EMMA Software Architecture Pattern for Embedded Systems

MARKEK KRAJEWSKI

whoami

- C++ (et al.)
- Networking protocols
- Client-Server
- Qt and UI
- Linux & Windows
- Embedded
- Freelancer

What is order, what is chaos?

- Garden parable
- Natural state of software is…
- The need for architectural work

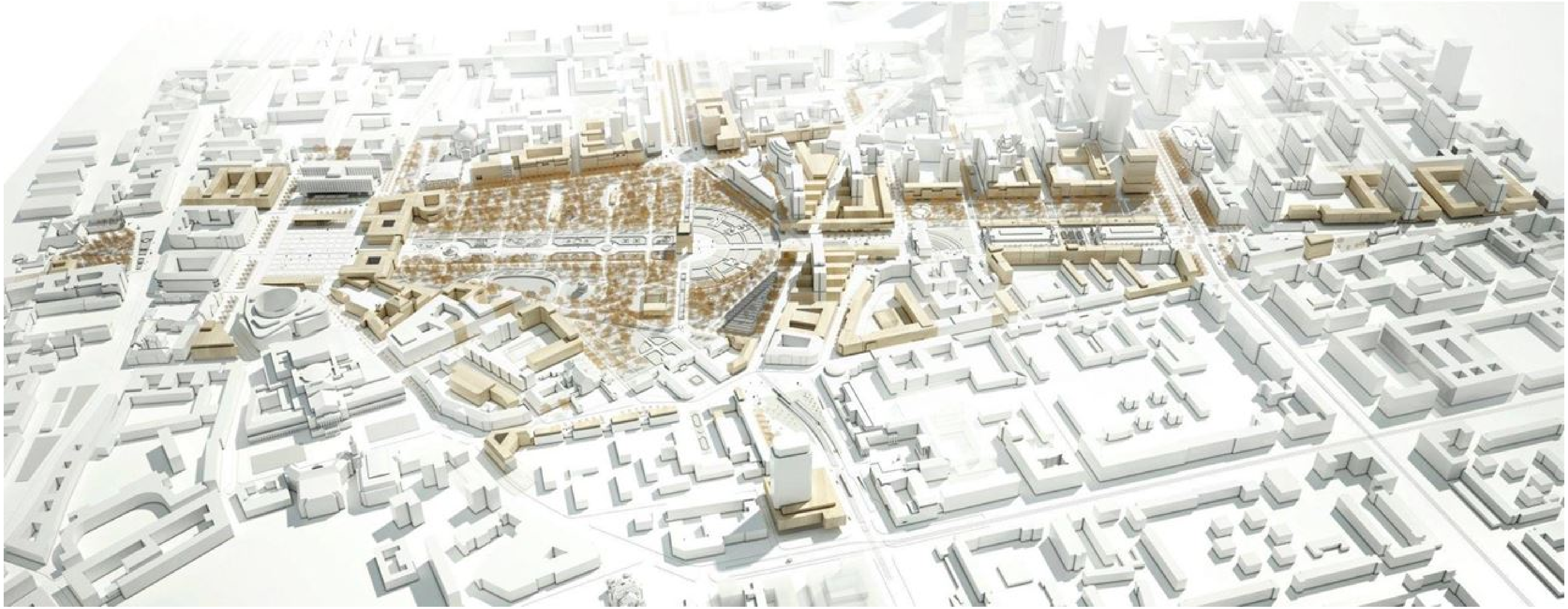
©www.mytravelingjoys.com

Architecture Pattern

> Architectural patterns are a **method of arranging blocks of functionality** to address a need.
>
> …
>
> Good pattern expressions tell you how to use them, and when, why, and **what trade-offs** to make in doing so.

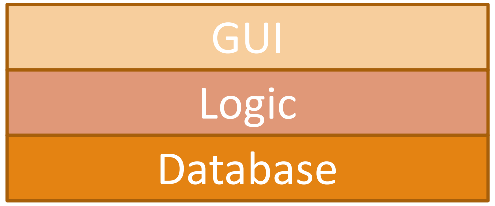
MITRE Systems Engineering Guide, 2014

City Planning – the „Axis“ Pattern

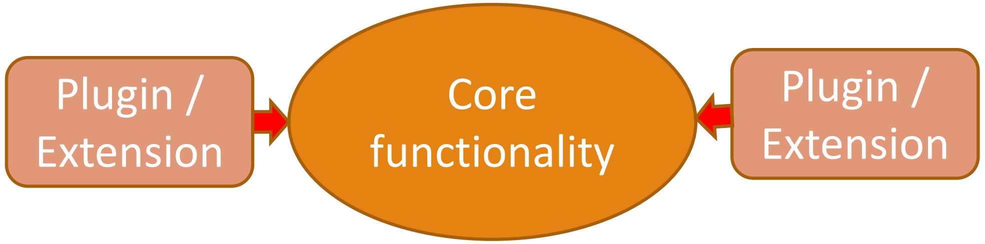


Architecture Pattern Examples

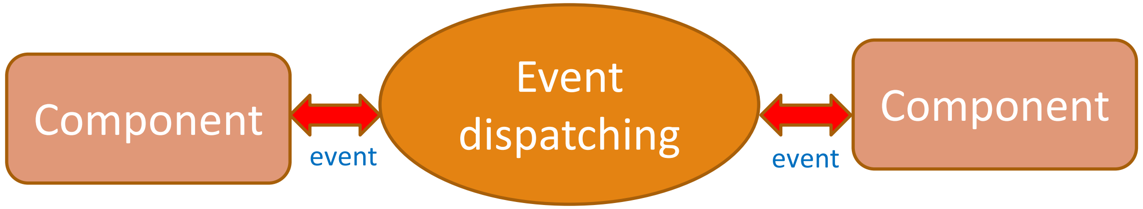
- Layered architecture pattern



- Microkernel pattern

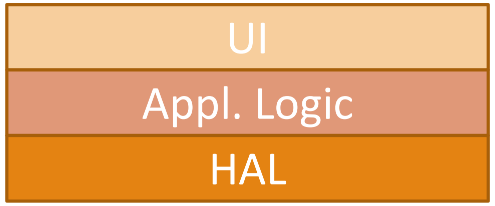


- Event-driven architecture pattern



Embedded Systems and Architecture

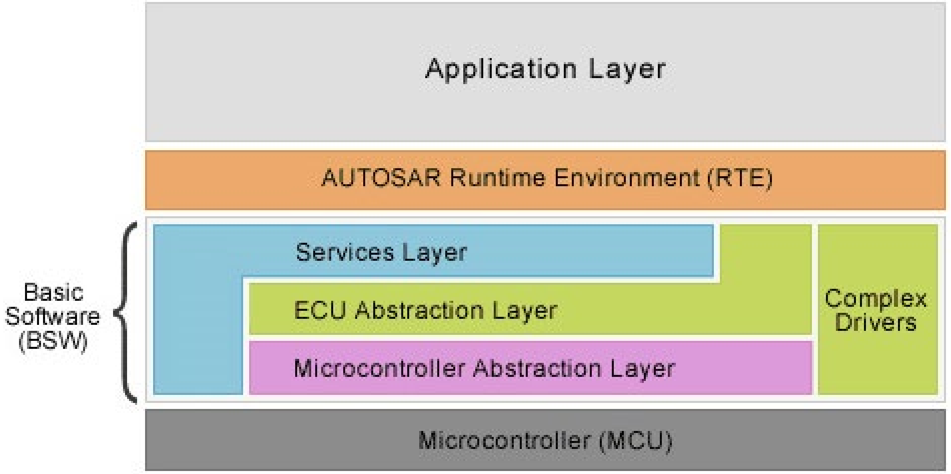
- Traditionally we are more concerned with HW architecture!
- Normally when architecting :
  - what processor, what periphery,
  - what OS, how to build the OS,
  - which graphic stack, what update mechanism, etc.
- If there is an SW architecture, than rougly the one in the upper corner:
  - Exception: AUTOSAR



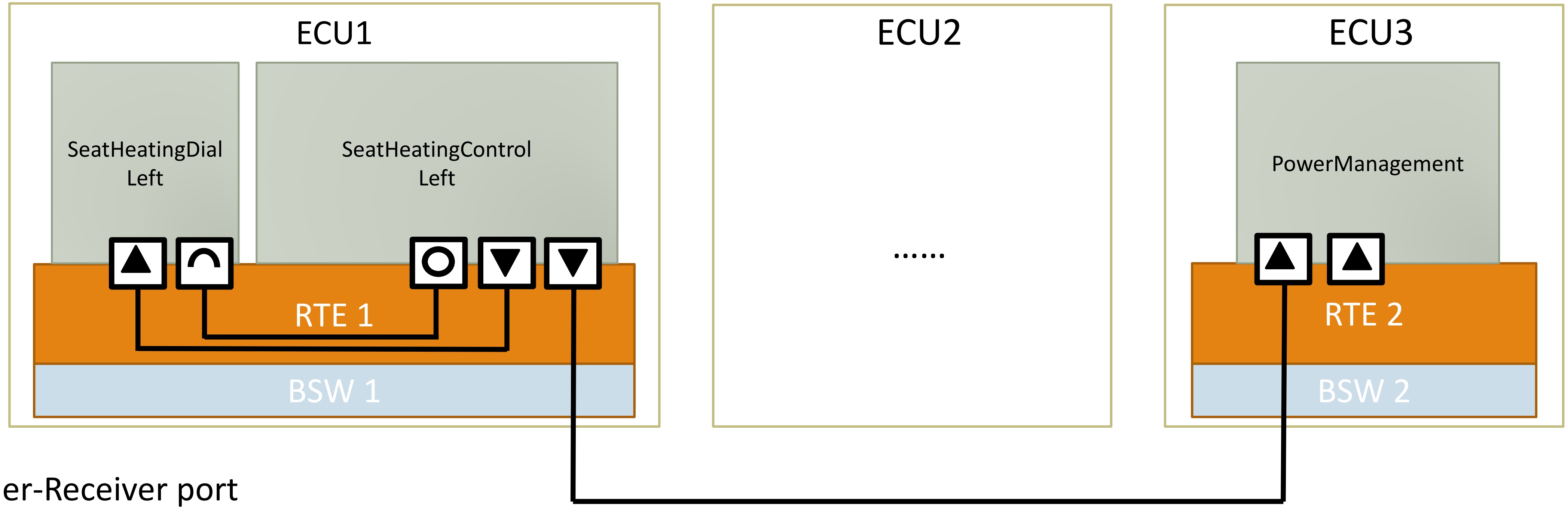
UI

Appl. Logic

HAL



AUTOSAR Architecture



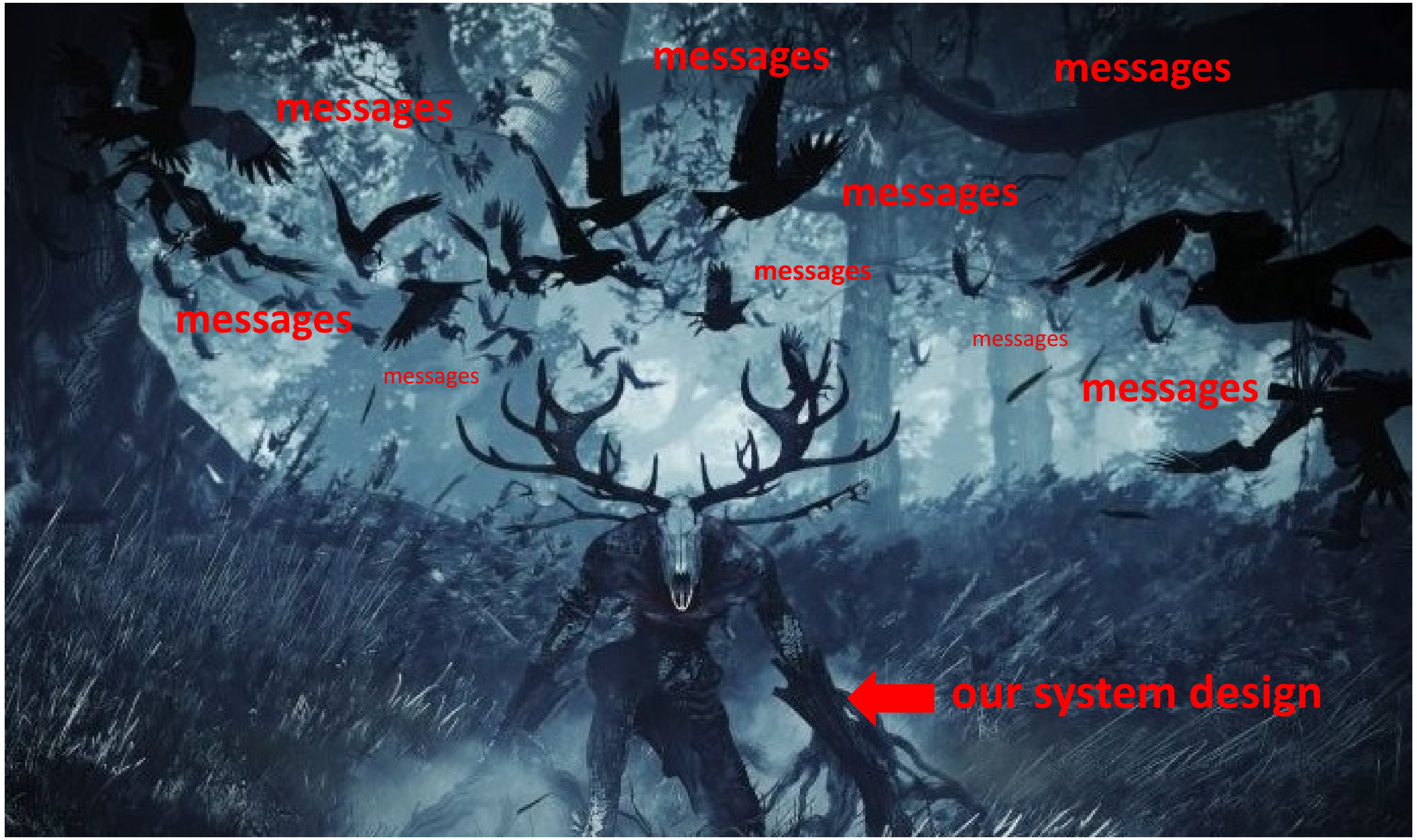
▲ Sender-Receiver port

○ Client-Server port

Cautionary (QNX) Tale

- Message Passing architecture
  - a little like AUTOSAR‘s (!)
- Many separate processes
- Decoupling (!!!)
- That can‘t be bad, right?

