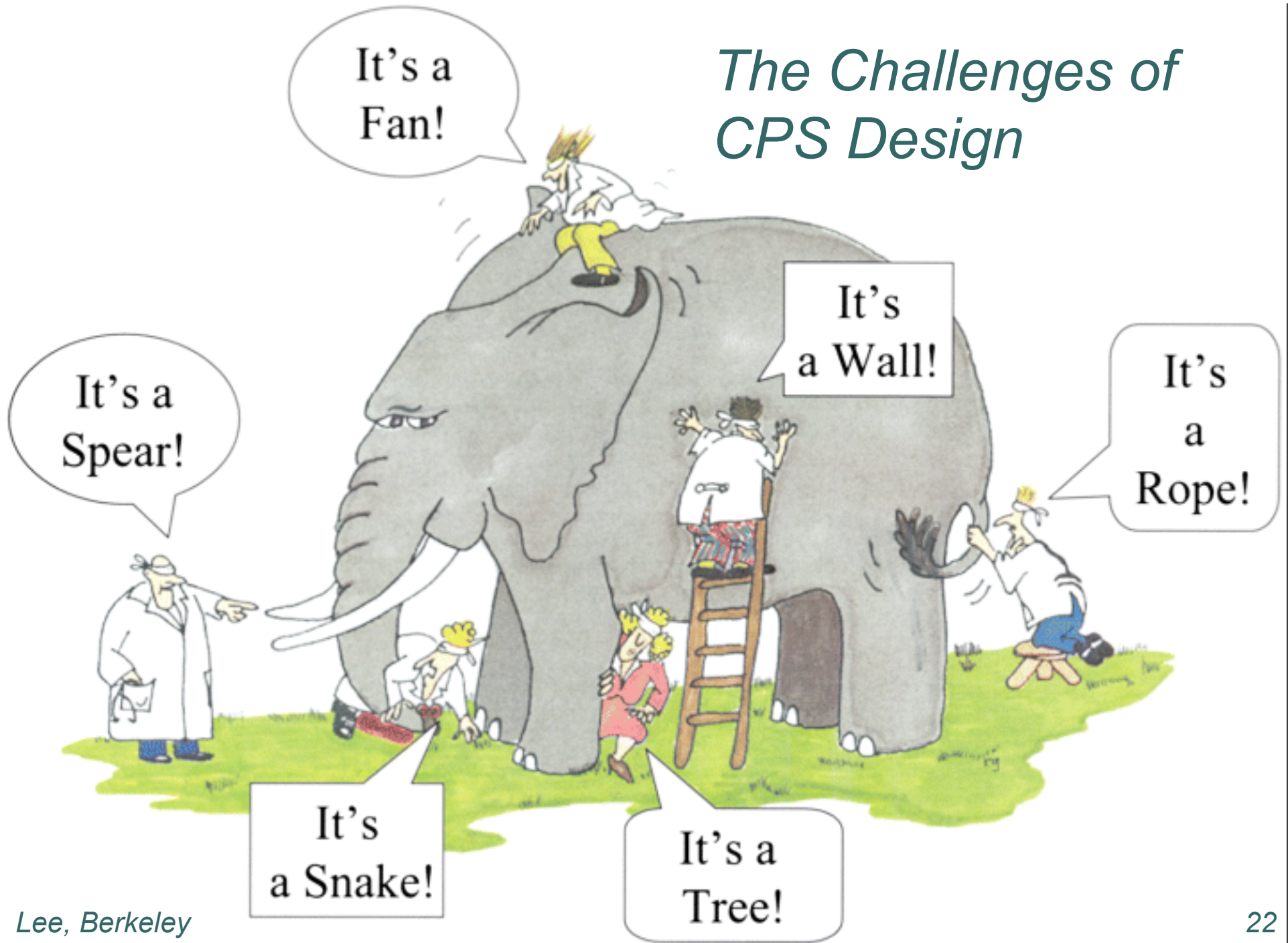
i.e. we need determinate CPS models

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

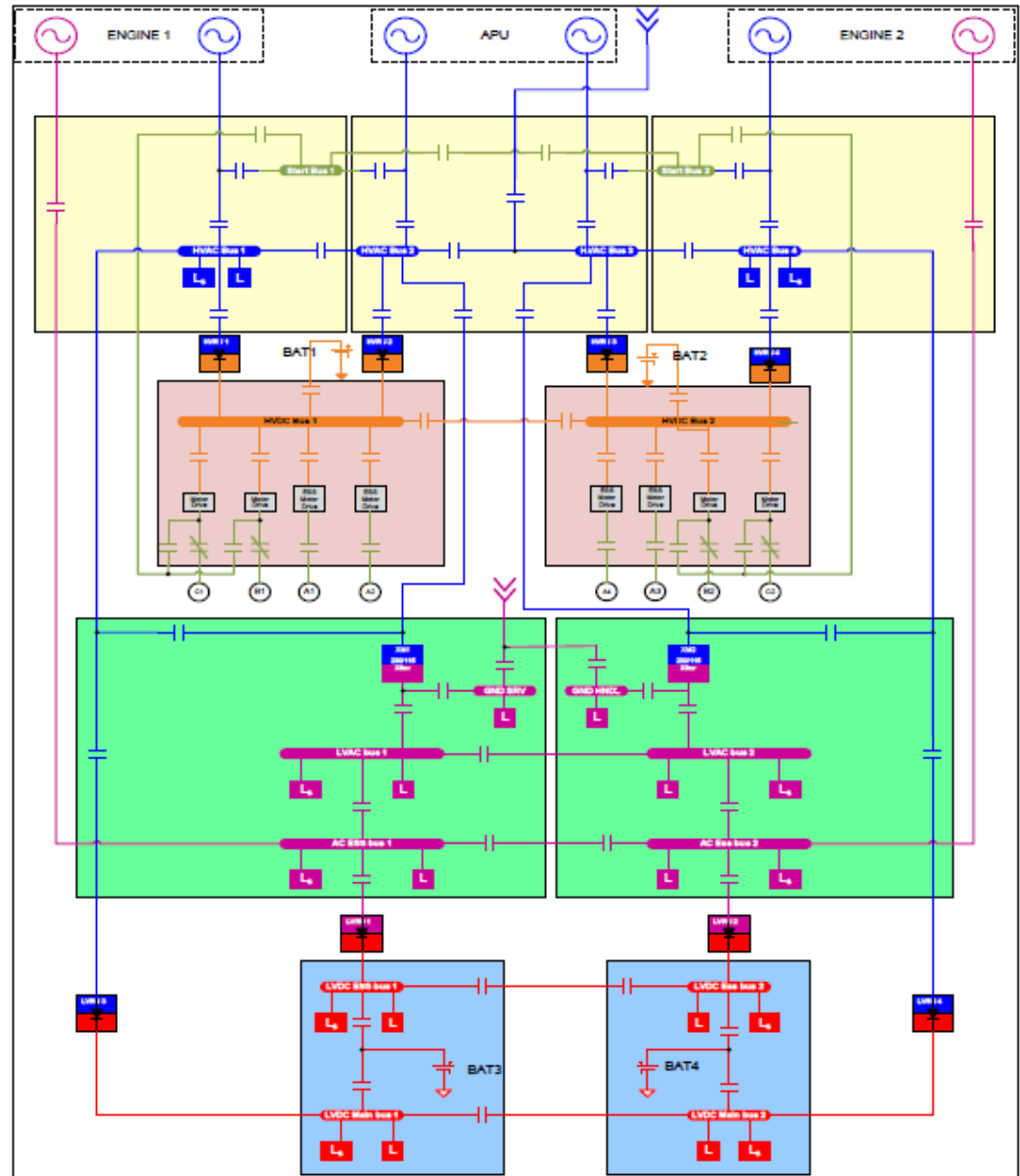
