# Firmware Framework-M 1.1 Feature Update in Trusted Firmware-M

Linaro Connect
VIRTUAL • SPRING 2021

# Platform Security Architecture Firmware Framework-M

- Firmware Framework-M is a specification that describes a standardized interface and programming environment for secure applications on Cortex-M devices.

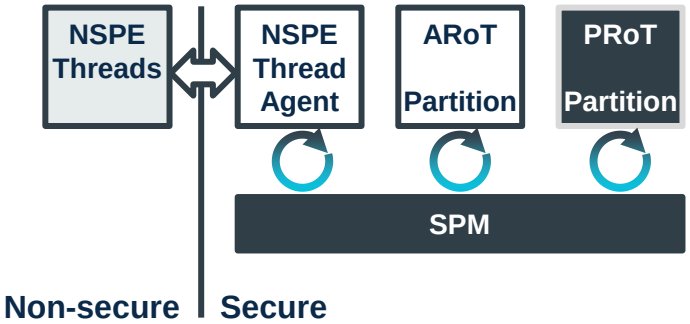
- TF-M provides a reference implementation of this standard. FF-M v1.0 has been a challenge to implement on very small footprint devices. A new version, FF-M is updated to v1.1 to address this, by introducing a Secure Function model and Stateless services, also enhanced peripheral supporting. This talk is about the new features provided in FF-M v1.1.

- https://developer.arm.com/architectures/security-architectures/platform-security-architecture

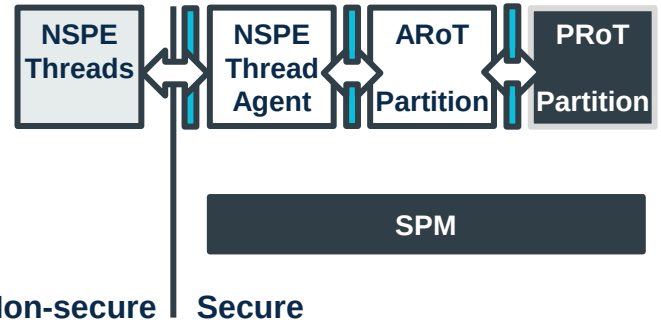
# Firmware Framework-M 1.1 Extensions

- Secure Functions
  - There should be continuity between any new lighter-weight framework and FF-M v1.0, enabling the same RoT Service code to be used with different types of framework.
  - Target a smaller or simpler system, functions like **memory-mapped IOVECs** are also applicable.
- Stateless RoT Services
  - Provide an optimized mechanism for common Secure Partition RoT Service development patterns.
  - Improving efficiency for these use cases requires a reduction in flexibility or security mitigation.
- Interrupt Handling
  - The signal-based mechanism makes it very difficult to handle interrupts in a bounded time.
  - The FF-M v1.0 API assumes that the peripheral MMIO interface is sufficient to control its interrupts.
- Improvements and compatibilities
  - A few smaller clarifications and improvements.
  - Make migration to FF-M v1.1, and adoption of FF-M v1.1 features easy.

# Secure Function Implementation




IPC Based Programming Model




Secure Function Based Programming Model

# Secure Functions – Manifest changes

**Manifest**

```
"model": "IPC",
"entry_point": "partition1_main",
"services": [
    {
        "name": "service1",
        "sid": "0x00000081",
    },
],
```

**Sources**

```
void partition1_main(void)
{
    while(1) {
        signals = psa_wait(PSA_WAIT_ANY, PSA_BLOCK);
        if (signals & SERVICE1_SIGNAL) {/* .... */}
    }
}
```

