PSI 3442 Projeto de Sistemas Embarcados 2020

Fechamento do Curso

Cyber-Physical Systems (CPS)
Contradictory Requirements

It's not just information technology anymore:

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

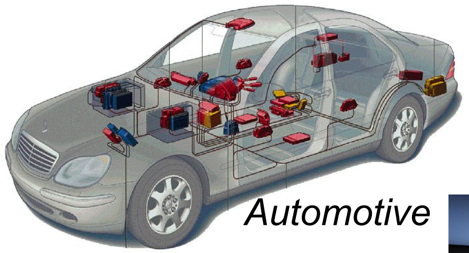
Contradictions:

- Adaptability vs. Repeatability
- High connectivity vs. *Security and Privacy*
- High performance vs. *Low Energy*
- Asynchrony vs. *Coordination/Cooperation*
- Scalability vs. *Reliability and Predictability*
- Laws and Regulations vs. *Technical Possibilities*
- Economies of scale (cloud) vs. *Locality (fog)*
- Open vs. *Proprietary*
- Algorithms vs. *Dynamics*

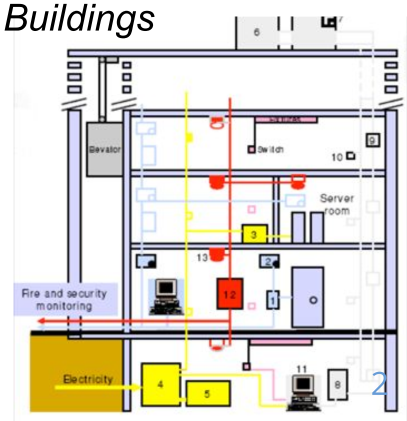
Innovation:

Cyber-physical systems require new engineering methods and models to address these contradictions.

Lee, Berkeley


Automotive

Manufacturing



Play Detective

In the real world, it’s more likely you will be working with existing (legacy) embedded systems rather than building your own from scratch.

Let’s go through a few stories of large scale design issues or failures with industrial embedded systems and try to guess what topics from the course can help.

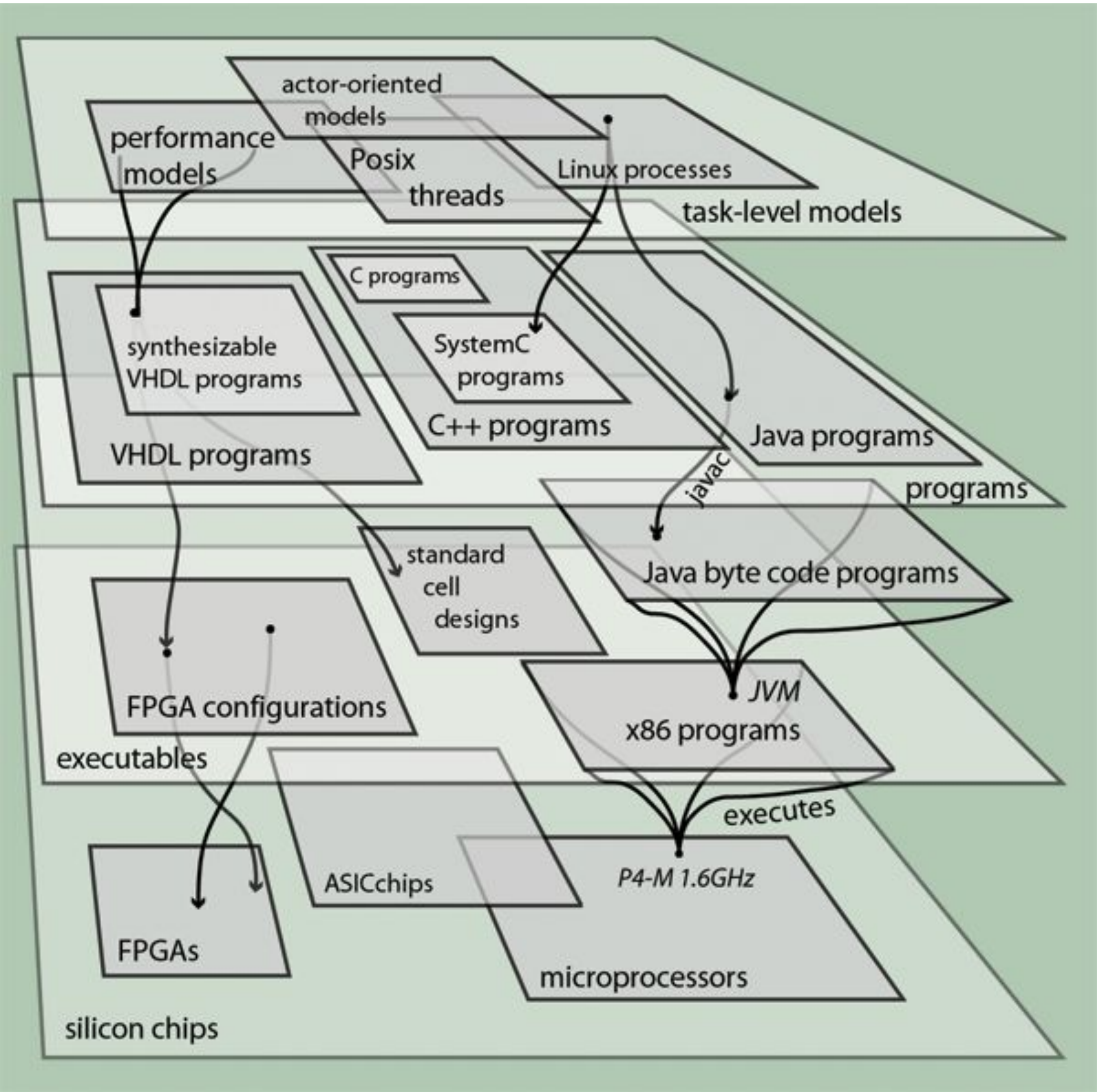
The Boeing 777 Problem

The Boeing 777 was Boeing's first fly-by-wire aircraft, controlled by software. It is deployed, appears to be reliable, and is succeeding in the marketplace. Therefore, it must be a success. However…

Boeing was forced to purchase and store an advance supply of the microprocessors that will run the software, sufficient to last for the estimated 50 year production run of the aircraft and another many years of maintenance.

Why?

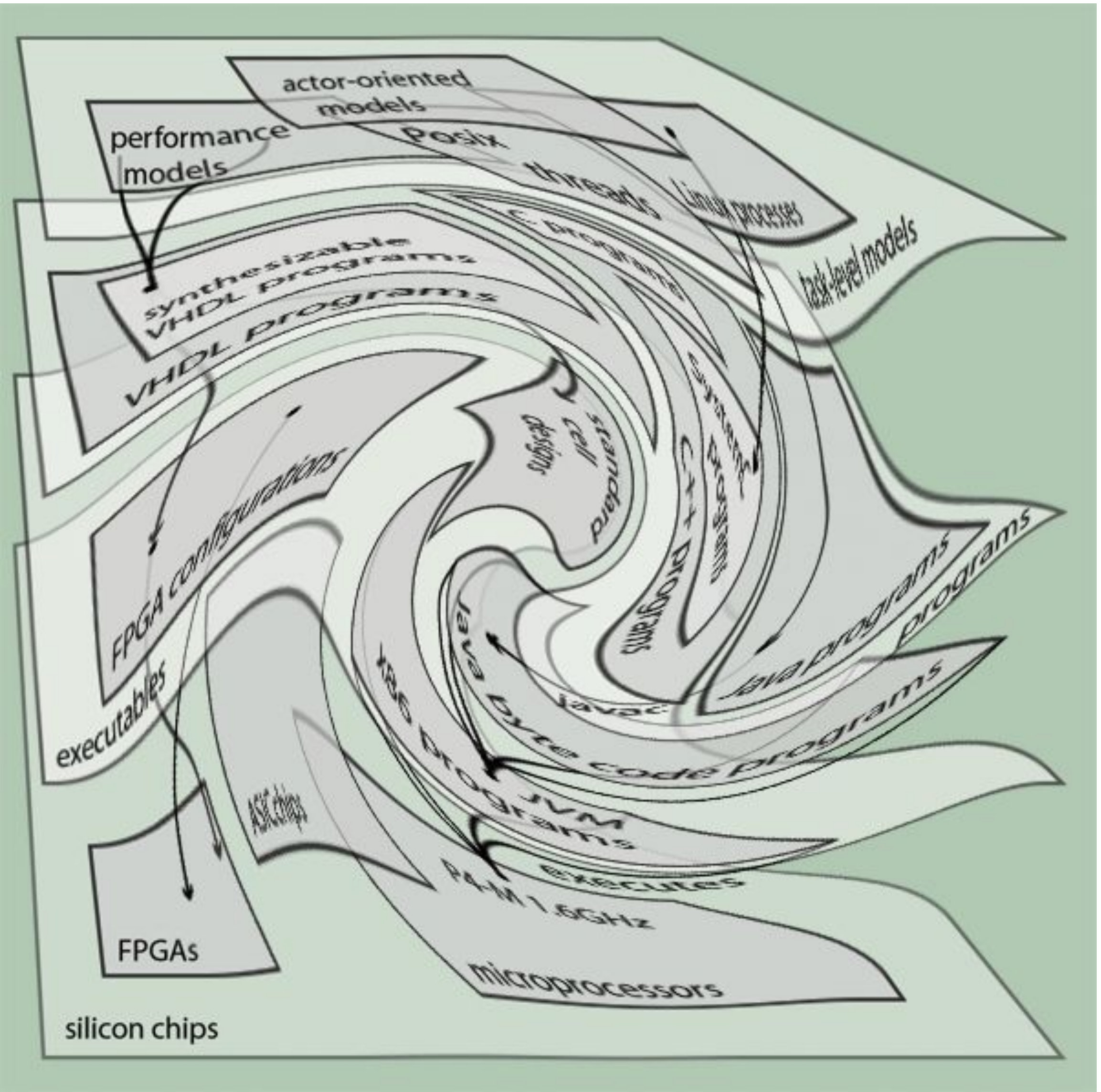
Abstraction Layers
All of which are models except the bottom



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.

