AUTOSAR

Isabella Stilkerich

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Department of Computer Science 4
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

http://www4.cs.fau.de/Research/KESO
www.autosar.org
http://www4.cs.fau.de/~isa
isabella.stilkerich@cs.fau.de

Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

Motivation

Climate Control Unit

Infotainment

Cruise control

Navigation

Fuel-injection

Rain sensors

etc.

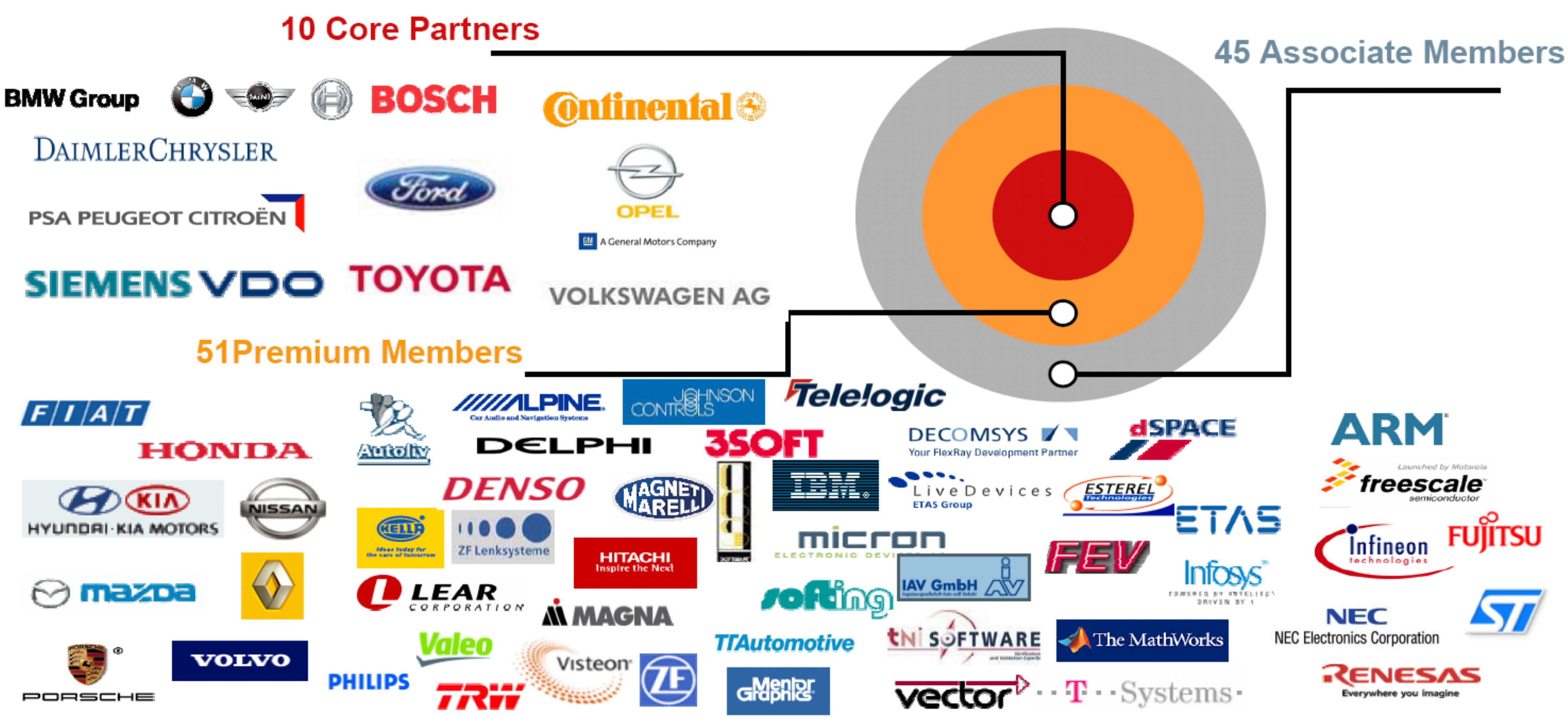
Brake system

AUTOSAR

- AUTOSAR development partnership founded in 2003
- Open standard for the E-/E-Architecture in the automotive field

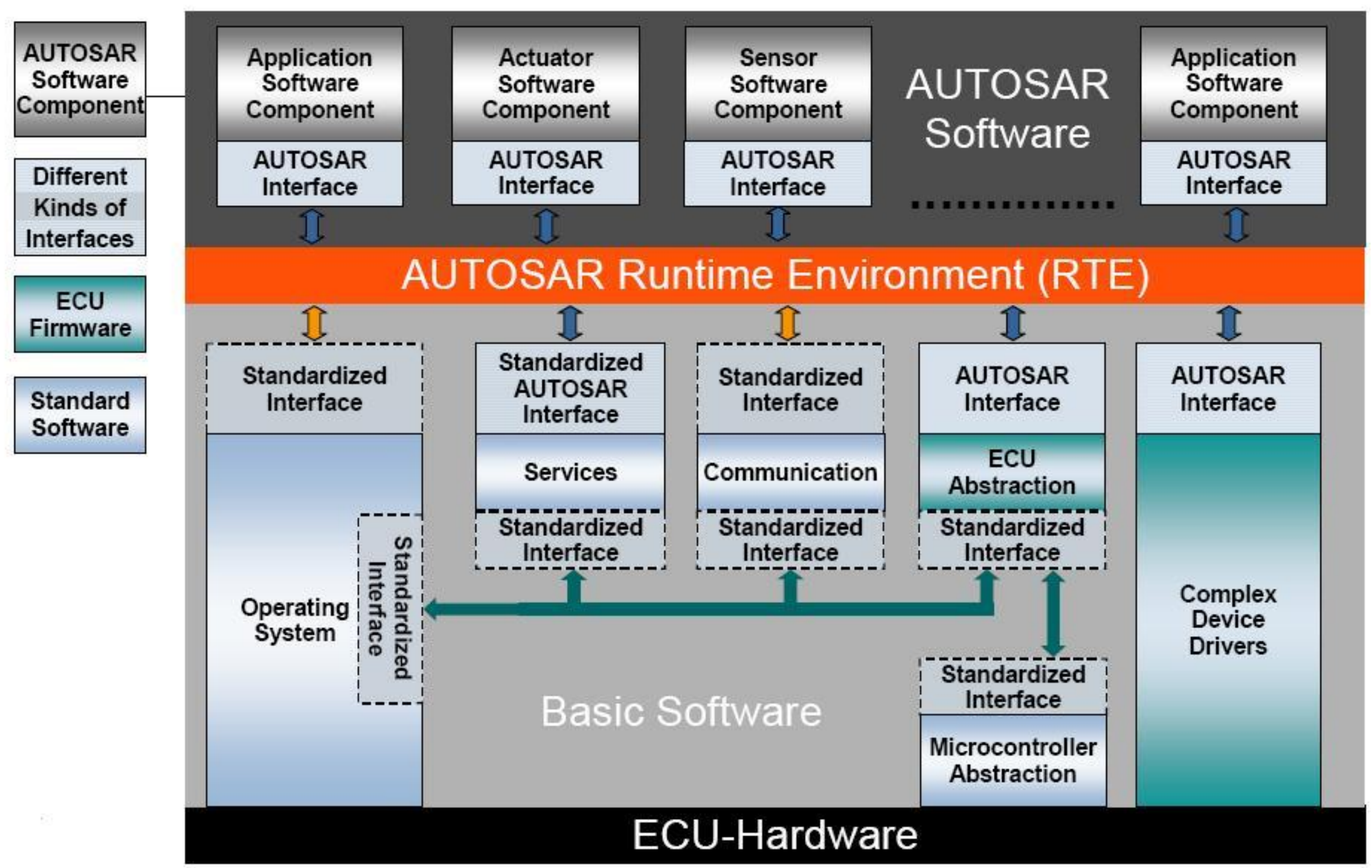
Cooperate on standards – compete on implementation

