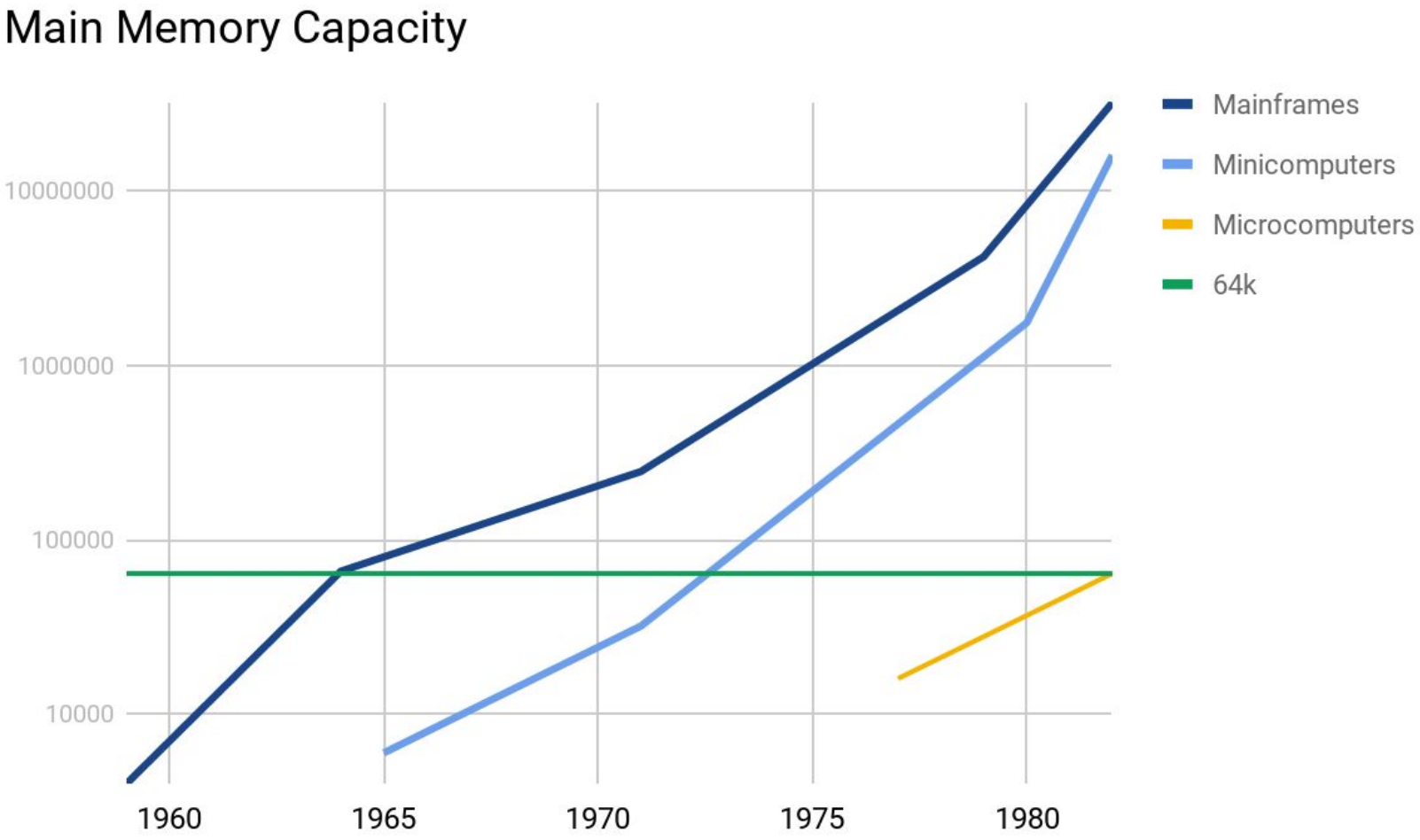
“Multiprogramming a 64 kB Computer Safely and Efficiently”

Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis

Presented by: Sean McBride,

A dude that builds React apps that slurp up 100MB+ of RAM (¬‿¬)



IBM Mainframes, DEC Minis, the Apple II, and the Commodore 64

Who is limited to 64k in 2020?

Who is limited to 64k in 2020?

