WebPerf WG @ TPAC 2018

bit.ly/webperf-tpac18

Logistics

Location & participation

- Cité Centre de Congrès de Lyon, 50, quai Charles de Gaulle
- Remote participation:
    - **WebEx Meeting number: 646 029 056**
    - **Password: click this member-only archive or ask the co-chairs**
    - **Click to join by browser**
    - **Join by phone: +1-617-324-0000,,,646029056#**

Resources
- TPAC Schedule
- Web Performance Working Group charter
    - Spec status spreadsheet
    - WebPerf WPT tests

Attendees

- Todd Reifsteck - Microsoft (Thursday, Friday)
- Yoav Weiss - Google (Thursday, Friday)

- Nicolás Peña Moreno - Google
- Shubhie Panicker - Google
- Tim Dresser - Google
- Garrett Berg - Airbnb
- Philip Walton - Google *(attending Thursday and Friday, but will probably miss parts of Thursday to attend CSS Houdini)*
- Alex Christensen - Apple
- Ryosuke Niwa - Apple (Thursday afternoon; lost Thursday morning for more web components meeting)
- Kinuko Yasuda - Google (Friday, maybe Thursday too but only when I'm not in SW F2F)
- Nate Schloss - Facebook (Friday)
- Benoit Girard - Facebook
- Eric Faust - Facebook
- Andrew Comminos - Facebook
- Xiaolu Huang - 360 Technology Co., Ltd.(Thursday, Friday)
- Bowen Liu - 360 Technology Co., Ltd.(Thursday, Friday)
- Qingqian Tao - Baidu(Thursday, Friday)
- Addy Osmani - Google (Thursday, pending remote participation)
- Ben Kelly - Google (Friday)
- Xiaoqian Wu - W3C (Thursday, Friday)
- Charles Vazac - Akamai
- Gilles Dubuc - Wikimedia (Thursday, Friday)
- Thomas Steiner - Google (partly on Thursday and Friday)
- Nic Jansma - Akamai (Thursday, Friday)
- Weichai Huang - Baidu
- Chris Harrelson - Google
- Markus Stange - Mozilla
- Kouhei Ueno - Google
- Douglas Creager - Google (Thursday afternoon, remote)
- Makoto Shimazu - Google (Friday)
- Matt Falkenhagen - Google
- Marijn Kruisselbrink - Google (Friday)
- Surma - Google

Observers

- Thomas McCabe - Squarespace (Thursday, Friday)
- Rob Buis - Igalia (Friday only)
- Yutaka Hirano - Google (Friday)

Agenda

Thursday October 25th - **Triage**

(Overlaps with Houdini and Service Workers)

Timeslot	Subject	Scribe
8:45~9:00	Intros & agenda review	
9:00~9:30	Goals and how we get there	npm
9:30-10:45	Preload, Resource Hints	xq
10:45~11:00	Break	
11:00~12:30	Performance Timeline, Resource Timing, Navigation Timing L2	
12:30~14:00	Lunch	
14:00~15:45	Long Tasks, Paint Timing, Page Visibility, requestIdleCallback	cvazac
15:45~16:00	Break	
16:00~17:30	NEL, Reporting, Server Timing, Distributed Tracing collaboration	tdresser

Friday October 26th - **Design**

Timeslot	Subject	Scribe
8:45~9:00	Intros & agenda review	
9:00~10:45	Preload (yoav) Input Timing (npm) Prefetch (yoav)	njansma
10:45~11:00	Break	
11:00~12:30	Scheduling API - slides (panicker) hasPendingUserInput - slides (tdresser) FetchEvent Worker Timing (wanderview)	philipwalton

12:30~14:00	Lunch	
14:00~15:45	In-progress requests, (npm) Scheduling Off Main Thread: Task worklet - slides (panicker) JS profiling - slides (acomminos) Page Prerendering - slides (Qingqian)	Yoav
15:45~16:00	Break	
16:00~17:30	Priority Hints (yoav) Page Lifecycle: brief update - slides (panicker) Element Timing (explainer) (npm) Layout Stability - slides (tdresser) Platform supported A/B testing - slides (tdresser)	

Breakout sessions

- RT issue 70 - discuss with SW and webappsec folks

Thursday minutes

Goals and how to get there

Yoav: let's talk about what this group does, why, and how. Our mission is to enable developers to monitor and improve the performance in ways that improve the user experience. We have two categories of features: the monitoring features (measure user experience), and performance enhancing features (improve user experience by making it faster).

Yoav: We want to ship features in both specs and implementations. In specs, this means graduating the spec to CR. That can only happen if there are two browser implementors, which requires the features to be stable.

Yoav: There are two specs ready to move from PR to REC: requestIdleCallback and PageVisibility. Four are ready to move from CR to PR: HR Time, User Timing, Beacon, and PerformanceTimeline. And ServerTiming is ready to go from WD to CR. However, PageVisibility and Preload have some issues that make them harder to move along. ResourceTiming and ResourceHints have a lot of historical debt that needs to be addressed. LongTasks, PaintTiming, DeviceMemory, Reporting, NetworkErrorLogging are only implemented in one browser, and we'd like to see broader adoption of them.

Todd: Maybe we need to understand why other browser vendors choose not to implement these and arrive at a consensus.

Yoav: We do a F2F usually sometime in the summer. We also have bi-weekly calls that are split between issue triage and new proposals. So far we've made great progress. But we also have technical debt, missing spec infrastructure integration (mostly integration with the Fetch spec to properly define what features are doing and what they're not and avoid tricky bugs). I plan to work on this in the near future. Not all features are shipped in all browsers, so like Todd said it would be great to see if there are any blockers or reasons for lack of interest.

Yoav: It would be great to have more spec editors and more people writing WPTs. This would allow us for example to build more features on top of ResourceTiming and PerformanceTimeline. If anyone here wants to volunteer to do some of this work that would be greatly appreciated.

Yoav: For requestIdleCallback, there is 1 PR open in the HTML spec in order to integrate it in the event loop. That PR hasn't landed yet: there was some feedback and it hasn't landed yet. Looking at tests: it is almost green on two implementations and entirely red on other two (not implemented yet).

Todd: Should we review the details in each of the slots throughout the day?

Tim: Are there slots for all of them?

Yoav: No, not all of them.

Todd: Ok, let's spend time on those that do not have slots allocated later.

Yoav: For HRTime, Todd opened an issue 11 minutes ago. Spectre is an issue for HRTime, so currently the spec allows browsers to clamp. But Phillipe wants us to re-run the spec through a security review to ensure it is safe. That means we cannot move it forward at this point.

Tim: Should we assign that to someone?

Todd: I'll assign it to myself. I'll get it scheduled for a security review.

Yoav: So, for HRTime, 0 open issues is a lie. In terms of tests,

Charles: They made a change in the IDL harness script that is only supported in Chrome 71. So there is a flag that you can add, maybe label:experimental. Does that help?

Todd: It showed more green.

Yoav: But there’s still some failure on the IDL harness side.

Tim: I’ll file a Chrome bug

Markus: I’ll file a Firefox bug

Yoav: UserTiming: 0 bugs. Are there any objections to move UserTiming to L2?

Todd: Can you remind us what is different in UT L2 versus L1? We added performance.mark and measure in workers and added clarifications to the mark names. When you reference a string in NavTiming, we’ve clarified the errors that should be thrown.

Yoav: And for L3, the key change is to be able to create arbitrary blobs of data that you would want.

Todd: So for L2, it’s just adding the workers. So based on this, is there any push back?

Yoav: Same question for Beacon. 0 open issues. Regarding tests...

Todd: Almost green in 4 implementations. The tests are a bit flaky. The problem is that they rely on the server and client timing. We also need to look into the header tests, which were added during the security review. Is there push back assuming we follow up on the tests?

Xiaoqian: Do we know if there are any changes from CR?

Todd: No, mostly on the fetch spec itself

Yoav: Keepalive?

Todd: No, the specific test for CORS behavior.

Yoav: I’ve filed Chrome bugs.

Alex: I’ve made a reminder to follow up on tests.

Markus: I can file a bug for Firefox, no context on this.

Yoav: Skip ServerTiming for later.

Xiaoqian: Do we want privacy/security review for ServerTiming?

Yoav: Is that a requirement to ship to CR? Then yes.

Yoav: Most specs are using Respec, some of the newer ones are using bikeshed. Everyone familiar with those?

Markus: no, but no explanation needed.

Yoav: Respec folks have been helpful when there are issues. Some issues are related to broken links. I talked to the Respec folks and this should be fixed shortly. That is, links and anchors will be verified. We should also host our stable versions, instead of linking to the latest, which potentially could break

Tim: do others do this?

Yoav: yes. There are some advantages to Bikeshed: verifies links, not sure if it verifies the anchors though? So there can still be issues. And Respec, there is some problems with flash. My opinion is that if we fix broken links then we can stick around with Respec.

Tim: not worth converting things, but what should we use for new specifications? Do you propose we stick to Respec?

Yoav: We could stick to Respec for the next few months and see if link issues are fixed?

Todd: We should have someone assigned to this.

Yoav: Do we have a tracker for this?

Todd: Not for this meta issue. Can you follow up on this, Yoav?

Yoav: I can follow up on that. Can you open an issue and assign it to me?

Todd: Sure.

Preload

Yoav: 2 parts, today and tomorrow
Current blocking issues and feature requests

Issue 127
Yoav: Reason for the problem: mismatch of resource and fetch in the implementation
… one solution is to add the integrity attribute, by the developers
Tim: both the link and header?
Yoav: right
… simpler solution for the implementers, extra burden for the developers
… another proposal is to include the hash

… simpler to do the matching
… 3rd option is the link preload will not include any integrity
Tim: including the integrity only in preload?
Todd: only include it in the preload will be problematic if the preload fails
Alex: will be problematic if the data is in a hash form
Todd: it will require to storage all the hashes
Kouhei: Option 1 will allow to save memory, as for Chrome
Alex: if it will allow cache improvement, makes sense to me
Todd: putting it in preload will prevent it getting into cache
Alex: no reason to specify the hash type
Todd: wait until someone use it
Alex: wait until the download?
Stefan: maybe just specify the hash in the algorithm
Tim: if we know the hash for the whole time, it's heavy
Todd: it can choose to use a certain kind of hash
.. allowing the link for preload in the hash seems a good thing
Yoav: talked to webappsec, folks found value in the 1st proposal
… we can further improve it in the future
Gilles: there is a use case for this, both is useful, parallel to the issue
Yoav: have you quantified the memory saved in that way?
Kouhei: no exact number, we looked at a few cases, this is just one use case
… if they use SRI, it's possible to save MBs
Yoav: will it be possible get rid of the buffers?
Kouhei: not likely
Yoav & Tim: you can have multiple script tags, not sure if it will break
Alex: this is easy to fix
Kouhei: do we need the hash all the time?
Alex: by the time preload is generated, no
Todd: when hash comes down
Gilles: if content changes in the script
Todd: is hash itself in the body or header?
Gilles: body, in the html
Todd: when do the server actually provide the hash?
Gilles: with H2-priority, both parties can compute that
Todd: UA will have to provide piece of memory to contain the cache
Gilles: it's a tradeoff
Yoav: if you don't specify anything, it will assume all the documents are using it
… if we include the hash, and make sure all browsers support it
… step 2 is to improve how we compute the hash
Todd: no differences to what preload already allows today
Resolution: specify the full integrity to link rel preload in the immediate. Specify a mechanism to describe just the hash as a secondary priority.

