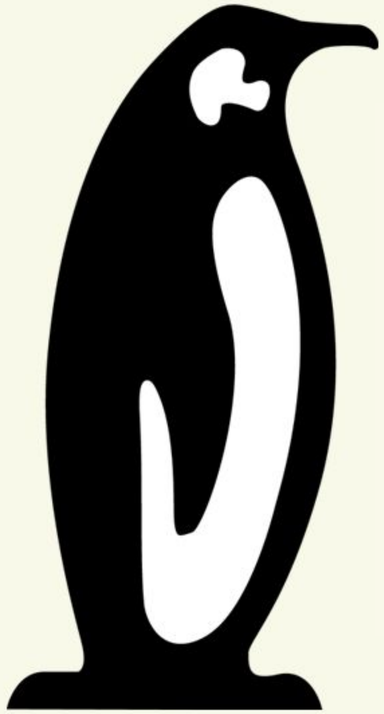


# Android - 16K Page Size Support

## LPC 2023

Richmond, Virginia | November 13-15, 2023

**Kalesh Singh**
Google

**Juan Yescas**
Google

# Why 16k page sizes?

## Performance Benchmarks on Pixel 6 and Pixel 6 Pro showed

- **4x reduction** in page faults
- Faster boot time (0.8 seconds faster)
- Faster app launch time (~3.16%)
  - **~17% for Google Search**
  - **~30% for Google News**
- Power consumption of the phone was reduced by **4.56%** on average
- Several other Industry standard benchmarks such as Geekbench, GFXbench, Speedometer, etc showed between 2%-10% perf improvements.
- Other device vendors have seen similar perf gains

## Trade Offs

Increase in **memory usage** due:

- ELF Segments are 16k and cause ELF fragmentation (2.19% for 4k page size vs 9.57% for 16k page size)

Minimal increase in **disk space**

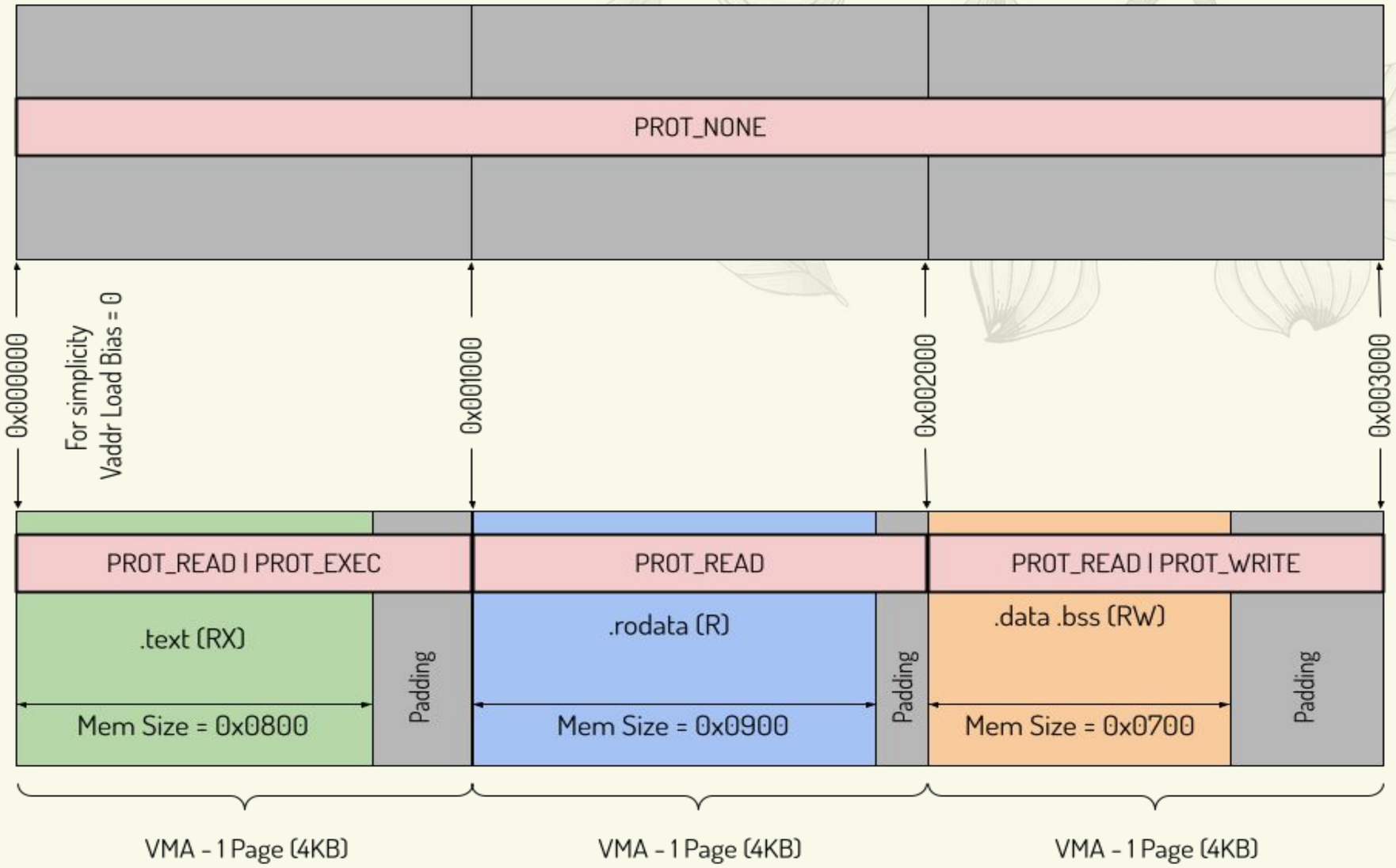
- minimal increase in disk size in F2FS and EXT4 filesystem - 0.03%

