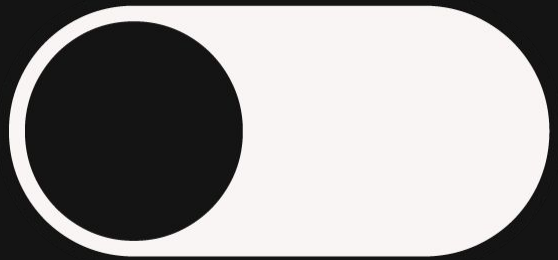
Build a Fake Phone Find Real Bugs

Qualcomm GPU Emulation and Fuzzing with LibAFL QEMU



Context

Romain Malmain

- PhD student at EURECOM
- Work done during a 3-month internship this summer

Scott Bauer

Qualcomm Product Security

- Technical Team Lead – PSIRT
- Internship technical mentor

What’s the goal?

A modern smartphone attack surface

- Gigantic attack surface (Android is ~2M LoC)
- The security scope goes beyond the Android kernel

Many bugs reported every year by CMs

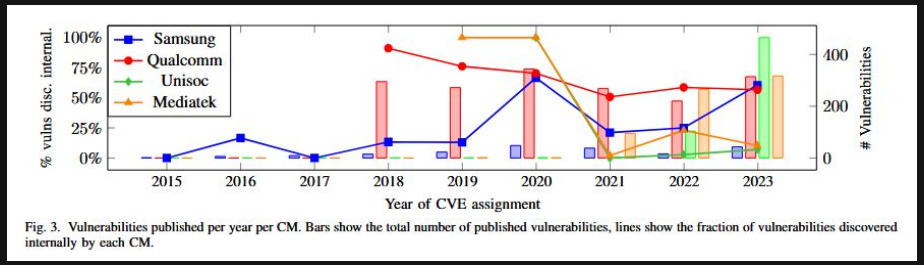
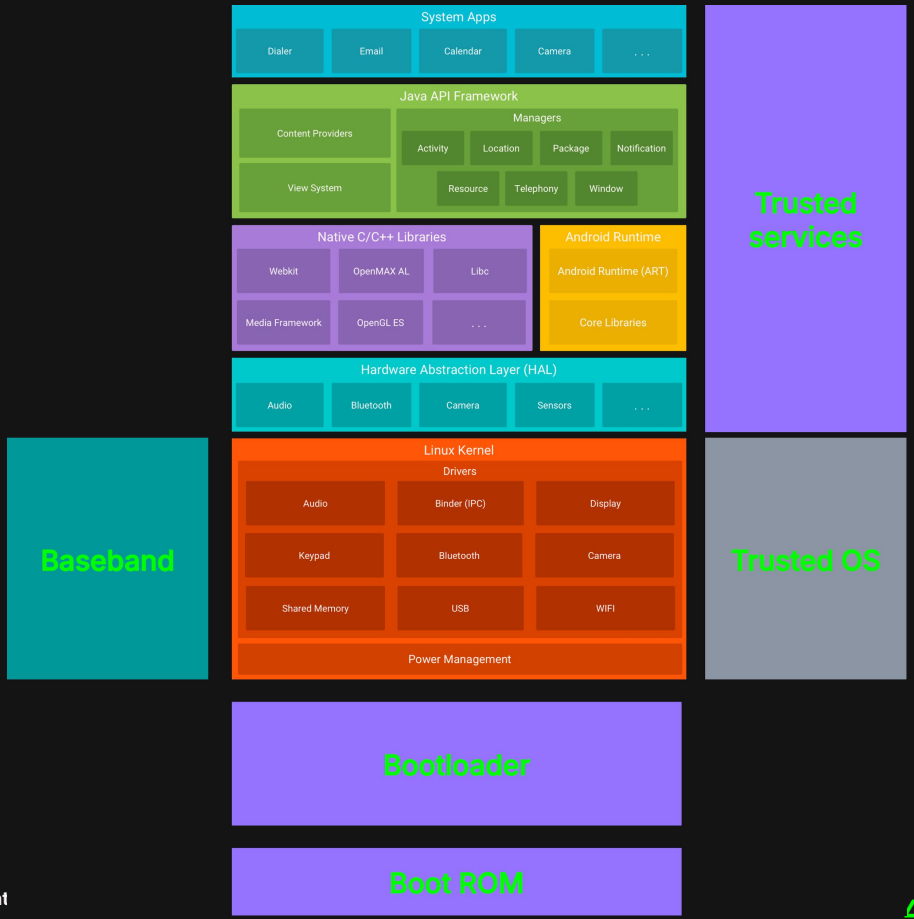
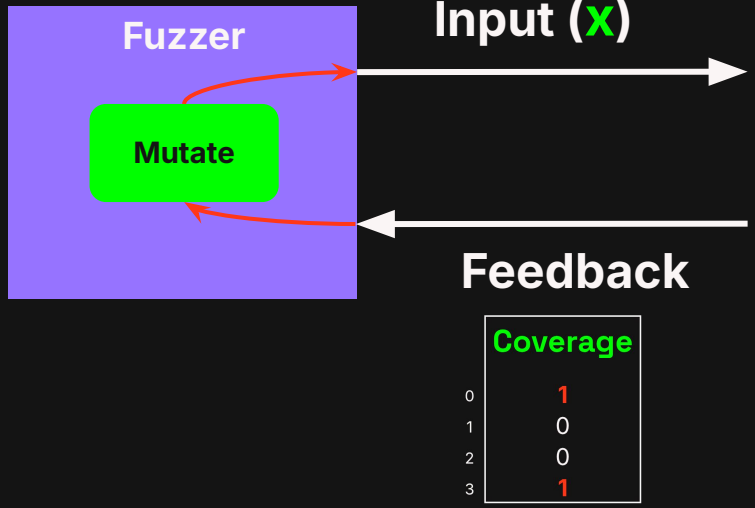


Fig. 3. Vulnerabilities published per year per CM. Bars show the total number of published vulnerabilities, lines show the fraction of vulnerabilities discovered internally by each CM.

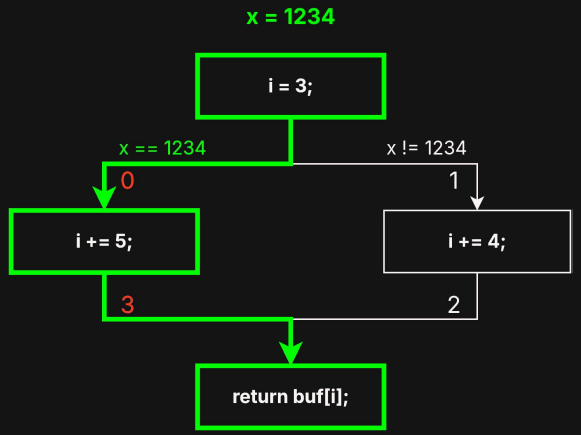


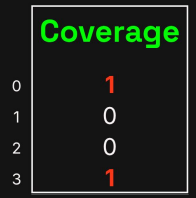
Source: D. Klischies et al. **Vulnerability, Where Art Thou? An Investigation of Vulnerability Management in Smartphone Chipsets**. NDSS 2025

Yet another fuzzing 101 slide



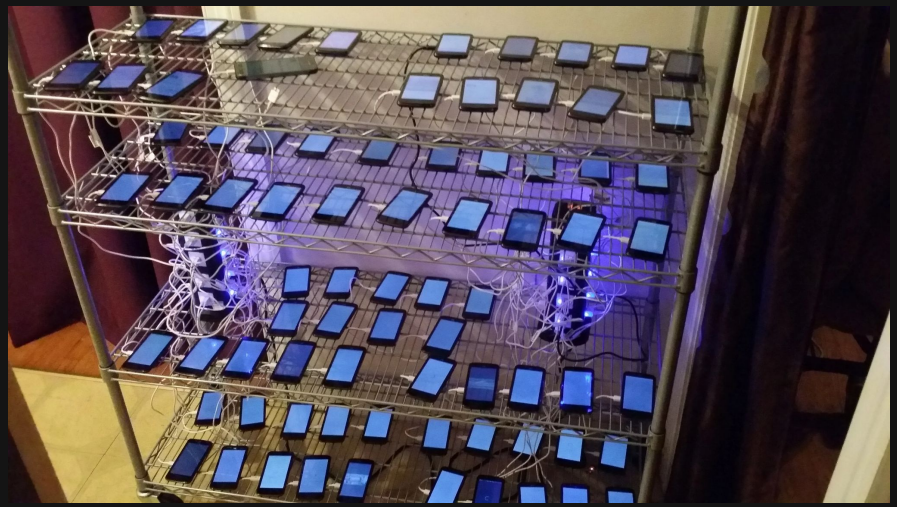

```
char buf[8];

char fuzzing_target(int x)
{
    size_t i = 3;

    if (x == 1234) {
       i += 5
    } else {
       i += 4;
    }

    return buf[i];
}
```





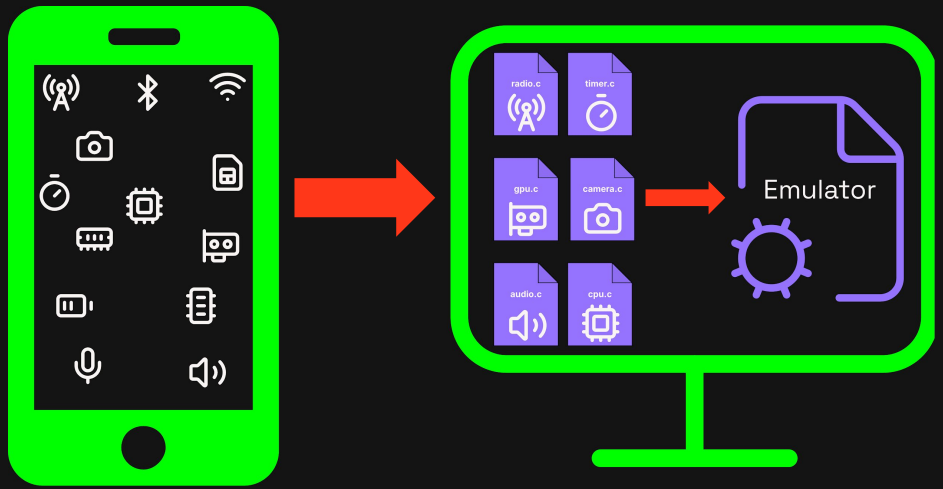
On-device fuzzing limitations

- Does **not scale**
- **Expensive**
- What if the **hardware** gets **updated**?
- Can make **debugging harder**
  - JTAG used for kernel / firmware debugging
- **Crash detection** and **handling**
  - reset / reboot the phone
  - sometimes manually via **POWER CYCLES**
- Hard to **integrate** in a **CI pipeline**

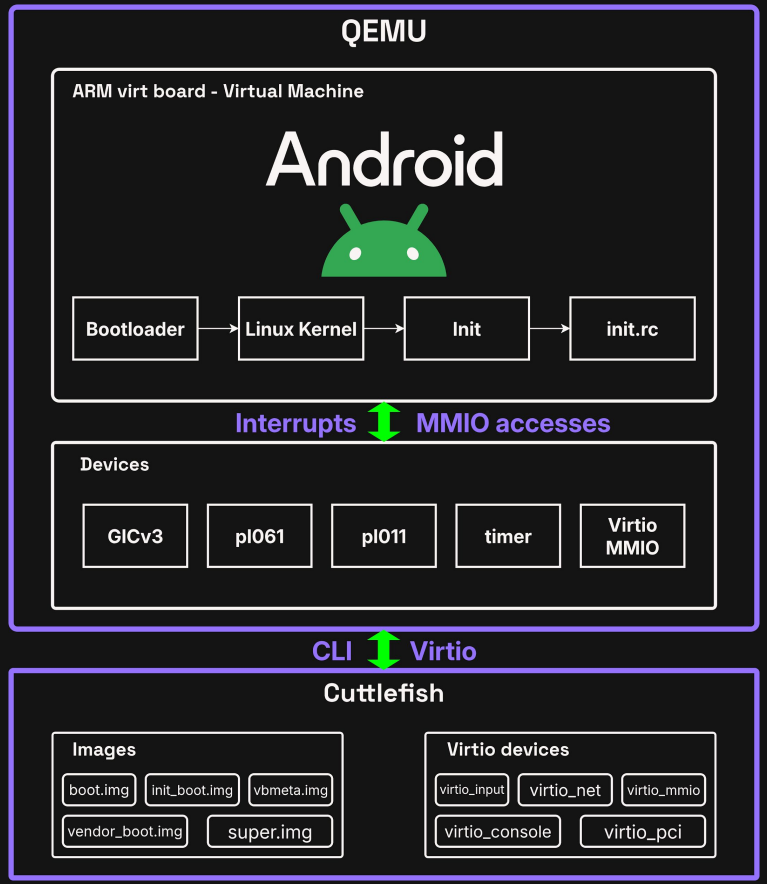


Emulation 101

- The emulated machine (i.e. the phone) is called the **guest**
- **Emulation** — run the guest software on another computer (the **host**)
- Reproduce the **environment** of the guest
  - **CPU**, **MMU**, **memory layout**, Interrupt controller, etc.
  - **Devices** — **Timers**, **video**, audio, radio, etc.



