Safer Sloth: Efficient Hardware-Tailored Memory Protection

Daniel Danner, Rainer Müller,
Wolfgang Schröder-Preikschat, Wanja Hofer, Daniel Lohmann

April 15, 2014

Laziness Is Beautiful: Let the Hardware Do the Work!

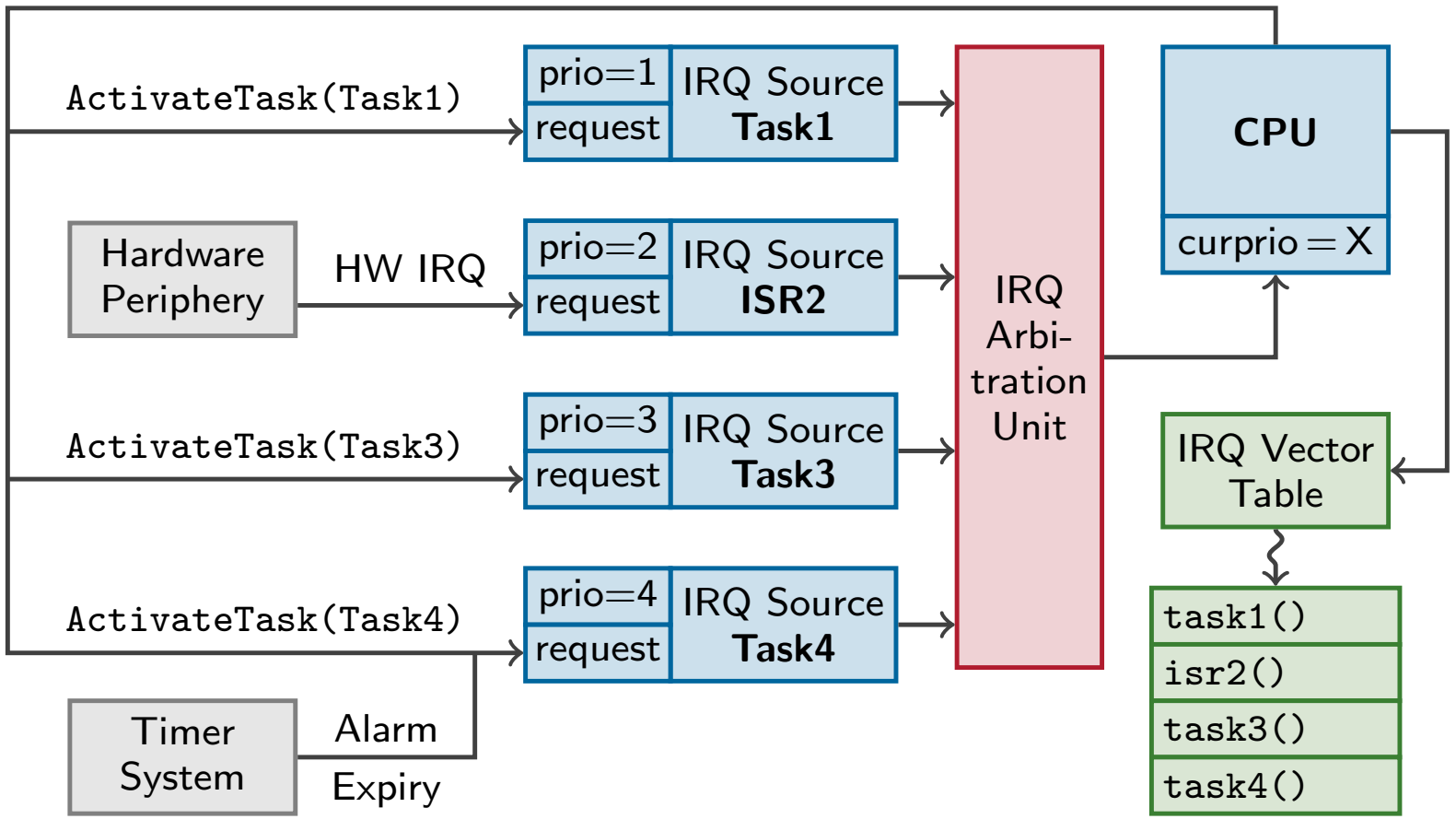
SLOTH kernels use hardware for OS purposes, and

