Chrome Compositor and Display Scheduling

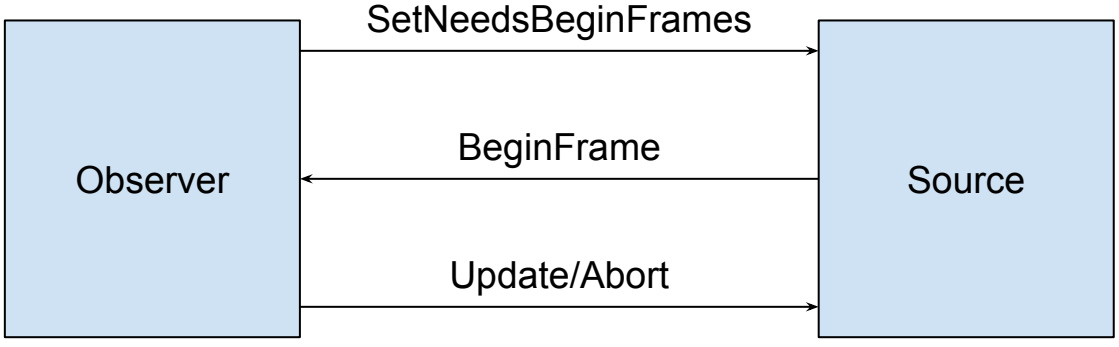
October 6, 2015

brianderson

Overview

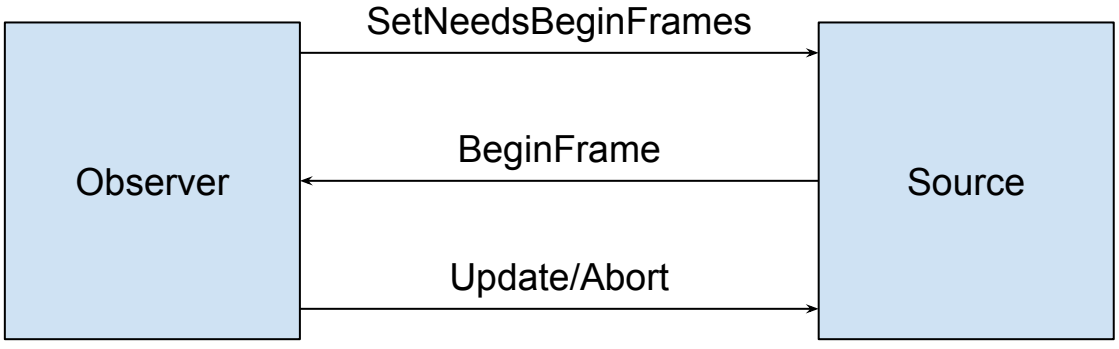
- BeginFrame Source and Observer pattern
- Pipeline Management
- Compositor Scheduler
- Surfaces and DisplayScheduler

BeginFrame



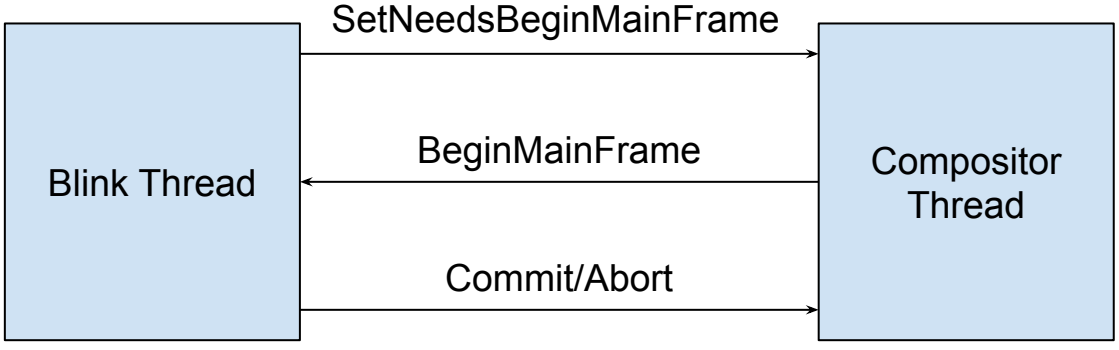
- The BeginFrame marks the start of a frame for a given pipeline stage.
- The Observer calls SetNeedsBeginFrames on the Source to indicate whether it is active or not.
- The Source triggers the BeginFrame and the Observer responds with an Update or Abort.

BeginFrame



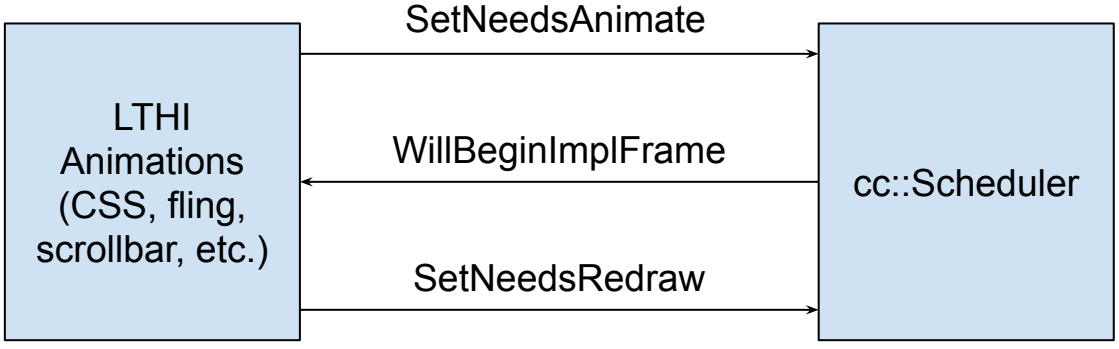
- **BeginFrame has:** frame time, deadline, interval, critical path, user input.
- **Observers include:** Display, UI, Renderer, CSS, Video, Blink.
- **Sources include:** External, DelayBased, BackToBack, WebView, Custom
- tansell@ formalized the class structure for many of the source types.

