Technische Universität Dresden

barkhausen institut

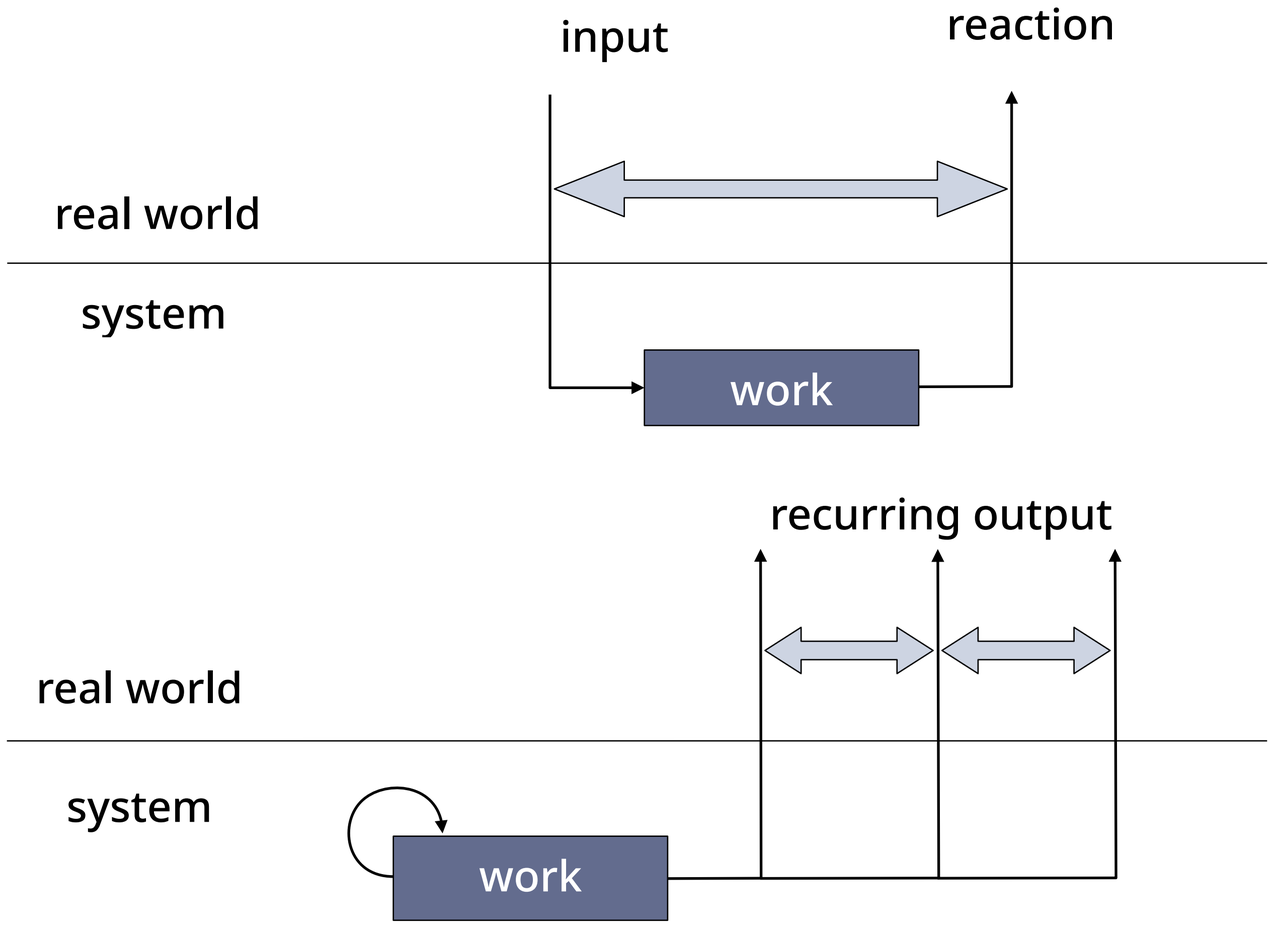
Introduction

Real-Time Systems

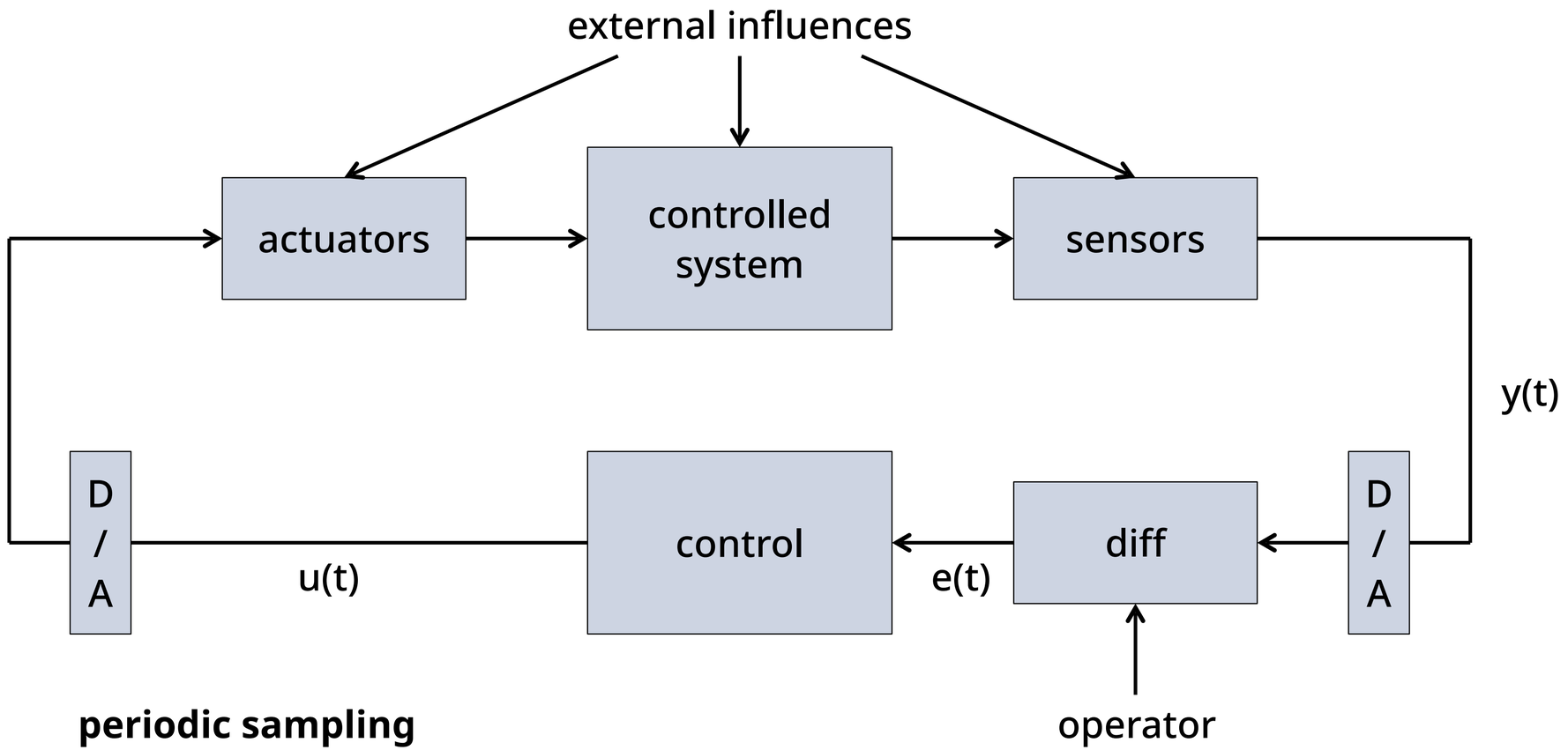
Examples of Real-Time-Systems

- planes, trains, cars, e-bikes
- ABS, ESP, autonomous driving
- industrial control systems
- robot control: stability (short term) and movement (longer term)
- communication protocol: cellular, WiFi
- touch interaction and haptic (force feed back) devices
- video and audio processing
- game engines

