GKI

Ending kernel fragmentations on Android

Juhyung Park

arter97@dgist.ac.kr

arter97

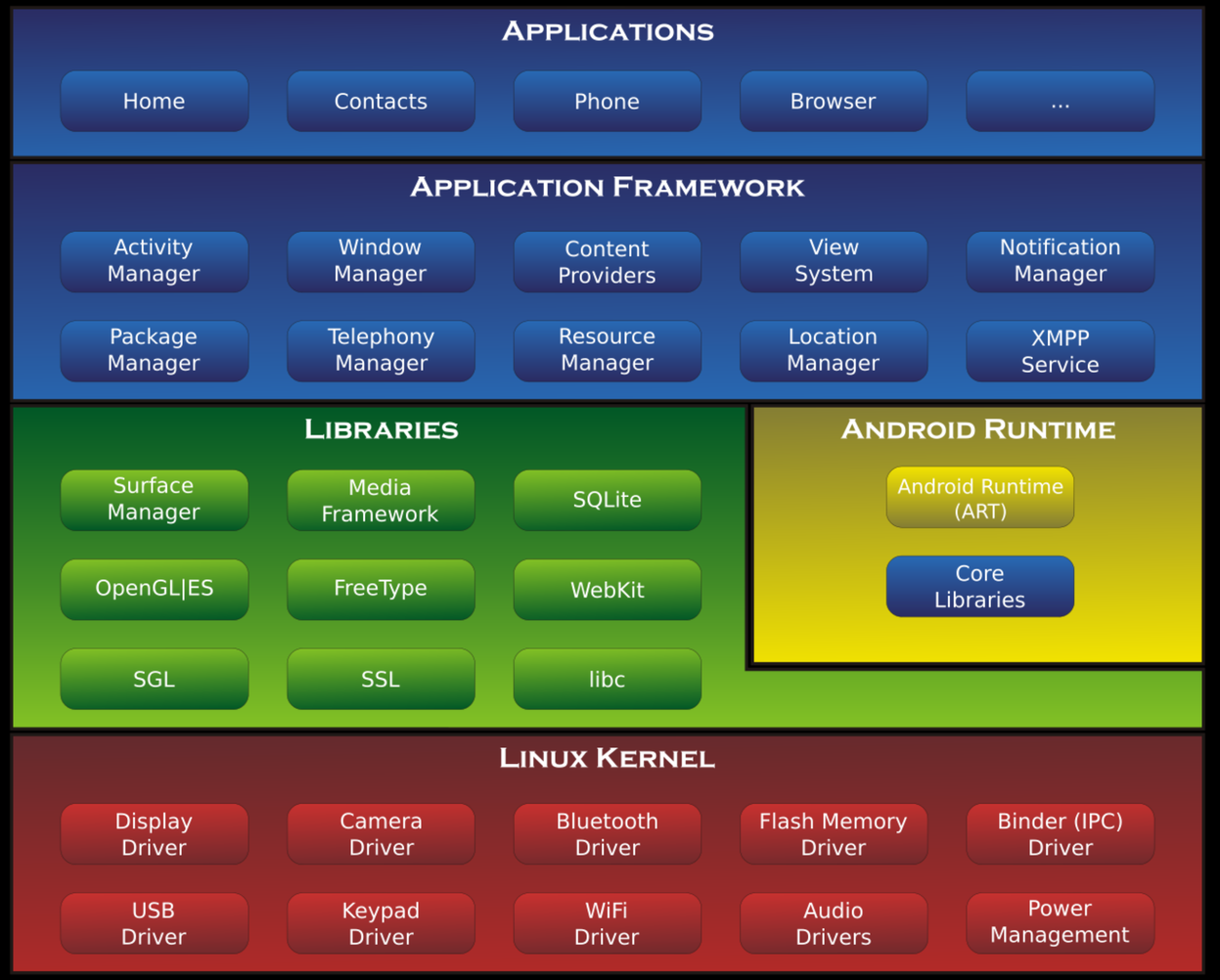
@arter97

About me

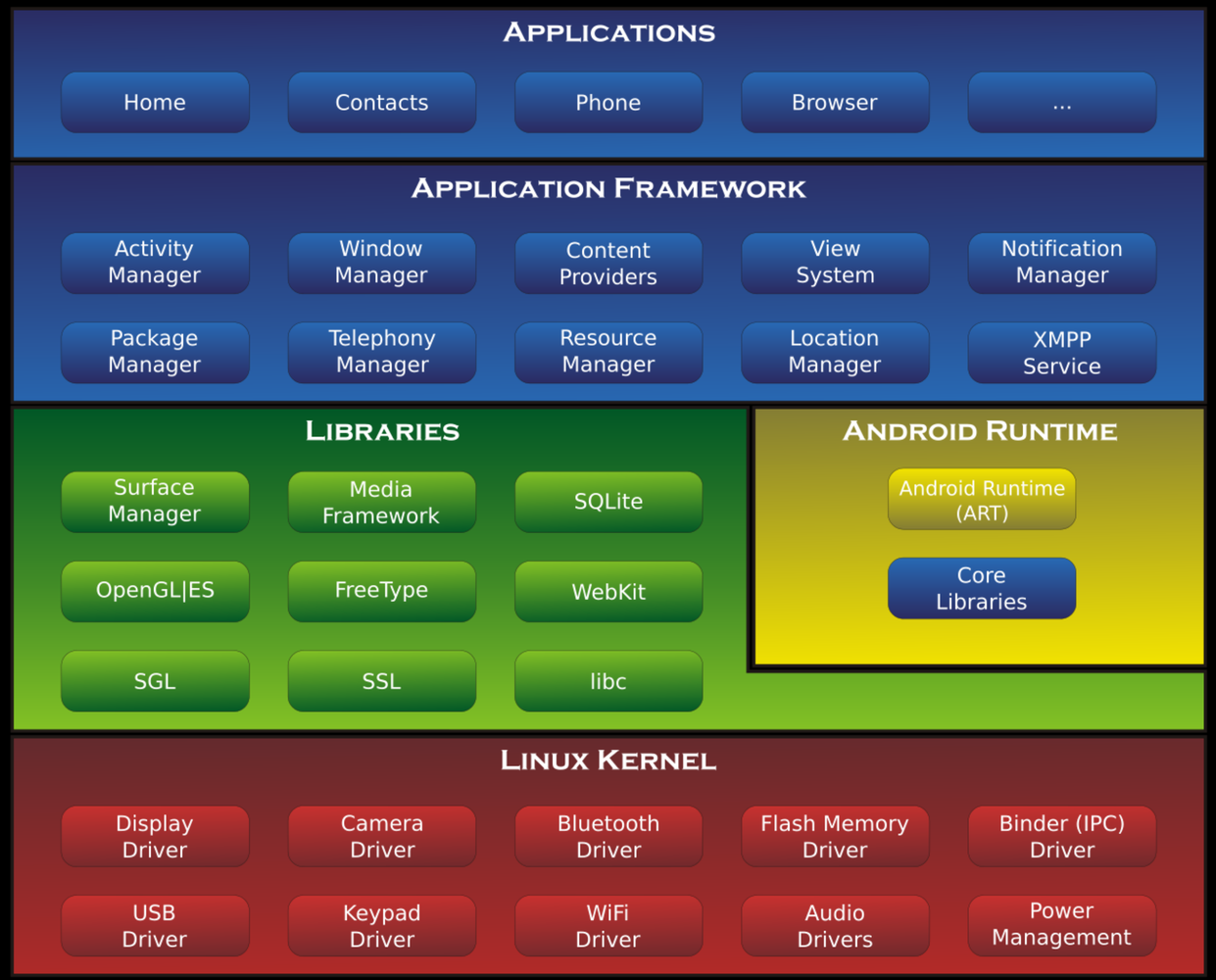
- Hobbyist Android / kernel developer since 2012
- Ph.D. student @ DGIST DataLab
- Interests: storage, Android

