Secure Partition Manager Implementation Update

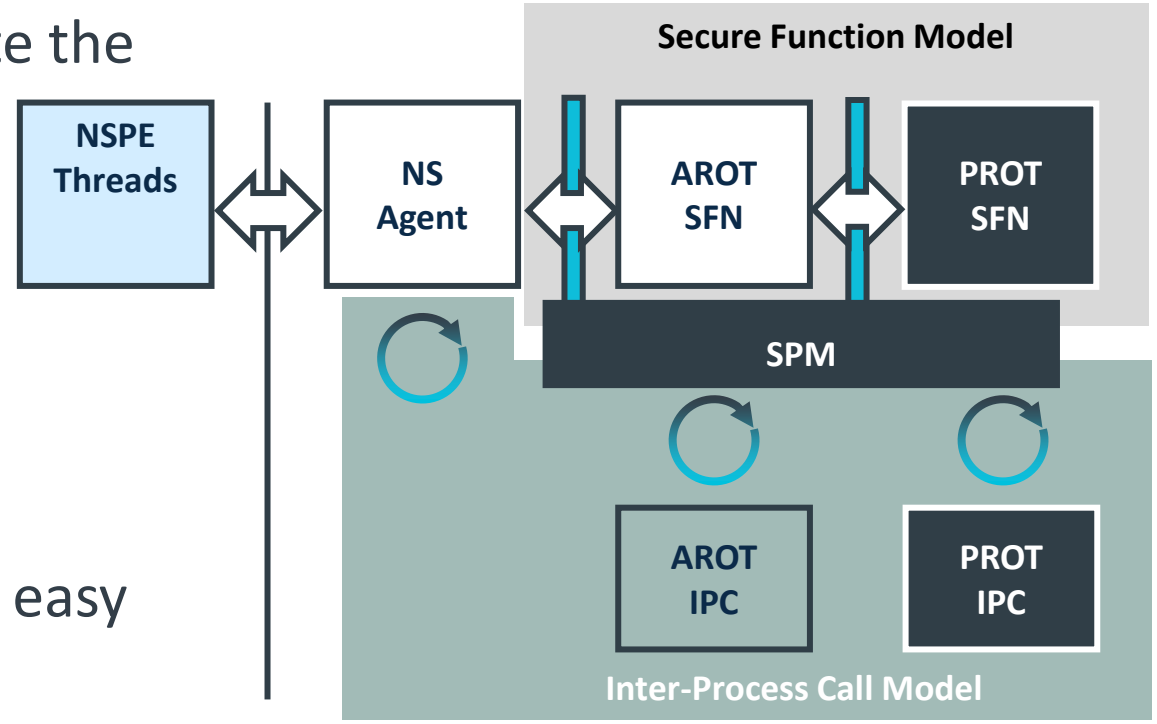
To be simple and straight

Ken Liu

Sep 11th

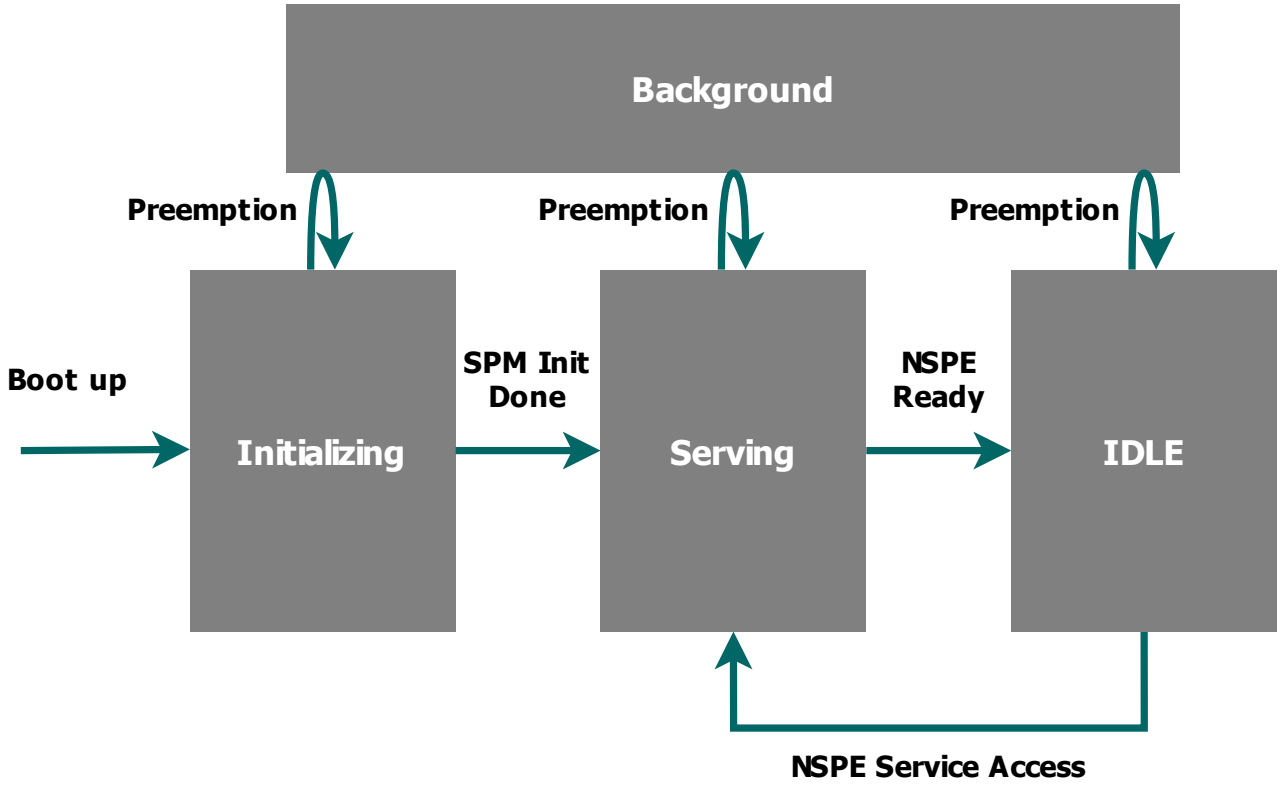
Introduction

- SPM – the FF-M item. After going through the specification (1.0 and 1.1), we can know:
  - Isolation rules, levels and boundaries.
  - SPM uses handles to represent the connection between client and services, and each access is packaged into a message bound to a handle.
  - SPM delivery messages to service under some mechanisms (1.0 with IPC, 1.1 support SFN – function call), and service replies to the client – which is implemented in IPC model already.
- And we may not be able to estimate the implementation details:
  - How Secure Function (SFN) model can be implemented?
  - Can it co-work with IPC runtime model?
  - High-level isolation level SFN possible?
