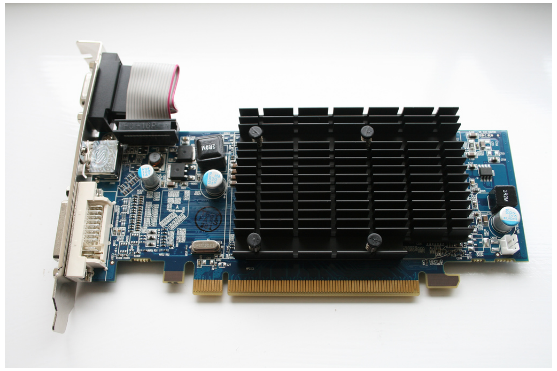
Embedded Graphics Drivers in Mesa

Neil Roberts

Overview

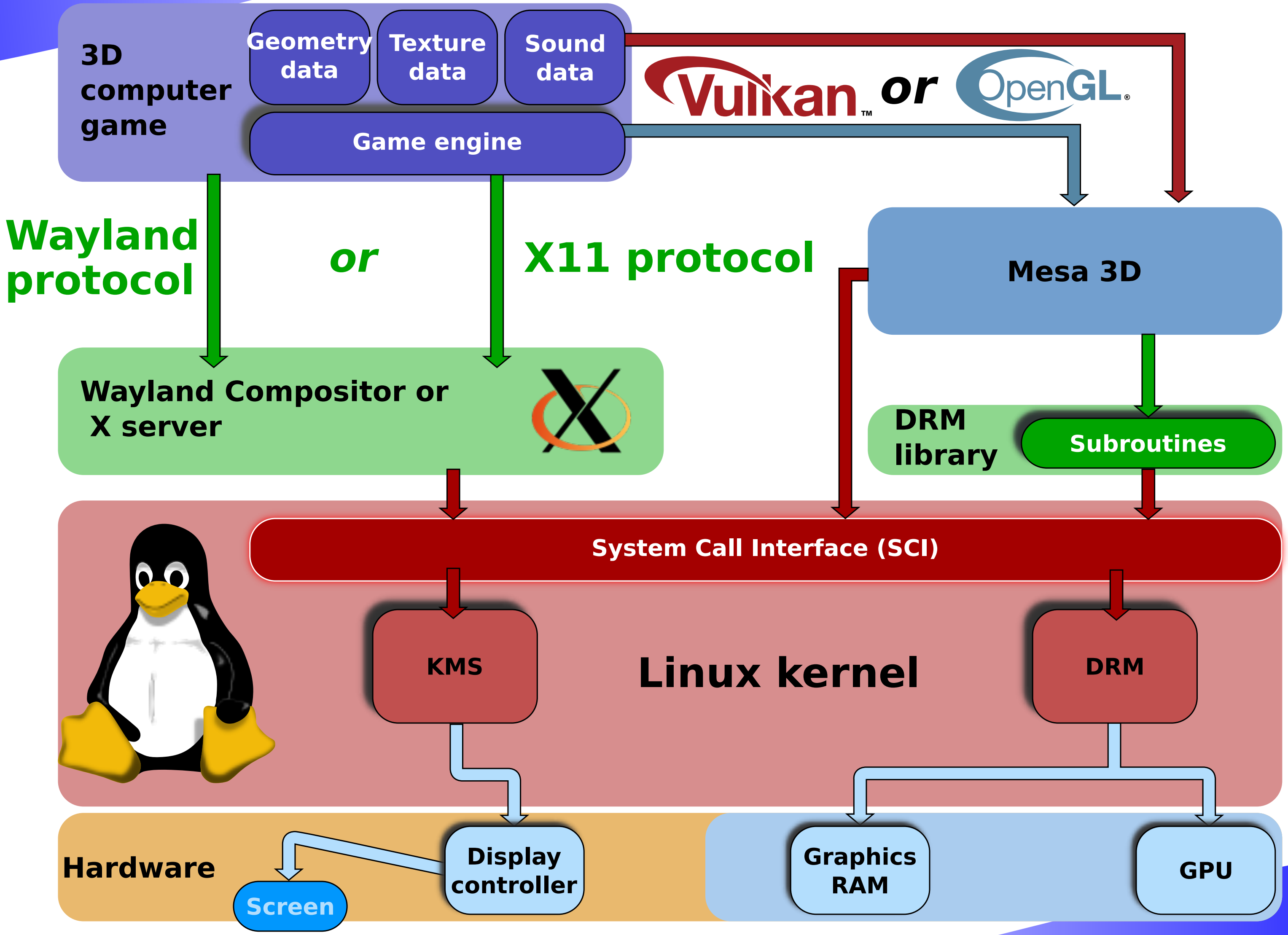
About GPUs

- It is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Wikipedia.