- 5th kernel meet-up (2019)
  - Historical overview of Android and its kernel

Android fragmentation



Android fragmentation

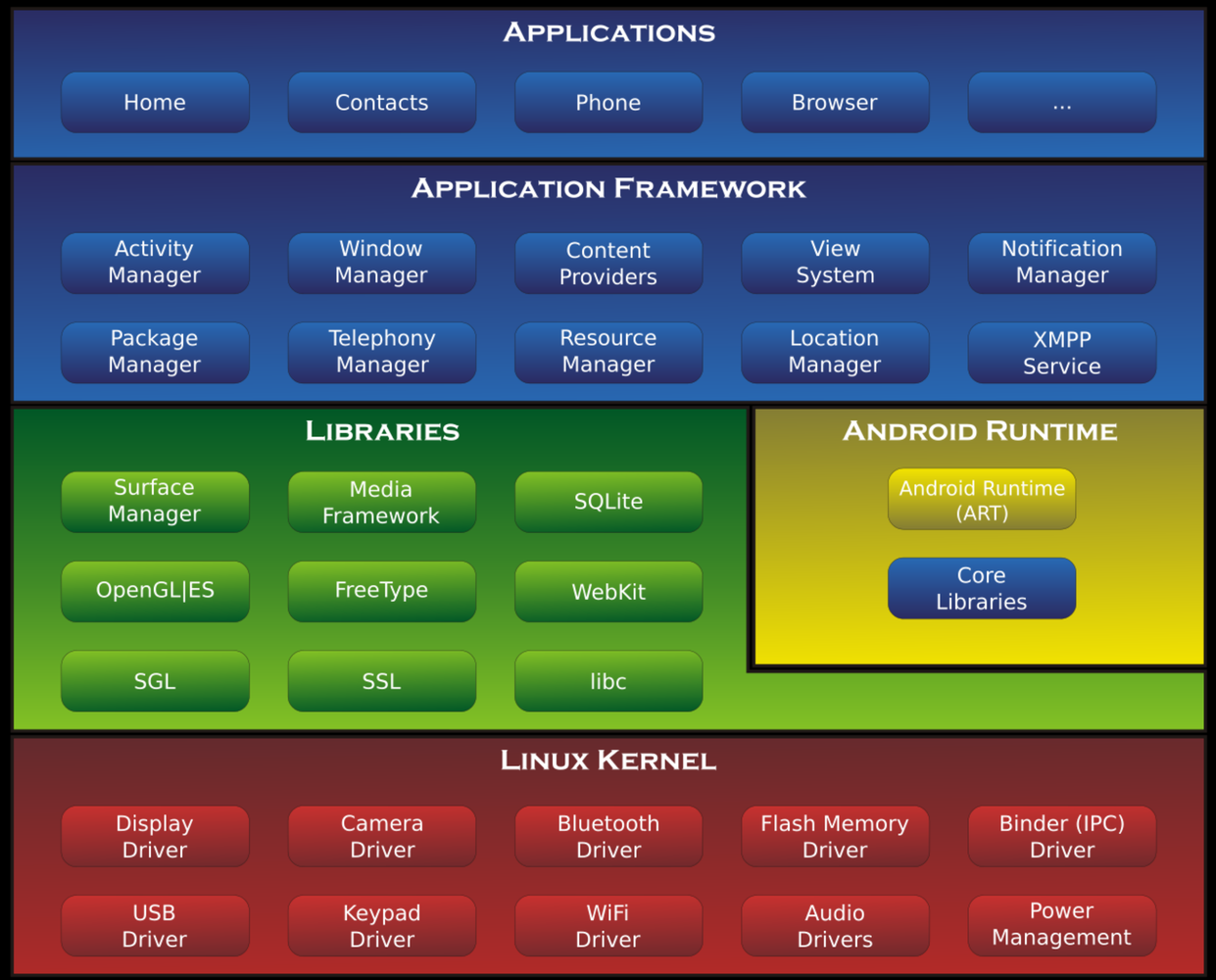


Regular app developers

OEM customizations - Userspace

OEM customizations - Kernel

Android fragmentation



Regular app developers

OEM customizations - Userspace

HAL standardized with Project Treble

OEM customizations - Kernel

“Standardizing mobile kernels? Pfff, what a joke.”

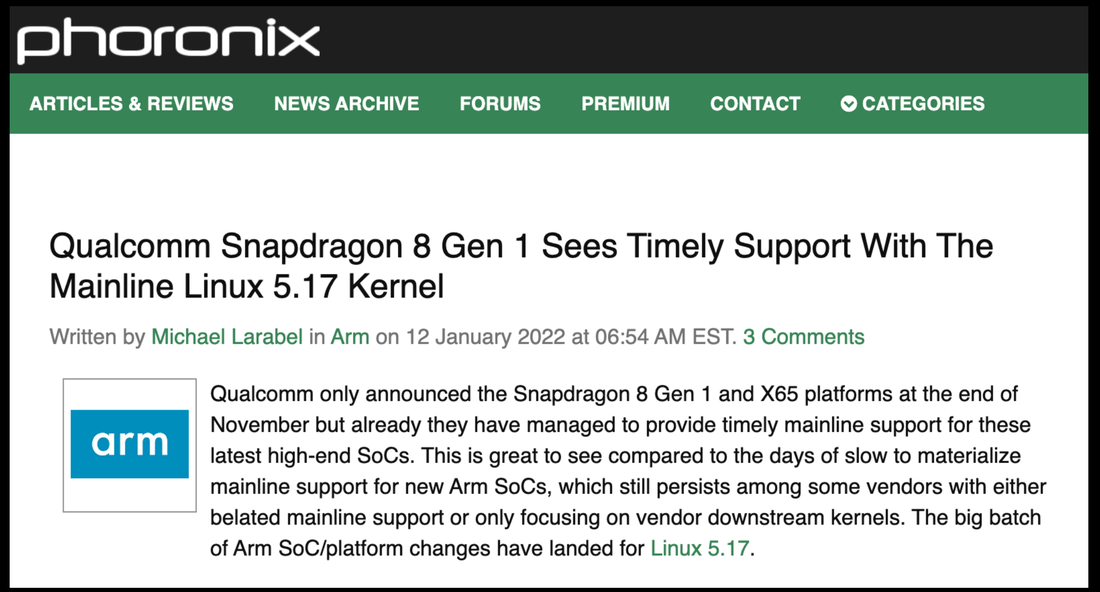
ARM kernel fragmentation

```
From          Linus Torvalds <>
Date          Thu, 17 Mar 2011 19:50:36 -0700
Subject       Re: [GIT PULL] omap changes for v2.6.39 merge window

Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.

You need to stop stepping on each others toes.

<...>

Somebody needs to get a grip in the ARM community. I do want to do
these merges, just to see how screwed up things are, but guys, this is
just ridiculous. The pure amount of crazy churn is annoying in itself,
but when I then get these "independent" pull requests from four
different people, and they touch the same files, that indicates that
something is wrong.

Somebody in the ARM community really needs to step up and tell people
to stop dicking around.
```

```
$ tree -L 1 arch/arm
arch/arm
├── boot
├── common
├── configs
├── crypto
├── include
├── Kconfig
├── Kconfig.debug
├── Kconfig-nommu
├── kernel
├── kvm
├── lib
├── mach-actions
├── mach-alpine
...
├── mach-vt8500
├── mach-zx
├── mach-zynq
├── Makefile
├── mm
├── net
├── nwfpe
├── oprofile
├── plat-omap
├── plat-orion
├── plat-pxa
├── plat-samsung
├── plat-versatile
├── probes
├── tools
├── vdso
├── vfp
└── xen

94 directories, 4 files
```

* arm64 is slightly better with “device tree”

Mainline kernel?



https://www.phoronix.com/news/Linux-5.17-Arm

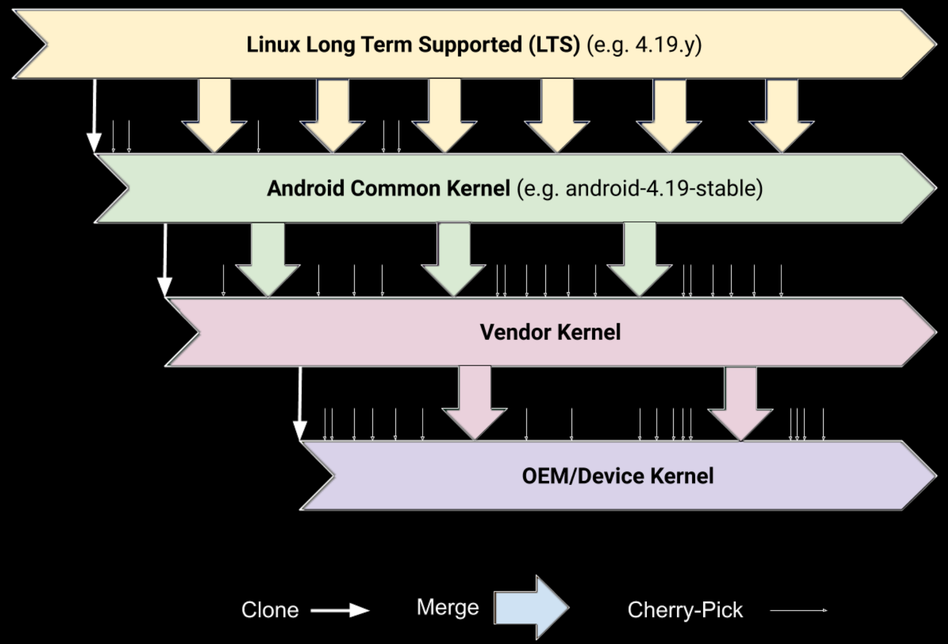
Mainline kernel?

Nope.