Core partners and members

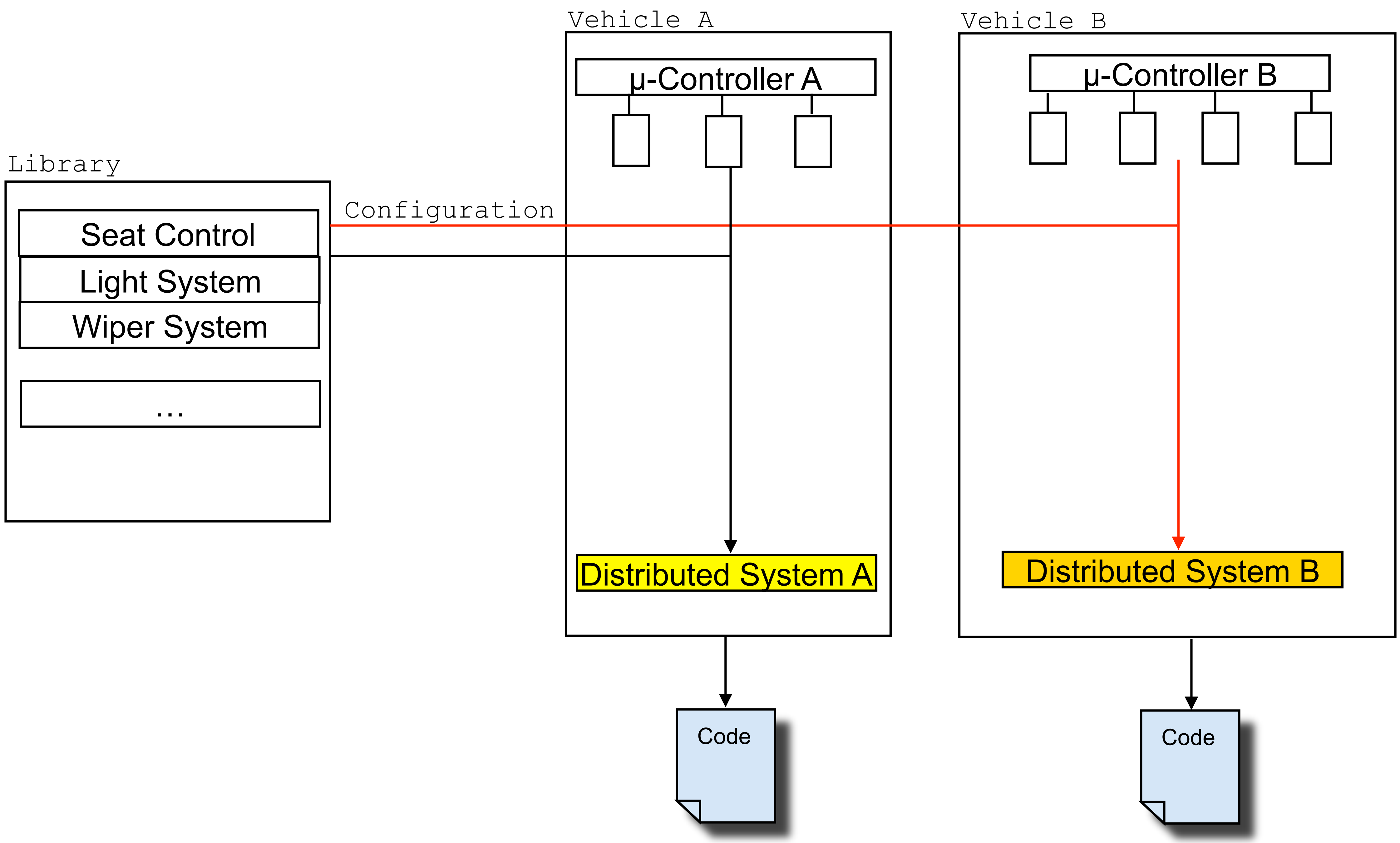


Benefits

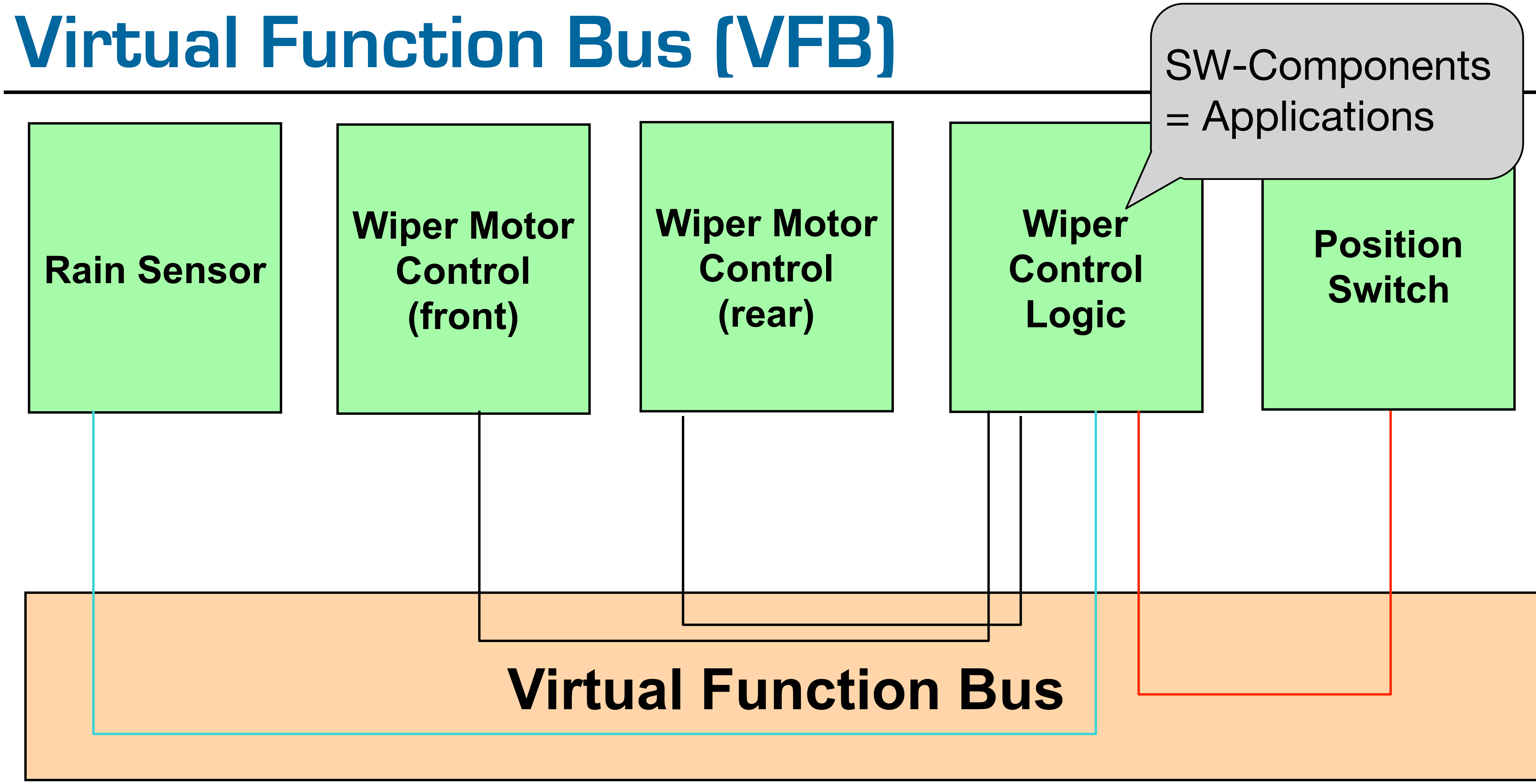
- E/E-Architecture is devided into several layers
- Abstraction from underlying hardware
- Provision of Basic Software (BSW) Modules
- Integration of application modules from various suppliers
- Standardization of
  - Exchange formats
  - Interfaces
  - Methodology



Reusability of function in different vehicles



Virtual Function Bus (VFB)

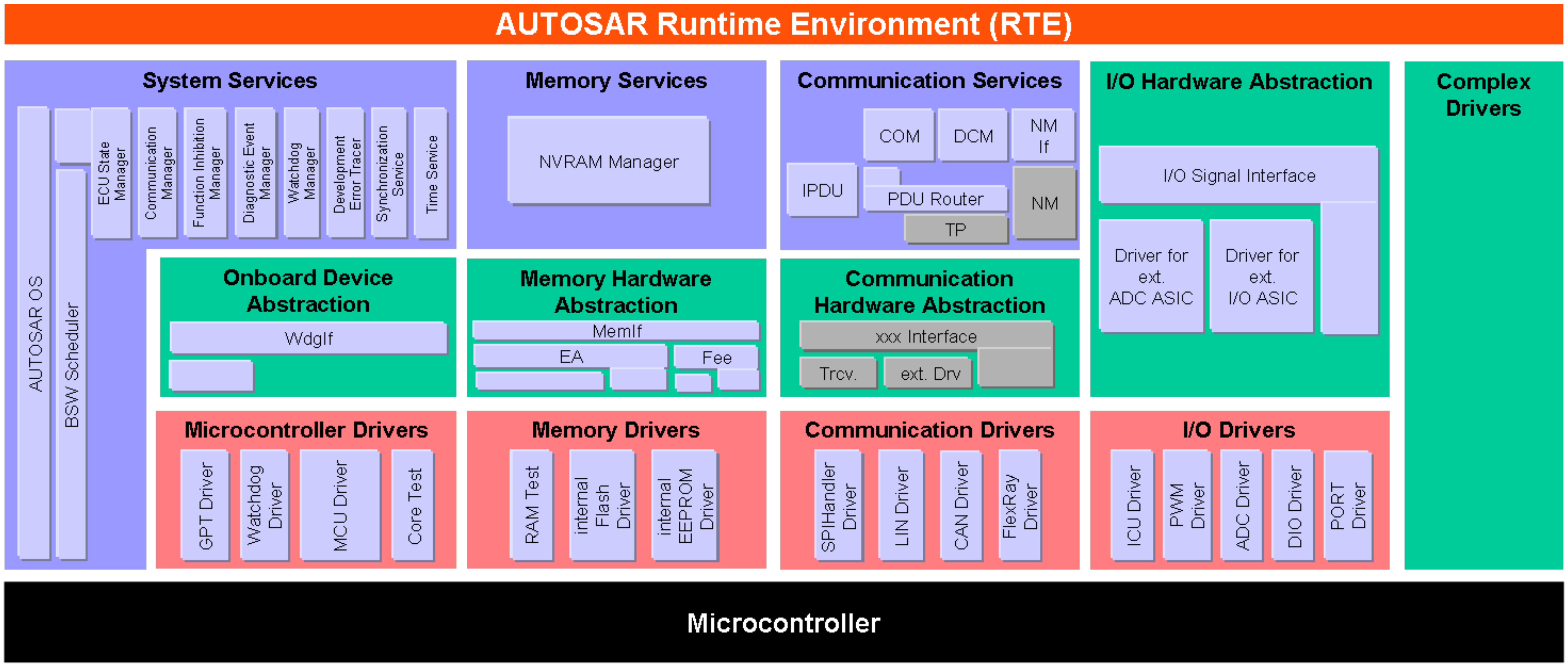


- Strict separation of application and infrastructure logic
- The VFB concept is implemented by the Runtime Environment (RTE)
- The RTE provides an abstract “instruction set”

Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

Architecture

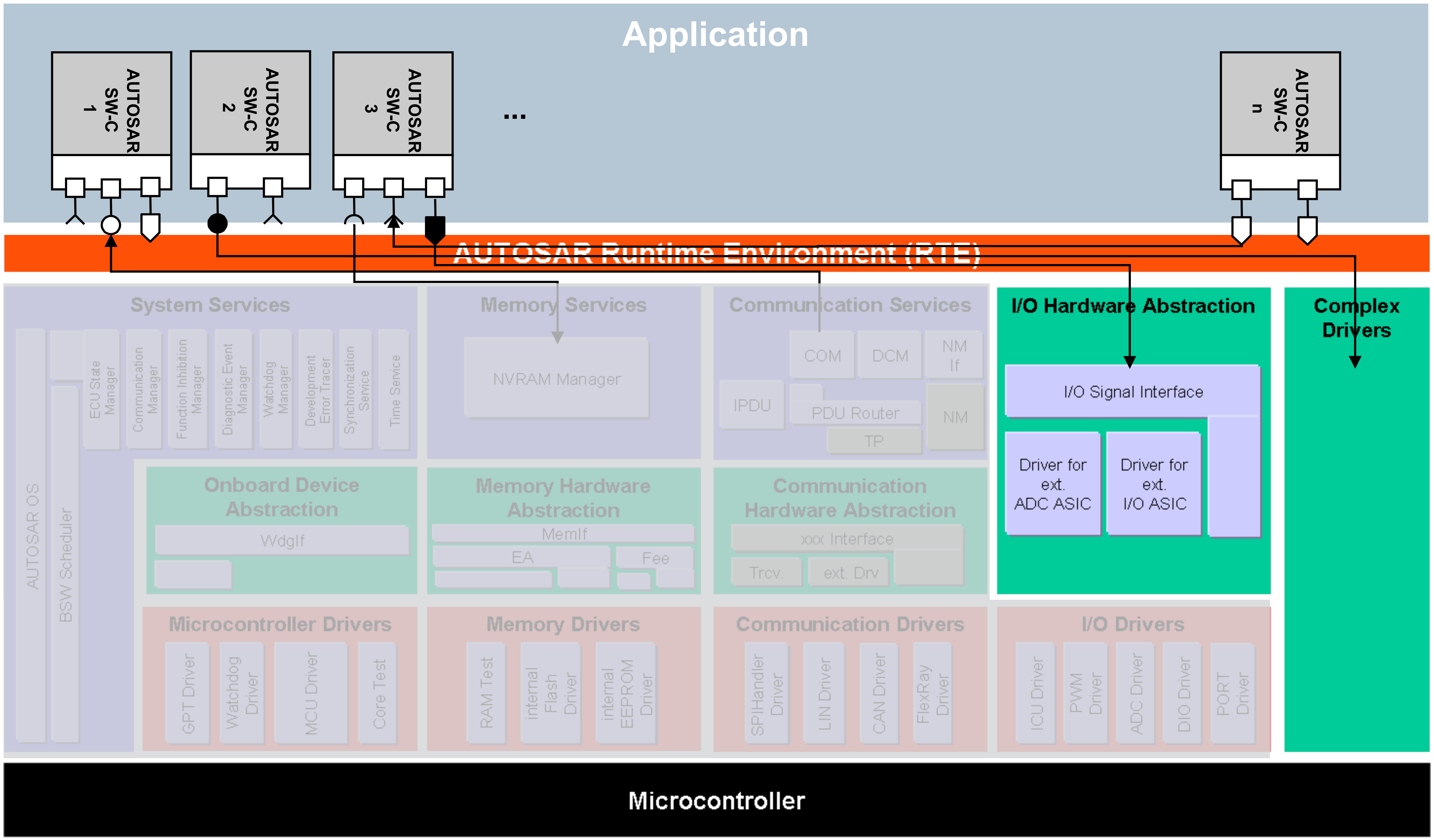


Service Layer : OS (Operating System), I/O (Input/Output), etc.

ECU (Electronic Control Unit) Abstraction Layer

MCAL (Microncontroller Abstraction Layer)

Discipline “Software Architecture”



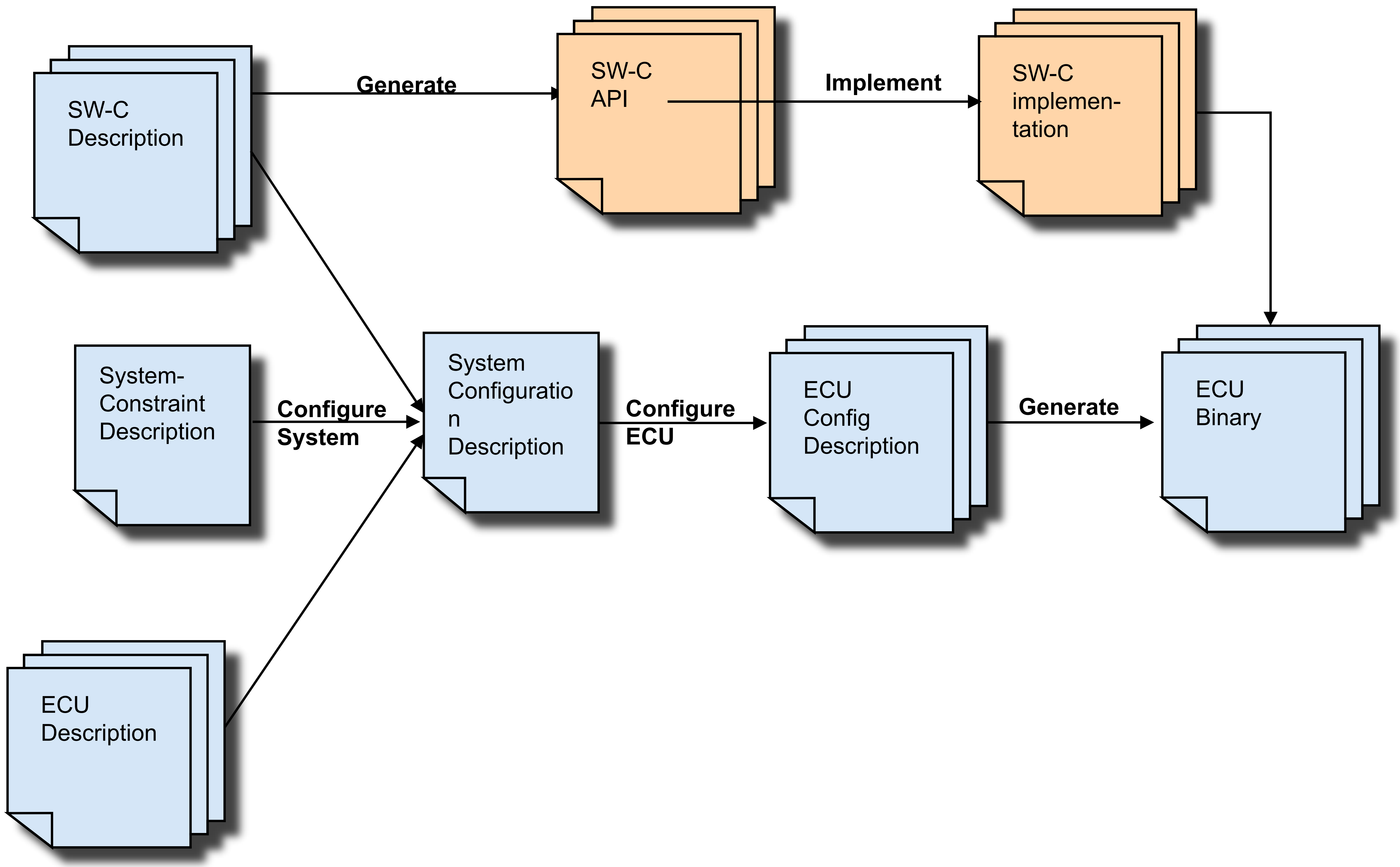
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

