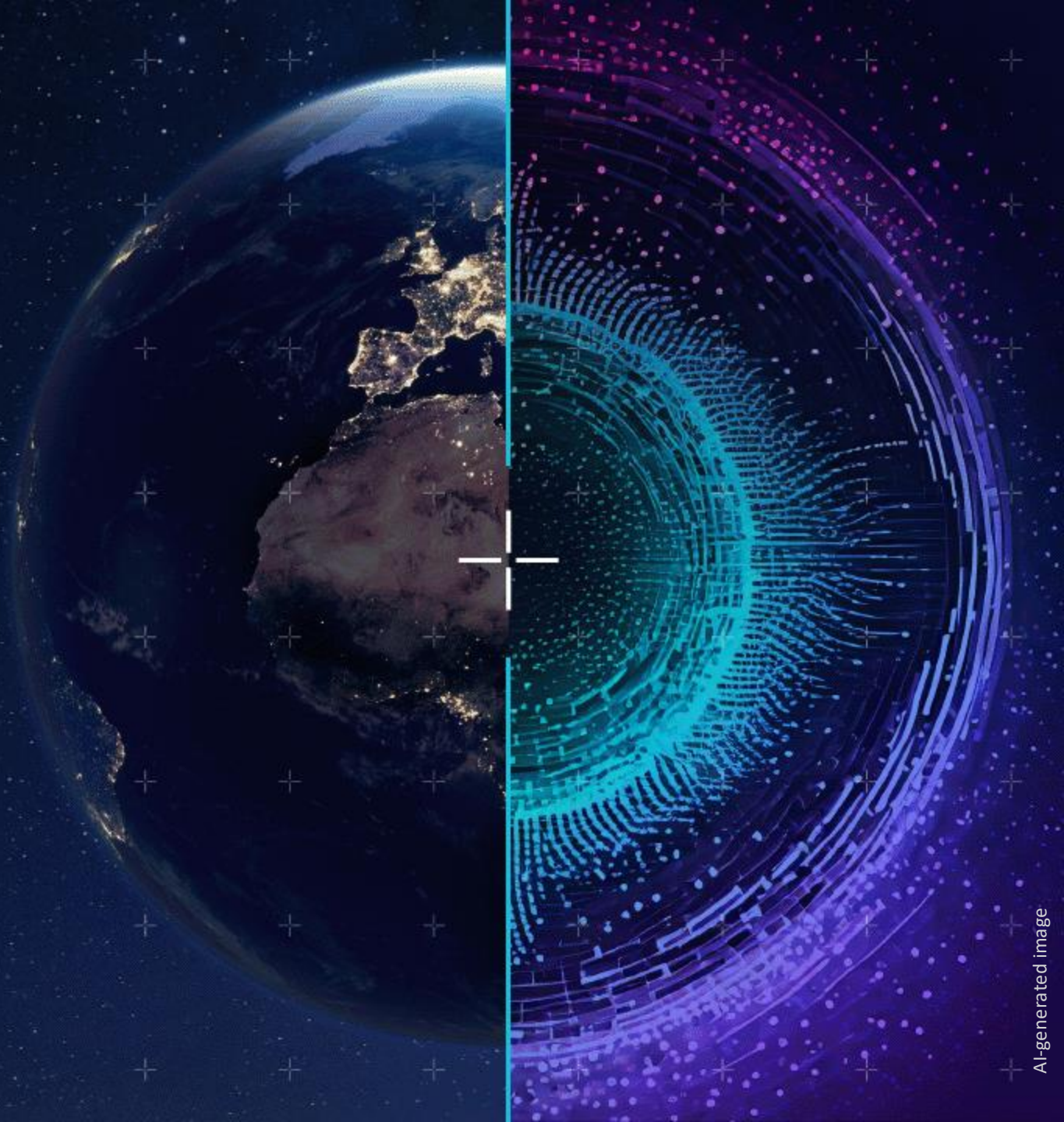
arm

# Secure Partition Live Activation Support



Madhukar Pappireddy

Oct 2025



AI-generated image

# Agenda

- Background
- Live Firmware Activation
- Partition Lifecycle Support
- Requirements for SP Live Activation
- Software architecture
- Implementation overview
- Challenges and constraints
- Prerequisites
- References
- Status

# Background

- Live Firmware activation allows firmware update without rebooting the system
- A Secure Partition can be live activated based on guidance in FF-A v1.3 ALP2 spec
- Use cases: Code patching of secure services in datacenter/cloud environments
- Requirement: Minimal disruption to the services
- Constraint:
  - Old and new SP instances must be compatible*
  - Only UP SP (single execution context) supported
  - Consequently, no need for system wide CPU Rendezvous