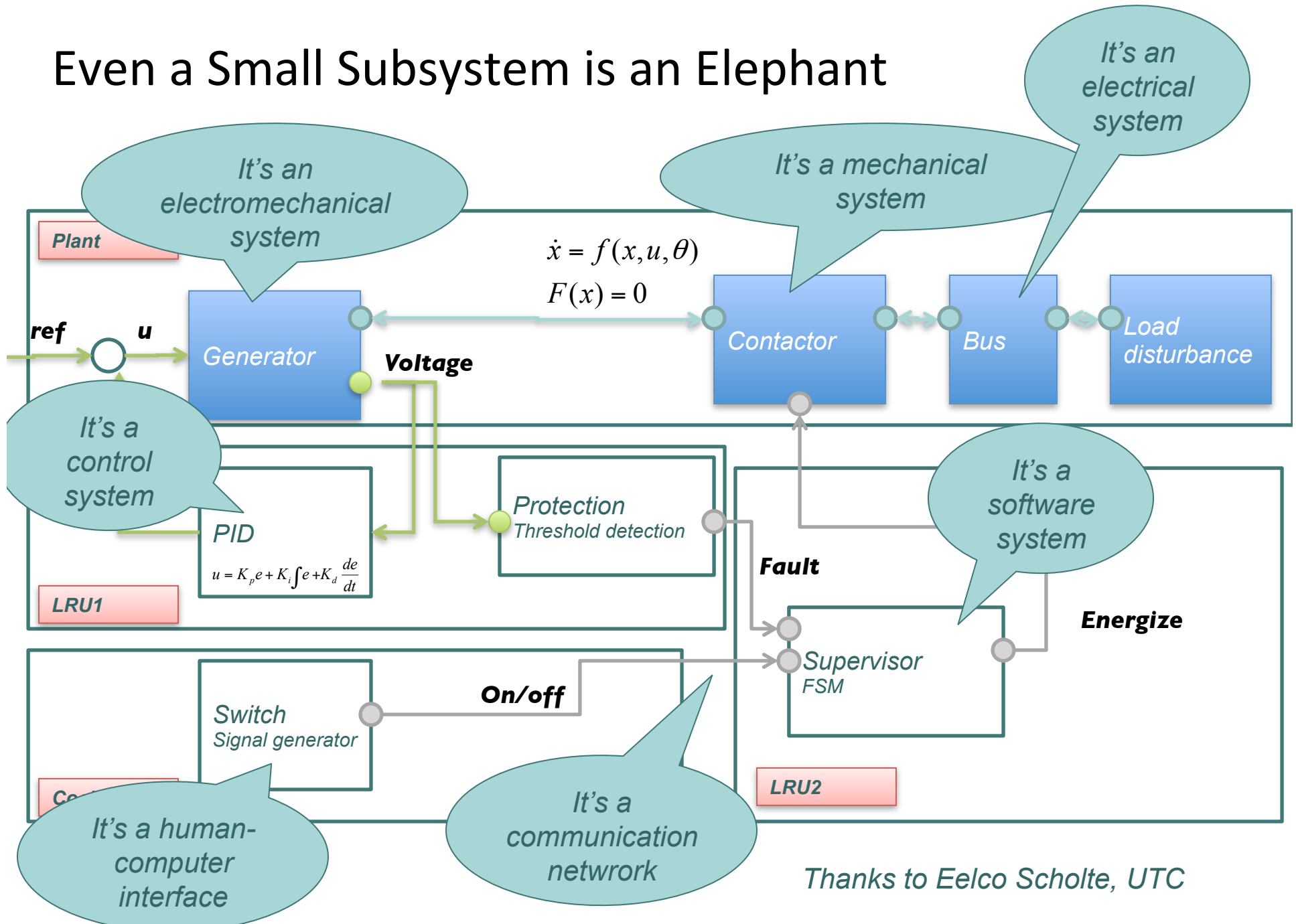


Figure 1: Single line diagram of an electric power system adapted from Honeywell Patent US 7,439,634 B2. Figure courtesy of Rich Poisson, Hamilton-Sundstrand.