	Mainline	Downstream
CPU	✓	✓
GPU	▲	✓
Storage	▲	✓
Modem	✕	✓
Wi-Fi	✕	✓
Bluetooth	✕	✓
Camera	✕	✓
Battery	✕	✓
Audio	✕	✓
Biometrics	✕	✓
Touch panel	✕	✓

Android downstream kernel

- Upstream: The Linux kernel from kernel.org
- AOSP: Additional Android-specific patches from AOSP common kernels
- Vendor: SoC and peripheral enablement and optimization patches from vendors
- OEM/device: Additional device drivers and customizations



Android downstream kernel

Cost of fragmentation

- Security updates are labor intensive
- Merging linux-stable is difficult
  - 90% of kernel security issues are resolved from linux-stable
- Android platform upgrade conflicts
  - New version of Android may require new kernel changes
- Difficult to upstream changes

GKI

Generic Kernel Image

- Unifying the core kernel and separating SoC / board support
- Maintaining a stable ABI (KMI) for vendor modules
  - Allows modules and the core kernel to be updated independently
  - Huh?

ABI stability in the Linux kernel?
stable-api-nonsense

Greg Kroah-Hartman <greg@kroah.com>

This is being written to try to explain why Linux **does not have a binary kernel interface, nor does it have a stable kernel interface**.

You think you want a stable kernel interface, but you really do not, and you don't even know it.

https://www.kernel.org/doc/Documentation/process/stable-api-nonsense.rst

ABI stability in the Linux kernel?
stable-api-nonsense

Ever wondered why your NVIDIA / VMware drivers are broken or rebuilt upon kernel upgrades?

https://www.kernel.org/doc/Documentation/process/stable-api-nonsense.rst

ABI stability in the Linux kernel?
stable-api-nonsense

- The compiler may change data structure order and padding
- Different kernel configuration may produce incompatible changes
  - Structures containing different fields
  - Unimplemented features
  - Memory alignment changes
- New commits and fixes may change data structure

https://www.kernel.org/doc/Documentation/process/stable-api-nonsense.rst

ABI stability in the Linux kernel?

What's the upstream's answer?

"Simple, get your kernel driver into the main kernel tree."

Infeasible for Android OEMs for obvious reasons.

https://www.kernel.org/doc/Documentation/process/stable-api-nonsense.rst

Maintaining a stable KMI
Kernel Module Interface

- Only a single configuration, gki_defconfig, is used to build the core kernel
- KMI is maintained within the same LTS and Android version of a kernel
  - e.g., android13-5.10, android12-5.10, android13-5.15
- Only one **specific LLVM toolchain** can be used to build the kernel and modules
- Symbols used from modules are whitelisted/exported
  - EXPORT_SYMBOL() isn’t enough
  - Vendors have to upload patches to change the symbols list

Maintaining a stable KMI

KMI breakages

```
diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
 index 5ed8f6292a53..f2ecb34c7645 100644
 --- a/include/linux/mm_types.h
 +++ b/include/linux/mm_types.h
 @@ -339,6 +339,7 @@ struct core_state {
  struct kioctx_table;
  struct mm_struct {
     struct {
 +       int dummy;
         struct vm_area_struct *mmap;            /* list of VMAs */
         struct rb_root mm_rb;
         u64 vmacache_seqnum;                    /* per-thread vmacache */
```

Maintaining a stable KMI

Detecting KMI breakages

```
Leaf changes summary: 1 artifact changed
 Changed leaf types summary: 1 leaf type changed
 Removed/Changed/Added functions summary: 0 Removed, 0 Changed, 0 Added function
 Removed/Changed/Added variables summary: 0 Removed, 0 Changed, 0 Added variable

 'struct mm_struct at mm_types.h:372:1' changed:
   type size changed from 6848 to 6912 (in bits)
   there are data member changes:
 [...]
```

Maintaining a stable KMI

KABI struct reservation

- Paddings are reserved for future uses
- Mostly taken from RedHat’s RHEL/CentOS KABI patches

ANDROID: GKI: add some padding to some driver core structures

```
These structures are fundamental to implementing fw_devlink and
sync_state(). Since they are still evolving, add some padding in case we
need to backport any important bug fixes.

        struct device_link
        struct class
        struct fwnode_handle
        struct fwnode_link

Bug: 183615740
Signed-off-by: Saravana Kannan <saravanak@google.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@google.com>
Change-Id: Id9daf7cf9ae5d94fb0134144f8220a241ccbaef8
```

android13-5.15 + android13-5.15-2022-06 + android13-5.15-2022-07 + android13-5.15-2022-08 + android13-5.15-2022-09 + android13-5.15-arcvm + android13-5.15-lts

Saravana Kannan authored and **Carlos Llamas** committed on Jun 19

Showing **3 changed files** with **8 additions** and **0 deletions**.

Filter changed files

- include/linux
  - device.h
- device
  - class.h
  - fwnode.h

2 include/linux/device.h

```
@@ -612,6 +612,8 @@ struct device_link {
612 612         struct kref kref;
613 613         struct work_struct rm_work;
614 614         bool supplier_preactivated; /* Owned by consumer probe. */
    615 +       ANDROID_KABI_RESERVE(1);
    616 +       ANDROID_KABI_RESERVE(2);
615 617 };
616 618
617 619 static inline struct device *kobj_to_dev(struct kobject *kobj)
```

Maintaining a stable KMI
Downstream kernel development

- If data structures are changed, send it to Android Common Kernel (ACK)
  - Preferably before KMI is frozen
- Refactor so that data structure doesn’t need to diverge from ACK
- Paddings are used here too

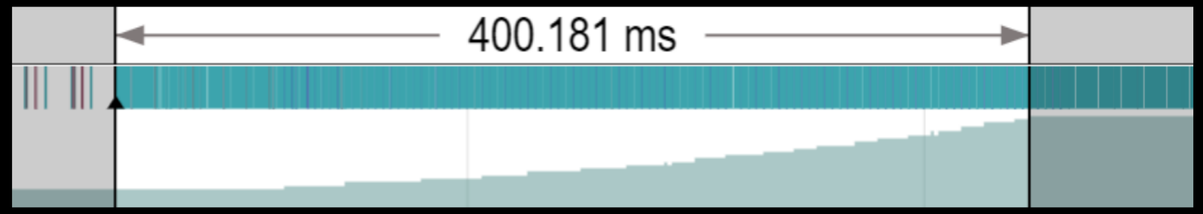
What about un-upstreamable changes?

- Vendor-hooks to modify default behavior of GKI
  - Preferred than “weak” functions
- Not enough; some OEMs heavily modifies the kernel
  - Very core - arch, mm, sched, etc
  - Some are even hw-specific
- GKI 2.0’s goal - “Don't introduce significant performance or power regressions when replacing the product kernel with the GKI kernel.”

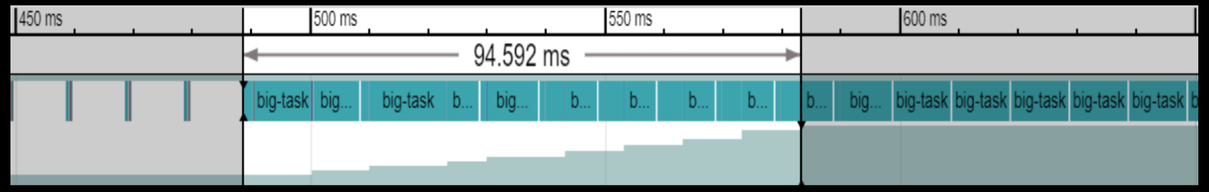
Downstream kernel development

Case-study: Qualcomm's WALT scheduler

- PELT (Per Entity Load Tracking) uses moving load average to calculate the next task demand
  - Slow ramp-up



- WALT (Window Assisted Load Tracking) references previous N windows to predict the next task demand



https://resources.linaro.org/en/resource/dLCbKcfzmgKthDRCRCzd9v

Downstream kernel development
Case-study: Qualcomm's WALT scheduler

- How many *core* files were changed with v4.19?
  - Excluding walt-specific files

```
git grep -i walt LA.UM.9.12.1.r1-03100-SMxx50.QSSI12.0 -- kernel/sched | grep -vP 'walt.*\.?:' | wc -l
250
```

- What about v5.10? (GKI 2.0)

```
git grep -i walt google/android13-5.10 -- kernel/sched | grep -vP 'walt.*\.?:' | wc -l
0
```

  - How?

