Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

INTERNET DES OBJETS

RTOS PROGRAMMING FOR THE INTERNET OF THINGS

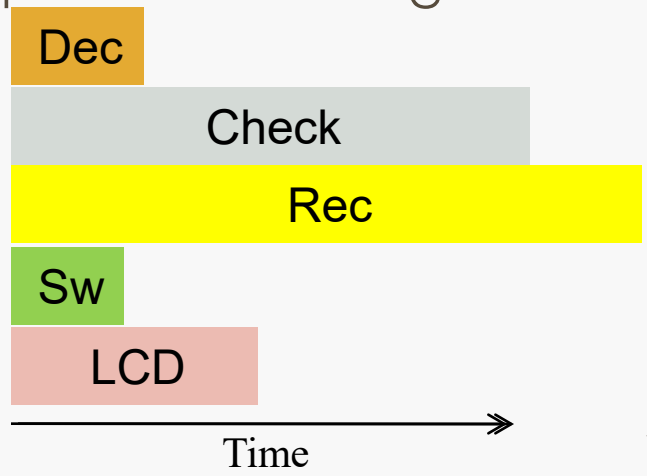
SERGE AYER - HEIA-FR – TÉLÉCOMMUNICATIONS
CLASSES ISC-2D // 2023-2024

WHY A RTOS ?

- RTOS stands for Real-time Operating Systems
  - Designed to meet timing constraints
    - Hard real-time – critical tasks have to be completed on time
    - Soft real-time – may continue finishing the task even missing the deadline
- The key design requirement is predictability and determinism
- So is the solution a huge sequential loop ?
  - Poor predictability and extensibility
- With RTOS, encapsulate each computation request into a task and schedule tasks on demand

A CASE IN POINT

- GPS based speed limit alarm with moving map
  - Sounds alarm when approaching a speed limit change
  - Display vehicle's position on LCD
  - Let user confirm automatic speed change
  - Log applied speed limit and car's position
- Tasks:
  - Decode GPS information to find current vehicle position
  - Check to see if approaching any speed limit change location
  - Record position to flash memory
  - Read user input
  - Update LCD



A CASE IN POINT

- How to implement this behaviour in a super loop?
  - Do tasks run in the same order every time?
  - Allow pre-emption?
- Super-loop implementation is simple but…

  - Always run the same schedule, regardless of changing conditions and relative importance of tasks.
  - All tasks run at the same rate. Changing rates requires adding extra calls to the function (e.g. requiring user input multiple times)
  - Maximum delay is the sum of all task run times. Polling/execution rate is equal to 1/maximum delay.

