iCyPhy

Actors Revisited for Predictable Systems

Edward A. Lee

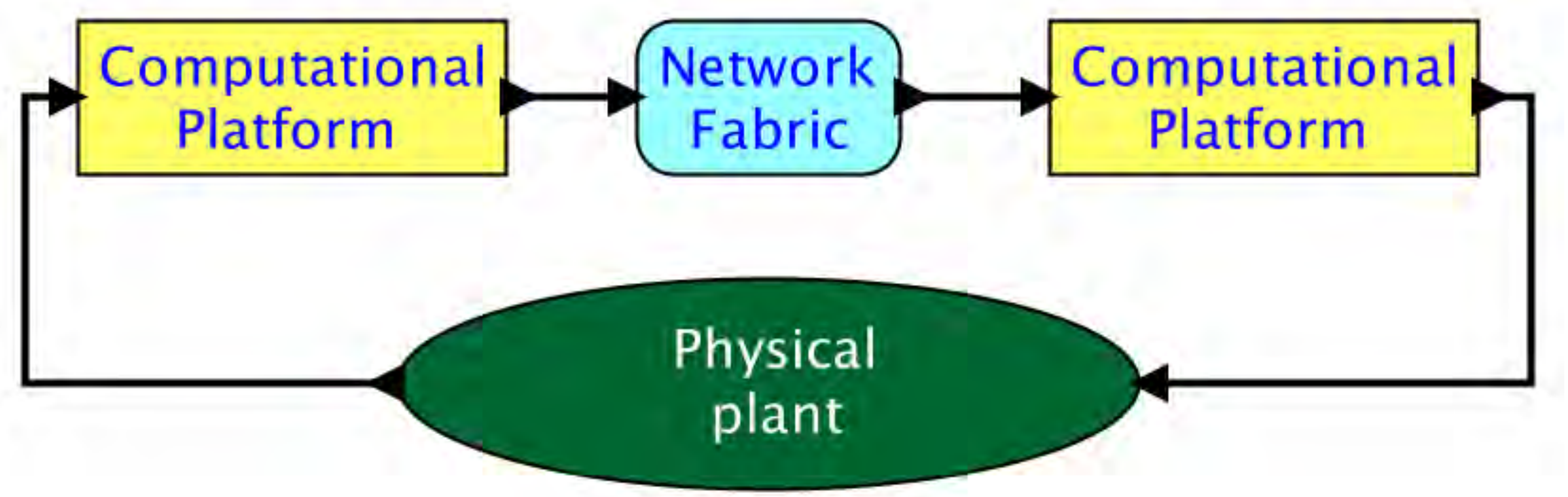
Keynote: CyPhy 2019

Model-Based Design of Cyber Physical Systems, in conjunction with ES Week
New York, October 17-18, 2019

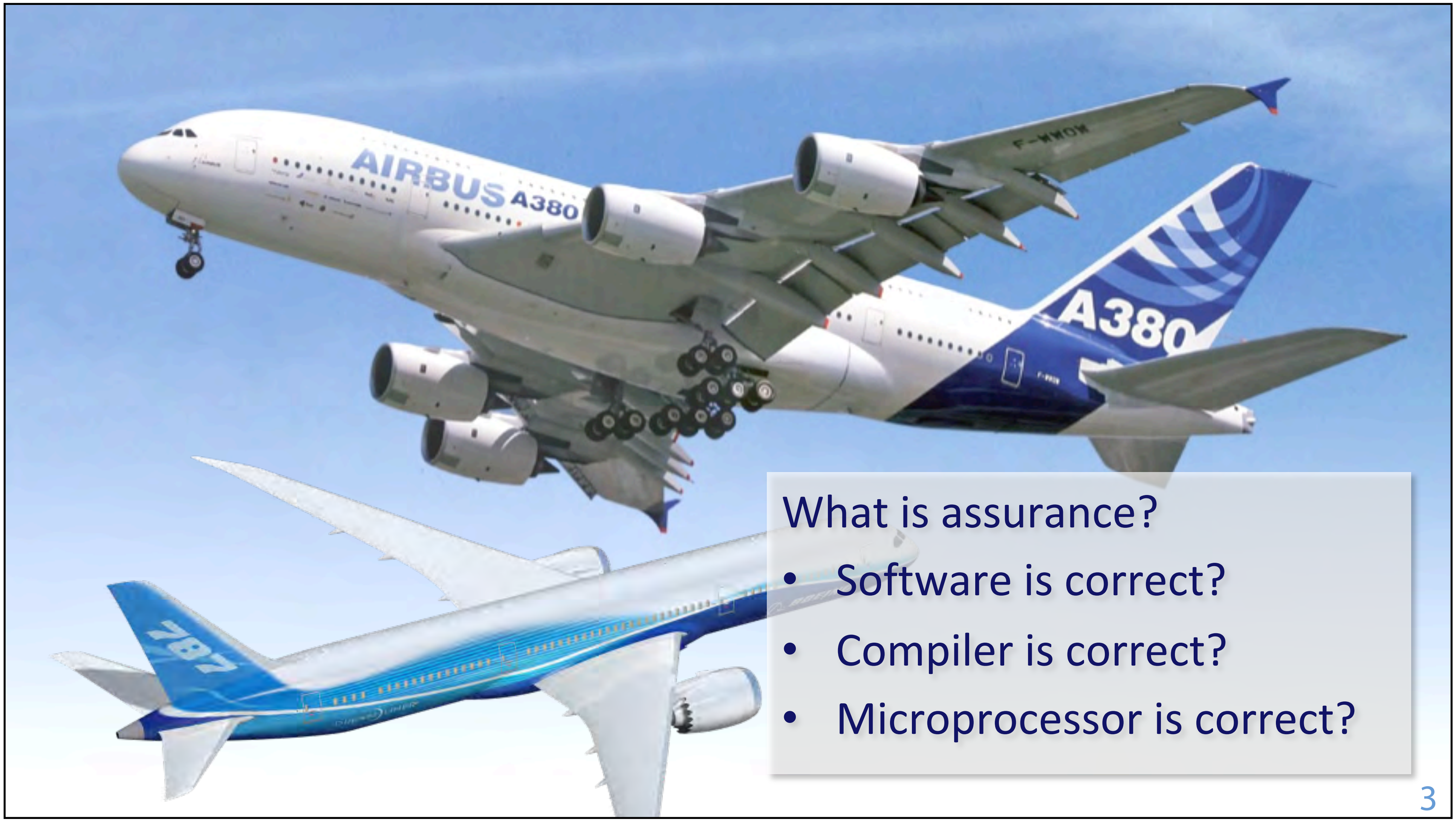
With thanks to:
Marten Lohstroh
Martin Schoeberl
Andrés Goens
Mehrdad Niknami
Christopher Gill
Marjan Sirjani
Matt Weber

University of California at Berkeley

Cyber-Physical Systems



Predictability requires determinacy and depends on timing, including execution times and network delays.



