Swap-RFH-at-CommitNavigation

Author: rakina@, altimin@, alexmos@, fergal@, haraken@, ...

Background

Current State

On cross-RenderFrameHost navigations, we keep the new RenderFrameHost as a “speculative RFH” and the previous RFH will stay as the current RFH until the navigation finishes committing.

When committing a cross-RFH navigation, we will call CommitNavigation on the speculative RFH to ask the renderer to commit, while keeping the previous RFH as the “current RFH” (this means embedders can still access/send messages to the previous RFH while the new RFH is committing). We will make the speculative RFH the current RFH in CommitPending on the browser side after we received a DidCommitNavigation call from the renderer.

Proposal

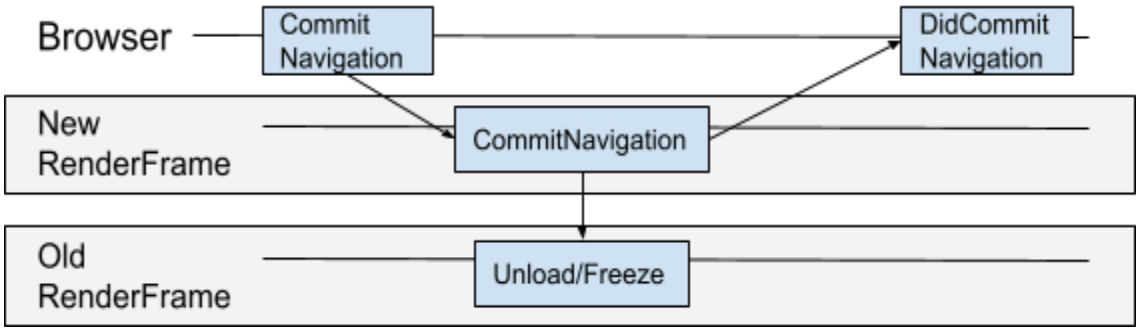
We should swap RenderFrameHosts on the browser side when we decide to commit the speculative/new RenderFrameHost (so, from CommitNavigation instead of after DidCommitNavigation). Or, even better if it’s possible: consider navigation as finished when we commit.

Motivation: Same-site RenderFrameHost swap

Most same-site navigations will be cross-RenderFrameHost navigations with RenderDocument/proactive BrowsingInstance swap/same-site bfcache. Previously, only cross-site navigations (and a small number of same-site navigations where we start doing dynamic isolation on a site) can be cross-RenderFrameHost navigations.

When we do a cross-RFH navigation, we will run the unload handler of the previous document after the commit of the new document finishes. This is different from same-RFH navigations where we will run the unload handler of the previous document before we commit the new document, so this will be an observable behavior change to web pages for same-site navigations (because the old page’s unload handlers might write to local storage that is also accessible by the new page, etc.), which is undesirable. See more details here.

To be able to run unload handlers for same-site cross-RFH navigation before the new RFH renders, we need to run unload just before the commit of the new RFH, as described here, and illustrated in the image below.