Air France 447 – June 2009

“The Airbus A330, operated by Air France from Brazil to Paris, entered an aerodynamic stall from which it did not recover and crashed into the Atlantic Ocean at 02:14 UTC, killing all 228 passengers and crew.”

“The aircraft crashed after temporary inconsistencies between the airspeed measurements – likely due to the aircraft's pitot tubes being obstructed by ice crystals – caused the autopilot to disconnect, after which the crew reacted incorrectly and ultimately led the aircraft to an aerodynamic stall from which they did not recover.” [Source: Wikipedia]

What are potential causes?

Pitot tube for Airbus A380, together with an angle-of-attack vane (left). Air-flow is right to left.

787 Power System Issue
[Identified by FAA, May 2015]

"A Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode."

This condition is caused by the state of a software counter internal to the GCUs reached after 248 days of continuous power and could lead to loss of all AC electrical power, which could result in loss of control of the airplane.

What might the software bug be? How to catch it?

248 days == 2^31 100ths of a second.

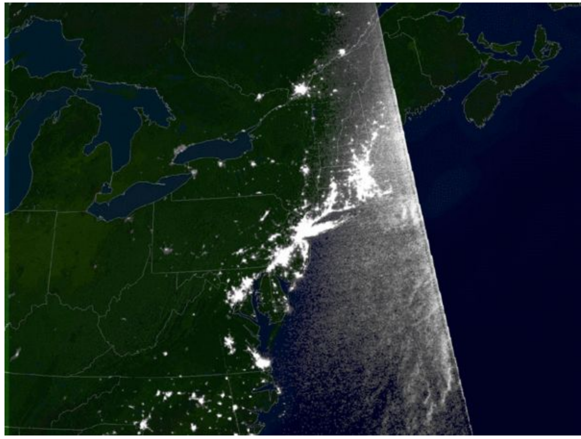
GM Cadillac bug (2004)

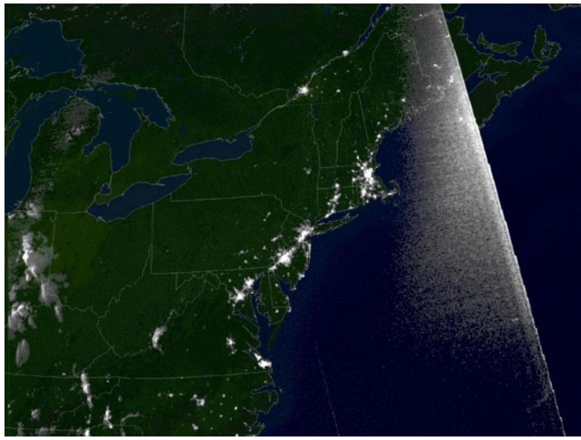
“General Motors Corp will recall 12,329 Cadillac SRXs equipped with all-wheel drive, following two reports of a software anomaly that causes a one-second delay in the anti-lock brakes activating to stop the vehicle -- reportedly only in the first few seconds of driving when the SUV is moving slowly. One owner crashed his SRX into his garage wall following the brake delay, but was uninjured.”

[Source: Reuters, 2 Apr 2004; obtained from RISKS digest]

What might the software bug be? How to catch it?

2003 Northeast Blackout (Power Grid failure)





"Worst outage in North American history."

"One [reason] was buried in a massive piece of software compiled from four million lines of C code and running on an energy management computer in Ohio… A silent failure of the alarm function in FirstEnergy's computerized Energy Management System (EMS) is listed in the final report as one of the direct causes of a blackout that eventually cut off electricity to 50 million people in eight states and Canada."
[The Register, UK, 8 Apr 2004.]

What could have gone wrong?
[Hint: multi-threaded software]

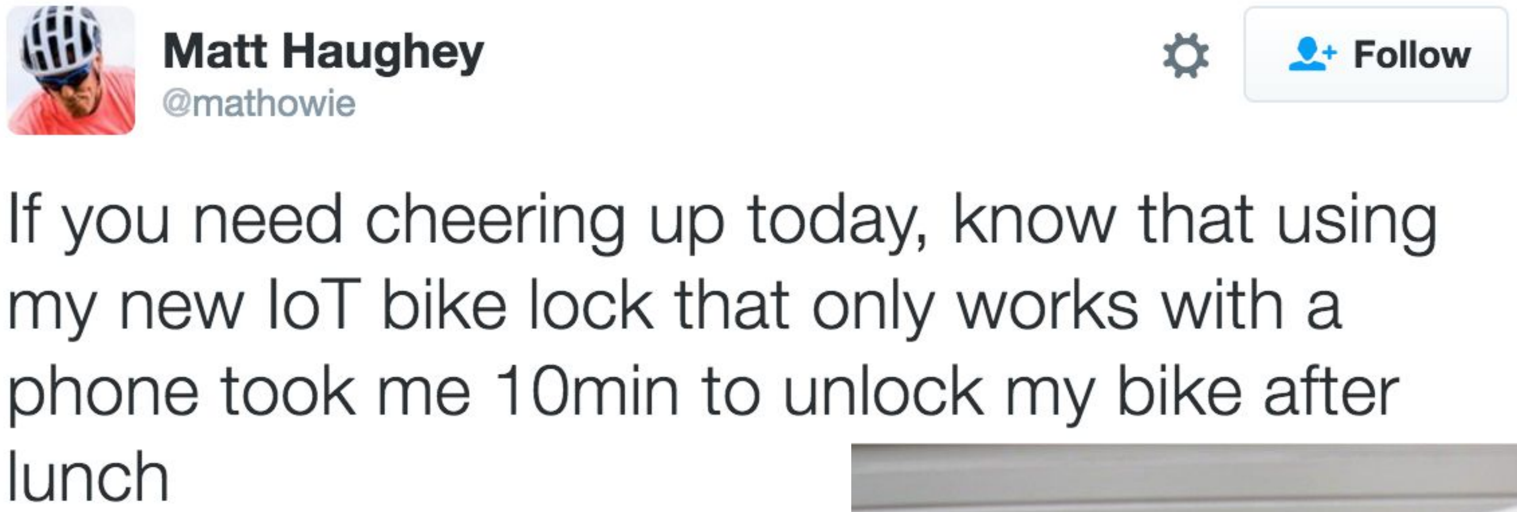
An overloaded transmission line sagged into unpruned foliage.

A race condition disabled the alarm system in a control room in Ohio.

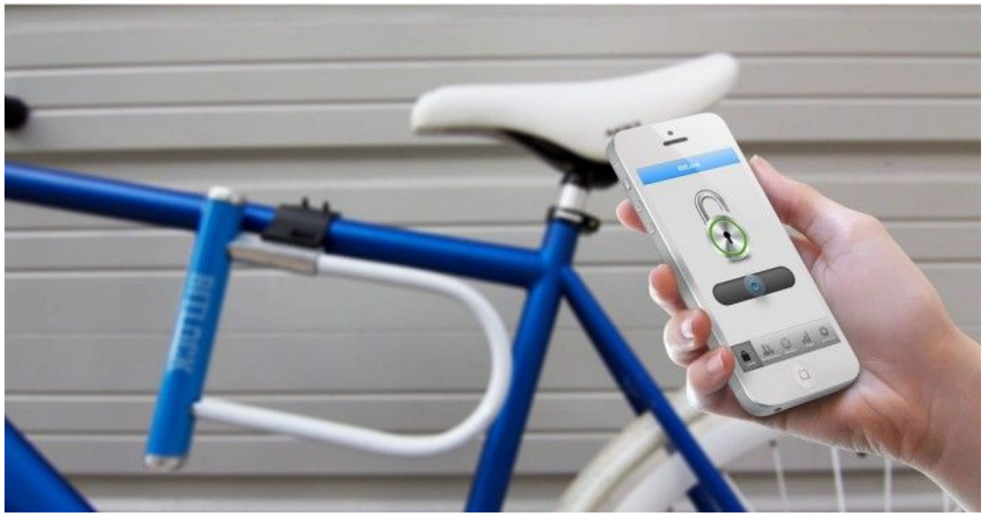
Cascading failures followed.

