Introduction to Intelligent Vehicles
[10. Verification]

Chung-Wei Lin

cwlin@csie.ntu.edu.tw

CSIE Department

National Taiwan University

Lecture Plan

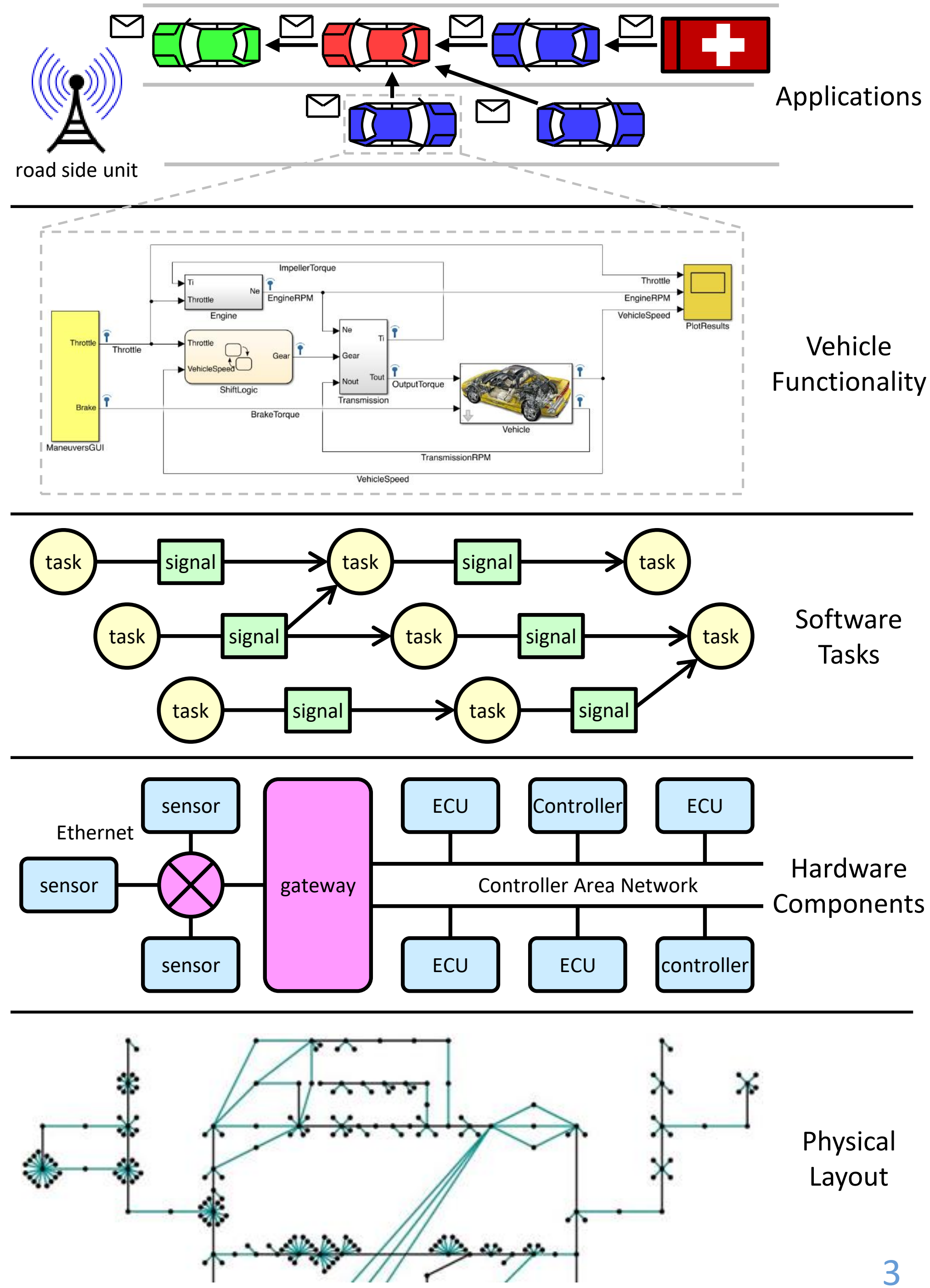
- Four parts in sequence
  - [Part 1] Preliminary
  - [Part 2] Applications
  - [Part 3] Intelligent Technology
  - **[Part 4] Advanced Topics**

Lecture Plan

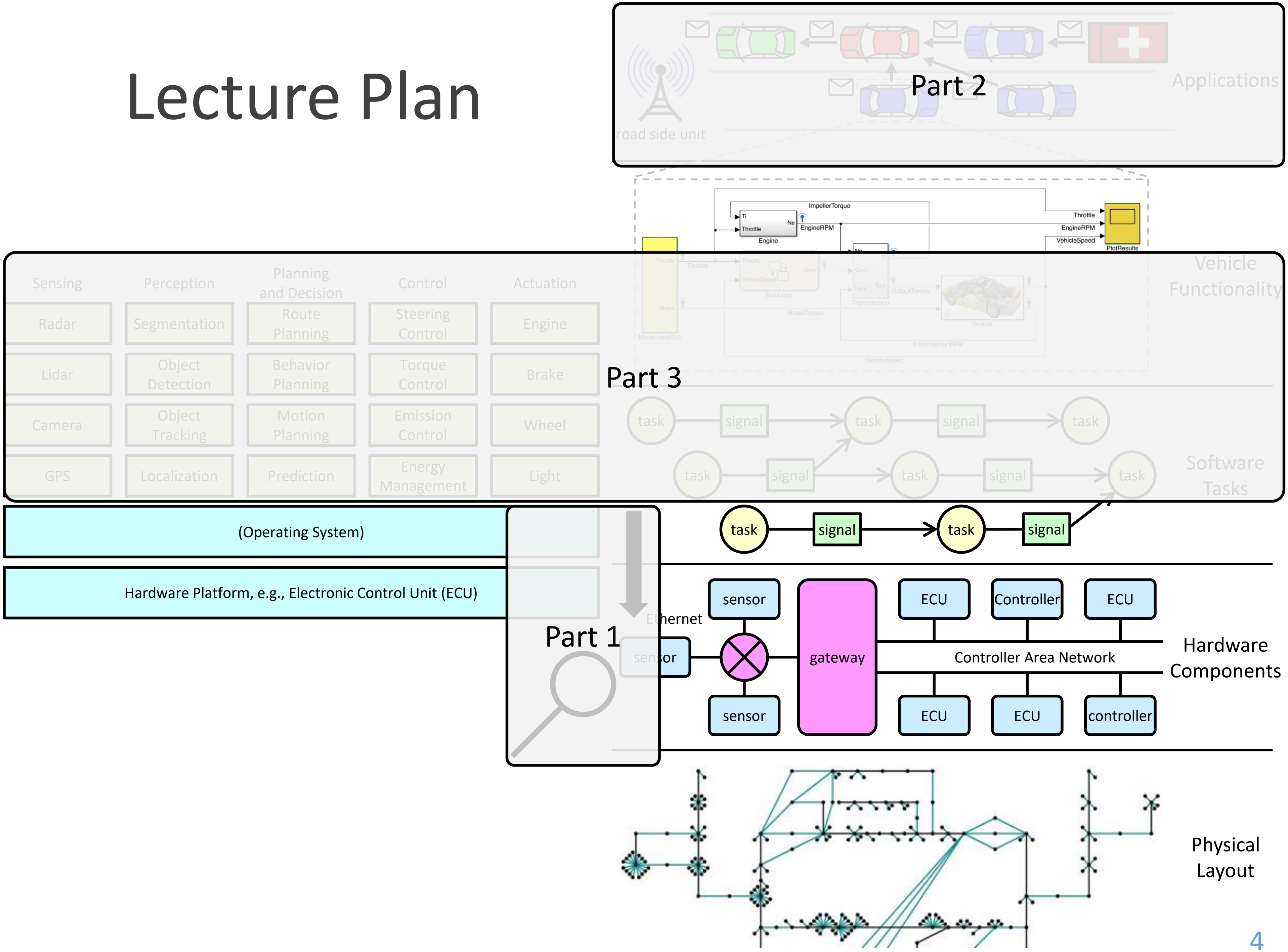
Sensing	Perception	Planning and Decision	Control	Actuation
Radar	Segmentation	Route Planning	Steering Control	Engine
Lidar	Object Detection	Behavior Planning	Torque Control	Brake
Camera	Object Tracking	Motion Planning	Emission Control	Wheel
GPS	Localization	Prediction	Energy Management	Light

(Operating System)

Hardware Platform, e.g., Electronic Control Unit (ECU)

