Multiprogramming a 64 kB Computer Safely and Efficiently

Amit Levy[†] Brad Campbell[*] Branden Ghena[‡] Daniel B. Giffin[†]
Pat Pannuto[‡] Prabal Dutta[‡] Philip Levis[†]

Stanford University[†] University of Virginia[*] U.C. Berkeley[‡]

SITP '17, December 4th 2017

Microcontrollers Becoming Platforms

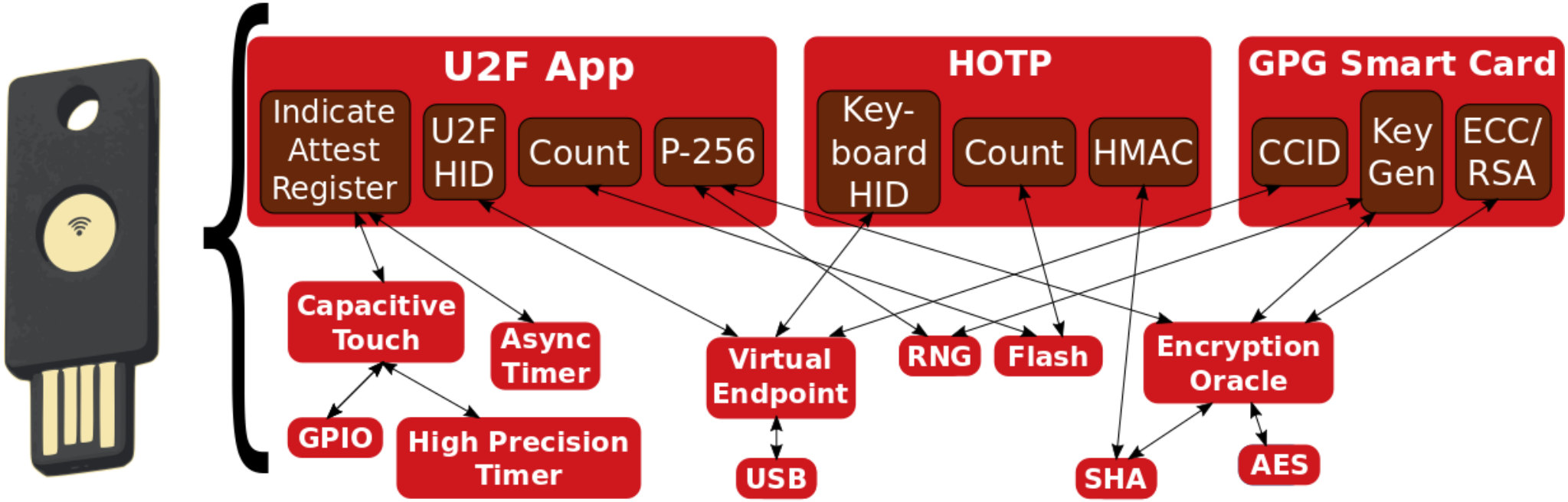
- Low energy: µA sleep current
- Low memory: ~64 kB RAM

- USB authentication keys have multiple functions
  - U2F, SSH, GPG, HOTP
- Sensor networks run several experiments at once
- Fitness watches support different activities
- This should scare you!


Windows
A fatal exception 0E has occurred at 0028:C0031E36 in VXD VMM(01) +
00005E36. The current application will be terminated.
• Press any key to terminate the current application.
• Press CTRL+ALT+DEL again to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue _

Example: USB Authentication Key

- Multiple independent applications
- No programmability in favor of security



Microcontrollers need fault isolation for applications and OS services.

Fault Isolation on Microcontrollers

But new isolation tools available

- **MPU: Protection bits for 8 memory regions**
- **Rust: a non-GC'd type safe systems language**

What are the operating systems abstractions?

Design Goals

- **Isolate drivers at no resource cost**
  - Rust type-system to prevent safety violations
- **Isolate applications**
  - Process isolation using MPU
  - Must immediately reclaim upon termination
- **Support concurrent applications & I/O**
  - Need memory abstractions in lieu of heap

Tock: A Microcontroller OS

- Kernel components in Rust
- Type-safe API for *safe* drivers development
- Processes abstraction with Memory Protection Unit
- *Grants*: concurrency without a global heap
  - Manages process-specific kernel heap
  - Ensures resources can be freed immediately

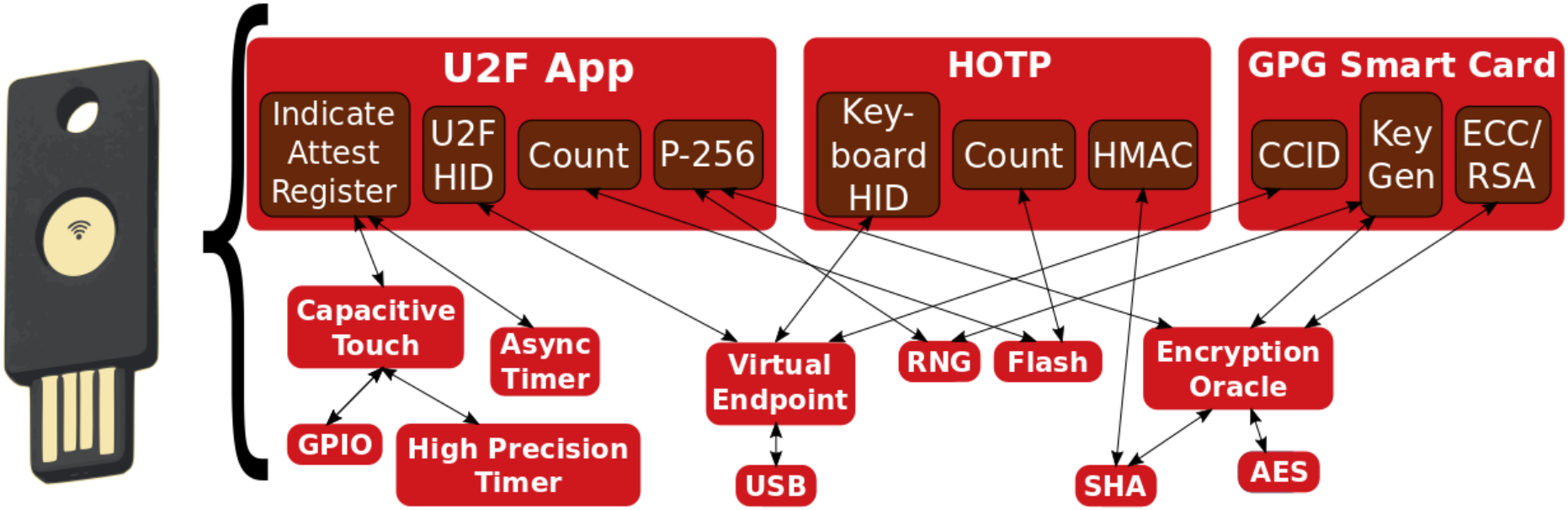
Outline

1. Why Rust?
2. Security Model
3. Two Isolation Mechanisms
4. Grants
5. Case Study: Signpost
6. Limitations & Future Work

Why Rust?

