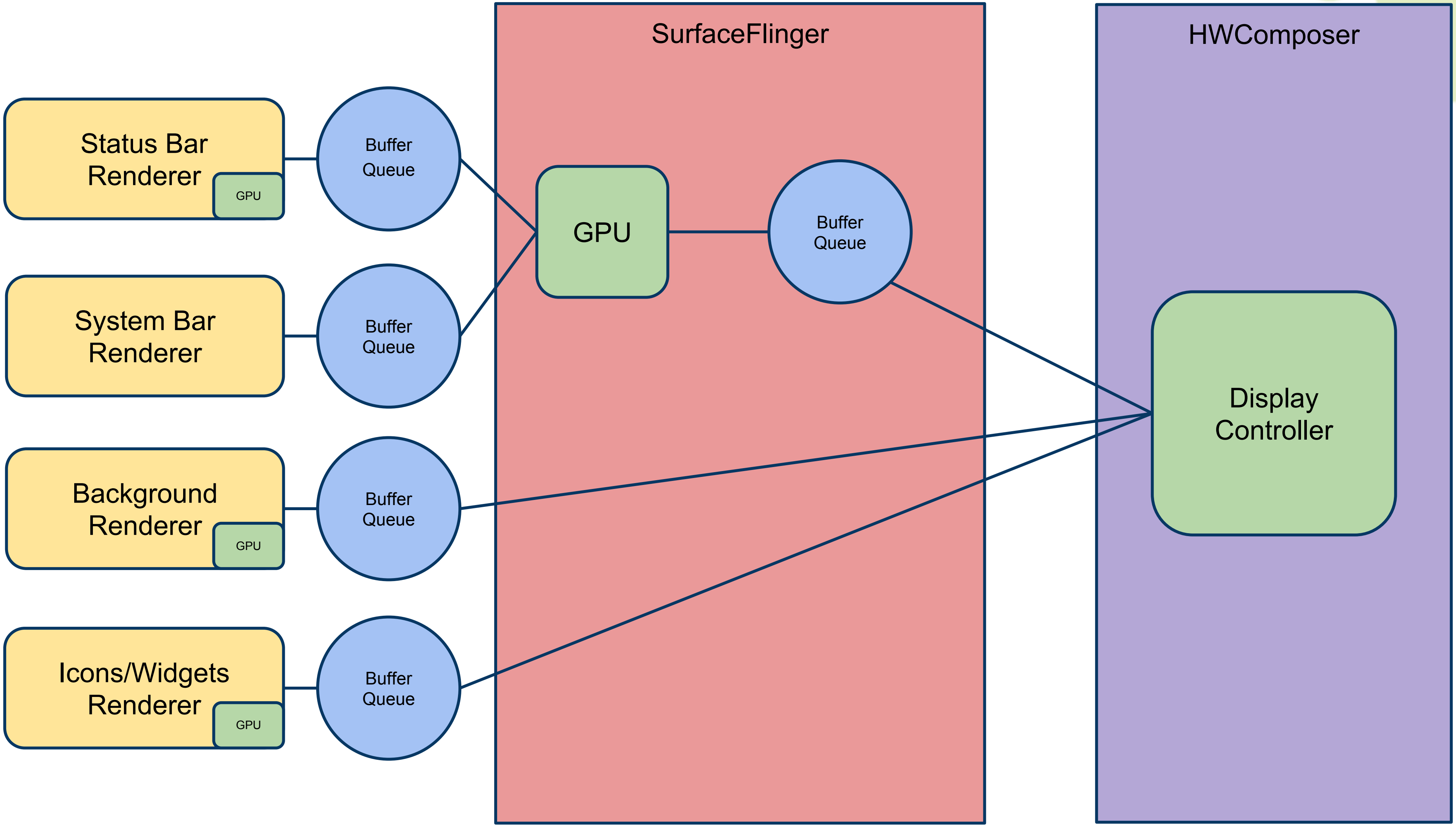
Android Sync

Riley Andrews

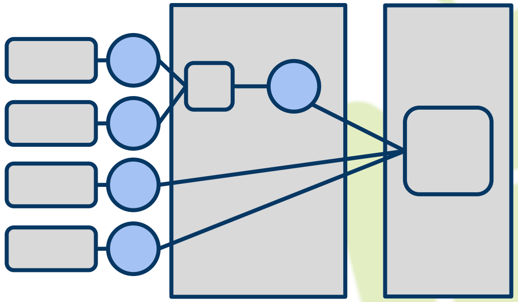
Adapted from slides by:
Erik Gilling & Jamie Gennis

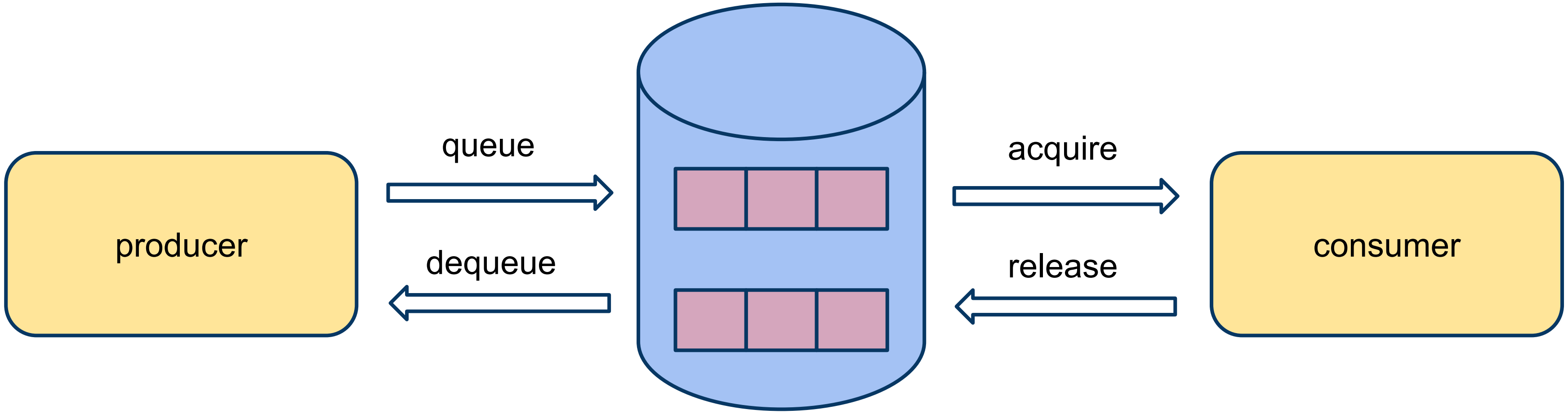
Android Graphics Pipeline Overview

Sync - Android Graphics Pipeline



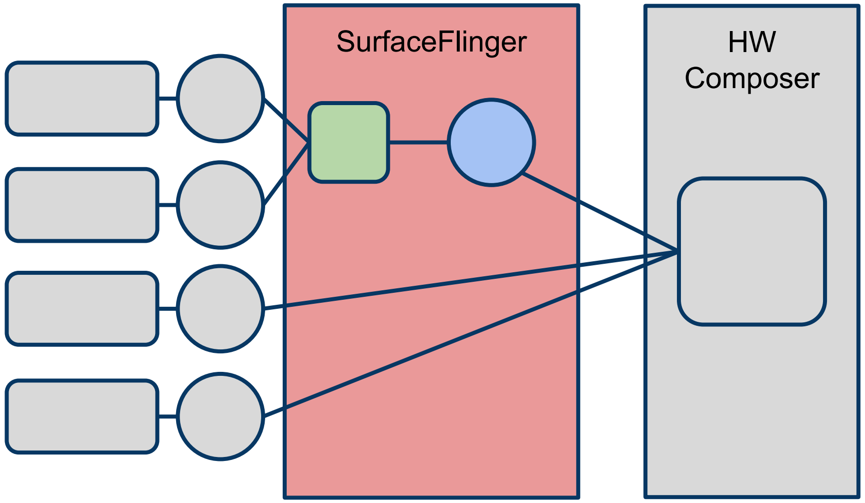
Sync - BufferQueue





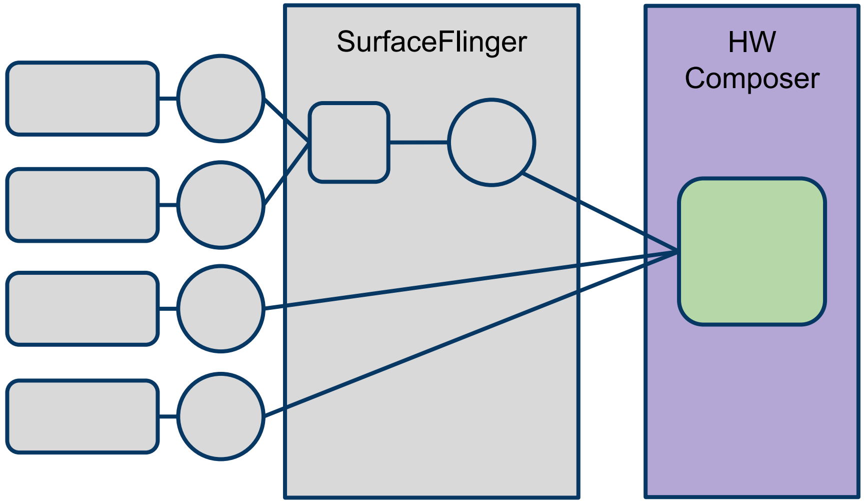
- manages flow of buffers between producers and consumers
- two queues
- producers dequeue unused buffers, fill them, the queue them
- consumers acquire filled buffers, use them, then release them when done

Sync - SurfaceFlinger



- Responsible for compositing all windows into the display(s)
- Just another GL client

Sync - HW Composer



- Started as a HAL for accelerating composition
- Becoming the HAL for all things display

Sync - Looks Great! What's Broken?

- No explicit parallelism

- Every vendor implements implicit synchronization.

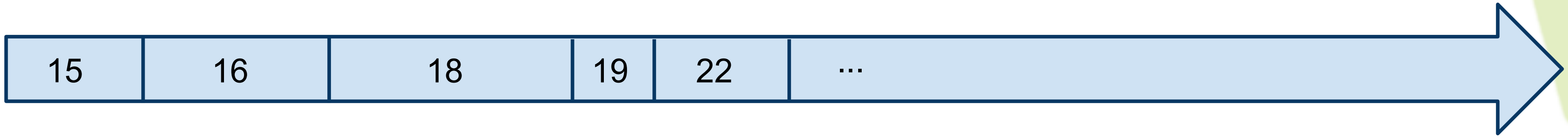
- Historically this has been the source of many hard to debug graphics pipeline lock ups.

Sync - Framework Goals

- Provide a simple API to let components signal when buffers are ready/released.
- Allow synchronization primitives to be passed between processes and between userspace and the kernel.
- Allow implementers to exploit hardware sync support
- Provide visibility into the graphics pipeline for debugging

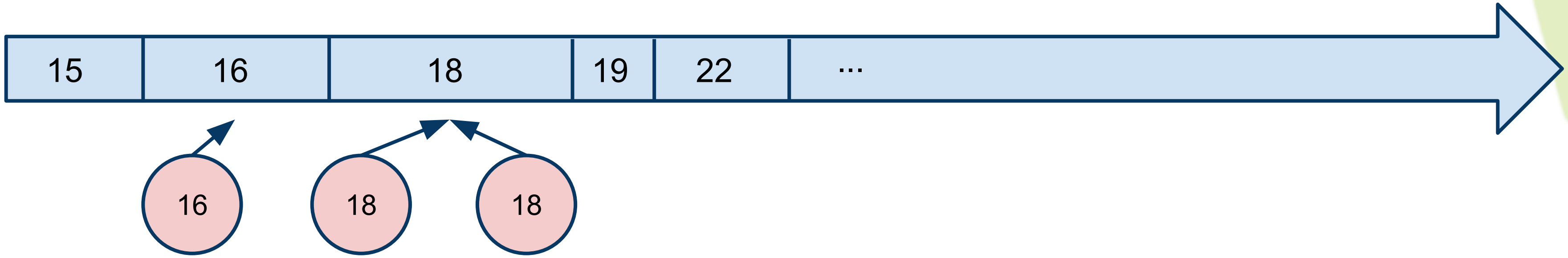
Kernel Sync Building Blocks

Sync - sync_timeline



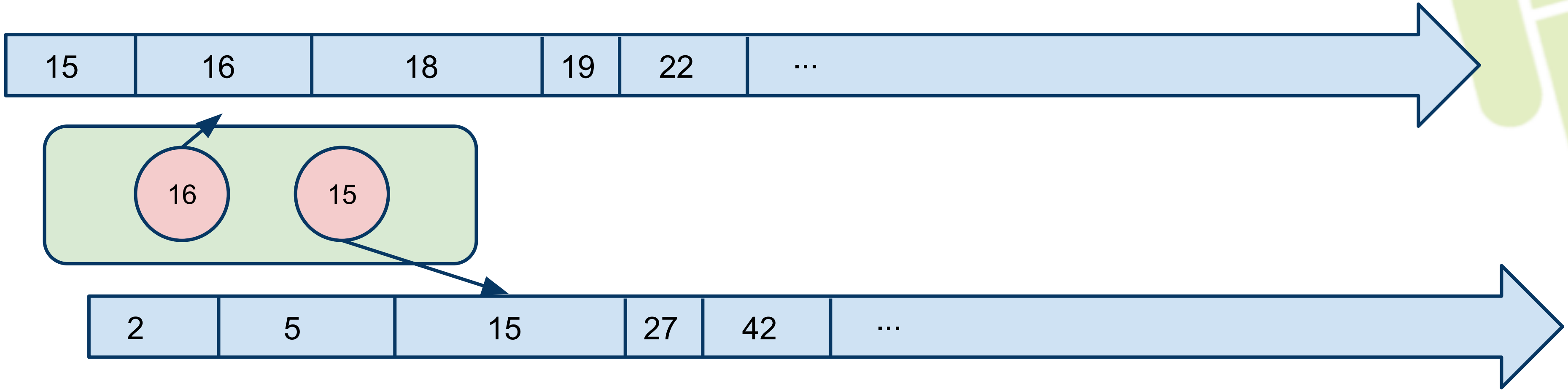
- Represents monotonically increasing counter.
- Generally one instance per driver context
- allows hardware specific implementation
- sw_sync implementation provided

Sync - sync_pt



- Represents a specific values on a parent timeline
- 3 states
  - active
  - signaled
  - error
- starts active and transitions once to either signaled or error

Sync - sync_fence

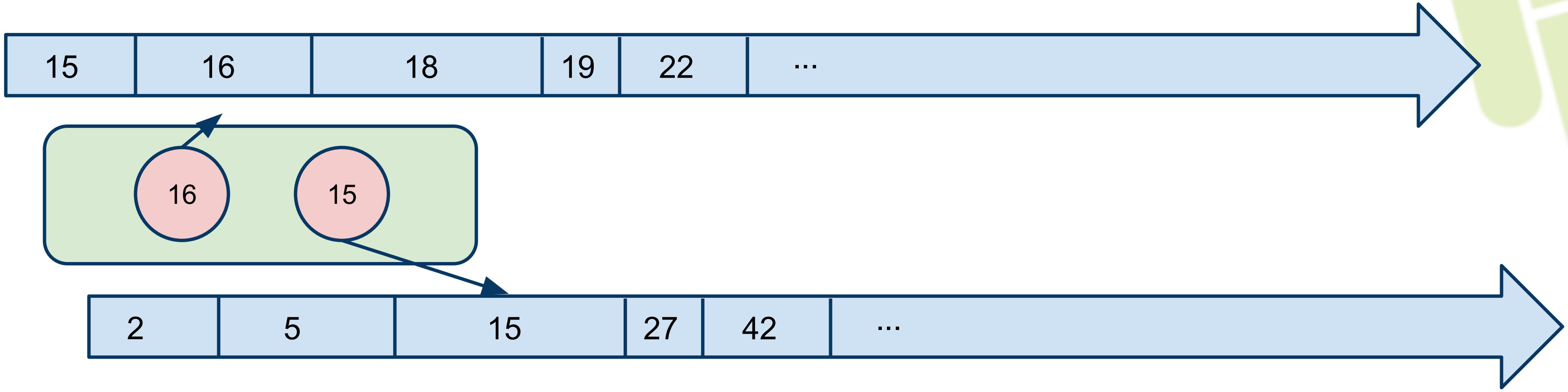


- A collection of sync_pts

- Backed by a file and can be passed to userspace.

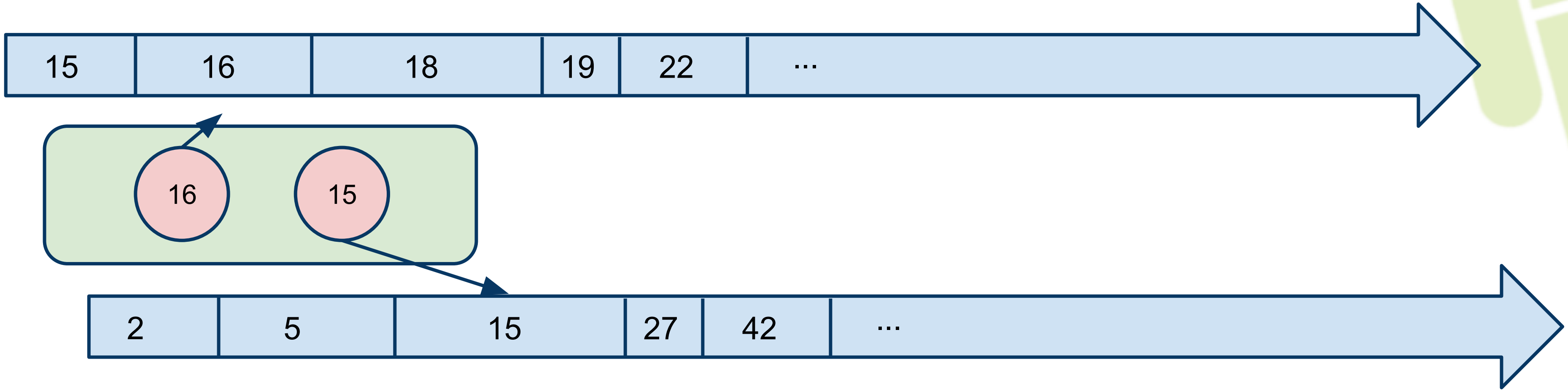
- Main primitive drivers and userspace use to describe sync events/dependencies.

Sync - sync_fence (the promise)



- Fences are a promise by the kernel
    - that work has been queued
    - and will complete in a "timely" manner

Sync - sync_fence (more details)



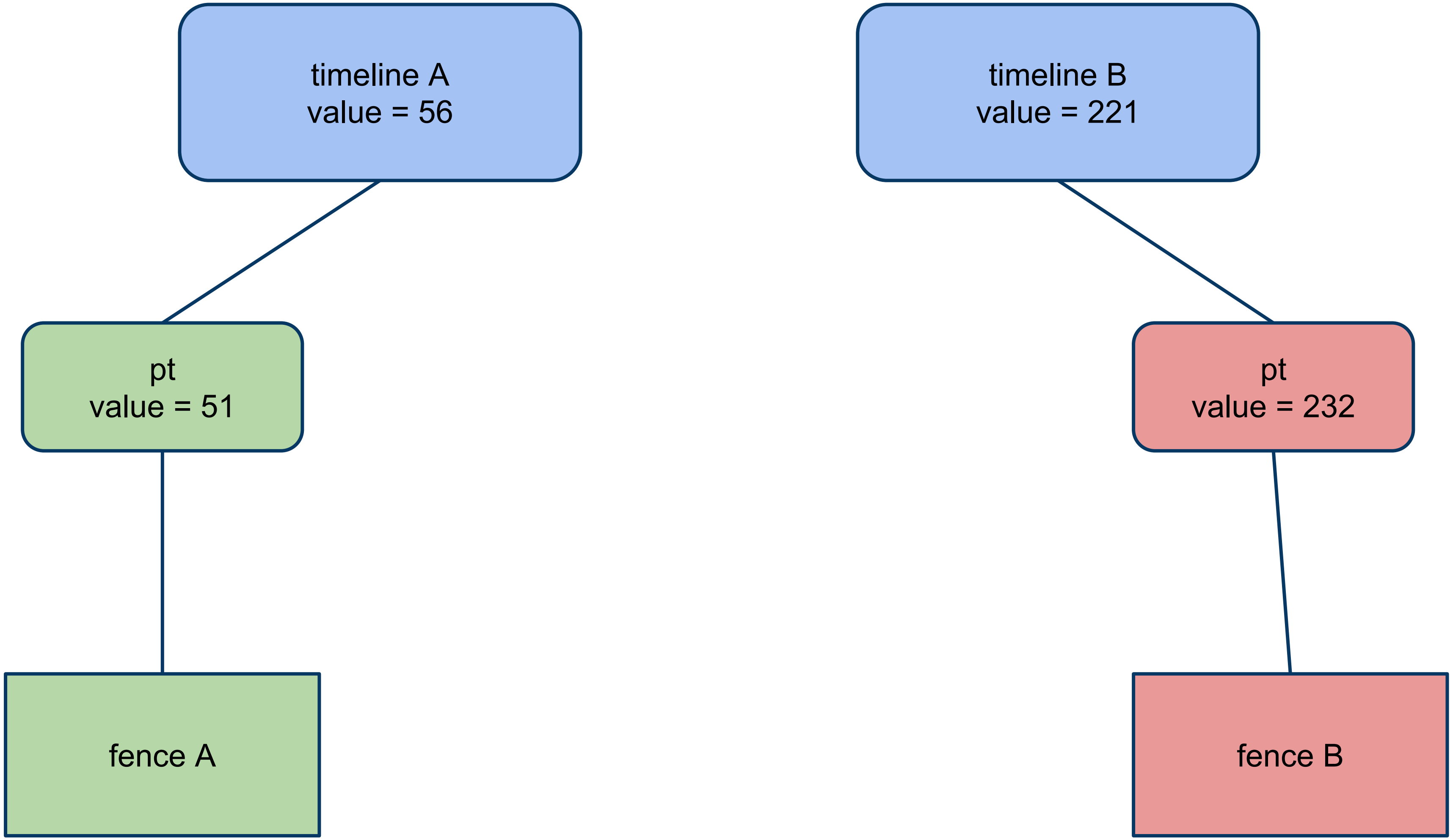
- Starts active and transitions to signaled with all of its sync_pts become signaled or one becomes errored

- The list of sync_pts is immutable after fence creation

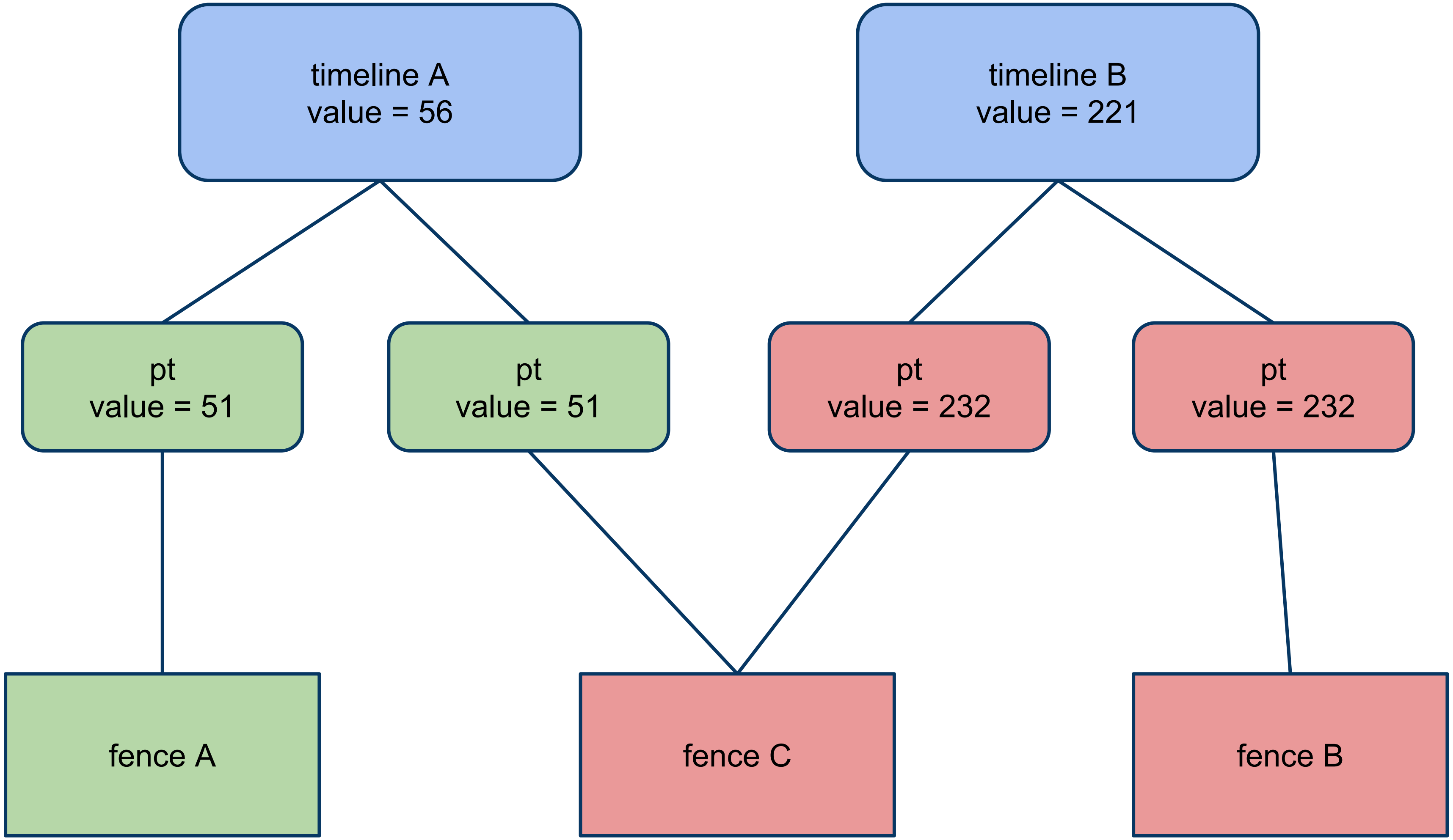
- A sync_pt can only be in one fence.

- Two fences can be merged to create a third fence containing copies of the sync points in both.

Sync - Before Merge



Sync - After Merge



Implementing Sync

Sync - Core Implementation

- supported in android-3.10 kernel + staged for quite some time

- Core
  - drivers/staging/android/sync.c
  - drivers/staging/android/sync.h

- sw_sync
  - drivers/staging/android/sw_sync.c
  - drivers/staging/android/sw_sync.h

- Docs
  - Documentation/sync.txt

Sync - Imaginary Display Driver

Before Sync:

```
/*
 * assumes buf is ready to be displayed.
 * returns when buffer is no longer on screen.
 */
void display_buffer(struct dma_buf *buf);
```

After Sync:

```
/*
 * will display buf when fence is signaled.
 * returns immediately with a fence that will signal when buf
 * is no longer displayed.
 */
struct sync_fence* display_buffer(struct dma_buf *buf,
                                  struct sync_fence *fence);
```

Sync - Implementing a sync_timeline

- Don't. Try using sw_sync first.

- Use sw_sync as a starting point.

- Don'ts
  - Don't base a timeline on any "real" time.
  - Don't allow userspace to explicitly
    - create a fence
    - signal a fence
  - Don't access sync_timeline, sync_pt, or sync_fence elements explicitly

Sync - Implementing a sync_timeline (cont.)

- Dos
  - Do provide useful names
  - Do implement timeline_value str and pt_value_str
  - Do implement fill driver_data

Sync Integration

Sync - OpenGL ES Integration

- EGL_ANDROID_native_fence_sync
  - Wrap an Android fence fd in an EGLSyncKHR
  - Create an Android fence fd from an EGLSyncKHR
- EGL_ANDROID_wait_sync
  - Essentially the same as EGL_KHR_wait_sync
  - Make the GPU wait for an EGLSyncKHR

Sync - EGL_ANDROID_native_fence_sync

- New "native fence" EGLSync object type
- New "native fence fd" attribute
  - Can be set at creation time to either a valid fence fd or -1
  - Can not be queried from an existing sync object
- New DupNativeFenceFD function
  - Returns a dup of the "native fence fd" attribute
- Destroying the EGLSync closes the fence fd

Sync - Advantages of Explicit Sync

- Less behavior variation between devices
- Better debugging support
- Upcoming jank metrics
  - SurfaceFlinger presentation timestamps
  - Flatland GPU benchmark

Dma Fence – Upstream graphics synchronization

- Upstream solution for cross device synchronization
  - In for 3.17
  - Needed to support optimus hardware (?)

- what are dma fences for?
  - unified interface for cross driver synchronization
  - used for tracking work on a dma buf

Dma Fence – compared to sync

- one shot fences (active -> completed)
- supports timeline-esqe sequences number based fences
- support HW device to device sync (e.g. nv semaphores)
- synchronous waits:
  - dma fence: sync_fence_wait()
  - sync: sync_fence_wait()
- asynchronous callbacks
  - dma fence: fence_add_callback()
  - sync: sync_fence_wait_async()

Dma Fence – contrast with sync

- Fences are attached to dma buf directly.
  - No userspace sync objects!
  - Update dma fences based on read/write access to buffers on pushbuffer submit.
- No merging of dma fences, just track lots of them.
- No timelines, no sync points.

Sync – no more

- Maarten Lankhorst has implemented sync on dma-fence!
  - Each sync point is implemented with a dma-fence callback.
  - Merging is handled by adding a “context id” to each dma-fence, so that fences can be compared

Questions –

- Is there a need for explicit sync? Do we need both?
  - Performance of bindless/compute
  - Making performance w/suballocation fast
- How sync be de-staged, and work alongside dma fence?