QEMU Android virtualization stack in 2025



Cuttlefish reproduces a **realistic virtual environment** to run **Android**

- Supports **multiple emulators / hypervisors** (**crosvm** and **QEMU**)
- **Does not require to maintain a separate fork**
- Support for **ADB**

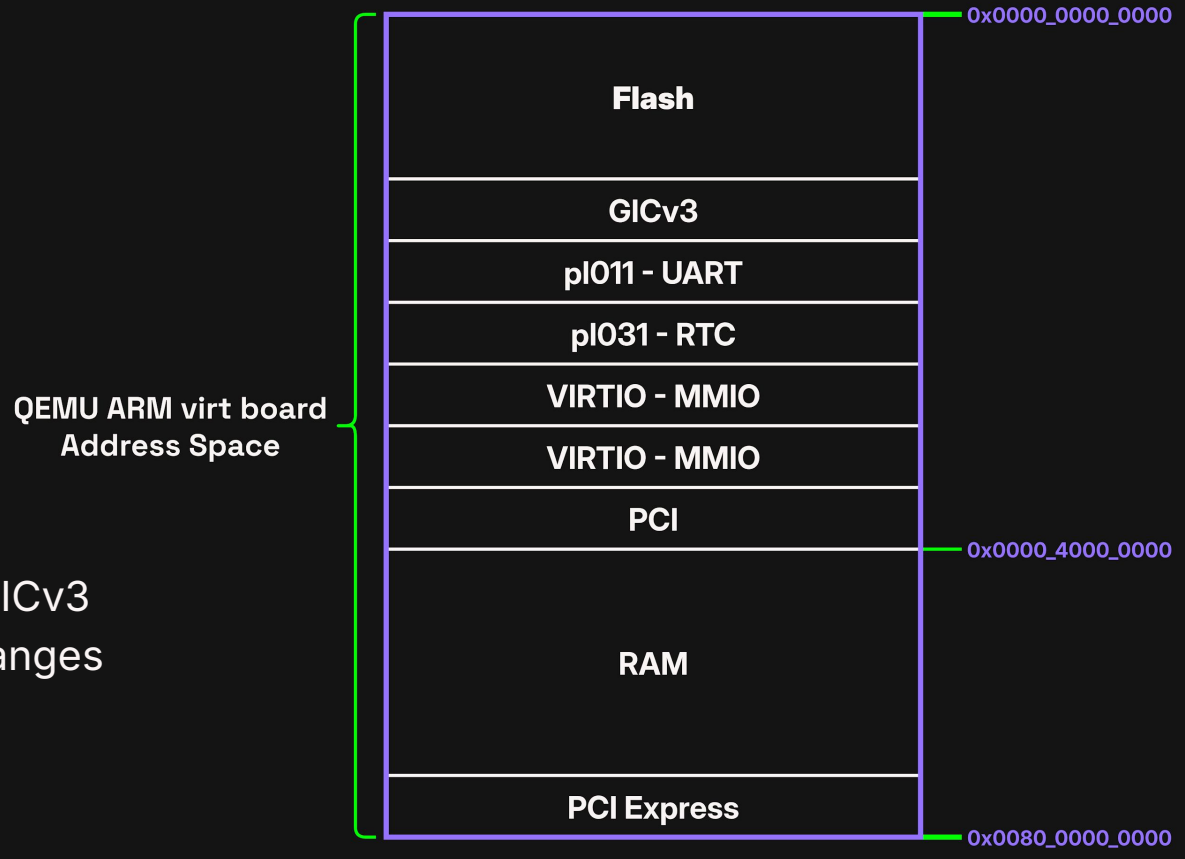
Cuttlefish harnesses **QEMU**

- Creates **QCOW2 disk overlays**
- Handles a bunch of **Virtio devices**
- Feeds **the right parameters to QEMU** (quite complex!)
- QEMU runs the ARM **virt board** and the VM boots with the **u-boot** bootloader.

ARM virt board — physical address space

QEMU Machine Specification

- **CPU** — Cortex-A57
- **RAM**
- **UART** — pl011
- **RTC** — pl031
- **PCI**
- **PCI-express**
- **Interrupt controller** — GICv3
- A bunch of **Virtio MMIO** ranges



The plan

Find a way to emulate **latest Qualcomm's GPU kernel driver** in QEMU

- Design a **new Snapdragon board** based on the ARM virt board
- **Boot** the Android kernel
- Implement the **necessary QEMU devices** along the way
- **Start** and **interact** with the **GPU kernel module**

Merge our changes with **LibAFL QEMU**

- Out-of-the-box support for **modern VM fuzzing stuff**
  - **Command system** between host and guest
  - **Blazing fast** snapshots
  - Usual mutators, **cmplog**, etc.
  - Harnessing library
- Close to **latest QEMU version**
  - Will work with the latest Cuttlefish changes
  - Support for modern ARM64 hardware

Simple, right?

The DeviceTree specification

The **DeviceTree (DT)** specifies how to **describe hardware in an abstract way**.

- It is used by the software (mostly **kernels** and **bootloaders**) to be **aware of the hardware it is running on**.
- The **target hardware** (for us, the final phone) is represented in a **tree structure**

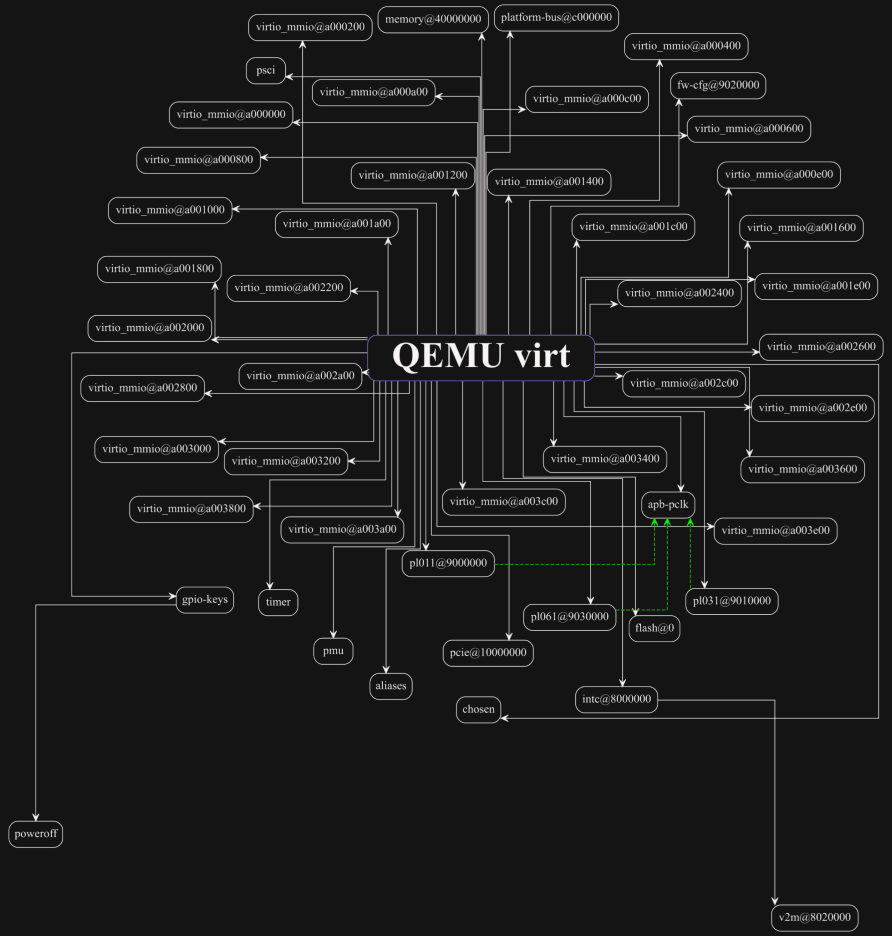
Each **node** represents a **device**, and hold information about it, called **properties:**

- The **MMIO address ranges**
- The **interrupt lines**
- Links to other devices

The DeviceTree is saved in a binary format called a **DeviceTree Binary (DTB)**

➔ This is what is **exported in the final image**, to be parsed by the phone

QEMU ARM virt board — DTB visualization



fdtviz is a tool generating a **graph representation** from a **DTB**

- useful to visualize the **size**, the **complexity,** and the **node relationship** of a **DT**
- **root node** — running board name, **placed at the middle for better readability**
- **fdtviz** makes phandle references apparent, which is usually **hard to observe manually**

white edge **parent ➔ child** relationship

- A ➔ B means that **A owns B**

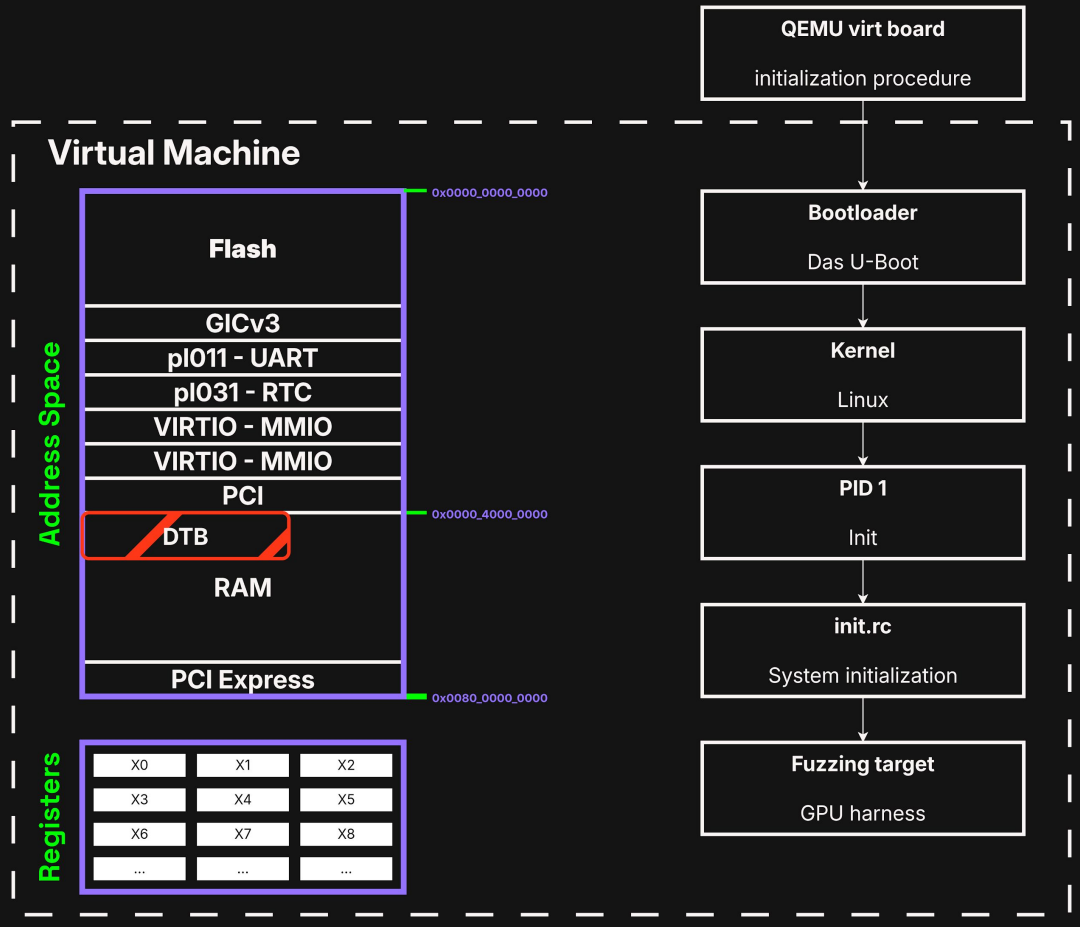
tree — a child can only have **one parent**, **no cycle**

green edge **phandle (“pointer”) relationship**

A ➔ B means that **A has a reference to B**

They are only references, **phandle cycles can exist**

QEMU ARM virt board — Specification



Hardware configuration information for bare-metal programming

The virt board automatically generates a device tree blob ("dtb") which it passes to the guest. This provides information at

