Cyber Physical Systems - Hybrid Control

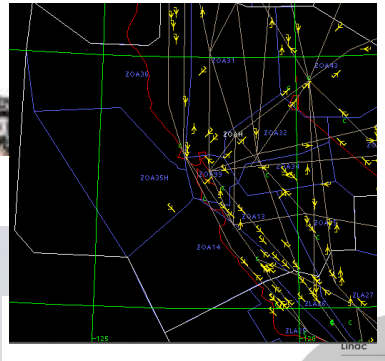
Lecture 1: Introduction

Andreas Podelski

with material from
Edward A. Lee & Sanjit A. Seshia,
Introduction to Embedded Systems
leeseshia.org

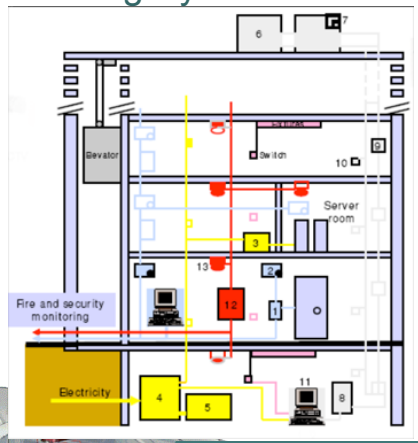
Cyber-Physical Systems

Avionics



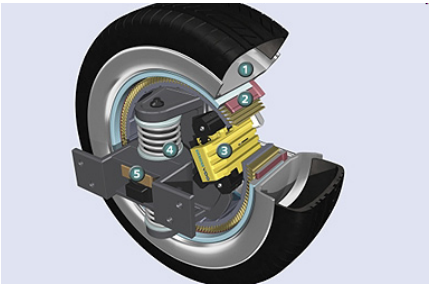
Transportation (Air traffic control at SFO)

Building Systems

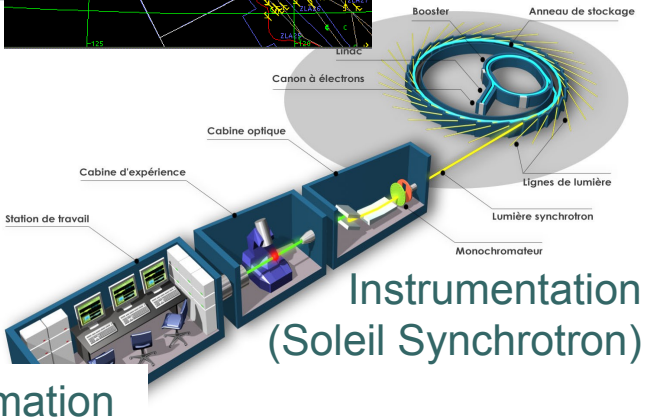


Telecommunications

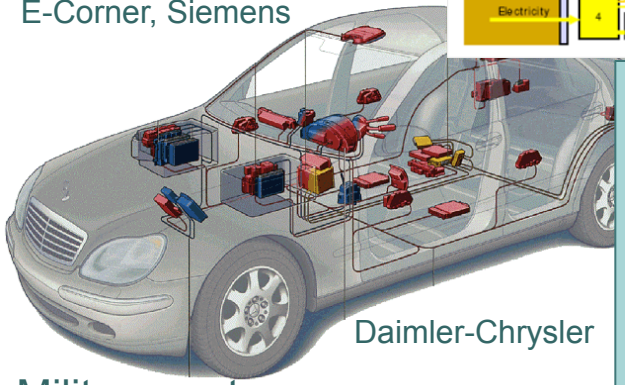
Automotive



E-Corner, Siemens

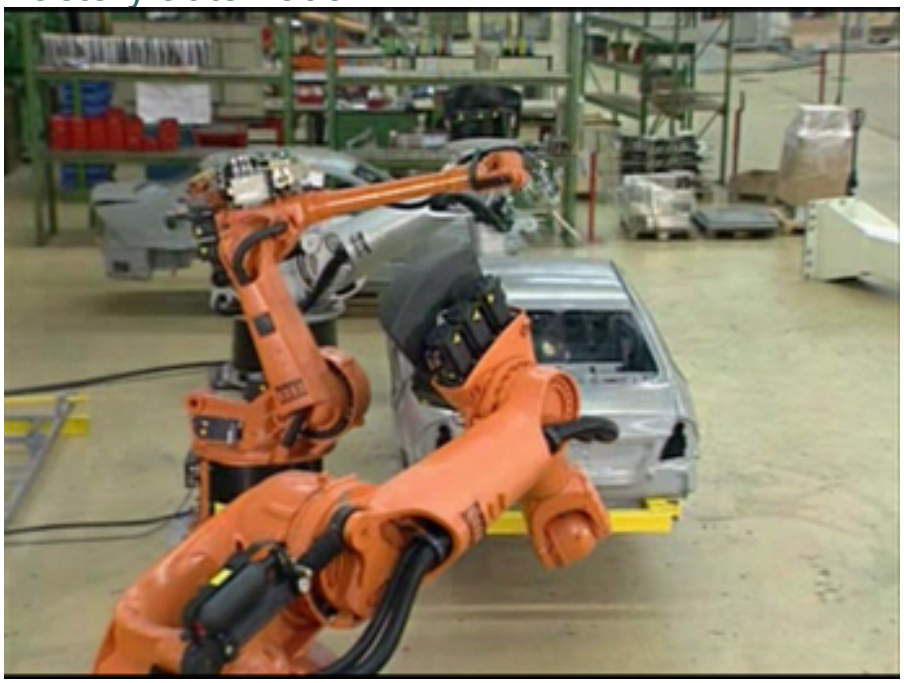


Instrumentation (Soleil Synchrotron)



