Platform Software for Safety-Critical Multicore Systems

Echtzeitsysteme 09.01.2023

Dr. Isabella Stilkerich

SCHAEFFLER

Overview

Systems and Software Development

Architecture Goals

Dependability and Functional Safety

Real-Time and Concurrency

Overview

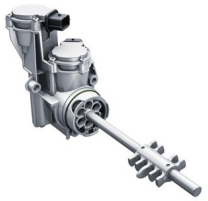
Systems and Software Development

Architecture Goals

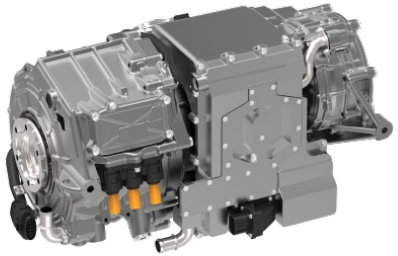
Dependability and Functional Safety

Real-Time and Concurrency

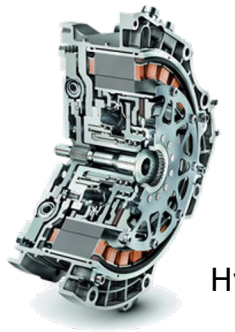
Example: Schaeffler's Embedded Systems



Gearbox actuator



eAxle



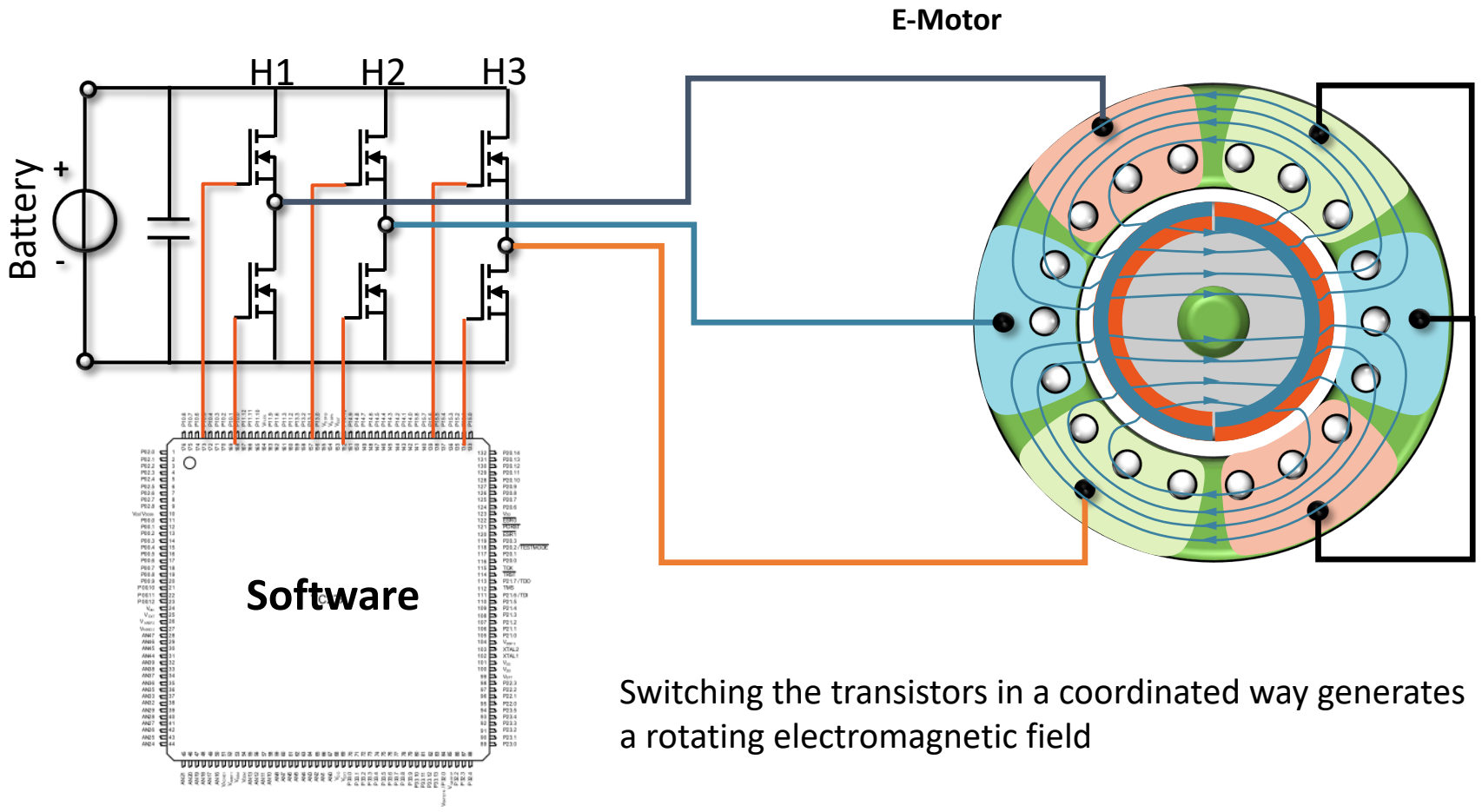
Hybrid Module

Active Roll-Stabilizer

E-Wheel Drive

A wide range of these applications use an embedded system for e-motor control

Embedded System E-Motor Control



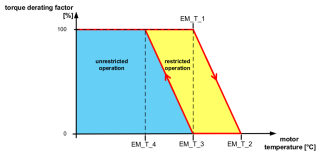
Switching the transistors in a coordinated way generates a rotating electromagnetic field

Functional Features

Motor types

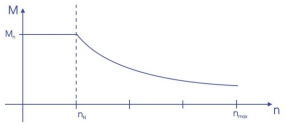
- Permanent magnet synchronous motor
- Asynchronous induction motor

Derating and Diagnostics



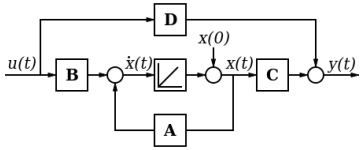
- Self protection and fault detection
- Performance derating

Electric current control



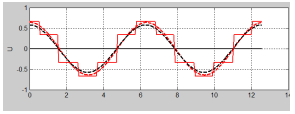
- Field oriented control
- Feed forward, magnetic saturation, reluctance
- Field weakening control
- (Over-)modulation schemes and variable switching frequencies

Sensors and Observers



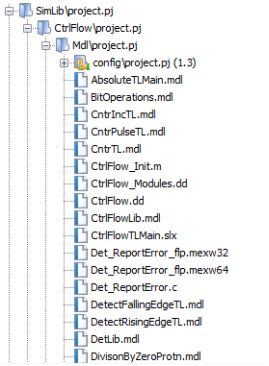
- Angle tracking observer
- Power loss and temperature estimation
- Magnetic flux in stator windings

Superimposed controllers



- Speed (window) control
- Jerk control

Libraries for various utilities



- Table lookup and interpolation
- Numerical routines
- Signal filters

Overview

Systems and Software Development

Architecture Goals

Dependability and Functional Safety

Real-Time and Concurrency

Variability: Software-Platform Approach

E-Motor Control SW Platform

Mechatronic
Project #1

Mechatronic
Project #2

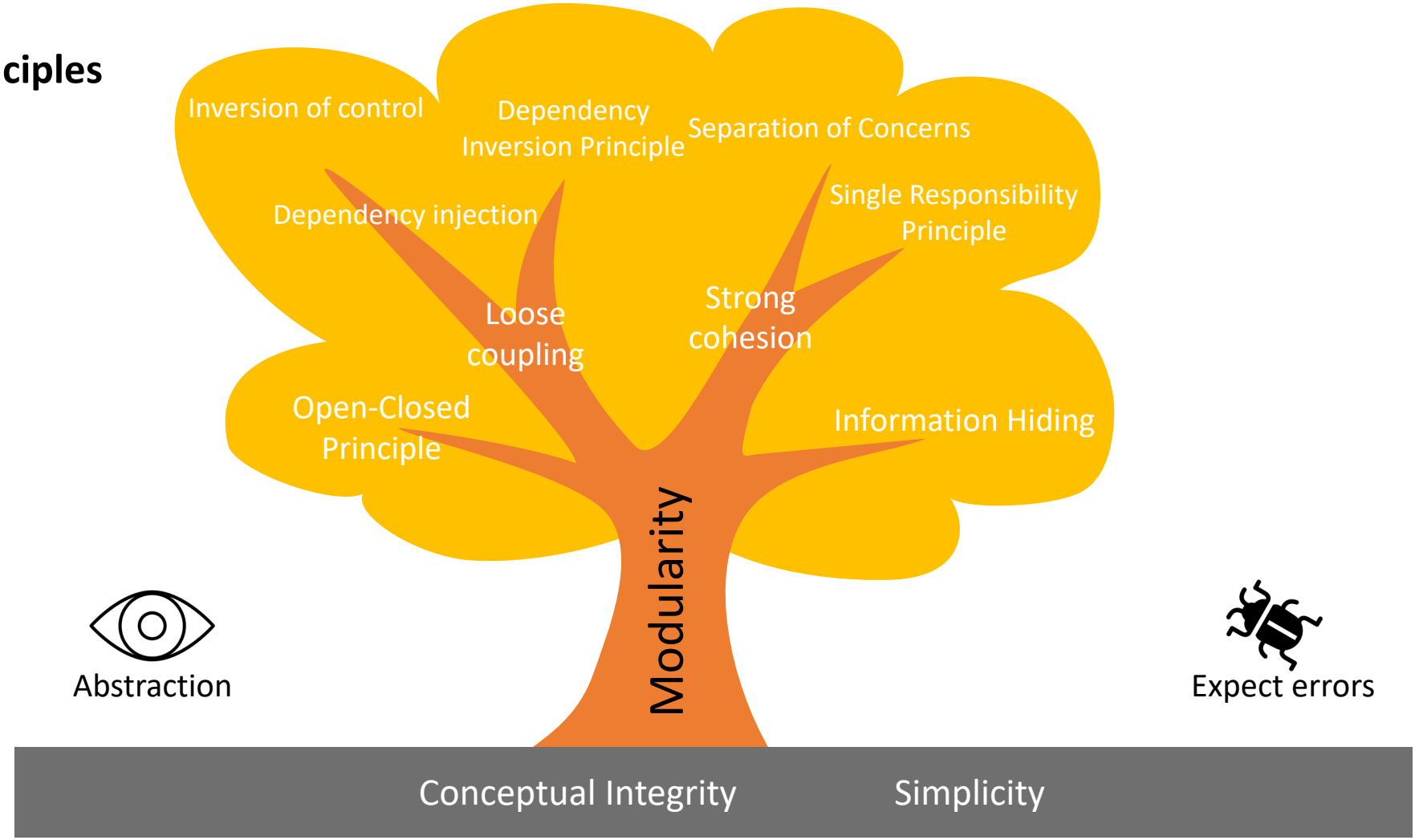
Library functions are developed using Matlab / Simulink

Important Qualities: Architecture Goals

- High intelligence and complexity of the control software (selected of qualities):
  - Functional correctness: torque precision, dynamics, safety
  - Performance efficiency: time behavior, resource utilization, energy
  - Reliability: availability
  - Security
  - Portability: variability, adaptability
  - Maintainability
- Qualities are often cross-cutting concerns
- Technical constraint: Use of AUTOSAR (Automotive Open System Architecture)

