Continental
The Future in Motion

AUTOSAR Standard
Overview

"Cooperate on standards, compete on implementation"

Agenda
AUTOSAR Standard

Continental

Space for Sender Information
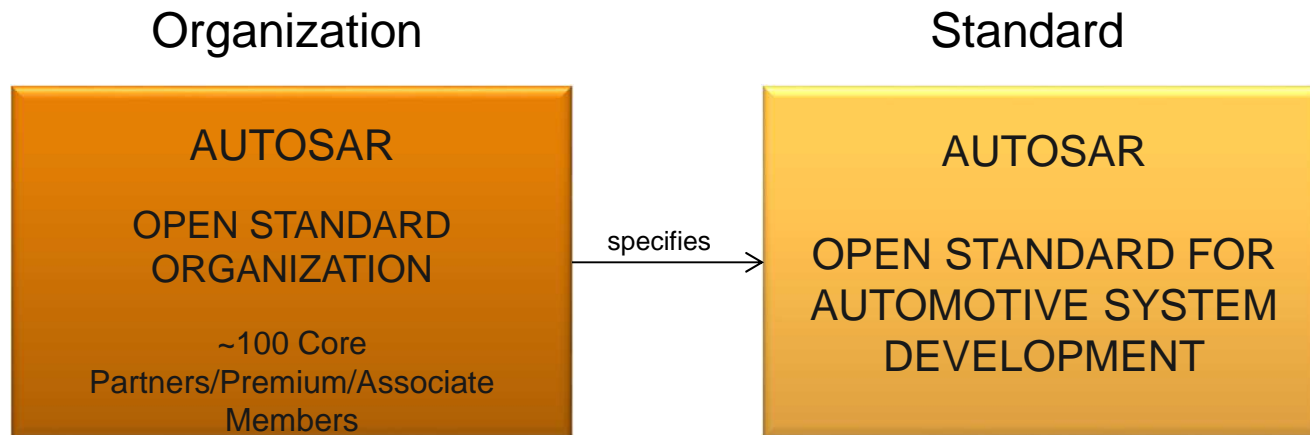Confidential

1 November 2018

Introduction to the AUTOSAR Standard

› Managing Complexity by Exchangeability and Reuse of SW Components

› AUTOSAR stands for AUTomotive Open System ARchitecture
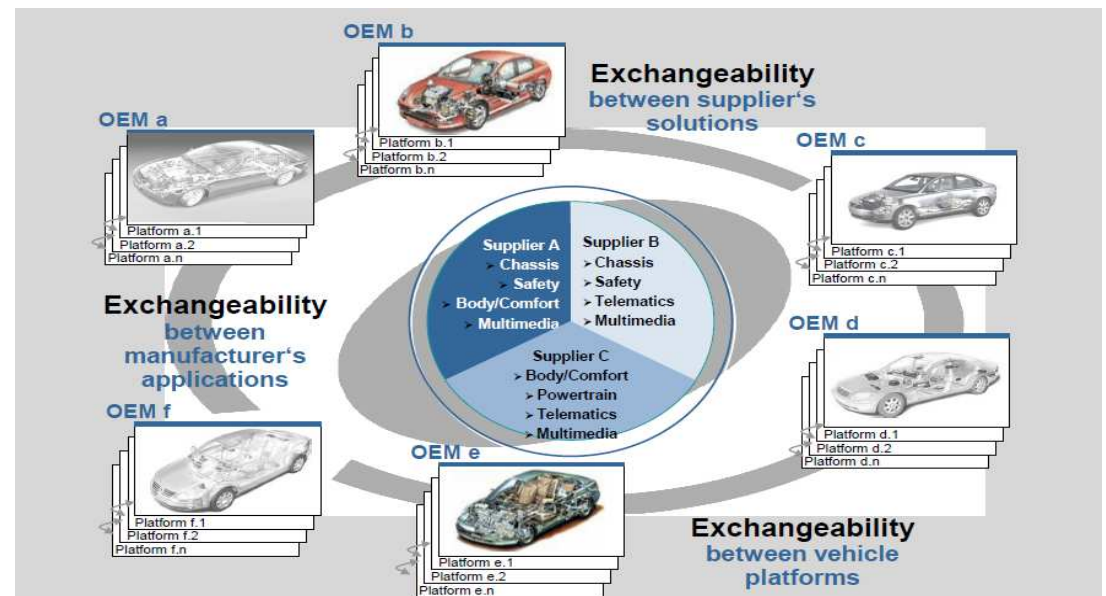
Introduction to the AUTOSAR Standard

› Organization specifies the Standard

› Official website of the Standard : http://www.autosar.org/

Organization   Standard

AUTOSAR	specifies →	AUTOSAR
OPEN STANDARD ORGANIZATION		OPEN STANDARD FOR AUTOMOTIVE SYSTEM DEVELOPMENT
~100 Core Partners/Premium/Associate Members		

Introduction to the AUTOSAR Standard

› AUTOSAR partnership structure

  › Core Partners

    › Organizational control

    › Technical contributions

    › Administrative control

  › Premium members

    › Leadership of Working Groups

    › Involvement in Working Groups

    › Technical contributions

  › Associate Members

    › Access to finalized documents

   › Utilization of AUTOSAR standard

› Partners :

  › Core Group: BMW Group, Bosch, Daimler, Ford Motor Company, General Motors, PSA Peugeot, Toyota, Volkswagen AG, Continental

Introduction to the AUTOSAR Standard

› Objectives:

  › Standardization of basic software functionality of automotive ECUs

  › Scalability to different vehicle and platform variants

  › Definition of an open architecture

  › Collaboration between various partners

  › Support of applicable automotive international standards and state-of-the-art technologies

  › Transferability of functions throughout network

  › Increased use of "Commercial off the shelf hardware"

Ontinental⑂

Introduction to the AUTOSAR Standard

› Main working topics
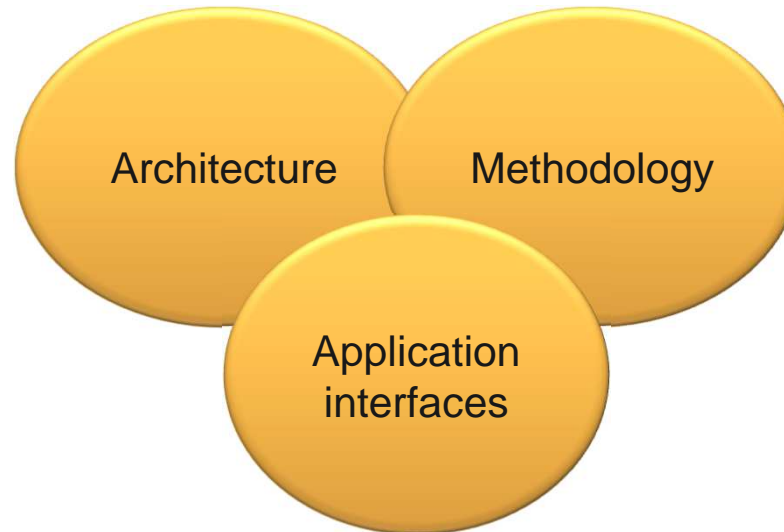
Agenda
AUTOSAR Standard

1	**Introduction to the AUTOSAR Standard**

2	**AUTOSAR - Layered Architecture**

3	**Basic Software**

4	**Runtime Environment**

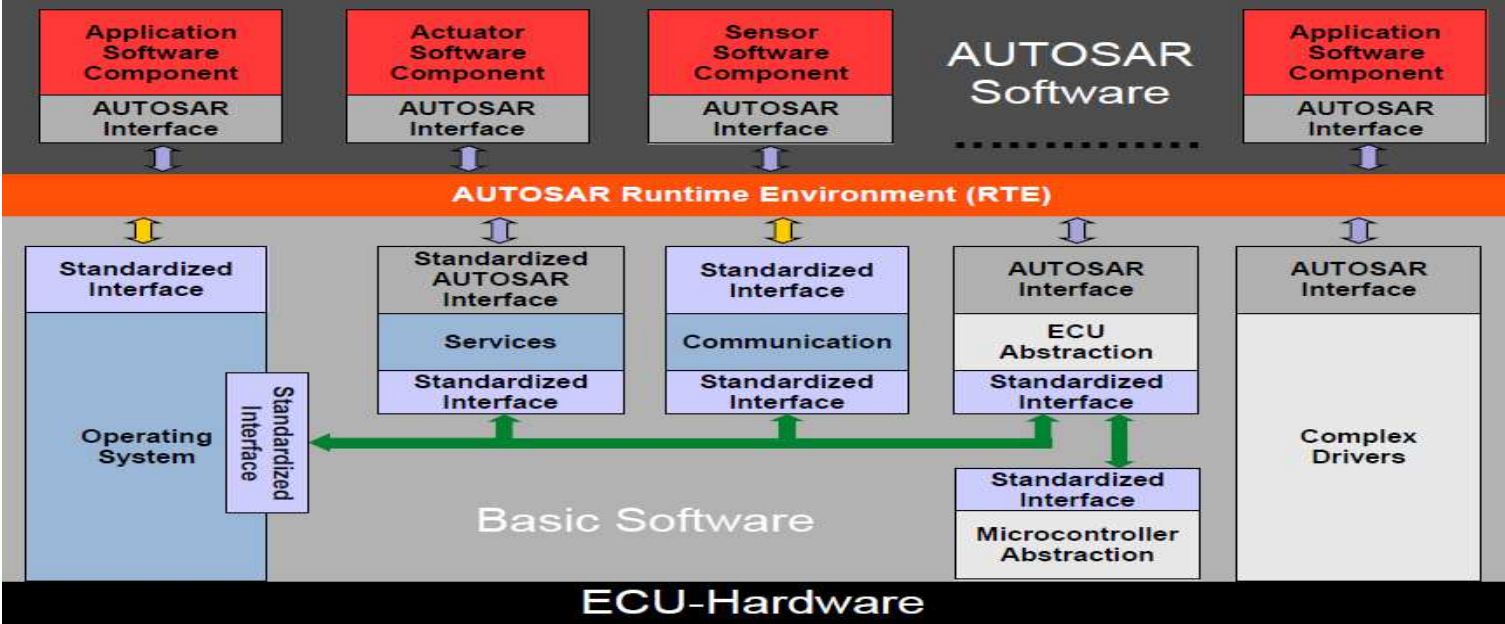|	
5	**Application - Software Components**

