Towards Availability and Real-Time Guarantees for Protected Module Architectures

Jo Van Bulck, Job Noorman, Jan Tobias Mühlberg and Frank Piessens

March 14, 2016

KU LEUVEN
DISTRINET RESEARCH GROUP

iMinds
CONNECT.INNOVATE.CREATE

“Embedded-systems security is, for lack of a better word, a mess.”

– John Viega & Hugh Thompson

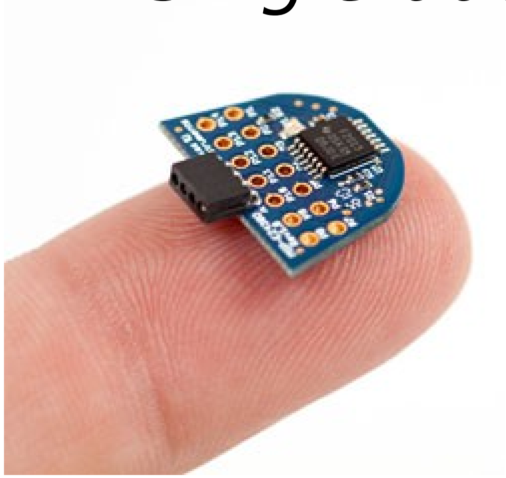
VIEGA John, THOMPSON Hugh, *The state of embedded-device security (spoiler alert: It's bad)*, IEEE Security & Privacy (10.5), September 2012, pp. 68-70.

Motivation: Embedded Systems Security

Embedded

- Low-cost, low-power
- Mixed-criticality context

=> *Single-address-space*



Conventional

- Resource-intensive
- General-purpose

=> MMU/MPU

=> Kernel mode

<> *TCB reduction*

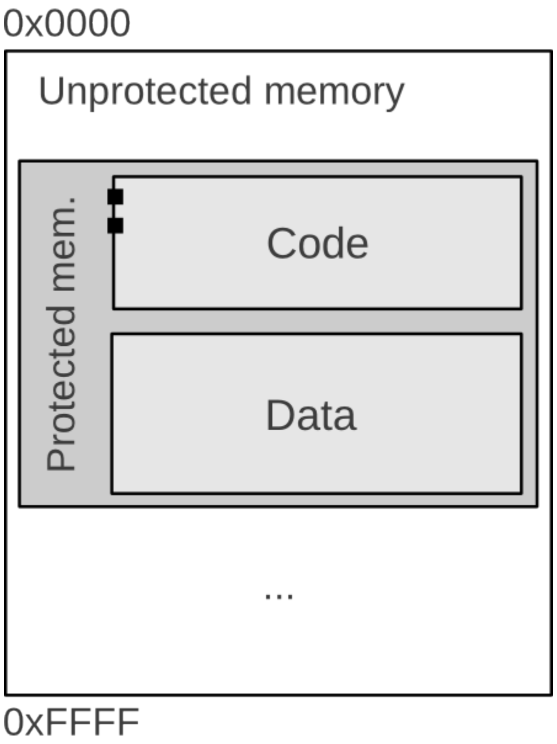
KOEBERL, Patrick, et al. *Trustlite: A security architecture for tiny embedded devices.* EuroSys. ACM (2014).
MCKEEN, Frank, et al. *Innovative instructions and software model for isolated execution.* HASP@ ISCA. 2013.

Roadmap

1. Protected Module Architectures
2. Research Objectives
3. Interruptible Isolated Execution
4. Secure Multithreading
5. Conclusion

KU LEUVEN

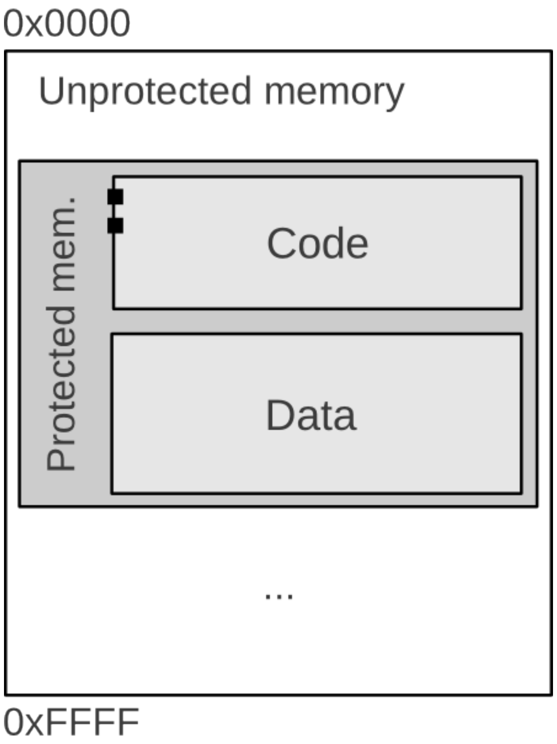
Protected Module Architectures



- **Isolated execution** areas in a single-address-space

STRACKX Raoul et al., *Protected Software Module Architectures*, ISSE 2013 Securing Electronic Business Processes, Springer Fachmedien Wiesbaden, 2013, pp. 241-251.