What is assurance?

- Software is correct?
- Compiler is correct?
- Microprocessor is correct?

Correct execution of correct software provides little assurance.

Key Challenges

***Timing* and *Concurrency* in software yield non-repeatable behavior.**

A Simple Challenge Problem

A software component with two inputs:

1. open_door
2. disarm_door

What should it do when it receives a message open_door?

By Christopher Doyle from Horley, United Kingdom - A321 Exit Door, CC BY-SA 2.0

A Simple Challenge Problem

A software component with two inputs:

1. open_door
2. disarm_door

What should it do when it receives a message open_door?

Image from *The Telegraph*, Sept. 9, 2015

Some Solutions (?)

1. Just open the door.

   How much to test? How much formal verification? How to constrain the design of other components?

2. Send a message “ok_to_open?” Wait for responses.

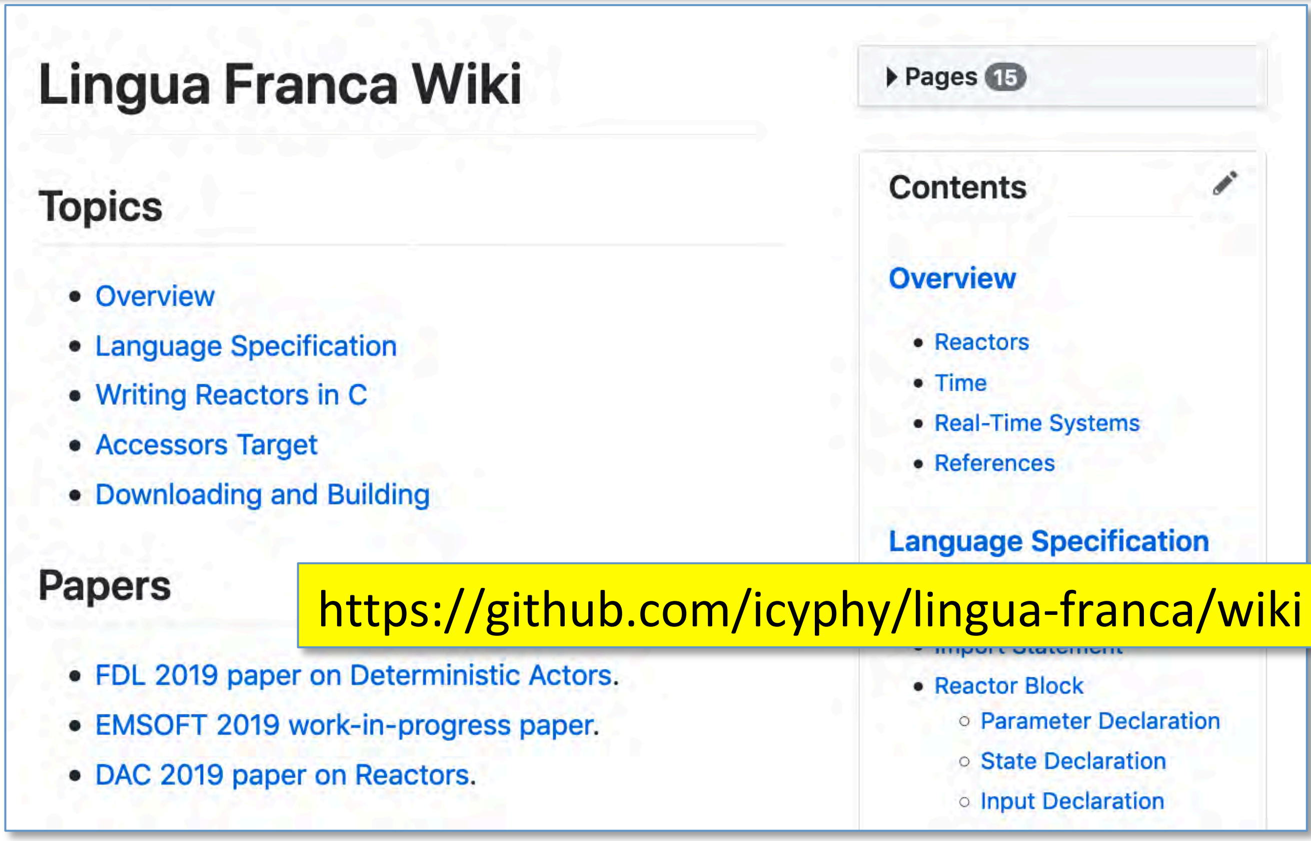
   How many responses? How long to wait? What if a component has failed and never responds?

3. Wait a while and then open.

   How long to wait?

Lingua Franca

A polyglot meta-language for deterministic, concurrent, time-sensitive systems.



Lingua Franca Wiki

Topics

- Overview
- Language Specification
- Writing Reactors in C
- Accessors Target
- Downloading and Building

Papers

- FDL 2019 paper on Deterministic Actors.
- EMSOFT 2019 work-in-progress paper.
- DAC 2019 paper on Reactors.

