Trusted Firmware-M

FP Support in TF-M

Feder Liang

Apr 15 2021

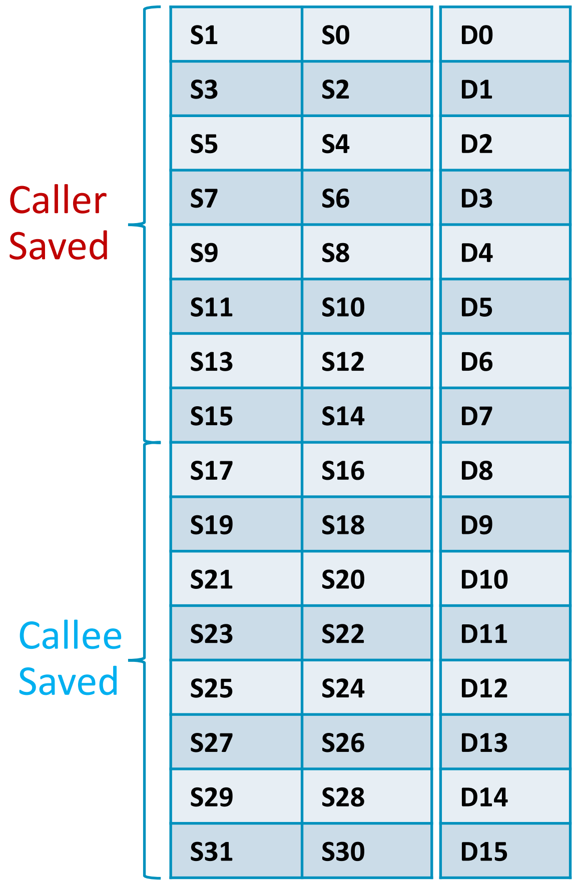
Floating-point unit (FPU)

- Floating-point unit (FPU) is the hardware unit integrated in the processor which can be used to accelerate the calculations of Floating-point numbers.
- Floating-point calculations (arithmetic operation, compare, convert and load/store) require a lot of resources.
- With FPU, floating-point operations are entirely done by hardware in a single cycle, for most of the instructions.

Floating-point Extension

- The Arm architecture provides high-performance and high-efficiency hardware support for floating-point operation.
- Arm floating point technology is fully IEEE-754 compliant with full software library support.
- Optional FPU on Cortex-M
  - Coprocessors 10 and 11 support the extension.
- This document is based on Armv8.0-M and later.

FP Registers



Caller Saved	S1	S0	D0
	S3	S2	D1
	S5	S4	D2
	S7	S6	D3
	S9	S8	D4
	S11	S10	D5
	S13	S12	D6
	S15	S14	D7
Callee Saved	S17	S16	D8
	S19	S18	D9
	S21	S20	D10
	S23	S22	D11
	S25	S24	D12
	S27	S26	D13
	S29	S28	D14
	S31	S30	D15

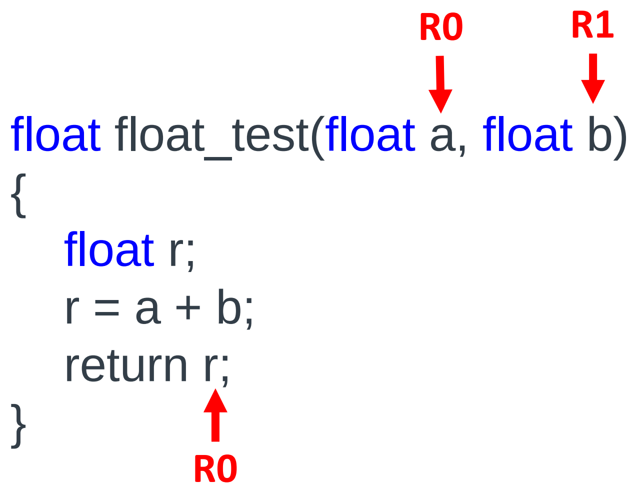
Register	Name	Description
FP extension Registers	32bit S0–S31 (64bit alias D0–D15)	Procedure Call Standard for the Arm® Architecture (AAPCS ABI) definitions: • S0–S15 are caller save registers • S16–S31 are callee save registers. Just like [R0 – R3, R12] vs [R4 – R11].
FP extension System register	FPSCR	Floating-point Status and Control Register
SCB registers (Address-mapped registers)	FPCCR	Floating Point Context Control Register
	FPCAR	Floating Point Context Address Register
	FPDSCR	Floating Point Default Status Control Register
	MVFR0, MVFR1, MVFR2	Media and FP Feature Register 0, 1, 2

FP options

Normally, there are three options provided by compiler and linker for FP support in imple mentation.

Software FP

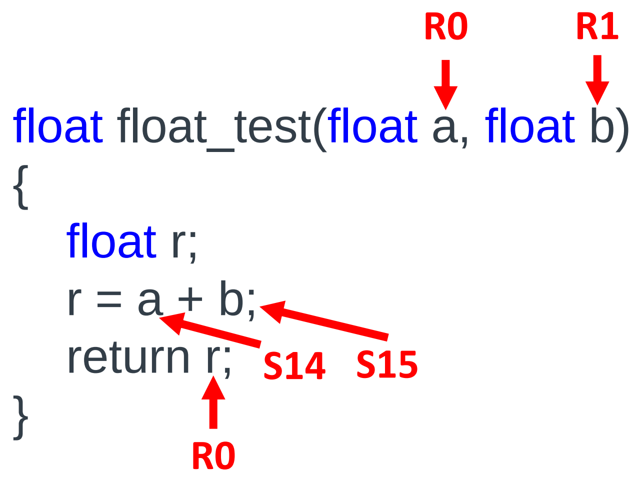
- Software library functions for floating-point operations and softwar e floating-point linkage.



```
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;
}
```

Hybrid FP

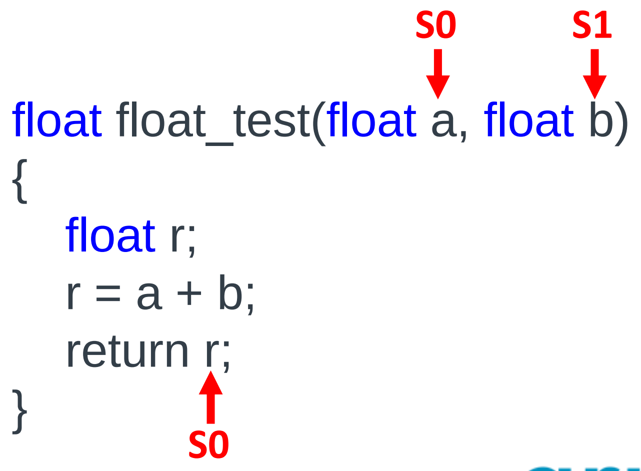
- Hardware floating-point instructions and softwa re floating-point linkage.



```
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;
}
```

