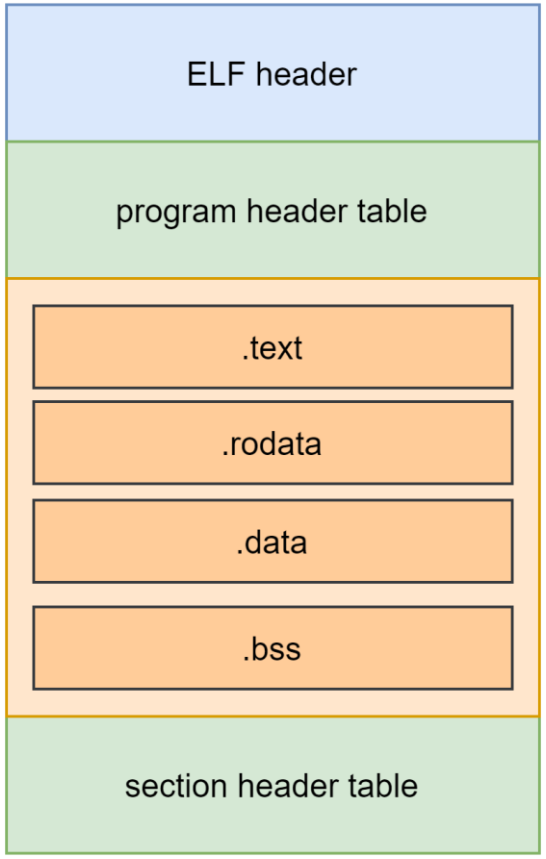
How to boot the kernel

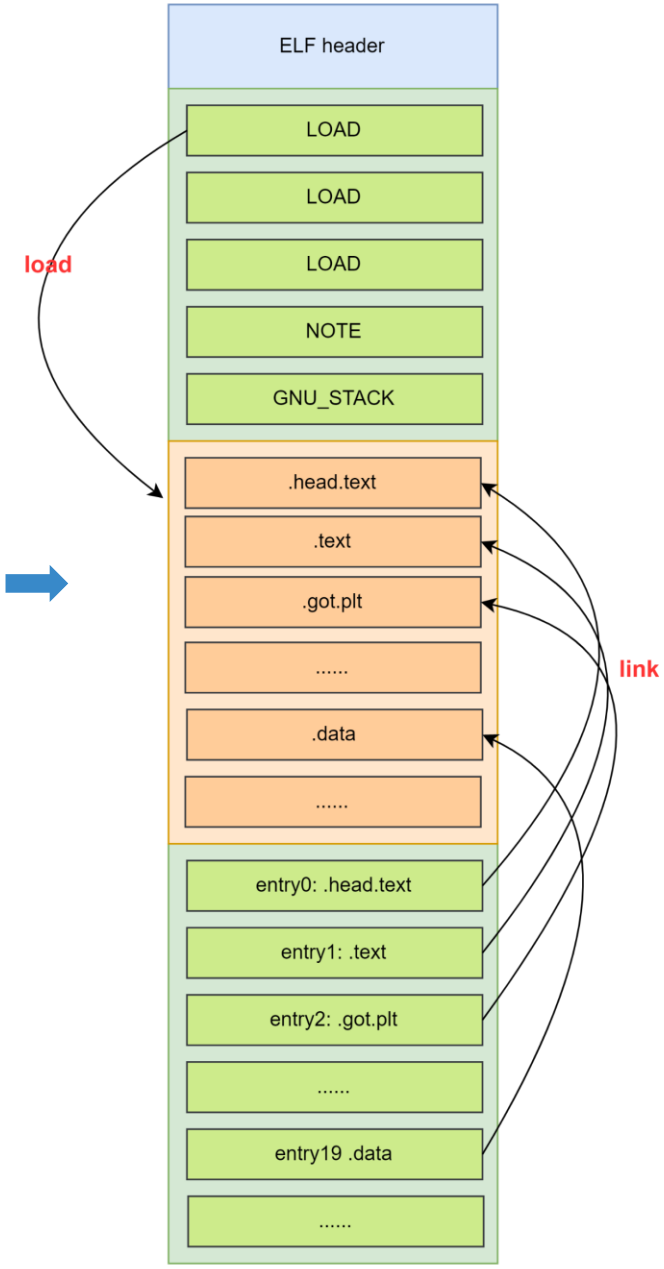
Peter Liu

vmlinux ELF




```
$ readelf -S vmlinux
There are 38 section headers, starting at offset 0x1d7c3678:

Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .head.text        PROGBITS        ffff800010000000  00010000
       0000000000010000  0000000000000000  AX       0     0     65536
  [ 2] .text             PROGBITS        ffff800010010000  00020000
       00000000012ac188  0000000000000008  AX       0     0     2048
  [ 3] .got.plt          PROGBITS        ffff8000112bc188  012cc188
       0000000000000018  0000000000000008  WA       0     0     8
  [ 4] .rodata           PROGBITS        ffff8000112c0000  012d0000
       00000000008ae430  0000000000000000  WA       0     0     4096
  ......
  [36] .strtab           STRTAB          0000000000000000  1d49cb60
       0000000000326979  0000000000000000           0     0     1
  [37] .shstrtab         STRTAB          0000000000000000  1d7c34d9
       0000000000000198  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

```
$ readelf -l vmlinux

Elf file type is DYN (Shared object file)
Entry point 0xffff800010000000
There are 5 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000010000 0xffff800010000000 0xffff800010000000
                 0x0000000001beacdc 0x0000000001beacdc  RWE    10000
  LOAD           0x0000000001c00000 0xffff800011c00000 0xffff800011c00000
                 0x00000000000c899c 0x00000000000c899c  R E    10000
  LOAD           0x0000000001cd0000 0xffff800011cd0000 0xffff800011cd0000
                 0x0000000000876200 0x0000000000905794  RW     10000
  NOTE           0x0000000001bfaca0 0xffff800011beaca0 0xffff800011beaca0
                 0x000000000000003c 0x000000000000003c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10


