Trusted Firmware Deep Dive

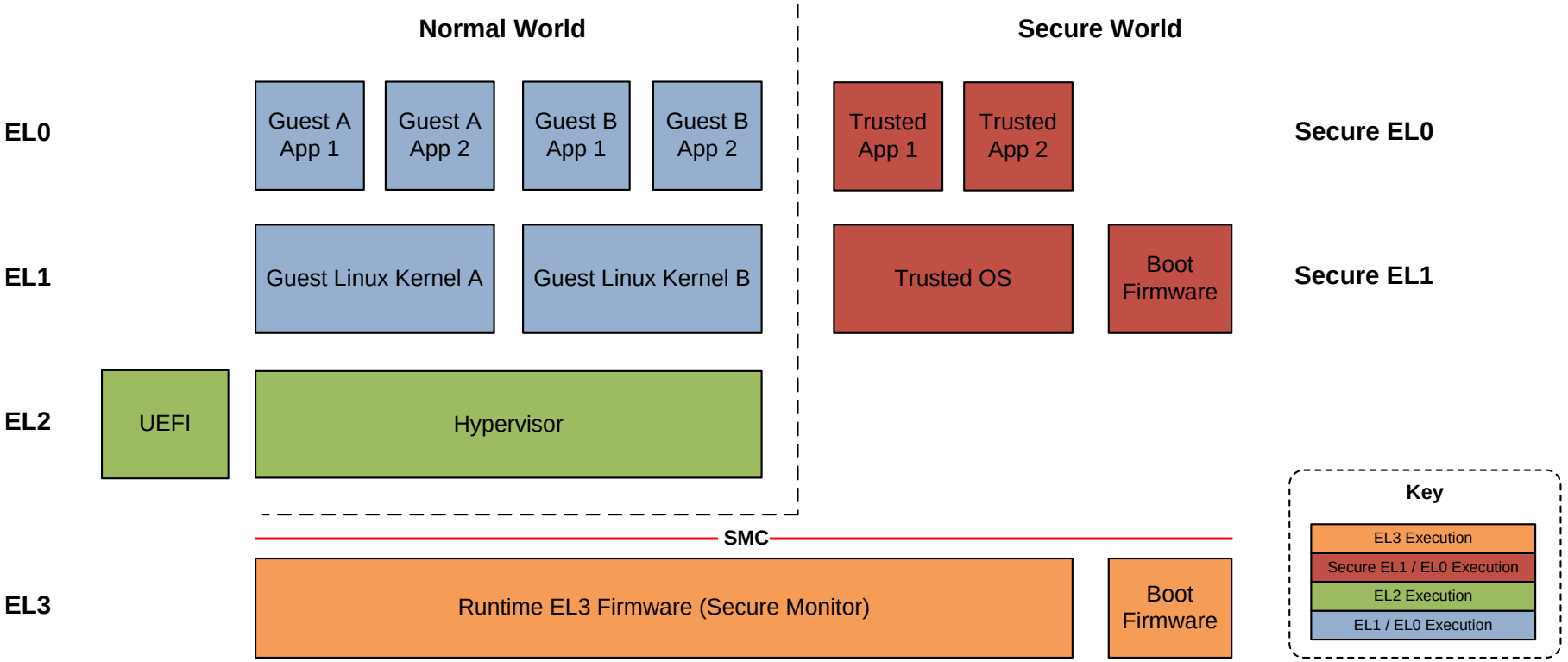
Dan Handley

Charles Garcia-Tobin

Agenda

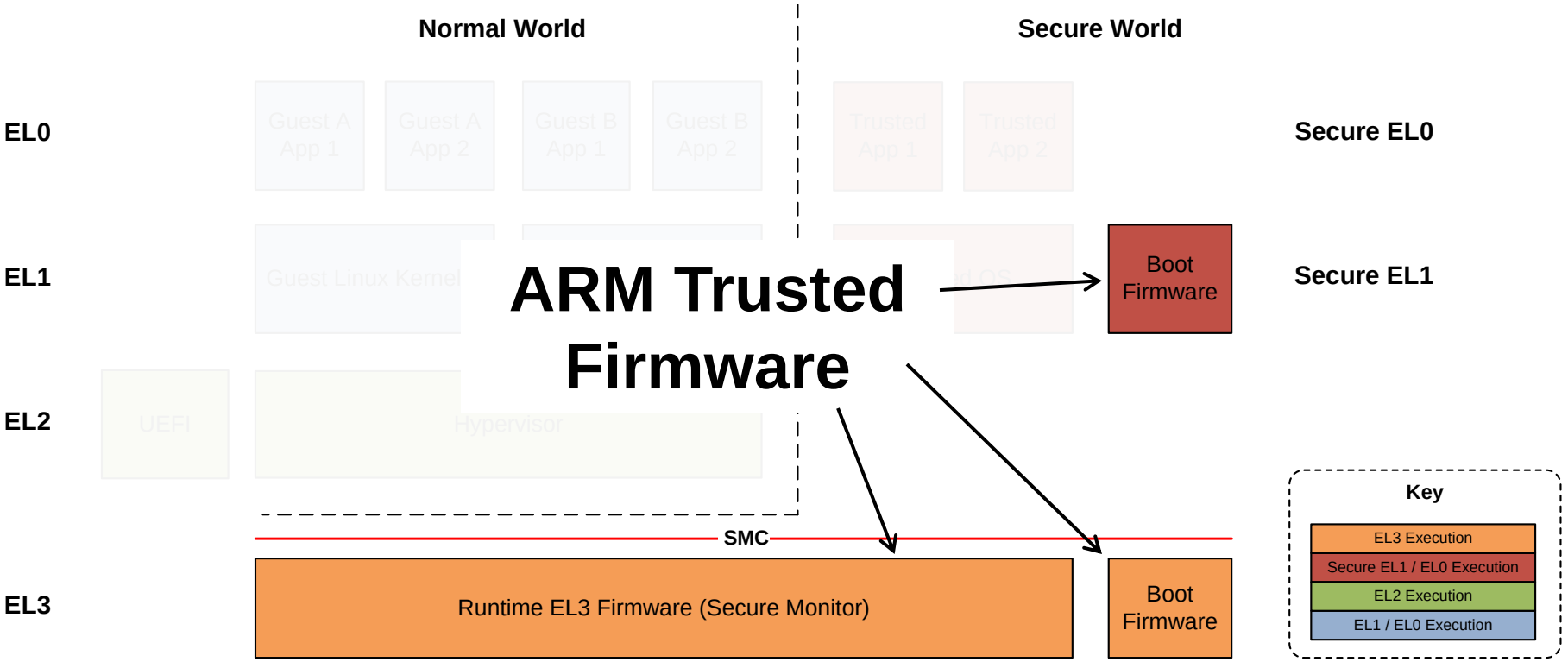
- Architecture overview
- Memory usage
- Code organisation
- Cold boot deep dive
- PSCI deep dive

