Predictable Virtualization on Memory Protection Unit-based Microcontrollers

Runyu Pan, Gregor Peach, Yuxin Ren, Gabriel Parmer

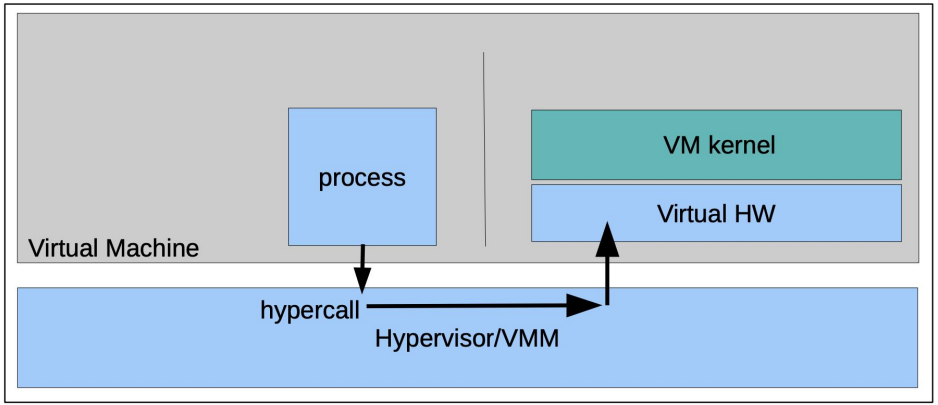
Presented by: Sam Frey

Virtualization

...is the process of running a virtual instance of a computer system in a layer abstracted from the actual hardware.

Each virtual instance *thinks* that it has complete control over a resource.

This is largely due to address space virtualization.



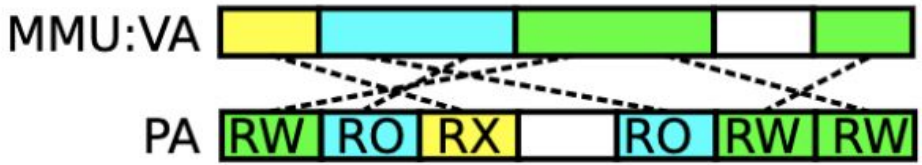
Credit for graphic: Gabe Parmer (CSCI 3411, Fall 2019)

Memory Management Unit

...is what actually provides address virtualization by translating an instance's virtual addresses into physical memory addresses.

Virtual addresses can be mapped anywhere in physical memory, **and they don't need to be contiguous.**

Translating virtual addresses is expensive.



Credit for graphic: Authors

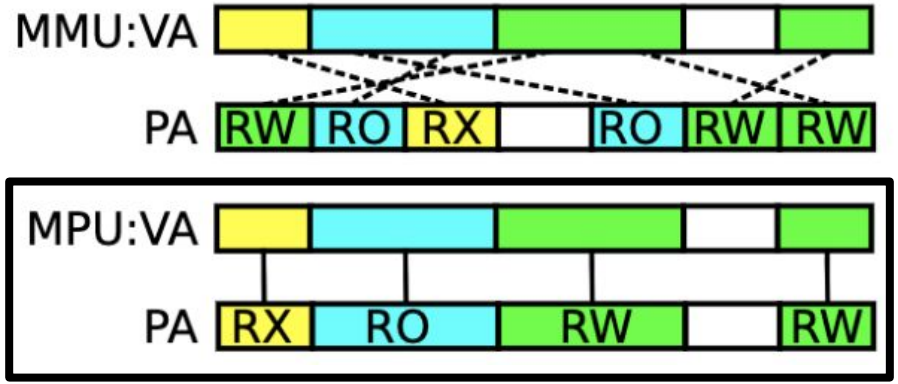
Memory Protection Unit

...provides protection over a region of memory with a given size and starting address.

With a MPU, the OS must run with a single address space (SASOS), and **regions must be mapped contiguously.**

MPUs require a static allocation of memory.