Even a Small Subsystem is an Elephant



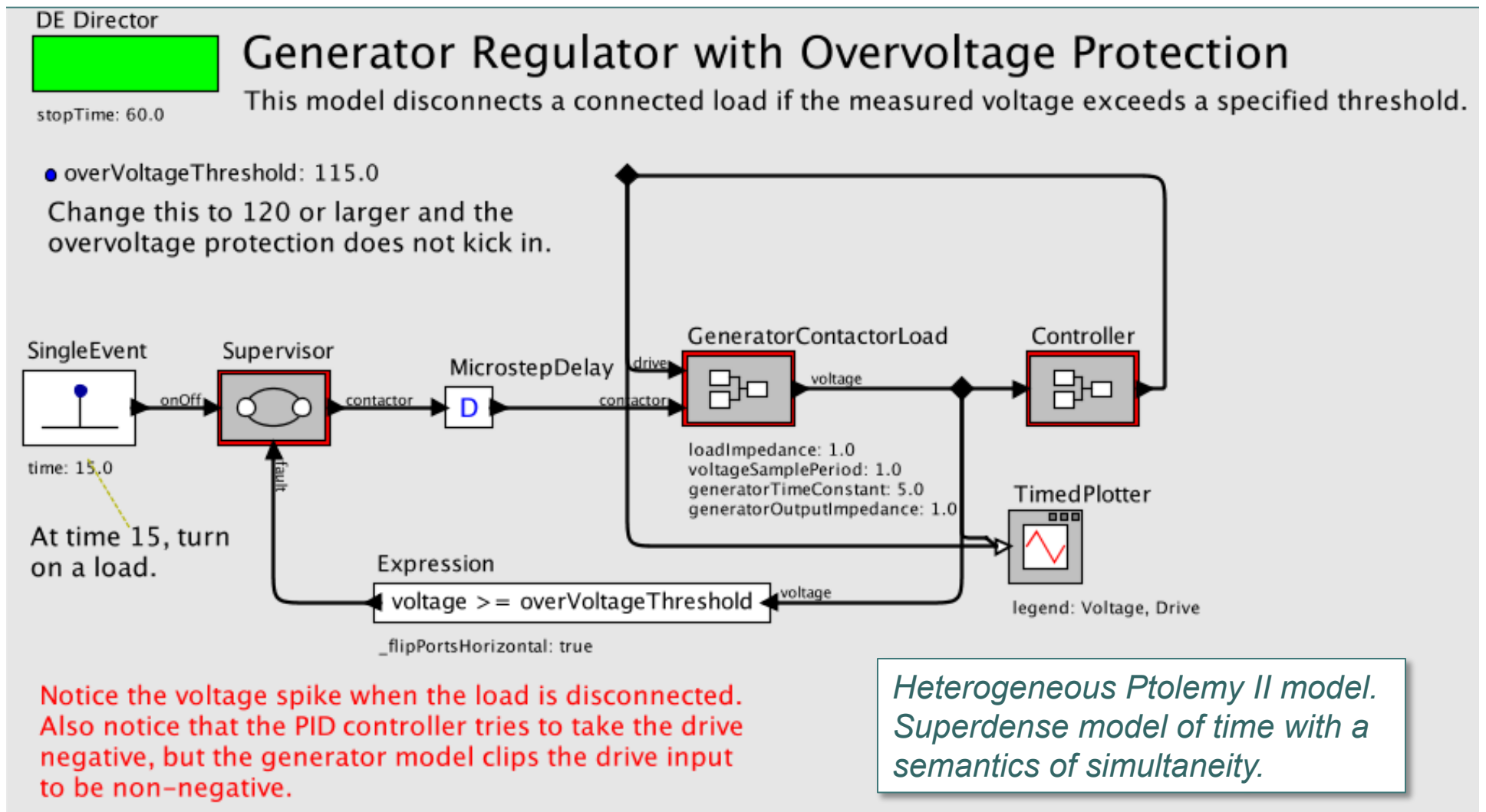
Thanks to Eelco Scholte, UTC

Challenge

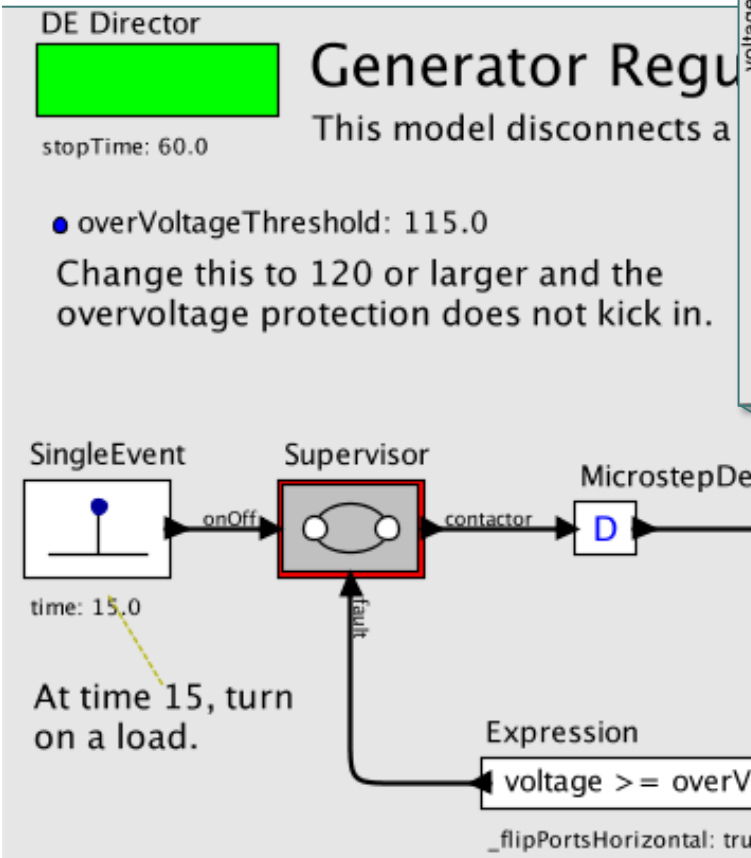
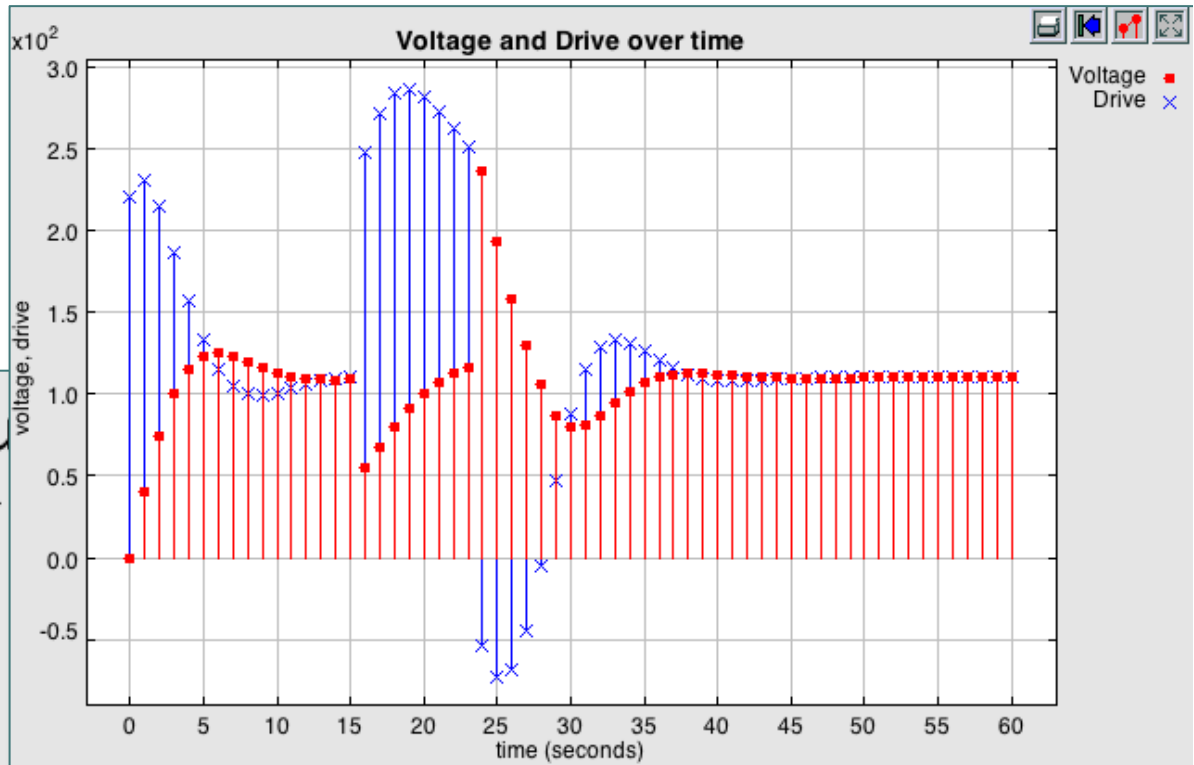
How can we define interfaces between components that bridge engineering disciplines and clarify requirements and expectations?

i.e. we need a discipline of “model engineering”

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



Execution of the Ptolemy II 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