Example System Architecture



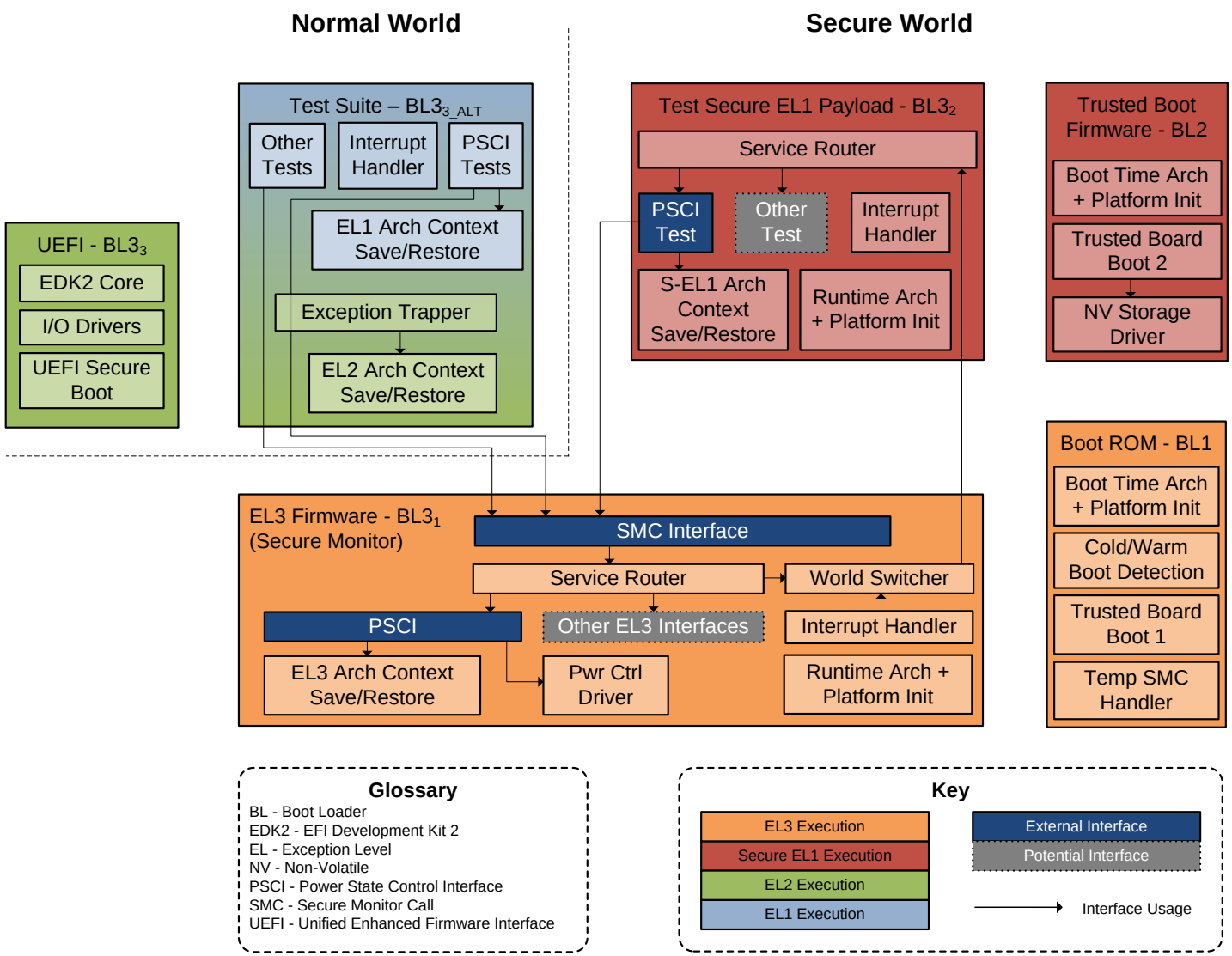
- SMC Calling Convention Interface (ARM DEN 0028A) is gateway to:
  - Runtime EL3 Firmware
  - Trusted OS / TEE services
- Power State Coordination Interface (PSCI) (ARM DEN 0022B.b)
  - Transported by SMC calls
- Also see ARMv8-A Architecture Manual (AR150-DA-70000)

