Embedded Systems

Embedded Systems

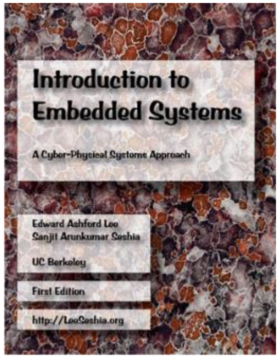
- Ruzica Piskac (ruzica.piskac@yale.edu)
- Leander Tentrup (tentrup@cs.uni-saarland.de)
- Michael Gerke (gerke@cs.uni-saarland.de)
- Felix Klein (klein@cs.uni-saarland.de)

- Stammvorlesung 9 CP

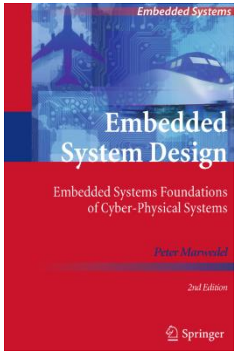
- Lectures:
  - Tuesdays, 16:15-18:00
  - Thursdays, 10:15-12:00

Textbooks

- Edward A. Lee and Sanjit A. Seshia,
  Introduction to Embedded Systems,
  A Cyber-Physical Approach, 2011.
  Available online from leeseshia.org



- Peter Marwedel,
  Embedded System Design.
  Springer, Berlin; 2nd Edition, 2011.



- Giorgio C. Buttazzo
  Hard Real-Time Computing Systems:
  Predictable Scheduling Algorithms and
  Applications, Springer, 2011

Problem Sets

- Website:

http://react.cs.uni-sb.de/teaching/embedded-systems-14/

- Problem sets released every Tuesday (first on May 06)
- Due next **Tuesday afternoon before the lecture** (you can just hand it in before the lecture), work in groups of three students
- Weekly discussion sessions (15 minutes each)
- **Individual** feedback:
  mandatory discussion slot per group
- Format: 15 minutes, slots on Thursday and Friday
- No grading / solutions only presented in tutorials

Exam Policy

- **Qualification:** Miss at most two discussion slots & hand in solutions to all problem sets, have to complete the projects
- Project: more details during next lectures
- Three exams: Midterm/End-of-Term Exam/End-of-Semester Exam
- Need to pass **2 out of 3** to pass the course
- Grading: average of **best 2**

Embedded Systems

Computers whose job is not primarily information processing, but rather is interacting with physical processes.

A broader view is that of *cyber-physical systems (CPS)*

Estimates for number of embedded systems in current use: >10^{10}

[Rammig 2000, Motorola 2001]







Key Terms

- **Embedded Systems:** Information processing systems embedded into enclosing products.

Examples: Systems with real-time constraints and efficiency requirements like automobiles, telecommunication or fabrication equipment

- **Cyber-Physical Systems:** Integrations of computation and physical processes.

Example: Networked systems of embedded computers linking together a range of devices and sensors for information sharing


400 horses
100 microprocessors
M NH 2519

Stanford, IEEE Spectrum

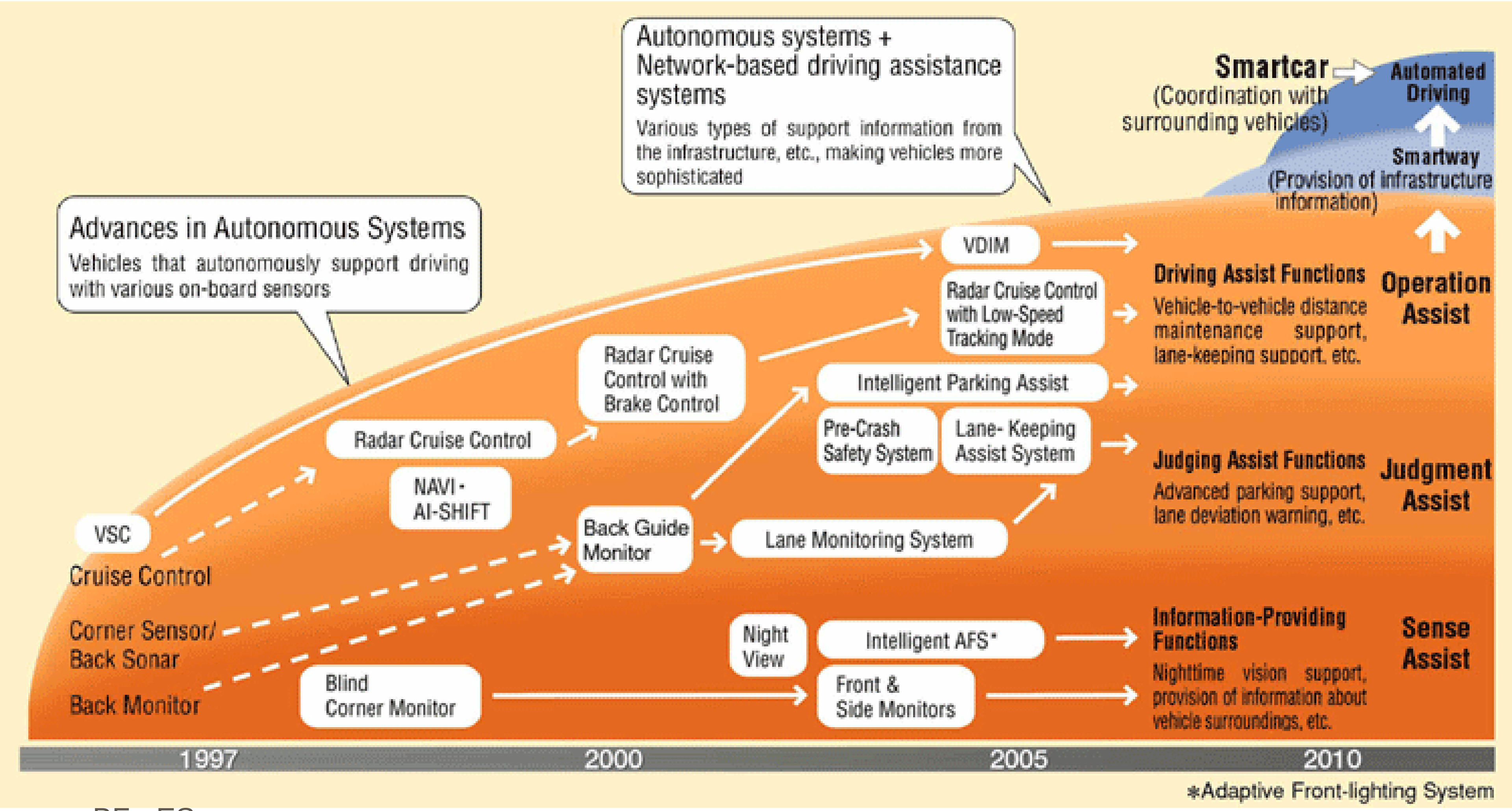
Automotive Software: An Emerging Domain
A Software Perspective

- Up to 40% of the vehicles' costs are determined by electronics and software
- 90% of all innovations are driven by electronics and software
- 50 – 70% of the development costs for an ECU are related to software
- Premium cars have up to 70 ECUs, connected by 5 system busses

- **Growing system complexity**
- **More dependencies**
- **Costs play significant role**

Example: Toyota autonomous vehicle technology roadmap, c. 2007

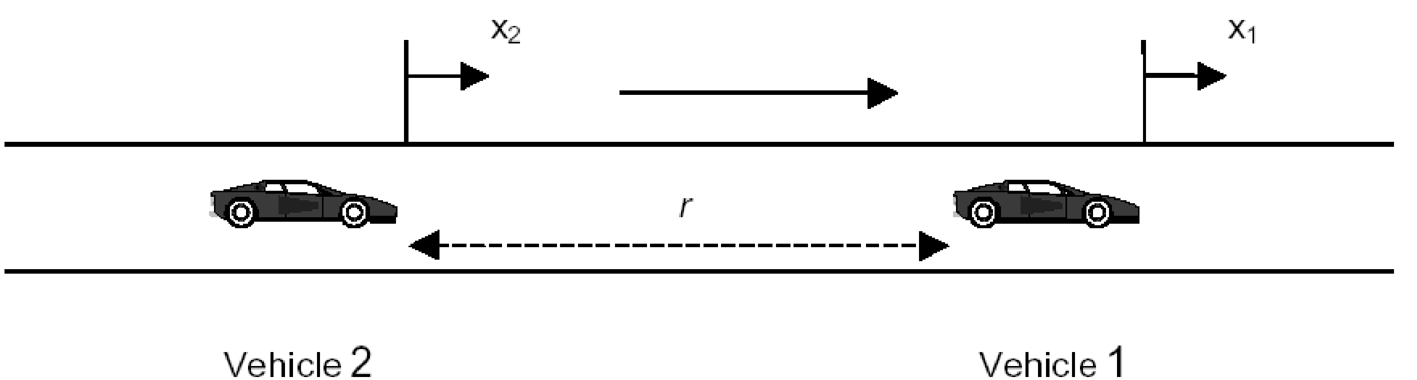
Source: Toyota Web site



Intelligent Cruise Control

Cooperative Adaptive (Intelligent) Cruise Control (CACC):

