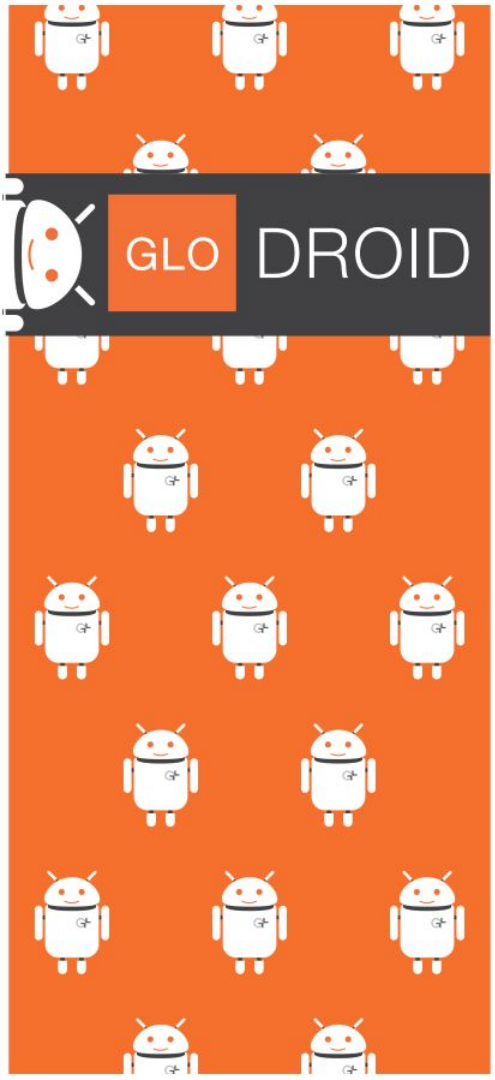
GLO DROID

GloDroid

or boosting true open source Android stack development.

1. Introduction
2. Components selection
3. Graphic buffer allocation
4. Hardware composition
5. Audio configuration
6. Meson and Soong

Roman Stratiienko, GlobalLogic Ukraine



1. Introduction

Supported

Raspberry PI 4

PinePhone

Orange PIs (-PLUS-2E, -PC, -3)

Pending

PineTab

Orange PI 4 (Rockchip)

Khadas VIM3

GloDroid components and mainlining status

CLEAN **AOSP** v11rX (>700 repositories)

1. **GloDroid** device configuration repository
2. Mainline **mesa3d** fork + 14 patches
3. Mainline **u-boot** fork + 17 patches
4. Mainline **drm_hwcomposer** fork + 1 patch
5. Mainline **gbm_fralloc** fork + 3 patches
6. Mainline **tinyhal**
7. Mainline **ATF**

Kernel:

8. Sunxi: integration branch from @megous mixed with Google kernel branch + 11 GloDroid patches

9. Broadcom: Raspberry PI integration branch + Google patches, 0 GloDroid patches.

+ Firmwares (6 repositories)
+ Linaro ARM compilers (3 repositories)
+ FOSS Applications repository (SkyTube, etc.)

Current mesa3d “git log”

```
74f9609318082 Fix v3d linking error
a438de834ddd0 android: mesa: Move the FXT1 compressor/decompressor to util/
a4e3678d6ff0b android: egl: Implement EGL_KHR_swap_buffers_with_damage
11556ea6af6ec Revert "lima: use linear layout for shared buffers ...
40542e6feb7db HACK: Support for Android-Q
68a80844dbf57 [RFC] v3d: Use v3d with vc4 for kmsro.
6ac46e9c167ce vc4: Fix Android build
24e38d311e8ac v3d: Enable Android build
512464b287b54 egl: android: add gbm_gralloc and drm_gralloc support
39b24cdeb1de6 egl: android: add IMapper@4 metadata API buffer_info getter
b58d86a5385d7 egl: android: prepare code for adding more buffer_info getters
3a02a003c4e68 egl: android: use num_planes param in createImageFromDmaBufs()
fcf524e9866a4 frontend/dri: add AR12 and AR15 format support
a5d29cce84195 android: loader: convert Android.mk to Android.bp
66fd9de6f5e81 ----------------------- mainline mesa3d --------------------
7e77bfb68a9e4 ....
```

Global Logic Ukraine 2018

Manager A

We are growing and need more Android BSP engineers!

Unfortunately, we can't find anyone on the market.

Manager B

We've run linux kernel training program several times. Can we start the one for Android?

Sure.. Let's see... We need a development boards to train BSP engineers. Experts, can we use OrangePis from the kernel training program?

Engineer A

Looks like no. HiKey is currently the only board that can be used for these purposes. OrangePis has only prebuilt Android images and we can't use them.

Ok, Then let's buy some HiKeys. But wait. It costs 250$ each and they are no available on local market. We have to wait 2 months for arrival. Well, we have no choice, let's buy some.

Guys, are you sure there is no new Android BSP with open source for so popular Raspberry/Orange/BB boards we used before?

Looks really weird, Android is opensource and it is more then 10 years on the market. Why only closed ROMs are available for most of the boards? Let's do some quick investigation.

Engineer B

Month later

I have good and bad news.
Goon one: I was able to flash AOSP system and vendor images built for HiKey to the OrangePi and board got booted and stuck at UI loading stage.
Bad news: We do not have GPU GLES drivers for Android, SwiftShader crashes on 32bit ARM. Mesa3d software renderers are extremely slow (1FPS on single and 4FPS on multi-core implementations).

6 months later

MALI drivers are available in opensource domain and currently enters into mainline kernel and mesa3d as lima for MALI400-450 and panfrost for MALI MIDGARD and BIFROST.
Let's try running Android with them in a free time.

9 months later

After fixing minor segfaults I've got LIMA working. UI still has some artifacts but overall performance is good enough.
@Managers: Opensource portfolio is usually a good for the IT companies, Consider starting POC for bringing-up FOSS Android-BSP for OrangePis and other SBCs to the public domain.

Great news, Let's start the POC and motivate engineers to help us!

Sounds like a challenge:)
I will do my best!

Project goals:

Goal #1. Bring-up 100% open-source Android that can be downloaded and compiled using few shell commands.
Goal #2. Support most popular and accessible development platforms.
Goal #3. Be as close as possible to the mainline, so our project can be easily integrated with other up-to-date mainline-friendly components.

Solutions:

Solution for #1. Choose build system
Option A (current): Extend AOSP build system by creating custom rules.
Option B (may be the future): Create top-level build system (use yocto, etc.) and use AOSP as is.
Solution for #2.
Continuously do more research.
Different SOCs vendors have unique boot flow and driver set, different hardware acceleration features., etc. Common things should be combined and only differences should be described.
Solution for #3.
Find the appropriate components already available, choose the most advanced and up-to-date from the list, always contribute our experience back to the mainline.

Top directives:

Do not change AOSP system part. Create only temporary forks of AOSP vendor part (to test or while waiting up-streaming). Reuse as much as possible from Linux world (ARMBIAN scripts, etc.)

Limitations:

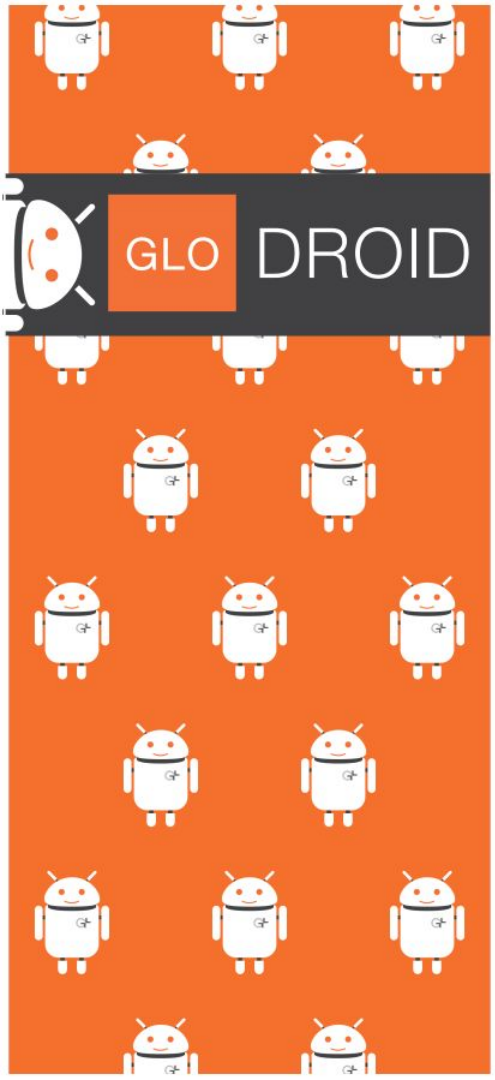
All above should be implemented considering constraints on available engineering resources for POCs in s/w service company (part-time assignments with low priority). Otherwise we are risking to slow down or completely stop.

