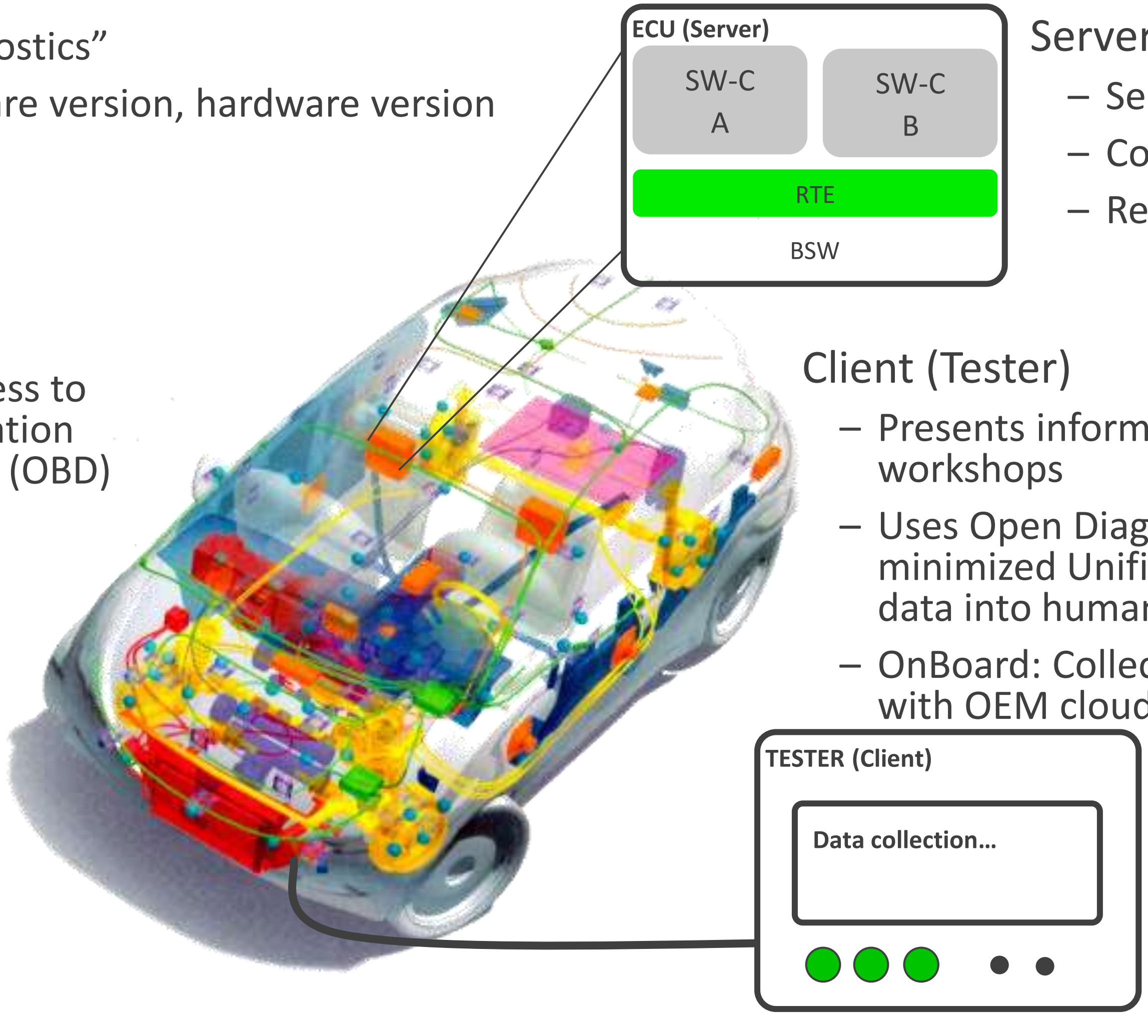
EB tresos classic AUTOSAR training - Diagnostics

Chapter overview

Overview

Introduction to Diagnostics

- What is covered by "Diagnostics"
  - Identification e.g. software version, hardware version
  - Fault management
  - Coding and Calibration
  - Programming
  - Function check
  - Authorities demand access to emission related information → On-Board Diagnostics (OBD)



Server (ECU)

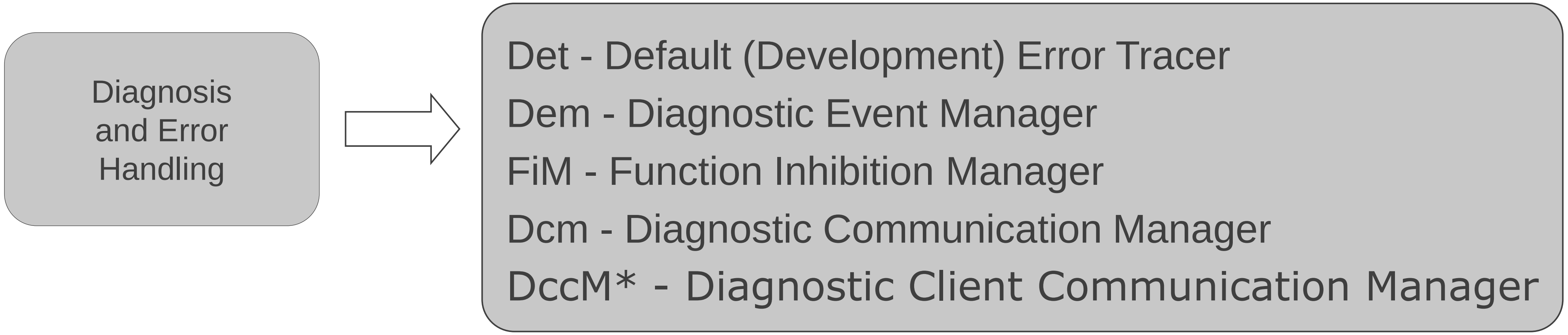
- Service provider
- Collects data ECU internal
- Receives requests and executes services

Client (Tester)

- Presents information and provides repair instructions for workshops
- Uses Open Diagnostics data eXchange (ODX) to translate minimized Unified Diagnostic Services (UDS) protocol data into human readable information and instructions
- OnBoard: Collects diagnostic data for synchronization with OEM cloud/back-end