Blink & CC



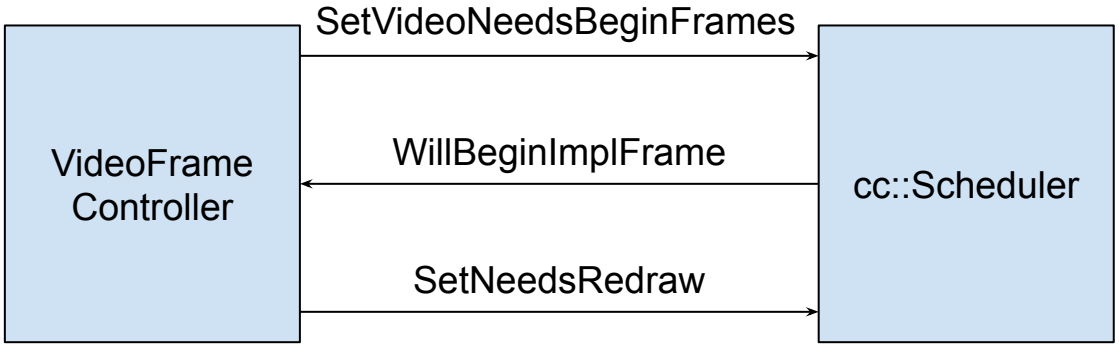
- Critical path signal used to prioritize the response. Set by compositor when it’s waiting for a commit to draw.
- Blink is the only source to explicitly abort its response currently. (enne@)
- Compositor sends an additional signal called BeginMainFrameNotExpectedSoon for long idle periods.
- skyostil@, alexclarke@, rmcilroy@

Animations + CC



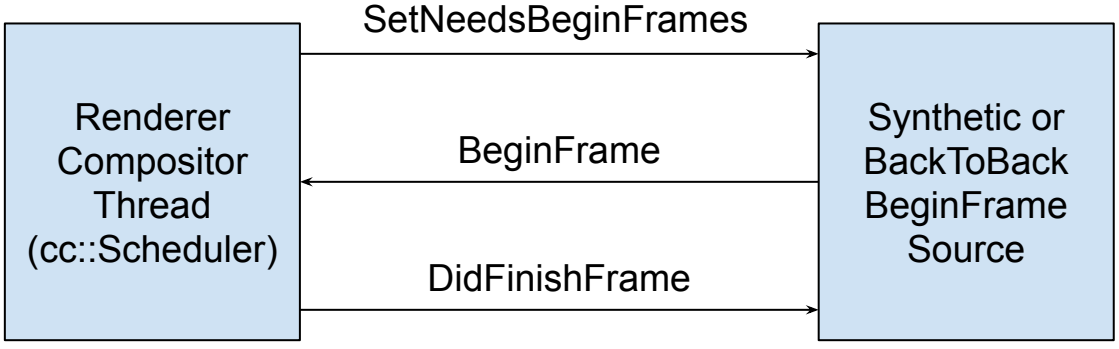
- Abort is implied if SetNeedsRedraw is not called.
- tansell@
  - Fixed all animations so they are hooked up to a single source.
  - Fixed all BeginFrame sources so timestamps are monotonic.
  - Removed background ticking.

Video + CC



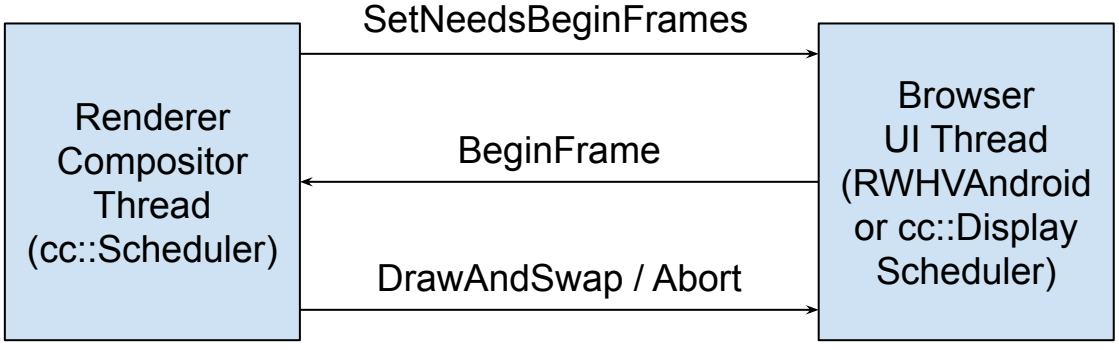
- sunnyps@ and dalecurtis@ worked to define and implement the interface as part of Project Butter.
- VideoFrameController talks to the decoder thread to choose a frame using provided frame interval and a hybrid coverage+cadance algorithm.
- Previously, frame was chosen at draw time rather than BeginFrame time and the video decoder ran an independent clock of its own.
- Corner cases: pause/resume, background video, WebGL.