# ELF Loading

Simplified ELF file (Sections omitted for simplicity)

```
Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  LOAD           0x000000 0x0000000000000000 0x0000000000000000 0x000300 0x000800 R E 0x1000
  LOAD           0x001000 0x0000000000001000 0x0000000000001000 0x000500 0x000900 R   0x1000
  LOAD           0x002000 0x0000000000002000 0x0000000000002000 0x000500 0x000700 RW  0x1000

 Section to Segment mapping:
  Segment Sections...
   00     .text
   01     .rodata
   02     .data .bss

Section Headers:
  [Nr] Name              Type             Address           Off    Size   ES Flg Lk Inf Al
  [ 0] .text             PROGBITS         0000000000000000 000000 000800 00  AX  0   0 16
  [ 1] .rodata           PROGBITS         0000000000001000 001000 000900 00   A  0   0 32
  [ 2] .data             PROGBITS         0000000000002000 002000 000500 00  WA  0   0 32
  [ 3] .bss              NOBITS           0000000000002100 002500 000200 00  WA  0   0 32

File Size: 0x002500
```

# ELF Loading


PROT_NONE
0x000000
For simplicity
Vaddr Load Bias = 0
0x001000
0x002000
0x003000
PROT_READ | PROT_EXEC
PROT_READ
PROT_READ | PROT_WRITE
.text (RX)
Mem Size = 0x0800
Padding
.rodata (R)
Mem Size = 0x0900
Padding
.data .bss (RW)
Mem Size = 0x0700
Padding
VMA - 1 Page (4KB)
VMA - 1 Page (4KB)
VMA - 1 Page (4KB)

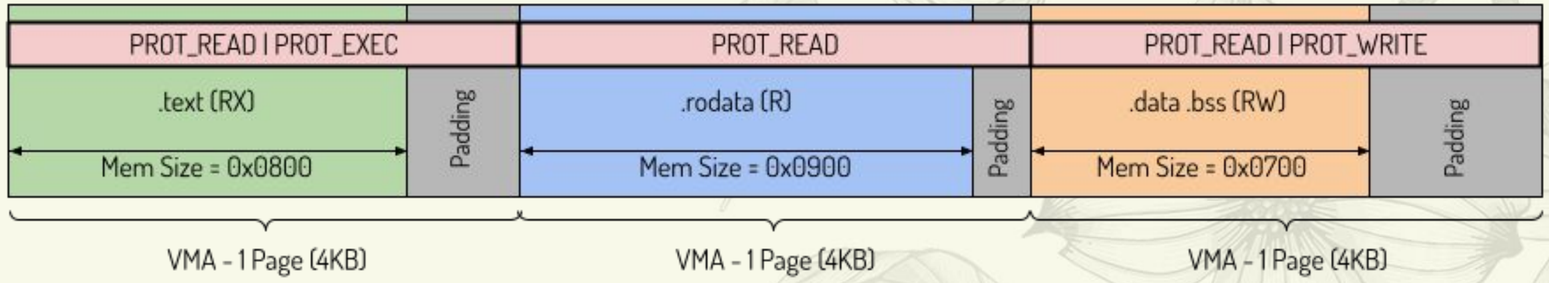
# ELF Alignment (16K Page Size)

Simplified ELF file (Sections omitted for simplicity)

```
Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  LOAD           0x000000 0x0000000000000000 0x0000000000000000 0x000300 0x000800 R E 0x4000
  LOAD           0x004000 0x0000000000004000 0x0000000000004000 0x000500 0x000900 R   0x4000
  LOAD           0x008000 0x0000000000008000 0x0000000000008000 0x000500 0x000700 RW  0x4000

 Section to Segment mapping:
  Segment Sections...
   00     .text
   01     .rodata
   02     .data .bss

Section Headers:
  [Nr] Name              Type             Address           Off    Size   ES Flg Lk Inf Al
  [ 0] .text             PROGBITS         0000000000000000 000000 000800 00  AX  0   0 16
  [ 1] .rodata           PROGBITS         0000000000004000 004000 000900 00   A  0   0 32
  [ 2] .data             PROGBITS         0000000000008000 008000 000500 00  WA  0   0 32
  [ 3] .bss              NOBITS           0000000000008500 008500 000200 00  WA  0   0 32

File Size: 0x008500
```