Problem: messages arriving to unloaded/frozen old/original RenderFrame

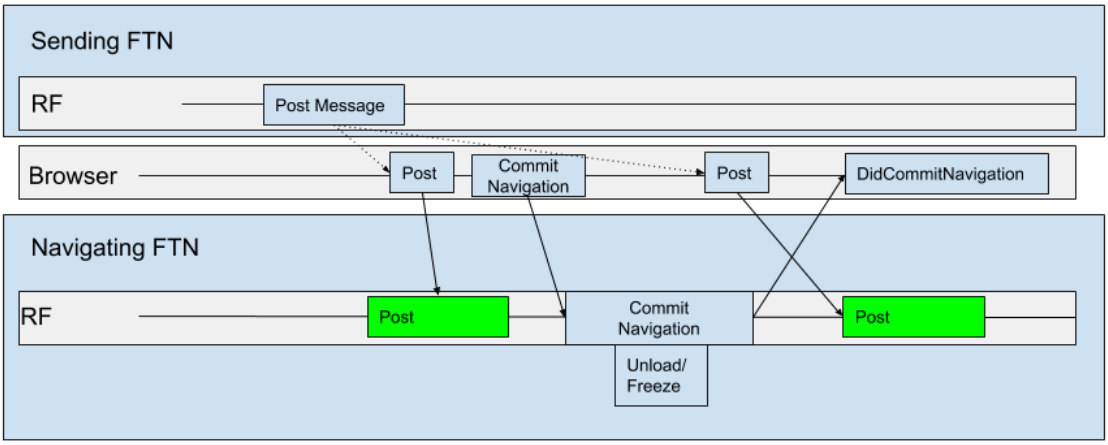
However, running unload just before the commit of the new RFH includes **unloading/freezing the old RenderFrame before the new RFH becomes the current RFH** (because it hasn't committed yet). This is problematic because **embedders or JS in other frames might send messages to the already-unloaded/frozen old RenderFrame**, and it will just silently get dropped. More details here and in the table below:

	Before CommitNavigation	CommitNavigation arrives in renderer, browser is waiting	After DidCommitNavigation
Current RFH	Old RFH	Old RFH	New RFH
Messages sent to current RFH	Processed in old RFH	Might be **silently dropped** if old RFH already unloaded/frozen	Processed in new RFH

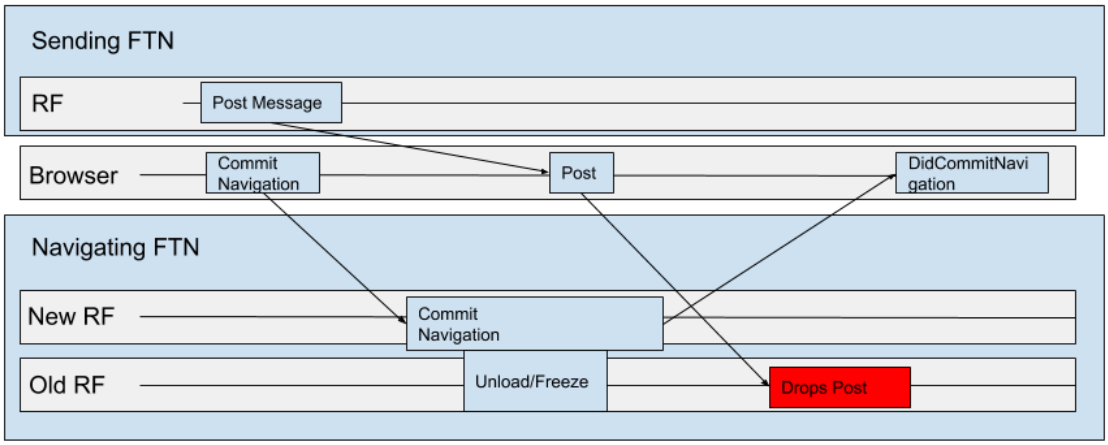
An example case (from altimin@):

- A frame sends postMessage to the main frame.
- Main frame navigates from a.com/1 to a.com/2.
- postMessage races with the commit: it is routed to a.com/1, but it's already unloaded and won't do anything. No postMessage handler script on the old RFH or the new RFH will run.

Before: Messages are never dropped



After: Messages may be dropped



In the example above the problem is with a JS-originated post-message. There can also be problems where some part of the browser sends a message to the current_render_frame but it lands on in an unloaded frame. In theory we could fix all of these but we cannot fix the JS-originated ones.

Same Site Proactive Browsing Instance Swap introduces this problem when a subframe sends a message to main frame as it is entering bfcache a subframe sends a message, it arrives after freeze.

RenderDocument for subframes makes this much easier to happen. The big question is whether it matters enough to be considered a blocker, in which case we need to do the Swap FRH at commit before launching RD-subframe past canary.

Proposed solutions

Proposed full solution: Swapping RFH at CommitNavigation (instead of DidCommitNavigation) will support unloading the old RenderFrame before the new RenderFrame commits while ensuring that messages sent when that happens won't silently get dropped - it will instead go to the new RenderFrame, which can decide to act on it or not.