CC & “Self ticking”



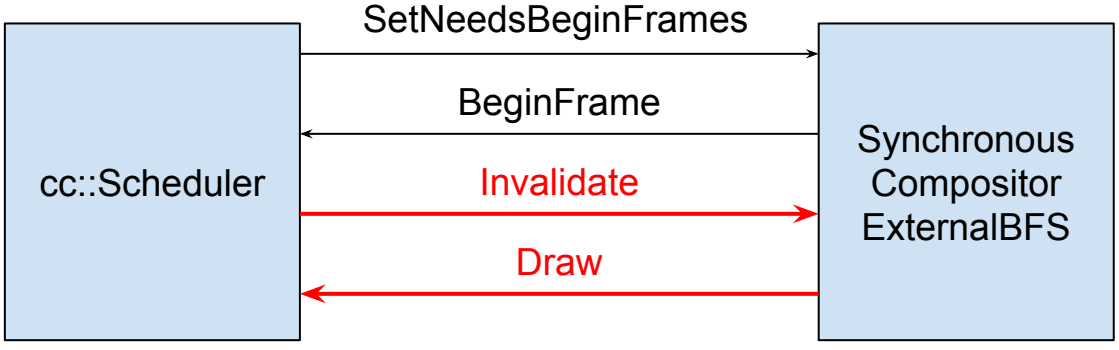
- This is the configuration used on most platforms today (other than Android) and for when --disable-vsync is used.
- The display refresh rate and period is distributed to all Synthetic or BackToBack BeginFrame sources so that they can self tick.
- Pro: There are no thread hops for the BeginFrame to make.
- Con: Difficult to guarantee order with input system and have the Browser provide feedback.

CC & Browser



- Currently a work in progress. AKA: Unified BeginFrame.
- Android has had this configuration for ~2 years.
- simonhong@ did much of the initial work, but changes are needed now that Surfaces have been enabled on all platforms.
- See tansell@'s design doc and later slides for new plan.

CC & WebView

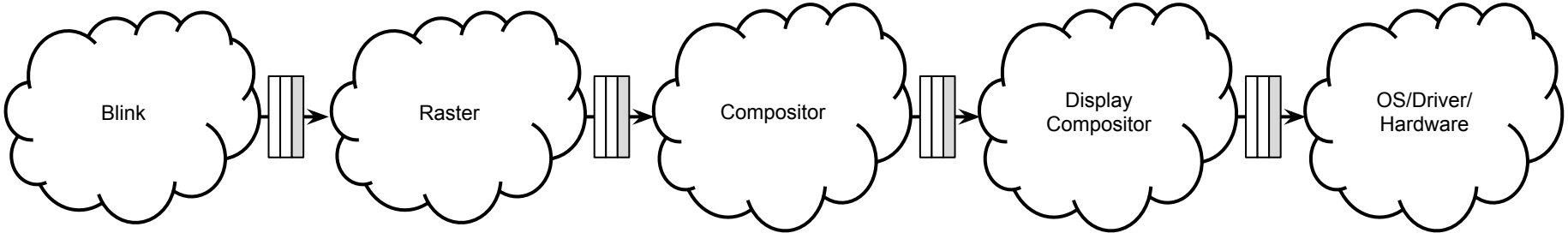


- WebView decides when to Draw and the compositor only indicates it wishes to draw through an Invalidate signal.
- Additionally, WebView can decide to draw at anytime, multiple times per frame, during an otherwise idle period, or not at all!
- Previously, SNBF implied Invalidate, which was inefficient and unreliable.
- sunnyps@ and boliu@ ironed out details in issue 439275.

BeginFrame: Questions?

Next Up…
Pipeline Management

Pipeline Management

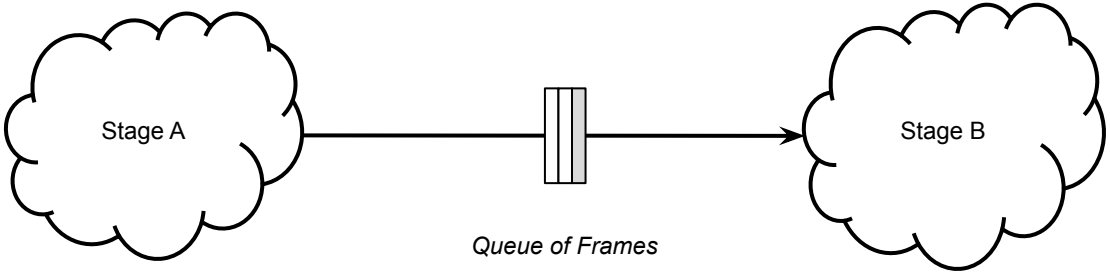


- Chrome has a lot of async nested BeginFrame Observers!
- Raster isn't normally nested as depicted, but can be.
  - --enable-main-frame-before-activation
- Not depicted:
  - Multiple producers join at the Display.
  - Interaction with the GPU pipeline that runs in parallel.
  - PAPPI