 Section to Segment mapping:
  Segment Sections...
   00     .head.text .text .got.plt .rodata .pci_fixup __ksymtab __ksymtab_g
   01     .init.text .exit.text .altinstructions
   02     .init.data .data..percpu .hyp.data..percpu .rela.dyn .data __bug_t
   03     .notes
   04
```



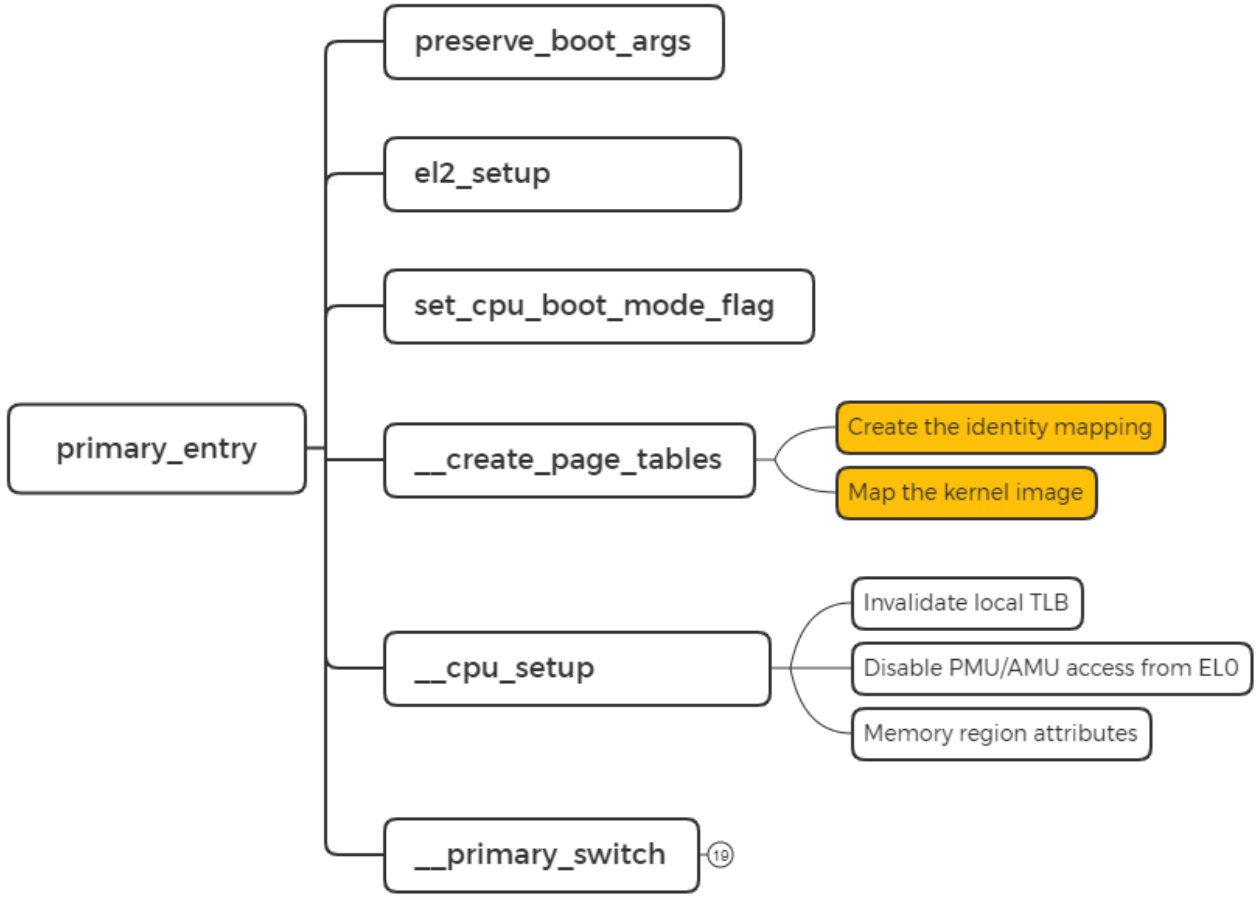
vmlinux.lds.S

arch/arm64/kernel/vmlinux.lds.S

head.S

```
        __HEAD
_head:
        /*
         * DO NOT MODIFY. Image header ex
         */
#ifdef CONFIG_EFI
        /*
         * This add instruction has no me
         * its opcode forms the magic "MZ
         */
        add     x13, x18, #0x16
        b       primary_entry
#else
        b       primary_entry
        .long   0
#endif
```

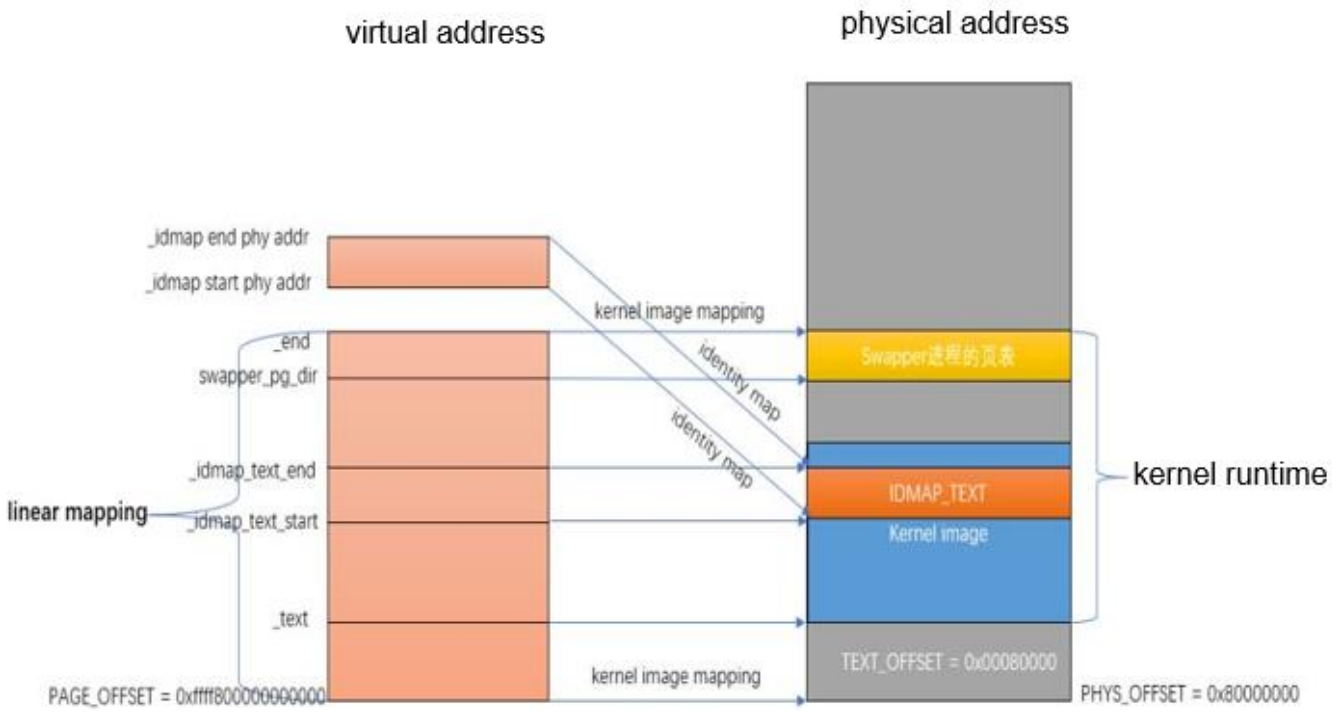
Save x0, x1, x2, x3 passed by bootloader into the boot_args array



__create_page_tables

1. **idmap_pg_dir** for MMU enablement code
   VA: Runtime __pa of section ".idmap.text"
   PA: Runtime __pa of section ".idmap.text“

2. **init_pg_dir** for kernel image mapping
   VA: KIMAGE_VADDR / **Compile time __va(text)**
   PA: Runtime __pa(_text) in DRAM



__cpu_setup

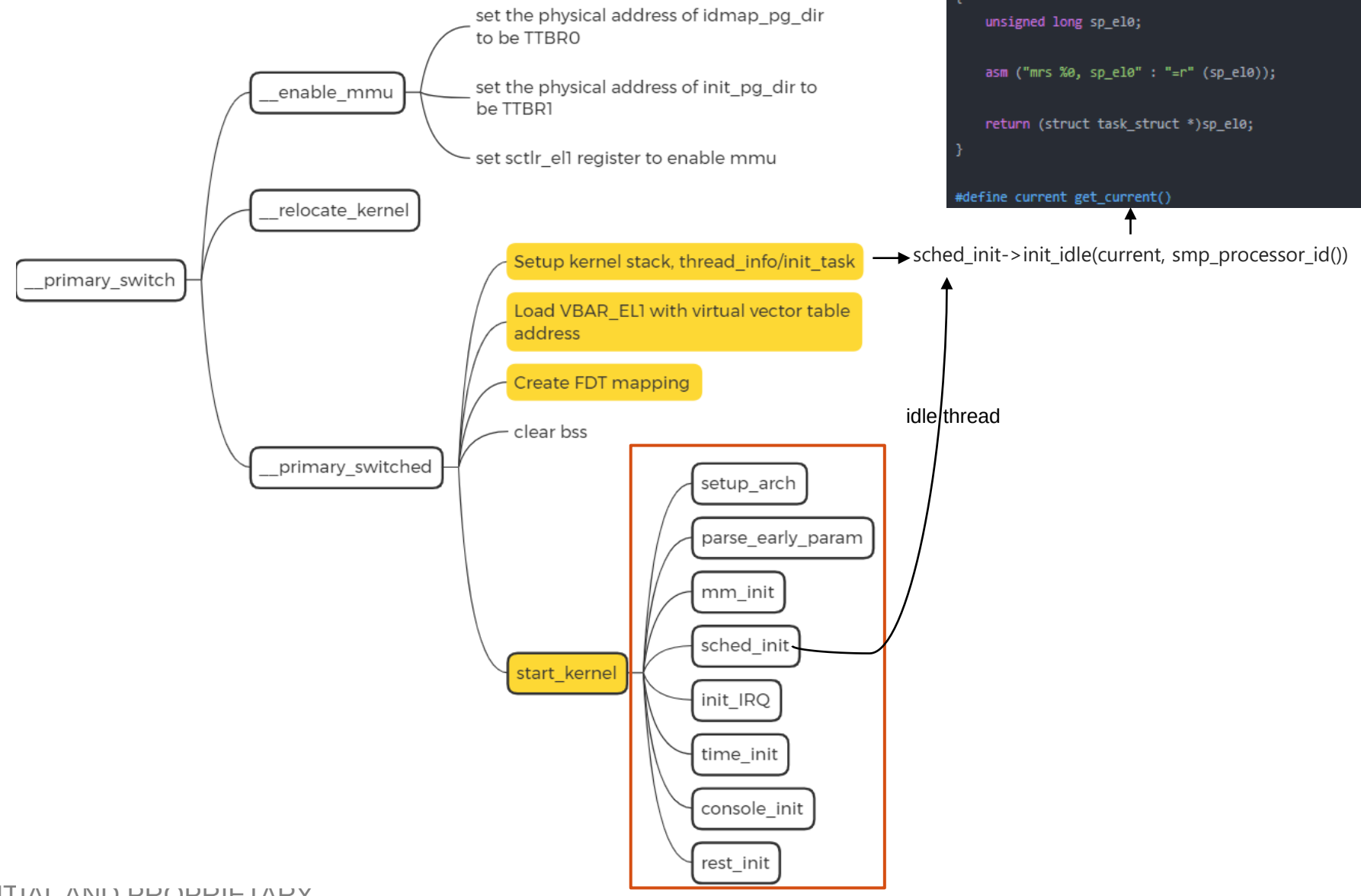
```
SYM_FUNC_START(__cpu_setup)
        tlbi    vmalle1                 // Invalidate local TLB
        dsb     nsh

