Using **XIPFS** module in **RIOT** for post-issuance software deployment

funded by the IPCEI-CIS European program

September 02-04 2026

Gregory Guche Gilles Grimaud
2XS/SISE – CRIStAL/IRCICA – CNRS/University of Lille

Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

cnrs

With the contribution of orange™

Plan

- **Post-issuance software deployment.**
  - What, who and why ?
- **Previous work and assessment.**
  - What has been done and where are we now ?
- **Building executables.**
  - Executables file format for XiPFS, from C and Rust sources.
- **Offline deployment.**
  - **Making XiPFS mountpoint images.**
  - **Embedding XiPFS images in RIOT.**
- **Online deployment and execution.**

Post-issuance software deployment

« In an industrial mass-production setting, post-issuance deployment primarily allows for a basic system image to be written to ROM, while still leaving the door open for future extensions, amendments, and partnerships once millions of units are already deployed all around the world. »

Post-issuance software deployment

Offline deployment

Chip Maker

- Silicon production
- Software in ROM

Manufacturer

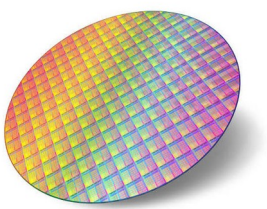
- Device production
- Single shot software in Flash

Retailer

- Device distribution
- Software personalization.
- Third party software.

Consumer

- Device usage
- Software provisioning
- Updates
- Customization
- Management









Online deployment

Timeline

Previous work and assessment. (1/5)

- "RIOT over PIP embedding secure OTA deployment"
  - G. Grimaud/D. Amara, 2XS, University of Lille.
  - RIOT Summit 2023
  - grimaud.pdf
  - youtube

- In a Nutshell
  - e**X**ecute **i**n-**P**lace **F**ile**S**ystem (XiPFS).
    - As a **V**irtual **F**ile **S**ystem module in RIOT OS
    - Execute code from Flash, executable read/write data + stack in RAM
  - RIOT over PIP *(hypervisor)*
  - Pip
    - Proto-kernel formally proven
      - memory partitioning
      - execution flow transfer
      - hardware interrupt handling
      - hardware register access

XiPFS

RIOT

Pip

Previous work and assessment. (2/5)

- **RIOT over Pip + XiPFS**
  - **Pros**
    - It works ! *(yay)*
    - Formally proven with Coq proof assistant.
    - Memory partitioning through MPU slows down code execution by 5 to 10%.
  - **Cons**
    - From maintainers point of view :
      - Keeping up that port with RIOT releases may involve a substantial amount of work.
    - For newcoming developers :
      - Need to setup an extra software environment for Pip (OPAM, Coq, DX, etc...)
      - Learning curve to make actual firmware.
      - Might be an impediment for creating software.

Previous work and assessment. (3/5)

- **A complementary approach to ease the development cycle**
  - **What about :**
    - Removing Pip from the equation.
    - Keeping XiPFS as a RIOT module.
    - Adding memory isolation support into XiPFS.
      - By switching to **Unprivileged Thread mode** *(vertical isolation)*
      - By **hardware Memory Protection Unit** *(horizontal isolation)*
        - RIOT pseudomodules : «stack guard» + «no RAM execution»
    - Leaving other features to RIOT OS
      - Threads, scheduling, interrupts, etc.

XiPFS
UserMode+MPU

RIOT

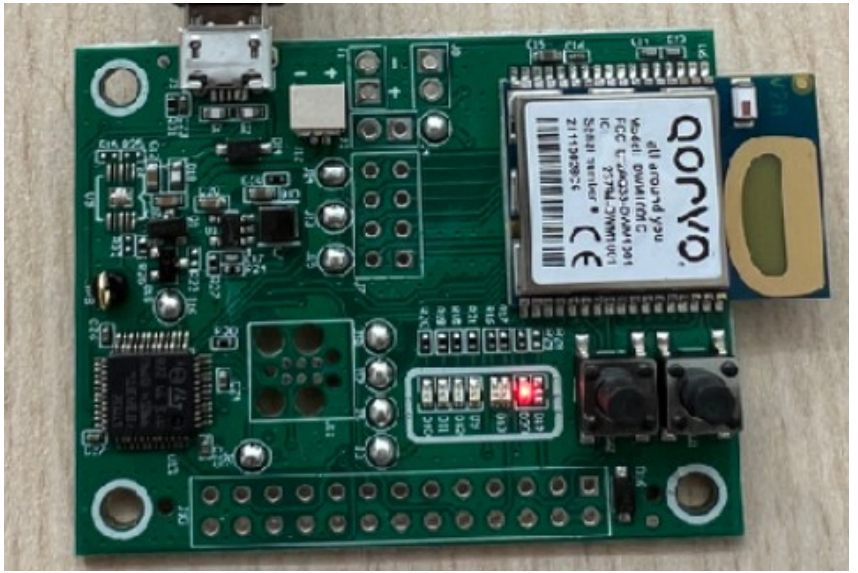
  - **Use XiPFS+MPU for prototyping**
  - **Keep Pip for industrial projects with strong security concerns**

Previous work and assessment. (4/5)

XiPFS in RIOT now :

- Should run on platforms with :
  - ARMV7 Microcontroller profile
    - 32 bits CPU (cortex-m{3, 4, 7}).
    - Thumb instructions mode *(v2 here)*.
  - Addressable flash memory with pages.
  - When memory isolation is required :
    - Hardware **M**emory **P**rotection **U**nit (PMSAv7).
    - Supervisor calls support (svc).

- So far, only the DWM1001 board has been tested and is confirmed to function correctly.



DWM1001

nrf52832 SOC:

- *Cortex-m4.*
- *Hardware MPU.*
- *RAM : 64 KiB. (0x20000000 – 0x2000FFFF)*
- *Flash : 512 KiB (0x00000000 - 0x0007FFFF)*

Previous work and assessment. (5/5)

XiPFS in RIOT now :

- RIOT/examples/advanced/xipfs : minimal/memory image/complete
- Code

```
int xipfs_extended_driver_new_file(const char *full_path, uint32_t size, uint32_t exec);
/* No memory protection */
int xipfs_extended_driver_execv(const char *full_path, char *const argv[]);
/* With memory protection */
int xipfs_extended_driver_safe_execv(const char *full_path, char *const argv[]);
```

- Pyterm shell

```
$ create_executable full_path_to_exe bytesize
# No memory protection
$ execute full_path_to_exe [args…]
# With memory protection
$ execute -p full_path_to_exe [args...]
```

« As a software developer, I would like to... »

Building executables

- **«... build executables from C sources»**
  - **FAE format**
    - Tool converting ELF files to FAE files *(ad hoc binary file format)*
      - relocation information.
      - .text, .got, .data sections.
    - ~20% in size of the equivalent ELF file for the same software.
    - **https://github.com/2xs/fae_format**
  - **XiPFS demonstrations**
    - Various examples sources with required GCC compilation flags.
    - Rely on **FAE format** to make FAE files.
    - **https://github.com/2xs/riot-xipfs-demonstrations**
- **«... build executables from Rust sources »**
  - **XIPRFS**
    - Tool compiling Rust sources to ELF, then converted to FAE format.
    - No dependency with FAE format repo. Comes with examples.
    - Git repository soon publicly available

« As a manufacturer, I would like to… »

«... prepare my filesystem in its initial state on software creation»

- Developers need a tool to create a filesystem in a desired initial state.
- Once done, the filesystem must be included in source code.
- The filesystem must be compiled to produce the initial firmware.

- At first boot, the software finds the initial filesystem.
- Then the files evolve according to the software uses.
- Files evolutions stored in the filesystem stay persistent across following reboots.

- Like constfs, XiPFS allows to build an initial memory image.
- Like spiffs and others, XiPFS allows to keep evolutions persistent in Flash memory
- Moreover, XiPFS allows to run executable files.

