WebPerf WG @ TPAC 2019

bit.ly/webperf-tpac19

Logistics

Location & participation

Hilton Fukuoka, Sakura, 3F
Remote participation/presentation link

Attendees (On location)

Attendance for the Web Performance WG meeting

- Yoav Weiss - Google (all days)
- Ilya Grigorik - Google (all days)
- Todd Reifsteck - Microsoft
- Nic Jansma - Akamai (all days)
- Nicolás Peña - Google
- Ryosuke Niwa - Apple (Monday only)
- Alex Christensen - Apple
- Steven Bougon (All days)
- Philip Walton - Google
- Will Hawkins - Mozilla (all days)

- Nathan Schloss - Facebook (Tuesday only)
- Andrew Comminos - Facebook
- Vivek Sekhar - Google
- Ian Clelland - Google
- Tim Dresser - Google (all days)
- Xiaoqian Wu - W3C
- Markus Stange - Mozilla (all days, but partially (CSS))
- Matt Falkenhagen - Google (late Tuesday)

Remote

- Thomas Kelly - Shopify
- Jason Chase - Google (Monday morning)
- Gilles Dubuc - Wikimedia Foundation

Agenda

Monday (Design)

Timeslot	Subject	POC
08:30~09:15	**Intros & agenda review, meeting goals** + Overview of 2019 & roadmap for 2020	Yoav, Ilya, Todd
09:15-09:45	**EventTiming measureUntil (async duration)**	Nicolás
09:45-10:15	**isInputPending** + Nate: spec+implementation update; group status + Nic: adoption experience & lessons learned	Nate, Nicolás, Andrew
10:15~10:45	**Reporting API & Network Error Logging** Review feedback & proposed plan	Ian, Jason
10:45~11:00	Break	
11:00~12:30	**Next steps on SPA monitoring** + Review discussion from F2F & Tim's proposal + Map out 2020 work streams	Tim, Ilya
12:30~13:30	Lunch	

13:30-14:30	**CompressStreams** (explainer) + Review and discuss current proposal	Adam, Andrew
14:30~15:00	**JS-self profiling** + Proposal update & implementation status + Long Tasks and attribution?	Andrew, Tim
15:00~15:30	Break	
15:30~16:30	**Frame Timing** - Noam, Amiya (Eric/Todd may need to represent?), Nicolás - Noam will dial in from Israel (schedule late)	Nicolás, Noam
16:30-18:00	LCP implementation experience	

Tuesday (Issue & Hack-day)

Timeslot	Subject	POC
09:00~10:45	**Design discussion continued** *Reserving for followup from Day 1, plus other potential topics...* + Living Documents + ResourceTiming Visibility Improvements + Cross-Origin opt-in and Bubbles + isFramePending (szager)	
10:45~11:00	Break	
11:00~12:30	**W3C spec status review & next steps** + Review publication status of each spec + Identify next steps / unblock / move forward	Yoav, Ilya, Todd
12:30~14:00	Lunch	
14:00~15:45	**Working session: issue triage & resolution**	
15:45~16:00		
16:00~17:00		
17:00-18:00		

Spec spreadsheet

- Navigation Timing may have 2 issues that require some discussion,
- Otherwise for RT, NT, Page Visibility, rIC is Just Work™
- Preload, Resource Hints, Long Tasks, Device Memory, Reporting and NEL need a lot of love.

Potential issues to discuss

- Resource Timing
  - PR#214 - Align TAO with CORS
- Navigation Timing
  - #114 - Spec for PerformanceNavigationTiming.type does not match any implementations - hopefully only definition change + tests, so no need for discussion
  - #115 - Definition of "back_forward" navigation type does not make sense - discuss desired behavior
- Preload
  - Specifying Range in Link preload header for HTTP/2 Push
  - 103 and CSP
  - Urgency Hints
- Resource Hints
  - Prefetch and double-key caching
- Long tasks
  - "multiple-contexts" doesn't seem useful
- Reporting
  - https://github.com/w3c/reporting/issues/74 Observing reports cross-frame
    - No clear use-case
  - https://github.com/w3c/reporting/issues/126 Observe() should be promise-based
    - No
  - https://github.com/w3c/reporting/issues/131 Expose reporting api to workers
    - We should probably do that for reports that are triggered from workers
  - https://github.com/w3c/reporting/issues/132 Format of dates
    - We don’t have any date format we currently use
  - https://github.com/w3c/reporting/issues/147 Relative URLs in report config
    - No if we can convince current report-uri users to move
  - https://github.com/w3c/reporting/issues/161 Credential-less reporting
    - Same-origin can conflict with batching
- NEL
  - 14 open issues

Videos

Sessions

measureUntil

TL;DR

- Nicolás presented proposal for async measurement of event handlers.
- Folks mentioned that it seems like something that can be useful for User Timing as well
- Concerns were raised around exposing paint timestamps beyond rAF and the security implications of that (in the context of Event Timing and in general)
- Proposal to allow User Timing to declare connection to an event and have that timing entry also appear in Event Timing

Minutes

Nicolás: have some ideas to improve EventTiming
… in ET we expose hardware timestamp, before and after running handlers, and next paint
… duration is from hardware timestamp to next paint
… the objective of this exploration is to figure out how to measure asynchronous work
… e.g., frameworks can handle events asynchronously → the browser can't trace this
… e.g., got feedback from React that they can't use ET because duration is not accurate
… anytime you have async tasks (e.g. network fetch), you'll run into this issue
… Use cases
- Click -> fetch image -> add to DOM
- Drag video slider -> buffer video -> start playing again
- Enter query -> query server -> display result

… API shape

Proposed API shape

```
partial interface Event {
  void measureUntil(Promise<any> workPromise,
                    DOMString promiseName);
};

[Exposed=Window] interface EventDurations {
  readonly maplike<DOMString, DOMHighResTimeStamp>;
};

partial interface PerformanceEventTiming {
  readonly EventDurations asyncDurations;
};
```

… once the event is dispatched we wait for all promises to be resolved
… once promises resolve, the timestamp allows you to determine duration
… we could use the longest timestamp
… asyncDurations would keep a map of promise names to durations. For calculating if an event was long and we should expose it, we can take the longest duration.
Durations will be till the next paint after the promise was resolved.
Tim: I think we can't use next paint for security reasons, only the next paint from the longest promise.
Nicolás: certainly a security concern that needs to be considered. Not sure just the last one would be useful tho.
Yoav: what are the security concerns? Visited links?
Tim: yeah, all the usual suspects. If you can measure how long an arbitrary frame takes to paint, you can create visited attacks. We said it's OK for event timing because we only expose events longer than 50 ms. But not OK for arbitrary promises.
Nicolás: need to investigate more, maybe throttling, etc.
… the reason we have a map is we can have multiple 3rd parties instrumenting events and we don't want them to step on one another.
Ilya: So this assumes the vendor instruments the event?
Tim: We think frameworks would instrument their events and analytics relying on that.
… a single party might want to look at multiple events it created
Ilya: impact on current ET?
Nicolás: it would affect duration and when it's dispatched, but only for those that are instrumented
Tim: Not clear how much of that work is using promises today and how easy it would be for folks to switch.

