# CP275: Formal Analysis and Control of Autonomous Systems

**Instructor:** Pushpak Jagtap

# Formal Analysis and Control of Autonomous Systems (CP274)

Pushpak Jagtap

- **Email:** pushpak@iisc.ac.in
- **Days:** T Th
- **Time:** 15:30–17:00
- **Venue**: in person
- **To arrange personal meeting:** email me OR message/call over Team

Prerequisites

- basic knowledge of differential equations, linear algebra, calculus, and linear control theory

Grading & Evaluation

- **Assignments:** 30%
- **mid-term tests:** 30%
- **Final exam:** 40%

**Credits:** 2:1

**Compiled on 2023-01-03 at 09:59:26**

# Labs

- Introduction to ROS
- Introduction to Motion Capture Systems
- Implementation of control algorithms for various complex task using different approaches
  - Symbolic Control
  - Control Barrier Functions
  - Funnel-based Control

**Instructors:**
Jeel Chatrola, Jay Bhagiya, Nilay Srivastav, Saharsh

# More on Grading and Evaluation

**Assignments (30%):**

- We will form a group of 2 or 3 students
- Each group will choose 1 research paper (mostly on application of learned concepts)
- At the end of Semester, each group will present and discuss it in the class

**Two minor tests (30%):**

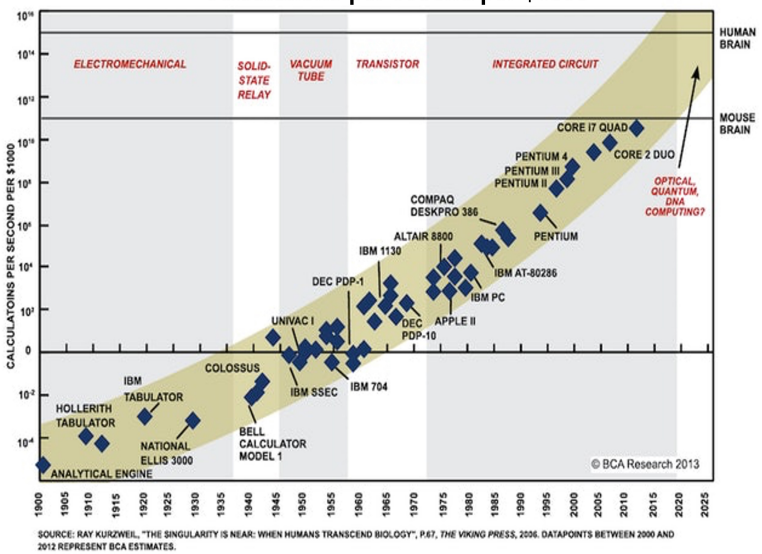
- Test-1: 10 true/false questions (negative marking) [The main purpose is to evaluate understanding of core concepts]
- Test-2: implementation of some tasks in lab

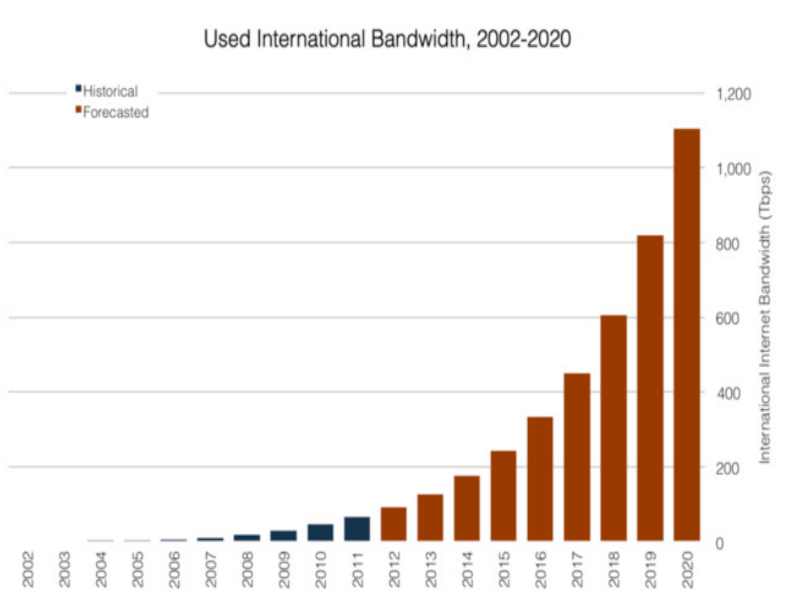
**Final Exam (40%)**

- Project based evaluation
- Group of 2-3 student will do literature review on specific topic and propose a novel idea/results.

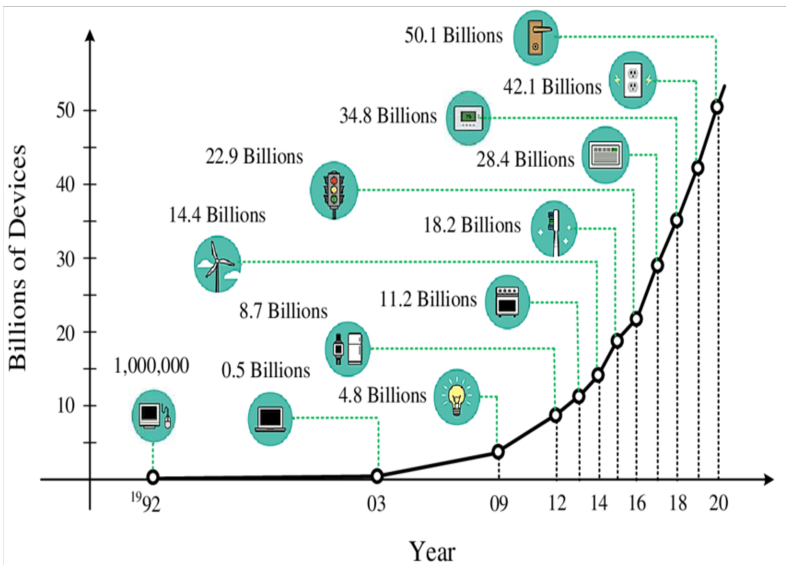
# Advances in computation and communication capabilities