Example System Architecture

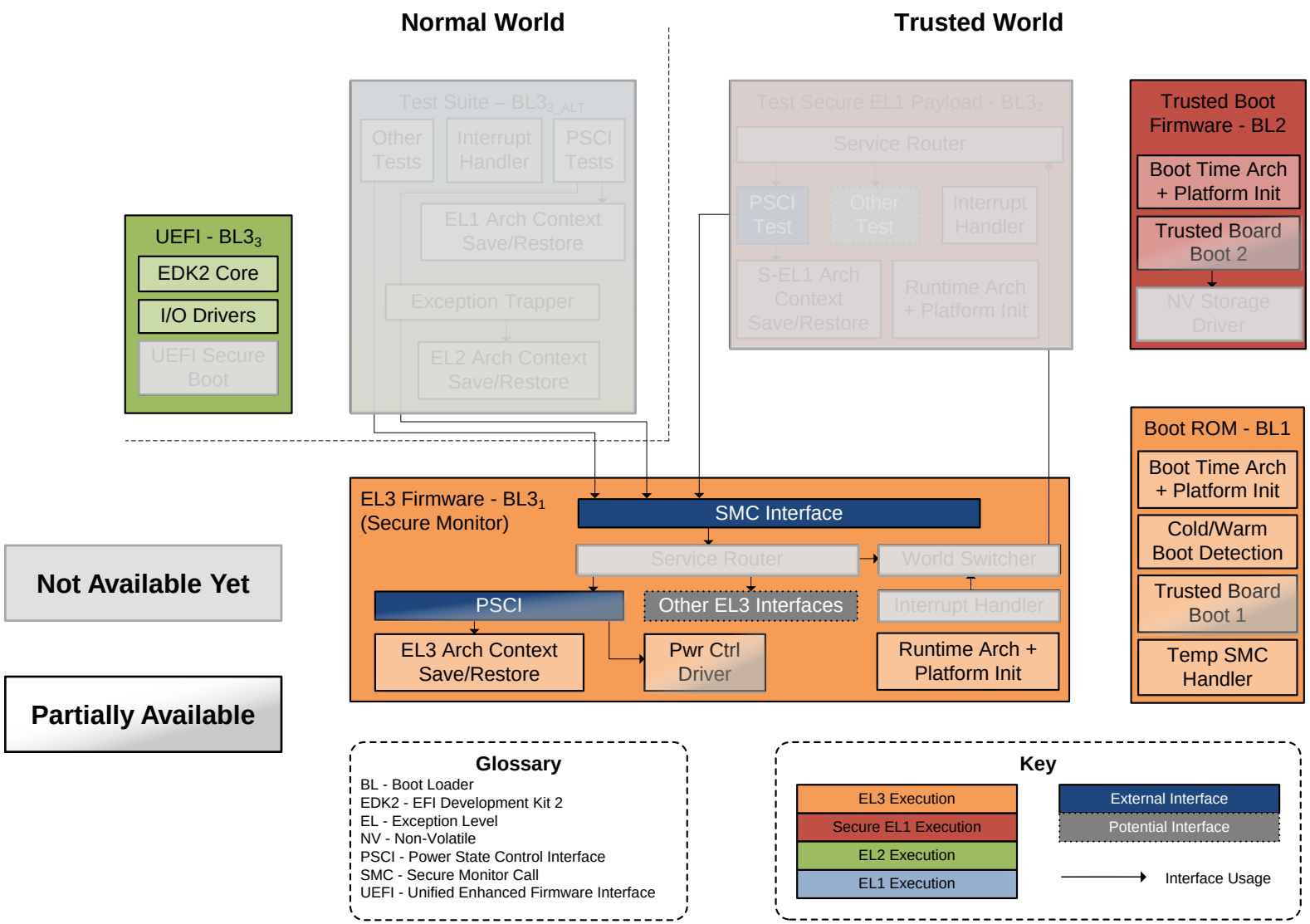


- SMC Calling Convention Interface (ARM DEN 0028A) is gateway to:
  - Runtime EL3 Firmware
  - Trusted OS / TEE services
- Power State Coordination Interface (PSCI) (ARM DEN 0022B.b)
  - Transported by SMC calls
- Also see ARMv8-A Architecture Manual (AR150-DA-70000)

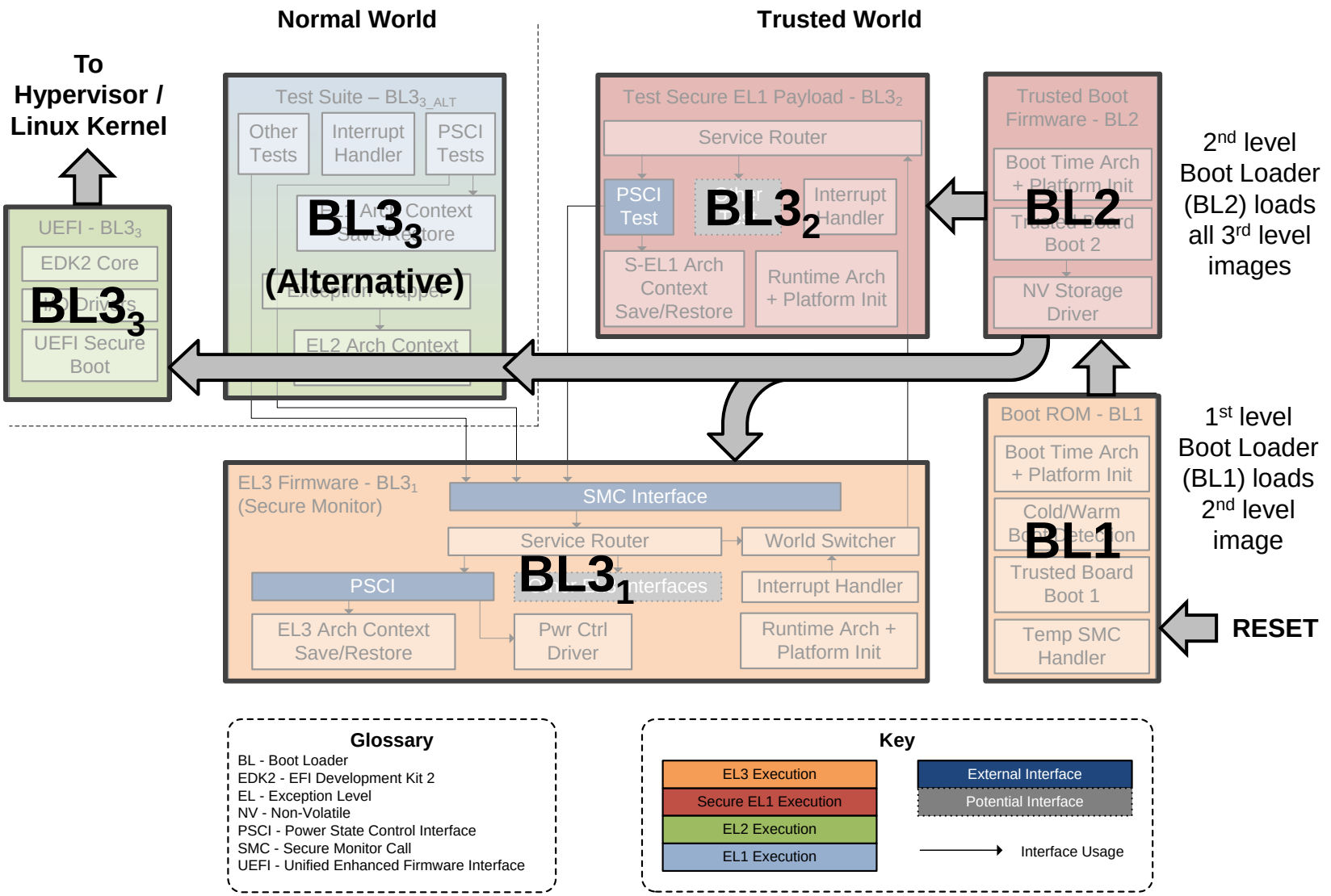
ARM Trusted Firmware Architecture



ARM Trusted Firmware Architecture



ARM Trusted Firmware Boot Flow



Current Memory Usage on FVP

Storage

- Secure ROM
  - BL1 code
- Semi-hosting
  - BL1, BL2, $BL3_1$ and Linux kernel code.
- NOR
  - UEFI code
- Virtio Block
  - Linux file system
  - Intend to use this for BL images instead of semi-hosting

