COL788: Advanced Topics in Embedded Computing

Lecture 15 – Embedded OS

Vireshwar Kumar
CSE@IITD

September 14, 2022

Semester I
2022-2023

Outline

- Previously
  - System Architecture
  - Processor Architecture
  - Memory

- Next
  - Boot Sequence
  - **Operating System**
  - Embedded Linux

Contiki OS

- Features
  - Memory Allocation
  - Full IP Networking
  - Power Awareness
  - 6Lowpan, RPL, CoAP
  - Dynamic Module Loading
  - Protothreads
  - File System
  - Contiki Shell

CooCox CoOS

- Features
  - Free and open real-time Operating System
  - Specially designed for Cortex-M series
  - Scalable, minimum system kernel is only 974Bytes
  - Adaptive Task Scheduling Algorithm.
  - Supports preemptive priority and round-robin
  - Interrupt latency is next to 0
  - Stack overflow detection option
  - Semaphore, Mutex, Flag, Mailbox and Queue for communication & synchronisation
  - Supports the platforms of ICCARM, ARMCC, GCC

TinyOS

- Overview
  - TinyOS is an "operating system" designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms.

FreeRTOS

- What is FreeRTOS?
  - FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller
- What is an RTOS?
  - A part of the operating system called the scheduler is responsible for deciding which program to run when and provides the illusion of simultaneous execution by rapidly switching between each program.
  - The scheduler in a Real Time Operating System (RTOS) is designed to provide a predictable (normally described as *deterministic*) execution pattern.

Embedded Operating System

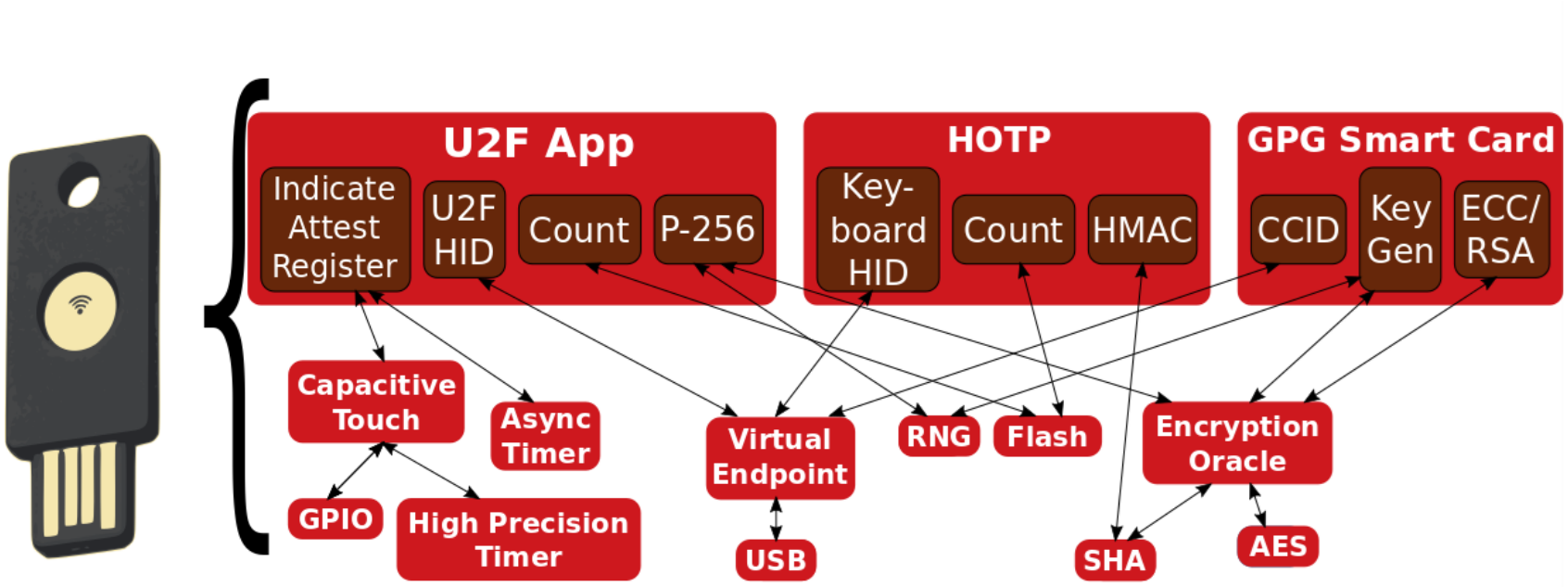
- Some subset of operating system principles, dictated by application
  - Hardware abstraction
    - Block storage versus file system?
    - But not architecture? [What’s a runtime?]
  - Resource management
    - Process isolation
    - Job control