- They are becoming increasingly general purpose processors that can run programs (shaders).
- They are highly threaded and typically use SIMD to operate on multiple inputs at the same time.
- Still contain fixed function pieces for graphics-specific functions:
  - Texture sampling
  - Primitive assembly
  - etc

Linux graphics stack


3D computer game
Geometry data
Texture data
Sound data
Game engine
Vulkan or OpenGL
Wayland protocol
or
X11 protocol
Mesa 3D
Wayland Compositor or X server
DRM library
Subroutines
System Call Interface (SCI)
KMS
Linux kernel
DRM
Hardware
Screen
Display controller
Graphics RAM
GPU

Graphics APIs

OpenGL®

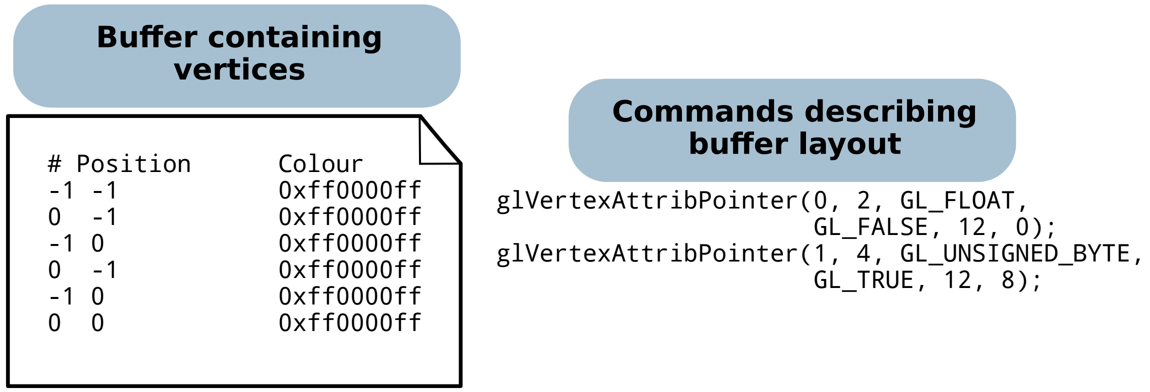
- OpenGL 1.0 was released in January 1992 by Silicon Graphics (SGI).
- Based around SGI hardware of the time which had very fixed functionality.
- Eg, explicit API to draw a triangle with a colour:

```
/* Set a blue colour */
glColor3f(0.0f, 0.0f, 1.0f);
/* Draw a triangle, describing its points */
glBegin(GL_TRIANGLES);
 glVertex3f(0.0f,1.0f,0.0f);
 glVertex3f(-1.0f,-1.0f,0.0f);
 glVertex3f(1.0f,-1.0f,0.0f);
glEnd();
```

- In 2004 OpenGL 2.0 was released.
- Introduced the concept of shaders.
- Can now influence the rendering with programs called shaders.
- Eg, choose a colour programatically:

```
void main()
{
        /* Choose the colour based on the X-position of the pixel */
        gl_FragColor = vec4(gl_FragCoord.x * 0.008 - 1.0, 0.0, 0.0, 1.0);
}
```

- In later versions of GL more and more functionality is moved into the programmable shaders.
- Much more programmable, much less fixed-function.
- Inputs are more often given in buffers rather than via API calls.
- Eg, vertex data now in a buffer:



```
# Position       Colour
-1 -1            0xff0000ff
0  -1            0xff0000ff
-1 0             0xff0000ff
0  -1            0xff0000ff
-1 0             0xff0000ff
0  0             0xff0000ff
```

```
glVertexAttribPointer(0, 2, GL_FLOAT,
                      GL_FALSE, 12, 0);
glVertexAttribPointer(1, 4, GL_UNSIGNED_BYTE,
                      GL_TRUE, 12, 8);
```

OpenGL ES

- Simplified version of OpenGL targetting embedded devices.
- Removes most of the legacy cruft and things that are hard to implement in hardware.
- Is increasingly similar to modern versions of OpenGL which also try to deprecate old functionality.

