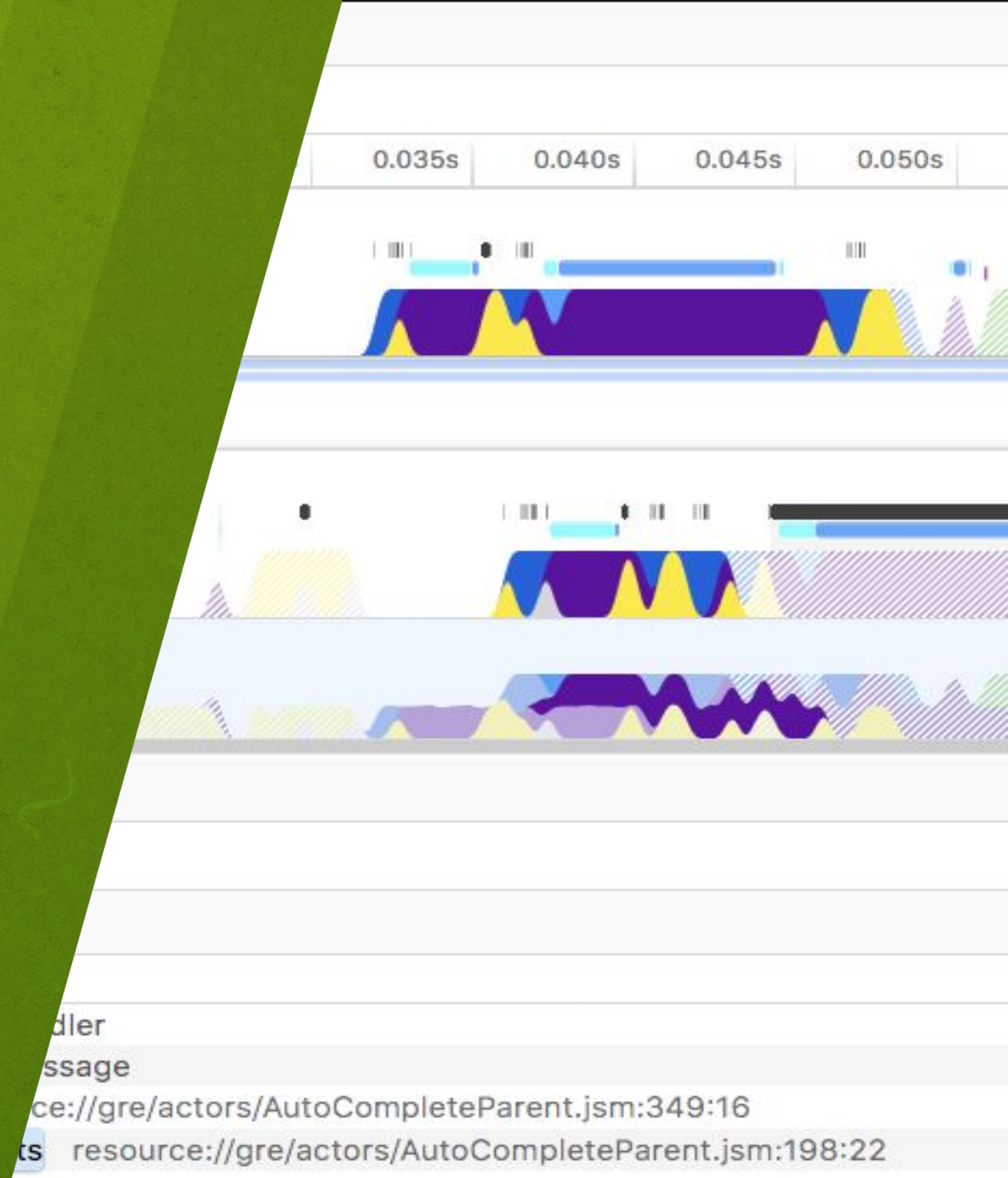
moz://a

Front End Performance & Profiling

What's slow, why it matters, and what to do about it

May 11, 2020

Bianca Danforth
Privacy & Security Engineering



Agenda

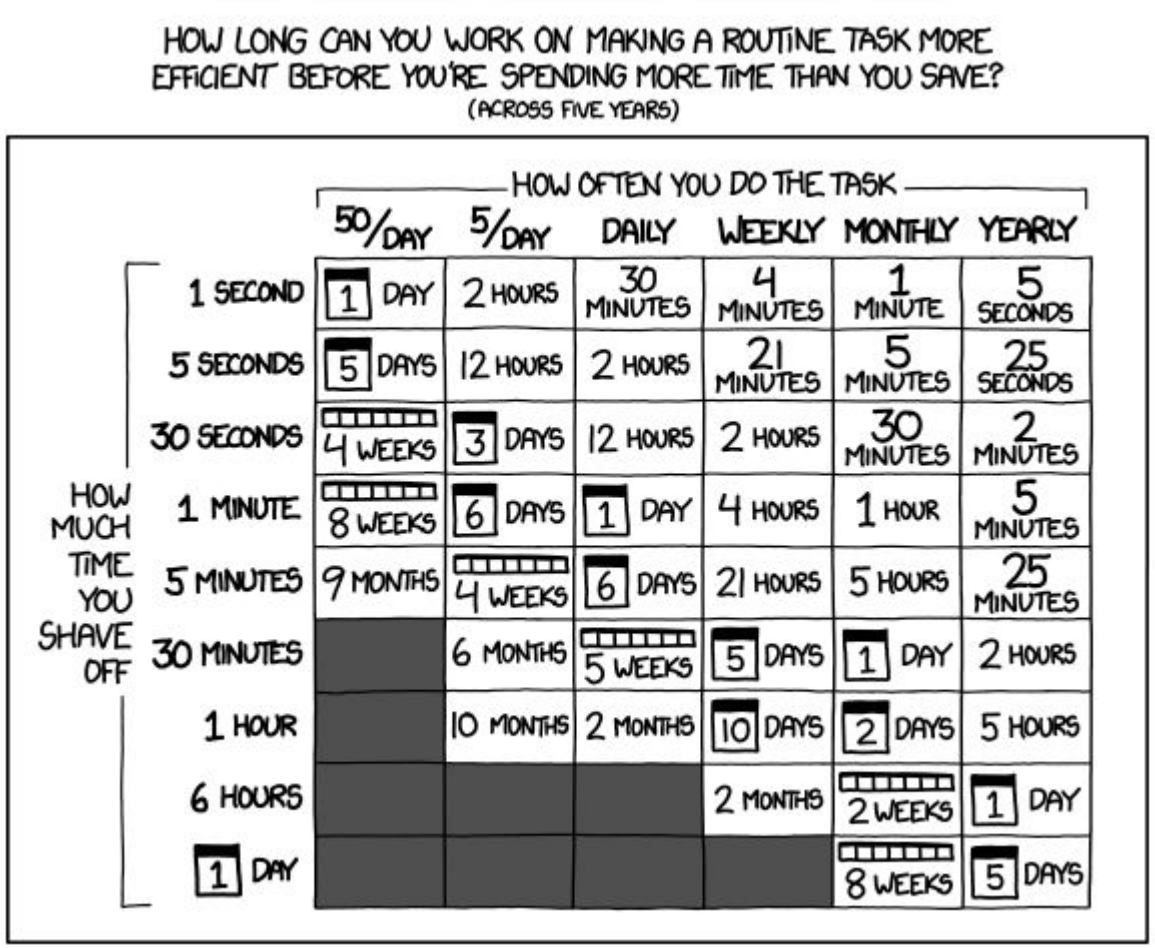
1. Why should we care?
2. Browser performance fundamentals
3. How do we measure?
4. Looking at a real performance bug
5. Q & A and additional resources

Why should we care?

Source: Firefox uninstall survey results

Context Matters

Sometimes, we just don't care as much.



