Graphics acceleration on Replicant

David Ludovino (@dllud) Ricardo Cabrita (@GrimKriegor)*

NLnet - NGI0 PET Fund

Saturday 27th July, 2019

*with great support from Joonas Kylmälä (@Putti)

Motivation

All supported devices lack a free software GPU driver.

Replicant 6 relies on libAGL which uses the libpixelflinger software render (both deprecated since 2013).

Motivation

Lack of GLES 2.0 leads some critical applications to crash (e.g. Firefox)

Rendering performance has degraded throughout Android versions.

Replicant relies on patches to the Android framework to make things like the camera application work.

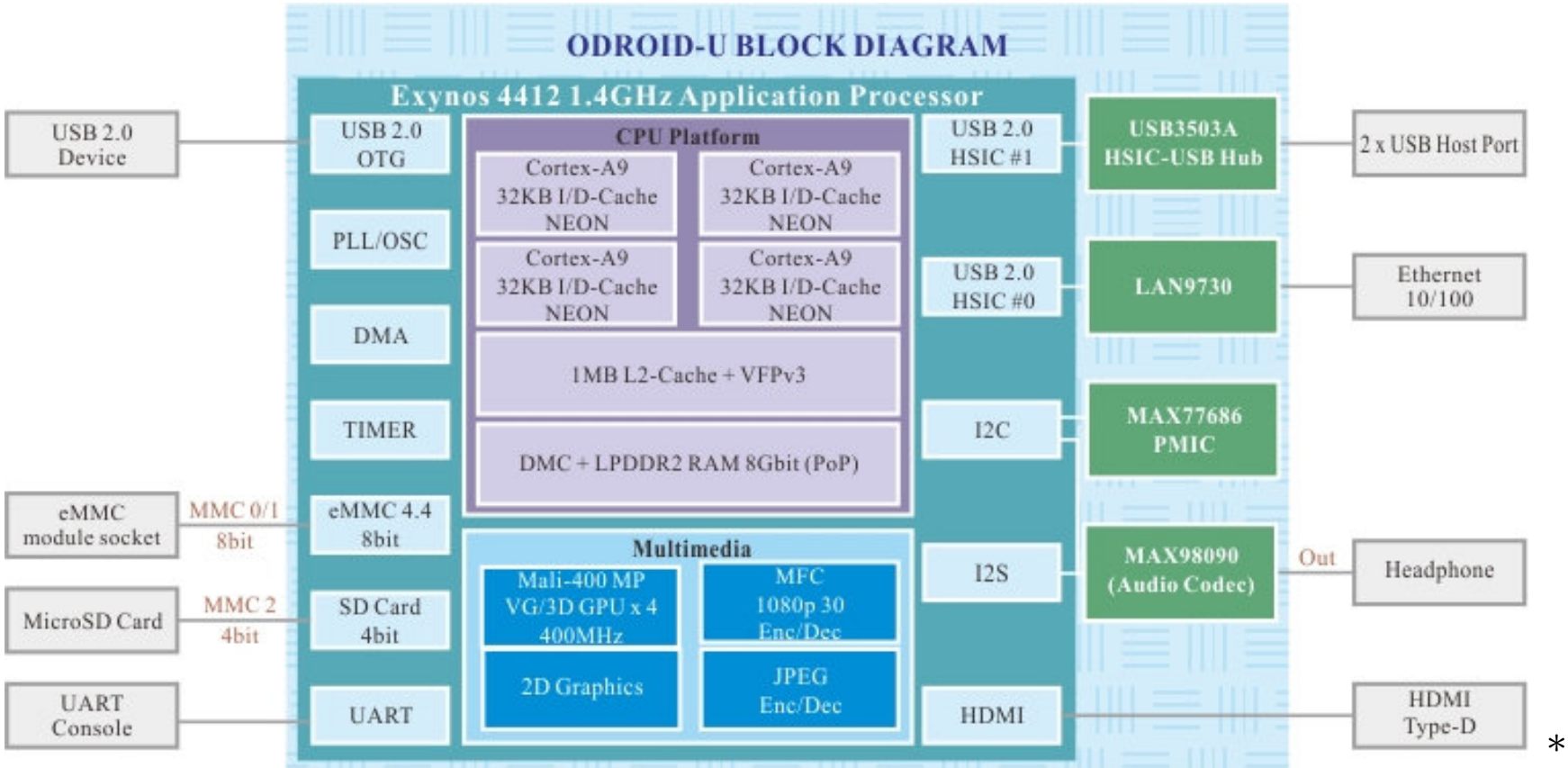
Objectives

Put together a graphics stack:

- Compatible with Android 9's HALs.
- Provides at least GLES 2.0.
- Flexible enough to do rendering with both Mesa and SwiftShader.
- Uses hardware rendering on devices with a free GPU driver.

Graphics hardware architecture

Graphics hardware architecture — Exynos 4412 SoC components

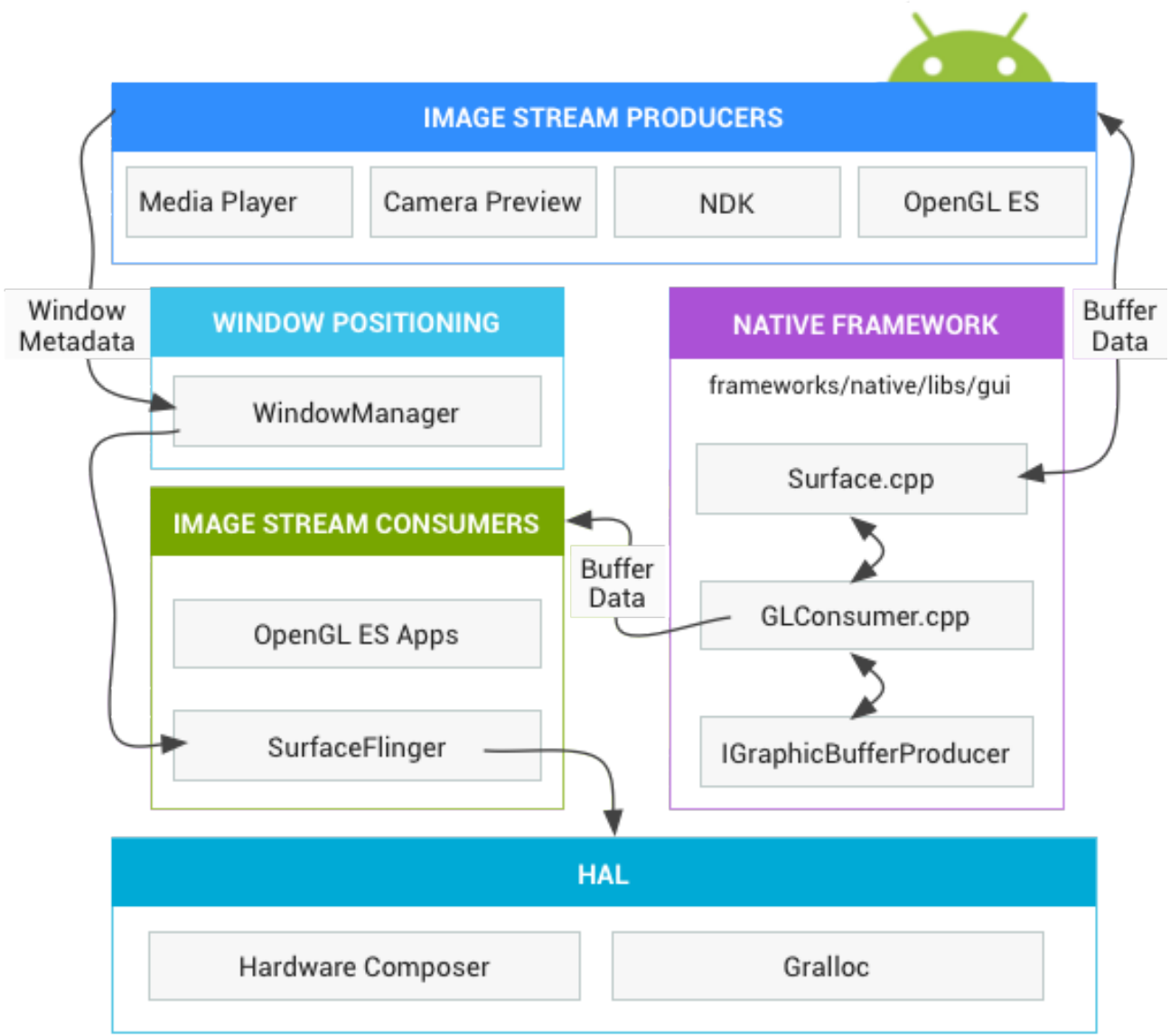


*

*Source: Hardkernel Co., Ltd.

Graphics software architecture

Graphics software architecture — Android 9



*

*Source: Android Open Source Project under CC BY 4.0

Graphics software architecture — Replicant 9 HWC HAL

Hardware Composer HAL: drm_hwcomposer

- Supports HWC2 HAL.
- Works on top of DRM (can use hardware composing acceleration).
- Under active maintenance (hosted by freedesktop.org).
- Also used by Android-x86.

Graphics software architecture — Replicant 9 Gralloc HAL

Gralloc HAL: gbm_gralloc

- Implements Android Gralloc HAL API version 0 and 1.
- Compatible with drm_hwcomposer.
- Compatible with Mesa.
- Uses Mesa's GBM (Generic Buffer Management) for buffer allocation through libgbm. GBM then calls DRM.
- Supports PRIME fd.
- Originally by Rob Herring, now maintained by Android-x86.

Graphics software architecture — Replicant 9 GLES

OpenGL ES renderer: Mesa

- Support for both software and hardware rendering.
- Big and active community (maintained for years to come).

Mesa driver: kms_swrast

- Uses any Gallium software renderer as backend (softpipe or llvmpipe).
- Does mode setting through the kernel (KMS).

Alternative GLES renderer: SwiftShader

- Optimized for ARM CPUs.
- Has Vulkan software rendering.

Implementation

Implementation — drm_hwcomposer + gbm_gralloc

Initially both required the use of the drm/exynos master node

1. DRM Auth hack (both on /dev/dri/card0)
2. DRM vGEM inclusion (gbm_gralloc on /dev/dri/card1)
3. DRM allow dumb buffers (gbm_gralloc on /dev/dri/renderD128)

At the time we had some graphical glitches we thought were due to inter driver memory sync.

Running on the same driver does not require memory synchronization.

Allows drm/exynos to allocate memory where adequate according to the type of plane (primary, overlay or cursor).

Implementation — Allow kms_swrast to use drm/exynos

Small tweak: Add exynos to the kms_swrast list on external_mesa3d.

How to upstream this?