        mov     x1, #3 << 20
        msr     cpacr_el1, x1           // Enable FP/ASIMD
        mov     x1, #1 << 12            // Reset mdscr_el1 and disable
        msr     mdscr_el1, x1           // access to the DCC from EL0
        isb                             // Unmask debug exceptions now,
        enable_dbg                      // since this is per-cpu
        reset_pmuserenr_el0 x1          // Disable PMU access from EL0
        reset_amuserenr_el0 x1          // Disable AMU access from EL0

        /*
         * Memory region attributes
         */
        mov_q   x5, MAIR_EL1_SET
        ......
        msr     mair_el1, x5
        /*
         * Set/prepare TCR and TTBR. We use 512GB (39-bit) address range for
         * both user and kernel.
         */
        mov_q   x10, TCR_TxSZ(VA_BITS) | TCR_CACHE_FLAGS | TCR_SMP_FLAGS | \
                        TCR_TG_FLAGS | TCR_KASLR_FLAGS | TCR_ASID16 | \
                        TCR_TBI0 | TCR_A1 | TCR_KASAN_FLAGS
        tcr_clear_errata_bits x10, x9, x5
        ......
```

Invalidate local tlb

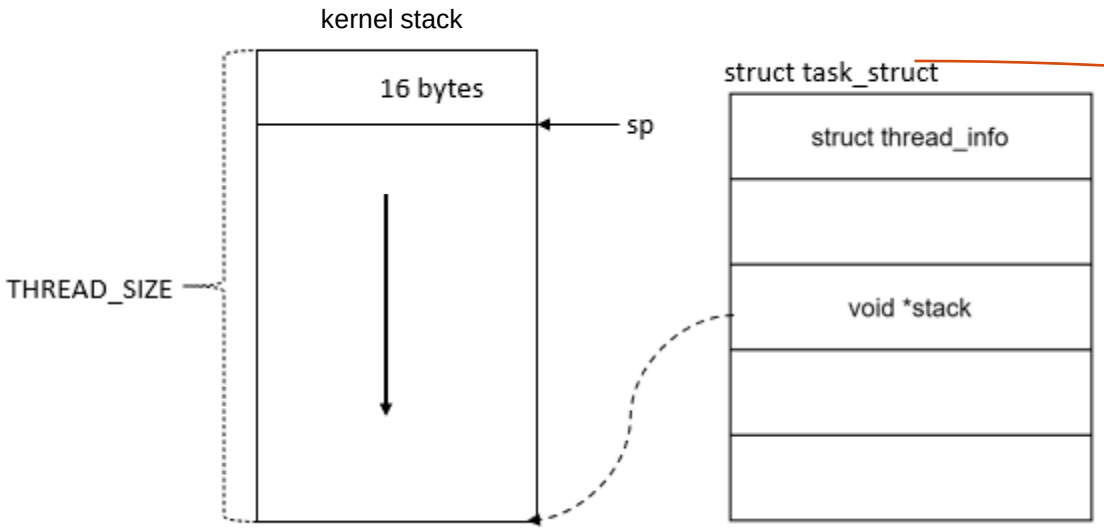
Memory region attributes

48-bit address range

__primary_switch



init_task



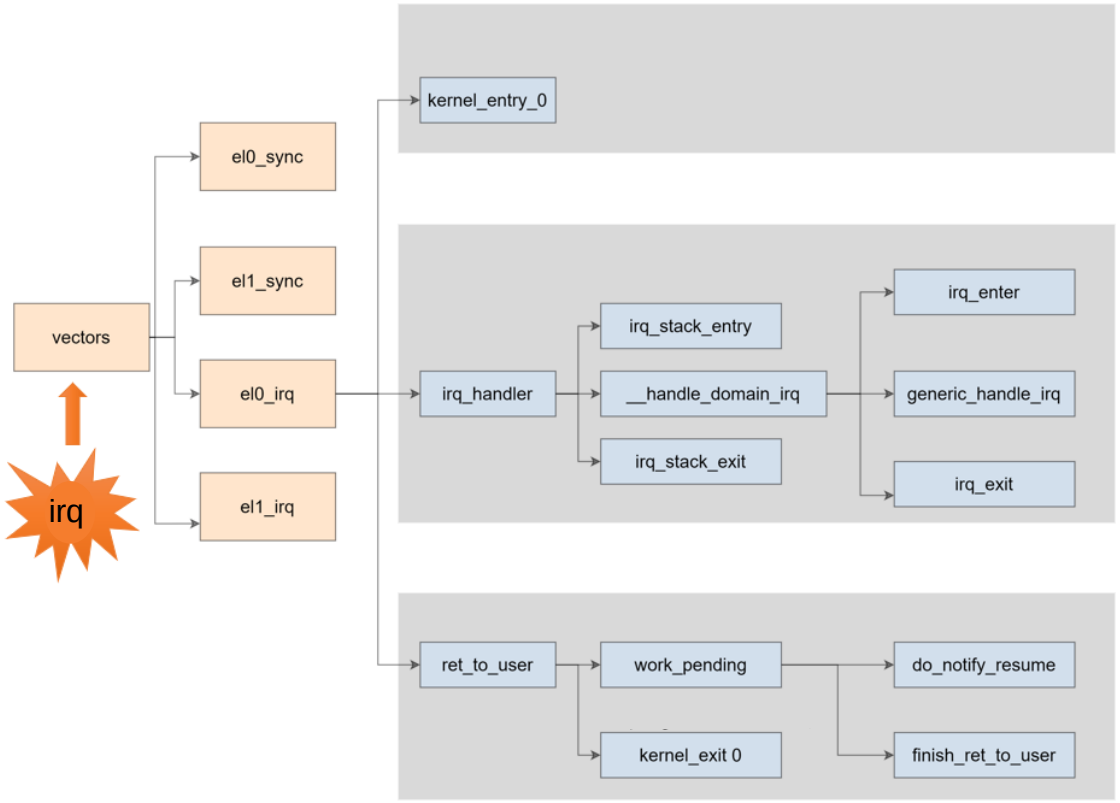

