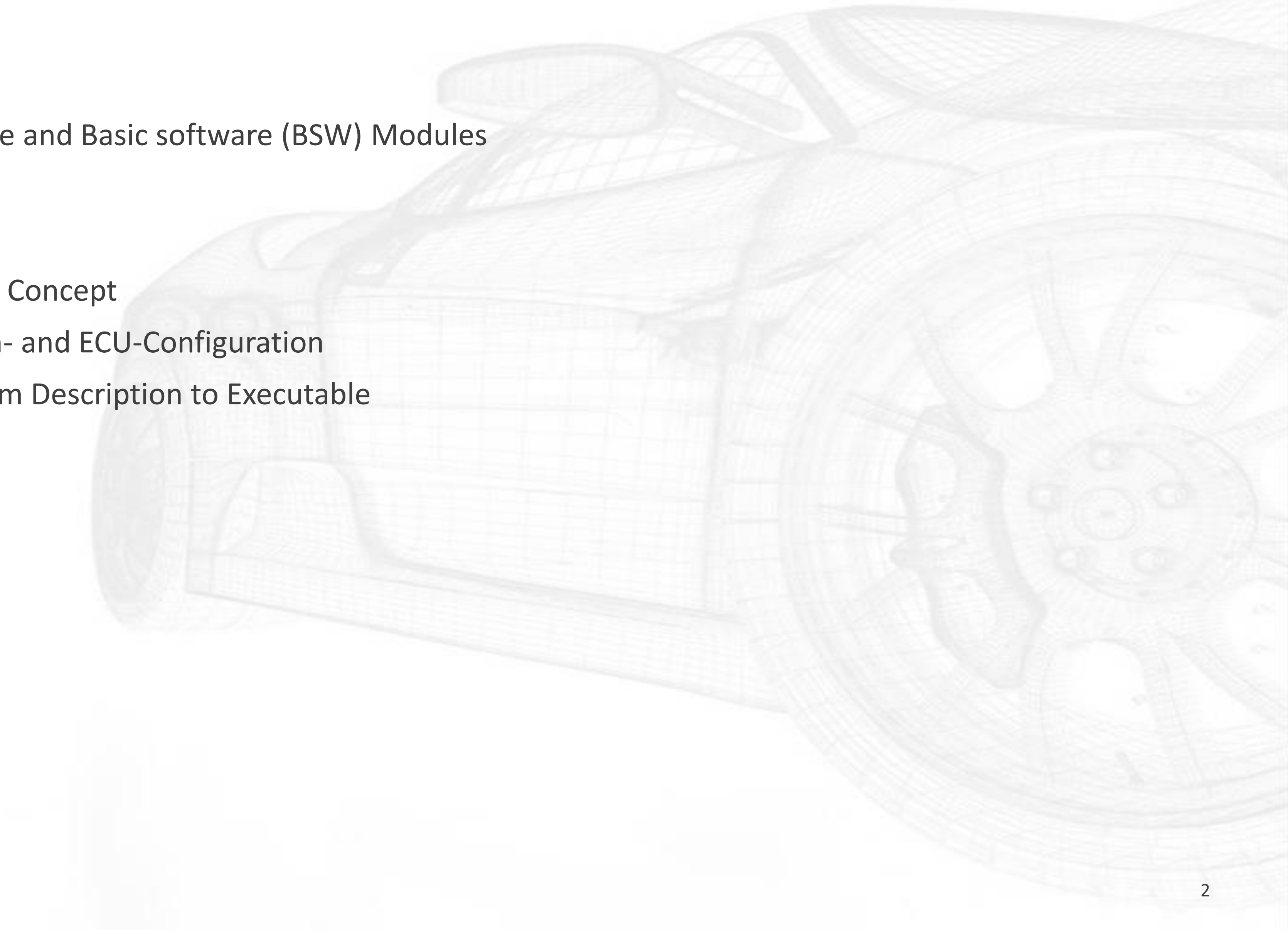
EB tresos Classic AUTOSAR Training

Elektrobit

Architecture & Methodology

Chapter overview



Architecture:

- Introduce the Layered Software Architecture and Basic software (BSW) Modules

Methodology

- Introduction to the AUTOSAR Configuration Concept
- Understand the Difference between System- and ECU-Configuration
- Understand the Complete Chain from System Description to Executable

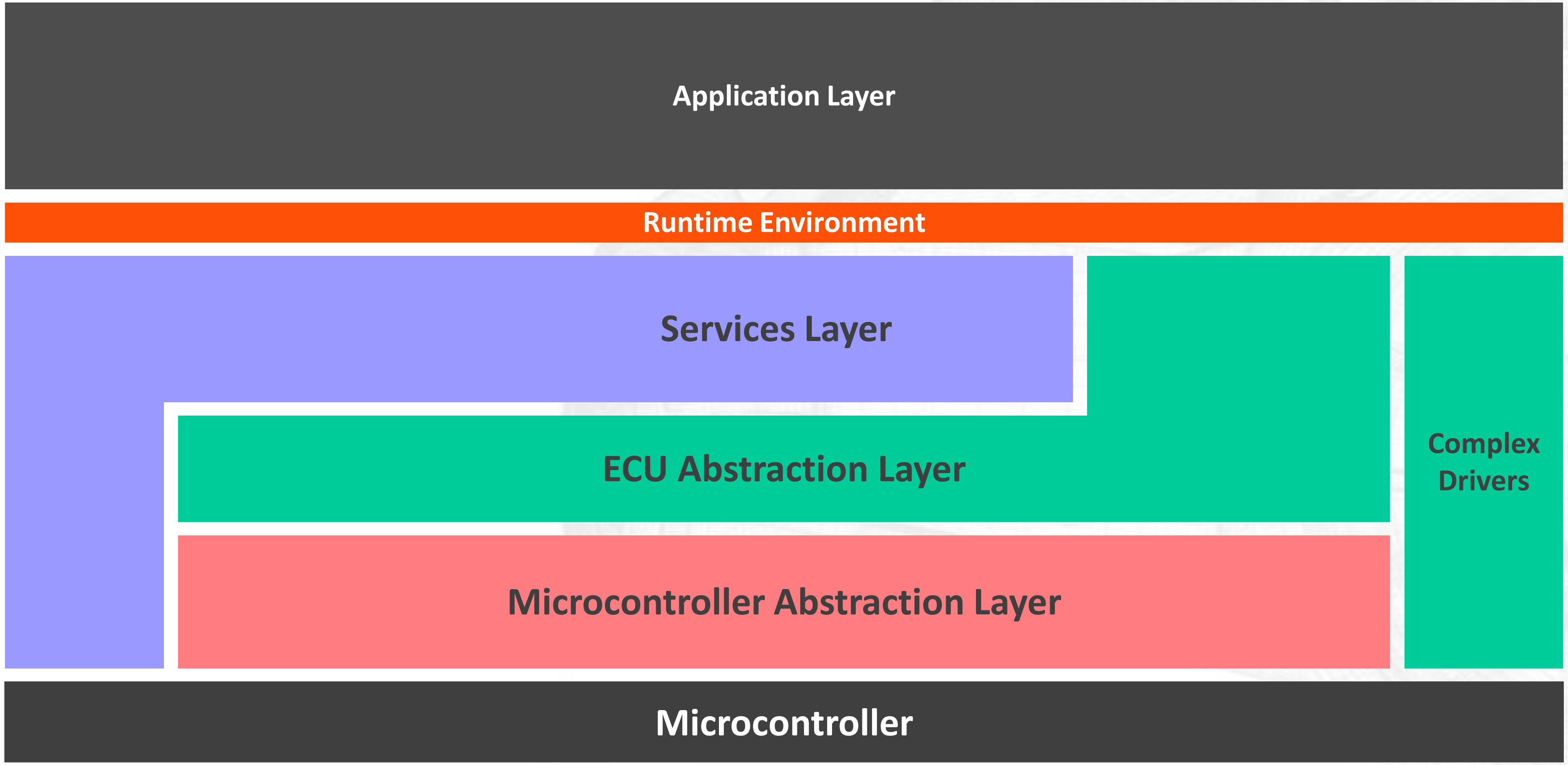
Migration & Integration Strategies:

- Implementation Classes
- Configuration Variants & Classes

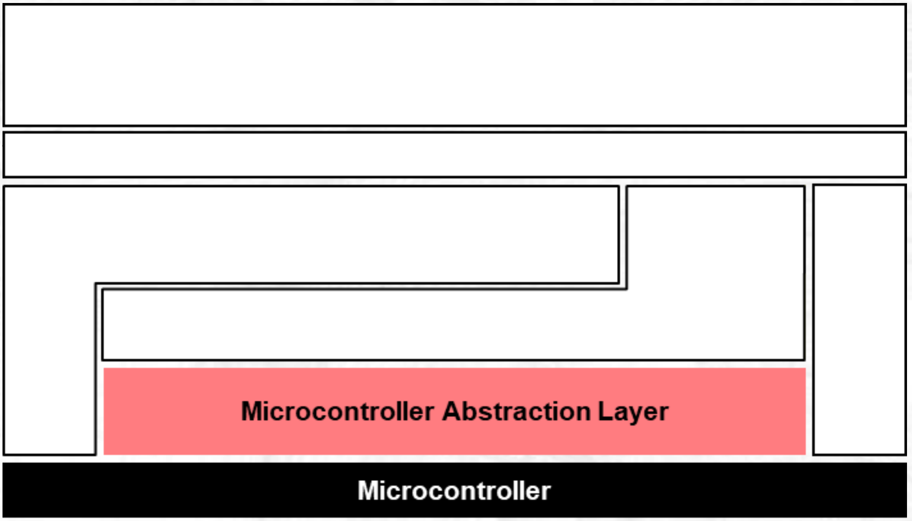
Layered Software Architecture

Elektrobit

Basic Software – Layers



Microcontroller Abstraction Layer (MCAL)



- The ***Microcontroller Abstraction Layer*** is the lowest software layer of the Basic So contains internal drivers, which are software modules with direct access to the µC peripherals and memory mapped µC external devices.

- Task:
  - Make higher software layers independent of µC

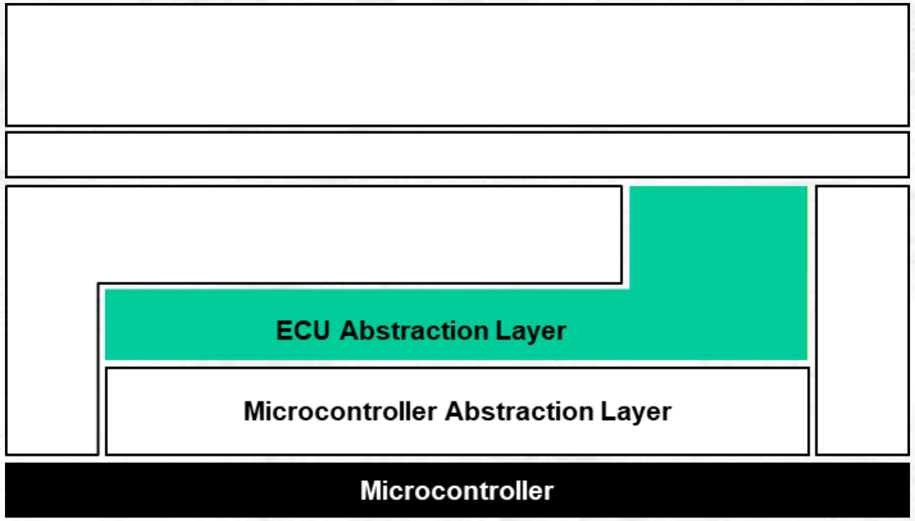
- Properties:
  - Implementation: µC dependent
  - Upper Interface: Standardized and µC independent

- MCAL functional gro

Microcontroller Drivers	Memory Drivers	Crypto Drivers	Wireless Communication Drivers	Communication Drivers	I/O Drivers

ECU Abstraction Layer

- The ***ECU Abstraction Layer***
  - interfaces the drivers of the Microcontroller Abstraction Layer.
  - contains drivers for external devices.
  - offers an API for access to peripherals and devices regardless of their location (µC internal/external) and their connection to the µC (port pins, type of interface)



- Task:
  - Make higher software layers independent of ECU hardware layout, e.g. bus types, memory devices

- Properties:
  - Implementation: µC independent, ECU hardware dependent
  - Upper Interface: µC and ECU hardware independent, dependent on signal type

- ECU abstraction layer functional groups:

Onboard Device Abstraction

Memory Hardware Abstraction

Crypto Hardware Abstraction

Wireless Communication HW Abstraction

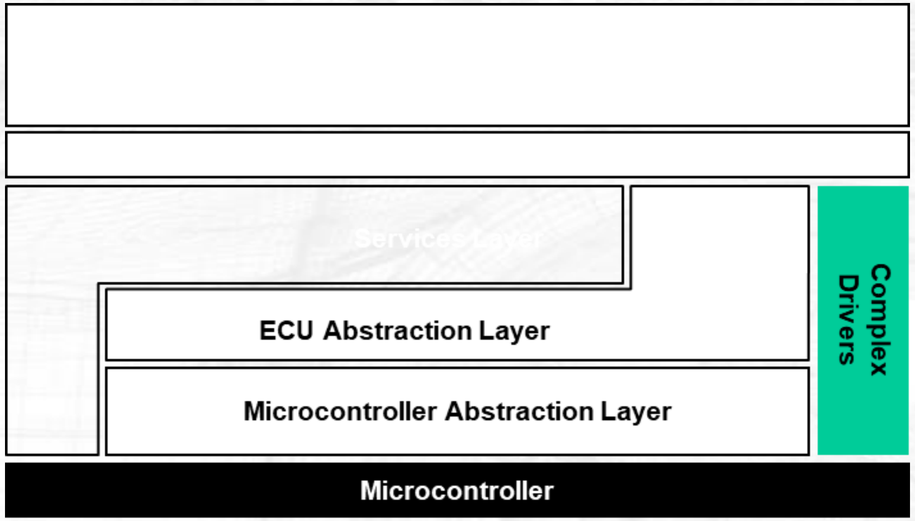
Communication Hardware Abstraction

Complex Drivers

- The **Complex Drivers Layer** spans from the hardware to the RTE.

