

Securing Real-Time Microcontroller Systems through Customized Memory View Switching

Chung Hwan Kim^{*}, Taegyung Kim, Hongjun Choi, Zhongshu Gu⁺,
Byoungyoung Lee, Xiangyu Zhang, Dongyan Xu

^{*}
NEC Laboratories
America
Relentless passion for innovation

PURDUE
UNIVERSITY

⁺
IBM Research

Security of Real-time Microcontrollers

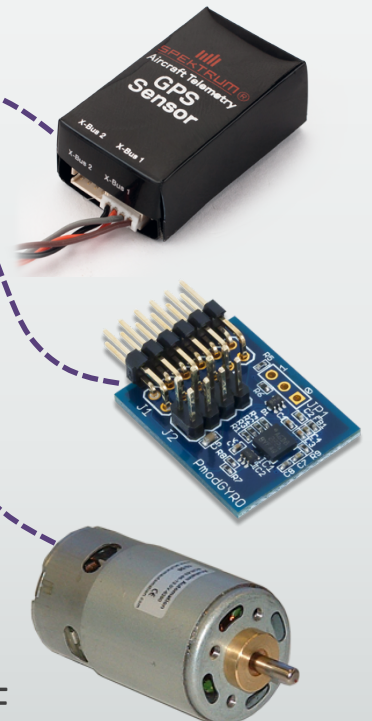
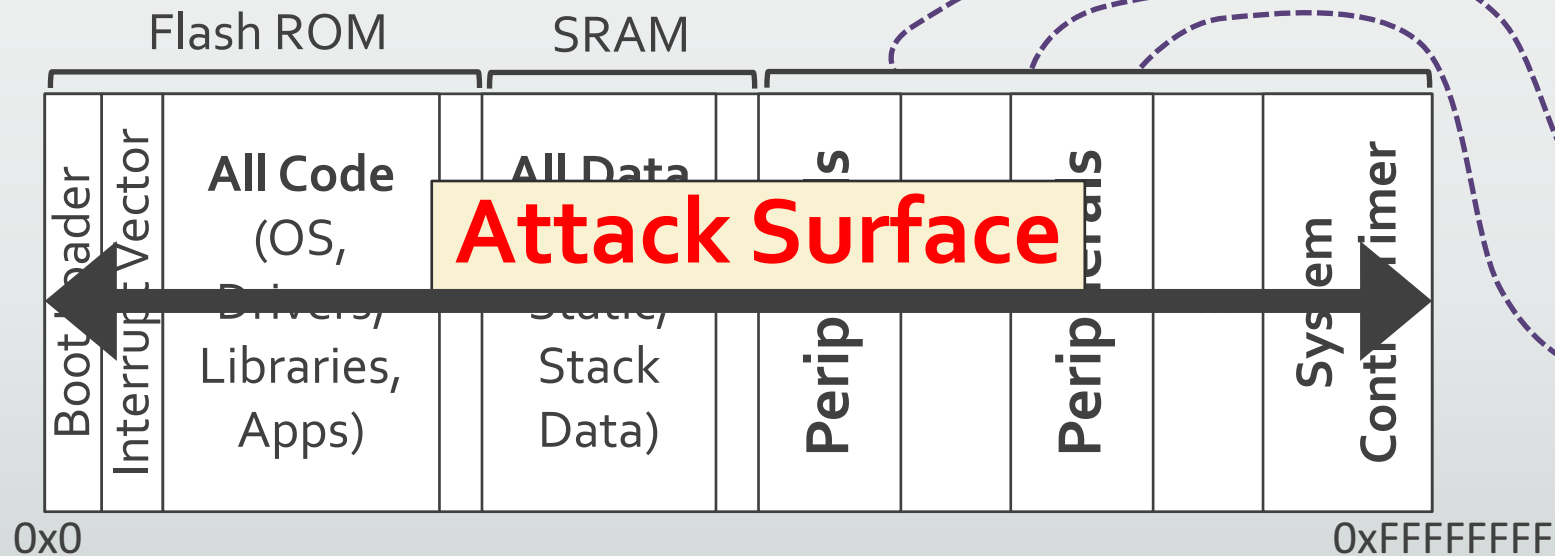
- Safety-critical embedded and cyber-physical systems



