# To CompositorFrameSink And Beyond

danakj@chromium.org

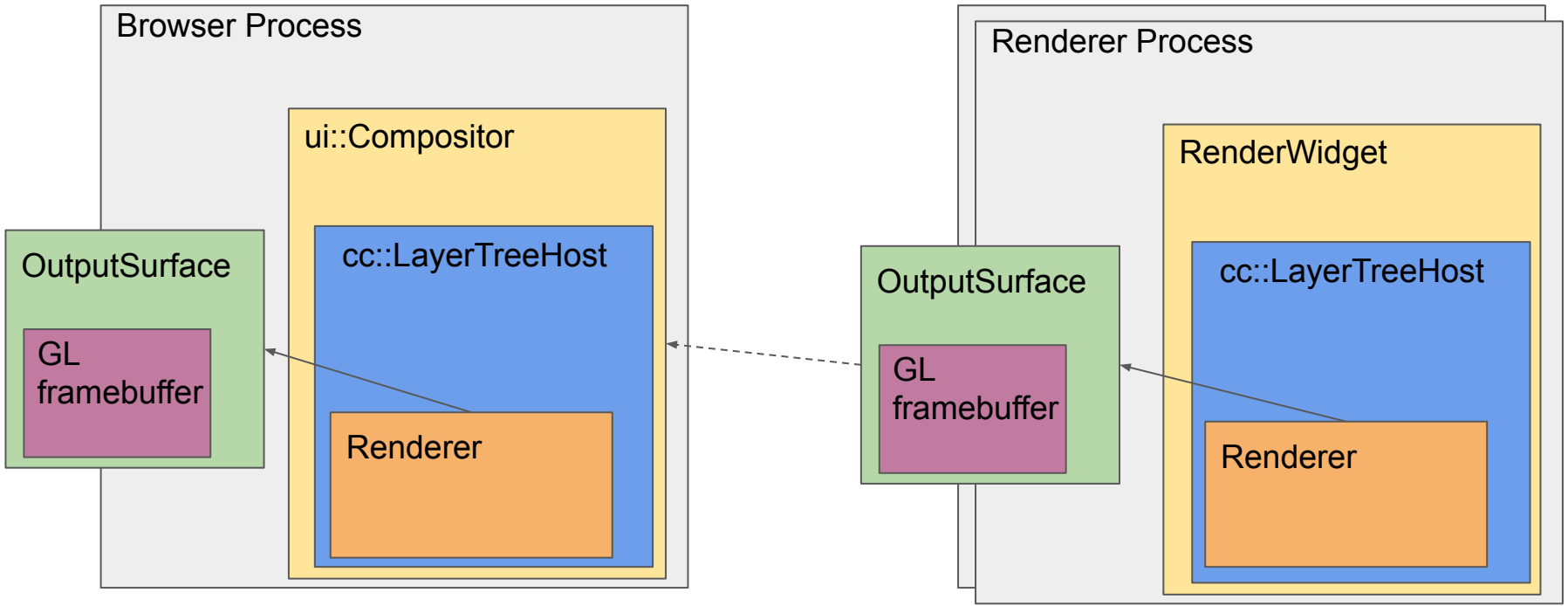
# Types of compositors through time

- Round 1: Prehistoric
  - Compositor (renderer and browser process)

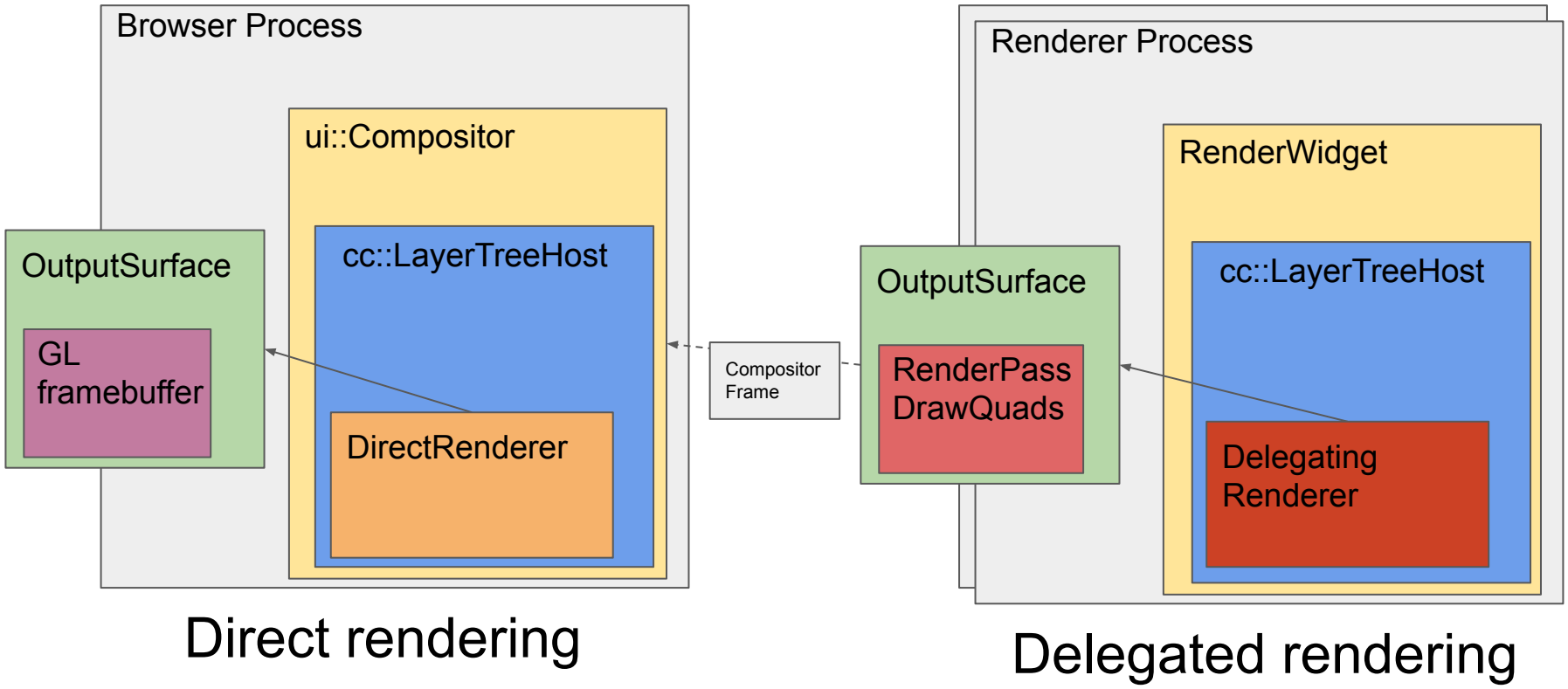
- Round 2: Ubercompositor
  - Delegating Compositor (renderer process)
  - Direct Compositor (browser process)