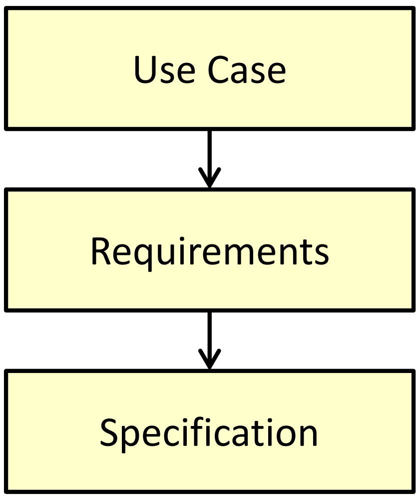


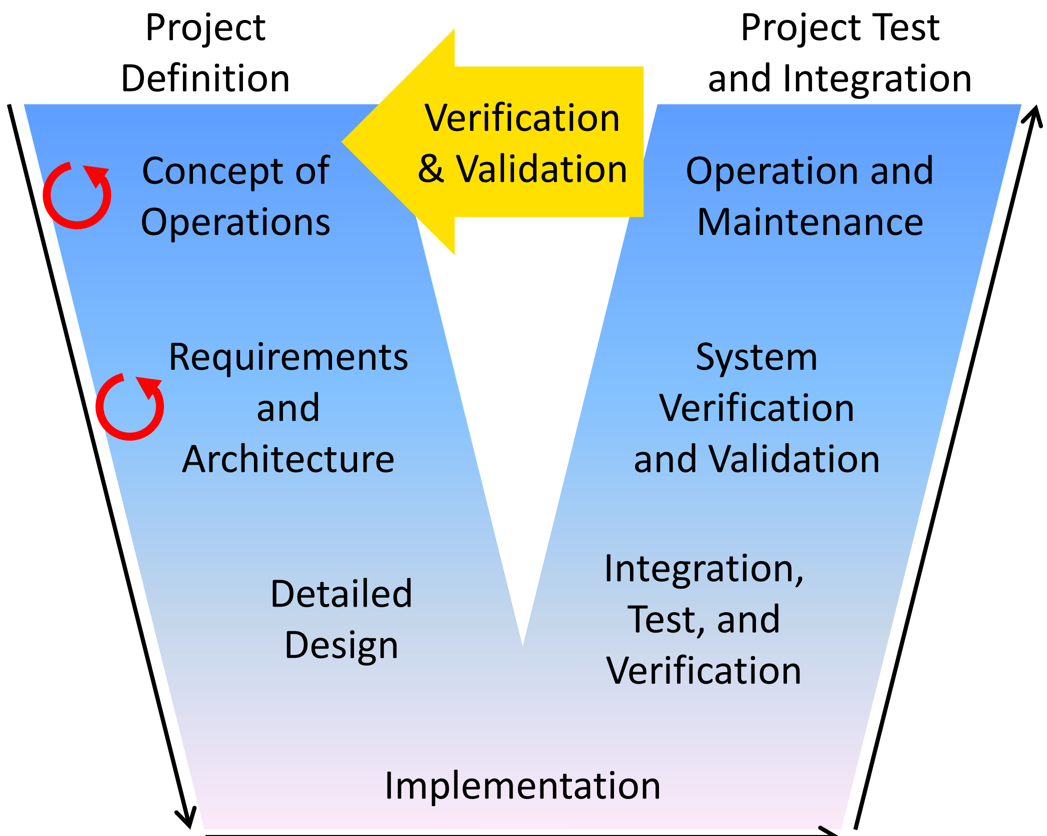
Lecture Plan



V Model

"因為自動飲料機
而延畢的那一年"





https://en.wikipedia.org/wiki/V-Model_(software_development)

Fundamental Challenges

- **How do you know**
  - **Your design is correct, i.e., satisfying its requirements?**
    - Including the compatibility of sub-systems after decomposition and composition
  - **Your implementation is correct, i.e., satisfying its specification?**

- Example
  - You need to have a correct algorithm and a correct implementation to complete the sorting task
- Goals
  - Consider different design metrics
    - Safety, reliability, robustness, performance, security, etc.
  - Assist system designers for early design decisions
    - More efficient process

Approaches

- Mathematical analysis, e.g., timing analysis
- Verification
  - The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition
    - From the Project Management Body of Knowledge (PMBOK) guide
    - It is often an internal process
- Validation
  - The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders
    - From the PMBOK guide
    - It often involves acceptance and suitability with external customers
- Simulation
- Testing

Formal Verification

- The act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property [Wikipedia]
  - Using formal methods of mathematics
  - Providing a formal proof on **an abstract mathematical model** of the system
    - Examples of mathematical objects
      - Finite state machines, labelled transition systems, Petri nets, vector addition systems, timed automata, hybrid automata, process algebra, formal semantics of programming language, etc.
- Approach 1: deductive verification
  - Boolean SATisfiability problem (SAT), Satisfiability Modulo Theories (SMT), etc.
- Approach 2: **model checking** (focus of this lecture)

Outline

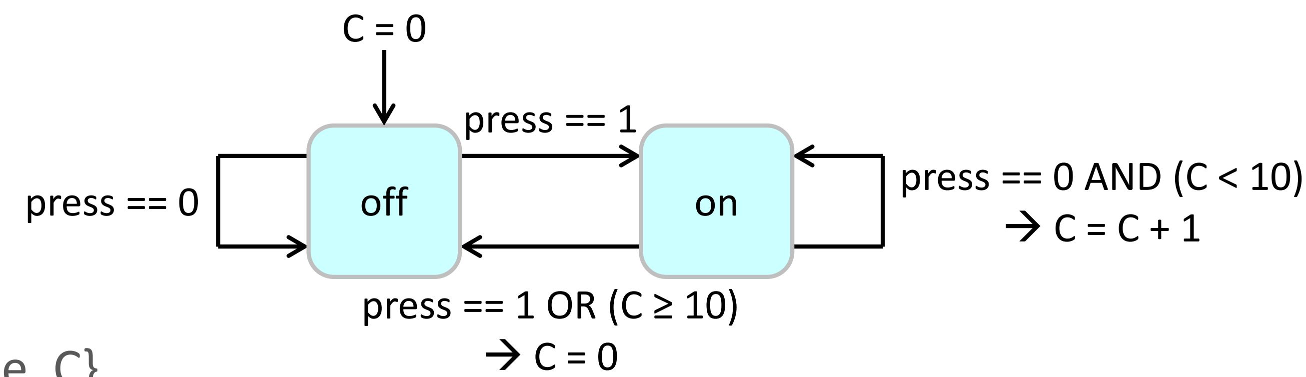
- Formal Verification
  - **Reachability Analysis**
  - Linear Temporal Logic (LTL)
  - Computation Tree Logic (CTL)
  - Signal Temporal Logic (STL)
  - Contract-Based Design
- Testing (and Simulation)

Transition Systems

- A transition system is a tuple (X, X_{init}, T)
  - X: state variables over finite or infinite domains
  - X_{init}: function mapping X to initial values
  - T: transition description to update variables in X
- States in a transition system
  - Q: set of all possible states (could be an infinite set)
    - A state is a combination of values assigned to state variables