- Security is often overlooked as a trade off
- Demand both **real-time guarantee** and **security**

Missing Memory Protection of RT Microcontrollers

- **No process memory isolation**
 - No MMU, no virtual memory
 - Memory space shared by all processes
 - Memory-mapped I/O (MMIO)

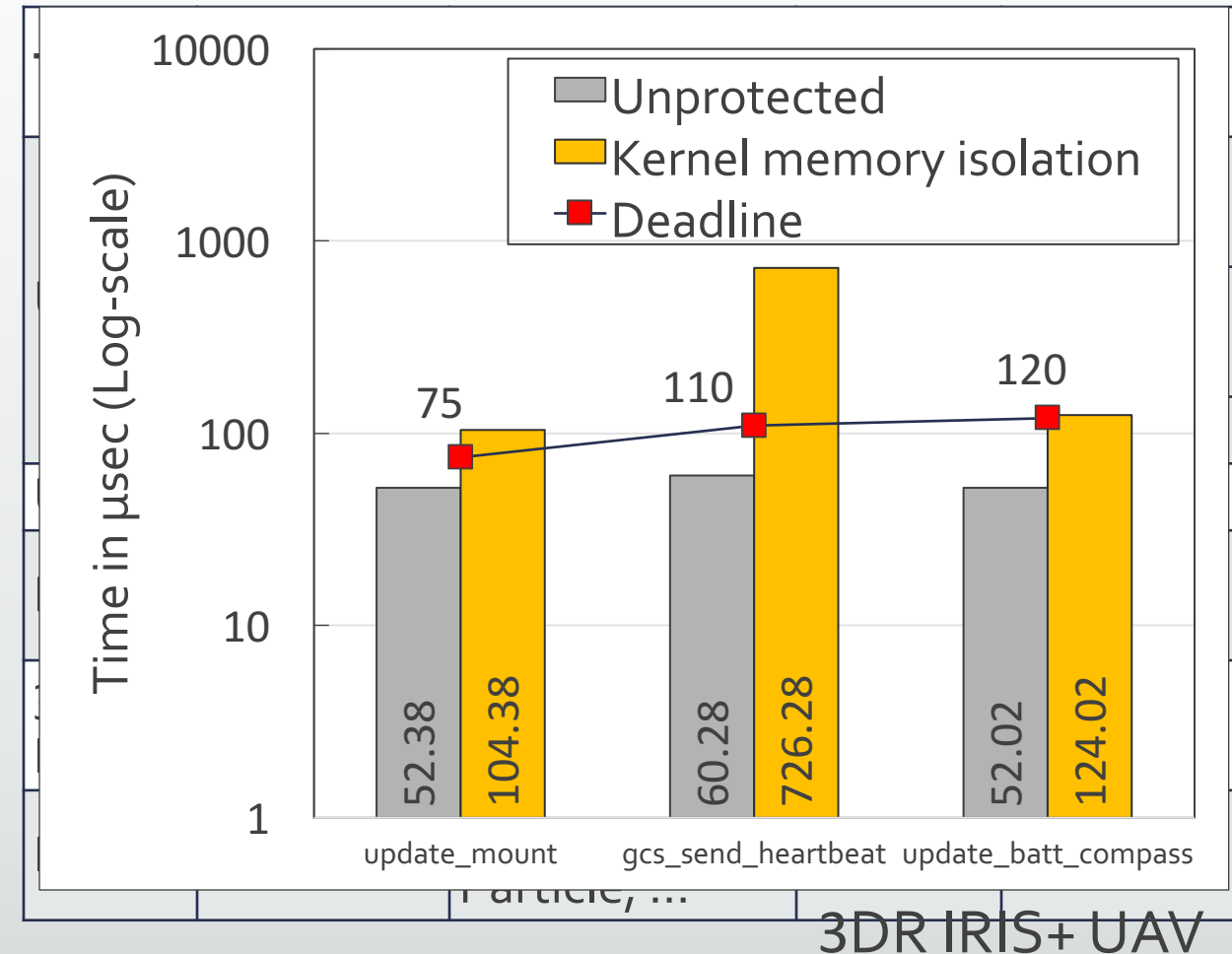


Control Parameter Attack



Missing Memory Protection of RT Microcontrollers

- **No kernel memory isolation**
 - Hardware and RTOS support
 - Privileged and unprivileged processor modes
 - Memory Protection Unit (MPU)
 - Many real-time microcontroller systems **do not employ** it
 - Verified with 67 commodity systems
 - Impact on **real-time constraints**
- **Frequent mode switching**

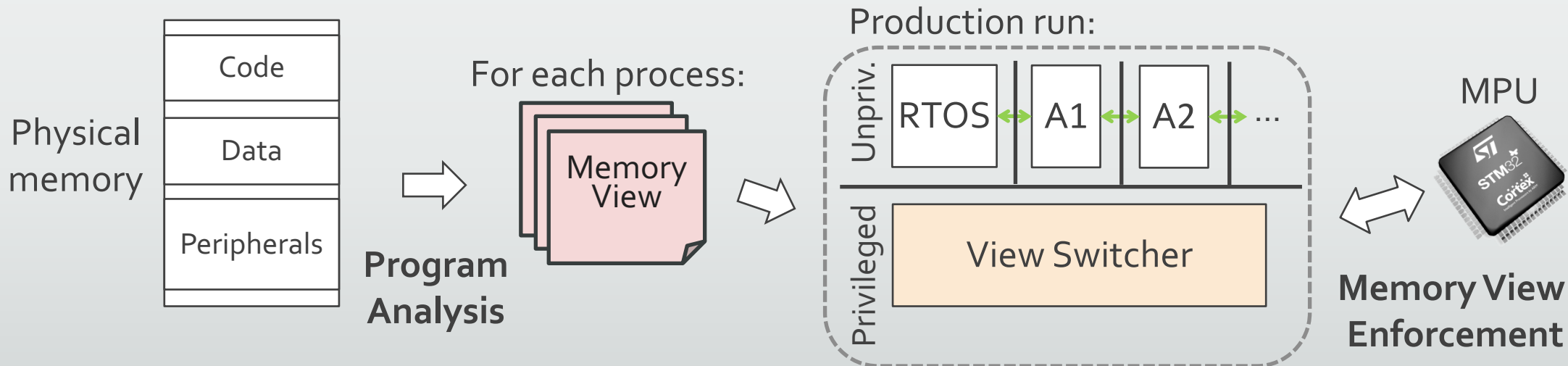


Hard Timer Attack



Minion: Customized Memory View Enforcement

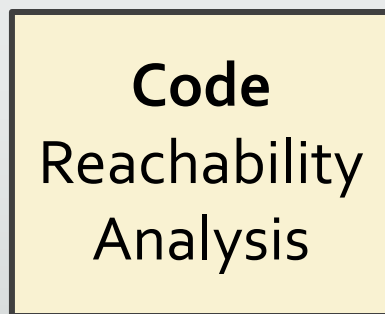
- Key ideas
 - Break physical memory space into per-process *memory views*
 - Use the memory views as **access control rules** during run-time
 - Execute RTOS and applications **in the same mode** (unprivileged)
 - Run a tiny *view switcher* in privileged mode to enforce views