Even small systems can be a problem…

Usability?


Matt Haughey
@mathowie

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



thanks to Bjoern Hartmann

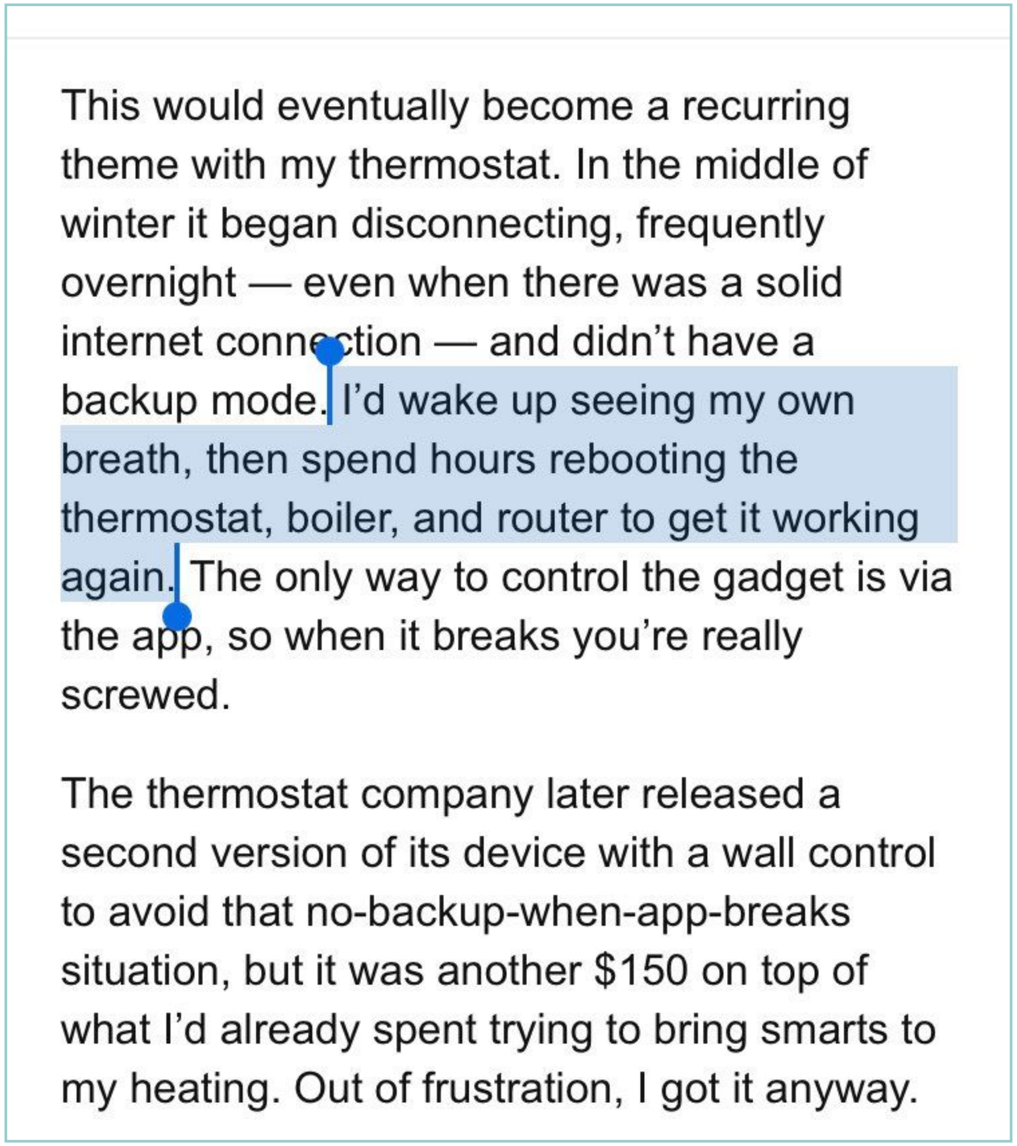
Lifespan?

Thanks to Bjoern Hartmann

Segal Lock.
Lifespan: ~100 years

August Bluetooth Lock.
Lifespan:?

Dependability?

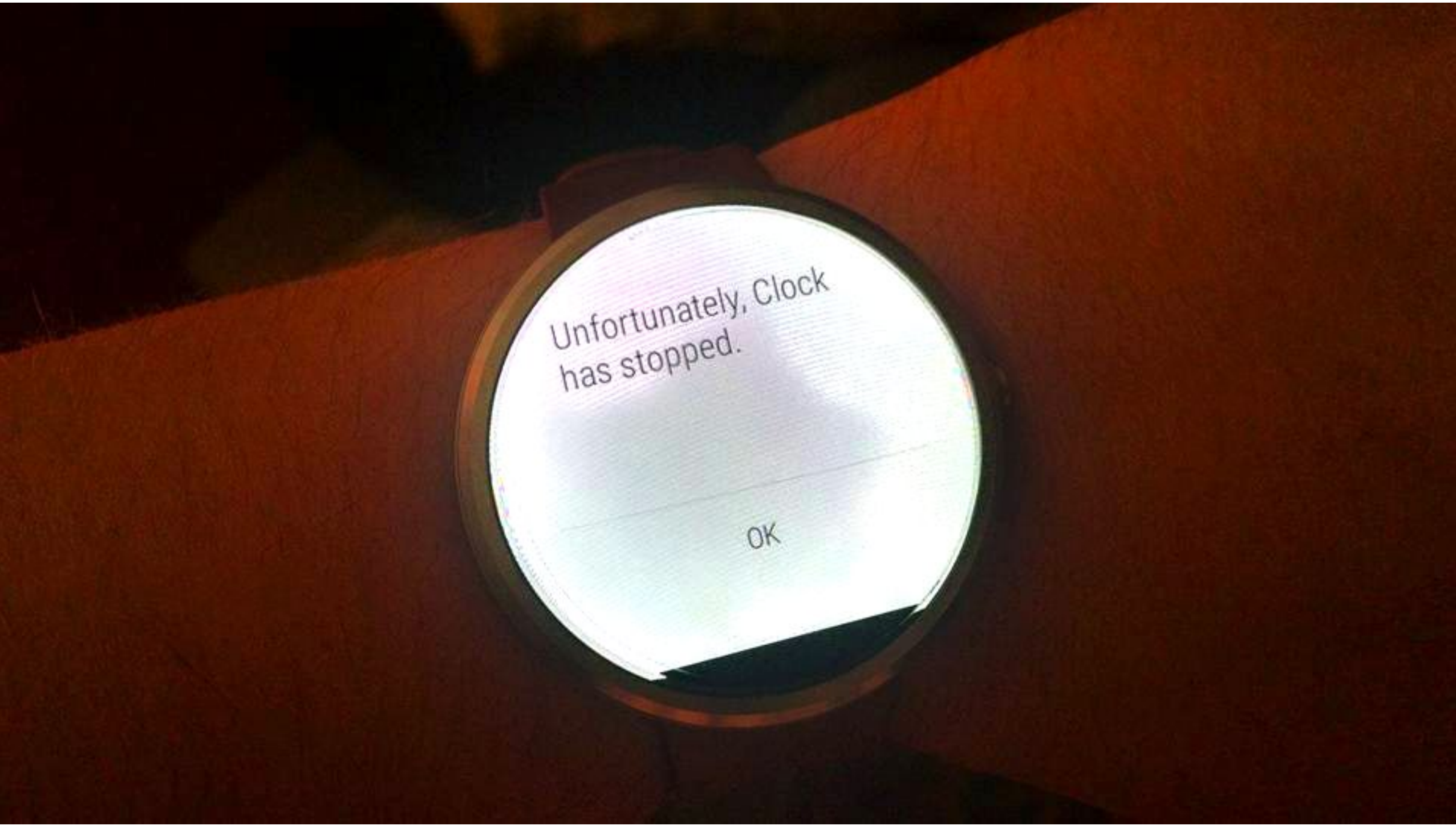


This would eventually become a recurring theme with my thermostat. In the middle of winter it began disconnecting, frequently overnight — even when there was a solid internet connection — and didn't have a backup mode. I'd wake up seeing my own breath, then spend hours rebooting the thermostat, boiler, 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.

Dependability?

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



Some Characteristics of Cyber-Physical Systems

Reactive

- operates at the speed of the environment

Real-time

- timing of events matters!

