Optimizing zlib for

A **deflated** story

Adenilson Cavalcanti
BS. MSc.
Staff Engineer - **Arm** San Jose (CA)

THE CONFERENCE MORNING SESSION

DAY 1
7:00am

DAY 2
7:00am

DAY 3
7:00am

LAST DAY
7:00am

WWW.PHDCOMICS.COM

What to optimize in Chromium

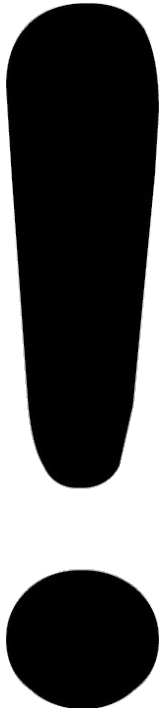
What to optimize in Chromium

- Too big.
- Too many areas.
- What would be helpful?

What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.



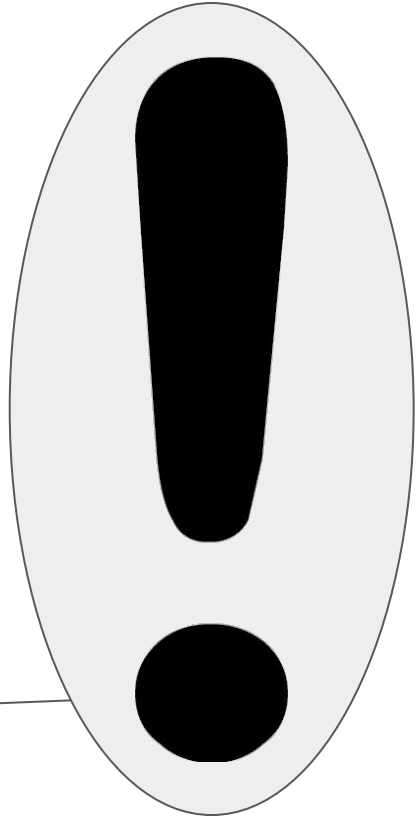
What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.

Text

Image



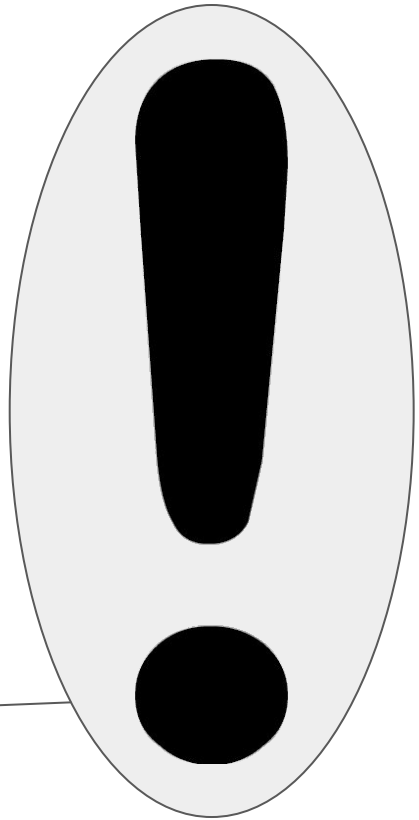
What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.

Text

Image



PNG

- Powerful format: Palette, pre-filters, compressed.
- Encoder affects behavior.
- Libpng and zlib are ‘Bros!’.