  - Improve efficiency
- How can we find the answers in an easy way?



Start with the state transition

- We expect to see the answers easily by checking an overall design summary.
- Start with the runtime state can be straight - avoid involving too many items:
  - 4 states: ‘Initializing’, ‘serving’, ‘IDLE’, and ‘background’, which indicates 4 modalized parts.

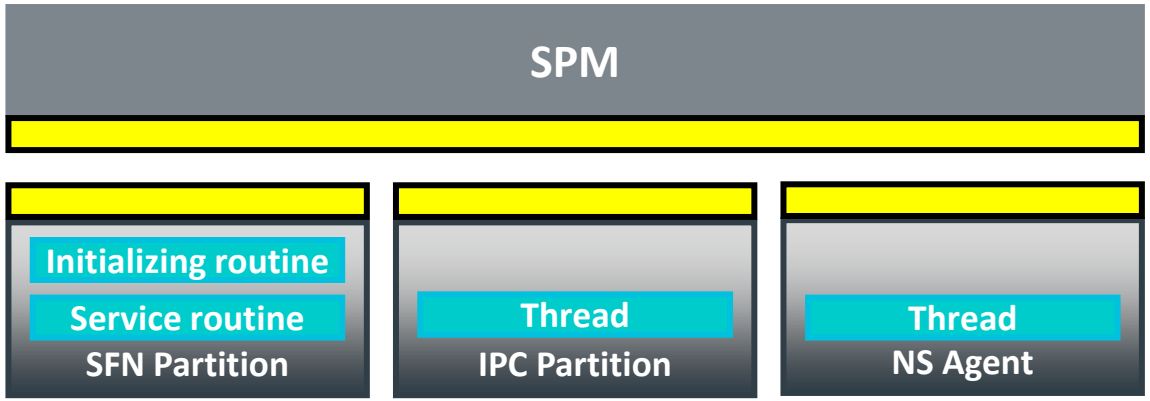


Initializing state

- State for SPM specific initializing. SPM entry get launched by platform startup code – for code re-use purpose.
  - The earlier initialization HAL- Performing extra security hardware settings – the re-used startup code may be not designed for a secure system.
  - SPM runtime setup – a internal initialization routine.
  - After setup, launch the first component and eventually go to the next state.