- Round 3: Surfaces
  - Layer Compositor (LayerTreeHost + LayerTreeHostImpl)
  - Display Compositor (Display)

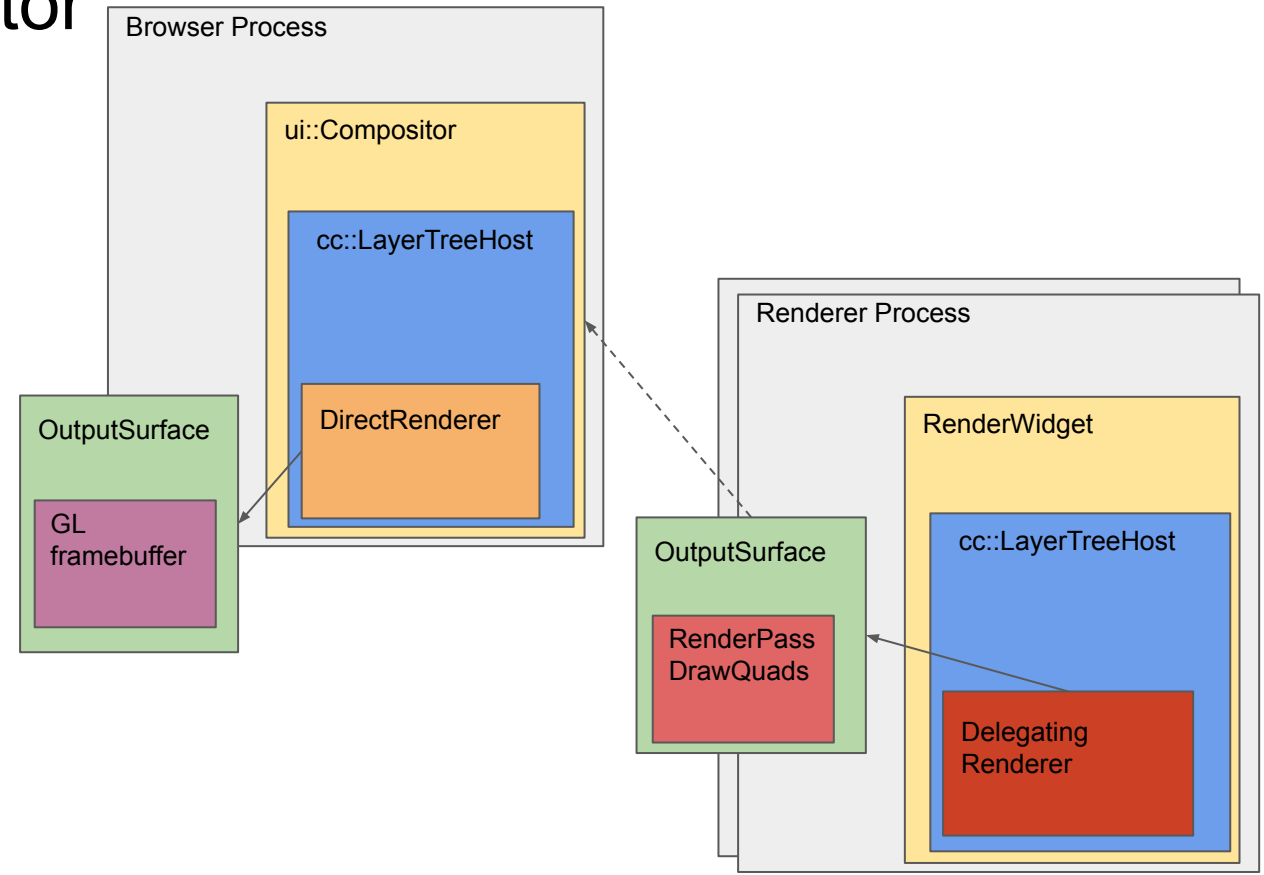
# 1. Prehistoric Compositor



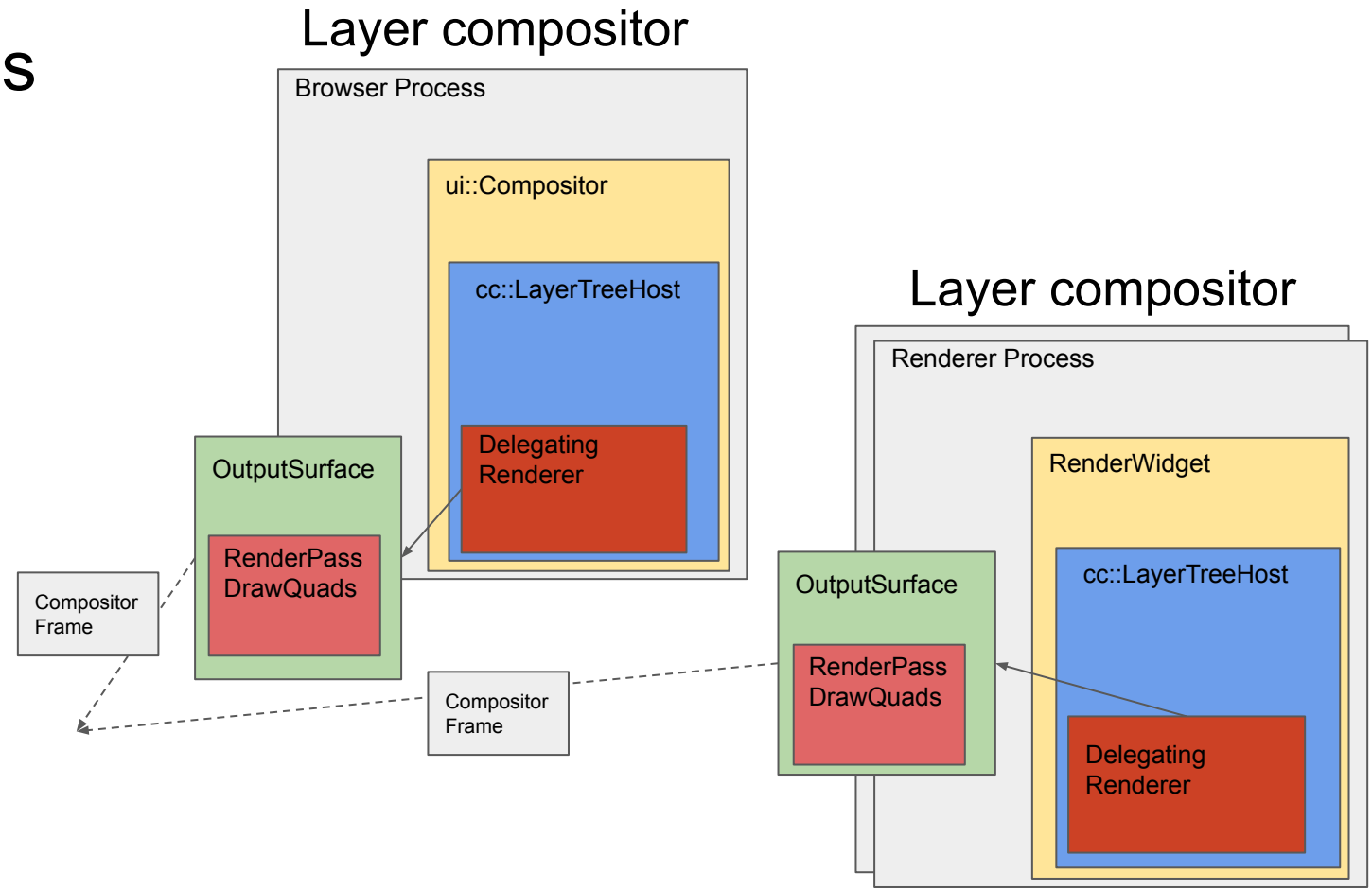
# 2. Ubercompositor (Direct vs Delegated rendering)



