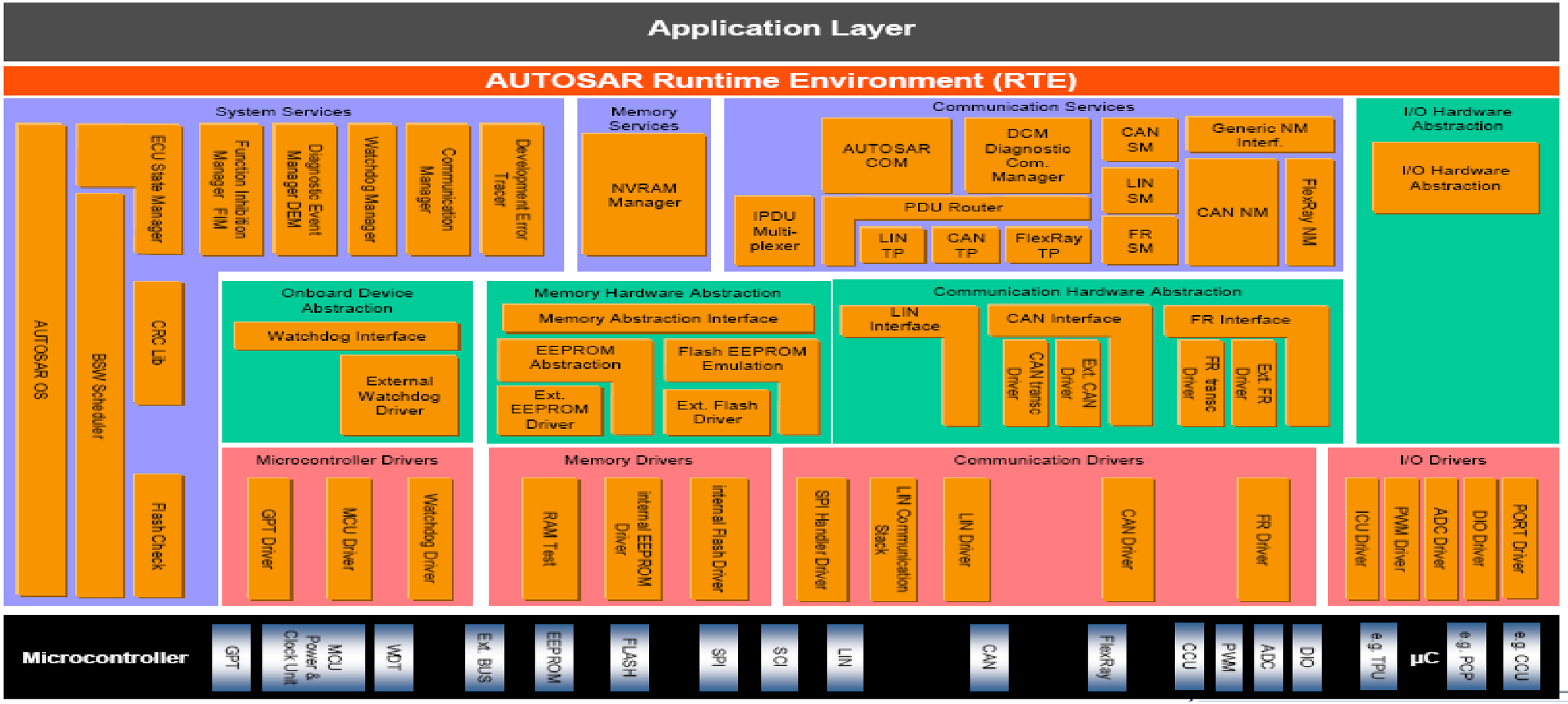
AUTOSAR BSW

KPIT

Introduction to Basic Software

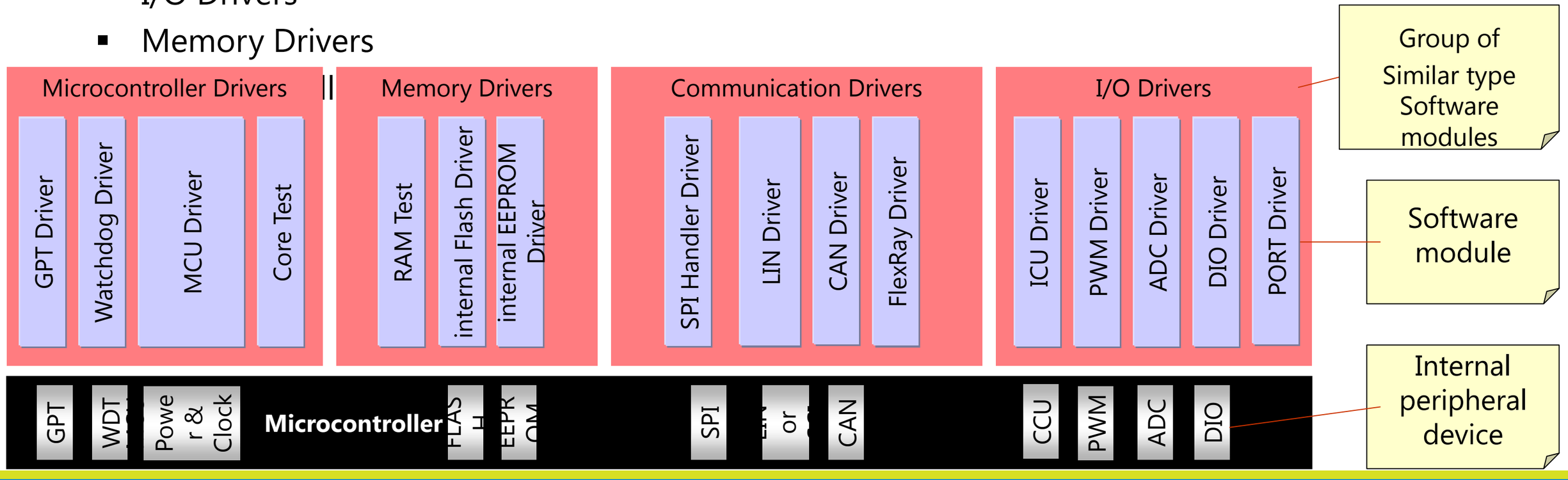
- Basic software can be explained as collaboration of
  - **Services**
    - **Input/Output (I/O)** provides standardized access to sensors, actuators and ECU onboard peripherals
    - **Memory services** provides standardized access to internal/external memory (non volatile memory)
    - **Communication services** provides standardized access to vehicle network systems, ECU onboard communication systems and ECU internal SW
    - **System Services** standardizes operating system, timers, error memory, etc along with ECU specific services (ECU state management, watchdog manager) and library functions
  - **Internal Drivers** contain the functionality to control and access an internal device and are located in the Microcontroller Abstraction Layer (MCAL). E.g. internal EEPROM, internal CAN controller, internal ADC.
  - **External Drivers** contain the functionality to control and access an external device. These drivers are located in the ECU Abstraction Layer and accesses the external device via drivers in MCAL. E.g. external EEPROM, external watchdog, ext. flash
  - **Interface (Interface Modules)** contains the functionality to abstract from modules which are architecturally placed below them. They are located in ECU Abstraction Layer. Example can be an interface for a CAN communication system provides a generic API to access CAN communication networks independent on the number of CAN Controllers within an ECU and independent of the hardware realization (on chip, off chip).
  - **Manager** offers specific services for multiple clients. Generally managers reside in service layer. E.g NVRAM Manager which manages the concurrent access to internal and/or external memory devices like flash and EEPROM memory.