Solution: Align the project with the company's interests:
Leverage the POC for training/mentoring activities
Start doing attempts to commercialize our work

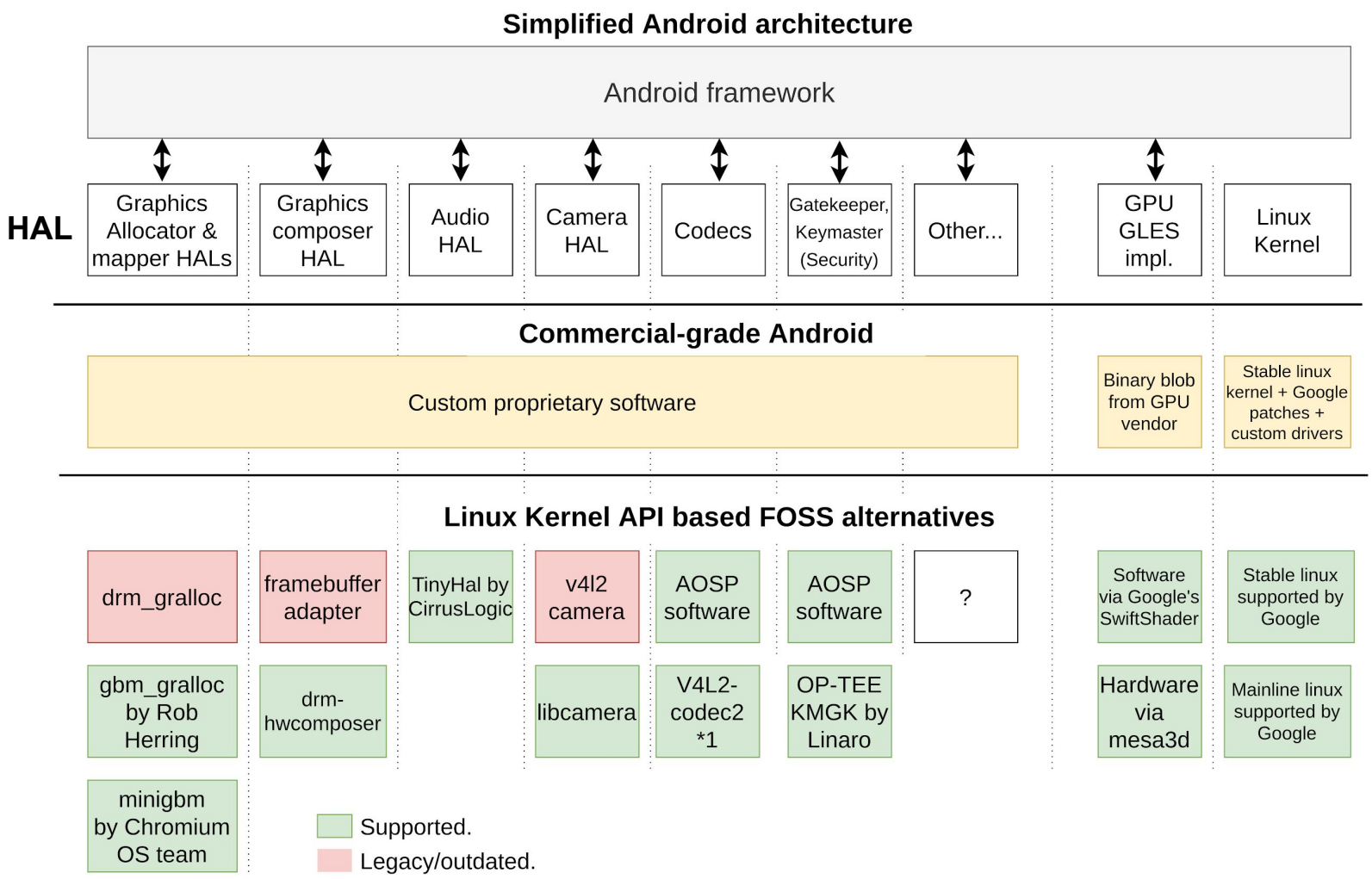
Commercial world philosophies:

1. All code should be kept proprietary, opening our work to the FOSS domain will reduce our advantages on the market.

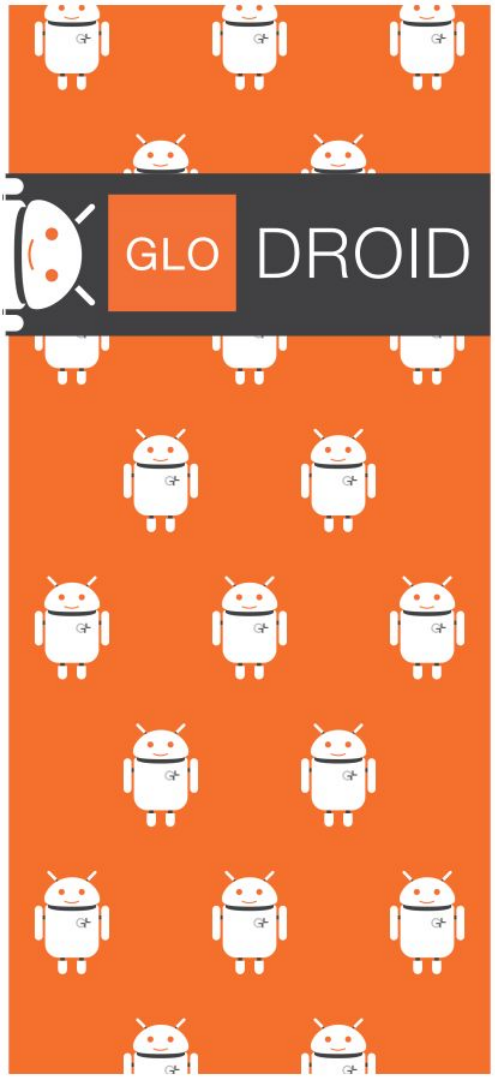
2. Part of the code that isn't at cutting edge of the technologies can be open-sourced.
We won't lose anything. Our competitors do have already these features. However, we can reduce our own effort by sharing our work with community. Instead we can invest freed up time into increasing our competitive advantages by doing more innovative stuff.



2. Search for already available components



**1. No opensouce mainline, only opensource snapshot available. Release cycle - 1 year.*



3. New vision for graphic buffer allocation

1. Simplified AOSP Gralloc API
2. gbm_gralloc
3. minigbm

Simplified AOSP Gralloc API

Supported buffer users:
- CPU (via mmap),
- GPU (Texture, Render target)
- Display controller (composer)
- Camera,
- Hardware video encoder
- Other

```
struct buffer_handle_t  {
  int buf_fds[];
  int gralloc_impl_spec_data[];
}
```

I need the buffer to use by GPU and by the DISPLAY controller

Binder/HIDL session

allocate(W,H,...)

buffer_handle_t

Allocator

Select optimal buffer location (cma, vmem, etc.) and optimizations (tiling, compression, etc) based on usage flags. Allocate.

I have raw temporary handle now, I must clone or import it.

import(buffer_handle_t)

buffer_handle_t

Mapper

Clone the **native_handle_t**, so the temporary one is no longer needed.
Register cloned handle.
Native_handle_t serves as a key which can be used now to access various Mapper API.

Buffer Metadata API (get fourcc, modifier, strides, pitches, other plane-related information etc.)

CPU access API (Map to userspace, lock region, etc.)