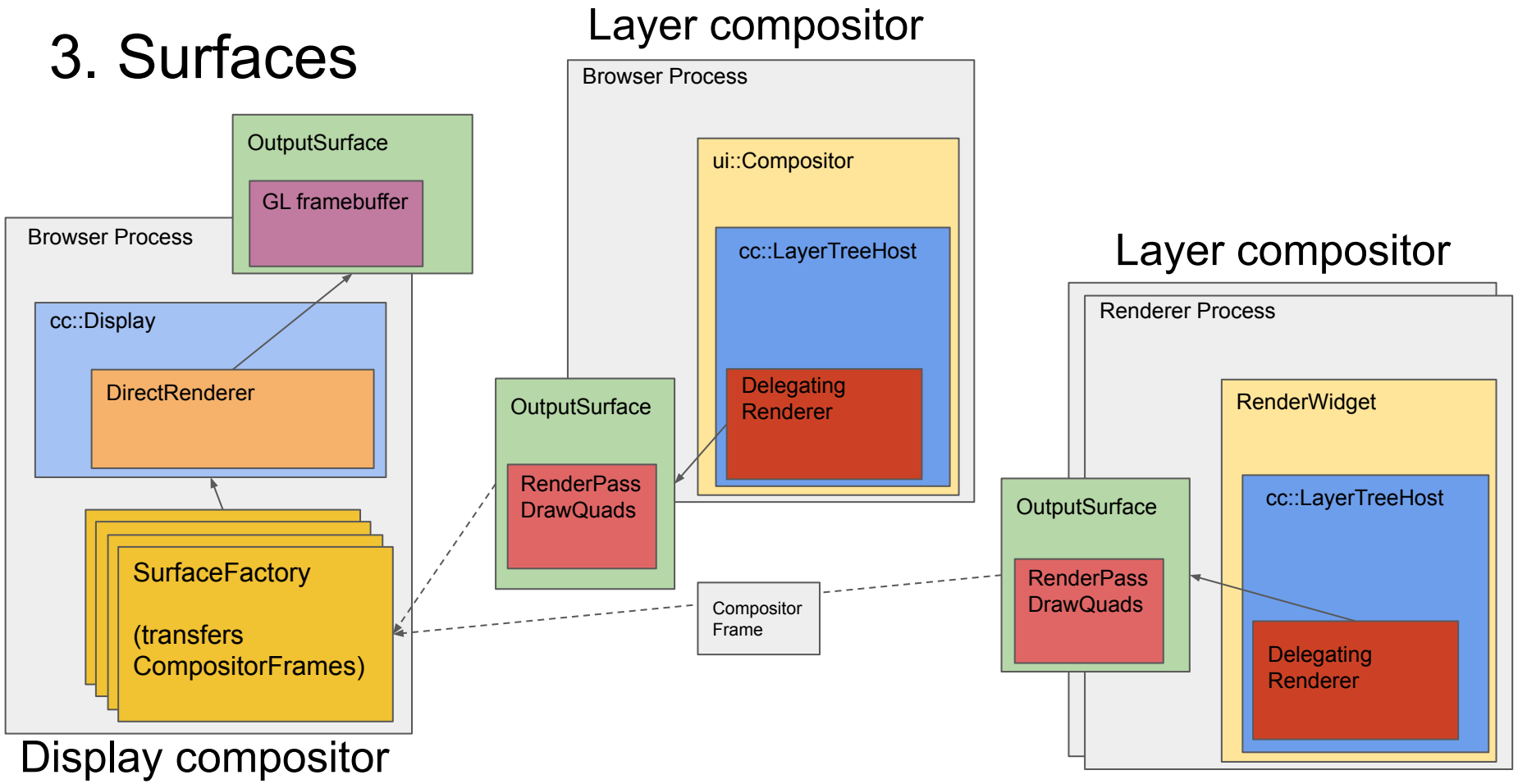
# 2. Ubercompositor



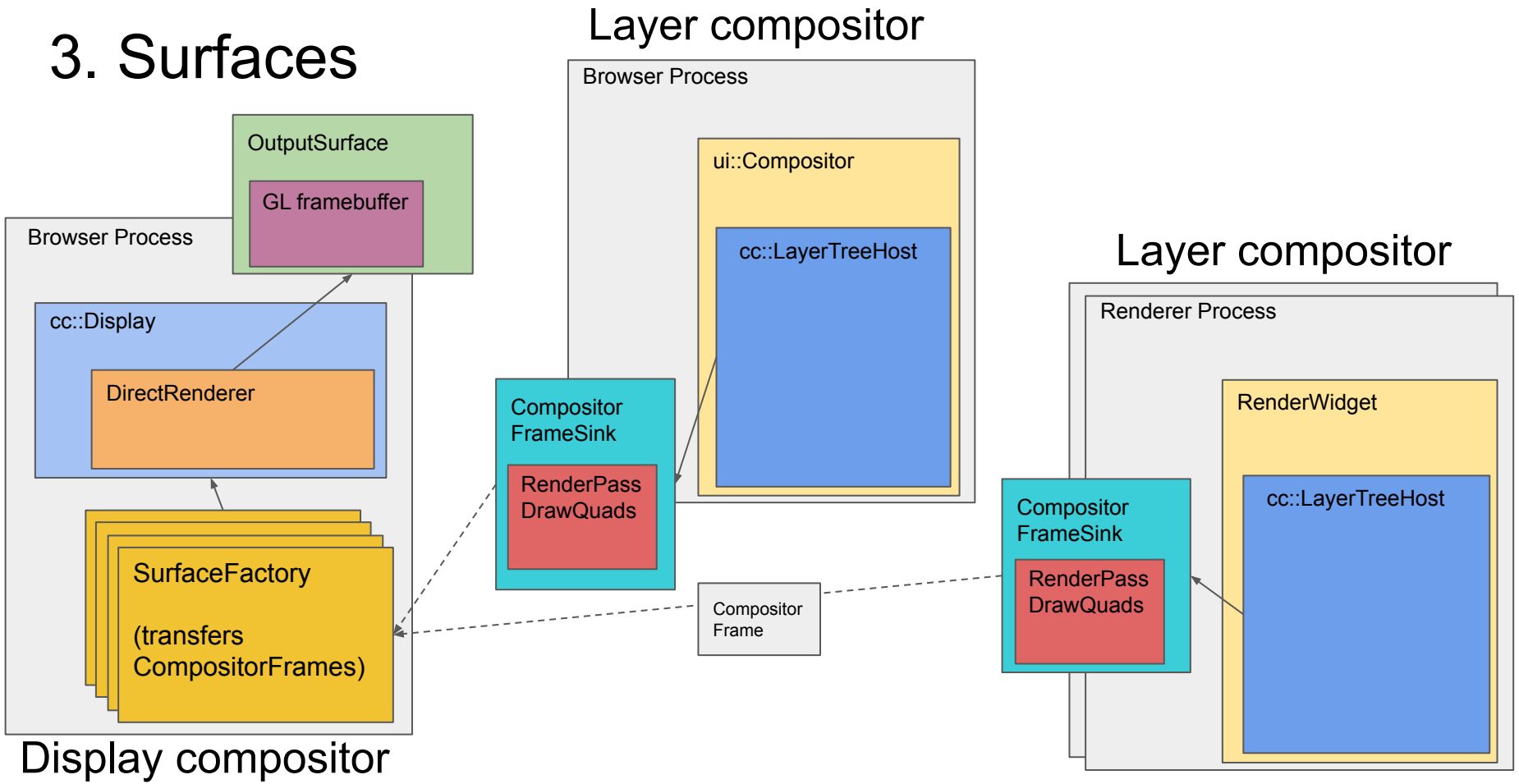
# 3. Surfaces


Layer compositor
Browser Process
ui::Compositor
cc::LayerTreeHost
Delegating Renderer
OutputSurface
RenderPass DrawQuads
Compositor Frame
Compositor Frame
Layer compositor
Renderer Process
RenderWidget
cc::LayerTreeHost
Delegating Renderer
OutputSurface
RenderPass DrawQuads

# 3. Surfaces


Layer compositor
Browser Process
ui::Compositor
cc::LayerTreeHost
Delegating Renderer
OutputSurface
RenderPass DrawQuads
Layer compositor
Renderer Process
RenderWidget
cc::LayerTreeHost
Delegating Renderer
OutputSurface
RenderPass DrawQuads
Compositor Frame
OutputSurface
GL framebuffer
Browser Process
cc::Display
DirectRenderer
SurfaceFactory
(transfers CompositorFrames)
Display compositor

# 3. Surfaces


Layer compositor
Browser Process
ui::Compositor
cc::LayerTreeHost
OutputSurface
GL framebuffer
Browser Process
cc::Display
DirectRenderer
Compositor
FrameSink
RenderPass
DrawQuads
SurfaceFactory
(transfers
CompositorFrames)
Compositor
Frame
Layer compositor
Renderer Process
RenderWidget
cc::LayerTreeHost
Compositor
FrameSink
RenderPass
DrawQuads
Display compositor

# OutputSurface API

Managing the backbuffer, swapping it, and using overlays.

```
class OutputSurface {
  virtual void EnsureBackbuffer() = 0;
  virtual void DiscardBackbuffer() = 0;

  virtual bool IsDisplayedAsOverlayPlane() const = 0;

  virtual void Reshape(const gfx::Size& size,
                       float device_scale_factor,
                       const gfx::ColorSpace& color_space,
                       bool has_alpha) = 0;

  virtual void SwapBuffers(OutputSurfaceFrame frame) = 0;
};
```

```
class OutputSurfaceClient {
  virtual void DidReceiveSwapBuffersAck() = 0;
  virtual void DidLoseOutputSurface() = 0;
};
```