«... make XiPFS memory images offline»

XiPFS' mkxipfs tool

- Workstation utility used to create, inspect, and modify XiPFS flash images (***.flash**).
  - The very same code is used on boards and workstations !
- **https://github.com/2xs/xipfs**

```
$ cd xipfs
$ make -C tools
```

- Building an initial filesystem from an existing directory.

```
$ tools/bin/mkxipfs --target dwm1001 build ~/flash/nvm0.flash ~/fae_files/  64k
```

- But we can also manipulate flash images with the tool.

```
$ tools/bin/mkxipfs --target dwm1001 create ~/flash/nvm1.flash 40k
$ tools/bin/mkxipfs --target dwm1001 --flash ~/flash/nvm1.flash put host_file.fae embedded_file.fae
$ tools/bin/mkxipfs --target dwm1001 ls ~/flash/nvm1.flash
```

- Other mkxipfs commands: *tree, get, mkdir, rm, rmdir, mv, help, etc...*

«... embed XiPFS images within RIOT» (1/2)

- Just use of RIOT's BLOBS mechanism :
  - Makefile :

```
BLOBS += nvme0p0.flash
```

  - main.c :

```
static
XIPFS_START_PARTITION_INCLUSION(nvme0p0)
#include "blob/nvme0p0.flash.h"
XIPFS_END_PARTITION_INCLUSION(nvme0p0, "/nvme0p0", nvme0p0_flash, nvme0p0_flash_len);
...
vfs_mount(&nvme0p0.vfs_mp)
...
```

«... embed XiPFS images within RIOT» (2/2)

- For reference :
  - RIOT/examples/advanced/xipfs/xipfs_memory_image_example/

```
$ ls RIOT/examples/advanced/xipfs/xipfs_memory_image_example/
Makefile  Makefile.ci  README.md  main.c  nvme0p0.flash
```

  - RIOT/examples/advanced/xipfs/xipfs_complete_example/

```
$ ls RIOT/examples/advanced/xipfs/xipfs_complete_example/
Makefile  Makefile.ci  README.md  main.c  nvme0p0.flash
```

«... embed XiPFS images within RIOT» (2/2)

- Just use of RIOT's BLOBS mechanism :
  - Makefile :

```
BLOBS += nvme0p0.flash
```

  - main.c :

```
static
XIPFS_START_PARTITION_INCLUSION(nvme0p0)
#include "blob/nvme0p0.flash.h"
XIPFS_END_PARTITION_INCLUSION(nvme0p0, "/nvme0p0", nvme0p0_flash, nvme0p0_flash_len);
...
vfs_mount(&nvme0p0.vfs_mp);
...
```

Demonstration !



« As a retailer or a customer, I would like to... »

Online deployment and execution

- **«… manage deployed contents OTA»**
  - To add/remove/update executable software, once products are in the wild.
- **«… run FAE files remotely»**
  - with and without memory protection.
- **Proof of concept**
  - Not yet available in RIOT - *Needs much polishing*
  - Ethernet Over Serial (**Ethos**)
  - **IPV6**
  - Constrained Application Protocol (**CoAP**)
    - **REST** software architecture (client-server)
    - **UDP**
    - **Disclaimer : we are not experienced in CoAP !**
  - Based on
    - RIOT examples : *gnrc_border_router, gcoap_fileserver, gcoap_block_server.*
    - https://connect.ed-diamond.com/open-silicium/os-019/mise-en-place-d-un-reseau-iot-avec-riot (Alexandre Abadie)

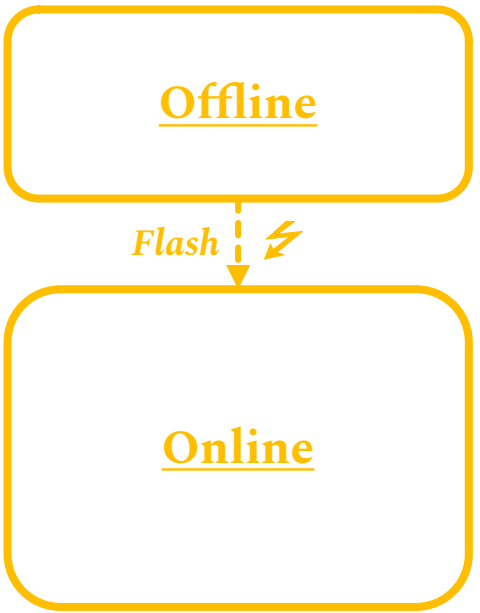
Online deployment and execution

- **«... manage deployed contents OTA»**
  - To add/remove/update executable software, once products are in the wild.
- **«... run FAE files remotely»**
  - with and without memory protection.
- **Proof of concept**
  - Not yet available in RIOT - *Needs much polishing*
  - Ethernet Over Serial (**Ethos**)
  - **IPV6**
  - Constrained Application Protocol (**CoAP**)
    - **REST** software architecture (client-server)
    - **UDP**
    - **Disclaimer : we are not specialized in CoAP !**
  - Based on
    - RIOT examples : *gnrc_border_router, gcoap_fileserver, gcoap_block_server.*
    - https://connect.ed-diamond.com/open-silicium/os-019/mise-en-place-d-un-reseau-iot-avec-riot (Alexandre Abadie)

Demonstration !

Conclusion

- **Safe and secure post-issuance software deployment is possible.**
- **We have seen how to :**
  - build executables for IOT platforms
    - from both C and Rust sources
  - embed them within RIOT OS at compilation time
  - run executables from shell in RIOT OS
  - run executables from your own software
    - with and without memory protection
  - deploy, install and remove software after an initial release
- **It's up to you now ! ;)**



Thanks

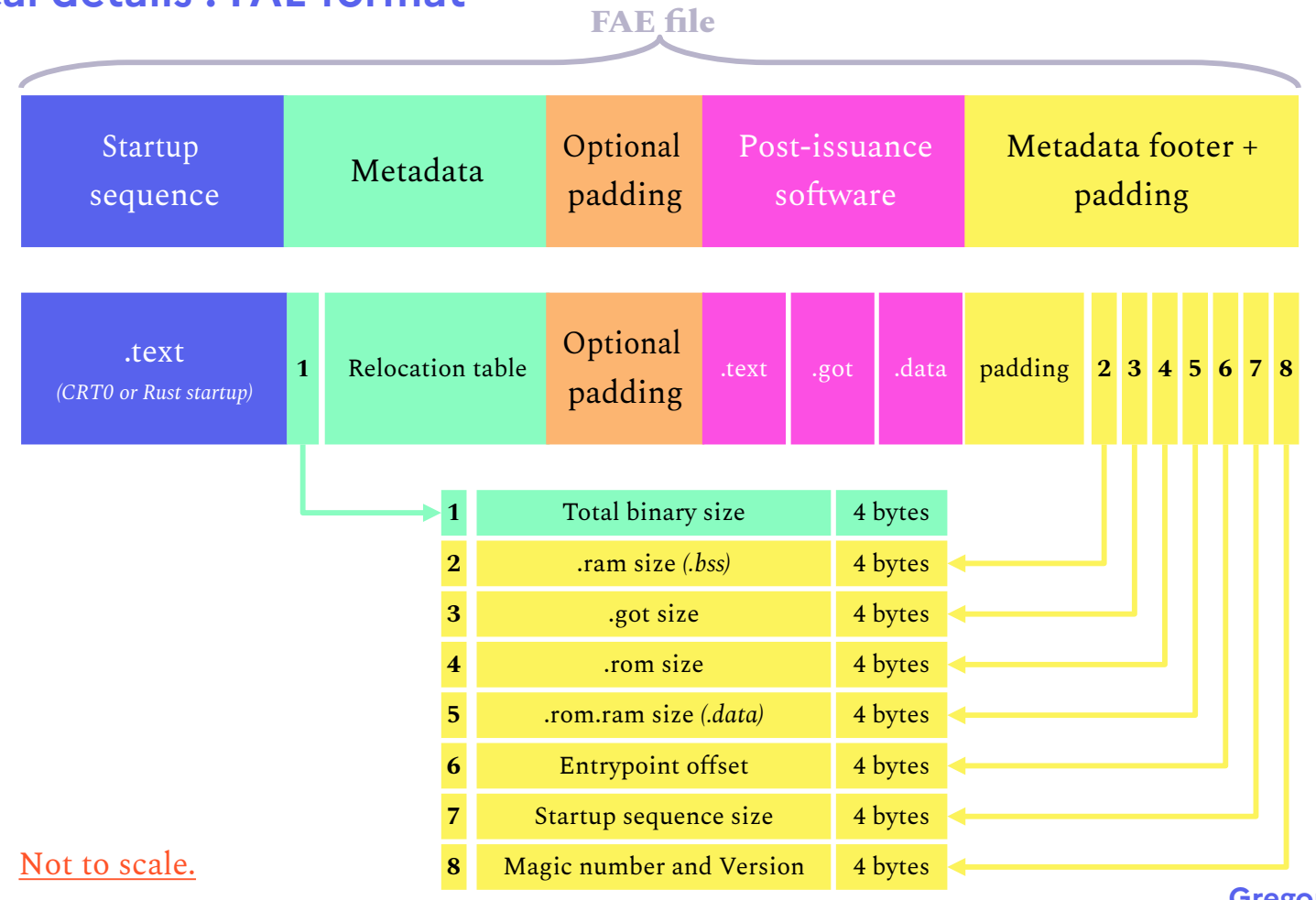
- **2XS team**
  - Gilles Grimaud and Damien Amara, for their advices and patience.
- **RIOT reviewers**
  - For their patience and suggestions
    - crasbe, AnnsAnns, aabadie, kfessel, maribu, miri64, Teufelchen1, benpicco...