Downstream kernel development

Case-study: Qualcomm's WALT scheduler

- Just a bunch of "hooks"


```
$ git log --oneline --no-merges google/android12-5.4..google/android13-5.10 -- kernel/sched | grep ANDROID: | grep -i vendor | grep -i hook
19e41a340441b ANDROID: sched: Add vendor hook for cpu distribution functions
65735b81dd25c ANDROID: sched: Add vendor hook for update_rq_clock_pelt
3f9db3f711817 ANDROID: sched: Add vendor hook for rt util update
e3356ca0a6c24 ANDROID: sched: Add vendor hook for util-update related functions
0975fd934e9e3 ANDROID: Add vendor hook for the sugov_get_util
cf33d6fae045e ANDROID: vendor_hooks: add a hook to control the delay time of frequency up and down.
61b66296b2a24 ANDROID: Add vendor hook for the uclamp_util
478f3ee1d3e5b ANDROID: Add vendor hook to the deadline scheduler
f9fcdaeab7006 ANDROID: sched: remove regular vendor hooks for 32bit execve
13af062abf9b3 ANDROID: vendor_hooks: Export the tracepoints sched_stat_sleep and sched_waking to let module probe them
27c285003d2a8 ANDROID: sched: Add vendor hook to select ilb cpu
a6bb1af39d11e ANDROID: vendor_hooks: Export the tracepoints sched_stat_iowait, sched_stat_blocked, sched_stat_wait to let modules probe them
fe580539f6cec ANDROID: vendor_hooks: Add hooks for account irqtime process tick
f3f8d55011837 ANDROID: sched: Add vendor hooks for update_load_avg
54f66141a8834 ANDROID: sched: Add vendor hooks for sched.
58b10706f3c8d ANDROID: sched: Add vendor hooks to compute new cpu freq.
50aa353c6957b ANDROID: sched: Add vendor hooks for cpu affinity.
b79d1815c400c ANDROID: psi: Add vendor hooks for PSI tracing
63399b4e2e25b ANDROID: vendor_hooks: Add hooks for account process tick
187306ab1aeef ANDROID: Partial revert of 06881e01b564 ("ANDROID: sched: Add vendor hooks for override sugov behavior")
b542f4c389e9a Revert "ANDROID: sched: Add vendor hooks for skipping sugov update"
475aea007d656 ANDROID: sched: Add vendor hook for util_est_update
53e8099784435 ANDROID: vendor_hooks: Add hooks for scheduler
cc1f93cb20c46 ANDROID: sched: Add vendor hook for uclamp_eff_get
067bf187ef4c0 ANDROID: sched: move vendor hook to check scheduling nice value
4b88cf85247f6 ANDROID: sched: Add vendor hook for uclamp_eff_value
97368fc2dcc29 ANDROID: Add a vendor hook that allow a module to modify the wake flag
f5998fbf2dbbc ANDROID: sched: Add vendor hook for cpu_overutilized
a56f081c5b2ec ANDROID: sched: Add restricted vendor hooks in CFS scheduler
18ebdc37464a1 ANDROID: sched: add vendor hooks for bad scheduling
4b5f345c7391b ANDROID: sched: Add restrict vendor hooks for balance_rt()
18b294fa4c884 ANDROID: sched: add em_cpu_energy vendor hook
11d9d07f3f686 ANDROID: schedutil: add vendor hook for adjusting util to freq calculation
fb03f7bef8ea4 ANDROID: sched: Export symbol for vendor RT hook funcion
95fd09fa3a0f1 ANDROID: sched: Export symbol for vendor RT hook funcion
d31ea0950a30a ANDROID: sched/fair: fix place_entity() vendor hook
292f430816f36 ANDROID: Sched: Add restricted vendor hooks for scheduler
f34f38632fb72 ANDROID: sched/core: Add vendor hook to change task affinity
846bf8e8cb392 ANDROID: sched: Add vendor hooks for skipping sugov update
06881e01b564a ANDROID: sched: Add vendor hooks for override sugov behavior
147a9b3d9eab7 ANDROID: sched: Add vendor hooks for find_energy_efficient_cpu
878495dacd22d ANDROID: sched: Add restrict vendor hooks for load balance
ebce8ec6bd55e ANDROID: sched: rt: rearrange invocation of find_lowest_rq() vendor hook
a2ca8408de618 ANDROID: sched: add restrict vendor hook to modify load balance behavior
2935d588b1378 ANDROID: sched: Use normal vendor hook in scheduler tick
9ad8ff902e39f ANDROID: vendor_hooks: add waiting information for blocked tasks
f39f3ac200dbb ANDROID: sched: add vendor hooks to handle scheduling priority
16330b35600f0 ANDROID: Add vendor hooks to the scheduler
a1fc1fba460be ANDROID: sched: add restrict vendor hook to modify task placement policy in EAS
```

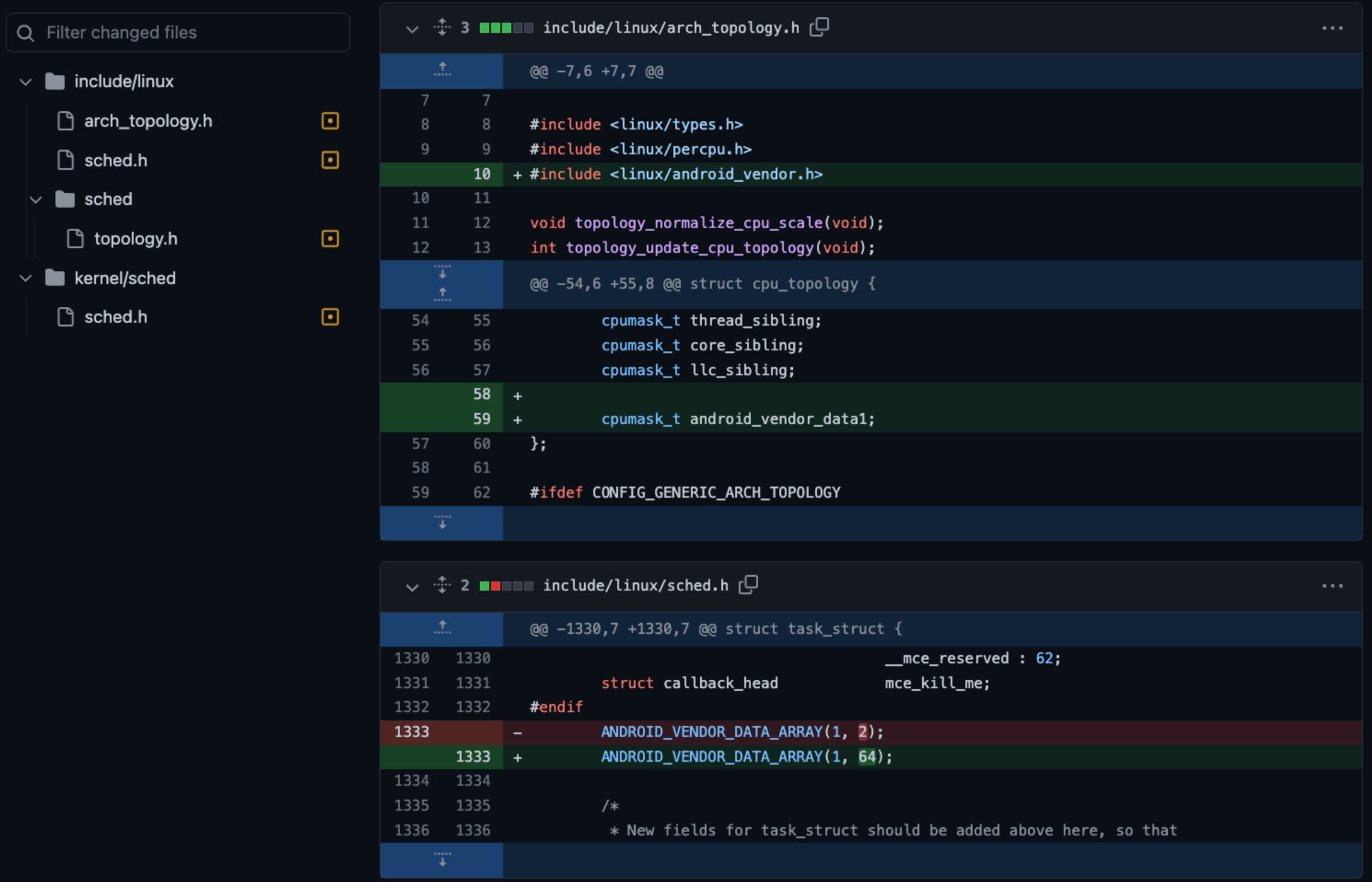
ANDROID: sched: gki: add padding to some structs to support WALT

Browse files

```
Add padding to below structs to support WALT based accounting:
        1. struct cpu_topology
        2. struct task_struct
        3. struct sched_domain_shared
        4. struct task_group
        5. struct root_domain
        6. struct rq

To accommodate potential future changes, reserving more memory than
what WALT needs today.

Bug: 171858786
Change-Id: If6d901174fc7963be3ae44daa799cb2953669ec1
Signed-off-by: Satya Durga Srinivasu Prabhala <satyap@codeaurora.org>
```

Satya Durga Srinivasu Prabhala authored and **toddkjos** committed on Oct 30, 2020 1 parent 8d5dc2c commit 7a2a316228bcd80821a15fed6cd70c80a5daa182

Showing **4 changed files** with **14 additions** and **1 deletion**. Split Unified