Protected Module Architectures

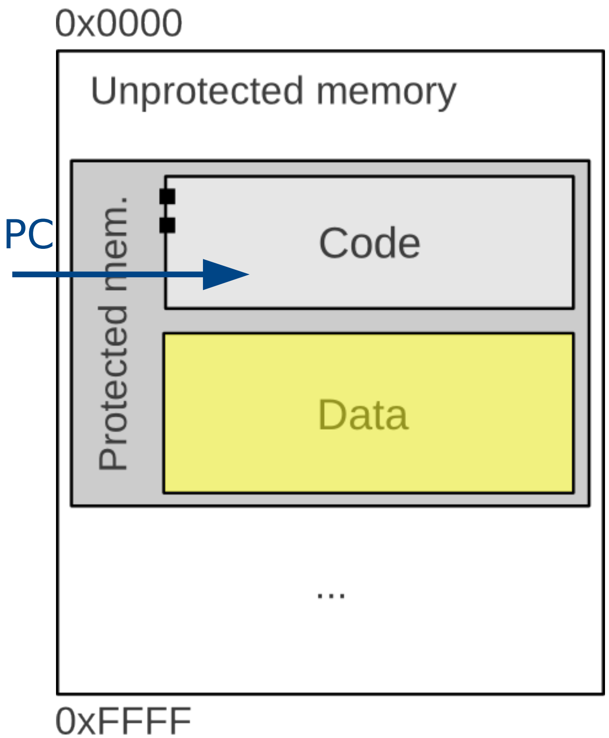


- **Isolated execution** areas in a single-address-space
- **Program counter** based access control mechanism

From \ to	Protected Entry	Protected Code	Protected Data	Unprotected
Protected	r-x	r-x	rw-	rwx
Unprotected / other SPM	r-x	r--	---	rwx

STRACKX Raoul et al., *Protected Software Module Architectures*, ISSE 2013 Securing Electronic Business Processes, Springer Fachmedien Wiesbaden, 2013, pp. 241-251.

Protected Module Architectures



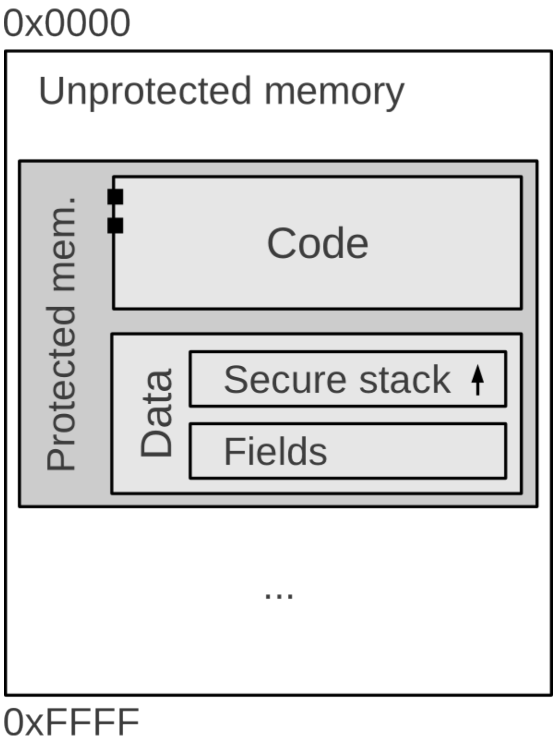
- **Isolated execution** areas in a single-address-space
- **Program counter** based access control mechanism

From \ to	Protected Entry	Protected Code	Protected Data	Unprotected
Protected	r-x	r-x	rw-	rwx
Unprotected / other SPM	r-x	r--	---	rwx

STRACKX Raoul et al., *Protected Software Module Architectures*, ISSE 2013 Securing Electronic Business Processes, Springer Fachmedien Wiesbaden, 2013, pp. 241-251.

KU LEUVEN

Protected Module Architectures



- **Isolated execution** areas in a single-address-space
- **Program counter** based access control mechanism
- Secure fully abstract **compilation**

From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwx
Unprotected / other SPM	r-x	r--	---	rwx

STRACKX Raoul et al., *Protected Software Module Architectures*, ISSE 2013 Securing Electronic Business Processes, Springer Fachmedien Wiesbaden, 2013, pp. 241-251.

Sancus PMA

- **Zero-software TCB**

  → extended openMSP430 instruction set

NOORMAN Job et al., *Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base,* Proceedings of the 22nd USENIX conference on Security symposium, 2013, pp. 479-494.

Sancus PMA

- **Zero-software TCB**

  → extended openMSP430 instruction set

- SM == unit of **isolation** + **authentication**

  → *remote attestation / secure linking*

  → hardware cryptographic key and ID per SM

NOORMAN Job et al., *Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base,* Proceedings of the 22nd USENIX conference on Security symposium, 2013, pp. 479-494.

