Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

Unit 1: Cyber Physical Systems (Ch. 1)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

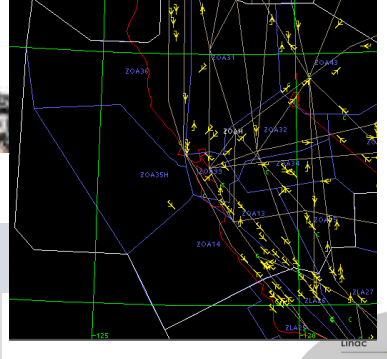
Recep: What are Cyber-Physical Systems

- Computational systems
  - but not general-purpose computers
- Integral with physical processes
  - sensors, actuators, physical dynamics
- Reactive
  - at the speed of the environment (timing matters!)
- Heterogeneous
  - hardware/software/networks, mixed architectures
- Networked
  - concurrent, distributed, dynamic

Cyber-Physical Systems (CPS):

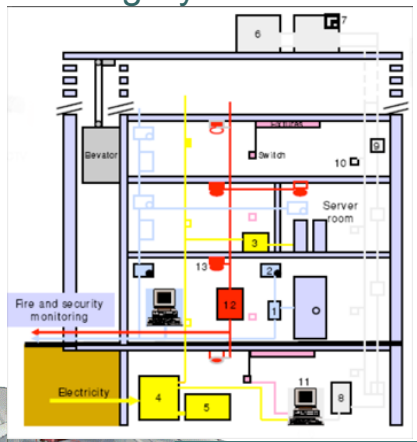
Orchestrating networked computational resources with physical systems

Avionics



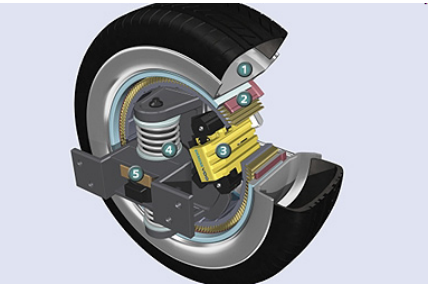
Transportation (Air traffic control at SFO)

