Lecture 19 — Performance Case Studies

Patrick Lam
patrick.lam@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

October 22, 2019

Case Study: Firefox Quantum

Some Firefox Perf Improvements, per Mike Conley

- don’t animate out-of-view elements
- move db init off main thread
- keep better profiling data
- parallel painting for macOS
- lazily instantiate Search Service
- halve size of the blocklist
- refactor to reduce main-thread IO
- don’t hold all frames of animated GIFs/APNGs in memory
- eliminate unnecessary hash table
- use more modern compiler

`https://mikeconley.ca/blog/2018/02/14/firefox-performance-update-1/`

We've seen this before:

- do less work (or do it sooner/later);
- use threads (move work off main thread);
- measure performance;

Which of the updates fall into which categories?

Some Firefox Perf Improvements, per Mike Conley

- don’t animate out-of-view elements
- move db init off main thread
- keep better profiling data
- parallel painting for macOS
- lazily instantiate Search Service
- halve size of the blocklist
- refactor to reduce main-thread IO
- don’t hold all frames of animated GIFs/APNGs in memory
- eliminate unnecessary hash table
- use more modern compiler

How?

- do less work (or do it sooner/later);
- use threads (move work off main thread);
- measure performance;

Tab Warming

https://mikeconley.ca/blog/2018/01/11/making-tab-switching-faster-in-firefox-with-tab-warming/.

"Maybe this is my Canadian-ness showing, but I like to think of it almost like coming in from shoveling snow off of the driveway, and somebody inside has *already made hot chocolate for you*, because they knew youâĂŹd probably be cold." — Mike Conley

How Can Tab Warming Work?

Before: Firefox requests paint of newly-active tab, and then waits for the result before switching.

Idea: reduce user-visible latency by predicting an imminent tab switch.

Q: How can we predict the future?

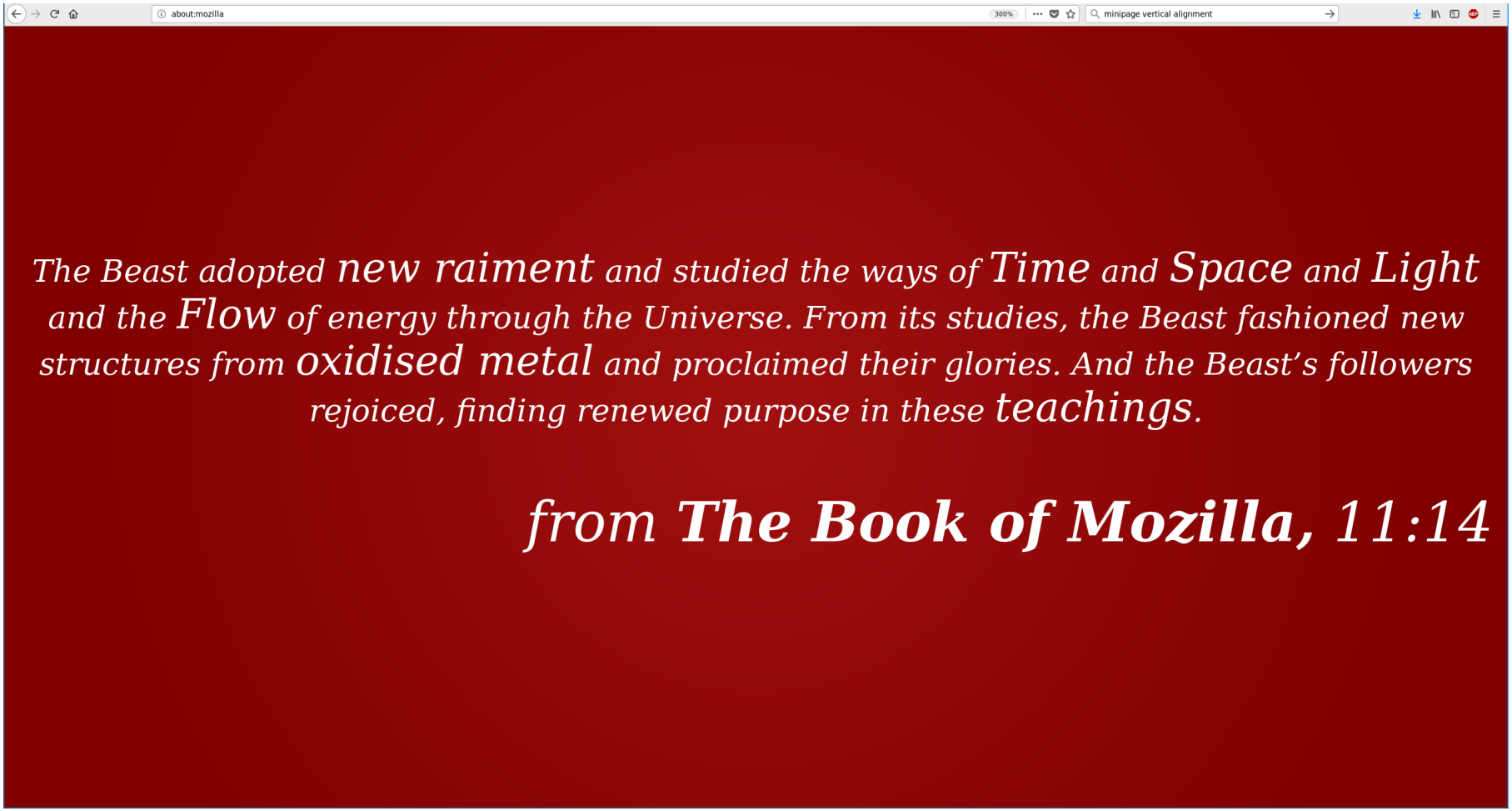
Q’: How can we predict which tab will be switched to?