▸ Pages 15

Contents

Overview

- Reactors
- Time
- Real-Time Systems
- References

Language Specification

- Import Statement
- Reactor Block
  - Parameter Declaration
  - State Declaration
  - Input Declaration

https://github.com/icyphy/lingua-franca/wiki

Hello World

```
target C;
main reactor HelloWorld {
    timer t;
    reaction(t) {=
        printf("Hello World.\n");
    =}
}
```

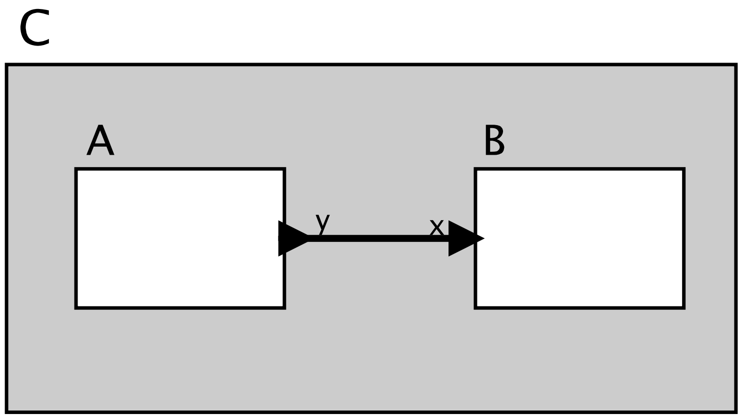
Target language (currently C or JavaScript. Plans for Python, C++, Rust)

Arbitrary code in the target language.

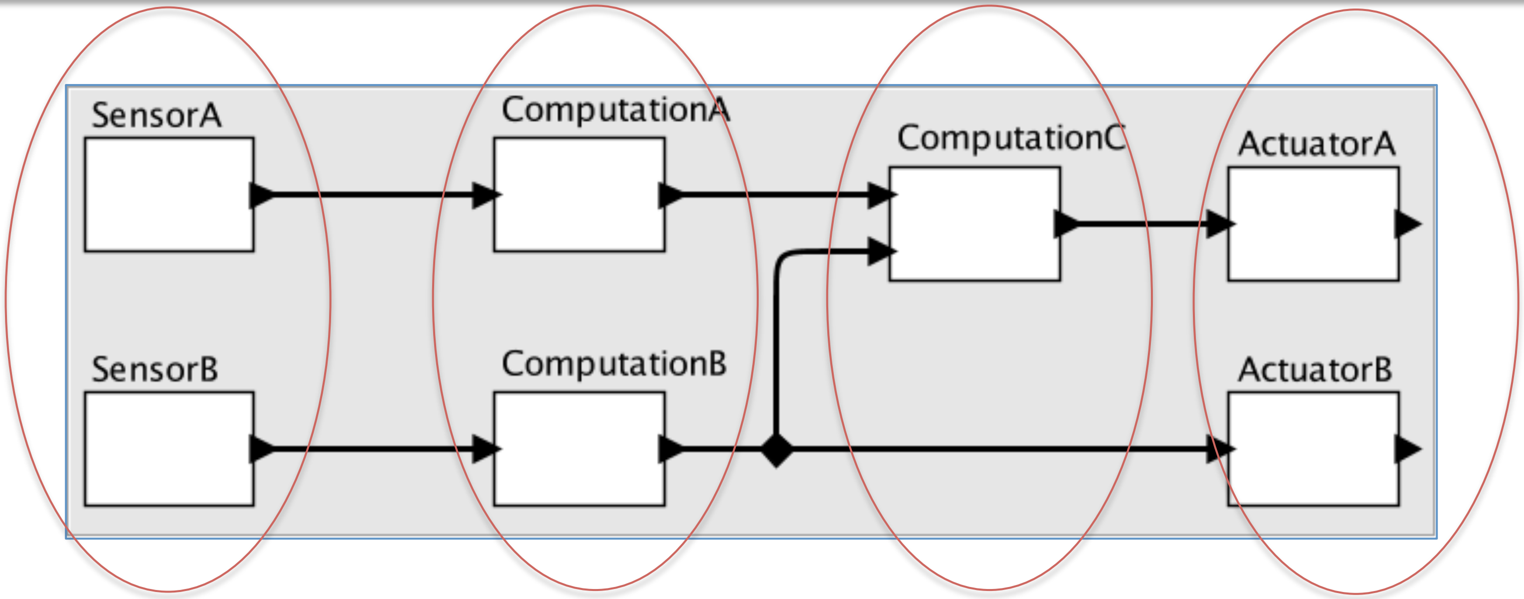
Events of various kinds trigger reactions

Composition

```
reactor A {
  output y;
  ...
}
reactor B {
  input x;
  ...
}
main reactor C {
  a = new A();
  b = new B();
  a.y -> b.x;
}
```



Questions Addressed by Lingua-Franca



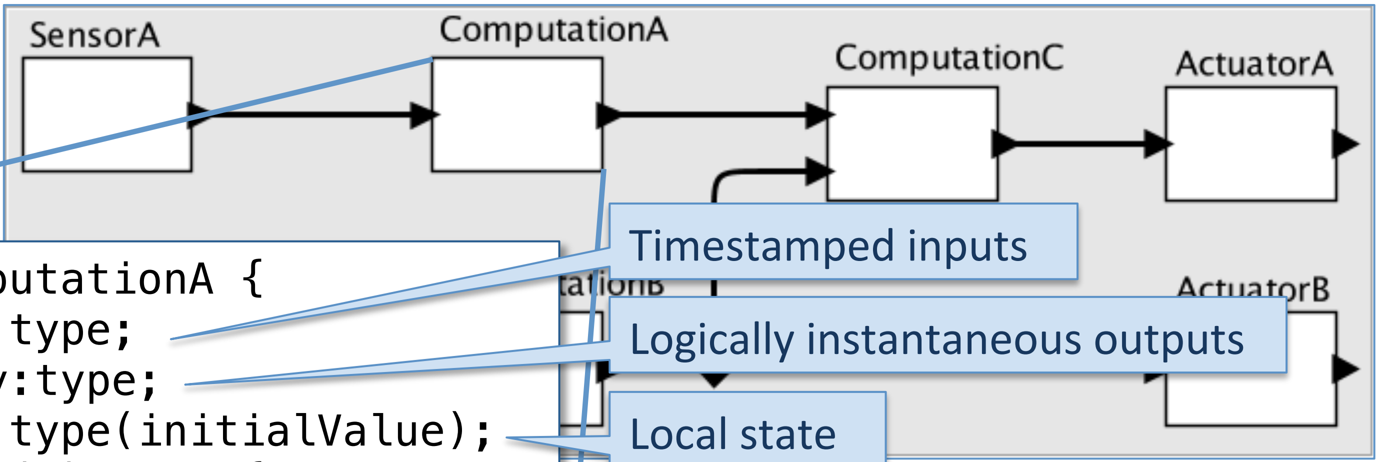
What combinations of periodic, sporadic, behaviors are manageable?

How do execution times affect feasibility? How can we know execution times?