6	**Methodology**

AUTOSAR - Layered Architecture

› Top - down approach

› Top view : 3 software layers

  › Application Layer

  › Runtime Env.
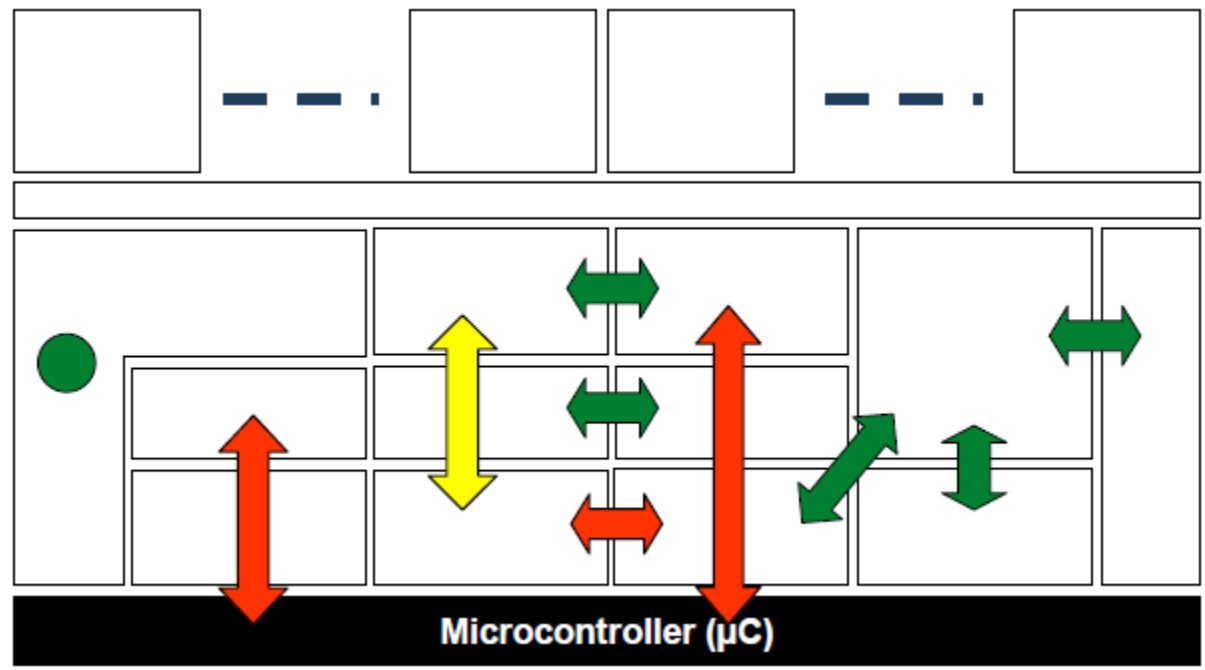
  › Basic SW
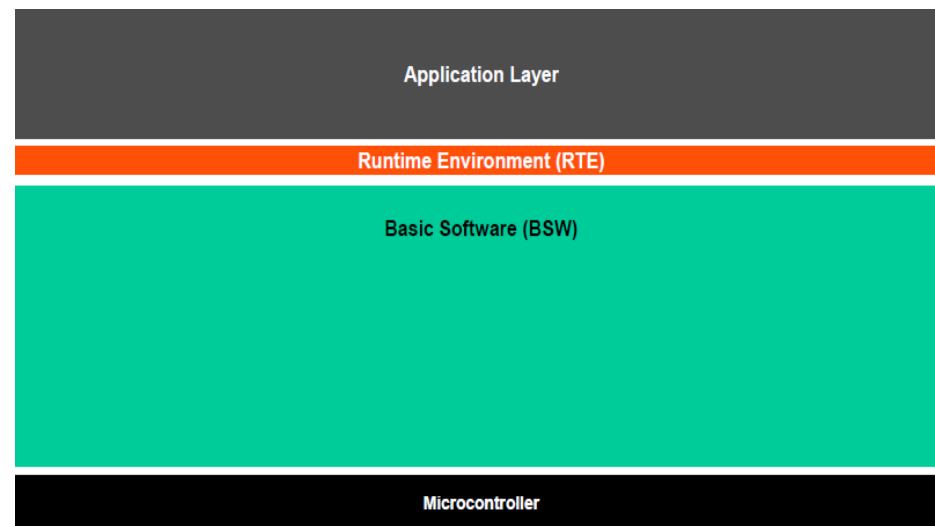
AUTOSAR - Layered Architecture

AUTOSAR - Layered Architecture

› AUTOSAR Interface

> › An "AUTOSAR Interface" defines the information exchanged between software components and/or BSW modules. This description is independent of a specific programming language, ECU or network technology.

› Standardized AUTOSAR Interface

> › A "Standardized AUTOSAR Interface" is an "AUTOSAR Interface" whose syntax and semantics are standardized in AUTOSAR. The "Standardized AUTOSAR Interfaces" are typically used to define AUTOSAR Services, which are standardized services provided by the AUTOSAR Basic Software to the application Software-Components.

› Standardized Interface

> › These "Standardized Interfaces" are typically defined for a specific programming language (like "C").They are typically used between software-modules which are always on the same ECU.

AUTOSAR - Layered Architecture

› General Interfacing Rules

 › Bypassing of two or more software layers is not allowed

 › Bypassing of the µC Abstraction Layer is not allowed

 › All layers may interact with system services ●

 › Bypassing of one software layer should be avoided

 › A complex driver may use selected other BSW modules

Microcontroller (µC)

AUTOSAR - Layered Architecture

› Advantages of the architecture

  › AUTOSAR provides an abstraction from Hardware

  › Supports the reallocation of Software Components to different ECUs

  › Provides standardized software interface for communication between Software Components

  › Provides network independent communication mechanisms for applications

  › Supports the principle of information hiding > reduce impact of modifications

› An information model is defined for the AUTOSAR Architecture (MetaModel – UML based)

  › Available MetaModels : V2.0, V3.0 , V3.1, V3.2, V4.0, V4.1, V4.2

AUTOSAR - Layered Architecture

› The AUTOSAR Timeline
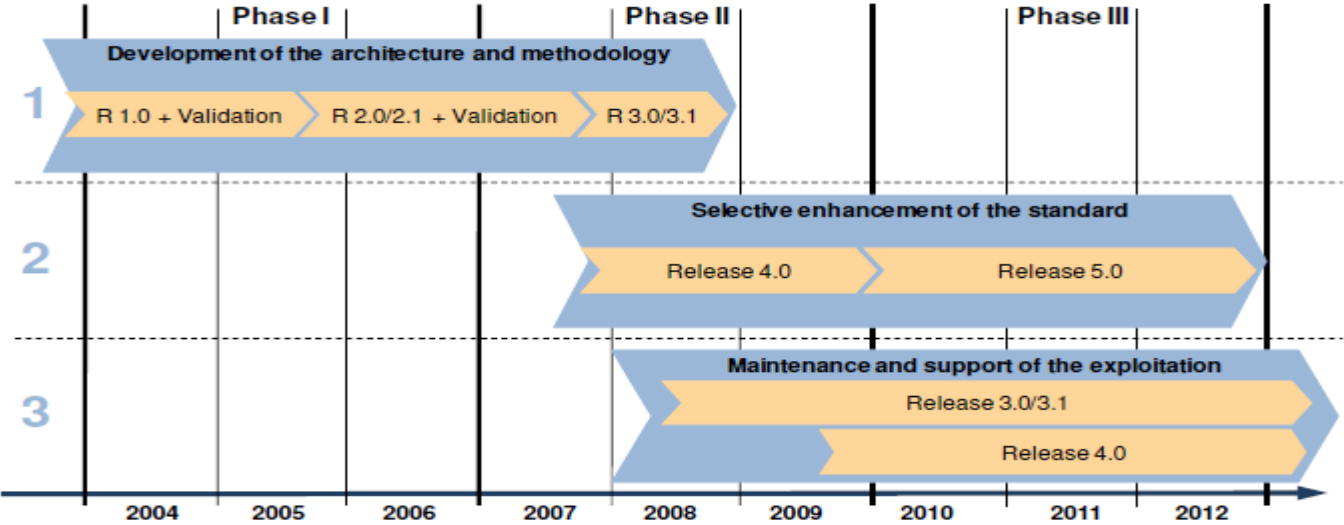
AUTOSAR - Layered Architecture

› **Release 4.0 – Summary of Changes**

› Functional Safety - main objective as AUTOSAR will support safety related applications and thereby has to consider the ISO 26262 standard

   › Memory Partitioning Concept

   › Time Determinism Concept

   › Program Flow Monitoring Concept

   › SW-C E2E Comm protection Concept

   › BSWM Defensive Behavior Concept (prevents data corruption and wrong service calls)

   › Dual Microcontroller Concept

AUTOSAR - Layered Architecture

› **Release 4.0 – Summary of Changes**

› Architectural improvement

   › Error Handling Concept

   › Multi Core Architectures Concept

   › Bootloader Interaction Concept

   › Build System Enhancement Concept

   › Memory Related Concept

   › Support of Windowed Watchdog Concept

   › Enabling CDDs in the BSW Architecture Concept

Continental

AUTOSAR - Layered Architecture

› **AUTOSAR extensibility**

› The AUTOSAR Software Architecture is a generic approach:

  › standard modules can be extended in functionality, while still being compliant, still, their configuration has to be considered in the automatic Basic SW configuration process!

