Physical Memory Models: the ways Linux kernel addresses physical memory (physical page frame)

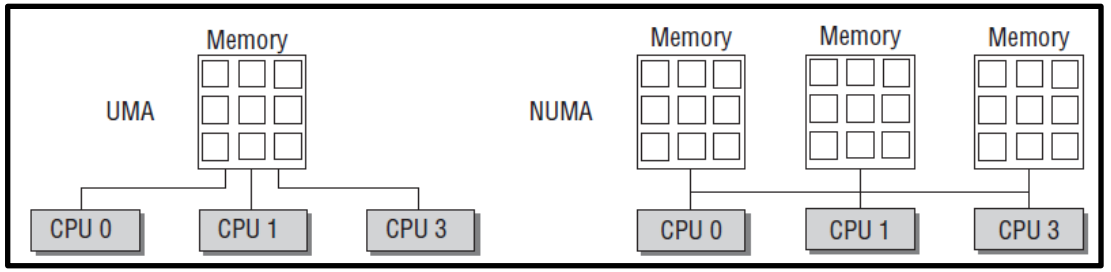
Adrian Huang | June, 2022

*** Kernel 5.11 (x86_64)**

Agenda

- Four physical memory models
  - ✓Purpose: page descriptor <-> PFN (Page Frame Number)
- Sparse memory model
- Sparse Memory Virtual Memmap: subsection
- page->flags

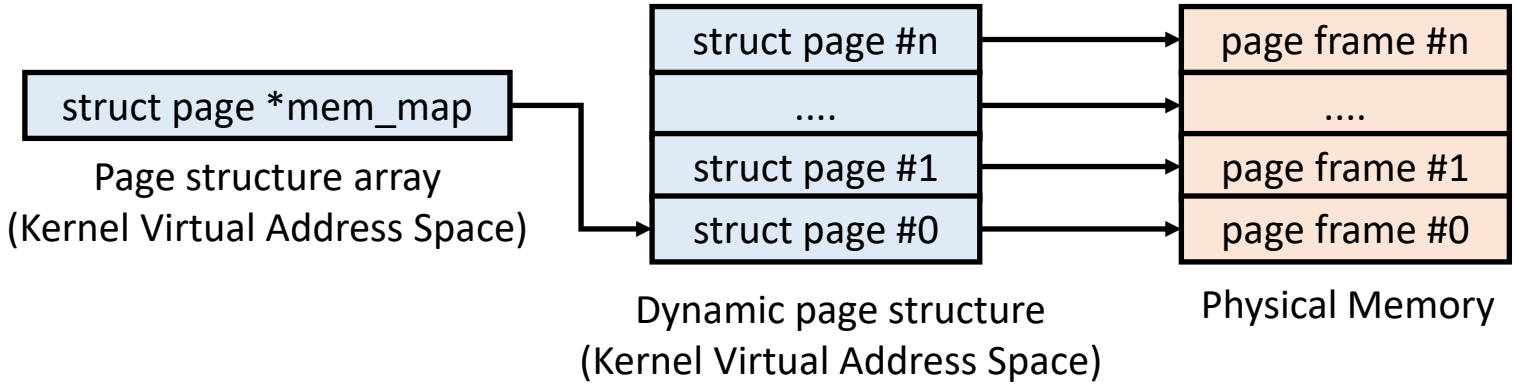
Four Physical Memory Models



- Flat Memory Model (CONFIG_FLATMEM)
  - ✓UMA (Uniform Memory Access) with mostly continuous physical memory
- Discontinuous Memory Model (CONFIG_DISCONTIGMEM)
  - ✓NUMA (Non-Uniform Memory Access) with mostly continuous physical memory
  - ✓Removed since v5.14 because sparse memory model can cover this scope
    - https://lore.kernel.org/linux-mm/20210602105348.13387-1-rppt@kernel.org/
- Sparse Memory (CONFIG_SPARSEMEM)
  - ✓NUMA with discontinuous physical memory
- Sparse Memory Virtual Memmap (CONFIG_SPARSEMEM_VMEMMAP)
  - ✓NUMA with discontinuous physical memory: A quick way to get page struct and pfn

Memory Model – Flat Memory

```
#define __pfn_to_page(pfn)      (mem_map + ((pfn) - ARCH_PFN_OFFSET))
#define __page_to_pfn(page)     ((unsigned long)((page) - mem_map) + \
                                 ARCH_PFN_OFFSET)
```



Note

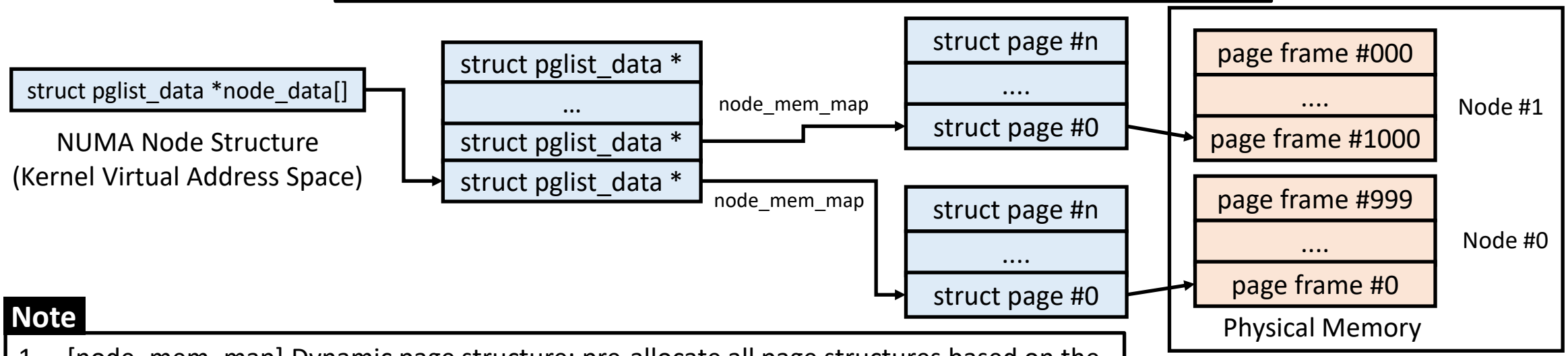
1. [mem_map] Dynamic page structure: pre-allocate all page structures based on the number of page frames
   - ✓ Allocate/Init page structures based on node's memory info (struct pglist_data)
     - Refer from: pglist_data.node_start_pfn & pglist_data.node_spanned_pages
2. Scenario: Continuous page frames (no memory holes) in UMA
3. Drawback
   - ✓ Waste node_mem_map space if memory holes
   - ✓ does not support memory hotplug
4. Check kernel function alloc_node_mem_map() in mm/page_alloc.c

Memory Model – Flat Memory

```
#ifdef CONFIG_FLAT_NODE_MEM_MAP
static void __ref alloc_node_mem_map(struct pglist_data *pgdat)
{
+--  7 lines: unsigned long __maybe_unused start = 0;-------------------------
        start = pgdat->node_start_pfn & ~(MAX_ORDER_NR_PAGES - 1);
        offset = pgdat->node_start_pfn - start;
        /* ia64 gets its own node_mem_map, before this, without bootmem */
        if (!pgdat->node_mem_map) {
                unsigned long size, end;
                struct page *map;

                /*
                 * The zone's endpoints aren't required to be MAX_ORDER
                 * aligned but the node_mem_map endpoints must be in order
                 * for the buddy allocator to function correctly.
                 */
                end = pgdat_end_pfn(pgdat);
                end = ALIGN(end, MAX_ORDER_NR_PAGES);
                size =  (end - start) * sizeof(struct page);
                map = memblock_alloc_node(size, SMP_CACHE_BYTES,
                                          pgdat->node_id);
                if (!map)
                        panic("Failed to allocate %ld bytes for node %d memory m
ap\n",
                                        size, pgdat->node_id);
                pgdat->node_mem_map = map + offset;
        }
+--  3 lines: pr_debug("%s: node %d, pgdat %08lx, node_mem_map %08lx\n",--------
#ifndef CONFIG_NEED_MULTIPLE_NODES
        /*
         * With no DISCONTIG, the global mem_map is just set as node 0's
         */
        if (pgdat == NODE_DATA(0)) {
                mem_map = NODE_DATA(0)->node_mem_map;
                if (page_to_pfn(mem_map) != pgdat->node_start_pfn)
                        mem_map -= offset;
        }
#endif
}
#else
static void __ref alloc_node_mem_map(struct pglist_data *pgdat) { }
#endif /* CONFIG_FLAT_NODE_MEM_MAP */

mm/page_alloc.c                                       6985,1-8       78%
```

Memory Model – Discontinuous Memory

```
#define __pfn_to_page(pfn)                      \
({      unsigned long __pfn = (pfn);            \
        unsigned long __nid = arch_pfn_to_nid(pfn);  \
        NODE_DATA(__nid)->node_mem_map + arch_local_page_offset(__pfn, __nid);\
})

#define __page_to_pfn(pg)                                               \
({      struct page *__pg = (pg);                                       \
        struct pglist_data *__pgdat = NODE_DATA(page_to_nid(__pg));     \
        (unsigned long)(__pg - __pgdat->node_mem_map) +                 \
         __pgdat->node_start_pfn;                                       \
})
```

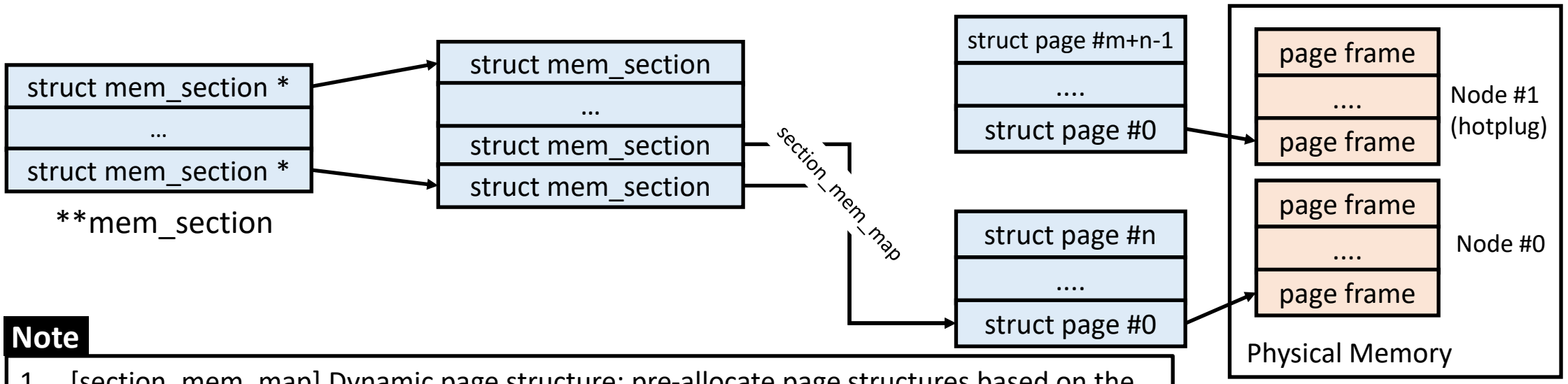


Note

1. [node_mem_map] Dynamic page structure: pre-allocate all page structures based on the number of page frames
   - ✓ Allocate/Init page structures based on node's memory info (struct pglist_data)
     - Refer from: pglist_data.node_start_pfn & pglist_data.node_spanned_pages
2. Scenario: Each node has continuous page frames (no memory holes) in NUMA
3. Drawback
   - ✓ Waste node_mem_map space if memory holes
   - ✓ does not support memory hotplug

Memory Model – Sparse Memory

```
#define __page_to_pfn(pg)                                       \
({      const struct page *__pg = (pg);                         \
        int __sec = page_to_section(__pg);                      \
        (unsigned long)(__pg - __section_mem_map_addr(__nr_to_section(__sec)));
\
})

#define __pfn_to_page(pfn)                                      \
({      unsigned long __pfn = (pfn);                            \
        struct mem_section *__sec = __pfn_to_section(__pfn);    \
        __section_mem_map_addr(__sec) + __pfn;                  \
})
```

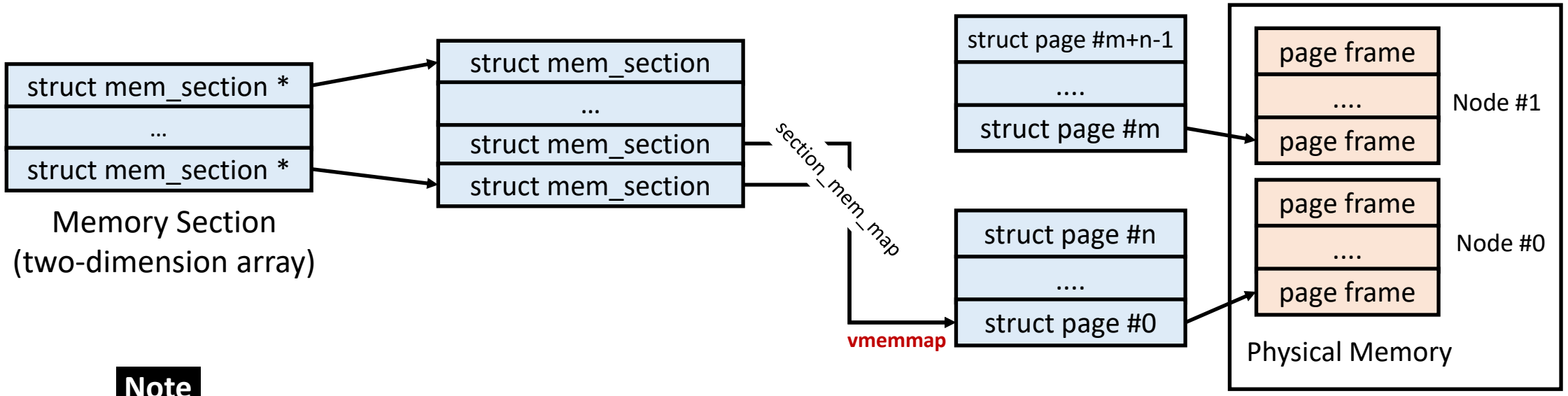


Note

1. [section_mem_map] Dynamic page structure: pre-allocate page structures based on the number of available page frames
   - ✓ Refer from: **memblock structure**
2. Support physical memory hotplug
3. Minimum unit: PAGES_PER_SECTION = 32768
   - ✓ Each memory section addresses the memory size: 32768 * 4KB (page size) = 128MB
4. [NUMA] : reduce the memory hole impact due to “struct mem_section”

Memory Model – Sparse Memory Virtual Memmap

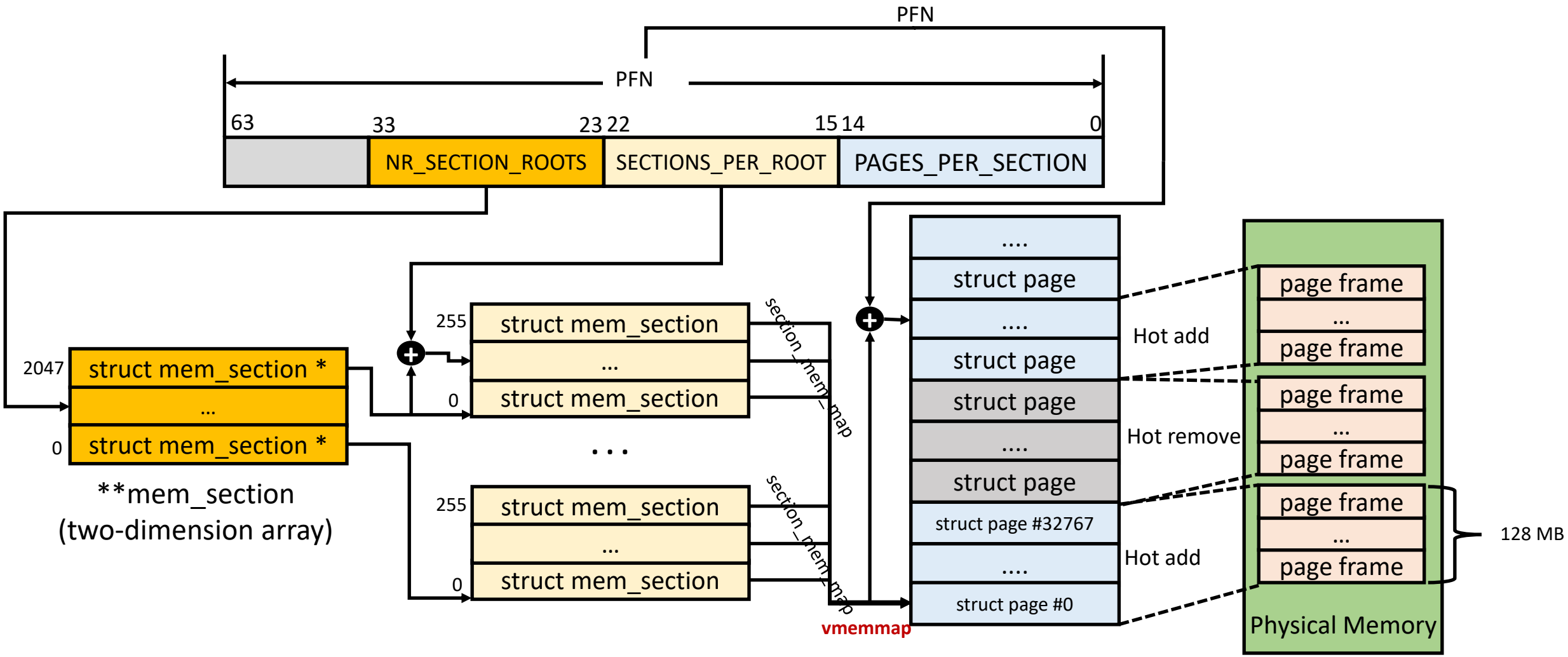
```
/* memmap is virtually contiguous.  */
#define __pfn_to_page(pfn)      (vmemmap + (pfn))
#define __page_to_pfn(page)     (unsigned long)((page) - vmemmap)
```



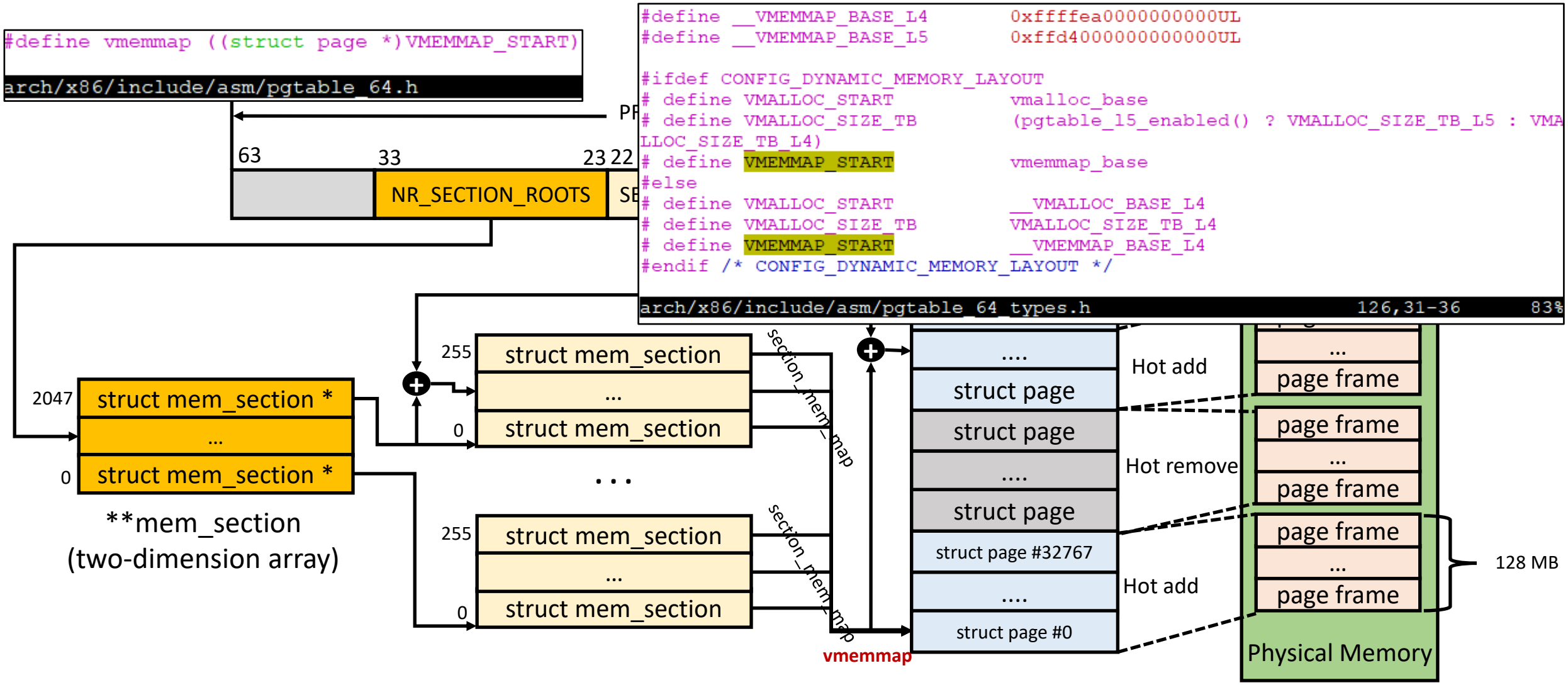
Note

1. [section_mem_map] Dynamic page structure: pre-allocate page structures based on the number of available page frames
   - ✓ Refer from: **memblock structure**
2. Support physical memory hotplug
3. Minimum unit: PAGES_PER_SECTION = 32768
   - ✓ Each memory section addresses the memory size: 32768 * 4KB (page size) = 128MB
4. [NUMA] : reduce the memory hole impact due to “struct mem_section”
5. Employ virtual memory map (vmemmap/ vmemmap_base) – A quick way to get page struct and pfn
6. **Default configuration in Linux kernel**

Memory Model – Sparse Memory Virtual Memmap: Detail



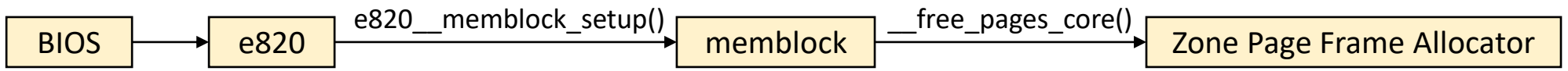
Memory Model – Sparse Memory Virtual Memmap: Detail



Sparse Memory Model

1. How to know available memory pages in a system?
2. Page Table Configuration for Direct Mapping
3. Sparse Memory Model Initialization – Detail

How to know available memory pages in a system?