How do we get repeatable and testable behavior even when communication is across networks?

How do we specify, ensure, and enforce deadlines?

Reactors



```
reactor ComputationA {
    input x:type;
    output y:type;
    state s:type(initialValue);
    reaction(x) -> y {=
        Target-language code
        referencing x, y, and s.
    =}
}
```

Timestamped inputs

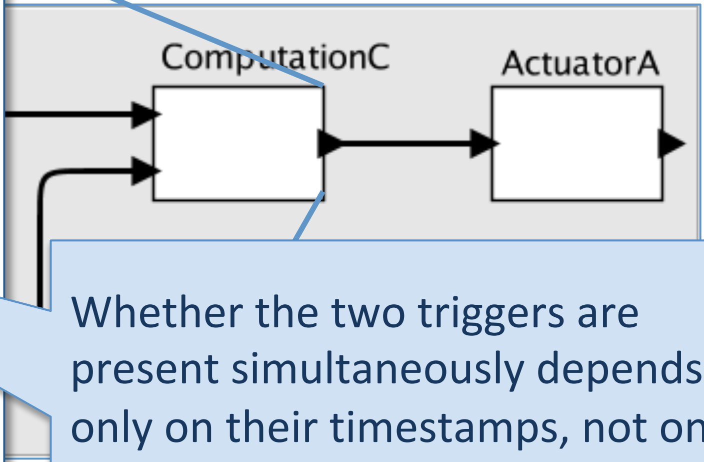
Logically instantaneous outputs

Local state

Reaction signature gives trigger(s) and production

Determinism

```
reactor Add {
    input in1:int;
    input in2:int;
    output out:int;
    reaction(in1, in2) -> out {=
        int result = 0;
        if (in1_is_present) {
            result += in1;
        }
        if (in2_is_present) {
            result += in2;
        }
        set(out, result);
    =}
}
```



Whether the two triggers are present simultaneously depends only on their timestamps, not on on when they are received nor on where in the network they are sent from.

More Deterministic than Hewitt Actors

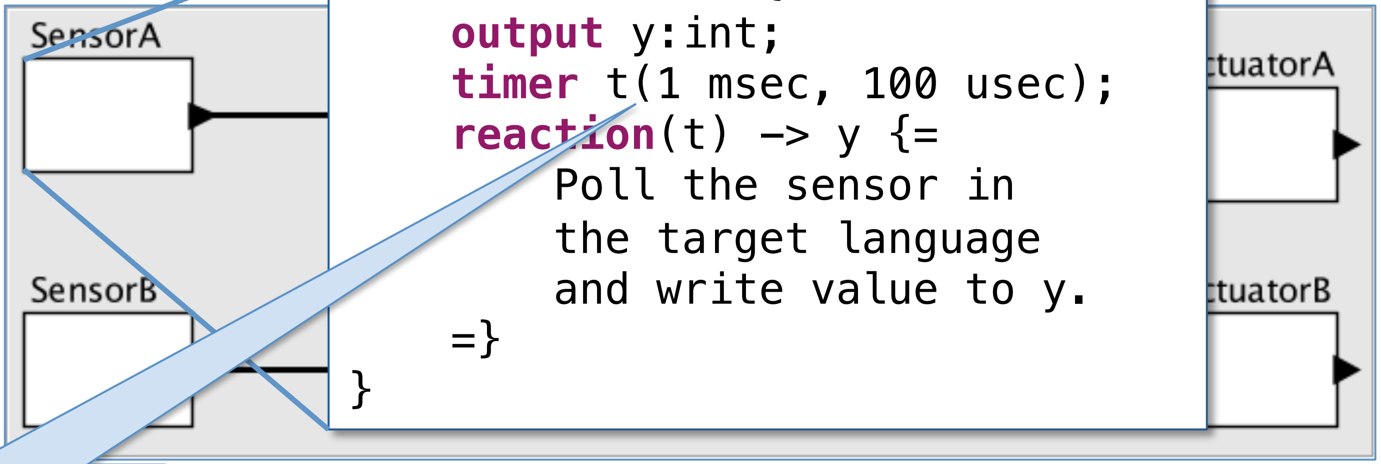
Realizations of Hewitt actors:

- Erlang [Armstrong, et al. 1996]
- Rebeca [Sirjani and Jaghoori, 2011]
- Akka [Roestenburg, et al. 2017]
- Ray [Moritz, et al. 2017]
- …

More Deterministic than Publish and Subscribe

- ROS
- MQTT
- Microsoft Azure
- Google Cloud Pub/Sub
- XMPP
- DDS
- Amazon SNS
- …

Periodic Behavior

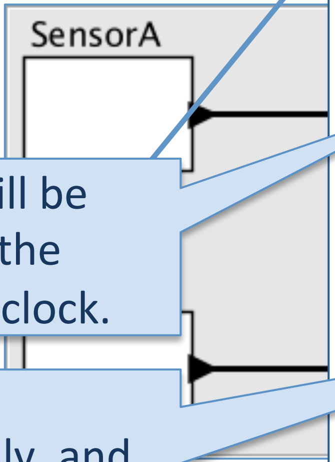


```
reactor SensorA {
    output y:int;
    timer t(1 msec, 100 usec);
    reaction(t) -> y {=
        Poll the sensor in
        the target language
        and write value to y.
    =}
}
```

Time as a first-class data type.

In our C target, timestamps are unsigned 64-bit integers representing the number of nanoseconds since Jan. 1, 1970 (if the platform has a clock) or the number of nanoseconds since starting (if not).

Event-Triggered Behavior



```
reactor SensorB {
    output y:int;
    physical action a:int;
    timer start;
    reaction(start) -> a {=
        Set up an interrupt service
        routine that will call:
        schedule(a, 0, value);
    =}
    reaction(a) -> y {=
        set(y, *(*int)(a->payload));
    =}
}
```

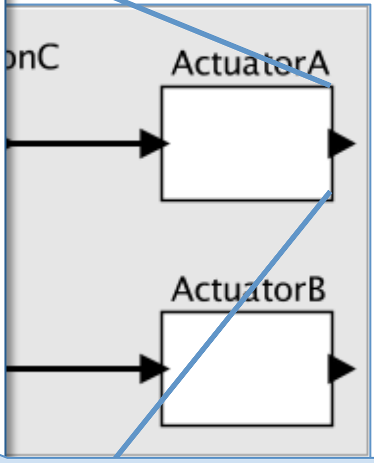
Timestamp will be derived from the local physical clock.