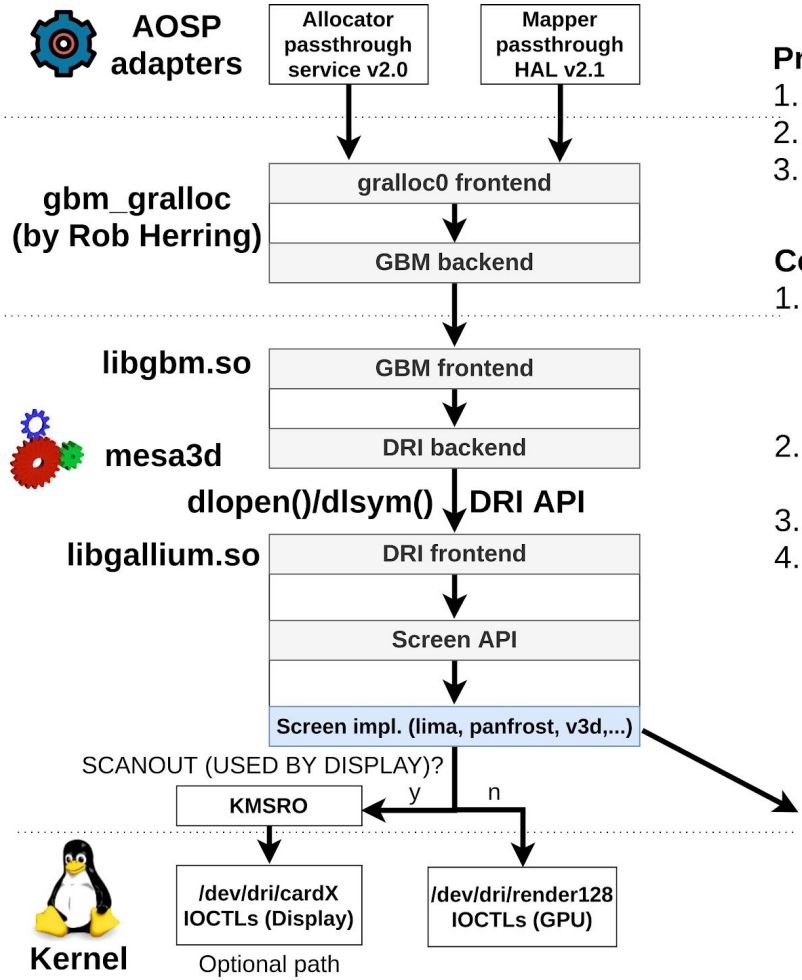
Now I have imported handle. I can use this buffer by GPU and send the handle to hwcomposer service to display it.

Good, I don't need this buffer anymore. Let's free the memory.

free(buffer_handle_t)

Unregister **native_handle_t.**
Close dma-buf fds.

gbm_gralloc



Pros:

1. Simplicity combined with functionality.
2. Relying on linux-side development/test cycle
3. Any new driver in mesa3d will be supported out-of-the-box

Cons:

1. Stuck at HAL@2.0, Android-10 require HALv3 Android-11 HALv4 for certification. Moreover HALv4 introduces metadata API which improves overall graphic stack.
2. Support for camera/video-encoder buffers isn't implemented.
3. Minimal community support
4. Long pass-through chain. Only mesa3d screen impl. does something really useful.

Selects correct optimizations based on many inputs: GPU family, type and usage flags: [SHARED, LINEAR, SCANOUT]

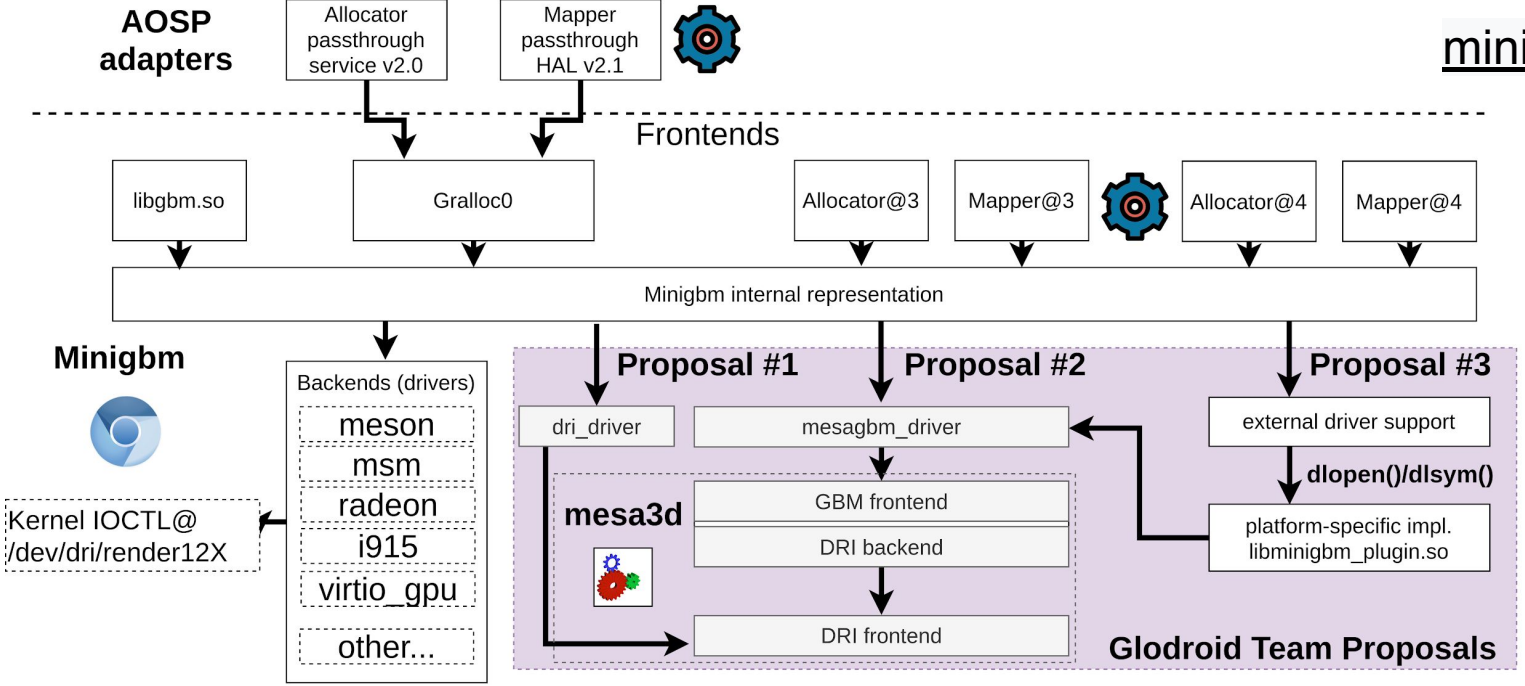
Optimizations example:

- Use of lossless compression (Reduces load on memory bus)
- Use of tiling (Optimizes partial update and more)

Allocation area example:

- Physically-contiguous memory
- Virtual memory

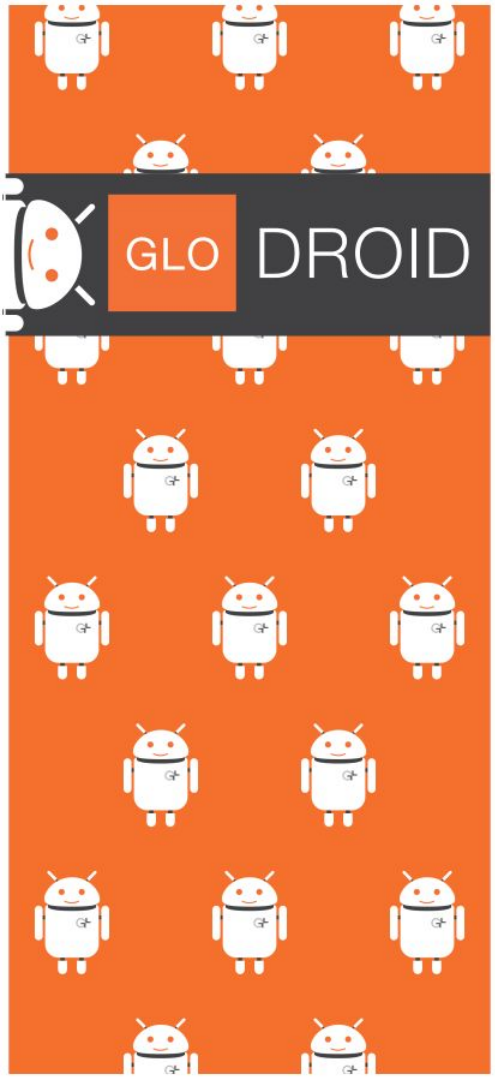
minigbm



Pros: Latest frontend APIs, reduced code-path comparing to gbm_gralloc, ChromeOS community support, included into the AOSP.