Transition Systems

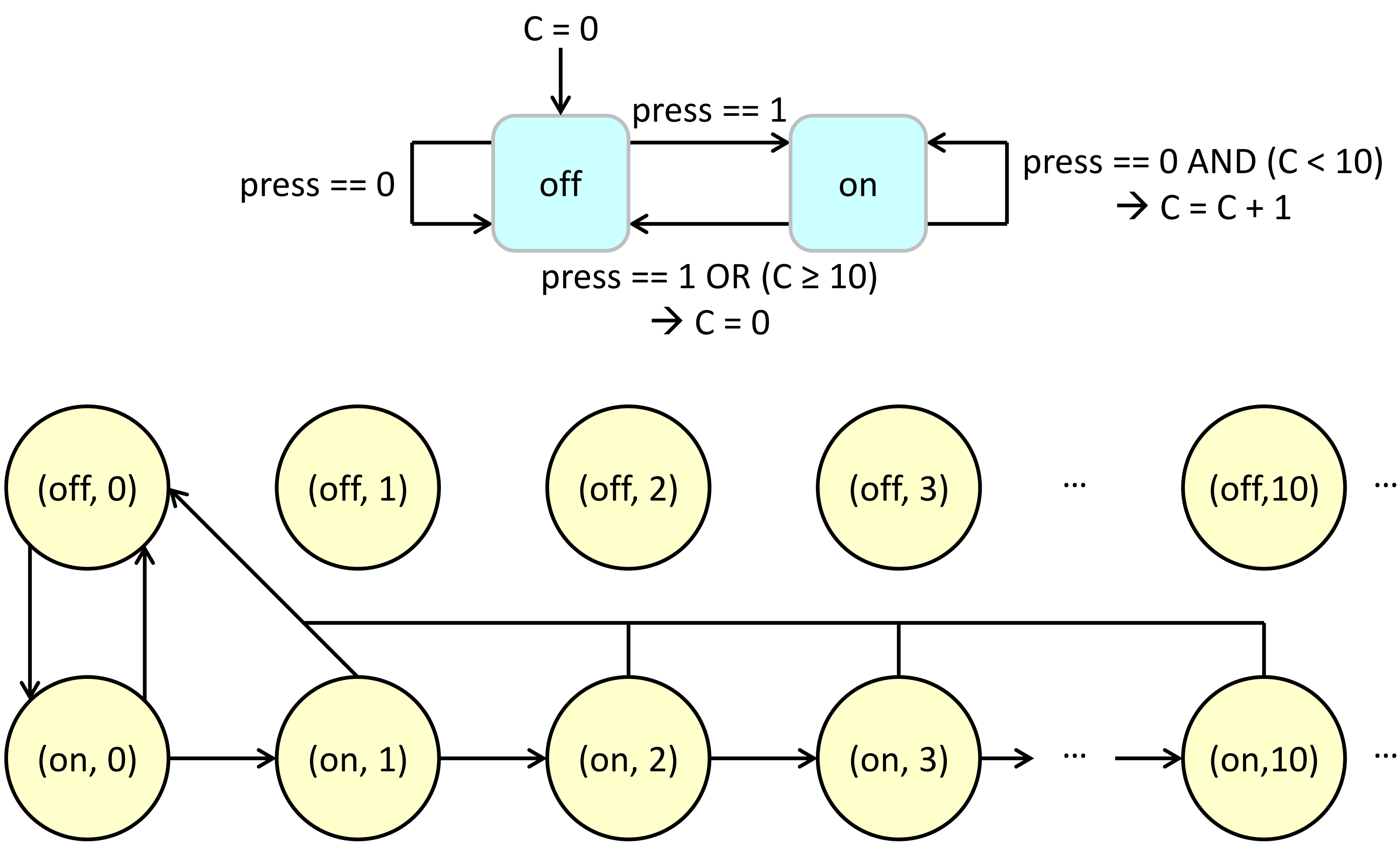
❑ Example



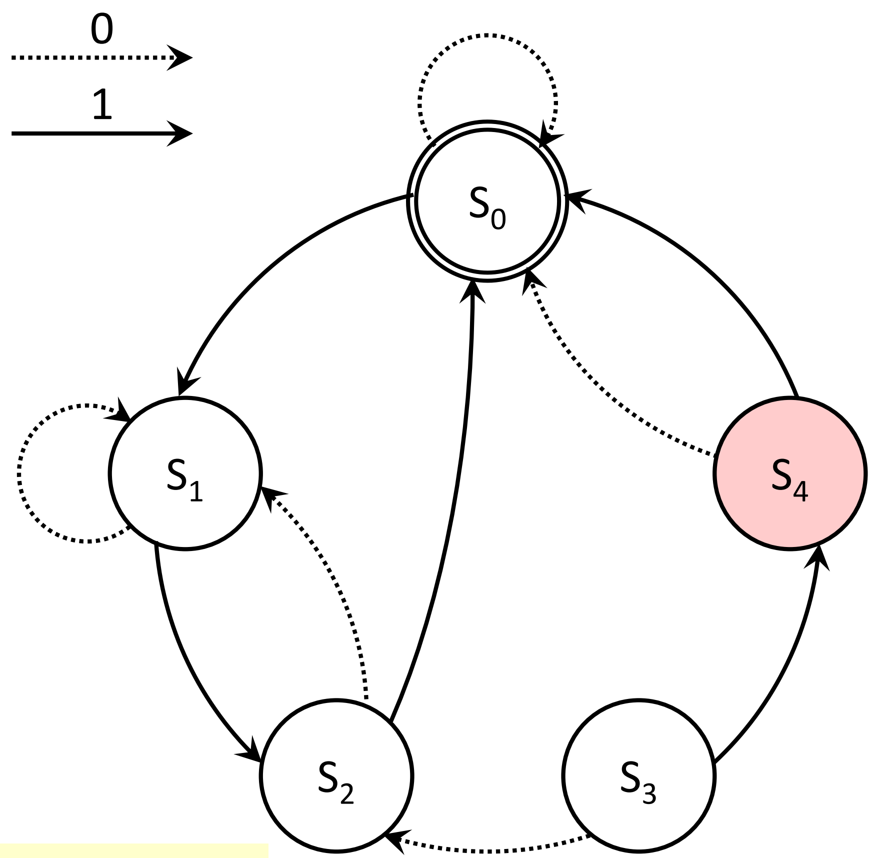
- X: {mode, C}
- X_{init}(mode, C) = (off, 0)
- T
  - (off, 0) → (off, 0)
  - (off, 0) → (on, 0)
  - (on, n) → (on, n+1) if n < 10
  - (on, n) → (off, 0) if n = 10

Reachability Analysis

- Is it possible that something bad (a bad state) will happen?



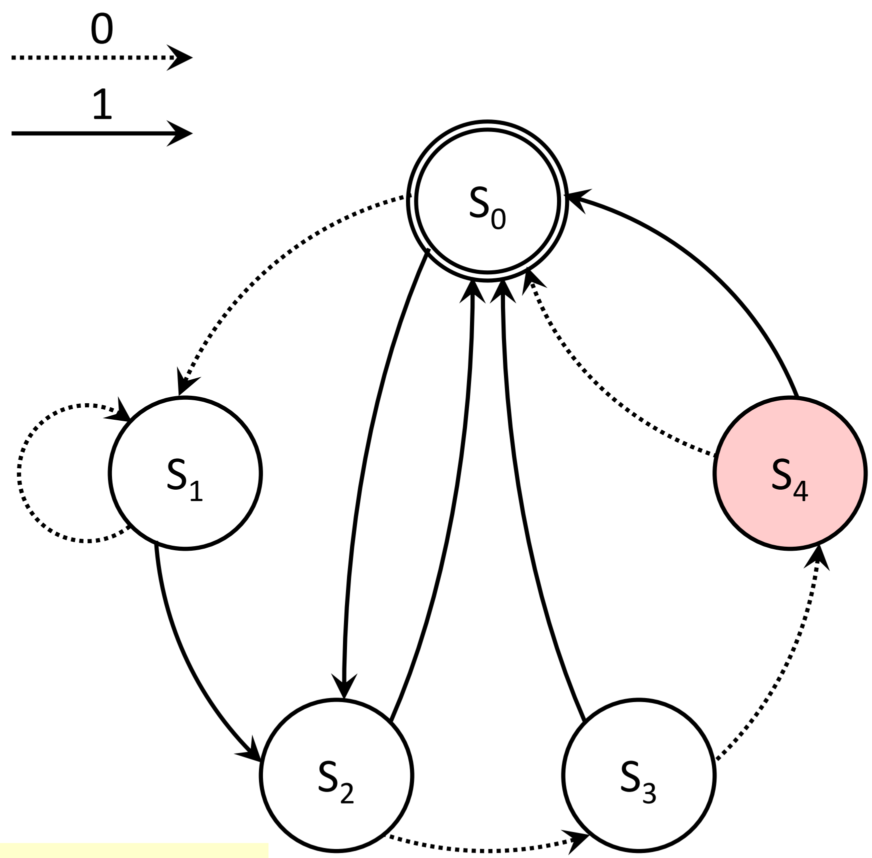
FSM Reachability Analysis (1/5)



State	Next 0	Next 1	Unsafe
S_0 (Initial)	S_0	S_1	0
S_1	S_1	S_2	0
S_2	S_1	S_0	0
S_3	S_2	S_4	0
S_4	S_0	S_0	1

Safe!