Olli: is this event time or is this about user timing?
Nicolás: the issue here is knowing the next paint can be a security concern, so want to expose that only in certain scenarios
Yoav: would this work for JS generated events?
Nicolás: no
Ryosuke: Can you keep adding promises to the event and continuously measure using that?
Tim: That's why we suggested only to measure the last promise resolved. Need to think about it a bit more.
Ryosuke: are we exposing different information from what rAF would expose?
Tim: yes, and we determined it was OK due to 50ms threshold we added
Ryosuke: you could harness mouse moves to get every paint
Tim: Good point. Worst case, we'll have to expose a less accurate timestamp
… coming back to User Timing question
… I want to instrument both browser generated and JS generated events
Olli: Might be a better generic API
Tim: Won't solve those 2 use-cases
Ryosuke: if there was a way to associate an event in UT..
Nicolás: You can already add metadata to UT
Tim: So I'd now need to listen to both event timing and user timing and combine those?
Ryosuke: we can add sugar to make this easier
Tim: We could, but not clear what's the advantage. But maybe performance.mark and measure can use a promise based API
Yoav: Right, in theory we could even include ET paint~like signals.
Tim: Maybe. But want a single place for analytics vendors to look when measuring event responsiveness
Ian?: There are multiple people involved: frameworks, developers, analytics providers.
Yoav: Sure, but how can analytics providers know the semantics of each promise?
Tim: I envision e.g. React to have an identifiable event names
Ryosuke: sure but naming is independent from the UT vs. ET question
Nicolás: Use case for non-event driven paints?
Yoav: Image gallery that changes image every X seconds
Tim: So the proposal: add something to UT L3 to say "here's my associated event", and if the event is present, we'd also surface this in the EventTiming API. But if you want to do ssimilar things without an event, you can.
Ilya: Also relevant to the SPA discussion
Nic: question on durations: you would still have current timestamps on top of the promise resolution timestamps?
Nicolás: yes
Nic: But it could delay the delivery of Event Timing.
Nicolás: true
Philip: Can you hold onto an event and add measureUntil later?
Tim: That won't work because the entry may have already surfaced
Nicolás: Argument for the UT approach

Yoav: Also a question of precision - can we only expose a rAF timestamp or something more accurate.
Tim: We see cases where compositor effects result in significant delay
Ryosuke: Some OSes cannot really measure that
Tim: The spec states a best-effort approach
Rick: Any data on the difference? Intuition is that compositor costs would be dwarfed by the rest
Tim: No hard data, but seen examples where it was significant. But might be less of an issue in other architectures
Yoav: Ties into Frame Timing as well
Tim: So we need a broader conversation about paint timestamps
Ryosuke: believe rAF can solve 80-90% of cases and everyone can implement
Ilya: +1
Tim: Scrolling is often compositing-bound, and want to solve scroll-jank. But we can maybe use a mix where if you don't use measureUntil you get the high quality timestamp and measureUntil falls back to rAF. In the async case, compositing is likely to be a smaller fraction.
Todd: we've seen some top properties that have significant compositor time costs. So need data on impact before discounting it.
Yoav: can we kill visited?
Tim: Even if we kill visited, cross-origin same-process iframes is still a thing

IsInputPending

TL;DR

- Great results from Chrome Origin Trial, both from Facebook and Akamai
- Current spec language is too vague for following implementations. Would need to be defined more precisely
- isXPending API shape may require some unification
- Iframe exposure requires some more thought around its security implications

Minutes

Nate: update! Origin Trial on desktop but not mobile.
Andrew: not yet available on mobile due to lack of site isolation
Nate: The new facebook.com has full scheduling but not enough traffic. Most of the trial used a naive implementation. Results looked good with 100ms reduction
Nic: similar results for us. Used that in boomerang and used it to avoid impacting user experience when doing work. Instrumented on some sites but not all, due to Origin Trial limitations. Will help them break up their tasks in a meaningful way. Want to use it everywhere.
Tim: Are you using it for attribution that event timing can give you?
Nic: Not only that, but also to avoid heavy processing
Navid: why those events and not all events?

Nate: intentionally left this vague in the spec. Browser may lie in some cases, or may return best-effort results.
Tim: Current implementation assumes that no events will get prevented.
Phil: How does that interact with FID. can we ignore long tasks if isInputPending was queried?
Tim: We could do that ideally, but need to dig into usage in the wild. You're still blocking rendering, but also you may not yield, so that may not work
Olli: the isInputPending spec is too vague. Would be impossible to implement based on it. Would need to look at source code :/
Todd: What's vague?
Olli: Click handling is a vague term. Also "may include any UI event"
Andrew: Fundamental problem specifying it, as we can't know what's in the JS event queue. precise frame tracking makes it tricky. Has to be best-effort. Would making it closer to hardware key events be better?
Olli: Closer to mouse down/up would be better
Andrew: Seems reasonable
Todd: Mapping directly to HTML definitions of the events would work?
Olli: It's not really defined anywhere
Nate: that's the problem we ran into. Can expand to mouse up/down.
Olli: Need to get rid of "MAY", as it's not clear what it means.
Tim: Small set of supported event types that would give us most of the utility?
Andrew: Something like mouse move would be incredibly noisy. Can add counters to see what people are listening for.
Navid: Why "may include UI events"?
Andrew: To avoid mouse move
Navid: but what other events may be included?
Nate: the idea was to make this future compatible when new events get introduced.
Nicolás: adding more events can make it easier to misuse. Need to be careful with which events we support
Tim: Worried with events that fire too often? One potential solution is to use a * event type to query for everything

Ryosuke: What other isXPending API we'd have? Could become awkward fast
Alex: What happened to shouldYield?
Nate: It became this. Different sites care about different things.

Navid: is this exposed to subframes?
Andrew: want this to rely on hit testing and expose to subframes as well. Can also apply to subframes that share event loops. Decided to go with per-document
Todd: so there could be input pending that hasn't yet reached the hit testing, and only after there is will be reported. Same event-loop frames would need access, but there's some security implications.
Andrew: Maybe we can filter out cross origin events, to make that safe.
Navid: what happens when hit testing fails to know the target?