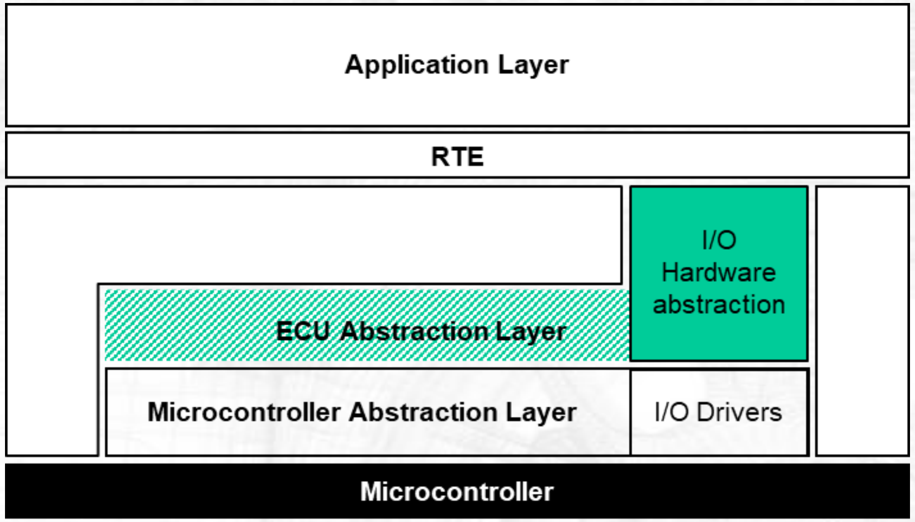
- Task
  - Provide the possibility to integrate special purpose functionality, e.g. drivers for devices:
    - which are not specified within AUTOSAR,
    - with very high timing constrains or
    - for migration purposes etc.

- Properties:
  – Implementation: might be application, µC and ECU hardware dependent
  – Upper Interface: ECU specific modeling of AUTOSAR interface description



I/O Hardware Abstraction

- The I/O Hardware Abstraction belongs to the ECU Abstraction Layer
- It abstracts from the location of peripheral I/O devices (on-chip or on-board) and the ECU hardware layout (e.g. µC pin connections and signal level inversions)
- Its upper layer is the RTE but there are no standardized AUTOSAR interfaces (i.e. no Service Layer modules which could provide these interfaces to the RTE)
  - Details are project specific - AUTOSAR provides only high-level requirements and guidelines



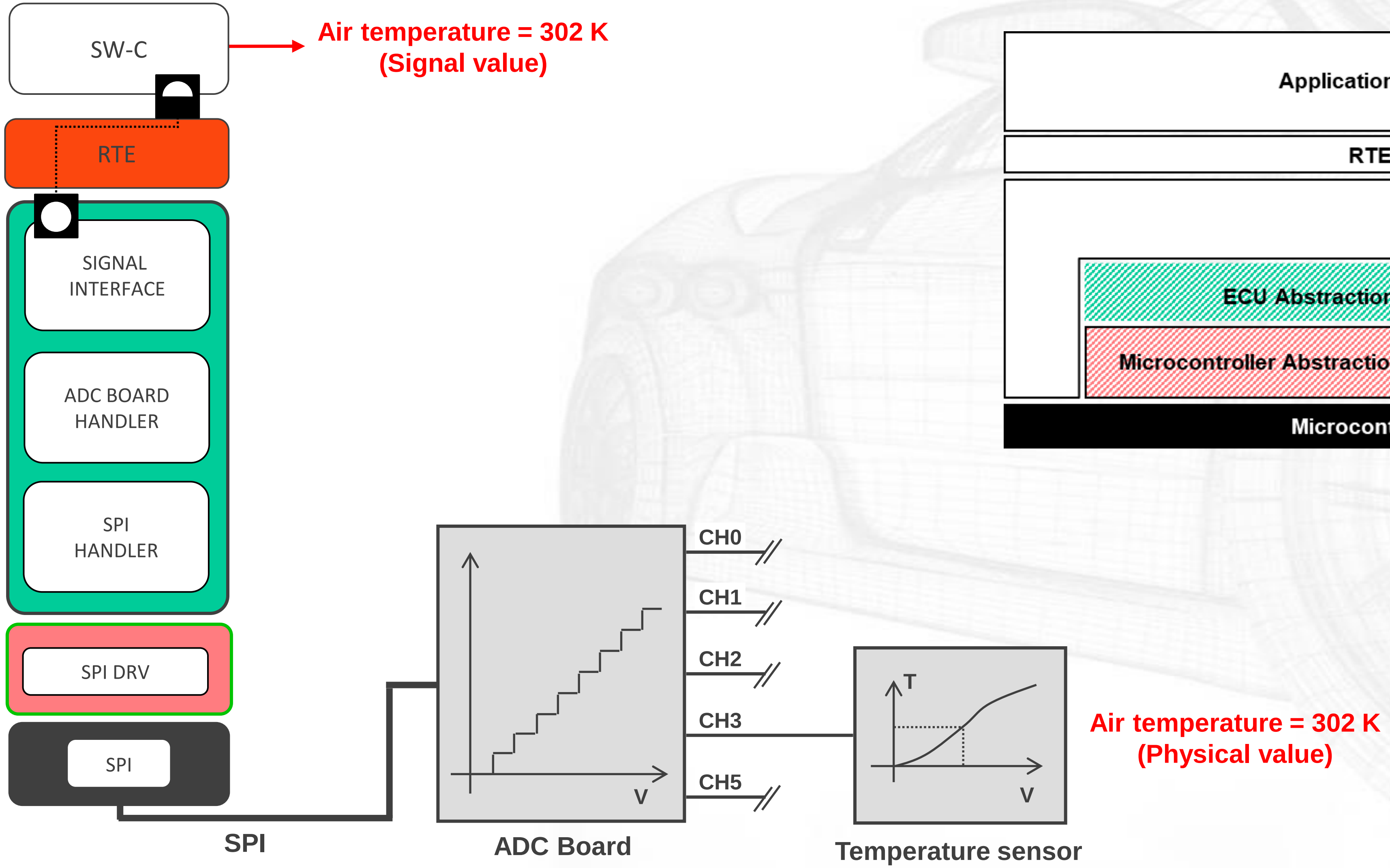
Task:

- Represent I/O signals as they are connected to the ECU hardware (e.g. current, voltage, frequency).
- Hide ECU hardware and layout properties from higher software layers.

Properties:

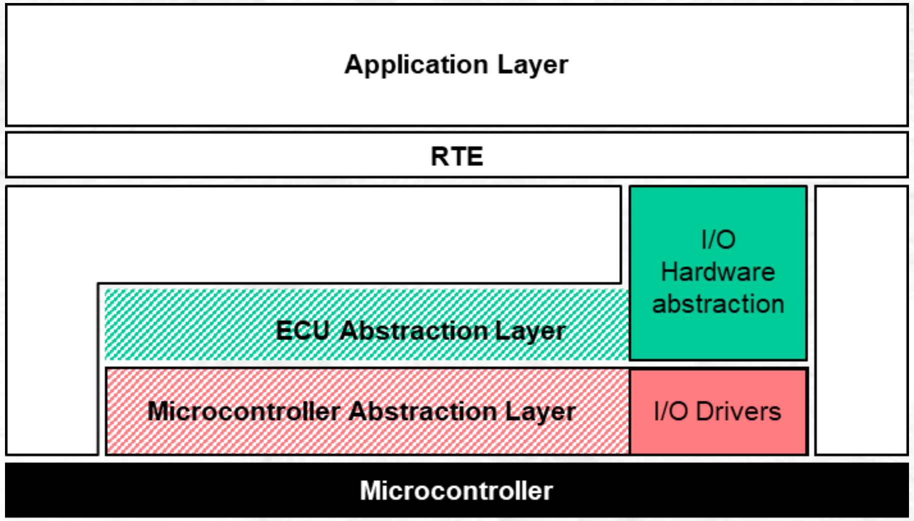
- Implementation: µC independent, ECU hardware dependent
- Upper Interface: µC and ECU hardware independent, dependent on signal type specified and implemented according to AUTOSAR (AUTOSAR interface)

I/O Hardware Abstraction - Example



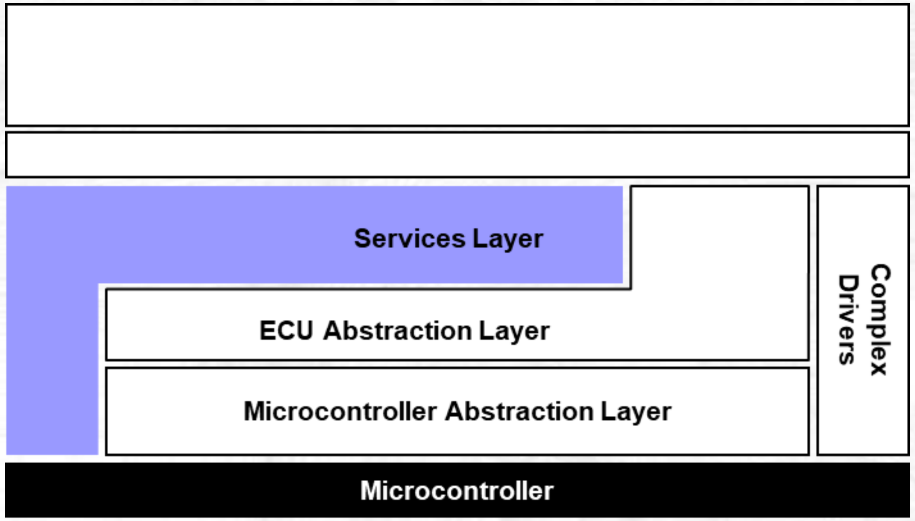
Air temperature = 302 K
(Signal value)

Air temperature = 302 K
(Physical value)



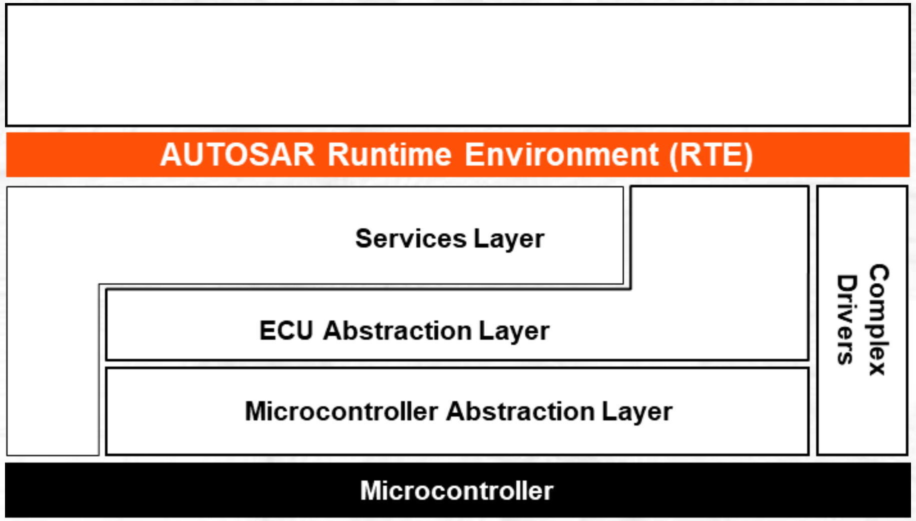
Services Layer

- The ***Service Layer*** is the highest layer of the Basic Software
- While access to I/O signals is covered by the ECU Abstraction Layer, the Services Layer offers:
  - Operating system functionality
  - Vehicle network communication and management services
  - Memory services (NVRAM management)
  - Diagnostic Services (including UDS communication, error memory and fault treatment)
  - ECU state management, mode management
  - Logical and temporal program flow monitoring (Wdg manager)
- Task:
  - Provide basic services for application, RTE and basic software modules.
- Properties:
  - Implementation: Mostly µC, ECU hardware independent (Exception: Os)
  - Upper Interface: µC and ECU hardware independent