Meet Mr. Parrot

Source: https://upload.wikimedia.org/wikipedia/commons/3/3f/ZebraHighRes.png

Parrots are not created equal

Parrots are not created equal

Original: 2.7MB

Palette: 0.8MB

Zopfli: 2.6MB

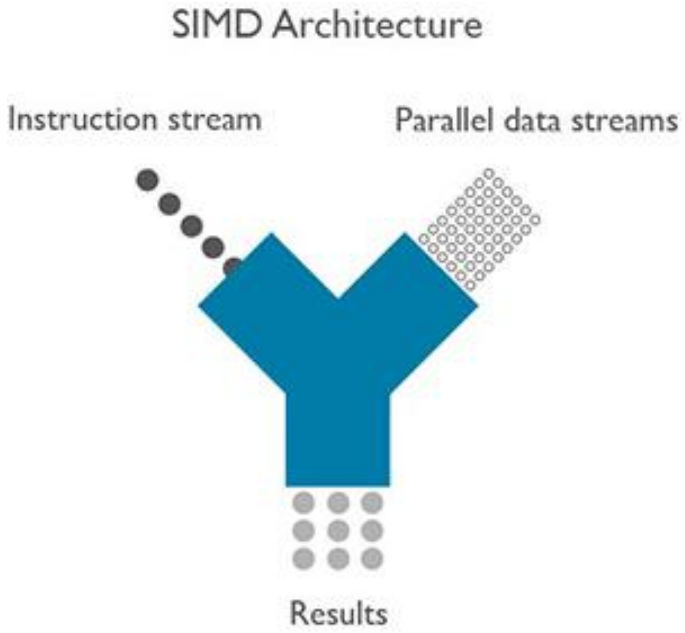
Features affect hotspots

```
== Image has pre-compression filters (2.7MB) ==
Lib     Command     SharedObj          method                          CPU (%)
zlib    TileWorker  liblink           inflate_fast ..................... 1.96
zlib    TileWorker  libblnk           adler32 .......................... 0.88
blink   TileWorker  liblink           ImageFrame::setRGBAPremultiply .. 0.45
blink   TileWorker  liblink           png_read_filter_row_up........... 0.03*

== Image was optimized using zopfli (2.6MB) ==
Lib     Command     SharedObj          method                          CPU (%)
zlib    TileWorker  liblink           inflate_fast ..................... 3.06
zlib    TileWorker  libblnk           adler32 .......................... 1.36
blink   TileWorker  liblink           ImageFrame::setRGBAPremultiply .. 0.70
blink   TileWorker  liblink           png_read_filter_row_up........... 0.48*


== Image has no pre-compression filters (0.9MB) ==
Lib     Command     SharedObj          method                          CPU (%)
libpng  TileWorker  liblink           cr_png_do_expand_palette ........ 0.88
zlib    TileWorker  liblink           inflate_fast ..................... 0.62
blink   TileWorker  liblink           ImageFrame::setRGBAPremultiply .. 0.49
zlib    TileWorker  libblnk           adler32 .......................... 0.31
```

NEON: Advanced SIMD
(Single Instruction Multiple Data)



- **Optional** in Armv7.
- Mandatory in Armv8.

