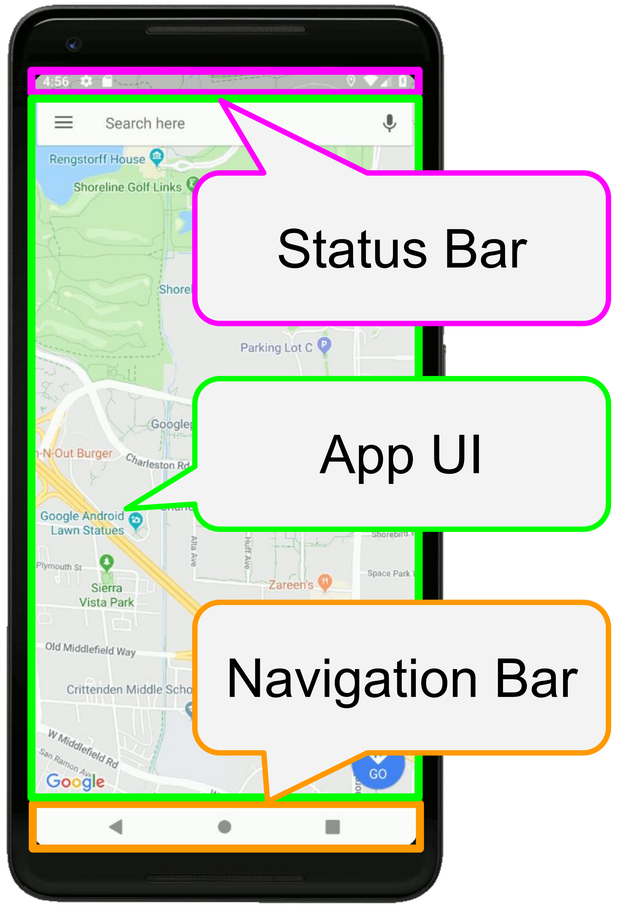
OSPM 2019, Pisa

Juggling scheduling entities:
the android display pipeline use case

-Alessio Balsini

What you see, and what you don’t



@VSYNC: Application

1. Reads input data
2. Prepares its rasterized frame
3. Commits the frame to the SurfaceFlinger

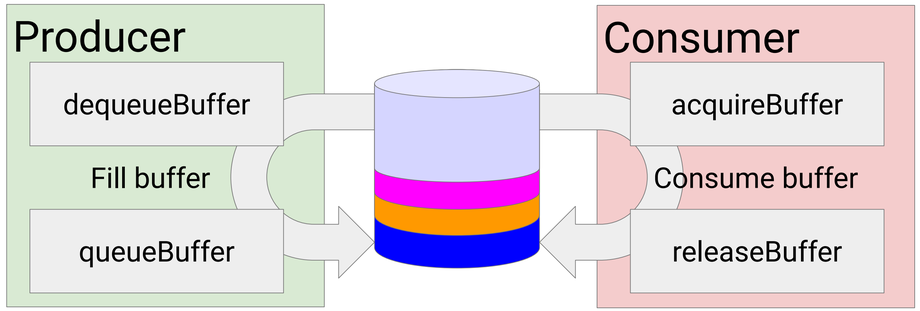
@VSYNC: SurfaceFlinger

1. Reads the buffers from App, Status Bar, Navigation Bar
2. Composites (merges) them
3. Commits the whole screen frame to the display