Filter changed files
include/linux
arch_topology.h
sched.h
sched
topology.h
kernel/sched
sched.h
3 include/linux/arch_topology.h
@@ -7,6 +7,7 @@
#include <linux/types.h>
#include <linux/percpu.h>
+ #include <linux/android_vendor.h>
void topology_normalize_cpu_scale(void);
int topology_update_cpu_topology(void);
@@ -54,6 +55,8 @@ struct cpu_topology {
cpumask_t thread_sibling;
cpumask_t core_sibling;
cpumask_t llc_sibling;
+
+ cpumask_t android_vendor_data1;
};
#ifdef CONFIG_GENERIC_ARCH_TOPOLOGY
2 include/linux/sched.h
@@ -1330,7 +1330,7 @@ struct task_struct {
__mce_reserved : 62;
struct callback_head mce_kill_me;
#endif
- ANDROID_VENDOR_DATA_ARRAY(1, 2);
+ ANDROID_VENDOR_DATA_ARRAY(1, 64);
/*
* New fields for task_struct should be added above here, so that

Downstream kernel development

Case-study: Qualcomm's WALT scheduler - Data reserved on structs

```
$ git grep -i android_ KERNEL.PLATFORM.1.0.r1-11200-kernel.0 kernel/sched/walt
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/boost.c:        wtg = (struct walt_task_group *) tg->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/boost.c:        wtg = (struct walt_task_group *) tg->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/boost.c:        wtg = (struct walt_task_group *) tg->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/fixup.c:static void android_rvh_show_max_freq(void *unused, struct cpufreq_policy *policy,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/fixup.c:        register_trace_android_rvh_show_max_freq(android_rvh_show_max_freq, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:      register_trace_android_rvh_irqs_disable(note_irq_disable, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:      register_trace_android_rvh_irqs_enable(test_irq_disable_long, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:      register_trace_android_rvh_preempt_disable(note_preempt_disable, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:      register_trace_android_rvh_preempt_enable(test_preempt_disable_long,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/sched_avg.c:    struct walt_rq *wrq = (struct walt_rq *) cpu_rq(cpu)->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/sysctl.c:               wts = (struct walt_task_struct *) task->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/sysctl.c:       wts = (struct walt_task_struct *) task->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/trace.c:        struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/trace.c:        struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/trace.h:                struct walt_rq *wrq = (struct walt_rq *) cpu_rq(cpu)->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/trace.h:                        ((struct walt_task_struct *) p->android_vendor_data1)->unfilter;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/trace.h:                        ((struct walt_task_struct *) p->android_vendor_data1)->iowaited;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) current->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) p->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) p->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) p->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) cpu_rq(cpu)->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) p->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_task_struct *wts = (struct walt_task_struct *) p->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) cpu_rq(cpu)->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c: struct walt_rq *wrq = (struct walt_rq *) rq->android_vendor_data1;
```

Downstream kernel development

Case-study: Qualcomm's WALT scheduler - Vendor hooks for altering behaviors

```
$ git grep -i android_ KERNEL.PLATFORM.1.0.r1-11200-kernel.0 kernel/sched/walt | grep rvh
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/fixup.c:static void android_rvh_show_max_freq(void *unused, struct cpufreq_policy *policy,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/fixup.c:register_trace_android_rvh_show_max_freq(android_rvh_show_max_freq, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:register_trace_android_rvh_irqs_disable(note_irq_disable, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:register_trace_android_rvh_irqs_enable(test_irq_disable_long, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:register_trace_android_rvh_preempt_disable(note_preempt_disable, NULL);
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/preemptirq_long.c:register_trace_android_rvh_preempt_enable(test_preempt_disable_long,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_cpu_cgroup_online(void *unused, struct cgroup_subsys_state *css)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_cpu_cgroup_attach(void *unused,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_wake_up_new_task(void *unused, struct task_struct *new)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_update_cpu_capacity(void *unused, int cpu, unsigned long *capacity)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_sched_cpu_starting(void *unused, int cpu)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_sched_cpu_dying(void *unused, int cpu)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_set_task_cpu(void *unused, struct task_struct *p, unsigned int new_cpu)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_new_task_stats(void *unused, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_account_irq(void *unused, struct task_struct *curr, int cpu, s64 delta)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_flush_task(void *unused, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_enqueue_task(void *unused, struct rq *rq, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_dequeue_task(void *unused, struct rq *rq, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_update_misfit_status(void *unused, struct task_struct *p,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_try_to_wake_up(void *unused, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_try_to_wake_up_success(void *unused, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_tick_entry(void *unused, struct rq *rq)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_schedule(void *unused, struct task_struct *prev,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_resume_cpus(void *unused, struct cpumask *resuming_cpus, int *err)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_update_cpus_allowed(void *unused, struct task_struct *p,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_sched_setaffinity(void *unused, struct task_struct *p,
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_sched_fork_init(void *unused, struct task_struct *p)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_ttwu_cond(void *unused, bool *cond)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_sched_exec(void *unused, bool *cond)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_build_perf_domains(void *unused, bool *eas_check)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_force_compatible_pre(void *unused, void *unused2)
KERNEL.PLATFORM.1.0.r1-11200-kernel.0:kernel/sched/walt/walt.c:static void android_rvh_force_compatible_post(void *unused, void *unused2)
```

Still not enough
GKI is not fully enforced

- “In GKI 2.0, devices launching with kernel version 5.10 or higher must ship with the GKI kernel (beginning with Android 12).”
  - Notice that it doesn’t say “*Google’s* GKI kernel”
- OEMs are allowed to change the core kernel as long as it follows the GKI / KMI model

https://source.android.com/docs/core/architecture/kernel/generic-kernel-image

In practice

Snapdragon 8 Gen 1 (2022)

```
taro:/dev/block/bootdevice/by-name # ls
ALIGN_TO_128K_1  dtbo_b          modemst1        tz_a
ALIGN_TO_128K_2  featenabler_a   modemst2        tz_b
abl_a            featenabler_b   multiimgoem_a   tzsc
abl_b            frp             multiimgoem_b   uefi_a
aop_a            fsc             multiimgqti_a   uefi_b
aop_b            fsg             multiimgqti_b   uefisecapp_a
aop_config_a     hyp_a           mysuper_a       uefisecapp_b
aop_config_b     hyp_b           persist         uefivarstore
apdp             imagefv_a       qmcs            userdata
apdpb            imagefv_b       qupfw_a         vbmeta_a
bluetooth_a      keymaster_a     qupfw_b         vbmeta_b
bluetooth_b      keymaster_b     qweslicstore_a  vbmeta_system_a
boot_a           keystore        qweslicstore_b  vbmeta_system_b
boot_b           limits          rawdump         vendor_boot_a
cdt              limits-cdsp     recovery_a      vendor_boot_b
connsec          logdump         recovery_b      vm-bootsys_a
cpucp_a          logfs           rtice           vm-bootsys_b
cpucp_b          mdcompress      secdata         vm-data
ddr              mdtp_a          shrm_a          xbl_a
devcfg_a         mdtp_b          shrm_b          xbl_b
devcfg_b         mdtpsecapp_a    splash          xbl_config_a
devinfo          mdtpsecapp_b    spunvm          xbl_config_b
dip              metadata        ssd             xbl_ramdump_a
dsp_a            misc            storsec         xbl_ramdump_b
dsp_b            modem_a         super           xbl_sc_logs
dtbo_a           modem_b         toolsfv         xbl_sc_test_mode
```

```
taro:/ # mount | grep dm- | head -n6
/dev/block/dm-2 on / type ext4 (ro,seclabel,nodev,relatime,discard)
/dev/block/dm-3 on /system_ext type ext4 (ro,seclabel,relatime,discard)
/dev/block/dm-1 on /product type ext4 (ro,seclabel,relatime,discard)
/dev/block/dm-4 on /vendor type ext4 (ro,seclabel,relatime,discard)
/dev/block/dm-5 on /vendor_dlkm type ext4 (ro,seclabel,relatime,discard)
/dev/block/dm-0 on /odm type ext4 (ro,seclabel,relatime,discard)
```

Loaded by the bootloader

boot	GKI core kernel image
vendor_boot	initramfs, first stage modules (e.g., UFS)
vendor_dlkm	Rest of the kernel modules (e.g., Wi-Fi)

Loaded by the Android’s init binary

In practice
Snapdragon 8 Gen 1 (2022)

First stage (vendor_boot)

```
# adb shell dmesg | grep 'init:'
[    5.085137] init: init first stage started!
[    5.090835] init: Loading module /lib/modules/qcom_hwspinlock.ko with args ''
[    5.106962] init: Loaded kernel module /lib/modules/qcom_hwspinlock.ko
...
[    6.914768] init: Loading module /lib/modules/msm-geni-se.ko with args ''
[    6.929909] init: Loaded kernel module /lib/modules/msm-geni-se.ko
[    6.936268] init: Loading module /lib/modules/msm_geni_serial.ko with args 'con_enabled=1'
[    6.958682] init: Loaded kernel module /lib/modules/msm_geni_serial.ko
[    6.965412] init: Loading module /lib/modules/ns.ko with args ''
[    6.971603] init: Loaded kernel module /lib/modules/ns.ko
...
[    7.867425] init: Loading module /lib/modules/bcl_pmic5.ko with args ''
[    7.883855] init: Loaded kernel module /lib/modules/bcl_pmic5.ko
[    7.890187] init: Loaded 100 kernel modules took 2800 ms
```