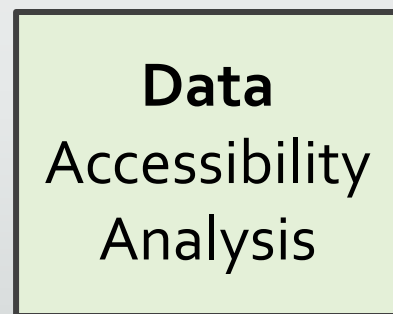
Memory View Tailoring

- **Memory view:** Memory required for a process to run correctly
- Find the physical memory regions **essential** for each process
- Static firmware analysis (LLVM IR)
- Code injection/reuse, data corruption, physical device abuse

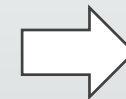
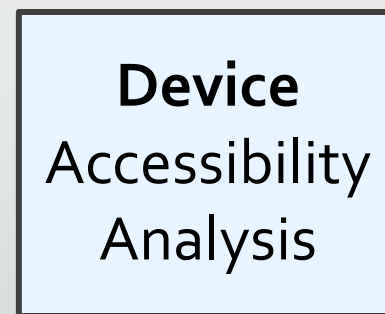
For each process:



+



+

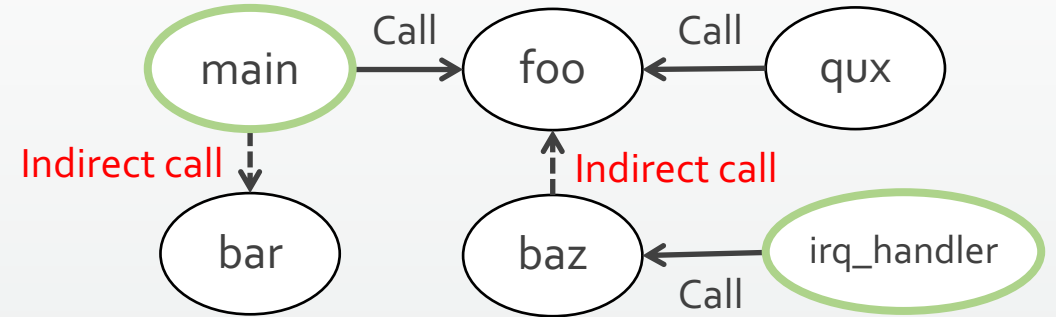


Access control rules:

#	Base	Size	rwx

Code Reachability Analysis

- Find all **reachable** functions from the entry functions
- **Entry functions**
 - Start function & interrupt handlers
 - Identified by analyzing a few RTOS functions
- Indirect calls?
 - **Inter-procedural points-to analysis**
- Build a list of executable memory regions for each process