# CompositorFrameSink API

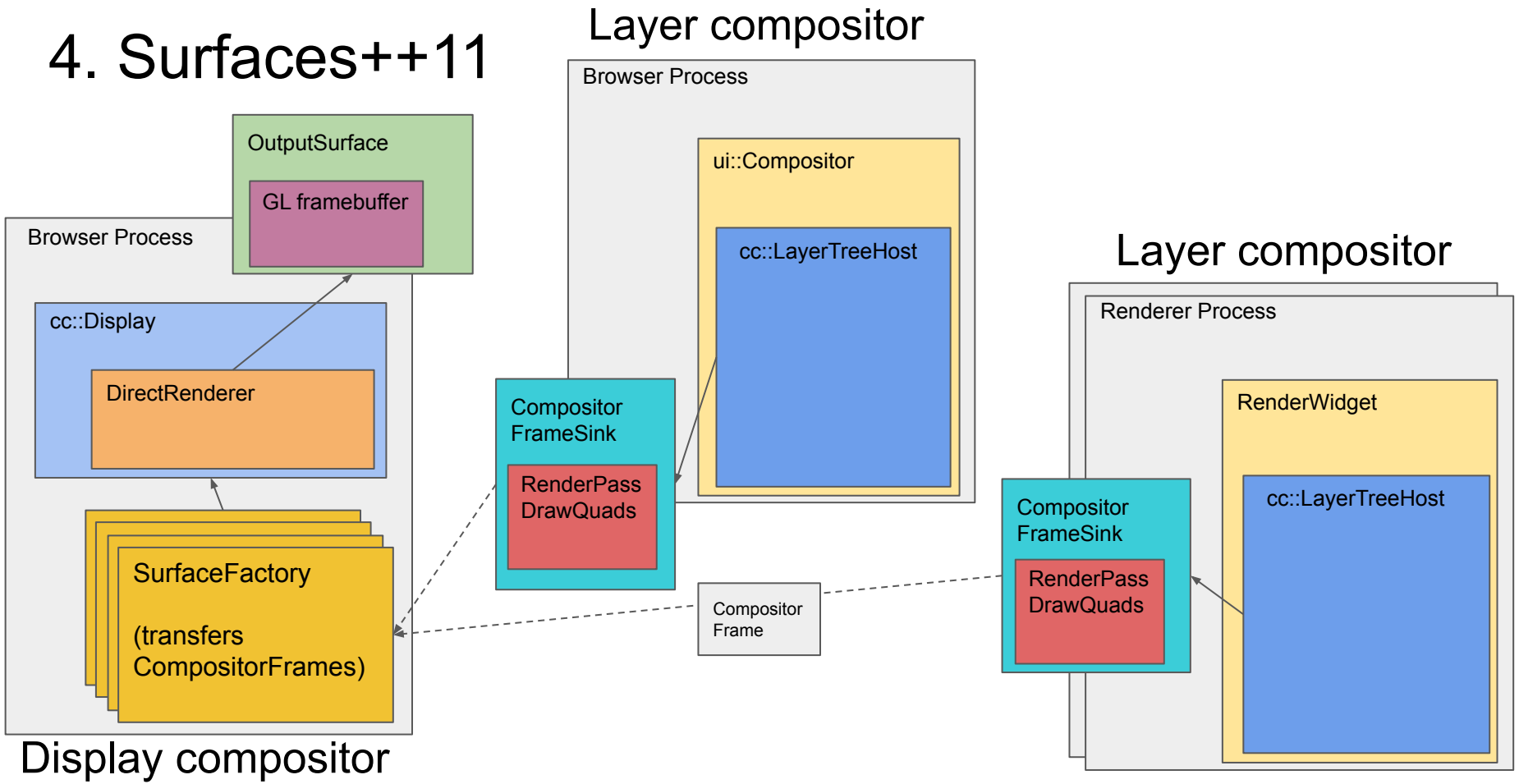
Managing the backbuffer, swapping it, and using overlays.

```
class CompositorFrameSink {
 virtual void SubmitCompositorFrame(
     CompositorFrame frame) = 0;
};
```

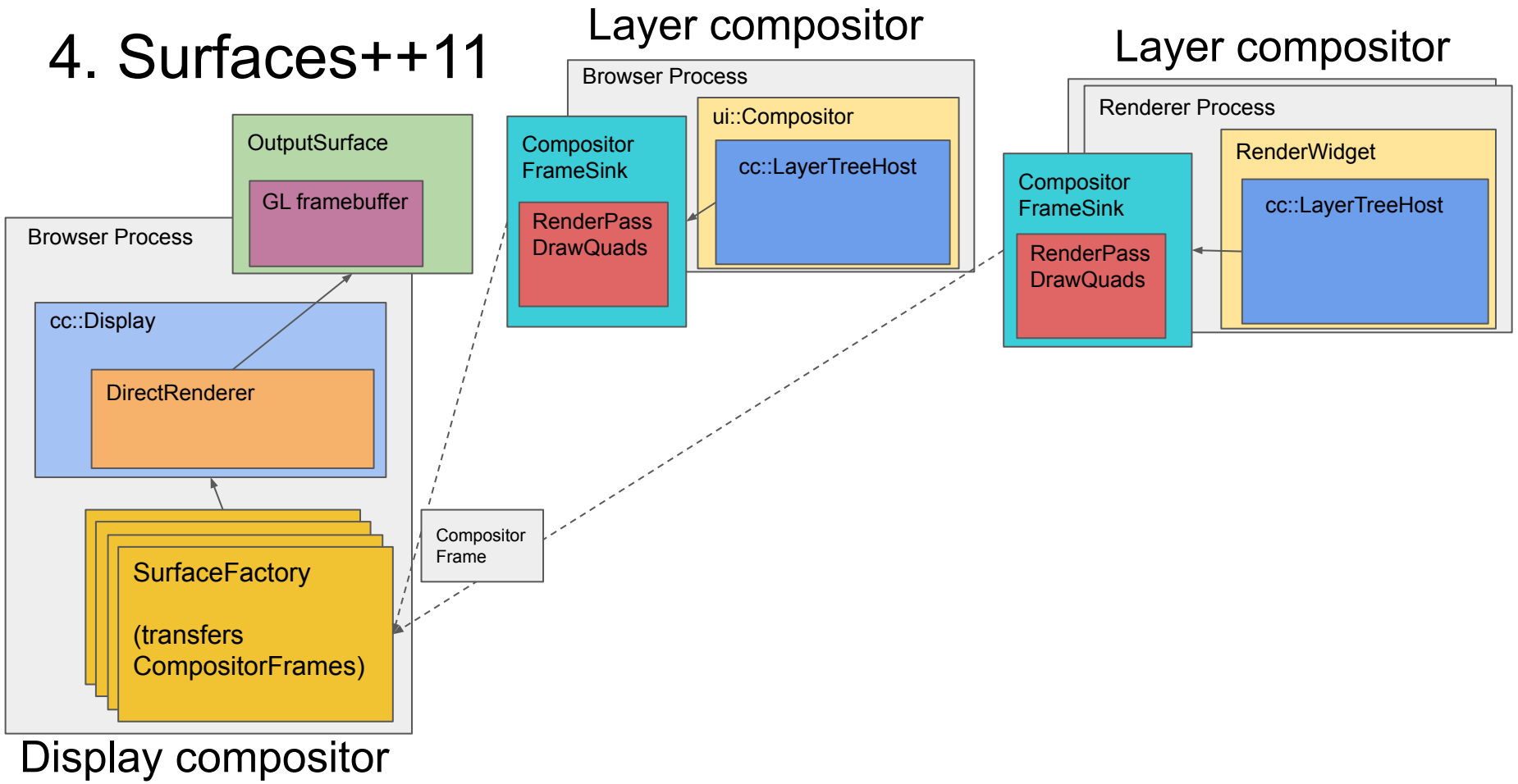
```
class CompositorFrameSinkClient {
  virtual void DidReceiveCompositorFrameAck() = 0;
  virtual void DidLoseCompositorFrameAck() = 0;

  // Layer compositor-specific stuff.
  virtual void SetTreeActivationCallback(
      const base::Closure& callback) = 0;
};
```