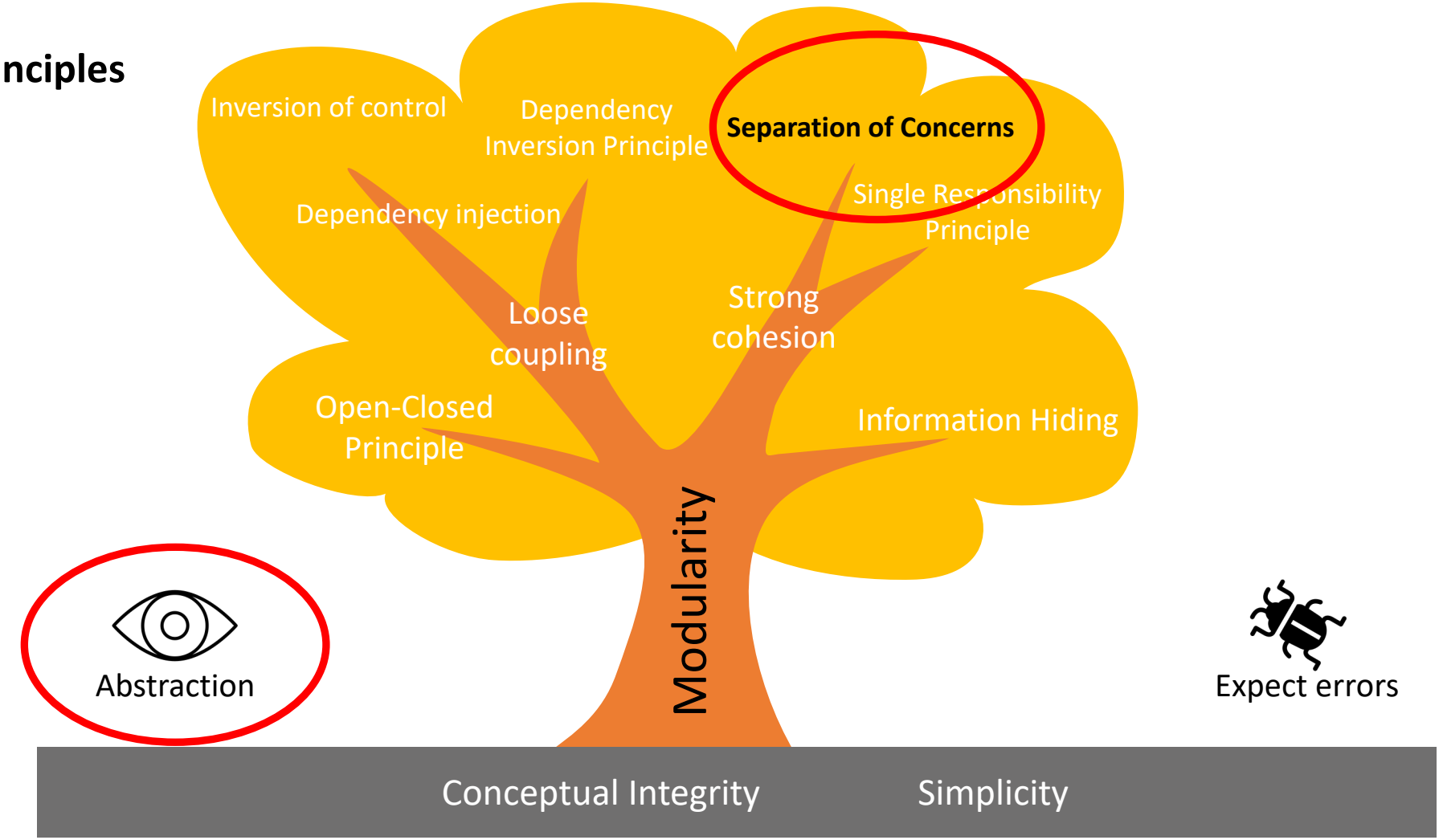
How to address these complex topics?

Design Principles



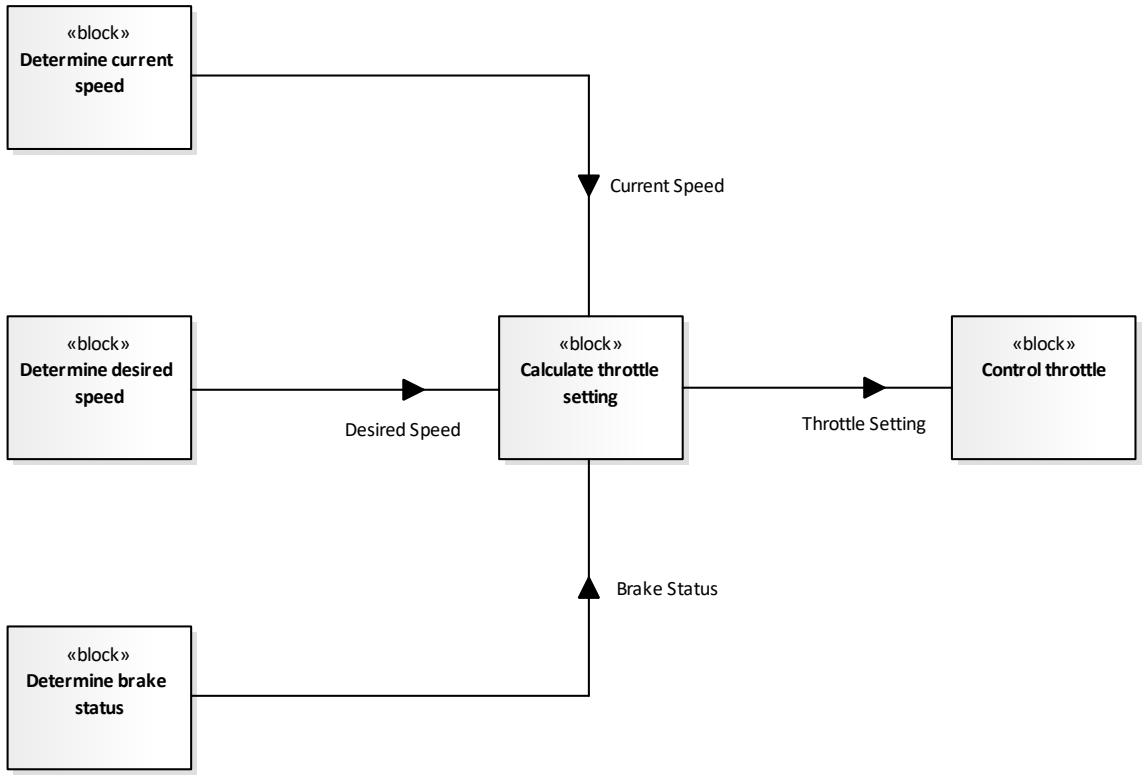
How to address these complex topics?

Design Principles



Functional Architecture (1)

Example: Cruise Control

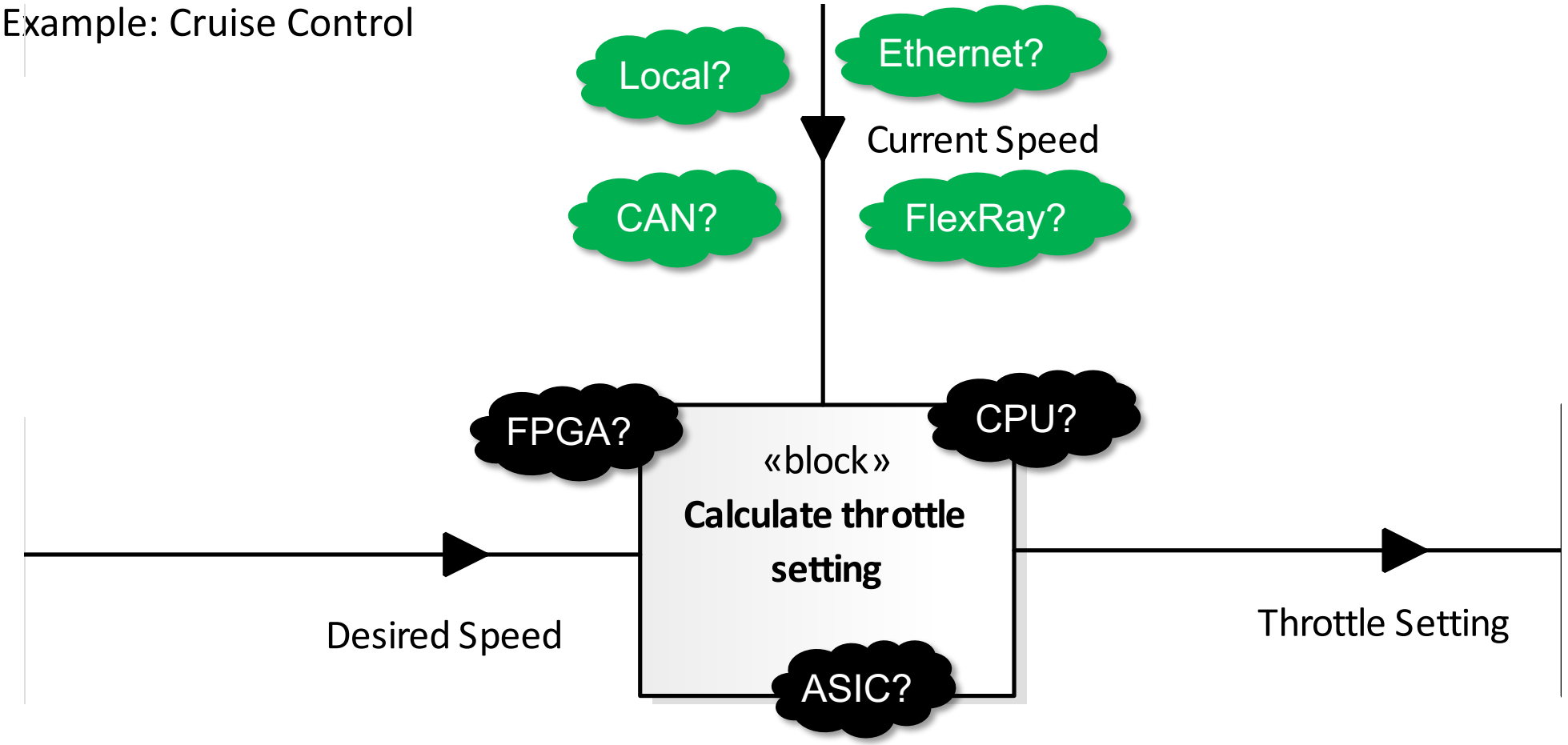


Functional Architecture (2)

- A **functional architecture** represents knowledge about the core function logic:
  - **Central concepts** of the core functional logic, their **attributes** and **relationships**
  - Enables a better **understanding** of the function logic
  - Establishes a **common language**
  - Helps to detect **inconsistencies** and **redundancies**
  - Builds the connection to requirements engineering (cf. domain models)
- Functional architectures **abstract from technical aspects**
  - The core functional architecture is **independent of technical concerns** and has an **independent life cycle**
- Modeling includes **structure** (e.g. representation of concepts and their relationships in a class diagram) and **behavior** (e.g. modeling of interaction of structural elements in a sequence diagram)
- **Experts** for **functional architectures** are often **not software developers**, but experts for electric motors, physicians, …