Diagnostic modules

- Diagnostics are done by several modules

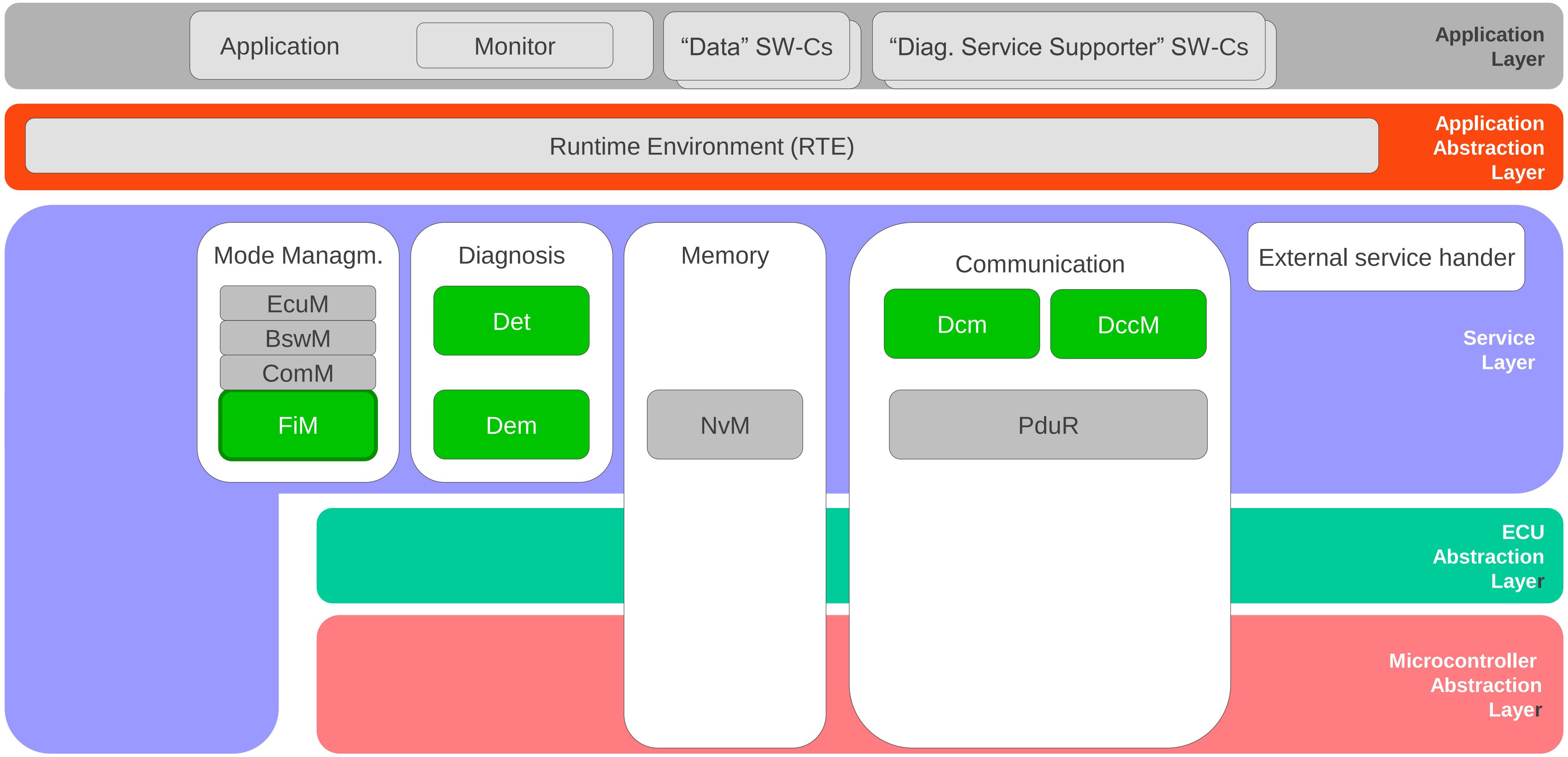


Det - Default (Development) Error Tracer
Dem - Diagnostic Event Manager
FiM - Function Inhibition Manager
Dcm - Diagnostic Communication Manager
DccM* - Diagnostic Client Communication Manager

- Supports different protocols
  - UDS (Unified Diagnostic Services) - Diagnostic communication protocol → specified in ISO 14229-1
  - OBD (On-board diagnostics) – standardizes access to the status of emissions-related diagnostics→ specified in ISO 15031-6

* EB specific implementation not standardized by AUTOSAR, requires additional licensing

Diagnostic - Overview



Default (Development) Error Tracer (Det)

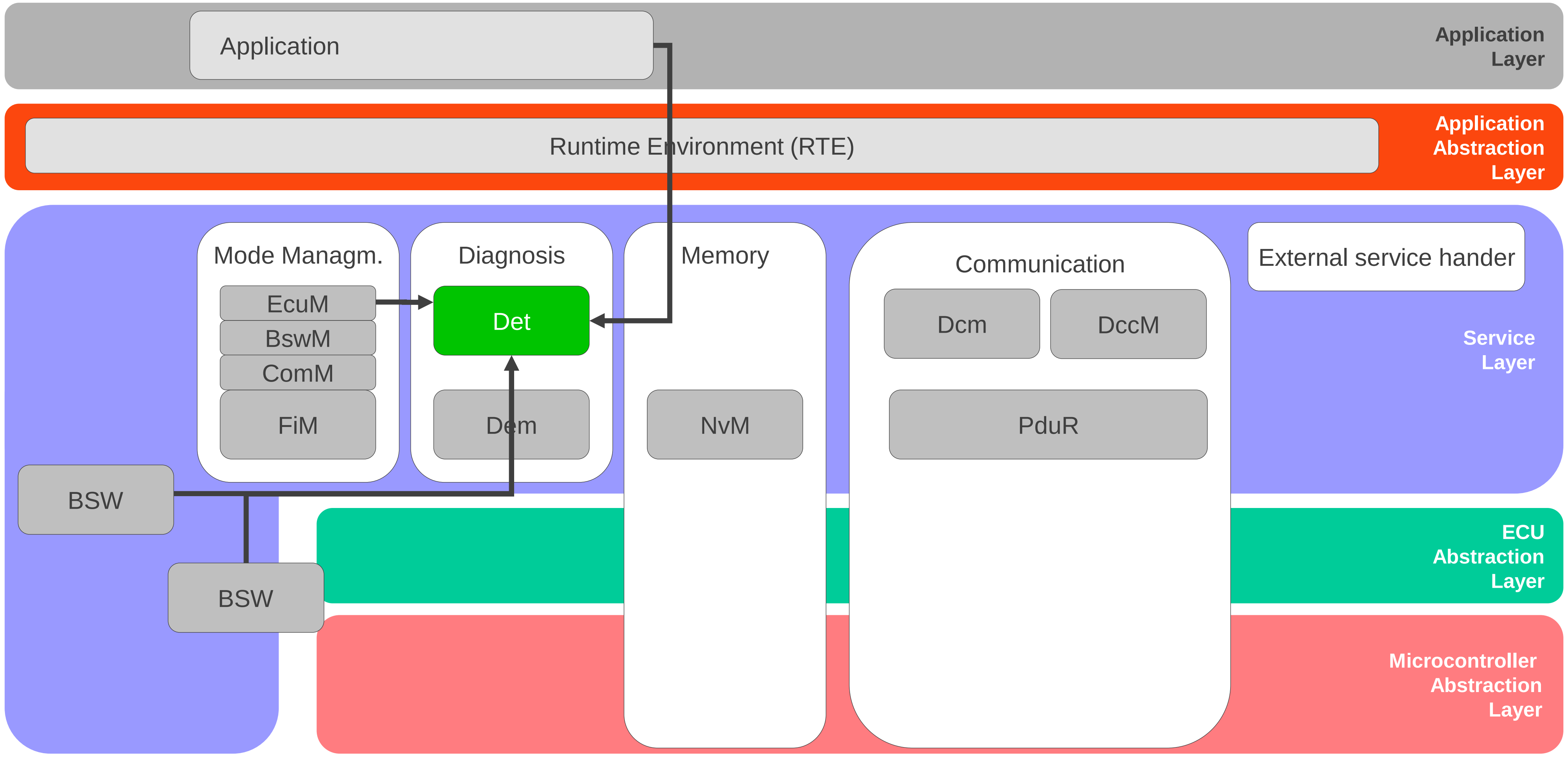
Det overview

- The Det provides interfaces for reporting detected 3 types of errors
  - Development errors (errors that shall be detected during development phase)
  - Runtime errors (systematic faults)
  - Transient errors (occur in the hardware)

- Only the API, but neither the implementation nor the complete configuration is defined by AUTOSAR

- EB tresos AutoCore solution provides comprehensive functionalities, which are highly configurable