People that choose to be : Demoscene Artists!

Why { Artists / Disruptive Innovators / Hackers } like Constraints

Daily life is full of obstacles [such as] a lack of resources standing in the way of realizing an ambitious plan.

How do people cognitively respond to such obstacles?

We propose that unless people are inclined to disengage prematurely from ongoing activities, **obstacles will prompt them to step back and adopt a more global, Gestalt-like processing style that allows them to look at the "big picture" and conceptually integrate seemingly unrelated pieces of information.**

Source: Stepping back to see the big picture: when obstacles elicit global processing.

64k Demoscene (2000 - Now)

demo

demoscene

demoparties

Who is limited to 64k in 2020?

People that have to: Microcontroller Developers

But hey… let’s flip the script and embrace the demoscene ethos!

Pebble Smartwatch Gen 1

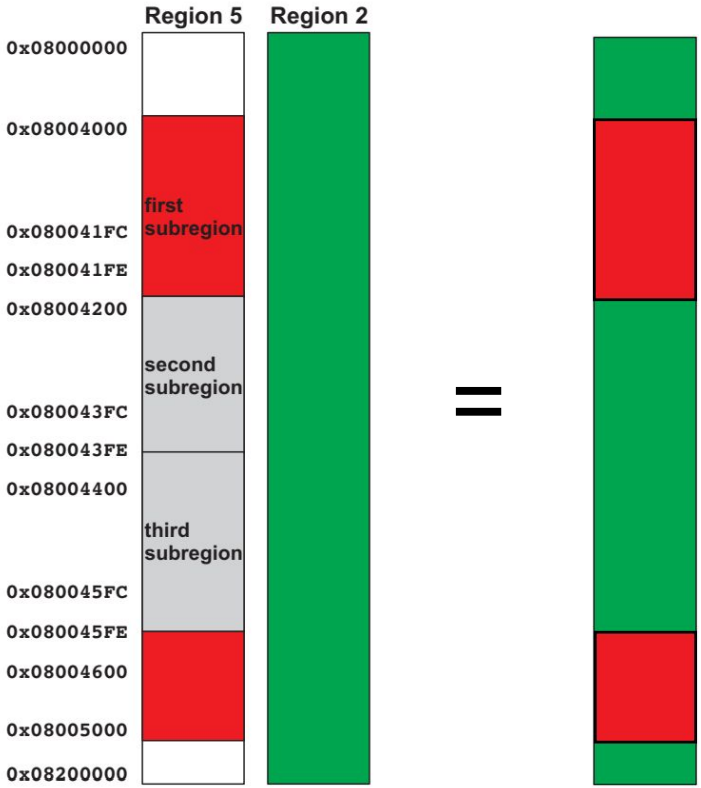
ARM®Cortex®-M3 32-bit CPU

- 120 MHz max
- **Memory protection unit**
- 132kb of SRAM

Pebble

- Customized **FreeRTOS** Kernel
- SDKs for Watchfaces, Apps, etc.
- **App Store**

Memory Protection Units are great!



A lightweight technique for memory protection!

Each MPU frame defines access permissions, memory type, and attributes for a range of addresses.

Each Region is automatically broken into 8 subregions and provides a bitmask to indicate which subregions the policy is enabled on.

Higher Numbered Regions have priority when addresses overlap, enabling **overlays**.

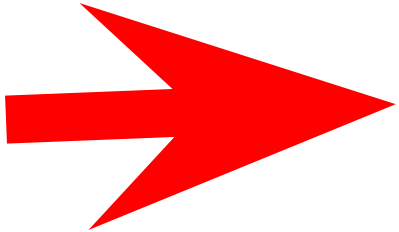
Source: Usage of MPU Subregions on the TI Hercules ARM Safety MCUs

Memory Protection Units aren’t a Silver Bullet!