Issue 130
Yoav: it's time to discuss whether the browsers should put limit to prefetch and preload chains
Tim: how is it different from other content?
Alex: can the UA say I'm not going to preload it?
Yoav: related to issue #129
… preload can be triggered in the header
… UA can decide to not to preload
… regarding the chains, preload open it for the contents
Todd: is it a behavior for the parser?
… prefetch says downloading has no preferences
Yoav: if it receives the header, it's under the regular fetch processing model
… once we accept that..
Todd: there will be a problem if the following fetch is not allowed
… fetch spec should be changed?
Tim: the UA can decide preload or prefetch can be ignored?
Yoav: we can discuss this tomorrow
Alex: we should say all of them are suggested
… just a suggested performance improvement
… we should change that
Yoav: preload is not in the header, but in the markup
Alex: net error happens, UA decides to block things
Yoav: some of the use cases will break. Various loading patterns rely on preload fetches to be as reliable as e.g. script fetches.
Gilles: may be a problem for those making websites
Yoav: if UA has chosen not to load the content, not a problem
… but if it decides to load the prefetch
Todd: in general it's fine, some UAs today will start the prefetch when you starting typing the URL, some in a different behavior
Tim: shall we make it transparent to the page?
Todd: meaning some type of flags?
Yoav: no document
Todd: prerender flag behaves not perfectly
Yoav: should be specified in the processing model
Todd: is it a blocker for the preload spec itself?
Yoav: should be specified in the fetch spec, may affect the chains
Alex: for a lot of fetchings no need to specify
… in reality anything can be blocked, some of the reasons to block things have been specified
… it should say all preloading is a suggestion
Todd: Fetch spec says it can fail
Alex: undetectable

Issue 97

Yoav: We need to define a preload cache. Webkit and Chrome rely on their MemoryCache to implement it, but it's underspecified, and I think Firefox ran into trouble there.
Markus: may be true for Firefox
Yoav: We need to define a minimal set of such cache in order for it to be interoperable
Todd: we all have memory cache…
Yoav: if there are multiple script tags, Chrome and Webkit will reuse the same resource. Other browsers may behave differently. We need to specify what the behavior should be and align.
Todd: important for images
Yoav: For images this is already defined in the html spec
Todd: memory cache will force double download
Tim: why not keep the same way for resource…
Todd: in general, we should, but may have edge cases for privacy reasons
Alex: like the idea not to double downloading
… defining things knowing you are not using heavy memory
Yoav: for memory pressures, dropping it from memory cache, and download it again…
Alex: we may make suggestions randomly for privacy reason
Yoav: UA can be allowed to make decisions

Resource Hints

Yoav: 18 open issues, many related to prefetch and dns prefetch
… Ilya suggested to break it into two parts, keep prefetch in the processing model of resource hints
… the implementations for the 4 attributes seems unrelated to each other
Tim: we should not make the inter-dependency problem worse, as the Firefox folks complained before
Yoav: not sure for the Chrome side
Resolution: split the 4 attributes up and achieve the resource hints

Issue 78 & 82
Yoav: double-key caching and prefetch don't really get along in browsers now
… the double-key caching model is not compatible with current processing model
… when it comes to service workers
… you don't know which service worker will handle the sub resources
Tim: any example?
Yoav: you don't know the origin that will load the sub resources
… you don't know whether it will be used by a.com or b.com
… even if we limit it to same origin, still no way to know which page will use it, and different pages can be handled by different scopes
Todd: we don't need to skip service workers
Yoav: you have to prefetch html
… I don't know what the html will be
… we need to figure out a way to define the scope

Todd: this is important as it will affect the network implementation
Yoav: not necessarily
… it will work if you want to skip service workers
… the developers shouldn't know
… for double-key we will need to skip it for privacy reason, but developers should be awared of
… please bring use cases tomorrow
… 9:00 - 9:45
Alex: prefetching of subresources can be a topic for tomorrow

Performance timeline

Yoav: It's the spec that underlines all the specs that we're will talk about. Underlines the layering of the spec. This is something that we're planning on keeping. We had complaints about the layering. We're planning on keeping it as the infrastructure for everything else.

… There's a few issues that are marked as L2 blockers. Generally the L3 parts were scheduled tomorrow but we switched this around for the SW talk tomorrow.

… issue 100: Ensure tests are clean and L2 is republished after updated.
Tod: I don't remember the specifics. This is really about are the tests clean.
Yoav: We could look at this that way. Tim can we file bugs and make that cleaner.
… We could rename this to ensure tests are clean. Should we close this? Do we need to spell it out?
Tod: Doesn't matter, we historical track issues that are blocking the spec. That's the reason I created. If you'd like to resolve the issue and track it another way.
Room: all browsers or two green implementation
Yoav: two green implementation. Ideally the same implementation being all green.
Tod: But it could be different implementations if we have consensus.

Issue 77
Nicolás: We need to include a small hook which allows other spec to add such a string to this static list. For example when we have (?) link timing. So that implementers know they have to update this static list. All that's left is for me to add this.
Yoav: To be clear the goal is that new things will have to register rather than having to update performance timing everytime something is added.
Tim: Are we willing to go back through and ask each spec to register.
Todd: The work to complete this if I understand correctly is an active pull request. Do we have someone from Mozilla to review these PR. This is about adding types to the performance observer.
Mstange: That person would be baku (sp?)
Alex: I have no objections.
Todd: So it would be a spec update, tests and implementation in two browsers.

Issue 103: EntryType-specific parameters
Nicolás: I can summarize. We have two possible entries in the dictionary. (1) … (2) we have the buffer flag that can dispatch to the performance timeline… There's two case, 1) observe multiple entry type with the default param 2) Observe a single entry type with the parameters, this one is stackable and can be called multiple times and have it observe multiple entry with different kind of param. With the same observer. The caveat is that it overrides, I think this will be maintained. It will not stack.
Will it nuke the ones that you've added it before?
Nicolás: No one has implemented this yet.
…
Tim: In terms of whether you nuke previous or not. I think you want to otherwise it's confusing. Multiple entry type, single entry type. It would be weird to only nuke some and not all.
Yoav: People shouldn't be using both.
Todd: The use case is 8 3rd party scripts. You don't want them to trash each other.
…
Yoav: It makes me want to ban mixing both.
Nicolás: We could throw an error since the stacking order is …
…
You can co-observe it will always add and never nuke anything.
Nicolás: I think we can break that, since people won't rely on it. We could make them all stacking.
Todd: How does it behave today?
… The call are completely overwritten.
Tim: I propose that we measure how often observe is called with different entry types. Get actual data.
….
Todd: There's two options. The question is can we have the protocol allow adding multiple entry type and be consistent.
Yoav: This was the previous proposal but it got push back because it's not consistent.
Alex: This API should be consistent to other API part of the web platform. If you can observe twice it will observe both. In this case existing behavior should be preserved. If there's multiple entry that are not in the spec, do the old behavior. If there's one entry type with the type key assume that you're using the correct specified behavior and behave accordingly.
Yoav: I understand that it's counter to other API in the web platform. But it doesn't seem like something we can deprecrate in the short term.
Todd: I think this is what they were asking. If the usage is low maybe we can break it now.
Nicolás: We're talking about changing multiple calls, not just one.
Tim: our hypothesis is that is rare to … its the only case that would break if we change the behavior. We can see how common this rare case is.
Alex: … if we wanted to deprecate existing behavior we could if it has an array of entry type we could… if it has a dictionary …
Yoav: if we're mixing two behavior its complex. Throwing seems like the most dev friendly thing to do.

Room: maybe…
Todd: The tricky thing with throwing is some UA wont throw yet.
Nicolás: Currently no one has implemented so there should be no risk of no one throwing.
… We could add a new type. The reason we add new type.. There's different types of entry.
Yoav: Tim you look concerned.
Tim: No it seems reasonable there's question if we should deprecate existing behavior, if we don;'t want to … and then they should be able to play nicely together.
Nicolás: if we throw wew can keep the old behavior there won't be finicky causes.
Tim: By can prevent mixing them by not throwing.
Yoav: EVen if we intent to keep the behavior, not allowing to mix them is hard to explain vs. either this or that.
Tim: If they never overwrite and only add it's better from an ergo perspective. You have multiple problem with multiple long tasks.
Tim: It should overwrite if it already exist.
Yoav: THat seems reasonably simple to exist and will enable either path, deprecation or not. Do we have stats on usage?
Tim: we have instrumentation for over, not specifics.
Yoav: We can track over time how many people are moving to the single type behavior and see if it's possible to deprecate in the future.
Todd: Is the reason, i going to ask the counter. Should observable support multiple entry…. What about the converse, should …
Tim: I think there's a difference that other calls have more metadata. The more the more difficult it is. One datapoint is that on chrome 13% of page visit have some use of performance observer.
Todd: Do they do anything of the interesting…
Yoav: We will need to move all those if we try to deprecate the list.
Todd: I bet all 12.9% of them are ???.
Yoav: I think we have understanding of what we want between throwing or not clob.
Tim: I would verify that we can not clobber and ship that. But if people think throwing is better.
Charles: I agree.
…
Tim: That could break existing content, we have to do research.
Todd: One could cause break leading to extra data. The fix is … take records of the disconnect.
Nicolás: I think it's likely that two observe call will be unlikely.
Yoav: If we're opting for that we can don't need more data, if we opt of throwing we need data.
Alex: I think throwing would be great but I don't feel strongly. It would be great for the future depreciation of the API. I think that would be a good thing to encourage.
Yoav: Slight pref towards throwing. The users in the room does it matter.
Charles: I think the observe … multiple calls, you sort on the entry type and compare. If the use count is very low we could compare an examine what it means.
…
How do you normally deprecate?

Tim: normally we do reach out but we don't deprecate something with that much usage. If its just a few libraries maybe my intuition is wrong.
Todd: Step 1 is … i read the thread and im struggling to see what the conclusions are. Can you summarize exactly what we will do at the end of the issue?
Yoav: At the top can you summarize the issue and say the current status.
Todd: .... I'm not trying to micromanage, just trying to understanding.
Tim: Best data I know how to gather how many libraries use this is to use HTTP archive.
Todd: Performance.observe with minification you're going to have to wait for real world.
Tim: But we can't get library because of privacy reasons.
…
Yoav: if we're pending the discussion if were going to deprecate in the future, we need to know if its 5 users or a million. Updating the libraries won't magically update their snippets.
Todd: I assert we shouldn't base our decision base on deprecation issue. What were looking at is how do we get the behavior we need and how to we get there with breaking the web.
Tim: I agree, if we want to deprecate at some point. The shape of the API gives us leverage to get people to use the new API.
Todd: Deprecation is something for the UA, it's not a blocker.
Tim: Should we check if we can add multiple entry type instead of throwing.
Todd: I don't have a pref, that's why i'm asking you guys. Web compat makes the call given 12% adoption.
Tim: My impression is neither option will break the web.
Todd: If we have the string we're already different. We can chose to throw or chose to add.
Yoav: We can throw regardless, even if depcaration is not viable. Choosing to throw will steer people towards the right thing.
Tim: Low man hours to get the data but high latency to get the data in the pipeline.
Yoav: Let's conclude that we will get the data and make the decision.
Tim: My concern is were throwing we don't need the data.
Todd: We want to move people towards the new API, if no one is overwriting it's safe to add the throwing.
Tim: Ohhh… you want throw in the case that you add twice.
Todd: …. *giving example* Once you first call observe you stuck in that mode and if you ever…
Alex: With that observer.
Yoav: I think we should use that.
Nicolás: If they start using the new calls its easy to migrate.
Tim: Are there examples where a single observer is shared between multiple scripts.
Room: Let's throw!

Tag review?
…: I think we need to go to CR again. Do you want to tag review again.
Todd: Should we pull performance timeline back to draft(?) since we're waiting for these changes to be done?
Tim: I don't have a sense of the overhead but it sounds correct to me.

Consensus: Tag review sounds appropriate, we want to wait for these issues to be done first.
Yoav: Can file a TAG review but they won't review it until these changes are done.

Buffered flag for performance observer - issue 81
Charles: Motivation is to register … notify for entry that happen before you registered. I landed a spec change on the dot observe flags and while queueing the entries in the spe. The ergo was a bit weird. The perf timeline will have a hook like other spec such as long task can register themselves. Define that type they are and can register the initial size of the buffer. The buffering logic will live in the perf timeline spec, wont be specific to the downstream spec. THat's the spec changes, clearing buffer and changing size of the buffer will only live in the resource timing spec. For legacy reason. That won't get upstream, will stay in the timing spec. The result that will change the web today, one of which is long task. Let's say we want to buffer a 1000 of them, that will never be clearable, we cap out at 1000, those will never be clearable. THe other new thing is that we were moving towards a model where new spec will be observable only, were changing the spec the new spec and future new spec, those will be observe in the …. You'll be able to get task by type.
Yoav: We could chose not to expose them. We could not allow access.
Tim: I would prefer not to expose those via getEntry.
Yoav: the part that I have PR that we will talk about that defines buffering logic. That buffering logic has two different options we discussed, its overly complex and requiring copying. Would be great if we can kill all that and barring that only have that complexity in resource timing. From now on buffer by default up to only a certain limiting. Performance observer will do the buffering.
Todd: Can someone declare a proposal? Let me explain my current state of thing, … other types of timing we should be able to get script….
Tim: I thought that for all types we want to access entries. The question is whether you access them via the performance timing or the buffered flag.
Todd: If we want to allow that we need the concept of buffering everywhere. How do you define the buffers and do you allow them to define early in HTML. MY question is where is all of this written down?
Yoav: Its in this issues (performance-timeline 81).
Todd: We need to sort of buffering to ship. THis is the core of performance boserver. You can have a bunch of UA shiping with someone thing buffered and not and not have it specific.
Charles: Today no one implemented the buffered flag.
Yoav: We could ship L2 and remove the buffer flag, add it to L3 with that logic.
Todd: Now that I'm clear on that i understand. In a world where we ship it with no buffering, performance timlineline and buffering will have to exist. Is that a fair place to ship specs, is that going to fulfil the goals and dream.
Tim: will let you do that you want, won't be pretty.
Nicolás: It's a lot of work to do the buffering.
Yoav: We're changing the model. Current model works for most use cases other than long task. Since we can't get long tasks from the past.
Todd: With long task you maybe get … of them overtime. It's an anti buffer argument for long task.