Pipeline Management Goals / Metrics

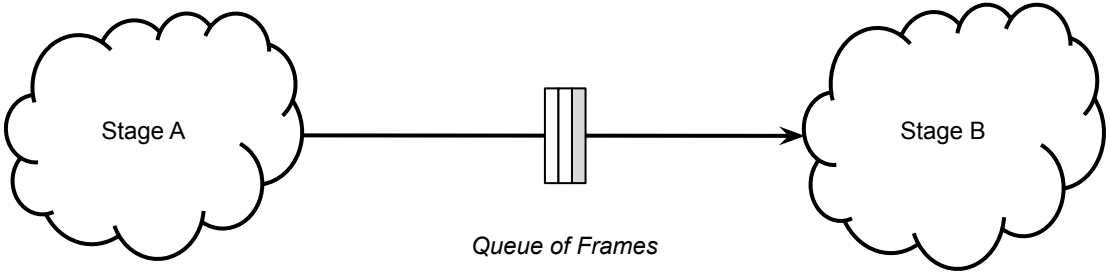
- Latency
- Throughput
- Smoothness
- Synchronization
- Memory

Low vs. High Latency Modes

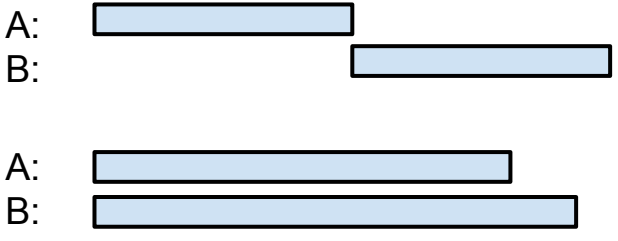


- Stage A is in a **low latency mode** if the queue is empty on Stage B's BeginFrame.
- Stage A is in a **high latency mode** if the queue has a frame on Stage B's BeginFrame.

Latency vs. Throughput

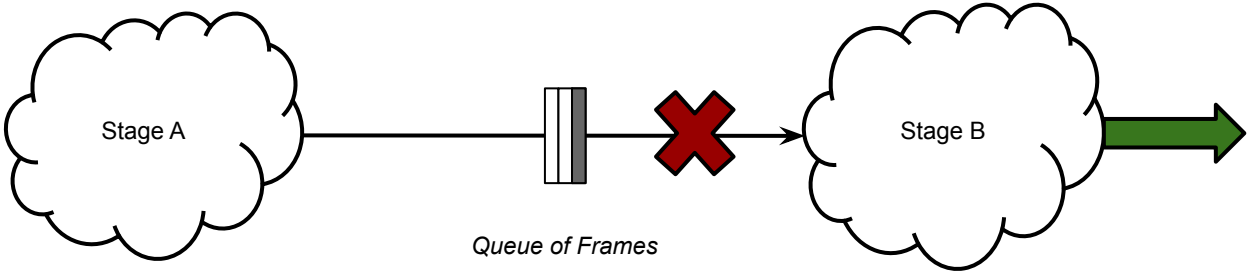


- In a **low latency mode,** stage A and B must run serially.
- In a **high latency mode,** stage A and B can run concurrently.



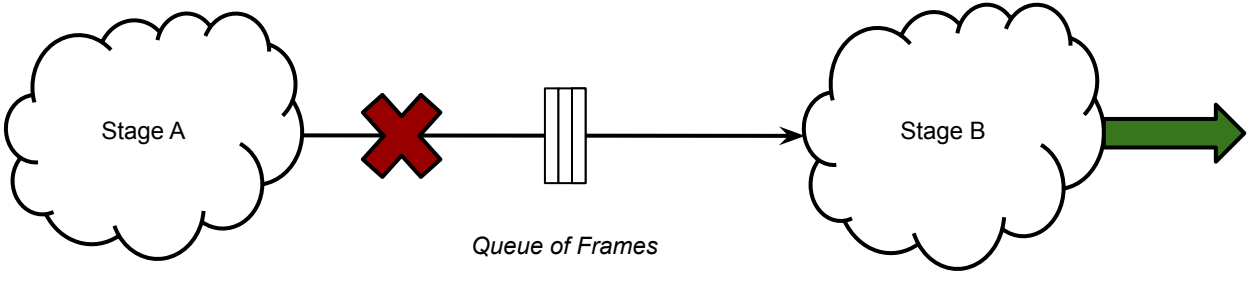
Problem: Concurrency doesn't necessarily help throughput if we are already CPU bound.

High Latency Mode



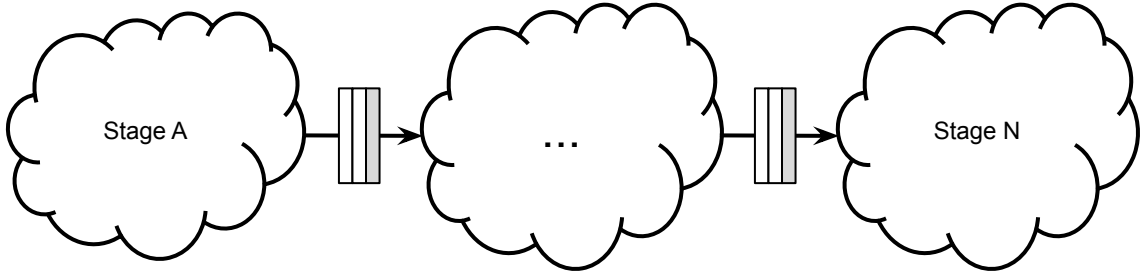
- Stage A will enter a high latency mode if B produces a frame without consuming a new frame from A. (Throughput jank.)
- This usually occurs when Stage A takes too long, missing B's **deadline**.
- If Stage B can't trust stage A, high latency mode is inevitable.
- Synchronization of stages is difficult in high latency mode.