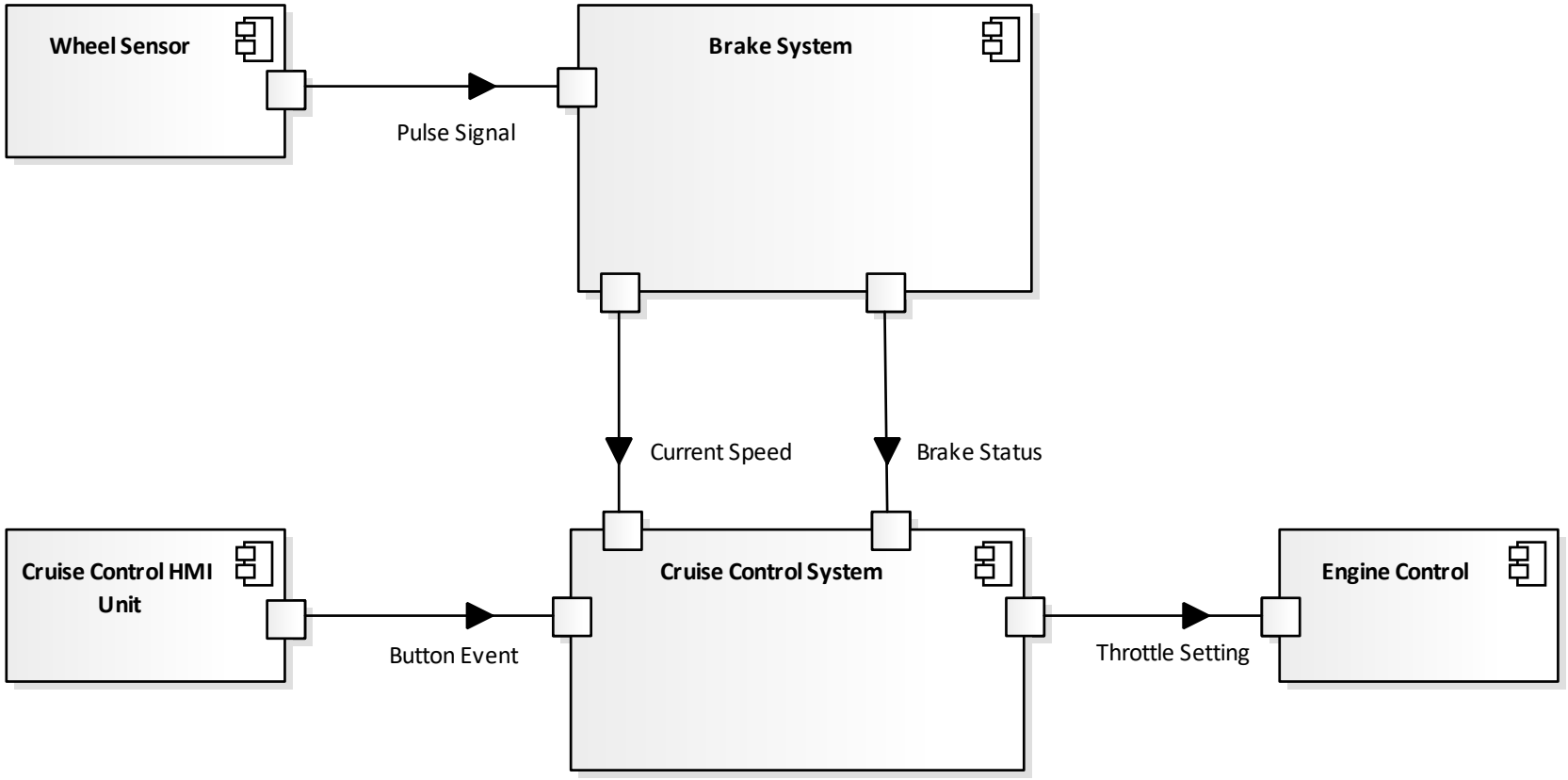
Functional Architecture (3)

Example: Cruise Control

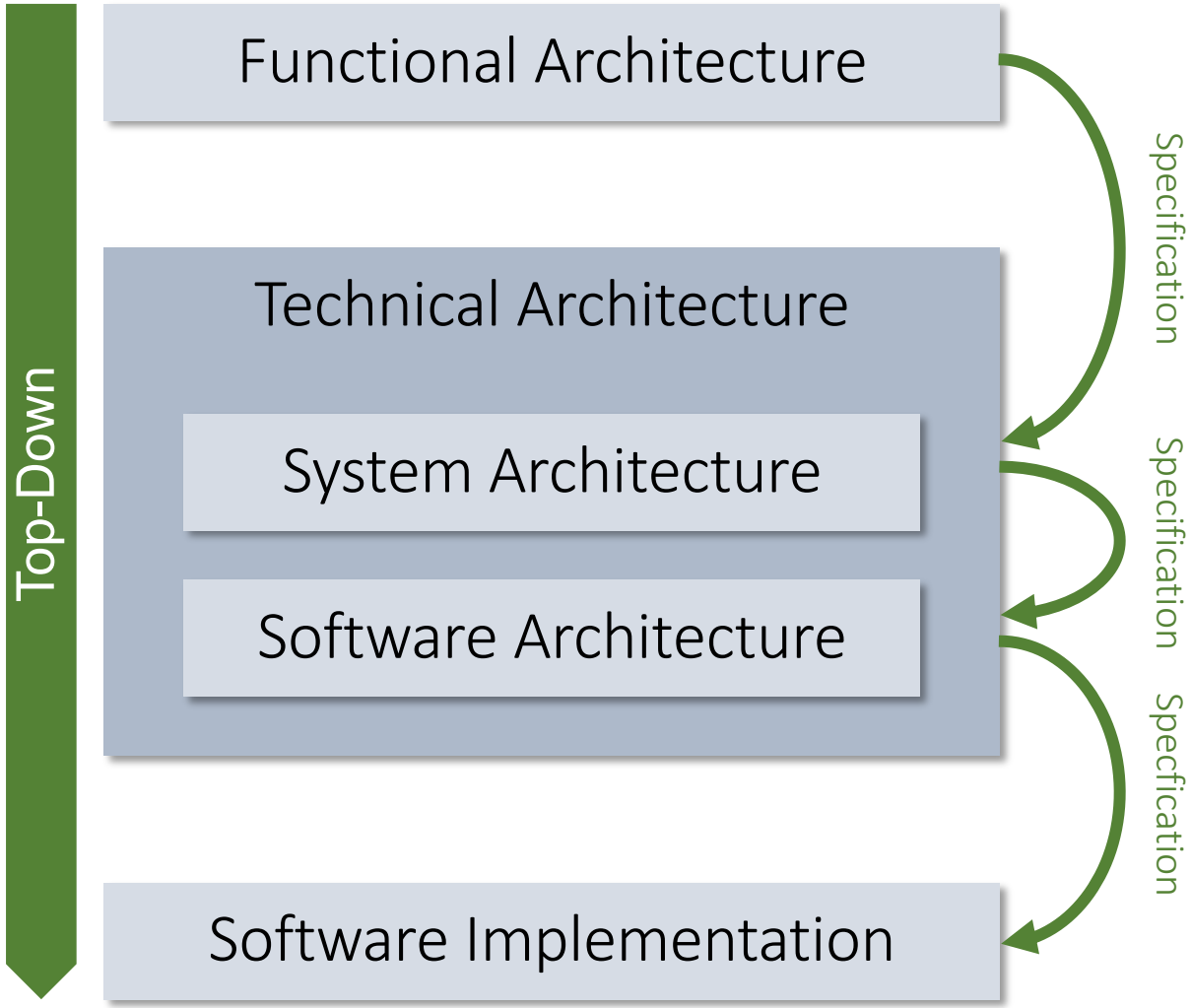


Technical Architecture (First Sketch)

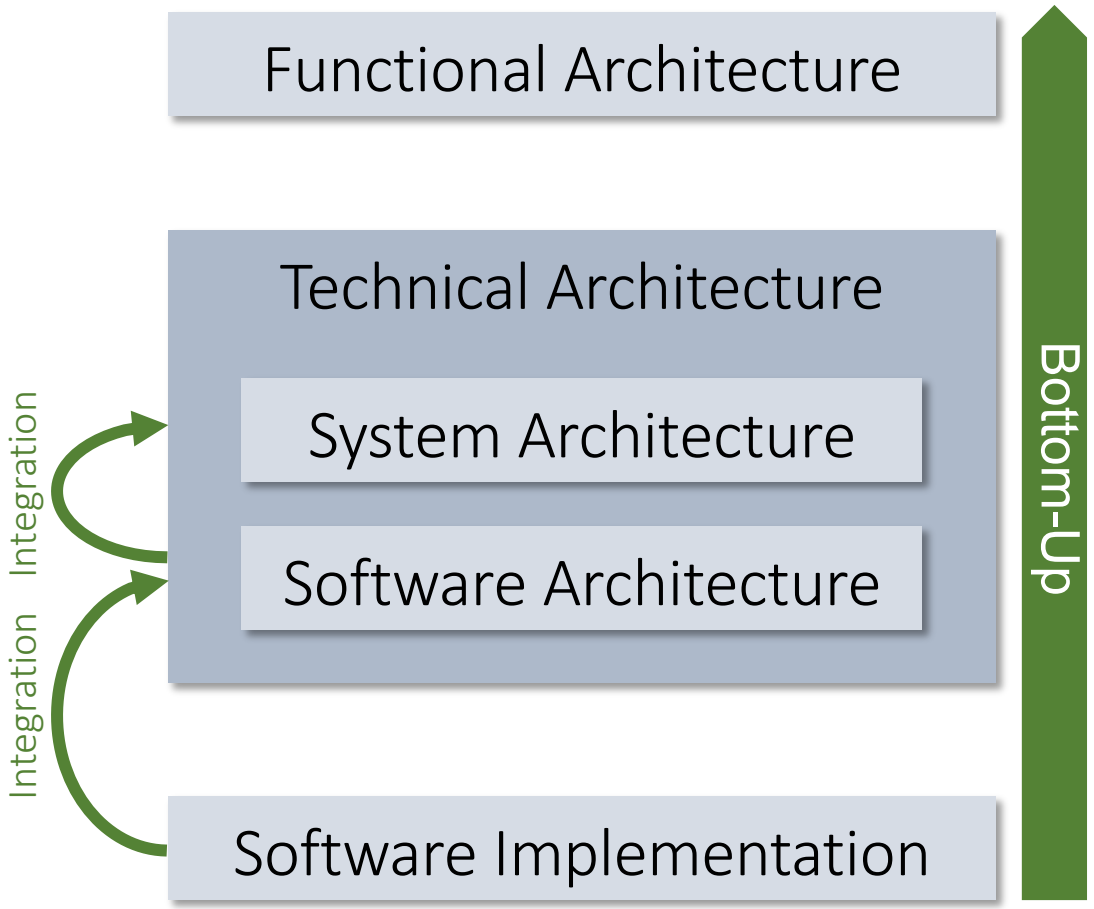
Example: Cruise Control



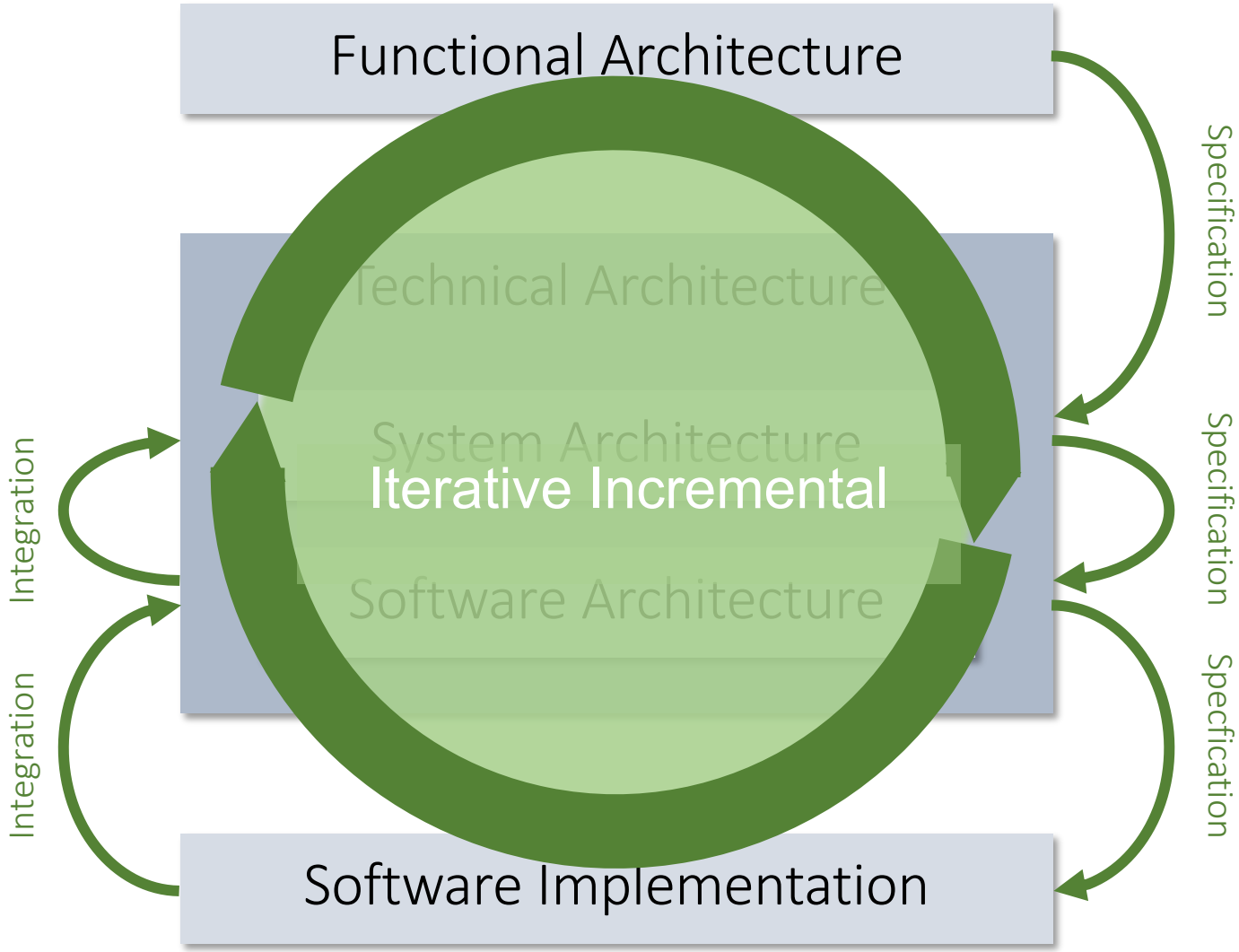
How to Construct a Dependable Embedded System?