**Manifest**

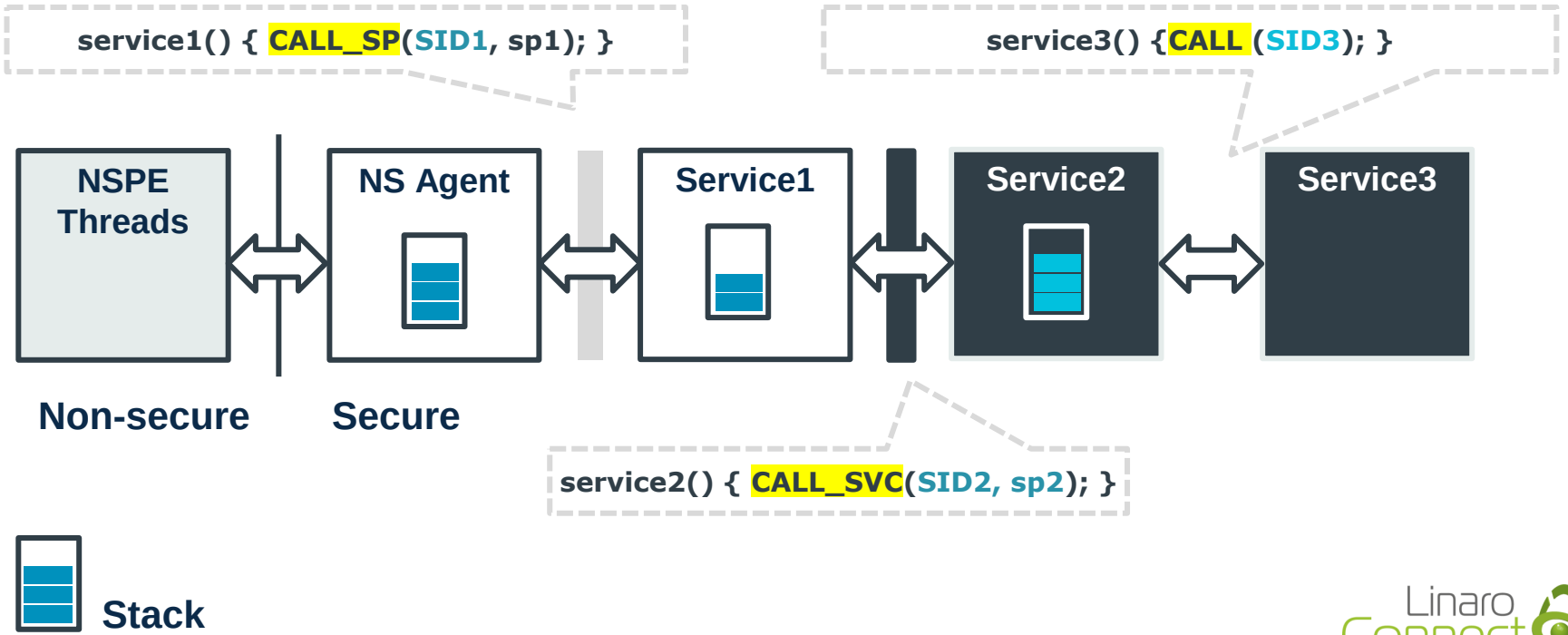
```
"model": "SFN",
"entry_init": "partition1_init",
"services": [
    {
        "name": "service1",
        "sid": "0x00000081",
    },
],
```

