Discarding, Throttling and Freezing Background Pages

BlinkOn 9

altimin@, chrisha@, fmeawad@, panicker@google.com

Getting the Background work out of the way

- **Discard** background tabs that are not useful to the user to free resources
- **Delay** loading new background tabs
- Create a structured **lifecycle** for the app to maintain the state and inform the developers
- **Throttle** and **freeze** background tabs and workers to free CPU
- For discarded tabs, enable work to be done using **Background APIs**

Discarding Background Tabs

- If resources get tight, discard least important tab (since ~M55, refinements in M68 - M69)
- If expected tab utility drops below a certain threshold, proactively discard (M68 - M69)
- Initially using heuristics to be mindful of user pain, but new platform APIs to handle valid use cases (M70+)

Delay loading background tabs

- Same as discarding logic, in reverse
- Load most important tabs first (M66)
- Budget-based scheduling to ensure don’t exceed CPU, network and memory budgets while loading tabs (M68 - M69)
- If predicted tab importance too low, or system resources too tight, delay loading indefinitely until tab focused (M68 - M69)

Delaying and discarding background Tabs

For more details, please find and talk to:

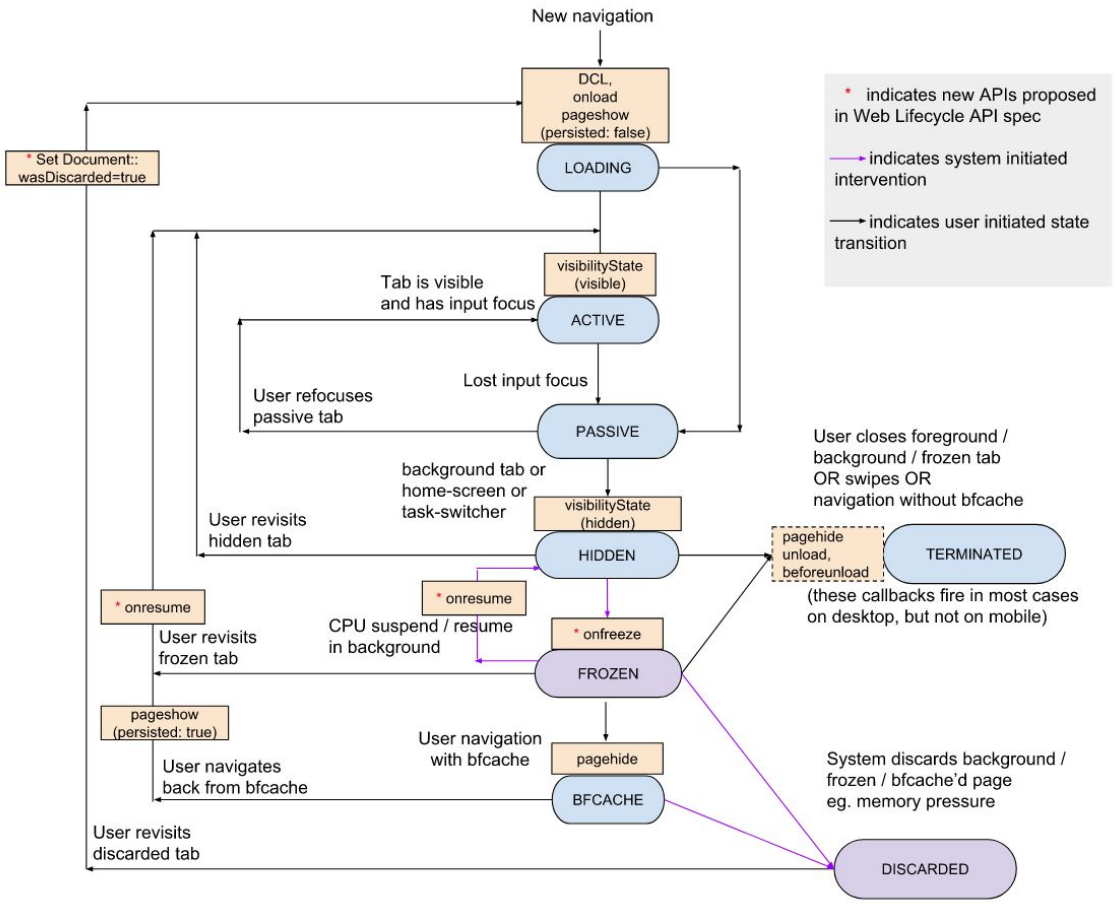
chrisha@, fdoray@, sebmarchand@

Lifecycle for the Web (update)

Since last BlinkOn talk we have:

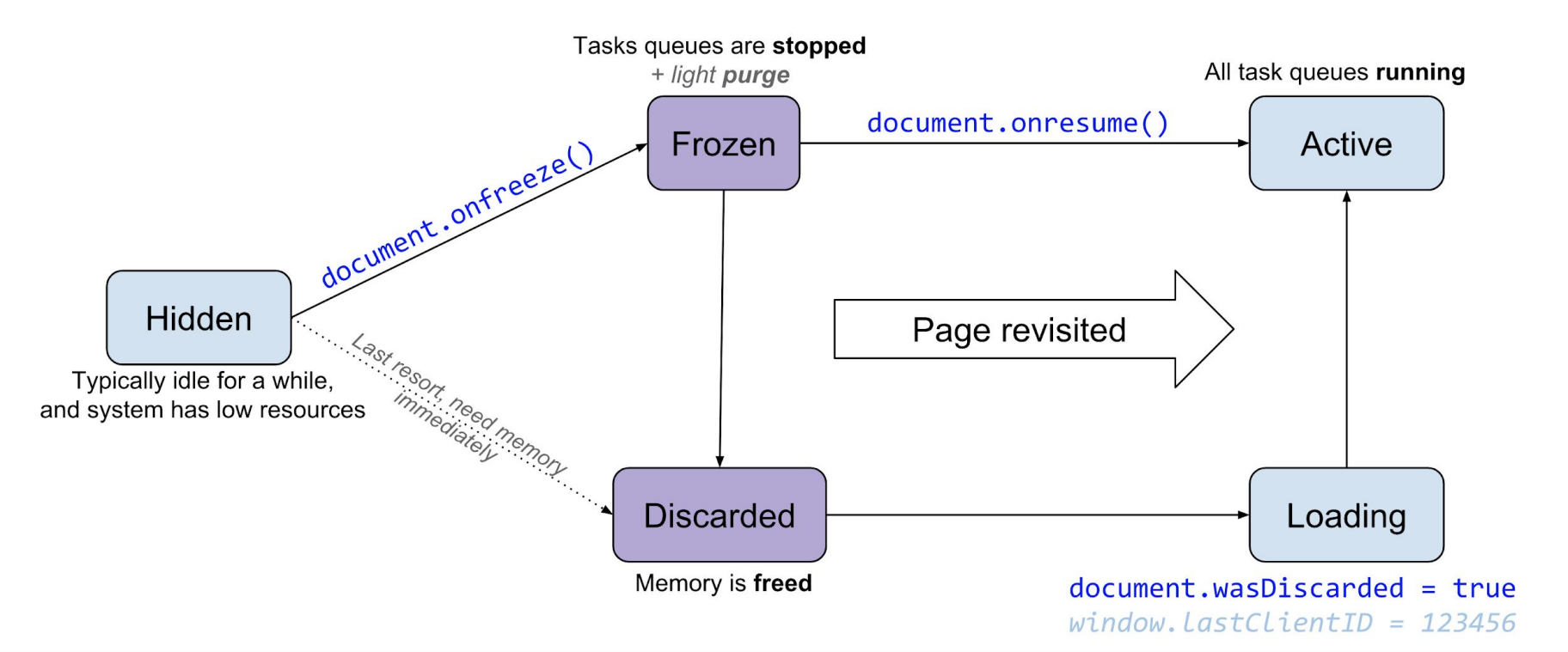
- Refined the APIs and built consensus from other vendors
- Spec drafted
- Implemented the API behind a blink runtime flag: PageLifecycle (M66-M67)
- Integration with chrome://discards for testing

