Platform Software for Safety-Critical Multicore Systems

Echtzeitsysteme 18.01.2022

Isabella Stilkerich

SCHAEFFLER

Agenda

Motivation: Software Platform Development for Motor-Control Systems

System Properties: Functional safety, real-time capability, computational space-time

Logical / Functional vs Technical Architecture

Technical Architecture

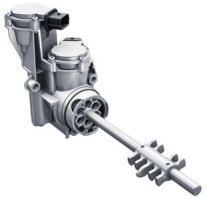
- Deployments: tasks and memory
- Multicore, memory mapping and isolation properties
- Scheduling and component deployment at the architecture level: ASSIST

Implementation Analyses

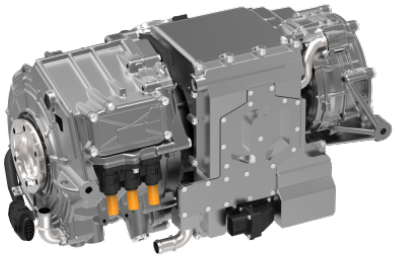
- Memory handling, spatial isolation and timing analyses by abstract interpretation
- Implementation of architectural scheduling and mapping analyses at the level of the operating system

E-Mobility / actuator applications comprise, for instance:

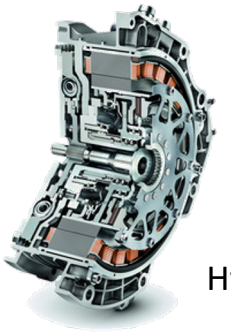
Schaeffler Products



Gearbox actuator



eAxle



Hybrid Module

Active Roll-Stabilizer

E-Wheel Drive

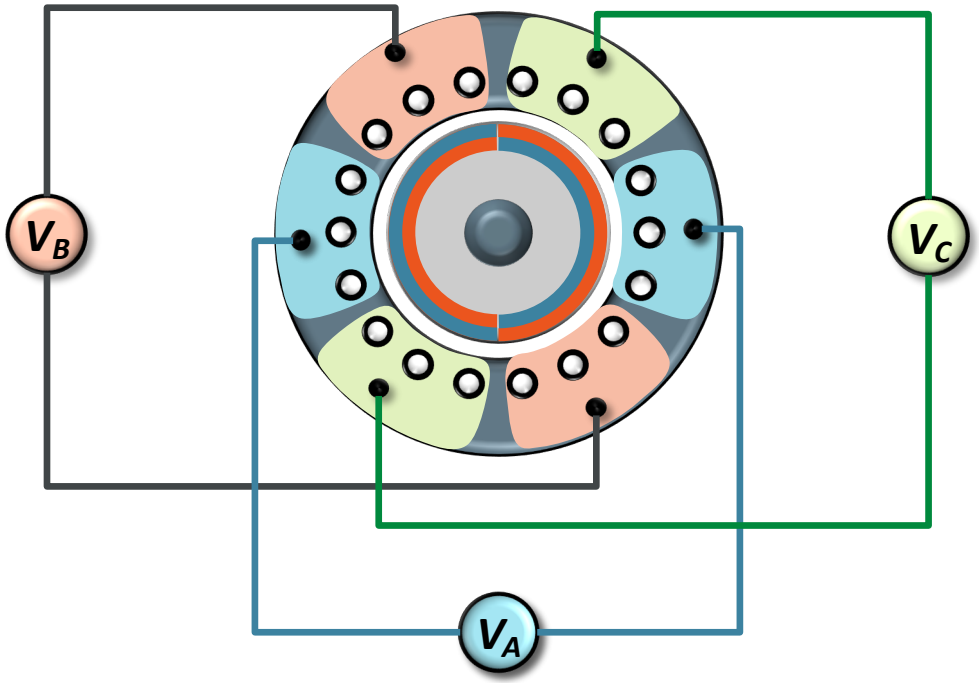
A wide range of these applications need an e-motor control

Why do we need Software for Electric Drives at all?

Basic components of many modern electrical motors:

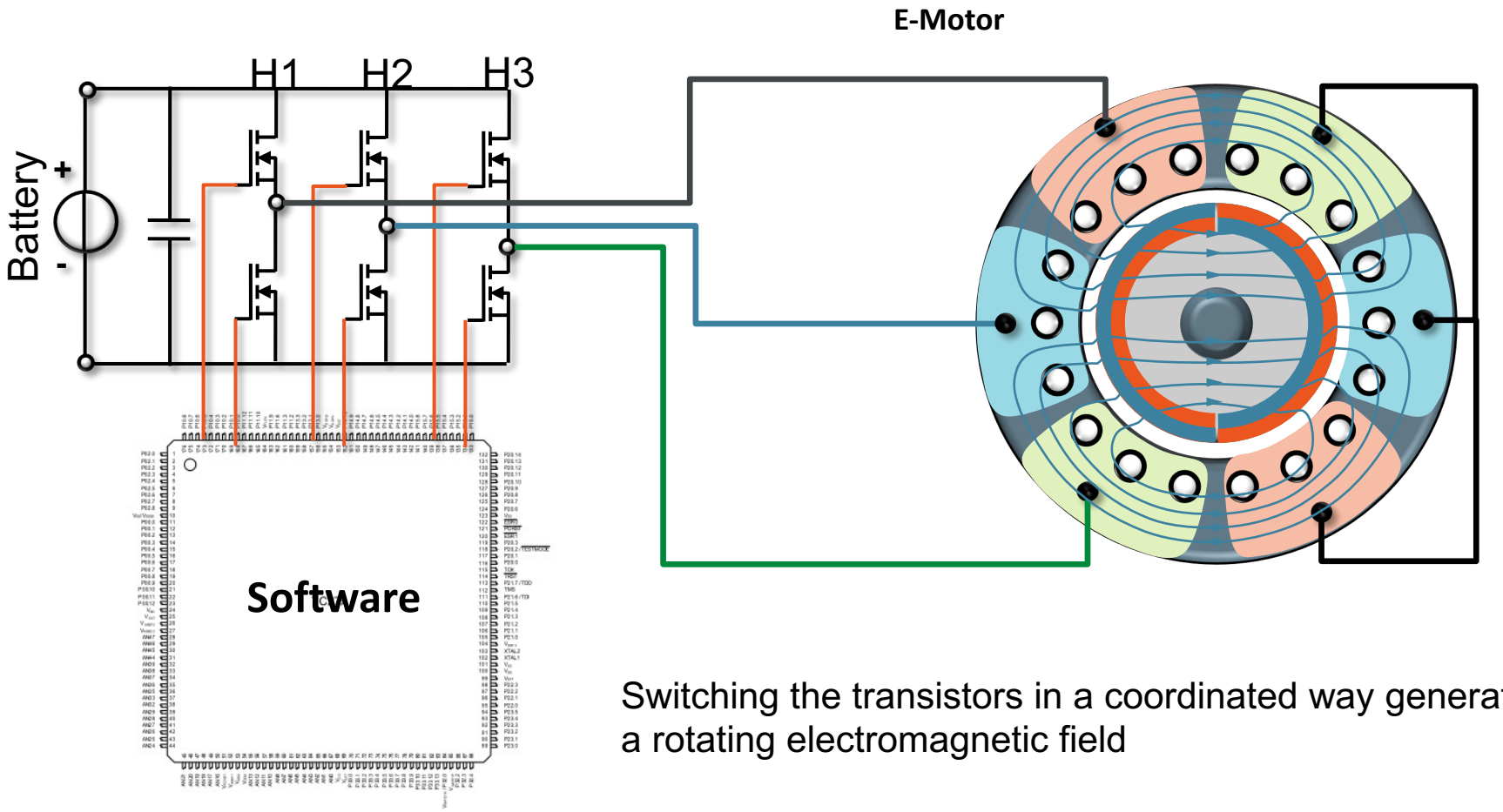
- Rotating armature with or without magnet
- Three phase windings in the stator

Connecting such a motor to a DC battery will not generate continuous rotation of the rotor…



Source: A. Leitner, G. Drenkhahn@Schaeffler R&D Conference 2017

Why do we need Software for Electric Drives at all?



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

Source: A. Leitner, G. Drenkhahn@Schaeffler R&D Conference 2017

Why do we need Software for Electric Drives at all?

High demands on

- energy efficiency
- torque precision
- dynamics
- reliability
- availability

High intelligence and complexity of the control software!

A significant part of our **motor-control know how** gets condensed into **software**

Software Variability in Automotive Mechatronic Projects

How to build software in a way that it can be reused in a mechatronic product line?