**Sources**

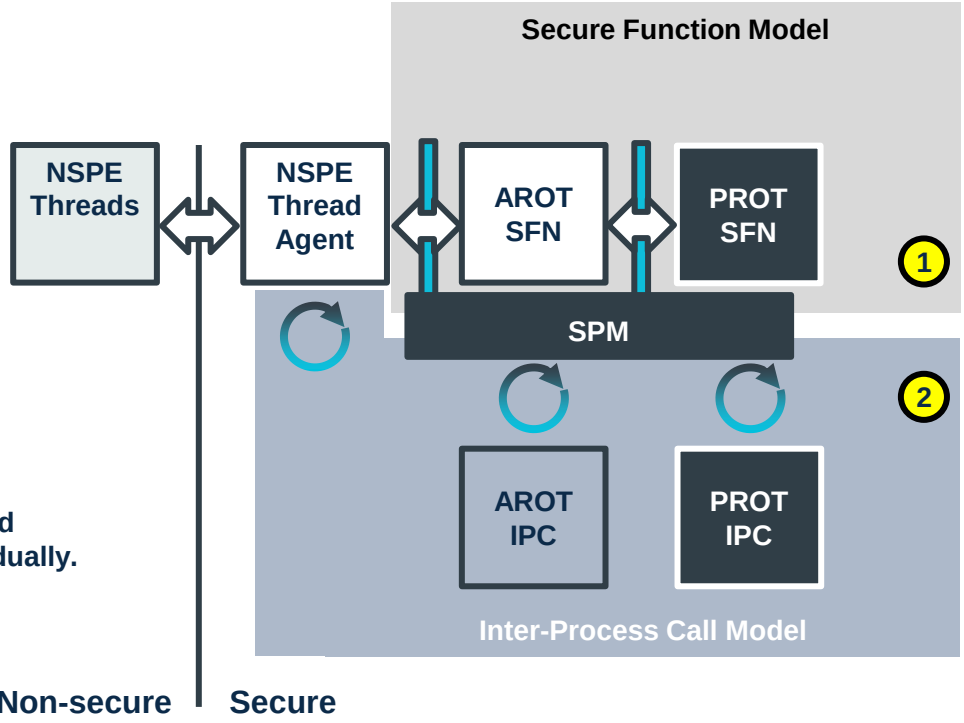
```
psa_status_t partition1_init(void)
{
    return PSA_SUCCESS;
}

psa_status_t service1_sfn(const psa_msg_t* msg)
{
    /* .... */
}
```

# Secure Functions – Stack Usage and Call Type



# Secure Functions – Hybrid Model




1 and 2 can be provided by the implementation individually.

# Memory-mapped IOVEC – Optional Feature

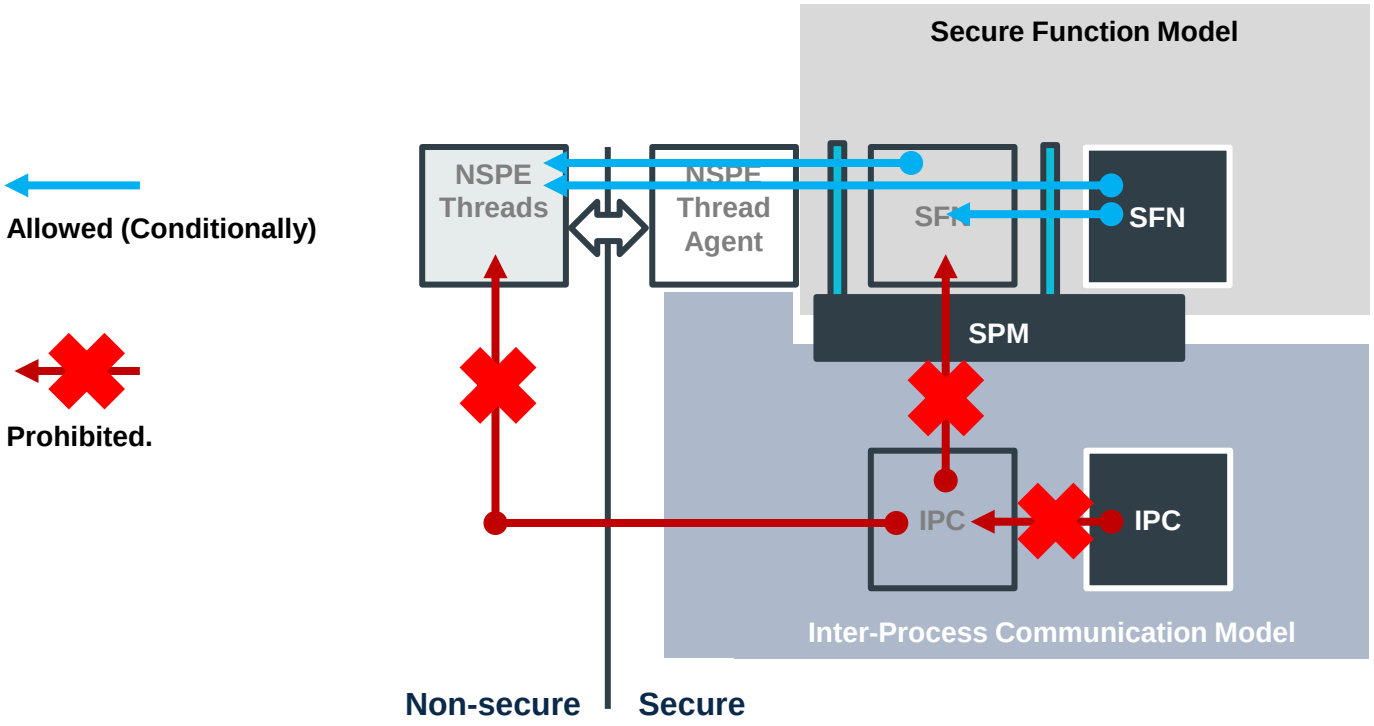
- Mapped for direct access vs psa_read/psa_write.
- This is an optional API for frameworks implementing FF-M v1.1
  - Direct access to client parameters is important for smaller, simpler systems.
  - Direct access might be unacceptable in secure systems as it bypasses some memory safety mitigations.
  - Direct access can be difficult to implement in systems with complex memory architectures.
- A RoT Service must explicitly enable MM IOVEC in the manifest file to use the API.
  - This limits the potential for exploiting this feature.
- Each parameter can either be mapped using psa_map_invec () or psa_map_outvec () or accessed using the existing psa_read () or psa_write () functions.
  - There is limited value in mixing these APIs, and the restriction simplifies the implementation
- Unmapping parameters is automatic, but explicit unmapping is required for output parameters to provide the length of output that was written.