“FreeRTOS… supports using a memory protection unit to prevent an application from writing to kernel memory. However, **the FreeRTOS system call interface requires the kernel to trust and pointers passed through system calls,** such that an application can make the kernel read or write arbitrary memory”

~Multiprogramming a 64kB Computer Safely and Efficiently

As the Pebble Smartwatch Folks Learned 😿



Pebble Troubleshooter

APP CRASH / UNPREDICTABLE BEHAVIOR

REPEATED APP CRASH or UNPREDICTABLE BEHAVIOR

Please "scrub" – perform these steps – in this order

- go to pebble app
- unload everything from your watch.
- go to your locker
- trash everything from locker - tap on it, tap on trash can
- uninstall pebble app
- restart your phone
- restart your watch (see NOTE 1)
- when phone comes back, install pebble app
- turn developer mode on
- Find latest CGM watchface – if from labs, turn developer mode on; make sure it says NEW
- Load latest CGM watchface, which loads on watch
- put in endpoint

NOTE 1 – For continued problems, you may want to try this again, but factory reset your watch instead of just restarting your watch. On watch, press middle button, go to settings, factory reset, yes. When it comes back, you will have to re-pair the bluetooth of your watch with the phone that has the pebble app, then proceed to next step.

As the Pebble Smartwatch Folks Learned 😿

Dear Pebblers,

Thank you all for being such loyal supporters and champions of the Pebble community and brand. You helped start something fantastic when you backed our first Kickstarter project (and shout-out to the first inPulse users). Since then, we've shipped over 2 million Pebbles around the world!

FreeRTOS' weak implementation of memory protection

However—due to ~~various factors~~—Pebble is no longer able to operate as an independent entity. We have made the tough decision to shut down the company and no longer manufacture Pebble devices. This news has several major implications, and we hope to answer as many questions as possible here, in Kickstarter Update #17, and on our Support site.

TOCK - Loadable Apps that won't wreck your startup!

Dependability (from resource exhaustion)

Application Updates at Runtime

System	Concurrency	Memory Efficiency	Dependability	Fault Isolation	Loadable Applications
Arduino [6]		✓			
RIOT OS [5]		✓			
Contiki [14]	✓	✓			✓
FreeRTOS [8]	✓		✓		
TinyOS [33]	✓	✓	✓		
TOSThreads [28]	✓		✓		✓
SOS [23]	✓	✓			✓
Tock	✓	✓	✓	✓	✓

Powered by the Rust Programming Language

“The problem with C is that [it] doesn't provide you with many mechanisms for making sure that your code is secure. So, there's lots of ways in which you can inadvertently make a mistake and the compiler is not able to help you... **C is by design a very, very sharp tool. And you can cut yourself as well as cutting through things.**