- Well…
  - Too many messages
  - Too many modules
  - Unclear responsibilities
  - Chaos



EMMA?

- **E** – event-driven
- **M** – multi-layered
- **M** – multi-threaded
- **A** - autonomous

What Embedded Systems?

i.e. when to use EMMA:

- Low-power 8-bit MCUs 🚫
- Cortex M0-M4 class, 32-bit MCUs (e.g. STM32F4xx), RTOS, C ✓
- Cortex A class, Linux, C++ ... 🙄 ???

Motivations

> A major motivation for developing Emma were issues **repetitively found in earlier projects.**
>
> EMMA Software Architecture Guide

Recurrent Issues

What we all observe wery often:

- Solving the same tasks again and again (but in a different way)
- Dependencies and cyclic dependencies redering modules untestable
  - (and not reusable as well!)
- Modules often using several modules from different layers
- Control flow and system structure not easily recognizable from source code
- Asynchronities/ISR hardly recognizable in source code
- Wild, unstructured threading (!!!)
- Not designed with testability in mind

Purpose

What we want to achieve:

- Clear assignment of specific tasks to a hierarchy layer (gradation of application logic).
- Comprehensible Control Flow, even across interrupt- and thread-boundaries.
- Achieve stable parallel processing by enforcing thread-safety and regarding ISR restrictions.
- Improved maintainability of code.
- Improved reusability of modules between projects.
- Provide a high level of testability.

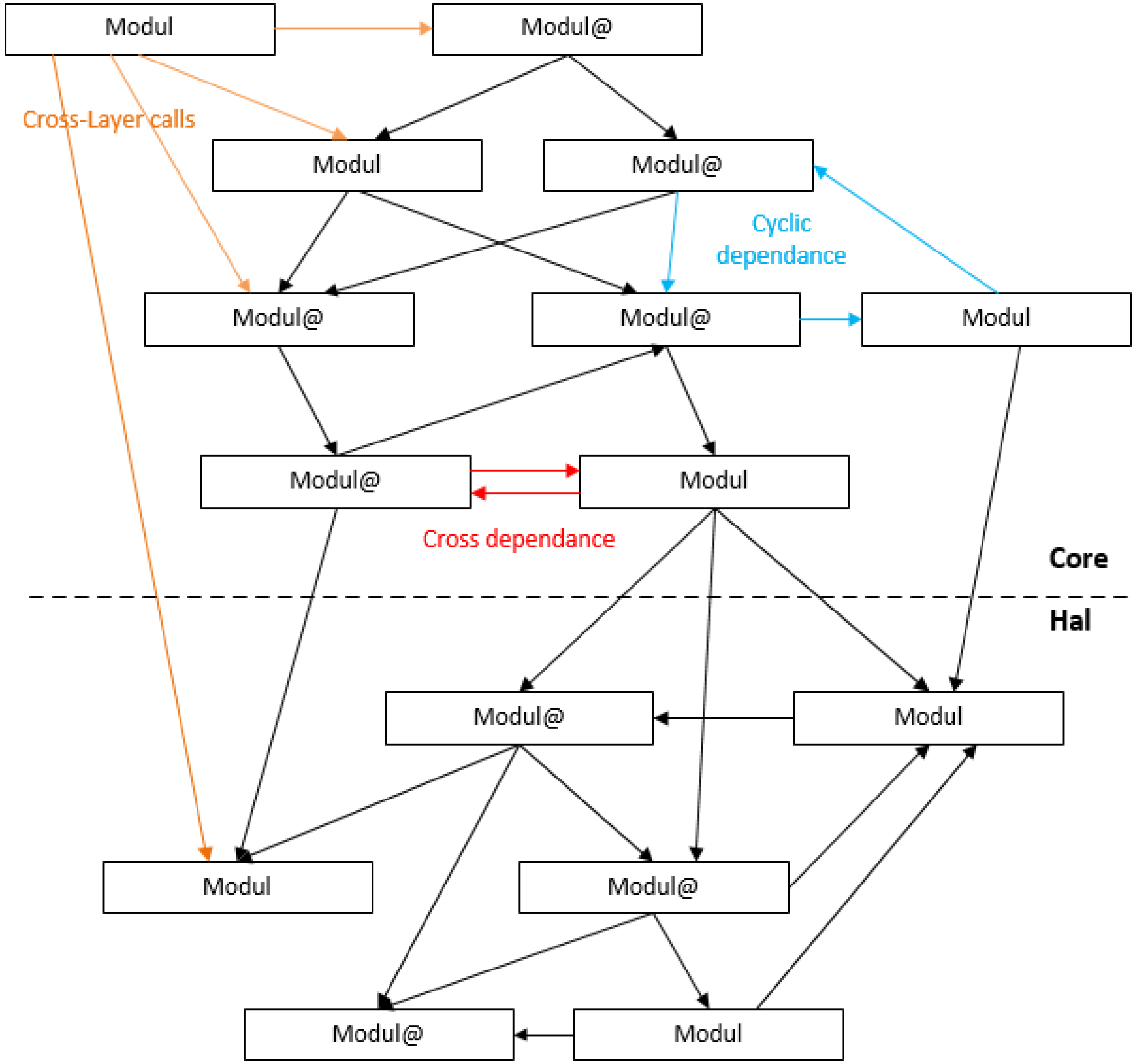
Trade-offs

The price we are willing to pay:

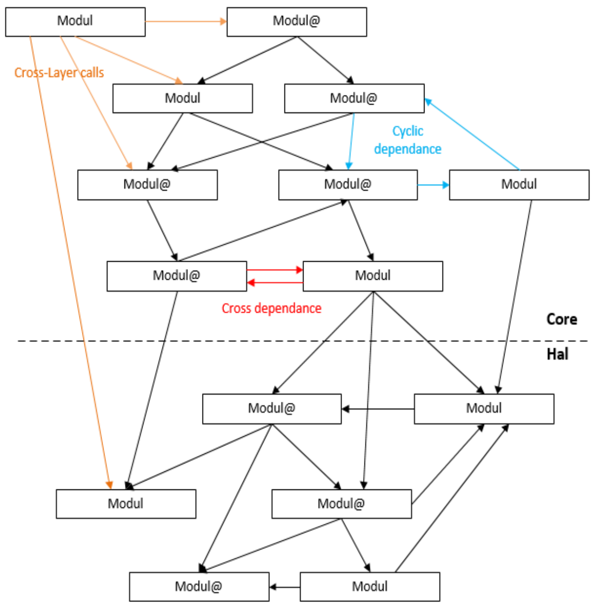
- Empty, forwarding-only modules may be required to keep the layer structure in order.
- Higher effort for putting up the skeleton of a new project.
- More discipline while designing and coding.
- Requires more MCU resources (RAM, Flash…)

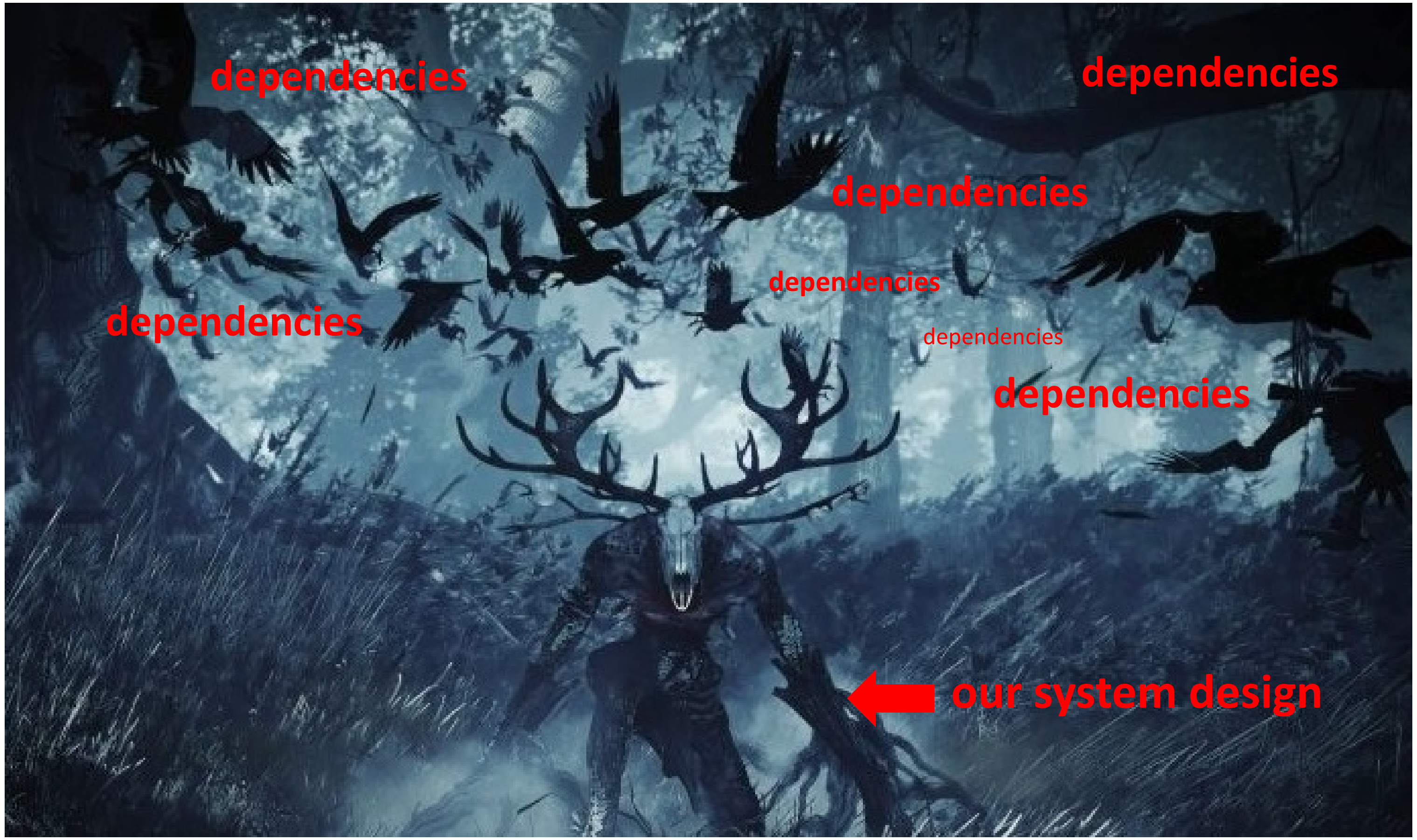
Software designs EMMA wants to avoid

- Cross-layer calls
- Intra-layer dependencies
- Cyclic dependencies
- Unrestricted threading