Concurrent

- system + environment, at a minimum

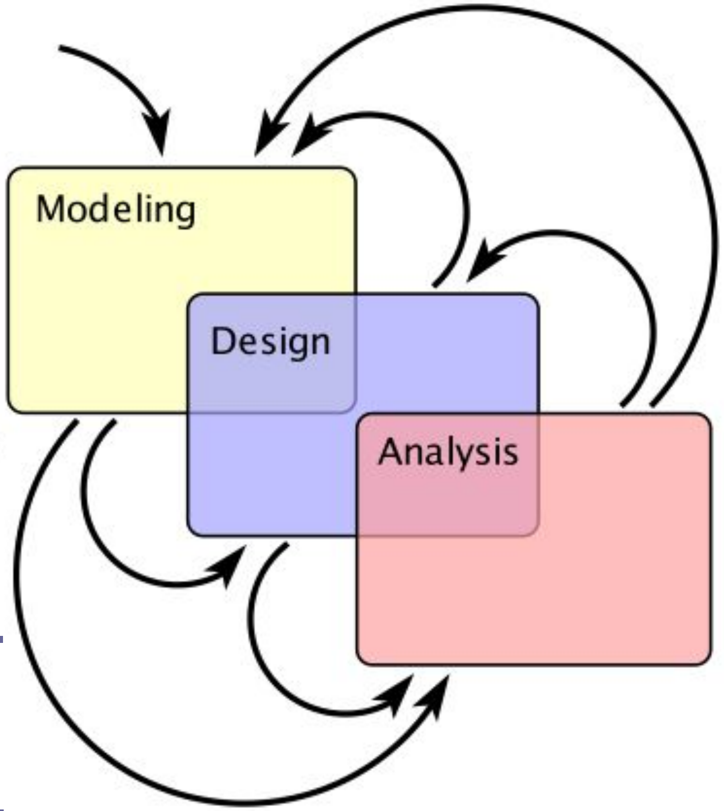
Heterogeneous

- hardware/software/networks, physical processes

(increasingly) Networked

- distributed, exposed to attacks

Modeling, Design, Analysis



Modeling is the process of gaining a deeper understanding of a system through imitation. Models specify **what** a system does.

Design is the structured creation of artifacts. It specifies **how** a system does what it does. This includes optimization.

Analysis is the process of gaining a deeper understanding of a system through dissection. It specifies **why** a system does what it does (or fails to do what a model says it should do).

Determinacy

Some of the most valuable models are *deterministic*.

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

Deterministic models have proven extremely valuable in the past. It simplifies design and enables analysis.

Laplace’s Demon

“We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes.”
— *Pierre Simon Laplace*

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

Did quantum mechanics 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)

Nevertheless, Laplace’s Demon cannot exist.

In 2008, David Wolpert, then at NASA, now at the Santa Fe Research Institute, used Cantor’s diagonalization technique to prove 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

The Koptez Principle

Hermann Kopetz
Professor (Emeritus)
TU Vienna

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

If we accept this, then it makes no sense to talk about whether the physical world is deterministic. It only makes sense to talk about whether *models* of the physical world are deterministic.

The question switches from whether a model is *True* to whether it is *Useful*

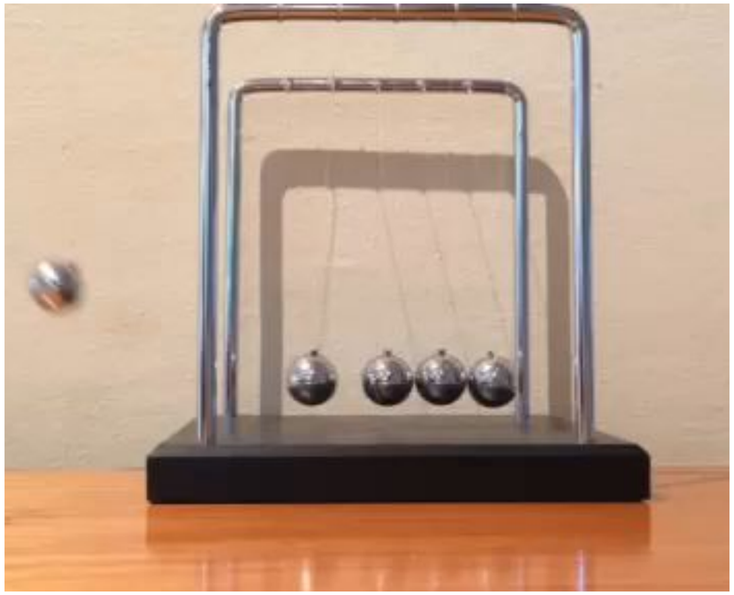
“Essentially, all models are wrong,
but some are useful.”

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

Physicists continue to debate whether the world is deterministic

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

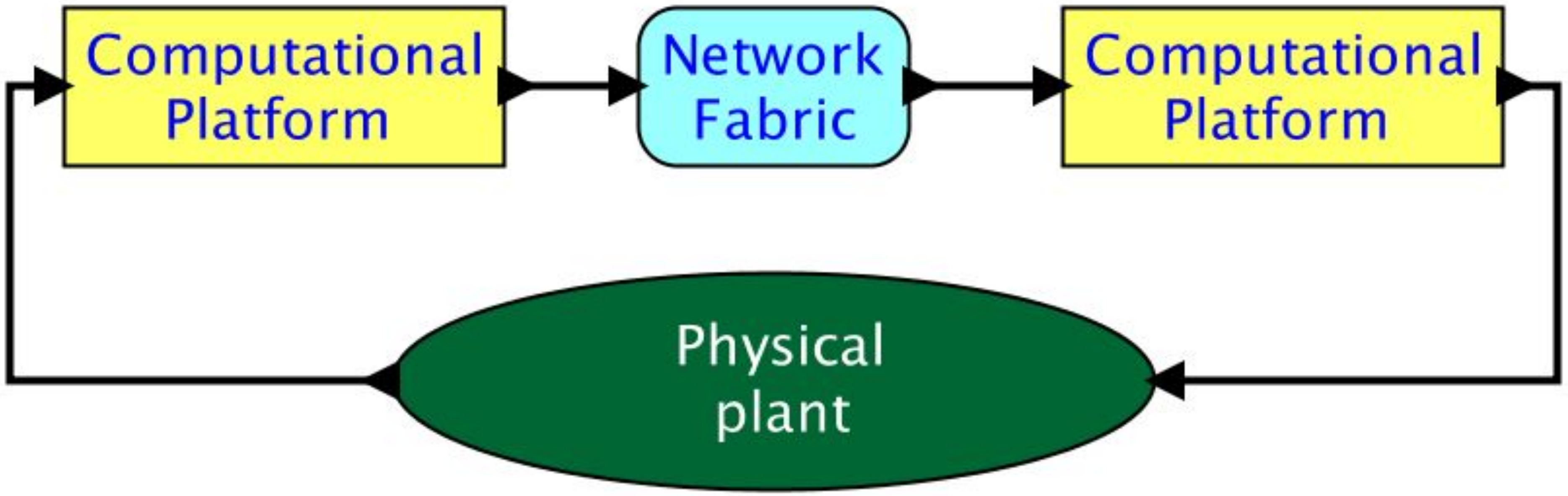
Deterministic model



Deterministic system?

Determinism is a property of models, not a property of the systems they model.

Schematic of a simple Cyber-Physical System



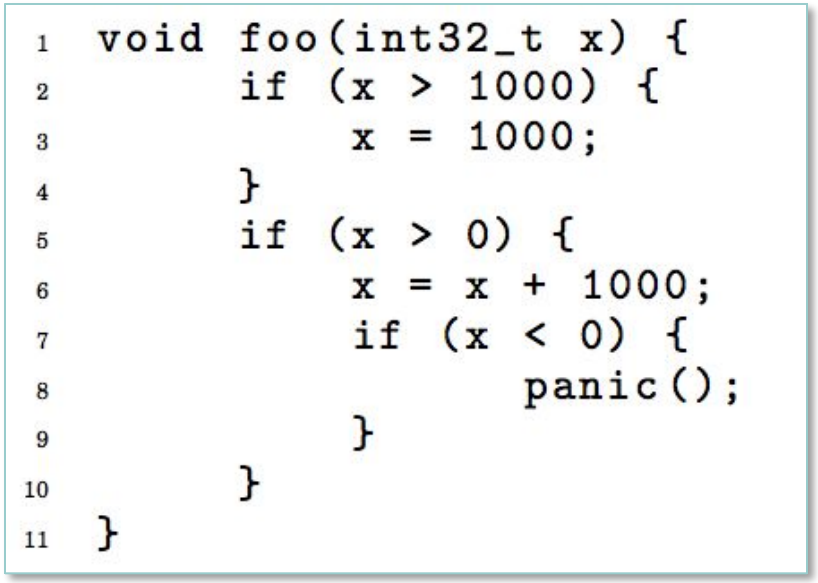
What kinds of models should we use?

Let's look at the most successful kinds of models from the cyber and the physical worlds.