~Brian Kernighan

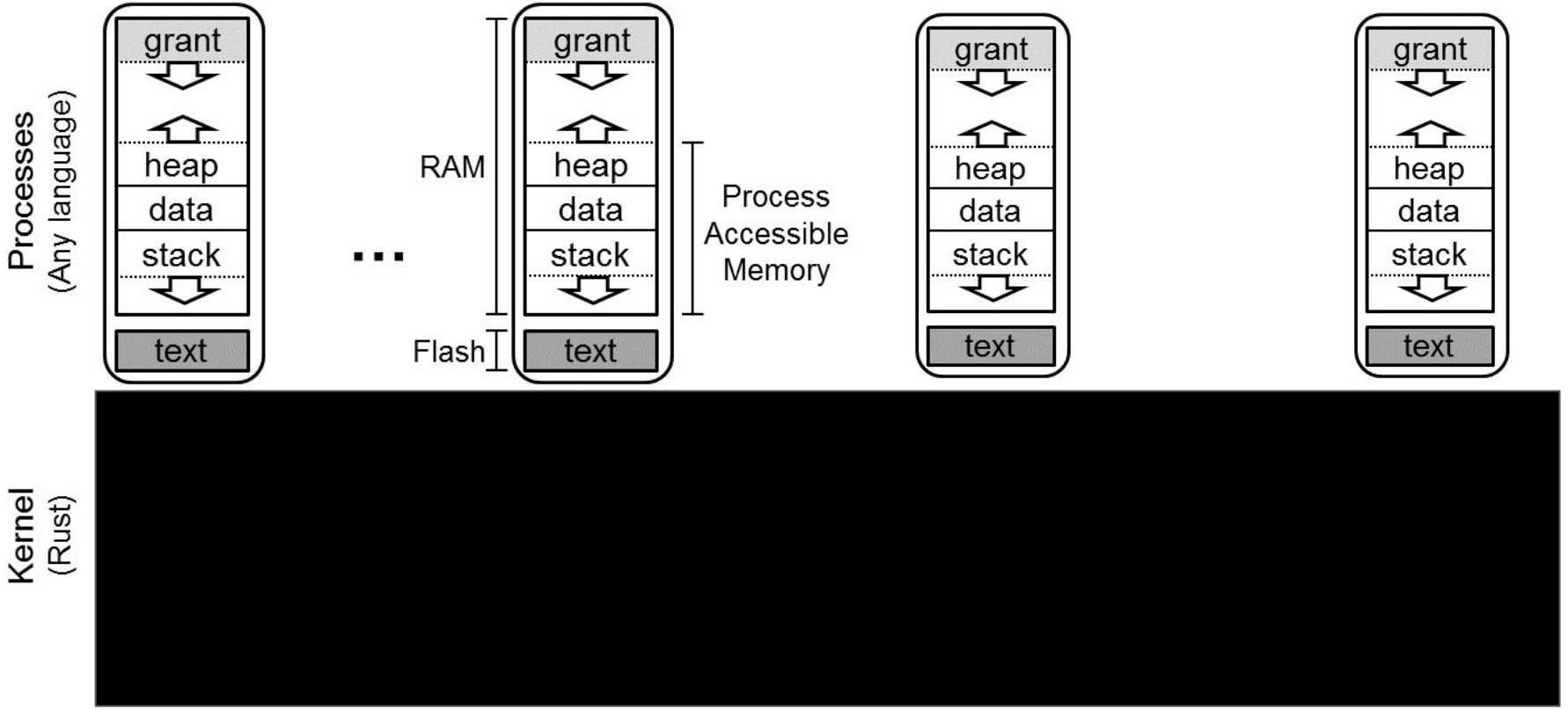
Source: YouTube

“Safety in the systems space is Rust's raison d'être... [It] has been the overriding design priority from the very beginning: no wild pointers, no null pointers, no shared mutable state. Take a look at the very first slide deck. **That's what Rust has always been about: bringing safety features to the systems niche,** a niche that has stubbornly resisted them for decades while higher-level languages have moved ahead in safety.”

~Graydon Hoare

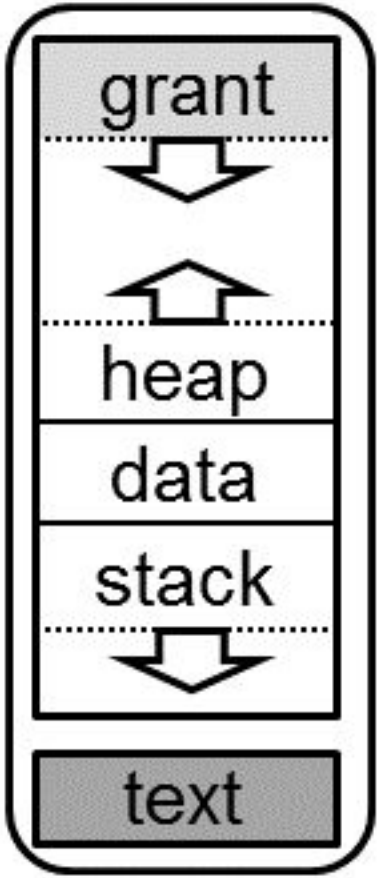
Source: Rust is mostly safety

System Design



Processes

- A **preemptable sandbox** for untrusted apps
- Capabilities
  - **Run untrusted code** in an arbitrary language
  - Issue non-blocking IPC to other processes
  - **Tell the kernel to do something** w/ COMMAND **syscall**
  - **Register a callback to run on an event** with SUBSCRIBE
  - Pass a buffer to the kernel via ALLOW
  - **YIELD, blocking until a callback is ready to run**
- How this differs from xv6
  - Not virtual memory! Addresses are all backed!
  - Syscalls are non-blocking
  - “kernel space” is dynamic via the Grants API and MPU