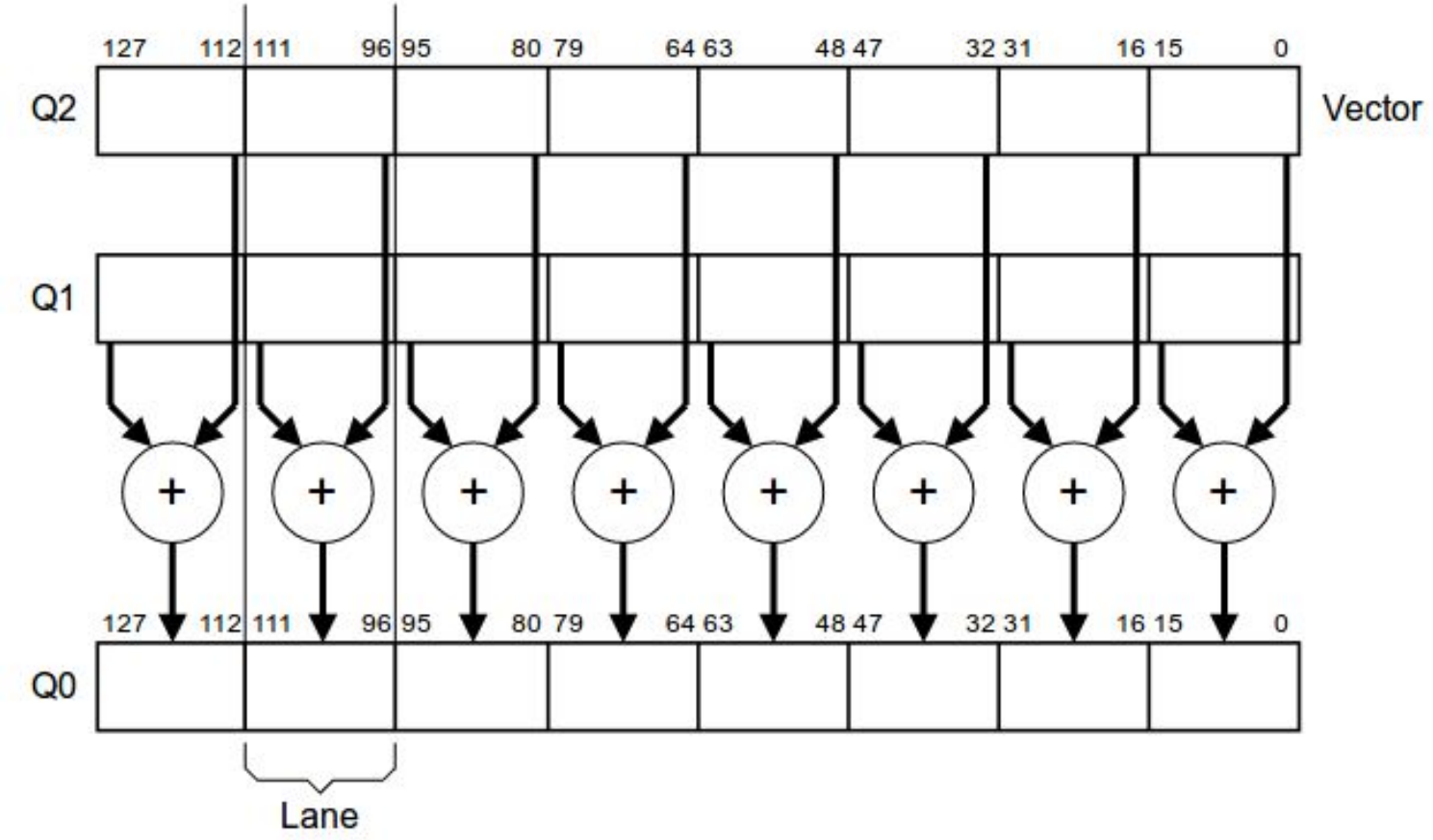
Registers@Armv7

- 16 registers@128 bits: Q0 - Q15.
- 32 registers@64bits: D0 - D31.
- Varied set of instructions: load, store, add, mul, etc.

Registers@Armv8 (SIMD&FP, V0 - V31)

- 32 registers@128 bits: Q0 - V31.
- 32 registers@64bits: D0 - D31.
- 32 registers@32bits: S0 - S31.
- 32 registers@8bits: H0 - H31.
- Varied set of instructions: load, store, add, mul, etc.

An example: VADD.I16 Q0, Q1, Q2



Candidates

- Inflate_fast: **zlib**.
- Adler32: **zlib**.
- ImageFrame: Blink.
- png_do_expand_palette: libpng.

Why zlib?

Zlib

Used everywhere (libpng, Skia, freetype, **cronet**, blink, chrome, linux kernel, etc).

Old code base released in 1995.

Written in K&R C style.

Context

Lacks any optimizations for ARM CPUs.

Problem statement

Identify potential optimization candidates and verify positive effects in Chromium.

Potential problems

- Viability of optimization.
- Positive effects.
- **Upstreaming**.

Implementation

Adler-32

$$A = 1 + D_1 + D_2 + \ldots + D_n \pmod{65521}$$

$$B = (1 + D_1) + (1 + D_1 + D_2) + \ldots + (1 + D_1 + D_2 + \ldots + D_n) \pmod{65521}$$

$$= n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \ldots + D_n + n \pmod{65521}$$

$$Adler\text{-}32(D) = B \times 65536 + A$$

https://en.wikipedia.org/wiki/Adler-32

Adler-32: simplistic implementation

```
// From: https://en.wikipedia.org/wiki/Adler-32
const int MOD_ADLER = 65521;
unsigned long naive_adler32(unsigned char *data,
                            unsigned long len)
{
    uint32_t a = 1, b = 0;
    unsigned long index;

    for (index = 0; index < len; ++index) {
        a = (a + data[index]) % MOD_ADLER;
        b = (b + a) % MOD_ADLER;
    }

    return (b << 16) | a;
}
```

Problems

- Zlib's Adler-32 was more than **7x faster** than naive implementation.
- It is hard to vectorize the following computation:

```
void accum(uint32_t *pair, const unsigned char *buf,
           unsigned int len)
{
    unsigned int i;
    for (i = 0; i < len; ++i) {
        pair[0] += buf[i];
        pair[1] += pair[0];
    }
}
```

Problems: how to represent pair[1] or ‘B’?

```
void accum(uint32_t *pair, const unsigned char *buf,
           unsigned int len)
{
    unsigned int i;
    for (i = 0; i < len; ++i) {
        pair[0] += buf[i];
        pair[1] += pair[0];
    }
}
```

Problems: how to represent pair[1] or ‘B’?