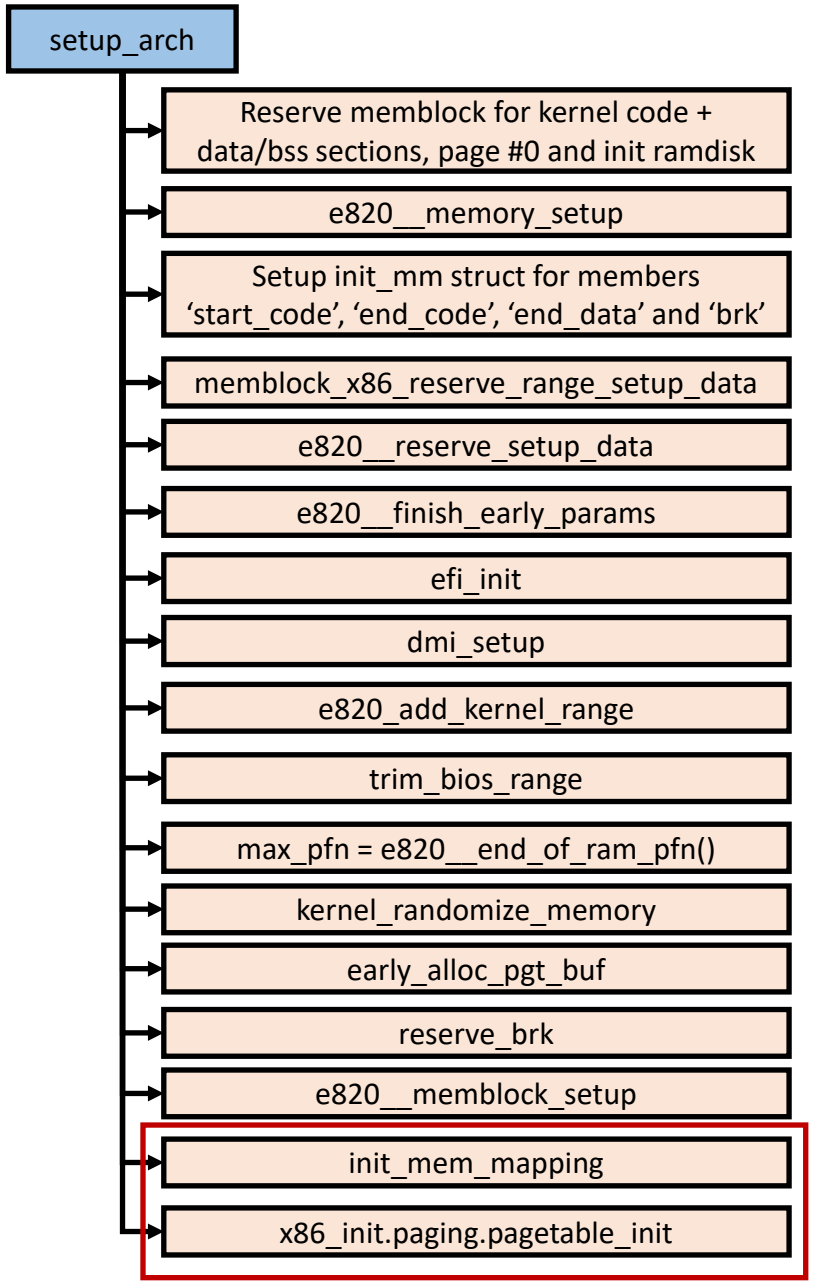
[Call Path] memblock frees available memory space to zone page frame allocator

```
start_kernel [init/main.c]
  |- mm_init
    |- mem_init
      |- memblock_free_all
        |- free_low_memory_core_early
          for_each_reserved_mem_range(i, &start, &end)
                reserve_bootmem_region(start, end);
          for_each_free_mem_range(...)
                count += __free_memory_core(start, end);
```

```
__free_memory_core [mm/memblock.c]
  |- __free_pages_memory
    |- memblock_free_pages [mm/page_alloc.c]
      |- __free_pages_core
        Clear flag 'PG_reserved' of each page struct
        Add number of pages to zone.managed_pages
```

Zone page allocator detail will be discussed in another session: physical memory management

setup_arch() -- Focus on memory portion



init_memory_mapping()

- Create 4-level page table (direct mapping) based on ‘memory’ type of memblock configuration.

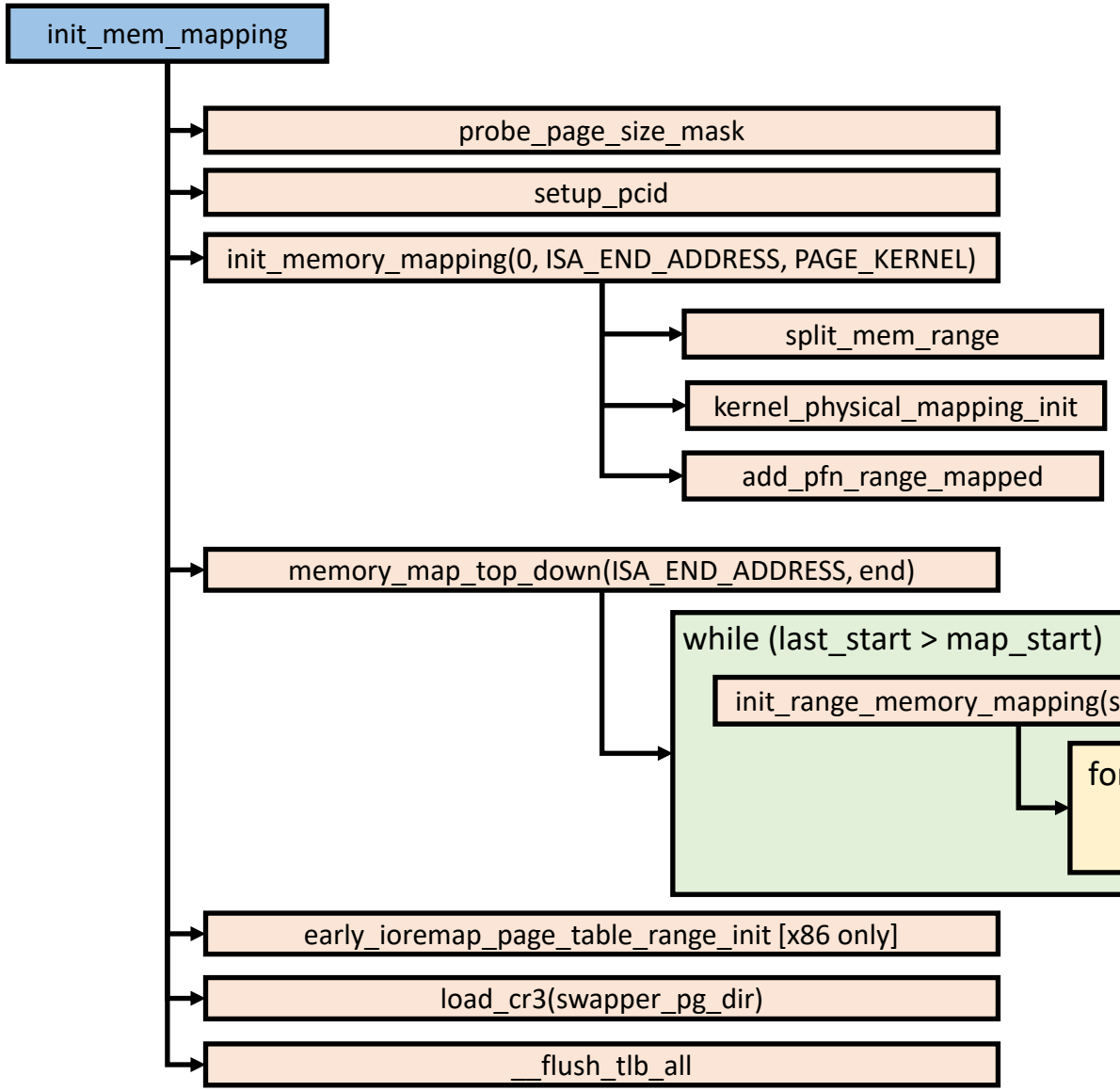
x86_init.paging.pagetable_init()

- Init sparse
- Init zone structure

x86 - setup_arch() -- init_mem_mapping() – Page Table Configuration for Direct Mapping

```
#define ISA_START_ADDRESS       0x000a0000
#define ISA_END_ADDRESS         0x00100000
```

arch/x86/include/asm/e820/types.h



init_memory_mapping() -> kernel_physical_mapping_init()

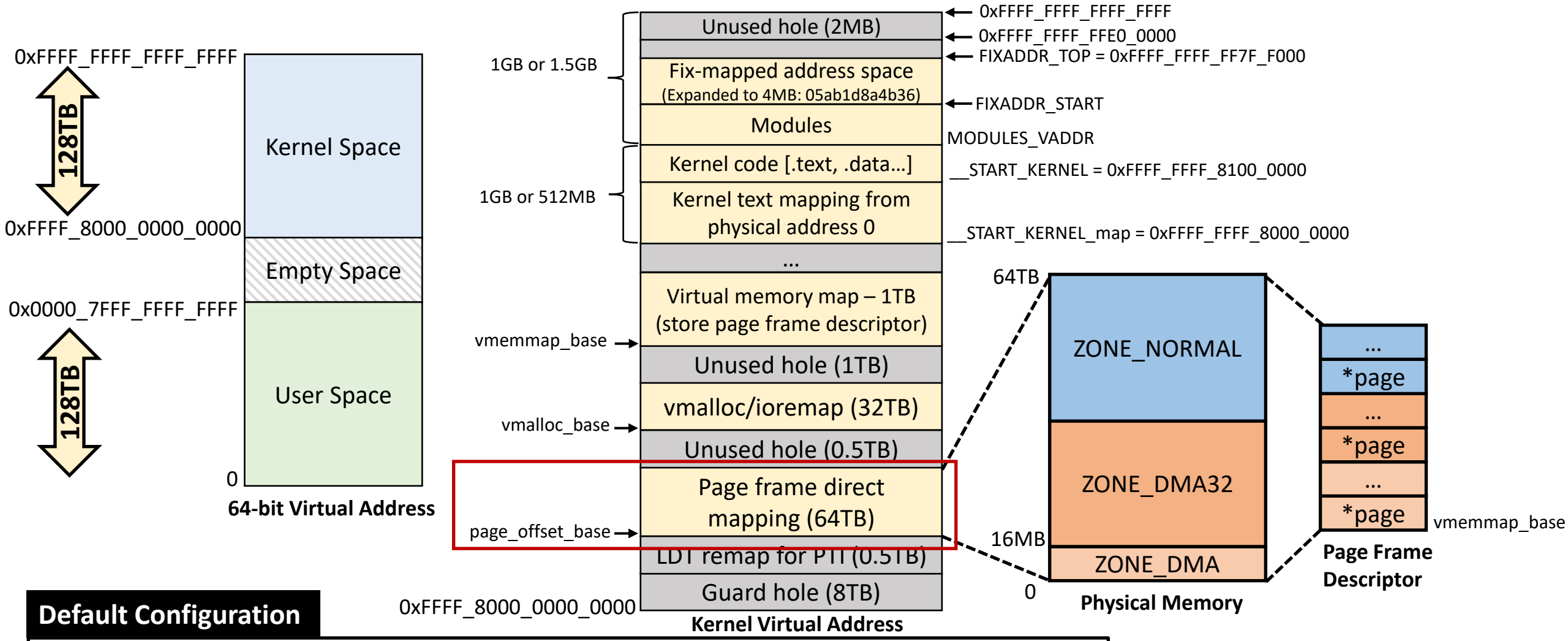
- Create 4-level page table (direct mapping) based on 'memory' type of memblock configuration.

split_mem_range()

- Split different the groups of page size based on the input memory range (start address and end address)
  - ✓ Try larger page size first
    - 1G huge page -> 2M huge page -> 4K page

Page Table Configuration for Direct Mapping

Reference: Documentation/x86/x86_64/mm.rst

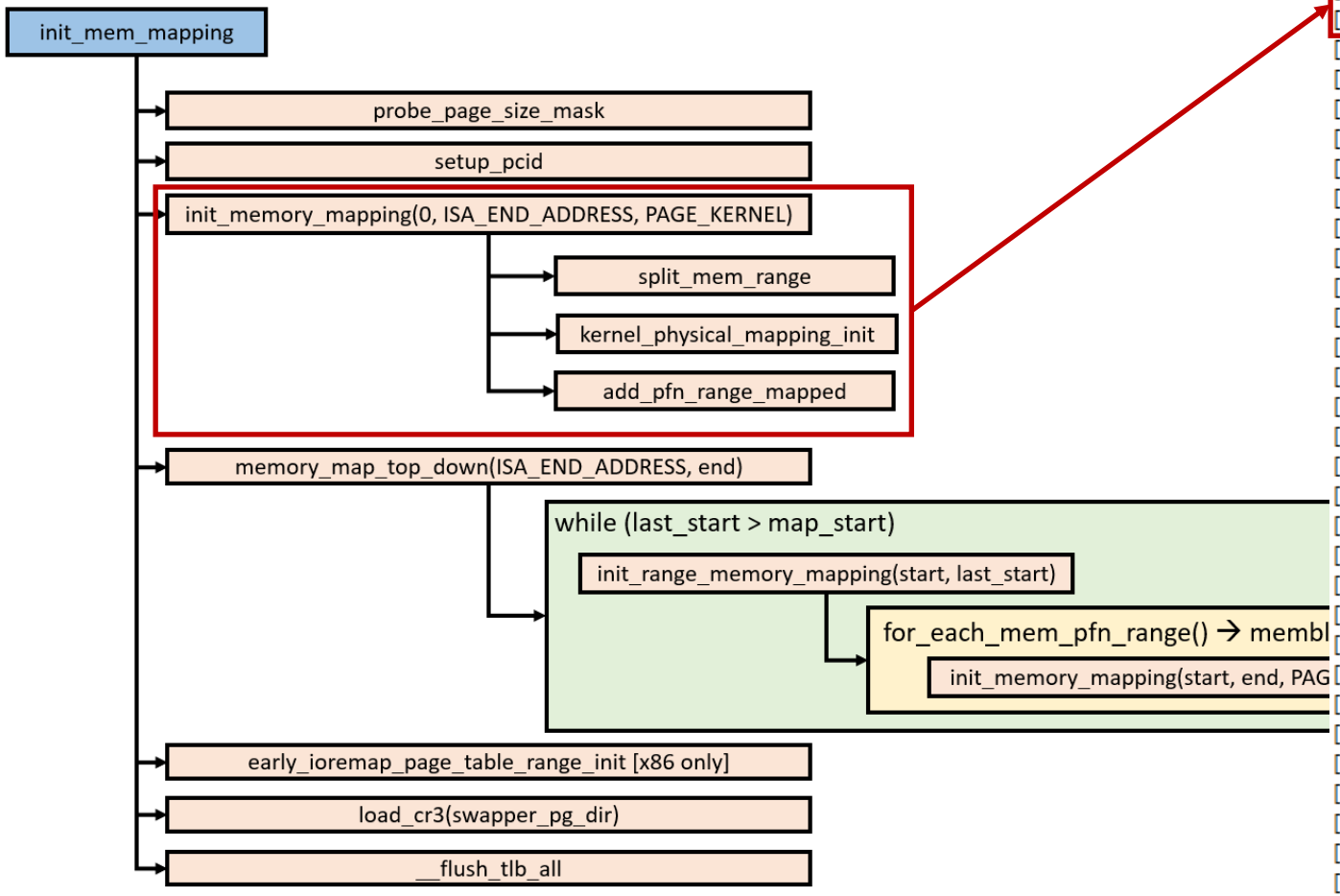


Default Configuration

page_ofset_base = 0xFFFF_8880_0000_0000
vmalloc_base = 0xFFFF_C900_0000_0000
vmemmap_base = 0xFFFF_EA00_0000_0000
* Can be dynamically configured by KASLR (Kernel Address Space Layout Randomization - "arch/x86/mm/kaslr.c")

