Ongoing Challenges of Large Page Sizes

Juan Yescas & Kalesh Singh
Google

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Contiguous Memory Allocator (CMA)

```
mynode: mynode {
    compatible = "shared-dma-pool";
    size = <0x02000000>;
}
```

Contiguous Memory Allocator (CMA) - alignment

```
#define pageblock_order     MAX_PAGE_ORDER
#define pageblock_nr_pages (1UL << pageblock_order)

#define CMA_MIN_ALIGNMENT_PAGES pageblock_nr_pages
#define CMA_MIN_ALIGNMENT_BYTES (PAGE_SIZE * CMA_MIN_ALIGNMENT_PAGES)

static int __init rmem_cma_setup(struct reserved_mem *rmem)
 {
    ….

   if (!IS_ALIGNED(rmem->base | rmem->size, CMA_MIN_ALIGNMENT_BYTES)) {
       pr_err("Reserved memory: incorrect alignment of CMA region\n");
       return -EINVAL;
   }
   …
}
```

See https://elixir.bootlin.com/linux/v6.9.4/source/arch/arm64/Kconfig#L1550

Contiguous Memory Allocator (CMA) - default alignment and max alignment

PAGE_SIZE	default MAX_PAGE_ORDER	CMA_MIN_ALIGNMENT_BYTES
4KiB	10	4KiB * 1KiB = **4MiB**
16Kib	11	16KiB * 2KiB = **32MiB**
64KiB	13	64KiB * 8KiB = **512MiB**

If ARCH_FORCE_MAX_ORDER is configured to the max MAX_PAGE_ORDER

PAGE_SIZE	max MAX_PAGE_ORDER	CMA_MIN_ALIGNMENT_BYTES
4KiB	15	4KiB * 32KiB = **128MiB**
16Kib	13	16KiB * 8KiB = **128MiB**
64KiB	13	64KiB * 8KiB = **512MiB**

/proc/locks

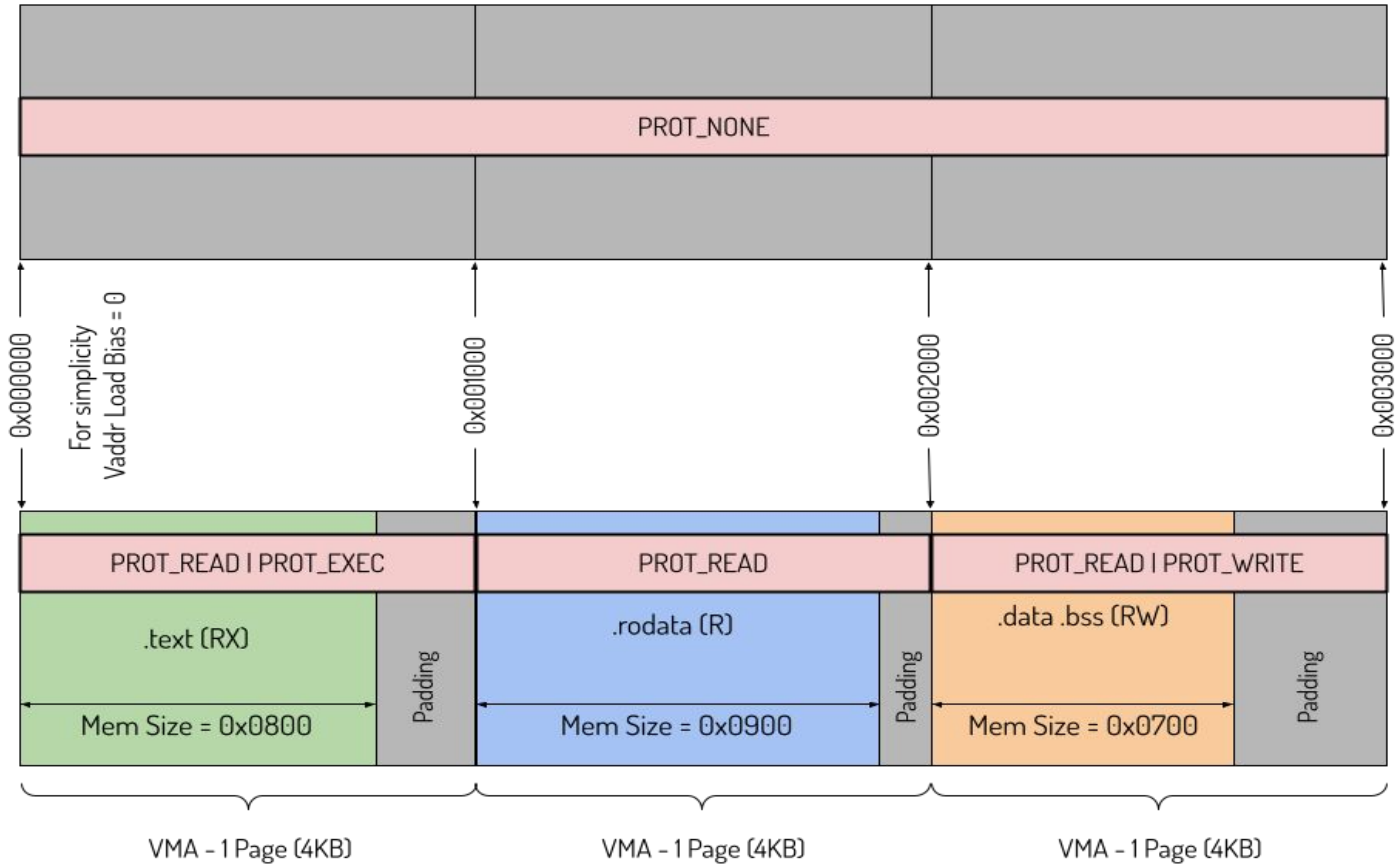
/proc/locks entries

In 16kb kernels, we have observed that the number of entries in /proc/locks increases by 20% to 30% in comparison with 4kb kernels.

```
$ cat /proc/locks
1: POSIX  ADVISORY  WRITE 24570 fe:39:22685 0 EOF
2: POSIX  ADVISORY  READ 27148 fe:39:13802 128 128
3: POSIX  ADVISORY  READ 26662 fe:39:17567 128 128
4: POSIX  ADVISORY  READ 26662 fe:39:16145 1073741826 1073742335
5: POSIX  ADVISORY  READ 26589 fe:39:12434 128 128
6: POSIX  ADVISORY  READ 26589 fe:39:12427 1073741826 1073742335
7: POSIX  ADVISORY  READ 26279 fe:39:11353 128 128
8: POSIX  ADVISORY  READ 26279 fe:39:11340 1073741826 1073742335
9: POSIX  ADVISORY  READ 26078 fe:39:8860 128 128
10: POSIX  ADVISORY  WRITE 24570 fe:39:16588 0 EOF
11: POSIX  ADVISORY  WRITE 24570 fe:39:15318 1073
...
…
…
```