Proposed temporary solution: Dispatch pagehide & visibilitychange before new page commits, but dispatch unload after new page commits. The actual unloading/freezing of the old page will still be done after the new page finishes committing. (see CL)

Since pagehide & visibilitychange handlers will run before the new page commits, this will lower the amount of potential problems. The only observable behavior change now will be for unload handlers.

However, there are still some possible race condition cases. An example case (from altimin@):

- A frame sends postMessage to the main frame.
- Main frame navigates from a.com/1 to a.com/2.
- postMessage races with the commit: it is routed to a.com/1, but arrives after pagehide(). postMessage tries to navigate via setting window.location.
    - before: postMessage would be dispatched after the commit and the navigation will proceed.
    - after: postMessage starts a navigation, which is ignored as it was started in an inactive frame (message from renderer to browser to start a navigation arrives after the old frame is replaced by the new frame).

Possible mitigations:

- Allow non-current documents to start a navigation if they are same-site with the current document.
- Queue postMessages if there is an ongoing navigation in FTN and dispatch them after the new document is swapped in.

Also, running pagehide & visibilitychange without actually unloading/freezing the page might increase the likelihood of script running after pagehide is dispatched, but it's always been possible for that to happen so it might not be a big problem [see comment]

Main question: Is swap-RFH-at-commit a blocker for same-site bfcache?

Should this be required/block same-site proactive BrowsingInstance swap (and bfcache)? Without swap-RFH-at-commit, it is impossible to run unload handlers before/during commit of new page, so unload handlers of the old page might run after the new page runs, which is an observable behavior change for same-site navigations. See more details here.

However, as mentioned in the previous section, we can run pagehide and visibilitychange handlers before the new page commits. Also, we only need to run unload handlers when the old page is not eligible for bfcache when the navigation is committing (as we won't fire unload handlers if the page gets bfcached), so this is **only a problem when at the time the navigation started, the page is eligible for bfcache** (so we decided to do a RenderFrameHost swap to prepare it to be bfcached)**,** then **at commit time found out the page can't be bfcached** - which might be a rare case, especially combined with **the possibility that the site depends on the "unload runs before commit on same-site navigations" behavior** (should be rare too).

If it is still considered a blocker, we might consider having incremental support for same-site bfcache (starting with caching only pages with no unload handlers). See plan.

All unload scenarios for same-site bfcache

- Page is **eligible for bfcache when navigation starts**: we will try to bfcache it and trigger a RenderFrameHost swap.
    - If page is **still eligible for bfcache when the navigation commits** [very likely - no problems here]:
        - pagehide & visibilitychange will be dispatched before the new page commits/runs
        - unload will not be dispatched because page is saved into bfcache.
        - Page will be frozen after the new page finished committing.
    - If page is **no longer eligible for bfcache when the navigation commits**:
        - pagehide & visibilitychange will be dispatched before the new page commits/runs
        - Page will be unloaded & **unload will be dispatched after the new page finished committing** [might be an observable change]
        - [TODO: add metrics to measure how often this happens. This should be rare because of the short timeframe between navigation start and navigation commit.]

- Page is **not eligible for bfcache when navigation starts**: we won't trigger a RenderFrameHost swap on same-site navigation, so it will go through the normal flow of same-site navigation [no problems here]:
    - pagehide, visibilitychange, unload will be dispatched before the new page commits/runs
    - Page will be unloaded before the new page commits/runs

Is swap-RFH-at-commit a blocker for RenderDocument?

We face a similar problem (messages arriving to currently-unloading old RenderFrame at commit time), but there's also another problem caused by not swapping RenderViewHosts. We can't keep the old RenderFrame fully functional after swapping RenderFrames during commit, because the old RenderFrame and the new RenderFrame shares a lot of objects (Page, RenderView, RenderWidget, etc.). So we can't keep the old RenderFrame around and unload it at a later time, unless we swap the shared objects to keep the two RenderFrames independent (which might be doable for main frames, but not fully doable for subframes). Thus, we need to do the unload when doing the RenderFrame swap, and swap RenderFrameHost at commit.