Thank you for your attention.

Any questions ?

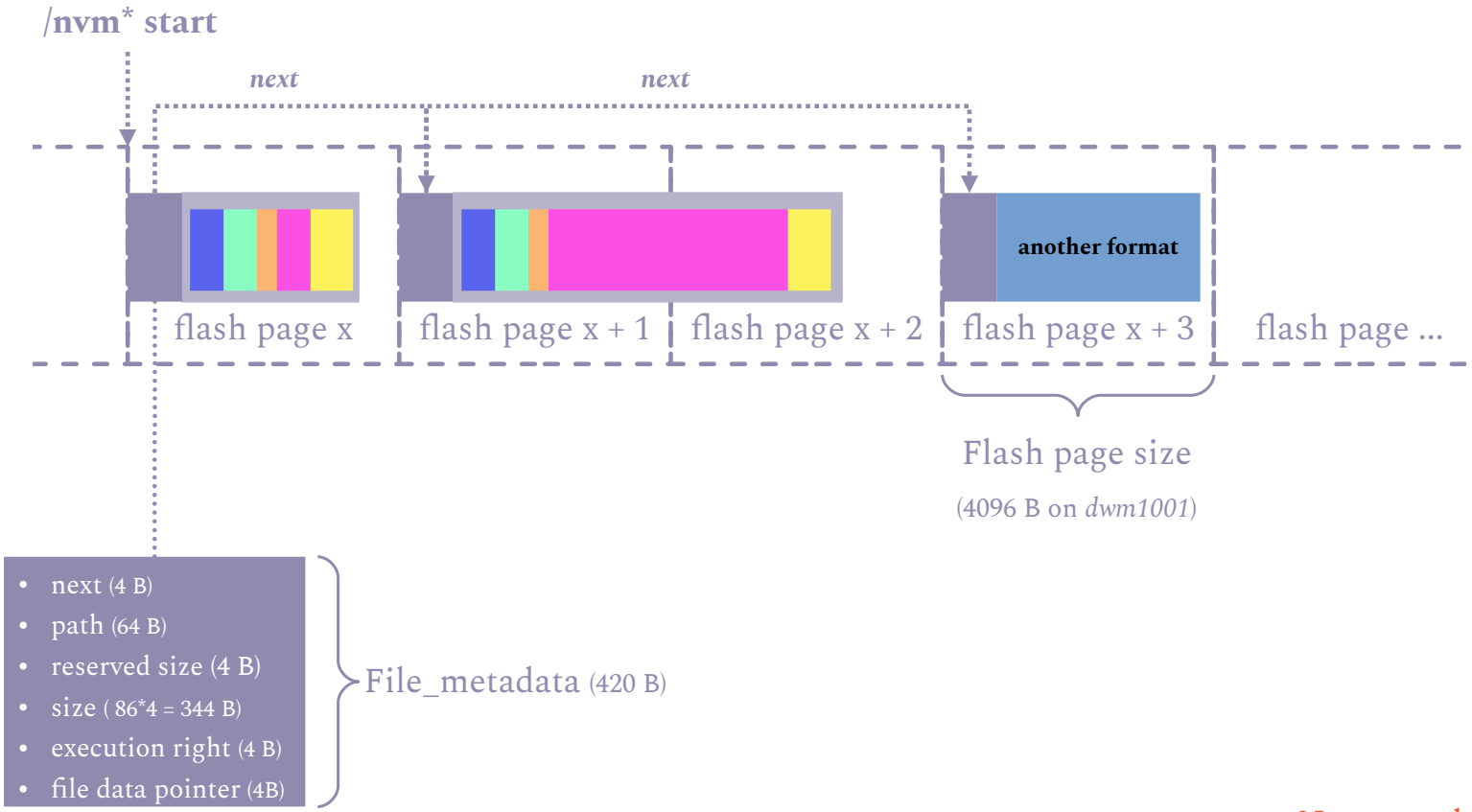
Contacts :
gregory.guche@univ-lille.fr
gilles.grimaud@univ-lille.fr

Technical details : FAE format



Not to scale.

Technical details : XiPFS



Not to scale.

Technical details : Shared API

Features set shared between the host OS and executables

- Without MPU protection, simple array of functions pointers
- With MPU protection, array of trampoline functions pointers triggering **S**uper**V**isor **C**alls.
- Main branch:
  - *exit, vprintf, get_temp, isprint, strtol, get_led, set_led, copy_file, get_file_size, memset*
- Branch vfs_syscalls:
  - *vsnprintf, memcmp, strcmp, strncmp*
  - *vfs_open, vfs_close, vfs_lseek, vfs_write, vfs_read, vfs_readline, vfs_stat, vfs_fstat, vfs_statvfs, vfs_fstatvfs, vfs_rename, vfs_normalize_path, vfs_fsync, vfs_fcntl, vfs_mkdir*
  - *open, close, lseek, write, read, stat, fstat, statvfs, fstatvfs, rename, fsync, fcntl, mkdir*
- Branch rust_support:
  - *print_fmt*

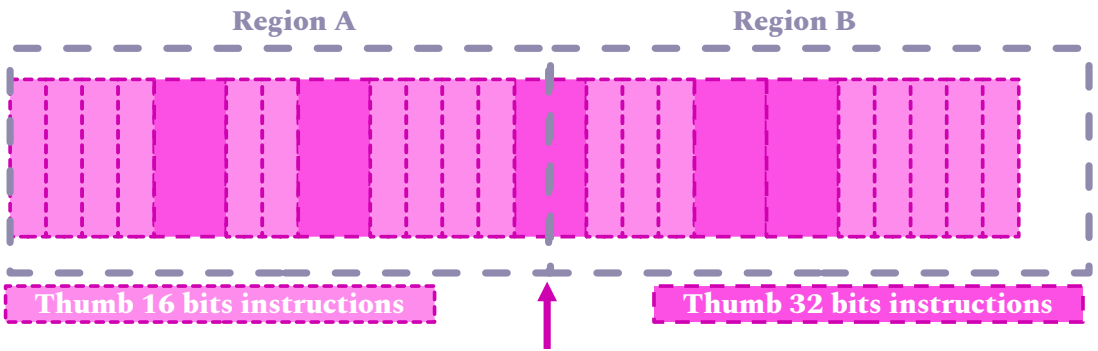
Technical details : PMSAv7 : MPU

Regions

- User defined **permissions rights *(rwx)*** on memory areas given a starting address and a size
  - Checked on memory accesses
  - **Size** must be a power of 2 *(MPU_RASR)*
    - ≥ 32 B
    - <= 4 GiB
  - **Address** must be aligned on an integer multiple of **size** *(MPU_RBAR)*
- *« Where there is an overlap between two regions, the register with the highest region number takes priority »*
- Armv7-M only supports a unified MPU *(MPU_TYPE)*
  - No difference between instructions and data
- Potentially up to 256 regions *(MPU_TYPE and MPU_RNR)*
  - But usually **8** to 16 regions *(MPU_TYPE and MPU_RBAR)*
- Potentially up to 8 sub-regions per region *(MPU_RASR)*
  - for regions which size ≥ 256 bytes *(not used by XiPFS and RIOT)*