Reactive and Generative Real-Time Systems



Simple Digital Control System



PID Controller – Continuous to Discrete

Continuous formula: $$u(t) = k_p e(t) + k_i \int_{\tau=0}^{t} e(\tau)d\tau + k_d \frac{de(t)}{dt}$$

Approximation by periodic sampling (rate T)

Integral via Simpson's Rule: $$\frac{T}{3} * (e_{k-2} + 4e_{k-1} + e_k)$$

Differential: $$\frac{e_k - e_{k-1}}{T}$$

Then: $$u_k = u_{k-2} + ae_k + be_{k-1} + ce_{k-2}$$

With $$a = k_p + \frac{k_i T}{3} + \frac{k_d}{T}, \quad b = \frac{4k_i T}{3} - \frac{k_d}{T}, \quad c = \frac{k_i T}{3}$$

Template for a Digital Controller

```
at every sample period time units do
    read y and compute e
    u_k := u_k-2 + a*e_k + b*e_k-1 + c*e_k-2
    write u
done
```

sample period depends on:

- reactivity of person (<100 ms)
- reactivity of controlled object

Terminology – Non-Real-Time vs. Real-Time

Metric	Non-Real-Time System	Real-Time System
Capacity	High Throughput	Schedulable Utilization
Responsiveness	Low Average Latency	Guaranteed Latency
Overload Behavior	Fairness	Stability, Criticality

Real-Time Systems

Definition (strict)

Systems, whose correctness depends

- (not only) on the correct logical results of computations
- (but also) on meeting **all** deadlines.

Deadlines are dictated by the environment of the system.

Results and deadlines must be specified.

Real-Time Systems

Definition (weaker)

Systems, whose quality depends

- (not only) on the logical results of computations
- (but also) on the time these results are produced.

Required timing characteristics originate from the environment of the system.

Surrounding Terminology

Embedded Systems

- computers as part of something else
- can be Real-Time Systems

Cyber-Physical Systems

- interaction with physical environment
- often Real-Time Systems

Safety-Critical Systems

- humans in danger, mission in danger

Internet-of-Things

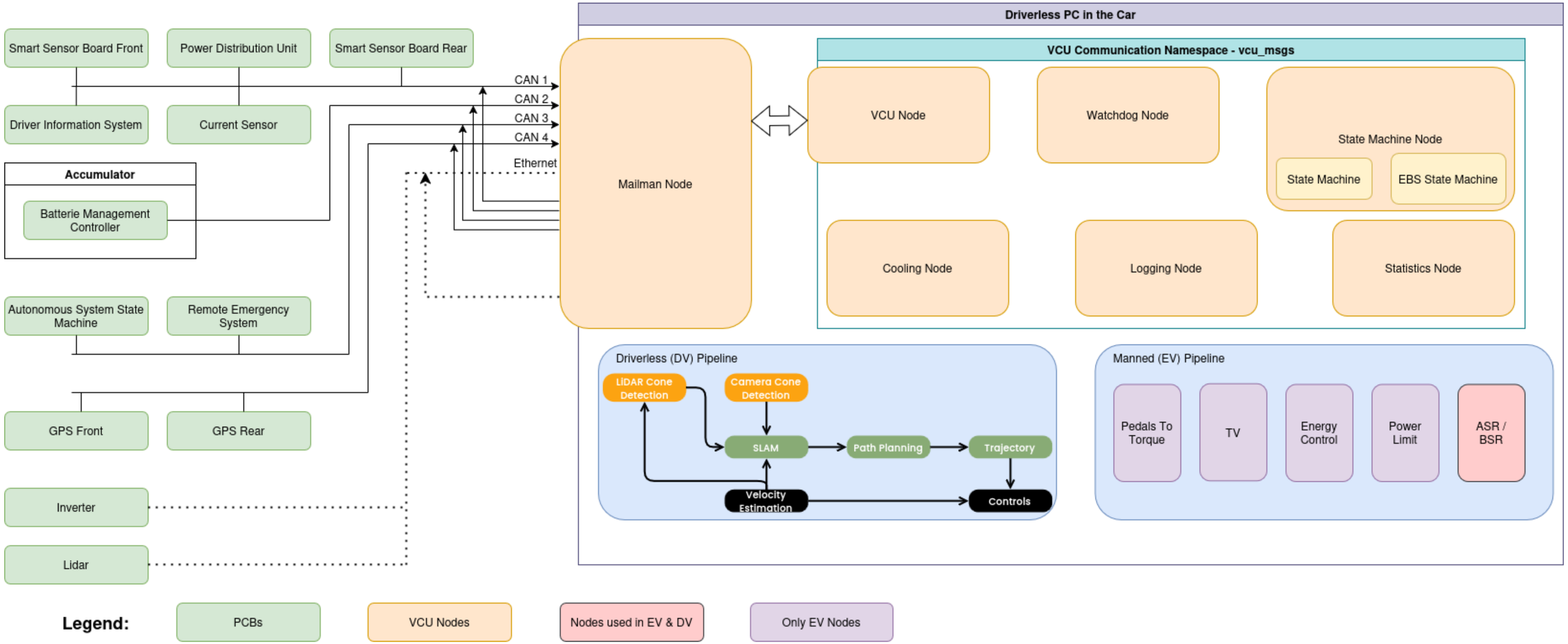
- connectivity for configuration and telemetry

Example: Driverless Car Control

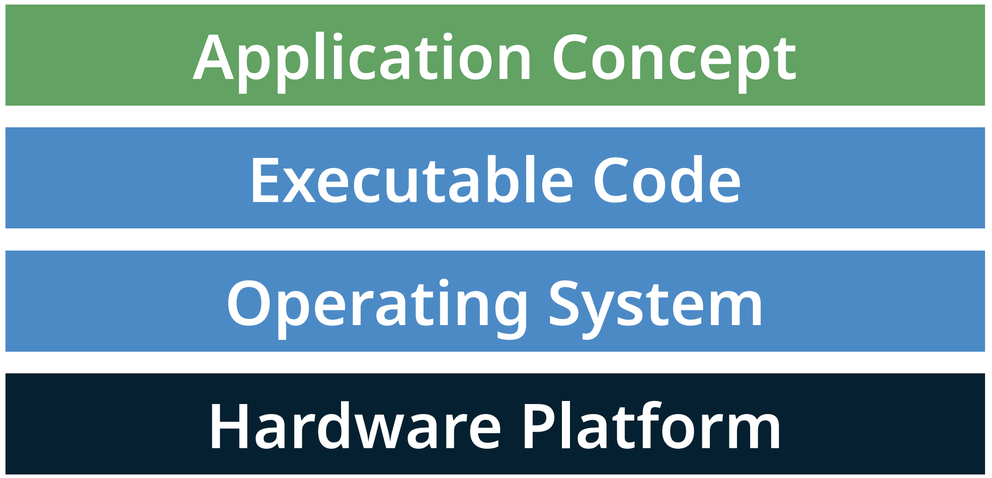
ELBFLORACE
FORMULA STUDENT TEAM TU DRESDEN

- project at TUD, founded in 2006
- constructs a new race car every year since 2008
- since 2010: fully electric
- since 2017: also driverless
- team ranks in the top 20 of the world ranking list