```c
/* include/linux/init_task.h */
#define INIT_TASK_COMM "swapper"

/* init/init_task.c */
struct task_struct init_task
#ifdef CONFIG_ARCH_TASK_STRUCT_ON_STACK
        __init_task_data
#endif
        __aligned(L1_CACHE_BYTES)
= {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        .thread_info    = INIT_THREAD_INFO(init_task),
        .stack_refcount = REFCOUNT_INIT(1),
#endif
        .state          = 0,
        .stack          = init_stack,
        .usage          = REFCOUNT_INIT(2),
        .flags          = PF_KTHREAD,
        .prio           = MAX_PRIO - 20,
        .static_prio    = MAX_PRIO - 20,
        .normal_prio    = MAX_PRIO - 20,
        .policy         = SCHED_NORMAL,
        .cpus_ptr       = &init_task.cpus_mask,
        .cpus_mask      = CPU_MASK_ALL,
        .nr_cpus_allowed= NR_CPUS,
        .mm             = NULL,
        .active_mm      = &init_mm,
  ......
        .comm           = INIT_TASK_COMM,
        .thread         = INIT_THREAD,
        .fs             = &init_fs,
        .files          = &init_files,
  ......
};
EXPORT_SYMBOL(init_task);
```

IRQ Vectors

Table D1-5 Vector offsets from vector table base address

Exception taken from	Offset for exception type			
	Synchronous	IRQ or vIRQ	FIQ or vFIQ	SError or vSError
Current Exception level with SP_EL0.	0x000[a]	0x080	0x100	0x180
Current Exception level with SP_ELx, x>0.	0x200[a]	0x280	0x300	0x380
Lower Exception level, where the implemented level immediately lower than the target level is using AArch64.[b]	0x400[a]	0x480	0x500	0x580
Lower Exception level, where the implemented level immediately lower than the target level is using AArch32.[b]	0x600[a]	0x680	0x700	0x780

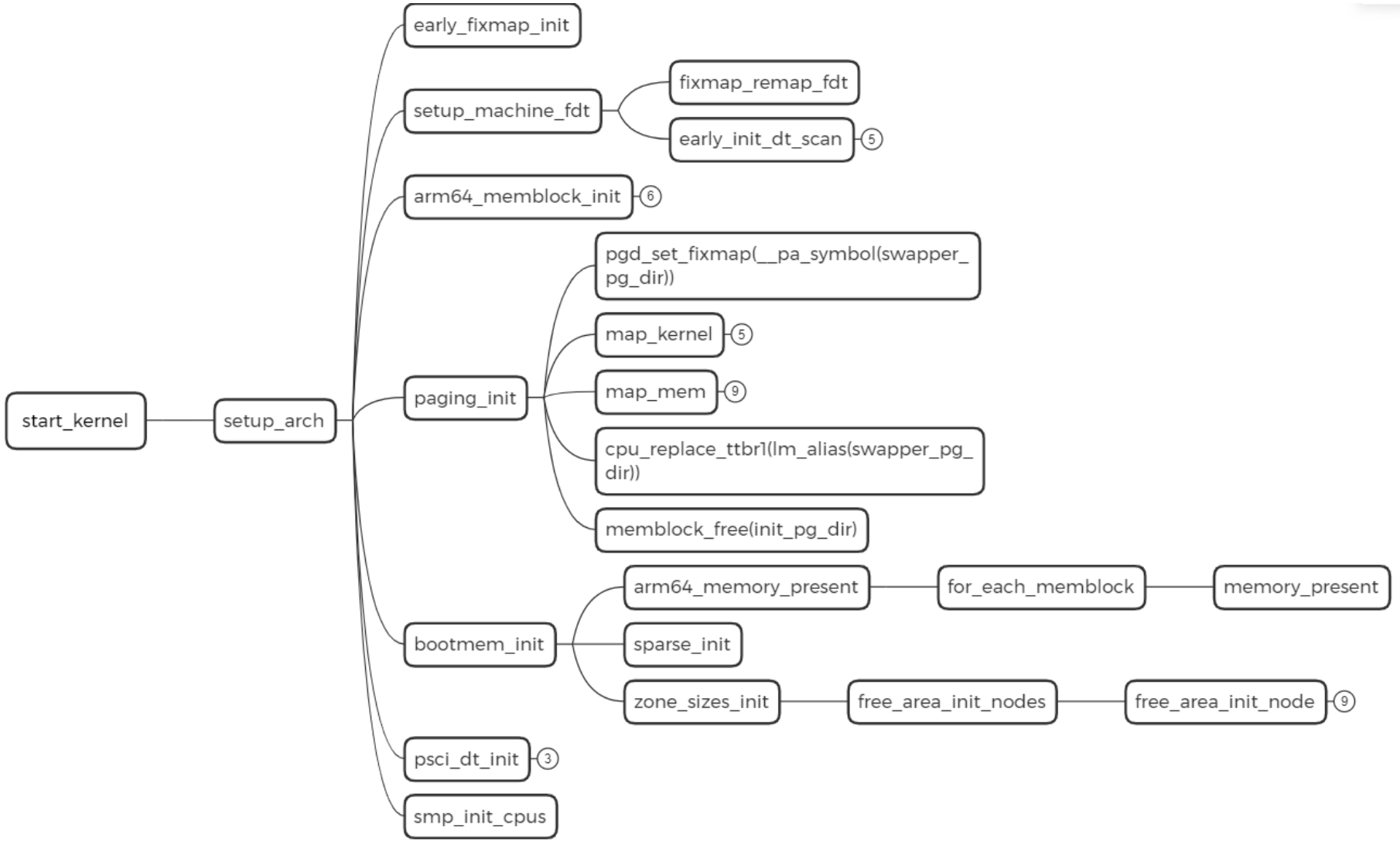
```
/*
 * Exception vectors.
 */
        .pushsection ".entry.text", "ax"

        .align  11
SYM_CODE_START(vectors)
        ......