**Calculations per second per $1000**

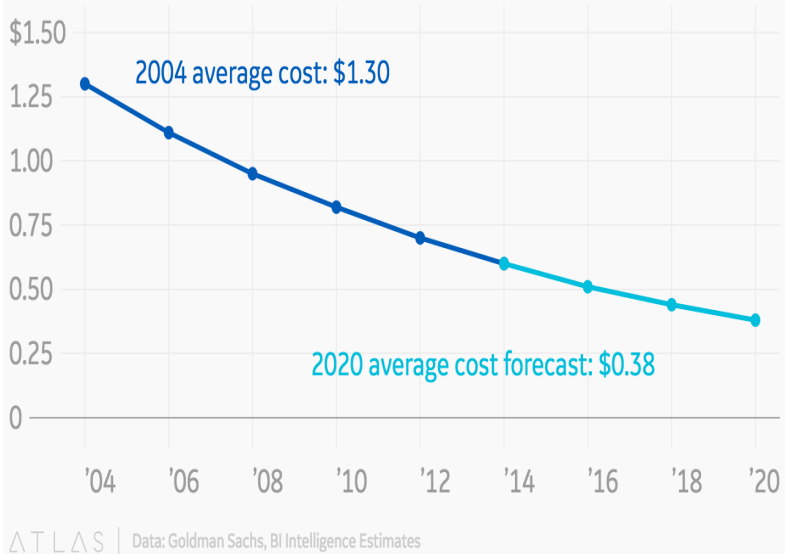


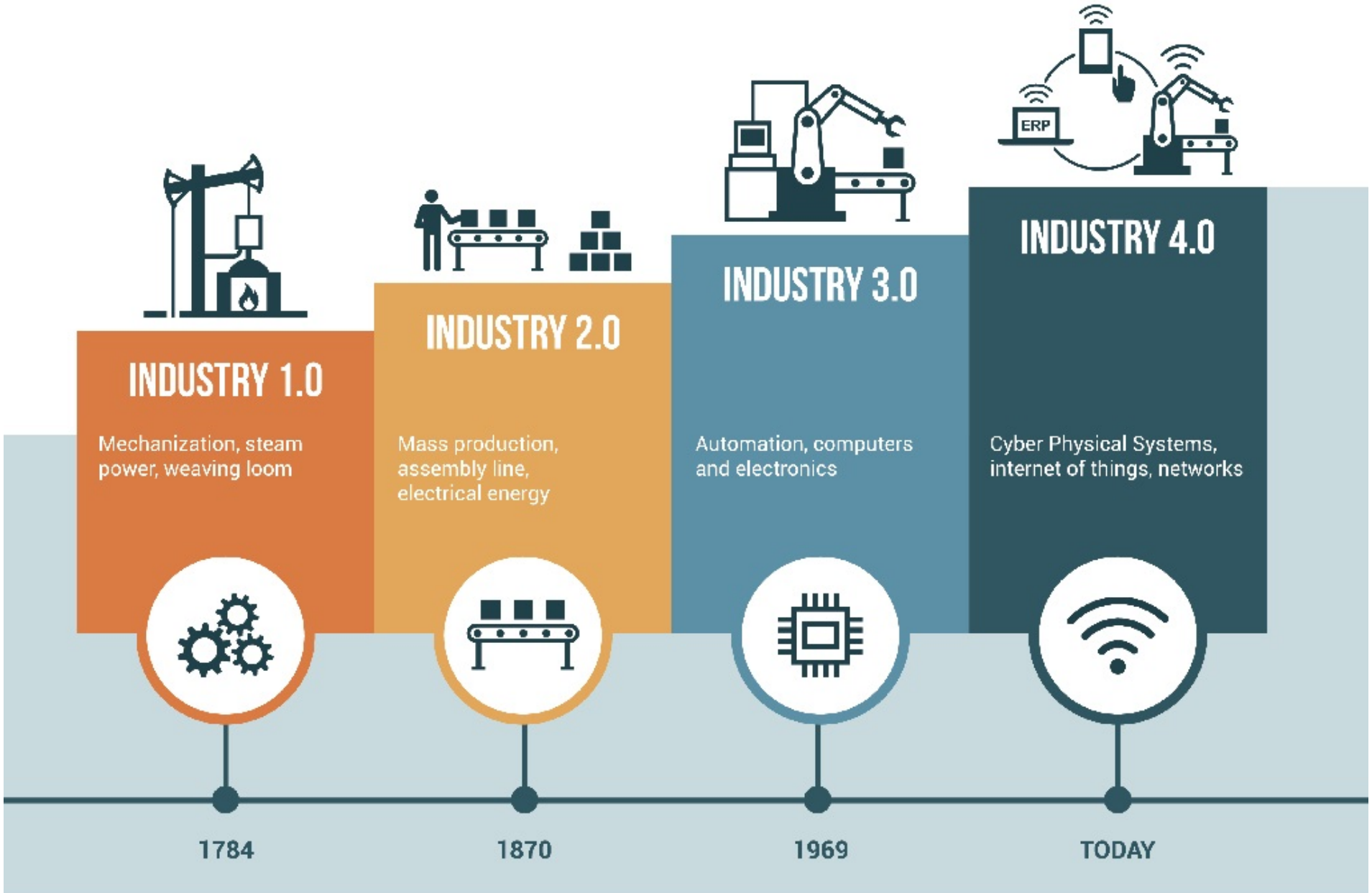

**Devices connected to Internet**




**The average cost of IoT sensors is falling**




*taken from Internet sources*


ERP
INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
1784
1870
1969
TODAY

# 1 Introduction

## 1.1 Autonomous Systems

## 1.2 Low-Level Specifications

## 1.3 High-Level Specifications

## 1.4 Formal Verification

## 1.5 Formal Synthesis

## 1.6 Comments

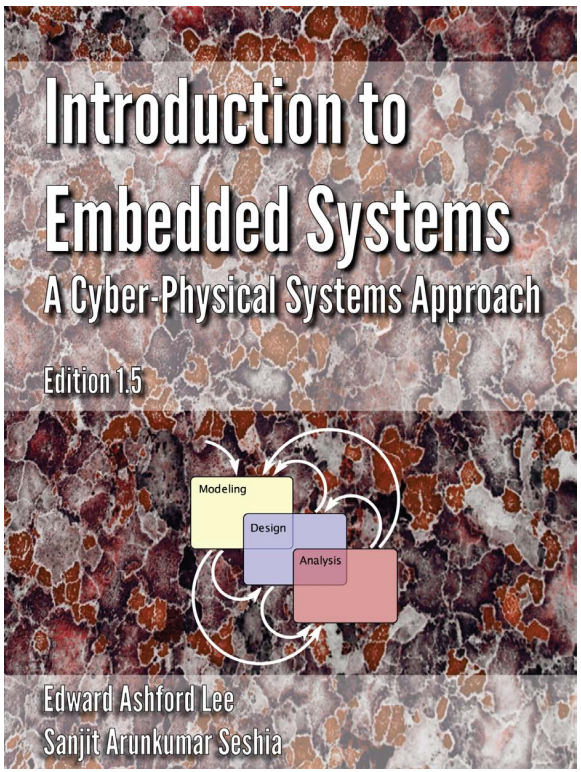
# 1.1 Autonomous Systems

## What is an autonomous system?

In this lecture we follow the interpretation of autonomous systems as embedded systems or cyber-physical systems according to

**E. Lee and S. Seshia**, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach, 2014*. Online available at www.leeseshia.org.

"A cyber-physical system (CPS) is an integration of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa."



**E. Lee**, *Computing foundations and practice for cyber-physical systems: A preliminary report. 2007*

"..., we may think of cyber-physical systems to comprise embedded systems (the information processing part) and the physical environment."

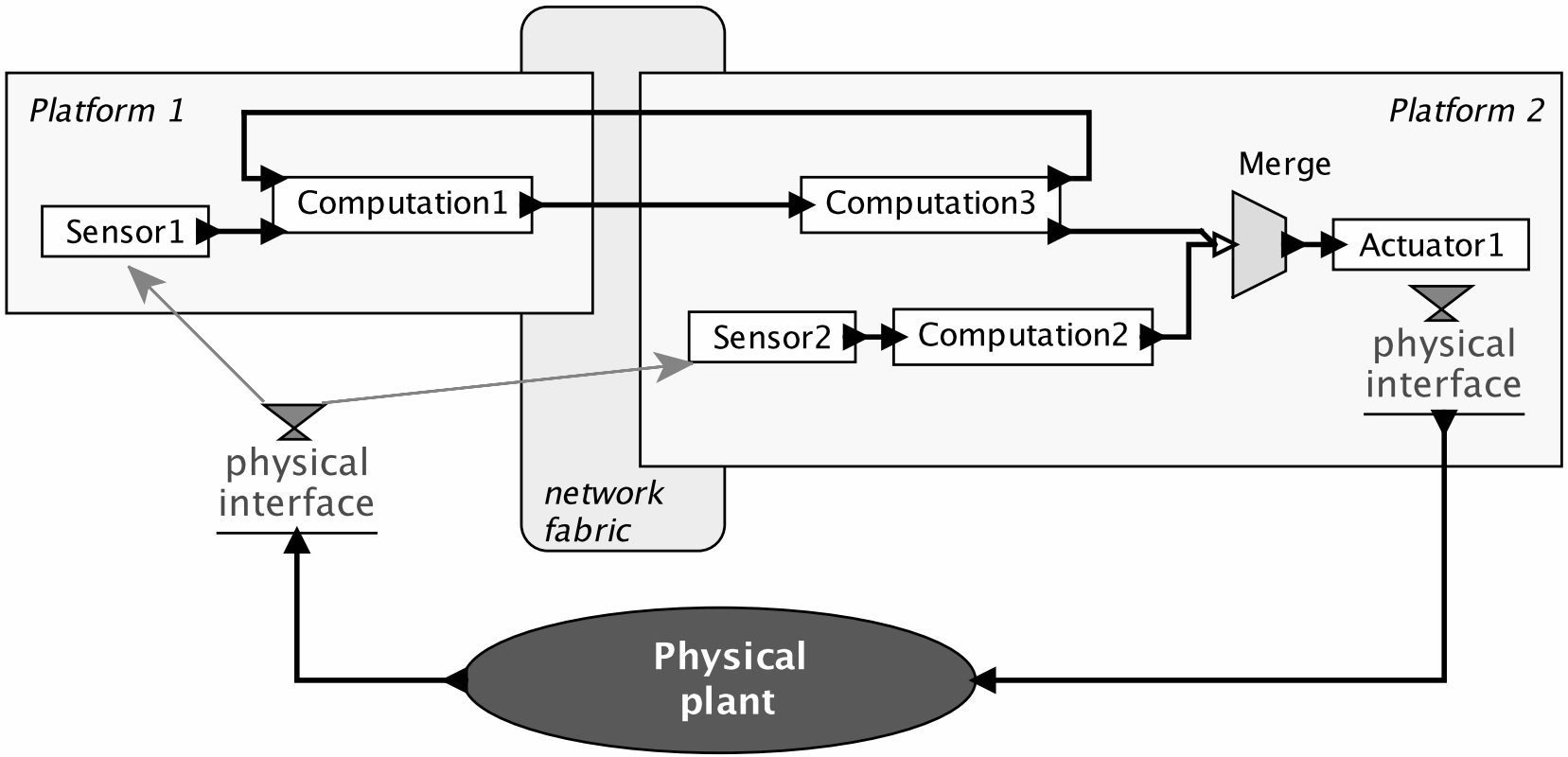



Figure: Common Structure of an autonomous/cyber-physical system. Source: LeeSeshia14

## Components of an autonomous/cyber-physical system

- Physical plant is the physical part of the cyber-physical system consisting e.g. mechanical/electrical parts; biological/chemical processes.
- Cyber part consists of digital sensors, actuators, embedded computers, software and communication infrastructure.