vm_area_struct
Slab Memory Increase

Loading 4KiB ELFs on 4KiB Devices



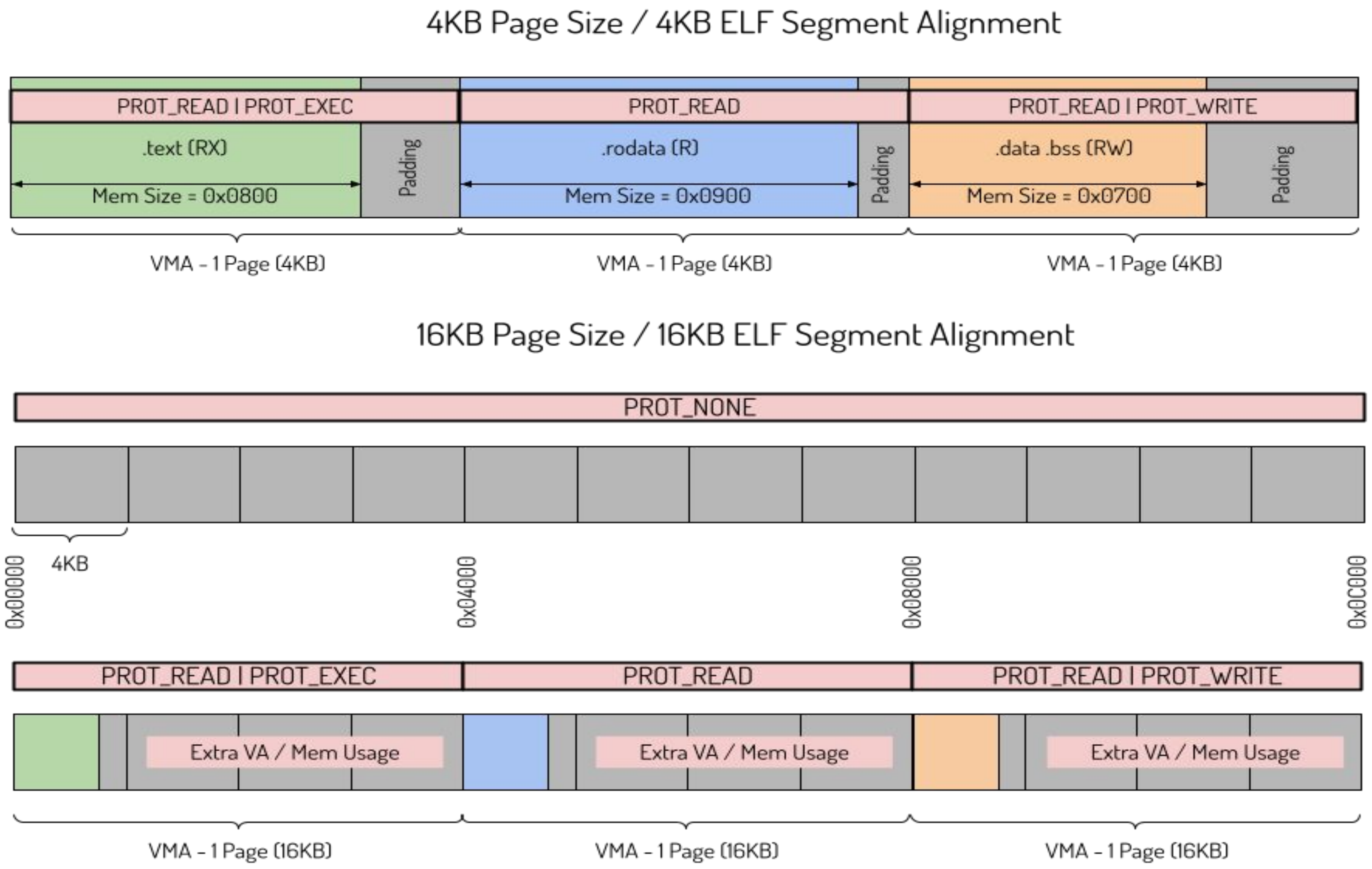
The loader reserves the VA space to load the ELF as PROT non

Then maps in each segment laid out relative to the PROT_NONE mapping start according to the segment vaddr

If the elf is built with -Wl,-z,max-page-size=0x1000

On a 4KiB base-page-size device, the segments are usually all laid out contiguously in the VA space.

