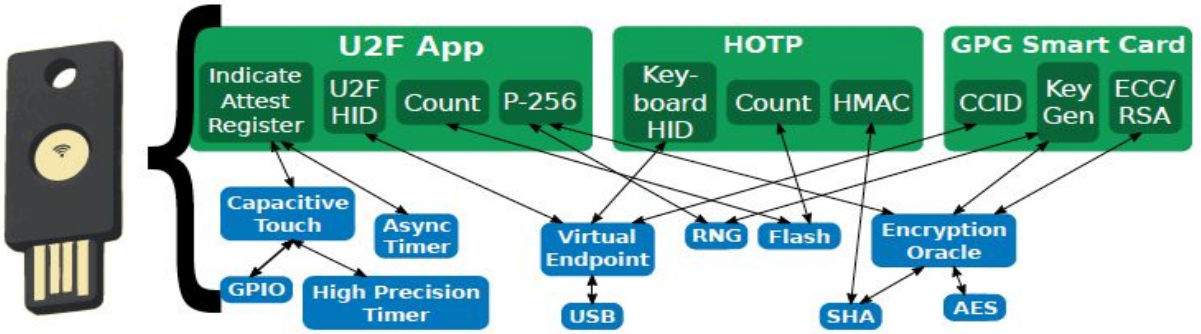
Multiprogramming a 64 kB Computer Safely and Efficiently

Amit Levy et al, Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017

Presented by Chaitra Niddodi

Problem

- Microcontrollers have limited resources – low power budget, low memory capacity, limited hardware protection mechanism
- These systems often use the same memory regions for applications and the OS
- Emerging class of embedded applications are software platforms, rather than single purpose devices
- Lack of support for multiprogramming features – fault isolation, dynamic memory allocation, flexible concurrency



Previous Approaches & Issues

- Give up on isolation - write completely bug-free code -> ***No isolation between components***

- Whole system updates only -> ***Cannot replace individual components without restarting the whole system***

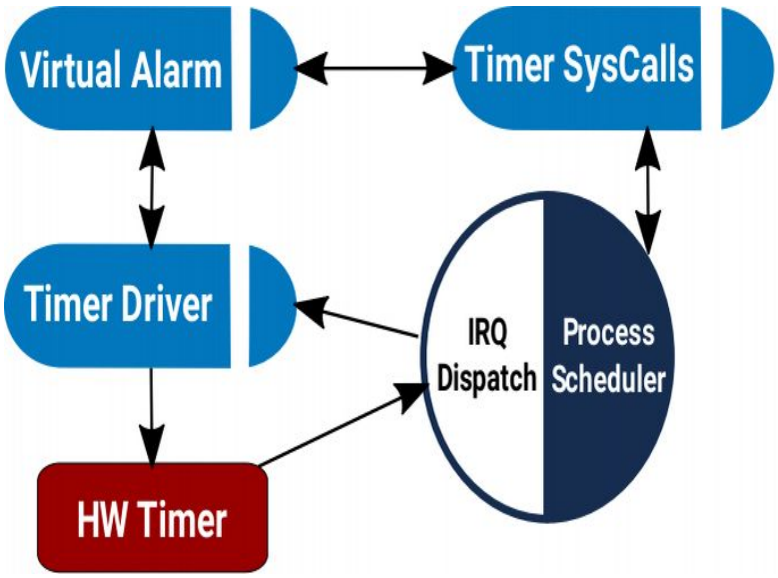
- Static memory allocation to ensure long-running and fault-free operation -> ***Fixed concurrency at compile-time***

Proposed Approach - Tock

- Tock - new operating system for low-power platforms that takes advantage of
  - protection mechanisms provided by Memory Protection Unit
  - type-safety features of *Rust* to provide a multiprogramming environment
- Tock supports
  - ***Isolation of software components***
  - ***Update/restart/remove individual (user-space) components independently***
  - ***Balance safety and reliability of static allocation with flexibility of dynamic allocation***

Capsules

- Rust code linked into kernel
- Event-driven execution with asynchronous I/O
- Shared stack, no heap
- Communicate via references & method calls
- Low overhead
- Used for device drivers, timers
- Trusted



Kernel Memory Consumption

Example 1: “blink”