Runtime Environment (RTE)

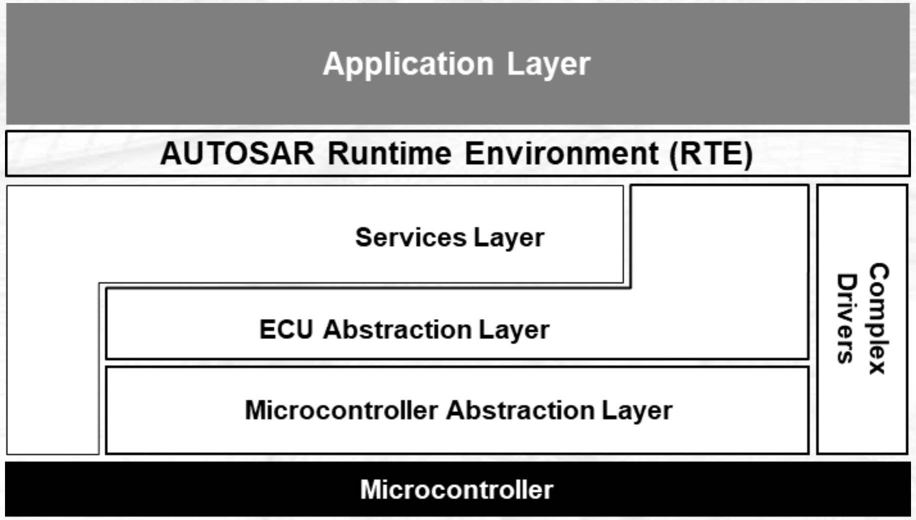
- The **RTE** is a layer providing communication services to the application software (AUTOSAR Software Components and/or AUTOSAR Sensor/Actuator components).
- Above the RTE the software architecture style changes from “layered“ to “component style“.
- The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) and/or services via the RTE.


AUTOSAR Runtime Environment (RTE)
Services Layer
ECU Abstraction Layer
Microcontroller Abstraction Layer
Complex Drivers
Microcontroller

- Task:
  - Make AUTOSAR Software Components independent from the mapping to a specific ECU
- Properties:
  - Implementation: ECU and application specific (generated individually for each ECU)
  - Upper Interface: Completely ECU independent

Application Layer

- The ***Application Layer*** is a layer consisting of the application software:
  - AUTOSAR Software Components and/or
  - AUTOSAR Sensor/Actuator components).
- Above the RTE the software architecture style changes from “layered“ to “component style“. The AUTOSAR Software Components communicate with other components (inter and/or intra ECU) and/or services via the RTE.

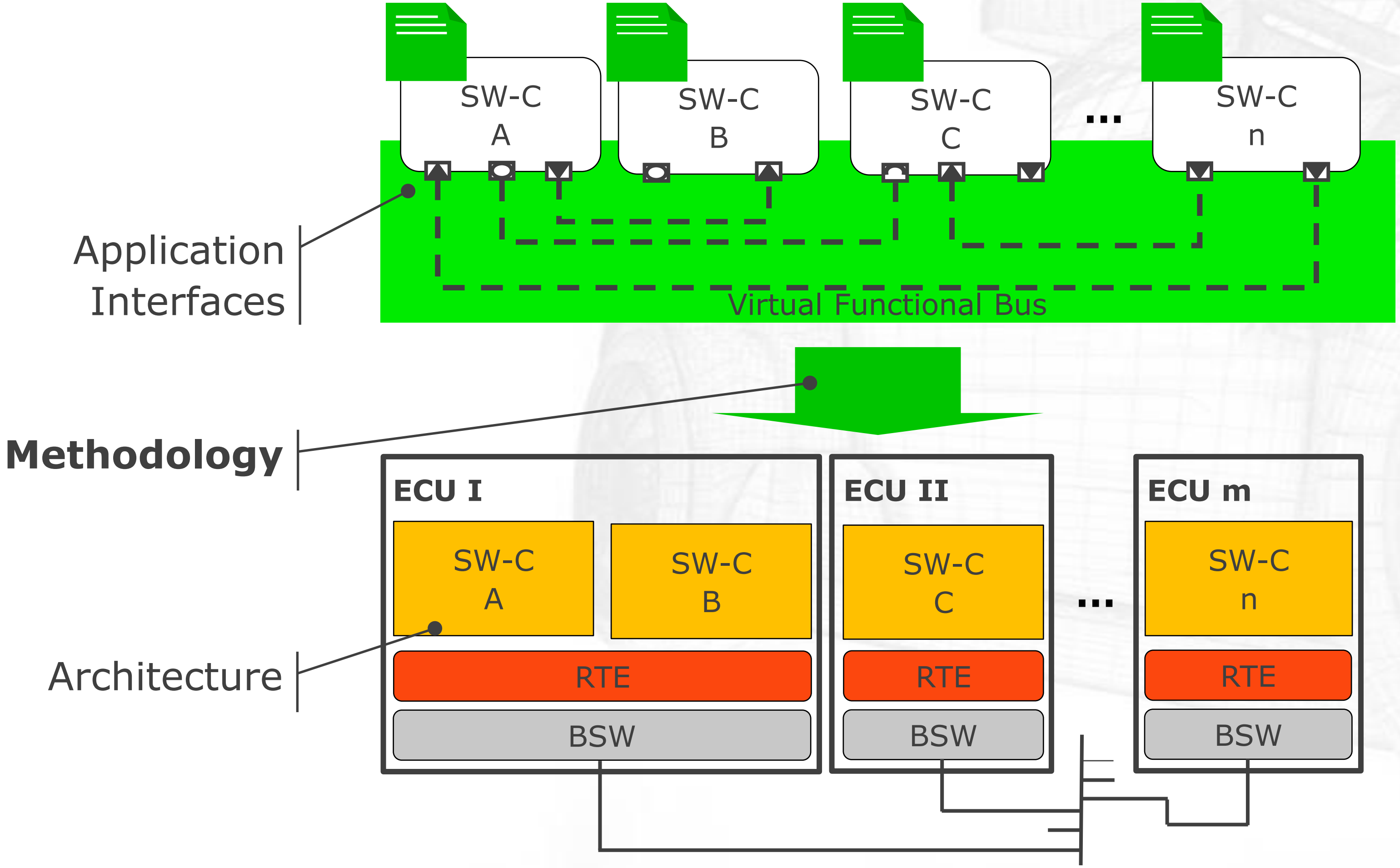


- Task:
  - Implement applications (runnables) that are executed by the RTE

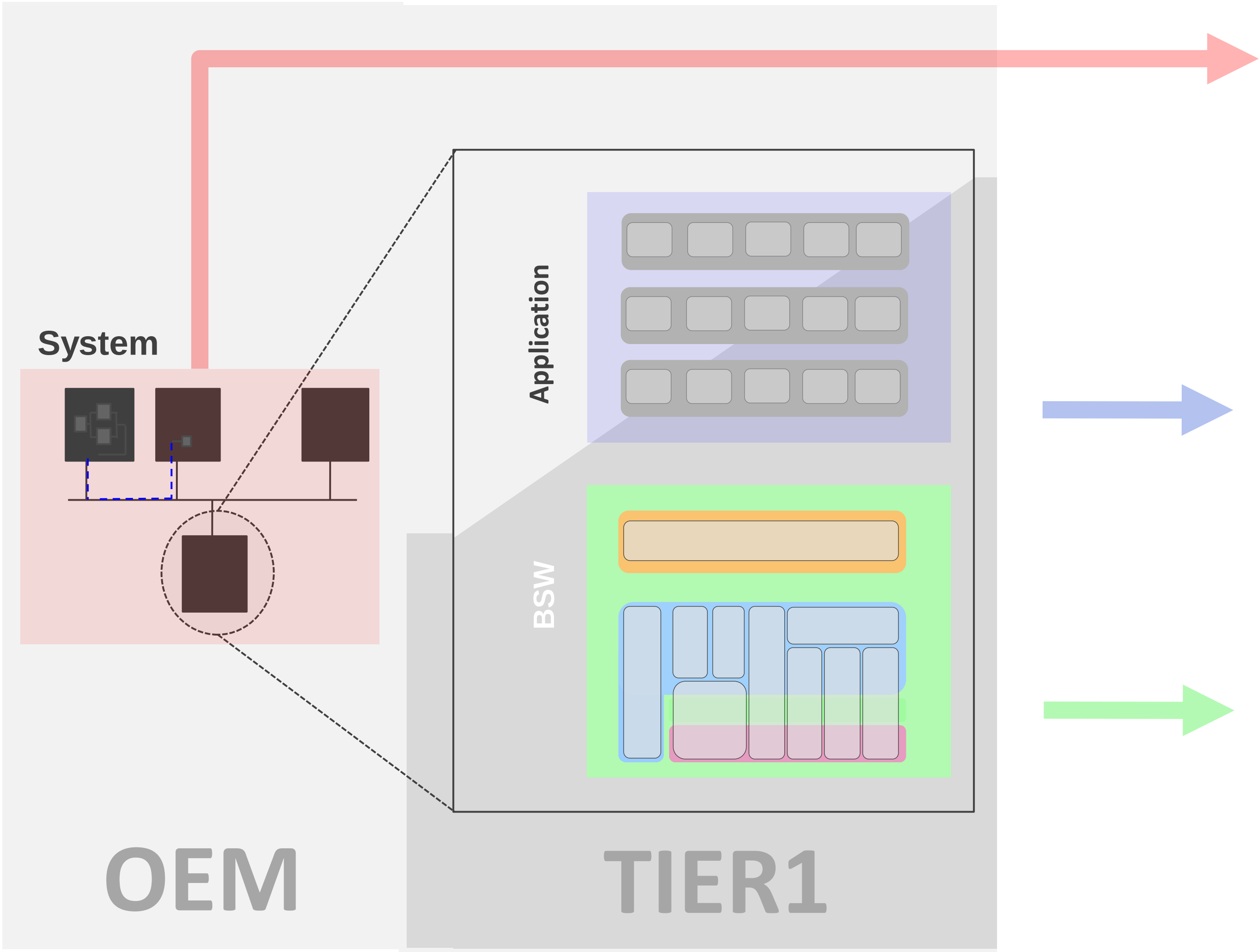
- Properties:
  - Applications completely ECU independent.
  - Sensor/Actuator SW-Cs are dependent on the specifics of a sensor or actuator.

Methodology

Methodology: From Virtual Function Bus to ECU level



AUTOSAR Methodology



AUTOSAR

Exchange Files

AUTOSAR SYS-D *.arxml*

AUTOSAR SWC-D *.arxml*

AUTOSAR BSW-Config *.epc, .xdm*

Tooling

Systemdesign, Software Architecture

- Define Hardware Topology
- Define SWCs, Runnables, Data
- Mapping of SWCs to ECUs
- Communication Matrix
- Export as AUTOSAR Sys-D