Latency Recovery



- **Problem:** Once stage A enters a high latency mode, it will stay there.
- **Solution:** Skip a frame of production in stage A.
- **New problem:** Latency jank!

Latency and Memory



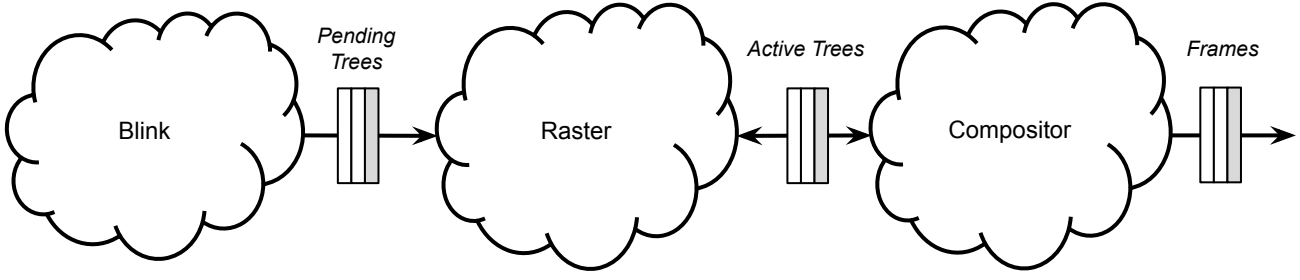
- Let P be the number of stages in a high latency mode between A and N.
- There must be at least P+1 buffers to avoid write-after-read hazards.

There are ways to reduce the number of framebuffers needed with command deferral, but it gets tricky.

Pipeline Management: Questions?

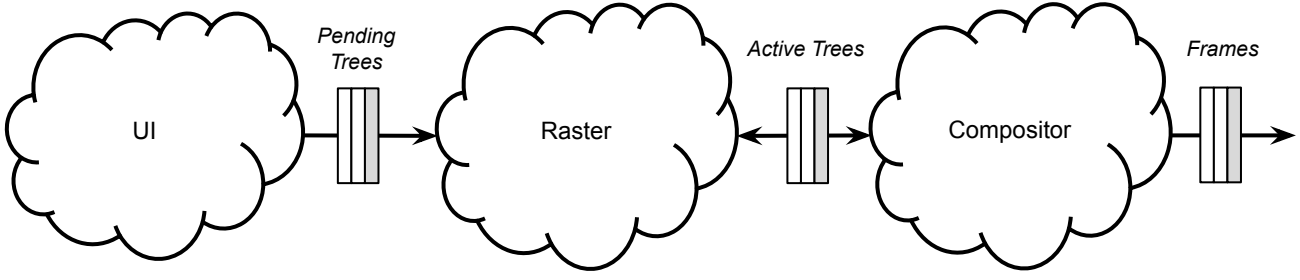
Next Up…
Compositor Scheduler

Compositor Pipeline: Renderer



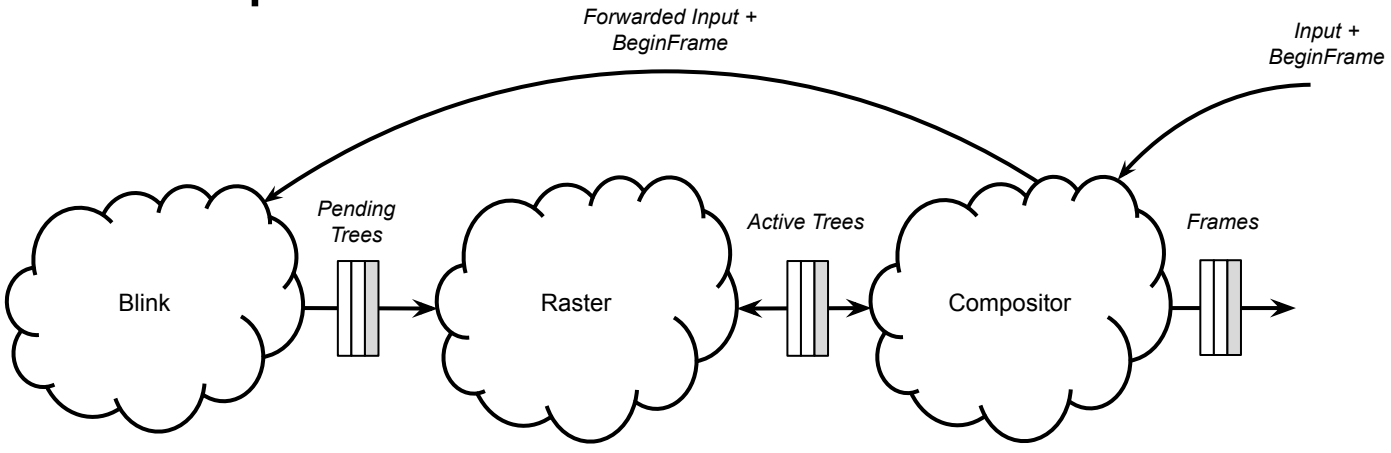
- All platforms limit the size of every queue to 1 frame.
- Blink is allowed to run in a high latency mode relative to the Compositor.
- We recover latency between Blink and Compositor if possible.
- Blink is forced into a low latency mode with Raster.