Variability points in projects

- System environment (e.g. the concrete vehicle)
- E-Motor type and power class (0,5 -200 kW)
- Microcontroller (derivative) and sensors
- Safety requirements such as spatial and temporal freedom from interference
- Software architecture

Challenge:

Reuse established motor-control functions in a diverse and variable project environment

The E-Motor-Control Software Library

E-Motor Control SW Platform

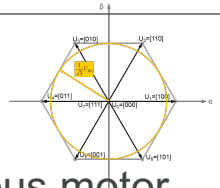
Mechatronic Project #1

Mechatronic Project #2

Library functions are developed using Matlab / Simulink

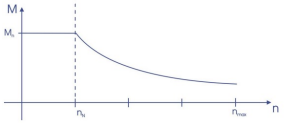
Functional Features

Motor types



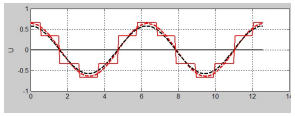
- Permanent magnet synchronous motor
- Asynchronous induction motor

Electric current control



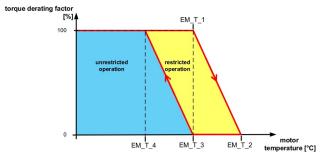
- Field oriented control
- Feed forward, magnetic saturation, reluctance
- Field weakening control
- (Over-)modulation schemes and variable switching frequencies

Superimposed controllers



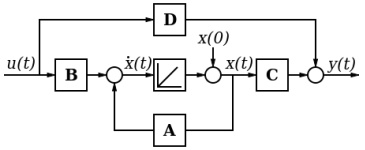
- Speed (window) control
- Jerk control

Derating and Diagnostics



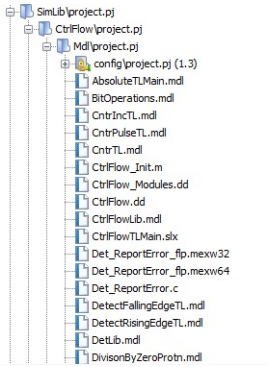
- Self protection and fault detection
- Performance derating

Sensors and Observers



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

Libraries for various utilities



- Table lookup and interpolation
- Numerical routines
- Signal filters

But: Features are not Everything

Project-specific development and the design of software architectures is often driven by functional requirements (features, functionality)

But non-functional requirements (properties) such as memory management, security, multicore-usage are equally important

Platform development suffers from the same problems:

Function-focused development in Matlab/Simulink **does not work** by itself to realize the platform concept

- Hardware-agnostic development is problematic
- **Non-functional properties** such as project-specific **space-time criteria, safety** or **adaptability** are equally important for e-mobility applications!

Selection of a Microcontroller Unit

The decision of a microcontroller unit (MCU) is not only driven by technical reasons

- Cost per unit
- Physical space occupied and weight
- People …

Arbitrary selection of an MCU is **usually a really bad idea!**

MCUs from **different vendors** differ in various ways

- Communication, latencies
- Memory architecture
- Cache coherence
- Pinning
- etc.

Solution: Decision for a processor family from a single vendor supports construction-kit approach. An **analysis** of the available **derivatives** is still **needed**!

Modular Controlboard Hardware Platform

For e-mobility applications, we develop a particular electronic control unit (ECU) we termed **Controlboard:**

- Generic control board for electric drive applications
- "Construction kit" to quickly derive project-specific variants

Characteristics:

- A great choice of different interfaces available:
  - CAN/Flexray
  - Resolver/Induction/Hall sensors
  - Temperature sensors
- PWM/SPI/ADC type interfaces
- Independent drive of two motors possible (traction, actor)
- Infineon AURIX multicore processor family (currently TC277 derivative)
- Safety functions ASIL-D possible by redundancies and external watchdog

Functionality, Safety, Computational Space-Time

Functionality, safety, computational space-time: Alignment of design goals

- Functionality often benefits from methods applied in the context of safety-relevant systems
- 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

Safety focus our project: Provide real-time capability in a multicore environment and provide spatial isolation and good code/data placement

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.

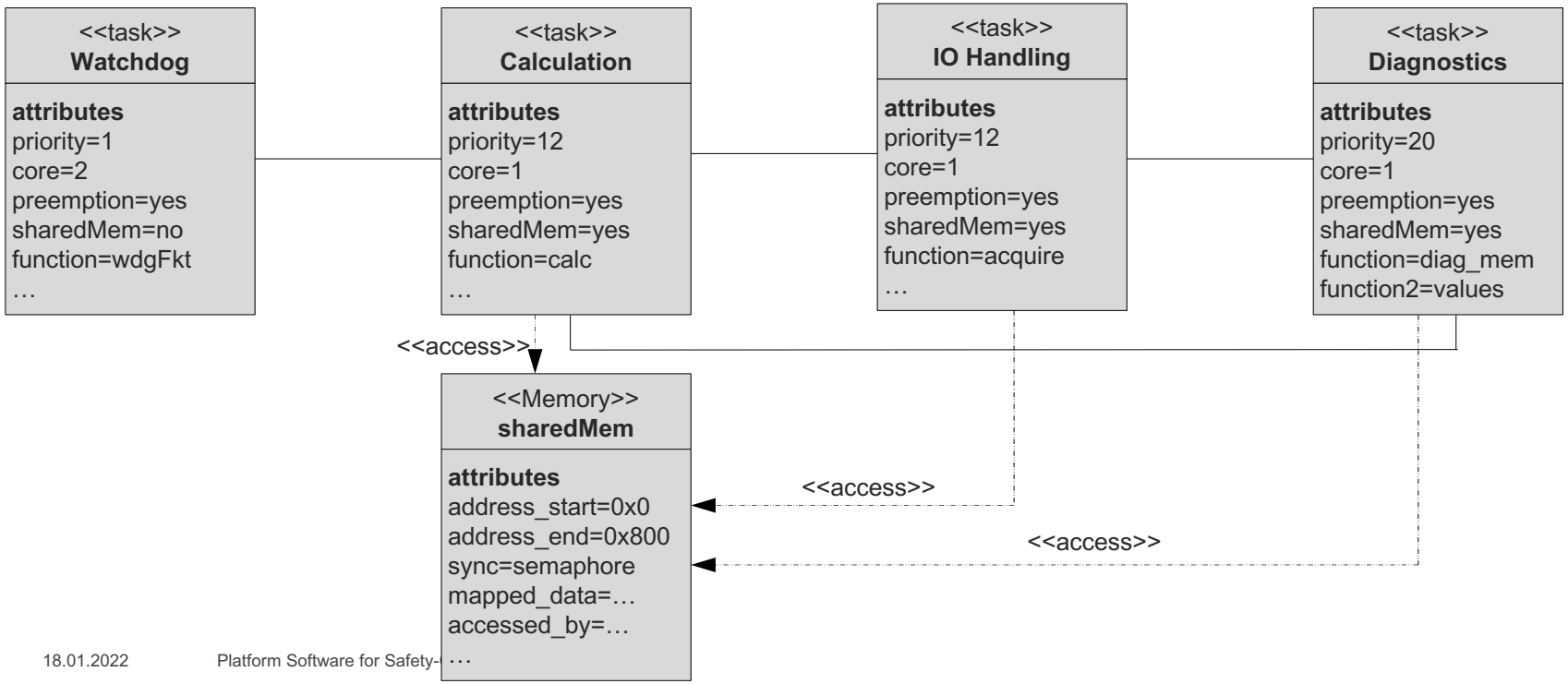
Functionalities may also be deployed on the same MCU: **Integrated architecture**

- 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

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:

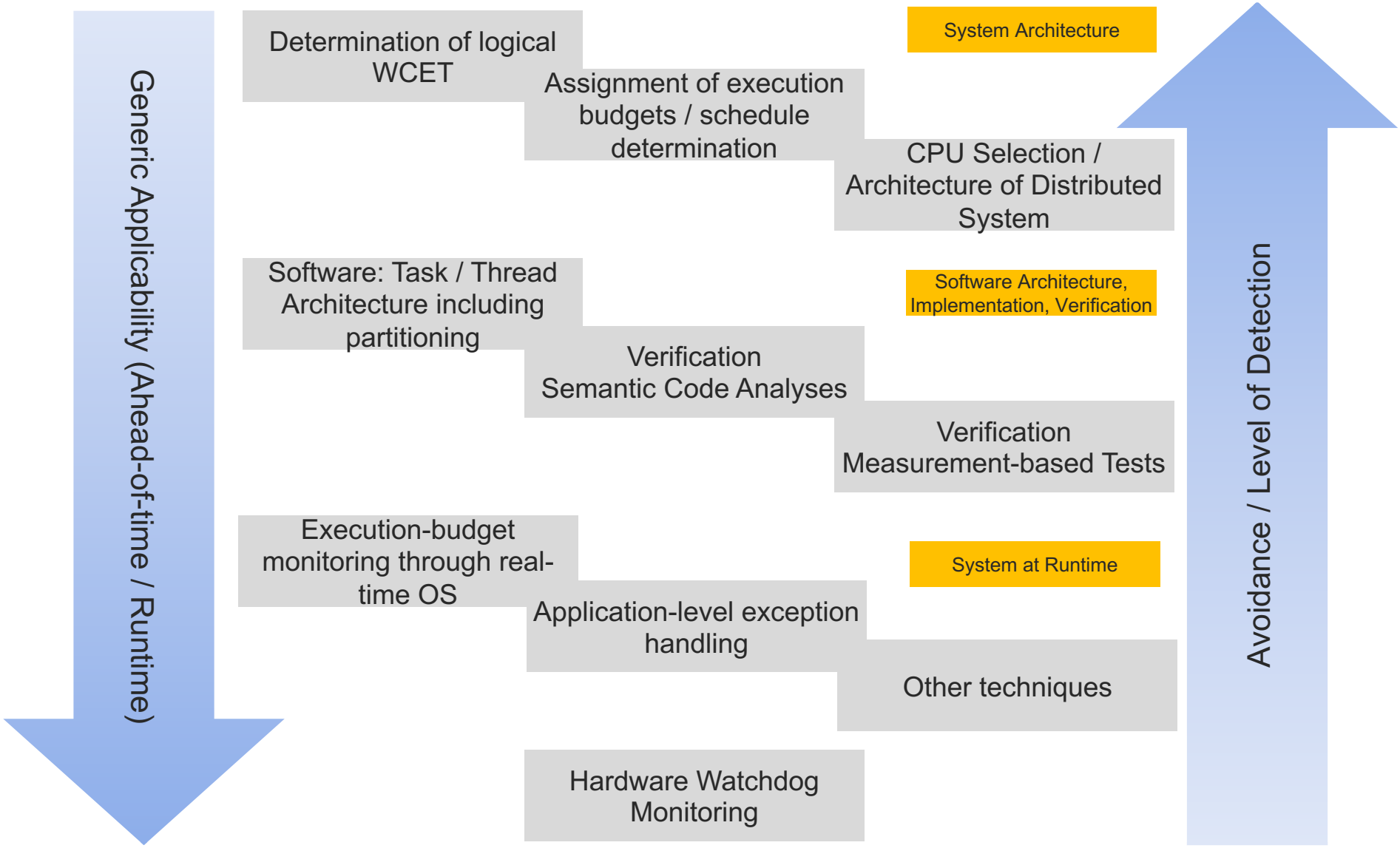
- The temporal /spatial **partitioning** is dependent on the **system requirements / architecture**
- CPU selection
- Distributed network of MCUs, etc.
- Aspects at all system-architectural levels influence each other

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 Safe / Efficient Memory Management

- Memory protection / management can be implemented at different granularities
- The techniques can be combined to achieve a suitable level of protection

Examples for techniques:

At the program level and architectural-design level (ahead-of-time)

Hardware-based mechanisms

Mechanisms for Providing Safe / Efficient Memory Management

At the program and architectural-design level (ahead-of-time)

- Use of a programming-language subset in order to deal with systematic faults at the software level (bugs)
  - Example: Restrict the use of untyped memory (cf. semantic code analyses (more than MISRA-C!)), use of memory-safe/type-safe language
- Syntactic and semantic code analyses identifying false pointer usage, false use of operations and bogus data flows
- Etc

Use language properties, deployment decisions and/or semantic-analysis results or to provide data and code mapping

Example for deployment decisions: assign data / code to threads and partitions

- Define protection boundaries for an application
- Control-flow (i.e., thread) and partition isolation with respect to memory

Note: The thread architecture is part of the software-architecture design. The thread architecture includes information on the set of threads, their interplay and characteristics (priority, stack size, dependencies etc.). Verification of task-local (thread-local) memory usage such as stack usage or memory-region usage supports spatial isolation.

Mechanisms for Providing Safe / Efficient Memory Management

Hardware-based mechanisms

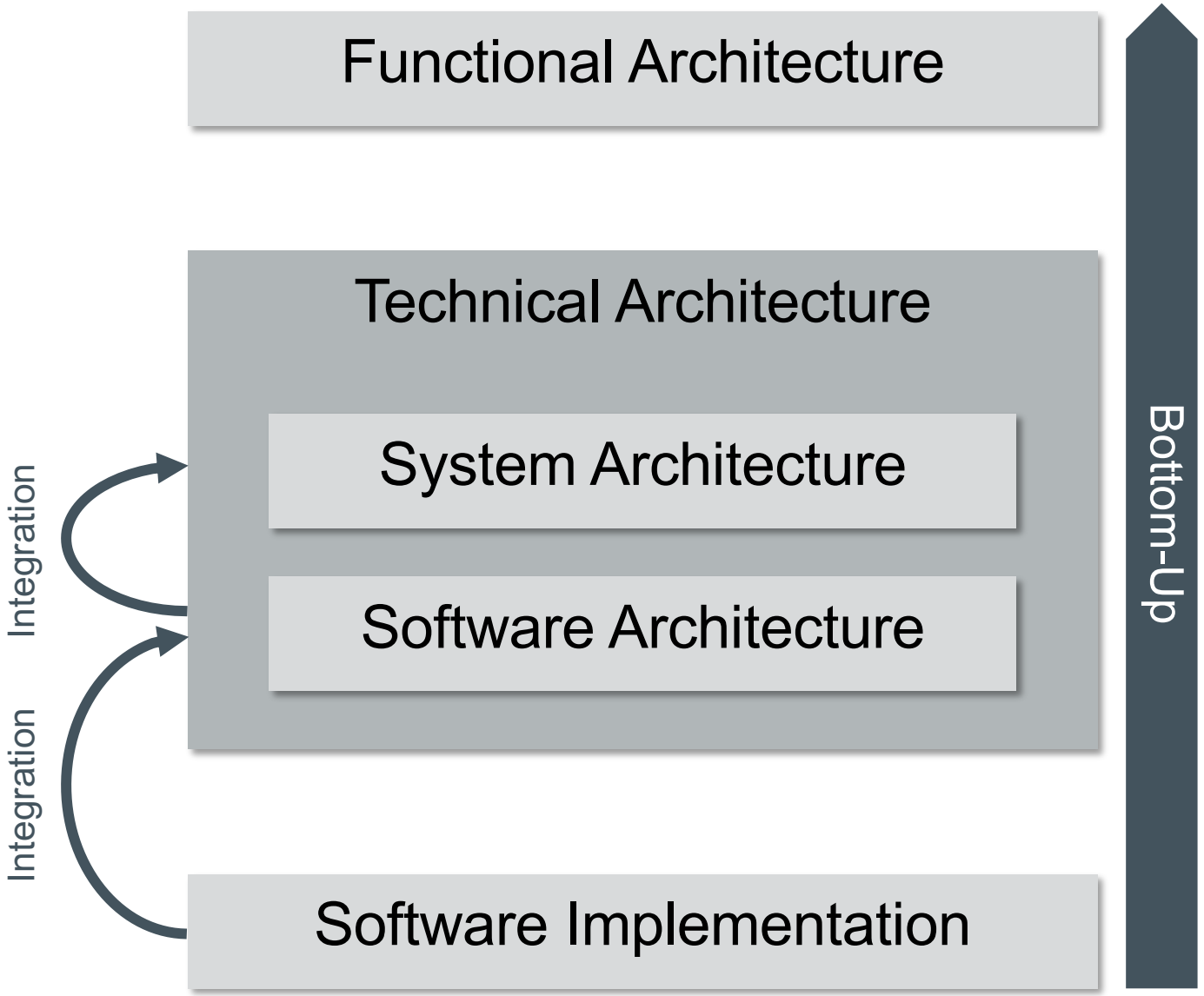
- Error-correction codes to deal with (random) permanent/transient faults in memories: Ensure that pointers and types are not affected by memory faults.

Example: ECC protection of memories may be provided by the MCU or it can alternatively be applied at the software level.