Note: Refer from page #5 in the slide deck Decompressed vmlinux: linux kernel initialization from page table configuration perspective

init_mem_mapping() – Page Table Configuration for Direct Mapping

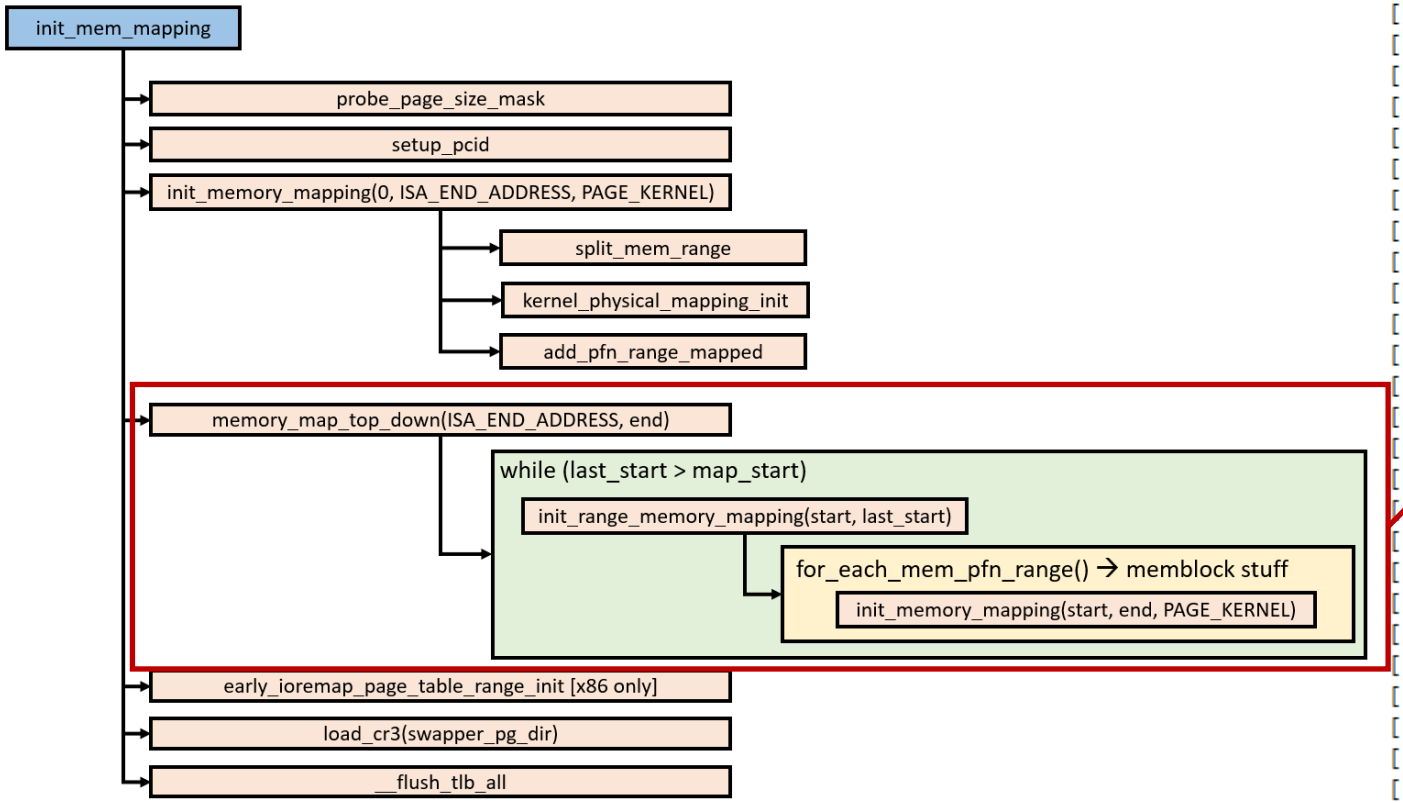



```
[    0.005977] init_memory_mapping: [mem 0x00000000-0x000fffff]
[    0.005979]  [mem 0x00000000-0x000fffff] page 4k
[    0.006032] init_memory_mapping: [mem 0x84fa00000-0x84fbfffff]
[    0.006033]  [mem 0x84fa00000-0x84fbfffff] page 2M
[    0.006102] init_memory_mapping: [mem 0x840000000-0x84f9fffff]
[    0.006102]  [mem 0x840000000-0x84f9fffff] page 2M
[    0.006172] init_memory_mapping: [mem 0x820000000-0x83fffffff]
[    0.006172]  [mem 0x820000000-0x83fffffff] page 1G
[    0.006202] init_memory_mapping: [mem 0x00100000-0x2fffffff]
[    0.006203]  [mem 0x00100000-0x001fffff] page 4k
[    0.006204]  [mem 0x00200000-0x2fffffff] page 2M
[    0.006278] init_memory_mapping: [mem 0x30042000-0x31daffff]
[    0.006278]  [mem 0x30042000-0x301fffff] page 4k
[    0.006279]  [mem 0x30200000-0x31bfffff] page 2M
[    0.006280]  [mem 0x31c00000-0x31daffff] page 4k
[    0.006386] init_memory_mapping: [mem 0x32000000-0xa3b29fff]
[    0.006387]  [mem 0x32000000-0x3fffffff] page 2M
[    0.006388]  [mem 0x40000000-0x7fffffff] page 1G
[    0.006389]  [mem 0x80000000-0xa39fffff] page 2M
[    0.006389]  [mem 0xa3a00000-0xa3b29fff] page 4k
[    0.006573] init_memory_mapping: [mem 0xa7855000-0xabf91fff]
[    0.006574]  [mem 0xa7855000-0xa79fffff] page 4k
[    0.006575]  [mem 0xa7a00000-0xabdfffff] page 2M
[    0.006575]  [mem 0xabe00000-0xabf91fff] page 4k
[    0.006724] init_memory_mapping: [mem 0xabfc4000-0xabffffff]
[    0.006725]  [mem 0xabfc4000-0xabffffff] page 4k
[    0.006773] init_memory_mapping: [mem 0x100000000-0x44effffff]
[    0.006773]  [mem 0x100000000-0x43fffffff] page 1G
[    0.006774]  [mem 0x440000000-0x44effffff] page 2M
[    0.006852] init_memory_mapping: [mem 0x450000000-0x81fffffff]
[    0.006853]  [mem 0x450000000-0x47fffffff] page 2M
[    0.006853]  [mem 0x480000000-0x81fffffff] page 1G
```

Note

- 2-socket server with 32GB memory

init_mem_mapping() – Page Table Configuration for Direct Mapping




```
[    0.005977] init_memory_mapping: [mem 0x00000000-0x000fffff]
[    0.005979]  [mem 0x00000000-0x000fffff] page 4k
[    0.006032] init_memory_mapping: [mem 0x84fa00000-0x84fbfffff]
[    0.006033]  [mem 0x84fa00000-0x84fbfffff] page 2M
[    0.006102] init_memory_mapping: [mem 0x840000000-0x84f9fffff]
[    0.006102]  [mem 0x840000000-0x84f9fffff] page 2M
[    0.006172] init_memory_mapping: [mem 0x820000000-0x83fffffff]
[    0.006172]  [mem 0x820000000-0x83fffffff] page 1G
[    0.006202] init_memory_mapping: [mem 0x00100000-0x2fffffff]
[    0.006203]  [mem 0x00100000-0x001fffff] page 4k
[    0.006204]  [mem 0x00200000-0x2fffffff] page 2M
[    0.006278] init_memory_mapping: [mem 0x30042000-0x31daffff]
[    0.006278]  [mem 0x30042000-0x301fffff] page 4k
[    0.006279]  [mem 0x30200000-0x31bfffff] page 2M
[    0.006280]  [mem 0x31c00000-0x31daffff] page 4k
[    0.006386] init_memory_mapping: [mem 0x32000000-0xa3b29fff]
[    0.006387]  [mem 0x32000000-0x3fffffff] page 2M
[    0.006388]  [mem 0x40000000-0x7fffffff] page 1G
[    0.006389]  [mem 0x80000000-0xa39fffff] page 2M
[    0.006389]  [mem 0xa3a00000-0xa3b29fff] page 4k
[    0.006573] init_memory_mapping: [mem 0xa7855000-0xabf91fff]
[    0.006574]  [mem 0xa7855000-0xa79fffff] page 4k
[    0.006575]  [mem 0xa7a00000-0xabdfffff] page 2M
[    0.006575]  [mem 0xabe00000-0xabf91fff] page 4k
[    0.006724] init_memory_mapping: [mem 0xabfc4000-0xabffffff]
[    0.006725]  [mem 0xabfc4000-0xabffffff] page 4k
[    0.006773] init_memory_mapping: [mem 0x100000000-0x44effffff]
[    0.006773]  [mem 0x100000000-0x43fffffff] page 1G
[    0.006774]  [mem 0x440000000-0x44effffff] page 2M
[    0.006852] init_memory_mapping: [mem 0x450000000-0x81fffffff]
[    0.006853]  [mem 0x450000000-0x47fffffff] page 2M
[    0.006853]  [mem 0x480000000-0x81fffffff] page 1G
```

Note

- 2-socket server with 32GB memory

setup_arch() -- init_mem_mapping() – Page Table Configuration for Direct Mapping

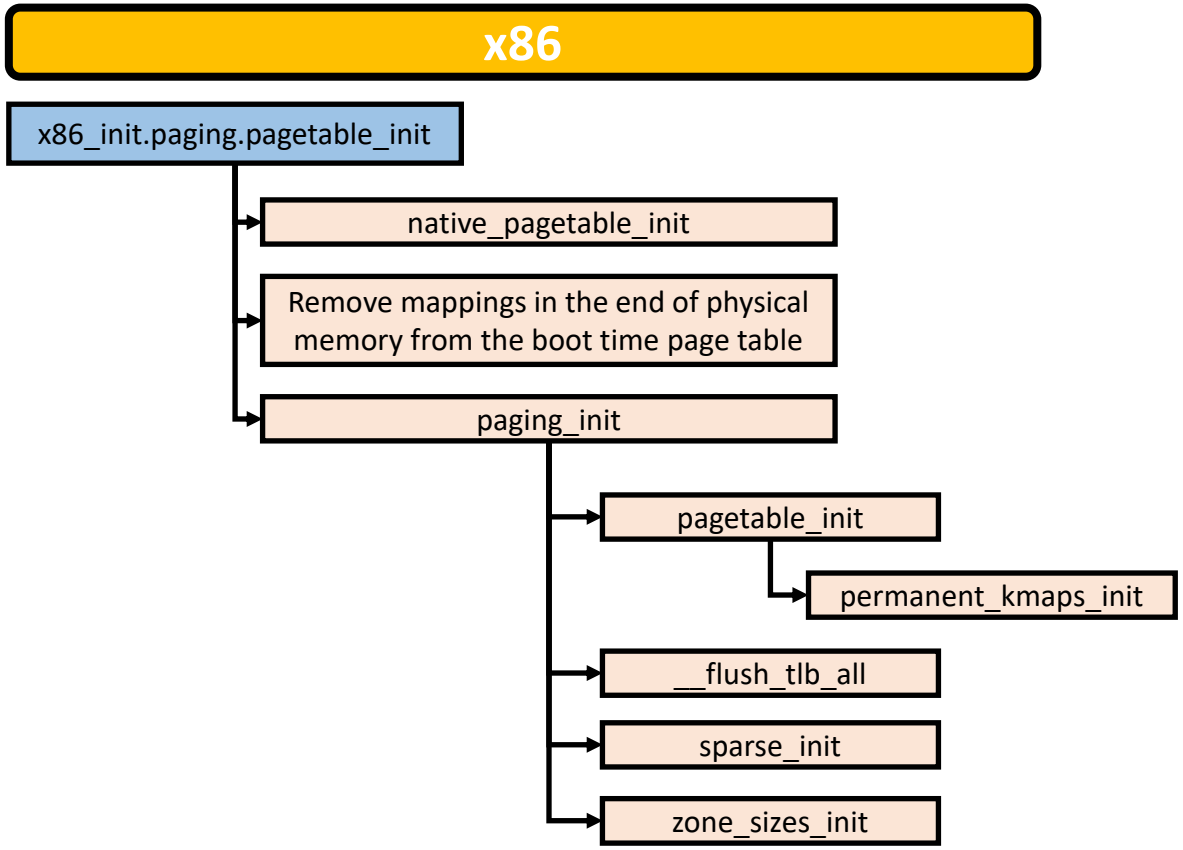
```
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009ffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000000a0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x000000002fffffff] usable
[    0.000000] BIOS-e820: [mem 0x0000000030000000-0x0000000030041fff] ACPI NVS
[    0.000000] BIOS-e820: [mem 0x0000000030042000-0x000000031dafffff] usable
[    0.000000] BIOS-e820: [mem 0x000000031db0000-0x000000031fffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000032000000-0x00000000a3b29fff] usable
[    0.000000] BIOS-e820: [mem 0x00000000a3b2a000-0x00000000a59cefff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000a59cf000-0x00000000a5b87fff] ACPI data
[    0.000000] BIOS-e820: [mem 0x00000000a5b88000-0x00000000a619bfff] ACPI NVS
[    0.000000] BIOS-e820: [mem 0x00000000a619c000-0x00000000a7854fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000a7855000-0x00000000abf91fff] usable
[    0.000000] BIOS-e820: [mem 0x00000000abf92000-0x00000000abfc3fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000abfc4000-0x00000000abffffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000ac000000-0x00000000afffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000b4000000-0x00000000b5ffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000be000000-0x00000000bfffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000c8000000-0x00000000c9ffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000f4000000-0x00000000f5ffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fe000000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000044effffff] usable
[    0.000000] BIOS-e820: [mem 0x000000044f000000-0x000000044fffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000450000000-0x000000084fbfffff] usable
[    0.000000] BIOS-e820: [mem 0x000000084fc00000-0x000000084fffffff] reserved
```

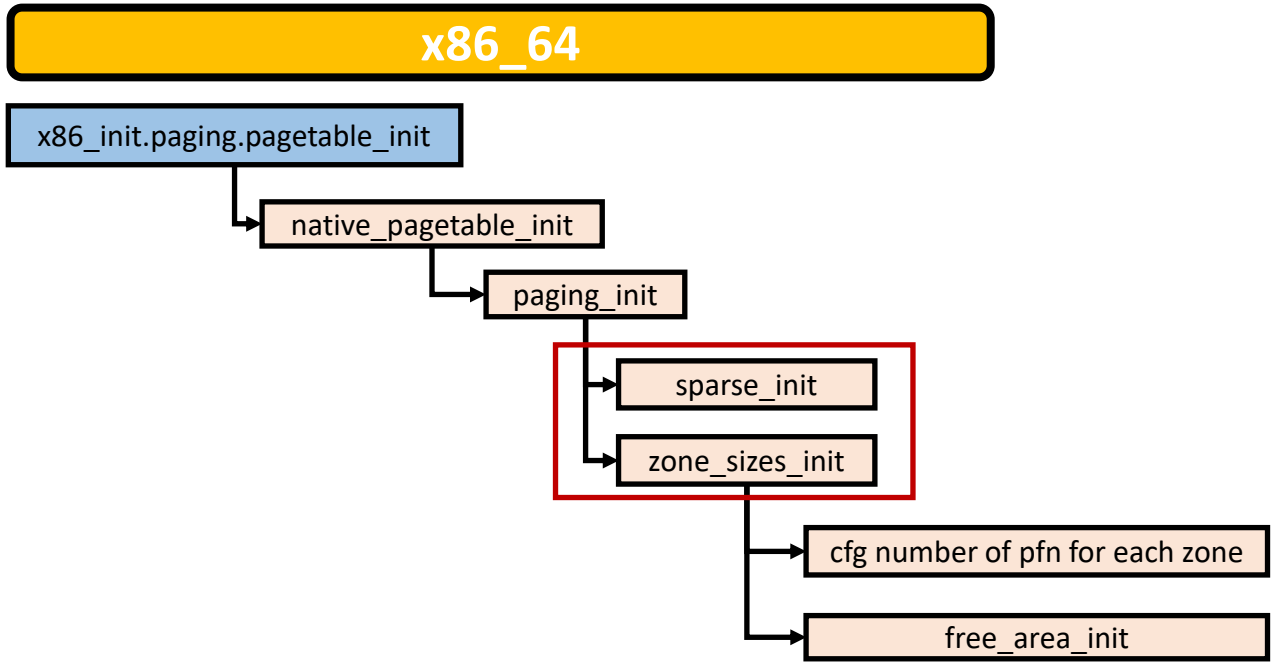
init_memory_mapping() -> kernel_physical_mapping_init()

- Create 4-level page table (direct mapping) based on ‘memory’ type of the memblock configuration.

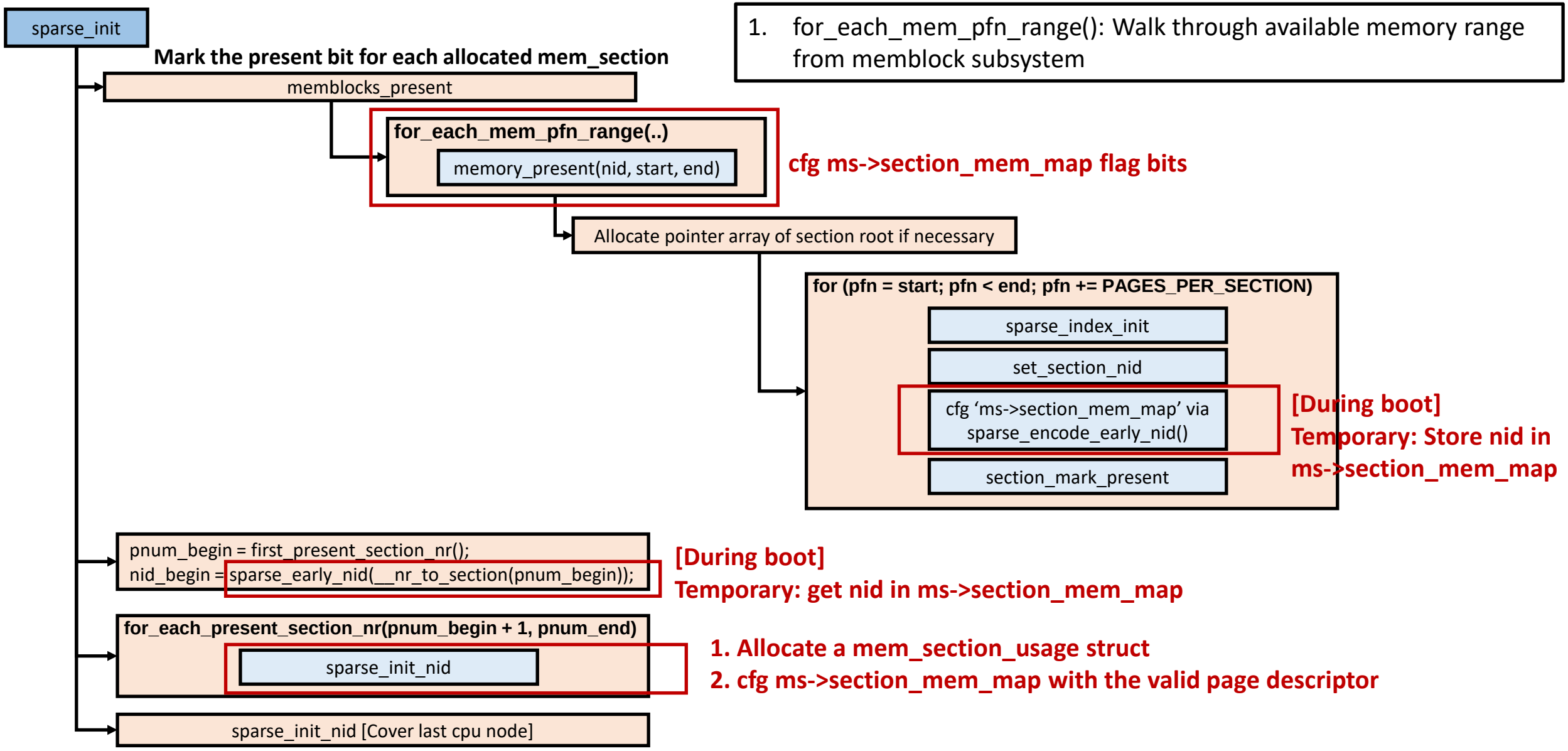
```
[    0.007312] MEMBLOCK configuration:
[    0.007312]  memory size = 0x00000007f6bb0000 reserved size = 0x000000001469a522
[    0.007313]  memory.cnt  = 0x8
[    0.007314]  memory[0x0]     [0x0000000000001000-0x000000000009ffff], 0x000000000009f000 bytes on node 0 flags: 0x0
[    0.007316]  memory[0x1]     [0x0000000000100000-0x000000002fffffff], 0x000000002ff00000 bytes on node 0 flags: 0x0
[    0.007317]  memory[0x2]     [0x0000000030042000-0x000000031dafffff], 0x000000001d6e000 bytes on node 0 flags: 0x0
[    0.007318]  memory[0x3]     [0x0000000032000000-0x00000000a3b29fff], 0x0000000071b2a000 bytes on node 0 flags: 0x0
[    0.007320]  memory[0x4]     [0x00000000a7855000-0x00000000abf91fff], 0x000000000473d000 bytes on node 0 flags: 0x0
[    0.007321]  memory[0x5]     [0x00000000abfc4000-0x00000000abffffff], 0x000000000003c000 bytes on node 0 flags: 0x0
[    0.007322]  memory[0x6]     [0x0000000100000000-0x000000044effffff], 0x000000034f000000 bytes on node 0 flags: 0x0
[    0.007323]  memory[0x7]     [0x0000000450000000-0x000000084fbfffff], 0x00000003ffc00000 bytes on node 1 flags: 0x0
```

x86 - setup_arch() -- x86_init.paging.pagetable_init()