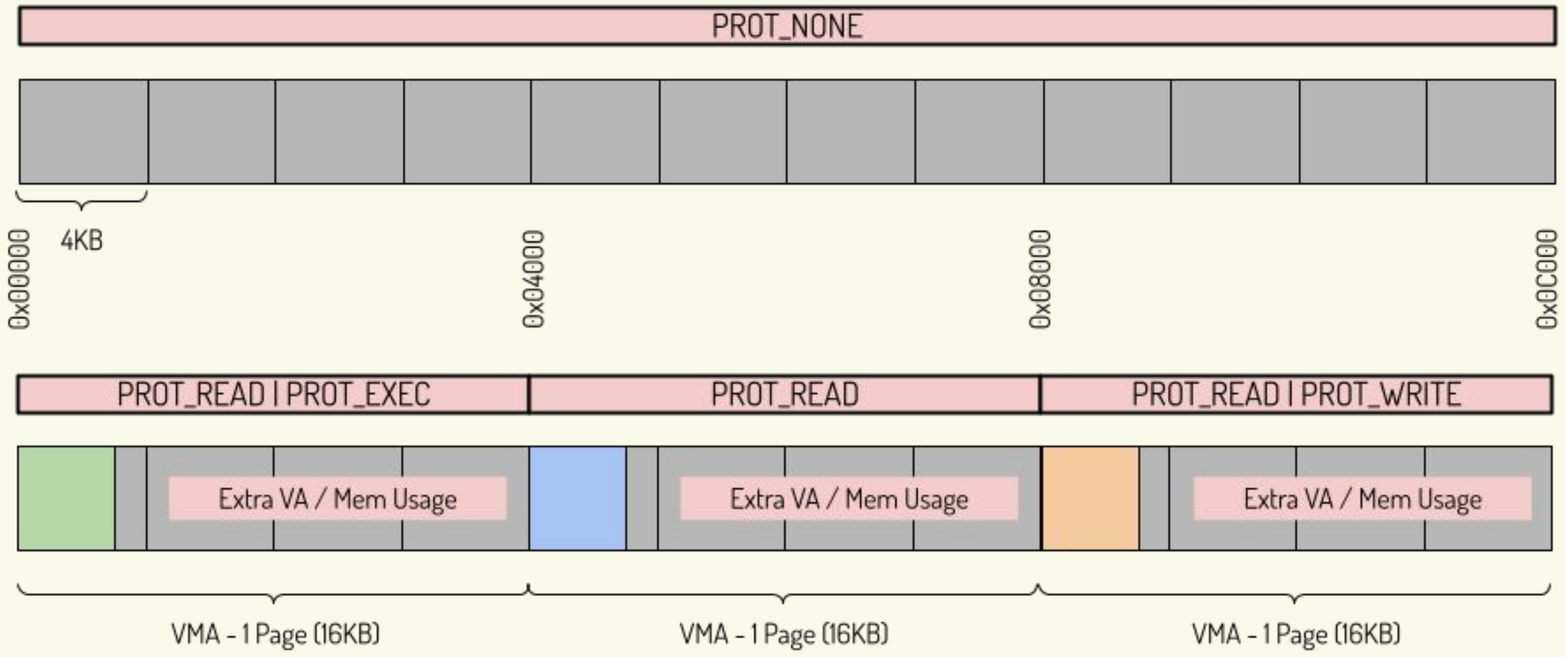
Android set LLVM max-page-size default to 4096