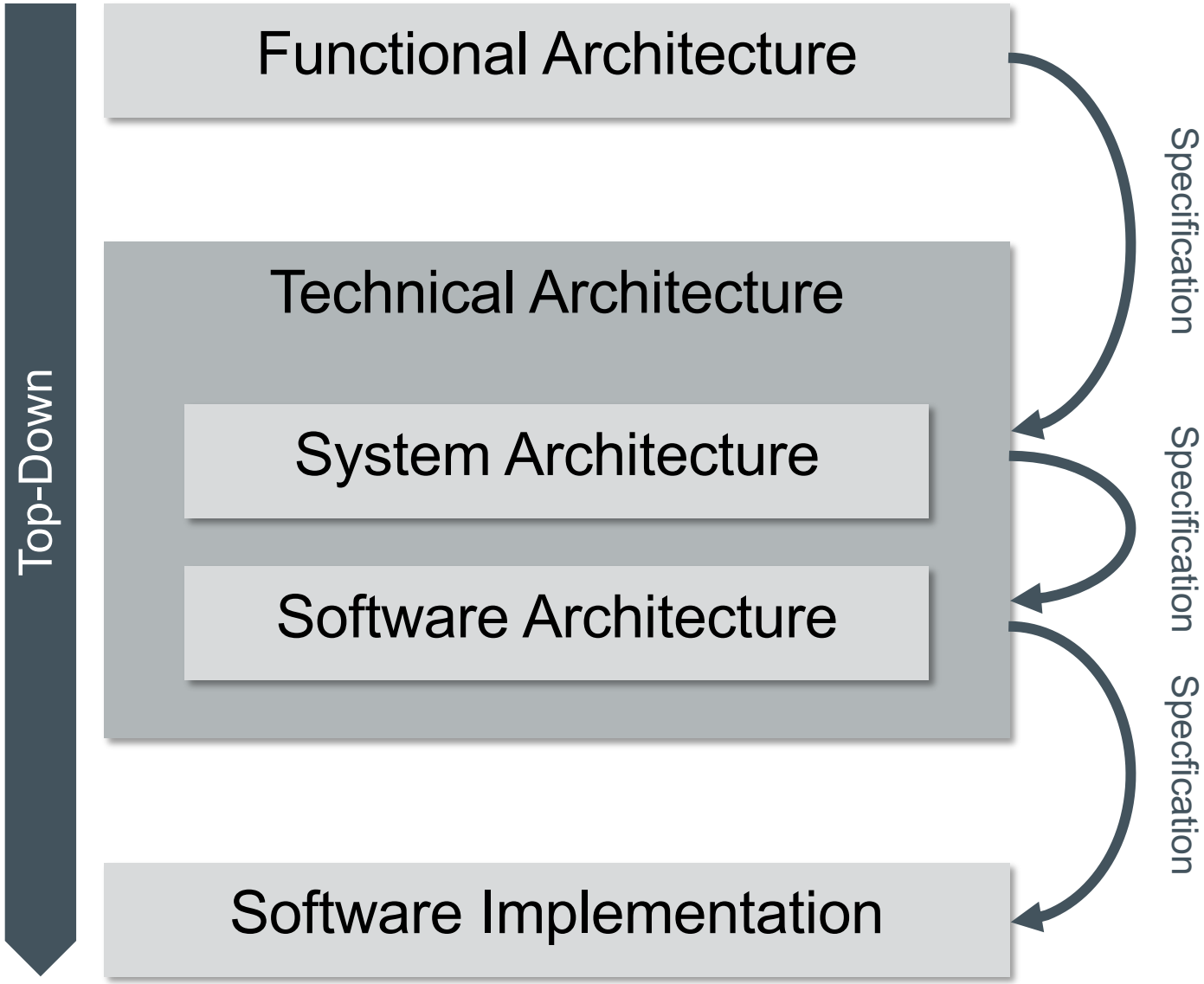
- Hardware-based memory protection, for instance, a region-based memory-protection unit

Note: MPU-based partitions are broadly used and belong to the sandboxing techniques, that is, an MPU cannot detect semantic faults but it restricts error spreading to the software partition that exposes a memory-safety defect (e.g., false pointers, stack overflow, bit flip in memory occupied by data relevant to memory safety).

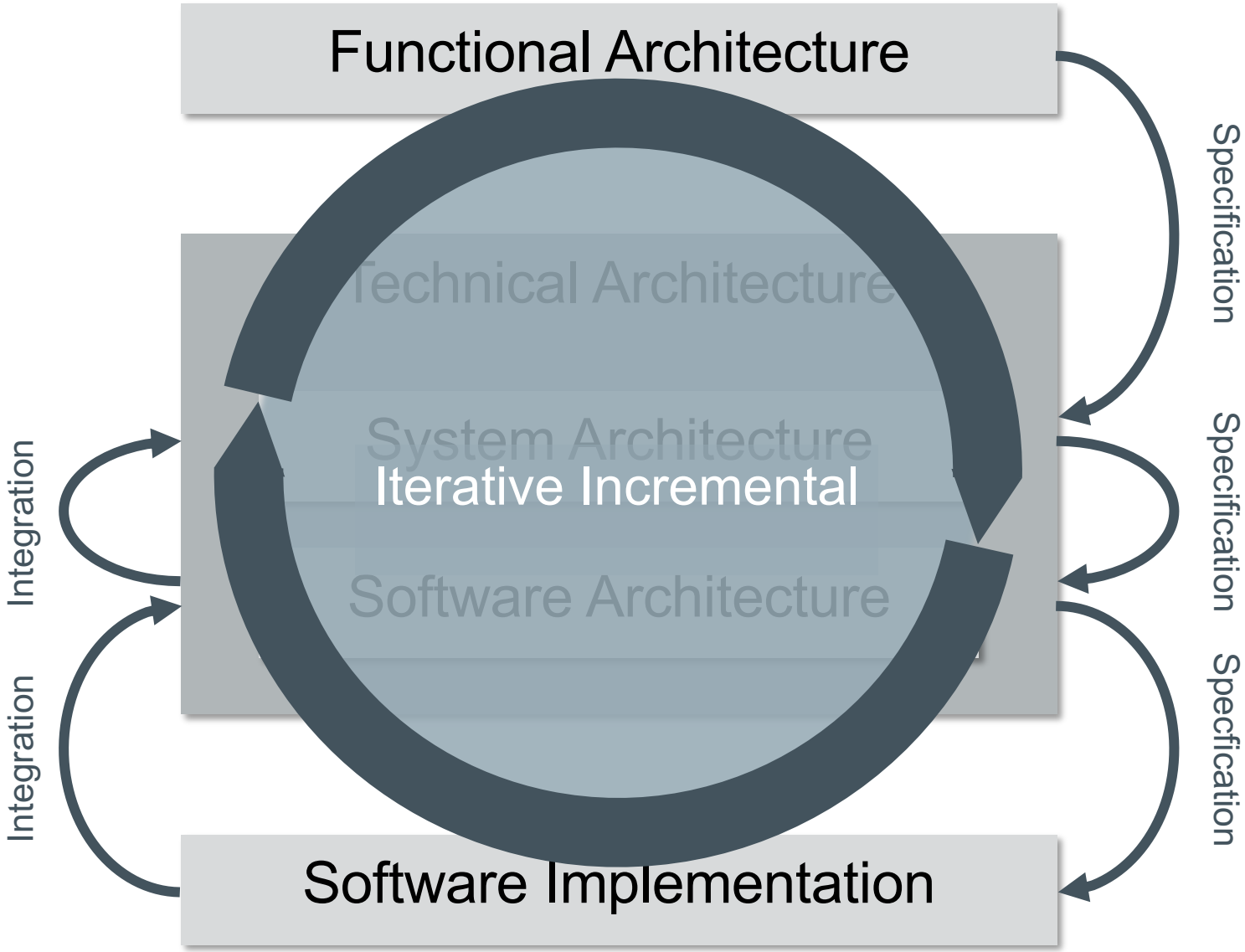
How to Construct A Safe Real-Time System?



How to Construct A Safe Real-Time System?



How to Construct A Safe Real-Time 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**

Building Blocks of the Engineering Framework

Functional / technical architecture analyses

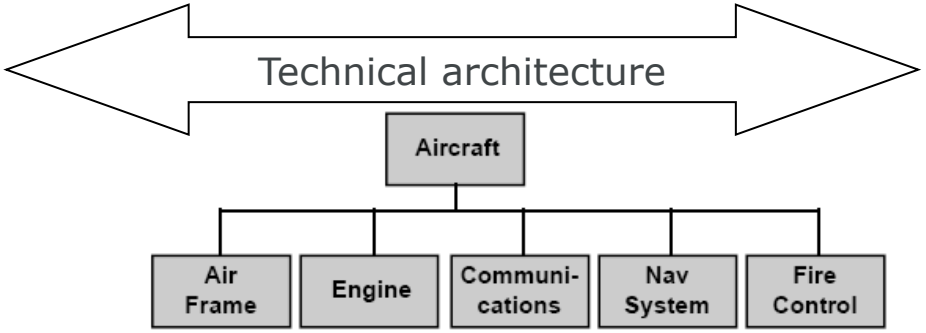
- Functional and data dependencies
- Logical spatial isolation realms
- Temporal isolation and logical execution time
- Task deployment and hardware constraints

Source- and binary-code analyses

- Data- and code accesses at the source-code level: scopes and isolation realms
- Timing: Reconstruction of control-flow graph from binary and measurement of basic-block execution on target hardware