Andrew: Still TBD.
Tim: Worst case, we want to let the browser say “I can’t reliably tell which frame this input is associated with, so I won’t tell anyone about it”
Nicolás: could encourage more blocking scripts
Tim: We’d also want to measure how often that happens
Andrew: Main things to be concerned about are focus and frame movement. Can hamstring the API when frames get moved. Focus is a bit more common.

Reporting

TL;DR

- Agreed to split ephemeral reporting from persistent reporting, where persistent will be incubated as a separate document.
- Batching and sampling can remain in ephemeral reporting
- Move to structured headers would limit per endpoint parameters - need to open issues on SH
- Apple needs to further review the security aspects of ephemeral reporting

Minutes

Ian: Recently took over editing. Excited to see Mozilla involved
Two distinct use cases: Tentatively calling them ephemeral and persistent.
Ephemeral reporting is for all reports which can be tied to events occurring within a document: CSP, Feature Policy, Crash Reports, Deprecations, Interventions.
For ephemeral Report-To should just work
Persistent reporting is another use-case. Main case is NEL, but also other out-of-band reports. For these, we want to support persistent reporting configuration, and report completely out-of-band, for cases where perhaps there isn't an active document.
Mozilla would like to support ephemeral, so we want to move persistent out so that it can be reasoned about separately, but still leave building blocks so that we can build NEL on top of it..
Don't need for ephemeral: Persistent configuration, delivery after the document has been closed, batching reports, possibly failover and retry (though those may be useful), sampling (Could also be useful, but may not technically be needed)
Ilya: We did hear about the sampling use-case for CSP. So we do need it for ephemeral. It does feel like there are some low-level primitives that can be useful there.
Anne: How does sampling work?
Ian: It generates a random number for each report based on the sampling rate
Ian: How can we remove the persistent requirements? Make them optional? Make them a separate spec? Or make a Level 1 / Level 2 spec?
Ryosuke: What is the goal? What are you trying to do?

Ian: Enable shipping ephemeral reporting without things needed only for persistent reporting, without forcing implementers to build support for features they are not intending to ship.
Right now implementing the reporting spec means you need to implement persistency features that are not needed for per-document reporting.
Ilya: when we incubated reporting, NEL was the main driver for many of the features. The two were developed side by side. Only afterwards we added intervention reports, which have a much simpler subset of requirements.
Alex: This really feels like Level 1 / Level 2 to me.
Anne: You could have the core thing, that I think most browsers agree on, and then the persistent thing separate. There is also some sorting out to be done there because some of it is just not acceptable.
Yoav: So, separate document, or…?
Anne: Separate document, if we don't agree on the design.
Ryosuke: I think it needs to be a separate document, otherwise you could very easily introduce interdependencies within the same document.
Yoav: We got feedback from Mozilla on Navigation Timing and Resource Timing, and the dependencies between them, that Mozilla would like to see them united
Anne: If at some point we can agree on the design, then we can merge them. Incubate first as a separate document.
Alex: Concerned about the testability. If it's only usable by these other things that build on top of it, could we have a collection of unit tests for reporting, or do we have to go through the other APIs?
Ian: WPT has a webdriver API that can drive those reports and makes them testable
Ilya: Also reporting has a few features that can may be testable
Ian: Yes, but those features are Crash, Deprecation, and Intervention, and those are not
Ryosuke: Sounds like persistency is L2
Ilya: let’s talk about the features that should go into ephemeral
Ian: Proposal - move report config syntax to Origin Policy. Restrict batching to per document or per agent cluster. Remove max_age
For the persistent functionality, we decided that we want to move it to a separate document.
Anne: For batching and sampling, they seem fine to remain in ephemeral
Ian: For opt-in, Brave wanted user opt-in for 3rd party reporting.
Ilya: Related, how does report API interacts with extensions/fetch?
Ian: Reports skip SW. Extensions are UA defined.
Anne: also per-document reporting gets complicated with batching. SW should definitely skipped in those cases
Ilya: Need to note that in the privacy section.
Ryosuke: users won’t understand the implications of error reporting to different origins. Prompting that won’t be a good security mitigation
Ian: similar to background sync
Ryosuke: users are more likely to understand that
Anne: 3rd parties are allowed to fetch
Yoav: there’s a difference between in-document third party reporting and persistent one.

Anne: Oh, yeah. In the persistent that's indeed different.

Ian: Let's talk about structured headers - currently we have JSON. hard to fit into structured headers
Simple dict - lose parameters on different URLs
Dictionaries with inner list - proposal to add parameters to inner lists. We lose the ability to have parameters that apply to the end point
Ilya: sampling is one example. Regarding priority, need to talk to NEL
Ian: Another option - define endpoints and groups in different headers, but it's not great

Yoav: Have Apple looked into ephemeral reporting? Interested in that?
Ryosuke: still concerned about cross-origin reporting even for per-document reporting without credentials, as well as tracking potential using the persistent API.
Anne: cross-origin reporting is equivalent to fetching the same data
Ryosuke: could be OK. Need to do a strict sec review

Todd: So the idea is that there's a mechanism and a few features that use it (e.g. CSP). Then there are crashes, deprecations and interventions that are defined directly in the API.
Anne: Where do crash reports stand?
Yoav: In the spec. Would be nicer to split features from infrastructure.
Anne: Crashes that impact the entire agent cluster pose interesting questions as to where those reports should be sent.
Todd: spec defines oom, unresponsive and else for crash reasons
Vlad: important to Facebook to distinguish those reporting concerns
Ilya: can consider pulling some of the features to a stand alone spec
Anne: might be OK to leave crashes in
Todd: features beyond CSP?
Anne: Interested in crashes, COOP, feature policy
<discussion on feature policy and opinionated reporting>

Single Page Apps Monitoring

TL;DR

- Tim Dresser presented a proposal to measure soft navigations.
- Agreement in the room on
    - the shape of the API proposed, and that it should re-dispatch various "First-*" entries (as security permits) as a carrot for developers to use this
    - avoiding side-effects to such measurement (e.g. killing in-flight requests)
    - aiming to solve navigations that cover most of the page, rather than smaller transitions. Specifically, simultaneous transitions are out-of-scope.
        - At the same time, there was interest in LCP data for smaller transitions

        - In the case of navigations, “end” timestamp probably doesn’t make sense and/or may not be enough

Minutes

Tim: being clear on use cases we’re solving for is critical. Not all of them are navigations
… UC1: click → full page transition, maybe url change
… this is the use case that we probably need to prioritize
… UC2: click → minor transition
… UC3: a video site transitions to the next video: no input, URL change
… UC4: Ad rotation, makes things tricky because you now may have multiple transitions
… Goals:
- Redispatch load entries
- Read entries relative to transition start time.
- Can use input event timestamp as start time but input isn’t required.
- Doesn’t require history.pushState, but can optionally associate URL change
- No explicit integration with analytics providers required.
- Transitions can take place simultaneously