# Memory-mapped IOVEC - Implementation

- TF-M provides it as an example for the system that:
  - A very simple SFN model only system that SPE could access both SPE and NSPE memory.
  - A complex system but can grantee successful memory mapping.
- Practical design needs to review the system design before applying MM-IOVEC.
  - Secure and Non-secure has aligned memory access and synchronization.

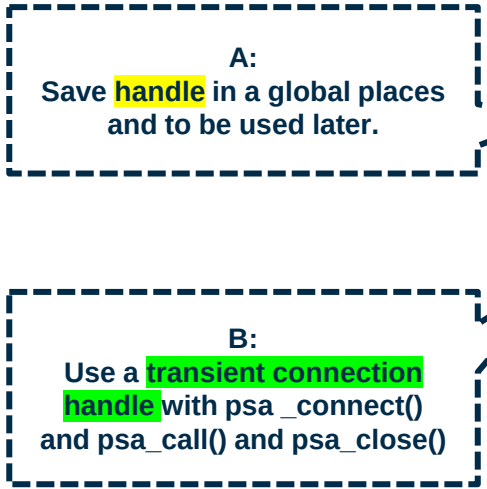
```
const void * psa_map_invec(psa_handle_t msg_handle, uint32_t invec_idx);
void psa_unmap_invec(psa_handle_t msg_handle, uint32_t invec_idx);
void *psa_map_outvec(psa_handle_t msg_handle, uint32_t outvec_idx);
void psa_unmap_outvec(psa_handle_t msg_handle, uint32_t outvec_idx, size_t len);
```

# MM-IOVEC Functionalities



# Stateless RoT Services – Background

- Many RoT Service APIs provide standalone operations that do not maintain any state in the RoT Service or do not expose any kind of context to the caller.
- To implement these functions as a RoT Service using the version 1.0 API, the client-side implementation of the service must use one of the following techniques:




```
int32_t RoTService(void)
{
    handle = psa_connect(SID, VERSION);
    if (!PSA_HANDLE_IS_VALID(handle) {
        return PSA_HANDLE_TO_ERROR(handle);
    }
    status = psa_call(handle, PSA_IPC_CALL, NULL, 0, NULL, 0);
    psa_close(handle);

    return status;
}
```

# Stateless RoT Services – Implementations

- Stateless RoT Services do not use connections.
  - psa_call() with a static handle in the reserved range for stateless RoT Services.
  - psa_connect() and psa_close() would fail when operating on stateless RoT Service.
  - psa_set_rhandle() would fail when operating on stateless RoT Service messages.