System Architecture

Example: Functional / Logical vs Technical Architectures



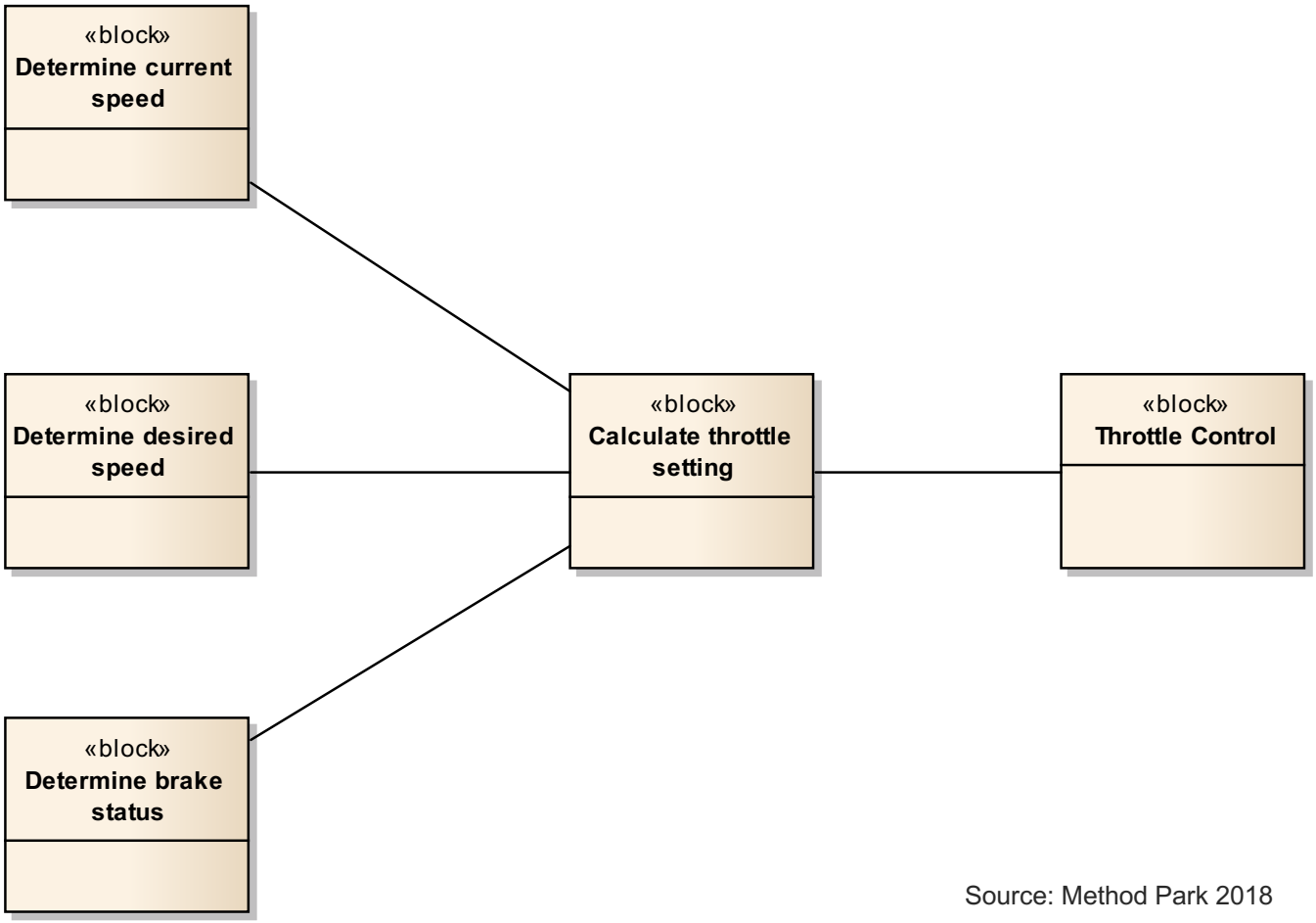
Functional architecture

Function Performed					
Preflight check	X	X	X	X	X
Fly					
Load	X				
Taxi	X	X	X		
Take-off	X	X			
Cruise	X	X	X	X	
Recon	X	X	X	X	
Communicate			X		
–					
–					
Surveillance					
–					
–					

Source: Method Park 2018

System Architecture

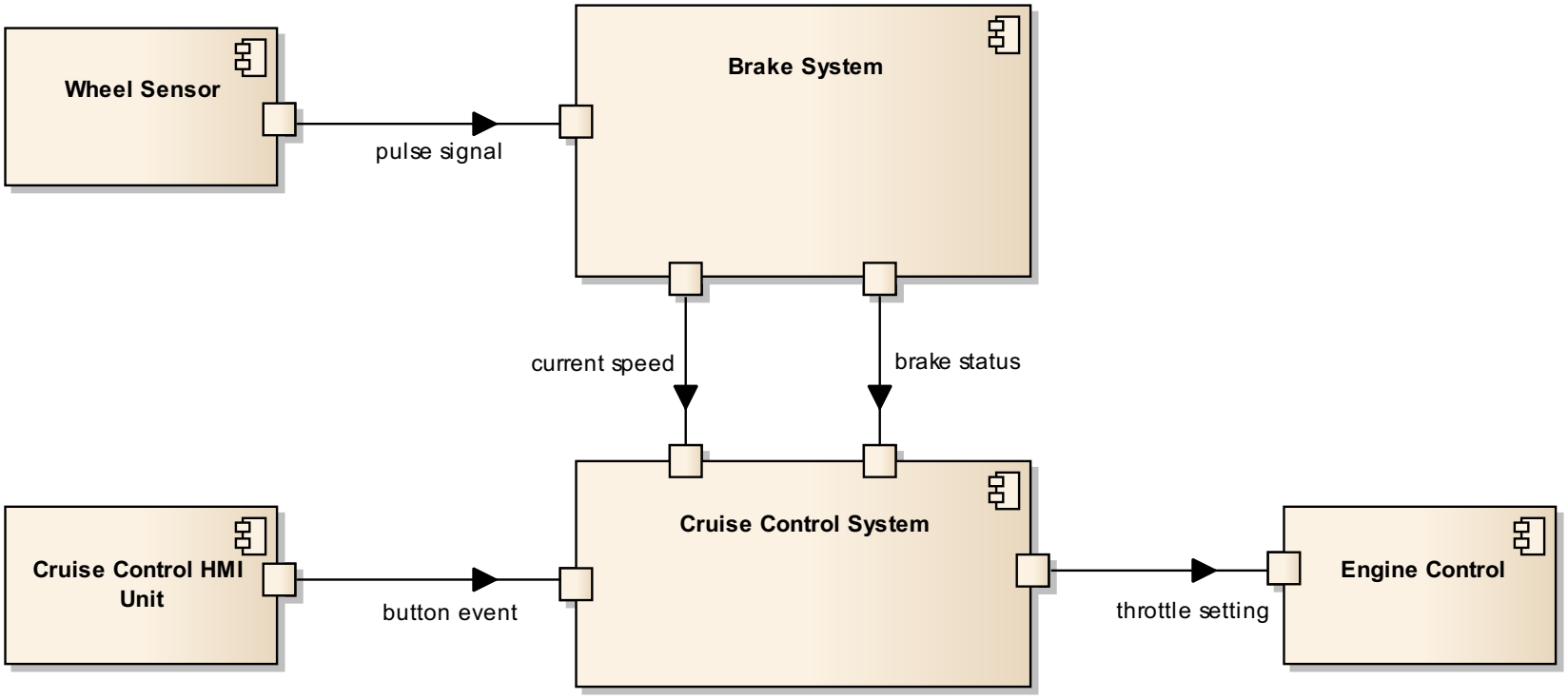
Cruise Control – Functional architecture



