The Android graphics path

in depth

License

About Chris Simmonds

- Consultant and trainer
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at `http://2net.co.uk/`

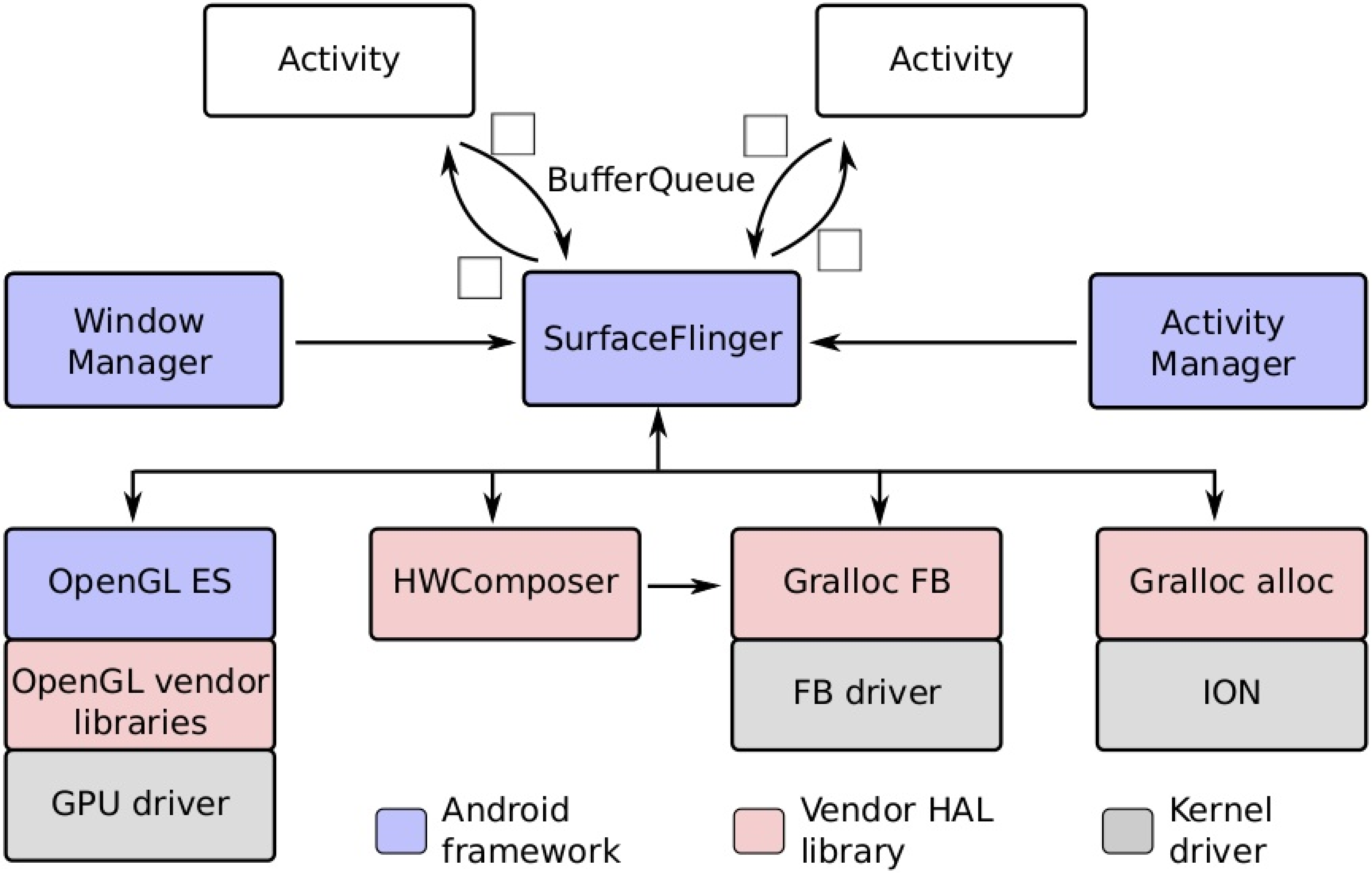
in `https://uk.linkedin.com/in/chrisdsimmonds/`

g+ `https://google.com/+chrissimmonds`

Overview

- The Android graphics stack changed a lot in Jelly Bean as a result of *project Butter*
- This presentation describes the current (JB) graphics stack from top to bottom
- Main topics covered
  - The application layer
  - SurfaceFlinger, interfaces and buffer queues
  - The hardware modules HWComposer and Gralloc
  - OpenGL ES and EGL