A CASE IN POINT

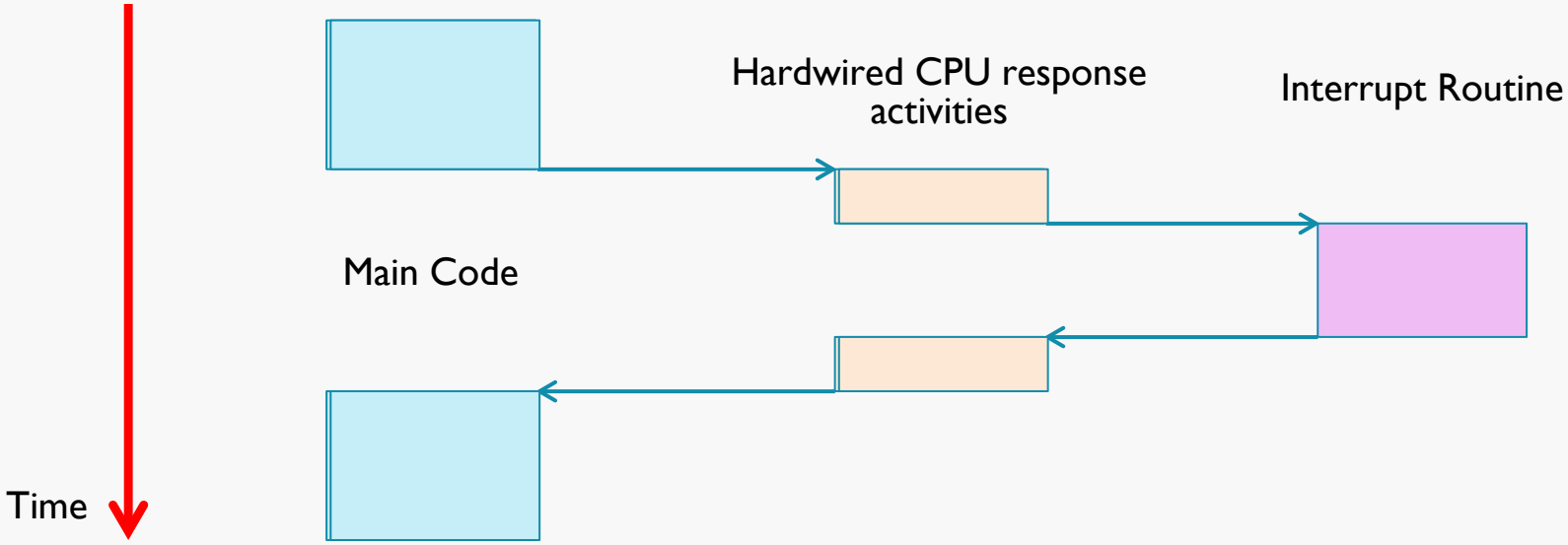
- This approach is simple but it has many drawbacks
  - What if we receive GPS position right after another task starts running?
  - Have to wait for Rec, Sw, LCD before we start decoding position with Dec.
  - Have to wait for Rec, Sw, LCD, Dec, Check before we know if we are approaching a speed limit change!
- One needs to be able to define tasks and to schedule them based on priority and preemption
  - Need for a RTOS !

ANOTHER CASE IN POINT

- What about interrupt handling?
- Consider reacting to user input (e.g. pressing a button). How to handle it?
  - Polling – use software to regularly check it
    - Slow
    - Wasteful of CPU time
    - Scales badly
  - Interrupt – use special hardware in MCU to detect even and run the Interrupt Service Routine (ISR) in response
    - Efficient
    - Fast
    - Scales well

INTERRUPT PROCESSING SEQUENCE

- Main code is running
- Interrupt trigger occurs
  - Processor does some hard-wired processing
  - Processor executes the ISR, including return-from-interrupt instruction at the end
- Processor resumes to main code



NEEDS FOR DEVELOPING IOT EMBEDDED APPLICATIONS

- Define tasks and to schedule them as requested
- Handle interrupts in a proper way
- Care easily with power consumption.
- Encapsulation of main functionalities into properly designed APIs