Current Implementation



Simplified Non-Real-Time Technology Stack



What we need – Application Timing Requirements

Application Concept

Timing Requirements

Task Model

Executable Code

Operating System

Hardware Platform

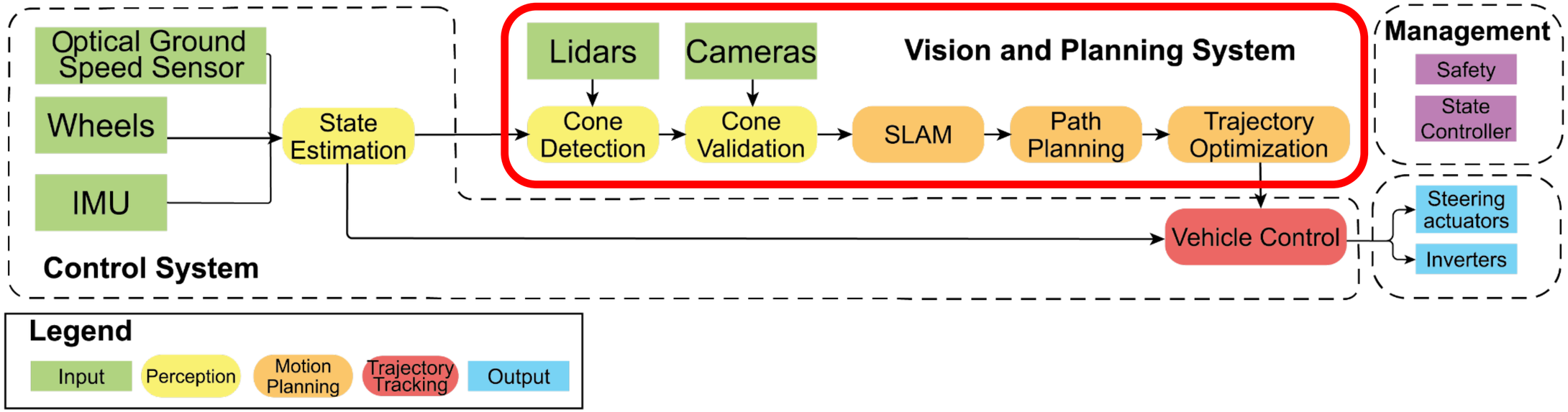
Problem Statement: Stay within the Cones

Driving Specs

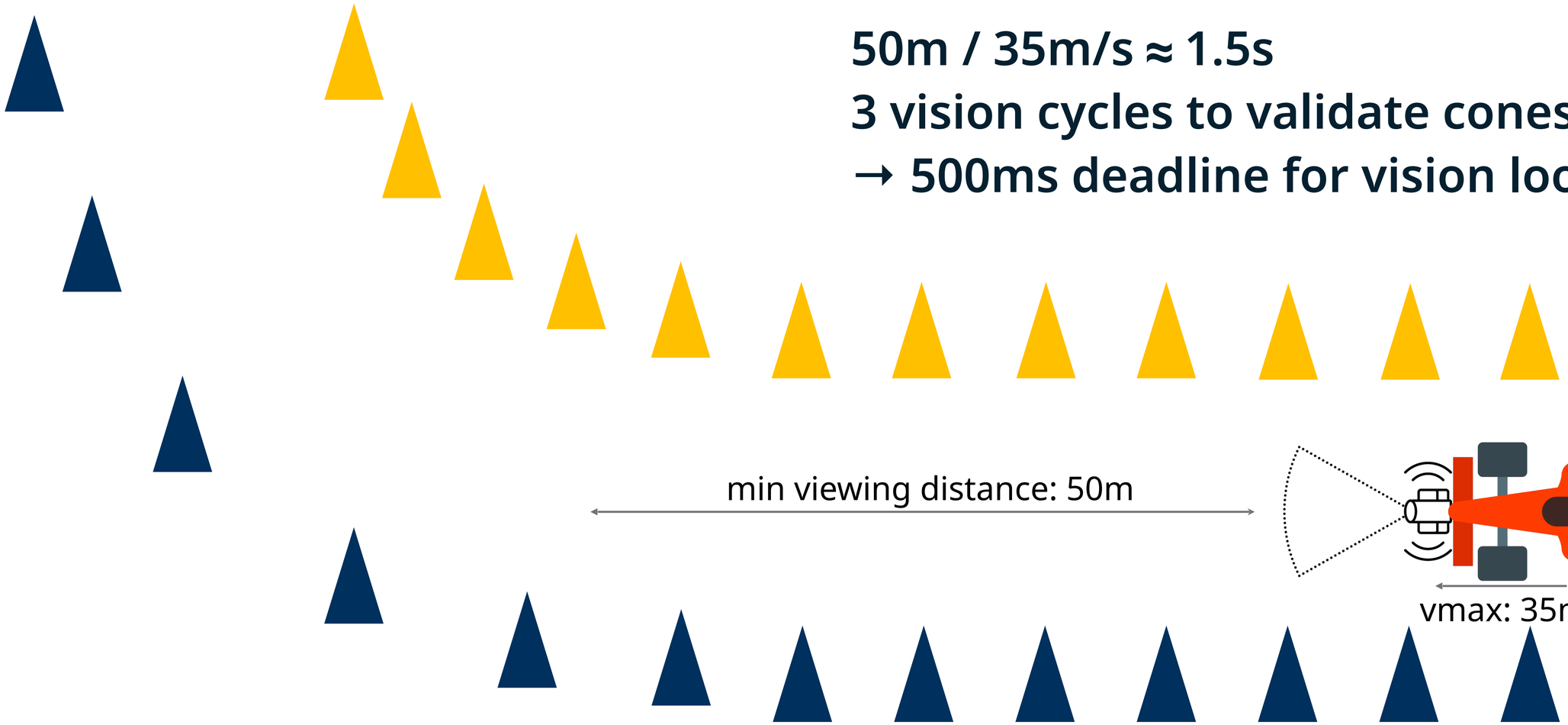
- v_{max} = 125km/h = 35m/s
- 0–100 in < 2.4s