## 4KB Page Size / 4KB ELF Segment Alignment



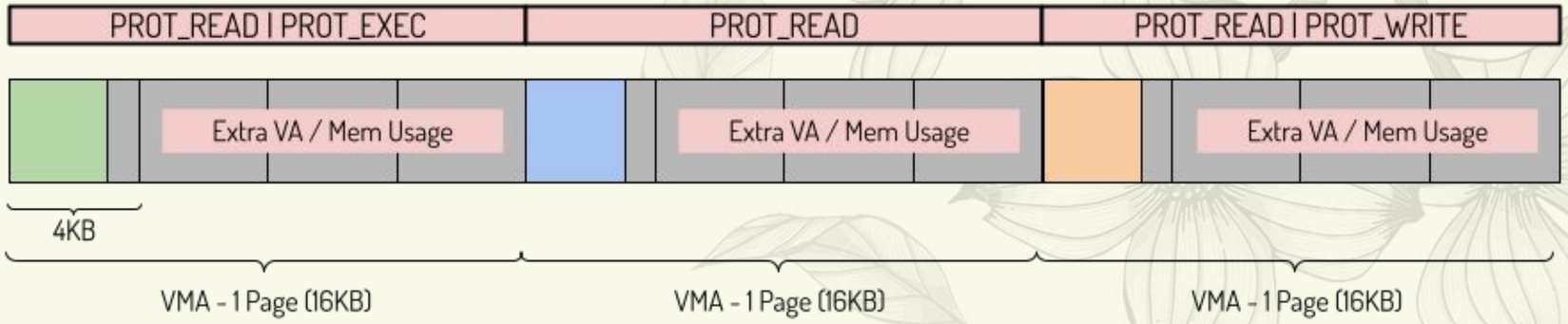

## 16KB Page Size / 16KB ELF Segment Alignment

**-z, max-page-size=16384**

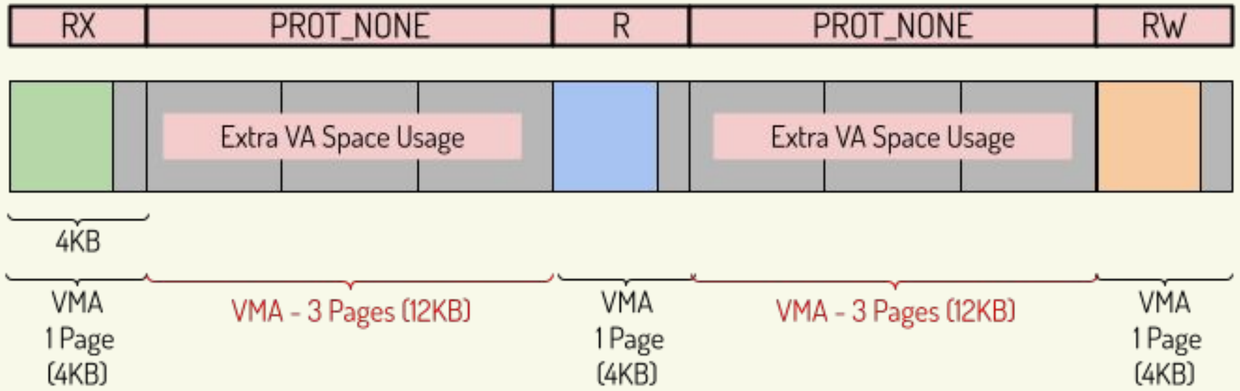


# VMA Slab Memory Increase

## 16KB Page Size / 16KB ELF Segment Alignment




## 4KB Page Size / 16KB ELF Segment Alignment

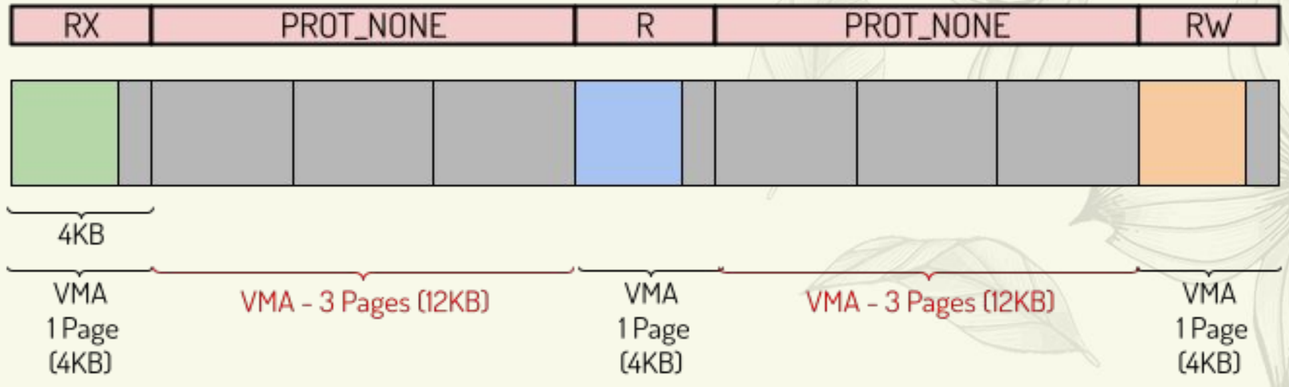



Last Segment doesn't need extra padding (VMA)

~25-30 MB increase in vm_area_struct slab memory

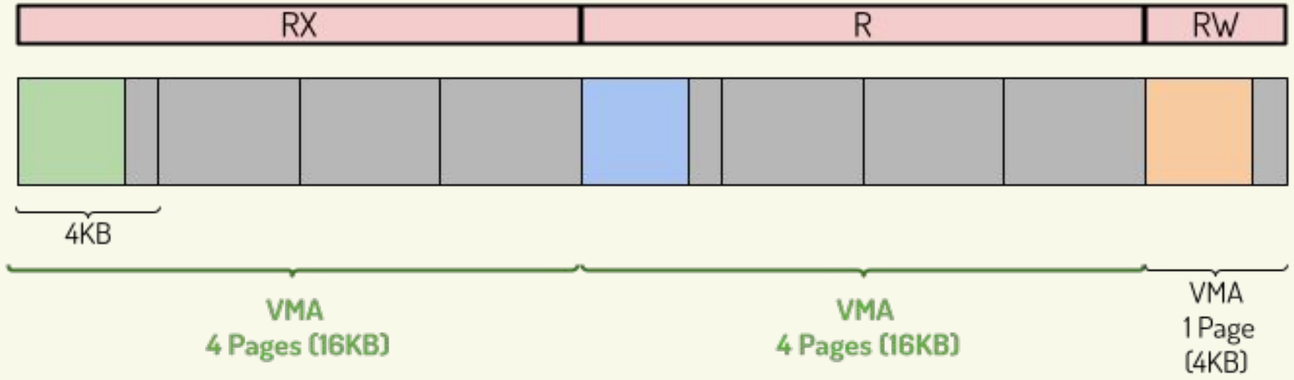
# Bionic Loader Updates

## VMA Slab Memory Increase




Last Segment doesn't need extra padding (VMA)

## Bionic Loader Changes




Last Segment doesn't need extra padding (VMA)

***Extend the LOAD segment mapping?***

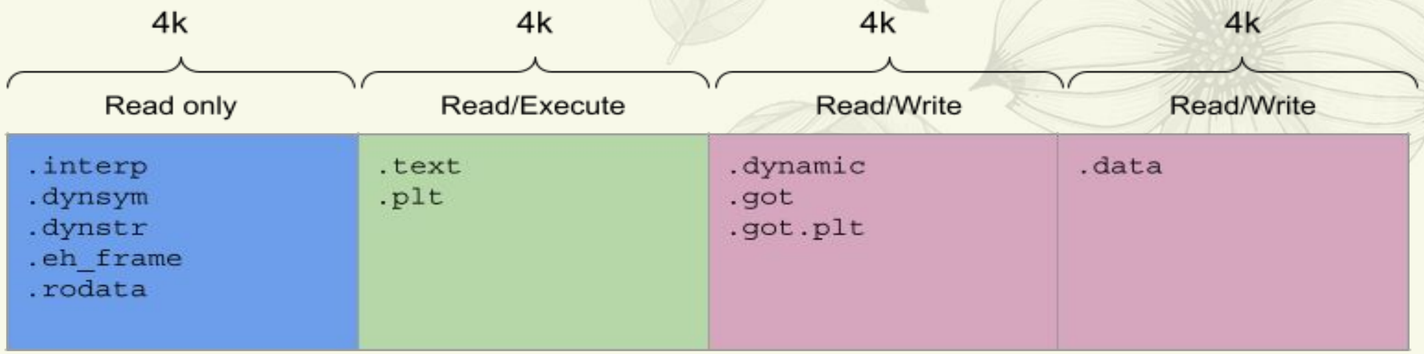
***Alternative, leave gaps between LOAD segments unmapped?***