Cons: Duplication of optimizations selection code with mesa3d, which is **likely** not synced with mesa3d driver internals. No support for systems where buffers for display controller and buffers for GPU should be allocated in separate memory spaces. (CMA, VMEM). Limited driver support.

- Proposal #1: Implemented at : https://chromium-review.googlesource.com/c/chromiumos/platform/minigbm/+/2584842
  Cons: To make pure-generic implementation significant part of additional logic must be copied from libmesagbm.so
- Proposal #2: Should address cons from proposal 1.
  Cons: No flexible per-device configuration.
- Proposal #3: Aims to address cons of proposals 1 and 2. Still slow code-path, is it really slow...?



4. Our architecture proposal for hardware-accelerated graphic compositor.

1. Simplified AOSP HWComposer2 API
2. drm_hwcomposer project introduction
3. Proposal for structural redesign
4. Introduce component level planning
5. Improve DRM-level planning
6. Optimization

HWComposer2 API

I've finalized rendering to the graphic buffers. Now I need to merge them and send to the user's display.
The platform I am running on may have 2D accelerators to complete the merging, so GPU power can be used to draw something else during this time.

HWComposer, here is my composition, can you handle it?

set layers info.
ValidateDisplay()

HWComposer

Well, I can handle 2 layers only, you gave me 4. Please squash 3 into single layer.

Good, which layers should I squash using GPU?

GetChangedCompositionTypes()

Layers #2, #3 and #4

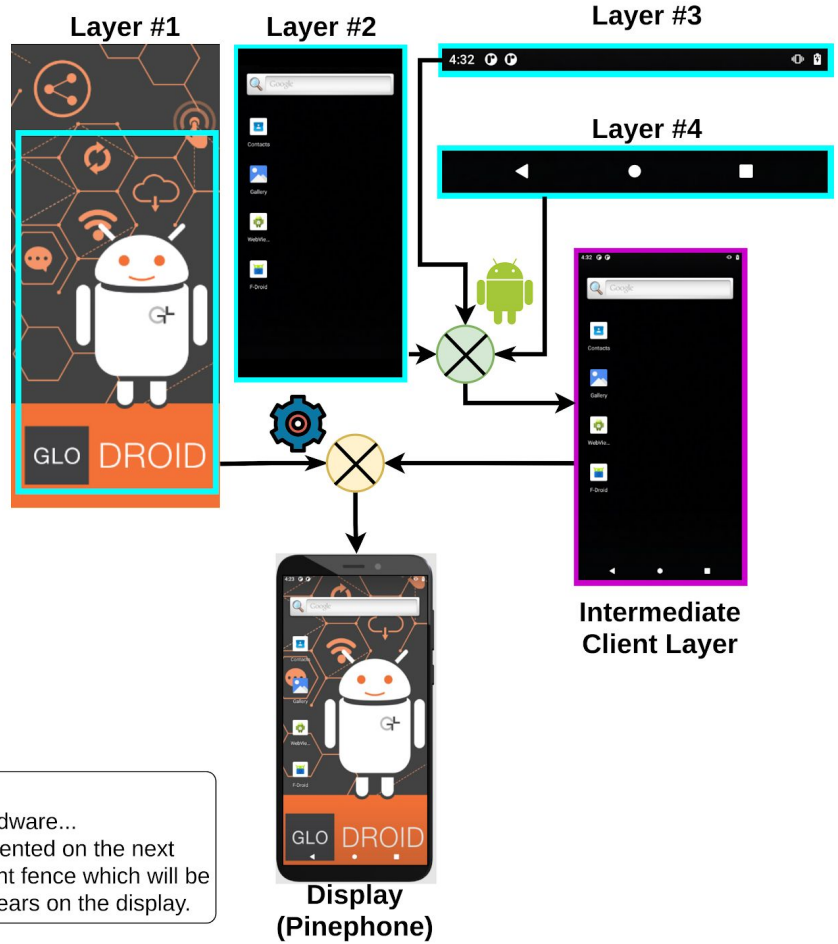
Squashing...
RenderEngine(), do the thing!

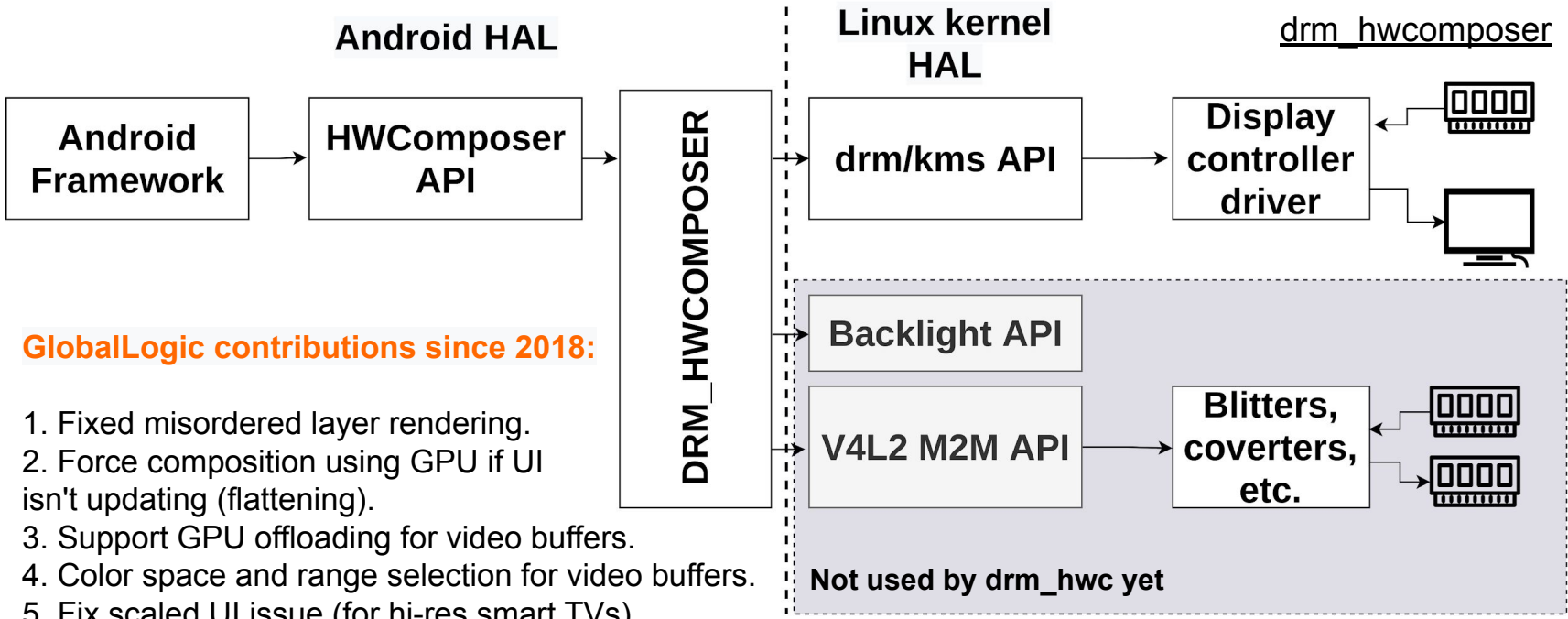
Done. Here is the squashed layer. Let's display the composition.

SetClientTarget()
PresentDisplay()

Thanks!
Configuring the hardware...
Your composition will be presented on the next VSYNC event. Here is the present fence which will be signaled once new picture appears on the display.