Tim: We got overaggressive in shipping it before the buffered task was specified.
Todd: Core gap for analytics vendors, not able to get long tasks for startup.
Charles: Specifically we don't want time….
Yoav: We should not block on pretty, we should block on working.
Tim: my concern is if we ship, should we be able to get entries for long tasks? I'm with todd that it seems bad. Ideally we would use long task as an opportunity to guide people to use the buffered flag. If we don't want to ship on the current state the only other option is to postpone shipping the spec.
Mstange: I think that's desirable. What makes adding buffering for all of those a lot of work.
Nicolás: Because they behave differently.
Todd: When we say it's a lot of work, i don't think its lot of work, it's that these people have a big stack of work. Is that fair?
Nicolás: im not sure its not a lot of work, but its prioritization, it would take a while to get done.
Yoav: Would be blocking a lot of other things where we want… we decided that are hand wave-y specified. That is in L3 and we want to ship the functional version, want to close holes and wait for refactoring. We have functioning API with some gaps. Analytics providers are collecting the data, it may not be pretty, but it is working. I don't think we need to wait for perfect.
Todd: How does it collect first paint and ???. It is expose because we didn't get buffering done? My point is its explicitly the opposite of what we wanted to do with performance observer. Chrome has made a decision to ship sooner rather than implementing buffering. I'm just saying I don't actually care. I understand what analytics provider want …. Why don't we just do it (fix it)?
Tim: I agree with you, doing it feels the right thing, just concern how it might push other timeline. I'm leaning that we should block on them, need someone to do the work. Should be good at finding the prioritization. Raise it as problem.
Todd: If reasonable buffering shows up that it would be something implementation in safari or Firefox?
Yoav: Firefox is green. If we were to block on redefining buffer, would Firefox follow in terms of implementation.
Mstange: can't commit but I don't think there would be a problem. Yes it's in our interest to follow there.
Nicolás: How much would change with the new model.
Todd: As each timing is implemented in the browser when buffering is enabled. They would all have to buffer some count of entries when the page is loaded then stop when its reached.
…
Todd: long task is new has no existence until a perf observe subscript to it. It would need some concept of buffering.
Yoav: concern is overhead of measurement. Do we want to start buffering long task even if page aren't interested. Topic of header registration.
Charles: And its' not clearable.
Todd: I dont think its hard to define, but will take end-to-end work to define.
Yoav: I think we have the overall picture of where we want things to be. Disagreement is whether we make it right now or in the future. L2 blocking or not. I understand the argument of make it so, but they are many moving pieces that will…

Todd: if its for long task sub its fair to say no. If its for others it should not be in L2. We will need to pull the buffer flag. We will need to remove the buffer flag in L2, put it in L3 with all those changes.
Charles: There will be a gap for historical long task data.
Mstange: Seems reasonable for me, implementing the buffer flag means that turning on long task for each page load.
Yoav: We will need another flag for expensive measurements.
Charles: We will need a PR.
Yoav: Open the L3 branch and throw it all in there.
Charles: Im ok with moving it and having a copy somewhere.
Alex: from my POV that they wont think its an L2 blocker, I think we would want to implement it with the ability to clear this buffer and turning it on and off in the buffer even if its a draft spec in L3.
Yoav: buffer is okay, but clearing is a footgun where one vendor is clearing it for others.
Alex: leaks are also bad.
Yoav: don't disagree, we may need to think of ways where clearing could, time blocking can work without enabling one vendors to steal entries from another.
Consensus:
Yoav: L2 we want to remove the buffer flag form the observe calls, we reinstate for the L3 for that, otherwise this issue should not block the current behavior, which is functional but not great.
Nicolás: Can we assigned someone?
Yoav: Charles.

Nav timing

Yoav: One new issue filed 8 days ago we need to triage (87). When reloading the page there's navigation type entry which seems different in different implementation.
Todd: This one will be hard to add a WPT. It requires a dep on the URL bar.
Yoav: You can't test that in an iframe, manual test are a thing.
Tim: According to the spec it should only be a reload if its a JS reload.
Yoav: This is a chrome bug that will be fixed, its not testable, just not a blocker.
Tim: I'll link the bug in the issue and close it.

Yoav: Various L2 issues, again we have nav type (issue 85), should it be updated according to the current page. Should the iframe say reload vs navigate?
Todd: Do we have good tests for these things? Lets add sub documents for each of them, should be fairly quick to ensure browsers are behaving properly.
Mstange: All I can say is yes please.
Todd: We don't need to go deep on this.
Yoav: Need work and test.
Nicolás: what the expected behavior?
Todd: Does a sub doc always inherit the parent document. The answer should be yes? Should it inherit for xorigin iframe?

Yoav: I don't think we need to define that here. Definition in the spec point to the HTML and fetch, and then I don't know what nested browsing contexts do, we should do whatever is defined there. We need to go through a chain of links to find that.

Yoav: Issue 84: Need to make consistent have a single definition.
Yoav: Issue 83:
Todd: When you unload there's unload time for the iframe, should unload includes the sub document unload time.
Tim: Knowing xorigin iframe sounds like a leak?
Yoav: Sound like something we should spec. We need to find the definition and adding tests.
Todd: Glance and each one occurs in order, we just don't have a hook.
Yoav: Issue 65: Talked to marcos about this. There infra in place, will happen soon, boris will be happy.
Nicolás: Did they break over time?
Yoav: I haven't found why they were broken, respec changed the way links were done,these things can be fragile. TAlking to marcos he says there's causes where we used the wrong format. Respec doesn't make it easier unless you click on each.
Todd: The reason we haven't closed this since its 5 separate issues. We need to merge specs, should we use bikeshed. This is an uber issue.
Markus: Close this one since all links are verified.
Yoav: Verify, make respec a local tested.
Todd: Can anyone verify?
Markus: I can do that.

(scribe: cvazac)

Resource Timing

Yoav: 2 issues, both around resource timing buffer.
https://github.com/w3c/resource-timing/pull/163 is about a secondary buffer and moving entries to the primary. But, if entries come in during that moment, or if developer doesn't increase buffer size or clear, then entries get dropped on the floor.
168: accumulate entries in secondary buffer, when copying over to primary, fire the bufferful event synchronously in a loop, while there is room in the primary buffer. Will fire in a sync. Loop while developer is making room for them, else will drop entries. What's currently implemented in webkit is a bit harder to explain in terms of spec processing model. With these two models, let's weight complexity over dropping entries.

Ryosuke: the problem with (one) approach is that you can have too many entries before script gets a chance to execute.

Yoav: if buffer size is small, yes. Entries accumulate and overflow secondary buffer.

Ryosuke: if we have default 250, if you get 600 IMG elements, 100 entries will get dropped.

Yoav: yes if that happens in between the time we queue the task and execute the task.

Ryosuke: this is an issue whether dev changes buffer size or not. Problem is, we drop before script even has a chance to do anything. We want script to be able to react to the full event. Why not just set buffer size to be bigger?

Todd: focus! :)

Tim: in the second approach, repeatedly fire event synchronously. Ah, they can be queued, but not…

Todd: primary buffer gets full at 200, more resource timing entries coming in in the background.

Ryosuke: if primary buffer has space, entries go to primary. Otherwise secondary.

Yoav: PRs 168 & 163

Todd: all members of the WG need to be able to see the status of both proposals

Ryosuke: [summarizes two proposals]

Yoav: one proposal: secondary buffer is unlimited, fire event sync until secondary not empty. Only drop entries if buffer size isn't changed / cleared.

Markus: why not have zero sized primary buffer if secondary buffer is unlimited?

[clarification here]

Markus: this is the same as having a 2x buffer, and firing event when it's half full

Todd: problem is that we implemented it a certain way, need to find something that works now

Yoav: webkit implemented their processing model, other browsers are firing event sync, guarantees that you can't have more entries come in, until the event callback has fired.

Todd: we don't implement event, we just allow setting buffer size

Ryosuke: we had to change the way we we do things for security reasons

[some security items mentioned here]

Markus: [to check with current Firefox impl]

Yoav: having drafted these two, the outcome of this is just spec changes to the RT spec, no other spec. So I’m fine with either one.

Todd: this same type of problem will come up with all buffered entry types, even user timing

Nic: but this is the only one with a bufferfull event

Yoav: with the L3 timeline updates we plan on doing, this is easier because we just drop when limit is reached, and you should have had a PO registered

Ryosuke: edge doesn’t support this event? Weird.

Todd: yeah, we just increased our buffer, but no stats. We think it’s not yet an issue.It’s probably 500 or 100.

Markus: firefox bufferfull event is synchronous

Todd: is full buffer a bad thing?

Ryosuke: you can’t stop javascript from taking more memory, nothing preventing users from inserting 1000s of dom nodes

Yoav: in 168, you could define unlimited as 5x or some big enough number to have a cap, but ensure that we really don’t drop entries.

Tim: someone who doesn’t care about resource timing entries has no idea that they are using this memory

Ryosuke: but if we fire the event and no one does anything, we drop the secondary buffer

Todd: if we increase buffer size by 1, will that…? [Receives clarification]

Nicolas: secondary buffer should always be empty at the end of the sync loop

Todd: this is a safe design, pages can’t mess this up. Outside of the problems PO solves.

Ryosuke: Problem is, firing the bufferfull event itself can add more entries (sync XHR, sync fetch image from catch)

Todd: don’t blame UAs for not conforming to specs that they weren’t consulted on

Ryosuke: example: when you fire event, callback might insert background image, image might be cached, would add another entry

Yoav: according to spec, that image req should be async

Todd: background downloads _could_ be completing, and put into the timeline.

Tim: is queuing an entry sync/async?

Ryosuke: queuing an entry for the PO has to happen sync. At end of microtask, that's when we invoke observers.
Todd: resources _could_ inject themselves magically during that moment.

Yoav: sync XHR, everyone loves! :) We need to reexamine the secondary buffer, copy things from it, don't assume it's empty [impl. Details…]. Otherwise, we agree to go with the 168 design and add the tests and modify the impls.

Todd: and we should use sync XHRs in the test!

Next issue: clarify when an entry is added to the buffer.

Yoav: it's not spec-ed if sync or async. Given discussion, don't know what's right call. Previously on call, we said sync. So we can know that the entry is there when onload for the resource fires. But some UAs inject off thread, so is this viable?

Todd: i assert that these should be synchronous. Specs shouldn't encourage suboptimal behavior.

Markus: if you listen to load for a resource, you'd be able to get PO notification before the load event fired.

Todd: if you call getEntries consecutively, new entries could be added in between

Ryosuke: we need to integrate this into fetch. It's not speced when load event finishes.

Yoav: yes, tie into fetch integration.

Nic: we've seen two things, 1) if last element of page, in the load handler, sometimes the RT entry isn't there. 2) in edge, sometimes the responseEnd of the RT entry might be zero (incomplete). Weird oddities with this being inconsistent.

Ryosuke: What do you mean by RT entry not being there?

Nic: main page load event, it might not have the RT entry for the final resource that was loaded, causing the load event of the page to fire.

Todd: the spec ambiguity leads to bugs? You'd prefer sync?

Nic: yes, during page onload, i'd like all entries to be there. If image delays onload, it should be in timeline during page onload handler.

Yoav: and this would be better for testing, more deterministic

Gilles: so you setTimeout? But do the timestamps still lineup?

Nic: yes they line up. Also, for xhrs, in onload handler for XHRs, the XHR might not be there.

Tim: if this better or worse with PO v. timeline? We are trying to motivate people to PO, but it doesn't help you.

Markus: issues in firefox?

Nic: I will check.

Markus: we submit entry and then fire event, so they should be there (in firefox)

[clarification here]

Yoav: a) we should define this and b) we should dispatch sync and c) to define this, we need fetch integration, so that's L3

Tim: cheap hand-wavy solution could be…

Ryosuke: but not every subresource blocks onload

Yoav: i'm talking about its OWN load event. IMG.onload, the entry needs to be there.

Todd: because IMGs are weird between full download and onload event…

Yoav: onload should not fire on image dimensions. That happens much earlier. But - let's skip that discussion.

Yoav: so in general, we agree, when onload fires, entry should be there

Ryosuke: resource could be finished before trailers are parsed.

Yoav: but that’s only server timing on firefox, right?

Alex: supported - but not used (webkit)

Ryosuke: respondEnd…

Todd: right, is that the body or the trailers.

Ryosuke: yes, it’s weird to wait on the trailed to fire onload. Or in terms of calc. responseEnd time.

Yoav: that is a separate issue, we want to say: “those entries are added when the resource download is complete”, then define “download complete” when we have fetch integration.

Ryosuke: hand wavy in L2 is not what we want, maybe an informal note, otherwise wait for L3

Todd: we want massive rewrite to be in L3, so nothing in L2 about this?

Ryosuke: informal note is fine. Also, not all elements have load events.

Yoav: I want to talk about dispatching entries sync, not load events.

Ryosuke: useless to add dispatch entries sync, if we don’t define download complete.

Takeaway: Ryosuke to summarize Issue 82 in the note.

Third issue: Navigation Preloads can have negative times

Yoav: should we save this for when service workers people here? A step or two might be out of order, from fetch spec, seeing some negative numbers, chrome impl, as big as -100ms.

Tim: why is that bad?

Yoav: yeah. We have historically avoided negative numbers.

Nicolás: maybe just add a note to the spec, sometimes timestamps can be negative.

