iCyPhy

Model-Based Design of Cyber-Physical Systems

Edward A. Lee

Design, Modeling, and Architecture of Complex Industrial Systems (COMASIC) Masters Program

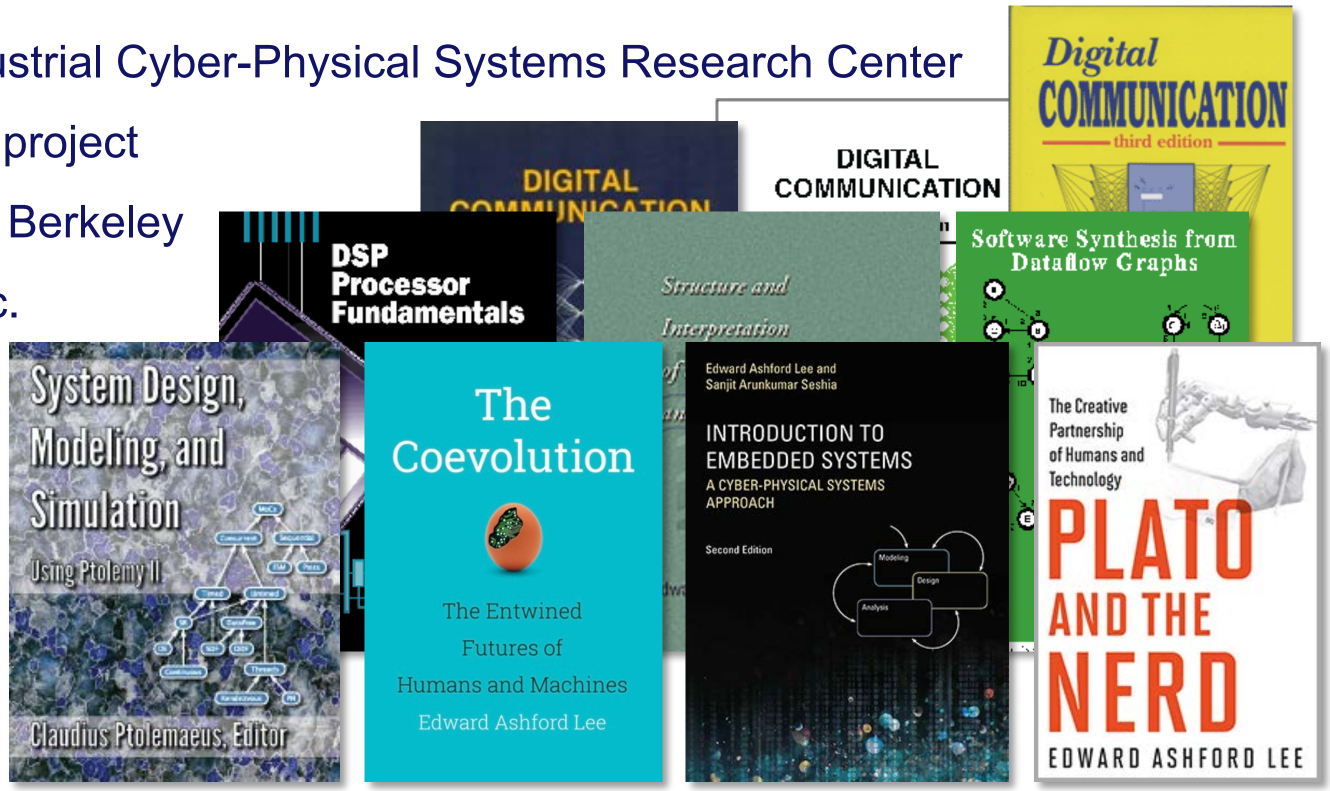
Université Paris-Saclay

Saclay, France, Jan. 23, 2020

University of California, Berkeley

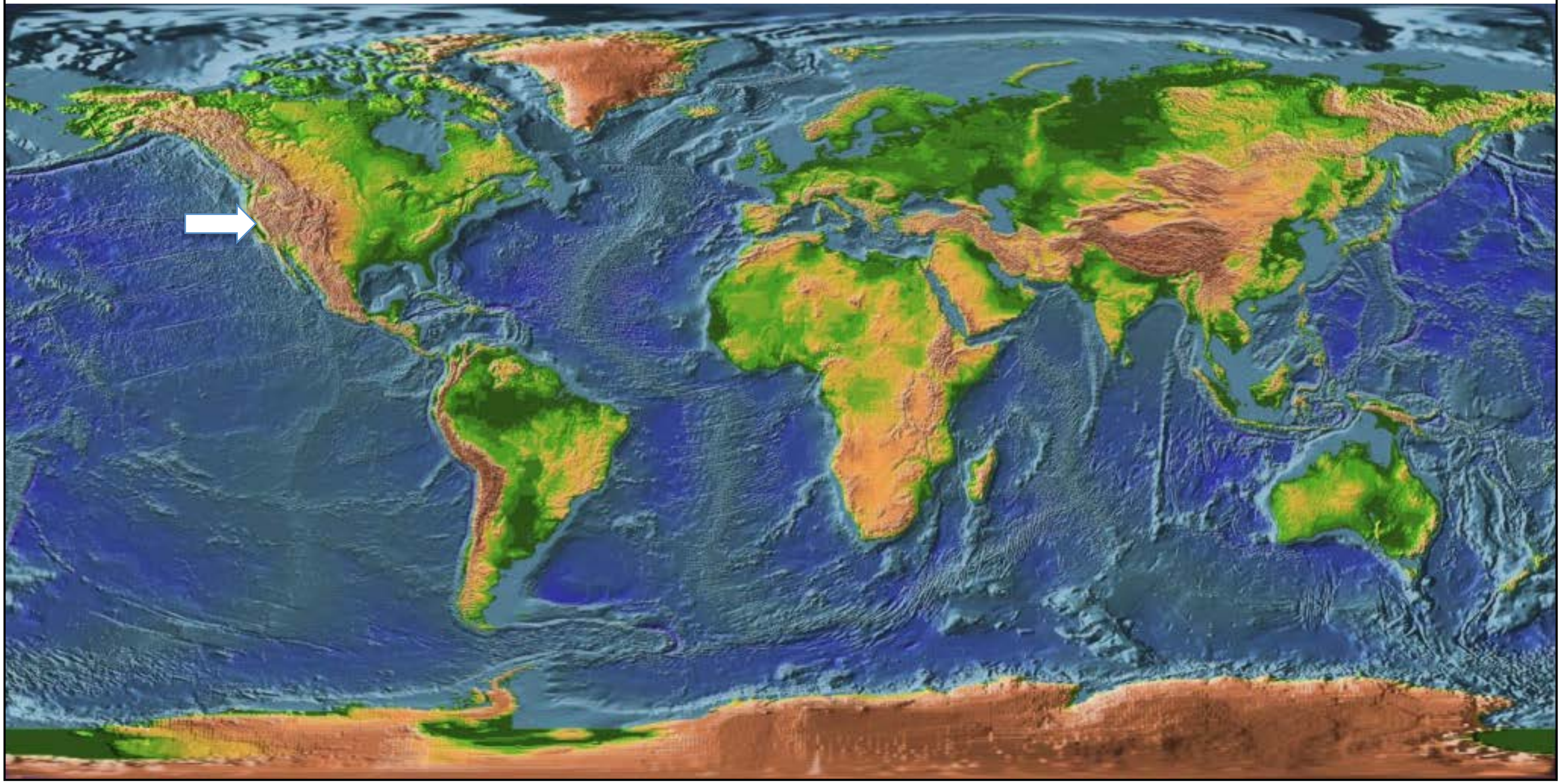
Introducing Edward A. Lee

- BS (Yale), SM (MIT), PhD (Berkeley)
- Bell Labs in the early 1980s
- Berkeley EECS faculty since 1986
- Working on embedded software since 1978
- Director of iCyPhy, Industrial Cyber-Physical Systems Research Center
- Director of the Ptolemy project
- Former Chair of EECS, Berkeley
- Co-founder of BDTI, Inc.
- Books…



http://ptolemy.org/~eal
eal@berkeley.edu

Location



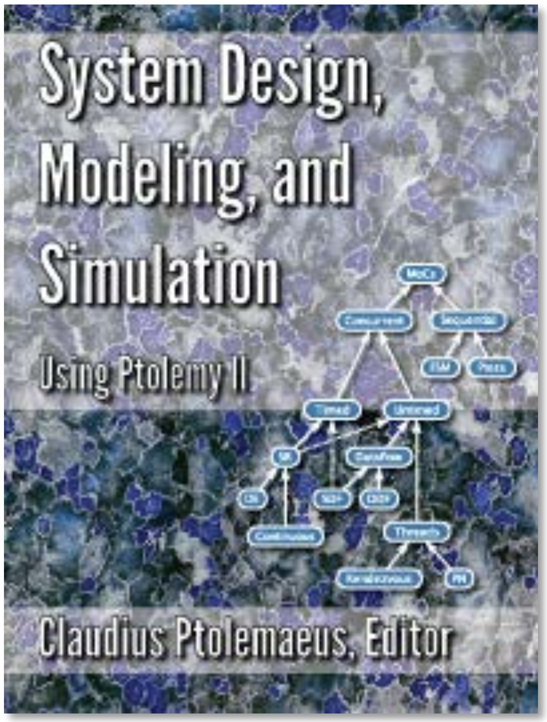
The University of California at Berkeley

Disclaimer

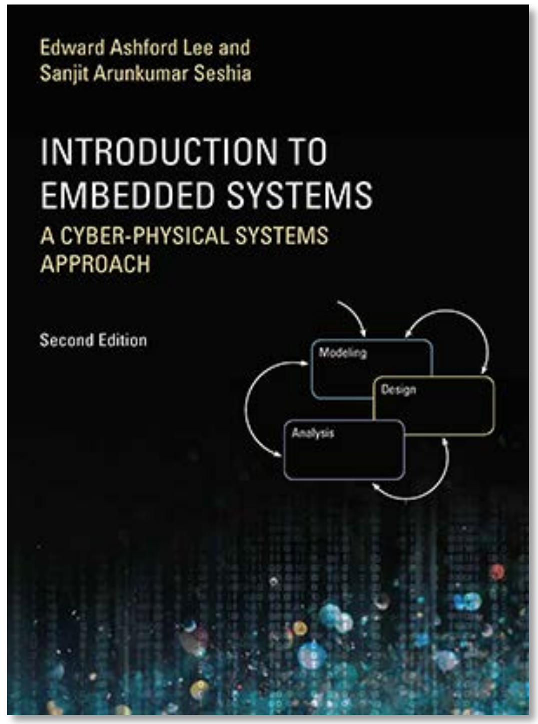
This is not a survey of the field.

I will give you a narrow Berkeley view with a lot of opinions and personal perspectives.

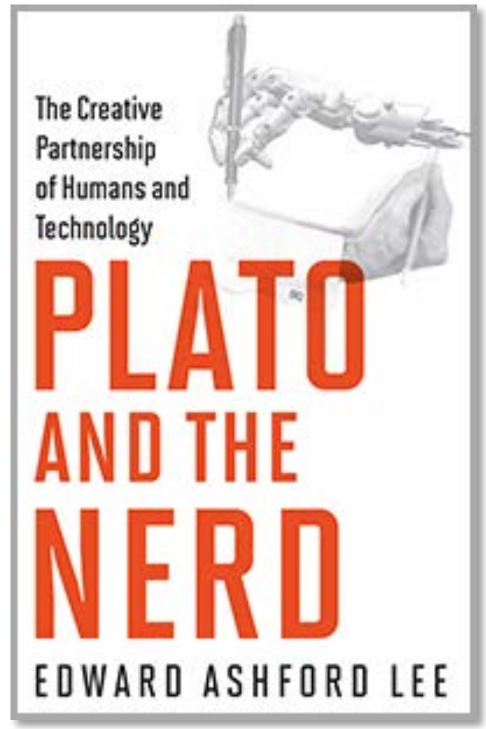
Resources



http://ptolemy.org/systems



http://leeseshia.org



http://platoandthenerd.org

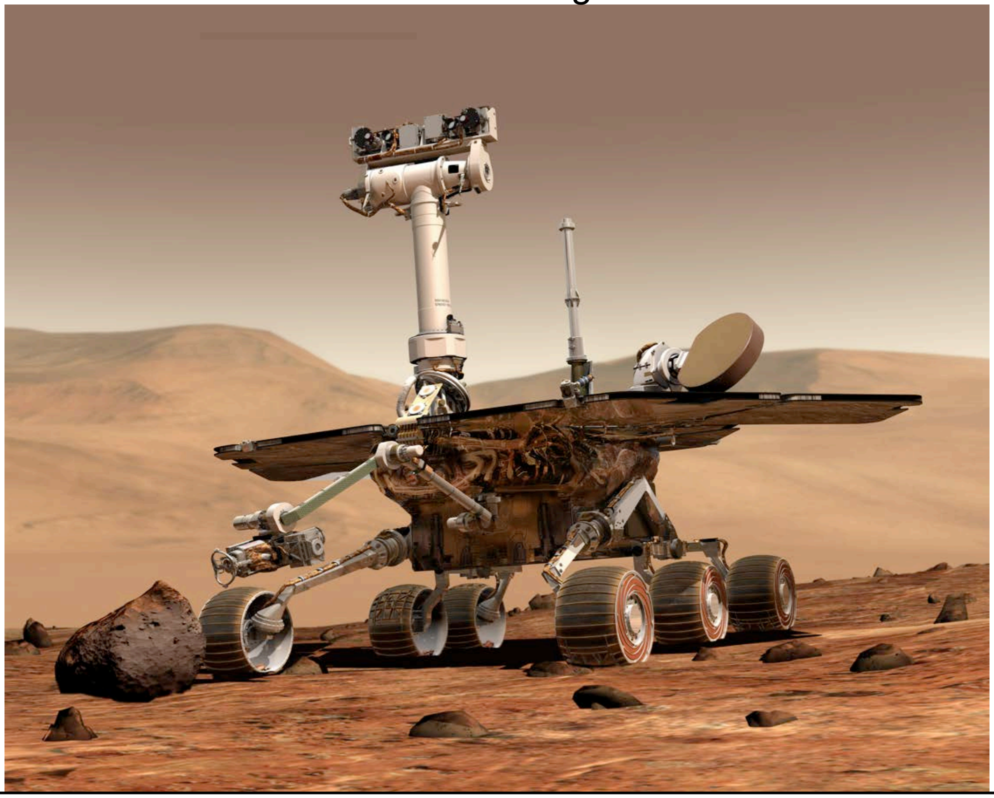
Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

Cyber-Physical Systems

Orchestrating networked computational resources and physical systems.

Image: Wikimedia Commons



Roots:

- Term coined around 2006 by Helen Gill at the National Science Foundation in the US.
- **Cyberspace**: attributed William Gibson, who used the term in the novel Neuromancer.
- **Cybernetics**: coined by Norbert Wiener in 1948, to mean the conjunction of control and communication.

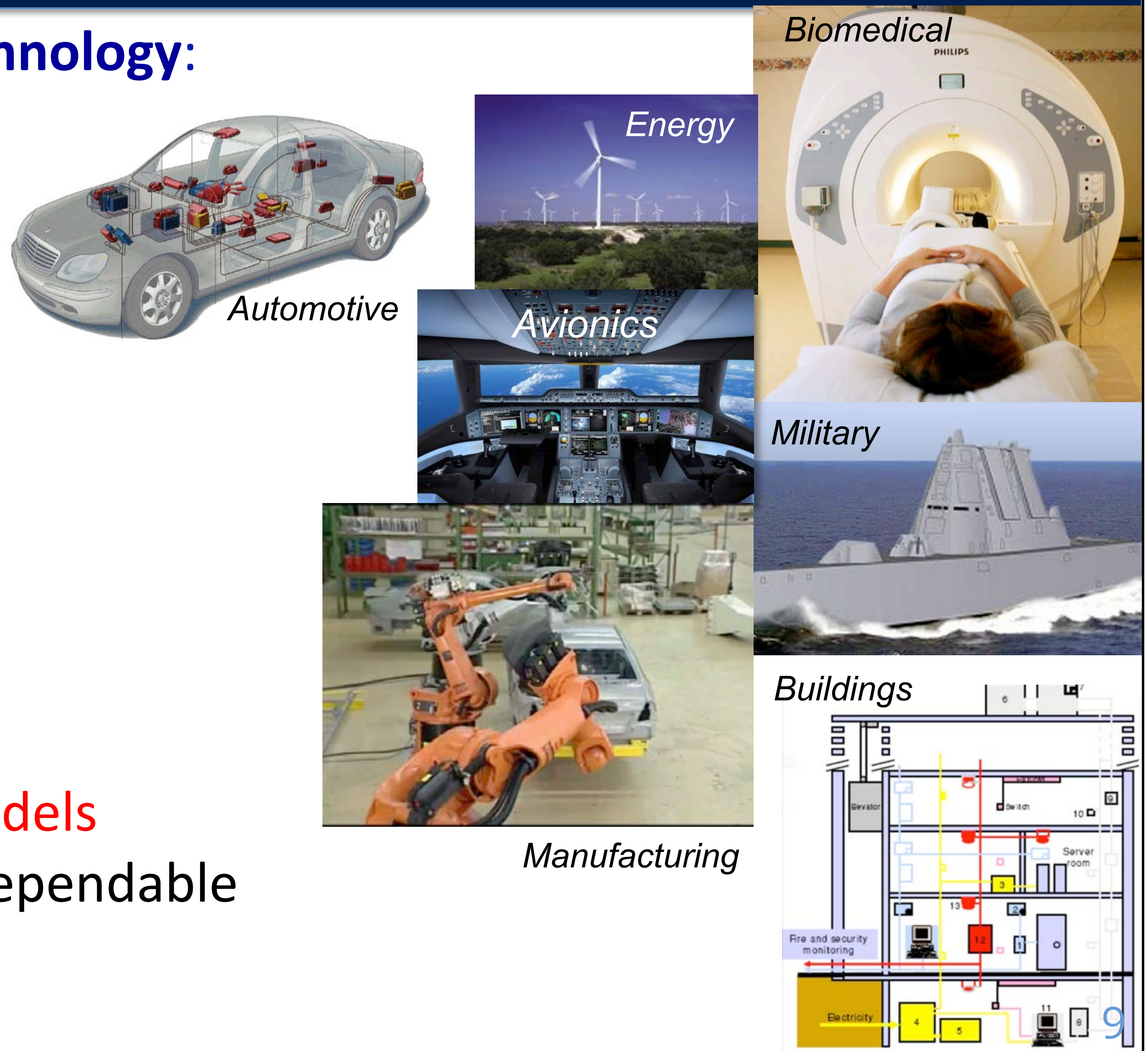
Cyber-Physical Systems

Not just information technology:

- Cyber + Physical
- Computation + Dynamics
- Security + Safety

Properties:

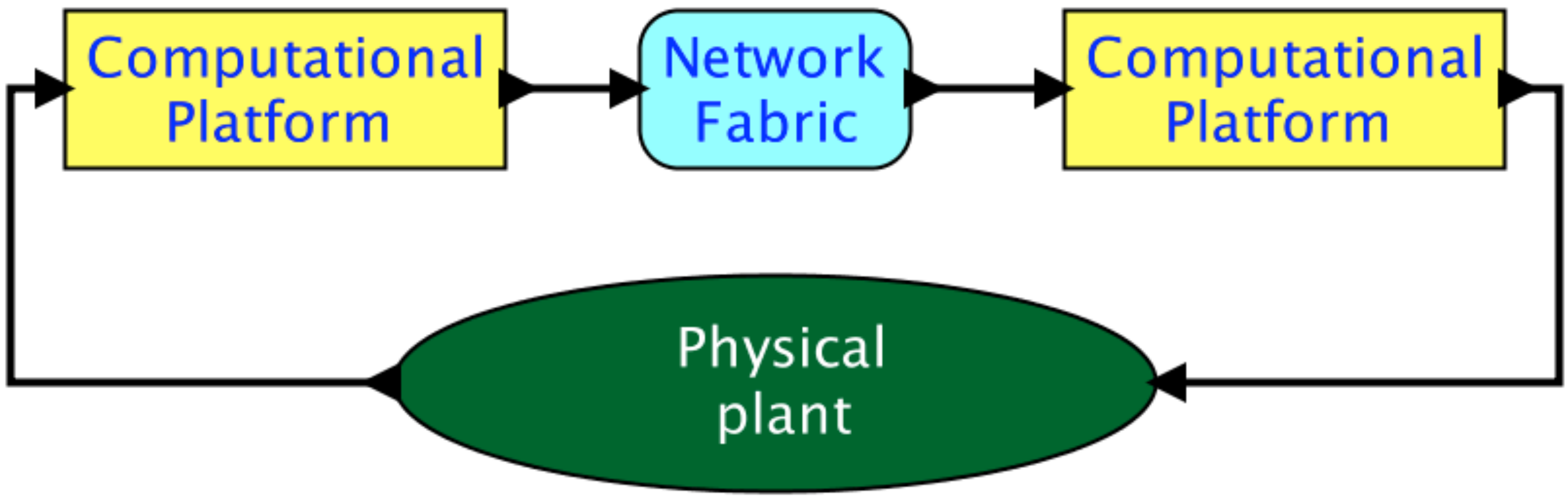
- Highly dynamic
- Safety critical
- Uncertain environment
- Physically distributed
- Sporadic connectivity
- Resource constrained



We need engineering models and methodologies for dependable cyber-physical systems.

Lee, Berkeley

Cyber-Physical Systems Pattern



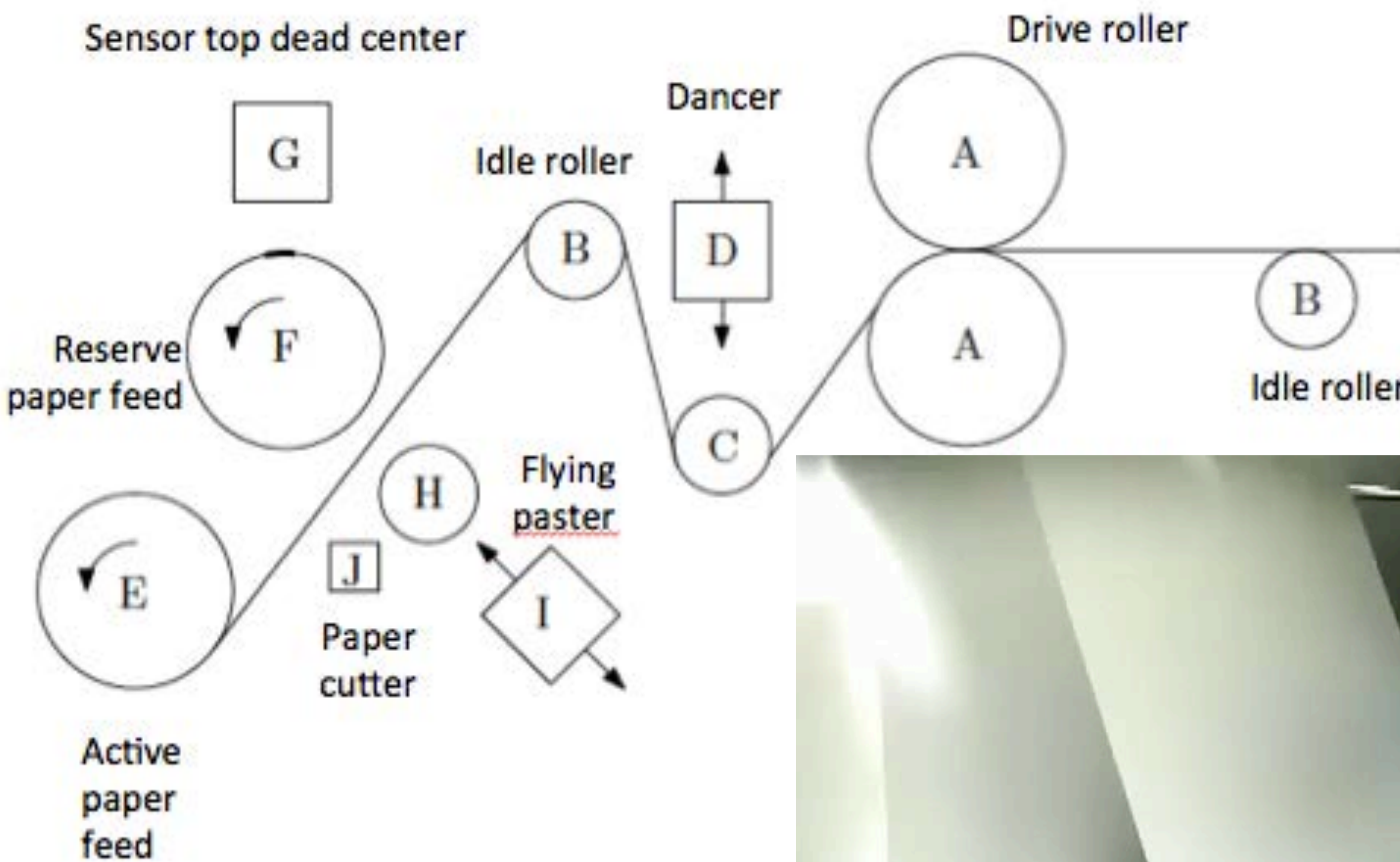
Often safety critical, real time, and resource constrained.

Example

Hundreds of microcontrollers orchestrating depositing ink and slicing paper flying through the machine at 100 km/hr.

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.

Example – Flying Paster



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

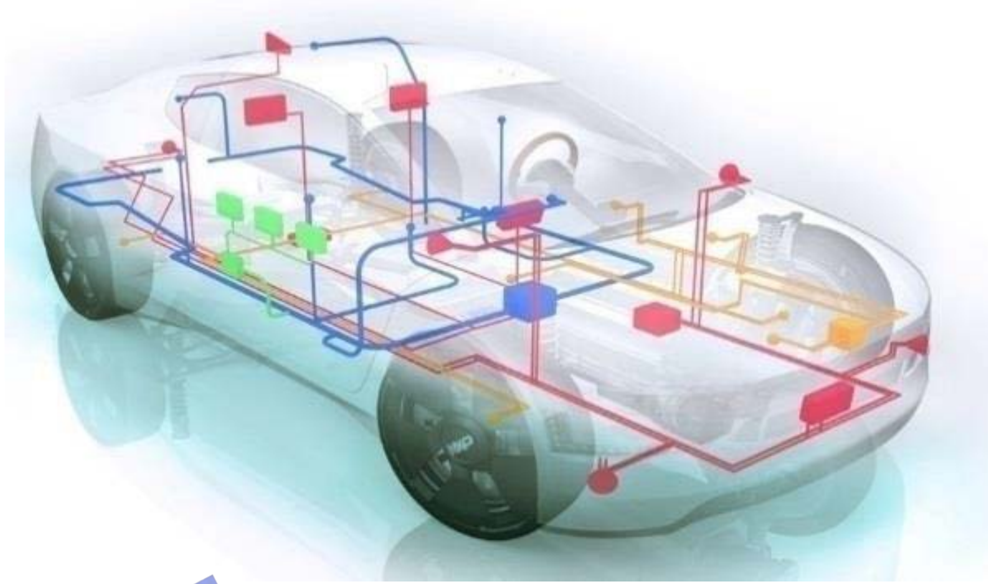
Example – Flying Paster

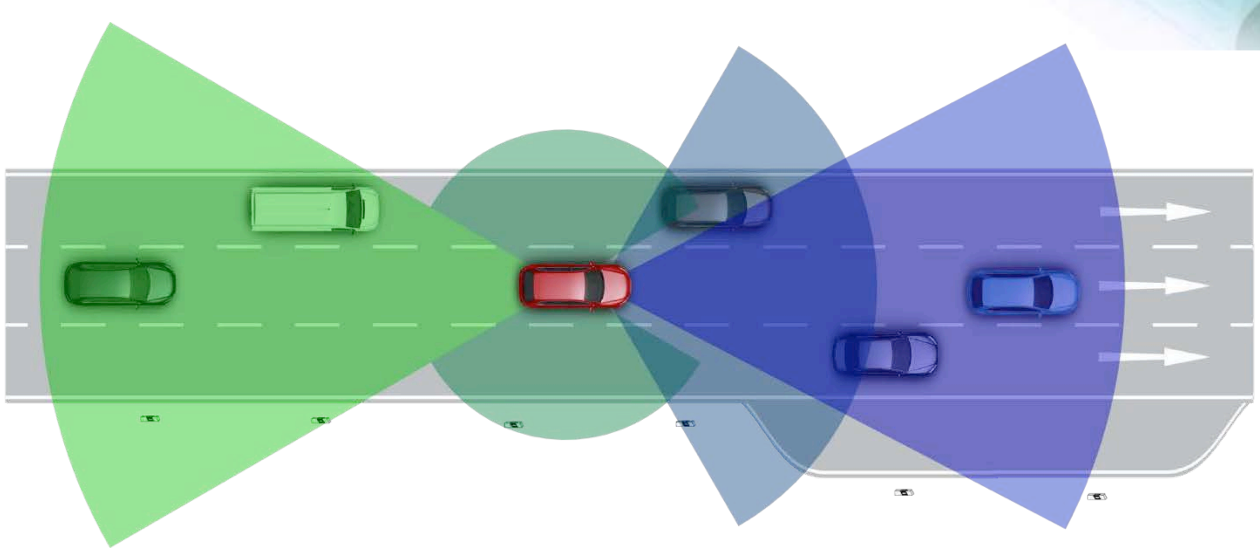
Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.html