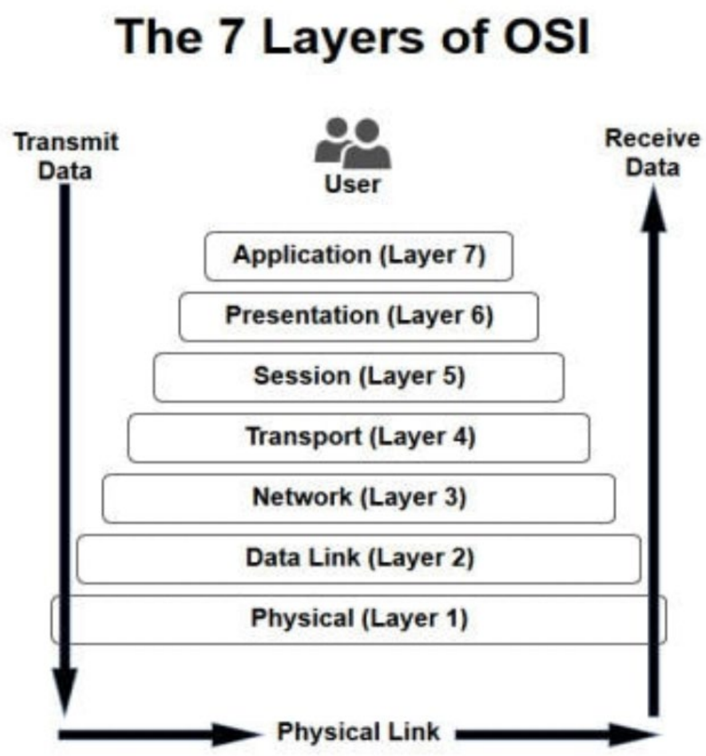


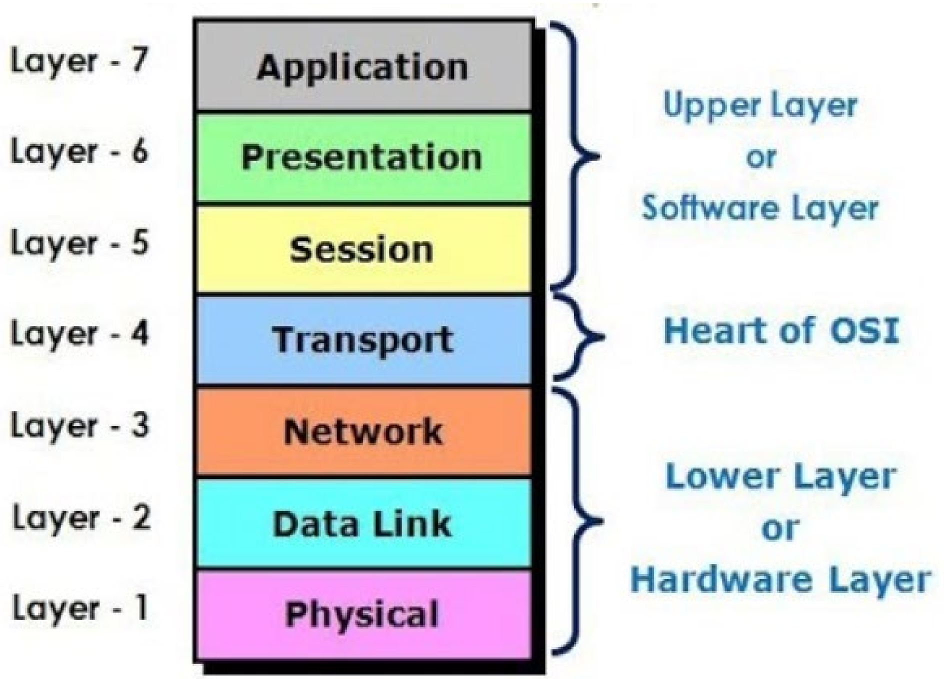
Software designs we want to avoid



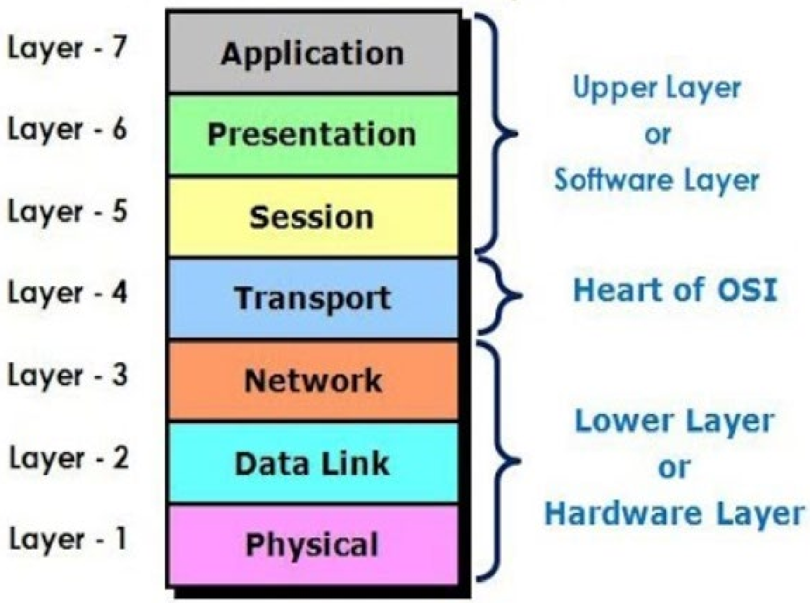


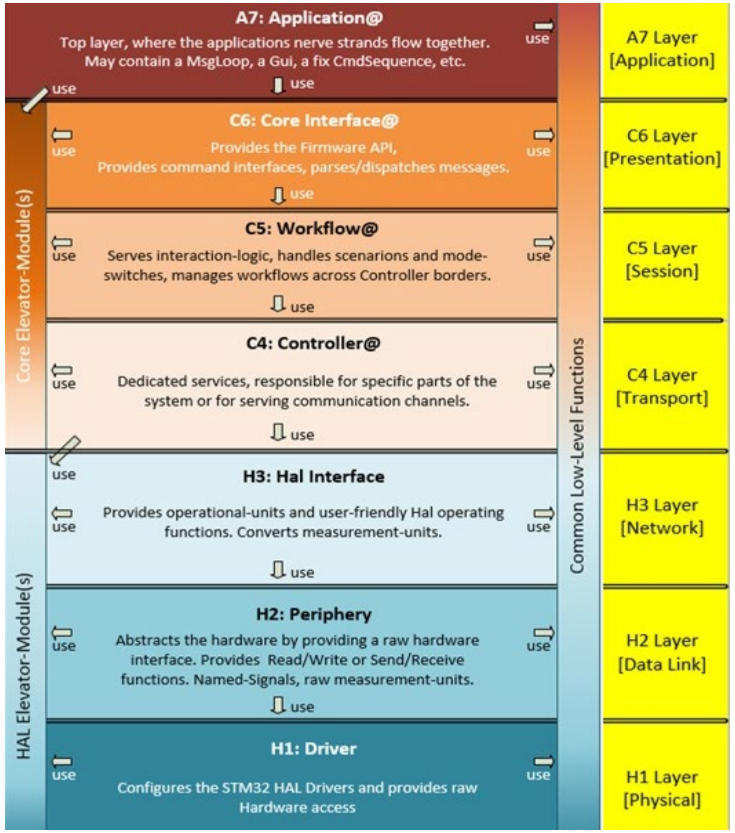
Inspiration





Inspiration 2





A7: Application@

Top layer, where the applications nerve strands flow together. May contain a MsgLoop, a Gui, a fix CmdSequence, etc.

C6: Core Interface@

Provides the Firmware API,
Provides command interfaces, parses/dispatches messages.

C5: Workflow@

Serves interaction-logic, handles scenarions and mode-switches, manages workflows across Controller borders.

C4: Controller@

Dedicated services, responsible for specific parts of the system or for serving communication channels.

H3: Hal Interface

Provides operational-units and user-friendly Hal operating functions. Converts measurement-units.

H2: Periphery

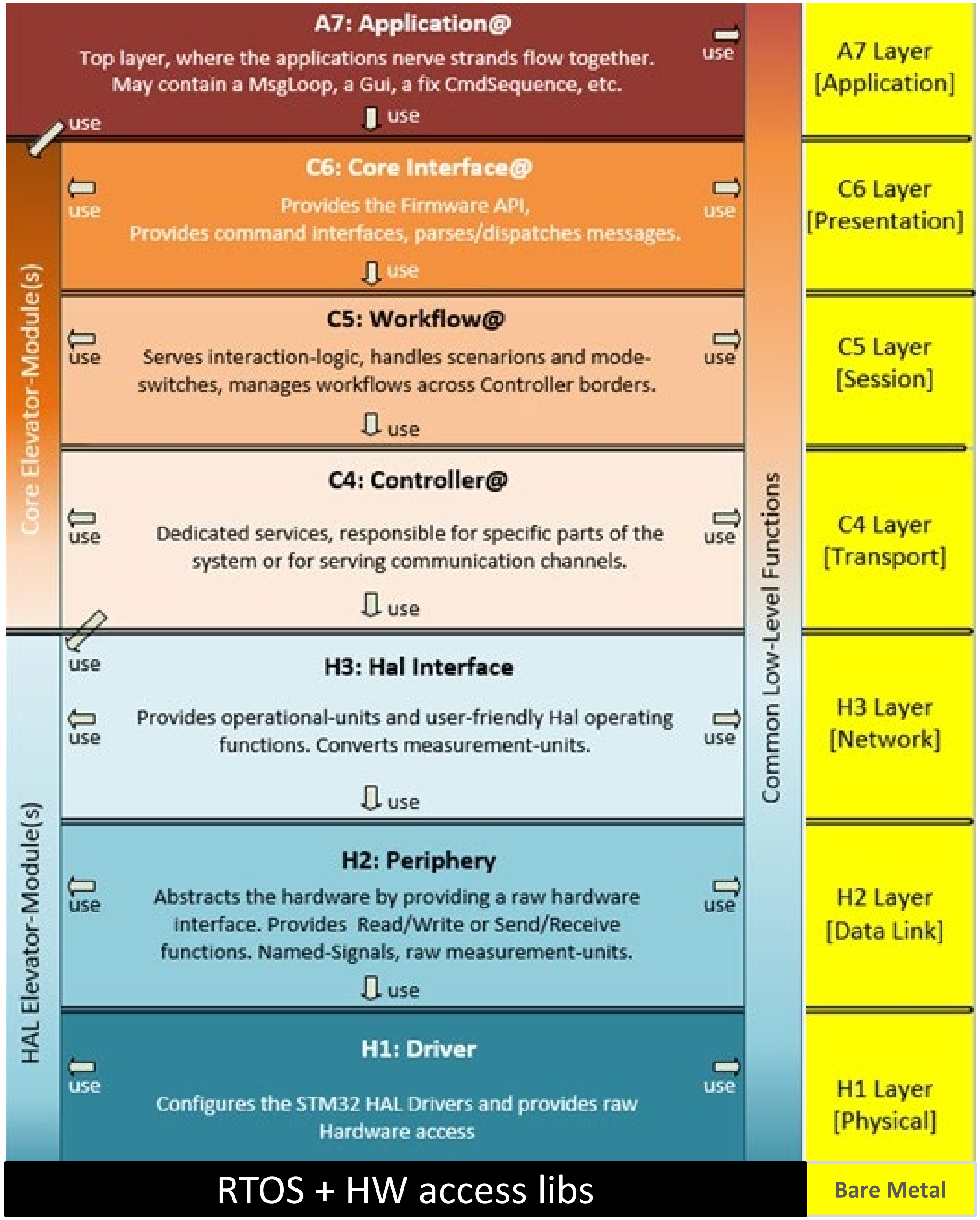
Abstracts the hardware by providing a raw hardware interface. Provides Read/Write or Send/Receive functions. Named-Signals, raw measurement-units.

H1: Driver

Configures the STM32 HAL Drivers and provides raw Hardware access

EMMA Overview

- Layers
- Elevators
- Infrastucture and Utilities



A7: Application@
Top layer, where the applications nerve strands flow together. May contain a MsgLoop, a Gui, a fix CmdSequence, etc.

C6: Core Interface@
Provides the Firmware API, Provides command interfaces, parses/dispatches messages.

C5: Workflow@
Serves interaction-logic, handles scenarions and mode-switches, manages workflows across Controller borders.

C4: Controller@
Dedicated services, responsible for specific parts of the system or for serving communication channels.

H3: Hal Interface
Provides operational-units and user-friendly Hal operating functions. Converts measurement-units.

H2: Periphery
Abstracts the hardware by providing a raw hardware interface. Provides Read/Write or Send/Receive functions. Named-Signals, raw measurement-units.