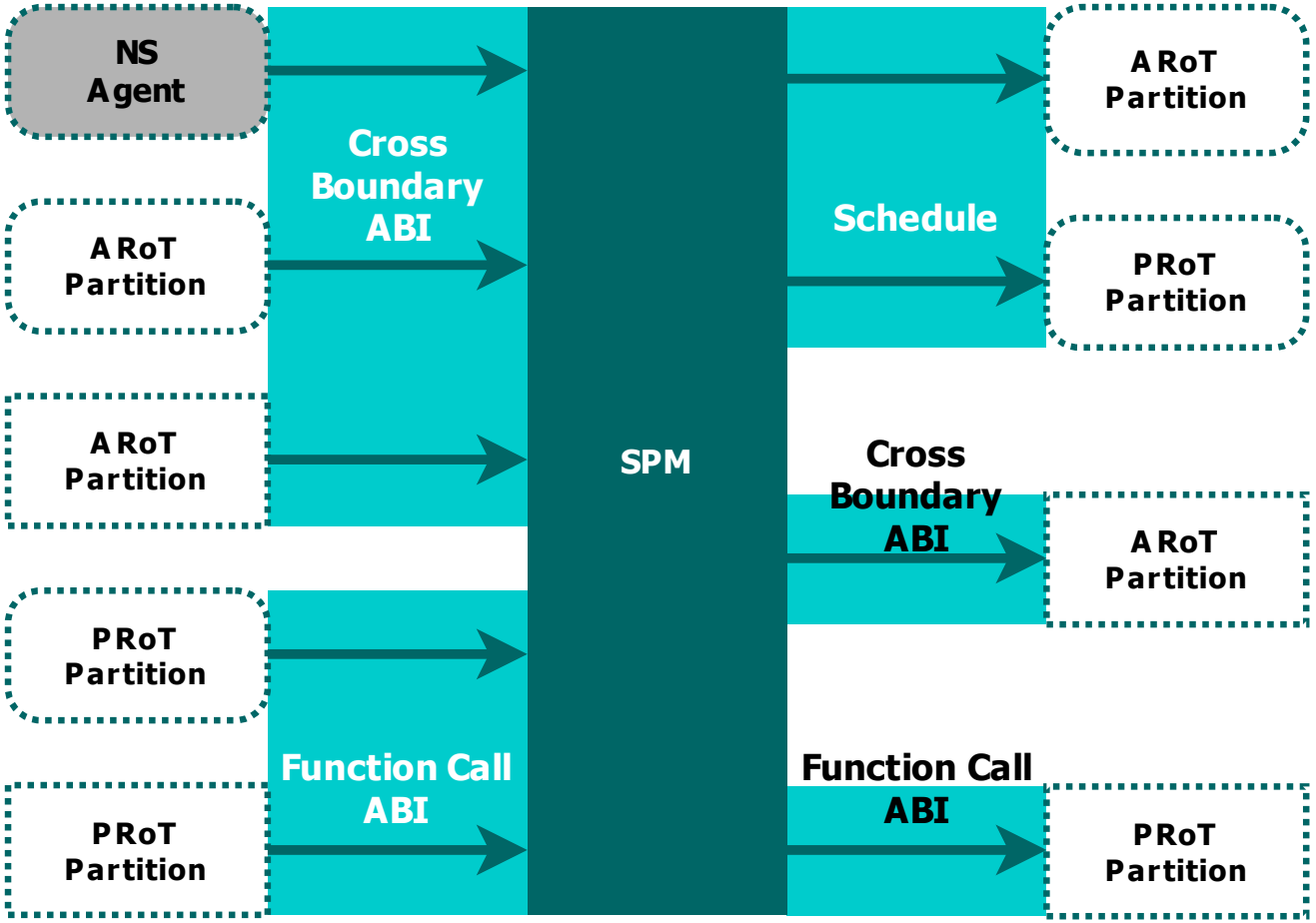
Serving state

- The main working state.
  - Runtime components under SPM management: ‘partitions’ (FF-M concept) and ‘NS Agent’ (Implementation-specific).
  - SPM is also a special ‘component’ – centre of the FF-M implementation, a library-like component provides API to other components. (Word `component` in the following sections no including SPM).
  - **Components’ initialization routine** and **service accessing procedure** run under this state, with the same runtime environment – the ‘application’ running environment.
- Following pages focus on the PSA API Call and Messaging procedure for this state.
  - The service request handling part following FF-M is well-known by us: Client, services, polices, messages and so on…
  - We need to expand the PSA API Call and Messaging to support 1.1 Features (SFN).



Serving state cont.

- PSA API Call:
  - SPM gets the calling component info and target service info then use **message** to connect them.