CPS Challenge Problem: *Prevent This*

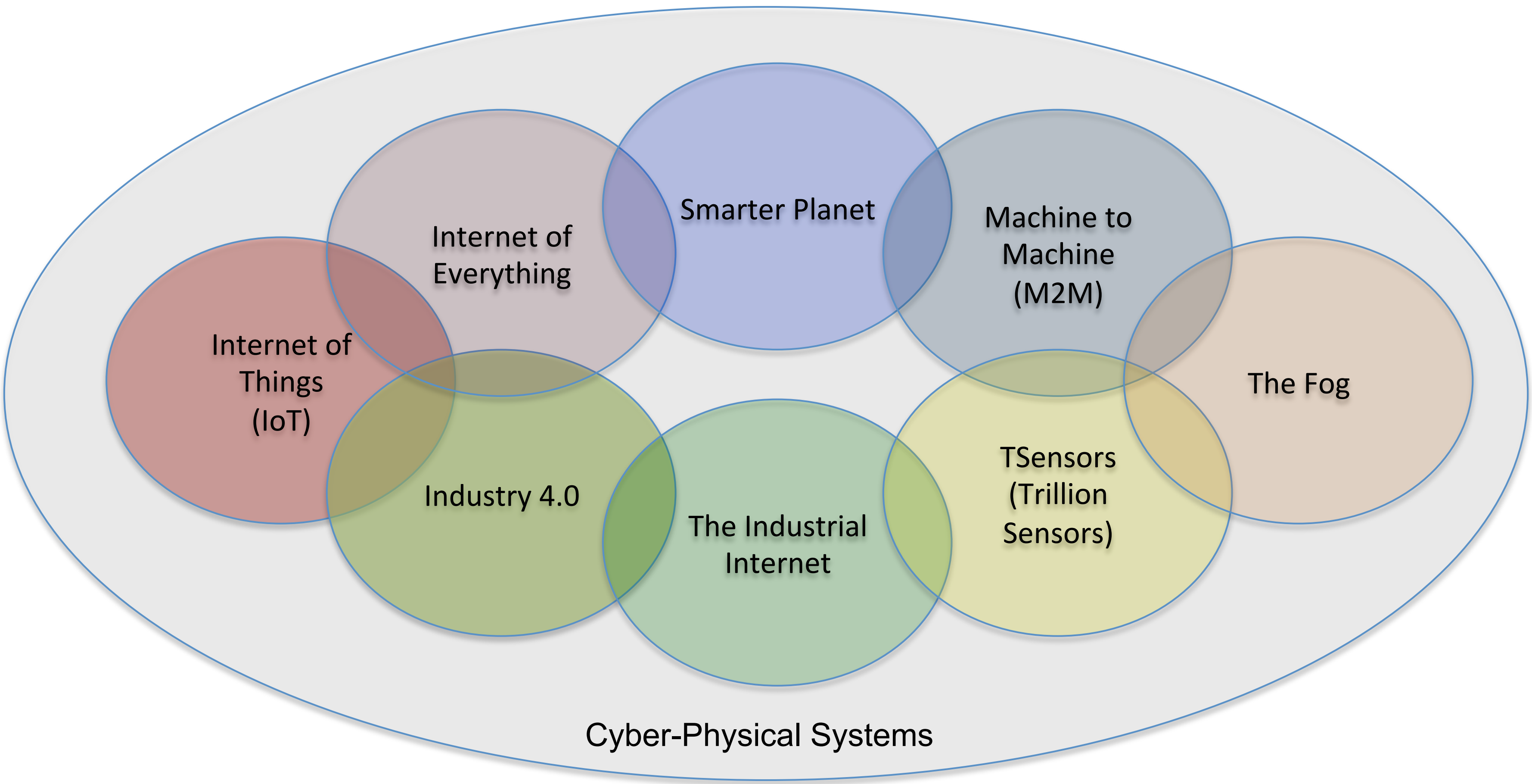
Automotive CPS and Societal Challenges

- Safer Transportation
- Reduced Emissions
- Smart Transportation
- Energy Efficiency
- Climate Change
- Human-Robot Collaboration





CPS-related terms



Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

Software Problems

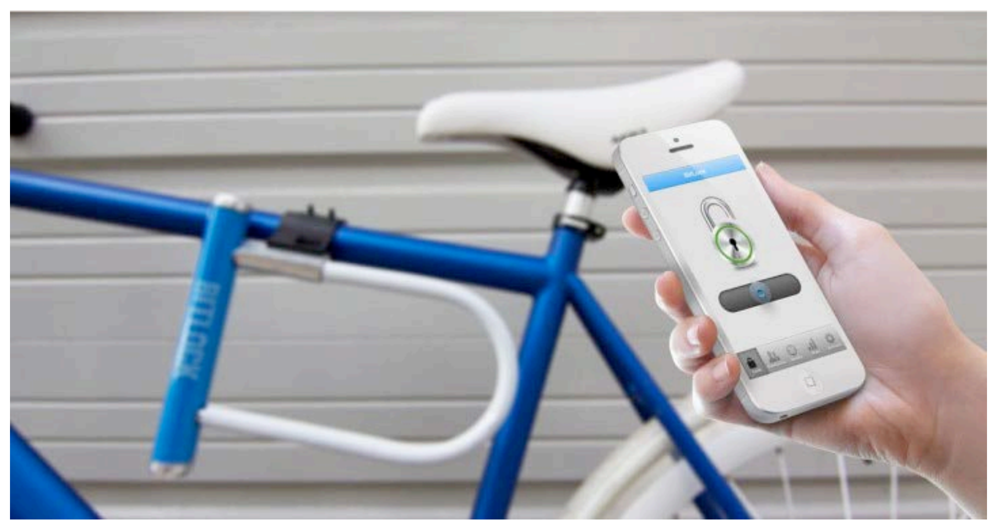
https://twitter.com/dakotathekat/status/744504193847791616

Bad Design

Matt Haughey
@mathowie

Follow

If you need cheering up today, know that using my new IoT bike lock that only works with a phone took me 10min to unlock my bike after lunch



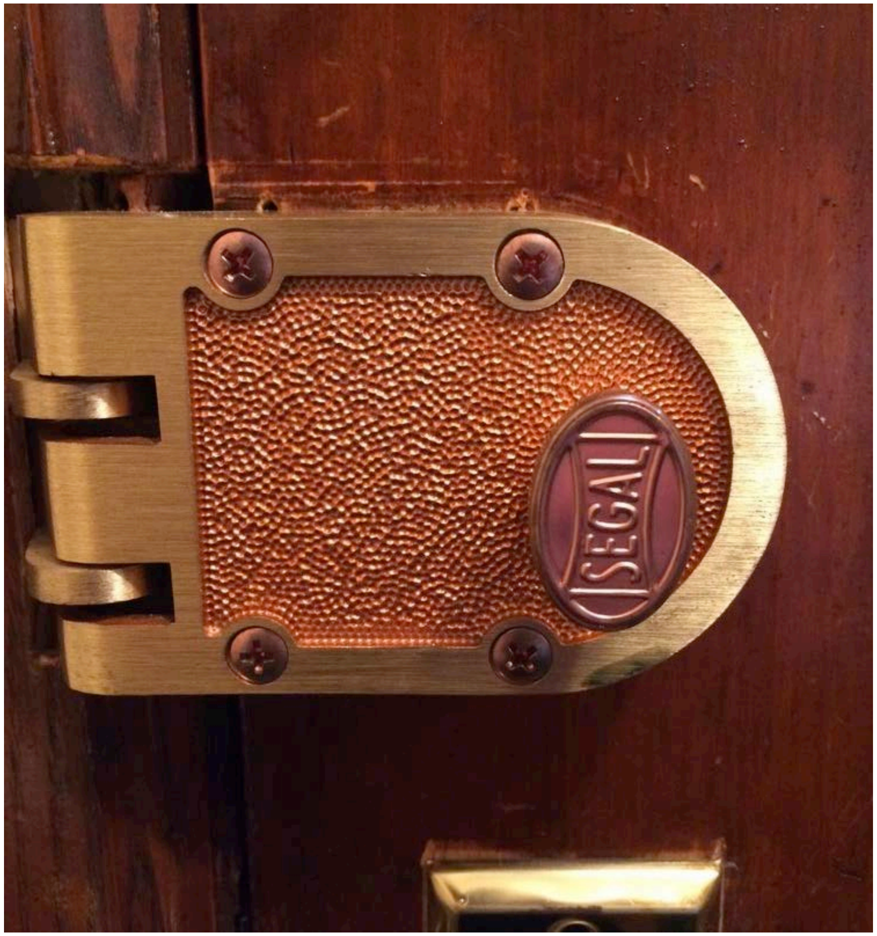
My egg tray doesn’t like my Wi-Fi network. That may sound like a Mad Lib, but I’m serious. It took me 15 minutes to correctly pair Quirky’s $15 Egg Minder with the iPhone app, which gives you a count of remaining eggs. Yet when I removed eggs from the tray to make breakfast, one of them remained virtually present. I guess you could say the app was… scrambled.

Updates

Lightbulb firmware update

“My bulbs are at 7E. I keep getting prompted to update every once in a while. About 70% of the time I get an upgrade failed message. The rest of the time I get the update completed message, but bulbs still show 7E. “

Lifespan



Segal Lock.
Lifespan: ~100 years

August Bluetooth Lock.
Lifespan:?

Lee, Berkeley, with thanks to Bjoern Hartmann

Irreproducible Results

This would eventually become a recurring theme with my thermostat. In the middle of winter it began disconnecting, frequently overnight — even when there was a solid internet connection — and didn't have a backup mode. I'd wake up seeing my own breath, then spend hours rebooting the thermostat, boiler, and router to get it working again. The only way to control the gadget is via the app, so when it breaks you're really screwed.

The thermostat company later released a second version of its device with a wall control to avoid that no-backup-when-app-breaks situation, but it was another $150 on top of what I'd already spent trying to bring smarts to my heating. Out of frustration, I got it anyway.

Lee, Berkeley, with thanks to Bjoern Hartmann

Problems are not just annoying

NASA's Toyota Study Released by Dept. of Transportation released in 2011 found that Toyota software was “untestable.”

Possible victim of unintended acceleration.

Security Risk

The New York Times

© 2016 The New York Times Company NEW YORK, SATURDAY, OCTOBER 22, 2016

Lee, Berkeley

New Weapons Used in Attack On the Internet

By NICOLE PERLROTH

SAN FRANCISCO — Major websites were inaccessible to people across wide swaths of the United States on Friday after a company that manages crucial parts of the internet's infrastructure said it was under attack.

Users reported sporadic problems reaching several websites, including Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud and The New York Times.

The company, Dyn, whose servers monitor and reroute internet traffic, said it began experiencing what security experts called a distributed denial-of-service attack just after 7 a.m. Reports that many sites were inaccessible started on the East Coast, but spread westward in three waves as the day wore on and into the evening.

A Simple Challenge Problem

A software component on a microprocessor in an aircraft door provides two network services:

1. "open"
2. "disarm"

Assume state is closed and armed.

What should it do when it receives a request "open"?

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

A Simple Challenge Problem

A software component on a microprocessor in an aircraft door provides two network services:

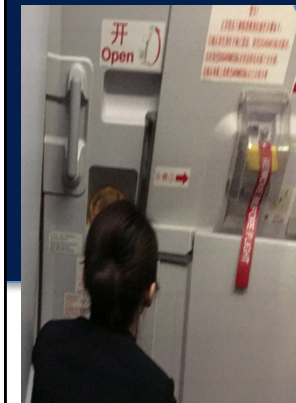
1. "open"
2. "disarm"

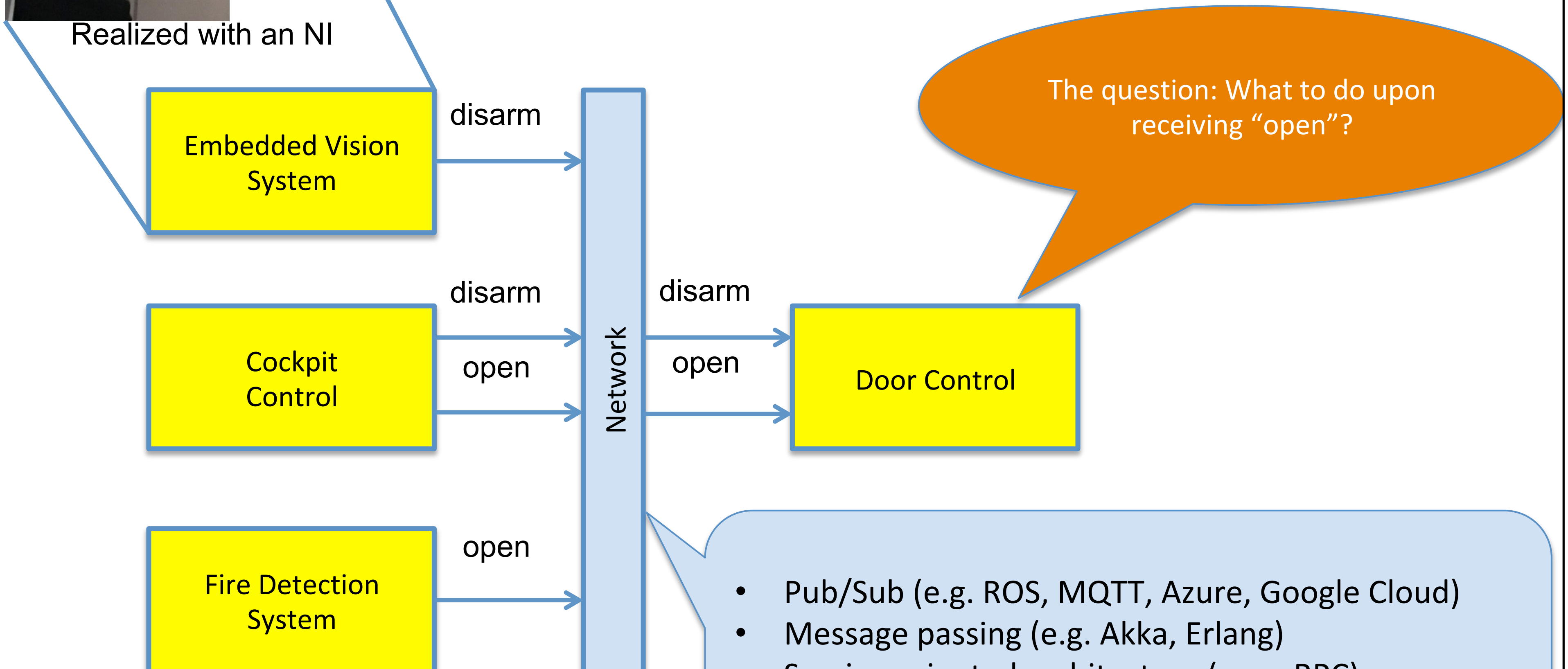
Assume state is closed and armed.

What should it do when it receives a request "open"?

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

Possible Architectures





- Pub/Sub (e.g. ROS, MQTT, Azure, Google Cloud)
- Message passing (e.g. Akka, Erlang)
- Service-oriented architecture (e.g. gRPC)
- Shared memory (e.g. Linda)

Some Solutions (?)

1. Just open the door.

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

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?

Better go read all of Lamport’s papers.

Avionics

What is assurance?

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

Correct execution of correct software provides little assurance.

CCA 2.0
Boeing Dreamscape

Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

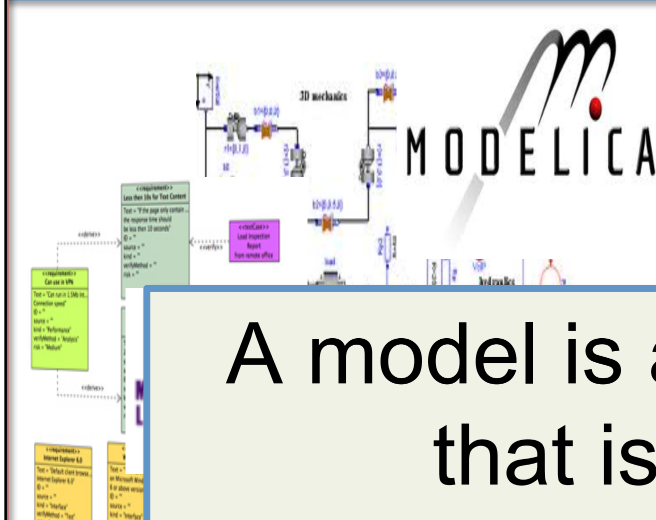
The Kopetz Principle

Hermann Kopetz
Professor (Emeritus)
TU Vienna

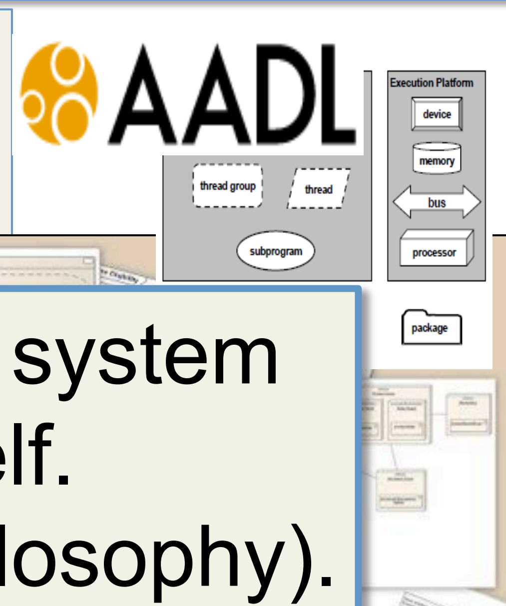
Many properties that we assert about systems (safety, determinism, timeliness, reliability) are in fact not properties of the system, but rather properties of a *model* of the system.

Assurance is about models, not things.

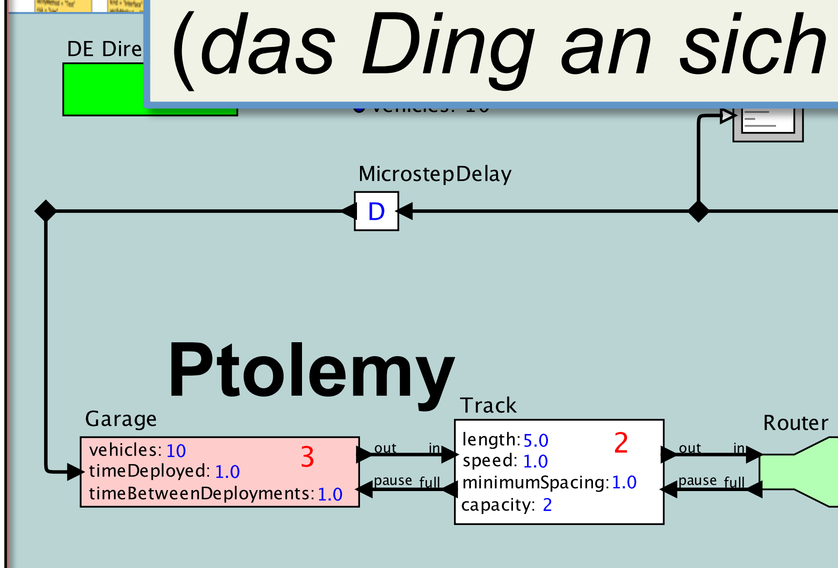
What is a model?

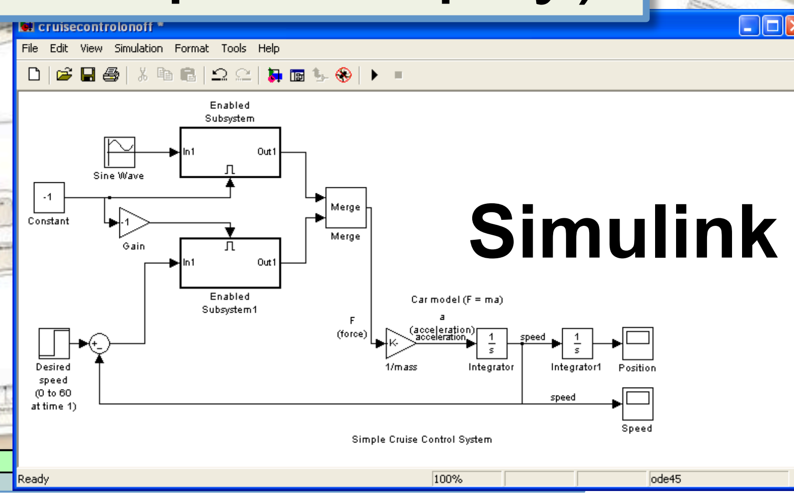



```
// Source code is a model:
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  notify(x[i]);
}
```



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





Quiz: Is this a Model?

```
void foo(int32_t x) {
      if (x > 1000) {
            x = 1000;
      }
      if (x > 0) {
            x = x + 1000;
            if (x < 0) {
                  panic();
            }
      }
}
```

The physical system has many properties not represented in the model (e.g. timing, temperature, physical volume).


Ceci n'est pas une pipe.

Purposes of Models

- Describe structure, weight, dimensions, …
- Describe energy needs, temperatures, …
- Describe dynamics
- Document a design
- Simulate a behavior
- Verify that conformance with a requirement
- Specify a requirement
- …

Properties of Models

Formal?

A *formal model* is a model given in a well-defined, machine-readable syntax and can be mechanistically manipulated using well-defined rules to derive properties of the model.

Executable?

An *executable model* is a formal model describing the dynamic behavior of a system where a machine can use the model to simulate that dynamic behavior.

Faithful?

A *faithful model* is a model that reasonably accurately conforms to properties of the thing being modeled.

Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

Deterministic Models

	Single-threaded imperative programs	Instruction set architecture (ISA)	Synchronous digital logic

Model

Single-threaded imperative programs

```
void foo(int32_t x) {
        if (x > 1000) {
            x = 1000;
        }
        if (x > 0) {
            x = x + 1000;
            if (x < 0) {
                  panic();
            }
        }
}
```

Instruction set architecture (ISA)

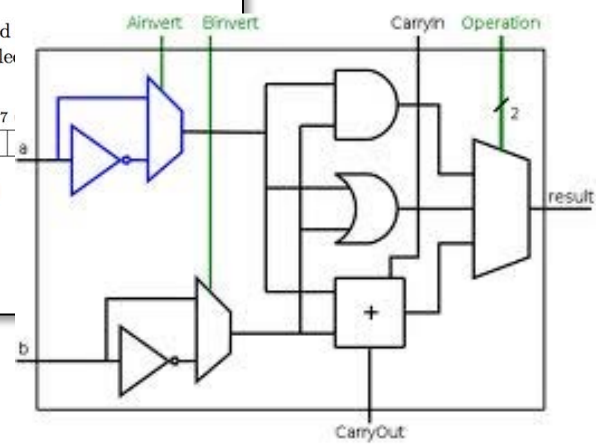
Integer Register-Register Operations

RISC-V defines several arithmetic R-type operations. All operations read as source operands and write the result into register *rd*. The *funct* field sele

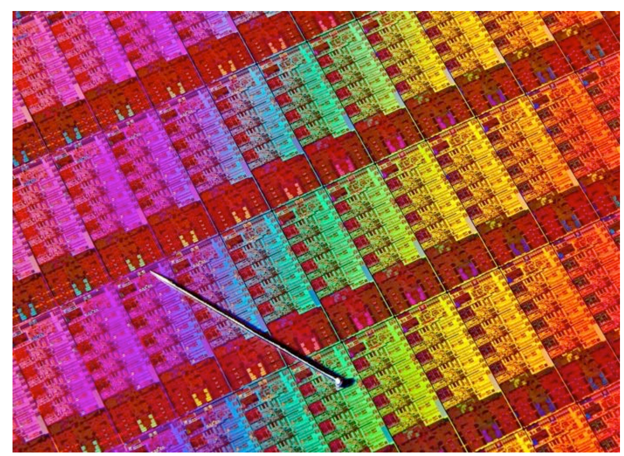
31 27	26 22	21 17	16 7
rd	rs1	rs2	funct10
5	5	5	10
dest	src1	src2	ADD/SUB/SLT/SLTU
dest	src1	src2	AND/OR/XOR
dest	src1	src2	SLL/SRL/SRA
dest	src1	src2	ADDW/SUBW
dest	src1	src2	SLLW/SRLW/SRAW

Waterman, et al., The RISC-V Instruction Set Manual, UCB/EECS-2011-62, 2011

Synchronous digital logic



Physical System



Images: Wikimedia Commons

Determinism as a Property of Models

A **model** is *deterministic* if, given the initial *state* and the *inputs*, the model defines exactly one *behavior*.

Deterministic models have proven valuable

- They enable testing.
  - Known inputs => known outputs
- Analysis is more tractable.
  - Math: Boolean algebra, calculus, etc.
- Simulation is more useful.
  - One input yields one trace.
- Verification scales better.
  - Much smaller state space.
- More certifiable.
  - What is being certified is clearer.

Some nondeterministic designs are untestable

NASA's Toyota Study (US Dept. of Transportation, 2011) found that Toyota software was “untestable.”

Possible victim of unintended acceleration

Determinism in Physics: Laplace's Demon

Pierre Simon Laplace

Pierre-Simon Laplace (1749–1827).
Portrait by Joan-Baptiste Paulin Guérin, 1838

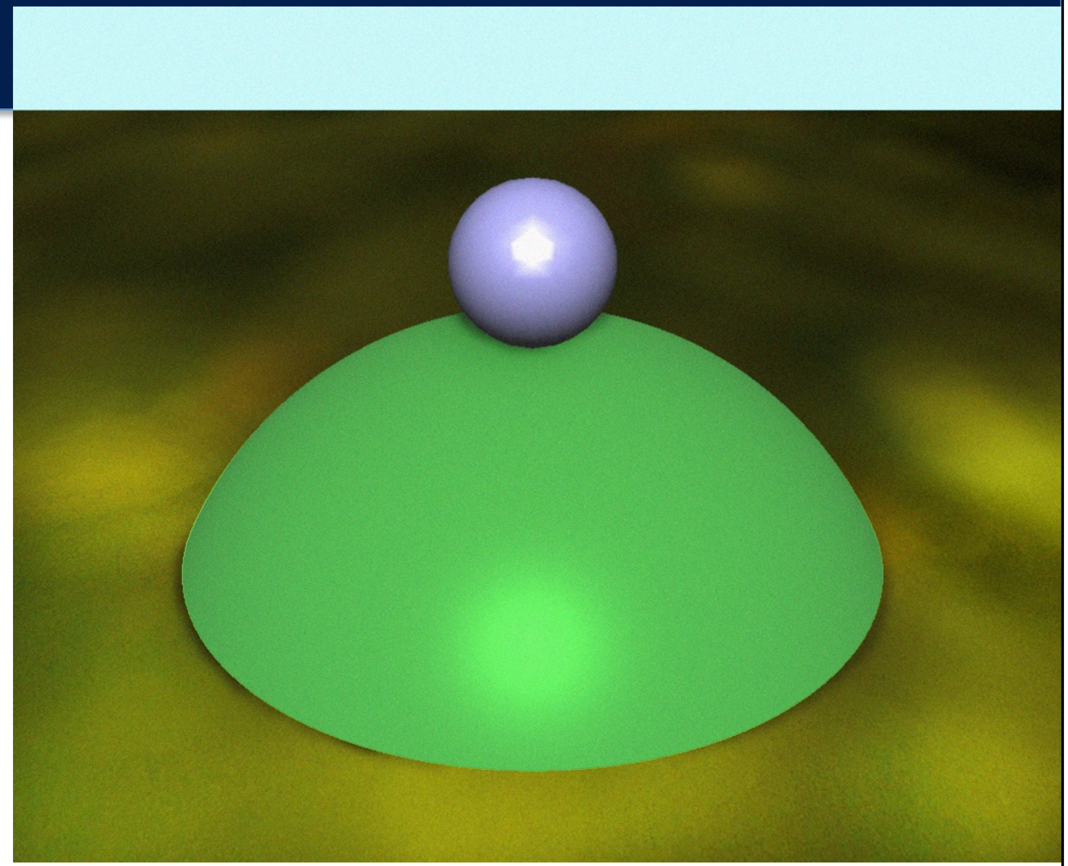
Lee, Berkeley

Did quantum physics dash this hope?

"At first, it seemed that these hopes for a complete determinism would be dashed by the discovery early in the 20th century that events like the decay of radioactive atoms seemed to take place at random. It was as if God was playing dice, in Einstein's phrase. But science snatched victory from the jaws of defeat by moving the goal posts and redefining what is meant by a complete knowledge of the universe."

(Stephen Hawking, 2002)

Norton's Hill



Even without quantum physics, Newtonian physics is not deterministic.

Norton, J. D. (2007). Causation as Folk Science. *In Causation, Physics, and the Constitution of Reality*, Oxford, Clarendon Press.

Metastable system that obeys all of Newton's laws but is nondeterministic.

Laplace's Demon cannot exist.

In 2008, David Wolpert proved that Laplace's demon cannot exist.

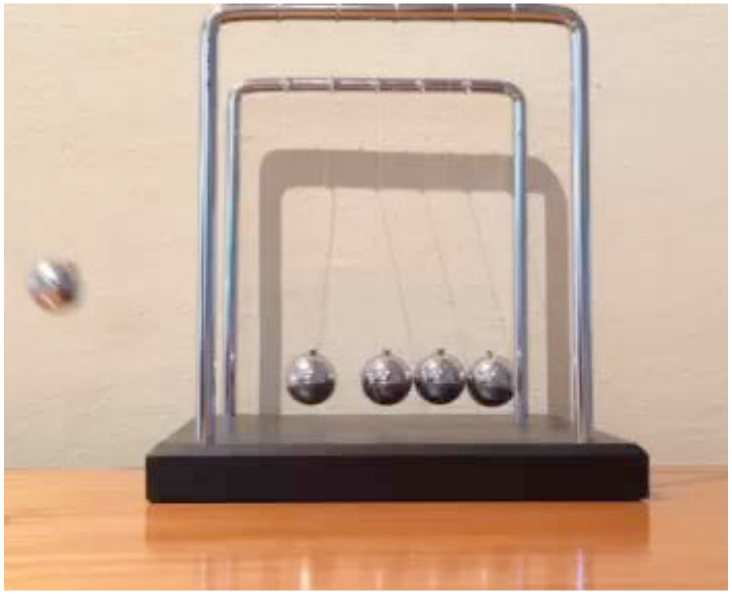
His proof relies on the observation that such a demon, were it to exist, would have to exist in the very physical world that it predicts.