Prerequisites: inter-task communication

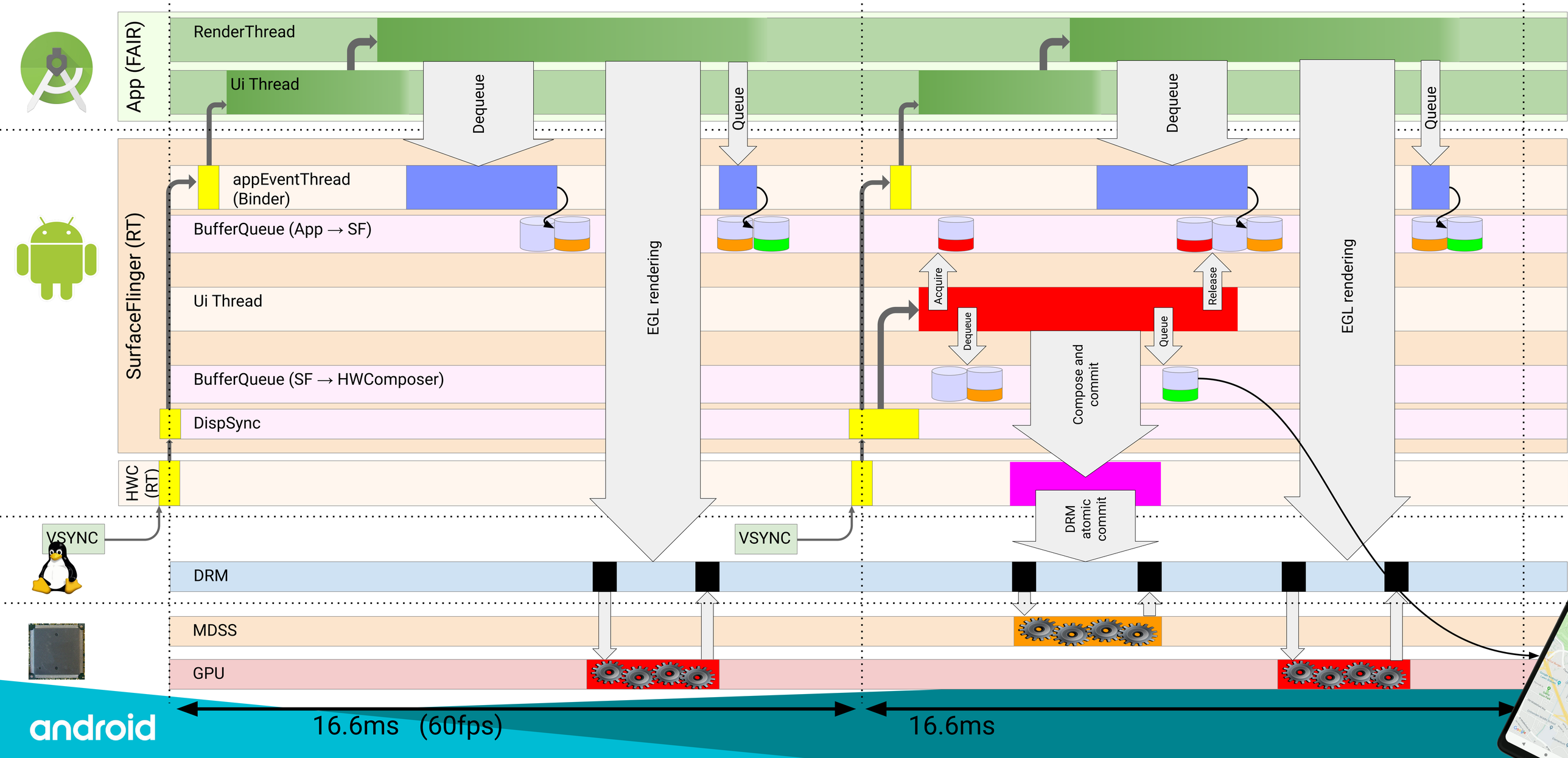
When not specified, tasks notification mechanism uses `signalfd` + `epoll`