- Flash memory starts at address 0x0000_0000
- RAM starts at 0x4000_0000

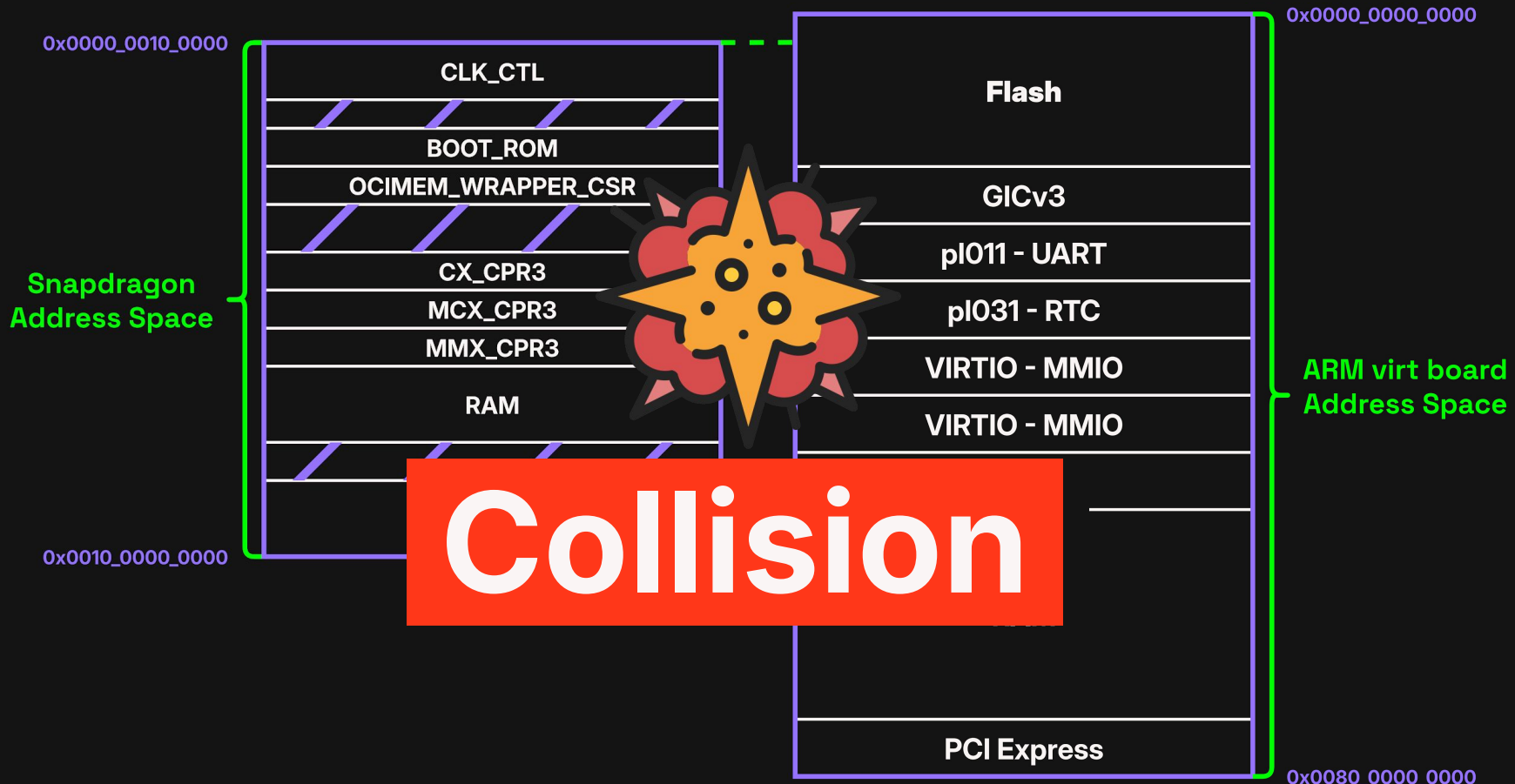
All other information about device locations may change between QEMU versions, so guest code must look in the DTB.

QEMU supports two types of guest image boot for virt, and the way for the guest code to locate the dtb binary differs:

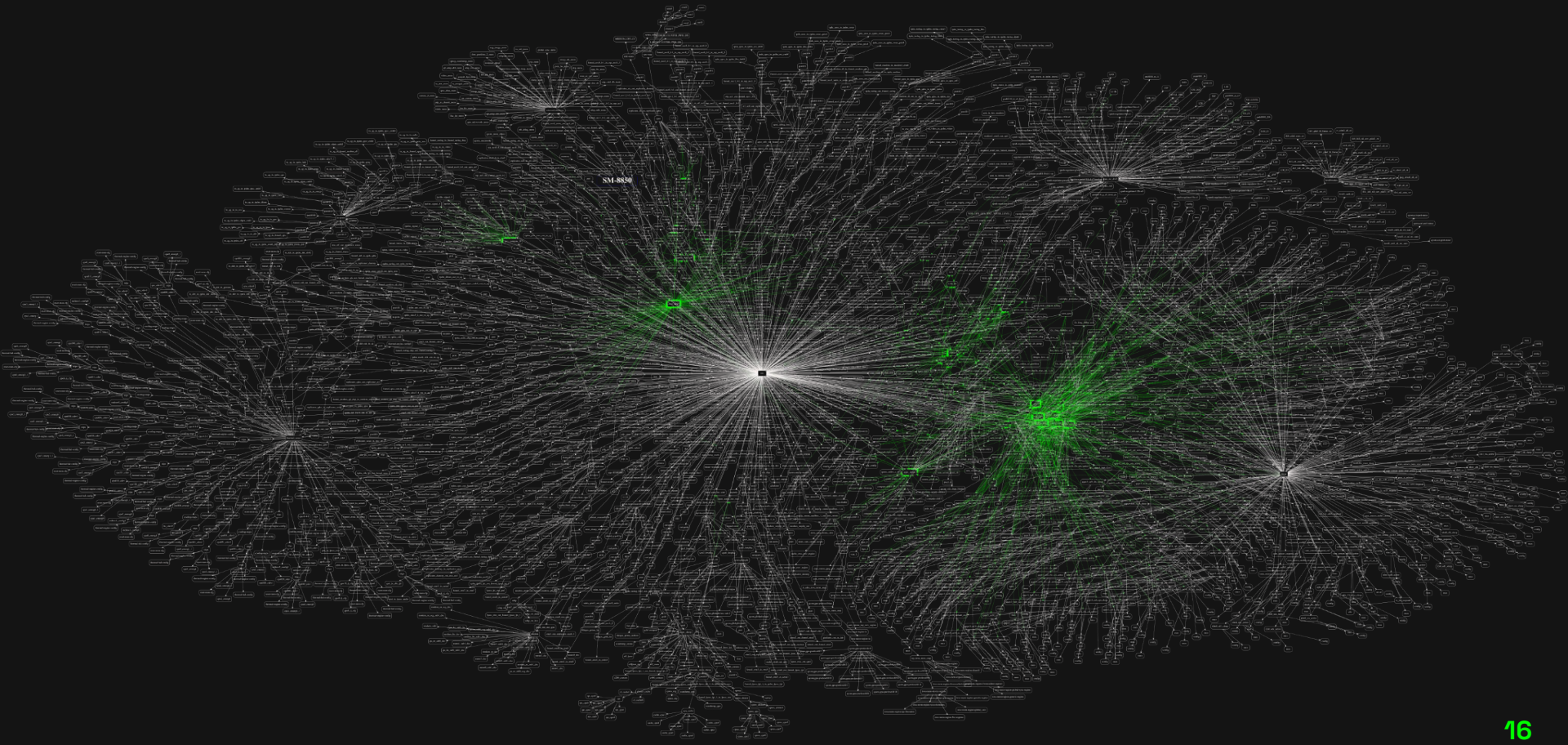
- For guests using the Linux kernel boot protocol (this means any non-ELF file passed to the QEMU -kernel option) the
- For guests booting as "bare-metal" (any other kind of boot), the DTB is at the start of RAM (0x4000_0000)

```
Before jumping into the kernel, the following conditions must be met:

- Quiesce all DMA capable devices so that memory does not get
  corrupted by bogus network packets or disk data.  This will save
  you many hours of debug.

- Primary CPU general-purpose register settings:

    - x0 = physical address of device tree blob (dtb) in system RAM.
    - x1 = 0 (reserved for future use)
    - x2 = 0 (reserved for future use)
    - x3 = 0 (reserved for future use)
```

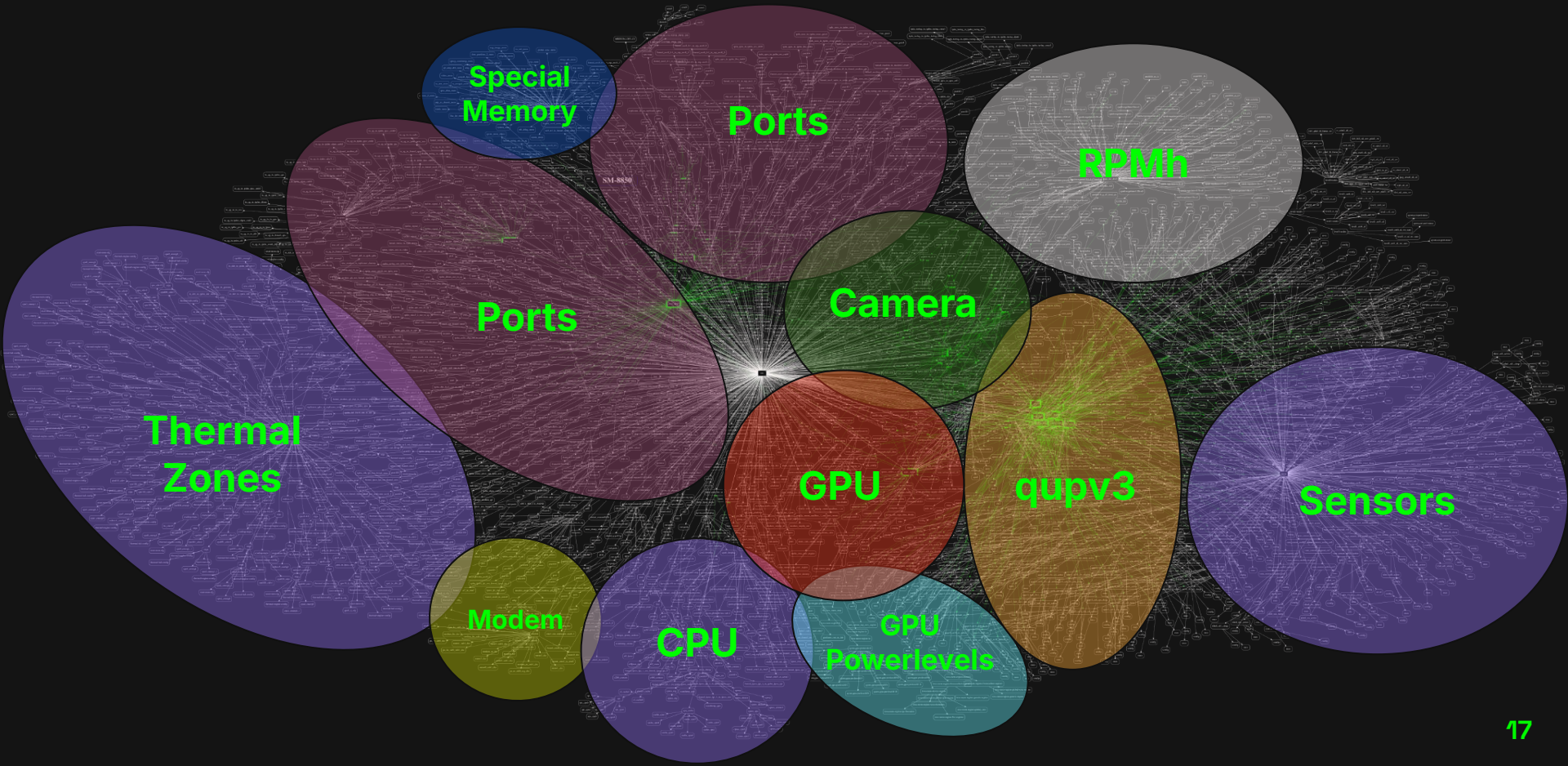
First problem — QEMU ARM virt board + Snapdragon



