Trusted Firmware – M Interrupt Handling

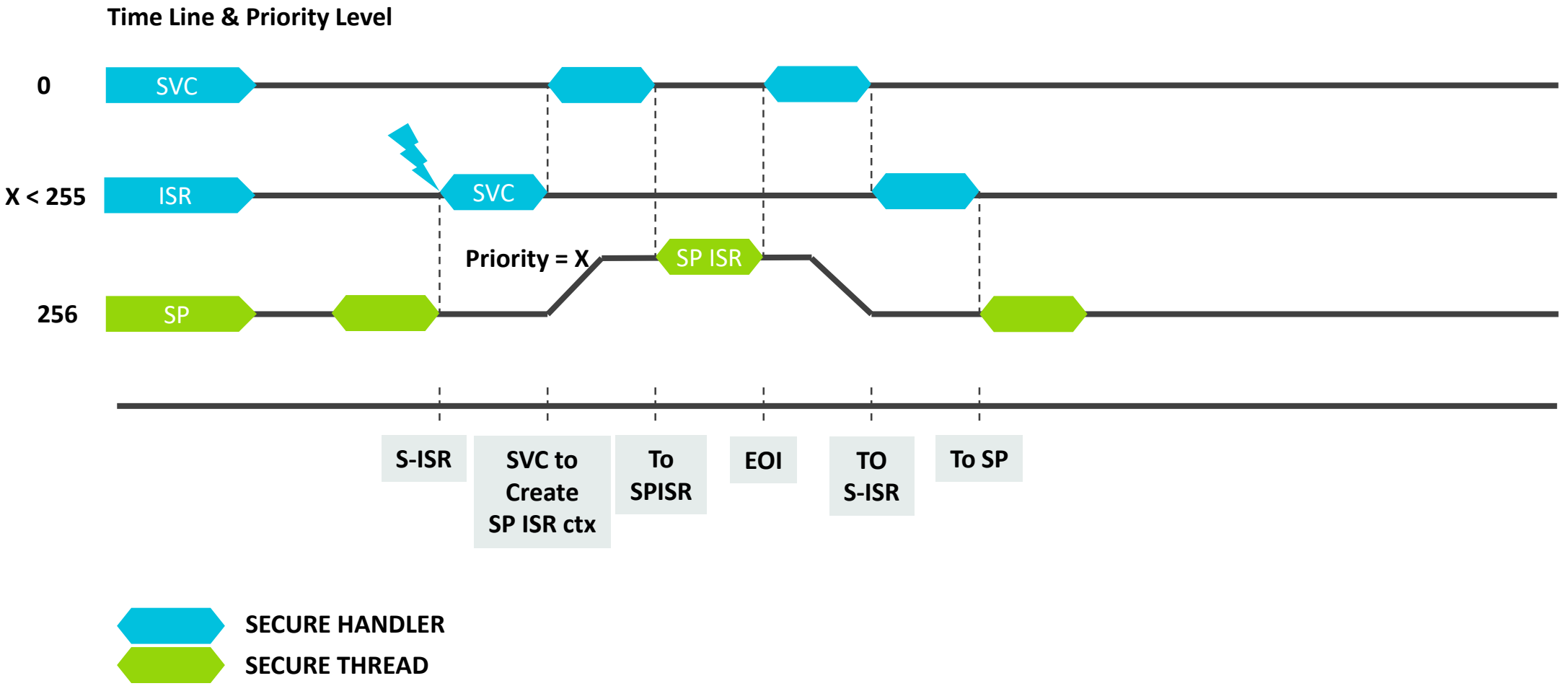
Ken Liu

Content

- Secure Interrupt Handling in library model
- Secure Interrupt Handling in IPC model
- Interactions
- IDLE processing

