



Reducing kernel stack memory (on arm64)

Will Deacon <will@kernel.org>

Android Systems, Google

October, 2026



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

\$ whoami

- Active upstream maintainer
- Co-maintain arm64 architecture with Catalin Marinas
- Development work mostly motivated by Android
- Fishing and beer
- Not an mm expert - thank you for having me!
 - Also thanks to Pasha, David S, Suren, Nick D, Linus W, ...



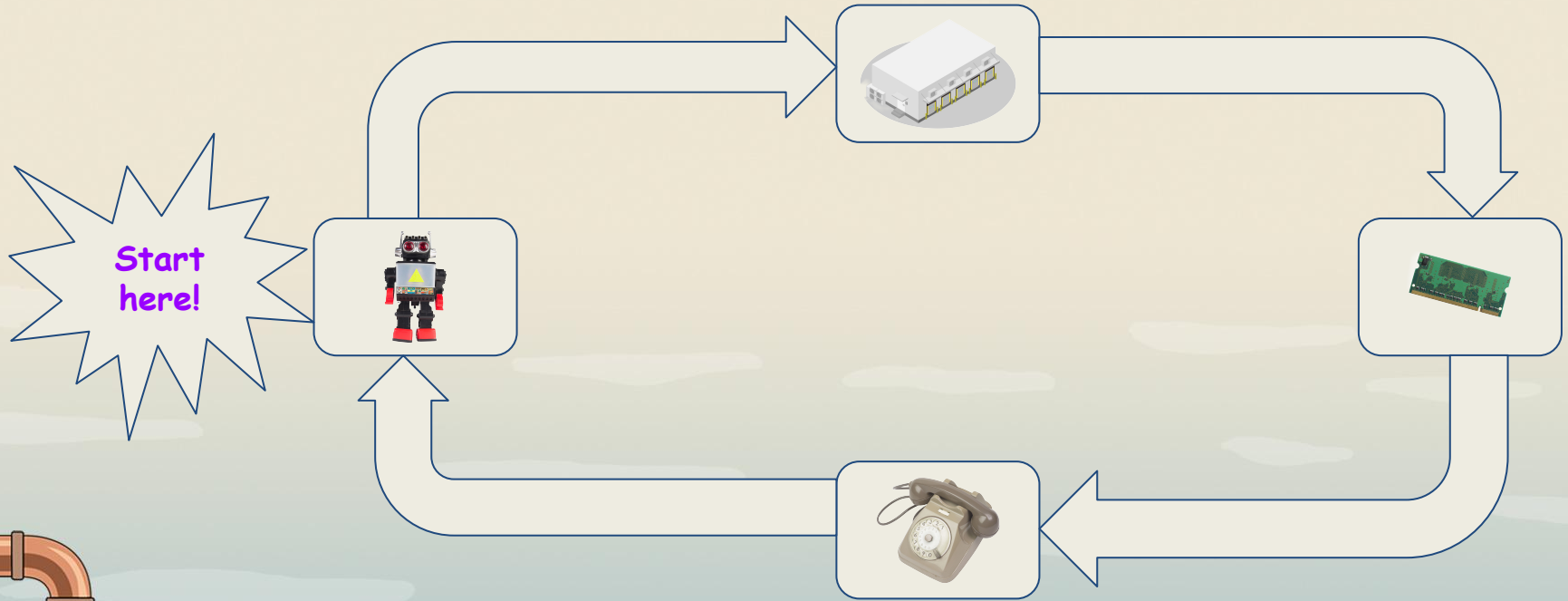
LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



DRAM "crisis"



LINUX
PLUMBERS CONFERENCE PRAGUE, CZECHIA / OCT. 5-7, 2026



Kernel stacks

- Each task has a dedicated (16k) kernel stack
 - memory is pinned and over-aligned
- Android *really* likes threads
 - 2-3k system threads + 1000s of user threads [[1](#)]
 - Idle device can have >5k threads
 - ~100MiB of kernel stack under load
- But Cloud cares too
 - “millions of threads [[2](#)] ... unused kernel stack pages occupy petabytes of memory across the fleet [[3](#)]”
- Stack memory is typically under-utilised...
- ... but overflow is fatal



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Time for some archaeology...