Hardware FP

- Hardware floating-point instructions and hardw are floating-point linkage.



```
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;
}
```

Assembly Code - Software FP option

(GNU Tools for Arm Embedded Processors 7-2018-q2-update)

Code

R0 R1

```
A.
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;
}
```

R0

```
B.
float a = 0.375;
float b = 0.5;
bool flag = false;
if (float_test(a, b) > 0.0) {
    flag = true;
} else {
    flag = false;
}
```

Function

```
0x1008CD52 : PUSH {r7,lr}
0x1008CD54 : SUB sp,sp,#0x10
0x1008CD56 : ADD r7,sp,#0
0x1008CD58 : STR r0,[r7,#4]
0x1008CD5A : STR r1,[r7,#0]
0x1008CD5C : LDR r1,[r7,#0]
0x1008CD5E : LDR r0,[r7,#4]
0x1008CD60 : BL __addsf3 ; 0x10080D5C
0x1008CD64 : MOV r3,r0
0x1008CD66 : STR r3,[r7,#0xc]
0x1008CD68 : LDR r3,[r7,#0xc]
0x1008CD6A : MOV r0,r3
0x1008CD6C : ADDS r7,r7,#0x10
0x1008CD6E : MOV sp,r7
0x1008CD70 : POP {r7,pc}
```

ABI

```
0x1008CD88 : STR r3,[r7,#4]
0x1008CD8A : MOV.W r3,#0x3ec00000
0x1008CD8E : STR r3,[r7,#0x1c]
0x1008CD90 : MOV.W r3,#0x3f000000
0x1008CD94 : STR r3,[r7,#0x18]
0x1008CD96 : MOVS r3,#0
0x1008CD98 : STRB r3,[r7,#0x17]
0x1008CD9A : LDR r1,[r7,#0x18]
0x1008CD9C : LDR r0,[r7,#0x1c]
0x1008CD9E : BL float_test ; 0x1008CD52
0x1008CDA2 : MOV r3,r0
0x1008CDA4 : MOV.W r1,#0
0x1008CDA8 : MOV r0,r3
0x1008CDAA : BL __aeabi_fcmpgt ; 0x10081044
0x1008CDAE : MOV r3,r0
0x1008CDB0 : CMP r3,#0
0x1008CDB2 : BEQ
fpu_client_set_fp_register_test+68 ; 0x1008CDBA
0x1008CDB4 : MOVS r3,#1
0x1008CDB6 : STRB r3,[r7,#0x17]
0x1008CDB8 : B fpu_client_set_fp_register_test+72
; 0x1008CDBE
0x1008CDBA : MOVS r3,#0
0x1008CDBC : STRB r3,[r7,#0x17]
```

Assembly Code - Hybird FP option

(GNU Tools for Arm Embedded Processors 7-2018-q2-update)

Code

```
                       R0     R1
A.
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;  S14  S15
}
         R0
```

```
B.
float a = 0.375;
float b = 0.5;
bool flag = false;
if (float_test(a, b) > 0.0) {
    flag = true;
} else {
    flag = false;
}
```

Function

```
0x1008C57A : PUSH {r7}
0x1008C57C : SUB sp,sp,#0x14
0x1008C57E : ADD r7,sp,#0
0x1008C580 : STR r0,[r7,#4]
0x1008C582 : STR r1,[r7,#0]
0x1008C584 : VLDR s14,[r7,#4]
0x1008C588 : VLDR s15,[r7,#0]
0x1008C58C : VADD.F32 s15,s14,s15
0x1008C590 : VSTR s15,[r7,#0xc]
0x1008C594 : LDR r3,[r7,#0xc]
0x1008C596 : MOV r0,r3
0x1008C598 : ADDS r7,r7,#0x14
0x1008C59A : MOV sp,r7
0x1008C59C : POP.W {r7}
0x1008C5A0 : BX lr
```

ABI

```
0x1008C5BC : STR r3,[r7,#4]
0x1008C5BE : MOV.W r3,#0x3ec00000
0x1008C5C2 : STR r3,[r7,#0x1c]
0x1008C5C4 : MOV.W r3,#0x3f000000
0x1008C5C8 : STR r3,[r7,#0x18]
0x1008C5CA : MOVS r3,#0
0x1008C5CC : STRB r3,[r7,#0x17]
0x1008C5CE : LDR r1,[r7,#0x18]
0x1008C5D0 : LDR r0,[r7,#0x1c]
0x1008C5D2 : BL float_test ; 0x1008C57A
0x1008C5D6 : VMOV s15,r0
0x1008C5DA : VCMPE.F32 s15,#0.0
0x1008C5DE : VMRS APSR_nzcv,FPSCR
0x1008C5E2 : BLE
fpu_client_set_fp_register_test+64 ; 0x1008C5EA
0x1008C5E4 : MOVS r3,#1
0x1008C5E6 : STRB r3,[r7,#0x17]
0x1008C5E8 : B fpu_client_set_fp_register_test+68
; 0x1008C5EE
0x1008C5EA : MOVS r3,#0
0x1008C5EC : STRB r3,[r7,#0x17]
```

Assembly Code - Hardware FP option

(GNU Tools for Arm Embedded Processors 7-2018-q2-update)

Code

```
                          S0       S1
A.                        ↓        ↓
float float_test(float a, float b)
{
    float r;
    r = a + b;
    return r;
}          ↑
           S0

B.
float a = 0.375;
float b = 0.5;
bool flag = false;
if (float_test(a, b) > 0.0) {
    flag = true;
} else {
    flag = false;
}
```

Function

```
0x1008C790 : PUSH {r7}
0x1008C792 : SUB sp,sp,#0x14
0x1008C794 : ADD r7,sp,#0
0x1008C796 : VSTR s0,[r7,#4]
0x1008C79A : VSTR s1,[r7,#0]
0x1008C79E : VLDR s14,[r7,#4]
0x1008C7A2 : VLDR s15,[r7,#0]
0x1008C7A6 : VADD.F32 s15,s14,s15
0x1008C7AA : VSTR s15,[r7,#0xc]
0x1008C7AE : LDR r3,[r7,#0xc]
0x1008C7B0 : VMOV s15,r3
0x1008C7B4 : VMOV.F32 s0,s15
0x1008C7B8 : ADDS r7,r7,#0x14
0x1008C7BA : MOV sp,r7
0x1008C7BC : POP.W {r7}
0x1008C7C0 : BX lr
```