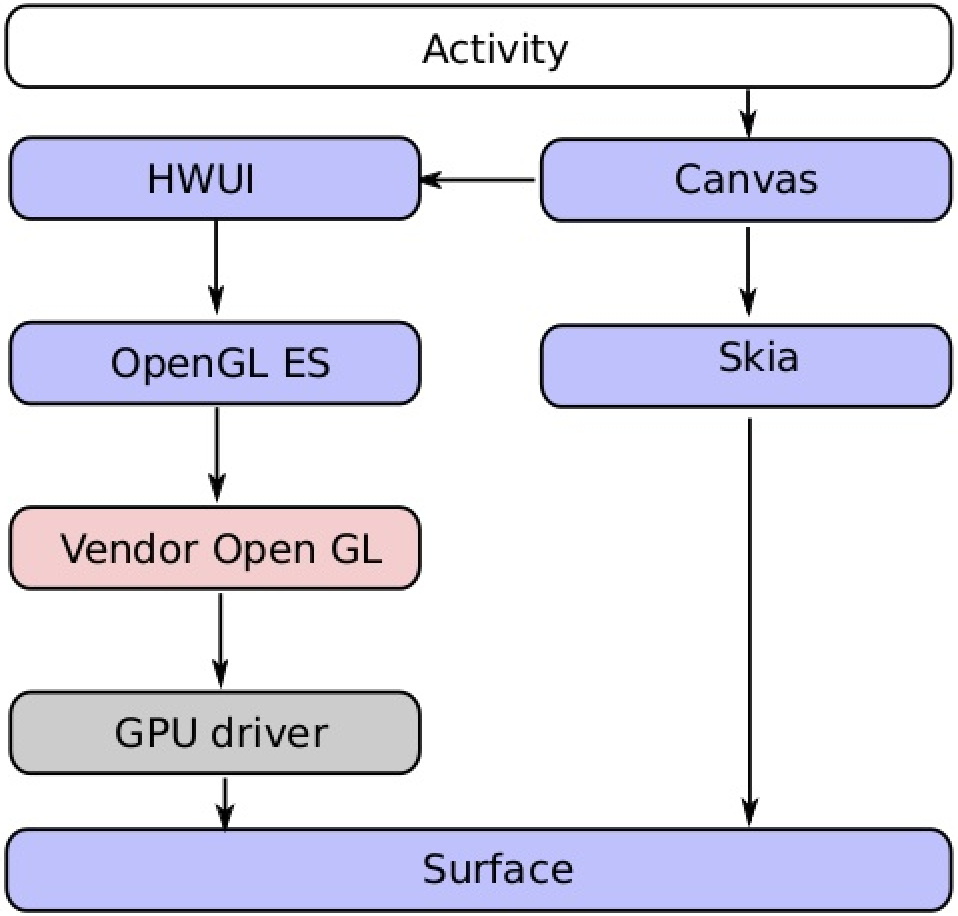
The big picture



Inception of a pixel

- Everything begins when an activity draws to a surface
- 2D applications can use
  - drawing functions in Canvas to write to a Bitmap: `android.graphics.Canvas.drawRect()`, `drawText()`, etc
  - descendants of the View class to draw objects such as buttons and lists
  - a custom View class to implement your own appearance and behaviour
- In all cases the drawing is rendered to a *Surface* which contains a *GraphicBuffer*