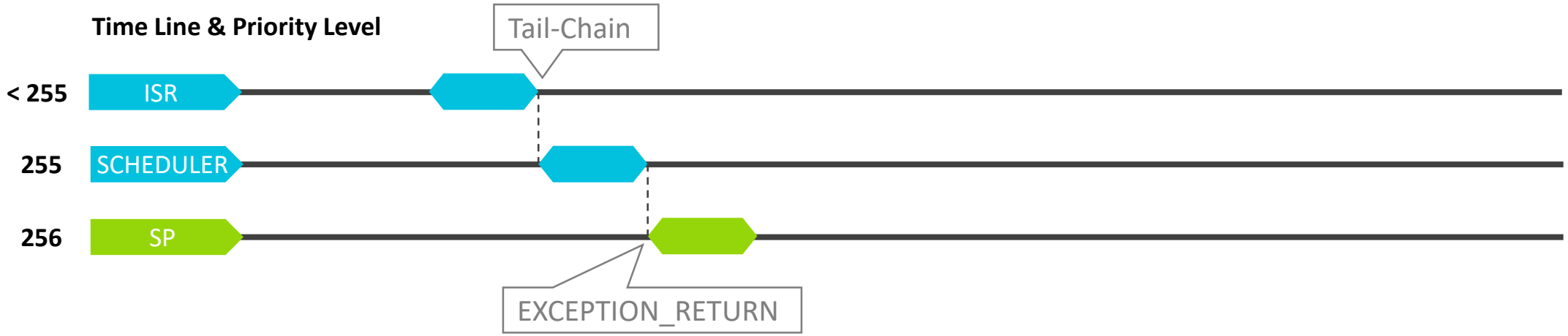
This topic is the prologue of incoming designs. Interrupt has to be related with scheduling.

Secure Interrupt Handling in library model



arm

Secure Interrupt Handling in IPC model



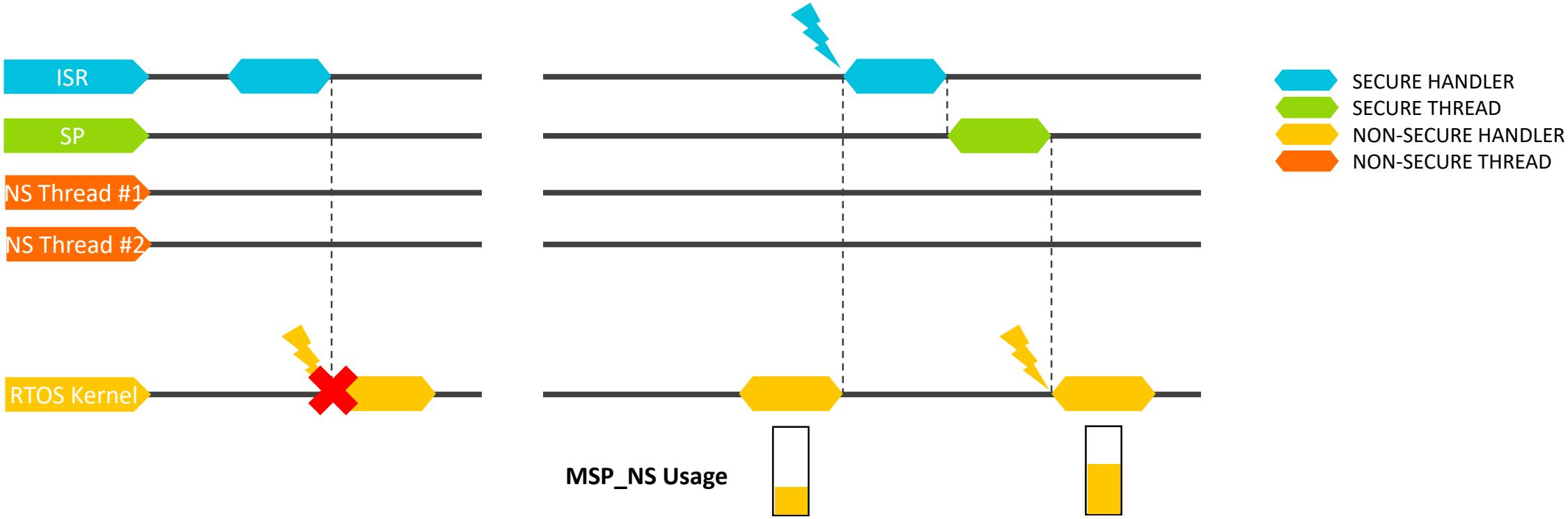

```
S_ISR()
{
   set_irq_signal(&THIS_IRQ->owner_sp, THIS_IRQ->signal);
}
```

NOTE:

