AUTOSAR ARCHITECTURE

10+ years experience in Autosar & Automotive domain

Working for 2 out of the 9 Autosar core organizations

Copyright Information

Autosar Architecture

Outcome from this course:

- Complete understanding on the Autosar architecture and its needs
- Understand the BSW layers and some examples to understand how to write a BSW layer
- Complete knowledge on the ASW and the RTE layer concepts, and understanding with live examples
- To write an Autosar software with simple commonly available tools

Theory- 2hrs
Demo- 2.5hrs

- Introduction to Autosar
- Need for Autosar
- Autosar Architecture and layers
- Autosar BSW layers
- Autosar Interfaces
- ASW (Application Software)
  - Software Components and types with Live example
  - Ports and Port Interfaces with Live Example
    - Sender Receiver Interface and Client Server Interface
  - Compositions and Connectors
  - Runnables and Events
  - Application Software Consolidated summary
- Autosar RTE Layer (Run Time Environment)
  - RTE API's for Sender Receiver and Client Server Interface
  - RTE Layer Communication and Scheduling
  - RTE Generator and Tools overview
- Autosar Methodology with live example
- **Design an Autosar software**
  - **Complete Demo with a live use-case example**
  - Comparison of Autosar and Non Autosar software

Best way to approach this course ?

Introduction

Theory

- BSW
- ASW
- RTE

What is it ?
How to Choose ?

Demo

- BSW (MCAL, EcuAbstraction)
- ASW
- RTE

How to Implement?

Take the course continuously without gaps

Interrelate the Demo And Theory section

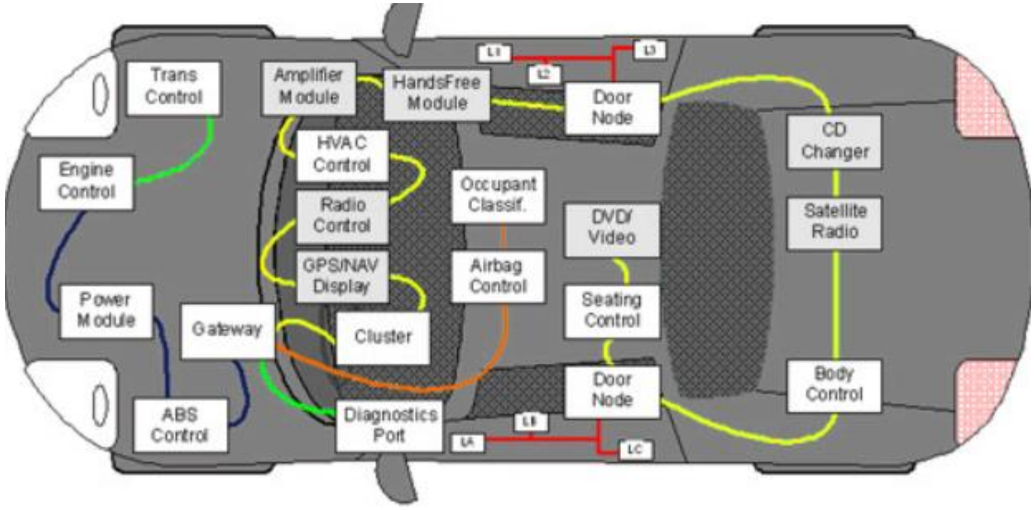
Queries? Instantly post them on Q&A section and I will answer them for you

Wish you a good learning!.

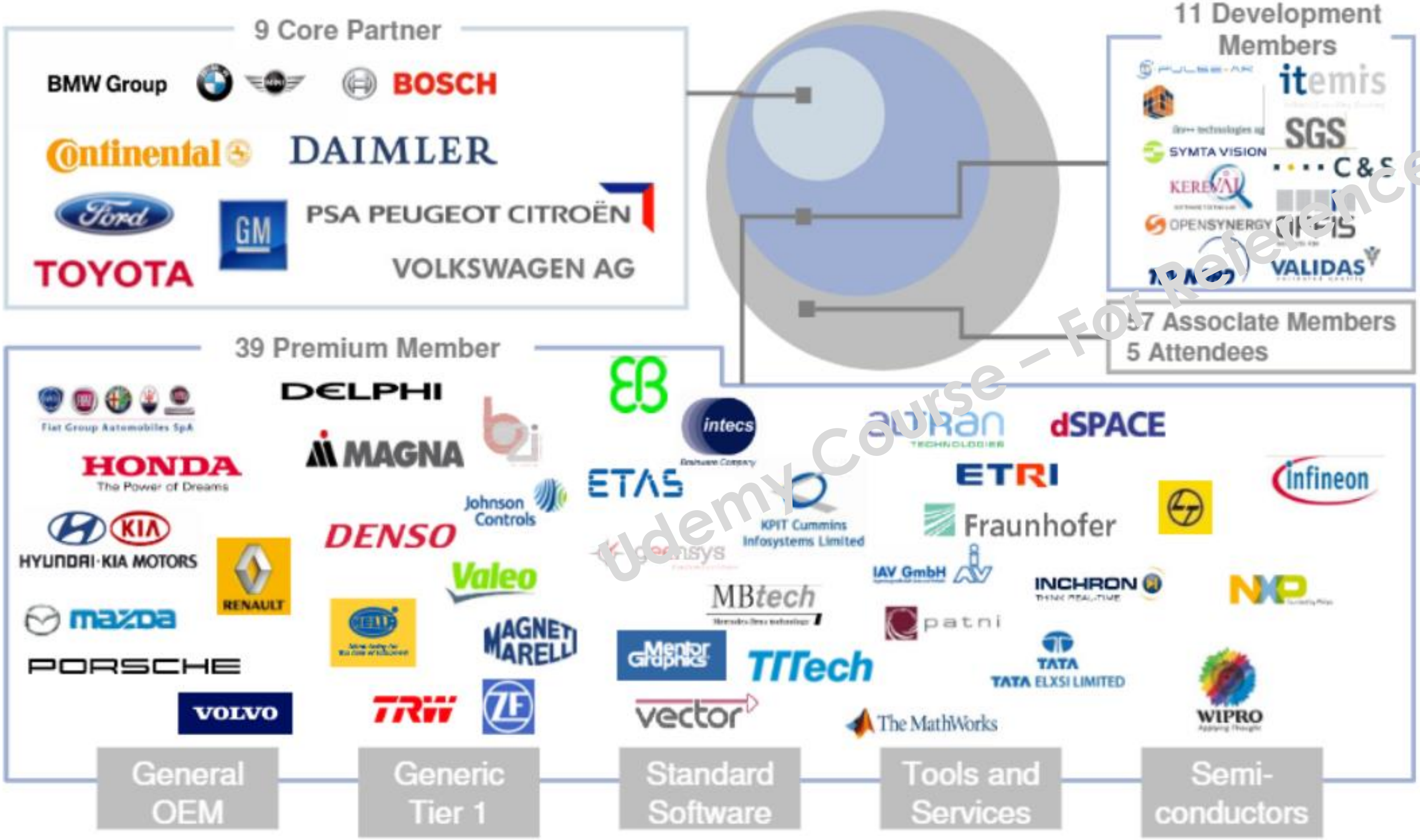
AUTOSAR

AUTOmotive Open System Architecture

- Layered Architecture which was developed in 2003
- Worldwide development partnership from automotive OEMs, suppliers and other companies in the software, semiconductor and electronics industries
- AUTOSAR aims to standardize the software architecture of Electronic Control Units (ECUs)
- Open Standard https://www.autosar.org/



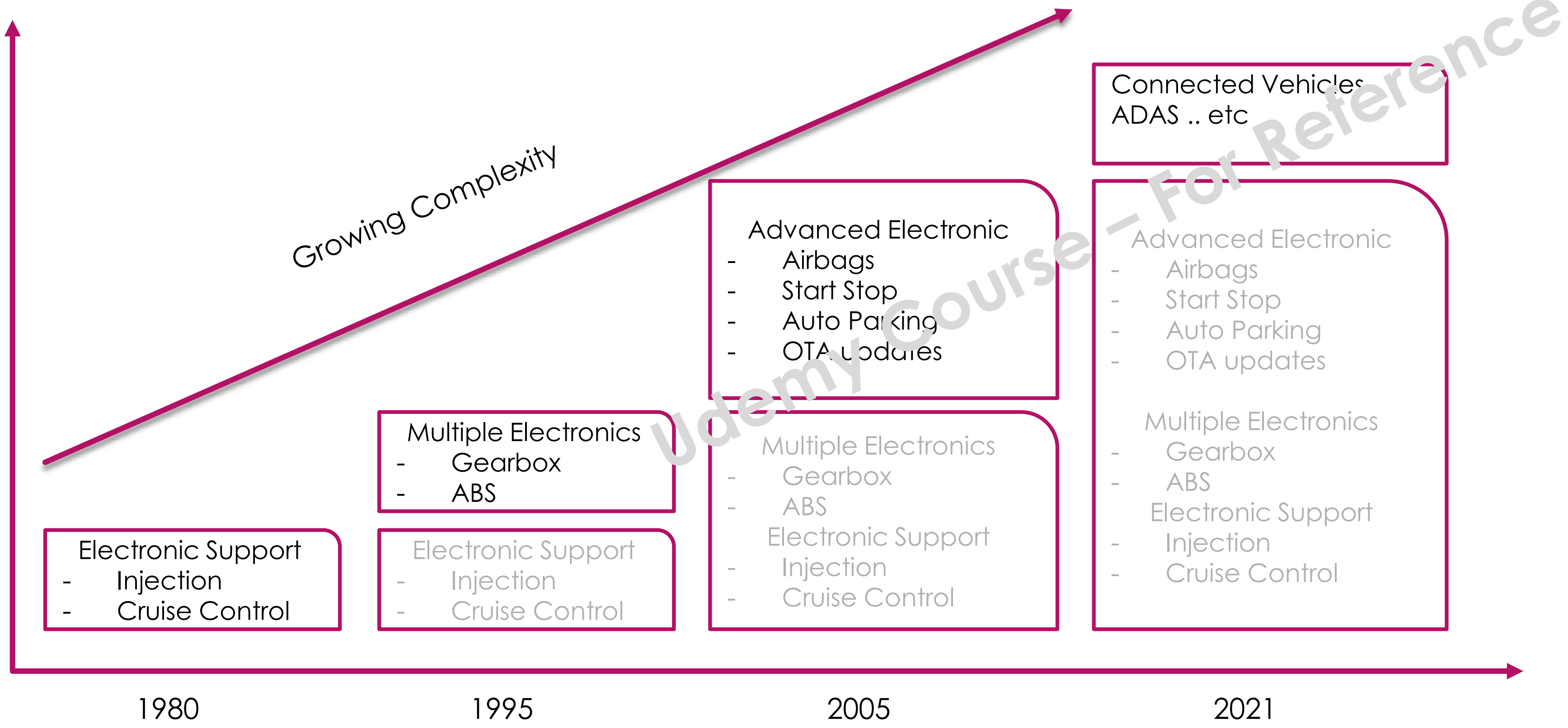
About Autosar

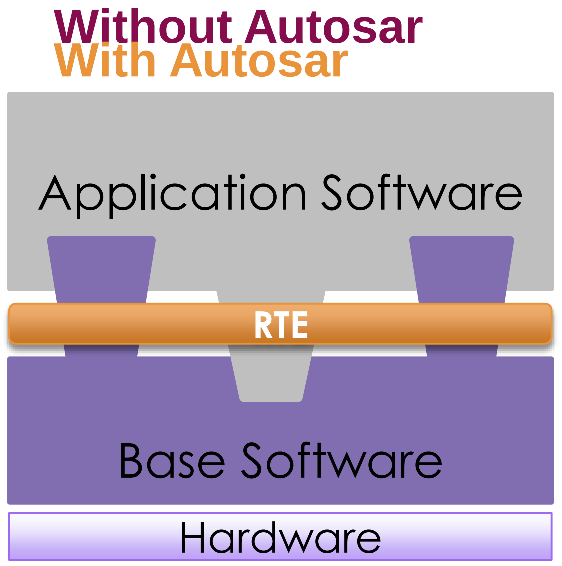


- Core Partner
  - 9 Core Partners
- Premium Members
  - Leading organisations from different industries
- Development Members
  - Small companies and start-ups
- Associate Members
  - Autosar followers
  - Can utilize the standards

Why do we need Autosar ?

