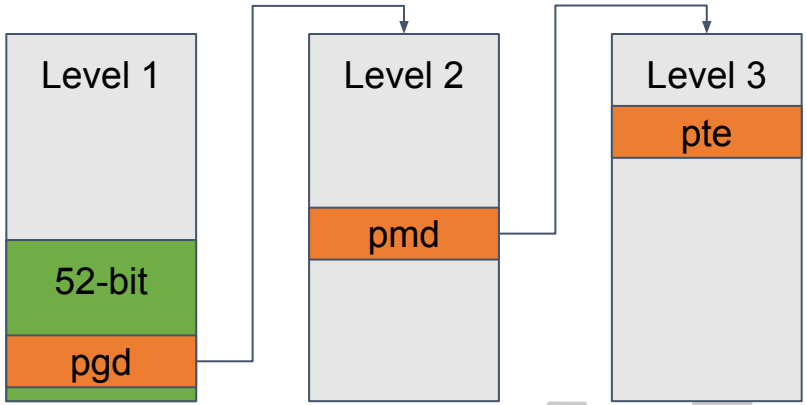
YVR18-119 - 52 shades of VA

Steve Capper <steve.capper@arm.com>

Architectural support for 52-bit VA

- The optional feature, ARMv8.2-LVA, introduces 52-bit virtual addresses,
- Of particular note is that one needs to be running with a 64KB granule (or PAGE_SIZE) to use this feature,
- 52 bit VAs require 3-levels of page table (just like the 48-bit case).



Virtual addresses in Linux

Kernel VAs

- One may wish to employ 52-bit VAs to address a large direct linear map,
- (we already have 52-bit physical address support),
- Unfortunately VA_BITS is a compile time option; making it hard to enable 52-bit dynamically.

Userspace VAs

- Applications may wish to allocate larger ranges with “gaps” or indeed use large amounts of memory,
- Easier to implement 52-bit userspace VA support, but one needs to maintain compatibility with software expecting 48-bit VAs.

Extending the kernel VA space size

- We wish to have a single kernel image to support multiple hardware configurations, thus the 52-bit kernel VAs must be enabled at boot,
- There is some de-constifying to perform in a few areas,
- As the VA space size will change we have to re-arrange the kernel memory map,
- This includes a minor tweak to KASAN (we specify KASAN_SHADOW_OFFSET instead of KASAN_SHADOW_START, this also better matches x86 code).

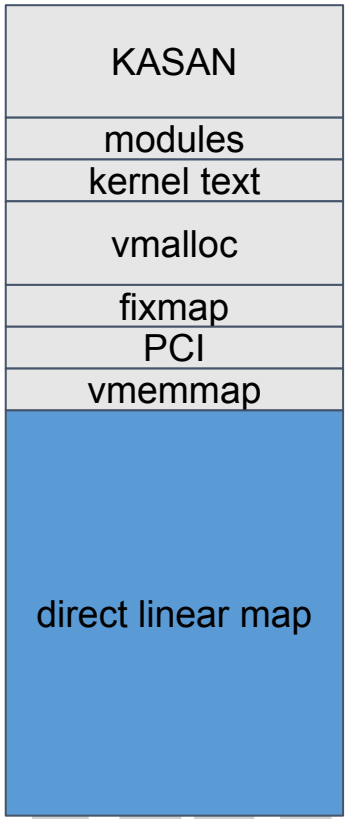
De-constifying VA_BITS

VA_BITS is an obvious candidate to change, after my patch set I proposed to redefine VA_BITS and introduce a couple more quantities:

VA_BITS	Compile time constant	Maximum size of VA space, used for things like static array and region sizes
VA_BITS_MIN	Compile time constant	Minimum size of VA space, used to ensure pointers are addressable
VA_BITS_ACTUAL	Variable	The actual size of the kernel VA space

Kernel memory map

- The direct linear map is in the “higher half” of the VA space,
- Whilst everything else is in the lower half.
- Going from 48 to 52-bits we definitely want the kernel text addresses to stay constant,
- Thus we need to flip the VA space.