1. **In further discussion the SCHEDULER in PendSV is not listed individually.**
2. **Assume the non-secure scheduler ISR/SCHEDULER is running under Handler mode.**

arm

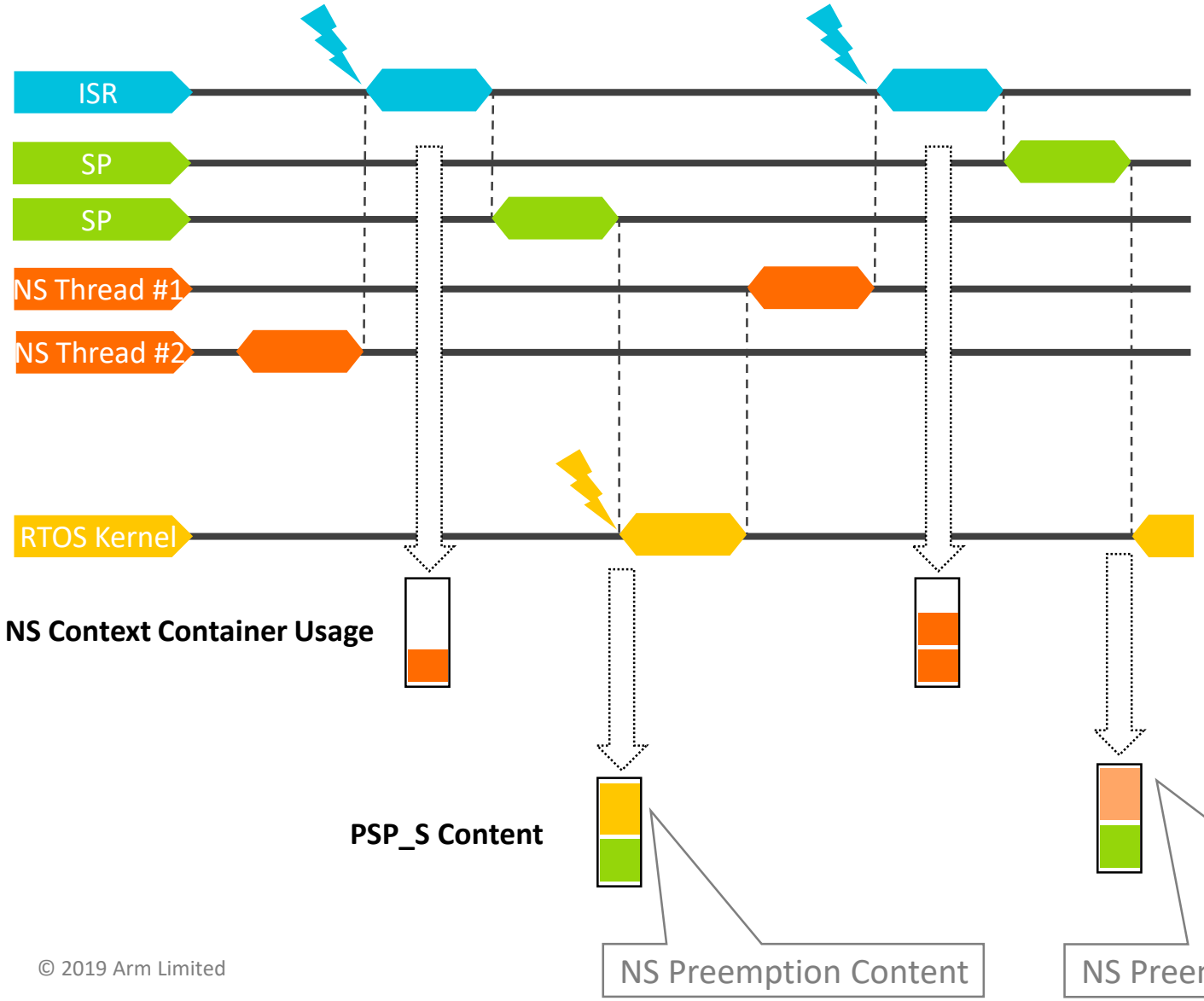
Interactions – Preempts Non-secure Handler Execution