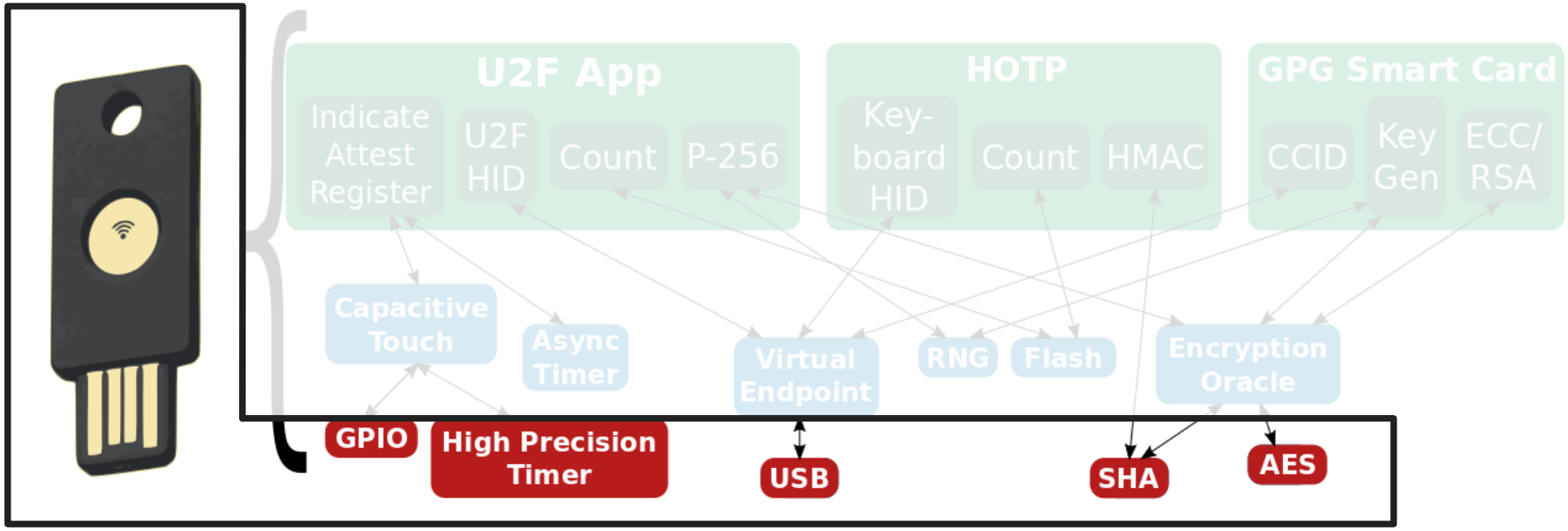
- **Type and memory safe**
  - No buffer overflows, dangling pointers, type confusion…
- **Compile-time enforced type system**
  - No type artifacts at run time
- **No garbage collection**
  - Control over memory layout and execution
- **Runtime behavior similar to C**

Security in a Multiprogrammable MCU



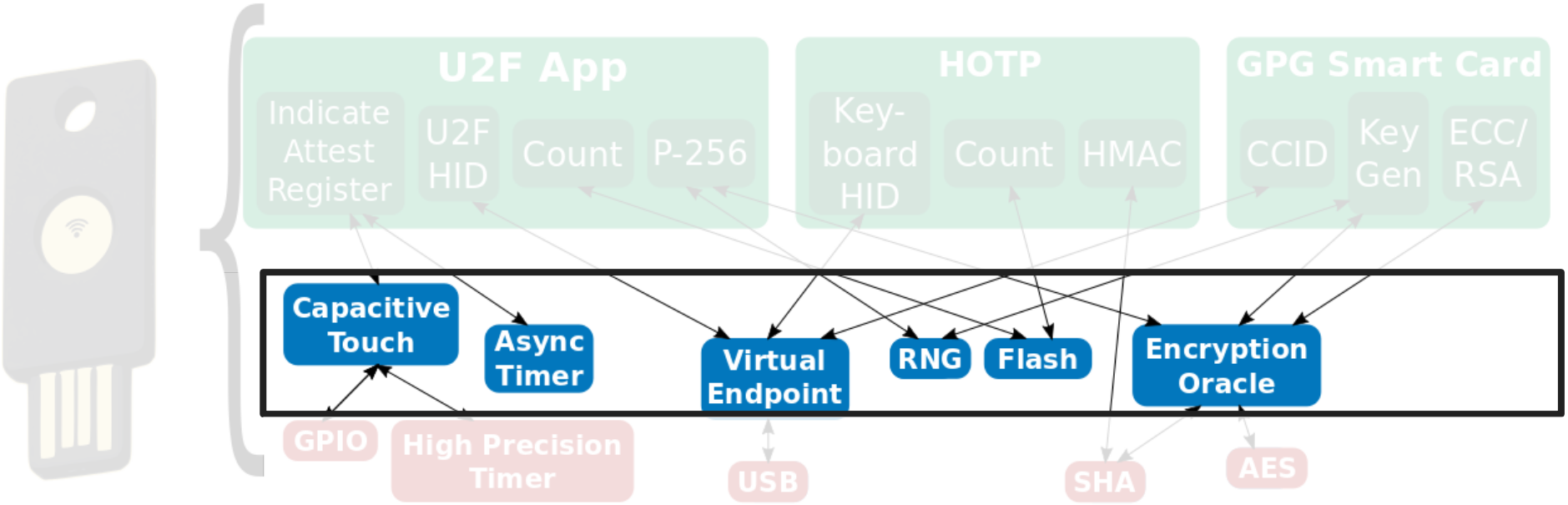
Platform Provider

- Build the hardware
- Responsible for TCB: core kernel, MCU-specific code
- Trusted: complete control over firmware & hardware



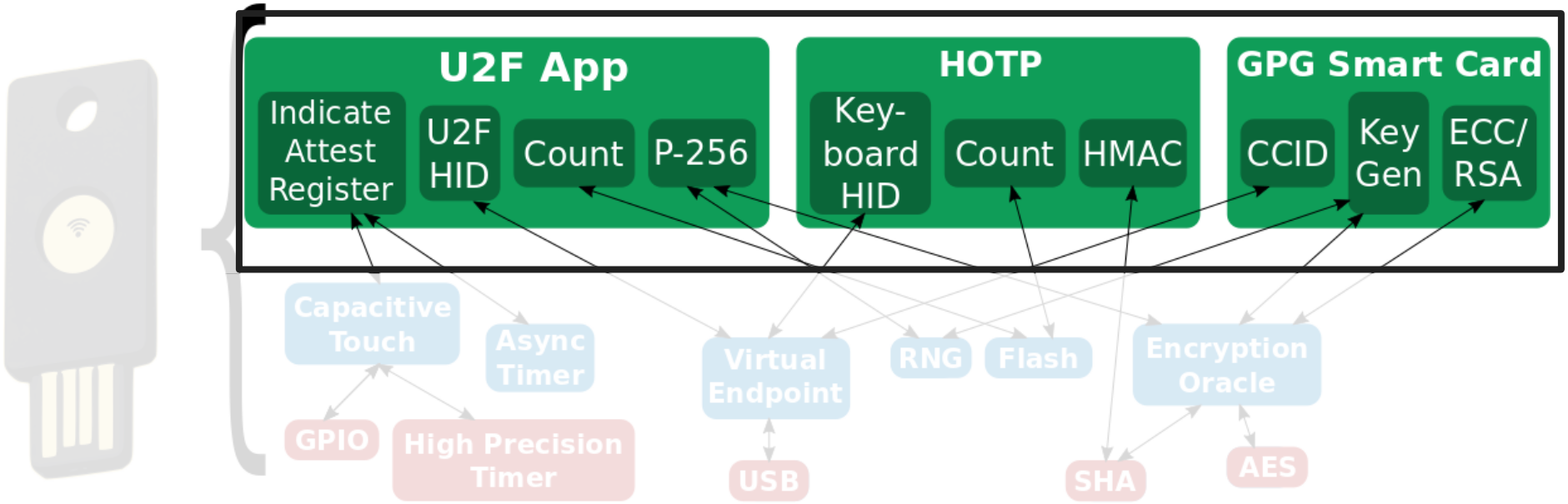
Kernel Component Developers

- Build most kernel functionality
  - Device drivers, networking protocols, timers...
- Platform provider select: audit but won't catch all bugs