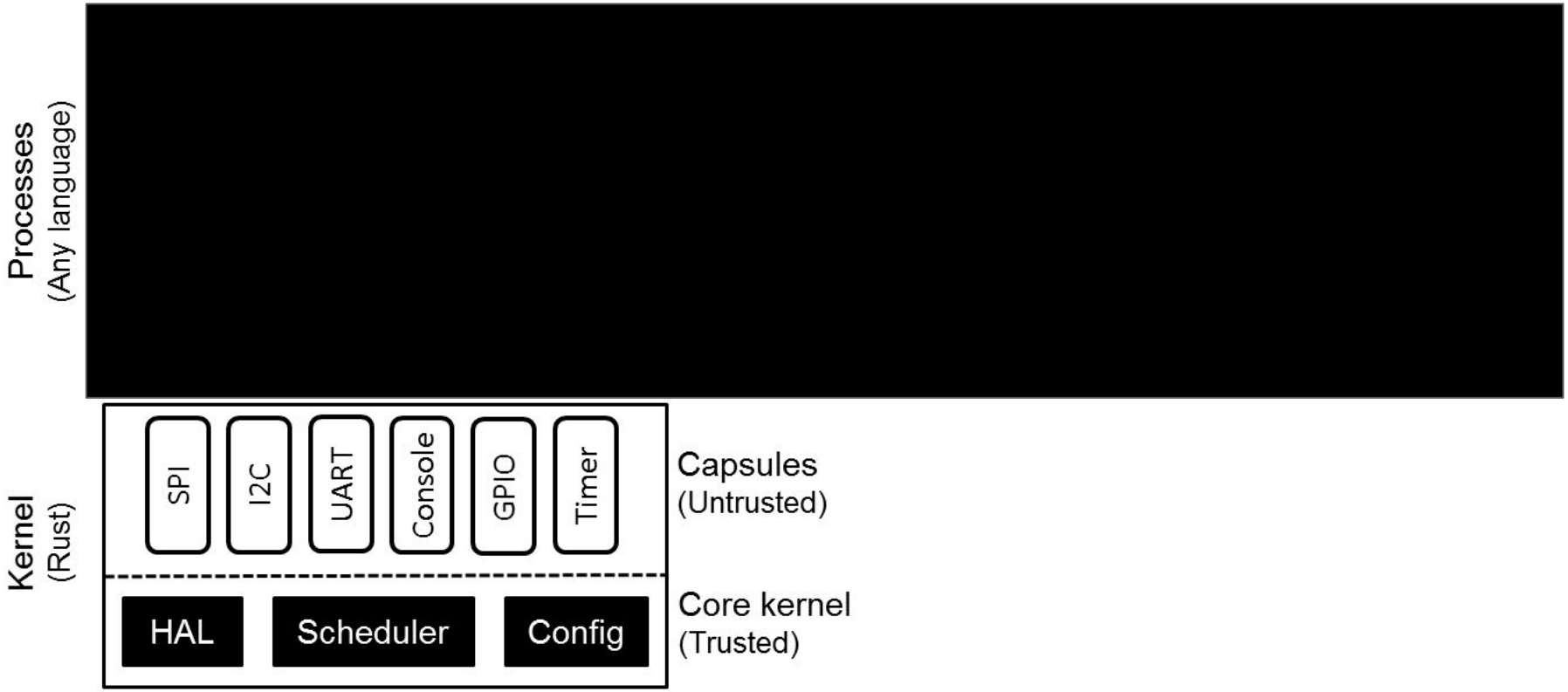


Grants API

- The MPU marks a range of addresses as privileged and **inaccessible to the process**
- Kernel data structures are **allocated** there **dynamically** and **lazily** (during first use).
- If the privileged grant region is exhausted, it is grown (similar to sbrk)
- When a process terminates, it drops the entire address space, including all grants!

```
// Here is an example of how the grants are laid out in a
// process's memory:
//
// Mem. Addr.
// 0x0040000  ┌────────────────────
//            │   GrantPointer0 [0x003FFC8]
//            │   GrantPointer1 [0x003FFC0]
//            │   ...
//            │   GrantPointerN [0x0000000 (NULL)]
// 0x003FFE0  ├────────────────────
//            │   GrantRegion0
// 0x003FFC8  ├────────────────────
//            │   GrantRegion1
// 0x003FFC0  ├────────────────────
//            │
//            │   --unallocated--
//            │
//            └────────────────────
//
// An array of pointers (one per possible grant region)
// point to where the actual grant memory is allocated
// inside of the process. The grant memory is not allocated
// until the actual grant region is actually used.
```

