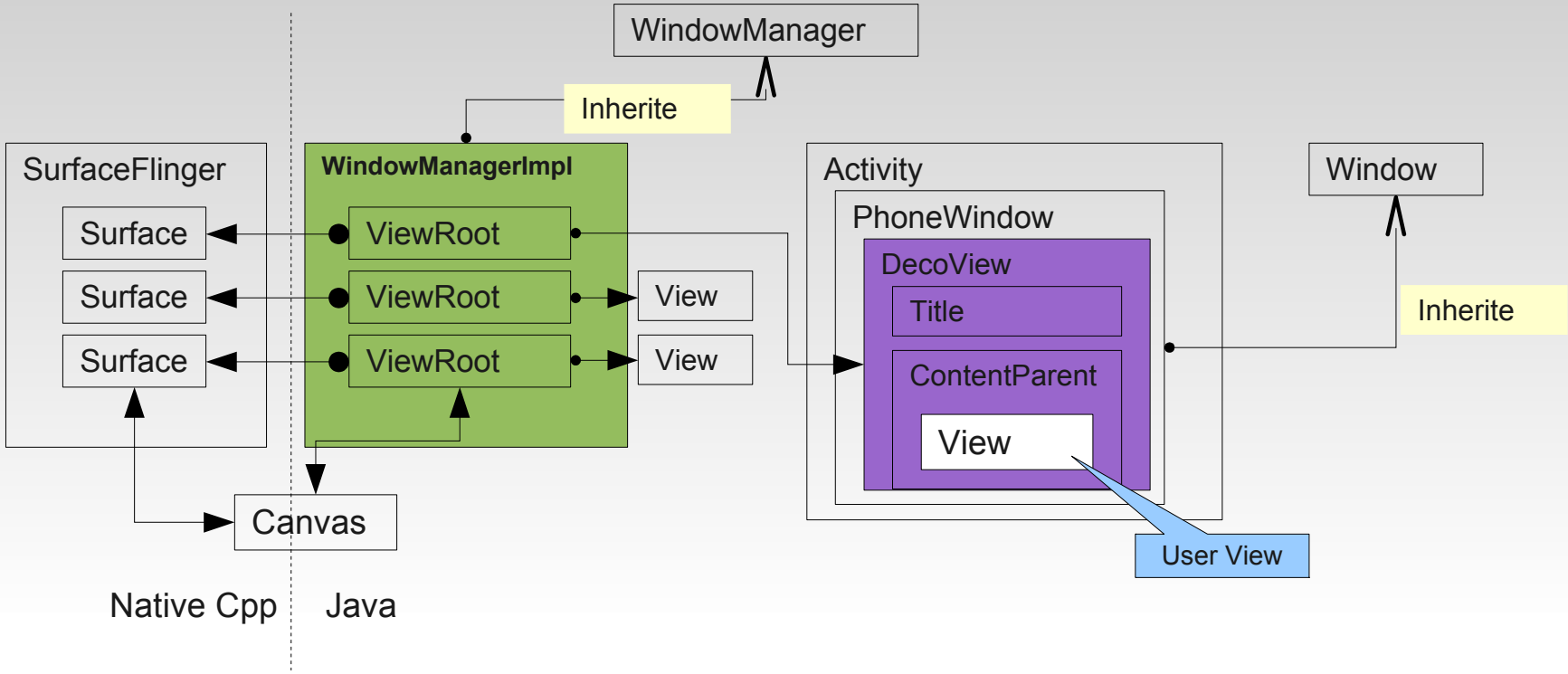
Android UI framework & infrastructures

Ruijun Xu © shakewave@hotmail.com



- **How does a View in an Activity draw itself on the screen**
- 1. ActivityManagerService → ActivityThread → launch an Activity,
- calling Instrumentation.newActivity() to create an Activity
- 2. Then a PhoneWindow has been created in Activity.attach(), and the Activity acquired a globally unique WindowManager instance.