Display Pipeline data is transferred among tasks via `BufferQueues` + `gralloc`

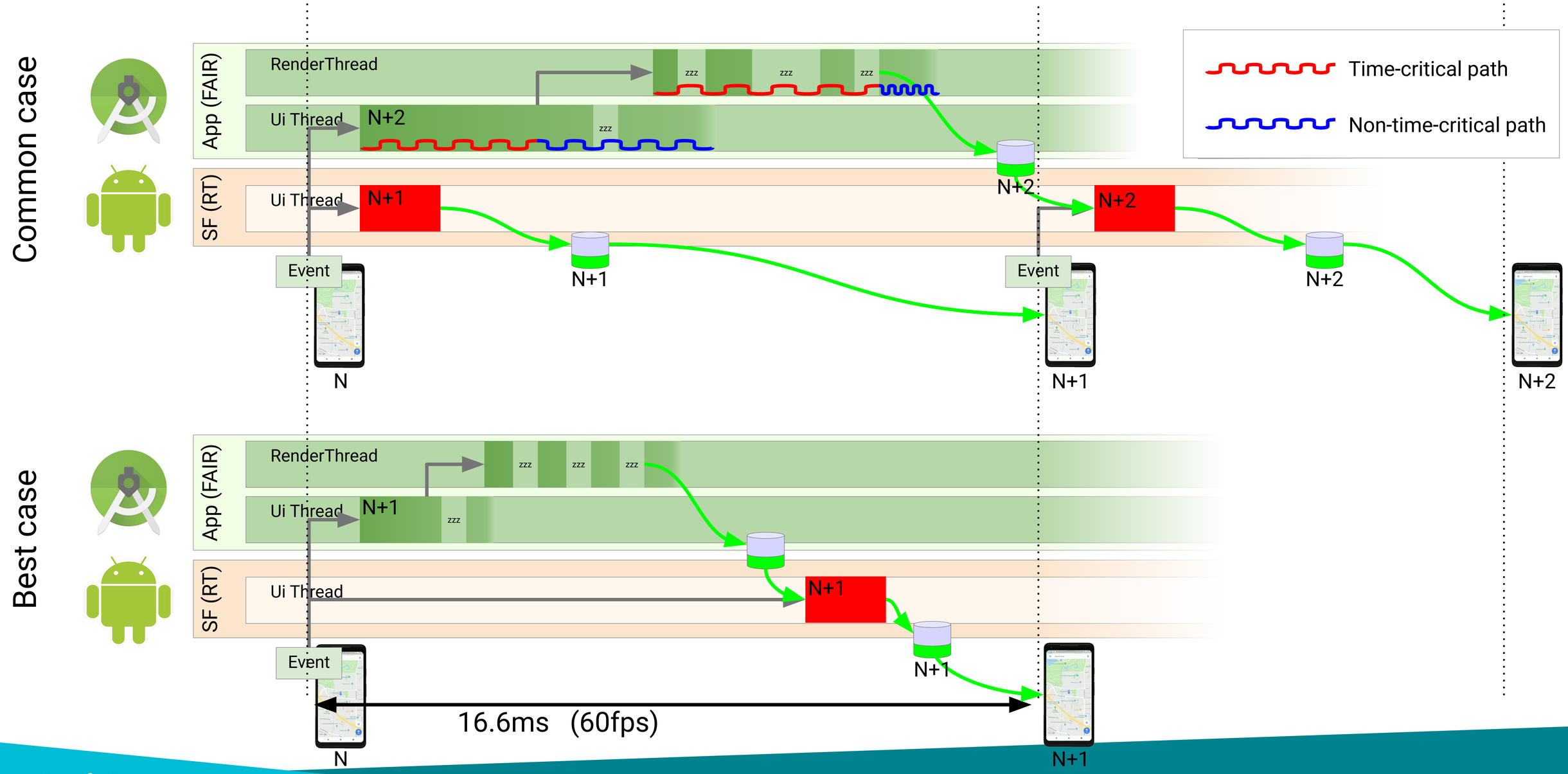


GPU operations between