Daimler-Chrysler

Factory automation

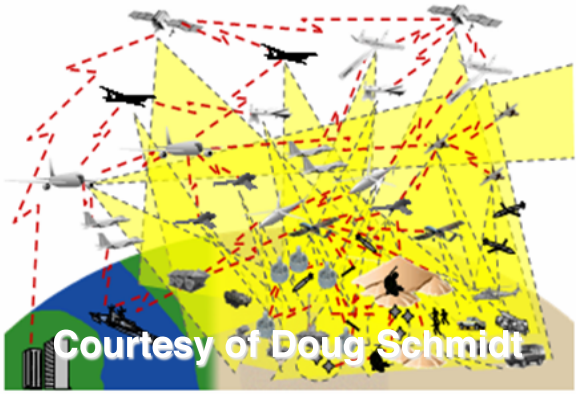


Courtesy of Kuka Robotics Corp.

Power generation and distribution

Courtesy of General Electric

Military systems:



Cyber-Physical Systems (CPS)

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,…)

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

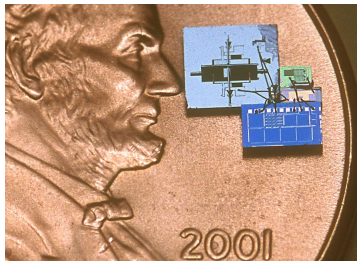
Cyber-Physical Systems (CPS)

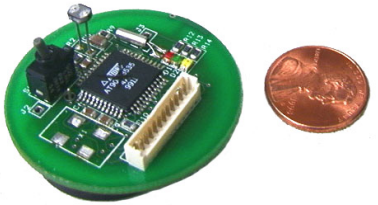
networked computational resources interacting with physical systems

- Automotive controllers
- Avionics
- Medical devices
- Industrial control
- Energy management and conservation
- . . .

US Research Council Report

"Driven by the increasing capabilities and ever declining costs of computing and communications devices, **IT is being embedded into a growing range of physical devices linked together through networks** and will become ever more pervasive as the component technologies become smaller, faster, and cheaper... These networked systems of embedded computers ... have the potential to change radically the way people interact with their environment by linking together a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways."





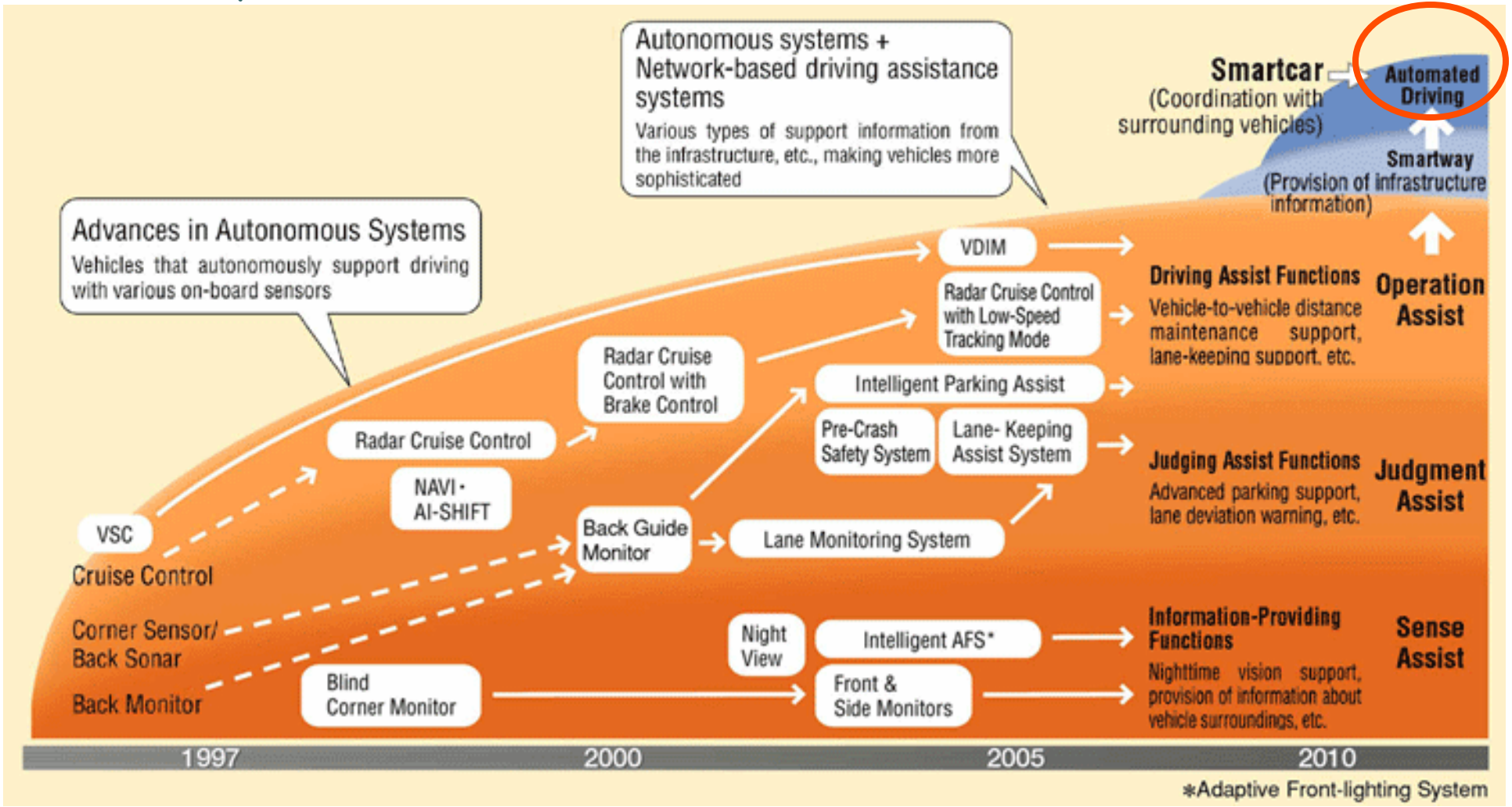
Automotive electronics today

About 80 computers (ECUs) in a car:

- 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

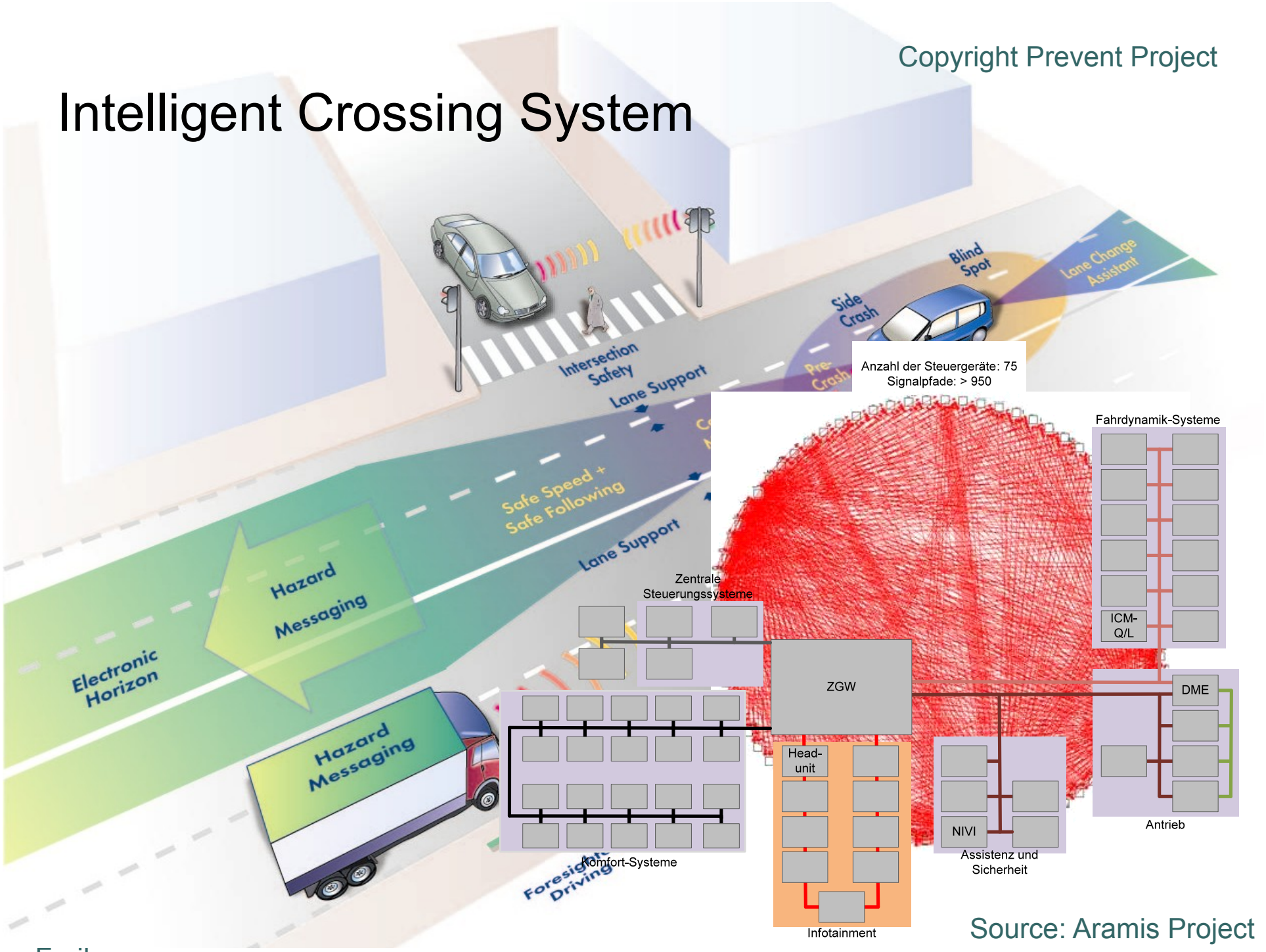
Toyota autonomous vehicle technology roadmap

Source: Toyota Web site



Copyright Prevent Project

Intelligent Crossing System



Source: Aramis Project

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.

Good Engineering?

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.

A Key Challenge: Real Time

basic premise for programming (in C, C#, Java, etc.):
correctness of a program does not depend on real time.

useful programming abstraction:
ignore timing behavior

Abstraction of time exploited everywhere

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)
…

Real time

Make it faster!

What if you need "absolutely positively on time"?

Practice in the past: write code, build your system, and test for timing

Model-based design: specify model and analyze dynamic behavior / timing

CPS vs. Embedded Systems

embedded systems view:

software on small computers => limited resources

technical problem: extract performance

CPS view:

computation and networking integrated with physical processes

technical problem: manage dynamics, time, and concurrency

Cyber-Physical Systems are . . .

Computational

- but not first-and-foremost a computer

Integral with physical processes

- sensors, actuators, physical dynamics

Reactive

- at speed of environment (timing!)