# Compatibility Solutions?

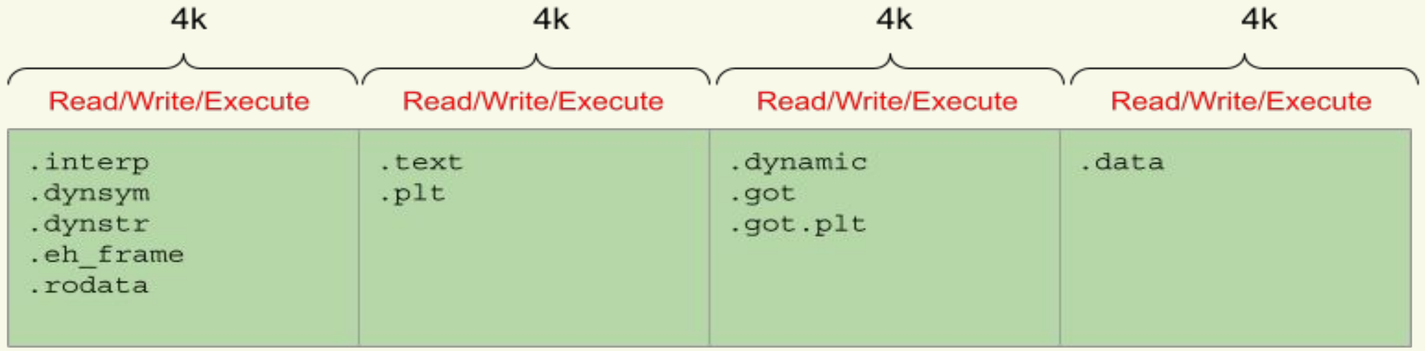
( 4k binaries on 16k page size kernel)

# Map all the segments as **RWX**

**4k elf aligned binary**




**Memory Map every ELF segment with Read/Write/Execute permissions**



# Realign the ELF files to 16k

**What could change when this -Wl,-z,max-page-size=16384 linker flag is used?**

- Program headers (struct Elf64_Phdr)
- Sections that contain code (.text, .init, etc)

**Tool to compare elf64 files struct field by struct field** system/memory/libmeminfo/+/2624789

**And we realized**

- Section .dynsym
- Section .rela.dyn
- Section .rela.plt
- Section .plt
- Section .dynamic
- Section .got
- Section .data

**We tried**

- Linear disassembly
- Recursive disassembly

# Drivers Issues

# One Particular UFS host controller

## Symptom

- Partitions couldn't be found during booting

## Causes

- The UFS Host Controller used by the device didn't follow the Host Controller Interface (HCI).
- The UFS Host controller uses segments smaller than the page size, which it is not supported in Linux.

## Solution

- Add support in Linux to handle segments smaller than the page size.
- See Bart Van Assche's patches [PATCH v6 0/8] Support limits below the page size

# Trusty (TEE OS)

**Shared Memory Size and Alignment**

- The transfer of information between Linux and Trusty involves the setting up of shared memory buffers.
- Importantly the translation regimes (linux kernel, el2 hypervisor, and trusty) involved can all have different translation granules.
- If X is the larger translation granule size, then the size of the memory region must be a multiple of X.
- The base address of the memory region must be aligned to X. Arm Firmware Framework for Arm A-profile - 4.6 Memory granularity and alignment

**Memory Sizes Expressed as Page Counts**

- Arm Firmware Framework for Arm A-profile expresses the size of memory regions as counts of 4K pages.
- The trusty driver updated to manage buffer sizes using 4K granule counts instead of PAGE_SIZE granule; since the kernel PAGE_SIZE can now be greater than 4K.