Ben: is simultaneous transitions a critical req?
Tim: good question, that’s what we want to answer here. If we punt, life is easier.
Ben: let’s punt it
Yoav: ad use case is probably addressed via iframes, they have their own timeline
Ryosuke: the use case is ~main part of application is changing
Tim: yes
Ben: What do you mean by “explicit integration with analytics”
Tim: Analytics providers don’t need to be aware of how your app is built and annotated
Ilya: redispatching entries is one of the main carrots here, but we need to be careful as not everything can be redispatched, for security reasons
Tim: few potential solutions: low quality timestamps, throttle the API, etc. Need to give it some thought
Yoav: At the F2F we mentioned killing network requests in soft navigation scenarios. Could be a carrot.
Ryosuke: Feels like a giant footgun
Tim: Redispatching entries should be sufficient

Tim:

Proposal

```
// Similar to performance.mark() and performance.measure().
// On |window| not |performance| due to pushState.
window.transitionStart('name', {
  startTime: timestamp,
  pushState: url, // Optional, but does pushState if present.
  detail: {}
});

window.transitionEnd('name', {end:timestamp, detail: {}});
```

When a transition occurs, we add an entry with some data, including the pushState URL if there is one. Feels annoying to have to call both pushState and add data about it here.
Ilya: Yeah, but this API is about measurement, and we don't want it to have side effects. Also, different frameworks have different patterns to when the URL is changed.
Tim: So should the URL be in details?
Ben: Yes.
Ryosuke: Also, the browser knows what was the URL when this was called
Yoav: Yeah, but the URL may be changed after transitionStart.
Yoav: Also, I wouldn't tie to pushState as the history API may be revamped in ways that can help that API's goals.
Tim: And if it does, we can still tie in to that later. Also, if we're dropping the transition use case and just focus on navigations, we can drop the "end time" part.
Tim: one of the key things you need to do is being able to query transitions
… effectively, this is a signal about your time origin. Seems better than changing the time of the different entries. All performance entries will be surfaces with the page's time origin.
Yoav: OK. Regarding "name", even when multiple simultaneous transitions are not needed, "name" can help people understand which is which.
Ben: yeah. A list of names rather than a stack.
Ilya: Would people need to full list or just the last one?
Tim: Probably need the full list to reason about the different performance entries.
Ilya: Global time origin enables us to coordinate time origins between window and workers. Maybe we can create a "transition time origin" that can help coordinate those different time origins in a similar way
Nic: is there an observer to be notified of new transitions?
Tim: Good point, there should be.

Todd: Will we have transition mark start/end, etc?
Tim: If we focus on navigation, there's no "end". But you could imagine UT L3 adding an "is transition" flag rather than a new API call.
Todd: So developers can add their own "end" UT calls
Yoav: one use case of having an end is to allow developers to signal own ~end, and analytics can collect it
Tim: My intuition is that one timestamp is not enough
Ryosuke: this feels related to the broader discussion of "end" signal, we can separate this
Steven: So "transitionStart" will reset all the navigation related entries?
Tim: It would cause them to be re-dispatched
Ian: what happens if you have transition when one is active
Ryosuke: you probably only dispatch the last one, we wouldn't claim that in flight one is finished
Todd: How do we handle async transition start where user input may be happening while we're handling it?
Tim: Feels like this can also on full page loads.

Ben: what about redirects?
Yoav: redirects are handled further down the stack.
Ilya: There could be cases where you have multiple transitions: one kicking off a spinner and then another that loads the final page.

Will: regarding naming, would be nice to avoid transition- prefixes on all the metrics. Maybe have multiple entries.
Tim: There may be compatibility problems with current handlers or with current data collectors in case developers adopted the API before their analytics providers
Ilya: We can probably talk to analytics folks to prevent that.

Philip: want to consider use case of URL changes, hanging on pushState
… that would enable all of the existing apps to have this auto-instrumented
Tim: so it the question "do we care about transitions without URL change?"
Philip: I think UA can take care of that.
Yoav: We definitely want an explicit signal for people who care. And then we can ask: can browsers heuristically point transition start without it? Because different frameworks kick off transitions at different times compared to pushState, this can be hard.
Philip: Maybe a flag on the performance entry can indicate what heuristic was considered
Ilya: I can imagine analytics providers using the same heuristics today
Todd: If you have to opt-in to PerfObserver to get the entries, you'd need some change in code to collect it.
Yoav: You want analytics providers to collect it without application developers changing code
Philip: Also CrUX
Tim: But those numbers won't be comparable.
Eric: you would be able to compare yourself to yourself.
Yoav: Maybe we can use frameworks to make sure this happens in reasonable timeframes.

Ryosuke: When the user clicks, you may not know where they'd end up.
Yoav: Sure but you can still mark transition start.
Steven: You can use that for any UI changing event, no?
Tim: It doesn't make sense to use it too often. There are always security issues.
Ryosuke: I don't think we should care about partial updates.
Tim: Event Timing should handle those cases
Steven: And the developer will indicate done at the end of UI transitions?
Yoav: measureUntil
Steven: But in our case, we have no access to Event Timing, as we instrument other people's code
Tim: Current proposal doesn't allow getting paints without reseting the time origin. Would that help?
Steven: It would.
Todd: And for long tasks, you'd be able to correlate them with transitions
Steven: Yeah, and naming transitions would help there with long task attribution to user gestures.
Yoav: I wonder if "transition" is the right terminology
Tim: Yeah, it may be over general.

Tim: Shall we talk about cancelling in-flight requests?
Ilya: Seems related to the problem of observing what's in-flight
Yoav: Maybe when we would observe in-flight fetches, we could make those fetches cancellable
Will: Agree that we should avoid side effects for measurements
Ryosuke: There are many cases where you don't want to cancel in-flight requests during transitions, so need to give granular controls

Will: Do we want to consider simultaneous transitions in the future? Or are we painting ourselves into a corner by not considering them?
Yoav: Tim explicitly asked that question, and no one seems eager to tackle them as part of the same use-case.
Tim: incremental step forward can be analytics vendors looking at the latest transition.
Ryosuke: Seems like most of them can be done using measureUntil
Yoav: iframes can also tackle the ad transition use-case
Steven: The nice part about this is getting the Largest-Contentful-Paint info. Can we get that with measureUntil?
Yoav: That's just the first paint
Steven: I can definitely see scenarios where LCP would be useful here.
Tim: as we think about connection event timing and user timing, we can come up with patterns that would enable you to do that. More thought required.
Ryosuke: you want to support non-navigation changes, right? We decided against supporting it here. Maybe we can have options on measureUntil to provide LCP after it.
Philip: Do we really want to reset the time origin?
Tim: Not really resetting but providing an alternative one.

Tim: Got a lot of great feedback, thanks!