- v3.7 (2012): arm64 port merged with 8k stacks



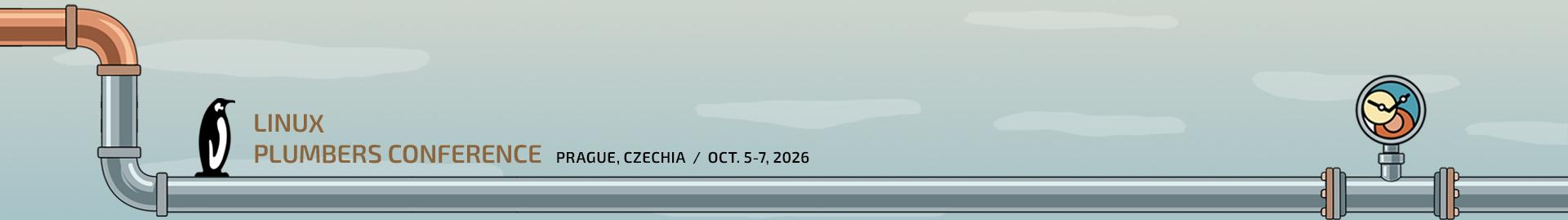
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



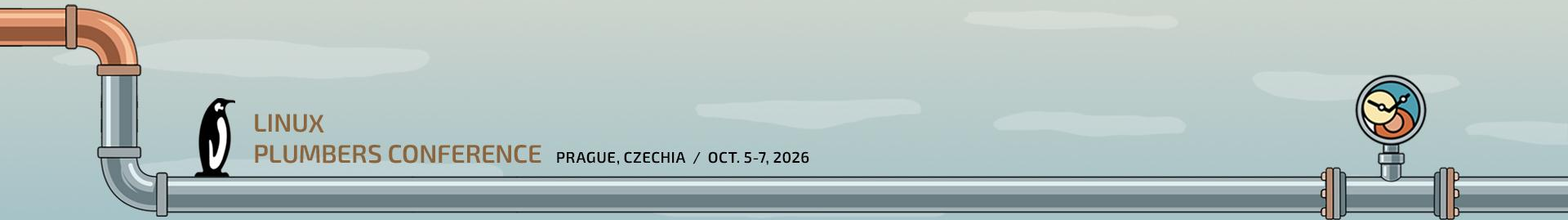
Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released



Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released
- **v3.11 (July, 2013):** arm64 [moves to 16k stacks](#) (*“This is needed because of the failures encountered when running SpecWeb benchmark test.”*).
No mailing list thread that I can find!



Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released
- **v3.11 (July, 2013):** arm64 [moves to 16k stacks](#) (*“This is needed because of the failures encountered when running SpecWeb benchmark test.”*).
No mailing list thread that I can find!
- **v3.16:** x86 [moves to 16k stacks](#) (*“arm64 already expanded (sic) to 16K. [...] So, my stupid idea is just let's expand stack size”*). Linus adds: (*“the problem is exacerbated by certain gcc versions [...] Rusty saw gcc-4.6.4 using 35% more stack on the virtio path than 4.8.2 does”*)



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released
- **v3.11 (July, 2013):** arm64 [moves to 16k stacks](#) (*“This is needed because of the failures encountered when running SpecWeb benchmark test.”*).
No mailing list thread that I can find!
- **v3.16:** x86 [moves to 16k stacks](#) (*“arm64 already expanded (sic) to 16K. [...] So, my stupid idea is just let's expand stack size”*). Linus adds: (*“the problem is exacerbated by certain gcc versions [...] Rusty saw gcc-4.6.4 using 35% more stack on the virtio path than 4.8.2 does”*)
- **v4.5 (2016):** arm64 implements per-cpu IRQ stacks



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released
- **v3.11 (July, 2013):** arm64 [moves to 16k stacks](#) (*“This is needed because of the failures encountered when running SpecWeb benchmark test.”*).
No mailing list thread that I can find!
- **v3.16:** x86 [moves to 16k stacks](#) (*“arm64 already expanded (sic) to 16K. [...] So, my stupid idea is just let's expand stack size”*). Linus adds: (*“the problem is exacerbated by certain gcc versions [...] Rusty saw gcc-4.6.4 using 35% more stack on the virtio path than 4.8.2 does”*)
- **v4.5 (2016):** arm64 implements per-cpu IRQ stacks
- **v4.14 (2017):** arm64 implements vmap'd stacks



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Time for some archaeology...