userspace and kernel are synchronized with `sync_fences`

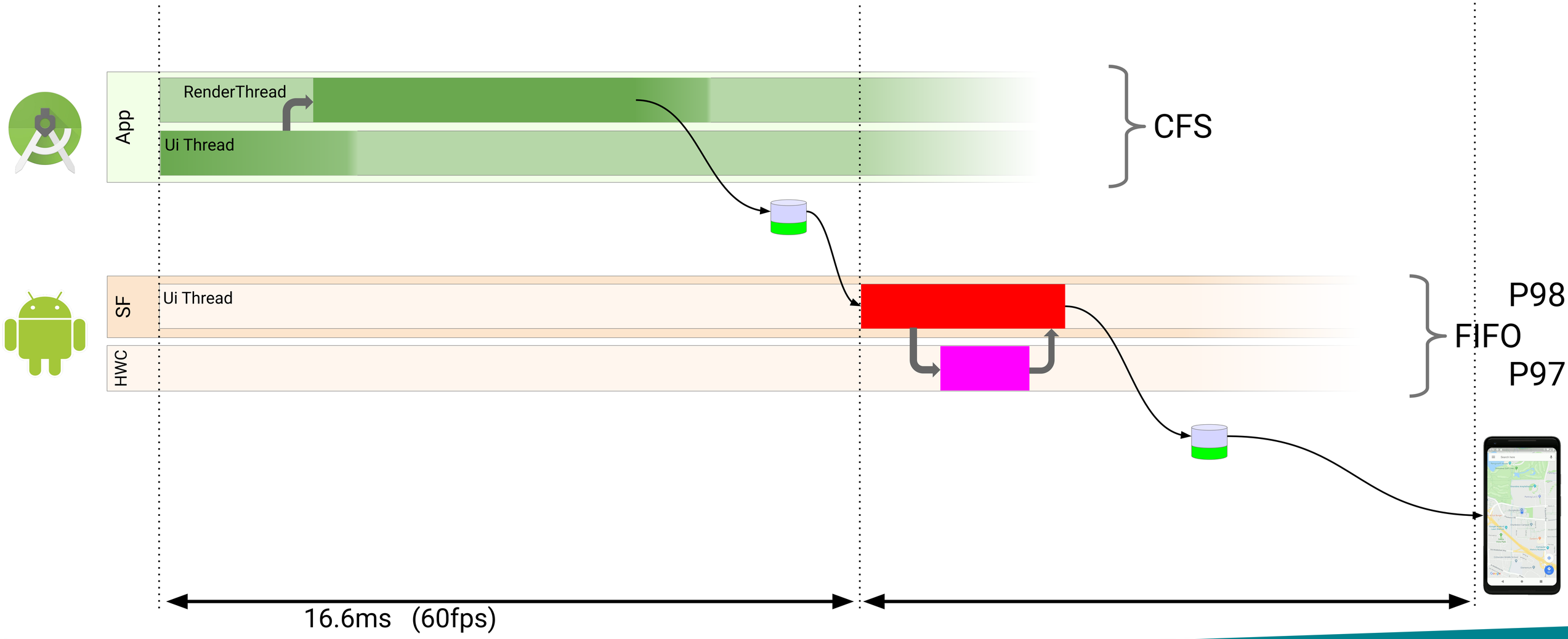
Almost fully detailed Android Display Pipeline