Current Design



More at:
https://github.com/WICG/web-lifecycle

A bit zoomed in (and simplified)



Developer Expectation

State Transition	Expected Developer Action
HIDDEN → FROZEN	Report to analytics Teardown, Release resources Hand off for background work and stop execution Save transient UI state in case app is moved to DISCARDED
FROZEN → ACTIVE	Undo what was done above Report resumption to analytics
FROZEN → DISCARDED	(they cannot do anything)
DISCARDED → ACTIVE	Restore transient UI state

Code Example

```
const cleanUpFinished = function {
  // Close any open IndexedDB connections.
  // Release any web locks.
  // Stop timers or polling.
};

const reInitializeApp = async () => {
  // Restore IndexedDB connections.
  // Re-acquire any needed web locks.
  // Restart timers or polling.
};

document.addEventListener('freeze', cleanUpFinished);

document.addEventListener('resume', reInitializeApp);
```

- Please try the new API and let us know what works and what does not
- Reach us at chrome-lifecycle-apis@google.com OR file an issue at https://github.com/WICG/web-lifecycle

Demo

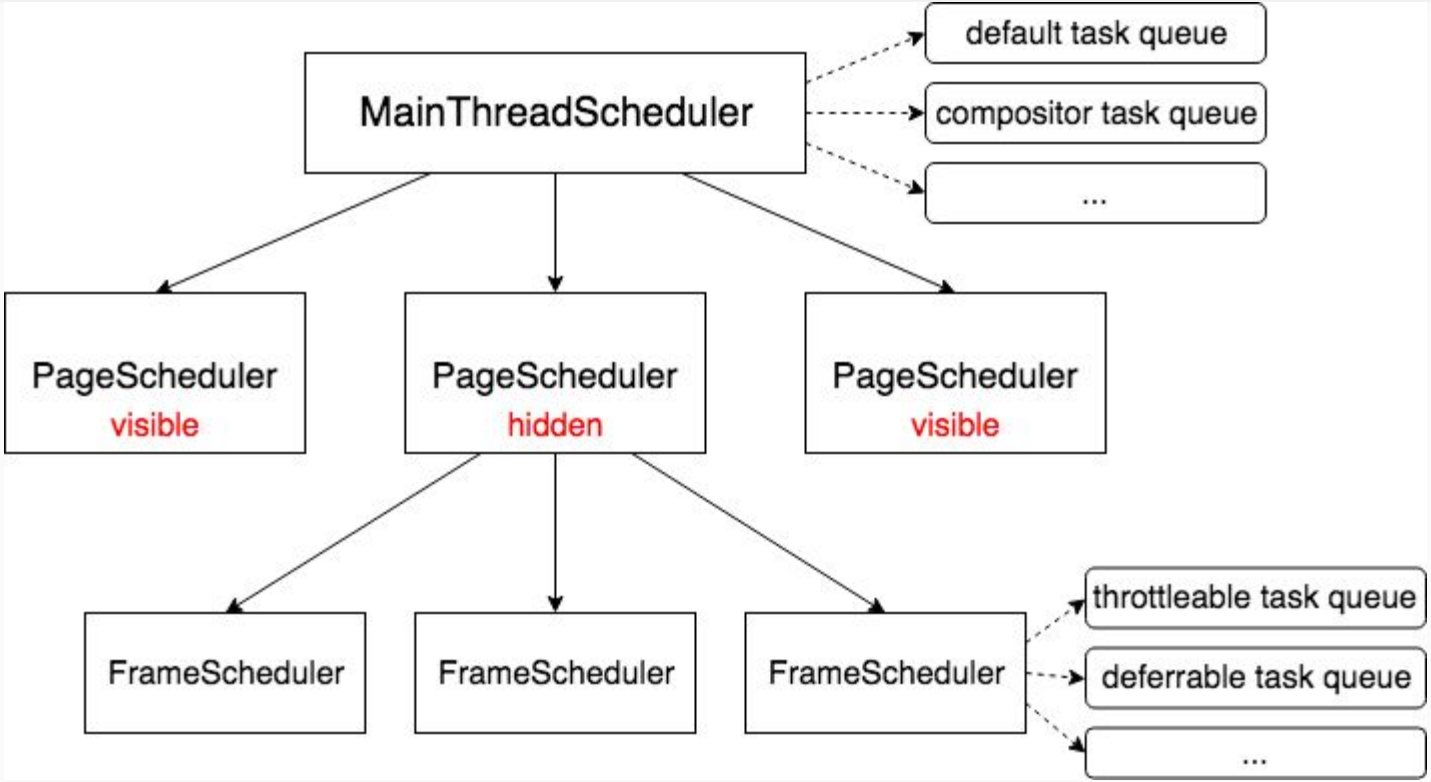
Freezing & throttling

- Moving to lifecycle world step-by-step
- Freezing:
  - Invoking onfreeze() callback
  - Doing it more frequently and affecting more tasks
- Throttling:
  - Limiting ability to do arbitrary work in the background tabs
  - Expanding to more types of work
  - Increasing aggressiveness
  - Removing opt-outs