Cruise at given speed when the road is clear (cruise control), otherwise follow the car in front, using radar (adaptive) and/or communications (cooperative).



https://www.youtube.com/watch?v=PDPuNlWXtlI

Thanks to PATH publication unit

Manufacturing Robots Automated Assembly

Printing Press

Bosch-Rexroth

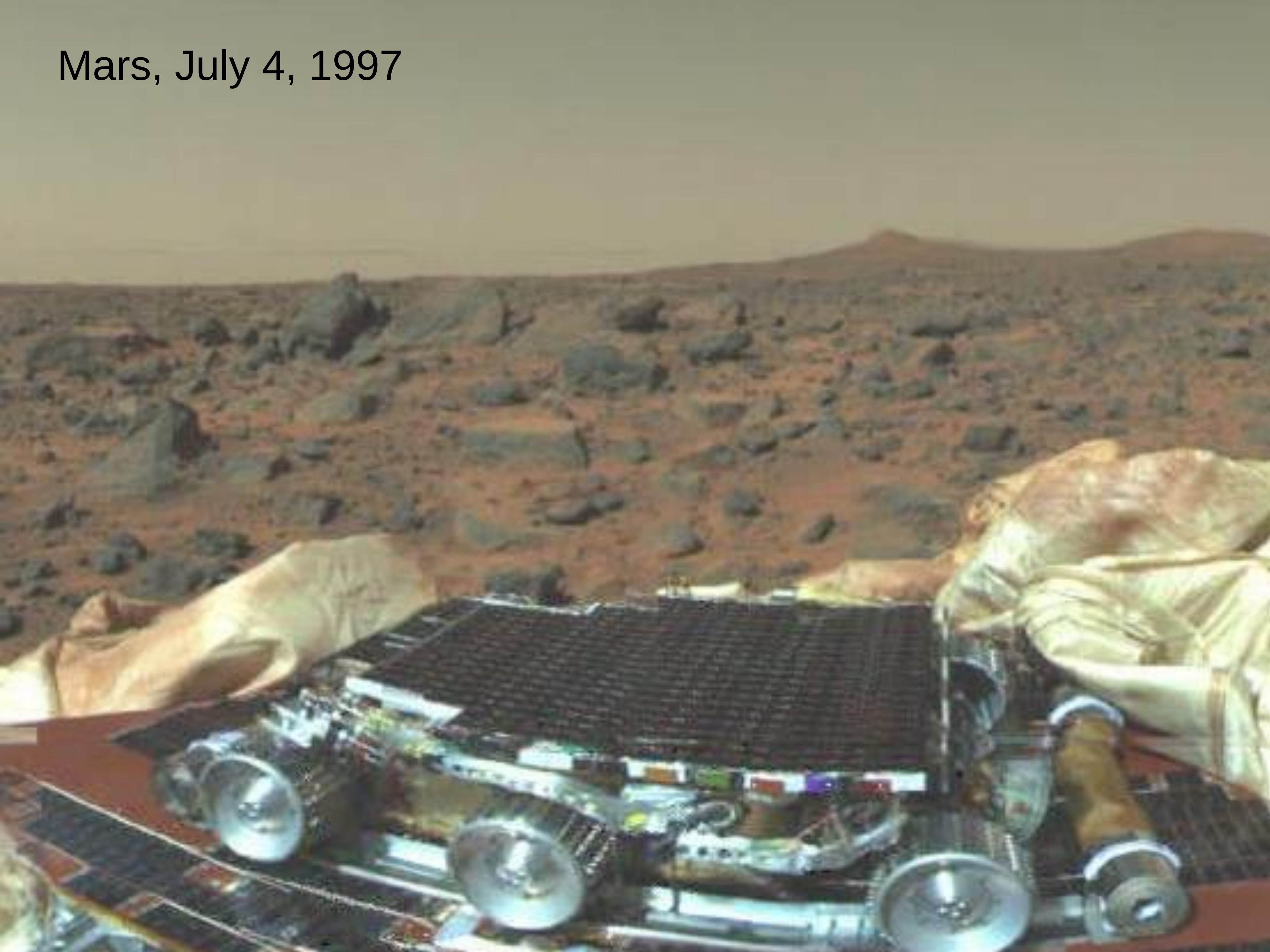
- High-speed, high precision
  - Speed: 1 inch/ms
  - Precision: 0.01 inch

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

http[illegible]pressman.blogspot.de/2011/03/how-flying-paster-works.html

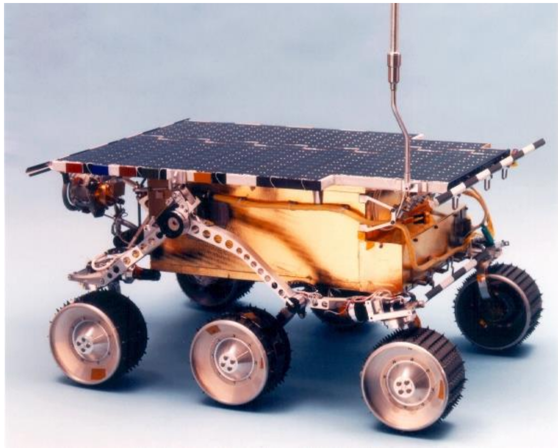
The DLR heart
http://www.dualis-medtech.com

Mars, July 4, 1997



The MARS Pathfinder problem

“But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".” …



The MARS Pathfinder problem

- System overview:
  - ***Information Bus (IB):***
    - Buffer for exchanging data between different tasks
    - Shared resource of two tasks M and B
  - Three tasks:
    - ***Meteorological data gathering task (M):***
      - – collects meteorological data
      - – reserves IB, writes data to IB, releases IB
      - – infrequent task, low priority
    - ***Bus management (B):***
      - – data transport from IB to destination
      - – reserves IB, data transport, releases IB
      - – frequent task, high priority

The MARS Pathfinder problem

- Three tasks:
  - ***...***
  - ***"Communication task" (C):***
    - medium priority, does not use IB
- Scheduling with fixed priorities.
- ***Watch dog timer (W):***
  - Execution of B as indicator of system hang-up
  - If B is not activated for certain amount of time: Reset the system

The MARS Pathfinder problem

(see http://research.microsoft.com/~mbj/Mars_Pathfinder/)

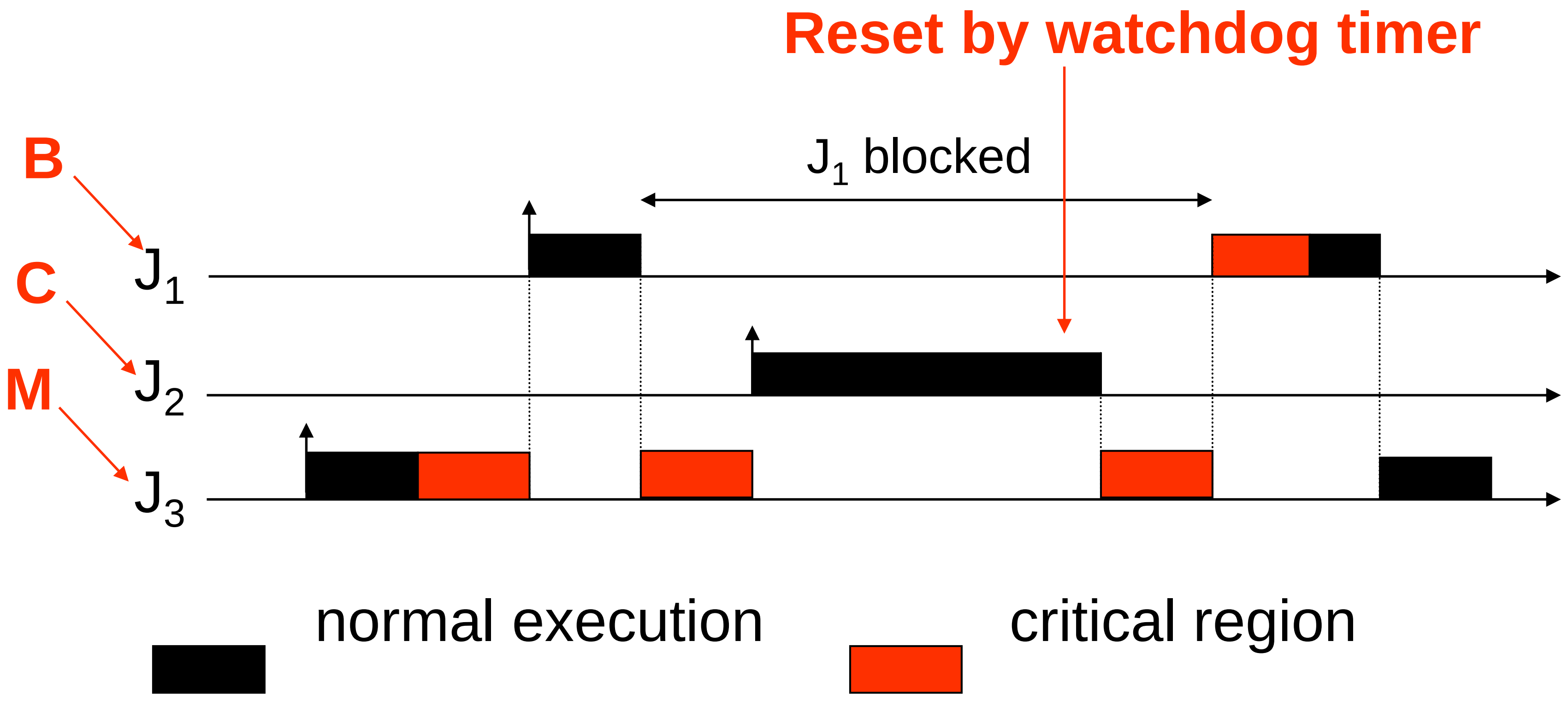
“Most of the time this combination worked fine.