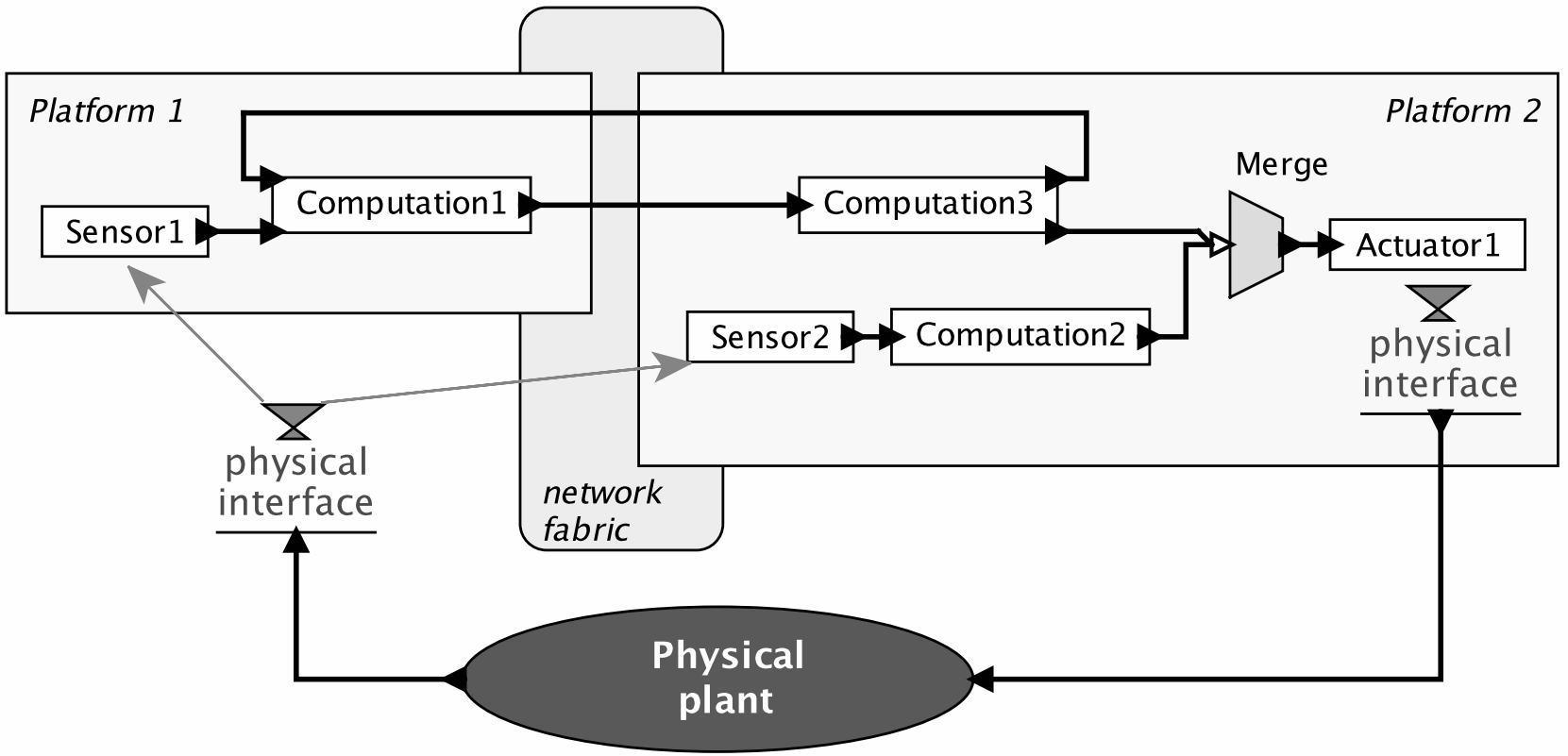



Figure: Common Structure of an autonomous/cyber-physical system. Source: LeeSeshia14