Building Systems

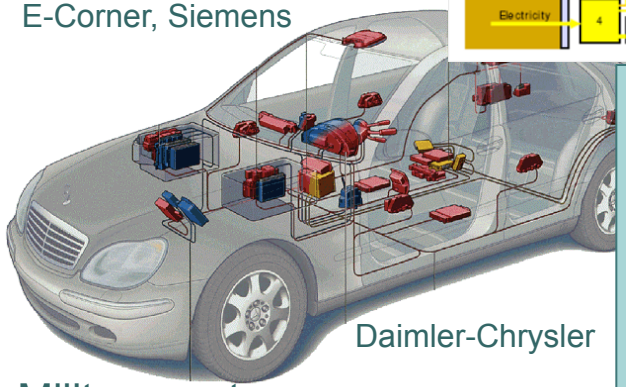


Telecommunications

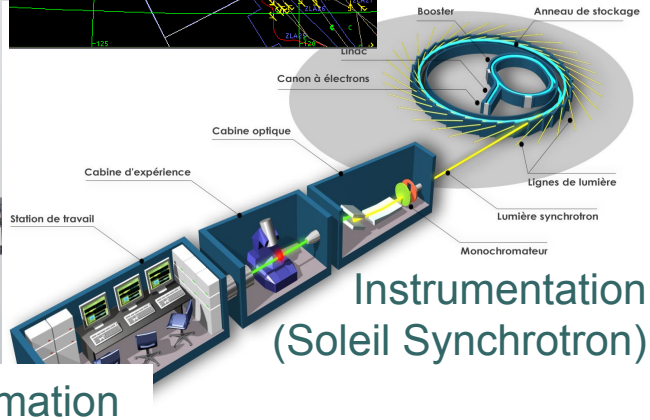
Automotive



E-Corner, Siemens

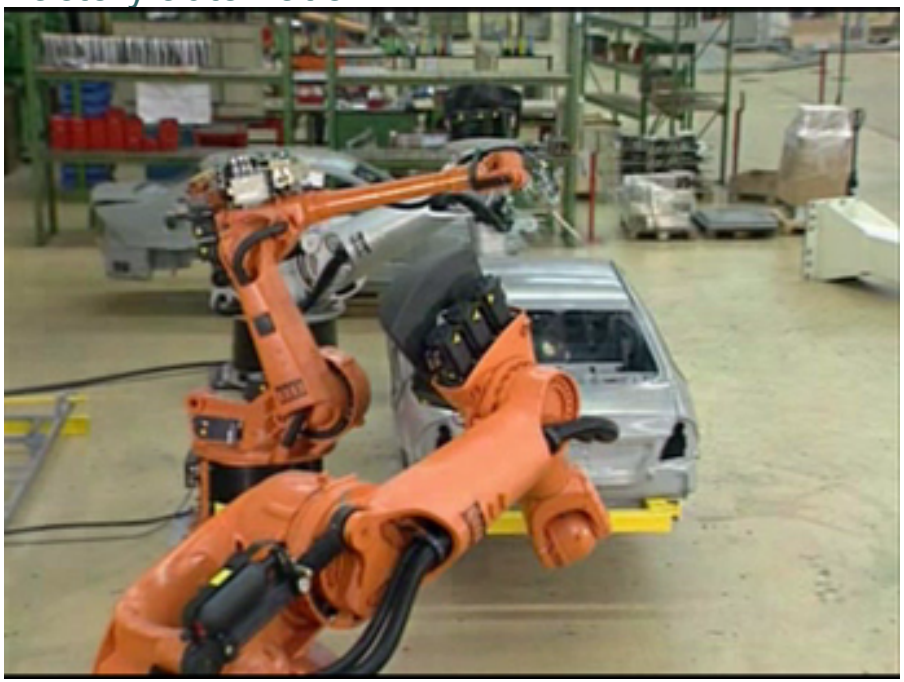


Daimler-Chrysler



Instrumentation (Soleil Synchrotron)

Factory automation

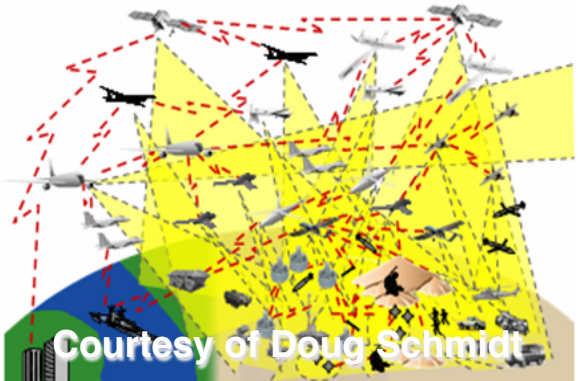


Courtesy of Kuka Robotics Corp.

Power generation and distribution

Courtesy of General Electric

Military systems:



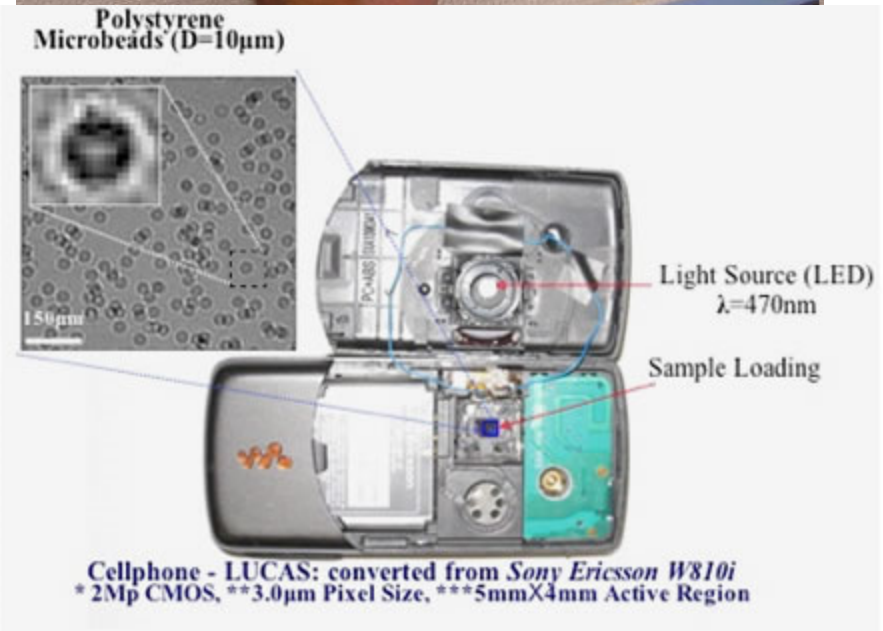
Courtesy of Doug Schmidt

CPS Example: Medical Devices

Emerging direction: Smartphone based medical devices for affordable healthcare

For example, “Telemicroscopy” project at Berkeley

For example, Cell-phone based blood testing device developed at UCLA



CPS Example: Printing Press

Bosch-Rexroth

- High-speed, high precision
  - Speed: 1 inch/ms
  - Precision: 0.01 inch

    -> Time accuracy: 10us
- Open standards (Ethernet)
  - Synchronous, Time-Triggered
  - IEEE 1588 time-sync protocol
- Application aspects
  - local (control)
  - distributed (coordination)
  - global (modes)