Technical details : code isolation with MPU

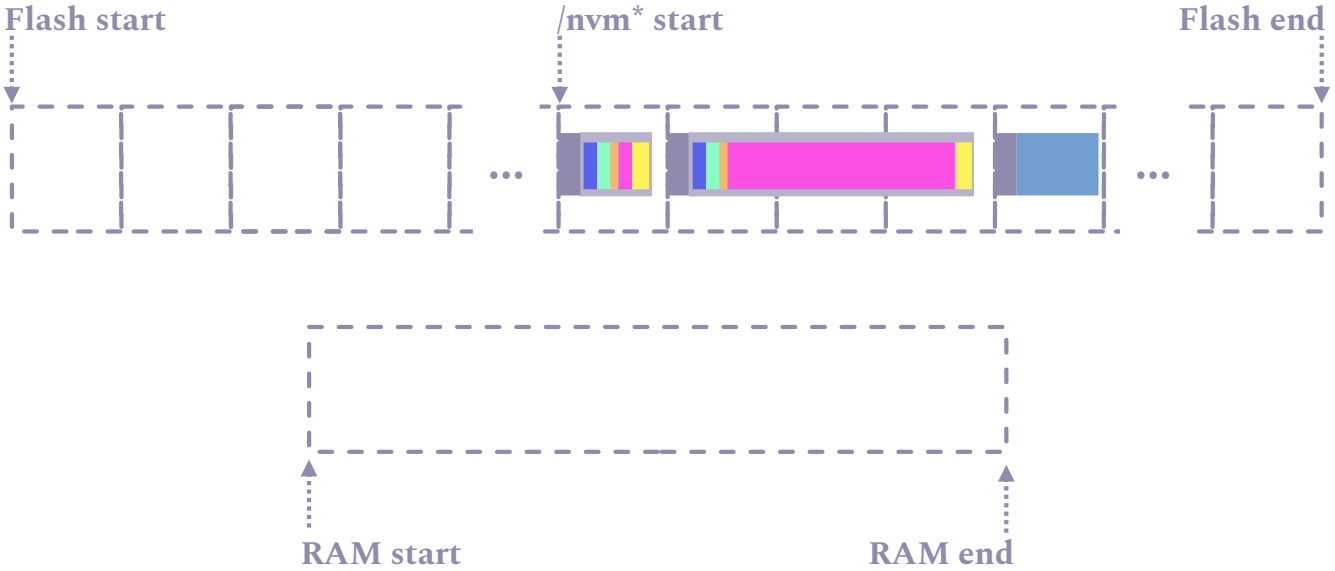
- Not possible to use only 1 region, because of :
  - MPU constraints
    - 32B ≤ size ≤ 4 GiB
    - Size must be a power of 2
      - Flash pages size is a power of 2
    - Address is an integer multiple of size.
  - Thumb v2 code can straddle a memory region boundary
    - Mix between 16 and 32 bits instructions.
    - Need to have 2 memory regions defined at the same time.



Technical details : MPU settings : current (1/5)

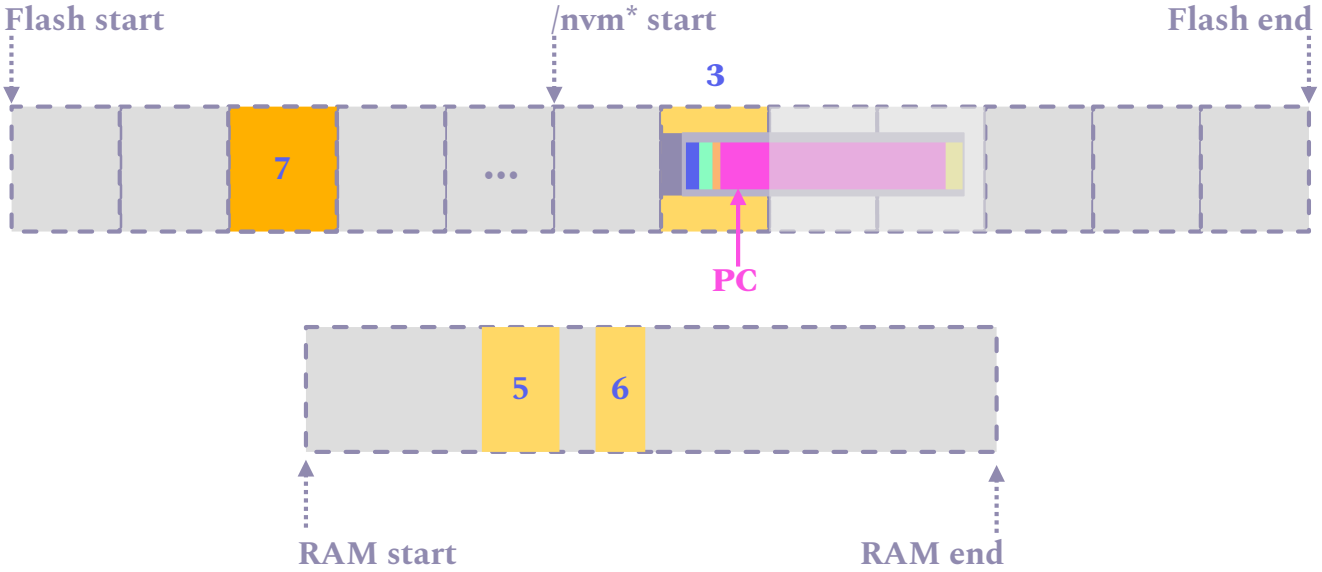
Region #	Owner	Name	Target	Memory	Privileged rights	User rights	Execute	Size
0	RIOT	No execution	Whole RAM	RAM	RW	RW	N	512KiB
1	RIOT	Stack guard	ISR stack	RAM	RO	RO	N	32B
2	RIOT	Stack guard	Current thread stack	RAM	RO	RO	N	32B
3	XiPFS	.text (1/2)	Exe code	Flash	RO	RO	Y	4KiB*(may vary)*
4	XiPFS	.text (2/2)	Exe code	Flash	RO	RO	Y	4KiB*(may vary)*
5	XiPFS	.rom_ram +.ram +.bss + FREE	Exe RAM	RAM	RW	RW	N	4KiB*(may vary)*
6	XiPFS	stack	Exe stack	RAM	RW	RW	N	2KiB*(may vary)*
7	XiPFS	Shared API	Shared API	Flash	RO	RO	Y	4KiB*(may vary)*

Technical details : MPU settings : current (2/5)



Not to scale.

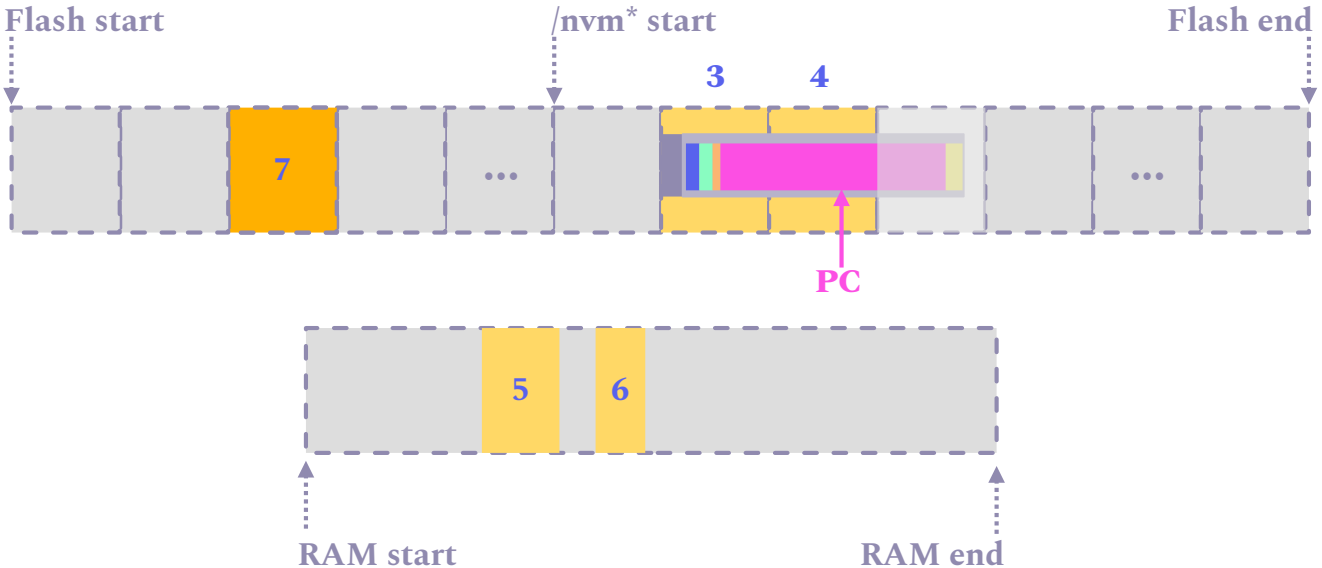
Technical details : MPU settings : current (3/5)



Region #	Target	Privileged rights	User rights	Execute
3	Exe code	RO	RO	Y
4	Exe code	RO	RO	Y
5	Exe RAM	RW	RW	N
6	Exe stack	RW	RW	N
7	Shared API	RO	RO	Y

Not to scale.