  › non-standard modules can be integrated into AUTOSAR-based systems as Complex Drivers and

› further layers cannot be added.

Agenda
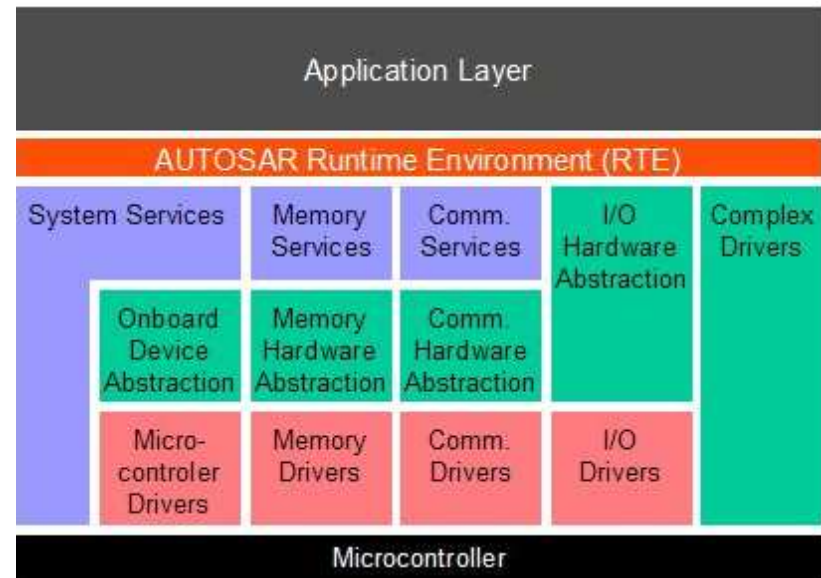## AUTOSAR Standard

Ontinental

Basic Software

› Software layers

  › Application Layer
  › Runtime Enviroment.

  › Basic SW

    › MCAL

    › ECU Abstraction

    › Complex Drivers

    › Services
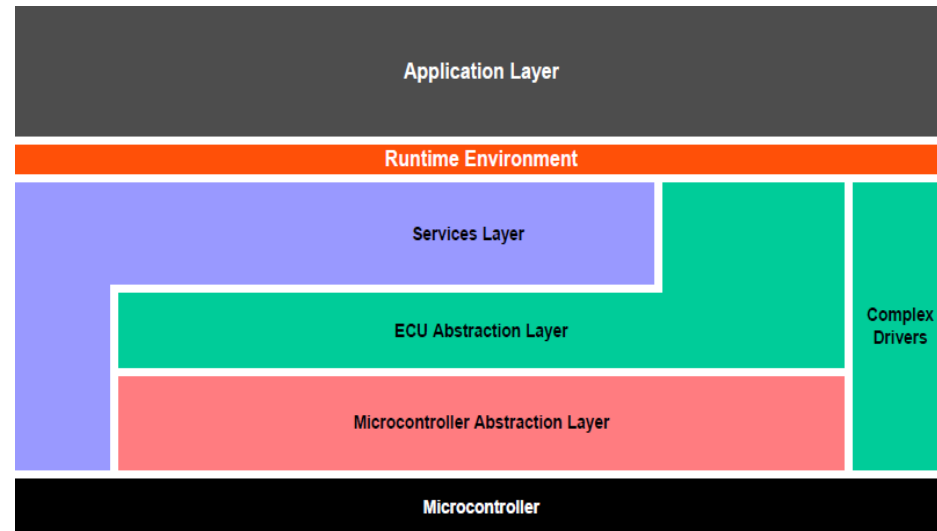
Basic Software

› BSW

  › System Stack

 › Memory Stack

  › Communication Stack

  › IO HW Abstraction

BSW – Types of Services

› The Basic Software can be subdivided into the following types of services:

› **Input/Output** (I/O)

  › Standardized access to sensors, actuators and ECU onboard peripherals

› **Memory**

  › Standardized access to internal/external memory (non volatile memory)

› **Communication**

  › Standardized access to: vehicle network systems, ECU onboard communication systems and ECU internal SW

› **System**

  › Provision of standardizeable (operating system, timers, error memory) and ECU specific (ECU state management, watchdog manager) services and library functions
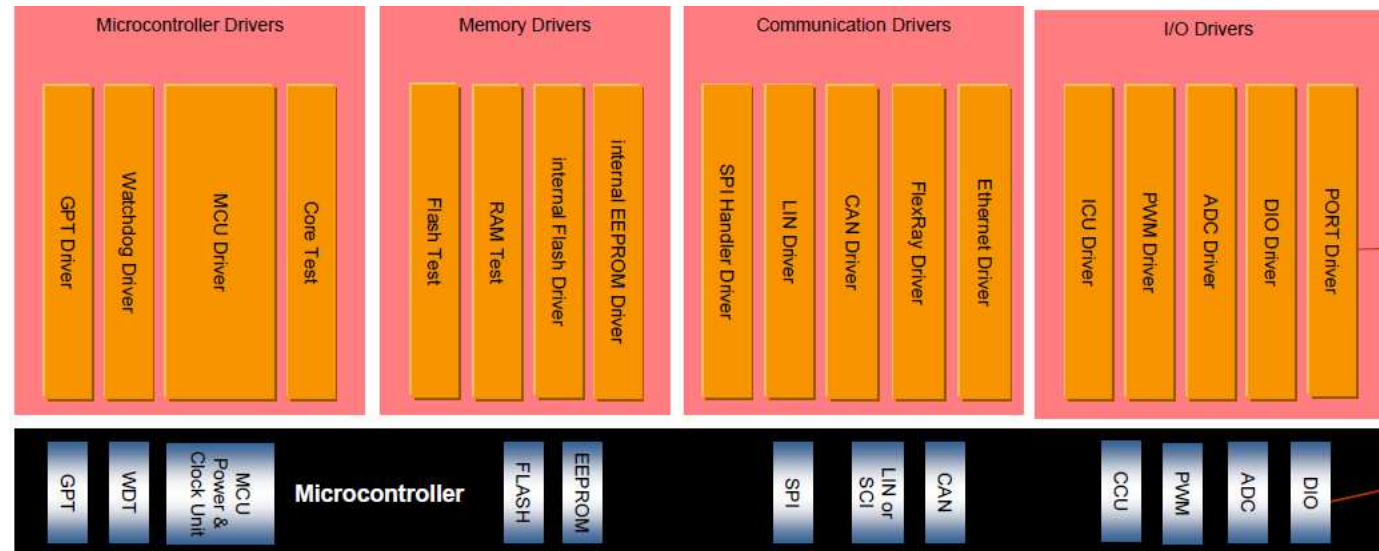
BSW – Microcontroller Abstraction Layer (MCAL)

› Contains internal drivers, with direct access to the μC and internal peripherals

› Task : make higher layers independent of μC

› Properties:

 › μC dependent

 › Upper interface : standardized

   and μC independent
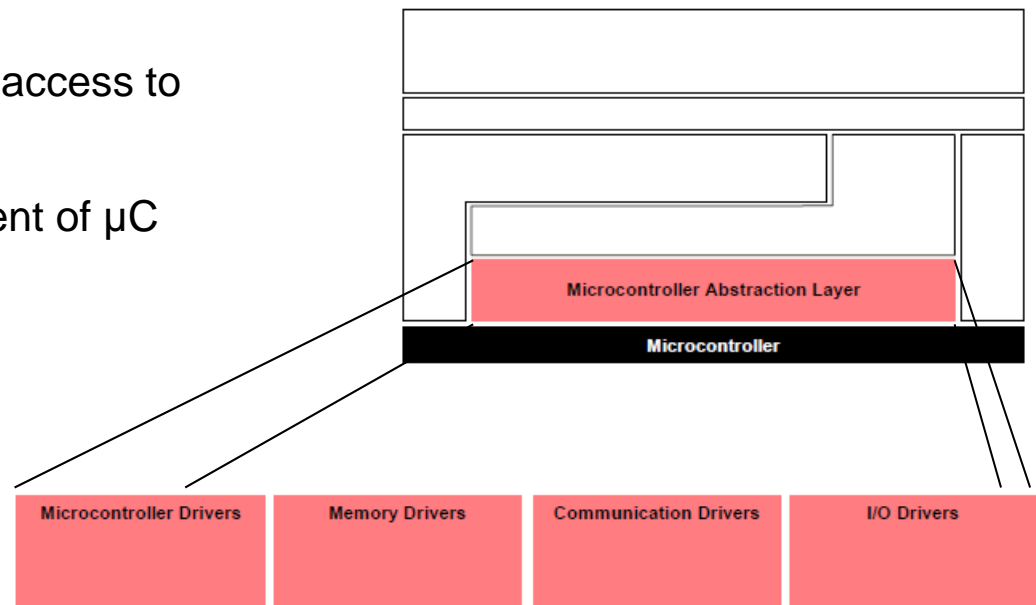
# BSW – Microcontroller Abstraction Layer (MCAL)

