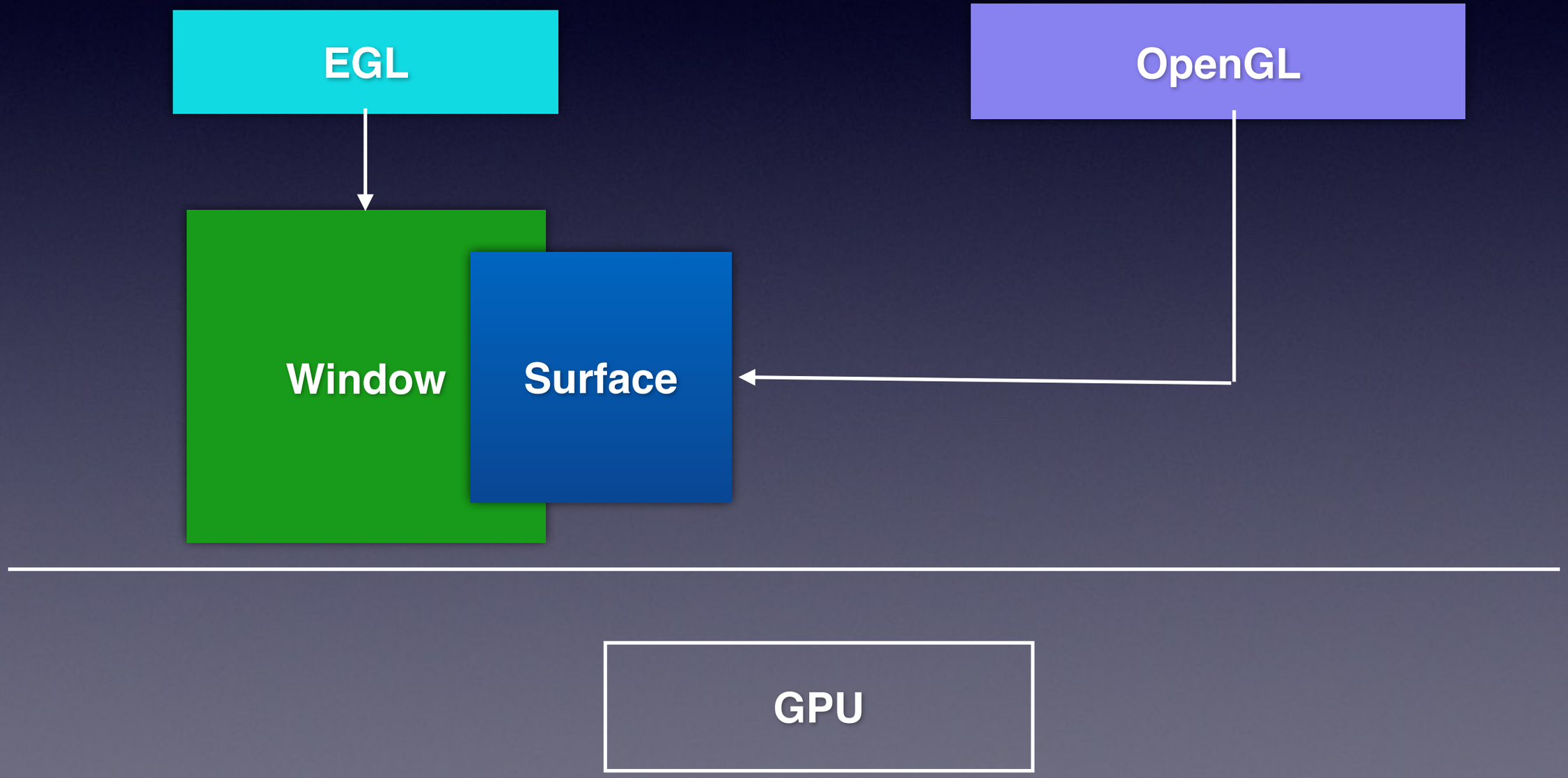
SurfaceFlinger基础

jacob_shen@realsil.com.cn
Tel:5307

OpenGL & EGL



NativeWindow

窗口是什么

描述窗口的一系列属性

窗口缓冲区

操作窗口及其缓冲区的一系列方法

NativeWindow

Android窗口如何定义的

窗口的定义看出，是对其形态和功能的一个规定

```
struct ANativeWindow
{
    const uint32_t flags;
    const int   minSwapInterval;
    const int   maxSwapInterval;
    const float xdpi;
    const float ydpi;

    int     (*setSwapInterval)(struct ANativeWindow* window,
                int interval);
    int     (*query)(const struct ANativeWindow* window,
                int what, int* value);
    int     (*perform)(struct ANativeWindow* window,
                int operation, ... );
    int     (*dequeueBuffer)(struct ANativeWindow* window,
                struct ANativeWindowBuffer** buffer, int* fenceFd);
    int     (*queueBuffer)(struct ANativeWindow* window,
                struct ANativeWindowBuffer* buffer, int fenceFd);
    int     (*cancelBuffer)(struct ANativeWindow* window,
                struct ANativeWindowBuffer* buffer, int fenceFd);
};
```

NativeWindow

OpenGL如何使用Window的

```
EGLSurface eglCreateWindowSurface(  EGLDisplay dpy, EGLConfig config,
                                    NativeWindowType window,
                                    const EGLint *attrib_list)
{
    return createWindowSurface(dpy, config, window, attrib_list);
}
```

```
#if defined(_WIN32) || defined(__VC32__) && !defined(__CYGWIN__) && !d

#include <windows.h>

typedef HWND    EGLNativeWindowType;

#elif defined(__WINSCW__) || defined(__SYMBIAN32__)  /* Symbian */

typedef void *EGLNativeWindowType;

#elif defined(__ANDROID__) || defined(ANDROID)

struct ANativeWindow;
typedef struct ANativeWindow*           EGLNativeWindowType;

#elif defined(__unix__)

/* X11 (tentative)  */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

typedef Window   EGLNativeWindowType;

#else
```

NativeWindow

OpenGL如何使用Window的

```
EGLBoolean egl_window_surface_v2_t::connect()
{
    // we're intending to do software rendering
    native_window_set_usage(nativeWindow,
            GRALLOC_USAGE_SW_READ_OFTEN | GRALLOC_USAGE_SW_WRITE_OFTEN);

// dequeue a buffer
int fenceFd = -1;
if (nativeWindow->dequeueBuffer(nativeWindow, &buffer,
        &fenceFd) != NO_ERROR) {
    return setError(EGL_BAD_ALLOC, EGL_FALSE);
}

// wait for the buffer
sp<Fence> fence(new Fence(fenceFd));
if (fence->wait(Fence::TIMEOUT_NEVER) != NO_ERROR) {
    nativeWindow->cancelBuffer(nativeWindow, buffer, fenceFd);
    return setError(EGL_BAD_ALLOC, EGL_FALSE);
}

// allocate a corresponding depth-buffer
width = buffer->width;
height = buffer->height;
```