Loading 16KiB ELFs on 16KiB Devices

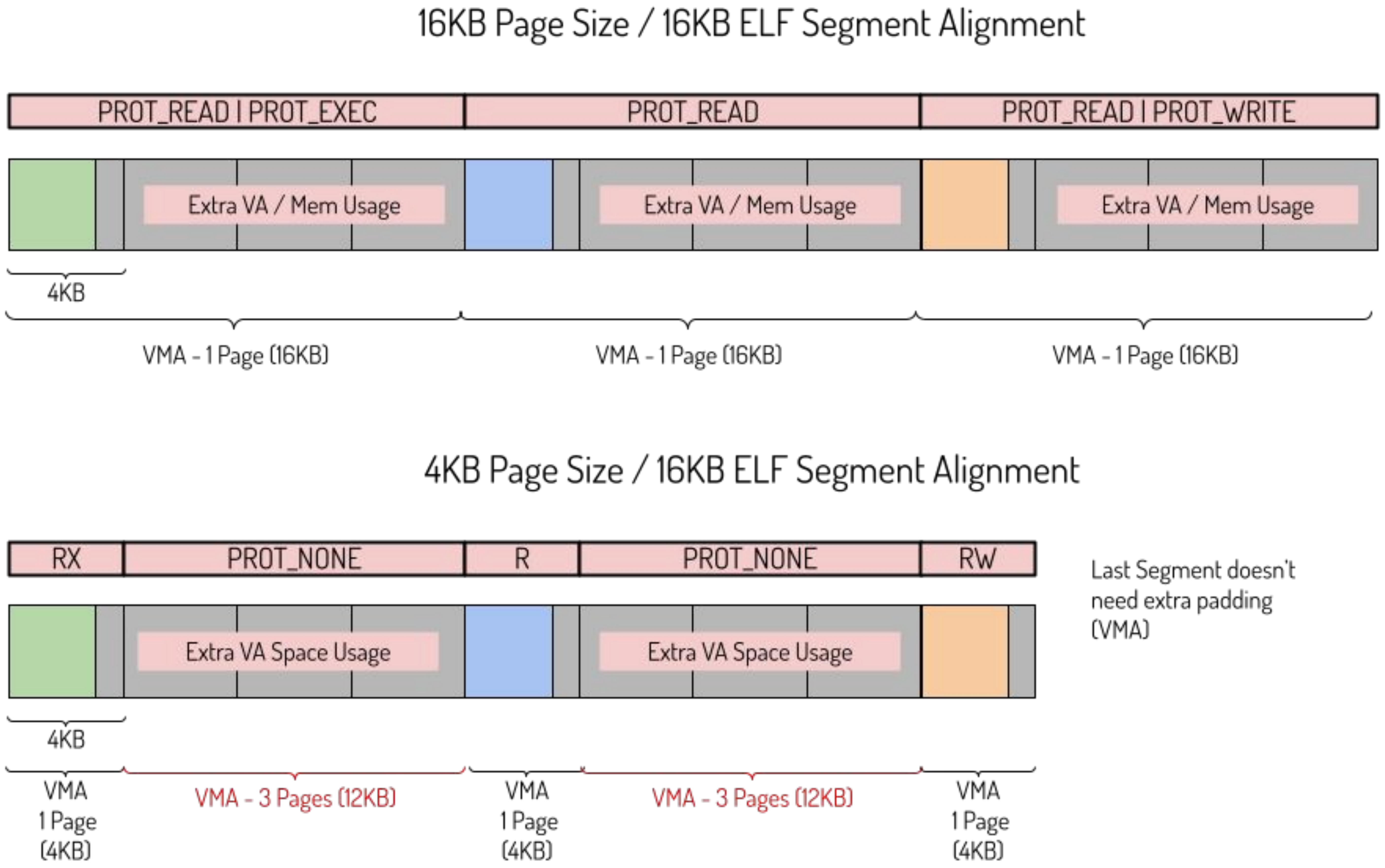


If the elf is built with -Wl,-z,max-page-size=0x4000

On a 16KiB base-page-size device, the segments are usually all laid out contiguously in the VA space.

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Loading 16KiB ELFs on 4KiB Devices


16KB Page Size / 16KB ELF Segment Alignment

4KB Page Size / 16KB ELF Segment Alignment

Last Segment doesn't need extra padding (VMA)

~25-30 MB increase in vm_area_struct slab memory

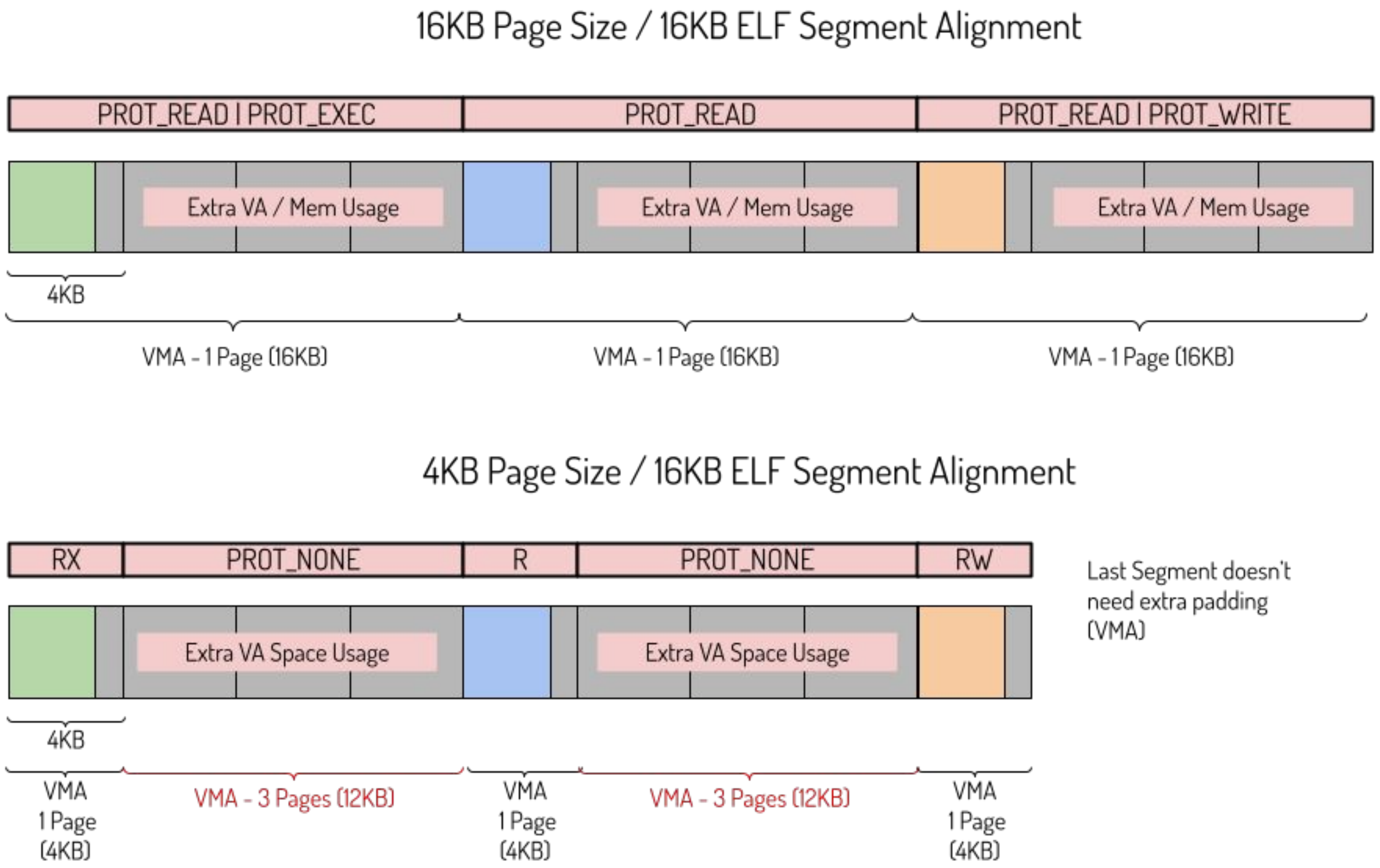
If the elf is built with
-Wl,-z,max-page-size=0x4000