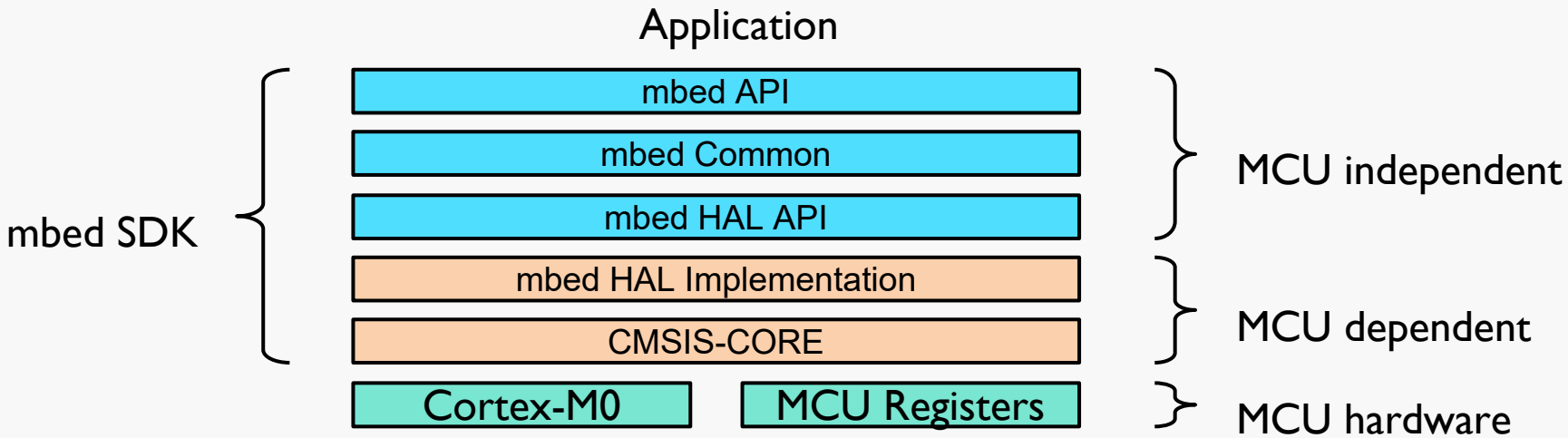
INTRODUCTION TO MBED-OS

- What is Mbed OS?
  - Easy prototyping
  - For ARM Cortex-M-based microcontrollers
- The Mbed OS platform provides
  - Open software libraries
  - Open hardware designs
  - Tools for professional rapid prototyping of products
- The Mbed OS platform includes
  - C/C++ Software Development Kit (SDK)
  - A microcontroller Hardware Development Kit (HDK) and supported development boards
  - Off-line and online IDEs

USING AN OS VS LOW-LEVEL PROGRAMMING

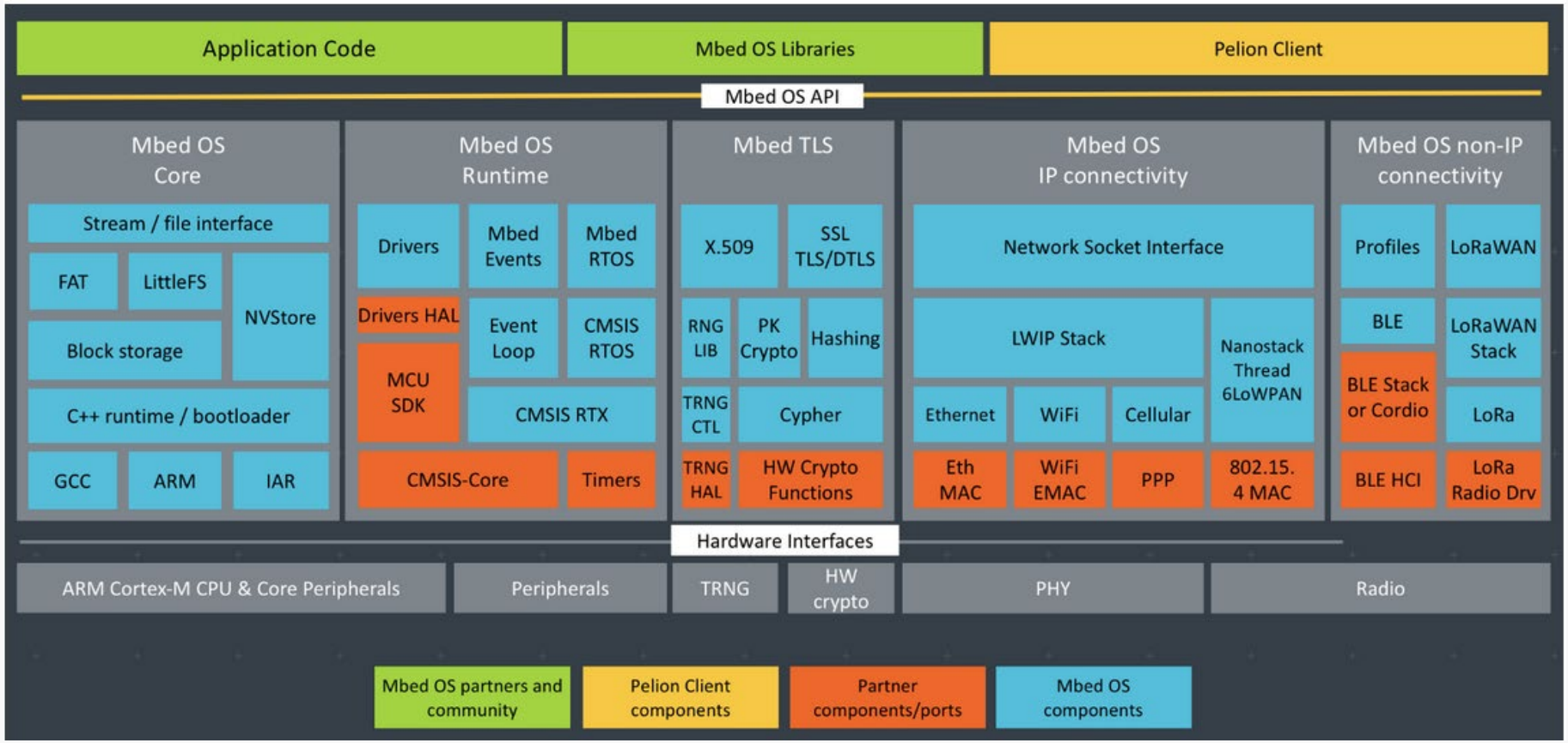
- Low-level Programming: great flexibility but
  - Low productivity
  - Less portable from one device to another device
  - Code is more difficult to read, reuse, and maintain
- With high-level APIs, we may achieve:
  - Higher productivity (less development time)
  - Portability across devices
  - Code easier to read, reuse and maintain by others
  - Sometimes, even more efficient code (code density and performance)