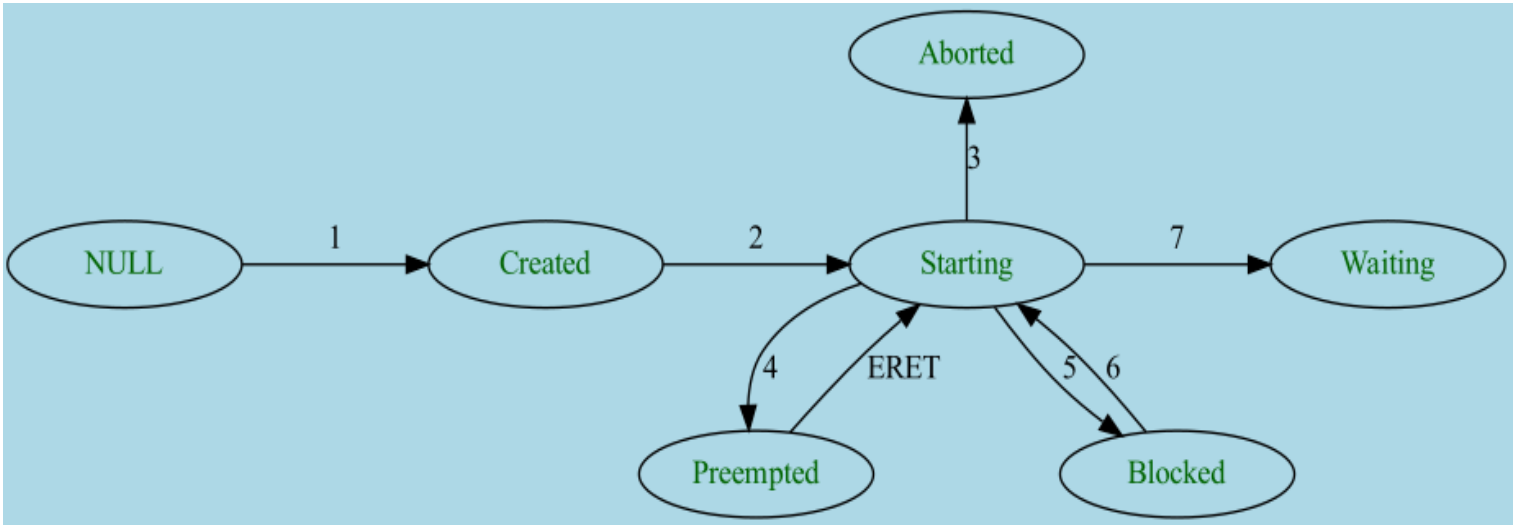
# Live Firmware Activation

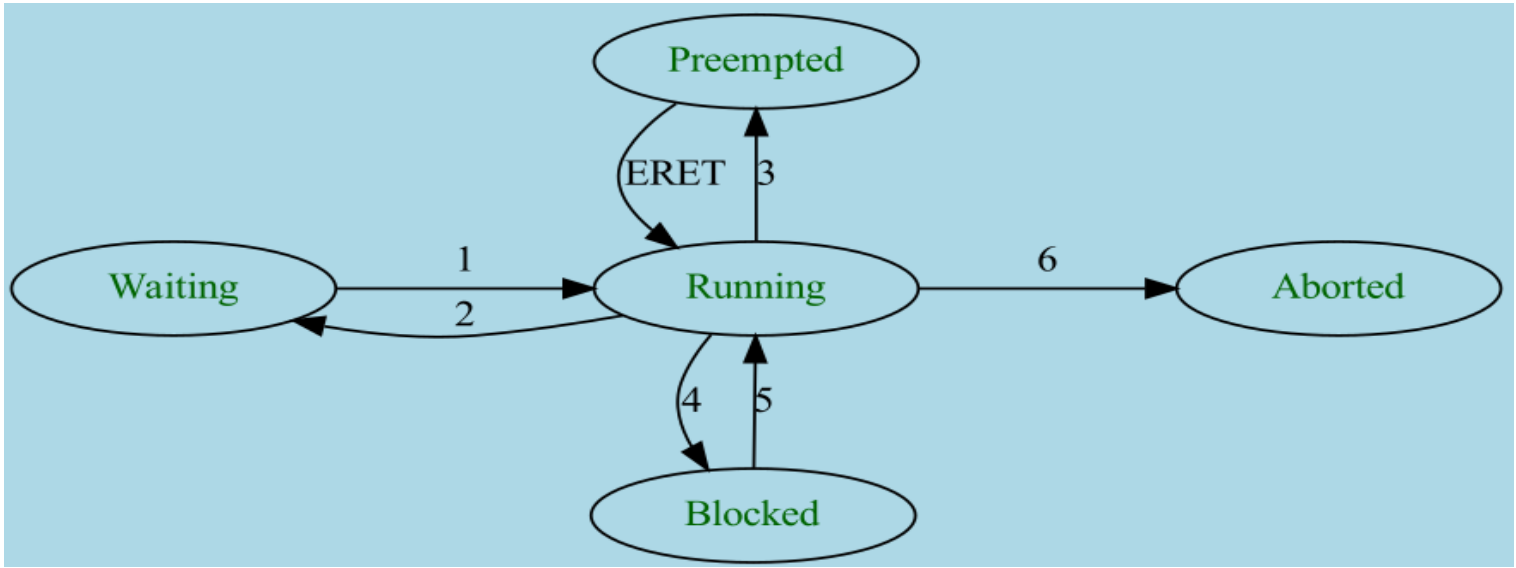
- Spec provides various SMC calls to orchestrate live activation of a platform firmware component.
- Assuming a simple scenario (i.e., no CPU Rendezvous, No reset)

| SMC | Purpose | Usage |
|---|---|---|
| LFA_GET_INFO | To query the LFA Agent for overall information about the platform's live activation environment. | Called at the start to learn how many firmware components are managed by the LFA Agent. |
| LFA_GET_INVENTORY | To discover properties and capabilities of each live-activation-capable firmware component. | Enumerate through all components to check which can be live-activated, whether CPU rendezvous is required, whether CPU reset is needed, and to retrieve component GUIDs and flags. |
| LFA_PRIME | To stage (load and authenticate) the new firmware image for the target component, preparing for activation. | Invoked (multiple times if needed) to copy the new firmware image into secure memory and perform necessary validation, without stopping the existing firmware yet. |
| LFA_ACTIVATE | To switch execution from the current firmware image to the new image. | Called after LFA_PRIME has completed. The call causes the LFA Agent to stop the old firmware, activate the new image, and resume operation. |

# Partition Lifecycle Support

- Provides guidance for:
  - Full lifecycle of a partition
  - Abort handling upon fatal errors
- Starting an SP vCPU
- Runtime of SP vCPU