ABI

```
0x1008C82E : STR r3,[r7,#4]
0x1008C830 : MOV.W r3,#0x3ec00000
0x1008C834 : STR r3,[r7,#0x1c]
0x1008C836 : MOV.W r3,#0x3f000000
0x1008C83A : STR r3,[r7,#0x18]
0x1008C83C : MOVS r3,#0
0x1008C83E : STRB r3,[r7,#0x17]
0x1008C840 : VLDR s1,[r7,#0x18]
0x1008C844 : VLDR s0,[r7,#0x1c]
0x1008C848 : BL float_test ; 0x1008C790
0x1008C84C : VMOV.F32 s15,s0
0x1008C850 : VCMPE.F32 s15,#0.0
0x1008C854 : VMRS APSR_nzcv,FPSCR
0x1008C858 : BLE
fpu_client_set_fp_register_test+68 ;
0x1008C860
0x1008C85A : MOVS r3,#1
0x1008C85C : STRB r3,[r7,#0x17]
0x1008C85E : B
fpu_client_set_fp_register_test+72 ;
0x1008C864
0x1008C860 : MOVS r3,#0
0x1008C862 : STRB r3,[r7,#0x17]
```

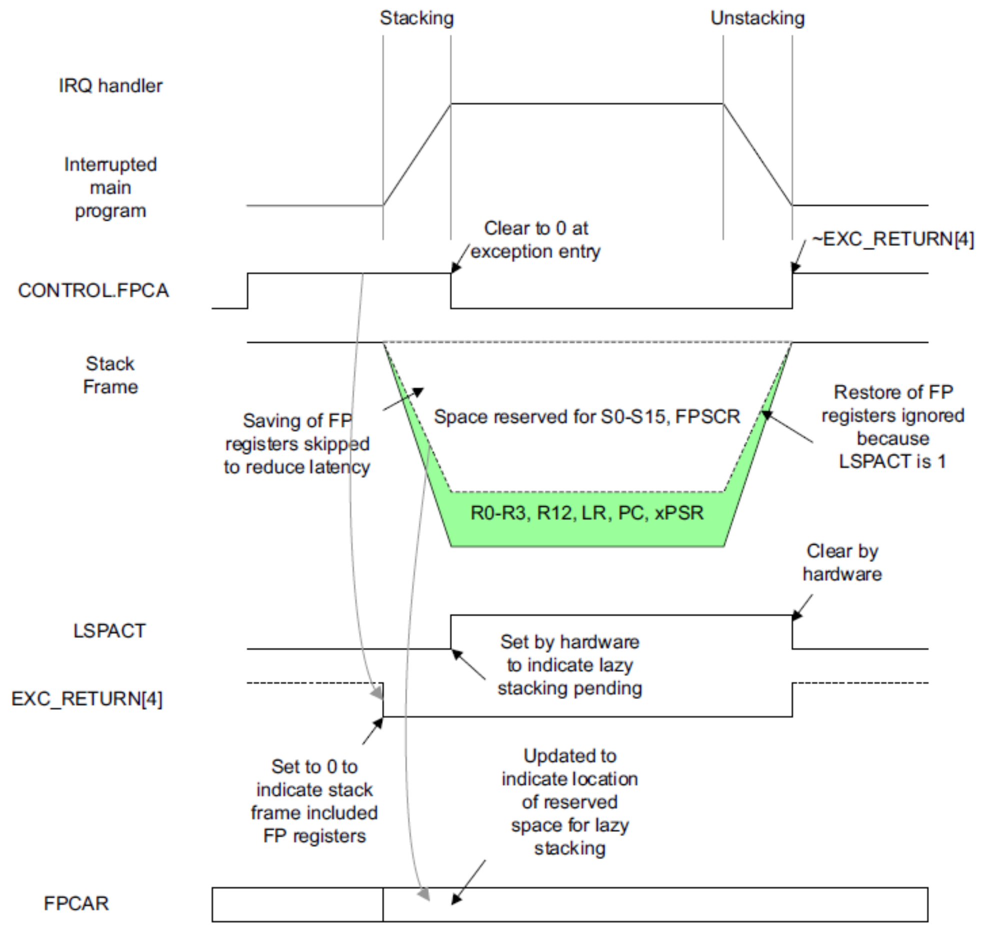
FP Usage in General Cortex-M System

- Enabled FPU extension
  - Specifying privilege of CP10 and CP11 in Coprocessor Access Control Register (CPACR).
- General exceptions won’t affect thread FP context.
  - Exception entry
    - S0–S15, FPSCR are stacked by hardware while exception happens if FP is active (CONTROL.FPCA == 1).
  - Exception Handler
    - FP usage can use S0-S15 for they are already stacked. Save S16–S31 before use and restore them before exception return.
  - Exception return
    - S0–S15, FPSCR are restored by hardware from the stacked content.
    - Clear Floating-point caller saved registers on exception return (if FPCCR.CLRONRET = 1).
- Scheduler
  - During context save for current thread.
    - Save FP callee registers (S16-S31) to current thread’s stack.
  - After context restore for next thread
    - Restore FP callee registers (S16-S31) from next thread’s stack.