Yoav: does the group have any objections about this. Nic?

Nic: we do data validation, might be throwing out anything with negative numbers. I will check.

Yoav: you throw away, does it break stuff?

Nic: depends. If WG allows negative, we will take that into account in boomerang.

Yoav: either we delay navigation preload by 100s of ms, or we redefine (some steps of fetch) - but not sure we can, adding note is reasonable, unless people thinks this breaks the world.

Ryosuke: but navigation preload is opt-in. So that makes it ok? (to shift the time earlier)

Facebook fella: you can have a preload, but once a service worker starts up, it can fetch something else that won't line up…. (with time origin?).

Ryosuke: only time origin shifts

Tim: do we decide what TO is, just pick lowest timestamp we see?

Nicolás: that will break….

Markus: making some numbers bigger makes things bad, and the negative number is actually correct.

Yoav: [mentions wanderviews's comment in thread]. Negative is the better option.

Nic: one other issue, tracking the onbeforeunload event, and that would have been negative time, but we avoided. But if we're ok with negative numbers, that might open some things up.

Tim: if we see breakage, what would we see? Analytics packages throwing exceptions

Yoav: collection code dying

Nic: collection code but also backend validation, and that we wouldn't have visibility until someone noticed that beacon counts were low, might never know

Tim: [approves]

Yoav: everything else in RT is just legwork - motivated people are encouraged to help Yoav with it.

Page Visibility

https://github.com/w3c/page-visibility/issues/34

Todd: this is just waiting for me, I just need free time. But - not before 11/5.

Nicolás: 11/6th it is!

Todd: [will do the best he can]

Tim: Shubhie could maybe do this?

Xiaoqian: but is this not testable?

Todd: it is, just not in WPTs

Ryosuke: you can use WebDriver

Tim: there's an open bug for WPT to use WD, bug is still open

Ryosuke: even if you use WD, it's tied to specific platform. But, manual test is ok. We will survive.

Todd: one step at time, let's update the text, check current tests, what manual tests do we need, then can we automate the manual tests? Those are the steps.

Ryosuke: [details manual test]

Todd: spec will say MAY, we are not requiring every browser to ….

Ryosuke: there exists platforms where a test is not really possible

Xiaoqian: is this not even normative text?

Todd: there could be existing browsers that break the normative text already.

Xiaoqian: if we are able to test the obscured window…

Todd: yes, we could test who has implemented this

Alex: a custom a11y tool perhaps?

Ryosuke: we all agree that this doesn't affect spec status?

Todd: if this is only adding a MAY…. this might be us just republishing a PR. might require a MAY, get thru MUSTs and MUST NOTs

Yoav: that’s one thing, other issue is adding hooks into where hidden and visible actually happen so other specs can use those definitions, which is editorial, doesn’t require new tests. This exposes the definitions, even if they are badly defined.
https://github.com/w3c/page-visibility/issues/40

Tim: defining the hook is trivial, whoever wants to use the hook should just do it. Is punting reasonable?

Ryosuke: this is completely editorial.

Todd: anyone can take this from me.

requestIdleCallback

Yoav: #70: but what is “idle time”? That requires HTML PR #4104 to land, but I couldn’t find a committer to get this merged quite yet. So, we are good to close this when the HTML one is merged? Also, will the HTML change also close 71?

Ryosuke: [looking]

Tim: doesn’t appear fixed to me. “Idle period deadline” isn’t defined.

Ryosuke: PR fixes when ambiguity around when algorithm runs. But, in step 5… With iframes, each document has different timers….

Yoav: cross origin iframes aren’t on same event loop (chrome only)

Ryosuke: window.open, can have timers in there (or same origin iframes)....

Ryosuke: AFAICT, we need same refactoring in HTML spec, event loop needs to be aware of which tasks to invoke… The times are part of the global object, but there could be multiple global objects. We can’t say “we have no timers on my global object” and say that means we’re idle.

Ryosuke: In the same place we’re modifying in HTML spec, we need to specify time when next task will run. Does that make sense?

Yoav: I’m pretty far out from this.

Todd: yep, this makes sense, but editor isn’t in the room. But this aligns with blink same-origin, so maybe this won’t be controversial?

Takeaway: Ryosuke to comment in case.

Ryosuke: [type type type]

Long Tasks

Nicolás: to begin, what is the status of LT for other browsers?

Tim: can we do anything from spec perspective to encourage other browsers to implement?

Markus: we are in progress

Todd: we want “all tasks” to be complete, not just callback tasks

Nicolas: I already merged that to HTML spec, so we have micro tasks included in duration, and “update the rendering” step. And we include work outside of the event loop.

Todd: that is what we’d like to see. We intend to implement after we get to PO

Tim: we don’t have (microtask?) implemented.

Ryosuke: I’m looking forward to hearing success stories for people who got benefit out of this.

Nic: we use for TTI, similar to definition of Lighthouse and WPT, but we’re RUM, so don’t have all the data. This is for busy-ness of main thread. Also - diagnostics. Customers can look at our waterfall, all RT data, page characteristics, long tasks overlayed. Customers see periods of time with no network activity, but LTs also help to explain that. Gives customers visibility into what’s going on in the page.

Ryosuke: but that’s just collecting information. But how are they using that to fix problems?

Nic: getting the data is the first step. Our LT instrumentation allowed a customer to see a 5 second LT, see what happens before/after, encourage them to break up script. Also checking in page interactions. Rage clicks.

Todd: have worked with internal MSFT sites, they use rAF w duration, which kills battery. In browsers that have LT, the register a PO, and then they measure the impact of third-party ads, and when they are interfering, or when own scripts are causing LTs. They were using LTs to blame ads, but they can’t use it for that anymore. They would kick out ad (vendors) when detect LTs. Perhaps FB uses as well?

Benoit: nope. Maybe for non JS issues.

Todd: yes, your instrumentation is so good, LTs not necessary.

Benoit: because not as useful without the attribution because we have so much code.

Todd: Ryosuke wants to know the tangible benefit.

Gilles: we want historical LTs, buffer all the things! We believe the bottlenecks happen before we register. Also, we let some users to write script. We want to measure that terrible javascript, but we can't register in time.

Todd: Ryosuke, you are waiting to hear that this is super useful before you implement.

Gilles: we need our measurement to be not invasive, loads late

Nicolas: other issues are how to make LTs more useful, to find the culprit, attribution and call stack.

Tim: punt because it's the same issue as the self profiling spec for getting stacks

Tim: one proposal: identify something as part of rendering pipeline, javascript, other browser work.

Nicolás: event loop, render, non event loop - this should be trivial to add to LTs, would be good first step.

Alex: for us, not clear break between render and "other" tasks

Todd: in Edge, we've been putting markers in our code in accordance with HTML spec, for consistency

Ryosuke: task could be compute scrolloffset, most time would be in layout

Tim: in the HTML spec, we could put start/stop markers

Ryosuke: example: on a page, there's a mousemove event that forces sync layout, will never see time spent in rendering.

Todd: time would be in mousemove event? It'd be ~50ms, you'd see LT for that. If you got callstack, you could diagnose the sync layout - in some sort of imaginary far off world.

Tim: if we were to do attribution, we need more than the three categories

Todd: some devs are ready to build on this model. If you knew “rendering” it’d give you more info.

Benoit: yes, that would help - but difficult in practice.

Todd: both of these approaches help different devs.

Tim: worried that going from coarse to specific, there isn’t really a migration path

Ryosuke: this is sort of an expert feature, and yet, it could be misleading to get a certain type of LT, especially if the type is happening because of something else.

Yoav: maybe this is only useful for devs that are used to tracing with Chrome devtools. Useful for some people, but how far from are we from actual attribution.

Tim: I agree that this is low priority

Todd: but my sites that use rAf instead of LTs… and my devs have found limitations in chrome impl. They want to know about ANY long tasks. Even if there are things not from our script.

Nicolas: this isn’t about when we are firing events, we are talking about entry classification

Todd: if LT fired all types of things, they’d throw away the rAf impl. They would be happy with unnamed/untyped entries.

Nicolas: we currently just call them script. There’s a bug, we don’t expose microtasks. We don’t consider the time of the microtasks. Should be included. Just a bug.

Todd: GC 60ms will cause LT of “script”? (Yes.)

Yoav: rename “script” to “unknown” (or “unclassified”).

Todd: still want to see all the long things.

Ryosuke: let’s break!

Tim: next steps: don’t call them “scripts”, low-priority: let’s track better attribution at some point, also classification.

NEL & Distributed Tracing

- Douglas Creager:
  Slides
  NEL overview:
  - Looking at how we can collaborate better with distributed tracing
  - Without NEL:
    - End user makes request to server
    - Server logs things about request
    - BUT! If the request doesn't make it to the server, then we don't know anything bad happened.
  - Instead, we can collect information about failing requests in the client, and then send this data up to some collector, which aggregates information about failures.
    - The collector needs to be super resilient.
  - Security & Privacy:
    - Only report information the server already has access to.
      - E.g., can't expose which DNS server is being used for privacy reasons.
    - Server administrator controls whether reports are gathered for their domain.
  - Integration with Distributed Tracing:
    - Request from Comcast and others.
    - Have the NEL policy header contain a request to include arbitrary request & response headers in the NEL reports.
      - This enables requesting that the trace parent header is included in NEL reports.
      - Could also ask for etags or cache-validation headers.
    - Syntax doesn't seem too controversial.
    - We're primarily worried about security & privacy.
      - The server does already have the headers, so this should be fine.
- Morgan Maclean: Do we mean client side tracing?
- Doug: ?
- Ted Young:
  - If distributed tracing is initiated in the browser, and the request never gets to the server, this isn't super useful.
  - If we want to collect distributed tracing but not have a security nightmare, could the server talk directly to the collector?
- Doug:
  - We've tried to split out reporting uploads and NEL.
  - CSP & others are also working on using the reporting API for this.
- Ted: This would be more necessary if more of this ran natively in browsers.
- Doug: This would factor out needing to think about delivery.

- Yoav: What kind of traces are we referring to?
- Ted: Analytic information - the user does a sequence of activities, making requests etc.
- Alois: When we talk about traces:
    - Grouping of interaction that started somewhere, with some sub-interactions, or "spans".
    - Traces that start in the browser: span's based on user timing, or XHRs.
    - A trace is a hierarchy of spans which may come from different nodes.
- Yoav: In JS, using existing performance APIs, you'd construct spans yourself.
- Ted: Those APIs don't have hierarchies. We'd ideally have nested blocks.
- Yoav: We want to use current mechanisms, potentially require L3. But we'll need to write the logic to produce the trace in JS.
- Alois: Trace-Context & Trace-State. Trace-State lets you add extra information.
- Ryosuke: How does this relate to NEL?
- Douglas: Yes, we've moved from discussing NEL to one use-case that this would allow.
- Yoav: The use case is adding the trace id to NEL reports?
- Douglas: It's open to figure out how distributed tracing will work in the browser. To avoid depending on how distributed tracing will work, enabling arbitrary headers will let us integrate no matter what.
- Yoav: Let's try to wrap up the discussion on this issue, talk about server timing, and then get an overview on use-cases, primitives, etc.
- Todd: If the only use-case is for distributed timing, why would we do this first?
- Douglas: Other use cases for this (NEL) mechanism:
    - Anycast IP addresses - IP doesn't indicate which location served a request. This could be used to indicate where the request was served.
- Yoav: So this would give you the last server that actually worked.
- Ryosuke: We're talking request/response headers.
- Douglas: The browser chooses an ID which is sent via a request header.
- Ryosuke: This is an entirely new feature?
- Ted: Distributed tracing as it currently stands doesn't benefit, but as we improve distributed tracing, this will become important.
- Yoav: It sounds like we want to split out these concepts.
- Ryosuke: In the case where we report a response header, what do we include?
- Doug: The browser will receive a response from the server closest to the user, and reflect any response headers into the NEL reports.
- Ryosuke:
    - I load A.com, which fails to load B.png (on a separate origin).
    - What header is sent, to which collector?
- Doug: No report is sent to A.com.
- Ryosuke: I load A.com, which fails to load B.png (on the same origin).
- Doug: We'd receive a success report for A.com, then a separate report for B.png.
    - If the server asked for a specific header, then whatever etag header came in the index.html response would be reflected.
    - The failure report wouldn't include any headers.

    - This is all isolated to individual network requests.
- Tim: Is this useful without distributed tracing?
- Ted: Yes.
- Doug: Ilya commented that we should elaborate on other use-cases.
- Nick: From a distributed tracing standpoint. This is useful, if distributed tracing headers are sent with requests.