- are concise (200–500 LoC)
- are small (300–900 bytes)
- are fast (latency speed-up 2x to 170x)
- implement industry standards (OSEK, OSEKtime, AUTOSAR OS)

SLOTH Recap

Main Idea

Threads are interrupt handlers, synchronous thread activation is IRQ
⇒ Interrupt subsystem does scheduling and dispatching work



Safer Sloth: Motivation and Goals

Threads as Interrupts?

- But what about safety?

Motivation for Safer Sloth

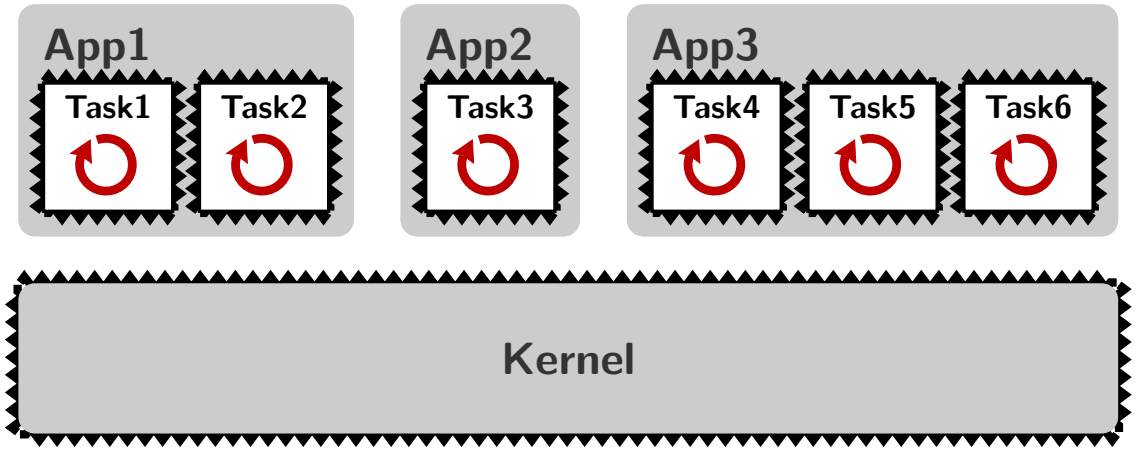
- Sloth has been criticized for lack of isolation
  - ⇒ Tasks are executed in IRQ handler context
  - ⇒ Application code has supervisor privileges

Goals

- Effectively isolate kernel and application
- Maintain design principles of Sloth

Memory Protection in Embedded Systems

- *Safety, but not security*
- *Protect the data, but not the code*
- Safety model based on AUTOSAR OS
- MPU-based isolation



- **Vertically:** Protect kernel state and MPU configuration
- **Horizontally:** Isolate applications or even tasks from each other

Hardware Model for Safer Sloth

- Exploit as much knowledge about target hardware as possible
- Tailor kernel to fit both the platform and the application
- Taking into account:
  - Extent and layout of MPU configuration
  - Method for re-programming the MPU
  - Available hardware privilege levels
  - Is MPU active in all levels?
  - Degree of safety required by the application