Current Memory Usage on FVP

Storage

- Secure ROM
  - BL1 code
- Semi-hosting
  - BL1, BL2, $BL3_1$ and Linux kernel code.
- NOR
  - UEFI code
- Virtio Block
  - Linux file system
  - Intend to use this for BL images instead of semi-hosting

Execution

- Secure ROM
  - BL1 code
- Secure SRAM
  - BL1 data (bottom)
  - BL2 code & data (top)
  - $BL3_1$ code & data (middle, above BL1)
- Secure DRAM
  - Hand-off structures between BL images
  - When secure_memory parameter is defined, will need to program TZC to access DRAM
- NOR
  - Early UEFI code
- DRAM
  - UEFI code and data
  - Linux code and data

Code Organisation

- `arch` - Architecture specific code (just AArch64 for now)
- `plat` - Platform specific code (i.e. porting layer)
- `bl<X>` - BL specific code
- `common` - architecture/platform neutral code used by all BLs
- `lib` - library functionality common to all other code
- `drivers` - e.g. CCI, UART, FVP power controller
- `include`
- `docs`
- `fdts` - not necessarily the final home for these!

- Conforms to Linux coding standard

COLD BOOT DEEP DIVE

CPU States

- **Running.** CPU is executing normally
- **Idle.** States where OS is aware of CPU being idle
  - Execution resumes if new OS thread needs scheduling or interrupt is received
  - **Idle Standby.** Typically entered via WFI/WFE and exited via wake-up event
    - All caches and memories required for execution remain powered on and coherent
    - All CPU state (context) is preserved
    - Execution resumes from line after WFI/WFE
  - **Idle Retention.** Similar to standby except cannot access debug registers
  - **Idle Power Down.** CPU is powered off
    - CPU state at each EL must be saved
    - Execution starts at the reset vector, after which CPU state at each EL must be restored
    - Local caches may be off
- **Off**. States where the OS is not using the CPU for scheduling
  - Execution starts from the reset vector and no CPU state is restored
  - Enabling a CPU in this state requires a **Hotplug**

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

Always do basic CPU init

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

If PSCI provided an entrypoint, then jump to $BL3_1$ (either hotplug or resume from idle)

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

If there's no entrypoint and this is the primary CPU, then continue with cold boot

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

Otherwise put the secondary CPU in a safe state (e.g. On FVP, power off CPU)

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Jump to the main function. Returning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

Initialize the secure memory used by BL1

(On FVP, just zero out the entrypoint mailboxes)

```
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Jump to the main function. Returning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack
...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack
                                                  ----
                                                  it
                                                  ----
    /* ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

Create a small stack in "always uncached" memory to allow "C" code execution as soon as possible

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Jump to the main function. Returning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

"early" means "before MMU is enabled"

In this case, just calculate extents of memory

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Jump to the main function. Returning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

Now create simple page tables and enable MMU / caches

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack
```

Create a stack in normal memory

```
    ---------------
    urning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20

cb_init_panic:
    b   cb_init_panic
```

./plat/fvp/aarch64/bl1_plat_helpers.S

```
platform_cold_boot_init:; .type platform_cold_boot_init, %function
    mov x20, x0
    bl  platform_mem_init
    bl  read_mpidr
    mov x19, x0

    /* ---------------------------------------------
     * Give ourselves a small coherent stack to
     * ease the pain of initializing the MMU and
     * CCI in assembler
     * ---------------------------------------------
     */
    bl  platform_set_coherent_stack

...
    /* ---------------------------------------------
     * Architectural init. can be generic e.g.
     * enabling stack alignment and platform spec-
     * ific e.g. MMU & page table setup as per the
     * platform memory map. Perform the latter here
     * and the former in bl1_main.
     * ---------------------------------------------
     */
    bl  bl1_early_platform_setup
    bl  bl1_plat_arch_setup
```

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Jump to the main function. Returning from it
     * is a terminal error.
     * ---------------------------------------------
     */
    blr x20
cb_init_panic:
    b   cb_init_panic
```

Branch to main "C" function, `bl1_main()`

./bl1/bl1_main.c

```
void bl1_main(void)
{
    unsigned long sctlr_el3 = read_sctlr();
    unsigned long bl2_base;
    unsigned int load_type = TOP_LOAD, spsr;
    meminfo bl1_tzram_layout, *bl2_tzram_layout = 0x0;
...
    /* Perform remaining generic architectural setup from EL3 */
    bl1_arch_setup();

    /* Perform platform setup in BL1. */
    bl1_platform_setup();
...
    /*
     * Find out how much free trusted ram remains after BL1 load
     * & load the BL2 image at its top
     */
    bl1_tzram_layout = bl1_get_sec_mem_layout();
    bl2_base = load_image(&bl1_tzram_layout,
                  (const char *) BL2_IMAGE_NAME,
                  load_type, BL2_BASE);
...
    if (bl2_base) {
        bl1_arch_next_el_setup();
        spsr = make_spsr(MODE_EL1, MODE_SP_ELX, MODE_RW_64);
        printf("Booting trusted firmware boot loader stage 2\n\r");
...
        run_image(bl2_base, spsr, SECURE, bl2_tzram_layout, 0);
    }
...
    printf("Failed to load boot loader stage 2 (BL2) firmware.\n\r");
    return;
}
```

./bl1/bl1_main.c

```
void bl1_main(void)
{
    unsigned long sctlr_el3 = read_sctlr();
    unsigned long bl2_base;
    unsigned int load_type = TOP_LOAD, spsr;
    meminfo bl1_tzram_layout, *bl2_tzram_layout = 0x0;
...
    /* Perform remaining generic arch
    bl1_arch_setup();

    /* Perform platform setup in BL1
    bl1_platform_setup();
...
    /*
     * Find out how much free trusted ram remains after BL1 load
     * & load the BL2 image at its top
     */
    bl1_tzram_layout = bl1_get_sec_mem_layout();
    bl2_base = load_image(&bl1_tzram_layout,
                  (const char *) BL2_IMAGE_NAME,
                  load_type, BL2_BASE);
...
    if (bl2_base) {
        bl1_arch_next_el_setup();
        spsr = make_spsr(MODE_EL1, MODE_SP_ELX, MODE_RW_64);
        printf("Booting trusted firmware boot loader stage 2\n\r");
...
        run_image(bl2_base, spsr, SECURE, bl2_tzram_layout, 0);
    }
...
    printf("Failed to load boot loader stage 2 (BL2) firmware.\n\r");
    return;
}
```

Do remaining architectural / platform setup now we are executing in normal memory with MMU / caches enabled

E.g. control registers, generic timer, CCI snoops, console, ...

./bl1/bl1_main.c