Application Developers

- Implement end-user functionality
- “Third-party” developers: unknown to board integrators
- Modeled as *malicious*



Tock's Isolation Mechanisms

Trusted for liveness, not safety

Capsules

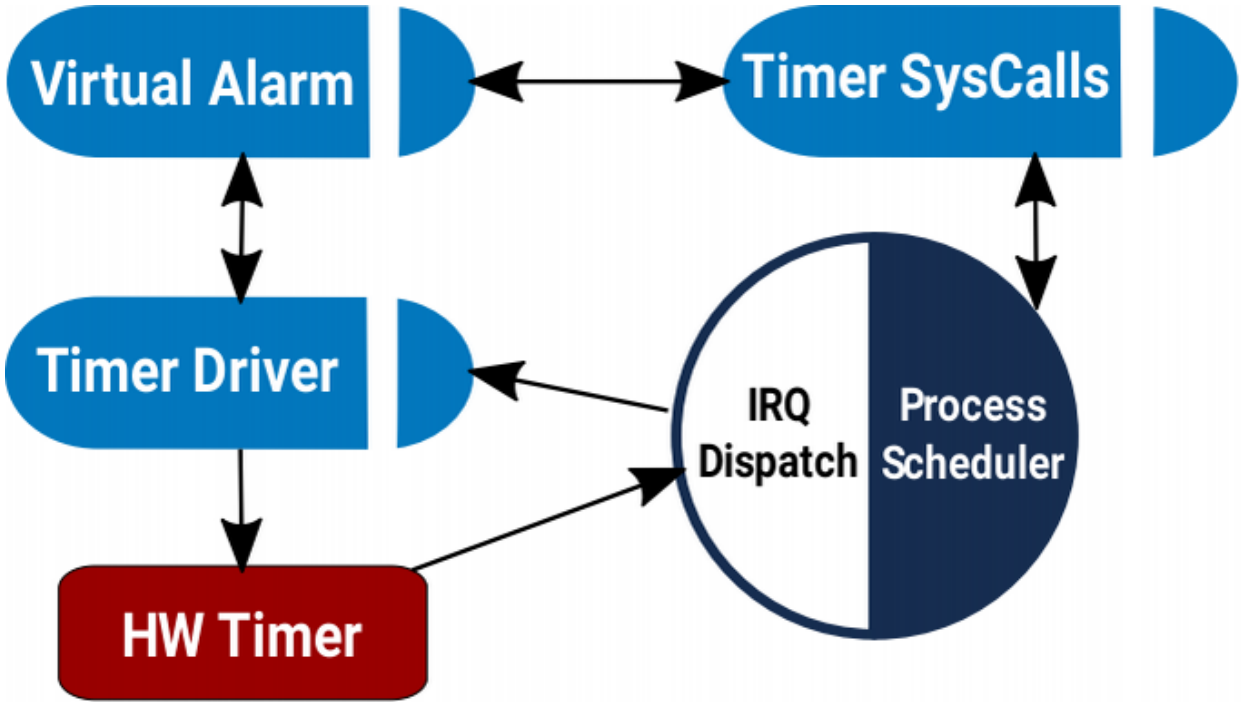
- Rust code linked into kernel
- Isolation enforced at compile-time
- Low overhead
- Used for device drivers, protocols, timers...

Totally untrusted

Processes

- Standalone executable in any language
- Isolation enforced at runtime
- Higher overhead
- Applications

Capsules



- A Rust module and structs
- Event-driven execution with asynchronous I/O
- Shared stack, no heap
- Communicate via references & method calls
  - Often inlined

Kernel Memory Consumption

Example 1: “blink”

	ROM size (B)	RAM size (B)
Tock	3208	916
TinyOS	5296	72
(AWS) FreeRTOS	4848	2984

Example 2: Networked sensor

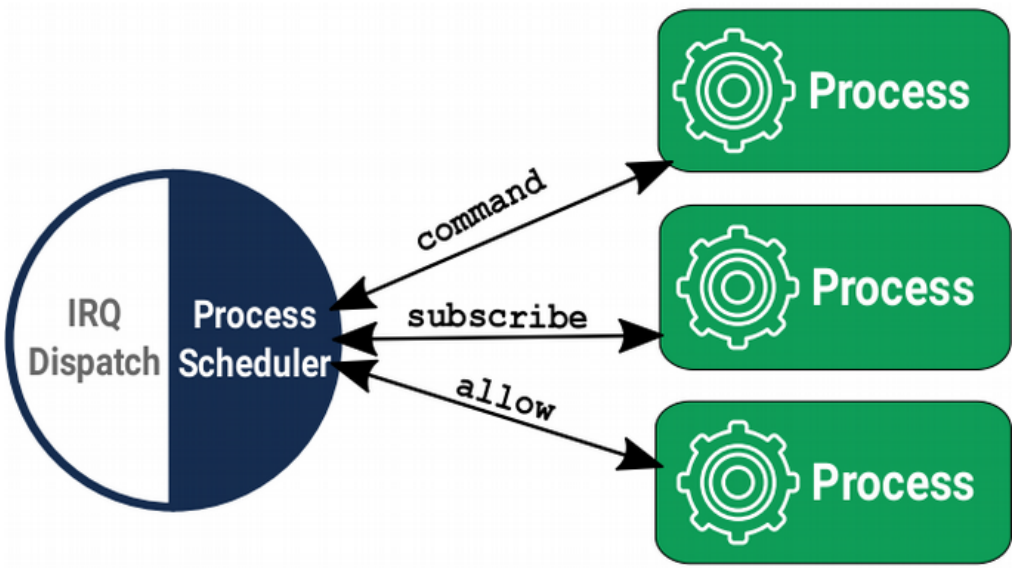
	ROM size (B)	RAM size (B)
Tock	41744	9704
TinyOS	39604	10460

Capsule Isolation

```
struct DMAChannel {
    length: u32,
    base_ptr: *const u8,
}

impl DMAChannel {
  fn set_dma_buffer(&self, buf: &'static [u8]) {
    self.length = buf.len();
    self.base_ptr = buf.as_ref();
  }
}
```

- Exposes the DMA base pointer and length as a Rust *slice*
- Type-safety guarantees user has access to memory