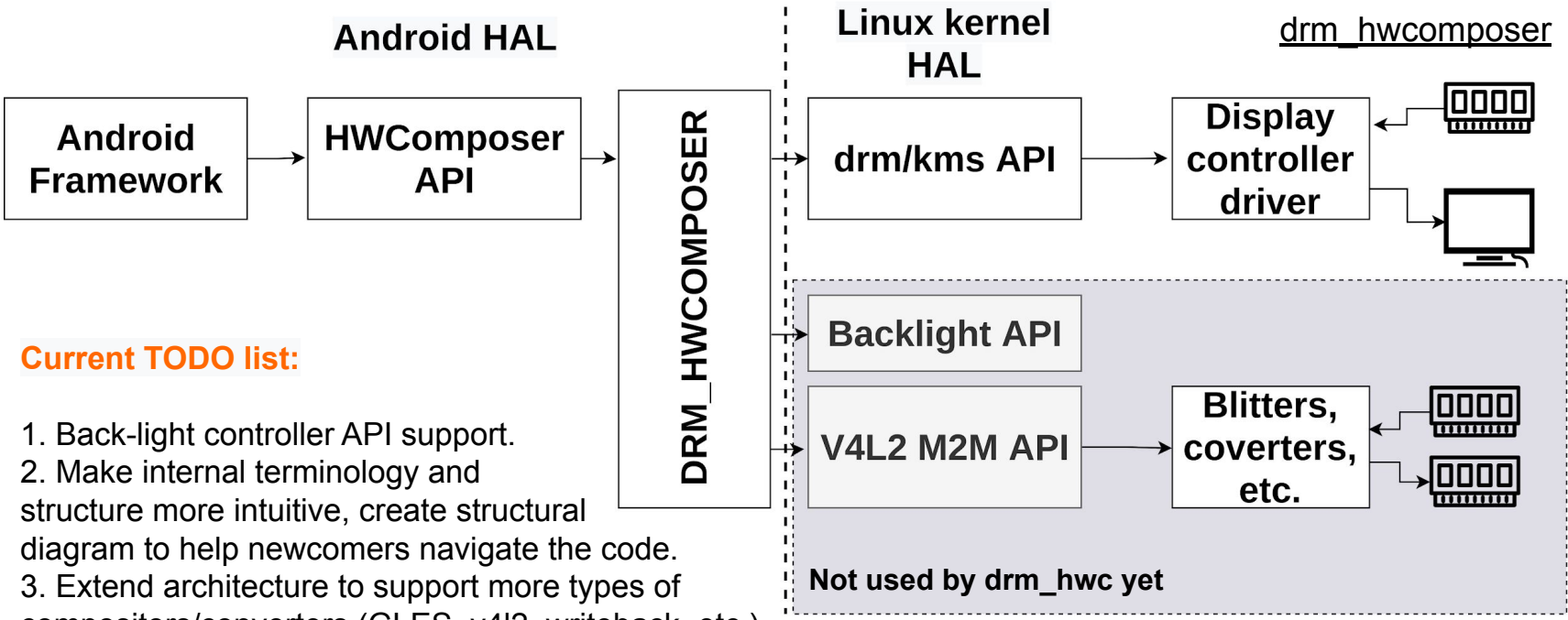
Good. Let's move to the next frame.





GlobalLogic contributions since 2018:

1. Fixed misordered layer rendering.
2. Force composition using GPU if UI isn't updating (flattening).
3. Support GPU offloading for video buffers.
4. Color space and range selection for video buffers.
5. Fix scaled UI issue (for hi-res smart TVs).
6. Fixed some of AOSP VTS tests.
7. Ability to select primary display across multiple external displays.
8. KMS device lookup feature.
9. Add basic support for Imagination Grallocs buffer metadata extraction.
10. Implemented algorithm that reduces load on the GPU by correctly choosing layer set to squash by GPU.
11. First steps to reorganize project structure. (decouple Gralloc-specific buffer information converters from platform customization logic)



Current TODO list:

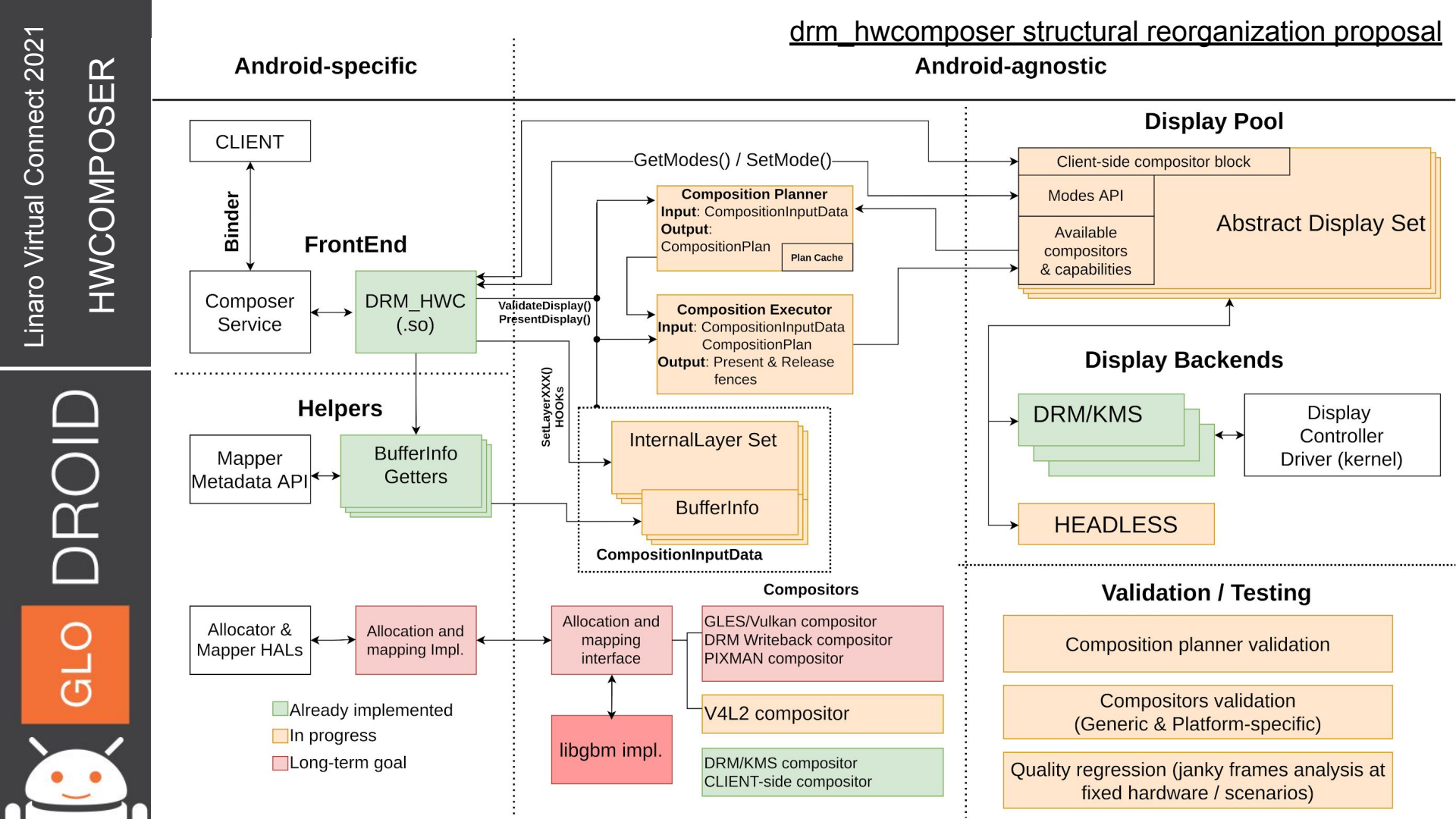
1. Back-light controller API support.
2. Make internal terminology and structure more intuitive, create structural diagram to help newcomers navigate the code.
3. Extend architecture to support more types of compositors/converters (GLES, v4l2, writeback, etc.)
4. Fix FPS rate drop on multiple displays by switching to asynchronous atomic commit.
5. Use advanced algorithm for layer to plane mapping.
6. Properly use shared KMS planes.
7. Fix other reported issues.