System Design



Capsules

- A **3rd-party** Tock **driver** written in Rust for **new sensors, syscalls, etc.**
- Memory protection provided by Rust Type System
- Cooperatively scheduled and **untrusted**, so an buggy capsule can cause an infinite loop that takes the entire system down
- Capabilities
  - **Exposes actions and events to processes** by implementing the Driver trait (interface)
  - **Execute actions** as directed by a COMMAND syscall for a process
  - **Triggers events**, executing callback handlers processes register vis SUBSCRIBE
  - **Access public Struct fields** exposed by other capsules
  - **Invoke public methods** exposed by other capsules
- State: Frames on a Global Kernel Stack and Per-process Grants

What is the Memory Overhead of Capsules?

System	text (B)	data (B)	bss (B)	Total RAM (B)
Tock	3208	812	104	916
TinyOS	5296	0	72	72
FreeRTOS	4848	1080	1904	2984

“Hello World” Blink App

System	text (B)	data (B)	bss (B)	Total RAM (B)
Tock	41744	2824	6880	9704
TinyOS	39604	1228	9232	10460

Real World App

Bottom Line: Tock’s capsules provide kernel-level isolation while maintaining a memory footprint inline with existing RTOSes that lack isolation

What is the Memory Overhead of Processes?

Current Process Data Structure	164 bytes
RIOT-like Task Structure	14 bytes
Process Stack	256-512 bytes

Signpost w/ Processes	13kB (RIOT + 512b stacks) 100kB (Status Quo + 4kB stacks)
Signpost w/ Capsules	12kB

How do we size a process or kernel stack?

“Since the stack size must be able to fit the largest depth a process may ever reach, **accurately choosing is difficult** and stacks are often over-allocated.”

What is the Latency Overhead of Capsules? Processes?

Process Context Switch	340 cycles
Capsule-to-Capsule Function Call	0-4 cycles

	Handler	Latency
No sleep	Interrupt Service routine	0.87 µs
	Tock Capsule	2.03 µs
	Tock Process	36.8 µs
From deep sleep	Interrupt Service routine	2.29 ms
	Tock Capsule	2.29 ms
	Tock Process	2.34 ms

Critique

Numerous innovations throughout the stack, leading to a novel RTOS design.

Thorough evaluation with good baselines.

Heavily downplayed the latency and memory overheads of processes, discussing possible future enhancements not in the mainline codebase (a linked list of grants, a RIOT-like Task Structure).

Barely mentioned that Signpost had to use capsules instead of processes to fit on the microcontroller.

Conclusions

Provides processes for preemptive execution of untrusted code

Provides capsules, lightweight kernel isolation for drivers and kernel extensions

Novel use of a shared kernel stack and grants to limit kernel memory consumption

Also generally a VERY POLISHED open-source project with good documentation, tutorials, and supported microcontrollers for sale on their website. Seems like a good possible choice for course projects!

https://www.tockos.org/

Hail Development Module

$60.00

BUY NOW

Hail is an IoT development module that runs the Tock operating system. It features:

- SAM4L Cortex-M4
- nRF51822 BLE Radio
- SI7021 Temperature and Humidity Sensor
- ISL29035 Light Sensor
- FXOS8700CQ 6-axis Accelerometer and Magnetometer
- RGB LED
- USB Programming
- Particle Photon Form-Factor and Pinout (0.8" x 1.44")