        kernel_ventry   1, sync                 // el1 下的同步异常，例如指令执行异常、缺页中断等
        kernel_ventry   1, irq                  // el1 下的异步异常，硬件中断。 1代表异常等级
        kernel_ventry   1, fiq_invalid          // FIQ EL1h
        kernel_ventry   1, error                // Error EL1h

        kernel_ventry   0, sync                 // el0 下的同步异常，例如指令执行异常、缺页中断（就
        kernel_ventry   0, irq                  // el0 下的异步异常，硬件中断。0代表异常等级
        kernel_ventry   0, fiq_invalid          // FIQ 64-bit EL0
        kernel_ventry   0, error                // Error 64-bit EL0

        ......
#endif
SYM_CODE_END(vectors)
```



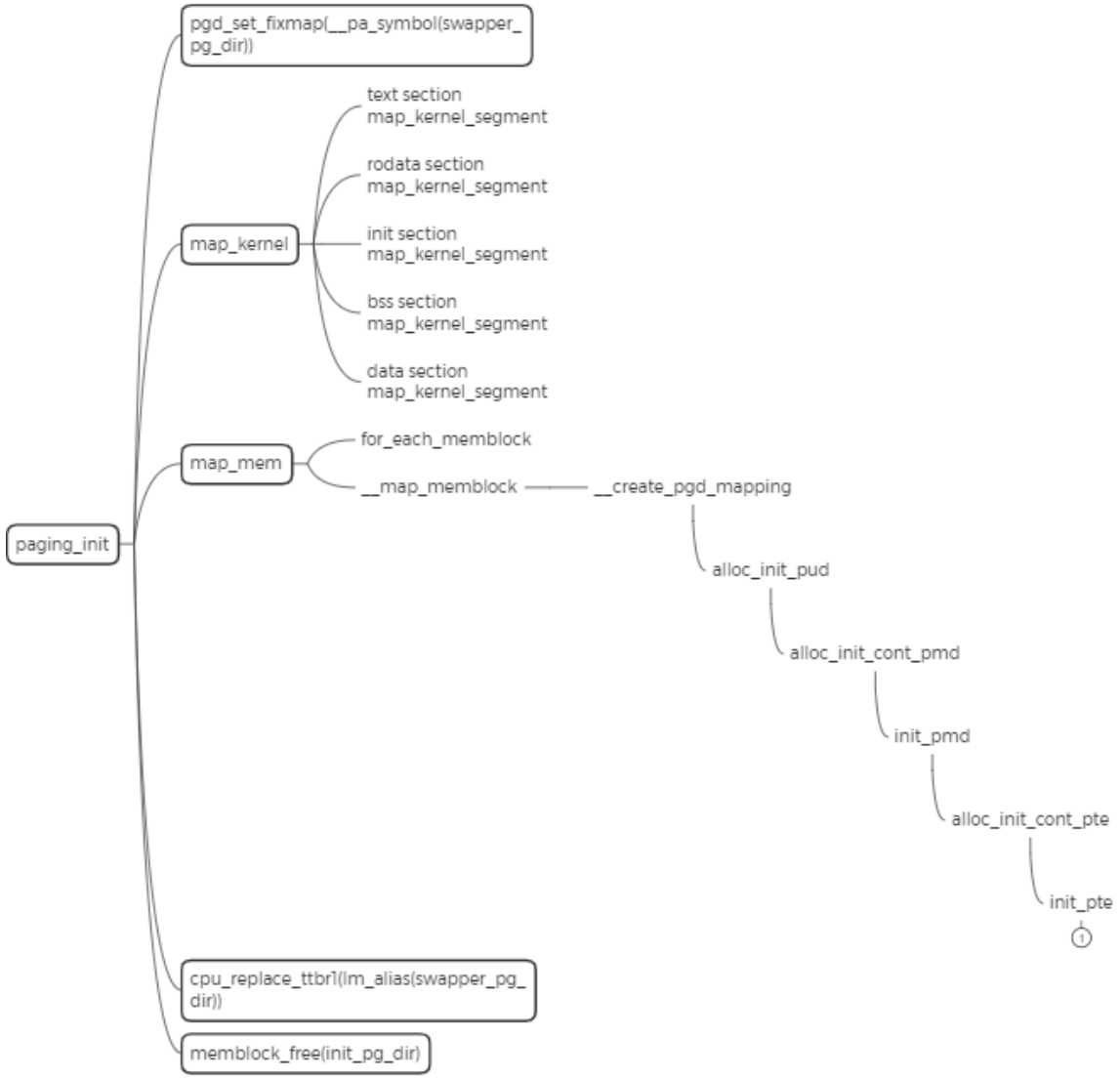
start_kernel

- Architecture Setup (setup_arch)
- Memory Subsystem init
- Schedule init
- IRQ init
- Timer init
- Console init
- Reset Init

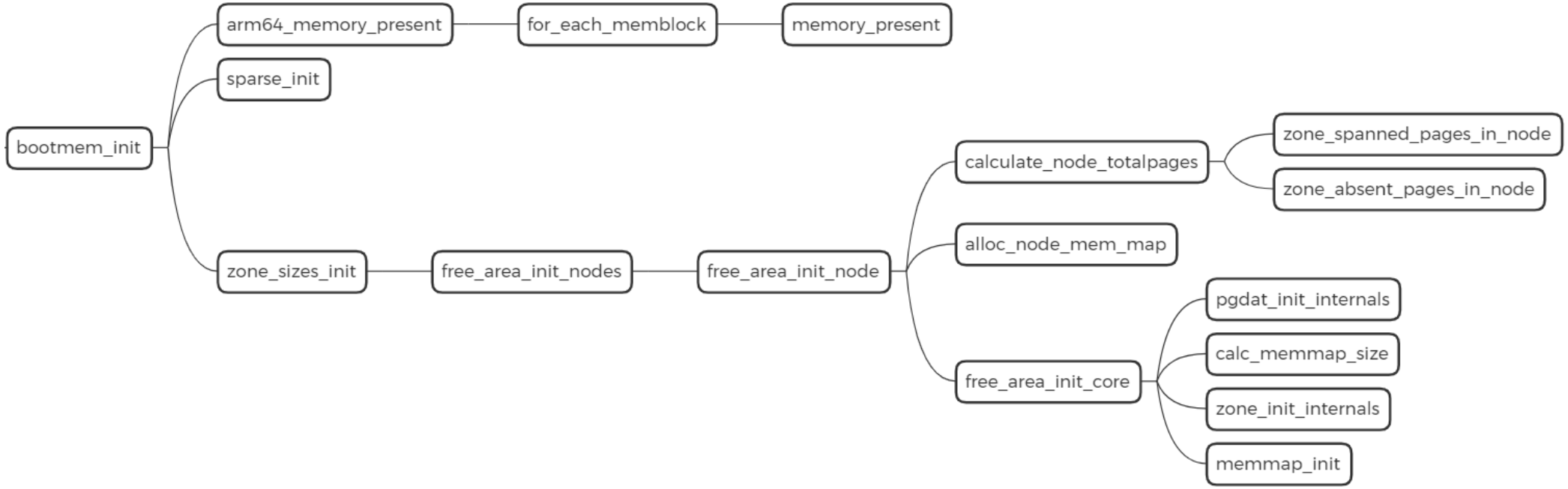
start_kernel -> setup_arch



start_kernel -> setup_arch -> paging_init

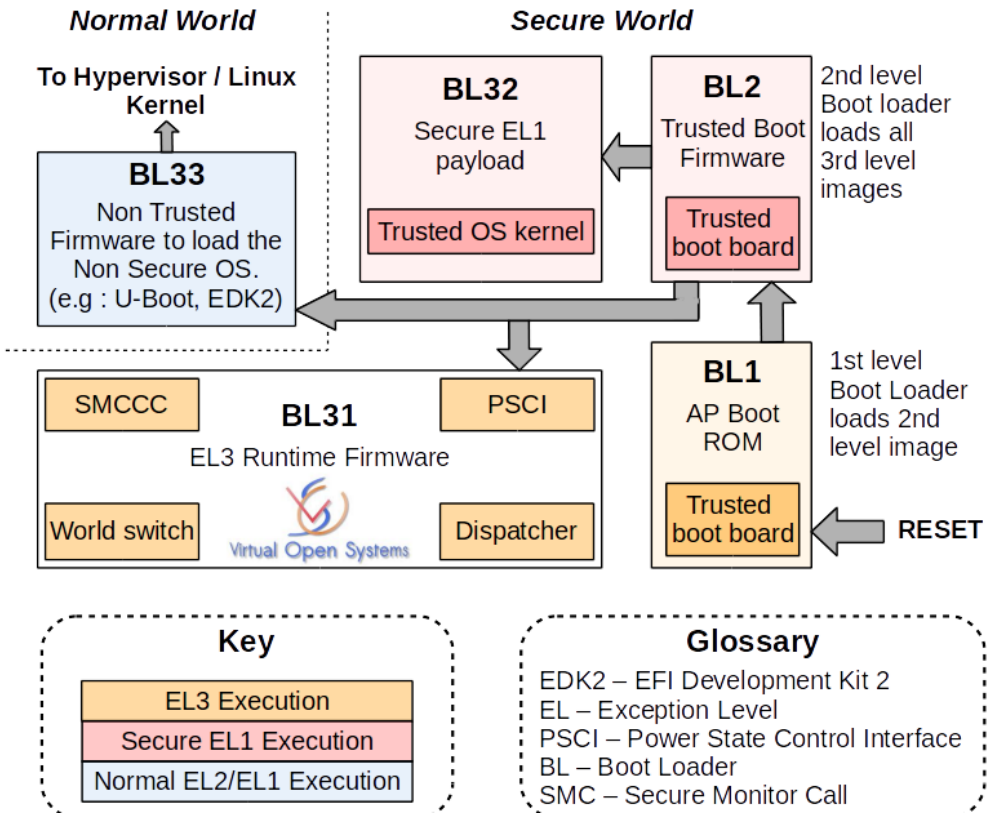


start_kernel -> setup_arch -> bootmem_init



start_kernel -> setup_arch -> psci_dt_init

- Firmware intrface implementing CPU power related operations specified by ARM PSCI spec
- Including CPU_ON/OFF/SUSPNED/MIGRATION and etc.

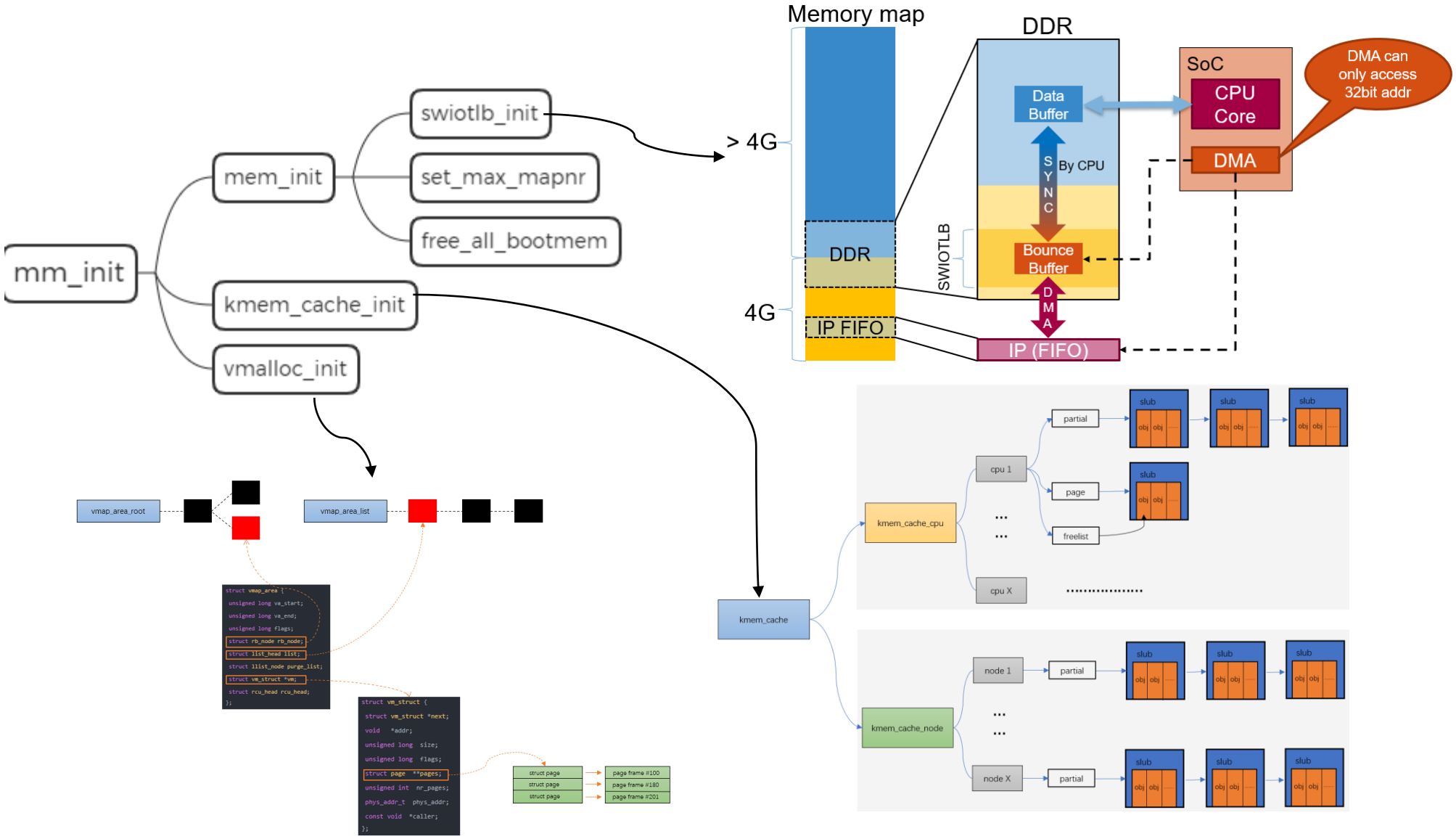