Example: USB Authentication Key

Multiple independent applications

No programmability in favor of security

Result: Handful of programmers control software stack

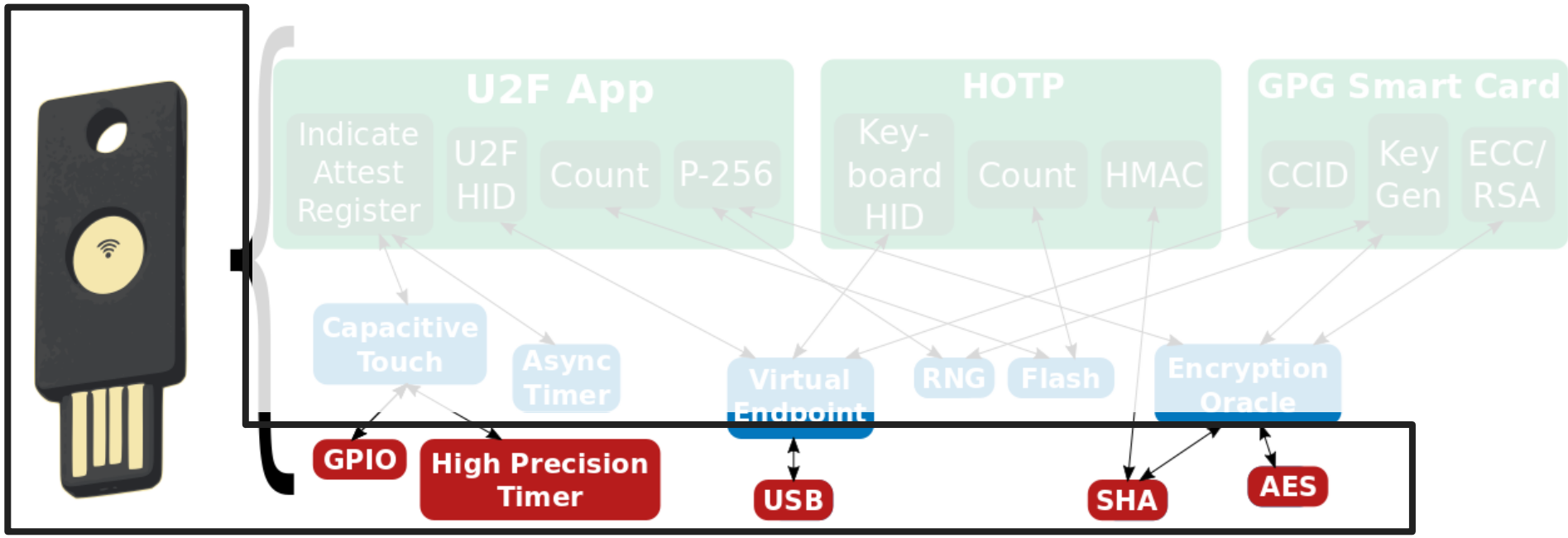


Platform

Build the hardware

Responsible for TCB: core kernel, MCU-specific code

Trusted: complete control over firmware & hardware