However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

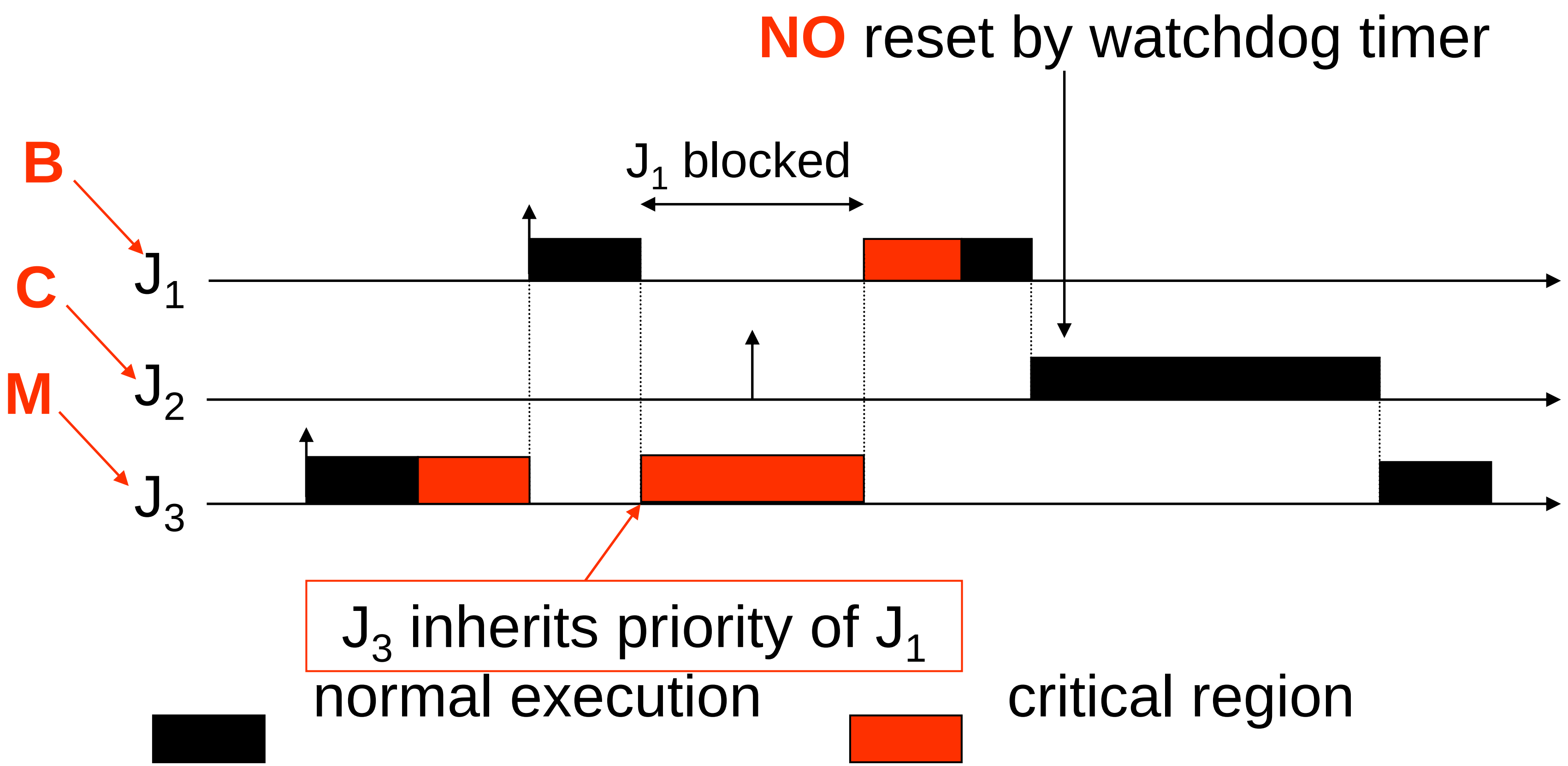
This scenario is a classic case of priority inversion.”

Priority inversion



$\text{priority}(J_1) > \text{priority}(J_2) > \text{priority}(J_3)$

Classic solution: Priority inheritance

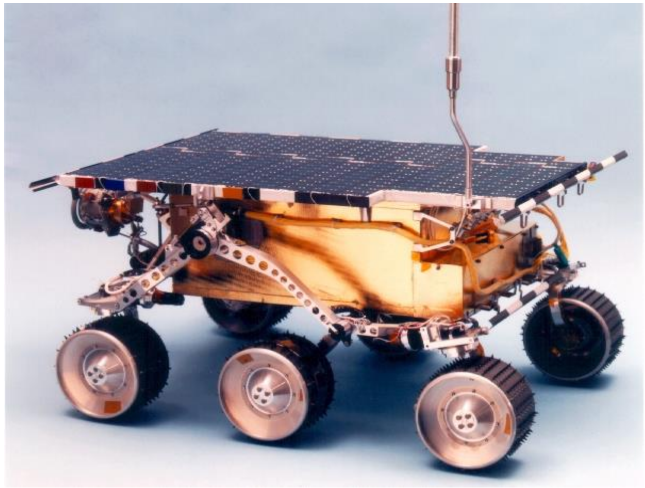


priority(J_1) > priority(J_2) > priority(J_3)

Priority inversion on Mars

- Priority inheritance also solved the Mars Pathfinder problem:
    - the VxWorks operating system used in the pathfinder implements a flag for the calls to mutual exclusion primitives.
    - This flag allows priority inheritance to be set to “on”.
    - When the software was shipped, it was set to “off”.

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to “on”, while the Pathfinder was already on the Mars [Jones, 1997].



Embedded Systems

Embedded system = **engineering artifact involving computation that is subject to physical constraints**

Constraint #1: Reaction to the physical environment

Reaction constraints: deadlines, throughput, jitter

Constraint #2: Execution on a physical platform

Execution constraints: Bounds on available processor speeds, power, hardware failure rates

Challenge: Gain control over the interplay of computation with reaction and execution constraints, so as to meet given requirements.

Characteristics of Embedded Systems

Must be **efficient:**

- **Energy** efficient
- **Code-size** efficient (especially for systems on a chip)
- **Run-time** efficient
- **Weight** efficient
- **Cost** efficient

Dedicated towards a certain application

Knowledge about behavior at design time can be used to minimize resources and to maximize robustness

Dedicated user interface

(no mouse, keyboard and screen)

Characteristics of Embedded Systems

Many ES must meet real-time constraints

A real-time system must react to stimuli from the controlled object (or the operator) within the time interval dictated by the environment.

For real-time systems, right answers arriving too late are wrong.

„A real-time constraint is called **hard**, if not meeting that constraint could result in a catastrophe“ [Kopetz, 1997].

All other time-constraints are called **soft.**

Characteristics of Embedded Systems

Frequently connected to physical environment through sensors and actuators.

Typically Embedded Systems are

- **Hybrid systems** (analog + digital parts)
- **Reactive systems**

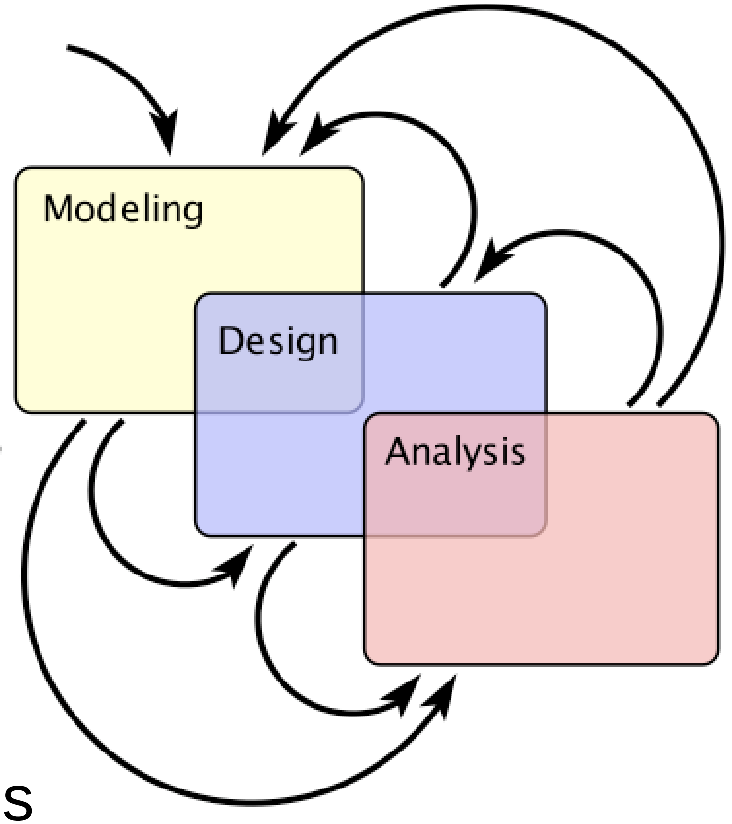
„A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment“ [Bergé, 1995]

Behavior depends on input and current state.

Course Topics

- Model-Based Design
  - Implementation based on a mathematical model
- Embedded Systems Hardware
  - Sensors, processing units, communication
- Embedded Systems Software
  - Scheduling
- Hardware-Software Codesign
  - methods for the optimal division of labor
- System Analysis
  - Testing, reliability, worst-case execution time, etc.

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).

What is Modeling?

- Developing insight about a system, process, or artifact through imitation.

- A *model* is the artifact that imitates the system, process, or artifact of interest.

- A *mathematical model* is a model in the form of a set of definitions and mathematical formulas.

What is Model-Based Design?

1. Create a mathematical model of all the parts of the embedded system

    - Physical world
    - Control system
    - Software environment
    - Hardware platform
    - Network
    - Sensors and actuators

2. Construct the implementation from the model

    - Construction may be automated, like a compiler
    - Some parts are automatically constructed

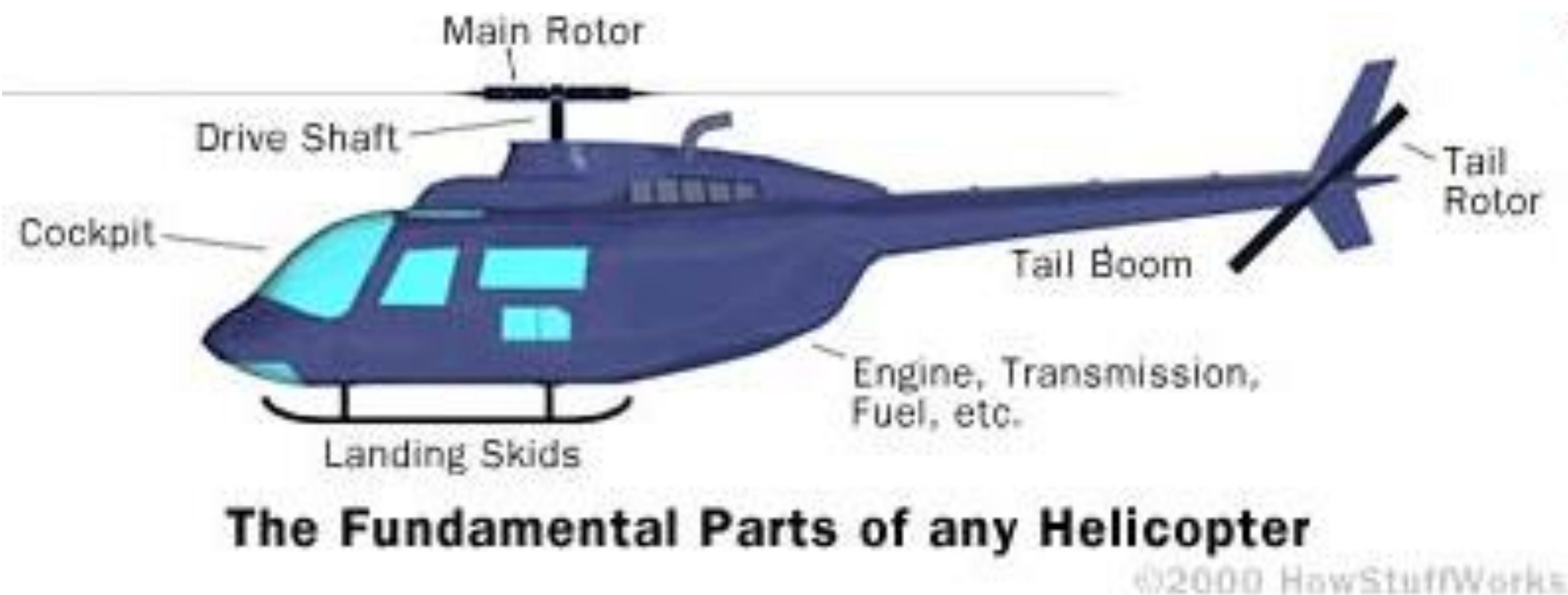
Modeling Techniques in this Course

Models that are abstractions of **system dynamics** (how things change over time)

Examples:

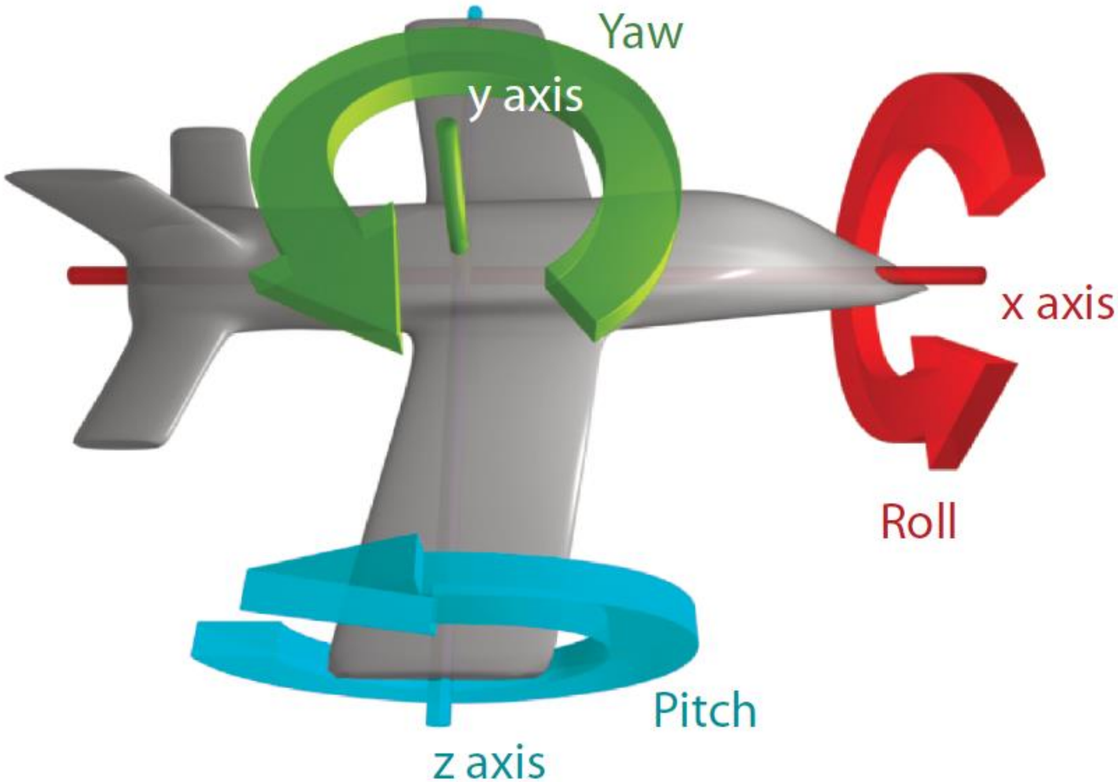
- Modeling physical phenomena – ODEs
- Modeling modal behavior – FSMs, hybrid automata
- Real-time constraints – timed automata
- Hierarchy – StateCharts
- Concurrency – Petri Nets
- Modeling networks – RTC

An Example: 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:

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

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

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

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

Notation

Velocity

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

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

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

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

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

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

Newton's Second Law

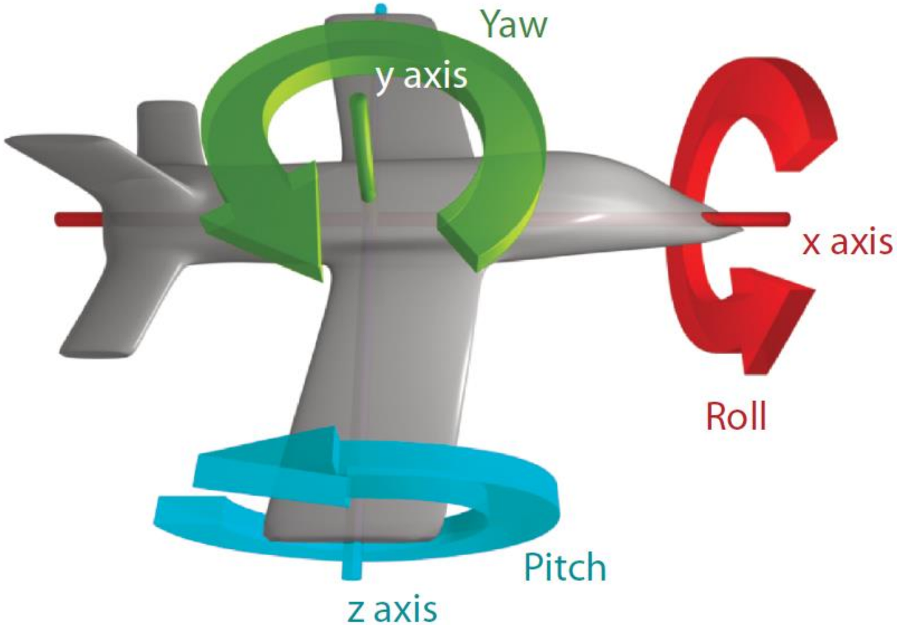
Newton's second law states $\forall\, t \in \mathbb{R}$,

$$\mathbf{F}(t) = M\ddot{\mathbf{x}}(t)$$

where M is the mass. To account for initial position and velocity, convert this to an integral equation

$$\begin{aligned}\mathbf{x}(t) &= \mathbf{x}(0) + \int_0^t \dot{\mathbf{x}}(\tau)d\tau \\ &= \mathbf{x}(0) + t\dot{\mathbf{x}}(0) + \frac{1}{M}\int_0^t \int_0^\tau \mathbf{F}(\alpha)d\alpha d\tau,\end{aligned}$$

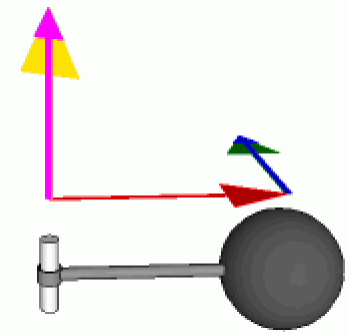
Orientation



- 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: $\mathbf{T} : \mathbb{R} \rightarrow \mathbb{R}^3$

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

Torque



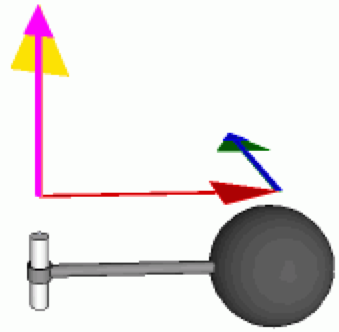
Yaw dynamics:

$$T_y(t) = I_{yy}\ddot{\theta}_y(t)$$

To account for initial angular velocity, write as

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}}\int_0^t T_y(\tau)d\tau.$$

Feedback Control Problem

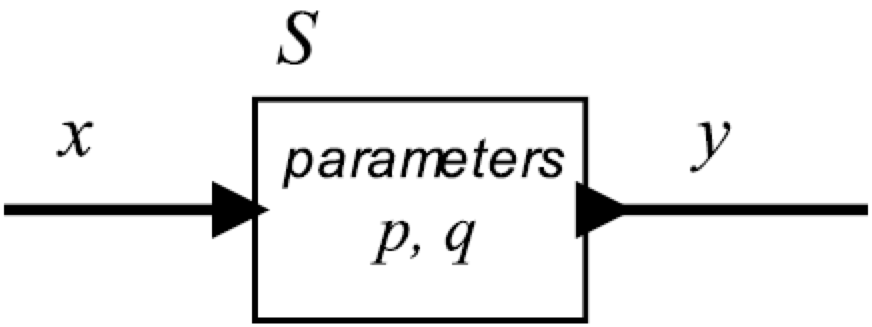


A helicopter without a tail rotor, like the one below, will spin uncontrollably due to the torque induced by friction in the rotor shaft.

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

Actor Model of Systems

- A *system* is a function that accepts an input *signal* and yields an output signal.



- The domain and range of the system function are sets of signals, which themselves are functions.

$$x: \mathbb{R} \to \mathbb{R}, \quad y: \mathbb{R} \to \mathbb{R}$$

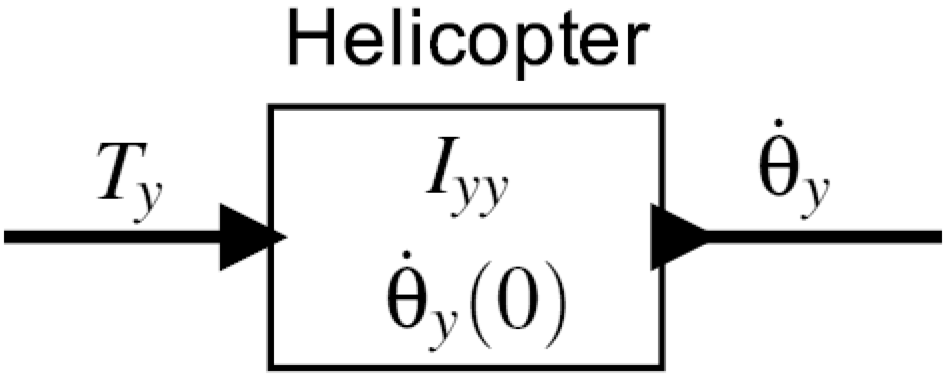
$$S: X \to Y$$

$$X = Y = (\mathbb{R} \to \mathbb{R})$$

- Parameters may affect the definition of the function *S*.

Actor model of the helicopter

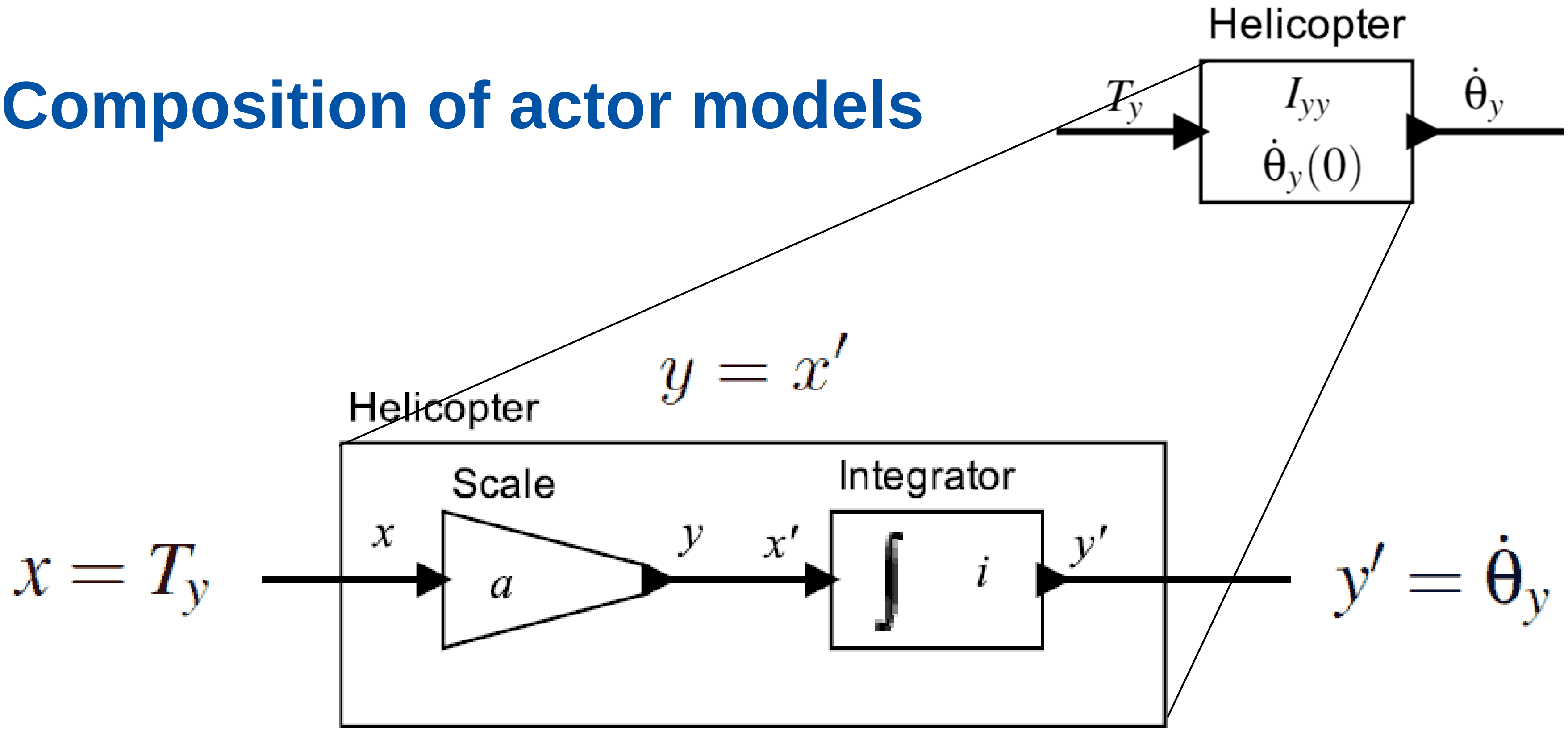
- Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the *y* axis.



Parameters of the model are shown in the box. The input and output relation is given by the equation to the right.