Server Timing

- https://github.com/w3c/server-timing/issues/42
- Charlie:
    - Server timing items can be separated by a comma
    - We try to keep parsing when we can
    - On tests, Chrome & Safari fail two degenerate cases where we don't keep going.
    - Can we be more strict with the parsing strategy?
- Yoav:
    - General philosophy question: should we parse slightly invalid input.
- Tim:
    - Wouldn't we break some stuff?
- Yoav:
    - Some server aggregation, yes.
- Todd:
    - These folks are already somewhat broken.
- Charlie:
    - If we were to do it over, we'd only allow sloppy whitespace.
- Alex:
    - In general, we should be as strict as we possibly can.
    - If something doesn't fit, completely discard it and fail horribly.
    - Otherwise we encourage development which takes advantage of a quirk, and we get pressure to become forgiving for the sake of compat.
    - This makes an unruly parser.
- Yoav:
    - The processing model is well defined, it doesn't encourage a slippery slope of becoming more forgiving overtime. It's forgiving in well-defined ways.
    - I agree we should be strict.
- Yoav:
    - Anyone disagree on being strict?
- Charlie:
    - The backstory here is that they were passing in Chrome and Webkit.
    - Mozilla made a change and upstreamed, breaking tests in Chrome & Webkit.
- Todd:
    - How many months has this been shipped?
- Charlie: 7-8 months

- Todd: The question is how much is this depended on?
- Alex: How long has it been shipping in FF?
- Markus: 3-4 months
- Charlie: We could add a use counter.
- Yoav: Yeah, we should do that. How used is server timing today?
- Todd:
    - We can drive the decision from the data. If the data says we can make this stricter, we should do it, otherwise, we shouldn't.
    - Anyone disagree?
- Charlie: Less than 1.8% of sites use this.
- Yoav: Any other issues?
    - Server timing in the context of distributed tracing.
    - Distributed tracing reached out to us because server timing seems potentially relevant.
- Alois:
    - Brief Primer
        - If Chrome requests something, DT creates a span.
        - Server timing today has timing data. DT wants to provide a unique ID.
        - Trace Context wants to add trace context data to server timing.
    - We want to provide an arbitrary string.
- Charlie:
    - Someone creates a GUID, this is sent to JS, and that is shared with whoever needs it.
    - Or does it need to be fed through other requests.
- Alois:
    - For now, this GUID wouldn't need to be forwarded.
    - Two uses cases:
        - Browser starts a trace. (We're not talking about this case)
        - Server Timing responds with trace context ID.
- Charlie:
    - Where would this data go in the server timing header? |description|?
- Alois:
    - Systems we want to trace are becoming complicated.
    - Tracing providers have for a long time sent a header with a unique ID
- Yoav: What's the parent ID if the browser sent a request?
- Alois: Just a random ID.
- Ryosuke: What is a parent/child relationship in this context?
    - These are different hops?
- Alois: Yes.
- Alois: For each new request, you create a new parent ID.
- Yoav: So the root of the tree would have a null parent ID?
- Alois: No, it would be a GUID.
- Alois: You can do all this today.

    - But if we have 3 nodes, one of which doesn't understand the header this alls apart.
    - 2 different implementations may not use the same header.
    - We're proposing: Trace-Parent, Trace-State.
    - We just need to agree on the header, to improve co-ordination.
    - Trace-State contains additional information that individual monitoring providers require. (basically just a blob that gets propagated).
- Yoav: If the browser passes an ID, it's the root of the tree.
- Ryosuke: What's the benefit of including the browser?
- ?: If a browser is misbehaving we could identify this.
- Yoav: So the benefit is that this information would be available to JS in the browser, and analytics could communicate their parent ID.
- Alois: folks already do this via Cookies
- Yoav: Does server timing allow you to address the same use-case?
- Ryosuke: I understand the use-case, except for sending the header in the browser.
- Alois: To see the whole trace from end-to-end.
- Yoav: But with server timing, you can do this without the browser initiating the tracing context.
- Alois: Example: Make 2 XHR requests.
    - If the browser starts the transaction, we could associate the 2 requests.
    - Transaction-Id changes every request, Parent-Id doesn't.
- Yoav: Server timing should still be able to associate two requests. Make up your own parent id / trace id for the collection of these requests.
    - I don't think we want the browser to start sending an ID on these requests.
- ?: This is a somewhat different use case. Instead of just correlating with immediate request the browser made, we want to correlate with background services.
- Yoav: You can do this already with server timing. ~ explanation ~
- Alois: Maybe for resource timing, but we also want use timing etc.
- Todd: L3 User Timing will allow user-defined data. This doesn't explicitly allow hierarchies, but you could construct them.
- Ryosuke: you should be able to produce hierarchy afterwards.
- Alois: How? Time-based correlation? This would be heuristic driven and not work well.
- Todd:
    - Use case:
        - We've got some script, and we want to start a transaction, with a transaction ID.
        - We want to pass the transaction ID to network requests (XHR's, fetch calls)
        - What about other kinds of requests? Scripts / css. Do they need the transaction ID?
- Alois: This is a bit fuzzy.
- Todd:
        - What's the lifetime of the transaction ID?

        - Script often comes from multiple sources.
- Alois:
    - We want to enable either direction, starting from the server or the browser.
- Agreement: Having the server send these headers makes sense.
- Alois: Standard server timing header for this.
- Yoav: Or server timing convention.
- Todd: The concept of a transaction doesn't make much sense, because work is interleaved.
- ?: It's more of a user session ID.
- Yoav: This would leak information.
- Todd: Yeah, this would need CORS protection.
- Yoav: An anonymizer proxy wouldn't strip these, make me less anonymized.
    - Just looking at user timing, resource timing, server-timing, if you sort these, isn't this enough?
- ?: How do we correlate timings with a user?
- Yoav: If you include your tracing context when you upload timings, this should work.
- Ryosuke: The browser doesn't have enough context to do this, but a framework should know.
- Ted: A framework should know.
- Ryosuke: Maybe a site would know.
    - This can't really be a hierarchy
    - You could just report all of your performance at once to a server.
- Yoav: you could have a convention where JS applies a header.
- ?: Does resource timing indicate if an asset was served from a cache? I.e., not from the origin.
- Yoav: No.
- Todd: NEL reports failing requests.
- Yoav: If the failure is complete, reported by NEL, otherwise in resource timing.
- Doug: Successful requests also show up in NEL (sampled).
- Yoav: If you're not sending a session ID, but are sending a request ID, that could fly.
- Todd: Then in JS, you'd combine your request IDs with your timing data, and upload that.
- Todd: You'll still run into CORs preflight costs for every request.
- Yoav: In order to make sure that the server is supposed to be exposed, for every non-safe request header, send an extra head request ahead of time, getting permission to send the non-safe headers.
- ?: But what if we can get them standardized?
- Yoav: Even if they're standardized, getting them on the safe-list could still be hard.
- Yoav: This should be opt in, with the bare minimum number of resources that you need.
- Ted: For same origin - this is fine.
- Yoav: For request headers, even without CORS, there are cases where we run into header name collisions.

- Todd: We aren’t saying this is impossible, we’re just saying it’s hard.
- Todd: Was this the discussion you were looking for?
- Ted: It sounds like we need to go try to leverage what’s available.
- Yoav: I’m happy to help moving forward.

(Scribe: cvazac)

Paint Timing

Issue 29:
Nicolas: what happens with invisible text, whitespace, or a slow loading web font
Tim: we wouldn't want to trigger a paint event, if it was invisible, but in practice this is harder to do. So the question is, can we specify this in a way that a browser can ATTEMPT to exclude whitespace, but maybe it would fail.
Yoav: does this expose a way to game the paint timing?
Tim: this is already possible
Yoav: does include invisible text change the gameability
Tim: we aren’t worried about that, we are focused on well-meaning web devs. Paint is a lower bound anyway.
Benoit: can we even define invisible text from a spec perspective
Rysokue: if there’s no box, then its invisible… or collapsed whitespace. There are some clear cases. We keep metrics for init. painting. Turns out, we weren’t painting at the right time, because of whitespace in the page. If you have space between tags…. Ideally we’d have a concrete definition.
Tim: strawman: if known whitespace chars….
Todd: need to explicitly define whitespace
Tim: we aren’t talking about an invisible D for one particular font
Todd: are things that are characterized as whitespace, are those paints? CSS attributes applied to elements, does that control a paint. Or - what about size=0. There are a lot of corner cases here.
Gilles: shouldn’t you know based on rendering pipeline, did I draw something or not?
Everyone: yes in theory, hard in practice, disparate parts of the system
Tim: we could specifically enumerate corner cases
Yoav: is this really a problem in real life?
Rysokue: yes!
Todd: [summarizing webkit/Rysoke issue] - Did chrome also learn about wrong paint times based on your internal analytics?
Nicolas: in general, this isn’t a problem for chrome. We don’t think that changing definition of whitespace moves the needle that much.
Todd: we don’t care about averages, we care about if anyone moves
Tim: should we gather the data to find out how much this matters?
Todd: yes

Tim: ok, let’s by-hand analyze every page. We can only do a diff if we implement the “fixes”
Yoav: will WPT help? Differences between start render and first paint?
Tim: design this later, maybe first contentful paint might be better
Rysokue: [to identify some of their whitespace cases]
Alex: whitespace only drawing - we should definitely consider that. But, white or invisible text - that’s meaningful. The developer did that. We should report on that.
Alex: what if there is no content?

Next item, Issue #20: (clipped contents in first contextual paint?)
Yoav: please describe this for us canvas-challenged folk
Tim: we are talking about drawing to a canvas versus trying to draw to canvas
Todd: [clarifying] if we paint to canvas, and it’s white…. that’s still content?
Rysokue: because SVG is a vector format, determining…
Rysokue: we should just assume that canvas has a paint
Yoav: empty canvas that gets drawn on later shouldn't be counted as a paint?
Everyone: this is common
Markus: firefox, we know about canvases that haven't been drawn to
Yoav: yes, this isn't for SVG, other than gaming, and forget those people

Next issue, #19 (exact timing of a navigation)
Nicolas: this is just a misunderstanding, probably just a misworning of the spec
Todd: we can eliminate "before the navigation"....
Rysokue: what would be the definition then? when the browser first rendered in this browsing context?
Dominic: no, this doesn't work, context is created when you open new browsing context
Markus: then how is the time origin defined?
Todd: let's look at hr-time spec! section #3
Yoav: [reading spec aloud]
Todd: technically, let's just point to this hr-time spec
Nicolas: our convention is: diff between now and origin, ...
Todd: but when do we record this, per navigation? first navigation?
Yoav: first rendered after what? first after time origin is zero?
Todd: is it navigate? (link to hr-time)
Todd: I assert that first paint needs to be relative to navigation (HTML5)
Dominic: just ecluse first two cases?
Todd: i’m concerned about unload case
Rysokue: we are getting to the can of worms, navigation is complicated, to pinpoint start render, for different browsers) [is going to be a problem].
Yoav: we are trying to determine WHEN we want to measure first-paint, are we able to identify the lower bound?
Todd: the trigger - next render after this - is going to be first paint
Rysokue: that’s too early

Alex: could be animating from previous page
Gilles: [suggestion]
Todd: can't see timeOrigin from previous page
Dominic: don't say any paint at all, paint on the document
Todd: primary document first paint event
Rysokue: need to check that, is this rendering happening for THIS document that we just navigated to
Alex: this could be a good starting time, but we will need to define when we start painting the new page, as having come from data received…
Dominic: if you are updating the rendering for the document that i care about….
Todd: so all IFRAMEs get their own first paints?
Everyone: yes.
Rysokue: so that needs to be fixed.
Tim: proposal: first paint, in the document we care about, after time origin is set.
Everyone: yes.