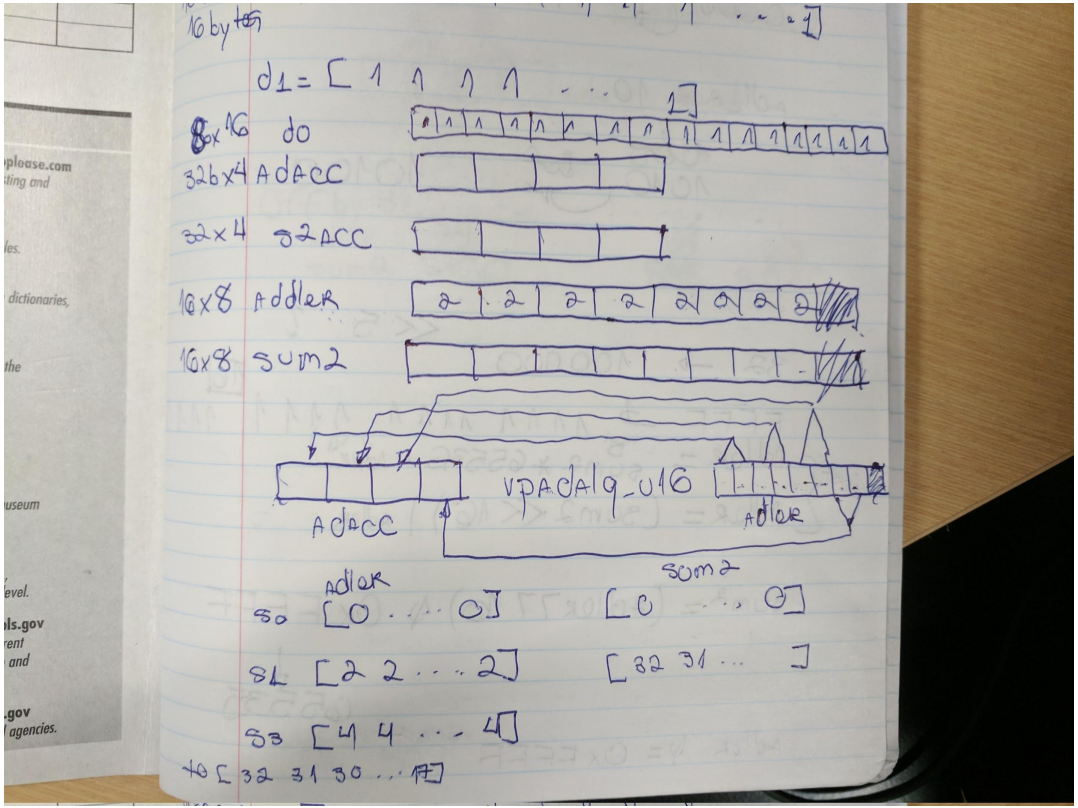
$$A = 1 + D_1 + D_2 + \ldots + D_n \pmod{65521}$$

$$B = (1 + D_1) + (1 + D_1 + D_2) + \ldots + (1 + D_1 + D_2 + \ldots + D_n) \pmod{65521}$$