ISR executes asynchronously, and schedule() function is thread safe.

Deadlines

Timestamp is derived from logical time, not physical time.

```
reactor Container {
    logical action panic;
    reaction(panic) -> d.x {=
        printf("**** Deadline miss detected.\n");
        Can pass modified value to d.x or
        pass nothing.
    =}
    s = new ComputationC();
    d = new ActuatorA();
    s.y -> d.x;
    deadline(d.x, 100 usec, panic);
}
```



Deadline violation is handled by the container (composite) reactor.

Deadline is violated if the input d.x triggers more than 100 usec (in physical time) after the timestamp of the trigger.

Status

https://github.com/icyphy/lingua-franca

- Eclipse/Xtext-based IDE
- C and JavaScript targets
- C code runs on Mac, Linux, Windows, and bare iron
- Command-line compiler
- Regression test suite
- Wiki documentation

Performance

Behaviors of the C target in the regression tests running on a 2.6 GHz Intel Core i7 running MacOS:

- Up to 23 million reactions per second (43 ns per).
- Linear speedup on four cores.
- Code size is tens of kilobytes.

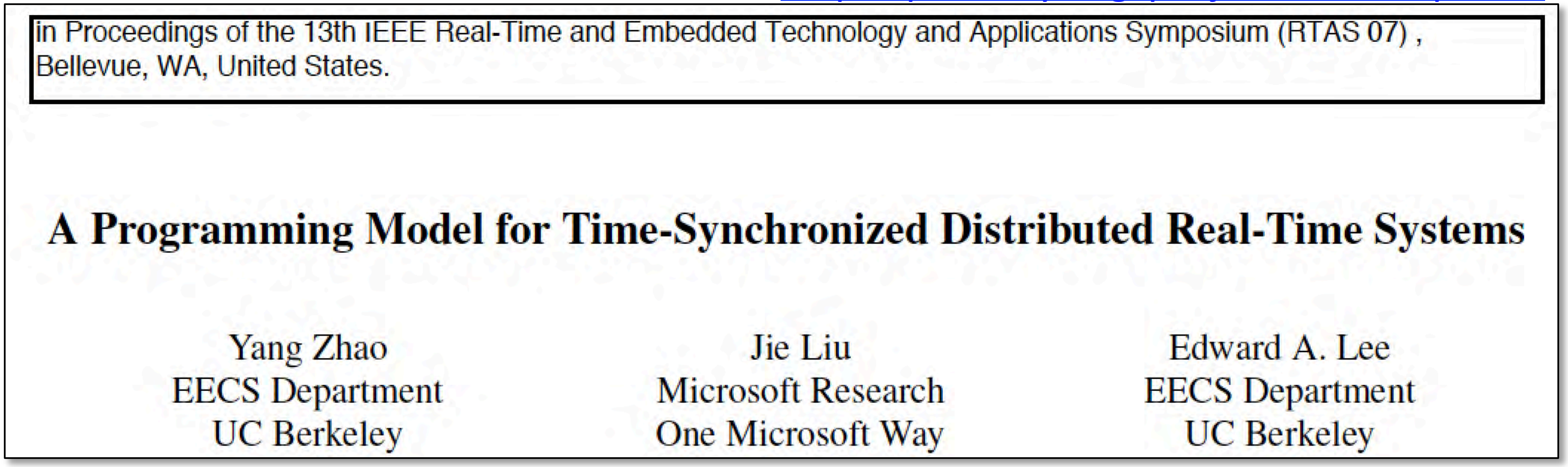
Work in Progress

- EDF scheduling on multicore.
- Distributed execution based on Ptides.
- Targeting PRET machines for real time.
- Leverage Google’s Protobufs and gRPC.
  - Complex datatypes
  - Polyglot systems

Background Project:
PTIDES: Deterministic Distributed Real Time

Engineering models for distributed real time systems, with cost effective realizations.

http://ptolemy.org/projects/chess/ptides



in Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 07) , Bellevue, WA, United States.

A Programming Model for Time-Synchronized Distributed Real-Time Systems

Yang Zhao
EECS Department
UC Berkeley

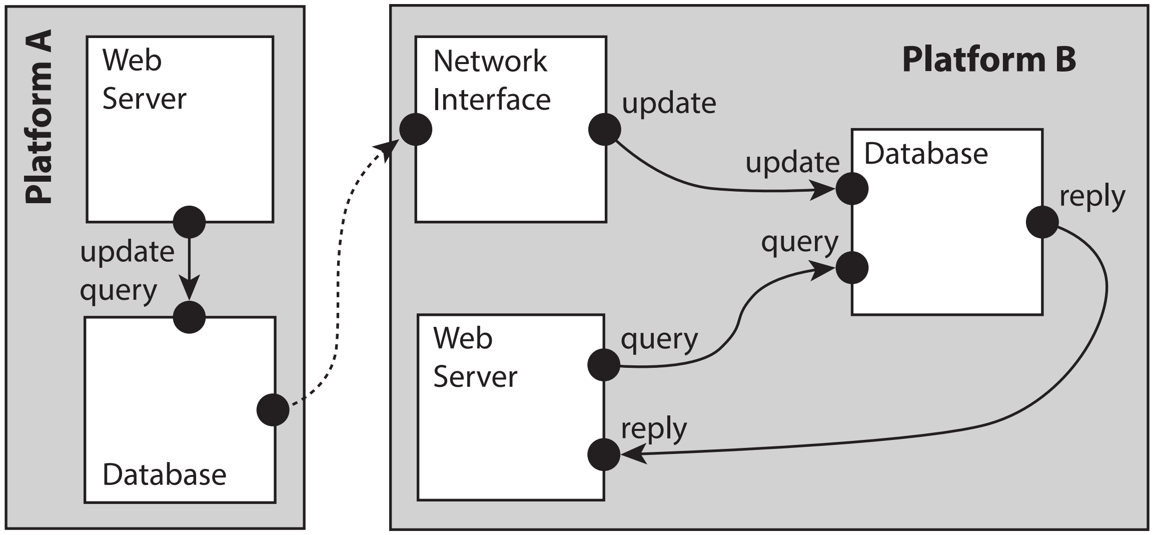
Jie Liu
Microsoft Research
One Microsoft Way

Edward A. Lee
EECS Department
UC Berkeley

Distributed Execution in a Spanner-like Application

Ptides/Spanner

- Bounded clock synchronization error.
- Bounded network latency.
- Bounded execution times or deadlines.