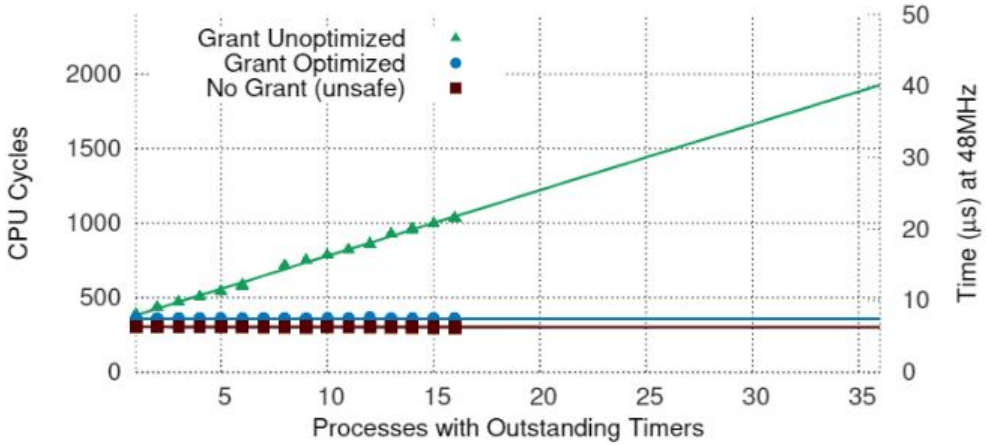
Backup Slides

Algorithmic Overhead of having to “map”

Even if a timer is set for a single process, the timer capsule still has to map over each process to check for a match.

x = # processes

f(x) = 387 + 44(x-1) cycles



The “Grant Optimized” solution treats the processes as a linked list sorted in order of timer expiration. Map is modified to walk the linked list.

Five Requirements of a Modern Embedded OS

- Concurrency
  - Run things in parallel to spend more time sleeping
- Dependability from Resource Exhaustion
  - By allocating all memory statically
  - By testing everything in advance
- Fault Isolation
  - By using a bytecode interpreter
  - By using the MPU
- Memory Efficiency
  - By dynamically loading modules
- Application Updates at Runtime

64k Memory costs $80k (early 70s)

64k was the max capacity for the IBM System/360 Model 30

“[I]n the early 70's a company in Edina Minnesota, Fabri-tek, manufactured core memory for IBM computers. 32K was about $40,000 dollars”. -Ed Thelen

64k DRAM causes Trade War (late 70s)

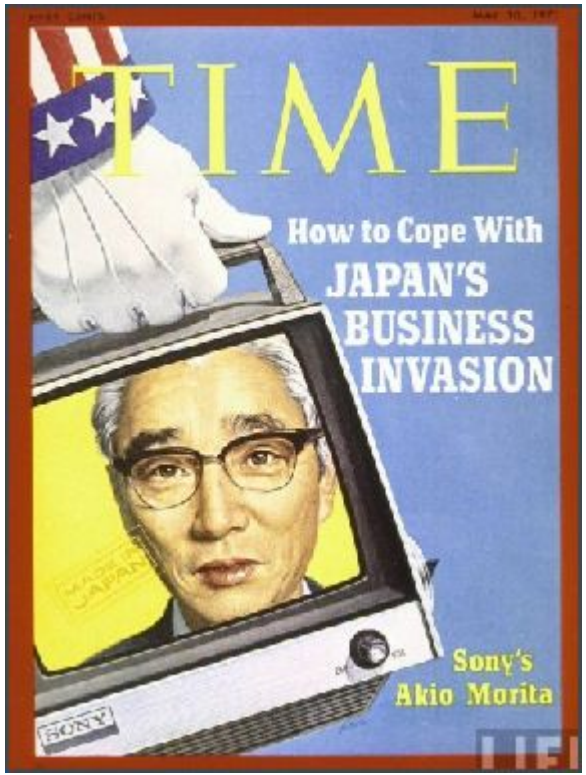
You likely know about the geopolitical tension over Huawei and 5G.

“In 1976, the Japanese Trade Ministry saw a chance to make Japan a leader in this new industry. It funded Fujitsu, Hitachi, Mitsubishi, NEC, and Toshiba to develop 64K DRAMs. The consortium triumphed, decimating American memory suppliers and provoking the U.S. government to threaten trade sanctions.

Eventually, a Japanese-Americans agreement eased tensions. But it didn’t ease competition.

Korea soon eclipsed both.”

Source



64k Home Computers can game (80s)

FitBit Flex Gen. 1 (STM32L161C6)

ARM®Cortex®-M3 32-bit CPU

- From 32 kHz up to 32 MHz max
- Memory protection unit
- 32kb of SRAM
- Ultra-low-power platform, down to 0.3 µA in standby