2D rendering path



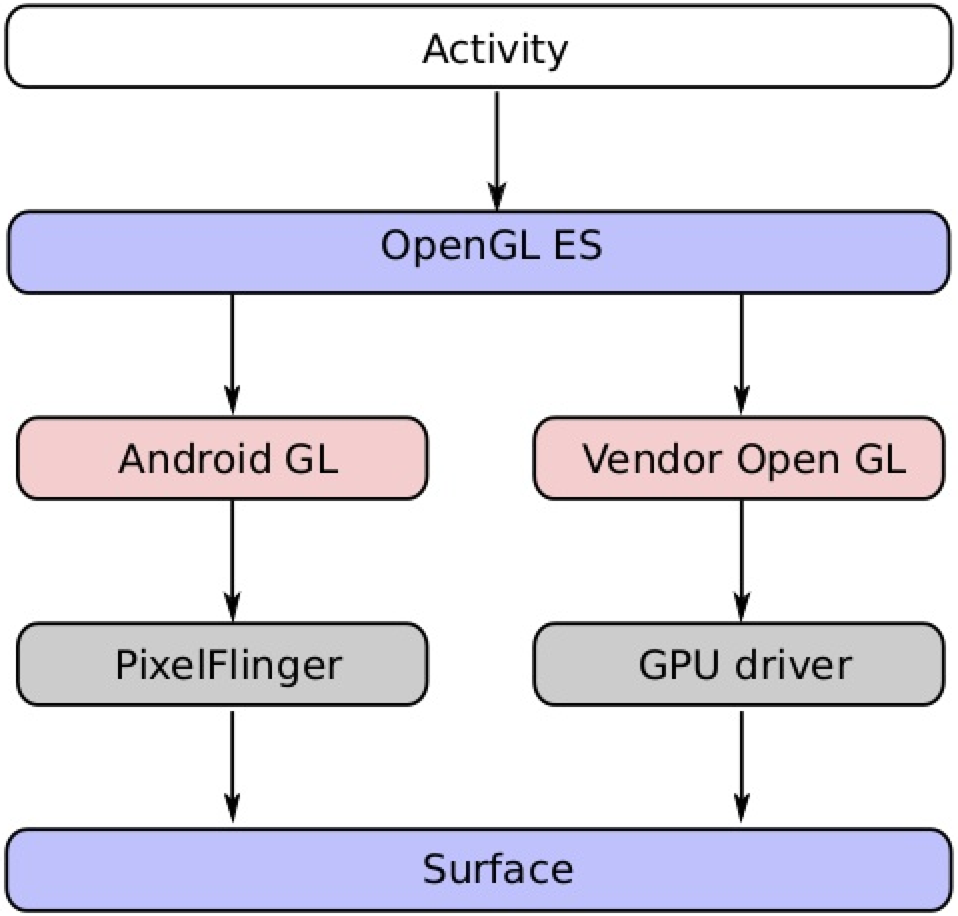
Skia and hwui

- For 2D drawing there are two rendering paths
    - hwui: (libwhui.so) hardware accelerated using OpenGL ES 2.0
    - skia: (libskia.so) software render engine
- hwui is the default
- Hardware rendering can be disabled per view, window, activity, application or for the whole device
    - Maybe for comparability reasons: hwui produces results different to skia in some (rare) cases

3D rendering path

- An activity can instead create a *GLSurfaceView* and use OpenGL ES bindings for Java (the android.opengl.* classes)
- Using either the vendor GPU driver (which must support OpenGL ES 2.0 and optinally 3.0)
- Or as a fall-back, using PixelFlinger, a software GPU that implements OpenGL ES 1.0 only
- Once again, the drawing is rendered to a Surface