CPS Example: Automotive Electronics Today

About 80 computers (*electronic control units,* ECUs) in a premium car today:

- engine control, transmission, anti-lock brakes, electronic suspension, parking assistance, climate control, audio system, “body electronics” (seat belt, etc.), display and instrument panel, etc.
- linked together by CAN bus (today), FlexRay (tomorrow) with up to 2km of wiring.
- growing fraction of development costs, manufacturing costs, and fuel consumption.

Where CPS Differs from the Traditional Embedded System Problems

The traditional embedded systems problem:

Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources and extracting performance).

The CPS problem:

Computation and networking integrated with physical processes. The technical problem is managing dynamics, time, and concurrency in networked computational + physical systems.

A Key Challenge on the Cyber Side: Real-Time Software

Correct execution of a program in C, C#, Java, Haskell, etc. has nothing to do with how long it takes to do anything. All our computation and networking abstractions are built on this premise.

Timing of programs is not repeatable, except at very coarse granularity.

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

Techniques Exploiting the Fact that Time is Irrelevant

Programming languages
Virtual memory
Caches
Dynamic dispatch
Speculative execution
Power management (voltage scaling)
Memory management (garbage collection)
Just-in-time (JIT) compilation
Multitasking (threads and processes)
Component technologies (OO design)
Networking (TCP)
…

Course Focuses

Embedded Systems Courses

- Hardware interfacing
- Interrupts
- Memory systems
- C programming
- Assembly language
- FPGA design
- RTOS design
- …

Cyber-Physical Courses

- Modeling
- Timing
- Dynamics
- Imperative logic
- Concurrency
- Verification
- …

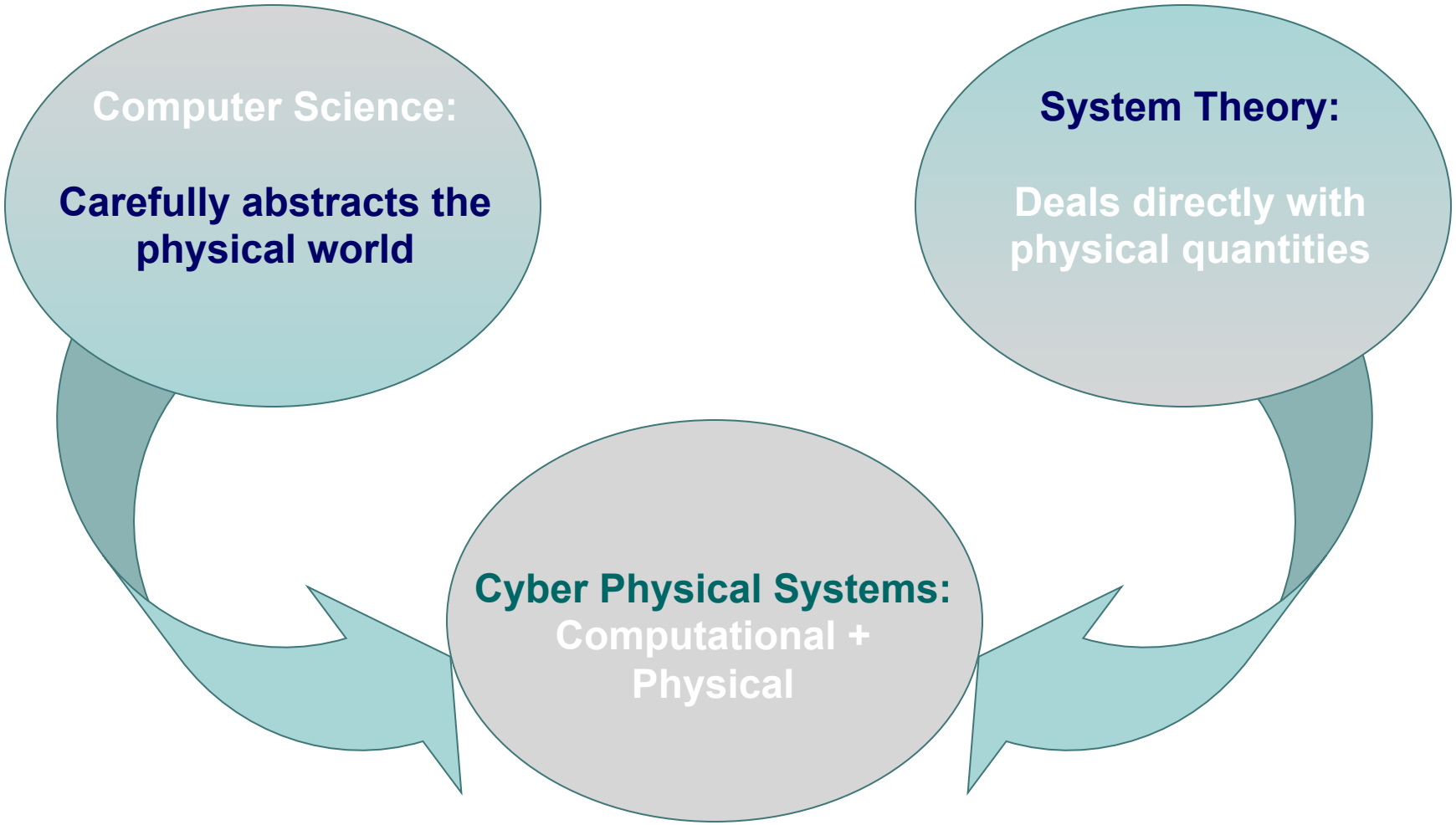
A Theme of Our Course: Model-Based Design