need: calculation from vision to actuation

Timing Requirements – Vision and Motion Planning

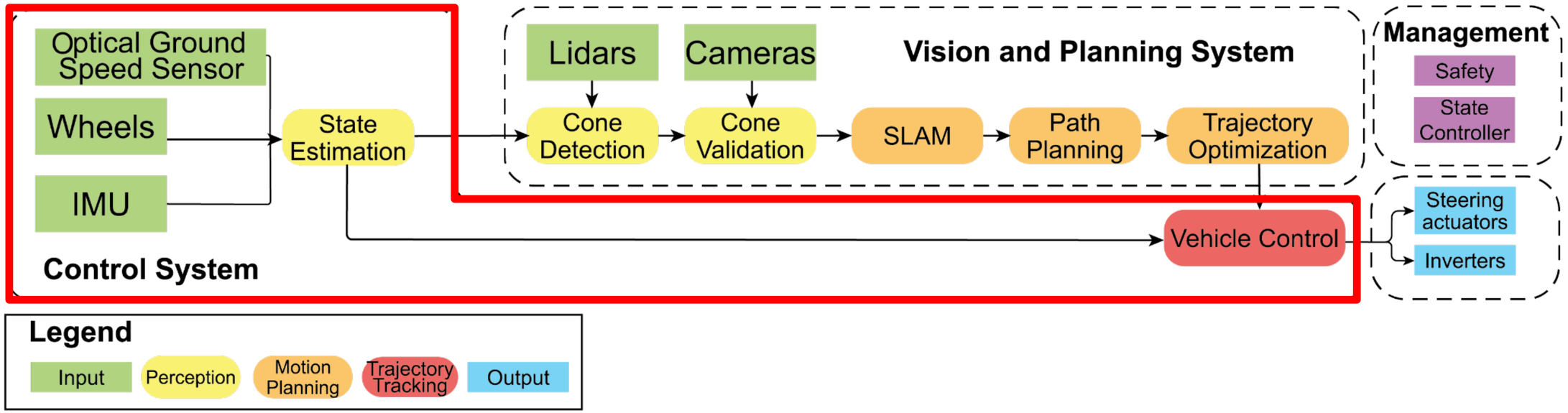


Timing Requirements – Vision and Motion Planning

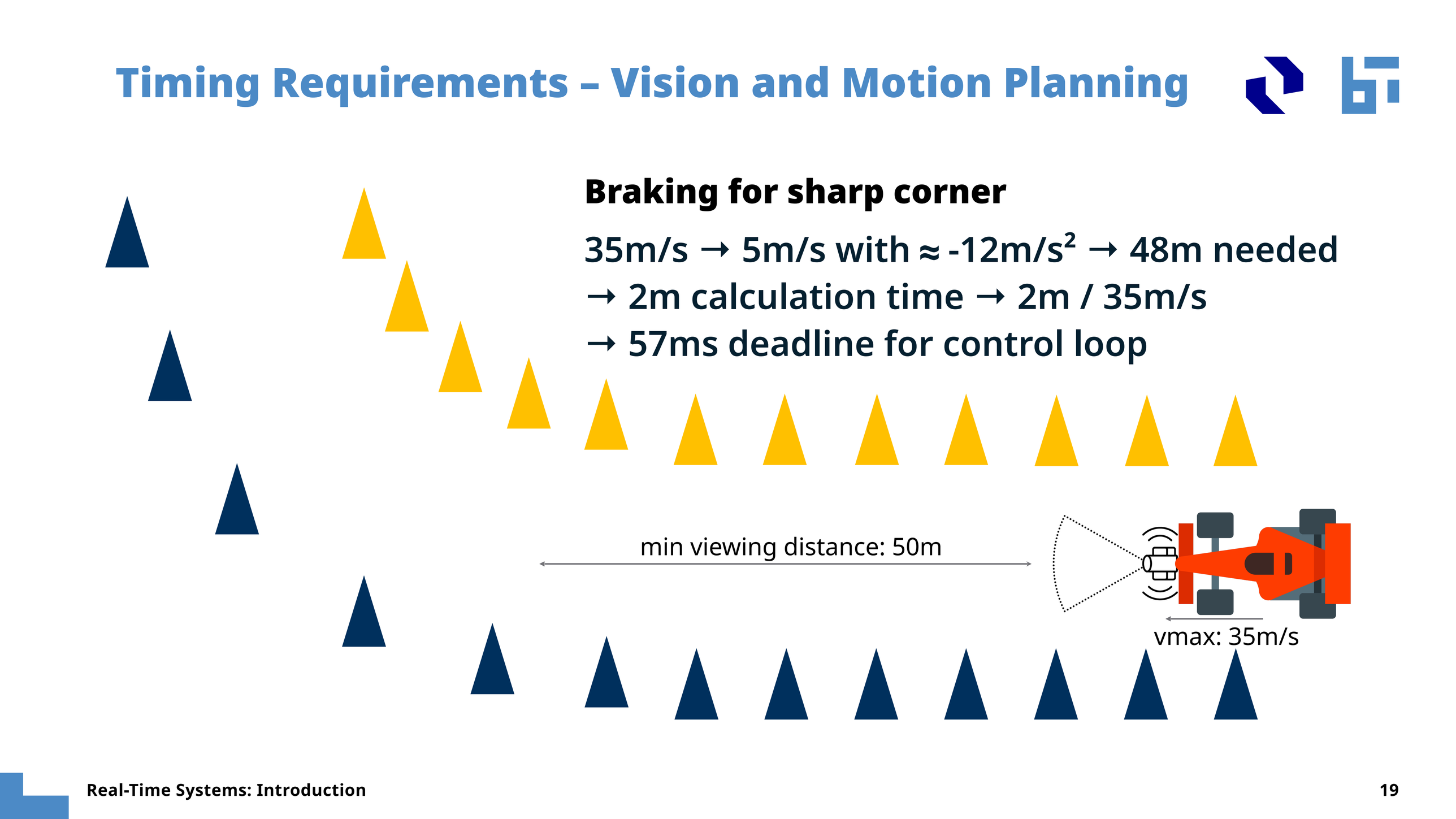


50m / 35m/s ≈ 1.5s
3 vision cycles to validate cones
→ 500ms deadline for vision loop

Timing Requirements – Control System



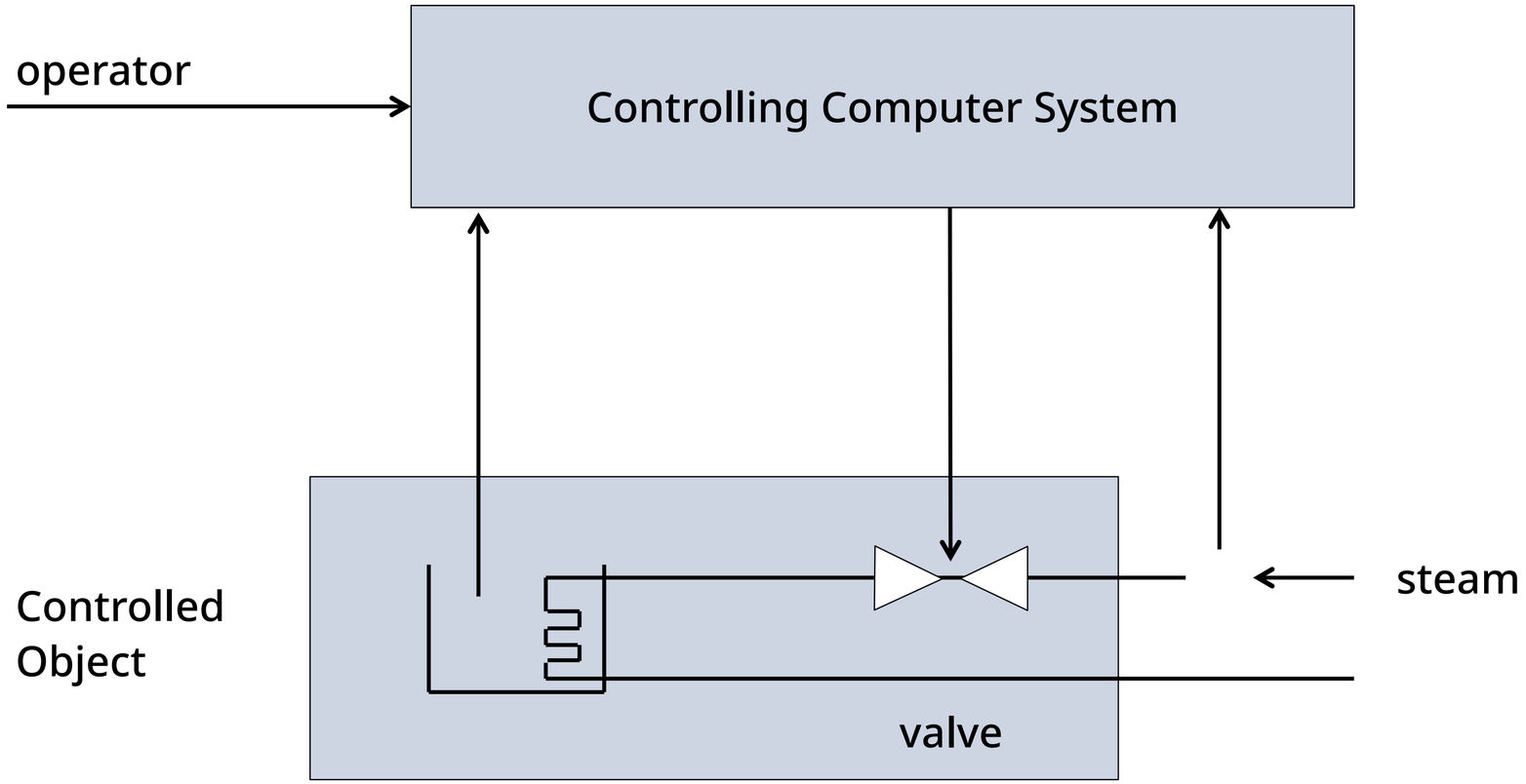
Timing Requirements – Vision and Motion Planning