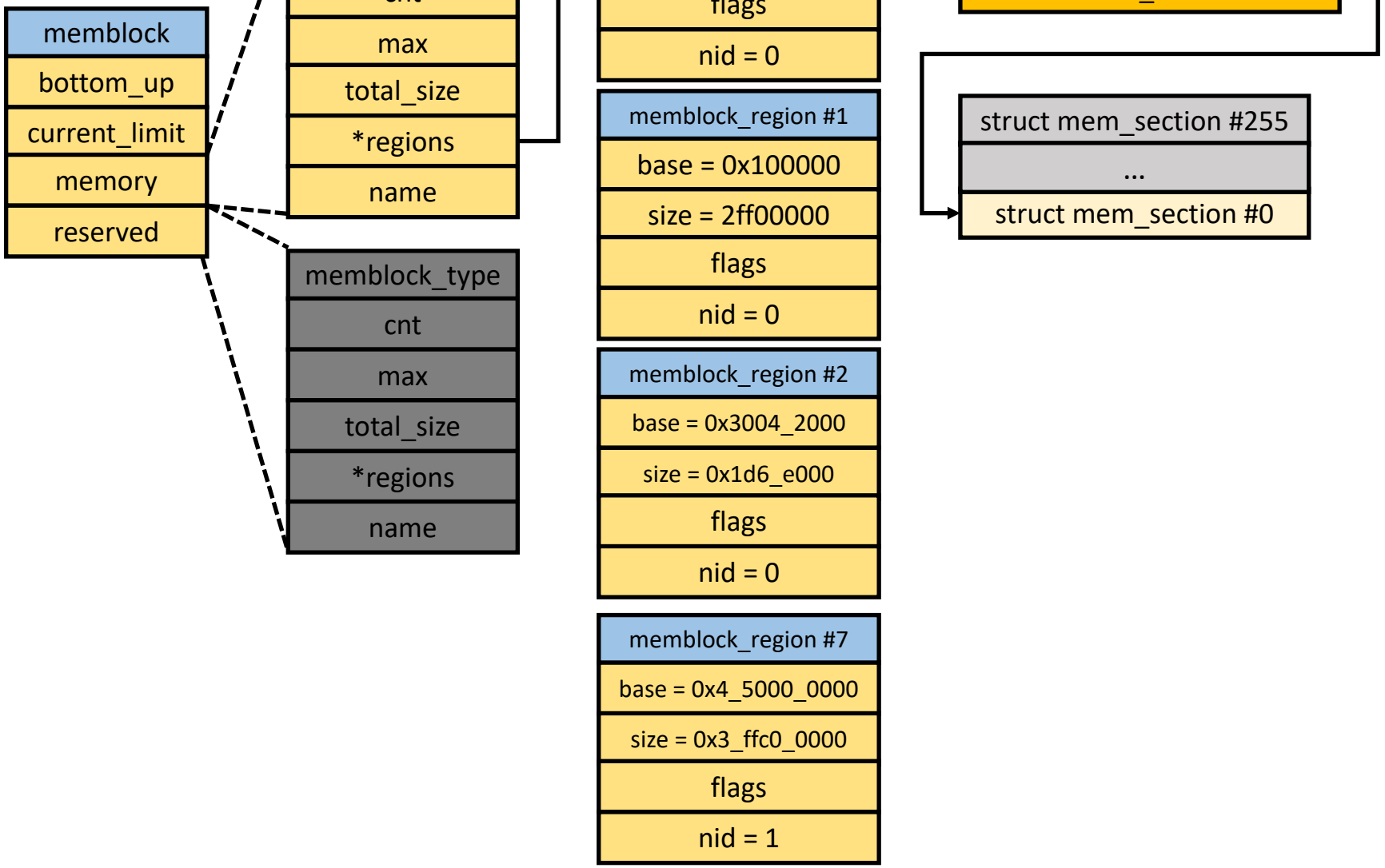


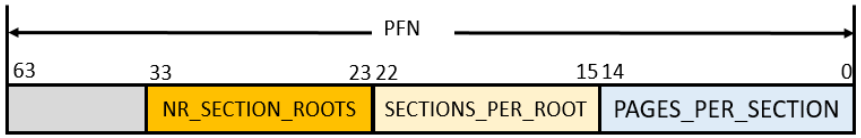
Sparse Memory Model Initialization: sparse_init()

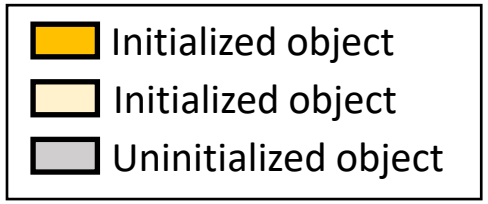


1. for_each_mem_pfn_range(): Walk through available memory range from memblock subsystem

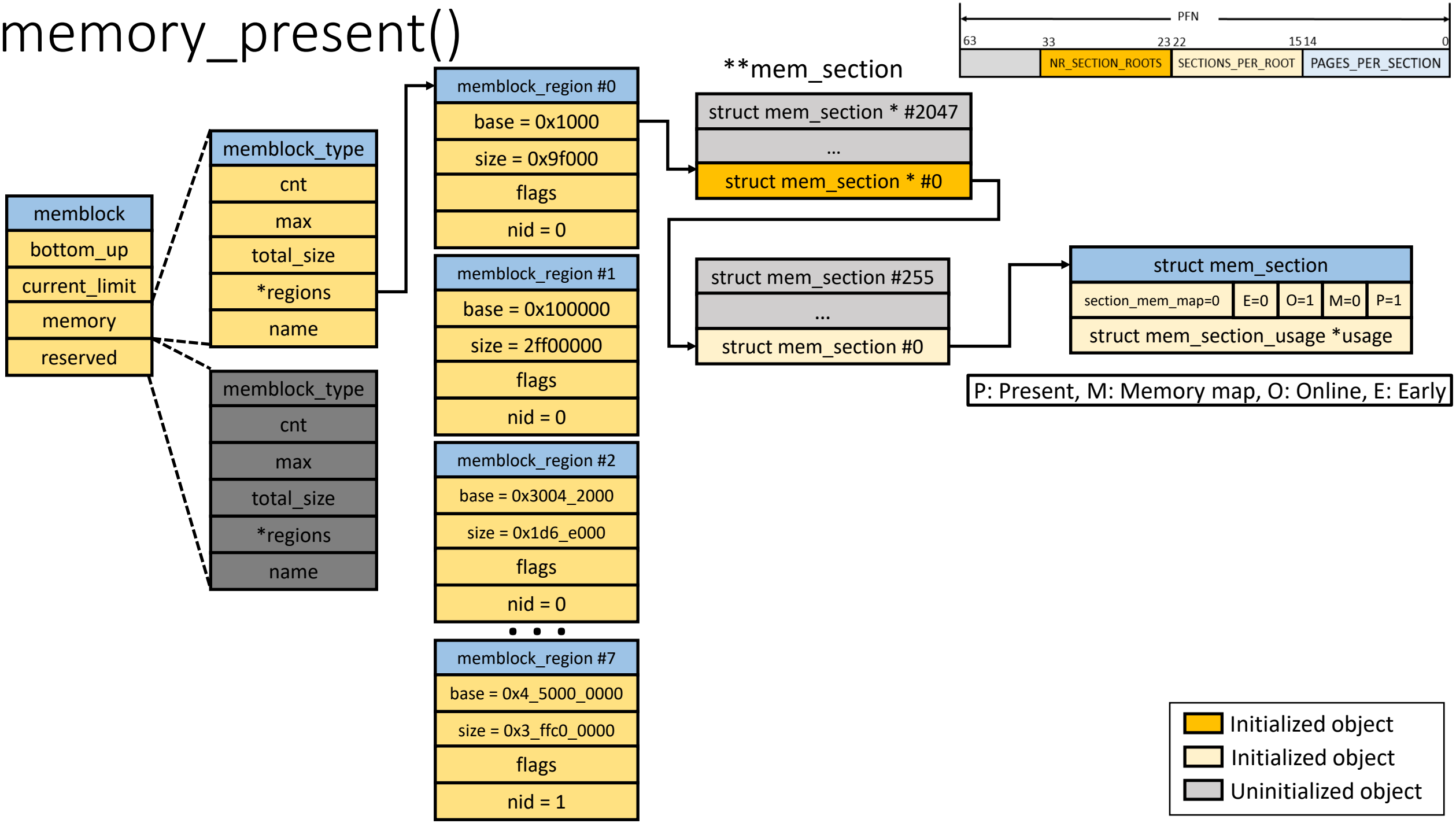
memory_present()



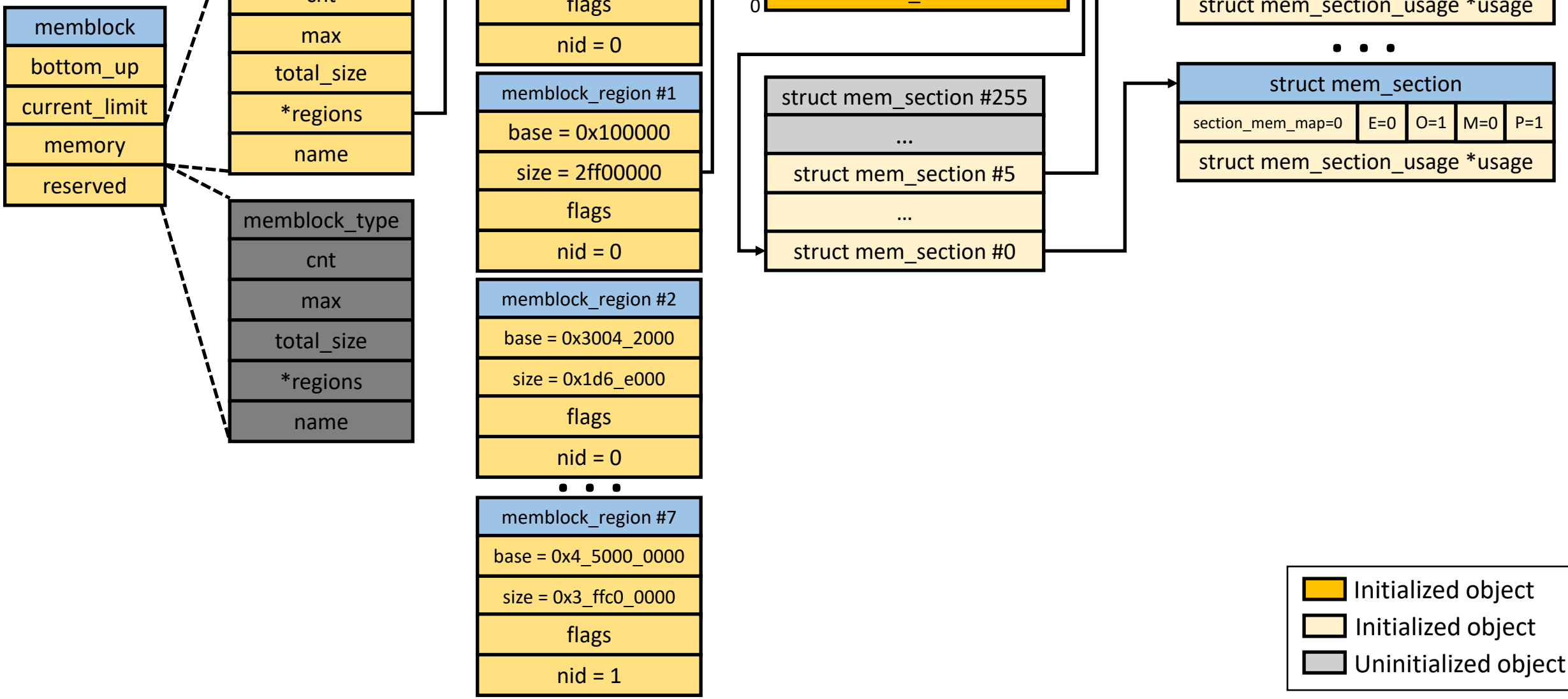




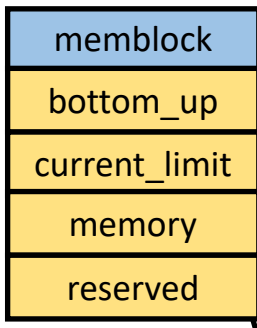
memory_present()

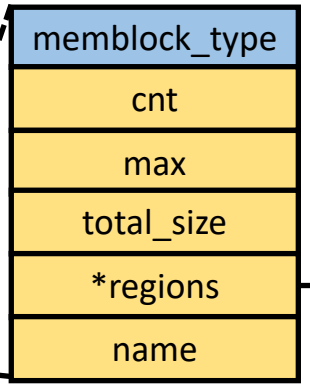


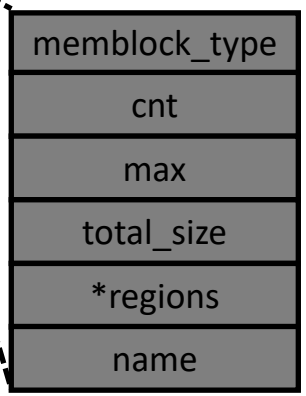
memory_present()

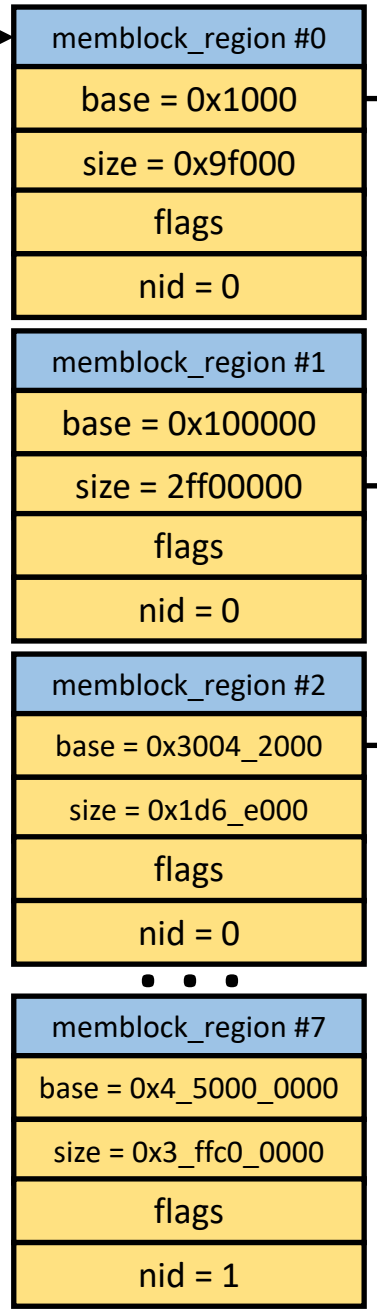


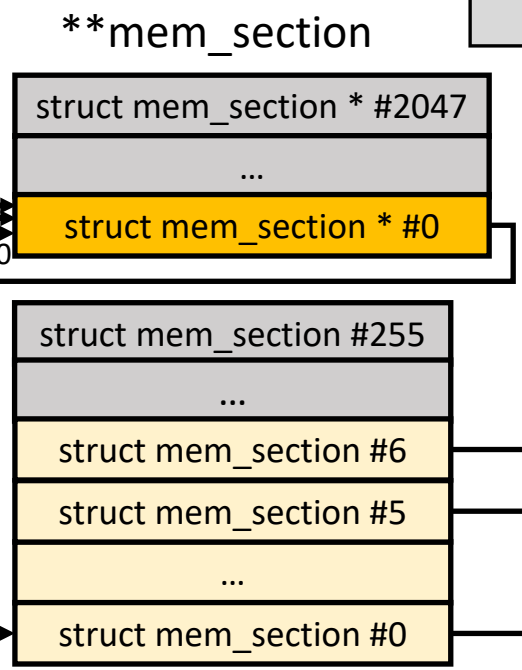
memory_present()

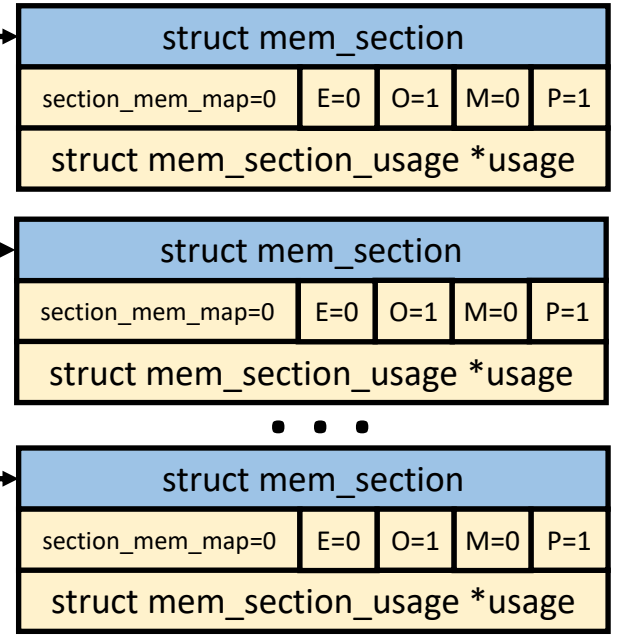


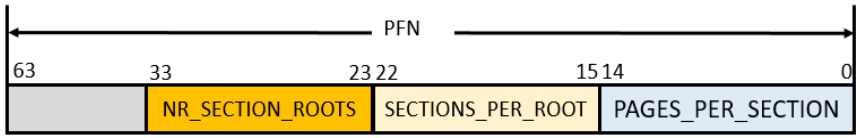


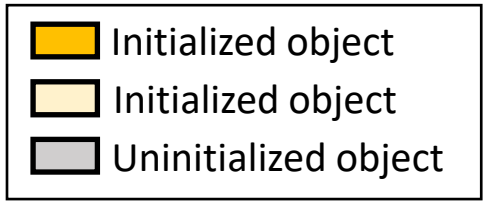




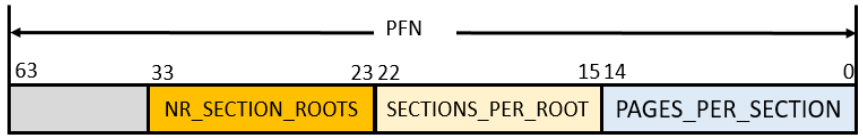








memory_present()



memblock
bottom_up
current_limit
memory
reserved

memblock_type
cnt
max
total_size
*regions
name

memblock_type
cnt
max
total_size
*regions
name

memblock_region #0
base = 0x1000
size = 0x9f000
flags
nid = 0

memblock_region #1
base = 0x100000
size = 2ff00000
flags
nid = 0

memblock_region #2
base = 0x3004_2000
size = 0x1d6_e000
flags
nid = 0

• • •

memblock_region #7
base = 0x4_5000_0000
size = 0x3_ffc0_0000
flags
nid = 1

**mem_section

struct mem_section * #2047
...
struct mem_section * #1
struct mem_section * #0

struct mem_section #255
...
struct mem_section #138
...
struct mem_section #6
struct mem_section #5
...
struct mem_section #0

struct mem_section #255
...
struct mem_section #9
...
struct mem_section #0

struct mem_section
section_mem_map=0 | E=0 | O=1 | M=0 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map=0 | E=0 | O=1 | M=0 | P=1
struct mem_section_usage *usage

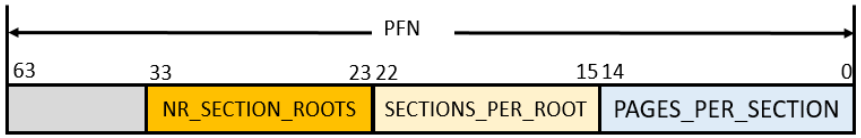
struct mem_section
section_mem_map=0 | E=0 | O=1 | M=0 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map=0 | E=0 | O=1 | M=0 | P=1
struct mem_section_usage *usage

Initialized object
Initialized object
Uninitialized object

sparse_init_nid(): cfg mem_section_map



memblock_region #0
base = 0x1000
size = 0x9f000
flags
nid = 0

memblock_region #1
base = 0x100000
size = 2ff00000
flags
nid = 0

memblock_region #2
base = 0x3004_2000
size = 0x1d6_e000
flags
nid = 0

• • •

memblock_region #7
base = 0x4_5000_0000
size = 0x3_ffc0_0000
flags
nid = 1

**mem_section

struct mem_section * #2047
...
struct mem_section * #1
struct mem_section * #0

struct mem_section #255
...
struct mem_section #138
...
struct mem_section #6
struct mem_section #5
...
struct mem_section #0

struct mem_section #255
...
struct mem_section #9
...
struct mem_section #0

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=0
struct mem_section_usage *usage

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

...
struct page #65535
....
struct page #32768
struct page #32767
...
struct page #0

section #1
section_roots #0

section #0
section_roots #0

vmemmap = VMEMMAP_START = vmemmap_base

struct mem_section_usage #n
...
struct mem_section_usage #0

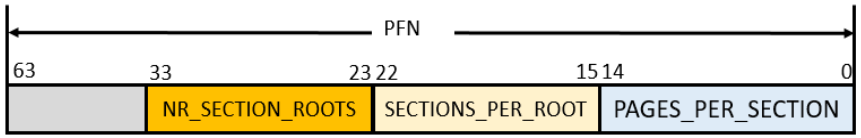
Per-node basis
Number of available 'struct mem_section (map_count)'.