David Wolpert

Determinism is a property of models, not things

$$
\begin{aligned}
x(t) &= x(0) + \int_0^t v(\tau)d\tau \\
v(t) &= v(0) + \frac{1}{m}\int_0^t F(\tau)d\tau,
\end{aligned}
$$

Deterministic model



Deterministic system?

Determinism as a Property of Models

A **model** is *deterministic* if, given the initial *state* and the *inputs*, the model defines exactly one *behavior*.

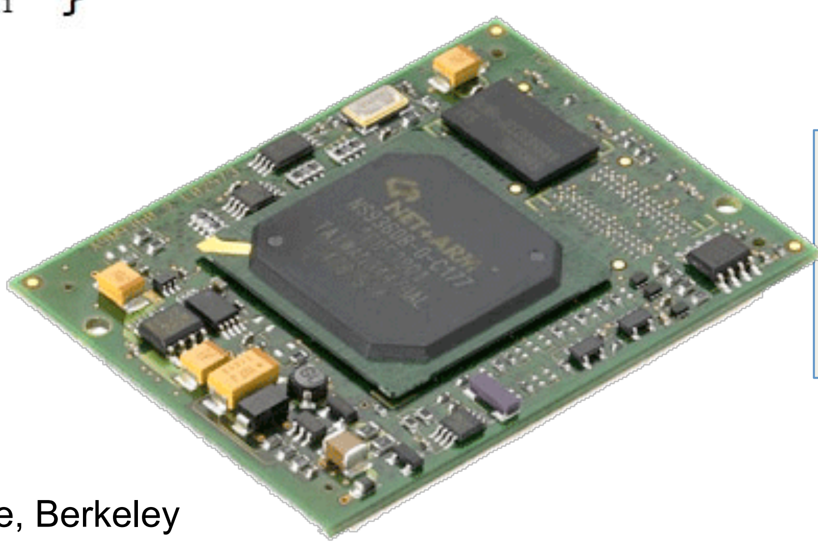
Our most valuable models are *deterministic*.

Software as a Model

```c
void foo(int32_t x) {
        if (x > 1000) {
                x = 1000;
        }
        if (x > 0) {
                x = x + 1000;
                if (x < 0) {
                        panic();
                }
        }
}
```

This program defines exactly one behavior, given the input x.

The modeling framework defines state, input, and behavior.



The physical system has many properties not represented in the model (e.g. timing, temperature, …).

Architecture as a Model

Physical System

Model

Image: Wikimedia Commons

Integer Register-Register Operations

RISC-V defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct* field selects the type of operation.

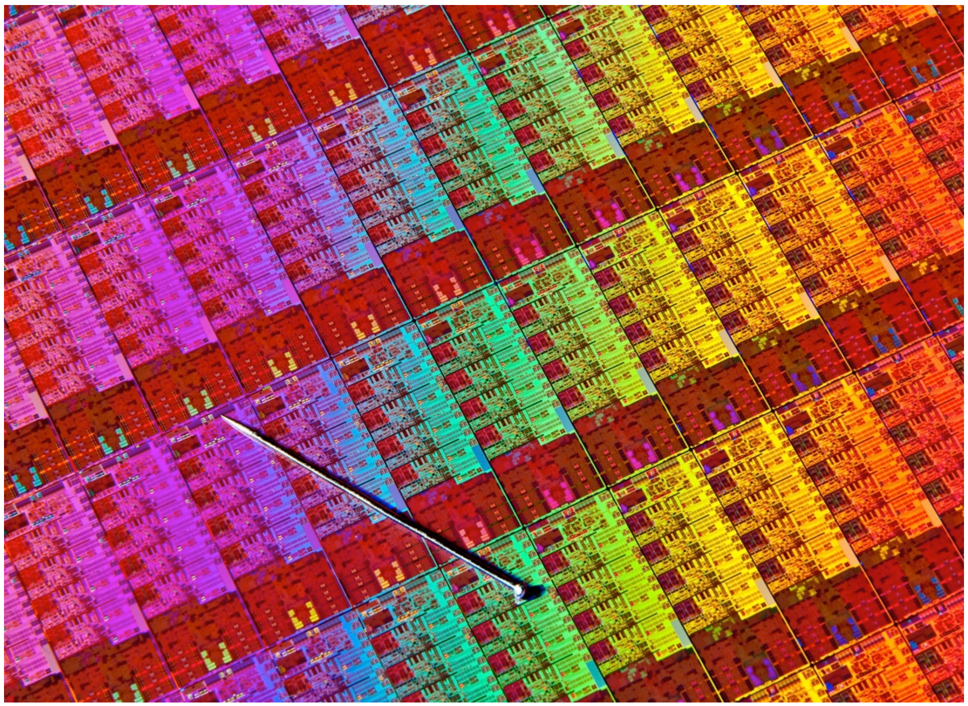
31 27	26 22	21 17	16 7	6 0
rd	rs1	rs2	funct10	opcode
5	5	5	10	7
dest	src1	src2	ADD/SUB/SLT/SLTU	OP
dest	src1	src2	AND/OR/XOR	OP
dest	src1	src2	SLL/SRL/SRA	OP
dest	src1	src2	ADDW/SUBW	OP-32
dest	src1	src2	SLLW/SRLW/SRAW	OP-32

Waterman, et al., The RISC-V Instruction Set Manual, UCB/EECS-2011-62, 2011

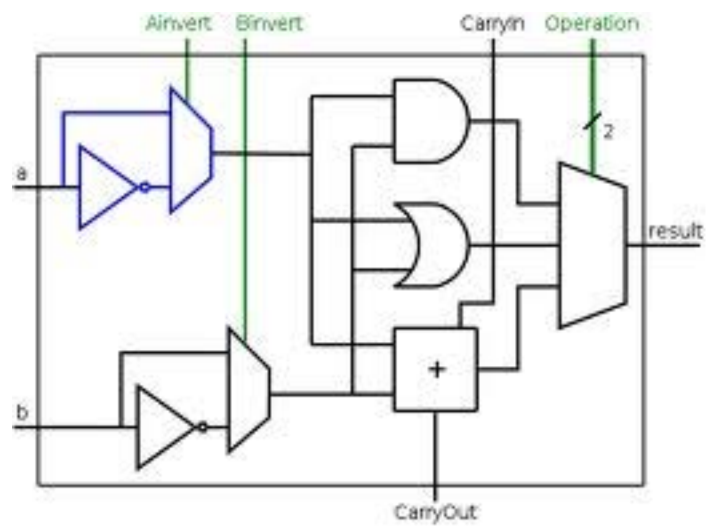
Instruction Set Architectures (ISAs) are deterministic models

Digital Circuits as Models

Physical System



Model



Synchronous digital logic
is a deterministic model

Physical Dynamics as a Model

Physical System

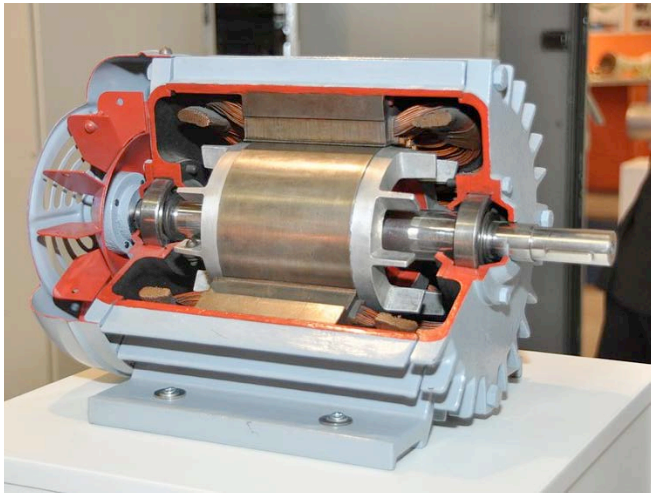


Image: Wikimedia Commons

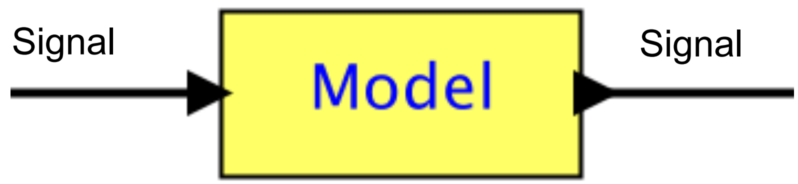
Model

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

Differential Equations
are deterministic models

CPS combinations of deterministic models are nondeterministic

```
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
}
```



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

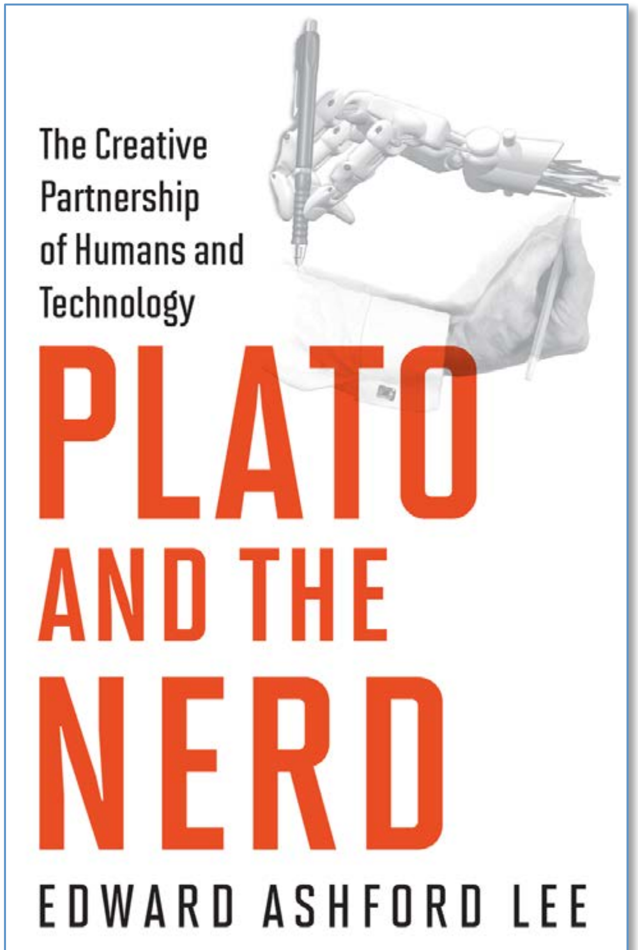
Image: Wikimedia Commons

Determinism?
What about resilience? Adaptability?

Deterministic models do not eliminate the need for robust, fault-tolerant designs.

In fact, they *enable* such designs, because they make it much clearer what it means to have a fault!

An Epiphany


The Creative
Partnership
of Humans and
Technology
PLATO
AND THE
NERD
EDWARD ASHFORD LEE


PLATO
AND THE
NERD
EDWARD ASHFORD LEE

Two Usage Patterns for Models

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

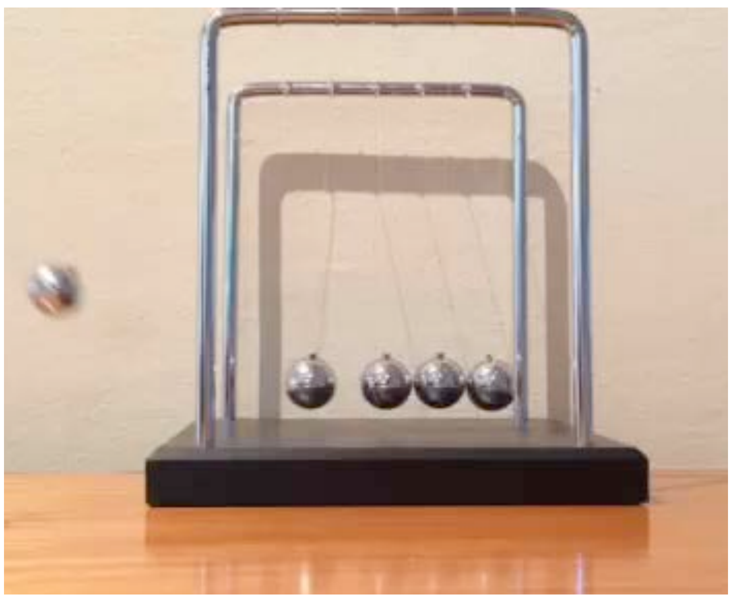
- In *engineering*, the value of a *physical system* lies in how well its behavior matches that of the model.

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

Models vs. Reality

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

The model



The target (the thing being modeled).

In this example, the *modeling framework* is calculus and Newton's laws.

Fidelity is how well the model and its target match

A Model

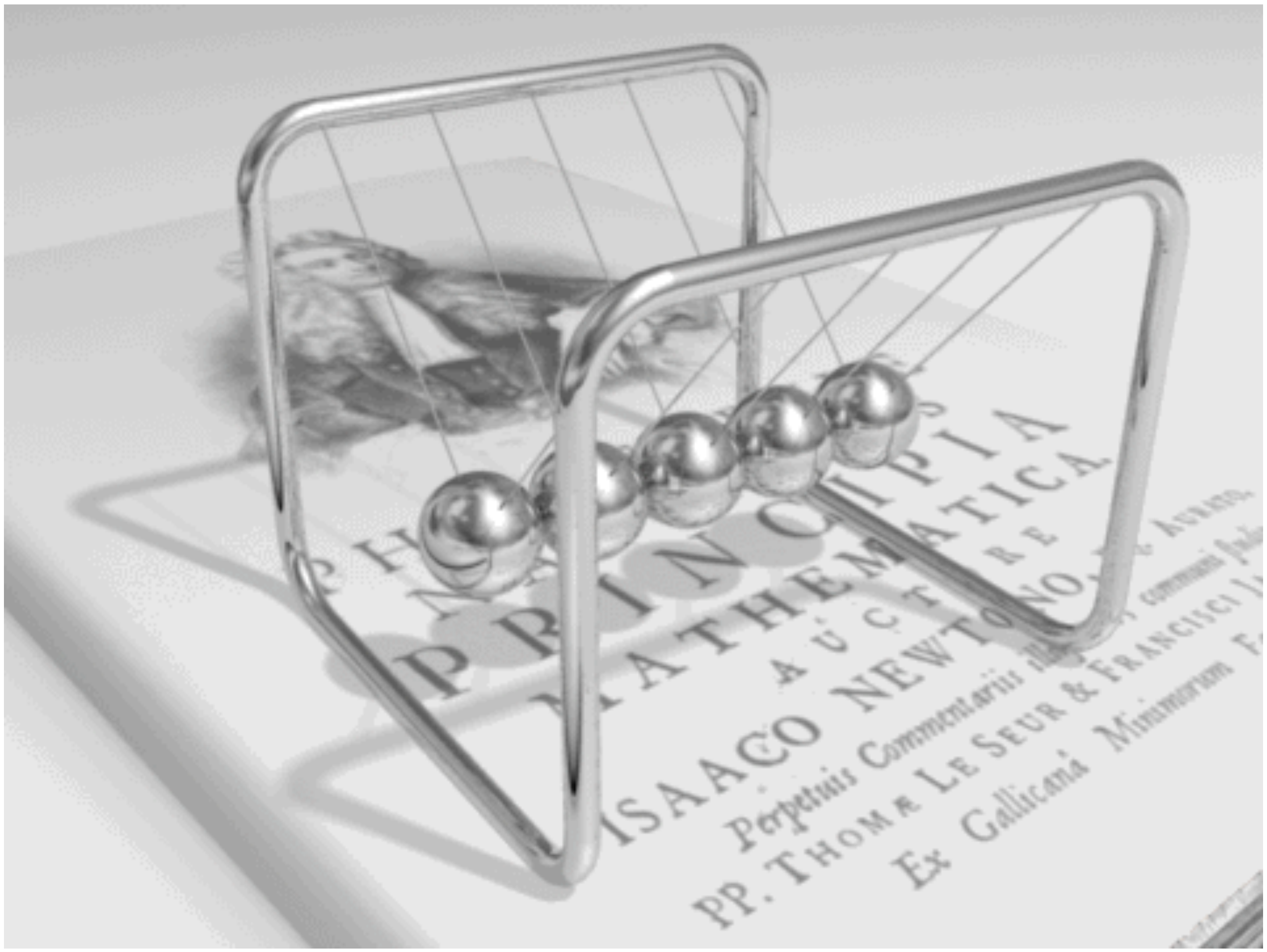
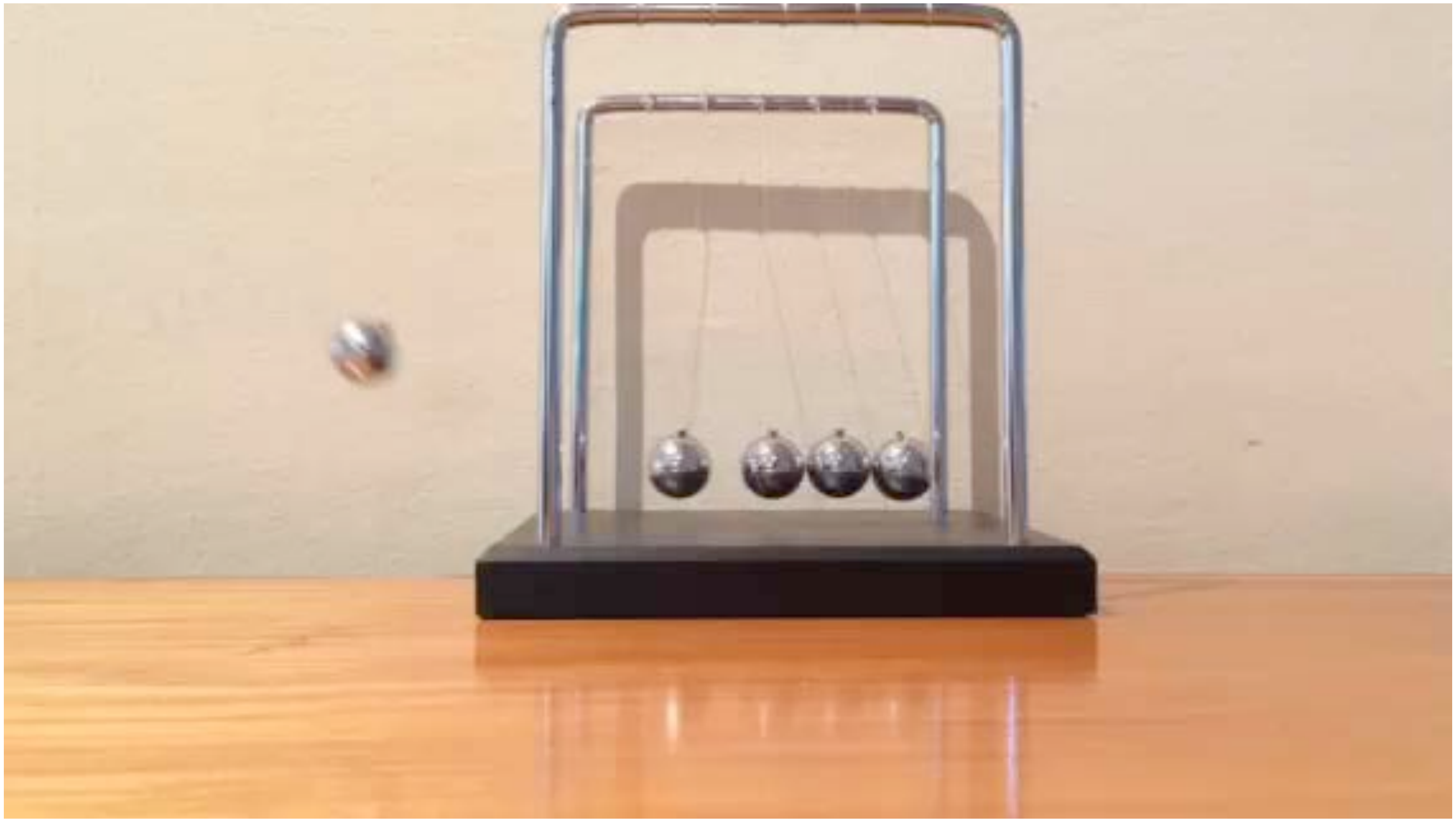


Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

A Physical Realization



Model Fidelity

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

To a realist, both are flawed…

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.

The Value of Simulation

"Simulation is doomed to succeed."

Could this statement be confusing engineering and scientific models?

Figure 1: Three scenes generated from a single ~20-line SCENIC scenario representing bumper-to-bumper traffic.

[Fremont, et al., Scenic: Language-Based Scene Generation, Arxiv.org, Sept. 2018]

Engineers often confuse the model with its target

You will never strike oil by drilling through the map!

Solomon Wolf Golomb

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

Model-Based Design of Cyber-Physical Systems

Changing the Question:

Is the question whether we can build models describing the behavior of cyber-physical systems?

Or

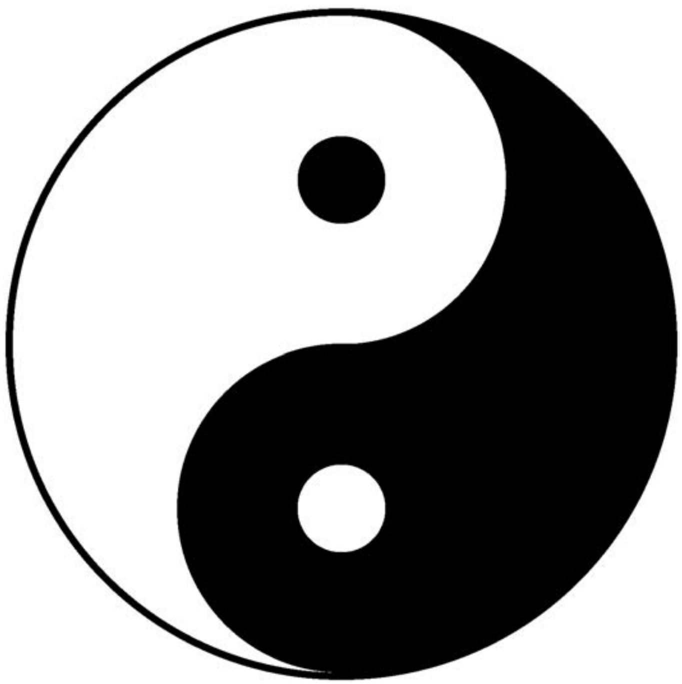
Is the question whether we can make cyber-physical systems that behave like our models?

Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

But…

Determinism has its limits.



- Complexity
- Uncertainty
- Chaos
- Incompleteness

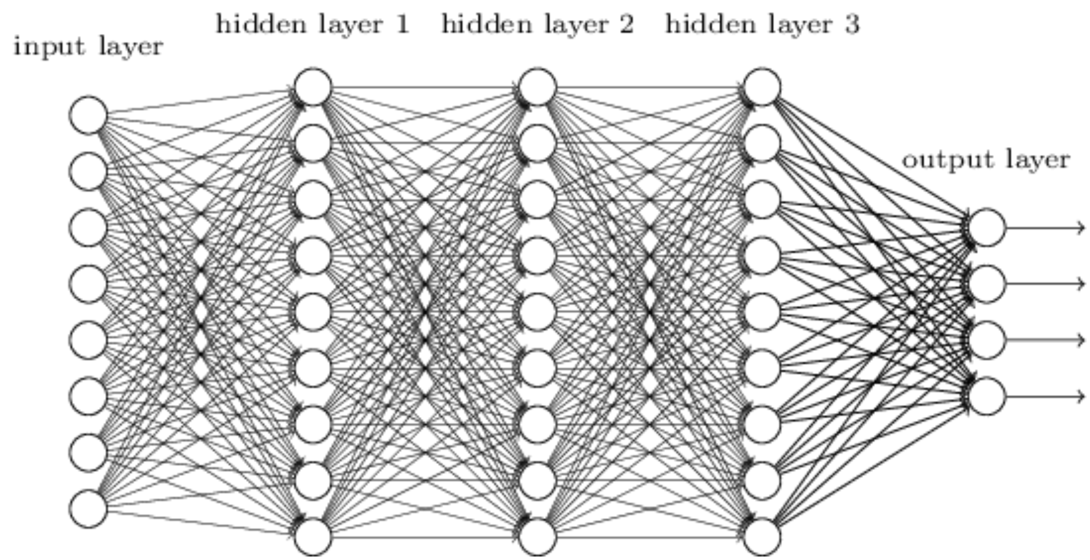
Complexity

- Some systems are too complex for deterministic models.
- Nondeterministic abstractions become useful.

“Iron wing” model of an Airbus A350.

Complexity

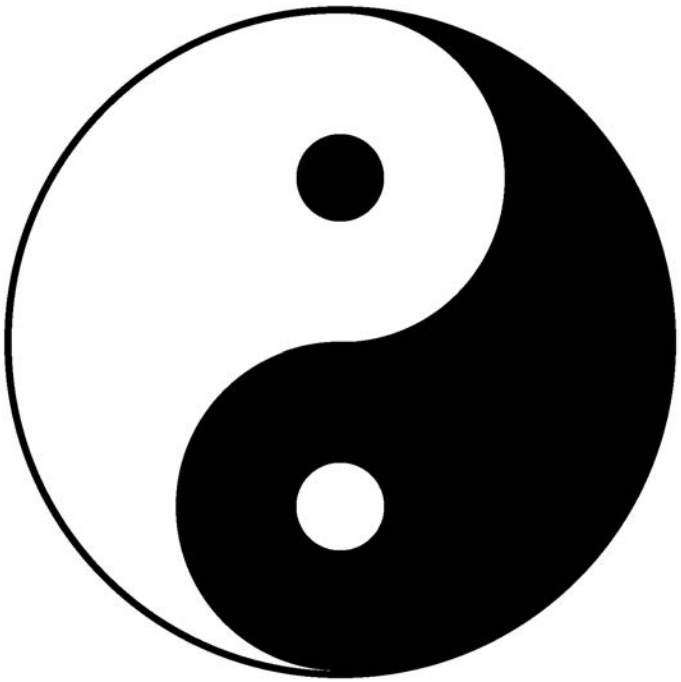
- Some systems are too complex for deterministic models.
- Nondeterministic abstractions become useful.



Deep Learning, draft book in preparation, by Yoshua Bengio, Ian Goodfellow, and Aaron Courville. http://www.deeplearningbook.org/

But…

Determinism has its limits.



- Complexity
- Uncertainty
- Chaos
- Incompleteness

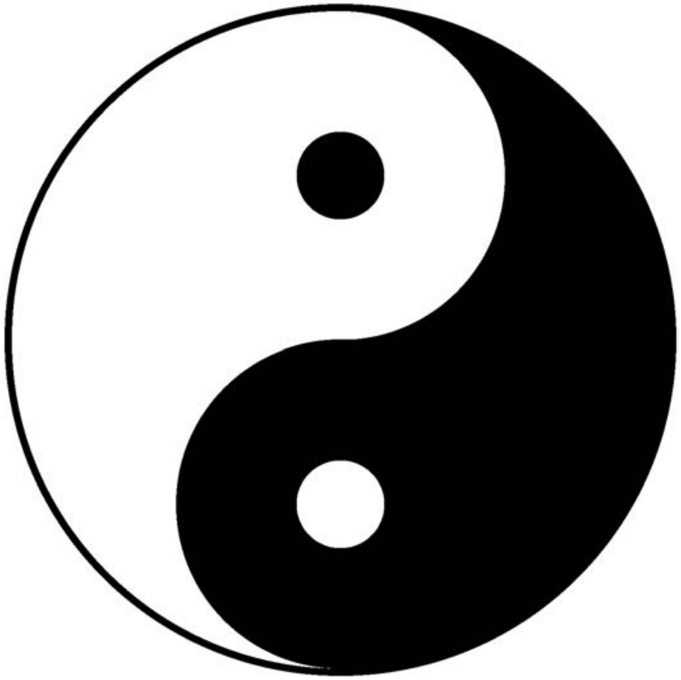
Uncertainty

- We can't construct deterministic models of what we don't know.
- For this, nondeterminism is useful.
- Bayesian probability (which is mostly due to Laplace) quantifies uncertainty.

Portrait of Reverend Thomas Bayes (1701 - 1761) that is probably not actually him.

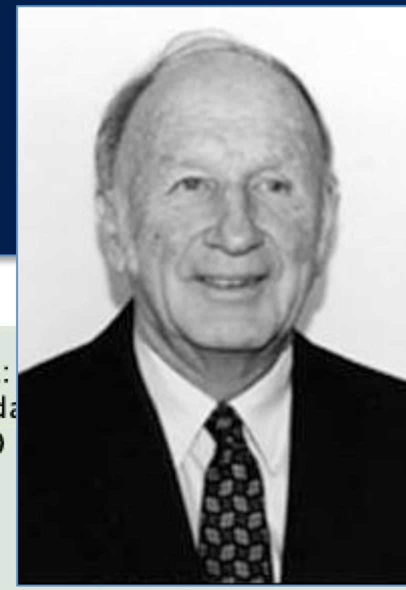
But…

Determinism has its limits.