CompressStream

TL;DR

- Use cases seem interesting, would be good to gather all possible ones and then decide which ones are worth supporting
    - Some related to capabilities (e.g. ZIP, EPUB, SSH)
    - Others related to compression ratios (e.g. Brotli compression, zopfli)
    - Support has maintenance and binary size costs
- No need to specify binary output of the algorithms. Options (e.g. compression level) can provide hints. Potential foot-gun options should be gated behind scary option names.
- Apple wants clear criteria regarding when support for new algorithms should be added
- ZIP bombs can be an issue. Developers should be careful when decompressing unvalidated input.

Minutes

Adam: use cases: compressed upload, particularly analytics use cases
… Lazy decompression for downloads, compression/decompression for native files, etc.
… Applications try to do this today
Andrew: @FB we tried to compile gzip to WASM and we saw 20% hit compared to native
Adam: status, FB has prototype of CompressionStream, Google has DecompressionStream
… aiming to get experimental support in Chrome this month
Yoav: why limit gzip and deflate? What about brotli, z-std, etc?
Adam: we could add more in the future, the question is of cost of including library in binary
… e.g., Brotli compression would require including ~140KB in the binary
… because we only include decompression
… we could entertain only decompression, but is that useful?
Yoav: seems like the value of brotli 11 for upload compression is questionable
Eric: which “deflate” variant are you talking about?
Adam: Need to clarify that
Eric: particular use case I’m interested in zip reader
Yoav: Other specific use-cases: browser based SSH client would require options
Domenic: second that, epubs, different publishing formats, etc
Adam: had folks saying we should support “deflate” rather than “gzip”. Need to come up with a new name for “deflate” if it’s ambiguous
Eric: how liberal will you be? E.g. gzip checksums, etc.
Adam: interesting, will have to think about it

Yoav: Worthwhile to flesh out the use cases: for gzip you might want to specify option for flush strategy
… for example, SSH clients require particular strategy and Java didn't support it
… which made implementing a SSH client impossible in Java
Adam: what about levels? 1-9, fast-medium-high, …? Don't want to force specific byte output in the standard. Would prevent implementation improvements over time
Yoav: what are the problems with not specifying exact byte output? We can treat the levels as a hint to the compressor as long as the output is zlib compatible
Adam: Also, people suggested Zopfli should be level 11
Yoav: Similarly to the brotli 11 use case - farfetched for in-browser compression, and the scale is not linear
Ryosuke: Different browsers will encode differently, but that's not a problem
Will: Will that compression definition matter for compatibility?
Yoav: Decompressors shouldn't care. Upload size may be different between implementations
Domenic: second Ryosuke's point on not over-specifying the byte format. Canvas has a similar 0-1 quality parameters and implementations can interpret that as they want. If the goal is heuristic control over the CPU/byte-size tradeoff something like that is enough.
Ryosuke: lossy compression may be different, but OK for lossless
Yoav: as long as the decompressor can deal with it, that's fine
Eric: If we wanted to add brotli/zopli 11, would be unfortunate if web applications became unusable
Yoav: maybe gate that use-case (if necessary) behind a different option, not just one more level
Eric: window sizes can also be abused
Adam: It's fine, as you can define the same on the decompressor side
Yoav: isn't gzip limited to 32K windows?
Eric: there are different scenarios - ZIP can use larger windows. So you want to keep that option available, just make sure it's not a footgun. How much can we guide developers towards safe options.
Ryosuke: if the decompressor can't deal with the output, that'd be an issue.
Eric: Depending on what the decompressor is. Should the API (as a compressor) be able to produce deflate64? To what extent should the API guide developers towards "standard" algorithms. So that without options, the output is guaranteed to be browser decodable. And with specific options, I can go beyond that.
Yoav: Yeah, but browsers using this API for decompression could decode any output, e.g. using compression dictionaries. So if you control both ends, you can do weird things.
… Otherwise, potential footguns should have a scarier name than just a different level.
… EPUB, ZIP, etc are capabilities that would be good to support. Extra 20% of compression at the price of x1000 CPU time doesn't seem like a critical use case to me for one time uploads

Adam: future algorithms

- LZ4 - very high throughput, low compression ratio
- Snappy - similar to LZ4, already compiled into Chrome and Firefox
- Zstd - medium throughput, medium compression ratio

- Brotli - low compression throughput, high compression ratio

Domenic: Can we say that we'll never remove brotli from the platform?
Yoav: yeah, brotli is web-exposed not going anywhere
Eric: yeah, that's different from Snappy, that's used in Chromium internally, but can be replaced in the future
Todd: so brotli and zlib are both exposed and reusable. There's the question of when should we add another compression algorithm. When does it add sufficient benefits to outweigh the costs.
Adam: we can do origin trials of different algorithms to measure benefits/costs before shipping.
Eric: can you detect which algorithms supported, would it be possible to polyfill?
Adam: yes, we can expose algorithms and app can add own to the list
Alex: Would be good to have clear criteria as to when new algorithms will be added.
Adam: binary size would be a limiting factor to adding new algorithms. Dominic Cooney suggested randomizing the results of feature detection to prevent UA sniffing
Yoav: may cause an opposite impact. Also, not a problem specific to this API, but a generic one

Yoav: worker availability?
Adam: yep, planning on it
Ryosuke: in the explainer, would be good to have some concrete use cases + why some algorithms were added

Eric: How would decompression stream be consumed? Would zip bomb attacks be an issue here?
Adam: were looking at making additions to Streams to better handle it. Best current approach is to split into 1K chunks. Stream should provide a feedback that allow the transformer to know when stuff is read.
Yoav: 1K chunk can decompress to a very large one. Can the API refuse to decompress zip bombs?
Adam: May not be necessary. There may be use cases for intentional zip bombs. We could implement it, but seems extreme. If you're reading in a streaming fashion with small chunks, you'd be fine. If you throw it into the Response object, you'd run out of memory.
Eric: OOM exception seems fine as long as it's not a slow and weird failure
Adam: you'd get a clean OOM crash, killing the renderer
Yoav: and you shouldn't decompress random stuff. So you probably want to do some validation of that. You could similarly create such a zip bomb against the current network stack decompressors.
Eric: Also, server side validation can result in DoS. One of the reasons upload compression is not universally supported.