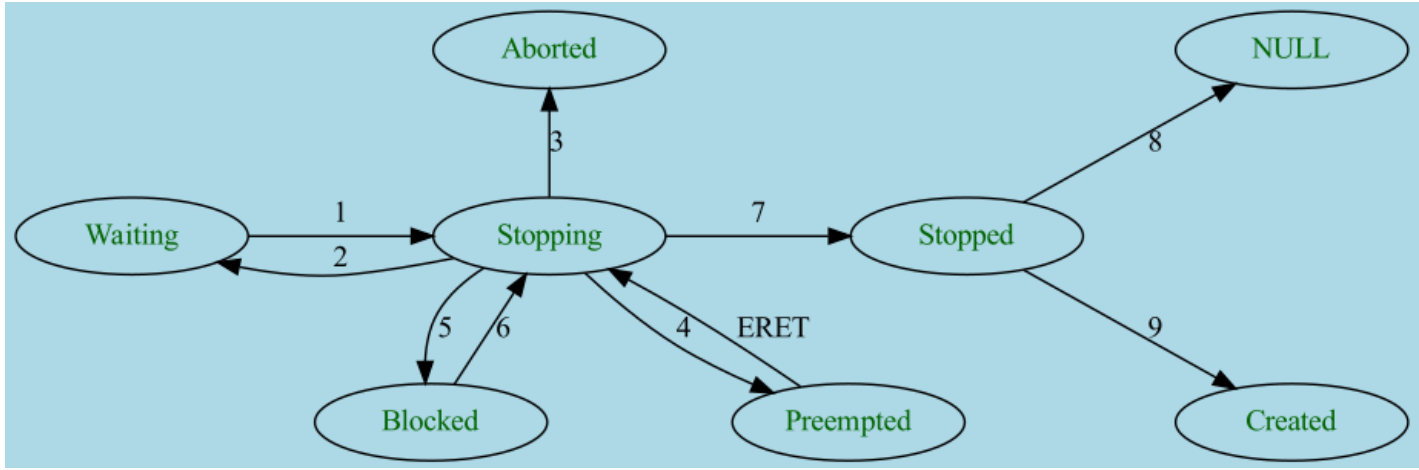



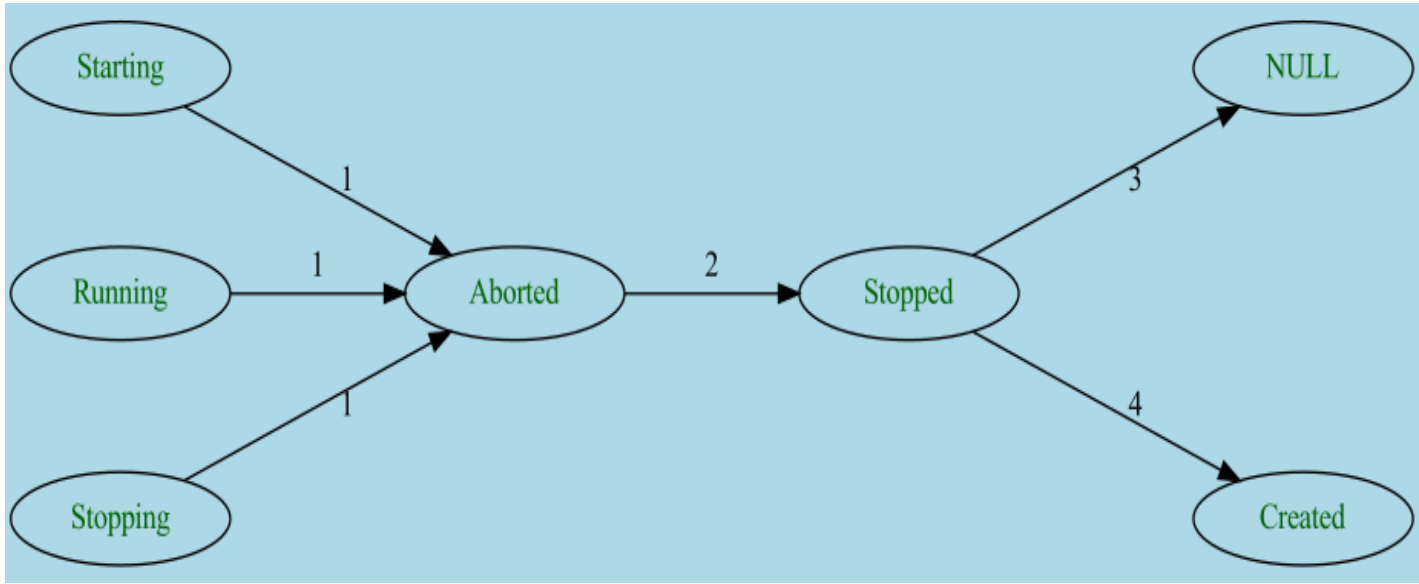



arm

# Partition Lifecycle Support

- Extends the life of a Partition:
  - From the time it is created
  - Until it is destroyed or live activates or restarted
- Stopping an SP vCPU
- Aborting an SP vCPU



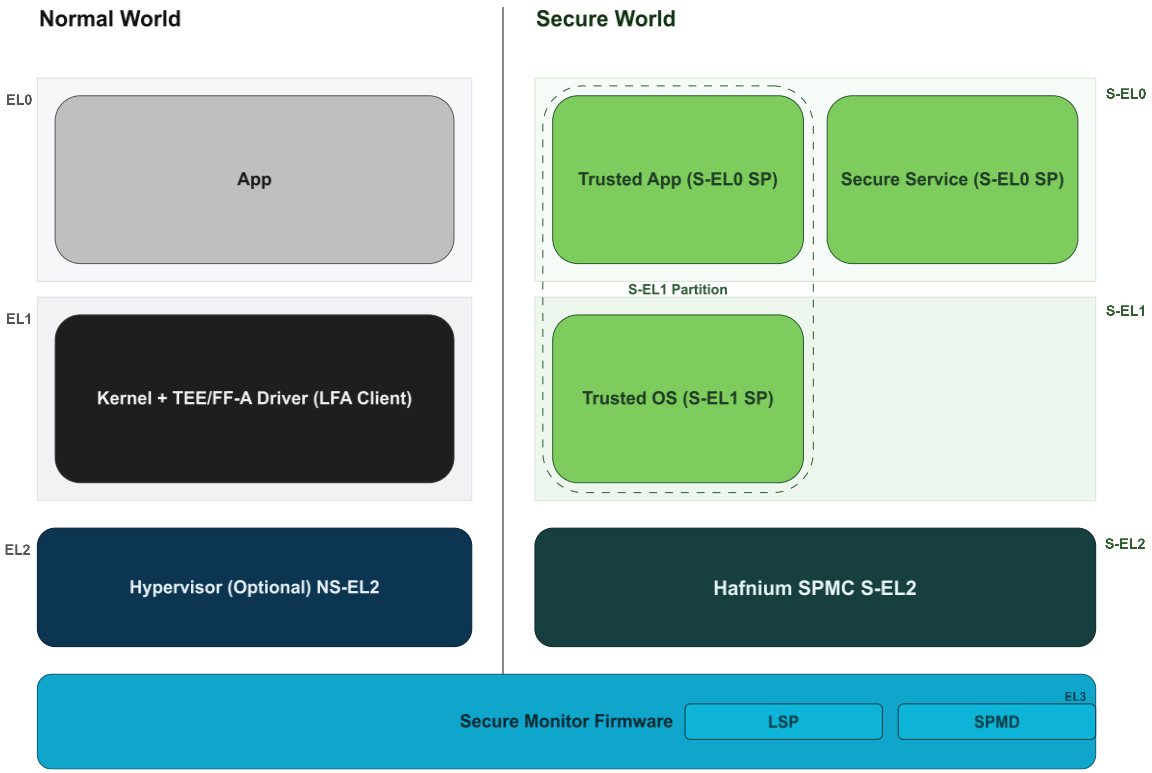
# SP Live activation requirements

- Framework state is preserved
- If RX/TX buffers are mapped:
  - Their contents are preserved
  - Mapped at same VA or IPA in target Image as current Image
- Bindigs of all VM and SP notifications are preserved
- Pending state of all VM, SP and framework notifications is preserved
- For memory regions shared/lent by another endpoint:
  - Mapped at same VA or IPA in target Image as current Image
  - With same memory attributes
- Meta data associated with memory management transactions are preserved
- This allows target SP Image to perform the same operations on memory as before

# SP Live activation requirements

- For stateful update, a special buffer is needed by an SP
- This memory region helps to transfer implementation defined state
- Base address specified by partition is:
  - VA (for S-EL0 partitions) or IPA (for S-EL1 partitions)
  - Aligned to translation granule used by SP
- Size is multiples of translation granules (equal to page size)
- SPMC need not map it in its translation regime
- Occupies the same physical address space in current and target SP image