- Complexity
- Uncertainty
- Chaos
- Incompleteness

Determinism does not imply predictability

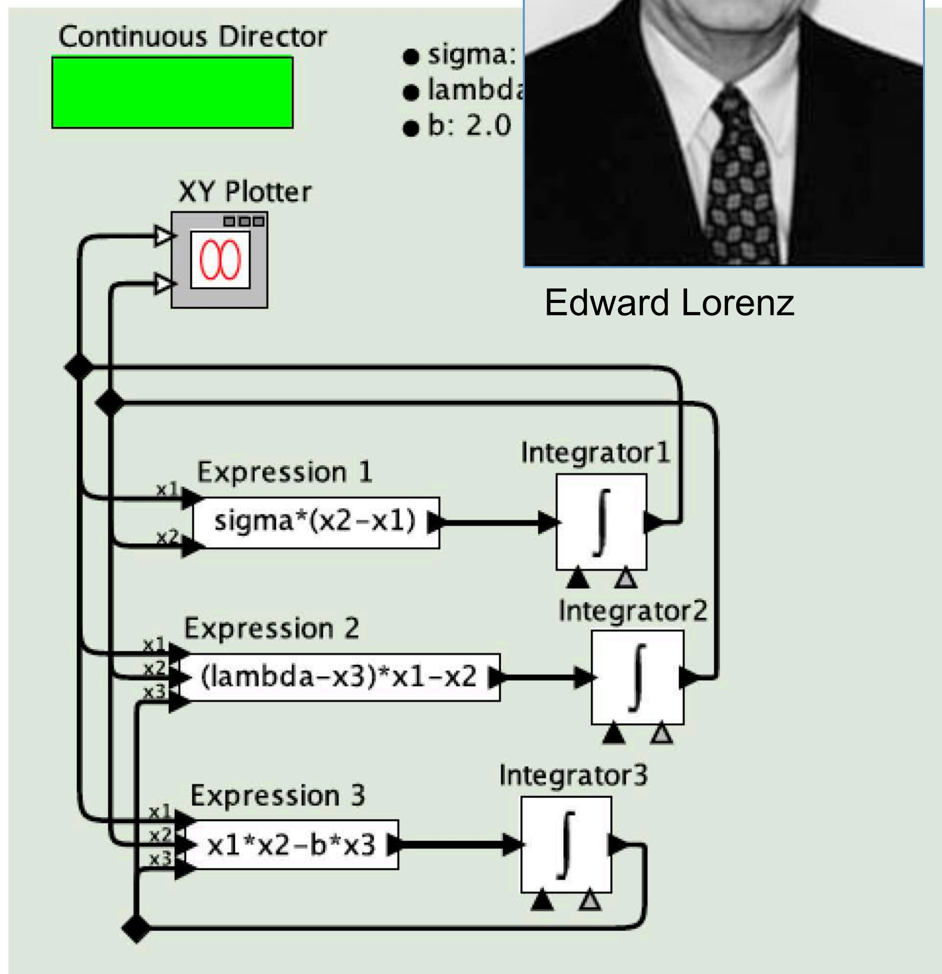


Edward Lorenz

Lorenz attractor:

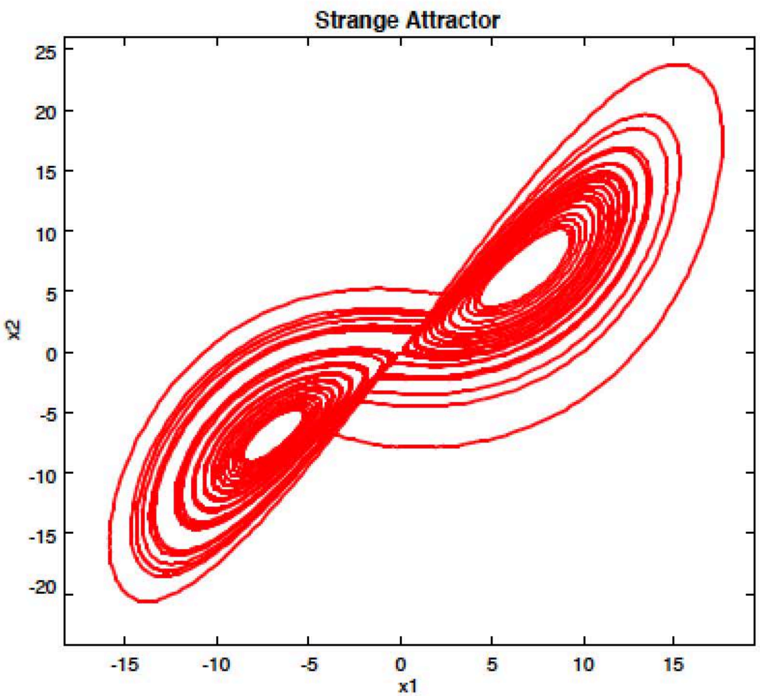
$$
\begin{aligned}
\dot{x}_1(t) &= \sigma(x_2(t) - x_1(t)) \\
\dot{x}_2(t) &= (\lambda - x_3(t))x_1(t) - x_2(t) \\
\dot{x}_3(t) &= x_1(t)x_2(t) - bx_3(t)
\end{aligned}
$$

This is a chaotic system, so arbitrarily small perturbations have arbitrarily large consequences.

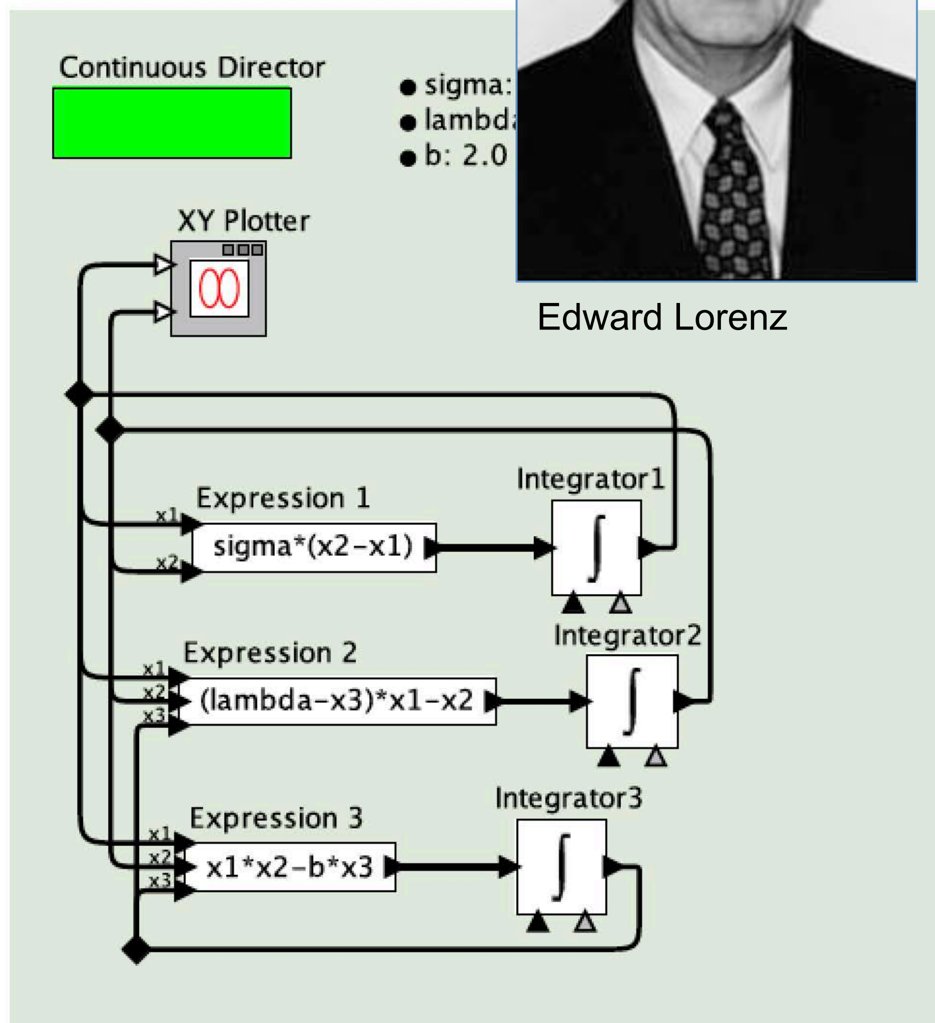


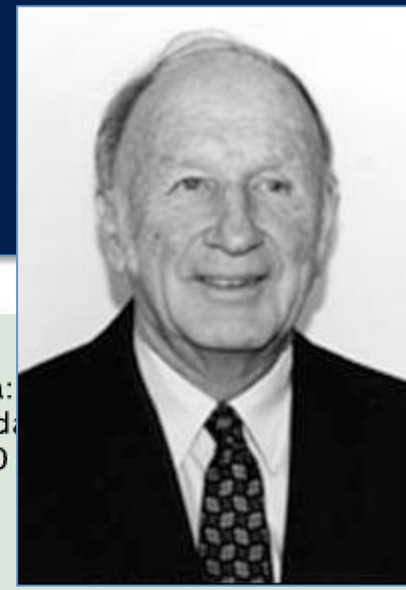
Determinism does not imply predictability

Plot of x_1 vs. x_2:



The position of a point is not meaningfully predictable even though the system is deterministic.





Edward Lorenz

Determinism does not imply predictability

Deterministic real-time scheduling results in chaos.

[Thiele and Kumar, EMSOFT 2015]

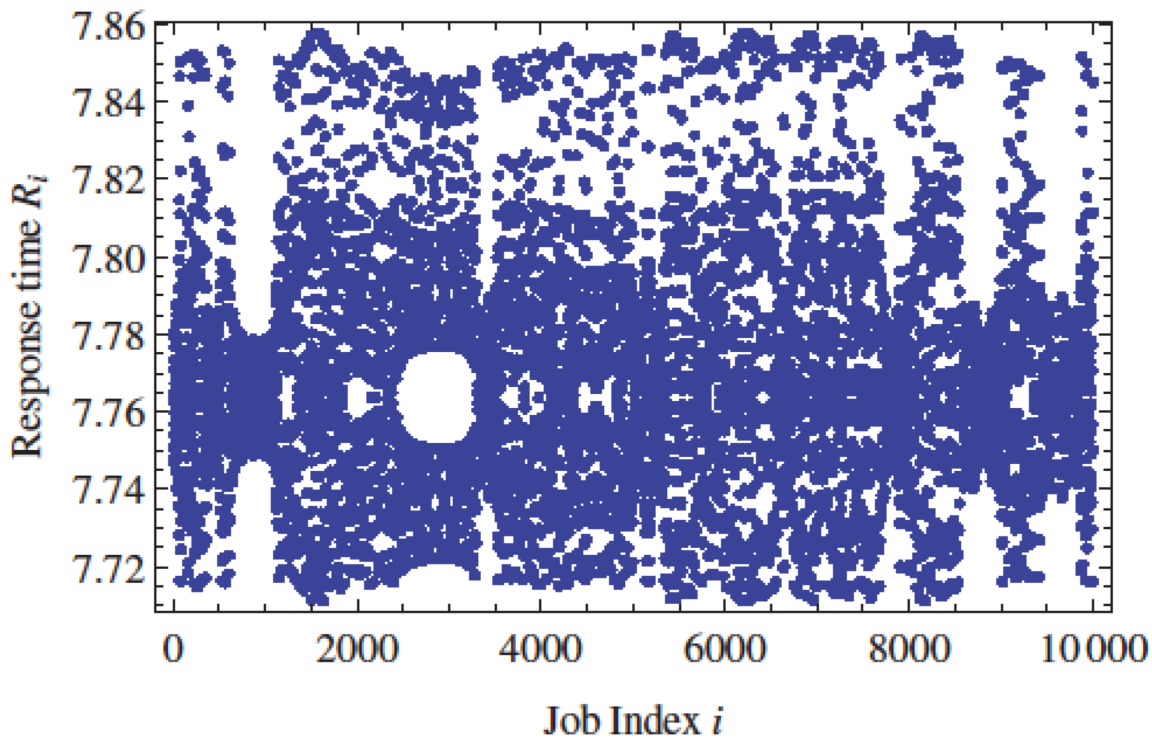
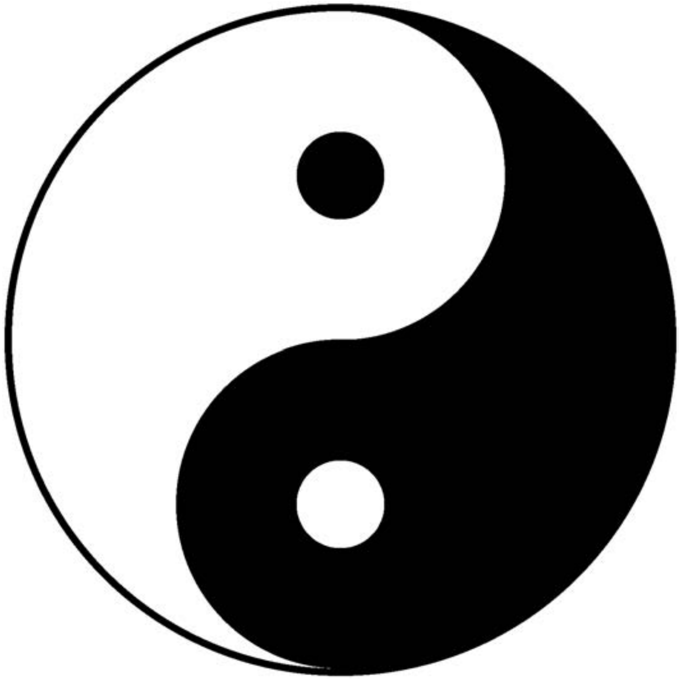


Fig. 15. Response time across jobs for the multi-resource scheduler with $R_s(i-1) = 7.76$ and $R_s(i-2) = 7.74$.

But…

Determinism has its limits.

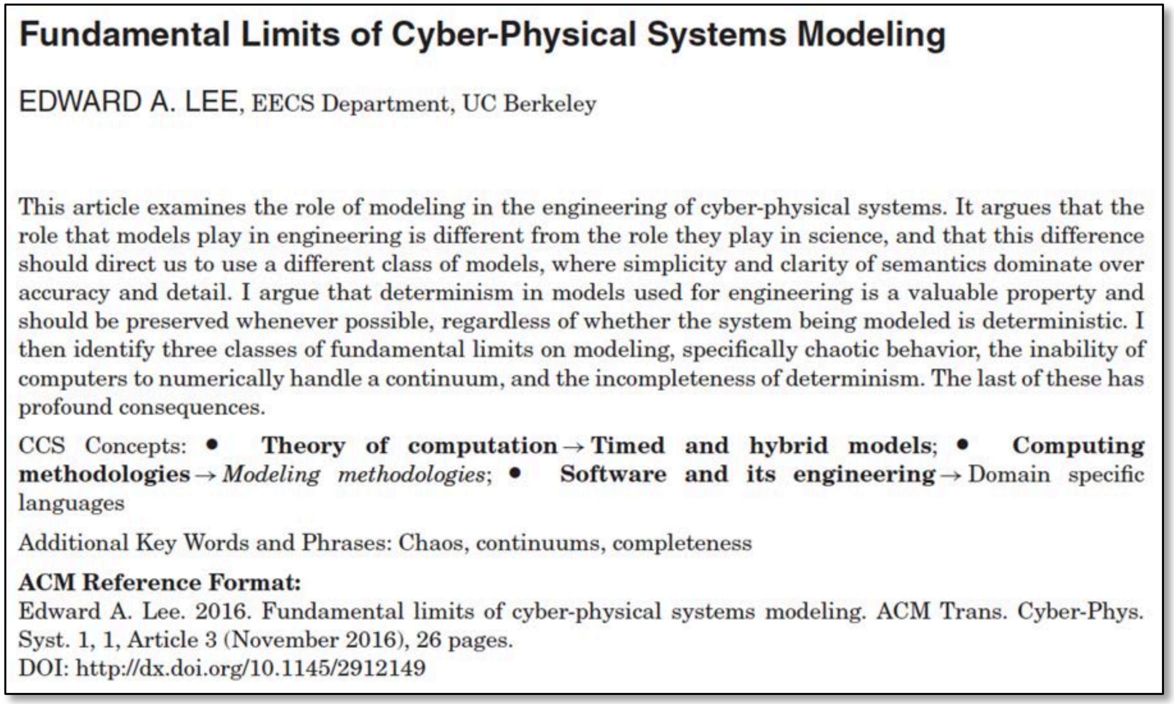


- Complexity
- Uncertainty
- Chaos
- Incompleteness

Incompleteness of Determinism

Any set of deterministic models rich enough to encompass Newton's laws plus discrete transitions is incomplete.

Lee, *Fundamental Limits of Cyber-Physical Systems Modeling*, ACM Tr. on CPS, Vol. 1, No. 1, November 2016



Fundamental Limits of Cyber-Physical Systems Modeling

EDWARD A. LEE, EECS Department, UC Berkeley

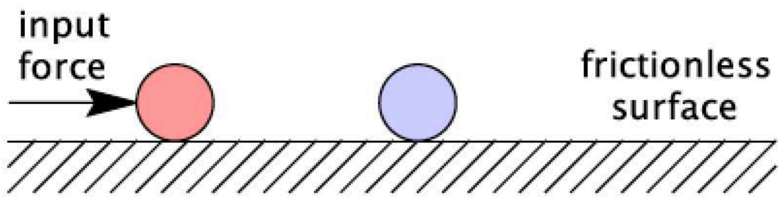
This article examines the role of modeling in the engineering of cyber-physical systems. It argues that the role that models play in engineering is different from the role they play in science, and that this difference should direct us to use a different class of models, where simplicity and clarity of semantics dominate over accuracy and detail. I argue that determinism in models used for engineering is a valuable property and should be preserved whenever possible, regardless of whether the system being modeled is deterministic. I then identify three classes of fundamental limits on modeling, specifically chaotic behavior, the inability of computers to numerically handle a continuum, and the incompleteness of determinism. The last of these has profound consequences.

CCS Concepts: • **Theory of computation** → **Timed and hybrid models**; • **Computing methodologies** → *Modeling methodologies*; • **Software and its engineering** → Domain specific languages

Additional Key Words and Phrases: Chaos, continuums, completeness

ACM Reference Format:
Edward A. Lee. 2016. Fundamental limits of cyber-physical systems modeling. ACM Trans. Cyber-Phys. Syst. 1, 1, Article 3 (November 2016), 26 pages.
DOI: http://dx.doi.org/10.1145/2912149

Illustration of the Incompleteness of Determinism



Conservation of momentum:

$$m_1 v_1' + m_2 v_2' = m_1 v_1 + m_2 v_2 .$$

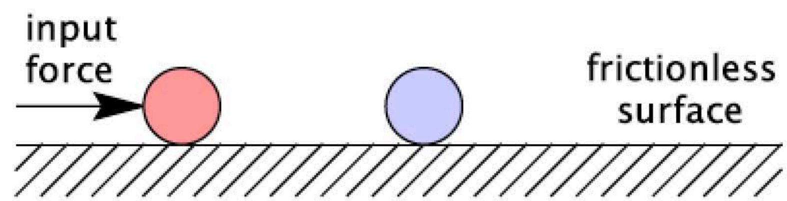
Conservation of kinetic energy:

$$\frac{m_1 (v_1')^2}{2} + \frac{m_2 (v_2')^2}{2} = \frac{m_1 (v_1)^2}{2} + \frac{m_2 (v_2)^2}{2} .$$

We have two equations and two unknowns, v_1' and v_2'.

Illustration of the Incompleteness of Determinism

Quadratic problem has two solutions.



Solution 1: $v_1' = v_1,\ v_2' = v_2$ (ignore collision).

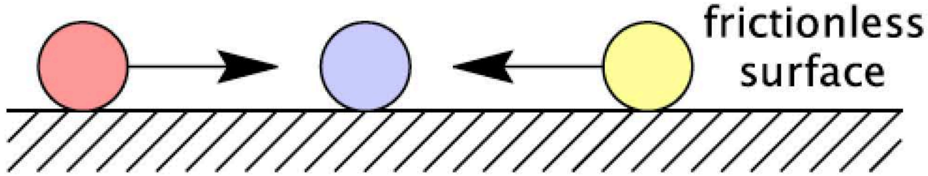
Solution 2:

$$\begin{aligned} v_1' &= \frac{v_1(m_1 - m_2) + 2m_2v_2}{m_1 + m_2} \\ v_2' &= \frac{v_2(m_2 - m_1) + 2m_1v_1}{m_1 + m_2}. \end{aligned}$$

Note that if $m_1 = m_2$, then the two masses simply exchange velocities (Newton's cradle).

Illustration of the Incompleteness of Determinism

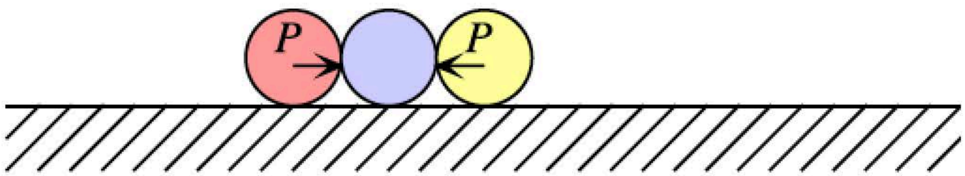
Consider this scenario:



Simultaneous collisions where one collision does not cause the other.

Illustration of the Incompleteness of Determinism

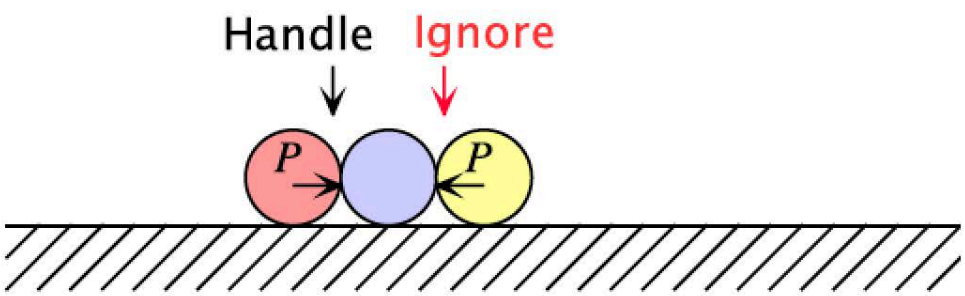
One solution: nondeterministic interleaving of the collisions:



At superdense time $(\tau, 0)$, we have two simultaneous collisions.

Illustration of the Incompleteness of Determinism

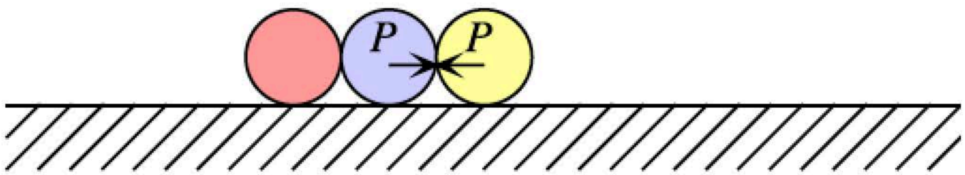
One solution: nondeterministic interleaving of the collisions:



At superdense time $(\tau, 1)$, choose arbitrarily to handle the left collision.

Illustration of the Incompleteness of Determinism

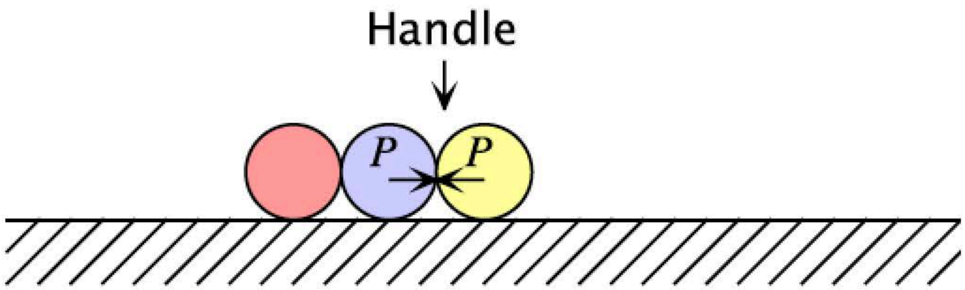
One solution: nondeterministic interleaving of the collisions:



After superdense time $(\tau, 1)$, the momentums are as shown.

Illustration of the Incompleteness of Determinism

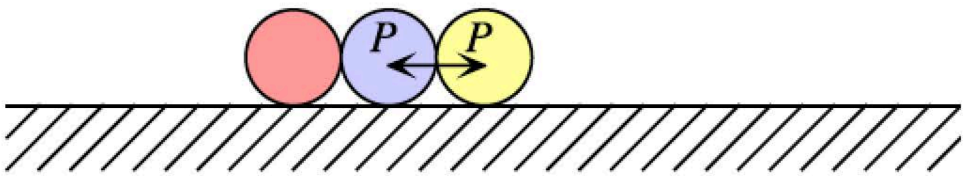
One solution: nondeterministic interleaving of the collisions:



At superdense time $(\tau, 2)$, handle the new collision.

Illustration of the Incompleteness of Determinism

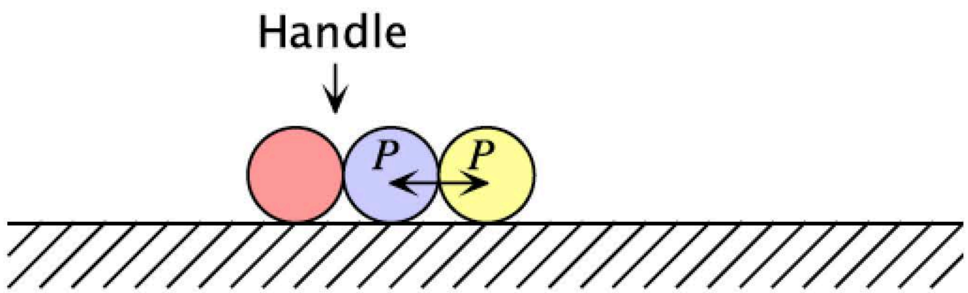
One solution: nondeterministic interleaving of the collisions:



After superdense time $(\tau, 2)$, the momentums are as shown.

Illustration of the Incompleteness of Determinism

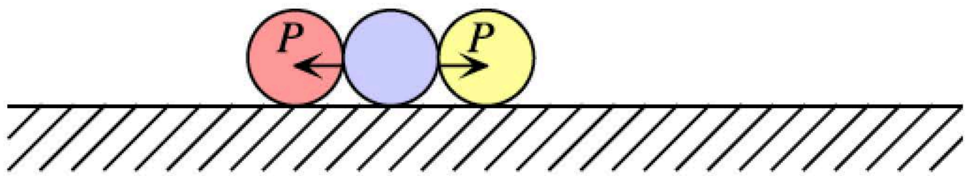
One solution: nondeterministic interleaving of the collisions:



At superdense time $(\tau, 3)$, handle the new collision.

Illustration of the Incompleteness of Determinism

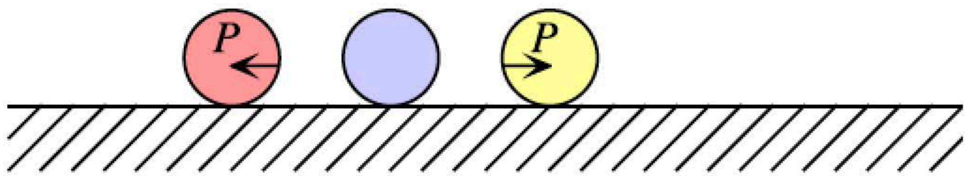
One solution: nondeterministic interleaving of the collisions:



After superdense time $(\tau, 3)$, the momentums are as shown.

Illustration of the Incompleteness of Determinism

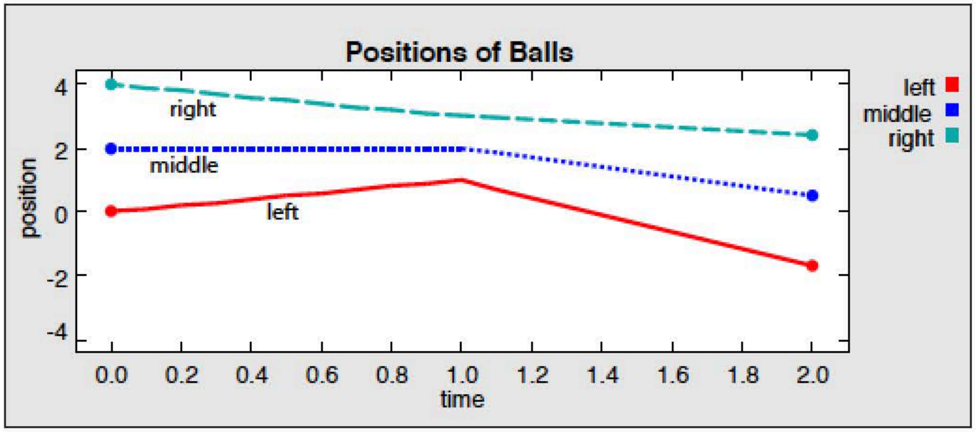
One solution: nondeterministic interleaving of the collisions:



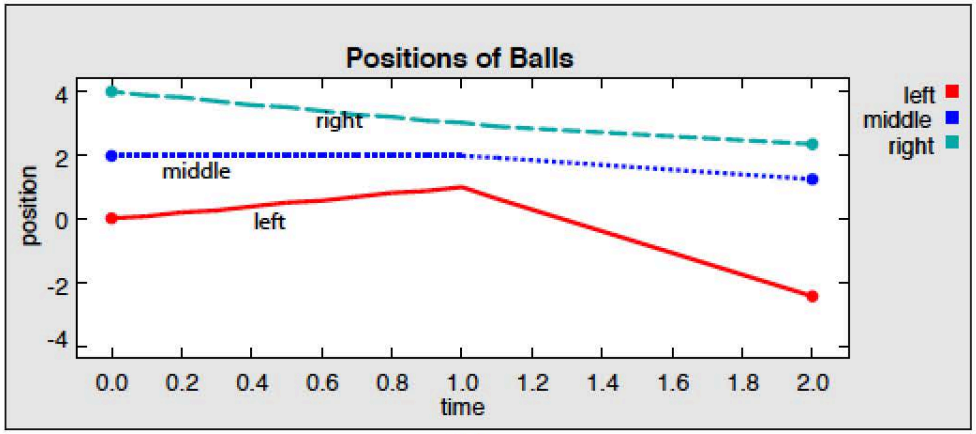
The balls move away at equal speed (if their masses are the same!)

Arbitrary Interleaving Yields Nondeterminism

If the masses are different, the behavior depends on which collision is handled first!