```
void bl1_main(void)
{
    unsigned long sctlr_el3 = read_sctlr();
    unsigned long bl2_base;
    unsigned int load_type = TOP_LOAD, spsr;
    meminfo bl1_tzram_layout, *bl2_tzram_layout = 0x0;
...
    /* Perform remaining generic architectural setup from EL3 */
    bl1_arch_setup();

    /* Perform platform setup in BL1. */
    bl1_platform_setup();
...
    /*
     * Find out how much free trusted ram remains after BL1 load
     * & load the BL2 image at its top
     */
    bl1_tzram_layout = bl1_get_sec_mem_layout();
    bl2_base = load_image(&bl1_tzram_layout,
                  (const char *) BL2_IMAGE_NAME,
                  load_type, BL2_BASE);
...
    if (bl2_base) {
        bl1_arch_next_el_setup();
        spsr = make_spsr(MODE_EL1, MODE_SP_ELX, MODE_RW_64);
        printf("Booting trusted firmware boot loader stage 2\n\r");
...
        run_image(bl2_base, spsr, SECURE, bl2_tzram_layout, 0);
    }
...
    printf("Failed to load boot loader stage 2 (BL2) firmware.\n\r");
    return;
}
```

Calculate where to load BL2, load it, then run it

BL2 and $BL3_1$

- Entrypoints are similar to BL1's `platform_cold_boot_init()`
  - Create a small stack in coherent / always uncached memory)
  - Early platform setup (e.g. unpack image hand-off information)
  - Platform-specific architectural setup (e.g. enable MMU / caches)
  - Create a stack in normal memory
  - Branch to main "C" function for remaining platform / arch setup
- BL2 loads $BL3_{<X>}$ images similarly to how BL1 loads BL2
- BL2 passes information about $BL3_3$ (e.g. UEFI) to $BL3_1$, before running $BL3_1$
  - Means that $BL3_1$ can jump to $BL3_3$ without going back to BL2
  - Has to go via BL1 SMC handler to jump from Secure EL1 to EL3
    - See `SynchronousExceptionA64` in ./bl1/aarch64/early_exceptions.S

$BL3_1$ Initialization

- Can override any BL1 initialization
  - Reinitializes exception vectors, MMU, control registers, etc ...
- Installs runtime SMC handler
- Initializes platform for normal world software
  - Initializes GIC, runtime services (e.g. PSCI)
- Returns from exception into normal world boot loader, $BL3_3$ (e.g UEFI)

PSCI DEEP DIVE

PSCI Status

- Work in progress. Key functions for boot and hotplug are functional. Idle is next on the radar

PSCI Function	Implementation Status
PSCI_VERSION	OK
CPU_ON	OK
CPU_SUSPEND	NOK (code present not ready)
CPU_OFF	OK
AFFINTY_INFO	OK
MIGRATE	Not present
MIGRATE_INFO_TYPE	Not present
SYSTEM_OFF	Not present
SYSTEM_RESET	Not present

PSCI Topology

- PSCI needs to build a map of system topology
  - How many clusters, cores per cluster etc
  - Used for last man tracking

- Topology information is provided by the platform using the following functions:

```
int plat_get_max_afflvl()

int plat_get_aff_count(unsigned int aff_lvl,
                       unsigned long mpidr)

unsigned int plat_get_aff_state(unsigned int aff_lvl,
                                unsigned long mpidr)
```

PSCI Topology

`plat_get_max_afflvl()`

Returns highest affinity level implemented by the platform

e.g. For FVP models:

```
int plat_get_max_afflvl()
{
        return MPIDR_AFFLVL1;
}
```

PSCI Topology

`plat_get_aff_count(aff_lvl, mpidr)`

Given an MPIDR and a level of affinity return how many instances are implemented at that affinity level

For example consider 2x3 system: 2 clusters, 2 cores in cluster 0, and 3 in cluster 1

mpidr	aff_lvl	Return Value
0000 (Cluster 0, Core 0) 0001 (Cluster 0, Core 1)	0	2 (two affinity level 0 instances, or cores, in cluster 0)
	1	2 (There are two affinity level 1 instances or clusters in the system)
0100 (Cluster 1, Core 0) 0101 (Cluster 1, Core 1)	0	3 (three affinity level 0 instances, or cores, in cluster 1)
	1	2 (There are two affinity level 1 instances or clusters in the system)

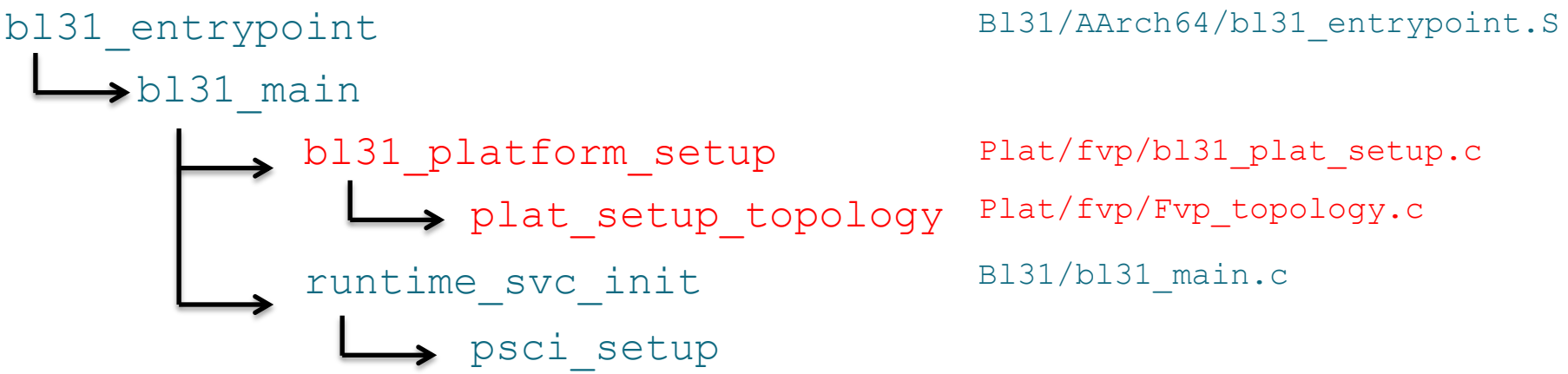
PSCI Topology