NativeWindow

Android窗口具体实现

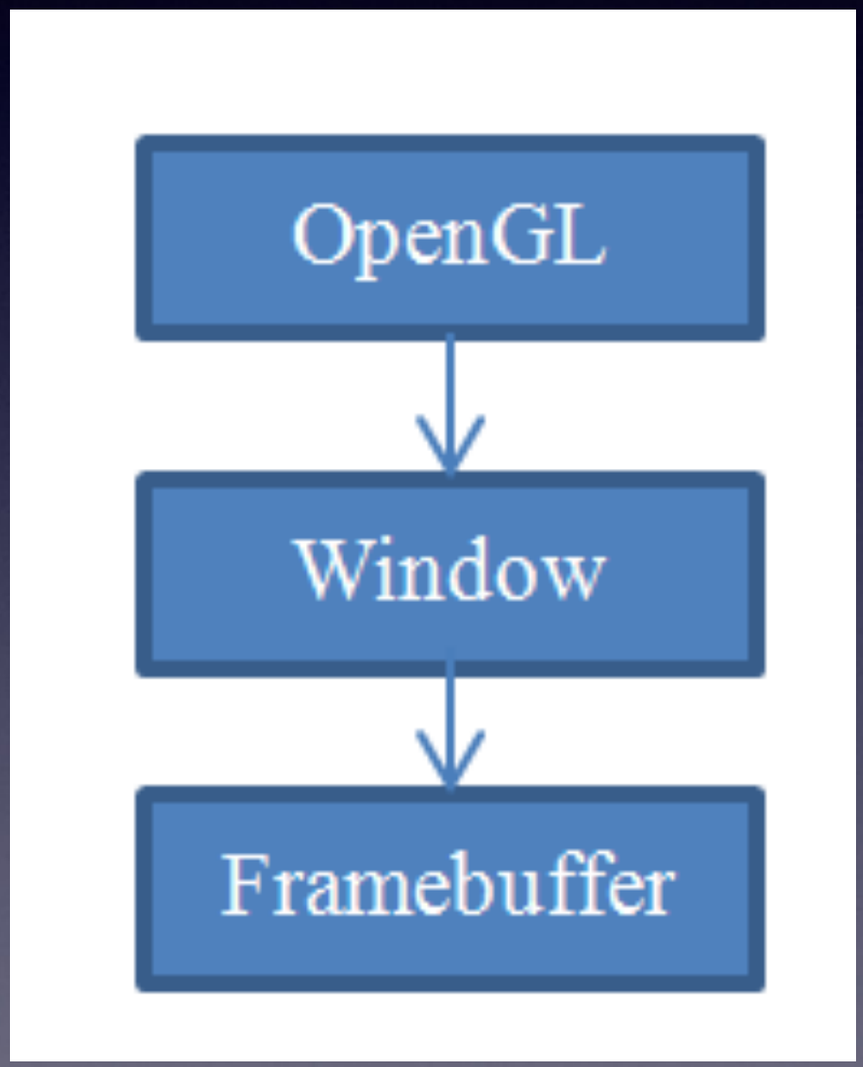
```
class Surface
    : public ANativeObjectBase<ANativeWindow, Surface, RefBase>
{
```