Diagnostic - Det



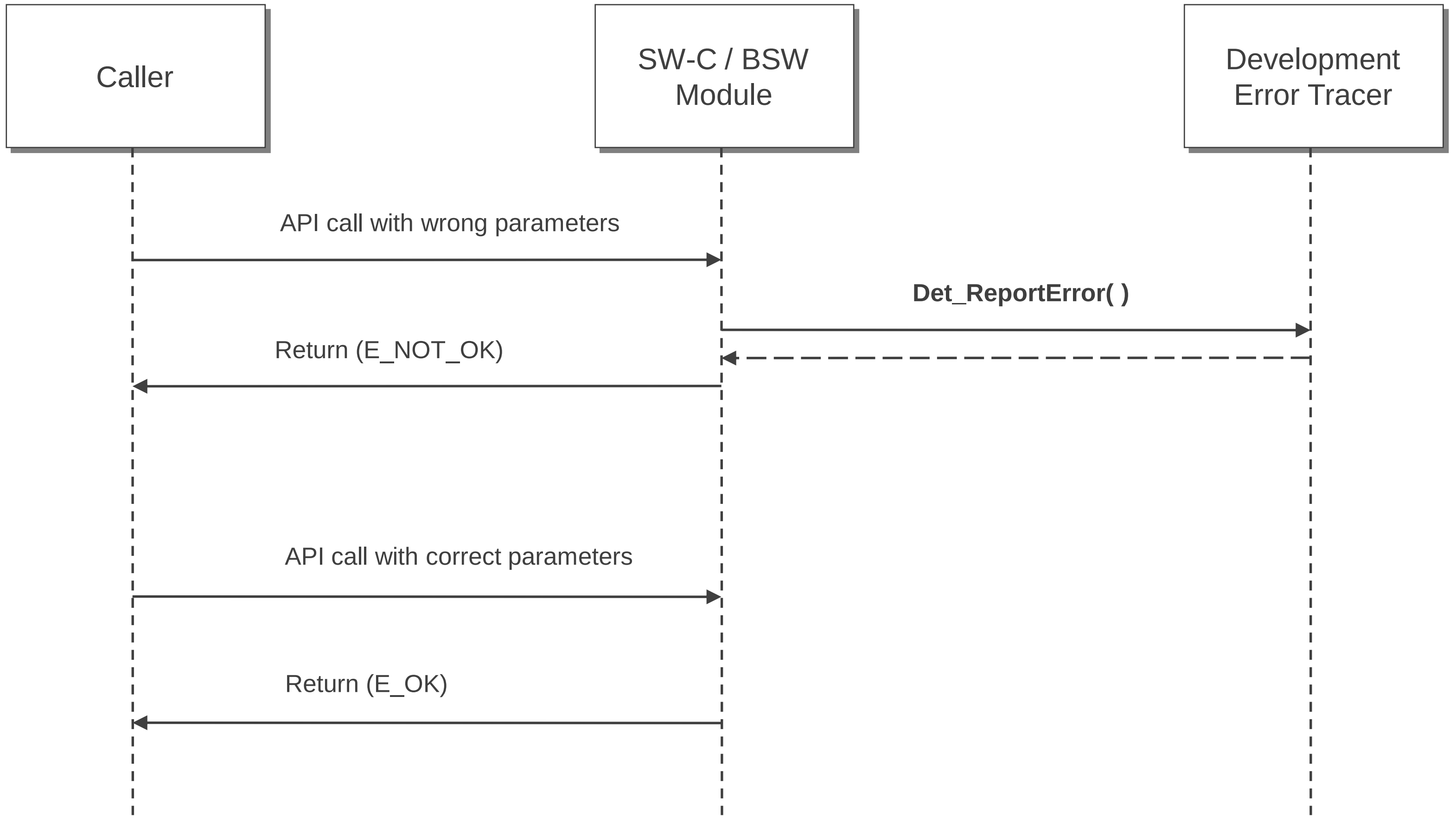
Development error functionality

- Det service: **Det_ReportError**
- Scope: Errors that shall be detected during development phase caused by incorrect integration or invalid configuration
- Additional **checks** in the BSW when enabled in configuration (to be done for **each BSW module**)
  - Examples of checks: Wrong parameter value, buffer overflow, wrong sequence of calls, etc.
- Configurable behavior of the control flow of execution (EB specific parameter: EnableReportErrorStopExecution)
  - True: Development errors shall stop execution of the entire process / False: Det does not stop the execution and just stores the reported errors
- All default (development) errors are reported to Det:

```
Det_ReportError( ModuleId, InstanceId, ApiId, ErrorId )
```

- Also usable for the **application** via Service port

Det sequence



Runtime and Transient errors functionality

- Runtime and Transient faults will not cause assertions and therefore even not necessarily cause the abortion of the ‘normal’ control flow of execution (as development errors will do)

- Runtime errors:
  - Det service: **Det_ReportRuntimeError**
  - Runtime errors are systematic faults that do not necessarily affect the overall system behavior (e.g. wrong PDU-Ids, wrong post-build configurations)
  - An error handler of runtime errors is executed synchronously and may only store the corresponding events to a memory, may call Dem and may execute any reasonable action

- Transient faults:
  - Det service: **Det_ReportTransientFault**
  - Transient faults occur in the hardware due to particle passages or thermal noise for instance and may cause software issues
  - they may heal in a sense that they disappear again or get masked or get corrected by software activity
  - monitors of transient errors (if any) shall be used also in production code

Det features

- **Statistical data**: count errors, count lost errors
- Configurable **buffer** for first/last appeared development errors
- Internal and breakpoint **logging mode**
- **Callback** notification
- For multicore projects, development errors, runtime errors and trasient faults can be reported from different cores

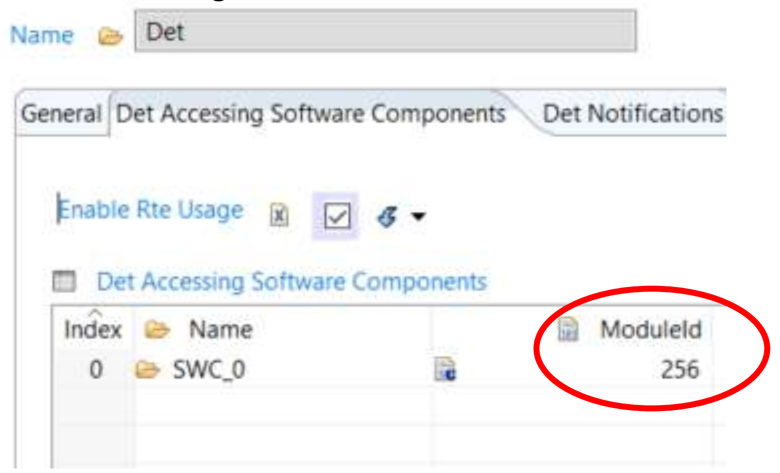
Interlude: Port defined arguments

- SW-C use a client-server service interface to communicate with the Det
- In order to reuse the C API already defined in the Default Error Tracer BSW module, the Det service uses the same argument names as in the C API (`Det_ReportError( ModuleId, InstanceId, ApiId, ErrorId )`
- In order to keep the client code independent from the configuration of number of clients, the "Module IDs" are not passed from the clients to Det but are modeled as "port defined argument values" of the Provide ports on the Det side
- Hence the "Module IDs" will not show up as arguments in the operation of the client-server interface

From the generated Rte.c (simplified) – based on the exercise project

```
Rte_Call_SwcCyclicCounter_PortDet_ReportError (uint8 InstanceId, uint8 ApiId,
uint8 ErrorId)
{
  Std_ReturnType Rte_Status = RTE_E_OK;
  Rte_Status = Det_Rte_ReportError(256U, InstanceId, ApiId, ErrorId);
}
```

From Det configuration

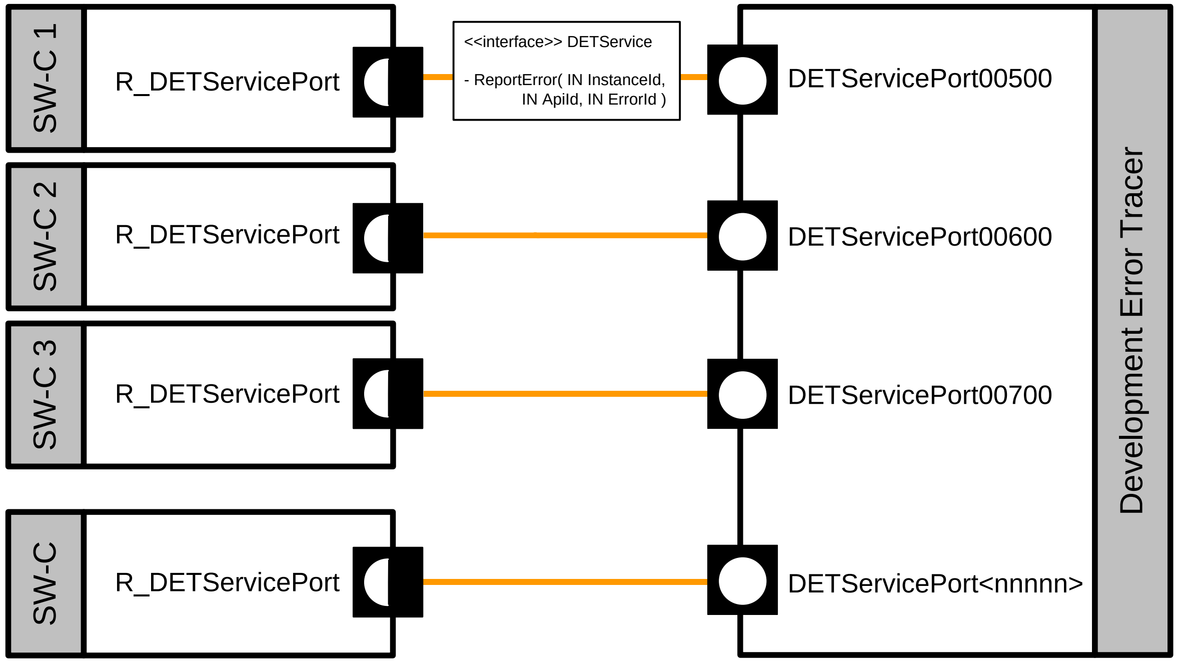

General
Det Accessing Software Components
Det Notifications
Enable Rte Usage
Det Accessing Software Components

Name: Det

Index	Name	ModuleId
0	SWC_0	256

Det service component

- Example of several SW-Cs connected to the Det
- Each Accessing SW-C needs to be configured in the Det configuration
- The Module ID type is of range 0…65535
- Values in the range of 0...254 are reserved for Basic Software Modules, complex drivers use either 255 or a value between 2048 and 4095.
- All others can be used for application software components



Diagnostic Event Manager (Dem)

Dem overview

- The Dem receives the result of a monitor (Diagnostic Event) → serves as an event sink

- The Dem module processes **diagnostic events** which are reported as either **passed or failed**
- Dependent on the Dem configuration (e.g. **operation cycles, debouncing, environmental conditions)** the event qualifying takes place
- **Error** means always failed (and is already debounced from failed-report-sequences)

- The Dem is responsible for the storage handling. Besides the event status the Dem can also capture and store environmental data for events

- Interaction with other Software Modules
  - Event reporting from SW-C via the RTE or directly from other BSW modules
  - It uses the Non-volatile RAM Manager (NvM) to **store** the fault data **persistently**
  - **Provides access** to fault data via the Diagnostic Communication Manager (Dcm, J1939Dcm), e.g., reading/clearing stored DTC information from the event memory
  - Serves as **source for functional degradation** → computed by the Functional inhibition Manager (FiM)

Usage of Dem

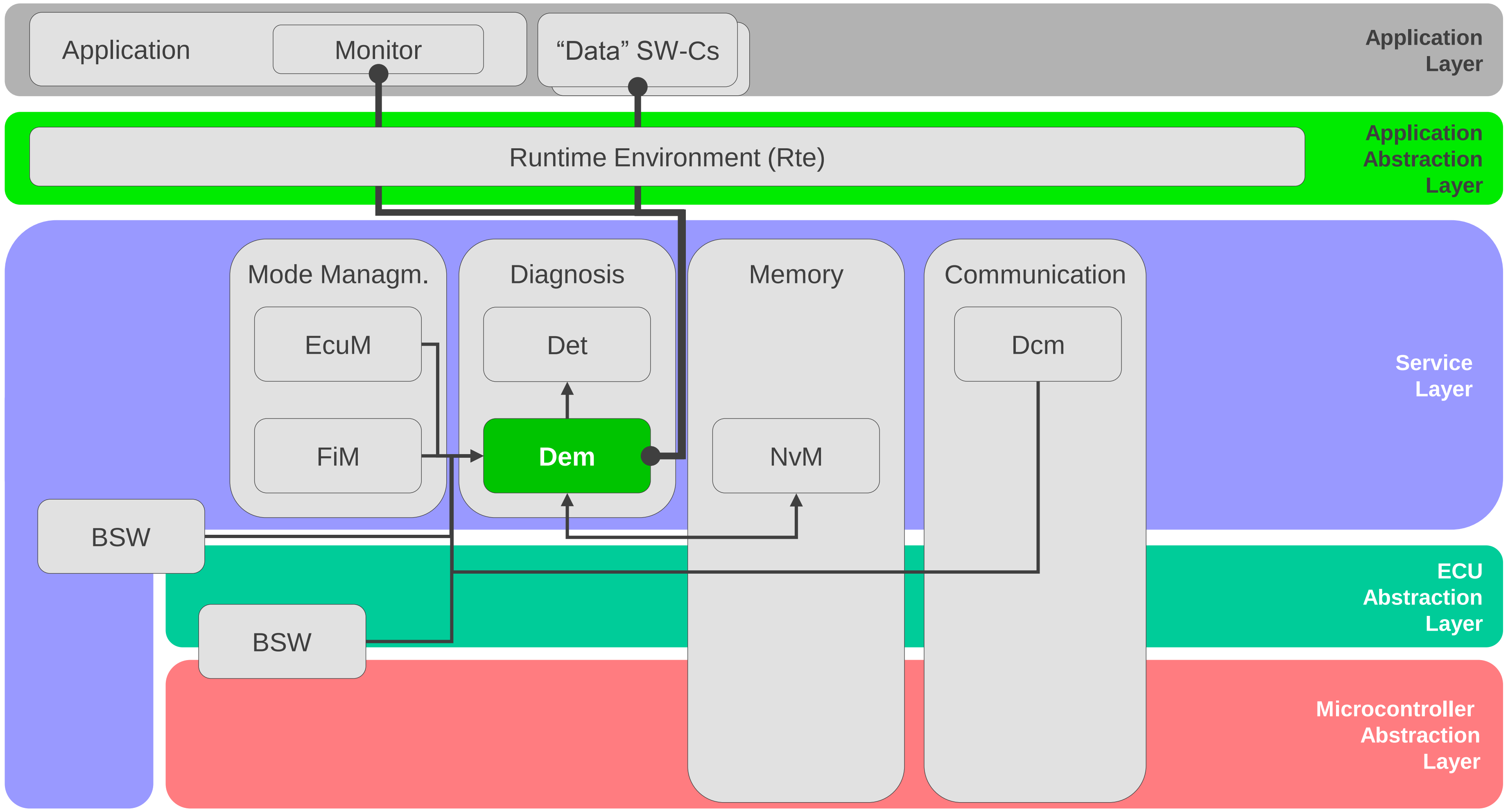
- **Car manufactures** (OEM)
  - define DTCs, event trigger points, precondition for event detection and environment data captured for events
  - Format: Excel sheet or in a Diagnostic Extract as requirements for ECU projects

- **Tier1s** (ECU Developer)
  - ECU configuration and integration
  - the integrator must know the specific OEM requirements concerning diagnostic usage

- For **application developers** the Dem just acts as destination for results from monitors

- The Exchange of Diagnostic requirements between OEM <-> Tier1 <-> Application Developer and Integrator must be organized in terms of format and process
- Responsibilities must be clear

Diagnostic - Dem



Dem initialization

- Chicken-Egg issue
  - Dem requires its permanently stored data from NvM
  - **But**: NvM (and sub-modules) may report errors to Dem while reading

- Solution
  - Dem will be pre-initialized → errors are queued
  - NvM will be initialized → read all NVRAM data
  - Dem will be initialized and enters errors from queue into event memory

- Note: Initialization is handled by the ECU State Manager (EcuM - part of Mode Management chapter) and defined in a user callout

Functionality – overview of main features

- Event **status** and data processing and storage
- **Operation cycle** management and **Enable condition** handling (for event qualifying)
- Event **aging**
- Configurable **event memories + storage** of event related data (Freeze Frames – FF / Extended Data Records – EDR)

More details on next slides

- Status/Data-change & further **callbacks**
- Event report **debouncing** (for pre-passed / pre-failed)
- **OBD**-related functionality*
- Combination of events → 1 **Diagnostic Trouble Code (**DTC) for multiple events

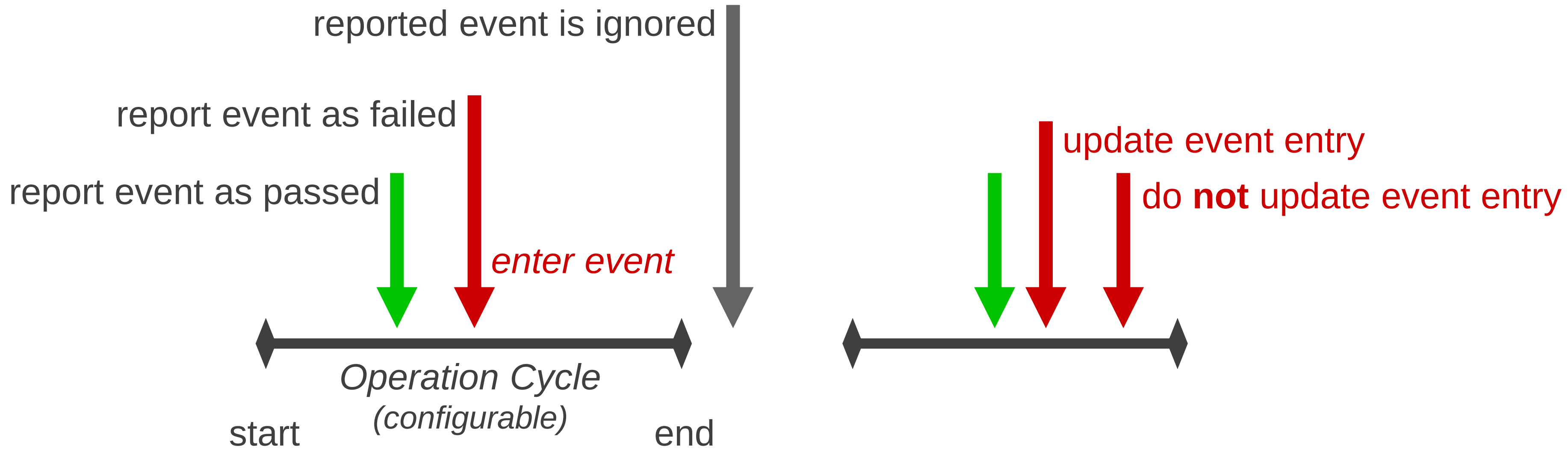
The complete list of Dem features is much longer → see product description / documentation

**These features require additional licensing*

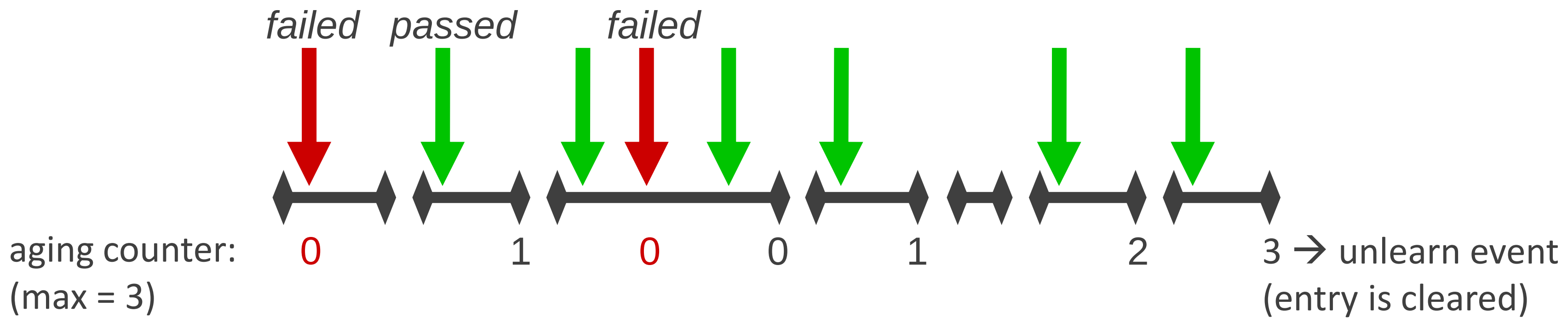
Event Reporting and Qualifying

Examples for Operation Cycles

- Startup (no bus error logging because of ongoing bus sync)
- Normal
- Shutdown



Event aging


failed
passed
failed
aging counter:
(max = 3)
0
1
0
0
1
2
3 → unlearn event
(entry is cleared)

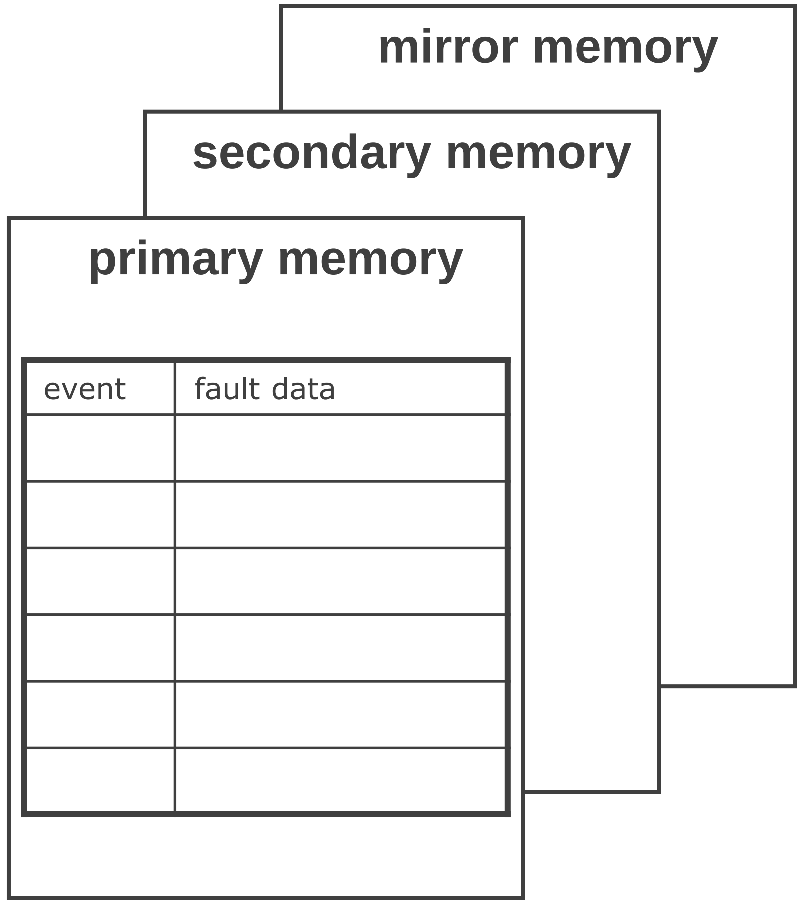
Event Status

- Dem administrates a *status byte* for each event (independent from an event memory entry) consisting of following flags (refer to UDS):
  - Test failed
  - Test failed this operation cycle
  - Pending DTC
  - Confirmed DTC
  - Test not completed since last clear
  - Test failed since last clear
  - Test not completed this operation cycle
  - Warning indicator

Elektrobit

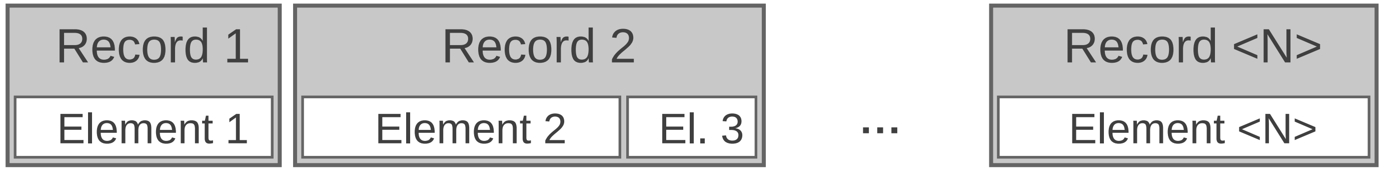
Event Memories (configurable)

- Maximal number of entries is much less than events in ECU
- Fault data
  - Extended data records
  - Freeze Frame data
- Displacement is based on event priority, status and occurrence time

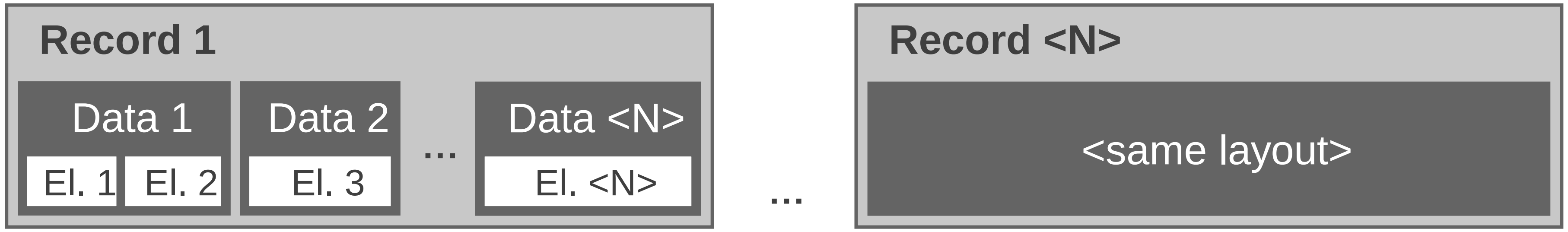


Fault Data (configurable)