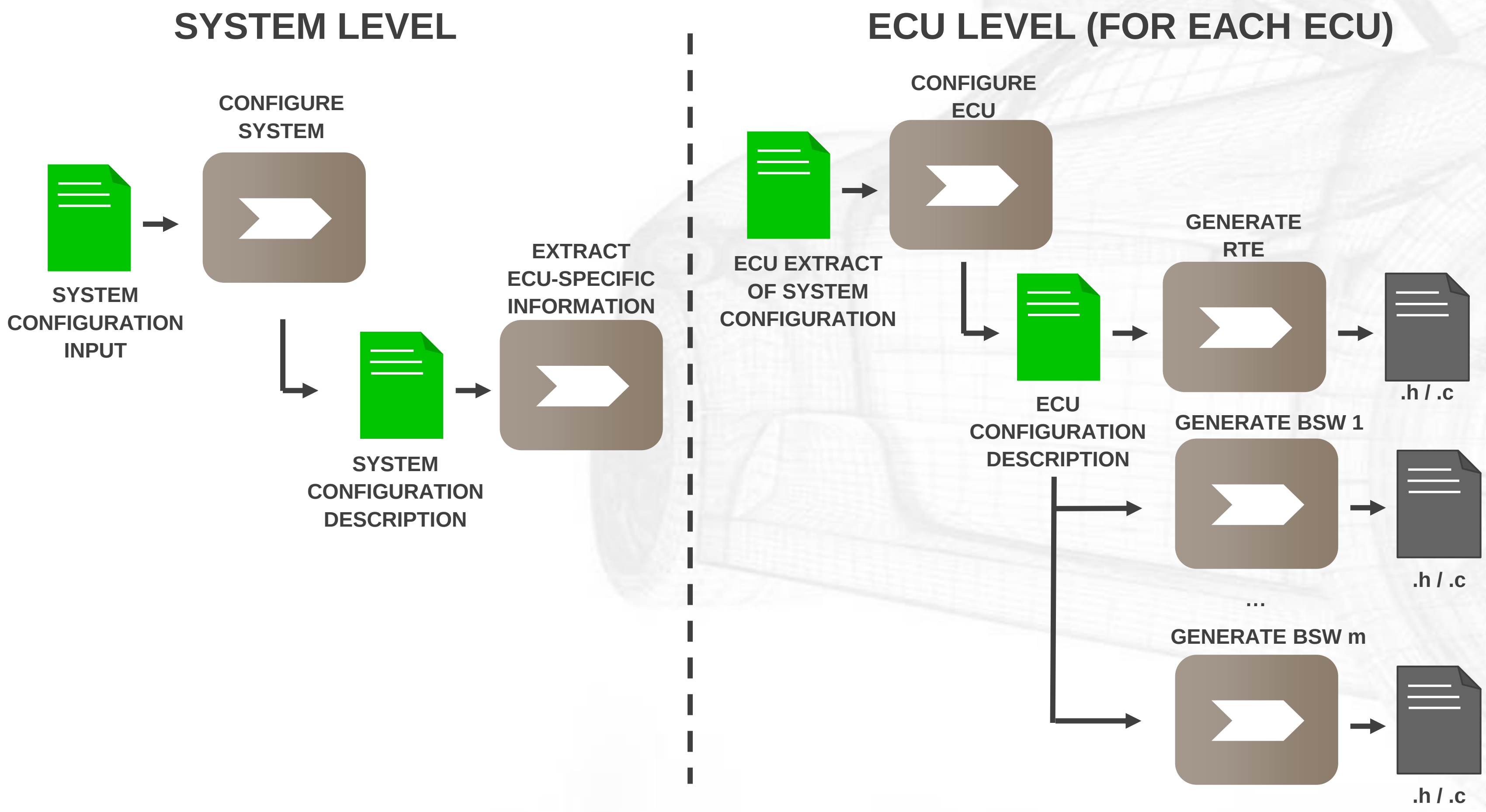
Definition of ECU Application (SWC)

- Model Application Behaviour
- Define ports and data types
- Create SWC Description
- Export SWC Description
- Generate application code

Configuration of ECU Basic Software

EB tresos Studio
EB tresos AutoCore

Configuration



SYSTEM LEVEL

CONFIGURE SYSTEM

SYSTEM CONFIGURATION INPUT

EXTRACT ECU-SPECIFIC INFORMATION

SYSTEM CONFIGURATION DESCRIPTION

ECU LEVEL (FOR EACH ECU)

CONFIGURE ECU

ECU EXTRACT OF SYSTEM CONFIGURATION

GENERATE RTE

.h / .c

ECU CONFIGURATION DESCRIPTION

GENERATE BSW 1

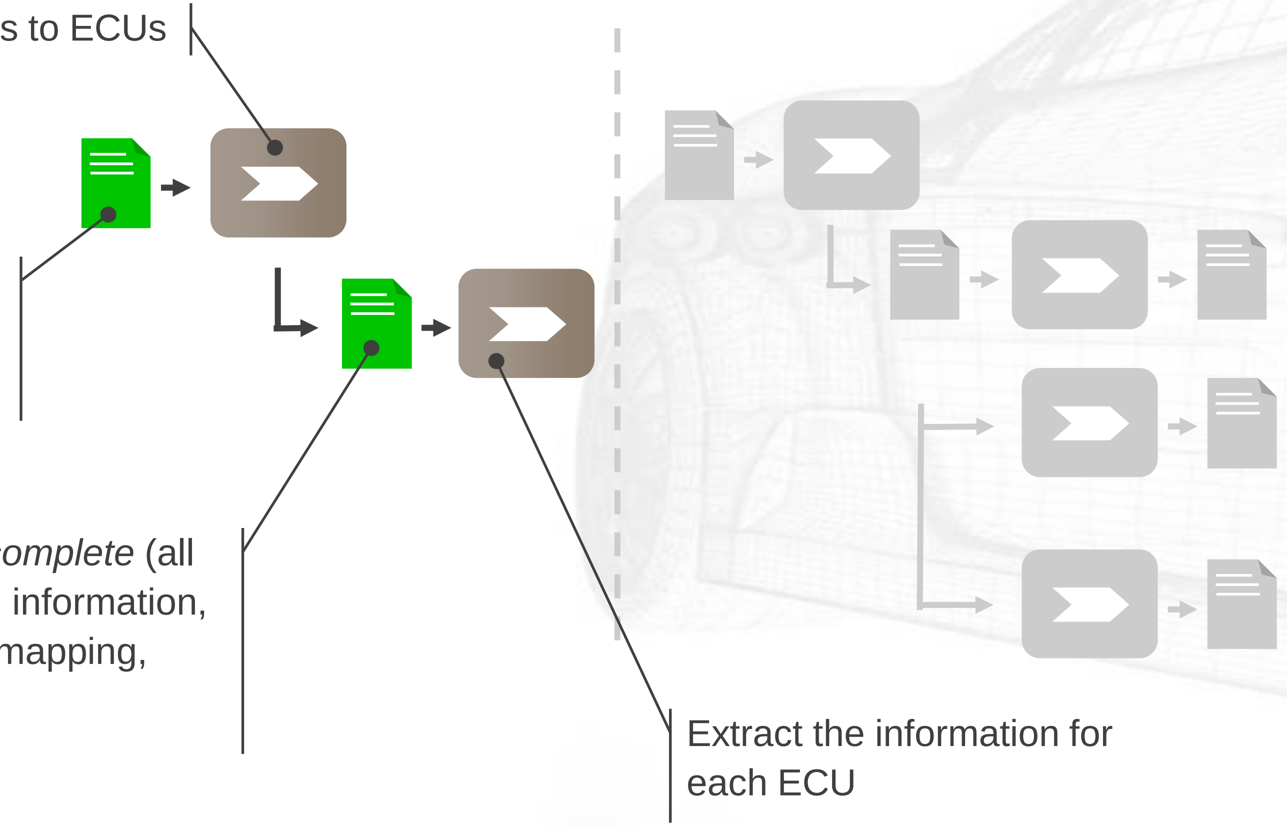
.h / .c

…

GENERATE BSW m

.h / .c

System Description



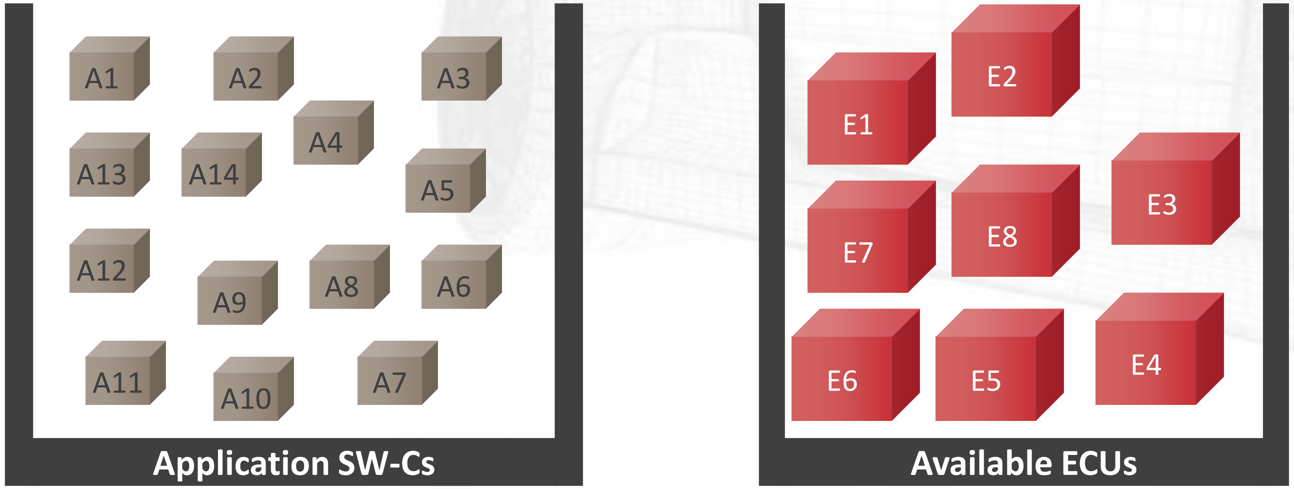
Map SW-Cs to ECUs

Defines SW-Cs
Defines ECUs
Defines Topology

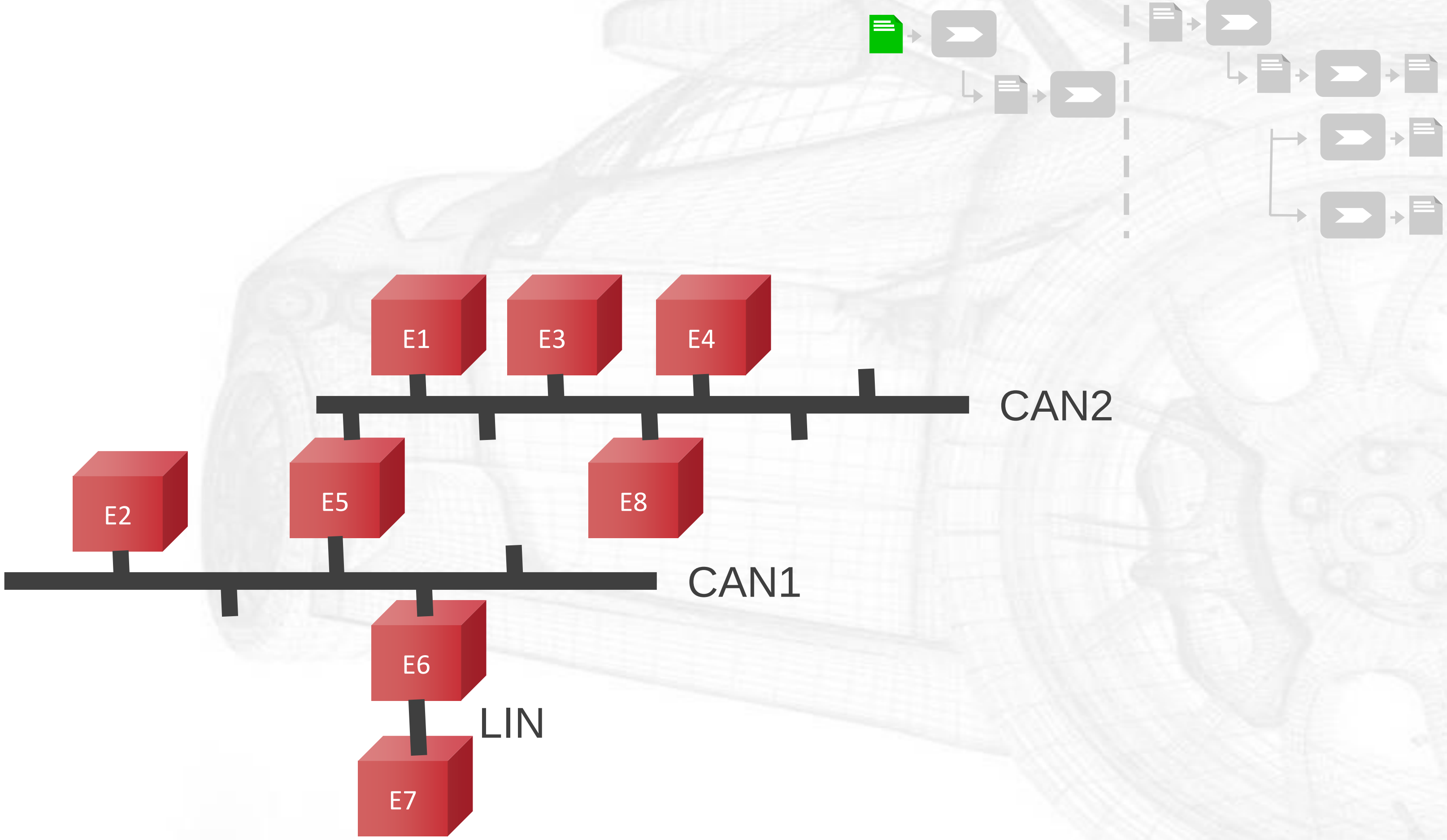
Contains the *complete* (all ECUs) system information, including bus-mapping, topology etc.