› Communication Drivers

› I/O Drivers

› Memory Drivers

› Microcontroller Drivers

Microcontroller Drivers

GPT Driver | Watchdog Driver | MCU Driver | Core Test

Memory Drivers

Flash Test | RAM Test | internal Flash Driver | internal EEPROM Driver

Communication Drivers

SPI Handler Driver | LIN Driver | CAN Driver | FlexRay Driver | Ethernet Driver

I/O Drivers

ICU Driver | PWM Driver | ADC Driver | DIO Driver | PORT Driver

GPT | WDT | MCU Power & Clock Unit | **Microcontroller** | FLASH | EEPROM | SPI | LIN or SCI | CAN | CCU | PWM | ADC | DIO
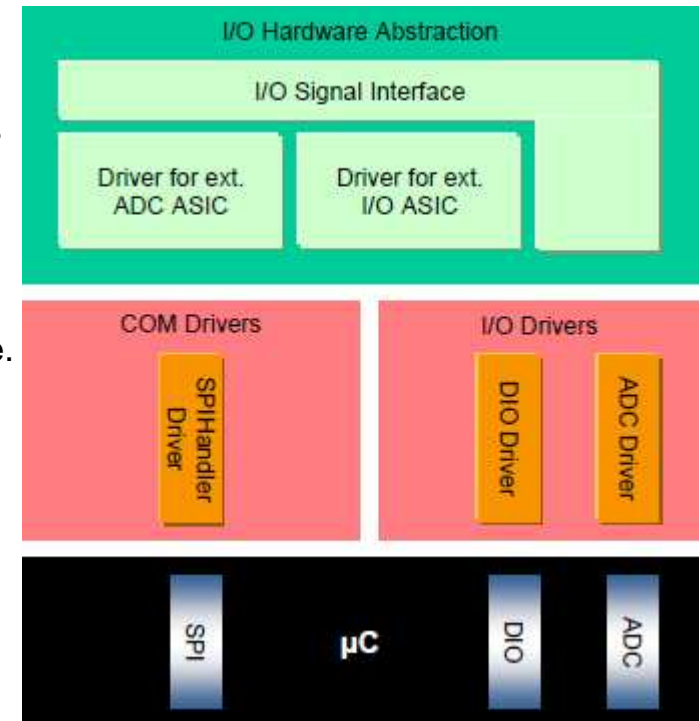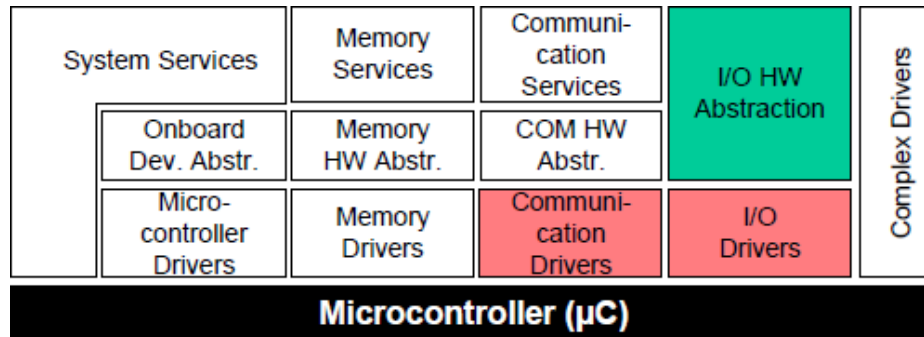
BSW – ECU Abstraction Layer

› Offers an API for access to peripherals and devices regardless of their location (µC internal/external) and their connection to the µC

› Task: make higher layers independent of ECU hardware layout

› Properties:

  › µC independent, ECU hardware dependent
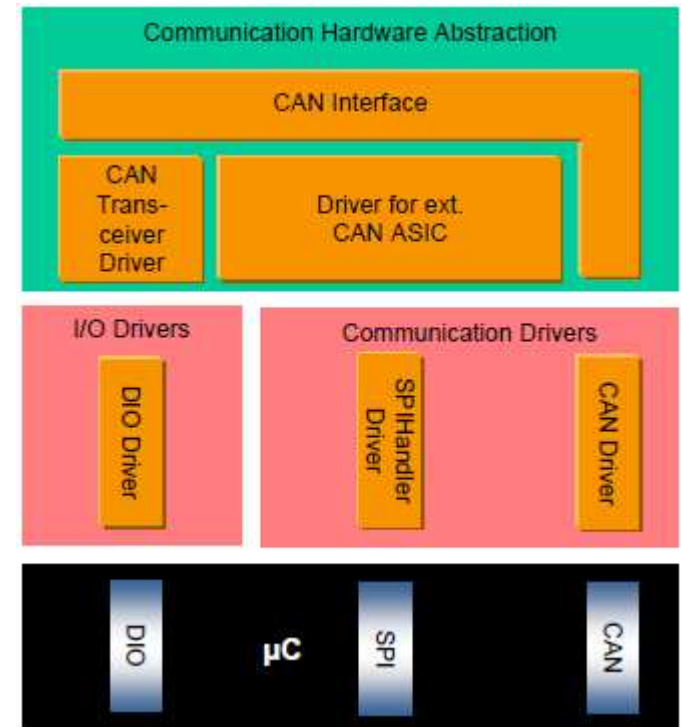
  › Upper interface: µC and ECU

       independent
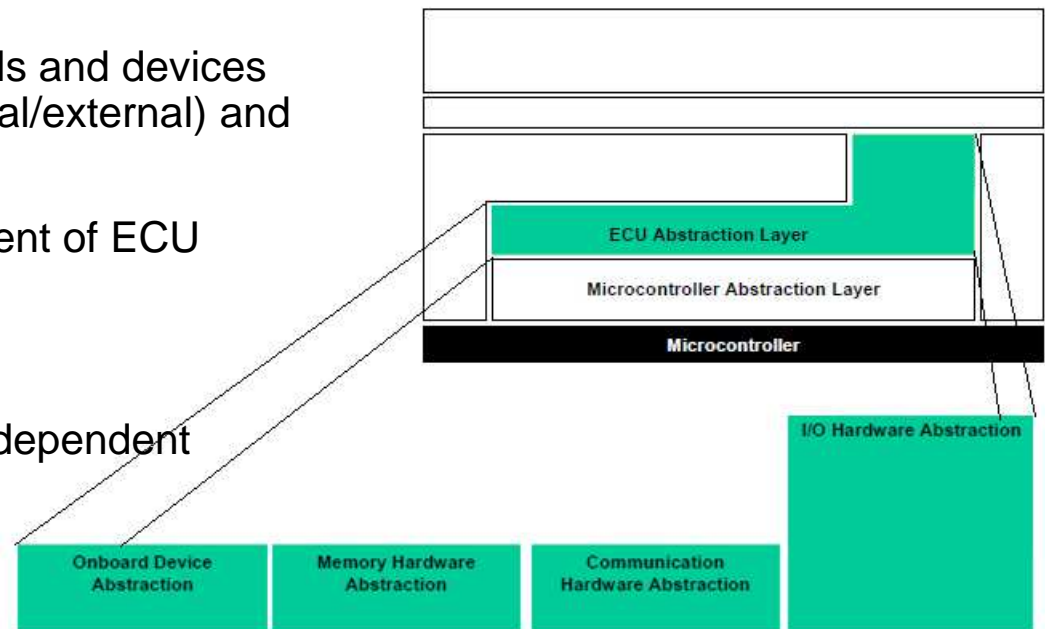
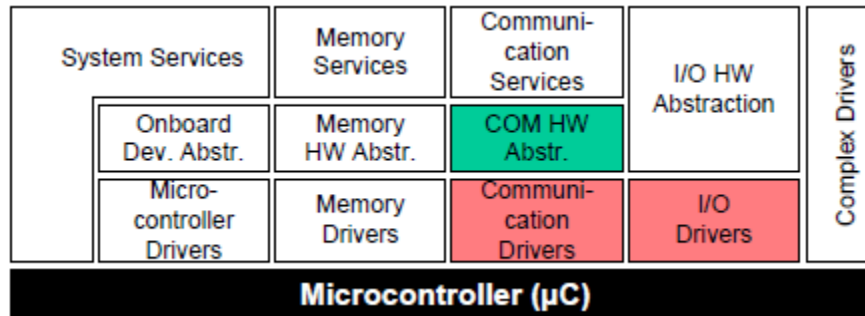# BSW – ECU Abstraction Layer

› The **I/O Hardware Abstraction** is a group of modules which 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). The I/O Hardware Abstraction does not abstract from the sensors/actuators!

› The different I/O devices might be accessed via an I/O signal interface.