Friday Minutes

Morning Agenda:

- Preload
- Input Timing
- Prefetch
- Long Tasks

Preload

Kinuko: srcset / sizes for images
https://github.com/w3c/preload/issues/120
Kinuko: For responsive images
Yoav: <live coding, nothing can go wrong here>
https://github.com/w3c/preload/issues/120#issuecomment-433307060
Kinuko: Attributes like imagesrcset="" and imagesizes=""
Yoav: In these cases all images are representative of the same image, just in different resolutions
Kinuko: This won't capture <picture> tag cases though
Ryosuke: I presume this would have the same handling as the <img> element
Kinuko: yes

Yoav: In markup this could work, but if Link tag is delivered via HTTP header, the browser possibly doesn't yet know the viewport dimensions, so I'd "hold on" to the preload until the dimensions are known
Gilles: Could this still be fetched prior to HTML parsing on a mobile device because the viewport is better known (fixed)?
Yoav: Markup such as <meta viewport> can affect the viewport size
Tim: Having the viewport tag appear before the preload tag might cause developer or browser ergonomic problems
Yoav: Processing model could mention that preloads might not be triggered until the browser has finished receiving the bytes of the <head> (so any viewport tags can be read)
Yoav: No major objections with this, seems like this is a use-cases we should tackle, knowing there are viewport issues we need to address

Yoav: Brings us to the other gap regarding images with preload, <picture>
https://github.com/w3c/preload/issues/131
Yoav: Picture's sub-elements, whichever is the first sub-element that matches (media attribute), triggers the actual download
Yoav: One way is to have your <link rel=preloads> have mutually exclusive media="" queries so the browser will only pick the "one" that matches. Puts a lot of burden on developers. Not easy to do.
Yoav: <picture> also allows type="" attributes (e.g. for webp), which isn't in the <link rel=preload> case
Yoav: Alternative proposal is to somehow declare multiple preloads as a "group", which the browser picks only one, in a similar way to how the browser picks a <picture> element
Markus: Could we have a new top-level grouping element like <picture> is?
Ryosuke: This won't work for Link HTTP header-based preload
Todd: Could we put the actual element in the head, with some sort of attribute saying "don't use this yet"? Not sure this a strong option
Ryosuke: <imgs> are more straightforward because it will always be shown, whereas <picture> elements can be affected by orientation, viewport, etc. Not sure developers will be able to understand how to best configure their preloads in <picture> cases
Yoav: One use-case is for hero images, which might be in a <picture> element, and they want this image kicked off earlier
Gilles: Very similar use-case for Wikipedia logo
Ryosuke: Developers could add an invisible/1px div that has the same <picture> element in the top of the document so that it's loaded earlier. Avoids duplicating everything <picture> element does in the <link rel=preload> element.
Yoav: Sounds like we all think this is complex idea. Are there concrete use-cases from customers?
Yoav: Opinion is we should go ahead with srcset for imgs, punt on this for now, until there are more concrete use-cases. Developers can use media-query hacks today.
https://github.com/w3c/preload/issues/131#issuecomment-433314755

Input Timing

https://docs.google.com/presentation/d/1pxx3qMLqzcdtJ_o46XLCgE3uCSI0X_jwgqh-subIgWY/edit#slide=id.p

Nicolás: Proposal for Input Timing (was previously called Event Timing)
Nicolás: Timing about input events, better name
Nicolás: Provide web developers and analytics providers a way to measure input latency
Nicolás: Can do today, not performant or reliable
Nicolás: <examples in slides>
Nicolás: Measures mouse, point, touch, kb, input, wheel, composition events
Nicolás: Expose slow inputs (event handler duration), how long UA took to respond to input event
Yoav: How does that work for animated things, like a flyout. Does it measure the "first" time it is seen, or full animation?
Nicolás: First time it is painted
Nicolás: Measure queuing time (HW input vs when first event handler is called)
Nicolás: Slow events > 52ms
Nicolás: Future: provide event.measureUntil() for async work
Nicolás: Security: Same accuracy as performance.now()
Nicolás: Rendering duration is rounded (multiple of 8)
Todd: Is this the rendering timestamp (main thread) or the graphics-card display timestamp?
Nicolás: As close as possible to when pixels are displayed
Todd: There's talk of a post-render-steps callback, would that enable more attacks since that would give more precision
Tim: Had security review, but can sync with them again based on that
Todd: Combined with PaintTiming, can this help us understand the total input+display latency
Tim: This is focused on just slow input events, not all events
Tim: Lower-level primitive would be all paint events + all input events, but we can't expose all of that for privacy and security
Charlie: I'd also like a similar thing for the work based off the output of XHRs (e.g. DOM updates)
Markus: Likes this a lot, we want to measure from hardware timestamp to display. Rendering timestamps we get are from the start of the timeline and pages can take a long time (e.g. 200+ms on complex pages), want to make sure that duration is reflected
Ryosuke: Don't want to expose the timestamp of when rendering is done (security concerns)
Markus: What happens if input event doesn't cause any visual changes.
Nicolás: We want to still have the event measured, with a display timestamp "close to" the time when rendering might have happened (security concerns)
Falken: Make this a generic PerformanceEventTiming, esp for service worker events? Other examples included fetch, background API, etc

Tim: Spec is focused on input events right now, but this could be extended in the future to other events
Yoav: Could keep EventTiming as a parent interface for IDL, with InputTiming implemented as
Nicolás: Not enough to expose entries for slow inputs, need to know % of overall total
Nicolás: For each event time, number of events dispatched
Nicolás: <IDL explainer>
Tim: How to expose total event counts: as a # on the slow event (which means you'll not get info about non-slow events at the end) or on something like performance.eventCounts which might be slightly off due to async nature of updating
Yoav+others: Slight error sounds better than blackhole of missing events at the end
Todd: Why async update?
Tim: Some events will be handled on the main thread by JS, others not, and both need to be counted in overall total
Nicolás: Want to expose first input latency.   First input is always slowest. Don't show trivial inputs, so only key down, mouse down, pointer down+up, click
Nicolás: Exposed as a single observable event always
Yoav: Can I get the thoughts of UAs on desire to implement?
<UAs discuss there's not strong opposition>
Mozilla and Microsoft folks seems fairly enthusiastic about tackling the use case
Apple says it doesn't look awful, but early to commit.
Tim: Discusses why not just putting this information in Long Tasks (no concept of HW input timestamp, correlation with rendering, etc)

Prefetch

https://github.com/whatwg/html/pull/4115
Yoav: Looking into better defining prefetch
Yoav: A couple questions came along
Yoav: 1. How should prefetch behave when cache is double-keyed
Yoav: 2. How should prefetch behave with Service Workers (don't know which SW will service them)
Yoav: Need to set a bunch of flags (e.g. no-referrer, no-service-worker), etc. Do we re-use the "keepalive" flag (used for Beacon), which has some other implications like not caring for the response
Youenn: While there are similarities to needs of keepalive flag, there are some differences like 64kb limit, what to do with response if there's no current browser context, etc
Yoav: New "speculative" flag, which could apply different restrictions
Yoav: RE: ServiceWorker questions, should we have different behavior in double-keyed cache world for what goes to SW vs not
Todd: Can this be solved with some "prerender" cases where the core document is fetched and the UA decides if it wants to drive the parser, see preload tags, etc
Falkenhagen: How do you know if the prefetch is for a main resource or a sub-resource
Yoav: We don't know today, as="document"

Falkenhagen: There’s a valid use case for prefetching a worker as well
Alex: There are use cases for subresources in same-domain, but concerns about different domain in terms of privacy
Youenn: There are also concerns that developers will not get it right, so we may prefetch resources and they will not be used. Better to avoid footguns. Better to restrict prefetch to main document, and see later.
Kinuko: Browsers that don’t implement double-keying may prefetch subresources, and double keyed caches can just ignore them.
Youenn: That runs a risk of forking content. We may not want that.
Alex: There’s still a valid use case for subresources for same-origin. So we definitely need as=document or something similar. If we need to skip service workers, we’re doing the wrong thing.
Tim: Can be confusion if SW is not always hit
Kinuko: Can’t we have SW for same origin subresources
Yoav: not sure what the scope would be
Ben: This smells like Foreign Fetch and risky in terms of privacy. The conservative case would be to skip the SW for anything cross-origin. Concerned with letting prefetch running JS on their own origin. Even in the non-double-keyed-cache case. Would also apply to Chrome when disabling third party cookies.
Yoav: That changes things, so we should probably skip SW for all cross-origin navigations
Ben: We could apply same policies like storage partitions, etc, but seems to complicate the cases here. Conservative is to bypass SW.
Alex: Concerned no matter what outcome, will cause a mess. w/ SW a bunch of SW maybe aren’t needed or security implications. w/ no SW, and if developer only wanted to load something via SW now they can’t.
Ben: The cross origin subresource case shouldn’t really exist

Scheduling API

proposal / slides

Shubhie: websites need to be responsive, primary issue comes down to script executing blocking responsives (see slides).
Apps and sites that build their own scheduling libraries have achieved good performance that way.
Schedulers are build on top of native APIs (like rAF) and have built-in priorities.
Schedulers have to yield frequently *just in case* but they'd rather not have to yield if they had better signals from the browser.
...
Tim: ember uses microtasks for it's scheduler, how does that work since it can't yield?
Shubhie: they primarily use it for ordering
...

Tim: what does coordinate mean in the context of a scheduler?
Shubhie: handing rendering, pending fetches, input etc. low priority xhr responses don’t preempt my ongoing high priority work
…
Shubhie: what's needed from the platform
...since there's been alignment on handling of rAF, was there agreement on handling of input handlers cross-browser?
Chris: no agreement on input scheduling yet…
smaug: In some cases you want input to run before rAF
Shubhie: if rAF and input handlers are properly aligned cross-browser, that would make defining a scheduler easier.
…
default priority slide
Tim: is before work before next frame what we want for "default priority"?
Shubhie: Apps can't use rIC in some cases due to starvation, rAF
Tim: postMessage is an alternative to setTimeout(0)
Chris: that's a platform hack, why not setImmediate?
Yoav: idle until urgent pattern for setTimeout
Phil: there could be IUU with different priorities
Chris: priorities could get upgraded
Shubhie: what new primitives should we add to the platform to handle these use cases?
Option A:
A unified API for posting work + sematic priorities
Users can create their own task queue
Option B:
Standardized JS Scheduler
Yoav: a high level API would be useful, so would underlying APIs
Shubhie: they've not necessarily mutually exclusive
Shubhie: <top feature requests slide>
Ryosuke: who is asking for these features?
Shubhie: Google Maps, Ember, React
Tim: game developers are asking for it. They can't hit 60fps, so they degrade
Shubhie: developers also want a way for promises to yield.
Nate: agree
Ryosuke: how would you break up promises into multiple tasks?
Nate: more what we want is a way to break up promises chains and yield to input
Ryosuke: this would require a change to JS via TC39
Nate: most developers don't understand enough about the event loop to write code that yields to input well.
Yoav: and arguably they shouldn't have to
Tim: it's possible to polyfill microtasks that yield
Phil: with async functions it's not easy to polyfill without a build step
Priority hints

Ryosuke: what are the use cases?
Shubhie: maps wanted this for fetching tiles
Nate: logs shouldn't block other fetches
Todd: after multiple fetches do we want priority on which callback runs first?
Chris: no, priority for network resource use. But important to consider the resolution priority as well
Ryosuke: what is the facebook use case?
Nate: fetch a component that's not visible on the screen, this is lower priority at the time
Chris: there's a fear of putting something at low priority because you can't always easily upgrade.
Yoav: yes, reprioritizing is a key feature
Alex: changing the priority of in-flight or queued network requests is hard
Yoav: this may become easier with H2
Benoit: servers would test reprioritization logic if the API existed
Shubhie: should we poll the group for the need of these APIs?
Ryosuke: there are different concerns being discussed here, which specifically are we talking about
Nate: breaking part promise chains is critical for us
Chris: non-rendering, high priority work, is important but shouldn't need to use rAF (but still yield)
Shubhie: user-blocking and default priority are the main ones we need
Ryosuke: network request priority and handler priority cannot necessarily be inferred by the browser
Yoav: priority hints already help with network priority
Shubhie: yes, ultimately it's about providing hints to the browser.
Ryosuke: there may be priority inheritance problems, high priority request may not imply high priority response or callback

hasPendingUserInput

proposal / slides

Tim: this was previously presented as shouldYield…
[slides]...
Tim: how should we handle long tasks in a world where a developer is using hasPendingUserInput
Phil: input timing solves this problem to some degree
Benoit: is there a danger of yielding to the wrong task?
Tim: yes, once you yield who gets to go next is still a problem:
Benoit: a scheduler would help with that
Shubhie: why did the timestamp get removed? (time input has been pending)
Tim: are there use cases? Isn't input always the highest priority?

Nate: there are some cases where you need to finish work before yielding.
Ryosuke: events like mousemove and mousedown will likely have different priorities
Todd: originally proposed time, but maybe wait until good use cases arrive
Alex: this could quickly lead to API bloat
Yoav: maybe also include a reason for yield
Ryosuke: what about a mouse move with a hover effect, it still may be lower priority
Tim: sometimes you have work that should or should not block the next frame, and you might want to know if input is pending before making that decision.
Chris: task priorities can take that into account
Tim: this proposal is a low level solution that would make things better than they are today, higher-level things could be better but are harder to implement
...concern that the more we let UAs prioritize the higher chance we have to interop issues
Tim: we could take some time to look into what a more complex API might look like if there's concern with this API
Smaug: have we defined what pending user input means?
Tim: we have some spec work for this for sure
Benoit: could this lead to yielding multiple times per frame?
Ryosuke: on mac there's at most 1 event type per frame in most cases (mousemove)

FetchEvent Worker Timing

proposal

Ben: for simple FetchEvents, what we already expose in resource timing is sufficient, but for complex SW responses with strategies that read from IDB, etc we can't capture that time.
Like we do with server timing, we could have a worker timing header that we can write artibrary times.
Npm: why isn't user timing sufficient?
Ben: it doesn't get associated with the particular request
Nate: you can also have multiple tabs accessing the same SW, and you don't always know which one, so you need to match it
Yoav: we could have a fetch ID to make the matching, but that’s far-future and worse ergonomics
Nate: we try to do this measurement now, and it's very hard to get it right. This would be very useful
…
Markus: having a common ID for fetch events would be very useful for lots of reason, similar to yesterday’s discussion with the Distributed Tracing WG
Ben: some responses are not generated by servers...mutating headers in a SW is probably not an option for this solution
Nate: sometimes you don't respond from a SW at all, but you still want to measure the perf impact

Yoav: some responses are opaque and you can't modify them
Ryosuke: is this exposed in the fetch response?
Ben: the implementation of adding it to the fetch event vs. passing it to the fetch event can be handled in an issue, will follow up.
Ben: for cases where you want to measure what happens after event.respondWith, you could have the perf entry delayed until after the promise resolves, and it would still add timing data to that fetch
...User timing L3 could also handle this case if we can attach metadata to an entry
Charlie: I like this proposal as it solves a lot of the same issues I dealt with in server timing (e.g. duplicate names)
Ben: I plan to do some implementation and experimentation in Chrome in Q4