```
plat_get_aff_state(aff_lvl, mpidr)
```

- Returns whether an affinity instance is present or absent
- You can use it to deal with hierarchies that are asymmetric
  - For example a cluster and a single core sharing an interconnect
  - Saves on having to take locks for affinity levels that don't exist

FVP topology

- FVP model sets up topology information as part of cold boot path (called from primary CPU)



- `plat_setup_topology()` sets up necessary data to allow the following to work
  - `plat_get_max_aff_lvl()`, `plat_get_aff_count()`,`plat_get_aff_state()`
- $BL3_1$ then moves on to set up PSCI

PSCI Deep Dive

- After cold boot path calls `plat_setup_topology()`, it calls `psci_setup()`
- This functions create a topology map for the system based on the platform specific functions

- Map is an array of `aff_map_node` pointers

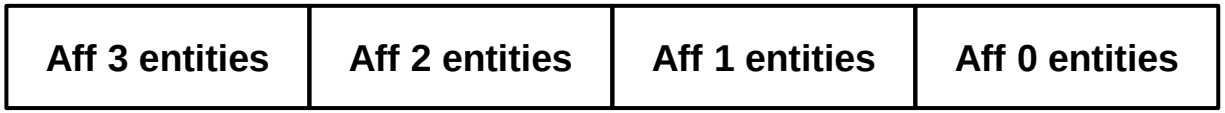
```
typedef struct {
        unsigned long mpidr;     //mpidr of node
        unsigned char state;     //[present||absent]||[PSCI state]
        char level;              //aff level
        unsigned int data;       //cookie (holds index into
                                 //non-secure data for CPU_ON/CPU_SUSPEND
        bakery_lock lock;        //Lock for node
} aff_map_node;
```

(`bakery_lock` will be replaced with a more abstract lock API)

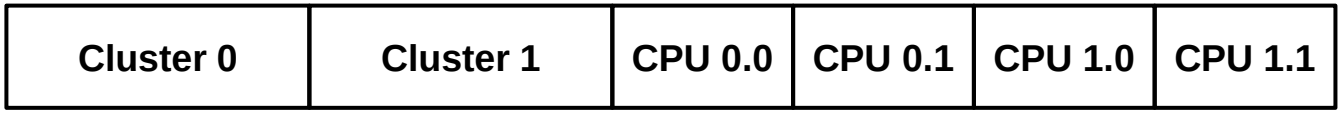
See `psci_setup.c/psci_common.c/psci_private.h`

PSCI Topology

- `psci_aff_map` holds topology tree (array of `aff_map_node` (s))
  - Held in device ordered memory
- Array is populated in a breadth first way



- e.g. for a 2x2 system



- Additional affinity information arrays:

`psci_aff_limits` : indices to start and end of each affinty level in topology tree

`psci_ns_entry_info`: array of `ns_entry_info`. Structure to hold entry information into EL[2|1] for `CPU_ON/CPU_SUSPEND`

`psci_secure_context`: array of `secure_context`. Structure to hold secure context information that needs saving when powering down

PSCI Topology

- `psci_setup_up` completes by:
  - Initialising the state of the primary CPU (and containing higher affinity levels, e.g cluster) to `PSCI_STATE_ON`
    - All others default to `PSCI_STATE_OFF`
  - Calls into platform to set up platform specific PSCI operations

```
plat\fvp\Fvp_pm.c
int platform_setup_pm(plat_pm_ops **plat_ops)
{
        *plat_ops = &fvp_plat_pm_ops;
        return 0;
}

static plat_pm_ops fvp_plat_pm_ops = {
   0,                                  // standby (not used in FVP)
   fvp_affinst_on,                     // cpu_on will come here
   fvp_affinst_off,                    // cpu_off will come here
   fvp_affinst_suspend,                // cpu_suspend
   fvp_affinst_on_finish,              // called on wake up path of cpu being turned on
   fvp_affinst_suspend_finish,         // called on wake up path of cpu waking from suspend
};
```

__psci_cpu_off

common/psci/psci_entry.S

```
__psci_cpu_off:
        func_prologue
        sub     sp, sp, #0x10
        stp     x19, x20, [sp, #0]
        mov     x19, sp
        bl      read_mpidr
        bl      platform_set_coherent_stack << Switch stack
        bl      psci_cpu_off
        mov     x1, #PSCI_E_SUCCESS
        cmp     x0, x1
        b.eq    final_wfi
        mov     sp, x19
        ldp     x19, x20, [sp,#0]
        add     sp, sp, #0x10
        func_epilogue
        ret
```

__psci_cpu_off

common/psci/psci_entry.S

```
__psci_cpu_off:
        func_prologue
        sub     sp, sp, #0x10
        stp     x19, x20, [sp, #0]
        mov     x19, sp
        bl      read_mpidr
        bl      platform_set_coherent_stack
        bl      psci_cpu_off      << do work of switching off
        mov     x1, #PSCI_E_SUCCESS
        cmp     x0, x1
        b.eq    final_wfi
        mov     sp, x19
        ldp     x19, x20, [sp,#0]
        add     sp, sp, #0x10
        func_epilogue
        ret
```

psci_cpu_off

common/psci/psci_main.c

```
int psci_cpu_off(void)
{
…
        int target_afflvl = get_max_afflvl();

        mpidr = read_mpidr();

        /*
         * Traverse from the highest to the lowest affinity level. When the
         * lowest affinity level is hit, all the locks are acquired. State
         * management is done immediately followed by cpu, cluster ...
         * ..target_afflvl specific actions as this function unwinds back.
         */
        rc = psci_afflvl_off(mpidr, target_afflvl, MPIDR_AFFLVL0);

        if (rc != PSCI_E_SUCCESS) {
                assert(rc == PSCI_E_DENIED);
        }

        return rc;
}
```

psci_afflvl_off – FVP success

Take Cluster Node Lock — **Affinity Level 1/Cluster**

Not at AffinityLevel0 so recurse down a level

Take CPU Node Lock — **Affinity Level 0/CPU**

cpustate = CPU_OFF

if last man cluster state = OFF

`psci_afflvl0_off(…)` — //bl31/common/psci_afflvl_off.c

flush PoU

`fvp_affinst_off(..,AffLevel0,…)` — //plat/fvp/Fvp_pm.c

take core out of coherency

prevent IRQ spurious wakeups

power power controller

Release CPU Node Lock

`psci_afflvl1_off(…)` — //bl31/common/psci_afflvl_off.c

if cluster_state == OFF

flush to PoC

`fvp_affinst_off(..,AffLevel1,…)` — //plat/fvp/Fvp_pm.c

disable cci

power power controller

Release Cluster Node Lock — **Affinity Level 1/Cluster**

psci_afflvl_off

common/psci/psci_aff_lvl_off.c