Networked

- concurrent, distributed, dynamic

Focus of traditional embedded systems vs. CPS

Traditional embedded:

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

CPS:

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

Embedded systems problem: resource limitation

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

Engineering emphasizes 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

Fundamental problems

time matters

- "as fast as possible" is not good enough

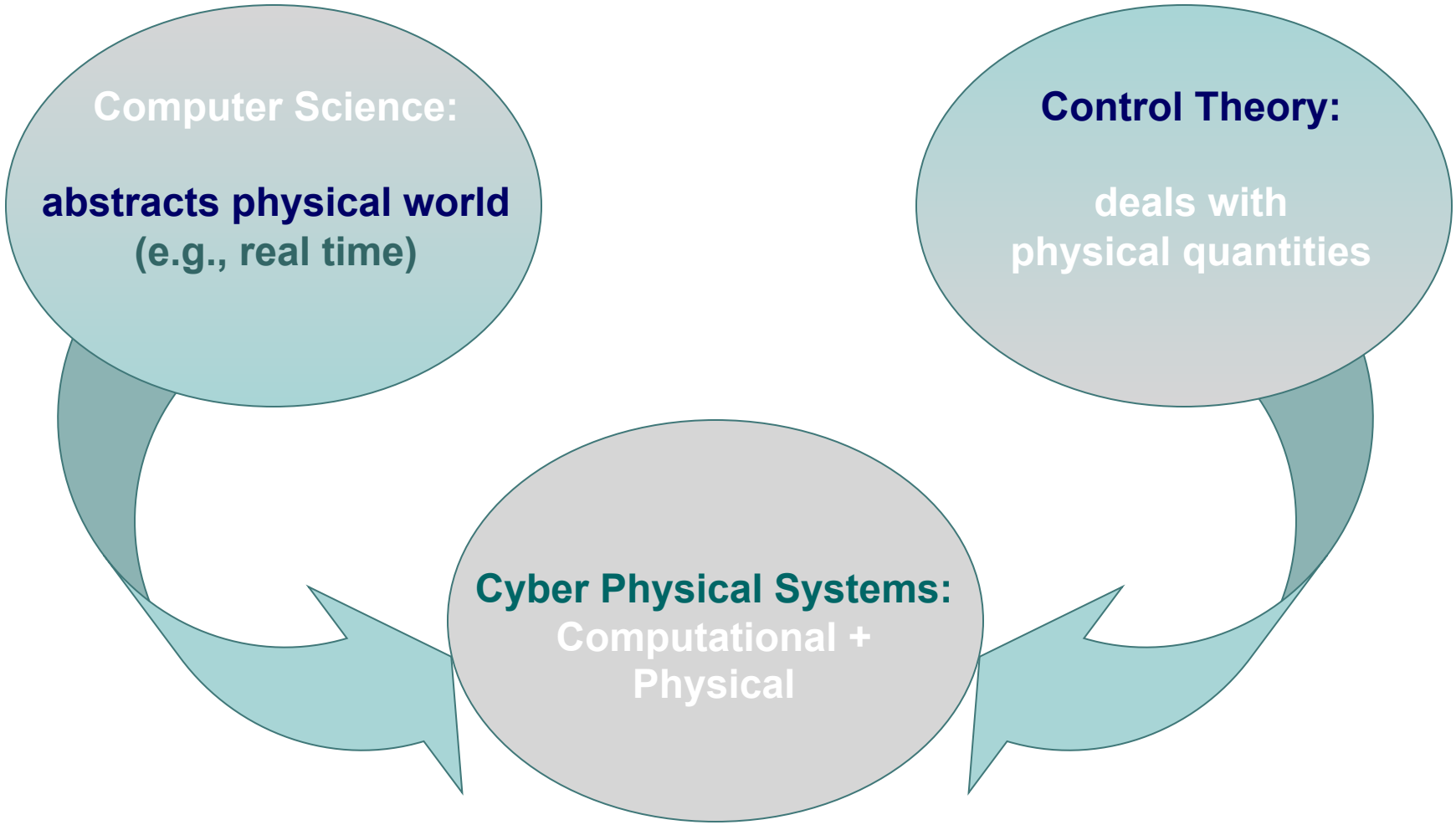
concurrency is intrinsic

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

environment is physical

- behavior obeys physical laws
- depends on continuous variables (force, acceleration, speed, position)

CPS is Multidisciplinary



First Challenge

Models for the physical world and for computation diverge.

- physical: time continuum, differential equations, dynamics
- computational: algorithm, procedure, state transitions, logic

bridge the gap:

- use physical world models to specify behavior of systems
- use computational view to study dynamics of physical system

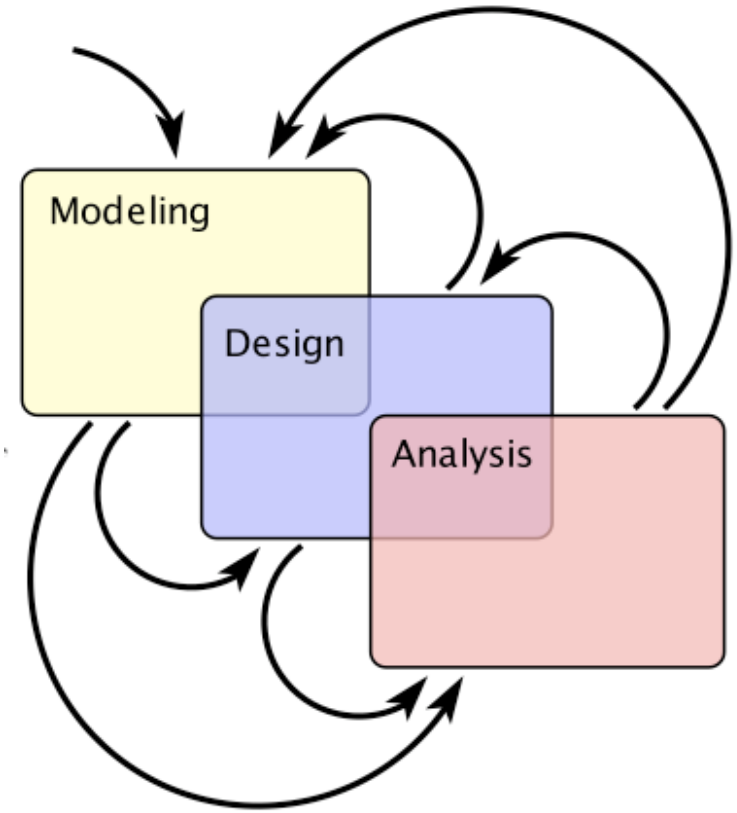
Models in Model-Based Design & Verification