NativeWindow

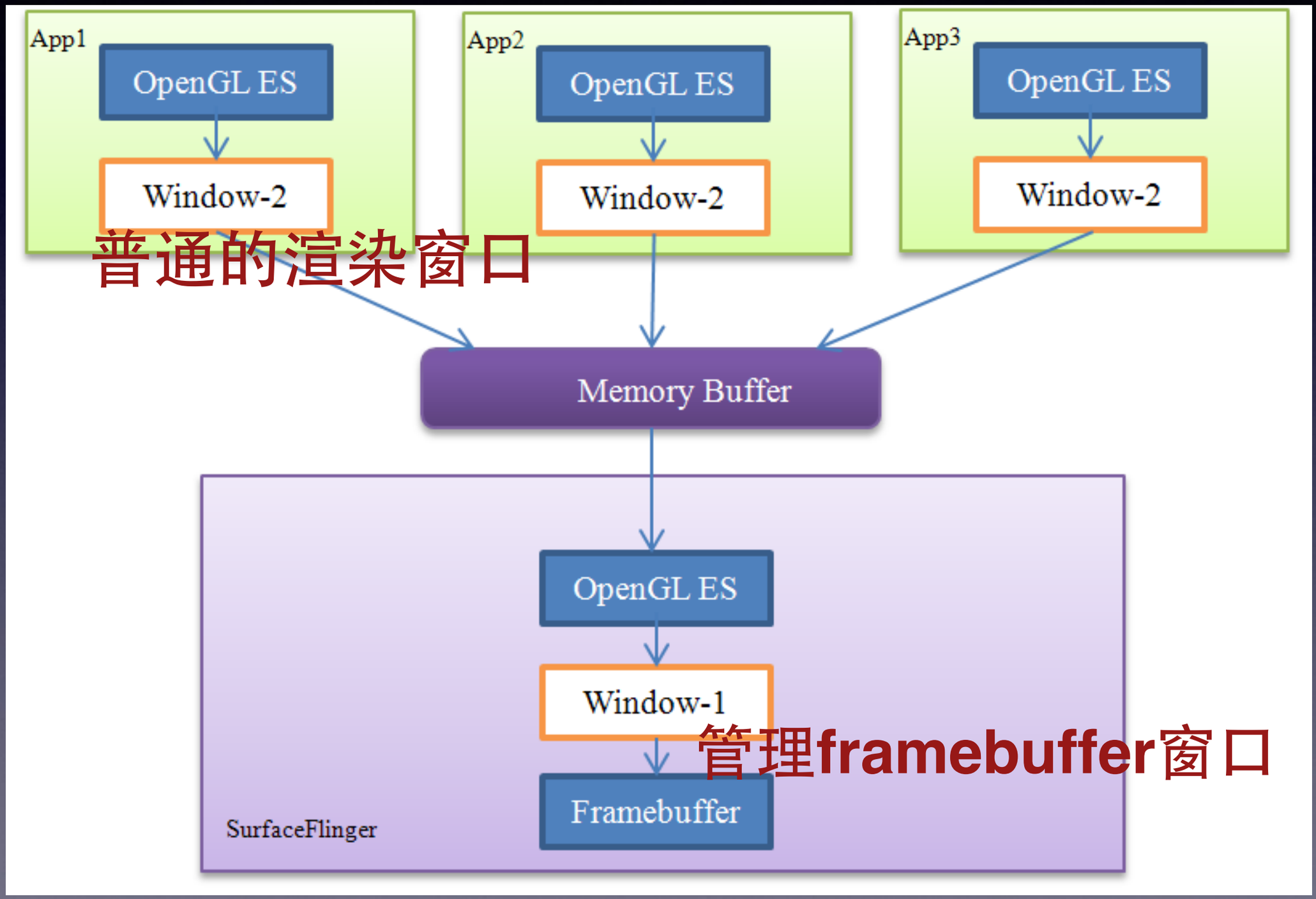
Android会遇到哪些窗口

先看一个设计模型

NativeWindow



NativeWindow

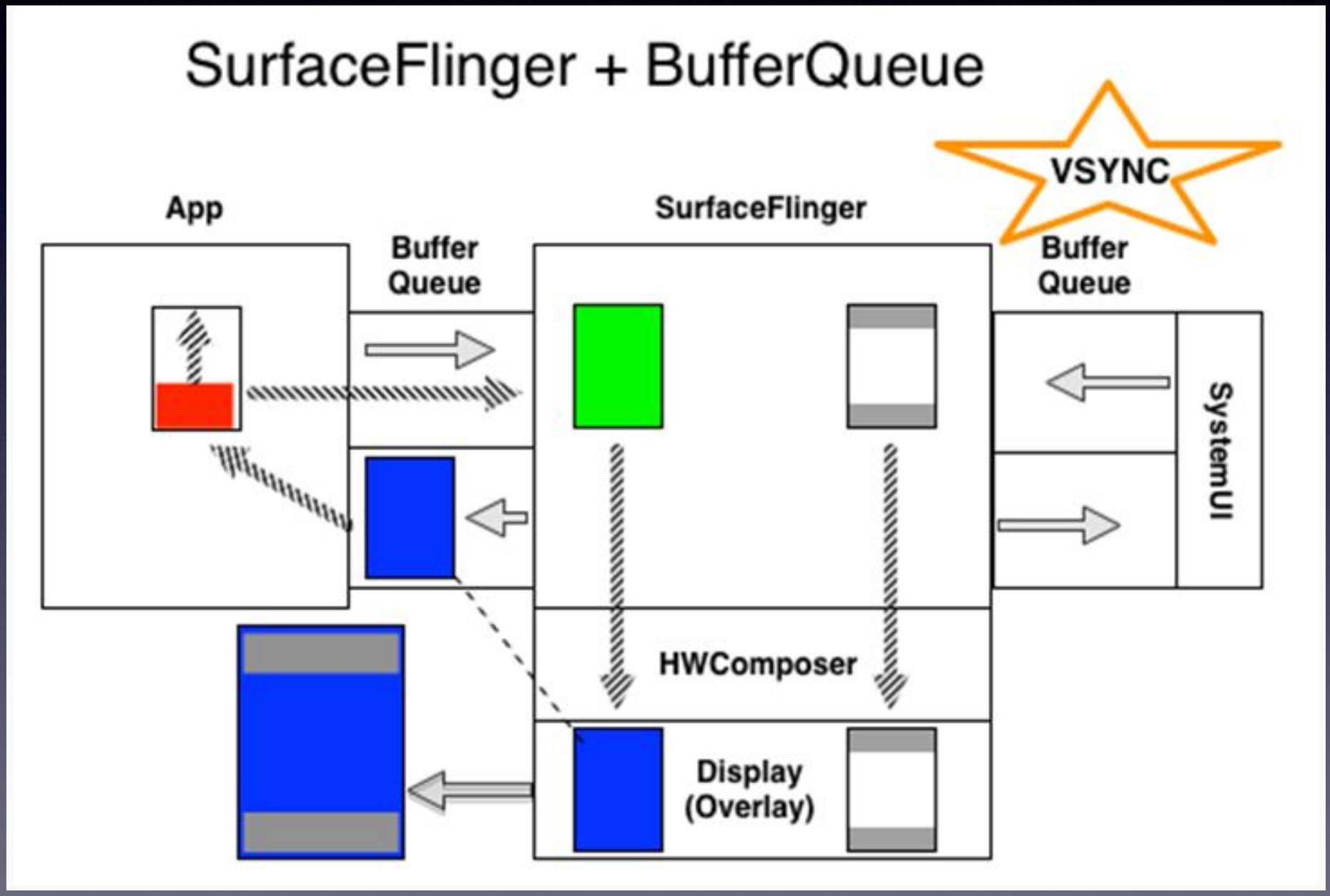


NativeWindow

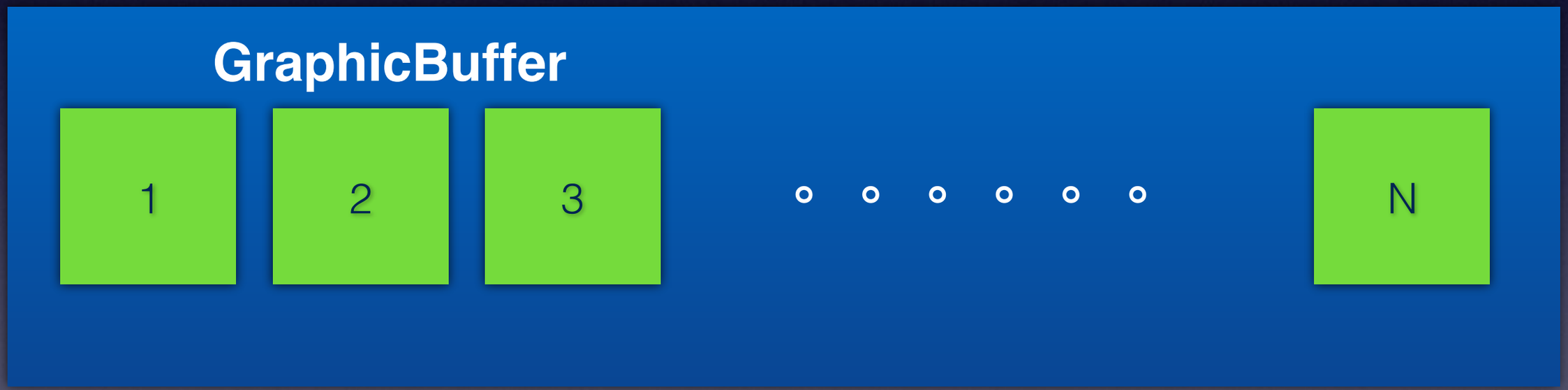
窗口的缓冲区哪里来

```
class BufferQueue : public BnGraphicBufferProducer,
                    public BnGraphicBufferConsumer,
                    private IBinder::DeathRecipient {
public:
    enum { MIN_UNDEQUEUED_BUFFERS = 2 };
    enum { NUM_BUFFER_SLOTS = 32 };
    enum { NO_CONNECTED_API = 0 };
```