Predicting the future

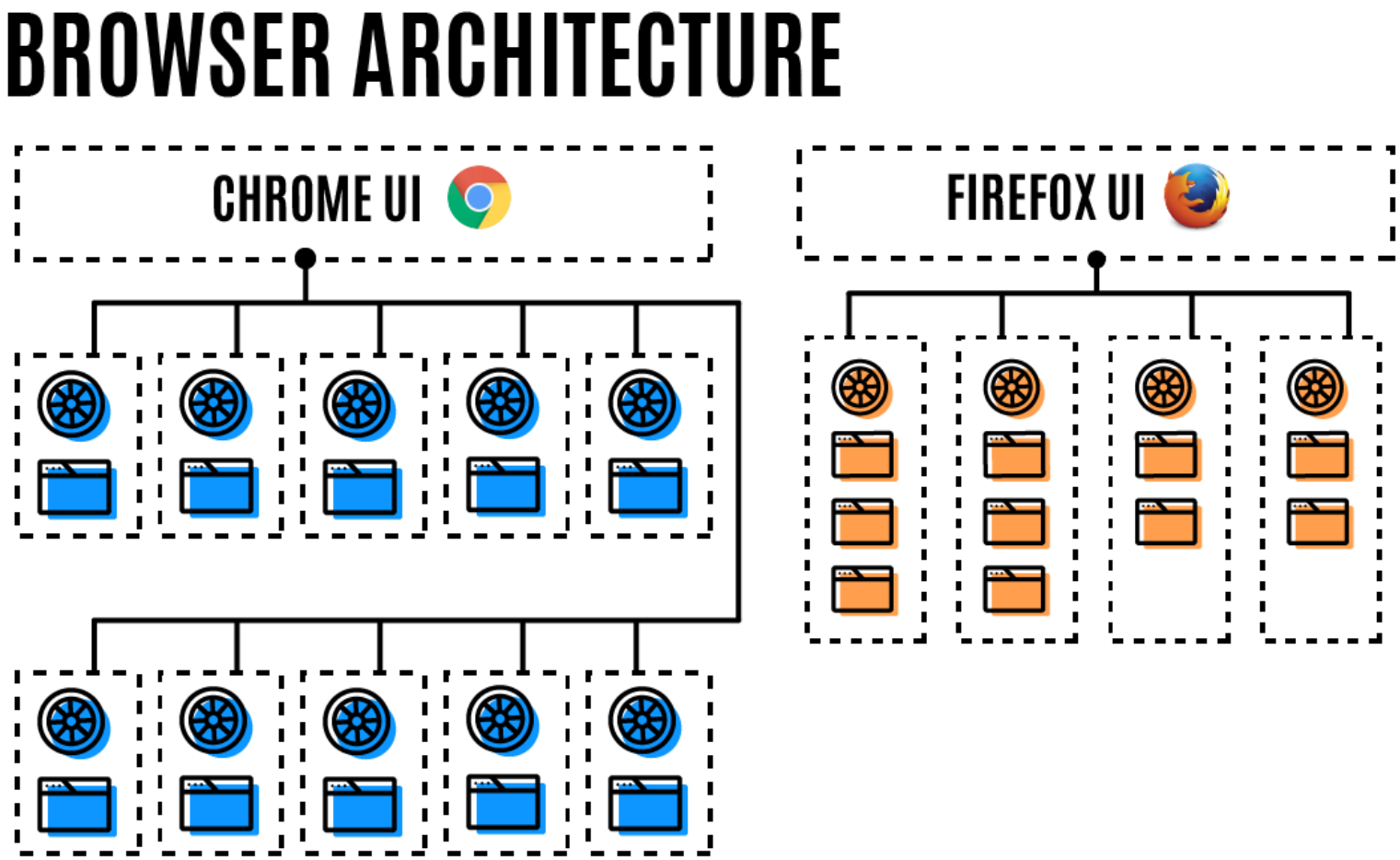
A: When the user has a mouse, then the mouse cursor will hover over the next tab.