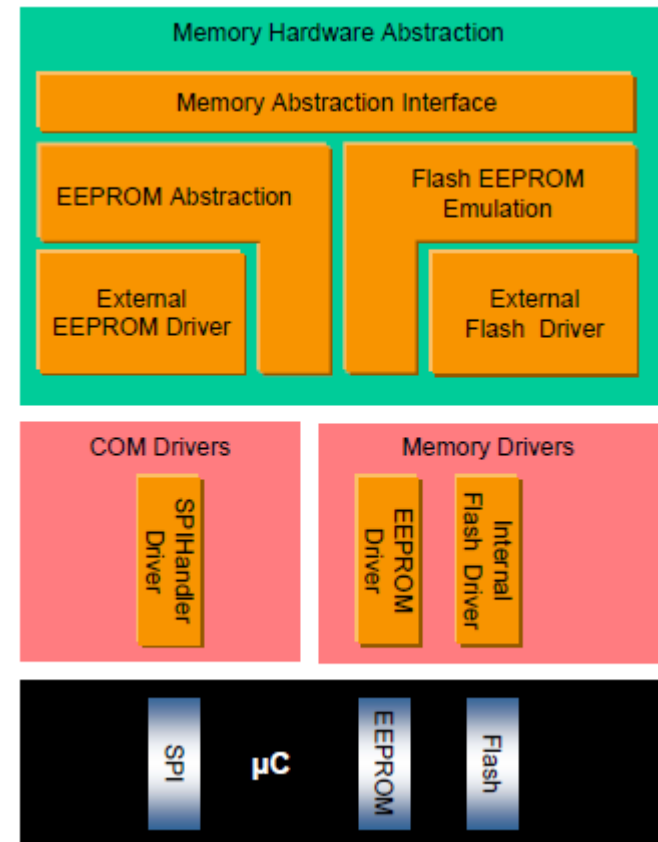# BSW – ECU Abstraction Layer

› The **Communication Hardware Abstraction** is a group of modules which abstracts from the location of communication controllers and the ECU hardware layout. For all communication systems a specific Communication Hardware Abstraction is required (e.g. for LIN, CAN, FlexRay).

› Provides equal mechanisms to access a bus channel regardless of it's location (on-chip / on-board)

BSW – ECU Abstraction Layer

› The **Memory Hardware Abstraction** is a group of modules which abstracts from the location of peripheral memory devices (on-chip or on-board) and the ECU hardware layout.

› The memory drivers are accessed via memory specific abstraction/emulation modules (e.g. EEPROM Abstraction).
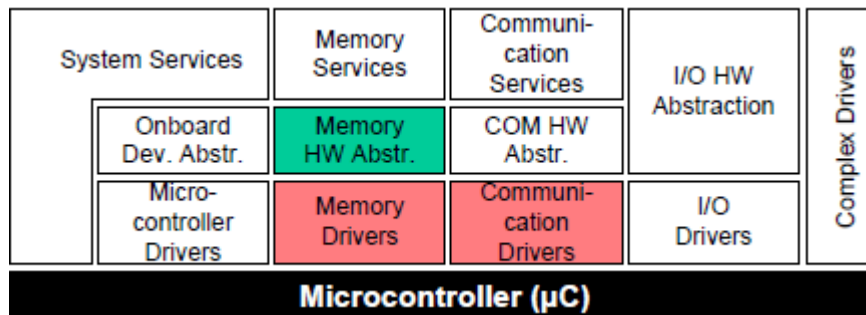
BSW – ECU Abstraction Layer

› The **Onboard Device Abstraction** contains drivers for ECU onboard devices which cannot be seen as sensors or actuators like internal or external watchdogs. Those drivers access the ECU onboard devices via the µC Abstraction Layer.
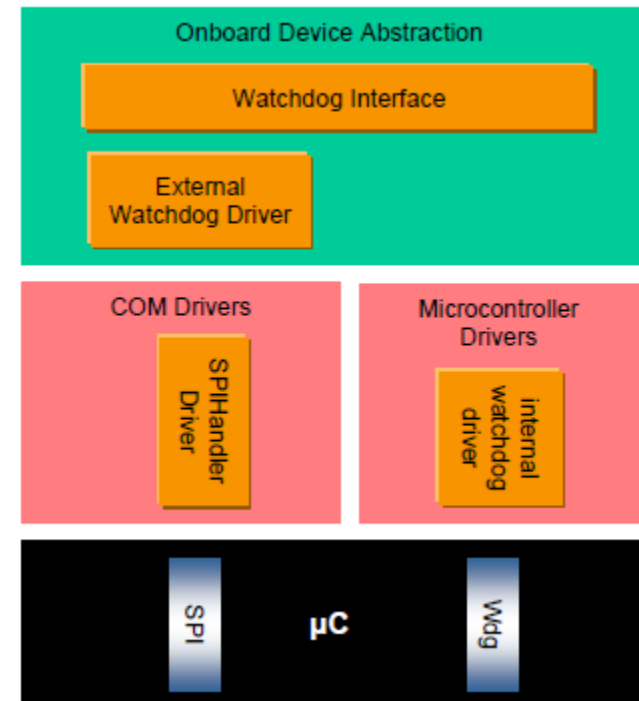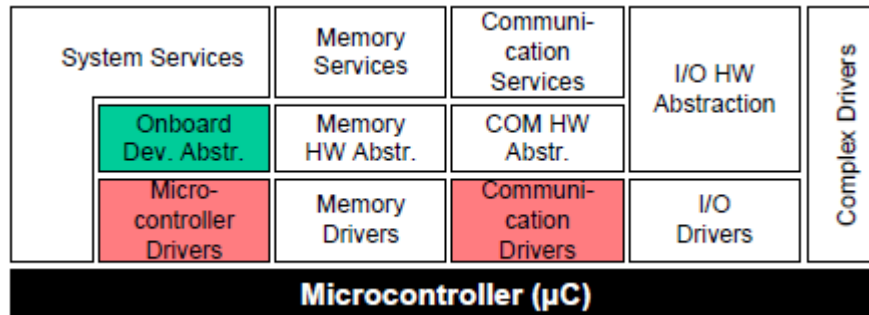
BSW – Complex Drivers

› Spans from the hardware to the RTE

› Task: provides the possibility to integrate special purpose functionality:

  › Which are not specified within AUTOSAR

  › With high timing constraints, etc

› Properties :

  › Might be application, µC, ECU hw dependent

  › Upper interface: Might be application, µC, ECU hw dependent

ECU Abstraction Layer

Microcontroller Abstraction Layer

Complex Drivers

Microcontroller
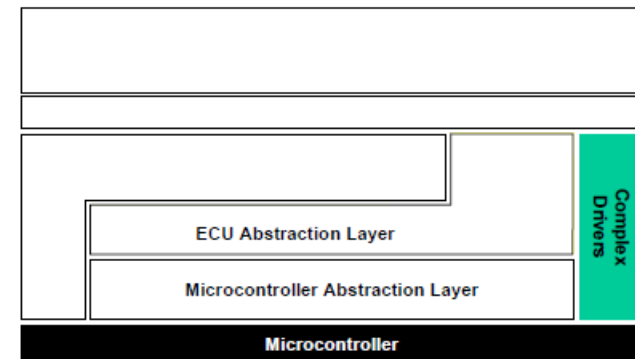
BSW – Services Layer

› Offers:

    › Operating system functionality

    › Vehicle network communication and management services

    › Memory services and management

    › Diagnostic services

    › ECU state management, mode management

› Task: provide basic services for applications and basic software modules

› Properties:

    › µC and ECU hardware independent
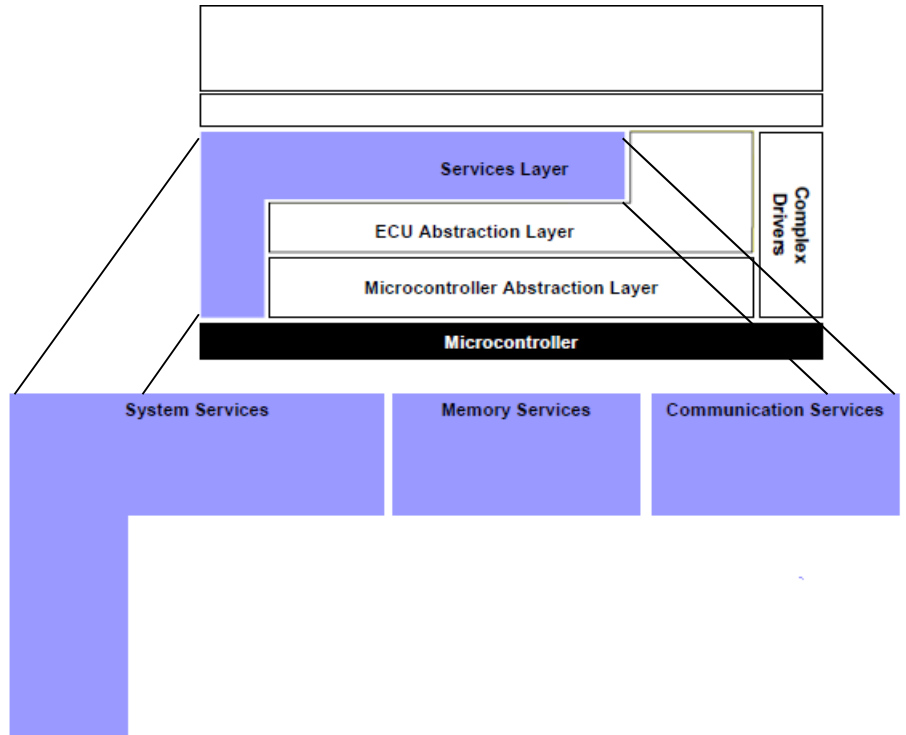
BSW – Services Layer

› The **Communication Services** are a group of modules for vehicle network communication (CAN, LIN and FlexRay). They interface with the communication drivers via the communication hardware abstraction.

› Provide uniform interfaces to the vehicle network for communication.