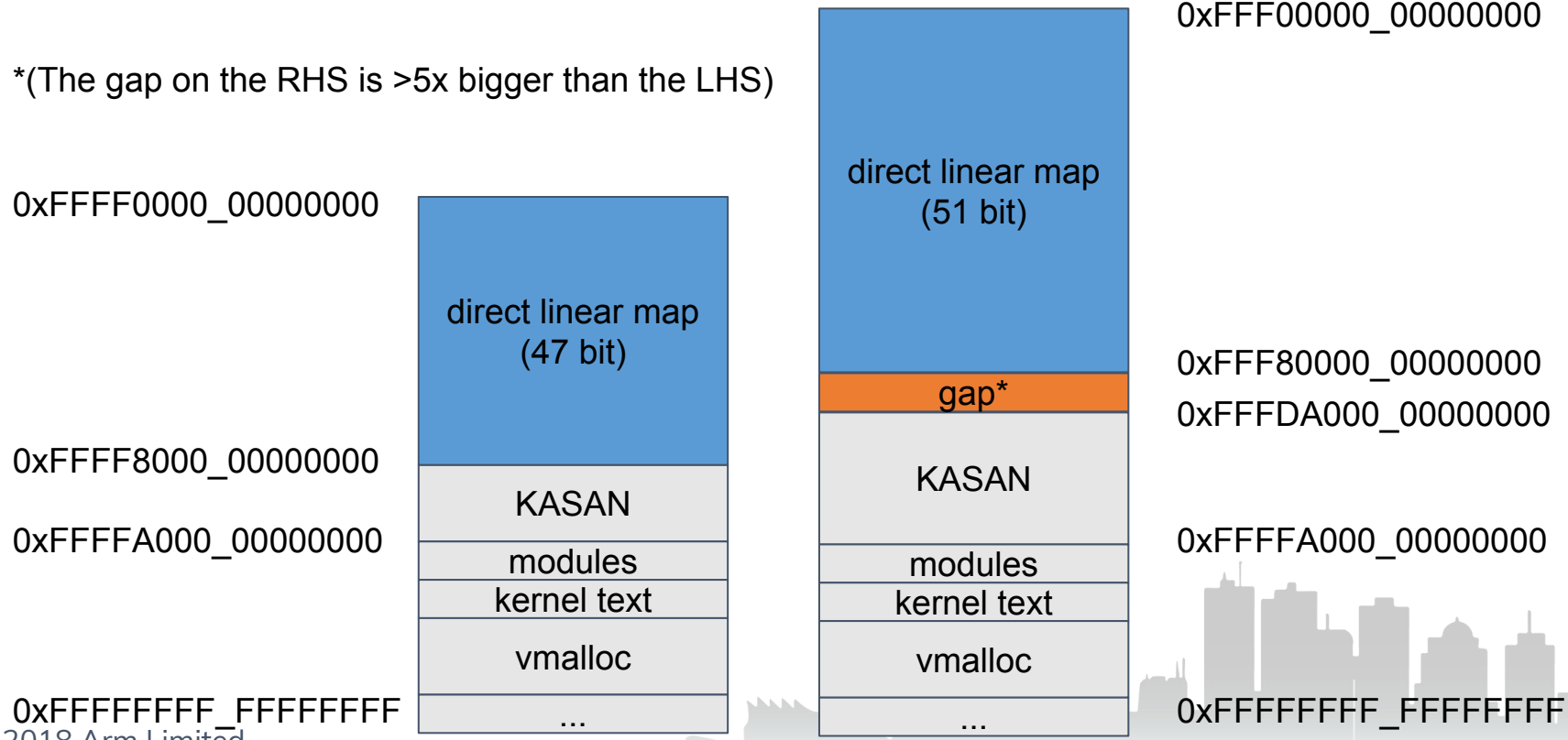
0xFFFF0000_00000000

0xFFFF8000_00000000

0xFFFFFFFF_FFFFFFFF

Adjusting the memory map

*(The gap on the RHS is >5x bigger than the LHS)



52-bit userspace VAs

- Userspace support for 52-bit VAs is much more straightforward. On Arm one can run with a 48-bit kernel VA and a 52-bit user VA simultaneously, (We can just set TCR_EL1.T0SZ s.t. userspace is 52-bit, make PGDIR_SIZE large enough for 52-bit and tweak TASK_SIZE and PGDIR_MASK),
- The major source of complexity is retaining compatibility with software that assumes a 48-bit userspace VA,
- If userspace calls mmap with an address hint with VA[51:48] != 0, then it is provided with a “high” address, (thanks to Jon Masters’ feedback a future patch will also look at the size of the requested allocation),
- For all other cases (including the dynamic loader, stack etc) we get “low” addresses.

Ramifications for userspace

- If one is assuming that pointers are at most 48-bit and they **completely control** the sources of their pointers they should not notice anything (the ELF loader will load all .so's within 48 bits and the stack will be within 48-bits just as before),
- However, **library code** that is supplied pointers, must not assume that those pointers are limited to 48-bits,
- It is usually JITs and instrumentation tools which employ tagged pointers that are affected by VA space size changes.

Current status

- Support for 52-bit kernel space has been posted upstream
  “[PATCH V3 0/8] 52-bit kernel VAs for arm64”
  http://lists.infradead.org/pipermail/linux-arm-kernel/2018-May/576842.html
- Also, support for 52-bit user space has been posted
  “[PATCH 0/5] 52-bit userspace VAs”
  http://lists.infradead.org/pipermail/linux-arm-kernel/2018-August/598588.html
- If one is testing 52-bit user space they only need to merge the second patch set.