More....

Proposal main goals:

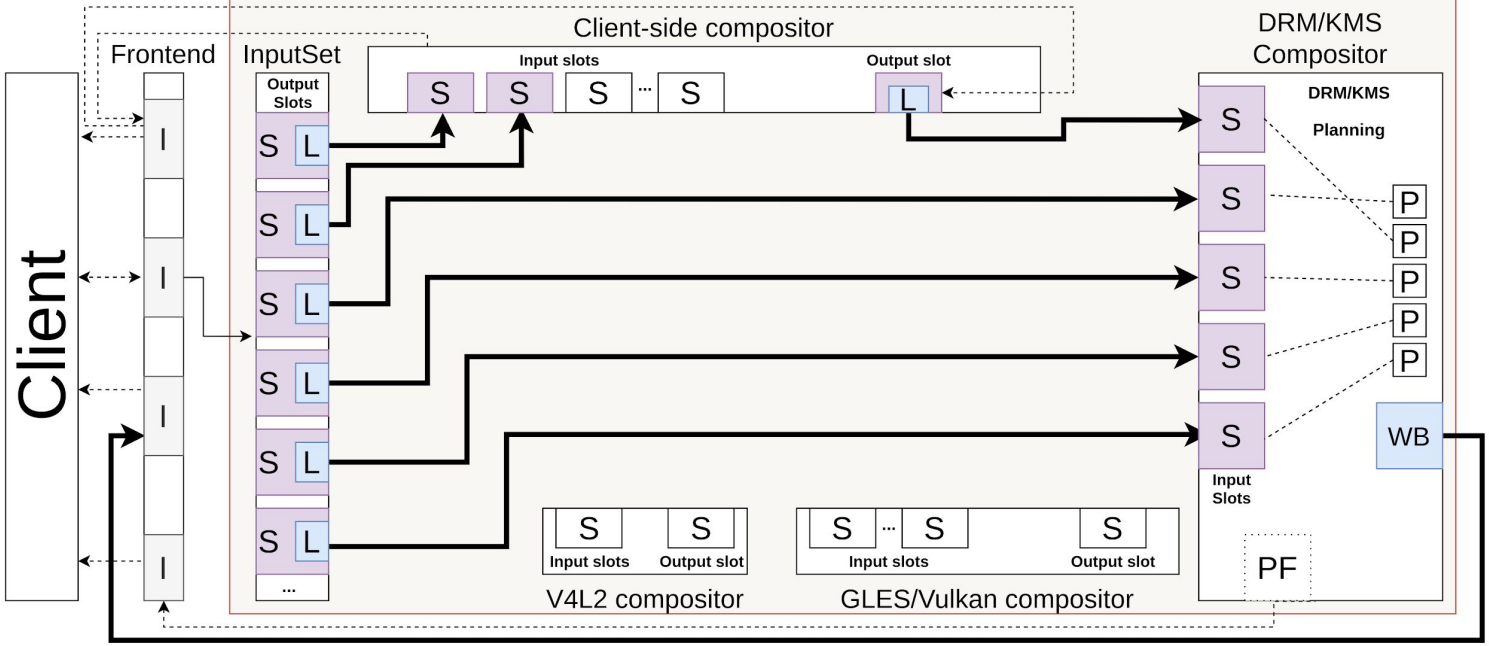
1. Help newcomers to navigate the code.
2. Ability to create automated test scenarios and share them with kernel-side driver maintainers as pure-linux application to report kernel issues.
3. Add new compositors and other techniques to improve GPU-offloading efficiency.

drm_hwcomposer structural reorganization proposal

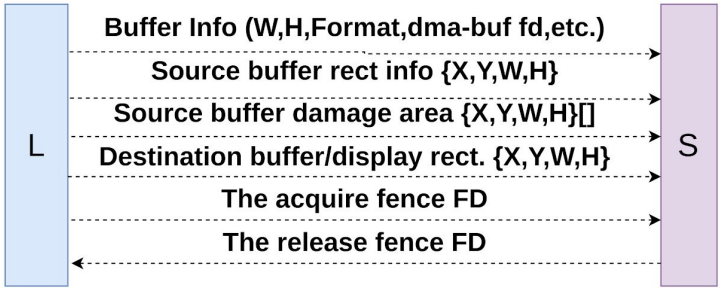


GloDroid team proposal

Component-level generic planning

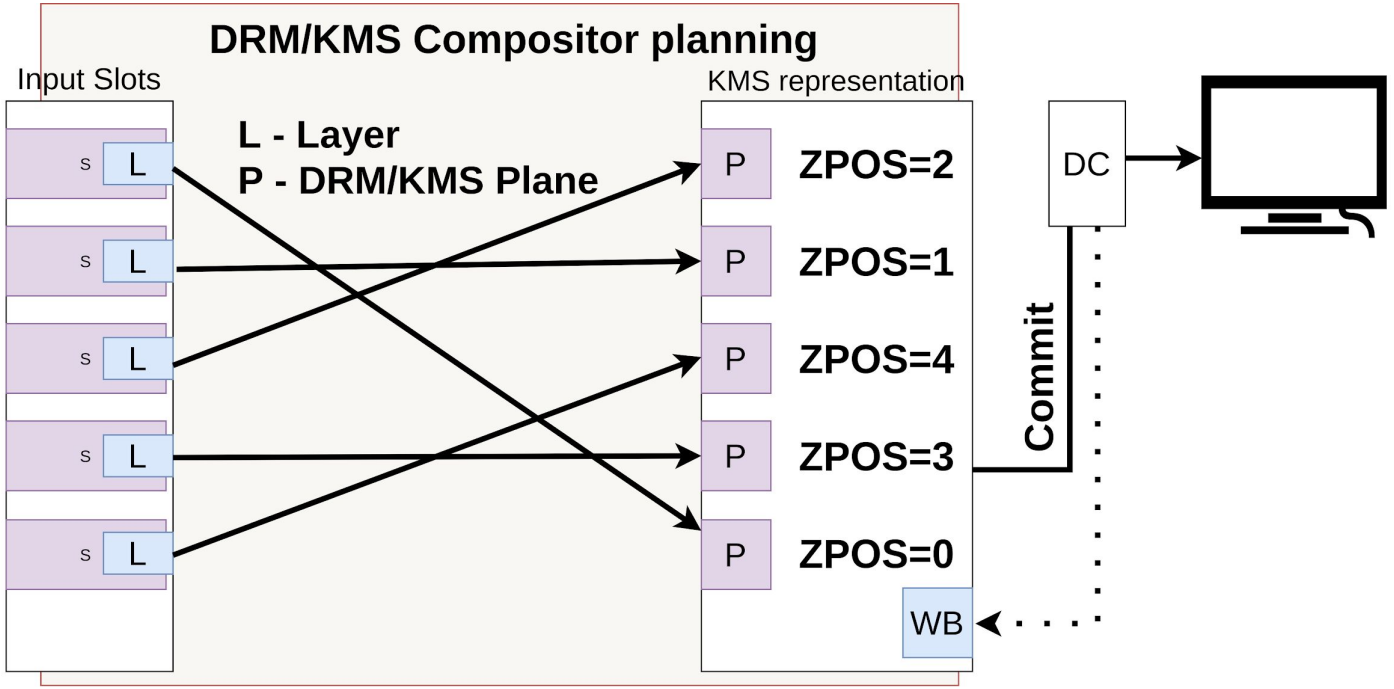


L - Layer
S - Slot
P - DRM planes
I - Interface
PF - Present fence
WB - Writeback



ComponentPlan

```
struct {
  u8 SrcComponentType;
  u8 SrcComponentInstance;
  u8 SrcComponentSlot;
  u8 DstComponentType;
  u8 DstComponentInstance;
  u8 DstComponentSlot;
  void *DstComponentData;
}
```



DC - Display Controller
WB - Writeback buffer (optional)

Constraints:

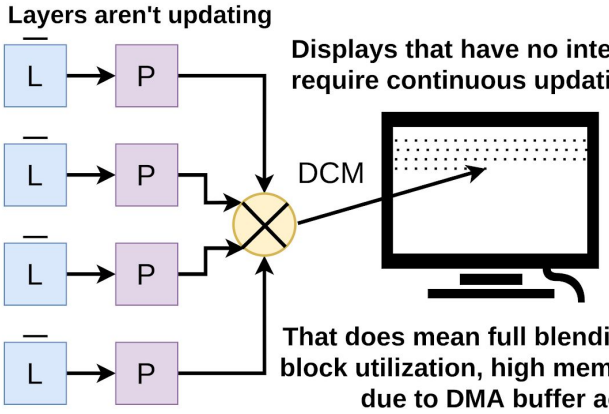
1. Plane should support layer characteristics.
2. Plane can have fixed or floating ZPOS.