- **Easy Handling:** Handling Increased complexity of Automotive software
- **Abstraction:** Abstraction of hardware from software, making development more flexible
- **Reusability:** Reuse software modules across Customers
- **Fast To Market:** Establish development distribution among suppliers
- **Competition:** Compete on innovative functions with increased design flexibility





Autosar Architecture

Application Software (ASW)

Run Time Environment (RTE)

Base Software (BSW)

Microcontroller

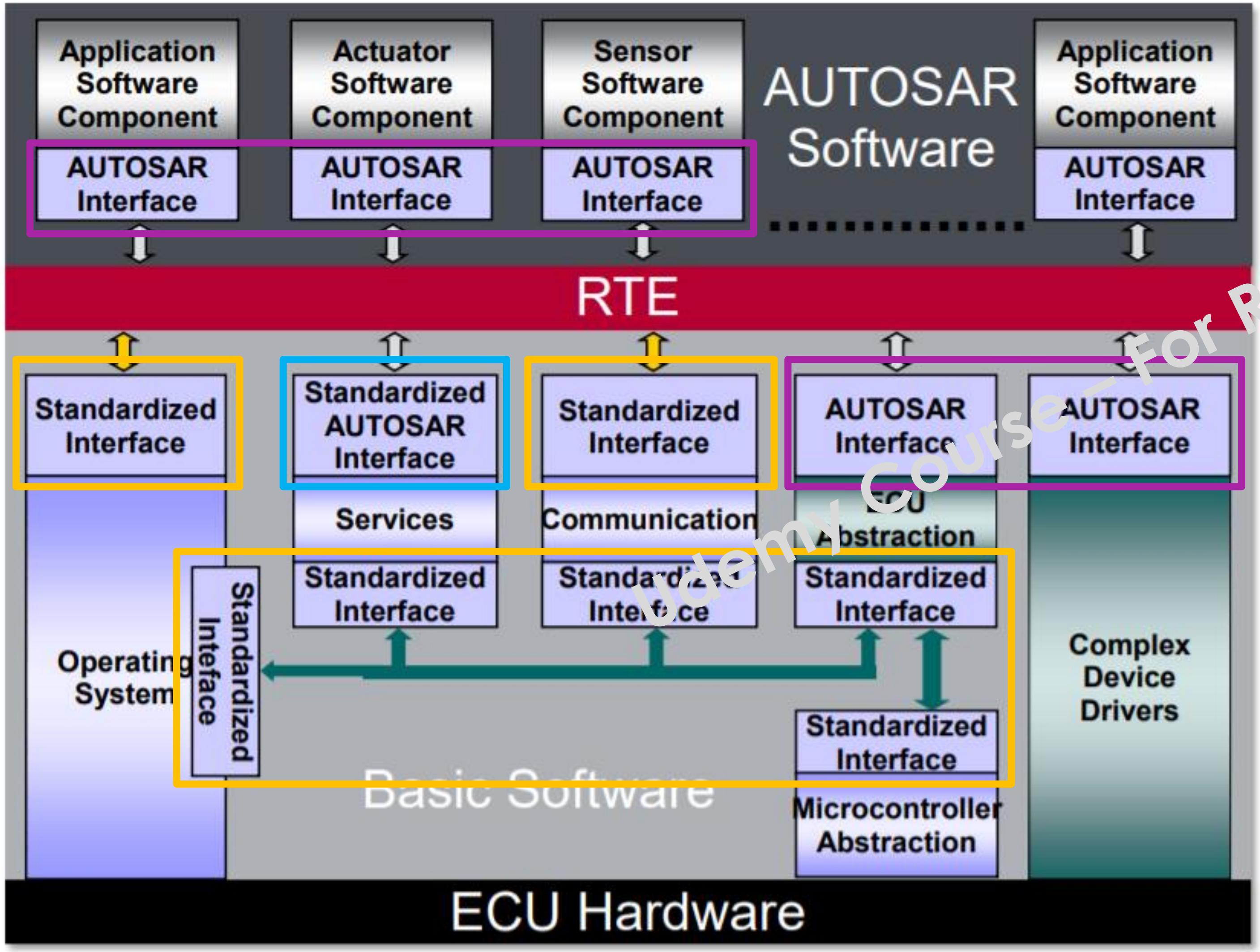
Layered Architecture

- The application software layer is mostly hardware independent
- Communication between software components and access to BSW via RTE
- The BSW is divided in three major layers and complex drivers:
  - Services layer
  - ECU abstraction layer
  - Microcontroller abstraction layer

Key Take Aways:

- **ASW** (Application Software)
  - Functional Modules (application software) that communicate with RTE
- **RTE** (Run Time Environment)
  - Middleware which realizes the communication between the software components and basic software
- **BSW** (Base Software)
  - Standardized software modules that are necessary for the functioning of the higher software layers

Autosar Interfaces



Autosar Interfaces:

- Derived Interface API's from input configurations like Ports etc., that are specified as per the Autosar standards
- Used for communication between ASWs or Complex Device Drivers (CDD)

Example: *Rte_read*, Rte_Write**

Standardized Autosar Interfaces:

- Interfaces that are specially predefined by Autosar standards as API's in C language
- Used as services between ASW and BSW modules like ECU State or Diagnostics Manager

Standardized Interfaces:

- Interfaces that are predefined by Autosar standards as API's in C language
- Used between BSW modules or RTE / BSW / OS

Example: "*DIO_ReadChannel*" API is defined by Autosar to Read an IO pin by other BSW layers

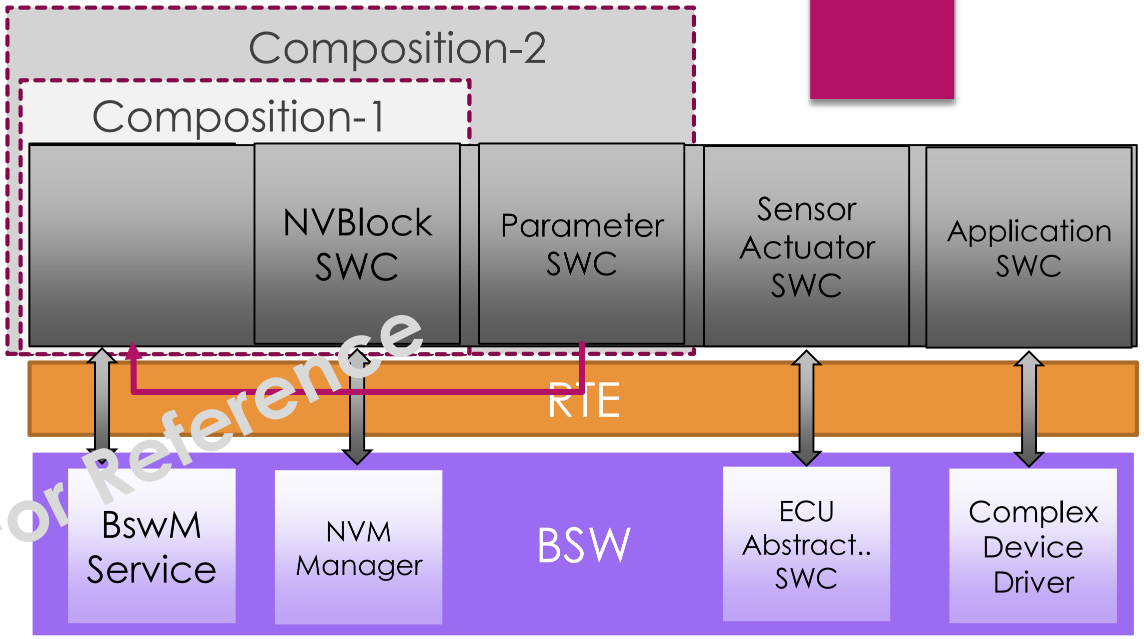
Software Component and Compositions

Software Components

- Application software within Autosar is organized in self contained units called Atomic Software Component types.
- Such Atomic Software components together form the complete functional implementation of the software

Software components takes one of the below types

- ApplicationSwComponent
- NVBlockSwComponent
- ComplexDeviceDriverSwComponent
- ServiceSwComponent
- ServiceProxySwComponent
- EcuAbstractionSwComponent
- SensorActuatorSwComponent
- ParameterSwComponent
- CompositionSwComponent

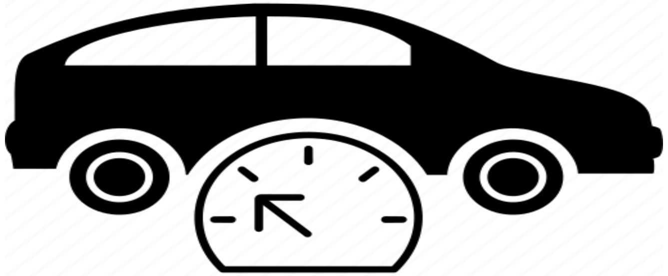


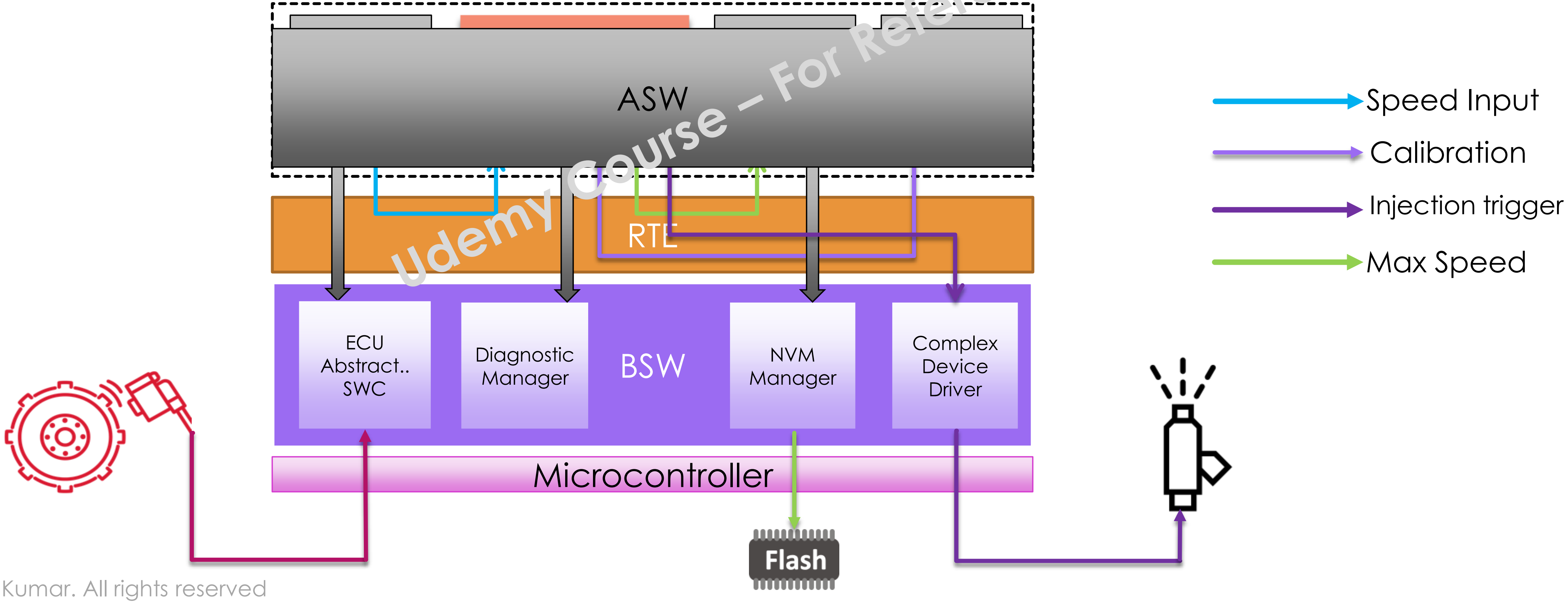
- **Application** software component holds the functionality of the software. Example: Calculations, Functional/decision making Algorithms etc..
- **NVBlock** component is used when we have interfaces on the application layer to be stored on NVM memory. It interacts with the NVRAM Manager
- **CDD** component provides an easy access to hardware directly from application layer to fulfil special timings and functional requirements
- **Service** component is used for configuring services for a particular control unit
- **Service proxy** component is used when a particular service component is to be accessed from different control units
- **EcuAbstraction** component is a part of BSW, which acts as an interface between MCAL and SensorActuator component on the ASW
- **SensorActuator** component is used on the application layer to interact with the BSW ECUAbstraction layer, and acts as a interface to the other application compoents
- **Parameter** component is a part of software component types that provide only calibrations to the software. Example: Tuning vehicle performance using parameters during testing's
- **Composition** aggregates components and connections between its sub Components

Software Component types

Functional Requirement:

- Get speed from the speed sensor
- If the speed is more than a calibrated set point then
  - Cut off injection immediately
  - Log diagnostics error in-case this condition happens
- Save the maximum speed reached to NVM memory

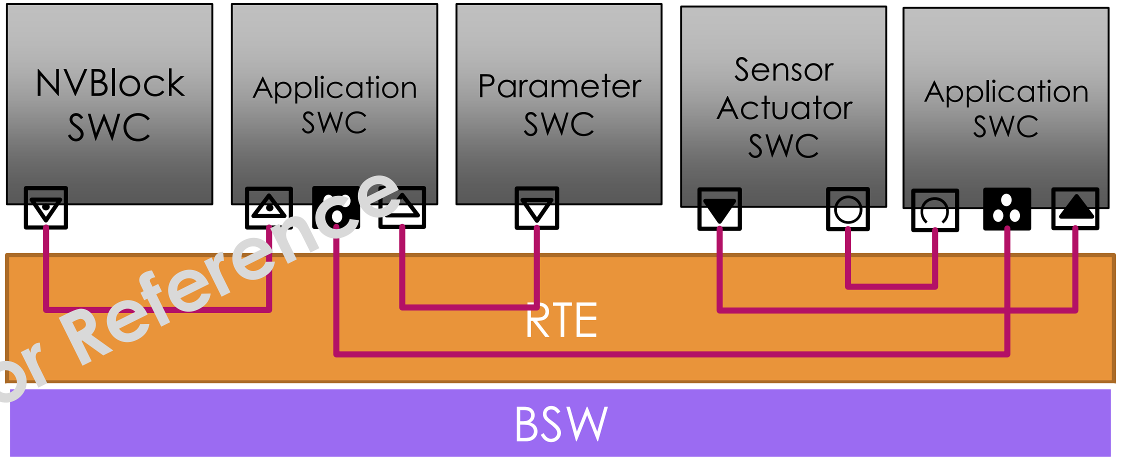




Autosar Ports and Port Interfaces

Ports: Autosar architecture proposes Ports as the mode of communication between Autosar modules