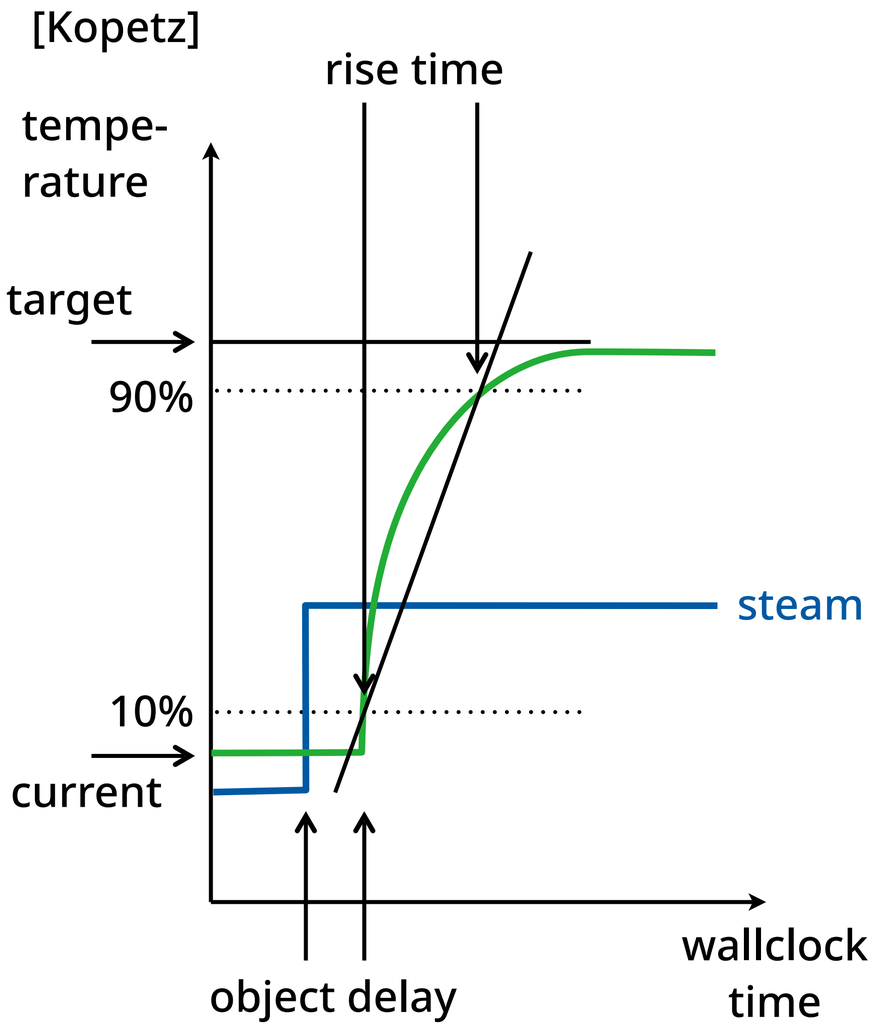
Braking for sharp corner

35m/s → 5m/s with ≈ -12m/s² → 48m needed
→ 2m calculation time → 2m / 35m/s
→ 57ms deadline for control loop

Problem Statement: Industrial Control Example



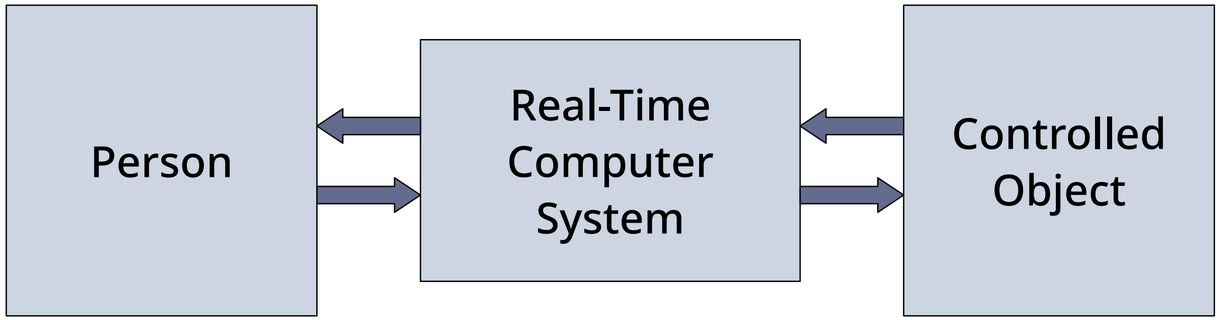
Timing Requirements – Analysis of System Physics



- rise time: 10% or other small neighborhood
- object delay: inertia of control process
- computation delay and jitter:
  < sample period
- deadtime: object delay + computation delay
- sampling period: rule of thumb
  < 1/10 to 1/20 rise time
- shorter sampling periods result in:
  smoother operation, less oscillation, more resources used

Top-Level and Derived Timing Requirements

For many control applications (cyber physical systems), a real-time system sits between a human operator and the controlled object:

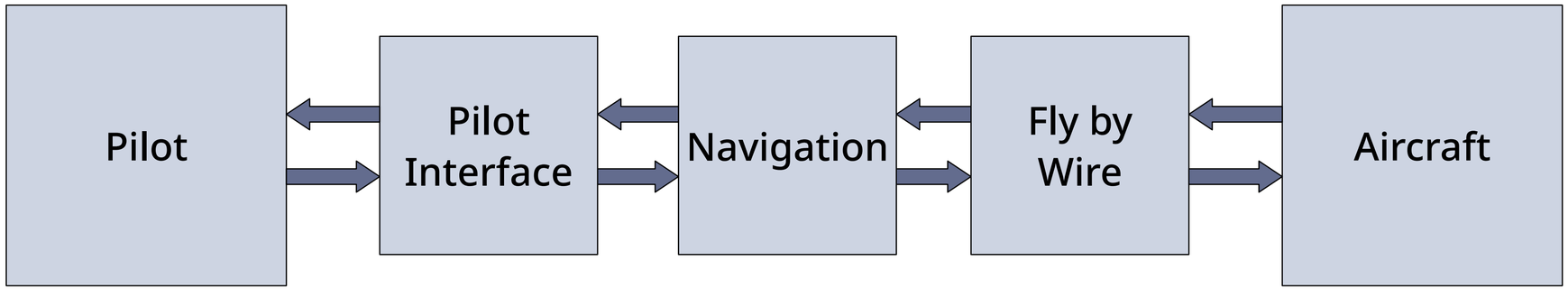


Sources of Timing Requirements

- top-level requirements: dictated by or imposed on controlled object
- derived requirements: dictated by interfaces or abstraction layers

Top-Level and Derived Timing Requirements

Multiple stages may induce multiple derived requirements:



Deadline Flavors

- some deadlines are more important than others
  (later topics: imprecise computations, mixed criticality)
- occasional misses of deadlines can be OK (e.g., 3 in 10, 30%)
- approximate values may be sufficient
  (later topics: approximate computing, energy saving)
- the value of a result depends on the time it becomes available:
  an imperfect early result may be better than a perfect late result
- the more results can be obtained before a given deadline the better
- explicit functional mapping of result availability time to result value

Deadline Flavors

Specification needed for:

- results, deadlines AND
- ‘importance’ of certain deadlines OR
- how many deadlines per time period may be missed OR
- mapping of time to values of results OR …

A saying by Doug Jensen (?):

Hard real-time systems are hard to build,
soft real-time systems are even harder.

Deadline Flavors – Hard, Firm, Soft

hard real-time systems

- deadlines are strict:
  miss has fatal consequences for controlled object or humans