(a)

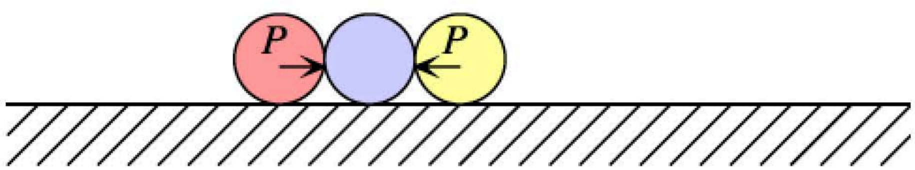


(b)

Recall the Heisenberg Uncertainty Principle

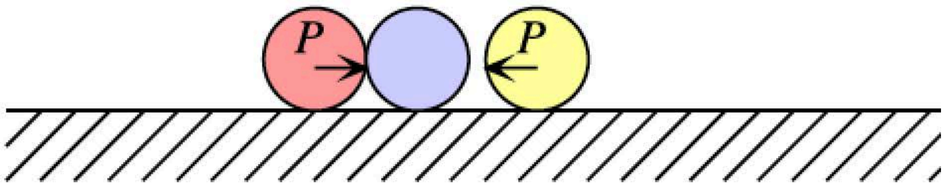
We cannot simultaneously know the position and momentum of an object to arbitrary precision.

But the reaction to these collisions depends on knowing position and momentum precisely.



Is Determinism Incomplete?

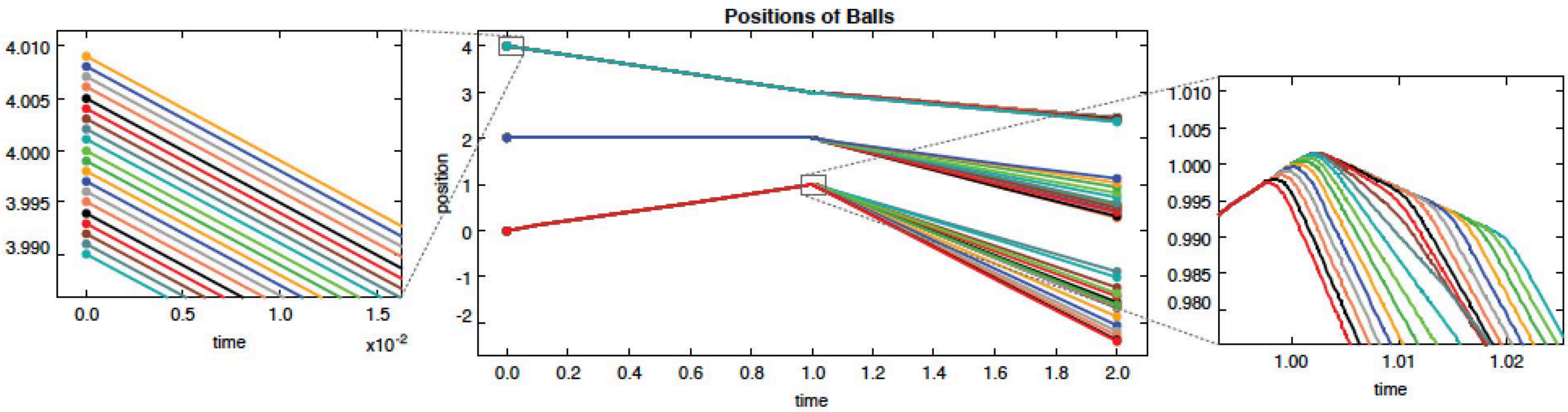
Let τ be the time between collisions. Consider a sequence of models for $\tau > 0$ where $\tau \to 0$.



Every model in the sequence is deterministic, but the limit model is not.

- In Lee (2016), I show that this sequence of models is Cauchy, so the space of deterministic models is incomplete (it does not contain its own limit points).
- In Lee (2014), I show that a direct description of this scenario results in a non-constructive model. The nondeterminism arises in making this model constructive.

Rejecting discreteness leads to deterministic chaos

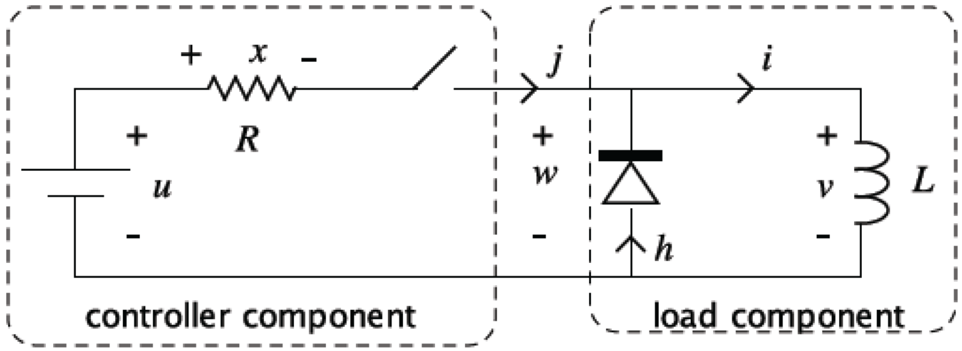


[Lee, ACM Tr. on CPS, 2016]

A continuous deterministic model that models collisions as elastic springs is chaotic.

Rejecting discreteness requires rejecting causality

Example from Lee, "Constructive Models of Discrete and Continuous Physical Phenomena," IEEE Access, 2014



A flyback diode is a commonly used circuit that prevents arcing when disconnecting an inductive load (like a motor) from a power source.

When the switch goes from closed to open, the causality and direct feedthrough properties of the two components reverse.

There is no logic that can transition from A causes B to B causes A smoothly without passing through non-constructive models.

Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

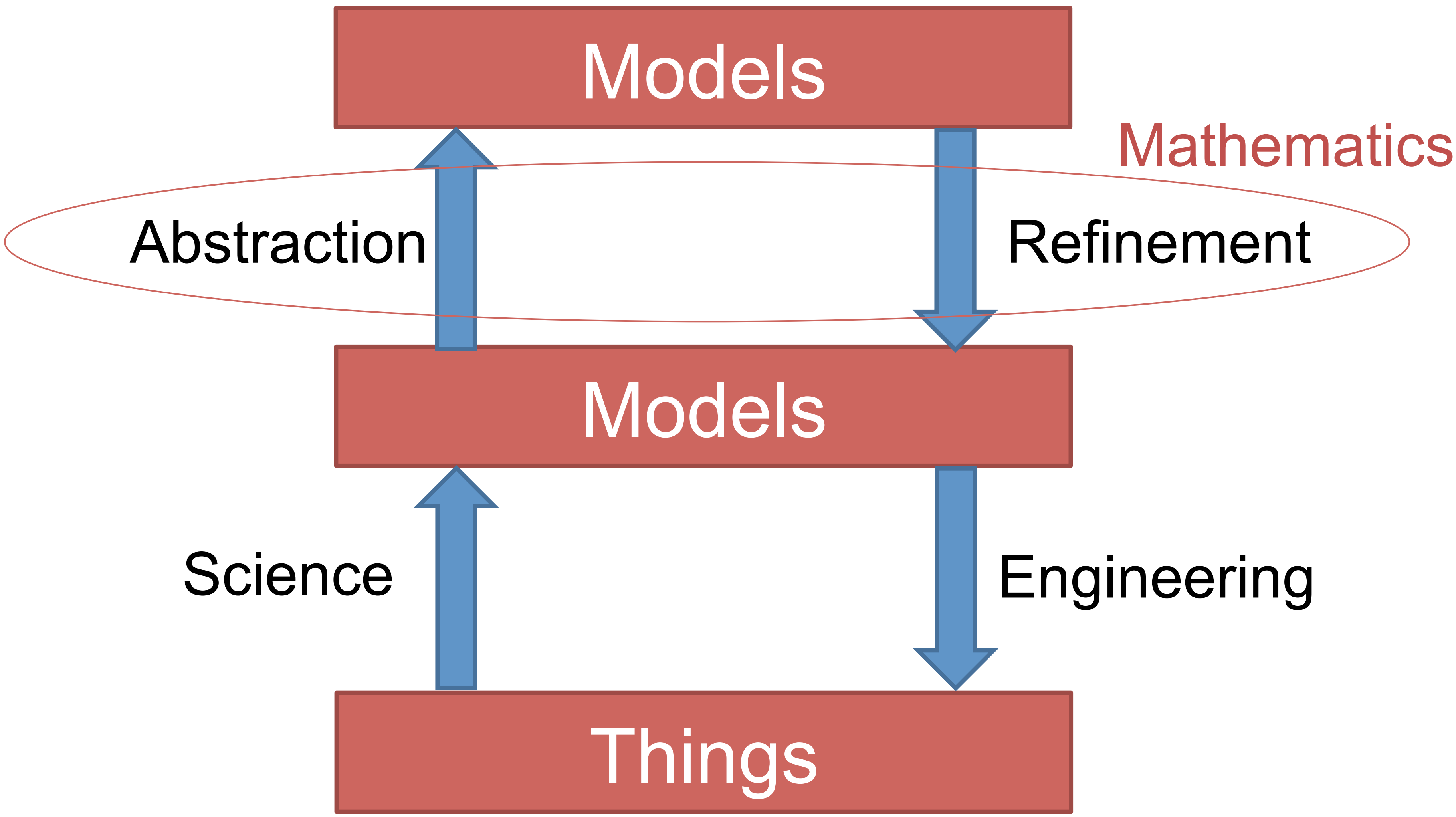
Abstraction and Refinement

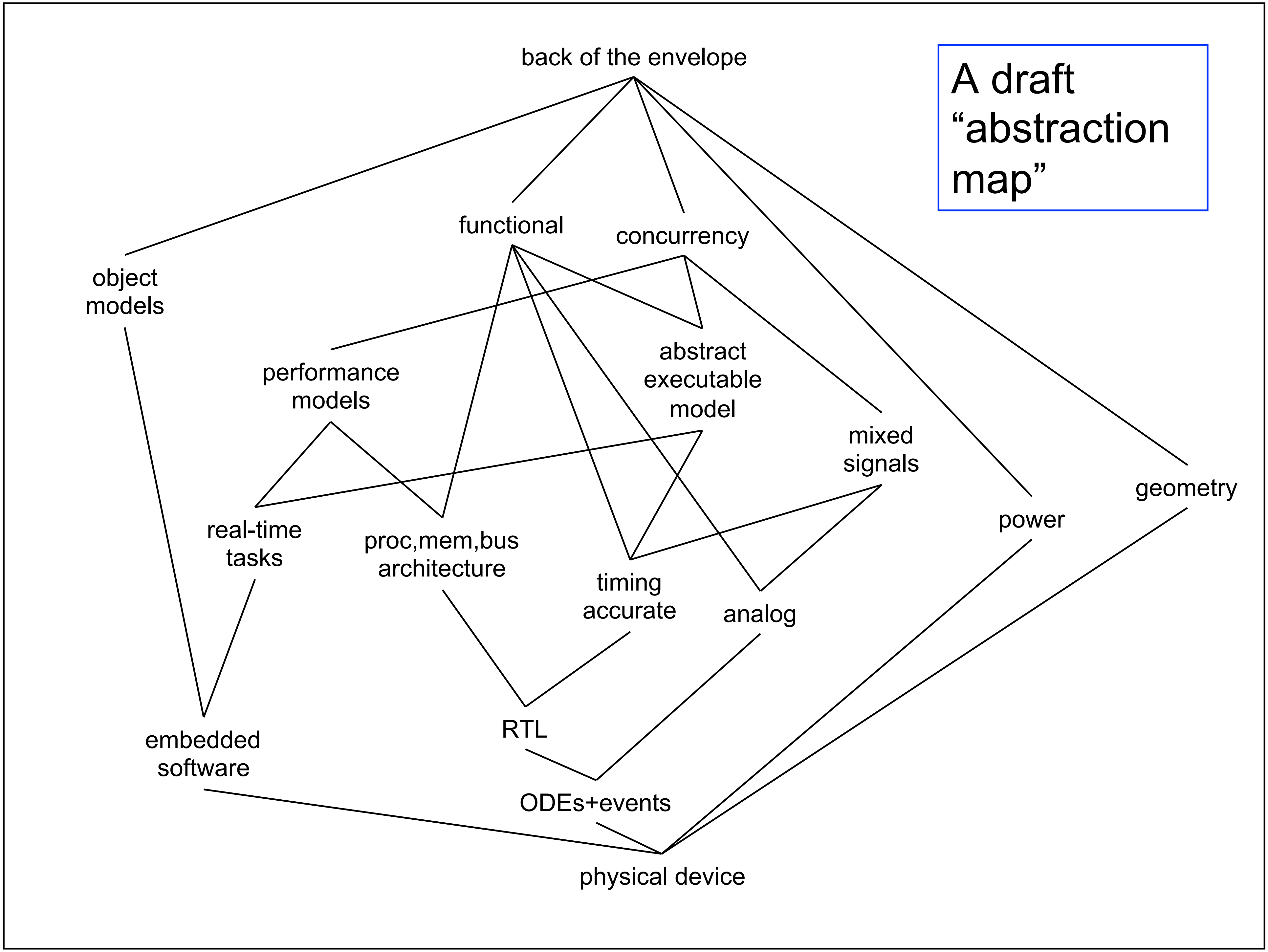
- An abstraction **A** of **B** is sound if every *property of interest* that is true for **A** is also true for **B**

- If **A** is a sound abstraction of **B**, then **B** is a refinement of **A**

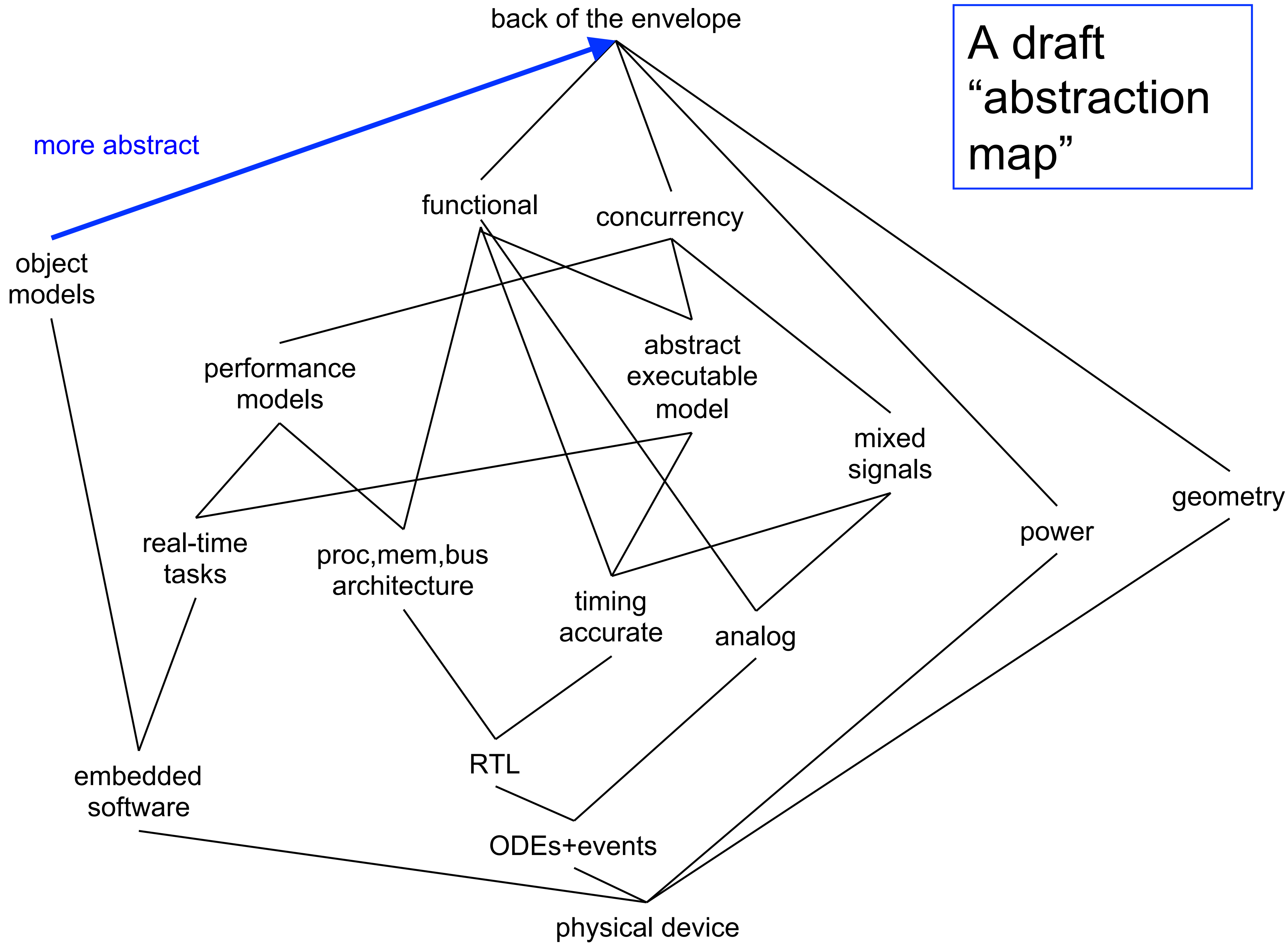
A simulation (of model **A**) is “doomed to succeed” in that it will not reveal properties of a refinement (**B**) nor of the thing-in-itself that are not also properties of **A**.

Models and Things (vs. Models and Models)

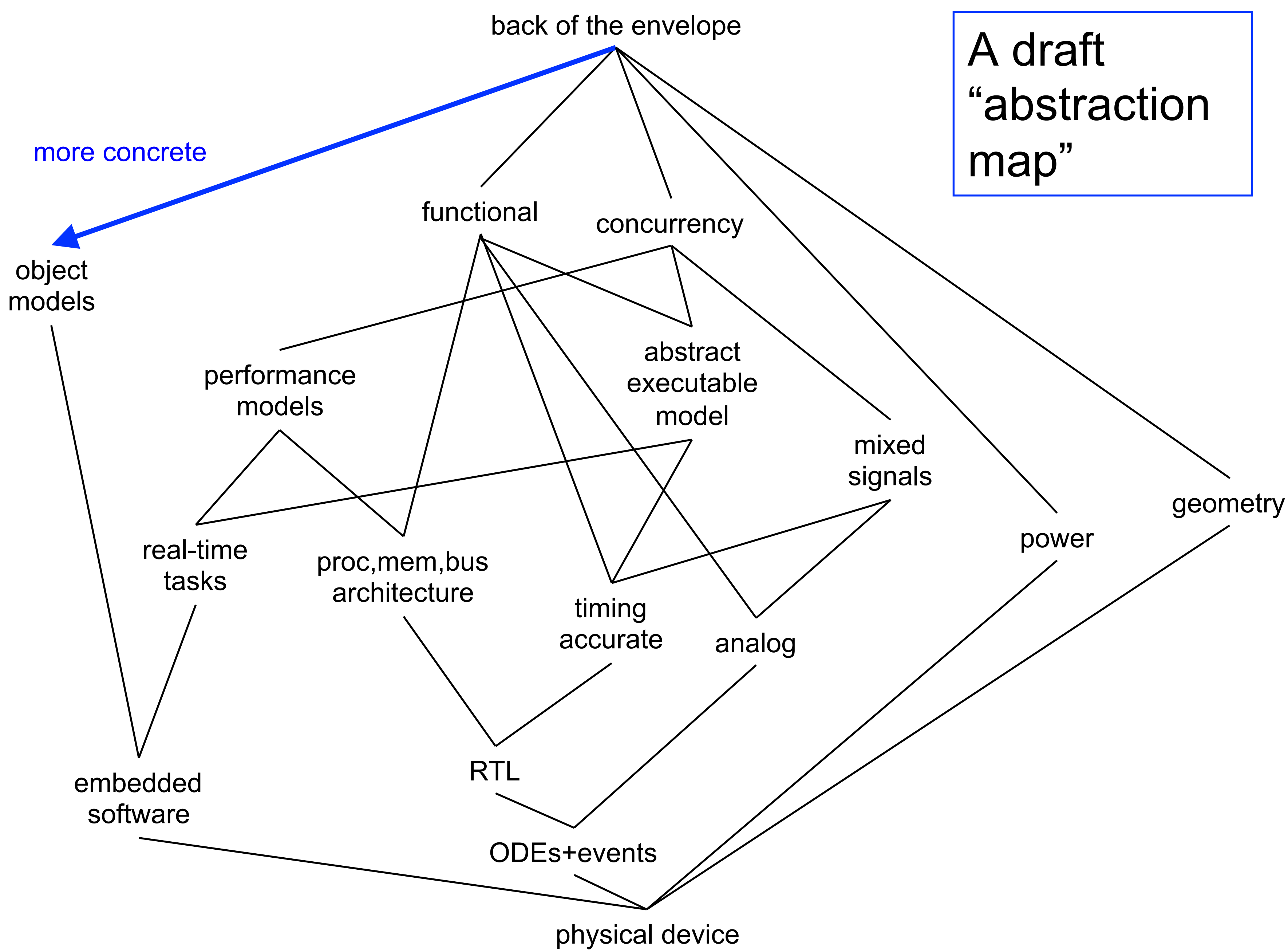



A draft "abstraction map"
back of the envelope
functional
concurrency
object models
abstract executable model
performance models
mixed signals
geometry
power
real-time tasks
proc,mem,bus architecture
timing accurate
analog
embedded software
RTL
ODEs+events
physical device

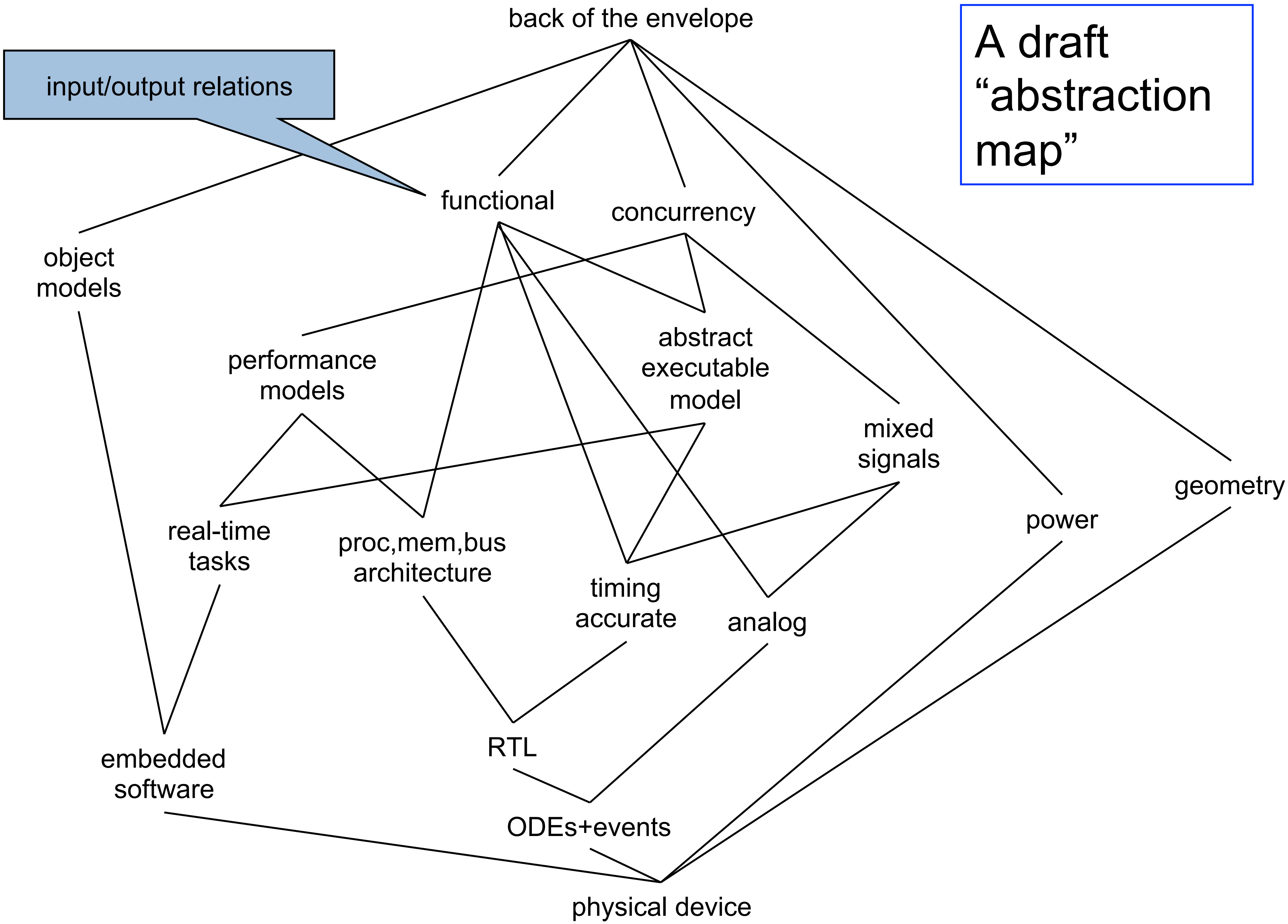
A draft "abstraction map"

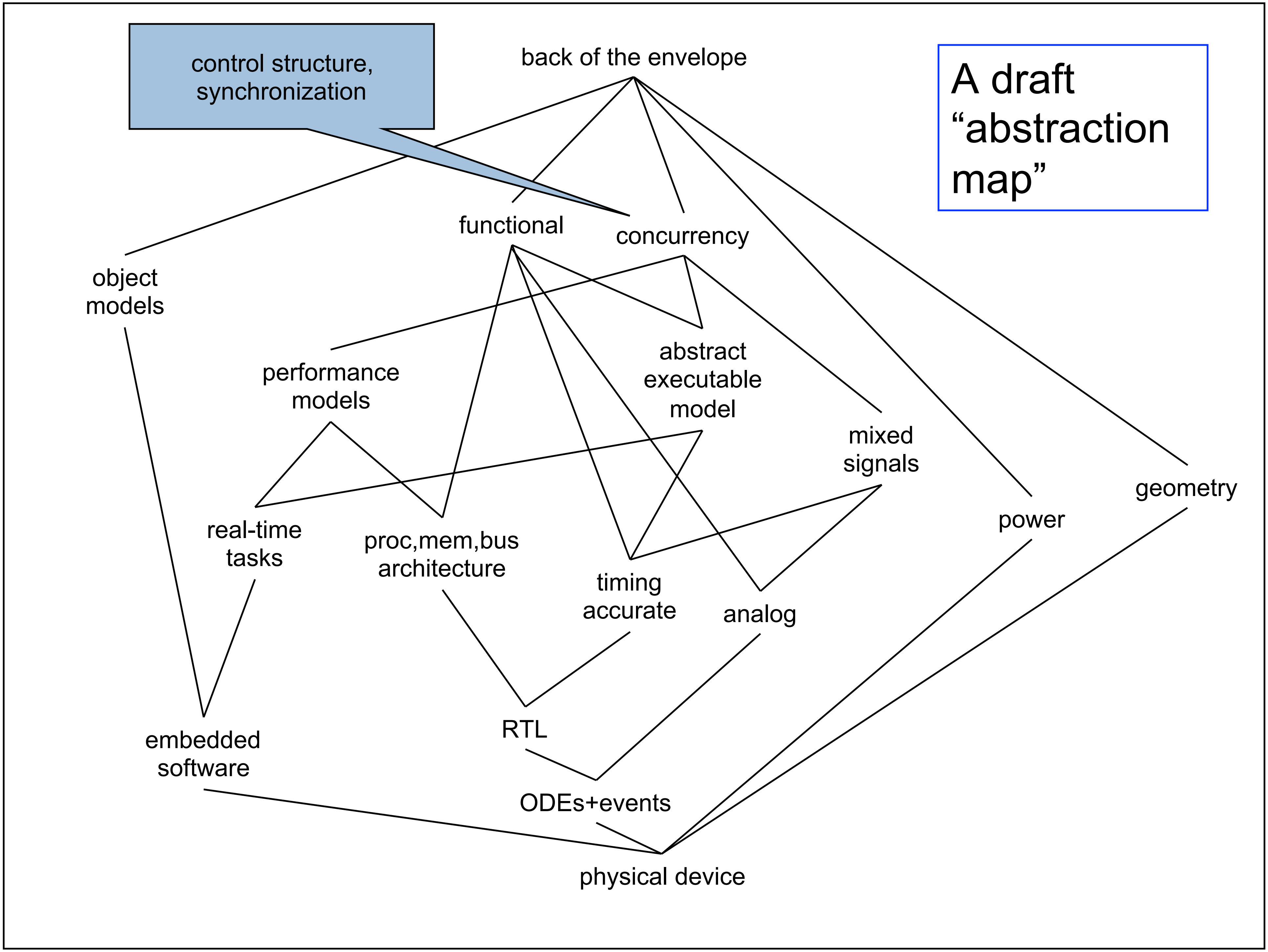


A draft "abstraction map"