**FFA_PAGE_COUNT = KERNEL_PAGE_COUNT x (KERNEL_PAGE_SIZE / FFA_PAGE_SIZE)**

**FFA_PAGE_SIZE = 4KB**

**KERNEL_PAGE_SIZE = [4KB | 16KB | 64KB]**

# Emulating 16KB Page Size on x86

**Why?**

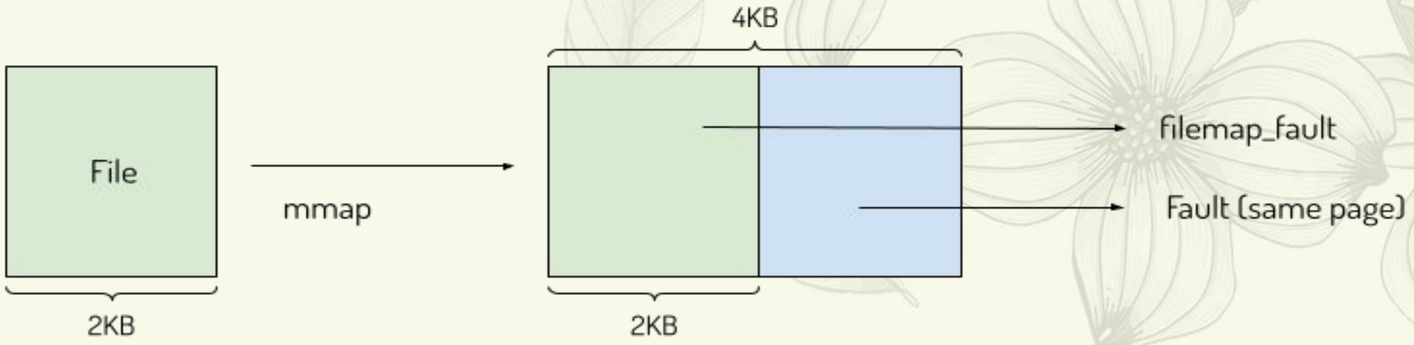
- Majority of Android app developers develop  on x86 (Windows)
- ARM64 Android emulator on x86 is very slow (impractical)
- Need to provide testing platform for x86 developers

**How?**

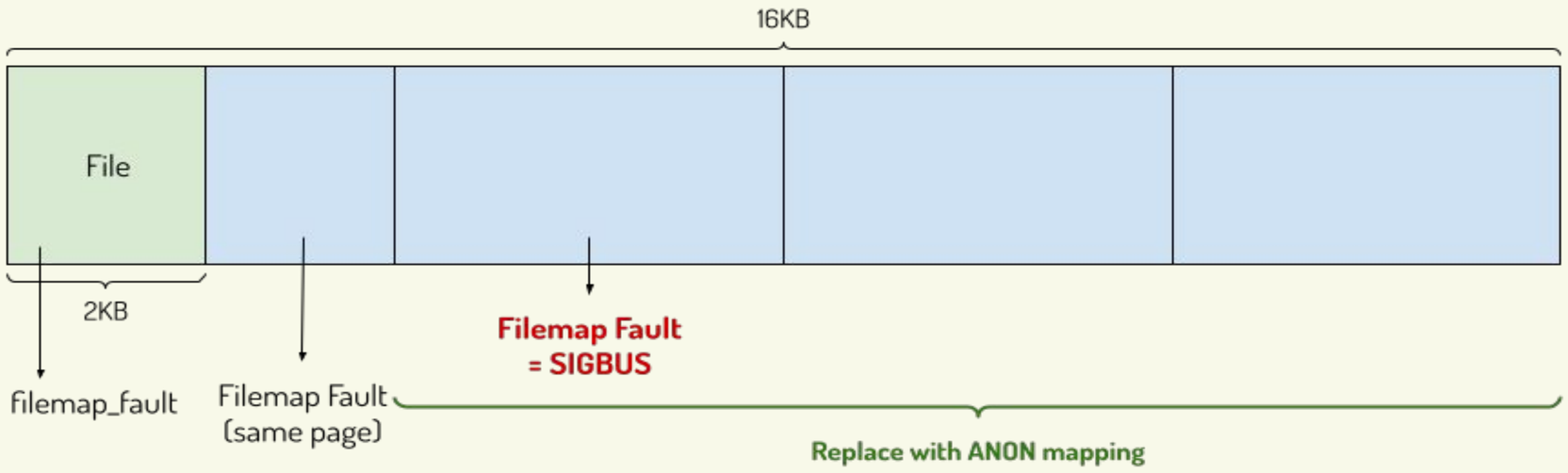
- Kernel presents a 16KB page size to userspace
- Only allow mmap/mprotect (and friends) to operate on 16KB aligned addresses and 16KB multiple sizes.
- VMAs are always 16K aligned and 16K multiple sized.