# Software Architecture Overview

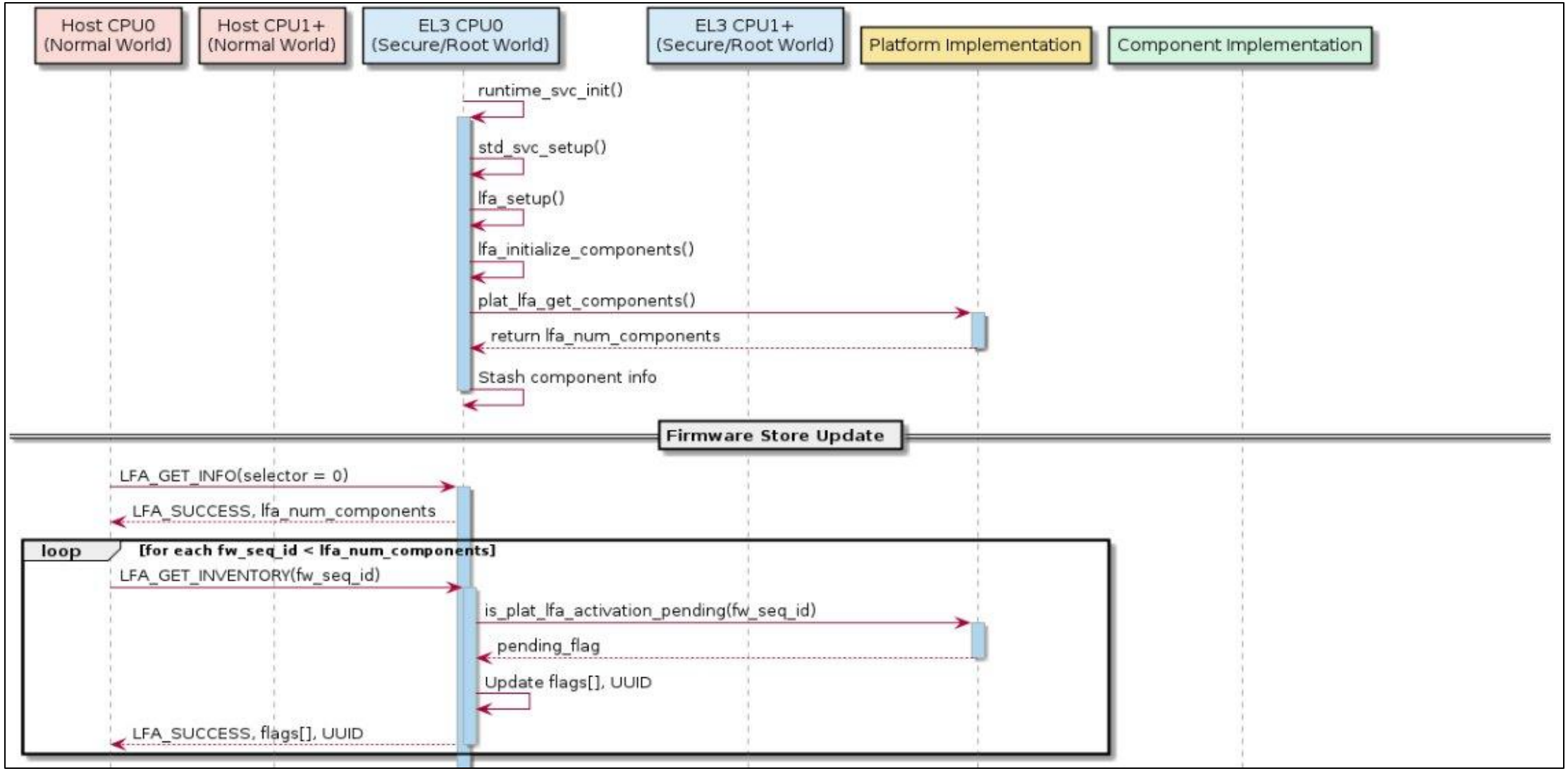



- Live activation of a SP needs LFA and FFA spec to be brought together
- Figure showing various entities involved in live activation flow & their roles