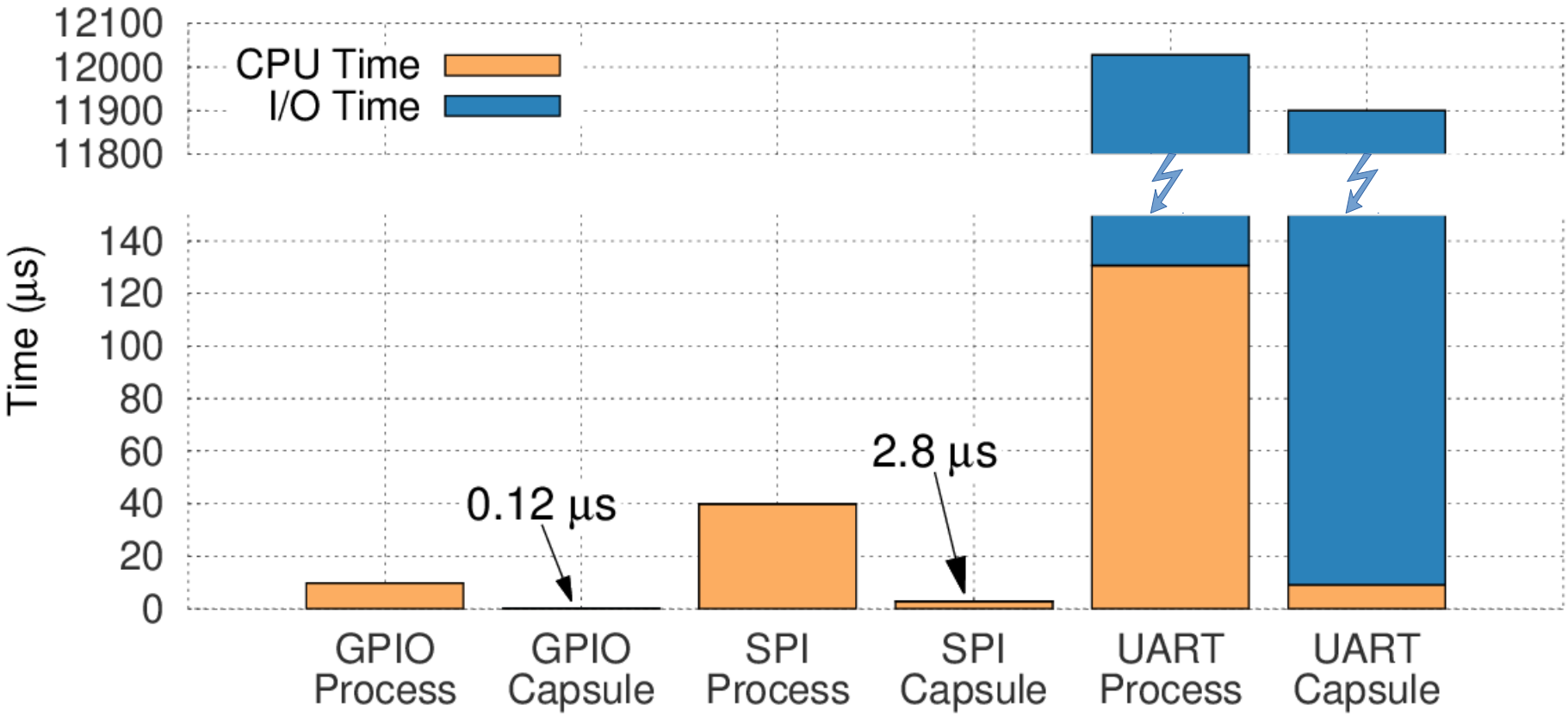
Processes



- Hardware-isolated concurrent executions of programs
  - MPU to protect memory regions without virtualization
  - Independent stack, heap, static variables
- Updated dynamically with position independent code
- Scheduled preemptively
- System calls & IPC for communication

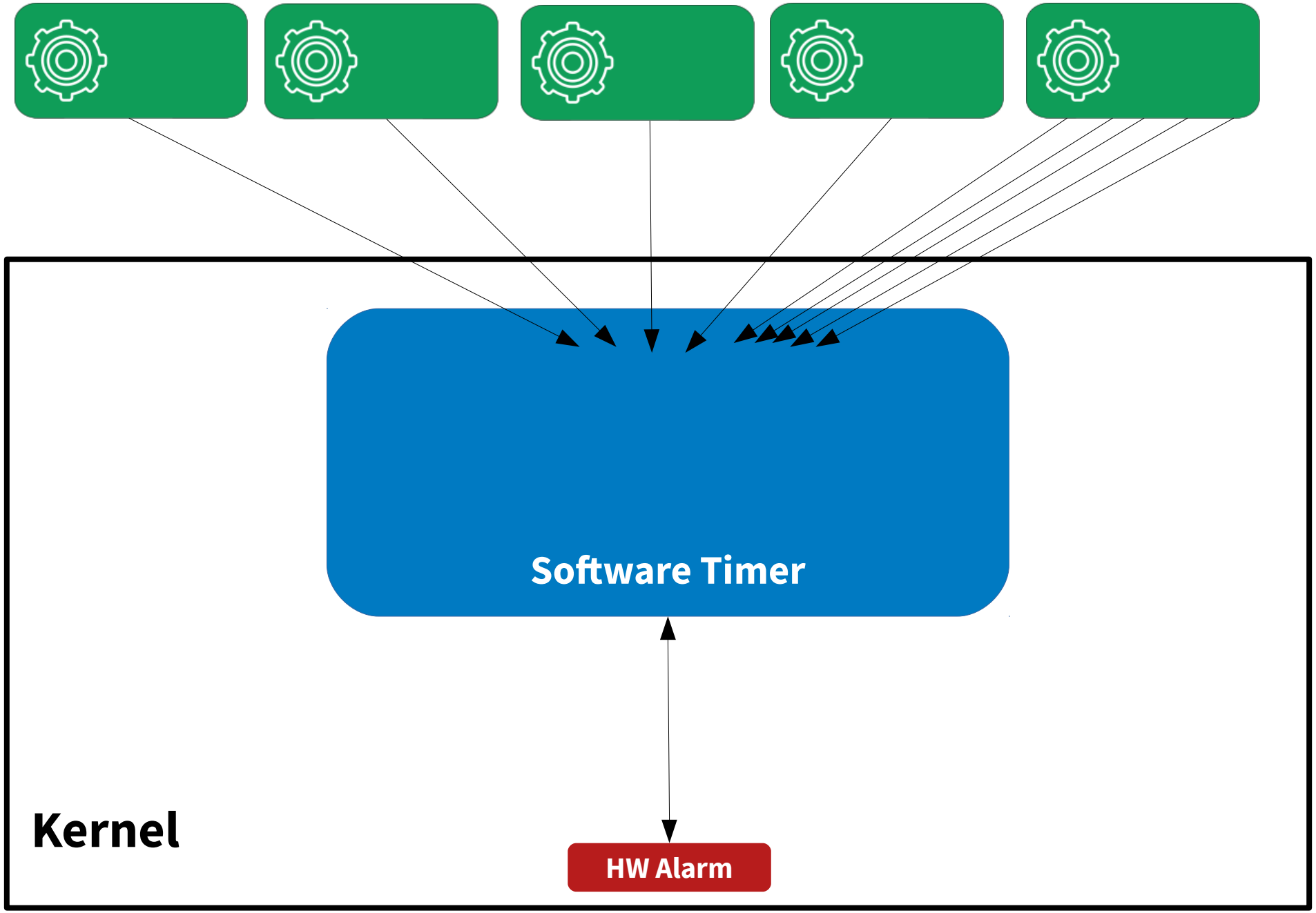
Process Overhead

- Dedicated memory region (at least a stack)
- Context switch for communication (340 cycles)