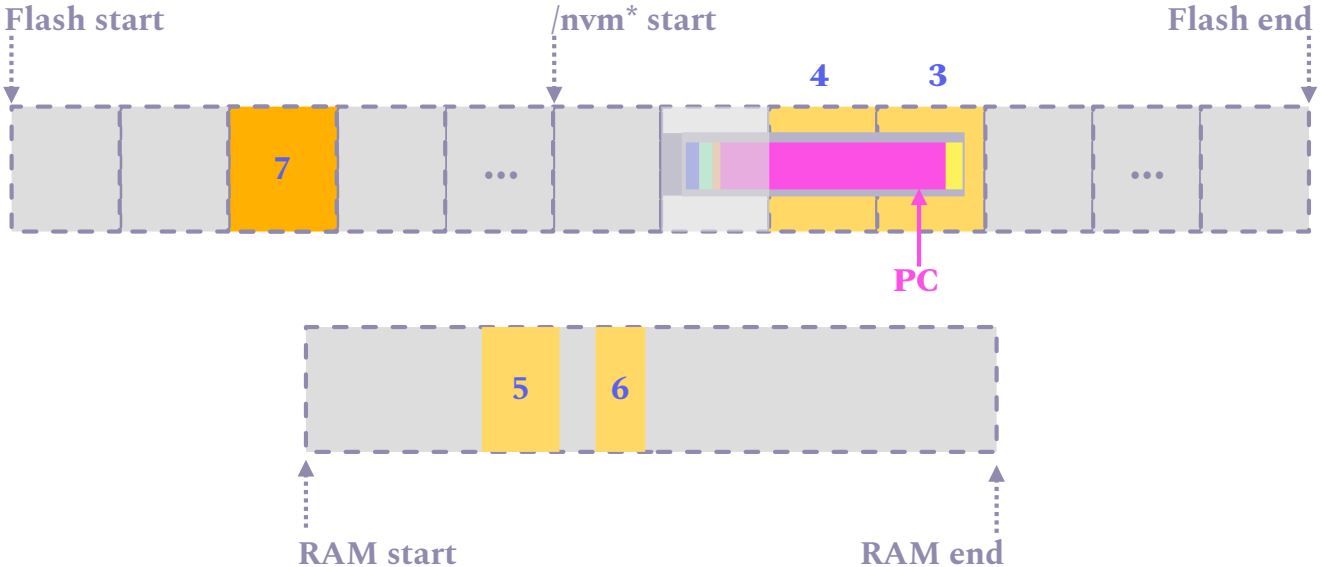
Technical details : MPU settings : current (4/5)



Region #	Target	Privileged rights	User rights	Execute
3	Exe code	RO	RO	Y
4	Exe code	RO	RO	Y
5	Exe RAM	RW	RW	N
6	Exe stack	RW	RW	N
7	Shared API	RO	RO	Y

Not to scale.

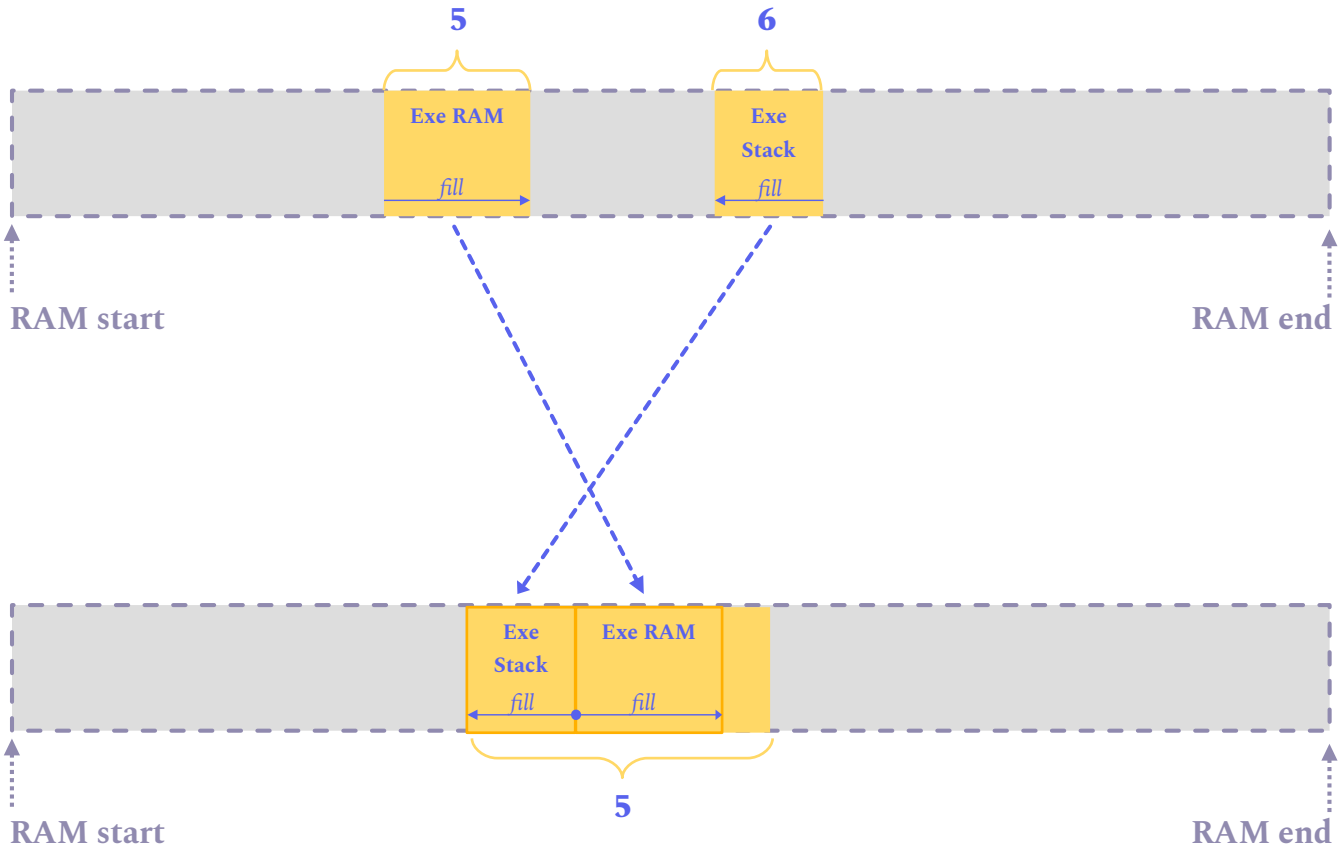
Technical details : MPU settings : current (5/5)



Region #	Target	Privileged rights	User rights	Execute
3	Exe code	RO	RO	Y
4	Exe code	RO	RO	Y
5	Exe RAM	RW	RW	N
6	Exe stack	RW	RW	N
7	Shared API	RO	RO	Y

Not to scale.