1. **Secure Handler is not preemptable to avoid secure execution stalling.**
2. **Preempt a non-secure Handler and do scheduling may cause NS stalls.**
3. **MSP_NS keeps increasing due to un-released execution.**

arm

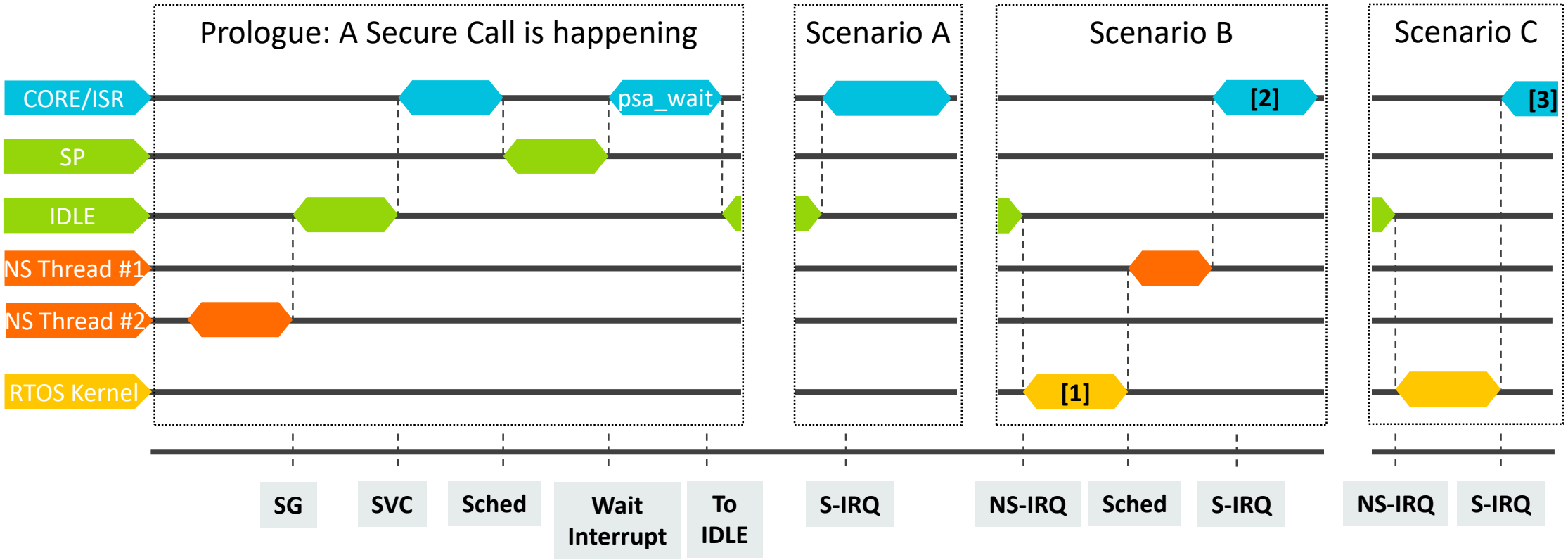
Interactions – Preempts S/NS Thread Execution



1. **May lack of NS context containers – need to allocate one context container if SPE don't know which non-secure thread is running.**
2. **Potential different NS preemption context MAY cause Faults.**

arm

Interactions – Scenarios while Secure IDLE is ongoing



1. **RTOS Kernel preempts the IDLE thread and regard the context as NS Thread #2**
2. **SPM preempts NS Thread #2 and regard the context as IDLE veneer**
3. **Scenario C described in the 'Interactions' page, do not schedule in SPE.**