Note

Allocate page structs for each mem_section and map them to the page table (Virtual Memory Map)

Initialized object
Uninitialized object

sparse_init_nid(): cfg mem_section_map



memblock_region #0
base = 0x1000
size = 0x9f000
flags
nid = 0

memblock_region #1
base = 0x100000
size = 2ff00000
flags
nid = 0

memblock_region #2
base = 0x3004_2000
size = 0x1d6_e000
flags
nid = 0

• • •

memblock_region #7
base = 0x4_5000_0000
size = 0x3_ffc0_0000
flags
nid = 1

**mem_section

struct mem_section * #2047
...
struct mem_section * #1
struct mem_section * #0

struct mem_section #255
...
struct mem_section #138
...
struct mem_section #6
struct mem_section #5
...
struct mem_section #0

struct mem_section #255
...
struct mem_section #9
...
struct mem_section #0

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=0
struct mem_section_usage *usage

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

• • •

struct mem_section
section_mem_map | E=1 | O=1 | M=1 | P=1
struct mem_section_usage *usage

...
struct page #8683520
...
struct page #8388608
struct page #8388607
...
struct page #4521984
...
...
struct page #229375
...
struct page #65535
...
struct page #32768
struct page #32767
...
struct page #0

section #0-9, section_roots #1
section #138-255, section_roots #0
section #2-6, section_roots #0
section #1, section_roots #0
section #0, section_roots #0

vmemmap = VMEMMAP_START = vmemmap_base

struct mem_section_usage #n
...
struct mem_section_usage #0

Per-node basis
Number of available 'struct mem_section (map_count)'.

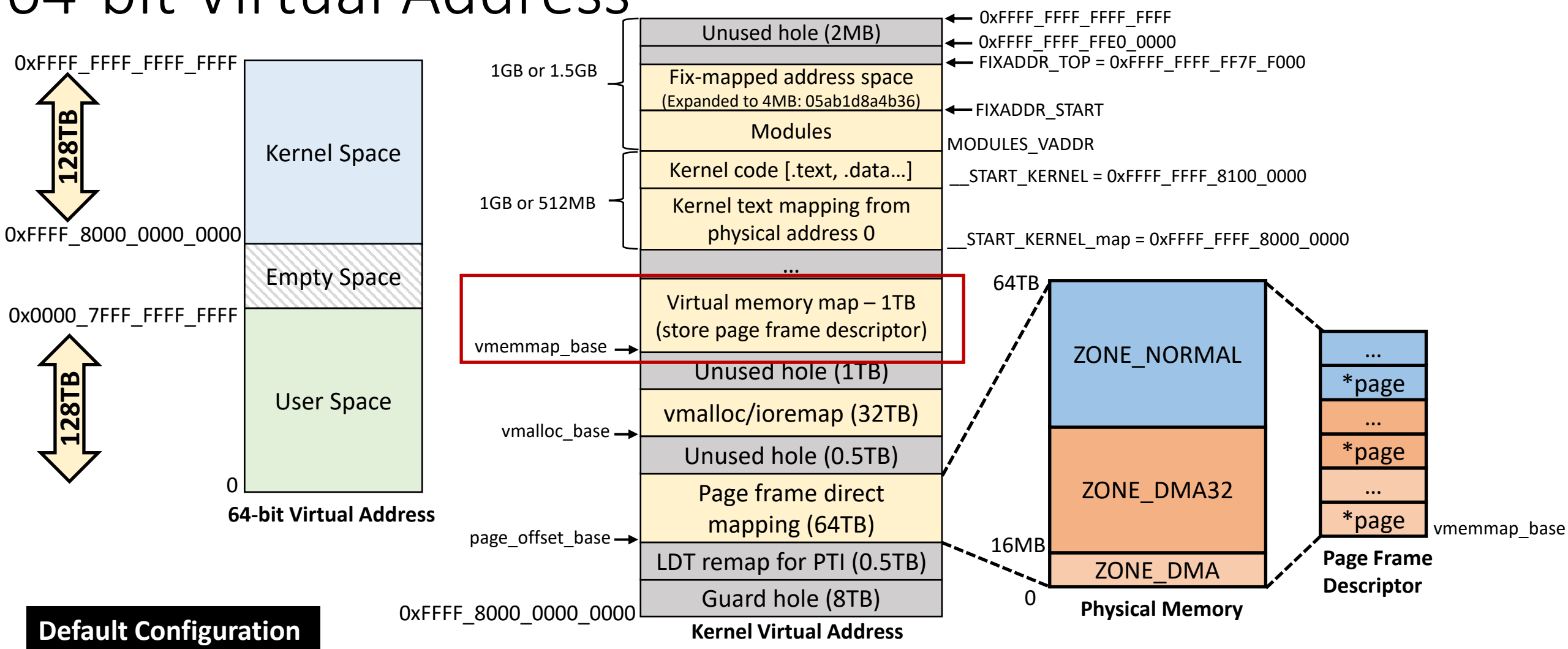
Note

Allocate page structs for each mem_section and map them to the page table (Virtual Memory Map)

Initialized object
Allocated & Uninitialized object
Unallocated object

64-bit Virtual Address

Reference: Documentation/x86/x86_64/mm.rst

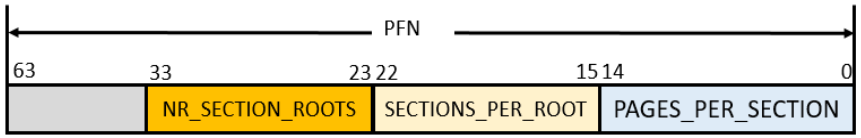


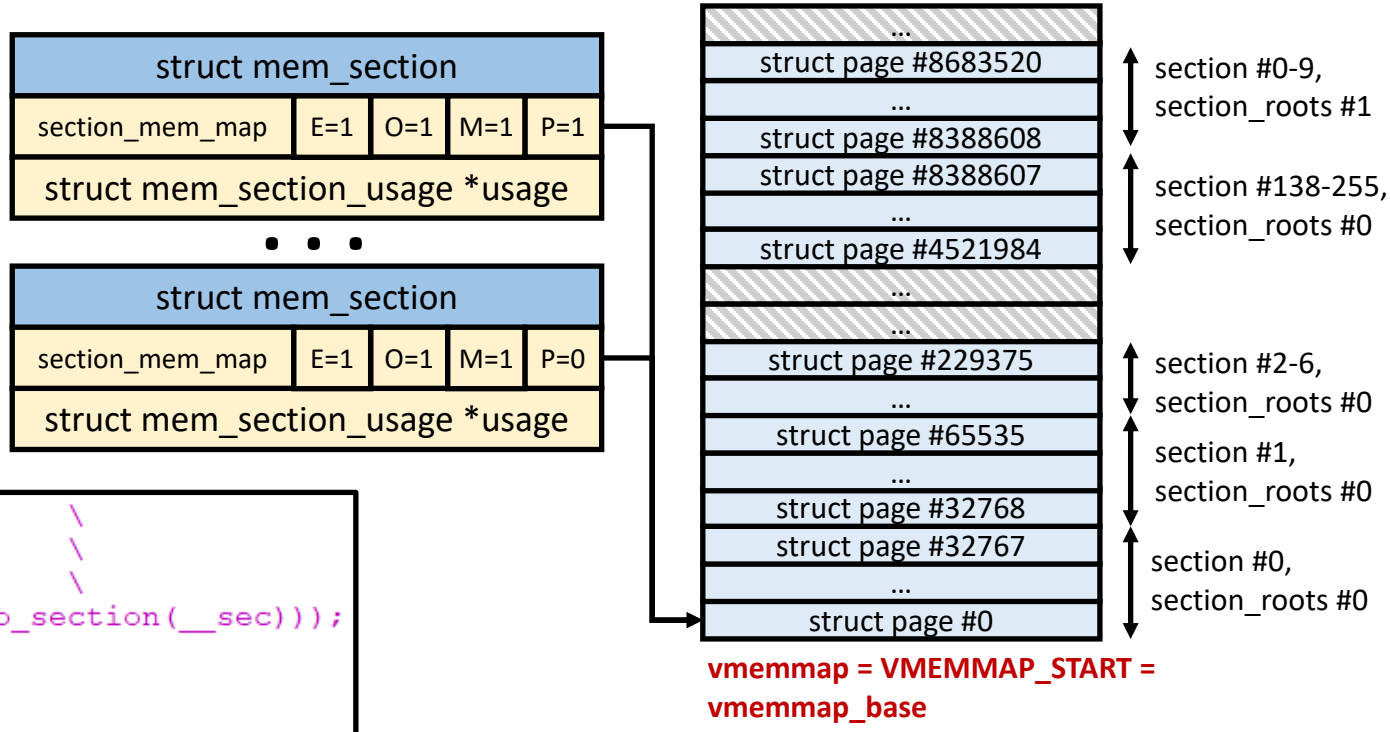
Default Configuration

page_ofset_base = 0xFFFF_8880_0000_0000
vmalloc_base = 0xFFFF_C900_0000_0000
vmemmap_base = 0xFFFF_EA00_0000_0000
* Can be dynamically configured by KASLR (Kernel Address Space Layout Randomization - "arch/x86/mm/kaslr.c")

Note: Refer from page #5 in the slide deck Decompressed vmlinux: linux kernel initialization from page table configuration perspective

Re-visit sparse memory





Sparse Memory: Refer to section_mem_map

```
#define __page_to_pfn(pg)                                       \
({      const struct page *__pg = (pg);                         \
        int __sec = page_to_section(__pg);                      \
        (unsigned long)(__pg - __section_mem_map_addr(__nr_to_section(__sec)));
\
})

#define __pfn_to_page(pfn)                                      \
({      unsigned long __pfn = (pfn);                            \
        struct mem_section *__sec = __pfn_to_section(__pfn);    \
        __section_mem_map_addr(__sec) + __pfn;                  \
})
```

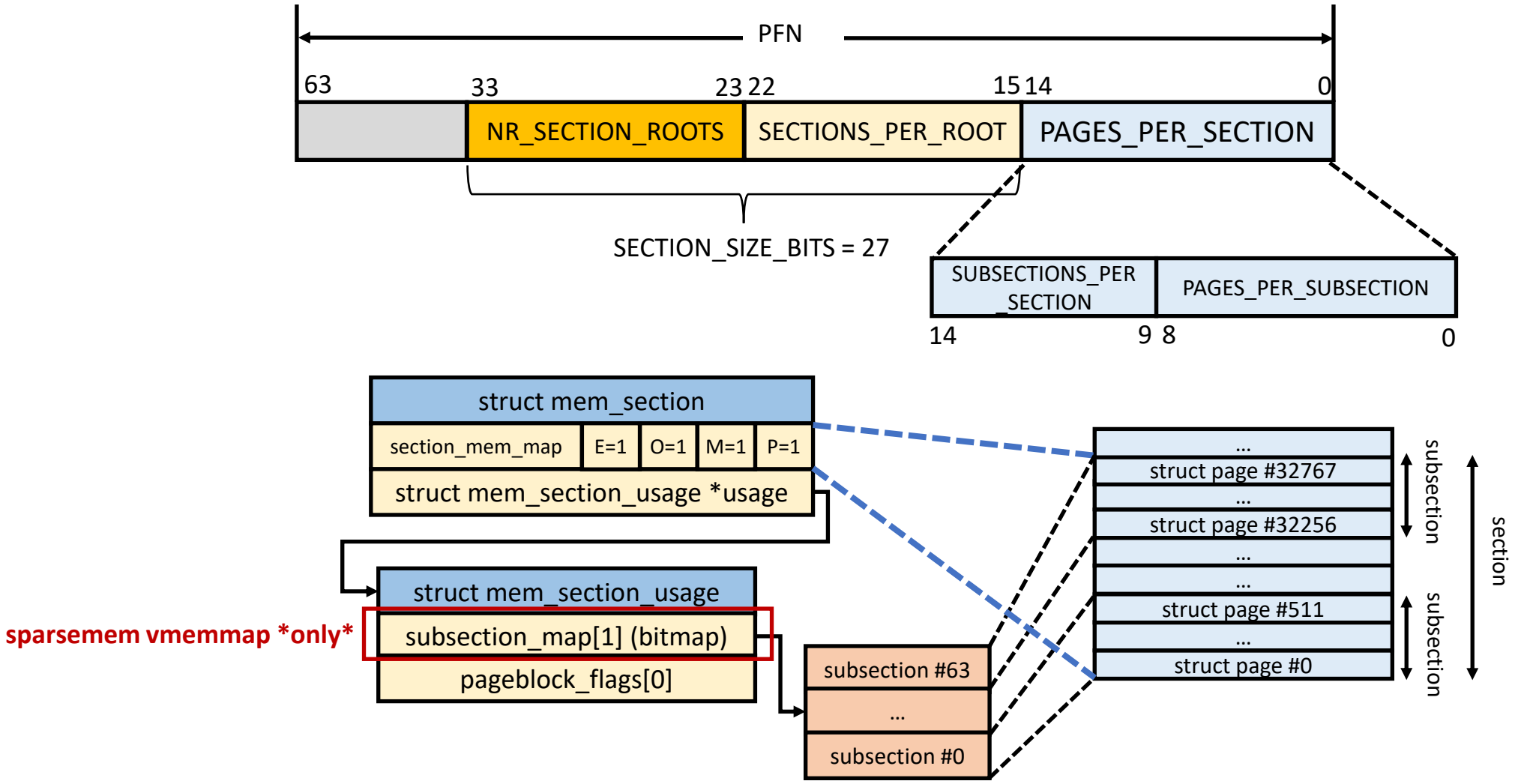
Sparse Memory with vmemmap: Refer to vmemmap

```
/* memmap is virtually contiguous.  */
#define __pfn_to_page(pfn)      (vmemmap + (pfn))
#define __page_to_pfn(page)     (unsigned long)((page) - vmemmap)
```

Sparse Memory Virtual Memmap: subsection

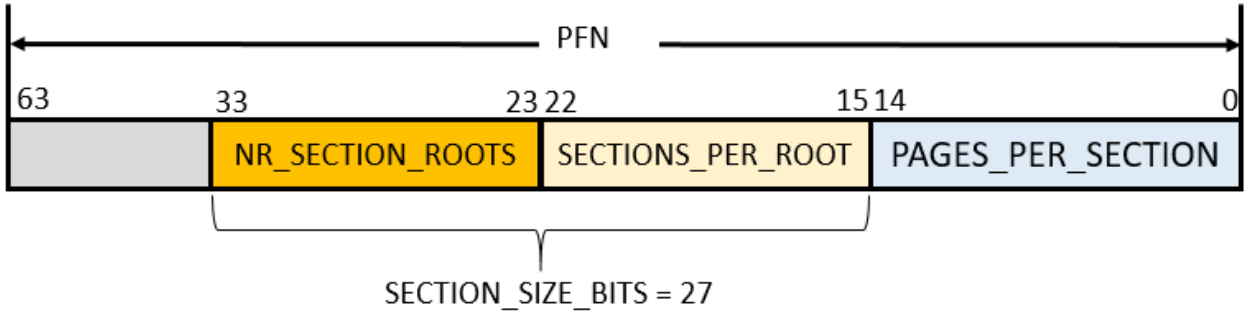
1. Introduction
2. Subsection users?
3. pageblock_flags: pageblock migration type

Sparse Memory Virtual Memmap: subsection (1/4)



- subsection_map: bitmap to indicate if the corresponding subsection is valid
- pageblock_flags: pages of a subsection have the same flag (migration type)

Sparse Memory Virtual Memmap: subsection (2/4)




```
#ifdef CONFIG_X86_32
# ifdef CONFIG_X86_PAE
#  define SECTION_SIZE_BITS     29
#  define MAX_PHYSMEM_BITS      36
# else
#  define SECTION_SIZE_BITS     26
#  define MAX_PHYSMEM_BITS      32
# endif
#else /* CONFIG_X86_32 */
# define SECTION_SIZE_BITS      27 /* matt - 128 is convenient right now */
# define MAX_PHYSMEM_BITS       (pgtable_l5_enabled() ? 52 : 46)
#endif
arch/x86/include/asm/sparsemem.h                                3,1
```

```
#ifdef CONFIG_SPARSEMEM
#include <asm/sparsemem.h>

/* SECTION_SHIFT        #bits space required to store a section # */
#define SECTIONS_SHIFT  (MAX_PHYSMEM_BITS - SECTION_SIZE_BITS) =
                        (46 - 27) = 19
include/linux/page-flags-layout.h                               7,0-1
```

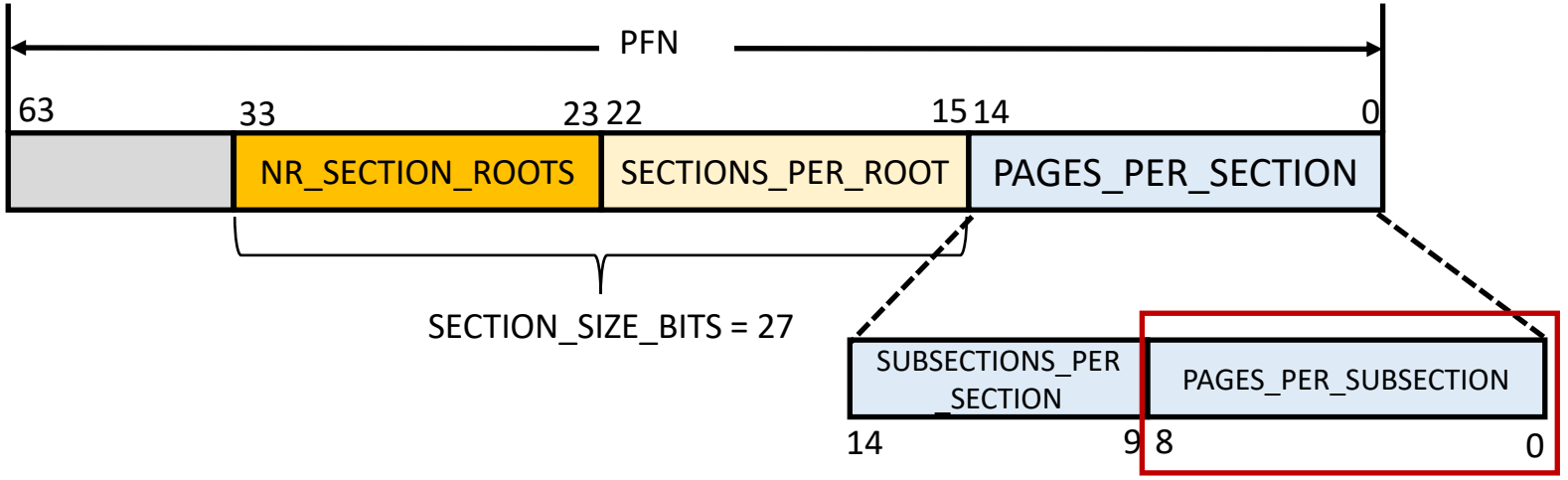
```
#ifdef CONFIG_SPARSEMEM

/*
 * SECTION_SHIFT                #bits space required to store a section #
 *
 * PA_SECTION_SHIFT             physical address to/from section number
 * PFN_SECTION_SHIFT            pfn to/from section number
 */
#define PA_SECTION_SHIFT        (SECTION_SIZE_BITS) = 27
#define PFN_SECTION_SHIFT       (SECTION_SIZE_BITS - PAGE_SHIFT) =
                                (27 - 12) = 15

#define NR_MEM_SECTIONS         (1UL << SECTIONS_SHIFT) = (1UL << 19)

#define PAGES_PER_SECTION       (1UL << PFN_SECTION_SHIFT) = (1UL << 15)
#define PAGE_SECTION_MASK       (~(PAGES_PER_SECTION-1))

#define SECTION_BLOCKFLAGS_BITS \
        ((1UL << (PFN_SECTION_SHIFT - pageblock_order)) * NR_PAGEBLOCK_BITS)
        = ((1UL << (15 - 9)) * 4)
        = (64 * 4) = 256
include/linux/mmzone.h                                          1131,0-1
```

```
#ifdef CONFIG_SPARSEMEM_EXTREME
#define SECTIONS_PER_ROOT       (PAGE_SIZE / sizeof (struct mem_section))
                                = (4096 / 16) = 256
#else
#define SECTIONS_PER_ROOT       1
#endif

#define SECTION_NR_TO_ROOT(sec) ((sec) / SECTIONS_PER_ROOT)
#define NR_SECTION_ROOTS        DIV_ROUND_UP(NR_MEM_SECTIONS, SECTIONS_PER_ROOT)
                                = ((1 << 19) / 256) = 2048
#define SECTION_ROOT_MASK       (SECTIONS_PER_ROOT - 1)
include/linux/mmzone.h                                  1218,28         86%
```