› Provide uniform services for network management

› Provide uniform interface to the vehicle network for diagnostic communication

› Hide protocol and message properties from the application.
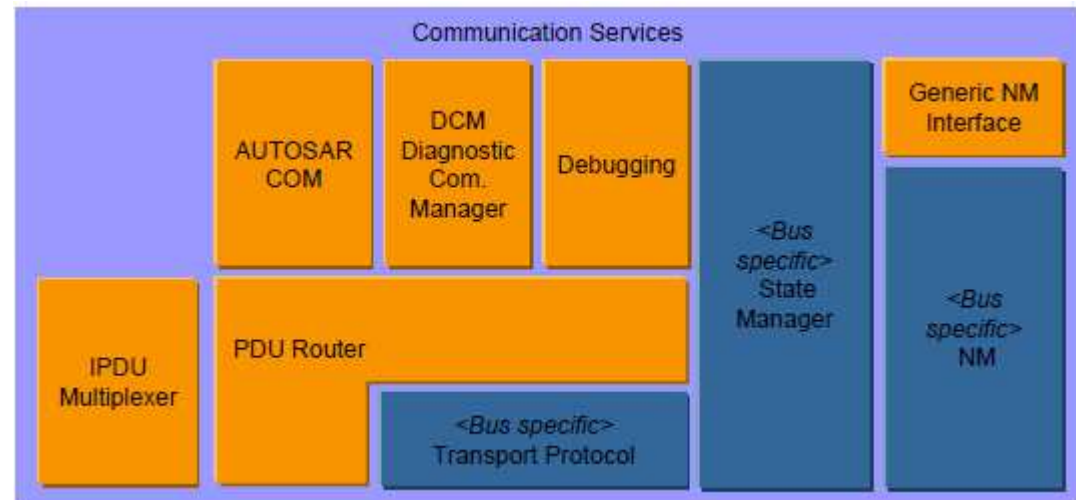
BSW – Services Layer

› The **Memory Services** consist of one module, the NVRAM Manager. It is responsible for the management of non volatile data (read/write from different memory drivers).

› Provide non volatile data to the application in a uniform way. Abstract from memory locations and properties. Provide mechanisms for non volatile data management like saving, loading, checksum protection and verification, reliable storage
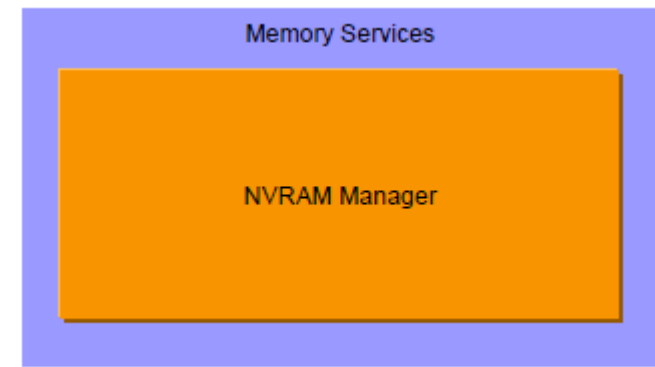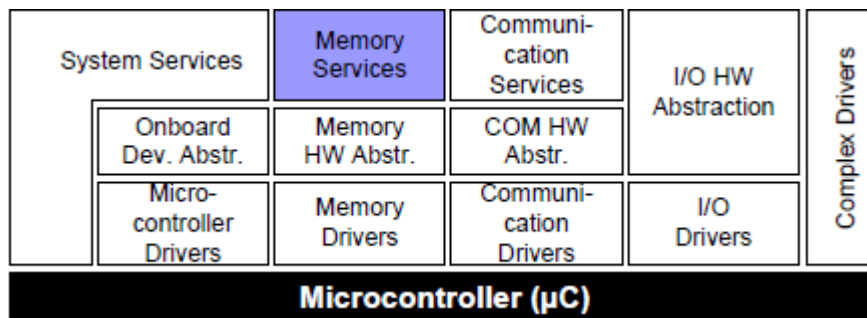
BSW – Services Layer

› The **System Services** are a group of modules and functions which can be used by modules of all layers. Examples are Real Time Operating System (which includes timer services) and Error Manager.

› Some of these services are µC dependent (like OS), partly ECU hardware and application dependent (like ECU State Manager) or hardware and µC independent.

System Services

AUTOSAR OS

ECU State Manager (EcuM)

Communication Manager (ComM)

Diagnostic Event Manager (Dem)

Function Inhibition Manager (FIM)

Development Error Tracer (Det)

Diagnostic Log and Trace (Dlt)

Synchronized Time-base Manager (StbM)

Watchdog Manager (WdgM)

Basic Software Mode Manager (BswM)
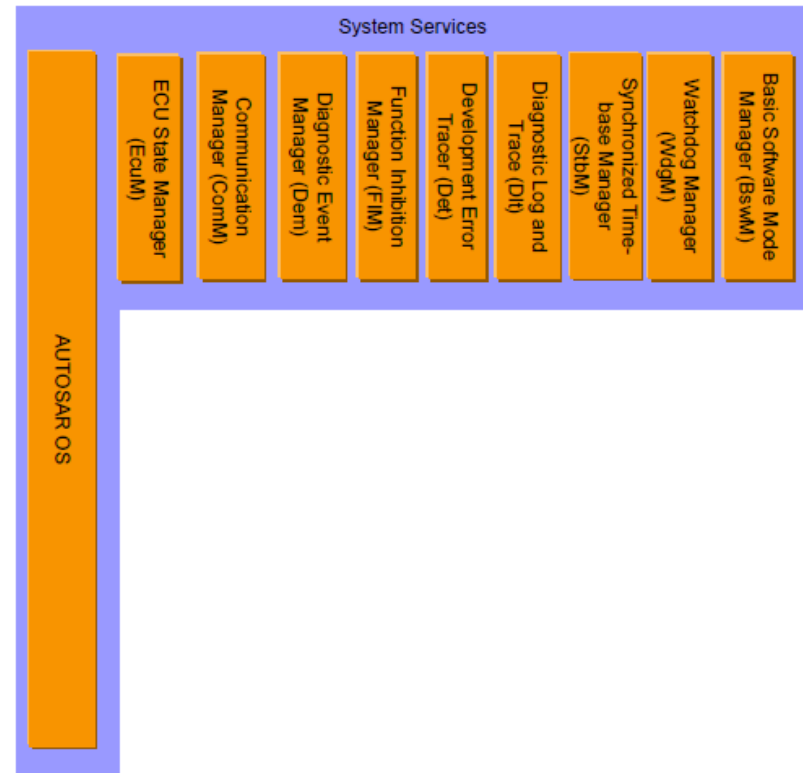
Configuration

› The AUTOSAR Basic Software supports the following configuration classes:

  › **Pre-compile time**

    › Preprocessor instructions

    › Code generation (selection or synthetization)

  › **Link time**

    › Constant data outside the module; the data can be configured after the module has been compiled

  › **Post-build time**

    › Loadable constant data outside the module. Very similar to link time, but the data is located in a specific memory segment that allows reloading (e.g. reflashing in ECU production line)

› Single or multiple configuration sets can be provided. In case that multiple configuration sets are provided, the actually used configuration set is to be specified at runtime

Configuration – Pre-compile time

› **Use cases**

  › Enabling/disabling optional functionality This allows to exclude parts of the source code that are not needed

  › Optimization of performance and code size Using #defines results in most cases in more efficient code than access to constants or even access to constants via pointers. Generated code avoids code and runtime overhead.

› **Restrictions**

  › The module must be available as source code

  › The configuration is static. To change the configuration, the module has to be recompiled

› **Required implementation**

  › Pre-compile time configuration shall be done via the module's two configuration files (*_Cfg.h, *_Cfg.c) and/or by code generation

Configuration – Link time

› **Use cases**

  › Configuration of modules that are only available as object code (e.g. IP protection or warranty reasons)

  › Selection of configuration set after compilation but before linking.

› **Required implementation**

  › One configuration set, no runtime selection Configuration data shall be captured in external constants. These external constants are located in a separate file. The module has direct access to these external constants.

Configuration – Post-build time

› **Use cases**

  › Configuration of data where only the structure is defined but the contents not known during ECU-build time

  › Configuration of data that is likely to change or has to be adapted after ECU-build time (e.g. end of line, during test & calibration)

  › Reusability of ECUs across different product lines (same code, different configuration data)

› **Restrictions**

  › Implementation requires dereferencing which has impact on performance, code and data size

› **Required implementation**

  › One configuration set, no runtime selection (**loadable) .** Configuration data shall be captured in external constant structs. These external structs are located in a separate memory segment that can be individually reloaded.

Agenda
AUTOSAR Standard

** Continental**

Runtime Environment (RTE)

› Provides communication services to the application software

› Makes the mapping of the AUTOSAR interfaces between the Application and BSW

› Task :   make AUTOSAR SWC independent from the mapping to a specific ECU

› Properties:

   › ECU and application specific

   › Upper interface: completely ECU independent

AUTOSAR Runtime Environment (RTE)

Services Layer

ECU Abstraction Layer

Microcontroller Abstraction Layer

Complex Drivers

Microcontroller
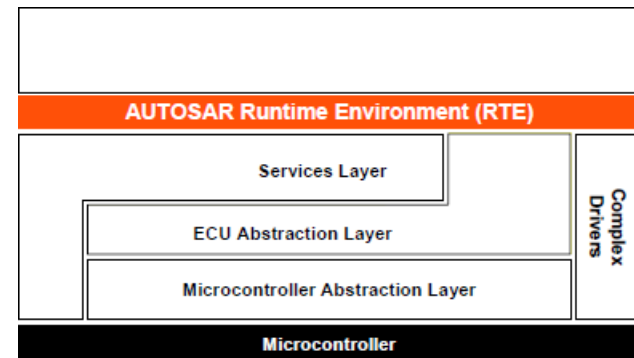
Runtime Environment (RTE)

› The AUTOSAR Runtime Environment (RTE) acts as a system level communication center for inter- and intra-ECU information exchange.