- Provider Port (P-Port)
- Receiver Port (R-Port)
- ProviderReceiver Port (PR-Port)



Port Interfaces: The kind of information that are communicated between ports are defined by port interfaces

- Sender Receiver Interface **P-Port** **R-Port**
- Client Server Interface **P-Port** **R-Port**
- NVData Interface **P-Port** **R-Port**
- Parameter Interface **P-Port** **R-Port**
- ModeSwitch Interface **P-Port** **R-Port**
- Trigger Interface **P-Port** **R-Port**

SR Interface is used to communicate interfaces between components Port writing the interface is the Provider and receiving end is the Receiver

CS Interface is used to call services or functions from another components Component owning the service is the server and the caller is the Client

NVData Interface is to communicate with NVBlock SWC to send and receive non volatile memory interface

Parameter interface is used for exchanging calibrations or constants across components

Mode switch interface is used for notification of a software component of different states that the system can enter

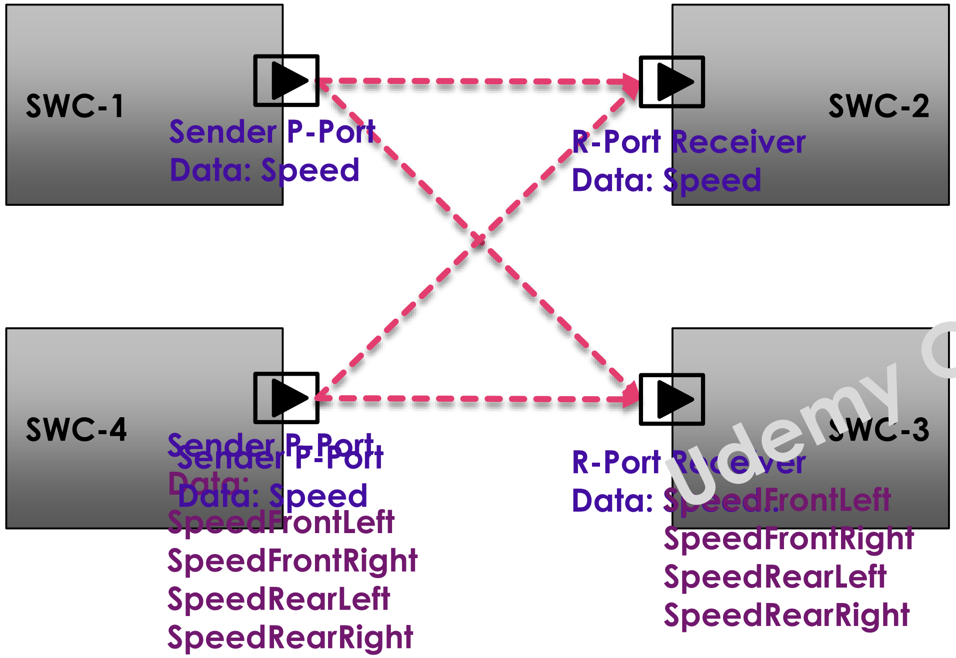
Trigger interface induces as a trigger execution for other components

Sender Receiver Interface

Sender Receiver interface is used to send or receive data between software components

Its an asynchronous communication

- 1:1 communication
- 1:N communication
- M:1 communication
- M:N communication




```xml
<SENDER-RECEIVER-INTERFACE>
  <SHORT-NAME>IF_Speed</SHORT-
  <DATA-ELEMENTS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>Speed</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/DataTypes/uint16</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/DataTypes/uint16</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
```


Provider Port configuration


```xml
<P-PORT-PROTOTYPE>
  <SHORT-NAME>PP_Speed</SHORT-NAME>
  <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/PortInterfaces/IF_Speed</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```


Receiver Port configuration

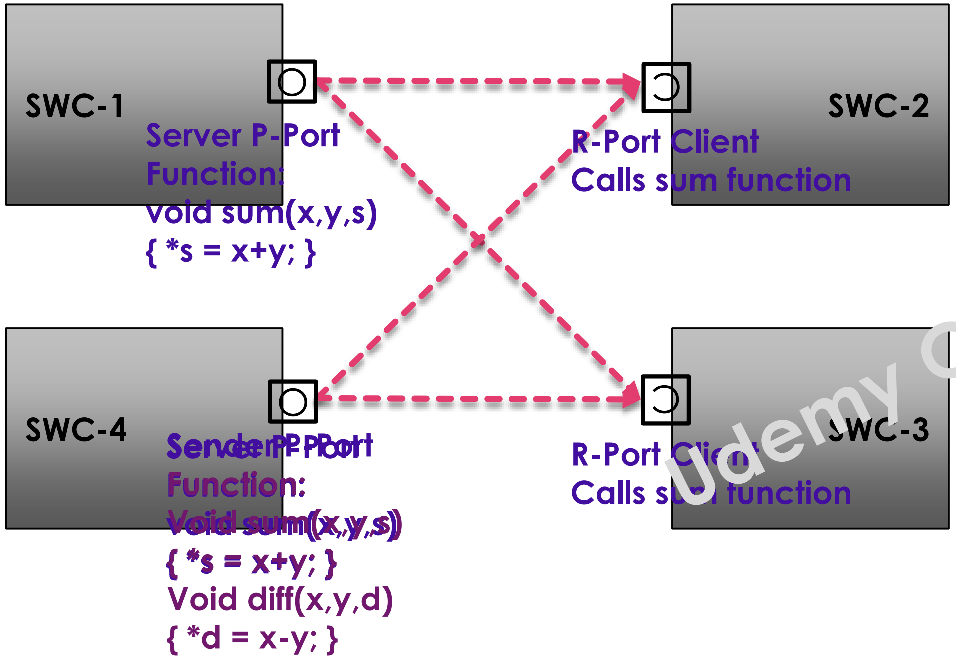

```xml
<R-PORT-PROTOTYPE>
  <SHORT-NAME>RP_Speed</SHORT-NAME>
  <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/PortInterfaces/IF_Speed</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Client Server Interface

Client server interface is used for function calls. Server is the Provider that has the P-Port and Client is the receiver who has the R-Port

- Synchronous call (Client waits until server function runs and completes)
- Asynchronous call (Client triggers server function and just proceeds. Server results are later fetch when needed)

- 1:1 communication ✓
- 1:N communication ✓
- M:1 communication ⃠
- M:N communication ⃠




```xml
<CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>IF_Math</SHORT-NAME>
    <OPERATIONS>
        <CLIENT-SERVER-OPERATION>
            <SHORT-NAME>Sum</SHORT-NAME>
            <ARGUMENTS>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>x</SHORT-NAME>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>y</SHORT-NAME>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>s</SHORT-NAME>
                    <DIRECTION>OUT</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
            </ARGUMENTS>
            <POSSIBLE-ERROR-REFS>
                <POSSIBLE-ERROR-REF DEST="APPLICATION-ERROR">/PortInterfaces/IF_Math/E_NOTOK</POSSIBLE-ERROR-REF>
            </POSSIBLE-ERROR-REFS>
        </CLIENT-SERVER-OPERATION>
    </OPERATIONS>
    <POSSIBLE-ERRORS>
        <APPLICATION-ERROR>
            <SHORT-NAME>E_NOTOK</SHORT-NAME>
            <ERROR-CODE>0</ERROR-CODE>
        </APPLICATION-ERROR>
    </POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
```


Provider Port configuration


```xml
<P-PORT-PROTOTYPE>
    <SHORT-NAME>PP_ServerMath</SHORT-NAME>
    <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/PortInterfaces/IF_Math</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```

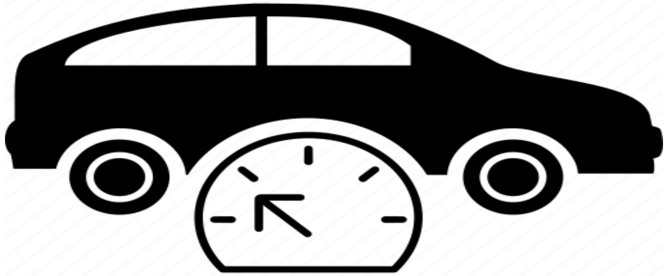

Receiver Port configuration

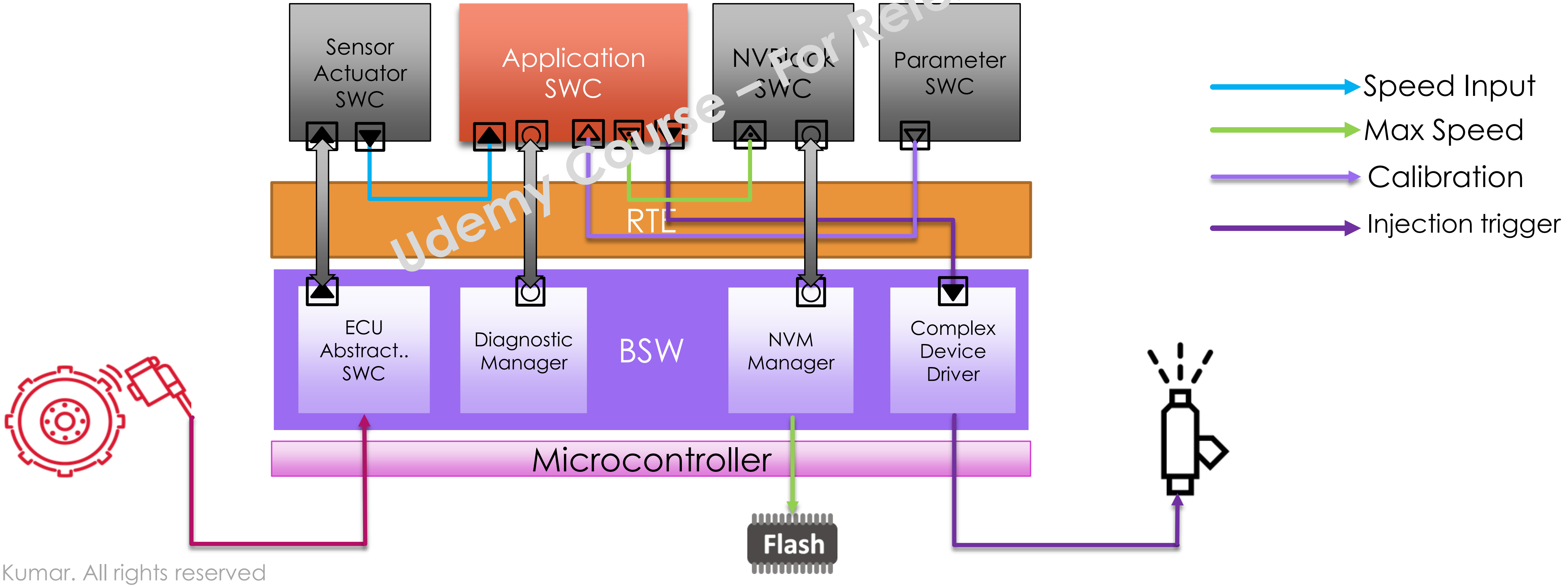

```xml
<R-PORT-PROTOTYPE>
    <SHORT-NAME>RP_ClientMath</SHORT-NAME>
    <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/PortInterfaces/IF_Math</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

Autosar Ports and Port Interfaces

Functional Requirement:

- Get speed from the speed sensor
- If the speed is more than a calibrated set point then
  - Cut off injection immediately
  - Log diagnostics error in-case this condition happens
- Save the maximum speed reached to NVM memory

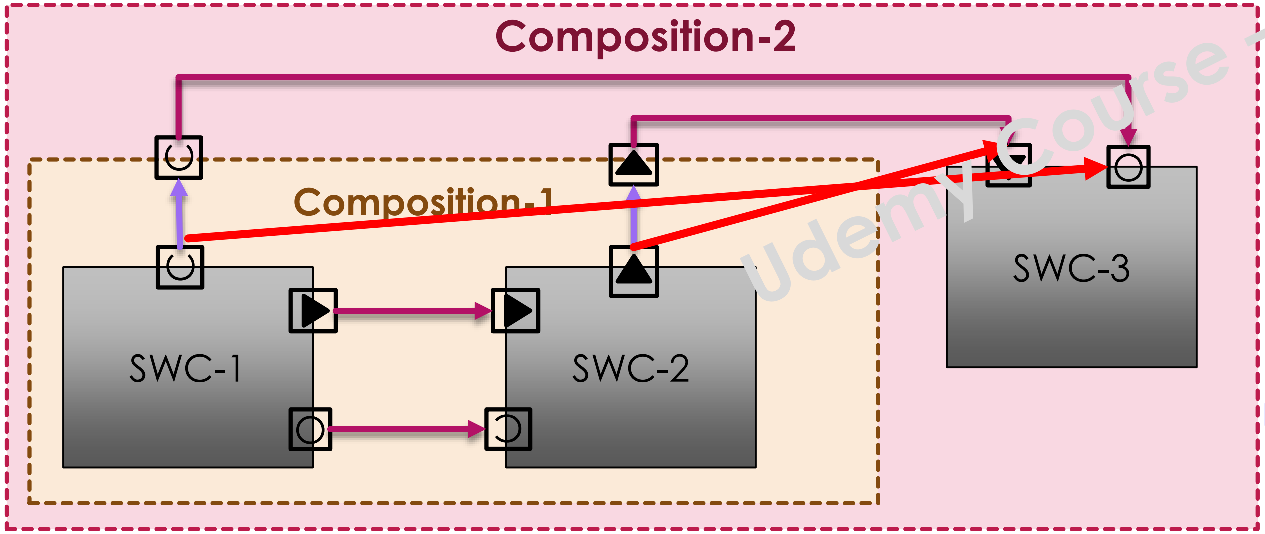




Compositions and Connectors

Compositions:
A software component type that aggregates software components or compositions