DE Director
 stopTime: 60.0

Generator Re
 This model disconn

- overVoltageThreshold: 115.0

Change this to 120 or larger and the overvoltage protection does not kick

Continuous Director
 Plant Model with Generator, Load, and Contactor

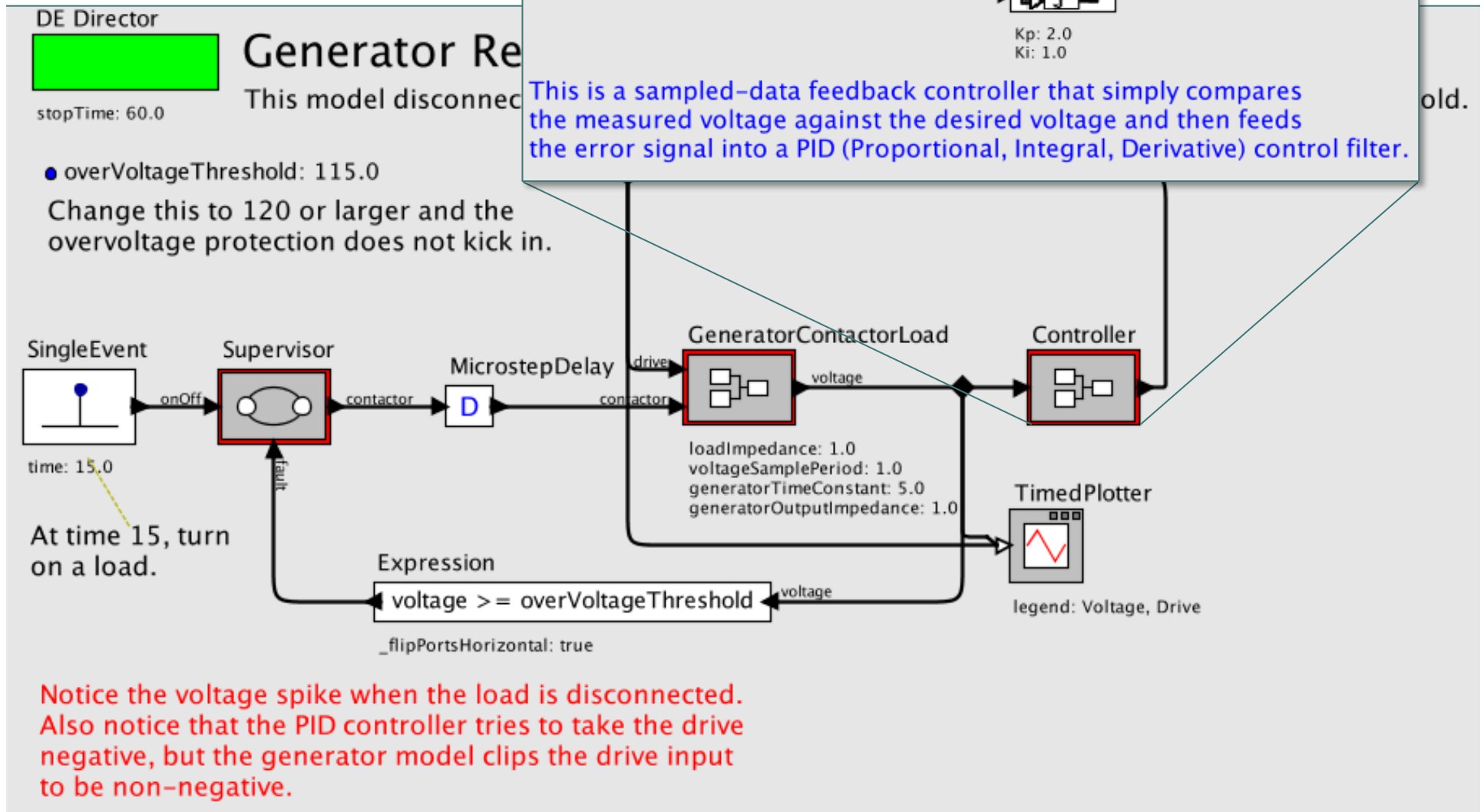
This is a model of generator with a load with a digital interface. The inputs are discrete events, and the output is a periodically sampled measurement of the voltage.