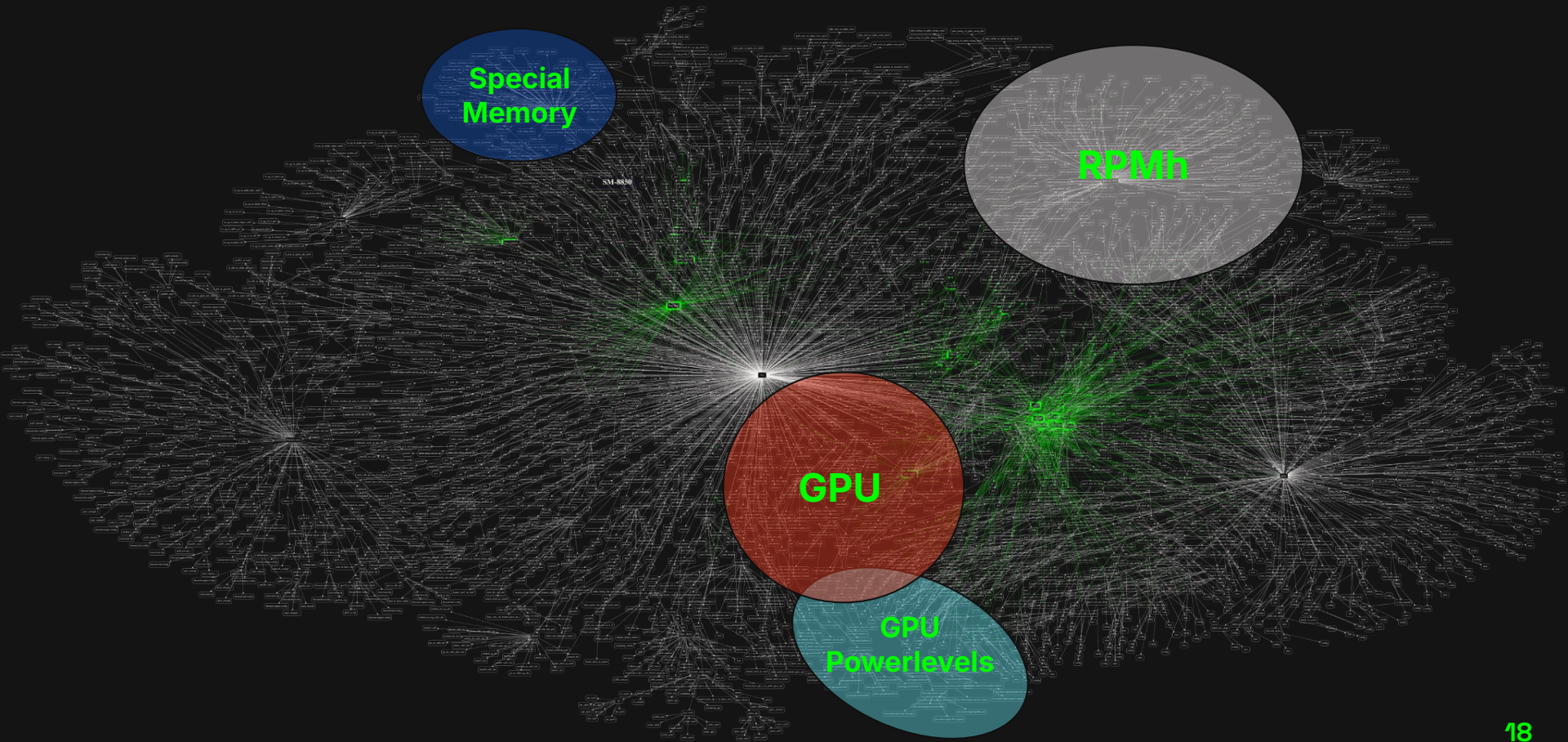
Second problem — how many devices already?



Snapdragon DeviceTree Overview



What we will focus on



Hacking into the Android build system

Build system — what needs to be adapted

Linux Kernel configuration tweaks

- `CONFIG_VIRTIO_*` / `CONFIG_BATTERY_GOLDFISH` → enable
- CONFIG_CFG80211*
- KCOV / KASAN for fuzzing
- → ~125 additional kernel modules added to the image

Disable some modules

- The Gunyah hypervisor
- Synx
- KASLR

Remove some code (mostly to save time)

- Zap shaders
- Fuse read / write
- LLCC

Init patches

- Remove SELinux support
- Disable most RC scripts

What happens in practice

```
==========================================
[build.sh]: Command: "python -B /local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core
-utils/build/commonsys_intf_checker.py"
==========================================
Traceback (most recent call last):
  File "/local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core-utils/build/commonsys_i
ntf_checker.py", line 370, in <module>
    main()
  File "/local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core-utils/build/commonsys_i
ntf_checker.py", line 364, in main
    start_commonsys_intf_checker()
  File "/local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core-utils/build/commonsys_i
ntf_checker.py", line 323, in start_commonsys_intf_checker
    get_commonsys_intf_project_from_manifest()
  File "/local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core-utils/build/commonsys_i
ntf_checker.py", line 159, in get_commonsys_intf_project_from_manifest
    if "qc-common-sys-intf" in groups:
TypeError: argument of type 'NoneType' is not iterable

real    0m1.359s
user    0m0.890s
sys     0m0.289s
==========================================
[build.sh]: FAILED: python -B /local/mnt/workspace/Projects/adreno-qemu/android-qualcomm-fresh/vendor/qcom/opensource/core-u
tils/build/commonsys_intf_checker.py
==========================================

real    162m13.846s
user    2020m31.987s
sys     207m20.145s
```

```
./Android.bp
./build/kgsl_defs.bzl
./build/target_variants.bzl
./config/[redacted].conf
./config/[redacted].conf
./config/gki_bengal.conf
./config/gki_blair.conf
./config/gki_kalama.conf
./config/gki_khajedisp.conf
./config/gki_kona.conf
./config/gki_[redacted].conf
./config/gki_[redacted].conf
./config/gki_neo.conf
./config/gki_niobe.conf
./config/gki_pineapple.conf
./config/gki_pitti.conf
./config/gki_qcs405.conf
./config/gki_qcs605.conf
./config/gki_sa8155.conf
./config/gki_scuba.conf
./config/gki_trinket.conf
./config/gki_waipiodisp.conf
./config/anorak_consolidate_gpuconf
./config/anorak_gki_gpuconf
./config/blair_consolidate_gpuconf
./config/blair_gki_gpuconf
./config/canoe_consolidate_gpuconf
./config/canoe_perf_gpuconf
./config/[redacted]_consolidate_gpuconf
./config/[redacted]_gki_gpuconf
./config/[redacted]_perf_gpuconf
./config/neo-la_consolidate_gpuconf
./config/neo-la_gki_gpuconf
./config/niobe_consolidate_gpuconf
./config/niobe_gki_gpuconf
./config/parrot_consolidate_gpuconf
./config/parrot_perf_gpuconf
./config/pineapple_consolidate_gpuconf
./config/pineapple_gki_gpuconf
./config/pineapple_perf_gpuconf
./config/vienna_consolidate_gpuconf
./config/vienna_perf_gpuconf
./Kbuild
./Kconfig
./Makefile
```

