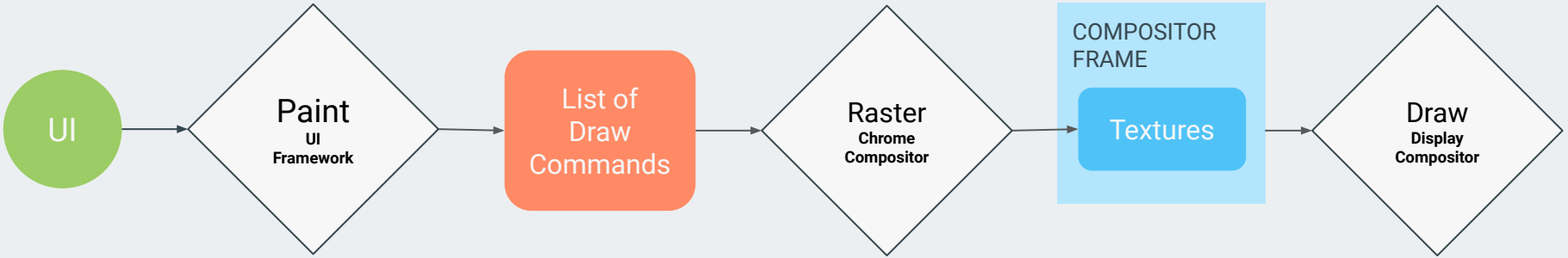
Chromium OS - Compositor Pipeline

malaykeshav@chromium.org | April 2019

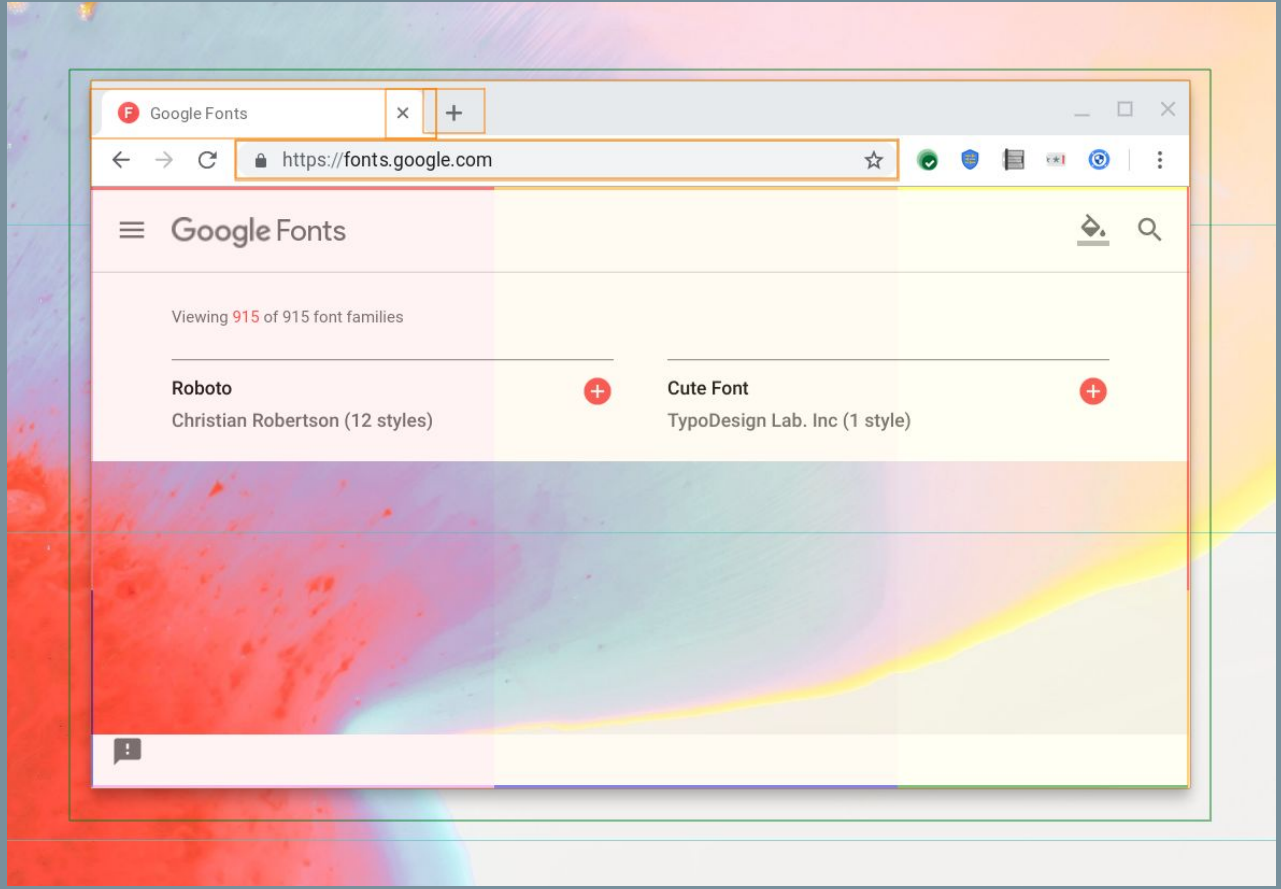
go/cros-compositor-pipeline

Contents


UI
Paint
UI
Framework
List of
Draw
Commands
Raster
Chrome
Compositor
COMPOSITOR
FRAME
Textures
Draw
Display
Compositor

Layers

Base entity which holds information on what needs to be displayed in the region marked by its bounds. A layer does not own its child layer however certain properties may affect the child layer.

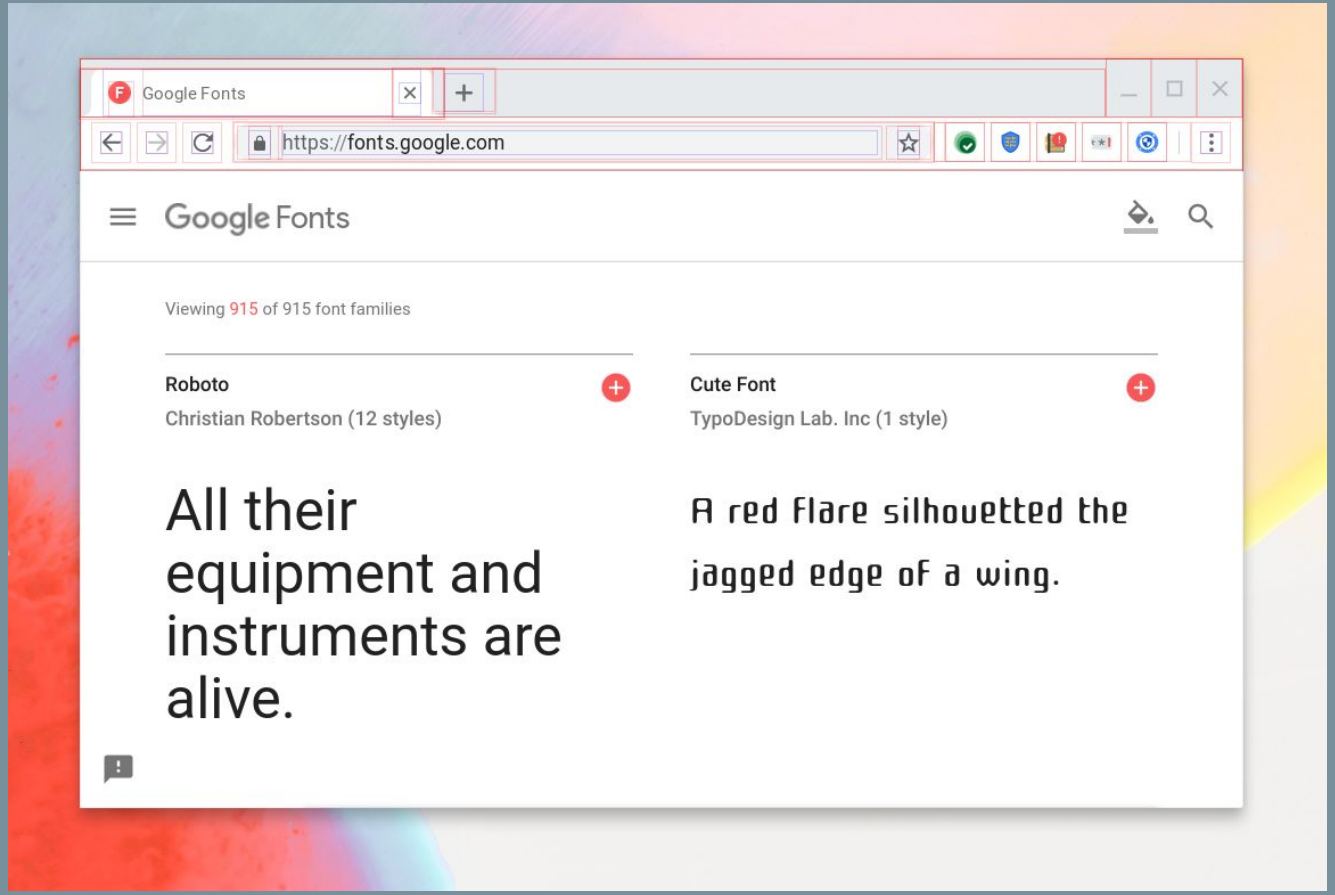


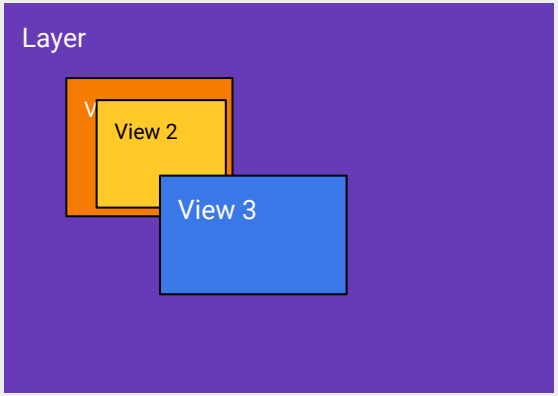
UI Layer Types

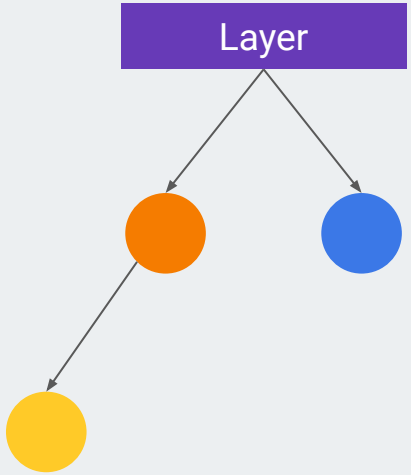
- Not Drawn
- Solid color layer
- Painted texture layer
- Transferable resource layer
- Surface layer
- Nine patch layer

Views

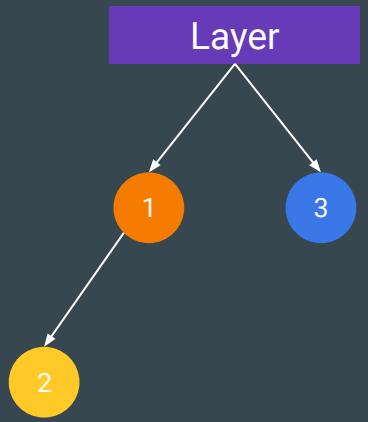
A toolkit to paint onto the ui layer in a hierarchical structure. Each view can only paint within its own bound which is a sub region of the ui layer it belongs to.



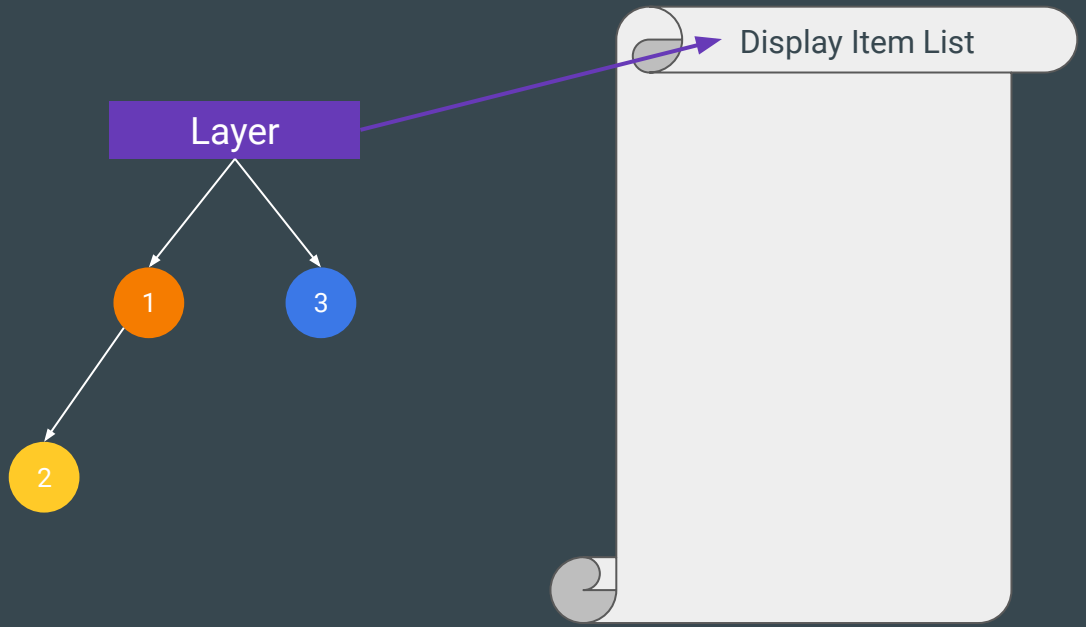

Layer
View 2
View 3


Layer

What do I mean when I say paint

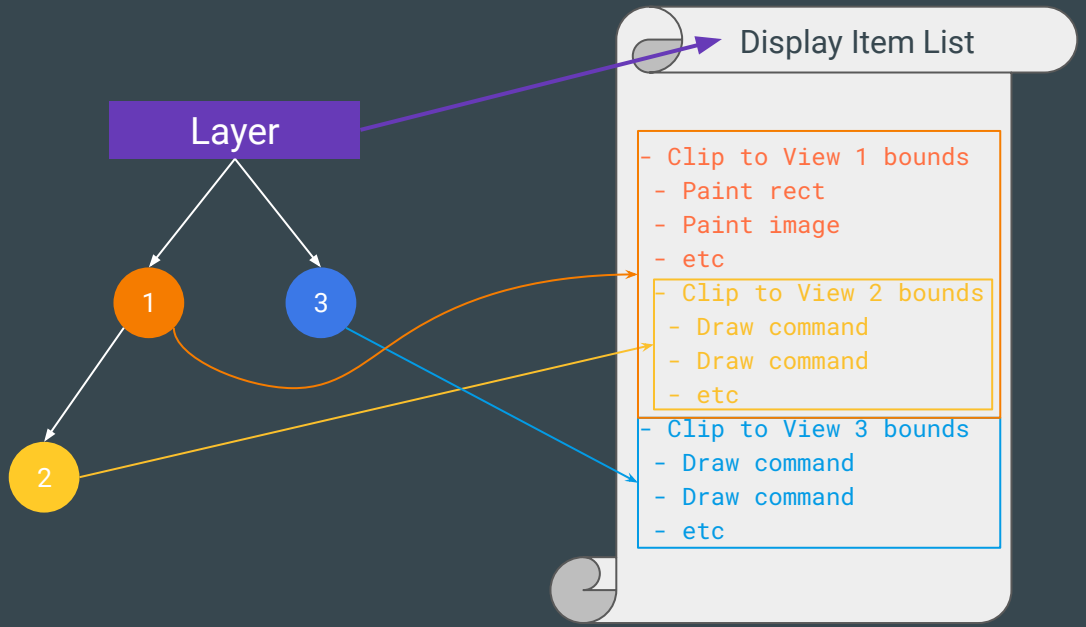


What do I mean when I say paint



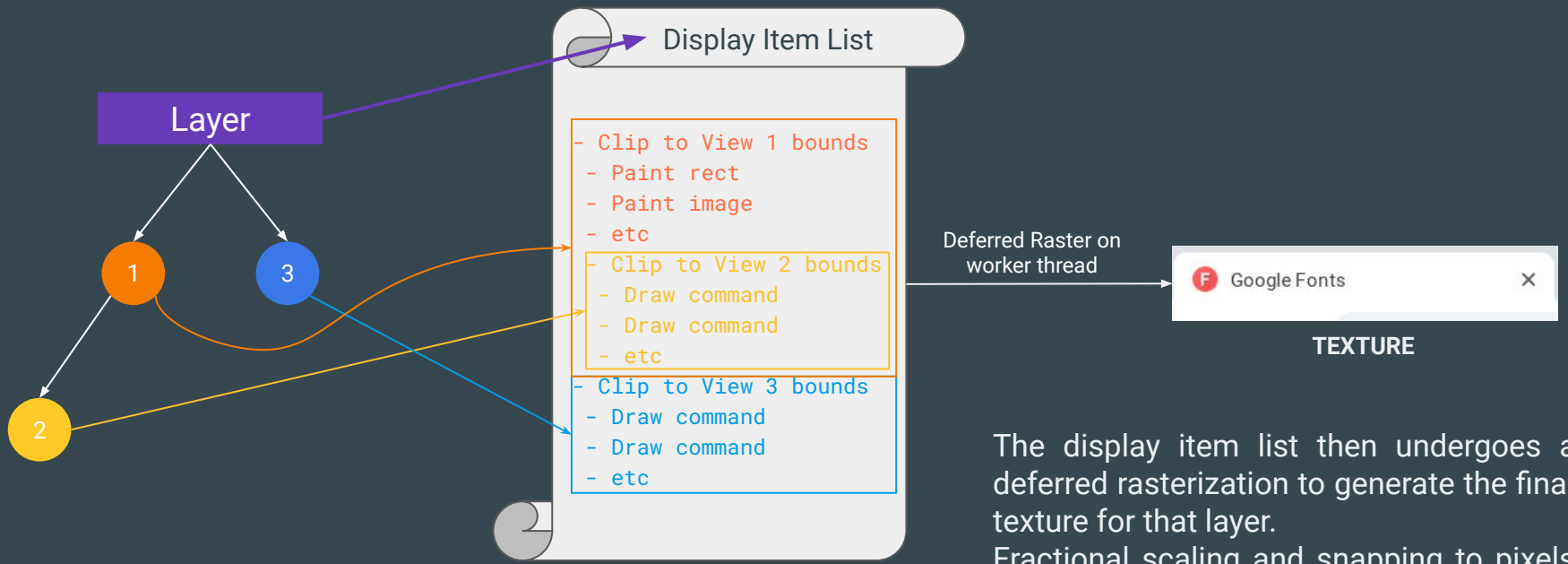
A DisplayItemList is associated with each layer at the time of paint. This list is passed to the layer's delegate to paint on.

What do I mean when I say paint



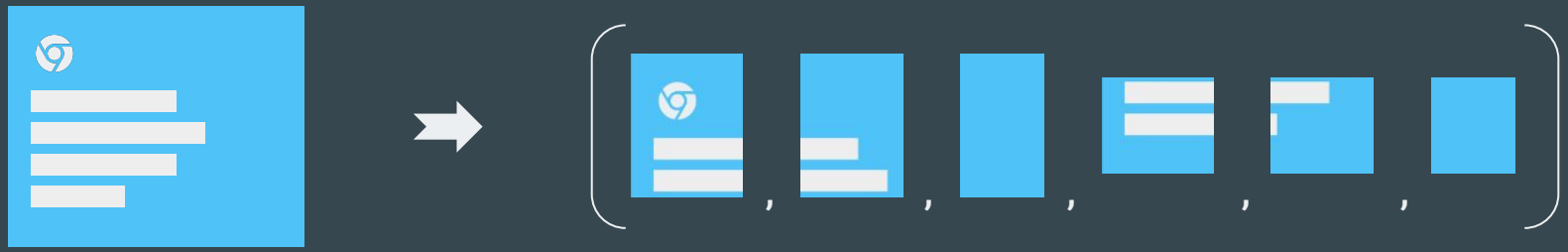
When a view paints, it appends its list of draw commands onto the layer's display item list.

What do I mean when I say paint



The display item list then undergoes a deferred rasterization to generate the final texture for that layer.
Fractional scaling and snapping to pixels is handled by the views framework.

Textures generated this way are tiles

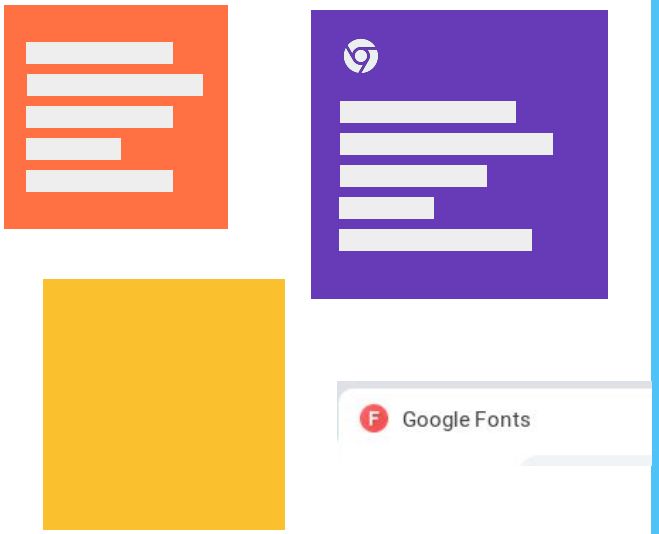


Why the need for multiple layers?

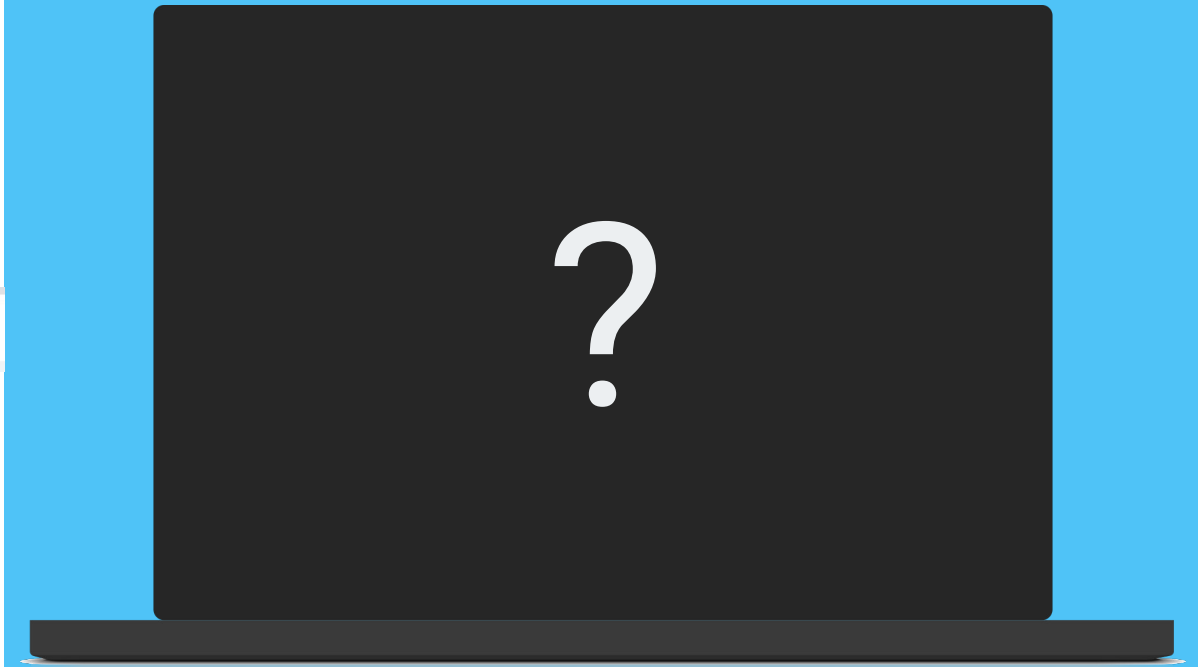
- Layers are expensive.
- They hold textures that take up memory.
- Have a larger overhead than views.

But

- In terms of performance, layers are faster to manipulate than views in terms of transform, effects and clipping.
- For a view to update and perform the same operation, it needs to repaint and regenerate the texture (repaint and re raster).

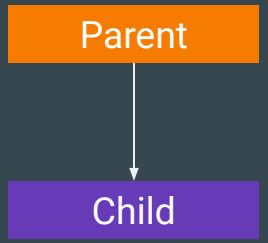


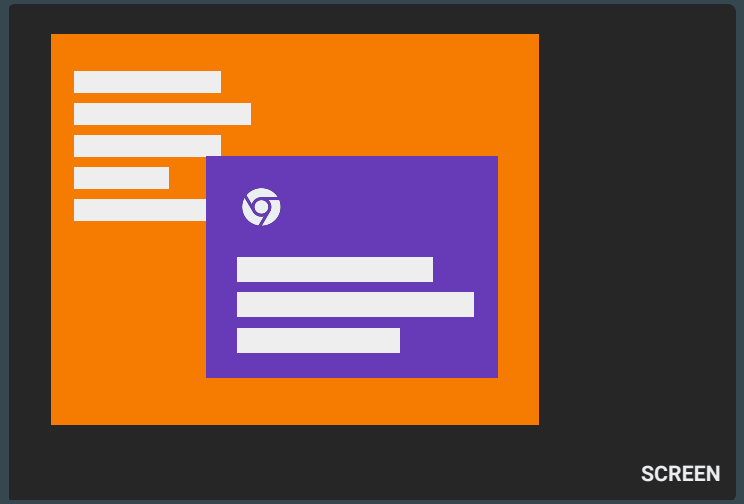
We now have textures from each of the layers that draw. The next step is to draw them onto the display buffer and apply any effects along the way. But how do we compute their final position, clip or effects like opacity and blur? Sometimes, these effects interact and depend on textures and properties from other layers.



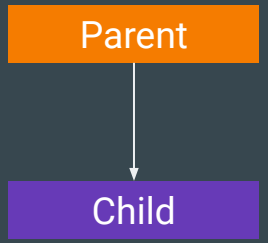
Different Properties of a layer

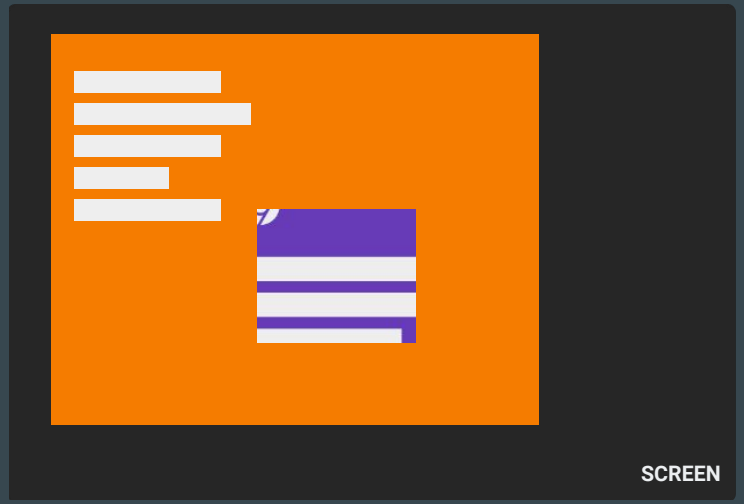
Clip





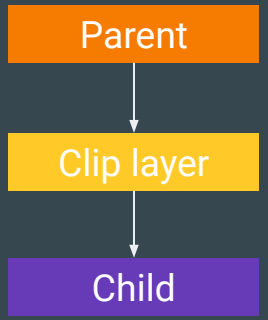
Clip

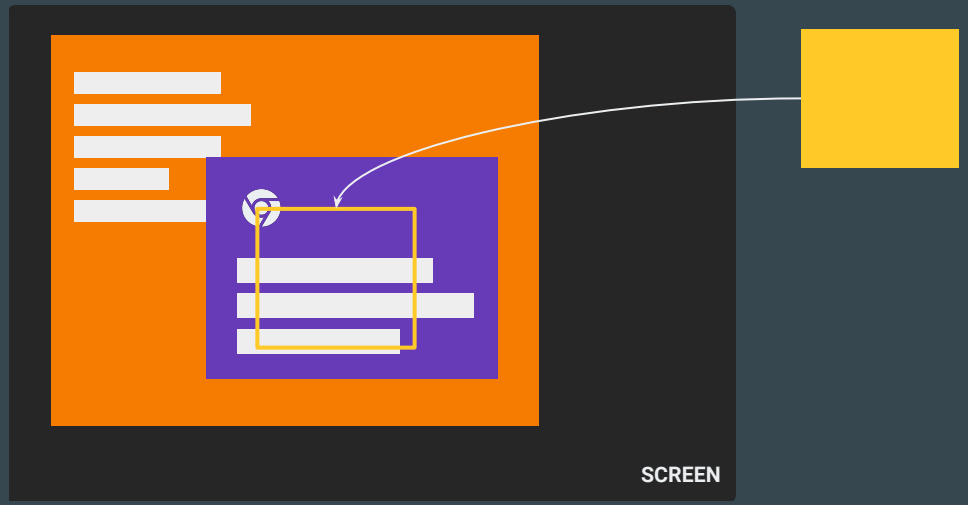




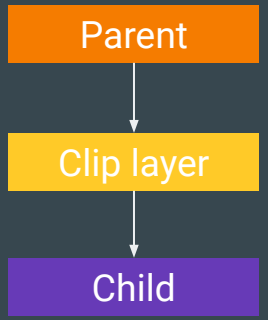
Clip

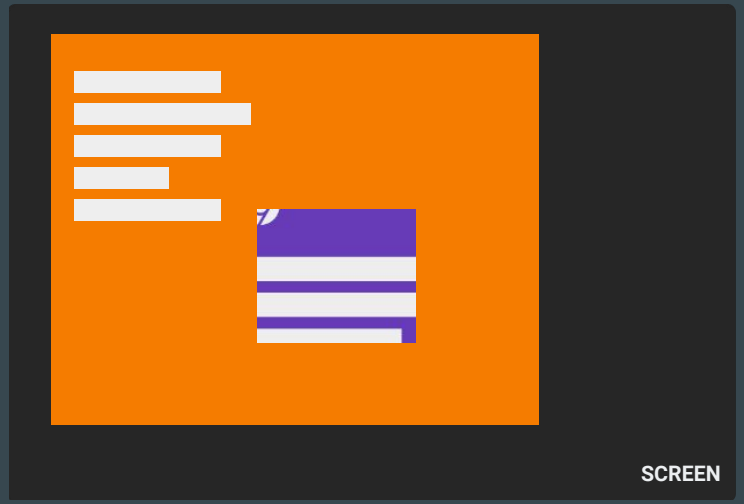
To achieve this, we add an intermediate layer with the bounds of the clip and its clip to bounds property set. This layer will now clip its subtree to its own bounds. This layer also does not have to draw any content.



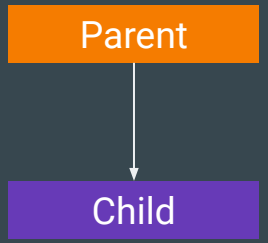


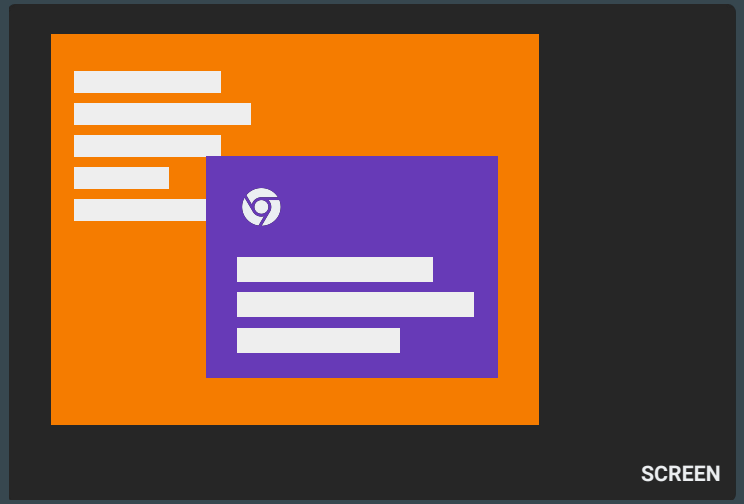
Clip



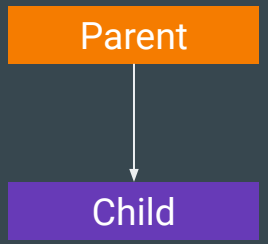


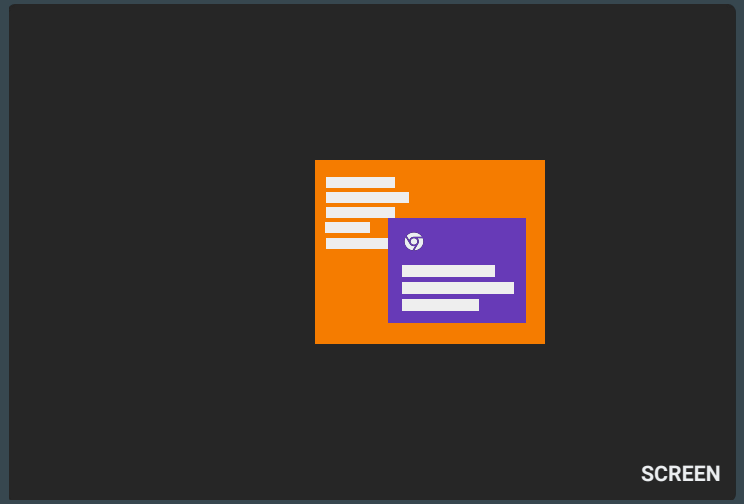
Transform



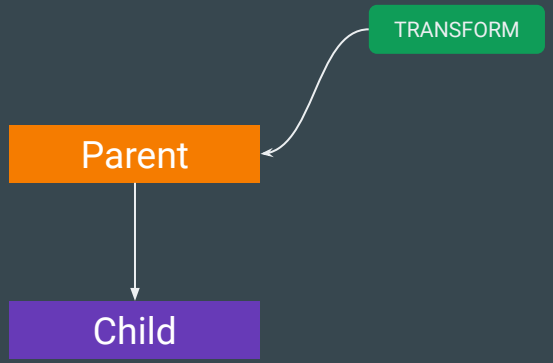


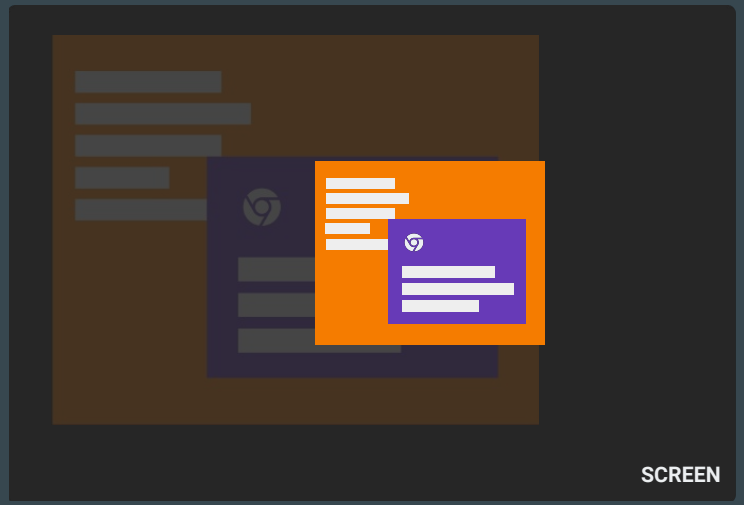
Transform





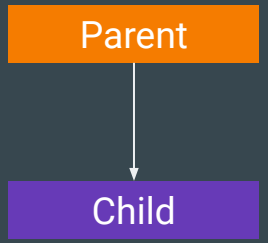
Transform

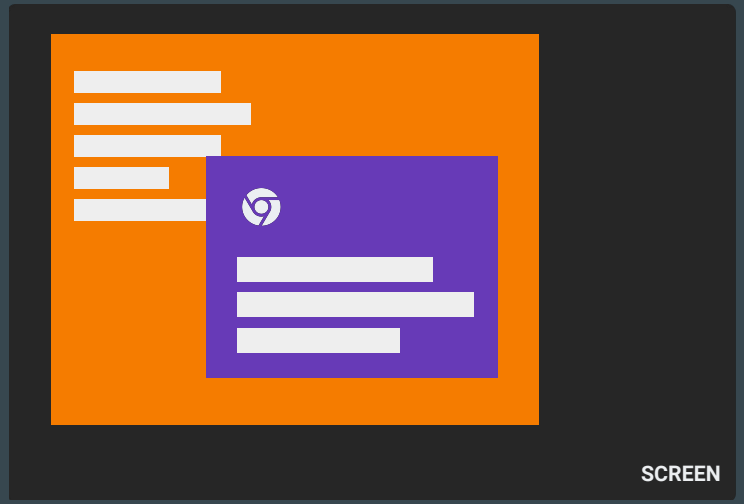




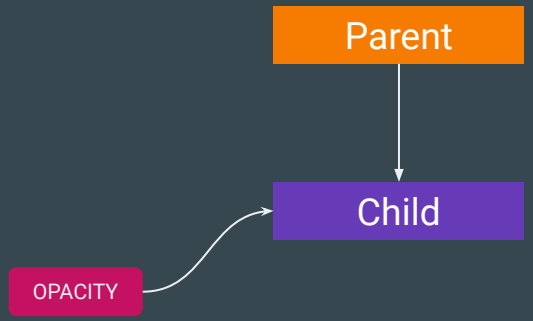
Apply the transform (translate and scale) on the parent layer and the child layer is also affected by it.

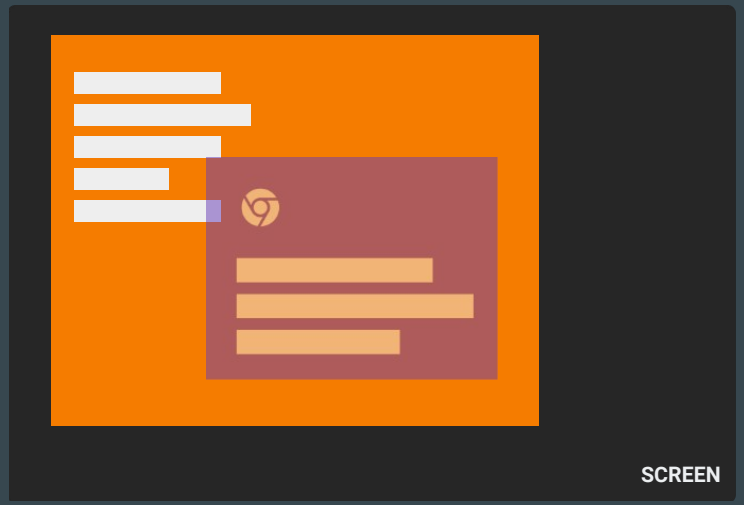
Effects



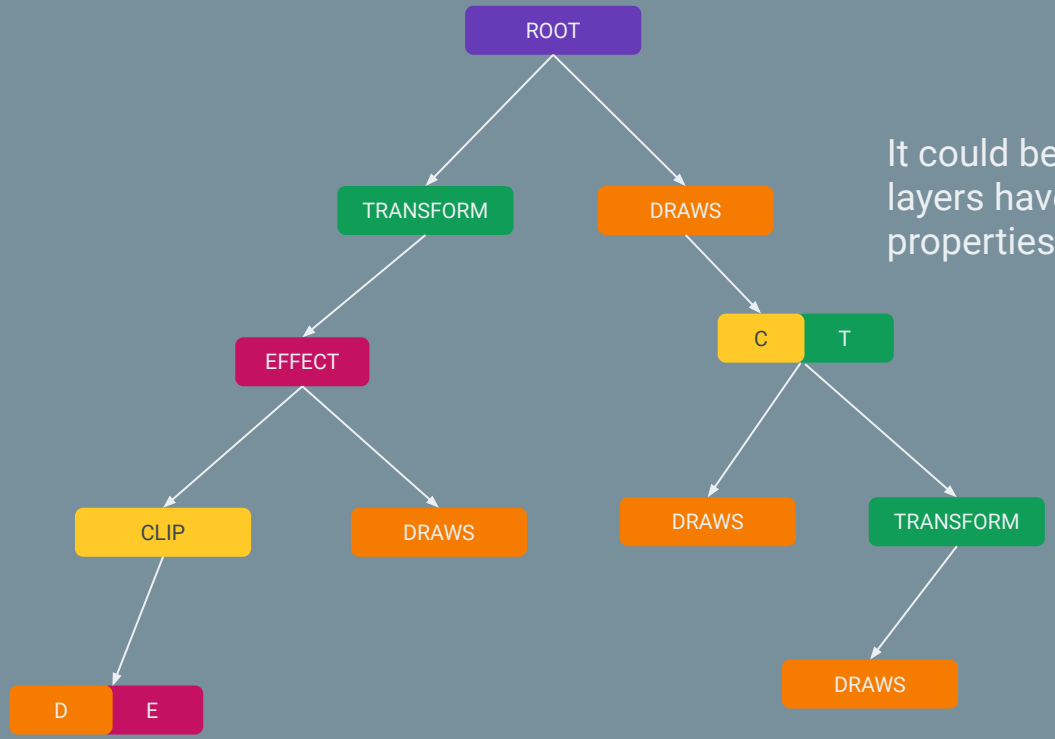


Effects



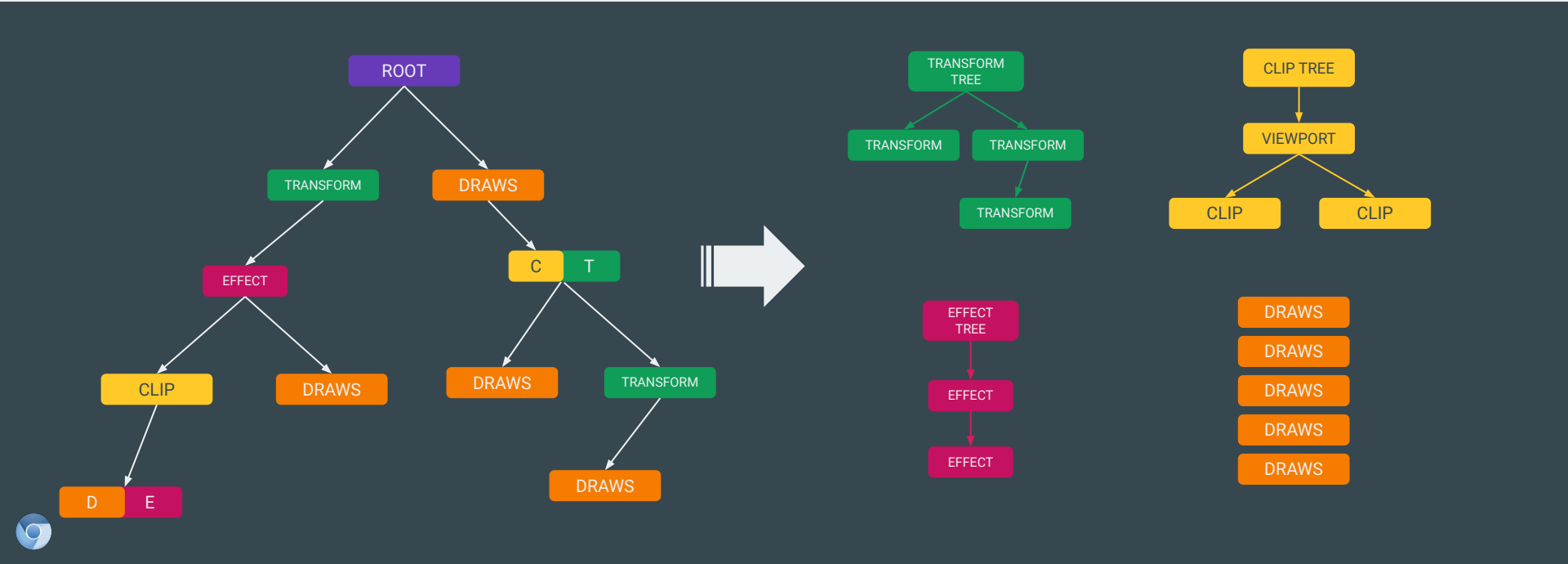


Layer Tree



It could be some layers have multiple properties at once

Property Trees + Draw Layer List



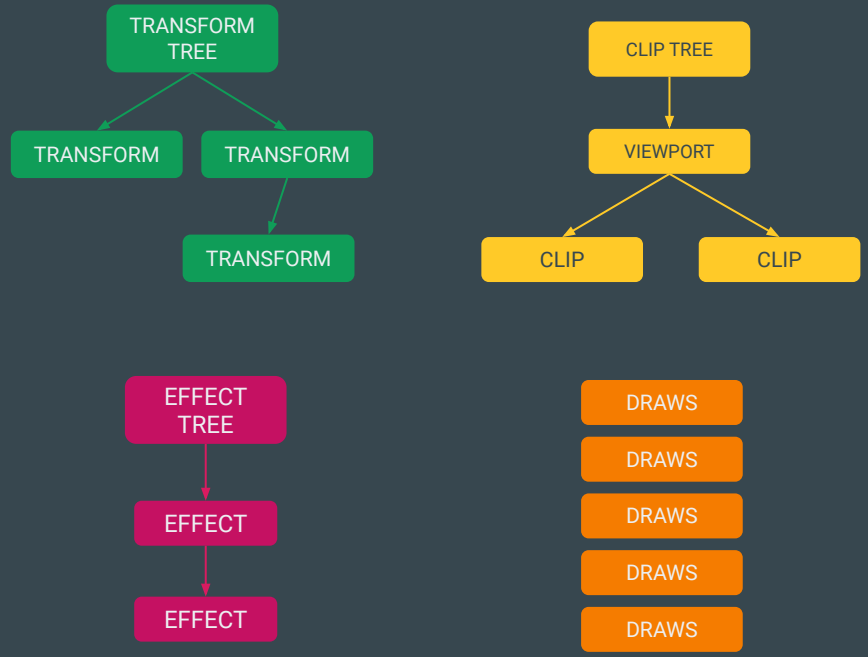
Property Trees

Trees are sparse -- not every layer has an interesting transform, clip, effect, or scroll.

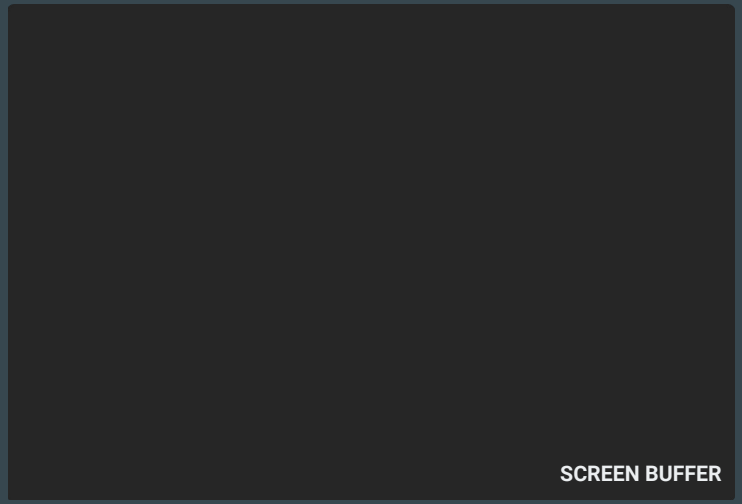
But cross-tree dependencies do exist

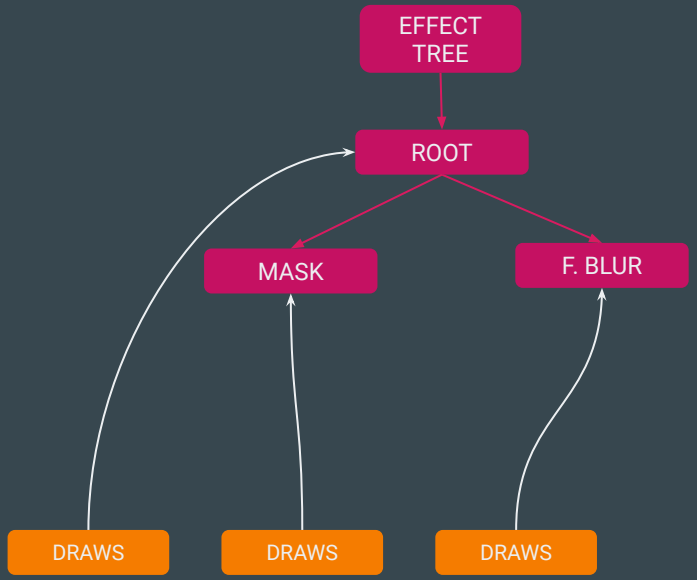
- e.g. clips and effects happen in a particular transform space

Layer list still has non-drawing layers (for now). For this talk lets only considers the layer that draw.

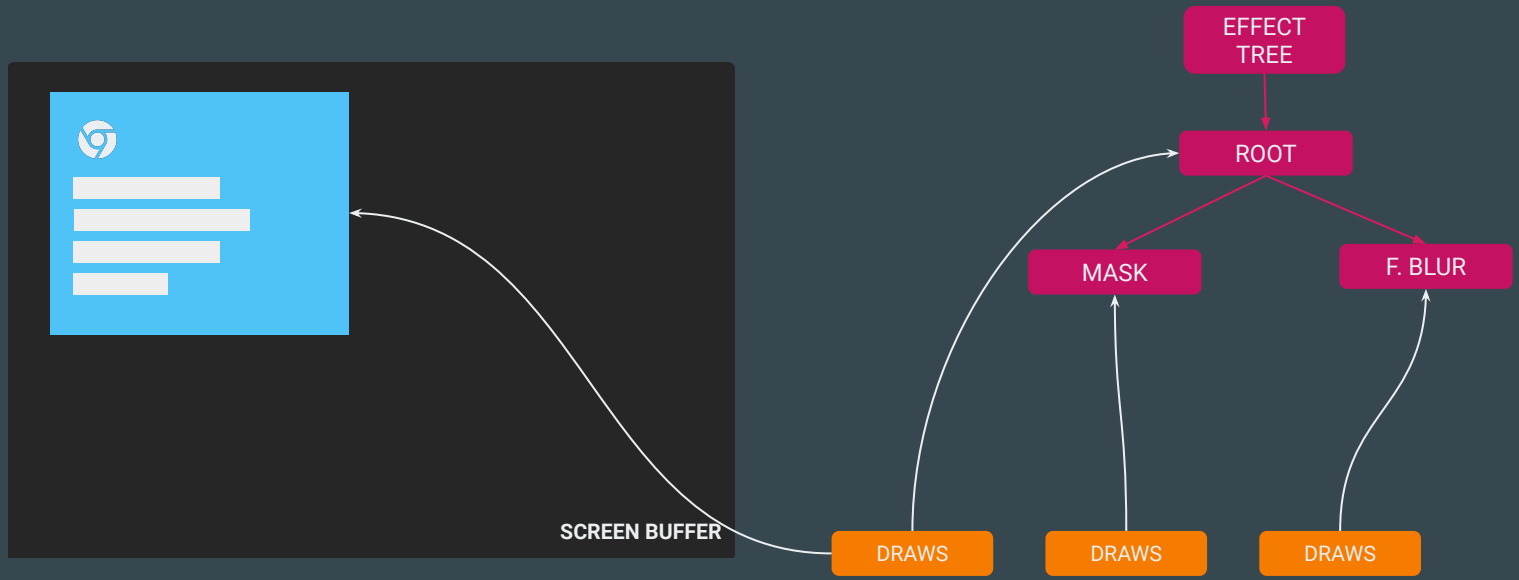


Effect Tree

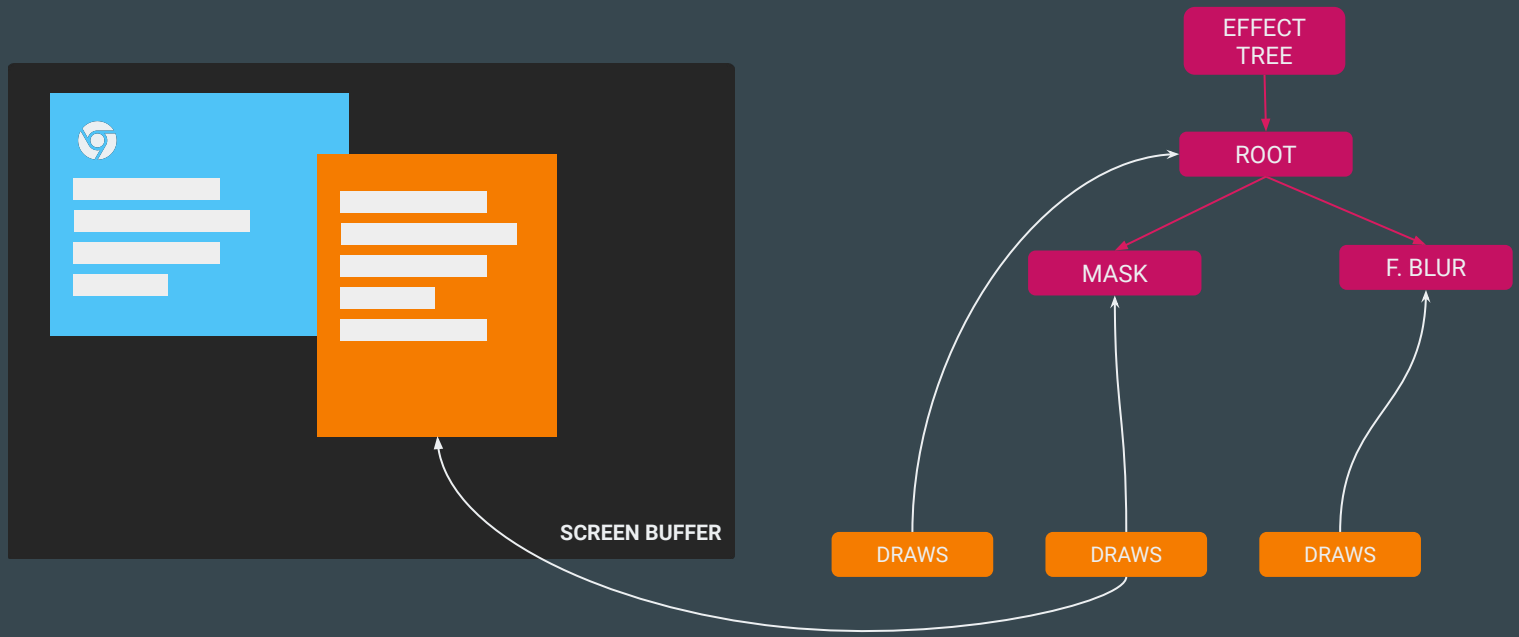

SCREEN BUFFER


EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS

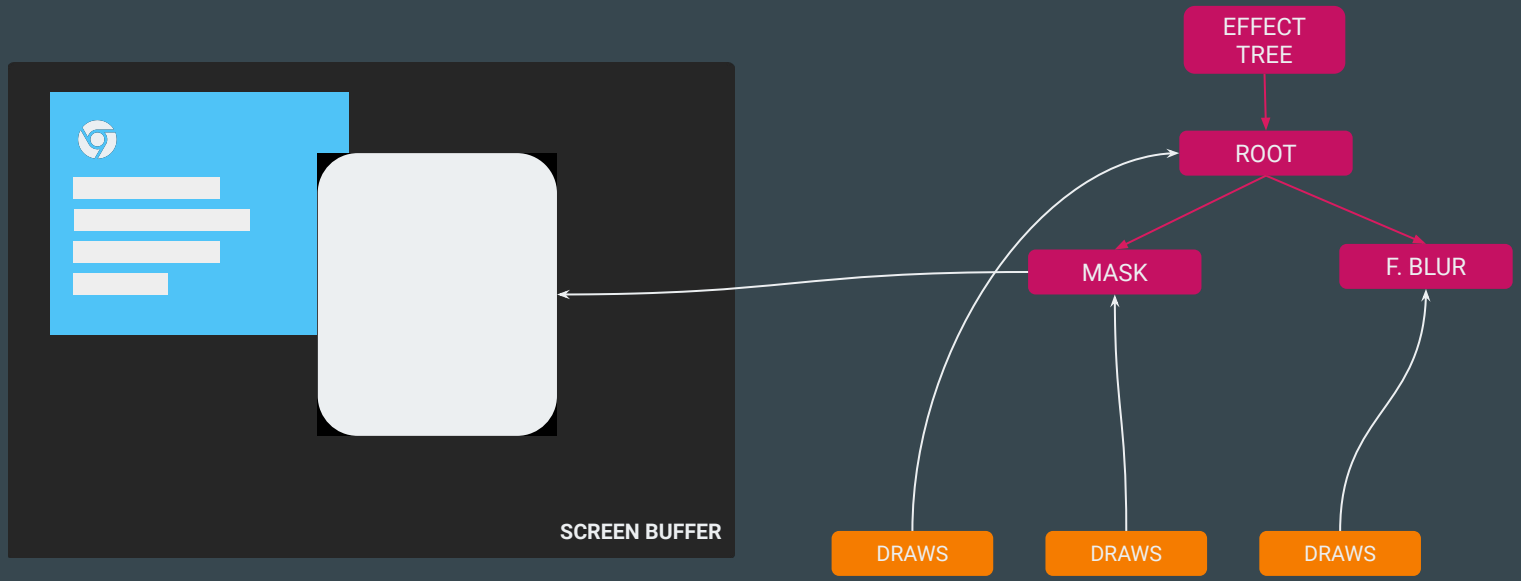
Effect Tree


EFFECT TREE
ROOT
MASK
F. BLUR
SCREEN BUFFER
DRAWS
DRAWS
DRAWS

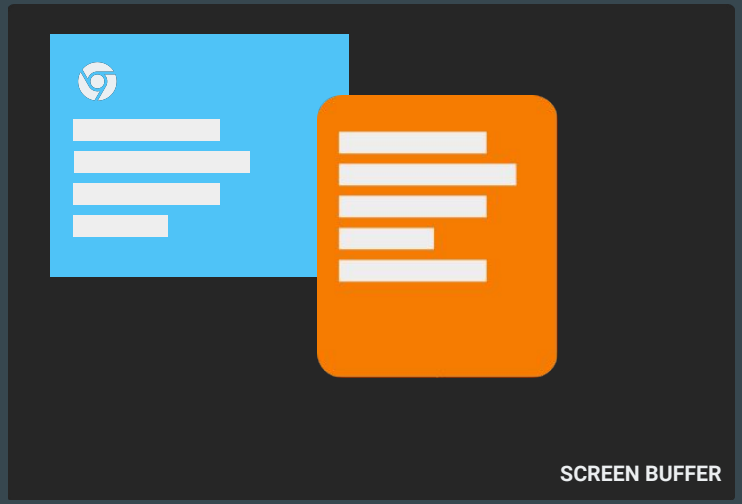
Effect Tree

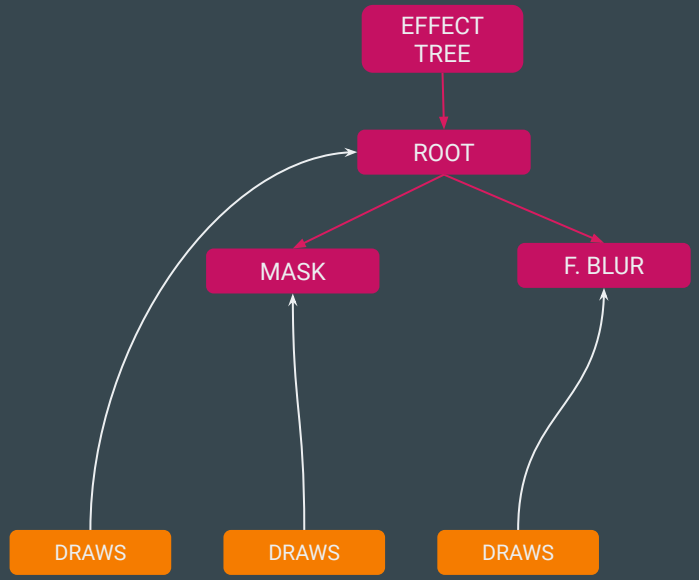

SCREEN BUFFER
EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS

Effect Tree

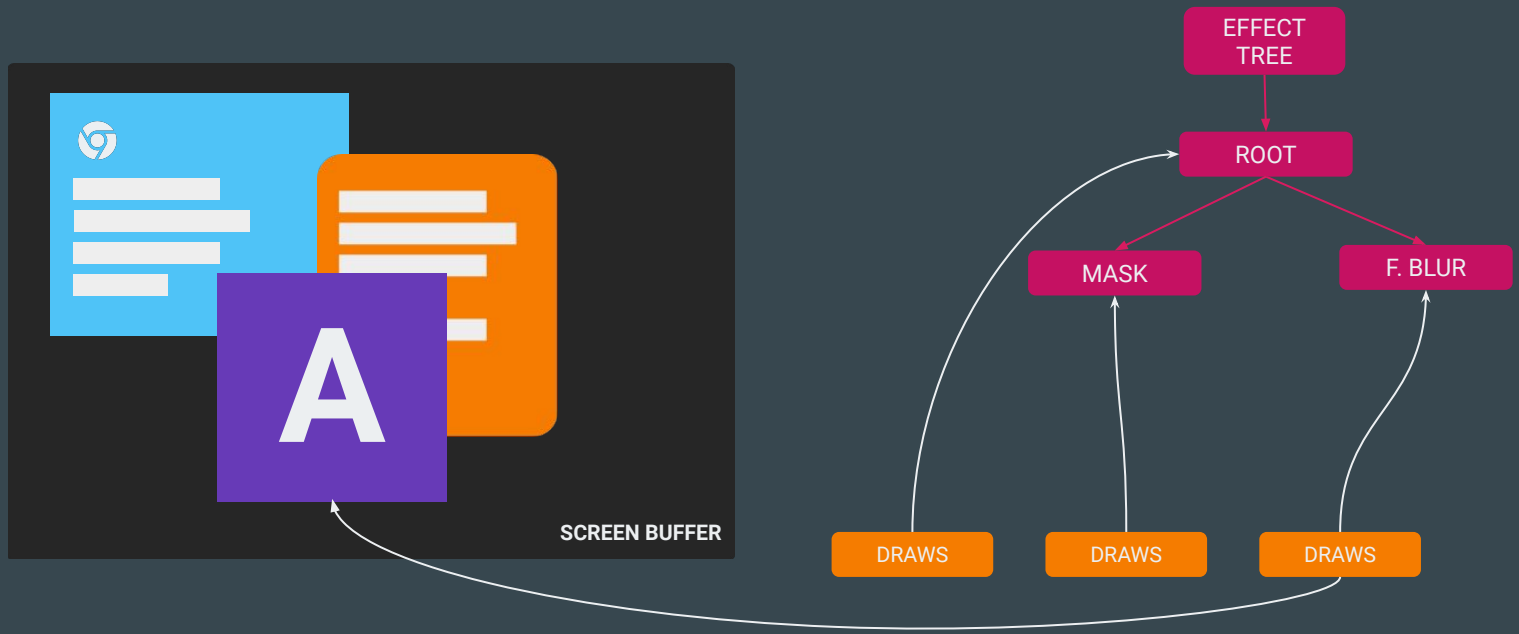

EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS
SCREEN BUFFER

Effect Tree

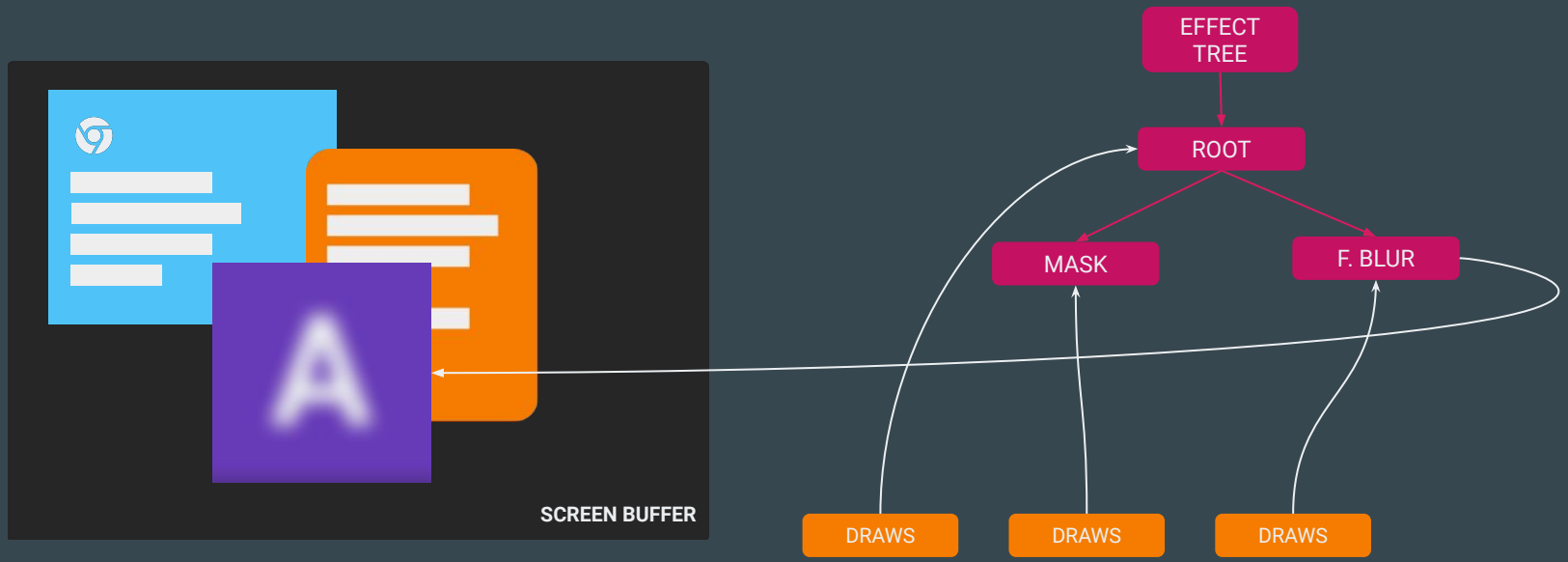

SCREEN BUFFER


EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS

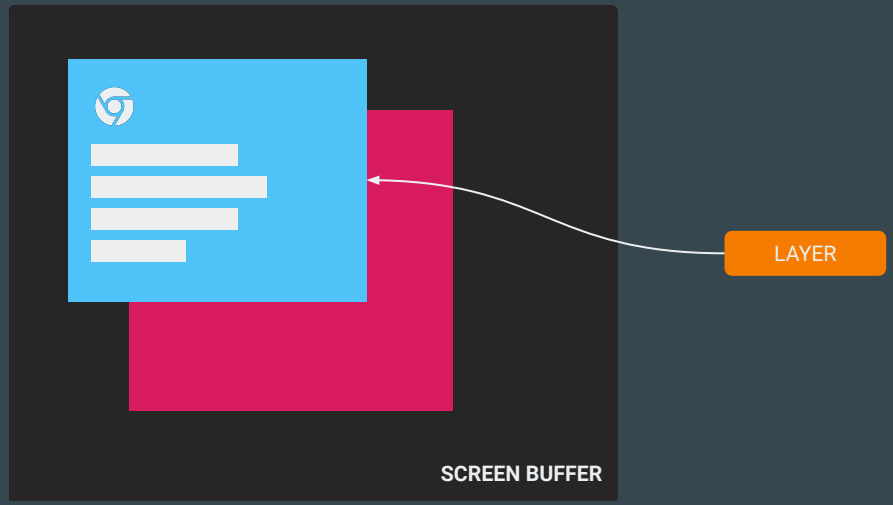
Effect Tree


A
SCREEN BUFFER
EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS

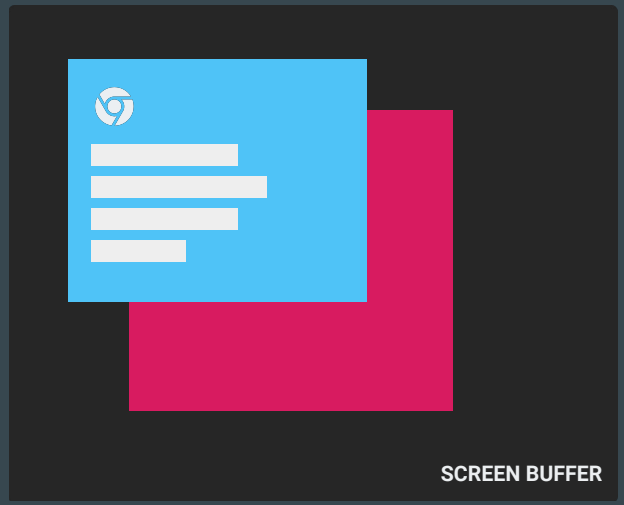
Effect Tree


EFFECT TREE
ROOT
MASK
F. BLUR
DRAWS
DRAWS
DRAWS
SCREEN BUFFER

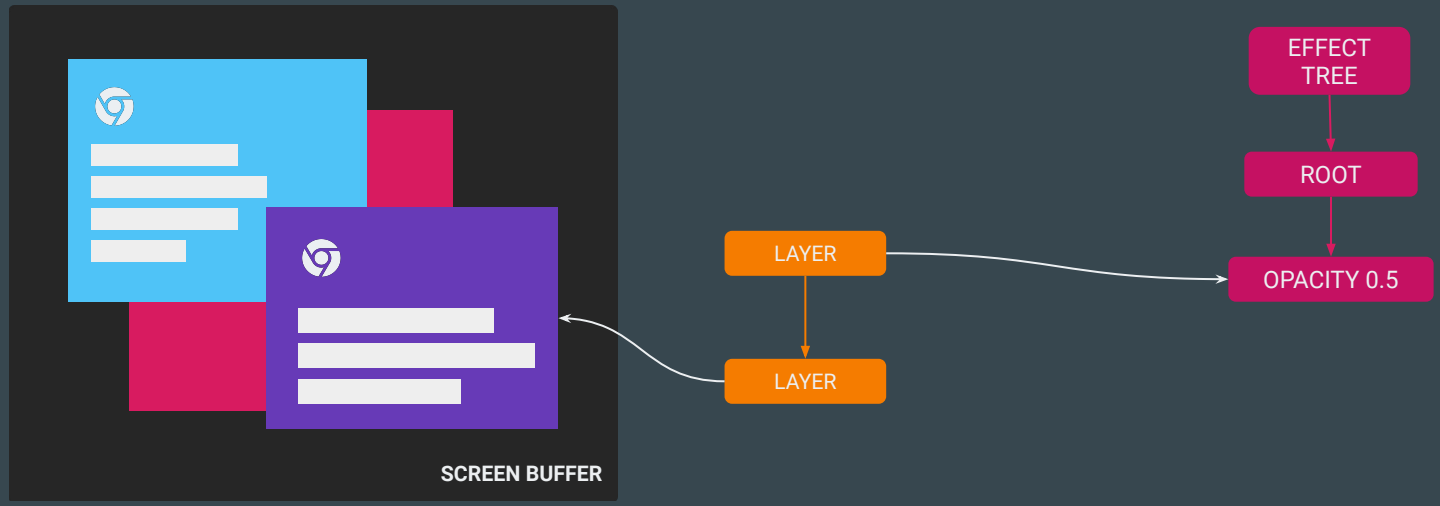
Effect Tree



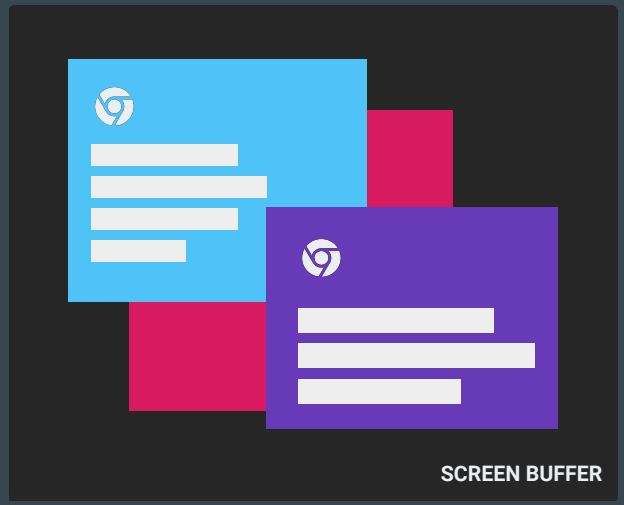
Effect Tree

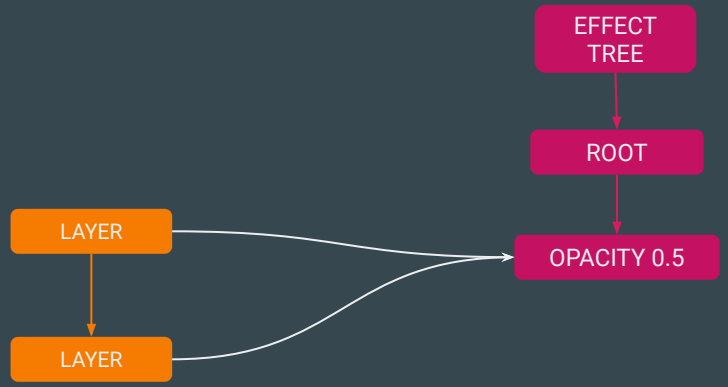


Effect Tree

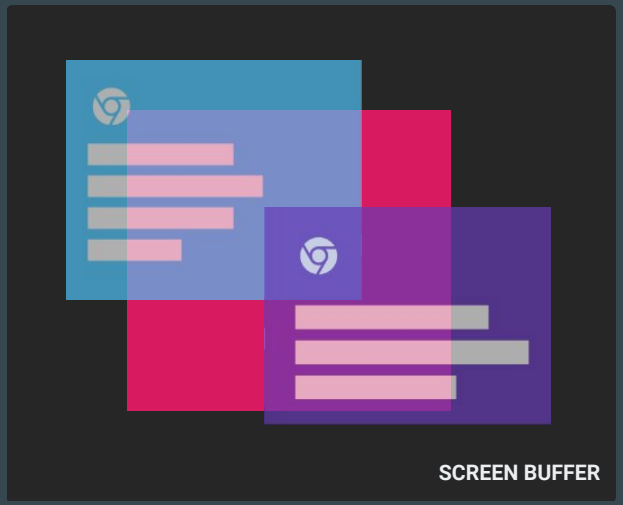


Effect Tree



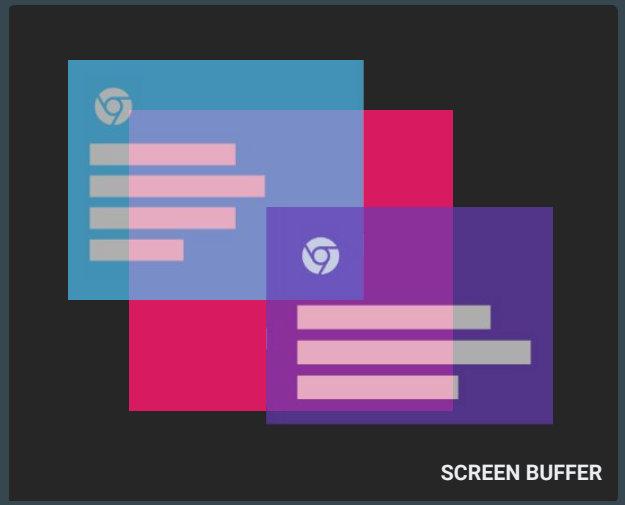


Effect Tree

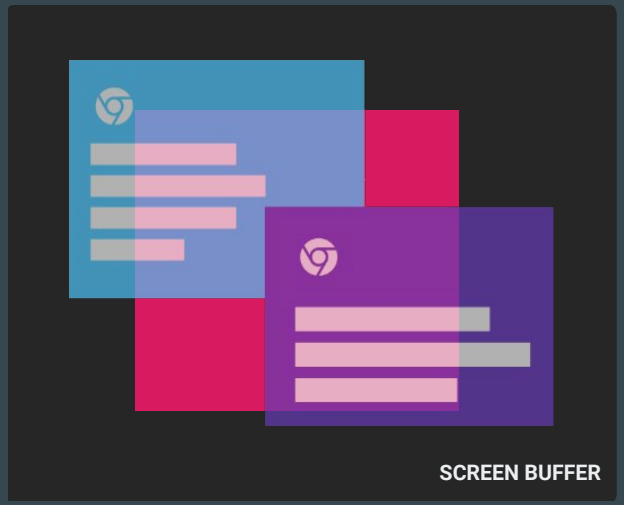


Drawing layer textures one by one will result in the two layer textures blending with each other.

Effect Tree



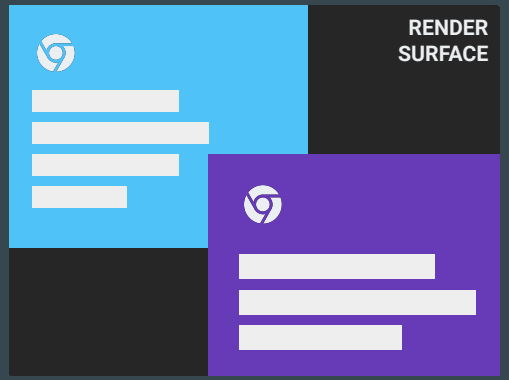
Drawing layer textures one by one will result in the two layer textures blending with each other.



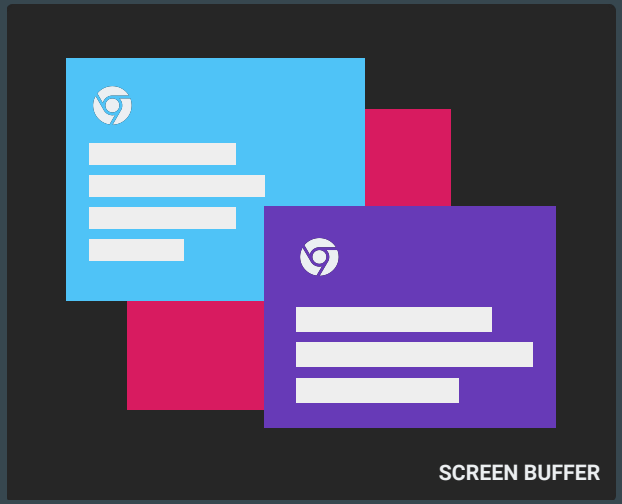
Desired result where the two layers don’t blend with each other and only blend with the background.

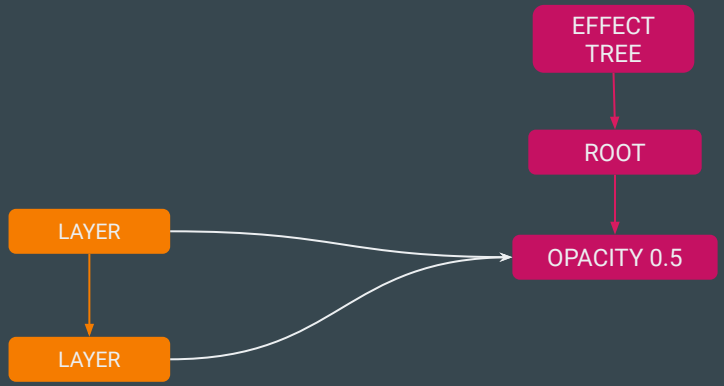
Render Surfaces

- Certain effects on a layer apply to the entire subtree rooted at that layer.
- Temporary buffers, called render surfaces, hold the draw output of the subtree and then applies an effect on the render surface buffer.
- Size of the render surface is the union of sizes of all the drawable layers in its subtree.
- Any property affecting the original layer now effects the render surface.
- Some operations that require this are:
  - opacity
  - filter (blur, contrast, saturation, etc.)
  - mask
  - non-axis-aligned clipping



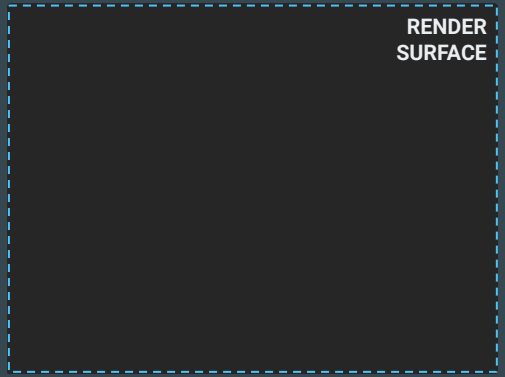
Going back to the previous example



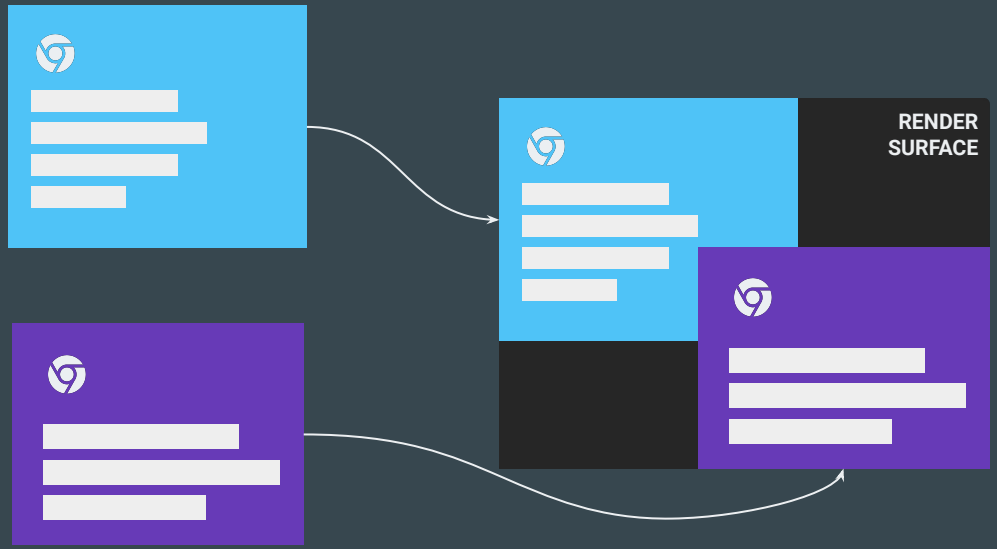


Render Surfaces

- Allocate a temporary buffer of the size of the subtree which is the union of sizes of all the layer in the subtree.

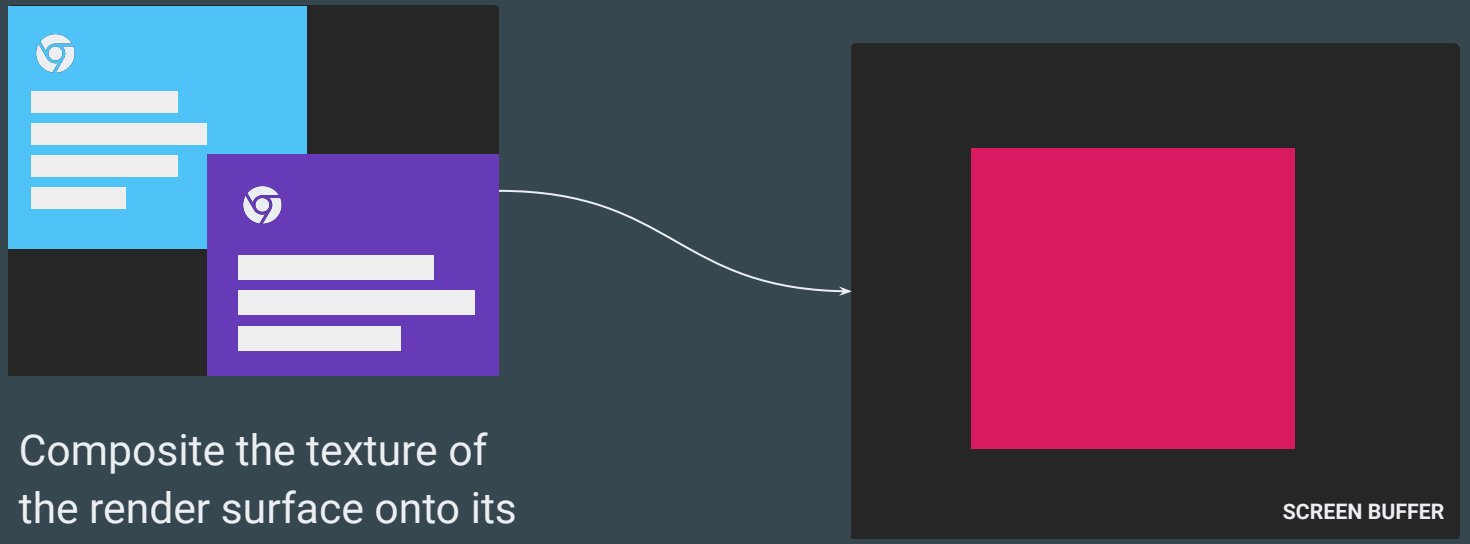


Render Surfaces



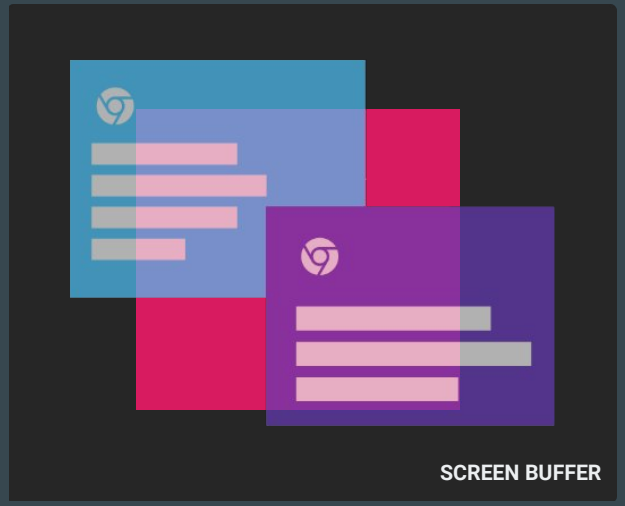
Draw the layer textures onto the render surface buffer

Render Surfaces

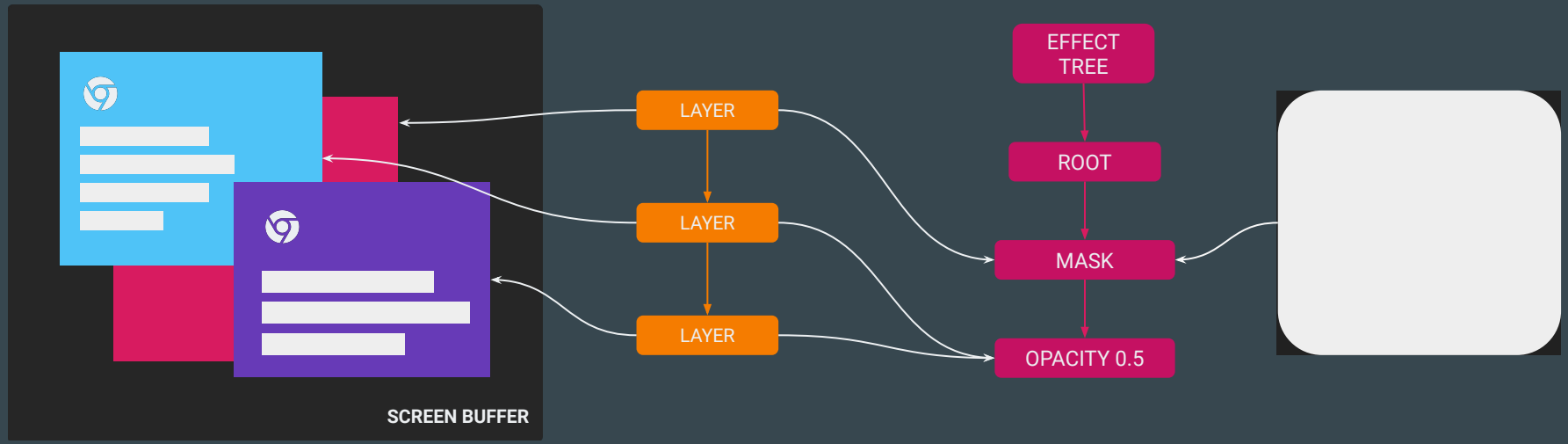


Composite the texture of the render surface onto its target buffer and apply the effect.

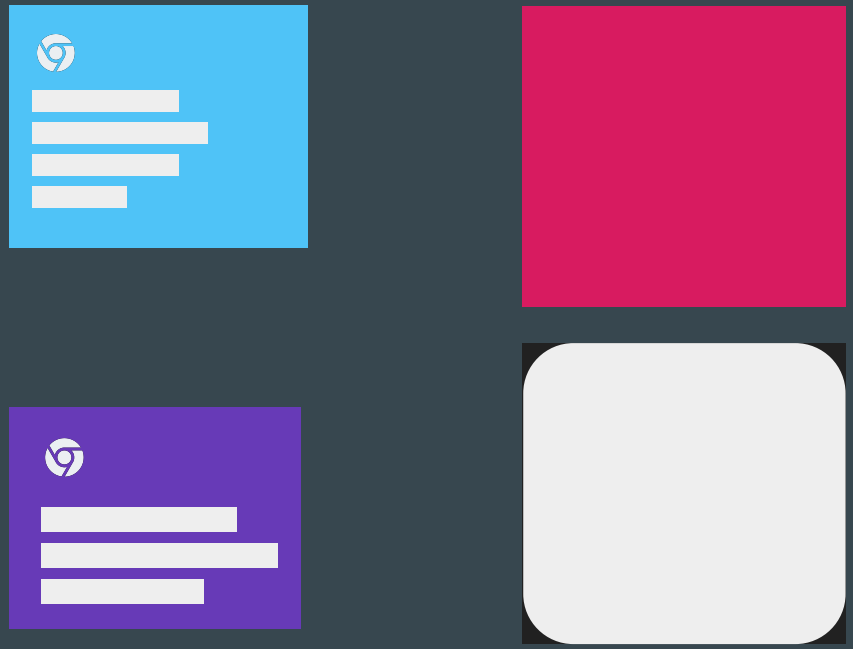
Render Surfaces



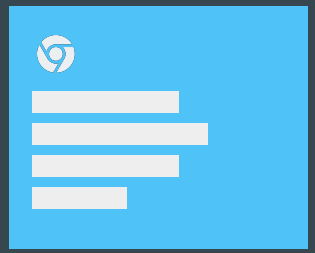
Memory Impact of Render Surfaces

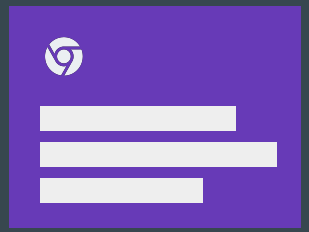


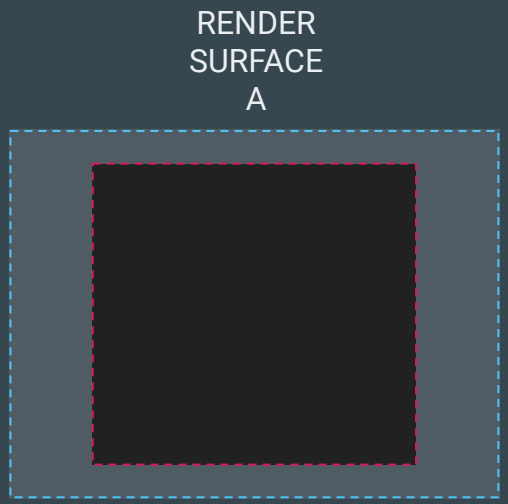
Textures already in memory

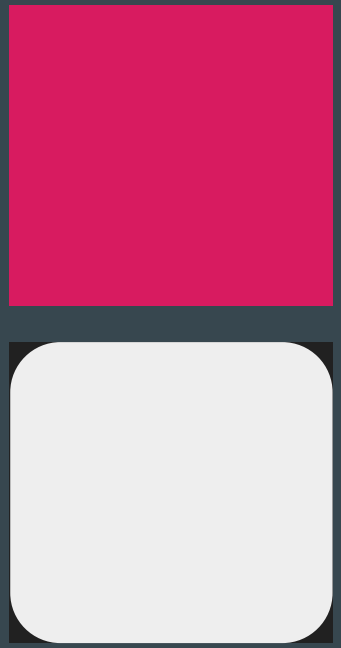


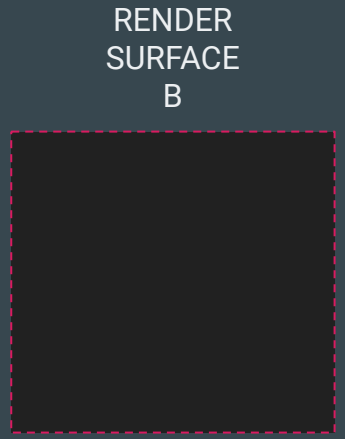
Additional Textures to allocate





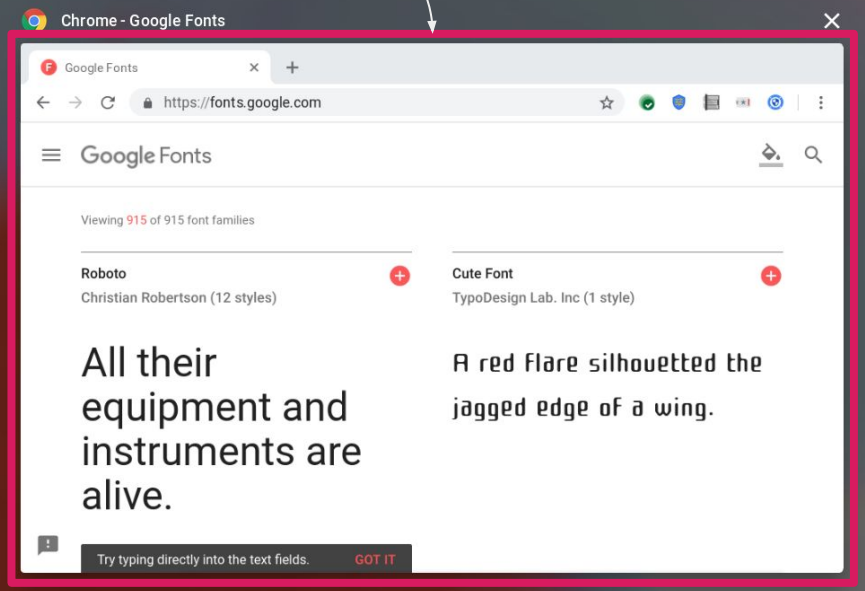






The clipped render surface buffer allocated is smaller due to clip from its parent.

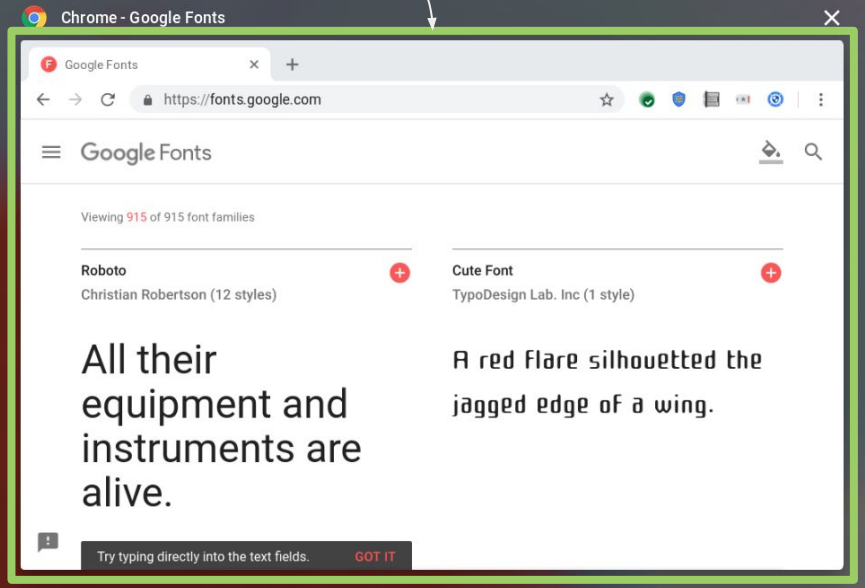
Window with rounded corner mask



This window could be maximized when not in overview mode. And would have a texture of the size of almost the entire screen. (In overview mode the textures are scaled down at draw step.)

On a pixel slate device with display resolution of 3000*2000, this means the texture is taking almost **~22mb**. To apply the rounded corner, we have a mask texture of the same size (**~22 mb**). And finally a render surface of the same size which requires another (**+22 mb along with allocation time for each frame**). That is ~**60-66 mb** per window.

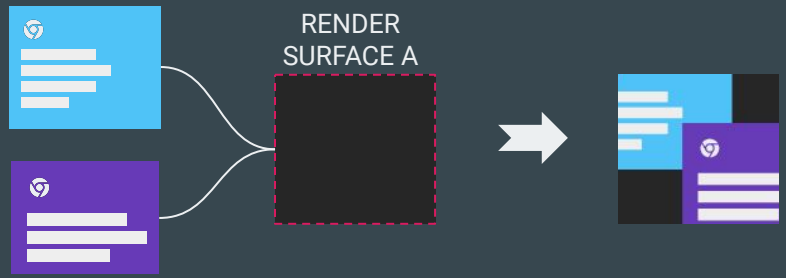
Window with new implementation of rounded corner



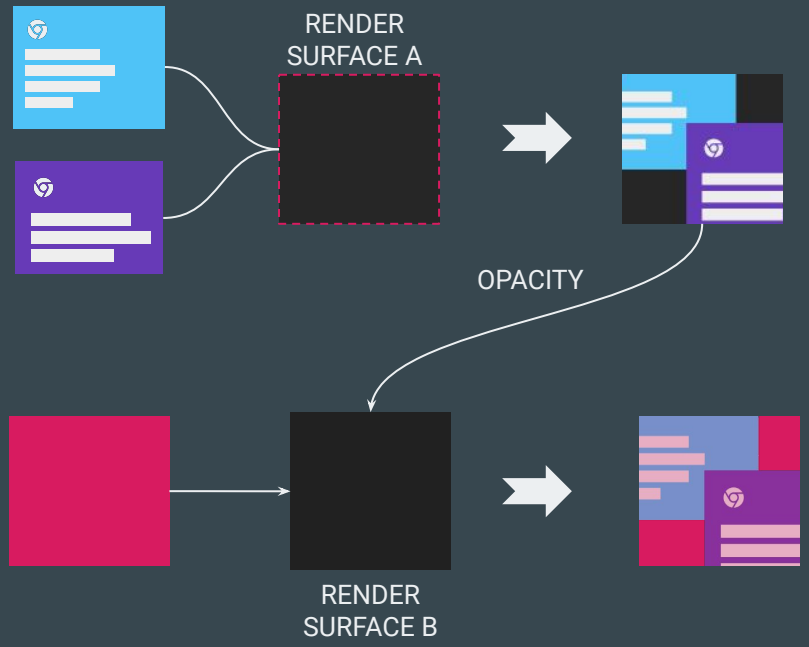
The new implementation of rounded corner does away with the use of render surface and mask layers. Instead it performs the rounding at the draw step of the pipeline by using OpenGL shaders.

The current memory usage for overview mode per window is **~22mb**. At the same time, the fps increase measured is around **10-15 fps** for pixel slate.

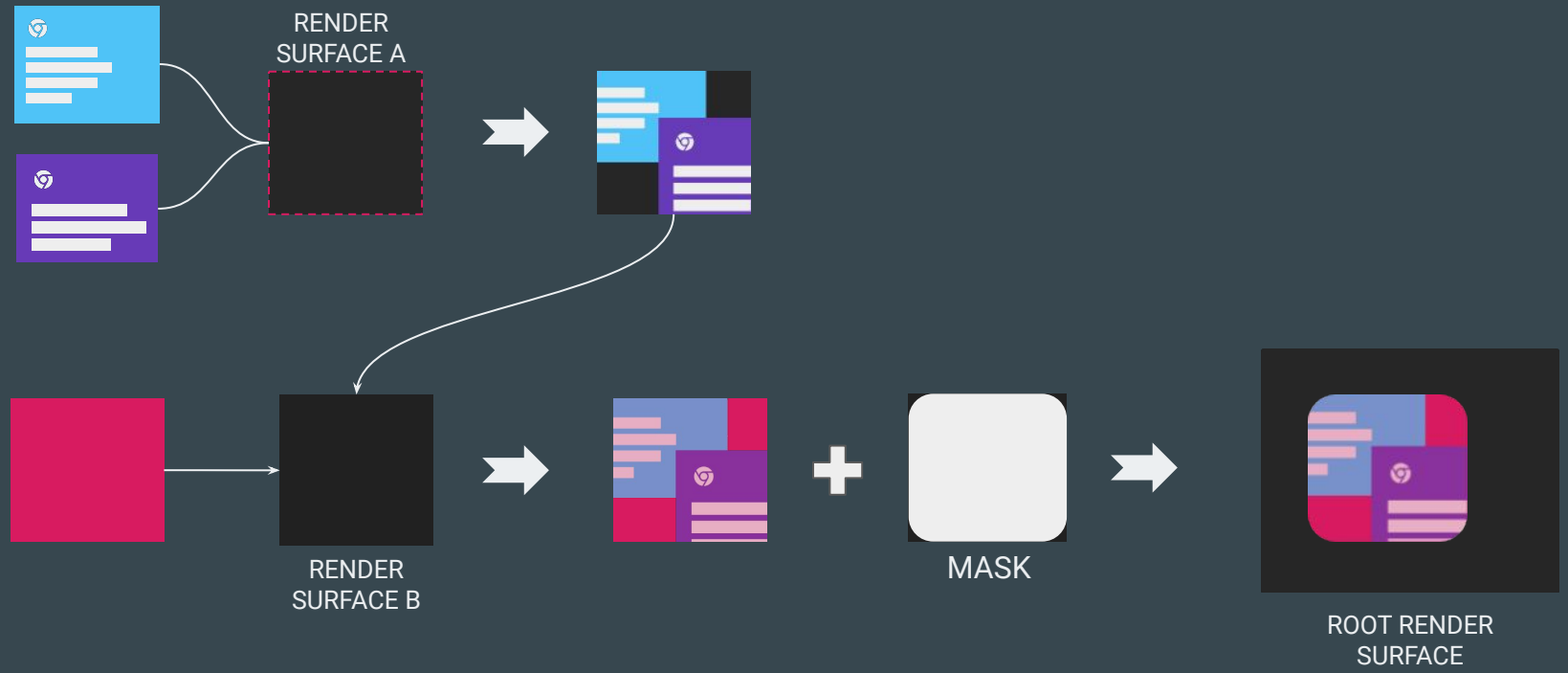
Draw



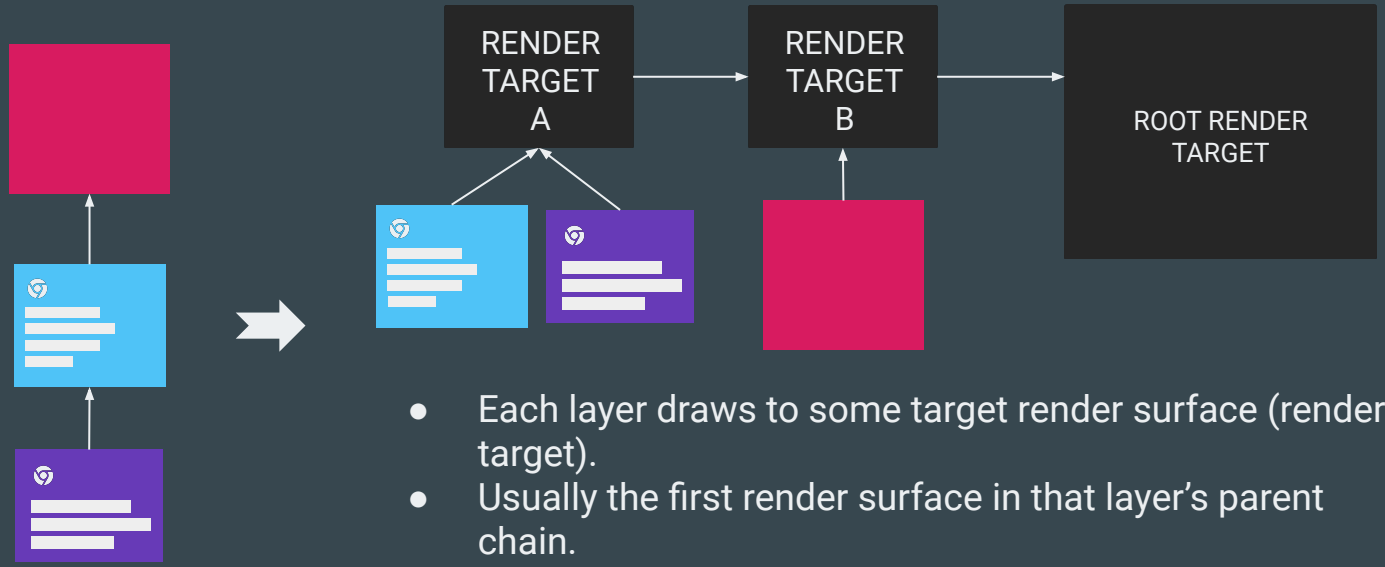
Draw



Draw



Render Target



- Each layer draws to some target render surface (render target).
- Usually the first render surface in that layer's parent chain.
- A layer that points to an property tree node (from any tree), respects all the nodes in its parent chain up to its render target.

Output of Property Trees

Layer draw properties

- Draw transform
- Draw opacity
- Clip bounds
- Visible bounds
- Render Target

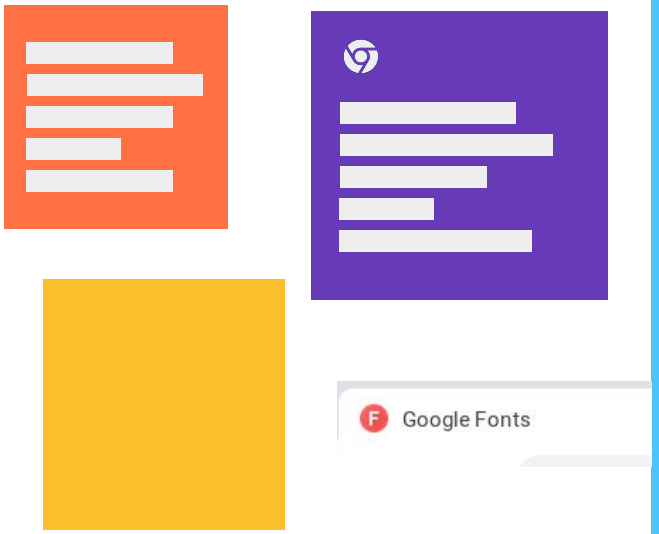
And a list of render surfaces...

Render surface draw properties

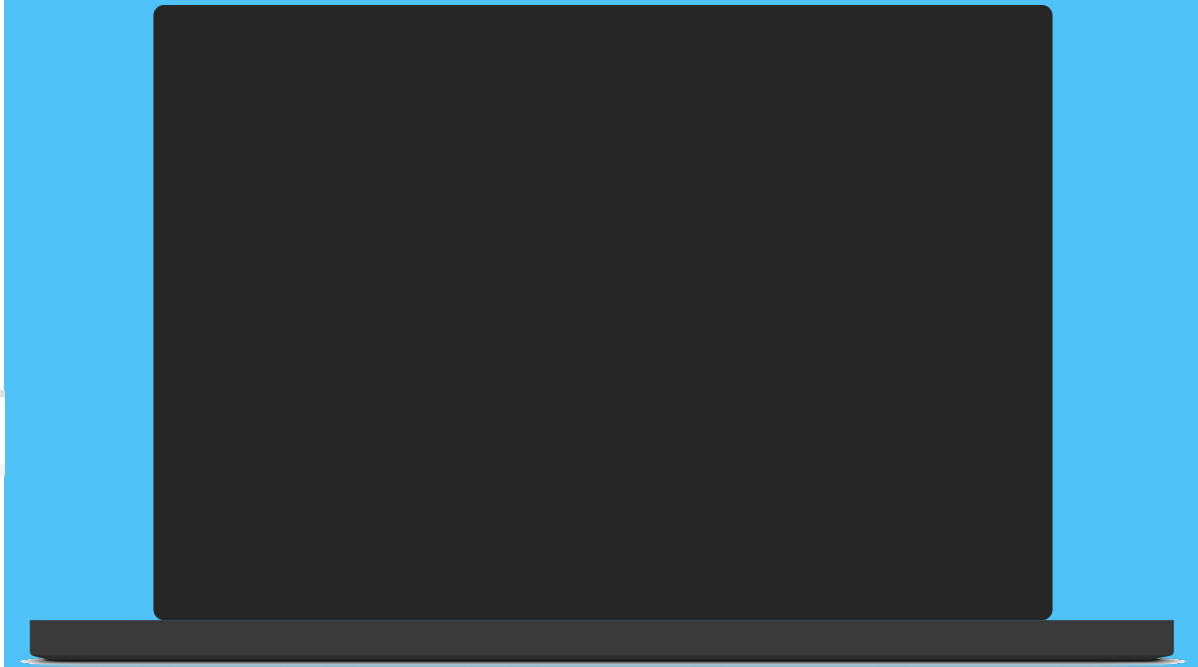
- Screen space transform
- Draw transform
- List of effects
- Content bounds
- Clip rect

Output of cc - Draw Quads

- Each quad holds a resource that will be is composited onto its target buffer.
- A draw quad additionally holds a bag of data used for drawing a quad onto the buffer.
- Depending on what kind of resource the quad holds, there are different kinds of draw quads including:
  - Solid Color
  - Texture
  - Tile
  - Surface
  - Video
  - Render Pass

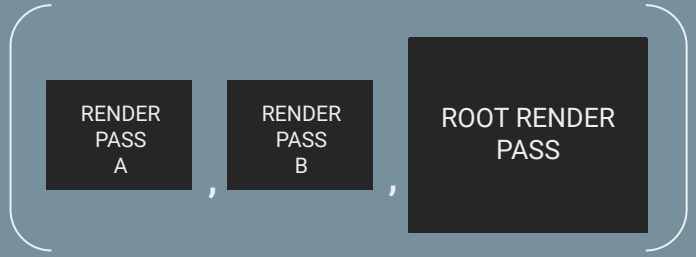


We now have all the info needed to put textures and actual pixels on the screen.

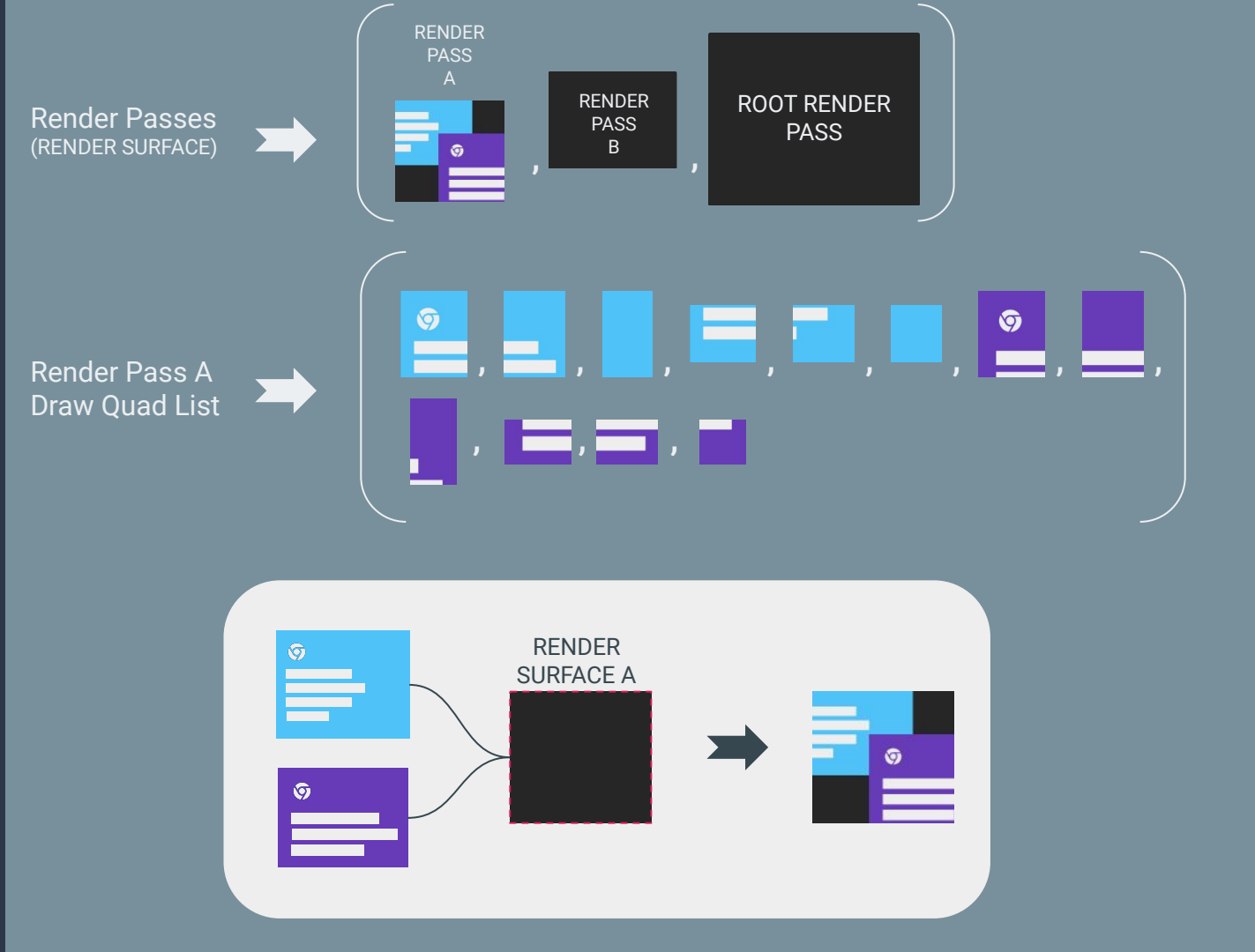


Compositor Frame

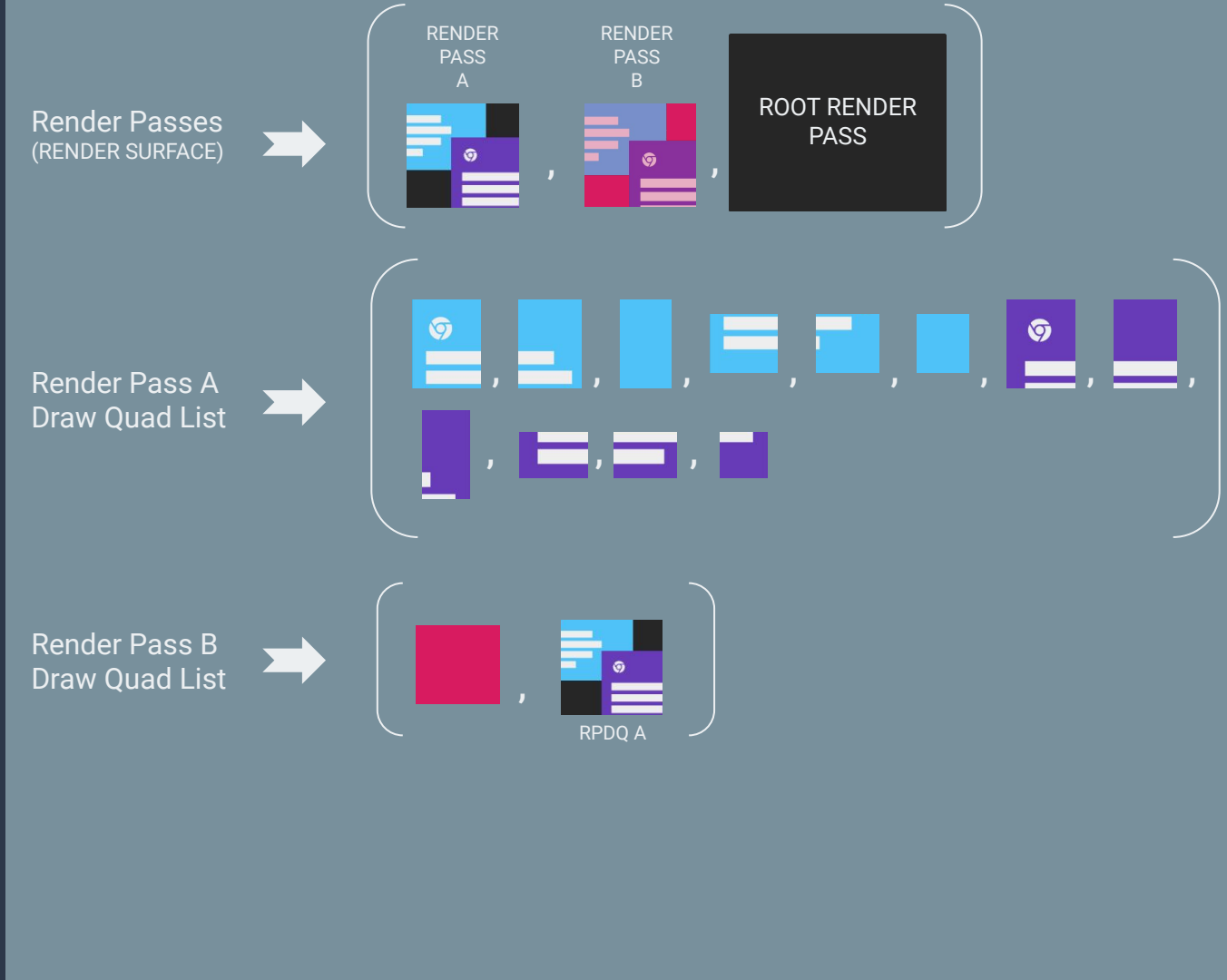
Render Passes
(RENDER SURFACE)



Compositor Frame

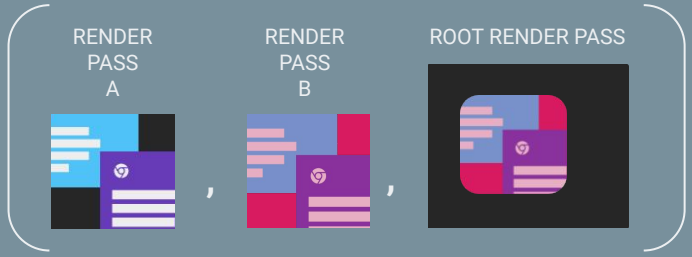


Compositor Frame

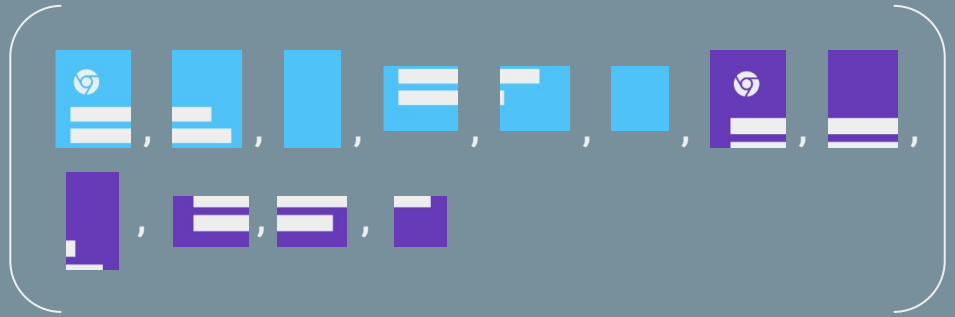


Compositor Frame

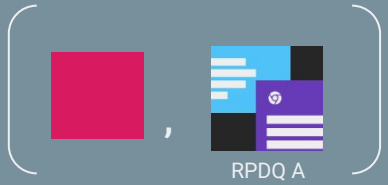
Render Passes
(RENDER SURFACE)



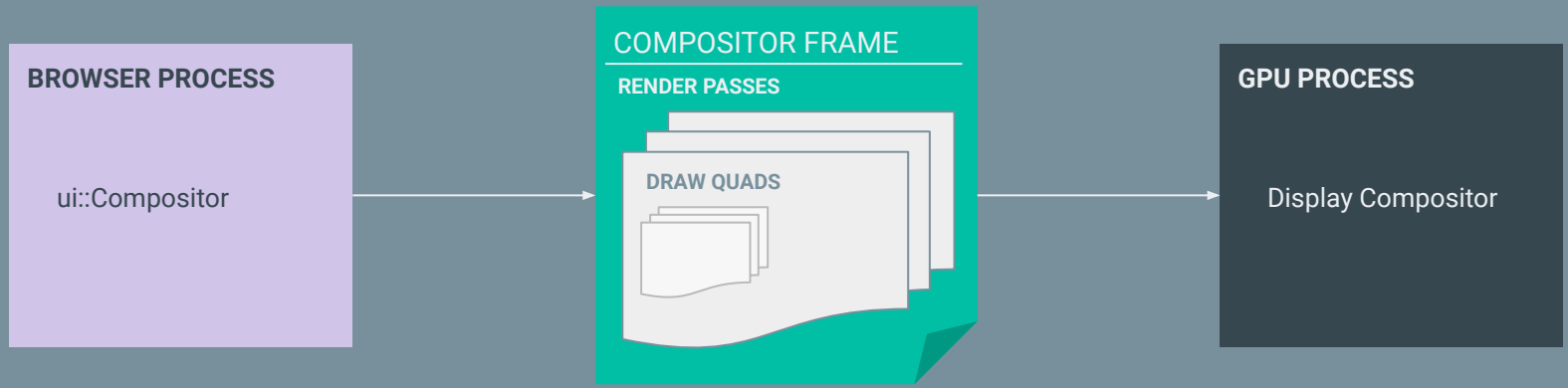
Render Pass A
Draw Quad List

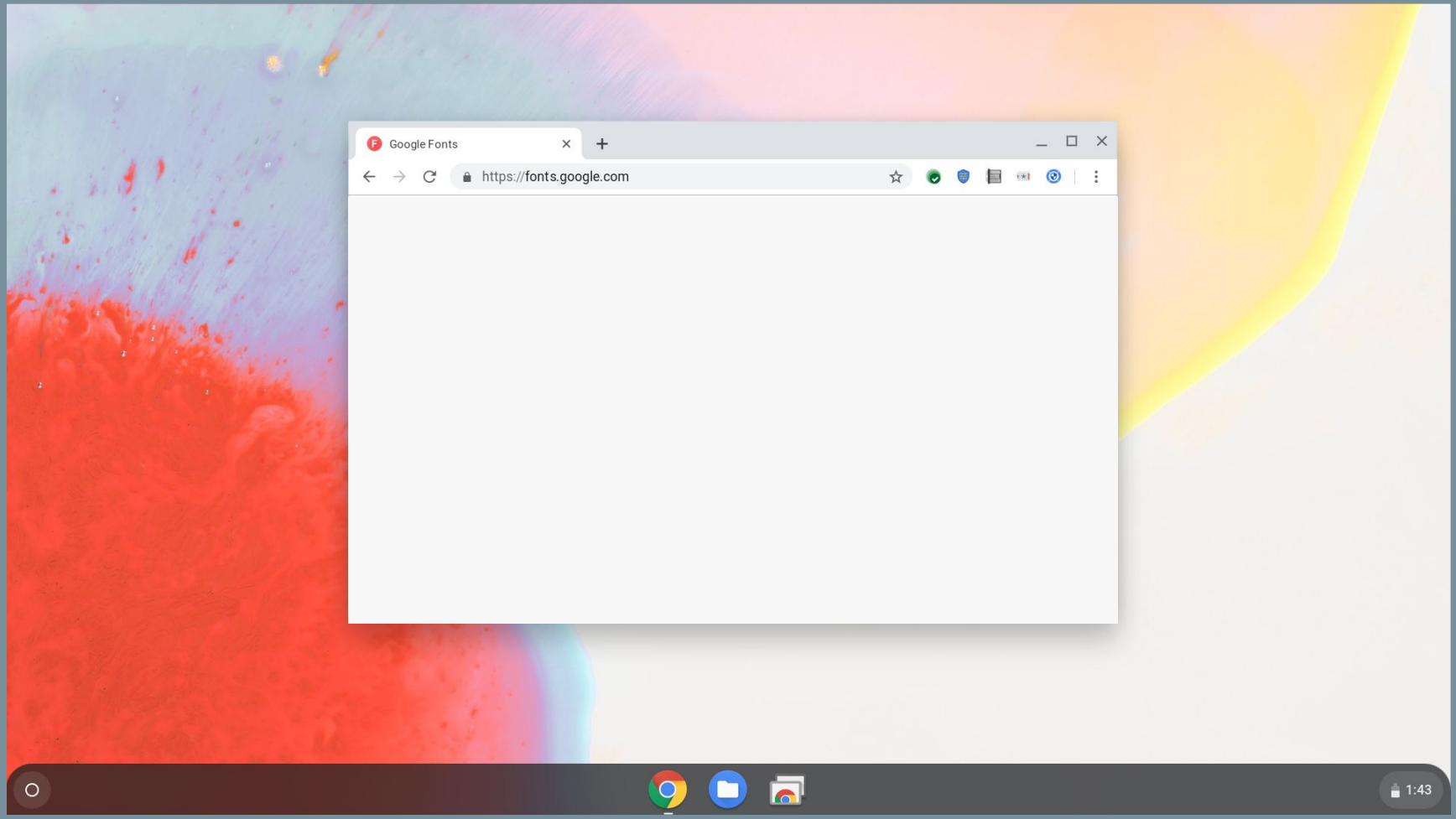


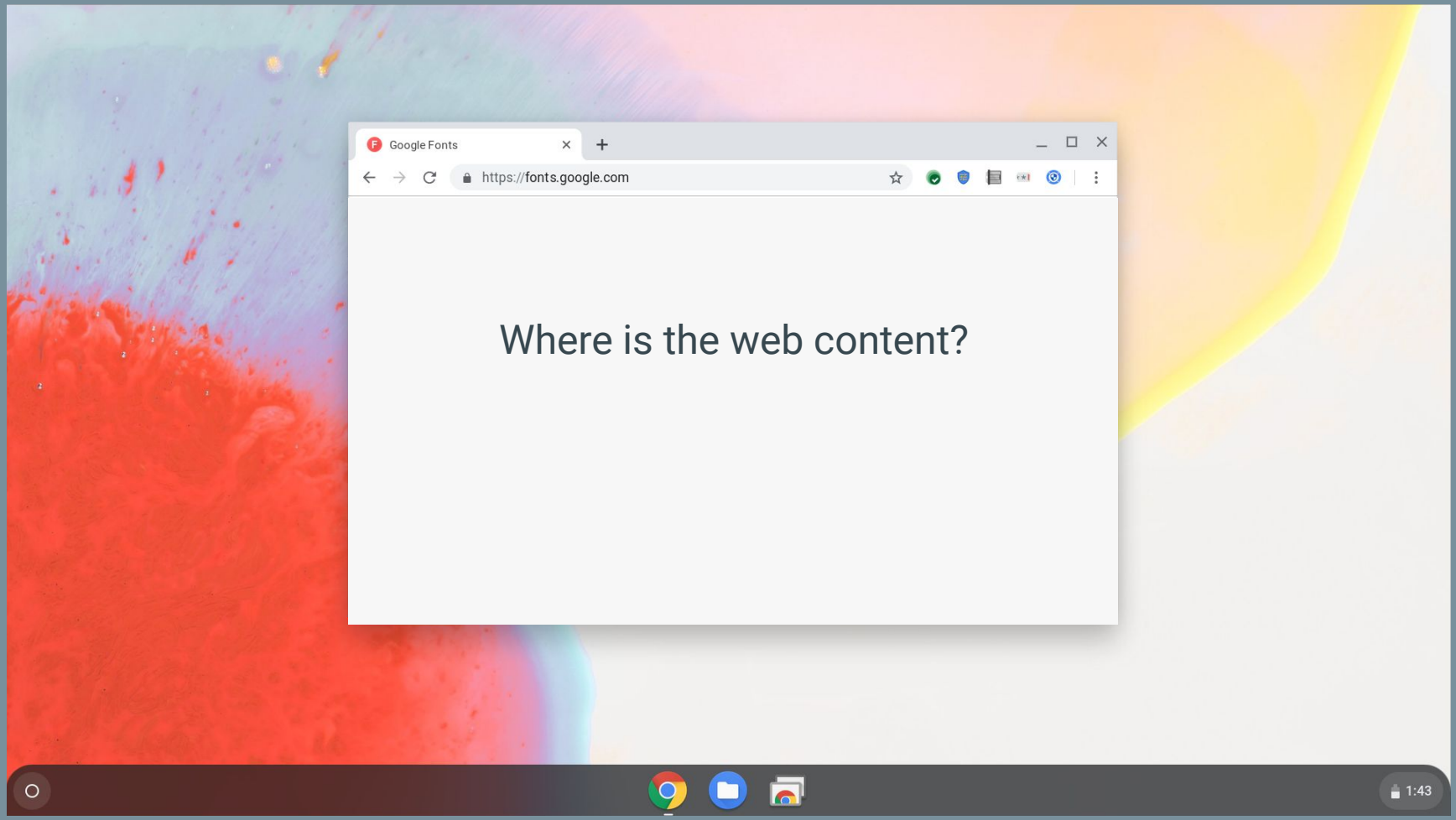
Render Pass B
Draw Quad List



Root Render Pass
Draw Quad List


BROWSER PROCESS
ui::Compositor
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS
GPU PROCESS
Display Compositor


Google Fonts
https://fonts.google.com
1:43


Google Fonts
https://fonts.google.com
Where is the web content?
1:43


ROOT
SOLID
TEXTURE
NO DRAWN
SURFACE
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS
Draw Quad a
Surface Draw Quad b
Draw Quad c

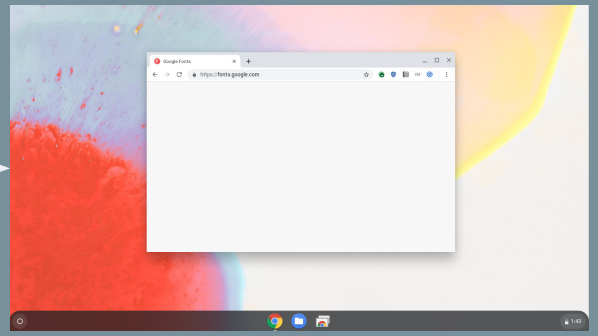
BROWSER PROCESS

ui::Compositor

COMPOSITOR FRAME

RENDER PASSES

DRAW QUADS



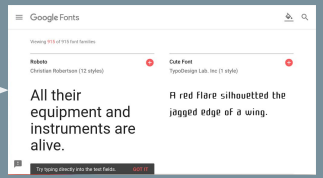
RENDERER PROCESS

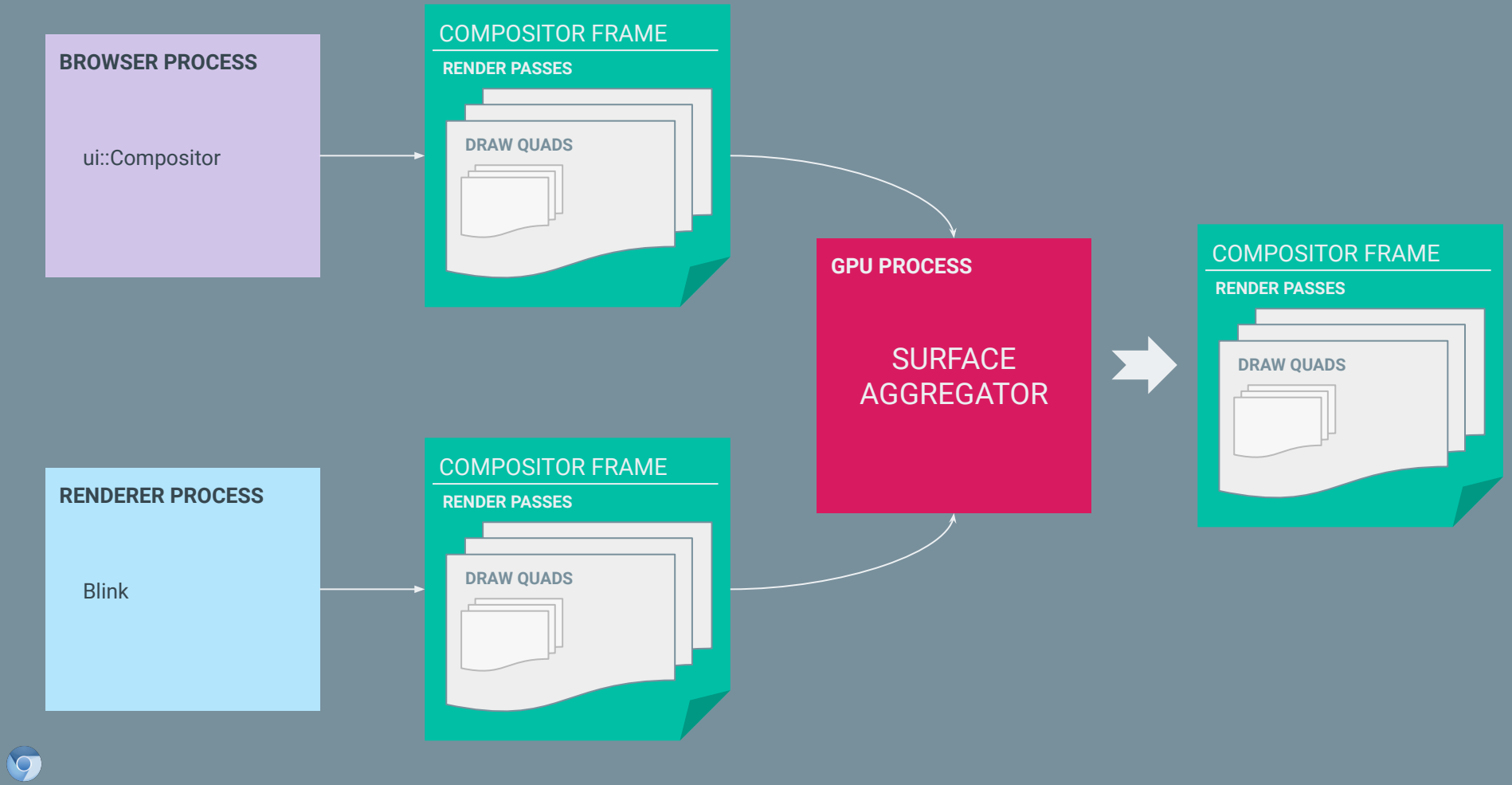
Blink

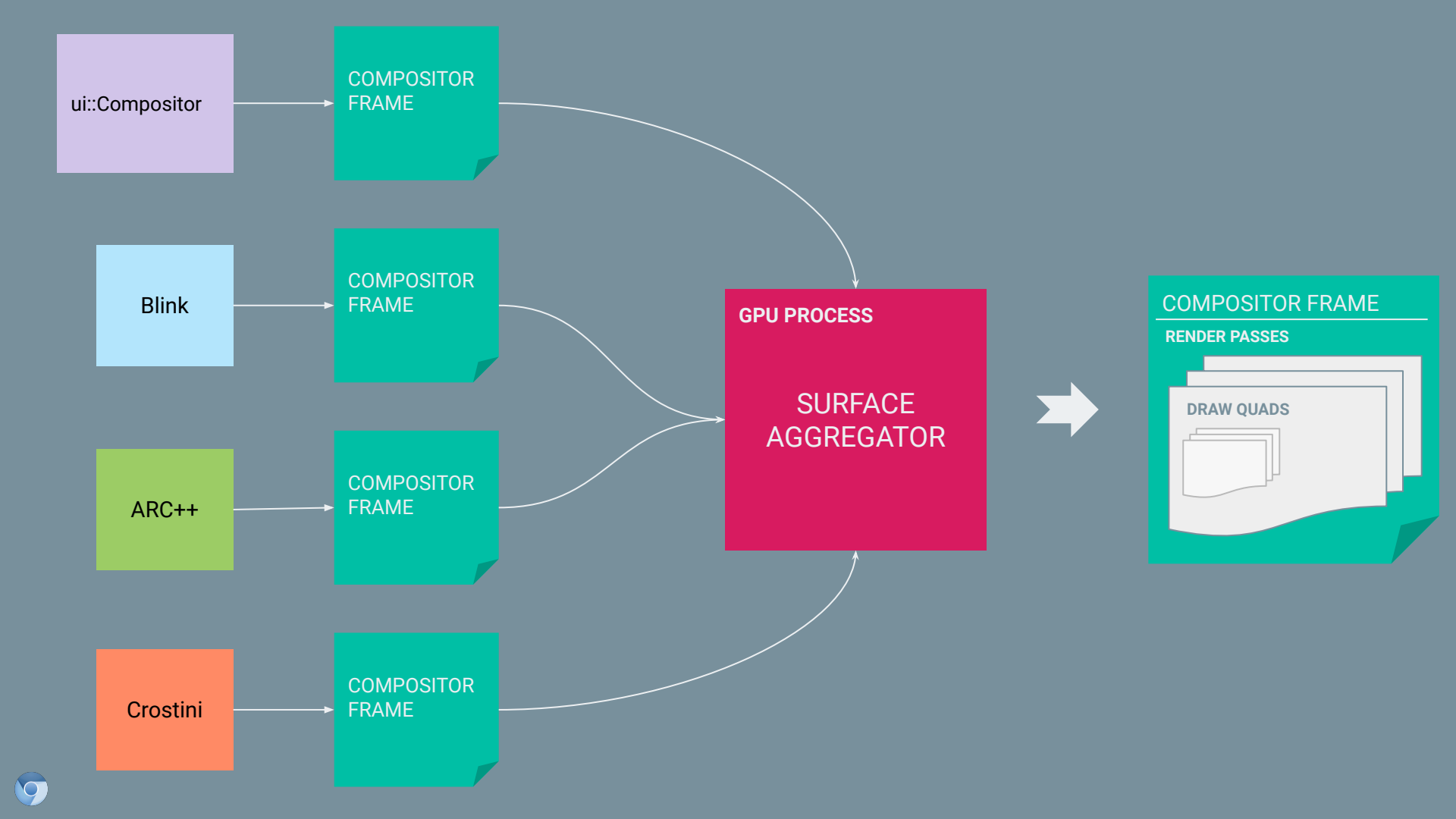
COMPOSITOR FRAME

RENDER PASSES

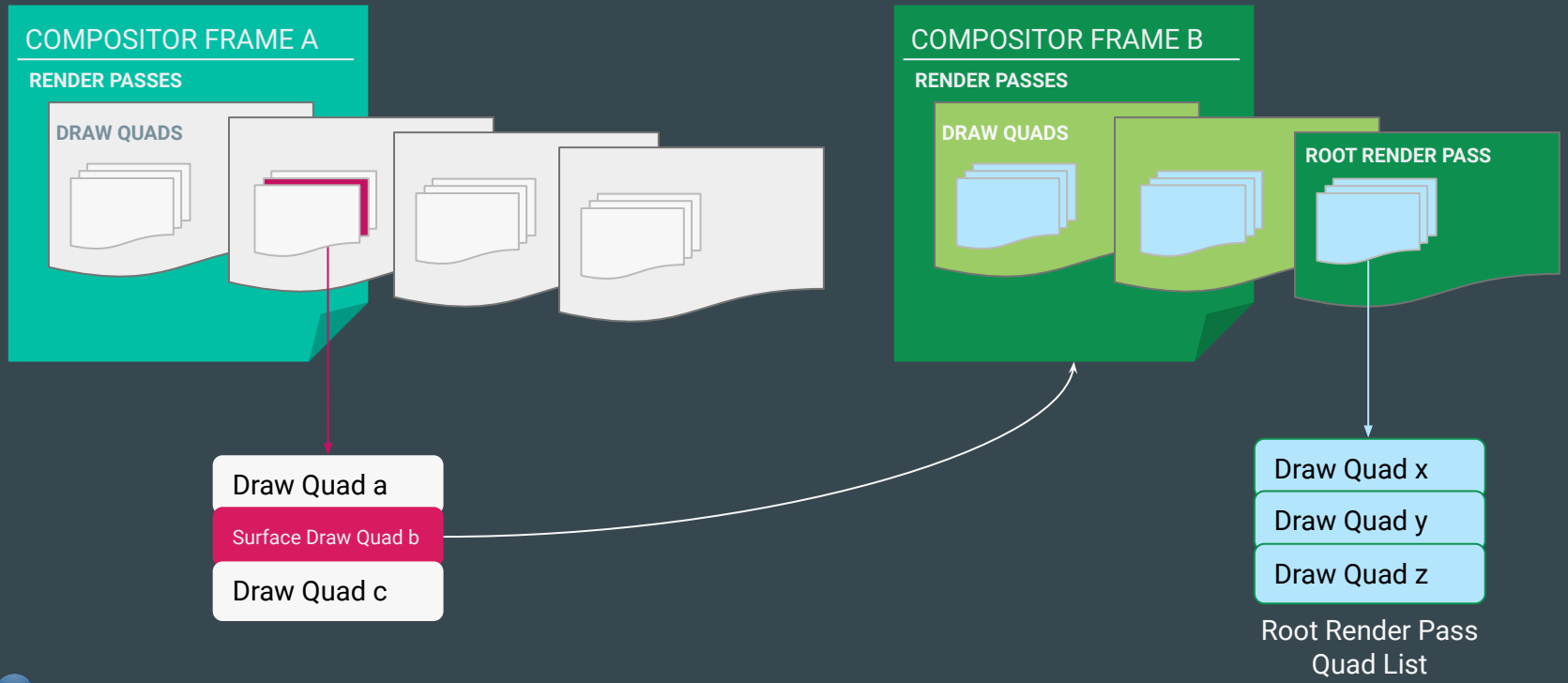
DRAW QUADS



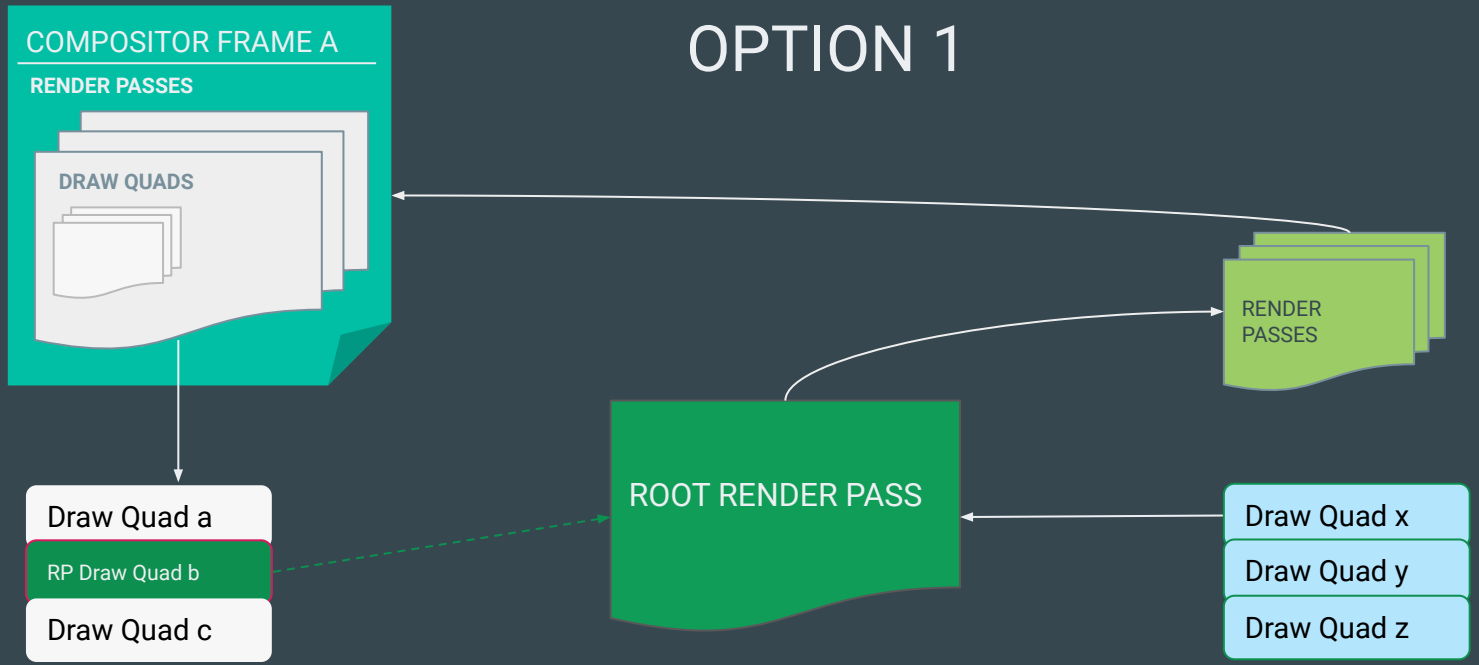

BROWSER PROCESS
ui::Compositor
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS
RENDERER PROCESS
Blink
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS
GPU PROCESS
SURFACE AGGREGATOR
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS


ui::Compositor
COMPOSITOR FRAME
Blink
COMPOSITOR FRAME
ARC++
COMPOSITOR FRAME
Crostini
COMPOSITOR FRAME
GPU PROCESS
SURFACE AGGREGATOR
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS

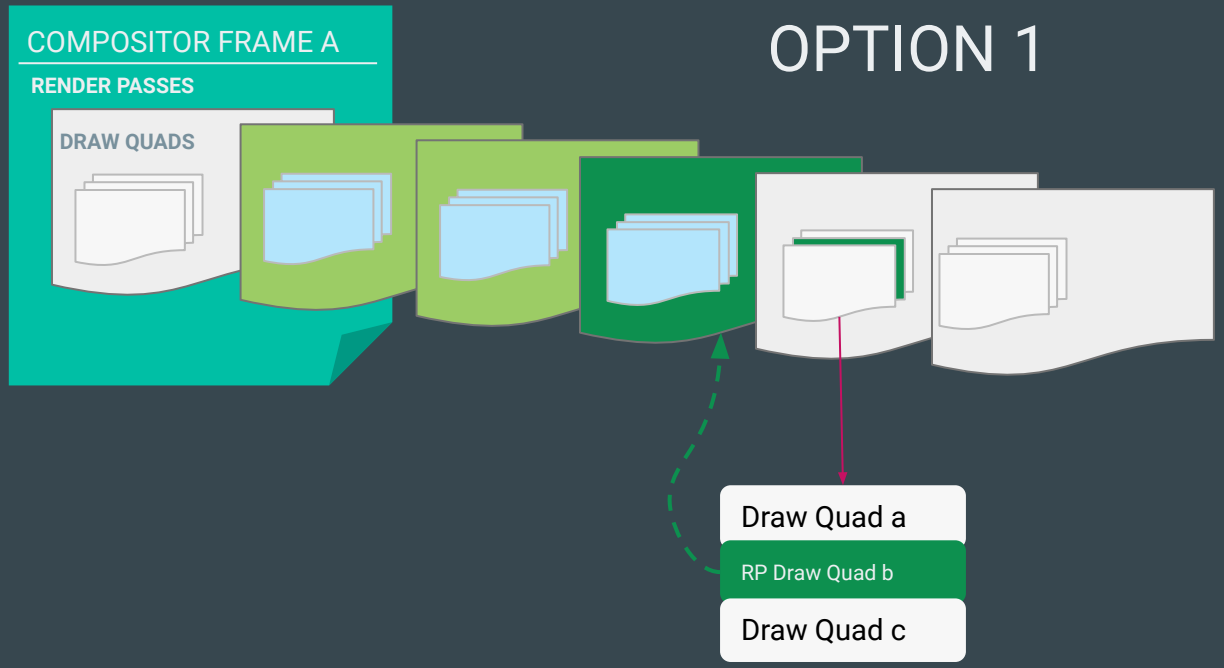
Ways for a Surface Aggregator to aggregate



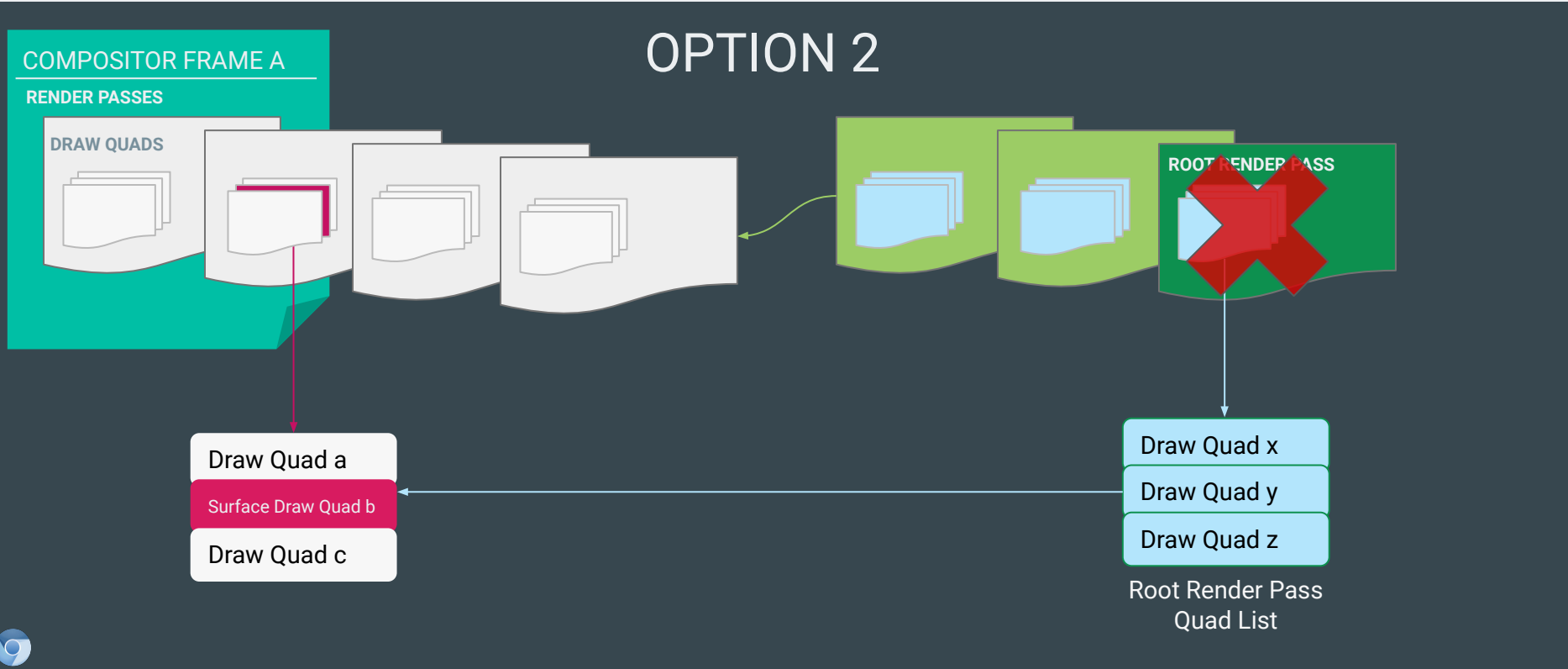
Ways for a Surface Aggregator to aggregate



Ways for a Surface Aggregator to aggregate

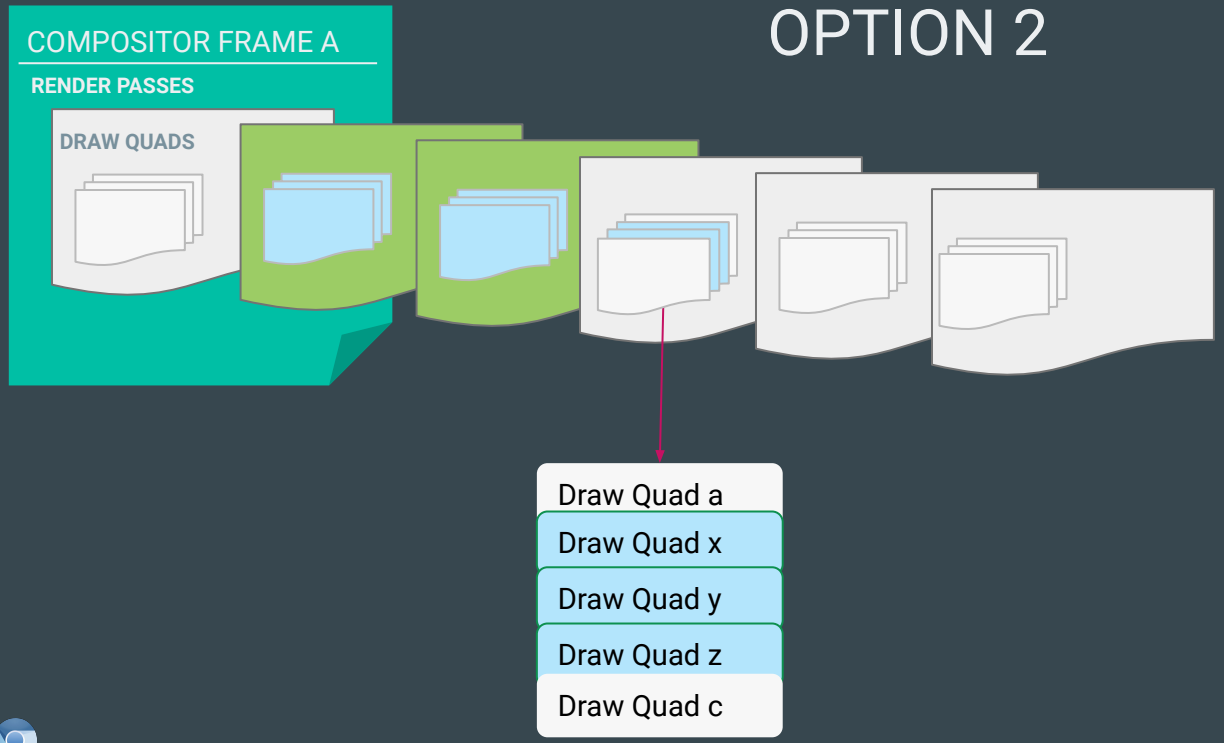


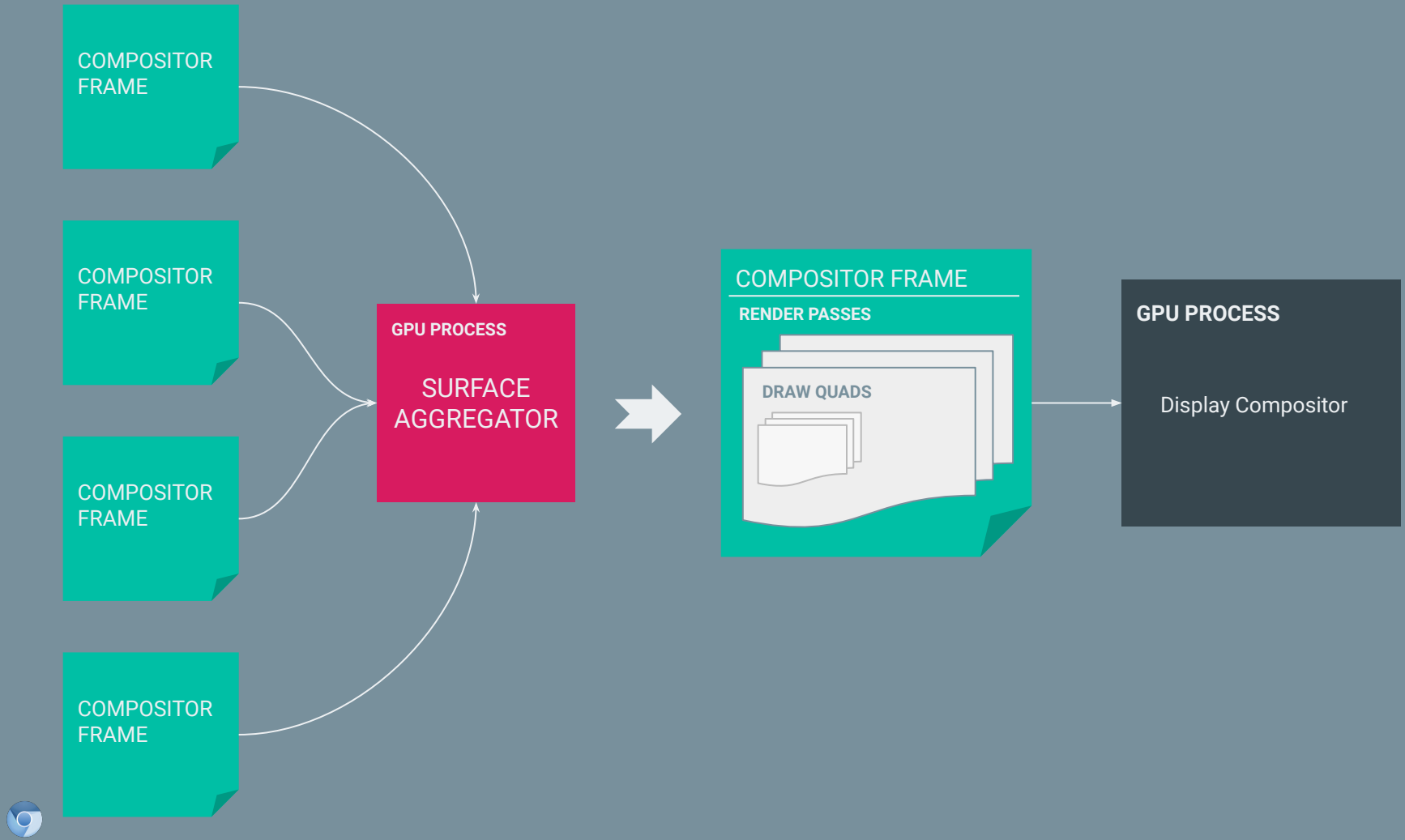
Ways for a Surface Aggregator to aggregate



Ways for a Surface Aggregator to aggregate

OPTION 2



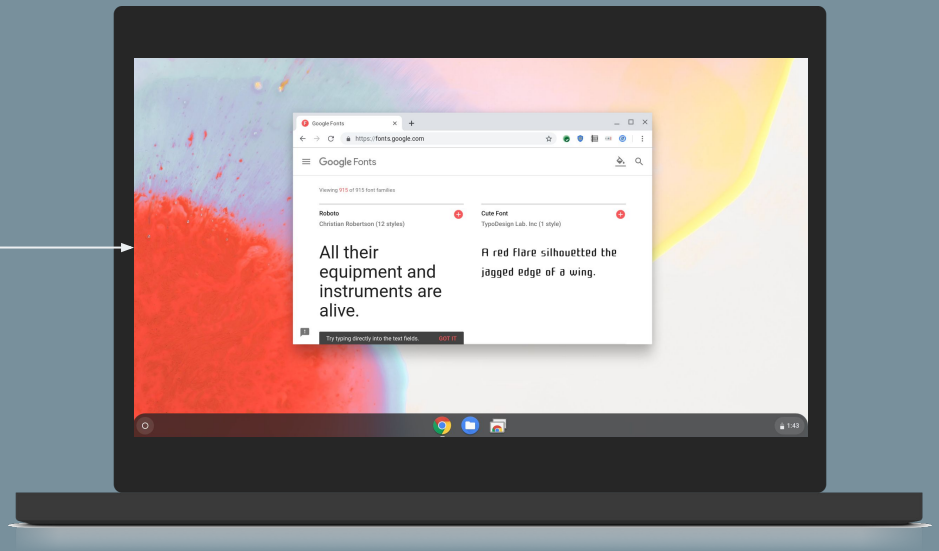

COMPOSITOR FRAME
COMPOSITOR FRAME
COMPOSITOR FRAME
COMPOSITOR FRAME
GPU PROCESS
SURFACE AGGREGATOR
COMPOSITOR FRAME
RENDER PASSES
DRAW QUADS
GPU PROCESS
Display Compositor

GPU PROCESS

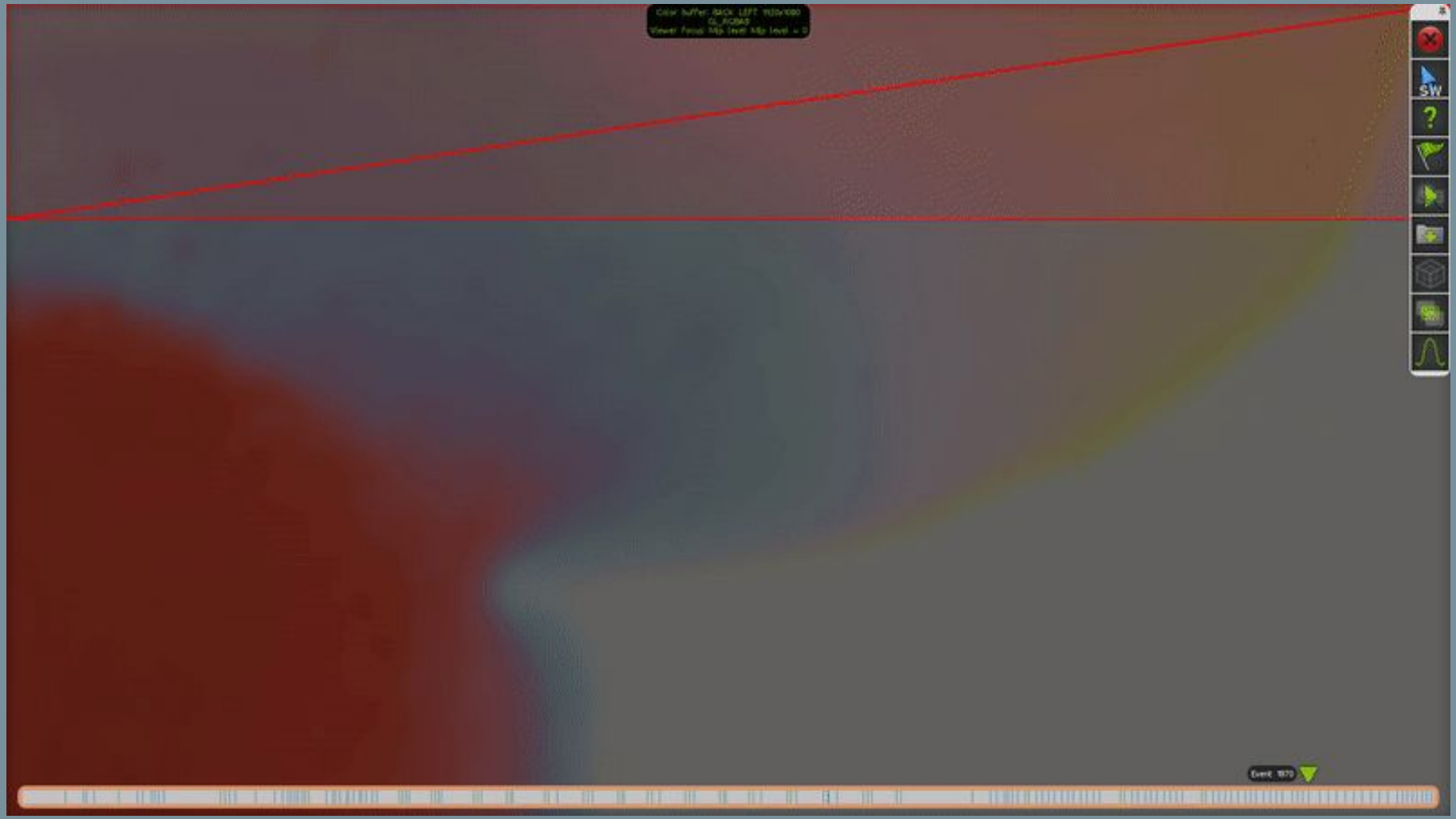
Display Compositor

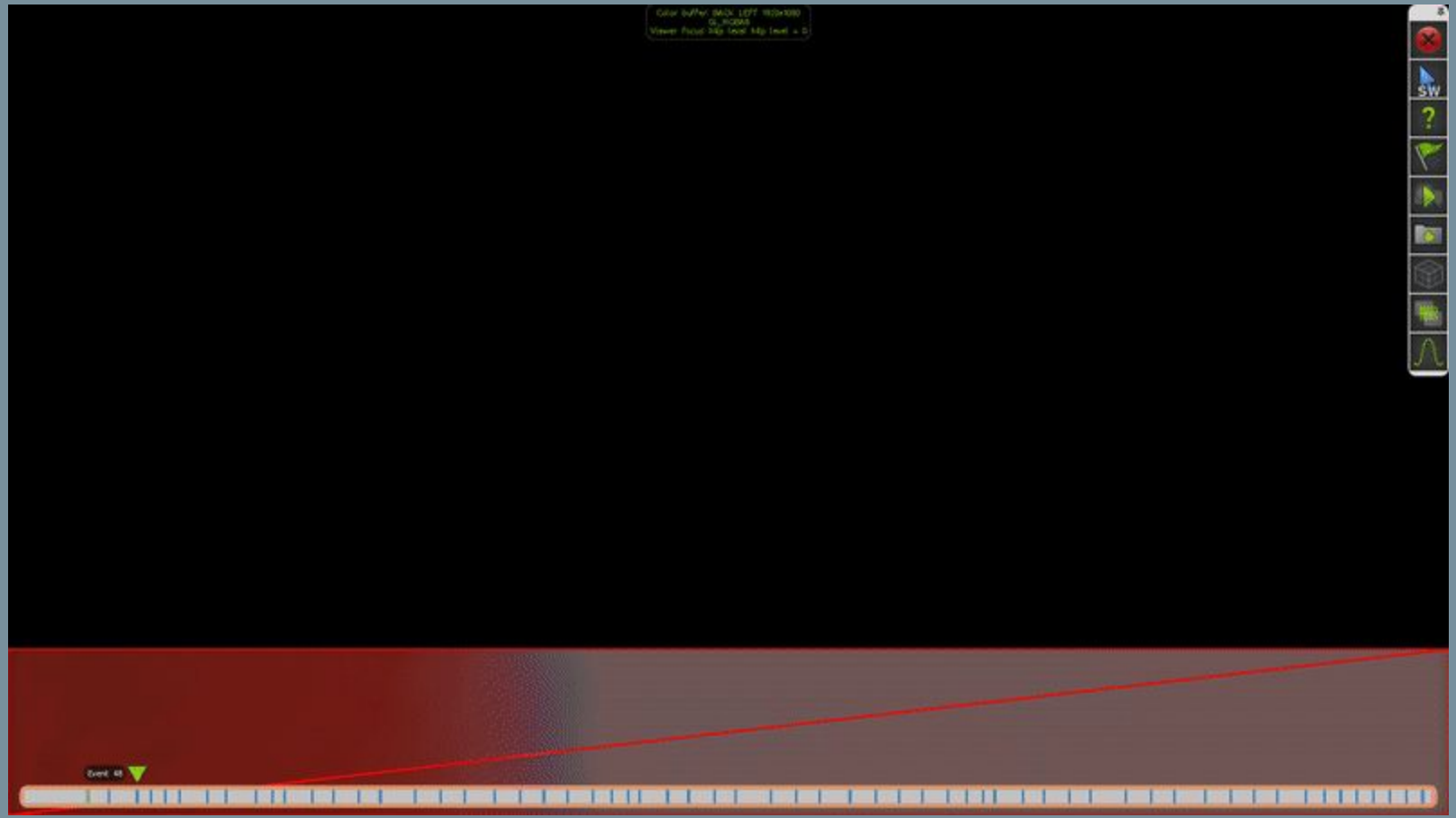
GPU PROCESS

Display Renderer




FBO attachment GL_TEXTURE_2D COLOR 0
1280x768
Internal Format GL_RGBA8 (0x8058)
Viewer Focus: Mip level: Mip level = 0
SW
Event 1474


SW


SW

Thank you!

Questions?

GPU Brownbag for Surface aggregation: Link

GPU Brownbag for OOP-Display Compositor: Link