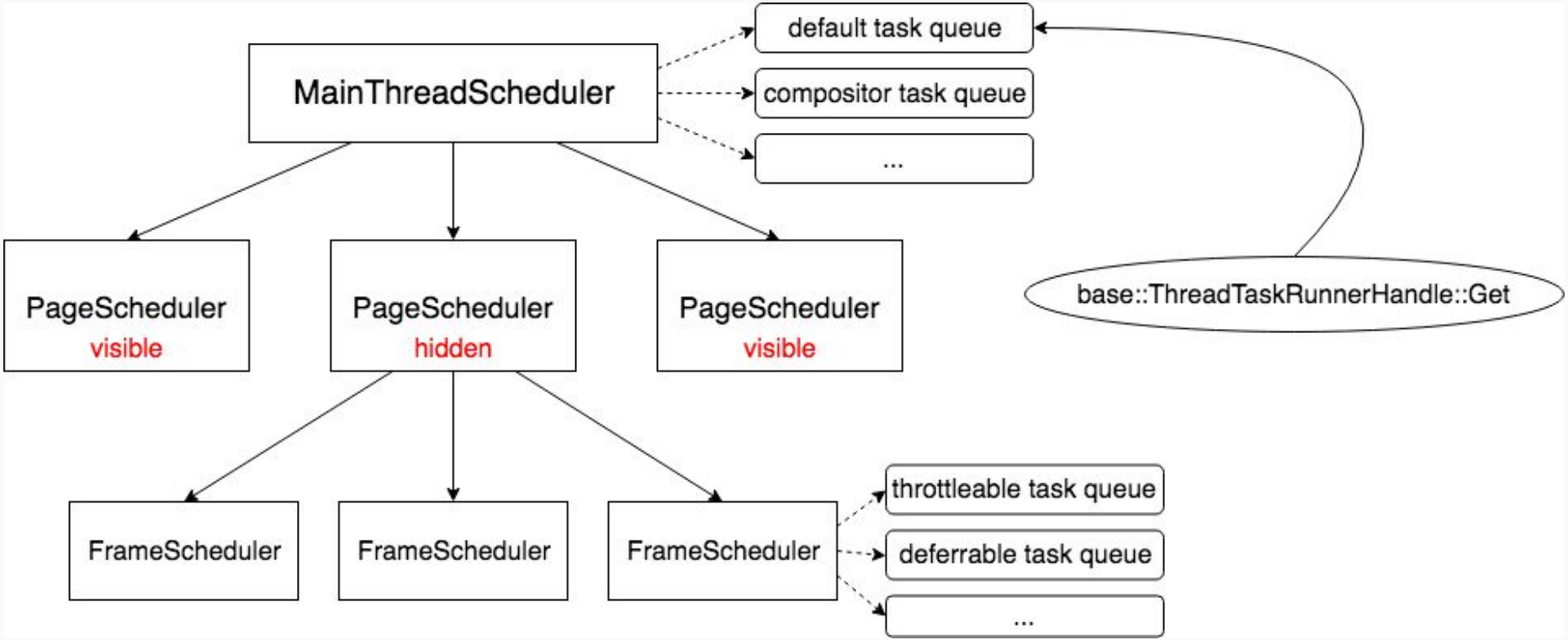
Per-frame scheduling

- “Default” task queue:
  - base::ThreadTaskRunnerHandle::Get()
  - No scheduling metadata
- Converting callsites to use per-frame task runner:
  - WebLocalFrame/RenderFrame::GetTaskRunner(blink::TaskType::kMyTaskType)
- Number of tasks in the default task queue dropped from 20% to 4%.
- For more details please talk to hajimehoshi@