Extract the information for each ECU

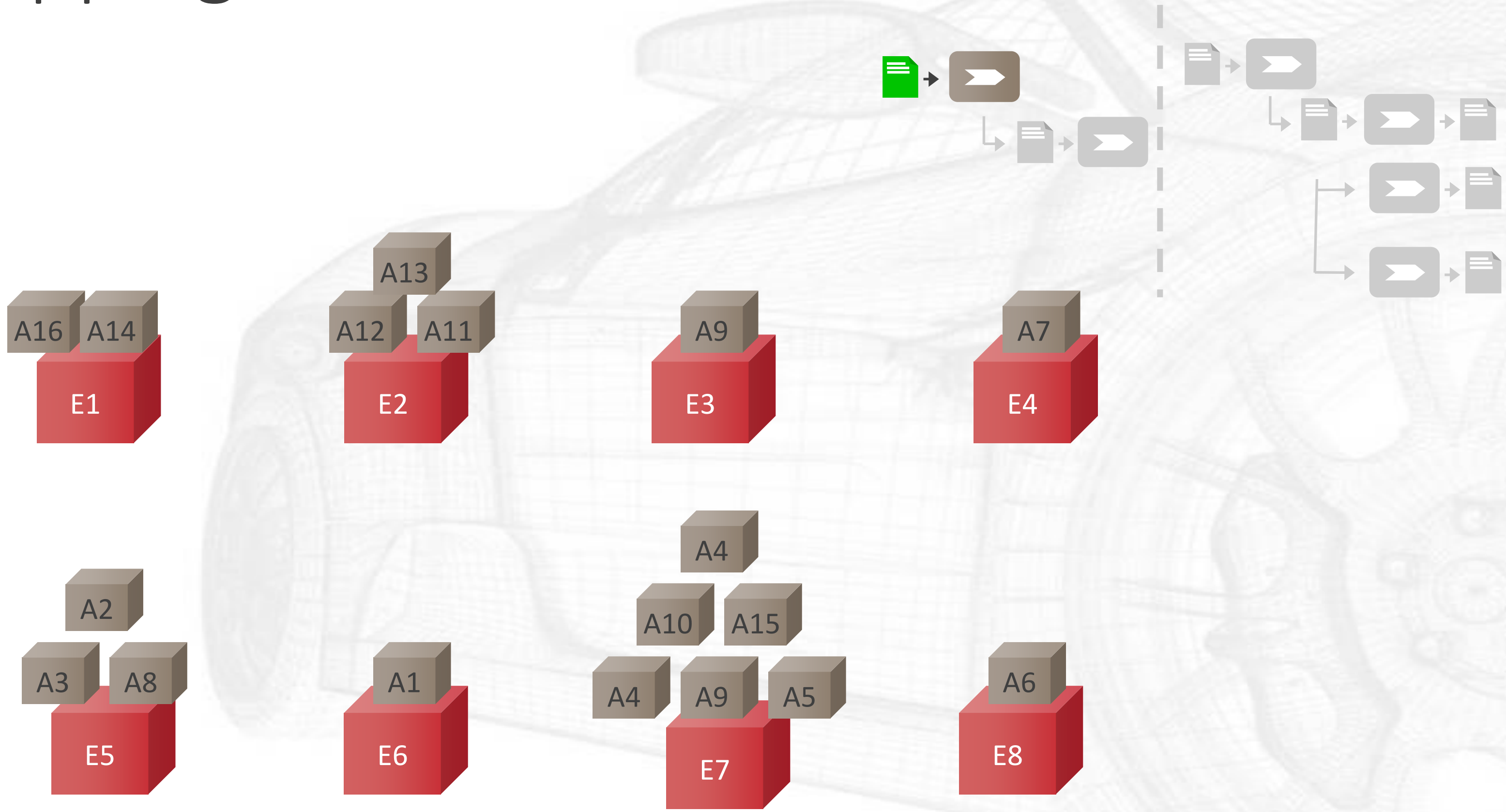
Example: Define SW-Cs and ECUs



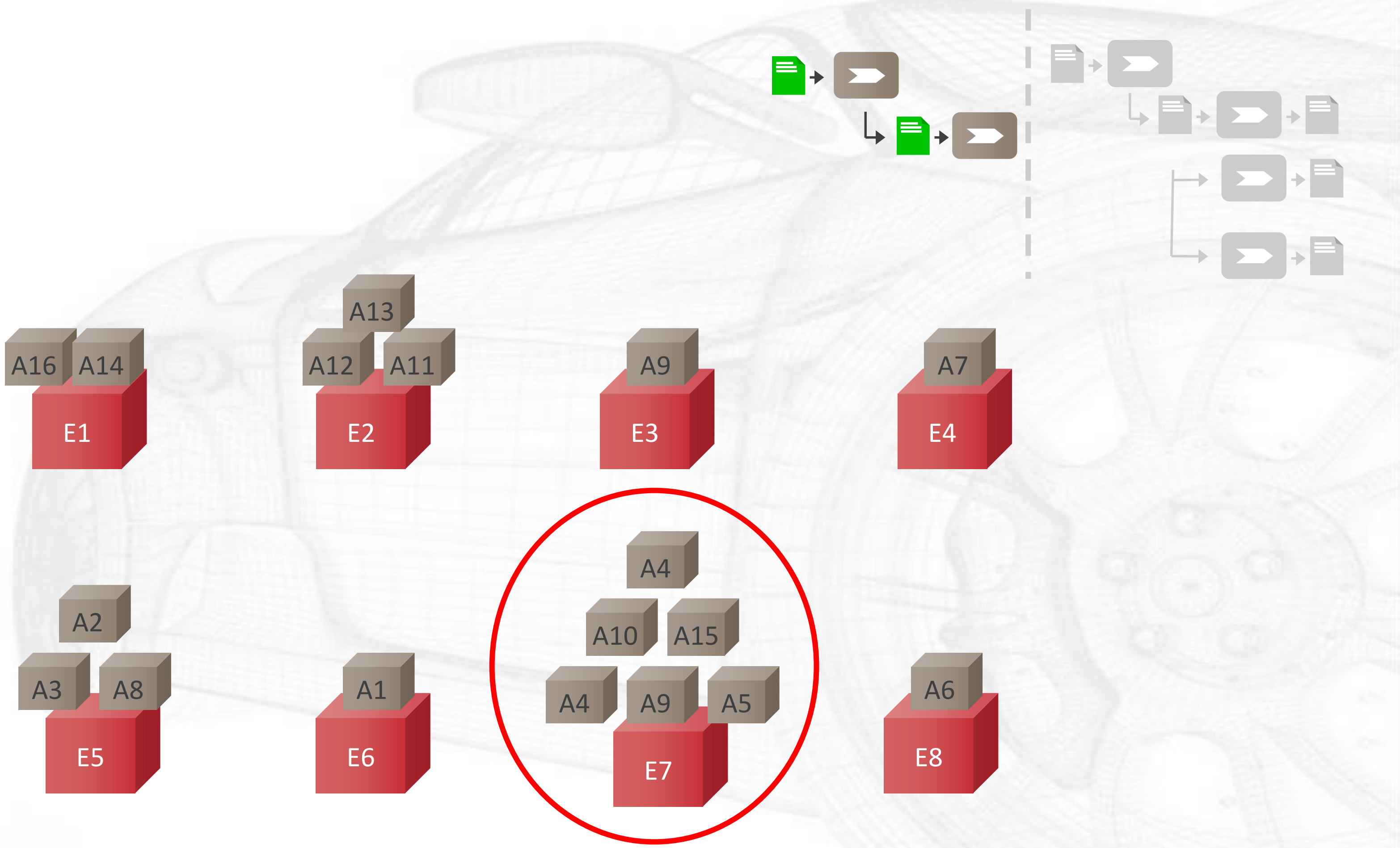
Example: Topology



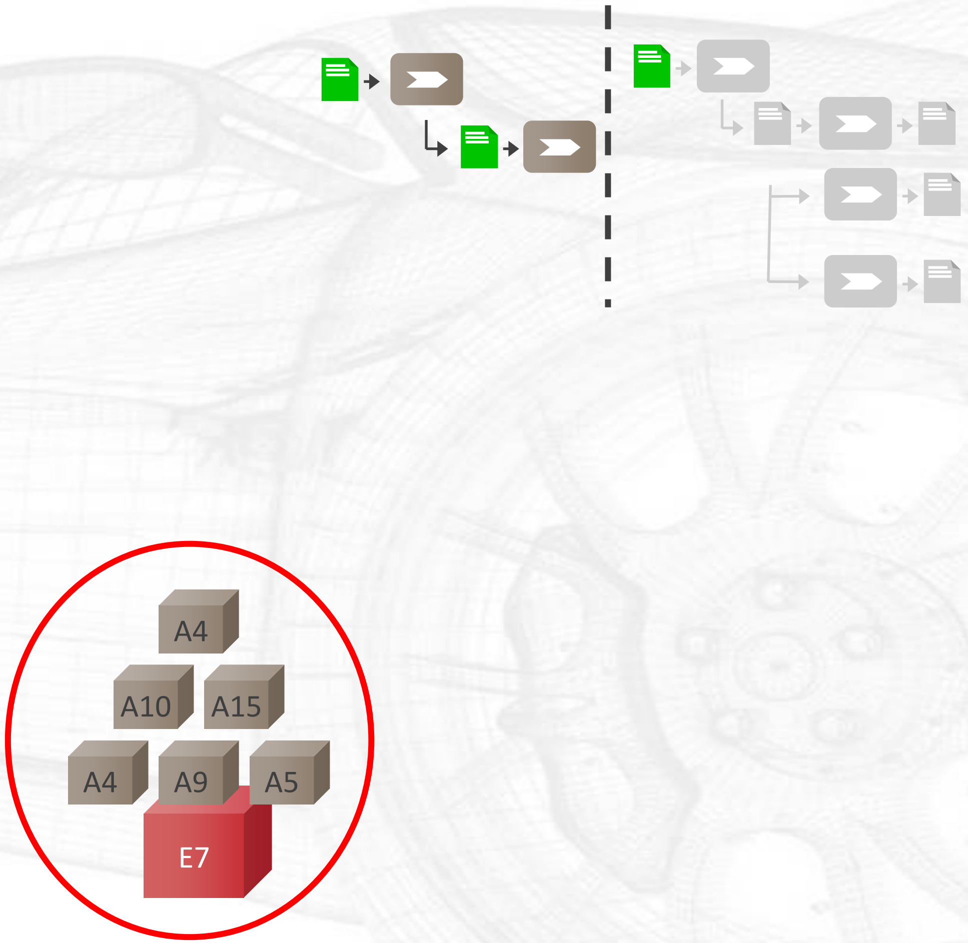
Example: SW-C Mapping



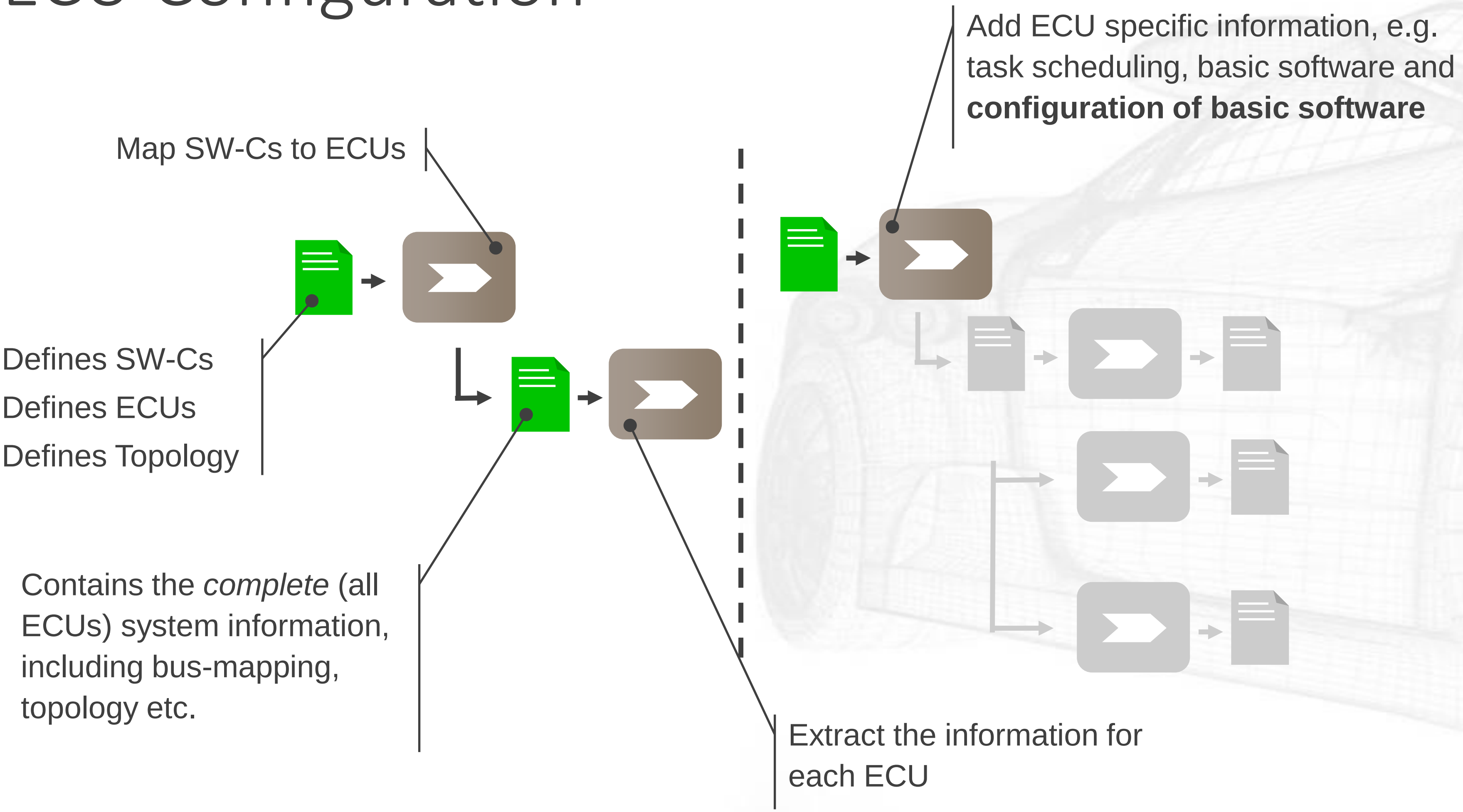
Example: ECU Extract



Example: ECU Extract



ECU Configuration



Map SW-Cs to ECUs

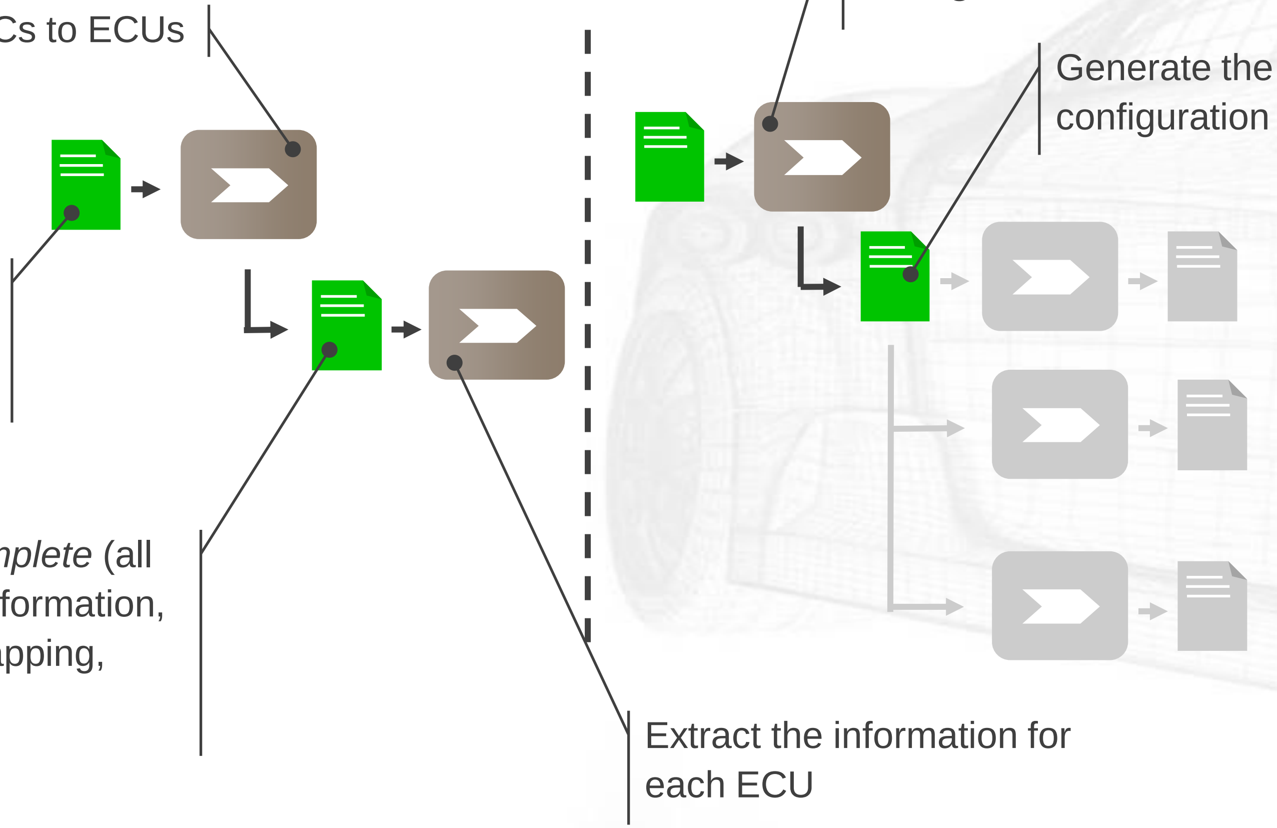
Defines SW-Cs
Defines ECUs
Defines Topology

Contains the *complete* (all ECUs) system information, including bus-mapping, topology etc.

Extract the information for each ECU

Add ECU specific information, e.g. task scheduling, basic software and **configuration of basic software**

ECU Configuration



Map SW-Cs to ECUs

Defines SW-Cs