Connectors:
Used to complete the connections between port prototypes
- Assembly Connector [Connects a provider and a receiver port]
- Delegation Connector [Connects the same port types to delegate ports to outer composition]
- Pass through connector



Assembly connector
Delegation connector

Runnables

Runnable Entities are the smallest code fragments that are provide by the component

- Runnable Entities together with Events are scheduled by the operating system
- An Atomic Software component has to provide an entry point to code for each runnable in its internal behaviour
- Runnables can be defined only for atomic software components. Composition software component or Parameter software component cannot have a runnable

```xml
<RUNNABLE-ENTITY>
    <SHORT-NAME>Runnable_Sum</SHORT-NAME>
    <DATA-READ-ACCESSS>
        <VARIABLE-ACCESS>
            <SHORT-NAME>DatRead_X</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                    <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/SwComponents/SWC_1/RP_X</PORT-PROTOTYPE-REF>
                    <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE">/PortInterfaces/IF_X/X</TARGET-DATA-PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
        <VARIABLE-ACCESS>
            <SHORT-NAME>DatRead_Y</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                    <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/SwComponents/SWC_1/RP_Y</PORT-PROTOTYPE-REF>
                    <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE">/PortInterfaces/IF_Y/Y</TARGET-DATA-PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </DATA-READ-ACCESSS>
    <DATA-WRITE-ACCESSS>
        <VARIABLE-ACCESS>
            <SHORT-NAME>DataWrite_S</SHORT-NAME>
            <ACCESSED-VARIABLE>
                <AUTOSAR-VARIABLE-IREF>
                    <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">/SwComponents/SWC_1/PP_S</PORT-PROTOTYPE-REF>
                    <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE">/PortInterfaces/IF_S/S</TARGET-DATA-PROTOTYPE-REF>
                </AUTOSAR-VARIABLE-IREF>
            </ACCESSED-VARIABLE>
        </VARIABLE-ACCESS>
    </DATA-WRITE-ACCESSS>
    <SYMBOL>Sum</SYMBOL>
</RUNNABLE-ENTITY>
```

```c
#inlcude "Rte_SWC_1.h"

void Sum()
{
    int X,Y;

    Rte_Read_RP_X_X(&X);
    Rte_Read_RP_Y_Y(&Y);
    Rte_Write_PP_Sum_Sum(X+Y);
}

void Diff(int x,int y, int* d)
{
    *d = x-y;
}

void Mul(int x,int y, long* m)
{
    *m = x*y;
}

void Div(int x,int y, float* d)
{
    *d = x/y;
}
```

Runnable Properties

Sender Receiver or NV Interface	Associated Port
Data Read Access (Implicit)	Receiver Port
Data Write Access (Implicit)	Provider Port
Data Receive point by Value	Receiver Port
Data Receive point by Argument	Receiver Port
Data Send Point	Provider Port

Local Interfaces	Associated Port
Read Local variable	No Port
Write Local variable	No Port

Client Server Interface	Associated Port
Asynchronous server call result point	Receiver Port
Synchronous server call point	Receiver Port

Parameter Interface	Associated Port
Parameter Access	Receiver Port

Mode Switch Interface	Associated Port
Mode Access Point	Receiver Port
Mode Switch Point	Provider Port

Triggers
External Trigger Point
Internal Trigger Point

Runnable Properties
Can be Invoked Concurrently
Symbol

Events

RTE Events are provided from the Autosar standards to specify the operating system on when and how to call the Runnables

Runnables are mapped to the RTE Events and further the operating system and the RTE layer together ensure that the runnable function is called in the expected manner

General Events:

Init Event

Timing Event

External Trigger Occurred Event

Internal Trigger Occurred Event

Background Event

Client Server Events:

Operation Invoked Events

Asynchronous Server call Result Event

Data Events:

Data Write complete Event

Data Send complete Event

Data Receive Event

Data Receive error event

Mode Events:

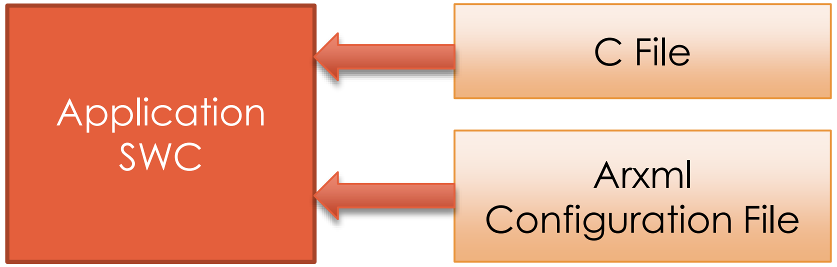
Mode Switch Event

Mode Manager Error Event

Mode Switch Ack Event

```xml
<INTERNAL-BEHAVIORS>
  <SWC-INTERNAL-BEHAVIOR>
    <SHORT-NAME>InternalBehavior</SHORT-NAME>
    <DATA-TYPE-MAPPING-REFS>
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">/Demo/DataTypeMappings/DataTypeMappingSet</DATA-TYPE-MAPPING-REF>
    </DATA-TYPE-MAPPING-REFS>
    <EVENTS>
      <TIMING-EVENT>
        <SHORT-NAME>Event_100ms</SHORT-NAME>
        <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/SwComponents/SWC_1/InternalBehavior/Runnable_Sum</START-ON-EVENT-REF>
        <PERIOD>0.1</PERIOD>
      </TIMING-EVENT>
      <INIT-EVENT>
        <SHORT-NAME>Event_Init</SHORT-NAME>
        <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/SwComponents/SWC_1/InternalBehavior/Runnable_Sum</START-ON-EVENT-REF>
      </INIT-EVENT>
      <DATA-RECEIVED-EVENT>
        <SHORT-NAME>DRE_Event</SHORT-NAME>
        <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/SwComponents/SWC_1/InternalBehavior/Runnable_Mul</START-ON-EVENT-REF>
        <DATA-IREF>
          <CONTEXT-R-PORT-REF DEST="R-PORT-PROTOTYPE">/SwComponents/SWC_1/RP_X</CONTEXT-R-PORT-REF>
        </DATA-IREF>
      </DATA-RECEIVED-EVENT>
      <OPERATION-INVOKED-EVENT>
        <SHORT-NAME>OIE_Event</SHORT-NAME>
        <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/SwComponents/SWC_1/InternalBehavior/Runnable_Div</START-ON-EVENT-REF>
        <OPERATION-IREF>
          <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">/SwComponents/SWC_1/PP_ServerMath</CONTEXT-P-PORT-REF>
          <TARGET-PROVIDED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">/PortInterfaces/IF_Math/Div</TARGET-PROVIDED-OPERATION-REF>
        </OPERATION-IREF>
      </OPERATION-INVOKED-EVENT>
    </EVENTS>
    <HANDLE-TERMINATION-AND-RESTART>NO-SUPPORT</HANDLE-TERMINATION-AND-RESTART>
    <RUNNABLES>
      <RUNNABLE-ENTITY>
        <SHORT-NAME>Runnable_Sum</SHORT-NAME>
        <DATA-READ-ACCESSS>
          <VARIABLE-ACCESS>
            <SHORT-NAME>DatRead_X</SHORT-NAME>
            <ACCESSED-VARIABLE>
              <AUTOSAR-VARIABLE-IREF>
                <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">/SwComponents/SWC_1/RP_X</PORT-PROTOTYPE-REF>
```

Autosar Application Software



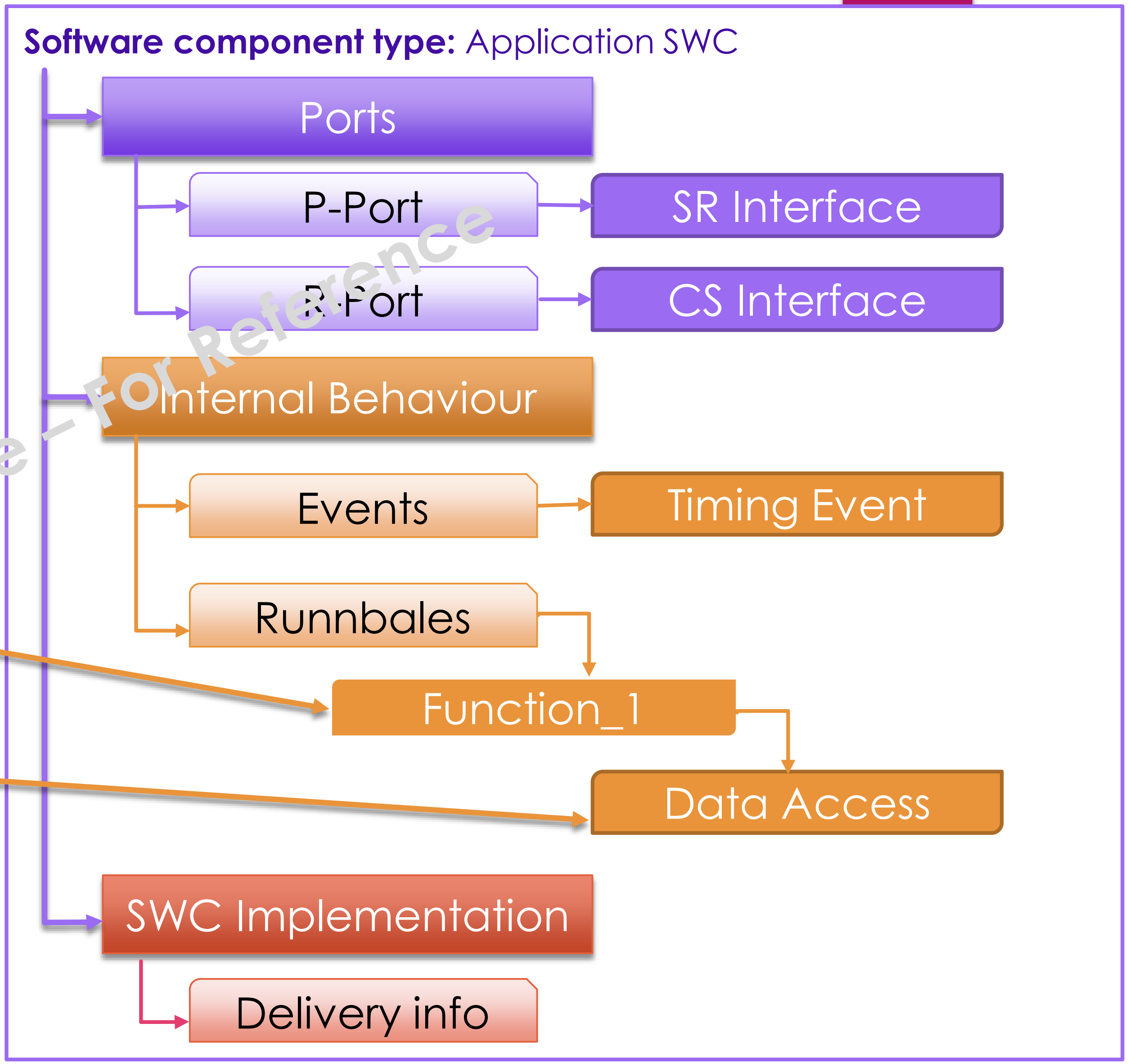
C File

RTE application header

```
#include "Rte_<SWCName>.h"

Void Function_1()
{
        int Speed;

        Speed = distance/time; // Some calculation

        // Send the calculated speed through a port
        Rte_Write_<Pport>_<VDPName>(Speed);

        // Call a service function from another component
        Rte_Call_<Rport>_<CSOperation>();
}
```

API's to access port

arxml File



Specification: AUTOSAR_TPS_SoftwareComponentTemplate.pdf

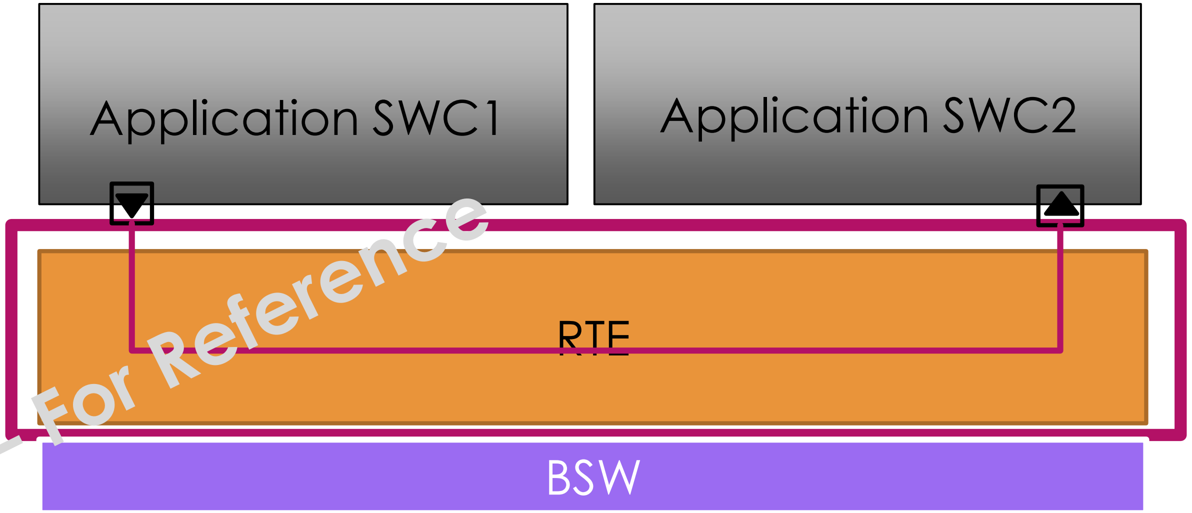
Autosar RTE Layer

The Run-Time Environment (RTE) is the heart of the AUTOSAR architecture

The RTE is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB)