- **v3.7 (2012):** arm64 port merged with 8k stacks
- **March 2013:** GCC 4.8 released
- **v3.11 (July, 2013):** arm64 [moves to 16k stacks](#) (*“This is needed because of the failures encountered when running SpecWeb benchmark test.”*).
No mailing list thread that I can find!
- **v3.16:** x86 [moves to 16k stacks](#) (*“arm64 already expanded (sic) to 16K. [...] So, my stupid idea is just let's expand stack size”*). Linus adds: (*“the problem is exacerbated by certain gcc versions [...] Rusty saw gcc-4.6.4 using 35% more stack on the virtio path than 4.8.2 does”*)
- **v4.5 (2016):** arm64 implements per-cpu IRQ stacks
- **v4.14 (2017):** arm64 implements vmap'd stacks
- **February 2024:** Pasha [proposes](#) “dynamic kernel stacks”



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



State of Android



- Billions of active devices
- Recent convergence on *Generic Kernel Image*
 - Upstream-first mentality
 - Vendor customisations via “hooks”
- Many vendors have out-of-tree dynamic stack implementations (fault-based allocation)
 - These are invasive changes that cannot be implemented using vendor hooks
 - May not be robust in all cases
 - Independent efforts are a constant maintenance burden
 - **Android cannot (yet) offer an alternative solution**



LINUX

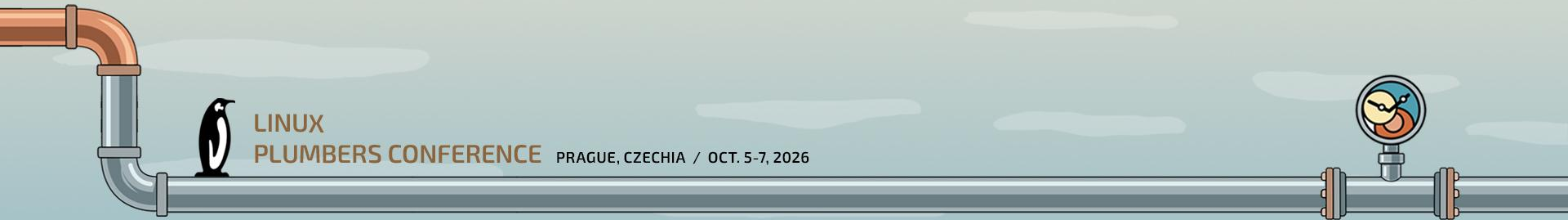
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



8K ought to be enough for anybody

- ✓ QEMU, Debian, kernel compile
- ✓ Android pre/post-submit testing
- ✓ Android CTS/VTs tests



8K ought to be enough for anybody



QEMU, Debian, kernel compile



Android pre/post-submit testing



Android CTS/VTs tests



MemoryPressureTest →

Kernel stack size	Symptom
4K	Crash at boot during driver probe
8K	Overflow in direct reclaim under stress
12K	Not supported with VMAP stack
16K	Poor memory utilisation

LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

```
<6>[ 123.859836] C3] Call trace:
<6>[ 123.859837] C3] dump_backtrace+0x0c/0x128
<6>[ 123.859843] C3] show_stack+0x18/0x28
<6>[ 123.859845] C3] dump_stack+0x10/0x0b8
<6>[ 123.859848] C3] dump_stack+0x18/0x28
<6>[ 123.859851] C3] panic+0x190/0x364
<6>[ 123.859852] C3] nmi_panic+0x30/0x88
<6>[ 123.859857] C3] panic_bad_stack+0x104/0x124
<6>[ 123.859860] C3] panic_bad_stack+0x44/0x48
<6>[ 123.859865] C3] _bad_stack+0x90/0x94
<6>[ 123.859866] C3] bpf_ktime_get_ns+0x4/0x20
<6>[ 123.859870] C3] trace_call_bpf+0xb8/0x1fc
<6>[ 123.859872] C3] perf_trace_run_bpf_submit+0x6c/0x1fc
<6>[ 123.859875] C3] perf_trace_sched_switch+0x138/0x198
<6>[ 123.859879] C3] _schedule+0x1684/0x257c
<6>[ 123.859882] C3] preempt_schedule_irq+0x60/0xb8
<6>[ 123.859888] C3] arm64_preempt_schedule_irq+0x30/0x48
<6>[ 123.859893] C3] ei1_interrupt+0x48/0x8c
<6>[ 123.859895] C3] ei1h_64_irq_handler+0x18/0x24
<6>[ 123.859898] C3] ei1h_64_irq+0x7c/0x80
<6>[ 123.859900] C3] _raw_read_unlock+0x38/0x68
<6>[ 123.859922] C3] submit_bio_noacct+0x27c/0x714
<6>[ 123.859925] C3] submit_bio+0x128/0x1c4
<6>[ 123.859926] C3] submit_bio_wait+0x50/0xb4
<6>[ 123.859928] C3] swap_writepage_bdev_sync+0x128/0x270
<6>[ 123.859932] C3] __swap_writepage_cold+0x80/0x1e4
<6>[ 123.859935] C3] swap_writepage+0x4/0x288
<6>[ 123.859936] C3] shrink_folio_list+0x640/0x1d08
<6>[ 123.859938] C3] evict_folios+0x123c/0x1b90
<6>[ 123.859940] C3] try_to_shrink_lruvec+0x1b8/0x424
<6>[ 123.859944] C3] shrink_one+0x9c/0x228
<6>[ 123.859946] C3] lru_gen_shrink_node+0x300/0x700
<6>[ 123.859951] C3] shrink_node_cold+0xd0/0x960
<6>[ 123.859954] C3] do_try_to_free_pages+0x108/0x410
<6>[ 123.859956] C3] try_to_free_pages+0x300/0xb40
<6>[ 123.859963] C3] _alloc_pages_slowpath+0x548/0x141c
<6>[ 123.859966] C3] __folio_alloc_noprof_cold+0x1cc/0x1fc
<6>[ 123.859969] C3] vma_alloc_zeroed_movable_folio+0x3c/0x7c
<6>[ 123.859972] C3] do_pte_missing+0xc0/0x14c
<6>[ 123.859974] C3] handle_mm_fault+0x218/0x640
<6>[ 123.859976] C3] do_page_fault+0x36c/0x804
<6>[ 123.859980] C3] do_translation_fault+0x38/0x54
<6>[ 123.859982] C3] do_mem_abort+0x54/0x70
<6>[ 123.859984] C3] ei1_abort+0x3c/0x5c
<6>[ 123.859986] C3] ei1h_64_sync_handler+0x40/0x0cc
<6>[ 123.859988] C3] ei1h_64_sync+0x7c/0x80
<6>[ 123.859991] C3] mm_release+0xb0/0x1a4
<6>[ 123.859994] C3] exit_mm_release+0x28/0x3c
<6>[ 123.859996] C3] do_exit+0x120/0xb8
<6>[ 123.859998] C3] do_group_exit+0x38/0x98
<6>[ 123.859999] C3] get_signal+0x58/0x750
<6>[ 123.860001] C3] do_signal+0x2b0/0x2208
<6>[ 123.860003] C3] do_notify_resume+0xe4/0x1e4
<6>[ 123.860005] C3] ei0_svc+0x84/0xd8
<6>[ 123.860007] C3] ei0t_64_sync_handler+0x6c/0xf4
<6>[ 123.860009] C3] ei0t_64_sync+0x1bc/0x1c0
<6>[ 123.860011] C3] SMP: stopping secondary CPUs
```