3. User call Activity.setContentView() to set user view layout. It will add the user view into the ContentParent of PhoneWindow

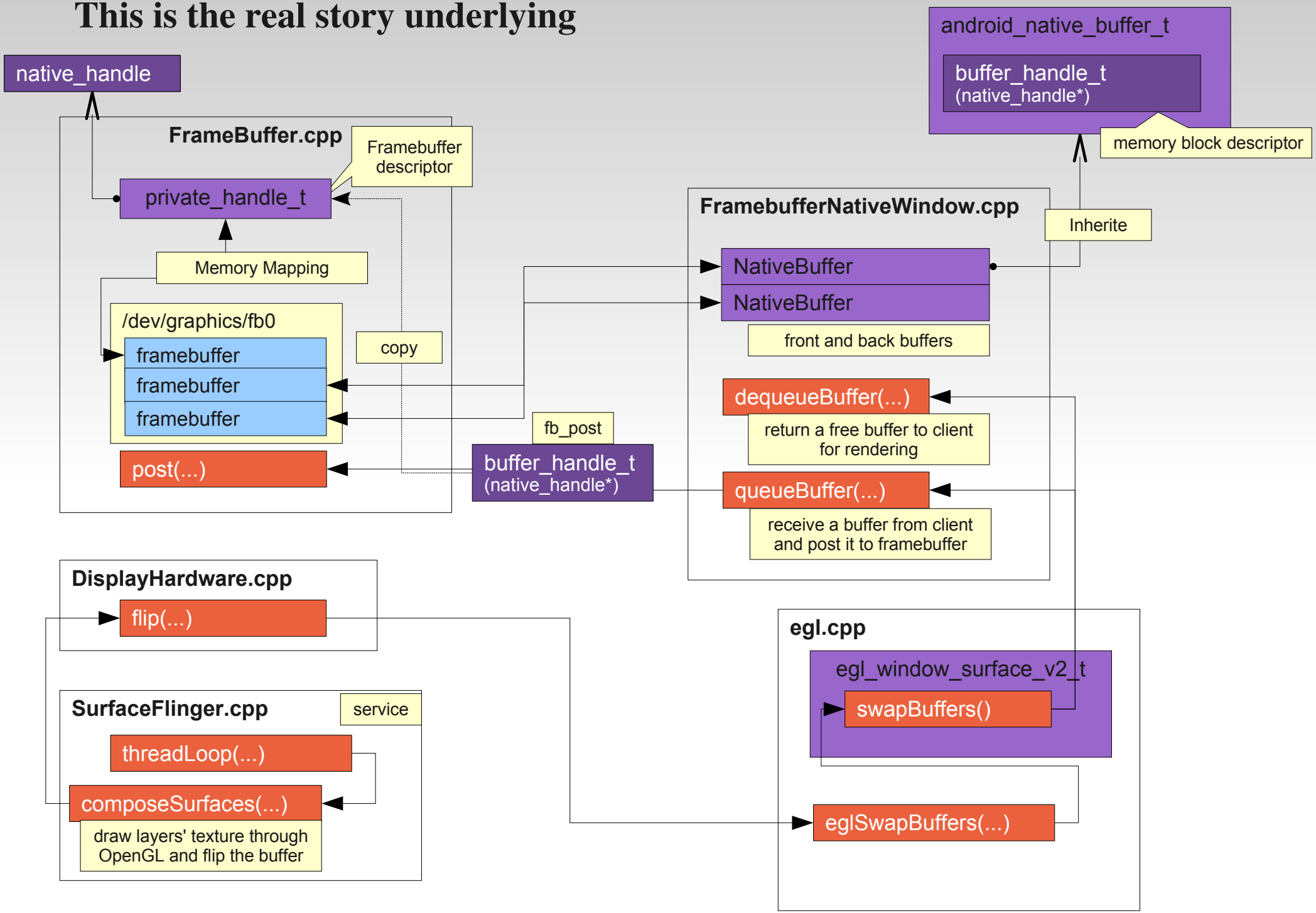
- 4. ActivityThread continues executing, calls Activity.makeVisible() to add the DecoView into WindowManager, WindowManager will create a coresponding ViewRoot for the DecoView.
- 5. Each ViewRoot owns a Surface, a Surface class will delegate to its nativity class to create a block of memory buffer for graphic drawing. Those memory buffers and their creations are transparent to Java layer, they are being managed by a native clas called SurfaceFlinger. SurfaceFlinger is also responsible for merging all Surfaces into one image and draw it on the device screen.
- 6. A user view draws its UI through a Canvas which is obtained from Surface orignally. If a View has changed its appearance, the notification will be escalated all the way up to the ViewRoot through the UI hierarchy. Then the ViewRoot obtains a Canvas from its Surface, and call View.onDraw(Canvas c) on the Canvas. Once the drawing has finished, it notifies the Surface to submit the canvas, then the SurfaceFlinger gets notified and merges the changed Surface together with other unchanged Surfaces to the device screen.