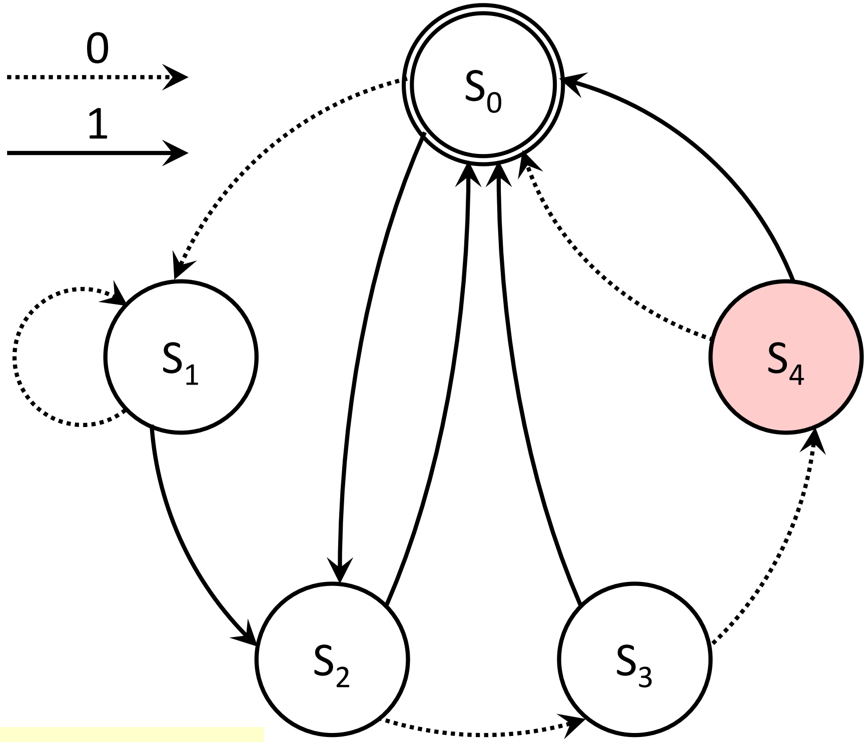
FSM Reachability Analysis (2/5)



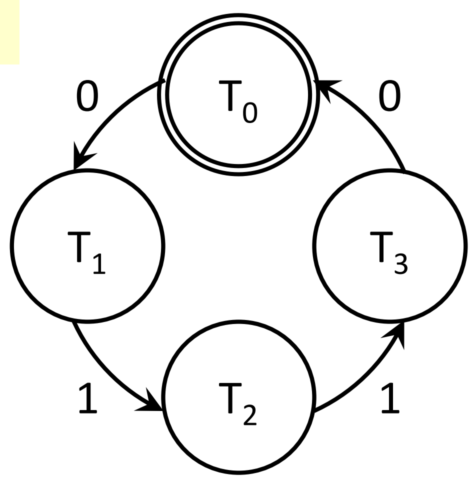
State	Next 0	Next 1	Unsafe
S_0 (Initial)	S_1	S_2	0
S_1	S_1	S_2	0
S_2	S_3	S_0	0
S_3	S_4	S_0	0
S_4	S_0	S_0	1

Unsafe!

FSM Reachability Analysis (3/5)



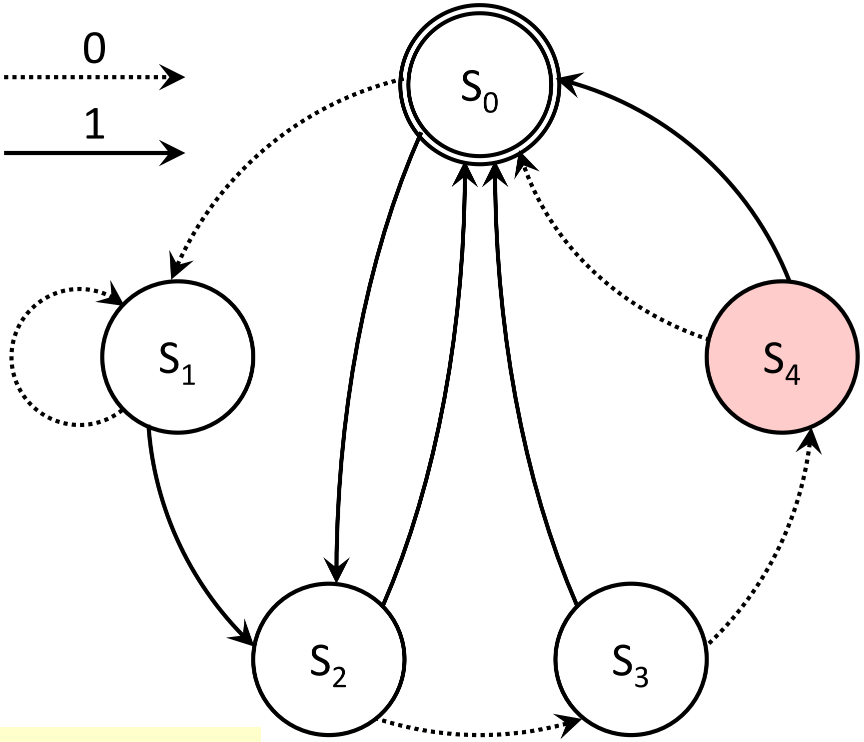
Safe!



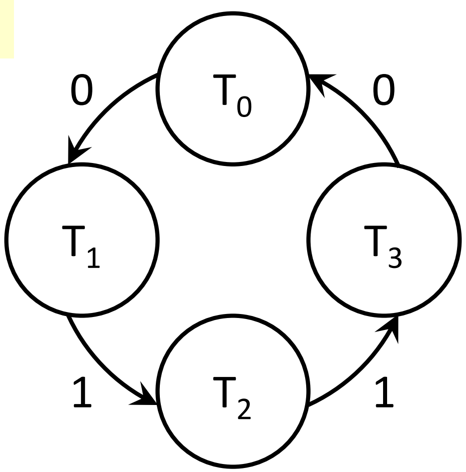
State	Next 0	Next 1	Unsafe
S_0 (Initial)	S_1	S_2	0
S_1	S_1	S_2	0
S_2	S_3	S_0	0
S_3	S_4	S_0	0
S_4	S_0	S_0	1

State	Next	Output
T_0 (Initial)	T_1	0
T_1	T_2	1
T_2	T_3	1
T_3	T_0	0

FSM Reachability Analysis (4/5)



Unsafe!



State	Next 0	Next 1	Unsafe
S_0 (Initial)	S_1	S_2	0
S_1	S_1	S_2	0
S_2	S_3	S_0	0
S_3	S_4	S_0	0
S_4	S_0	S_0	1

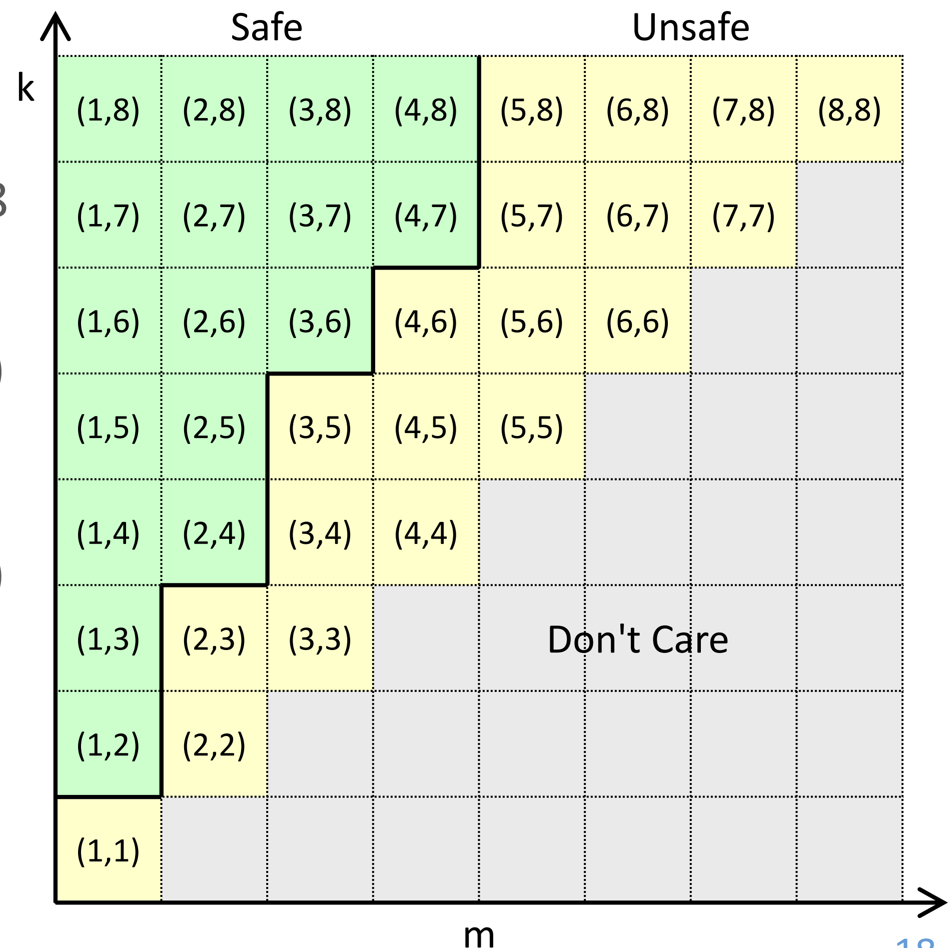
State	Next	Output
T_0	T_1	0
T_1	T_2	1
T_2	T_3	1
T_3	T_0	0