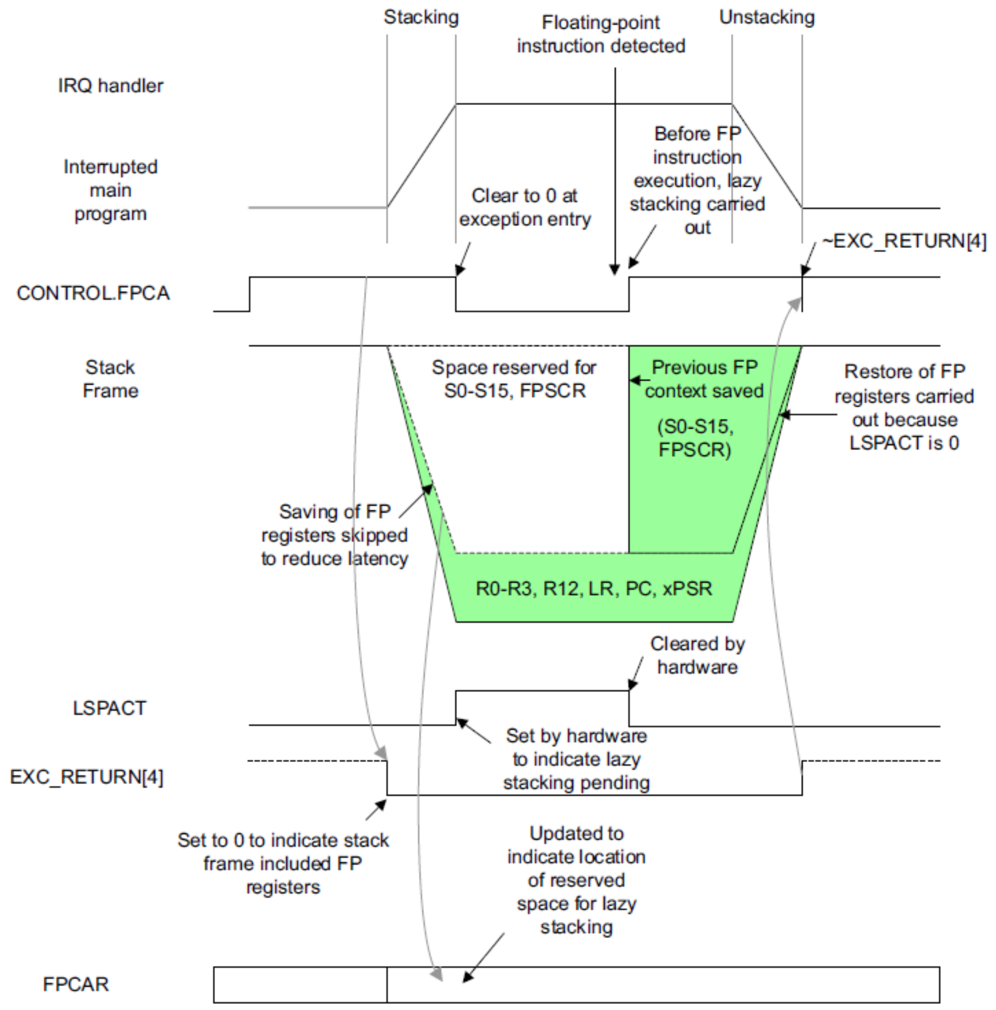
Lazy FP Stacking

- When an application has previously used the FPU, Floating Point Context Address (FPCAR) is set by hardware.
- If an exception occurs, processor reserves extra space in the stack frame for the S0-S15 registers and FPSCR. However, the actual stacking of those registers does not take place.
- If the exception handler does not use the FPU, when returning from the exception, unstackling of the floating-point registers is ignored.
- If the exception handler uses the FPU at some stage, the processor is stalled when the first floating-point instruction take place, while the floating-point registers, that is, S0-S15 registers and FPSCR, are pushed to the stack at the address which stored in FPCAR.
- The program execution then continues. At the end of the exception handler, unstacks FP registers accordingly.

Lazy FP Stacking - Diagram



Stacking
Unstacking
IRQ handler
Interrupted main program
Clear to 0 at exception entry
~EXC_RETURN[4]
CONTROL.FPCA
Stack Frame
Saving of FP registers skipped to reduce latency
Space reserved for S0-S15, FPSCR
Restore of FP registers ignored because LSPACT is 1
R0-R3, R12, LR, PC, xPSR
Clear by hardware
LSPACT
Set by hardware to indicate lazy stacking pending
EXC_RETURN[4]
Set to 0 to indicate stack frame included FP registers
Updated to indicate location of reserved space for lazy stacking
FPCAR



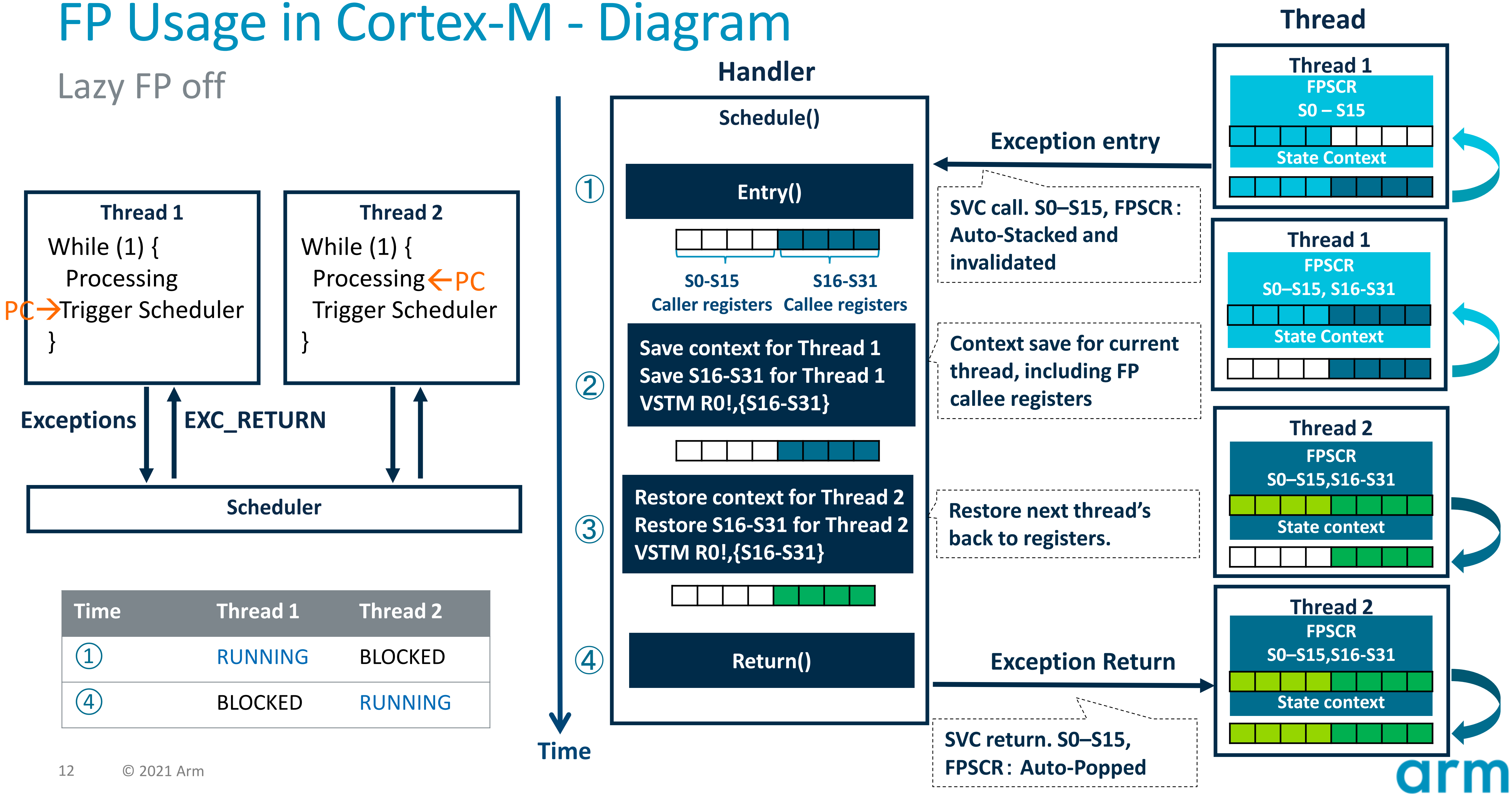
Stacking
Floating-point instruction detected
Unstacking
IRQ handler
Interrupted main program
Clear to 0 at exception entry
Before FP instruction execution, lazy stacking carried out
~EXC_RETURN[4]
CONTROL.FPCA
Stack Frame
Space reserved for S0-S15, FPSCR
Previous FP context saved (S0-S15, FPSCR)
Restore of FP registers carried out because LSPACT is 0
Saving of FP registers skipped to reduce latency
R0-R3, R12, LR, PC, xPSR
Cleared by hardware
LSPACT
Set by hardware to indicate lazy stacking pending
EXC_RETURN[4]
Set to 0 to indicate stack frame included FP registers
Updated to indicate location of reserved space for lazy stacking
FPCAR