MBED OS ARCHITECTURE



Connectivity and security: Mbed OS includes many APIs like TLS, Crypto, BLE, Cellular, IP, etc.

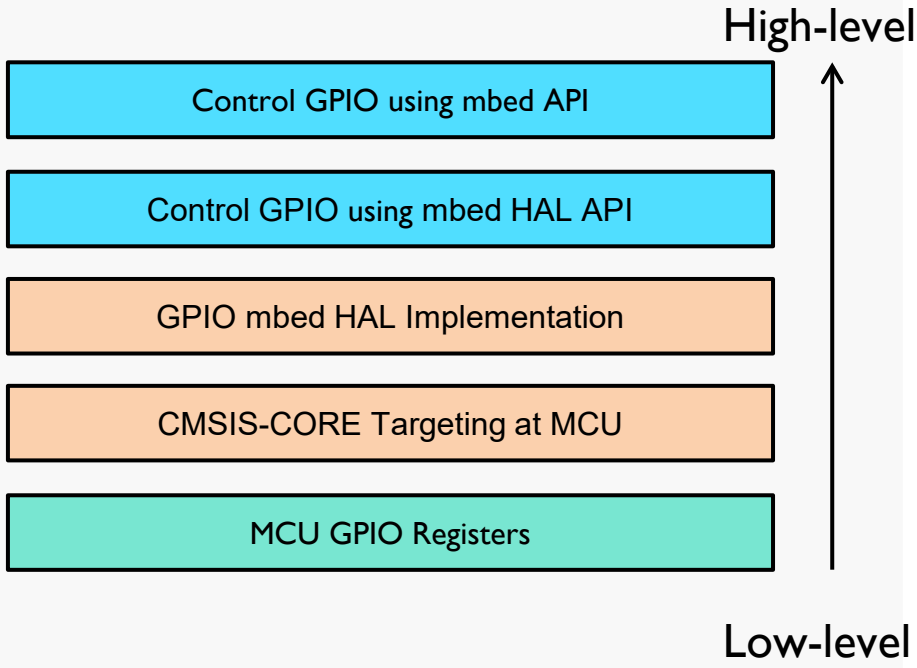
MBED OS ARCHITECTURE



WHY C++ RATHER THAN C

- Demonstrate how blinky (using GPIO) can be programmed at different levels, using libraries from the lowest layer to the highest layer

- MCU register layer
  - Blinky example by poking registers
- CMSIS-CORE
  - Blinky example using CMSIS-CORE structures
- mbed HAL API
  - Blinky example using mbed HAL API functions
- mbed API
  - Blinky example using mbed API functions



TASKS/THREAD IN MBED-OS

- For handling and scheduling parallel tasks, Mbed OS provides a Thread API
  - Creation with or without dynamic memory (heap memory)
  - Internal thread data structures hidden in the C++ class.
  - By default, the thread stack is allocated on the heap.
  - If you don't want to use dynamic memory for the stack, you can provide your own static memory using the constructor parameters.