## Example: Adaptive Cruise Control

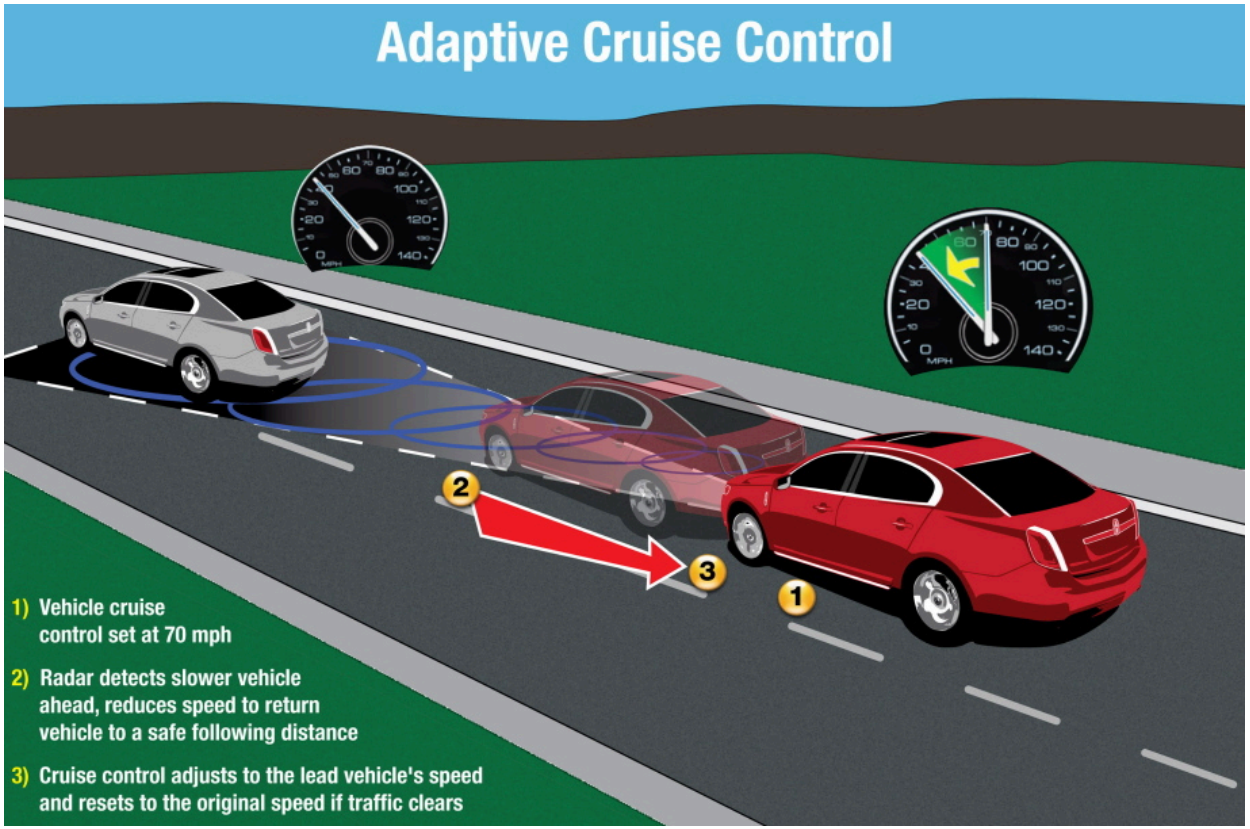


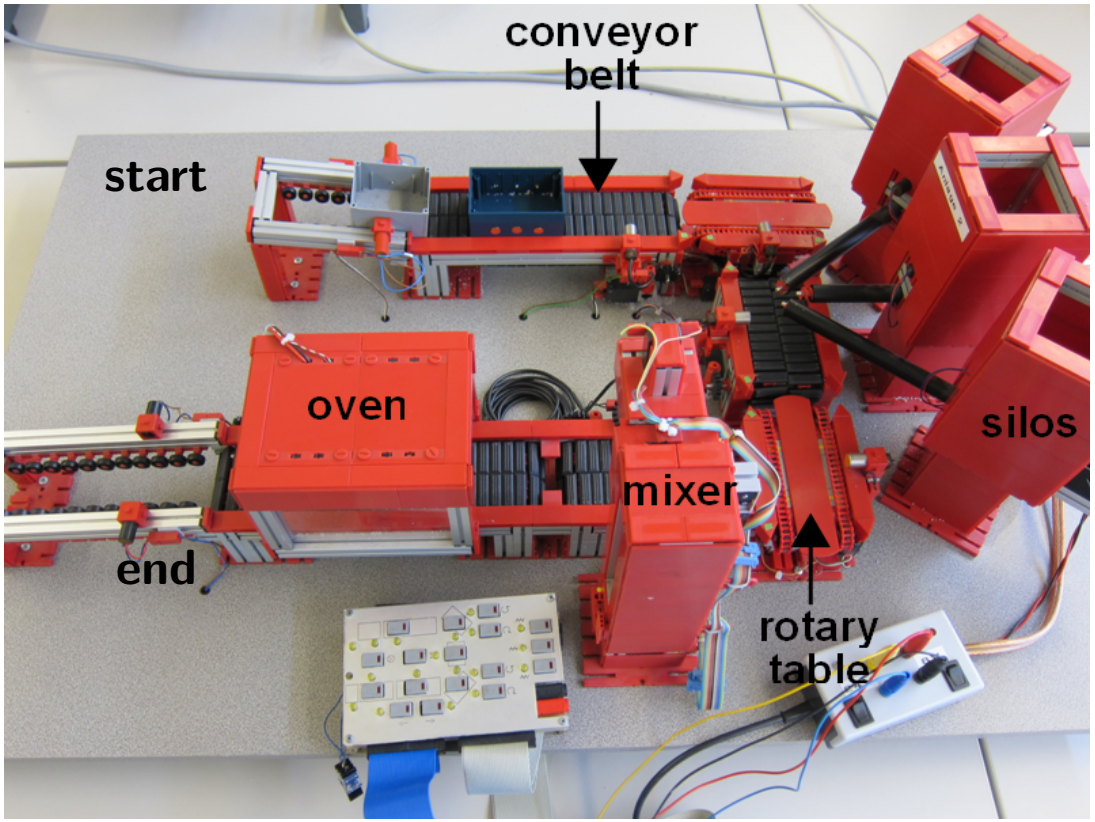

Image: http://blog.oakridgeford.com/

Different modeling frameworks

- Ordinary differential equations (e.g. $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$) or difference equations (e.g. $\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t))$) are used to describe the vehicle dynamics.
- Finite state machines are used to model the different modes of the system, such as "vehicle ahead" or "no vehicle ahead".

# Example: Production Lines

## Baking line



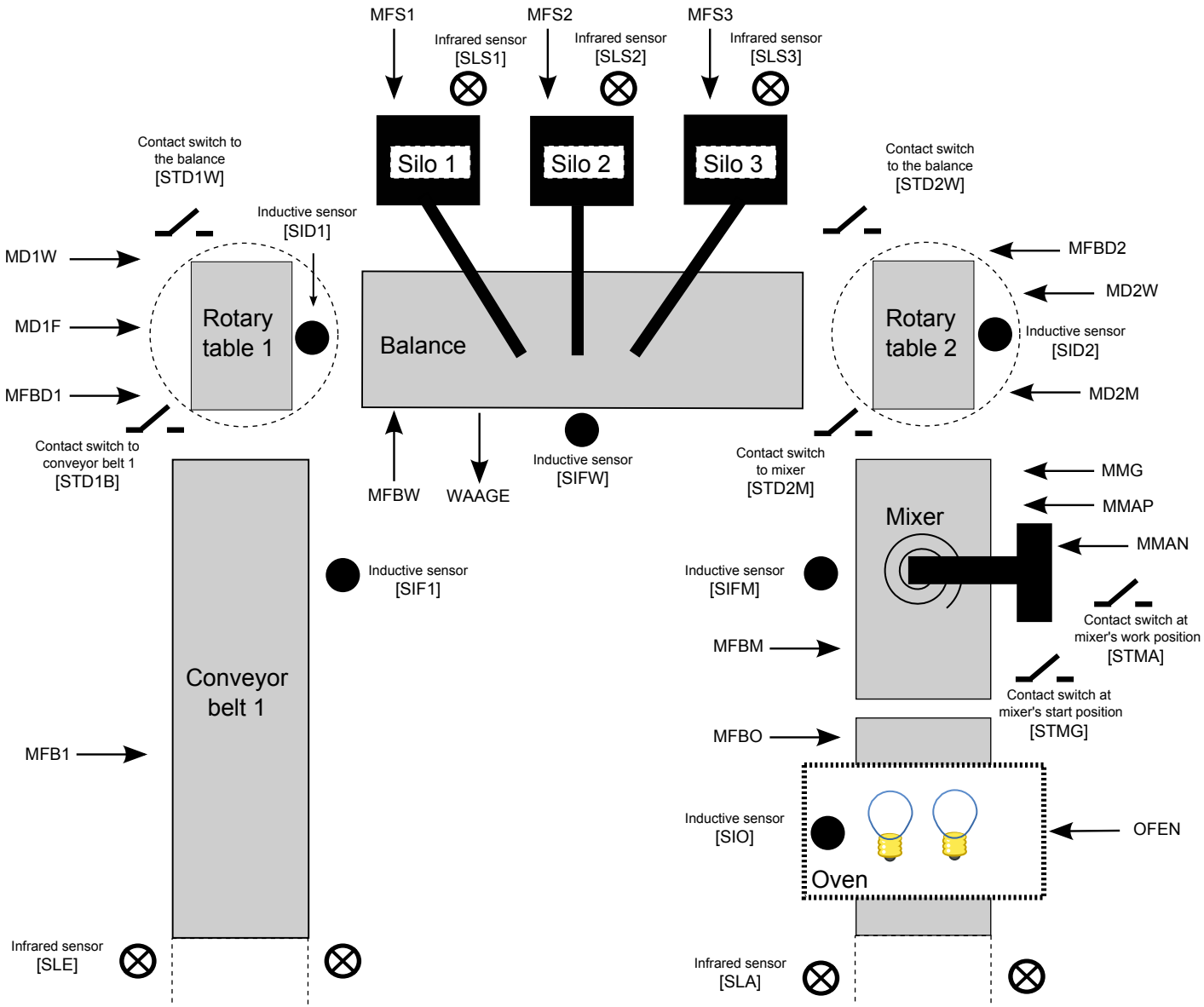

### Components

- Boxes, oven, mixer, silos, conveyor belts, rotatory tables,...