  - The call ABI is decided when **building** by calling component's type (ARoT/PRoT e.g.).
- Messaging
  - The messaging ABI is decided in **initializing/runtime** by service containing component's runtime model (IPC/SFN).
- *Boundary-switching is performed when component switching happens.*

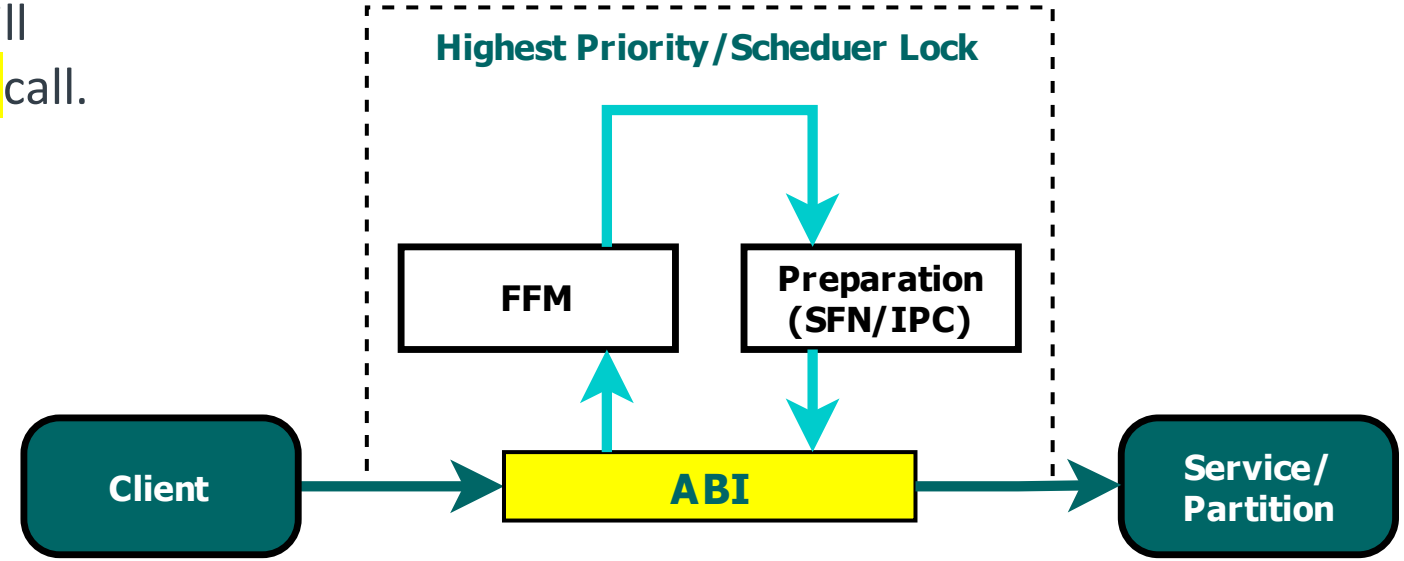


arm

Serving state – SPM internal working procedure

- PSA API Call/Messaging **ABI** in the same point
  - The ABI reaches the PSA API body. PSA API body contains the internal sub-routines: FFM compliance procedure (what we already know well), and messaging backends which is based on target component's runtime model (Enqueue messages when the target is IPC, push message into target runtime context when SFN).
  - The messaging backends return specific return codes to indicate the final action to service routine: '**NEED_SCHEDULE**' (IPC), '**CROSS_BOUND_CALL**'(Isolated SFN), and '**DIRECT_CALL**' (no boundaries between SPM and target SFN component).