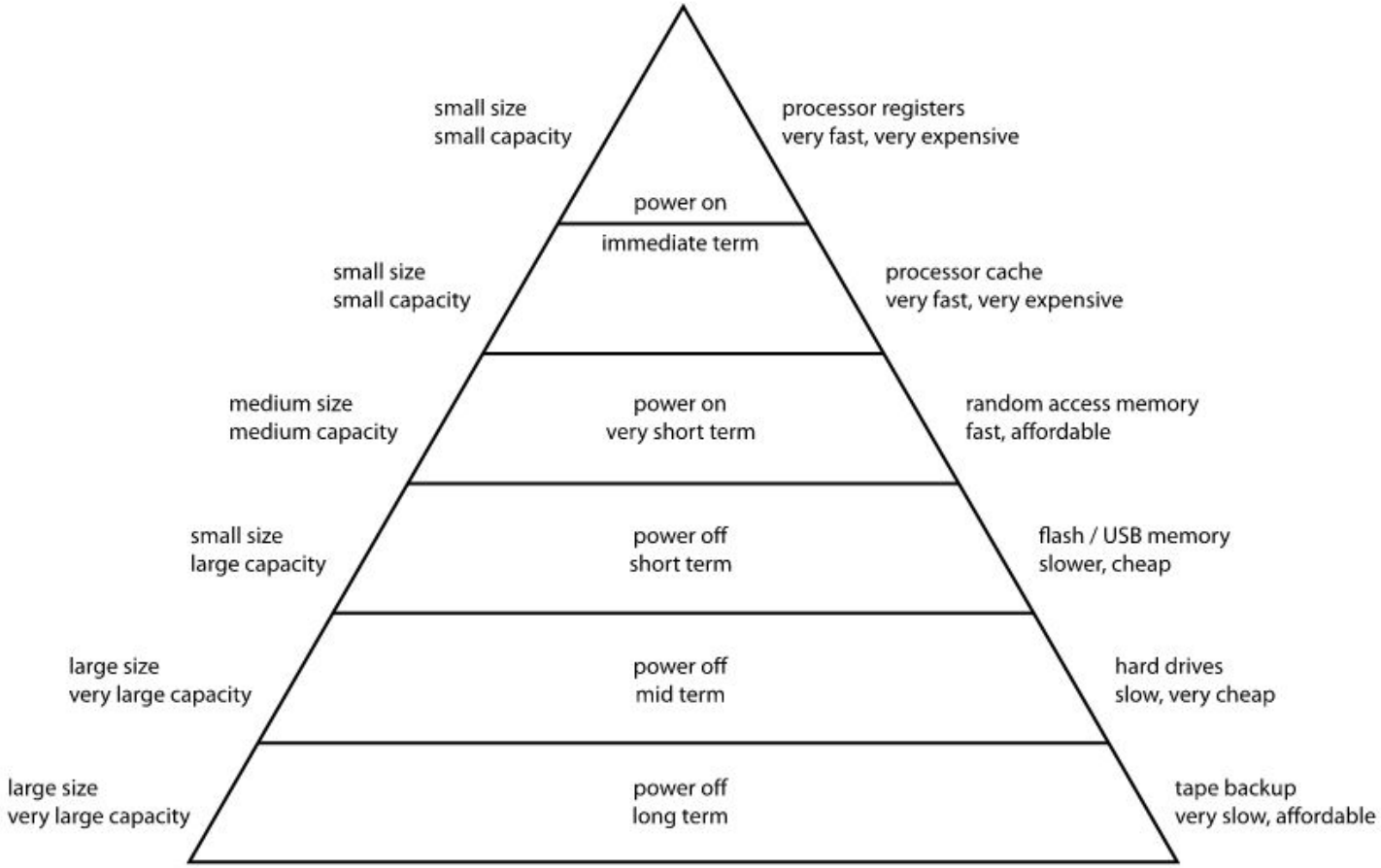
Source: https://xkcd.com/1205/

Browser performance fundamentals

Memory hierarchy, process/thread priority,
Gecko event loop and the pixel pipeline

Memory hierarchy

Each tier is larger but at least an order of magnitude slower than the previous tier