No FP usage in Exception handler

FP usage in Exception handler

FP Usage in Cortex-M - Diagram

Lazy FP off

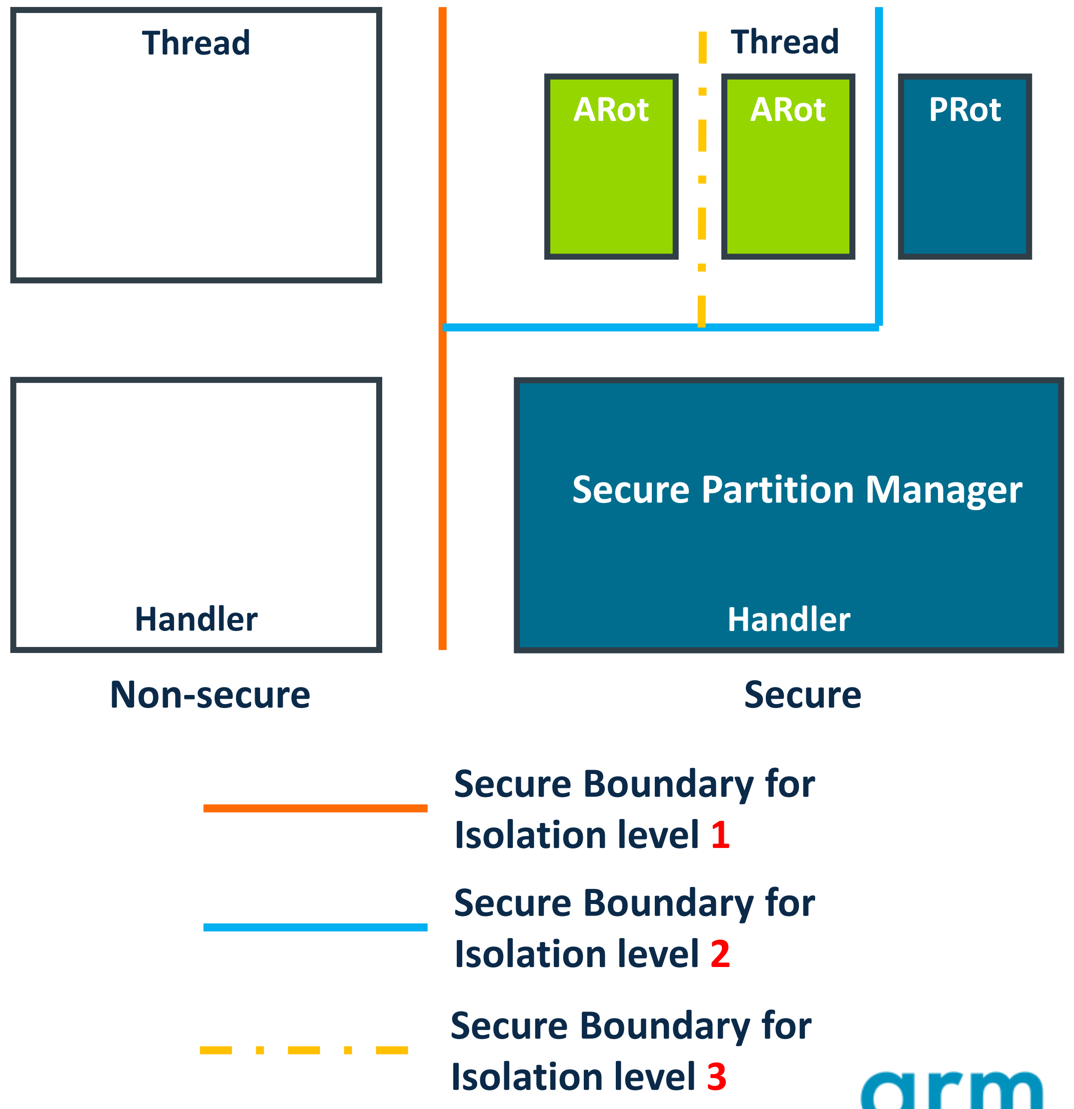


Time	Thread 1	Thread 2
①	RUNNING	BLOCKED
④	BLOCKED	RUNNING

FP Support in TF-M

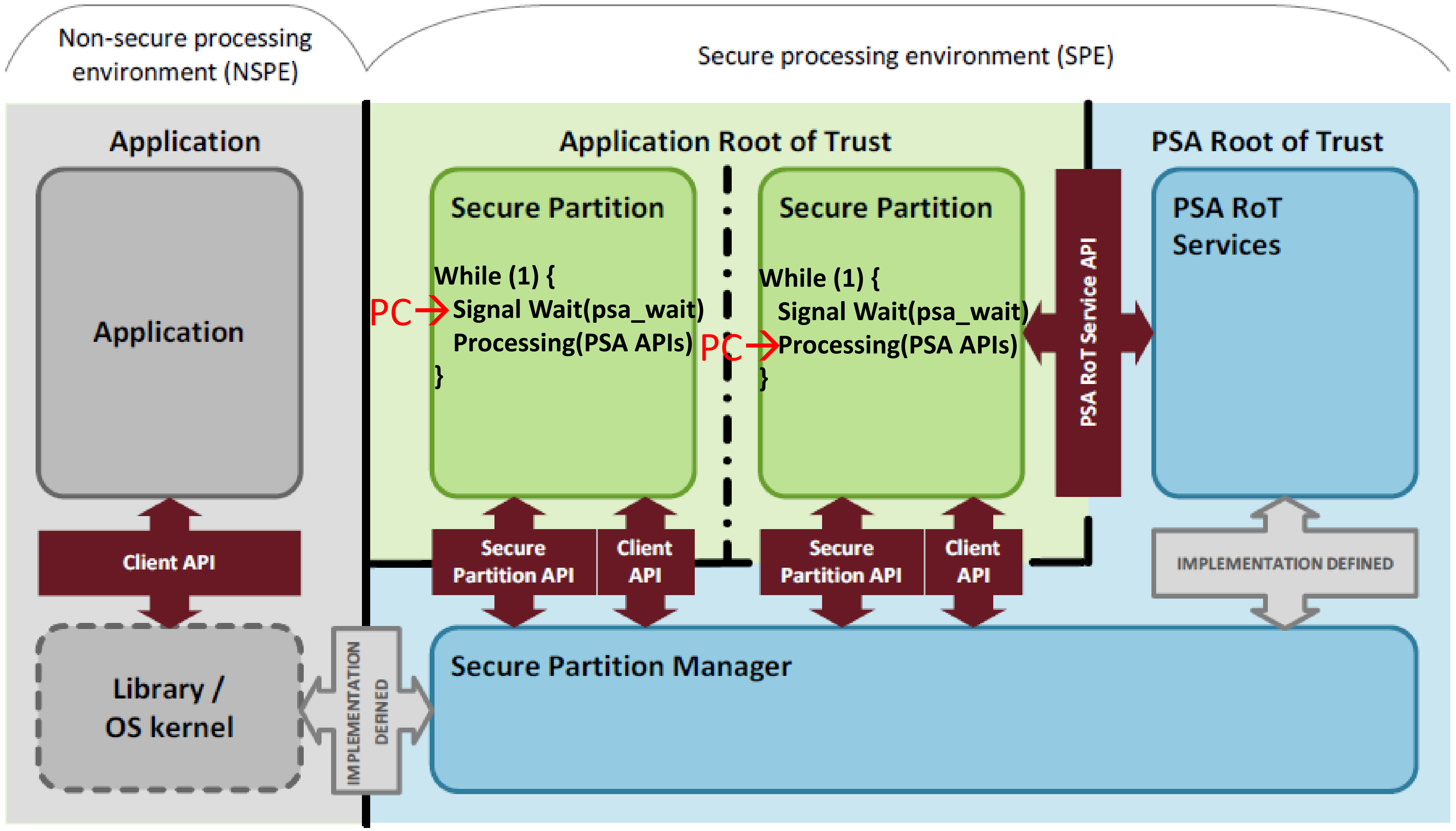
- In a secure system, to avoid information leakage, FP context needs to be invalidated while mode/state changing.

- FP context:
  - FP registers (S0 – S31).
  - Floating-point Status and Control Register (FPSCR).
  - Floating-Point Context Address Register (FPCAR) if lazy FP stacking is ON.

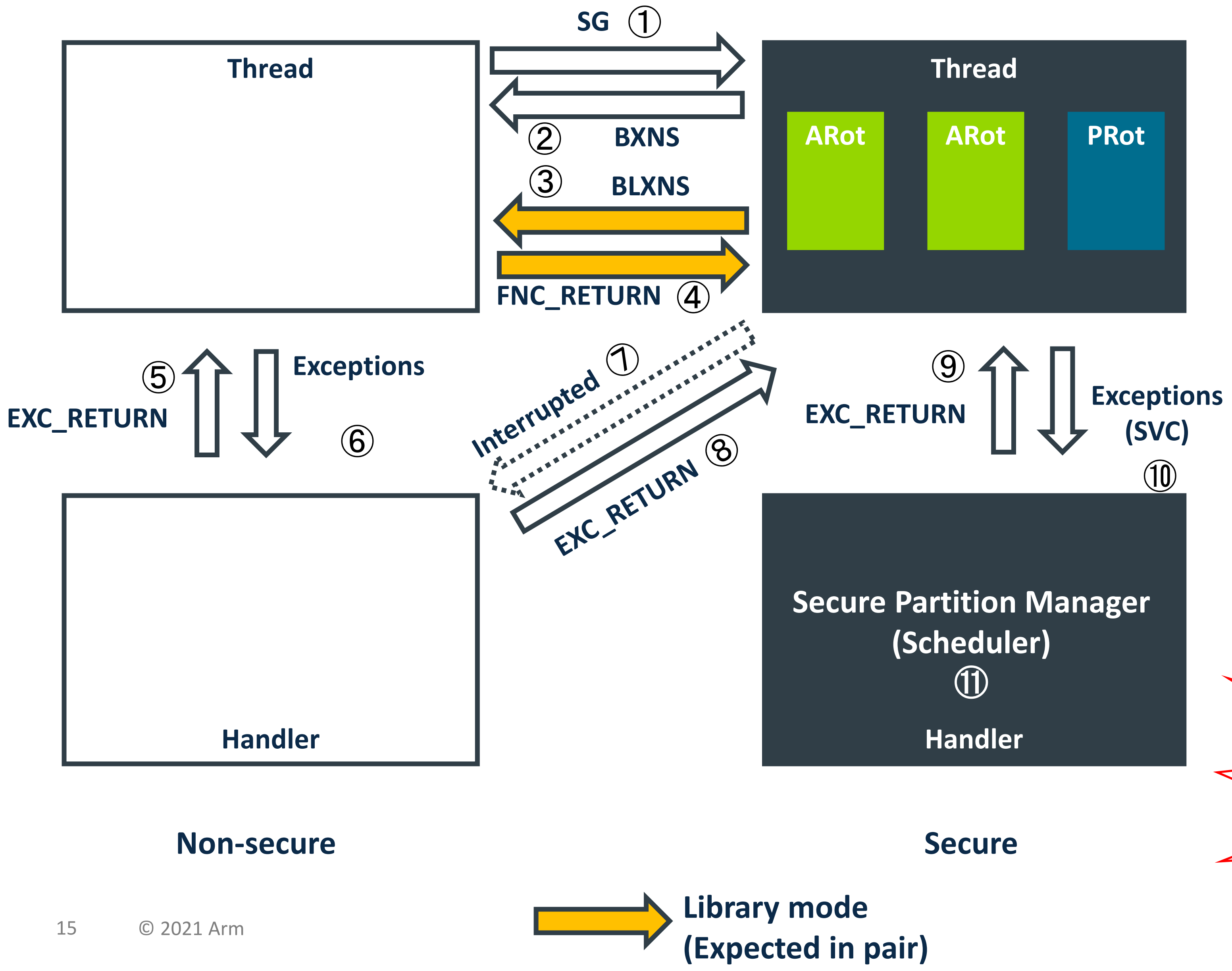

Thread
Handler
Non-secure
Thread
ARot
ARot
PRot
Secure Partition Manager
Handler
Secure
Secure Boundary for Isolation level 1
Secure Boundary for Isolation level 2
Secure Boundary for Isolation level 3

Secure Threads in TF-M

- Secure thread is **BLOCKED** after exception entry of secure API call.
- Secure thread becomes **RUNNING** after exception return from secure APIs call.
- Scheduling runs in PendSV handler, only triggered by secure APIs.



Data Flow in TF-M



① Non-secure function call to secure.

②, ③ Branch and Exchange to Non-secure.

④ Function return from non-secure.