> *Assuming a sufficiently long delay between hover and click, the tab switch should be perceived as instantaneous. If the delay was non-zero but still not long enough, we will have nonetheless shaved that time off in eventually presenting the tab to you.*
> *And in the event that we were wrong, and you werenâĂŹt interested in seeing the tab, we eventually throw the uploaded layers away.*

Blog post does not report performance numbers.


about:mozilla
300%
minipage vertical alignment
The Beast adopted new raiment and studied the ways of Time and Space and Light and the Flow of energy through the Universe. From its studies, the Beast fashioned new structures from oxidised metal and proclaimed their glories. And the Beast's followers rejoiced, finding renewed purpose in these teachings.
from The Book of Mozilla, 11:14

Electrolysis (2017): multiple OS-level processes.
(Think about threading models).



Chrome vs Firefox

Chrome: 1-process-per-tab.
Firefox: 4 shared content processes.

Firefox uses less memory (has less render state).

Electrolysis challenges:
internal architecture, and add-ons.

Electrolysis vs Flow

Two different Firefox projects:

Electrolysis = split across processes

Quantum Flow = leverage multithreading
(using Rust's "fearless concurrency"),
plus other improvements.

Quantum Flow

Steps:

1. Measure slowness & prioritize
2. Gather help
3. Fix all (well, some of) the things!

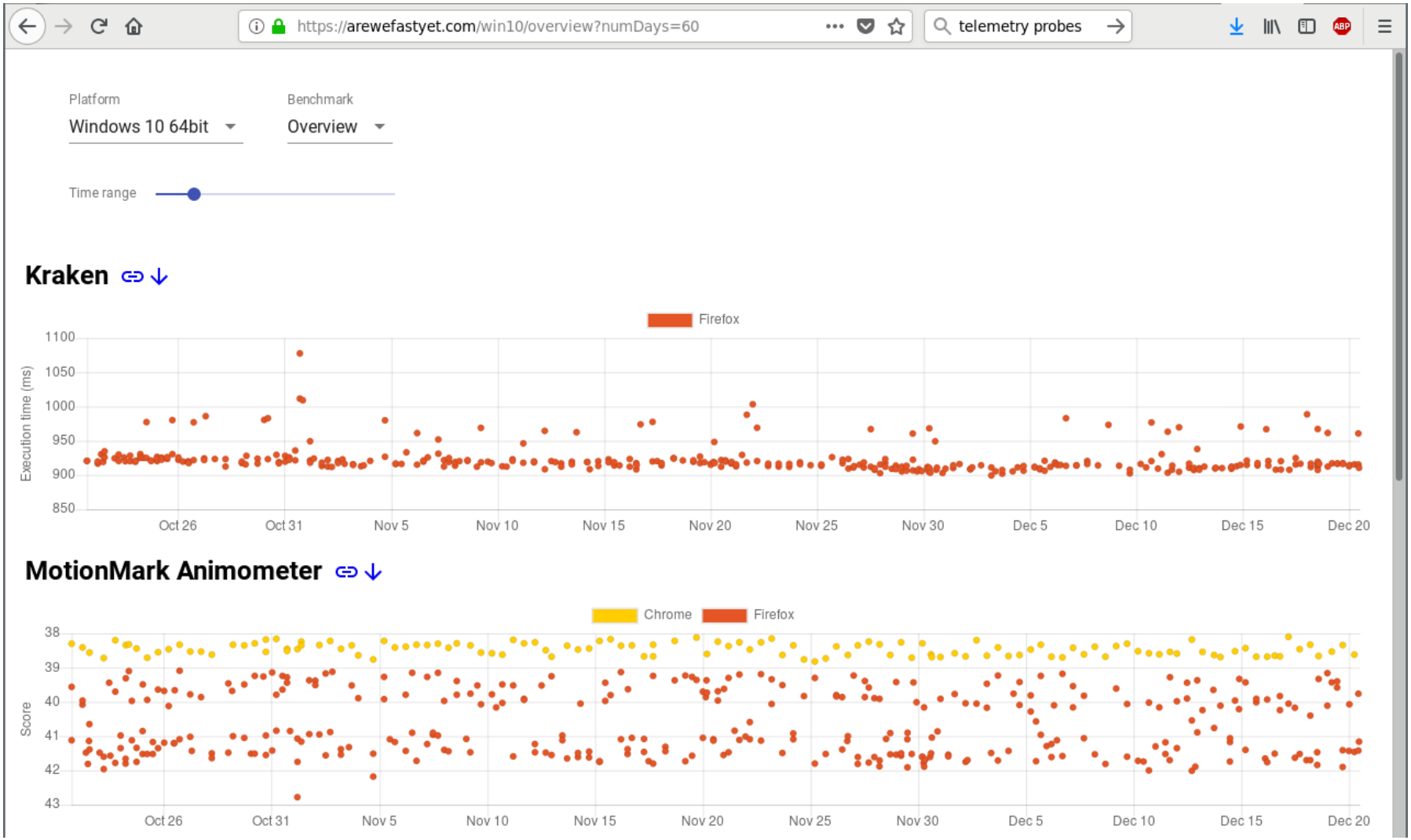
In 6 months:

prioritized 895 bugs, fixed 369.