Protection Modes in Safer Sloth

Unsafe

- The original Sloth OS, without isolation

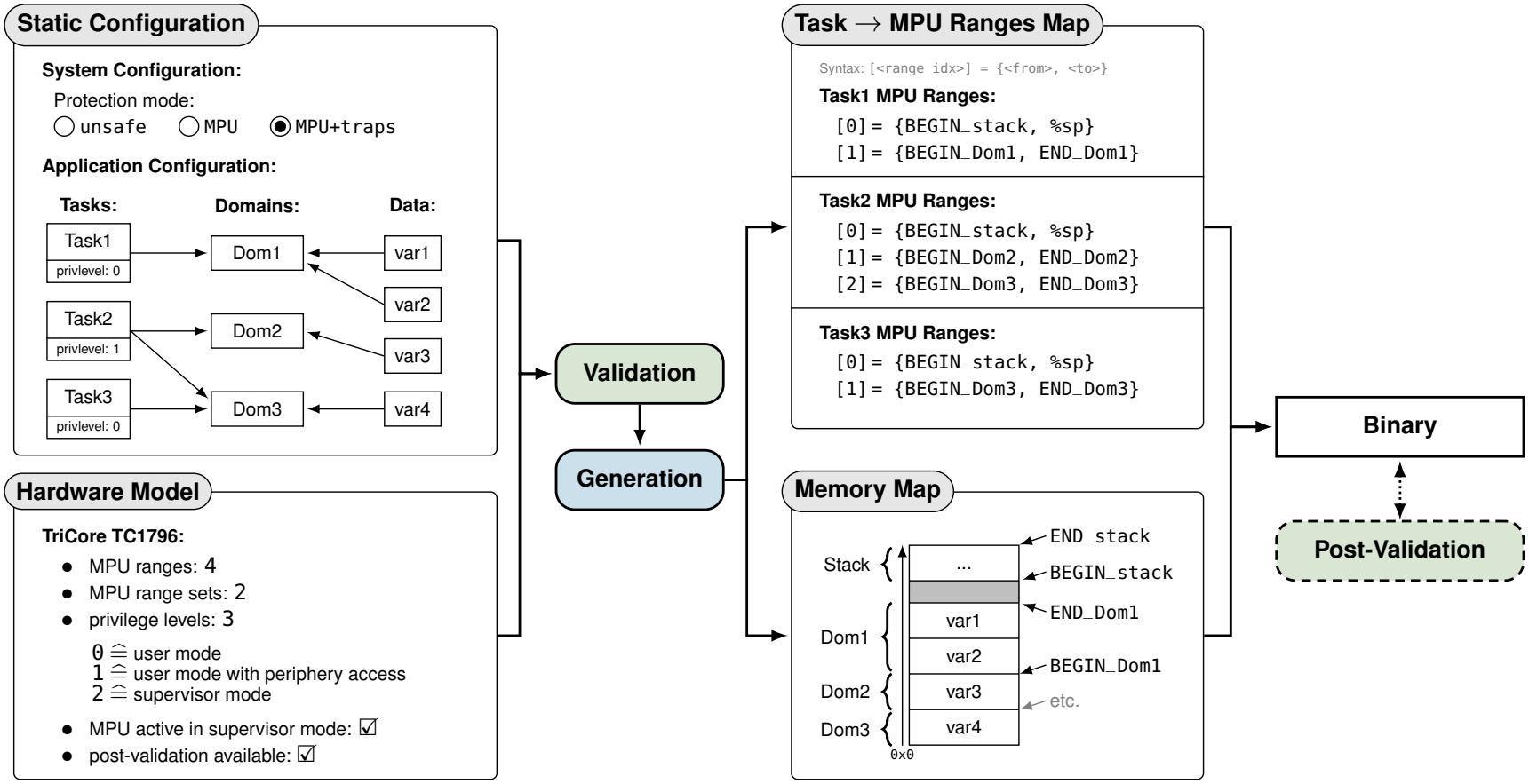
MPU

- MPU active, but tasks execute with supervisor privileges
- Vertical isolation ensured constructively in post-validation

MPU+traps

- Vertical isolation ensured by hardware privilege levels
- System services acquire kernel privileges via syscall mechanism

Safer Sloth: Architecture



MPU mode in Safer Sloth

unsafe Mode:

```
Task1:
...
; inlined call to
; GetResource(Res1):
prio = getCurPrio();
pushResourceStack(prio);
if (Res1 > prio) {
    setCurPrio(Res1);
}
...
```