* Per plane scaling support is not exposed via properties and should be detected in a different way.

DrmKmsPlan

```
struct {
  u8 SrcSlot;
  u8 DstPlane;
  u8 DstPlaneZpos;
}
```

Optimizations



Displays that have no internal memory require continuous updating every VSYNC

That does mean full blending hardware block utilization, high memory bus load due to DMA buffer access.

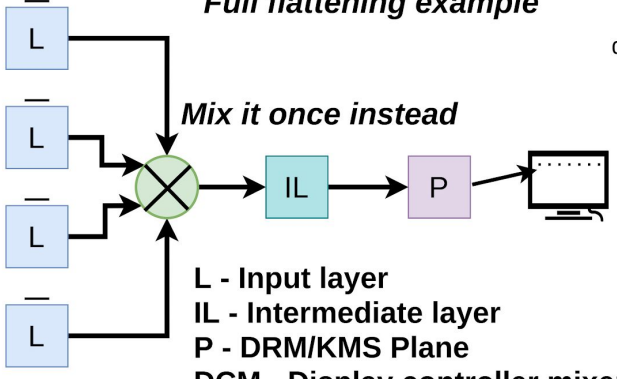
Instead the composition can be squashed into intermediate buffer by any available compositor, thus committing only single plane to the display controller.

This reduces load for the display hardware and memory bus significantly.

As a bonus free shared KMS planes can be reused by other displays (if supported by hardware).

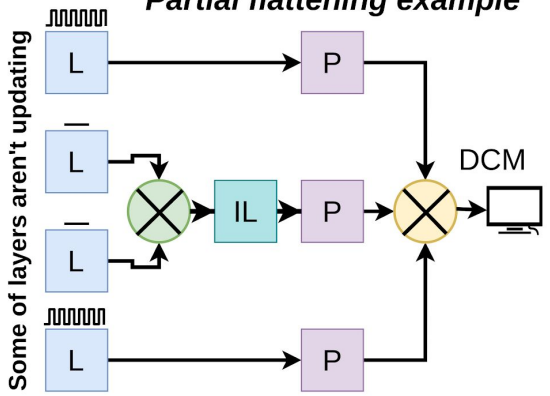
Full flattening example

Available in drm_hwcomposer

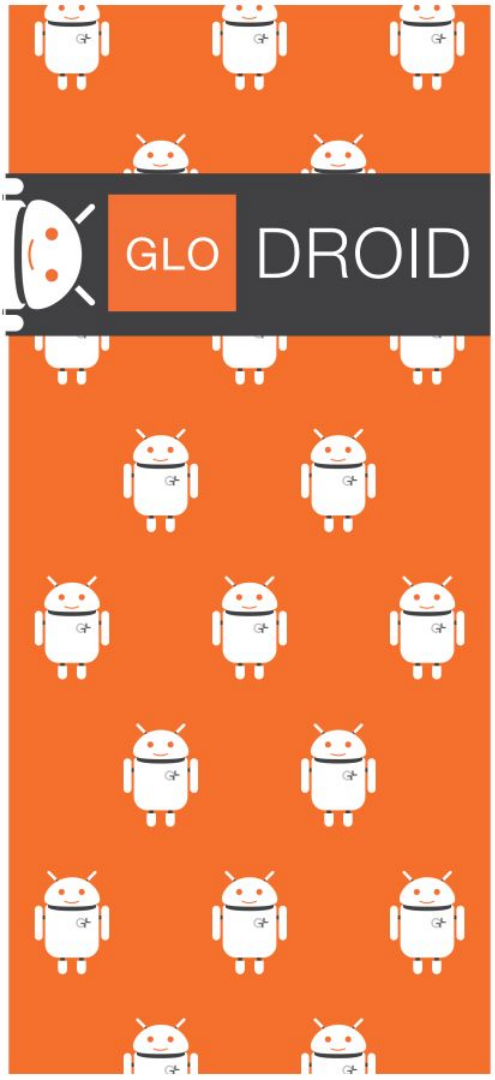


L - Input layer
IL - Intermediate layer
P - DRM/KMS Plane
DCM - Display controller mixer

Partial flattening example

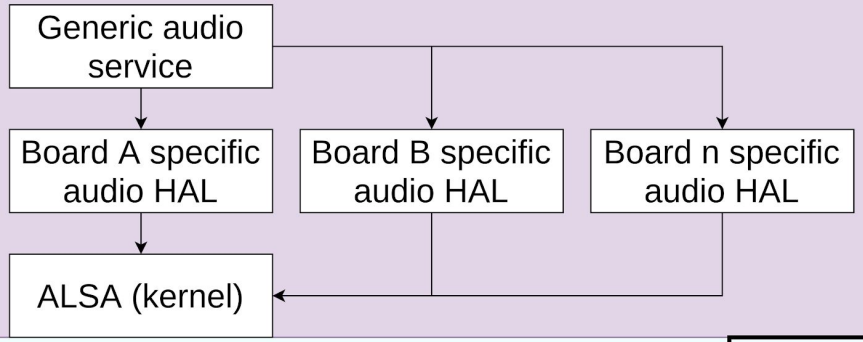


GloDroid team proposal
Should be done in low-prio context to avoid janky frames.



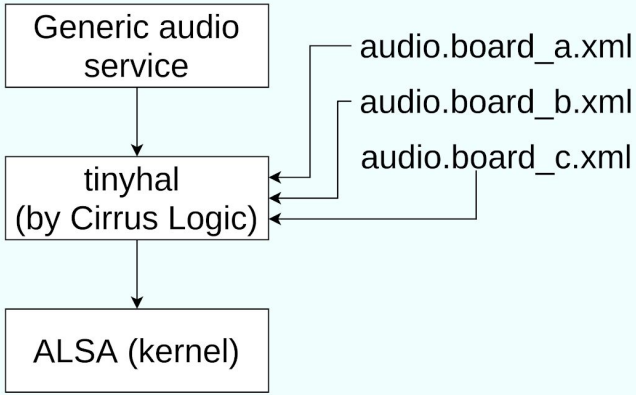
5. XML-based audio configuration

AOSP approach



Board specific audio library usually contains a lot of of common logic and differs only in hard-coded configuration definitions (card number, rate, bps, etc.)