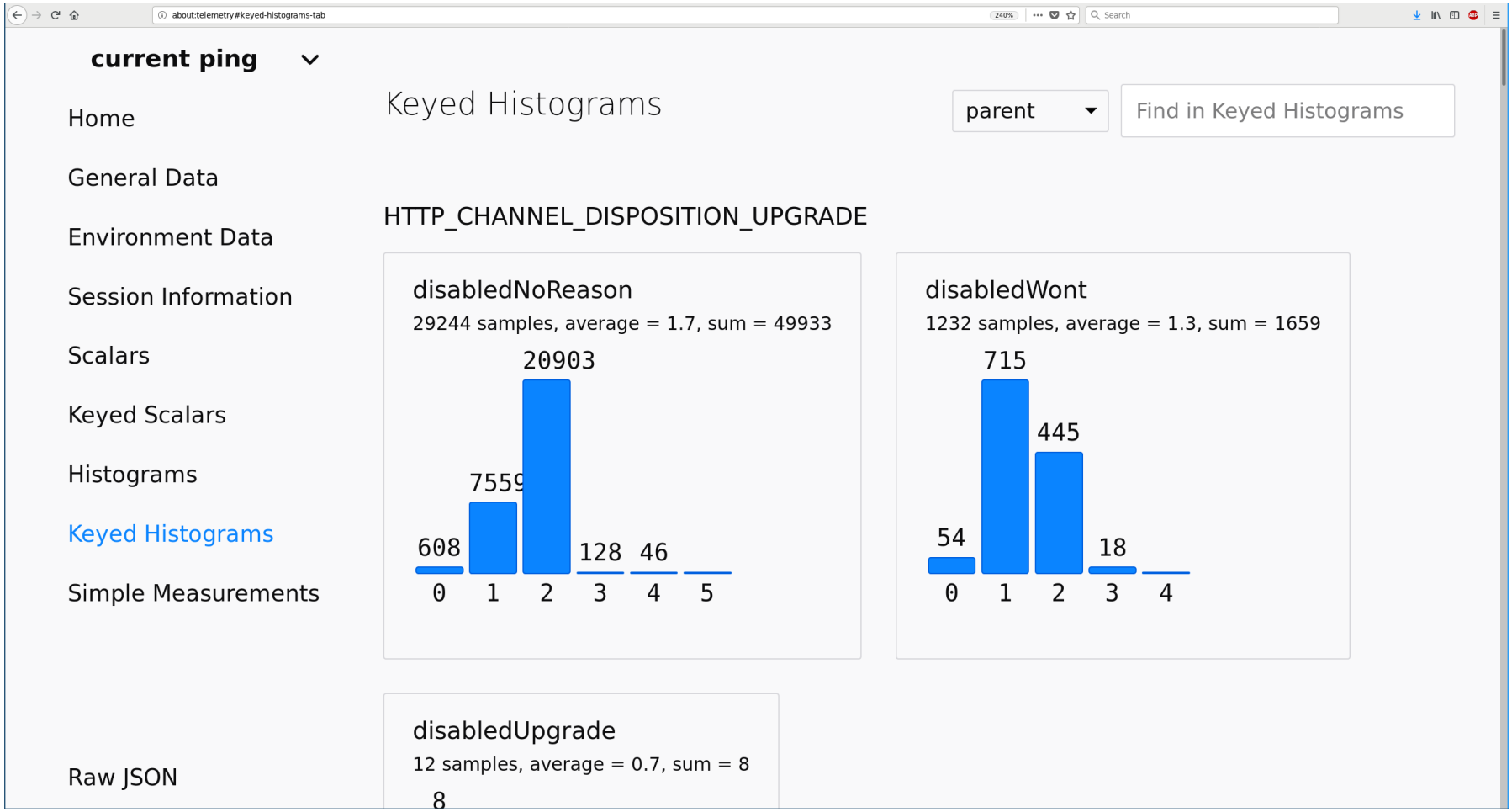
Key tool:

“Quantum Flow Engineering Newsletter”.

arewefastyet dot com



about:telemetry



About Telemetry

Idea: Ask questions first, act second.

Collect data about Firefox usage, then start hacking.

100s of GBs of anonymous metrics/day,
publicly available.

Analogous to CPU profiling, but massively distributed.

- collected much less often than CPU profiling data, but at much broader scope.

`https://telemetry.mozilla.org/`

Example Telemetry Questions

- Is Firefox the user's default browser? (69% yes)
- Does e10s make startup faster? (no, slower)
- Which plugins tend to freeze the browser on load? (Silverlight and Flash)

Can also see evolution of data over time.

Devs can propose new probes;
reviewed for data privacy plus normal code review.

ping!

Firefox sends pings:

- “main ping” every 24 hours;
- upon shutdown;
- upon environment change;
- upon abnormal shutdown.

Presumably compressed JSON to Mozilla servers.

common ping structure

```
{
  type: <string>, // "main", "activation", "optout", ...
  id: <UUID>, // a UUID that identifies this ping
  creationDate: <ISO date>, // the date the ping was generated
  version: <number>, // the version of the ping format

  application: {
    architecture: <string>, // build architecture, e.g. x86
    buildId: <string>, // "20141126041045"
    // etc
  },

  clientId: <UUID>, // optional
  environment: { ... }, // optional, not all pings contain
  payload: { ... }, // actual payload data for this ping type
}
```

Ping contents

1. Scalars (counts, booleans, strings)
2. Histograms = bucketed data (like grade distributions)

Both scalars and histograms can be keyed, e.g. how often searches happen for which search engines.

Single-Thread Performance

“Can you run faster just by trying harder?”

Single-Thread Performance

Performance improvements to date have used parallelism to improve throughput.

Decreasing latency is trickier—
often requires domain-specific tweaks.

Today: one example of decreasing latency:
Stream VByte.

I have a cunning plan...