How to Construct a Dependable Embedded System?



How to Construct a Dependable Embedded System?



Overview

Systems and Software Development

Architecture Goals

Dependability and Functional Safety

Real-Time and Concurrency

Example: Architecture Goals

Functionality, safety, real-time behavior: Alignment of design goals

- Functionality often benefits from methods applied in the context of safety-relevant systems, e.g., isolation and real-time properties
- Safety mechanisms should not just be „mounted on top of functionality“

Properties such as timing, memory usage and safety are a cross-cutting system aspect

- They have to be respected at all system, hardware and software levels
- The engineering disciplines rely on each other, they are equally important
- Properties should be included in the design process just as any other functionality or relevant property

Isolation in ISO 26262: Freedom from Interference (FFI)

From ISO26262-6, Annex D

- Software elements must not affect each other in an unintended and negative way
  - Errors in an application shall not spread to other applications
  - Errors in an application shall not spread to infrastructure services
  - Errors in an application shall not affect other system elements
- Elements subject to decomposition must be isolated from each other

Achievement of FFI

- Timing and execution: Temporal isolation: Scheduling, execution budgets, watchdogs, ...
- Memory: Spatial isolation: Semantic analysis, memory-protection unit, ...
- Safe exchange of information: Communication between isolated elements: checksums, ...

FFI in Space and Time

Physical isolation of software instances (e.g., independent MCUs): **Federated architecture**

All resources (memories, CPU time, etc.) can be assigned to a specific functionality

Often, functionalities need to cooperate, they have dependencies

- Safe data exchange between components
- Waiting times / latencies have to be respected in system design, etc.

Functionalities may also be deployed on the same MCU: **Integrated architecture**

- To reduce physical weight and size as well as costs
- Complicates the provision of FFI
- In contrast to physically isolated components, sophisticated mechanisms are needed for FFI

Overview

Systems and Software Development

Architecture Goals

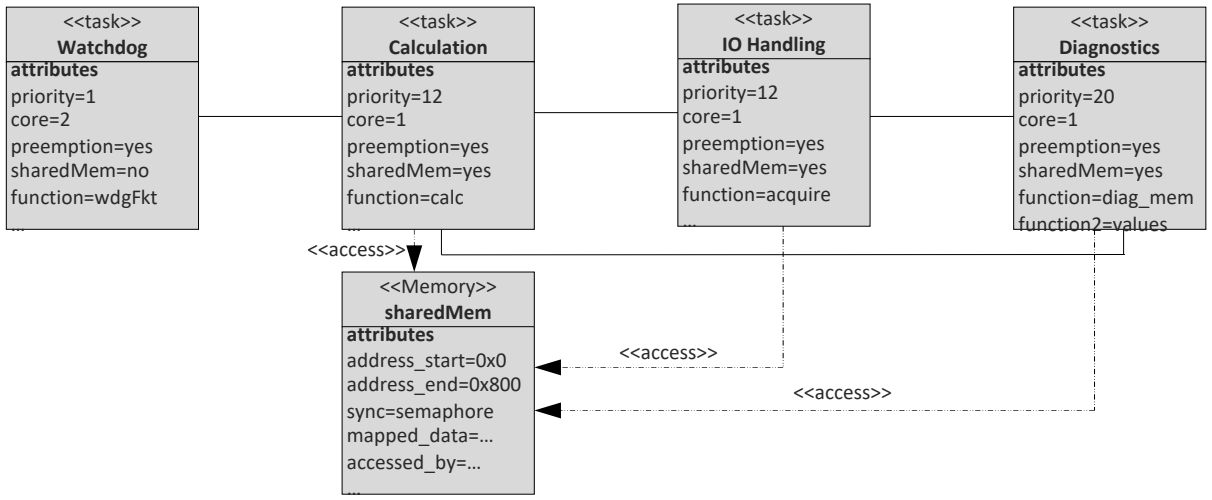
Dependability and Functional Safety

Real-Time and Concurrency

Temporal and Spatial Isolation: A Software Topic Only?

CPU time and memory must be shared across components

- CPU time sharing can be achieved by the use of an RTOS scheduler
- A scheduler provides a **framework** for the construction of a real-time system
  - An unfortunate application structure may impede timely execution
  - A proper thread / task architecture has to be created
- Memory partitions and their locations have to be defined, data and code has to be assigned



Temporal and Spatial Isolation: A Software Topic Only? No!

Scheduling and isolation are system-architectural topics:

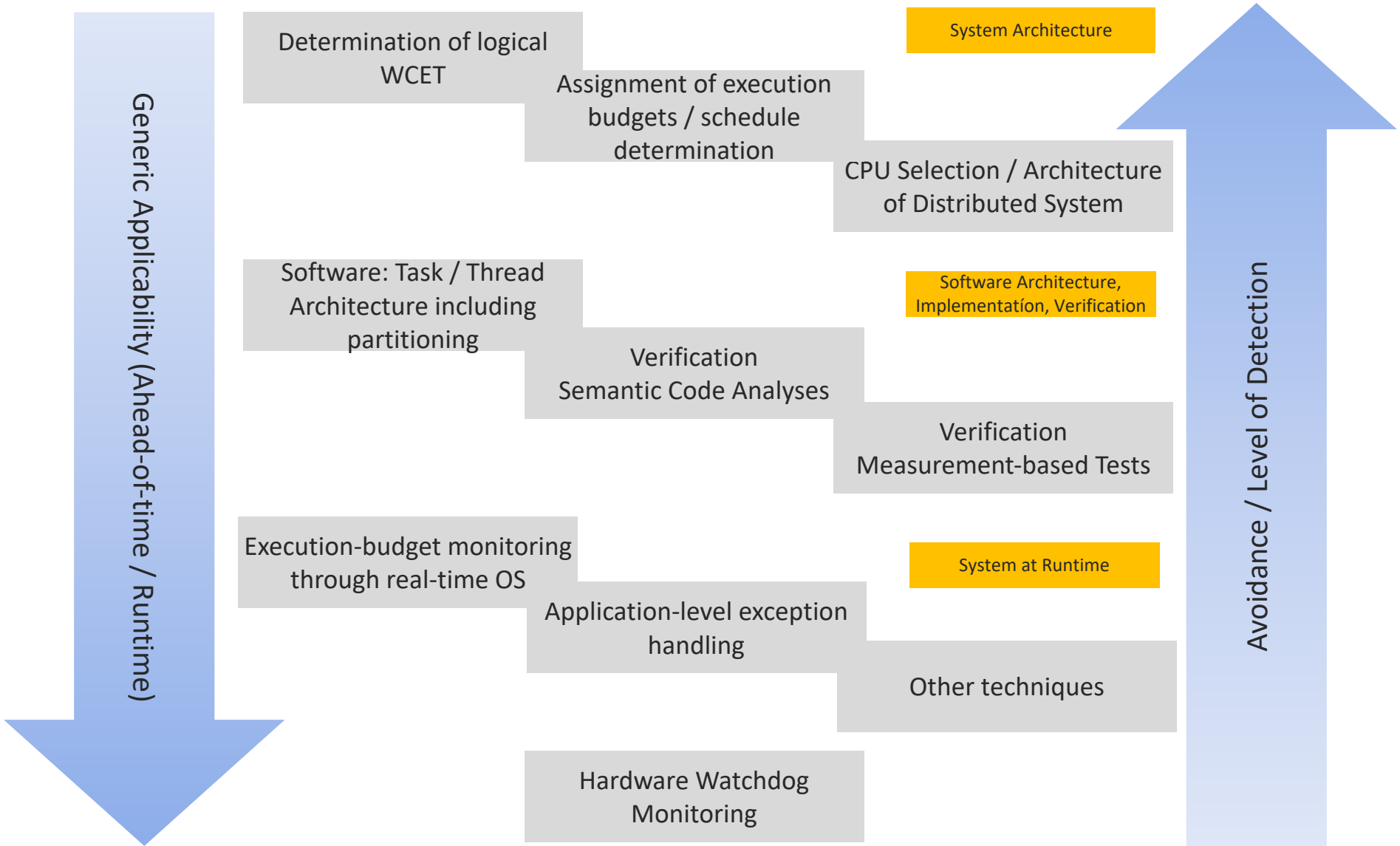
- The temporal /spatial **partitioning** is dependent on the **system requirements / architecture**
  - Mathematical **scheduling analyses** are performed on both **functional and technical architecture, e.g., rate-monotonic analysis (RMA)**
- CPU selection
- Distributed network of MCUs, etc.
- Aspects at all system-architectural levels influence each other