FSM Reachability Analysis (5/5)

- What if the input is bounded by (m,k)?
  - (m,k): there are at most m bad events for any k consecutive events

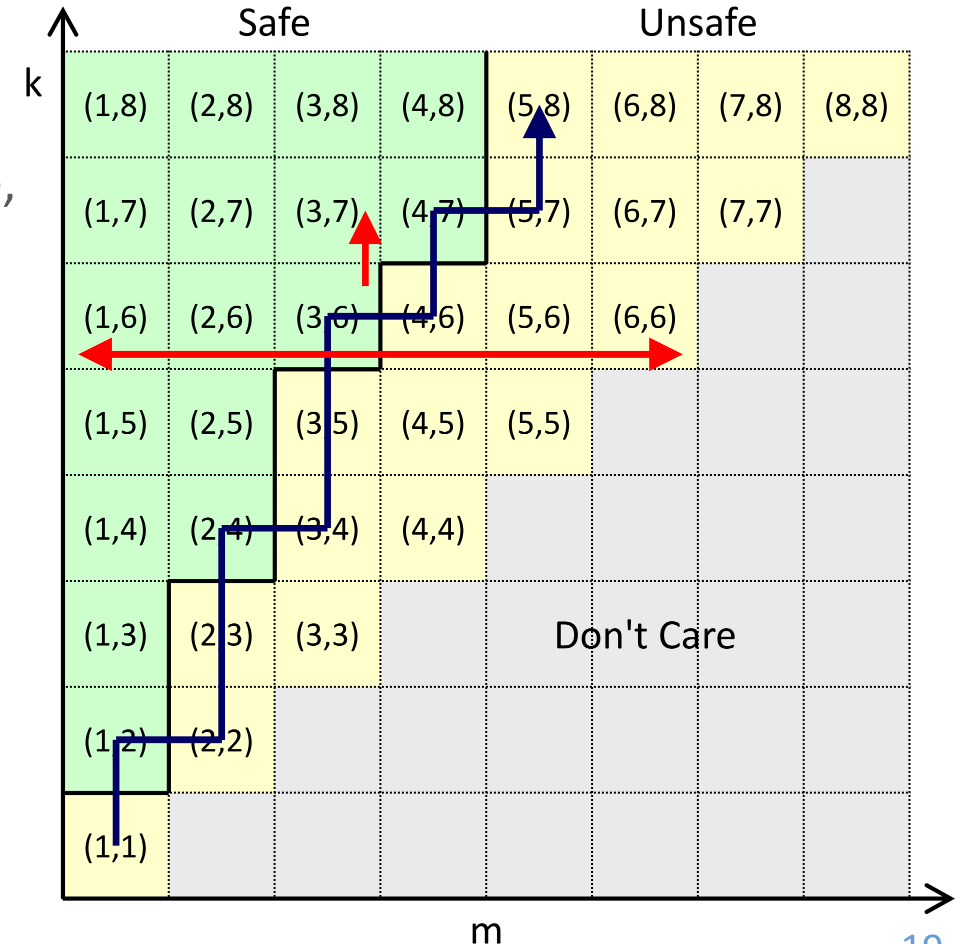
Safety Table

- **<u>Problem</u>**: given K and a system with faulty inputs modeled by weakly-hard constraints, is the system safe with each (m,k)?
  - 0 ≤ m ≤ k ≤ K
- i.e., construct a **<u>safety table</u>**
  - Example: a safety table with K = 8
- Objectives behind verification
  - (1,2), (2,4), (3,6), **<u>(4,7)</u>**: the (local) optimal strategies for system designers
  - (1,1), (2,3), **<u>(3,5)</u>**, (4,6): the (local) optimal strategies for attackers

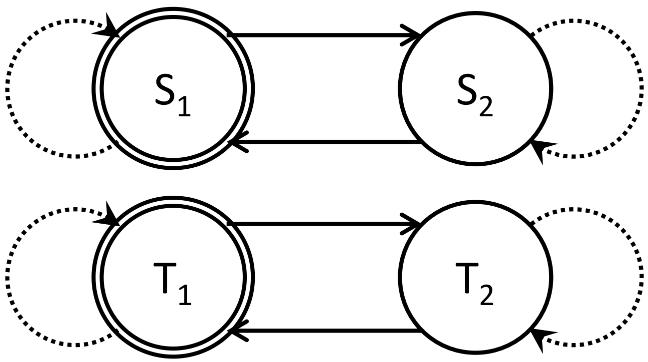


Safety Table Construction

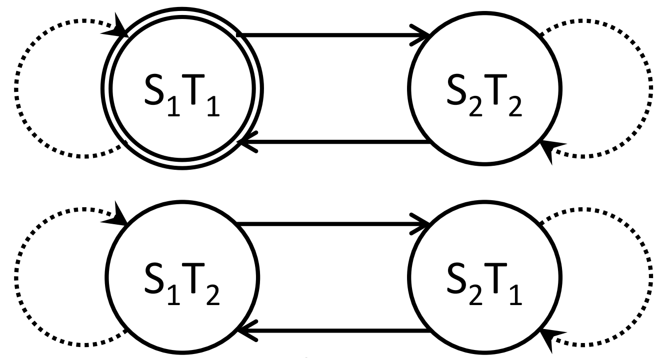
- Observation 1
  - If the system with (m,k) is safe, then, for all m* ≤ m, the system with (m*,k) is also safe
- Observation 2
  - If the system with (m,k) is unsafe, then, for all m* ≥ m, the system with (m*,k) is also unsafe
- Observation 3
  - If the system with (m,k) is safe, then the system with (m,k+1) is also safe
- We do not need to check all pairs of (m,k)



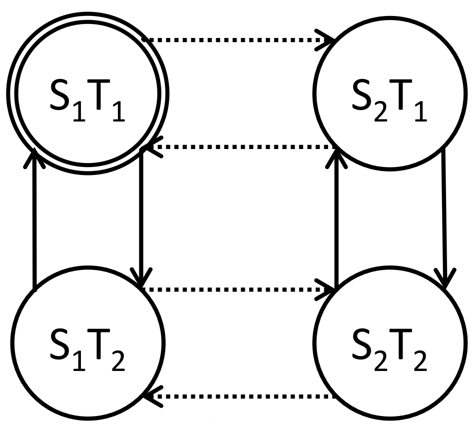
Composition of Finite-State Machines



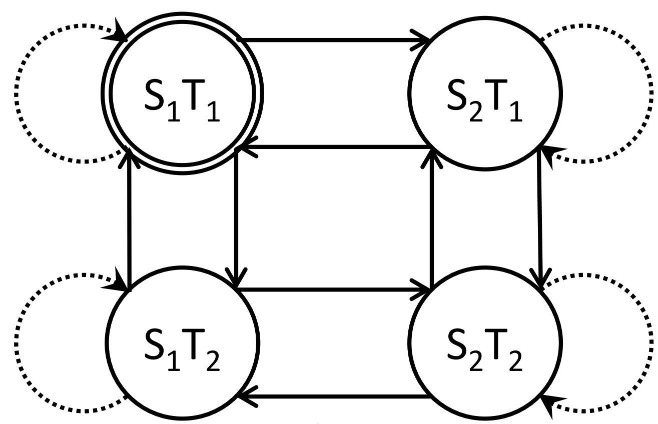
Given Machines



Synchronous
Parallel Composition



Sequential Composition
(Reversed Inputs)



Asynchronous
Parallel Composition

Outline

- Formal Verification
  - Reachability Analysis
  - **Linear Temporal Logic (LTL)**
  - Computation Tree Logic (CTL)
  - Signal Temporal Logic (STL)
  - Contract-Based Design
- Testing (and Simulation)