MPUs can only protect a limited number of regions.



Credit for graphic: Authors

The Problem

SASOS leave much to be desired when it comes to security and reliability.

Can isolation of each RTOS on the system be achieved using virtualization within the constraints of an MPU?

Can virtualization be achieved with a low enough cost to make the increased overhead worthwhile?

The Solution

- Pre-planned static memory layout
- Capability-based memory access protections
- Virtualize microcontroller hardware protections to provide multi-tenancy

This was accomplished using FreeRTOS for each VM and Composite OS as the virtual machine manager and host OS.

Memory Layout

Meeting MPU constraints requires a static memory layout.

Initial algorithm attempts used Satisfiability Modulo Theory, but took too long.

A greedy heuristic was added to shorten runtime.

Algorithm 1: Memory Address Assignment Heuristic

Input: C: Set of components, A: Set of SRAM arenas

while $|\{c \in C \mid \texttt{num\_allocated\_regions}(c) > |\mathcal{R}| - 1\}| > 0$ **do**
  $collapsable = \{c \in C \mid \exists_{a \in A} \texttt{is\_enabled}(a, c)\}$
  if $|collapsable| = 0$ **then**
    return *None*
  $c = \underset{c \in collapsable}{\arg\max}$ num_allocated_regions (c)
  $a_0 = \underset{a \in \texttt{accessible\_enabled\_arenas}(c)}{\arg\max} |\texttt{users}(a)|$
  $\mathcal{O}$ = accessible_enabled_arenas (c) $\setminus a_0$
  $partners = \{a \in \mathcal{O} \mid \texttt{subregions}(a) + \texttt{subregions}(a_0) \leq S\}$
  if $|partners| = 0$ **then**
    disable_arena (c, a_0)
    continue
  $a_1 = \underset{a \in partners}{\arg\max} |\texttt{users}(a_0) \cap \texttt{users}(a)|$
  a_{merged} = merge_arenas (a_0, a_1)
  $A = (A - \{a_0, a_1\}) \cup \{a_{merged}\}$
  reenable_all_arenas ()
end
return assign_addresses *(A)*