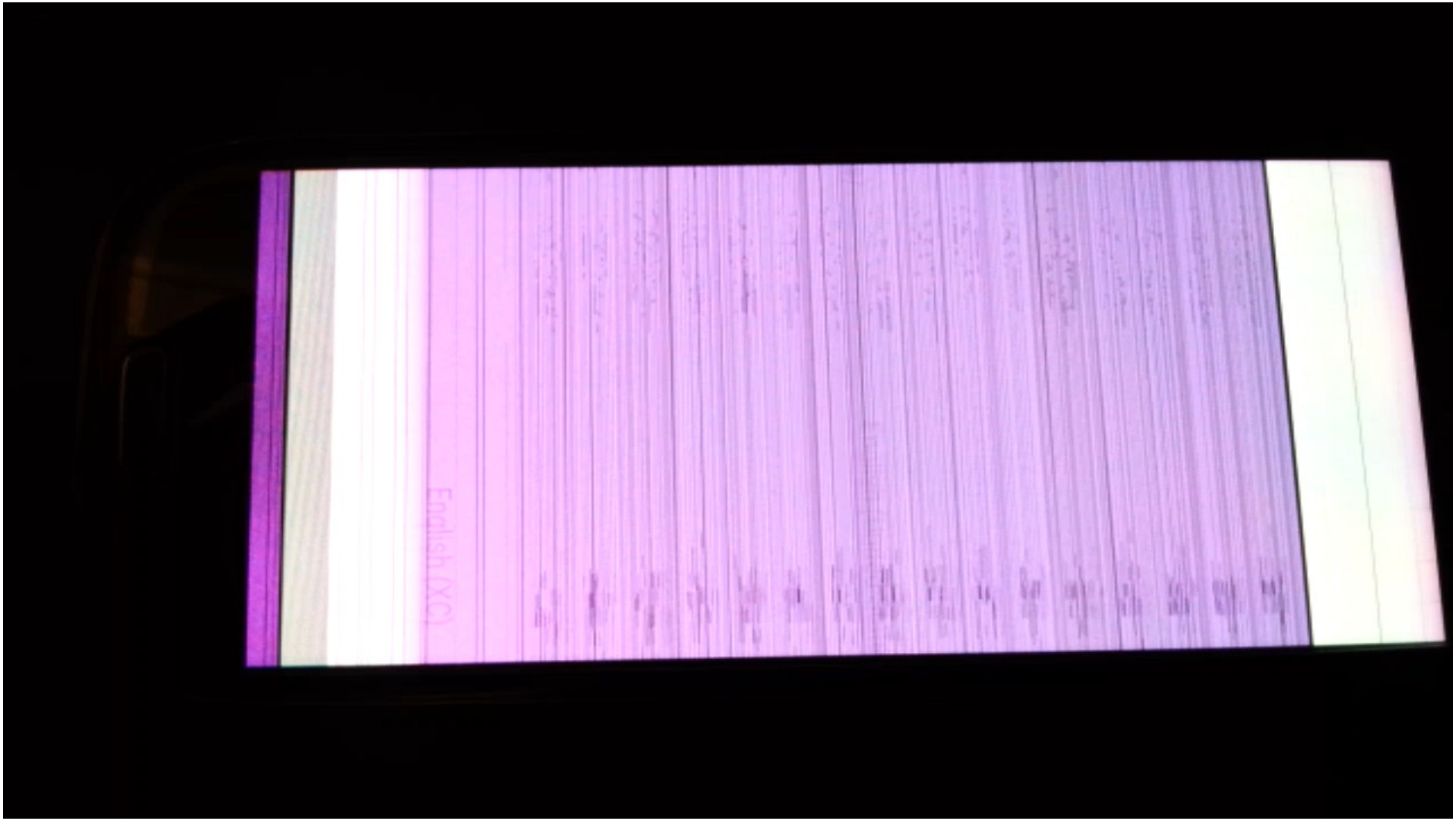
Implementation — HW planes + devfreq

We were then using kms_swrast with the softpipe backend.

Enabling DRM hardware planes was another attempt at squeezing some extra performance out of the hardware.

However this led to some interesting shenenigans.

Implementation — HW planes + devfreq



Implementation — HW planes + devfreq

Tentative explanation by ahajda:

1. devfreq lowers display clock frequencies too aggressively.
2. DMA transfers of overlays are too slow and result in screen corruption.

Temporary fix: disable devfreq.

Implementation — llvmpipe

kms_swrast with softpipe was unbearably slow, even with DRM HW planes enabled.

Required:

- Finding out what Android-x86 had previously done.
- Porting it to Android 9.

Implementation — llvmpipe

android: Enable llvmpipe when using the swrast driver
https://gitlab.freedesktop.org/mesa/mesa/merge_requests/1403

android: Fix build with LLVM for Android 9
https://gitlab.freedesktop.org/mesa/mesa/merge_requests/1402

Implementation — SwiftShader

Required:

- UDIV and SDIV instruction emulation (in the kernel).
- Android emulator composer: ranchu.
- Default Android gralloc.

Proved to be 1.5 - 2x faster than llvmpipe.

