

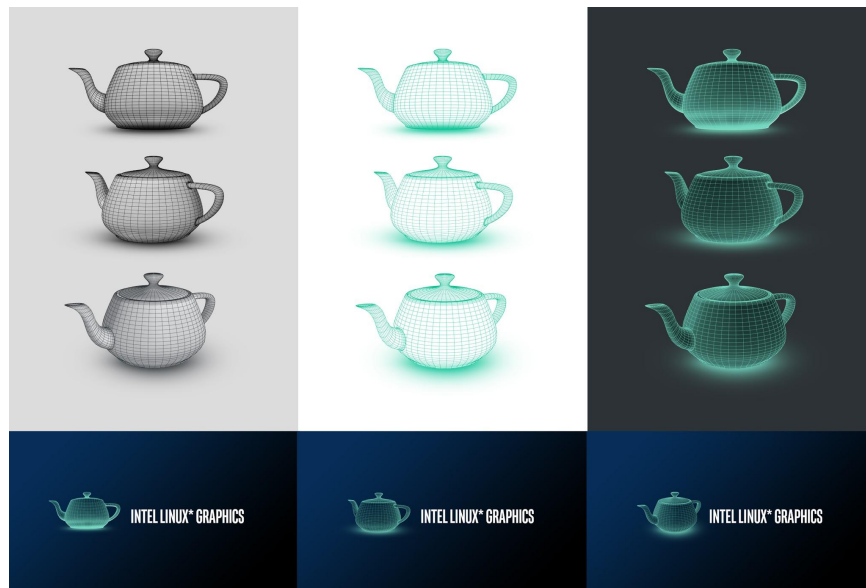
Framebuffer Modifiers

Supporting end-to-end graphics compression

Ben Widawsky

About Me

- Worked on all parts of graphics and other driver stacks
- Avid Buffer Modifier
- Motivated by disparity with closed implementations
 - If they can do it; damn it, we can too.



Summarizing the Work

	Linux DRM <ul style="list-style-type: none"> • blobifier • AddFB2 • multi-plane 	Linux i915 <ul style="list-style-type: none"> • blobifier • AddFB2 	Mesa <ul style="list-style-type: none"> • DRI • EGL • ANV • RADV • other 	Protocol <ul style="list-style-type: none"> • DRI3.1 <ul style="list-style-type: none"> ◦ X.org ◦ xcb • Wayland <ul style="list-style-type: none"> ◦ Mutter ◦ Weston 	Khronos <ul style="list-style-type: none"> • image_dma_buf_import_modifiers • VK_EXT_external_*
Intel	✓	✓	✓	✓	✓
Collabora	✓	✓	✓	✓	✓
Google	✓		✓		✓
Others	✓		✓	✓	

Status

- Many years in the making: almost there!
- Early modifier support already released: Mesa 17.2, Kernel 4.14
- Full compression support soon:
 - Modesetting
 - Wayland/Mutter?
 - X.org/DRI3
 - Mesa DRI3 support
 - Mesa Wayland support

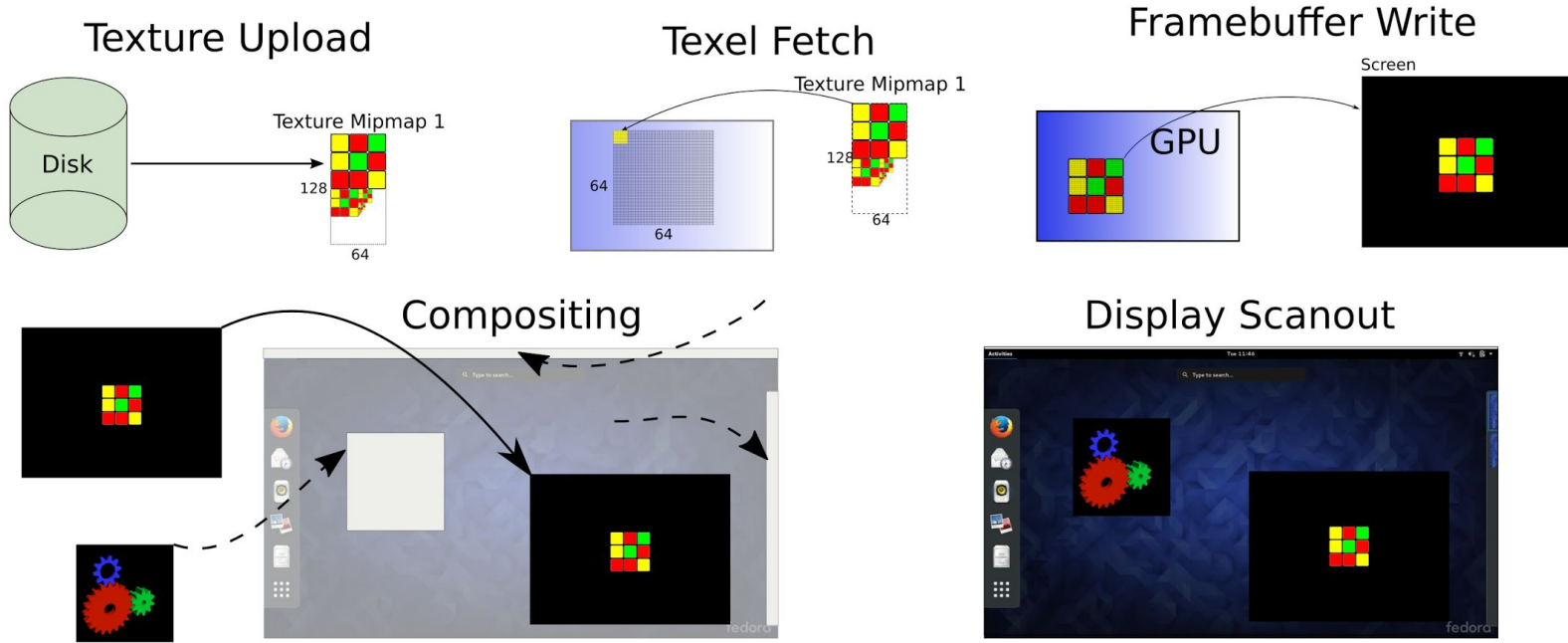
Mountains out of Molehills?