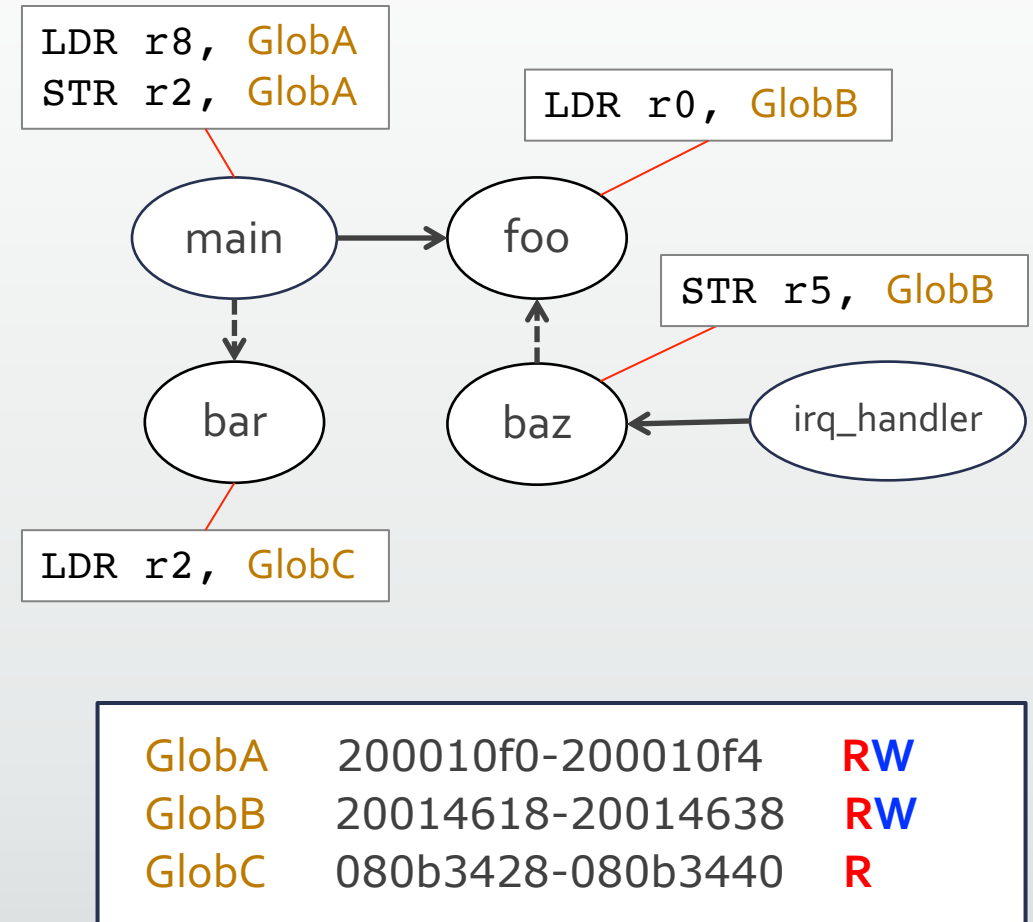


Value X	PointsTo: { bar }
Value Y	PointsTo: { foo }
Value Z	PointsTo: { bar }

main	08004970-08004988	X
irq_handler	08088050-080880cc	X
foo	0800498c-08004a7c	X
bar	08004a84-08004ad6	X
baz	08004ad8-08004b4c	X

Data Accessibility Analysis

- Global data
 - **Forward slicing** based on inter-procedural value flow graph
 - Build a list of global data for each process
- Stack and heap data
 - Memory pool **size profiling** with annotated memory allocator
 - **Per-process memory pool** allocation



Device Accessibility Analysis

- A few patterns cover most MMIO operations
- MMIO addresses are **embedded** in the firmware
- Case 1
- Case 2

```
#define DEVICE_X 0x50000804

void dev_reset(struct dev *priv)
{
    uint32_t val;
    val = (1 << 2) | (1 << 4);
    *(uint32_t *)DEVICE_X = val;
    ...
}
```

From NuttX RTOS (simplified)

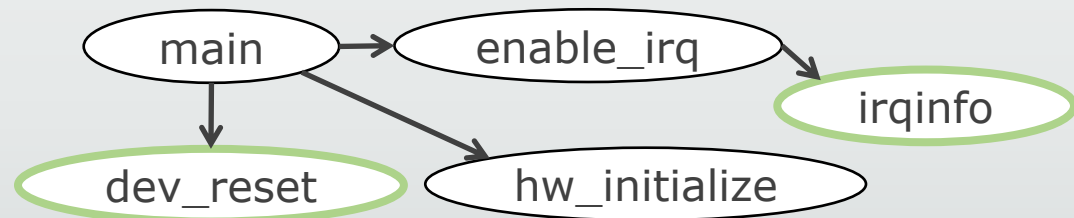
```
#define IRQ_A 1
#define IRQ_B 2
#define NVIC_A 0xe000e100
#define NVIC_B 0xe000e104

int irqinfo (int irq,
             uint32_t *addr)
{
    if (irq == IRQ_A) {
        *addr = NVIC_A;
    } else if (irq == IRQ_B) {
        *addr = NVIC_B;
    }
    ...
}
```

```
int enable_irq (int irq)
{
    uint32_t addr, val;
    if (irqinfo(irq, &addr) == OK) {
        val = *(uint32_t *)addr;
        val |= (1 << 1);
        *(uint32_t *)addr = val;
    }
}
```