Provides infrastructure for communications to happen between Application Software components and between ASW and BSW modules

Irrespective of SW Components split to different ECUs, RTE is responsible to realize the communications



Main responsibilities of RTE

- Communication between software components
- Message consistency mechanisms
- Scheduling of Runnables from software components

Specification: AUTOSAR_SWS_RTE.pdf

Autosar RTE API's

Specification AR4.4: AUTOSAR_SWS_RTE.pdf
Section- 5.6 API Reference

	Functionality	RTE API's
Sender Receiver Interface	dataReadAccess (Implicit)	Rte_IRead , Rte_IStatus, Rte_IsUpdated
	dataWriteAccess (Implicit)	Rte_IWrite, Rte_IWriteRef, Rte_IInvalidate, Rte_IFeedback
	dataSendPoint (Explicit)	Rte_Write (Non-Queued), Rte_Send (Queued)
	dataReceive- PointByArgument (Explicit)	Rte_Read
	dataReceive- PointByValue (Explicit)	Rte_DRead
Client Server Interface	Clientserver- ServerCallPoint	Rte_Call
	AsynchronousServerCallResultPoint	Rte_Result
Mode Switch Interface	ModeSwitchPoint	Rte_Switch, Rte_SwitchAck
	ModeAccessPoint	Rte_Mode
Parameter Interface	Port - ParameterInterface	Rte_Prm
	ParameterDataPrototype - shared or PerInstance	Rte_CData
	PerInstanceMemory	Rte_Pim
	readLocalVariable - Explicit IRV	Rte_IrvRead
	readLocalVariable - Implicit IRV	Rte_IrvIRead
	writeLocalVariable - Explicit IRV	Rte_IrvWrite
	writeLocalVariable - Implicit IRV	Rte_IrvIWrite
	InternalTriggeringPoint	Rte_IrTrigger
	ExternalTriggeringPoint	Rte_Trigger
	PortAPIOption-indirectAPI	Rte_Port
	ExclusiveArea	Rte_Enter, Rte_Exit

RTE APIs (Sender Receiver Interface)

```xml
<P-PORT-PROTOTYPE>
    <SHORT-NAME>PP_Speed</SHORT-NAME>
    <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/PortInterfaces/IF_Speed</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
<R-PORT-PROTOTYPE>
    <SHORT-NAME>RP_Speed</SHORT-NAME>
    <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/PortInterfaces/IF_Speed</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
<SENDER-RECEIVER-INTERFACE>
    <SHORT-NAME>IF_Speed</SHORT-NAME>
    <DATA-ELEMENTS>
        <VARIABLE-DATA-PROTOTYPE>
            <SHORT-NAME>Speed</SHORT-NAME>
            <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
                <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
                </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
            </SW-DATA-DEF-PROPS>
            <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">/DataTypes/uint16</TYPE-TREF>
        </VARIABLE-DATA-PROTOTYPE>
    </DATA-ELEMENTS>
</SENDER-RECEIVER-INTERFACE>
```

Rte_Write / Rte_Read: (From the standards)

Std_ReturnType Rte_[Byps_]Write/Read_<p>_<o>([IN Rte_Instance <instance>], IN/OUT <data>,[OUT Rte_TransformerError transformerError])

<p> => Name of the Provider/Receiver Port
<o> => Name of the Variable Data prototype
<data> => Data to be written/read to/from RTE

C File

```c
Void Function_1()
{
  /* Write to a P-Port */
  Value = 10;
  Rte_Write_PP_Speed_Speed(Value);
}

Void Function_2()
{
  /* Read from a Receiver Port */
  int Readdata;
  Rte_Read_RP_Speed_Speed (&Readdata);
}
```

RTE APIs (Client Server Interface)

```xml
<P-PORT-PROTOTYPE>
    <SHORT-NAME>PP_ServerMath</SHORT-NAME>
    <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/PortInterfaces/IF_Math</PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
<R-PORT-PROTOTYPE>
    <SHORT-NAME>RP_ClientMath</SHORT-NAME>
    <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">/PortInterfaces/IF_Math</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
<CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>IF_Math</SHORT-NAME>
    <OPERATIONS>
        <CLIENT-SERVER-OPERATION>
            <SHORT-NAME>Sum</SHORT-NAME>
            <ARGUMENTS>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>x</SHORT-NAME>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>y</SHORT-NAME>
                    <DIRECTION>IN</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
                <ARGUMENT-DATA-PROTOTYPE>
                    <SHORT-NAME>s</SHORT-NAME>
                    <DIRECTION>OUT</DIRECTION>
                </ARGUMENT-DATA-PROTOTYPE>
            </ARGUMENTS>
            <POSSIBLE-ERROR-REFS>
                <POSSIBLE-ERROR-REF DEST="APPLICATION-ERROR">/PortInterfaces/IF_Math/E_NOTOK</POSSIBLE-ERROR-REF>
            </POSSIBLE-ERROR-REFS>
```

C File (Client side)

```c
Void Function_1()
{
  int x = 5;     /* Input -1 */
  int y = 10;    /* Input -2 */
  int sum;       /* Output Result */

  /* Client to access a Server P-Port */
  Rte_Call_RP_ClientMath_Sum(x, y, &sum);
}
```

Rte_Call: (From the standards)

Std_ReturnType Rte_[Byps_]Call_<p>_<o>([IN Rte_Instance <instance>],
[IN | IN/OUT | OUT] <data_1>...
[IN | IN/OUT | OUT] <data_n>, [OUT Rte_TransformerError transformerError])

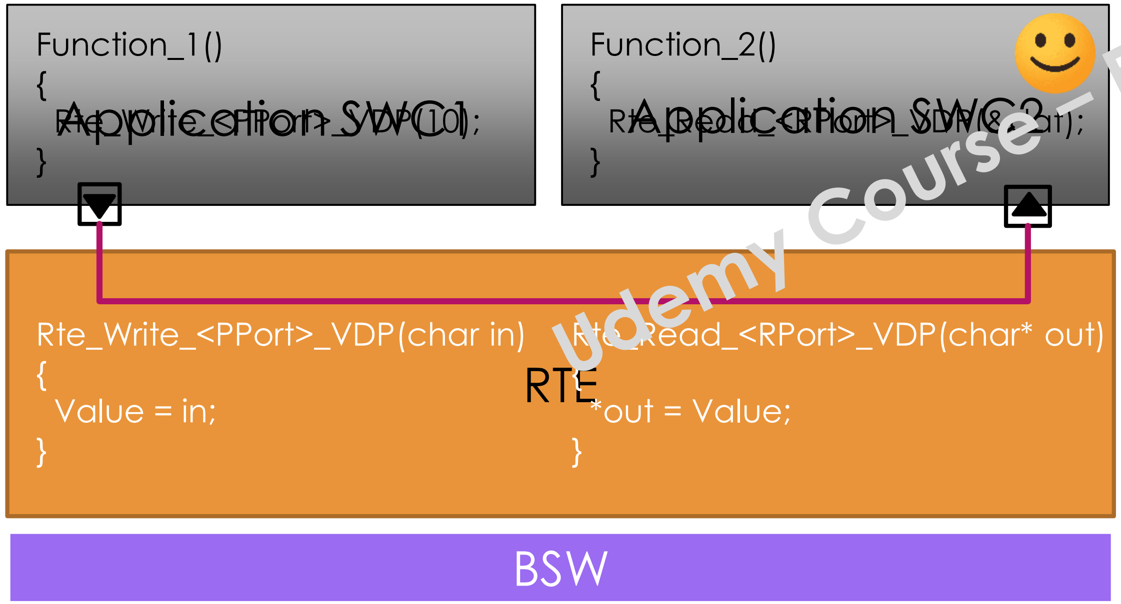
<p> => Name of the Provider Port
<o> => Name of the Variable Data prototype

RTE Layer Code for this API

```c
Std_ReturnType  Rte_Call_RP_ClientMath_Sum(x, y, &sum)
{
        sum(x,y,&sum);  /* RTE calls the server runnable */
        return(RTE_E_OK);
}
```

Autosar RTE Layer - Communication

- **Easy Handling**
- **Abstraction**
- **Reusability**
- **Fast To Market**
- **Competition**