3D rendering path



Activity

OpenGL ES

Android GL

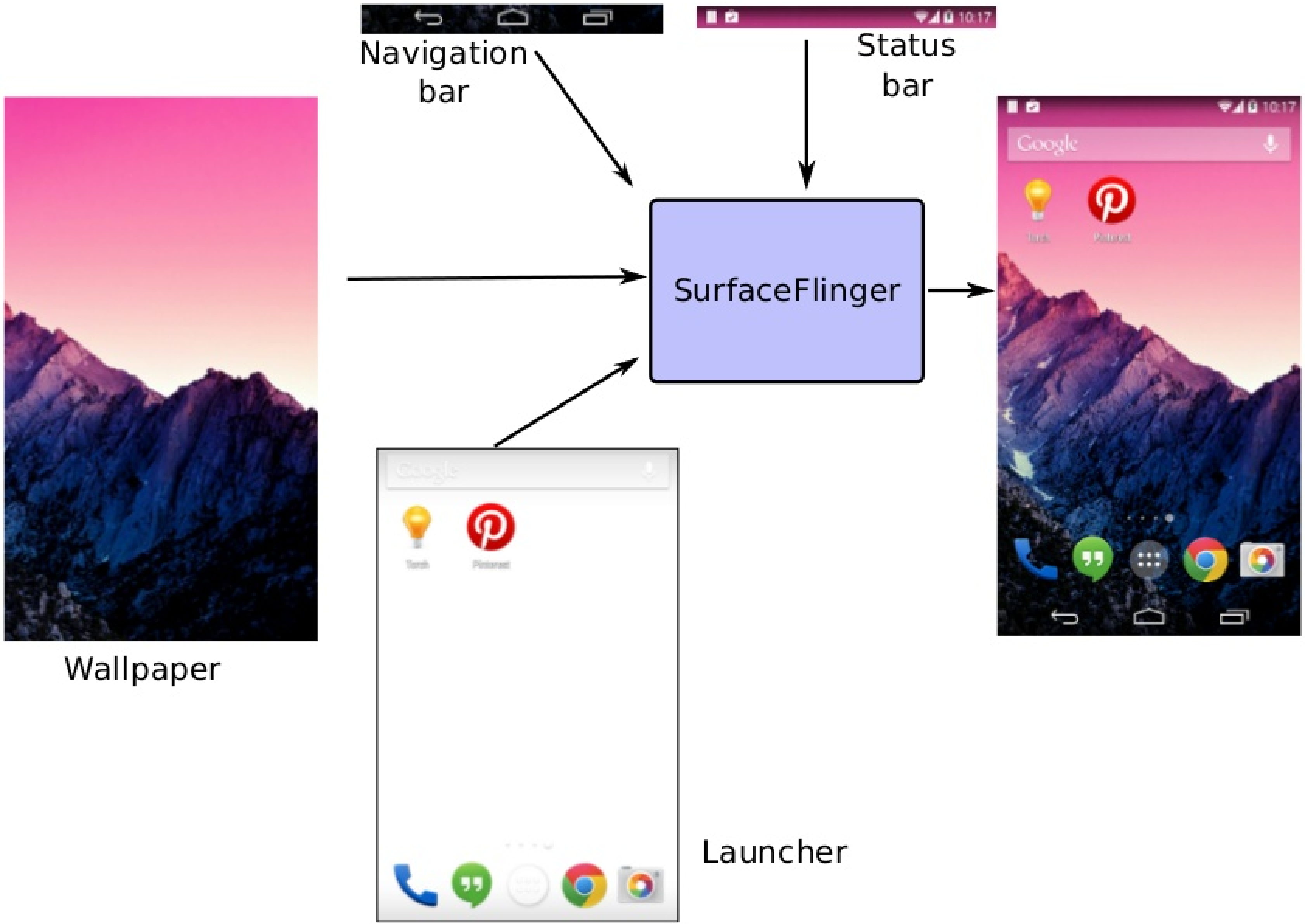
Vendor Open GL

PixelFlinger

GPU driver

Surface

Composition



Navigation bar

Status bar

Wallpaper

SurfaceFlinger

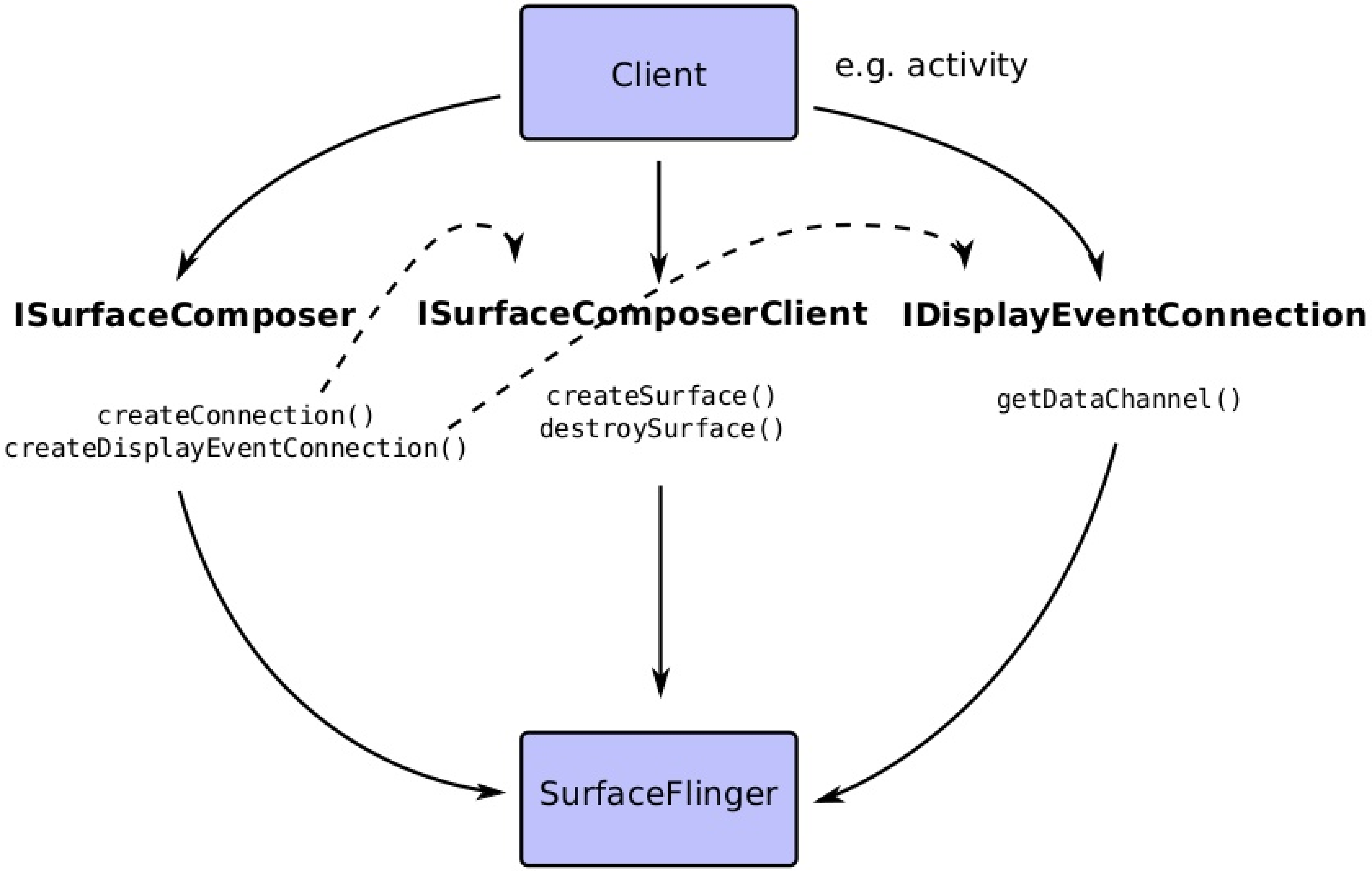
Launcher

SurfaceFlinger

`frameworks/native/services/surfaceflinger`

- A high-priority native (C++) daemon, started by init with UID=*system*
- Services connections from activities via Binder interface *ISurfaceComposer*
- Receives activity status from Activity Manager
- Receives window status (visibility, Z-order) from Window Manager
- Composits multiple Surfaces into a single image
- Passes image to one or more displays
- Manages buffer allocation, synchronisation

SurfaceFlinger binder interfaces



ISurfaceComposer

- ISurfaceComposer
  - Clients use this interface to set up a connection with SurfaceFlinger
  - Client begins by calling *createConnection()* which spawns an ISurfaceComposerClient
  - Client calls *createGraphicBufferAlloc()* to create an instance of IGraphicBufferAlloc (discussed later)
  - Client calls *createDisplayEventConnection()* to create an instance of IDisplayEventConnection
  - Other methods include *captureScreen()* and *setTransactionState()*

ISurfaceComposerClient

- ISurfaceComposerClient
  - This interface has two methods:
  - *createSurface()* asks SufraceFlinger to create a new Surface
  - *destroySurface()* destroys a Surface