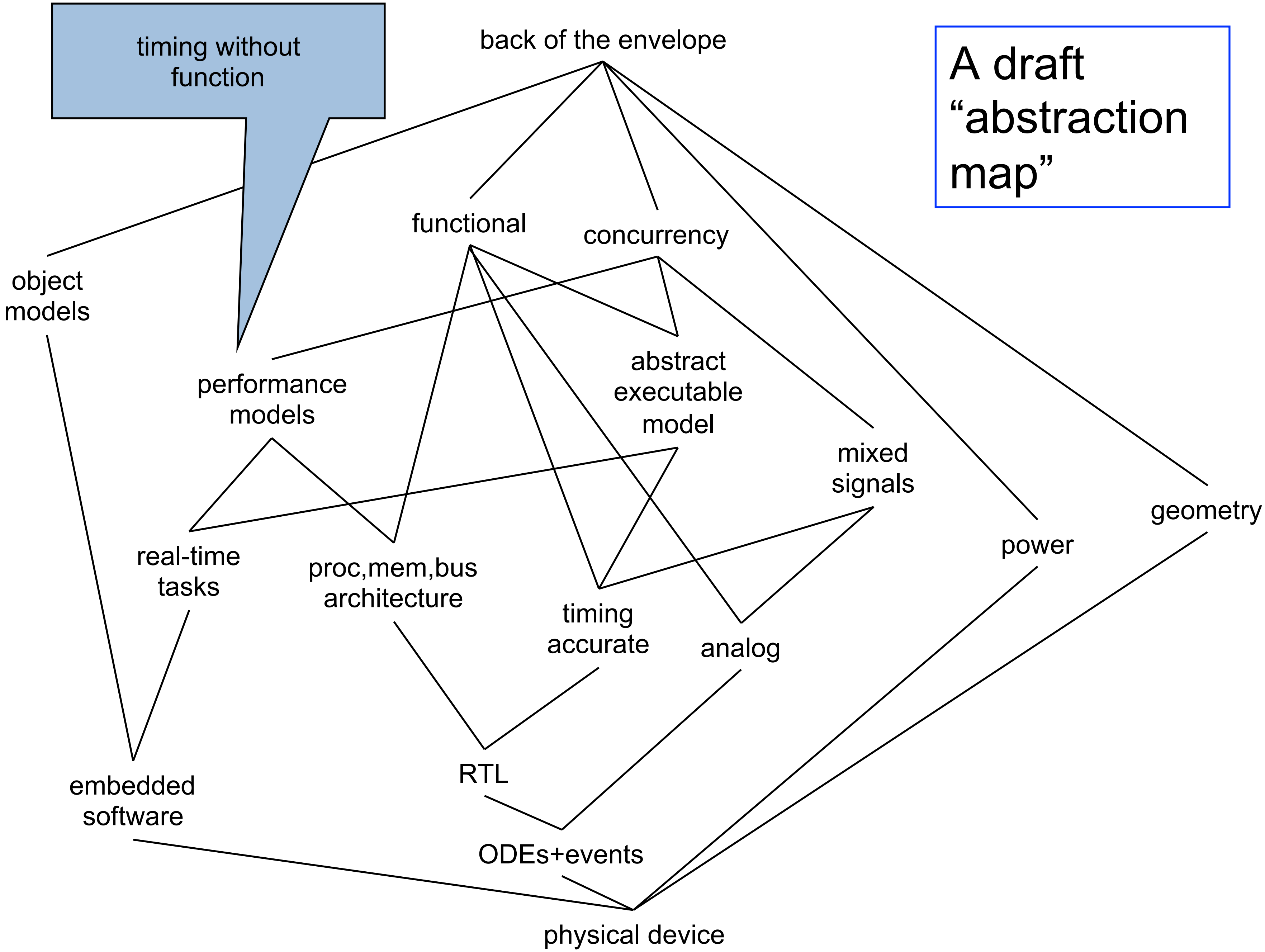


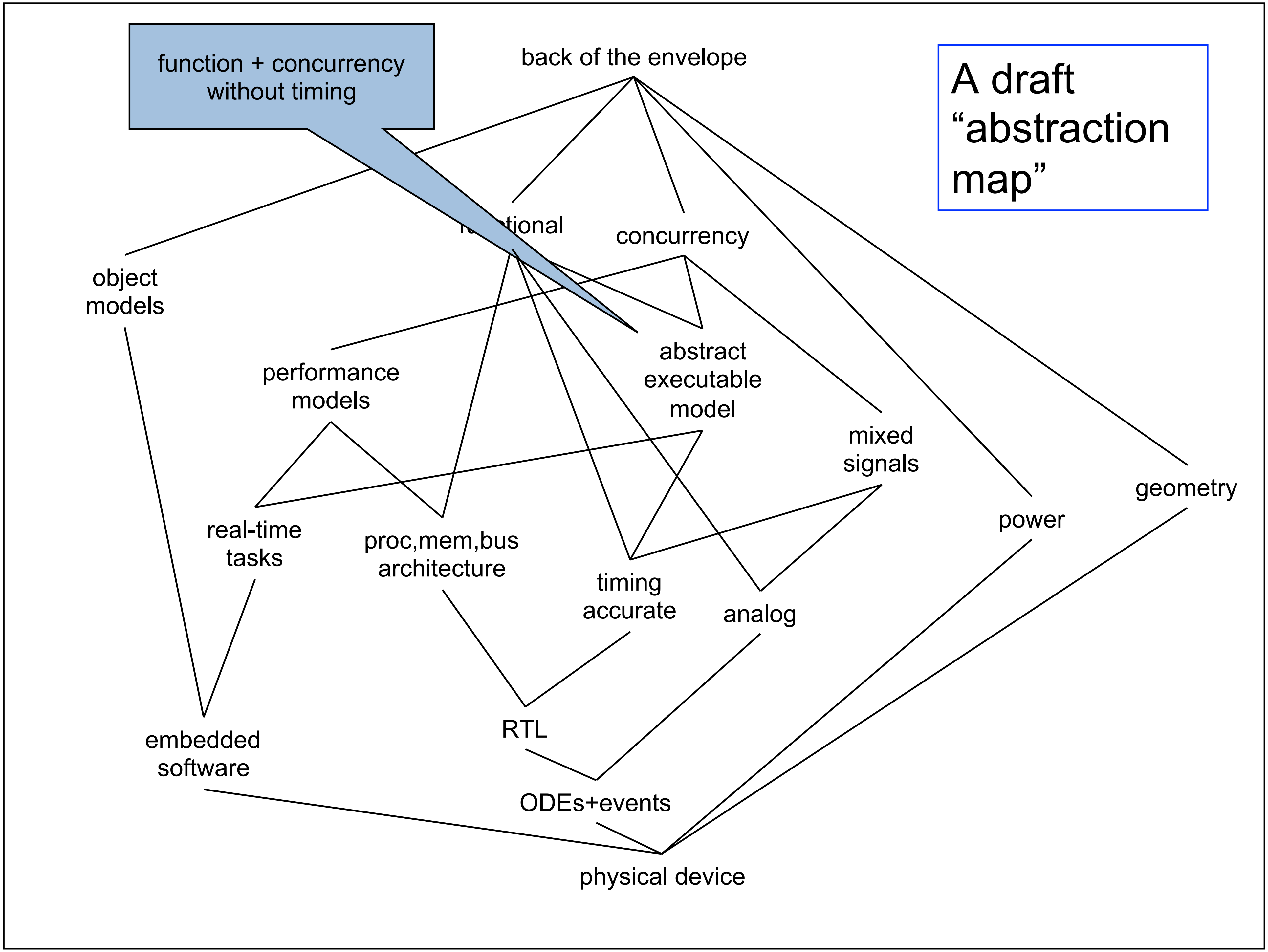
A draft “abstraction map”



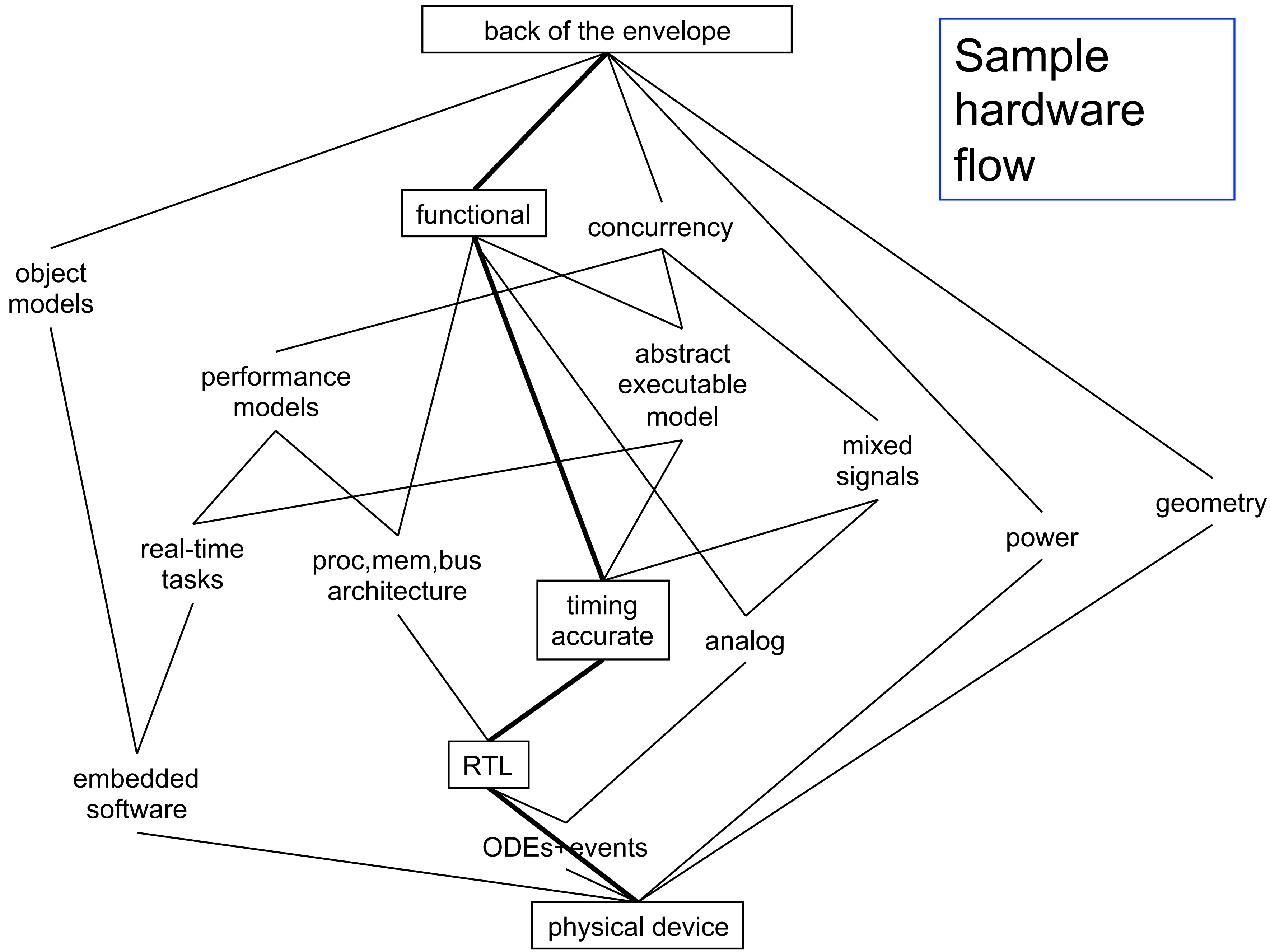

A draft "abstraction map"
control structure, synchronization
back of the envelope
functional
concurrency
object models
performance models
abstract executable model
mixed signals
power
geometry
real-time tasks
proc,mem,bus architecture
timing accurate
analog
embedded software
RTL
ODEs+events
physical device

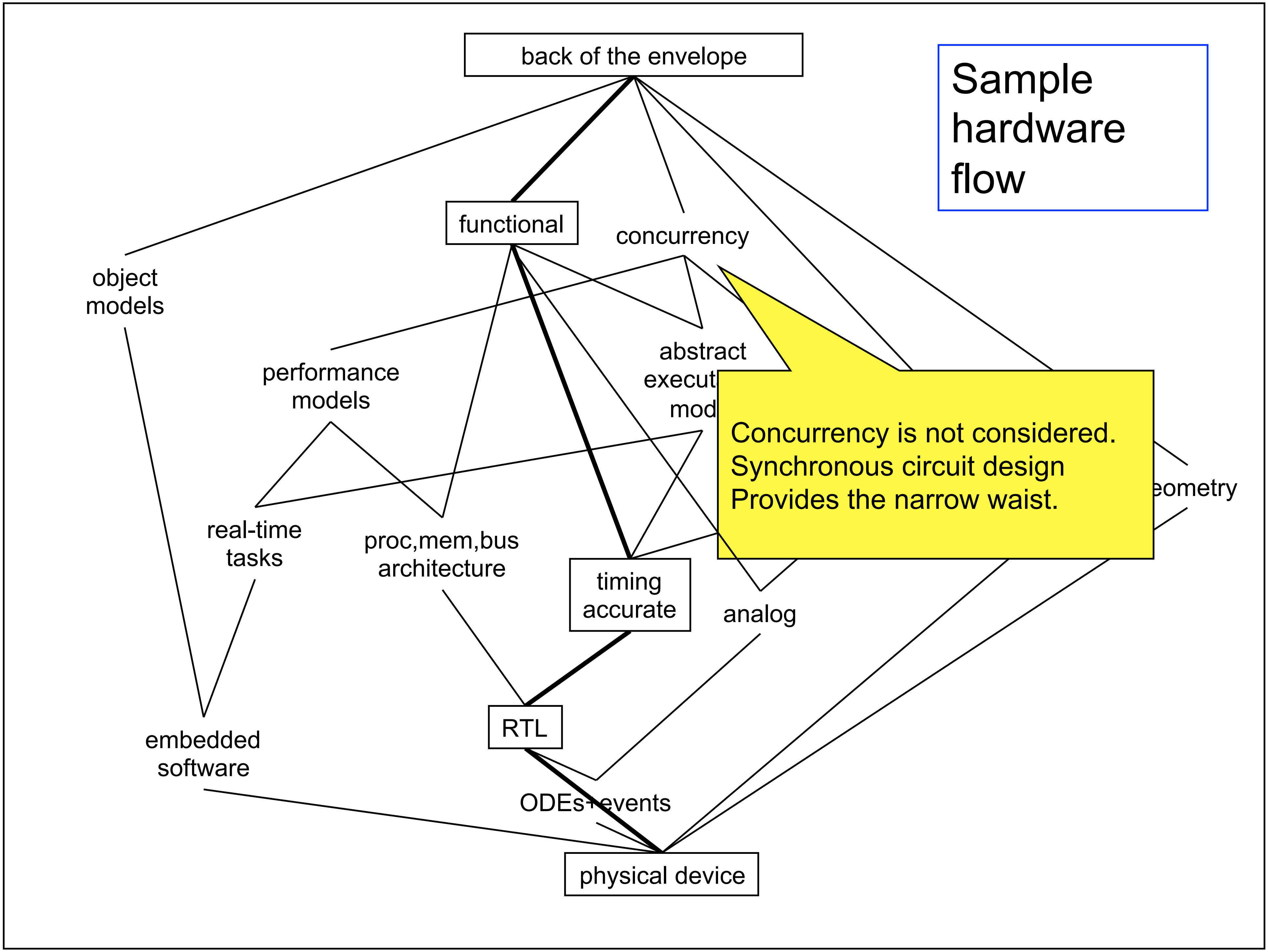
A draft “abstraction map”

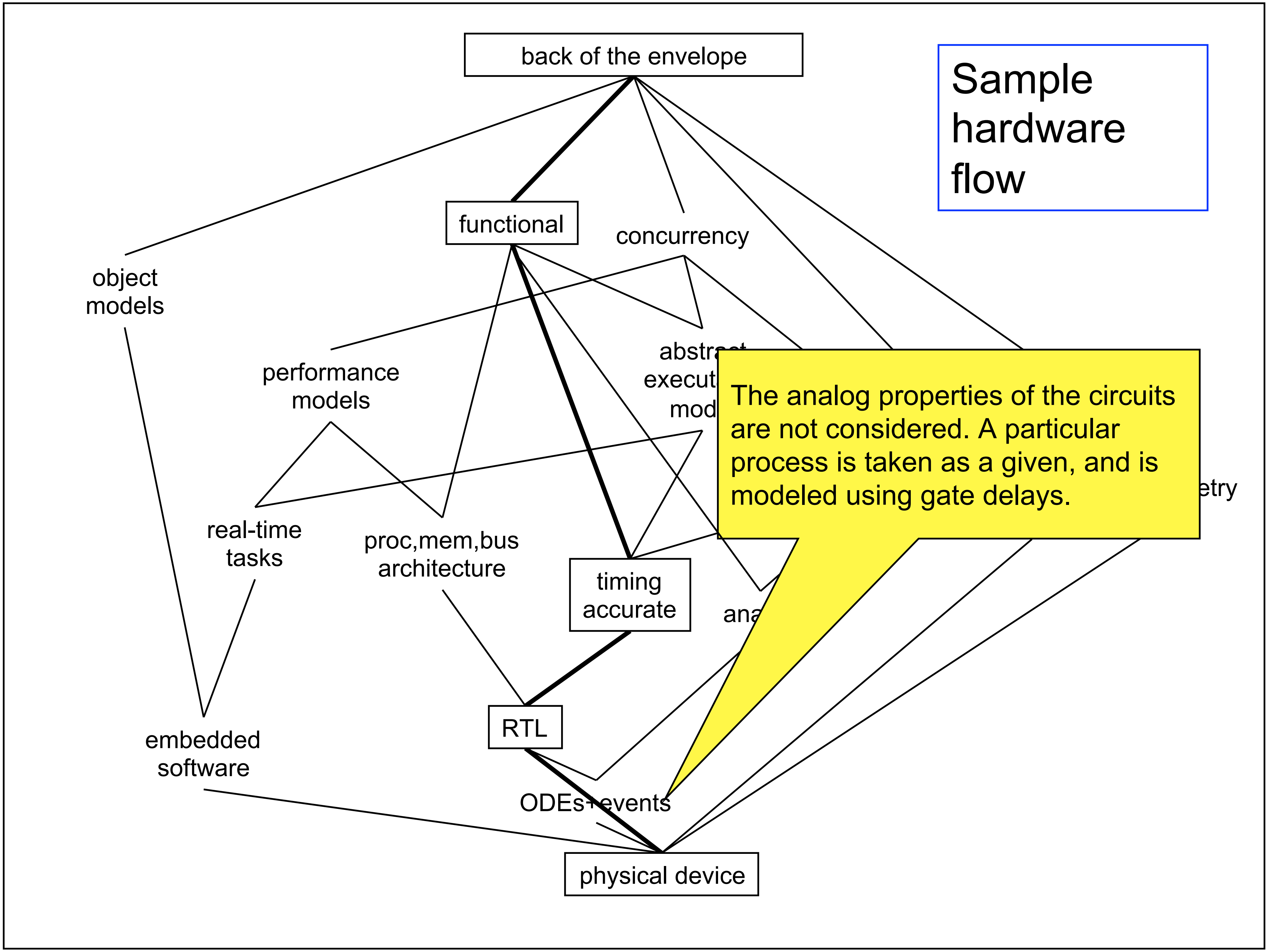



A draft “abstraction map”
function + concurrency without timing
back of the envelope
concurrency
object models
performance models
abstract executable model
mixed signals
geometry
power
real-time tasks
proc,mem,bus architecture
timing accurate
analog
embedded software
RTL
ODEs+events
physical device

Sample hardware flow

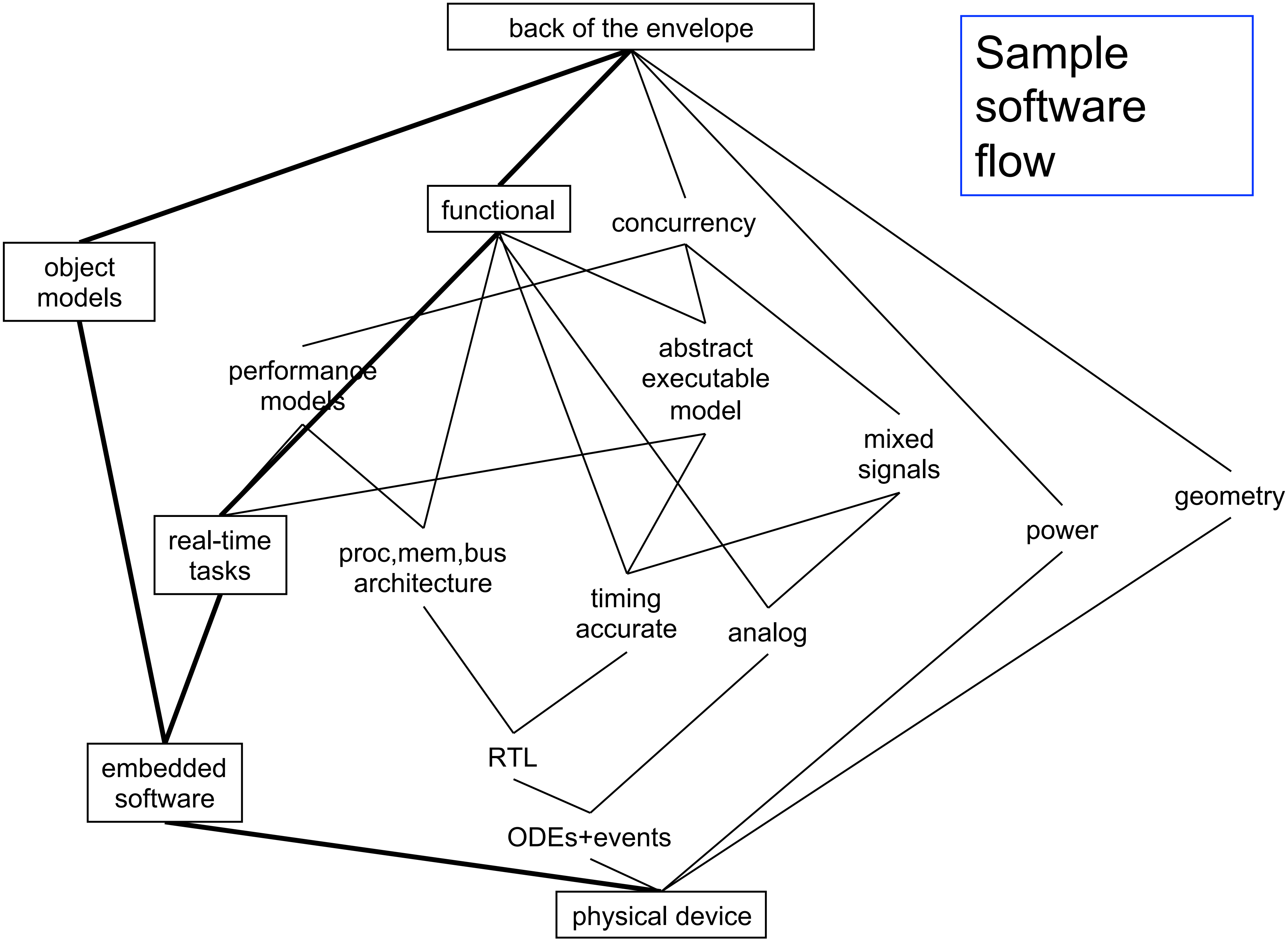


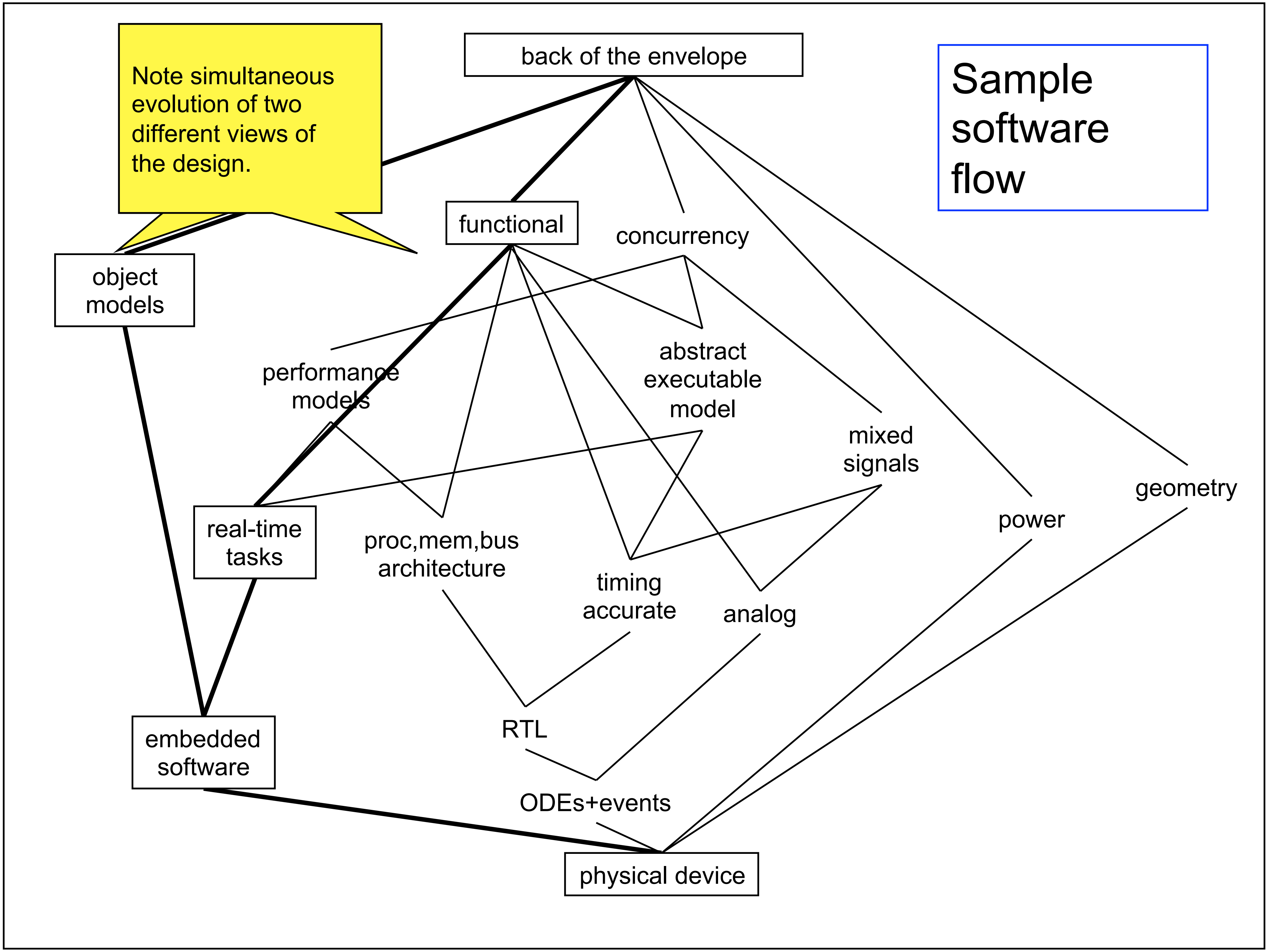

back of the envelope
Sample hardware flow
functional
concurrency
object models
abstract execut
mod
performance models
Concurrency is not considered.
Synchronous circuit design
Provides the narrow waist.
eometry
real-time tasks
proc,mem,bus architecture
timing accurate
analog
RTL
embedded software
ODEs+events
physical device

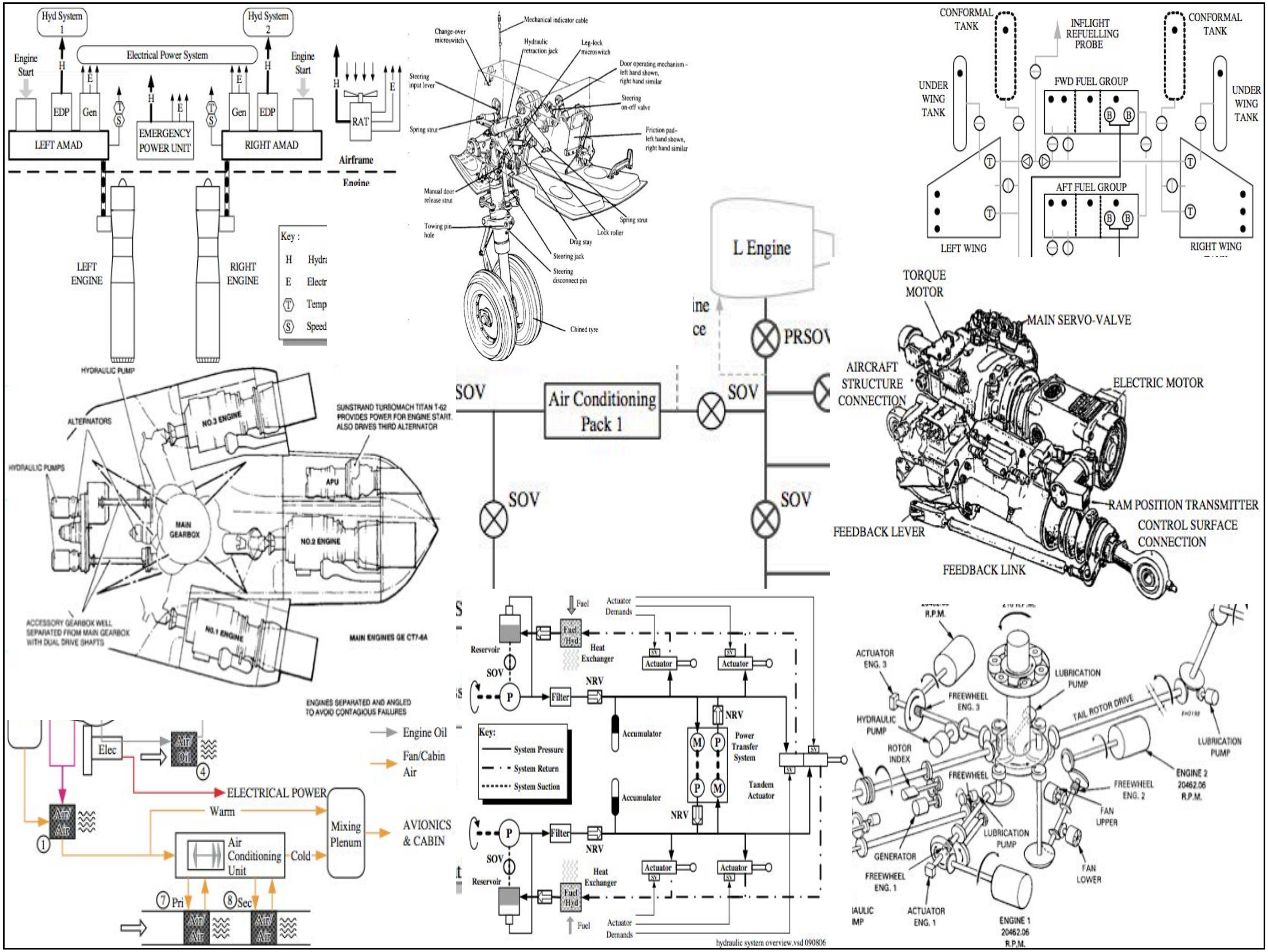


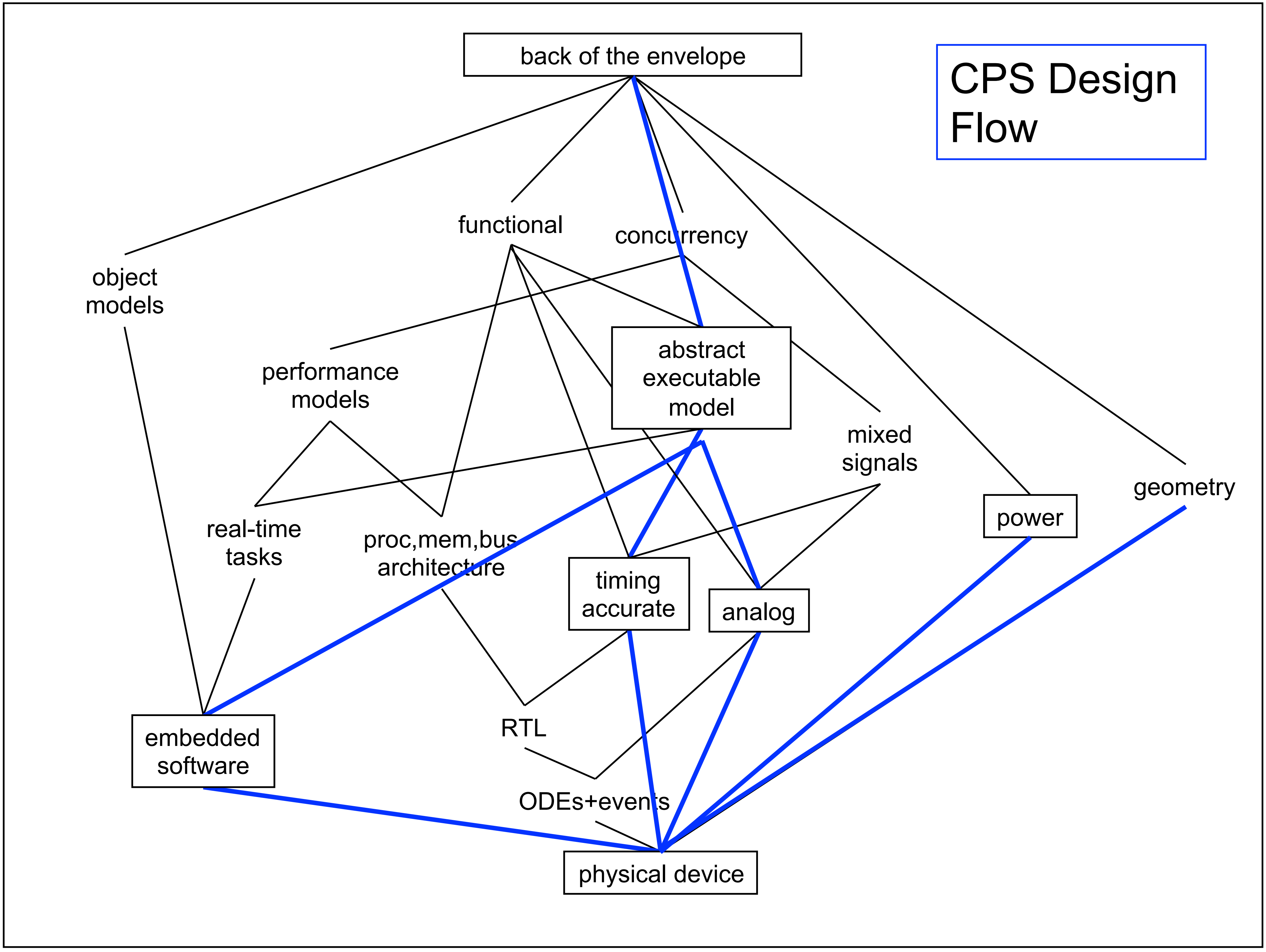
Sample hardware flow
back of the envelope
functional
concurrency
object models
performance models
abstract execution models
real-time tasks
proc,mem,bus architecture
timing accurate
RTL
embedded software
ODEs+events
physical device
The analog properties of the circuits are not considered. A particular process is taken as a given, and is modeled using gate delays.