Direct reclaim shows up a lot

```
<6>[ 123.859922][ C3] submit_bio_noacct+0x27c/0x714
<6>[ 123.859925][ C3] submit_bio+0x128/0x1c4
<6>[ 123.859926][ C3] submit_bio_wait+0x50/0xb4
<6>[ 123.859928][ C3] swap_writepage_bdev_sync+0x128/0x270
<6>[ 123.859932][ C3] __swap_writepage.cold+0x80/0x1e4
<6>[ 123.859935][ C3] swap_writepage+0xf4/0x288
<6>[ 123.859936][ C3] shrink_folio_list+0x640/0x10d8
<6>[ 123.859938][ C3] evict_folios+0x123c/0x1b90
<6>[ 123.859940][ C3] try_to_shrink_lruvec+0x1b8/0x424
<6>[ 123.859944][ C3] shrink_one+0x9c/0x228
<6>[ 123.859946][ C3] lru_gen_shrink_node+0x300/0x700
<6>[ 123.859951][ C3] shrink_node.cold+0xd0/0x960
<6>[ 123.859954][ C3] do_try_to_free_pages+0x108/0x410
<6>[ 123.859956][ C3] try_to_free_pages+0x300/0x6a0
<6>[ 123.859963][ C3] __alloc_pages_slowpath+0x548/0x141c
<6>[ 123.859966][ C3] __folio_alloc_noprof.cold+0x1cc/0x1fc
<6>[ 123.859969][ C3] vma_alloc_zeroed_movable_folio+0x3c/0x7c
<6>[ 123.859972][ C3] do_pte_missing+0xc0/0x14fc
<6>[ 123.859974][ C3] handle_mm_fault+0x218/0x640
```



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Compilers

- Toolchains play a big part in stack consumption
 - GCC introduced `-fconserve-stack` in 4.9
 - Nick tells me that Clang has known issues with excessive stack consumption in some cases but work is ongoing to improve this [[1](#), [2](#)]
- A compile-time option for the kernel stack size doesn't scale
 - It will always get bumped as soon as somebody hits an overflow in their environment



Recent upstream(ish) work

- [\[PATCH v2 00/21\] arm64: Move overflow sp into SP_EL1 and kernel sp into SP_EL0](#) [Will Deacon]
 - Dedicated exception stack for arm64 to generalise stack overflow handling
- [\[RFC PATCH 00/15\] arm64: Set kernel stack size from cmdline](#) [Mostafa]
 - Kernel stack size now configurable on cmdline for arm64 & support for 12K stacks
- [PATCH 1/1] mm/vmscan: execute direct reclaim in a worker [Suren]
 - Move direct reclaim (do_try_to_free_pages()) to a worker thread
 - 8k stacks back on the table
- [b4/aarch64-dynamic-kernel-stacks-v7.1-rc1](#) [Linus Walleij]
 - Rebase of Pasha's series and port to arm64
- [\[RFC 00/10\] Reclaimable kernel stacks](#) [David Stevens]
 - Alternative to dynamic stacks; reclaim stack pages from blocked tasks. Annotations to avoid deadlock.

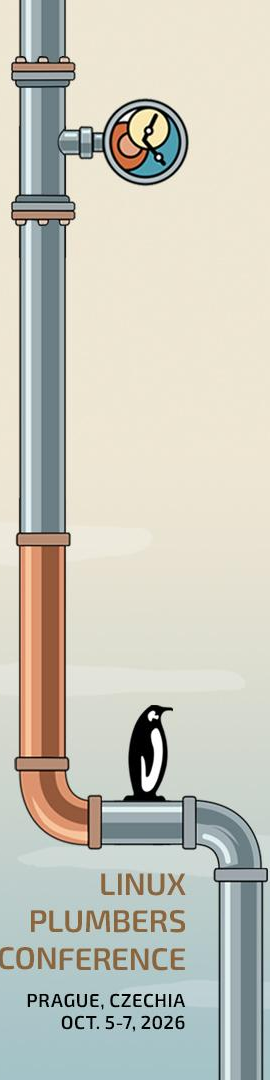


LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026





Discussion:

- Is there upstream interest in kernel stack reduction beyond arm64? (e.g. for x86)
- Is an 8KiB stack “good enough” with direct reclaim in a separate kthread?
- Can we improve kstack utilisation metrics?
- Is there any future for dynamic stacks?
- Can we improve stack usage in the toolchain?
- Global vs per-task stack size:
 - APIs to configure per-task kernel stack size?
 - ...from userspace?
- ???