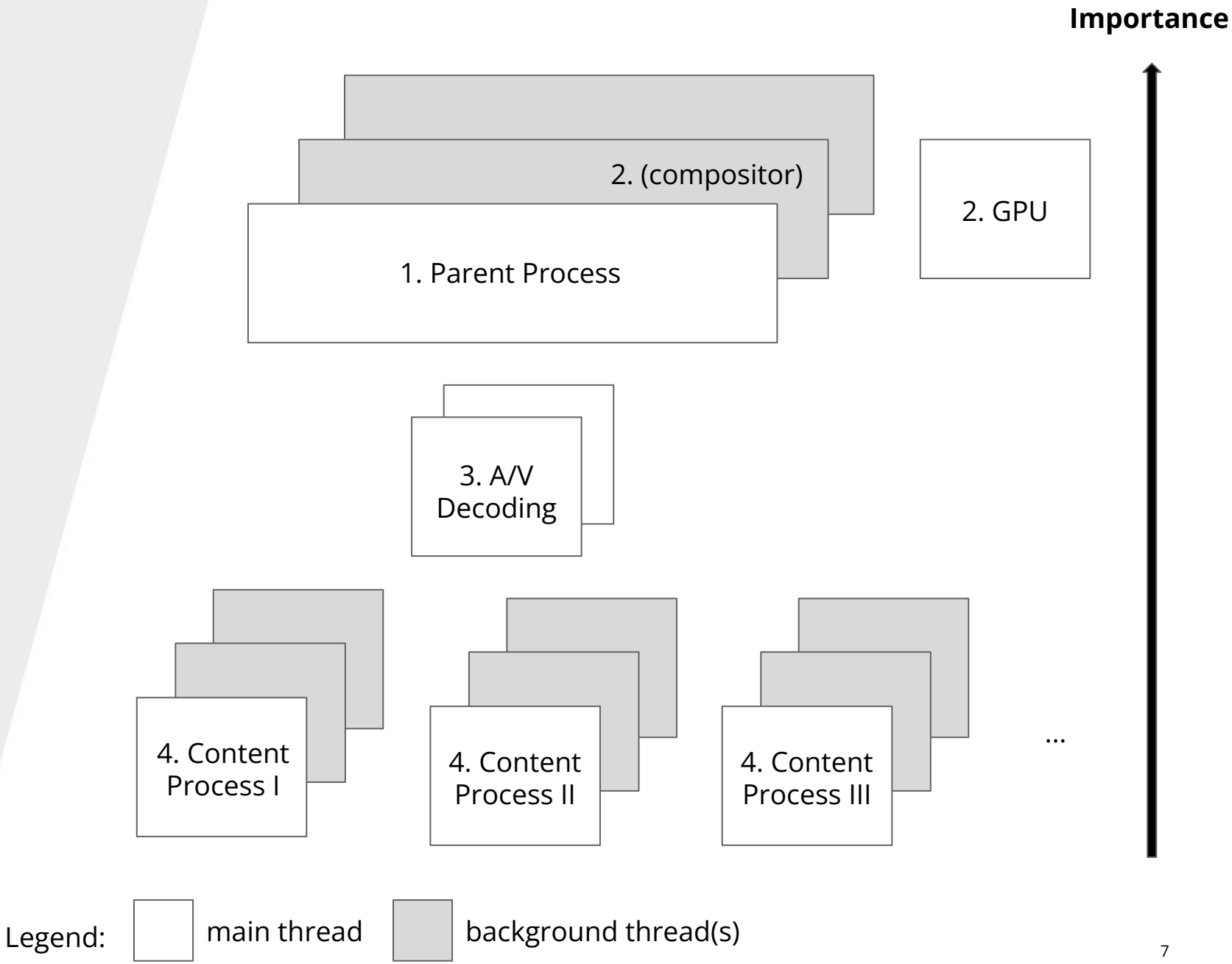
Source: https://en.wikipedia.org/wiki/Memory_hierarchy

Process/thread priority in Firefox (e10s)

The most important thread is the main thread in the parent process.