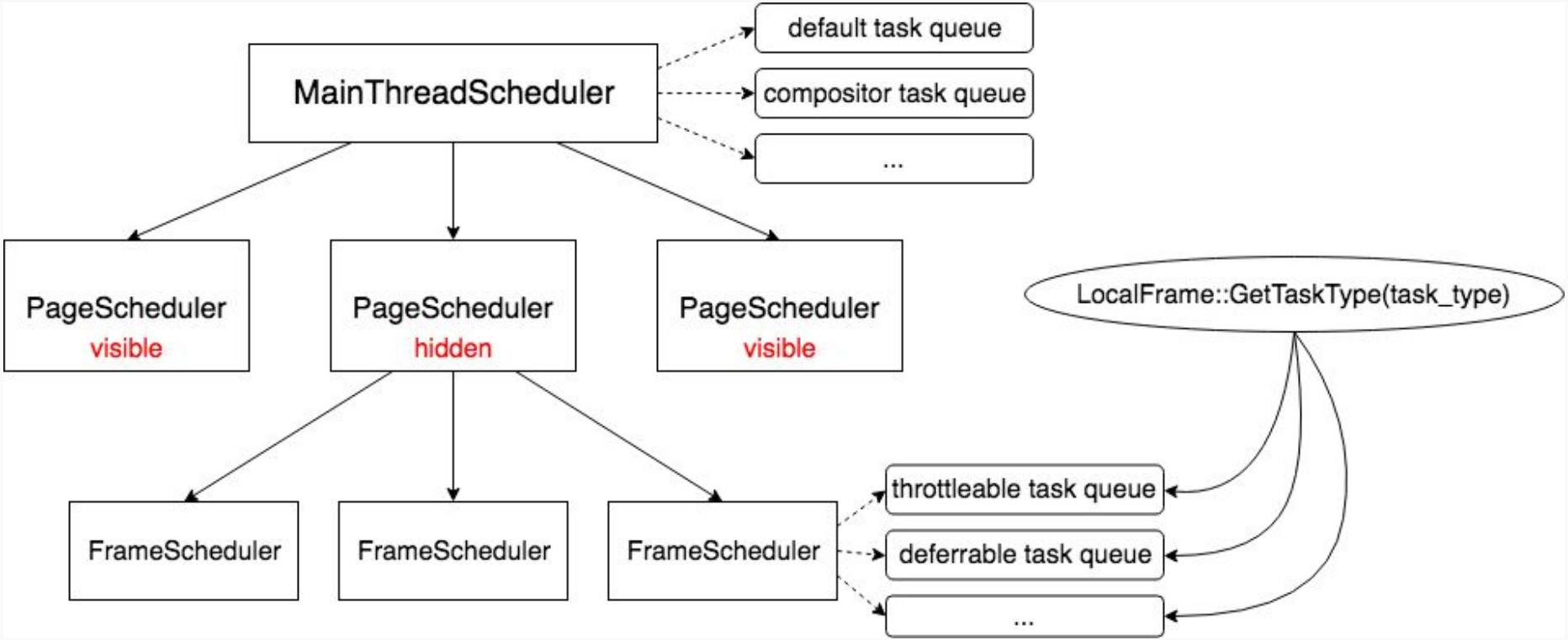
Per-frame scheduling



Per-frame scheduling



Per-frame scheduling



Per-frame scheduling

- “Default” task queue:
  - base::ThreadTaskRunnerHandle::Get()
  - No scheduling metadata
- Converting callsites to use per-frame task runner:
  - WebLocalFrame/RenderFrame::GetTaskRunner(blink::TaskType::kMyTaskType)
- Number of tasks in the default task queue dropped from 20% to 4%.
- For more details please talk to hajimehoshi@

Freezing

- Transitioning from **HIDDEN** → **FROZEN** state
- We're not running any javascript tasks*
  - Starting with timers & loading tasks as seen in the demo
- This change can break web content (i.e. chat apps refreshing in the background)
- Gradual rollout as a series of interventions
  - Invoking it more frequently based on heuristics
  - Freezing more task types

Freezing: roadmap

- **~2013**: Stopping timers on mobile after 5 minutes
- **M68**: onfreeze() / onresume() callbacks is invoked
- **M67**: loading tasks are stopped on mobile after 5 minutes
- **M68 (experiment)**: Page is frozen on desktop after 1 hour without network activity
- **M69**: Proactive discarding smartly freezes and discards page based on the api page uses.
- ~2020: Backgrounded page is frozen after it’s done loading.

Throttling

- Addressing **HIDDEN** state
- Resource usage is limited, but page can still do some work
  - Less breaking than freezing
- Page has a budget replenishing over time
  - Running tasks reduces this budget
- Gradual rollout as a series of interventions
  - Making throttling more aggressive
  - Removing opt-outs

Throttling: roadmap

- **2011**: setTimeout/setIntervals are fired once a second in background