Basic Software - A detailed view



Microcontroller Abstraction Layer

- The **Microcontroller Abstraction Layer** is the lowest software layer of the
- Basic Software
- It contains drivers, which are software modules with direct access to the µC
- internal peripherals and memory mapped µC external devices
  - Communication Drivers
  - I/O Drivers
  - Memory Drivers



Module Groups In MCAL

- **Communication Drivers**
  - Drivers for ECU onboard (e.g. SPI) and vehicle communication (e.g. CAN). OSI-Layer: Part of Data Link Layer
- **I/O Drivers**
  - Drivers for analog and digital I/O (e.g. ADC, PWM, DIO)
- **Memory Drivers**
  - Drivers for on-chip memory devices (e.g. internal Flash, internal EEPROM) and memory mapped external memory devices (e.g. external Flash)
- **Microcontroller Drivers**
  - Drivers for internal peripherals (e.g. Watchdog, General Purpose Timer) Functions with direct µC access (e.g. Core test)

Drivers Overview

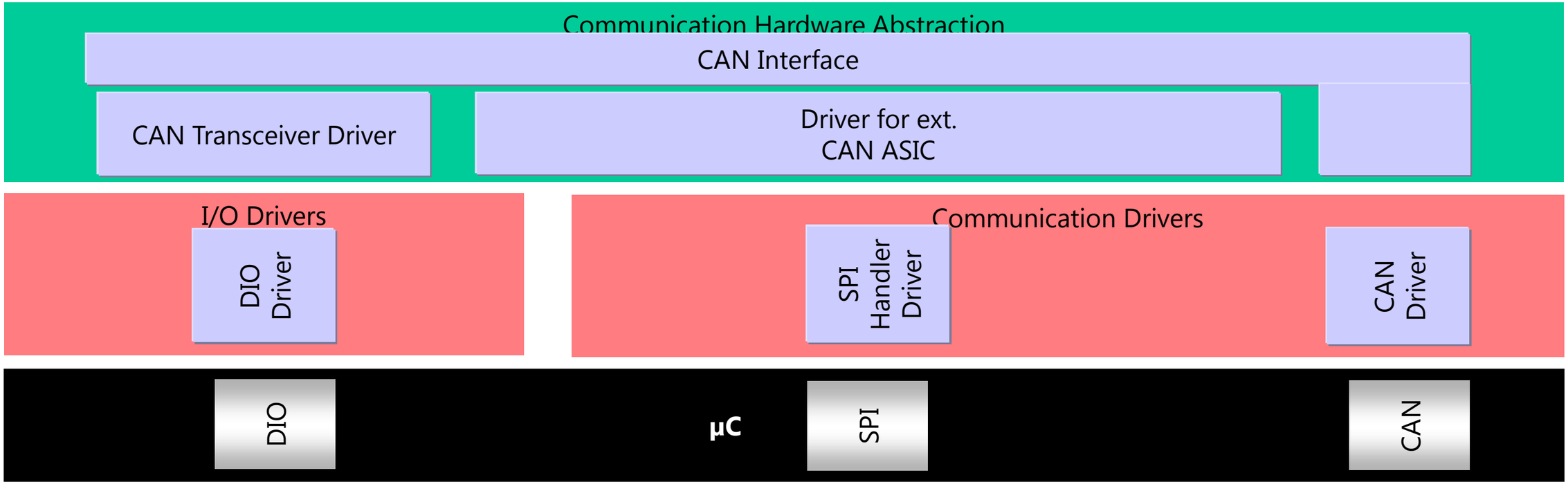
- **I/O Drivers**
- The ADC driver module initializes and controls the internal Analog to Digital Converter unit of the microcontroller
- The Digital Input and Output driver module provides services for reading from and writing to the DIO Channels, DIO Ports and DIO Channel Groups
- The PWM driver module provides functions for initialization and control of the microcontroller internal PWM unit
- The ICU Driver controls the input capture unit of the microcontroller
- Port Driver initializes the whole port structure of the microcontroller
- **Communication Drivers**
- The Communication drivers performs the hardware access and offers a hardware independent API to the upper layer
- **Memory Drivers**
- Memory driver module provides the interface for erasing, writing and reading the memory

Drivers Overview

- **Microcontroller Drivers**
- The GPT driver allows generating one-shot or continuous timer notifications
- The Watchdog driver module provides services for initialization, changing the operation mode and triggering the watchdog
- The MCU driver module provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required from other Low Level Driver modules

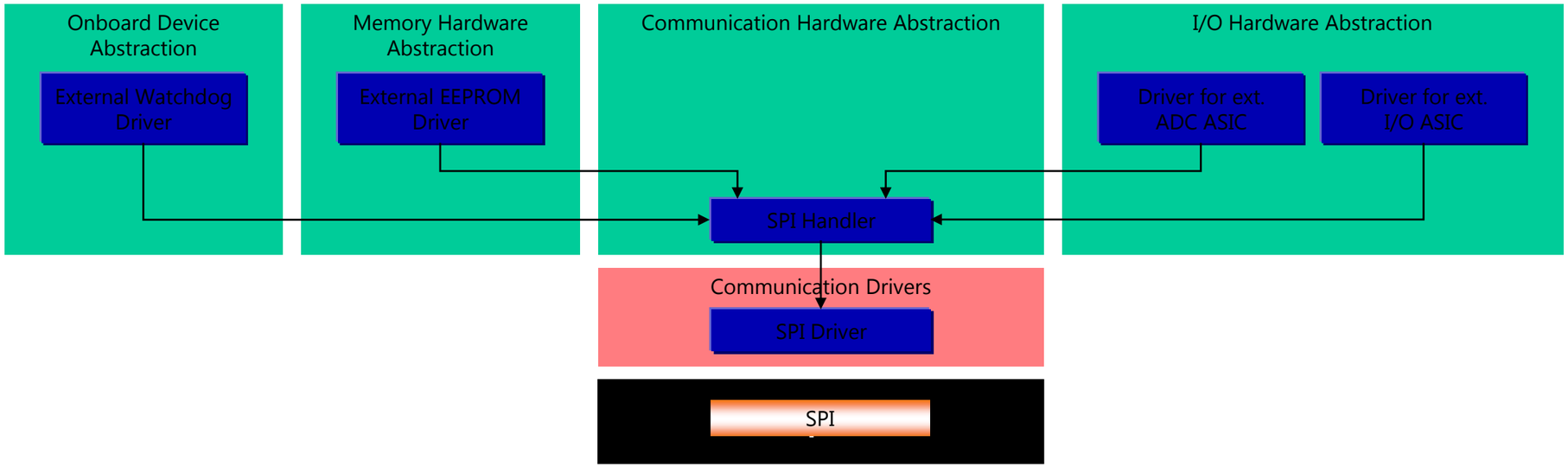
Communication Hardware Abstraction

- A group of modules which abstract from the location of communication controllers
- and the ECU hardware layout.
- For all communication systems a ***specific*** Communication Hardware Abstraction is
- required
  - An ECU has a microcontroller with 2 internal CAN channels and an additional on-board ASIC
  - with 4 CAN controllers. The CAN-ASIC is connected to the microcontroller via SPI.
  - The communication drivers are accessed via bus specific interfaces (e.g. CAN Interface)
- Provides equal mechanisms to access a bus channel regardless of its location (on-chip / on-board)