Models are abstractions of systems:

- structural (OO design)
- ontological (type systems)
- imperative logic ("procedural epistemology")
- functional logic
- actor-oriented (including dataflow models)

All of these have their place…

CPS is Multidisciplinary



First Challenge

Models for the physical world and for computation diverge.

- physical: time continuum, ODEs, dynamics
- computational: a “procedural epistemology,” logic

There is a huge cultural gap.

Physical system models must be viewed as semantic frameworks, and theories of computation must be viewed as alternative ways of talking about dynamics.

Second Challenge

We typically learn to *use* modeling techniques, not to *evaluate* modeling techniques.

- "this is how computers work"
- "this equation describes that feedback circuit"

rather than

- "this is how Von Neumann proposed that we control automatic machines"
- "ignoring the intrinsic randomness and latency in this circuit, Black proposed that we could idealize its behavior in this way"

We need to think about meta-modeling, not just modeling. They must learn to think critically about modeling methods, not just about models.

What this course is about

A principled, scientific approach to designing and implementing embedded systems

Not just hacking!!

Hacking can be fun, but it can also be very painful when things go wrong…

Focus on *model-based system design*, and on *embedded software*

Traditionally, embedded systems has been an industrial (not academic) problem, principally about resource limitations.

- small memory
- small data word sizes
- relatively slow clocks

When these are the key problems, emphasize efficiency:

- write software at a low level (in assembly code or C)
- avoid operating systems with a rich suite of services
- develop specialized computer architectures:
  - programmable DSPs
  - network processors
- develop specialized networks
  - Can, FlexRay, TTP/C, MOST, etc.

This is how embedded SW has been designed for 30 years

But embedded systems do have more fundamental differences from general-purpose computation:

time matters

- "as fast as possible" is not good enough

concurrency is intrinsic

- it's not an illusion (as in time sharing), and
- it's not (necessarily) about exploiting parallelism

processor requirements can be specialized

- predictable, repeatable timing
- support for common operations (e.g. FIR filters)
- need for specialized data types (fixed point, bit vectors)