Soong

Bazel

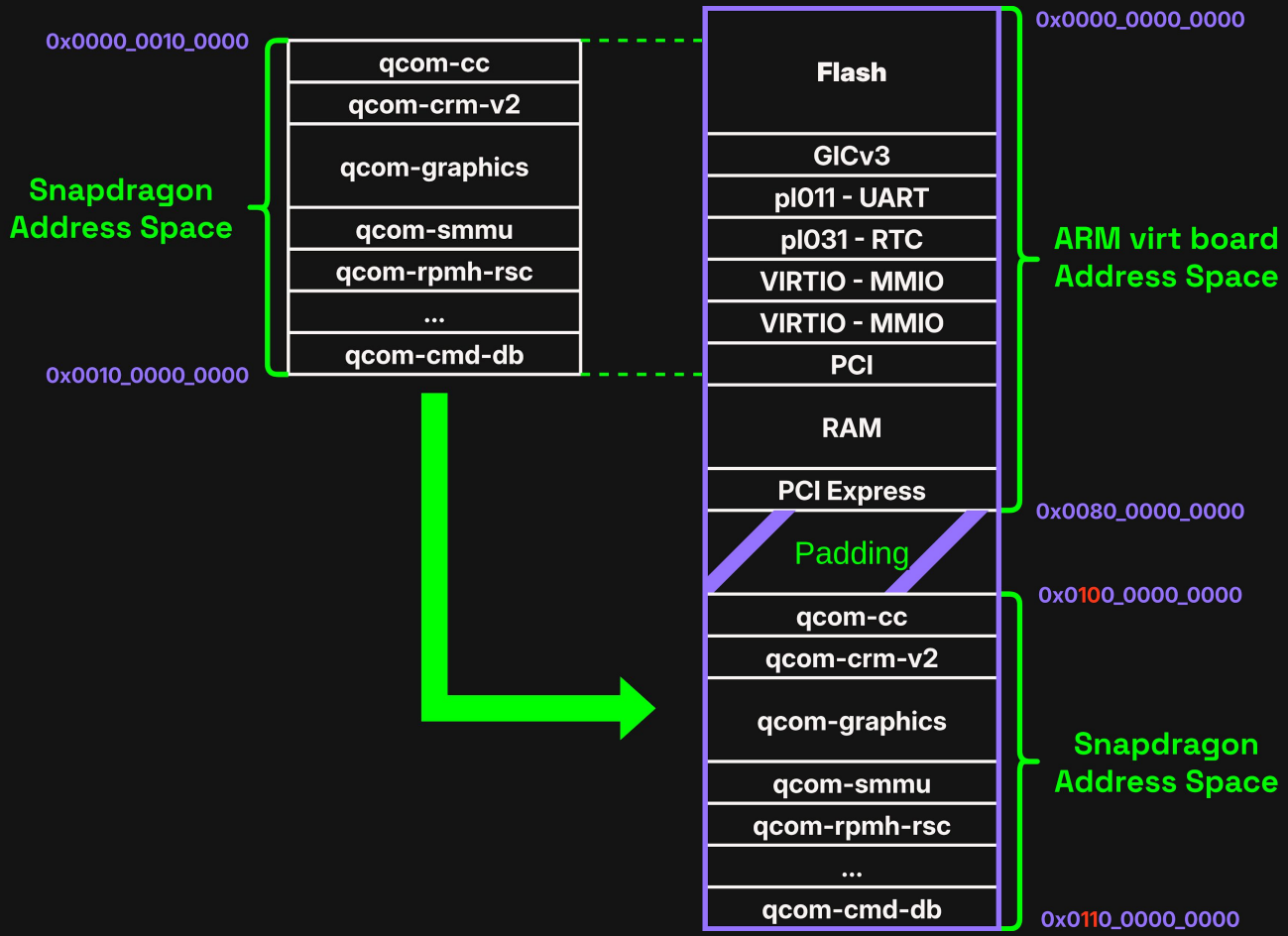
Kbuild

Make

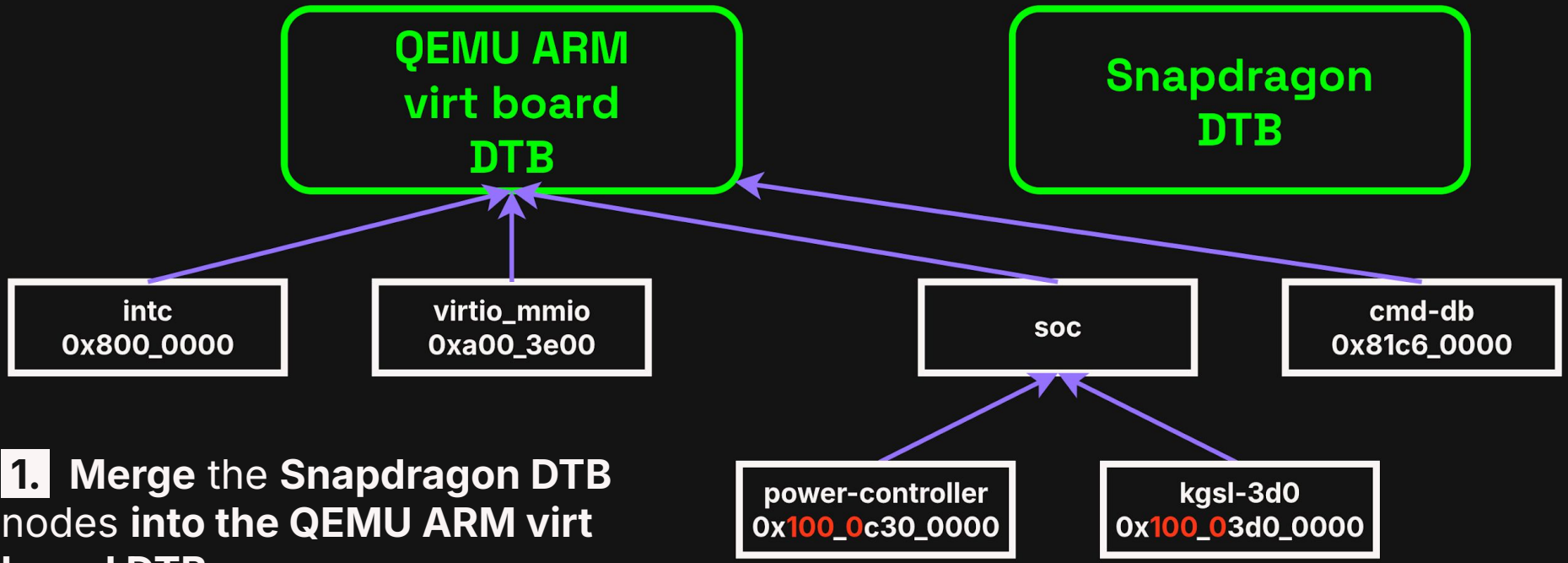
The preferred name for the kbuild files are `Makefile` but `Kbuild` can be used and if both a `Makefile` and a `Kbuild` file exists, then the `Kbuild` file will be used.

Snapdragon integration in QEMU

Handling address spaces collision



Merging Devicetree Blobs (DTB)



1. **Merge** the **Snapdragon DTB** nodes **into the QEMU ARM virt board DTB**

2. **Relocate** the addresses recursively to **fit the new address space**

sysbus-of — QEMU device blueprint using DTB

The **DeviceTree** contains **a lot of information** about devices

- The **MMIO** address ranges
- The **interrupt** structure
- The **links to other devices**
- The **compatibility string**
- etc...

sysbus-of is a generic QEMU object built using these information

- **parses** the device information
- **prefills** the internal **structure**
- helps **resolving device links** (phandle)

```
struct OfSysBusDevice {
    /*< private >*/
    SysBusDevice parent_obj;
    /*< public >*/

    const char* name;

    /* <snip> */

    // base address, as extracted during init.
    // if NULL, no base address was found.
    hwaddr* base_addr;

    // registers
    //
    // addresses are normalized at 0.
    // in other words, the first register always starts at 0
    struct fdt_reg* regs;
    uint32_t nb_regs;

    // interrupts
    // the parent must plug interrupts accordingly.
    struct fdt_interrupts* interrupts;
};
```

Towards GPU support in QEMU

What could go wrong?

Infinite loops

- **Easy case** — wait for a value with a timeout, return an error if the timeout triggers
- **More tricky** — the driver waits for a device value forever

```
/* Wait forever for a free tcs. It better be there eventually! */
wait_event_lock_irq(drv->tcs_wait,
                    (tcs_id = claim_tcs_for_req(drv, tcs, msg)) >= 0,
                    drv->lock);
```

No "compatible" device found in the DTB

- The drive **probes** for compatible device
- It is **specified in the DTB**
- If not found, the driver is **silently ignored**.

```
static const struct of_device_id adreno_match_table[] = {
    { .compatible = "qcom,kgsl-3d0", .data = &device_3d0 },
    { },
};
```

Kernel module deferral

- If the module needs to wait for another kernel module, returns `-EPROBE_DEFER`
- The kernel will **try to re-probe the module later**

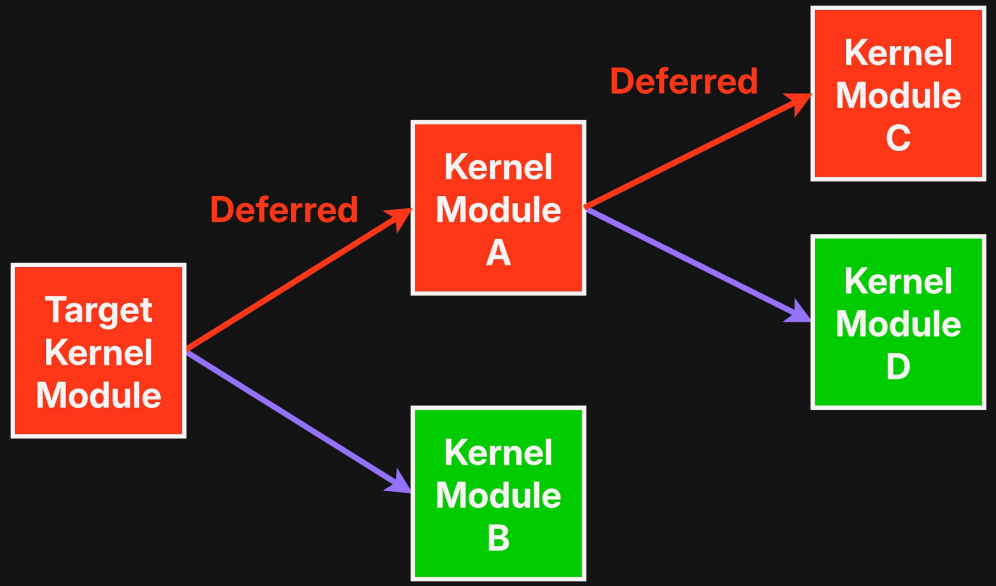
> Optionally, probe() may return -EPROBE_DEFER if the driver depends on resources that are not yet available (e.g., supplied by a driver that hasn't initialized yet). The driver core will put the device onto the deferred probe list and will try to call it again later. If a driver must defer, it should return -EPROBE_DEFER as early as possible to reduce the amount of time spent on setup work that will need to be unwound and reexecuted at a later time.

The deferred mess

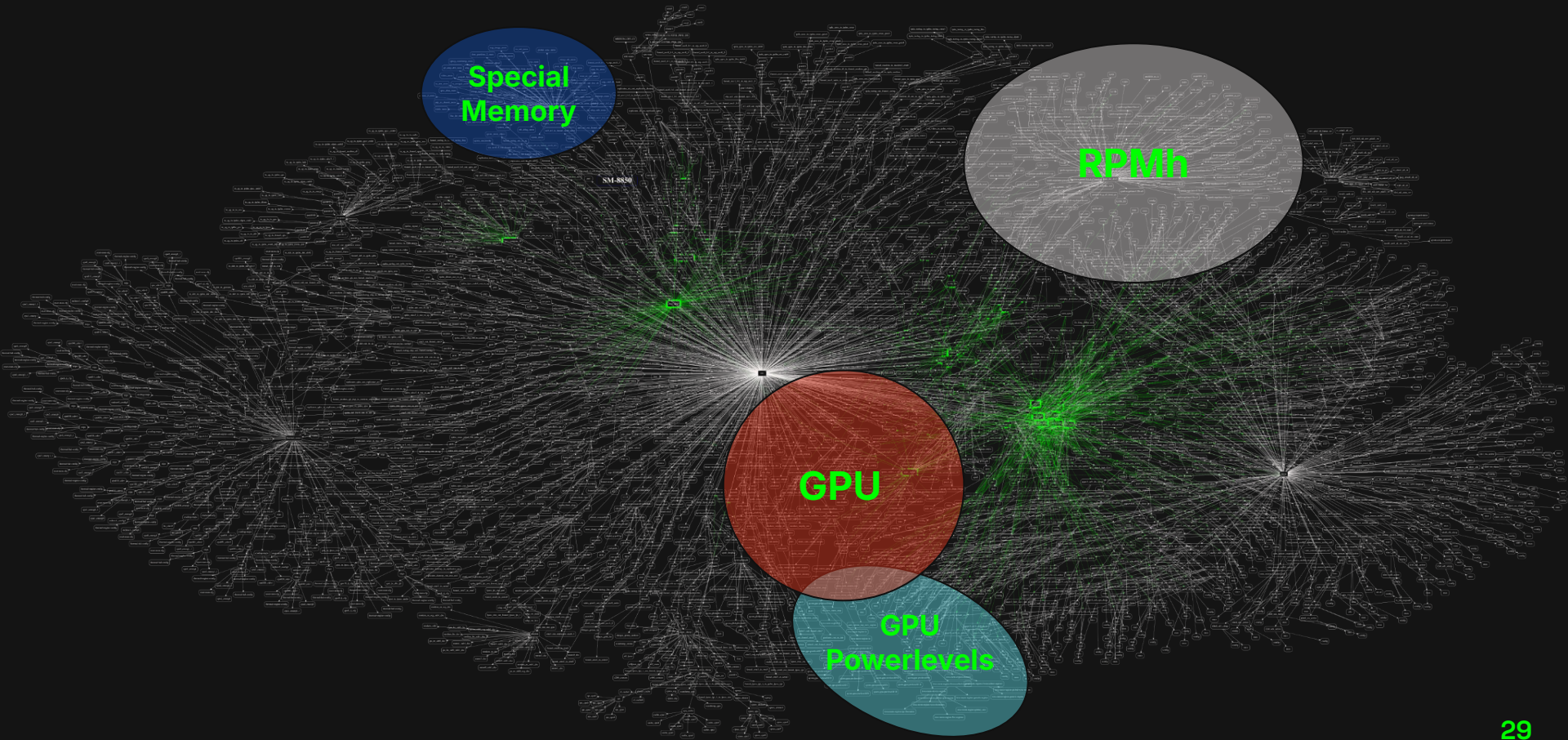
Deferral is much harder to deal with for reverse engineers

- Mostly happens **without any clue** on **how**, **where** and **why** it happens
- Deferrals are usually **chained** (see below for an example)
- It can **happen** even on **fully working systems** (mostly to wait for another driver)