What's needed for swap-RFH-at-commit?

Remove DidCommitProvisionalLoad_Params

- We need to make sure we have all the information we need to finish the navigation in the browser side at commit time, which means we can't depend on DidCommitProvisionalLoad_Params.
- See this doc and this doc for more details.

Make it impossible to fail a commit?

Cases where the commit might fail from arthursonzogni@'s email:

- CSP embedded enforcement. This is going to be removed thanks to antoniosartori@.
- There was CSP:frame-ancestor, but this has been removed thanks to lfg@.

However, it seems like **there are cases that depend on the ability to abort navigation during Commit** - see: crbug.com/1099193, crbug.com/763106. Is it possible to remove all cases that lead to navigation cancellation in between CommitNavigation and DidCommitNavigation? What if another navigation starts when another is committing, etc.?

If we can truly remove all cases where a commit might fail, will this solve various longstanding issues associated with that?

Relevant bug:
crbug.com/1117282: Ensure NavigationClient::CommitNavigation always succeeds

Possible approach: Consider navigation as “finished” when committing

- Cancellations to navigation during commit time should not be treated as “canceling” but instead just another way to get error pages?
- If another navigation starts when another is committing, we will be able to handle it and start it immediately as the previous navigation has finished.
- Things currently done/calculated in DID_COMMIT state should be moved to READY_TO_COMMIT. Some are covered by this step already, see others in this comment thread.

Audit DidFinishNavigation & RenderFrameHostChanged

From altimin@:
WebContentsObserver::DidFinishNavigation is currently and will still be fired after the renderer finished committing/DidCommitProvisionalLoad). Meanwhile WebContentsObserver::RenderFrameHostChanged is currently fired after the renderer finished committing, but after we implement Swap-RFH-at-CommitNavigation probably should be dispatched when we actually swap the RenderFrameHosts within CommitNavigation?

This presents a problem for embedders which use DidFinishNavigation to track RFH changes and reset per-page/frame state. We have made some progress there by converting these features
to per-document, but there is still a substantial amount of work left to be done there (especially for desktop-only features, which we haven't really looked at).

Note: this is not a problem if we consider commit == finish. DidFinishNavigation and RenderFrameHostChanged will both be fired immediately after sending CommitNavigation.

[please add if we need to do other things :)]

Discussions Dec 23

- AssertNavigationCommit crashes needs to be solved
- Paint holding will always happen now
    - Is this bad? How to make it better?
    - Navigation Transitions won’t have this problem
- Unload runs earlier -> might compete with resources needed to run commit
- DidCommitParams header parsing
    - Make sure headers that affect commit/anything that the browser process cares about is parseable before commit (not in the renderer)

- Non-associated interfaces currently race?
    - This will be improved because we've updated the state on the browser process earlier
    - Fewer things will need to be associated?!? Whee
- Android WebView
- Renderer sending IPCs from Commit..DidCommit which is for the prev document?
- Can make Navigation Transitions better, but maybe don't want to make it exclusive to Navigation Transitions (swap-RFH-at-commit should apply to all navigations)
- Consolidate IPCs for PageBroadcast and stuff

Metrics Nov 23

- Want to know
    - How long do we show paint holding
        - Currently (starting from RFH swap at DidCommit)
        - If we move RFH swap to commit
        - Basically, how long from swap to FCP?
    - How long does it take from commit to FCP
        - How to answer this lol
- crrev.com/c/5086331 - compositor creation time to RFH swap time vs Commit time
- crrev.com/c/5057424
    - Record the delay between FCP frame submission to surface embedding, by tracking the timestamp of FCP frame and recording the metric when the surface is embedded in CompositorFrameSinkSupport. We need to send whether a frame is for FCP or not through CompositorFrameMetadata to CFSS.
    - Record the delay between surface embedding for the FCP frame to the presentation time of the FCP frame. (BTW is it possible to embed the surface used for FCP before the actual frame for FCP is submitted? How would we know that the surface is for FCP frame in that case?)
    - Record the delay between FCP's navigation start time to surface embedding, so that we can compare the time with FCP directly? We need to send the navigation start time through CompositorFrameMetadata to CFSS.