› Provide a uniform environment to AUTOSAR Software Components to make the implementation of the software components independent from communication mechanisms and channels.

› The RTE achieves this by mapping the communication relationships between components, that are specified in the different templates, to a specific intra-ECU communication mechanism, such as a function call, or an inter-ECU communication mechanism, such as a COM message which leads to CAN communication.

Runtime Environment (RTE)

› The implementation of an AUTOSAR Software Component is not allowed to use the communication layer, for example OSEK COM, directly. To communicate with other software components it uses ports and client-server communication or sender-receiver communication.

› Firstly, the access to services, to the ECU abstraction, or to Complex Device Drivers is abstracted via ports and AUTOSAR interfaces. With respect to the component implementation, the RTE provides appropriately generated APIs for Basic Software access.

› The RTE is responsible for the lifecycle management of AUTOSAR Software Components. It has to invoke startup and shutdown functions of the software component.

Cntinental⅄

Runtime Environment (RTE)

› Integration aspects

    › Mapping of runnables

    › Partitioning

        › used as error containment regions, terminated or restarted during run-time as a result of a detected error

    › Scheduling

        › Single BSW modules do not know about : ECU wide timing dependencies,Scheduling implicationsMost efficient way to implement data consistency

        › Restrict the usage of OS functionality : Only the BSW Scheduler and the RTE shall use OS objects or OS services (exceptions: EcuM, Complex Drivers and services: GetCounterValue and GetElapsedCounterValue of OS; MCAL modules may enable/disable interrupts)

Ⓒntinental

Runtime Environment (RTE)

› Integration aspects

   › Processing of Mode Requests

      › The basic idea of vehicle mode management is to distribute and arbitrate mode requests and to control the BSW locally based on the results.

      › This implies that on each ECU, there has to be a mode manager that switches the modes for its local mode users and controls the BSW. Of course there can also be multiple mode managers that switch different Modes.

      › The mode request is a "normal" sender/receiver communication (system wide) while the mode switch always a local service.

   › Safety End To End Communication Protection Logic

      › For each RTE Write or Read function that transmits safety-related data (like Rte_Write_<p>_<o>()), there is the corresponding E2E protection wrapper function. The wrapper function invokes AUTOSAR E2E Library. The function is a part of Software Component and is preferably generated.

Agenda
AUTOSAR Standard

1   **Introduction to the AUTOSAR Standard**
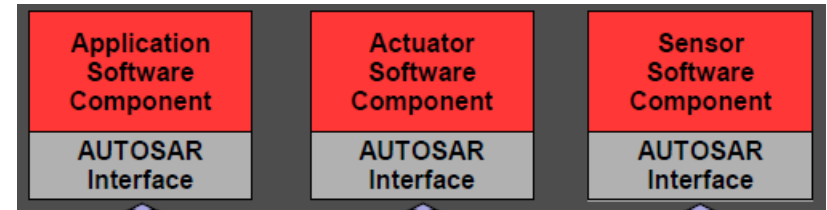
2   **AUTOSAR - Layered Architecture**

3   **Basic Software**

4   **Runtime Environment**

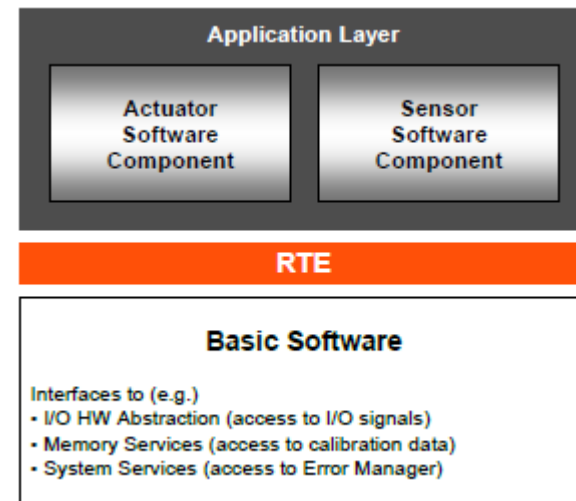**5**   **Application - Software Components**

6   **Methodology**

Software Components (SWC)

› A SWC is an encapsulated part of software realizing a particular functionality

› SW-components communicate via ports only

› Activity within SWC by means of runnables which are mapped to the OS tasks.

Software Components (SWC)

› The **Sensor/Actuator AUTOSAR Software Component** is a specific type of AUTOSAR Software Component for sensor evaluation and actuator control.

› It has been decided to locate the Sensor/Actuator SW Components above the RTE for integration reasons (standardized interface implementation and interface description). Because of their strong interaction with raw local signals, relocatability is restricted.

› Task: Provide an abstraction from the specific physical properties of hardware sensors and actuators, which are connected to an ECU.

Agenda
AUTOSAR Standard

1	**Introduction to the AUTOSAR Standard**

2	**AUTOSAR - Layered Architecture**

3	**Basic Software**

4	**Runtime Environment**

5	**Application – Software Components**

6	**Methodology**

Ⓒntinental⅄
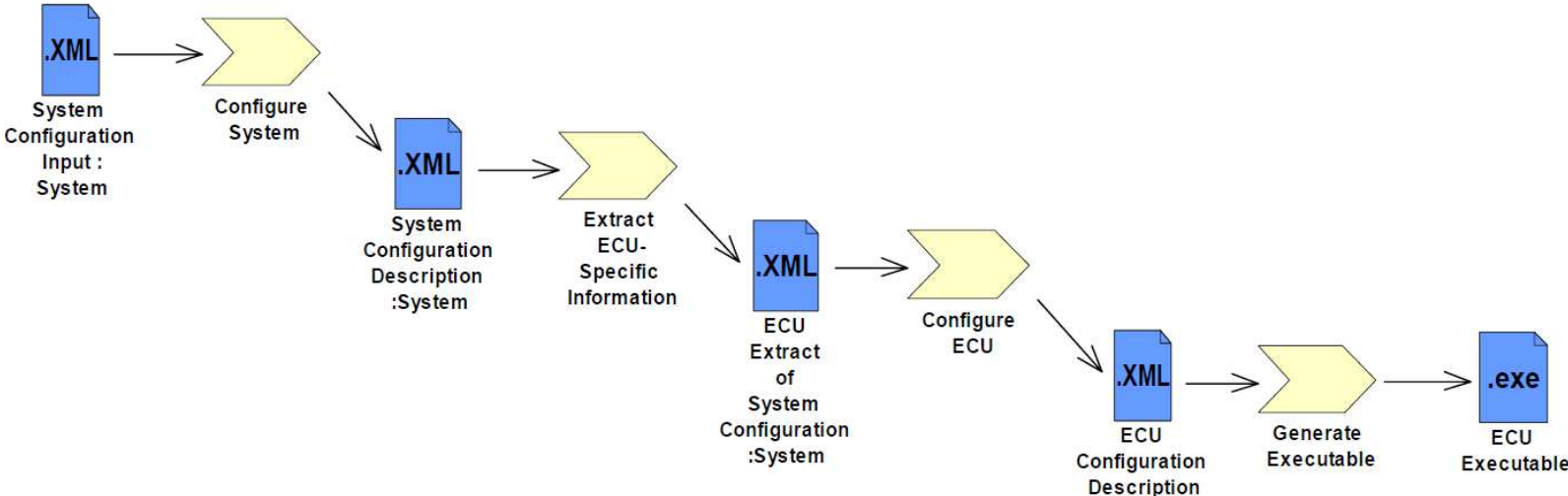
Methodology

Overview AUTOSAR Methodology

Methodology

› **AUTOSAR XML** – an XML (Extensible Markup Language) data exchanged format that is supported by the AUTOSAR standard/tools.

› **System Configuration Input** – system design (the software components and the hardware have to be selected and overall system constraints have to be identified)

› **System Configuration Description** – includes all system information (bus mapping, topology and the mapping of which SW component is located on which ECU)

› **ECU Extract of System Configuration** – it is the system configuration description for one ECU