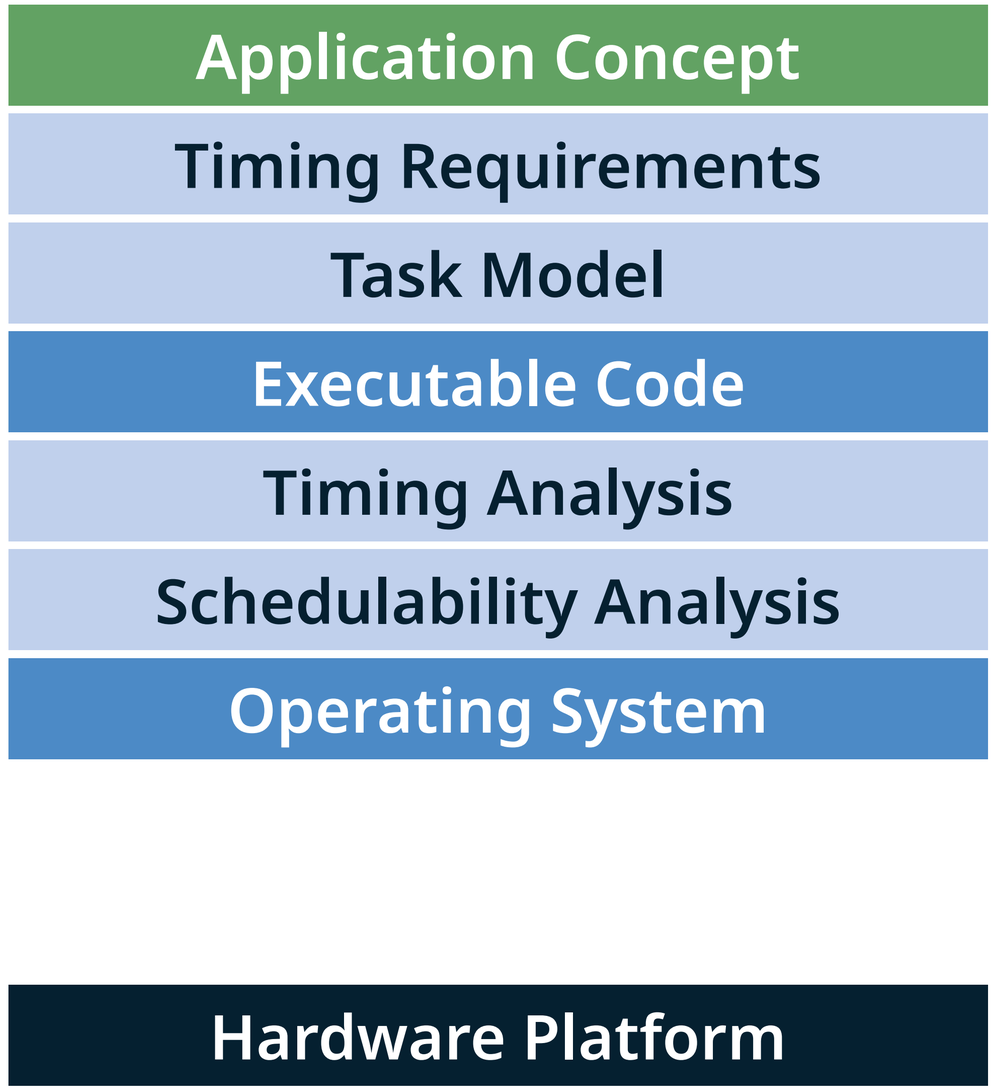
firm real-time systems

- deadlines are strict: late results have no benefit
- some limited deadline misses can be tolerated

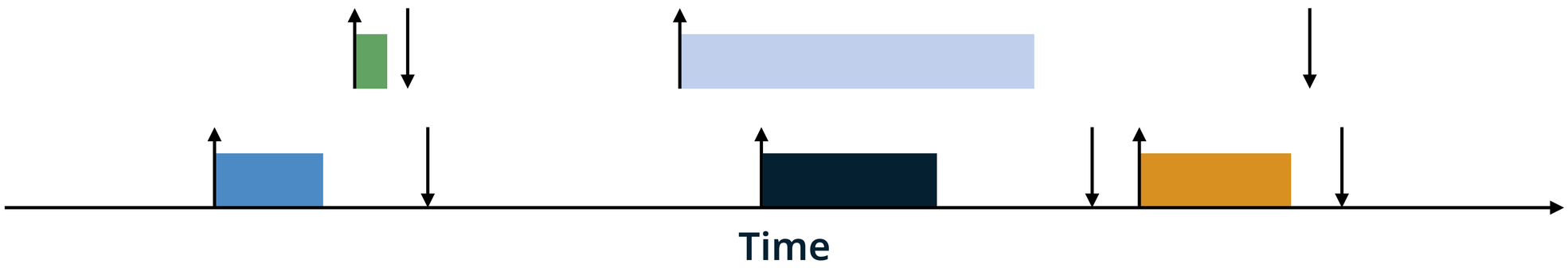
soft real-time systems

- deadlines should be met
- value of results decreases with time
- graceful degradation under peak load is acceptable

Does it work – Timing and Scheduling Analysis



Scheduling – Fundamental Approach



- job: release time, deadline, required execution time
- set of jobs to be scheduled
- in general case: **NP-hard** problem
- can always **enumerate all possible schedules** and find a working one
- simplify by restricting the problem space by introducing **task models**

Scheduling – Terminology

schedule

- some order of the jobs to be scheduled

valid schedule

- an order that meets all job constraints (precedence, release times, execution times)

feasible schedule

- valid schedule that also meets all timing requirements (deadlines)
- job sets for which a feasible schedule exist are called **schedulable**

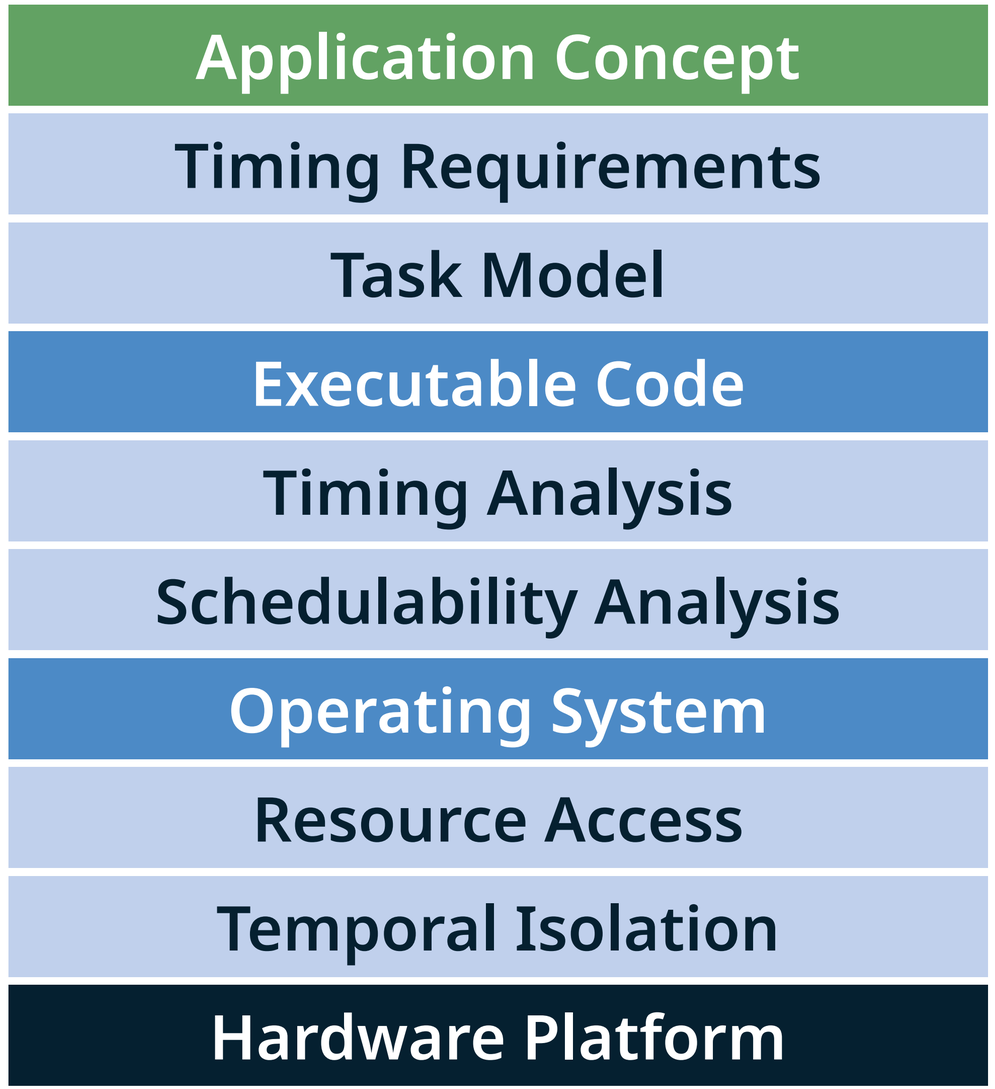
Scheduling – Worst Case Execution

- execution times used to construct a schedule must constitute **worst case**
- measured worst case vs. true worst case vs. formal code analysis
- during execution we can always **pad** execution up to the worst case
- not needing to pad execution can be a non-trivial property

sustainable scheduling policy

A scheduling policy or a schedulability test is sustainable with respect to a particular workload model if any task system represented in that model that is determined to be schedulable remains so if it behaves “better” than mandated by its specifications. (Baker, Baruah)