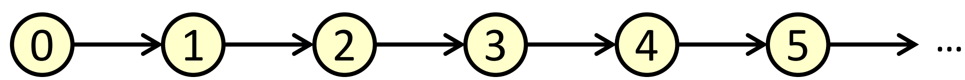
LTL Basics

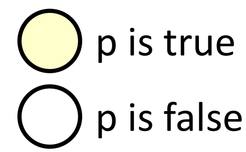
- A logic interpreted over an infinite trace
  - The trace is a discrete-time trace with equal time intervals
    - Actual interval between time-points does not matter
  - Time evolves in a linear fashion
    - Other logics (we will show) have "branching"
  - It can express safety and liveness properties
- Without specification, we are checking the trace from the initial time (at time 0)

LTL Operators

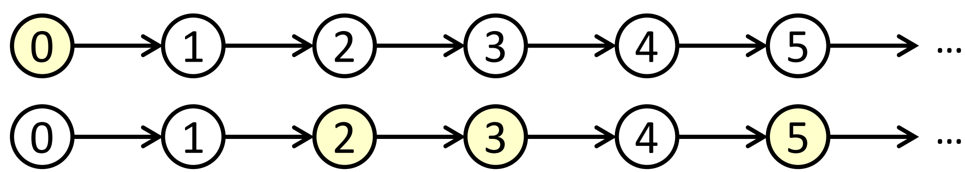
Operators

- p, q atomic proposition
- **G** p p is always true

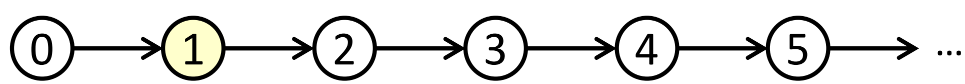




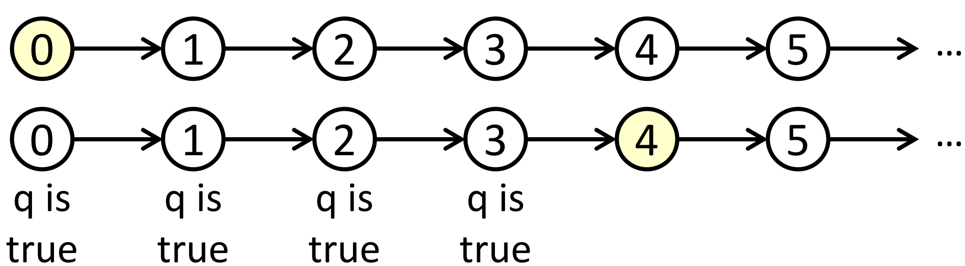
- **F** p p will be true at some point



- **X** p p is true at the next step



- q **U** p p will be true at some point, and q is true until that time

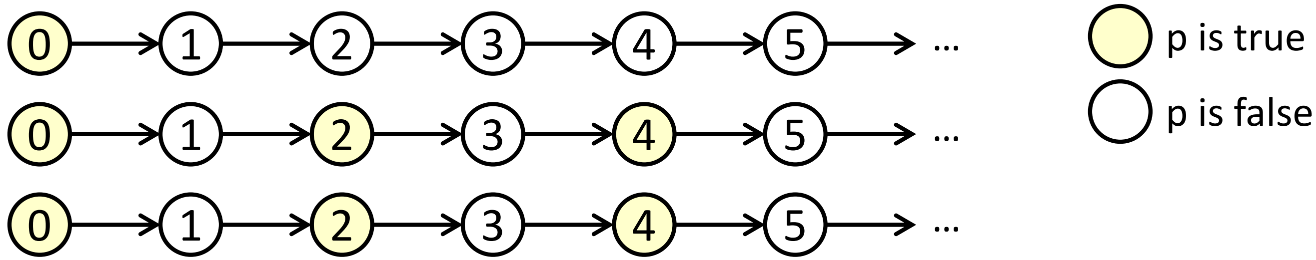


LTL Simple Examples

- Example 1 of LTL
  - p: the security system is on
  - **G** p: the security system is always on
- Example 2 of LTL
  - q: the door is locked
  - **X** q: the door is locked at the next step
- Example 3 of LTL
  - q **U** p: the security system will be on at some point, and the door is locked until that time

LTL Nested Examples

- **XF** p
- **GF** p
- **FG** p



- $p_0 \wedge \mathbf{X}\ (\ p_1 \wedge \mathbf{X}\ (\ p_2 \wedge \mathbf{X}\ (\ p_3 \wedge \mathbf{X}\ p_4\ )))$
- **G** (p → **F** q)

LTL for Model Checking

- Codes

```
x = 0
while ( 1 )
    x = ( x + 1 ) % 10
end while
```

- Example properties
  - **G** ($x \leq 10$)
  - **F** ($x = 5$)

Outline

- Formal Verification
  - Reachability Analysis
  - Linear Temporal Logic (LTL)
  - **Computation Tree Logic (CTL)**
  - Signal Temporal Logic (STL)
  - Contract-Based Design
- Testing (and Simulation)

LTL for Model Checking

- Codes (y as input)

```
x = 0
if ( y == 1 )
    while ( 1 )
        x = ( x + 1 ) % 10
    end while
end if
```

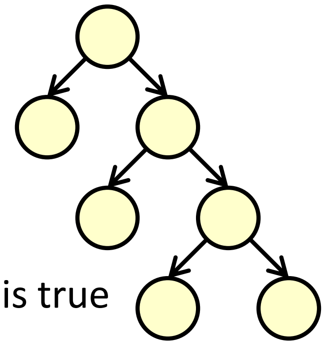
- Example properties
  - **G** ($x \leq 10$)
  - **F** ($x = 5$)
    - What is the result?

CTL Basics and Operators

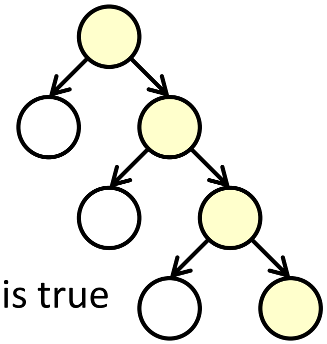
- A logic where we reason over the tree of executions generated by a program, also known as the computation tree
  - Some properties cannot be expressed in LTL but can be expressed in CTL
- Operators
  - **A** for all paths
  - **E** exists a path
- Examples
  - **AG** p, **AF** p, **AX** p, **A** (q **U** p)
  - **EG** p, **EF** p, **EX** p, **E** (q **U** p)

CTL Simple Examples

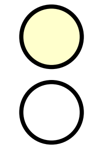
- **AG** p
- **EG** p
- **AF** p
- **EF** p
- **AX** p
- **EX** p



AG p is true

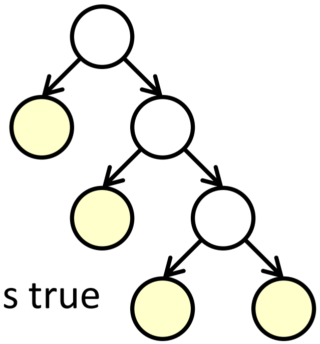


EG p is true

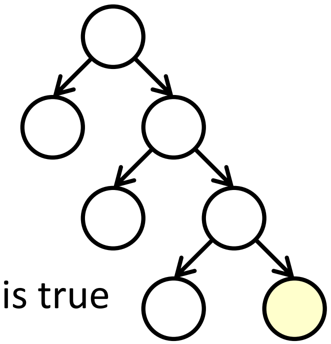


p is true

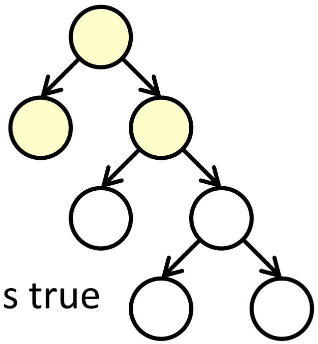
p is false



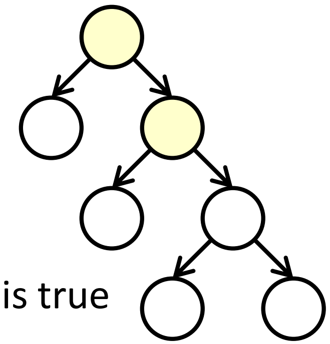
AF p is true



EF p is true



AX p is true



EX p is true

CTL Nested Examples

- **AGEF** p
- **AGAF** p
- **EGAF** p
- **AG** (p → **EX** q)