Software is a Model

Physical System

Model



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

Single-threaded imperative programs are deterministic models

Software relies on another deterministic model that abstracts the hardware

Physical System

Model

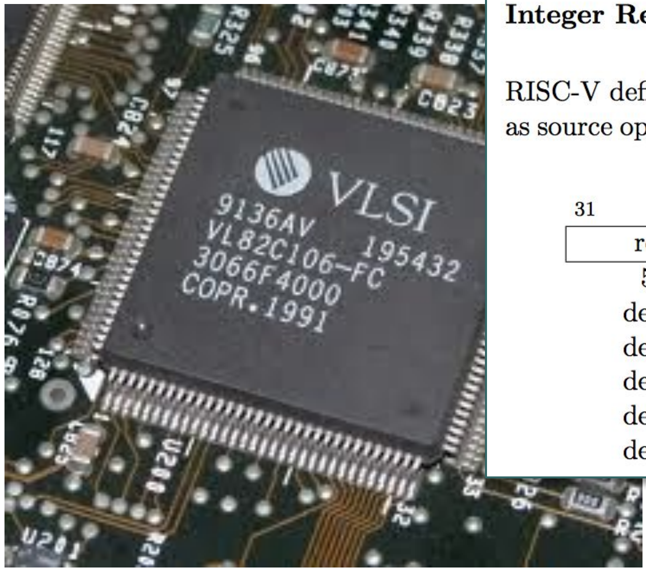
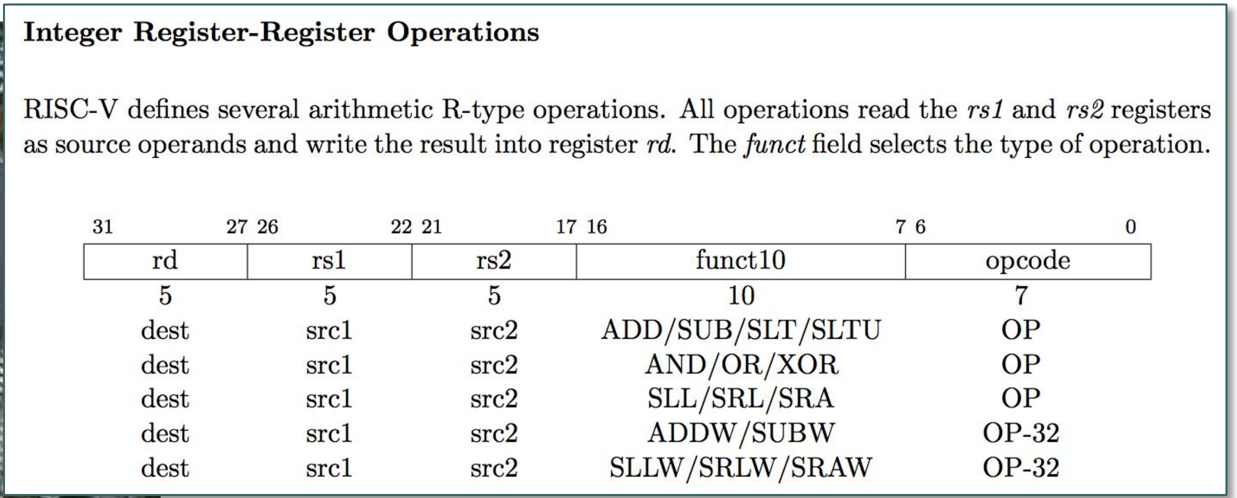


Image: Wikimedia Commons



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

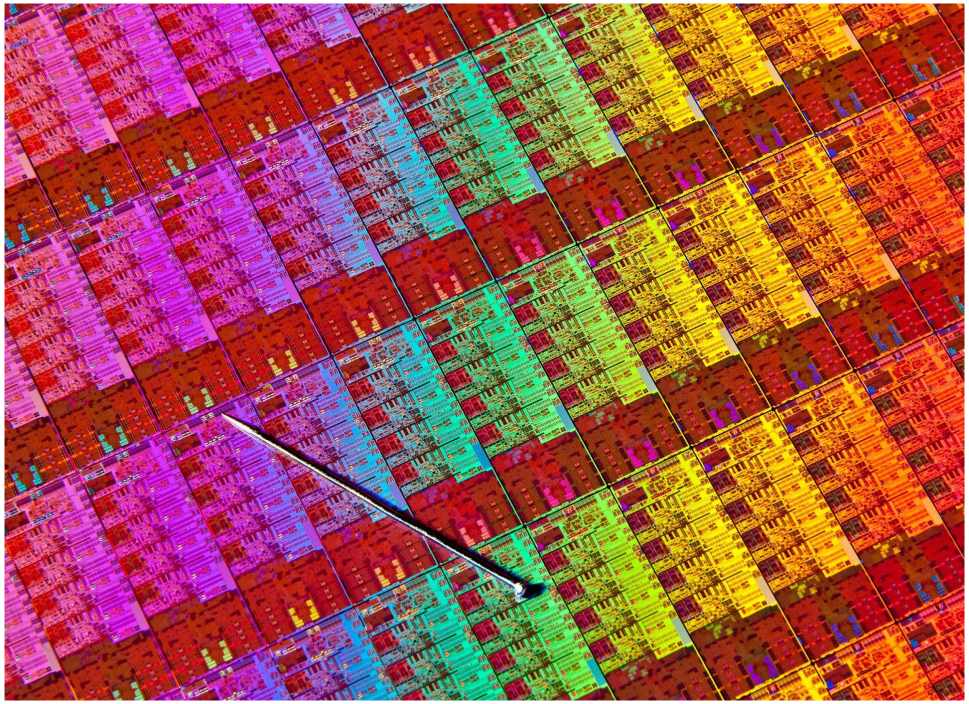
Waterman, et al., The RISC-V Instruction Set Manual, UCB/EECS-2011-62, 2011

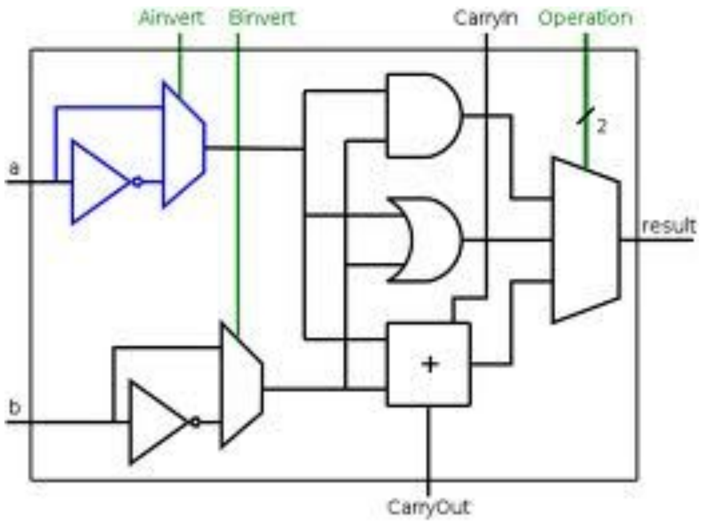
Instruction Set Architectures (ISAs) are deterministic models.

… which relies on yet another deterministic model

Physical System

Model





Synchronous digital logic is a deterministic model.

Deterministic Models for the Physical Side of CPS

Physical System

Model

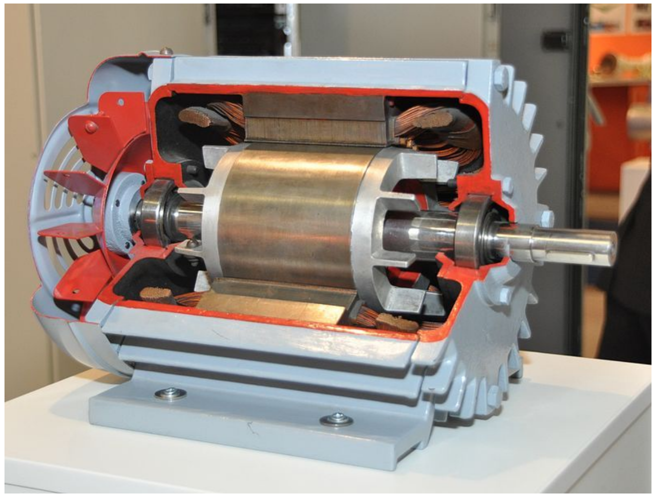
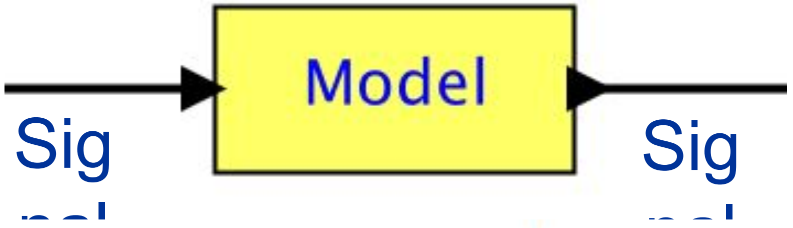


Image: Wikimedia Commons



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

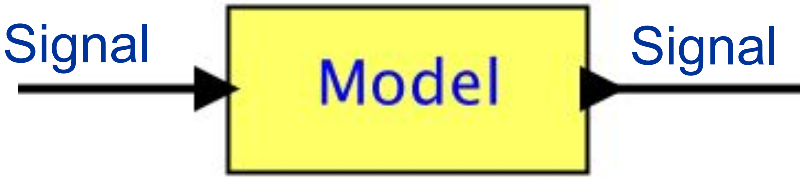
Differential Equations
are deterministic models.