How to enforce – Isolating Multiple Applications



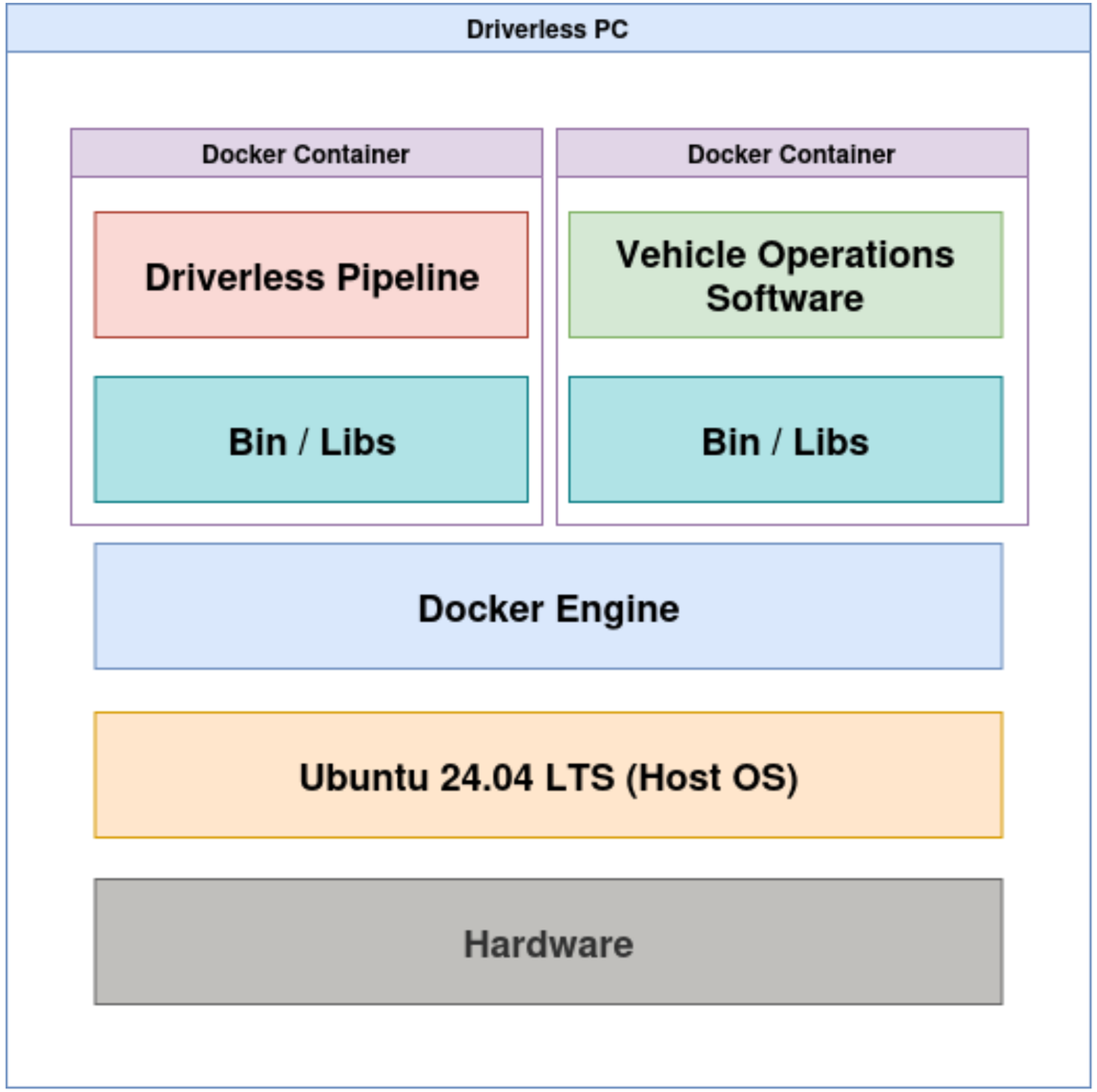
Predictability of Hardware and Software

Average Case vs. Worst Case Improvements

- code (re-)compilation:
  out-of-order execution, compiler optimisations for common execution path
- caches:
  CPU caches, TLBs, SSD caches, buffer cache in operating system
- interference on shared resources:
  cache coherence network, memory bus, PCI bus, system services
- arbitration:
  SSD request ordering, DRAM request ordering, network packet ordering
- complex interposition:
  system management mode, middleware layers

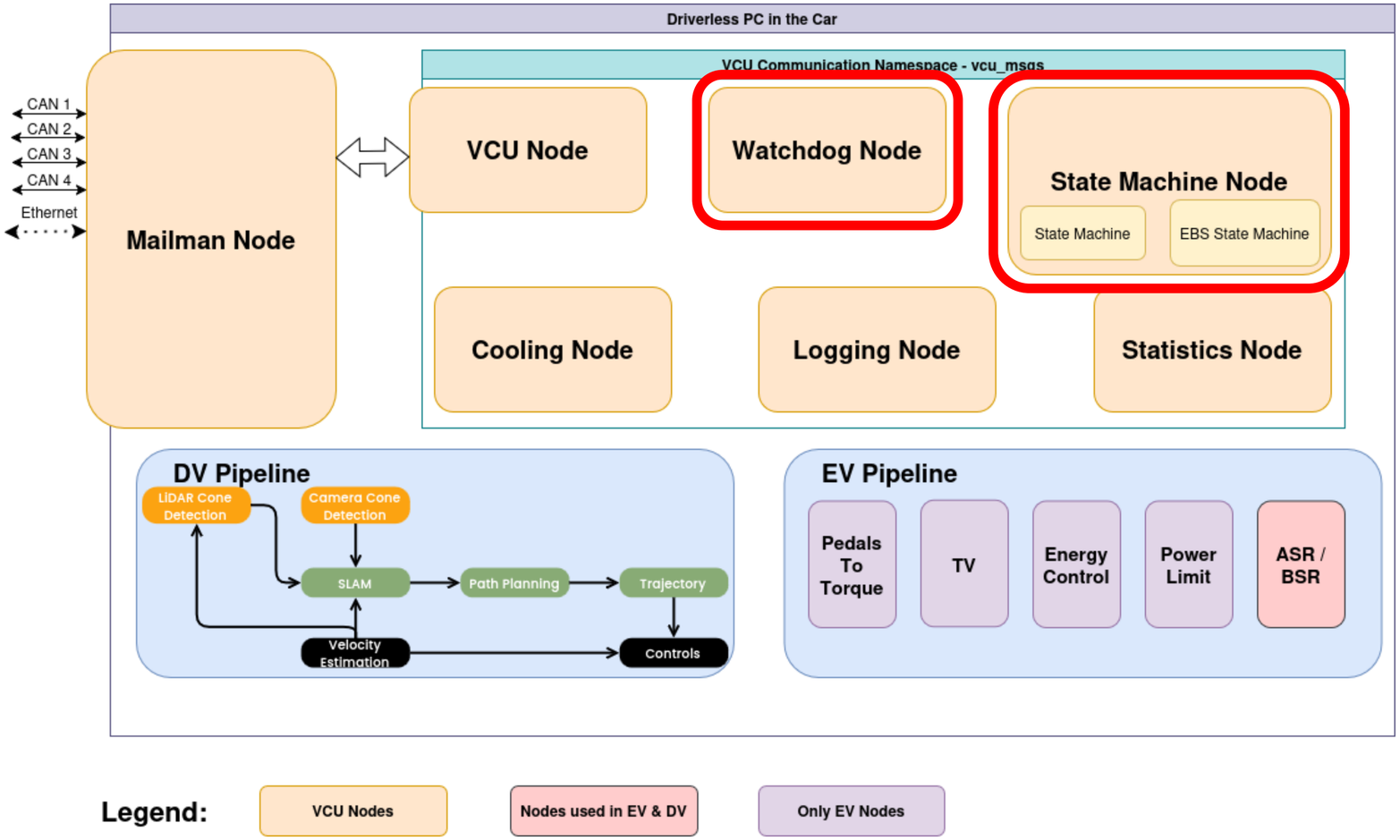
Elbflorace Example: Hardware & OS

- x86_64 PC with 64 GB RAM
- CAN-to-M.2 card for communication with PCBs
- Ethernet for communication with inverter