- Models describe physical dynamics.
- Specifications are executable models.
- Models are composed to form designs.
- Models evolve during design.
- Deployed code may be (partially) generated from models.
- Modeling languages have semantics.
- Modeling languages themselves may be modeled (meta models).

For cyber-physical systems, the model accounts for:

- Time
- Concurrency
- Dynamics

Modeling, Design, Analysis



Model

specifies **what** a system does.

Design

specifies **how** a system does what it does.

Analysis

specifies **why** a system does what it does
(or fails to do what it should do).

Model

artifact that imitates the system

mathematical model:

- definitions in terms of mathematical formulas
- mathematical correctness statement
- formal, automatic correctness proof

formal: mathematical, logical, machine checkable

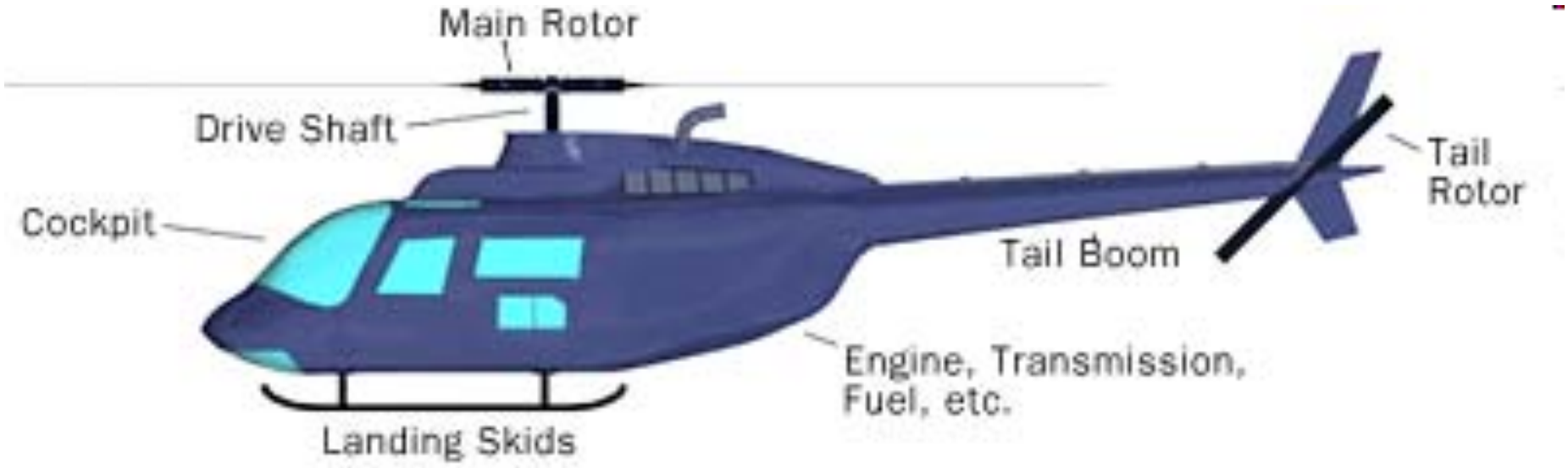
automatic: push-button, scalable

Modeling Techniques

Model = abstraction of **system dynamics**
(how things change over time)

- physical phenomena: differential equations
- mode behavior: finite-state automata
- combination: hybrid automata