Example: Temporal Constraints, Computational Spacetime, Error Spreading

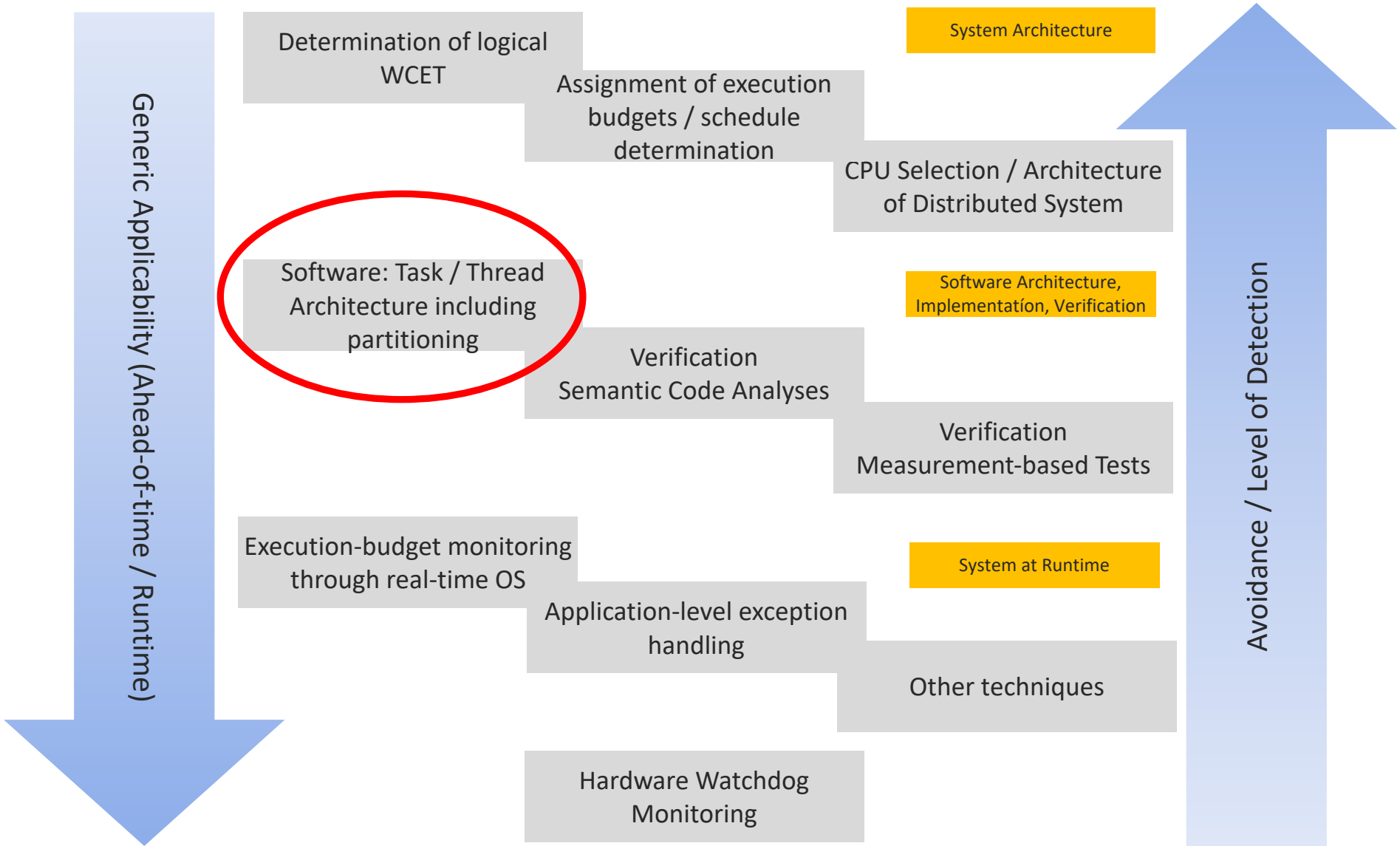
- Undesired memory accesses may induce temporal faults
- Unspecified or faulty sensor values may induce temporal faults
- A faulty design specification may induce temporal faults
- Measures (e.g., software-based replication) meant to provide safety
  - Affect timing behavior
  - May in turn induce temporal faults

The holistic solution has to be respected during analyses!

Mechanisms for Providing Timely Execution



Mechanisms for Providing Timely Execution



Separation of Concerns

Planning of temporal handling and dispatching of threads

1. Scheduling is the planning **strategy**
   - Construction of a thread-execution plan, which defines the order thread processing; statically at design time or dynamically at runtime
2. Dispatching is the thread-management **mechanism**
   - Implementation of the thread-execution plan
   - Overhead depends on **thread type** (process, user-level, kernel-level, i.e., memory-protection-zone assignment) being used

Scheduling at the Implementation Level

- Scheduling deals with the determination of **points in time** at which **work units** are executed on a **particular processor**
- Scheduling is a two-phase approach
  1. Work units have to be assigned to threads (statically at design time) — Software Architect
  2. Threads have to be assigned to processors (statically / dynamically) — Software Architect / Operating System

An Engineering Framework for Generic Application Software

We have been implementing an engineering framework that will support us in building our Software Platform according to the construction-kit approach by cross-architectural space-time analyses:

- **Schedulability analyses** at the level of the **functional / technical architecture**
- **Spatial isolation** specification at the level of the **functional / technical architecture**
- Hybrid semantic and dynamic **timing analyses** at the **binary-code level**
- Semantic **reachability** and **scope analyses** at the **source-code level**

In this way, we both support **correctness** and **safe / efficient mapping of tasks to multicores and data / code to physical memories**

Thread of Control (1)

- An OS thread / task is an abstraction of the operating system provided to
  - programs from the application layer
  - infrastructure-software programs (e.g., drivers)
- A thread executes (parts of a) program(s) and is a modelling element in a software architecture
  - The thread-architecture view is defined by the architect
    - Thread structure (relations, dependabilities)
    - Assignment of properties: priority, preemption, events
    - Assignment to memory-protection zones (address spaces)

Thread of Control (2)

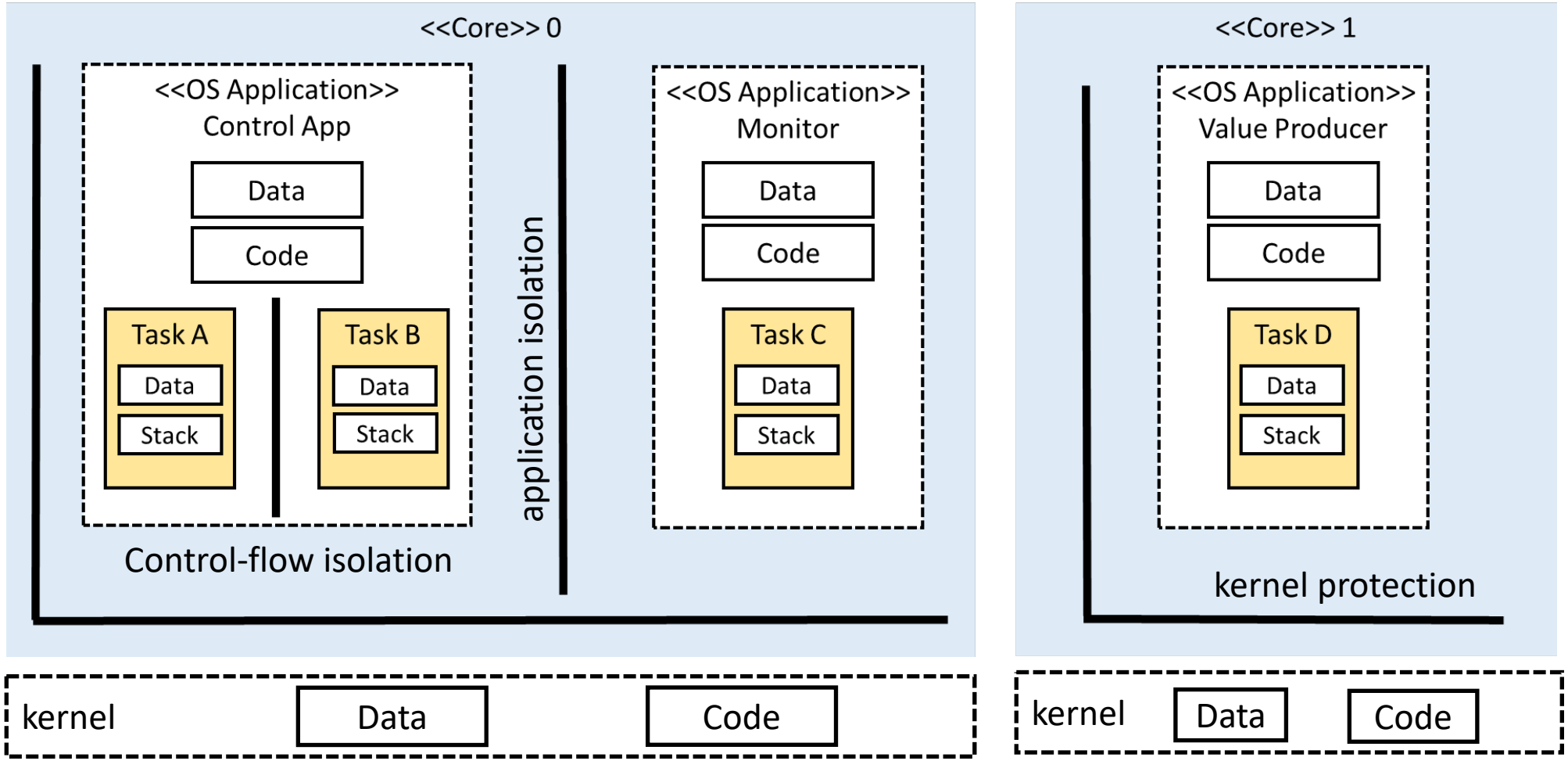
This approach is a realization of the separation-of-concerns principle

- Separate what (code) from how (execution)
- An OS partially encapsulates the architecture goal *timing behavior* in a software architecture
- Supports code **reusability** and **extensibility** (in contrast to (manually applied) Cyclic Executive Pattern)

The thread-management overhead of the OS depends on the thread-architecture

- Single-threaded program
- Multi-threaded program
  - Single address space
  - Isolated OS kernel
  - Multiple isolated address spaces

AUTOSAR OS for FFI: Memory-Protection Zones



ASSIST: Scheduling at the Level of Functional Architectures

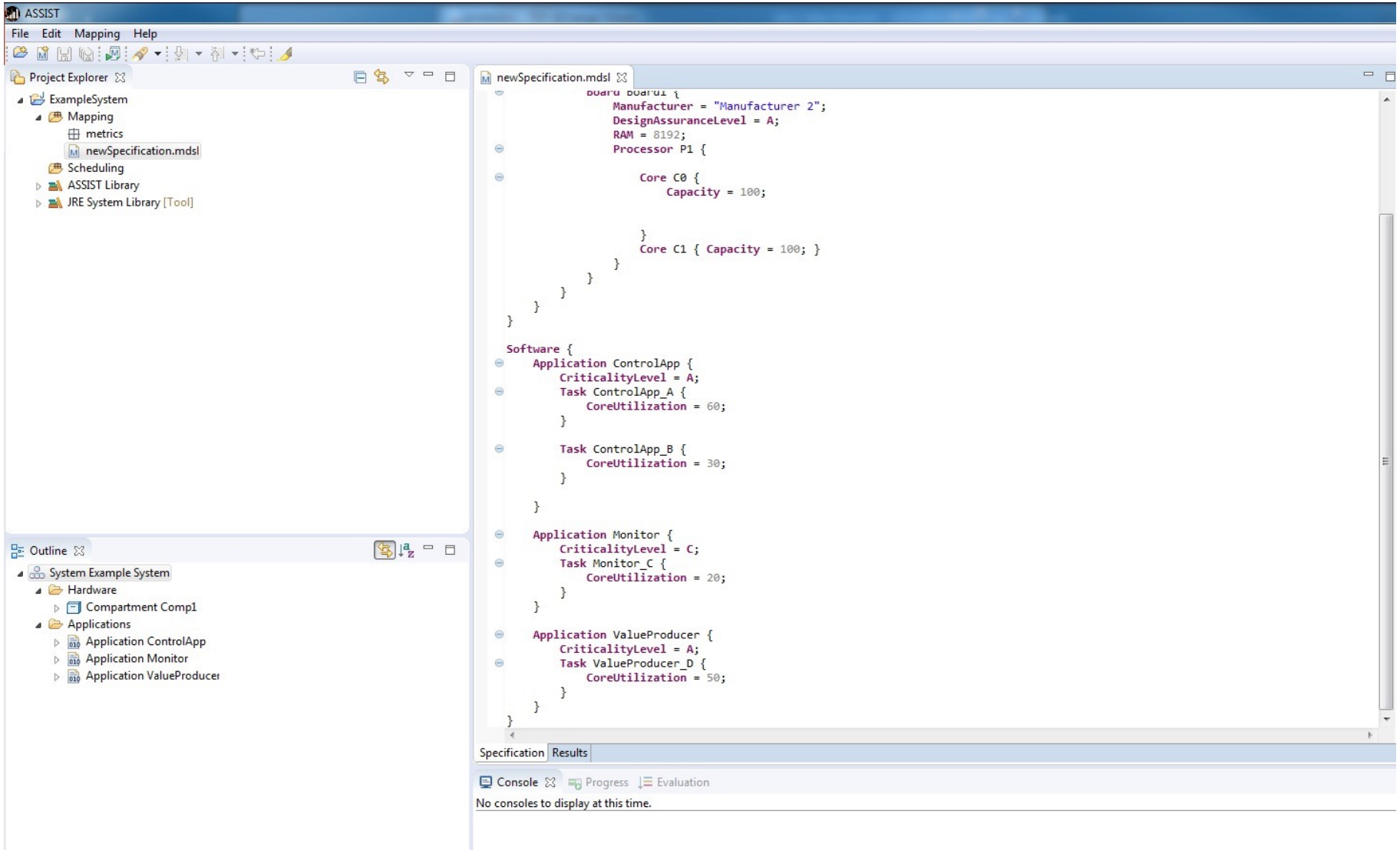
How to derive a technical architecture from a functional architecture?