Background Project: PRET Machines

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

http://ptolemy.org/projects/chess/pret

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

Computing

+

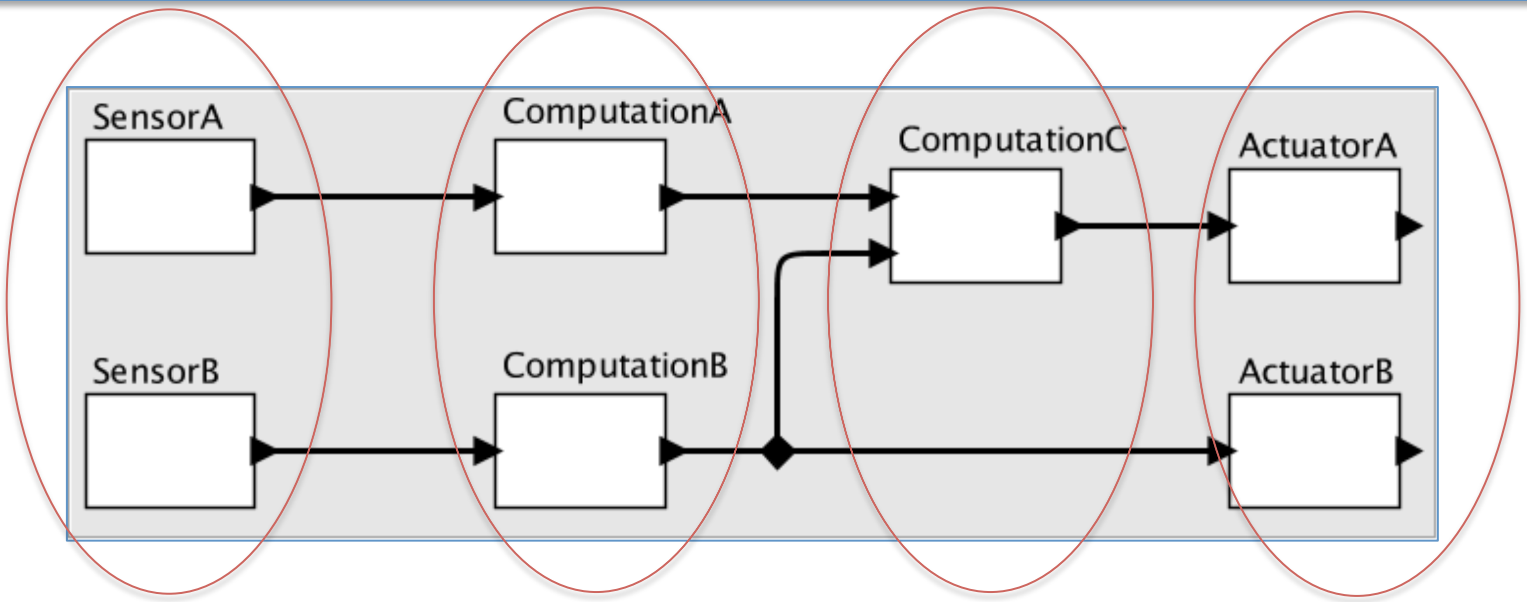
= PRET

With time

PRET targets

- Dedicate hard real-time threads to reactors that have deadlines or that send messages to reactors that have deadline.
- Synthesize schedules for hard real-time threads to just meet deadlines.
- Verify whether deadlines can be violated.

Questions Addressed by Lingua-Franca



What combinations of periodic, sporadic, behaviors are manageable?

How do execution times affect feasibility? How can we know execution times?

How do we get repeatable and testable behavior even when communication is across networks?

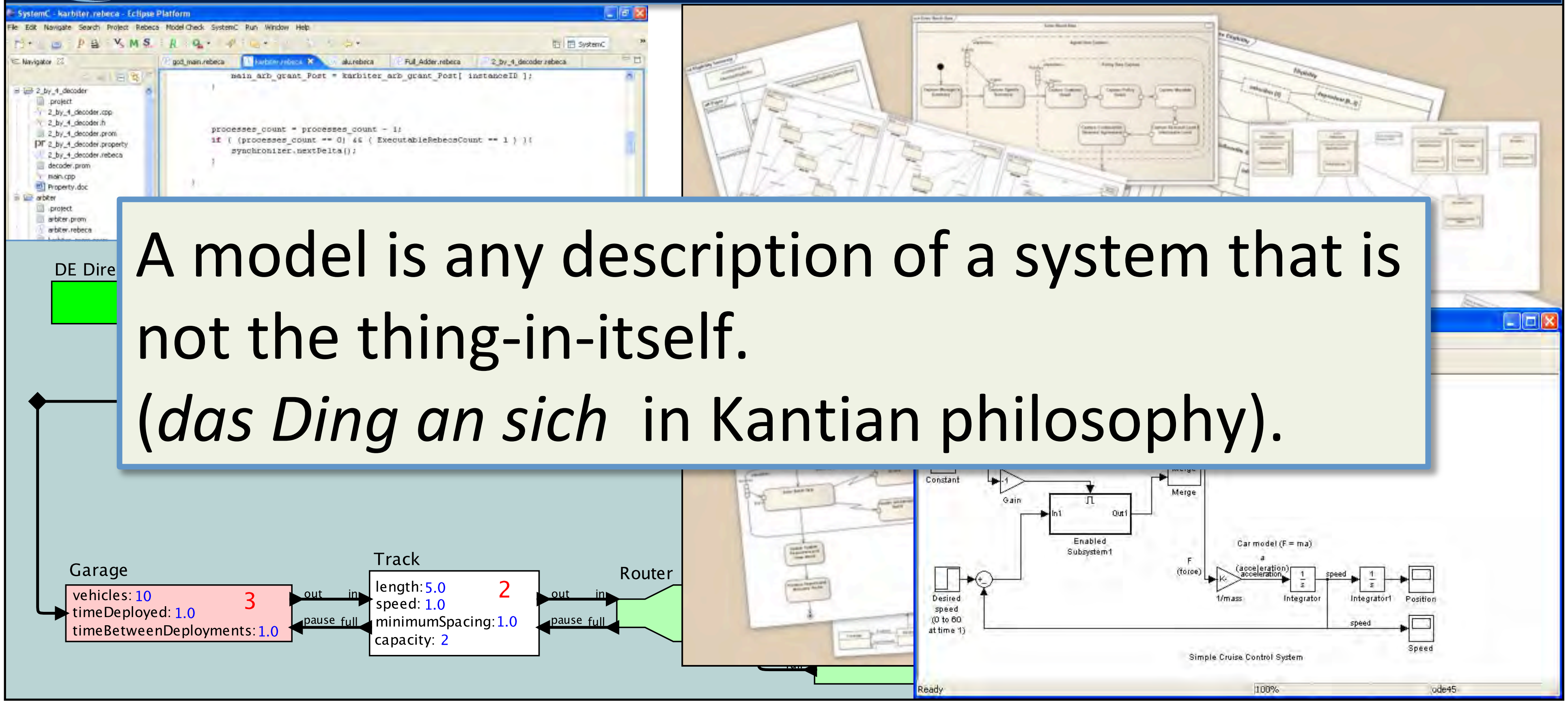
How do we specify, ensure, and enforce deadlines?