- loadImpedance: 10.0
- voltageSamplePeriod: 1.0
- generatorTimeConstant: 1.0
- generatorOutputImpedance: 1.0

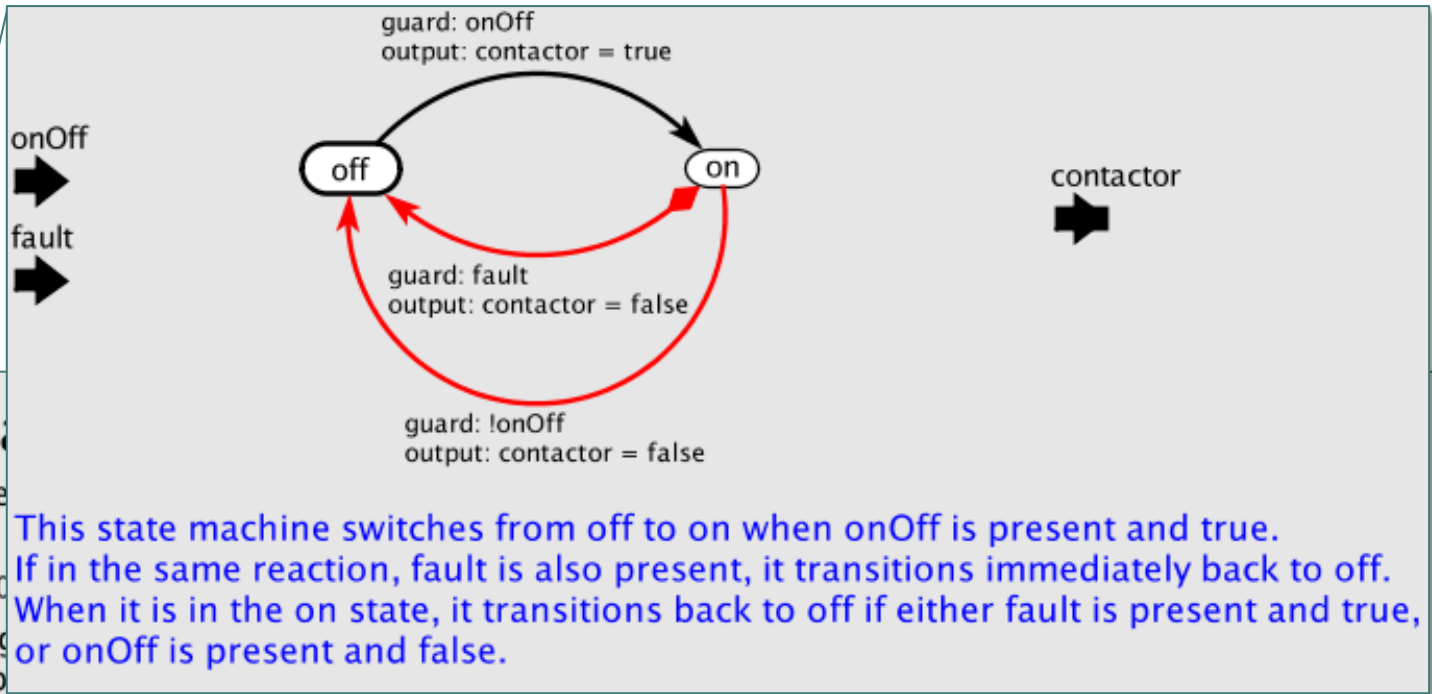
At time 15, turn on a load.

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.

Dataflow Model of a Sampled-Data Controller



State Machine Model of a Supervisory Controller



This state machine switches from off to on when onOff is present and true. If in the same reaction, fault is also present, it transitions immediately back to off. When it is in the on state, it transitions back to off if either fault is present and true, or onOff is present and false.