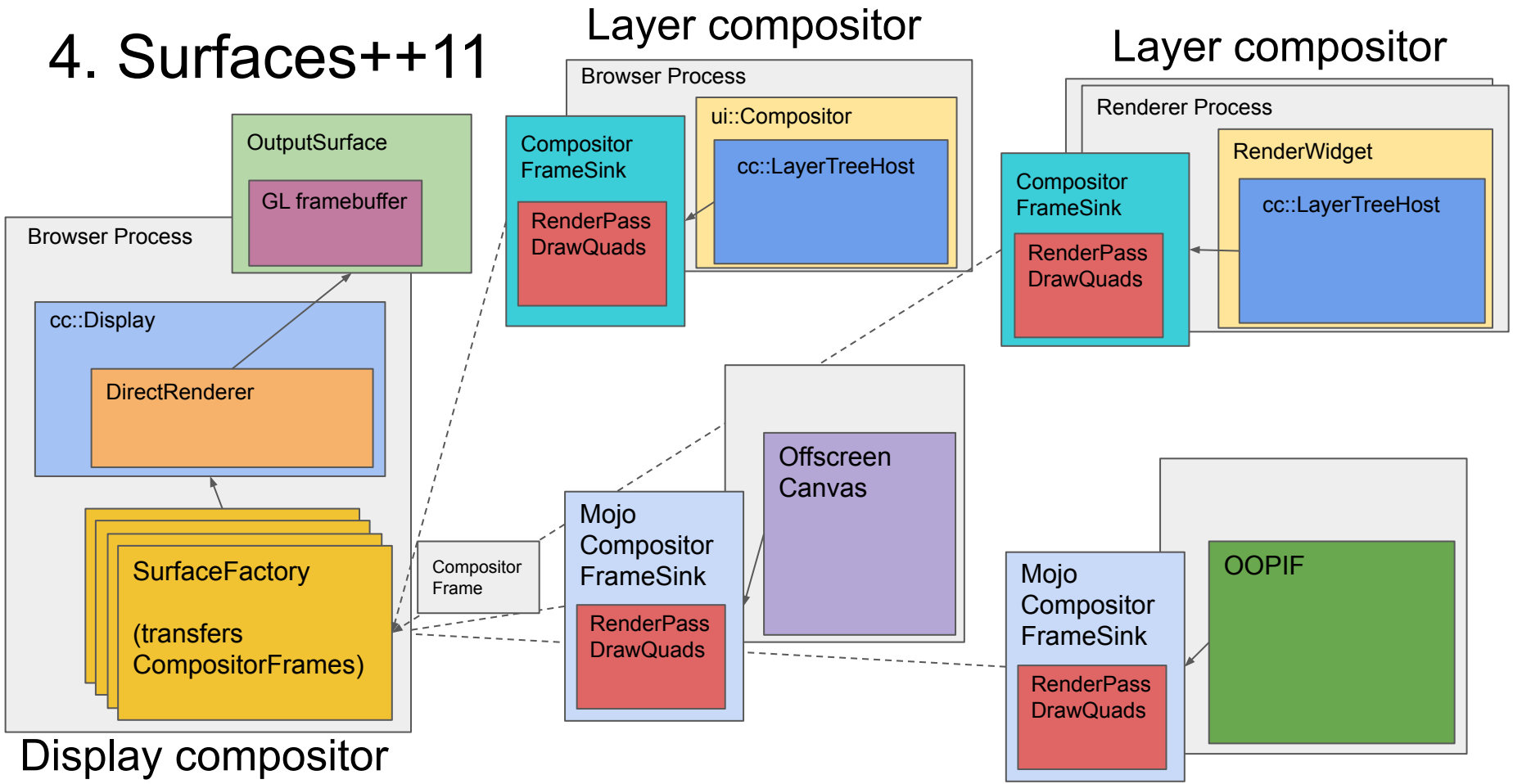
# 4. Surfaces++11



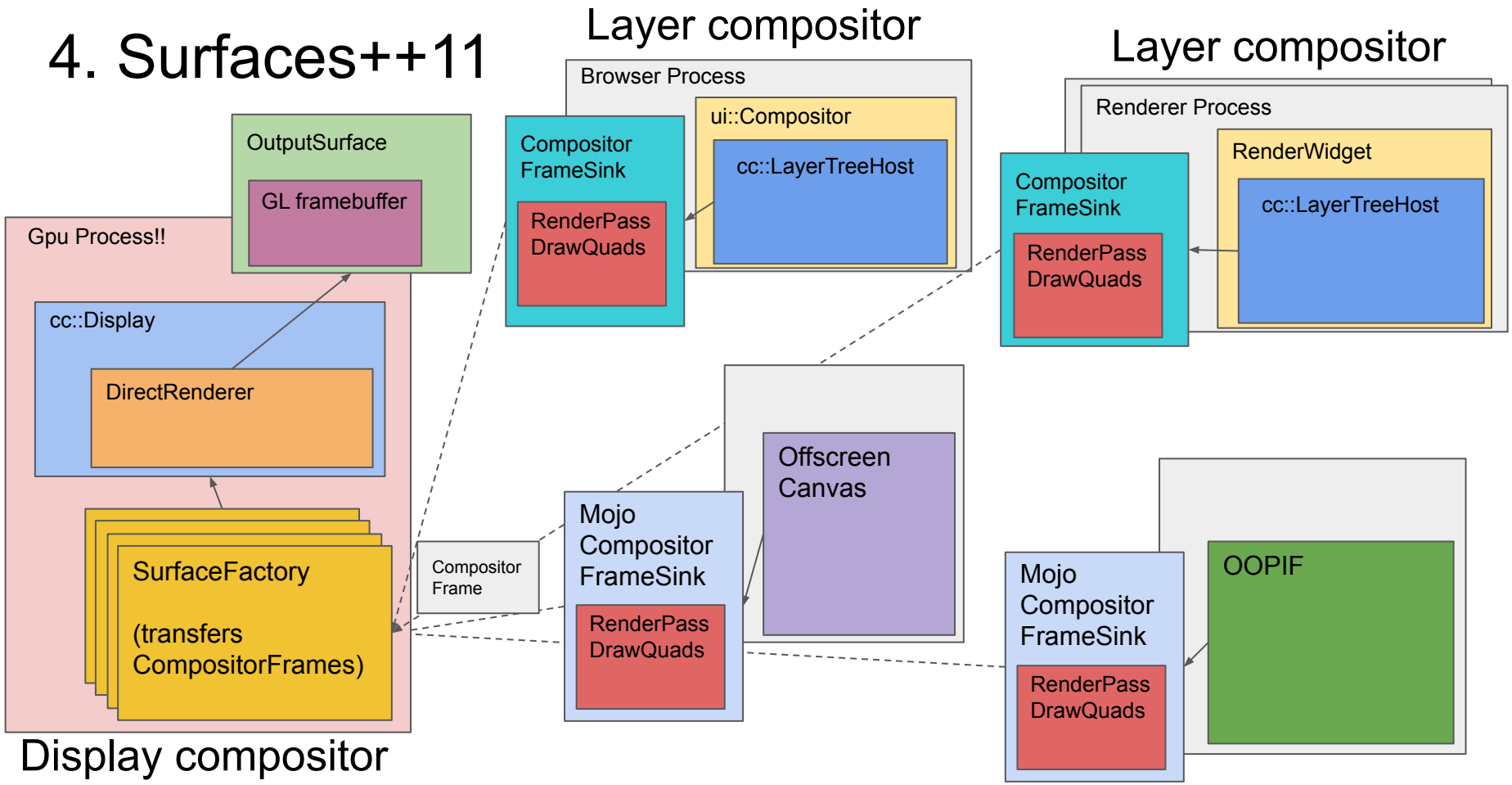
# 4. Surfaces++11



# 4. Surfaces++11



# 4. Surfaces++11



# Why Display compositor in the Gpu process?

- Vulkan API is very low level, more tightly couples the Vulkan client code to the driver.
- Requires in-process GPU access which is scary for security, want to do it from the sandboxed GPU process.

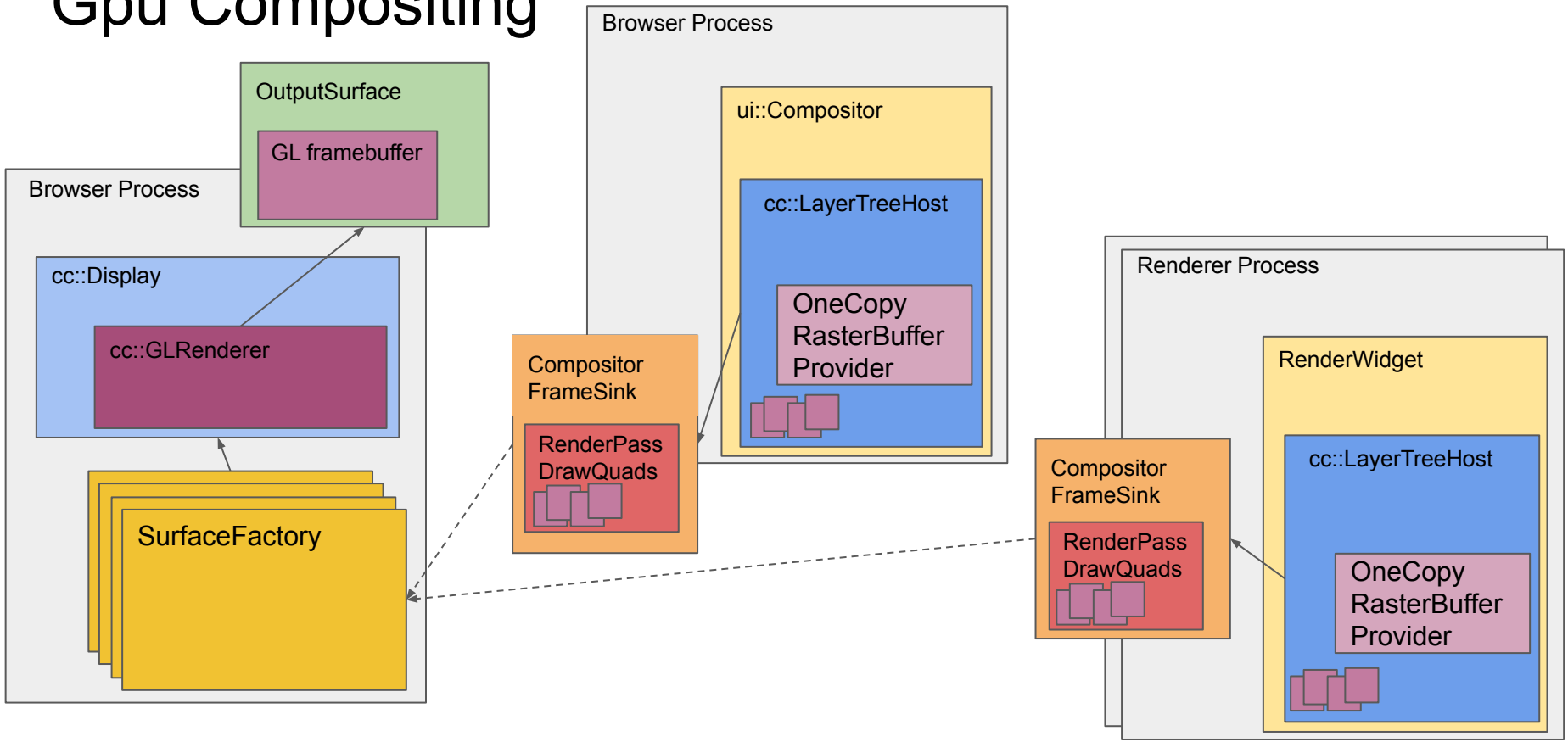
- Remove the command buffer from the Display compositor’s GL contexts (strictly overhead).
- Better scheduling possibilities with frames that take a long time to draw when we can see the actual draw time, not the command buffer time.