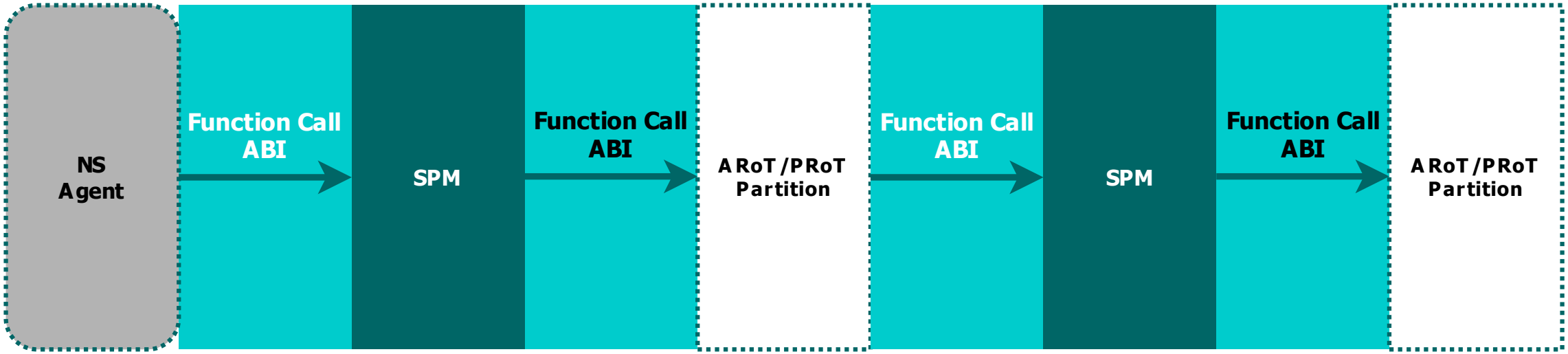
  - 'Scheduler lock': SPM internal logic is still preemptable but avoids nested PSA API call.
    For example:
    An interrupt preempts SPM API and mark one partition runnable. We still need to ensure preempted SPM API runs firstly after interrupt execution to avoid nested PSA API call.



Serving state for the simplest SFN model implementation

- There is only one ABI, both PSA API and service routine get called directly.
- No scheduling logic, no boundary cross ABI.

- *Note: Returning to caller is not mentioned in the diagram, as it is just a return of the calling ABI.*



arm

Serving state – Specific built-in services

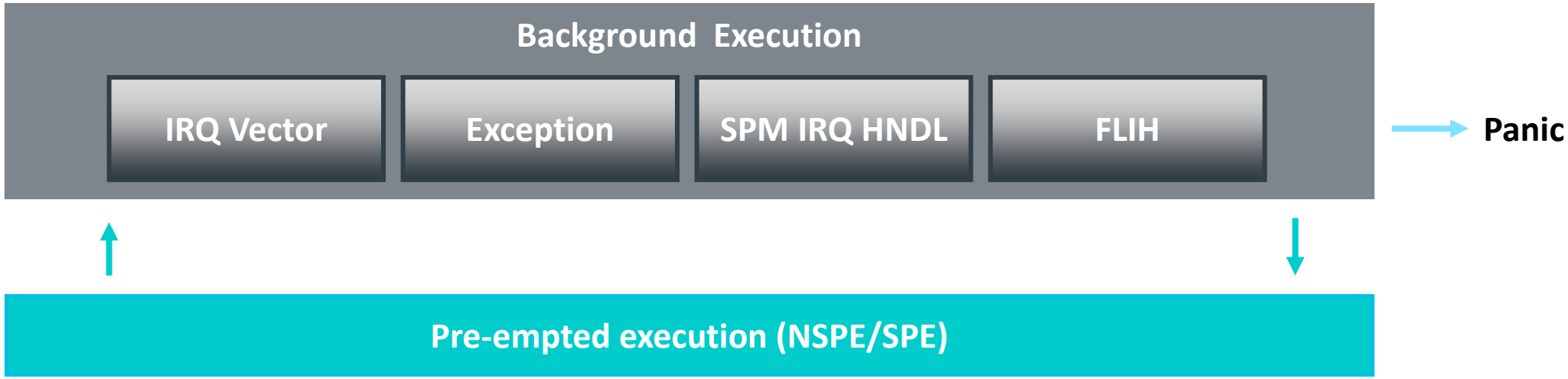
- For implementation-specific purposes
  - Trustzone context control.
  - Multiple-core client ID manipulation.
  - Shared boot data retrieving.
  - Accessed by Client API is a strong requirement, with manifests in C source – modularization purpose.
  - Can call SPM internal API directly – in the same domain as SPM.
  - Having special interfaces out of PSA defined is **restricted**.