Compositor Pipeline: UI



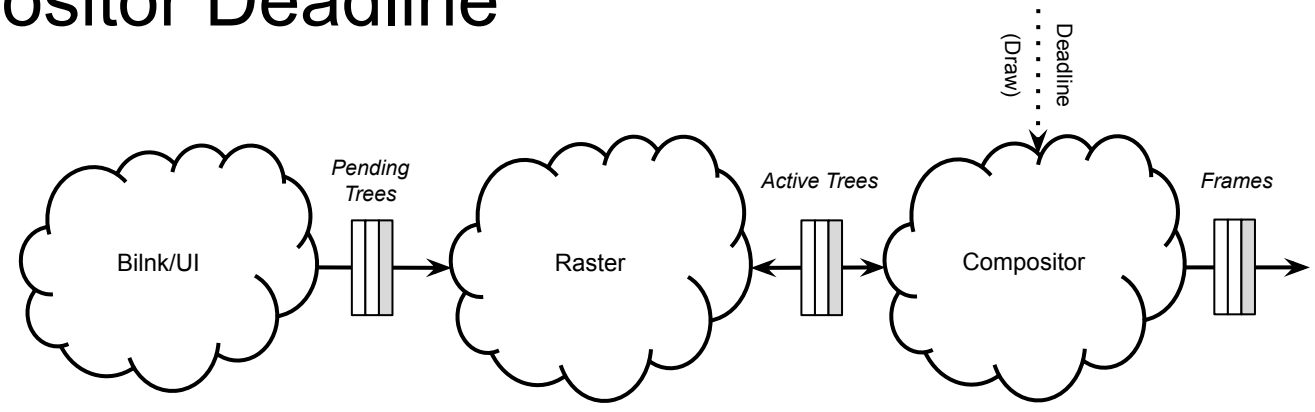
- Also uses cc::Scheduler with SingleThreadProxy (@enne, @weiliangc)
- UI is forced into a low latency mode with Compositor. (weiliangc@)
- Commit goes directly to the active tree. (danakj@)
- Non-impl-side code path deprecated and removed!

Renderer + Input



- The BeginFrame messages effectively split user input into vsync intervals.
- High latency modes introduce synchronization issues when input causes side effects in both Blink and the Compositor.
- Solutions: Best effort vs. Sync Scroll vs. CompositorWorker

Compositor Deadline



- Compositor has a **deadline** to decide when to give up on raster, which is:
  - Immediate if we are prioritizing Compositor latency.
  - A whole frame away if the Compositor is idle.
  - A fractional frame away if we are trying to be balanced. (skyostil@)
  - None for WebView. (sunnyps@)
  - Tied to NotifyReadyToDraw for the UI. (weiliangc@)
- The deadline is **triggered early** once Rasterization completes.
- An adjusted deadline is forwarded to the Blink scheduler.

cc::Scheduler

- Renderer and UI both have a cc::Scheduler
  - cc::Scheduler deals with time (the analog bits).
  - cc::SchedulerStateMachine deals with state, flags, and counters (the digital bits).
- Actions to control:
  - Animate
  - SendBeginMainFrame
  - Commit
  - PrepareTiles - Interface to TileManager
  - ActivateSyncTree
  - DrawAndSwap / InvalidateOutputSurface
  - BeginOutputSurfaceCreation
- See Should<Action> in code for when action will be triggered.

Renderer CompositorTimingHistory

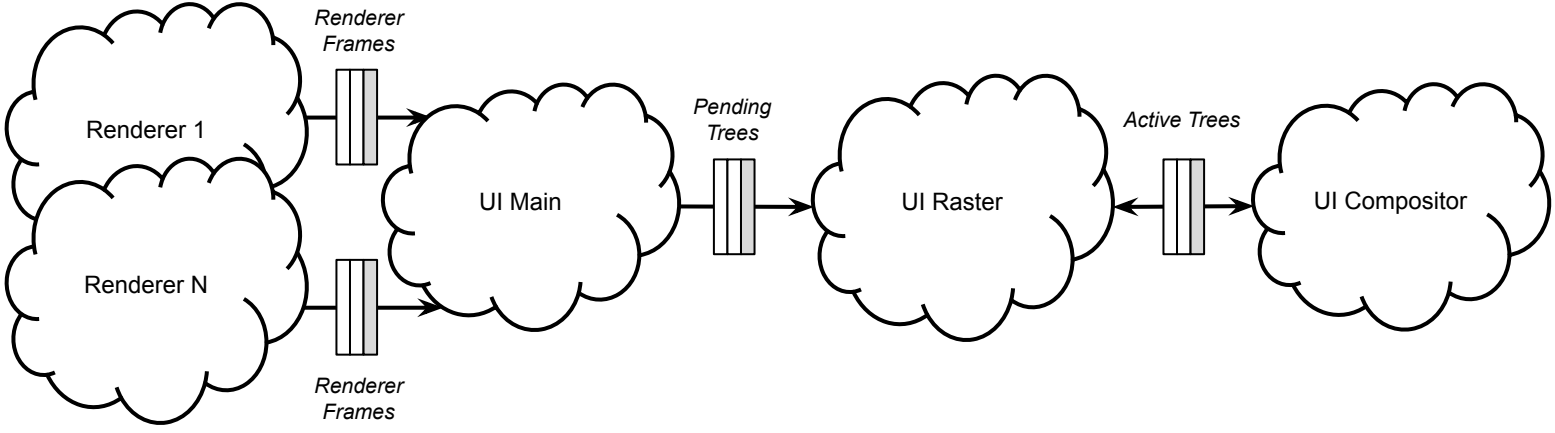
UMAs 9/28 : Android	50%	90%	99%
BeginMainFrameToCommit	4.28 ms	20.33 ms	184.86 ms
PrepareTiles	0.17 ms	0.61 ms	**2.90 ms**
CommitToReadyToActivate	1.96 ms	23.15 ms	184.86 ms
Activate	0.32 ms	0.90 ms	**2.90 ms**
Draw	1.17 ms	2.90 ms	9.33 ms