**partition_a.json:**

```
"name": "SERVICE_A",
"statelss_handle": auto
```

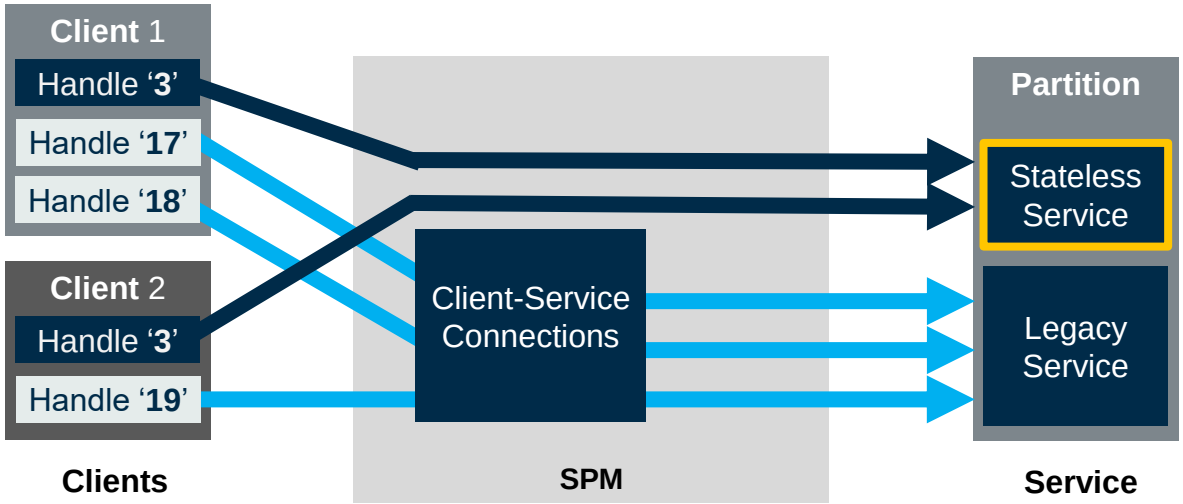
**psa_manifest/sid.h:**

```
/* Auto-Generated file, DO NOT MODIFY! */
#define SERVICE_A_HANDLE    ((psa_handle_t)(version | 3))
```

**psa_service_a_client.c:**

```
/* RoT Service API */
psa_status_t rot_service_a(void)
{
    return psa_call(SERVICE_A_HANDLE, PSA_IPC_CALL, NULL, 0, NULL, 0);
}
```