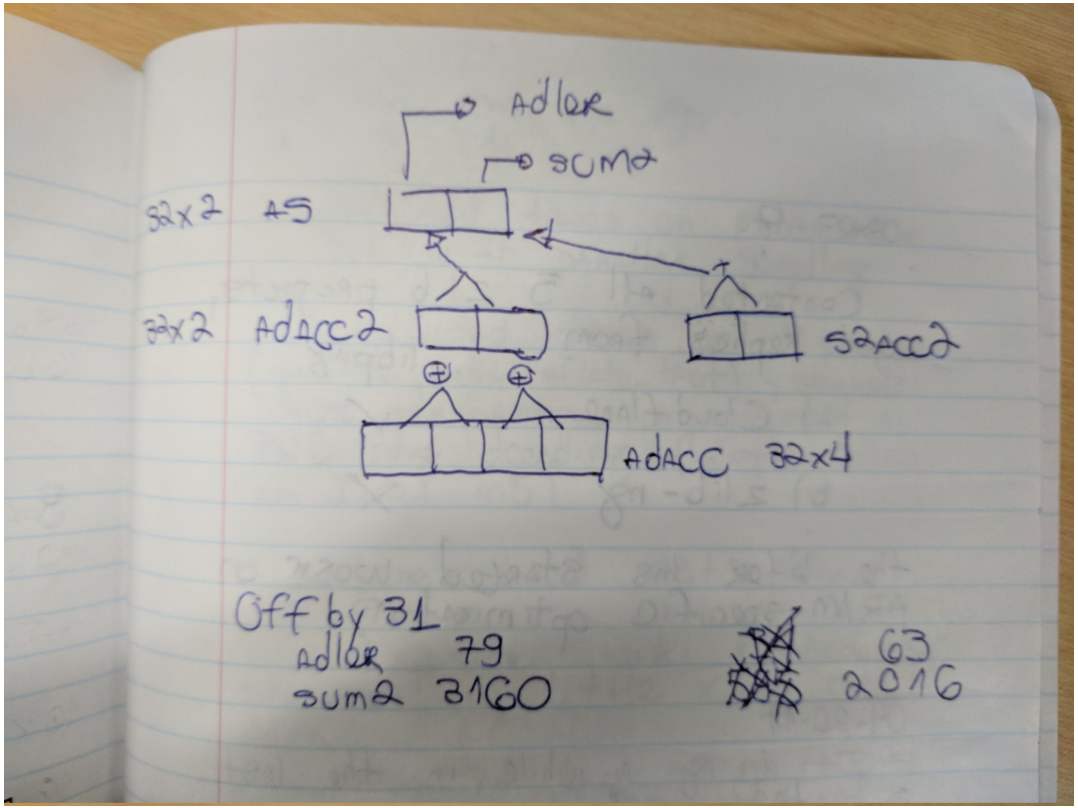
$$= n \times D_1 + (n-1) \times D_2 + (n-2) \times D_3 + \ldots + D_n + n \pmod{65521}$$

$$Adler\text{-}32(D) = B \times 65536 + A$$

Highly technical drawing (Jan 2017)



Highly technical drawing (Jan 2017)



‘Taps’ to the rescue

```
static void NEON_accum32(uint32_t *s, const unsigned char *buf,
                         unsigned int len)
{
    static const uint8_t taps[32] = {
        32, 31, 30, 29, 28, 27, 26, 25,
        24, 23, 22, 21, 20, 19, 18, 17,
        16, 15, 14, 13, 12, 11, 10, 9,
        8, 7, 6, 5, 4, 3, 2, 1 };

    uint32x2_t adacc2, s2acc2, as;
    uint8x16_t t0 = vld1q_u8(taps), t1 = vld1q_u8(taps + 16);

    uint32x4_t adacc = vdupq_n_u32(0), s2acc = vdupq_n_u32(0);
    adacc = vsetq_lane_u32(s[0], adacc, 0);
    s2acc = vsetq_lane_u32(s[1], s2acc, 0);

    while (len >= 2) {
        uint8x16_t d0 = vld1q_u8(buf), d1 = vld1q_u8(buf + 16);
        uint16x8_t adler, sum2;
        s2acc = vaddq_u32(s2acc, vshlq_n_u32(adacc, 5));
        adler = vpaddlq_u8(       d0);
        adler = vpadalq_u8(adler, d1);
        sum2 = vmull_u8(      vget_low_u8(t0), vget_low_u8(d0));
        sum2 = vmlal_u8(sum2, vget_high_u8(t0), vget_high_u8(d0));
        sum2 = vmlal_u8(sum2, vget_low_u8(t1), vget_low_u8(d1));
        sum2 = vmlal_u8(sum2, vget_high_u8(t1), vget_high_u8(d1));
        adacc = vpadalq_u16(adacc, adler);
        s2acc = vpadalq_u16(s2acc, sum2);
        len -= 2;
        buf += 32;
    }
```

Assembly:
https://godbolt.org/g/KMeBAJ

Happy end! Up to 18% performance gain in PNG

```
commit d400b38e450a71e9ed75bd4c1dda6329267e9ce0
Author: Adenilson Cavalcanti <adenilson.cavalcanti@arm.com>
Date:   Mon Jan 30 15:30:38 2017 -0800

    NEON implementation for Adler32

    The checksum is calculated in the uncompressed PNG data and can be made
    much faster by using SIMD.

    Tests in ARMv8 yielded an improvement of about 3x (e.g. walltime was
    350ms x 125ms for a 4096x4096 bytes executed 30 times).

    This alone yields a performance boost for PNG decoding ranging
    from 5% to 18% depending on a few factors (SoC, battery status,
    big/little, etc).
```

https://bugs.chromium.org/p/chromium/issues/detail?id=688601

Inffast (Simon Hosie)