A major problem for 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
                       );

                        != 0) {
                       un for 2 seconds
    }
    ... // other code
}
```

Not Dependable!



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

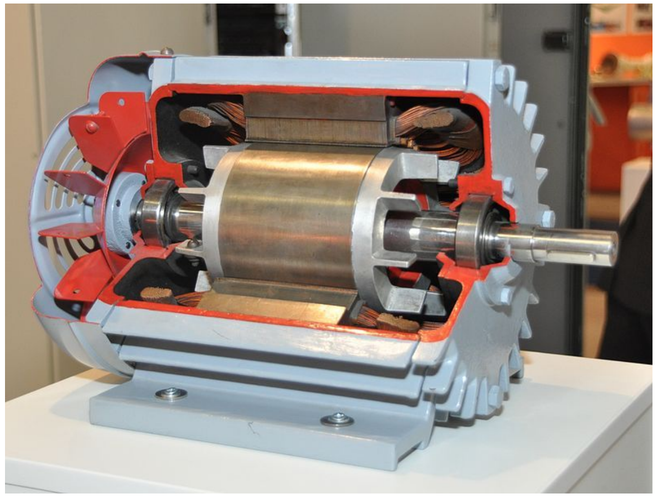


Image: Wikimedia Commons

Timing is not part of software and network semantics

***Correct execution** of a program in all widely used programming languages, and **correct delivery** of a network message in all general-purpose networks has nothing to do with how long it takes to do anything.*

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

CPS designers have no map!

The Value of Models

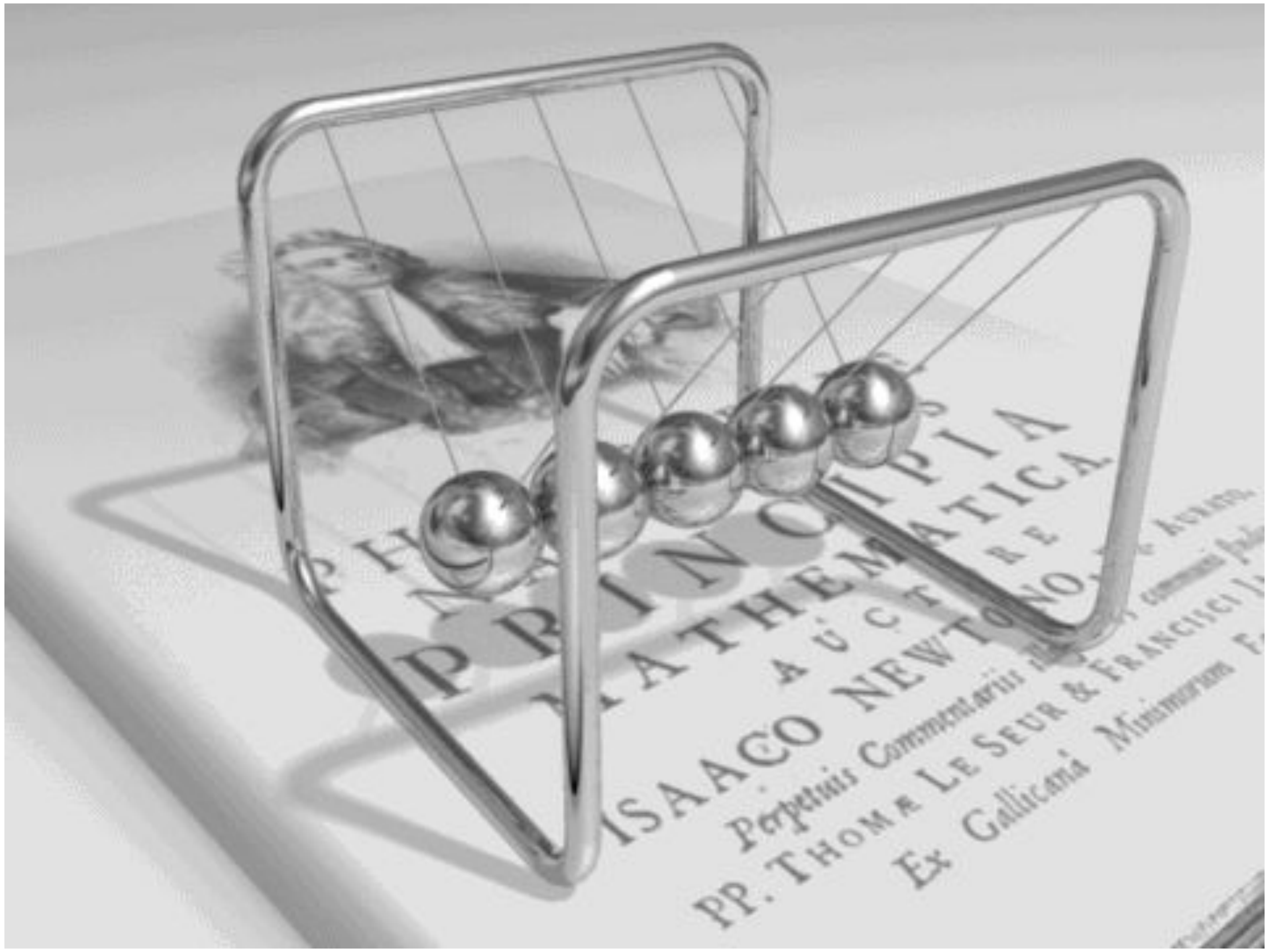
In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.

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

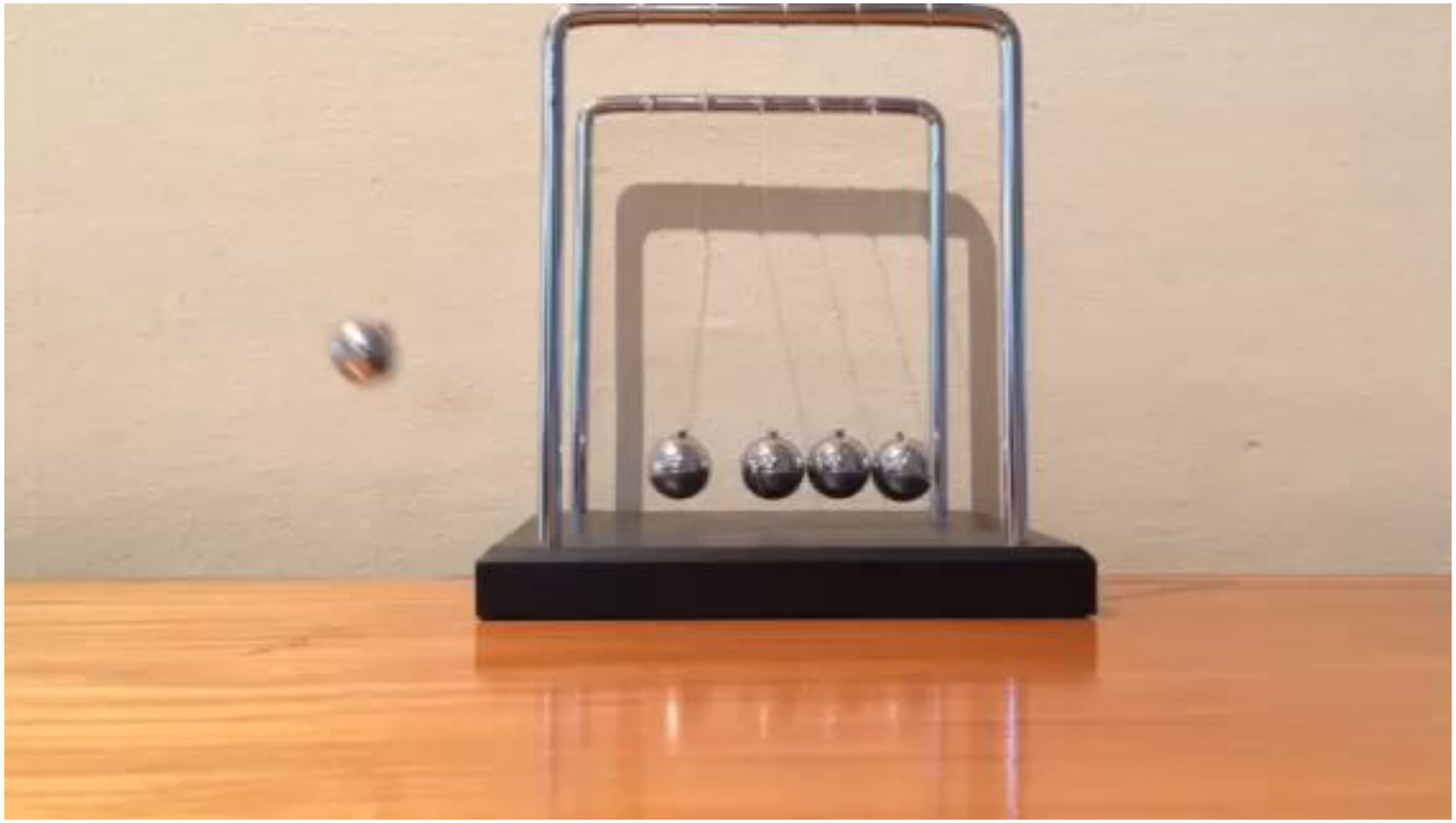
In engineering, model fidelity is a two-way street!

For a model to be useful, it is necessary (but not sufficient) to be able to be able to construct a faithful physical realization.

A Model


PRINCIPIA
MATHEMATICA
AUCTORE
ISAACO NEWTONO
Perpetuis Commentariis
PP. THOMÆ LE SEUR & FRANCISCI
Ex Gallicanâ Minimorum

A Physical Realization



Model Fidelity

To a *scientist*, the model is flawed.

To an *engineer*, the realization is flawed.

I’m an engineer…

For CPS, we need to change the question

The question is *not* whether deterministic models can describe the behavior of cyber-physical systems (with high fidelity).

The question is whether we can build cyber-physical systems whose behavior matches that of a deterministic model (with high probability).

Existence proofs that useful deterministic models for CPS exist

Deterministic models for CPS with faithful implementations exist:

PTIDES: distributed real-time software

- http://chess.eecs.berkeley.edu/ptides

PRET: time-deterministic architectures

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

These two projects ended in 2015.

Together, these technologies give a programming model for distributed and concurrent real-time systems that is deterministic in the sense of single-threaded imperative programs, and also deterministic w.r.t. to timing of external interactions.

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!

Conclusion

Think critically!

Plato and the Nerd
On Technology and Creativity

Edward Ashford Lee
MIT Press, 2017

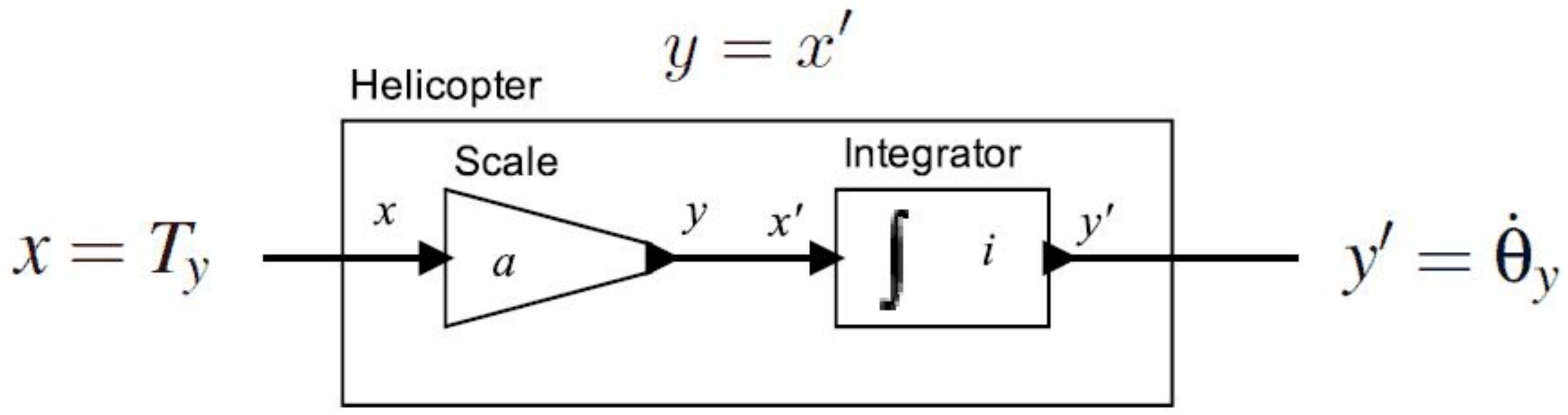
Forthcoming book
My first for a general audience

Model-Based Design

1. Create a **mathematical model** of all the parts of the cyber-physical system
   - Physical processes