DE Director

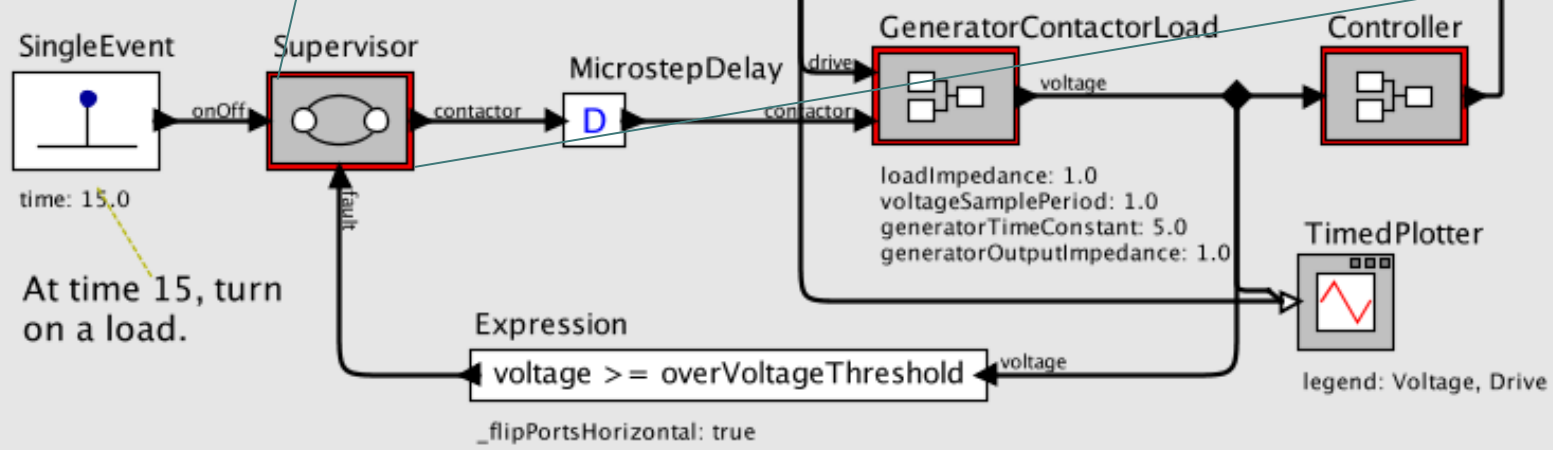
stopTime: 60.0

Generator

This mode

overVoltageThreshold: 115.0

Change this to 120 or larger for overvoltage protection do



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.

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

DE Director



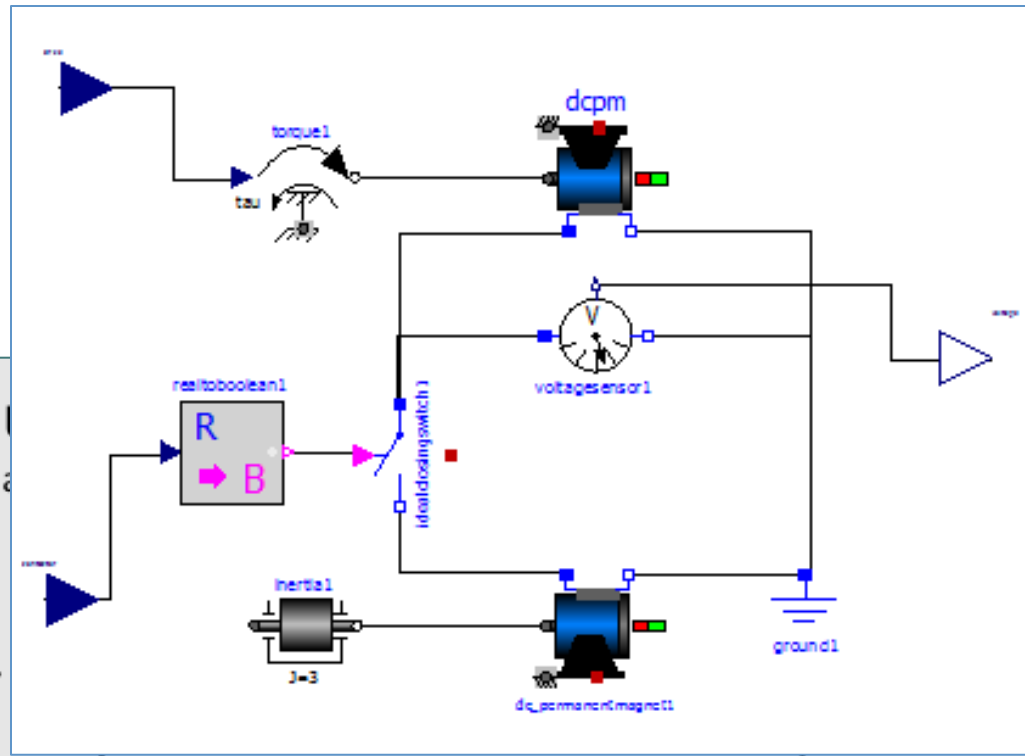
stopTime: 60.0

Generator Regulator

This model disconnects a

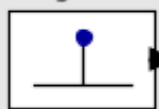
● overVoltageThreshold: 115.0

Change this to 120 or larger and the overvoltage protection does not kick in.



threshold.

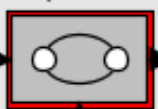
SingleEvent



time: 15.0

At time 15, turn on a load.

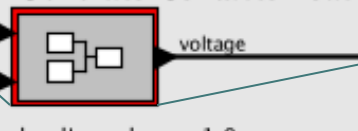
Supervisor



MicrostepDelay

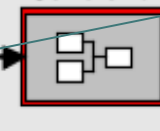


GeneratorContactorLoad



loadImpedance: 1.0
voltageSamplePeriod: 1.0
generatorTimeConstant: 5.0
generatorOutputImpedance: 1.0

Controller



TimedPlotter



legend: Voltage, Drive

Expression

`voltage >= overVoltageThreshold`

_flipPortsHorizontal: true

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.