```
int psci_afflvl_off(unsigned long mpidr,int
                    cur_afflvl, int tgt_afflvl)
{
…
    bakery_lock_get(mpidr, &aff_node->lock);

    /* Keep the old state and the next one handy */
    prev_state = psci_get_state(aff_node->state);
    next_state = PSCI_STATE_OFF;

    /*
     * We start from the highest affinity level
     *  and work our way
     * downwards to the lowest i.e. MPIDR_AFFLVL0.
     */
    if (aff_node->level == tgt_afflvl) {
        psci_change_state(mpidr,
                  tgt_afflvl,
                  get_max_afflvl(),
                  next_state);
    } else {
        rc = psci_afflvl_off(mpidr, level - 1,
tgt_afflvl);
            if (rc != PSCI_E_SUCCESS) {
                psci_set_state(aff_node->state,
prev_state);
                goto exit;
            }
        }

…
        rc = psci_afflvl_off_handlers[level](mpidr,
aff_node);
        if (rc != PSCI_E_SUCCESS) {
            psci_set_state(aff_node->state, prev_state);
            goto exit;
        }

…
exit:
        bakery_lock_release(mpidr, &aff_node->lock);
```

When we get to CPU Level (Aff0) we set state. This also sets higher affinity level states if last man

psci_afflvl_off

common/psci/psci_aff_lvl_off.c

```
int psci_afflvl_off(unsigned long mpidr,int
                    cur_afflvl, int tgt_afflvl)
{
…
    bakery_lock_get(mpidr, &aff_node->lock);

    /* Keep the old state and the next one handy */
    prev_state = psci_get_state(aff_node->state);
    next_state = PSCI_STATE_OFF;

    /*
     * We start from the highest affinity level
     *  and work our way
     * downwards to the lowest i.e. MPIDR_AFFLVL0.
     */
    if (aff_node->level == tgt_afflvl) {
        psci_change_state(mpidr,
                  tgt_afflvl,
                  get_max_afflvl(),
                  next_state);
    } else {
        rc = psci_afflvl_off(mpidr, level - 1,
tgt_afflvl);
            if (rc != PSCI_E_SUCCESS) {
                psci_set_state(aff_node->state,
prev_state);
                goto exit;
            }
        }

…
        rc = psci_afflvl_off_handlers[level](mpidr,
aff_node);
        if (rc != PSCI_E_SUCCESS) {
            psci_set_state(aff_node->state, prev_state);
            goto exit;
        }

…
exit:
    bakery_lock_release(mpidr, &aff_node->lock);
    return rc;
}
```

Handlers do actual powering down

__psci_cpu_off

common/psci/psci_entry.S

```
__psci_cpu_off:
        func_prologue
        sub     sp, sp, #0x10
        stp     x19, x20, [sp, #0]
        mov     x19, sp
        bl      read_mpidr
        bl      platform_set_coherent_stack
        bl      psci_cpu_off
        mov     x1, #PSCI_E_SUCCESS
        cmp     x0, x1
        b.eq    final_wfi << all OK final WFI
        mov     sp, x19
        ldp     x19, x20, [sp,#0]
        add     sp, sp, #0x10
        func_epilogue
        ret
```

__psci_cpu_off

common/psci/psci_entry.S

```
__psci_cpu_off:
        func_prologue
        sub     sp, sp, #0x10
        stp     x19, x20, [sp, #0]
        mov     x19, sp
        bl      read_mpidr
        bl      platform_set_coherent_stack
        bl      psci_cpu_off
        mov     x1, #PSCI_E_SUCCESS
        cmp     x0, x1
        b.eq    final_wfi
        mov     sp, x19  << else switch stack back and return
        ldp     x19, x20, [sp,#0]
        add     sp, sp, #0x10
        func_epilogue
        ret
```

psci_cpu_on

common/psci/psci_main.c

```
int psci_cpu_on(unsigned long target_cpu, unsigned long entrypoint,
            unsigned long context_id)

{
      int rc;
      unsigned int start_afflvl, target_afflvl;

      /* Determine if the cpu exists of not */
      rc = psci_validate_mpidr(target_cpu, MPIDR_AFFLVL0);
      if (rc != PSCI_E_SUCCESS) {
            goto exit;
      }

      start_afflvl = get_max_afflvl();
      target_afflvl = MPIDR_AFFLVL0;
      rc = psci_afflvl_on(target_cpu,
                  entrypoint,
                  context_id,
                  start_afflvl,
                  target_afflvl);

exit:
      return rc;
}
```

Basic error checking

Heavy lifting

psci_afflvl_on – FVP success

Take Cluster Node Lock — **Affinity Level 1/Cluster**

Take CPU Node Lock — **Affinity Level 0/CPU**

Affinity Level 1/Cluster

`psci_afflvl1_on(…)` //bl31/common/psci_afflvl_on.c

`fvp_affinst_on(..,AffLevel1,…)` //plat/fvp/Fvp_pm.c

basic validation

`psci_afflvl0_on(…)` //bl31/common/psci_afflvl_on.c

store ns_entry_point and context_id (passed from by OSPM)

`fvp_affinst_on(..,AffLevel0,…)` //plat/fvp/Fvp_pm.c

Wait for any pending off to complete (CPU that you are turning ON, could have been turning itself OFF)

Set up a mailbox so booting core takes warm boot path

program power controller

Affinity Level 0/CPU

Change CPU Node state to ON_PENDING — **Affinity Level 0/CPU**

Change Cluster Node state to ON_PENDING — **Affinity Level 1/Cluster**

Release CPU Node Lock — **Affinity Level 0/CPU**

Release Cluster Node Lock — **Affinity Level 1/Cluster**

psci_afflvl_on

common/psci/psci_aff_lvl_on.c

```
int psci_afflvl_on(unsigned long target_cpu,
                   unsigned long entrypoint,
                   unsigned long context_id,
                   int current_afflvl,
                   int target_afflvl)
{
…
```

Lock

```
    for (level = current_afflvl;
         level >= target_afflvl; level--) {
         aff_node = psci_get_aff_map_node…
         if (aff_node)
              bakery_lock_get(mpidr, &aff_node->lock);
    }
```

Call each level's on handler

```
…
    for (level = current_afflvl;
         level >= target_afflvl; level--) {
…        psci_afflvl_on_handlers[level](target_cpu,
                                        aff_node,
                                        entrypoint,
                                        context_id);
              }
```

Set Hierarchy to ON_PENDING

```
/*
 * State management: Update the states …
 */
psci_change_state(target_cpu,
                  target_afflvl,
                  get_max_afflvl(),
                  PSCI_STATE_ON_PENDING);
```