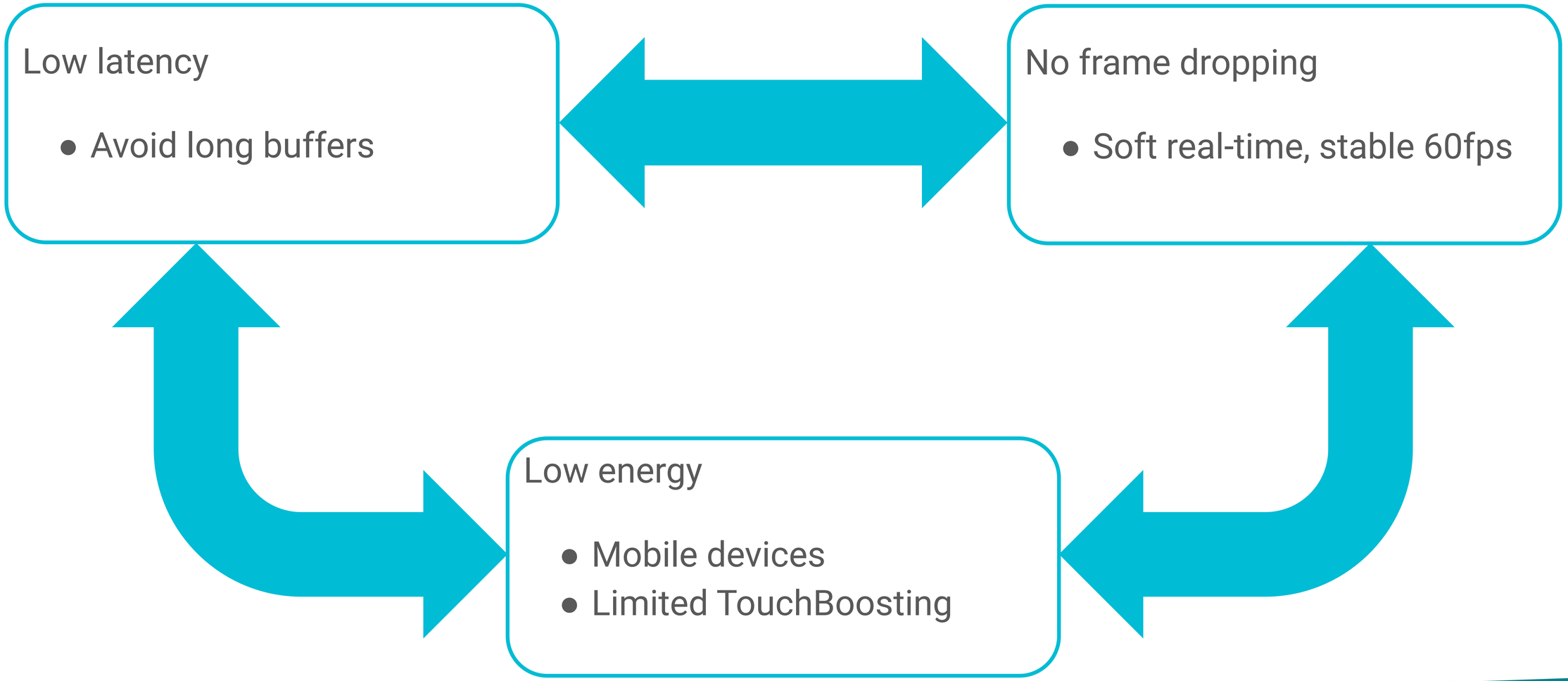
Simplified Display Pipeline



The two pipelines

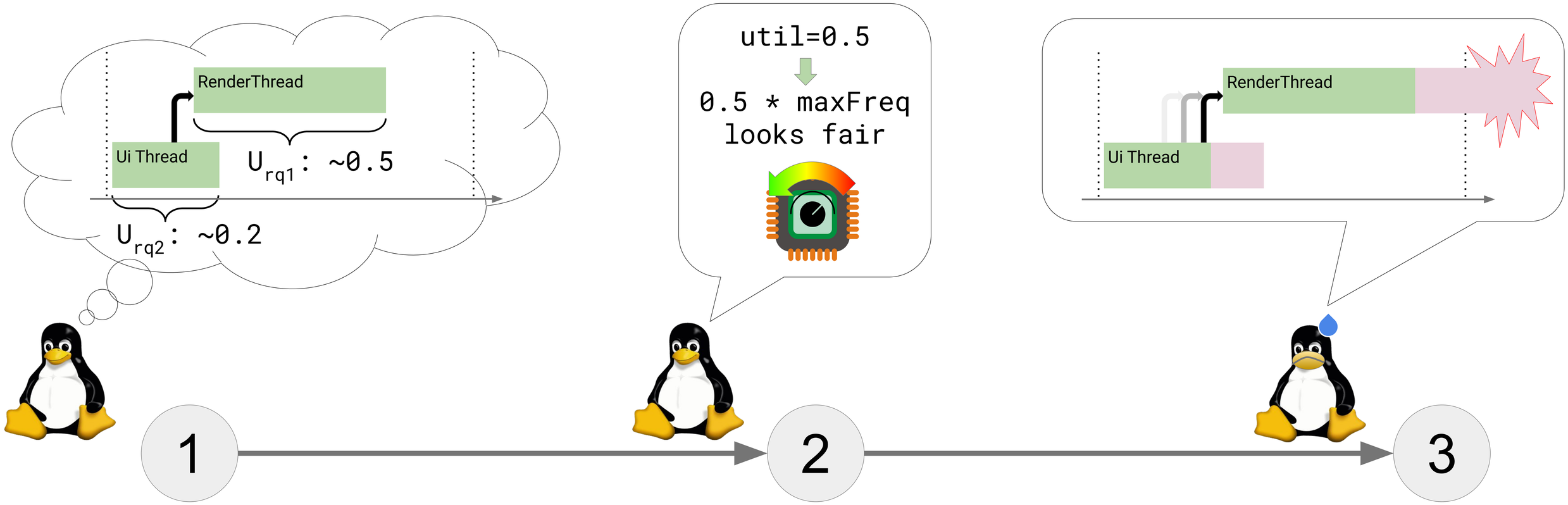


Wishlist



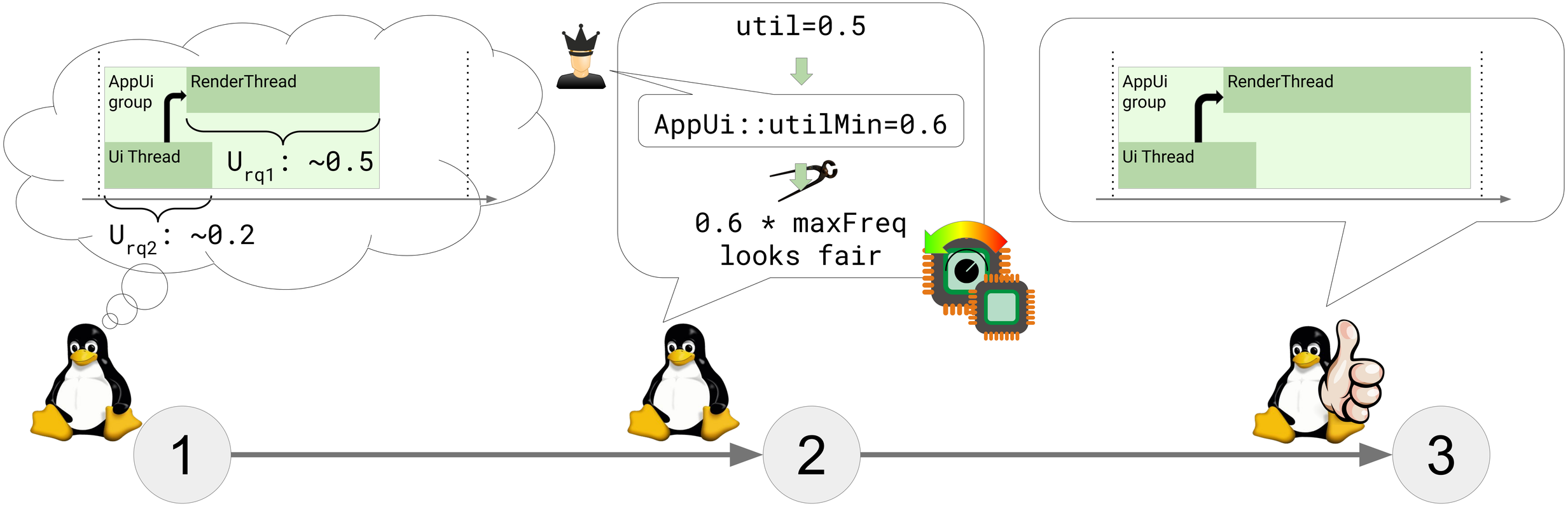
Open issue: CFS

The kernel thinks Ui Thread and RenderThread are independent, so are their utilizations



What if CFS + UtilClamp?

UtilClamp allows to constraint min and/or max utilization of single (or a groups of) threads

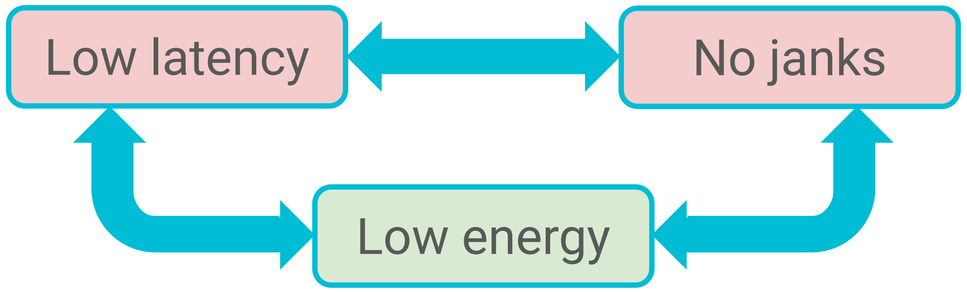


[PATCH v9 00/16] Add utilization clamping support --- Patrick Bellasi
https://lore.kernel.org/lkml/20190515094459.10317-1-patrick.bellasi@arm.com/

Open issue: CFS

Pros:

- Utilization-driven **OPP selection**
- **Fair** load distribution



Cons:

- **App threads are fundamental for the user experience**
  - They **should not fight against other low-priority services** for the CPU
  - They should be **scheduled ASAP**
- No notion of deadline