- **M56**: Budget-based throttling: timers are delayed to limit CPU usage to 1%
- **M68**: Throttling dedicated workers in background pages
- 2018: Non-timer tasks are throttled
- TBD: Throttling aggressiveness increases with time spent in background
- ~2020: Backgrounded page is throttled during loading depending on the foreground activity.

Implementing freezing and throttling

- It’s hard!
- Blink & Chrome were not written with the support for stopping tasks
  - Service worker failures increased when loading queue was stopped
  - Accidental throttling of IndexedDB tasks broke Google Docs loading
- Is your component affected?
  - Will things break if tasks stop running in background?
  - Please talk to us: scheduler-dev@chromium.org

Stop loading queues on mobile

- Do not run loading tasks on mobile after 5 minutes
  - Increased (and fixed) timeout failures for service worker startup
- Experiment on Beta:
  - -12% FCP/FMP for scenarios with 2+ tabs loading
  - Switch time between tabs has improved
  - Planning to ship in M67

Background APIs (early proposal)

- A tab in the background can interact by changing its Title, favicon or playing a sound ♫.
- Instead of waking up the page, let the service worker do it on behalf of the page using its `WindowClient`
  - `WindowClient.updateTitle(title);`
  - `WindowClient.updateFavicon(url);`
- Still under investigation: Who wakes up the worker? Where is the client?
- For more information, find: fmeawad@

Eng Teams

Lifecycle, MTV

- ellenpli@
- fmeawad@
- ojan@
- panicker@
- philipwalton@

Blink Scheduler, LON

- alexclarke@
- altimin@
- skyostil@

Blink Scheduler, TOK

- hajimehoshi@
- haraken@
- yutak@

Desktop, MON

- chrisha@
- fdoray@
- sebmarchand@