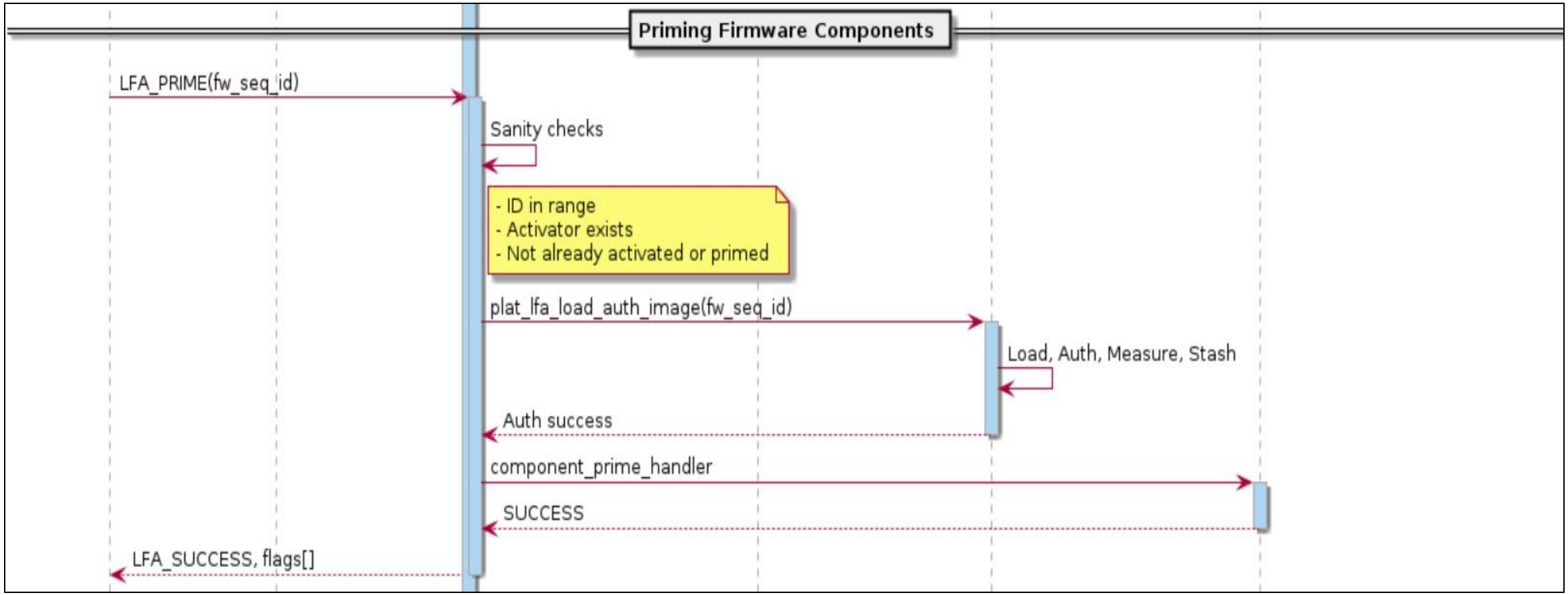


arm

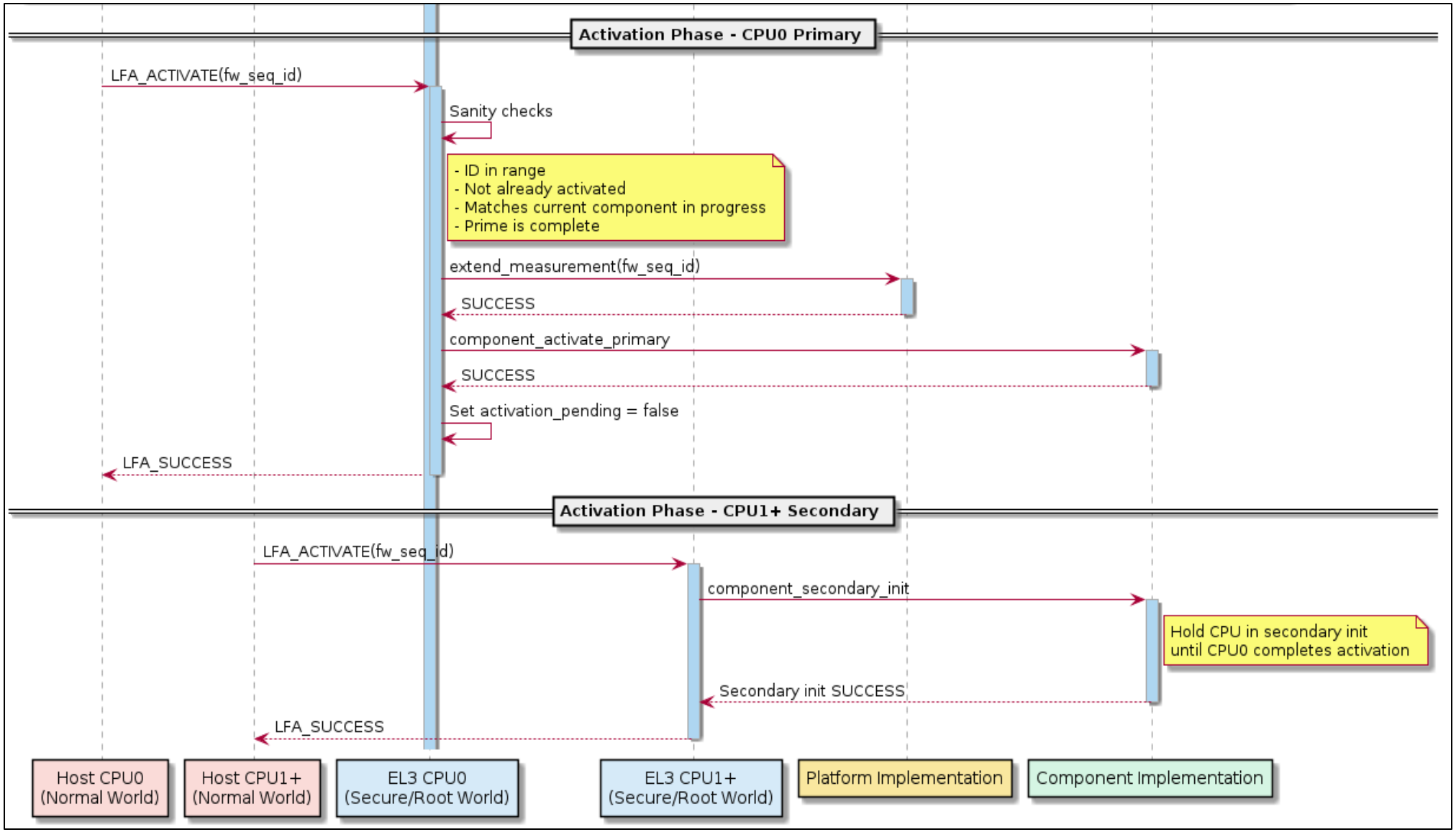
# LFA Flow – Firmware Discovery



# LFA Flow – Priming Firmware Component



# LFA Flow – Activating Firmware Component





# Implementation Overview

- Live Activation of any firmware component leverages the agent framework in TF-A
- The framework expects the platform to export callbacks for each operation
- Prime and Activate operations for each firmware component
- Mainline TF-A provides reference call back operations for BL31 and RMM
- Unlike BL31 and RMM, SP not a standardized component
- Hence additional complexity
- Responsibility is with platform port