H1: Driver
Configures the STM32 HAL Drivers and provides raw Hardware access

RTOS + HW access libs

A7 Layer [Application]

C6 Layer [Presentation]

C5 Layer [Session]

C4 Layer [Transport]

H3 Layer [Network]

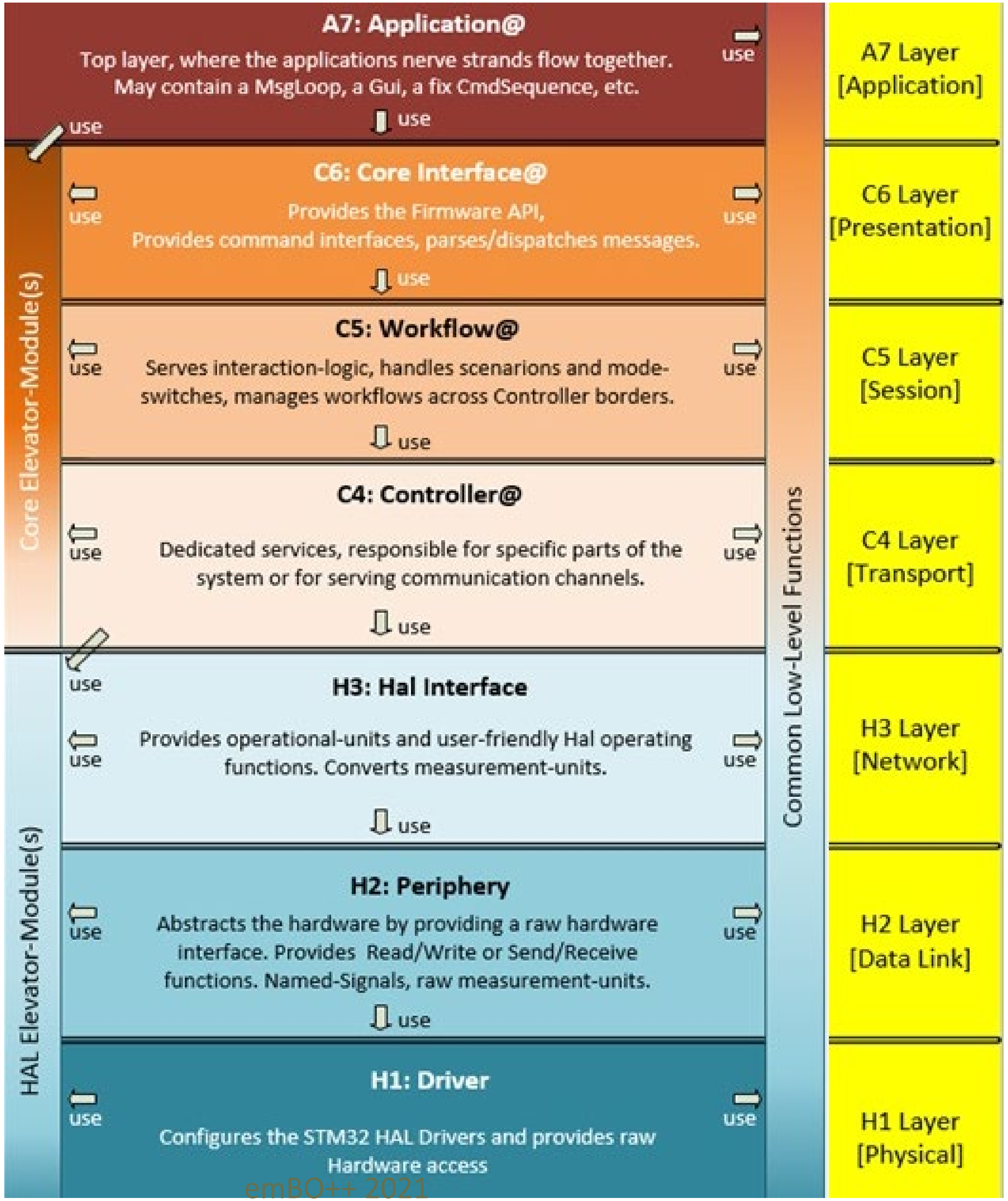
H2 Layer [Data Link]

H1 Layer [Physical]

Bare Metal

EMMA Rules

- Autonomous modules
  - Only dependent on lower modules and elevators!
- Module‘s **input** comes from an upstream module (or elevator…)
- Module‘s **output** goes to a downstream module
- **ALL** interrupts are transported over a single elevator
- **ALL** events are transported over a single elevator too
- Naming conventions (!)



A7: Application@
Top layer, where the applications nerve strands flow together.
May contain a MsgLoop, a Gui, a fix CmdSequence, etc.

C6: Core Interface@
Provides the Firmware API,
Provides command interfaces, parses/dispatches messages.

C5: Workflow@
Serves interaction-logic, handles scenarions and mode-switches, manages workflows across Controller borders.

C4: Controller@
Dedicated services, responsible for specific parts of the system or for serving communication channels.

H3: Hal Interface
Provides operational-units and user-friendly Hal operating functions. Converts measurement-units.

H2: Periphery
Abstracts the hardware by providing a raw hardware interface. Provides Read/Write or Send/Receive functions. Named-Signals, raw measurement-units.

H1: Driver
Configures the STM32 HAL Drivers and provides raw Hardware access

Not (!) part of EMMA

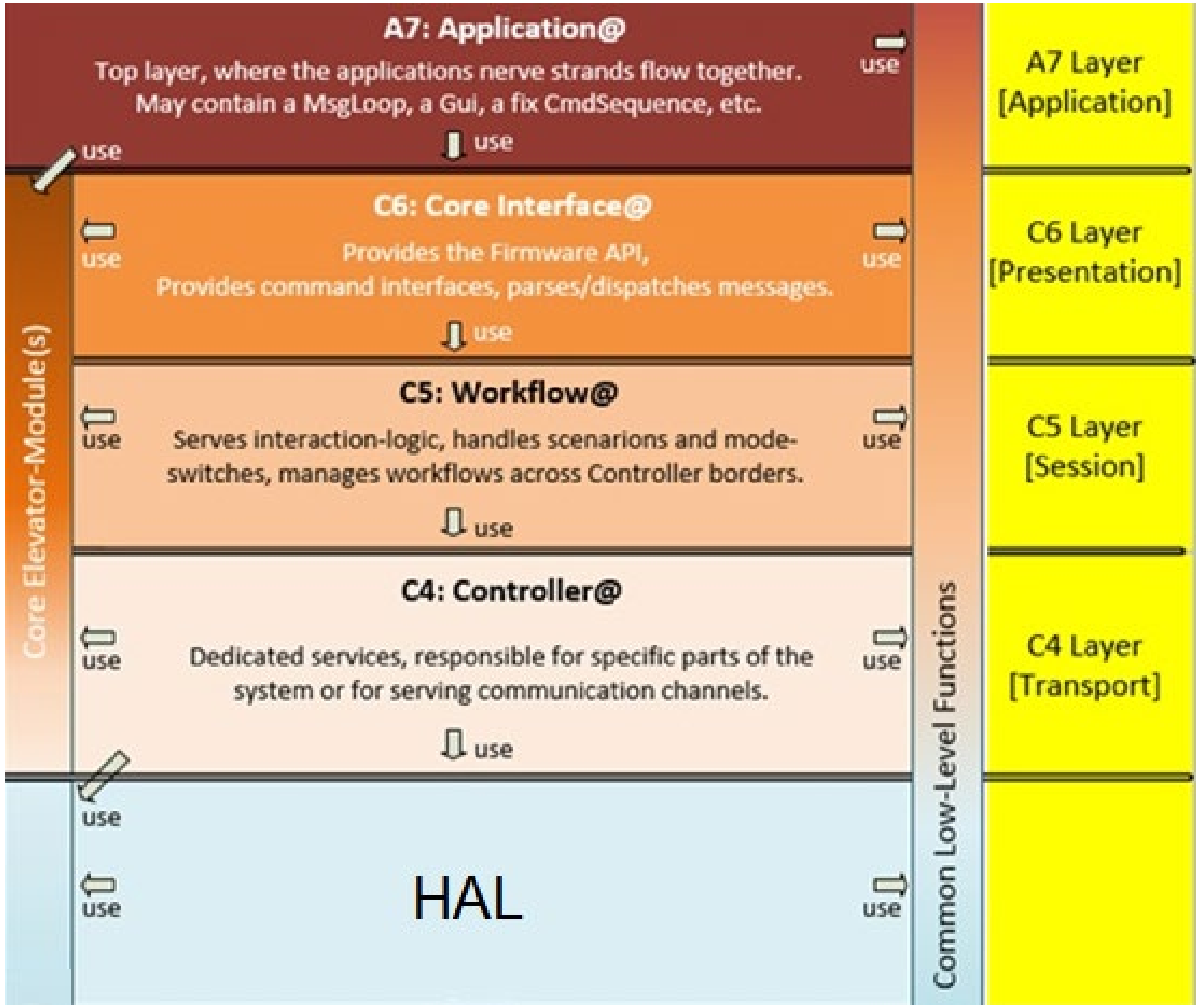
- Error handling
- Data persistency and data structures
- GUI (???)
- Maintenance & Upgrades
- Internalization and Localization

Layering

- Autonomous modules
  - Can work stand-alone once lower modules and elevators are initialized
  - Testable in isolation using stubs/mocks for lower modules
- Each module needs an *init()* and a *cleanup()* function
  - The capability of cleaning up an Emma module is essential for making isolated tests possible!
- In principle, each layer is **optional**!
- The *“Periphery”* layer is the layer on which MOCK-modules should be placed in Unit Test projects to simulate real hardware
  - Unit Tests in simulator-environment become possible for all higher-layer modules and Elevators!
- On the other hand, the *„Periphery“* layer with REAL modules in place, is the ideal entry point for automated, integrated Periphery-, Driver- and hardware tests.
- Threading only allowed in specific layers!

Threading

- Only in Core and Application layers!
- Allowed, but nor required



A7: Application@

Top layer, where the applications nerve strands flow together. May contain a MsgLoop, a Gui, a fix CmdSequence, etc.

C6: Core Interface@

Provides the Firmware API, Provides command interfaces, parses/dispatches messages.

C5: Workflow@

Serves interaction-logic, handles scenarions and mode-switches, manages workflows across Controller borders.

C4: Controller@

Dedicated services, responsible for specific parts of the system or for serving communication channels.

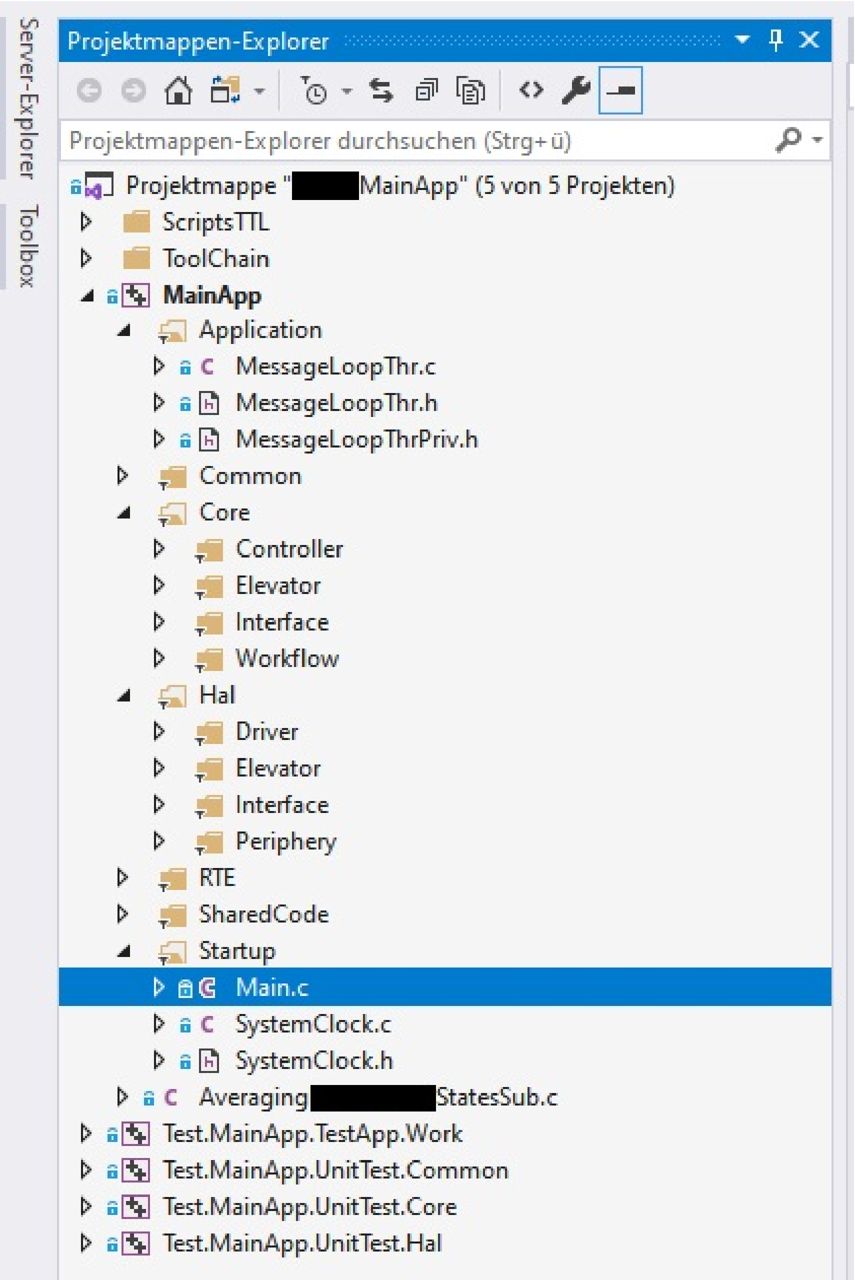
HAL

EMMA naming conventions

- The source file structure of an Emma application should reflect the layer to which a specific module belongs to.
- However, there’s **no need** to name the folders **exactly** as the layer-name is, but the layer must be recognizable from the folder-structure:
  - *MyProject/Source/Application/…*
  - *MyProject/Source/CoreInterface/…*
  - *MyProject/Source/Core Interface/…*
  - *MyProject/Source/Core/Interface/…*
  - *MyProject/Source/HalElevator/…*
  - *MyProject/Source/Hal/Elevator/…*