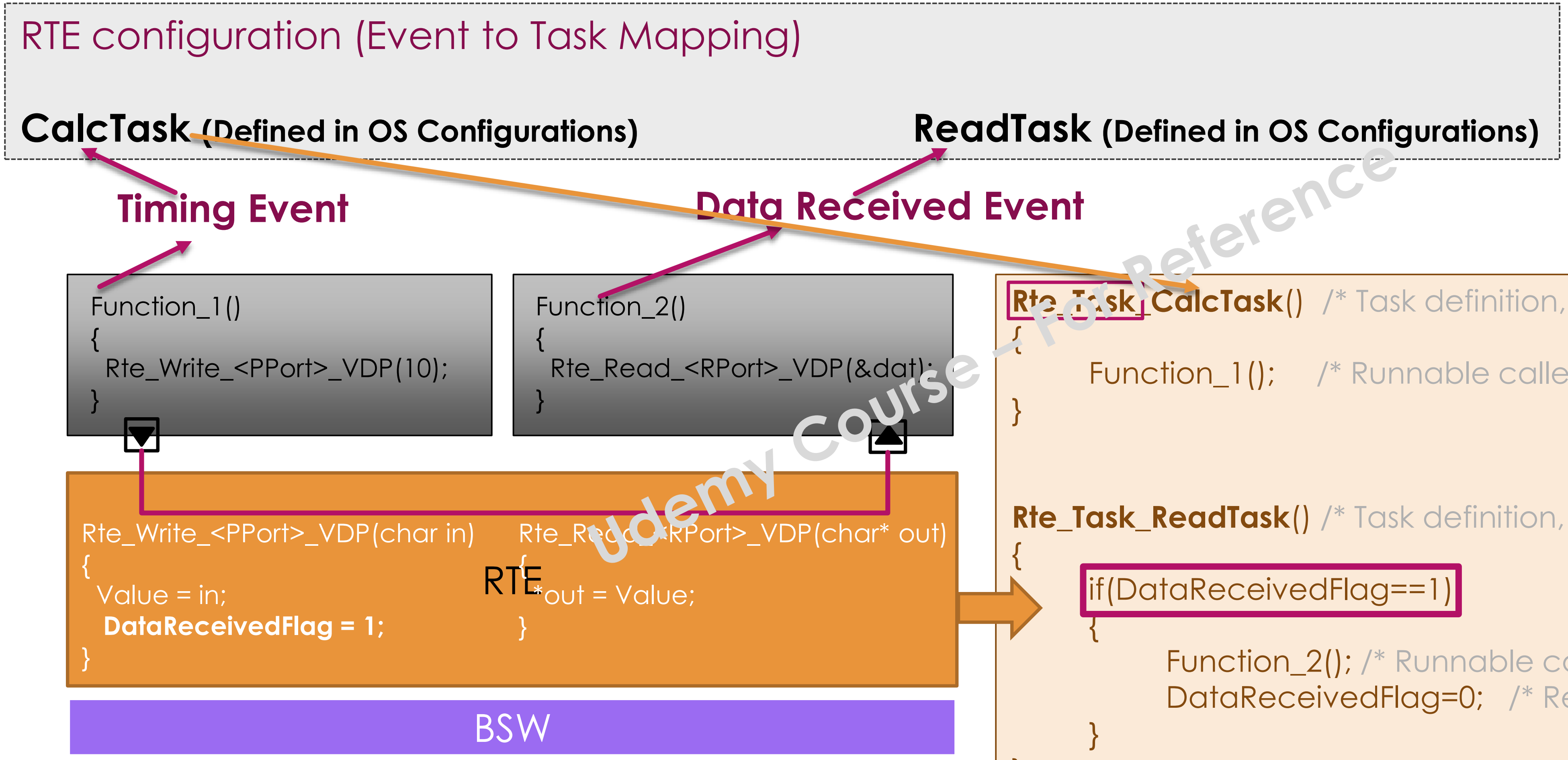


New Vendor
Application SWC-2

```
Function_2()
{
 Rte_Read_<RP>_VDP(&dat);
}
```

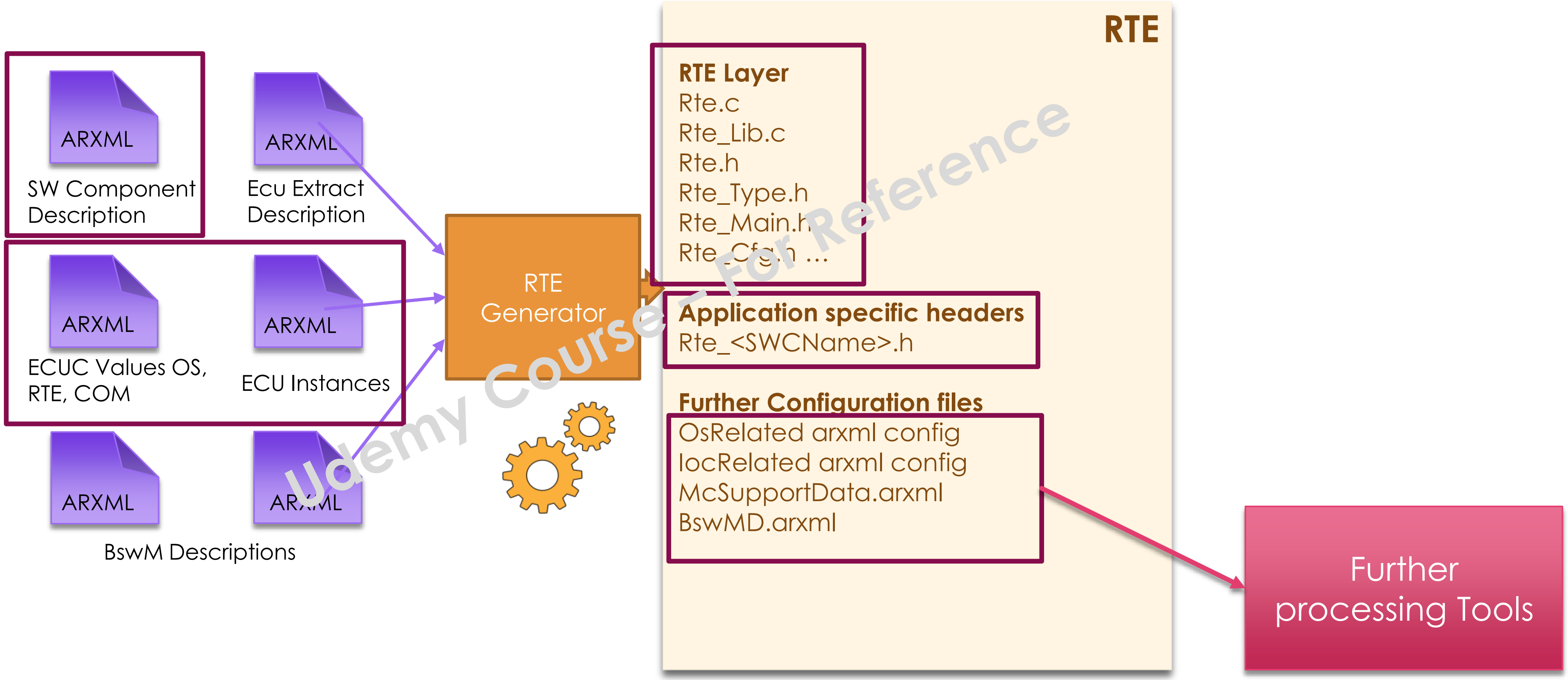
```
Rte_Read_<RP>_VDP(char* out)
{
 *out = Value;
}
```

Autosar RTE Layer - Scheduling




```
Rte_Task_CalcTask()  /* Task definition, called periodically */
{
        Function_1();     /* Runnable called periodically */
}

Rte_Task_ReadTask() /* Task definition, called periodically */
{
        if(DataReceivedFlag==1)
        {
                Function_2(); /* Runnable called when data received */
                DataReceivedFlag=0;   /* Reset flag*/
        }
}
```

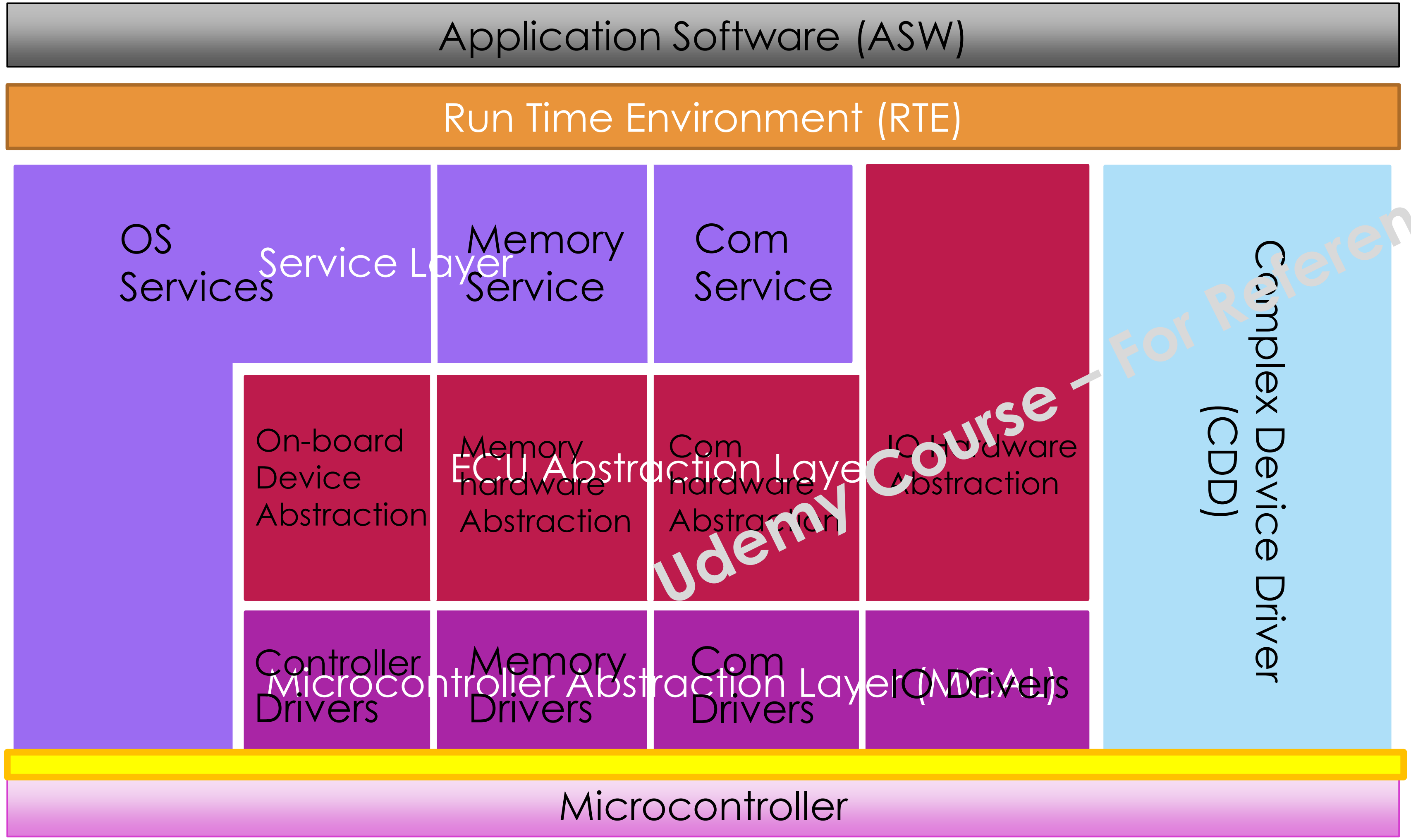
Autosar RTE Generator



Autosar Tools

Implementer	BSW/MCAL Implementation	BSW Configurator	RTE Generator	ASW Development	License
Vector Informatik GmbH	MICROSAR	DaVinci Configurator Pro	MICROSAR Rte Generator	PREEvision DaVinci Developer	Commercial
Elektrobit	EB Tresos AutoCore	EB Tresos Studio	EB Tresos studio		Commercial
ETAS			RTA	ISOLAR-A	Commercial
dSPACE			SystemDesk RTE Generator	SystemDesk	Commercial
Dassault Systems		GCE	RTEG	AAT	Commercial
KPIT Technologies Ltd.	K-SAR Suite	K-SAR Editor	Yes	K-SAR Editor	Commercial

Autosar BSW Layers (Overview)



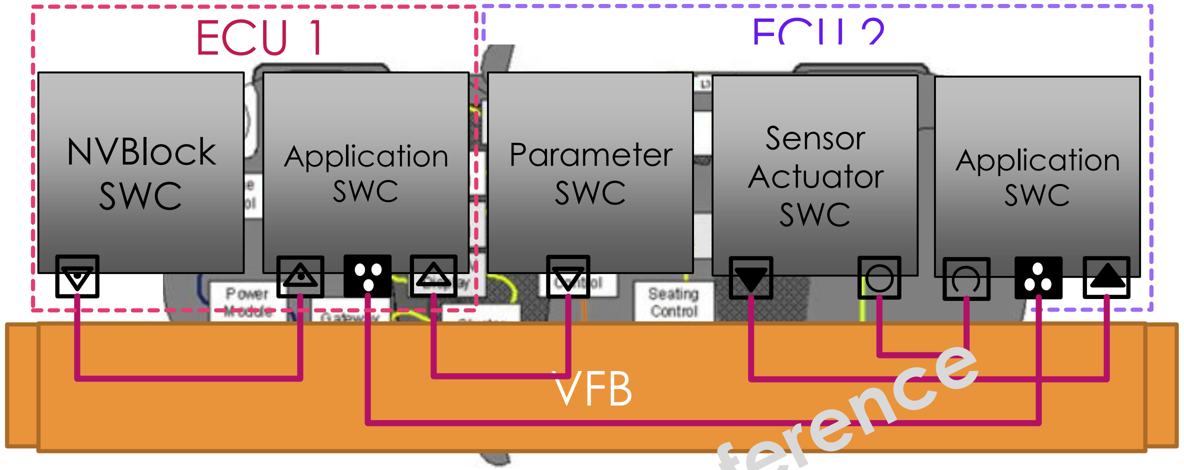
Microcontroller Abstraction Layer
- Lowest Software layer in BSW
- Access direct microcontroller internal peripherals
- Makes higher software layers independent of microcontroller

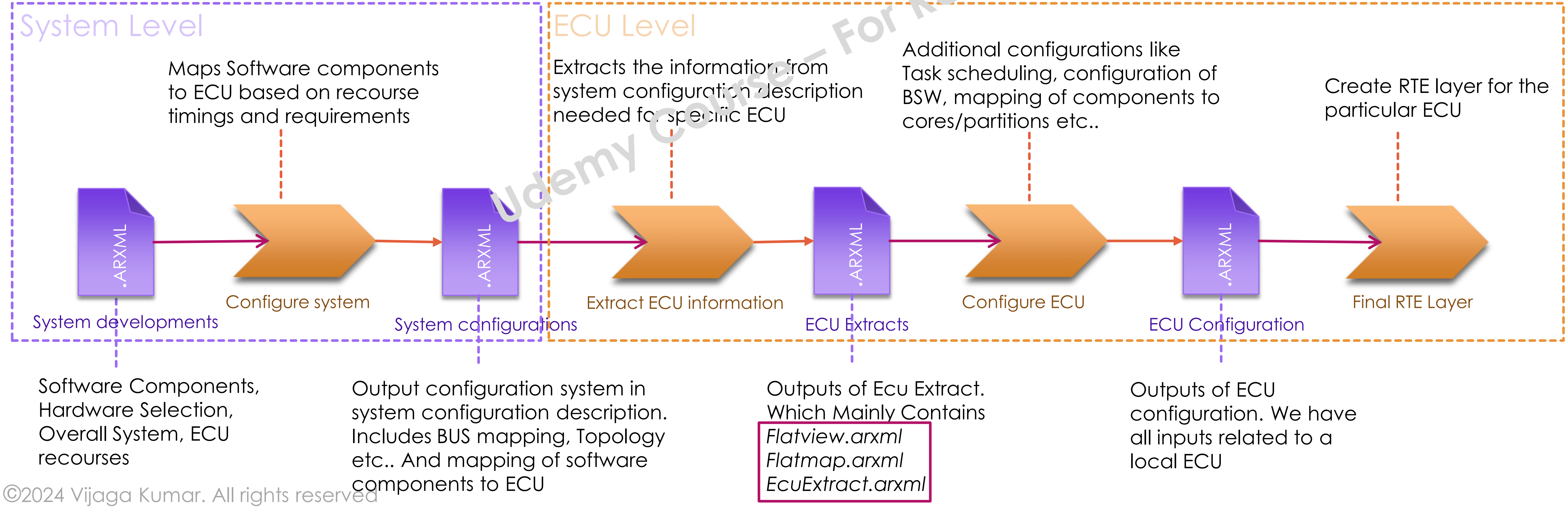
ECU Abstraction Layer
- Offers APIs to access internal or external peripherals
- Makes higher software layers independent of ECU Hardware

Service Layer
- Provides basic services to Application and RTE layers
- Includes OS, Communication, Memory, Diagnostic services