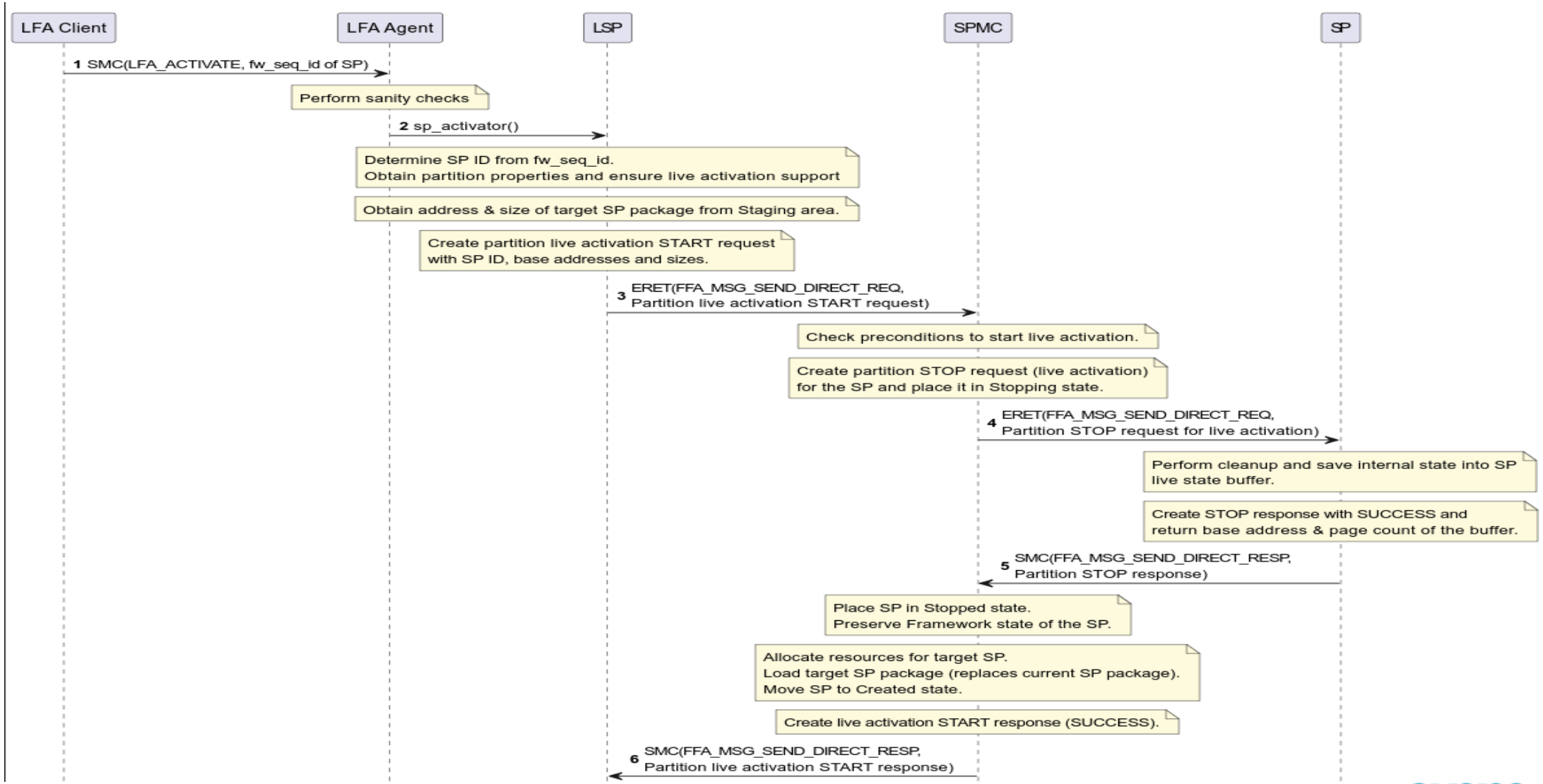
# Implementation Overview

- SPMC and SP only understand FF-A interfaces/protocols
- Need an intermediary entity to translate LFA to FF-A calls
- Hence the need for an LSP at EL3, managed by SPMD
- Live activation of an SP has several platform specific operations
- LSP must be independently implemented by each platform to suit their needs
- Aim is to provide common helper utilities to support SP live activation

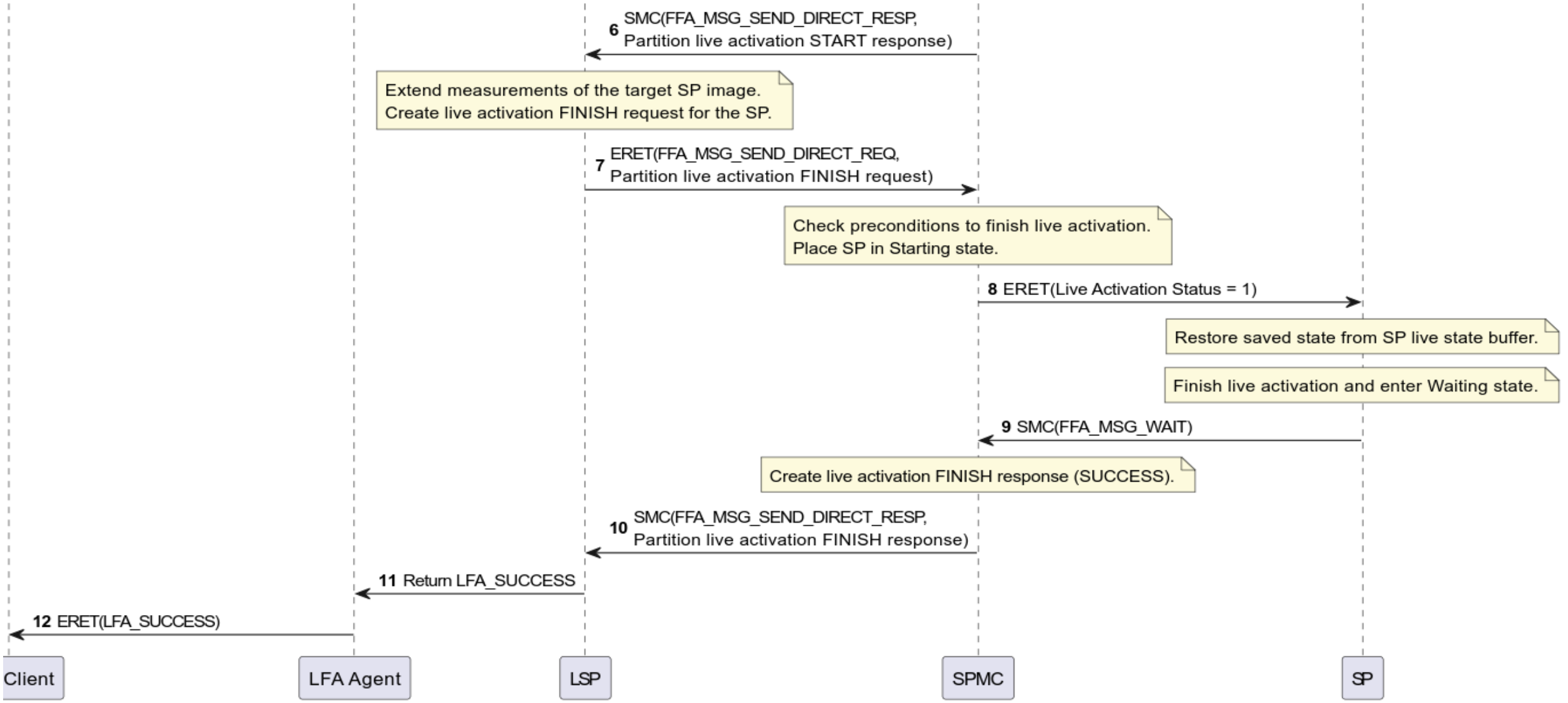
# Software Architecture Overview

- Important FF-A framework messages for LFA story:
  - Partition Stop request
  - Partition Stop response
  - Live activation Start request
  - Live activation Start response
  - Live activation Finish request
  - Live activation Finish response