Credit for pseudocode: Authors

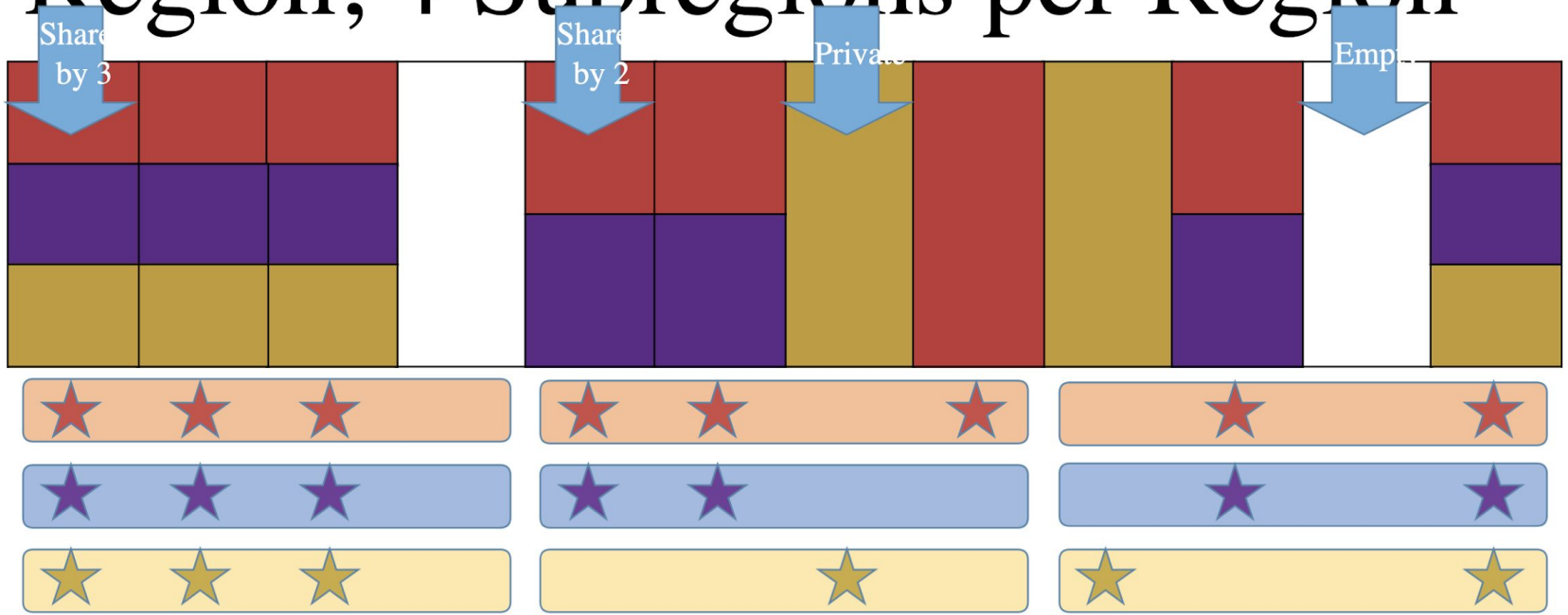
The Algorithm

Goal: Use as few memory regions as possible.

⇒ Maximize use within each region

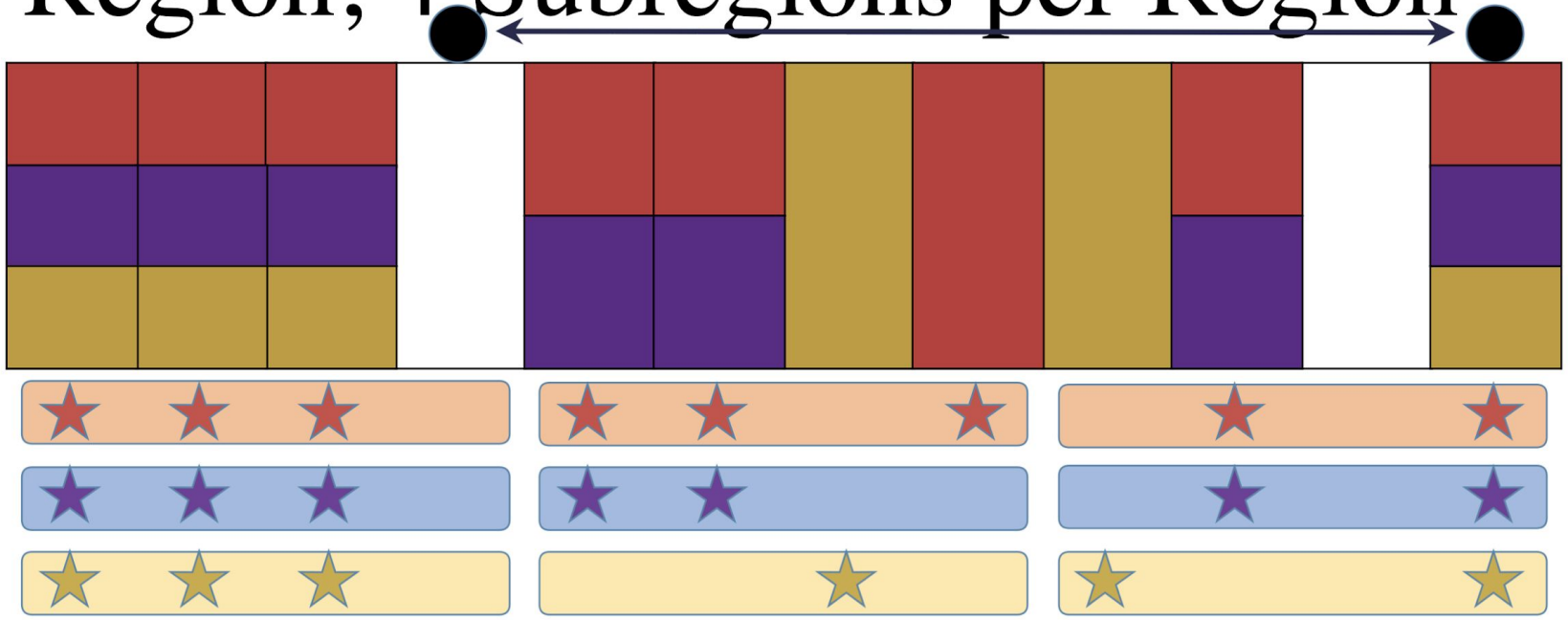
⇒ Move similar subregions into the same memory region.