On a 4KiB base-page-size device, the segments discontiguous -- there are PROT_NONE mapping between each consecutive segment due to segment alignment.

This leads to a huge increase in the number of vm_area_structs and a significant increase in VMA slab memory usage.

This is the common case in Android.

Loading 16KiB ELFs on 4KiB Devices


16KB Page Size / 16KB ELF Segment Alignment

4KB Page Size / 16KB ELF Segment Alignment

Last Segment doesn't need extra padding (VMA)

~25-30 MB increase in vm_area_struct slab memory

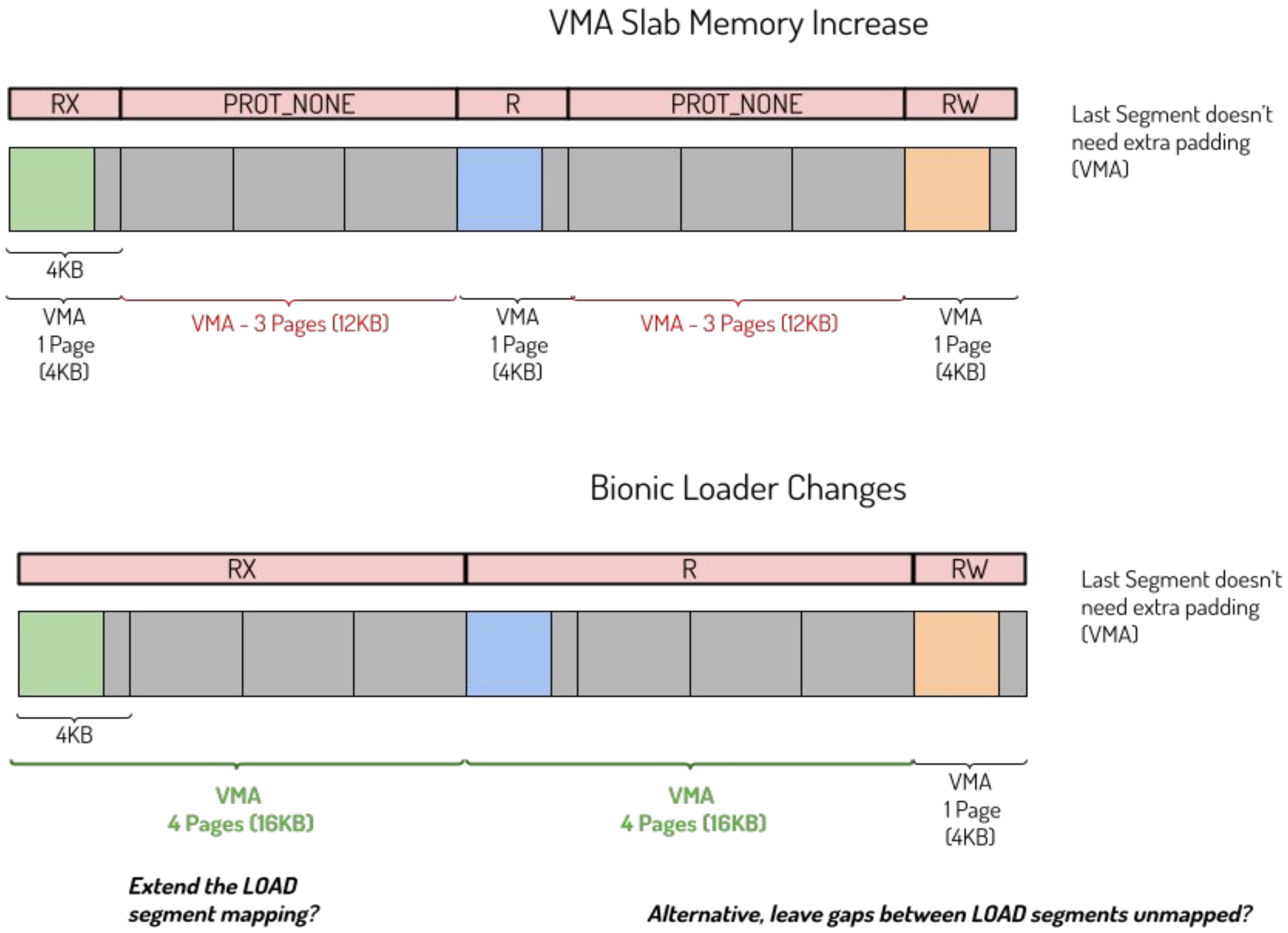
If the elf is built with
-Wl,-z,max-page-size=0x4000

On a 4KiB base-page-size device, the segments discontiguous -- there are PROT_NONE mapping between each consecutive segment due to segment alignment.

This leads to a huge increase in the number of vm_area_structs and a significant increase in VMA slab memory usage.

This is the common case in Android.

Loading 16KiB ELFs on 4KiB Devices



Option:

1. Unmap the “gap” PROT_NONE VMAs
2. Extend the segment VMA to cover the “gap”

Android extends the VMA to prevent unrelated mapping between the ELF segments.