Defines ECUs
Defines Topology

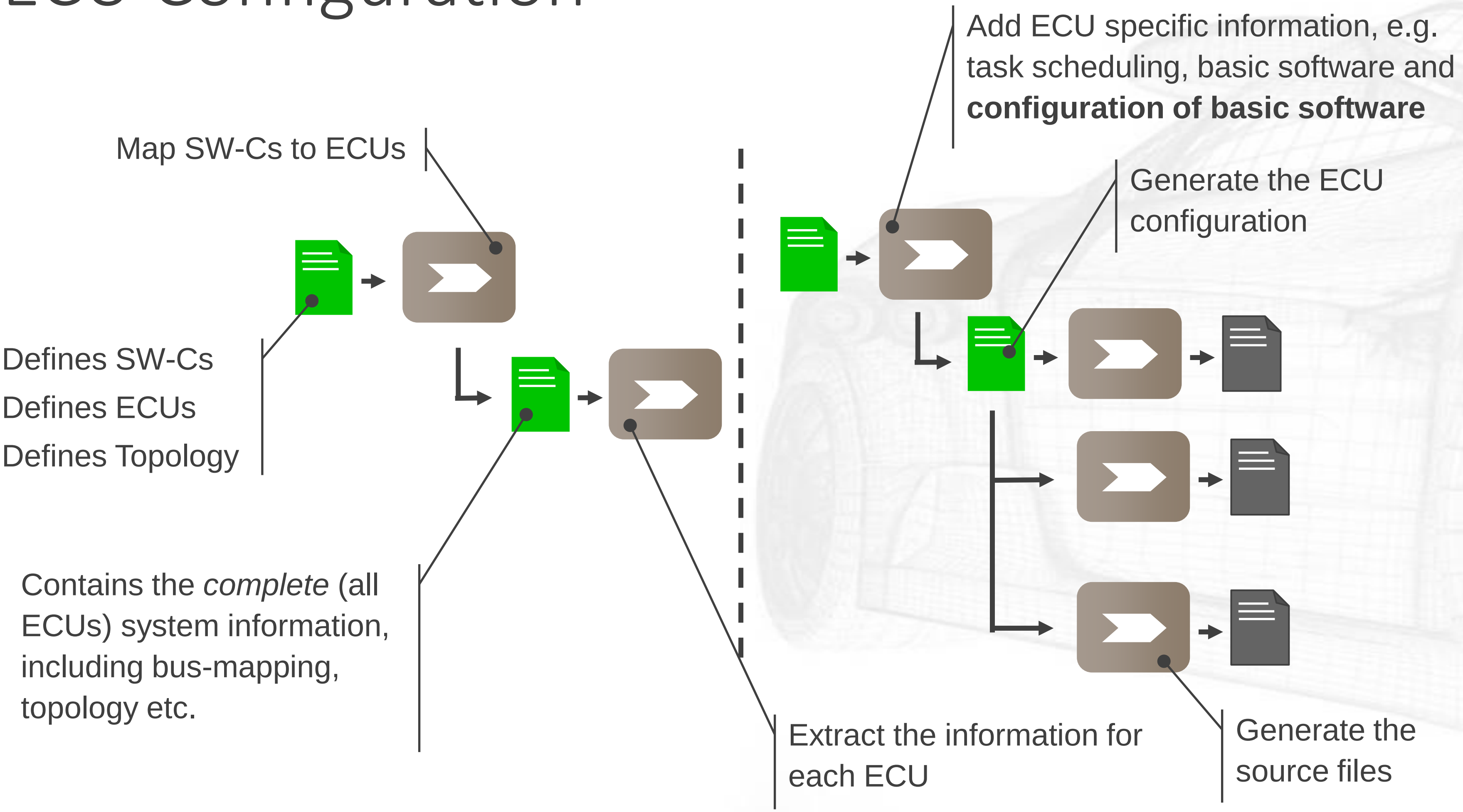
Contains the *complete* (all ECUs) system information, including bus-mapping, topology etc.

Add ECU specific information, e.g. task scheduling, basic software and **configuration of basic software**

Generate the ECU configuration

Extract the information for each ECU

ECU Configuration



Add ECU specific information, e.g. task scheduling, basic software and **configuration of basic software**

Map SW-Cs to ECUs

Generate the ECU configuration

Defines SW-Cs

Defines ECUs

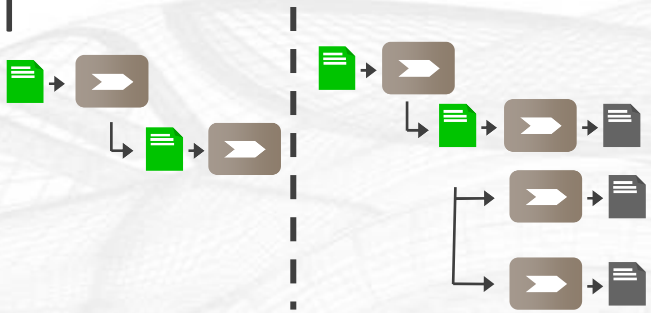
Defines Topology

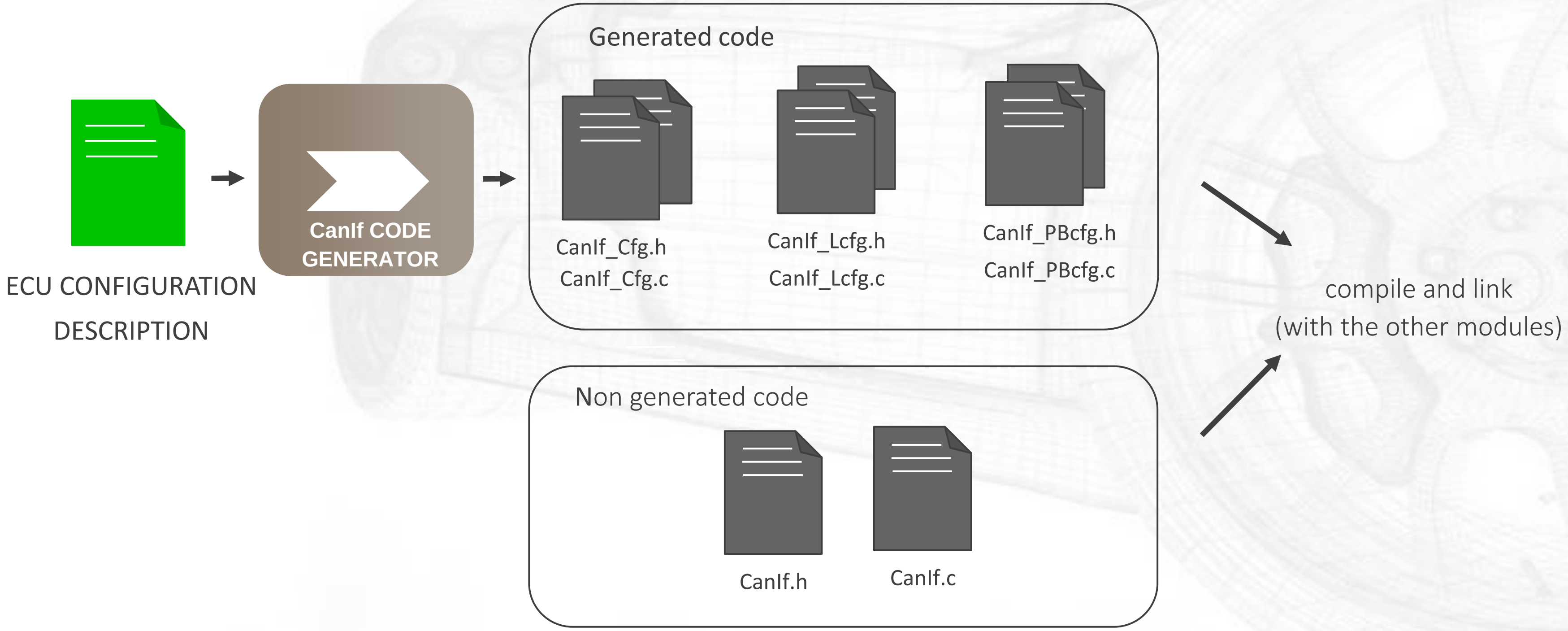
Contains the *complete* (all ECUs) system information, including bus-mapping, topology etc.