Define constraints for the technical architecture

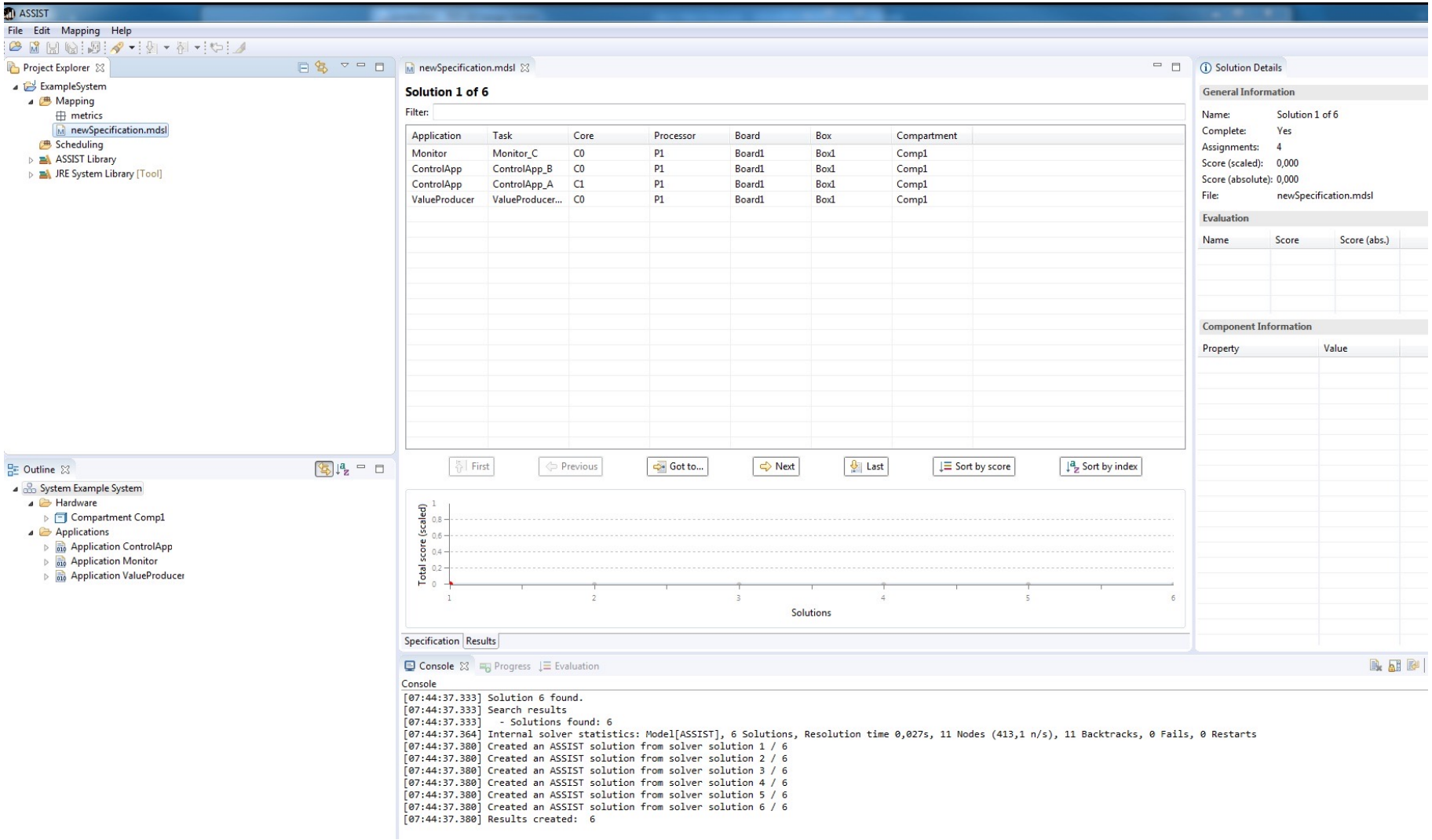
- Hardware resources
- Temporal isolation and other timing properties
- Spatial isolation
- etc.

Solve Constraint Satisfaction Problem

ASSIST


ASSIST
File Edit Mapping Help
Project Explorer
ExampleSystem
Mapping
metrics
newSpecification.mdsl
Scheduling
ASSIST Library
JRE System Library [Tool]
Outline
System Example System
Hardware
Compartment Comp1
Applications
Application ControlApp
Application Monitor
Application ValueProducer
newSpecification.mdsl
Manufacturer = "Manufacturer 2";
DesignAssuranceLevel = A;
RAM = 8192;
Processor P1 {
Core C0 {
Capacity = 100;
}
Core C1 { Capacity = 100; }
Software {
Application ControlApp {
CriticalityLevel = A;
Task ControlApp_A {
CoreUtilization = 60;
Task ControlApp_B {
CoreUtilization = 30;
Application Monitor {
CriticalityLevel = C;
Task Monitor_C {
CoreUtilization = 20;
Application ValueProducer {
CriticalityLevel = A;
Task ValueProducer_D {
CoreUtilization = 50;
Specification
Results
Console
Progress
Evaluation
No consoles to display at this time.

ASSIST: Mapping



ASSIST is extended to output an initial operating-system configuration

Implementation-Level Analyses

- The actual implementation affects timing and memory-handling properties
- Therefore, the implementation has to be analyzed
  - Source code
  - Binary code
- Depending on the programming being used, these analyses differ
  - Type-safe languages are already memory safe and support the correctness of memory handling
  - Programs coded in languages (e.g. C) that have a weak type system may be analyzed to establish memory safety
  - C programs are predominant in the embedded domain
  - Semantic analysis and abstract interpretation in particular can be used to make C programs memory-safe

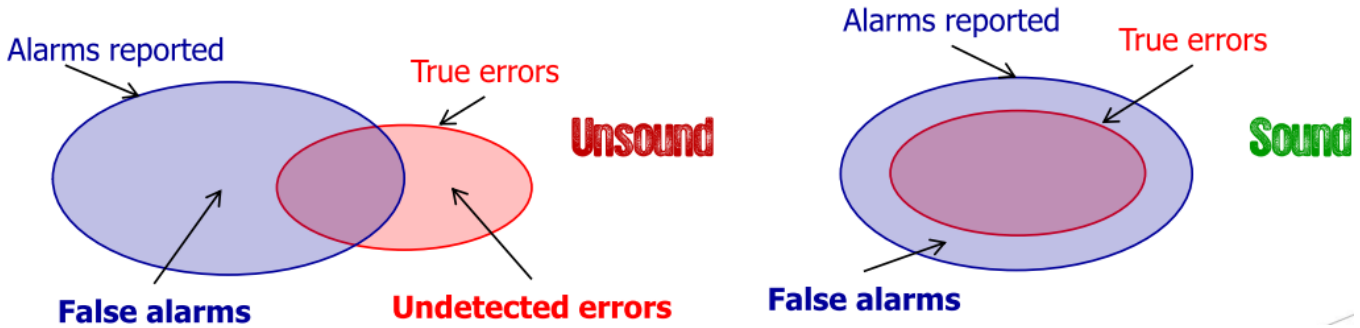
Semantic Code Analyses

Objective: Detection of runtime errors in programs (dynamic tests are often unsuitable)

- Sound vs. Unsound:
  - Unsound tools report only a subset of actual runtime errors (false alarms and undetected errors)
  - Sound tools reliably report supported runtime-error types (false alarms, but no undetected runtime errors) and prove their absence, accordingly

Quality of sound tools is measured by number of false alarms

- False alarms require manual analysis efforts
- High number of false alarms impedes efficient use during development („continuous verification“): Example for a sound tool is **Astrée**, which makes use of abstract interpretation



Retrofitting an Unsafe Language with Astrée

Astrée ensures memory safety by statically proving absence of certain types of runtime errors

- Invalid usage of pointers and arrays
- Invalid ranges and overflows
- Invalid shift argument
- Uninitialized variables
- Division or modulo by zero
- Failed or invalid directives
- Invalid function calls
- Data and control flow alarms
- Invalid concurrent behavior

Thus, we can establish memory safety in a C program! This, in turn, allows to

- Construct **spatial isolation realms** by logical separation of all global data
- Build **application-** and **hardware-tailored, safe memory management** using **Abstract Interpretation**

Schaeffler: KESO goes ASSIST and Astrée

KESO : Research project (2005-2017) in the domain of safety-critical Java (SCJ) Real-Time Specification for Java (RTSJ)

- Respect of system description and the operating-system model (AUTOSAR OS)
- Semantic analyses on type-safe code, e.g. reachability analyses to provide software-based spatial isolation and escape and region analyses for automatic memory management

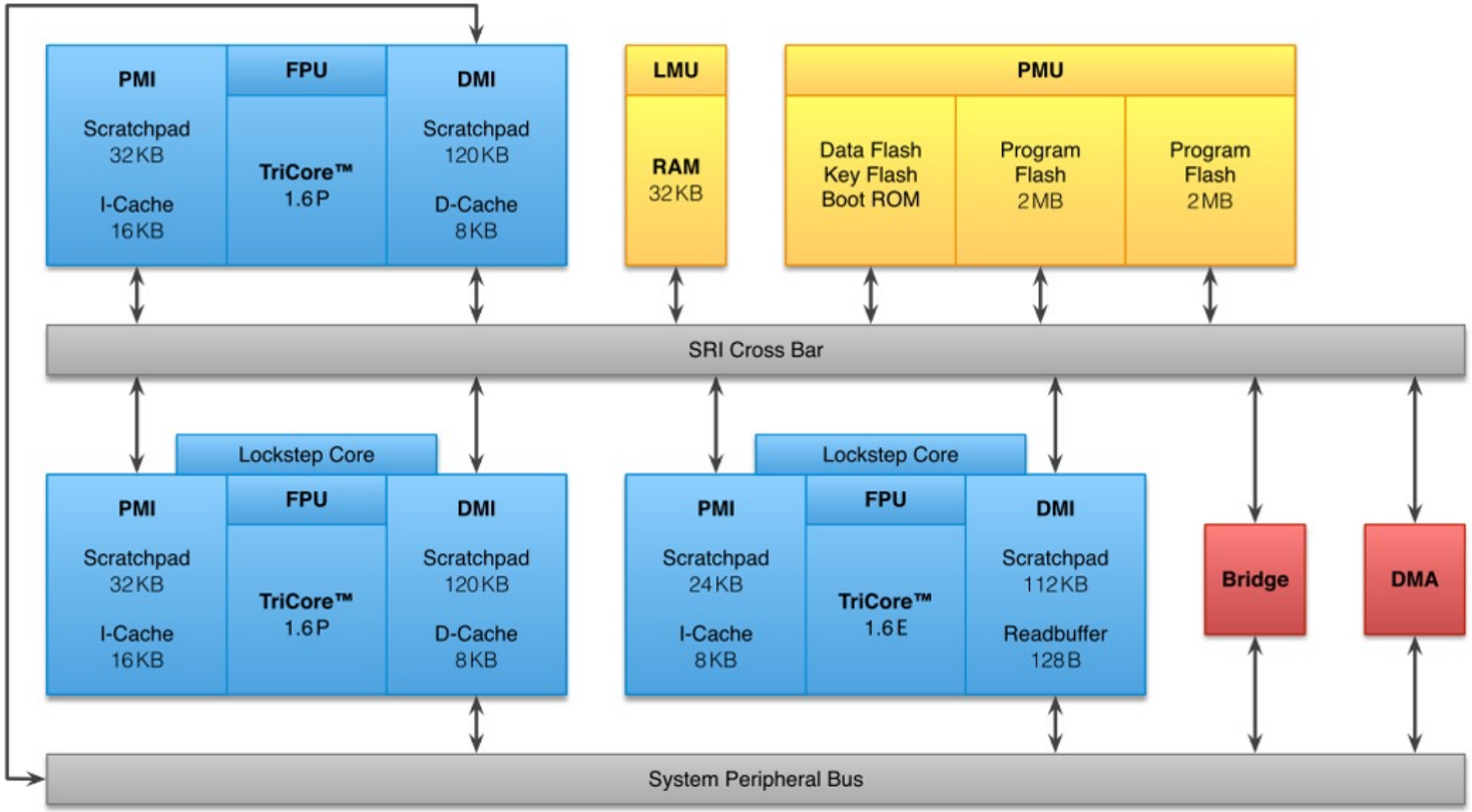
Astrée extensions being developed in cooperation with AbsInt:

- Respect AUTOSAR OS model
- Definition of spatial-isolation realms
- MCU‘s memory model
- Data classifications and assisted memory mapping
- Verification of synchronization mechanisms

Publications:

- ERTS 2020: Using Generic Software Components for Safety-Critical Embedded Systems - An Engineering Framework
- ERTS 2022: Whole-System Analysis for Memory Protection and Management

Respect the Memory Architecture: Infineon AURIX TC277



Hardware Specification with ASSIST

```
Hardware {
    Compartment Comp1 {
        Manufacturer = "Manufacturer 1";
        Box Box1 {
            Manufacturer = "Manufacturer 1";
            Board Board1 {
                Manufacturer = "Manufacturer 2";
                DesignAssuranceLevel = A;
                Processor P1 {
                    Manufacturer = "Infineon";
                    Type = "AURIX";
                    Provides 32768 of exclusive feature "LMU RAM";
                    Provides 4194304 of exclusive feature "PMU Program Flash";
                    Core C0 {
                        Architecture = "TriCore 1.6 P";
                        Provides shared feature "Performance";
                        Provides shared feature "FPU";
                        Provides 16384 of exclusive feature "I-Cache";
                        Provides 8192 of exclusive feature "D-Cache";
                    }
                    Core C1 {
                        Architecture = "TriCore 1.6 P";
                        Provides shared feature "Performance";
                        Provides shared feature "FPU";
                        Provides shared feature "Lockstep";
                        Provides 16384 of exclusive feature "I-Cache";
                        Provides 8192 of exclusive feature "D-Cache";
                    }
                    Core C2 {
                        Architecture = "TriCore 1.6 E";
                        Provides shared feature "Efficiency";
                        Provides shared feature "FPU";
                        Provides shared feature "Lockstep";
                        Provides 8192 of exclusive feature "I-Cache";
                        Provides 128 of exclusive feature "DMI Readbuffer";
                    }
                }
            }
        }
    }
}

Software {
    Application A1 {
        CriticalityLevel = A;
        Task A1_T1 {
            Requires shared Core feature "Lockstep";
            Requires shared Core feature "Efficiency";
            Requires 2000000 of exclusive Processor feature "PMU Program Flash";
        }
    }
```

ASSIST is used to define the hardware at the system-architectural level

- Information about task setup and isolation requirements is used to compute valid mappings

- Information on the memories is merely passed to Astrée

Apply KESO‘s memory handling to Astrée

Two-dimensional data classification

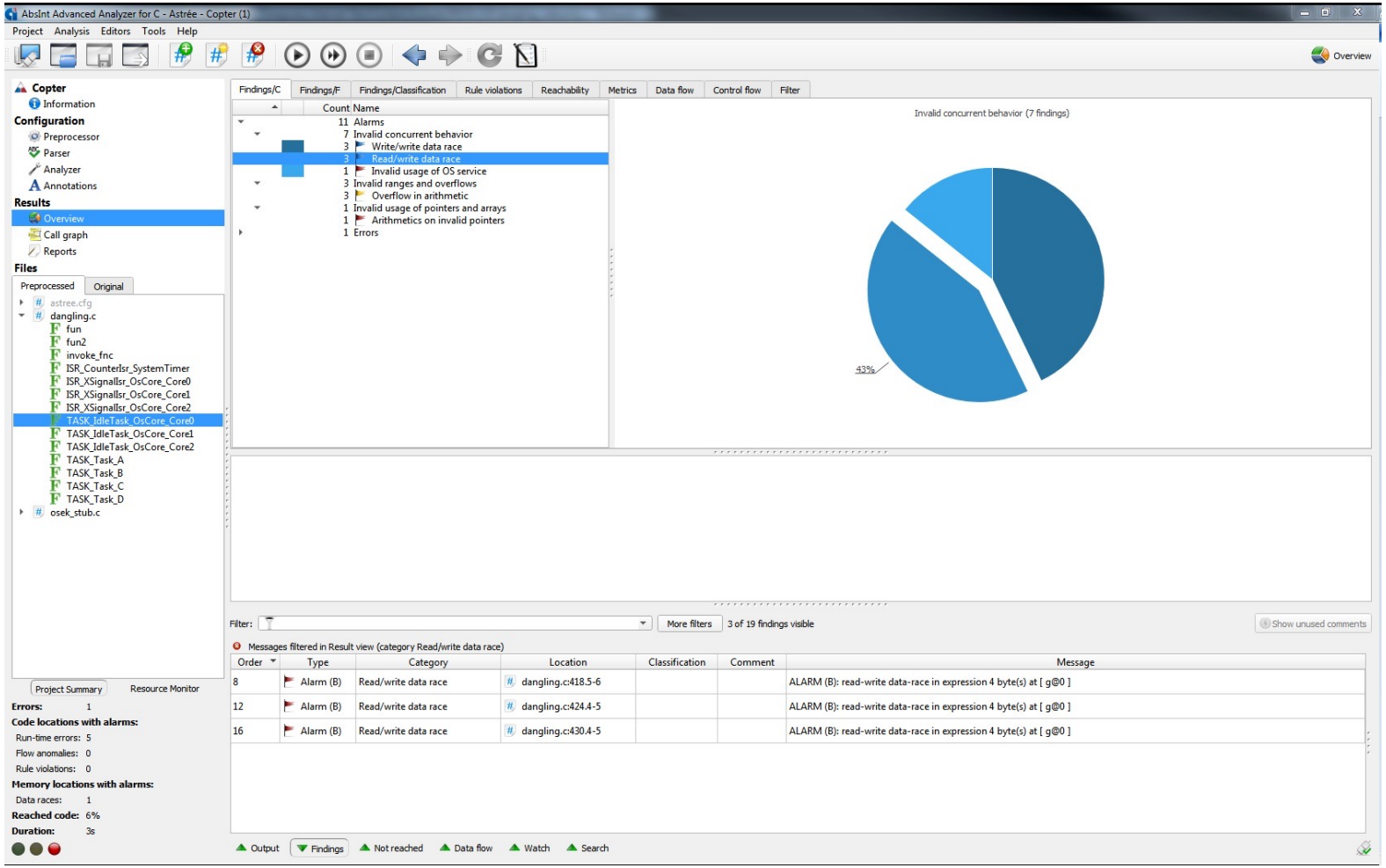
- Respect of program semantics: cf. RTSJ‘s memory model; stacks, data segments
- Extension for constant data
- Respect of the microcontroller‘s physical memory architecture

Data classes

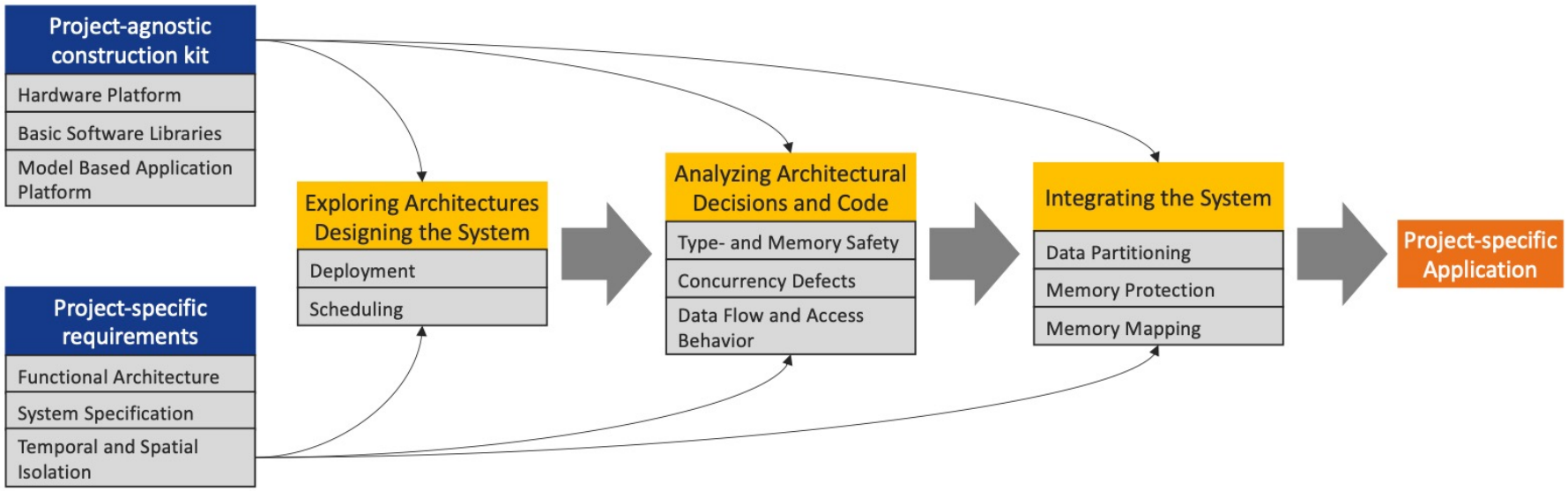
- Thread-local data, allocation in core-local memory, local access (e.g. stack assignment)
- Thread-local data, allocation in core-local memory, cross-core access
- Thread-global data, allocation in core-local memory
- Thread-global, core-global data
- Constant data
  - True constants (e.g. placement in flashes PF0 / PF1)
  - Runtime-constants (Calibration parameters)

For data classification, Astrée respects the OS‘s thread model and an application‘s OS configuration, the MCU‘s memories and the memory-safe C code

Astrée: Operating-System and Memory Scope Extensions



An Engineering Framework for Generic Application Software



Résumé

Construction of a Software Platform

- Generic applications developed using the Matlab / Simulink DSL
- ECU developed to construction-kit approach using a processor family
- Timing and memory analyses at the architecture and implementation level

Further reading:

Avoiding systematic faults in timing at the system-architectural level:

Using Generic Software Components for Safety-Critical Embedded Systems - An Engineering Framework

Real-Time Systems Lecture at Chair of Operating Systems, Computer Science Department at University of Erlangen-Nuremberg