MPU Mode:

```
Task1:
...
; disable MPU
mfcr %d15,$psw
insert %d15,%d15,0,12,1
mtcr $psw,%d15
prio = getCurPrio();
pushResourceStack(prio);
if (Res1 > prio) {
    setCurPrio(Res1);
}
; enable MPU
mfcr %d15,$psw
insert %d15,%d15,15,12,1
mtcr $psw,%d15
...
```

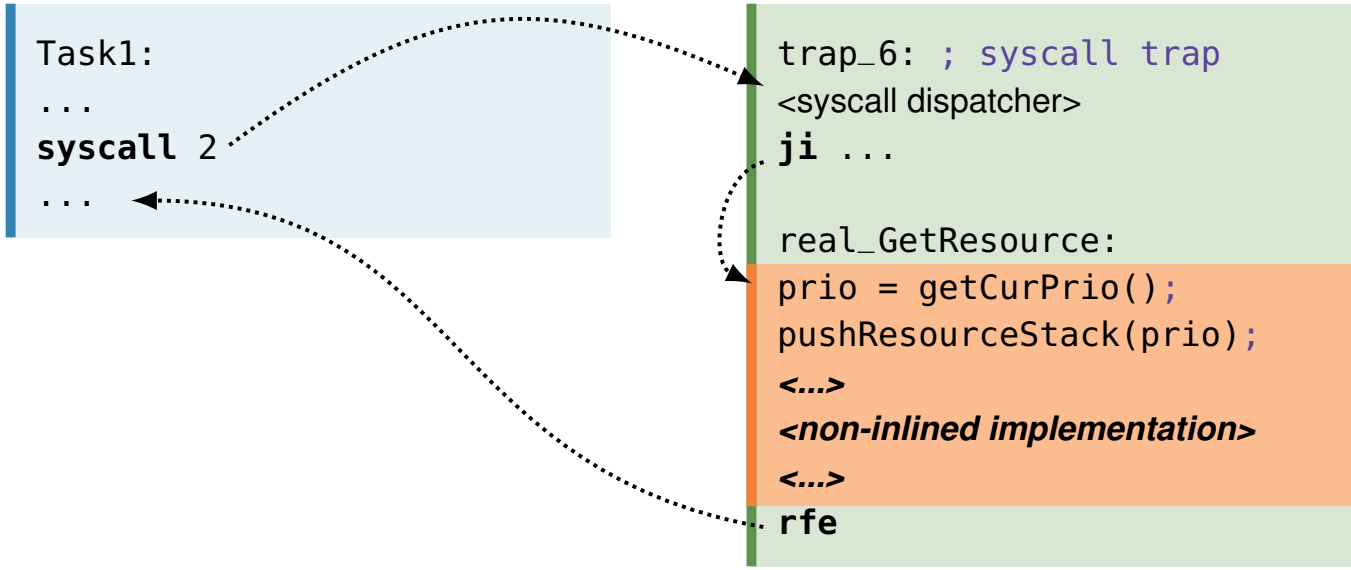
User mode

Supervisor mode

System service implementation

MPU+traps mode in Safer Sloth

MPU+traps Mode:



User mode

Supervisor mode

System service implementation

The Problem with Traps

- Sloth gains a lot through compiler optimizations.
  System services are short, parameters are mostly static, compilation done as a single unit.

⇒ Inlining of system service calls

⇒ Removal of dead code

⇒ Constant propagation

Traditional traps prohibit such optimizations

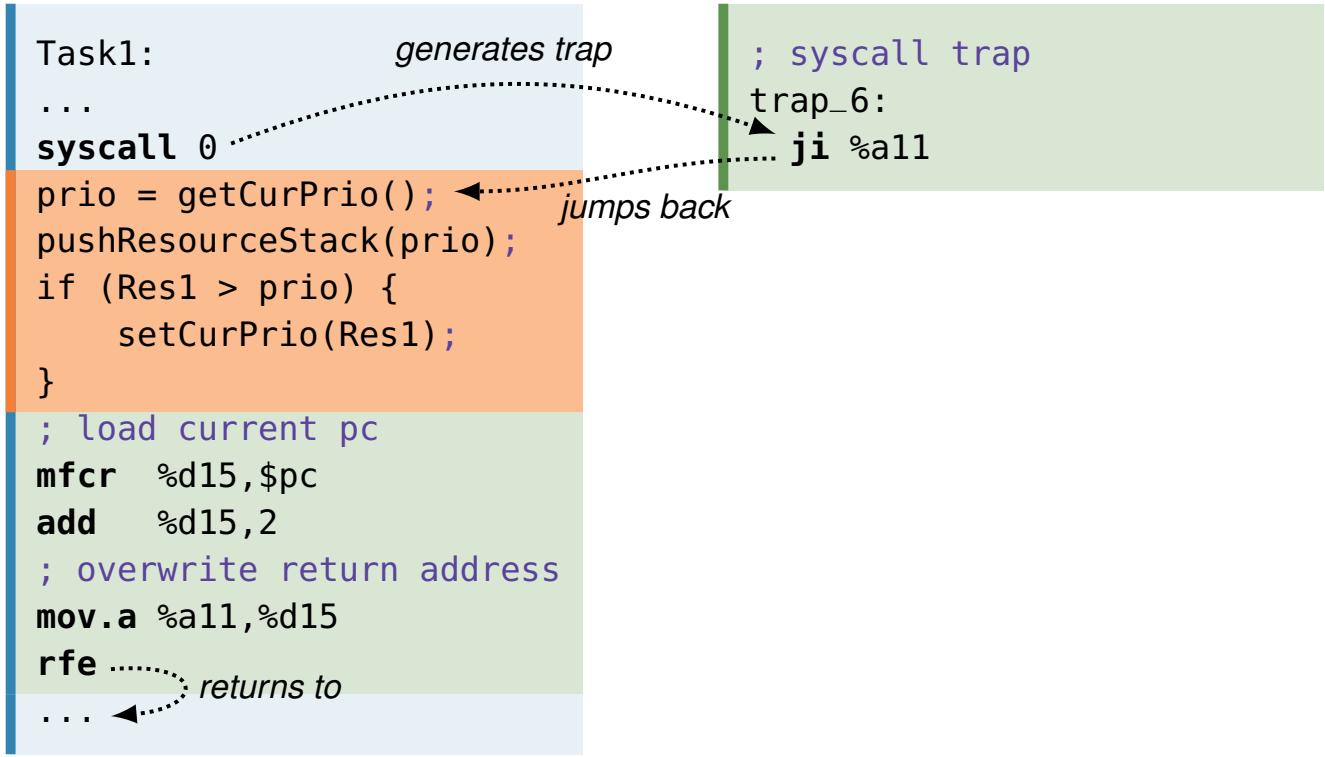
System services must be standalone functions, jumped to via a syscall dispatcher

Solution: Combination of `MPU` and `MPU+traps` mode

⇒ *Inline traps* as 4th protection mode (`MPU+itraps`)

Inline Traps in Safer Sloth

MPU+itraps Mode:

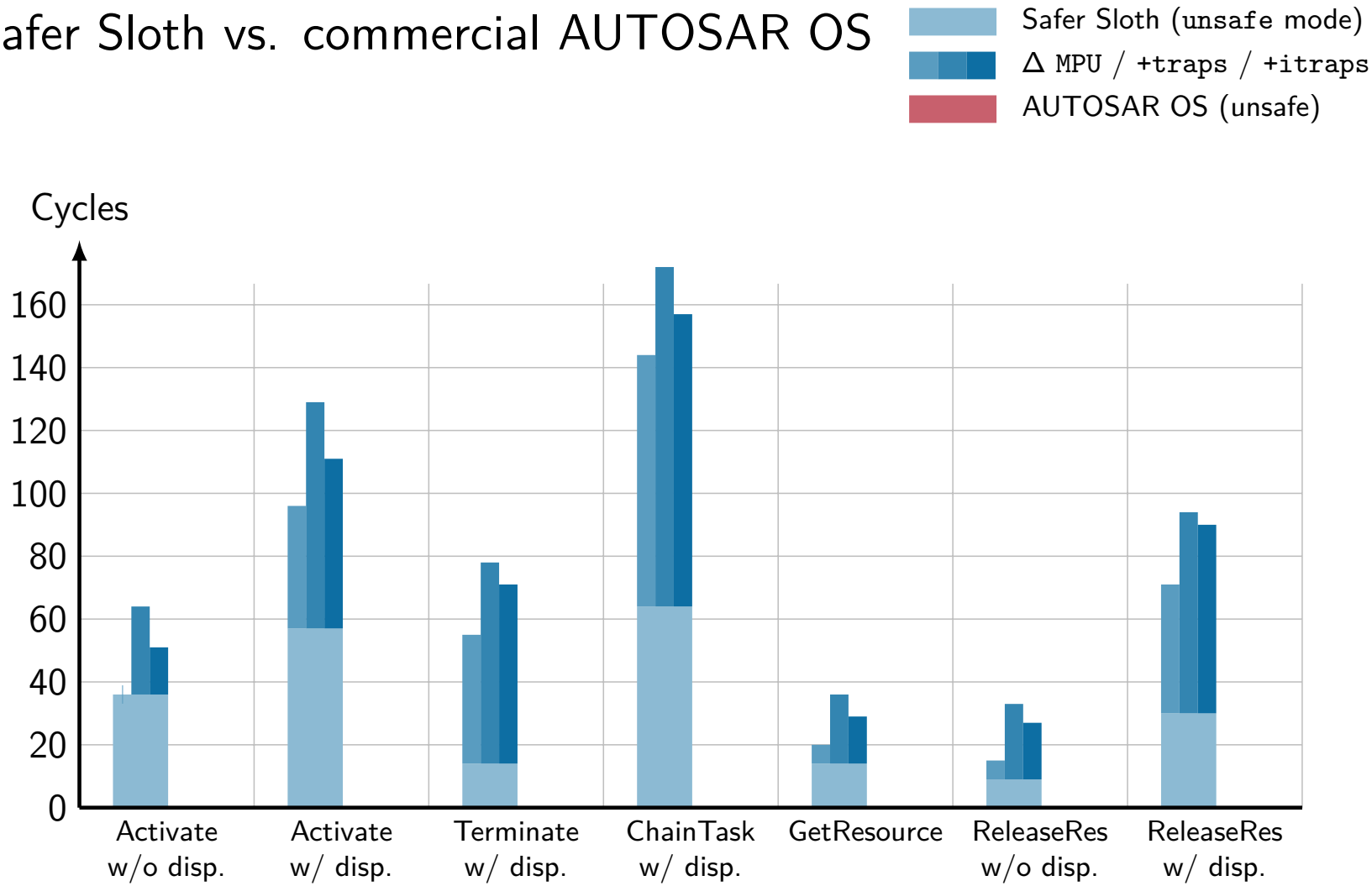


Evaluation Setup

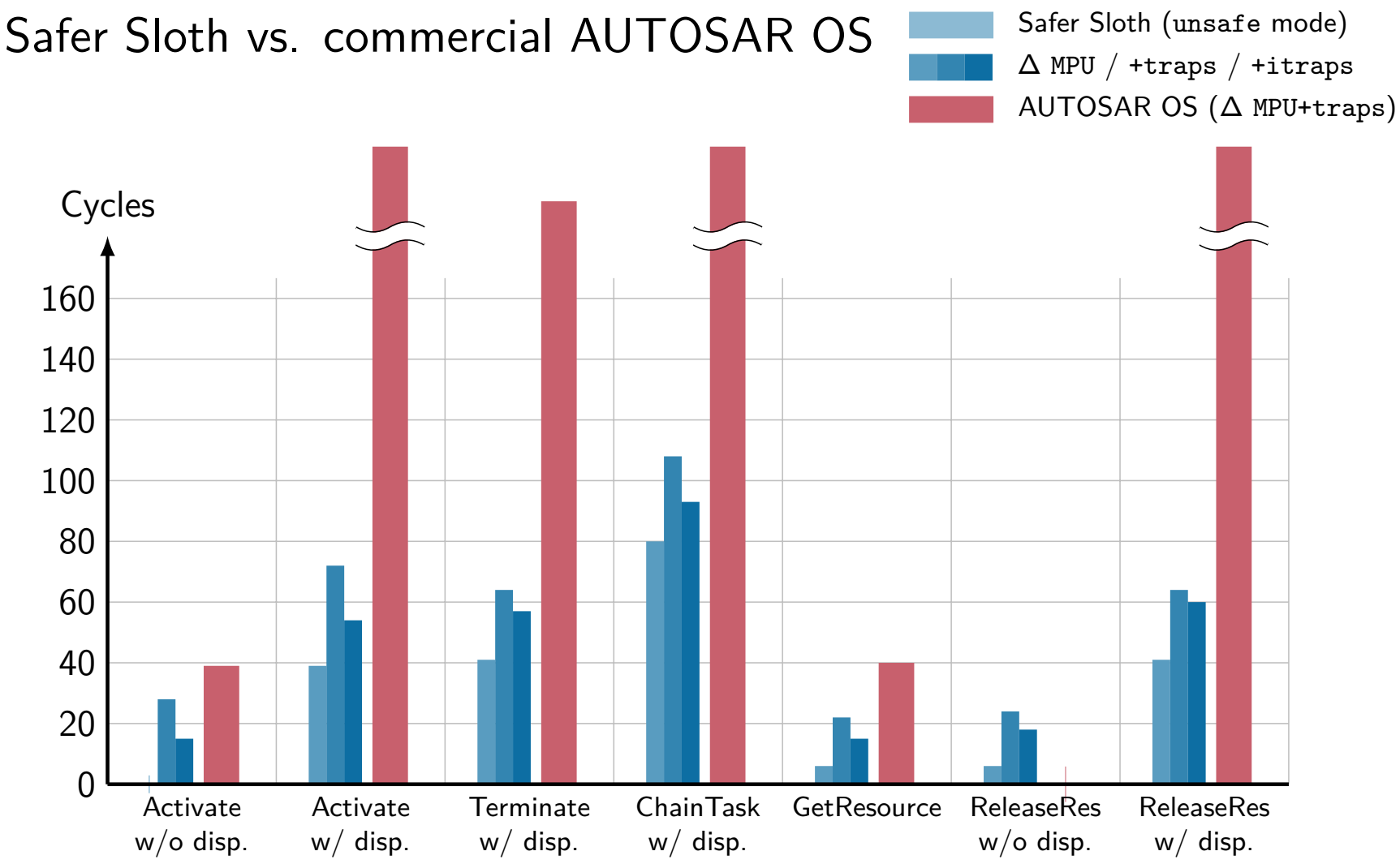
- **Evaluation platform:** Infineon TriCore TC1796
  - 32-bit RISC μ-Controller, clocked at 50 MHz
    - widely used in the automotive industry (BMW, Audi, ...)
    - IRQ system with 256 priority levels and 181 IRQ sources
  - Safety features:
    - 3 privilege levels
    - MPU with 2 protections sets, 4 memory ranges each
- **Comparison against:** Commercial AUTOSAR OS
  - Offers two modes of protection, equivalent to
    - `unsafe` mode
    - `MPU+traps` mode
- **Approach:**
  - Microbenchmarks of system service overheads including possible transitions

Evaluation Results: Total Overheads

Safer Sloth vs. commercial AUTOSAR OS



Evaluation Results: Additional Overheads



Conclusions

Safer Sloth...

- provides effective memory protection
- offers tailorability to both hardware *and* application
- maintains advantages of Sloth:
  - no rate-monotonic priority inversions
  - small footprint
  - minimal and constant latencies

$\Rightarrow$ excellent real-time characteristics