# Software architecture: Live Activation flow chart



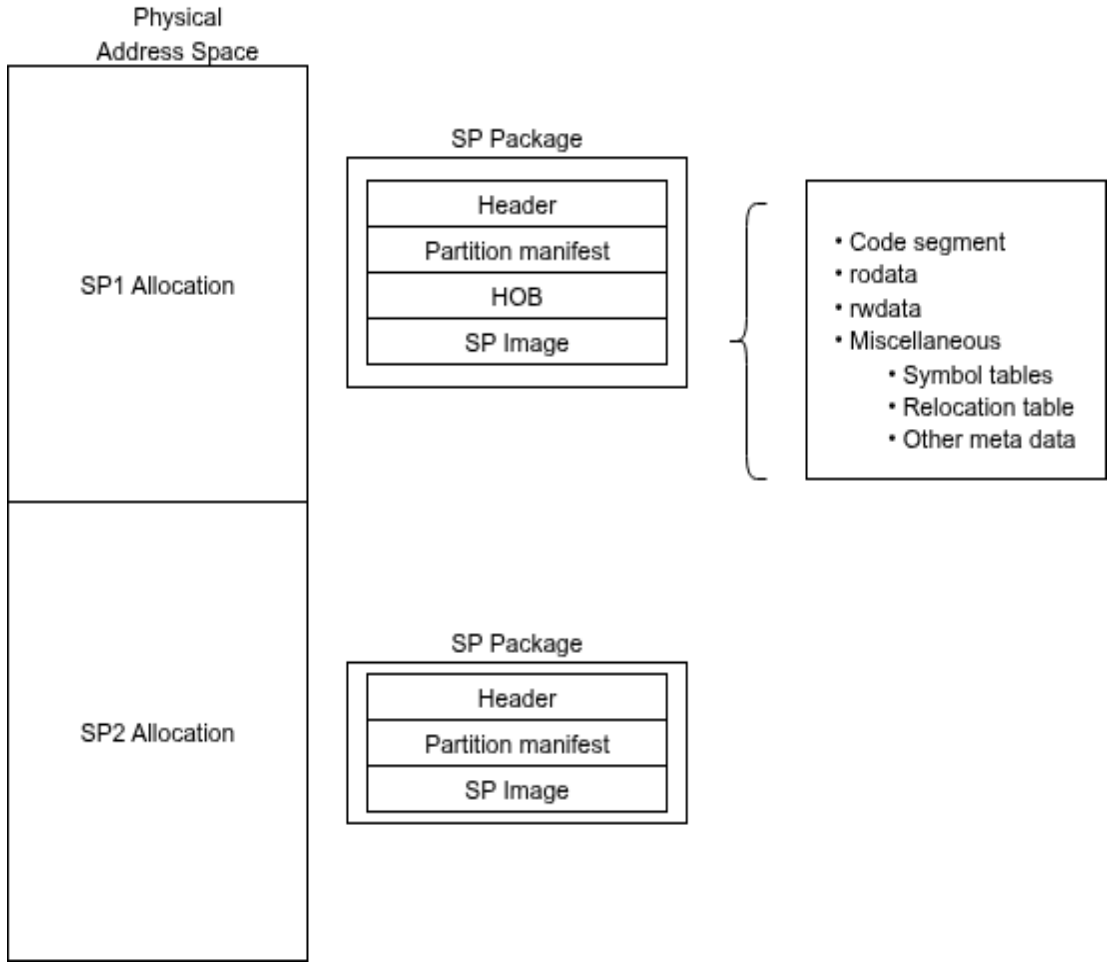
# Software architecture: Live Activation flow chart



# Challenges: SP image layout

- What can be preserved depends on how memory is allocated to SP
- Hafnium: pre determined load address and memory size for each SP:
  - SP1: Addr = 0x7100000; size: 0x100000
  - SP2: Addr = 0x7200000; size: 0x100000
- SP Package: special container
- Typically contains SP image binary (PROGITS only) and other payloads
- Must be large enough to host NOBITS segments during initialization





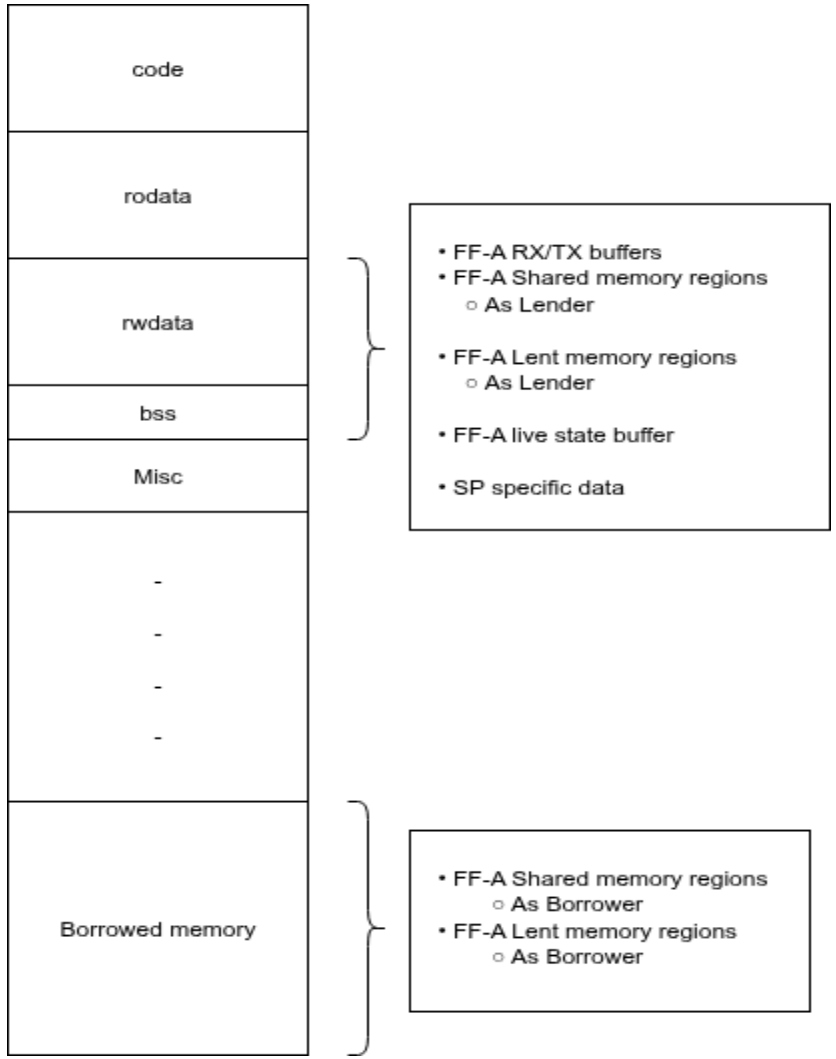

arm

# Challenges: SP memory categorization

- Working memory:
  - Allocated by SPMC (specified as load-address in SPMC manifest)
  - SP package gets loaded in runtime
- Static memory regions :
  - Declared in the SP manifest under "memory region" node
- Static device regions:
  - Declared in the SP manifest under "device region" node
- Dynamic regions:
  - Memory regions obtained by the SP via memory management transaction
  - Example: memory donated by another endpoint using FFA_MEM_DONATE