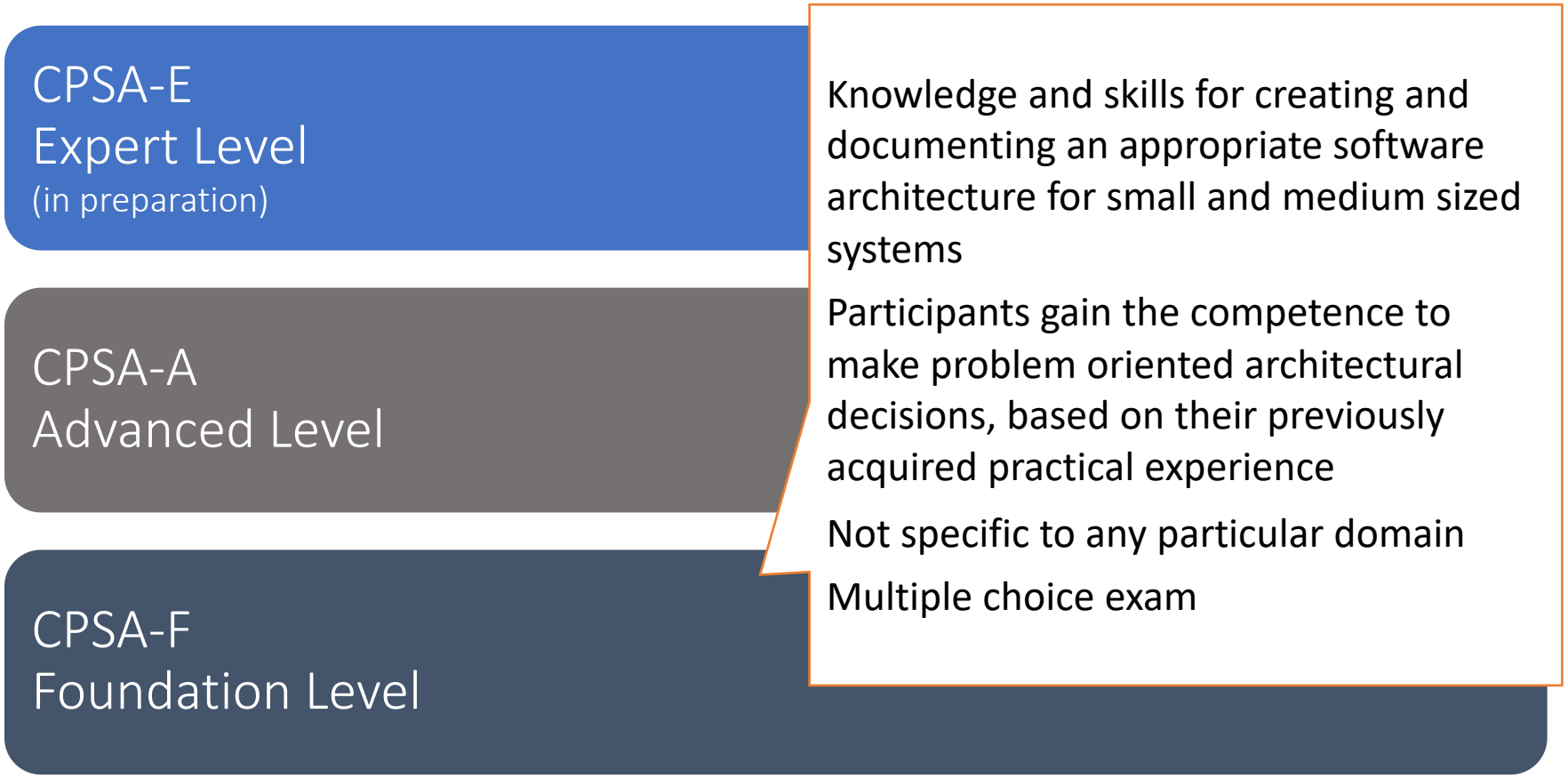
ARAMiS II Research Project

Astrée

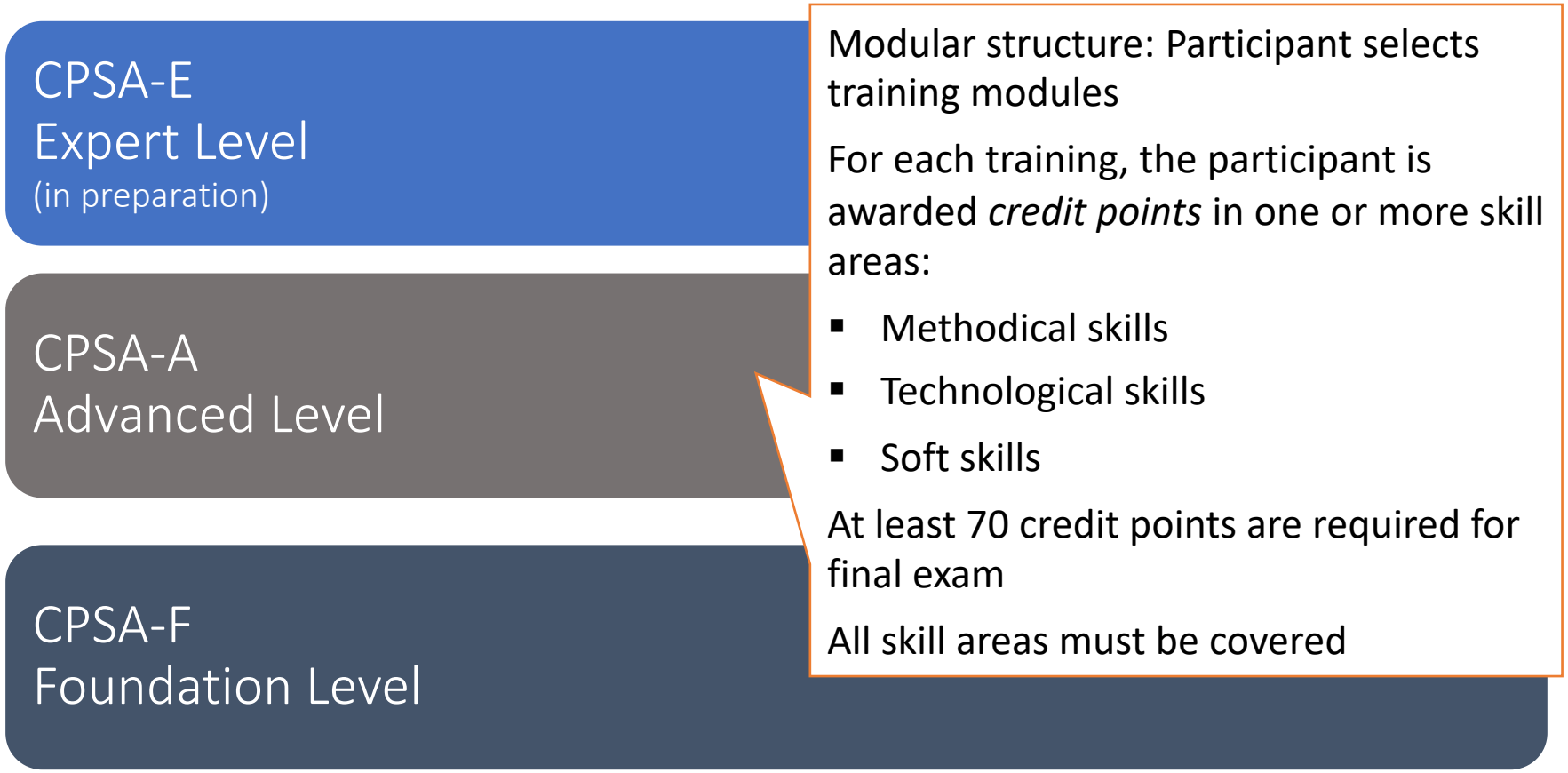
ASSIST

KESO

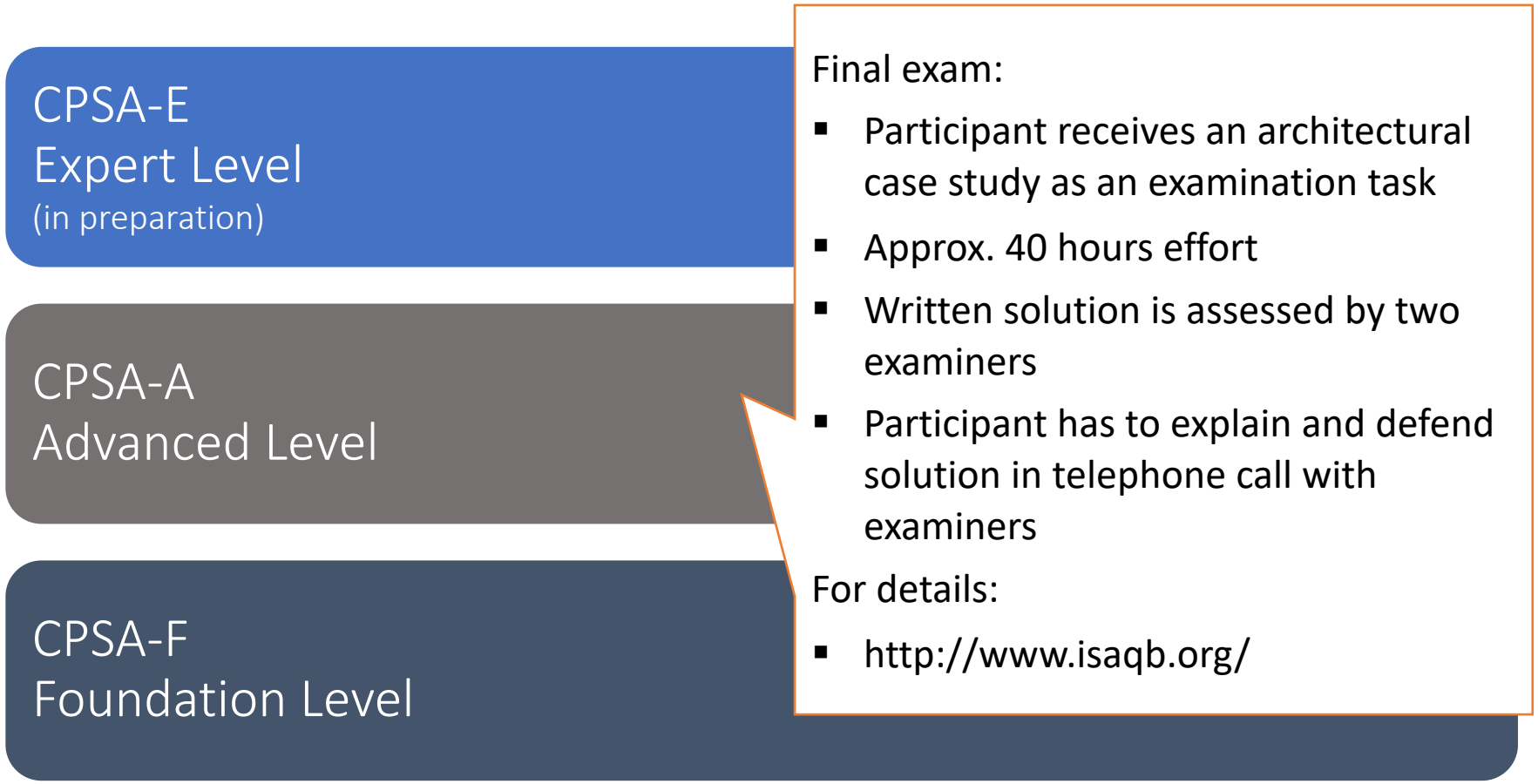
CPSA: Certification Program (1)



CPSA: Certification Program (2)



CPSA: Certification Program (3)



CPSA – Foundation Level

- Architecture in the development context
  - Stakeholders, organizational constraints
  - Role definition and assigning the architecture role
- Approach
- Requirements analysis
- Impact factors
- Design:
  - Approach
  - Principles
  - Models and views
  - Patterns and aspects
- Documentation and communication
- Quality and evaluation
- Implementation of architectures

https://isaqb-org.github.io/curriculum-foundation/curriculum-foundation-en.pdf

CPSA – Advanced Level: Dependable Embedded Systems

- System development for embedded systems
  - Importance of system architecture for key topics such as timing, reliability, and functional safety
  - Modeling and analysis of functional and technical architectures
- Software development for embedded systems
  - Software modeling for embedded systems
  - Approaches and paradigms for implementation and verification
- Reliability and functional safety
  - A systematic approach to the development of reliable and safety-critical embedded systems
  - Solutions at the architecture level
- Real-time and concurrency
  - Analysis of real-time requirements
  - Architectural solution approaches for real-time and concurrency
  - Analysis of real-time properties
- Adaptability and variability
  - Analysis and modeling of variability
  - Methods, principles, and solutions for adaptability and variability

https://isaqb-org.github.io/curriculum-embedded/curriculum-embedded-en.pdf