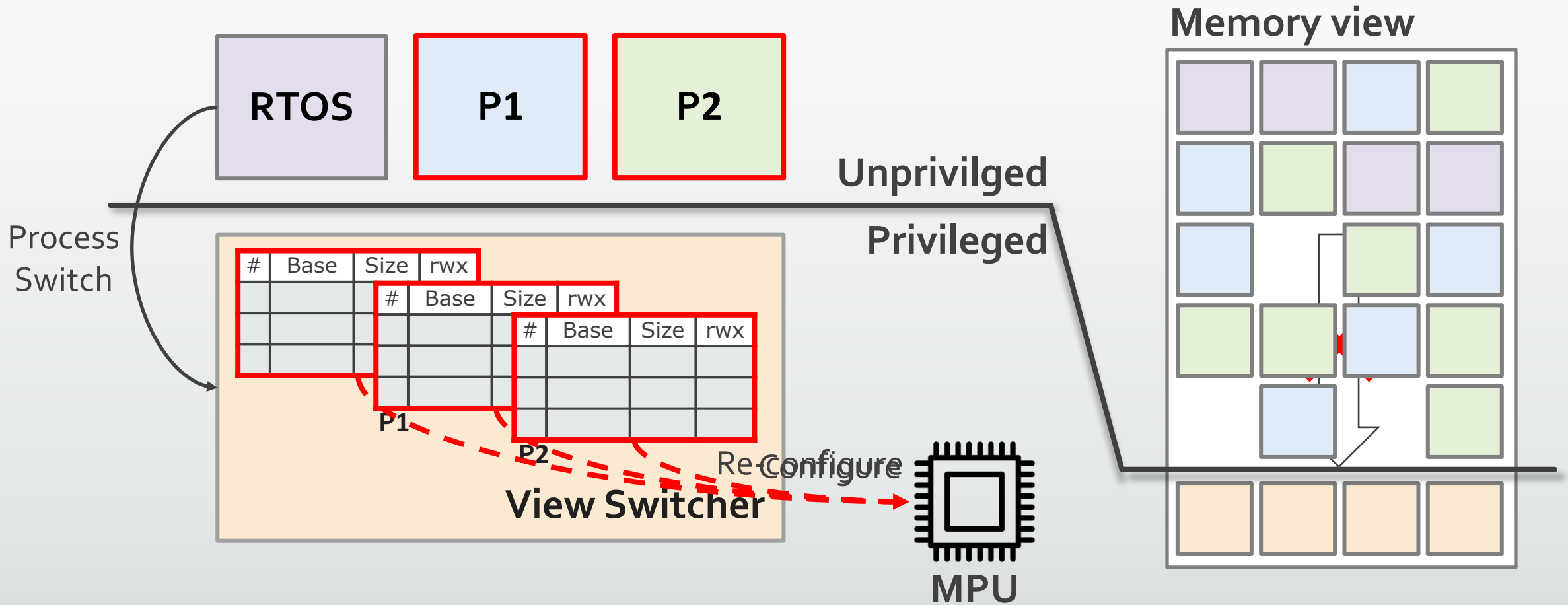
Device Accessibility Analysis

- Find load and store instructions with an MMIO address
- **Backward slicing** on inter-procedural value flow graph
- Build a list of **peripheral-mapped** memory regions for each process



DEVICE_X	50000804-50000808	W
NVIC_A	e000e100-e000e104	RW
NVIC_B	e000e104-e000e108	RW

Run-time Memory View Enforcement



Evaluation with Attack Cases

- Tested on a commodity UAV



3DR IRIS+

- Found **4 new vulnerabilities** in the firmware (confirmed and fixed)
- **76%** memory space reduction