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

Composition of actor models

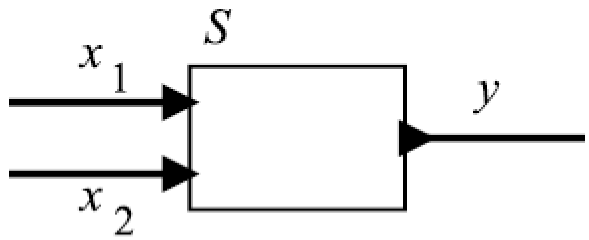


$$\forall\, t \in \mathbb{R}, \quad y(t) = a x(t) \qquad y'(t) = i + \int_0^t x'(\tau)\, d\tau$$

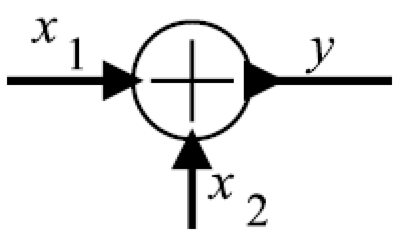
$$y = ax$$

$$a = 1/I_{yy} \qquad i = \dot{\theta}_y(0)$$

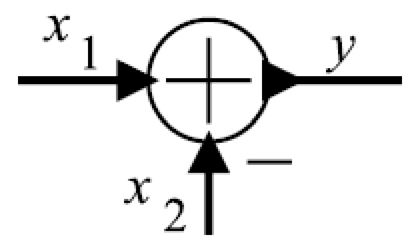
Actor models with multiple inputs



$$S\colon (\mathbb{R} \to \mathbb{R})^2 \to (\mathbb{R} \to \mathbb{R})$$



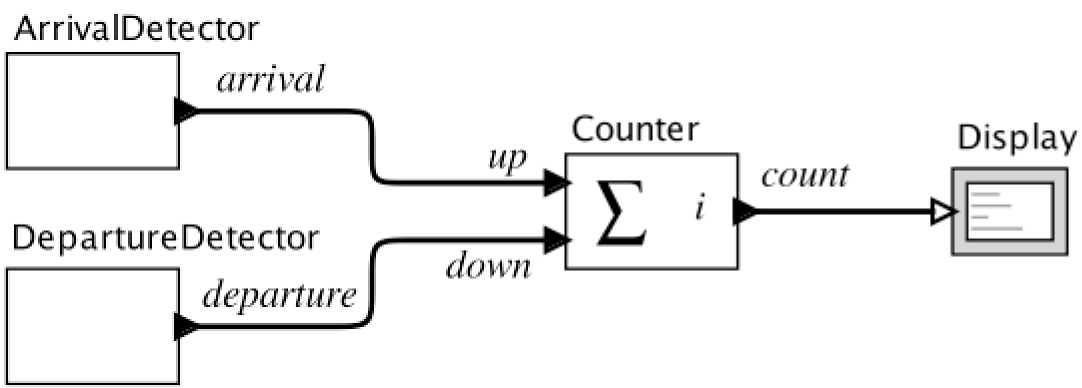
$$\forall\, t \in \mathbb{R}, \quad y(t) = x_1(t) + x_2(t)$$



$$(S(x_1, x_2))(t) = y(t) = x_1(t) - x_2(t)$$

Discrete Systems

- Example: count the number of cars that enter and leave a parking garage:



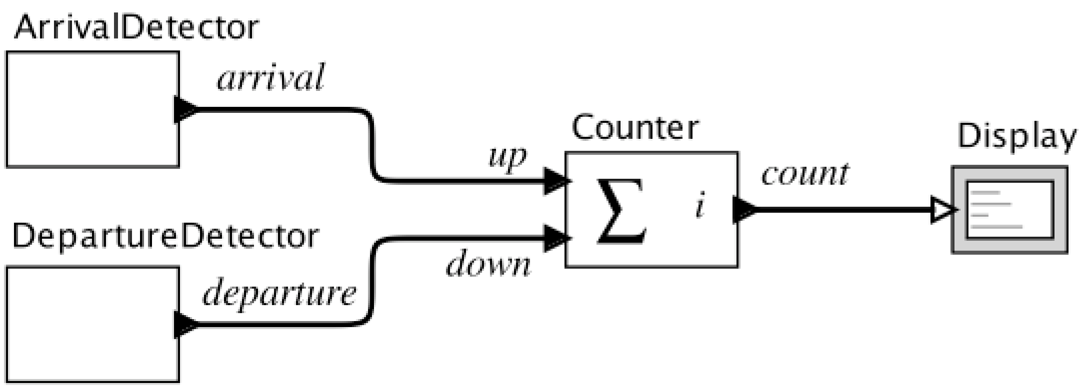
- Pure signal:
- Discrete actor: $up: \mathbb{R} \to \{absent, present\}$

$$Counter\colon (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$$

$$P = \{up, down\}$$

Reaction

For any $t \in \mathbb{R}$ where $up(t) \neq absent$ or $down(t) \neq absent$ the Counter **reacts**. It produces an output value in $\mathbb{N}$ and changes its internal **state**.

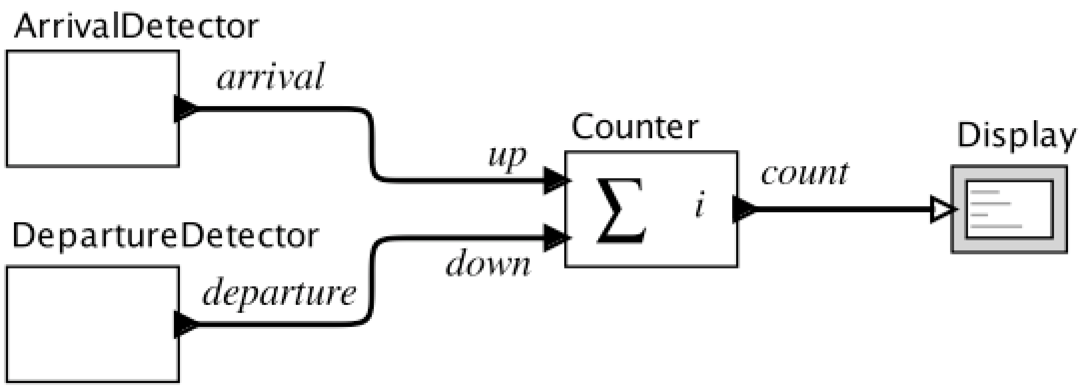


$$Counter\colon (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$$
$$P = \{up, down\}$$

Input and Output Valuations at a Reaction

For $t \in \mathbb{R}$ a port p has a **valuation**, which is an assignment of a value in V_p (the **type** of port p). A valuation of the input ports $P = \{up, down\}$ assigns to each port a value in $\{absent, present\}$.

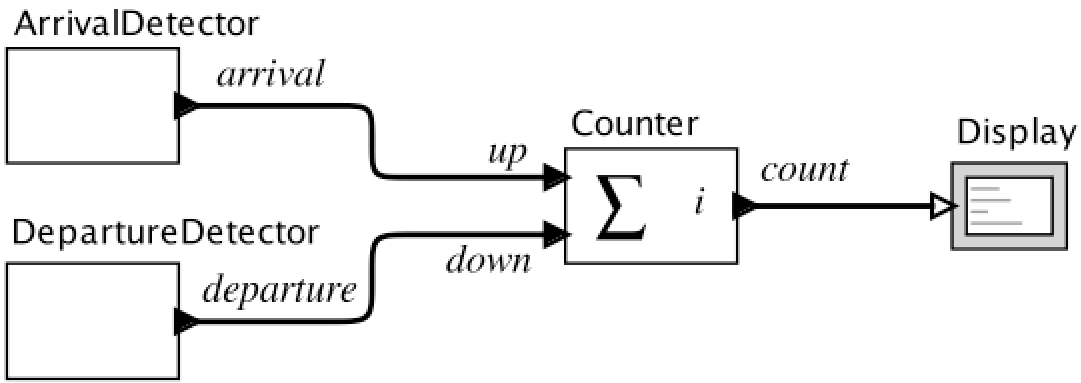
A **reaction** gives a valuation to the output port *count* in the set $\{absent\} \cup \mathbb{N}$.



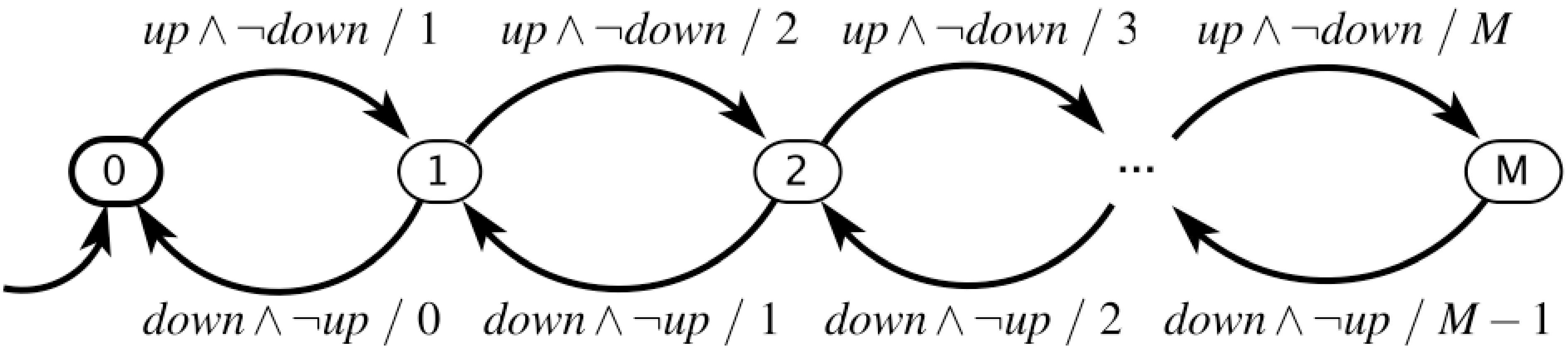
State Space

A practical parking garage has a finite number M of spaces, so the state space for the counter is