EMMA project structure

- 3 main parts - Application, Core, HAL
- Startup – main() function
- Common – error handling, utilities
- RTE – STM32f4xx library, RTOS config
- SharedCode – data stuctures, message parsing

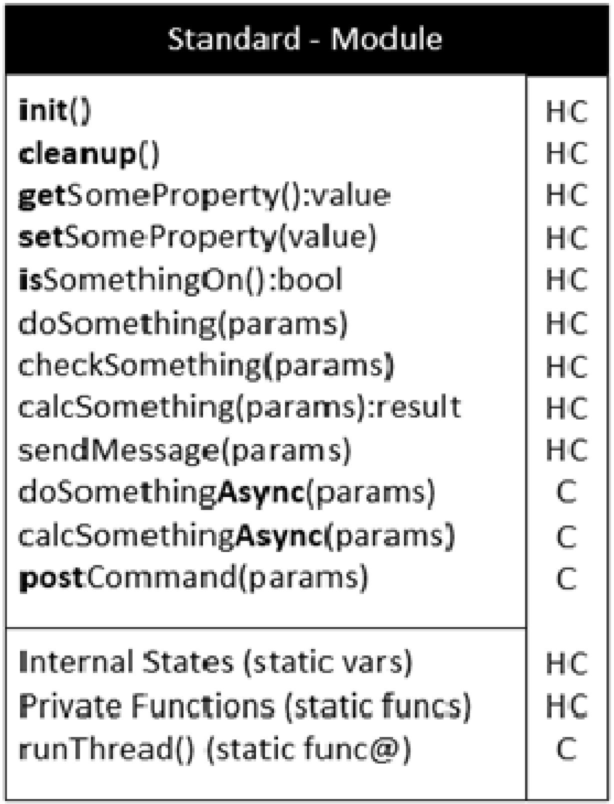


```
Projektmappe "[redacted]MainApp" (5 von 5 Projekten)
  ScriptsTTL
  ToolChain
  MainApp
    Application
      MessageLoopThr.c
      MessageLoopThr.h
      MessageLoopThrPriv.h
    Common
    Core
      Controller
      Elevator
      Interface
      Workflow
    Hal
      Driver
      Elevator
      Interface
      Periphery
    RTE
    SharedCode
    Startup
      Main.c
      SystemClock.c
      SystemClock.h
    Averaging[redacted]StatesSub.c
  Test.MainApp.TestApp.Work
  Test.MainApp.UnitTest.Common
  Test.MainApp.UnitTest.Core
  Test.MainApp.UnitTest.Hal
```

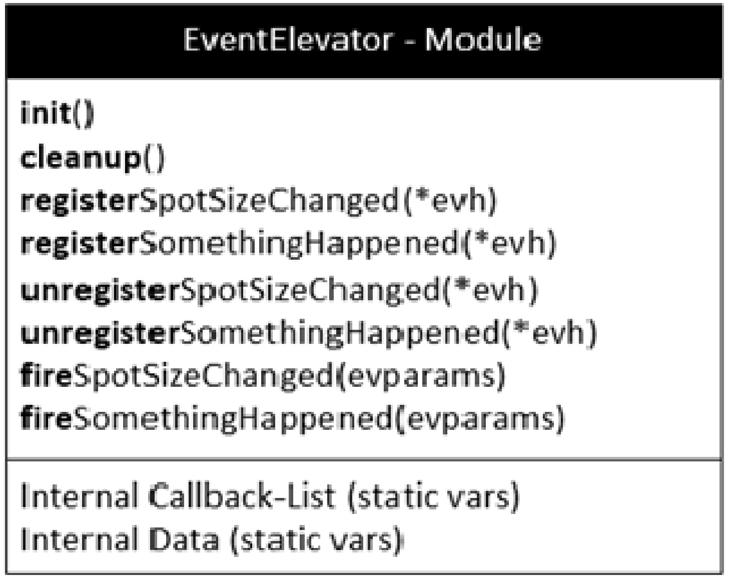
EMMA naming conventions contd.

- The names of modules which own a thread, shall end with „**Thr**“
- The names of functions which directly trigger an asynchronous operation, shall end with “**Async**”;
- Getter/Setter- functions shall start with „**get**“ or „**is**“ (for ‘bool’ type only) and „**set**“.
- The name of a Interrupt Service Routine shall have the postfix “**_Isr**”
  - The postfix shall clearly emphasize that the function runs on an interrupt.
- The name of Event Handler shall have the postfix “**_Evh**”
  - The postfix shall clearly emphasize that the function runs on a foreign thread.
- The name of a Callback-function shall have the postfix “**_Cbk**”
  - The postfix shall clearly emphasize that the function (usually) runs on a foreign thread.
- The functions for initializing and de-initializing are named “**init**” and “**cleanup**”.
- Events are „**fire**d“, Interrupts are „**raise**d“.

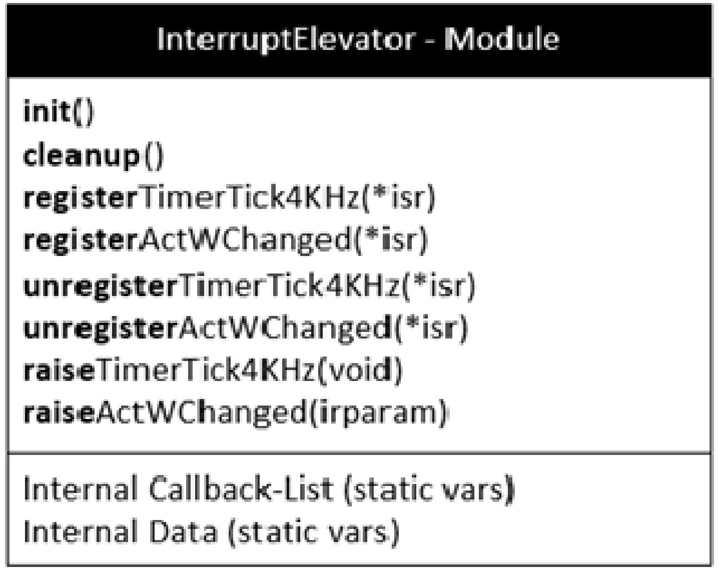
EMMA module examples



Standard - Module	
init()	HC
cleanup()	HC
getSomeProperty():value	HC
setSomeProperty(value)	HC
isSomethingOn():bool	HC
doSomething(params)	HC
checkSomething(params)	HC
calcSomething(params):result	HC
sendMessage(params)	HC
doSomething**Async**(params)	C
calcSomething**Async**(params)	C
postCommand(params)	C
Internal States (static vars)	HC
Private Functions (static funcs)	HC
runThread() (static func@)	C

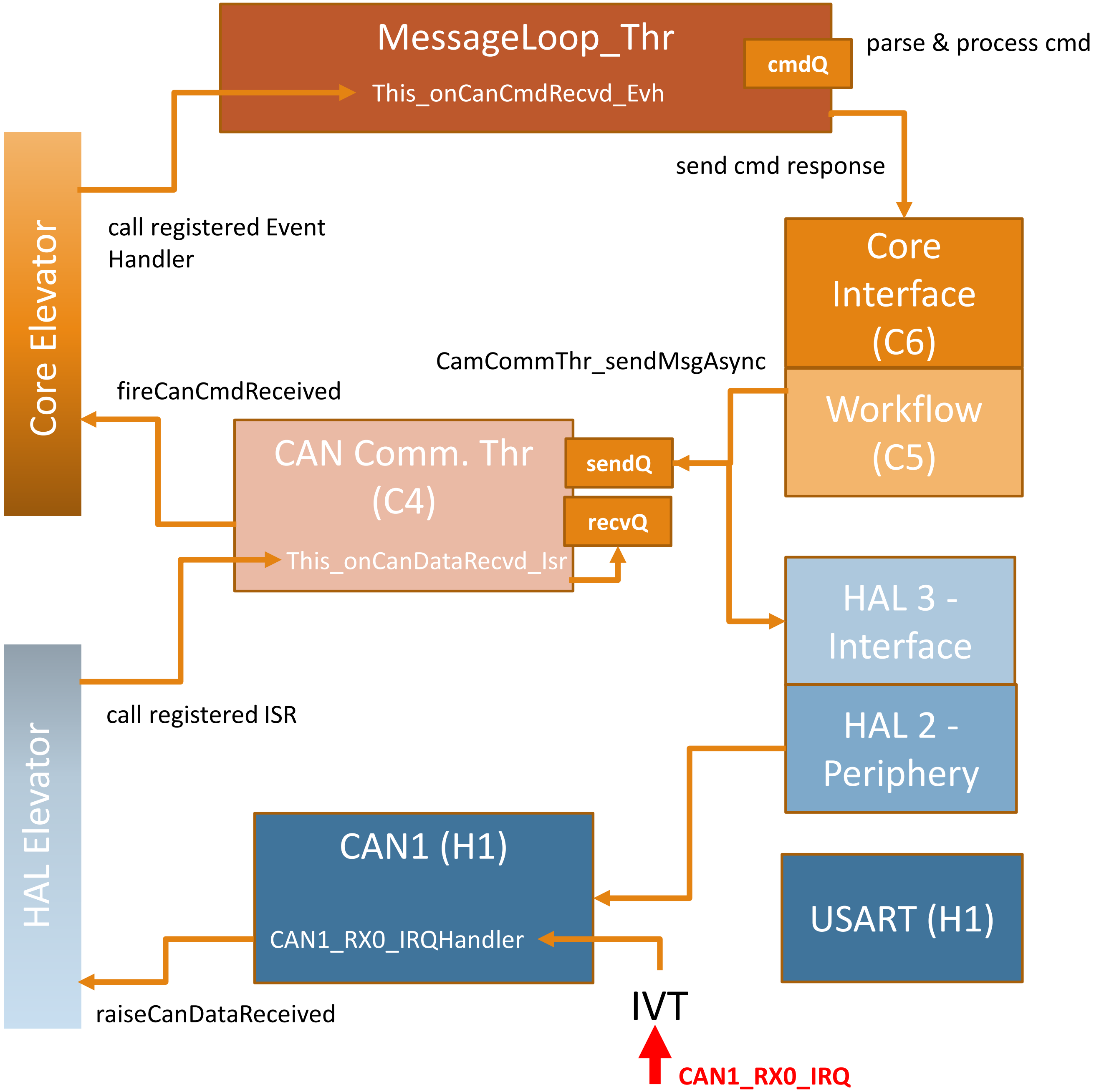


EventElevator - Module
init()
cleanup()
registerSpotSizeChanged(*evh)
registerSomethingHappened(*evh)
unregisterSpotSizeChanged(*evh)
unregisterSomethingHappened(*evh)
fireSpotSizeChanged(evparams)
fireSomethingHappened(evparams)
Internal Callback-List (static vars)
Internal Data (static vars)



InterruptElevator - Module
init()
cleanup()
registerTimerTick4KHz(*isr)
registerActWChanged(*isr)
unregisterTimerTick4KHz(*isr)
unregisterActWChanged(*isr)
raiseTimerTick4KHz(void)
raiseActWChanged(irparam)
Internal Callback-List (static vars)
Internal Data (static vars)