In flight resource request

slides
Npm: We've discussed inflight requests before. We want to expose information about requests that weren't yet completed.
Use cases:

- Allowing for find out if the network is idle and knowing how many requests there are
- Create a busy network indicator

Goal is to arrive to consensus on the need and decide on the shape of the API. Will present a proposal
Adding a ResourcetimingUpdate interface that notifies when attributes have been computed. Notifying only attributes that have reached their final states. So this will notify the beginning and the end of each network request.
In the future we can try to split requests according to redirects and could expose more details.
ResourceTimingUpdate includes entryType and attributesUpdated.
Fetch integration will need to happen as part of the overall ResourceTiming integration.
The fact that PO delivers entries sync means that some use cases (rendering busy widgets) may suffern from lags
Alternative: Modify PerformanceResourceTiming
Not backwards compatible and may be confusing to current users.
Alternative 2: Add event listeners when a request begins or when response received. Downside is that it would encourage early registration of event listeners.
Alternative 3: Create a FetchObserver
Could receive timing information for request/response, but not modify them (so not FetchEvent)
Tim: So this would recreate the buffering logic of PO in the Fetch spec.
Npm: yes, we would need to do that
Ben: I'd be very hesitant to do that
Yoav: Have you considered alternative 1, but with an opt-in? Notifying the observer twice per each entry? How does it work for navigation?
Tim: You need to poll it, you don't get updates

Yutaka: This will expose when requests fail. Would it be OK?
Tim: We would now need to expose failing requests, and that may be security sensitive
Phil: It doesn't look like most use-cases need such a complex API. Have we looked into just exposing the number of inflight requests?
Tim: Won't answer use-cases that want to filter requests by criteria (e.g. pending images, etc). We can't really expose it in a single number
Phil: Feels like we're trying to force RT to be high-priority
Todd: Use-cases are not sufficiently spelled out
Nic: We have 2 primary use-cases:

- Single page app - want to track soft navs, and want to know how long it took for them to be done. Currently use mutation observer, add event handlers and essentially build our own onload timestamp. So instead of being notified on downloads directly, we track elements that get added to the page that trigger downloads. We pay extra attention to user visible resources, so a single number may not be sufficient (but better than using MO all over the place).

Todd: Would element timing solve your use case?
Nic: That would require the application to do more of their own instrumentation, which they won't
Todd: So you're trying to hook that into existing frameworks?
Nic: Yeah
The second use case is to polyfill TimeToInteractive. We're not using network idleness as a signal because it's complex, but if we had the info we would use it.
Todd: And why can't you wait
Nic: We don't know when to stop waiting, because we don't know how many things are outstanding
Todd: Want to make sure that the use-cases are well defined
Yoav: So Phil's suggestion is an upper bound, but won't cover long standing polling requests which you may not care about
Charlie: So request start and end is enough?
Tim: If we add more stuff to RT, we'd expect them to show up dynamically
Nic: If the observer is too late to arrive, that may delay our decision to act on that info. But MutationObservers are not different.
Todd: Using the network as the measure, you want to notifications on start and stop. You don't want to poll it, so there's room to iterate on the API
Tim: There's inherent tension between buffering and async delivery here.
Rniwa: why buffering?
Tim: Imagine a really large resource. May be buffered really late
Yutaka: which events?
Tim: Events when the state of a resource request changes. There's a function call which says "give me information about resource requests" and events that fire as a result
Npm: Maybe a function that enables you to query the current state of the network.
Yoav: would we need to poll in that case? Maybe not as PO will fire if the network is not idle
Tim: Sounds concrete enough that we can draft yet another proposal

Matt: What happens if a request went to SW and SW responded with a Fetch event? Is that a new resource request
Npm: that would be in progress
Nic: we need failed network requests to PO as well, otherwise we will wait forever
Yutaka/Ben: no-cors CSS subresources would be invisible here as well, if they are invisible in RT and SW

Task Worklet

slides
Shubhie: been looking as using worklets to move script work off main thread.
Trying to tackle the use cases of moving heavy work: decompression, encoding, etc
Workers make sense for heavy work, but have a cost: startup, IPC, etc
Benefits have to justify the cost, and in some cases we found that it's not worth it: state management, rendering
So there's a clear perf win for heavy work cases.
postMessage is not ergonomic so proxying that work to a worker is nicer to work with, but then we hit surprise thread hops that introduce latency and memory overhead. And not APIs really work.
Scheduling based approach may do better, inspired by iOS Grand Central Dispatch.
Seems to work well for developers, and the system maintains control over thread management.
Encourages devs to structure their code in a way that works for background threads
Prototyped a worker task queue API, and looked at problems: thread hops add latency (mostly android, but also ios), structured cloning is also expensive. Android has async task API which is not great so trying to avoid posting results back to main thread.
Want to avoid posting tasks on same thread by default
Need to avoid making the main thread a single point of slowness.
Task Worklets
It's a simpler task queue API. Worklet has advantage vs. a worker: more restricted surface, has a registry pattern, etc.
<displays API>
The main thread registers a task, and lazily gets results (by awaiting on them)
When posting tasks to a queue, the results of one task are the input to the following task, so chaining the data rather than moving everything through the main thread.
The system can be smart enough to run these tasks in parallel.
Advantages of this are being able to pass the task as a pointer, almost like a representations of what the results would be. Assignment of tasks to thread can happen intelligently, avoiding the problems of moving everything through the main thread. Workers can coordinate data between themselves.
No explainer up yet, but WIP. Wanted to know what folks think.
Ben: Workers are heavy because of implementation decisions, but could be lighter. The spec doesn't require it. They don't have to map to a single thread.

Shubhie: we can incrementally improve workers
Todd: We used to have workers that didn't map to threads, but found interop issues and changed the implementation.
Similar to the C# task model. Syntactic sugar can be better. Is there a way to force block?
Shubhie: The "await .result()" is blocking
Todd: So while blocking on the result, the main thread could be one of the processing threads
Benoit: You example is really simple, could have done that as a subtask inside the worklet. What's the value of the mechanism?
Shubhie: The value is enabling the main thread to schedule that work. The main thread can decide what work happens and when, but the workers can coordinate to do that most efficiently.
Benoit: So that syntax makes it easier to chain?
Shubhie: Yeah, otherwise you can do that today but messaging between the worker and the main thread is awkward.
Todd: Composable worker heavy workloads is pretty cool.
Ben: Is this pollyfillable today? Can message passing implement this? Is there a core primitive missing?
Shubhie: we tried to implement a polyfill. Hard to implement the worklet-y parts but did that with a worker. Wasn't as efficient. Reduced the thread hops a bit, but in C++ we can avoid the main thread entirely
Ben: Channel and message ports can do the same between threads
Tim: You can't use your knowledge of available resources to decide how many threads
Shubhie: Dunno how good the JS implementation could be, but native impl would be better.
Ben: If Edge moved to strict 1:1 threading, that may not work, but maybe the browser can reuse threads
Todd: Maybe you're arguing for a native thread pool primitive.
Ben: A short worker is currently very slow in some implementations, but we could provide a thread pool and make it faster. The spec allows for it, and Firefox ship something like that. Doesn't save time to create globals, but can help. The syntactic sugar can help.
Shubhie: There are libraries that are solving similar things. No huge uptake, but a polyfill version of this may be better.
Todd: Can you define thread-hop?
Shubhie: Serialization, queing, hopping over to worker, and deserialization back. Some previous attempts created more hops. This one reduces them, by not coordinating through the main thread.
Markus: API is really neat. Declaring everything up front may not be enough, as logic of task running may vary based on the results. Also, does that support multiple tasks inputting to a single task or just linear chains.
Shubhie: Yeah, it does. We have a test that shows these cases
Benoit: Does the work getting posted immediately to a worklet or does it wait on the microtasks queue?
Shubhie: it gets posted immediately
Todd: Folks seem interested. Number of cases that need to be proven is significant. We can sync offline.

JS self profiling

proposal/slides
Andrew: sampling profiler for JS, that produces a trace.
Can be processed by client-side JS or sent to a server for aggregation.
Why do we want this?
In production, execution varies and to get a profile of user traces is painful. Having a native profiler would prevent the need to introduce instrumented code.
Serving the instrumentation only in certain cases break caches, and native profiler doesn't.
Use cases: complex web apps and analytics
Want to give UAs a lot of freedom in how they implement this
Want to create a structurally compressed format
Logical profiler - can spin up multiple per browsing context
Tim: If the sample interval is configurable, multiple profilers can end up sampling a lot!
Andrew: was thinking of common denominator
For trace format we used one that's similar to the chrome tracing format, which is trie
It's ordered by times
Privacy and security: dealing with third party frames needs to be careful. Should not expose more data than performance.now exposes, but fuzzing may be appropriate.
We want to see stack frames cross-origin
Implemented something similar on facebook.com, using a web worker. Instrumenting the JS and sends the info through a shared array buffer.
Hope a native implementation would be significantly faster.
Alex Russell: What do you mean by fuzzing?
Andrew: Sample times will have to be fuzzed
This will also include 3P JS frames running in the top level.
Chris: By 3P you mean a script on same document that came from a different origin
Andrew: Yeah, in the past we ran into security issues with those
So we propose to enable that for CORS enabled JS fetches, or with a Profiling-Allow-Origin header
Alex Russell: What's the motivation? Is there any information that's not owned by the origin? If you invited 3P script, it has no right to privacy? We want to protect the 3P source, but not necessarily its run times
Andrew: This will leak function names, which may contain sensitive data, if script evals
Tim: So if you evaled user/password, that would be bad
Andrew: In that case, what do we do with 3P scripts that didn't opt-in? Fuzz with "unknown" category.
Want the spec to be implementable, so want to give UA leeway on how to implement and instrument.
Need control over sampling rate, also concerns around providing new timers.
UA decides on supported sample rates and takes best-effort approach.

VMs should have the ability to inline or optimize away frames (JS stack frames).
start/stop should be async
Open questions:
If non-cors 3P calls 1P script, should we include them? It’s already exposed in the platform, not including them may make traces difficult.
Buffering: proposed sample count, but wondering if that is enough. Should we constrain on bytes/frames?
Tim: For UT, people can put arbitrary data, so same concern. Sample count seems to be enough.
Chris: How hard would it be to process on the client side?
Andrew: Main use case is server side. Client side use-case is for dev tools like thing.
Chris: If we keep it compressed on the client, it would save memory and can send it compressed
Alex: Would be better to have an optimized representation of the data
Chris: If no real use-case, that would be better, and extensions can do fancier things
Tim: Reporting API lets you specify a server to send it to. Could use something similar
Chris: Would be good for low-end devices
Andrew: Is there interest in other categories? Currently only defining JS, but maybe Layout is interesting
Tim: probably want to punt it. Also maybe you want to profile only long tasks to save on data
Chris: Would be interesting to have the function names there
Npm: How do you see the gaps when no JS ran
Benoit: You just know that you weren’t executing JS, but don’t know why.
Andrew: Browser could be working or could be idle
Benoit: Accounting for that would add more information, but would be more complex
Tim: Maybe add a “browser was doing something” category
Chris: That would be extremely valuable, as devs can take the traces and file bugs
Andrew: categories show up in frames, so one stack can have different categories
Tim: I don’t think we’d be exposing fined-grained categories in the next 5-10 years, so maybe not expose in registration
Andrew: and 3P JS as an “unknown” category
Andrew: considered interest in marker support. A bit redundant with User Timing.
Todd: Samples have timestamps based on the timeorigin, so can be sorted out later
Andrew: Last open question regarding format. Worth standardizing?
Tim: I think we should
Andrew: Questions?
Todd: Is there an entry point to differentiation set timeouts from event callbacks? If the implementation could, would that be useful?
Andrew: definitely worth considering
Markus: Just implemented that in the Gecko profiler
Tim: V8 wants to see examples of other customers, a more precise specification, and to verify that the performance won’t be too bad
Alex C: Similar question, how much overhead does the polyfill have?

Andrew: Got 5% degradation in Chrome, but was a little bit biased
Alex C: So with this enabled, everything is ~5% slower?
Andrew: May be closer to the devtools profiler
Tim: Not really, as devtools use a separate thread, which we can't do here. This will be slower
Alex C: Ideally a native implementation would be faster, but still slower than not profiling. This will hurt the performance of the profiled users, but will help improving it for everyone else. Would the users be able to opt-out? Can the UA have a minimum sampling rate?
Eric Faust: Talking about adding client code. Would that make other contexts slower? Because facebook has the right to make itself slower
Andrew: UAs can avoid spin up a profiler

Page prerendering

slides

Qingqian: Use case for baidu: 60% of users select the first result in the search page.
If we prerender that result, the UX to visit that page is seamless. It's a big win.
Time to render page is reduced by 60% if we prerender.

Less browsers support prerendering because it's misused frequently.