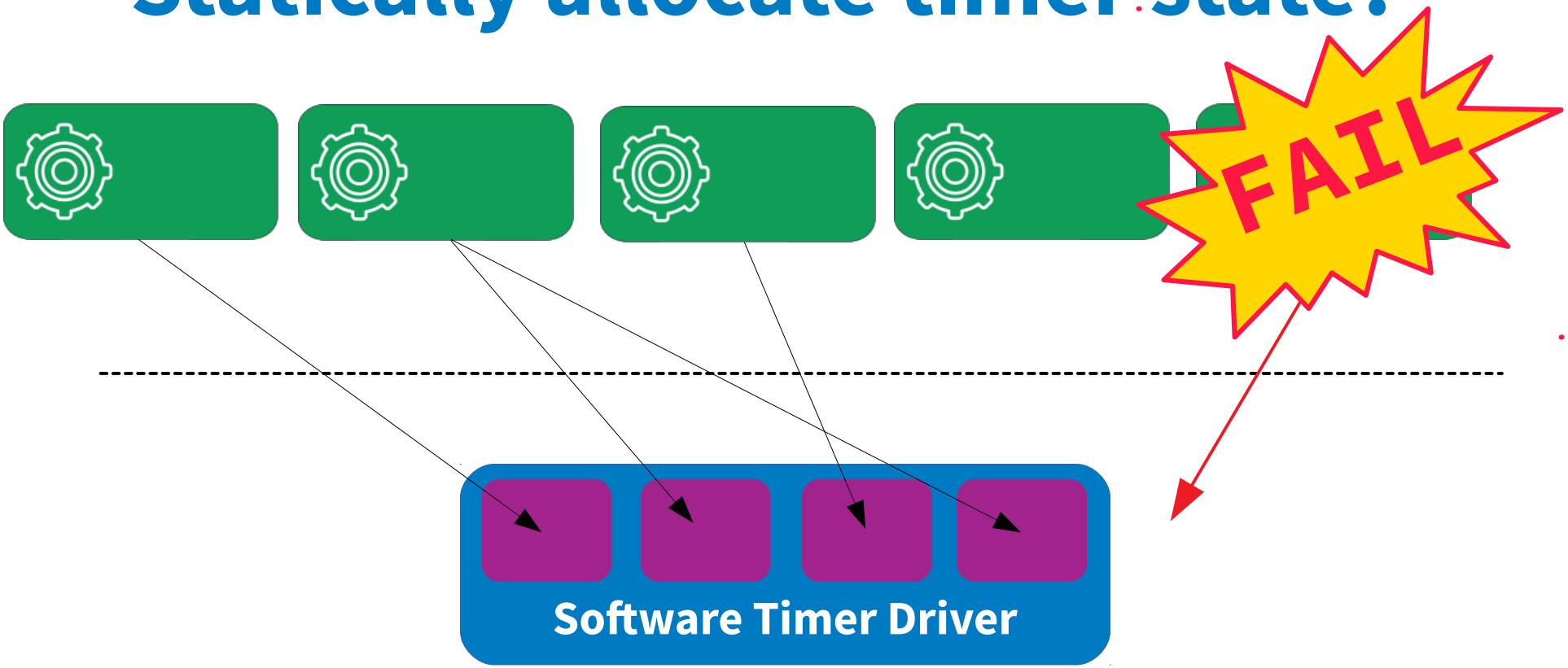


How do capsules and processes interact?

Working Example: Software Timer

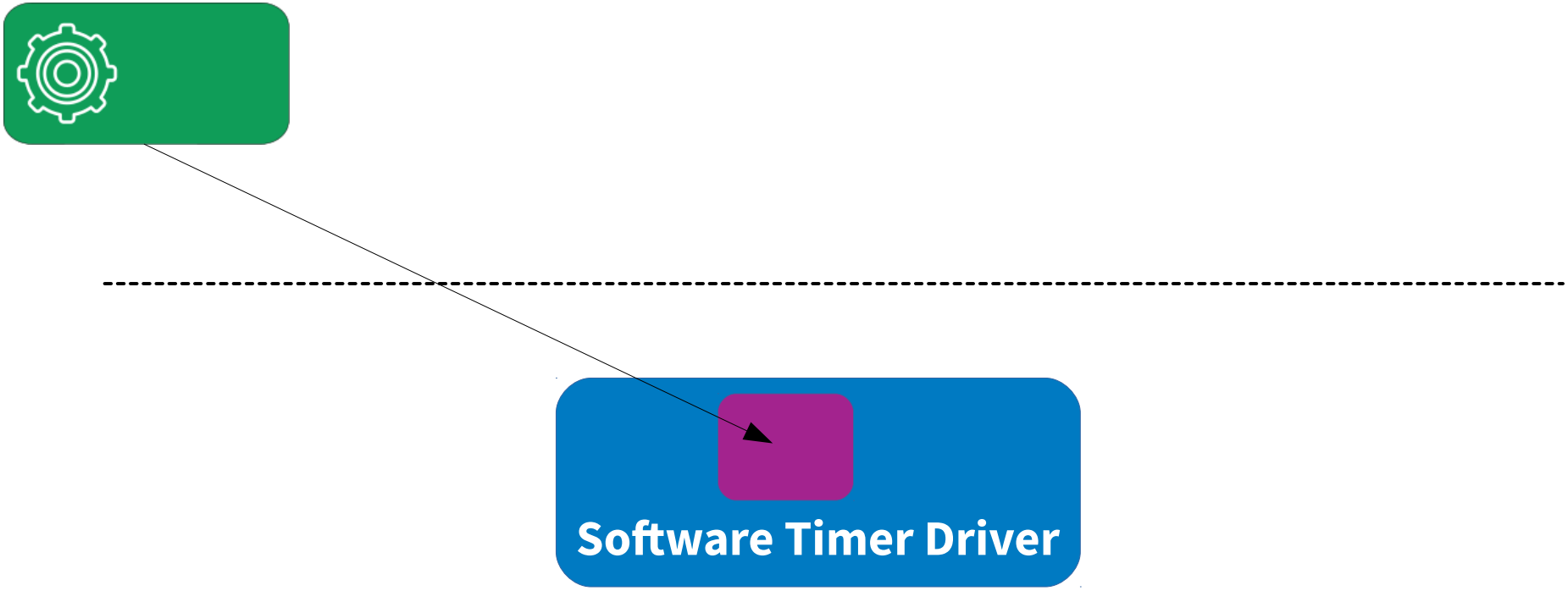


Statically allocate timer state?