IDLE state

- Jumps to NSPE in Trustzone based implementation.
- Or SPE ready signal sent to NSPE in Multiple-core based implementation.
- SPM regards the running component is NSPE or its agent.

Background state

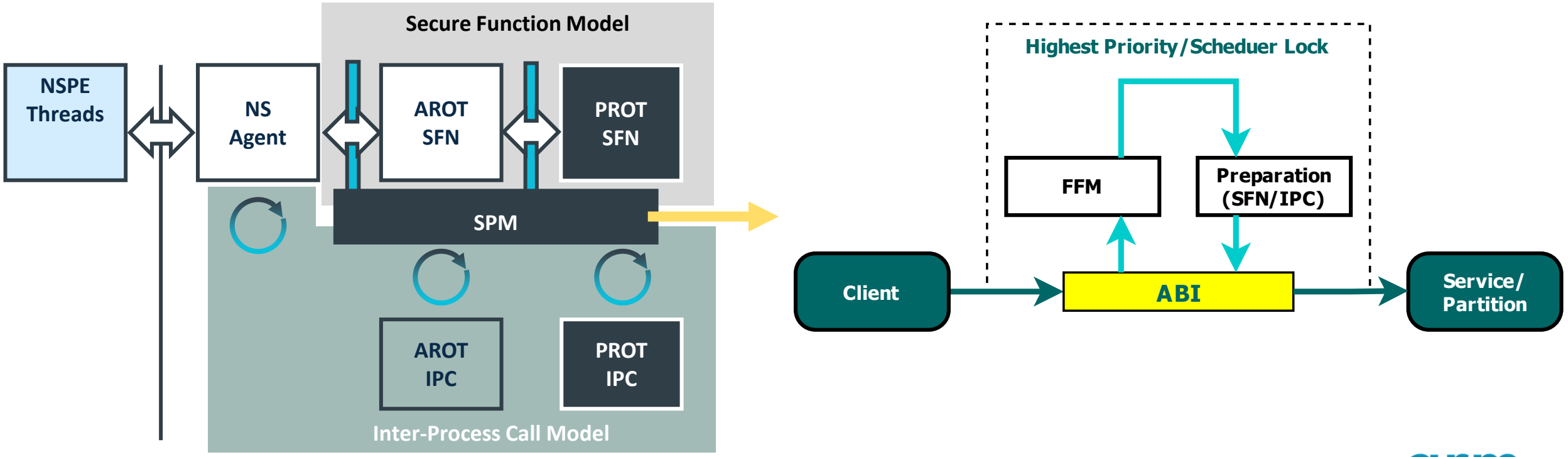
- Can happen anytime when interrupt or fault occurred.
  - The running component in the foreground may not be aware of these executions - that is why this state is called 'background'.
- The principle: **Return to preempted state in general case! (Or panic due to fault)**
  - Partition FLIH can be part of the background state, then the boundaries can be switched (multiple times) during background execution. But **MUST** recover to the original boundary setting of the preempted state before returning, if the boundaries were changed during background state.



arm

Summary

- Compare to the classic diagram mentioned in other SPM slides and the beginning page of this slides – focus on SPM now.
- Leave other affairs to services – make SPM a simple component.
  - Trustzone-specific management, e.g.



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