$$States = \{0, 1, 2, \cdots, M\} \ .$$



Garage Counter Finite State Machine (FSM) in Pictures

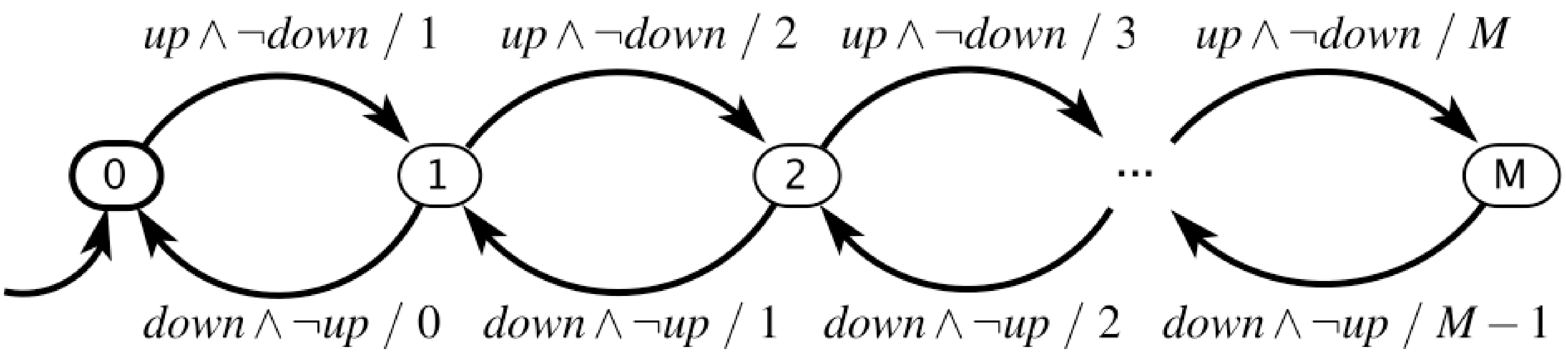


Guard g is specified using the predicate

$$up \wedge \neg down$$

which means that *up* has value $present$ and *down* has value $absent$.

Garage Counter Mathematical Model

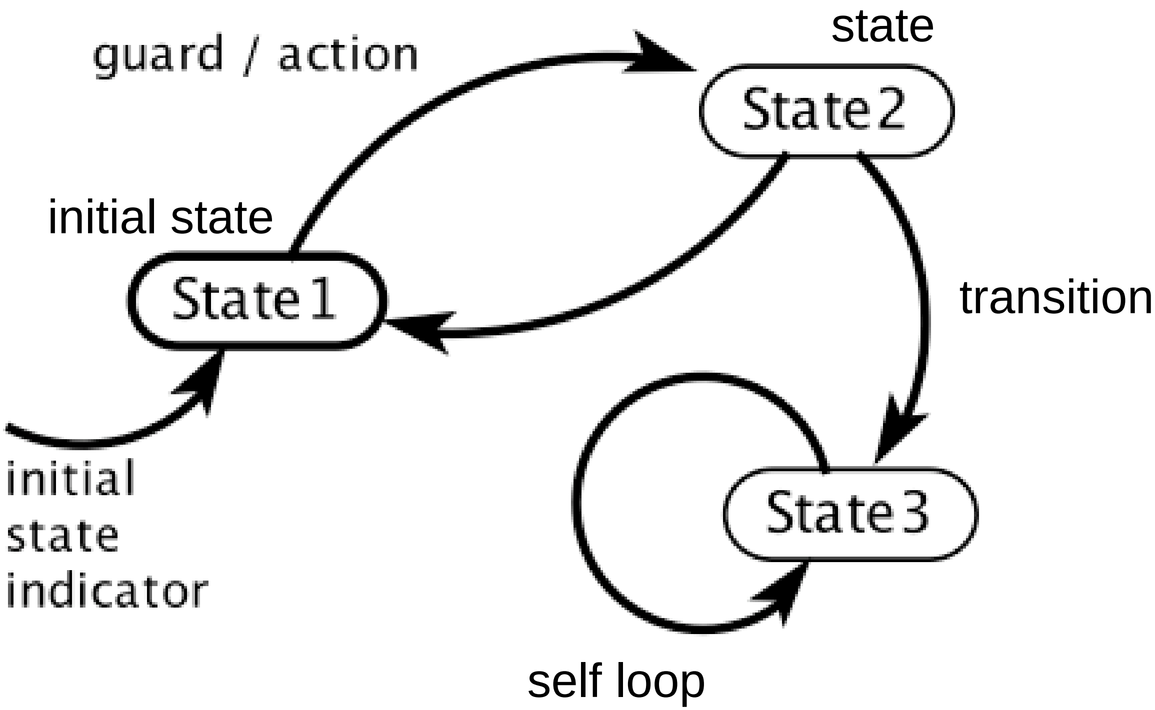


Formally: $(States, Inputs, Outputs, update, initialState)$, where

- $States = \{0, 1, \cdots, M\}$
- $Inputs$ is a set of input valuations
- $Outputs$ is a set of output valuations
- $update : States \times Inputs \rightarrow States \times Outputs$
- $initialState = 0$

The picture above defines the update function.

FSM Notation



Examples of Guards for Pure Signals

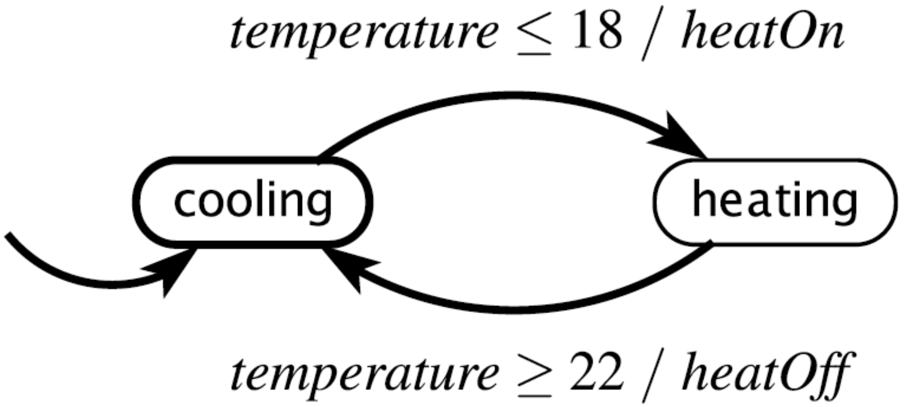
true	Transition is always enabled.
p_1	Transition is enabled if p_1 is $present$.
$\neg p_1$	Transition is enabled if p_1 is $absent$.
$p_1 \wedge p_2$	Transition is enabled if both p_1 and p_2 are $present$.
$p_1 \vee p_2$	Transition is enabled if either p_1 or p_2 is $present$.
$p_1 \wedge \neg p_2$	Transition is enabled if p_1 is $present$ and p_2 is $absent$.

Examples of Guards for Signals with Numerical Values

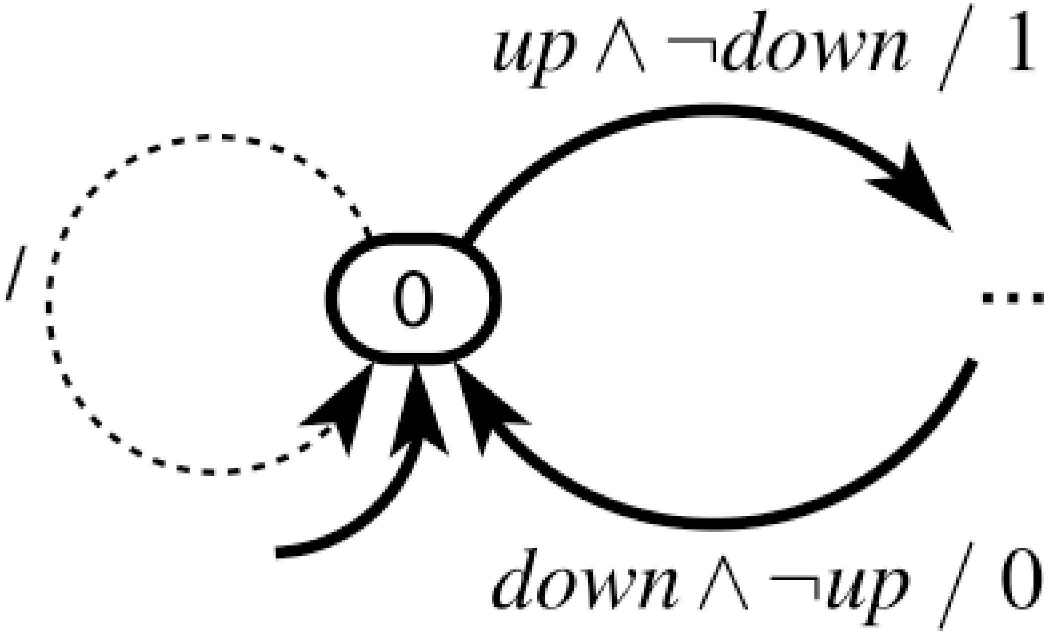
p_3	Transition is enabled if p_3 is *present* (not *absent*).
$p_3 = 1$	Transition is enabled if p_3 is *present* and has value 1.
$p_3 = 1 \wedge p_1$	Transition is enabled if p_3 has value 1 and p_1 is *present*.
$p_3 > 5$	Transition is enabled if p_3 is *present* with value greater than 5.