Aspect-Oriented Modeling in Ptolemy II

DE Director

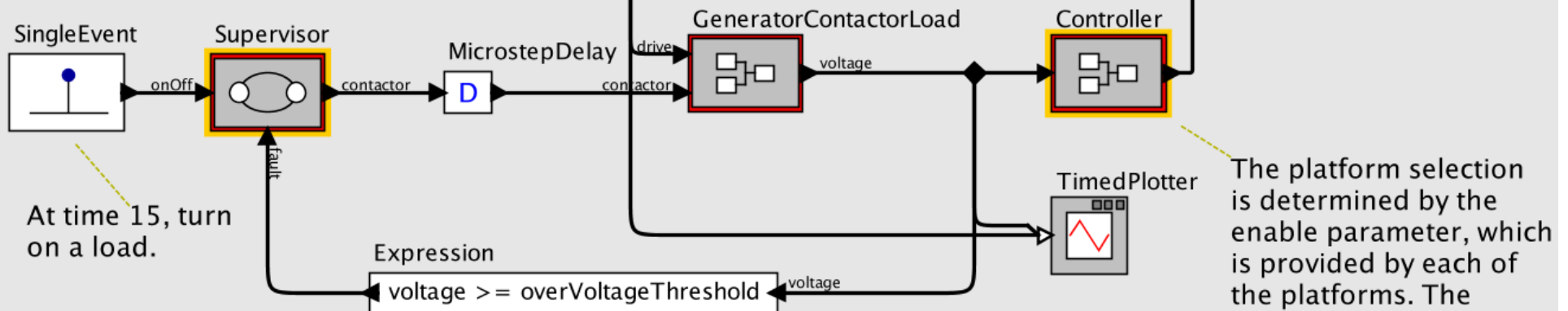


Generator Regulator with Architecture Alternatives

This model includes two possible implementation platforms, one with one processor, one with two processors. Change the selection using the useTwoProcessors parameter.

- useTwoProcessors: false
- overVoltageThreshold: 119.0

At 119, if the Supervisor and PID actors share the same processor, then overvoltage protection kicks in.

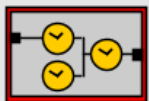


At time 15, turn on a load.

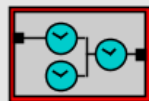
The platform selection is determined by the enable parameter, which is provided by each of the platforms. The platforms are "decorators."

Two alternative execution platforms. Select between them using the useTwoProcessors parameter.

1Processor



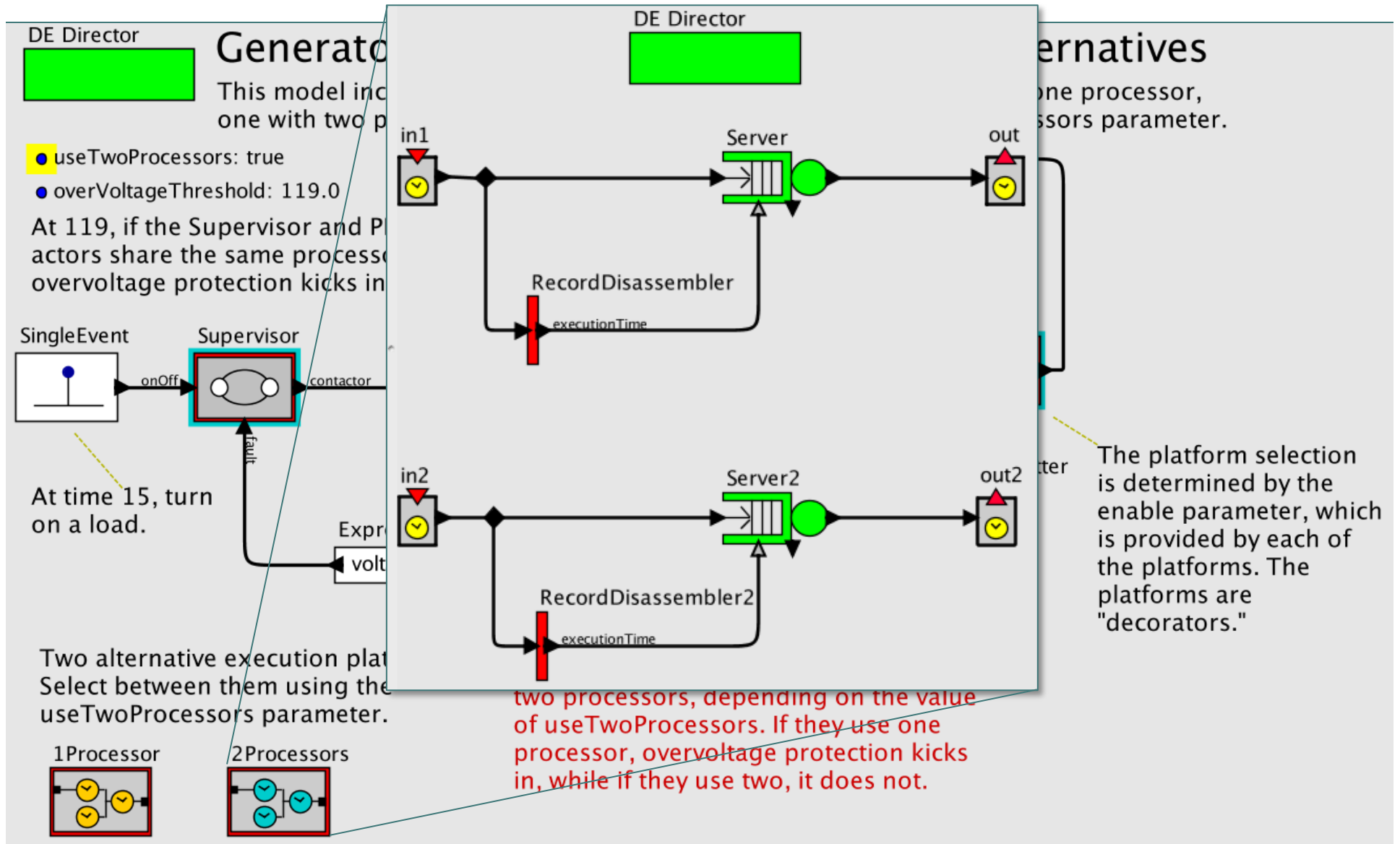
2Processors



The Supervisor and PID actors can execute on the same processor or on two processors, depending on the value of useTwoProcessors. If they use one processor, overvoltage protection kicks in, while if they use two, it does not.