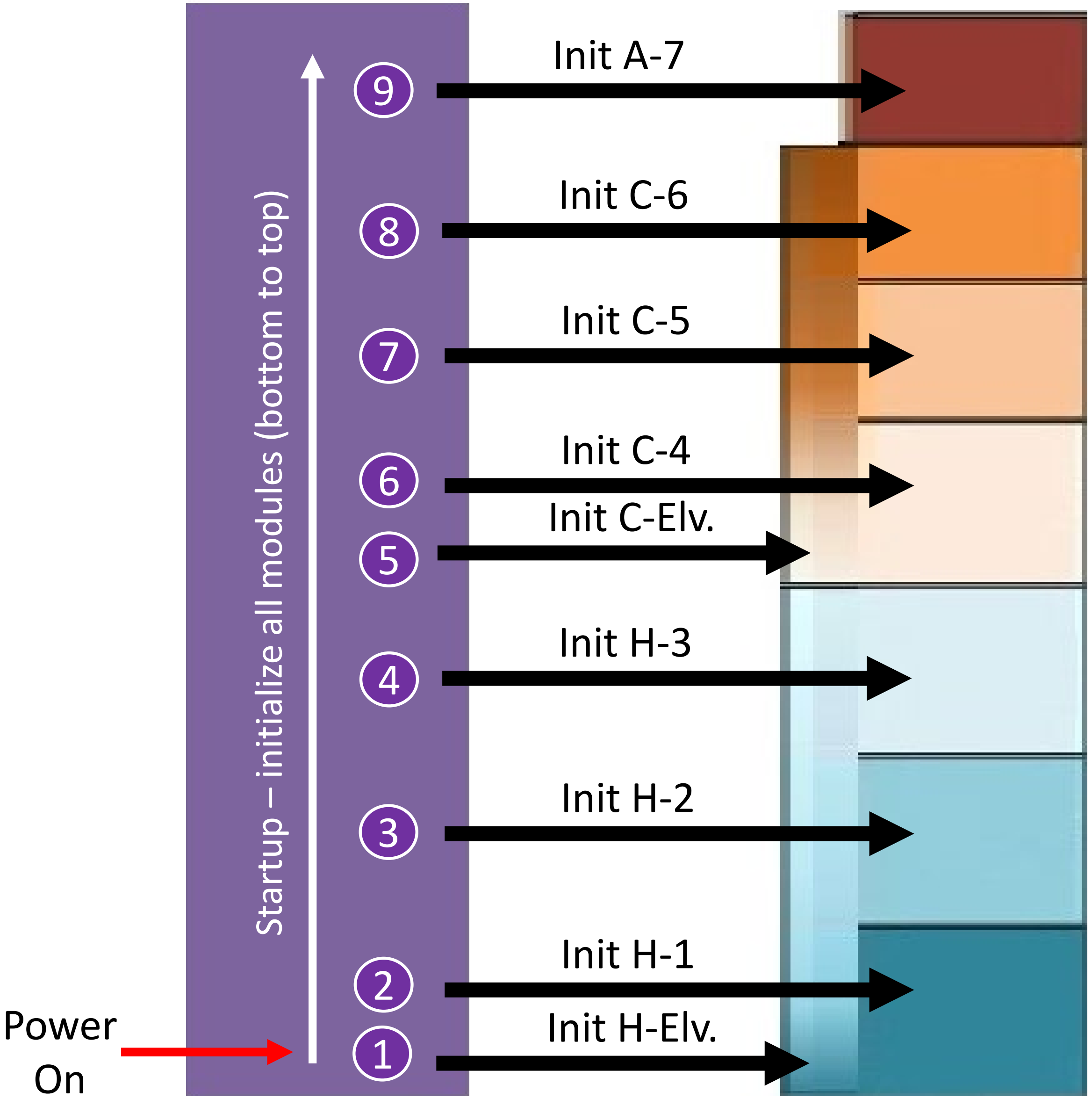
EMMA control flow



ib-krajewski.de

EMMA Startup

- Usually imlemented in the **main()** function
- Allowed to to access **all** modules in **any** layer (!!!)
- Even direct HW access or assembler if required (!!!)



ib-krajewski.de

Reality check

> Welcome to reality!
>
> For good reasons, you are **EVER free to deviate** from Emma.
>
> But before you do this: Think about your problem twice! Is it really such a very special use-case?

EMMA SOFTWARE ARCHITECTURE GUIDE

Discussion

How to evaluate an architecture?

- Six-pack architectural test ☺
- Architectural SCARS (Grady Booch)
  - **S**eparation of **C**oncerns, **A**bstractions, balaced **R**esponsibilities, **S**implify

How well does EMMA here?

- SCARS
- Code readibility
- Testability

Usage Case 1

BOOTLOADER FOR A MICROCONTROLLER

Bootloader on a Device Control MCU

The inteded use case!

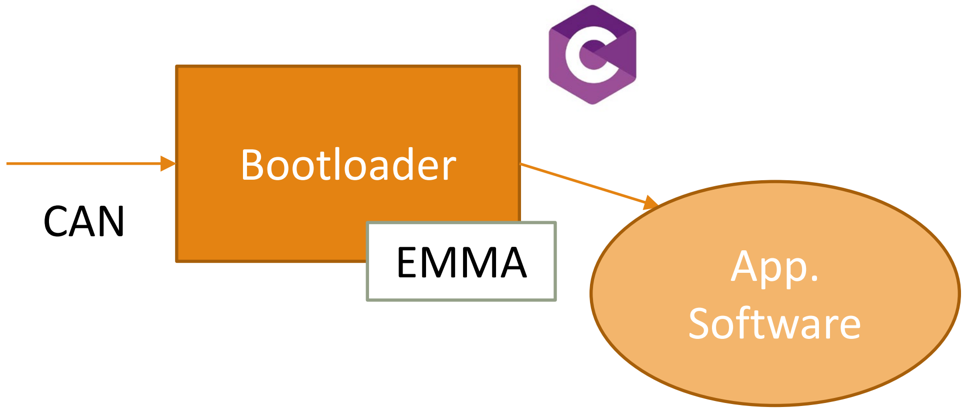
Because:

- STM32F4xx microcontroller with RTOS

Responsibilities:

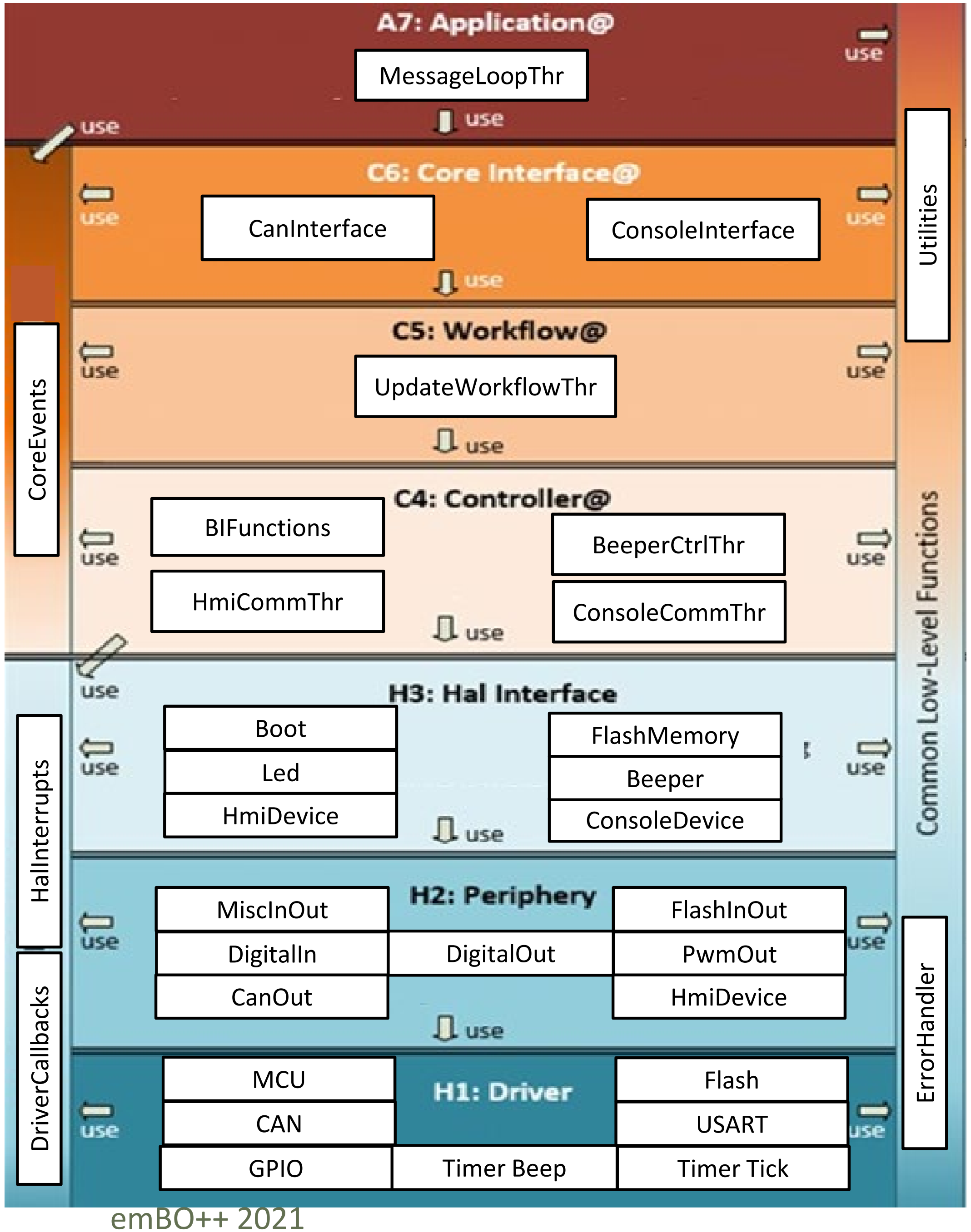
- start the Appl. Software on the MCU

- and: implement Appl. Software upgrade



Elements of Bootloader

- HW acces via and ST-Library (STM32F4xx_Lib)
- HAL and Core each with respectively 3 layers
- No direct HW access above H1!
- RTOS provides threading
- 2 types of HAL elevators



ib-krajewski.de

HAL Elevators

Interrupts:

```
void HalInterrupts_registerConsoleDataReceived(HalInterrupts_FunctionConsoleDataReceived_t pfuncIsr)
{
  This_registerISR( (PVOID)pfuncIsr, (PTRARRAY)m_aIsrSerialDataReceived, HALINTERRUPTS_CONSOLEDATARECEIVED_MAXSUBSCRIBERS);
}

void HAL_CAN_RxCpltCallback(CAN_HandleTypeDef* hcan)
{
   // Iterate through the ISR-table and call every registered ISR function
   for (int32_t i = 0; i < DRIVERCALLBACKS_HALCANRXCPLT_MAXSUBSCRIBERS; i++)
   {
     DriverCallbacks_FunctionHalCanRxCplt_t pfuncIsr = m_aIsrHalCanRxCplt[i];
     if (pfuncIsr != NULL) { pfuncIsr(hcan); }
   }
 }
```

Driver Callbacks:

```
void DriverCallbacks_registerHalUartError(DriverCallbacks_FunctionHalUartError_t pfuncIsr)
{
  This_registerCallback( (PVOID)pfuncIsr, (PTRARRAY)m_aIsrHalUartError, DRIVERCALLBACKS_HALUARTERROR_MAXSUBSCRIBERS);
}
```

Core Elevators

Core Events:

```
void CoreEvents_fireConsoleMsgReadyForSend(const char* pszConsoleMsgOut)
{
  // Iterate through the EvH-table and call every registered event handler function.
  for (int32_t i = 0; i <   COREEVENTS_CONSOLEMSGREADYFORSEND_MAXSUBSCRIBERS; i++)
  {
    CoreEvents_FunctionConsoleMsgReadyForSend_t pfuncEvh = m_aEvhConsoleMsgReadyForSend[i];
    if (pfuncEvh != NULL) { pfuncEvh(pszConsoleMsgOut); }
  }
}
```

Core Properties:

```
void CoreProps_setBeeperMute(const CoreTypes_EnumPropVal_t enumWhich, const bool bIsMute);

bool CoreProps_isBeeperMute(const CoreTypes_EnumPropVal_t enumWhich);
```

Startup without Appl. Software

```
int main(void)
{
    if (This_isUserAppSelected())
    {
        // start the App. Software  ...
    }
    else
    {
        // HAL Layer-1: Driver
        USART1_init();
        CAN1_init();
        Timer1_init();

        // HAL Layer-2: Periphery
        SerialOut_init();
        CanOut_init();
        DigitalOut_init();
        PwmOut_init();

        // HAL Layer-3: Hal-Interface
        ConsoleDevice_init();
        HmiDevice_init();
        Led_init();
        Beeper_init();

        // Core Layer-4: Controller
        ConsoleCommThr_init();
        HmiCommThr_init();
        BeeperCtrlThr_init();

        // Core Layer-5: Workflow
        UpdateWfThr_init();

        // Core Layer-6: Core-Interface
        ConsoleInterface_init();
        CgCanInterface_init();

        // Layer-7: Application
        MessageLoopThr_init();
    }

    return 0;
}
```

Startup loading App. Software

- We have to load the modules we will need to start the Application!

```
int main(void)
{
     // 1.  ensure, that vital system components like RTOS, HAL-library, etc. are available.
     This_systemCheckOrHaltOnFailure();

     // 2. initialize all modules required for checking and booting the UserApp.

     DriverCallbacks_init();      // HAL elevator for Drivers-only
     HalInterrupts_init();        // HAL elevator for distributing interrupts
     MCU_init();                  // HAL driver for the microcontroller unit.

     Flash_init();                // HAL driver for the onboard-flash.
     GPIO_init();                 // HAL driver for GPIO-pins.

     MiscInOut_init();            // HAL periphery module required for MCU access.
     FlashInOut_init();           // HAL periphery module required for Flash access.
     DigitalIn_init();            // HAL periphery module required for DigitalIn access

     Boot_init();                 // HAL interface module providing Boot-functions.
     FlashMemory_init();          // HAL interface module providing FlashMemory-functions.

     CoreEvents_init();           // Core elevator for distributing events.
     BlFunctions_init();          // Core controller containing high-level bootloader functions.

     if (This_isUserAppSelected ())
     {
        // Jump to the UserApp, never return.
        MCU_jumpToUserApp();
     }
     else …
```