JS-Self profiling

TL;DR

- Currently in Origin Trial
- Discussion about opt-in and security-related COEP limitations
- LongTask attribution may rely on that, but it may introduce unexpected overhead, or miss attribution from the beginning of those long tasks
    - Baking sampling into opt-ins may help reduce the overhead for the general population
- Apple concerned about performance implications and inconsistencies. Wants to further study the polyfill's performance improvements
- Microsoft sites may be good candidates for the Origin Trial

Minutes

Andrew: sampling profiler for user JS
… ECMA realm level isolation + only same origin / CORS exposed scripts
… current status: in blink/v8 and origin trial in M78
… constraints we arrived at: coarse sample interval on Windows (16ms)
… symbol map built on profile start, forced us to do some expensive work at start
… requires COOP/COEP as this has similar constraints to shared arraybuffer's
… Let first-party page specify if profiling is enabled.. What impact does it have? Would make it more challenging for 3rd party analytics to collect this.
Nic: Akamai could inject headers, but analytics on its own cannot - so would make some scenarios harder
Yoav: Considered Feature Policy for the opt-in?
Andrew: brought it up on a call, but subframe propagation was considered unnecessary
Ilya: Do you want subframes to enable profiling?
Andrew: interesting point
Yoav: motivation for opt-in is performance?
Andrew: Mostly. Also concerns that random 3P scripts would enable this
Ilya: Sense for the overhead?
Andrew: overhead: 10ms samples → ~4-5% overhead, but implementation specific
Todd: it is possible to build 1ms sampling profiler with 1% overhead. Windows built-in profiler does that.
Andrew: Also need to integrate this with WebDriver, to make it testable.
Yoav: combination of COOP/COEP and opt-in will make it harder to enable at scale. May be necessary, but will have that impact.
Andrew: Would be great if we can find a way to avoid adding that restriction.
???: How is sampling done?

Andrew: Sampling is done at the UA level using a sample interrupt. Part of the reason why the UA can override the developer request sampling interval. Current implementation is fairly conservative.
Steven: can we inspect extension code?
Andrew: only if it's CORS enabled. Only things that would show up in error stacks today

Tim: hope is that we can reuse this for LongTask attribution
… e.g., if you cross a threshold of task length, we start collecting samples
… this wouldn't give you attribution at the beginning of the task
Yoav: Can't we use the LT registration as a signal to profile?
Tim: That would add overhead to all your tasks. We want to sample only during LTs
Andrew: The first time the profiler is enabled, would need to walk the heap
Tim: Need to opt-in in advance. We want to see if people would need that attribution separate from the profiler, or if people would just turn on profiler and use timestamps to correlate.
Pierre: if you start the profiler only after the task is running, you won't know what started the task
Alex: Want to make sure most users don't have that enabled. Feels weird to turn it on as a side effect
Yoav: Maybe we need sampling for long task attributions. Sampled registration can help here
Tim: other vendors have thoughts?
Marcus: yeah. The restrictions seem necessary. Will need to recompile code when enabling profiling, so there would be an initial overhead. But if that's expected, that seems doable. The main problem is that currently it's not yet stable for production use.
Yoav: going back to sampling, an explicit opt-in that tell the browser to kick off profiling ahead of time, might be good to bake in sampling into that.
Steven: How would that look like?
Andrew: Header or meta tag
Yoav: meta tags are typically easier to deploy
Nic: If the goal is for the origin to opt-in, can 3P JS add that
Andrew: maybe only pre-parser based
Marcus: Developers may start to rely on specific strings they get from JS stacks, or if DOM calls are exposed. That can create compat issues.
Andrew: DOM calls seem relevant. The strings in JS stacks should fall from ECMA-262
Marcus: should the spec discourage people from relying on specific strings?
Andrew: Good point. Need to investigate DOM exposure.
Alex: Concerned by potential perf implications and inconsistencies in performance.
Andrew: At Facebook we serve instrumentation to all users, this will allow profiling only for a sample.
Alex: Want to see the list of issues fixed in the polyfill.
Marcus: same!
Todd: Microsoft sites are likely to want to get on the origin trial bandwagon.

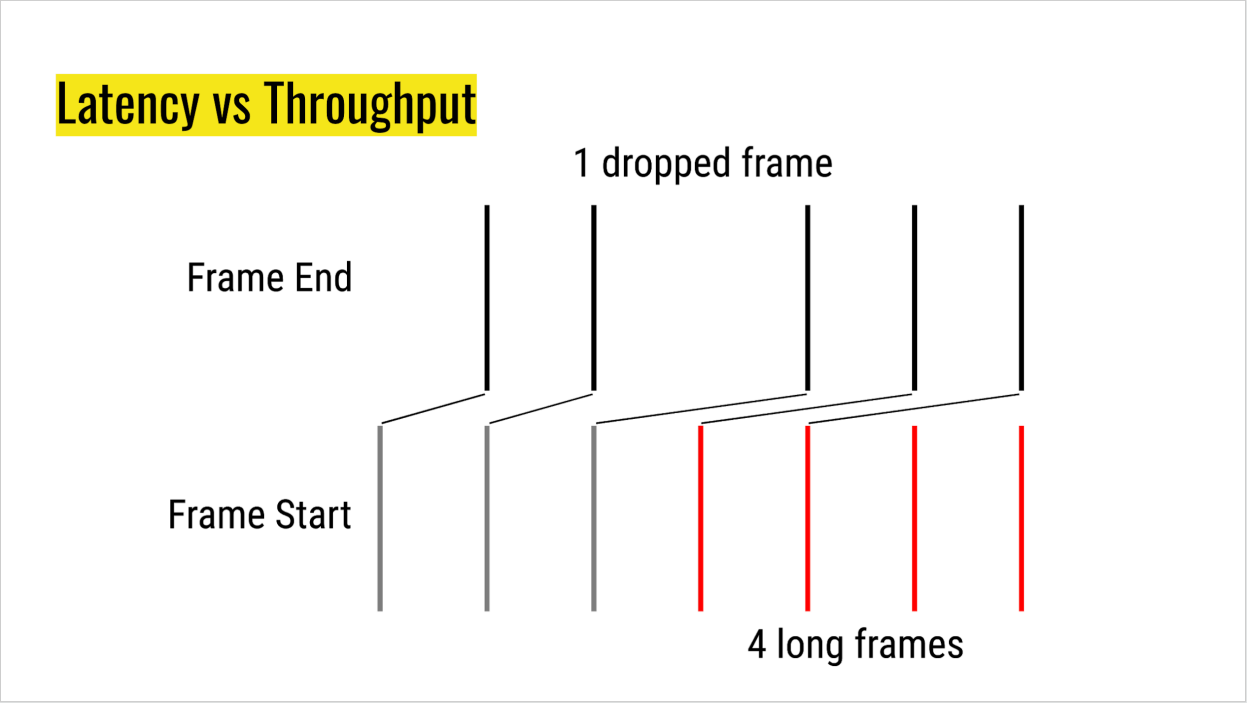
Frame Timing

TL;DR

- Use case seems legitimate. Everyone agrees that rAF polling is awful.
- Dropped frames seem more useful than long frames.
- Apple would have a hard time reporting dropped frames due to their architecture
- Excel would love to also have some coarse attribution for dropped/long frames