### Description

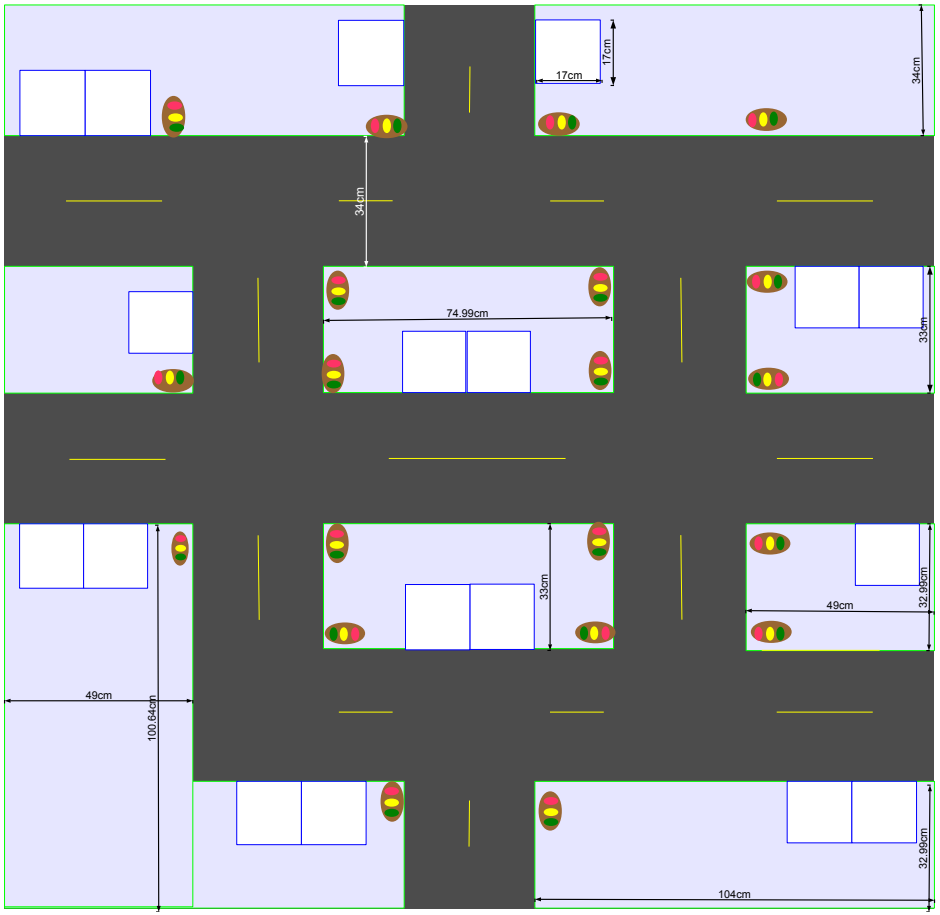
- Boxes are moved from start position to end position.
- First stage: filling the boxes with material.
- Second stage: mixing the granulate material.
- Third stage: baking the material at a certain temperature in the oven.

# **Baking line:** schematic overview with sensors and actuator signals



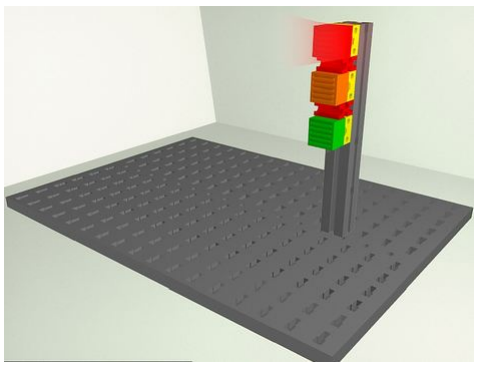
# Example: Mobile Robots in an Urban Environment

Road network with parking lots and traffic lights




Mobile robots

Traffic lights

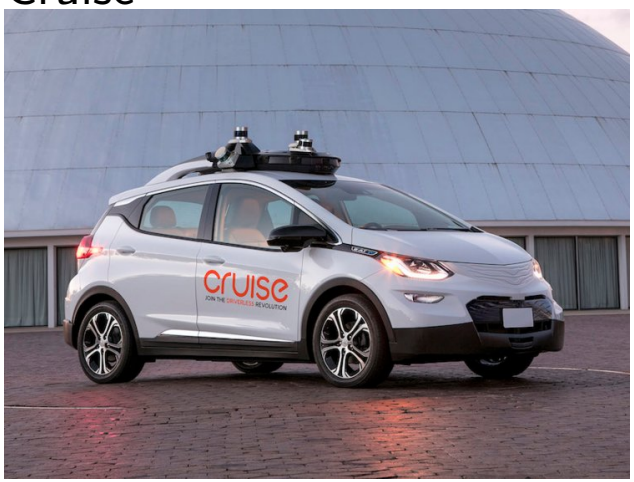


# Example: Self-Driving Cars

Waymo


Uber


Cruise




Nuro

## Autonomous systems are everywhere

- Avionics
- Railway systems
- Automotive
- Mobile communication
- Medical devices
- Robotics
- Buildings and home applications
- ...

## 1.2 Low-Level Specifications

We use stability (including tracking, synchronization) in general to express the low-level description of the expected system behavior.

### Definition

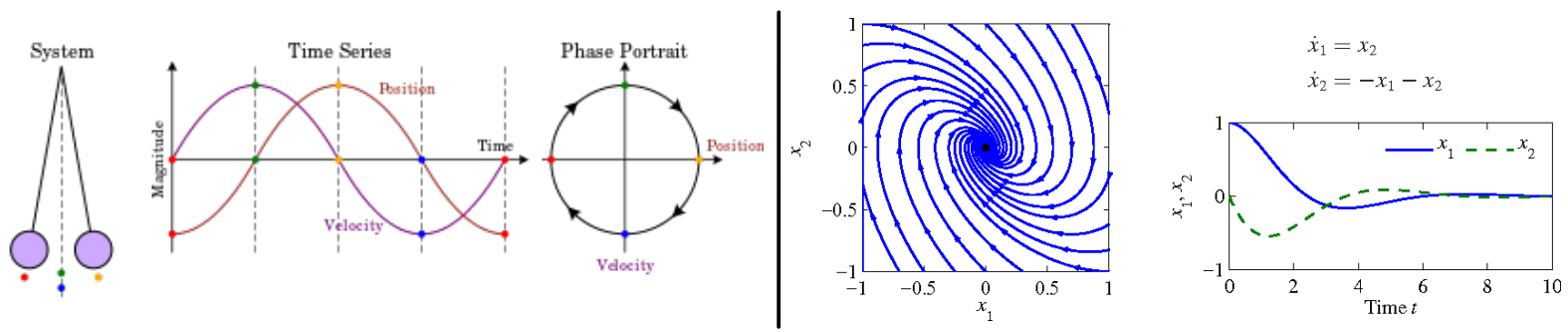
The nonlinear system $\mathbf{x}(t+1) = f(\mathbf{x}(t))$ is stable with respect to the origin (i.e. $x = 0$) if for each $\varepsilon > 0$ there exists $\delta > 0$ such that

$$\|\mathbf{x}(0)\| < \delta \implies \|\mathbf{x}(t)\| < \varepsilon, \quad \forall t \in \mathbb{N}.$$

### Definition

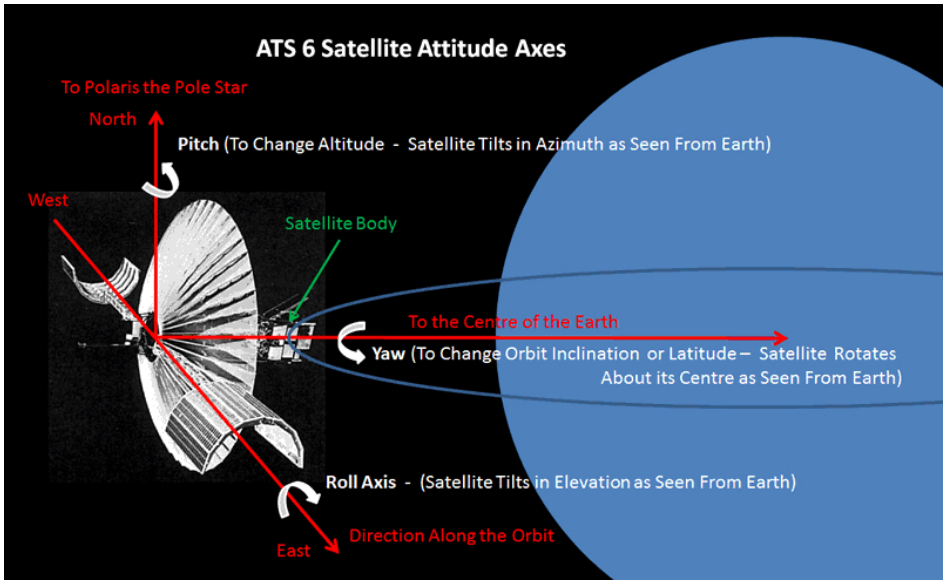
The nonlinear system $\mathbf{x}(t+1) = f(\mathbf{x}(t))$ is asymptotically stable with respect to the origin if it is stable and furthermore