- Each EU needs 1GB/s bandwidth
 - Texturing (trilinear, anisotropic)
 - Transparency/Blending
 - Antialiasing
- Display
 - $3840 \text{ px} * 2160 \text{ rows} * 4 \text{ Bpp} * 60 \text{ Hz} = 1.85\text{GB/s}$
 - It keeps getting worse
 - Increasing resolutions (5K, 8K)
 - Increasing refresh rates (120Hz, 240 Hz)
- Workloads are already memory bandwidth limited
 - Can't scale up compute without more bandwidth
 - Reduce visual effects
 - Decrease resolution

Giant
Salt
Grain

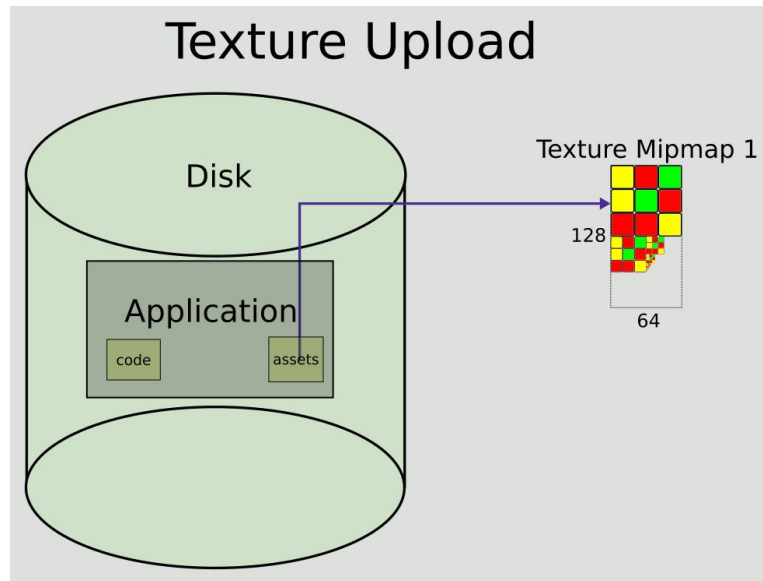


Admiring the Problem

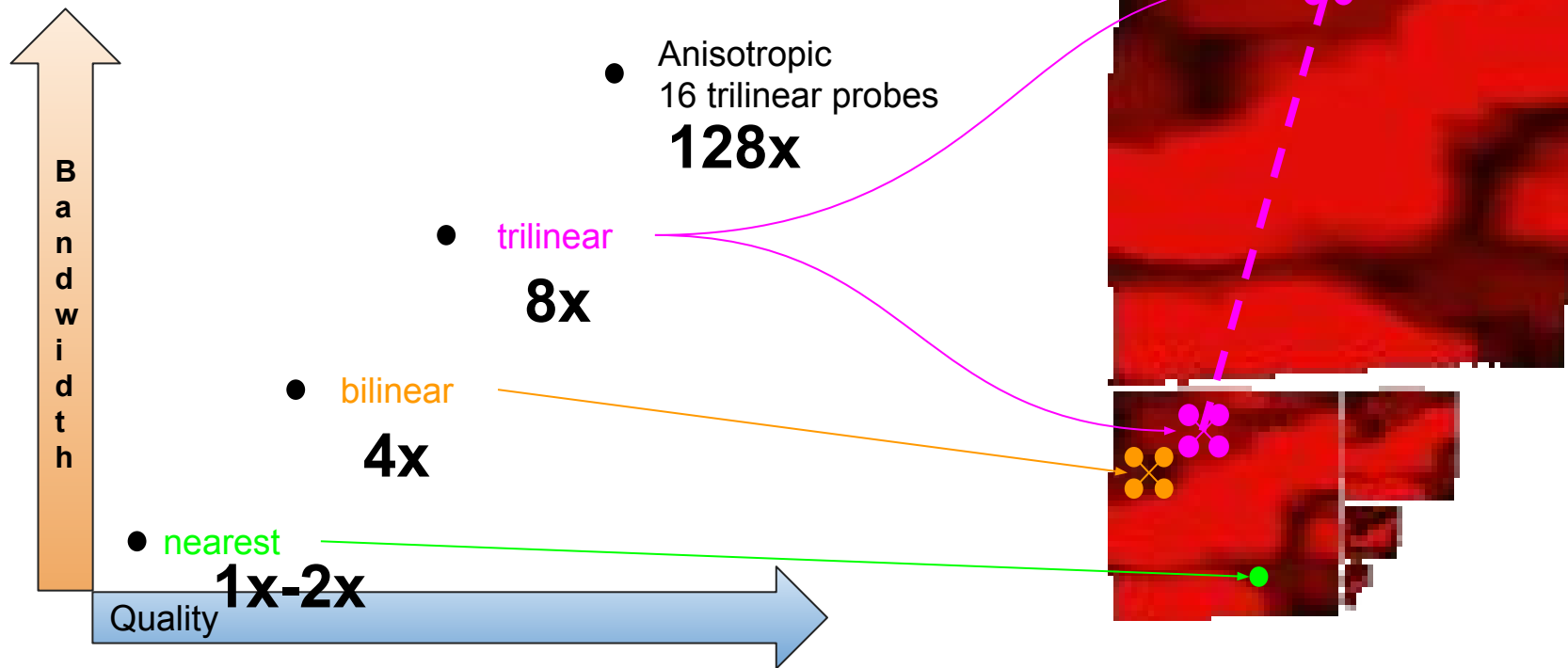


Texture Upload

The application needs to get its assets (geometry data, texture data, precompiled shaders, etc.) into memory from storage.