Extract the information for each ECU

Generate the source files

Example: Code Generation for Module CanIf





ECU CONFIGURATION DESCRIPTION

CanIf CODE GENERATOR

Generated code

CanIf_Cfg.h
CanIf_Cfg.c

CanIf_Lcfg.h
CanIf_Lcfg.c

CanIf_PBcfg.h
CanIf_PBcfg.c

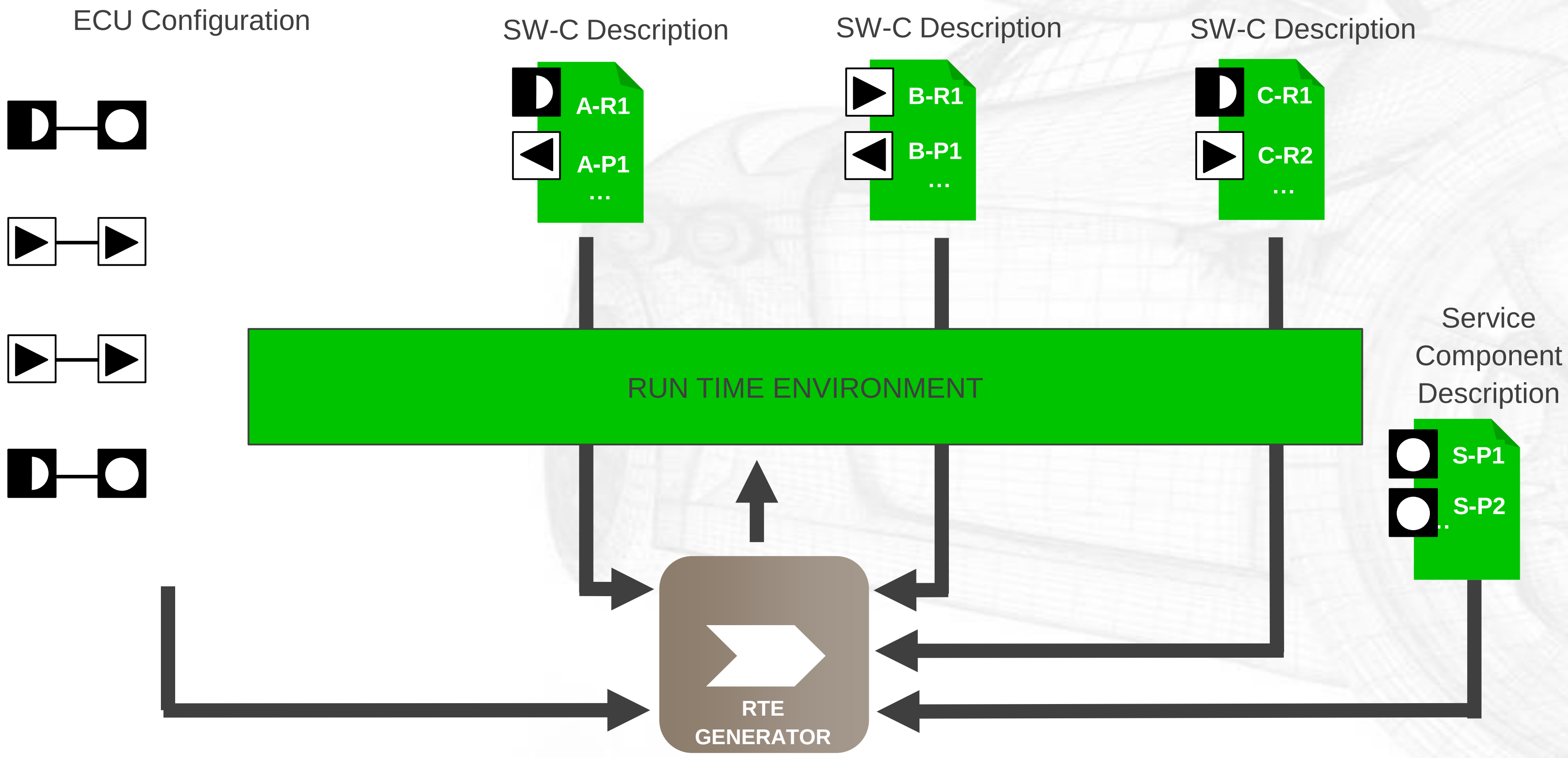
Non generated code

CanIf.h

CanIf.c

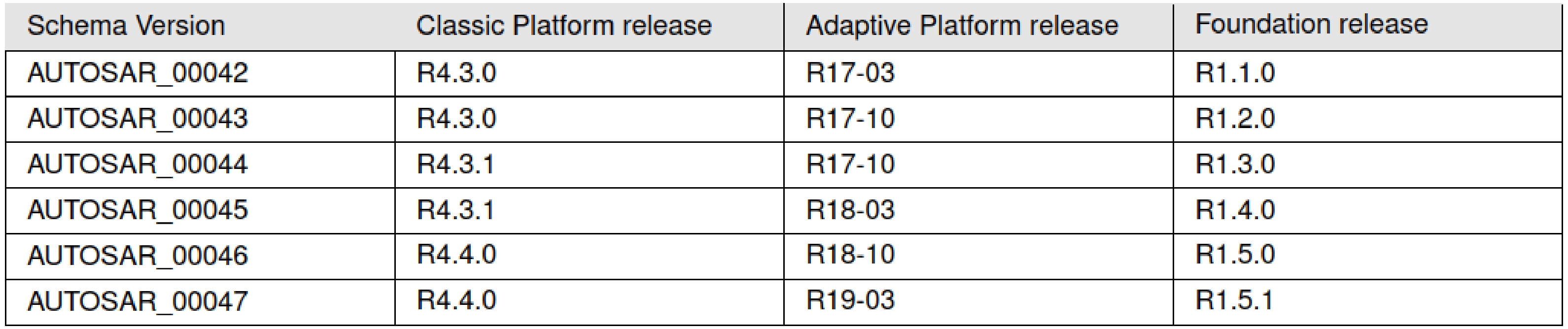
compile and link
(with the other modules)

Example: Code Generation for the RTE

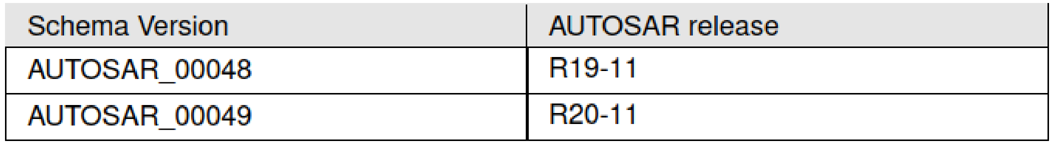


Overview of AUTOSAR schema versions vs. AUTOSAR release

- The AUTOSAR schema version needs to be referenced in all ARXML files
- Note that with the introduction of the Adaptive Platform, the schema version does not equal the Platform release anymore:



Schema Version	Classic Platform release	Adaptive Platform release	Foundation release
AUTOSAR_00042	R4.3.0	R17-03	R1.1.0
AUTOSAR_00043	R4.3.0	R17-10	R1.2.0
AUTOSAR_00044	R4.3.1	R17-10	R1.3.0
AUTOSAR_00045	R4.3.1	R18-03	R1.4.0
AUTOSAR_00046	R4.4.0	R18-10	R1.5.0
AUTOSAR_00047	R4.4.0	R19-03	R1.5.1