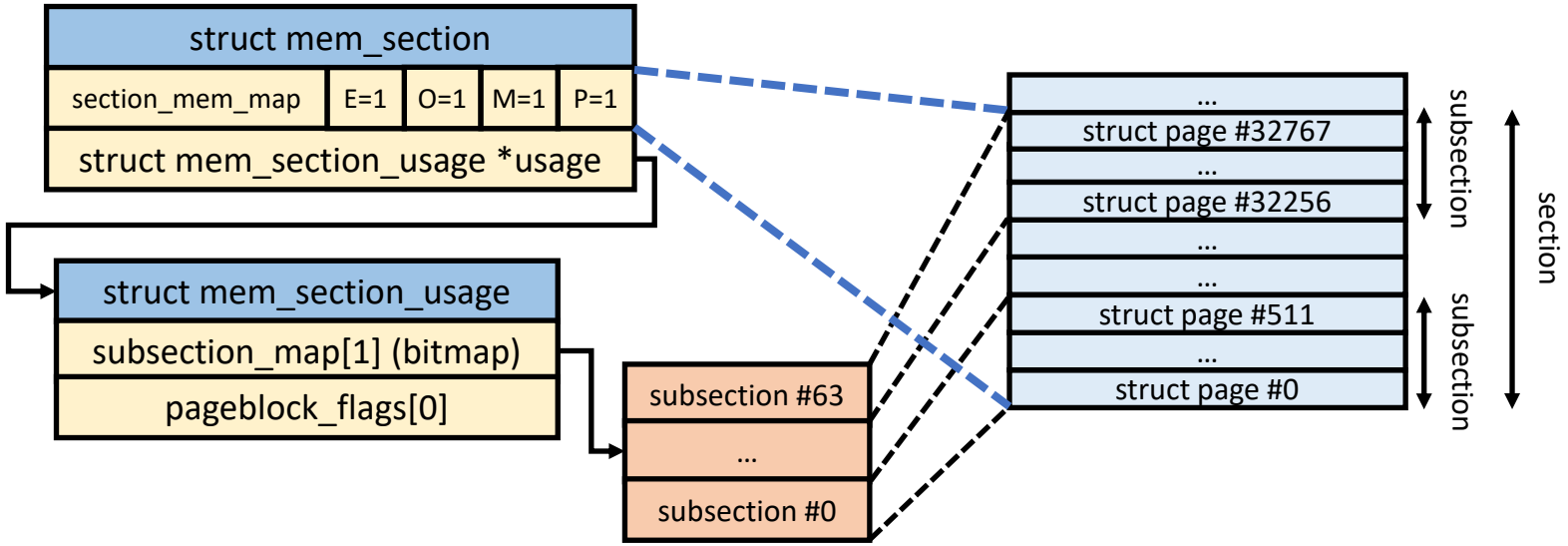
Note

Some macros are expanded manually

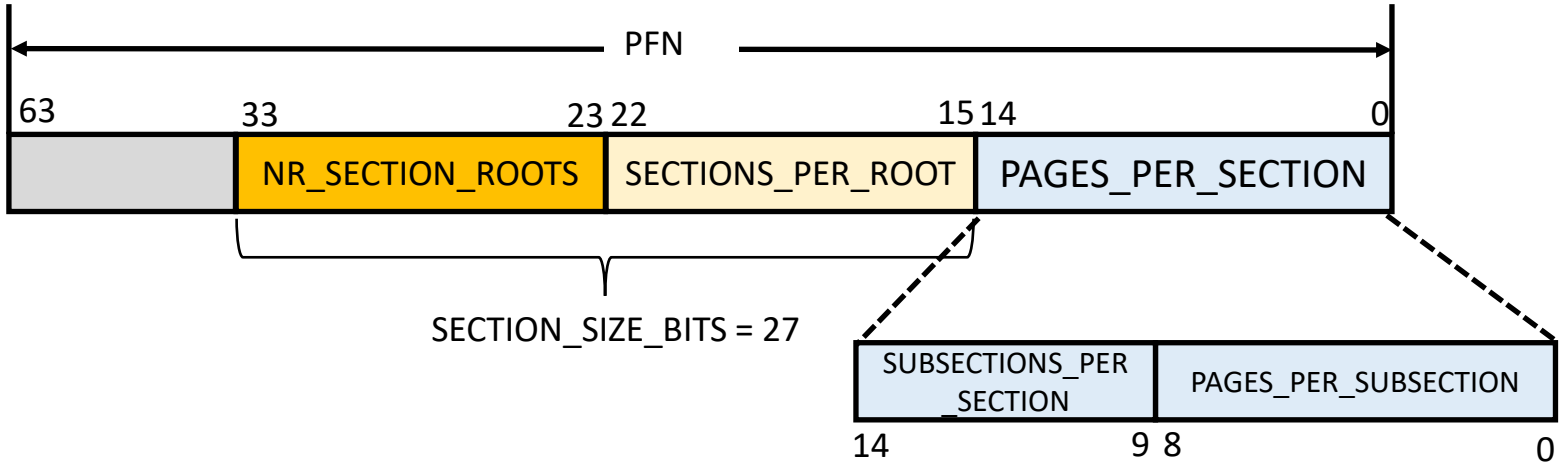
Sparse Memory Virtual Memmap: subsection (3/4)



- PAGES_PER_SUBSECTION = 512 pages
  - ✓ 512 pages * 4KB = 2MB → 2MB huge page in x86_64



Sparse Memory Virtual Memmap: subsection (4/4)



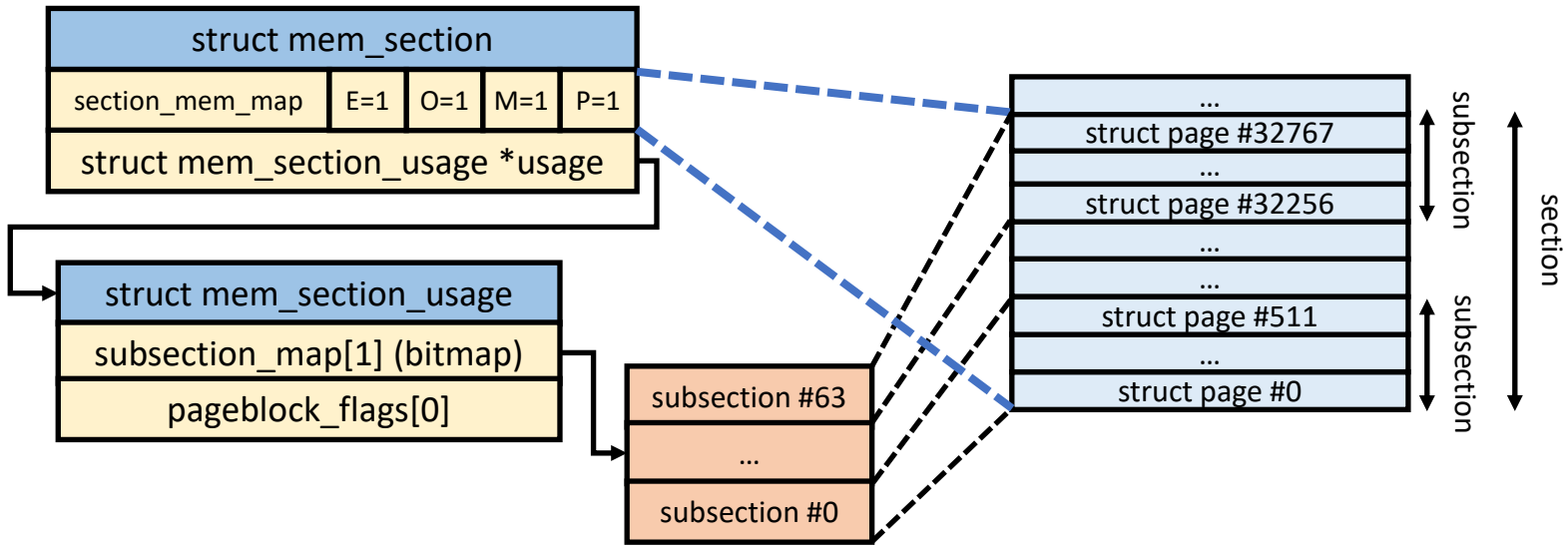

```
#define SUBSECTION_SHIFT 21
#define SUBSECTION_SIZE (1UL << SUBSECTION_SHIFT)

#define PFN_SUBSECTION_SHIFT (SUBSECTION_SHIFT - PAGE_SHIFT) = (21 - 12) = 9
#define PAGES_PER_SUBSECTION (1UL << PFN_SUBSECTION_SHIFT) = 512
#define PAGE_SUBSECTION_MASK (~(PAGES_PER_SUBSECTION-1))

#if SUBSECTION_SHIFT > SECTION_SIZE_BITS
#error Subsection size exceeds section size
#else
#define SUBSECTIONS_PER_SECTION (1UL << (SECTION_SIZE_BITS - SUBSECTION_SHIFT))
        = (1UL << (27 - 21)) = (1UL << 6) = 64
#endif
include/linux/mmzone.h                                  1171,9          82%
```

- SUBSECTION_SIZE
  - ✓ (1UL << 21) = 2MB → 2MB huge page in x86_64.

Note

Some macros are expanded manually

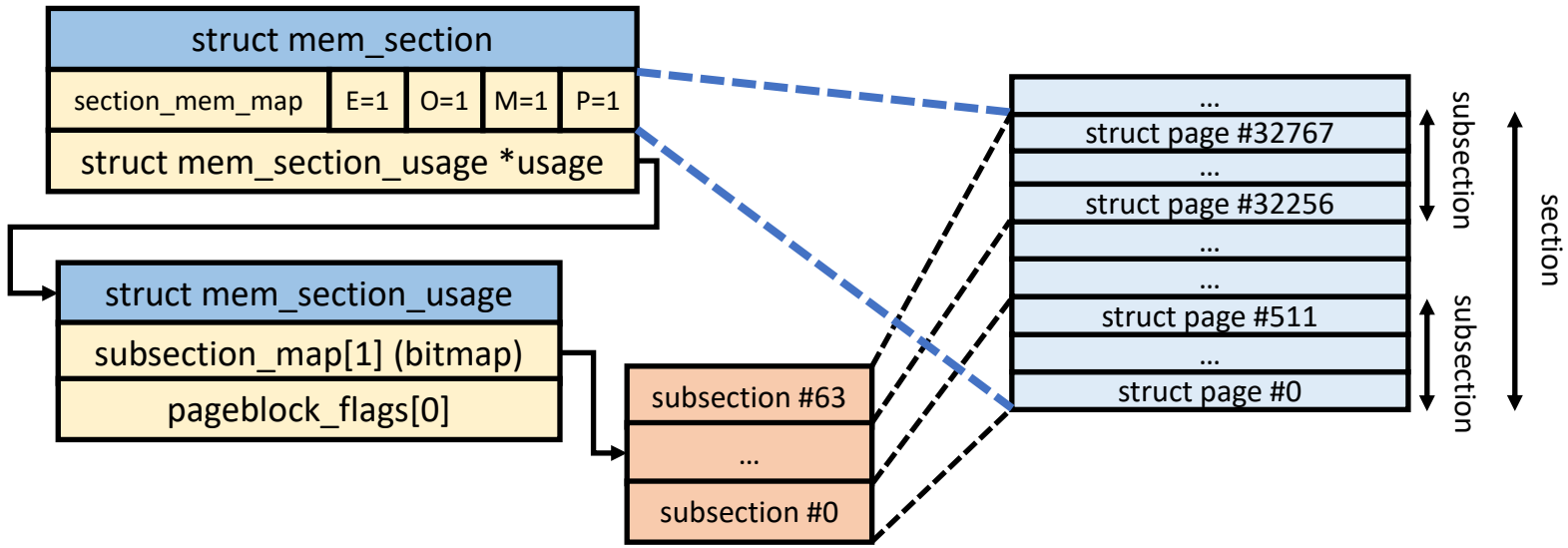
subsection: subsection_map users?



subsection_map users

- init stage
  - ✓ paging_init -> zone_sizes_init -> free_area_init -> subsection_map_init -> subsection_mask_set
    - ➢ Set the corresponding bit map for the specific subsection
- Reference stage
  - ✓ pfn_section_valid(struct mem_section *ms, unsigned long pfn)
    - ➢ Users
      - [mm/page_alloc.c: 5089] free_pages -> virt_addr_valid -> __virt_addr_valid -> pfn_valid -> pfn_section_valid
      - [drivers/char/mem.c: 416] mmap_kmem -> pfn_valid -> pfn_section_valid ➔ /dev/mem (`man mem`)
      - ...

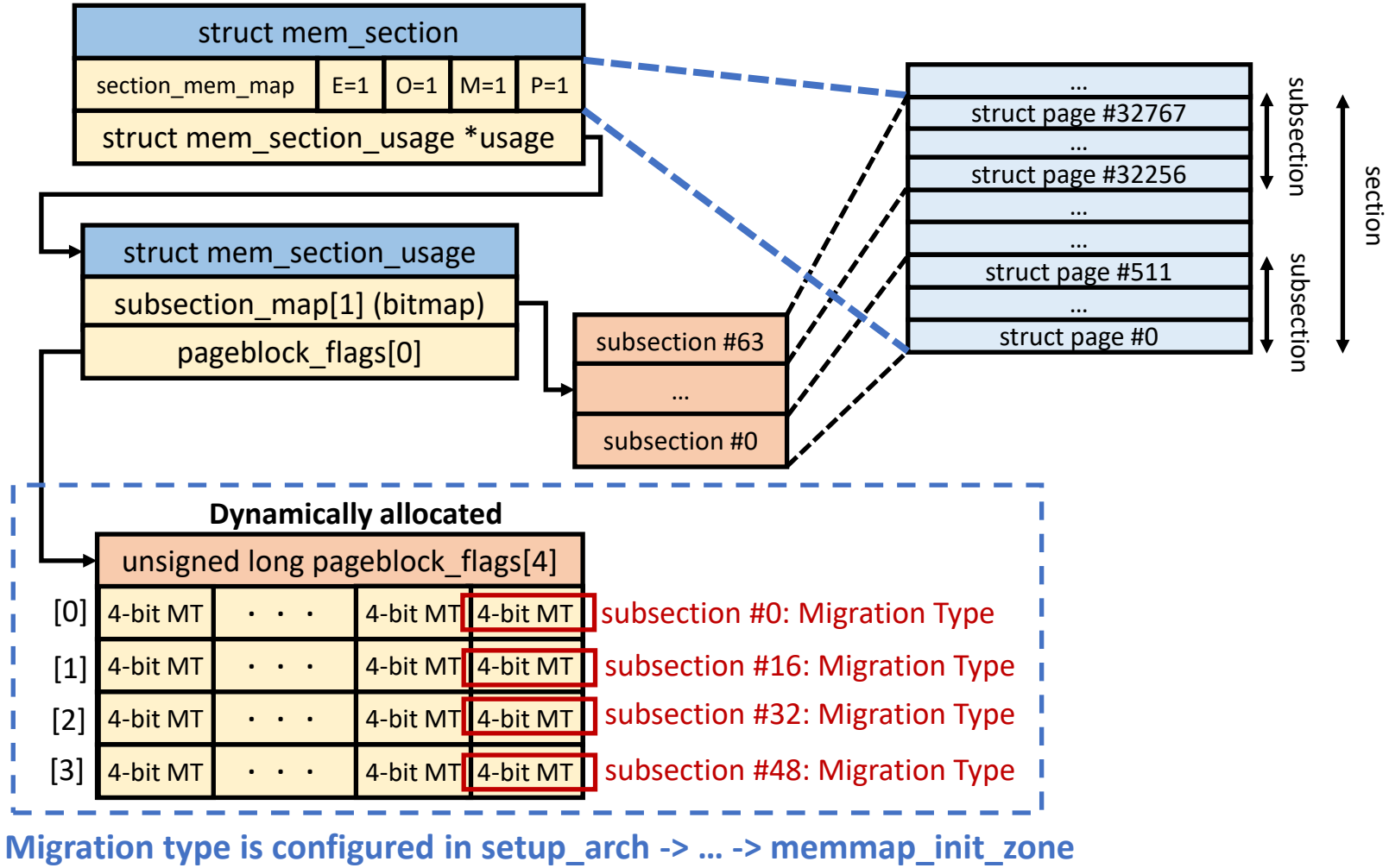
subsection: subsection_map users?



subsection_map users

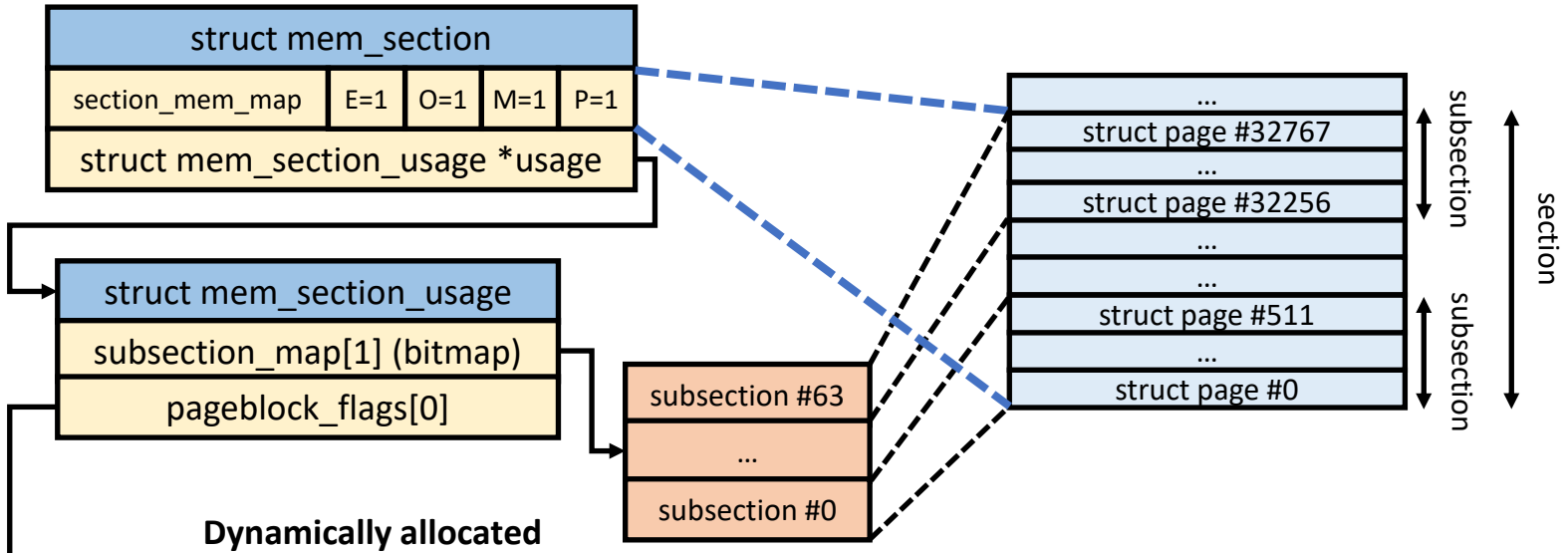
- Hotplug stage
  - ✓ Add
    - ➢ #A1 [drivers/acpi/acpi_memhotplug.c: 311] acpi_memory_device_add -> acpi_memory_enable_device -> __add_memory -> add_memory_resource -> arch_add_memory -> add_pages -> __add_pages -> **sparse_add_section** -> section_activate -> fill_subsection_map -> subsection_mask_set
    - ➢ #A2 [drivers/dax/kmem.c: 43] dev_dax_kmem_probe -> add_memory_driver_managed -> add_memory_resource -> same with #A1
  - ✓ Remove
    - ➢ #R1 [drivers/acpi/acpi_memhotplug.c: 311] acpi_memory_device_remove -> __remove_memory -> try_remove_memory -> arch_remove_memory -> __remove_pages -> __remove_section -> **sparse_remove_section** -> section_deactivate -> clear_subsection_map
    - ➢ #R2 [drivers/dax/kmem.c: 139] dev_dax_kmem_remove -> remove_memory -> try_remove_memory -> same with #R1

pageblock_flags: pageblock migration type



Migration type is configured in setup_arch -> … -> memmap_init_zone

pageblock_flags: pageblock migration type



Dynamically allocated

unsigned long pageblock_flags[4]