$$\lim_{t \to \infty} \|\mathbf{x}(t)\| = 0.$$

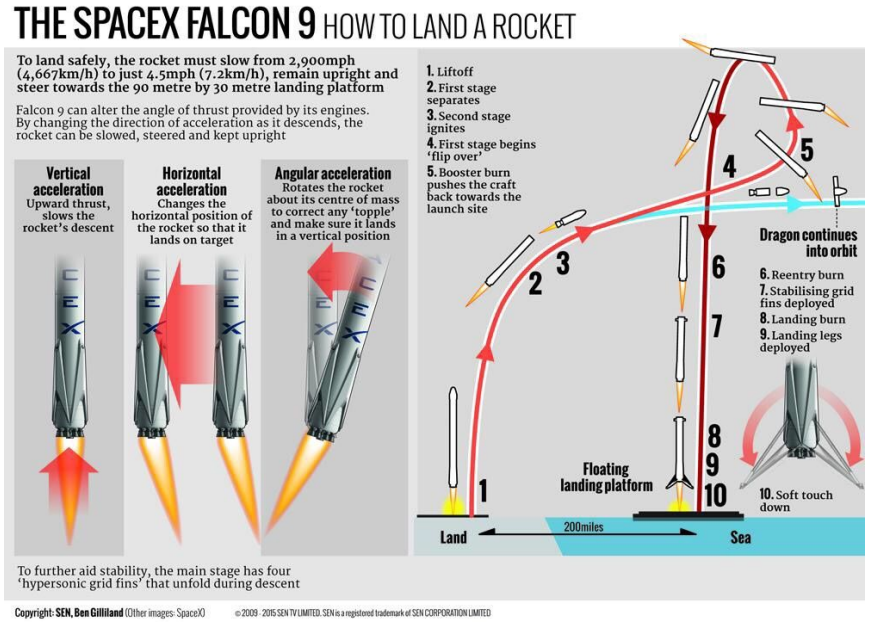


# Examples

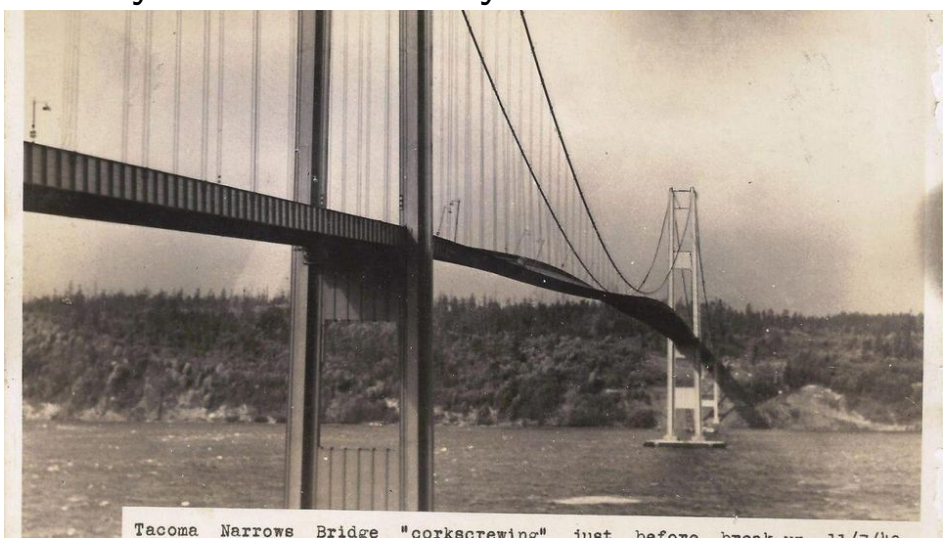
## Orbital Stability



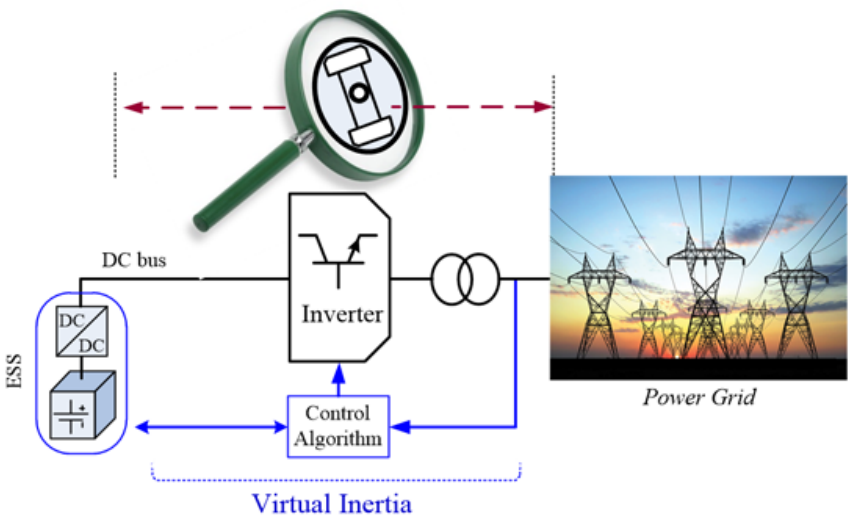

## Landing a rocket




## Aerodynamic Instability




## Frequency Stability



## 1.3 High-Level Specifications

We use linear temporal logic or automata on (in)finite strings to express the high-level description of the expected system behavior.

**Examples of high-level descriptions:**

- Safety properties $\varphi = \square \neg B$
  *The system should never enter the "bad" set $B$.*
- Reachability properties $\varphi = \lozenge T$
  *The system should eventually reach the "target" set $T$.*
- Persistence properties $\varphi = \lozenge \square T$
  *The system should eventually reach the "target" set $T$ and stay in it forever onwards.*
- Recurrence properties $\varphi = \square \lozenge T$
  *The system should repeatedly reach the "target" set $T$.*
- Reactive properties $\varphi = \square \lozenge A \rightarrow \square \lozenge G$ or $\varphi = \lozenge \square A \rightarrow \lozenge \square G$
  *If the assumption $A$ is repeatedly true, then the guarantee $G$ should be repeatedly true*
  *or*
  *if the assumption $A$ is eventually true and stays true forever onwards, then the guarantee $G$ should be eventually true and stay true forever onwards.*

## Example: Adaptive Cruise Control

Desired behavior

$$\big(\Box(\text{distance} > 0)\big) \wedge \big(\Diamond\Box(\text{lead car}) \rightarrow \Diamond\Box(\text{desired distance})\big) \wedge$$
$$\big(\Diamond\Box(\text{no lead car}) \rightarrow \Diamond\Box(\text{desired velocity})\big)$$

## Example: Baking Line

The desired specs:

- The mixing time should be 10 seconds for the first type of boxes. On the other hand, the second type of boxes should have a mixing of 15 seconds.

$$\Box(\text{box 1 in mixing position} \rightarrow \text{mixing for 10 sec}) \wedge$$
$$\Box(\text{box 2 in mixing position} \rightarrow \text{mixing for 15 sec})$$

- The baking temperature should be between 26°C to 28°C independently of the box type. Boxes should only enter the oven when this temperature has been reached.

$$\Box(\text{box in oven} \rightarrow \text{temperature} \in [26, 28]^\circ\text{C})$$

## Example: Mobile Robots

Parking at specific destination

$$\Diamond(\text{robot in parking lot 1})$$

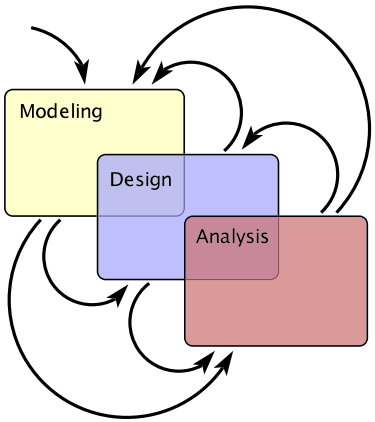
Patrolling between two positions with a traffic light in between

$$\Box\Diamond\,\text{green} \rightarrow \big(\Box\Diamond(\text{robot at pos1}) \wedge \Box\Diamond(\text{robot at pos2})\big)$$

# 1.4 Formal Verification

## Model-based design paradigm: specify-design-verify-refine

1. Specification/Requirement
   What the system is supposed to do?
2. Modeling
   Description of what the system actually does.
3. Design
   Structured creation of artifacts to modify the system behavior.
4. Verification
   Certify that the system does what it is supposed to do.
5. Refinement
   If the verification step fails go back and refine your design.




Source: LeeSehisa2014, www.leeseshia.org

**Verification step: How to assess system correctness?**

**Formal Methods:** Mathematically based techniques for the specification, development, and verification of software and hardware systems.