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

Example 2: Networked sensor

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

Capsule Isolation

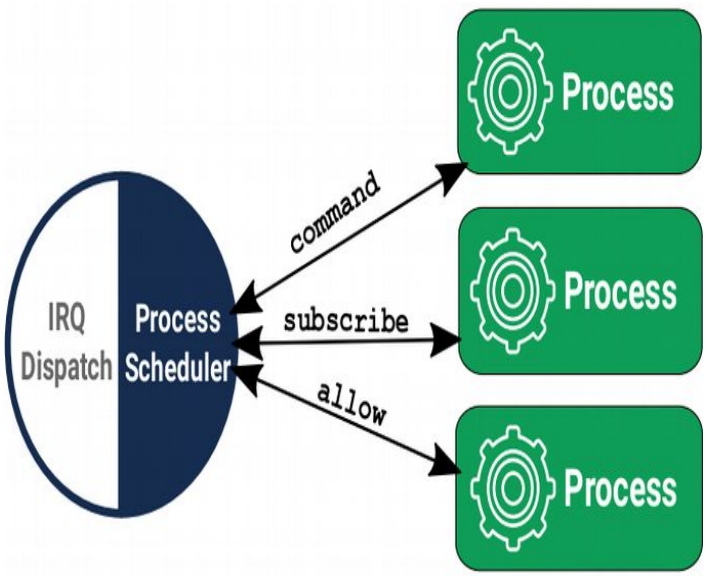
```
struct DMAChannel {
    length: u32,
    base_ptr: *const u8,
}

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*

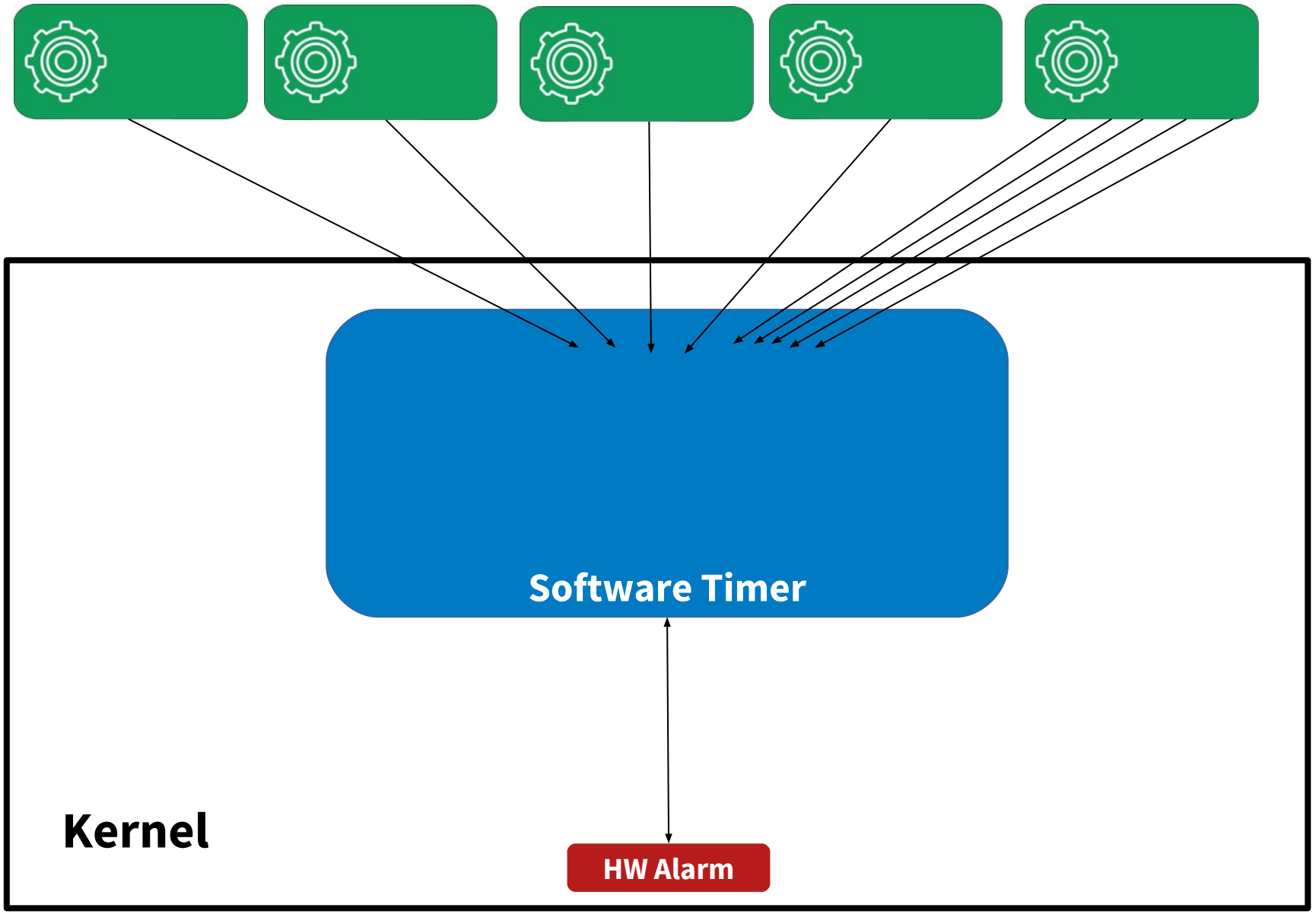
Processes & Isolation

- Standalone executables in any language
- Scheduled preemptively
- System calls & IPC for communication