⑤, ⑥ Exception for non-secure.

⑦ Secure thread can be interrupted by non-secure exceptions.

⑧ FP context is auto popped during exception return.

⑨ Secure thread call(SVC) for PSA APIs.

⑩ Exception return from Secure Partition Manager (Scheduler).

⑪Context switch fo[r t]hreads.

Information Leakage Risk for FP: ②, ③, ⑦, ⑩,⑪

Guidelines for FP Support in TF-M

- Compiler help to invalidate FP context by CMCE feature in compiler: cmse_nonsecure_entry. → **Cover risk in ②, ③**.
- Treat Floating-point registers as Secure (FPCCR.TS = 1)

  → **Protect FP context when state transition to non-secure, cover risk in ⑦.**
- Non-secure exceptions are de-prioritized (AIRCR.PRIS = 1).

  → **Secure exception cannot be interrupted by non-secure exception.**
- Clear Floating-point caller saved registers on exception return (if FPCCR.CLRONRET = 1).
- Supervisor Call (SVC) priority is set as 0 (**highest in exceptions except fault exceptions**)

  → **Secure APIs (SVC) exception cannot be interrupted by other non-faults secure exceptions, cover risk in ⑩, ⑪.**
- SPM does not touch float point calculation, but partitions may use them if necessary.

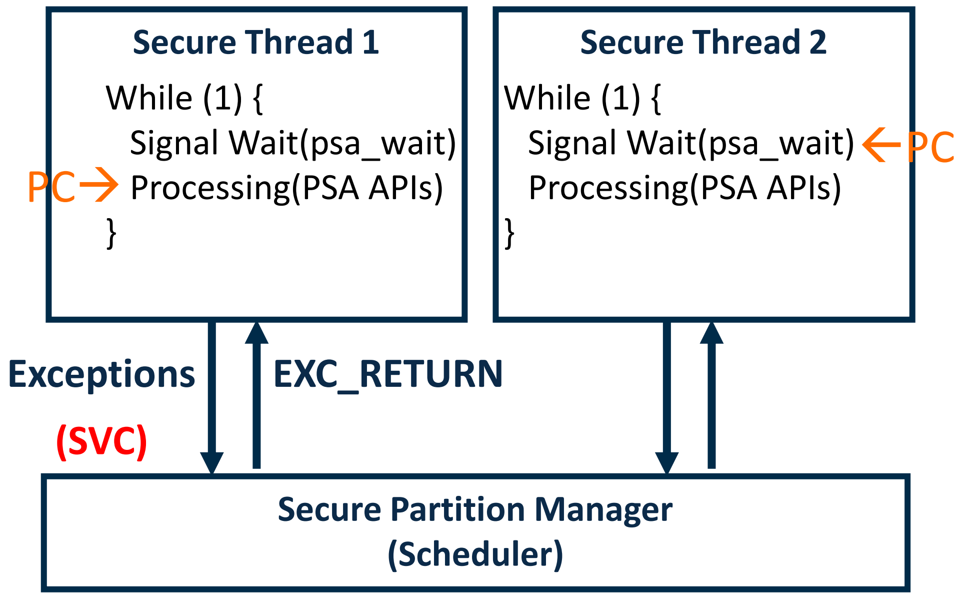
  → **Make sure no side effect caused by SPM itself.**

Actions for FP Context Protection in TF-M

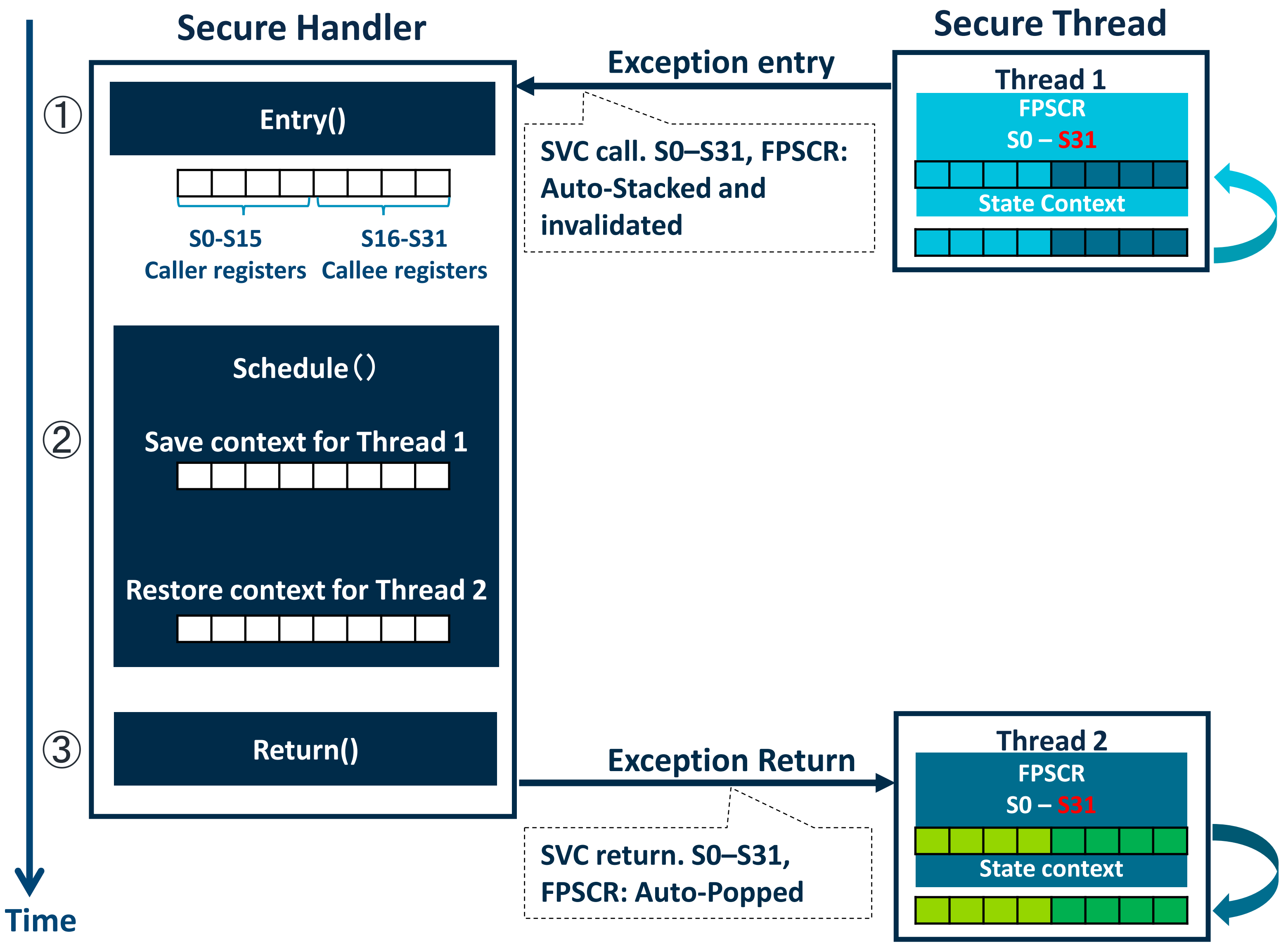
- 1. Isolation level 1, protect FP context in secure partition before switch to non-secure.
  - Function call to non-secure → FP context is invalidated by compiler.
  - Interrupt by Non-secure → FP context is saved and invalidated by hardware automatically (**FPCCR.TS = 1**).
- 2. Isolation level 2:
  - Protect FP context in PRot partition → **FP context should be saved and invalidated before switch to ARot partition.**
  - Protect FP context in secure partition before switch to non-secure → Same as item 1.
- 3. Isolation level 3:
  - Protect FP context in PRot partition → **FP context should be saved and invalidated before switch to ARot partition.**
  - Protect FP context in ARot partition → **FP context should be saved and invalidated before switch to ARot partition.**
  - Protect FP context in secure partition before switch to non-secure → Same as item 1.