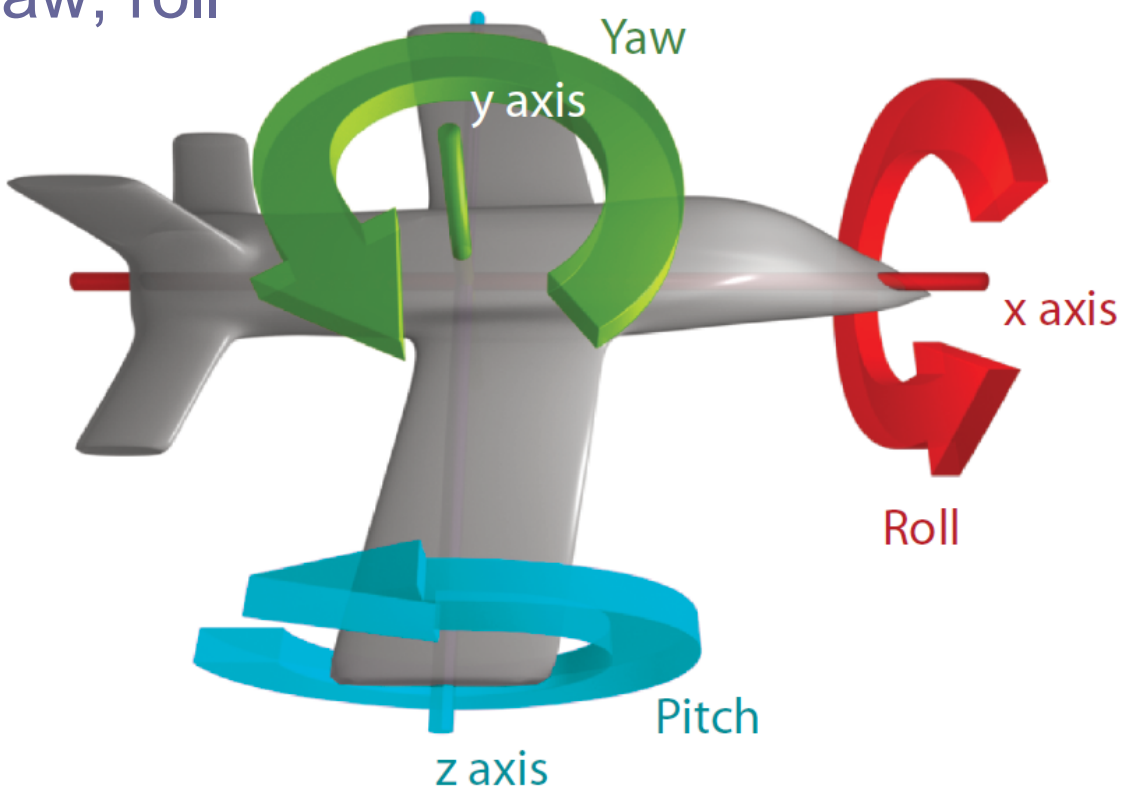
Modeling Helicopter Dynamics



Modeling Physical Motion

Six degrees of freedom:

- Position: x, y, z
- Orientation: pitch, yaw, roll



Notation

Position is given by three functions:

$$x : \mathbb{R} \to \mathbb{R}$$
$$y : \mathbb{R} \to \mathbb{R}$$
$$z : \mathbb{R} \to \mathbb{R}$$

where the domain $\mathbb{R}$ represents time and the co-domain (range) $\mathbb{R}$ represents position along the axis. Collecting into a vector:

$$\mathbf{x} : \mathbb{R} \to \mathbb{R}^3$$

Position at time $t \in \mathbb{R}$ is $\mathbf{x}(t) \in \mathbb{R}^3$.

Notation

Velocity

$$\dot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$$

is the derivative, $\forall\, t \in \mathbb{R}$,

$$\dot{\mathbf{x}}(t) = \frac{d}{dt}\mathbf{x}(t)$$

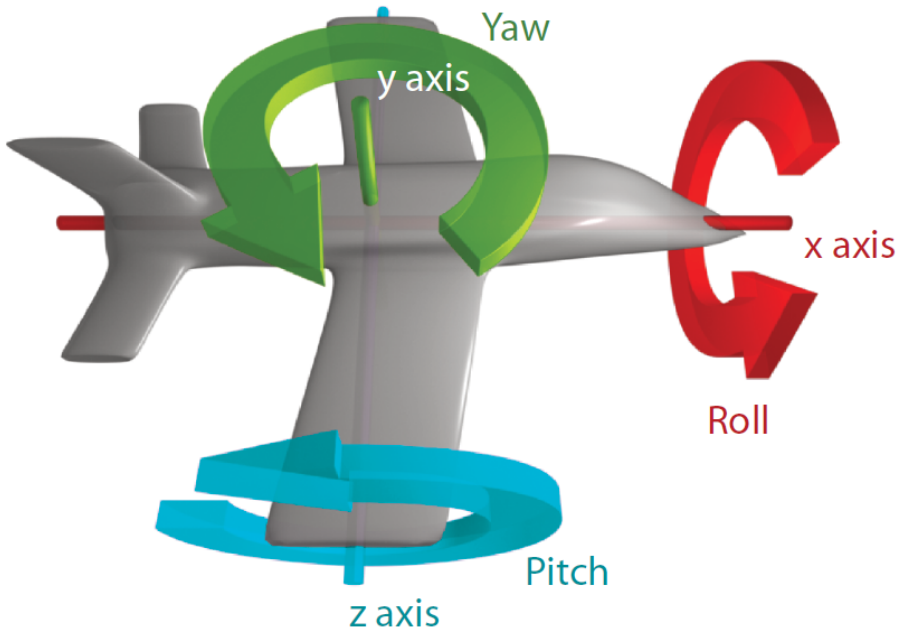
Acceleration $\ddot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$ is the second derivative,

$$\ddot{\mathbf{x}} = \frac{d^2}{dt^2}\mathbf{x}$$

Force on an object is $\mathbf{F}\colon \mathbb{R} \to \mathbb{R}^3$.

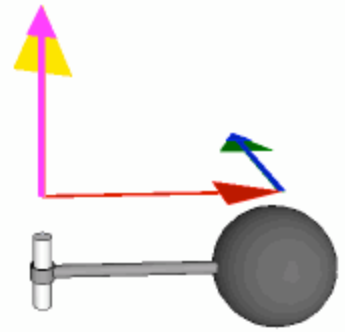
Orientation

- Orientation: $\theta : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular velocity: $\dot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular acceleration: $\ddot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Torque: $\mathbf{T} : \mathbb{R} \rightarrow \mathbb{R}^3$



$$\theta(t) = \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix} = \begin{bmatrix} \text{roll} \\ \text{yaw} \\ \text{pitch} \end{bmatrix}$$

Feedback Control Problem



A helicopter without a tail rotor, like the one below, will spin uncontrollably due to the torque induced by friction in the rotor shaft.