Methodology

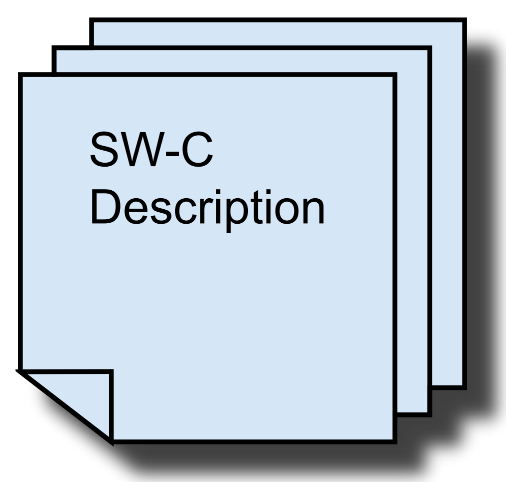
- Methodologies are similar to cooking recipes, they describe
  - the “ingredients“
  - when and how these are used
- AUTOSAR defines requirements on
  - the software architecture
  - the software development methodology

Methodology (Development)



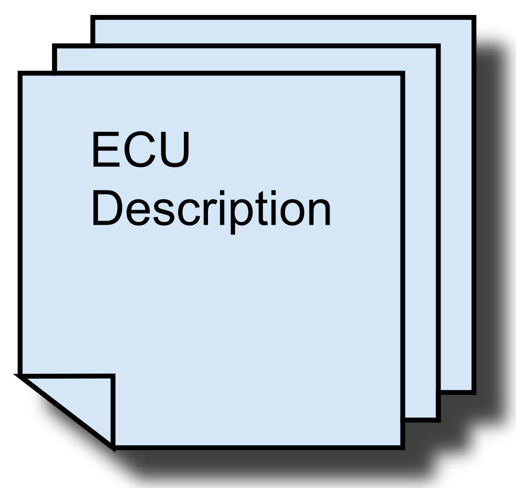
SW-C Description

- The description contains:
  - Component interfaces
  - Provided and required data and operations
  - Hardware and infrastructure requirements
  - Needs for services
  - Information on specific implementation
- Structure of information defined in the SW-C Template



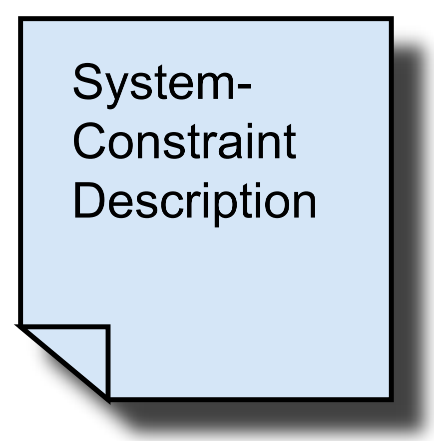
ECU Description

- ECU = Electronic Control Unit (*German*: “Steuergerät”)
- Description of
  - Hardware elements such as processing units, memory
  - Hardware ports
  - Hardware connections
- Structure is defined by the ECU Resource Template

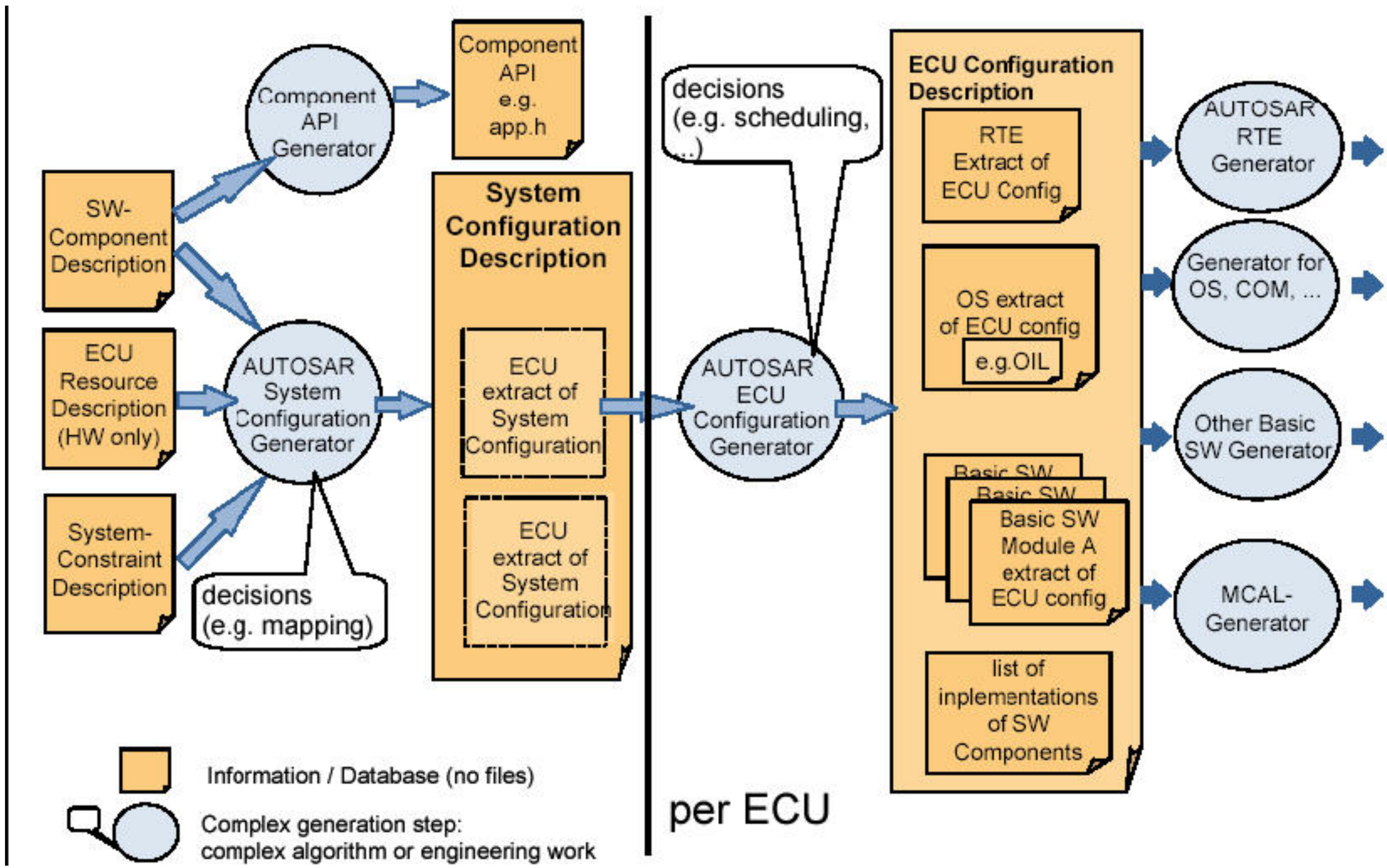


System Constraint Description

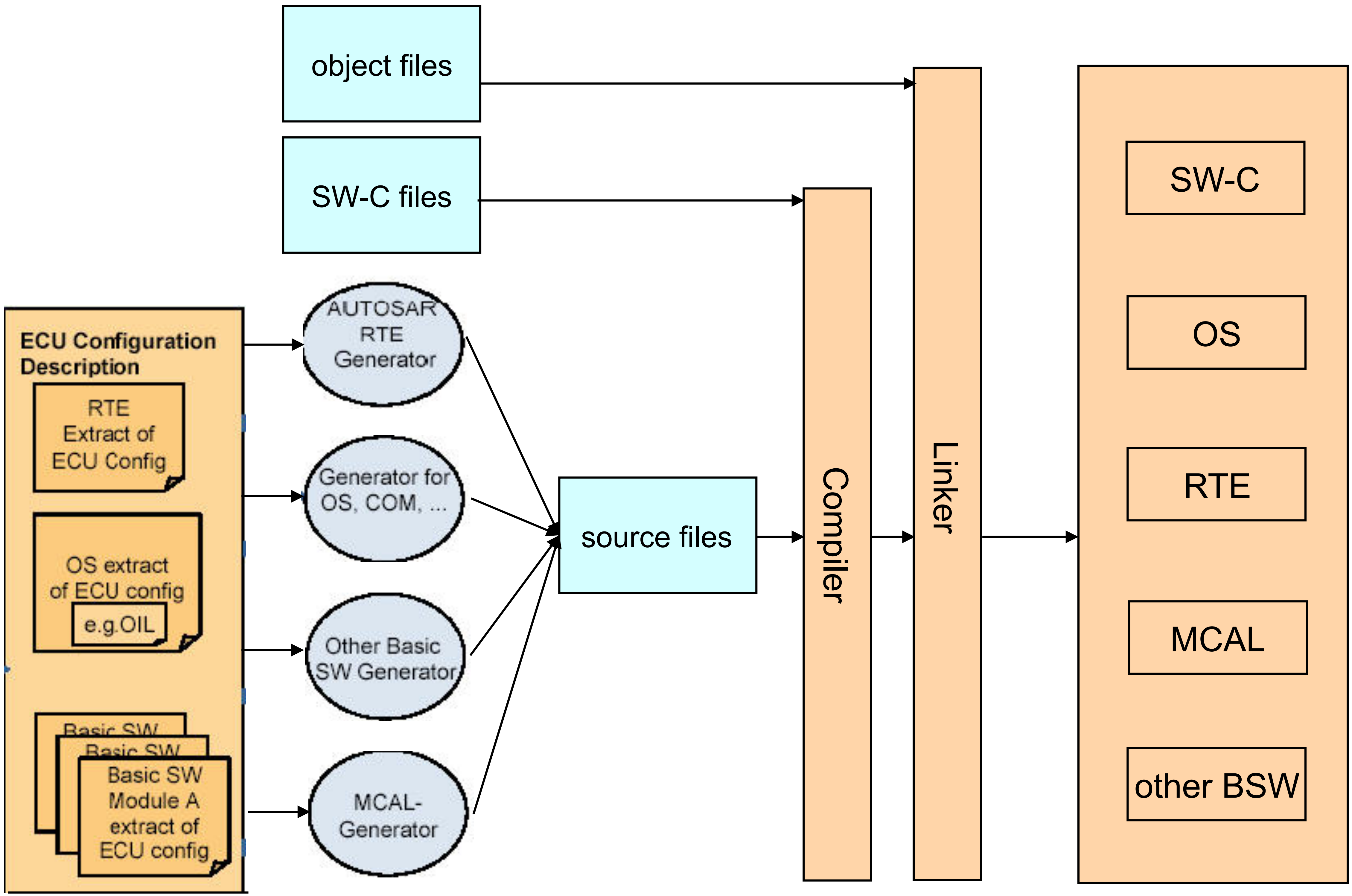
- Description contains:
  - Information on networking topologies (CAN, LIN, FlexRay)
  - Mapping and mapping constraints
  - Communication matrix
  - Protocols
- Structure defined by the System Template