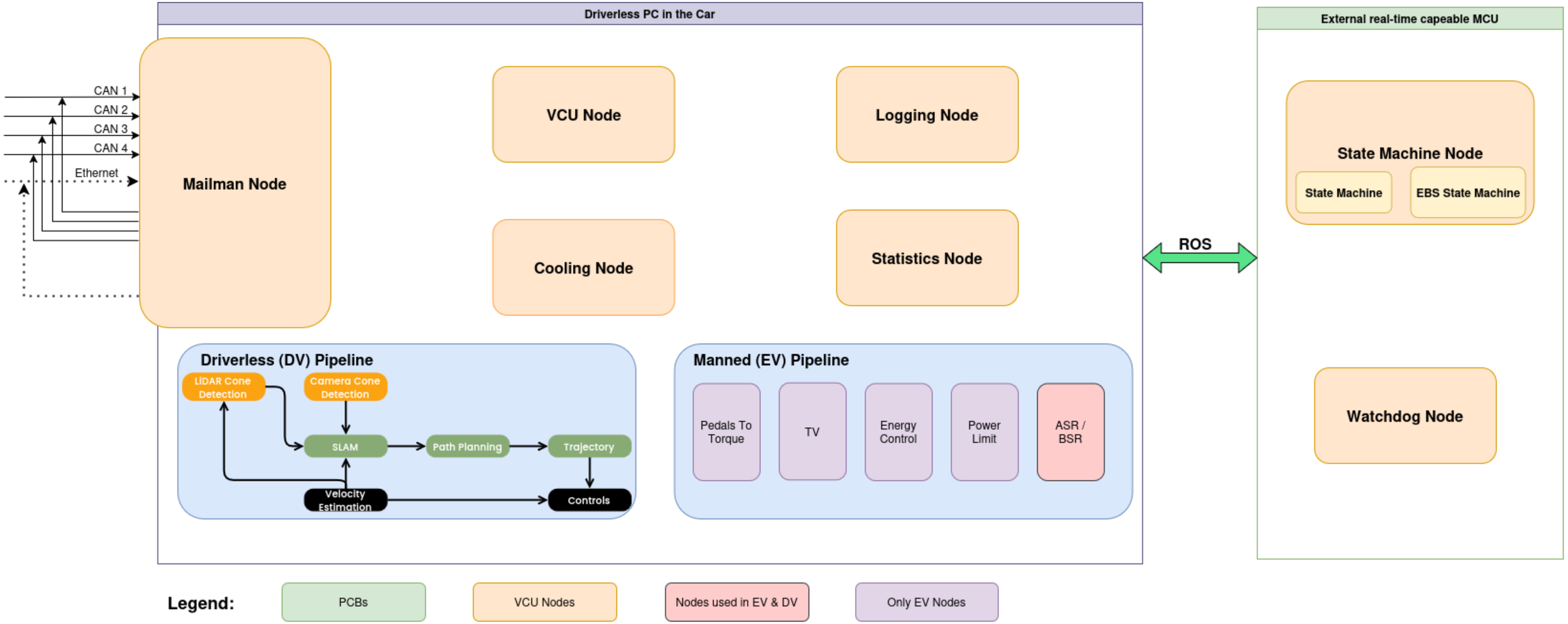


Elbflorace Example: Hardware & OS

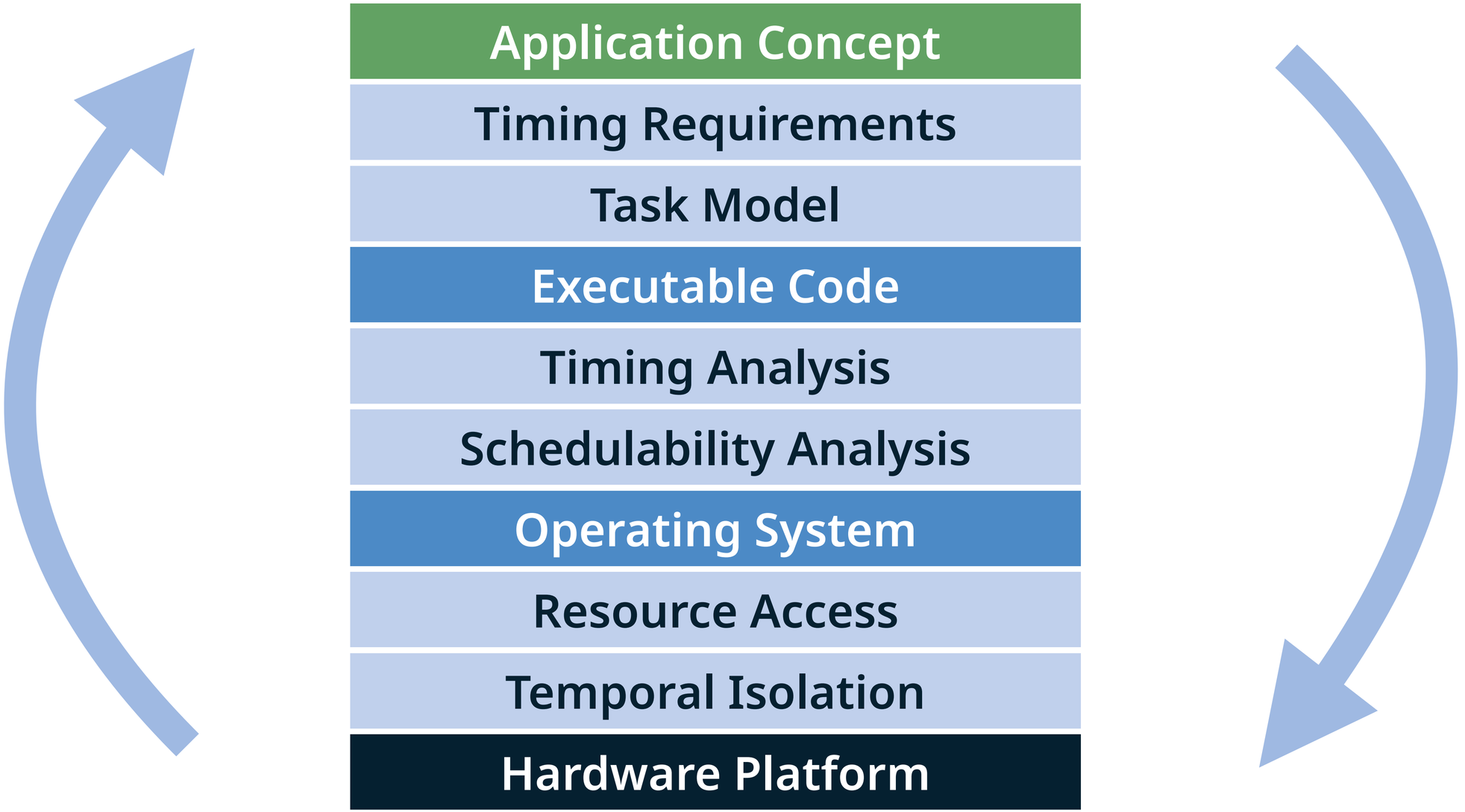
- vehicle operations software is important for safety
- but not all of it: two nodes handle all safety features
- trigger safety shutdown in case of problems



Possible Implementation: Split Vehicle Operations



Real-Time Systems Development Flow



Design Process

- modelling of application workload and desired outcome
  - workload: resource requirements like needed execution time
  - outcome: deadlines or other (hard/firm/soft) timing requirements
- workload depends on software (algorithm) and hardware (CPU speed)
  - compiler maps high-level code to machine code (average case vs. WCET)
  - may involve algorithmic flexibility (approximation)
  - may involve other (shared) resources
- scheduling to combine multiple applications
  - test for feasibility (mathematical analysis)
  - allocate resources (OS scheduling API)
  - enforce allocations

Computer Science Areas

Real Time Systems Are Cross Cutting

- computer architecture
- low level I/O: busses, caches
- parallel execution
- fault tolerance
- software engineering
- programming languages
- modeling techniques
- operations research
- computer science theory

Course Overview

What the Application Needs: Timing Requirements

- definitions, task models, terminology

Does it Work: Timing and Scheduling Analysis

- time-driven systems, event-driven systems
- scheduling analysis, admission
- resource access protocols

How to Enforce: Isolating Multiple Applications

- real-time operating systems
- clocks and event ordering
- advanced hardware: multicore, caches