Vulkan™

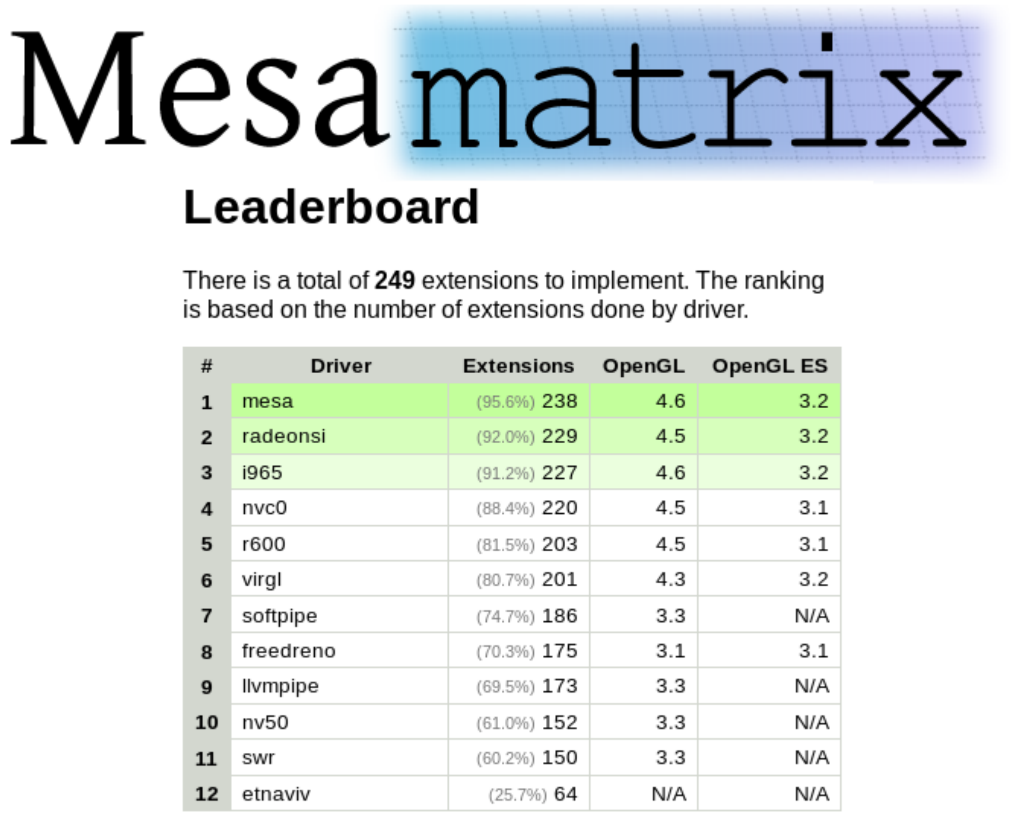
- Vulkan 1.0 released in 2016
- Clean break from legacy OpenGL
- Much less driver overhead
- Everything is specified in buffers
- The application has the responsibility to manage buffers and synchronisation.
- Harder to use but allows applications to exploit the hardware better
- Suitable for both embedded and desktop hardware

Mesa

- Open-source implementation of the OpenGL and Vulkan specifications for a variety of hardware on user-space as a library.
- The Mesa project was originally started by Brian Paul.
    - Version 1.0 released in February 1995.
    - Originally used only software rendering
    - Now has support for many different hardware devices
    - Current version is 18.0.

- There are drivers for:
  - Intel (i965, i915, anv)
  - AMD (radv, radeonsi, r600)
  - NVIDIA (nouveau)
  - Imagination Technologies (imx)
  - Broadcom (vc4, vc5)
  - Qualcomm (freedreno)
  - Software renderers (classic swrast, softpipe, llvmpipe, OpenSWR)
  - VMware virtual GPU
  - Etc

- Supports:
  - OpenGL 4.6
  - OpenGL ES 3.2
  - Vulkan 1.1
- All are the latest versions
- Caveat: not all drivers support the latest version