# Filemap Fault Handling with Emulated 16KB Page Size

## Filemap Fault Handling (4KB)




## Filemap Fault Handling (Emulated 16KB Page Size)



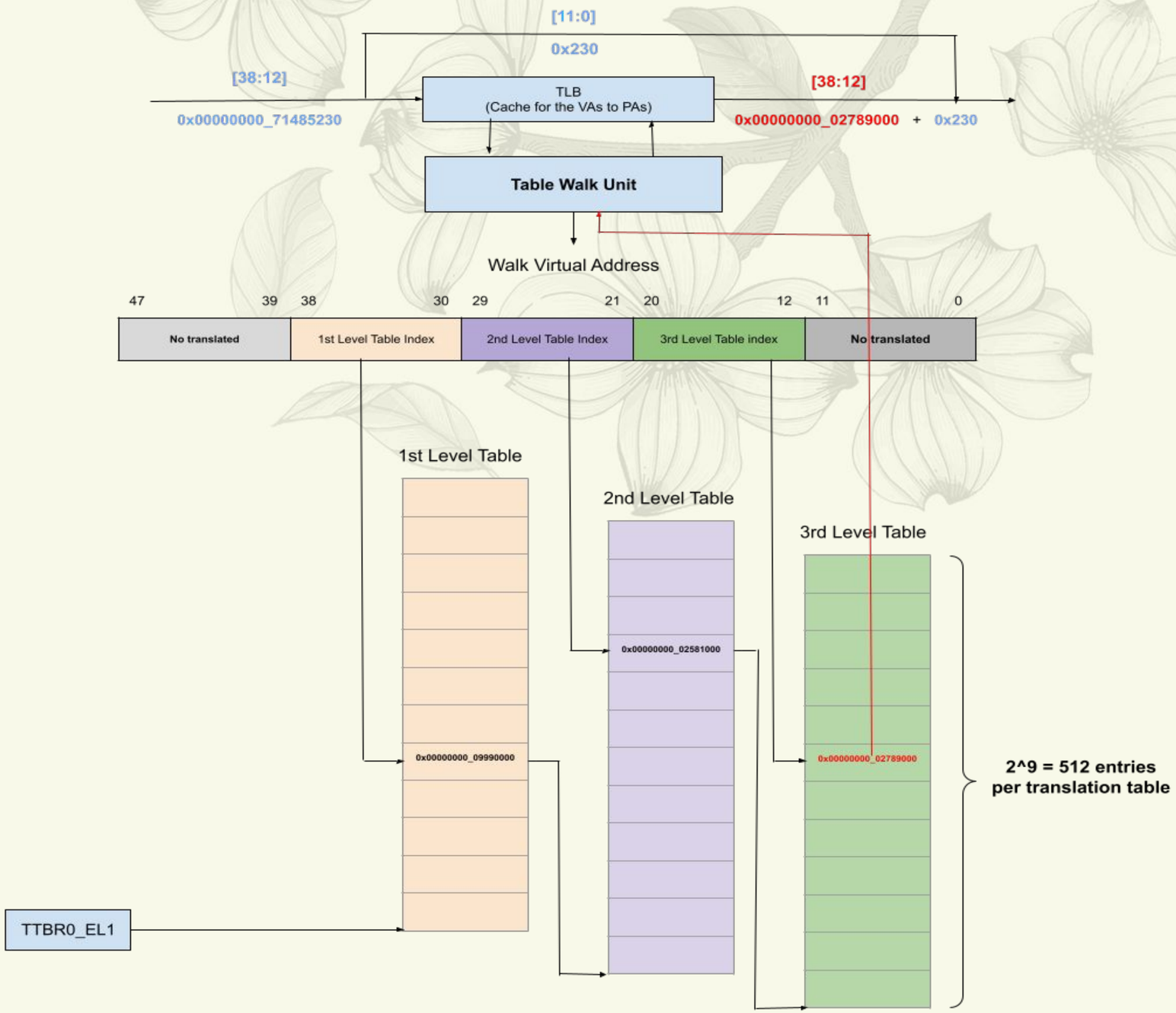
# Page Table Walks

# and

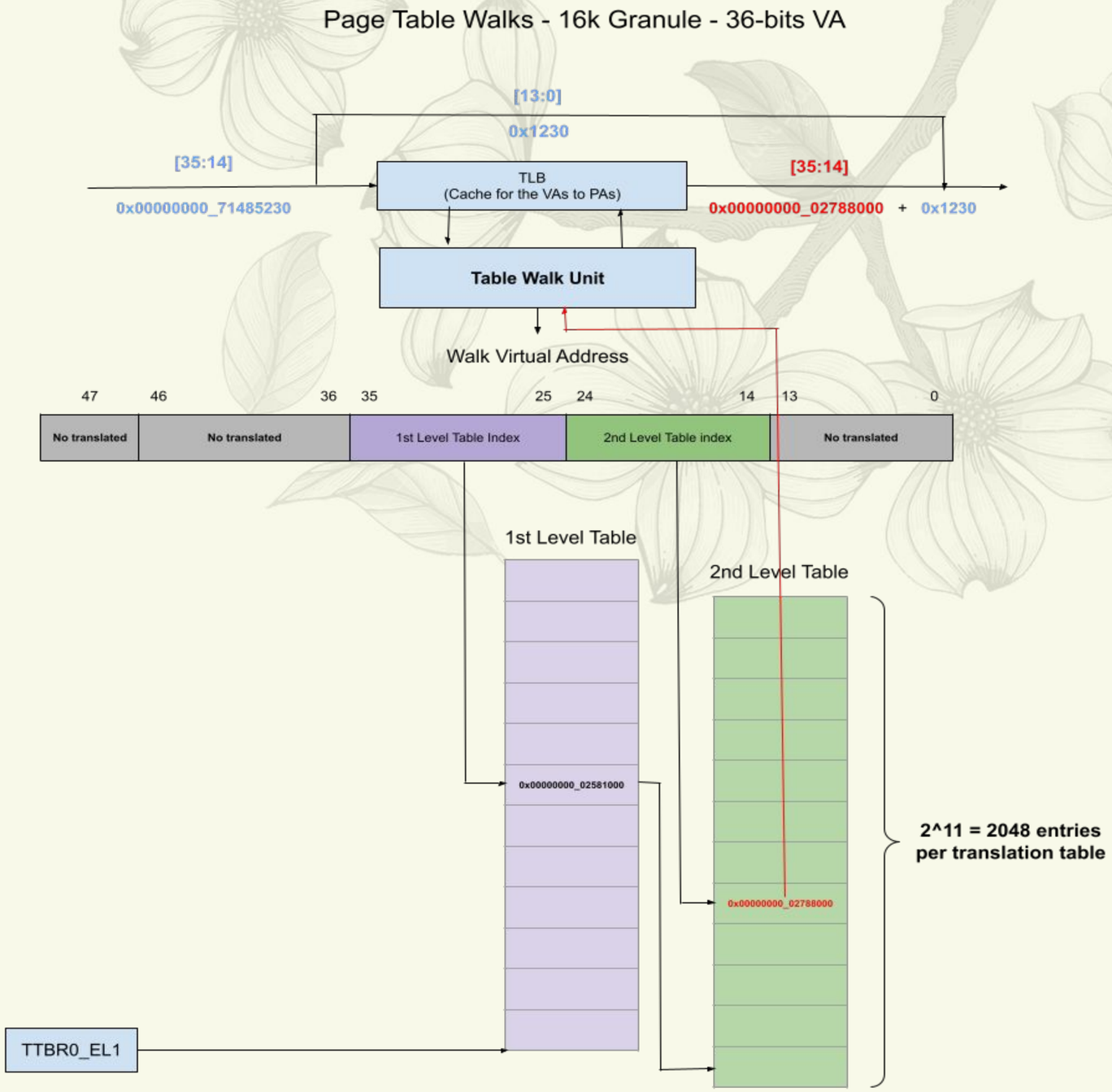
# Virtual Address Issues

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

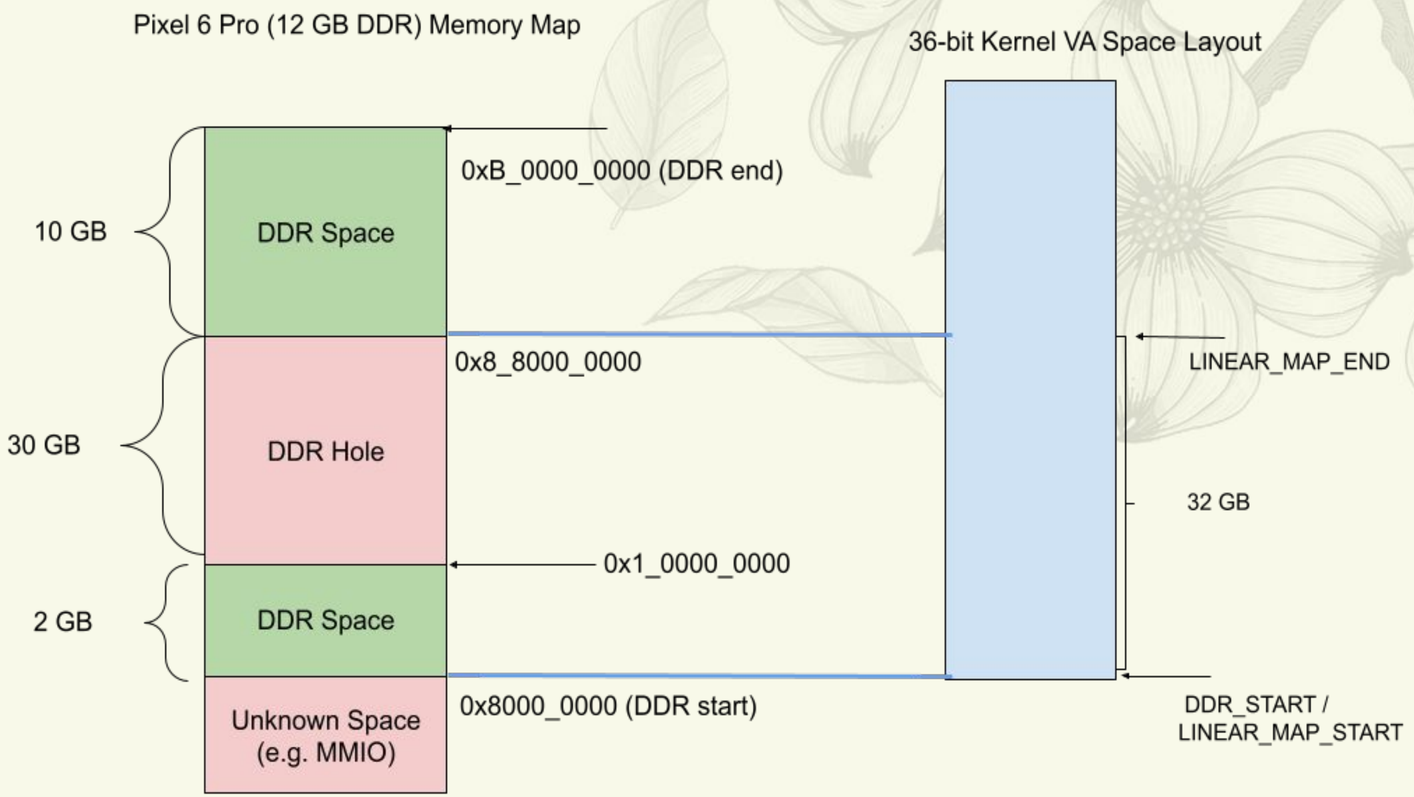
4k page size
and
39-bits VA



16 page size
and
36-bits VA



# 36-bits VA and 30 GB hole




Credits: Isaac Manjarres and William McVicker

Principles of ARM Memory Maps (ARM to publish updated documentation)

# *Early* Comparisons with Folios

| | 4k pages + Folios | 16k Pages |
|---|---|---|
| **Geekbench** | ~6.0% | ~9.0% |
| **Speedometer** | ~4.0% | ~7.0% |

Credits: Ryan Roberts (ARM)

# Questions?

- How do we engage with hardware vendor providers so they follow standards such a HCI?
- When hardware components don't implement the standards and there are a lot of devices using the component, Should the Linux Kernel add support for this hardware component?
- Does maintainer's view of waiting for better hardware for upstream (rather than adding lots of complexity for early "broken" controllers) a reasonable approach?
- Suggestions to achieve backwards compatibility of 4k binaries in 16k page size kernels?