Stepping Back: The Big Picture

A Lingua-Franca program is a *model* of the behavior we want to build. The code generator translates this model into an *implementation*.

Contrast this with the prevailing approach, where people build models of their implementation and then attempt to verify or validate those models.

What is a Model?



```
        main_arb_grant_Post = karbiter_arb_grant_Post[ instanceID ];
    }


    processes_count = processes_count - 1;
    if ( (processes_count == 0) && ( ExecutableRebecsCount == 1 ) ){
        synchronizer.nextDelta();
    }

}
```

vehicles: 10
timeDeployed: 1.0
timeBetweenDeployments: 1.0
3

length: 5.0
speed: 1.0
minimumSpacing: 1.0
capacity: 2
2

A model is any description of a system that is not the thing-in-itself.
(*das Ding an sich* in Kantian philosophy).

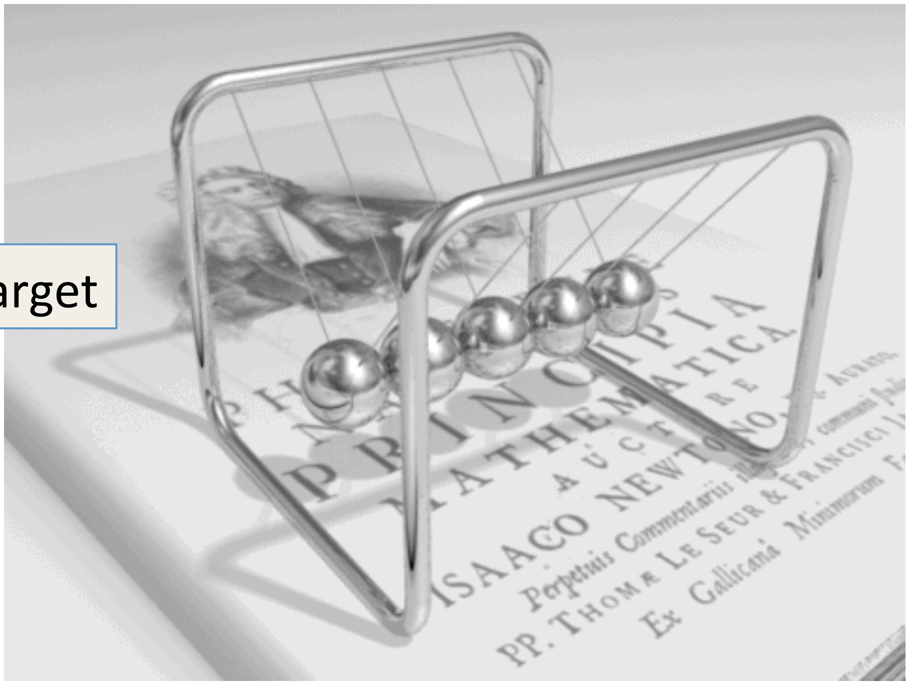
A Model

$$x(t) = x(0) + \int_0^t v(\tau)d\tau$$

$$v(t) = v(0) + \frac{1}{m}\int_0^t F(\tau)d\tau.$$

The model

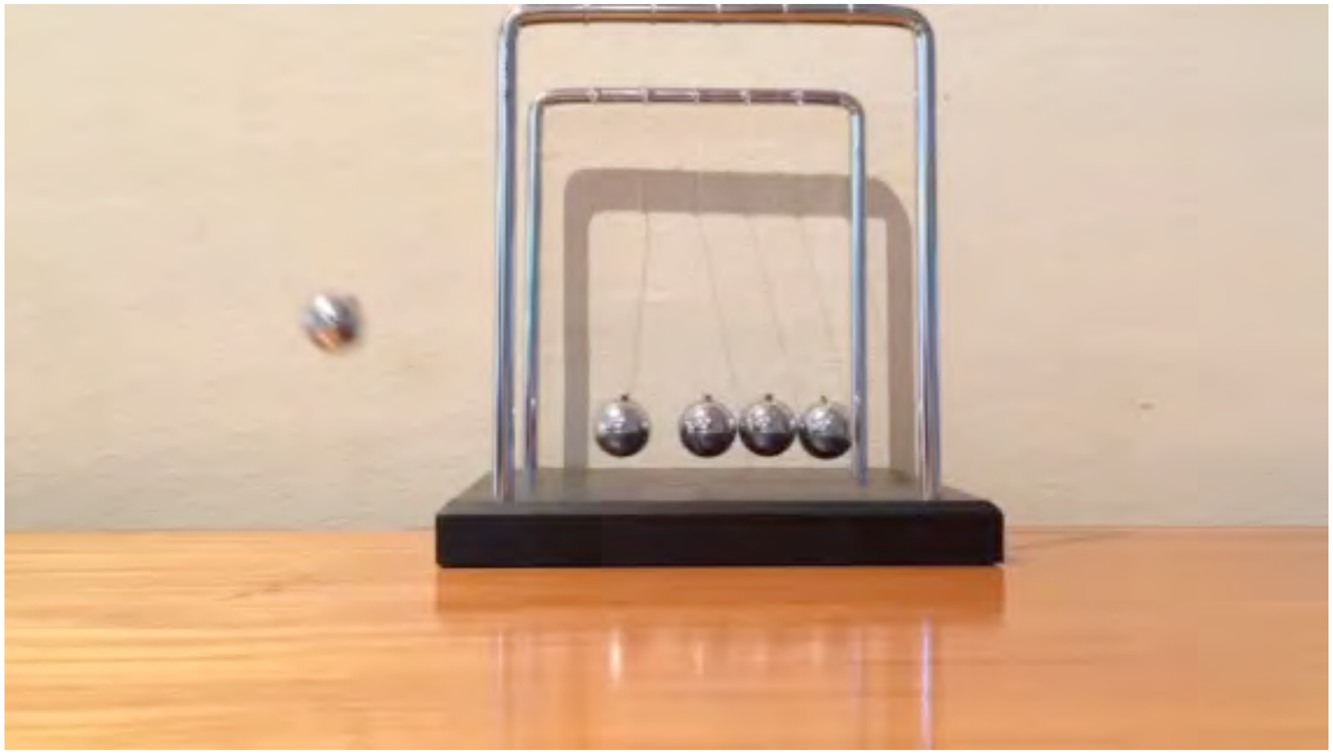
The target



In this example, the *modeling universe* is calculus and Newton's laws in a time and space continuum.

Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

A Physical Realization



A few things we need to model to explain this behavior:

- Plastic deformation
- Acoustic propagation
- Stretching of strings
- Gravity
- …

The Value of Models

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.
- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