Even Stream VByte uses parallelism:
vector instructions.

But there are sequential improvements,
e.g. Stream VByte takes care to be
predictable for the branch predictor.

Inverted Indexes (like it's CS 137 again!)

Abstractly: store a sequence of small integers.

Why Inverted indexes?

allow fast lookups by term;
support boolean queries combining terms.

Dogs, cats, cows, goats. In ur documents.

docid	terms
1	dog, cat, cow
2	cat
3	dog, goat
4	cow, cat, goat

Inverting the Index

Here's the index and the inverted index:

docid	terms
1	dog, cat, cow
2	cat
3	dog, goat
4	cow, cat, goat

term	docs
dog	1, 3
cat	1, 2, 4
cow	1, 4
goat	3, 4

Inverted indexes contain many small integers.

Deltas typically small if doc ids are sorted.

Storing inverted index lists: VByte

VByte uses a variable number of bytes to store integers.

Why? Most integers are small, especially on today's 64-bit processors.

How VByte Works

VByte works like this:

- x between 0 and $2^7 - 1$ (e.g. $17 = 0b10001$):
  $0xxxxxxx$, e.g. 00010001;
- x between 2^7 and $2^{14} - 1$ (e.g. $1729 = 0b11011000001$):
  $1xxxxxxx/0xxxxxxx$ (e.g. 11000001/00001101);
- x between 2^{14} and $2^{21} - 1$:
  $0xxxxxxx/1xxxxxxx/1xxxxxxx$;
- etc.

Control bit, or high-order bit, is:

0 once done representing the int,
1 if more bits remain.

Why VByte Helps

Isn't dealing with variable-byte integers harder?

- Yup!

But perf improves:

- We are using fewer bits!

We fit more information into RAM and cache, and can get higher throughput. (think inlining)

Storing and reading 0s isn't good use of resources.

However, a naive algorithm to decode VByte gives branch mispredicts.

Stream VByte

Stream VByte: a variant of VByte using SIMD.

Science is incremental.
Stream VByte builds on earlier work—
masked VByte, VARINT-GB, VARINT-G8IU.

Innovation in Stream VByte:
store the control and data streams separately.

Control Stream

Stream VByte's control stream uses two bits per integer to represent the size of the integer:

00	1 byte	10	3 bytes
01	2 bytes	11	4 bytes

Decoding Stream VByte

Per decode iteration:

reads 1 byte from the control stream,
and 16 bytes of data.

Lookup table on control stream byte: decide how many bytes it needs out of the 16 bytes it has read.

SIMD instructions:

shuffle the bits each into their own integers.

Unlike VByte,
Stream VByte uses all 8 bits of data bytes as data.

Stream VByte Example

Say control stream contains $0b1000\ 1100$.
Then the data stream contains the following sequence of integer sizes: $3, 1, 4, 1$.

Out of the 16 bytes read, this iteration uses 9 bytes;
$\Rightarrow$ it advances the data pointer by 9.

The SIMD "shuffle" instruction puts decoded integers from data stream at known positions in the 128-bit SIMD register.

Pad the first 3-byte integer with 1 byte, then the next 1-byte integer with 3 bytes, etc.

Stream VByte: Shuffling the Bits

Say the data input is:
`0xf823 e127 2524 9748 1b.. .... .... .....`

The 128-bit output is:
`0x00f8 23e1/0000 0027/2524 9748/0000/001b`
/s denote separation between outputs.

Shuffle mask is precomputed and read from an array.

The core of the implementation uses three SIMD instructions:

```
uint8_t C = lengthTable[control];
__m128i Data = _mm_loadu_si128 ((__m128i *) databytes);
__m128i Shuf = _mm_loadu_si128(shuffleTable[control]);
Data = _mm_shuffle_epi8(Data, Shuf);
databytes += C; control++;
```

But Does It Work?!

Stream VByte performs better than previous techniques on a realistic input.

Why?

- control bytes are sequential:
  CPU can always prefetch the next control byte, because its location is predictable;
- data bytes are sequential
  and loaded at high throughput;
- shuffling exploits the instruction set:
  takes 1 cycle;
- control-flow is regular
  (tight loop which retrieves/decodes control & data; no conditional jumps).