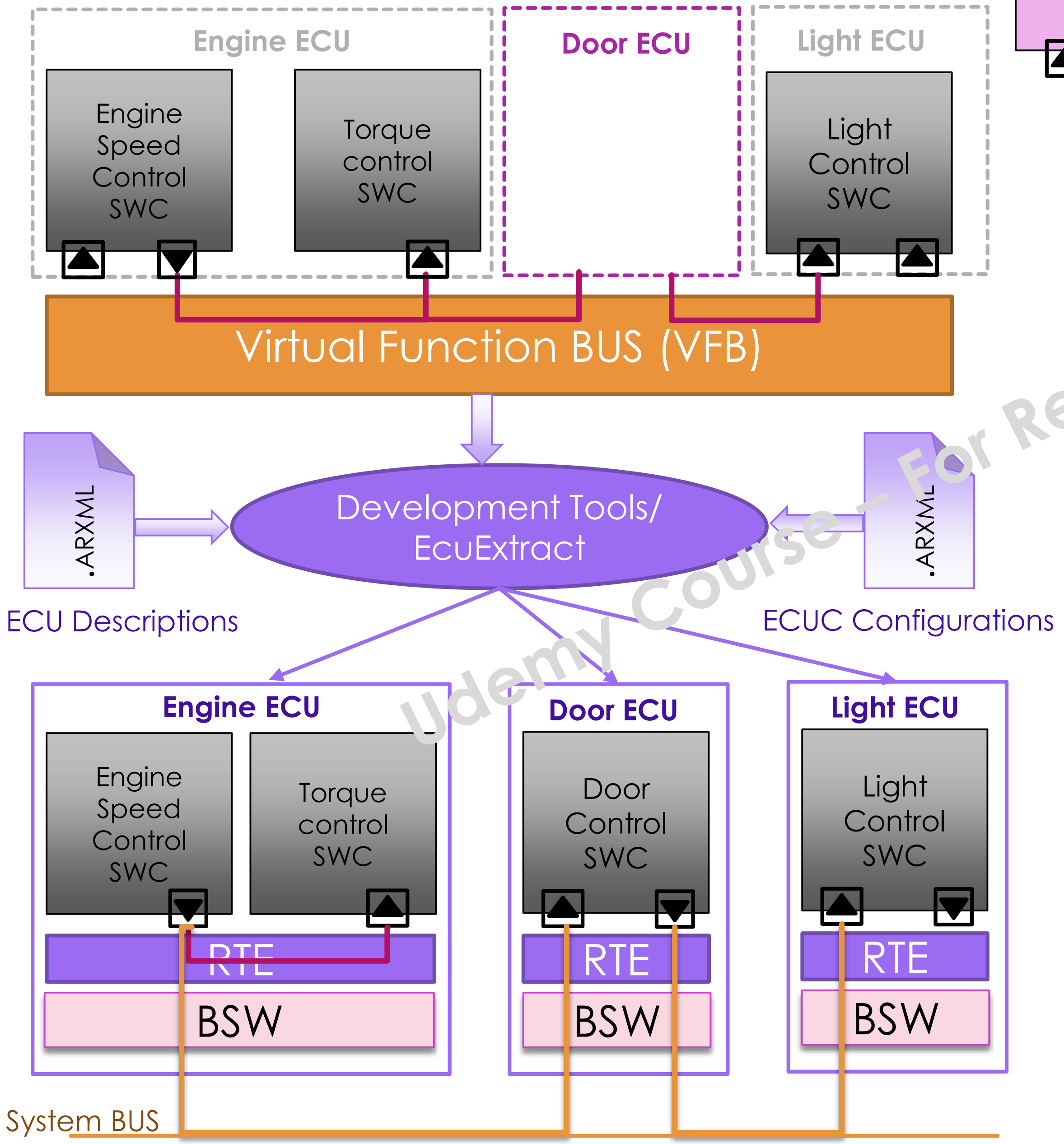
Complex Device Driver
- Direct interaction- Hardware to RTE
- Used for high time constraint applications
- Cases which are not specified by Autosar

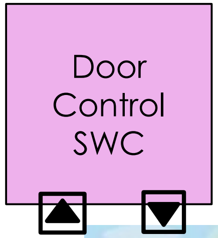
Autosar Methodology





Autosar Methodology





- Car door should unlock only when the vehicle speed is 0
- After successful opening of the door, the parking lights should turn ON

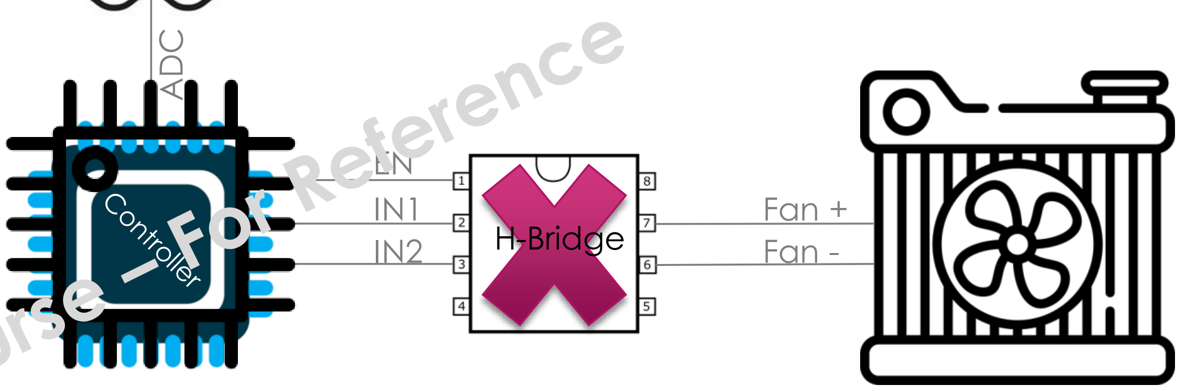
- Developments are done on a system level and the communications between software components are realized on VFB level
- VFB is Virtual Function Bus used in the development phase to enable integration of software components and realize the communication between them irrespective of the target ECUs they are placed
- A system can have any number of ECUs. Individual ECU contents are extracted in a process called ECUExtract for the particular target ECUs
- ECUExtract is done using specialized tools which takes other configuration inputs as well like target ECU description, ECU configurations etc.
- RTE takes care of communication between Software components through the configured communication channels, irrespective of on which ECUs they are placed

Practical Use-Case (Classic Software)

Typical Non-Autosar Software

```
Void ApplicationFunction()
{
    Codefor: ADC Pin Initialize
    Codefor: IO Pin Initialize
    Codefor: H-Bridge IC Initialize

    while(1)
    {
        Codefor: StartADCConversion

        Result = ReadADCValues;
        Temperature = Result * 0.5; // Raw to °C

        if(Temperature >= 50 )
        {
            IN1_PIN = 1; // Switch ON Motor
            IN2_PIN = 0;
        }
        else
        {
            IN1_PIN = 0; // Switch OFF Motor
            IN2_PIN = 0;
        }
    }
}
```

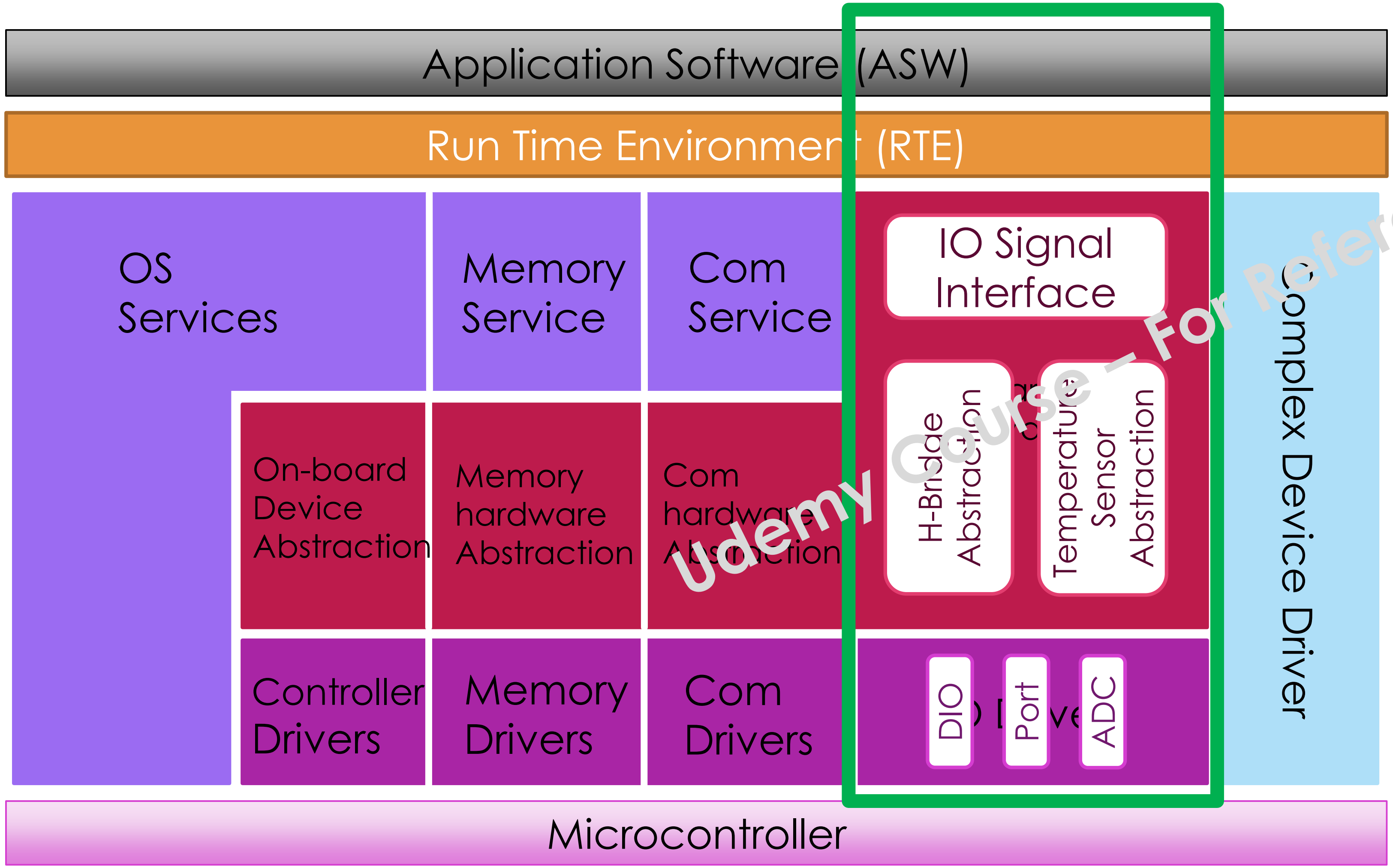
UseCase:
If Temperature >= 50°C [*Fan ON*]
Other cases [*Fan OFF*]



Conclusion (Non-Autosar Software):

- Application Software and hardware are tightly coupled
- Complete Software has to be changed in-case of any hardware changes
- Cost for software development is too huge
- Software not structured or reusable with different hardware's
- Customers (OEMs) become too much dependant to the suppliers for a long term supply of software and hardware during production

Cooling Fan Use-Case (Autosar Software)



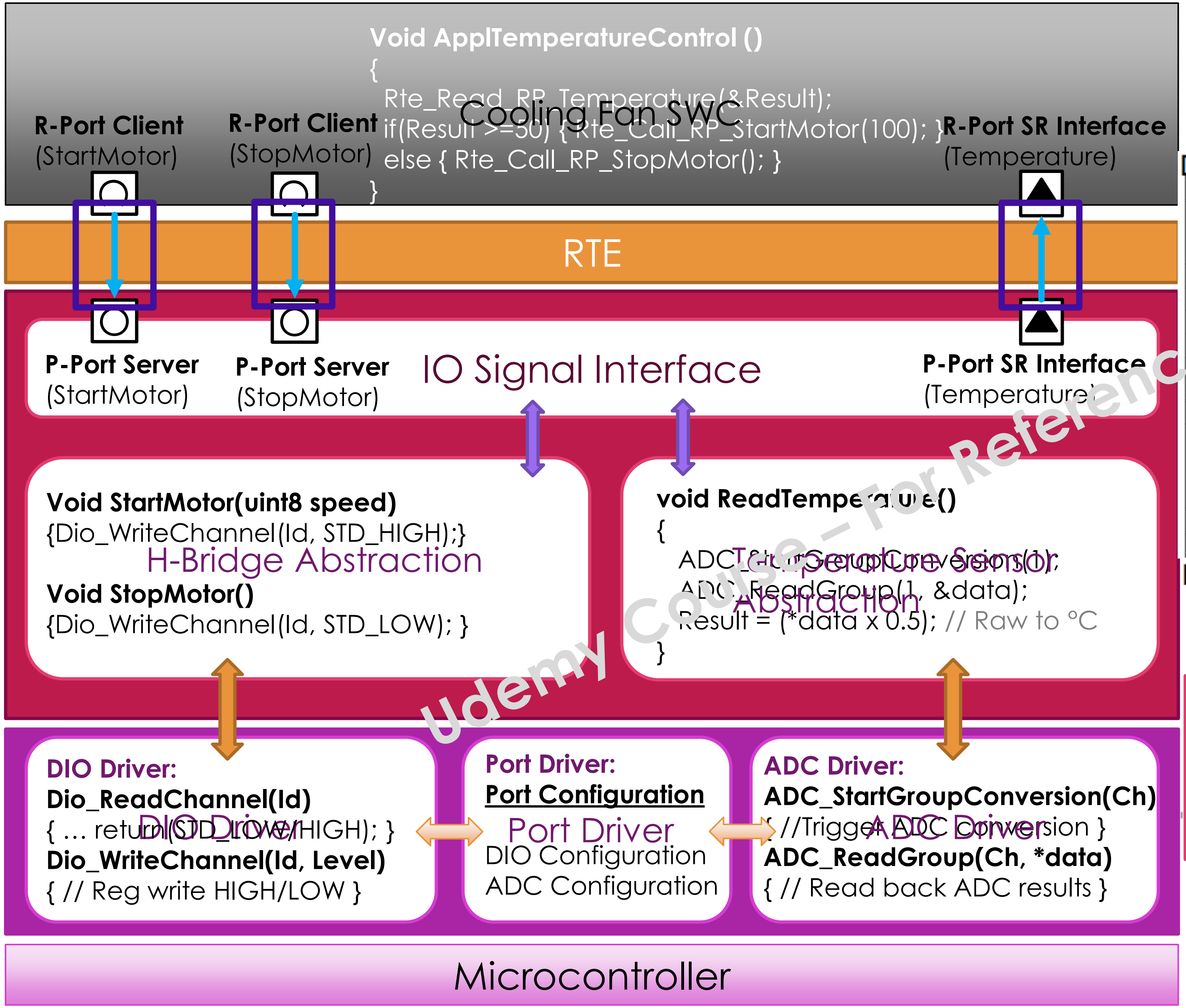
Requirements layer wise:

Application Software: (SWC)
- Control Fan based on Temperature

Drivers: (IO Abstraction)
- H-Bridge Driver
- Temperature Sensor Driver

Hardware Pins: (IO Driver)
- 3 Digital IO Ports
- 1 ADC Port

Cooling Fan Use-Case (Autosar Software)



[SWS_Adc_00367]

IO Abstraction:

- Acts as an interface layer between ASW and BSW
- It fakes Ecu Abstraction Software Component type

RTE API's (Generated)

- Acts as an interface layer between RTE and MCAL

```
/* Called from BSW to write temperature to RTE */
Std_ReturnType Rte_Write_PP_Temperature_Temperature(Result)
{
Temperature = Result; return (RTE_E_OK);
}
/* Called from ASW to receive temperature from RTE */
Std_ReturnType Rte_Read_RP_Temperature_Temperature(float* Result)
{
*Result = Temperature; return (RTE_E_OK);
}
/* Server calls from ASW to actuate Motor */
Std_ReturnType Rte_Call_RP_StartMotor_StartMotor()
{
StartMotor_Function(); return (RTE_E_OK);
}
Std_ReturnType Rte_Call_RP_StopMotor_StopMotor()
{
StopMotor_Function(); return (RTE_E_OK);
}
```

Initialize Temperature

- start/stop service to the application layer
- DIO ports

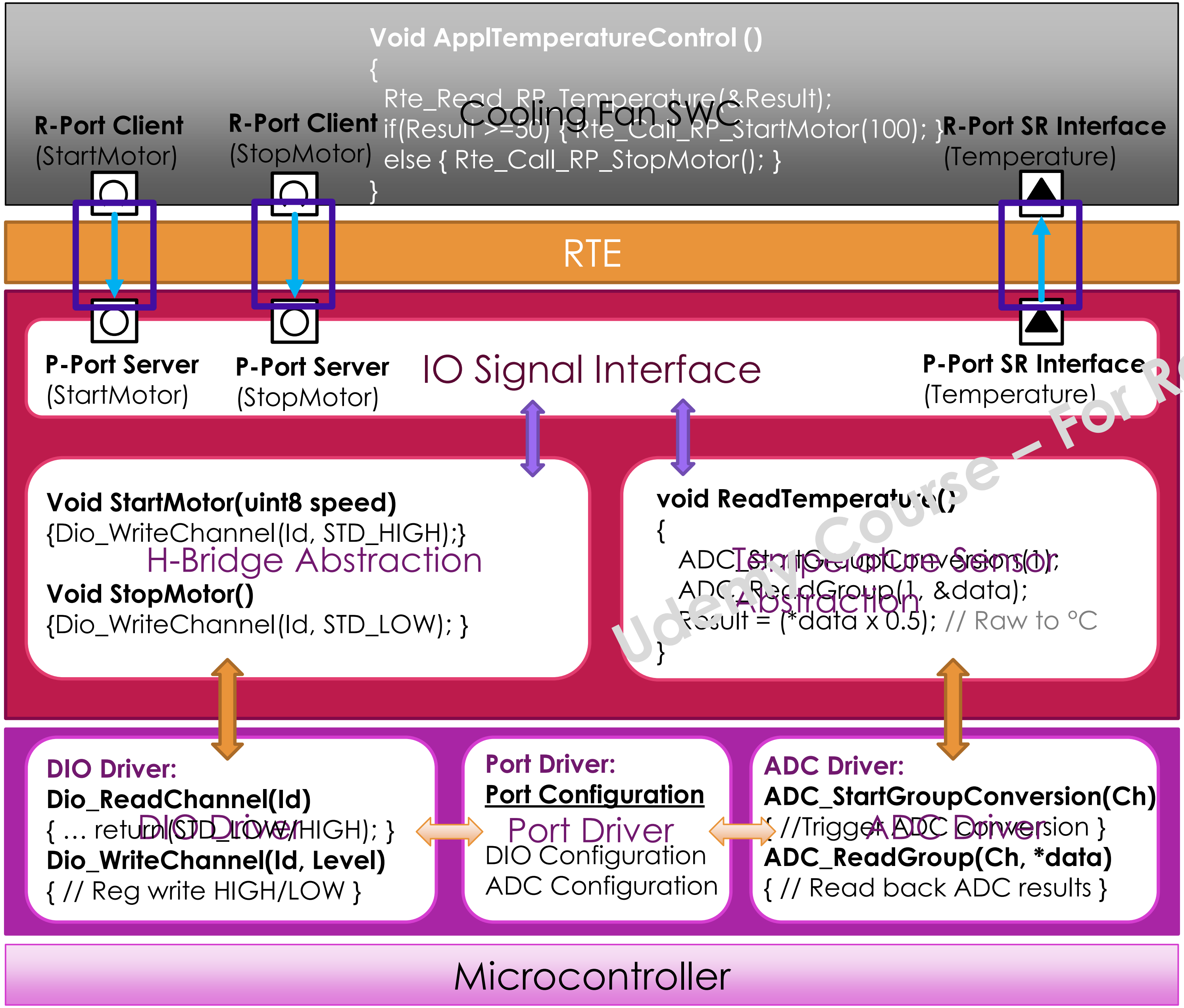
PORT Driver Configuration from AR4.4.0 standards

Parameters	Comments
PortPinId	Set ID for a port pin from (1-65535)
PortPinDirection	Set to Input or Output
PortPinDirectionChangeable	Set to TRUE is port pin direction is changeable during runtime
PortPinLevelValue	Set Port pin HIGH or LOW during initialization
PortPinInitialMode	Initial Port pins as ADC, DIO, SPI, PWM etc..
PortPinMode	Change Port pins as ADC, DIO, SPI, PWM etc..

	PortPinId	PortPinDirection	PortPinInitialMode
ADC Pin	10	PORT_PIN_IN	PORT_PIN_MODE_ADC
DIO_Enable	11	PORT_PIN_OUT	PORT_PIN_MODE_DIO
DIO_IN1	12	PORT_PIN_OUT	PORT_PIN_MODE_DIO
DIO_IN2	13	PORT_PIN_OUT	PORT_PIN_MODE_DIO

Service name:	Dio_ReadChannel
Syntax:	Dio_LevelType Dio_ReadChannel(ChannelId
Sync/Async:	Synchronous
Parameters (out):	None
Return value:	None
Description:	Service to set a level of a channel.
Available via:	Dio.h

Cooling Fan Use-Case (Autosar Software)



RTE

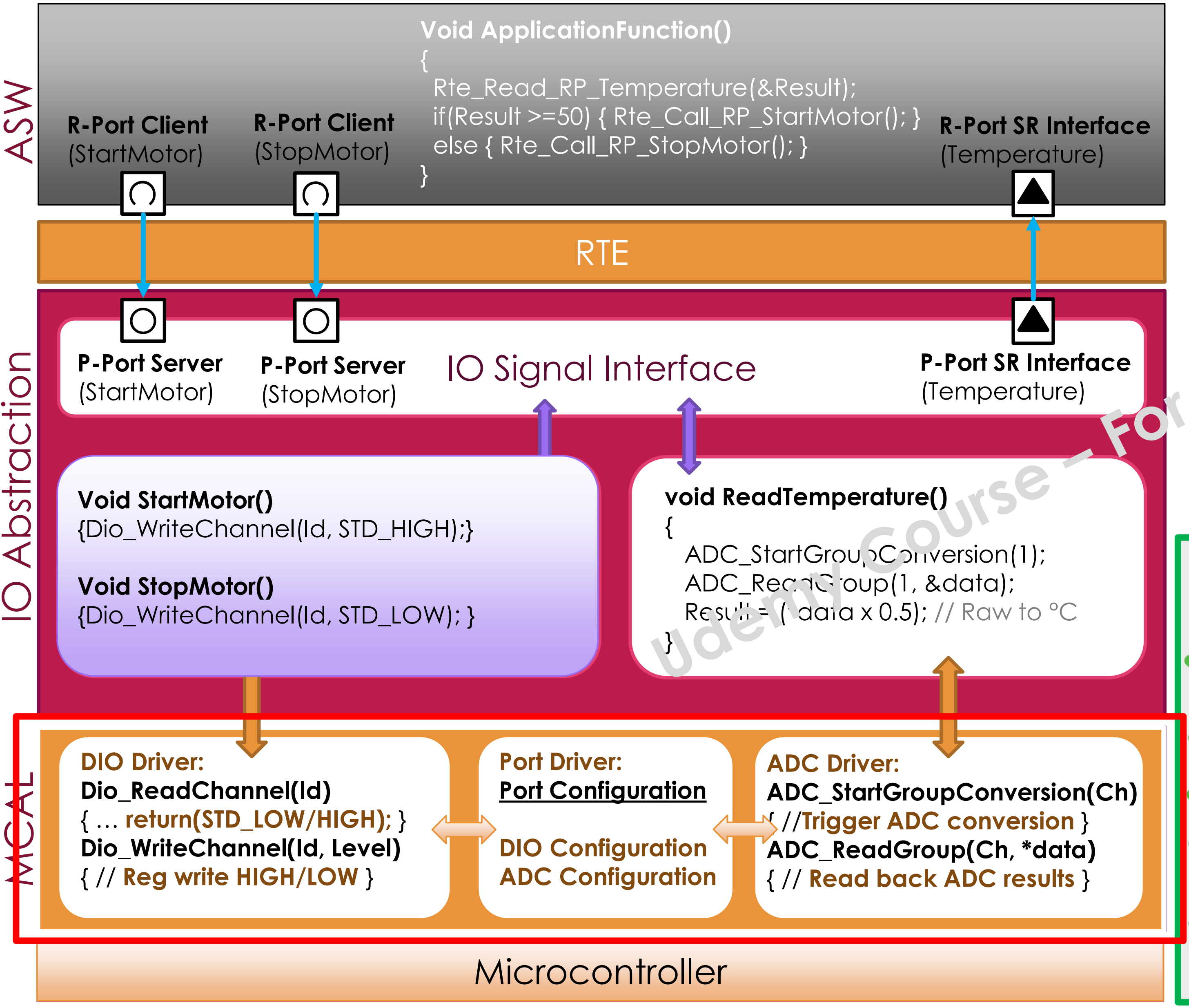
Acts as an interface layer between ASW and BSW

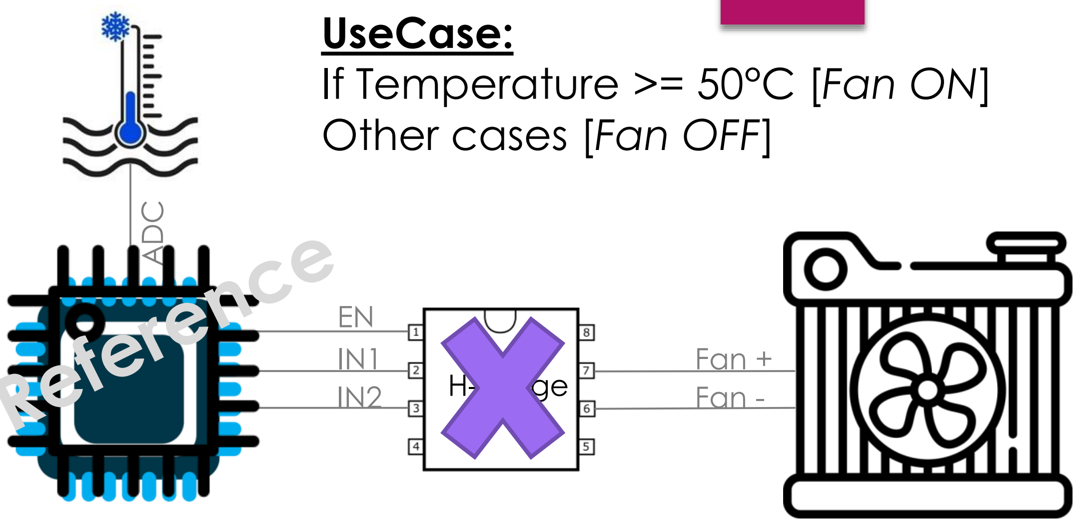
RTE API's (Generated)

```
/* Called from BSW to write temperature to RTE */
Std_ReturnTpe Rte_Write_PP_Temperature_Temperature(Result)
{
   Temperature = Result; return (RTE_E_OK);
}
/* Called from ASW to receive temperature from RTE */
Std_ReturnType Rte_Read_RP_Temperature_Temperature(float* Result)
{
   *Result = Temperature; return (RTE_E_OK);
}
```

```
/* Server calls from ASW to actuate Motor */
Std_ReturnType Rte_Call_RP_StartMotor_StartMotor(uint8 Fanspeed)
{
   StartMotor_Function(FanSpeed); return (RTE_E_OK);
}
Std_ReturnType Rte_Call_RP_StopMotor_StopMotor()
{
   StopMotor_Function(); return (RTE_E_OK);
}
```

Autosar Software (Summary)





UseCase:
If Temperature >= 50°C [*Fan ON*]
Other cases [*Fan OFF*]

Conclusion (Autosar Software):

- **Easy Handling:** Handling Increased complexity of Automotive software
- **Abstraction:** Abstraction of hardware from software, making development more flexible
- **Reusability:** Reuse software modules across Customers
- **Fast To Market:** Establish development distribution among suppliers
- **Competition:** Compete on innovative functions with increased design flexibility

Conclusion
AUTOSAR ARCHITECTURE

Thanks for your participation and hope you had a good learning!.

All the best!.