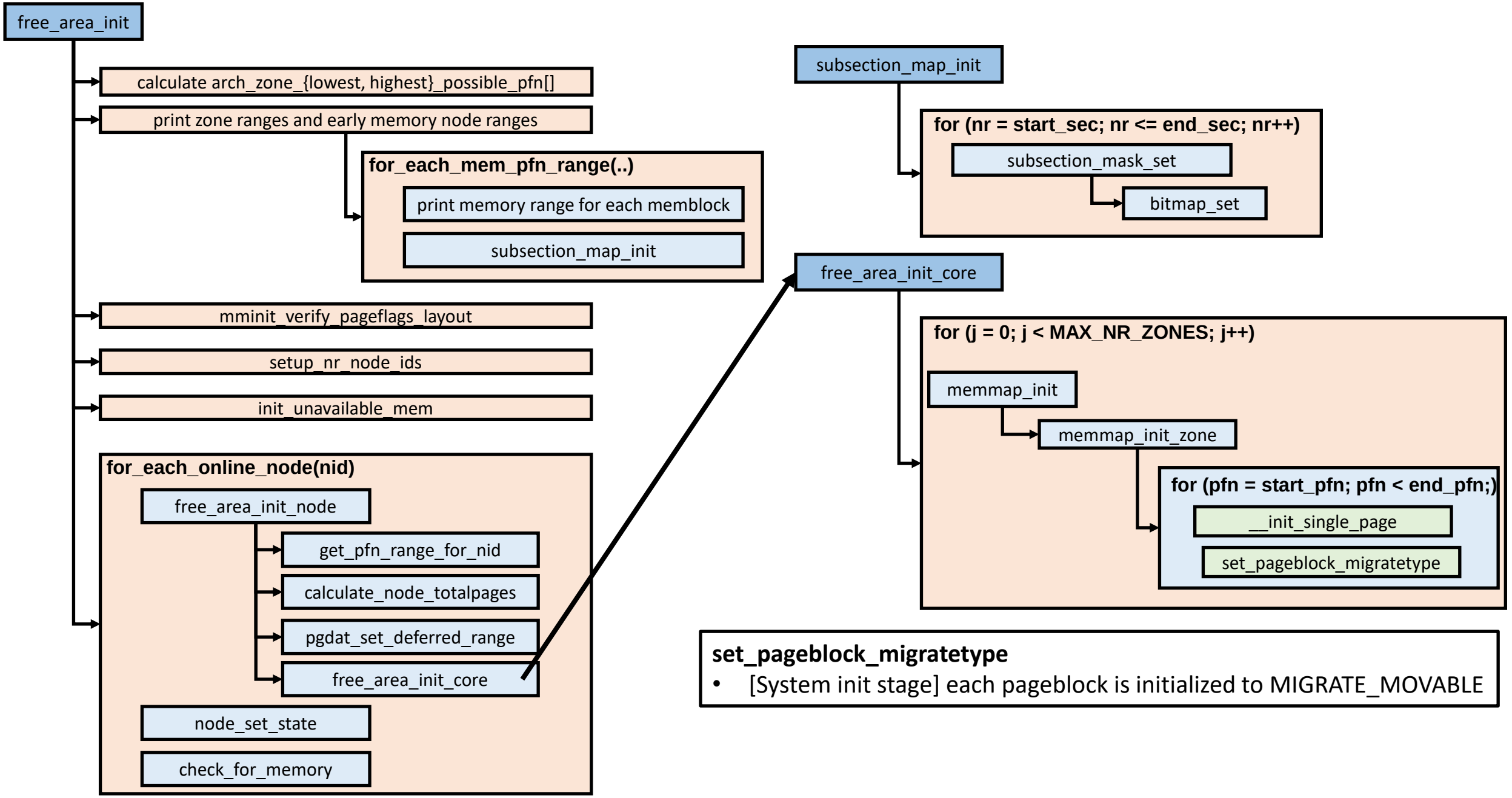
[0]	4-bit MT	· · ·	4-bit MT	4-bit MT	subsection #0: Migration Type
[1]	4-bit MT	· · ·	4-bit MT	4-bit MT	subsection #16: Migration Type
[2]	4-bit MT	· · ·	4-bit MT	4-bit MT	subsection #32: Migration Type
[3]	4-bit MT	· · ·	4-bit MT	4-bit MT	subsection #48: Migration Type

```
#define SECTION_BLOCKFLAGS_BITS \
        ((1UL << (PFN_SECTION_SHIFT - pageblock_order)) * NR_PAGEBLOCK_BITS)
        = ((1UL << (15 - 9)) * 4)
        = (64 * 4) = 256
include/linux/mmzone.h                                          1132,21
```

```
#define PB_migratetype_bits 3
/* Bit indices that affect a whole block of pages */
enum pageblock_bits {
        PB_migrate,
        PB_migrate_end = PB_migrate + PB_migratetype_bits - 1,
                        /* 3 bits required for migrate types */
        PB_migrate_skip,/* If set the block is skipped by compaction */

        /*
         * Assume the bits will always align on a word. If this assumption
         * changes then get/set pageblock needs updating.
         */
        NR_PAGEBLOCK_BITS
};
include/linux/pageblock-flags.h                                 5,1
```

```
enum migratetype {
        MIGRATE_UNMOVABLE,
        MIGRATE_MOVABLE,
        MIGRATE_RECLAIMABLE,
        MIGRATE_PCPTYPES,       /* the number of types on the pcp lists */
        MIGRATE_HIGHATOMIC = MIGRATE_PCPTYPES,
#ifdef CONFIG_CMA
+-- 13 lines: MIGRATE_CMA migration type is designed to mimic the way-----
        MIGRATE_CMA,
#endif
#ifdef CONFIG_MEMORY_ISOLATION
        MIGRATE_ISOLATE,        /* can't allocate from here */
#endif
        MIGRATE_TYPES
```

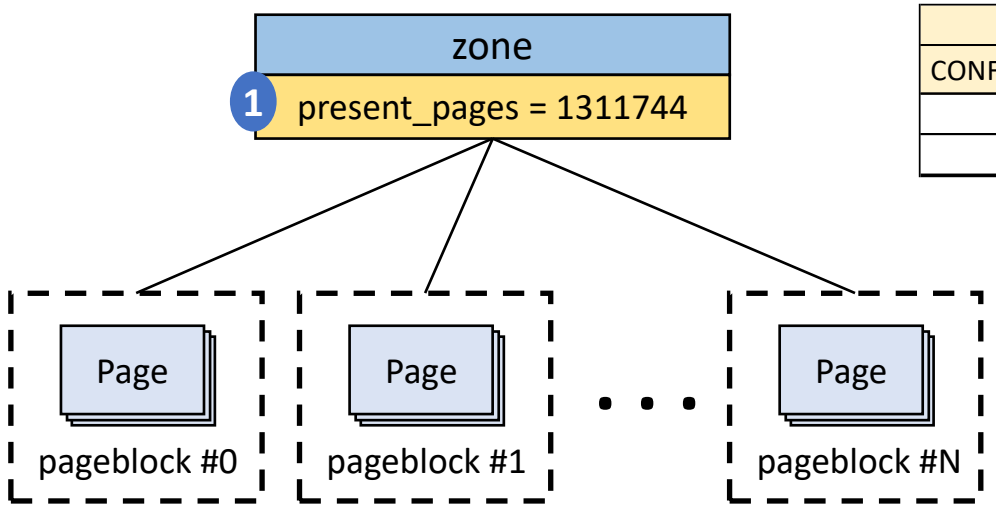
pageblock: set migration type



set_pageblock_migratetype

- [System init stage] each pageblock is initialized to MIGRATE_MOVABLE

pageblock



pageblock size	
CONFIG_HUGETLB_PAGE	Number of Pages
Y	512 = Huge page size
N	1024 (MAX_ORDER - 1)

```
(gdb) p node_data[0]->node_zones[1]
$5 = {
  _watermark = {7040, 8800, 10560},
  watermark_boost = 0,
  nr_reserved_highatomic = 0,
  lowmem_reserve = {0, 0, 0},
  node = 0,
  zone_pgdat = 0xffff8882403fa000,
  pageset = 0x21d28,
  pageset_high = 378,
  pageset_batch = 63,
  zone_start_pfn = 1048576,
  managed_pages = {
    counter = 1275118
  },
  spanned_pages = 1311744,
1 present_pages = 1311744,
  name = 0xffffffff818f86b5 "Normal",
  ...
}
```

N = round_up(present_pages / pageblock_size) - 1

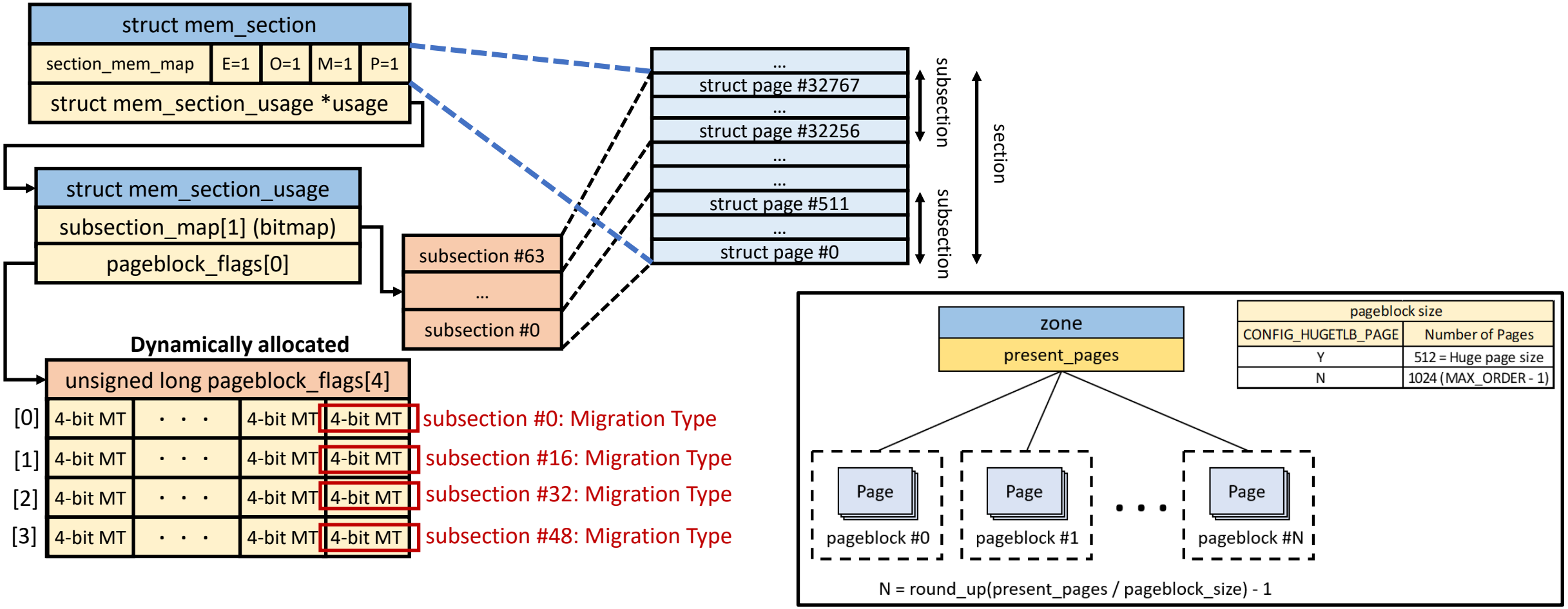
2 **Example**

pageblocks = round_up(1311744 / 512) = 2562

```
/ # cat /proc/pagetypeinfo
Page block order: 9
Pages per block:  512

Free pages count per migrate type at order       0      1      2      3      4      5      6      7      8      9     10
Node    0, zone      DMA32, type    Unmovable      0      1      0      1      0      0      1      0      1      0      0
Node    0, zone      DMA32, type      Movable      3      4      7      6      4      4      5      4      2      4    745
Node    0, zone      DMA32, type  Reclaimable      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone      DMA32, type   HighAtomic      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone     Normal, type    Unmovable      0      1      0      1      0      1      1      0      0      1      0
Node    0, zone     Normal, type      Movable      0      0      0      0      1      1      3      1      1      1   1235
Node    0, zone     Normal, type  Reclaimable      0      0      1      1      0      1      0      1      1      0      0
Node    0, zone     Normal, type   HighAtomic      0      0      0      0      0      0      0      0      0      0      0

Number of blocks type     Unmovable      Movable  Reclaimable   HighAtomic
Node 0, zone      DMA32             1         1535             0            0
Node 0, zone     Normal            16         2544             2            0
```

2 16 + 2544 + 2 = 2562

pageblock_flags: pageblock migration type

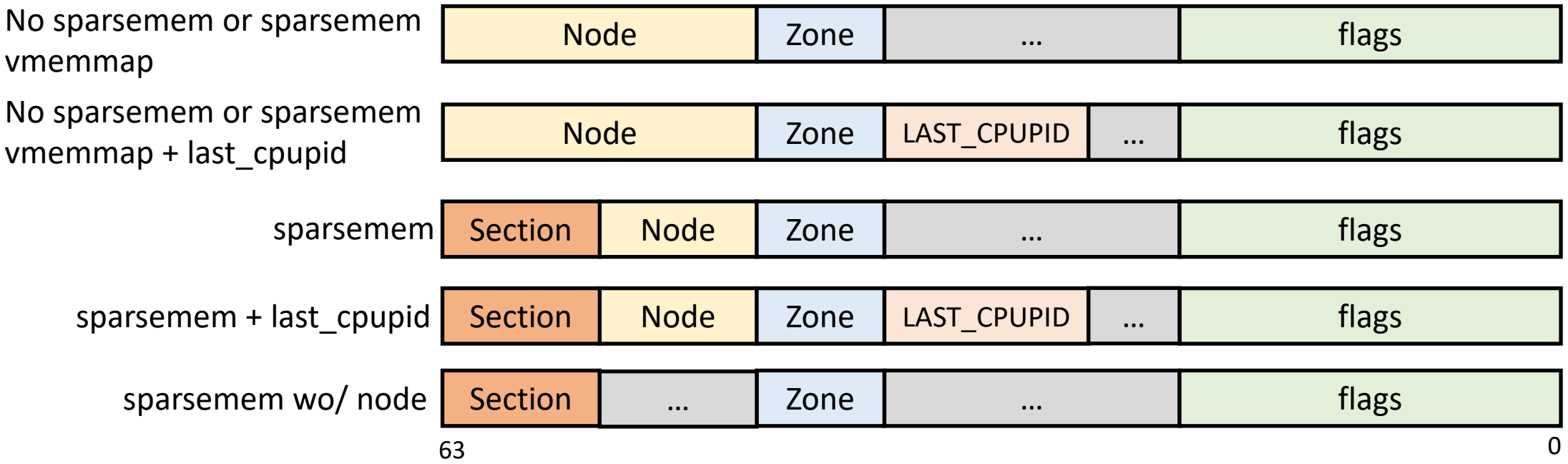


[CONFIG_HUGETLB_PAGE=y]

pages of subsection = pages of pageblock = 512 pages (order = 9)

page->flags layout

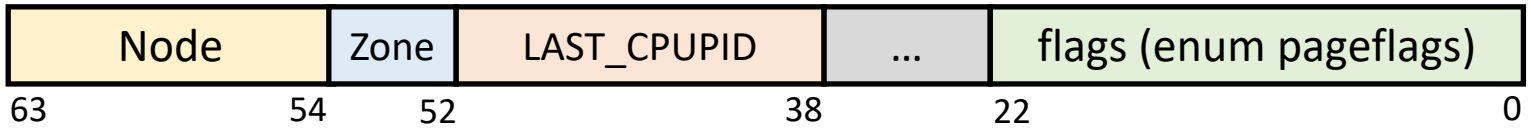
page->flags layout



Note

1. last_cpupid: Support for NUMA balancing (NUMA-optimizing scheduler)
2. sparsemem: Enabled by CONFIG_SPARSEMEM

page->flags layout: sparsemem vmemmap + last_cpupid



Kernel Configuration: qemu – v5.11 kernel

```
…
CONFIG_NUMA_BALANCING=y
CONFIG_NUMA_BALANCING_DEFAULT_ENABLED=y
…
CONFIG_NR_CPUS=64
…
CONFIG_NODES_SHIFT=10
…
CONFIG_SPARSEMEM_MANUAL=y
CONFIG_SPARSEMEM=y
CONFIG_NEED_MULTIPLE_NODES=y
CONFIG_SPARSEMEM_EXTREME=y
CONFIG_SPARSEMEM_VMEMMAP_ENABLE=y
CONFIG_SPARSEMEM_VMEMMAP=y
…
# CONFIG_KASAN is not set
```

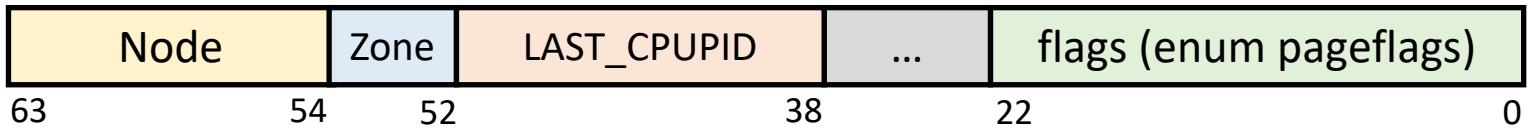
23-bit pageflags

```
(gdb) ptype enum pageflags
type = enum pageflags {PG_locked, PG_referenced, PG_uptodate, PG_dirty, PG_lru,
PG_active, PG_workingset, PG_waiters, PG_error, PG_slab, PG_owner_priv_1, PG_arc
h_1, PG_reserved, PG_private, PG_private_2, PG_writeback, PG_head, PG_mappedtodi
sk, PG_reclaim, PG_swapbacked, PG_unevictable, PG_mlocked, PG_arch_2, __NR_PAGEF
LAGS, PG_checked = 10, PG_swapcache = 10, PG_fscache = 14, PG_pinned = 10, PG_sa
vepinned = 3, PG_foreign = 10, PG_xen_remapped = 10, PG_slob_free = 13, PG_doubl
e_map = 6, PG_isolated = 18, PG_reported = 2}
(gdb) p (int) __NR_PAGEFLAGS
$9 = 23
(gdb) p NR_PAGEFLAGS
$10 = 23
```