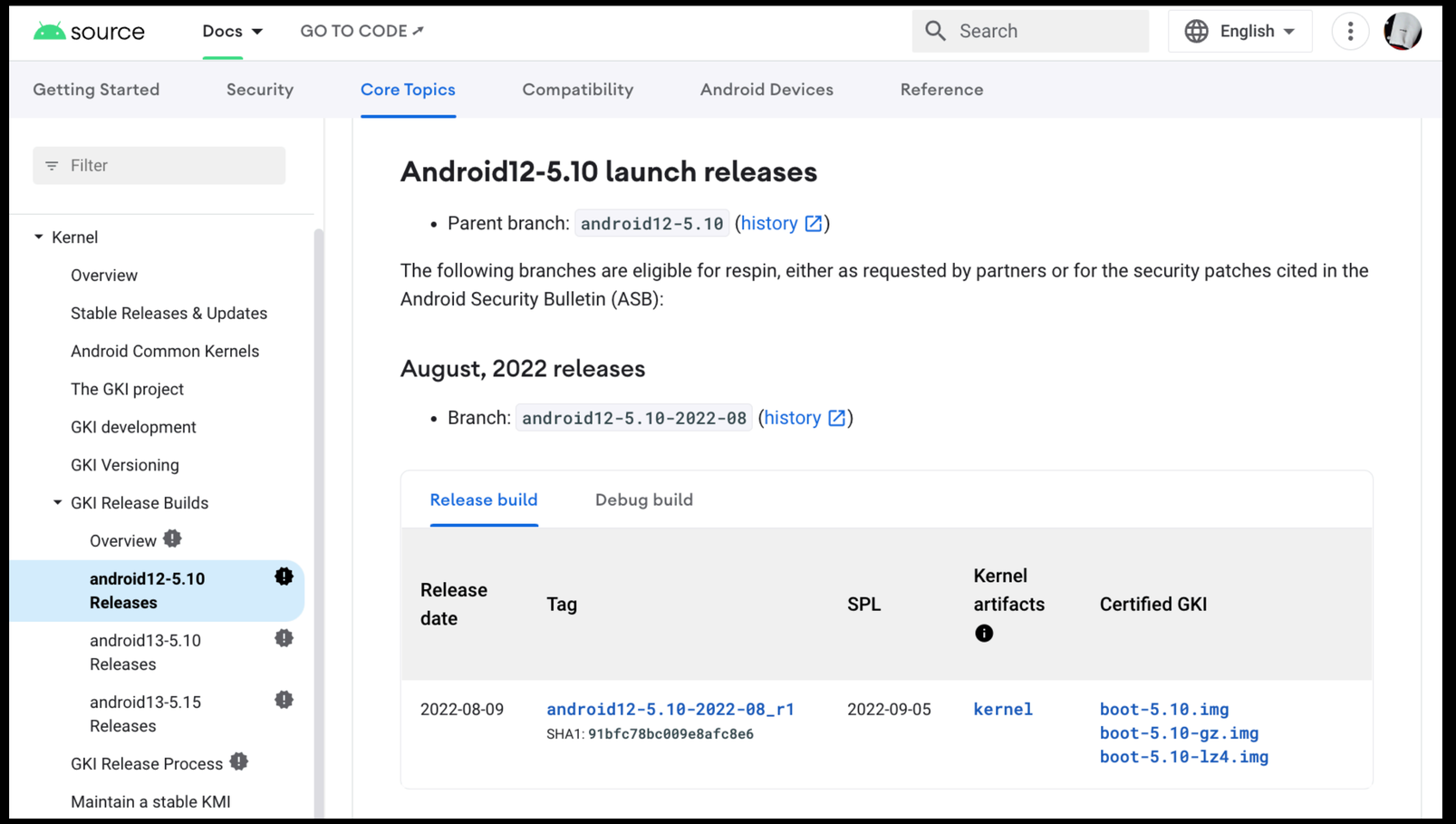
```
adb shell dmesg | grep 'modprobe:'
[   13.683976] modprobe: Loading module /vendor/lib/modules/mdt_loader.ko with args ''
[   13.702413] modprobe: Loaded kernel module /vendor/lib/modules/mdt_loader.ko
...
[   33.188493] modprobe: Loading module /vendor/lib/modules/msm_drm.ko with args
'dsi_display0=qcom,mdss_dsi_nt36672c_fhd_plus_90hz_video:'
...
[   41.771381] modprobe: Loaded kernel module /vendor/lib/modules/pinctrl_lpi_dlkm.ko
```

vendor_dlkm

```
taro:/ # find /vendor_dlkm/
/vendor_dlkm/
/vendor_dlkm/etc
/vendor_dlkm/etc/build.prop
/vendor_dlkm/etc/NOTICE.xml.gz
/vendor_dlkm/etc/fs_config_dirs
/vendor_dlkm/etc/fs_config_files
/vendor_dlkm/lib
/vendor_dlkm/lib/modules
/vendor_dlkm/lib/modules/adsp_loader_dlkm.ko
...
/vendor_dlkm/lib/modules/mmrm_test_module.ko
/vendor_dlkm/lib/modules/modules.alias
/vendor_dlkm/lib/modules/modules.blocklist
/vendor_dlkm/lib/modules/modules.dep
/vendor_dlkm/lib/modules/modules.load
/vendor_dlkm/lib/modules/modules.softdep
/vendor_dlkm/lib/modules/msm-cvp.ko
...
/vendor_dlkm/lib/modules/zsmalloc.ko
/vendor_dlkm/lost+found
```

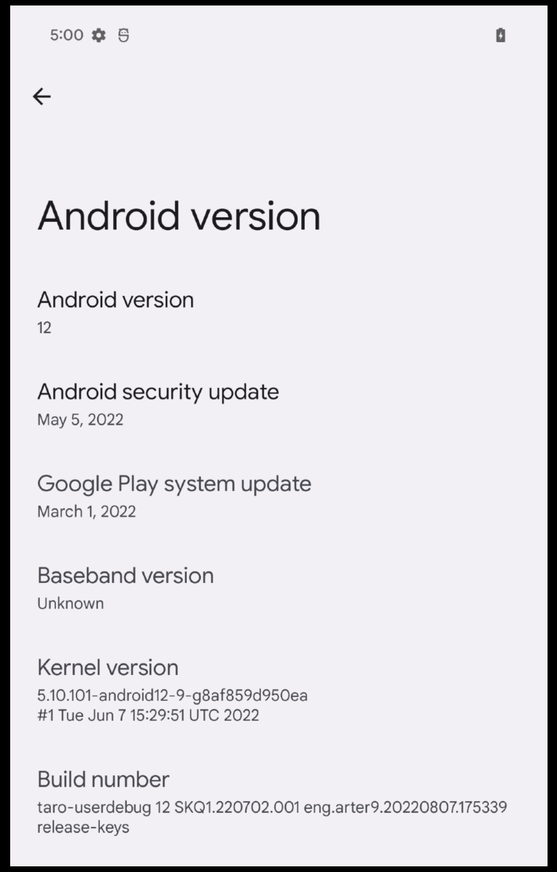
375 modules!

In practice
Using Google-provided GKI images

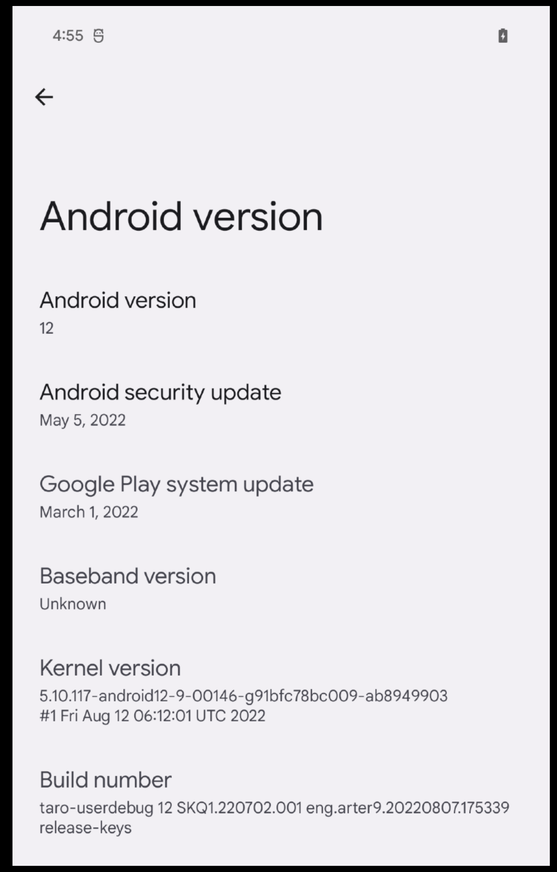


https://source.android.com/docs/core/architecture/kernel/gki-android12-5_10-release-builds