```
psci {
        compatible = "arm,psci-1.0";
        method = "smc";
};

cpus {
        #address-cells = <1>;
        #size-cells = <0>;

        idle-states {
                entry-method = "psci";

                cpu_pd_wait: cpu-pd-wait {
                        compatible = "arm,idle-state";
                        arm,psci-suspend-param = <0x0010033>;
                        local-timer-stop;
                        entry-latency-us = <1000>;
                        exit-latency-us = <700>;
                        min-residency-us = <2700>;
                };
        };

        A53_0: cpu@0 {
                device_type = "cpu";
                compatible = "arm,cortex-a53";
                reg = <0x0>;
                enable-method = "psci";
                next-level-cache = <&A53_L2>;
                operating-points-v2 = <&a53_opp_table>;
                cpu-idle-states = <&cpu_pd_wait>;
                #cooling-cells = <2>;
        };
        ...

};
```

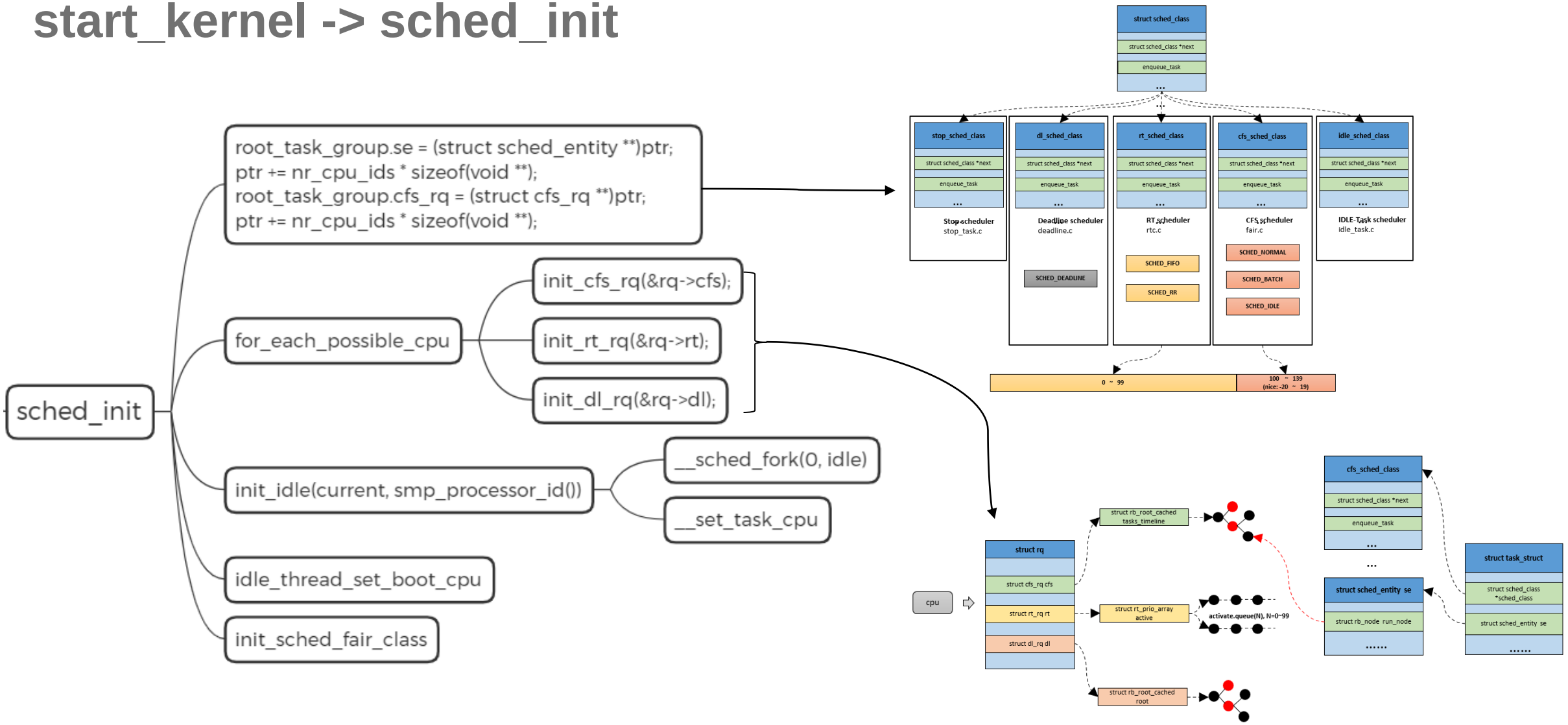
```
const struct cpu_operations cpu_psci_ops = {
        .name           = "psci",
        .cpu_init       = cpu_psci_cpu_init,
        .cpu_prepare    = cpu_psci_cpu_prepare,
        .cpu_boot       = cpu_psci_cpu_boot,
#ifdef CONFIG_HOTPLUG_CPU
        .cpu_can_disable = cpu_psci_cpu_can_disable,
        .cpu_disable    = cpu_psci_cpu_disable,
        .cpu_die        = cpu_psci_cpu_die,
        .cpu_kill       = cpu_psci_cpu_kill,
#endif
};
```

arch/arm64/kernel/psci.c

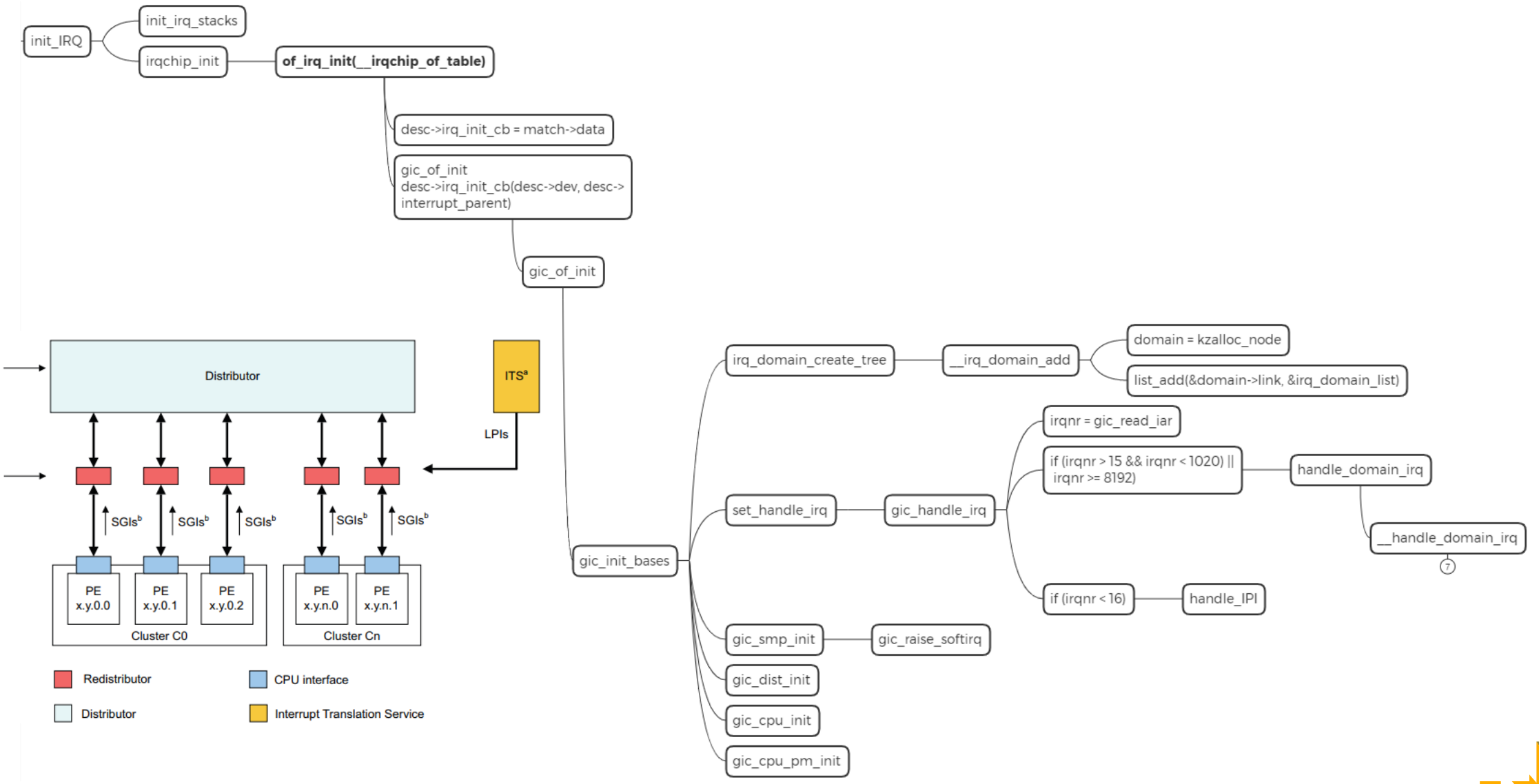
start_kernel -> mm_init



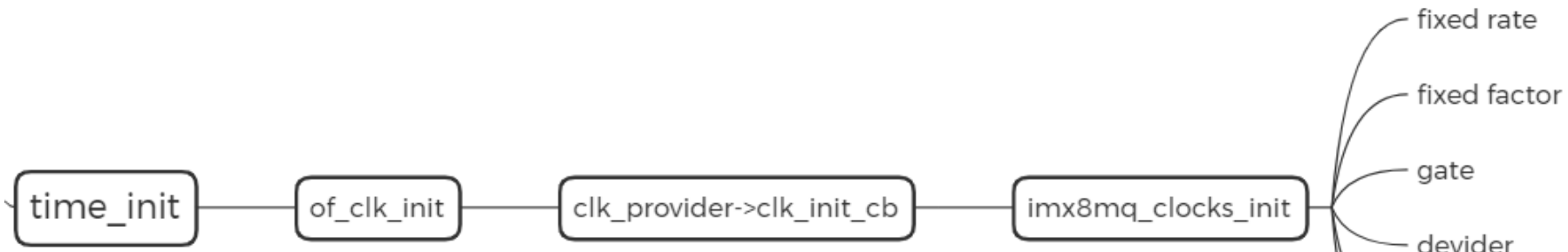
start_kernel -> sched_init



start_kernel -> init_IRQ

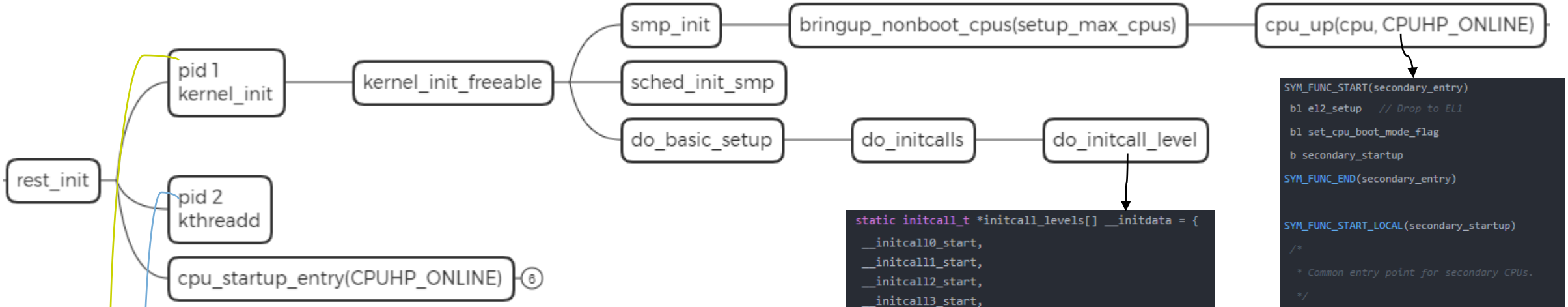


start_kernel -> time_init