2 Region, 4 Subregions per Region



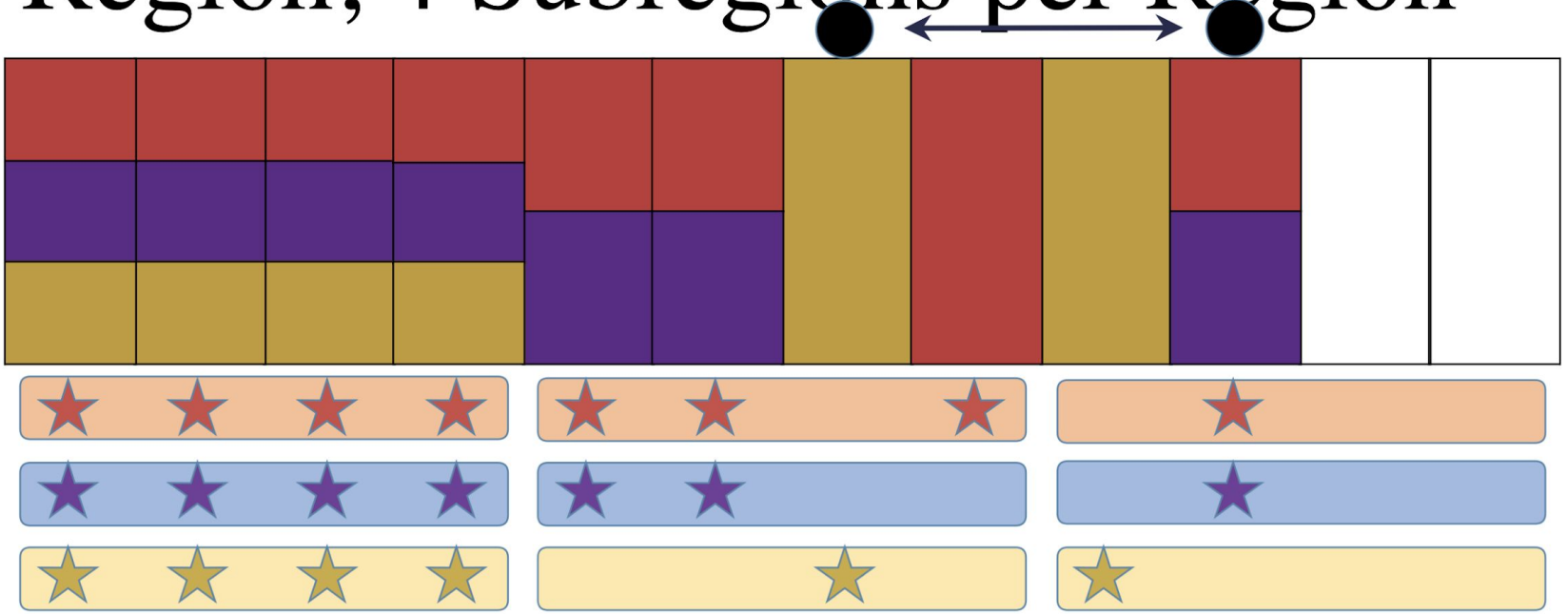
Requires 3 regions - Ran out of regions!