Hey @Jonathan Ross. I was talking to Rakina about measuring how often FCP frames from the new renderer are delayed by the browser embedding it. And I figured the most reasonable way is to probably generically track when a new SurfaceId is blocked from presentation because its waiting to be embedded by someone. And measure the latency from when it started waiting to when we realize that it has been embedded.
I figured the Viz stack would know this info but doesn't seem like we have such tracking at the moment.

Khushal Sagar, Nov 21, 11:02 PM

We're already tracking the timestamp for when a new frame is received here: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/frame_sinks/compositor_frame_sink_support.h;l=475;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2
IIUC, we'll create a new Surface when the FCP frame arrives in the GPU here: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/frame_sinks/compositor_frame_sink_support.cc;l=865;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2 . At this point, we need to know whether this Surface is already reachable via the root. Do you know where that info is? I'm guess we need that anyway to trigger a display redraw.
If its not reachable via the root at this time then we measure the time when it does become reachable and record that as "embedder delay".

Khushal Sagar, Nov 21, 11:16 PM
Ah, so this is where we know whether there is already an embedder which is embedding a SurfaceId: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/display/surface_aggregator.cc;l=2471;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2

Khushal Sagar, Nov 21, 11:20 PM
So @Rakina Zata Amni this is what I think you need to do:

1. At this point we create a new Surface: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/frame_sinks/compositor_frame_sink_support.cc;l=865;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2 . The FCP frame from the renderer will trigger this. At this point you can query if any embedder (a.k.a browser) is embedding this Surface by following a pattern of reaching all DisplayDamageTrackers via SurfaceObserver. Look at this code for an example of this pattern:
surface_manager_->SurfaceDamageExpected(current_surface->surface_id(),
last_begin_frame_args_);

2. If its not embedded, note the current time as "waiting for embedder".
Then you can use this hook to know when this Surface is drawn: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/surfaces/surface.cc;l=792;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2 which will indicate that the Surface has been embedded.
Actually the notification already reaches CompositorFrameSinkSupport: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/frame_sinks/compositor_frame_sink_support.cc;l=338;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2

Khushal Sagar, Nov 21, 11:24 PM, Edited
Add this delay to the presentationfeedback here: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/frame_sinks/compositor_frame_sink_support.cc;l=998;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2

Khushal Sagar, Nov 21, 11:28 PM
Finally if you follow the code from here: https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/paint/timing/paint_timing.cc;l=319;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2, you'll see this PresentationFeedback getting used in the renderer to record presentation time for FirstContentfulPaint.

Khushal Sagar, Nov 21, 11:32 PM
One edge case I can think of is the renderer IPC arriving first but before the next vsync. Since we only update whether a SurfaceId is reachable from the root when a draw occurs. So if you have a situation like:

- Vsync 0
- Renderer IPC
- Browser IPC
- Vsync 1

We'd see the renderer Surface is not embedded but that's because the next vsync just hasn't happened yet.
Will let @Jonathan Ross validate the idea and suggest how to deal with this situation.

You, Nov 21, 11:35 PM
Thanks a lot!! So for the edge case, we might record a longer delay because Vsync is when the embedded status update happens? And we can't just save the timestamp for the browser IPC? BTW on the CreateSurface call from your #1 link, is that path specifically only for when we commit a new document after navigation? Or do we need to filter out some cases?

Khushal Sagar, Nov 21, 11:39 PM
Its for any case where the SurfaceId changes. You don't need to filter it because once the PresentationFeedback for the new SurfaceId reaches the renderer, it would know if that SurfaceId corresponded to an FCP frame.
That's related to the code pointer here: https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/paint/timing/paint_timing.cc;l=319;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2
Viz supports getting "PresentationFeedback" from the GPU process to client processes (browser or renderer) for any frame. So my suggestion is to add a new "embedder delay" to this feedback: time spent waiting for this frame to be embedded. And then let the existing metrics code in the renderer record it as a part of recording the complete presentation delay for FCP.