Source: Method Park 2018

System Architecture

Cruise Control – Technical architecture



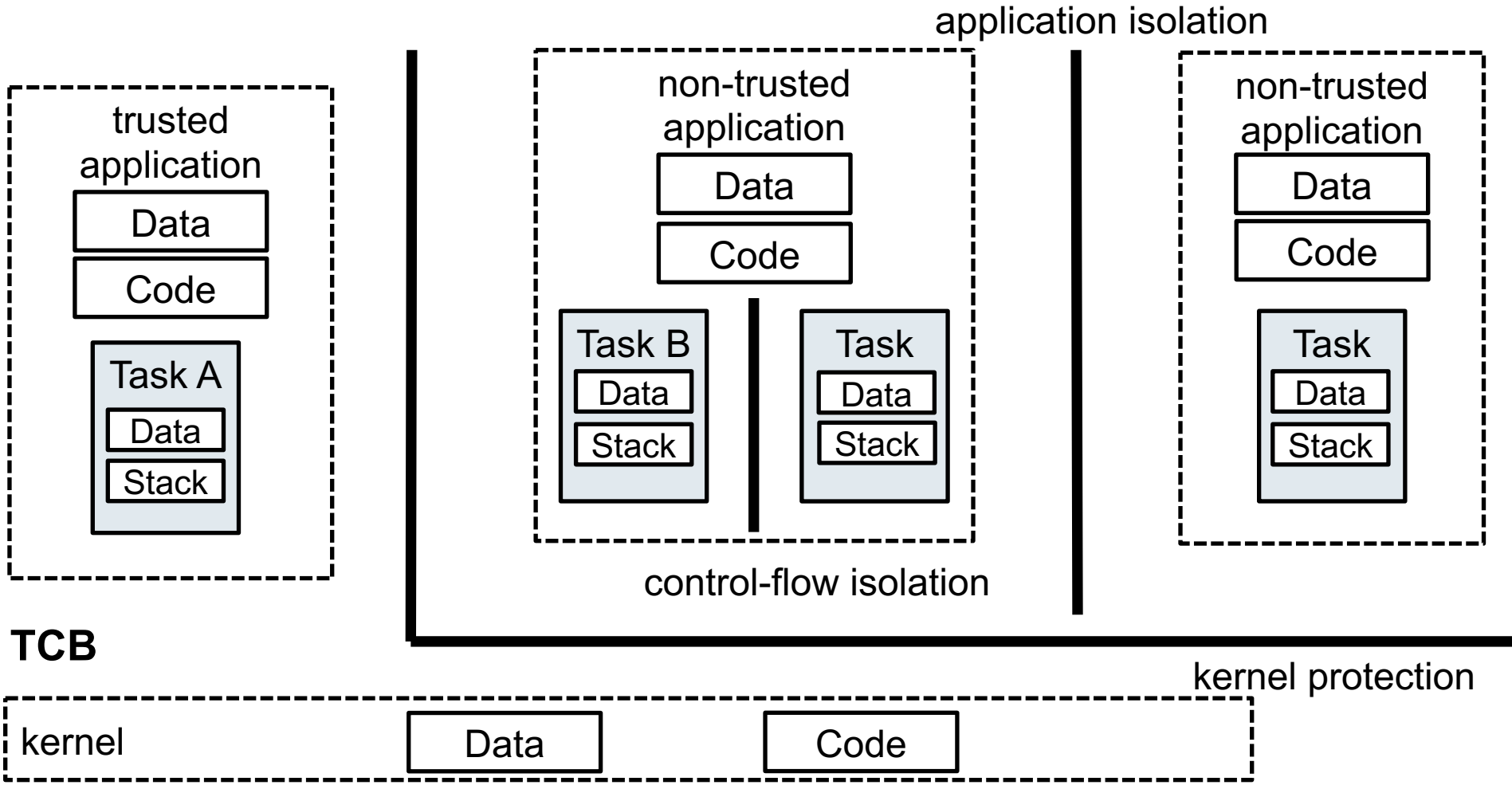
Source: Method Park 2018

Technical Architecture

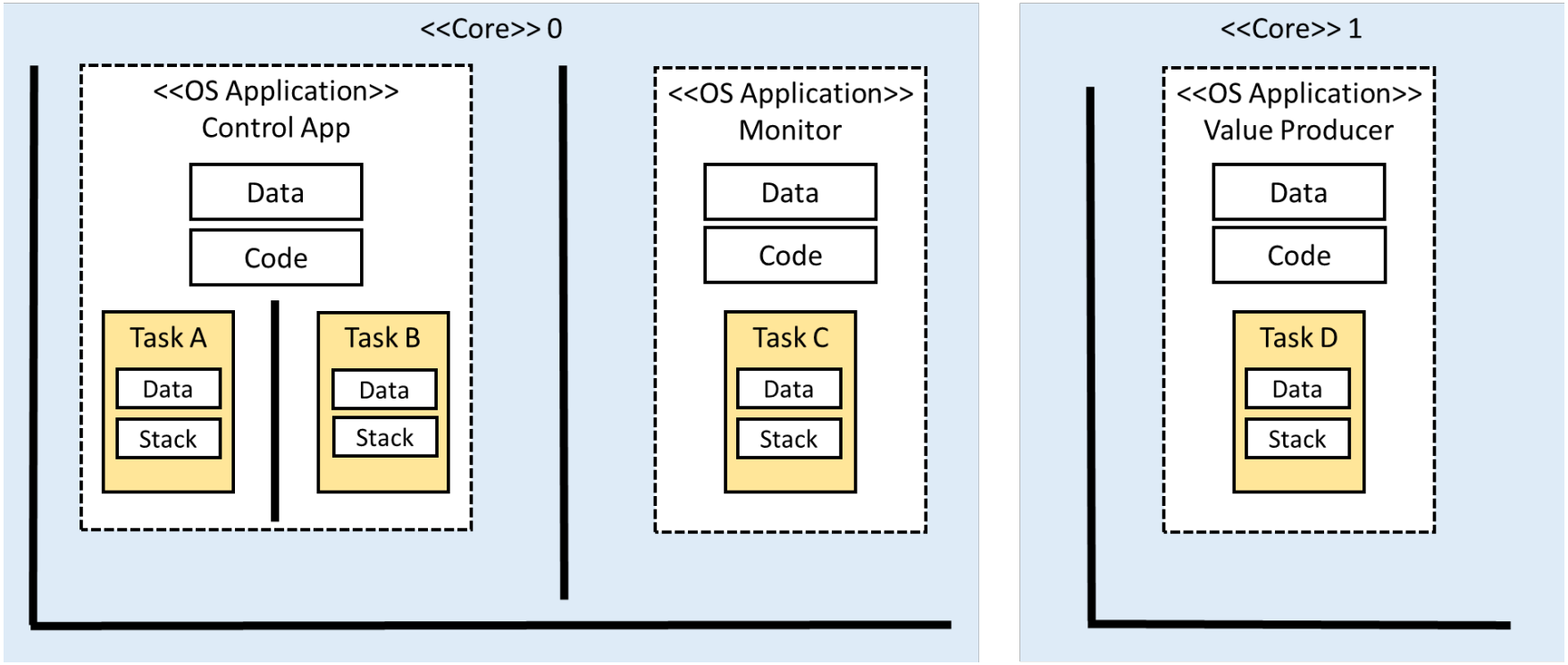
Definition and deployment of architectural building blocks is part of the technical architecture

- Selection of microcontroller
- Selection of an operating system and other infrastructure services
- Allocation of architectural building blocks to physical elements
  - Tasks to processors
  - Components to isolation realms
- Etc.

Memory-protection Model of AUTOSAR OS



Technical Architecture: Deployment View for Spatial Isolation



ASSIST: Scheduling at the Level of Functional Architectures

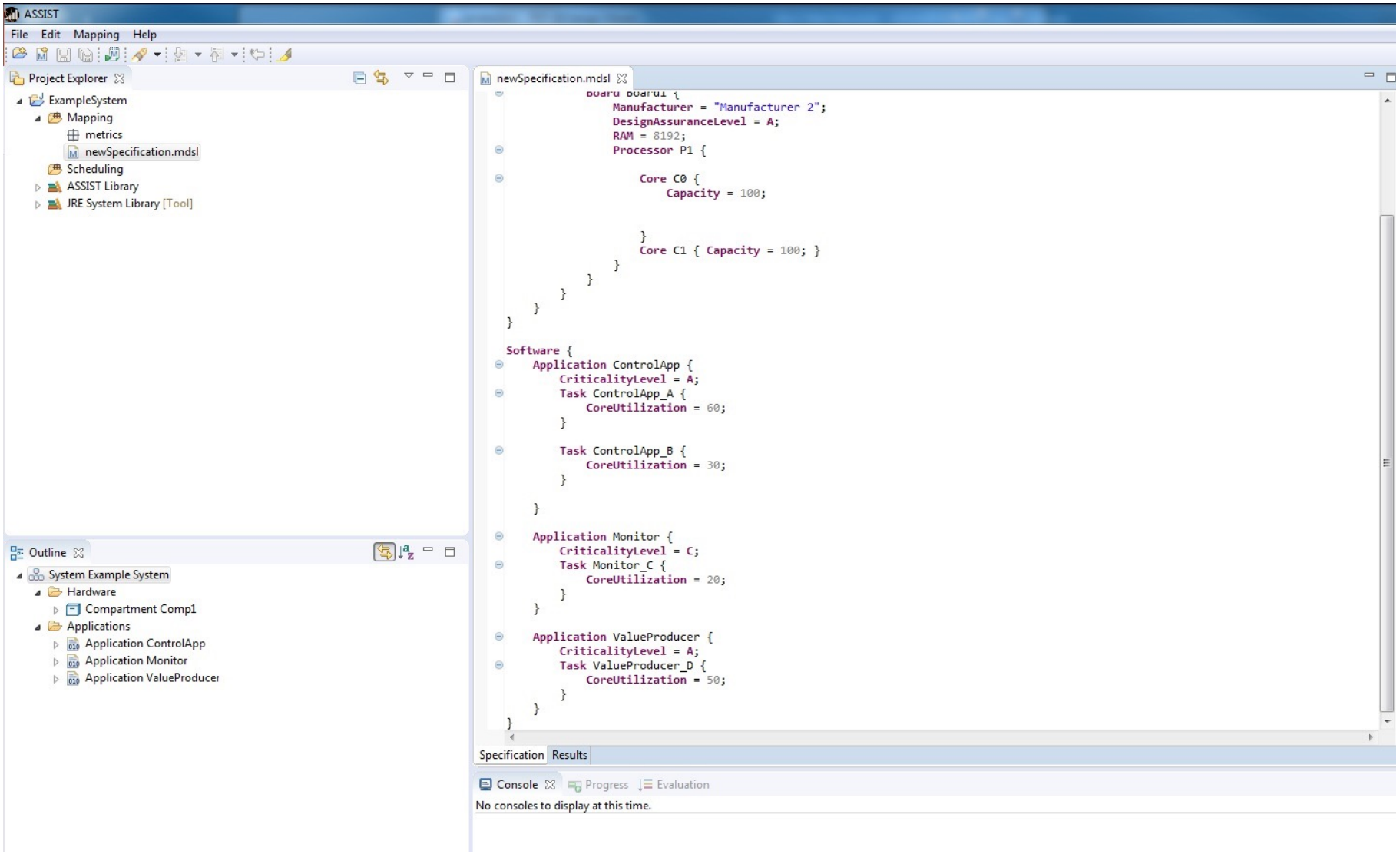
How to derive a technical architecture from a functional architecture?