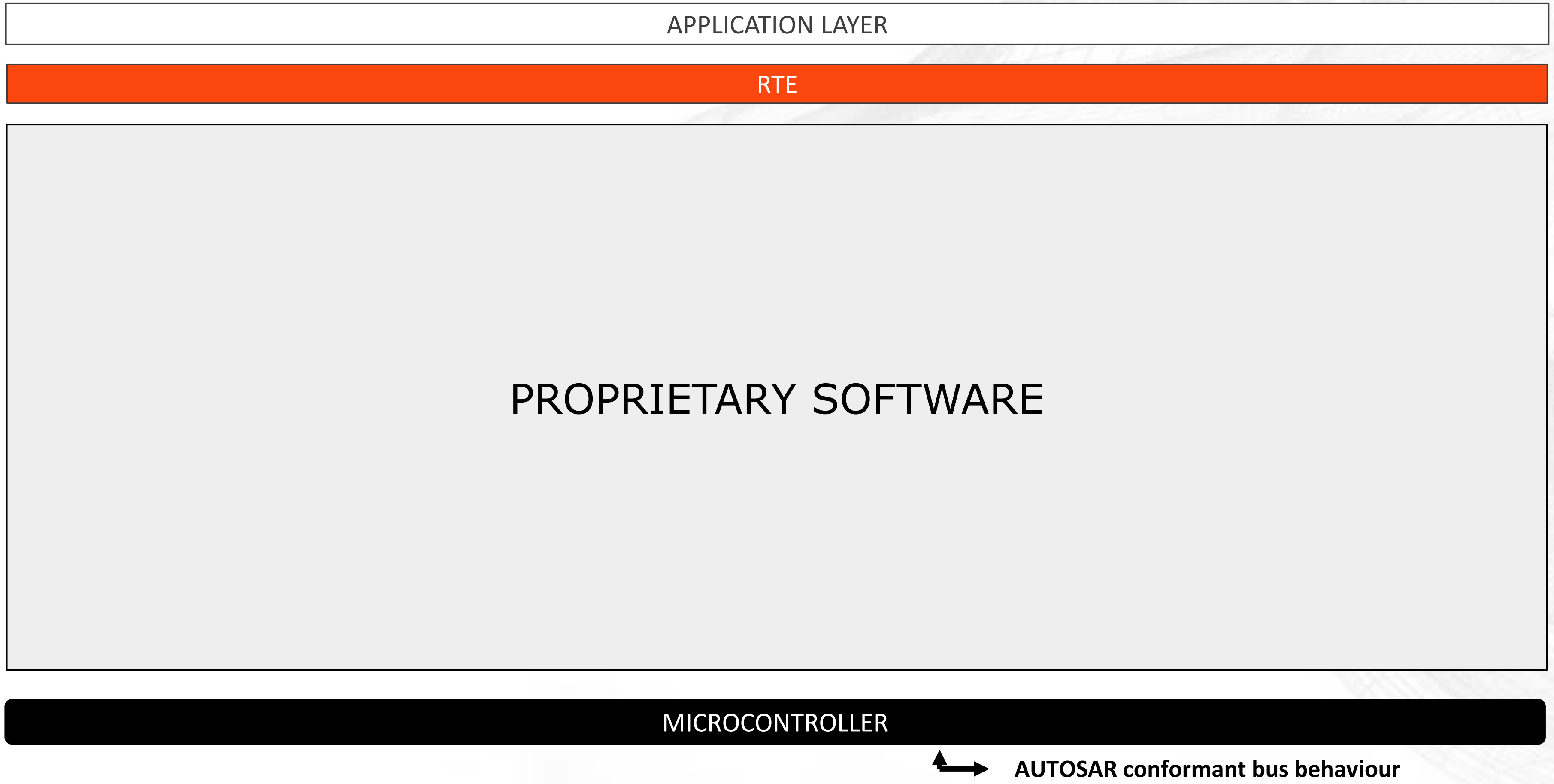


Schema Version	AUTOSAR release
AUTOSAR_00048	R19-11
AUTOSAR_00049	R20-11

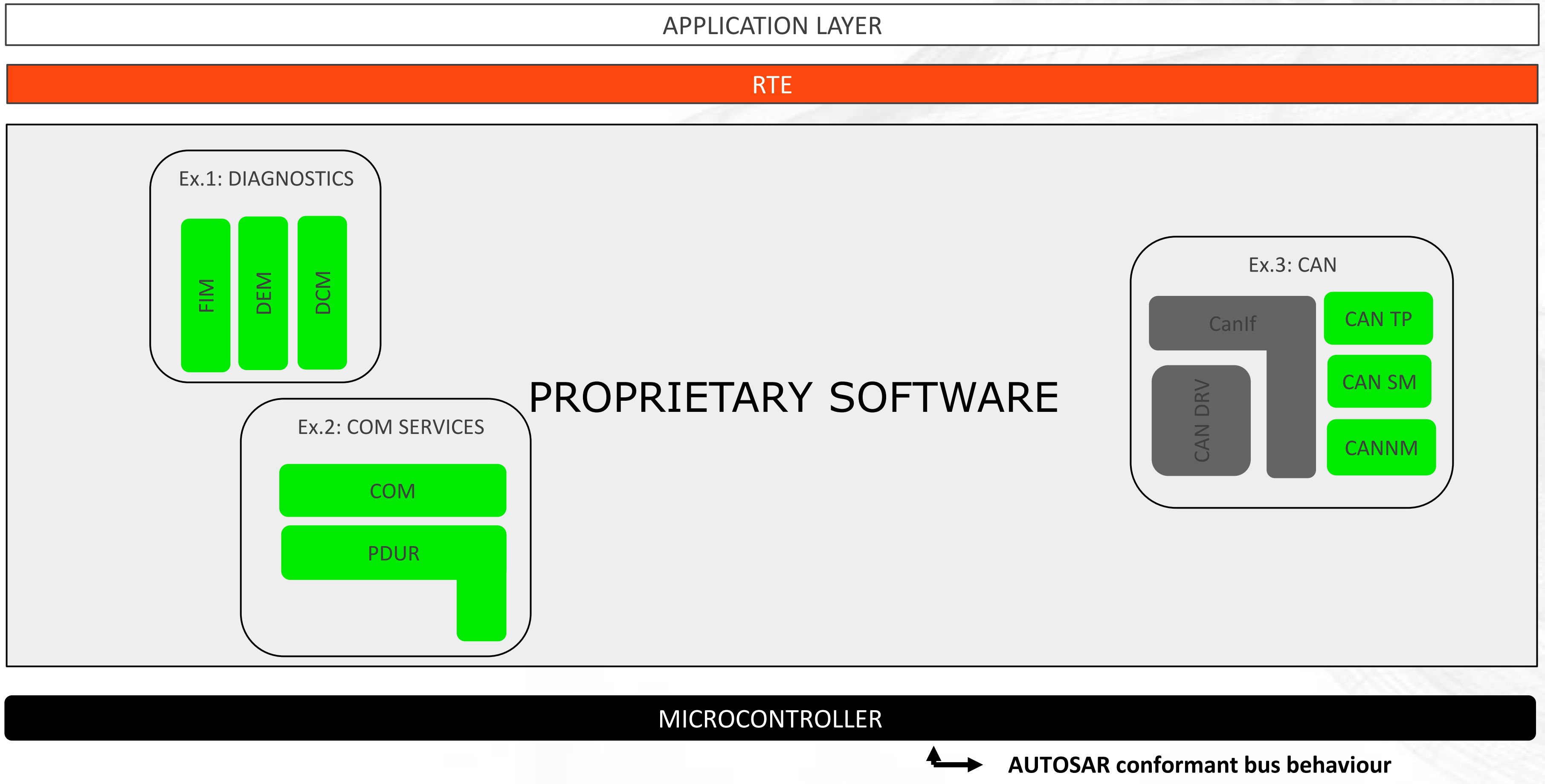
Migration & Integration Strategies

Elektrobit

Implementation Conformance Class 1 / ICC1

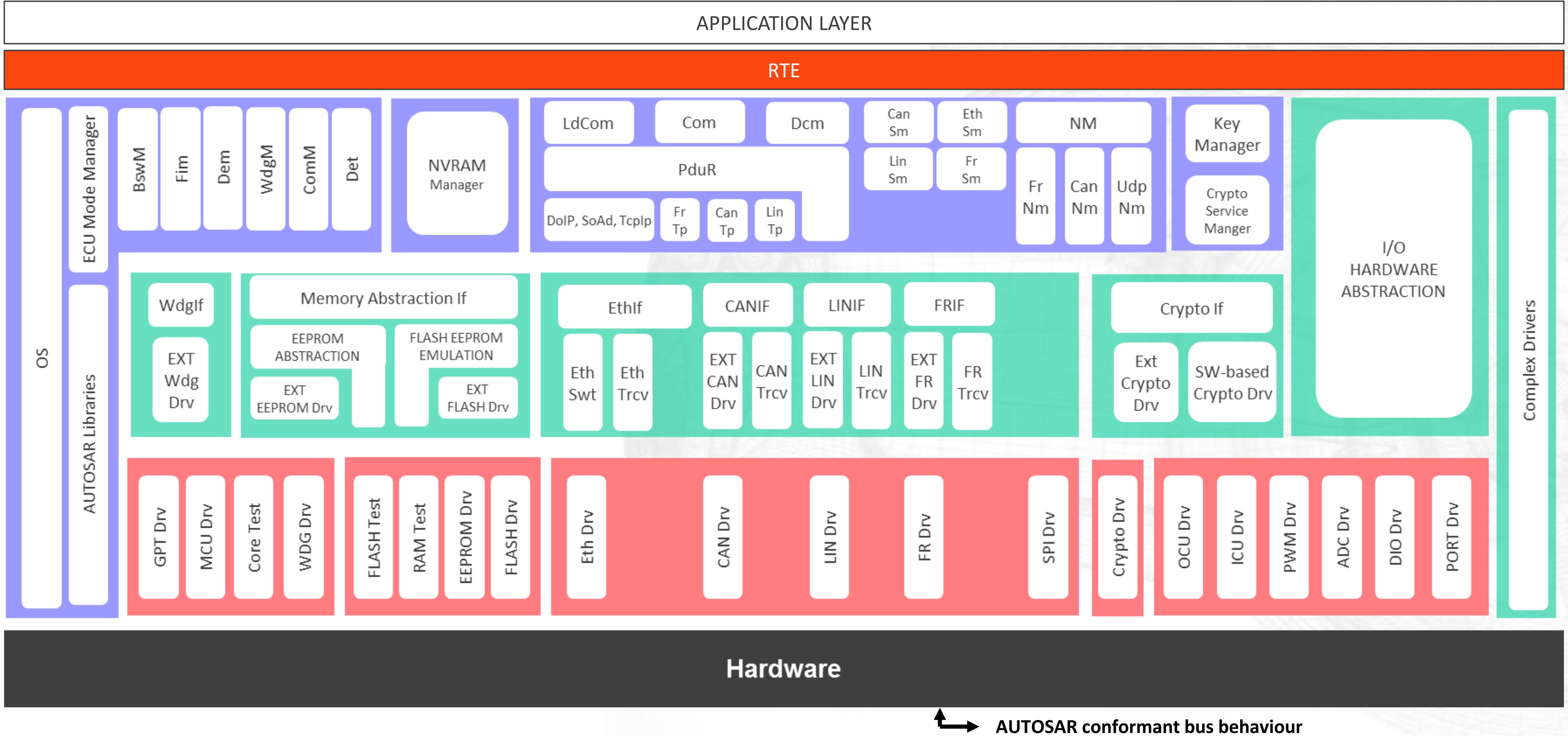


Implementation Conformance Class 2 / ICC2



AUTOSAR conformant bus behaviour

Implementation Conformance Class 3 / ICC3



Configuration Classes of Parameters

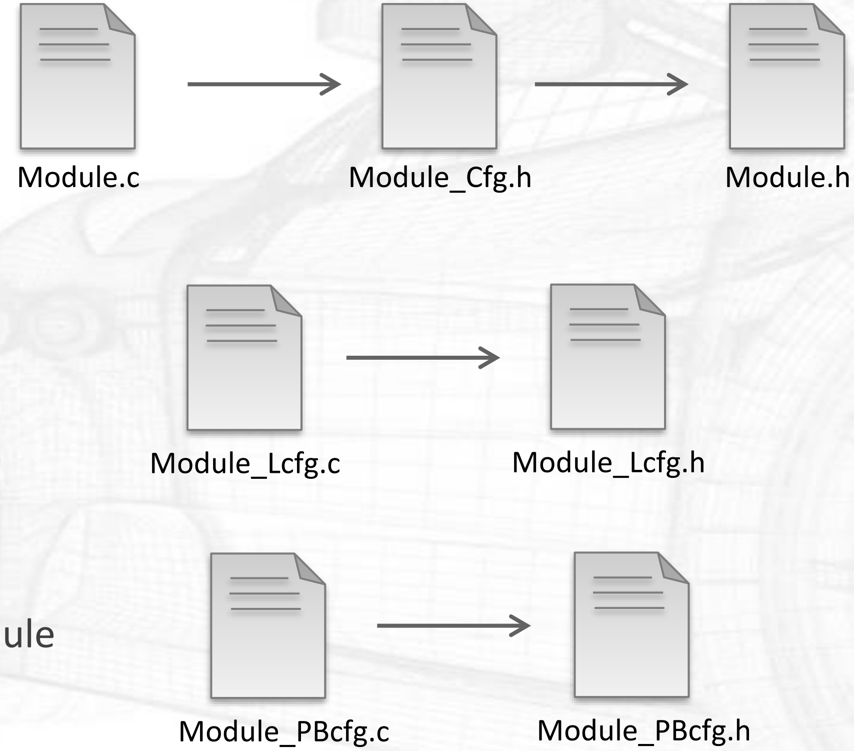
Configuration Class of Parameter	Attribute / Implementation of Parameter
Pre-compile time	▪ Can be changed at precompile time ▪ Optimization of performance and code size → Usually #defines
Link time	▪ Can be changed at link time or precompile time ▪ Usually implemented as const qualified variables ▪ Allows for library/object code delivery of module
Post-build time	▪ Can be changed at post-build time or link time or precompile time ▪ Configuration can be updated separately from the static module code → post-build loadable ▪ Switch between different configurations based on coding is possible → post-build selectable

Configuration Files

- Pre-compile time configuration
  - Preprocessor instructions

- Link-time configuration
  - Constant data outside the module

- Post-build time configuration
  - Loadable constant data outside the module



Note! The configuration parameters in one module could belong to different configuration classes.

Summary

Architecture

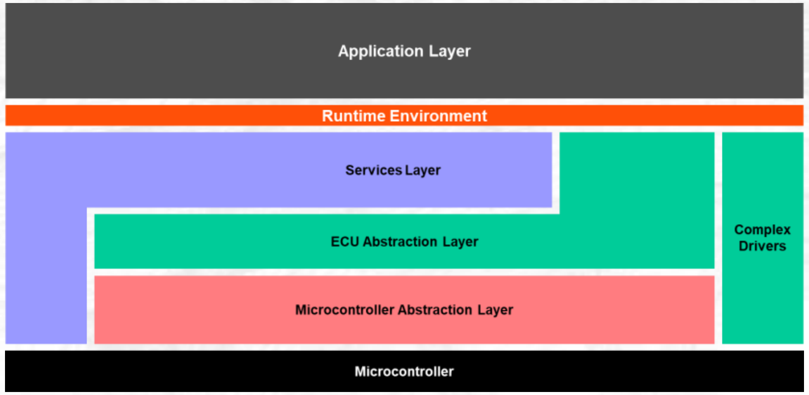
- Introduce the Layered Software Architecture and Basic software (BSW) Modules

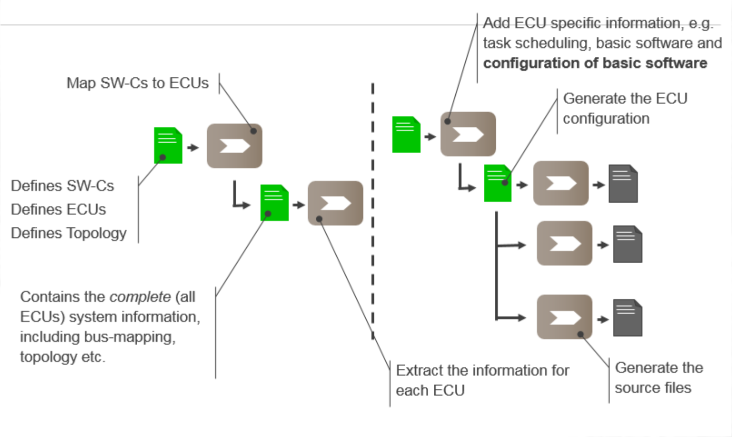
Methodology

- Understand the Difference between System- and ECU-Configuration
- Understand the Complete Chain from System Description to Executable

Migration & Integration Strategies

- Implementation Classes
- Configuration Classes





Add ECU specific information, e.g. task scheduling, basic software and **configuration of basic software**

Map SW-Cs to ECUs

Generate the ECU configuration

Defines SW-Cs
Defines ECUs
Defines Topology

Contains the *complete* (all ECUs) system information, including bus-mapping, topology etc.

Extract the information for each ECU

Generate the source files

Get in touch!

Elektrobit

sales@elektrobit.com
www.elektrobit.com