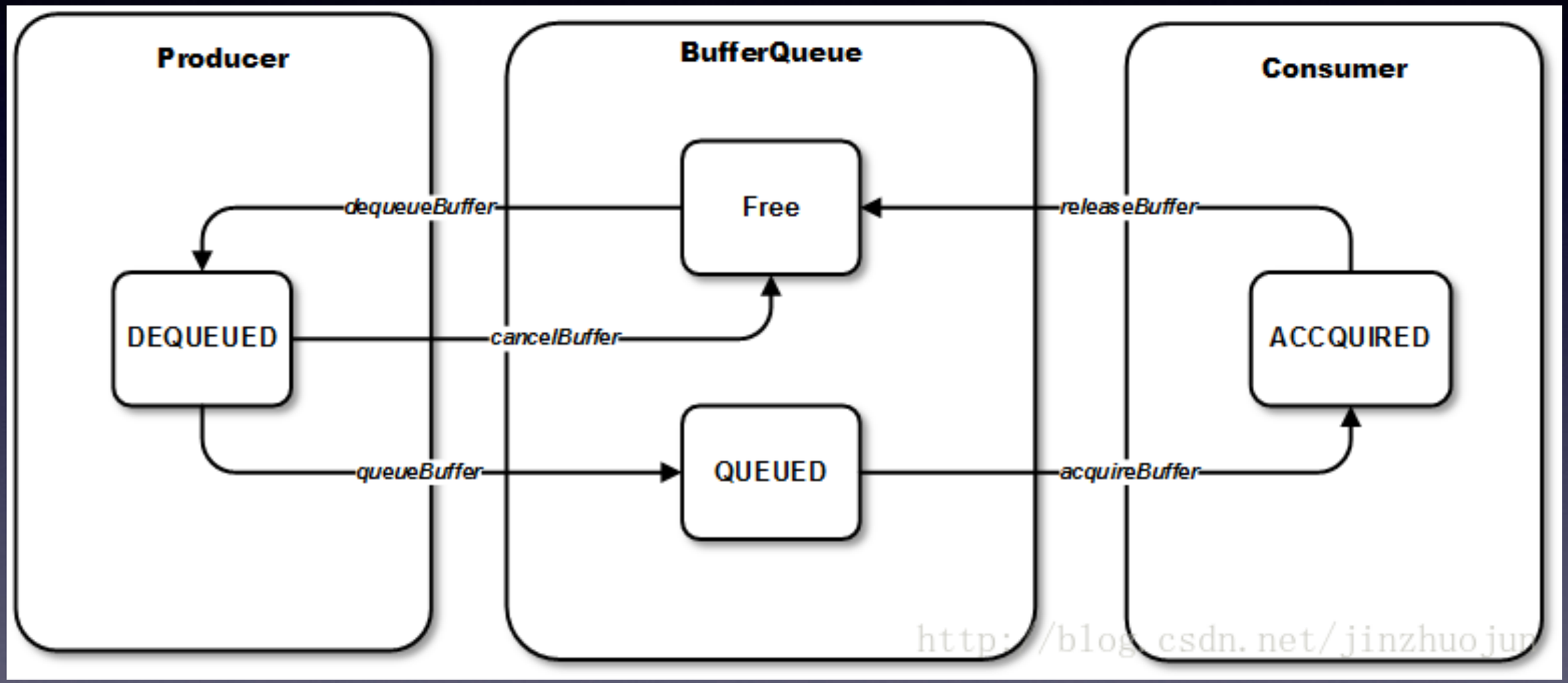
BufferQueue



BufferQueue

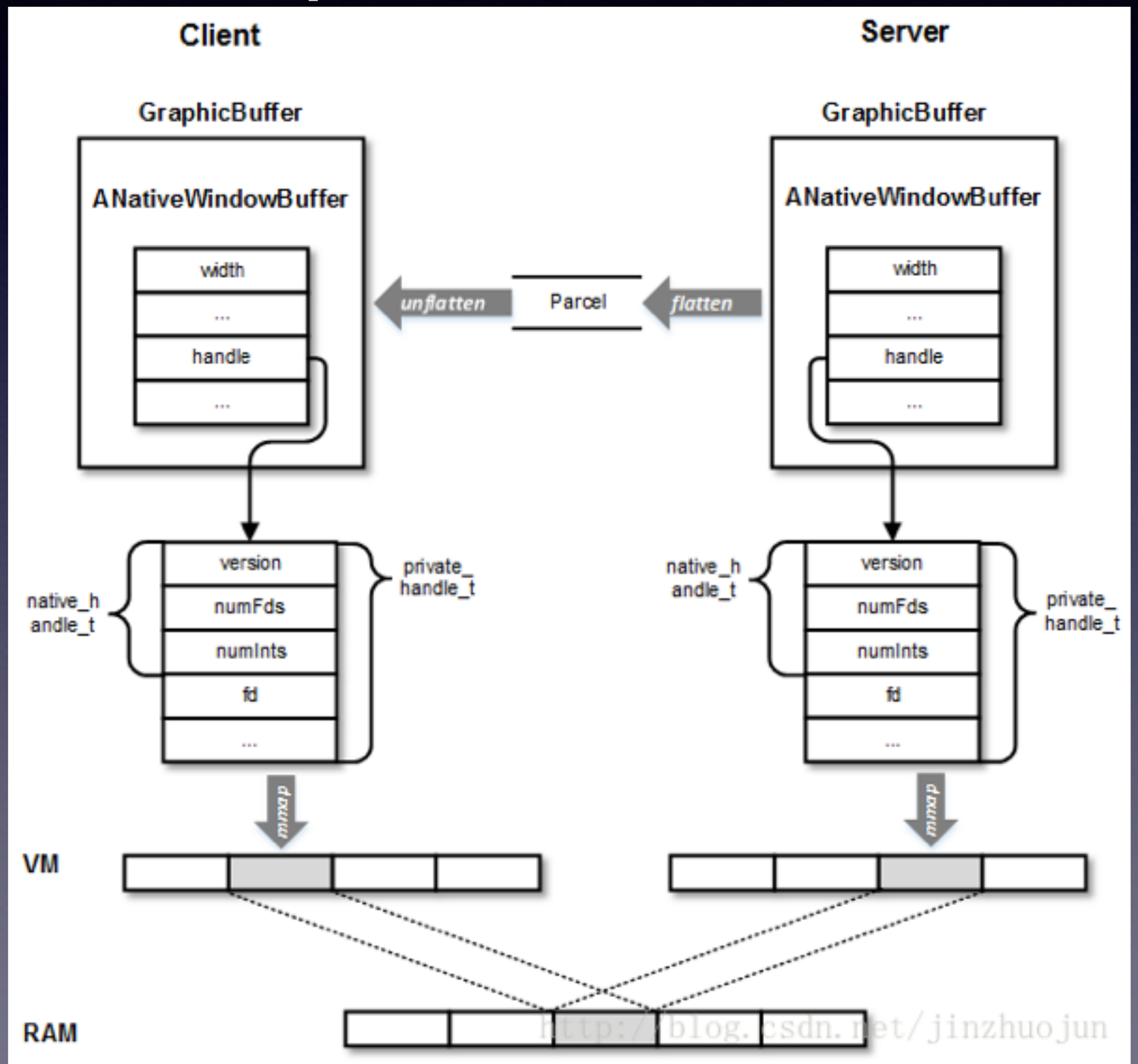


BufferQueue



BufferQueue

GraphicBuffer如何Share



BufferQueue

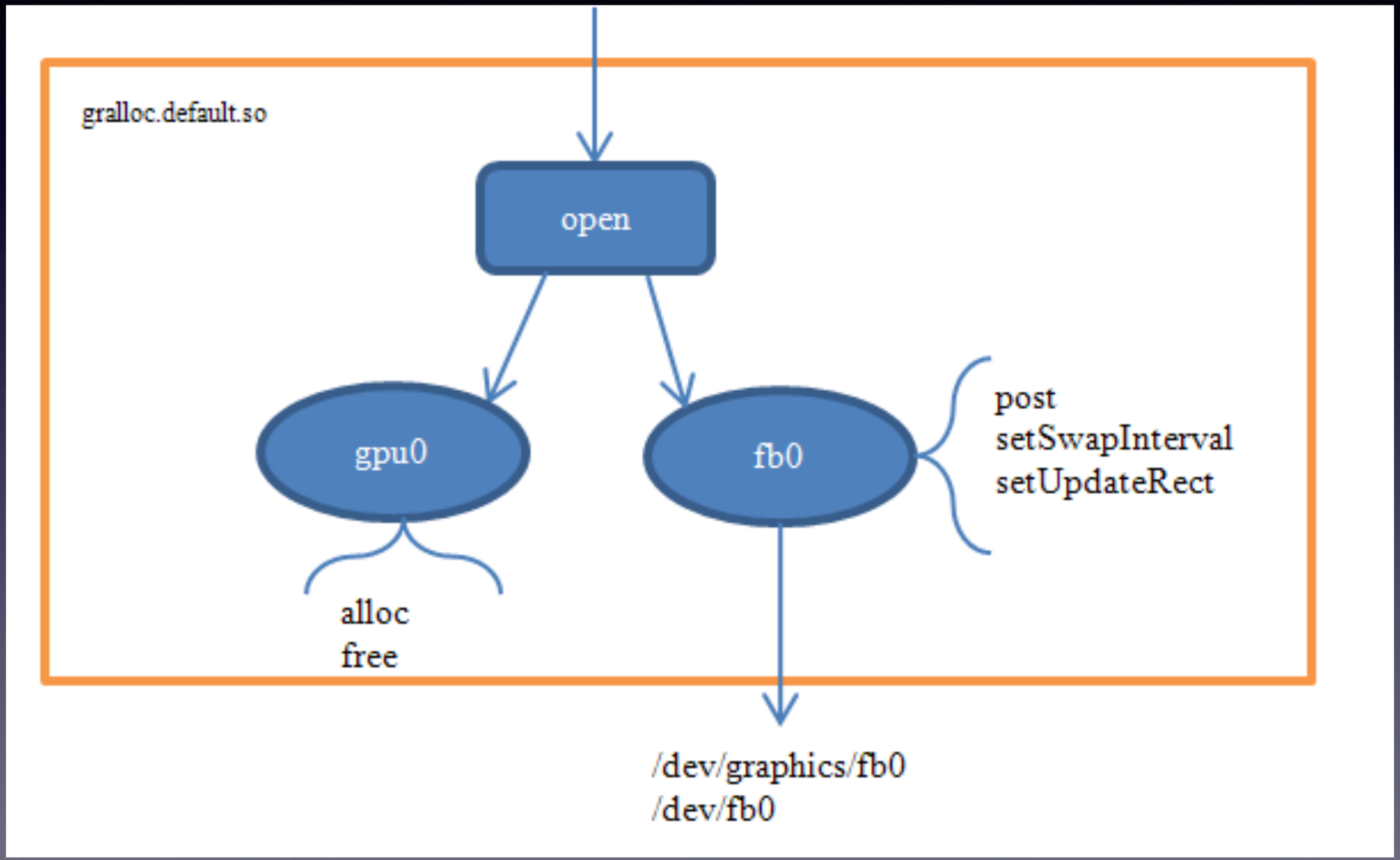
```
class BufferQueue : public BnGraphicBufferProducer,
                    public BnGraphicBufferConsumer,
                    private IBinder::DeathRecipient {
public:
    // for backward source compatibility
    typedef ::android::ConsumerListener ConsumerListener;

    class ProxyConsumerListener : public BnConsumerListener {
    public:
        ProxyConsumerListener(const wp<ConsumerListener>& consumerListener);
        virtual ~ProxyConsumerListener();
        virtual void onFrameAvailable();
        virtual void onBuffersReleased();
  private:
        wp<ConsumerListener> mConsumerListener;
  };
```

```
// mSlots is the array of buffer slots that must be mirrored on the
// producer side. This allows buffer ownership to be transferred between
// the producer and consumer without sending a GraphicBuffer over binder.
// The entire array is initialized to NULL at construction time, and
// buffers are allocated for a slot when requestBuffer is called with
// that slot's index.
BufferSlot mSlots[NUM_BUFFER_SLOTS];
```

BufferQueue

```
// BufferState represents the different states in which a buffer slot
// can be.  All slots are initially FREE.
enum BufferState {
    FREE = 0,

    DEQUEUED = 1,

    QUEUED = 2,

    ACQUIRED = 3
};

// mBufferState is the current state of this buffer slot.
BufferState mBufferState;
```

Gralloc


gralloc.default.so
open
gpu0
fb0
post
setSwapInterval
setUpdateRect
alloc
free
/dev/graphics/fb0
/dev/fb0

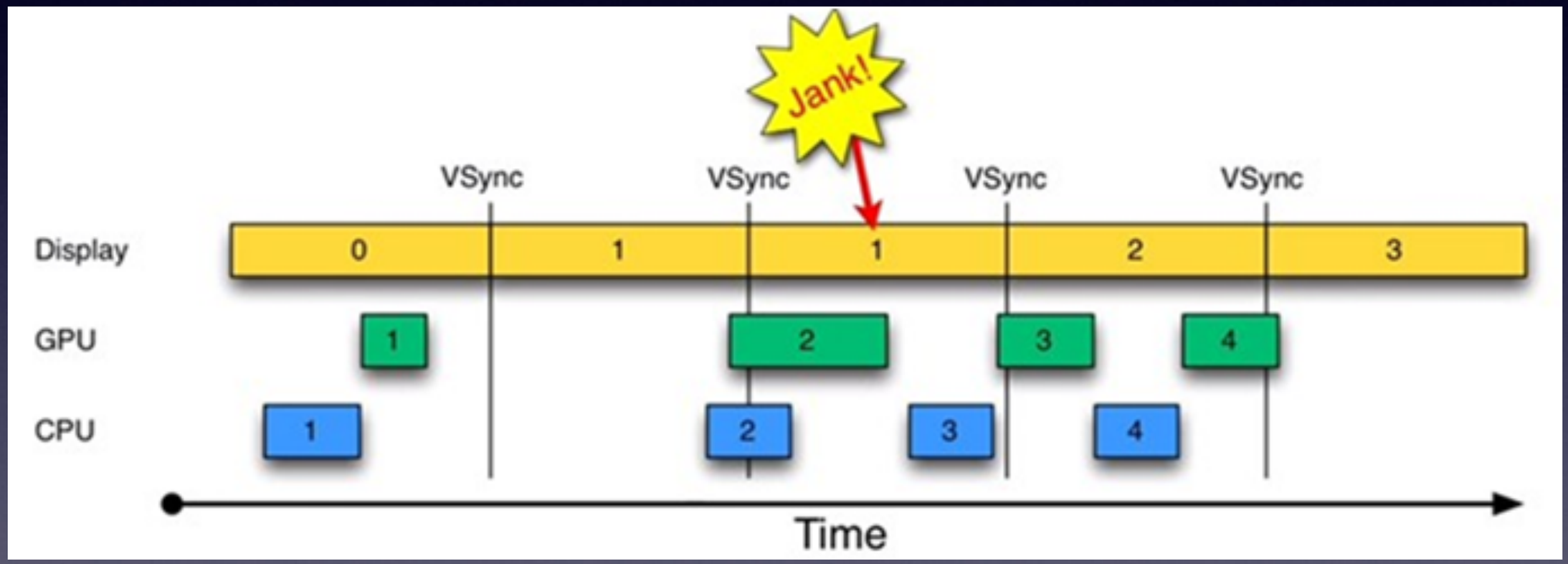
Gralloc

```
int gralloc_device_open(const hw_module_t* module, const char* name,
        hw_device_t** device)
{
    int status = -EINVAL;
    if (!strcmp(name, GRALLOC_HARDWARE_GPU0)) {
        gralloc_context_t *dev;
        dev = (gralloc_context_t*)malloc(sizeof(*dev));

        /* initialize our state here */
        memset(dev, 0, sizeof(*dev));

        /* initialize the procs */
        dev->device.common.tag = HARDWARE_DEVICE_TAG;
        dev->device.common.version = 0;
        dev->device.common.module = const_cast<hw_module_t*>(module);
        dev->device.common.close = gralloc_close;

        dev->device.alloc   = gralloc_alloc;
        dev->device.free    = gralloc_free;

        *device = &dev->device.common;
        status = 0;
    } else {
        status = fb_device_open(module, name, device);
    }
    return status;
```

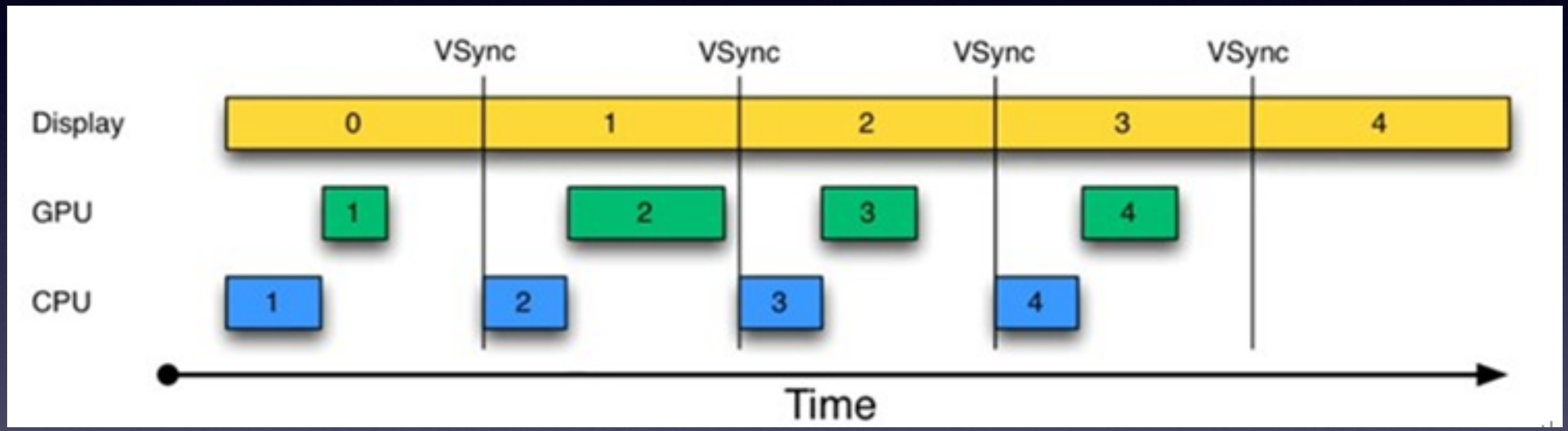
Gralloc

```
static int gralloc_alloc(alloc_device_t* dev,
        int w, int h, int format, int usage,
        buffer_handle_t* pHandle, int* pStride)
{
    int err;
    if (usage & GRALLOC_USAGE_HW_FB) {
        err = gralloc_alloc_framebuffer(dev, size, usage, pHandle);
    } else {
        err = gralloc_alloc_buffer(dev, size, usage, pHandle);
    }

    if (err < 0) {
        return err;
    }

    *pStride = stride;
    return 0;
}
```

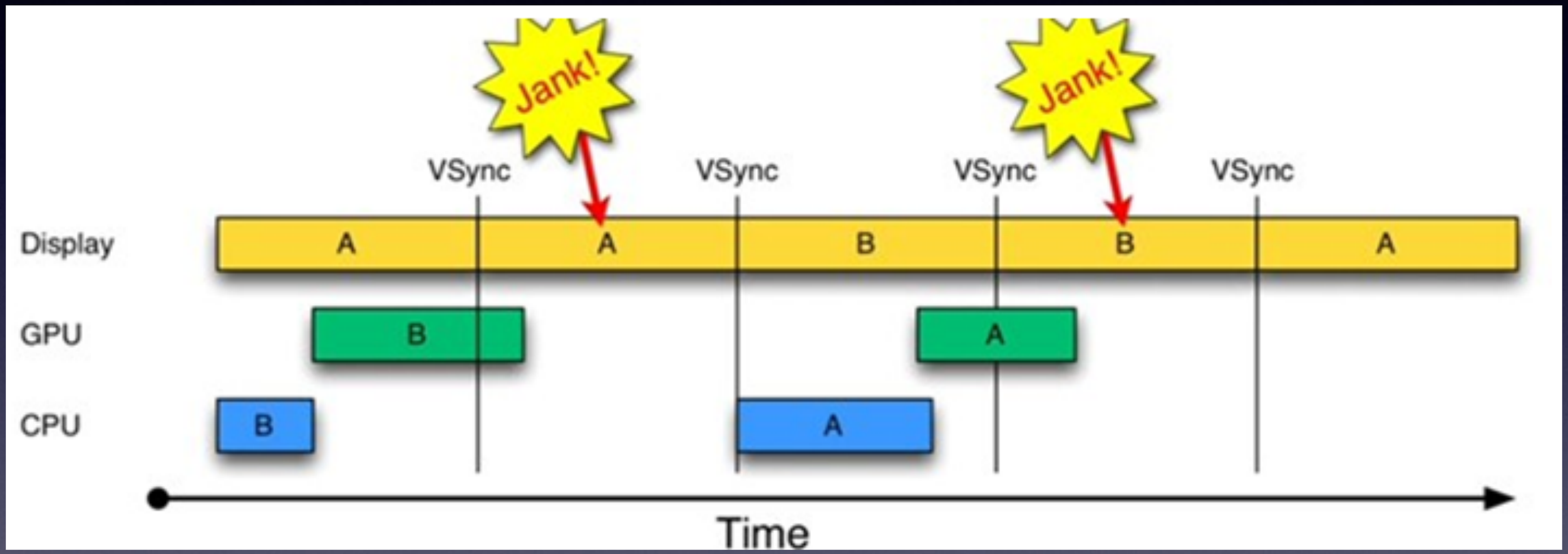
VSync


Jank!
VSync
VSync
VSync
VSync
Display
0
1
1
2
3
GPU
1
2
3
4
CPU
1
2
3
4
Time

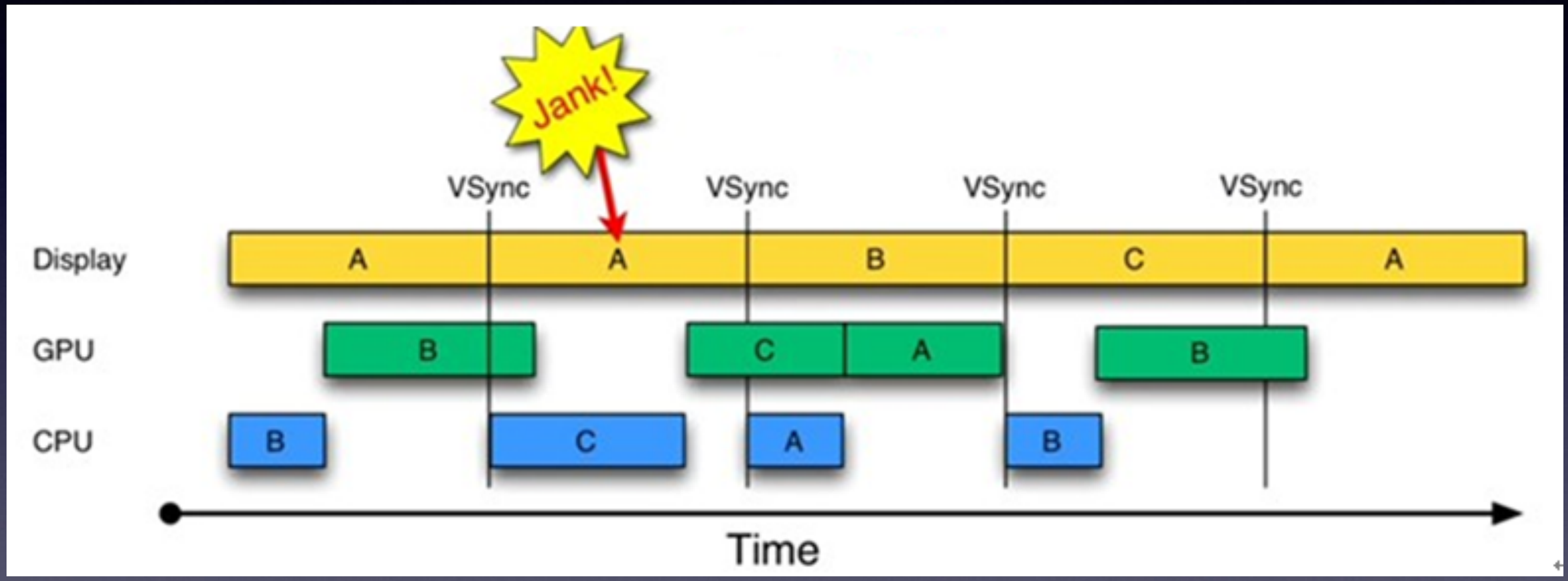
VSync


VSync
VSync
VSync
VSync
Display
0
1
2
3
4
GPU
1
2
3
4
CPU
1
2
3
4
Time

VSync


Jank!
Jank!
VSync
VSync
VSync
VSync
Display
A
A
B
B
A
GPU
B
A
CPU
B
A
Time

VSync

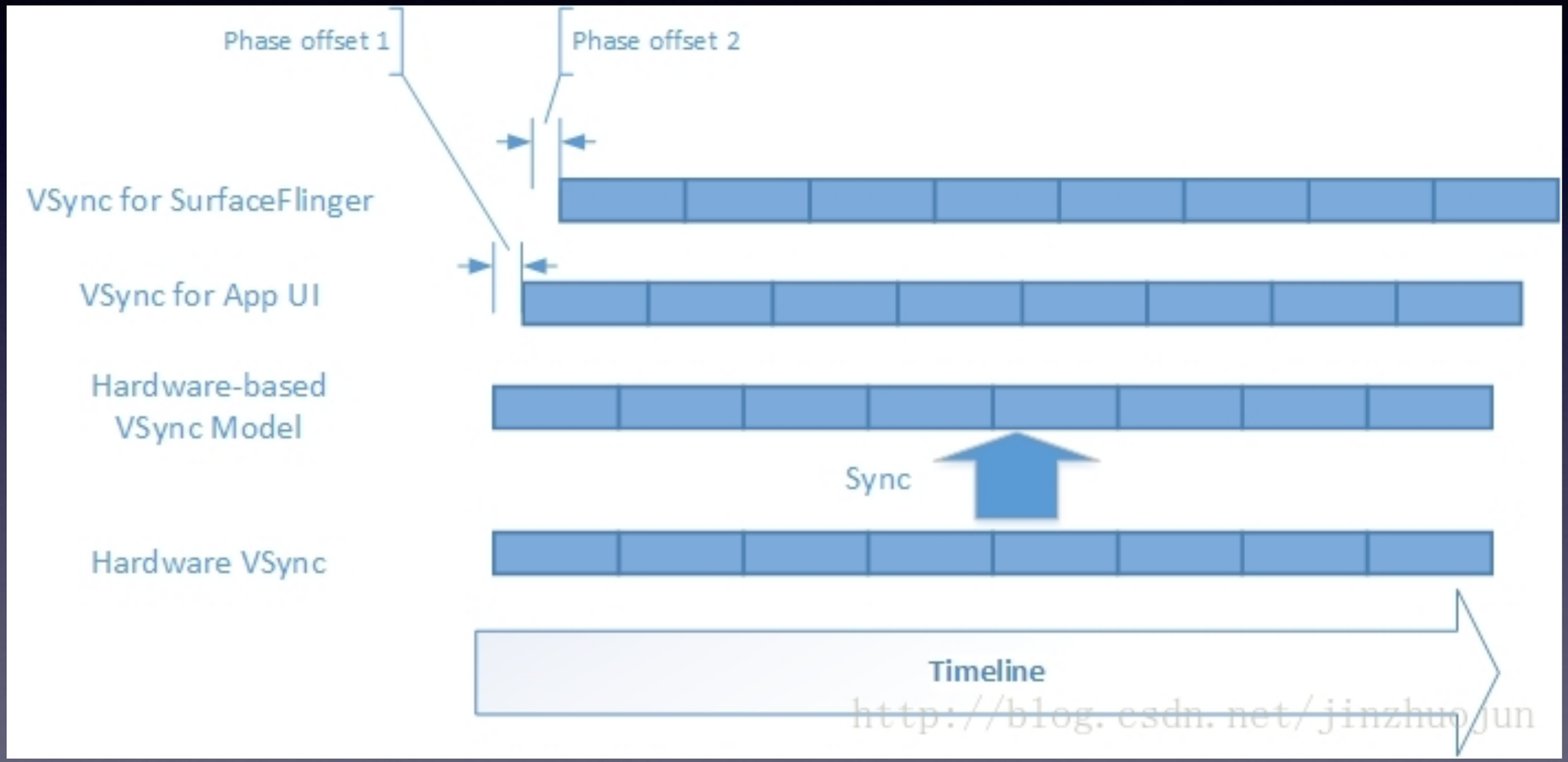

Jank!
VSync
VSync
VSync
VSync
Display
A
A
B
C
A
GPU
B
C
A
B
CPU
B
C
A
B
Time

VSync

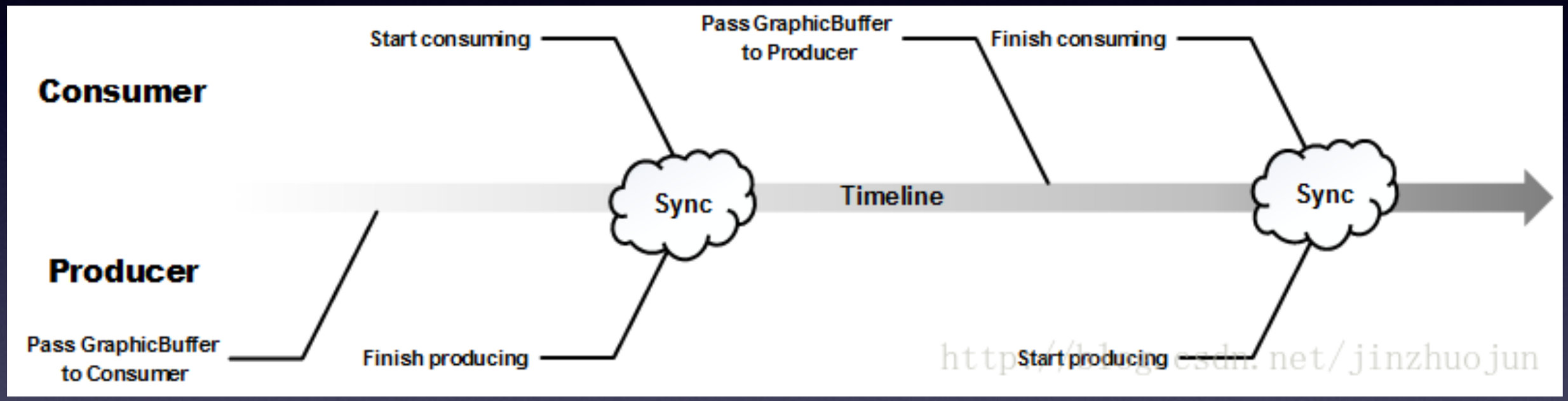
如何产生

```
HWComposer::HWComposer(
        const sp<SurfaceFlinger>& flinger,
        EventHandler& handler) {
    ......

    if (mHwc) {
        if (mHwc->registerProcs) {