Example: Thermostat

input: $temperature : \mathbb{R}$
outputs: $heatOn$, $heatOff$: pure



More Notation: Default Transitions

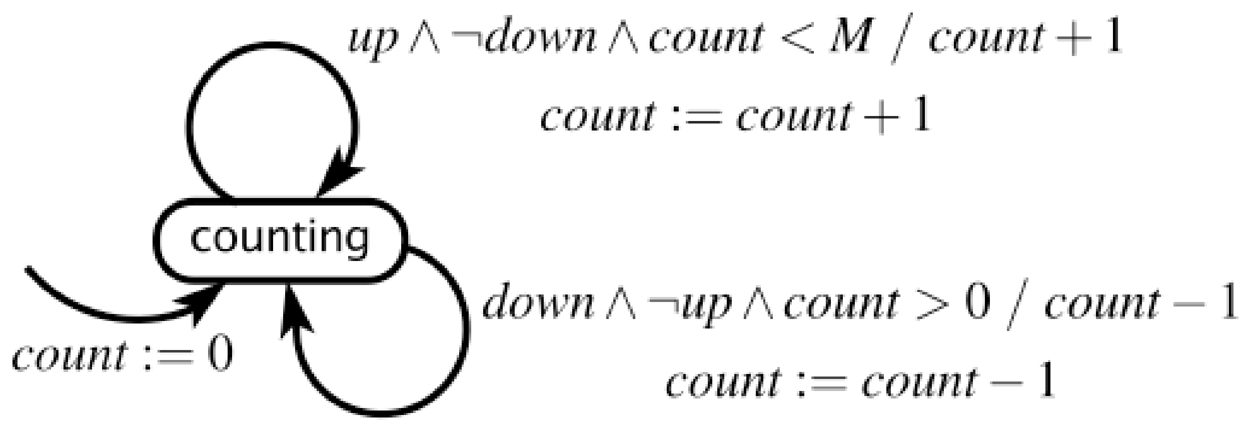


A default transition is enabled if no non-default transition is enabled and it either has no guard or the guard evaluates to true.

Extended State Machines

Extended state machines augment the FSM model with *variables* that may be read or written. E.g.:

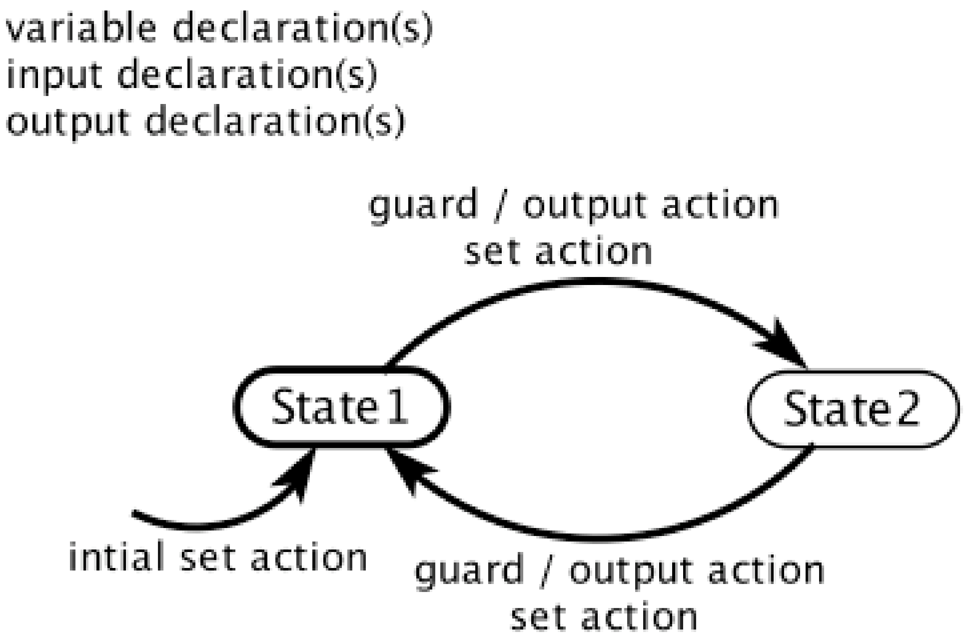
variable: $count \in \{0, \cdots, M\}$
inputs: $up, down \in \{present, absent\}$
output $\in \{0, \cdots, M\}$



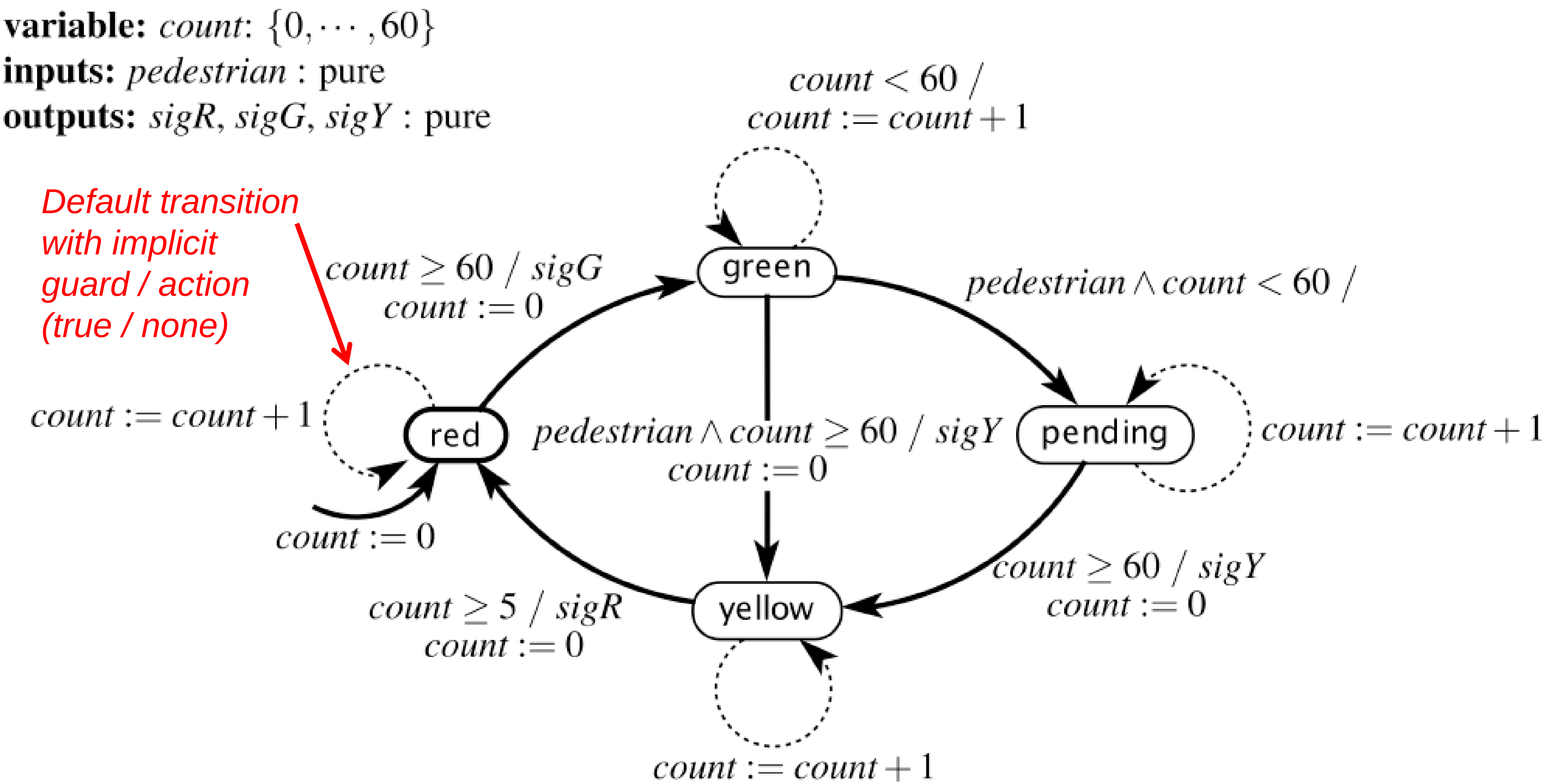
Question: What is the size of the state space?

General Notation for Extended State Machines

We make explicit declarations of variables, inputs, and outputs to help distinguish the three.



Extended state machine model of a traffic light controller at a pedestrian crossing:



- This model assumes one reaction per second
- (a *time-triggered* model)