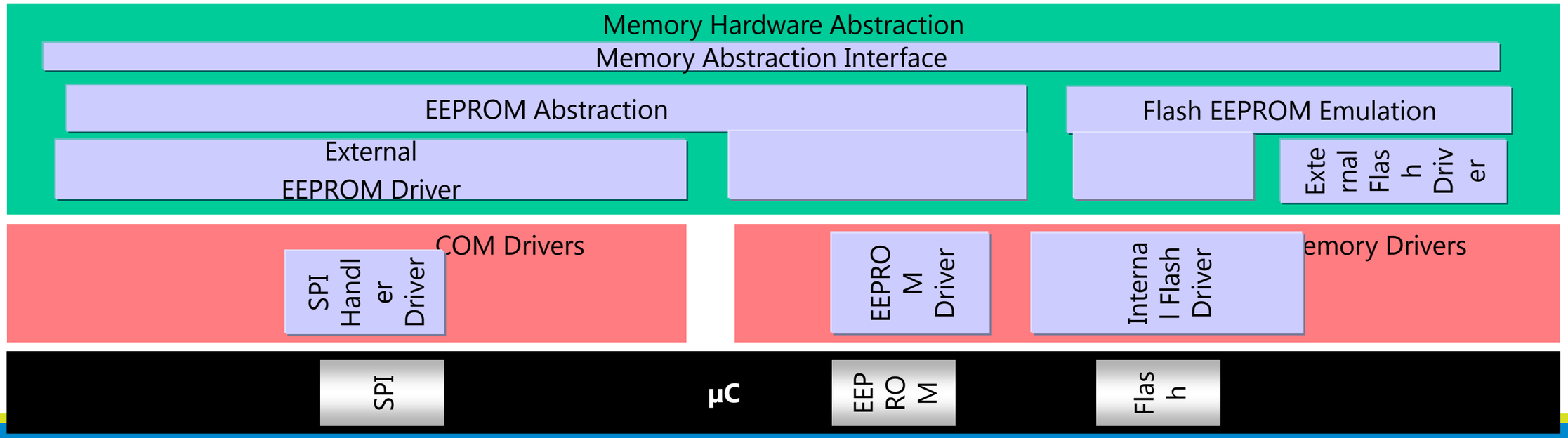
SPI Handler

- SPI Handler module allows concurrent access of several clients to one or more SPI buses
- In many ECUs, a lot of onboard hardware devices like external EEPROM, external I/O ASICs, external watchdogs, System Basis Chip, etc. are connected to the microcontroller via SPI
- Example:



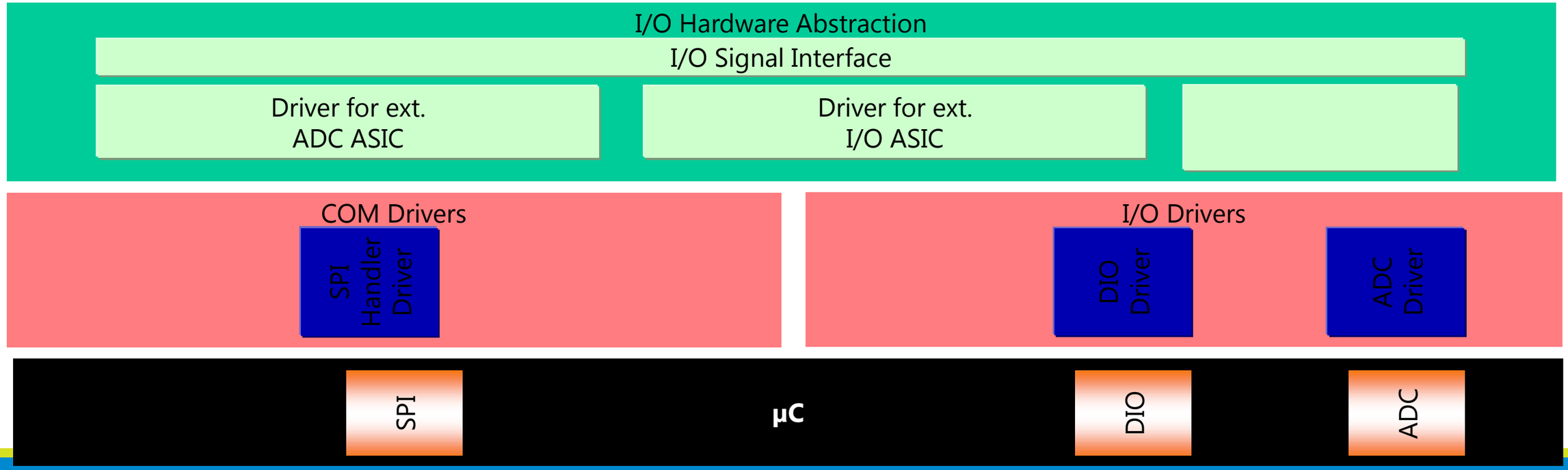
Memory Hardware Abstraction

- A group of modules which abstract from the location of memory devices (on-chip or
- on-board) and the ECU hardware layout
  - on-chip EEPROM and external EEPROM devices should be accessible via
  - an equal mechanism
- The memory drivers are accessed via memory specific abstraction (EEPROM
- Abstraction/Flash EEPROM Emulation) modules
- Provides equal mechanisms to access internal (on-chip) and external (on-
- board) memory devices and type of memory hardware (EEPROM, Flash)



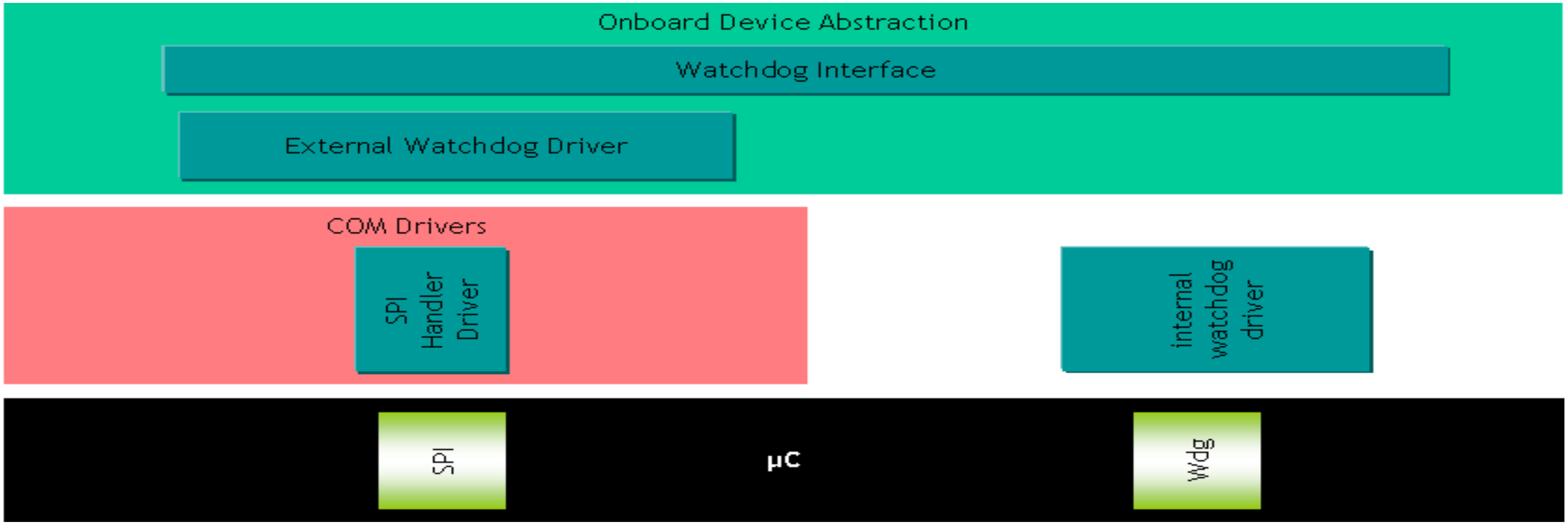
I/O Hardware Abstraction

- Group of modules which abstract from the location of I/O devices (on-chip or on-
- board) and the ECU hardware layout (e.g. µC pin connections and signal level
- inversions)
- The different I/O devices are accessed via an I/O signal interface
- Represent I/O signals (e.g. current, voltage, frequency) as they are connected to
- the ECU hardware
- Hide ECU hardware and layout properties from higher software layers