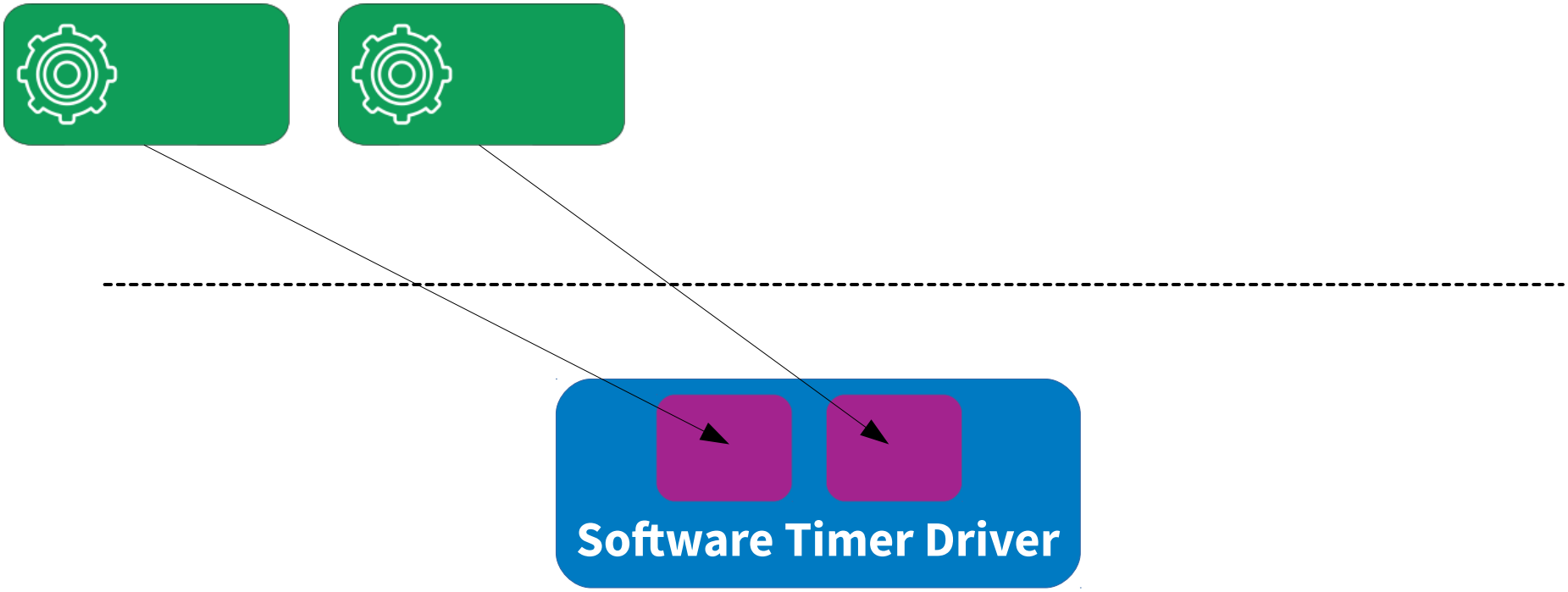


Static allocation must trade off memory efficiency and maximum concurrency

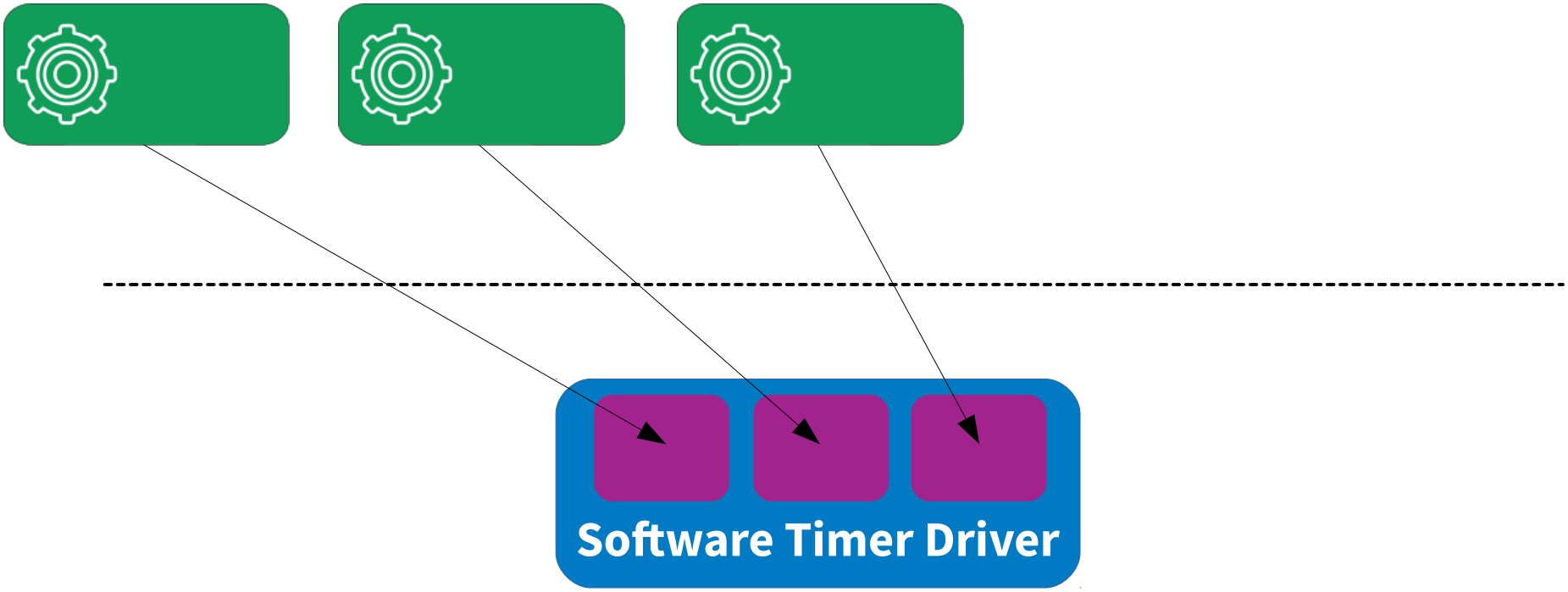
What About Dynamic Allocation?



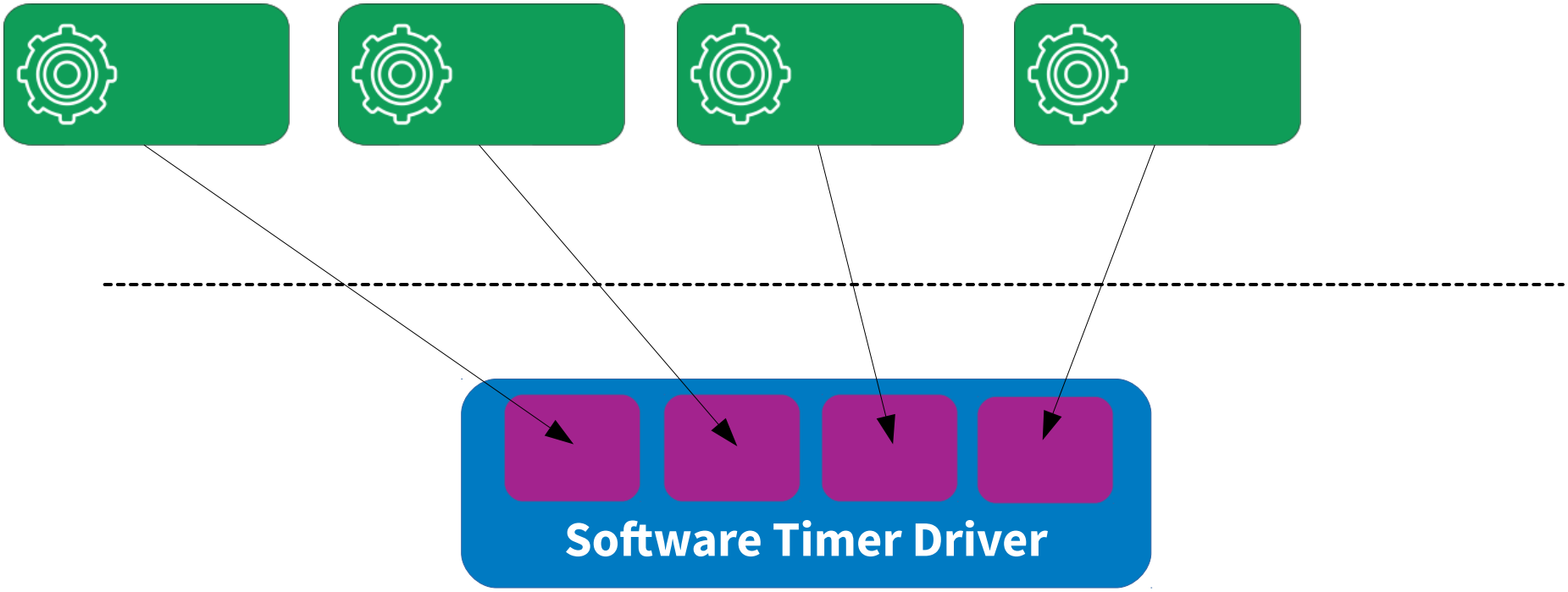
What About Dynamic Allocation?