2-bit zone

```
(gdb) ptype enum zone_type
type = enum zone_type {ZONE_DMA32, ZONE_NORMAL, ZONE_MOVABLE, __MAX_NR_ZONES}
```

page->flags layout - sparsemem vmemmap + last_cpupid




```
(gdb) ptype enum zone_type
type = enum zone_type {ZONE_DMA32, ZONE_NORMAL, ZONE_MOVABLE, __MAX_NR_ZONES}
(gdb) p (int) __MAX_NR_ZONES
$13 = 3
```

```
#if defined(CONFIG_SPARSEMEM) && !defined(CONFIG_SPARSEMEM_VMEMMAP)
#define SECTIONS_WIDTH          SECTIONS_SHIFT
#else
#define SECTIONS_WIDTH          0
#endif
include/linux/page-flags-layout.h                    34,0-1
```

```
#ifdef CONFIG_NODES_SHIFT
#define NODES_SHIFT     CONFIG_NODES_SHIFT = 10
#else
#define NODES_SHIFT     0
#endif
include/linux/numa.h
```

```
#if SECTIONS_WIDTH+ZONES_WIDTH+NODES_SHIFT <= BITS_PER_LONG - NR_PAGEFLAGS
#define NODES_WIDTH             NODES_SHIFT = 10
#else
#ifdef CONFIG_SPARSEMEM_VMEMMAP
#error "Vmemmap: No space for nodes field in page flags"
#endif
#define NODES_WIDTH             0
#endif
include/linux/page-flags-layout.h                    55,6
```

```
(gdb) p SECTIONS_WIDTH
$3 = 0
(gdb) p NODES_WIDTH
$4 = 10
(gdb) p ZONES_WIDTH
$5 = 2
(gdb) p LAST_CPUPID_WIDTH
$6 = 14
(gdb) p KASAN_TAG_WIDTH
$7 = 0
```

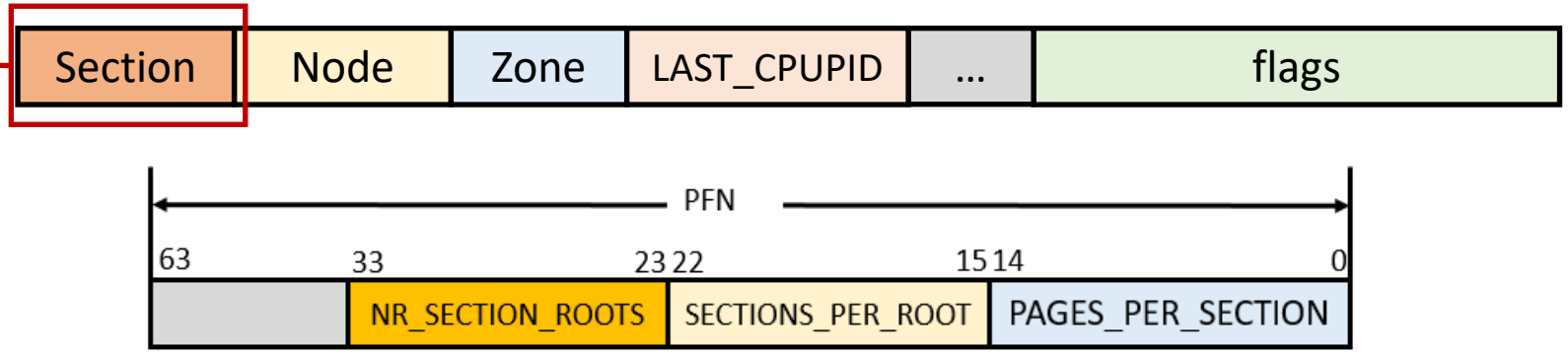
Kernel Configuration: qemu – v5.11 kernel

```
...
CONFIG_NUMA_BALANCING=y
CONFIG_NUMA_BALANCING_DEFAULT_ENABLED=y
...
CONFIG_NR_CPUS=64
...
CONFIG_NODES_SHIFT=10
...
CONFIG_SPARSEMEM_MANUAL=y
CONFIG_SPARSEMEM=y
CONFIG_NEED_MULTIPLE_NODES=y
CONFIG_SPARSEMEM_EXTREME=y
CONFIG_SPARSEMEM_VMEMMAP_ENABLE=y
CONFIG_SPARSEMEM_VMEMMAP=y
...
# CONFIG_KASAN is not set
```

```
/* Page flags: | [SECTION] | [NODE] | ZONE | [LAST_CPUPID] | ... | FLAGS | */
#define SECTIONS_PGOFF          ((sizeof(unsigned long)*8) - SECTIONS_WIDTH)
                                = (64 - 0) = 64
#define NODES_PGOFF             (SECTIONS_PGOFF - NODES_WIDTH) = 64 - 10 - 54
#define ZONES_PGOFF             (NODES_PGOFF - ZONES_WIDTH) = 54 - 2 = 52
#define LAST_CPUPID_PGOFF       (ZONES_PGOFF - LAST_CPUPID_WIDTH)
                                = 52 - 14 = 38
#define KASAN_TAG_PGOFF         (LAST_CPUPID_PGOFF - KASAN_TAG_WIDTH)
                                = 38 - 0 = 38

+--  5 lines: Define the bit shifts to access each section.  For non-existent---
#define SECTIONS_PGSHIFT        (SECTIONS_PGOFF * (SECTIONS_WIDTH != 0)) = 0
#define NODES_PGSHIFT           (NODES_PGOFF * (NODES_WIDTH != 0)) = 54
#define ZONES_PGSHIFT           (ZONES_PGOFF * (ZONES_WIDTH != 0)) = 52
#define LAST_CPUPID_PGSHIFT     (LAST_CPUPID_PGOFF * (LAST_CPUPID_WIDTH != 0))
                                = 38
#define KASAN_TAG_PGSHIFT       (KASAN_TAG_PGOFF * (KASAN_TAG_WIDTH != 0))
                                = 0
include/linux/mm.h                                   1054,1        33%
```

page->flags: section field (sparsemem wo/ vmemmap)

sparsemem + last_cpupid



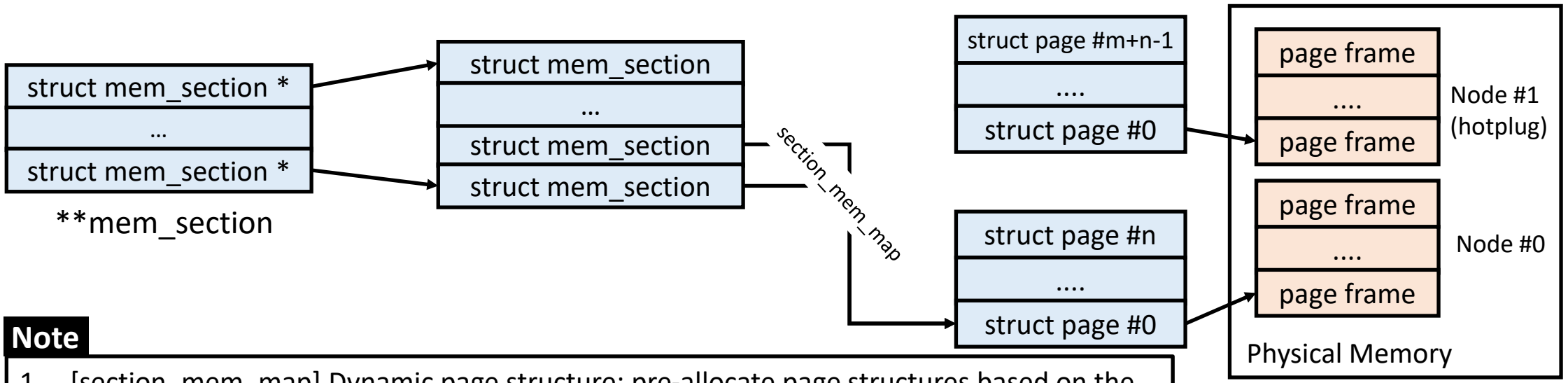
Sparse Memory: Refer to section_mem_map

```
#define __page_to_pfn(pg)                                               \
({      const struct page *__pg = (pg);                                 \
        int __sec = page_to_section(__pg);                              \
        (unsigned long)(__pg - __section_mem_map_addr(__nr_to_section(__sec)));
\
})

#define __pfn_to_page(pfn)                                      \
({      unsigned long __pfn = (pfn);                            \
        struct mem_section *__sec = __pfn_to_section(__pfn);    \
        __section_mem_map_addr(__sec) + __pfn;                  \
})
```

```
#ifdef SECTION_IN_PAGE_FLAGS
static inline void set_page_section(struct page *page, unsigned long section)
{
        page->flags &= ~(SECTIONS_MASK << SECTIONS_PGSHIFT);
        page->flags |= (section & SECTIONS_MASK) << SECTIONS_PGSHIFT;
}

static inline unsigned long page_to_section(const struct page *page)
{
        return (page->flags >> SECTIONS_PGSHIFT) & SECTIONS_MASK;
}
#endif
include/linux/mm.h                                              1468,1
```

Memory Model – Sparse Memory (sparsemem wo/ vmemmap)

```
#define __page_to_pfn(pg)                                       \
({      const struct page *__pg = (pg);                         \
        int __sec = page_to_section(__pg);                      \
        (unsigned long)(__pg - __section_mem_map_addr(__nr_to_section(__sec)));
\
})

#define __pfn_to_page(pfn)                                      \
({      unsigned long __pfn = (pfn);                            \
        struct mem_section *__sec = __pfn_to_section(__pfn);    \
        __section_mem_map_addr(__sec) + __pfn;                  \
})
```



Note

1. [section_mem_map] Dynamic page structure: pre-allocate page structures based on the number of available page frames
   - ✓ Refer from: **memblock structure**
2. Support physical memory hotplug
3. Minimum unit: mem_section - PAGES_PER_SECTION = 32768
   - ✓ Each memory section addresses the memory size: 32768 * 4KB (page size) = 128MB
4. [NUMA] : reduce the memory hole impact due to “struct mem_section”

Reference

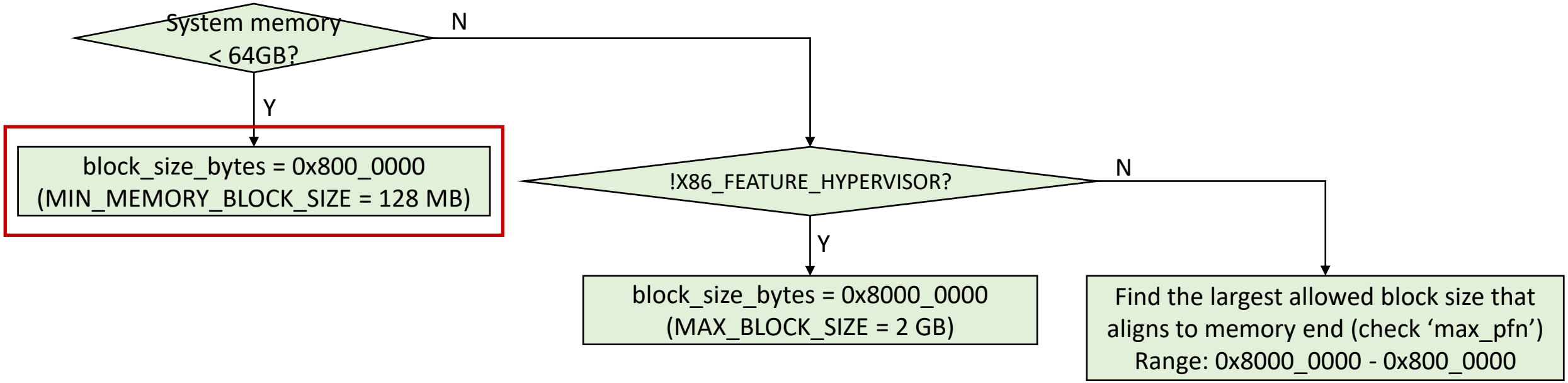
- https://www.kernel.org/doc/html/v5.17/vm/memory-model.html

Backup

/sys/devices/system/memory/block_size_bytes

```
Unit of Memory online/offline operation
---------------------------------------

Memory hotplug uses SPARSEMEM memory model which allows memory to be divided
into chunks of the same size. These chunks are called "sections". The size of
a memory section is architecture dependent. For example, power uses 16MiB, ia64
uses 1GiB.

Memory sections are combined into chunks referred to as "memory blocks". The
size of a memory block is architecture dependent and represents the logical
unit upon which memory online/offline operations are to be performed. The
default size of a memory block is the same as memory section size unless an
architecture specifies otherwise. (see :ref:`memory_hotplug_sysfs_files`.)

To determine the size (in bytes) of a memory block please read this file::

  /sys/devices/system/memory/block_size_bytes

Documentation/admin-guide/mm/memory-hotplug.rst          69,0-1          16%
```

/sys/devices/system/memory/block_size_bytes

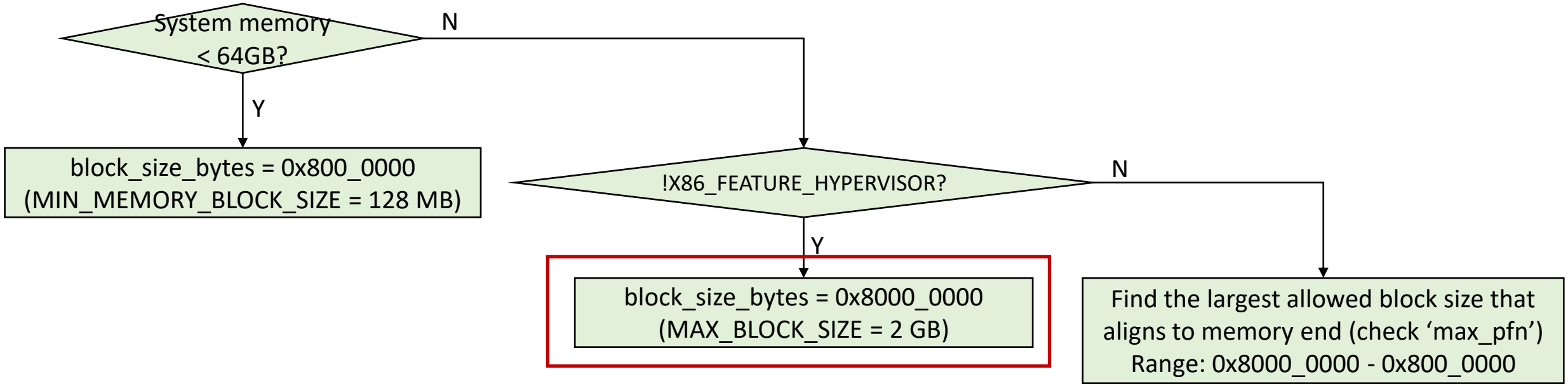



```
[root@rh82 ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:           31Gi       6.5Gi        15Gi       1.0Gi       8.9Gi        23Gi
Swap:         7.8Gi          0B       7.8Gi
[root@rh82 ~]# cat /sys/devices/system/memory/block_size_bytes
8000000
```

* Source code: arch/x86/mm/init_64.c: probe_memory_block_size()

* Ignore SGI UV system platform

/sys/devices/system/memory/block_size_bytes



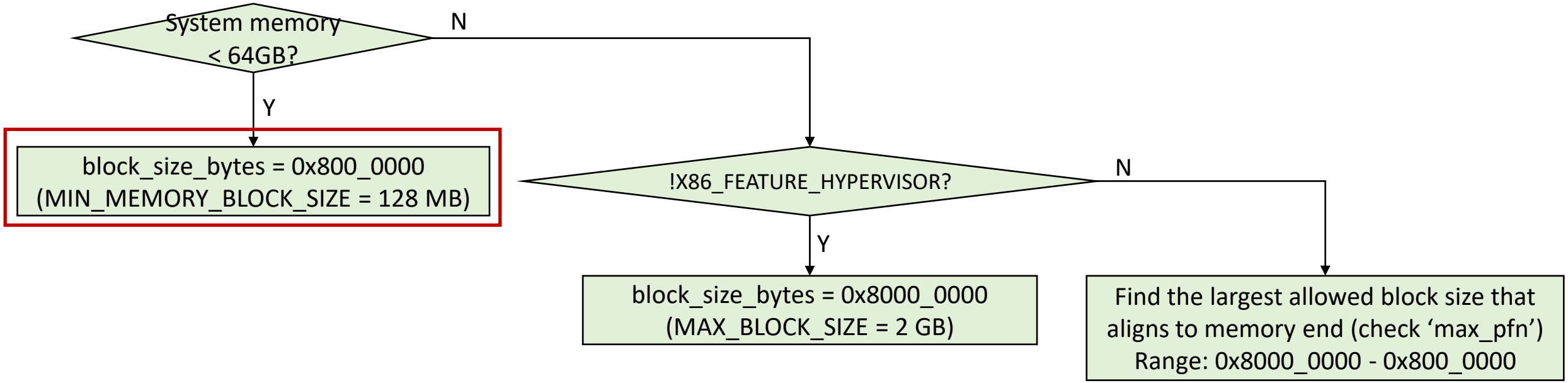

```
adrian@adrian-ubuntu:crash$ free -h
               total        used        free      shared  buff/cache   available
Mem:           125Gi       4.6Gi        20Gi        24Mi        99Gi       119Gi
Swap:          8.0Gi       2.0Mi       8.0Gi
adrian@adrian-ubuntu:crash$ grep hypervisor /proc/cpuinfo
adrian@adrian-ubuntu:crash$ cat /sys/devices/system/memory/block_size_bytes
80000000
```

```
adrian@adrian-ubuntu:~$ dmesg| grep "Memory block"
[    1.586609] x86/mm: Memory block size: 2048MB
```

* Source code: arch/x86/mm/init_64.c: probe_memory_block_size()

* Ignore SGI UV system platform

/sys/devices/system/memory/block_size_bytes



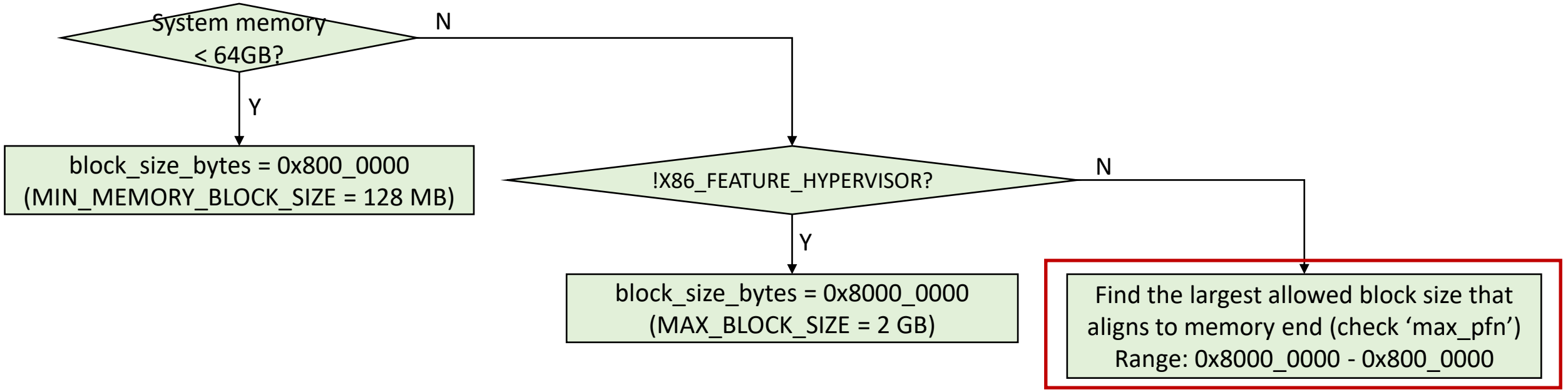
QEMU – Guest OS

```
/ # free -g
              total        used        free      shared  buff/cache   available
Mem:             15           0          15           0           0          15
Swap:             0           0           0
/ # grep -rw hypervisor /proc/cpuinfo  | head -1 | fold -w 80
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm constant_tsc rep_good nopl xtop
ology cpuid pni ssse3 cx16 sse4_1 sse4_2 popcnt hypervisor lahf_lm pti
/ # cat /sys/devices/system/memory/block_size_bytes
8000000
```

* Source code: arch/x86/mm/init_64.c: probe_memory_block_size()

* Ignore SGI UV system platform

/sys/devices/system/memory/block_size_bytes



QEMU – Guest OS

```
/ # free -g
              total        used        free      shared  buff/cache   available
Mem:             62           0          62           0           0          62
Swap:             0           0           0
/ # dmesg | grep pfn
last_pfn = 0x1040000 max_arch_pfn = 0x400000000
last_pfn = 0xbffdf max_arch_pfn = 0x400000000
/ # grep hypervisor /proc/cpuinfo | head -1 | fold -w 80
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx lm constant_tsc rep_good nopl xtop
ology cpuid pni ssse3 cx16 sse4_1 sse4_2 popcnt hypervisor lahf_lm pti
/ # cat /sys/devices/system/memory/block_size_bytes
40000000
```

* Ignore SGI UV system platform

* Source code: arch/x86/mm/init_64.c: probe_memory_block_size()