System / ECU Configuration



System Image

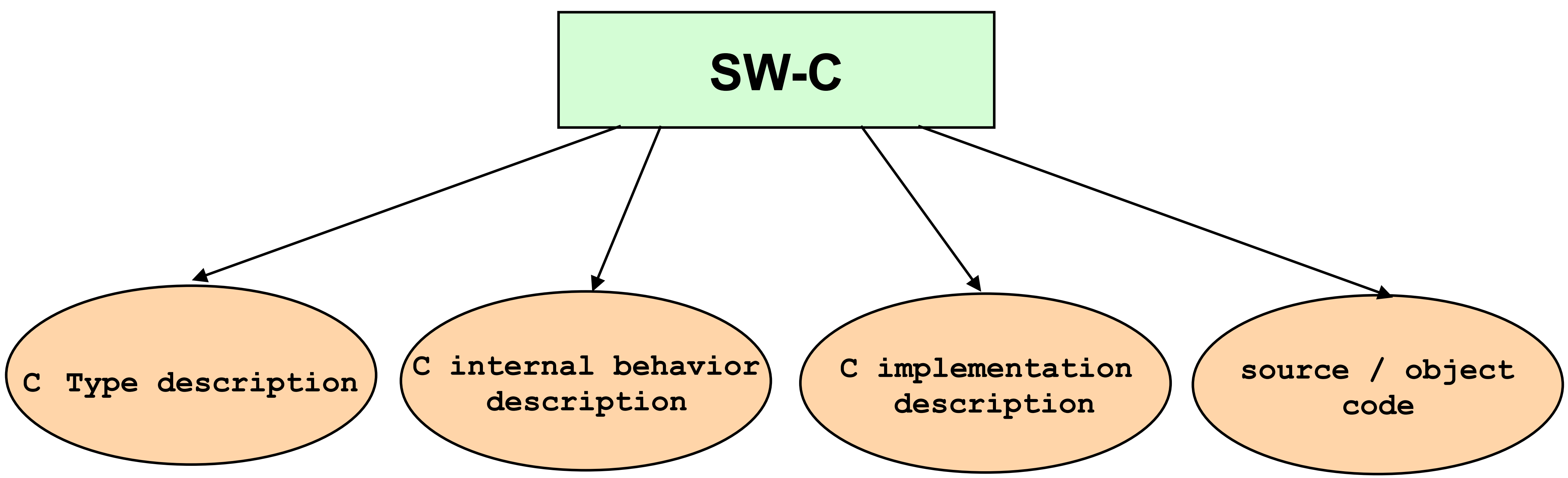


Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

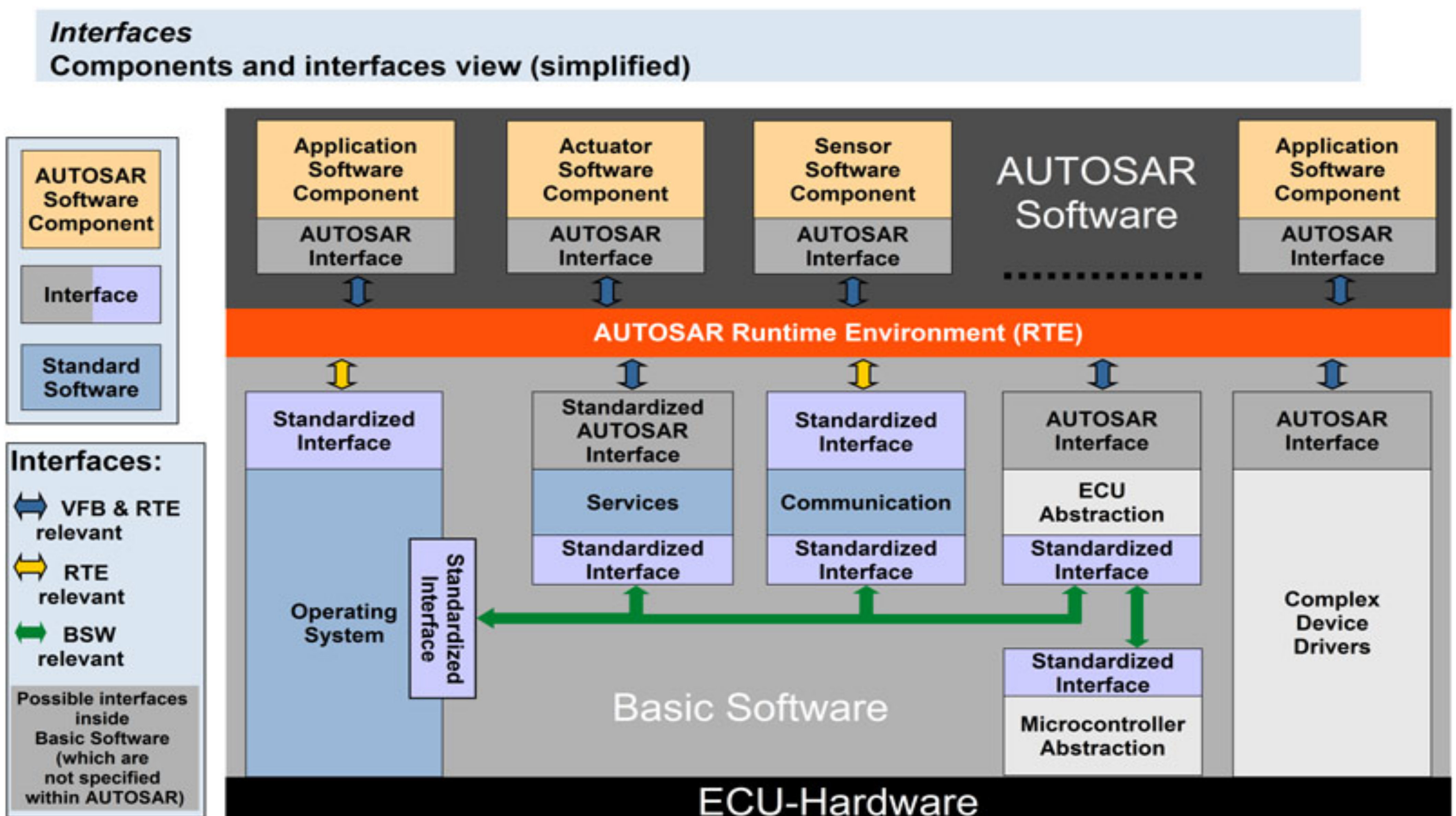
Software Components (SW-C)

- A SW-C needs to described in an abstract way (SW-C description)
  - Interfaces
  - Internal behavior (e.g. runnable entities, events)
  - Implementation
- Source/object code



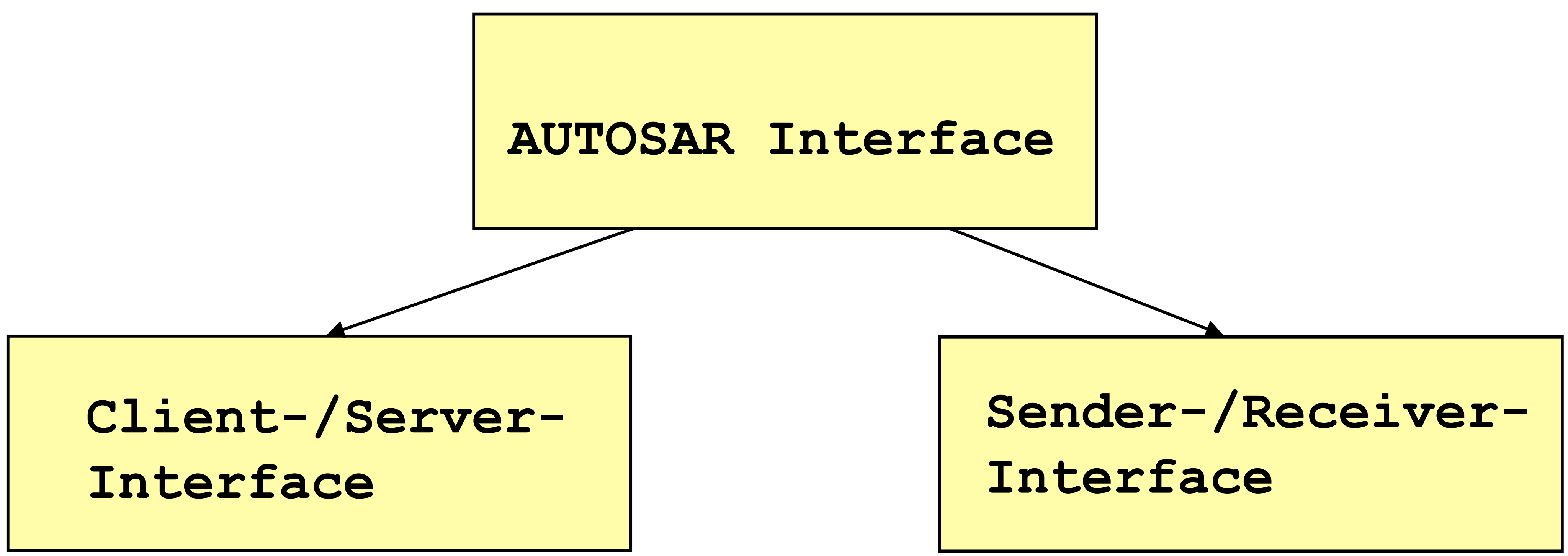
Interfaces (Classification)

- AUTOSAR Interface (relevant for SW-C)
- Standardized AUTOSAR interface
- Standardized interface



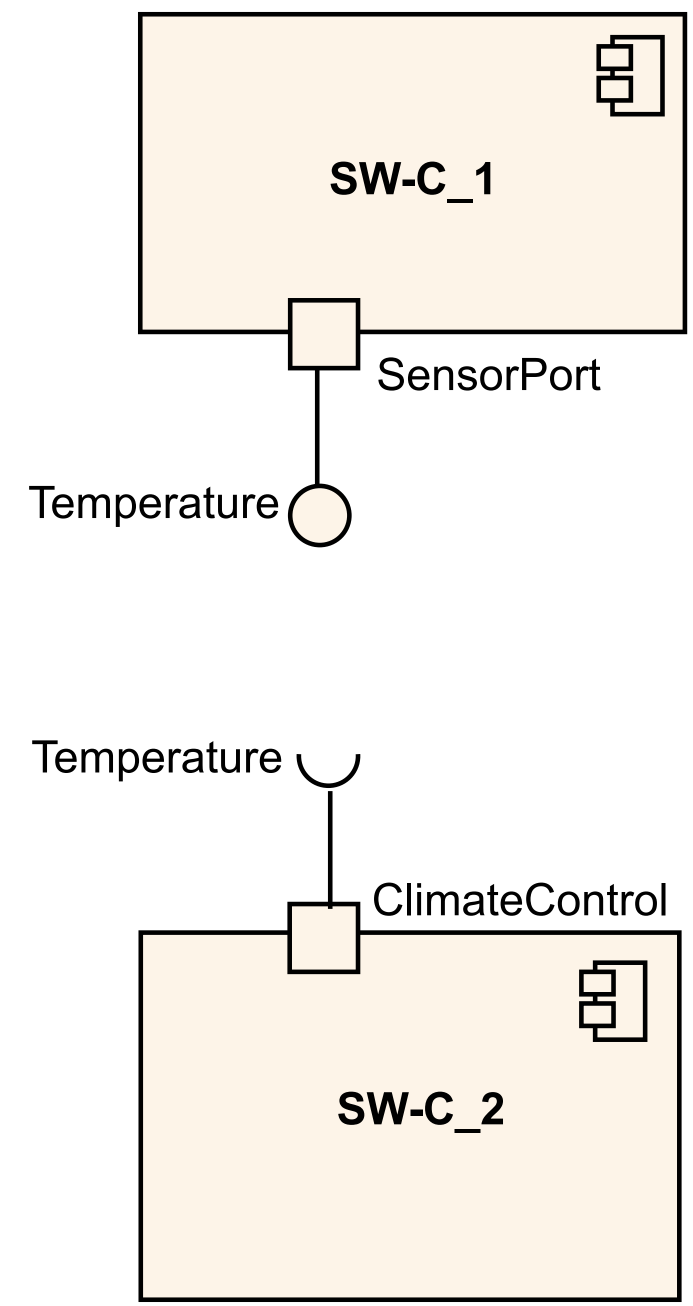
AUTOSAR Interface

- Interface=Typification of a connection between entities
- Definition of SW-C ports (interaction points=interface instance)
- Ports needed for SW-C Communication / Basic Software access
- Communication can be done locally or via a network