```
clk: clock-controller@30380000 {
        compatible = "fsl,imx8mq-ccm";
        reg = <0x30380000 0x10000>;
        interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 86 IRQ_TYPE_LEVEL_HIGH>;
        #clock-cells = <1>;
        clocks = <&ckil>, <&osc_25m>, <&osc_27m>,
                 <&clk_ext1>, <&clk_ext2>,
                 <&clk_ext3>, <&clk_ext4>;
        clock-names = "ckil", "osc_25m", "osc_27m",
                      "clk_ext1", "clk_ext2",
                      "clk_ext3", "clk_ext4";
        assigned-clocks = <&clk IMX8MQ_CLK_A53_SRC>,
                          <&clk IMX8MQ_CLK_A53_CORE>,
                          <&clk IMX8MQ_CLK_NAND_USDHC_BUS>,
                          <&clk IMX8MQ_CLK_NOC>,
                          <&clk IMX8MQ_CLK_AUDIO_AHB>,
                          <&clk IMX8MQ_AUDIO_PLL1>,
                          <&clk IMX8MQ_AUDIO_PLL2>;
        assigned-clock-rates = <0>, <0>, <266000000>,
                               <800000000>, <0>,
                               <786432000>,
                               <722534400>;
        assigned-clock-parents = <&clk IMX8MQ_SYS1_PLL_800M>,
                                 <&clk IMX8MQ_ARM_PLL_OUT>,
                                 <0>,
                                 <&clk IMX8MQ_SYS1_PLL_800M>,
                                 <&clk IMX8MQ_SYS2_PLL_500M>;
};
```

start_kernel -> rest_init




```
# ps -A
USER         PID  PPID    VSZ    RSS WCHAN            ADDR S NAME
root           1     0  18748   3492 SyS_epoll_wait      0 S init
root           2     0      0      0 kthreadd           0 S [kthreadd]
root           3     2      0      0 worker_thread      0 I [kworker/0:0]
root           4     2      0      0 worker_thread      0 I [kworker/0:0H]
root           5     2      0      0 worker_thread      0 I [kworker/u12:0]
root           6     2      0      0 rescuer_thread     0 I [mm_percpu_wq]
root           7     2      0      0 smpboot_thread_fn  0 S [ksoftirqd/0]
root           8     2      0      0 rcu_gp_kthread     0 I [rcu_preempt]
root           9     2      0      0 rcu_gp_kthread     0 I [rcu_sched]
root          10     2      0      0 rcu_gp_kthread     0 I [rcu_bh]
root          11     2      0      0 smpboot_thread_fn  0 S [migration/0]
root          12     2      0      0 smpboot_thread_fn  0 S [cpuhp/0]
root          13     2      0      0 smpboot_thread_fn  0 S [cpuhp/1]
root          14     2      0      0 smpboot_thread_fn  0 S [migration/1]
root          15     2      0      0 smpboot_thread_fn  0 S [ksoftirqd/1]
root          16     2      0      0 worker_thread      0 I [kworker/1:0]
root          17     2      0      0 worker_thread      0 I [kworker/1:0H]
```

```
static initcall_t *initcall_levels[] __initdata = {
 __initcall0_start,
 __initcall1_start,
 __initcall2_start,
 __initcall3_start,
 __initcall4_start,
 __initcall5_start,
 __initcall6_start,
 __initcall7_start,
 __initcall_end,
};

/* Keep these in sync with initcalls in include/linux
static char *initcall_level_names[] __initdata = {
 "early",
 "core",
 "postcore",
 "arch",
 "subsys",
 "fs",
 "device",
 "late",
};
```

```
SYM_FUNC_START(secondary_entry)
 bl el2_setup    // Drop to EL1
 bl set_cpu_boot_mode_flag
 b secondary_startup
SYM_FUNC_END(secondary_entry)

SYM_FUNC_START_LOCAL(secondary_startup)
 /*
  * Common entry point for secondary CPUs.
  */
 bl __cpu_secondary_check52bitva
 bl __cpu_setup    // initialise processor
 adrp x1, swapper_pg_dir
 bl __enable_mmu
 ldr x8, =__secondary_switched
 br x8
SYM_FUNC_END(secondary_startup)
```

Start_kernel -> rest_init

Enter the user world!