Page Cache Read Ahead
and
ELF alignment

Page Cache Read Ahead and Reads

When the shared libraries and executables are compiled with:

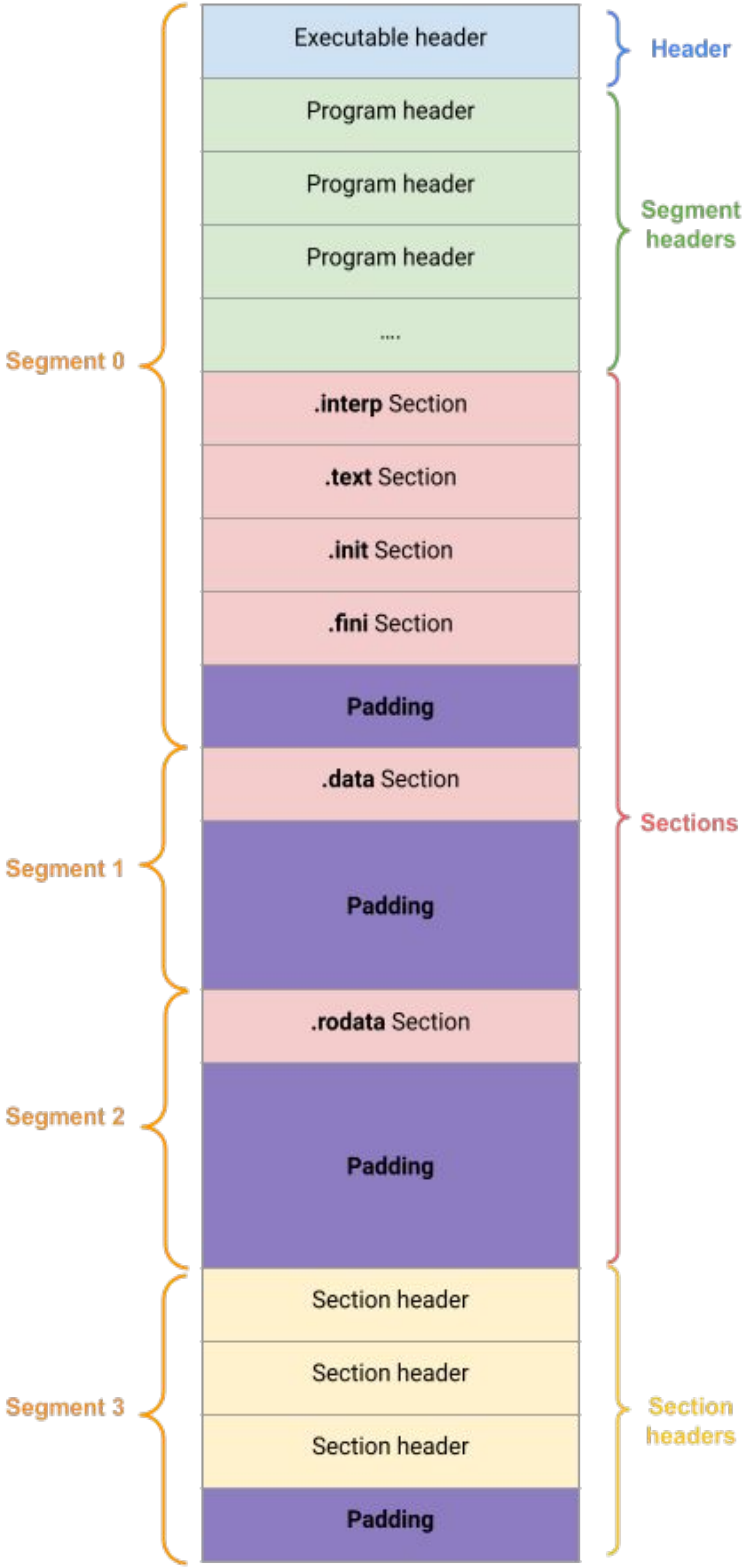
-Wl,-z,separate-loadable-segments

An extra padding is added between the segments and this padding is a multiple of

-Wl,-z,max-page-size=<value here>

The area in violet represents the the extra padding added.

This could increase the file size and has performance penalties due the **page cache readahead** has to issue reads to the block device for the zero blocks.



Filesystem Fault Around
And
Userspace Memory Accounting

Page Cache Read Ahead and Fault Around

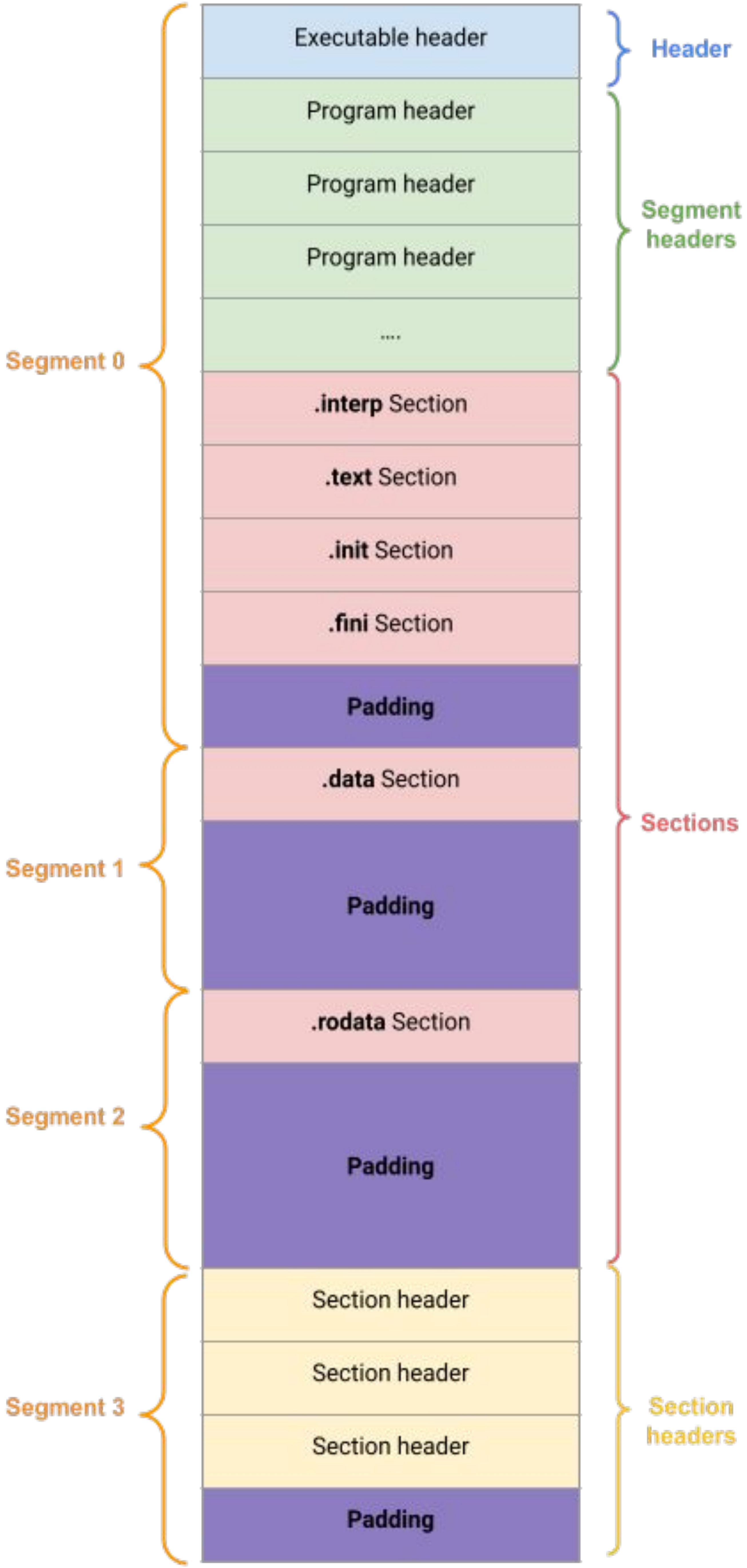
File Systems that implement fault around populate the PTEs for the pages in the page cache for the faulting VMA