Outline

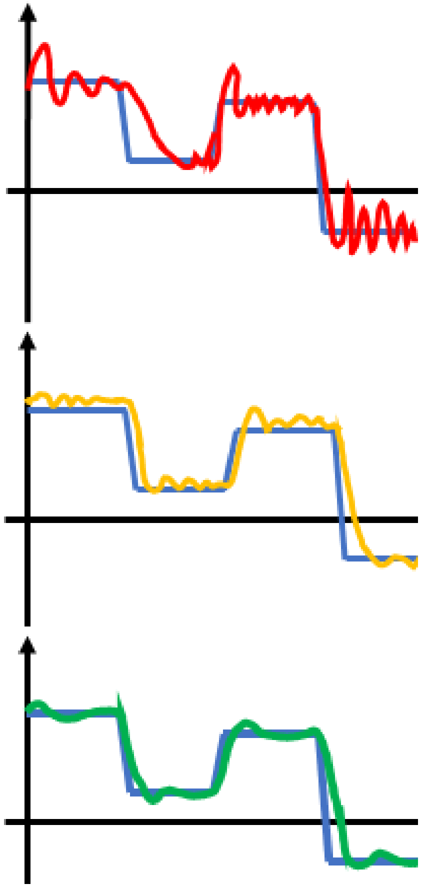
- Formal Verification
  - Reachability Analysis
  - Linear Temporal Logic (LTL)
  - Computation Tree Logic (CTL)
  - **Signal Temporal Logic (STL)**
  - Contract-Based Design
- Testing (and Simulation)

Basics and Operators

- A logic to formalize many control-theoretic properties
  - Express properties of mixed-signal and analog circuits
  - Express timing constraints and causality relations
  - Example: properties of path-planning algorithms
- **Signal** x is a function from a time domain to a value domain
- Operators
  - $\mathbf{G}_{[a,b]}$ p is always true in the internal [a,b]
  - $\mathbf{F}_{[a,b]}$ p will be true at some point in the interval [a,b]
  - $\mathbf{U}_{[a,b]}$ p will be true at some point in the interval [a,b], and q is true until that time

STL Example

- $\mathbf{G}_{[0,10]}$ (step → $\mathbf{G}_{[0,2]}$ ($f_{error}(x) < C$))



For Your Information

Logic	Logic Order	Temporal Semantics	Temporal Structure	Metric for Time	Decidability	Model Checking
LTL	Propositional	Point	Linear	No	Yes	Yes
QTL	First-order	Point	Linear	No	No	?
CTL	Propositional	Point	Branching	No	Yes	Yes
CTL*	Propositional	Point	Branching	No	Yes	Yes
CTL*[P]	Propositional	Point	Branching	No	Yes	Yes
HS	Propositional	Interval	Linear	No	No	No
CDT	Propositional	Interval	Linear	No	No	No
PNL	Propositional	Interval	Linear	No	No	No
ITL	First-order	Interval	Linear	No	No	No
NL	First-order	Interval	Linear	No	No	No
MTL	Propositional	Point/Interval	Linear	No	?	?
TLTL	Propositional	Point/Interval	Linear	Yes	?	?

Outline

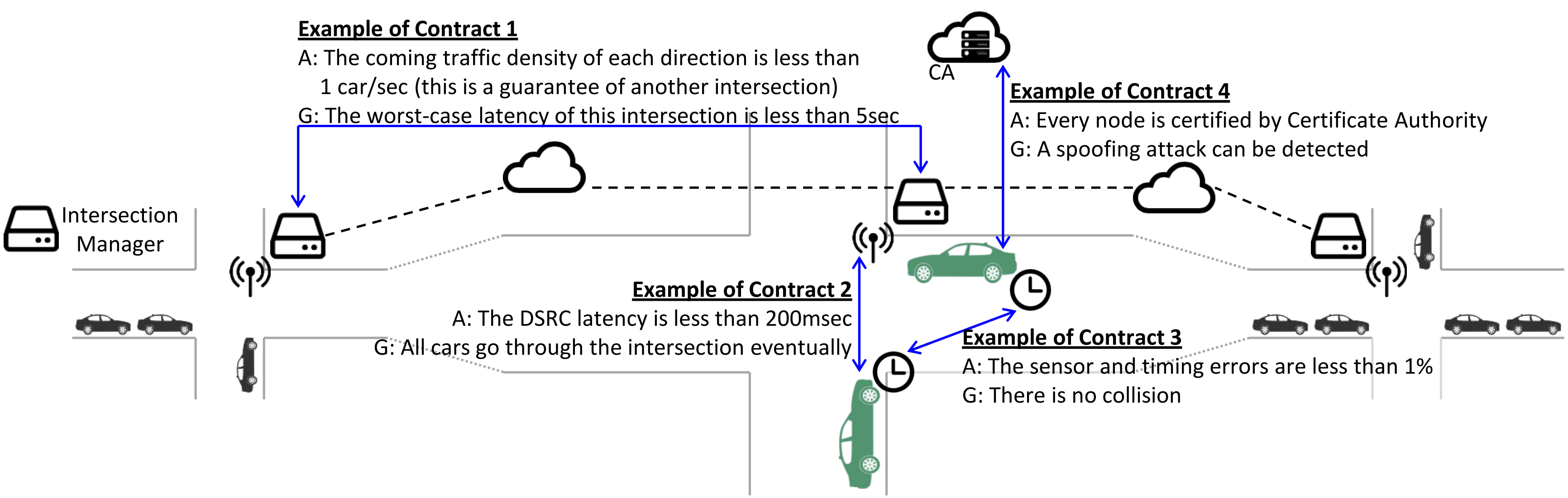
- Formal Verification
  - Reachability Analysis
  - Linear Temporal Logic (LTL)
  - Computation Tree Logic (CTL)
  - Signal Temporal Logic (STL)
  - **Contract-Based Design**
- Testing (and Simulation)

Levels of Contracts

- Level 1: basic contracts
  - They define the interfaces of components, probably by interface definition languages
- Level 2: behavior contracts
  - They define the preconditions and post-conditions of components
- Level 3: synchronization contracts
  - They introduce the timing which enriches the contract expressiveness to the dependency between components
- Level 4: QoS contracts
  - They quantify the expected behavior of components and evaluate their performance

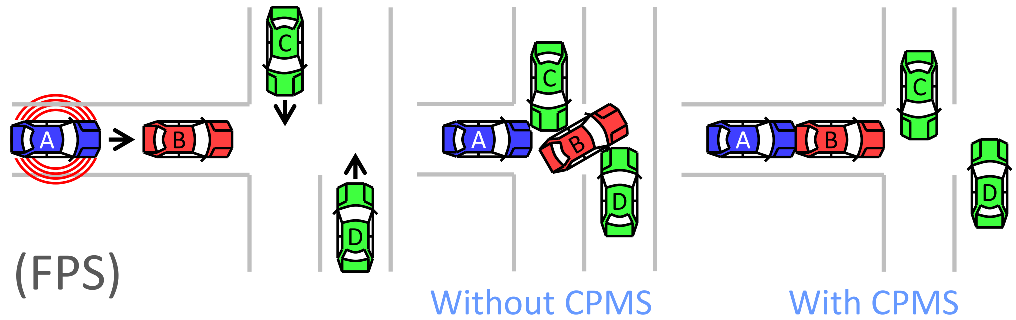
Assume-Guarantee Contracts

- A specification is defined by a contract C = (A, G)
  - A: set of model behaviors for assumptions
  - G: set of model behaviors for guarantees
- A component satisfies a contract if it provides the contract guarantees subject to the contract assumptions
- Check a specification (A→G) violation (implementation error) and an assumption (A) violation (design error)

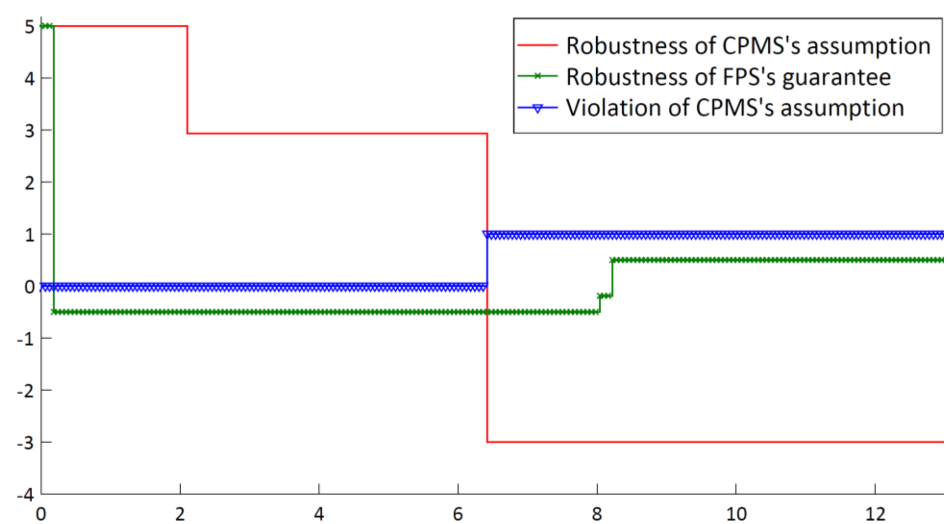


Compatibility of Systems

- Integration of two systems
  - Cooperative Pile-up Mitigation System (CPMS)
  - False-start Prevention System (FPS)



- Property specification language and automation tool
  - Signal Temporal Logic (STL)
    - Extend Linear Temporal Logic (LTL) to specify properties over real time
  - Breach [Donze '10]
    - Given a STL formula, synthesize an online monitor as a C++ program or a MATLAB S-function which can be realized as a Simulink block
- An assumption violation of CPMS is detected!