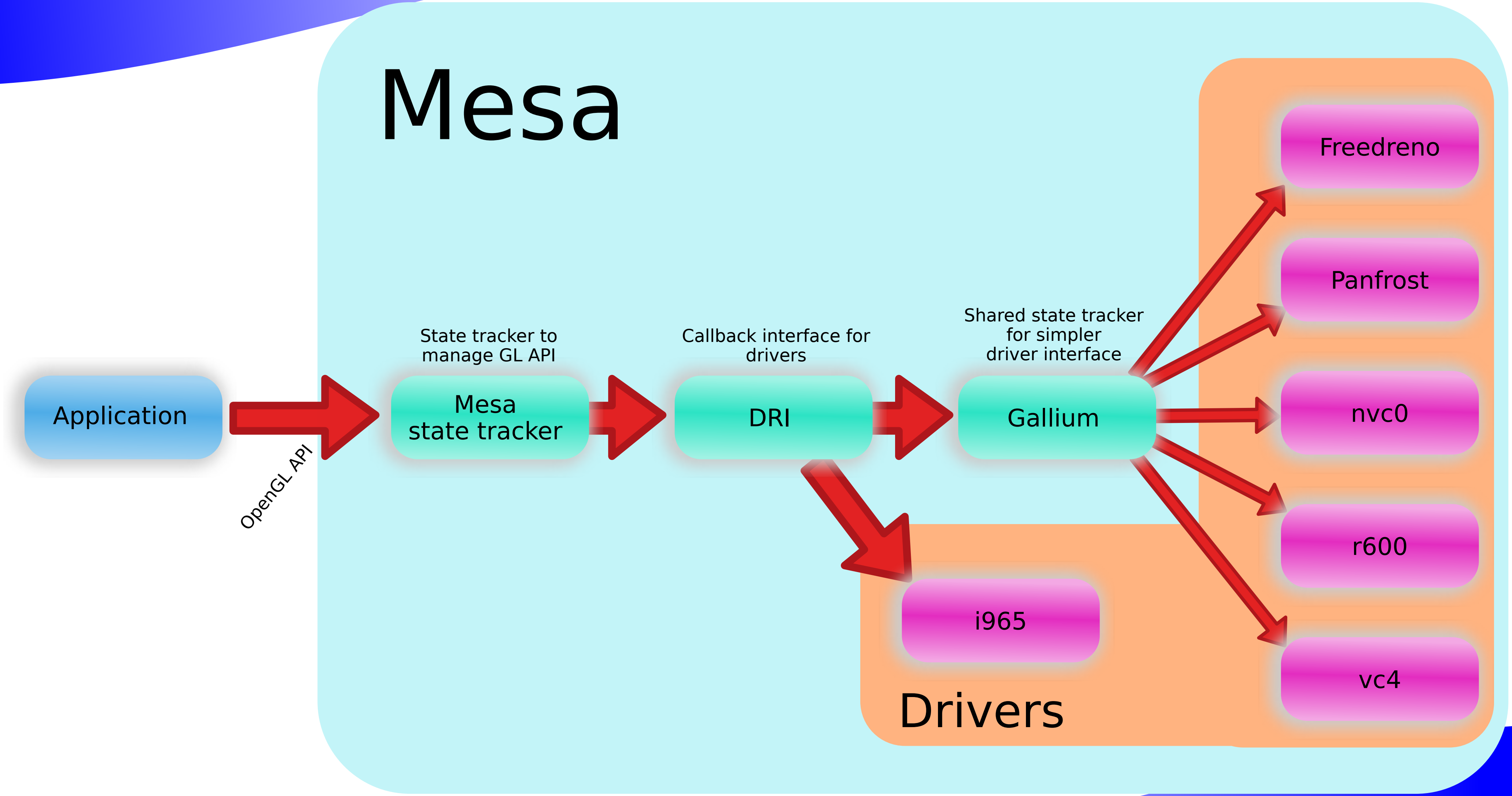
Mesamatrix

Leaderboard

There is a total of **249** extensions to implement. The ranking is based on the number of extensions done by driver.