programs need to run (essentially) forever

- memory usage has to be bounded (no leaks!!)
- rebooting is not acceptable

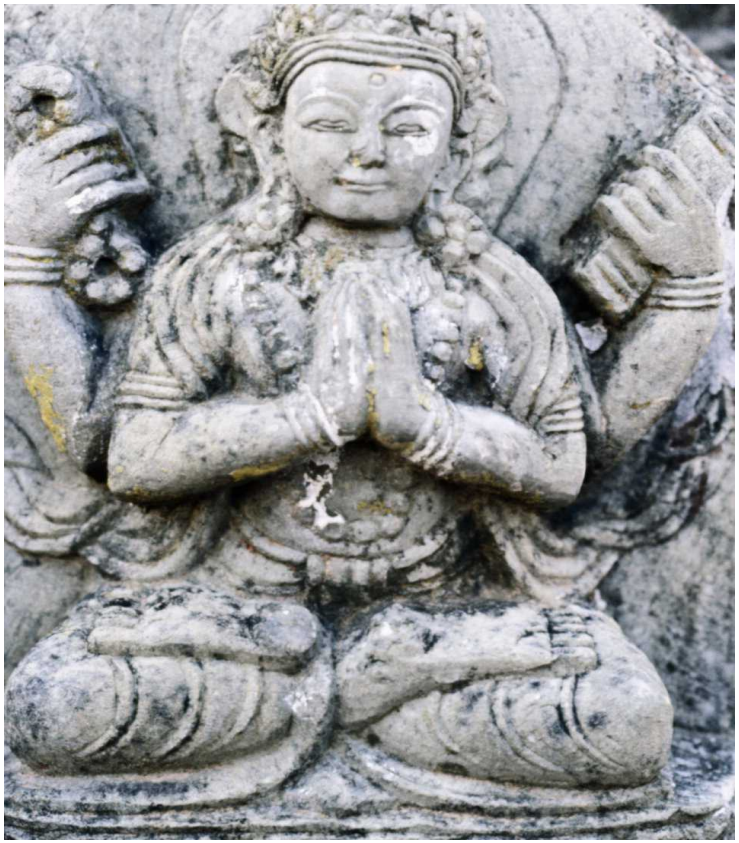
What about “real time”?

Make it faster!

What if you need “absolutely positively on time”?

Today, most embedded software engineers write code, build your system, and test for timing. Model-based design seeks to specify dynamic behavior (including timing) and “compile” implementations that meet the behavior.

Real-Time Multitasking?



Prioritize and Pray!

All too often, real-time operating systems (RTOSs) are used in a rather ad hoc way. Without any particular principles, engineers tweak priorities until the prototype works under test.

The resulting system is *brittle*, meaning the small changes in the operating conditions (or in the design of the system) can cause big changes in behavior. For example, replacing the processor with a faster one can cause real-time failures.

An engineer's responsibility

- Korean Air 747 in Guam, 200 deaths (1997)
- 30,000 deaths and 600,000 injuries from medical devices (1985-2005)
  - perhaps 8% due to software?