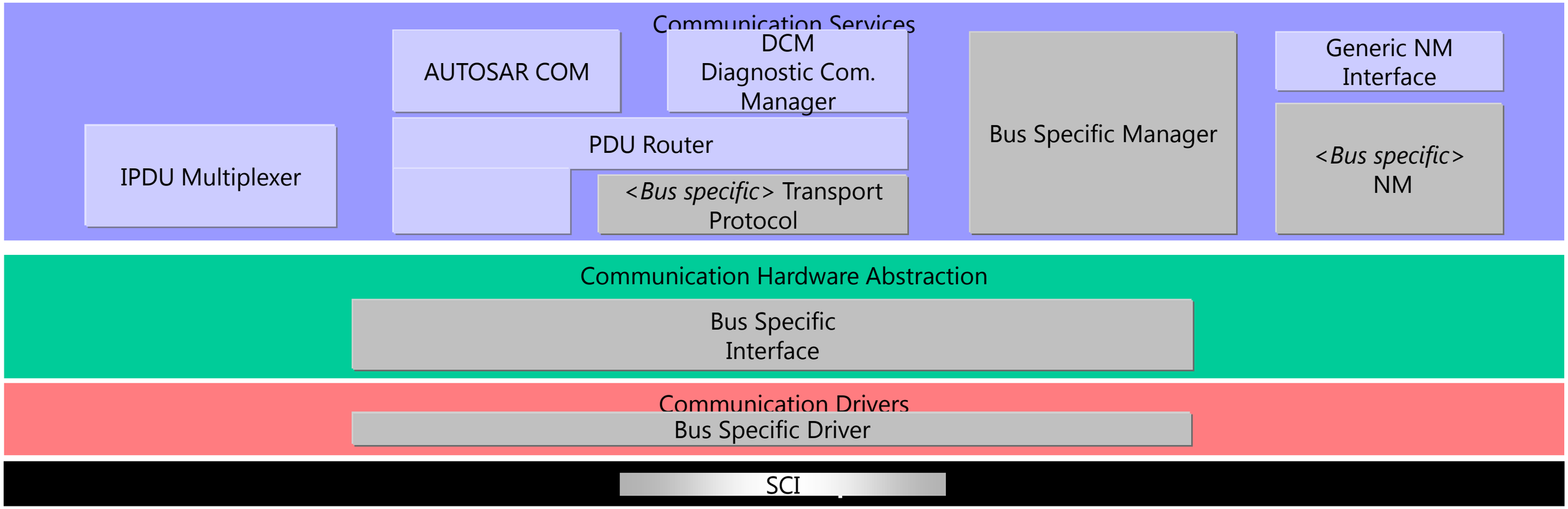
Onboard Device Abstraction

- Contains drivers for ECU onboard devices (which cannot be seen as sensors or actuators)
- like system basic chip, external watchdog**, etc.** Those drivers access the ECU onboard
- devices via the Microcontroller Abstraction Layer
- Abstracts from ECU specific onboard devices

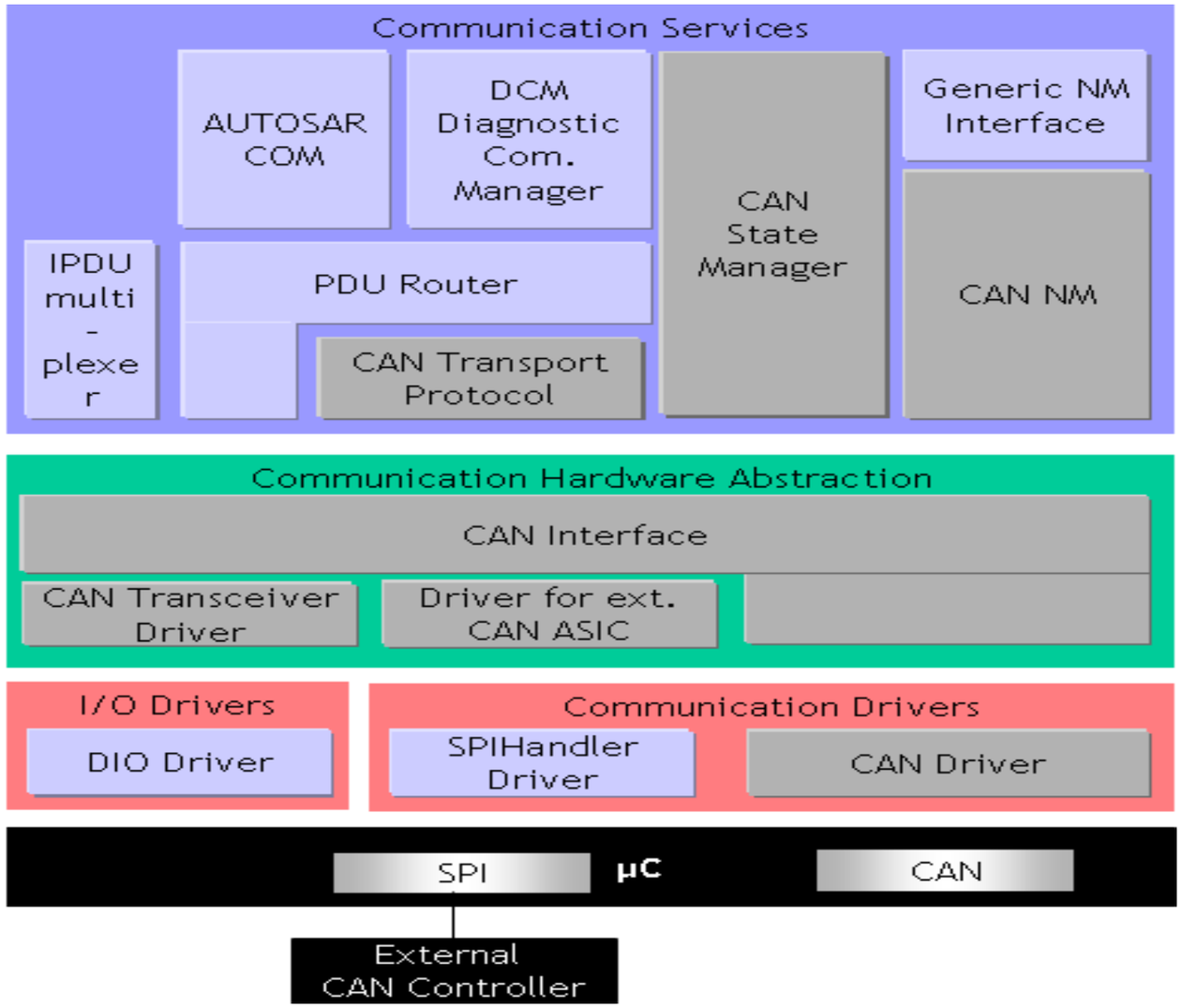


Communication Services

- Group of modules for vehicle network communication (CAN, LIN and FlexRay)
- Provide uniform interface to the vehicle network for communication between different applications
- Provide uniform services for network management
- Provide uniform interface to the vehicle network for diagnostic communication
- Hide protocol and message properties from the application