# Stateless RoT Services – SPM Internal Logic

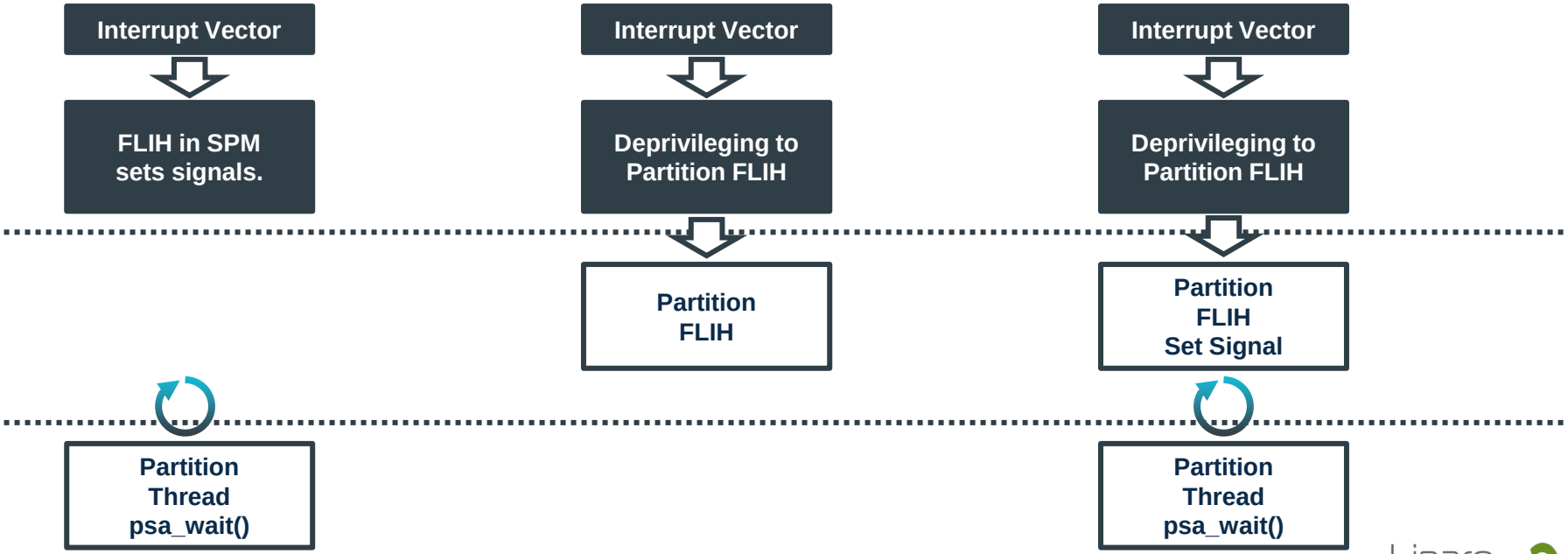


# Peripheral Support - Background

- The v1.0 API was designed to be simple and easy to use securely. It avoided concurrent execution within the Secure Partition by requiring the interrupt handler to run within the execution thread.
  - Working around the limitations of this API for lower latency interrupt requirements is painful, complex and error prone.
  - Framework support for immediate, asynchronous interrupt handling is needed in these use cases.
- The v 1.0 API assumed that peripheral interrupts are all behaved well:
  - The interrupt can be disabled at source using a peripheral control register.
  - The peripheral resets with all interrupts disabled.

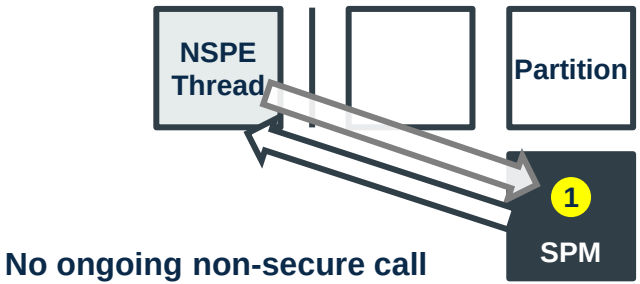
# Peripheral Support – Interrupt

- First Level Interrupt Handler (FLIH) and Second Level Interrupt Handler (SLIH).



# Peripheral Support – Secure IDLE Processing

- Secure IDLE needs special design under Trustzone-M based implementation.
  - An IDLE thread would work under RPC-based implementation.





1. Don’t schedule if no ongoing NS call (FLIH is still executable).
2. ‘WFI’ in secure entry if no registered NS IDLE handling.
3. Calling NS IDLE handling callback function.

# Peripheral Support – Non-secure IDLE callback

- Ideally, the callback is bound to each non-secure thread that needs to access secure services.
  - NS can share this callback between threads if it can handle this scenarios.

```
/* ‘sp’ and ‘splim’ can be special pattern to call callback with NS caller’s stack */
status  register_idle_callback(handle_t h, pfn_t cb, uintptr_t sp, uintptr_t splim);

/* ‘cparam’ is given when allocating handle ‘h’ */
typedef status (*pfn_t)(void *cparam);
```