- Higher overhead - Context switch for communication (340 cycles)
- Untrusted applications
- MPU provides *memory isolation* between applications as well as between applications and the kernel

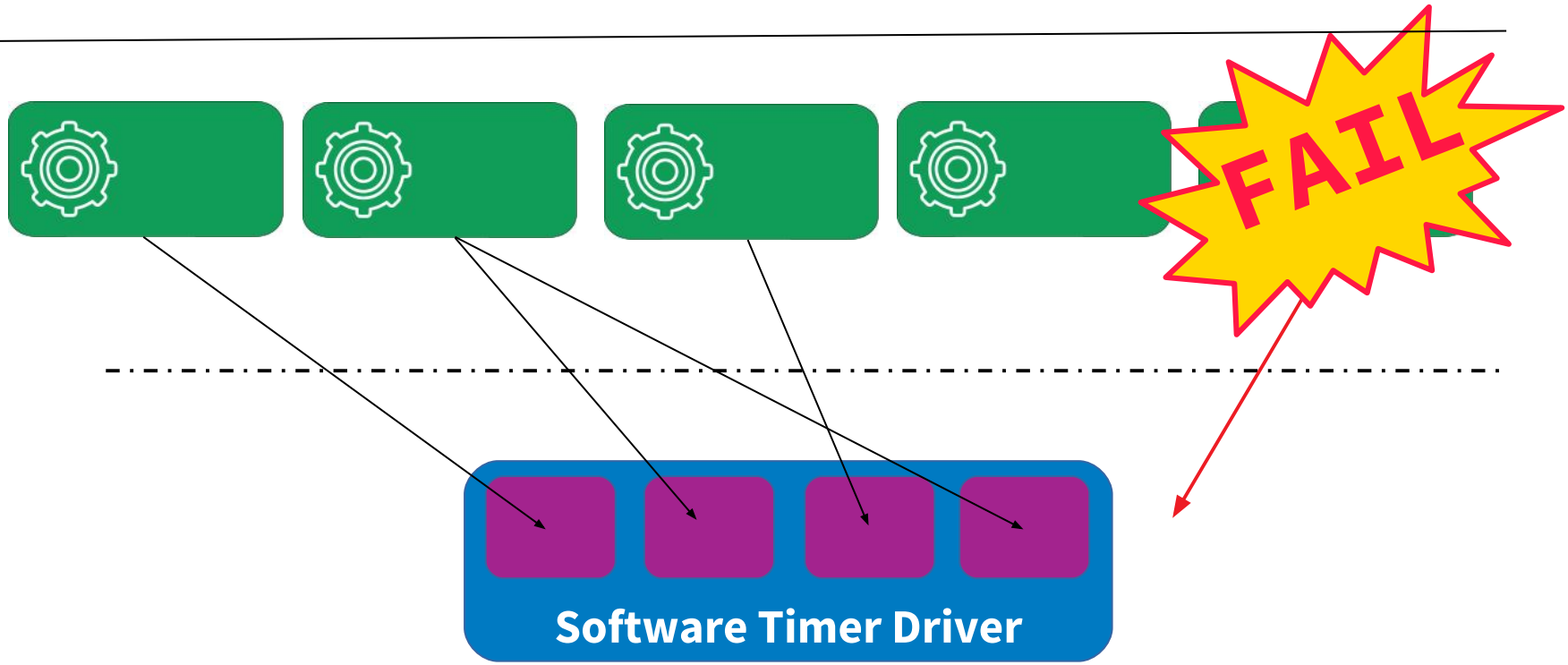


How do capsules and processes interact ?

Example - Software Timer

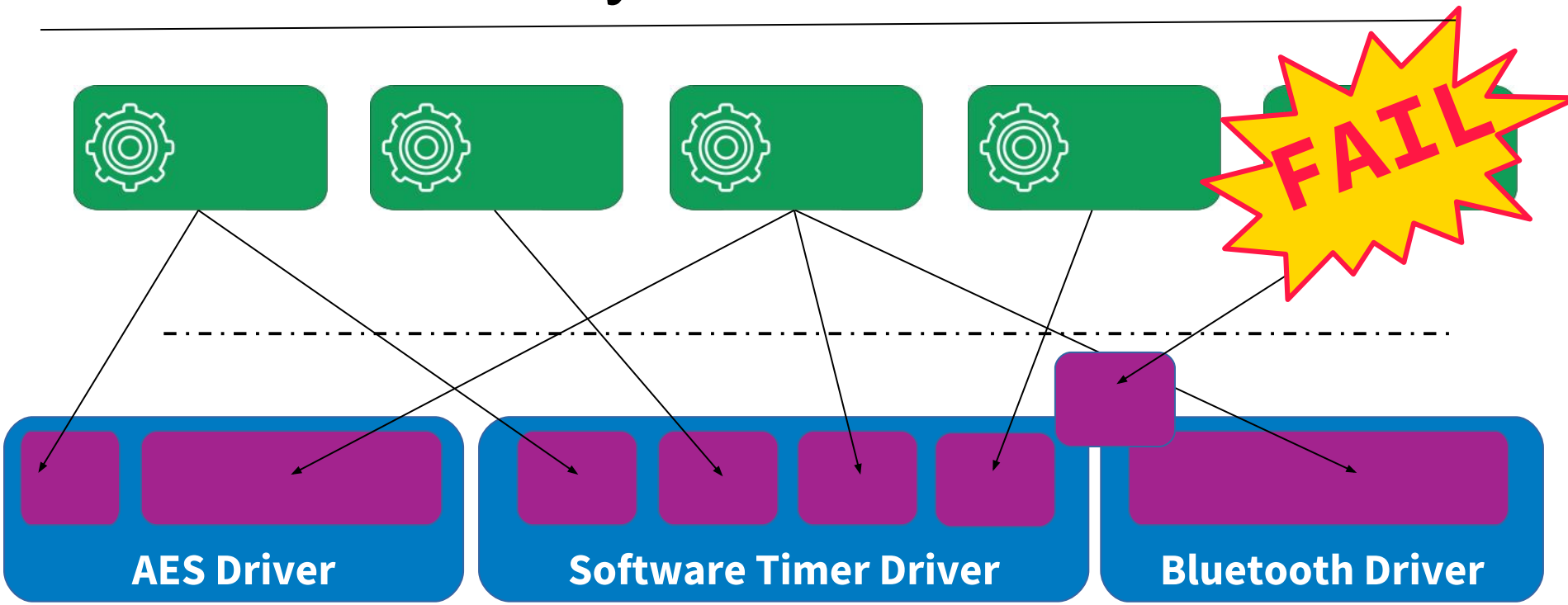


Timer State - Static Allocation



Static allocation must trade off memory efficiency and maximum concurrency

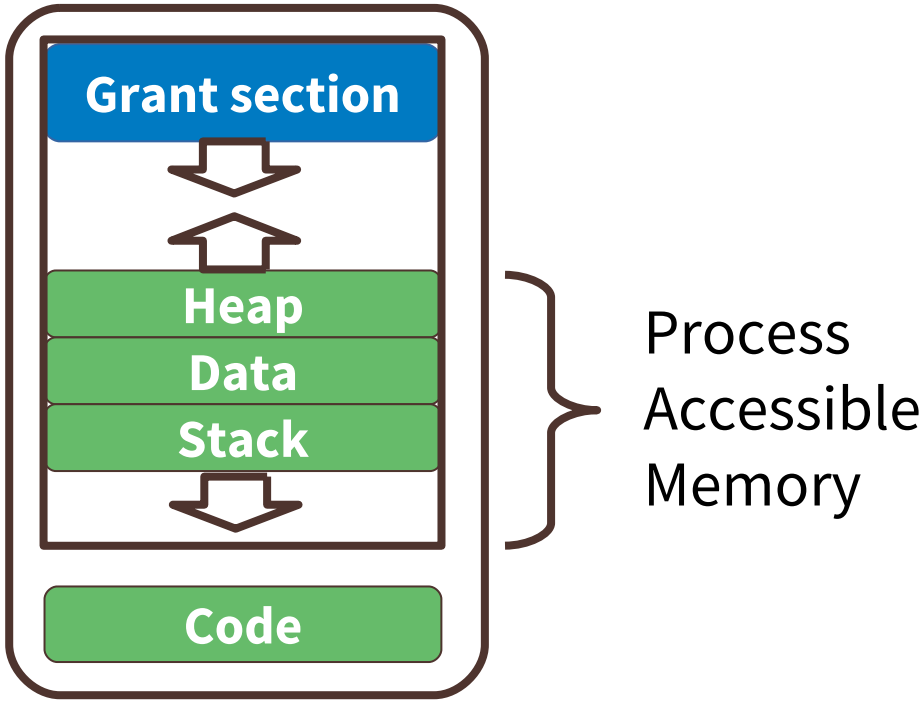