**Testing:** Simulation/execution based inspection of the system correctness.

**Example: Asymptotic Stability**

$$\mathbf{x}(t+1) = f(\mathbf{x}(t))$$

Suppose $f(0) = 0$: show asymptotic stability of $f$ with respect to the origin

**Formal Verification**

Find continuous function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ with

$$V(x) = 0 \text{ iff } x = 0$$

$$\forall x \in \mathbb{R}^n \setminus \{0\} \quad V(f(x)) - V(x) < 0.$$

**Testing**

Simulate the system for different initial conditions $X_0 = \{x_0, x_1, x_2, \ldots, x_n\}$

$$\mathbf{x}(t+1) = f(\mathbf{x}(t)), \quad \mathbf{x}(0) \in X_0$$

and observe $\mathbf{x}(t)$ as $t \to \infty$.

## Formal Verification vs Testing

### Formal Verification

- provides a mathematical proof of the absence of errors relative to the specifications and the model;
- uses formal specifications, whose formulation require highly qualified engineers;
- is computationally expensive and often fails in a naive application for real-world systems.

### Testing

- cannot guarantee absence of errors;
- requires less mathematical skills;
- is computationally cheap.

**E. W. Dijkstra** "Program testing can be used to show the presence of bugs, but never to show their absence!"

## Formal Verification vs Testing

### Formal Verification

- provides a mathematical proof of the absence of errors relative to the specifications and the model;
- uses formal specifications, whose formulation require highly qualified engineers;
- is computationally expensive and often fails in a naive application for real-world systems.

### Testing

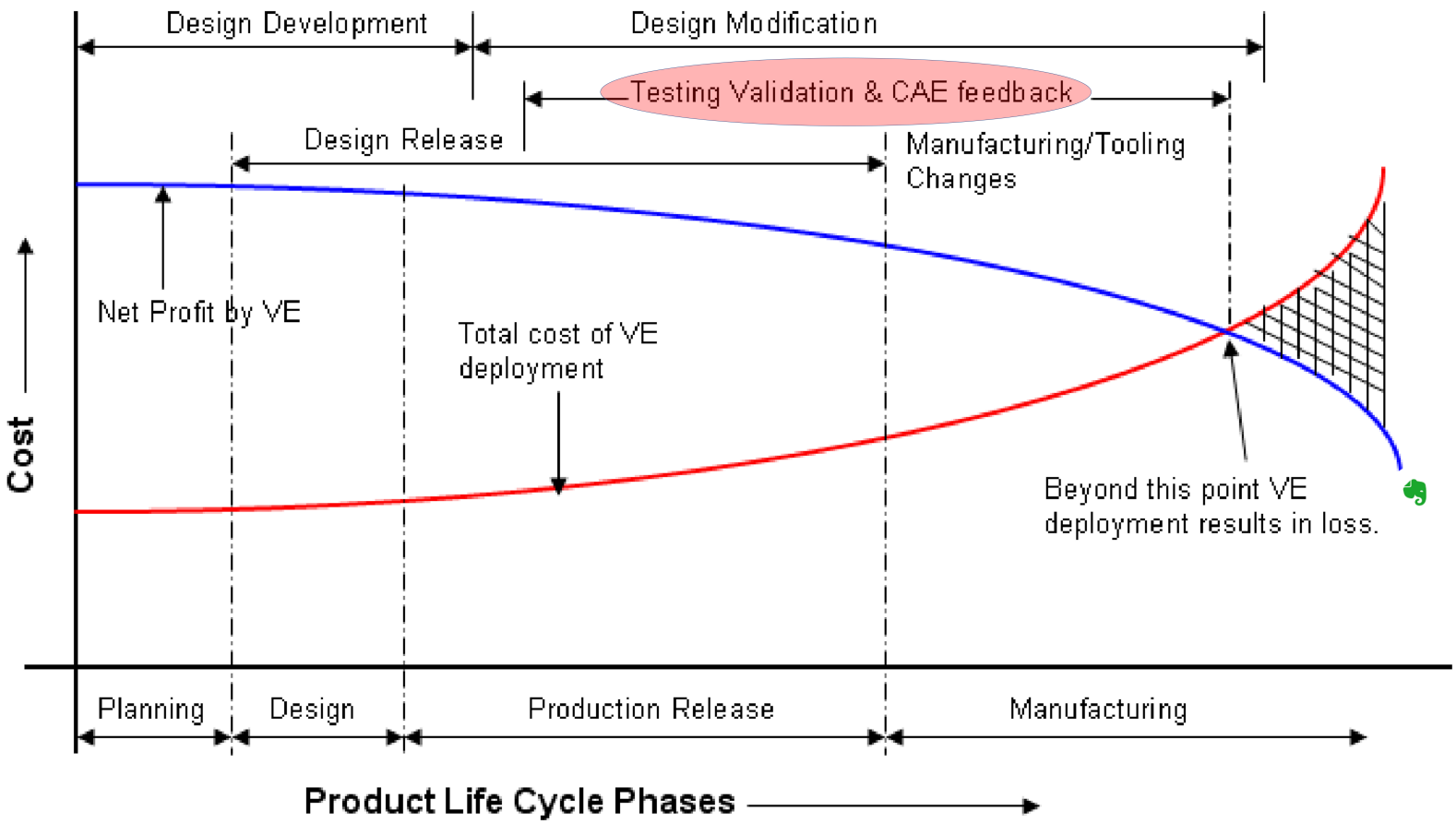
- cannot guarantee absence of errors;
- requires less mathematical skills;
- is computationally cheap.

**E. W. Dijkstra** "Program testing can be used to show the presence of bugs, but never to show their absence!"

**Autonomous Driving** [N. Kalra, S.M. Paddock (2016) More info here!]:

- 440 million km to demonstrate better performance that humans (95% confidence)
- 12.5 years with 100 test vehicles continuously driving
  - Costly, time-consuming and still not safe

# Why should we care about formal verification?



**Why should we care about formal verification?**

A formal proof of system correctness is absolutely necessary for safety-critical, highly reliable systems!

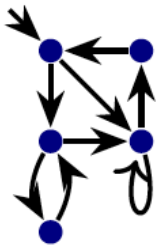
**Negative examples**

- 1996: Ariane 5 rocket destroyed, software defect due to conversion of a 64-bit floating point to 16-bit signed integer value. More info here!
- 2013: Honda had to recall 344,000 minivans, due to software defect that causes harsh application of brake without any driver action. More info here!
- 2018: Self-driving Uber car that struck and killed an Arizona woman More info here!
- ...

**Positive examples**

- 2012: Curiosity lands on Mars with verified code using `SPIN` model checker. **Good read:** *G. Holzmann, Mars Code, CACM, 2014:*
- 2017: Boeing "Unmanned Little Bird" helicopter with formally verified algorithms successfully completed mission in presences of mid-flight cyber attacks More info here!

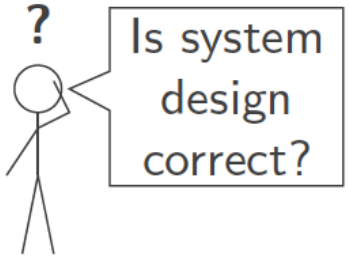
## Formal Verification/Model Checking



Finite state transition system

System

Abstract Model




Requirements

Specification

$\Box(request \rightarrow \Diamond enter\_crit)$

Temporal logic formula

**Model Checking**
**Check if model satisfies specification**

No + Counterexample

Yes

## 1.5 Formal Synthesis

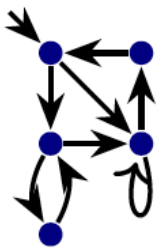
### What is formal synthesis?

- Formal synthesis is a methodology that mechanizes the design step and integrates the verification step in one unifying procedure.
- A new model-based design paradigm: "specify-synthesize-refine", also known as "correct-by-construction" synthesis.
- "Synthesis, if successful, avoids many manual design steps."