#	Driver	Extensions	OpenGL	OpenGL ES
1	mesa	(95.6%) 238	4.6	3.2
2	radeonsi	(92.0%) 229	4.5	3.2
3	i965	(91.2%) 227	4.6	3.2
4	nvc0	(88.4%) 220	4.5	3.1
5	r600	(81.5%) 203	4.5	3.1
6	virgl	(80.7%) 201	4.3	3.2
7	softpipe	(74.7%) 186	3.3	N/A
8	freedreno	(70.3%) 175	3.1	3.1
9	llvmpipe	(69.5%) 173	3.3	N/A
10	nv50	(61.0%) 152	3.3	N/A
11	swr	(60.2%) 150	3.3	N/A
12	etnaviv	(25.7%) 64	N/A	N/A

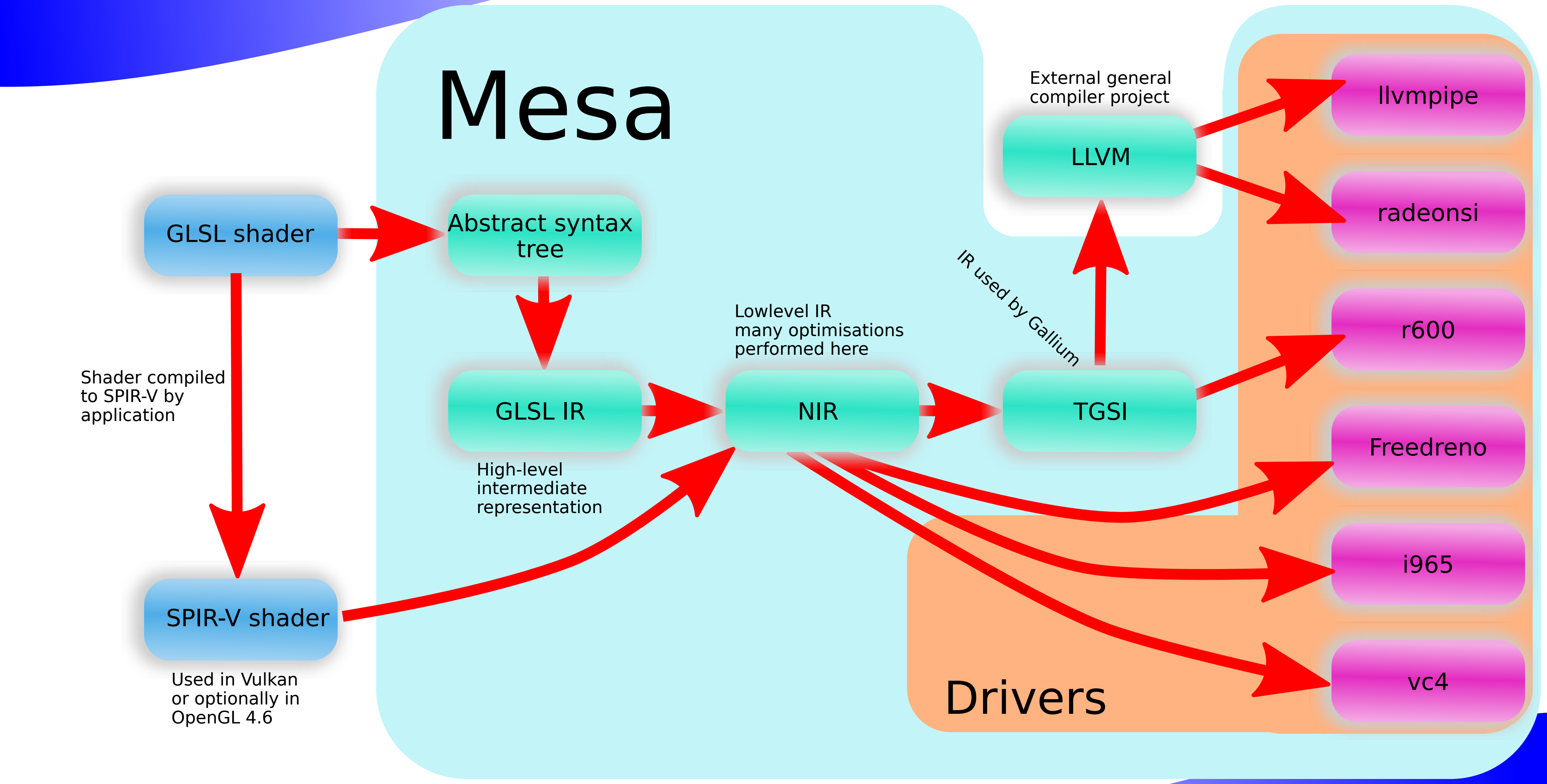
Architecture of Mesa


Mesa
Application
OpenGL API
State tracker to manage GL API
Mesa state tracker
Callback interface for drivers
DRI
Shared state tracker for simpler driver interface
Gallium
Freedreno
Panfrost
nvc0
r600
vc4
i965
Drivers