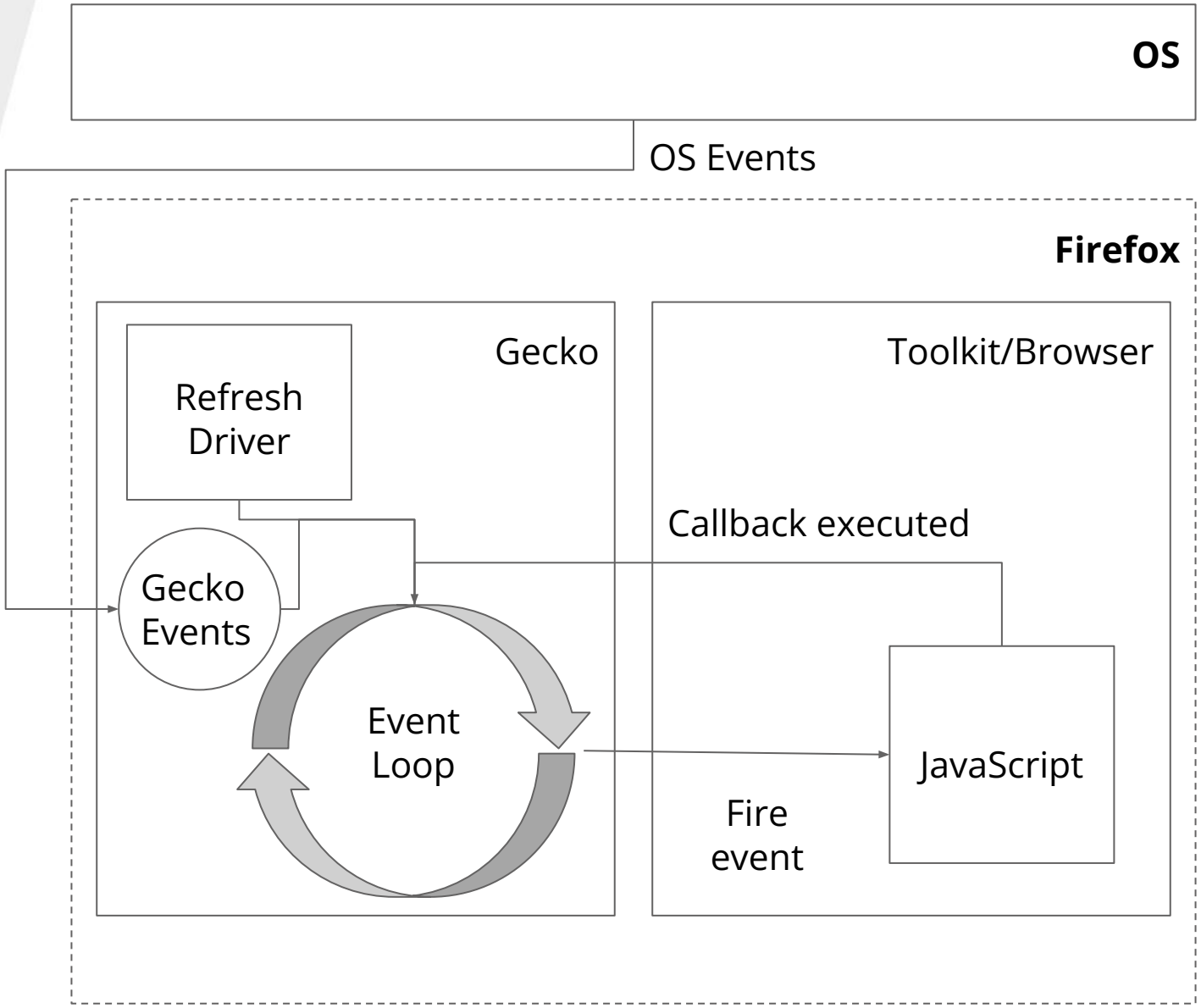
JavaScript runs in the main thread by default.

Not all processes shown.



(Gecko) Event Loop

JavaScript is constantly alternating between executing an event callback and waiting for the next event.