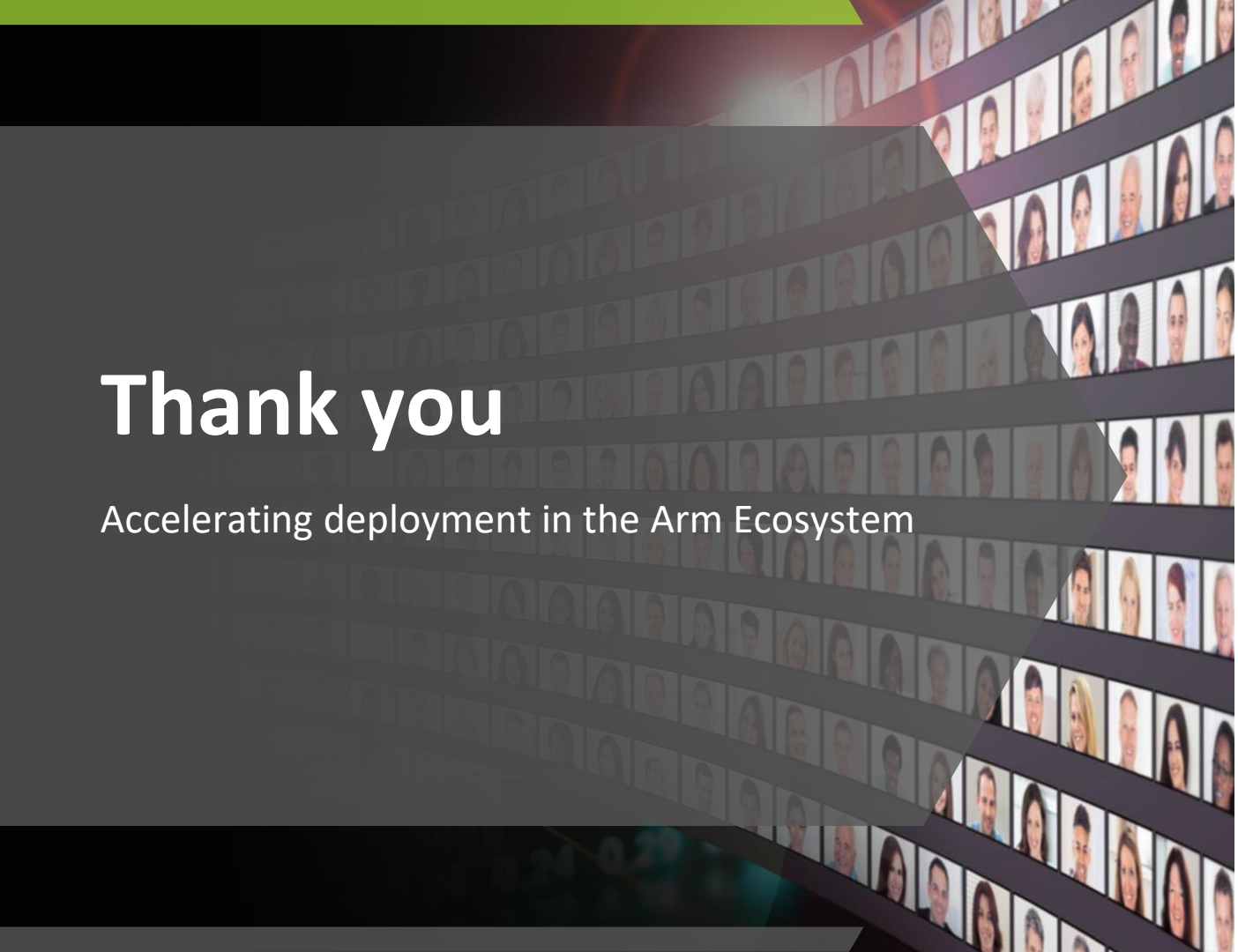
# Other FF-M v1.1 updates implementation

- Revision of the definition for RoT Service
  - Distinguishing between a generic RoT Service, and one that is implemented in a Secure Partition
- Clarification of the rules for a PSA RoT Service
  - Version 1.0 had conflicting rules about RoT Services that are deployed in the PSA Root of Trust
- Relaxation of the memory access rules for Constant data (I7)
- Permitting execute access to Constant data is an acceptable approach in some systems
- Permitting the use of symbolic definitions for `stack_size` and `heap_size` attributes in the manifest file
- Other changes and clarifications

# Last Page

- The FF-M v1.1 document update:
  - https://developer.arm.com/documentation/aes0039/latest

- The design listed here are still ongoing, may be adjusted:
  - Check the mailing list and Tech Forum for the latest update
  - https://lists.trustedfirmware.org/mailman/listinfo/tf-m
  - https://www.trustedfirmware.org/meetings/tf-m-technical-forum/



# Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect
VIRTUAL • SPRING 2021