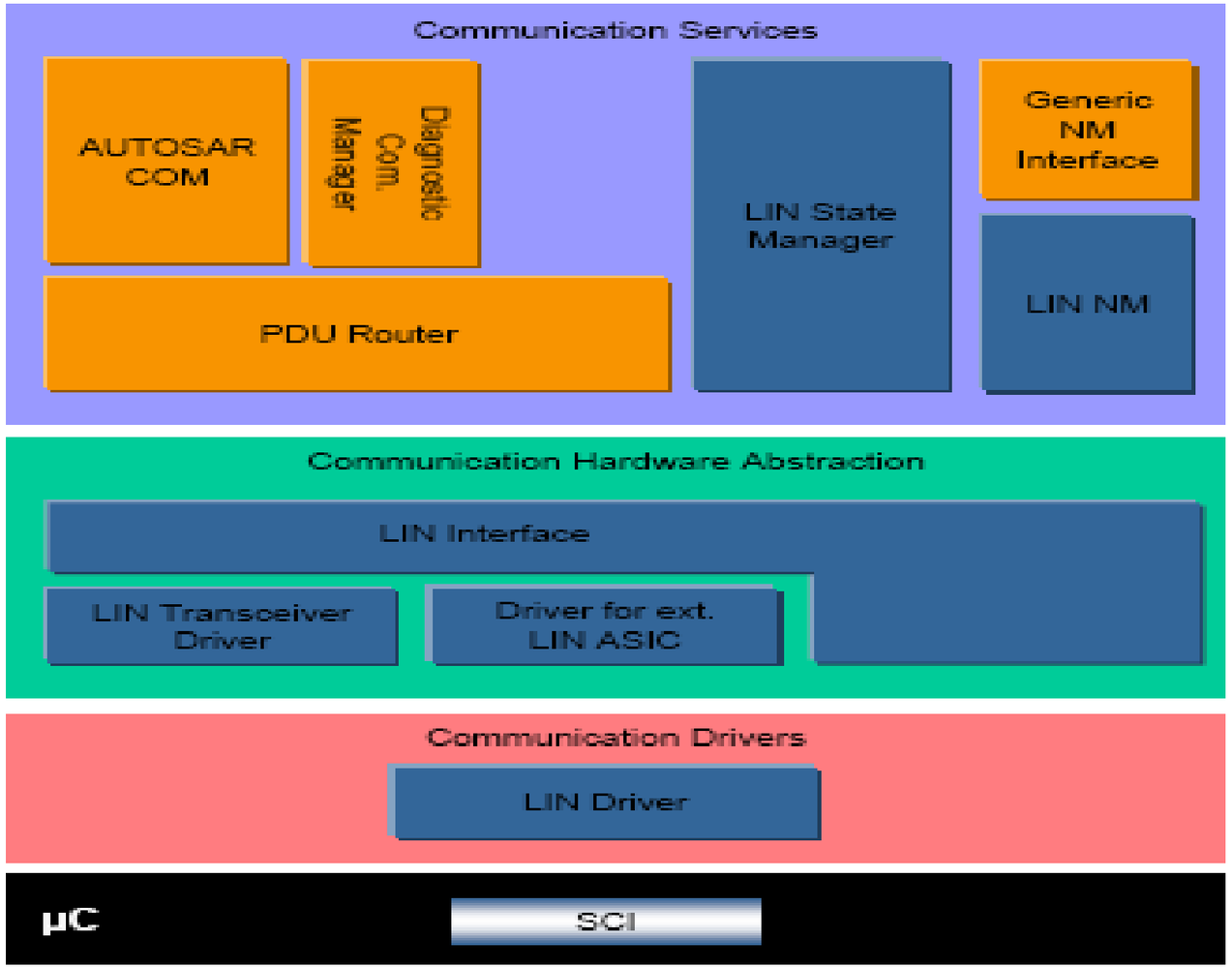


Communication Stack - CAN



- AUTOSAR COM, PDU Router, IPDUM, Generic NM Interface and Diagnostic Communication Manager are the same for all vehicle network systems
- A signal gateway is part of AUTOSAR COM to route signals
- PDU (frame) based Gateway is part of PDU Router
- IPDU multiplexing provides the possibility to add information to enable multiplexing of I-PDUs (different contents but same IDs)
- Generic NM Interface contains only a dispatcher
- CAN NM is specific for CAN networks & will be instantiated per CAN vehicle network system
- CAN State Manager handles the Start and Stop communication on CAN bus

Communication Stack – LIN

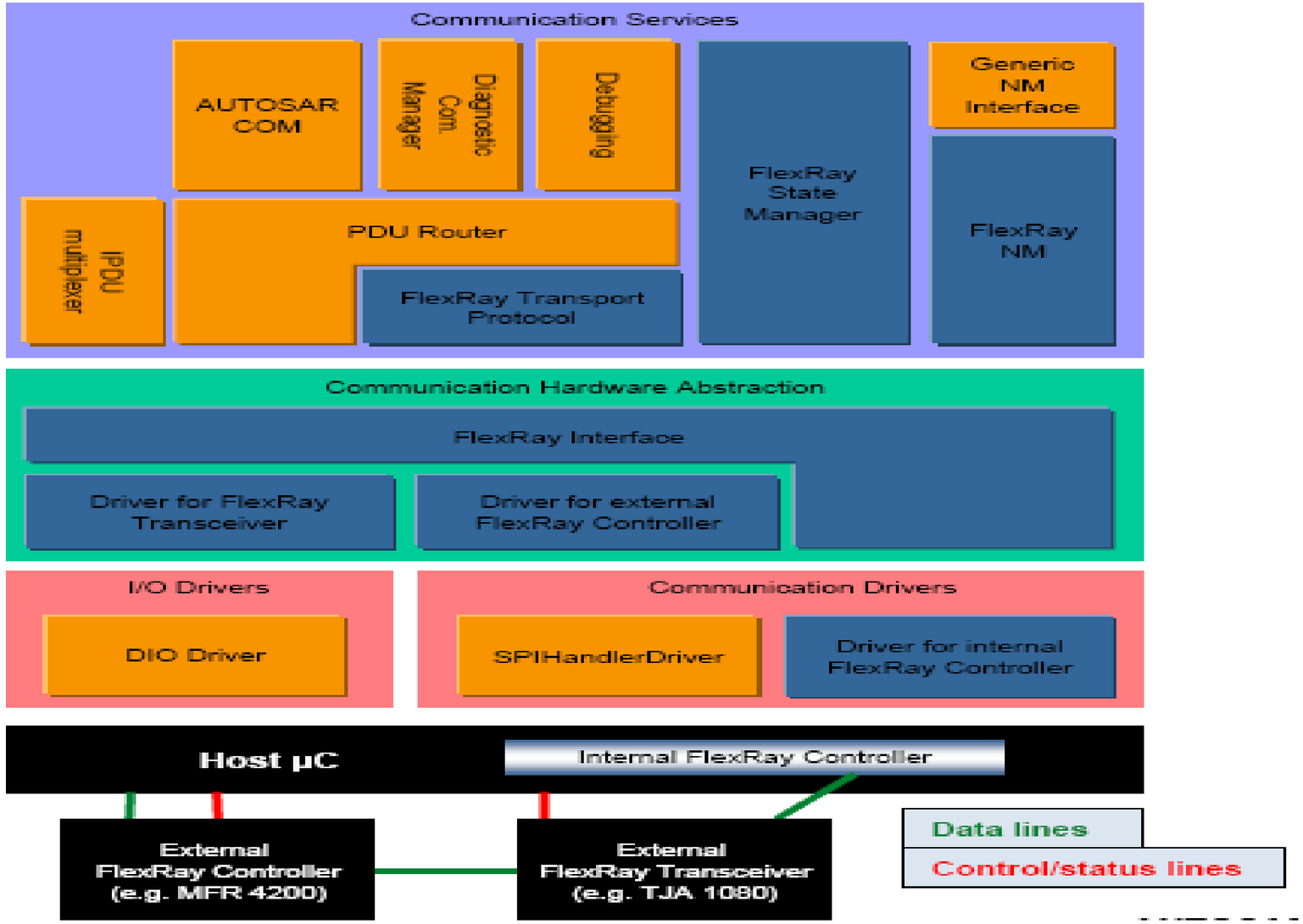


❖ Provide a uniform interface to the LIN network. Hide protocol and message properties from the application.

❖A LIN 2.1 compliant communication stack with
- ✓Schedule table manager for transmitting LIN frames and to handle requests to switch to other schedule tables
- ✓Transport protocol, used for diagnostics
- ✓A Wakeup and Sleep Interface

❖The CAN Interface with TTCAN can serve both a plain CAN Driver and a CAN Driver TTCAN

Communication Stack – FlexRay



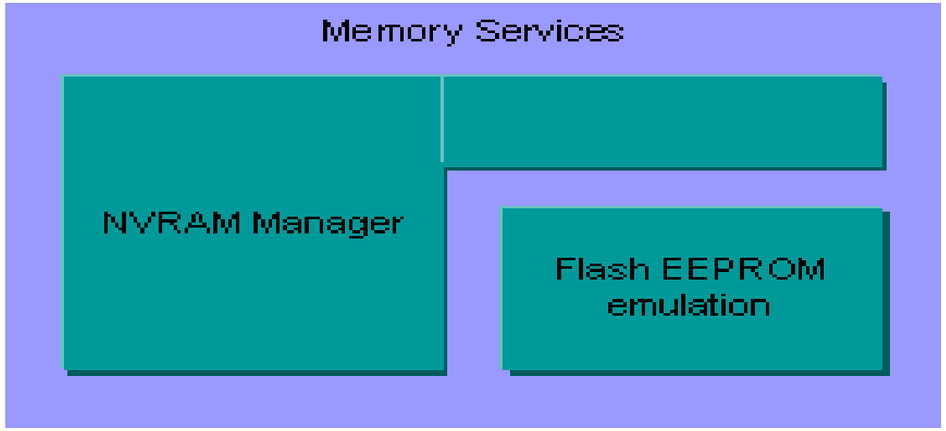
❖ Provide a uniform interface to the FlexRay network. Hide protocol and message properties from the application.