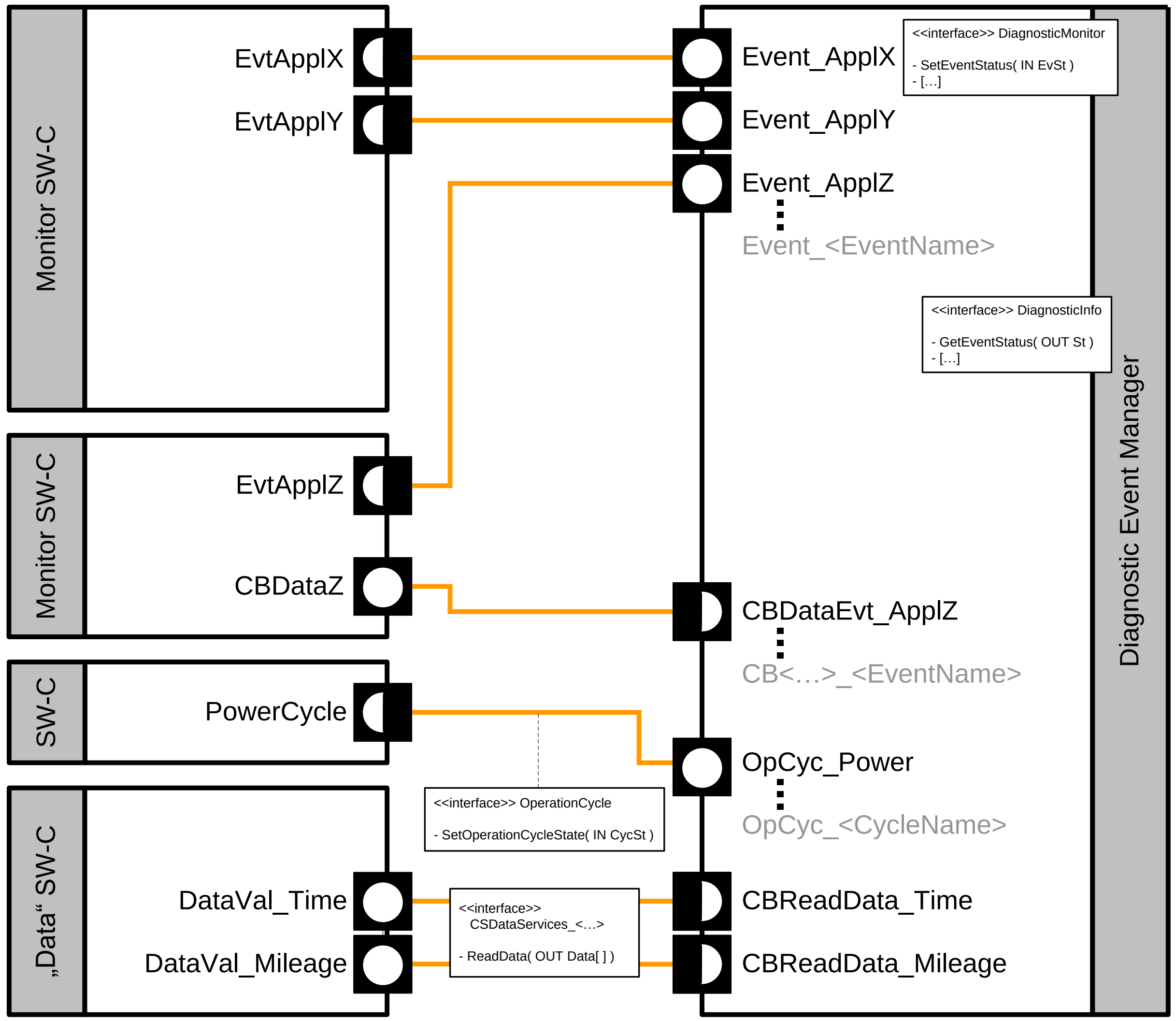
- Extended data records
  - are updated, if error re-occurs
  - can contain Dem-internal data like Event priority



- Freeze Frame data
  - are extended with new record, if error re-occurs



Dem service component



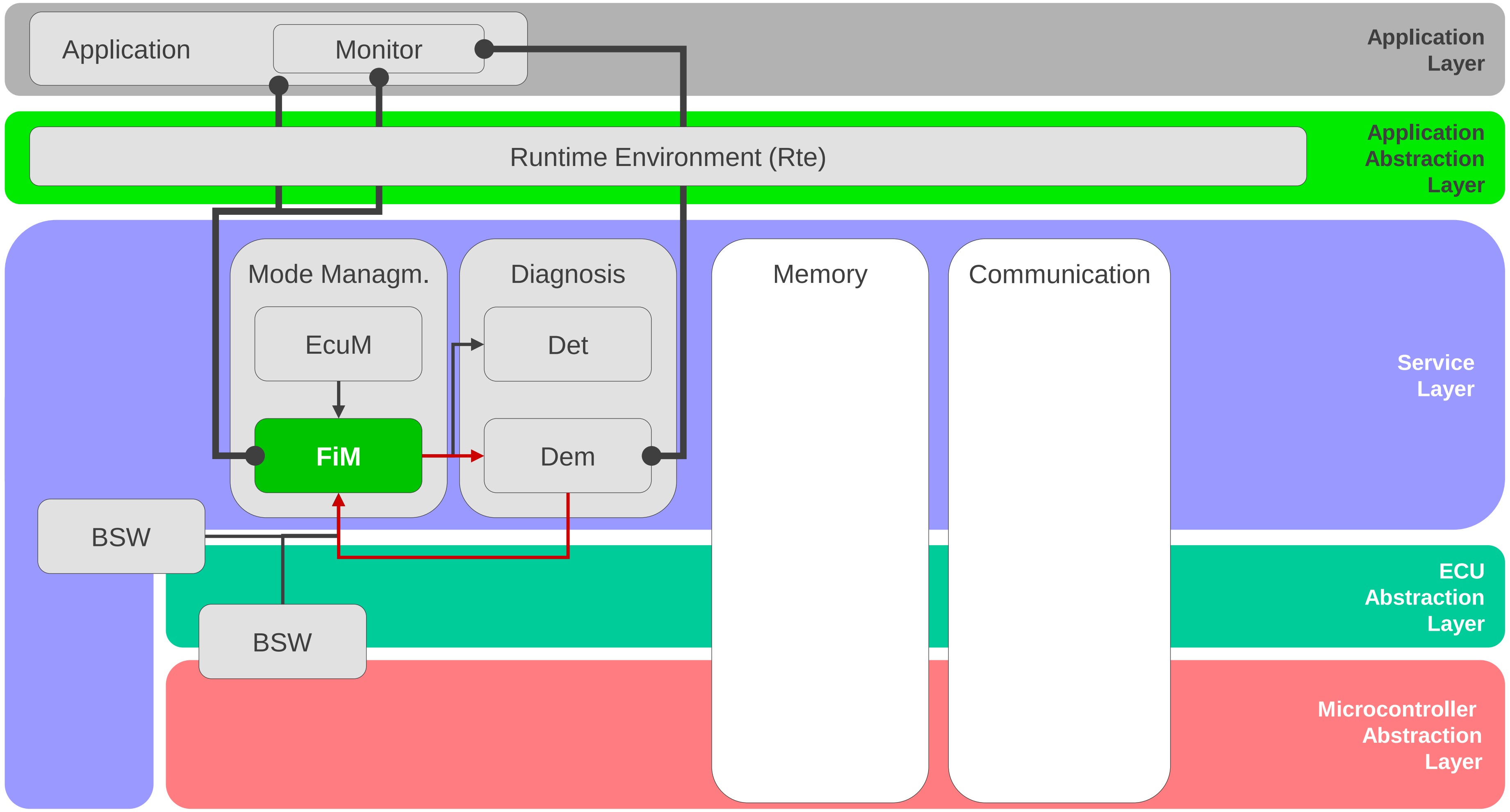
Function Inhibition Manager (FiM)