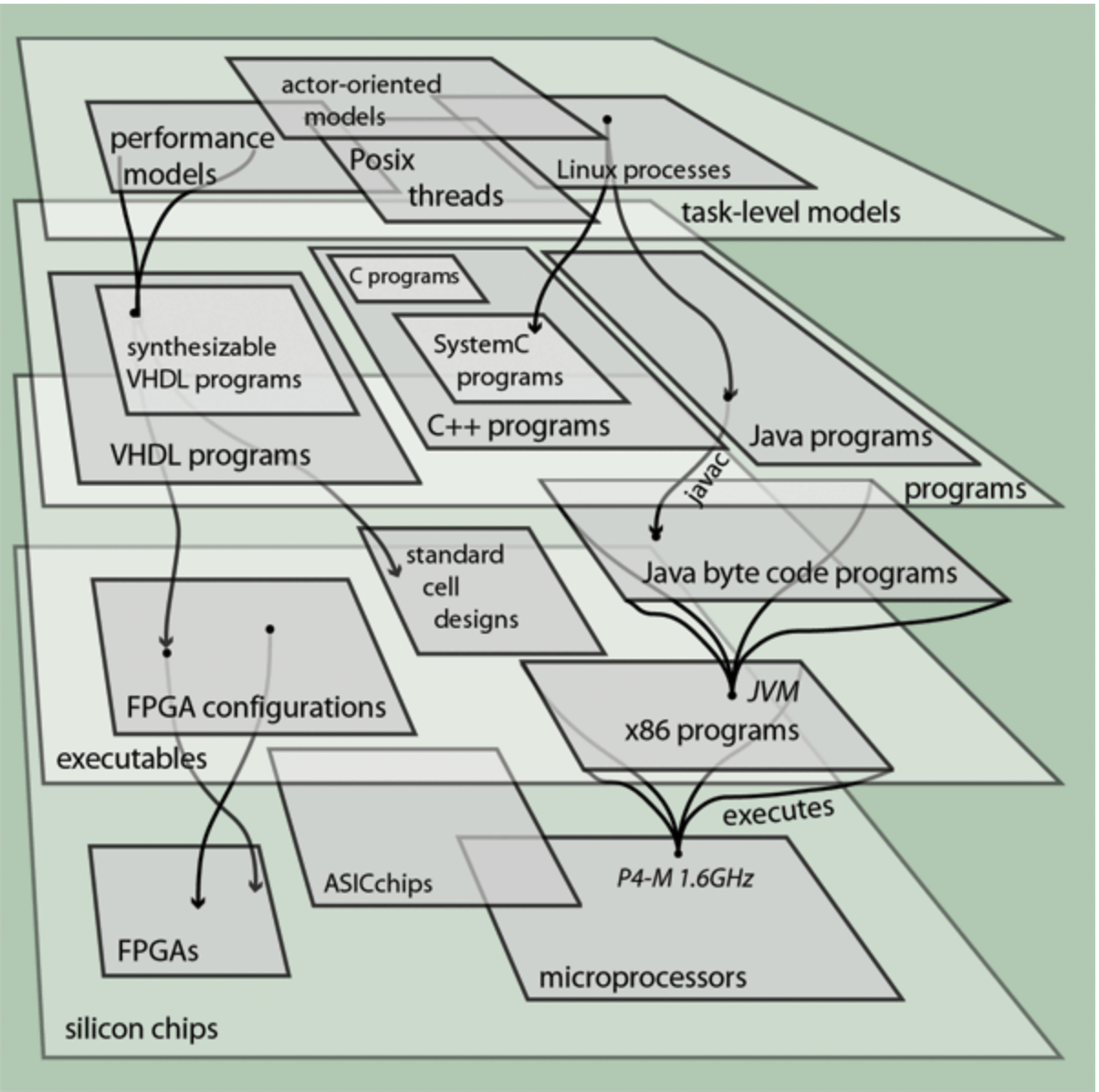
source: D. Jackson, M. Thomas, L. I. Millett, and the Committee on Certifiably Dependable Software Systems, "Software for Dependable Systems: Sufficient Evidence?," National Academies Press, May 9 2007.

A Real Story

A “fly by wire” aircraft, expected to be made for 50 years, requires a 50-year stockpile of the hardware components that execute the software.