- Same data is used to make runtime latency recovery decisions.
- Activate can be slow, likely due to blocking on the GPU service.
- cc::Scheduler currently calls PrepareTiles once per frame.
  - TileManager responds with NotifyReadyToDraw and NotifyReadyToActivate.
  - Raster priorities would be more up to date if we could call it 2x or 3x per frame.
  - NotifyReadyToDraw isn't reliable which results in checkerboard.
  - Fallback to calling PrepareTiles 2x per frame after first checkerboard currently tabled.

Compositor Scheduler: Questions?

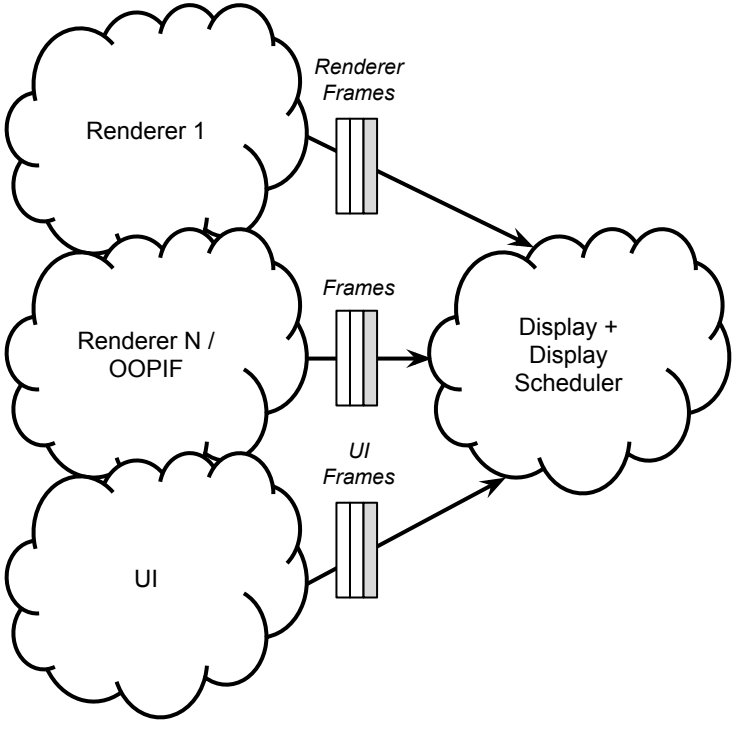
Next Up…
Surfaces and DisplayScheduler

Before Surfaces (BS)



- UI Main operates in a low latency mode with UI Raster.
- Single Renderer case is handled well and triggers UI immediately.
- Multiple Renderer case not handled well.

After Surfaces (Awesome!)



- Renderers and UI update asynchronously.
- Takes UI out of the critical path so the deadline can be more aggressive.
- Deadline can be triggered early once all active clients have queued a frame.
- Multiple windows per Display on CrOS.
- Other features:
  - Arbitrary embeddings
  - Frame synchronization

Surfaces Limitations

- To save memory, UI holds a write lock on raster resources that prevents the DisplayScheduler from acquiring a read lock to draw.
- Browser-only updates have a little more overhead.
- Multiple incoming frames per Surface not supported.
- Multiple Displays multiplexed on single UI thread.

DisplayScheduler

- Current version:
  - Consumes Renderer + UI updates simultaneously.
  - Recovers Renderer latency if swap ack takes a long time to return.
  - Helped enable Surfaces on Android.
- Upcoming version to have an explicit request/response.
  - Will help avoid unnecessary waiting and unnecessary latency recovery.
- Optimization opportunities for OOPIFs and Doghouse.
- Feedback from OS could help avoid blocking, triple buffering, and high latency modes.
- Resize could be handled better.

Unified Begin Frame

- What Display does a Surface belong to?
  - Use tree of RWHVs?
    - Pro: Hierarchy defined before first frame is even submitted.
    - Con: Hierarchy defined very far from where it's needed.
    - Con: Hierarchy is too rigid. A Surface can belong to multiple Displays.
  - Use tree of Surfaces?
    - Pro: Will be easy in the future to change policy.
    - Con: Hierarchy not defined until parent has submitted a frame.
- Decision: Use Surface hierarchy
  - For first few undefined frames use an "Orphaned" BeginFrameSource.
  - If we really need to, we can augment with the tree of RWHVs.