You, Nov 21, 11:51 PM
OK thanks!! I think I need to figure out how the members of PresentationFeedback currently gets funneled all the way to PaintTiming, but I'll try things out!

Jonathan Ross, Nov 22, 12:21 AM
This race may not require Vsync 1 to arrive.

Within DisplayScheduler::UpdateHasPendingSurfaces there's logic to check that all clients who've requested BeginFrames have submitted vs them. This is an input for updating the deadline to attempt to Draw.

So if both Renderer and Browser requested BeginFrames for VSync 0, received the VSync, and submitted before VSync 1. We can attempt to DrawAndSwap before VSync 1. Though if it successfully presents in time is racy/platform dependent
However any race condition around that would be encompassed by the PresentationFeedback given to the CompositorFrameSinkSupport
Adding to PresentationFeedback a flag about having been blocked on surface dependencies for activation would be a clean way to get this concept back to the FCP location in PaintTiming

Jonathan Ross, Nov 22, 12:28 AM
The rough path for PresentationFeedback:
GPU: CompositorFrameSinkSupport
Renderer Compositor Thread: AsyncLayerTreeFrameSink
Renderer Main Thread: LayerTreeHost::DidPresentCompositorFrame

Then for blink, iirc:
RunCallbackAfterPresentation
And PaintTiming somewhere creates the ReportTimeSwapPromise that registers that final callback

You, Nov 22, 12:52 AM
OK, thanks a lot! I'll try to write the CL today

You, Nov 22, 4:38 PM
OK I didn't manage to do that in time 😅, mostly I'm not sure how to check whether the surface that's newly created is already embedded (is that possible if the surface is newly created? Or is the surface not always newly created?). From Khushal's links it seems like I need to check that the surface ID is in damage_ranges_ in SurfaceAggregator: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/display/surface_aggregator.h;l=438;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2. To get that, maybe I should expose a new function to check for that in SurfaceAggregator -> DisplayDamageTracker -> SurfaceManager? Or is there a better way?

Jonathan Ross, Nov 22, 6:09 PM
When we call Surface::CommitFrame is knows if that particular frame has dependencies blocking immediate activation.

https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/surfaces/surface.cc;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2;l=264

IIRC the Browser's submission that contains the SurfaceRange would have the list of dependencies.

Looking at Surface::UpdateActivationDependencies we reach into SurfaceManager to find the SurfaceAllocationGroup for all these dependencies that block us. We could us that to notify Surfaces that were waiting of the fact they were blocking activation

Jonathan Ross, Nov 22, 6:21 PM
Spoke with kylechar about the Renderer frame arriving first. So when we create any Surface for the first time we add a 'temporary reference' to keep it alive SurfaceManager::AddTemporaryReference then when the Browser frame arrives to embed it for the first time, the temporary reference is removed and it becomes permanent Surface::RecomputeActiveReferencedSurfaces

Jonathan Ross, Nov 22, 6:27 PM
hmmm, we seem to always set that though. So right now when the Renderer frame arrives the only direct way to know it isnt embedded yet would be to see that it is not in SurfaceManager::references_ Though that is keyed on the parent ID. So it would be expensive to check all of that map to try to find if we exist

Khushal Sagar, Nov 22, 6:37 PM
Hmmm, we could only check the root SurfaceIds? And be explicit that the delay is only about whether this Surface is embedded by the root directly. FCP needs just that.

I don't like the idea because it assumes that the web page will be directly embedded by the root but just wanted to mention it.