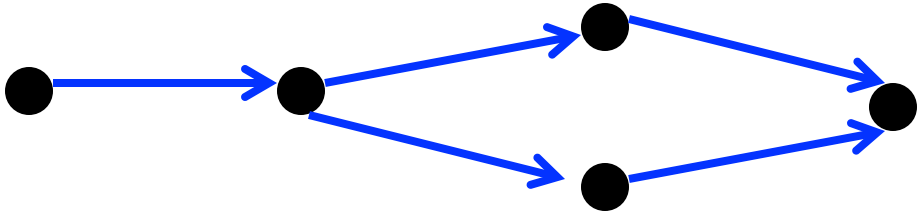
Control system problem:
Apply torque using the tail rotor to counterbalance the torque of the top rotor.

Next: Description of behavior by traces

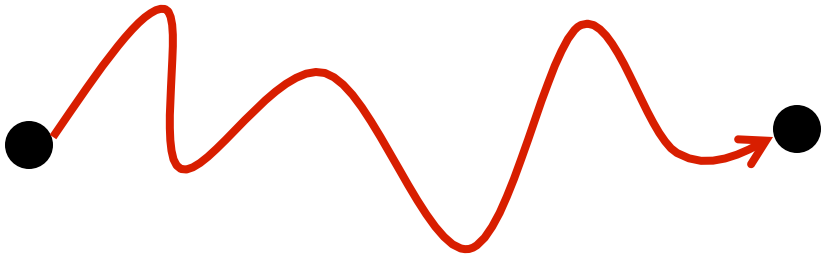
Model = abstraction of **system dynamics**

- physical phenomena:
  differential equations
- computation / discrete mode change:
  finite-state automata
- combination:
  hybrid automata

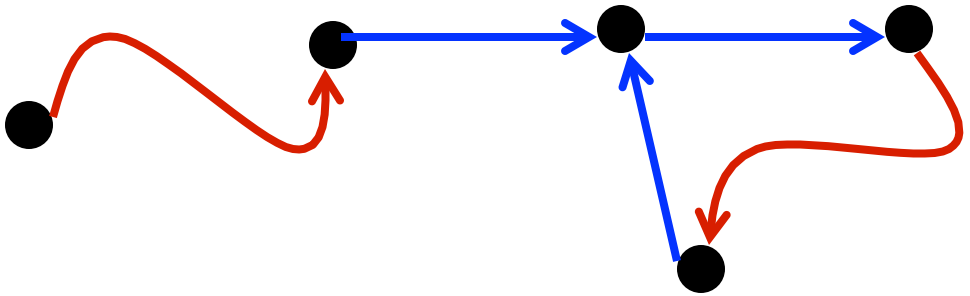
Discrete System (FSM)

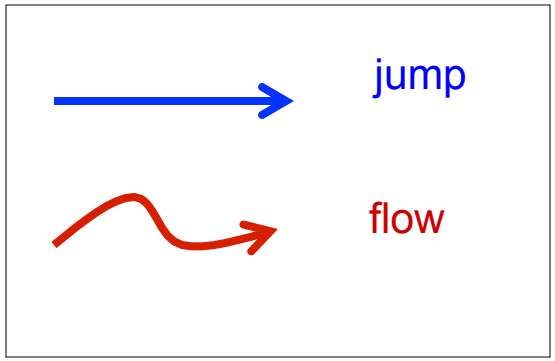


Continuous System



Hybrid System





Next:

- Examples of Hybrid Systems
- The *Hybrid Automaton* Model

Thermostat

State has both discrete and continuous components:

$x \in \mathbb{R}$ temperature

$h \in \{\text{on}, \text{off}\}$ heating mode

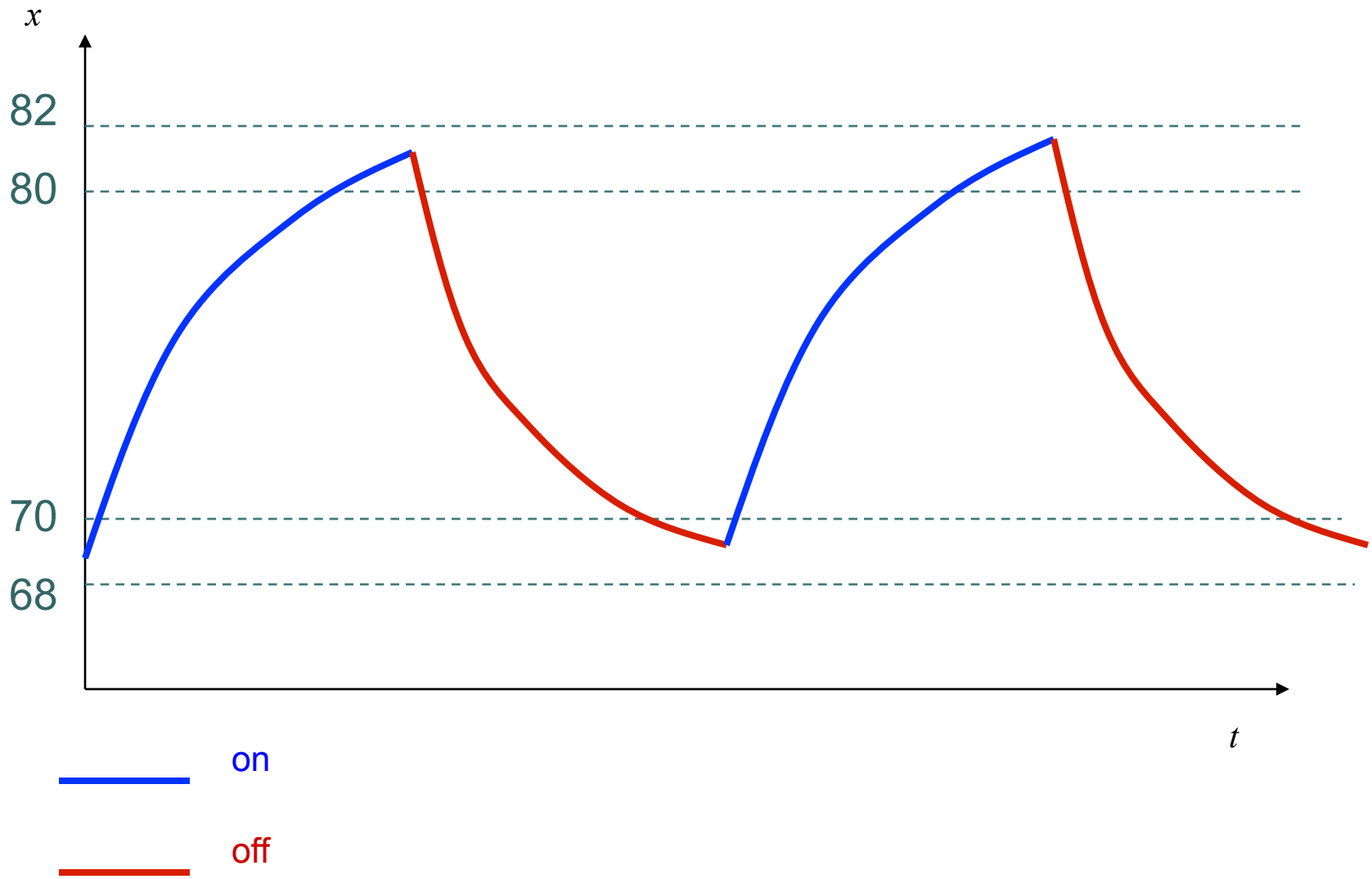
Flow in each mode is:

$$h = \text{on} \wedge x < 82 \qquad \dot{x} = K(100 - x)$$
$$h = \text{off} \wedge x > 68 \qquad \dot{x} = -Kx$$

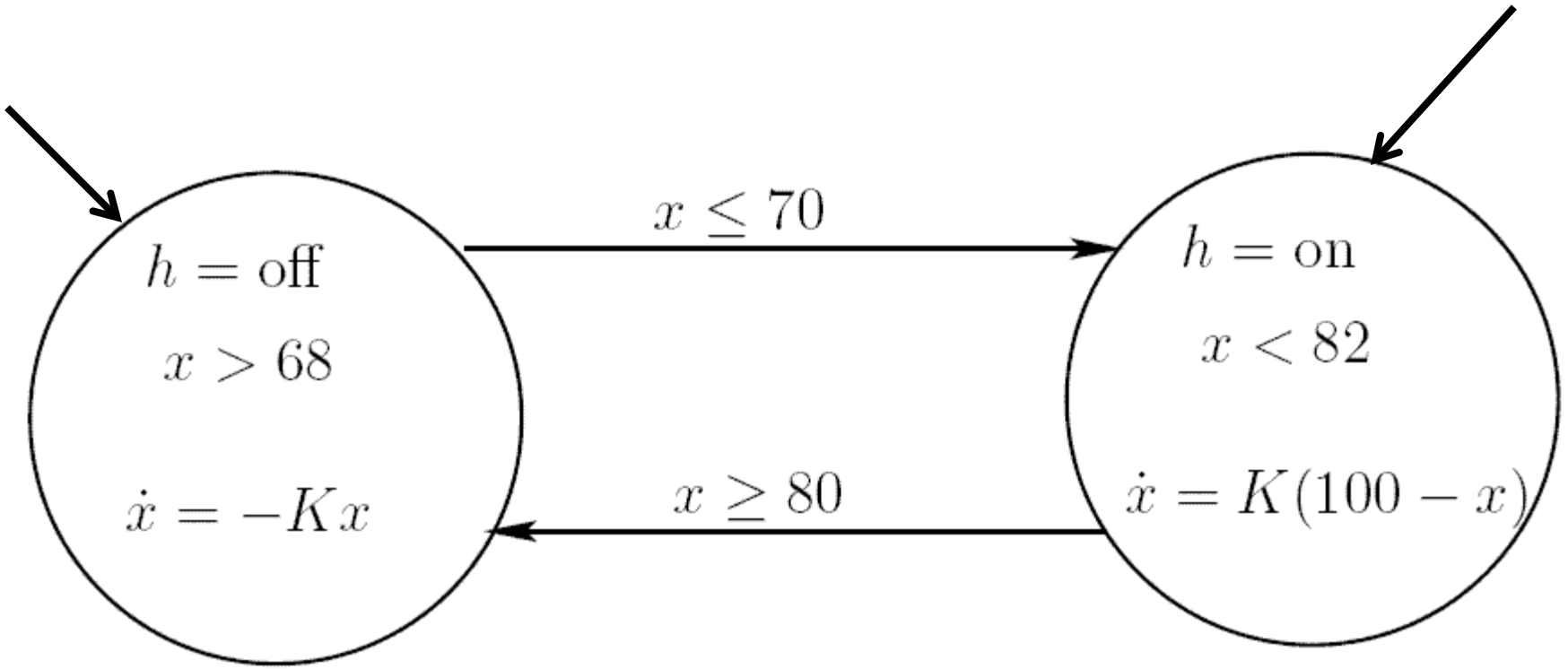
Jumps between modes: (happen instantaneously)

$$h = \text{on} \wedge x \geq 80 \quad \rightarrow \quad h := \text{off}$$
$$h = \text{off} \wedge x \leq 70 \quad \rightarrow \quad h := \text{on}$$

Dynamics of Thermostat



Hybrid Automaton for Thermostat



Is this automaton deterministic?