❖There are two transport protocol modules in the FlexRay stack which can be used alternatively

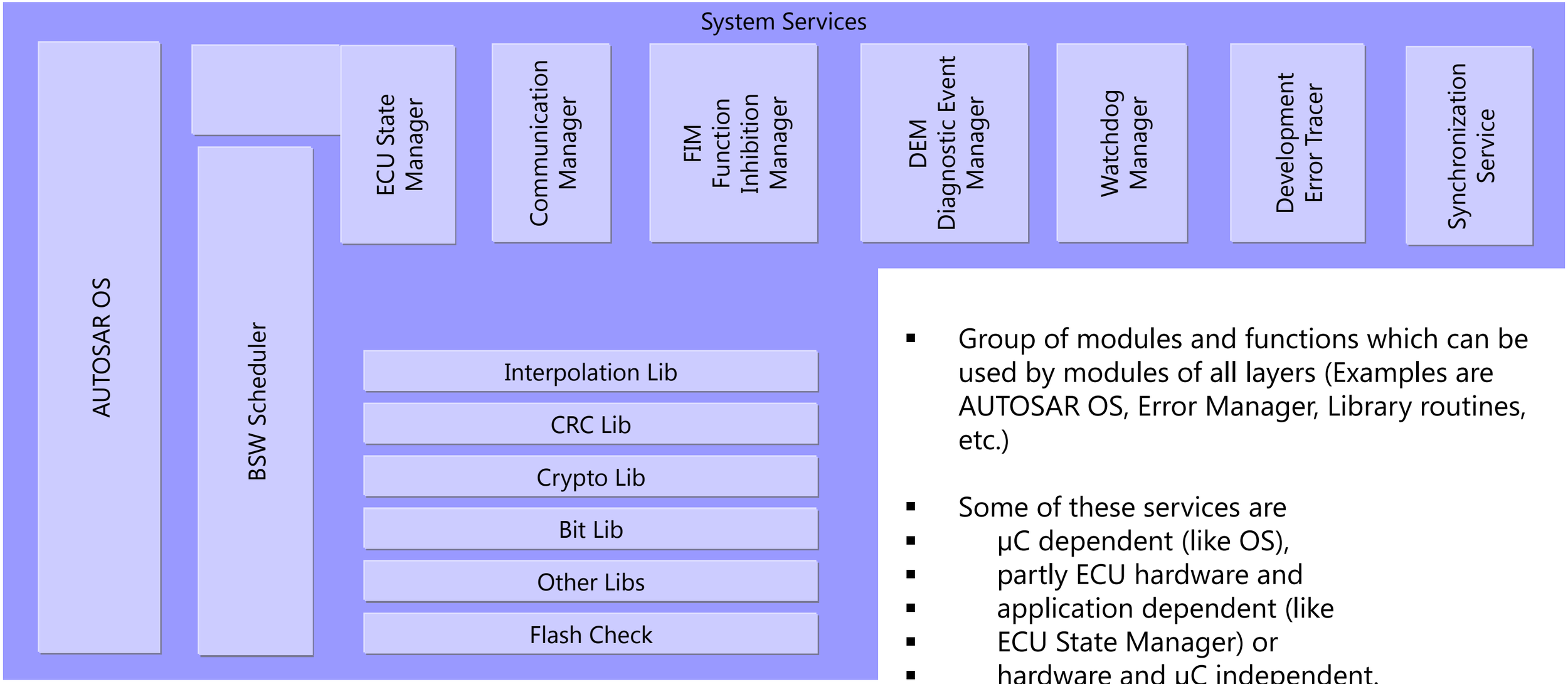
- ✓FrTp: FlexRay ISO Transport Layer Transport protocol, used for diagnostics
- ✓FrArTp: FlexRay AUTOSAR Transport Layer, provides bus compatibility to AUTOSAR R3.x

Memory Services

- Memory Services are a group of modules responsbile for the management of non volatile data (read/write from different memory drivers)
- NVRAM manager expects a RAM mirror as data interface to the application for fast read access
- 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 etc



System Services



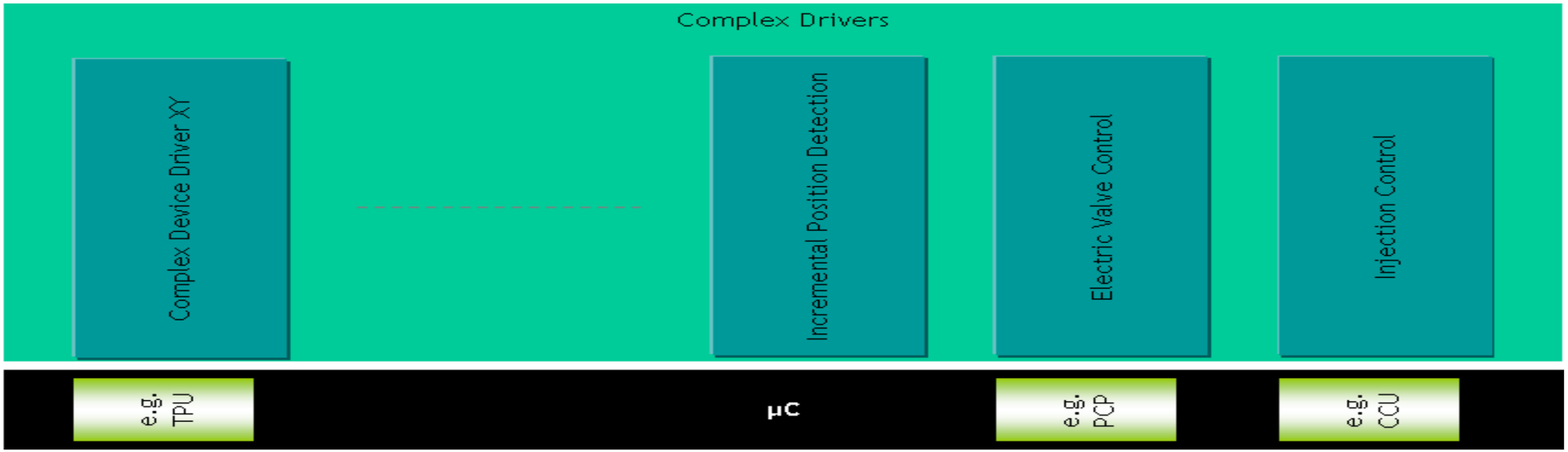
- Group of modules and functions which can be used by modules of all layers (Examples are AUTOSAR OS, Error Manager, Library routines, etc.)