Jonathan Ross, Nov 22, 6:40 PM
Yeah that would skip nested OOPIFs. Though IIRC FCP is only supposed to be once per overall page load. Being the first of the frames to paint
It might just be cleaner when the blocked Surface activates, to just notify all activation_dependencies that they had been blocking it

Khushal Sagar, Nov 22, 7:42 PM
The idea sounds interesting but I'm not sure how exactly we'd use it. The case we're interested in would have the browser's Surface never blocked because its activation dependencies were met when the frame arrived since the renderer frame came before browser's frame.

Jonathan Ross, Nov 22, 7:59 PM
Yeah. It could be useful but tricky/ugly

Renderer submits frame, activates immediately, kept alive by temporary refs
Browser submits frames, creates new Surface, and checks dependencies.
We could possibly use that to tell the dependency that it is now referenced for the first time.
Kinda inverse of adding the missing surface to the activation_dependencies list in the Browser's Surface

Khushal Sagar, Nov 22, 8:26 PM
Tbh, checking the map you mentioned above seems like the easiest option: https://source.chromium.org/chromium/chromium/src/+/main:components/viz/service/surfaces/surface_manager.h;l=322;drc=836b16f53763385ba17b35d359c1a4ee9f686a45. If you're worried about the overhead of doing it on every frame, we can ask the renderer to opt-in to computing this. Then we'll only be doing it for an FCP frame and that seems ok to me.
We still need a notification back to CFSS when its SurfaceId gets added to references_. That should be fine?

Jonathan Ross, Nov 22, 8:38 PM
Should be fine to add that

You, Nov 23, 12:10 AM
OK so the idea is to go through references_ when the surface is created in MaybeSubmitCompositorFrame, but only when the compositor frame is an FCP frame? Can we know that it's an FCP frame from that point or do we need to add something for that?

Khushal Sagar, Nov 23, 12:16 AM
We don't know it right now but you can plumb a bit to the GPU process on CompositorFrameMetadata.

Khushal Sagar, Nov 23, 12:20 AM
Take this API as an example: https://source.chromium.org/chromium/chromium/src/+/main:cc/trees/layer_tree_host.h;l=422;drc=ca067f2604f9bf0ff2fa070eafeed664698d819a. If you look up the stack, you'll find the spot where we're calling this API for the FCP case. At that point you can also set a bit on LayerTreeHost saying "include embedder delay in presentation feedback" and send that all the way though to Viz via CompositorFrameMetadata.
Also look at how the callback passed to RequestSuccessfulPresentationTimeForNextFrame is being plumbed through the system for an example of how data flows through CC to reach CompositorFrameMetadata which is serialized and sent to Viz.

You, Nov 23, 12:29 AM
Thanks! I thought RequestSuccessfulPresentationTimeForNextFrame is currently only used on tab switching and BFCache cases, but are you saying it is also called for all FCPs?

Khushal Sagar, Nov 23, 12:30 AM

Either this or the RequestPresentationTimeForNextFrame variant above it should be used for the FCP frame.

You, Nov 23, 12:42 AM
Is that because that path is called every time we show the new RenderWidgetHost after navigation? But from here it looks like we only call that if we have a VisibleTimeRequest, which is only set for tab switching/BFCache now
https://source.chromium.org/chromium/chromium/src/+/main:content/browser/renderer_host/render_widget_host_view_base.cc;l=1072-1073;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2

Khushal Sagar, Nov 23, 12:59 AM
Looks like this is what we're using for presentation feedback:
https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/frame/web_frame_widget_impl.cc;l=3303;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2
So you can actually use this hook to set a bit on the metadata if you want the embedder delay to be tracked:
https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/frame/web_frame_widget_impl.cc;l=3245;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2
Follow this up the stack:
https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/frame/web_frame_widget_impl.cc;l=3409;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2 and you'll see where it gets triggered for FCP.

You, Nov 23, 1:04 AM
Thanks, got it! Looks like the FCP one is here:
https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/paint/timing/paint_timing.cc;l=291;drc=d4a7d3fb6f5100019d6153d5cf00c60f06b1d0a2

Khushal Sagar, Nov 23, 1:13 AM
I'm going off corp now. Good luck!

You, Nov 23, 1:18 AM
Thanks for all the help!!