```
exit:
…
```

Unlock

```
for (level = target_afflvl;
     level <= current_afflvl; level++) {
     …
          bakery_lock_release(mpidr,
               &aff_node->lock);
…
}
```

fvp_affinst_on

plat/fvp/Fvp_pm.c

```
int fvp_affinst_on(unsigned long mpidr,
            unsigned long sec_entrypoint,
            unsigned long ns_entrypoint,
            unsigned int afflvl,
            unsigned int state)
{
…
```

Ensure entry point is valid

```
    if (ns_entrypoint < DRAM_BASE) {
        rc = PSCI_E_INVALID_PARAMS;
        goto exit;
    }
```

```
…

    if (afflvl != MPIDR_AFFLVL0)
        goto exit;
…
```

Deal with potential race with CPU_OFF

```
/*
 * Ensure that we do not cancel an inflight
 *  power off request
 * for the target cpu. That would leave
  * it in a zombie wfi…
 */
do {
    psysr = fvp_pwrc_read_psysr(mpidr);
} while (psysr & PSYSR_AFF_L0);
```

Set up warm boot

```
linear_id = platform_get_core_pos(mpidr);
fvp_mboxes = (mailbox *) (TZDRAM_BASE +
                MBOX_OFF);
fvp_mboxes[linear_id].value = sec_entrypoint;
flush_dcache_range(
     (unsigned long) &fvp_mboxes[linear_id],
       sizeof(unsigned long));
```

Program power controller

```
fvp_pwrc_write_pponr(mpidr);
…
```

./bl1/aarch64/bl1_entrypoint.S

```
reset_handler:; .type reset_handler, %function
    /* ---------------------------------------------
     * Perform any processor specific actions upon
     * reset e.g. cache, tlb invalidations etc.
     * ---------------------------------------------
     */
    bl  cpu_reset_handler

_wait_for_entrypoint:
    /* ---------------------------------------------
     * Find the type of reset and jump to handler
     * if present. If the handler is null then it is
     * a cold boot. The primary cpu will set up the
     * platform while the secondaries wait for
     * their turn to be woken up
     * ---------------------------------------------
     */
    bl  read_mpidr
    bl  platform_get_entrypoint
    cbnz    x0, _do_warm_boot
    bl  read_mpidr
    bl  platform_is_primary_cpu
    cbnz    x0, _do_cold_boot

    /* ---------------------------------------------
     * Perform any platform specific secondary cpu
     * actions
     * ---------------------------------------------
     */
    bl  plat_secondary_cold_boot_setup
    b   _wait_for_entrypoint
```

```
_do_cold_boot:
    /* ---------------------------------------------
     * Initialize platform and jump to our c-entry
     * point for this type of reset
     * ---------------------------------------------
     */
    adr x0, bl1_main
    bl  platform_cold_boot_init
    b   _panic

_do_warm_boot:
    /* ---------------------------------------------
     * Jump to BL31 for all warm boot init.
     * ---------------------------------------------
     */
    blr x0
```

If PSCI provided an entrypoint, then jump to $BL3_1$ (either hotplug or resume from idle)

common/psci/psci_entry.S

```
    /* ---------------------------------------------------
     * This cpu has been physically powered up. Depending
     * upon whether it was resumed from suspend or simply
     * turned on, call the common power on finisher with
     * the handlers (chosen depending upon original state).
     * For ease, the finisher is called with coherent
     * stacks. This allows the cluster/cpu finishers to
     * enter coherency and enable the mmu without running
     * into issues. We switch back to normal stacks once
     * all this is done.
     * ---------------------------------------------------
     */
psci_aff_on_finish_entry:
    adr x23, psci_afflvl_on_finishers
    b   psci_aff_common_finish_entry

psci_aff_suspend_finish_entry:
    adr x23, psci_afflvl_suspend_finishers

psci_aff_common_finish_entry:
    adr x22, psci_afflvl_power_on_finish
    bl  read_mpidr
    mov x19, x0
    bl  platform_set_coherent_stack

    /* ---------------------------------------------
     * Call the finishers starting from affinity
     * level 0.
     * ---------------------------------------------
     */
    bl  get_max_afflvl
    mov x3, x23
    mov x2, x0
    mov x0, x19
    mov x1, #MPIDR_AFFLVL0
    blr x22
    mov x21, x0
```

Call psci_afflvl_power_on_finish on coherent stacks

```
    /* ---------------------------------------------
     * Give ourselves a stack allocated in Normal
     * -IS-WBWA memory
     * ---------------------------------------------
     */
    mov x0, x19
    bl  platform_set_stack

    /* ---------------------------------------------
     * Restore the context id. value
     * ---------------------------------------------
     */
    mov x0, x21

    /* ---------------------------------------------
     * Jump back to the non-secure world assuming
     * that the elr and spsr setup has been done
     * by the finishers
     * ---------------------------------------------
     */
    eret
_panic:
    b   _panic
```

Jump into OSPM entry point

psci_afflvl_power_on_finish

Take CPU Node Lock //bl31/common/psci_common.c

`psci_afflvl0_on_finish(…)` //bl31/common/psci_afflvl_on.c

`fvp_affinst_on_finish(..,AffLevel0,…)` //plat/fvp/Fvp_pm.c

turn on intra cluster coherency

zero out mailbox, enable GIC, enable access to system counter

install exception handlers,enable mmu and caching, EL3 setup

`psci_get_ns_entry_info()` //bl31/common/psci_common.c

set up return non-secure Exception Level

Take Cluster Node Lock **Affinity Level 0/CPU**

`psci_afflvl1_on_finish(…)` //bl31/common/psci_afflvl_on.c

`fvp_affinst_on_finish(..,AffLevel1,…)` //plat/fvp/Fvp_pm.c

enable CCI

Affinity Level 1/Cluster

update CPU Node State to ON **Affinity Level 0/CPU**

update cluster Node State to ON **Affinity Level 1/Cluster**

Release Cluster Node Lock **Affinity Level 1/Cluster**

Release CPU Node Lock **Affinity Level 0/CPU**

Further reading…

- GitHub

  https://github.com/ARM-software/arm-trusted-firmware
- Usage Guide

  https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/user-guide.md
- Porting Guide

  https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/porting-guide.md
- SMC Calling Convention

  http://infocenter.arm.com/help/topic/com.arm.doc.den0028a/index.html
- PSCI spec

  http://infocenter.arm.com/help/topic/com.arm.doc.den0022b/index.html
- ARMv8 ARM

  http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487a.a_errata1/index.html