Sample software flow



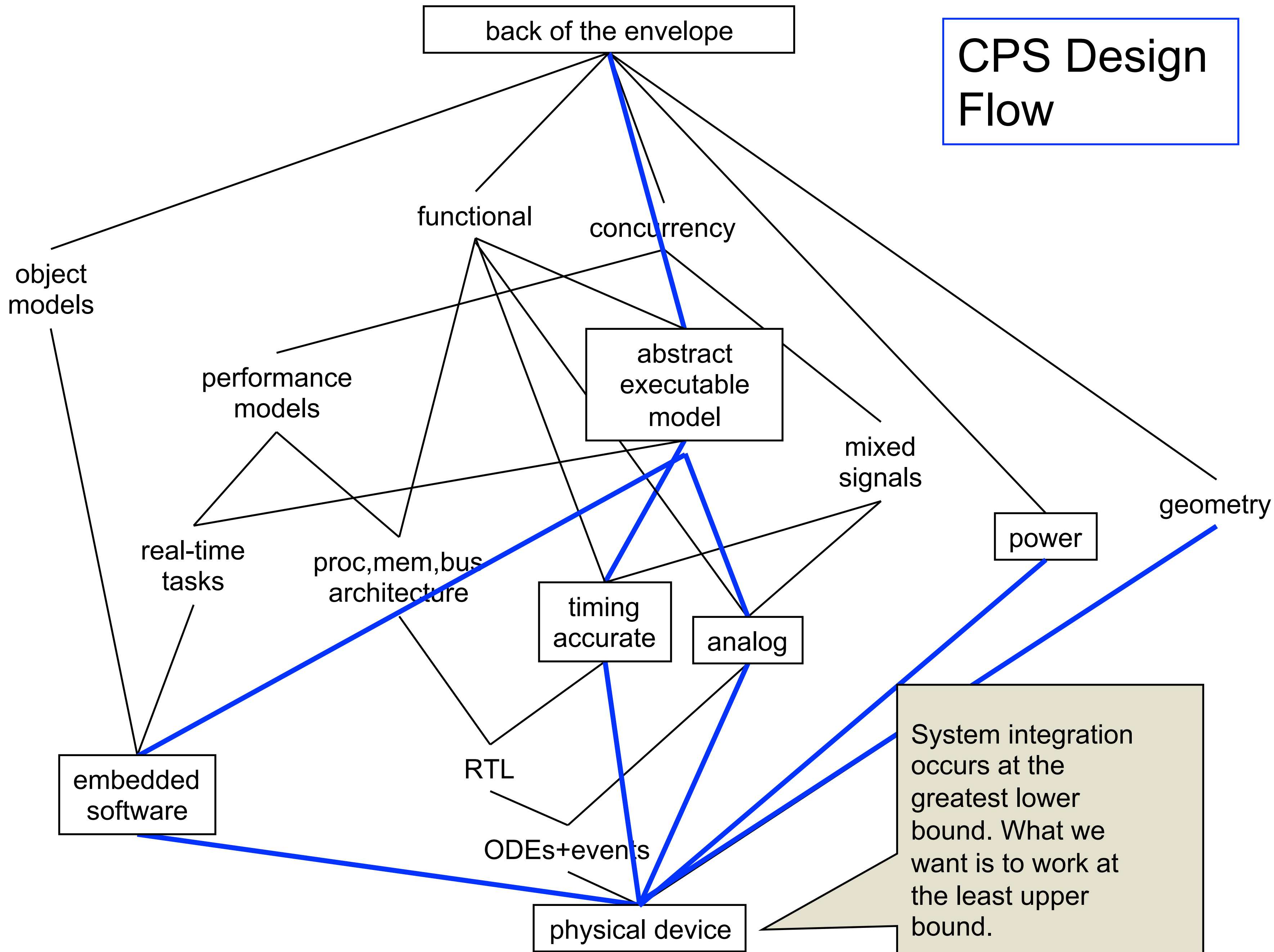

Sample software flow
Note simultaneous evolution of two different views of the design.
back of the envelope
functional
concurrency
object models
performance models
abstract executable model
mixed signals
geometry
power
real-time tasks
proc,mem,bus architecture
timing accurate
analog
embedded software
RTL
ODEs+events
physical device


Hyd System 1
Hyd System 2
Electrical Power System
Engine Start
EDP
Gen
LEFT AMAD
EMERGENCY POWER UNIT
RIGHT AMAD
RAT
Airframe
Engine
LEFT ENGINE
RIGHT ENGINE
Key :
HYDRAULIC PUMP
ALTERNATORS
HYDRAULIC PUMPS
MAIN GEARBOX
NO.3 ENGINE
NO.2 ENGINE
NO.1 ENGINE
APU
SUNSTRAND TURBOMACH TITAN T-62 PROVIDES POWER FOR ENGINE START. ALSO DRIVES THIRD ALTERNATOR
ACCESSORY GEARBOX WELL SEPARATED FROM MAIN GEARBOX WITH DUAL DRIVE SHAFTS
MAIN ENGINES GE CT7-6A
ENGINES SEPARATED AND ANGLED TO AVOID CONTAGIOUS FAILURES
Mechanical indicator cable
Change-over microswitch
Hydraulic retraction jack
Leg-lock microswitch
Door operating mechanism - left hand shown, right hand similar
Steering input lever
Steering on-off valve
Spring strut
Friction pad- left hand shown, right hand similar
Manual door release strut
Towing pin hole
Spring strut
Lock roller
Drag stay
Steering jack
Steering disconnect pin
Chined tyre
CONFORMAL TANK
INFLIGHT REFUELLING PROBE
UNDER WING TANK
FWD FUEL GROUP
AFT FUEL GROUP
LEFT WING
RIGHT WING
L Engine
PRSOV
SOV
Air Conditioning Pack 1
TORQUE MOTOR
MAIN SERVO-VALVE
AIRCRAFT STRUCTURE CONNECTION
ELECTRIC MOTOR
RAM POSITION TRANSMITTER
CONTROL SURFACE CONNECTION
FEEDBACK LEVER
FEEDBACK LINK
Elec
Engine Oil
Fan/Cabin Air
ELECTRICAL POWER
Warm
Mixing Plenum
AVIONICS & CABIN
Air Conditioning Unit
Cold
Pri
Sec
Fuel
Actuator Demands
Fuel /Hyd
Heat Exchanger
Reservoir
Actuator
NRV
Filter
Accumulator
Power Transfer System
Tandem Actuator
Key:
System Pressure
System Return
System Suction
hydraulic system overview.vsd 090806
ACTUATOR ENG. 3
FREEWHEEL ENG. 3
LUBRICATION PUMP
TAIL ROTOR DRIVE
HYDRAULIC PUMP
ROTOR INDEX
FREEWHEEL
ENGINE 2 20462.06 R.P.M.
FREEWHEEL ENG. 2
FAN UPPER
GENERATOR
LUBRICATION PUMP
FAN LOWER
FREEWHEEL ENG. 1
ACTUATOR ENG. 1
ENGINE 1 20462.06 R.P.M.



CPS Design Flow
back of the envelope
functional
concurrency
object models
performance models
abstract executable model
mixed signals
power
geometry
real-time tasks
proc,mem,bus architecture
timing accurate
analog
embedded software
RTL
ODEs+events
physical device

CPS Design Flow

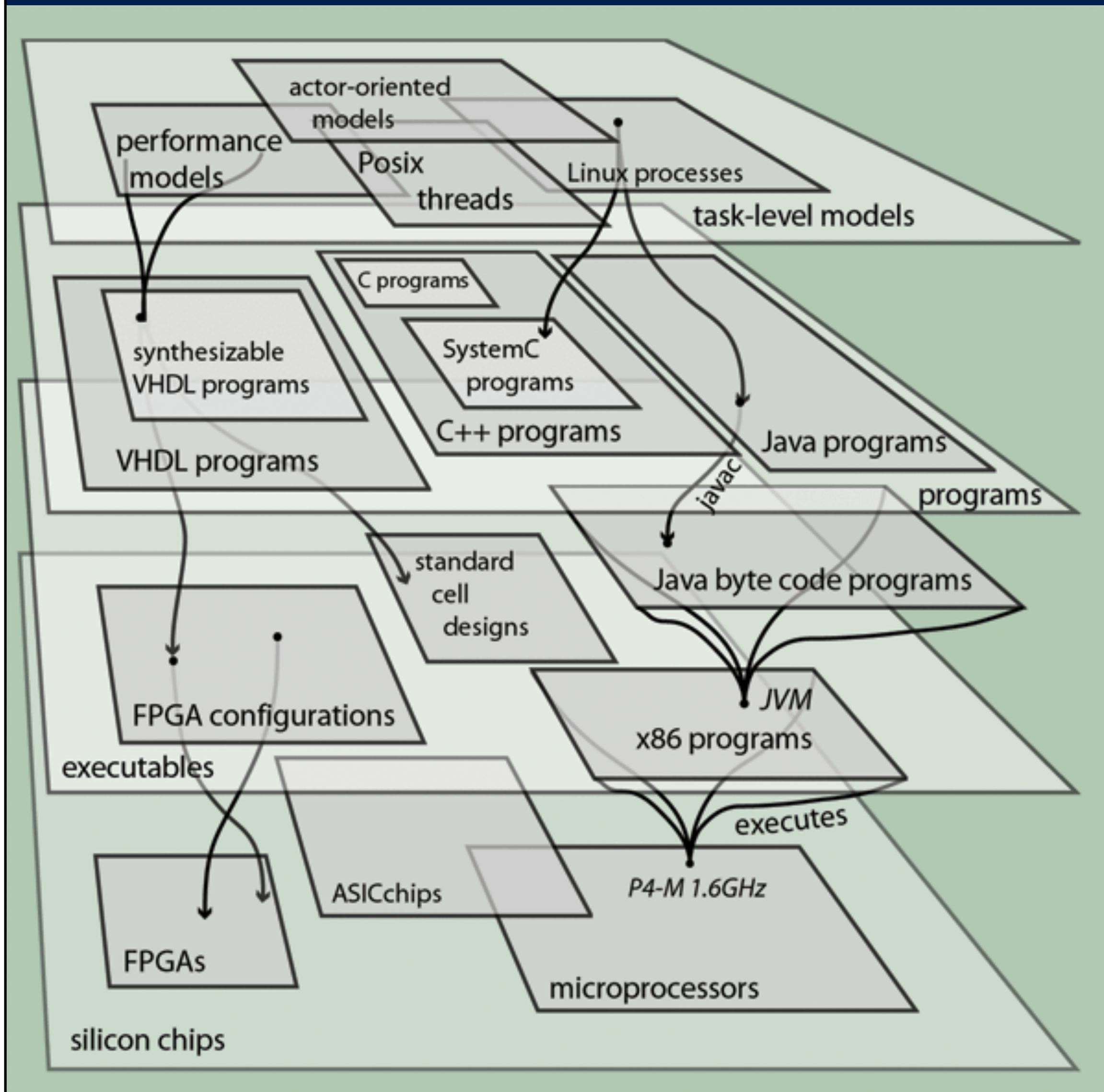


A Story

In "fly by wire" aircraft, computers control the plane, mediating pilot commands.

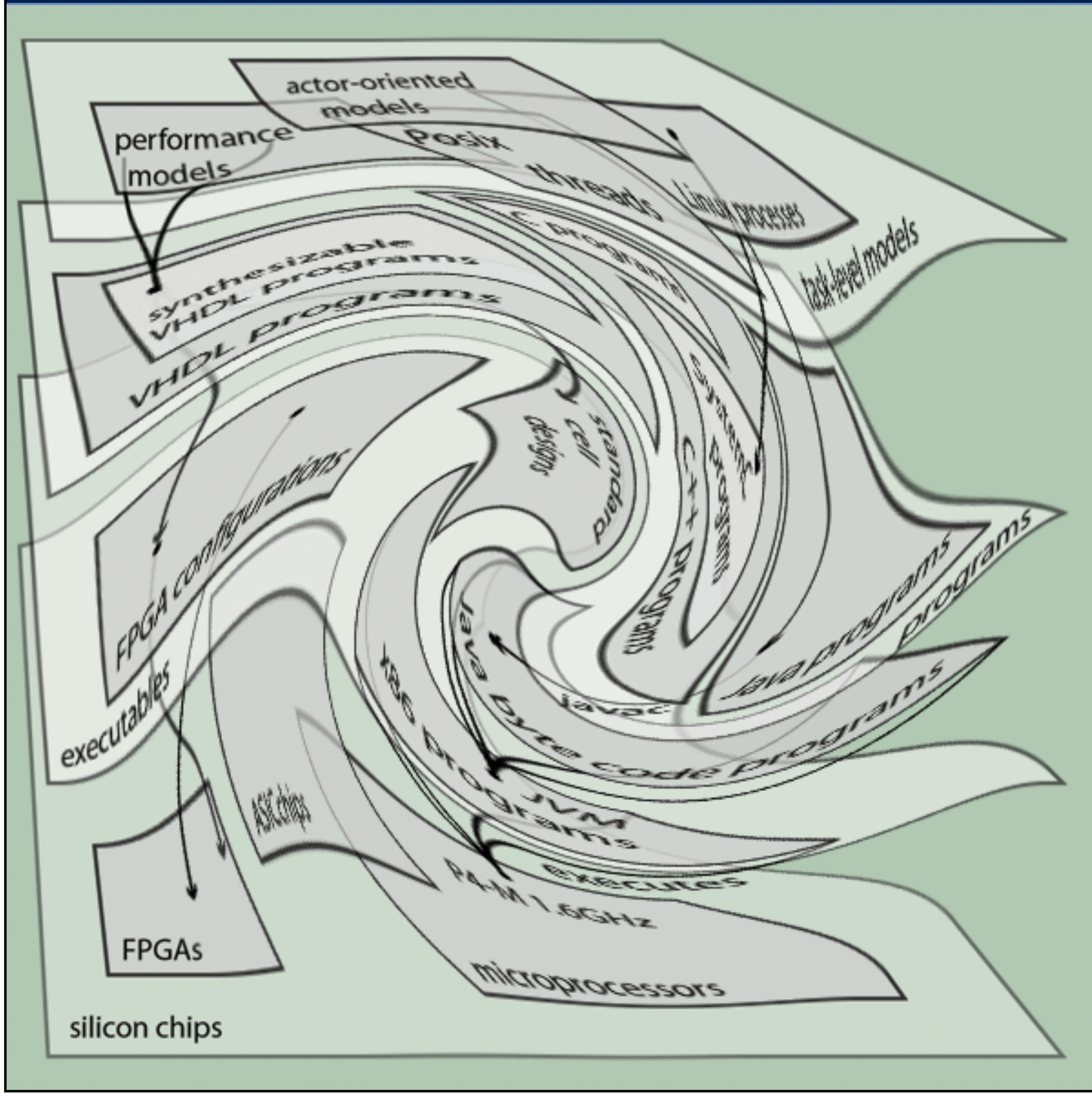
CCA 2.0
Boeing Dreamscape

Abstraction Layers
All of which are models except the bottom



The purpose of an abstraction is to hide details of the implementation below and provide a platform for design from above.

Abstraction Layers
All of which are models except the bottom



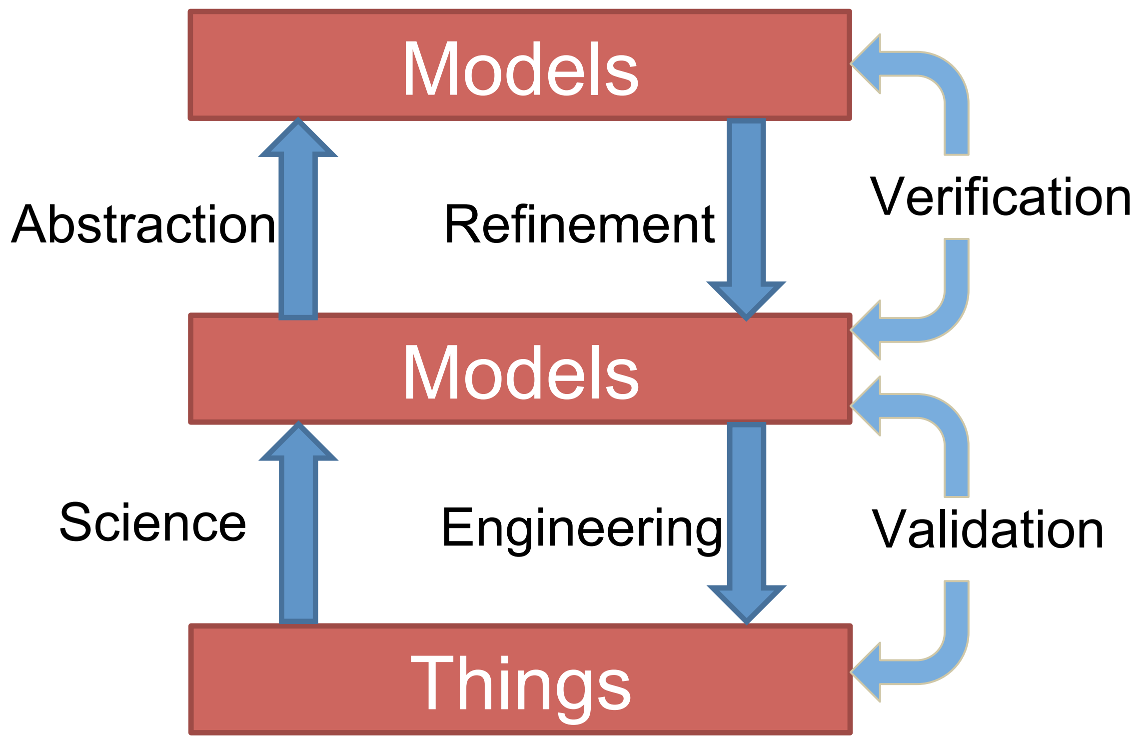
Every abstraction layer has failed for the aircraft designer.

The design *is* the implementation.

Verification and Validation

Per Boehm:

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

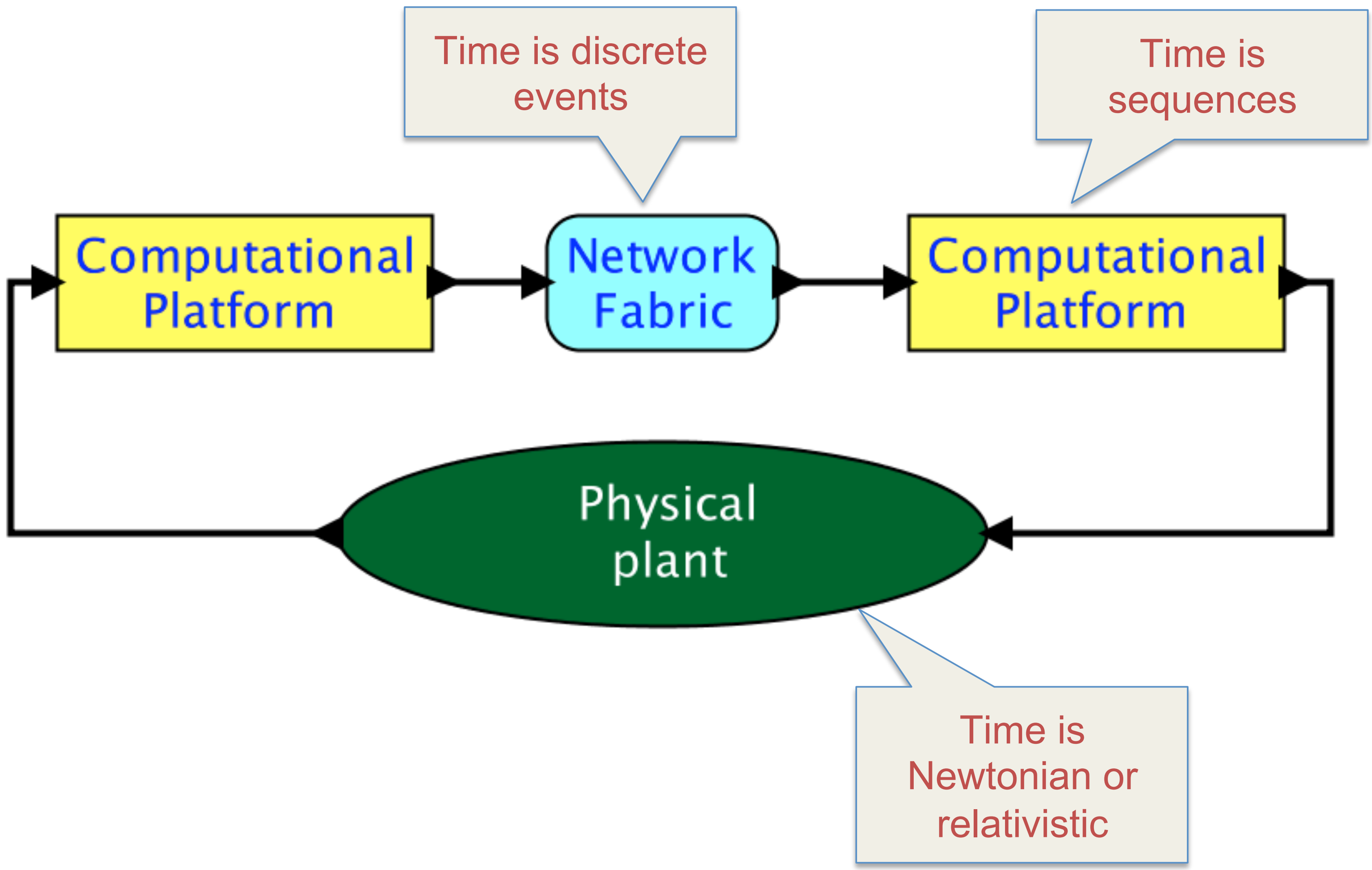


Formal methods can only address the Verification question.