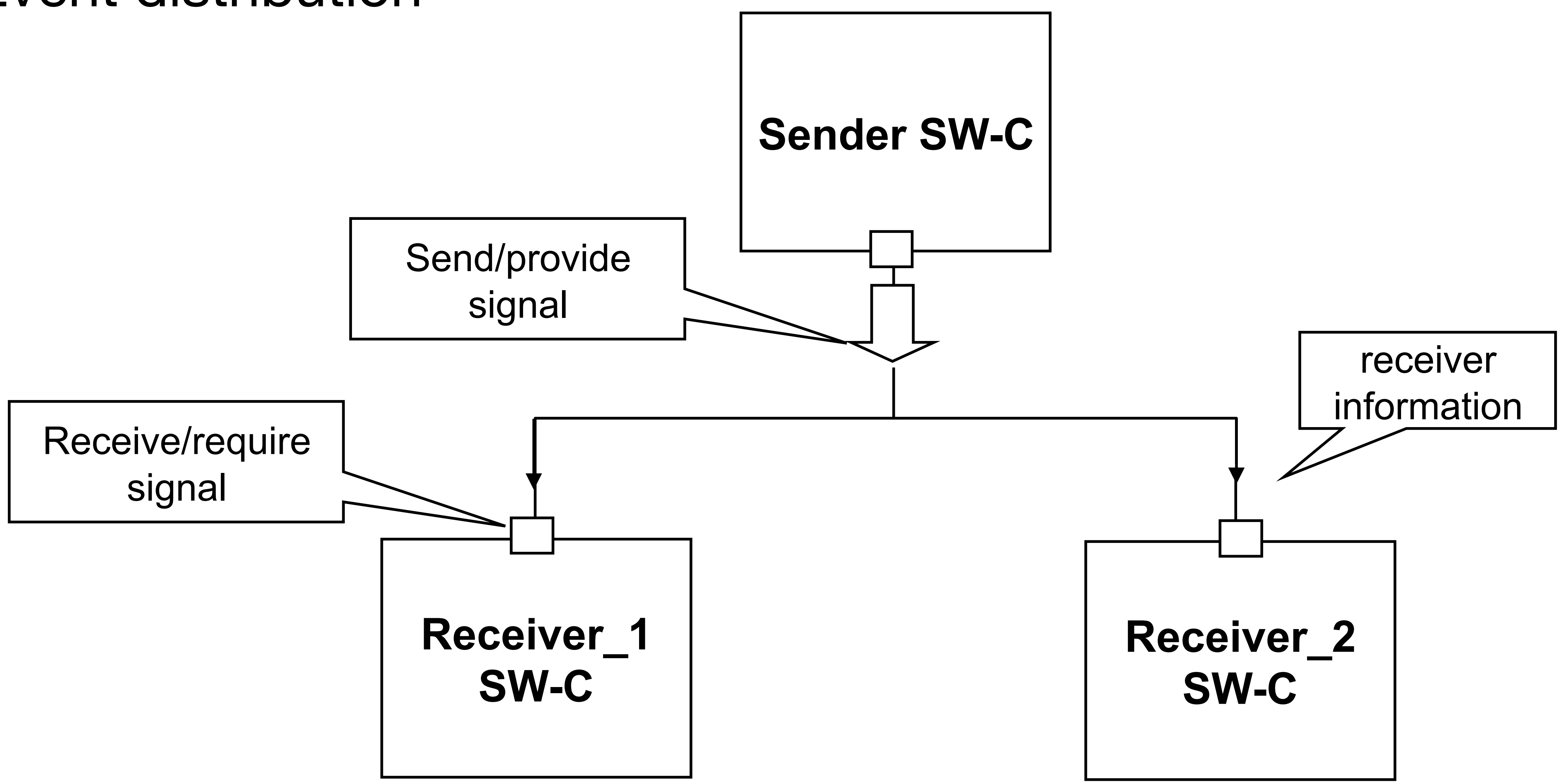
Interfaces and ports

- Interface
  - A type definition of an interaction point (port)
- Port
  - An instantiation of an interface.
- Provided port
  - A component offers an interface, that can be used by another component (created by providing component)
- Required port
  - A components needs access to an interface, that is provided by another component (created by the requiring component)
- Required and provided ports have to be connected in a composition component



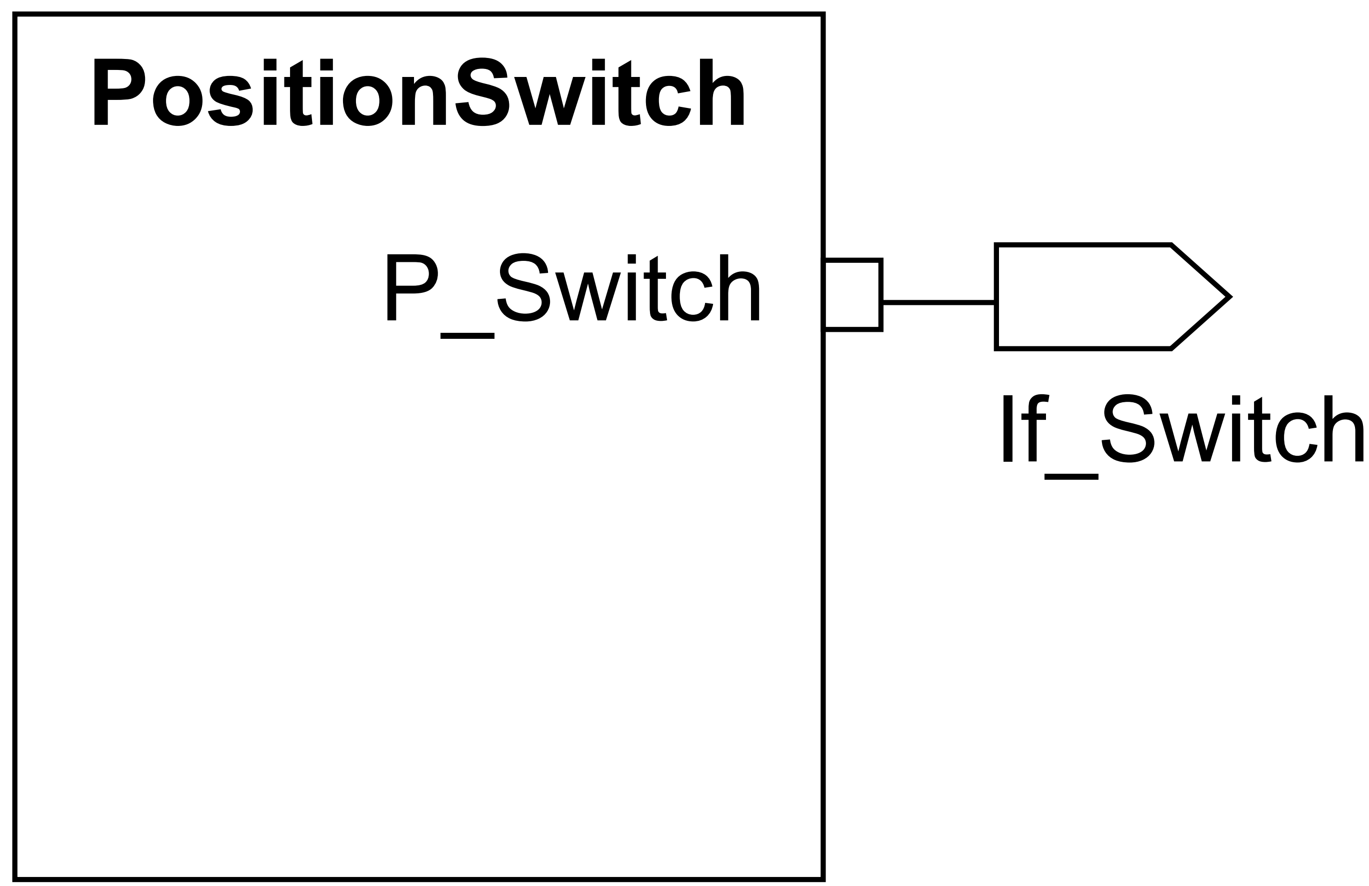
Sender-/Receiver-Interface

- m:n communication, uni-directional
- Different semantics
  - Data distribution: Last-is-best (explicit/implicit)
  - Event distribution



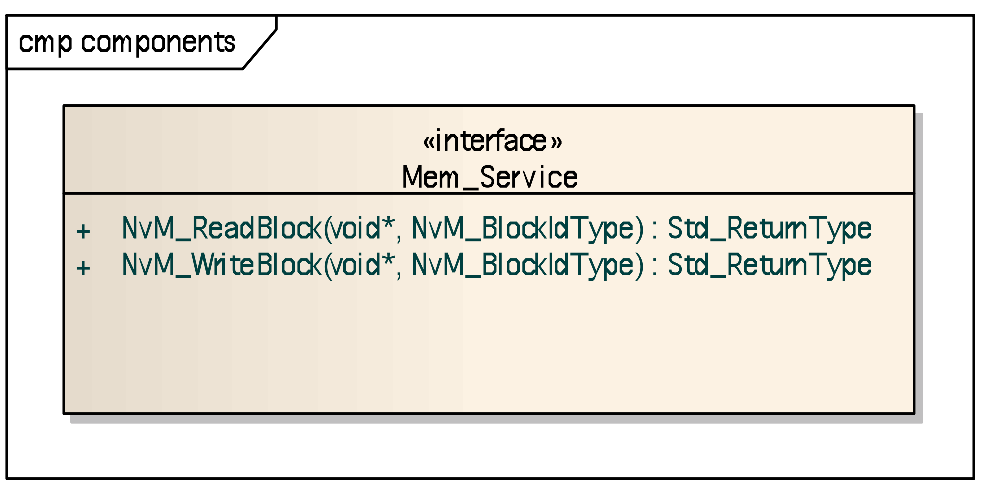
Sender-/Receiver-Interface (Example)

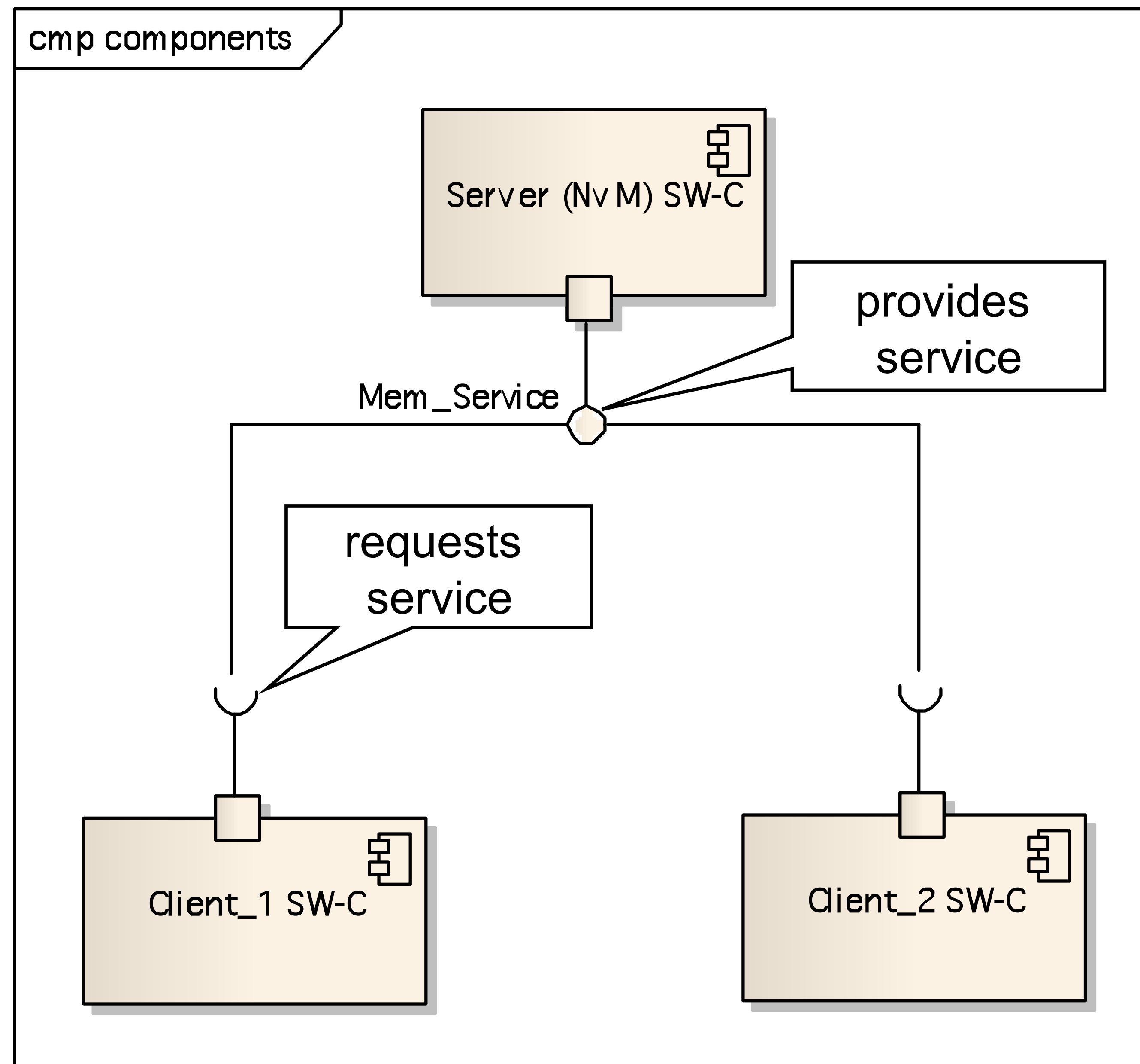
```
Sender-Receiver-Interface If_Switch {
        Boolean SwitchPosition
}
```