IDisplayEventConnection

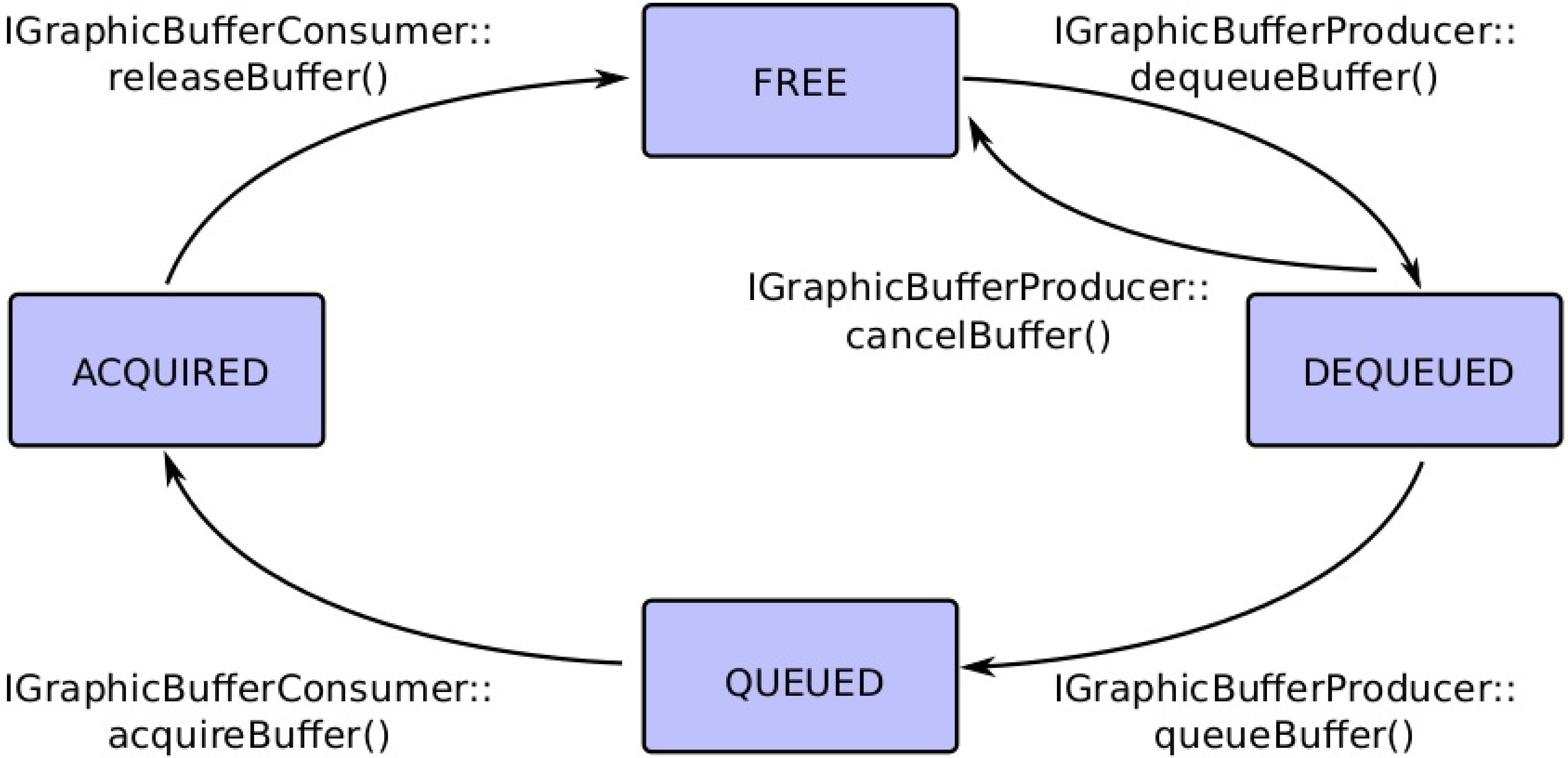
- IDisplayEventConnection
  - This interface passes vsync event information from SurfaceFlinger to the client
  - *setVsyncRate()* sets the vsync event delivery rate: value of 1 returns all events, 0 returns none
  - *requestNextVsync()* schedules the next vsync event: has no effect if the vsync rate is non zero
  - *getDataChannel()* returns a BitTube which can be used to receive events

BufferQueue

`frameworks/native/include/gui/BufferQueue.h`

- Mechanism for passing GraphicBuffers to SurfaceFlinger
- Contains an array of between 2 and 32 GraphicBuffers
- Uses interface *IGraphicBufferAlloc* to allocate buffers (see later)
- Provides two Binder interfaces
    - *IGraphicBufferProducer* for the client (Activity)
    - *IGraphicBufferConsumer* for the consumer (SurfaceFlinger)
- Buffers cycle between producer and consumer

BufferQueue state diagram



IGraphicBufferConsumer::
releaseBuffer()

FREE

IGraphicBufferProducer::
dequeueBuffer()

IGraphicBufferProducer::
cancelBuffer()

ACQUIRED

DEQUEUED

QUEUED

IGraphicBufferConsumer::
acquireBuffer()

IGraphicBufferProducer::
queueBuffer()

BufferQueue

- Default number of buffer slots since JB is 3 (previously 2)
  - In JB you can compile Layer.cpp with `TARGET_DISABLE_TRIPLE_BUFFERING` to return to 2 slots
- Call *setBufferCount()* to change the number of slots
- BufferQueue operates in two modes:
  - Synchronous: client blocks until there is a free slot
  - Asynchronous: queueBuffer() discards any existing buffers in QUEUED state so the queue only holds the most recent frame

GraphicBuffer

`frameworks/native/include/ui/GraphicBuffer.h`

- Represents a buffer, wraps ANativeWindowBuffer
- Attributes including *width, height, format, usage* inherited from ANativeWindowBuffer