Tips

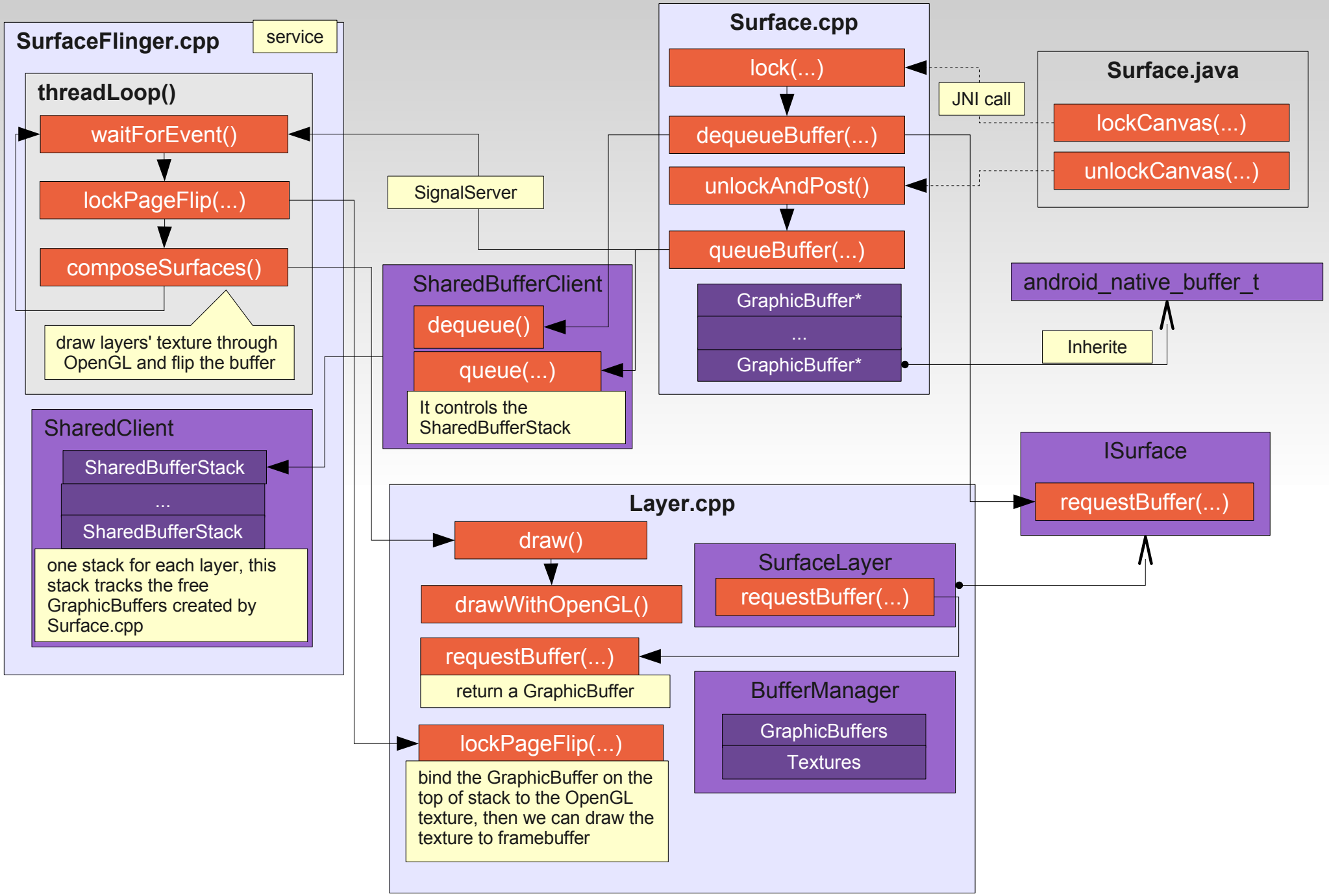
- 1. Programmer can show a View through WindowManager directly, not necessarily to create an Activity or Dialog.
- 2. Dialogs, Menus are also displayed through WindowManager.
- 3. Showing a View through WindowManager will increase memory usage, 'cause created an extra Surface.
- 4. If a View changes constantly or it performs a continous animation on the screen, and other parts of UI are complicated, it'd be better to display it through WindowManager (this reduces unnecessary redraws of other parts).
- 5. Android also provides a SurfaceView which is a View owns a Surface alone. A SurfaceView can be embeded into any layout just like a normal view. Programmers can leverage the Surface to control drawing efficiently and flexibly. And there is also a GLSurfaceView which supports OpenGL.