Define constraints for the technical architecture

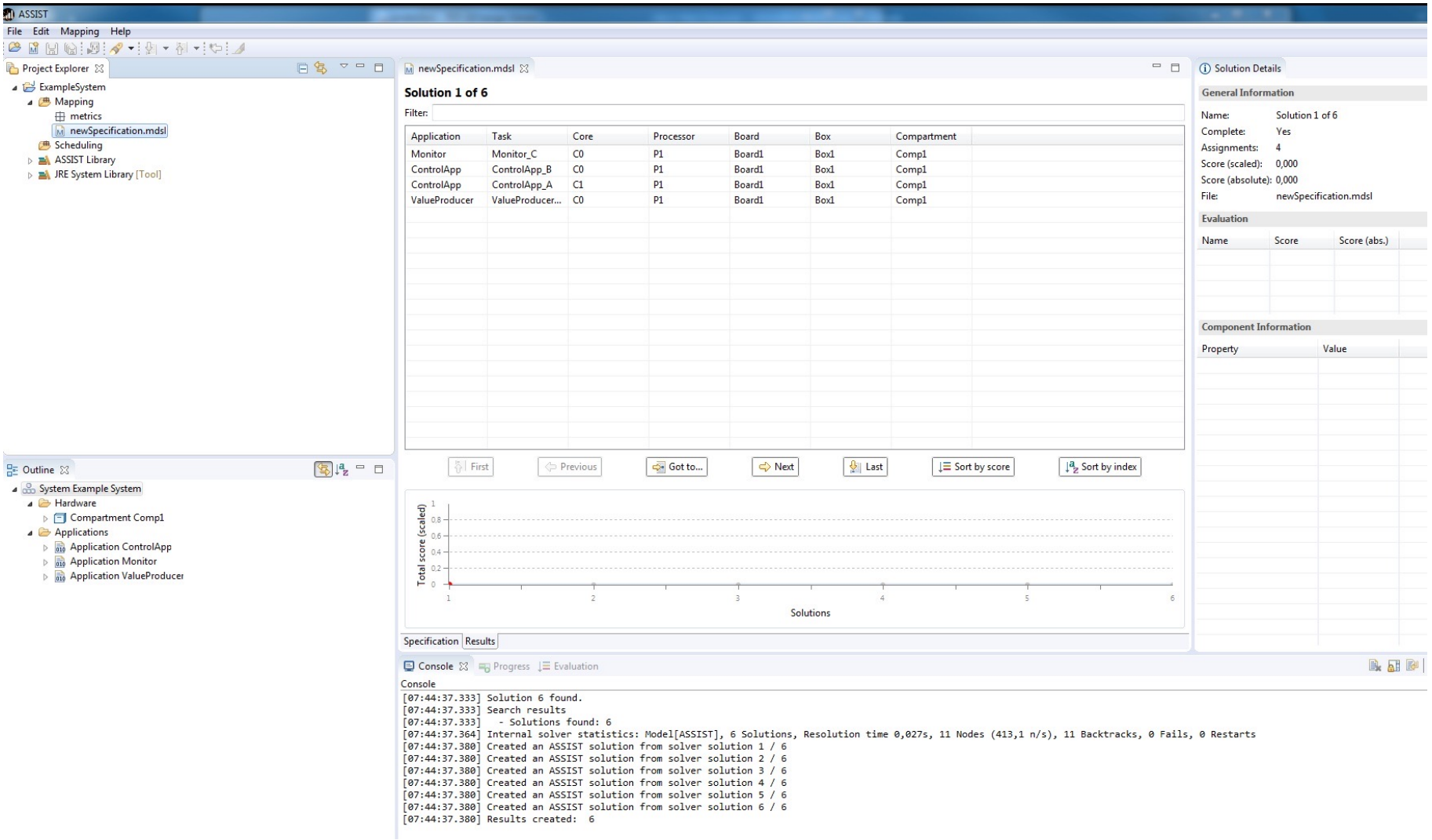
- Hardware resources
- Temporal isolation and other timing properties
- Spatial isolation
- etc.

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

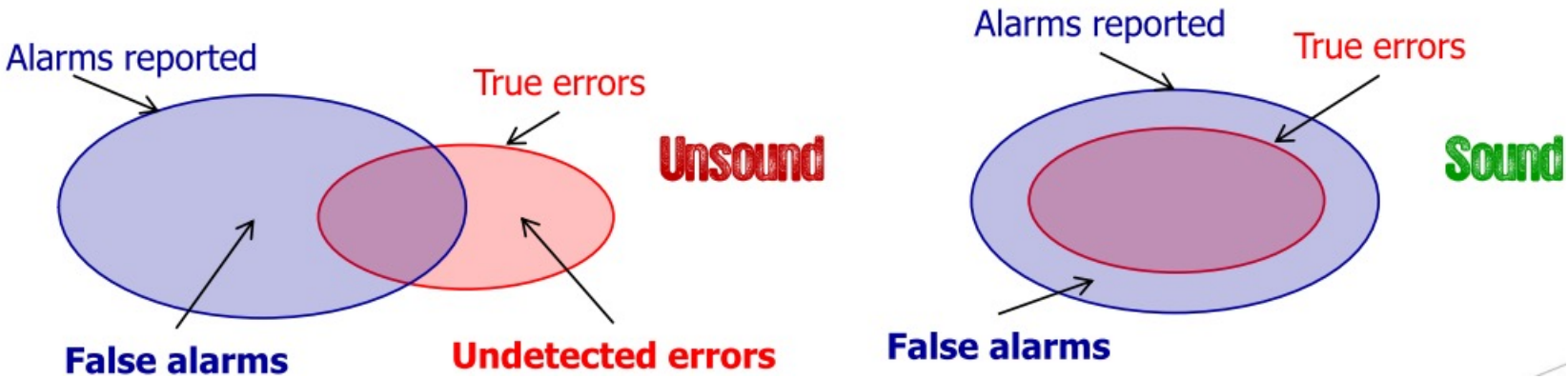
Objective: Detection of runtime errors in programs (dynamic tests are often unsuitable)

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

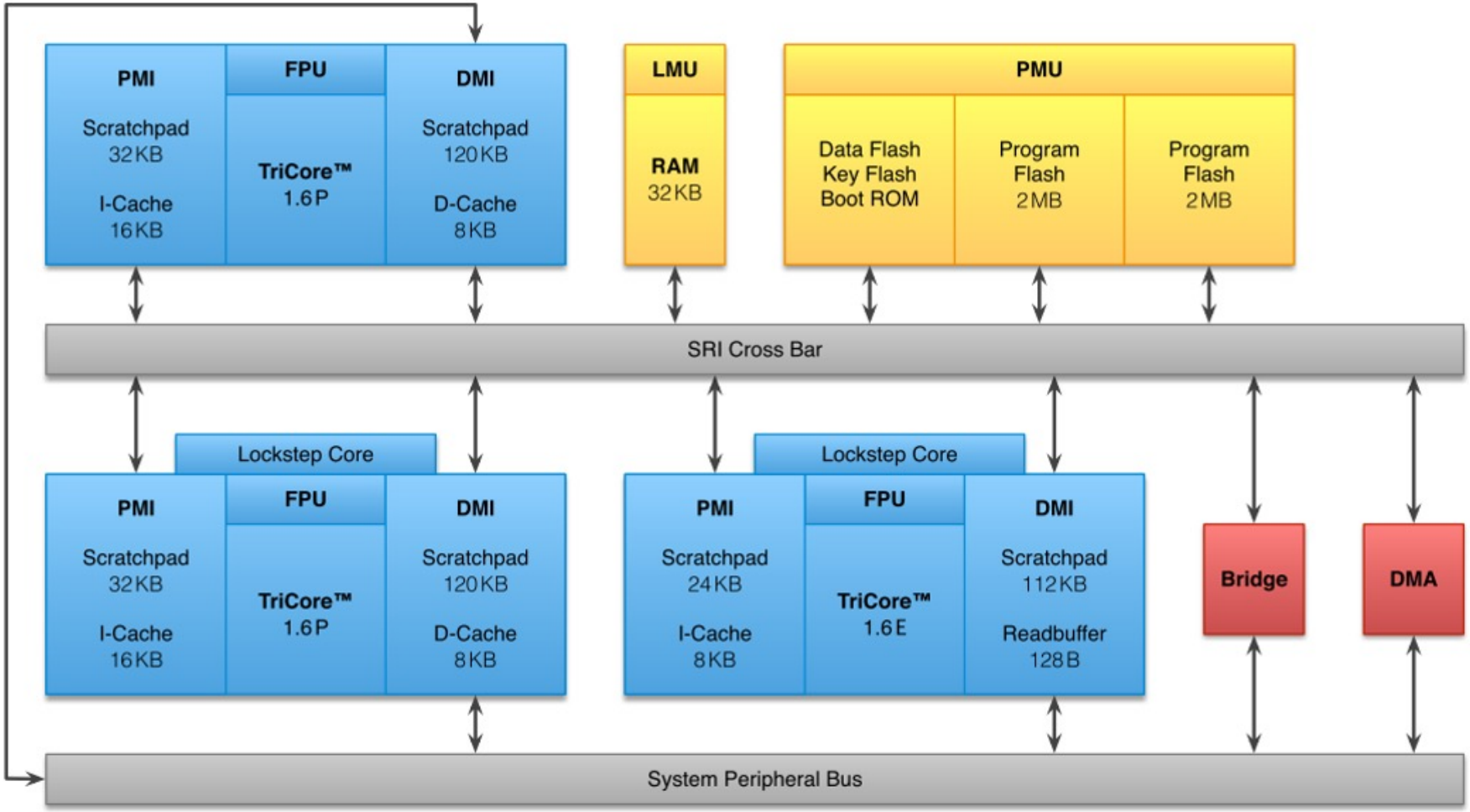
Real-Time Specification for Java (RTSJ)