Client-/Server-Interface

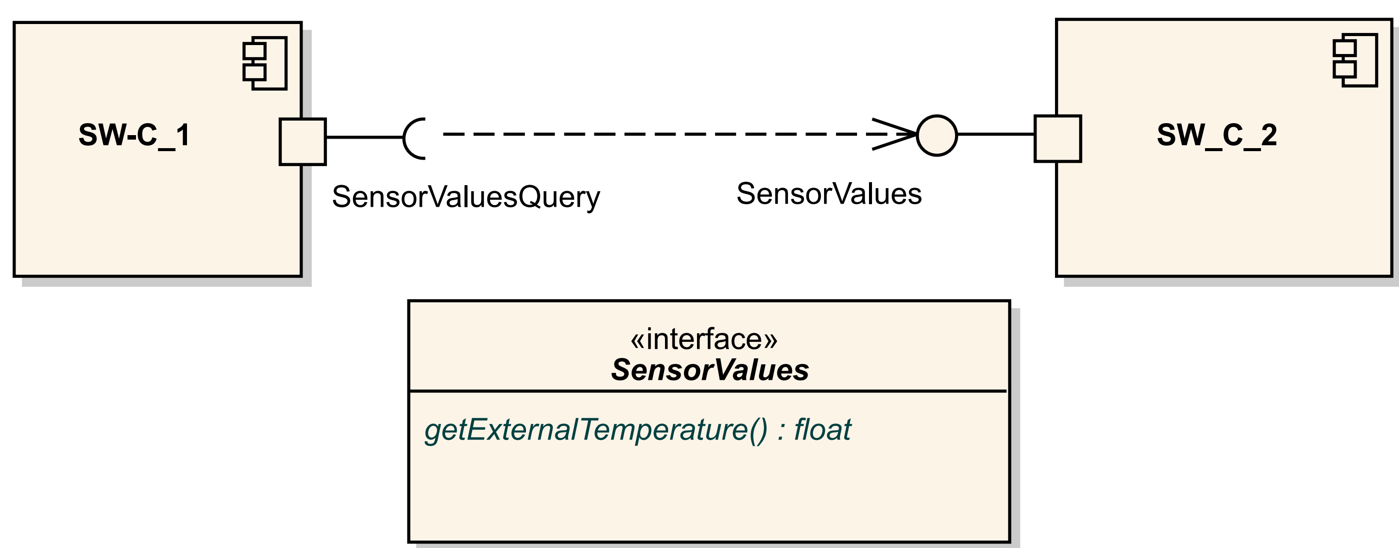
- n:1 communication
- Synchronous and asynchronous calls
- Server runnable in
  - Task context
  - Client context





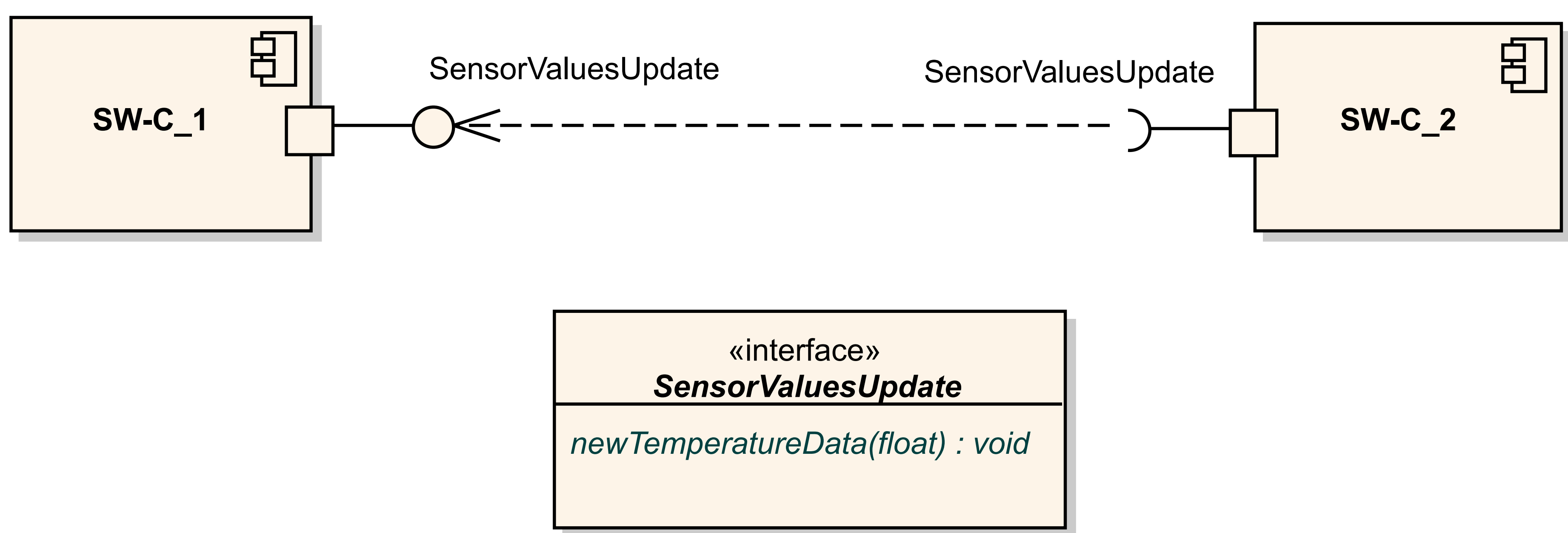
Interfaces and ports (revised)

- Sender-/Receiver-Interface:
  - Component sending the signal, defines the provided port – the receiver the required port
- Client-/Server-Interface: Different perspective
  - The direction of the control flow defines, which component has the provided or required port (function call direction)
- Example: SW-C_1 interested in sensor data by SW-C_2
  - SW-C_2 offers functionality, data flow from SW-C_2 to SW-C_1
  - Polling: SW-C_2 provides interface SensorValues



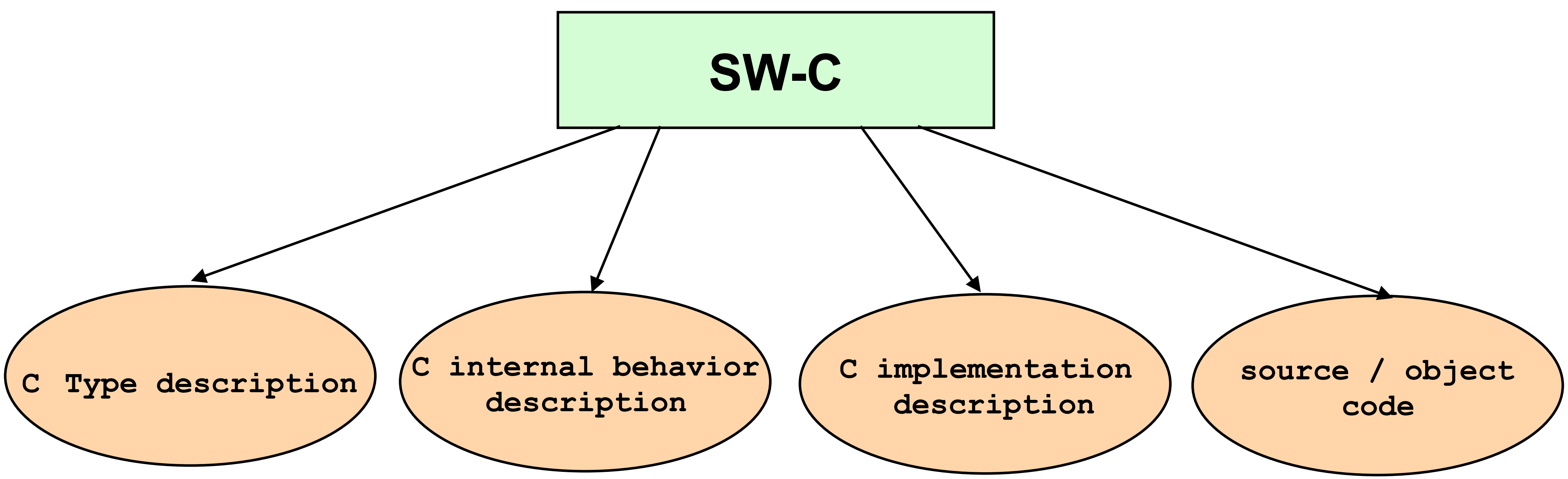
Interfaces and ports (revised)

- Direction of the control flow defines, which component has the provided or required port (function call direction)
- Example: SW-C_1 interested in sensor data by SW-C_2
  - SW-C_1 offers functionality, data flow from SW-C_2 to SW-C_1
  - Pushing: SW-C_1 provides notification interface SensorValuesUpdate



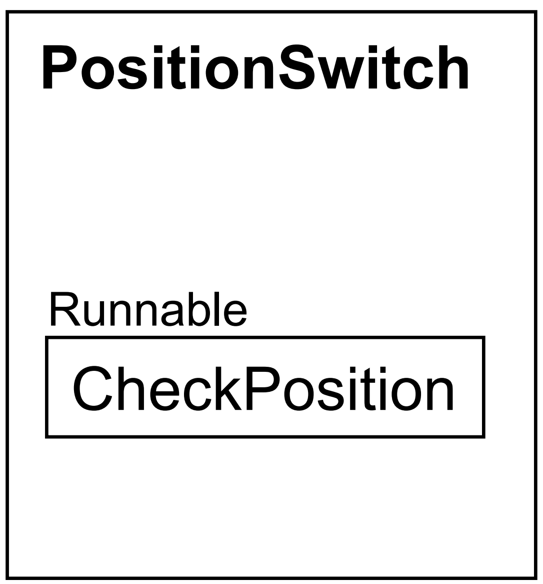
Software Components (SW-C)

- A SW-C needs to described in an abstract way (SW-C description)
  - Interfaces
  - Internal behavior (e.g. runnable entities, events)
  - Implementation
- Source/object code



Internal Behavior

- Runnable Entities (runnables)
  - Schedulable piece of code in SW-C
  - Usually mapped to tasks, execution triggered by RTE



Code in SW-C

```
void SwitchControl_CheckPosition() {
  Std_ReturnType success;
  boolean activated;
  … /* Read HW state and write to activated */ …
  success = Rte_Write_P_Switch_SwitchPosition(activated);
…
}
```

Generated RTE Code

```
TASK(Rte_Task) {
  …
  /* trigger Runnable Entity Switch_Position */
  SwitchControl_Switch_Position();
  …
}
```

Internal Behavior

- Events
  - Operation Invoked Event
  - Timing Event
  - Data Received Event
  - ...
- Points and accesses
  - Information about interaction between SW-C in Runnables
- Port API Options
  - Logical Representation of data entities on SW-C level