Technical details : MPU settings : next (1/3)



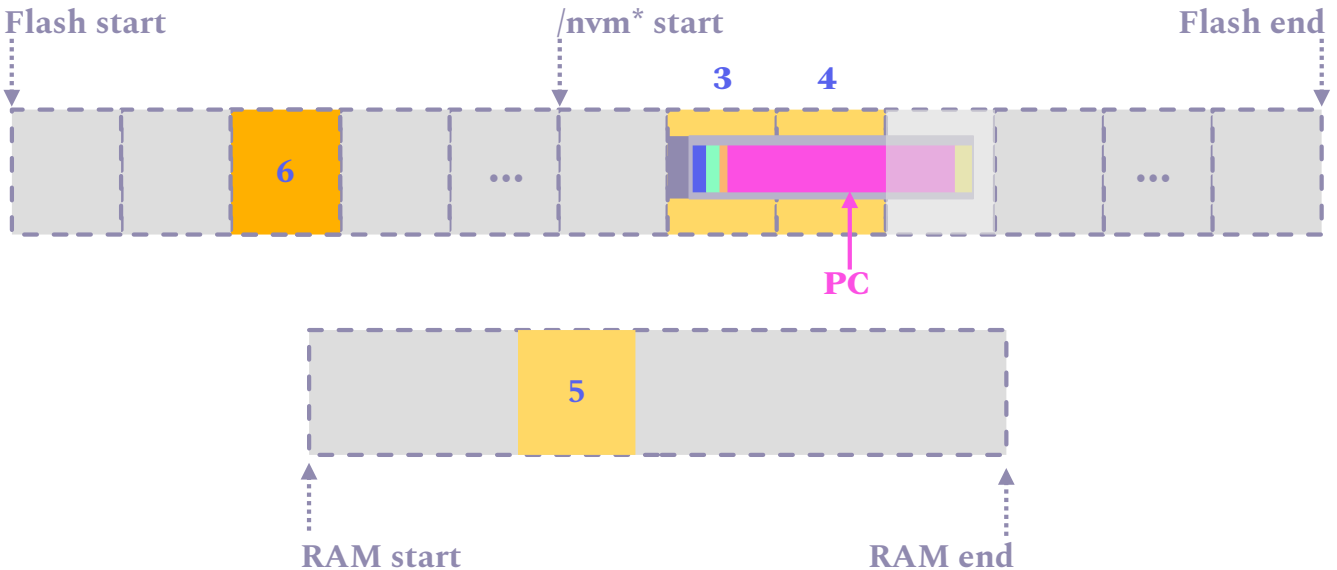
Not to scale.

Technical details : MPU settings : next (2/3)

Region #	Owner	Name	Target	Memory	Privileged rights	User rights	Execute	Size
0	RIOT	No execution	Whole RAM	RAM	RW	RW	N	512KiB
1	RIOT	Stack guard	ISR stack	RAM	RO	RO	N	32B
2	RIOT	Stack guard	Current thread stack	RAM	RO	RO	N	32B
3	XiPFS	.text (1/2)	Exe code	Flash	RO	RO	Y	4KiB*(may vary)*
4	XiPFS	.text (2/2)	Exe code	Flash	RO	RO	Y	4KiB*(may vary)*
5	XiPFS	Stack + .rom_ram +.ram +.bss + FREE	Exe Stack Exe RAM	RAM	RW	RW	N	8KiB*(may vary)*
6	XiPFS	Shared API	Shared API	Flash	RO	RO	Y	4KiB*(may vary)*
7	*Free*	*Free*	*Free*	*Free*	*Free*	*Free*	*Free*	*Free*

Université de Lille

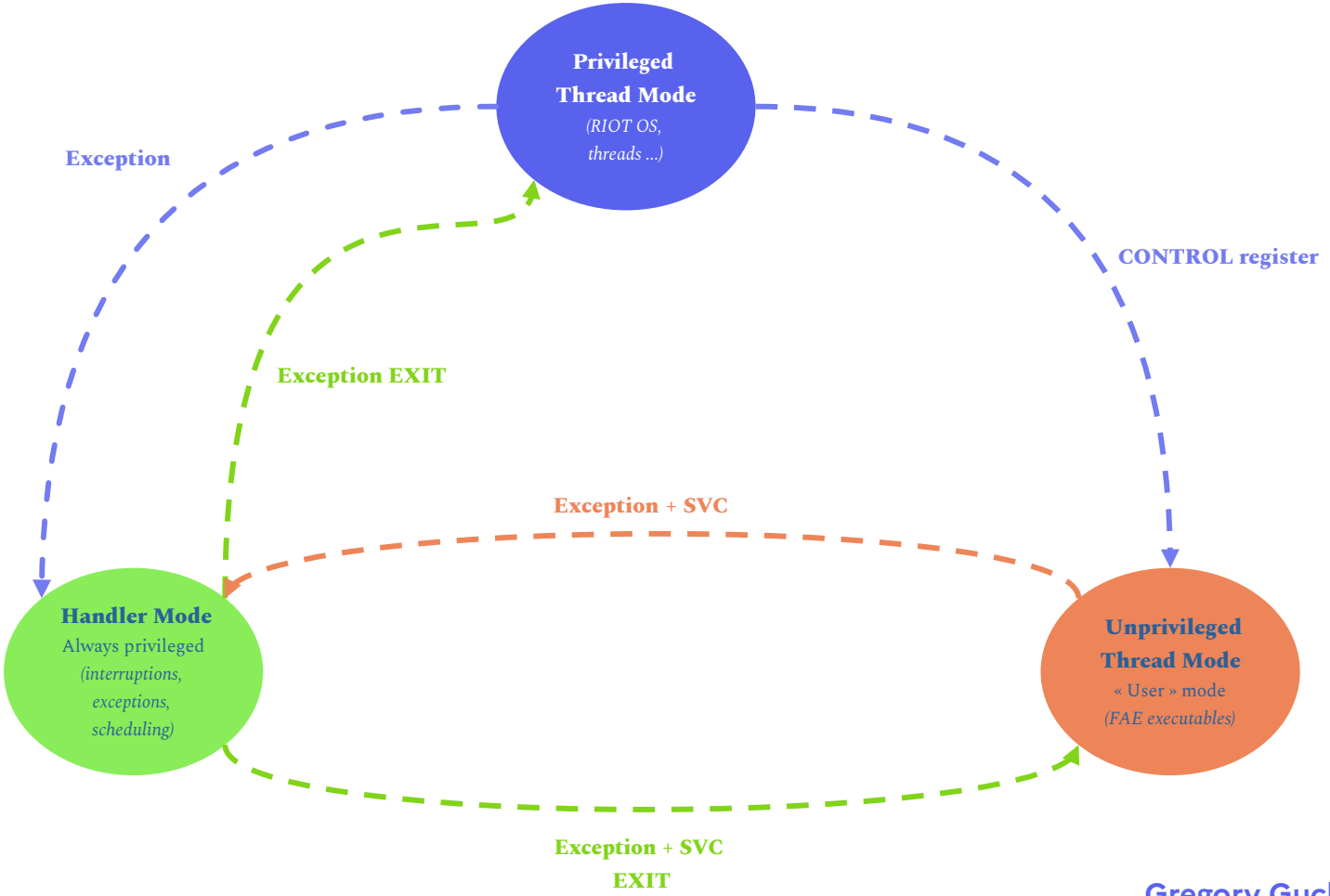
Technical details : MPU settings : next (3/3)



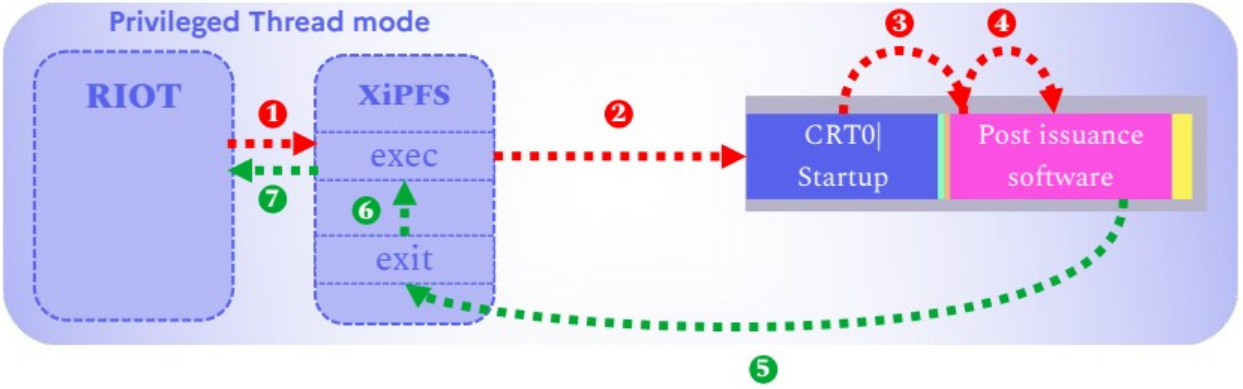
Region #	Target	Privileged rights	User rights	Execute
3	Exe code	RO	RO	Y
4	Exe code	RO	RO	Y
5	Exe STACK + RAM	RW	RW	N
6	Shared API	RO	RO	Y
7	*Free*	*Free*	*Free*	*Free*

Not to scale.