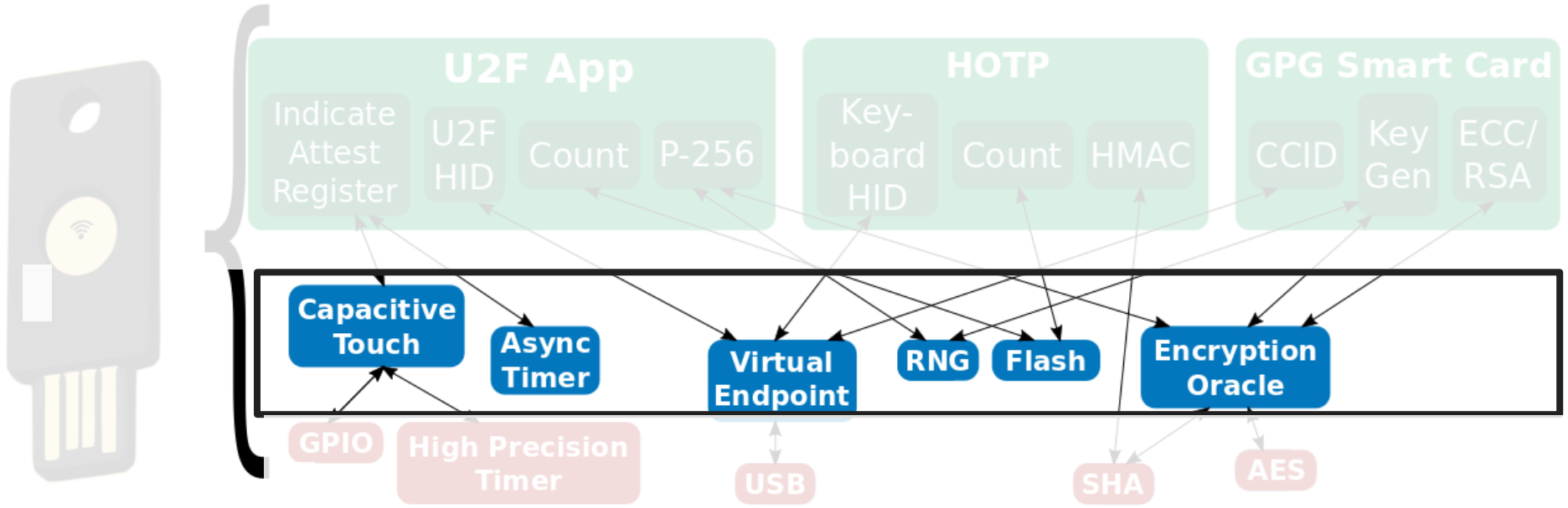
Goal: possible to correctly extend TCB

OS Services

Most OS services come from community

Device drivers, networking protocols, timers...

Platform provider can audit but won't catch all bugs



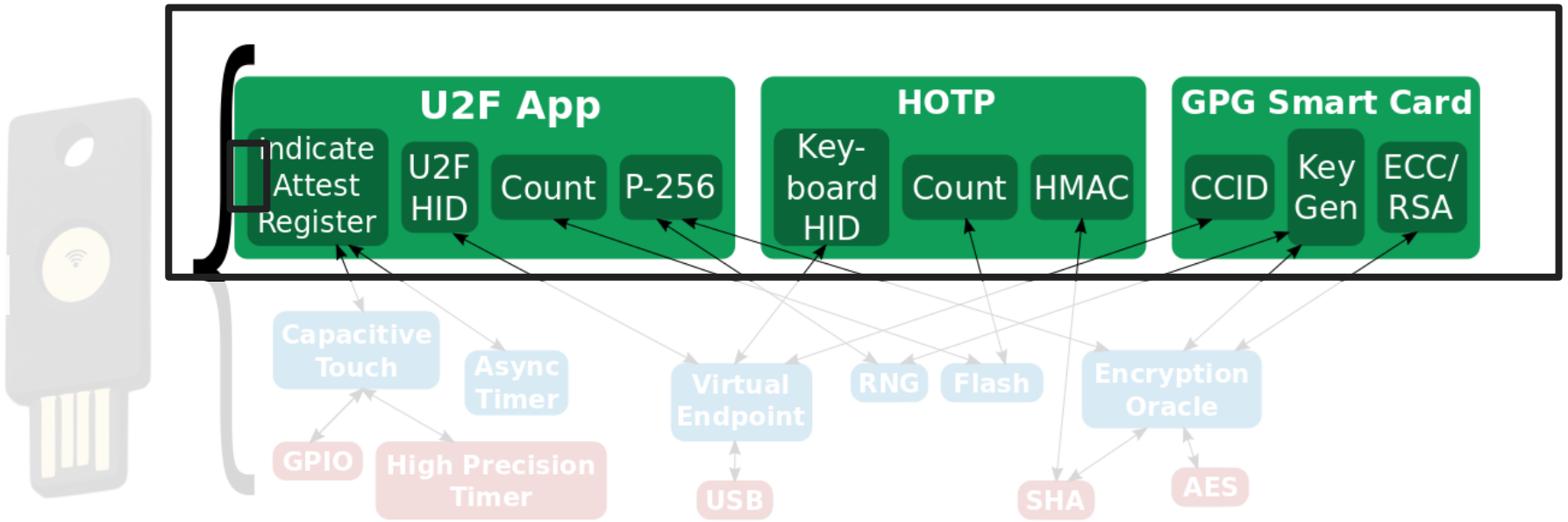
Goal: protect kernel from safety violations