Usage Case 2

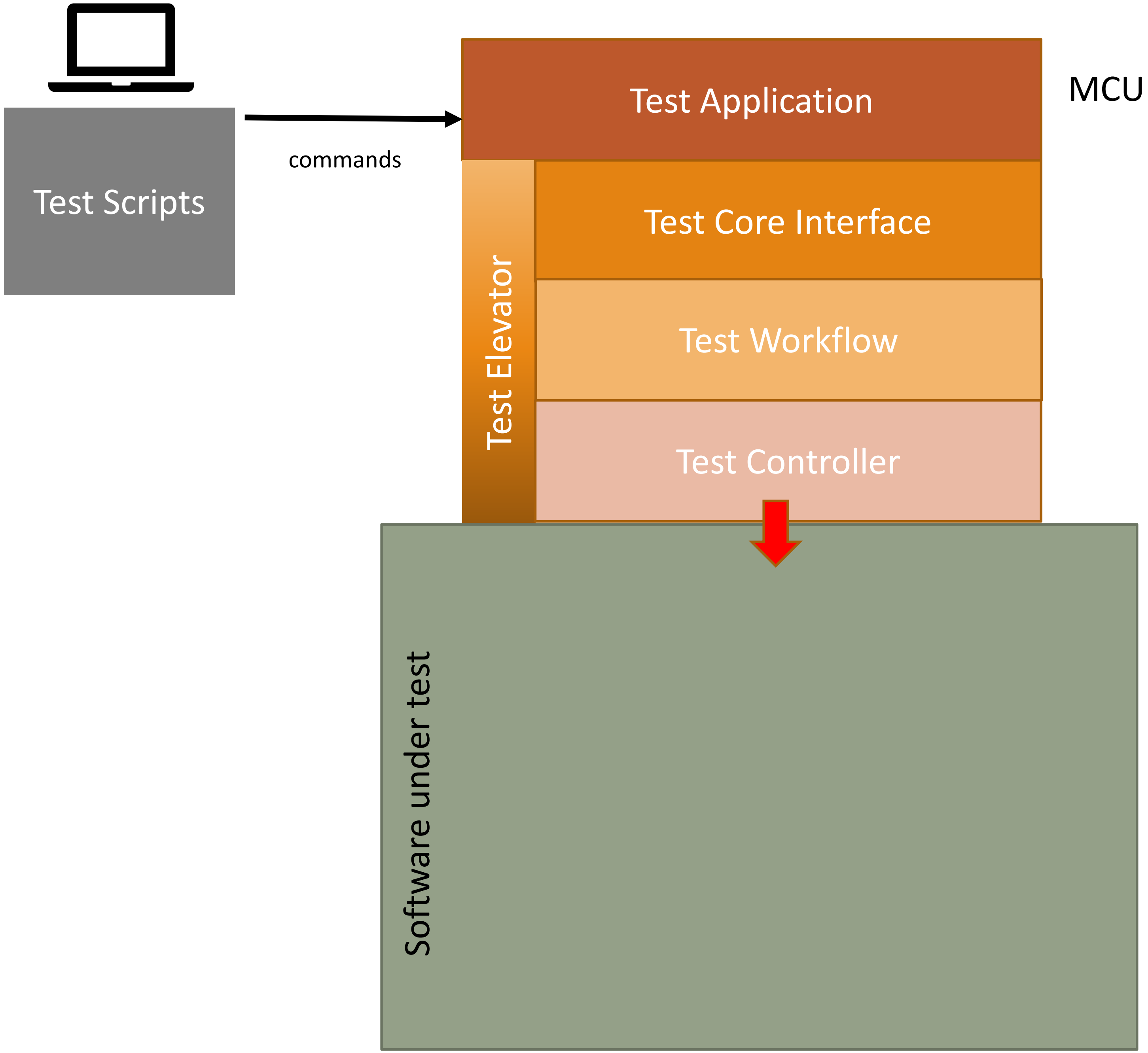
TESTING LOWER-LEVEL APPLICATION SOFTWARE ON TARGET

Target Tests for a Device

- We want to test the software directly on target
- We want to be able to automate the tests
  - i.e. some Test Runner on a dev. machine
- Can EMMA help us here?

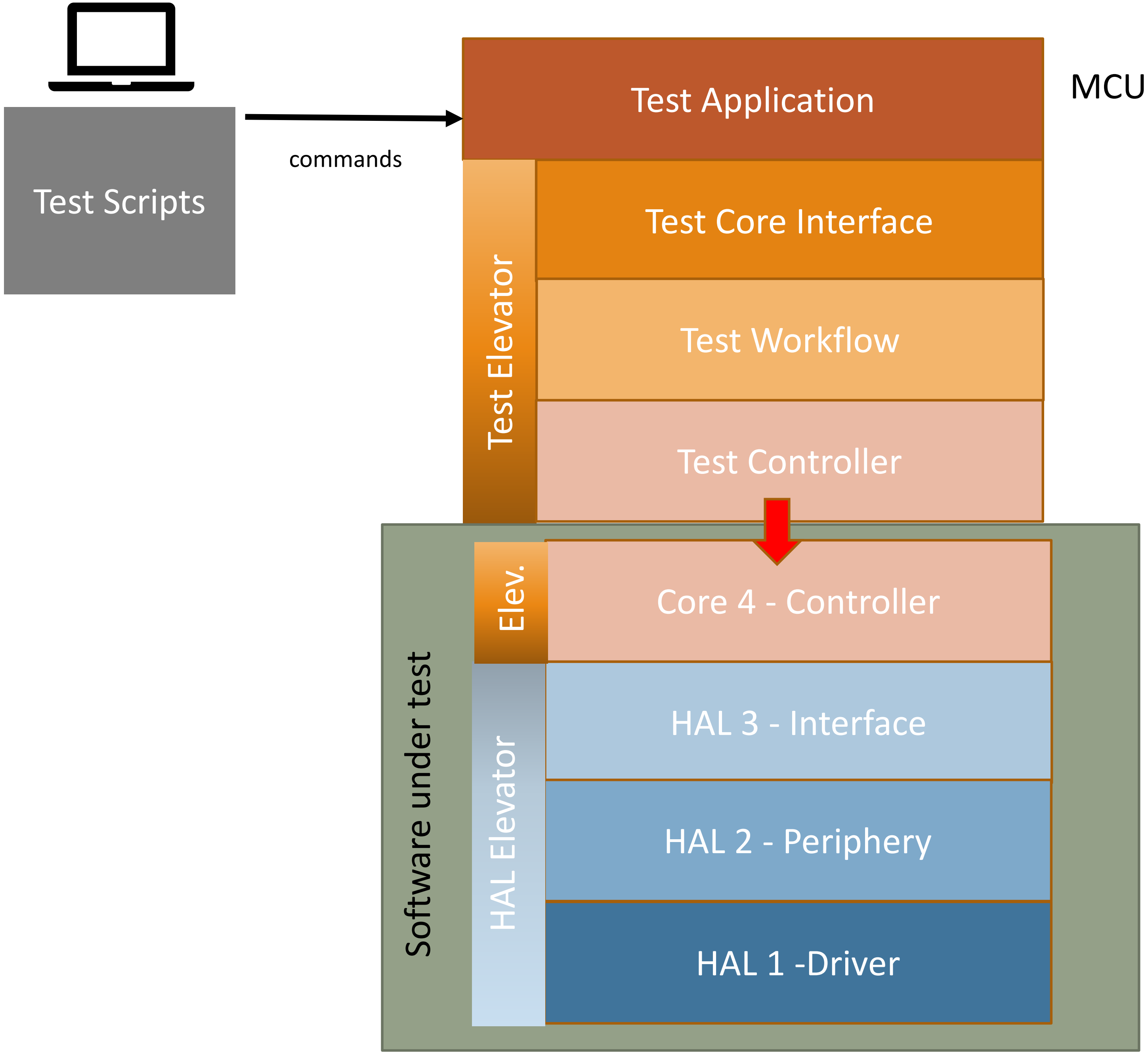
Test Architecture

Testing software directly on the MCU!

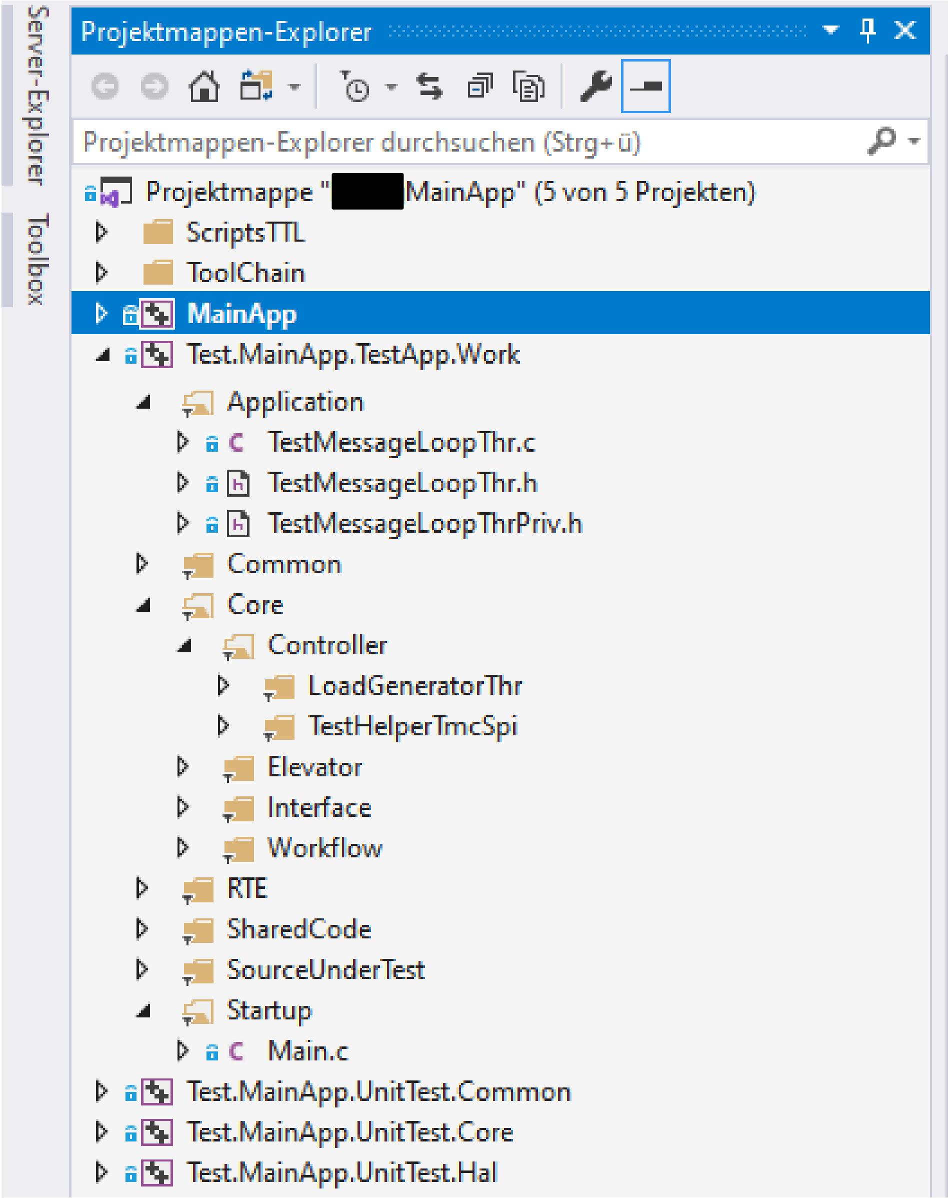


Test Architecture

Testing lower level layers directly on the MCU!