Elektrobit

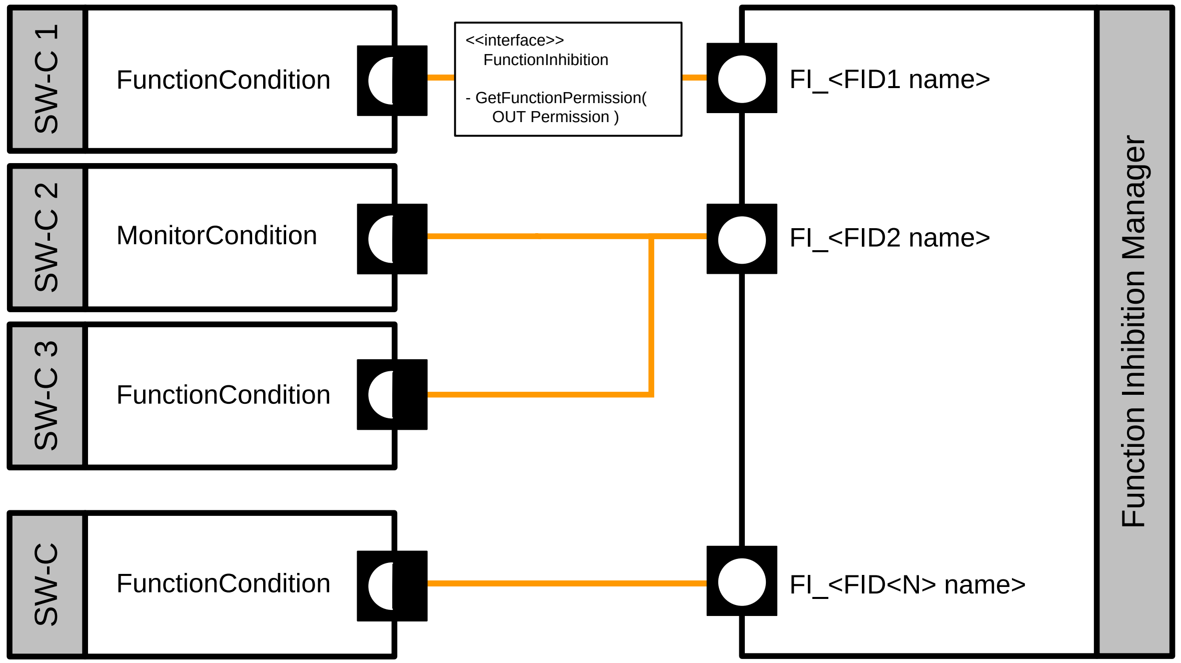
FiM - overview

- Calculates and summarizes **permission/inhibition conditions** on defined functionalities (FIDs or Function Identifiers)
- Each permission/inhibition condition is based on the **status of one or more diagnostic events** (from Dem)
- **Retrieves event status** by Dem_GetEventStatus() *or* is triggered by the Dem on event status change
- **Provides permission result** (of FID) to SW-Cs and BSW modules
- `FiM_GetFunctionPermission( FID, *Permission )`

Diagnostic - FiM



FiM service component

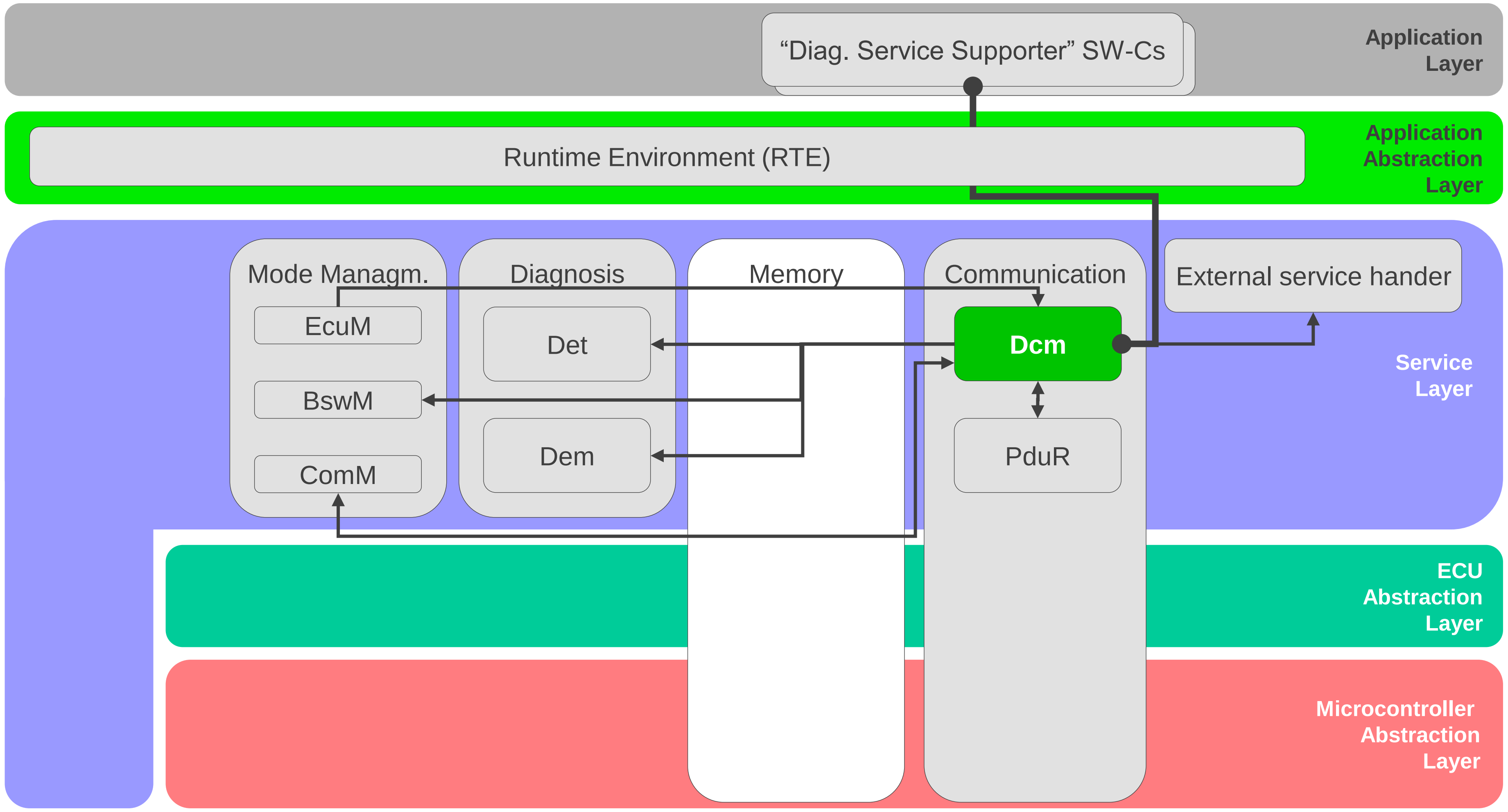


Diagnostic communication manager (Dcm)

Dcm - overview

- Acts as mediator between diagnostic services and the vehicle network communication (CAN, LIN, Flex Ray, Ethernet)
- Receives requests /sends responses for diagnostic messages from/to the PDU Router (PduR)
- Provides (or triggers) the appropriate service processors which retrieve required information from the application
- Note: For J1939 a separate Dcm exists –J1939Dcm – with it‘s own interface to Dem. Dcm and J1939Dcm can be used together if needed by the OEM