In practice
Using Google-provided GKI images

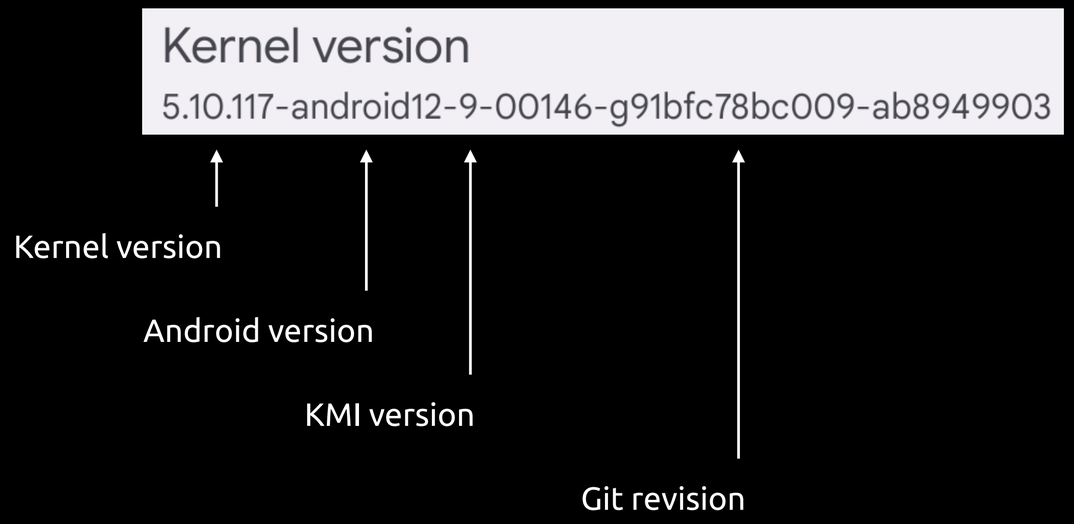


Stock kernel by Qualcomm



Google-provided GKI image

In practice
Using Google-provided GKI images

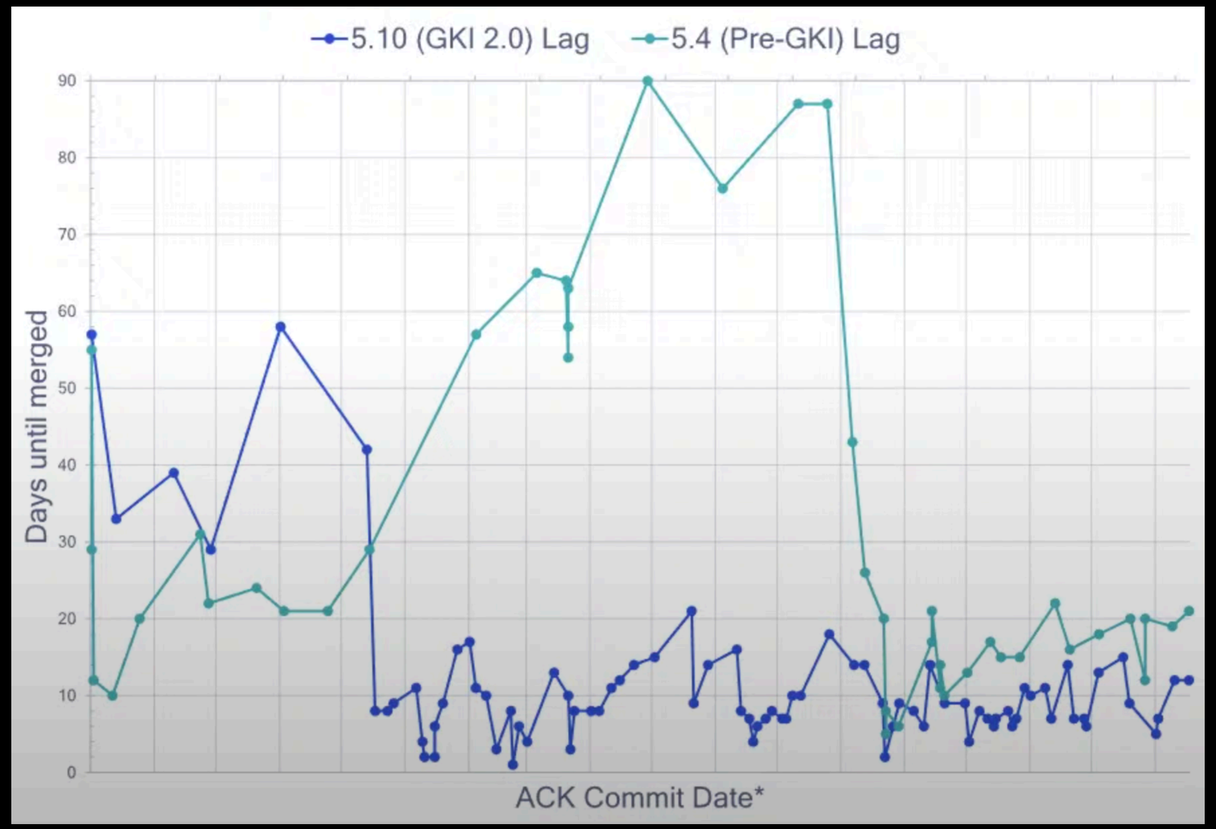


In practice
What GKI does and doesn’t

- GKI allows updating core kernel and vendor modules asynchronously
- GKI guarantees module compatibilities within the same KMI version
  - 5.10.y-android12-9
  - 5.10.y-android13-5
  - 5.15.y-android13-1
- GKI doesn’t guarantees stable ABIs across different major kernel versions
  - Upgrading a device from v5.10 to v5.15 is still a burden

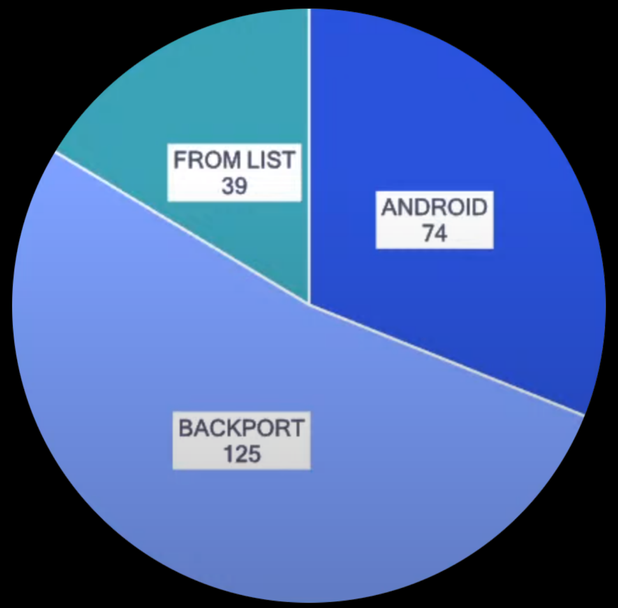
In practice
How effective is it?

- Security patches integrated quickly without any issues (e.g., Spectre-BHB)

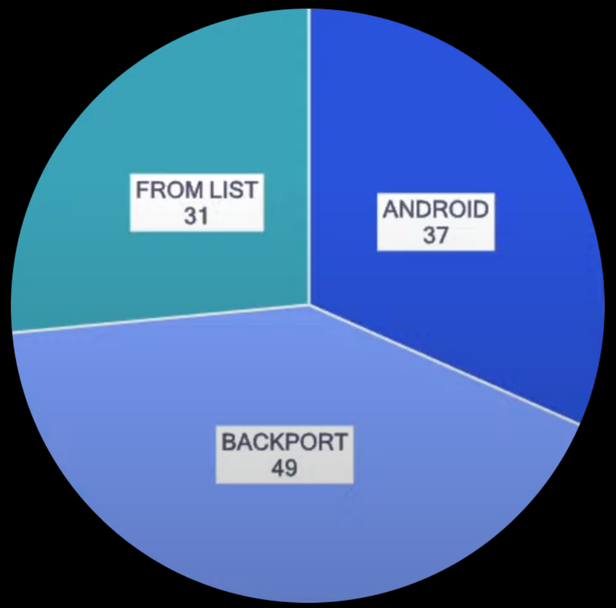


In practice
How effective is it?

android12-5.10 (238 commits)



android13-5.15 (117 commits)



GKI successfully reduces divergence from mainline

In practice
How effective is it?