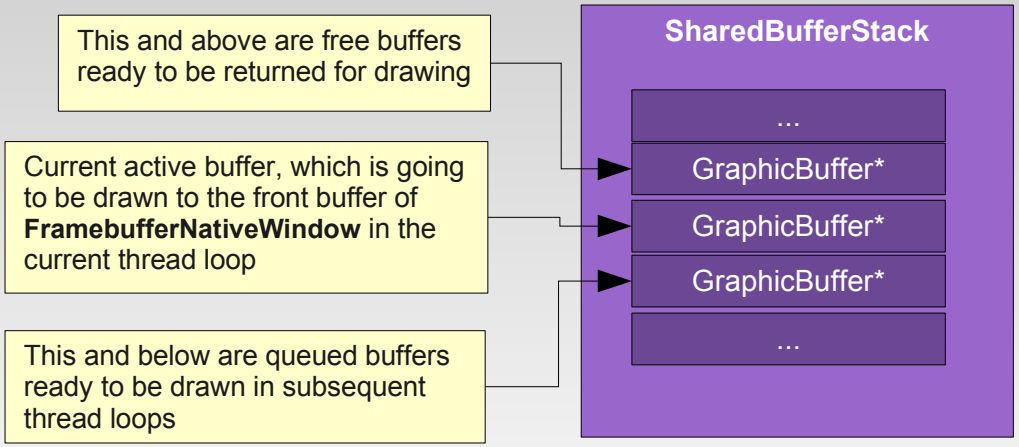
For example, if we want to implement a gif player control. SurfaceView would be a good choice. We could use a rendering thread to draw animation frames, the drawing is totally independ from the UI hierarchy.

This is the real story underlying



A picture is worth a thousand words


SurfaceFlinger.cpp
service
threadLoop()
waitForEvent()
lockPageFlip(...)
composeSurfaces()
draw layers' texture through OpenGL and flip the buffer
SignalServer
Surface.cpp
lock(...)
dequeueBuffer(...)
unlockAndPost()
queueBuffer(...)
GraphicBuffer*
...
GraphicBuffer*
JNI call
Surface.java
lockCanvas(...)
unlockCanvas(...)
android_native_buffer_t
Inherite
SharedBufferClient
dequeue()
queue(...)
It controls the SharedBufferStack
SharedClient
SharedBufferStack
...
SharedBufferStack
one stack for each layer, this stack tracks the free GraphicBuffers created by Surface.cpp
ISurface
requestBuffer(...)
Layer.cpp
draw()
drawWithOpenGL()
SurfaceLayer
requestBuffer(...)
requestBuffer(...)
return a GraphicBuffer
BufferManager
GraphicBuffers
Textures
lockPageFlip(...)
bind the GraphicBuffer on the top of stack to the OpenGL texture, then we can draw the texture to framebuffer


This and above are free buffers ready to be returned for drawing
Current active buffer, which is going to be drawn to the front buffer of **FramebufferNativeWindow** in the current thread loop
This and below are queued buffers ready to be drawn in subsequent thread loops
SharedBufferStack
...
GraphicBuffer*
GraphicBuffer*
GraphicBuffer*
...