This lead to userspace processes perceiving an increase in RSS due to the pages brought in by read ahead.

Application developers monitor RSS metrics.

Limit the fault around to exclude padding range for ELF segments VMAs

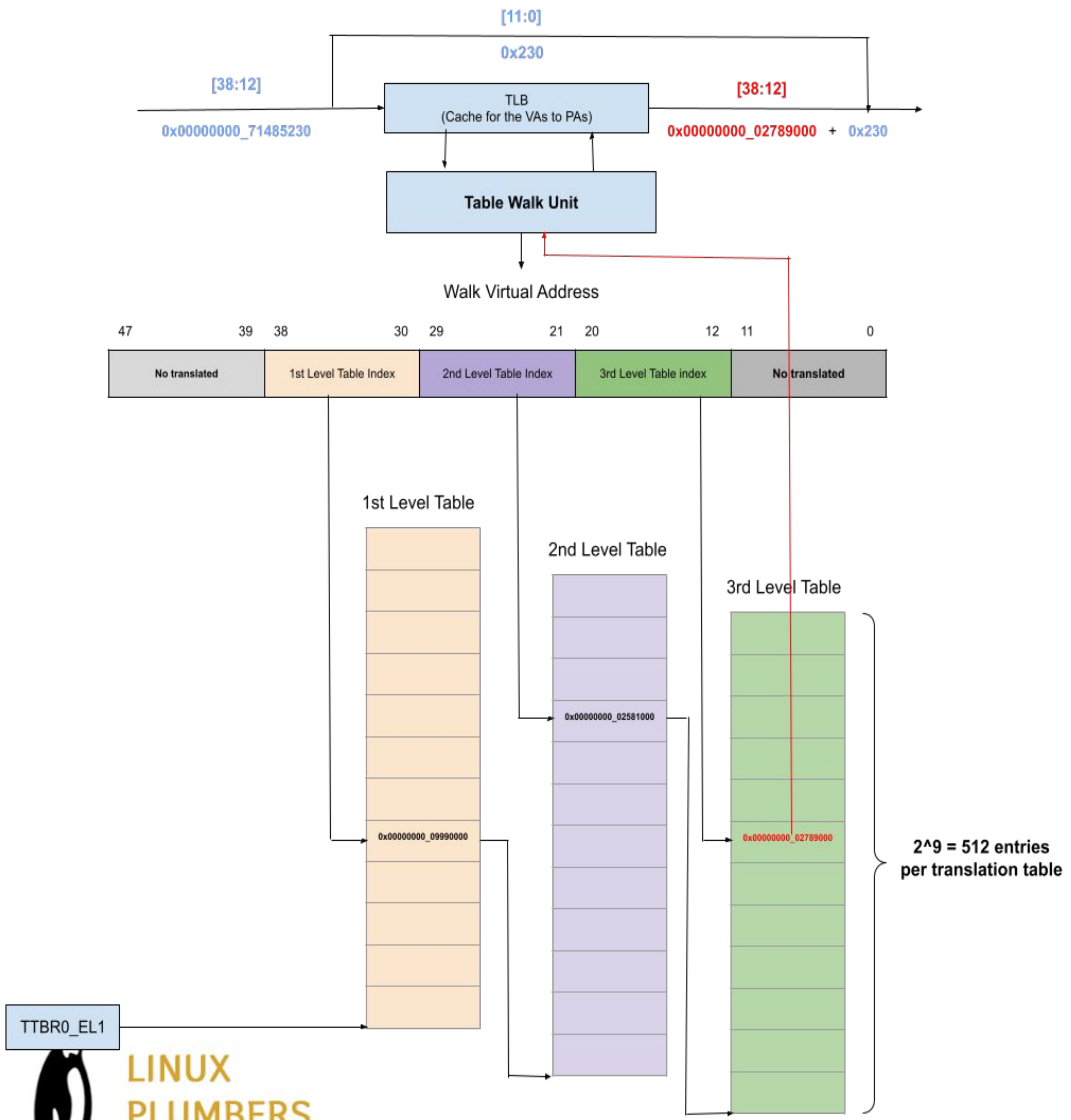
Ideally readahead wouldn't bring in these pages to the page cache.