- Second candidate in the perf profiling was **inflate_fast**.
- Very **high level** idea: perform long loads/stores in the byte array.
- Major gains: up to **30% faster**!

```
+                        */
+                       out = chunkcopy_safe(out, from, len, limit);
                    }
                }
                else {
-                   from = out - dist;          /* copy direct from output */
-                   do {                        /* minimum length is three */
-                       *out++ = *from++;
-                       *out++ = *from++;
-                       *out++ = *from++;
-                       len -= 3;
-                   } while (len > 2);
-                   if (len) {
-                       *out++ = *from++;
-                       if (len > 1)
-                           *out++ = *from++;
-                   }
+                   /* Whole reference is in range of current output.  No
+                      range checks are necessary because we start with room
+                      for at least 258 bytes of output, so unroll and roundoff
+                      operations can write beyond `out+len` so long as they
+                      stay within 258 bytes of `out`.
+                    */
+                   out = chunkcopy_lapped_relaxed(out, dist, len);
                }
```

https://bugs.chromium.org/p/chromium/issues/detail?id=697280

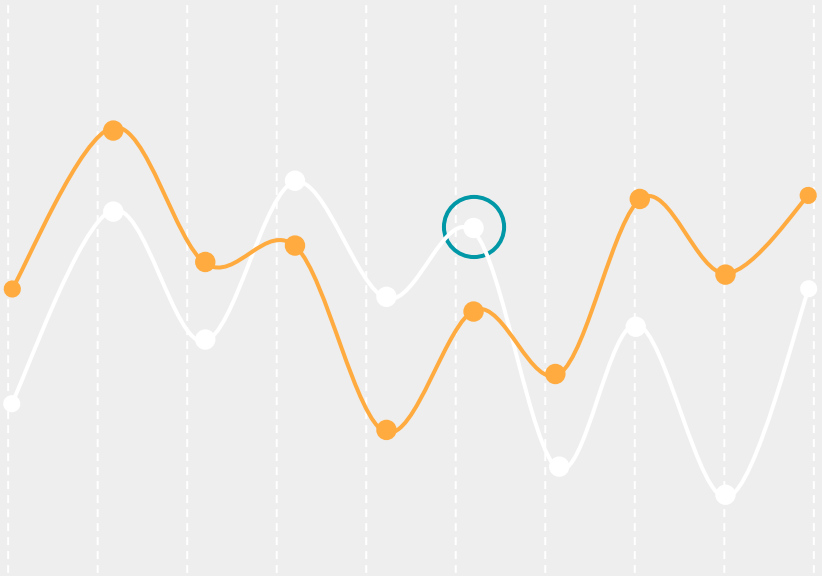
Libpng (Richard Townsend)

- NEON optimization in libpng.
- From 10 to 30% improvement.
- Depends on png using a palette.

https://bugs.chromium.org/p/chromium/issues/detail?id=706134

Impact

Combined effect of 3 patches



Chrome trace: vanilla Nexus6@2014 (116ms)

106 items selected. Slices (106)

Name ▽	Wall Duration ▽	CPU Duration ▽	Self time ▽	CPU Self Time ▼	Average CPU Duration ▽	Occurrences ▽
ImageFrameGenerator::decode	129.200 ms	116.187 ms	129.200 ms	116.187 ms	58.094 ms	2
GpuImageDecodeCache::DecodeImage	297.196 ms	234.969 ms	144.658 ms	106.867 ms	117.485 ms	2

Chrome trace: patched (73ms) 1.6x improvement

116 items selected. Slices (116)

Name	Wall Duration	CPU Duration	Self time	CPU Self Time	Average CPU Duration	Occurrences
GpuImageDecodeCache::DecodeImage	198.986 ms	179.268 ms	110.111 ms	94.025 ms	89.634 ms	2
ImageFrameGenerator::decode	77.021 ms	73.844 ms	77.021 ms	73.844 ms	36.922 ms	2

Comparing Arm x Intel

Source: https://commons.wikimedia.org/wiki/File:Apple_and_Orange_-_they_do_not_compare.jpg

Keeping in mind

- Snapdragon™ 805 @2014.
- 2.7Ghz Krait™ 450.
- 2MB L2 cache
- **28nm lithography.**
- Cellphone.
- EAS kernel.