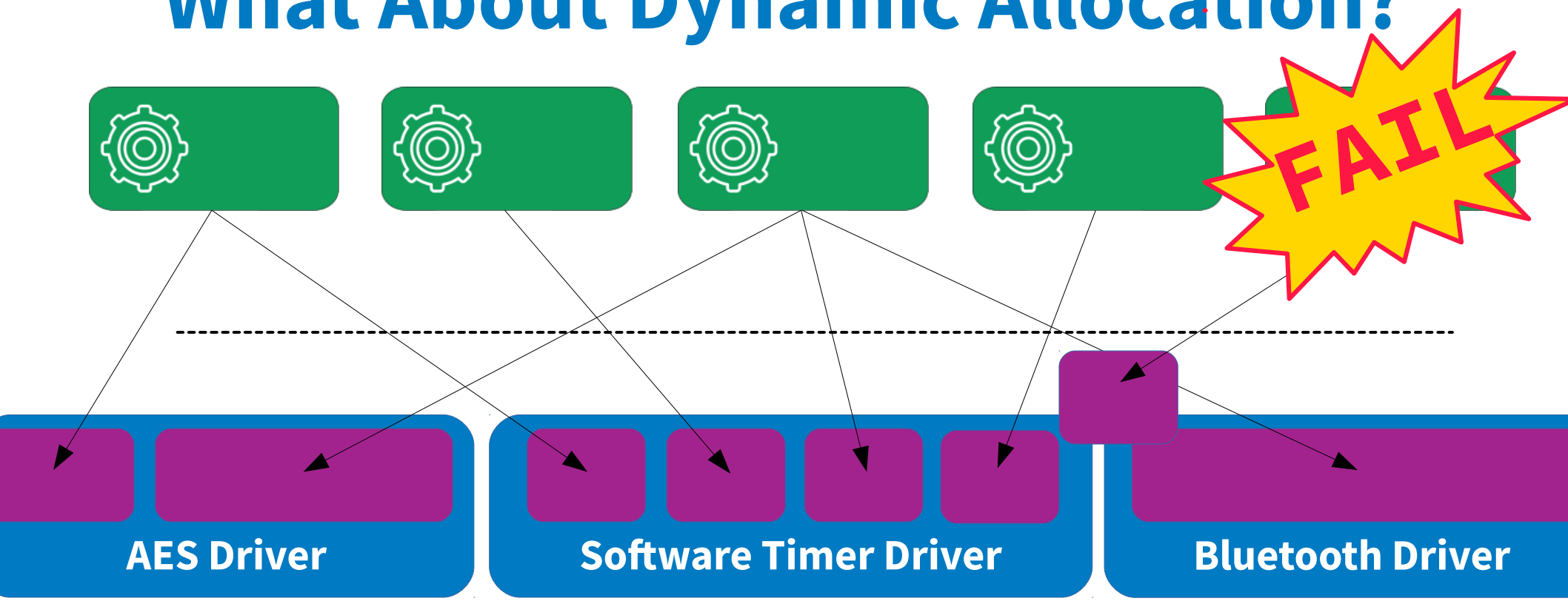
What About Dynamic Allocation?



What About Dynamic Allocation?



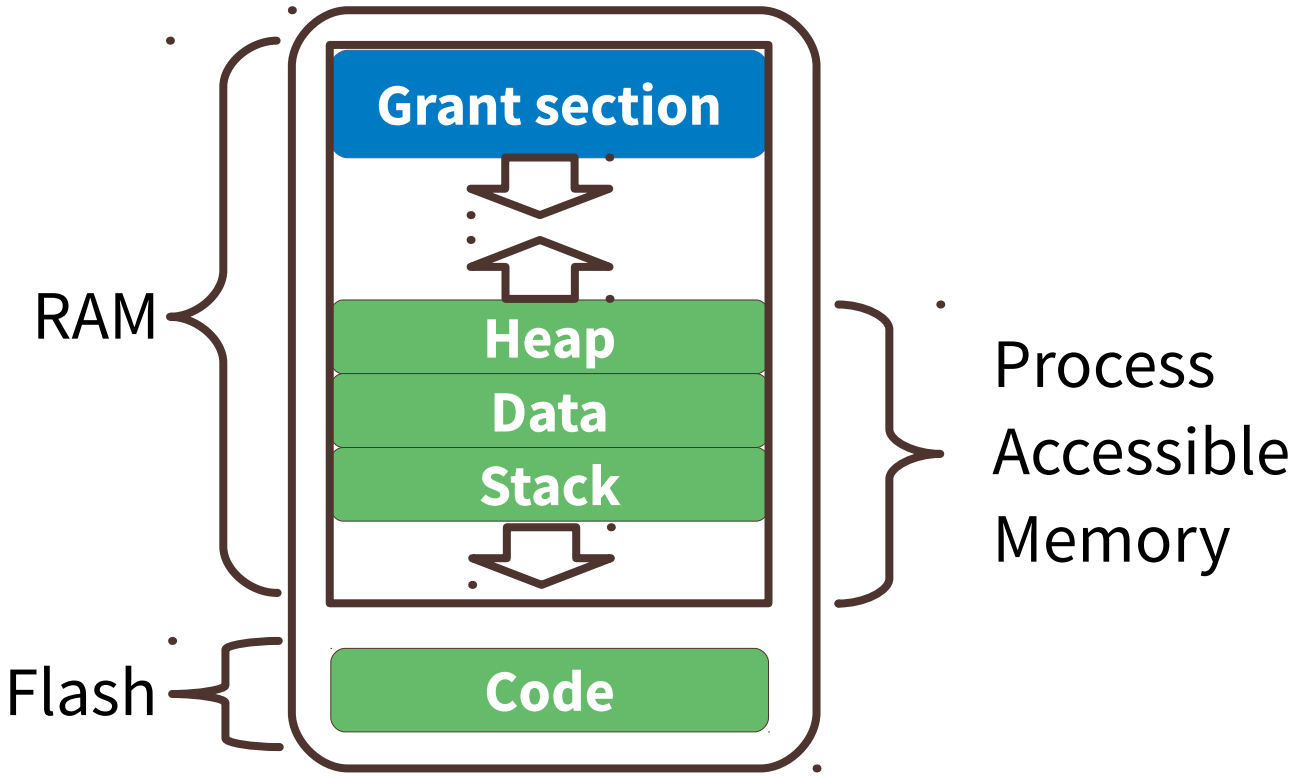
What About Dynamic Allocation?