- Mesa has a loader that selects the driver by asking for the vendor id, chip id... from the kernel driver via DRM.
- There is a map of PCI IDs and user-space Mesa drivers.
- When it is found, Mesa loads the respective driver and sees if the driver succeeds
- In case of failure, the loader tries software renderers.
- It is possible to force software renderer
  - LIBGL ALWAYS SOFTWARE=1

- The GL API is filtered through the Mesa state tracker into a simpler set of callbacks into the driver.
  - This handles many things such as GL’s weird object management.
  - Unifies different APIs from different versions of GL.
- For the i965 Intel driver, these callbacks are handled directly.
- For most other drivers, Gallium is used as an extra layer.
  - This handles even more state tracking such as caching state objects.
  - Drivers have even less code to implement.

Compiler architecture


Mesa
GLSL shader
Abstract syntax tree
Shader compiled to SPIR-V by application
GLSL IR
High-level intermediate representation
Lowlevel IR many optimisations performed here
NIR
IR used by Gallium
TGSI
External general compiler project
LLVM
SPIR-V shader
Used in Vulkan or optionally in OpenGL 4.6
llvmpipe
radeonsi
r600
Freedreno
i965
vc4
Drivers

GLSL example

```
uniform vec4 args1, args2;

void main()
{
        gl_FragColor = log2(args1) + args2;
}
```

GLSL IR

```
GLSL IR for native fragment shader 3:
(
(declare (location=2 shader_out ) vec4 gl_FragColor)
(declare (location=0 uniform ) vec4 args1)
(declare (location=1 uniform ) vec4 args2)
( function main
  (signature void
    (parameters)
    (
      (assign  (xyzw)
               (var_ref gl_FragColor)
               (expression vec4 + (expression vec4 log2 (var_ref args1) )
                                  (var_ref args2) ) )
    ))
)
)
```

NIR

```
impl main {
        block block_0:
        /* preds: */
        vec1 32 ssa_0 = load_const (0x00000000 /* 0.000000 */)
        vec4 32 ssa_1 = intrinsic load_uniform (ssa_0) (0, 16, 160)
        vec1 32 ssa_2 = flog2 ssa_1.x
        vec1 32 ssa_3 = flog2 ssa_1.y
        vec1 32 ssa_4 = flog2 ssa_1.z
        vec1 32 ssa_5 = flog2 ssa_1.w
        vec4 32 ssa_6 = intrinsic load_uniform (ssa_0) (16, 16, 160)
        vec1 32 ssa_7 = fadd ssa_2, ssa_6.x
        vec1 32 ssa_8 = fadd ssa_3, ssa_6.y
        vec1 32 ssa_9 = fadd ssa_4, ssa_6.z
        vec1 32 ssa_10 = fadd ssa_5, ssa_6.w
        vec4 32 ssa_11 = vec4 ssa_7, ssa_8, ssa_9, ssa_10
        intrinsic store_output (ssa_11, ssa_0) (4, 15, 0, 160)
        /* succs: block_1 */
        block block_1:
}
```

Intel i965 instruction set

```
   START B0 (54 cycles)
math log(16)    g3<1>F          g2<0,1,0>F      null<8,8,1>F
math log(16)    g5<1>F          g2.1<0,1,0>F    null<8,8,1>F
math log(16)    g7<1>F          g2.2<0,1,0>F    null<8,8,1>F
math log(16)    g9<1>F          g2.3<0,1,0>F    null<8,8,1>F
add(16)         g120<1>F        g3<8,8,1>F      g2.4<0,1,0>F
add(16)         g122<1>F        g5<8,8,1>F      g2.5<0,1,0>F
add(16)         g124<1>F        g7<8,8,1>F      g2.6<0,1,0>F
add(16)         g126<1>F        g9<8,8,1>F      g2.7<0,1,0>F
sendc(16)       null<1>UW       g120<8,8,1>UD   0x90031000
                render MsgDesc: RT write SIMD16 LastRT mlen 8 rlen 0
   END B0
```

Embedded drivers

Freedreno

Panfrost

Broadcom

Thanks.

Questions?