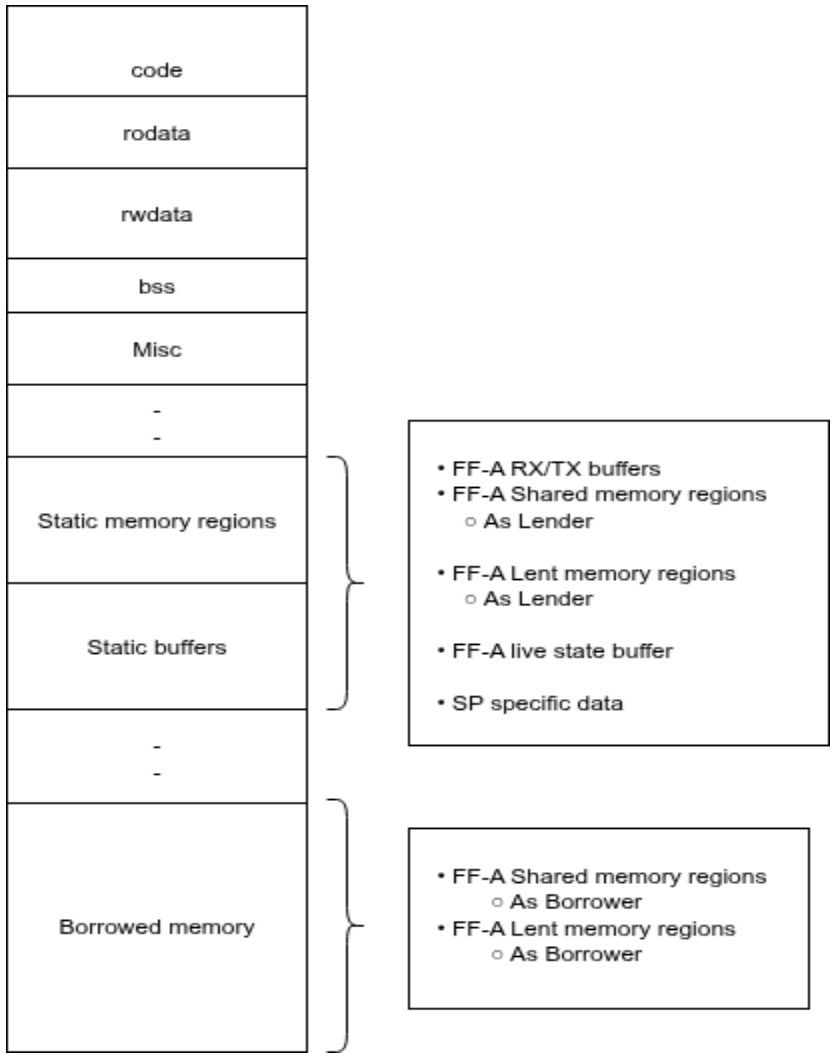
# Challenges: SP Image layout

- Hypothetical scenario: layout A
- Active (working) state in rwdata or bss
- Constraint: Hafnium performs in-place live activation
- Outcome: Framework state cannot be preserved



# Challenges: SP Image layout

- Hypothetical scenario: layout B
- Any working state pushed to SP live state buffer before live activation
- Take away: SP image to be carefully curated
- Categorize memory accessible to an SP before & after Live Activation



# Challenges: SP donate memory

- A memory region could be donated by SP (current instance) to another SP
- Upon live activation, target instance's memory map conflicts with another SP
- Needs to be addressed in spec
- Possible solutions:
  - Deny donate transaction
  - Terminate live activation and put in ABORTED state
- More corner cases?

# Conditions for SP live activation

- Implications for state full update :
  - Framework state or RXTX buffer must not overlap with working memory
  - No outstanding FF-A memory transaction regions overlapping with working memory
- Expectations from Platform/SP integrators:
  - SP live state buffer and RXTX buffer from static memory regions
  - SP manifest must be identical (Only code patching )
  - The new instance's package must be similar to old instance (except Image)
- If these conditions are not met, SPMC will abort the partition.

# Prerequisites

- Secure Partition
  - Count of vCPU = 1
  - FF-A version >= v1.3
  - Declare fields in partition manifest: "lifecycle-support" and "liveactivation-support"
  - Must specify Image UUID
  - No CPU Rendezvous required for UP SP
  - Curate image and allocate memory regions
- SPMC: Hafnium shall support Live activation
- TF-A:
  - SPD = spmd
  - SPMD_SPM_AT_SEL2 = 1
  - ENABLE_SPMD_LP = 1
  - Callbacks and hooks by platform port

# Sample partition manifest

- Preserve framework state

```
/dts-v1/;

/ {
        compatible = "arm,ffa-manifest-1.0";
        ffa-version = <0x00010003>;
        uuid = <0x9458bb2d 0x353b4ee2 0xaa25710c 0x99b73ddc>;
        execution-ctx-count = <1>;
        exception-level = <2>; /* S-EL1 */
        execution-state = <0>; /* AARCH64 */
        load-address = <0x6480000>;
        entrypoint-offset = <0x2000>;
        xlat-granule = <0>; /* 4KiB */
        messaging-method = <0x7>;
        ns-interrupts-action = <1>;
        boot-order = <1>;
        notification-support; /* Receipt of notifications. */
        gp-register-num = <0>;

        /* Lifecycle support and Live activation fields. */
        lifecycle-support;
        abort-action = <1>; /* Destroy */
        live-activation-support;
        live-activation-register = <1>;

        image-uuid = <0x962a7bf0 0x174d471d 0xa686c89e 0x5c3e254e>;

        /* Boot Info */
        boot-info {
                compatible = "arm,ffa-manifest-boot-info";
                ffa_manifest;
        };

        device-regions {
                compatible = "arm,ffa-manifest-device-regions";

                uart1 {
                        base-address = <0x00000000 0x1c0a0000>;
                        pages-count = <1>;
                        attributes = <0x3>; /* read-write */
                };

                sec_wdog {
                        /* SP805 Trusted Watchdog Module */
                        base-address = <0x00000000 0x2a490000>;
                        pages-count = <32>; /* Two 64KB pages */
                        attributes = <0x3>; /* read-write */
                        interrupts = <56 0x900>;
                };
        };

        memory-regions {

                compatible = "arm,ffa-manifest-memory-regions";

                live-state-buffer {
                        description = "live-state-buffer";
                        base-address = <0x00000000 0x6780000>;
                        pages-count = <1>;
                        attributes = <0x3>; /* read-write */
                };

                sp_heap: heap {
                        description = "heap";
                        base-address = <0x00000000 0x6780000>;
                        pages-count = <1>;
                        attributes = <0x3>; /* read-write */
                };

                sp_share_mem: secure-memory {
                        description = "share-memory";
                        base-address = <0x00000000 0x7100000>;
                        pages-count = <1>;
                        attributes = <0x3>; /* read-write */
                };

                rxbuffer: rx-buffer {
                        description = "rx-buffer";
                        pages-count = <1>;
                        base-address = <0x00000000 0x7300000>;
                        attributes = <0x1>; /* read-only */
                };

                txbuffer: tx-buffer {
                        description = "tx-buffer";
                        pages-count = <1>;
                        base-address = <0x00000000 0x7301000>;
                        attributes = <0x3>; /* read-write */
                };
        };
};
```

# References

- FF-A v1.3 (ALP2), Ch. 18.10 (Live Activation)
- Live Firmware Activation SMC Interface v1.0 (BETA)
- TF-A Tech forum titled "TF-RMM Live Activation" on June 12th 2025

# Status

- Prototyping live activation of a test SP in hafnium project
- Closely mimics S-EL0 StMM partition requirements
- Looking for additional insights
- Any concerns?
- Questions?

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