Texturing Fetch/Filtering



Sampling/Writing

```
#version 330
```

```
uniform sampler2D tex;
```

```
in vec2 texCoord;
```

```
out vec4 fragColor;
```

```
void main() {
```

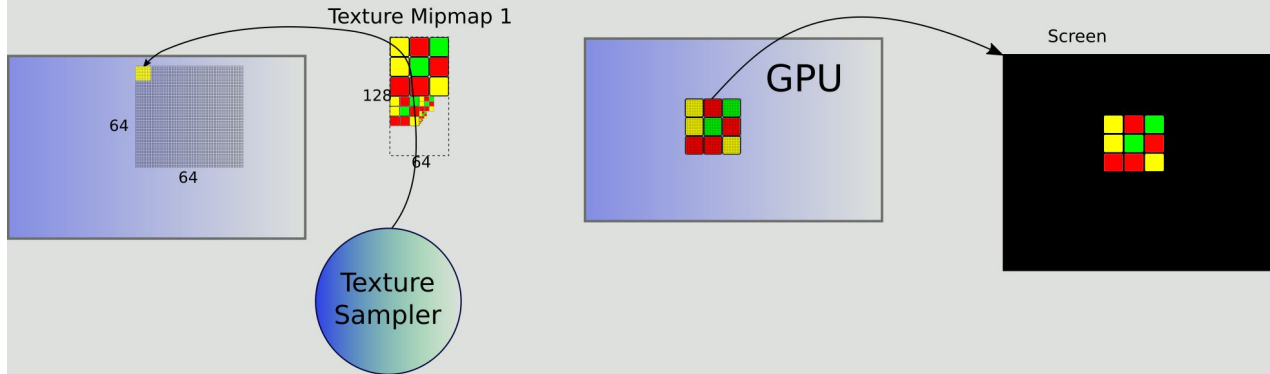
```
    vec4 temp = texelFetch(tex, ivec2(texCoord));
```

```
    fragColor = temp;
```

```
}
```

Texel Fetch

Framebuffer Write

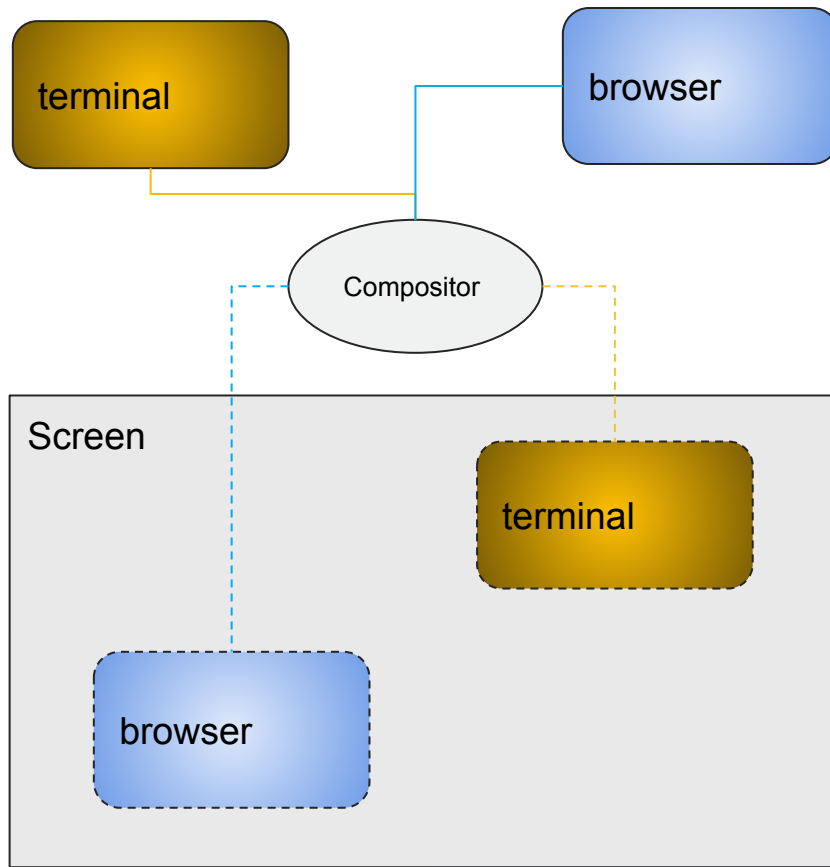


Compositing

Compositor is responsible for taking client application's window contents and amalgamates into a single image for display.

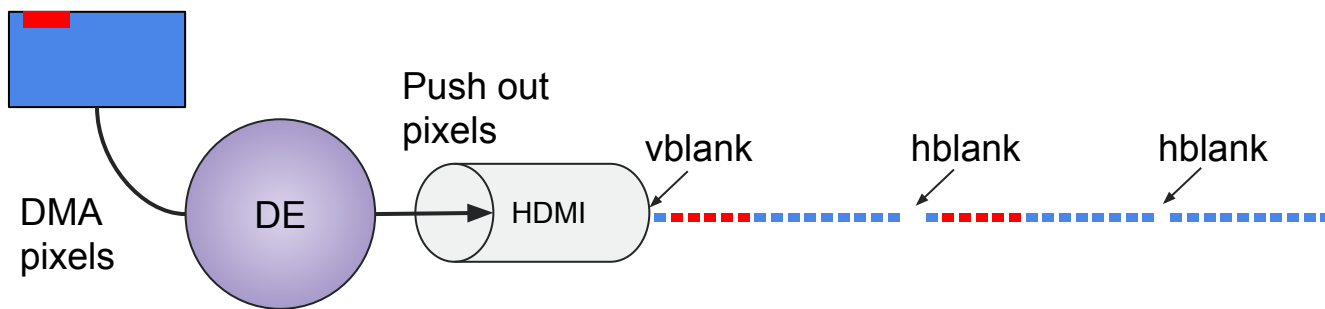
Like a window manager, but with offscreen buffers

- Needs to read from application's rendered data, and write to the screen



Display Engine

Specialized, fixed function hardware which sources pixel data and pushes it out over some display protocol; possibly blending, and scaling the pixels along the way.