Technical details : ARM execution modes

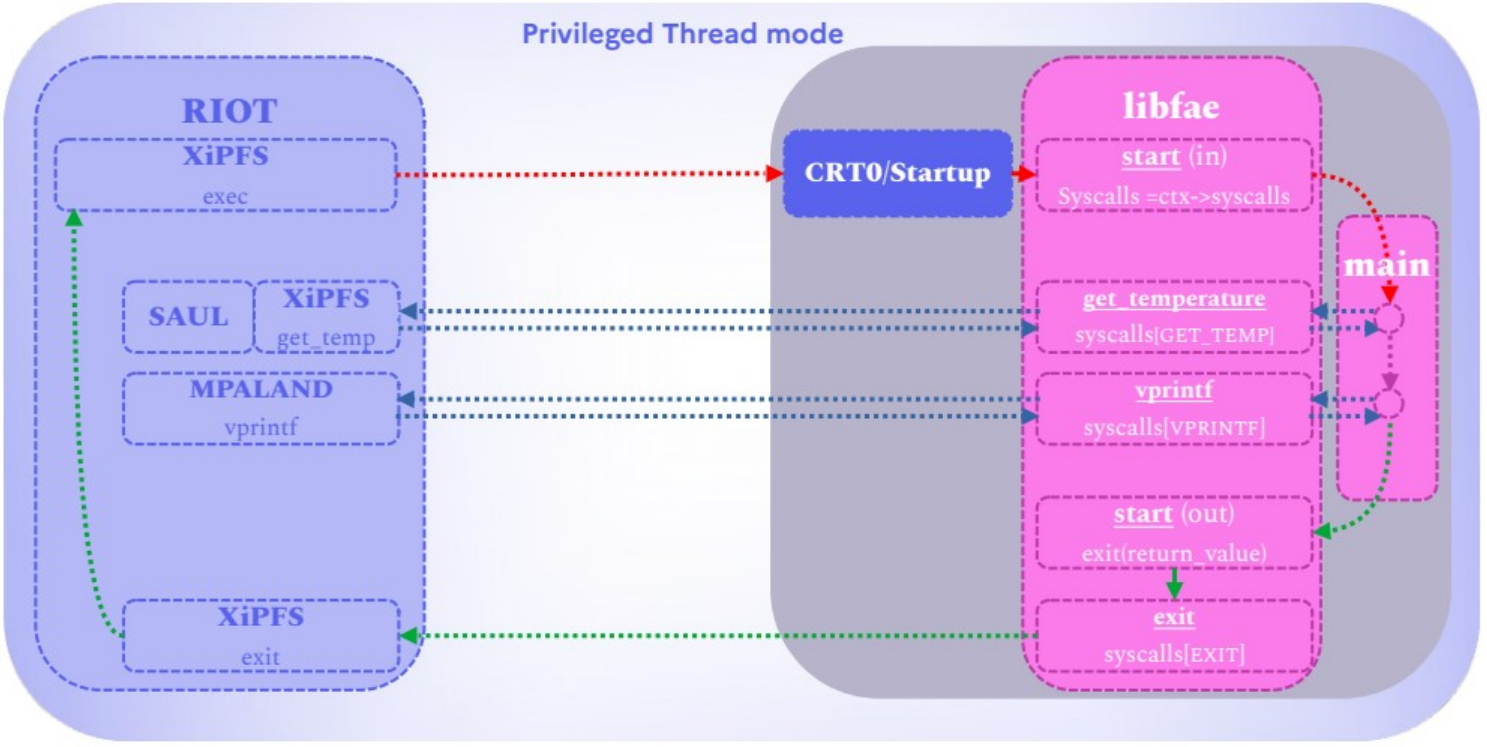


Technical details : XiPFS : execution (1/2)

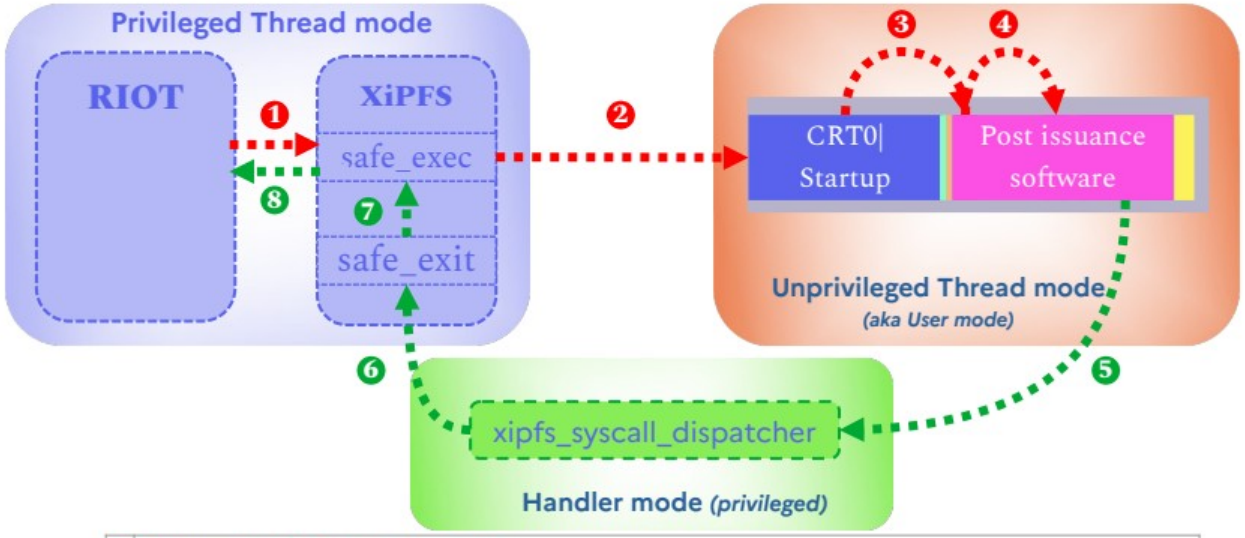


1	Call XiPFS with filename and arguments
2	• Saves current stack and PC, sets executable stack • Branches to the first 4 bytes of FAE file (CRT0/Startup) with context (args + syscalls table + ...)
3	CRT0/Startup relocates got + data, initialize bss, then jumps to software entrypoint *(start function)* with context
4	*start* sets software environment from context, then calls *main* with arguments
5	After *main* execution, start calls *exit* with main's return value
6	*exit* restores stack and PC, and sets return value into R0 register
7	*exec* clears executable memories and returns R0 register to callers (RIOT)

Technical details : XiPFS : execution (2/2)



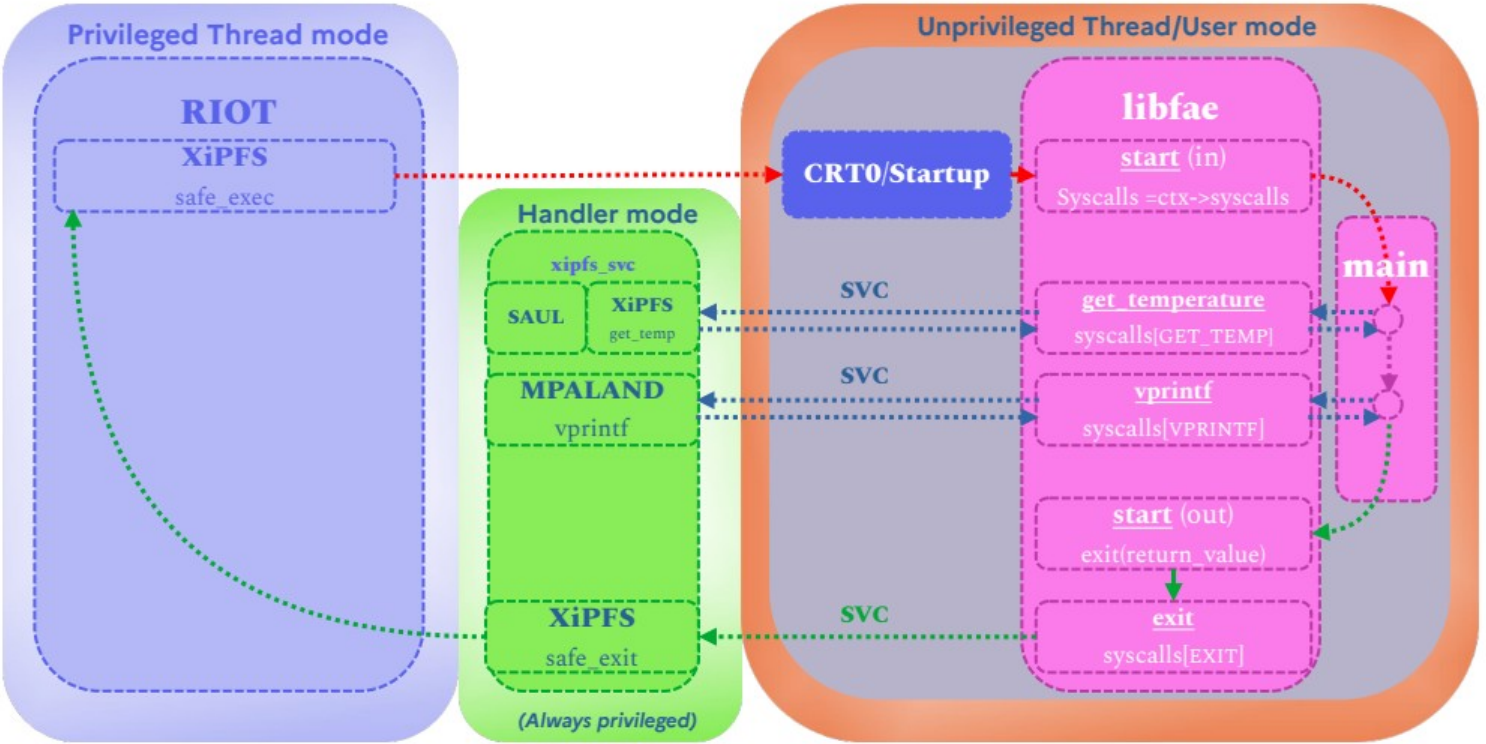
Technical details : XiPFS : safe execution (1/2)