arm

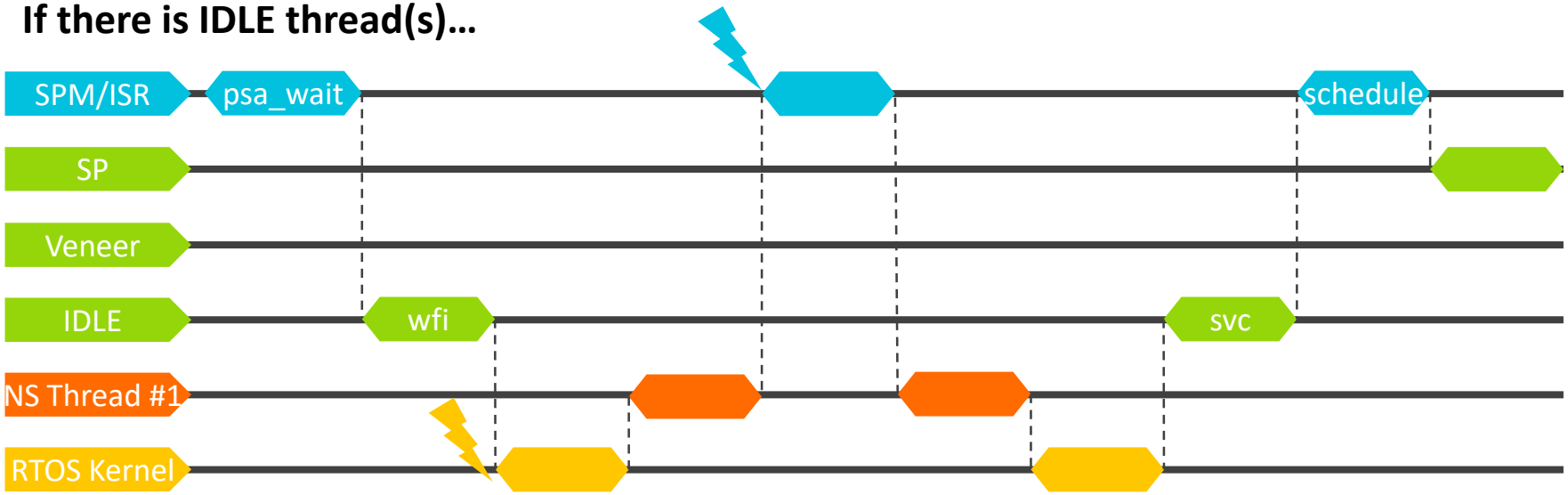
Interactions – Initial Scheduling Guidelines

Do not do scheduling while NSPE is executing.

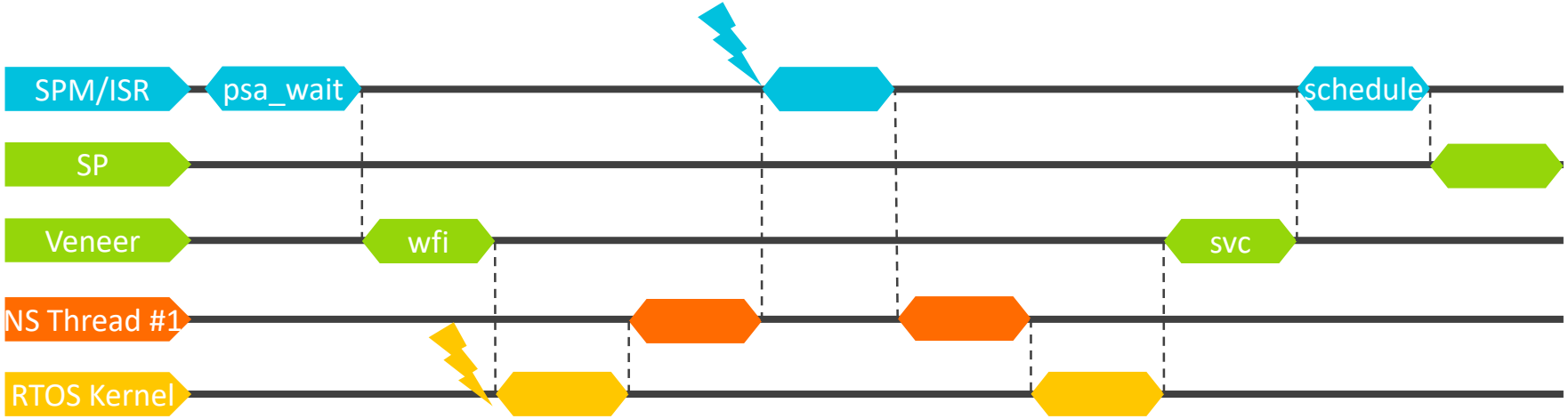
```
set_irq_signal(…)
{
   if ((EXC_RETURN & SECURE_BIT) == SECURE_BIT) {
      PendSV = 1;
   }
}
```