Applications

Implement end-user functionality

“Third-party” developers: unknown to platform provider

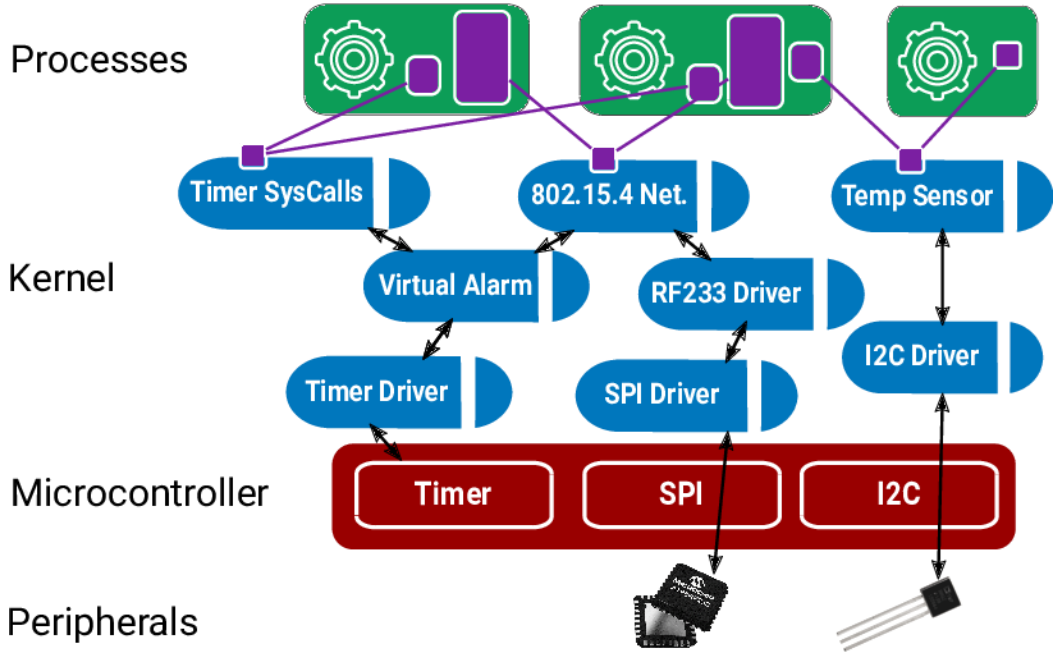
Potentially *malicious*



Goal: end-users can install 3rd-party apps

A Microcontroller OS

Processes: Use the Memory Protection Unit
Capsules: Type-safe Rust API for *safe* driver development
Grants: Bind dynamic kernel resources to process lifetime



What’s Next?

- Lecture 16
  - September 15, Thursday, 12 pm – 1 pm