- Major OEMs are not shipping Google’s GKI image
  - Upstreaming every value-adds and optimizations to Google is infeasible
- This means switching to Google’s GKI later via OTA is also infeasible
  - Imagine losing features in the name of enhanced security
- “One kernel image for all Android devices” is still left unrealized

Verdict

Current state of Android actively discourages modding

- 2011 - Close-sourced vendor modules
- 2013 - eFUSE
- 2014 - Unlocking bootloader breaks camera
- 2015 - Widevine L1
- 2017 - SafetyNet
- 2019 - R/O file-system by design, dynamic partitions
- 2021 - Re-introduction of kernel modules
- 2022 - GKI 2.0

Some things shouldn’t be this hard…

Not kernel-related, but still

- Obtaining root access on unlocked bootloader
  - Magisk became a very complicated piece of software
- Replacing system files
  - Recall that system partitions are R/O by design
    - Dedup-ext4, f2fs RO, EROFS
  - overlayfs is only available on debug firmware
- AOSP is too dependent on GMS to be usable

No one pressures OEMs for GPL

Not GKI-related, but still

- Major OEMs often take months to release the kernel’s source code
  - Even never
- Obviously without Git commit history
- Often times broken with partial source code
  - Compartmentalized modules made things even more complicated

GKI is a nightmare to tinkerers and hackers

Build environment

- GKI now requires OEMs to build kernel and modules within the GKI build environment
  - OEMs can fork the build environment
    - Build environment itself is not covered by GPL
  - Building for Pixels is easy
  - Building for anything else is not
    - No clear instructions given alongside the source code

GKI is a nightmare to tinkerers and hackers
LTO

- Changing a single line of source code takes **20 minutes** to recompile
  - On a 64-cores AMD Ryzen Threadripper 3990X
  - LTO takes too long
- Disabling LTO disables CFI
  - https://source.android.com/docs/security/test/kcfi
- Disabling CFI breaks module compatibility

GKI is a nightmare to tinkerers and hackers
Partition shenanigans

- Recall that you need to change 3 partitions
  - “boot” for the core kernel image
  - “vendor_boot” for the initial kernel modules
  - “vendor_dlkm” for the rest of the kernel modules
    - “vendor_dlkm” is inside the dynamic partition
    - Your device may even lack “vendor_dlkm”, which then you need to modify the “vendor” partition directly
- You better hope you got all of it correctly done to *just boot*

GKI is a nightmare to tinkerers and hackers

Monolithic kernel

- Disabling modules will produce a monolithic kernel
  - “This will make things simpler, right?”
- OEMs no longer test initcall orders
- Various modules will simply break because of orderings
  - X module may depend on Y module to be loaded prior
  - Z module may depend on a specific userspace file to be present
  - No logs available to debug

My first experience with GKI
HDK8450

- Snapdragon 8 Gen 1 Mobile HDK
  - Linux v5.10
- Google’s GKI worked nicely
  - But I wanted full-control over the kernel
- Thanksfully, Qualcomm’s environment is open-source
  - https://git.codelinaro.org/clo/la/kernel/manifest
  - Unfortunately, it didn’t build without proprietary BSP
  - Attempts to fix it to build doesn’t seem to be worth it

My first experience with GKI
HDK8450

- Monolithic kernel didn’t boot
  - Expected, but still sad :(

- Patched “vendor_boot” to enable earlycon
  - Got minicom to display kernel log!
- A whole bunch of initcalls causing kernel panics
  - Fixing it one-by-one for all 375 modules? No thanks.

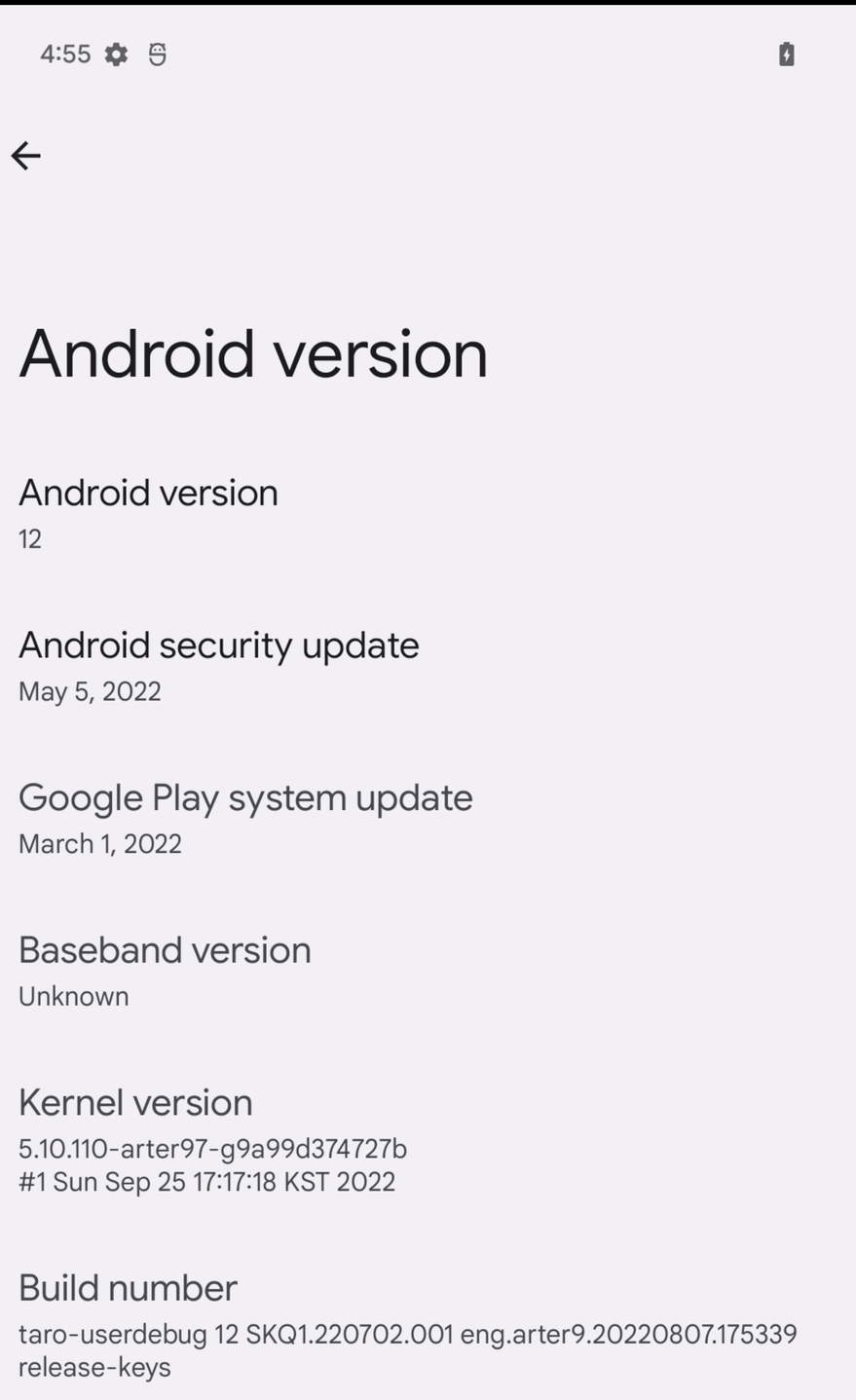
Lazy initcall
Allowing monolithic kernel

- Idea: defer initcalls until userspace asks for the corresponding module
- Mark device drivers to defer its initialization and add to the list
- Modify the init_module(2) system-call
  - Get the module name from the requested .ko file
  - Call the corresponding device driver’s init() function
- https://github.com/arter97/hdk-kernel/commit/e0085cde67a4
- https://github.com/arter97/hdk-kernel/commit/c425693e00d1

Lazy initcall

Allowing monolithic kernel

```
[    5.089408] init: init first stage started!
[    5.095136] init: Loading module /lib/modules/qcom_hwspinlock.ko with args ''
[    5.102852] lazy_initcall: lazy_initcalls[13]: qcom_hwspinlock's init function returned 0
[    5.111263] init: Loaded kernel module /lib/modules/qcom_hwspinlock.ko
[    5.117974] init: Loading module /lib/modules/smem.ko with args ''
[    5.124460] lazy_initcall: lazy_initcalls[21]: smem's init function returned 0
[    5.131888] init: Loaded kernel module /lib/modules/smem.ko
[    5.137643] init: Loading module /lib/modules/minidump.ko with args ''
[    5.146545] Minidump: Enabled with max number of regions 200
[    5.152528] lazy_initcall: lazy_initcalls[110]: minidump's init function returned 0
[    5.160410] init: Loaded kernel module /lib/modules/minidump.ko
```



Easily the hardest device I ever had to work on to *just boot*

Custom build environment

- Google’s GKI build environment - 20 minutes
- Full build - 50 seconds
- Full build with ccache - 17 seconds
- Incremental build - 15 seconds

- AMD Ryzen Threadripper 3990X (64-cores)

Few ideas for improvements…

🙏

- Properly mandate GPL
- Verify the released kernel source and environment properly boots and passes tests
- Verify CONFIG_MODULES=n passes tests
  - Tinkerers don’t need kernel modules
  - Too many things break on monolithic kernels
- Standardize and mandate a way to retrieve kernel panic logs
  - Some are straight-up unavailable on production devices
  - pstore-blk is now available upstream

Q & A