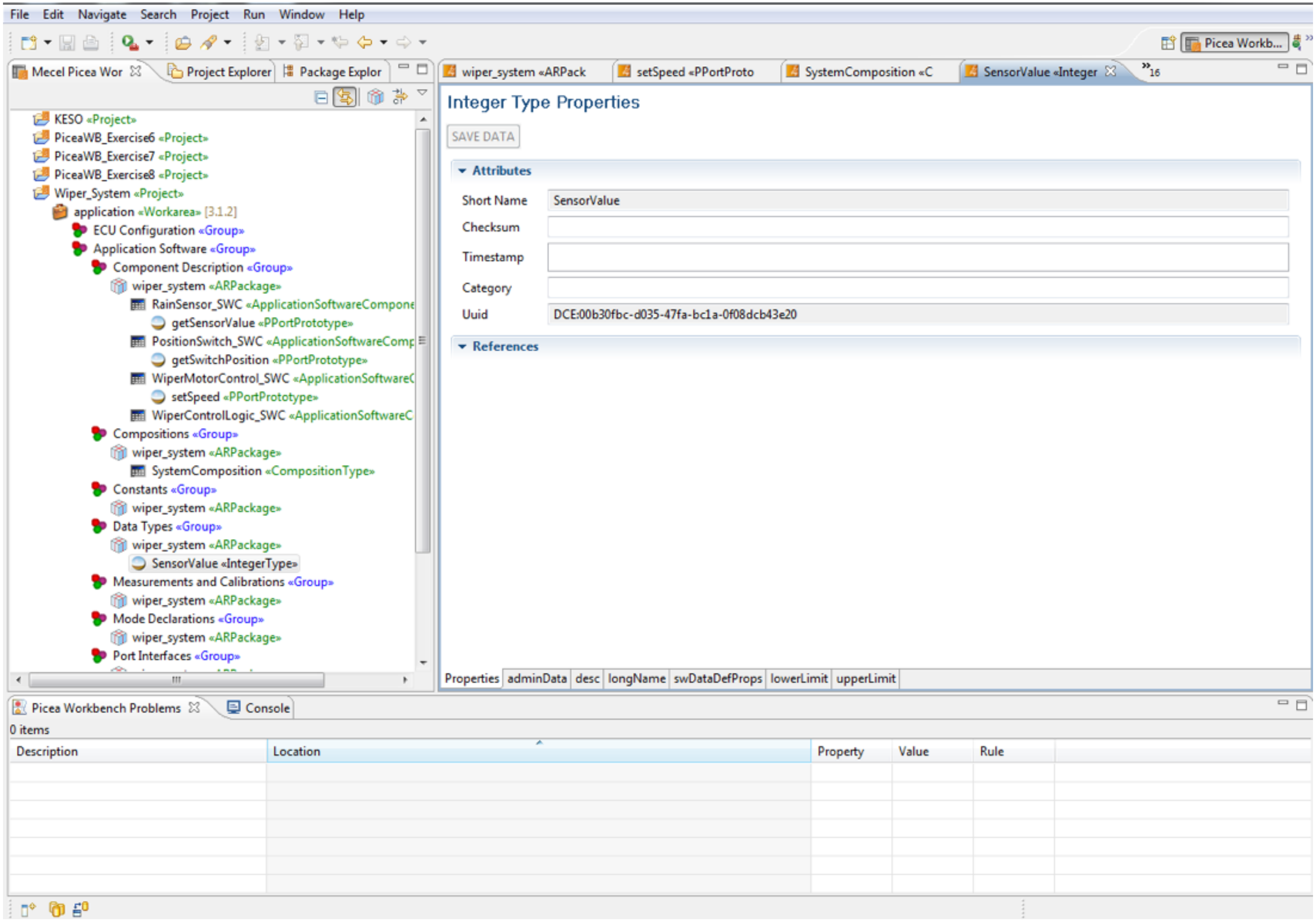
AUTOSAR SW-C vs Classical App

- No task bodies in SW-C
- SW-Cs communicate only via ports
  - No interrupt handlers in SW-C
  - No shared memory usage
- Abstract use of services via RTE
  - OS
  - Communication
  - Memory
  - I/O

Application Design

```xml
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/2.1.2">
  <TOP-LEVEL-PACKAGES>
    <AR-PACKAGE UUID="DCE:6c61e980-72a9-4ccb-b594-cc1396a065e8">
      <SHORT-NAME>bmw</SHORT-NAME>
      <ELEMENTS>
        <COMPOSITION-TYPE UUID="DCE:bfe7aa8f-0e08-4597-b728-10919a9277b6">
          <SHORT-NAME>TopLevelComposition</SHORT-NAME>
          <PORTS/>
          <COMPONENTS>
            <COMPONENT-PROTOTYPE UUID="DCE:755fcc65-7bc6-476d-b6f5-0dce15d7e1e2">
              <SHORT-NAME>VsmSample</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Appl/Vsm/VsmSample</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:ad22eadb-4e19-42d9-b1dc-76c2be761195">
              <SHORT-NAME>Cdc</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/Cdc/Cdc</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:4daa16c6-b27a-46ec-aea8-0eba88e81cb9">
              <SHORT-NAME>Coding</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/Coding/Coding</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:795eca54-4d5f-42f3-a27f-d8d302dda5af">
              <SHORT-NAME>Led</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/Led/Led</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:58acd0a0-85bd-4db4-b783-847dab2079d8">
              <SHORT-NAME>Pia</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/Pia/Pia</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:3def76dd-07da-44e0-938b-f60afd6282ab">
              <SHORT-NAME>SysTimeClient</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/SysTimeClient/SysTimeClient</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
            <COMPONENT-PROTOTYPE UUID="DCE:61b2e9cb-26bd-44c2-9eb6-a2725d7fba76">
              <SHORT-NAME>Vsm_Client</SHORT-NAME>
              <TYPE-TREF DEST="ATOMIC-SOFTWARE-COMPONENT-TYPE">/bmw/Services/Vsm/Vsm_Client</TYPE-TREF>
            </COMPONENT-PROTOTYPE>
```

Application Design (Revised)



SW-C development (Summary)

xml

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an event?)
  - or: mark as “can be invoked concurrently”

SW-C development (Summary)

xml

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an ev

programming language

- Writing source code for the SW-C
- Schedulable code resides in dedicated functions

SW-C development (Summary)

xml

- Model component (ports, interfaces)
- Connection between the SW-C
- Creation of a runnable, events
  - Entry in SW-C description
  - Needs on RTE (called cyclically, response to an ev

programming language

- Writing source code for the SW-C

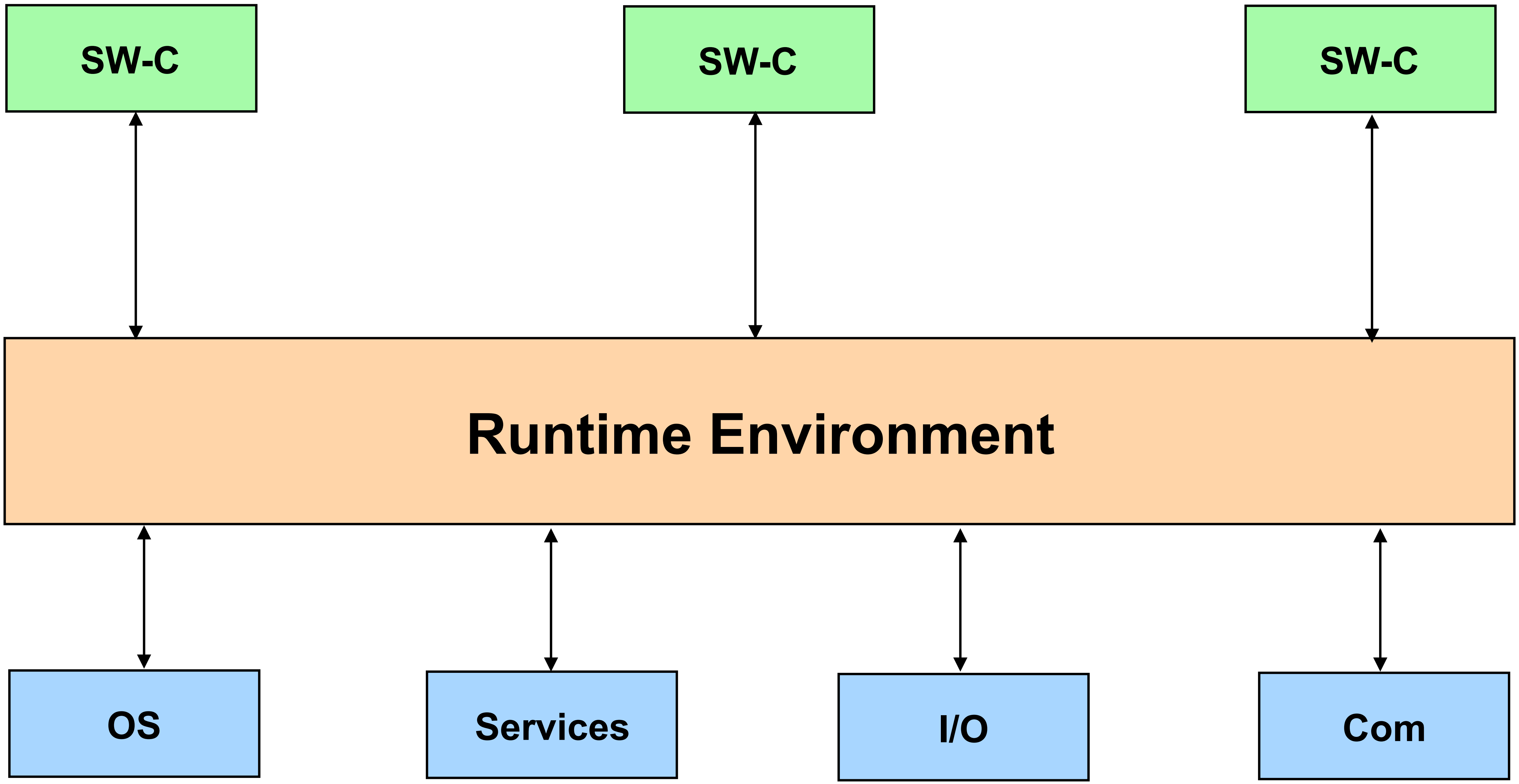
- Mapping of the function to an appropriate runnable

xml

Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

The Broker

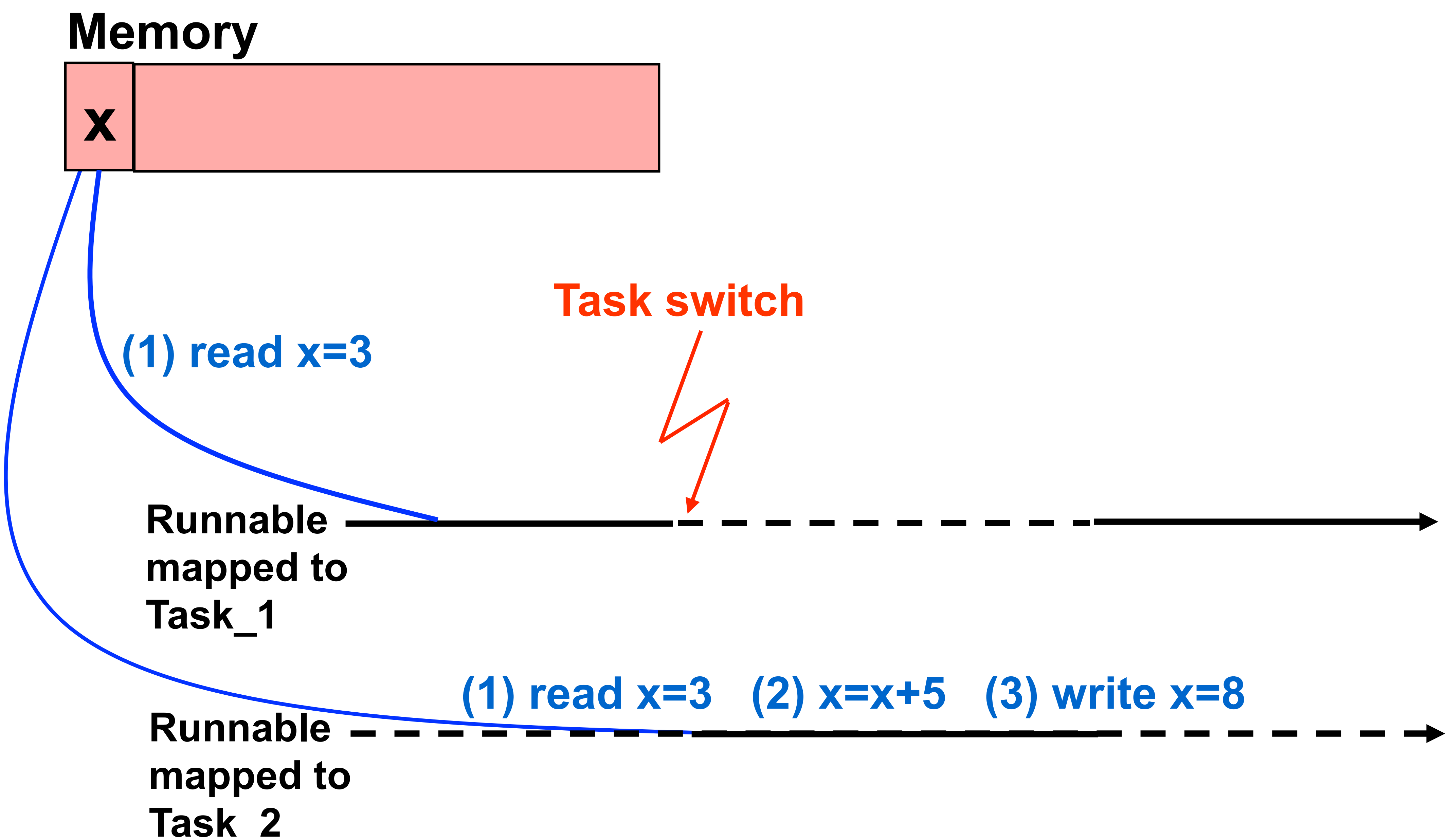


A Middleware for abstract Communication

- Classification of Communication
  - Intra-ECU com within a single SW-C (runnables)
  - Intra-ECU com between SW-C
  - Intra-ECU com between SW-C and BSW (system software)
  - Inter-ECU communication
- Implementation of communication patterns:
  - Client-Server
  - Sender-Receiver
- Tasks of the RTE are also:
  - Multiple instances of SW-Cs
  - Synchronization (concurrency issues)
- RTE code fully generated by the RTE generator
  - Two-phase approach