Limitations of prerendering: browser decides whether or not the page should be prerendered.

We experimented with prerendering using workarounds (iframe and MIP).

We need browser vendors to make some prerendering more tractable (?)

What we want: a policy for the page which will be prerendered. We want to reduce the unnecessary render by:

- Only prerendering the first screen DOM
- Only loading the first screen image
- Not executing the async script that just loaded it.

The question: should we use Feature Policy?

Yoav: have you looked at alternative ways to get the resources in? (prefetch navigation, preload subresources without prerendering)
What is the performance difference that you see?

Qingqian: It's a UX thing (?). It's a specification in the web performance working group. There is not a good use case for prerendering currently.

Yoav: Have you looked at upcoming standards that support that use case (portals)?

Qingqian: Can resolve part of the problem. Some pages may do many heavy tasks when being prerendered. We want to be able to run these.

Tim: You mentioned feature policy being useful here, are you suggesting we use feature policy here for things that are being prerendered?

Qingqian: I'm unsure whether or not we can add a feature-policy to be used in prerender.

Tim: We discussed a size policy, where you could only download so many bytes (most applicable). Consider no-script policy? Do you have ideas of other feature policies? They're likely to be useful for other contexts.

Qingqian: I need more suggestions.

Tim: no one size fit all solution here, want a different set of feature policies? I don't have enough context on feature policies in general to use it for this use case

Yoav: not sure as well. If you've looked at signed exchanges and portals, what's the gap from your perspective

Qingqian: the page in portals (?) is now AMP page, before google used portals to prerender pages. Now, google uses iframe to prerender AMP pages. AMP pages have problems with prerendering. In prerender, must be drawn in the web components runtime. If the page is not an AMP page, the page prerender may not run.

Tim: Understand use case, Google people not in room. Feel free to reach out.

Yoav: the piece that is missing is that there's no way to restrict the page to run outside of iframes, which is what AMP validation enables

Qingqian: maybe we can provide more details, more use cases, to explain what the limitations are and what we need for next time. Will write explainer on GitHub.

Priority hints

Yoav: Have made progress on the spec, chromium implementation, built for synthetic sites trying to emulate loading patterns. Haven't reached any conclusions with those experiments. H2 server prioritization is hard, the next step is to run similar experiments with (???) as server side transport stack so we exclude impact of H2 prioritization diluting benefits

Otherwise, work being done on more experiments. Still WIP, no conclusive evidence yet. Interested in exploring origin trials. Replace with native.

Page lifecycle

slides
Philip: API shipped in chrome 68. Some features not shipped yet, TBD. IndexedDB in progress. Missing client ID. Could use client ID to restore data for specific page. Work not started yet. Since shipping, freezing tabs on desktop/android. Discarding (only) on hold due to user complaints.

Shipped freezing loading in M67 (1-3% data saving), freezing non-timers in M68 (72% renderer CPU saving (battery) in background)

Work on bf-cache started as well. Bf-cache similar to frozen state, discussion regarding consolidation.

Perf wins: renderer memory reduction of -3.3% at 50th percentile, -3% at 75th percentile. CPU reduction of -16% main thread, -40% dedicated worker.

Nate: timeline on future work? For all lifecycle stuff.

Philip: API work and intervention work. We're trying to work out issues for desktop discarding. We're seeing obvious user benefits from this, make sure we do it right.

Yoav: instead of discarding serializing to disk, other options?

Tim: considering it.

Todd: edge does this, depends on OS infra.

Benoit: concerned about user backgrounding on the suspend case.

Tim: lots of heuristics detecting and excluding this. Heuristics not perfect, can incentivize doing horrible things. Need better ones, is hard

Philip: looking at APIs to make that possible

Element timing

slides/explainer

Npm: motivation: measure time that elements are rendered on the screen. We decided for reasons of complexity to restrict this to <img> tags for now. Plan to expand this to text and other elements in the future.
Benefits:

- Allow web devs to measure the first time that <img> elements show up to the screen
- Allows analytics providers to measure the first rendering times of key elements

What img elements do we care about?

- User-registered elements
- Elements that take a large percentage of the viewport (exact value tbd, starting at 15% of viewport). Automatically registered for observation. Enables analytics providers to provide some info

What about visibility/cross-origin?

- Entry creation cannot be strictly based on visibility
    - If the element is there but hidden by a pop-up, or the visibility is set to none
- Only entries corresponding to a CORS-same-origin response are exposed

What do we want to expose?

- The rendering timestamp of the image
    - Post-decode, available
- The intersection rect between image and viewport
    - Computed at the same time as rendering timestamp of image

Yoav: for progressive images, image is fully downloaded and fully decoded, yes?

Npm: yes. Will need to dig into html spec.

Gilles: based on flush, or when the person actually sees it?

Npm: first time that it renders post decode and load. I don't know how to fix this problem, hoping it's a rare case

Gilles: not rare when page is constructed during initial path, often browser will flush a whole lot of content at once. What if browser is blocked, has to wait?

Npm: something must be rendered for it to be computed.

Simplest use case is to measure the time that images are displayed on navigation.
Alternate use case: compute delta between user input that leads to <img> loading with the rendering timestamp.
The intersectionRect helps knowing how much of the image was visible during its first paint.
Useful beyond just pageload, even though it's not obvious

Re: IDL: name will be timing value, or ID, or "img" (generic image string). May not be enough to determine what the image is.

We propose also exposing the source of the image. entryType is “element”. Has 0 duration, startTime is rendering timestamp. intersectionRect between image and viewport at the time the first paint occurs.

Yoav: for resource timing the name is URL? We should make consistent. Maybe if there’s no element timing/ID, we add separate attributes?

Tim: gets messy for elements that are not images. Source is not populated in those cases. I guess you could have an image that’s an encoded blob.

Yoav: you still have a URL, don’t want to stuff into a performance entry. Let’s balance consistency with other performance APIs.

Npm: equivalent from resource timing would be name. Would we still want the attribute value?

Yoav: include other DOMStrings if present.

Tim: let’s bikeshed that offline

Qingqian: once the main image resource, not sure it will be rendered in the browser. I think it is timing of the image animate.

Tim: should we tie this to the resource timing attribute?

Qingqian: not sure it’s related. Can add other attribute to show the image resource has been fetched.

Tim: yep, related to consolidation discussion.

Yoav: nit: comment here says first paint.

Npm: yep it’s wrong

Yoav: it’s possible it’s painted before that, we should be careful there.

Npm: should we detect these cases?

Yoav: either the last paint before it’s available, or the first paint after?

Tim: let’s think about this some more.

Markus: seems reasonable enough

Nate: the way we embed images, we'll end up showing a sprited div background. Would be nice if future versions of the spec that could support this.

Npm: yes, let's expand this to other types of elements once we can figure out to measure.

Tim: seems easy to include background images that are > 15% of viewport

Yoav: for sprites we need to calculate the display size of the sprite particles?

Tim: intrinsic size of the image, or the display size. Whichever is smaller. Don't have a background size case yet.

Todd: for us, depends on implementation, adoption, having a web site say they could measure more accurately

Nate: if we the sprite thing, it would be super valuable.

Tim: origin trial the best way to get feedback?

Todd: we might get some things slightly wrong, it'd be great to get interest first

Benoit: gaming can introduce multimodality

Layout stability

slides
Tim: layout instability can be summed up as stuff jumping around when page is loading.
Stuff getting pushed around by ads, etc.
Irritating for users. They want this problem gone.
Let's surface via analytics provider to move the ecosystem.
We want to quantify instability in layout. Initial implementation ready, shows percentage of "layout unstable".
What causes layout instability?

- Ads
- Unsized images
- Web fonts
    - Async load
- Scrollbars
- Animations

How to compute?

- For each element whose origin changed, we take the union rect of old and new rect
- Report the fraction of viewport that is covered by this union.

Benoit: do you double-count based on DOM structure?

Tim: no, it’s a set union, can’t double count.

Markus: shifting by one pixel is the same as 200px regarding fixed cost? Yes.

Tim: element’s origin is its border box. Same with region, it’s the union of all of its box fragments.

Consider element that occupies 50% of viewport. It moves by a distance equal to have of its height. The layout instability metric is 75%. Caps out at 1 (100%).

It’s possible to get a score of zero on this metric.

To prevent animations from showing up as layout instability we check to see if the origin moved more than 3 css pixels. May want to parameterize this.

Benoit: can see how it’s practical.

Tim: have implementation in chrome. Similar to what’s described. When keeping track of union of boxes, we can’t do this in a way that’s perfect and adequately performant. We quantize this to a grid to reduce the complexity of the region. Lets us get good performance, pretty good quality. Worth allowing some amount of ambiguity?

Philip: what about elements added/removed to the page?

Tim: they’re fine, as long as they don’t move other stuff around

Expose this as performance entry with fractional jank. Maybe a little crazy to surface per frame, spammy. Want to do coalescing.

How long do you wait?

Cap on jank. If >4, page is already terrible. May be sane.

Yoav: as a browser, would report on register. What about analytics providers?

Tim: let’s put more thought into that. Graph displayed is on mobile.

Alex: CSS is loaded a second after rest of content. How to penalize?

Tim: we should catch this.

Markus: container box that is short, absolutely positioned element at the bottom edge. As content grows, absolute box moves. Should catch?

Tim: yep, we should.

Alex: how expensive is this to compute compared to the layout itself?

Tim: initially, terrible. Implemented quantization of regions, we can't see any latency increase inside the noise. Embeds nicely into our layout architecture, would be nice to know if it was performant in other architectures.

Is this reasonable? Is this the right place to pursue this

Alex: seems very useful for developer tools. We could be motivated by a success story of a web site that improved via this.

Tim: makes sense. We should do that.

Gilles: would be great to catch regressions with. Fundraising banners, etc

Benoit: would be very useful for FB. we care about this, useful measure to track

Tim: either we record it all the time, or we require header observation. Don't support header registration yet, part of that depends on the performance overhead.
Does anyone have thoughts on ambiguity in spec to make this performant?

Markus: as long as it's well-specified, won't be a problem. Good question whether to recommend a particular optimization.

Benoit: intuitively, cannot prove

Npm: in practice, might work

Benoit: maybe have an error bound? Not sure.

Philip: can we exclude animations for all? Css animations, transitions, etc?

Tim: can't capture one-frame animations that might game things. Do we actually care about this metric being gamed? If we try and make exceptions for animations, we might start missing legitimate jank

Philip: was thinking that most animations were on compositable properties. Not the case.

Tim: we'll come back with a crisper proposal at some point. Maybe better name. Change to layout instability from jank.

Platform supported A/B testing

slides
Tim: want to find out if this working group is the right place for the proposal.
Problem: RUM analytics A/B testing. Currently done using cookies, solved problem, tricky when you end up with multiple analytics packages on page. Having the browser provide support for analytics providers to coordinate provides lots of value.
Primary reason of interest is to provide privacy sensitive metrics. Chrome user experience report cares about these. Want to be able to get information about experiments without leaking data in traces. Having improved performance.memory and improved network usage metrics provides some value, but is not actionable. To make this actionable, we need to enable A/B testing integrated with the browser. Lets us aggregate effectively.

Want a privacy/security sensitive way of managing experiment groups. Proposal: have a browser send an experiment seed, page responds with a group count and a sticky flag. Let's stop and see how appropriate this is for this working group.

Yoav: with crux, this will support multiple equally sized groups. We cannot run experiments on a small number of users.

Tim: RUM analytics will be able to use same experiment ID.

Todd: If there are websites that want to use a standardized mechanism for experiments, it fits in the web perf wg because it involves gathering and reporting data

Tim: a good use case, but don't think the WG should prioritize for that reason

Yoav: explore the other use case, see if it's more generic and can have a better fit

Todd: ask to web sites if they want a standard way

Tim: in order to preserve privacy in the aggregate case, we need to make the API clumsier.

Todd: would other vendors say no due to this?

Tim: high interest in memory data from gsuite

Nate: would be excited about this. Would be nice if other sites couldn't figure out which experiments are running.

Yoav: block data exposed, not experiment?

Nate: yep

Tim: interest in other browser vendors similar to crux?

Markus: not what we do at FF

Alex: seems chrome-specific, shouldn't be standardized right now

Yoav: value for non-crux usage is the initial seed that A/B testers don't have. Not everyone has access to set-cookie

Tim: in this proposal, the browser comes up with the seed. Scope of seed is the same as scope of the cookie. What's the win over cookies?

Yoav: not everyone wants to set cookies on a site

Tim: could avoid namespace collision. Maybe an argument for browsers to support analytics collaboration.

Alex: right now, not too compelling

Benoit: don't you want independent experiment groups?

Tim: analytics wants to split up based on experiment groups.
Again, is this not a good venue for this discussion?

Markus: not sure, anything that browsers ask is going to be part of the platform, in which case needs to be standardized. May not need to be now

Yoav: if this will change in the future, won't break content in the future. Existing content could be transitioned.

What if we don't send a seed?

Tim: how to avoid this happening? Let's have this discussion in wicg

Alex: is seed new tracking vector? Not yet interested in implementing it.