   - Controllers: software, hardware, etc.
   - Software environment
   - Hardware platform
   - Network
   - Sensors and actuators
2. Construct the implementation from the model

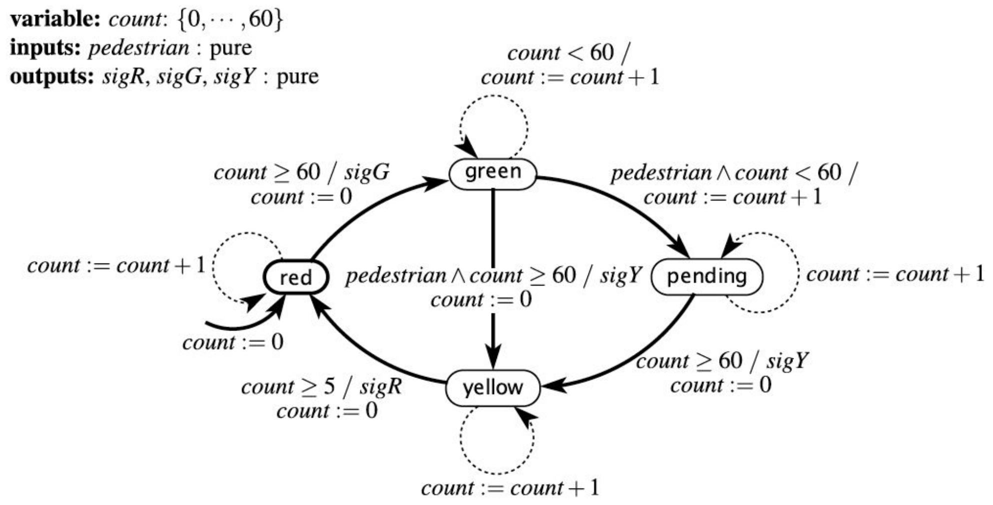
Modeling Techniques covered in the course (1)

- Differential Equations Physical processes
- Actor Models
- Time-domain modeling
- Feedback control



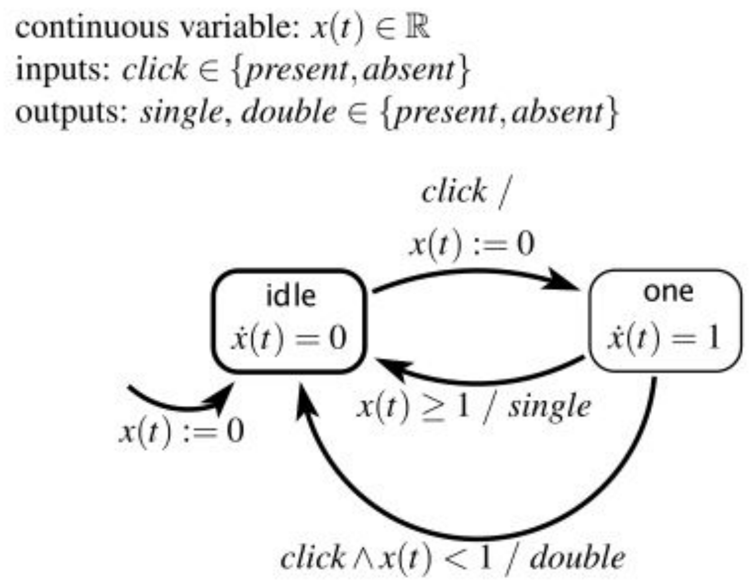
Modeling Techniques covered in the course (2)

- Finite-State Machines □ for Modal Behavior, as in a controller, software
- Determinism, Receptiveness
- Trace – modeling the input/output behavior of an FSM
- Composition and Hierarchy
  - Synchronous/Asynchronous composition, StateCharts


variable: $count$: $\{0, \cdots, 60\}$
inputs: $pedestrian$: pure
outputs: $sigR, sigG, sigY$: pure
$count < 60$ / $count := count + 1$
green
$count \geq 60$ / $sigG$ $count := 0$
$pedestrian \wedge count < 60$ / $count := count + 1$
$count := count + 1$
red
$pedestrian \wedge count \geq 60$ / $sigY$ $count := 0$
pending
$count := count + 1$
$count := 0$
$count \geq 60$ / $sigY$ $count := 0$
$count \geq 5$ / $sigR$ $count := 0$
yellow
$count := count + 1$

Modeling Techniques covered in the course (3)

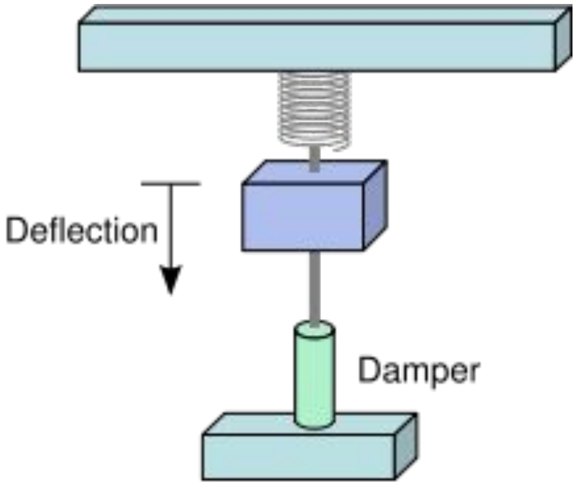
- Timed/Hybrid Automata □ for Modal Behavior + continuous dynamics
- Jumps and flows



continuous variable: $x(t) \in \mathbb{R}$
inputs: $click \in \{present, absent\}$
outputs: $single, double \in \{present, absent\}$

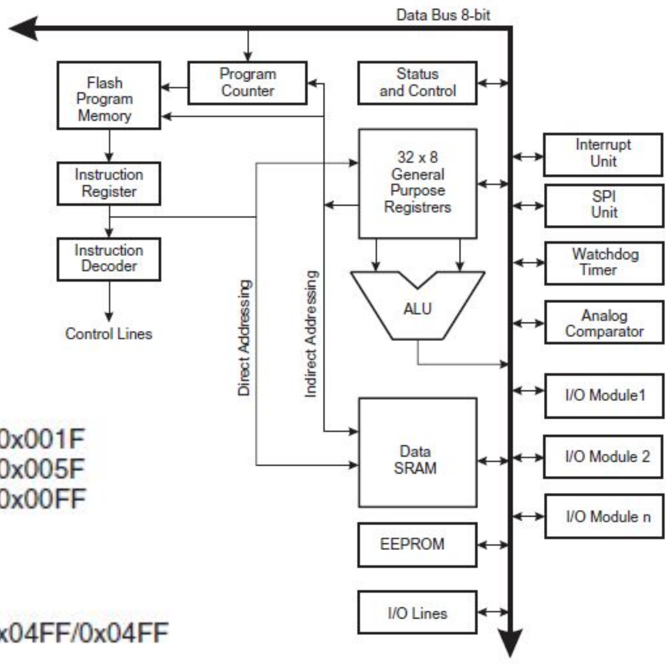
Modeling & Design: Sensors and Actuators

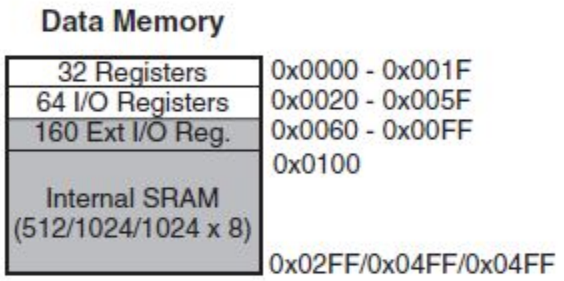
- How Sensors and Actuators Work: Basics
- Interfacing to Sensors
- Modeling Sensors and Actuators



Design: Memory Architectures

- Types of Memory
- Memory Maps and Organization
- Memory Model for C programs
- Memory Hierarchy and Protection




Data Memory

32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024 x 8)	0x0100 … 0x02FF/0x04FF/0x04FF

Source: ATmega168 Reference Manual

Design: Concurrent Programming with Interrupts

- I/O Mechanisms in Software: Polling vs. Interrupts
- Setting up Interrupts
- Reasoning about Interrupt-Driven Programs

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

Concurrency: Modeling and Design

- Threads
- Processes
- Multi-Tasking and Priorities
- Synchronous/Reactive Languages
- Dataflow

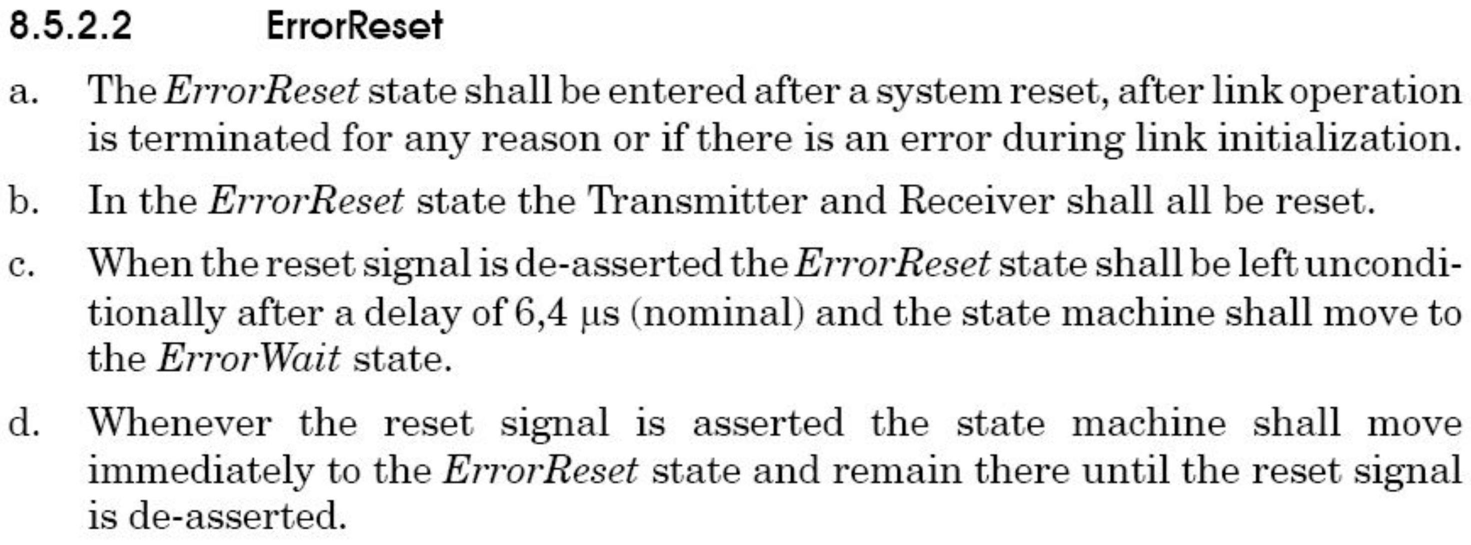
Real-Time: Design and Analysis

- Scheduling
  - Pre-emptive and non-preemptive
  - RMS vs EDF
  - Priority inversion, protocols: PIP, PCP
  - Anomalies in multiprocessor scheduling

- Execution Time Analysis
  - Blending measurements, platform modeling, and static analysis of code