KU LEUVEN

Sancus PMA

- **Zero-software TCB**

  → extended openMSP430 instruction set

- SM == unit of **isolation** + **authentication**

  → *remote attestation / secure linking*

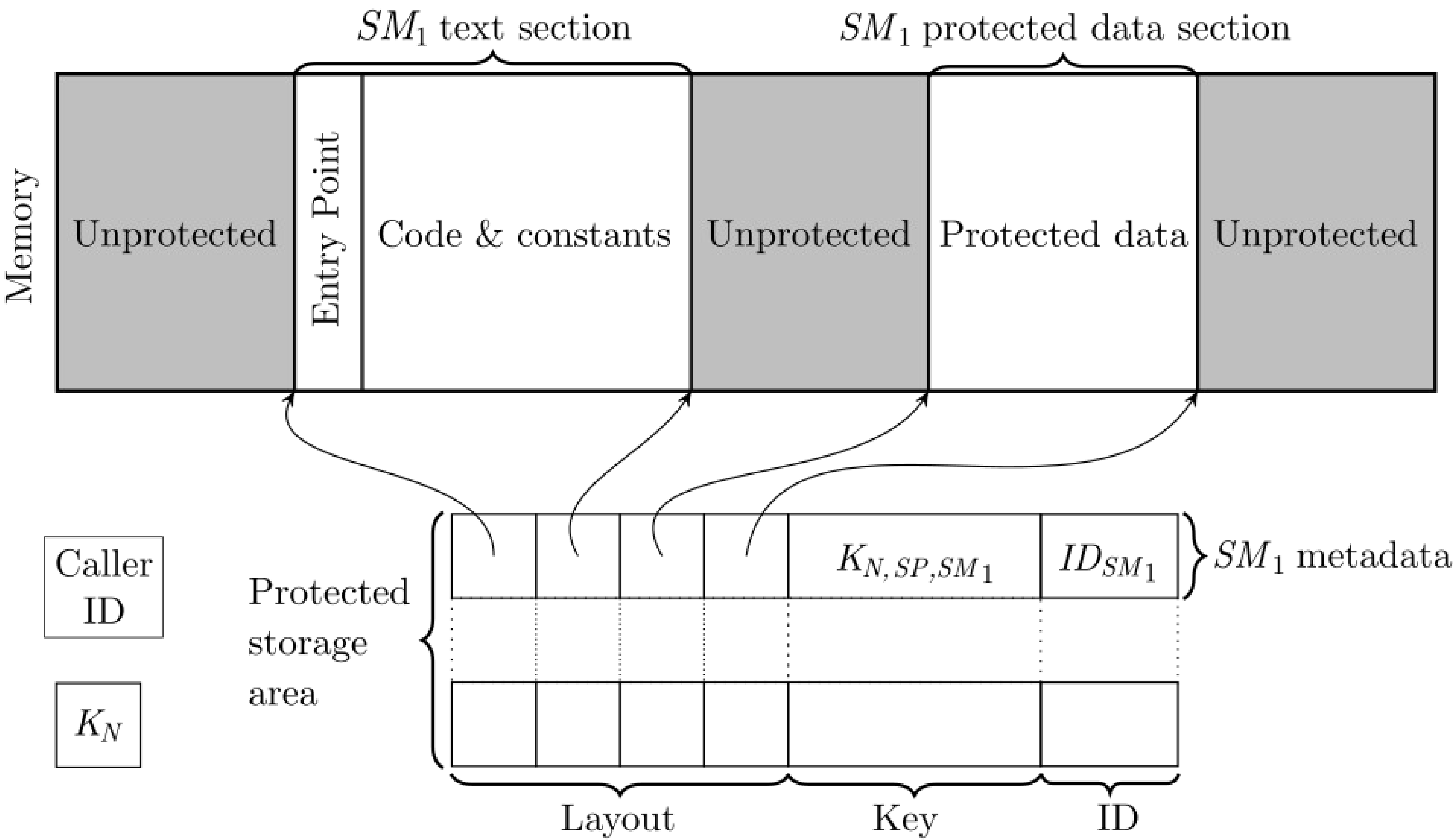
  → hardware cryptographic key and ID per SM

- Dedicated secure C **compiler**

  → generates `sm_entry/exit` asm stubs

NOORMAN Job et al., *Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base,* Proceedings of the 22nd USENIX conference on Security symposium, 2013, pp. 479-494.