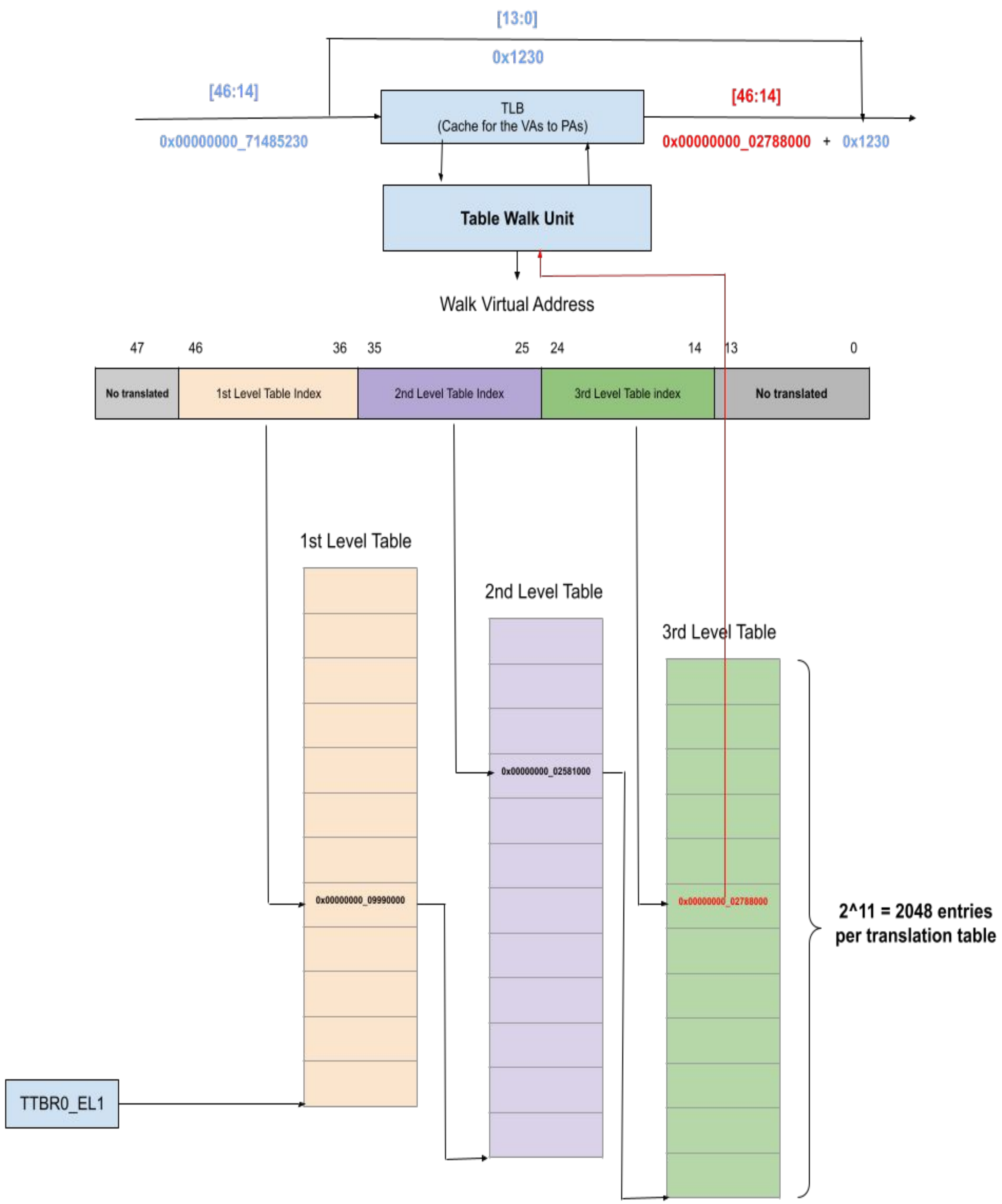
Discussion

Appendix

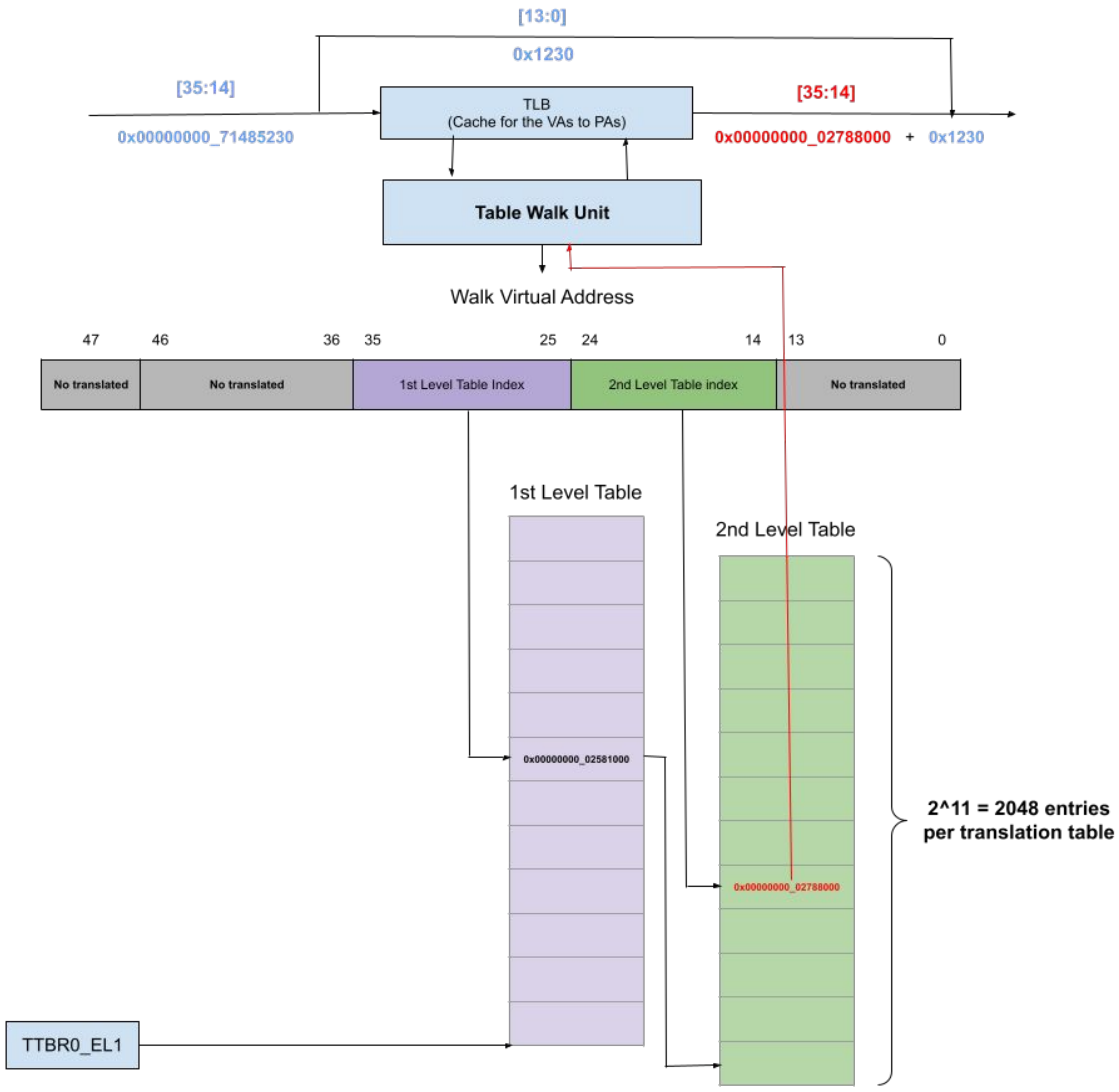
Page Table Walks - 4k Granule - 39-bits VA