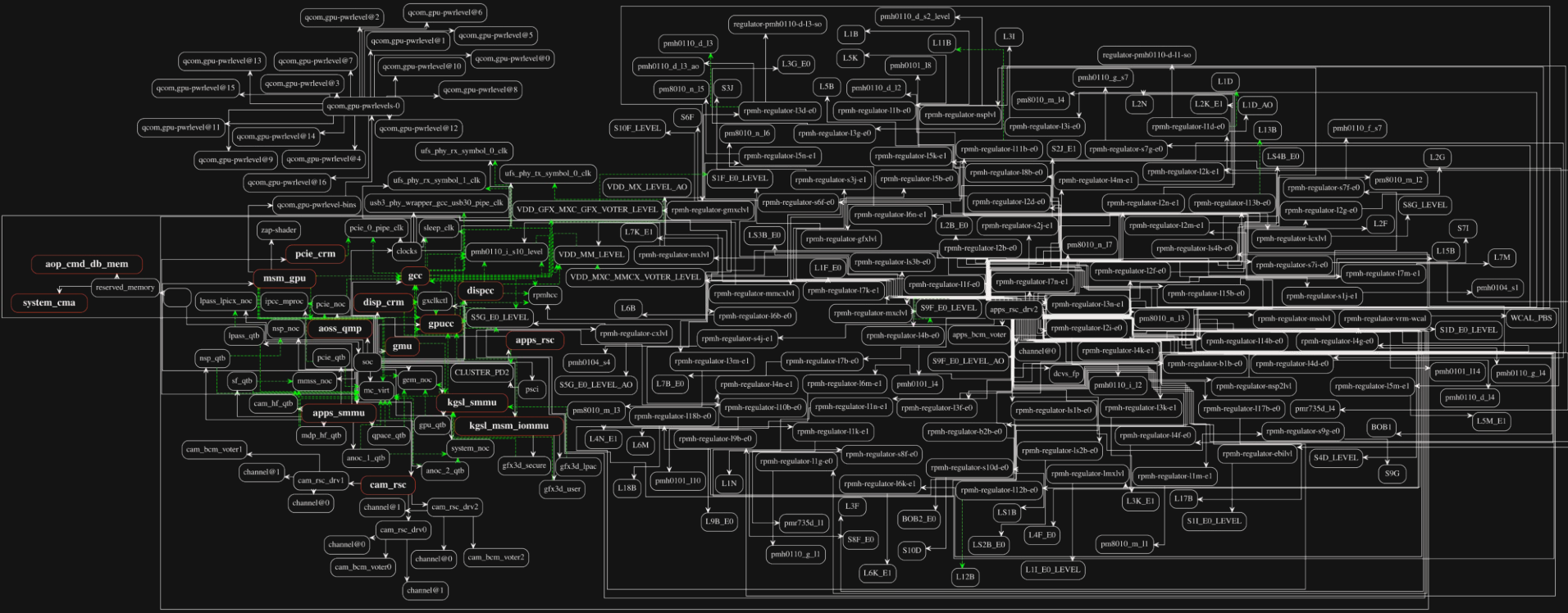
```
rmalmain@salnxbldlintlv001 /l/m/r/a/android (main)> rg -q --stats "\-EPROBE_DEFER" kernel_platform/common kernel_platform/soc-repo vendor/qcom/

2265 matches
```



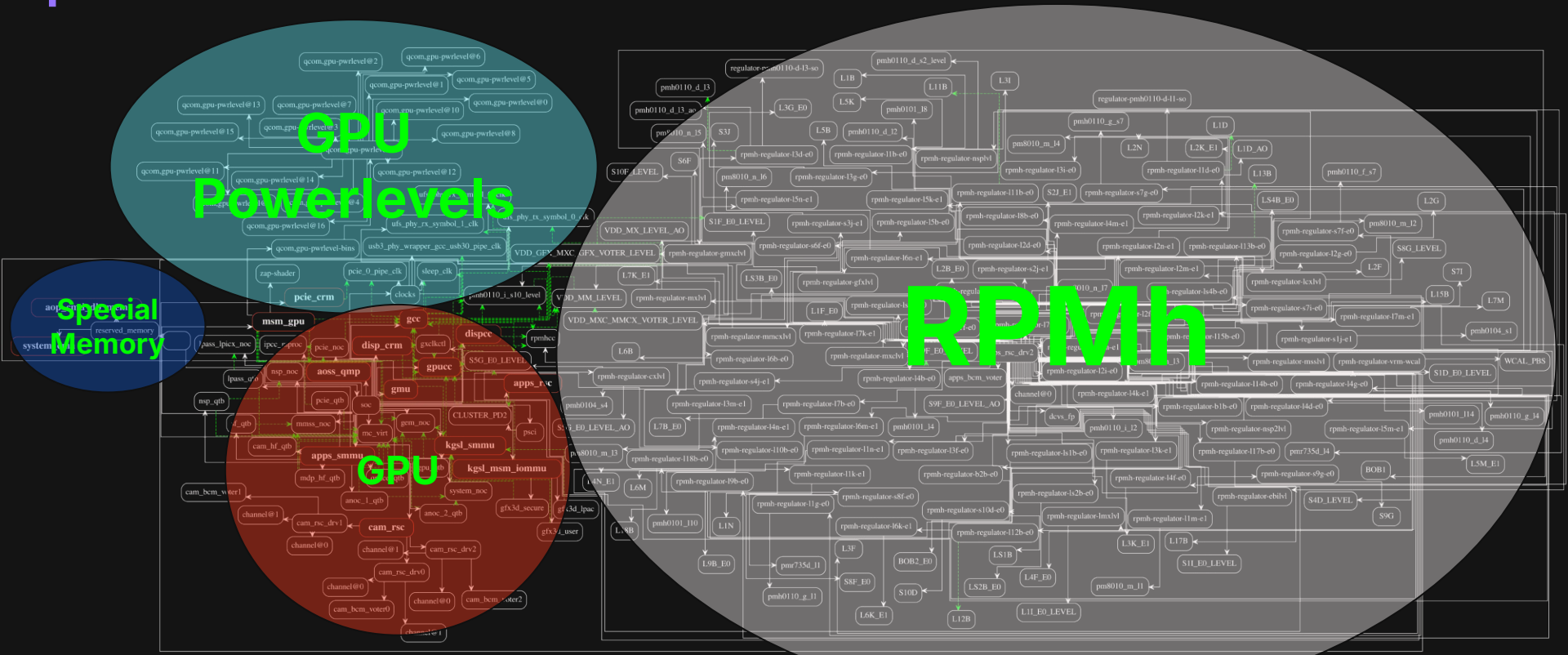
Snapdragon DeviceTree Overview



qcom-virt DeviceTree overview

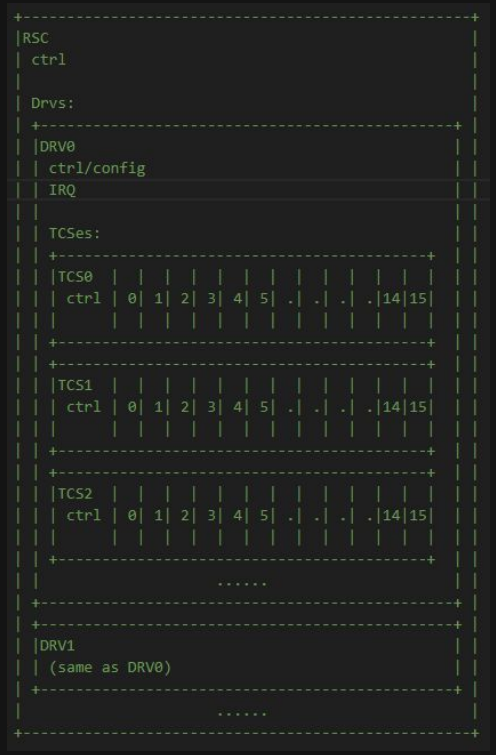


qcom-virt DeviceTree overview

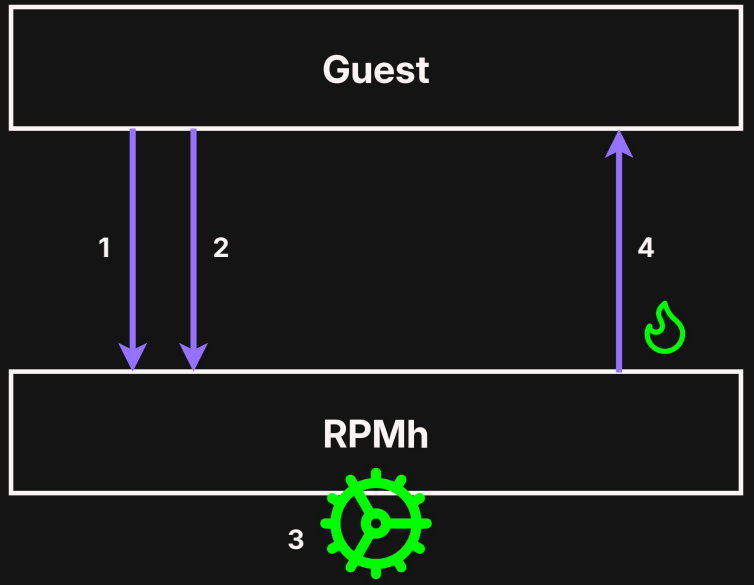


RPMh RSC implementation

- **RPMh** — Resource Power Manager hardened
- **RSC** — Resource State Coordinator
- Performs **power management tasks** — It's heavily used by the GPU
- **Asynchronous protocol** exchanging commands between the **host** and the **RPMh device**




```c
/**
 * struct tcs_cmd: an individual request to RPMH.
 *
 * @addr: the address of the resource slv_id:18:16 | offset:0:15
 * @data: the resource state request
 * @wait: ensure that this command is complete before returning.
 *        Setting "wait" here only makes sense during rpmh_write_batch() for
 *        active-only transfers, this is because:
 *        rpmh_write() - Always waits.
 *                       (DEFINE_RPMH_MSG_ONSTACK will set .wait_for_compl)
 *        rpmh_write_async() - Never waits.
 *                       (There's no request completion callback)
 */
Lina Iyer, 7 years ago | 1 author (Lina Iyer)
struct tcs_cmd {
    u32 addr;
    u32 data;
    u32 wait;
};
```



1. Write message
2. Trigger
3. Handle command(s)
4. Interrupt to signal commands completion

FrankSMMU

Purpose Map the address space of a device in the physical address space of the guest. The GPU has its own IOMMU called an SMMU

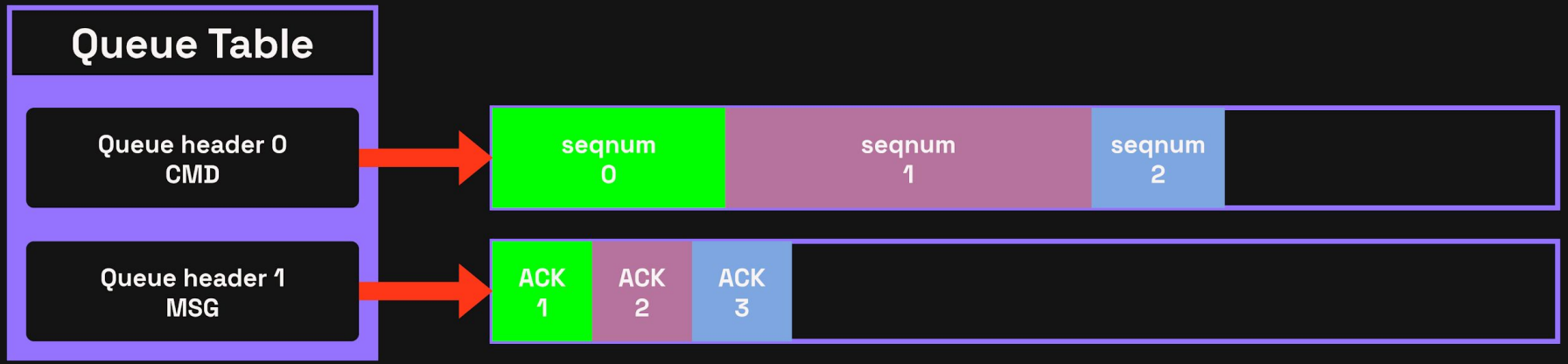
How We create a **fake MMIO device** (the *FrankSMMU*) playing the role of the Qualcomm's GPU IOMMU both in QEMU and the Android Kernel

```
/* Configuration registers for the franksmmu device */

#define QCOM_FRANKSMMU_PADDR_LO    0x39000   } Physical Address (PA)
#define QCOM_FRANKSMMU_PADDR_HI    0x39004   }
#define QCOM_FRANKSMMU_IOVA_LO     0x39008   } I/O Virtual Address (IOVA)
#define QCOM_FRANKSMMU_IOVA_HI     0x3900c   }
#define QCOM_FRANKSMMU_PGSIZE      0x39010   } Page size
#define QCOM_FRANKSMMU_PGCOUNT     0x39014   } Number of pages
#define QCOM_FRANKSMMU_PERM        0x39018   } Permissions
#define QCOM_FRANKSMMU_VMID        0x3901c   } VM ID
#define QCOM_FRANKSMMU_COMMIT      0x39020   } Commit → The request is registered by the SMMU
```

HFI

HFI (Host Firmware Interface) is the main hardware interface between the **GPU kernel driver** and the **hardware**

- It uses **DMA** for **exchange commands between the host** (the kernel driver) **and the firmware** (in the GPU).
  - **Asynchronous** — the host resumes execution immediately after a command is sent
  - **Bidirectional** — commands can be sent **from the host to the firmware and vice versa**
- The commands are **written in a ring buffer (in DMA)** and embed a **sequence number** to **keep track of the state of the communication**. Multiple ring buffers can be used in parallel.