IDLE Processing – Option 1: `wfi` in SPE

If there is IDLE thread(s)…



Re-use veneer context as IDLE Thread to save context

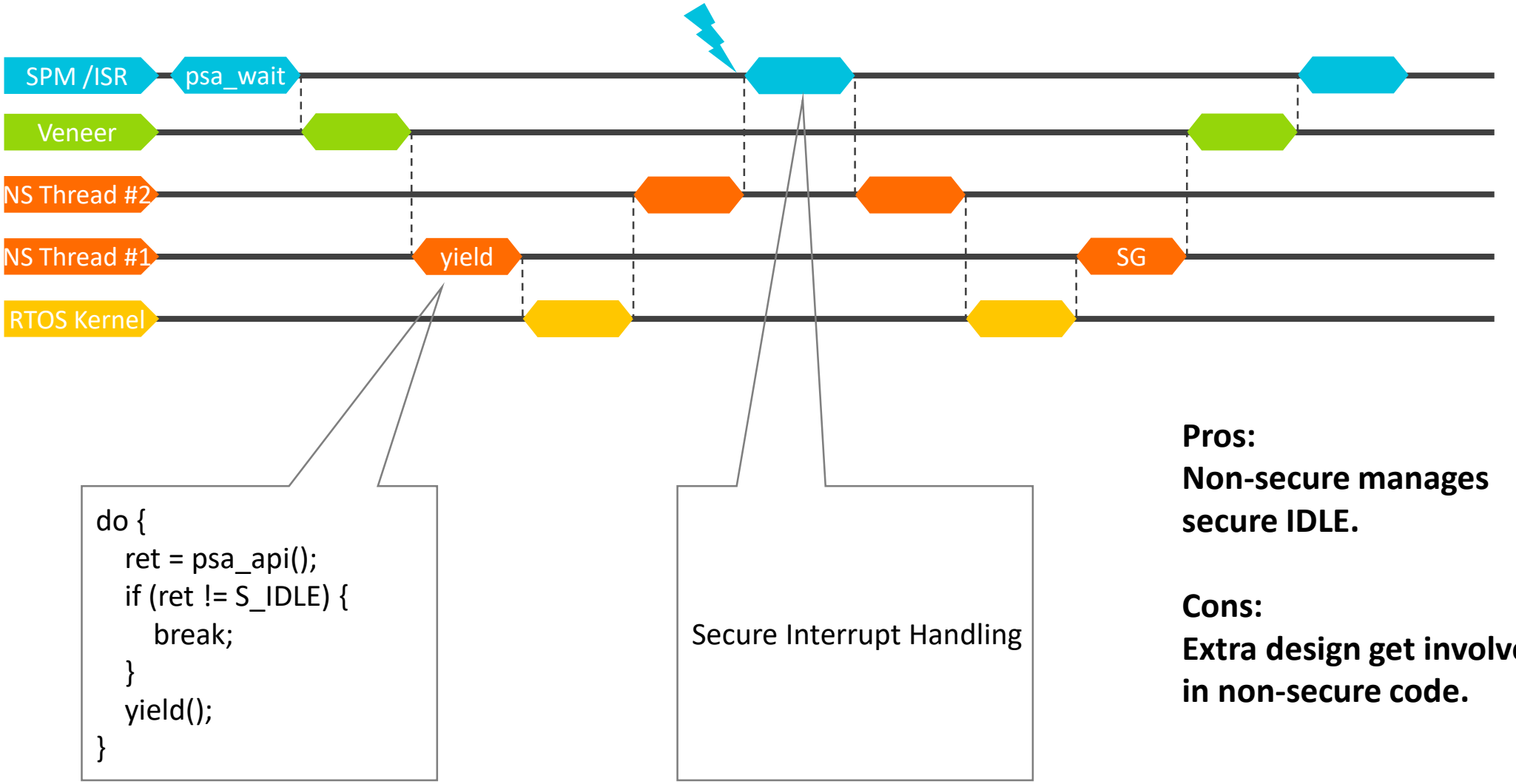


Pros:
No extra modification in NS RTOS sources.