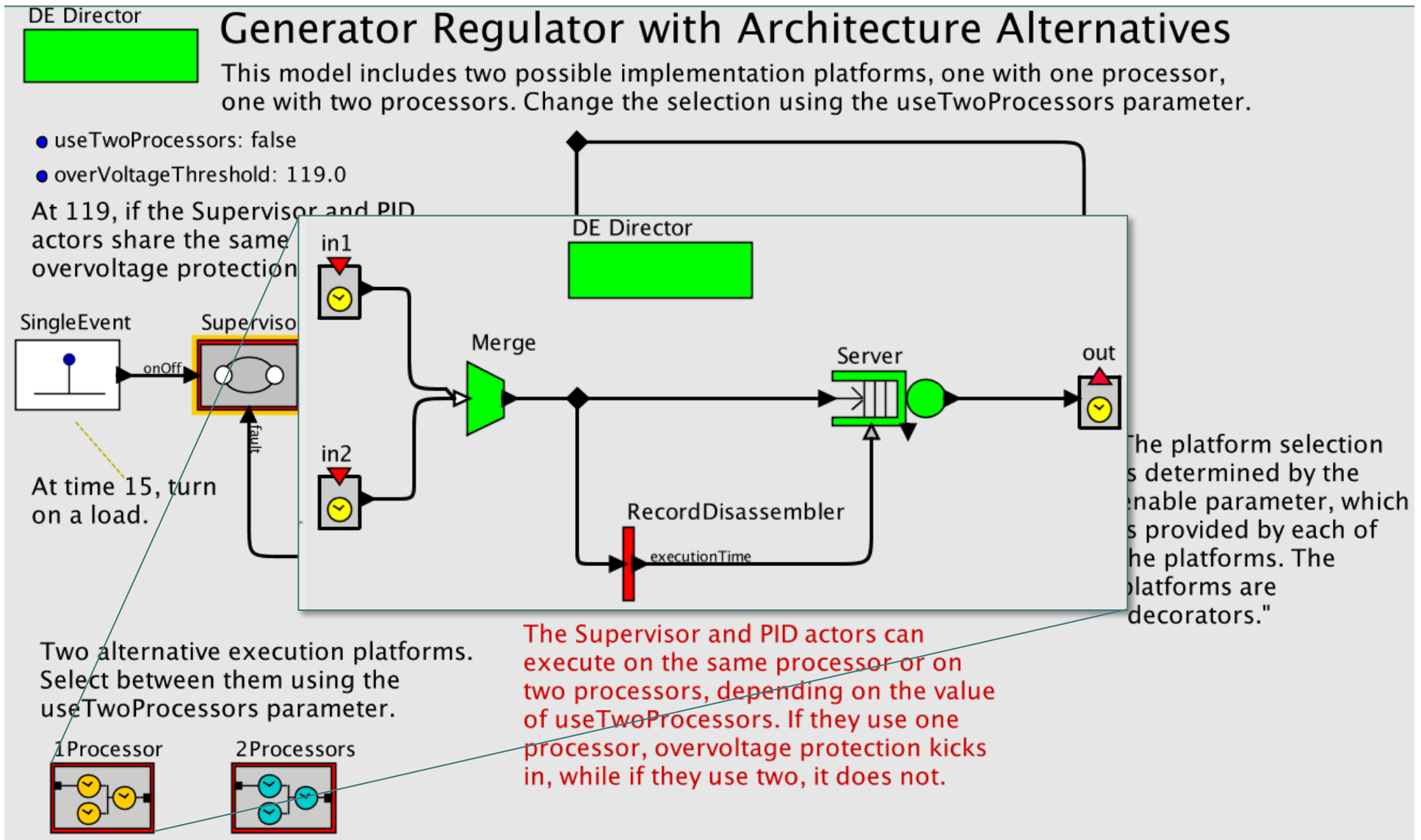
Aspect-Oriented Modeling in Ptolemy II

Two Processor Architecture Model



Aspect-Oriented Modeling in Ptolemy II

One Processor Architecture Model



Other Uses of Aspect-Oriented Modeling

- Modeling communication system architecture
- Modeling the effects of communication impairments
- Modeling faults
- Creating observers:
 - Anomaly detection
 - Contract monitoring
- Security attack models
- ...

In Ptolemy II, all of these models are cleanly separated from the functional system model.

Their effect on the model is “woven in” at run time.

Ptolemy II

<http://ptolemy.org>

- *Open source*
- *Open architecture*
- *Well documented*

Free Book:

Claudius Ptolemaeus, Editor

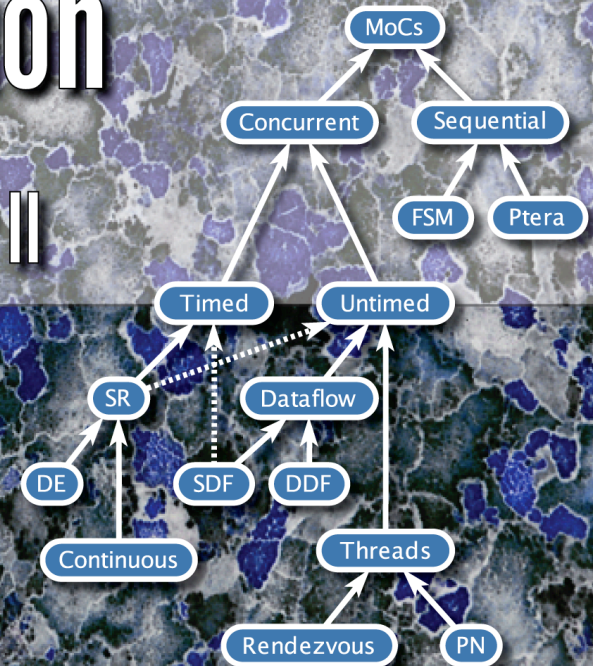
<http://ptolemy.org/systems>

This is an open-source book about open source software solutions to pressing industrial problems.

Lee, Berkeley

System Design, Modeling, and Simulation

Using Ptolemy II



Claudius Ptolemaeus, Editor

Systems of Systems Modeling

A discipline of “model engineering”

- Embrace models for virtual system integration
- Avoid accidental nondeterminism
- Embrace heterogeneity
- Use aspect-oriented modeling

Raffaello Sanzio da Urbino – The Athens School