Modeling & Analysis: Specification & Temporal Logic

- The Need for Formal Specification
- Linear Temporal Logic



8.5.2.2 ErrorReset

a. The *ErrorReset* state shall be entered after a system reset, after link operation is terminated for any reason or if there is an error during link initialization.

b. In the *ErrorReset* state the Transmitter and Receiver shall all be reset.

c. When the reset signal is de-asserted the *ErrorReset* state shall be left unconditionally after a delay of 6,4 μs (nominal) and the state machine shall move to the *ErrorWait* state.

d. Whenever the reset signal is asserted the state machine shall move immediately to the *ErrorReset* state and remain there until the reset signal is de-asserted.

Analysis and Verification

- Reachability Analysis
  - Compute the set of all states of the system reachable from any initial state
- Model Checking
  - Does the (closed-loop) system satisfy a temporal logic property?
- Equivalence and Refinement
  - When are two state machines equivalent?
  - When does one model refine another?

Fault Tolerance and Security

- Tolerating faults in sensors, computation, actuators
  - Self-checking, N-modular redundancy, interval readings for sensors, etc.

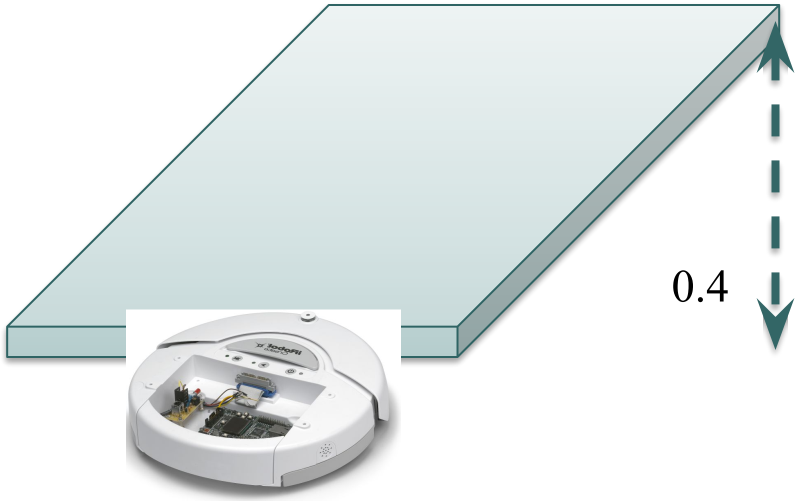
- Security & Privacy
  - Integrity, Confidentiality, Availability under attacks
  - Besides traditional issues, need to worry about physical properties and constraints (e.g. power)
  - Privacy properties and enforcement

Distributed Systems and Networking

- Proprietary protocols: CAN and FlexRay
- Clock synchronization (IEEE 1588)
- Wireless protocols: BLE, ZigBee, OpenWSN, …
- Time-Triggered Ethernet, …

The Lab

- CPS Programming in C (low-level language)
- CPS Programming in LabVIEW (modeling language)
- Modeling Physical Processes and Interfacing to Sensors and Actuators
- Specification & Temporal Logic



+

$$\mathbf{F}_{[0,40]}\ z \geq 0.4$$

Other Relevant Topics we didn’t cover in-depth

- Architecture for embedded systems
  - E.g. low power, predictable timing, etc.
- Programming languages and compilers
- Testing and debugging
- Controller synthesis
- Simulation strategies
- Hybrid systems (more than timed automata)
- …

Future of CPS Design

Rising trend: combine model-based design with data-driven methods (learning from data)

This course discussed how design is done today, but you can be sure that *the technology will change*!

Our goal has been to give you what you need to think *critically* about the technology.

Agradecemos ao Professor Edward Lee da Universidade de Berkeley por oferecer os slides que usamos como base neste curso