Pixel Pipeline

What it takes to draw pixels to the screen

Ideally, each step happens at most once per frame.

16ms frame budget

Source: https://developers.google.com/web/fundamentals/performance/rendering/

How do we measure?

Introduction to the Firefox Profiler

The Firefox Profiler

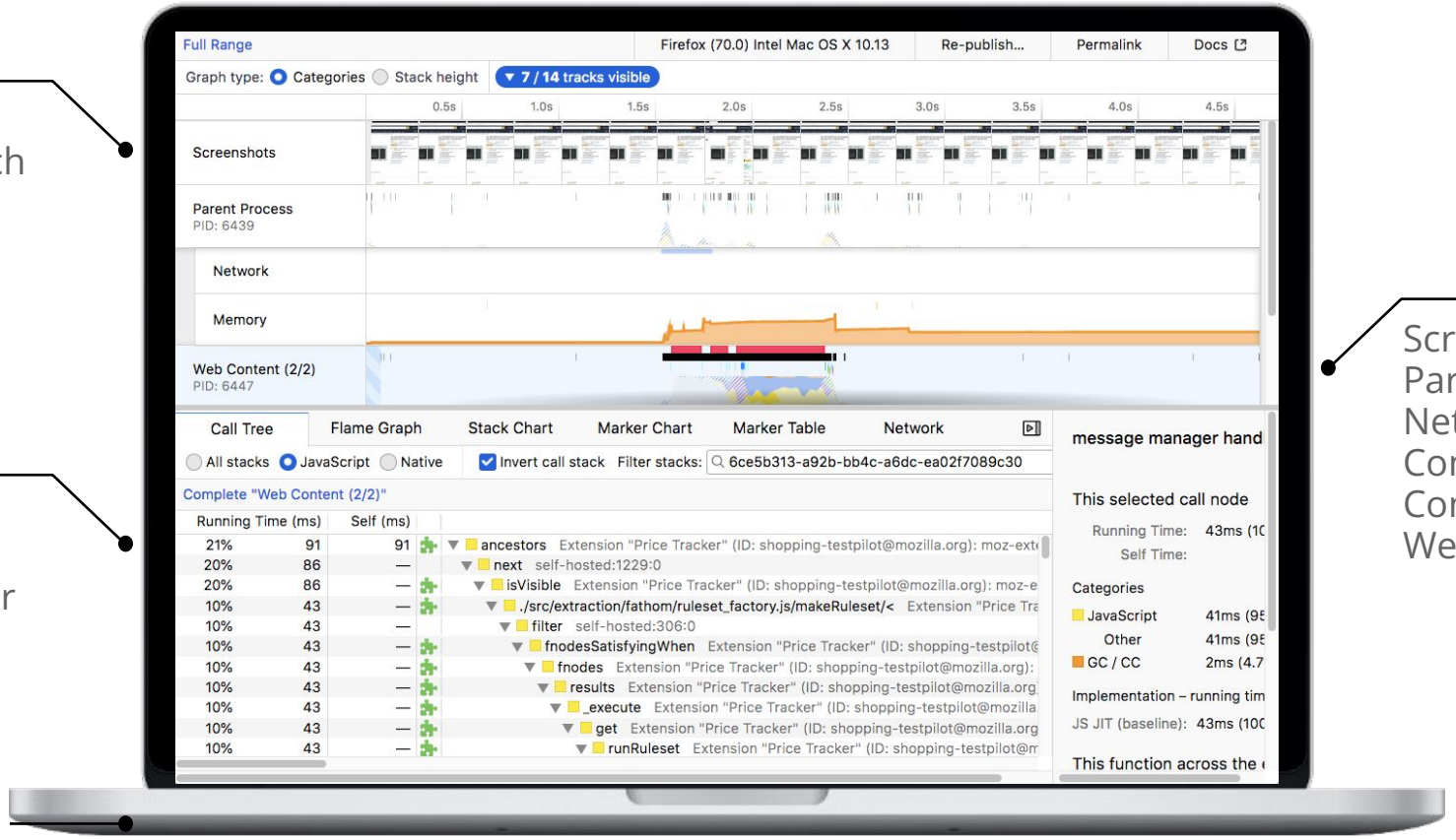
Different parts of the UI

TIMELINE

What was happening in each process/thread over time.