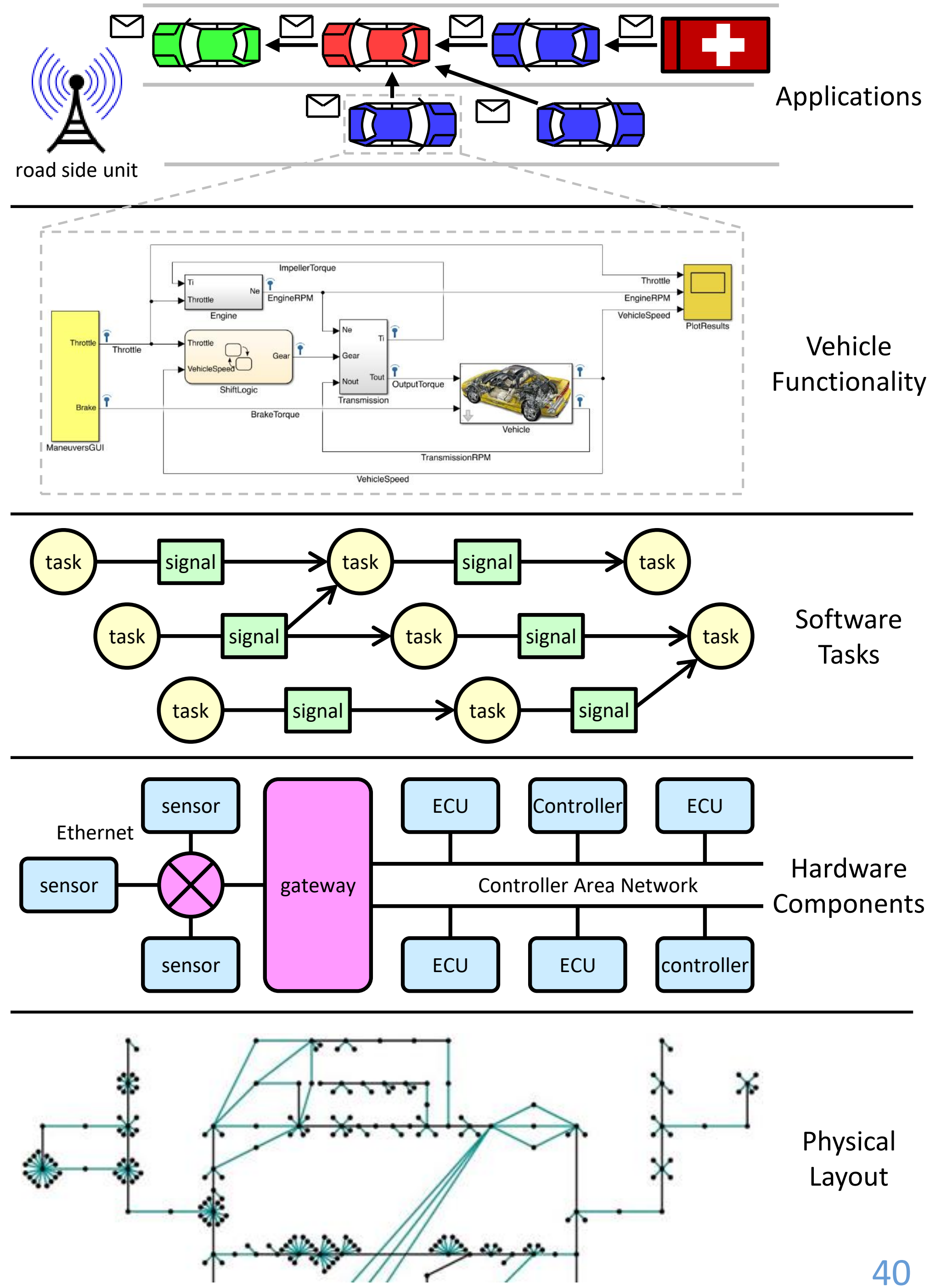


Platform-Based Design

Sensing	Perception	Planning and Decision	Control	Actuation
Radar	Segmentation	Route Planning	Steering Control	Engine
Lidar	Object Detection	Behavior Planning	Torque Control	Brake
Camera	Object Tracking	Motion Planning	Emission Control	Wheel
GPS	Localization	Prediction	Energy Management	Light

(Operating System)

Hardware Platform, e.g., Electronic Control Unit (ECU)



Outline

- Formal Verification
  - Reachability Analysis
  - Linear Temporal Logic (LTL)
  - Computation Tree Logic (CTL)
  - Signal Temporal Logic (STL)
  - Contract-Based Design
- **Testing (and Simulation)**

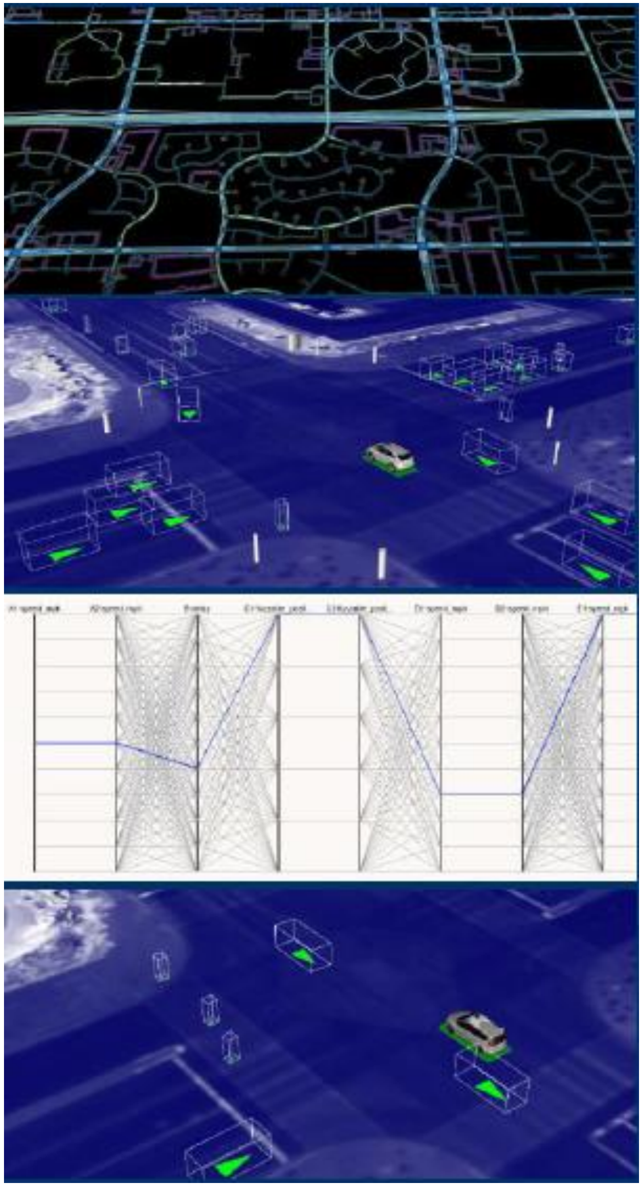
Testing

- Automotive Research & Testing Center (ARTC)
  - Test field in Changhua

https://www.artc.org.tw/chinese/01_testing/00_overview.aspx

Testing

- Waymo's testing
  - Self-driving hardware testing
  - Self-driving software testing
    - Simulation testing
      - Step 1: Start with a highly-detailed vision of the world
      - Step 2: drive, drive, and redrive
      - Step 3: Create thousands of variations
      - Step 4: Validate and iterate
    - Closed-course testing
    - Real-world driving



Simulation

- AirSim
  - https://www.youtube.com/watch?v=gnz1X3UNM5Y
- Carla
  - https://www.youtube.com/watch?v=BjH-pFGlZ0M
- Unity 3D
- SUMO

Philosophy

- Formal verification vs. simulation vs. testing

Q&A