Cons:
Execution needs to be activated by interrupt.

arm

IDLE Processing – Option 2: Return IDLE to Non-secure



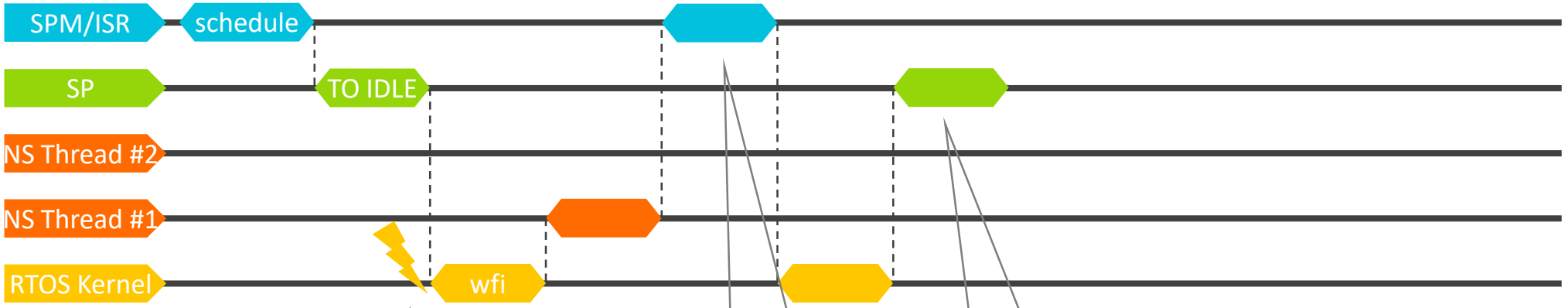
Pros:
Non-secure manages secure IDLE.

Cons:
Extra design get involved in non-secure code.

arm

IDLE Processing – Option3: Use IRQ/Event

If NS IRQ get triggered at SP…



Here preempts directly from SP since priority is the lowest. This indicates the secure execution is IDLE, non-secure can go IDLE if it also has nothing to do.

SPM can't do anything since execution belongs to non-secure now.

Have to wait non-secure scheduler back to preempted SP.

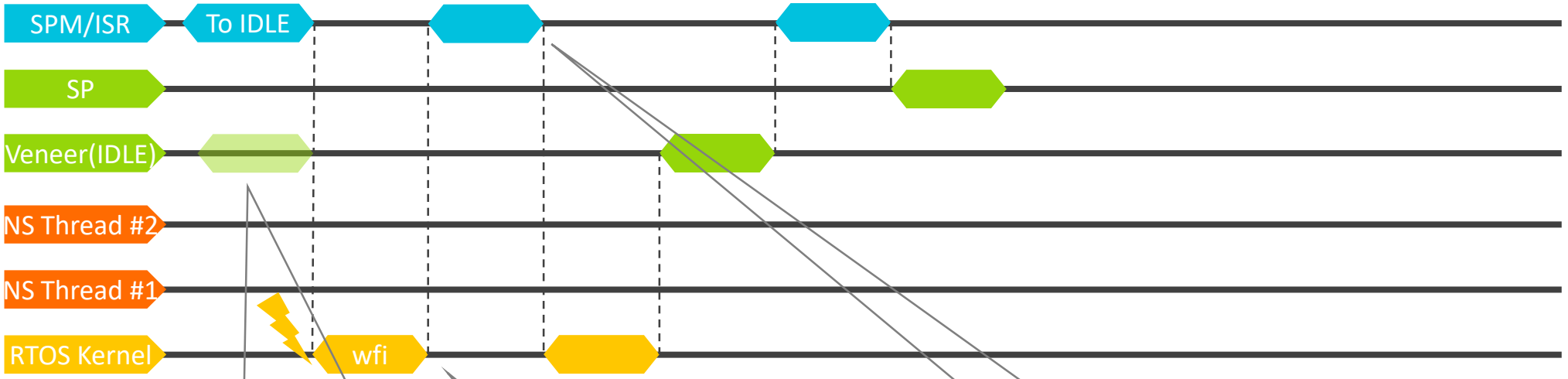
Pros:
Non-secure thread software did not aware of the code change (change in kernel part).

Cons:
Extra design get involved in non-secure scheduler mode.

arm

IDLE Processing – Option3: Use IRQ/Event

If NS IRQ get triggered at SPM…



Before Triggering IRQ, IDLE thread is the background context:

```
if (next_thread->r0 == S_IDLE) {
   trigger_ns_idle_irq();
}
```

The preempted context is the IDLE veneer.

S_ISR can’t scheduling. Tail-chain to NS Handler.

arm

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה