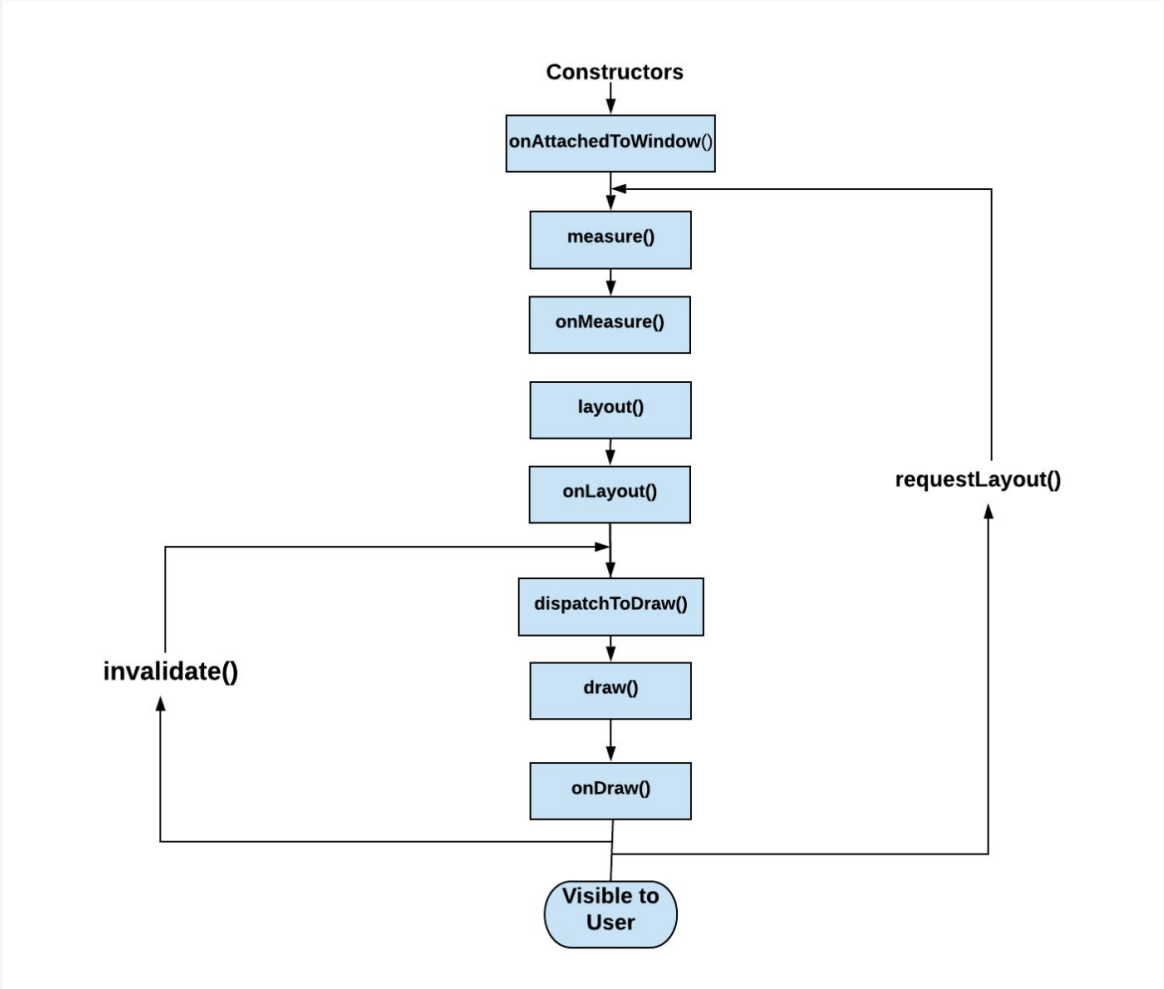
Draw : How Android Renders Things

- Anupam Singh
- Sr.Android Dev, Urban Company
- Co host- androidiots podcast

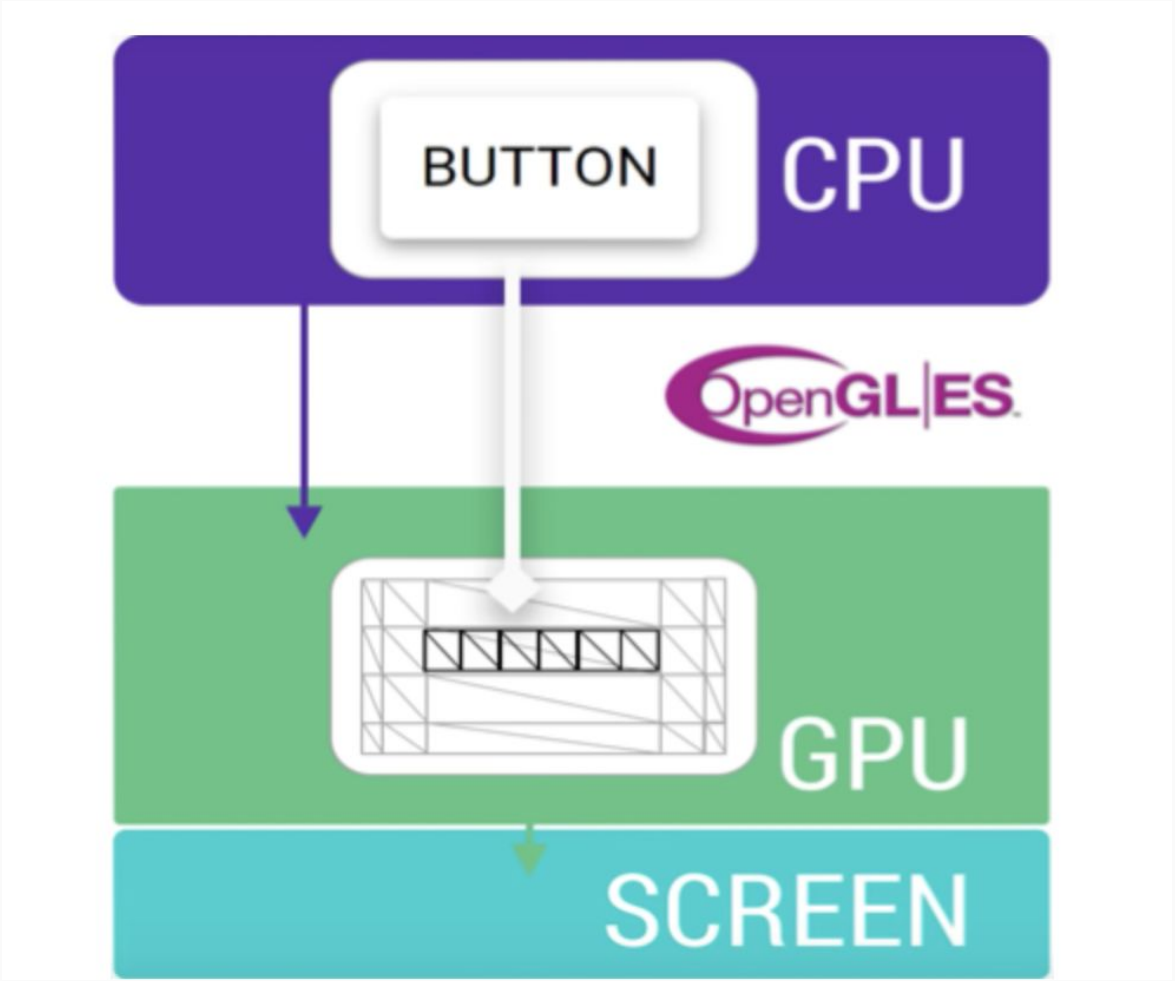
Things we know

The View Life cycle



Things we know

Rasterization



Things we dont know(Maybe)

- Choreographer
- Vsync
- HWC

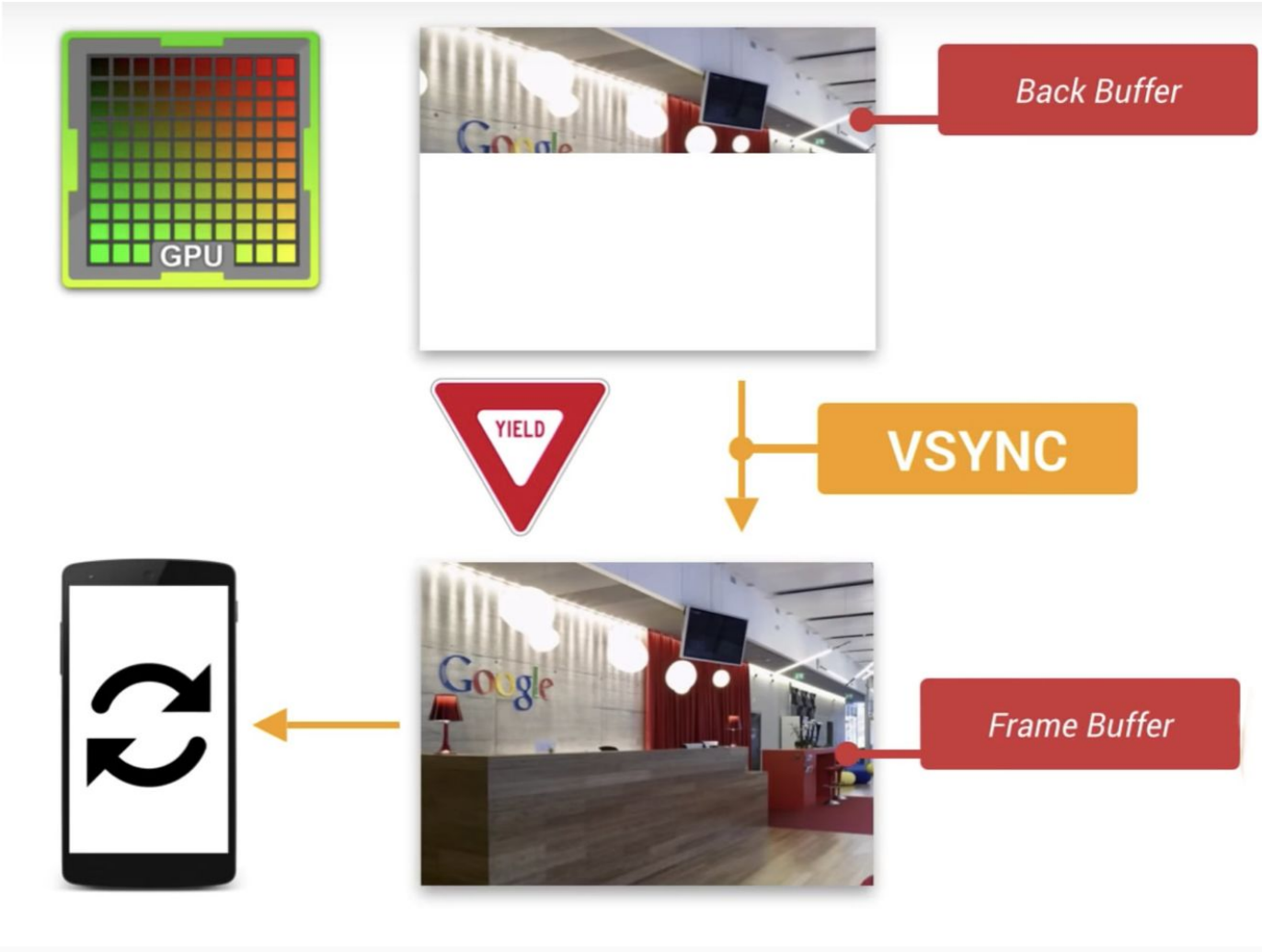
- Project Butter in Android 4.1
- VSYNC is the event posted periodically by the kernal at fixed interval where the input handling, Animation and Window drawing happening synchroniiously at the same

- The VSYNC signal synchronizes the display pipeline. The display pipeline consists of app rendering, SurfaceFlinger composition, and the Hardware Composer (HWC) presenting images on the display. VSYNC synchronizes the time apps wake up to start rendering, the time SurfaceFlinger wakes up to composite the screen, and the display refresh cycle. This synchronization eliminates stutter and improves the visual performance of graphics.

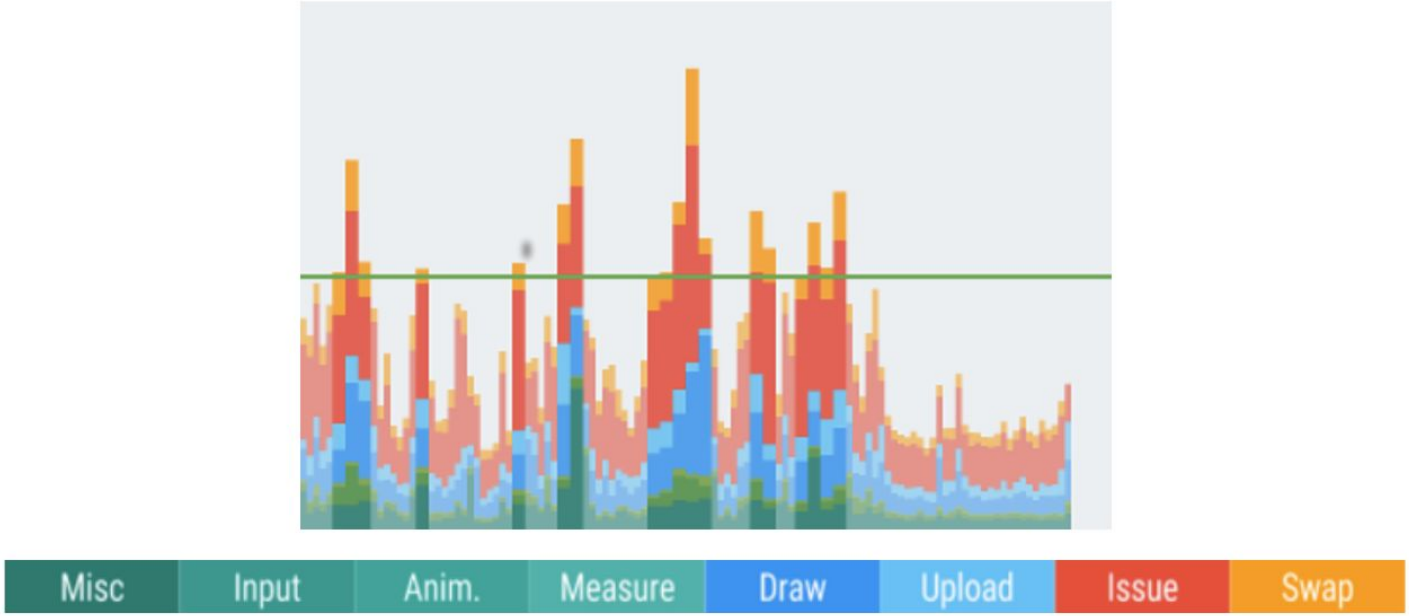
- *Choreographer* is the main component which registers application main thread to display system and coordinate between the application view drawing component to the display system for synchronizing VSYNC with the application event, *animation* and draw handling.

- When ever *ViewRootImpl* got request from View hierarchy to refresh or update or invalidate then it request *Choreographer* for refresh by registering a callback. When *Choreographer* got any request then it request for next VSYNC event from the lower layer and when it got he VSYNC event then it asks *ViewRootImpl* by calling the registered Callback to handle the drawing accordingly. Also when ever *ViewRootImpl* wants to redraw the view then this thing will repeat.


Whatever
I don't care anymore
I'm done.


GPU
Back Buffer
YIELD
VSYNC
Google
Frame Buffer

What are these colors?

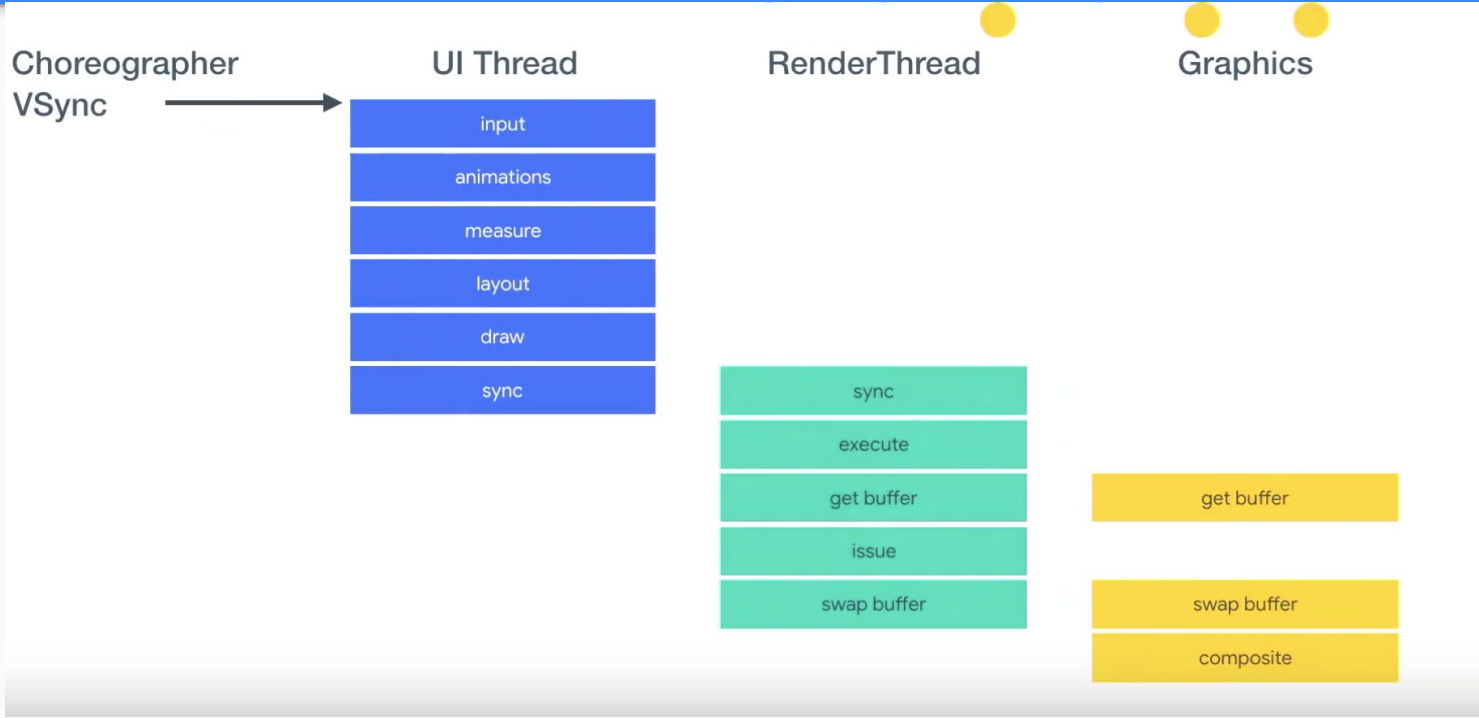


Misc : Work between consecutive frames
Input: Processing user input
Anim : Custom Animators

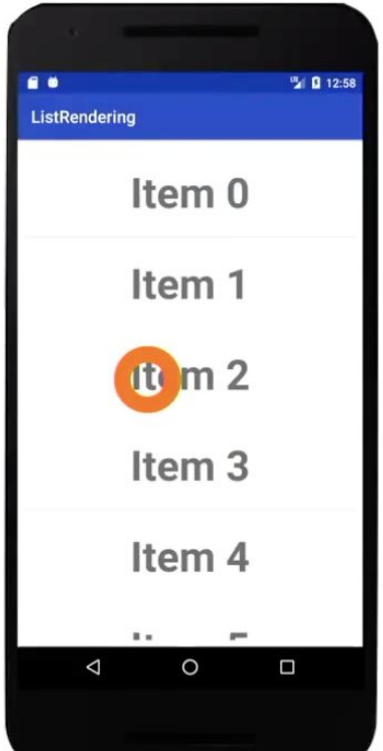
Measure : On Layout and On Measure Callback
Draw : Create and update View Display list

Upload : bitmap info to gpu
Issue : Commands to OpenGl to draw by 2d Renderer
Swap : Cpu waiting for GPU

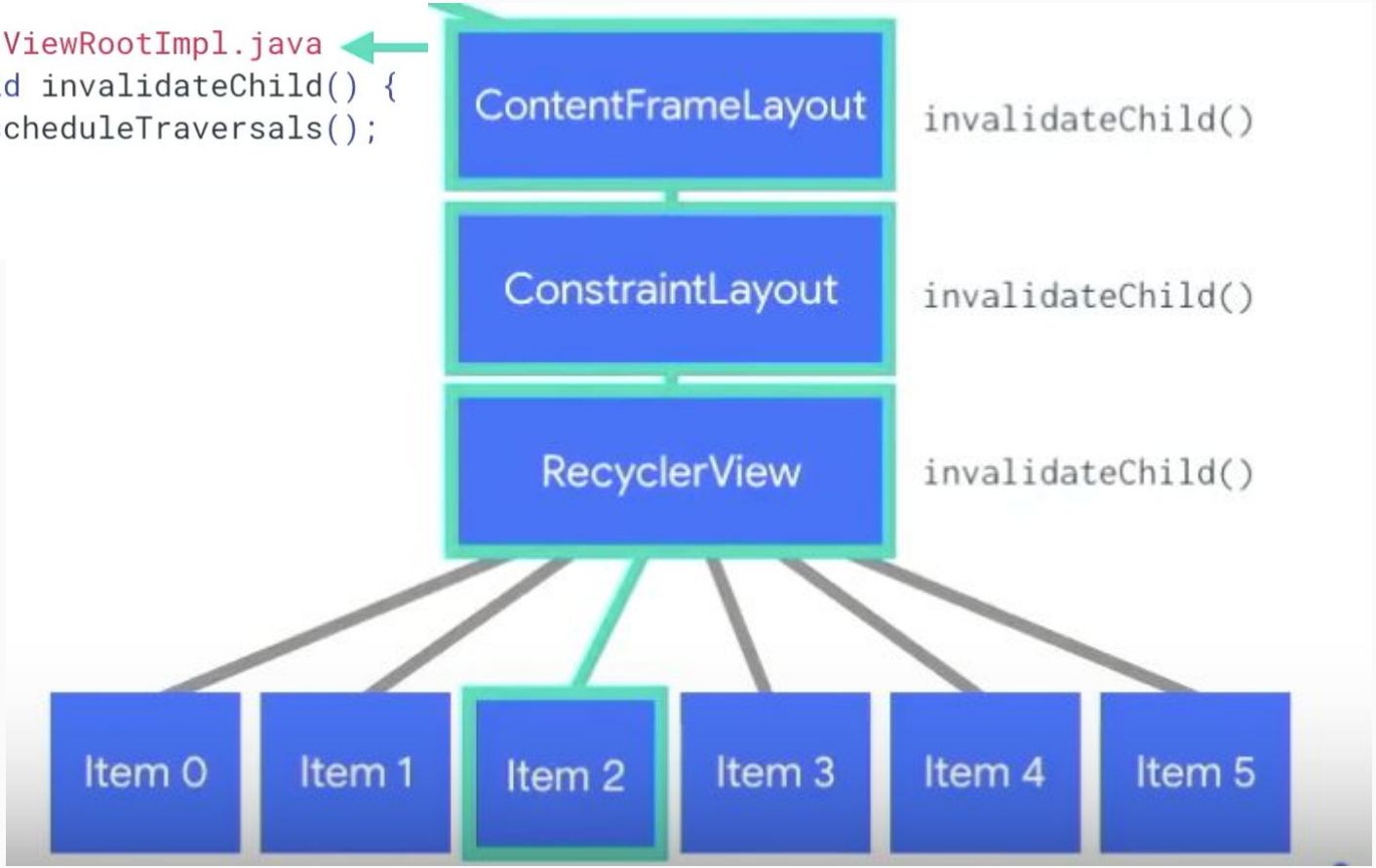
The Rendering pipe line



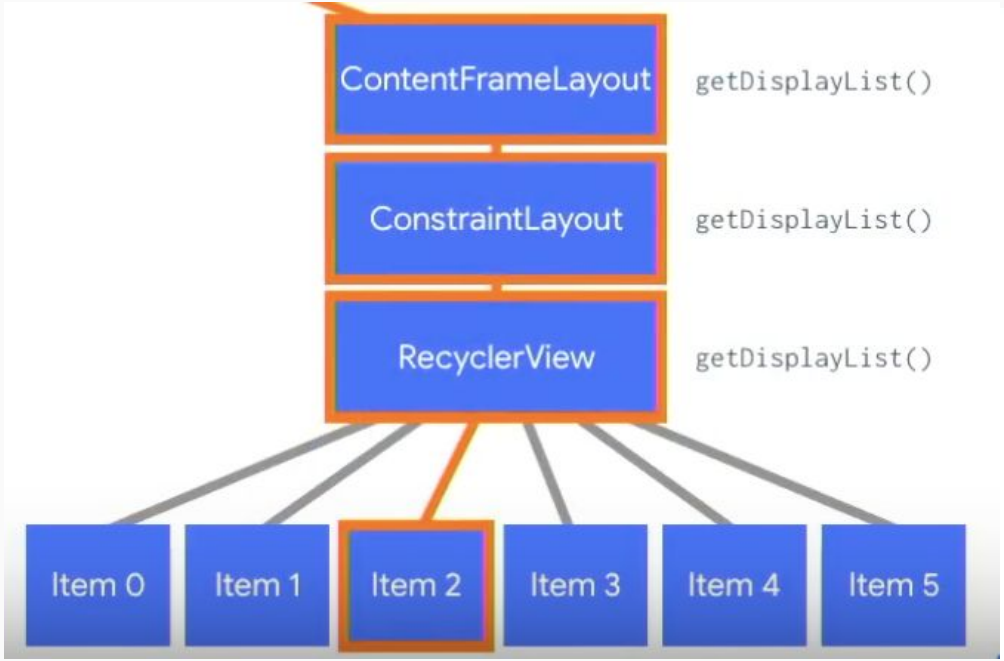

VSync
Input

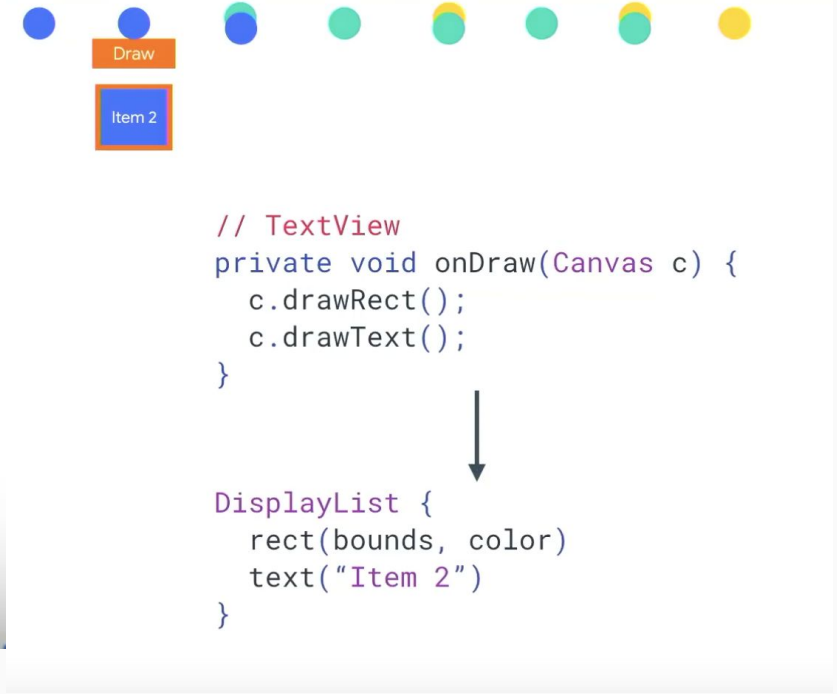

12:58
ListRendering
Item 0
Item 1
Item 2
Item 3
Item 4

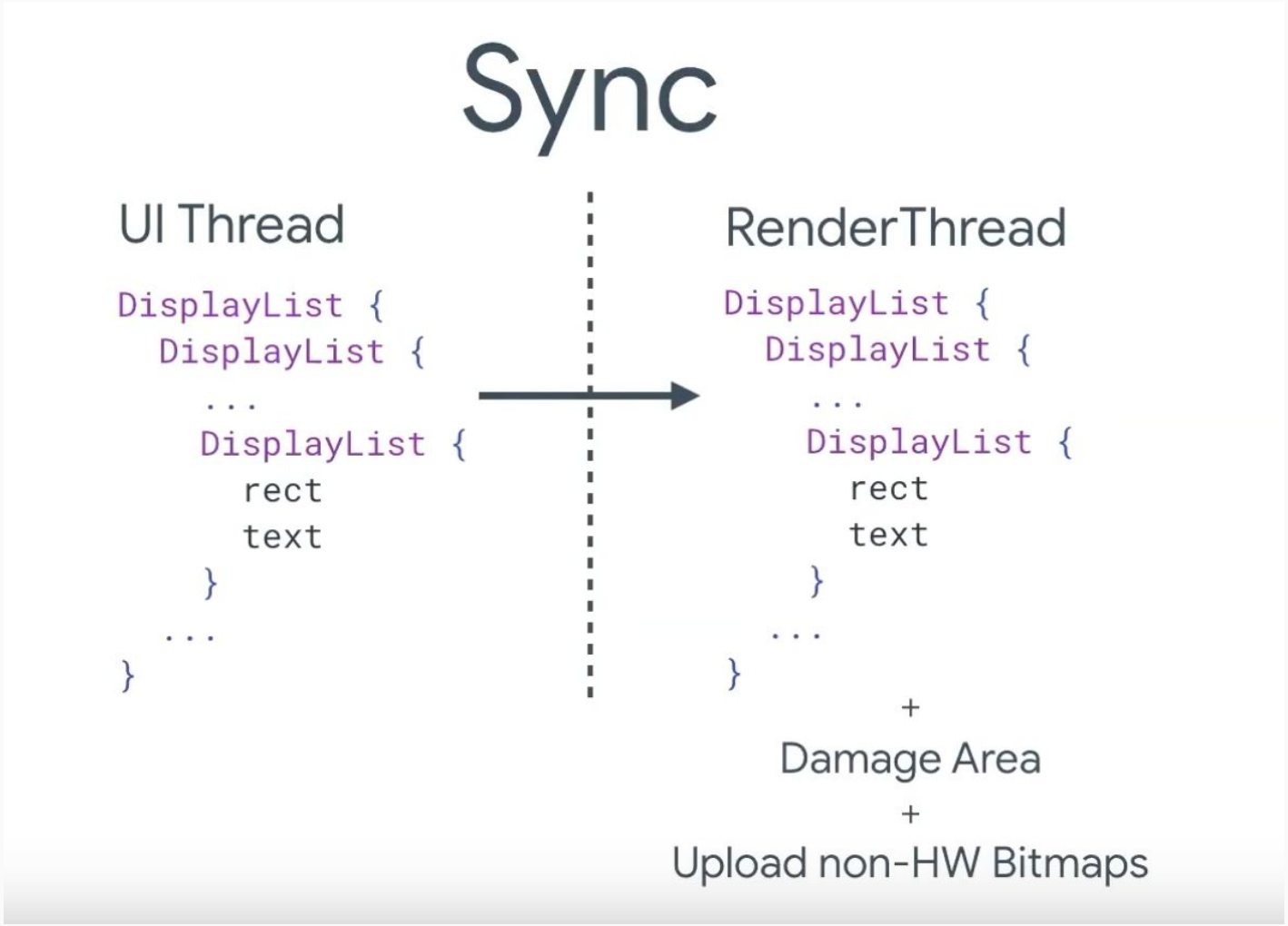
```
// ViewRootImpl.java
void invalidateChild() {
  scheduleTraversals();
}
```



- Traversal :
  - Measure, layout, Draw
- Display List : Encapsulation of rendering command of a view (draw background, draw rect)

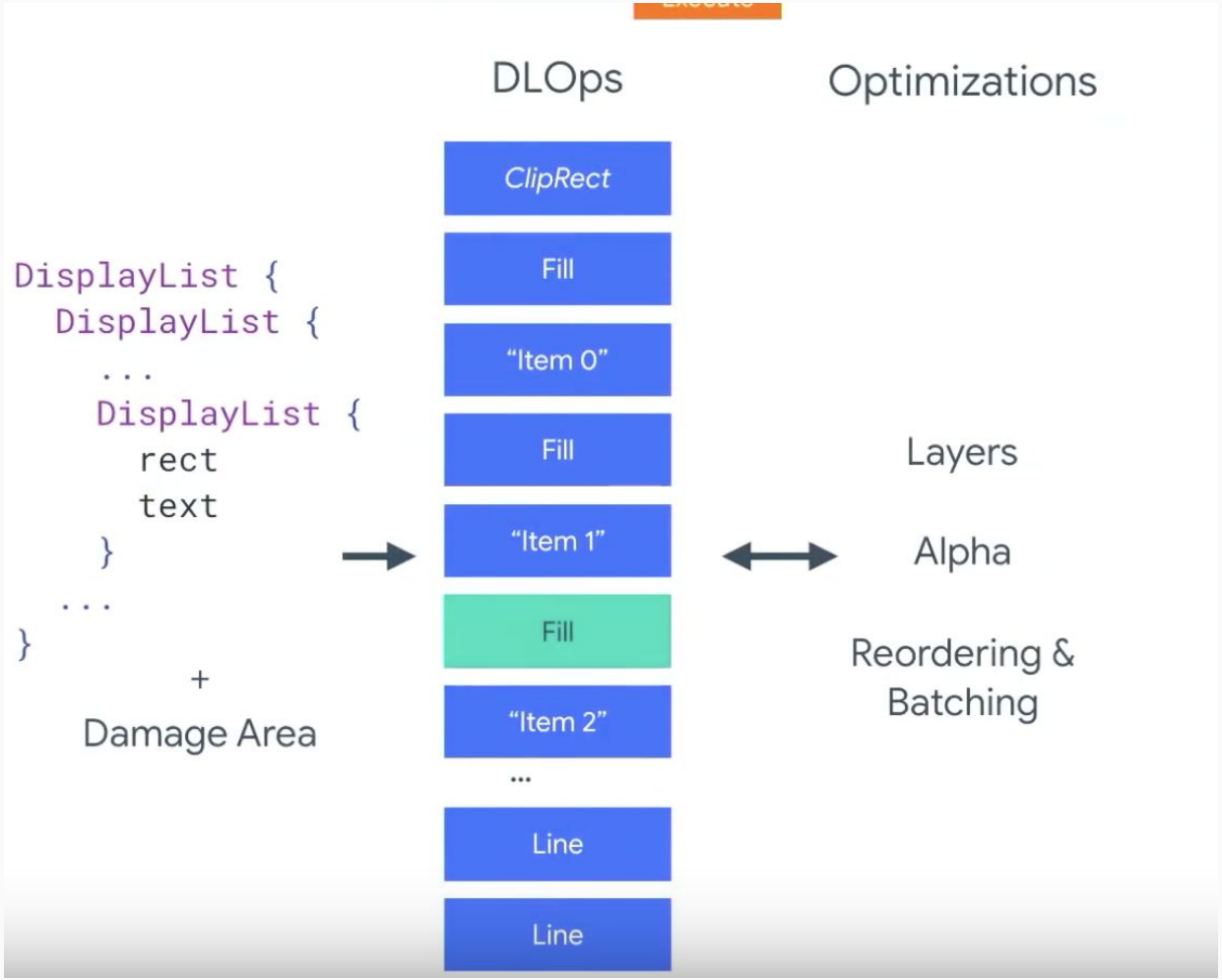


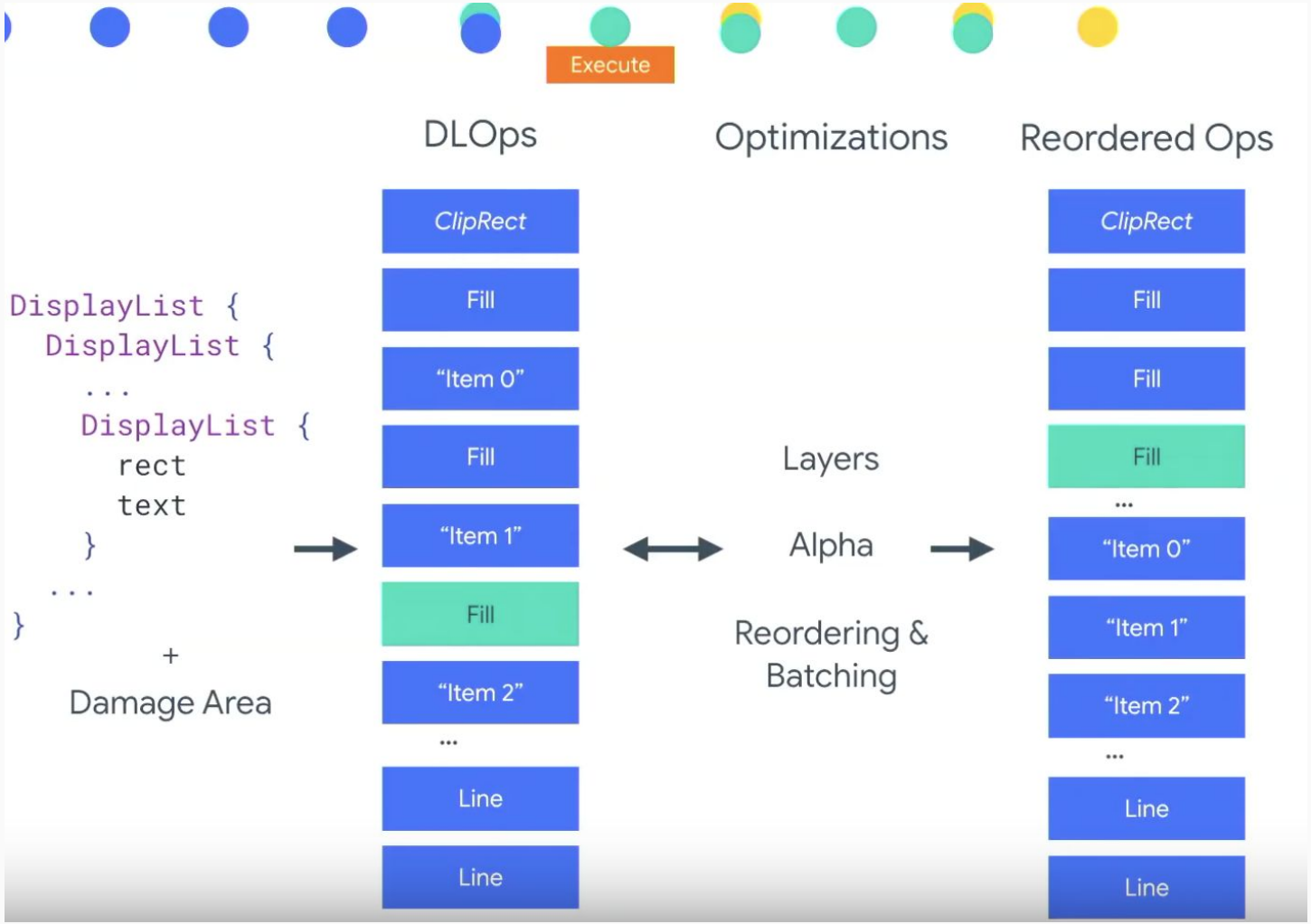


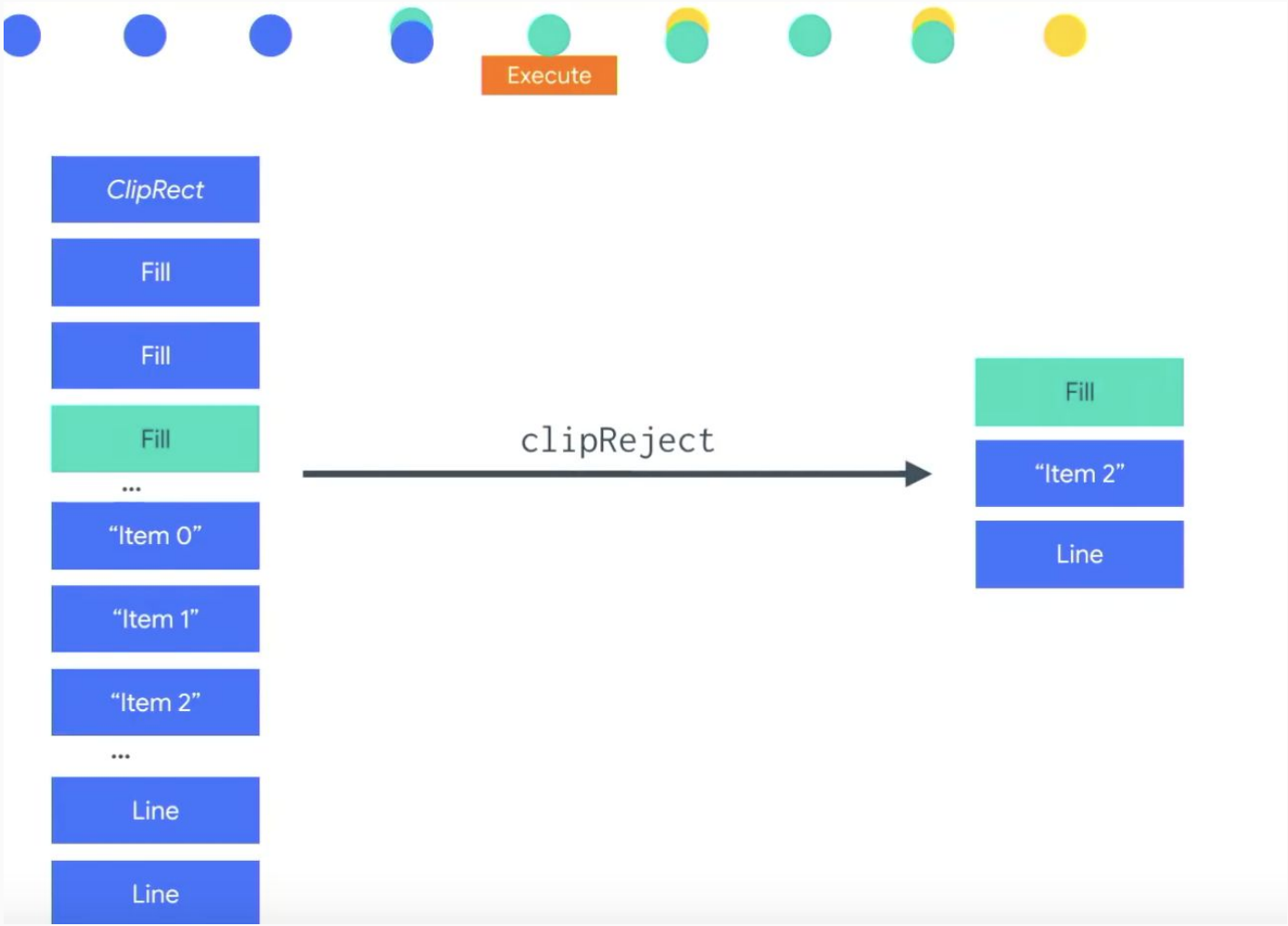


- HW bitmaps were added in android 0
- To optimize for expensive operation allocation a memory on java bitmap to be copied to gpu when its time to draw
- So if no updation in bitmap only on gpu use this

- Render thread :
  - Seperate thread which talks to gpu
  - Display list sync is serial but it can do things atomically , eg ripple animation without blocking UI thread
  - When idle can do some optimisation like idle prefetch for recycler view


DLOps
Optimizations
DisplayList {
DisplayList {
...
DisplayList {
rect
text
}
...
}
+
Damage Area
ClipRect
Fill
"Item 0"
Fill
"Item 1"
Fill
"Item 2"
...
Line
Line
Layers
Alpha
Reordering &
Batching


Execute
DLOps
Optimizations
Reordered Ops
DisplayList {
DisplayList {
...
DisplayList {
rect
text
}
...
}
+
Damage Area
ClipRect
Fill
"Item 0"
Fill
"Item 1"
Fill
"Item 2"
...
Line
Line
Layers
Alpha
Reordering &
Batching
ClipRect
Fill
Fill
Fill
...
"Item 0"
"Item 1"
"Item 2"
...
Line
Line

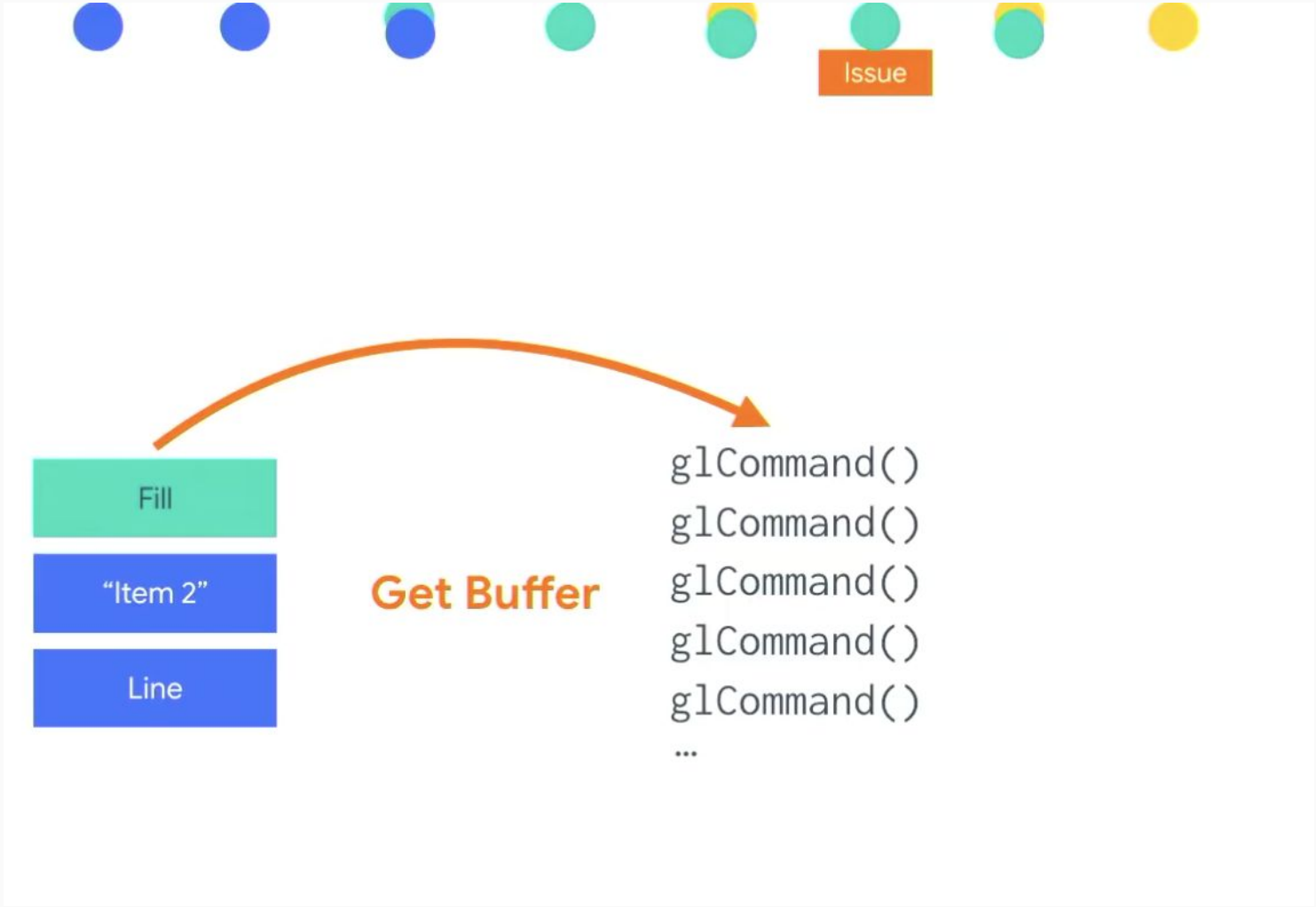

Execute
ClipRect
Fill
Fill
Fill
...
“Item 0”
“Item 1”
“Item 2”
...
Line
Line
clipReject
Fill
“Item 2”
Line

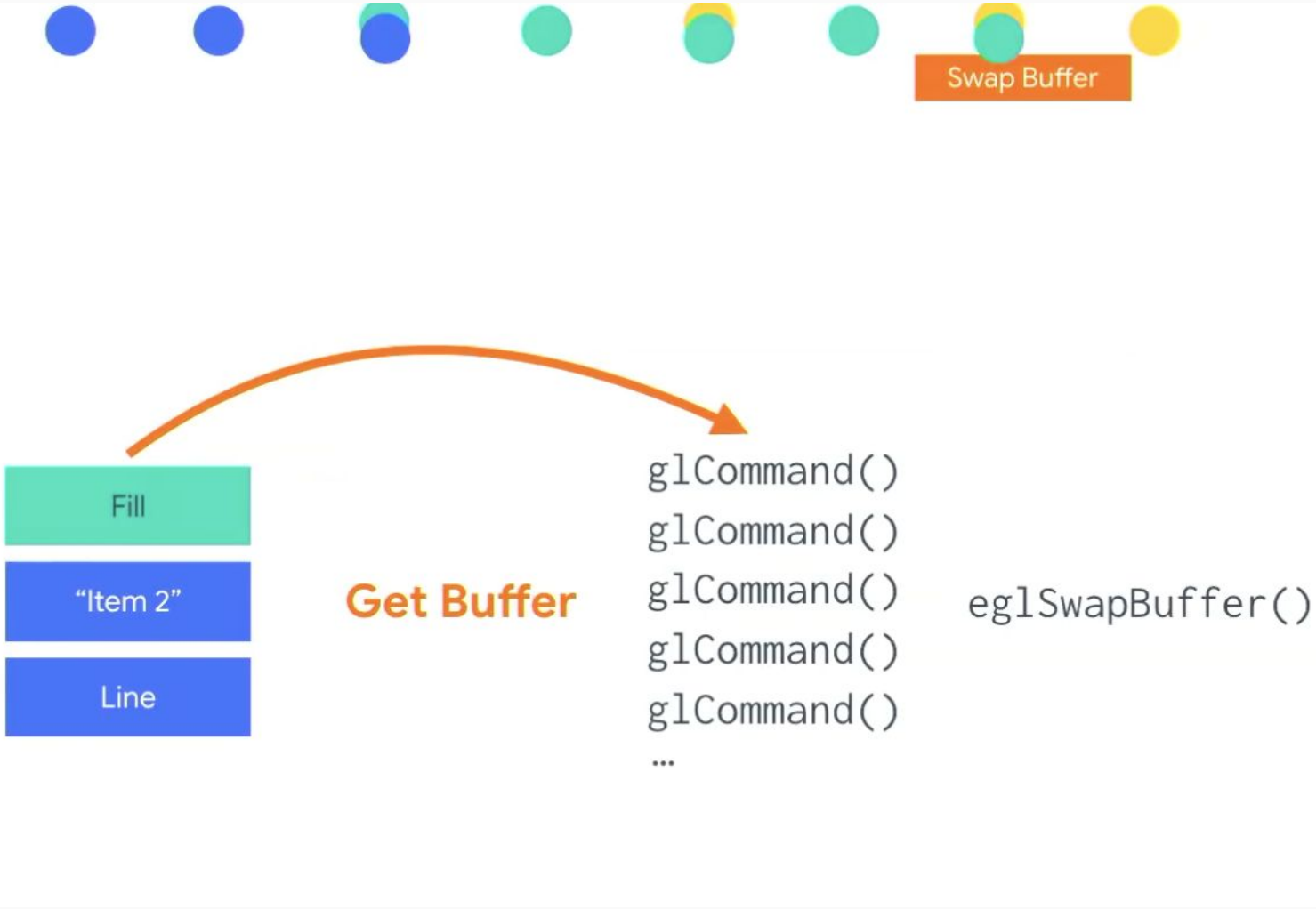
Fill

“Item 2”

Line

Get Buffer

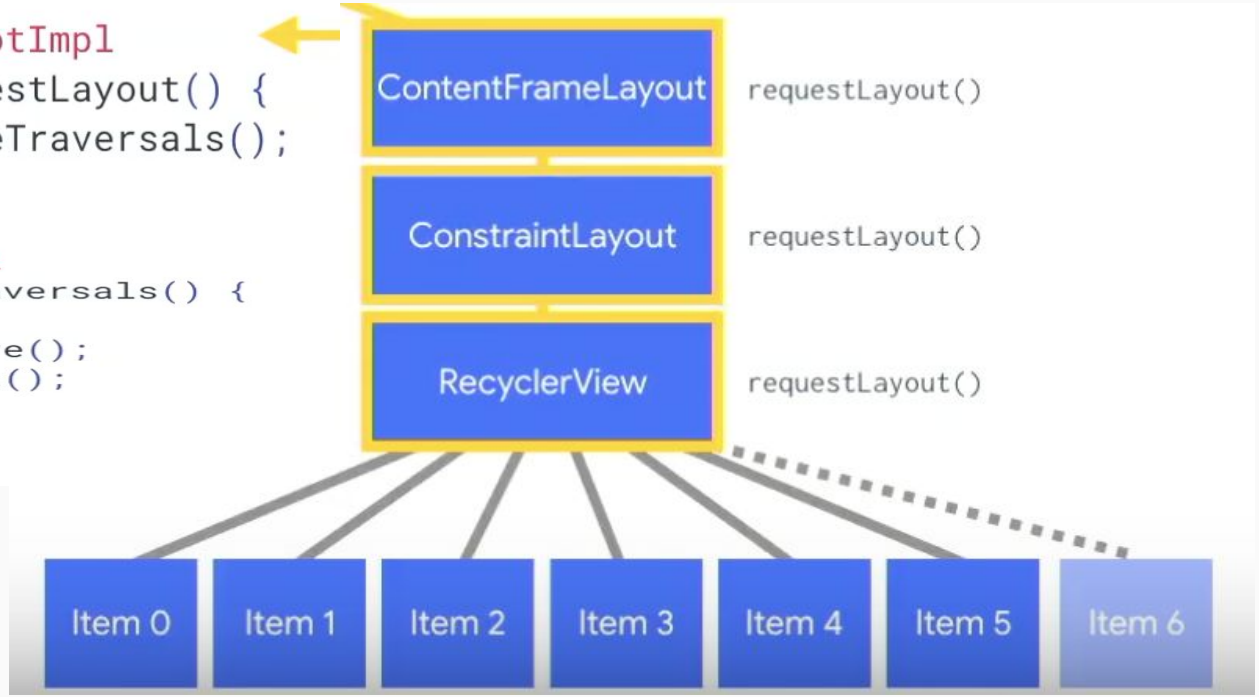

Issue
Fill
"Item 2"
Line
Get Buffer
glCommand()
glCommand()
glCommand()
glCommand()
glCommand()
...


Swap Buffer
Fill
“Item 2”
Line
Get Buffer
glCommand()
glCommand()
glCommand()
glCommand()
glCommand()
...
eglSwapBuffer()

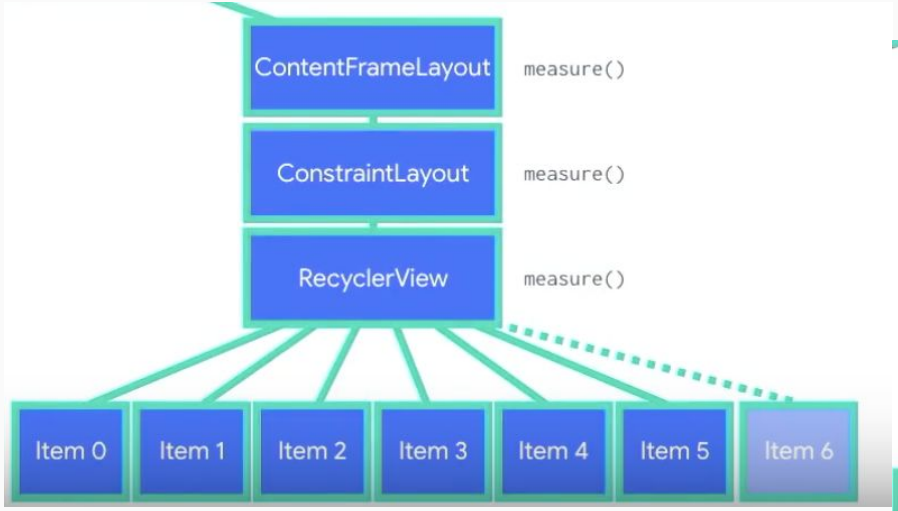
- Recycler View
- Display properties : Alpha, rotation : invalidateVp()
- Request Layout

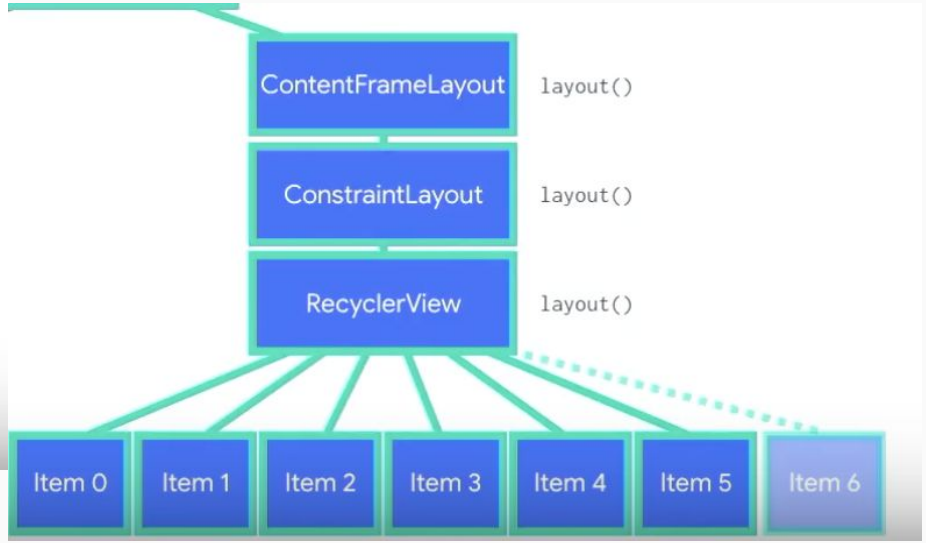
```
// ViewRootImpl
void requestLayout() {
  scheduleTraversals();
}
```

```
// ViewRootImpl
void performTraversals() {
  // ...
  performMeasure();
  performLayout();
}
```



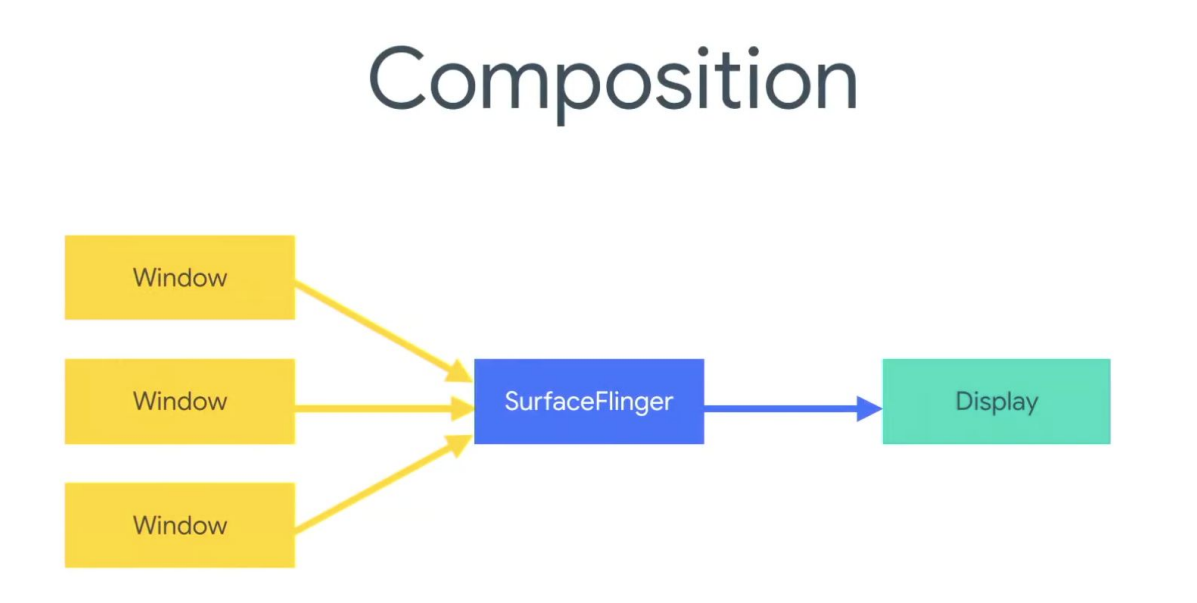
- Measure : view how big you like to be
- Layout : this is how big you are gonna be



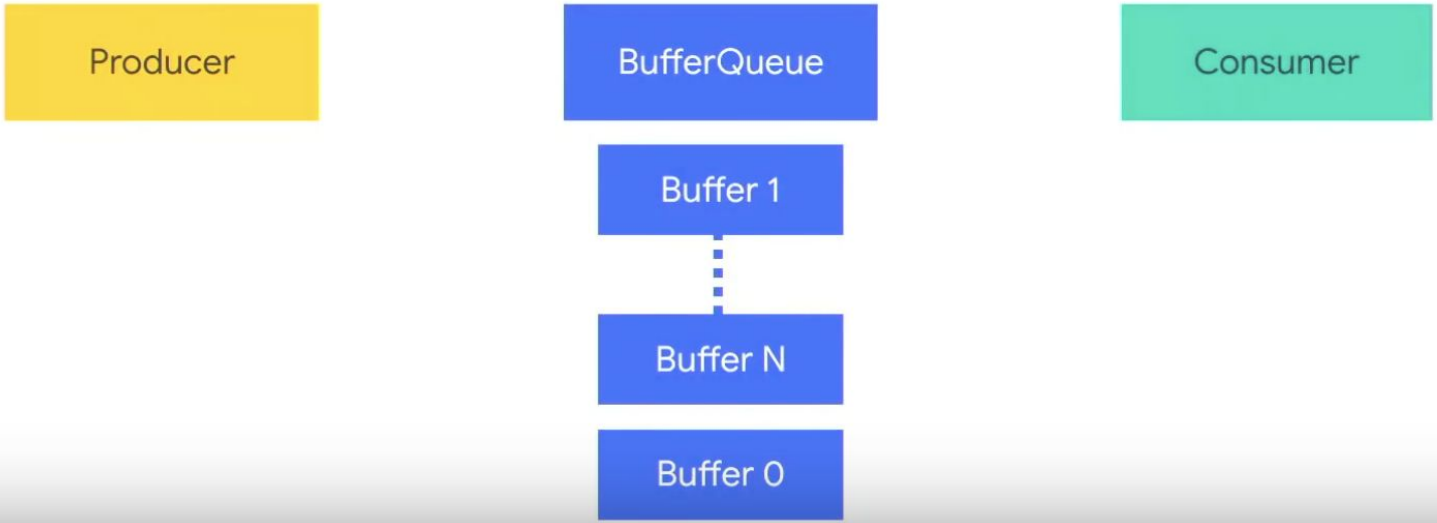


- After this the draw, sync, execute, getbuffer, issue, swap buffer, composition,

- Though recycler view is very smart, it knows about its parent and child, it can move views around and make space instead of calling request layout (Scrap Views)



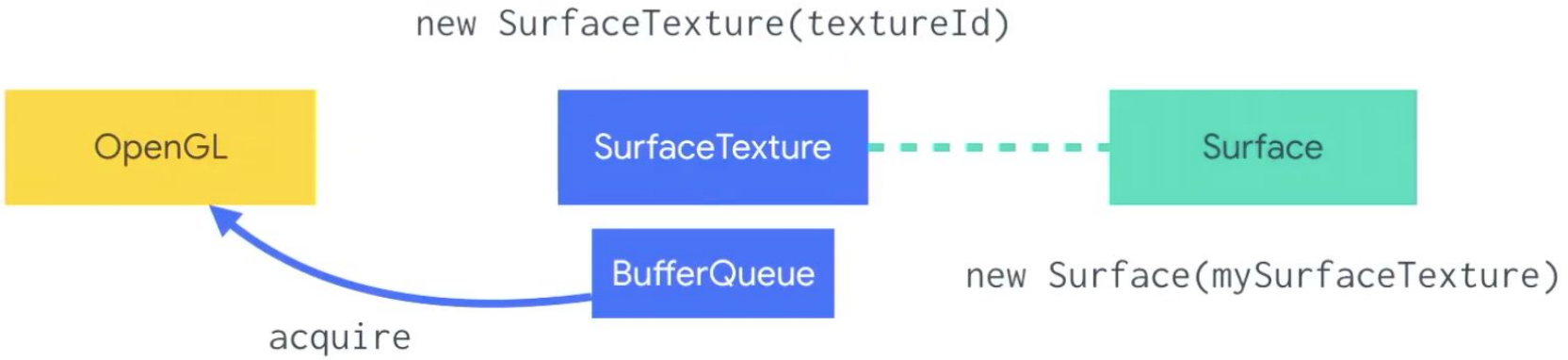
BufferQueue



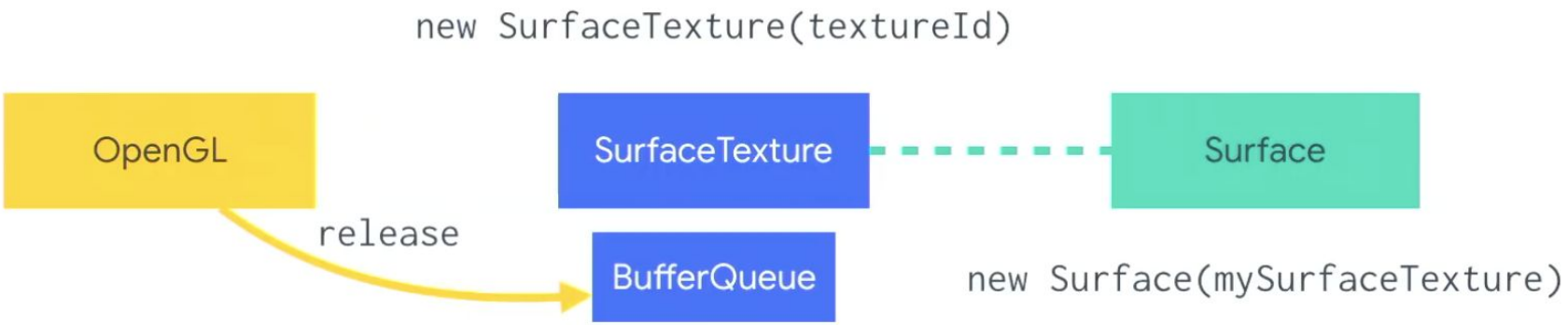
Creating a window

Creating a window

SurfaceTexture



SurfaceTexture



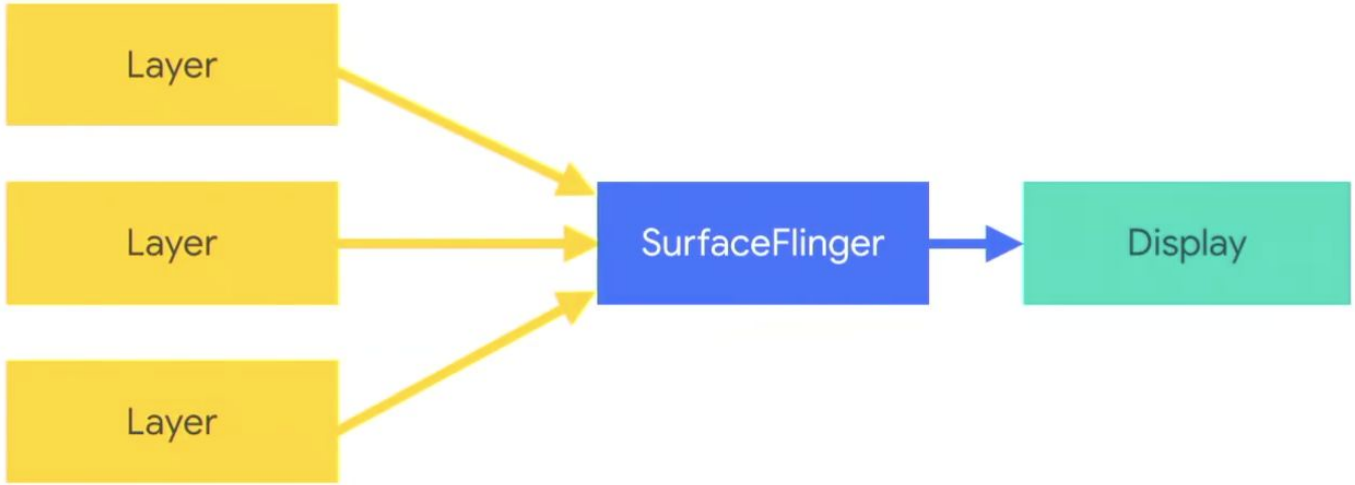
TextureView

- Creates a SurfaceTexture
- RenderThread is the consumer
  - Treats SurfaceTexture as a hardware layer under the hood
  - Kind of like a fancy ImageView
- Query the SurfaceTexture
  - To create the producer endpoint

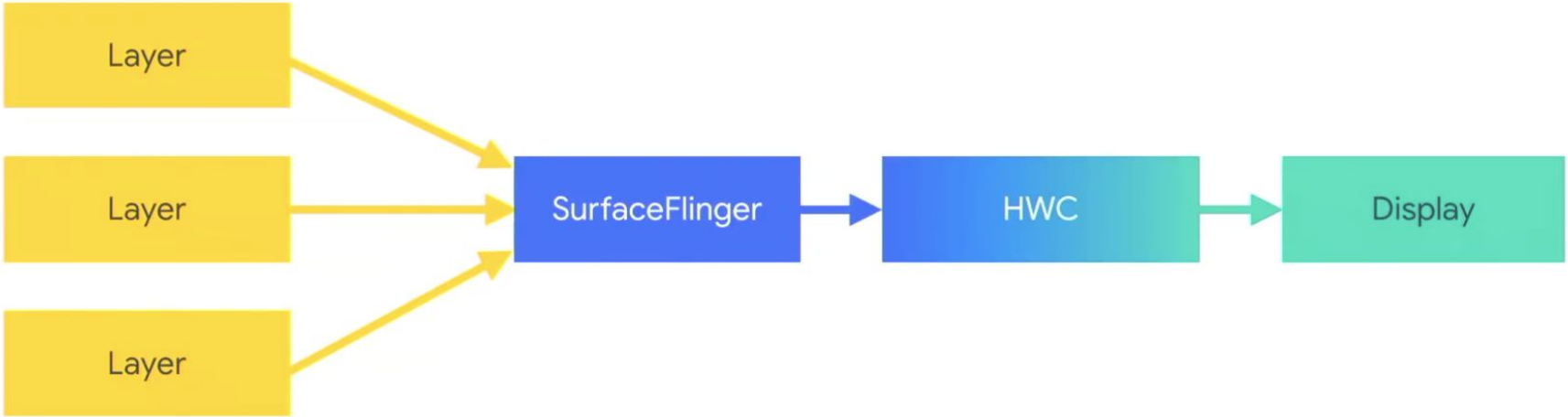
Producers/Consumers

- SurfaceView
- SurfaceTexture
- OpenGL ES
  - First draw: `dequeueBuffer()`
  - `eglSwapBuffers()`: `queueBuffer()`
- Vulkan
- MediaCodec
- MediaPlayer

Composition

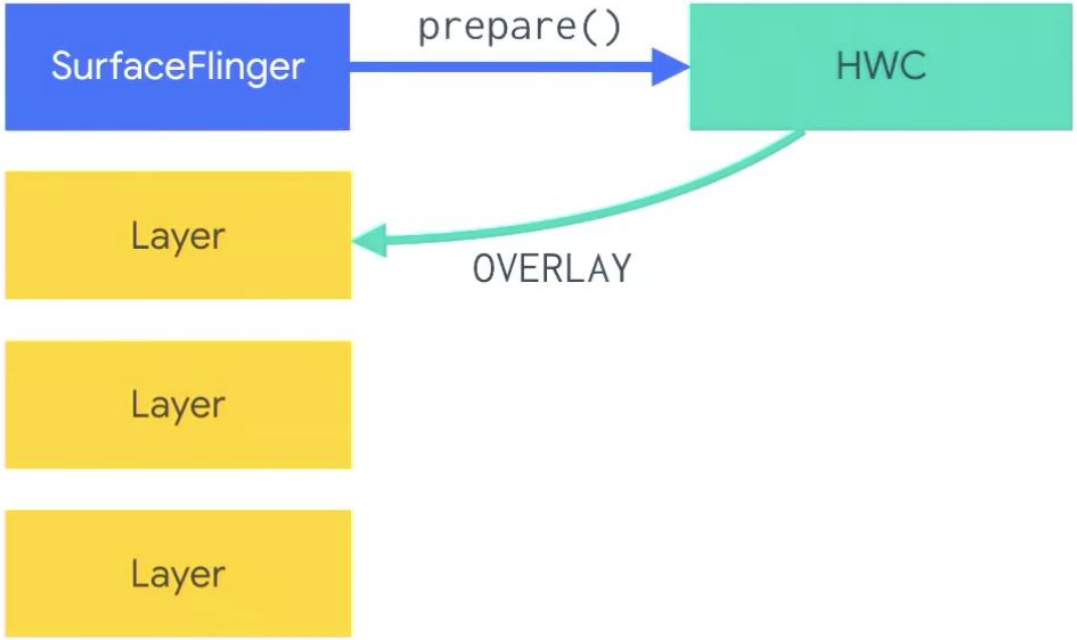


Composition

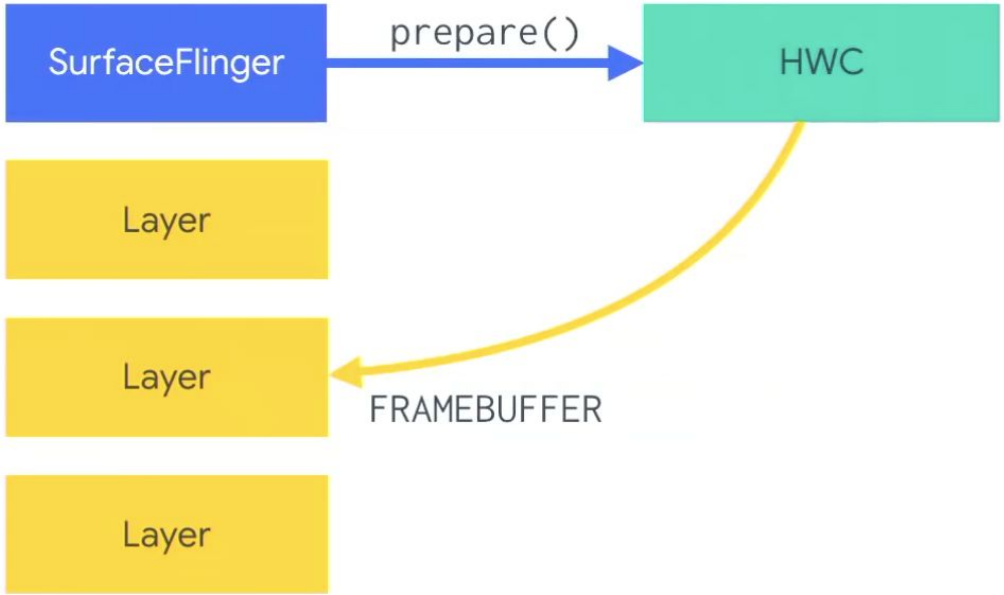


Hardware Composer : hardware abstraction layer,as android wants to avoid using gpu when while composition

Composition



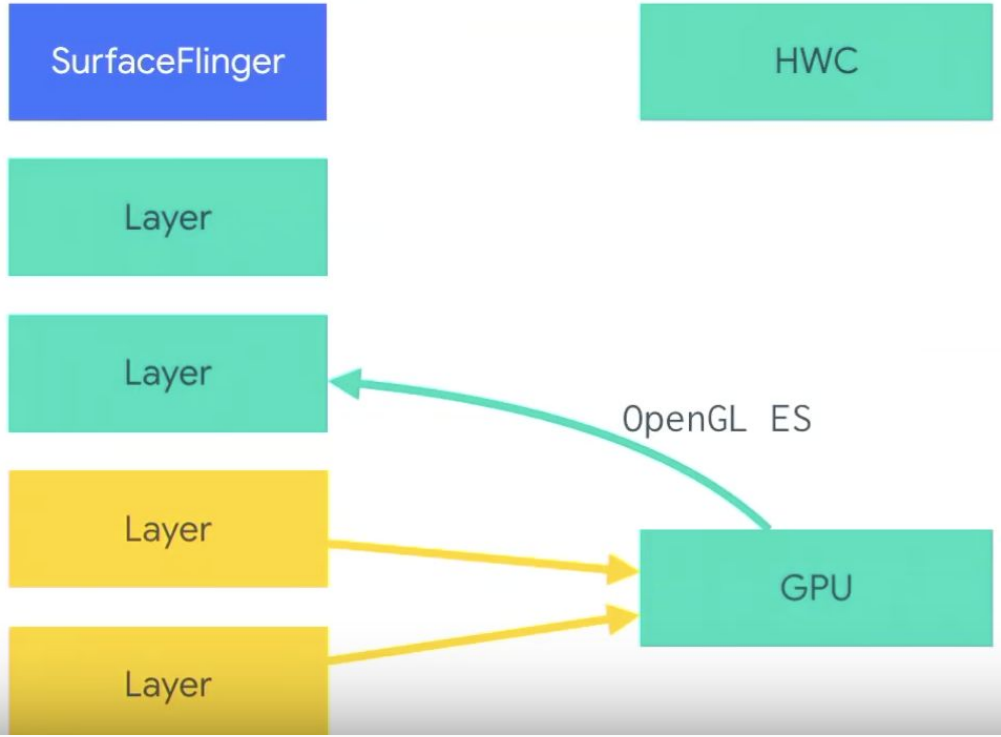
Composition



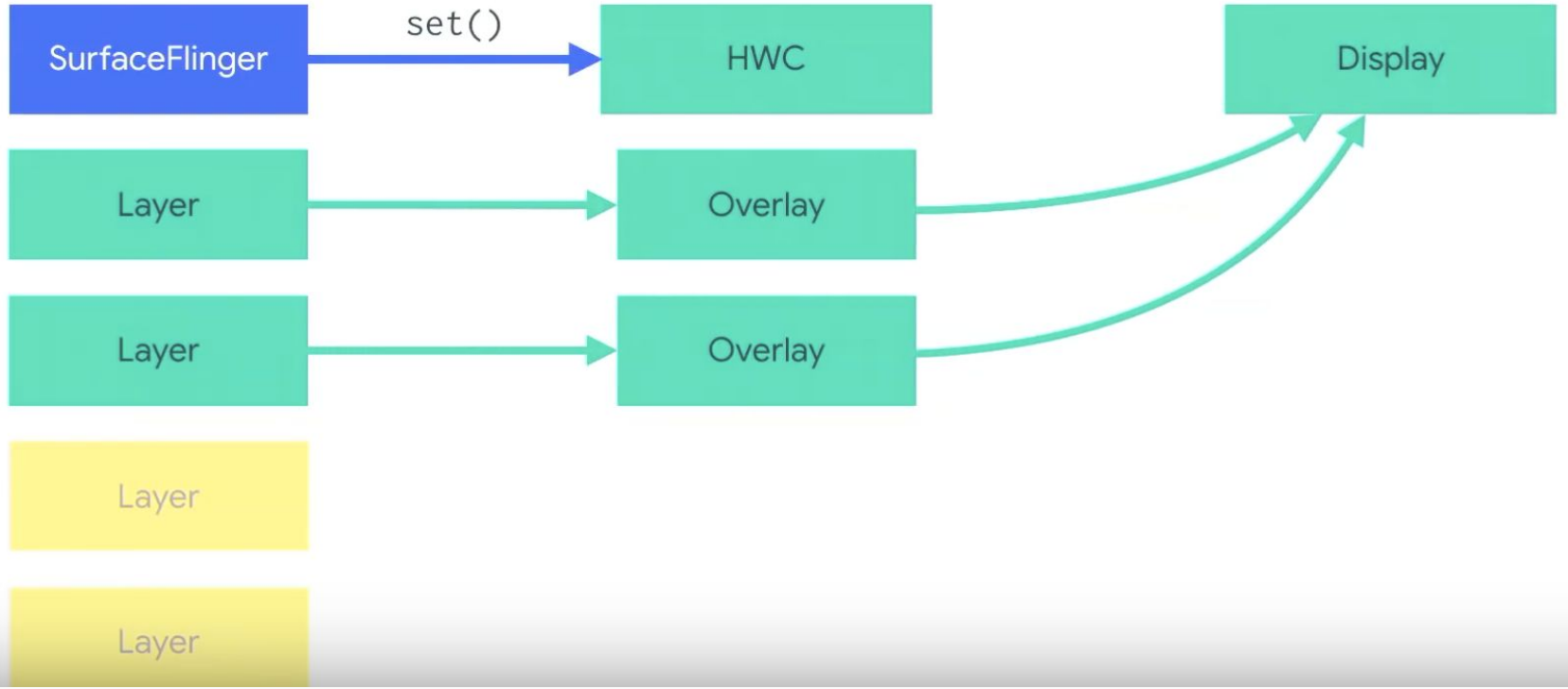
Too many layers, rotations

Pixel has 7 layers

Composition



Composition



Absolutely NO Questions Please

Thanks

References

- https://github.com/google/grafika
- https://source.android.com/devices/graphics/architecture
- https://www.youtube.com/watch?v=zdQRIYOST64