- 8 realistic attack cases

Name	Attack surface	Result
Process termination	RTOS function	✓
Control parameter attack	Control parameter	✓
RC disturbance	RC configuration	✓
Servo operation	Driver function	✓
Soft timer attack	Hardware timer	✓
Hard timer attack	Hardware timer	✓
Memory remapping	Flash patch unit	✓
Interrupt vector overriding	Interrupt vector	✓

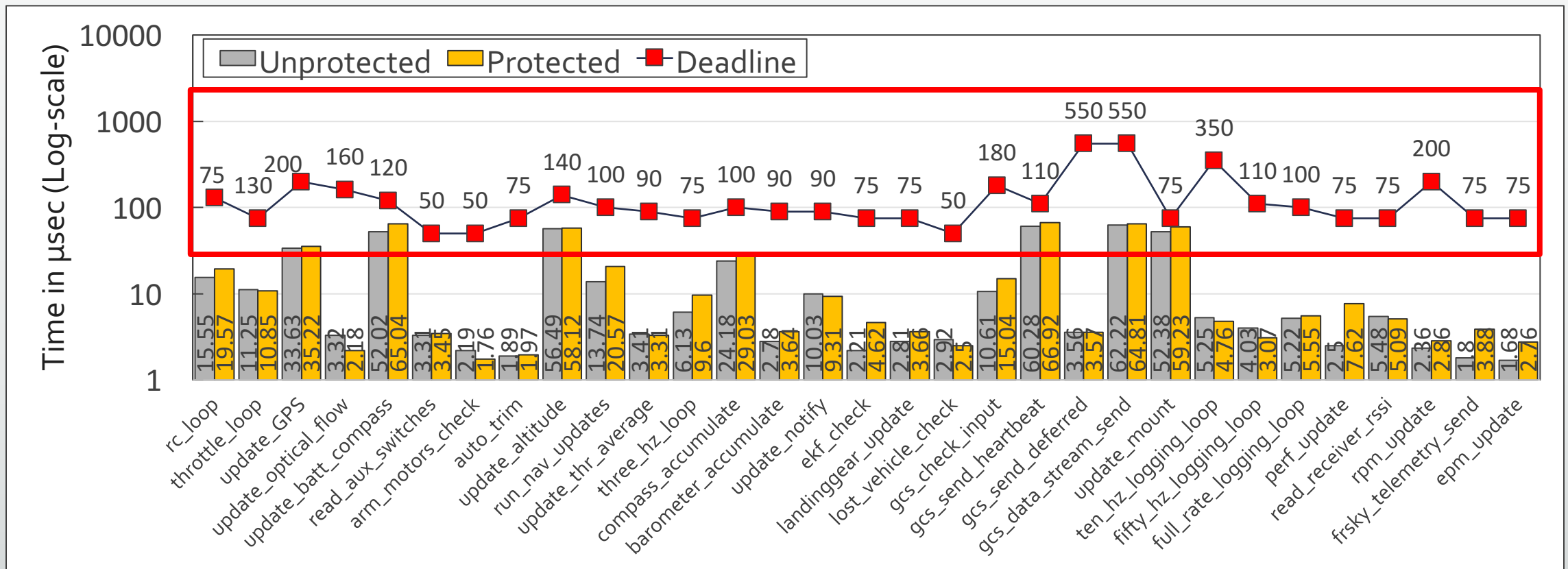
- All 8 attack cases blocked
- Zero violation of real-time constraints

Attack Under Minion's Protection



Performance Impact

- 31 real-time tasks with deadlines: 2% overhead
- **All deadline constraints satisfied**



Conclusion

- Memory protection in RT microcontrollers
- **Minion:** New architecture to bring memory isolation to RT microcontroller systems
- Significant memory space reduction with maintained RT responsiveness
- Attack cases and vulnerability discovery

This work is sponsored by ONR.

Thank you!

Questions?



<https://github.com/chungkim/minion>

Chung Hwan Kim <chungkim@nec-labs.com>