Minutes

Nicolás: Motivation is to measure how smoothly content is being rendered
… prevent expensive rAF-polling
Tim: dropped frames vs high-latency frames



In the above scenario, should we report the single dropped frame, or all the long frames that happened after it. Latency is already captured by event timing, so may be enough.
Yoav: is scrolling captured by Event Timing?
Tim: we don't do a good job of it today, but intent to make it work. Seems like we want to only surface the dropped frame.
Ilya: why should developers care about latency?
Tim: In cases of input, which Event Timing gives you
Noam: for us (excel) 60Hz is not realistic…
… when you switch to presentation mode, etc, we go down to 30Hz
Tim: we could provide a skip threshold (e.g. min 2)

Todd: as a web developer, how do I distinguish between 4hz vs 8hz vs 16hz…
Tim: 2 cases. Changed rate vs. missed a frame. Maybe we need to surface refresh rate for FT entries.
Yoav: Is frequency exposed today? We tell developers to rely on that…
Tim: Interval between rAFs, but that's expensive to measure
Marcus: different browsers do different things. Firefox will run rAF at 120Hz
Tim: Need to surface something about the screen's refresh rate for developers/analytics to make sense of this
Ilya: if a developer aims at 60Hz rAF and then hits a system running at 120Hz, that's a lot of skipped frames. What should they do with the telemetry. Surfacing refresh rate makes sense.
Todd: Excel doesn't care about anything faster than 30fps
Yoav: Makes sense to define what is long
Tim: Maybe let pages say to report only X dropped frames in a row
Noam: current rAF workarounds have many problems: expensive and impacts users. Also losing rIC calls.
… Want more attribution for why a frame was dropped - would enable us to focus on where to optimize
Yoav: do you need specific attribution or pointing towards the subsystem that caused it?
Tim: Specific attribution is a can of worms
Noam: Would be great to have attribution for long tasks if we have it. But big buckets pointing towards rendering/JS/GC would be a good first step.
Olli: and devtools profilers are not enough?
Noam: Can use them only when you have a repro, and we typically don't have the user's content
Ryosuke: Can you explain issues reported by users?
Noam: Trying to solve animation smoothness as well as responsiveness. But even if we knew when that happened, it's hard to know what was the cause.
… We're measuring rAFs once user interacted, for 5 seconds. So we miss the initial interaction. Event Timing can help with that.
Ryosuke: so you want to measure all missing frames all the time?
Noam: Any frame above 50ms. 30Hz is fine for us.
Tim: Why doesn't event timing tackle your use-case?
Noam: We get the event, we set a timer for 5 seconds, in which we measure the frame rate. Any interaction during the window, extends the window.
Todd: dropped frames can be unrelated to input when another user is editing the spreadsheet
Nicolás: right now no concrete proposal for attribution for long frames. Seems complex
Noam: attribution is not the key as long as we can capture everything
Nicolás: given your rAF polling, this will probably be useful for you
Todd: first priority is getting rid of rAF, right?
Noam: yeah. It will also enable us to collect more data for user analysis
Nicolás: can they detect bad ads OOP?
Todd: yup, in busy systems
Marcus: for Excel, long tasks should be sufficient

Tim: Not really, because LT's threshold may not fire. Also, Frame Timing can also account for compositor times in some platforms.
Todd: Can we replace the idea of dropped frames with the idea of running that code only when there are frames to render?
Tim: Tricky because we may not produce frames when there's nothing to produce. Also tricky due to visited links when we can't expose if a frame was produced.
Marcus: for the case of users hitting hard-to-reproduce issues, we ask them to capture a profile and send it.
Noam: won't work for us for many reasons.

Tim: how would analytics providers show that data?
Nic: Already use rAF for short periods of time. Report average frame time.
Tim: what do you surface?
Nic: average frame rate. Tracking individual deltas to show where the long frames happened. Long frames and dropped frames would be more valuable
Tim: Would your users use that data?
Nic: if they can figure it out, kinda like long tasks. Data is not necessarily actionable

Tim: thoughts on reporting dropped frames vs. long frames?
Ryosuke: we don't "expect" frames, especially in variable frame rate scenarios?
Yoav: Does it make sense to look at frame intervals?
Tim: Can be confusing when rate is variable. We can try to figure out what a definition of a "dropped frame" would look like and ask for feedback.
Todd: Use cases I heard require correlating the frame with some action
Tim: We'd surface timestamps
Todd: And is it all about the compositor thread? If all frames are long, that'd impact responsiveness, but maybe event timing is good enough for that.
Tim: yeah. Want to avoid penalizing cases where latency doesn't matter.
Ryosuke: We may not know when a frame was dropped.
Tim: You'd know on the main thread when a frame failed to produce
Ryosuke: no. We don't know what the deadlines are, so not know that we "missed"
Tim: So can have a rough approximation, but not better
Ryosuke: in some cases, we don't even know that.
Todd: What about long frames?
Ryosuke: Seems clearer.
Simon Fraser: Can report on the difference between scheduling the rendering update and the end of the rendering update. But JS can run between those.
Tim: Also use cases doc has great examples, but maybe need more concrete examples.
Noam: happy to provide more concrete use cases.
Todd: screenshots with specific updates?
Tim: yeah
Todd: are you favorable of the idea in general?
Ryosuke: the use case seems legit, the question is can it be useful and implementable

Tim: Thanks for the feedback!

Largest Contentful Paint

TL;DR

- Mozilla wanted to hear feedback from shipping LCP in Chromium.
- Akamai would be able to correlate LCP scores to business metrics in a few months
- Mozilla and Apple curious about heuristics picked and process
    - Chrome can show filmstrip results and correlation to SpeedIndex
- User happiness is hard to measure
    - Wikipedia is using surveys

Minutes

Ben: excluded background images?
Nicolás: only for background of the body
Tim: in our analysis we saw a large enough fraction, had to exclude it
Marcus: want data regarding user satisfaction
Nic: Can correlate business metrics to LCP scores in a few months
Tim: Once we have the data, what's the bar to adoption? It'll be better than FCP but not necessarily great on its own. We can do a better job at surfacing anecdata
Todd: also business metrics and user happiness are not the same. LCP could vary between sites. Slowing down sites could give us some data
Tim: what does that give?
Ryosuke: we can A/B test different heuristics
Yoav: hard to collect user satisfaction
Gilles: slowing down makes users go away for a while, so not ideal
Tim: we can show anecdotes. Also correlation to SpeedIndex.
Alex: A good way to find if LCP is a proxy for element timing. Once element timing is adopted, that would give us a good way to see if we hit or miss.
Gilles: LCP might be useful for a websites that want to separate rendering of their site vs. the ads. Otherwise, most of the stuff will appear all at once. Studies we've done show that it's poorly correlated. Have you looked at real world cases?
Tim: yeah
Noam: For excel it would measure the skeleton content more than the real content
Gilles: Common case for SPAs
Tim: So we have a heuristic for splashscreen. It would handle some cases but not all
Steven: LCP is the best solution when we don't have element timing. Need transitionStart to measure LCP with user satisfaction
Tim: correlation data should be easier to get out than film strips
Todd: LCP is multi-modal. Sites that measure user happiness scores can correlate those modes.