FP Usage in TF-M - Diagram

Lazy FP off

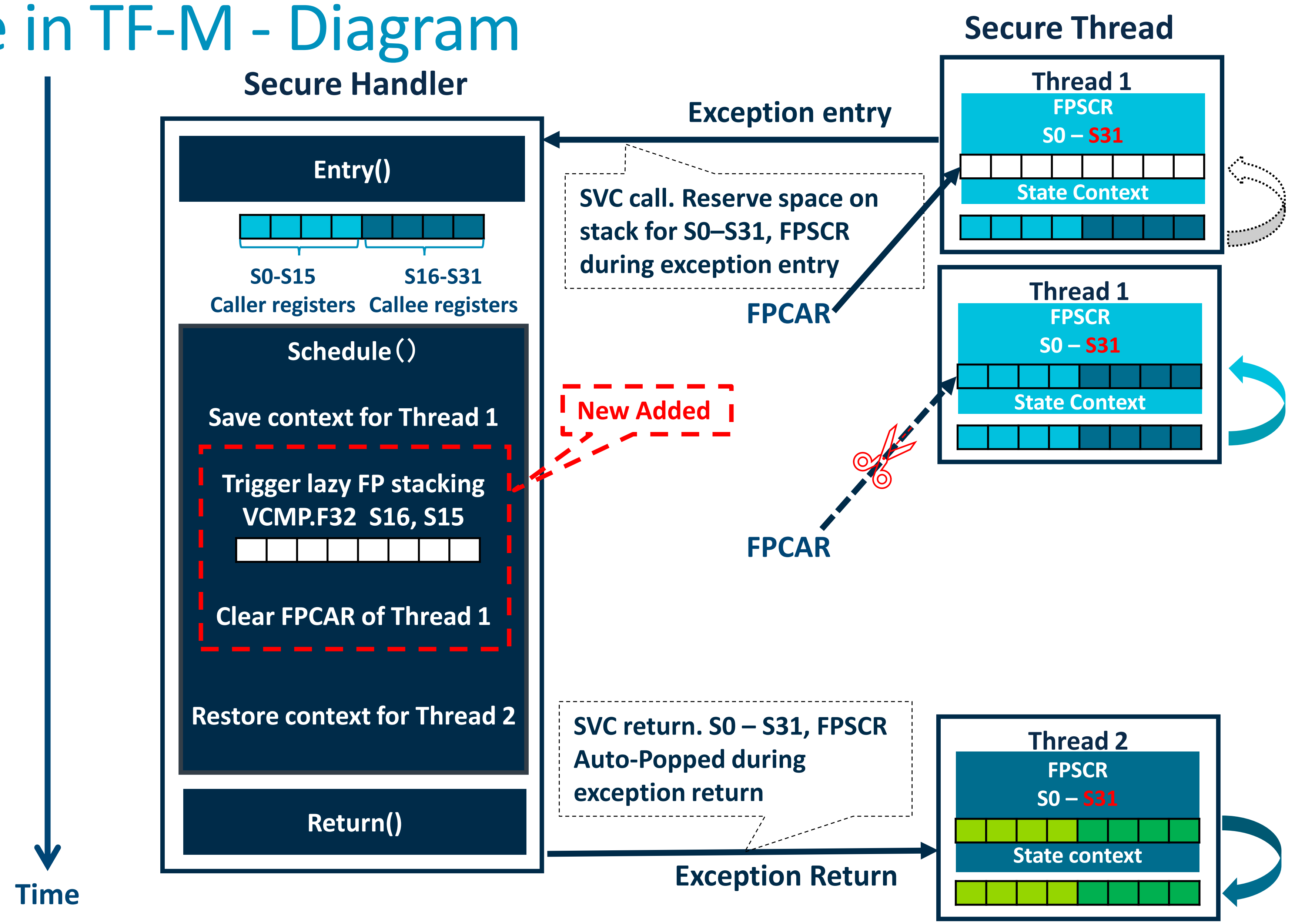


Time	Thread 1	Thread 2
①	RUNNING	BLOCKED
③	BLOCKED	RUNNING



FP Usage in TF-M - Diagram

Lazy FP on



Scheduling Logic Update

- Trigger lazy FP stacking in Scheduler before context switch to next thread.

```
PendSV_Hadler()
{
        Save state context for current thread;
        Save callee saved registers;
        /* For Isolation 2 and 3 */
        If (!isolation 1) {
                /* Lazy FP stacking enabled? */
                if (lazy fp enabled) {
                        /* Lazy FP state active? */
                        if (lazy fp state active) {
                                /* Trigger lazy fp stacking to save fp context into
                                 current thread's stack */
                                vcmp.f32  s16, s15;
                                /* Clear FPCAR of current thread */
                                Clear FP context address (FPCAR);
                        }
                }
        }
        Context switch to next thread;
        Restore callee saved registers from next thread's stack;
        Restore state context from next thread's stack;
}
```

FP Exception Handler

- FP exceptions :

Exception	Description
IXC	Inexact cumulative exception
UFC	Underflow cumulative exception
OFC	Overflow cumulative exception
DZC	Divide by Zero cumulative exception
IOC	Invalid Operation cumulative exception

- No individual mask and the enable/disable of the FPU interrupt at the interrupt controller level.
- Any exception flags (IOC, DZC, OFC, UFC, IXC) that occurs causes the associated cumulative bit in the FPSCR to be set.
- **No FP exception handler in secure world (SPM, partitions).**

Thank You

Danke

Gracias

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكرًا

ধন্যবাদ

תודה