KU LEUVEN


SM_1 text section
SM_1 protected data section
Memory
Unprotected
Entry Point
Code & constants
Unprotected
Protected data
Unprotected
Caller ID
K_N
Protected storage area
K_{N,SP,SM_1}
ID_{SM_1}
SM_1 metadata
Layout
Key
ID

Contents

KU LEUVEN

Research Objectives

PMAs assume the presence of an attacker:

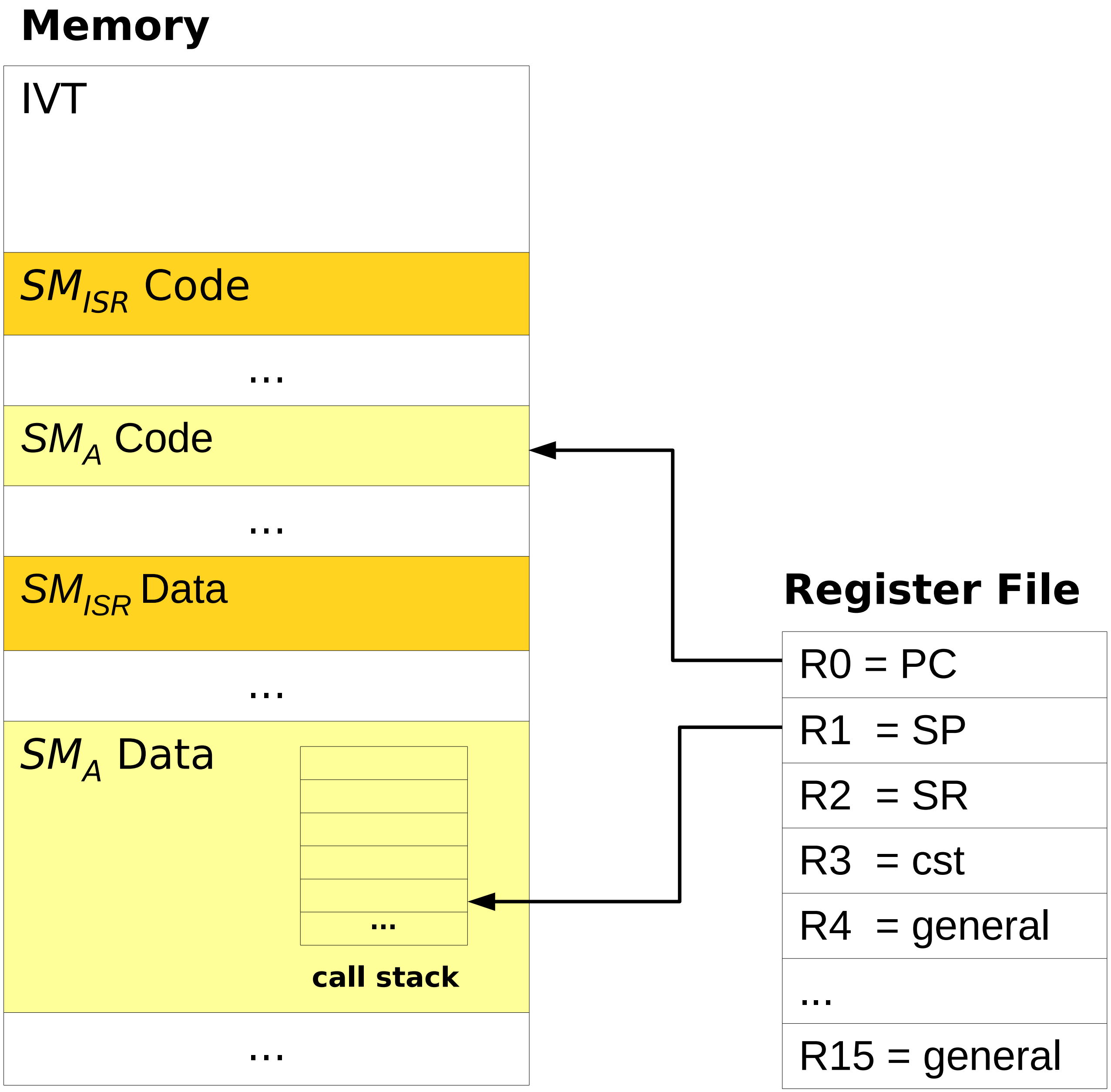
☺ HW-enforced SM **confidentiality / integrity**

☹ no **availability guarantees**

=> concurrent execution of *isolated threads* via an *unprivileged* preemptive scheduler