DETAILS

Different views based on either sample or marker data.



TRACKS

Screenshots, Parent Process, Network, Memory, Compositor, Web Content, WebExtension, ...

Source: https://profiler.firefox.com/docs, https://perfht.ml/3dzjVjl

Samples versus Markers

Samples are based on probability while markers are hand instrumented in the code

Samples

- The profiler periodically stops the thread from executing and takes a snapshot of the call stack.
- Default sample rate: 1ms
- With enough samples, the data should largely reflect the code that was run.

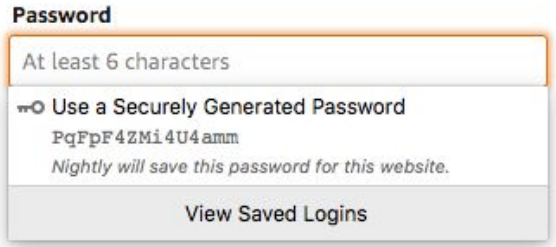
Markers

- Samples can miss Gecko events that happen quickly.
- Markers are instrumented code with a start and end time.

Source: https://profiler.firefox.com/docs

How to capture a profile

Assuming the code occurs after startup and can be manually triggered



1. Go to https://profiler.firefox.com/
2. Click “Enable Profiler Menu Button”
3. Start recording: Ctrl + Shift + 1
4. Trigger the desired code path to execute
5. Stop recording: Ctrl + Shift + 2

Profiling during startup or shutdown

Source: https://profiler.firefox.com/docs

Looking at a real performance bug

Bug 1630681

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%."

Donald Knuth

Eminent computer scientist and mathematician, inventor of TeX and author of *The Art of Computer Programming*

Source: "Structured Programming with Goto statements" (1974)

Beyond frequency and input size

Response times are determined by human perceptual abilities

100 ms < t <= 1s: user's flow of thought uninterrupted, but user will notice a delay

Bad UX

Response time: 1s

Source: https://www.nngroup.com/articles/response-times-3-important-limits/

Beyond frequency and input size

Response times are determined by human perceptual abilities

t <= 100 ms: User feels the response is instantaneous

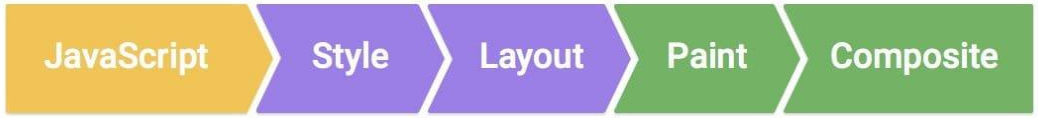
Good UX

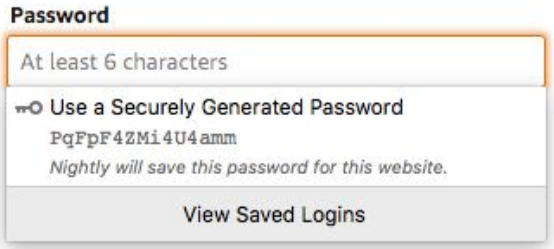
Response time: 25ms

Source: https://www.nngroup.com/articles/response-times-3-important-limits/

Real world performance bug: **before**

This forces a synchronous layout flush in the main thread of the parent process.





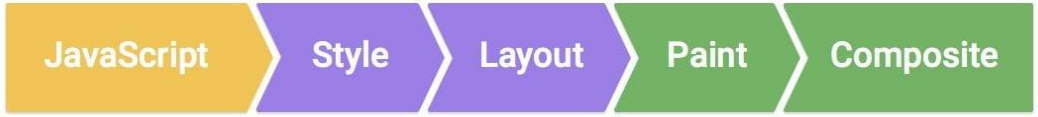

```
_adjustAcItem() {
  let outerBoxRect = this.parentNode.getBoundingClientRect();

  // Make item fit in popup as XUL box could not constrain
  // item's width
  this.firstElementChild.style.width = outerBoxRect.width + "px";
}
```

“Before” profile

Source: https://bugzilla.mozilla.org/show_bug.cgi?id=1630681

Real world performance bug: **after**

This forces a synchronous *style* flush in the main thread of the parent process.