A scientist asks, “Can I make a model for this thing?”
An engineer asks, “Can I make a thing for this model?”

Model Fidelity

- To a *scientist*, the model is flawed.
- To an *engineer*, the realization is flawed.

I’m an engineer…

Perhaps we should be trying to make systems behave like our specifications rather than building systems and then trying to figure out how they behave.

Useful Models and Useful Things

"Essentially, all models are wrong,
but some are useful."

Box, G. E. P. and N. R. Draper, 1987: *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics, Wiley.

"Essentially, all system implementations
are wrong, but some are useful."

Lee and Sirjani, "What good are models," FACS 2018.

Changing the Question

Is the question whether we can build models describing the behavior of cyberphysical systems?

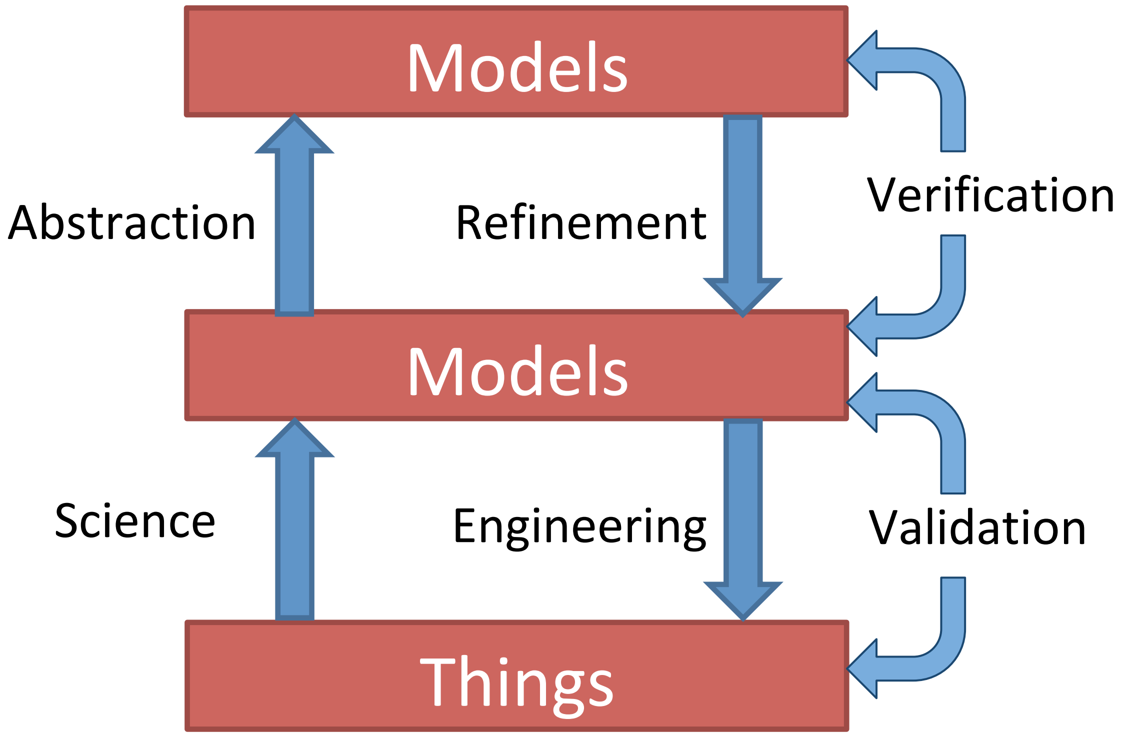
Or

Is the question whether we can make cyberphysical systems that behave like our models?

Towards Engineering-Model-Based Design of Real-Time Systems

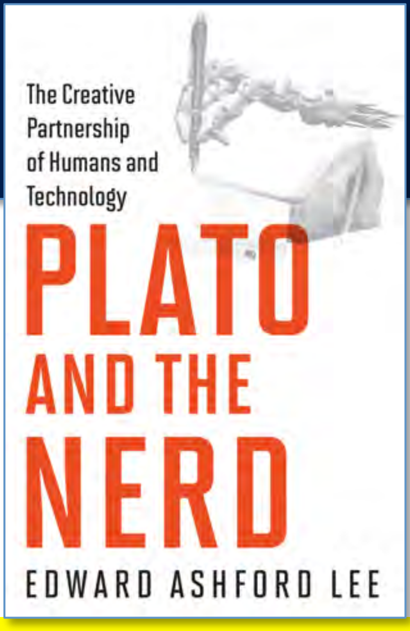
Per Boehm:

- Am I building the right product? (validation)
- Am I building the product right? (verification)



Conclusions

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.
- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.



MIT Press, 2017

My message:
Do less science and more engineering.

https://github.com/icyphy/lingua-franca
http://ptolemy.org/projects/chess/pret
http://ptolemy.org/projects/chess/ptides