            mCBContext->hwc = this;
            mCBContext->procs.invalidate = &hook_invalidate;
            mCBContext->procs.vsync = &hook_vsync;

            mHwc->registerProcs(mHwc, &mCBContext->procs);
        }

        // don't need a vsync thread if we have a hardware composer
        needVSyncThread = false;

    }

    ....

    if (needVSyncThread) {
        // we don't have VSYNC support, we need to fake it
        mVSyncThread = new VSyncThread(*this);
    }
}
```

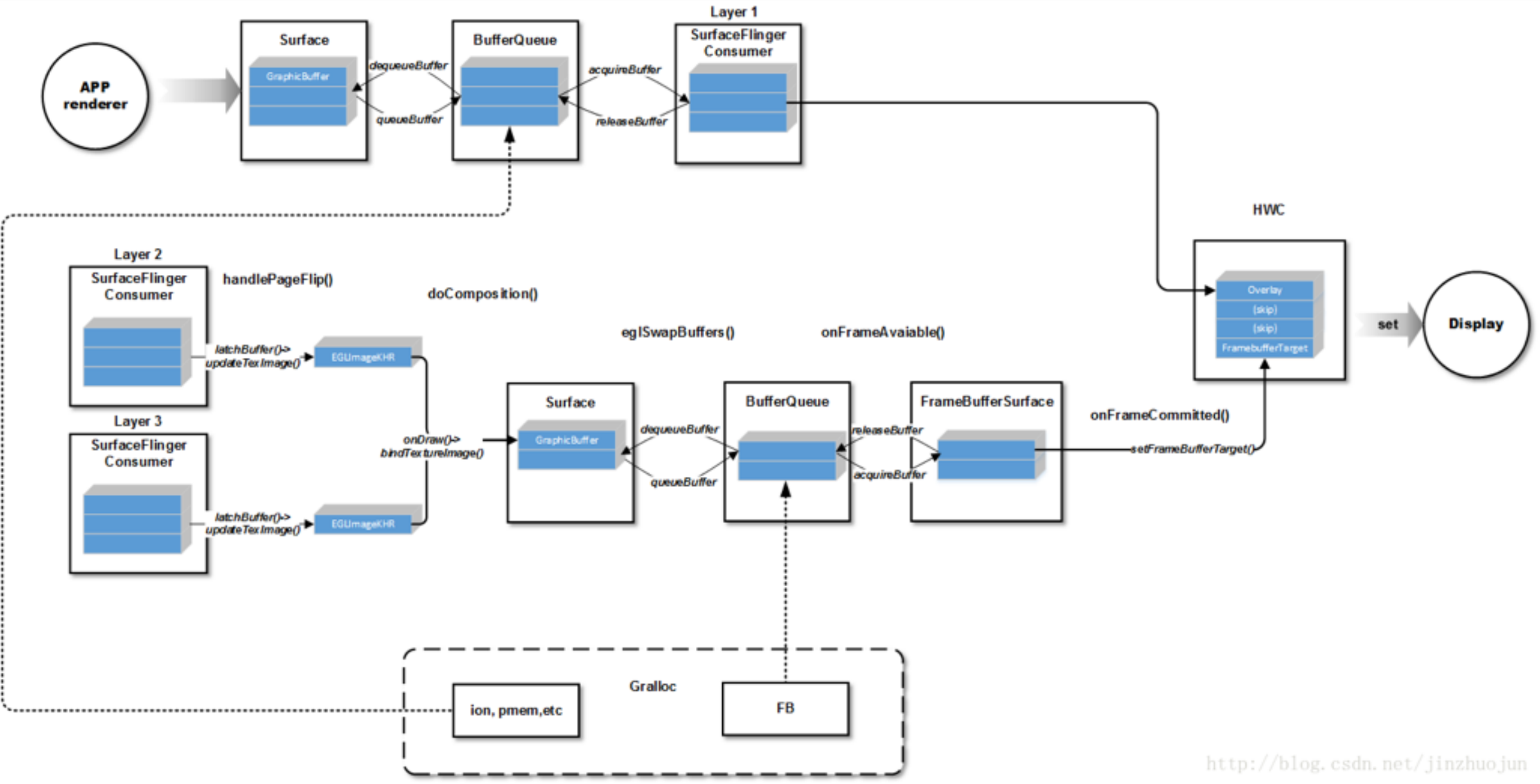
VSync


Phase offset 1
Phase offset 2
VSync for SurfaceFlinger
VSync for App UI
Hardware-based
VSync Model
Sync
Hardware VSync
Timeline
http://blog.csdn.net/jinzhuojun

Fence



最终流程



Q & A