How much work?

List of emulated devices

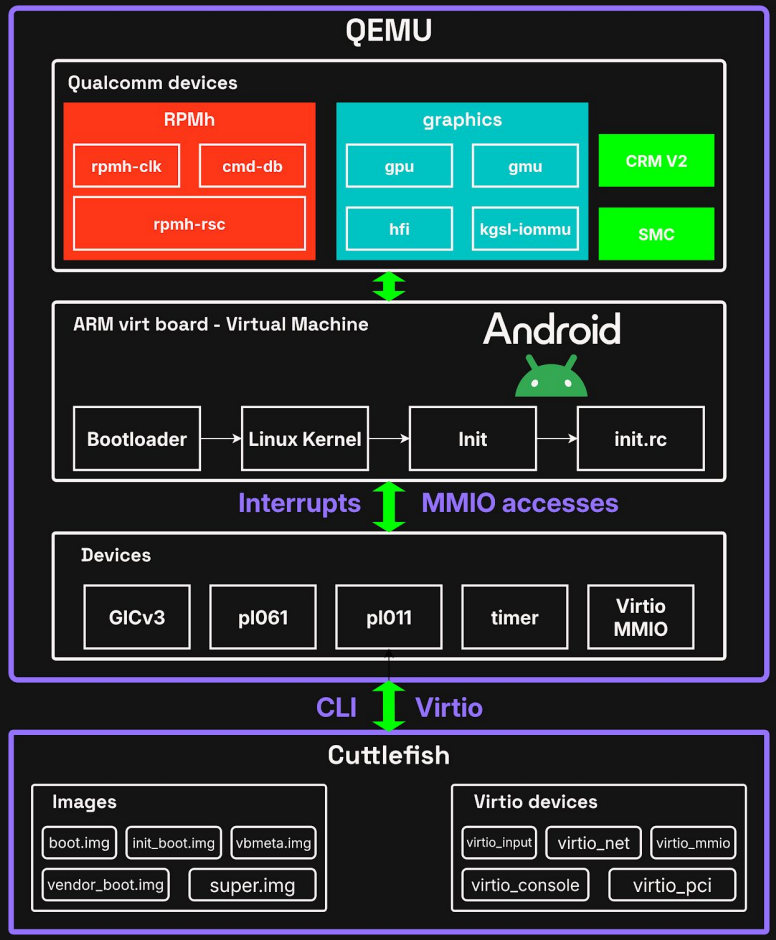
- cam-rsc, apps-rsc
- cmd-db
- disp-crm, pcie-crm
- dispcc, gpucc, gcc
- aoss-qmp
- kgsl-smmu, apps-smmu
- msm-gpu

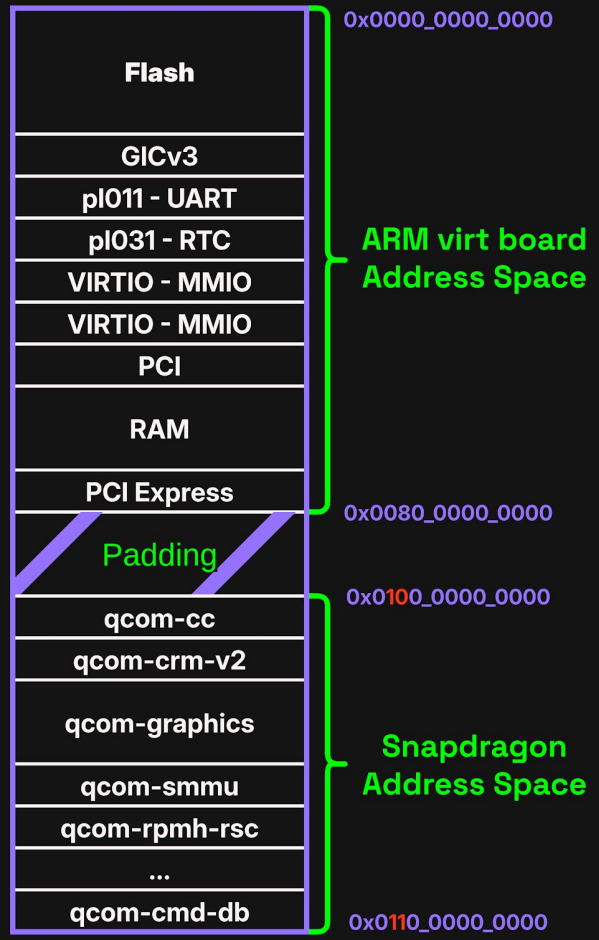
Work estimation?

+13,720 −49

```
README.rst                                 |   19
configs/devices/aarch64-softmmu/qcom.mak   |   26
hw/Kconfig                                 |    1
hw/arm/Kconfig                             |    5
hw/arm/boot.c                              |    2
hw/arm/imx8mp-evk.c                        |    2
hw/arm/meson.build                         |    1
hw/arm/qcom-virt.c                         |  574
hw/arm/raspi4b.c                           |    2
hw/arm/vexpress.c                          |    2
hw/arm/virt.c                              |    8
hw/arm/xlnx-versal-virt.c                  |    2
hw/arm/xlnx-zcu102.c                       |    2
hw/core/devlog.c                           |  238
hw/core/machine.c                          |   50
hw/core/meson.build                        |    2
hw/core/qdev-properties.c                  |    5
hw/core/qdev.c                             |   54
hw/core/sysbus-of.c                        |  237
hw/intc/arm_gicv3_cpuif.c                  |    2
hw/meson.build                             |    1
hw/qcom/Kconfig                            |   25
hw/qcom/cc/Kconfig                         |    3
hw/qcom/cc/cc.c                            |  270
hw/qcom/cc/clk-alpha-pll.c                 |  269
hw/qcom/cc/dispcc.c                        |  123
hw/qcom/cc/gcc.c                           |  493
hw/qcom/cc/gpucc.c                         |  170
hw/qcom/cc/meson.build                     |    6
hw/qcom/cmd-db.c                           |  562
hw/qcom/crm-v2.c                           |  860
hw/qcom/graphics/Kconfig                   |    3
hw/qcom/graphics/gmu.c                     |  365
hw/qcom/graphics/gpu.c                     |  207
hw/qcom/graphics/hfi.c                     |  864
hw/qcom/graphics/kgsl-iommu.c              |  112
hw/qcom/graphics/meson.build               |    4
hw/qcom/icc-rpmh.c                         | 1399
hw/qcom/logger.c                           |   65
hw/qcom/meson.build                        |   14
hw/qcom/qmp.c                              |  158
hw/qcom/rpmh-clk.c                         |  219
hw/qcom/rpmh-rsc.c                         |  720
hw/qcom/smc.c                              |  258
hw/qcom/smmu/Kconfig                       |    3
hw/qcom/smmu/meson.build                   |    2
hw/qcom/smmu/smmu-internal.h               |    0
hw/qcom/smmu/smmu.c                        |  249
include/hw/arm/boot.h                      |    2
include/hw/arm/qcom-virt.h                 |  124
include/hw/arm/virt.h                      |   47
include/hw/boards.h                        |   30
include/hw/core/devlog.h                   |   61
include/hw/qcom/cc/cc.h                    |  219
include/hw/qcom/cc/clk-alpha-pll.h         |   99
include/hw/qcom/cc/dispcc.h                |   14
include/hw/qcom/cc/gcc.h                   |   14
include/hw/qcom/cc/gpucc.h                 |   14
include/hw/qcom/cmd-db.h                   |   74
include/hw/qcom/crm-v2.h                   |   27
include/hw/qcom/graphics.h                 |   52
include/hw/qcom/graphics/cx_dbgc.h         |   14
include/hw/qcom/graphics/cx_misc.h         |   14
include/hw/qcom/graphics/gen8_reg.h        | 1672
include/hw/qcom/graphics/gmu.h             |   49
include/hw/qcom/graphics/gpu.h             |   43
include/hw/qcom/graphics/hfi.h             |  119
include/hw/qcom/graphics/kgsl-iommu.h      |   46
include/hw/qcom/graphics/kgsl.h            |   24
include/hw/qcom/graphics/rscc.h            |   14
include/hw/qcom/icc-rpmh.h                 |  103
include/hw/qcom/logger.h                   |   30
include/hw/qcom/qmp.h                      |   37
include/hw/qcom/rpmh-clk.h                 |   53
include/hw/qcom/rpmh-rsc.h                 |  142
include/hw/qcom/smc.h                      |   20
include/hw/qcom/smmu.h                     |   51
include/hw/qdev-core.h                     |   41
include/hw/qdev-properties.h               |   10
include/hw/sysbus-of.h                     |  129
include/qemu/bitops.h                      |  145
include/qemu/compiler.h                    |    3
include/qemu/iova-tree.h                   |    3
include/qemu/log.h                         |    1
include/system/device_tree.h               |  206
meson.build                                |    4
meson_options.txt                          |    5
qemu-options.hx                            |    8
qobject/qdict.c                            |    1
scripts/meson-buildoptions.sh              |    6
system/device_tree.c                       | 1232
system/vl.c                                |   86
target/arm/cpu.h                           |    2
target/arm/kvm-consts.h                    |    1
target/arm/tcg/psci.c                      |   18
util/log.c                                 |    2
96 files changed, 13720 insertions(+), 49 deletions(-)
```