# Gpu Compositing vs Software Compositing

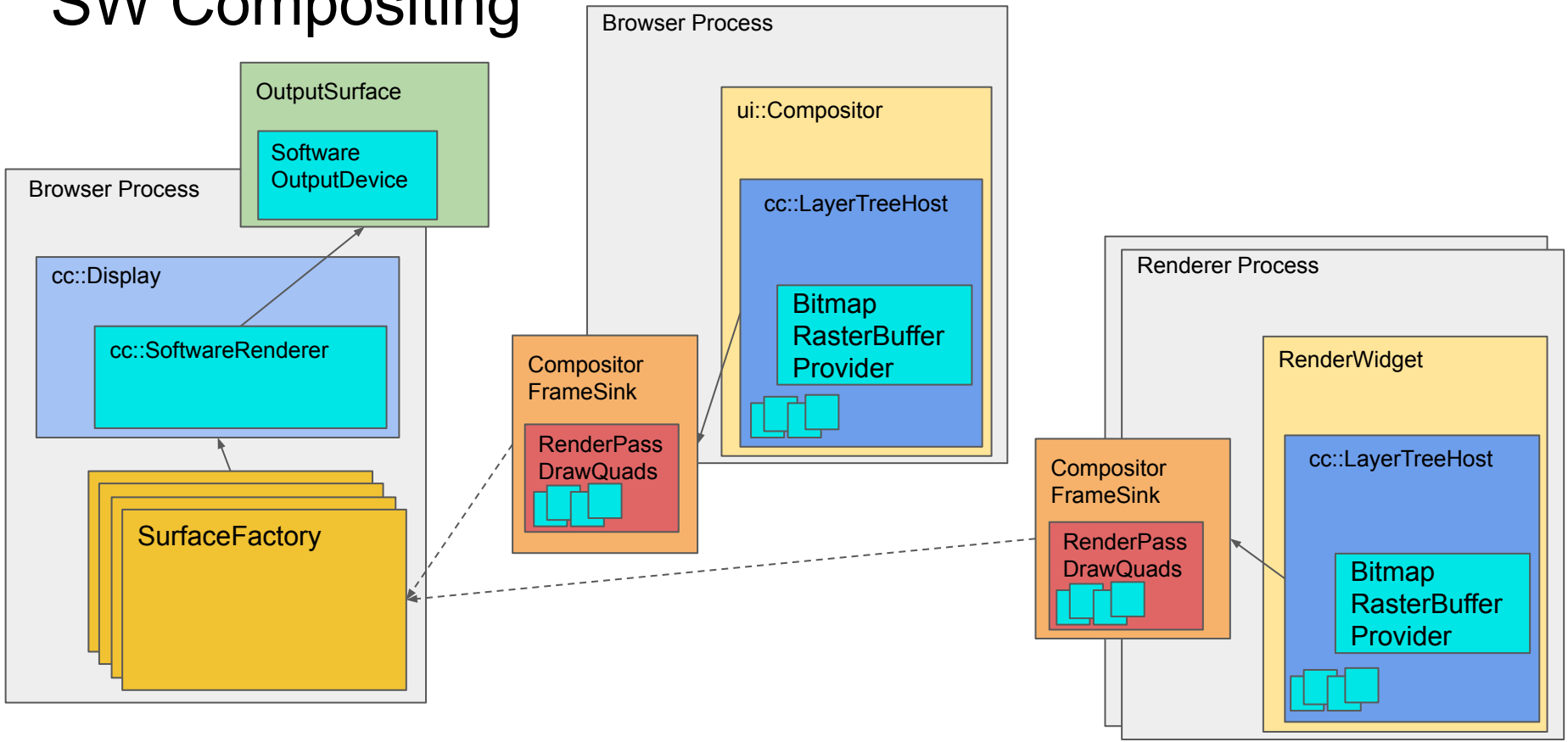
- What changes in the compositing stack between Gpu vs Software?

1. cc::CompositorFrameSink and cc::OutputSurface don’t come with a GL context (ie null ContextProvider)
2. Results in rastering display lists into bitmaps instead of textures.
3. CompositorFrame’s DrawQuads point to shared memory bitmaps instead of textures.
4. Display compositor uses cc::SoftwareRenderer instead of cc::GLRenderer to compose all of the DrawQuads.

# Gpu Compositing



# SW Compositing



# The End

I hope this was eye opening.

Here’s a beautiful cat.

**Questions?**