2 Region, 4 Subregions per Region



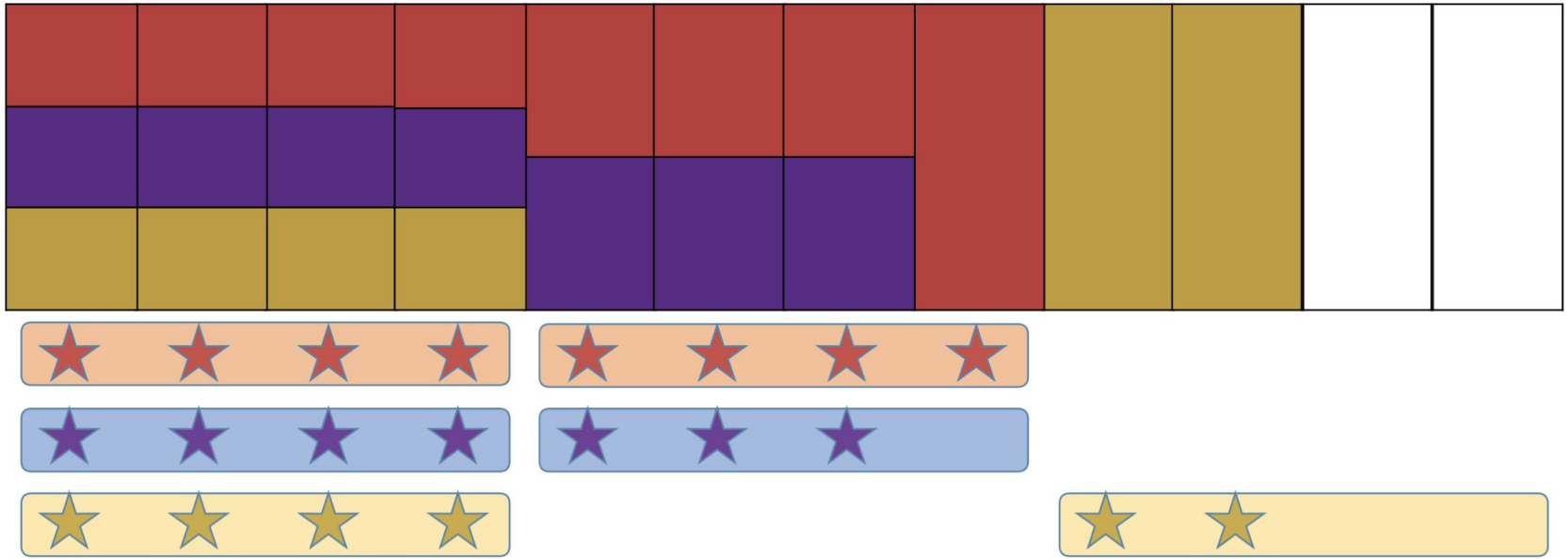
Requires 3 regions - Ran out of regions!

2 Region, 4 Subregions per Region



Requires 3 regions - Ran out of regions!

2 Region, 4 Subregions per Region



Requires 2 regions - Placement feasible!

Memory Isolation

No MMU means no virtual addresses.

Security and access control are provided by **capabilities**, unforgeable tokens that reference system resources.

They refer to a value that references an object along with an associated set of access rights.