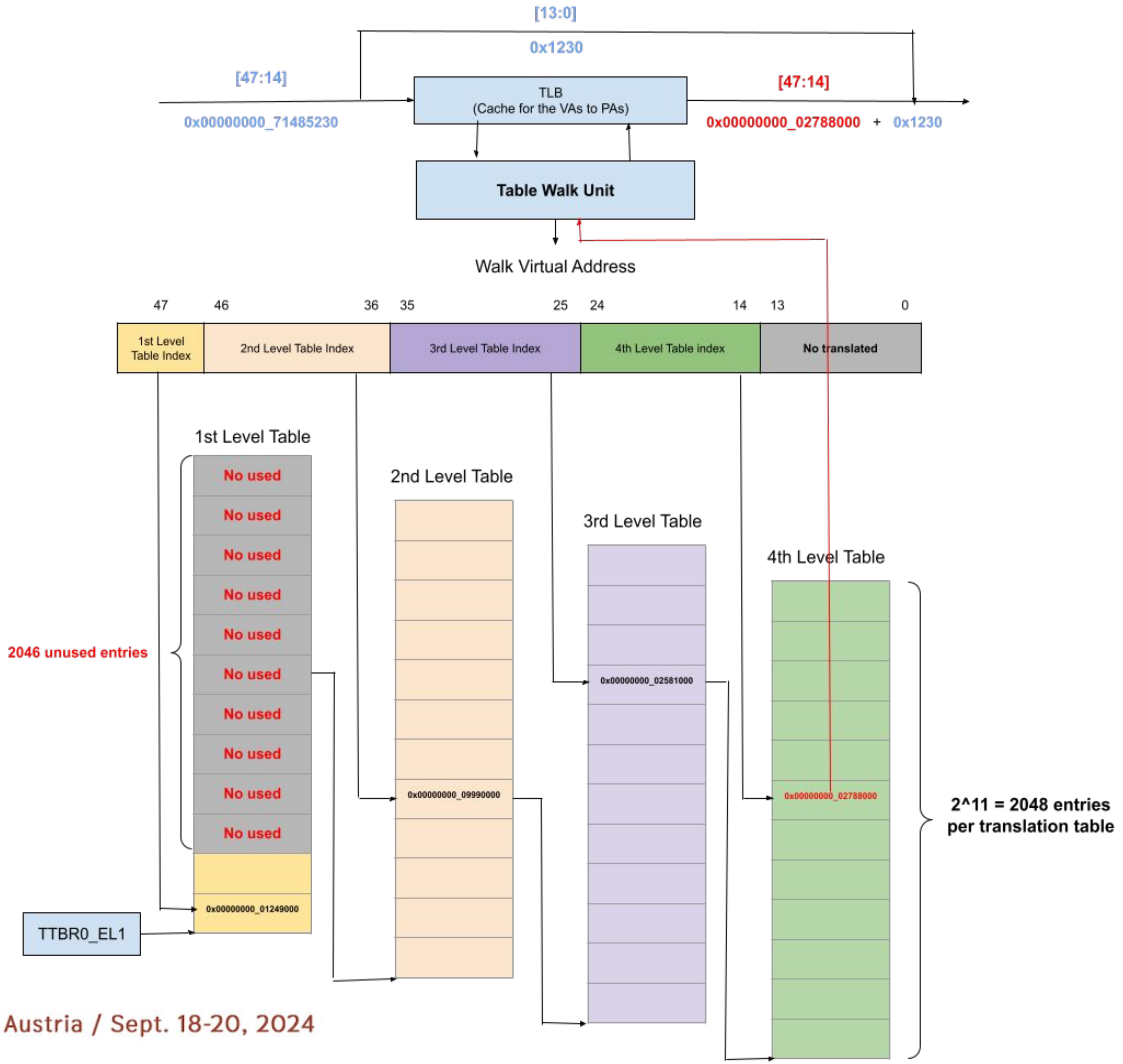
Page Table Walks - 16k Granule - 47-bits VA



Page Table Walks - 16k Granule - 36-bits VA



Page Table Walks - 16k Granule - 48-bits VA


[13:0]
0x1230
[47:14]
0x00000000_71485230
TLB
(Cache for the VAs to PAs)
[47:14]
0x00000000_02788000 + 0x1230
Table Walk Unit
Walk Virtual Address
47
46
36
35
25
24
14
13
0
1st Level Table Index
2nd Level Table Index
3rd Level Table Index
4th Level Table index
No translated
1st Level Table
No used
No used
No used
No used
No used
No used
No used
No used
No used
No used
0x00000000_01249000
2046 unused entries
TTBR0_EL1
2nd Level Table
0x00000000_09990000
3rd Level Table
0x00000000_02581000
4th Level Table
0x00000000_02788000
2^11 = 2048 entries
per translation table