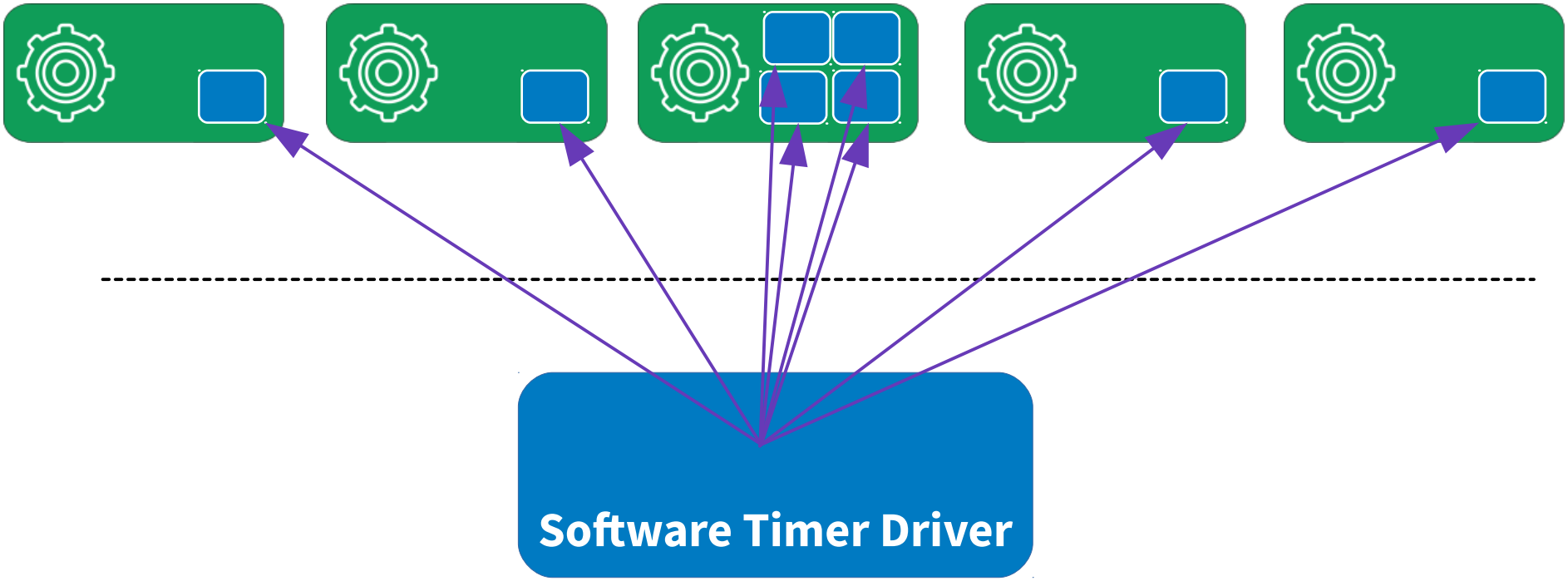
Can lead to unpredictable shortages.
One process's demands impacts capabilities of others.

Grants: Per-Process Kernel Heaps

- Allocations for one process do not affect others
- System proceeds if *one* grant section is exhausted
- All process resources freed on process termination



Grants:
Kernel heap *safely* borrowed from processes



Grants balance safety and reliability of static allocation with flexibility of dynamic allocation

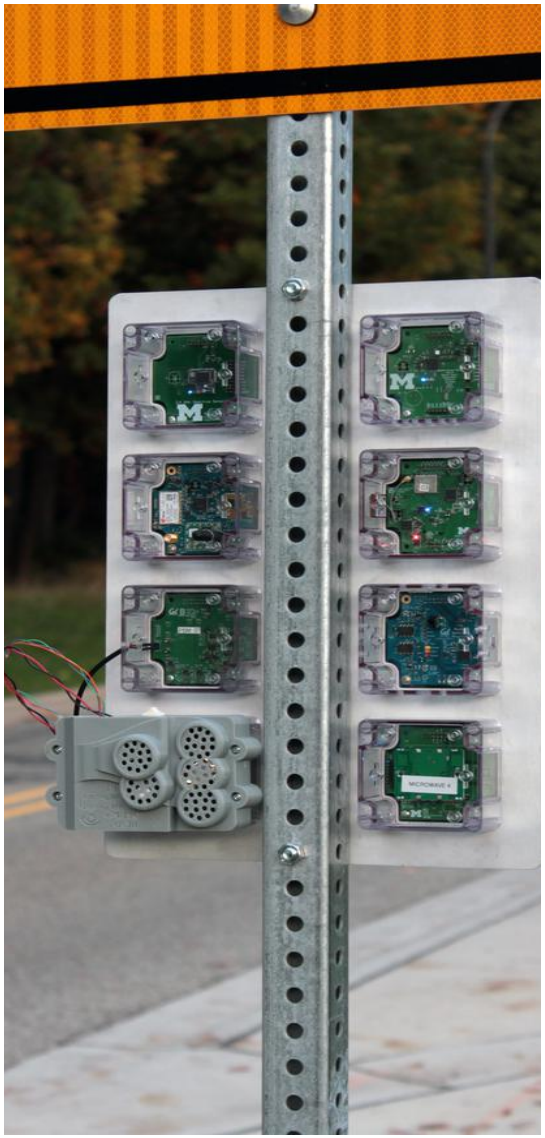
Grants use the type-system to ensure references only accessible when process is live

```
fn enter<'a, F>(&'a self, pid: ProcId, f: F) → where
    F: for<'b> FnOnce(&'b mut T)

    // Can't operate on timer data here

    timer_grant.enter(process_id, |timer| {
        // Can operate on timer data here
        if timer.expiration > cur_time {
            timer.fired = true;
        }
    });

    // timer data can't escape here
```

Case Study: The Signpost Platform

Signpost Overview

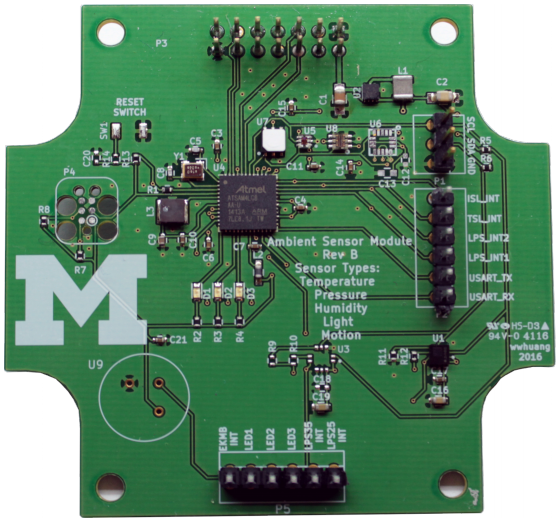
- Modular city-scale sensing platform
  - Ambient conditions tracking
  - Pedestrian density
  - Noise monitoring
- 8 pluggable modules
  - Instead of deploying a new platform
  - 15 mA power budget
  - Microcontroller + Sensors
- Sensing applications
  - Open research platform
  - Mostly run on modules
  - Several apps on the same module

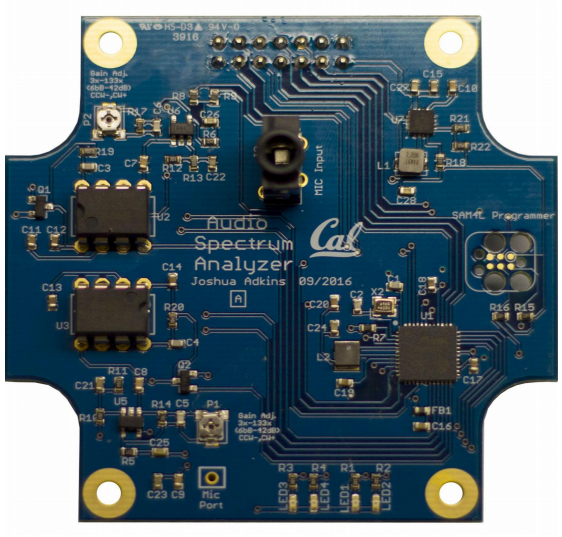


Currently deployed @ U.C. Berkeley

Tock on Signpost

Each Signpost module runs a Tock kernel





	Ambient Module	Audio Module
Process LoC	6990	6688
Capsules LoC	4479	3985
Platform LoC	3252	3244

Limitations & Future Work

- Other kernels in Rust?
- Relationship with verification
- Higher level security abstractions
  - e.g. Permissions? Capabilities? Naming?
- Distributed operating system
- Power management

Conclusion

- Tock: Multiprogramming with 64 kB
- Cheap kernel isolation with type-safe driver API in Rust
- Process abstraction without page virtualization
- Efficient concurrency without a kernel heap

https://tockos.org
github.com/helena-project/tock
@talkingtock
#tock on freenode
tock-dev@googlegroups.com

Thanks to the 40 (and growing) Tock contributers!

Grants Cost at Scale