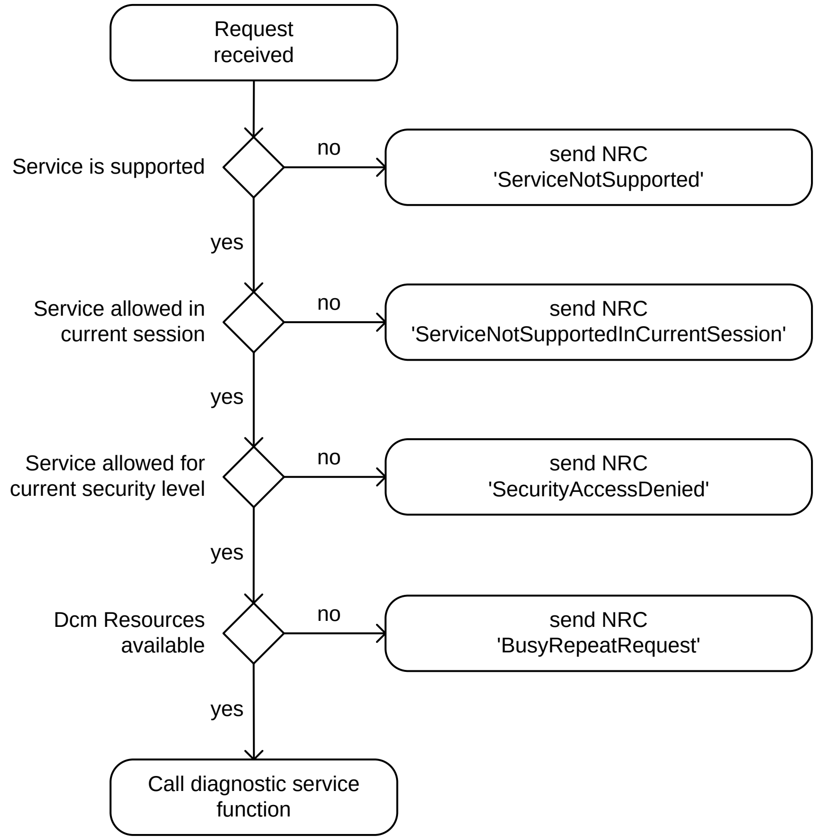
Diagnostic - Dcm



Dcm features

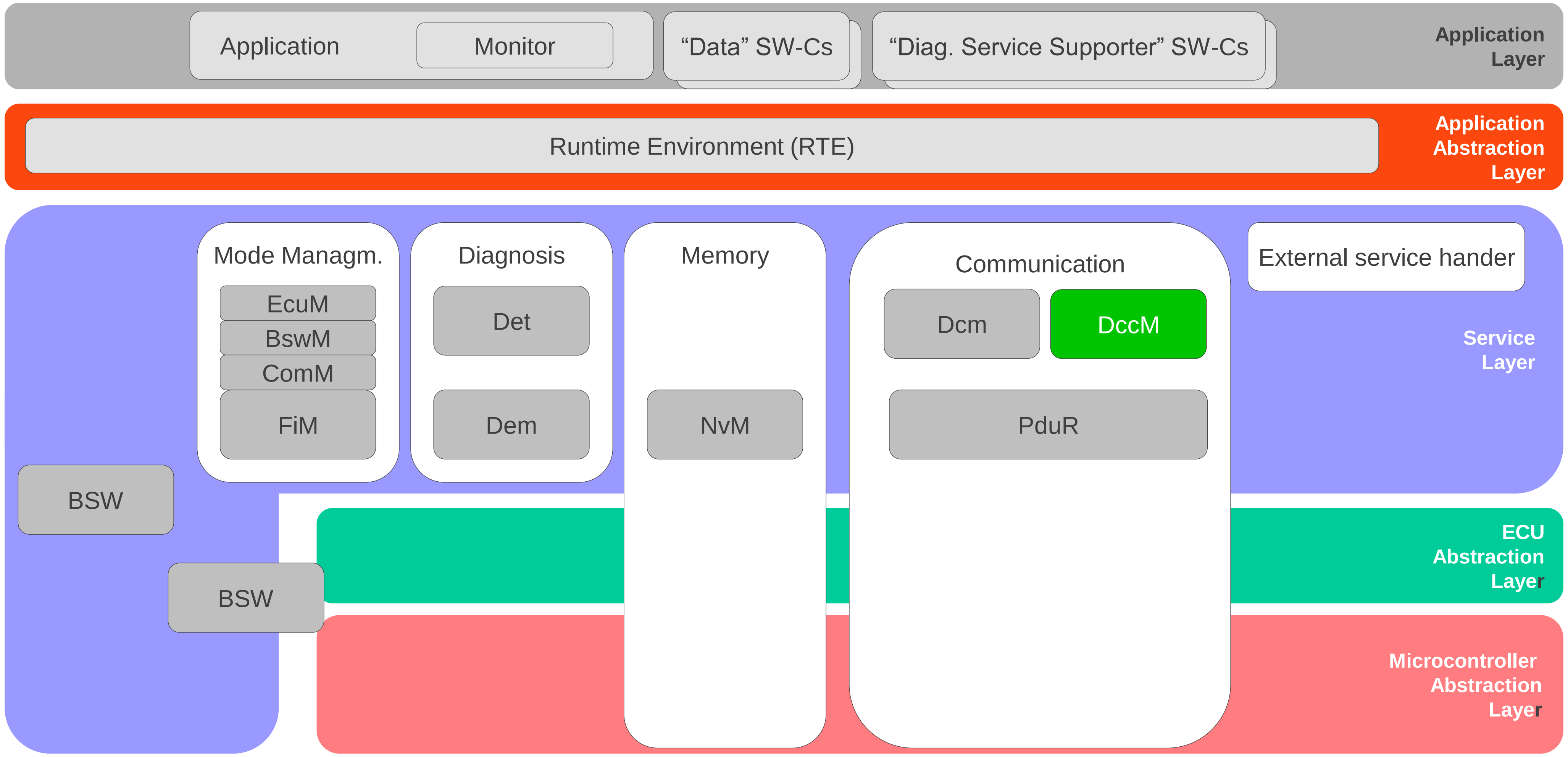
- **Independent** from network
- Implements support of the **UDS and OBD protocol** (e.g. pos./neg. response handling, physical and functional request handling, …)
- Provides UDS and OBD service handler which provides a complete or partial processing of the UDS/OBD service.
  - e.g. ReadDtcInformation sub-functions: completely implemented internally
  - e.g. ReadDataByIdentifier sub-functions: call of an application interface
- Hides **timing handling, session and security management** from the user
- Processes complete diagnostic **buffer-handling**
- Supports special UDS services like RoE or 0x2A for periodic sending of diagnostic data without requests

Dcm request verification


Request received
Service is supported
no
send NRC 'ServiceNotSupported'
yes
Service allowed in current session
no
send NRC 'ServiceNotSupportedInCurrentSession'
yes
Service allowed for current security level
no
send NRC 'SecurityAccessDenied'
yes
Dcm Resources available
no
send NRC 'BusyRepeatRequest'
yes
Call diagnostic service function

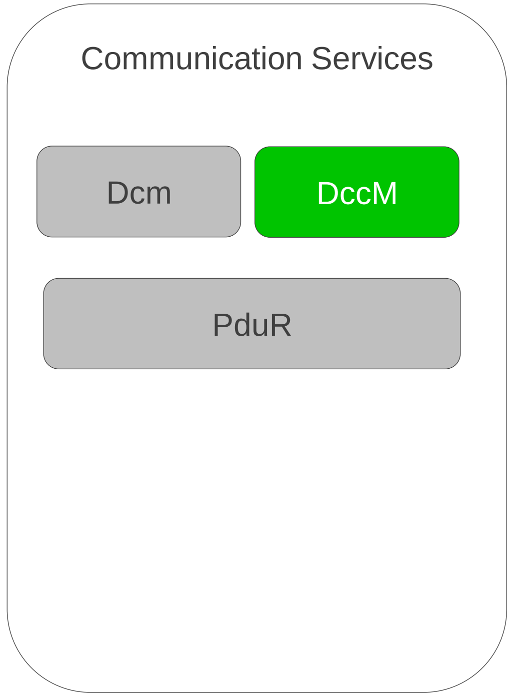
Diagnostic Client Communication Manager (DccM)

Diagnostic Client Communication Manager (DccM)



Diagnostic Client Communication Manager (DccM) - Scope

- The DccM is not defined by AUTOSAR but instead is an EB specific component implemented as an upper layer module to the PDU Router
- The Diagnostic Client Communication Manager (Dccm) module provides a UDS communication library that can speed up the development of a UDS client (Tester)
  → DccM is used to send diagnostic requests inside the vehicle network
- The DccM Library that is constantly extended with new functionalities



DccM Features

- DccM supports up to 255 **parallel** requests
- Generic send request: Any UDS payload can be built and provided to the Dccm for communication
- DccM supports both **functional** and **physical addressing**
- Functional communication channels can be configured to send a **periodic tester present** message
- DccM can be configured for **streaming**
- Functional communication with **no response from the server**
- Time-outs: DccM provides configurable **timing parameters**, i.e. P2Client, P2*Client, InternalTimeout
- Validation: DccM provides support for **validation** of response and request according to ISO 14229

Elektrobit

Summary

- Overview
- Default (Development) Error Tracer (Det)
- Diagnostic Event Manager (Dem)
- Function Inhibition Manager (FiM)
- Diagnostic Communication Manager (Dcm)
- Diagnostic Client Communication Manager (DccM)

Get in touch!

sales@elektrobit.com
www.elektrobit.com