Contents

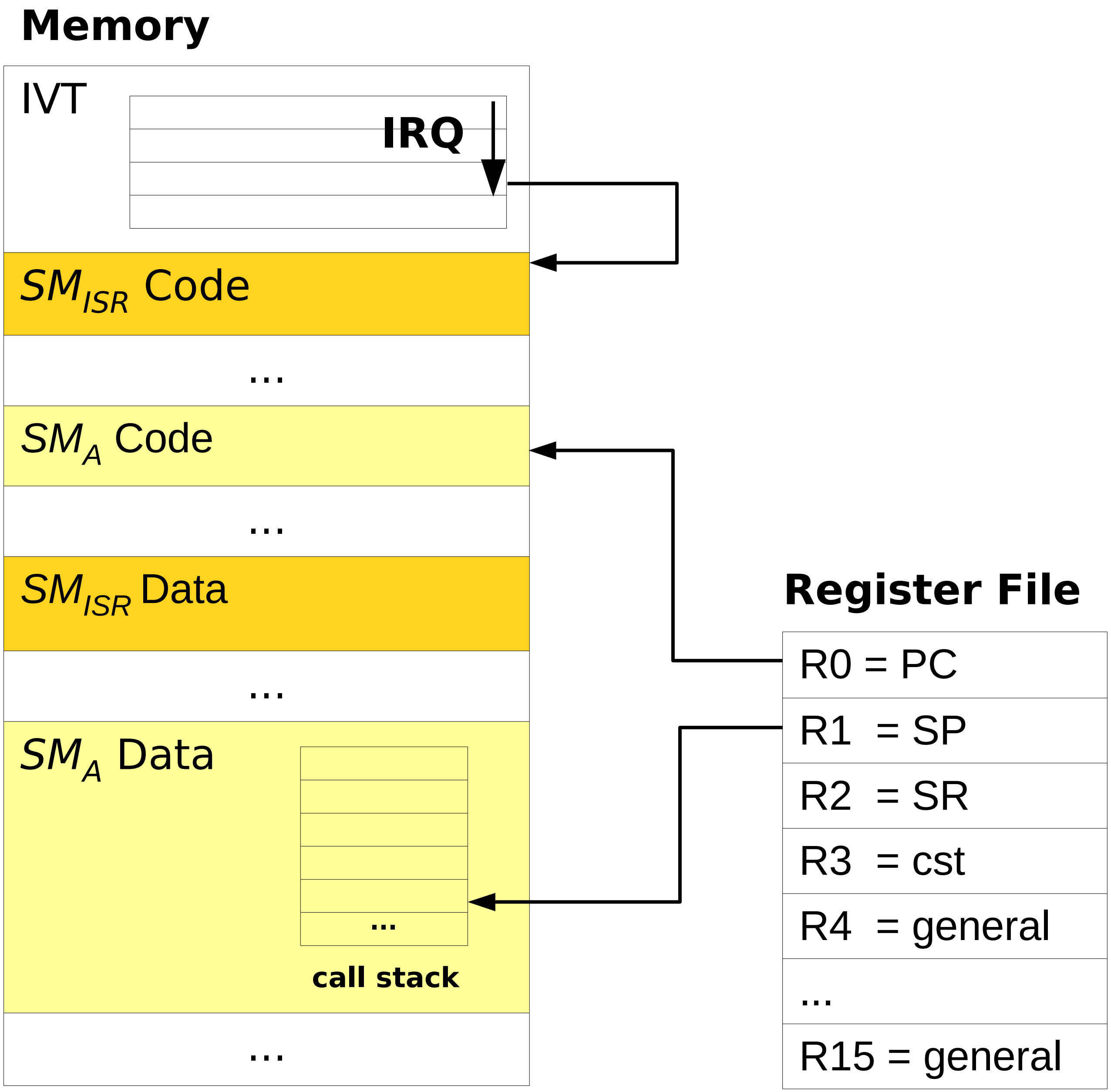
Interruptible and Reentrant SMs



Current SM = SM_A

Previous SM = x

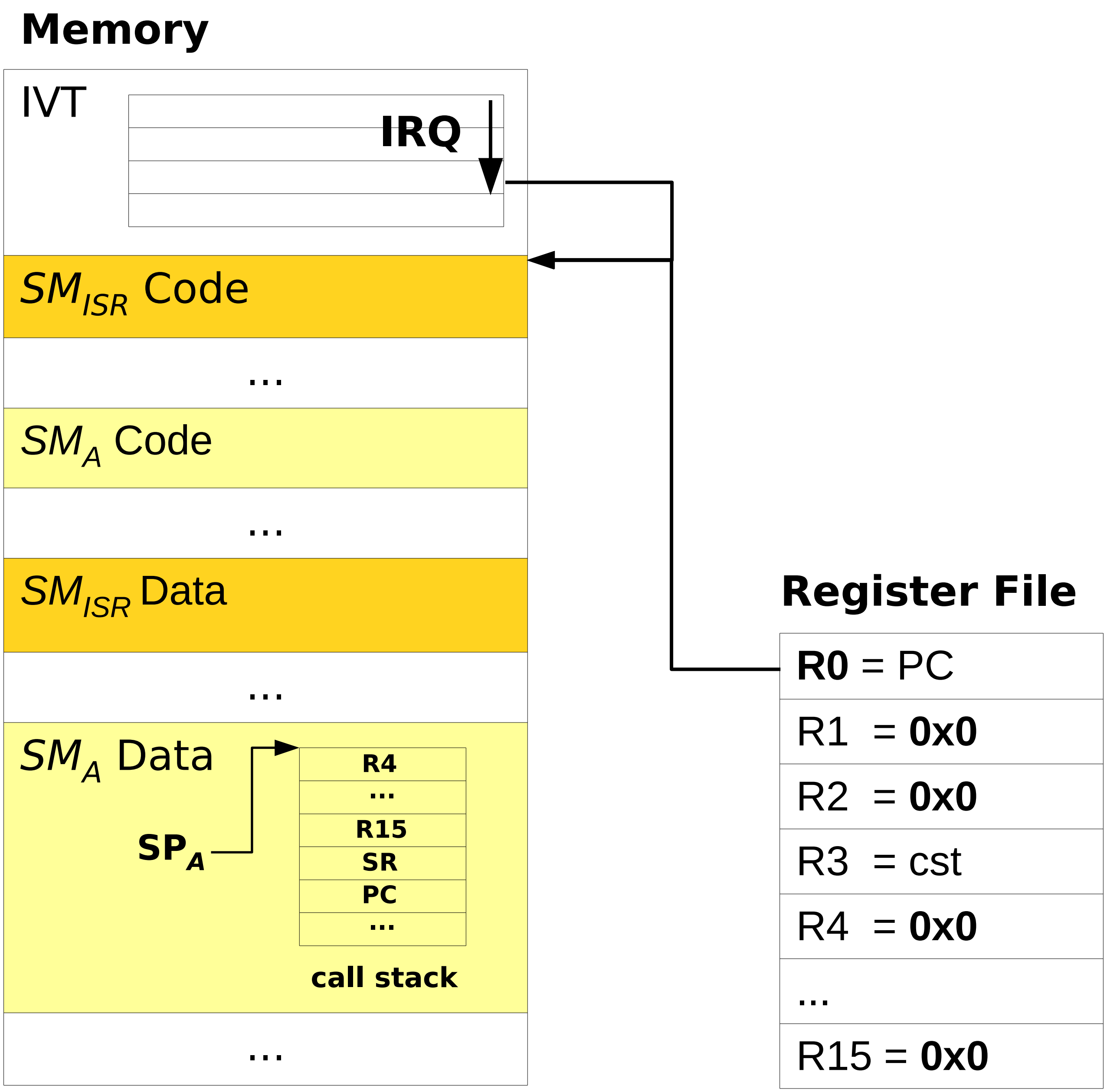
Interruptible and Reentrant SMs



Current SM = SM_A

Previous SM = x

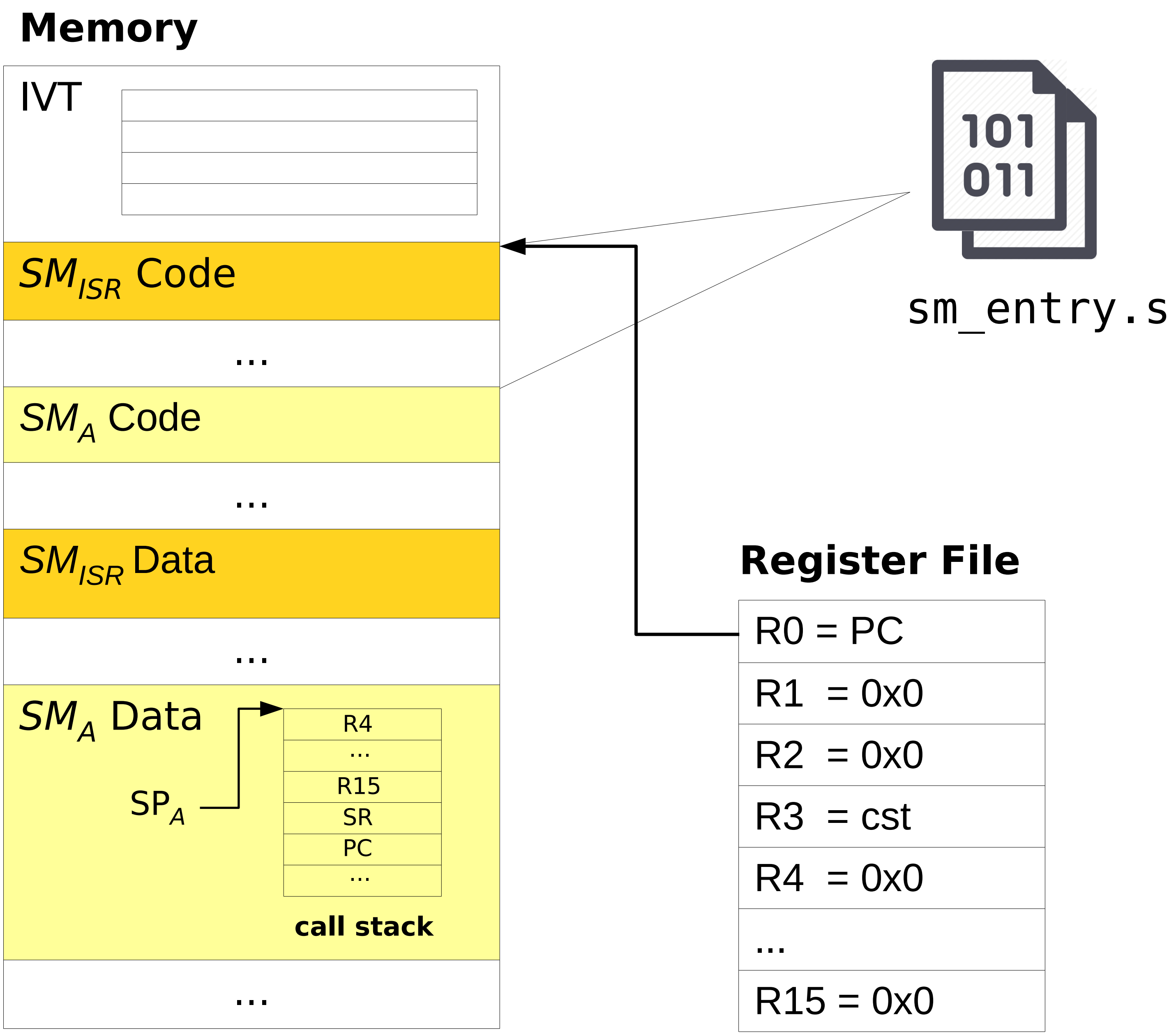
Interruptible and Reentrant SMs



Current SM = SM_A

Previous SM = x

Interruptible and Reentrant SMs



Current SM = $\boldsymbol{SM_{ISR}}$

Previous SM = ***IRQ***

KU LEUVEN

Discussion / Future Work

=> *Zero-software TCB for SM conf / int*

- **Atomicity** constraints (secure compilation)

  → deterministic *interrupt latency*

  → *TOCTOU*: callee authentication

  → `sm_entry`: restore SP, caller authentication

KU LEUVEN

Discussion / Future Work

=> *Zero-software TCB for SM conf / int*

- **Atomicity** constraints (secure compilation)
  - → deterministic *interrupt latency*
  - → *TOCTOU*: callee authentication
  - → `sm_entry`: restore SP, caller authentication
- **Untrusted ISRs**: integrity of `reti` flow