Memory management by means of **logical** memory types

- Local variables: Regional memory with smallest „scope“ (stack)
- ScopedMemory: Memory in scope of user-defined threads
- HeapMemory: Globally accessible, memory-managements strategy up tp the implementation (e.g. garbage collector)
- ImmortalMemory: cf C‘s global variables, statically allocated

	Reference to Heap	Reference to Immortal	Reference to Scoped
Heap	Yes	Yes	No
Immortal	Yes	Yes	No
Scoped	Yes	Yes	Yes, if same, outer, or shared scope
Local Variable	Yes	Yes	Yes, if same, outer, or shared scope

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

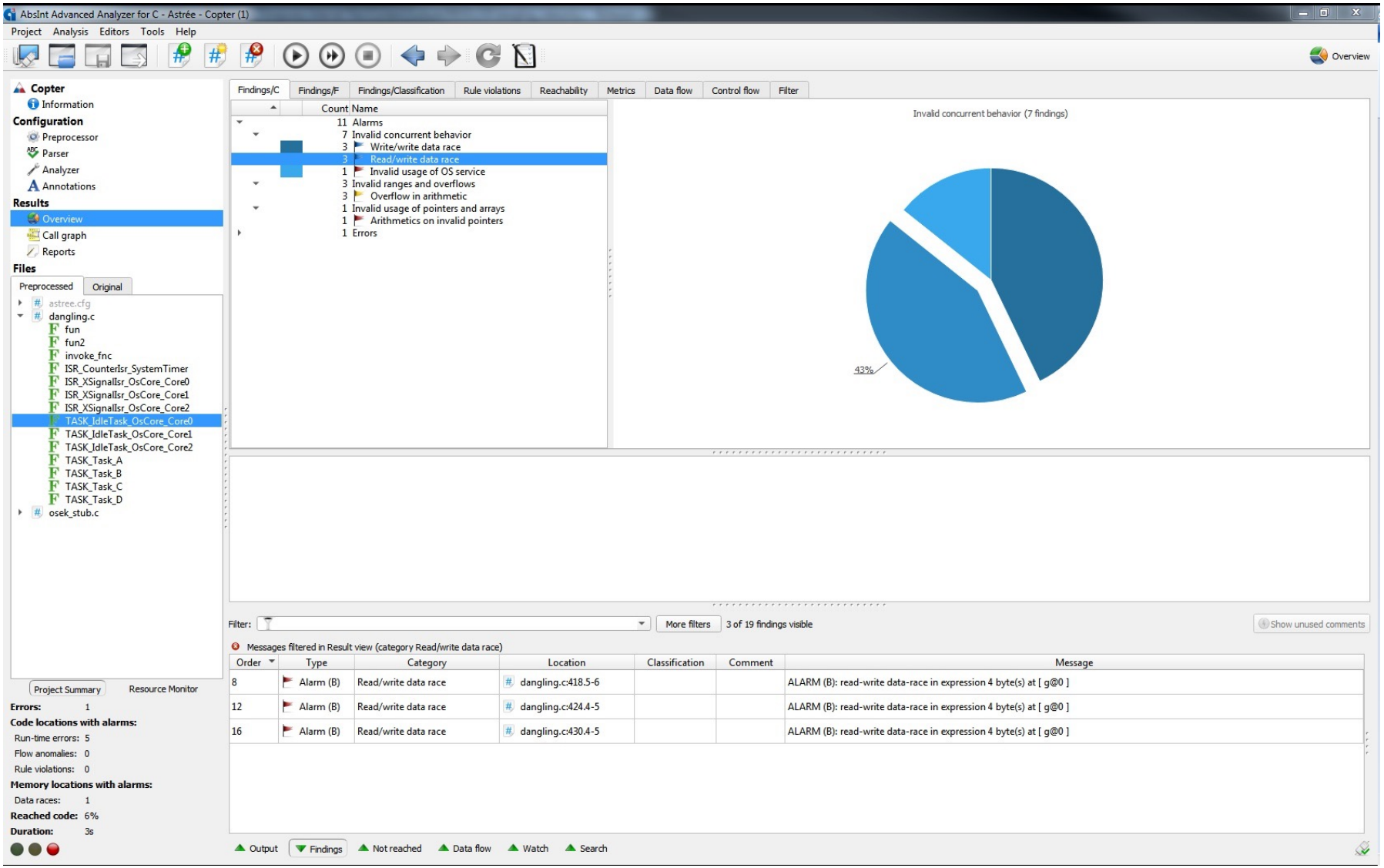
- Respect of program semantics: cf. RTSJ‘s memory model; stacks, data segments
- Extension for constant data
- Respect of the microcontroller‘s physical memory architecture

Data classes

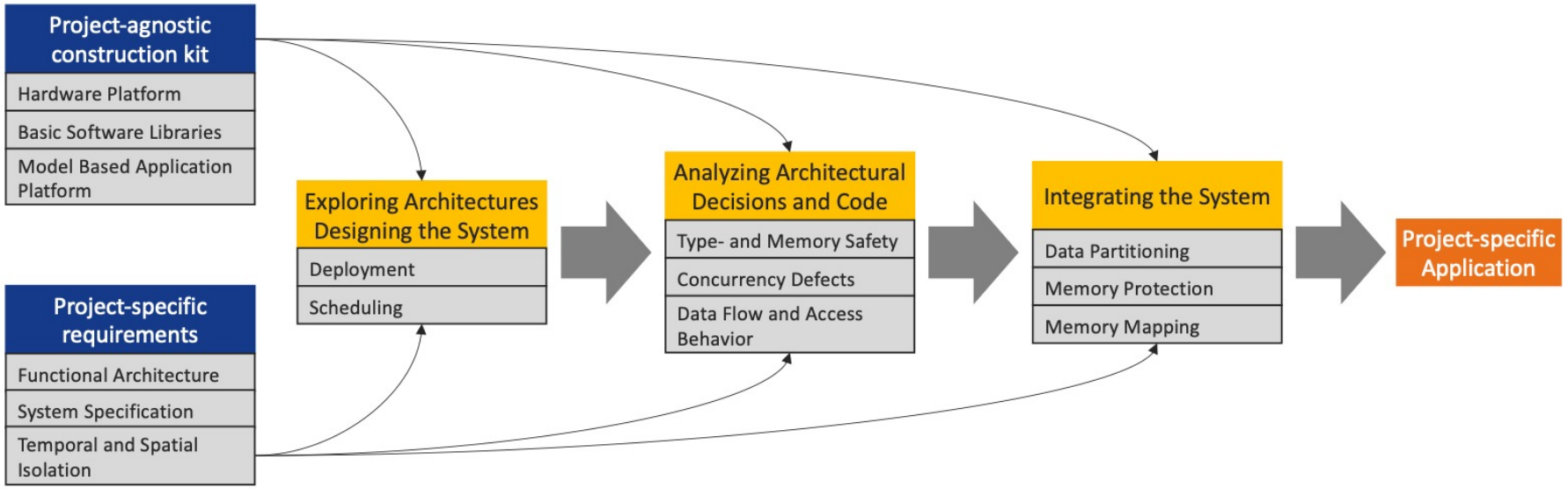
- Thread-local data, allocation in core-local memory, local access (e.g. stack assignment)
- Thread-local data, allocation in core-local memory, cross-core access
- Thread-global data, allocation in core-local memory
- Thread-global, core-global data
- Constant data
  - True constants (e.g. placement 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 SW 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