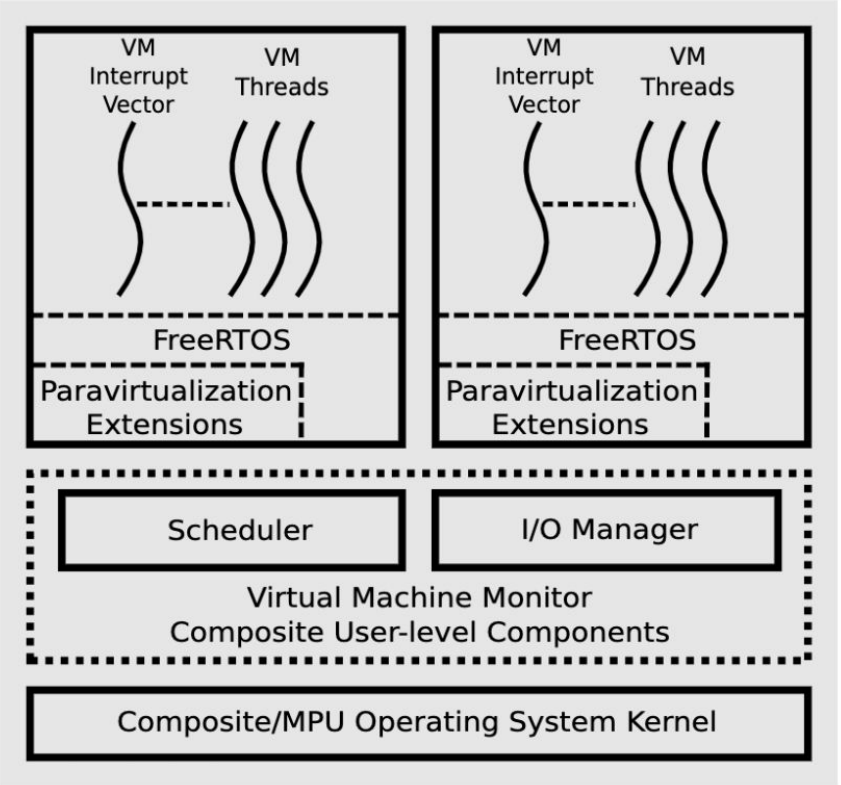
	Mem 1	Mem 2	Mem 3	Sensor	Actuator
VM 1	e	r/w		r	r/w
VM 2		r	e		r
VM 3		r/w	e	r	r/w

Virtualization

The host OS and each VM all have their own scheduler.

VMs make device data requests through the I/O manager.

The I/O manager multiplexes the devices required by multiple VMs.

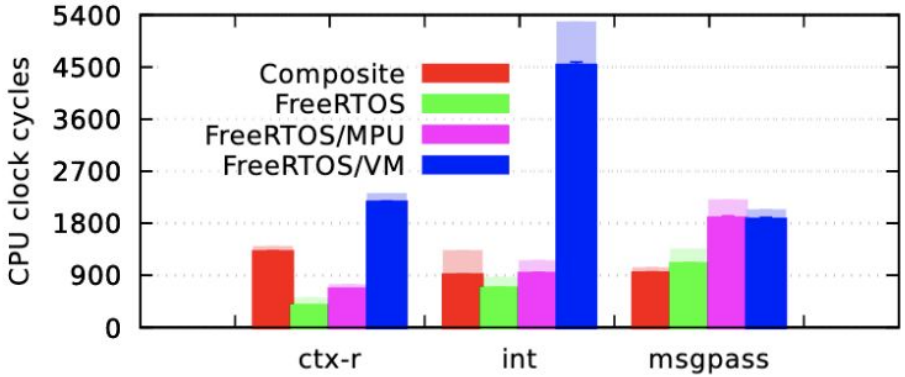


Evaluation

Memory overhead was far higher than the average overhead for power-of-two allocations.

Overhead increased reasonably given the added isolation of each VM.

But what about interrupts?



Critique

Can you multiplex an actuator?

Why is all of this necessary?

More specifically, why virtual machines?

Discussion (from Github)

Where / Why does memory fragmentation occur?

Where is this applicable in the real world?

Why is it always necessary to switch back to a scheduler thread?