1	Call XiPFS with filename and arguments
2	• Sets MPU regions + enable • Saves current stack and PC, sets executable stack • Enters User mode and branches to the first 4 bytes of FAE file (CRT0/Startup) with context (args + syscalls table + ...)
3	CRT0/Startup relocates got + data, initialize bss, then jumps to software entrypoint *(start function)* with context
4	*start* sets software environment from context, then calls *main* with arguments
5	After *main* execution, start calls safe_exit syscall with main's return value
6	Dispatcher calls the actual *safe_exit* with return value
7	*safe_exit* goes to Privileged Thread mode, restores stack and PC, and sets return value into R0 register
8	*safe_exec* clears executable memories, unsets MPU regions and returns R0 register to callers (RIOT)

Université de Lille

Technical details : XiPFS : safe execution (2/2)



Technical details : PMSAv7 : MPU_CTRL

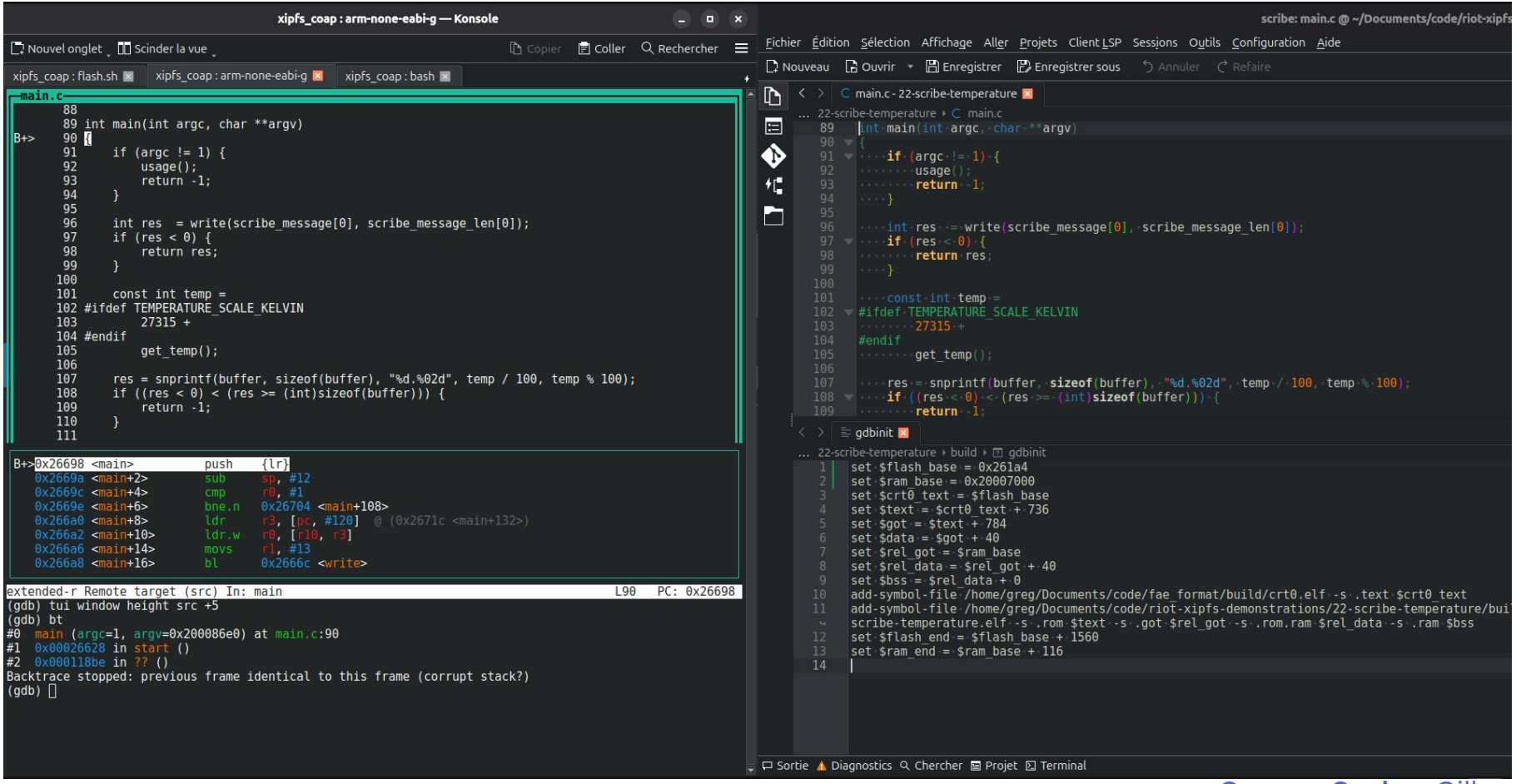
MPU_CTRL				
PRIVDEFENA	HFNIEMA	ENABLE	Handlers	Other privileged software
0	0	0	- Default memory map disabled - MPU disabled	- Default memory map disabled - MPU disabled
0	0	1	- Default memory map disabled - MPU enabled - Accesses : ?	- Default memory map disabled - MPU enabled - Using MPU
0	1	0	UNPREDICTABLE	UNPREDICTABLE
0	1	1	- Default memory map disabled - Using MPU	- Default memory map disabled - Using MPU
1	0	0	- Default memory map enabled - Using default memory map	- Default memory map enabled - Using default memory map
1	0	1	- Default memory map enabled - Using default memory map	- Default memory map enabled - Using MPU
1	1	0	UNPREDICTABLE	UNPREDICTABLE
1	1	1	- Default memory map enabled - Using MPU	- Default memory map enabled - Using MPU

RIOT XiPFS →

RIOT mpu_configure

Université de Lille

Technical details : Debugging FAE files



Work In Progress

- **XIPRFS (Rust)**
  - git repository made public.
- **MPU regions optimization**
  - for executables RAM and stack.
  - Shared memory with the free'd region ?
  - Virtual memory regions thanks to dynamic reconfiguration on memory errors ?
- **Shared API enhancement**
  - Runtime verification between in-kernel features set and applications requirements *(XiPFS)*
  - VFS functions
    - Finer mutex lock and extra checks mechanism ?
  - errno support
  - Rust related functions *(custom print)*
- **Executables lifecycle**
  - Tracking and automatic closing of still open file descriptors on executables exit. *(XiPFS)*
- **FAE format/XiPFS**
  - FAE file format : add one field with 3 possible values {CRT0 | Rust startup | Own startup sequence}
  - CRT0 and Rust Startup factorized into XiPFS

Future works

- Security policy in syscalls
- Resources access control according to users and groups permissions.
- Deployment of FAE files declaring their own CoAP resources ?
- Concurrent execution of multiple FAE files ?
- *And other ideas not yet fully matured...*