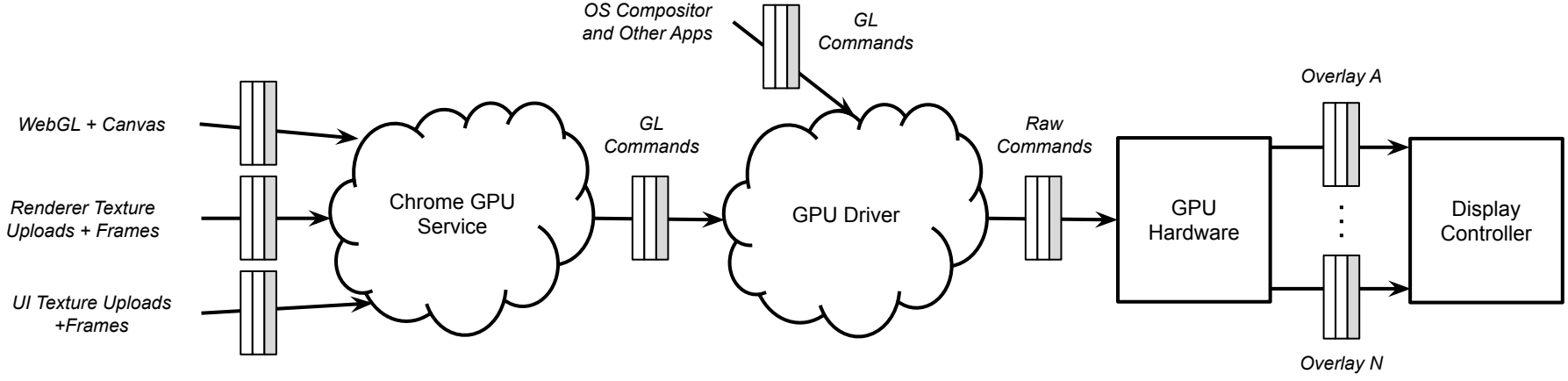
Surfaces and DisplayScheduler: Questions?

- Original Surfaces design doc. (jamesr@)
- Surfaces sequence numbers. (jamesr@ and jbauman@)
- Current status of Surfaces. (jbauman@)
- Detailed class/flow diagram of Surfaces. (brianderson@)
- DisplayScheduler Phase II design doc. (tansell@)

Recap

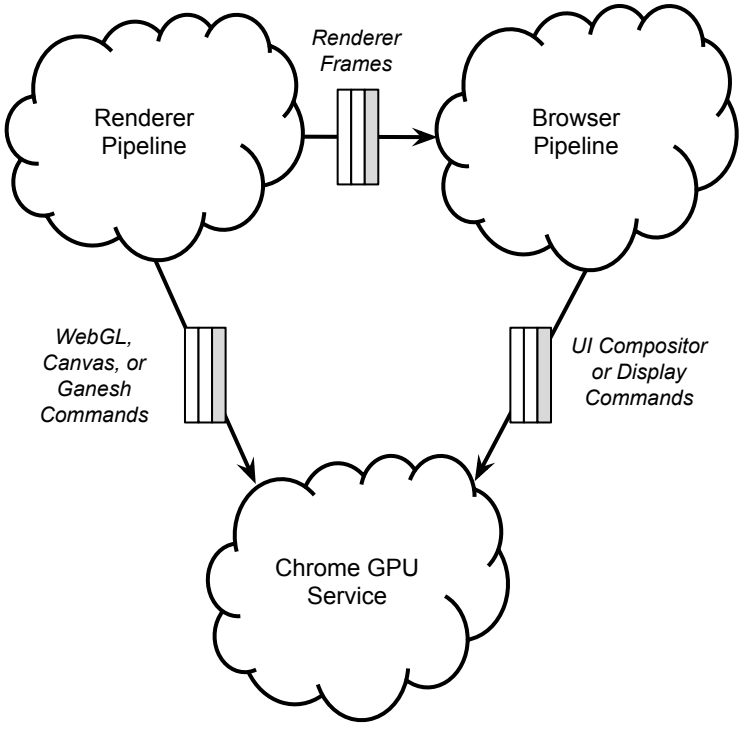
- BeginFrame
  - Blink, Animations, Video, WebView, Renderer.
- Pipeline Management
  - Latency, Throughput, Smoothness, Synchronization, Memory.
  - Latency recovery.
- Compositor
  - Deadline: Immediate, next frame, estimated, ready to draw, none, early.
  - Blink | Raster | Composite latency limitations.
  - Input synchronization: best effort, sync scroll, CompositorWorker.
- Compositor
  - Animate, SendBeginMainFrame, Commit, PrepareTiles, ActivateSyncTree, DrawAndSwap, InvalidateOutputSurface, BeginOutputSurfaceCreation.
- Surfaces
  - Takes UI out of critical path.
  - UI, Renderers, OOPIF async.
- DisplayScheduler
  - Unified BeginFrame.
  - OOPIF / Doghouse.
  - OS Latency feedback.

Hidden Latency



- Feedback regarding latency modes of later stages will help us implement **long deadlines**.
- Feedback regarding runtimes of later stages will help us implement more **precise deadlines**.
- Can we decouple BeginFrame phase from vsync phase?

Renderer + UI + GPU



- The GPU is a shared serial resource and context priorities are not available.
- To prevent the Renderer from queueing 2 frames before the UI queues 1, send a future sync point for the Renderer to wait on in advance.
- To reduce number of framebuffers needed, return buffer before it is actually released.
- To prevent a high-latency Renderer from pre-empting a low-latency Renderer, defer release of future sync point aggressively.

First Frame Latency is Very Important

The latency of all subsequent frames will only be as good as the first frame.

Some tricks we use to reduce first frame latency:

- Synthesize the first BeginFrame message in some cases.
- Proactive SetNeedsBeginFrame hides positive-edge latency.
- Retroactive BeginFrame attempts to catch up on frame production immediately if the deadline hasn’t passed.