Would `getBoundsWithoutFlushing` or `promiseDocumentFlushed` work?

Unfortunately not.

```
_adjustAcItem() {
  const popup = this.parentNode.parentNode;
  const minWidth = getComputedStyle(popup).minWidth.replace("px", "");
  // Make item fit in popup as XUL box could not constrain
  // item's width
  // popup.width is equal to the input field's width from the content process
  this.firstElementChild.style.width =
    Math.max(minWidth, popup.width) + "px";
}
```

“After” profile

Diff profile comparing “Before” and “After”

Source: https://bugzilla.mozilla.org/show_bug.cgi?id=1630681

TL;DR

- Performance matters! Some times more than others.
- Make files small, minimize file I/O and create fewer objects to reduce GCs.
- Optimize JS, especially running on the main thread in the parent process.
- Use async code where possible to keep the Gecko event loop running.
- Keep sync code under the 16 ms frame budget.
- Use the Profiler to find and fix bugs.

"The fastest code is code that isn't run at all."

Lots of people

moz://a

Thank You especially to **mconley** and **mythmon**

Additional resources:

- Other APIs/tools for measuring performance
- Anti-patterns and alternatives
- More helpful links

Other useful tools & techniques

(read only)

Other APIs/tools for measuring performance

TASK	API/TOOL
Firefox: Measure how long something takes across processes (starting in process A and ending in process B)	`Services.telemetry.msSystemNow()`
Firefox: Measure how long something takes within the same process without a `window`	`Cu.now()`, `ChromeUtils.addProfilerMarker` (helpfully, your marker will show up in the Marker Chart in the Profiler)
Firefox/Web: Measure how long something takes within the same process with a `window` or in a worker	`console.time`, `performance.now()`
Command line: Measure how long a script takes to run	`time` CLI E.g. `time node myScript.js` or `time python my_script.py`

Source: addProfileMarker

What is expensive?

ANTI-PATTERN	ALTERNATIVE
Main thread file I/O, especially at critical times like browser start-up	Store minimum needed data. Compress files (e.g. `Accept-Encoding: "gzip"`). Do I/O off the main thread (e.g. `PromiseWorker`)*. Read/write infrequently. Example: Bug 1621018
Doing work earlier than is necessary or that isn't necessary at all in some scenarios	Don't do the work! Do the work at browser idle times (e.g. `nsIdleService`). Lazy load modules (e.g. `ChromeUtils.defineModuleGetter`).
Churning (i.e. frequently creating and destroying objects) in visualizations, games, or other computationally intensive code	Keep object creation to a minimum. Re-use existing objects. Consider object pooling for complex objects.
Objects of unbounded size	Hardcode size limits. Use `WeakMap` and `WeakSet`.

* Thread proliferation is a problem in Firefox. On single core or dual core machines, threads aren't as cheap.

Source: https://mikeconley.ca/blog/2019/05/16/a-few-words-on-main-thread-disk-access-for-general-audiences/

What is expensive? continued...

ANTI-PATTERN	ALTERNATIVE
Sync code blocking the main thread in excess of the frame budget	Async code. Move code to a worker.* Run at browser idle. Profile. Microbenchmark (isolate and measure small chunks of code).
Moving large objects around especially across process	`nsInputStream` (e.g. Bug 1621032). Streaming is used e.g. for network packets and downloads.
Repeated and redundant expensive computations	Memoization (e.g. Bug 1595244, see `_cachedNewPasswordScore`)
Sync style or layout flushes	`promiseDocumentFlushed`, `windowUtils.getBoundsWithoutFlushing`, `requestAnimationFrame`

* Thread proliferation is a problem in Firefox. On single core or dual core machines, threads aren't as cheap.