Final overview of the Virtual Machine





Demo — Running the GPU test suite

CVE-2025-47397

Building emulators also helps to dig out new bugs

CVE-2025-47397 was found while writing the ***FrankSMMU*** device

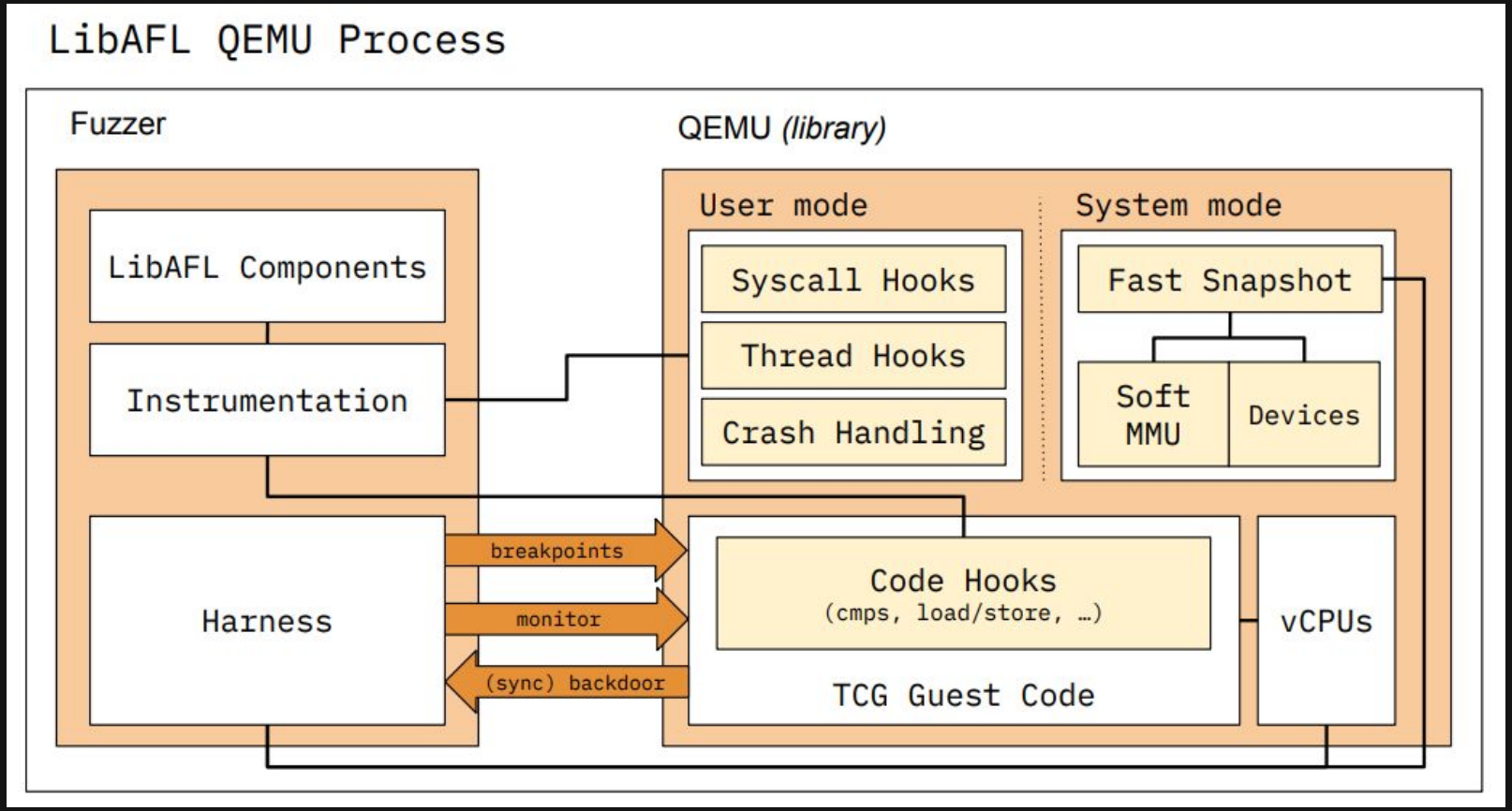
- read code to write emulated device
- saw strange logic
- looked into it
- bug found

➔ Fix sent to OEMs — public bulletin available soon™
(Technical details are still under embargo)

➔ We will update the presentation's repository (available on the last slide) with the technical details once the CVE is shared publicly

Fuzzing with LibAFL QEMU

LibAFL QEMU



Source: R. Malmain, A. Fioraldi, A. Francillon **LibAFL QEMU: A Library for Fuzzing-oriented Emulation**. BAR 2024

LibAFL QEMU — quick update since 37C3

- Generic **command handling** for both LibAFL and {kAFL,Nyx} targets
- Early support for **Intel PT tracing**
- many **bug fixes**, mostly thanks to the community
- A lot of other things I cannot cover here today

- Tooling:

  libvharess an easy-to-use library to create your LibAFL QEMU harnesses

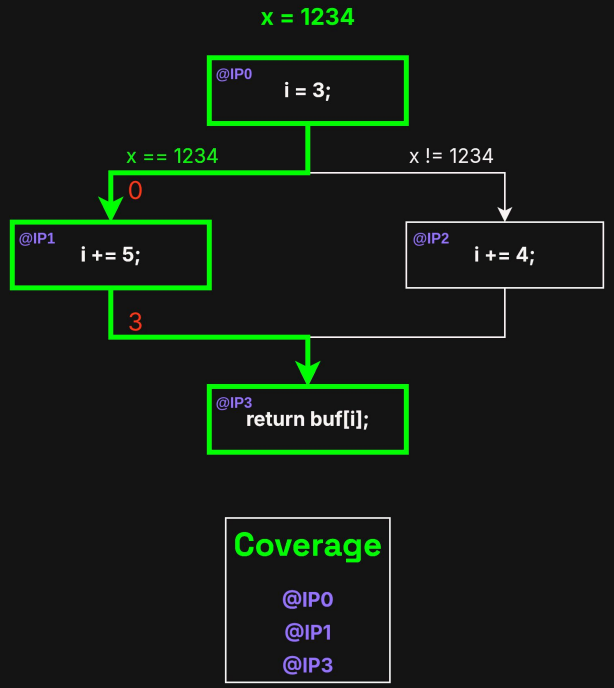
  linux-qemu-image-builder generates your own linux disk for fuzzing

KCOV implementation overview

KCOV is a Linux kernel extension able to collect coverage information for a given **task** (in the kernel sense).

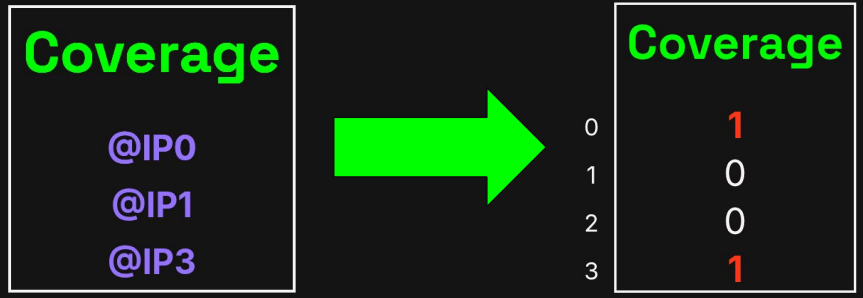
➔ **KCOV** outputs the coverage information as a **list of basic block PCs** through which the task has been.

```
char buf[8];

char fuzzing_target(int x)
{
    size_t i = 3;

    if (x == 1234) {
       i += 5
    } else {
       i += 4;
    }

    return buf[i];
}
```

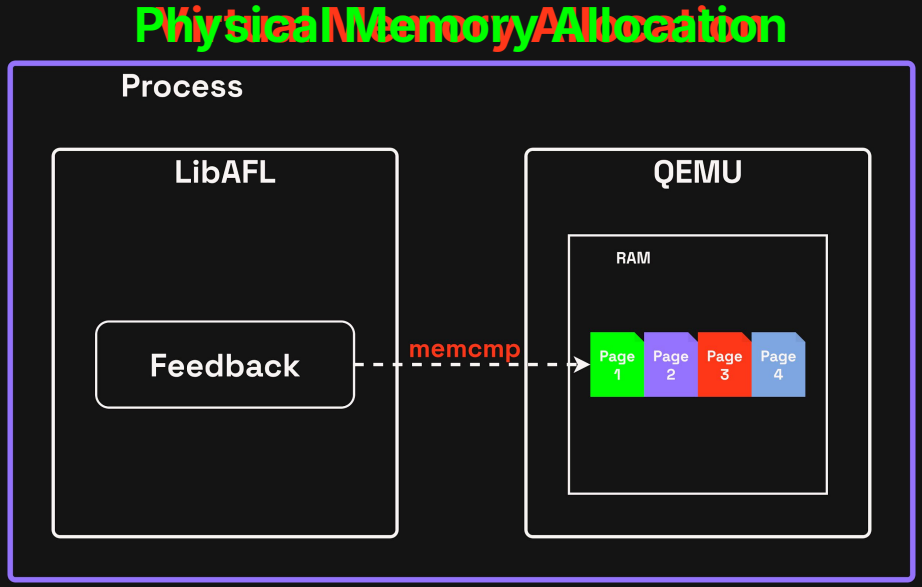


KCOV custom patch

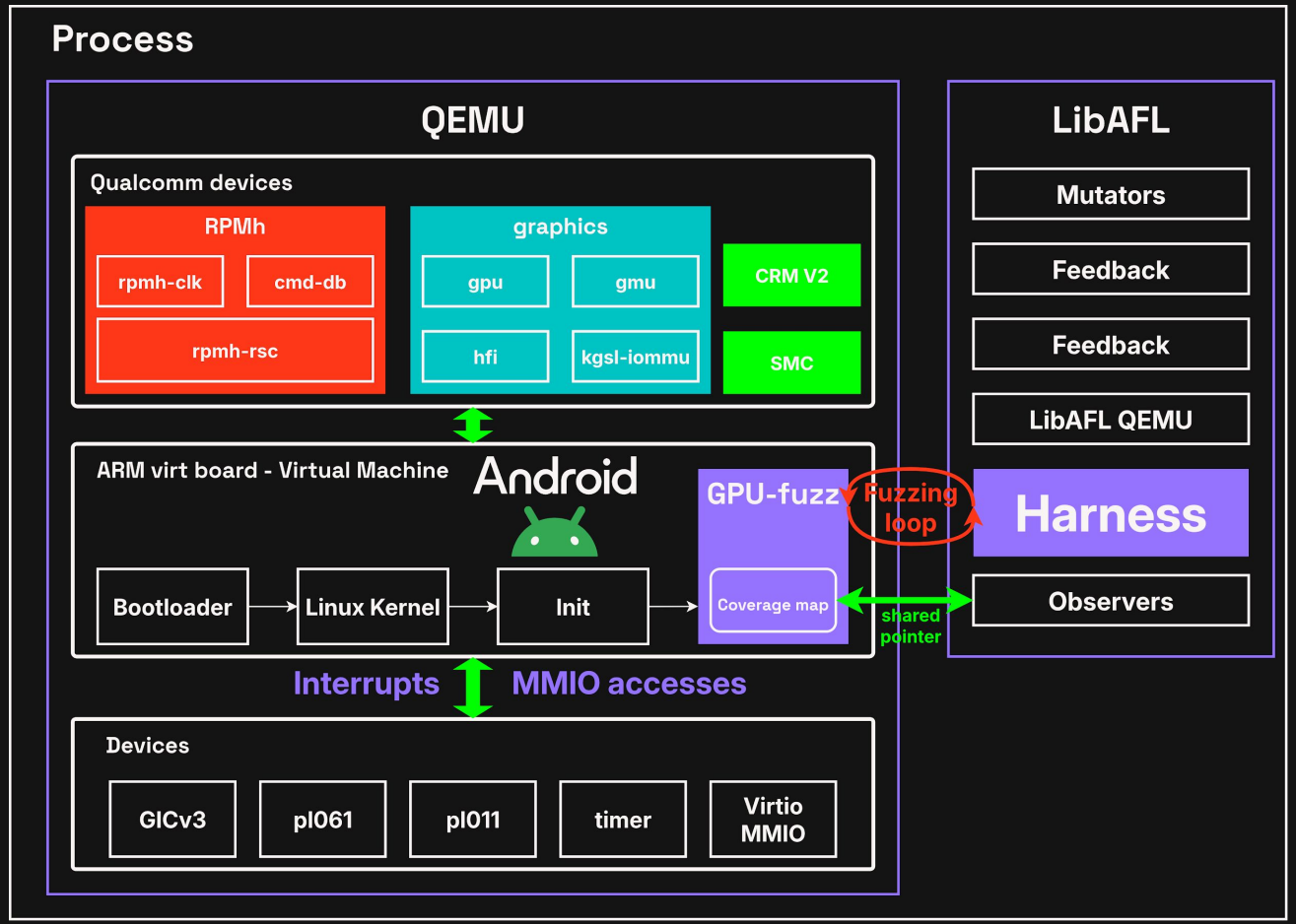
1. Turn IP coverage into edge coverage



2. Turn virtual memory mapping into physical memory mapping



Final integration with LibAFL QEMU



TL;DR

Available contributions

- QEMU patch with GPU support (PRs welcome!)
- LibAFL patches
  - libvharness
  - Support for in-VM coverage
- fdtviz: DTB visualization

Coming soon

- The KCOV patch
- The CVE technical details

Not available

- The fuzzing harness

https://github.com/rmalmain/39C3-build-a-fake-phone-find-real-bugs