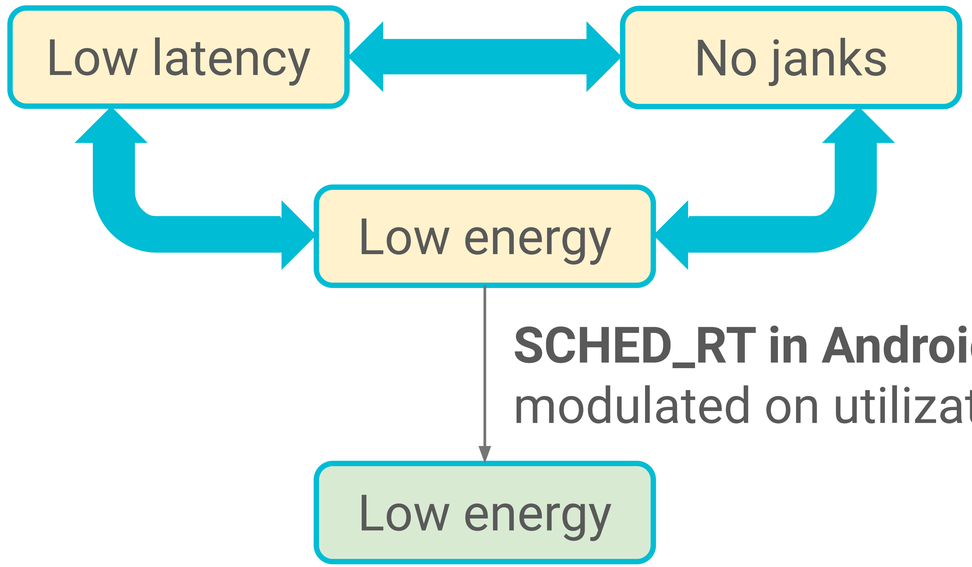
Open issue: RT

Pros:

- We start talking about **real-time**
- Better latency: **not preempted by CFS** tasks

Cons:

- Ui Thread and RenderThread are **App threads**, so we cannot trust their execution times
  - They should not **starve system** processes
  - Fundamental for the user experience
    - They **cannot be rt-throttled**, maybe demoted
- They are not time-critical all the time
- No notion of deadline

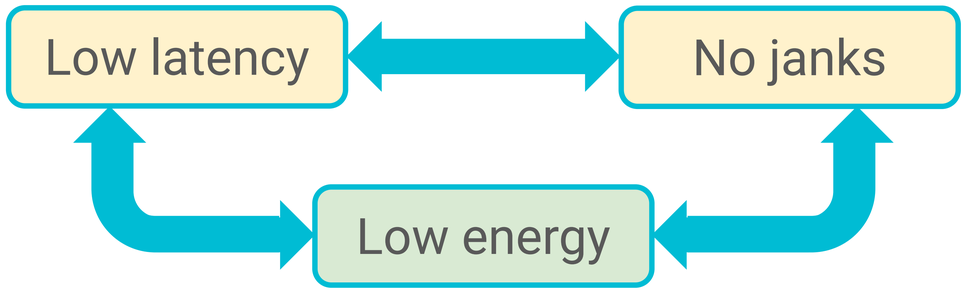


SCHED_RT in Android not implies max OPP, but is modulated on utilization heuristics with `schedutil`

Open issue: DL

Pros:

- Better latency: **highest priority** sched class
- **Tasks have a deadline**
- **Bandwidth constraints**
- **OPP selection** based on runtime and period



Cons:

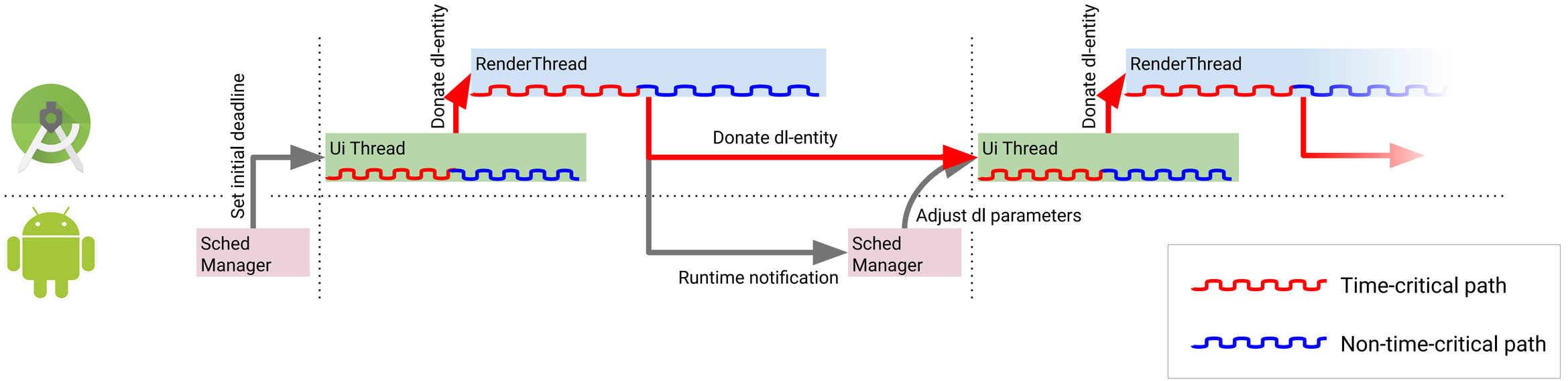
- Deadline **throttling** is still very aggressive
- Does not work well with task **suspensions**
- Does not work well with **inheritance**
- **Conservative**: a lot of bandwidth is required for both the tasks
  (the sum of the acceptably worst cases), but not all the sections of our tasks are **time-critical**

What if DL + Proxy Execution?