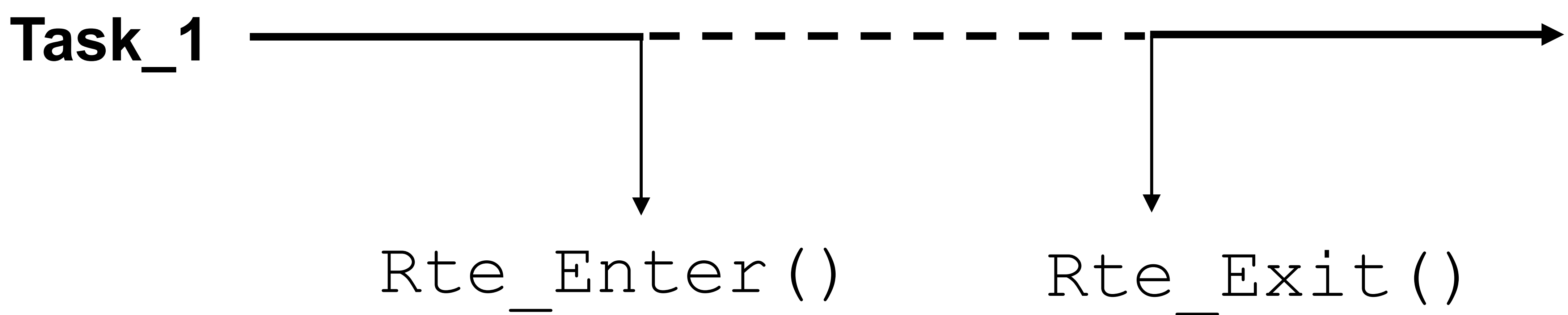
Intra SW-C Communication

- Example: Lost update issue
  - Notice: Can also occur, if REs have activated the canBeInvokedConcurrently feature, i.e. simple function call



Synchronization

- Developer sometimes has to ensure data consistency by
  - Exclusive areas
  - Inter-Runnable-Variables



Synchronization Techniques

- Disable / Enable interrupts (via OS API)
- Sequential scheduling
- Task blocking
- Employment of OS resources
- Cooperative runnable placement (Fine-grained task blocking)
- Copy strategy
  - Read, write copy (Implicit S/R communication)
  - Single writer
- Per-Instance memory (SW-C developer has to insure consistency)

- No semaphores allowed in AUTOSAR!

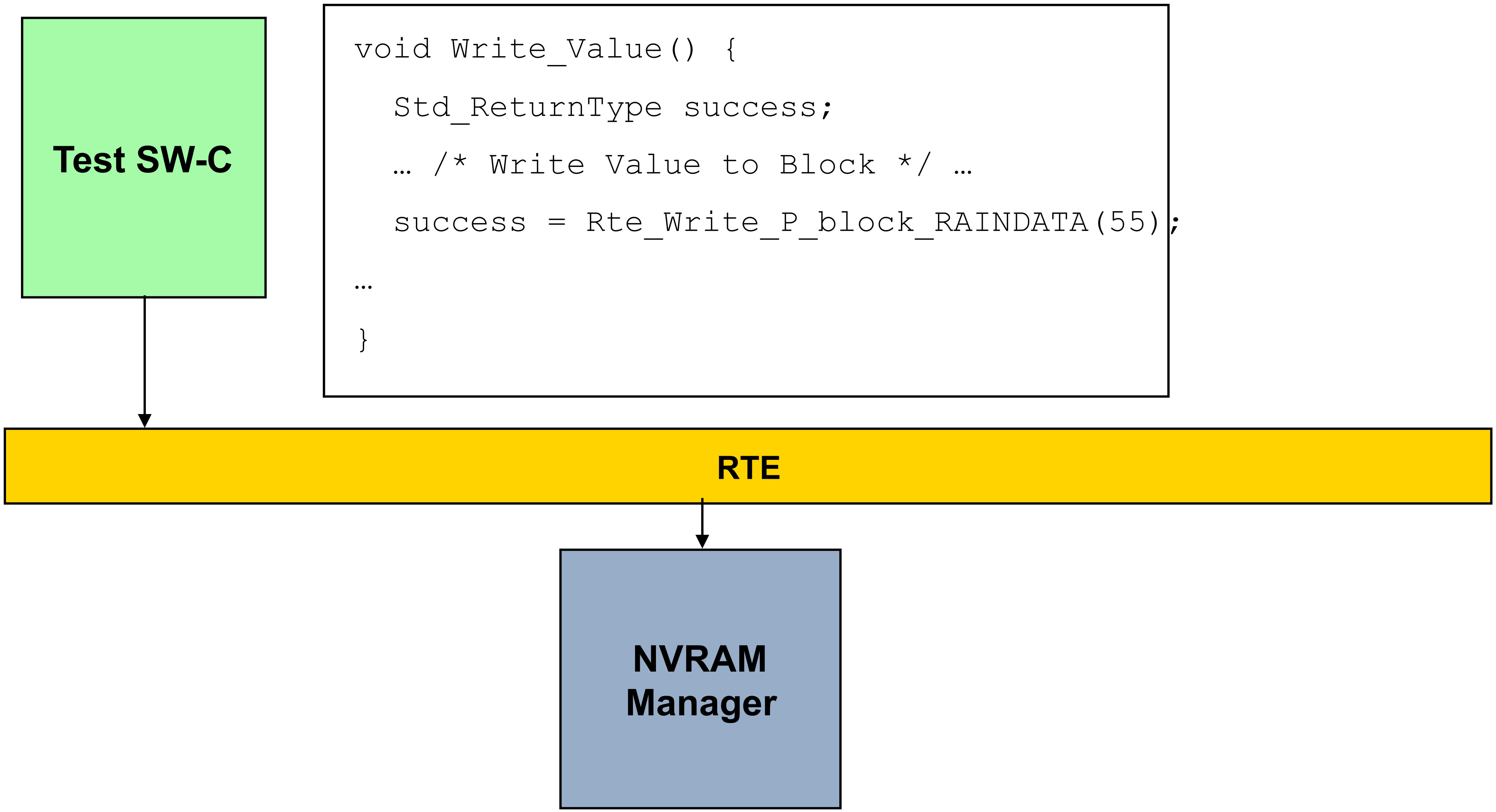
Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

AUTOSAR Basic Software (BSW)

- Communication Stack
- Memory Stack
- ECU State Management
- Complex Drivers
- System Stack
- I/O Stack

Memory Access - Example



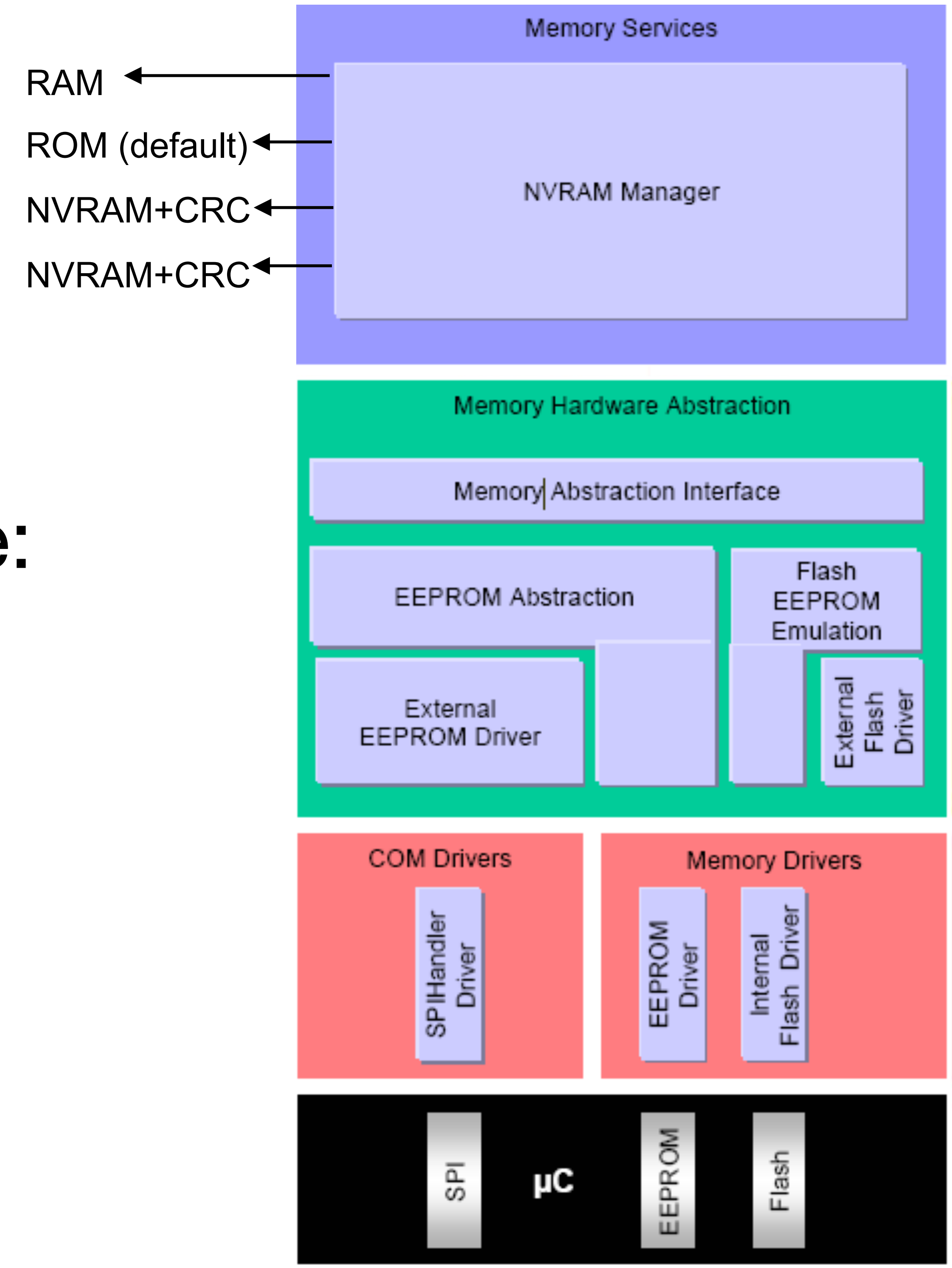

```
void Write_Value() {
  Std_ReturnType success;
  … /* Write Value to Block */ …
  success = Rte_Write_P_block_RAINDATA(55);
…
}
```

Memory Stack

- Memory services:
  - Abstract from location of memory devices
  - Checksum protection and reliable storage
- Memory Abstraction Interface: Broker

```
/* Memory Abstraction Interface */
Std_ReturnType MemIf_Write(
  uint8 DeviceIndex,
  uint16 BlockNumber,
  uint8* DataBufferPtr
)

/* Flash EEPROM Abstraction */
Std_ReturnType Fee_Write(
  uint16 BlockNumber,
  uint8* DataBufferPtr
)
```



Memory Interface

- Example 1: Single non-volatile device employed

```
#define MemIf_Write (DeviceIndex, BlockNumber, DataBufferPtr) \
  Fee_Write (BlockNumber, DataBufferPtr)
```

- Example 2: Multple device types employed

```
/* MemIf.h */
extern const WriteFctPtrType WriteFctPtr[2];
#define MemIf_Write(DeviceIndex, BlockNumber, DataBufferPtr) \
WriteFctPtr[DeviceIndex](BlockNumber, DataBufferPtr)

/* MemIf.c */
#include "Ea.h" /* for getting the API function addresses */
#include "Fee.h" /* for getting the API function addresses */
#include "MemIf.h" /* for getting the WriteFctPtrType */
const WriteFctPtrType WriteFctPtr[2] = {Ea_Write, Fee_Write};
```

Overview

- Motivation
- AUTOSAR Layered Architecture
- AUTOSAR Methodology
- AUTOSAR Software Components
- AUTOSAR RTE
- AUTOSAR Basic Software
- Conclusion

Conclusion

- Reduction of software complexity due to standardization
  - Costs
  - Time-to-market
- SW-C developers can concentrate on actual functionality
  - Model-driven approach
  - Faster development cycles
- Approved solutions for BSW and infrastructure
  - Code reusability
  - Reduce error-proneness
- Abstraction from Hardware facilitates portability
- Easier software module exchange
- Legacy software can be embedded in AUTOSAR architecture