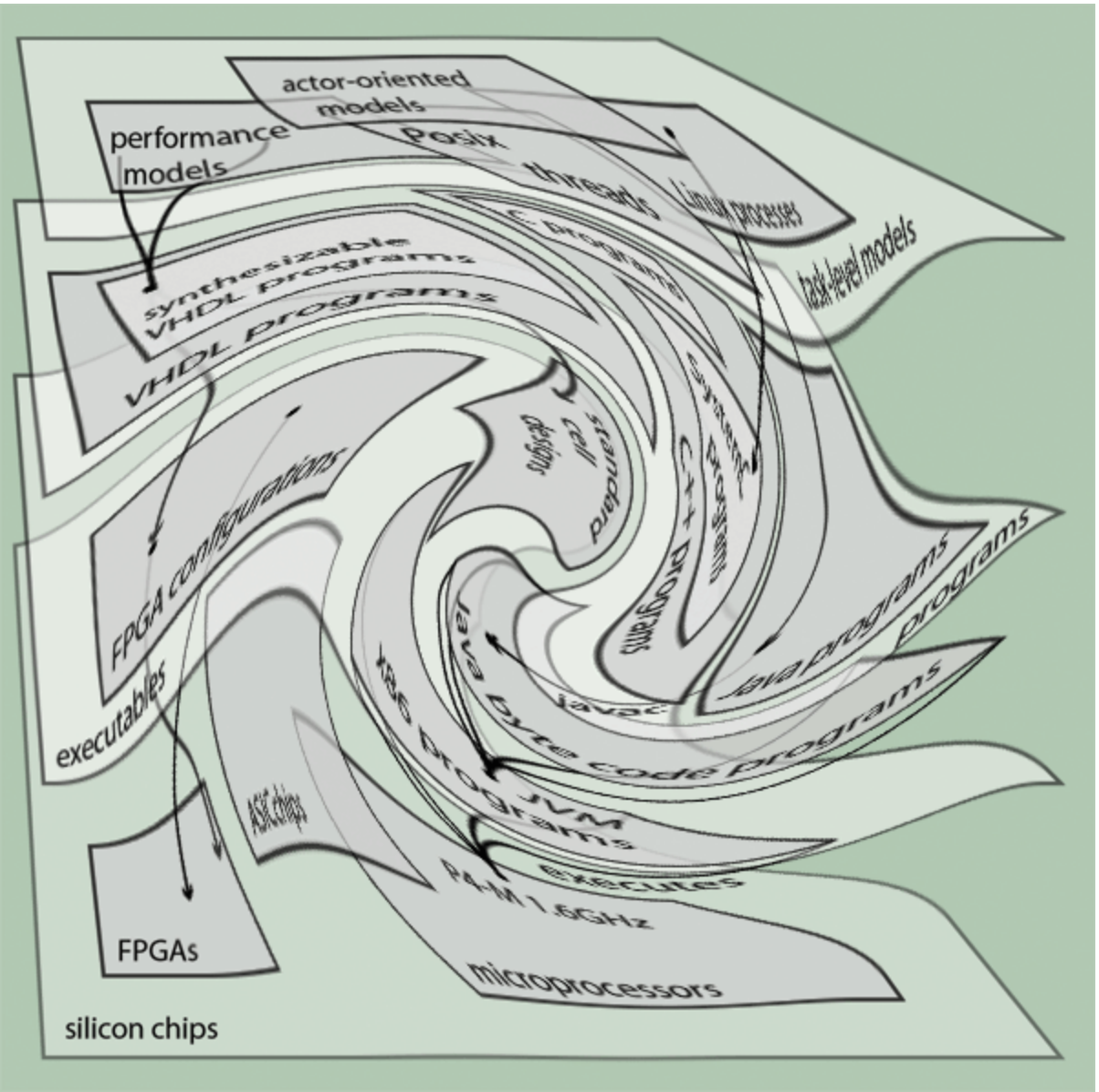
All must be made from the same mask set on the same production line. Even a slight change or “improvement” might affect timing and require the software to be re-certified.

Abstraction Layers



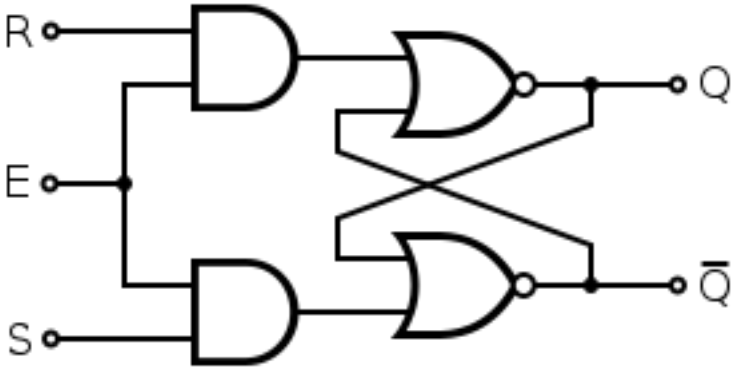
The purpose for an abstraction is to hide details of the implementation below and provide a platform for design from above.

Abstraction Layers



Every abstraction layer has failed for time-sensitive applications.

Is the problem intrinsic in the technology?



Electronics technology delivers highly repeatable and precise timing…

20.000 MHz (± 100 ppm)

… and the overlaying software abstractions discard it.

Some CPS applications:

telepresence
distributed physical games
traffic control and safety
financial networks
medical devices and systems
assisted living
advanced automotive systems,
energy conservation
environmental control
aviation systems
critical infrastructure (power, water)
distributed robotics
military systems
smart structures
biosystems (morphogenesis,…)

Dec. 11, 2006: Dancers in Berkeley dancing in real time with dancers in Urbana-Champagne

Potential impact
social networking and games
safe/efficient transportation
fair financial networks
integrated medical systems
distributed micro power generation
military dominance
economic dominance
disaster recovery
energy efficient buildings
alternative energy
pervasive adaptive communications
distributed service delivery
…

Topics we will study

Model-Based Design

- Implementation code based on a mathematical model

System Analysis

- Verify that your model & implementation will meet a spec.

Concurrency

- Run multiple tasks correctly and efficiently

Time & Resources

- Ensuring that tasks finish on time and within budgets

Networking and other Advanced Topics

- Automotive networks, mapping an area by a robot, etc.