› **ECU Configuration Description** – contains all the necessary information for implementation like task scheduling, configuration of basic software
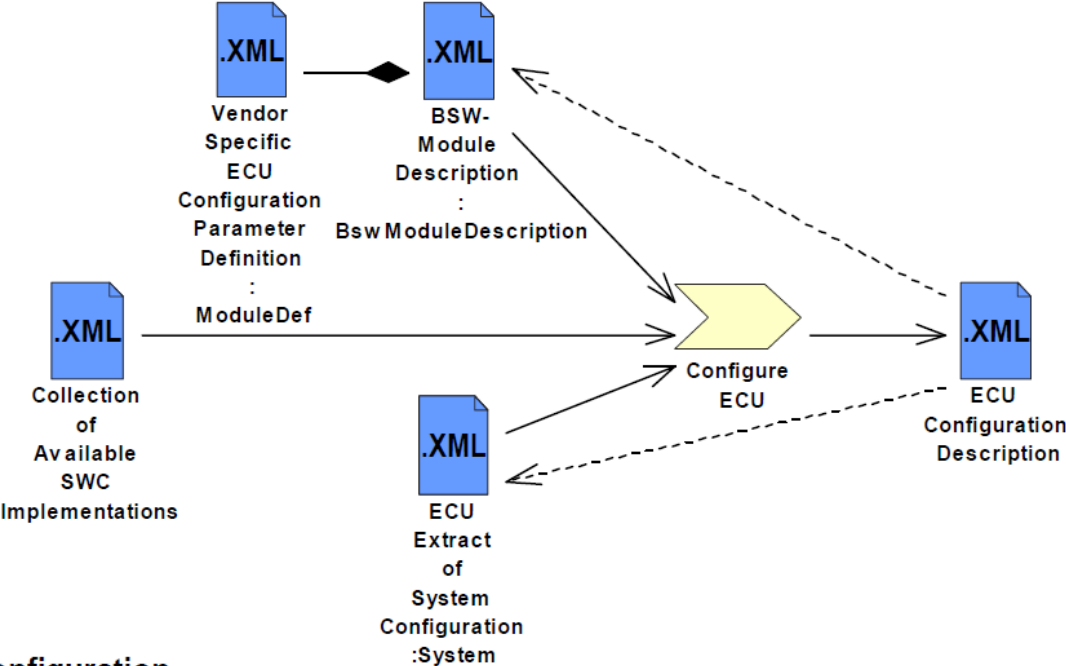
Methodology
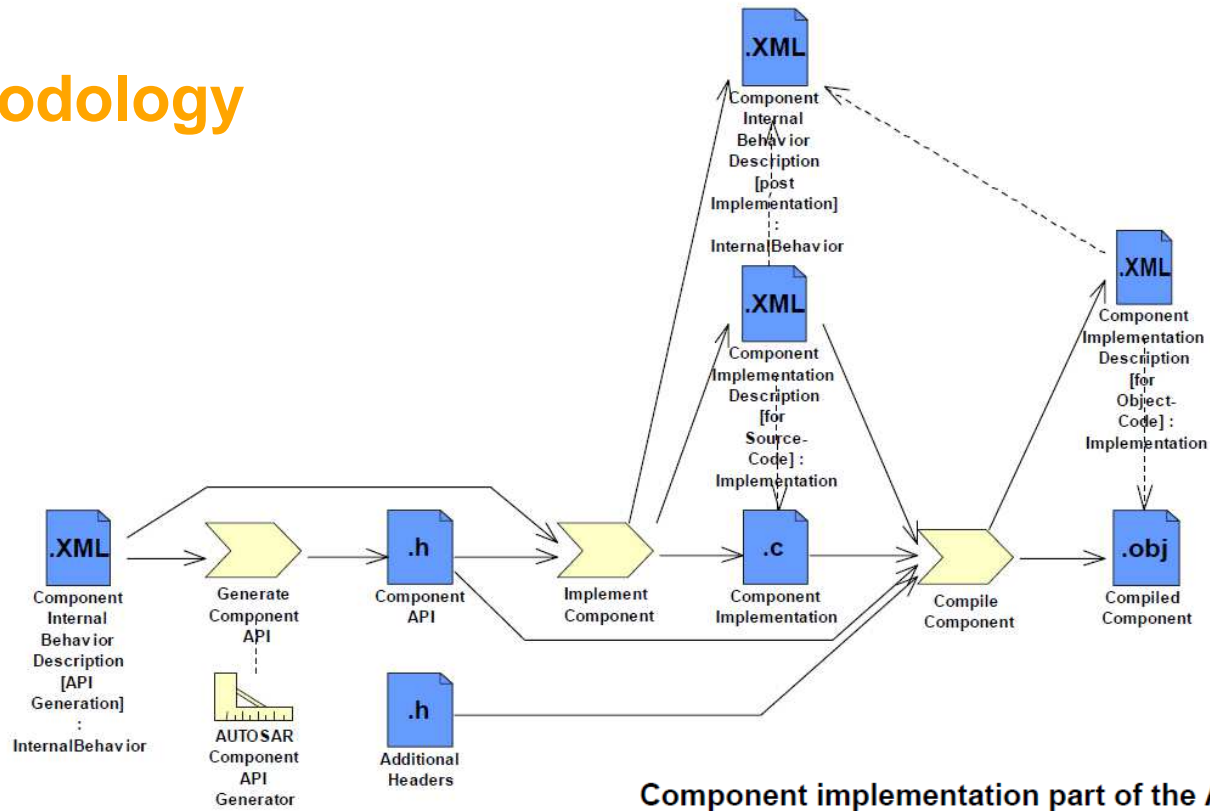
Component implementation part of the AUTOSAR Methodology

Methodology

› **Software Component (SW-C)** - is an encapsulated part of software realizing a particular functionality, communicates via ports only

› **Component Internal Behavior Description** – describes the scheduling relevant aspects of a component

› **Component API** – contains all header declarations for the RTE communication

Methodology

Overview about ECU configuration

Methodology

› **BSW Module Description** - *.autosar

› **Vendor Specific ECU Configuration Parameter Definition** – defines all possible configuration parameters and their structure

› Configure ECU – the detailed scheduling information or the configuration data for the communication module, the operating system or the AUTOSAR services have to be defines at this point.

Methodology
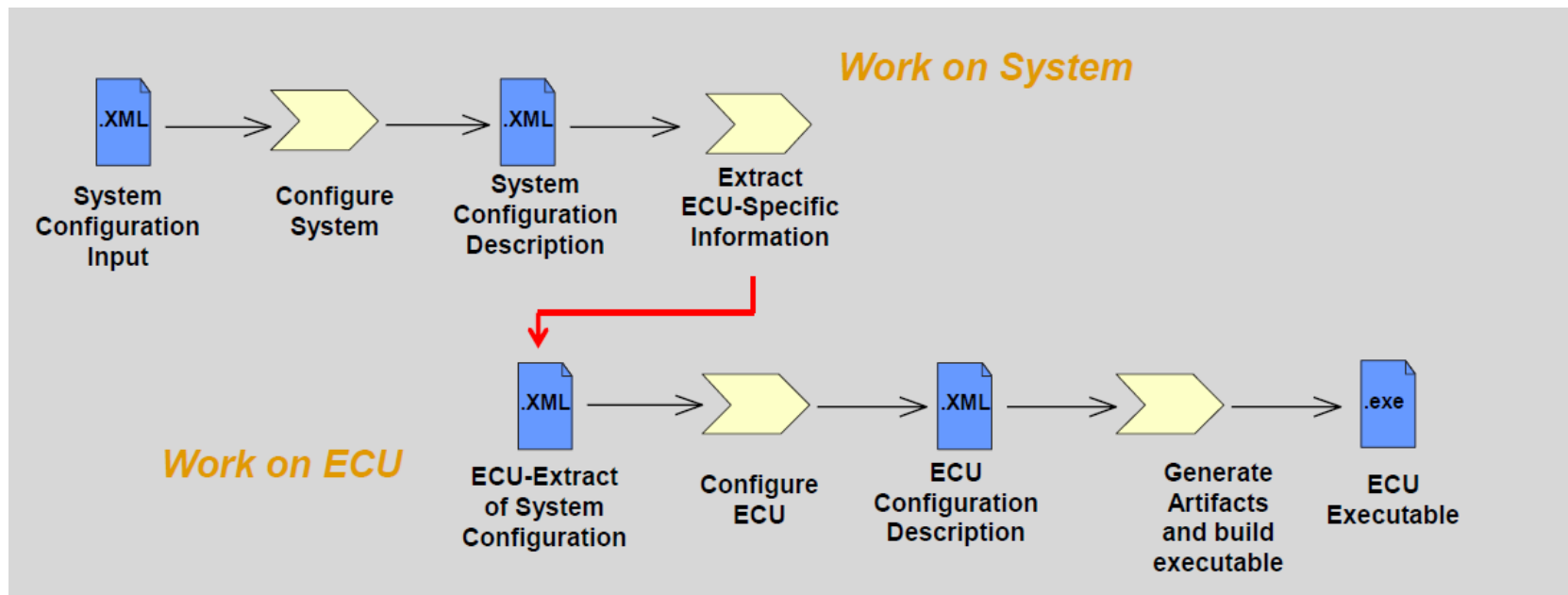
Workflow
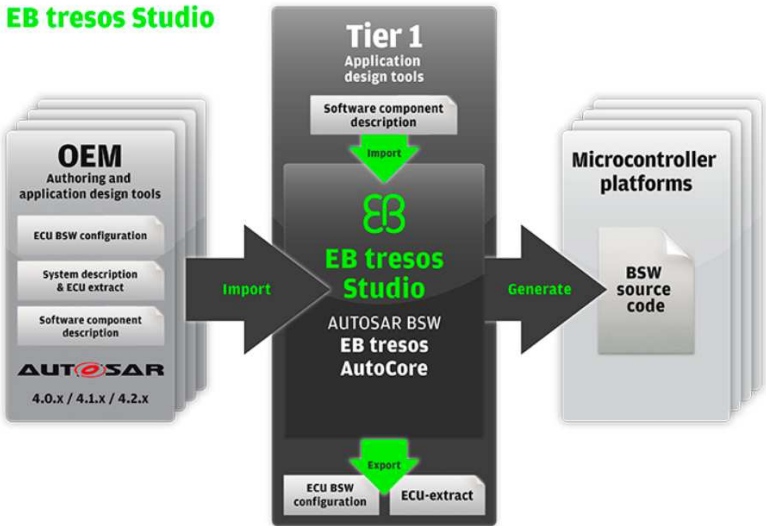
Integration with other EB products and technologies

Thank you
for your attention!