Gilles: not clear how we evaluate LCP's usefulness.
Ryosuke: FCP is clearcut. LCP is more complex
Tim: yeah, but FCP doesn't correlate to user experience
Ryosuke: sure, but not clear how we measure if this heuristic is good or bad
Gilles: SpeedIndex is not great
Ilya: SpeedIndex improvements had direct impact on business metrics
Ryosuke: how much of the heuristic complexity is actually needed? If future pages change behavior, do we change the heuristics?
Tim: Maybe we should be versioning these things?
Todd: We're all theorizing about what feels good, and have good data. But we don't know how these things *feel* to users
Gilles: need more systematic study to correlate to engagement and dollars
Todd: On ad revenue sites, dollars don't equate user happiness in many cases
Gilles: collecting 24K user surveys a day. Saw happiness increase in spanish wikipedia, and that correlates to people upgrading devices. Didn't pursue LCP, because Element Timing is better.
<ongoing discussion>
Tim: next steps?
Marcus: sharing filmstrips and speedindex correlation would go a long way. Higher rigor would be great, but really hard. Also measuring correlation with the other metrics, where if it's an independent variable would be more interesting.

Resource Timing Visibility

TL;DR

- Akamai proposes same-origin iframes to be able to opt-in to report their entries to their same-origin parent
- Not just for ResourceTiming, but to be able to surface UserTiming/etc as well
- Opt-in via some sort of HTTP header or Feature-Policy (with * or origins specified)
- They have partners that are interested in the API - would be good to talk to them directly to better understand the use case
- Resource Timing is not consistent across browsers
    - Apple not opposed to support the different sizes attributes, assuming TLS decoding is not included
    - Entries are not consistently reported (aborts, HTTP error codes, videos)

Minutes

Nic: Resource Timing doesn't give you the full picture: no TAO, cross-origin iframes, so can be misleading.
3 types of resources: full RT, restricted (no TAO) and invisible.
Safari doesn't support sizes

Alex: No reason not to expose this if that data doesn't include TLS overhead
Nic: want to enable cross-origin iframes to opt-in to share their RT data with the parent
Tim: Can people want to expose some entries but not others?
Nic: heard people that want to surface UT, but may not want the rest
Tim/Todd: we should start simple with an everything/nothing proposal
Tim: site should be able to opt-in to a specific host
Nic: 2 partners right now - 1 serving iframes from multiple origins, other is ad visibility network
Yoav: would be good to bring those partners on a call
Will: why are they not doing it in a hacky way right now? Would be interesting to hear that
Benjamin: getting more accurate RT data would be a good thing this group can do
Tim: value of RT data today?
Nic: check resource sizes and check byte size
Tim: unclear what the ad provider incentive would be
Nic: some large companies can use that as a carrot for third parties

isFramePending

TL;DR

- Discussed the different issues this proposal raises

Minutes

Stefan: very similar to isInputPending, but letting the developer know if there's a rendering frame pending
Large frameworks expressed interest.
Issues:

- Violates "run to completion" semantics - same as isInputPending
    - Tim: the fact that it's a method call makes this simpler
- Testability - hard to write WPTs
    - <discussion on rendering opportunities>
    - Tim: should be when rAF is scheduled, not at vsync time
    - Smfr: in iOS the vsync is handled on the main loop, so this will be difficult
- Detects dirty render
    - Tim: Visited links and cross-process iframes?
    - Stefan: we schedule a frame anyway in those cases
- Detects process isolation
    - Tim: also allows you to know if a cross-process iframe is producing frames, if you're not process isolated
    - Todd: Could also do "when is a render step going to run"

- Stefan: on some platforms we get explicit vsync and on some we don't, so we need to ask the OS for a frame. If we're not explicitly asking vsync, we'd get timer skew. Important case is when we're not constantly animating
- Tim: maybe we can lie on the first frame
- Stefan: as long as it's possible to specify and answers the use-case
- Marcus: Can we request a vsync once it's called?
- Stefan: would prevent us from running e.g. 100ms of uninterruptible script. Would be sad if we have to schedule a frame
- Todd: Is uninterruptible script a goal?
- Stefan: need to talk to partners
- Alex: might be nice to allow scheduler writers to unite isInputPending and isFramePending. isSomethingPending that returns an enum
- Stefan: need to figure out if detectability of dirty render is really a problem
- Tim: Use cases document please
- Alex: prototype implementation? Would be good to show before/after

Spec review

Performance Timeline L2

AI [plh, siusin]: transition to CR, send out call for review
AI: fix the flaky case sensitivity test and Frozen array test on supportedEntryTypes
AI: re-republish the WD from L2

Resource Timing L2

AI [plh, siusin]: send out call for review; not blocking on closing all issues
AI [plh, siusin]: transition L1 to REC No open issues

Navigation Timing L2

AI [plh, siusin] send out call for review

User Timing L3

AI [Nicolás] add Chrome use counters
AI [philip] fix explainer, reach out + do some activation

Page Visibility L2

AI [todd] chat w/ PLH about need, or not, for wide review
AI [todd] https://github.com/w3c/page-visibility/milestones/Level%202

Beacon L1

AI [] fix failing WPT, otherwise “ready to go”

requestIdleCallback L1

AI [yoav] Resolve https://github.com/w3c/requestidlecallback/pull/78 → REC, have 2 green implementations

Preload L1

AI [?] defining Preload cache is a blocker

Server Timing L1

AI [plh, siusin]: send out call for review

Test Triage

- Performance Timeline
    - Flaky test
- https://github.com/w3c/resource-timing/issues/200
-

Hackathon Issue Discussion

- Navigation Timing

Boris raised 2 issues. #114 - Spec for PerformanceNavigationTiming.type does not match any implementations - hopefully only definition change + tests, so no need for discussion

    - #115 - Definition of "back_forward" navigation type does not make sense - discuss desired behavior

Yoav, Will and Todd discussed and agreed that for location.href, the type should be back_forward before location.href is set and navigate after the location.href navigation. The spec should be reviewed to ensure the language allows for this. For a redirect after a back navigation, the final type should be navigate. The spec should be reviewed to ensure the language allows for this. It is likely that the HTML spec must be updated and then the Navigation Timing spec will need to reference.