Contents

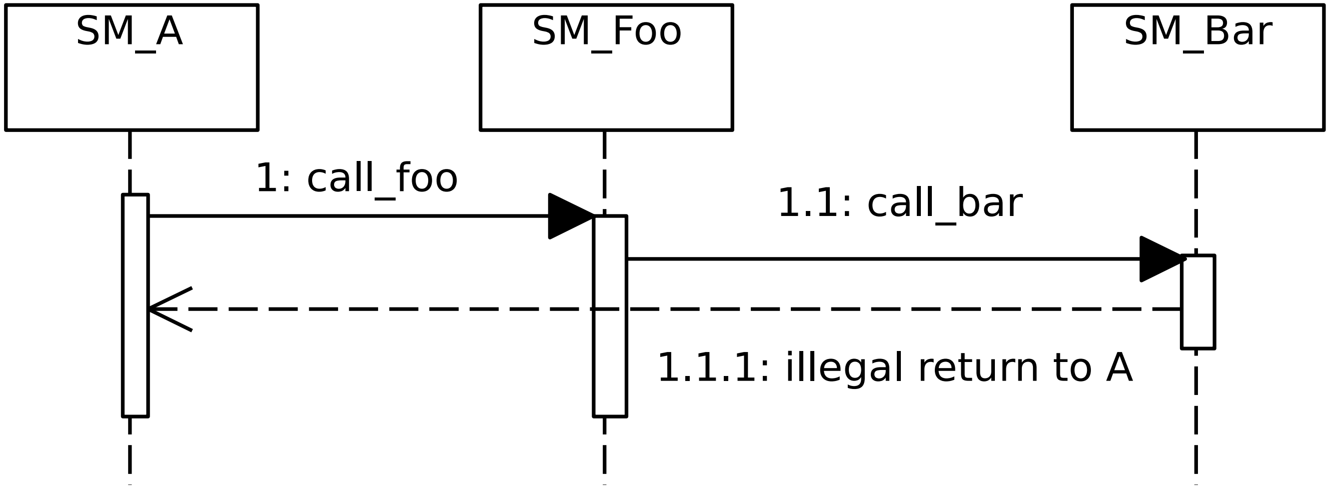
Traditional Multithreading vs. PMA

Synchronous control flow in address space

→ unit of **threading >> SM**

→ inter-SM call/return **integrity**

→ compiler-generated `sm_entry` stubs



KU LEUVEN

Protected FreeRTOS Scheduler

- **Interleaved** execution of multiple threads

  → cooperative prototype: `yield()`

Protected FreeRTOS Scheduler

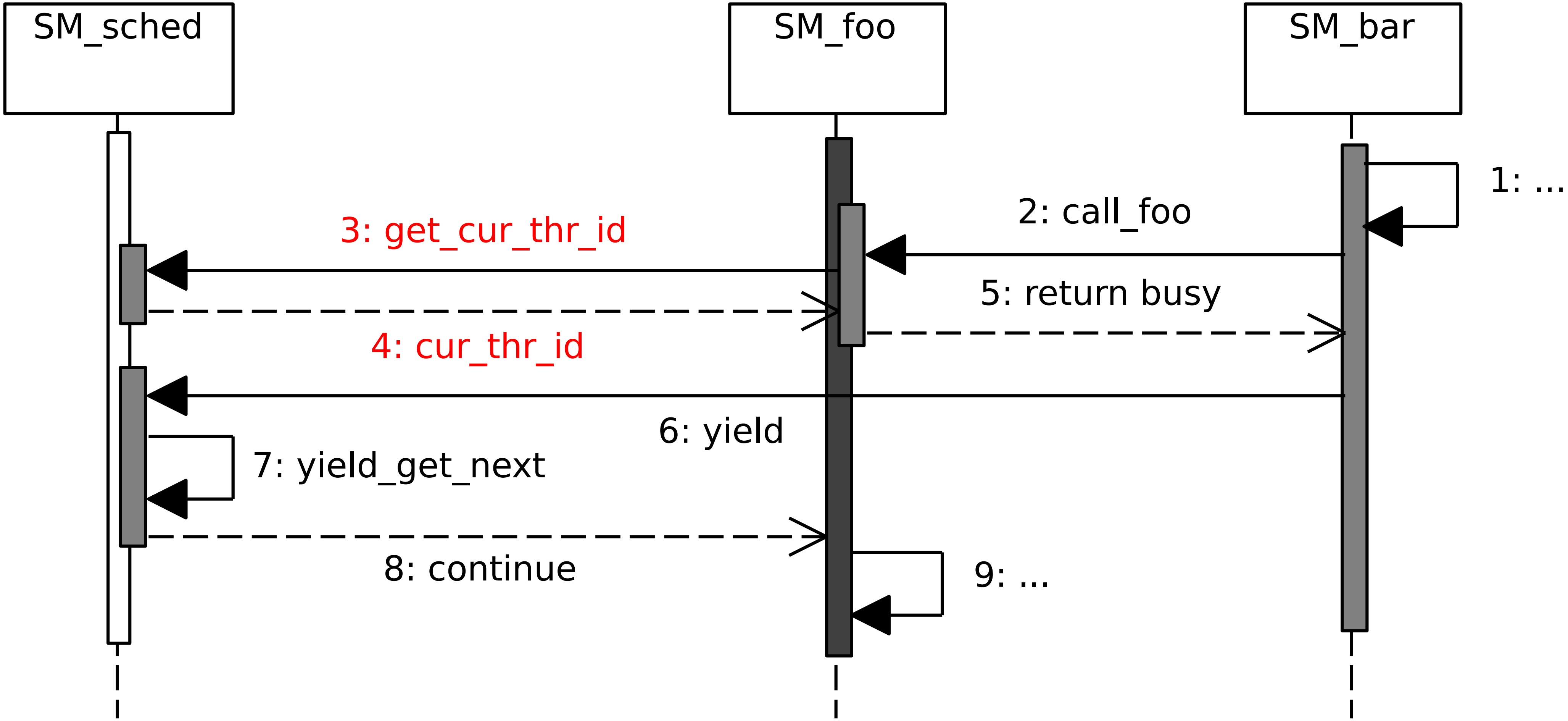
- **Interleaved** execution of multiple threads
  → cooperative prototype: `yield()`
- **Unprivileged**: scheduling decision only
  → store *"return address"* to continue thread
  → protected scheduler state

KU LEUVEN

Protected FreeRTOS Scheduler

- **Interleaved** execution of multiple threads
  → cooperative prototype: `yield()`
- **Unprivileged**: scheduling decision only
  → store *“return address”* to continue thread
  → protected scheduler state
- Secure **linking**: `sm_entry` violation report

Threading-aware SMs



Discussion / Future Work

=> *Isolated cross-SM control flow threads*

- Scheduling **policy encapsulation**
- **SMs** guard internal **consistency**

KU LEUVEN

Discussion / Future Work

=> *Isolated cross-SM control flow threads*

- Scheduling **policy encapsulation**
- **SMs** guard internal **consistency**

Future work:

→ preemptive FreeRTOS

→ SM-internal multithreading

→ asynchronous inter-thread communication

KU LEUVEN

Contents

KU LEUVEN

Conclusion

=> Strong availability (real-time) guarantees on a partially compromised platform

- Confined and explicit **TCB**
  - → HW-only for SM conf / int
  - → SW layer: *principle of least privilege*
- Secure **compilation** in preemptive context

KU LEUVEN

Towards Availability and Real-Time Guarantees for Protected Module Architectures

Jo Van Bulck, Job Noorman, Jan Tobias Mühlberg and Frank Piessens

https://distrinet.cs.kuleuven.be/software/sancus/

KU LEUVEN
DISTRINET RESEARCH GROUP

iMinds
CONNECT.INNOVATE.CREATE