### The conceptual synthesis problem

*Given a system $S$ and a specification $\varphi$, find a controller $C$, so that the composed system $C \times S$ provably satisfies $\varphi$.*

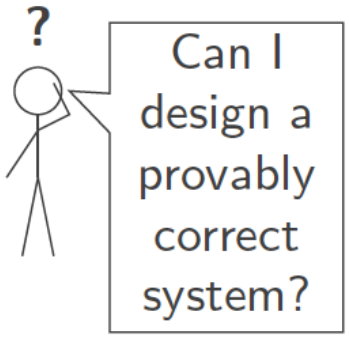
## Formal Synthesis



Finite state transition system

System

Abstract Model




Requirements

Specification

$\Box(request \rightarrow \Diamond enter\_crit)$

Temporal logic formula

**Synthesis**

**Synthesize a controller enforcing the specification on the system**

A provably-correct controller

## Example: Asymptotic Stabilization of a Linear System

1. Specification/Requirement

   The state of the system should converge to zero as time goes to infinity (a.k.a. asymptotic stability).

2. Modeling

   $$\mathbf{x}(t+1) = A\mathbf{x}(t) + B\mathbf{u}(t)$$

   with $\mathbf{x}(t) \in \mathbb{R}^n$, $\mathbf{u}(t) \in \mathbb{R}^m$.

3. Synthesis

   If matrix $[B\ AB\ A^2B\ \ldots\ A^{n-1}B]$ is full rank, we design a controller $\mathbf{u}(t) = K\mathbf{x}(t)$ by

   ```
   >> K=dlqr(A,B,eye(n),eye(m));
   ```

   which guarantees that $(A - BK)$ is Hurwitz (the composition of the system and controller is asymptotically stable!).

4. Refinement

   If matrix $[B\ AB\ A^2B\ \ldots\ A^{n-1}B]$ is not full rank, we modify the system design.

## Outline

1. Introduction
2. Modeling

   Difference equations, hybrid automata, labeled transition systems, behavior, parallel, serial and feedback composition
3. Low-Level Specifications

   Stability
4. High-Level Specifications

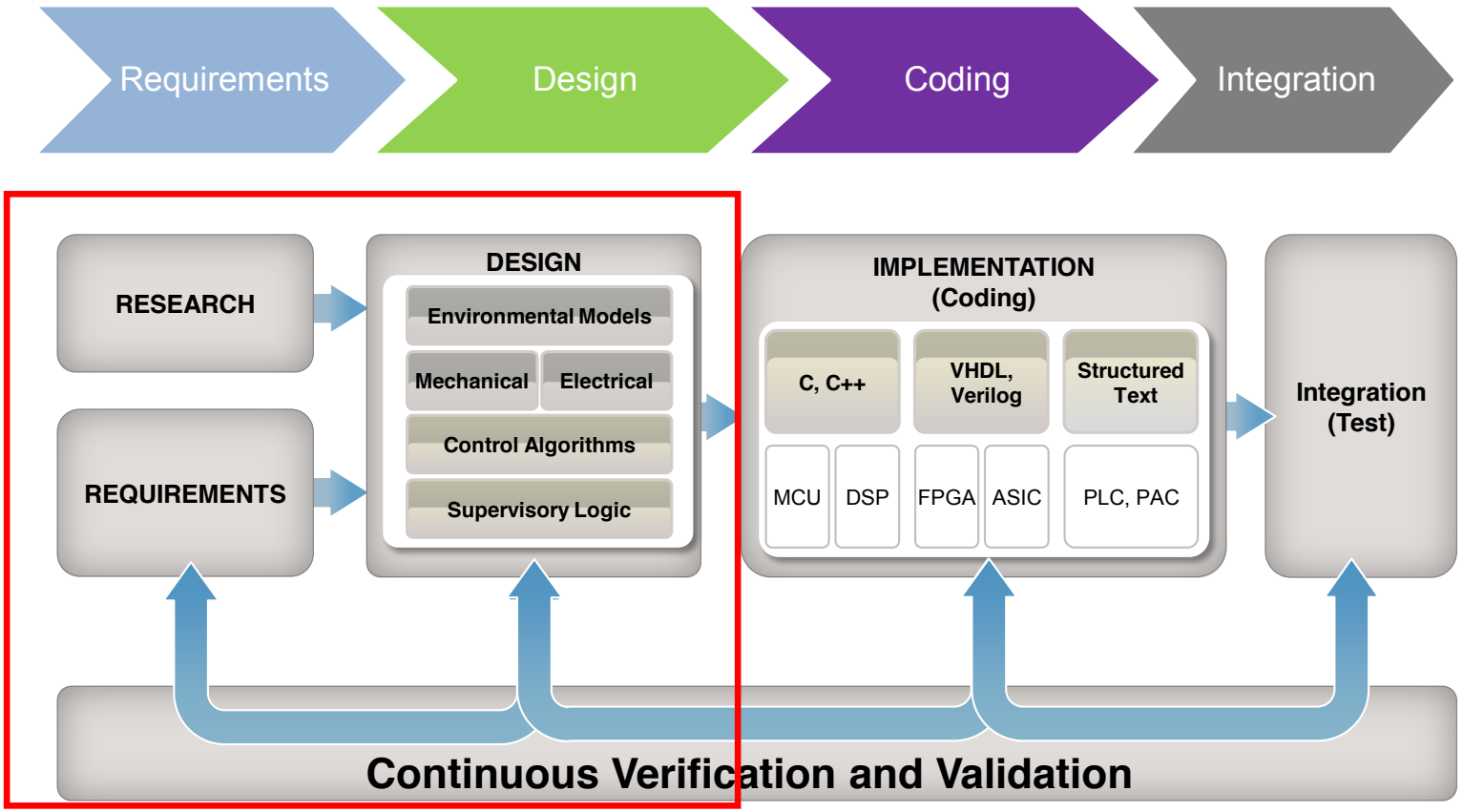
   Invariance, reachability, linear temporal logic, signal temporal logic
5. Formal Verification

   Lyapunov theory, reachability analysis, barrier certificate, model checking
6. Formal Synthesis

   Abstraction-based synthesis, control barrier certificate, funnel-based control, sampling-based motion planning

# 1.6 Comments

## The big picture: our focus




Jay Abraham, Mathworks

## The big picture: we do not cover

- Actuator and sensor models
- Embedded processor design

  Processor hardware, I/O hardware, processor types, memory models, communication buses.
- Real-time analysis/scheduling

  Response time analysis, worst-case execution time, multi-tasking.
- Concurrency

  Models of concurrent programs, synchronous reactive language.
- Embedded code generation

  Floating point, fixed point numbers, data types.

## Why you should be excited about this lecture!

- Formal methods for autonomous systems with mixed continuous and discrete dynamics is a hot topic in industry.
  - Many companies like WIPRO, TCS, Siemens, Toyota, Waymo, Zoox, GM Cruise, Nuro, Denso, Robert Bosch GmbH are hiring people with expertise on formal methods for hybrid systems.
- Formal verification and controller synthesis for mixed continuous and discrete systems is subject of current research.
  - Lots of opportunities for you to contribute in terms of an undergraduate and graduate theses, etc.

- Mature tools for formal verification of software.

  `SPIN`, `CPAchecker`, `nuSMV`, `BLAST`, `Polyspace`,...
- Academic tools for formal verification of hybrid systems.

  `PHAVer`, `SpaceX`, `Flow*`, `C2E2`, `JuliaReach`, `PIRK`...
- Academic synthesizers from LTL specification for discrete systems.

  `Acacia`$^+$, `Unbeast`, `Anzu`, `Lily`, `Slugs`,...
- Proof-of-concepts-level tools for formal synthesis of continuous/hybrid systems.

  `SCOTS`, `QUEST`, `pFaces`, `LTLMoP`, `TuLiP`, `AMYTISS`, `OmegaThreads`.

## After this lecture you should

- be able to create a formal model of an autonomous system that is useful for the design task;
- be able to express various system properties in a rigorous and precise manner using in particular temporal logic;
- understand the basic algorithms and notions underlying the verification for stability, invariance, and reachability;
- understand the basic algorithms and notions underlying the controller synthesis to enforce stability, invariance, and reachability;
- be able to show the correctness of the closed-loop systems;
- be able to use the learned topics and leverage them to verify or synthesize controllers for sample autonomous systems using existing tools.

## You are expected

- to actively participate in the class;
- to have some basic knowledge of linear algebra and differential equations;
- to think abstractly (the number of "real-world" examples are kept at a minimum);
  The sections on Modeling and Specifications are motivated by real-world examples. In the later sections, we will mostly work with toy examples for the sake of better illustration of proposed schemes.
- to like math (theorems and proofs) up to some degree;
- to be able to read math

$$F : X \times U \to 2^X, \quad K \subseteq X, \quad G(K) := \{x \in X \mid \exists u \in U : F(x, u) \subseteq K\}$$