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

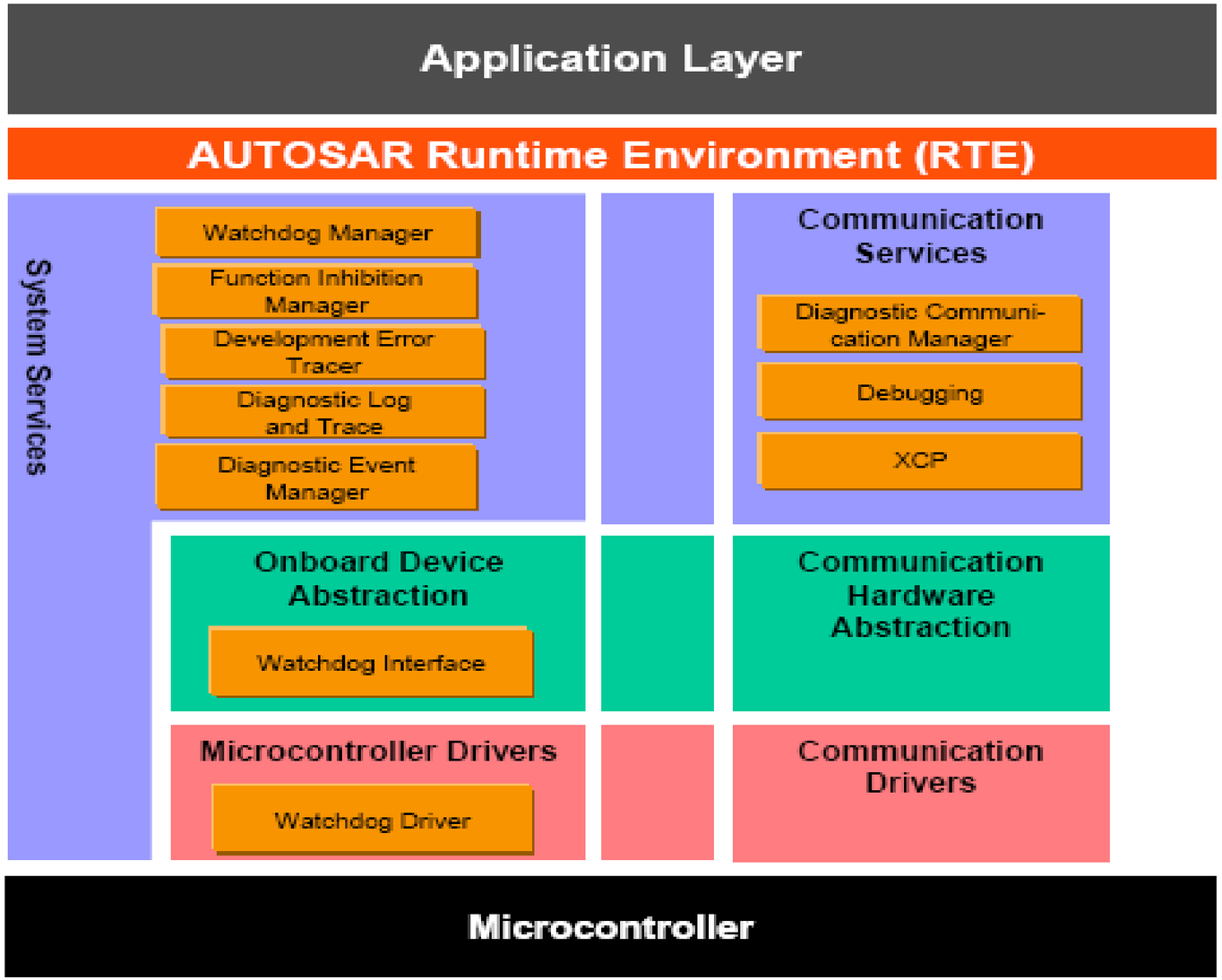
Complex Drivers

- Implements complex sensor evaluation and actuator control with direct
- access to the µC using specific interrupts and/or complex µC peripherals (like
- PCP, TPU); e.g. fuel injection control, electric valve control

- Should fulfill the special functional and timing requirements for handling
- complex sensors and actuators



Error Handling, Reporting and Diagnostic



❖ The Debugging module supports debugging of the AUTOSAR BSW. It interfaces to ECU internal modules and to an external host system via communication.

❖The Diagnostic Event Manager is responsible for processing and storing diagnostic events (errors) and associated FreezeFrame data

❖The module Diagnostic Log and Trace supports logging and tracing of applications. It collects user defined log messages and converts them into a standardized format

❖All detected development errors in the Basic Software are reported to Development Error Tracer

❖The Diagnostic Communication Manager provides a common API for diagnostic services

Interfaces in AUTOSAR & Architecture Overview

- AUTOSAR Interface
- Standardized AUTOSAR Interface
- Standardized Interface

- Implementation Conformance Class 3 - ICC3
- Implementation Conformance Classes – ICC2
- Implementation Conformance Classes – ICC1

Configuration Classes

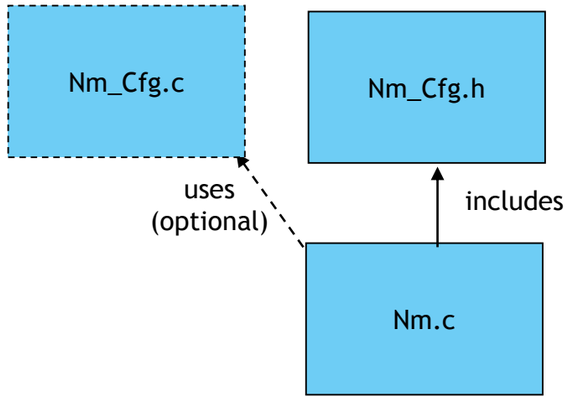
- **Pre compile time**
  - Preprocessor instructions
  - Code generation (selection or synthetization)

- **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. re-flashing in ECU production line)
  - Single or multiple configuration sets can be provided. In case multiple configuration sets are provided, the actually used configuration set is to be specified at runtime

Configuration Classes - Pre compile time

- Enabling/disabling optional functionality
- Optimization of performance and code size
- The module must be available as source code
- The configuration is static, To change the configuration, the module has to be recompiled