GloDroid approach



https://github.com/CirrusLogic/tinyhal

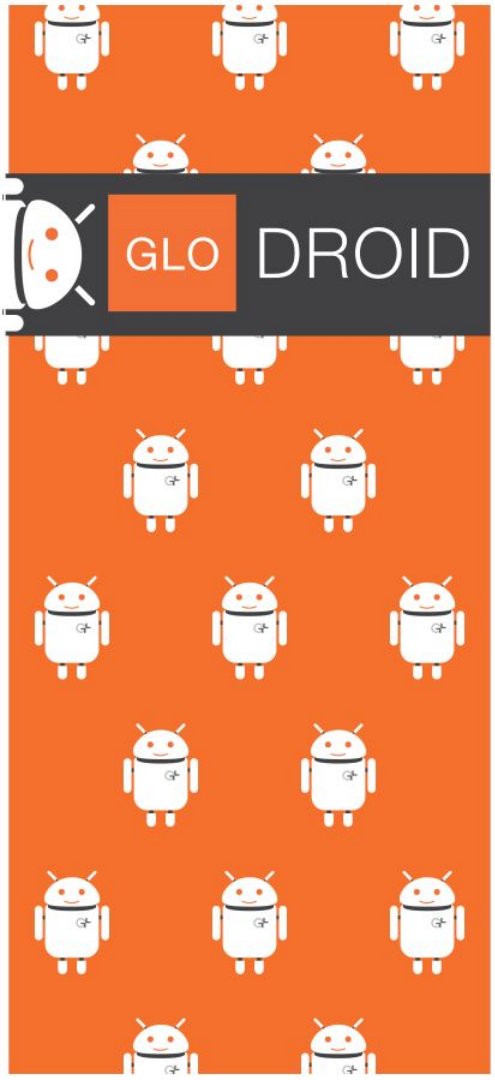
```
<audiohal>
  <mixer name="PinePhone">
    <init>
      <ctl name="AIF1 DA0 Playback Volume" val="160" />
      <ctl name="DAC Playback Volume" val="160" />
      <ctl name="Line Out Playback Volume" val="15" />
      <ctl name="AIF1 Slot 0 Digital DAC Playback Switch" val="1"/>
      <ctl name="DAC Playback Switch" val="1"/>
      <ctl name="Line Out Source Playback Route" val="Mono Differential"/>
    </init>
  </mixer>
  <device name="speaker">
    <path name="on">
      <ctl name="Line Out Playback Switch" val="1"/>
    </path>

    <path name="off">
      <ctl name="Line Out Playback Switch" val="0"/>
    </path>
  </device>

  <stream type="pcm" dir="out" cardname="PinePhone" device="0" rate="48000">
  </stream>

  <stream type="pcm" dir="in" cardname="PinePhone" device="0" rate="48000">
  </stream>
</audiohal>
```

audio.pinephone.xml



6. Meson to Soong bridge

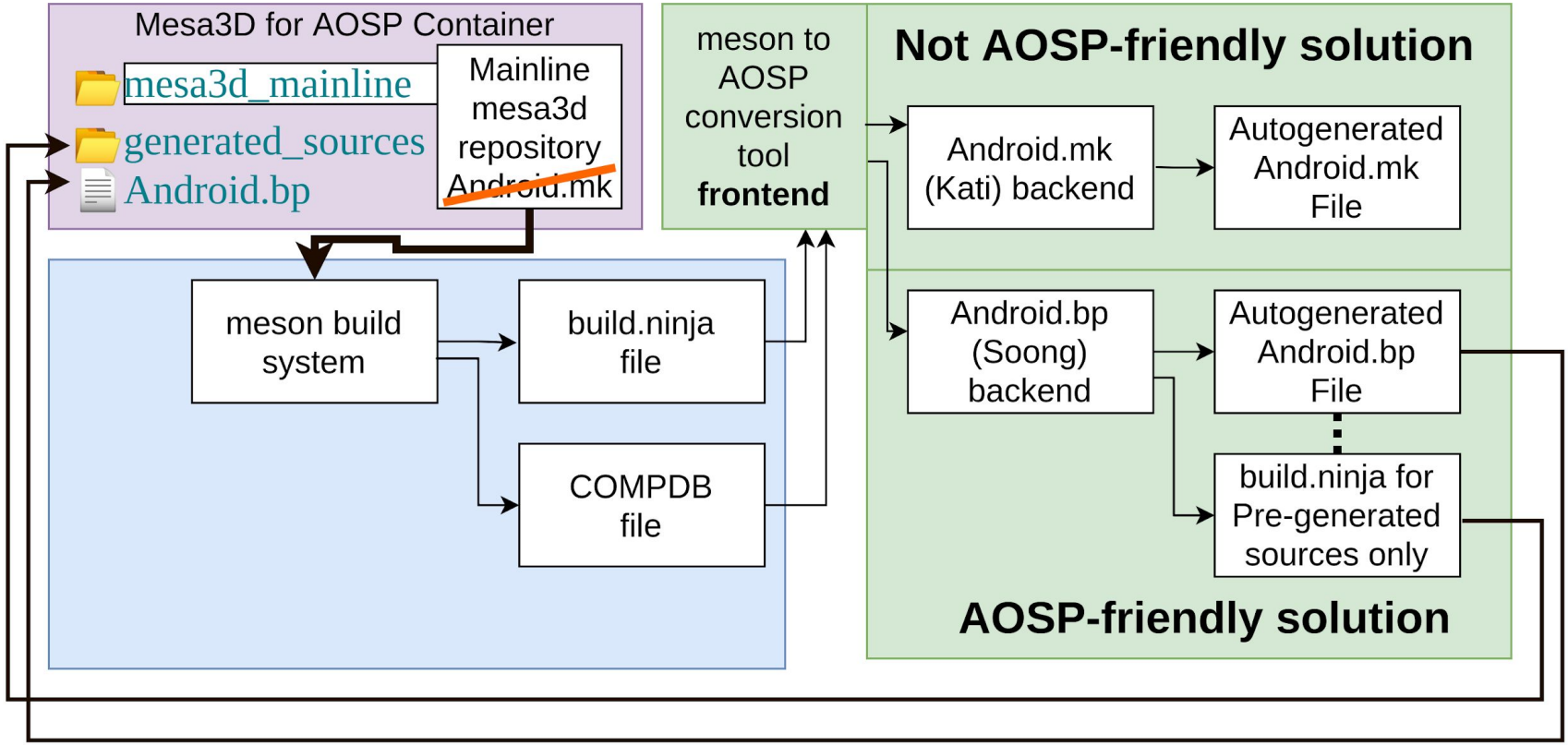
1. Problem description
2. Solution proposal

The problem:
1. Internally AOSP aims to support only Soong build system. This does mean any external component must be soong-compatible and must contain Android.bp file.
2. Currently AOSP can work with make-like (Android.mk) rules, but Google continuously restricts it and want to destage it some day.
3. Both **MESA3D** and **LIBCAMERA** project was initially created for linux and uses meson build system.
4. MESA3D project has set of Android.mks but it is not officially supported by maintainers and very soon became out-of-sync with Meson rules and require continuously fixing. Due to this out-of-sync nature, you can't build the project at every commit. You have to find working points or fix build manually. Bisect is almost impossible in this condition. In addition only limited set of drivers are available for build.
5. Soong build system can't be used by the software that require pre-generation of some of its parts.

Solution #1 (Chromium OS way).
- Build the project externally using Android-NDK and ship it as a binary blob.
Pros: Already working for mesa3d and libcamera.
Cons: Require external build system. Can not be embedded into AOSP. NDK has limited API, for example using Mapper metadata API to get buffer parameters isn't possible yet.

Solution #2 (Linaro/AOSP way).
- Manually adjust the rules, pre-generate required sources and put them together with original sources. Change Android.mk to use prebuilt sources.
Pros: Mesa3d can be embedded into the AOSP in source code form.
Cons: Each update require spending huge amount of work-hours. No bisect-ability and long upgrade cycle. Not mainline-friendly solution.

Solution #3 (GloDroid way).
- Add few hacks on top of mesa3d tree to overcome AOSP restrictions.
Pros: Only small difference with mainline is required (just a few code lines).
Cons: Having hacks is not best option. We are still relying on usually broken Android.mk.



1. **In case top-level build system exists it can trigger regeneration, so it is not necessary to store the pre-generated part.**
2. **For pure-AOSP integration regeneration must be triggered manually, so it is important to store pre-generated output in the repository (better to create separate repository for that).**

GLO DROID

THANK YOU

https://glodroid.github.io

Acknowledgments

Google for their awesome OS

Linaro and John Stultz for support

MESA3D and 3d-graphics experts:
Alyssa Rosenzweig
Qiang Yu
Vasily Khoruzhick
many other

Chromium OS team
Android-x86 team
libcamera team

GloDroid external contributors:
Sunxi platform contributor:
Icenowy Zheng
Amlogic platform contributor:
Neil Armstrong

FOSS Android HALs authors/contributors
Rob Herring Sean Paul Richard Fitzgerald
many other

GlobalLogic and GloDroid team:
Managing
Oleksiy Oguy Maryna Sergiyenko
Engineering
Roman Stratiienko
Daniil Petrov Roman Kovalivskyi
Matvii Zorin Maksym Prymierov
Aleksandr Bulyshchenko
Design **PR**
Mariia Herton Ellina Medynska

TL Lim and **Pine64** community

GLO DROID

Other open source communities

Roman Stratiienko, GlobalLogic Ukraine, 2021