Timer State - Dynamic Allocation



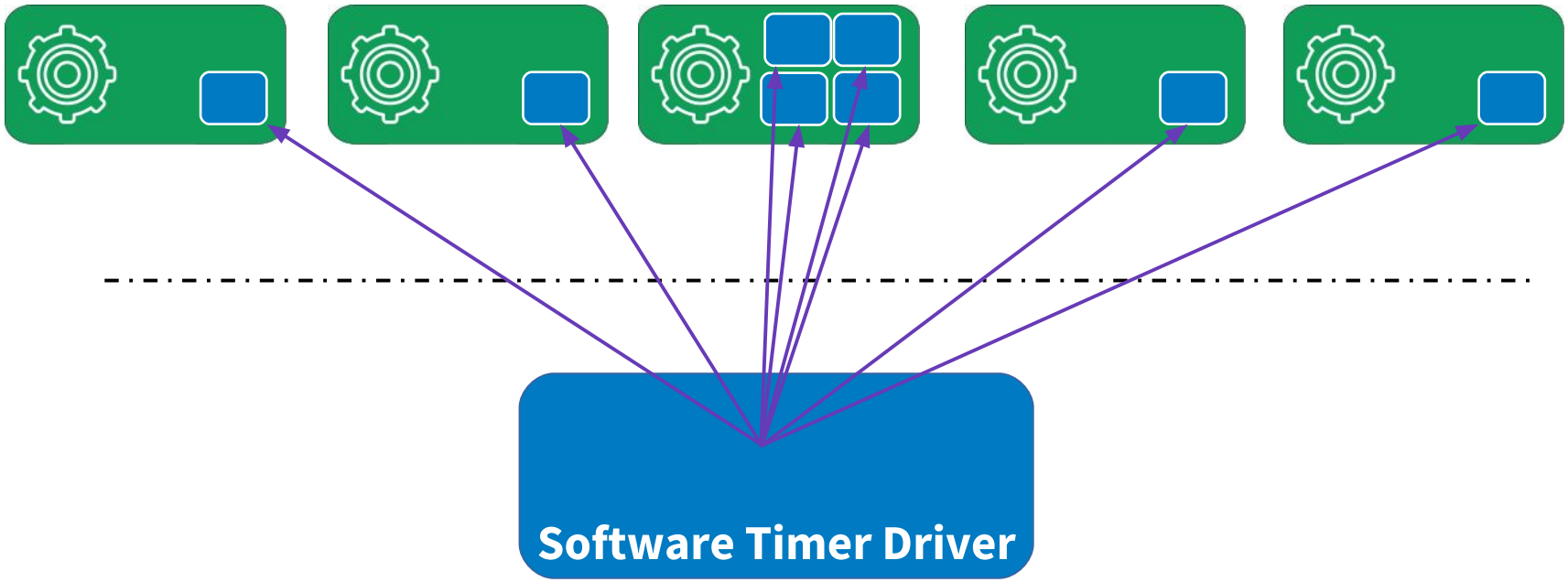
Can lead to unpredictable shortages.
One process's demands impacts capabilities of others.

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 ensure that references are only accessible when process is alive



Grants - Kernel heap safely borrowed from processes

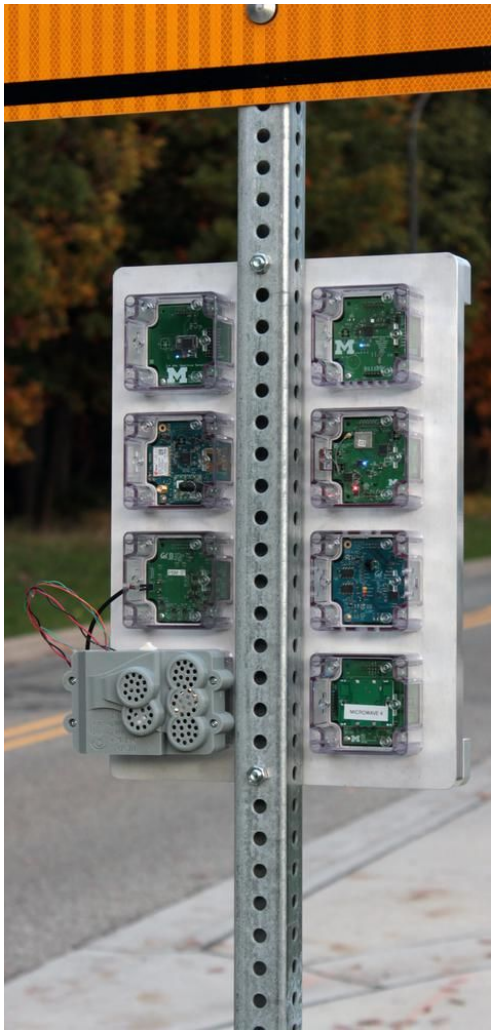


Grants balance safety and reliability of static allocation with flexibility of dynamic allocation

Case Study: The Signpost Platform

Signpost Overview

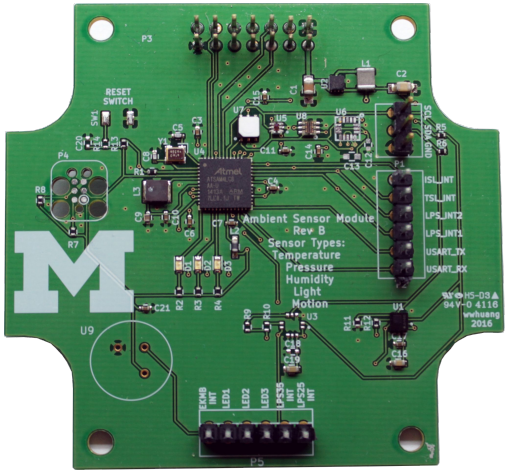
- Modular city-scale sensing platform
  - Ambient conditions tracking
  - Pedestrian density
  - Noise monitoring
- 8 pluggable modules
  - Instead of deploying a new platform
  - 15 mA power budget
  - Microcontroller + Sensors
- Sensing applications
  - Open research platform
  - Mostly run on modules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

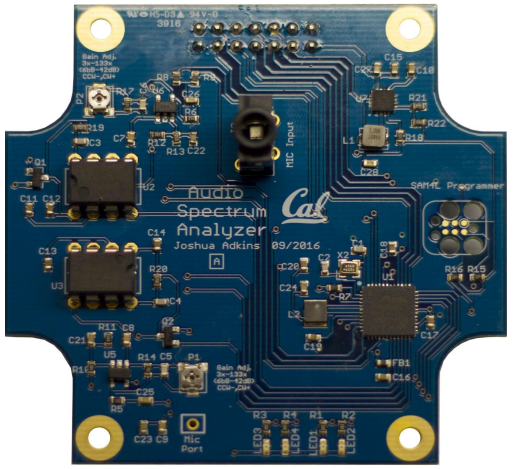


Currently deployed @ U.C. Berkeley

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

Future Work

- Kernels for multicore systems ?
- Higher level security abstractions - e.g. application permissions, specify policies in kernel
- Distributed operating system - platforms like Signpost running multiple microkernels

Comments

- Hardware parallelism ?
- Need to port applications
- Attack/Threat model ? Applications developed by 3rd parties are modelled as malicious
- Dependency on Rust programming language - support ?