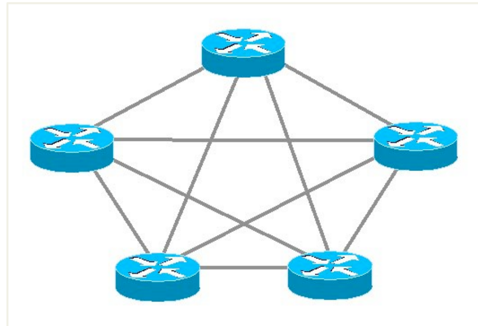
Outline

- Cyber-Physical Systems (CPS)
- Challenges
- Models
- Determinism
- Limits of Determinism
- Abstraction and Refinement
- Time

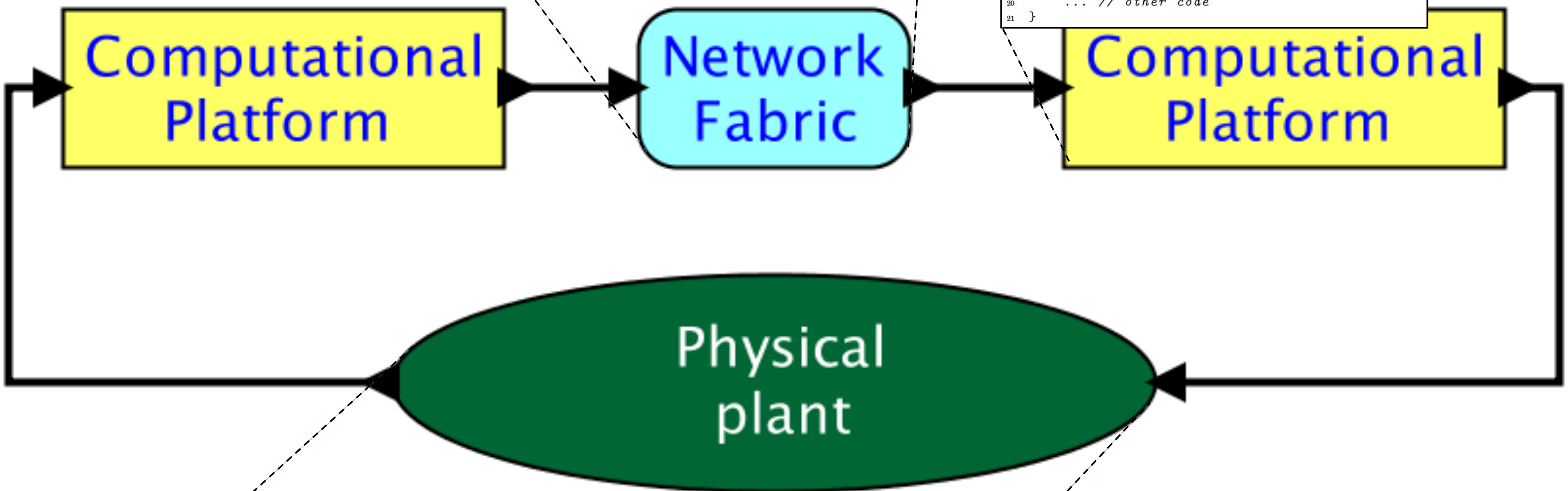
Cyber-Physical Systems Pattern



The models are more deterministic than the reality



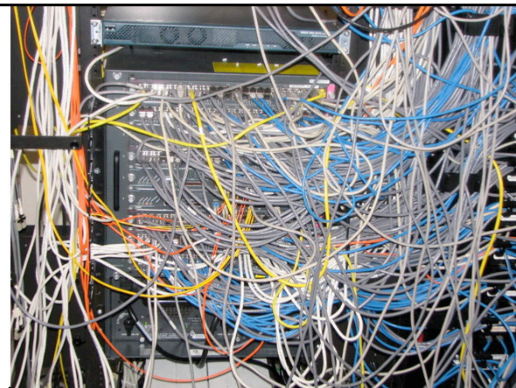
```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count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

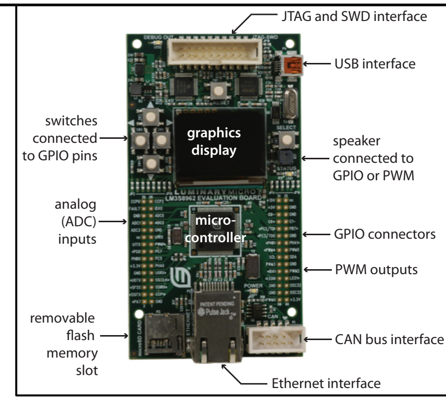


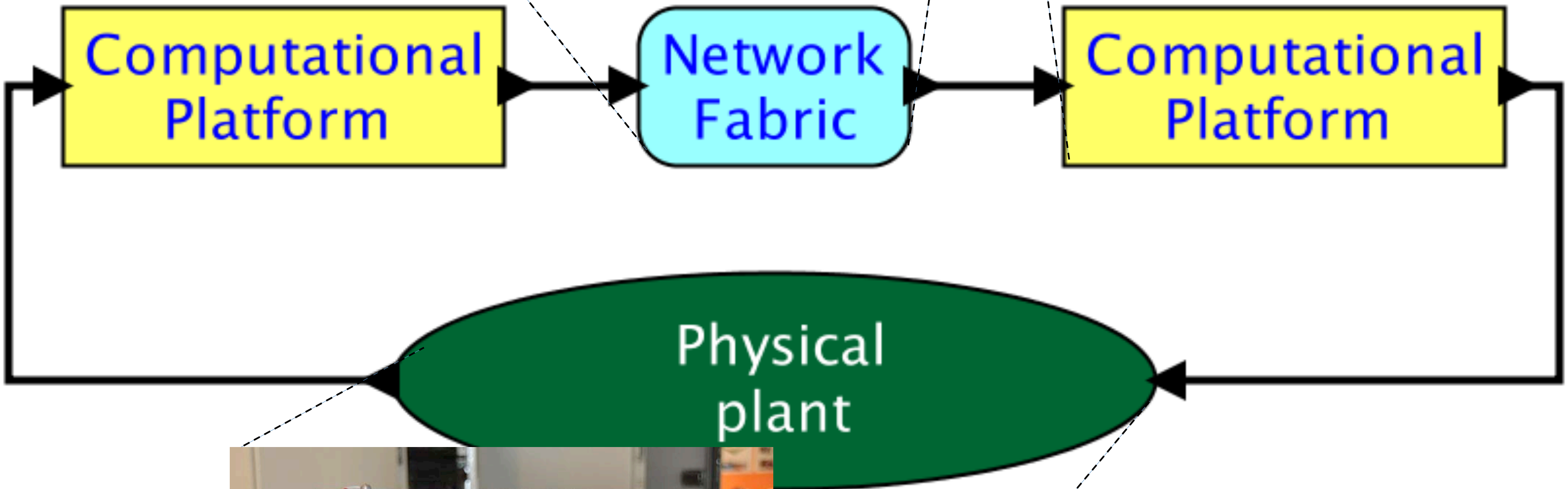
$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \mathbf{F}(\tau)d\tau$$

The modeling languages have disjoint, incompatible semantics

System dynamics emerges from the physical realization



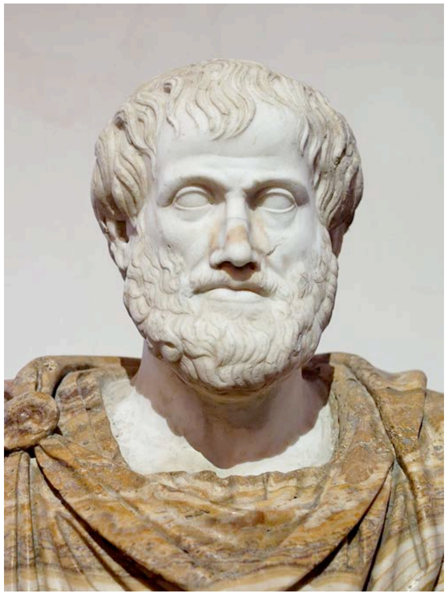
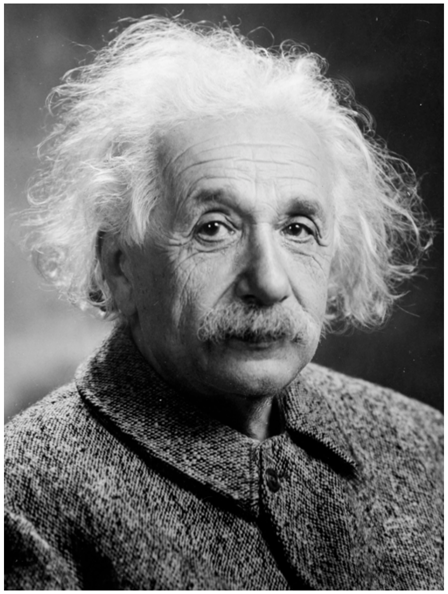
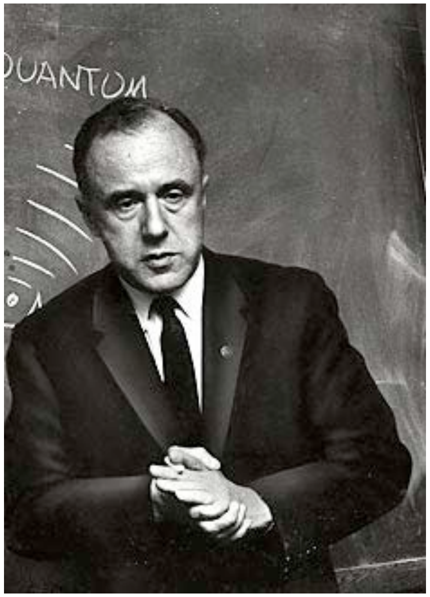




… leading to a "prototype and test" style of design

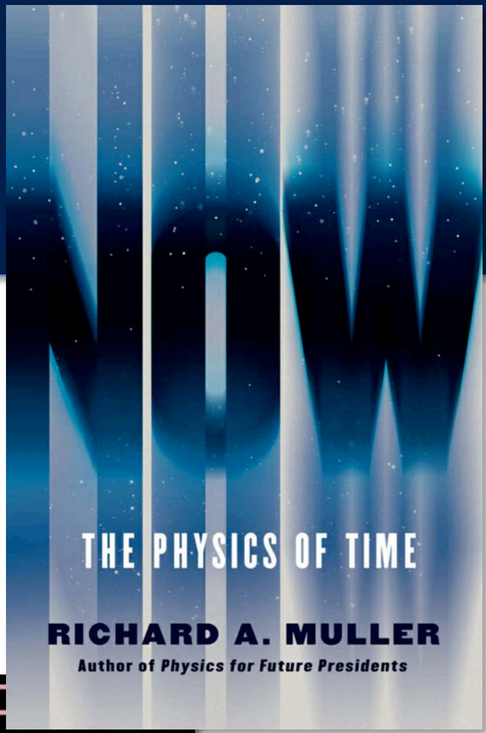
Image: Wikimedia Commons

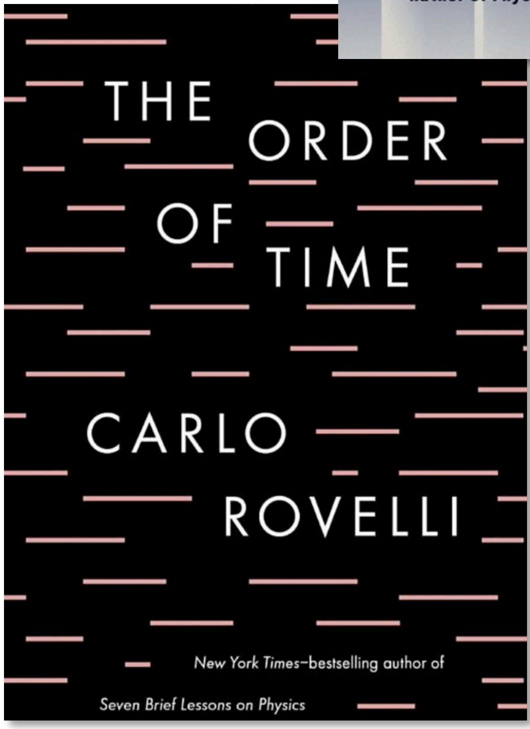
What is Time?

Change	Smooth	Relative	Discrete
			
Aristotle	Newton	Einstein	Wheeler

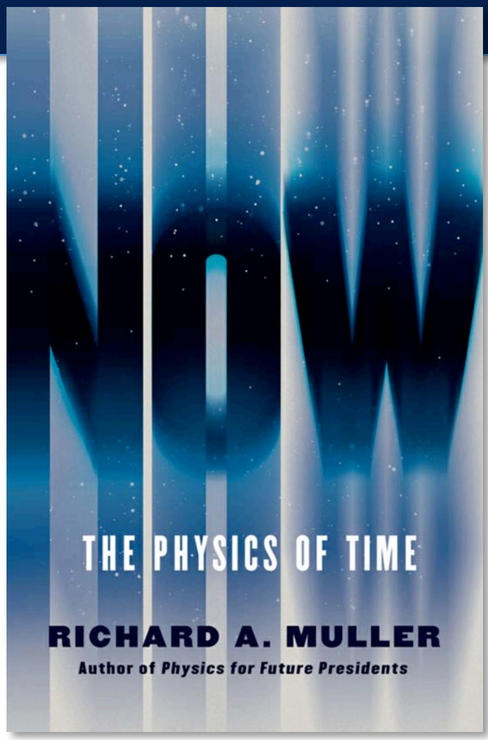
What is Time from a Physics Perspective?

Change?
Uniformly advancing in a continuum?
Arrow of increasing entropy?
Expansion of space-time?
Discrete or continuous?
Total or partial order?
A cognitive illusion?
An anthropic fact?

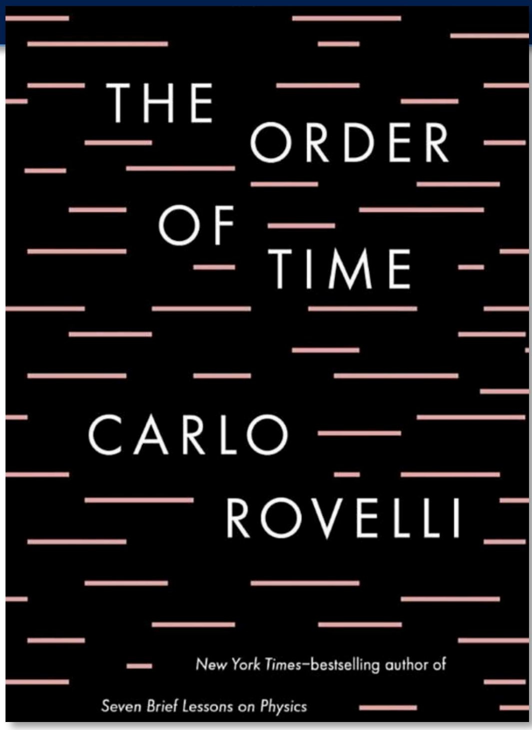




What is Time?



2016



2018

Muller: Gives a theory of time that requires big black holes to collide somewhere near us to test it.

Rovelli: “The nature of time is perhaps the greatest remaining mystery.”

How can we build systems based on something we do not understand

?

Deterministic Timing

Is our goal for our models to accurately reflect the timing of our implementation?

or

Is our goal for the implementation to accurately reflect the timing of the model?

If it's the latter, then deterministic timing makes perfect sense!

Desirable Properties in a Model of Time

- A "present" that separates the past and future
  - Needed for a notion of "state"
- Support for causality
  - If *A* causes *B,* then every observer should see *A* before *B.*
- A well-defined "observer"
  - Otherwise, stuck trying to solve the physics problem.
- A notion of "simultaneity"
  - If A can precede B and B can precede A, then A can be simultaneous with B

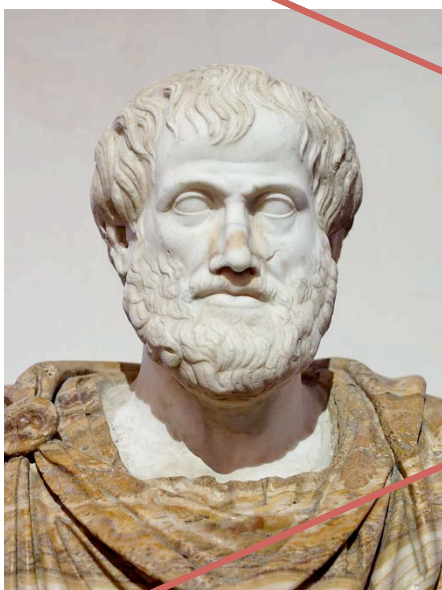
Desirable Properties in a Model of Time

- A “present” that separates the past and future
- Support for causality
- A well-defined “observer”
- A notion of “simultaneity”

All are problematic in physics but useful in models.

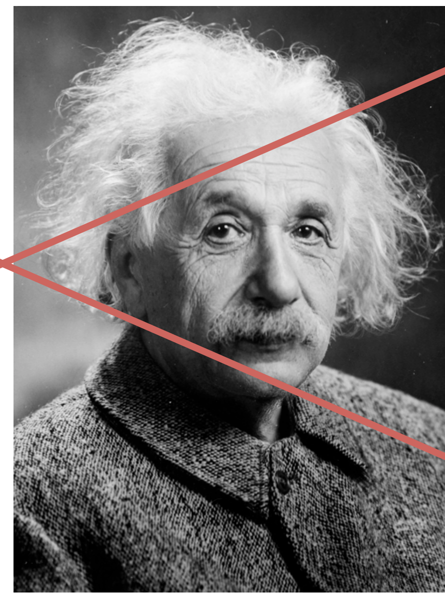
What is Time?

Change | Smooth | Relative | Discrete

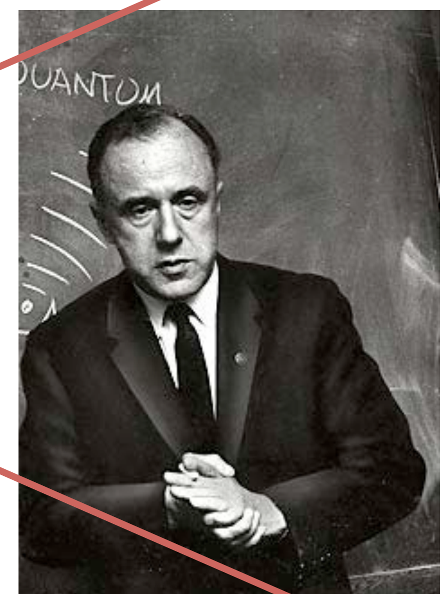


Aristotle

Newton



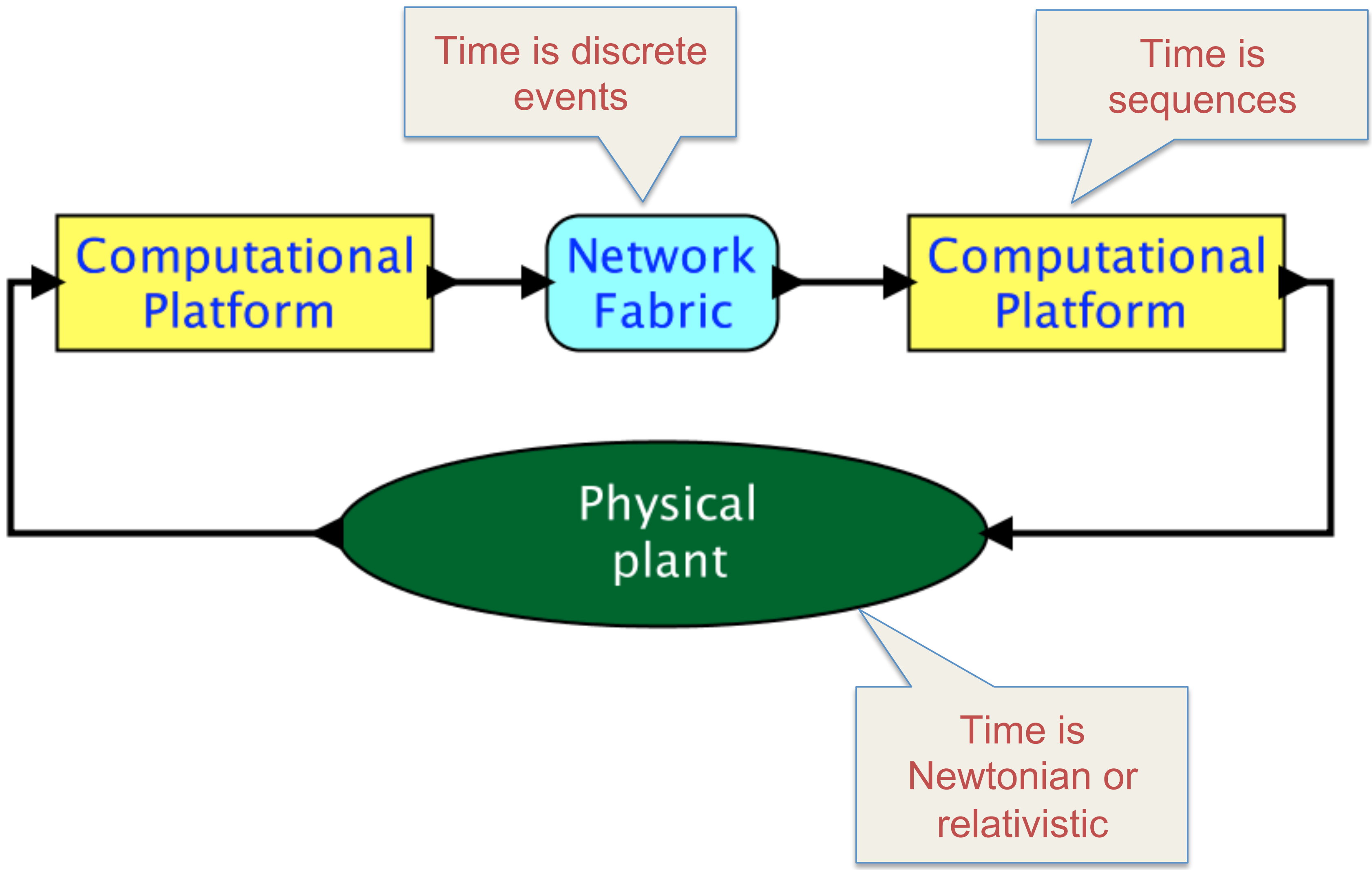
Einstein



Wheeler

All of these are about scientific models, not engineering models.

Cyber-Physical Systems Pattern



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

```
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
}
```

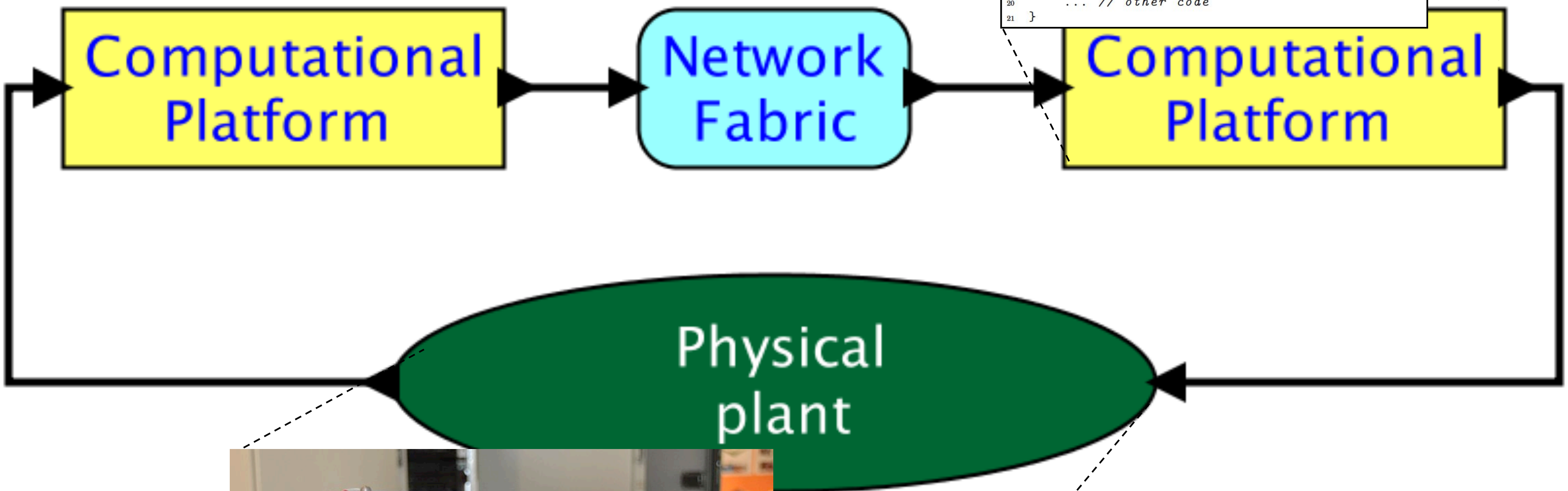
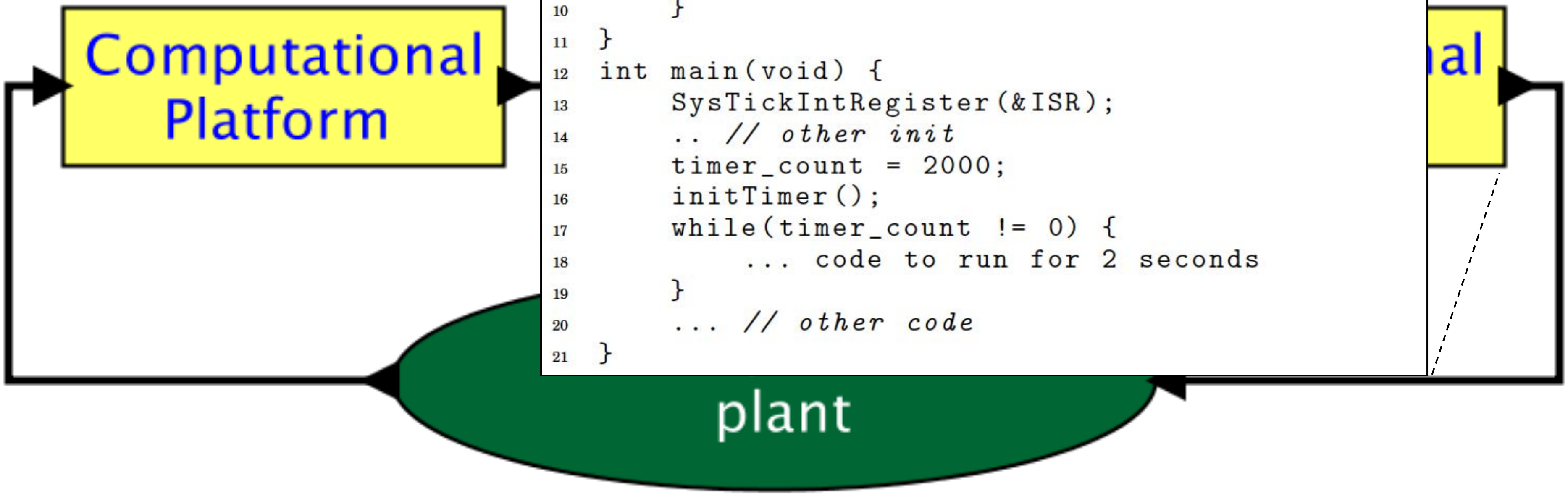


Image: Wikimedia Commons

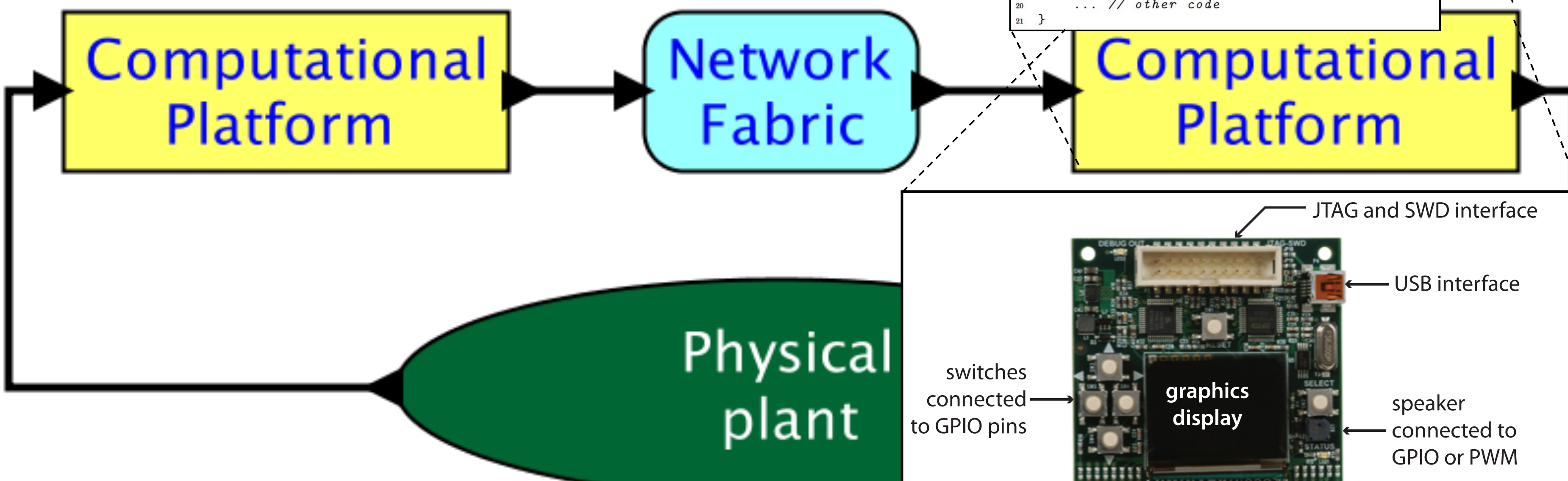
```c
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    .. // other init
    timer_count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

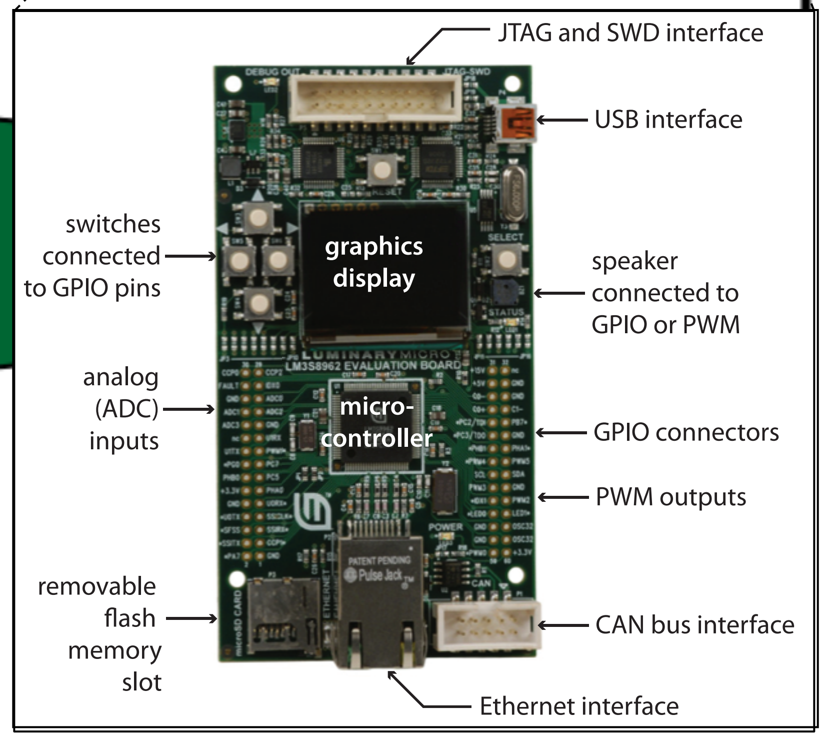


This code is attempting to control timing. But will it really?

Emergent Timing

```
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.

Stellaris LM3S8962 evaluation board (Luminary Micro 2008, now Texas Instruments)

Frozen Designs

Everything about the design, down to wire lengths and microprocessor chips, must be frozen at the time of design.

CCA 2.0
Boeing Dreamscape

Contrast with correctness criteria in software

We can safely assert that line 8 does not execute, *regardless of the choice of microprocessor*!

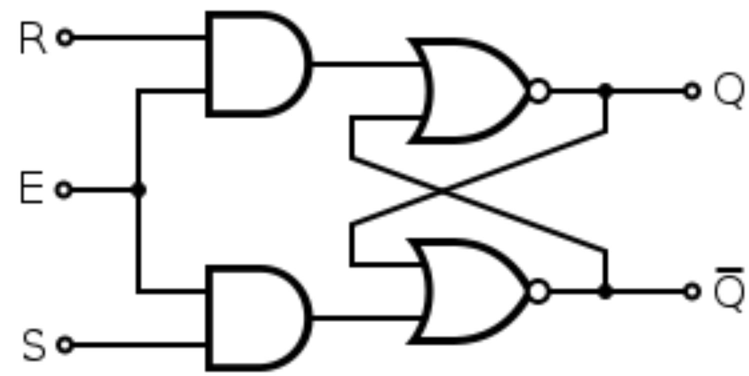
```
1  void foo(int32_t x) {
2          if (x > 1000) {
3                  x = 1000;
4          }
5          if (x > 0) {
6                  x = x + 1000;
7                  if (x < 0) {
8                          panic();
9                  }
10         }
11 }
```

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

But not with regards to timing!!

Hardware is Good at Timing

Synchronous digital logic delivers precise, repeatable timing.



… but the overlaying software abstractions discard timing.

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

Projects at Berkeley Focused on *Engineering Models* for CPS

Deterministic models for CPS:

- PTIDES: distributed real-time software
  - http://chess.eecs.berkeley.edu/ptides
- PRET: time-deterministic architectures
  - http://chess.eecs.berkeley.edu/pret
- Lingua Franca: a programming model
  - https://github.com/icyphy/lingua-franca

Together, these technologies give a model for distributed and concurrent real-time systems that is deterministic, has controlled timing, and is implementable.

Changing the Question:

Is the question whether we can build models describing the behavior of cyber-physical systems?

Or

Is the question whether we can make cyber-physical systems that behave like our models?