Our tasks have a **time-critical** path that is **sequential**

What if we provide a mechanism to **transfer** a “token” (**dl-entity** properties) **among tasks**?

Like a single dl-entity that is sequentially used by multiple tasks



Open issue: DL + Proxy Execution

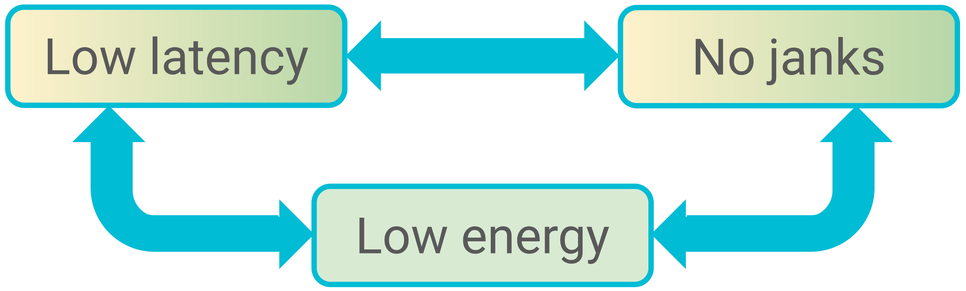
Parameters of the “shared” dl-entity updated with a feedback loop

Pros:

- Same as DL:
  - Better latency: **highest priority** sched class
  - **Tasks have a deadline**
  - **Bandwidth constraints**
  - **OPP selection** based on runtime and period
- **Less conservative**:
  (the acceptably worst case of the sum)

Cons:

- Deadline **throttling** is still very aggressive
- Does not work well with task **suspensions**
- Does not work well with **inheritance**



Open issue: Hierarchical DL/RT Scheduling

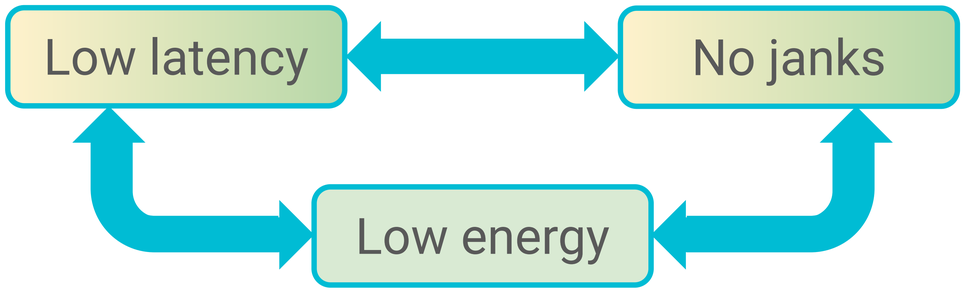
Special DL entities represent groups of RT tasks

Pros:

- Theoretically simpler schedulability analysis
- RT bandwidth constraints + deadlines
- A group of tasks shares the same DL
- GRUB-PA also for RT groups
- Clean RT and CFS bandwidth enforcement code duplications

Cons:

- Drawbacks of DL are inherited, e.g., affinities, bandwidth pessimism, …
- Overhead due to another scheduling layer
- Lots of migrations



Recap, Discussion, Action Items

CFS + UtilClamp

RT + UtilClamp

DL + Proxy Execution

DL/RT Hierarchical

Does it make sense to implement a **Proxy Execution** mechanism?

- How? Yet another `sched_setattr`?

What to do when tasks are **throttled**?

- Low OPP is hard to recover, but throttling is a harsh punishment

How to measure frequency/capacity invariant CPU time from userspace?

- CLOCK_PROCESS_CPUTIME**_INV**_ID
- CLOCK_THREAD_CPUTIME**_INV**_ID

OSPM 2019, Pisa

Juggling scheduling entities: the android display pipeline use case

Alessio Balsini <balsini@google.com> <balsini@android.com>