Bandwidth Costs

Bytes Per Component

Operation	Color Depth	Desc.	Bandwidth	R/W
Texture Upload	1Bpc (RGBX8)	File to DRAM	16KB (64 * 64 * 4)	W
Texel Fetch	1Bpc (RGBX8)	DRAM to Sampler	16KB (64 * 64 * 4)	R
FB Write	1Bpc (RGBX8)	GPU to DRAM	16KB (64 * 64 * 4)	W
Compositing	1Bpc (RGBX8)	DRAM to DRAM	32KB (64 * 64 * 4 * 2)	R+W
Display Scanout	1Bpc (RGBX8)	DRAM to PHY	16KB (64 * 64 * 4)	R
<i>Total Bandwidth = (16 + 16 + 16 + 32 + 16) * 60Hz = 5.625 MB/s</i>				

At Least it Looks Better

Filter Mode	Multiplier (texel fetch stage)	Total Bandwidth
Nearest	1x	5.625 MB/s
Bilinear	4x	11.25 MB/s
Trilinear	8x	18.75 MB/s
Aniso 4x	32x	63.75 MB/s*
Aniso 16x	128x	243.75 MB/s*

* Oblique angle + implementation details would reduce further

Proposed Solution: Increase Headroom

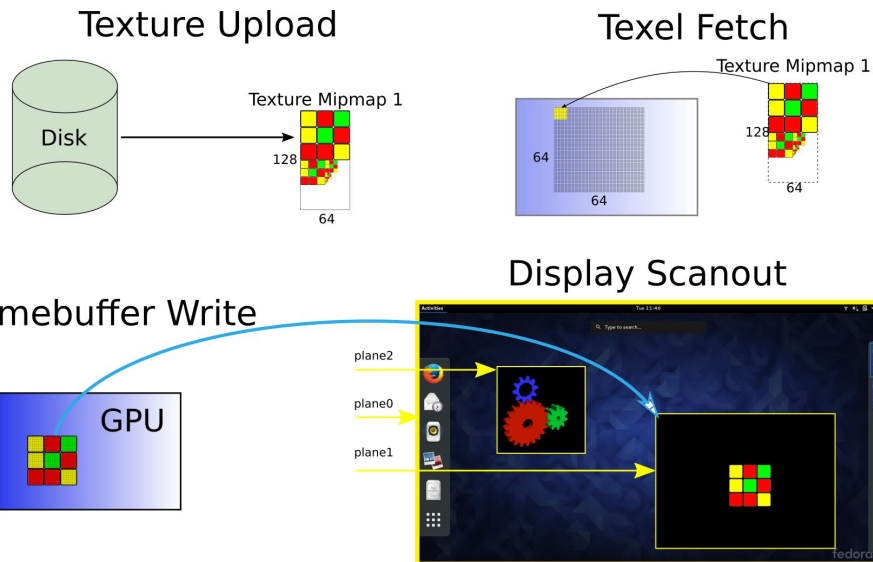
Color Depth	Operation	Bandwidth
1Bpc	Texture Upload	16KB (64 * 64 * 4)
1Bpc	Texel Fetch	16KB (64 * 64 * 4)
1Bpc	FB Write	16KB (64 * 64 * 4)
1Bpc	Composite	32KB (64 * 64 * 4 * 2)
1Bpc	Scanout	16KB (64 * 64 * 4)
<i>Total Bandwidth = 5.625 MB/s</i>		

Technology (~2013)	Technology (~2016)	Improvement
DDR3-2133 34GB/s (dual channel)	DDR4-3200 51.2 GB/s (dual channel)	50%
GTX 780 (Kepler) GDDR5 288 GB/s	GTX1080 (Pascal) GDDR5X 352 GB/s	22%
Radeon R9 290X (Hawaii) GDDR5 (320GB/s)	Radeon R9 Fury (Fiji) HBM1 512GB/s	60%
LPDDR3-1600 12.8 GB/s (single channel)	LPDDR4-3200 25.6 GB/s (single channel)	100%

Proposed Solution: Hardware Composition

Hardware is capable of having multiple hardware planes. Use them.

Color Depth	Operation	Bandwidth
1Bpc (RGBA8)	Texture Upload	16KB (64 * 64 * 4)
1Bpc (RGBA8)	Texel Fetch	16KB (64 * 64 * 4)
1Bpc (RGBA8)	FB Write	16KB (64 * 64 * 4)
4Bpc (RGBA8)	Composite	32KB (64 * 64 * 4 * 2)
1Bpc (RGBA8)	Scanout	16KB (64 * 64 * 4)
Total Bandwidth = 3.75 MB/s (33% savings)		



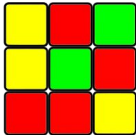
Proposed Solution: Texture Compression

- DXT1 (8:1)
- ETC1/2 (4:1)
- ASTC (variable, 6:1)

Color Depth	Operation	Bandwidth
DXT1	Texture Upload	16KB / 8
DXT1	Texel Fetch	16KB / 8
1Bpc	FB Write	16KB (64 * 64 * 4)
1Bpc	Composite	32KB (64 * 64 * 4 * 2)
1Bpc	Scanout	16KB (64 * 64 * 4)
Total Bandwidth = 3.925 MB/s (30%)		

nvcompress -nomips -nocuda -bcl ~/cube.png ~/cube.dds

DXT1
16 pixels
(4Bpp = 64B)




64 x 64 texture
4Bpp * 64pixels * 64rows
16K


64 x 64 texture
8BperBlock * 4blocks * 64rows
2K

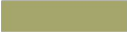
4b


2	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0

8b

0xF7C3 c₀ = (encoded 5:6:5) 

0x0822 c₁ = (encoded 5:6:5) 

0xA52D c₂ = $\frac{2}{3}c_0 + \frac{1}{3}c_1$ 

0x5296 c₃ = $\frac{1}{3}c_0 + \frac{2}{3}c_1$ 

Problems (Increase Bandwidth)

1. Limited by process and design
2. Costly for manufacturing
 - a. New memory modules
 - b. New boards
 - c. Utilizes new fabrication process
3. May be power hungry



Rating: Sure. Won't hold my breath

Problems (More Planes)

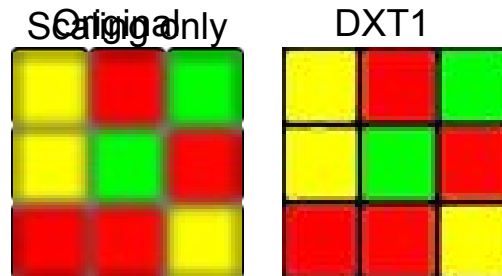


1. Hardware specific
 - a. Not all hardware can composite the same number of planes
2. Max planes is small
 - a. Increasing this significantly isn't feasible (die size)
3. Only helps compositing step

Rating: Great, doesn't scale

Problems (Texture Compression)

1. May be lossy
2. Hardware compatibility
 - a. Better formats require new hardware
 - b. Increased gate counts
3. Patents or proprietary
4. Misses display improvement
5. Doesn't play nicely with all filtering methods (aniso)



Rating: Great, but lacking

Introducing E2E Lossless Compression

- Supplement other bandwidth saving techniques
 - Doesn't reduce size (in fact internally things get larger).
- Internal hardware blocks compress/decompress on the fly.
 - Display
 - Media
 - Texture units

Pros

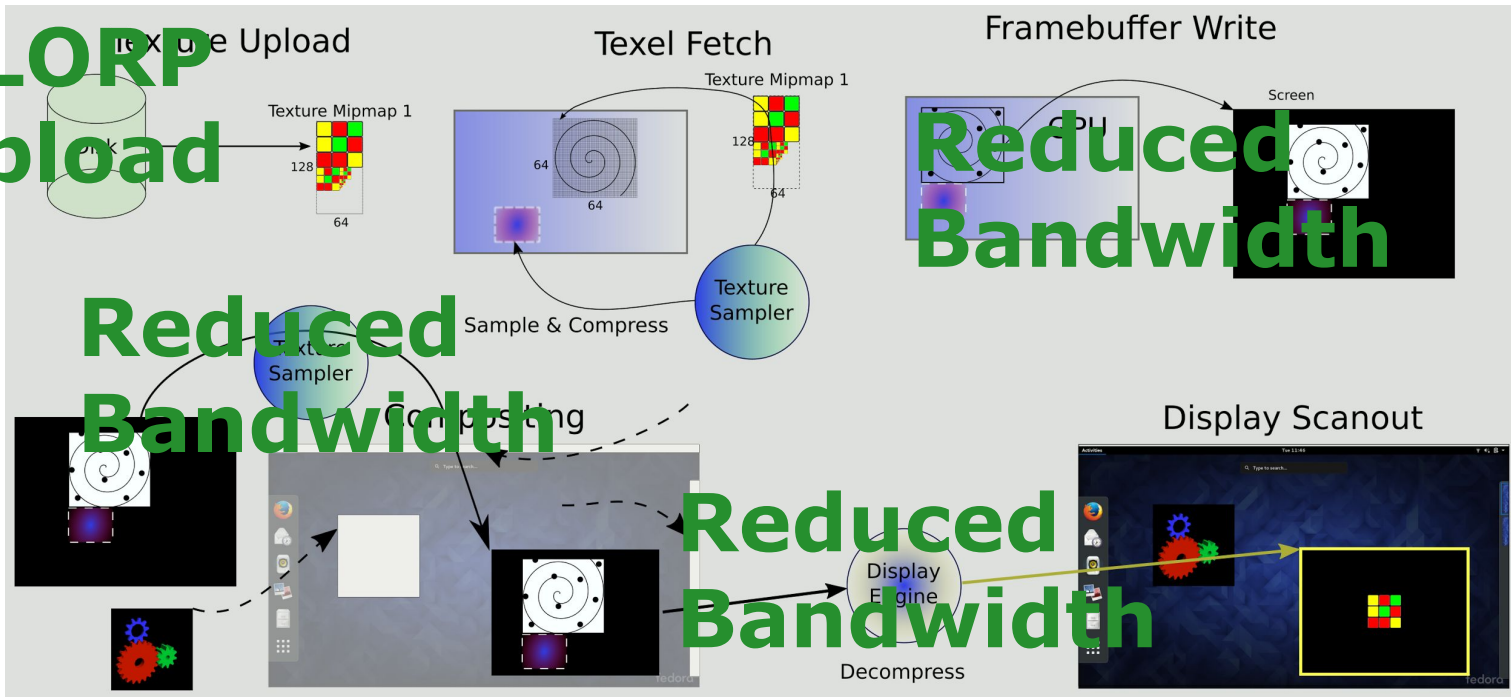
- Lossless
- Transparent to applications/tools
 - Easier development
 - Hardware improvements automatically help
- No offline compression necessary
- Compression benefits through display
- Can be huge when texturing is small amount of bandwidth consumption

Cons

- Relatively low compression
 - 2:1 max on current Intel
 - 4:1 max seems to be industry standard
 - **Not everything will be compressible**
 - Will never get max.
- *Limited by hardware*
 - **However**, many GPUs getting this
 - UBWC (Qualcomm), AFBC (ARM), DCC (AMD), DCC (Nvidia), PVRIC (Imagination)

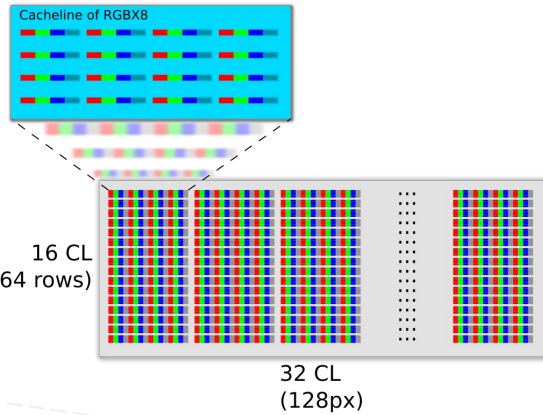
Several Opportunities for Savings

BLOPP
Upload

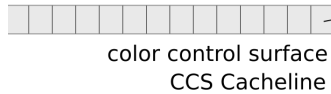


Dumb Example

Naive implementation will get 2:1 compression when a pair of cachelines has 12 or less colors

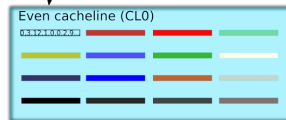


Semi-realistic example
2b per CL pair
00 = Clear Color
01 = Use LUT w/ CL0
10 = Invalid
11 = Uncompressed



CCS cacheline = 32K main frame

DW0	Indices for 0-7 (4b)
DW1	Indices for 8-15 (4b)
DW2	Indices for 16-23 (4b)
DW3	Indices for 24-31 (4b)
DW4-15	Color 0-11



E2E Bandwidth Savings (2:1 compression)

Operation	Color Depth	Desc.	Bandwidth	R/W
Texture Upload	1Bpc (RGBX8)	File to DRAM	16KB (64 * 64 * 4)	W
Texel Fetch	1Bpc (RGBX8)	DRAM to Sampler	16KB (64 * 64 * 4)	R
FB Write	Compressed (2:1)	GPU to DRAM	16KB (64 * 64 * 4) /2	W
Compositing	Compressed (2:1)	DRAM to DRAM	32KB (64 * 64 * 4 * 2) /2	R+W
Display Scanout	Compressed (2:1)	DRAM to PHY	16KB (64 * 64 * 4) /2	R
Total Bandwidth = (16 + 16 + 8 + 16 + 8) * 60Hz = 3.75 MB/s (33%)				

Molehills out of Mountains!

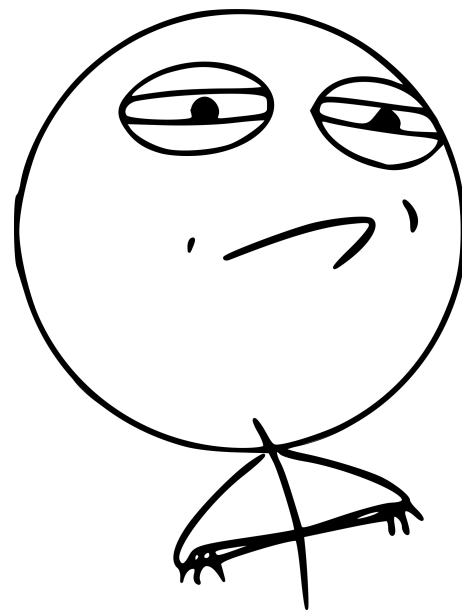
Technique	Bandwidth	BW Savings	Disk Savings
Base	5.625 MB/s $16 + 16 + 16 + 32 + 16$	-	-
+ HW compositing	3.75 MB/s $16 + 16 + 16 + 32 + 16$	33%	0%
+ DXT1 Compression	2.11 MB/s $2 + 2 + 16 + 32 + 16$	62%	30%
+ E2E Compression	1.17 MB/s $2 + 2 + 8 + 32 + 8$	79%	30%

Intermission



Implementation Challenges

1. Currently, everything treats a framebuffer as a buffer of pixels.
 - a. The main buffer is no longer just pixel data.
 - b. There's another buffer! (similar to planar formats)
2. Buffer allocation, buffer import/export, and display server protocol need to be made aware of this.
3. Applications and compositors cannot rely on compression working everywhere.
 - a. Ex. Skylake doesn't allow compression on pipe C



Several Solutions

1. Encode “modifiers” in fourcc format
 - a. V4L does this (include/uapi/linux/videodev2.h)
 - b. Works well for entirely proprietary formats
 - c. Concern about amount of bits for modifiers in DRM
 - i. Graphics formats combinatorially explode faster [apparently]
 - ii. Even 64b modifier was questioned
 - d. Never really considered (not sure why)
2. Intel specific plane property ([original proposal](#))
 - a. Many other drivers shared similar problem.
 - b. KMS clients wanted a hardware agnostic mechanism
 - c. Protocol still required anyway
3. dma-buf metadata
 - a. Just a get/set IOCTL for adding modifiers to a dma-buf

The Result - Modifiers

- *Some* support already landed
- Describes **modifications** to a buffer's layout
- Easy to add new modifiers to support different tiling formats
- Missing some key pieces
 - Query interface
 - Protocol
 - Driver implementation
- Compression somewhat muddies the definition

```
commit e3eb3250d84ef97b766312345774367b6a310db8
Author: Rob Clark <robdclark@gmail.com>
Date: Thu Feb 5 14:41:52 2015 +0000
```

```
drm: add support for tiled/compressed/etc modifier in addfb2
```

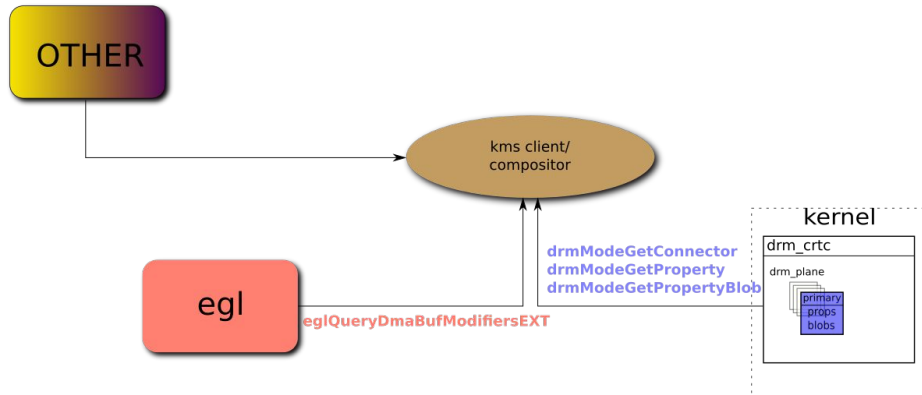
```
struct drm_mode_fb_cmd2 {
    __u32 fb_id;
    __u32 width;
    __u32 height;
    __u32 pixel_format; /* fourcc code from drm_fourcc.h */
    __u32 flags; /* see above flags */
};

/*
 * In case of planar formats, this ioctl allows up to 4
 * buffer objects with offsets and pitches per plane.
 * The pitch and offset order is dictated by the fourcc,
 * e.g. NV12 (http://fourcc.org/yuv.php#NV12) is described as:
 *
 * YUV 4:2:0 image with a plane of 8 bit Y samples
 * followed by an interleaved U/V plane containing
 * 8 bit 2x2 subsampled colour difference samples.
 *
 * So it would consist of Y as offsets[0] and UV as
 * offsets[1]. Note that offsets[0] will generally
 * be 0 (but this is not required).
 *
 * To accommodate tiled, compressed, etc formats, a
 * modifier can be specified. The default value of zero
 * indicates "native" format as specified by the fourcc.
 * Vendor specific modifier token. Note that even though
 * it looks like we have a modifier per-plane, we in fact
 * do not. The modifier for each plane must be identical.
 * Thus all combinations of different data layouts for
 * multi plane formats must be enumerated as separate
 * modifiers.
 */
__u32 handles[4];
__u32 pitches[4]; /* pitch for each plane */
__u32 offsets[4]; /* offset of each plane */
__u64 modifier[4]; /* ie, tiling, compress */
};
```

```
<drm/drm_mode.h [Git(drm-intel-next-queued)][cpp] (449,1) 58%
```

Step 1: Compositor Negotiation

Query all “sink” APIs to find out what modifiers are supported for the given format, and hardware.



Queries

- Blobifier (KMS blob property for `drm_plane`)
 - What modifiers does the plane support?
- EGL extensions
 - `EXT_image_dma_buf_import_modifiers`
 - `eglQueryDmaBufModifiersEXT`
 - What modifiers does my format support?
 - “is used to query the `dma_buf` format modifiers supported by `<dp>` for the given format.”
- Vulkan/WSI (WIP)
 - `VK_MESAX_external_image_dma_buf`.

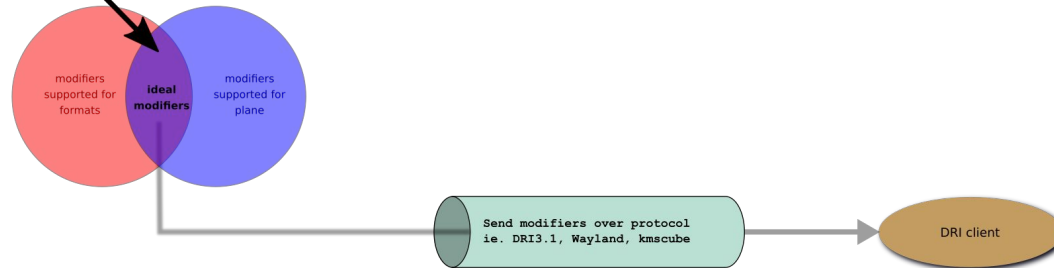
Plumbers: Collabora (funded by Intel and Google), Google, Intel

Step 2: Take That and Shove It

down your protocol pipe

With the optimal modifiers in hand, some protocol will tell the client which modifiers it might want to use.

Modifiers which can be used for direct scanout, and used to sample from EGLImages



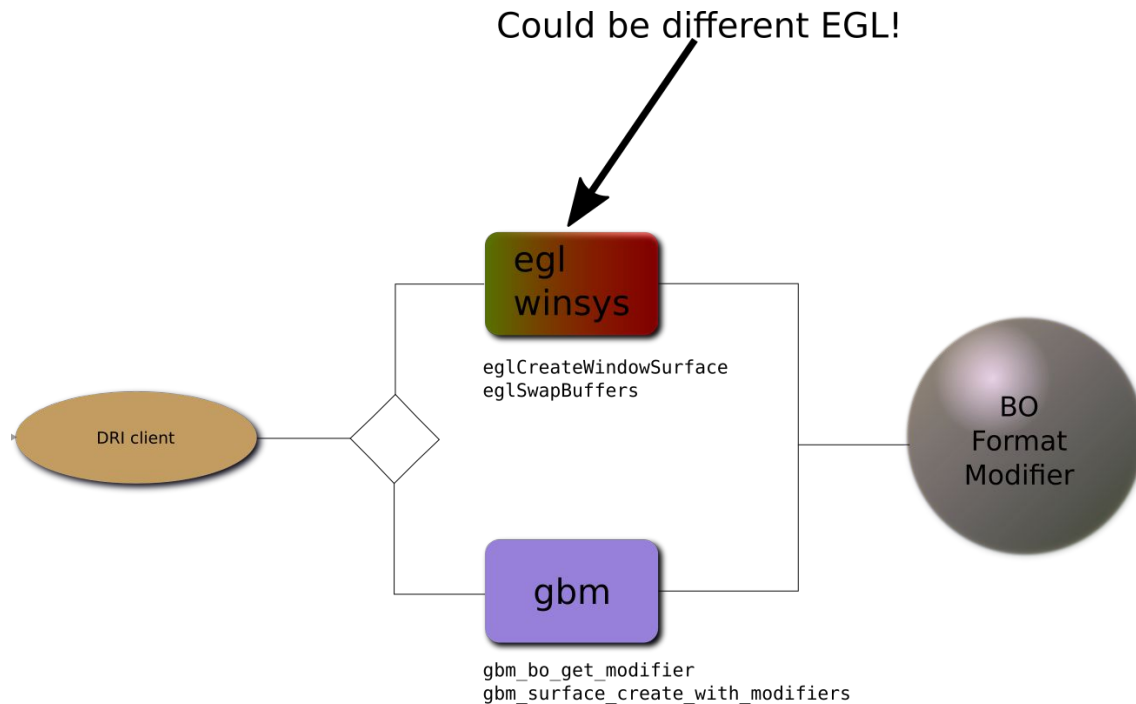
Protocols

- Wayland
 - “zwp_linux_buffer_params_v1” version="3”
- DRI3.1
 - Multi-plane support
 - xDRI3GetSupportedModifiers

Plumbers: Collabora (paid for by Intel)

Step 3: Making BOs

Next, the client creates the buffer either directly, or indirectly with the formats and modifiers it desires.

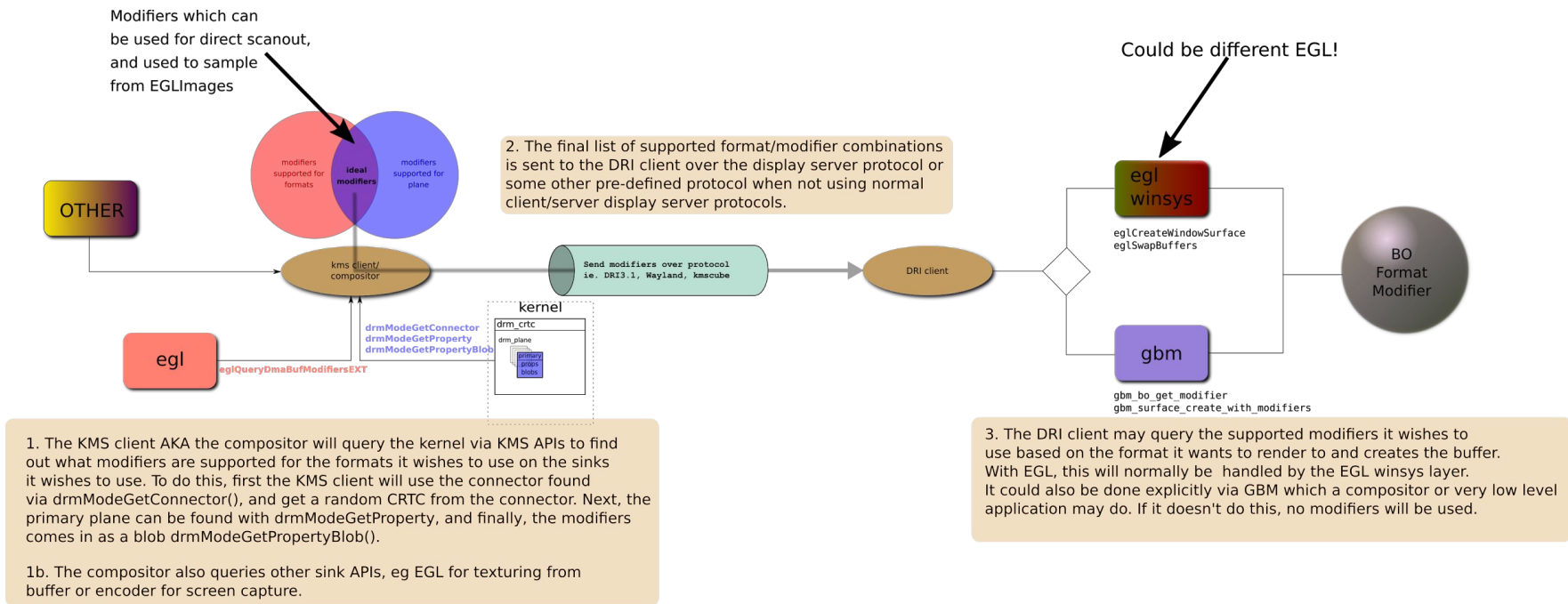


Buffer Creation

- EGL
 - `eglCreateWindowSurface`
 - Wayland
 - X11
 - (Mesa) Ask over DRI3.1 what's supported
 - (Mesa) Call into DRI driver to create an image
- GBM
 - `gbm_*_create_with_modifiers`
- DRImage
 - `createImageWithModifiers` (made for GBM)
 - `createImageFromDmaBufs2` (made for DRI 3.1)
- Vulkan/WSI (WIP)

Plumbers: Collabora, Google, Intel

The Whole Story Thus Far

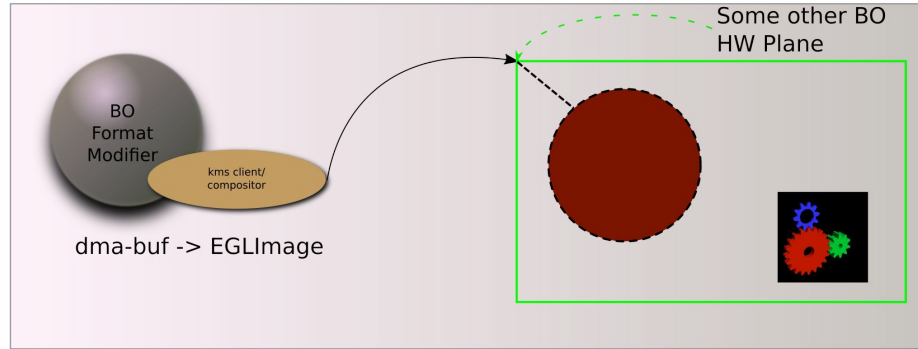


Display it

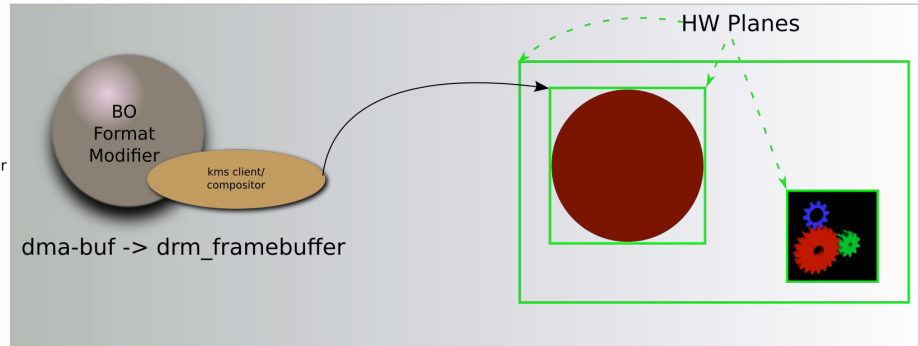
1. Software Compositing (Option)
 - a. EGL_EXT_image_dma_buf_import_modifiers
 - b. Much work required
2. Hardware compositing (Option)
 - a. drmModeAddFB2WithModifiers
 - b. Relatively minor changes required. AddFB2 already supported modifiers
 - i. Add new modifiers to drm_fourcc.h
 - ii. Added error checking when modifiers change plane count.
 - iii. Driver specific handling of modifiers.

He'll Flip You (for real)

Option 1:
Create EGLImage
then composite via
texturing



Option 2:
Create drm_framebuffer
from client's BO then
flip directly



Preliminary Results

“Benchmark”	Original	CCS	%improved
kmscube	1.22 GB/s	600 MB/s	51 (2x)
glxgears	1775 FPS	3900 FPS	54 (2.2x)
TRex			2.3 (.02x)

Takeaways

- Memory bandwidth requirements for graphics workloads can be astronomical.
- Don't assume texture compression is the end of the bandwidth story.
- Modifiers “modify” the framebuffer's pixel layout.
- Lossless compression reduces bandwidth, not size
 - Many GPUs support this transparently
- Hardware compositing is great.
- Getting features like this plumbed through can easily be a multi-year effort.
- Haiku isn't supported :/



Thank Yous

Platinum Level

Kristian Høgsberg, Google

Daniel Stone, Collabora

Gold Level

Rob Clark, Red Hat

Jason Ekstrand, Intel

Ville Syrjälä, Intel

Daniel Vetter, Intel

Liviu Dudau, Arm Ltd

Eric Engestrom, Imagination Technologies

Varad Gautam, Collabora

Topi Pohjolainen, Intel

Lucas Stach, Pengutronix

Emil Velikov, Collabora

Chad Versace, Google

Tomeu Vizoso, Collabora

Q&A (not about EGLStreams)