Performance

SwiftShader $>$ llvmpipe $>$ softpipe

Performance — SwiftShader with LLVM

We managed to find a SwiftShader revision that uses LLVM as a backend instead of SubZero and is still compatible with our frameworks_native.

```
Lineage 16 / Android 9 / Replicant 9
SurfaceFlinger: OpenGL ES 2.0 SwiftShader 4.0.0.4
```

```
Android Q
fde88d96a58b92beab76035393b3acd849445160
Default to LLVM 7.0 JIT in Android build
SurfaceFlinger: OpenGL ES 3.0 SwiftShader 4.1.0.5
```

No noticeable performance difference.

Performance — Why is Replicant 6 much faster?

Emulator switches? NO
ro.kernel.qemu=1

High end graphics options? NO
ro.config.avoid_gfx_accel=1

Pixel format (RGB565)? Paul says YES (very hardware dependent)

Future

Future — RGB565 across entire stack

- gbm_gralloc
- drm_hwcomposer
- drm/exynos

All using RGB565.

Potential performance breakthrough.

If so, how to futureproof this?

Future — devfreq: which device needs clock boost?

1. Test each device independently through sysfs.
2. Identify which one is causing the corruption (tip: FIMD/LCD path).
3. Boost clock/voltage on userspace or kernel config.
4. Re-enable devfreq.
5. Workout patch to fix upstream.

Future — SwiftShader + drm_hwcomposer

Advantages (vs ranchu):

- hardware planes
- DRM node instead of direct framebuffer

Future — Profiling, benchmarks and conformance

Profiling: turn on profiling switch on Mesa + simpleperf?

Benchmarks: ask Android-x86 (proprietary?)

Conformance: dEQP (drawElements Quality Program) and piglit

Future — 2D acceleration on drm_hwcomposer

Software-based: Pixman (has ARM NEON fast path)

Hardware-based: Exynos FIMG2D (Fully Integrated Mobile Graphics 2D)

Future — SDIV/UDIV on compiler-rt

- Patch with kernel emulation of SDIV/UDIV is not optimized.
- Try compiler-rt's builtins instead.

Future — ARM NEON on llvmpipe

ARM NEON: SIMD instructions

How to use:

- Tune **auto-vectorization** on LLVM: easy to try; possible to upstream.

Borrow ideas from Pixman, Skia and libyuv (all these have NEON fast paths).

Future — ARM NEON on llvmpipe

ARM NEON: SIMD instructions

How to use:

- Tune **auto-vectorization** on LLVM: easy to try; possible to upstream.
- Neon assembly: too cumbersome (e.g. manual register allocation).

Borrow ideas from Pixman, Skia and libyuv (all these have NEON fast paths).

Future — ARM NEON on llvmpipe

ARM NEON: SIMD instructions

How to use:

- Tune **auto-vectorization** on LLVM: easy to try; possible to upstream.
- Ne10 library: easy to use; difficult to upstream (requires new deps).
- Neon assembly: too cumbersome (e.g. manual register allocation).

Borrow ideas from Pixman, Skia and libyuv (all these have NEON fast paths).

Future — ARM NEON on llvmpipe

ARM NEON: SIMD instructions

How to use:

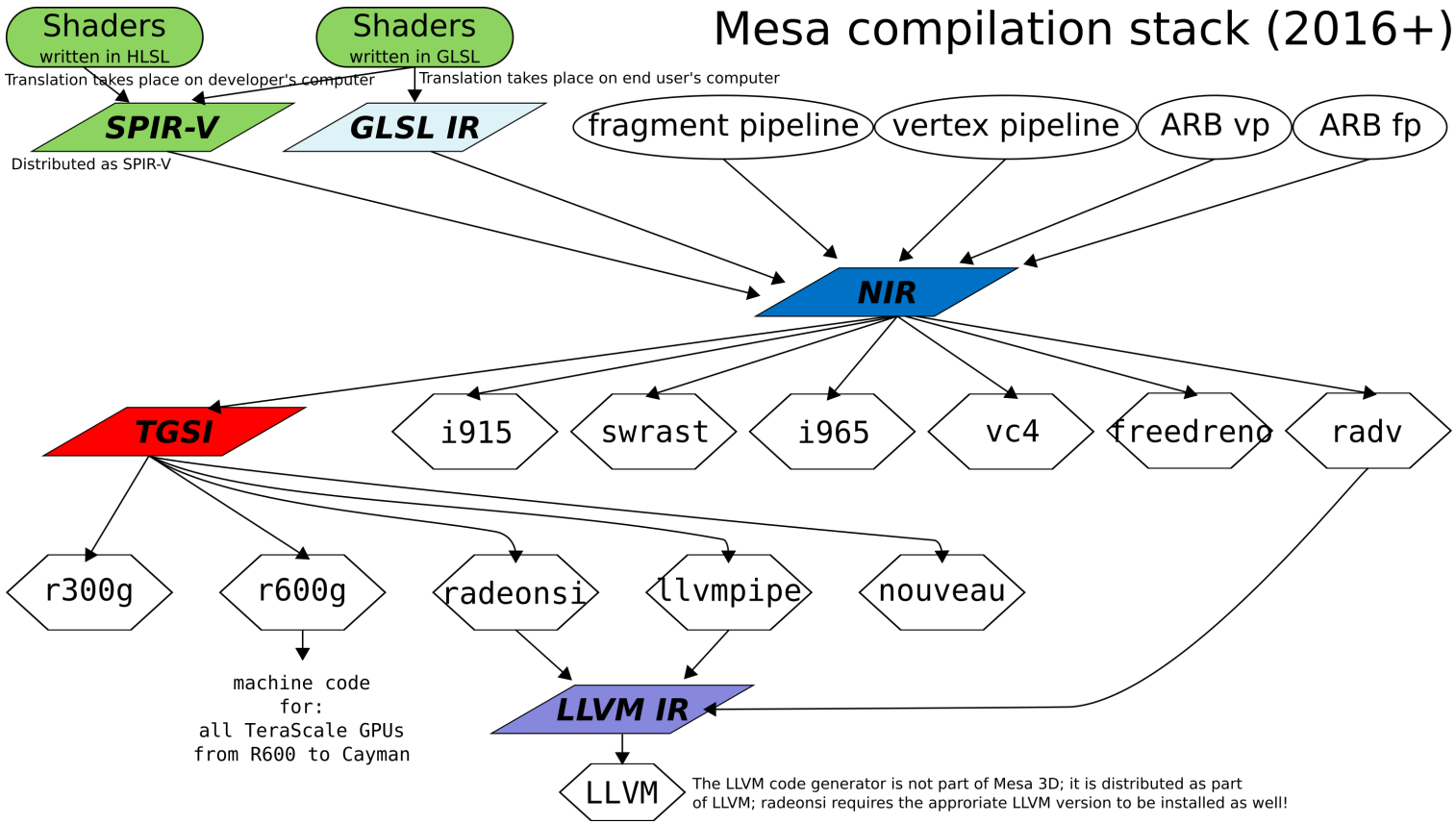
- Tune **auto-vectorization** on LLVM: easy to try; possible to upstream.
- Ne10 library: easy to use; difficult to upstream (requires new deps).
- **Neon intrinsics**: nice compromise between performance and code complexity; possible to upstream.

```
#include <arm_neon.h>
uint8x8_t va, vb, vr;
vr = vadd_u8(va, vb);
```

- Neon assembly: too cumbersome (e.g. manual register allocation).

Borrow ideas from Pixman, Skia and libyuv (all these have NEON fast paths).

Future — ARM NEON on llvmpipe



*

How to use intrinsics when llvmpipe must output LLVM IR?

Can LLVM IR contain ARM NEON assembly code?

*Source: ScotXW on Wikimedia under CC0

Future — Lima

The holy grail.

Quite active now. New commits every week.
No idea of current compliance (asked devs to update `features.txt`).

Planned approach: offload implemented GL operations to Lima.

- Where in the stack should we intercept GL operations? GLSL IR? TGSI?
- Won't the overhead of interception, introspection and dispatch kill any performance gains?

Questions?*

*Ask Putti the hard ones. xD