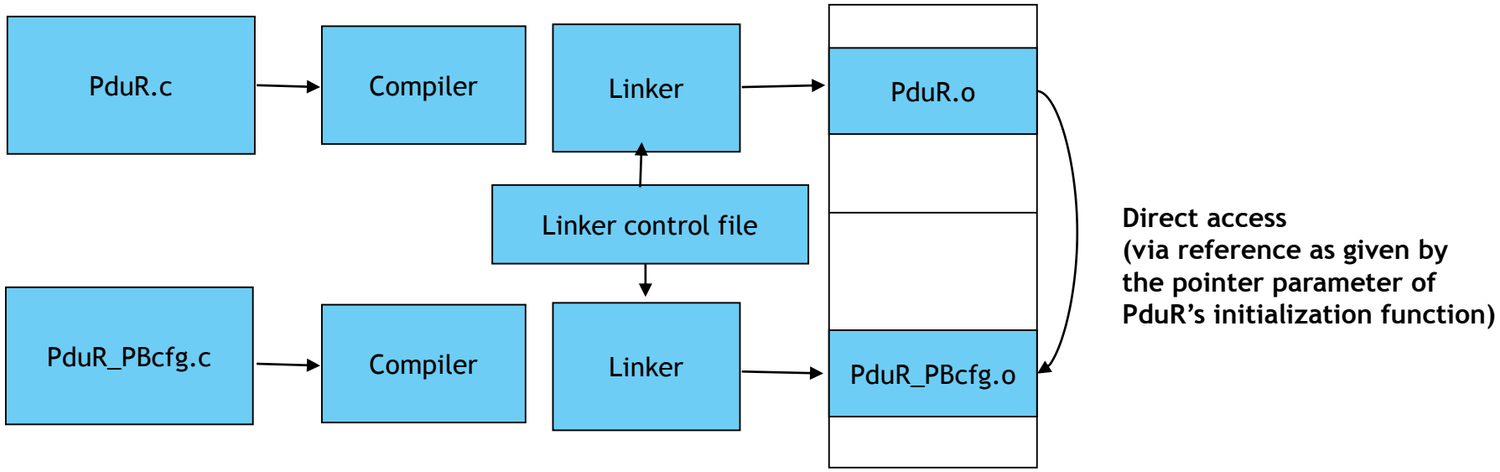


Configuration Classes - Link Time

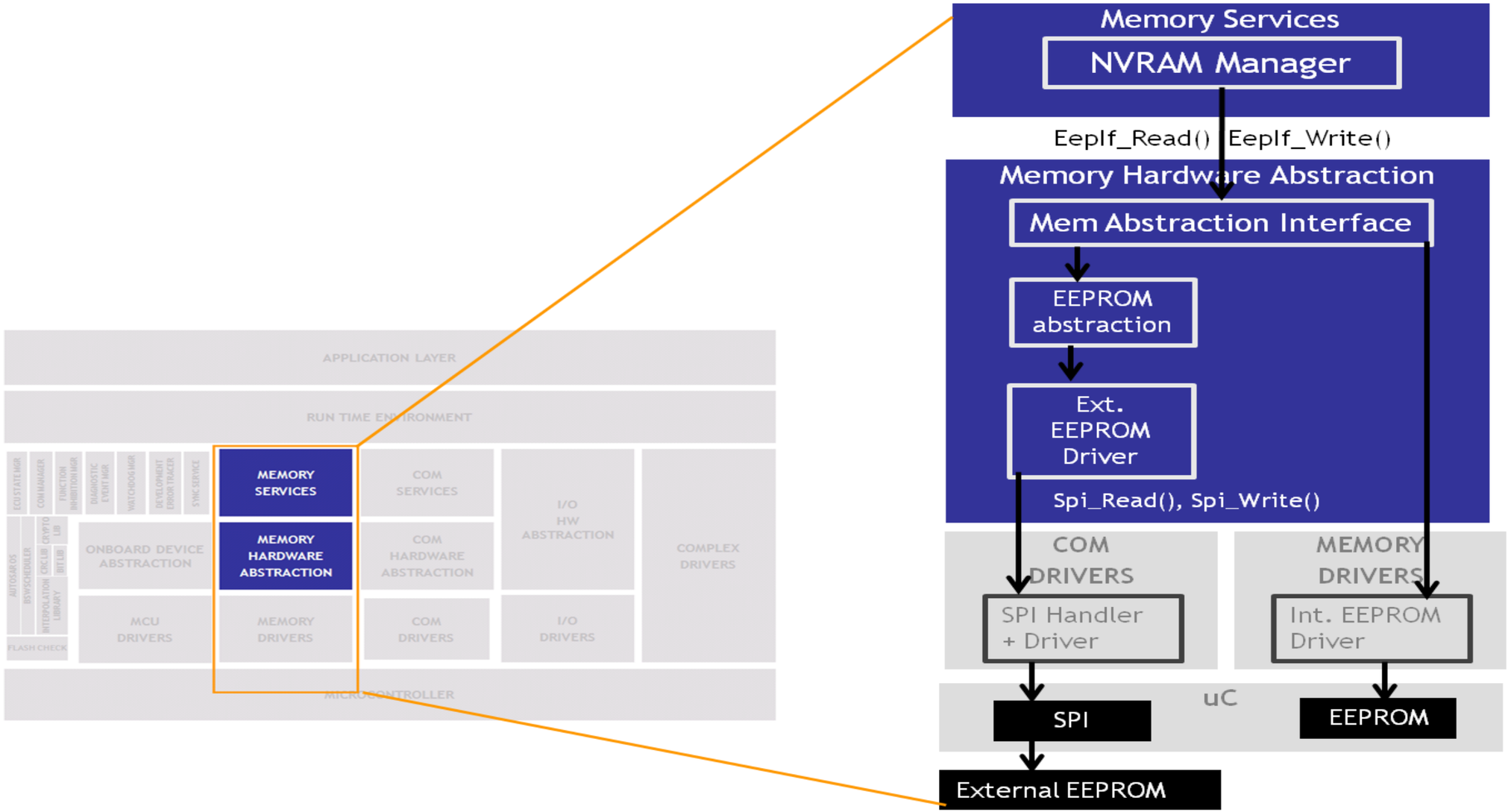
- Configuration of modules that are only available as object code (e.g. IP protection or warranty reasons)
- Selection of configuration set after compilation but before linking
- One configuration set, no runtime selection
- Configuration data shall be captured in external constants
- These external constants are located in a separate file
- The module has direct access to these external constants

Configuration Classes - Post build time

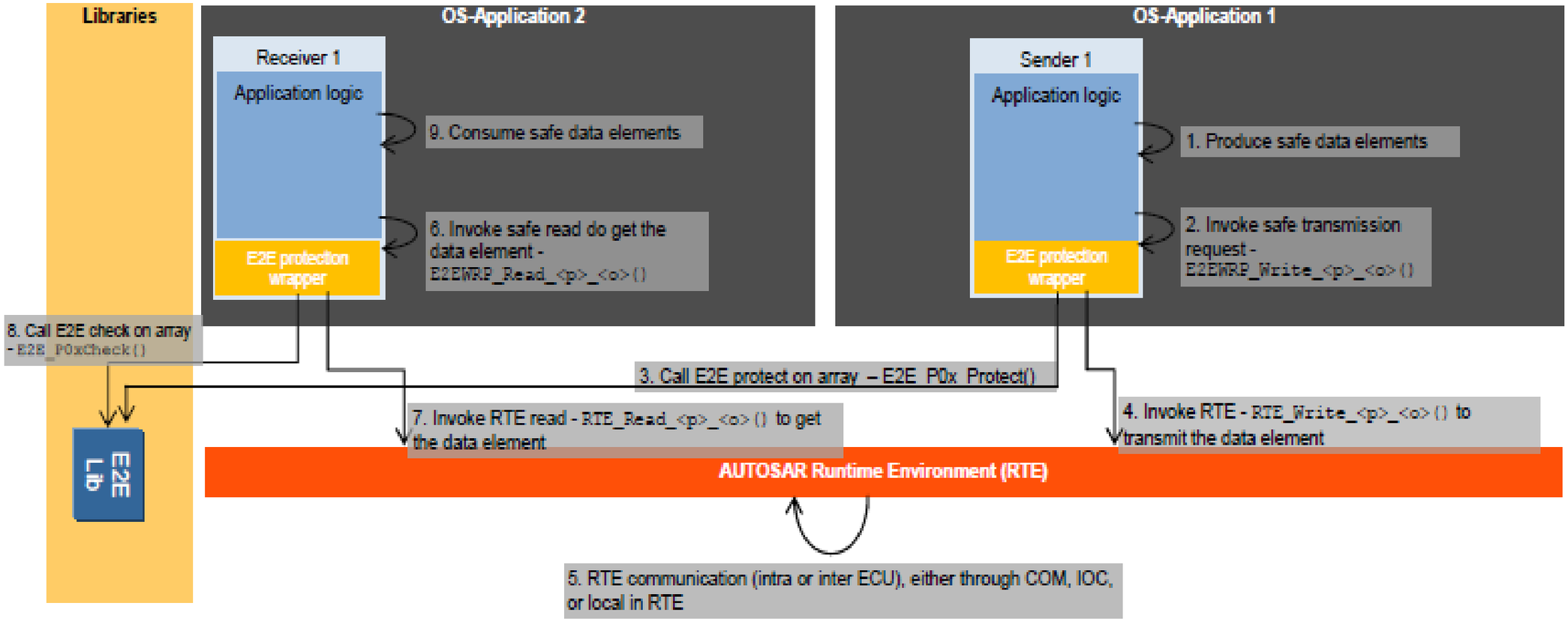
- Configuration of data where only the data structure is defined but the contents not known during ECU build time
- Configuration of data that is likely to change or has to be adapted after ECU build time
- Reusability of ECUs across different product lines
- Implementation requires dereferencing which has impact on performance, code and data size
- One configuration set, no runtime selection (loadable)
- 1..n configuration sets, runtime selection possible (selectable)



Communication between different Layers – Example of EEPROM



E2E



E2E

- The End-to-End protection allows the following:
  - ✓ It protects the safety-related data elements (resp. safety-related I-PDUs) to be sent over the RTE by attaching control data,
  - ✓ It verifies the safety-related data elements (resp. safety-related I-PDUs) received from the RTE using this control data, and
  - ✓ It indicates that received safety-related data elements (resp. safety-related I-PDUs) are faulty, which then has to be handled by the receiver SW-C.

Questions

Thank You

www.kpit.com