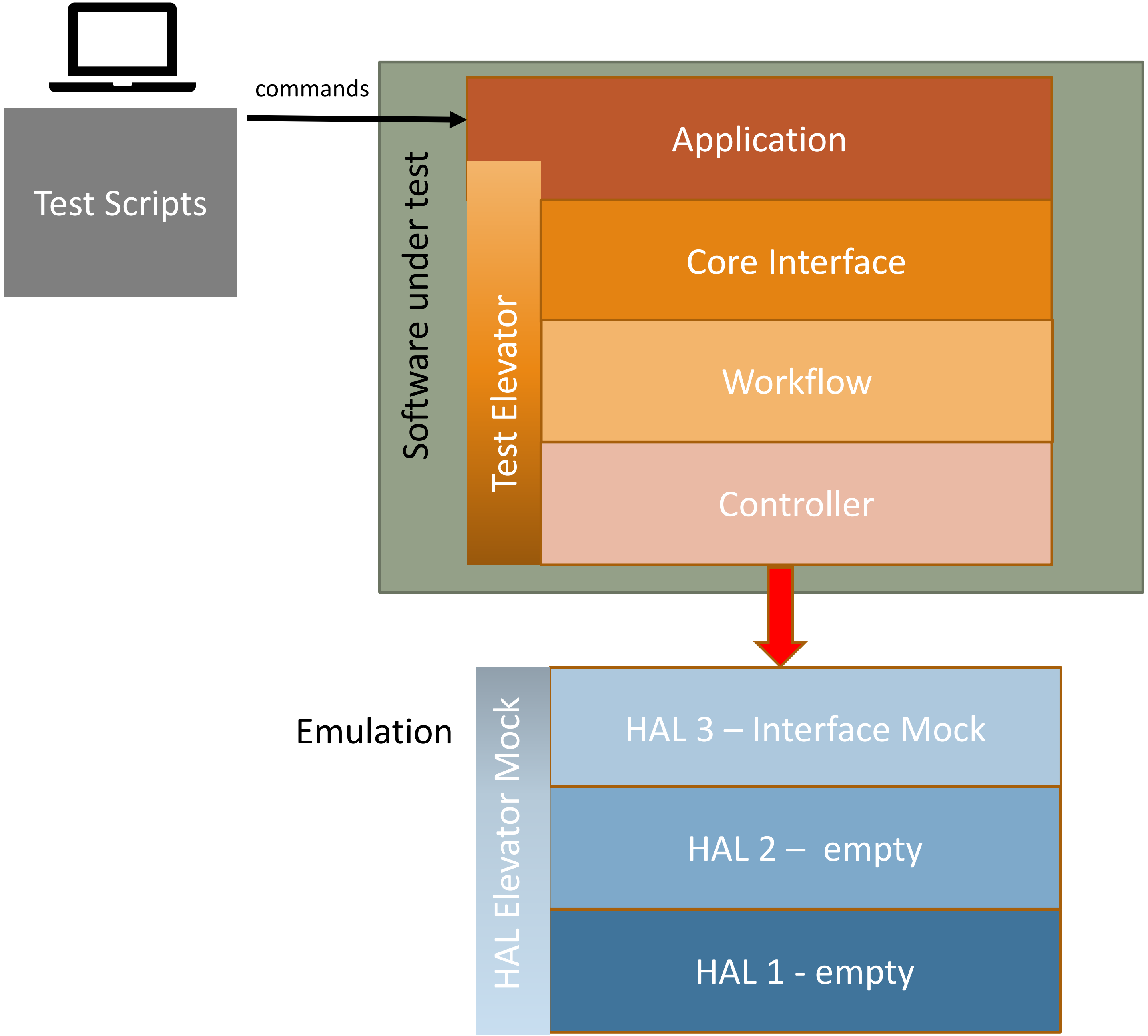
Test App. Project structure



- Projektmappe "[redacted]MainApp" (5 von 5 Projekten)
  - ScriptsTTL
  - ToolChain
  - MainApp
  - Test.MainApp.TestApp.Work
    - Application
      - TestMessageLoopThr.c
      - TestMessageLoopThr.h
      - TestMessageLoopThrPriv.h
    - Common
    - Core
      - Controller
        - LoadGeneratorThr
        - TestHelperTmcSpi
      - Elevator
      - Interface
      - Workflow
    - RTE
    - SharedCode
    - SourceUnderTest
    - Startup
      - Main.c
  - Test.MainApp.UnitTest.Common
  - Test.MainApp.UnitTest.Core
  - Test.MainApp.UnitTest.Hal

Test Architecture

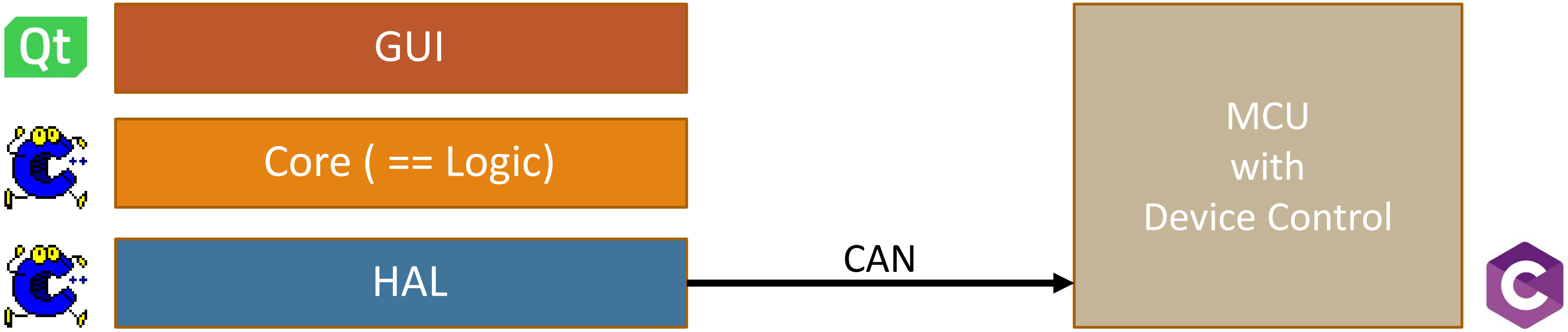
Testing upper level layers with simulated hardware!



Usage Case 3

QT BASED SOFTWARE ON EMBEDDED LINUX

3.1. Qt GUI on Yocto Linux and iMX-6



⚠ HAL and Core Elevators both use the Qt event system!

3.2. GUI and Central App. on Yocto Linux and iMX-6

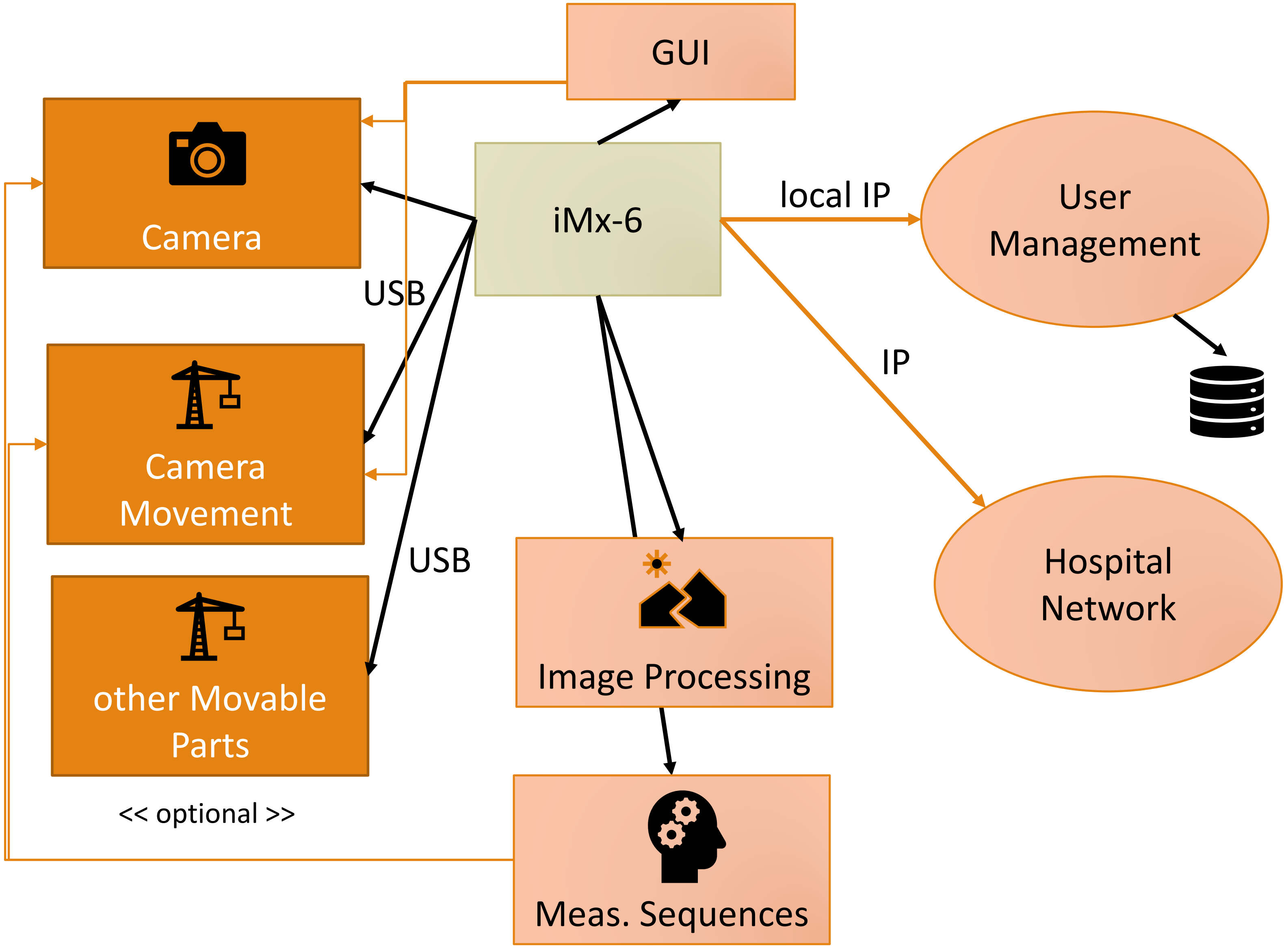
Not only a Device Controller this time!

Also:

- Measurement Logic,
- rich GUI, Qt/QML used
- User Management,
- IP Networking

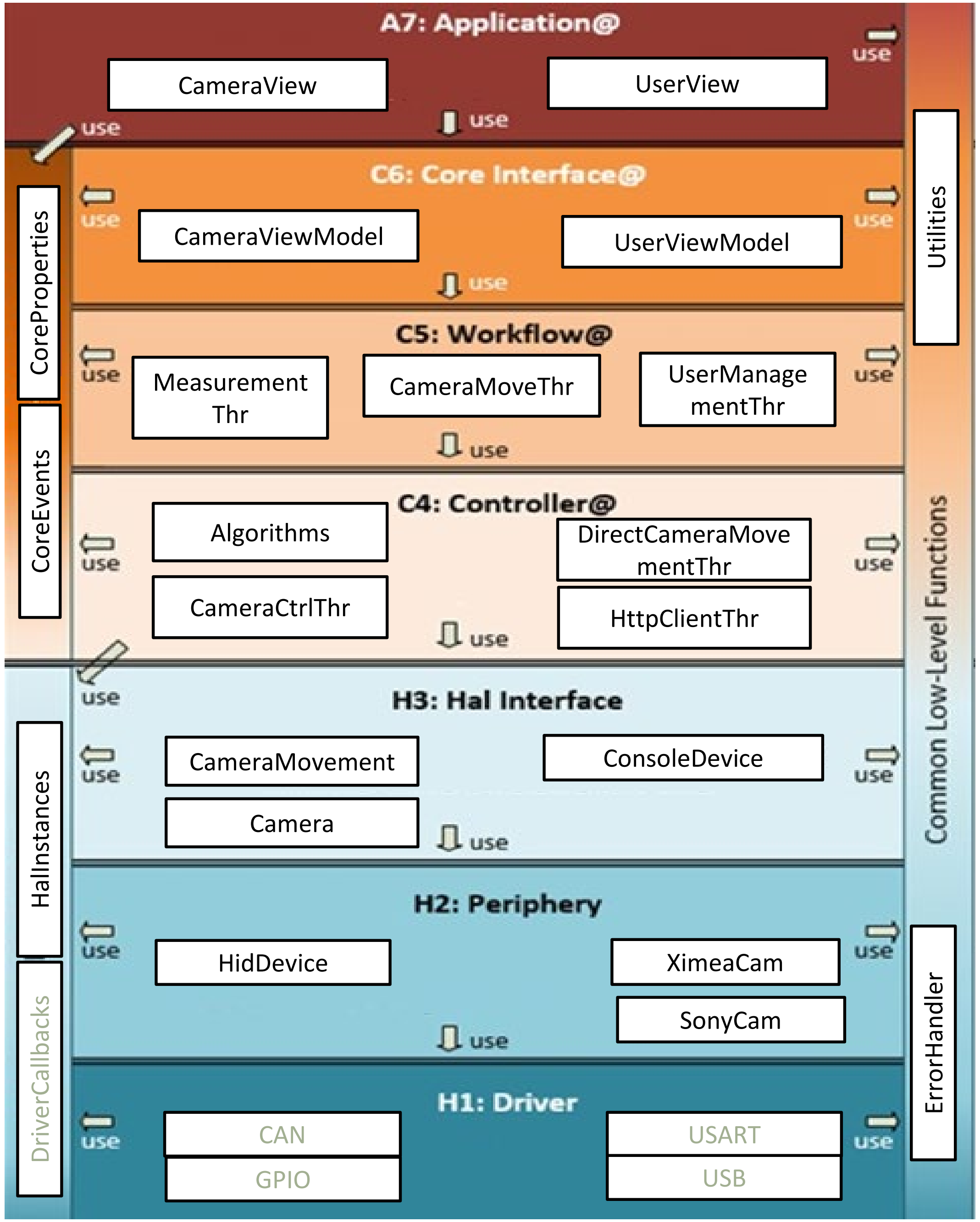
The System

- Networking
- Direct device control
- Rich GUI



Architecture Proposal

- No HAL-1 layer modules – Linux driver libraries (!)
- Application layer == GUI
- Network comminication
- Image processing algorithms
- Measurement sequences



Discussion of the Proposal

New/Open issues:

- New elevator type needed: Property Elevator
- UI in the Application Layer or partitioned between A7 and C6?
- User Management in C5 – Workflow (?)
- IP and HTTP Networking – in C4 – Controller (?)
- Core Elevators – use Qt events mechanism?
  - GUI uses Qt, thus we must translate between Qt/non-Qt events!
- HAL Elevator – only reports installed optional devices
- Startup from Qt or Qt started in an extra GUI thread?

Thank you!

Any questions?