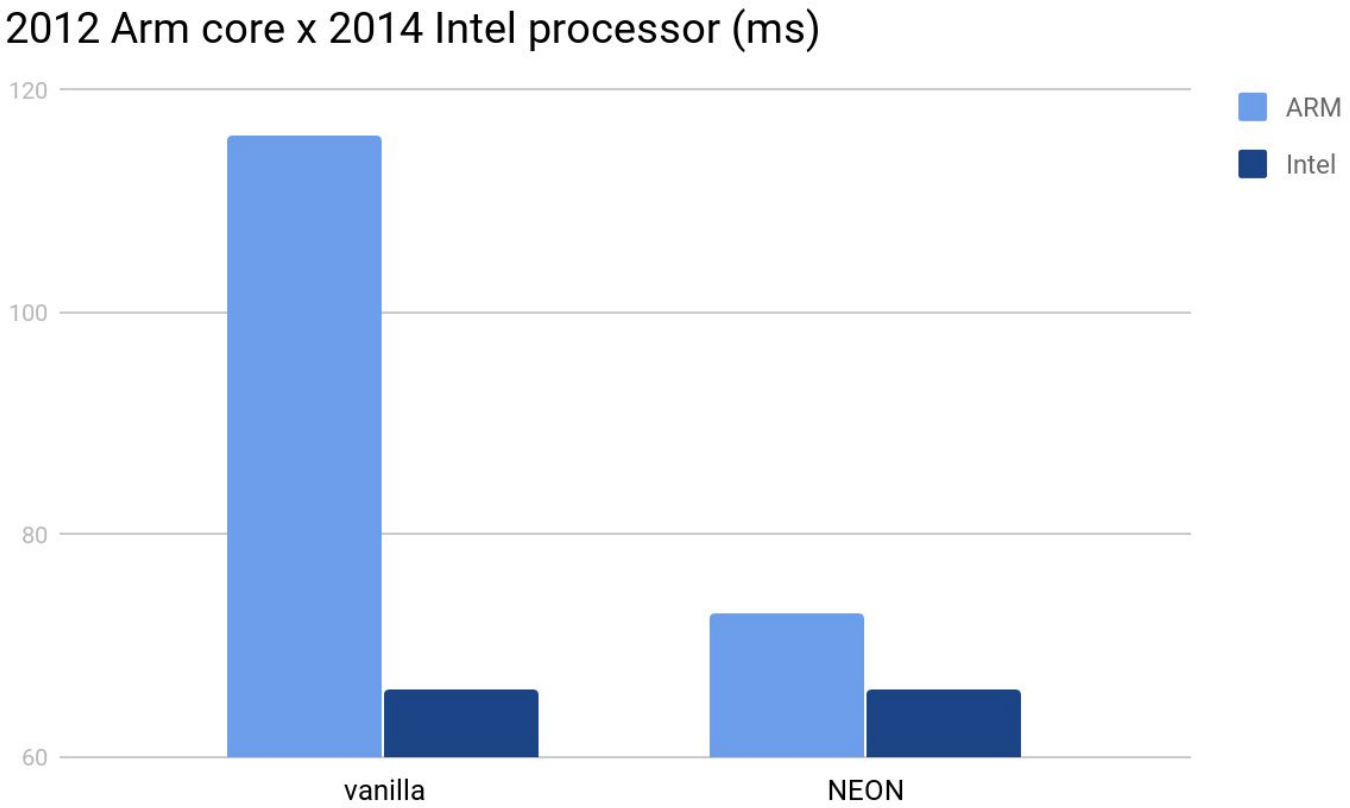
- 5Y10C launched @2015.
- 2Ghz Intel m5.
- 4MB cache.
- **14nm lithography.**
- Ultrabook.
- Regular linux kernel.

Chrome trace: Intel m5@2016 (66ms)

886 items selected. | Slices (654) | Thread Timeslices (232)

Name ▽	Wall Duration ▽	CPU Duration ▽	Self time ▽	CPU Self Time ▼	Average CPU Duration ▽	Occurrences ▽
SoftwareImageDecodeCache::DecodeImage	243.453 ms	239.738 ms	160.883 ms	158.763 ms	119.869 ms	2
ImageFrameGenerator::decode	67.785 ms	66.191 ms	67.785 ms	66.191 ms	33.096 ms	2

Effect of NEON optimization in Zlib



Lessons learned

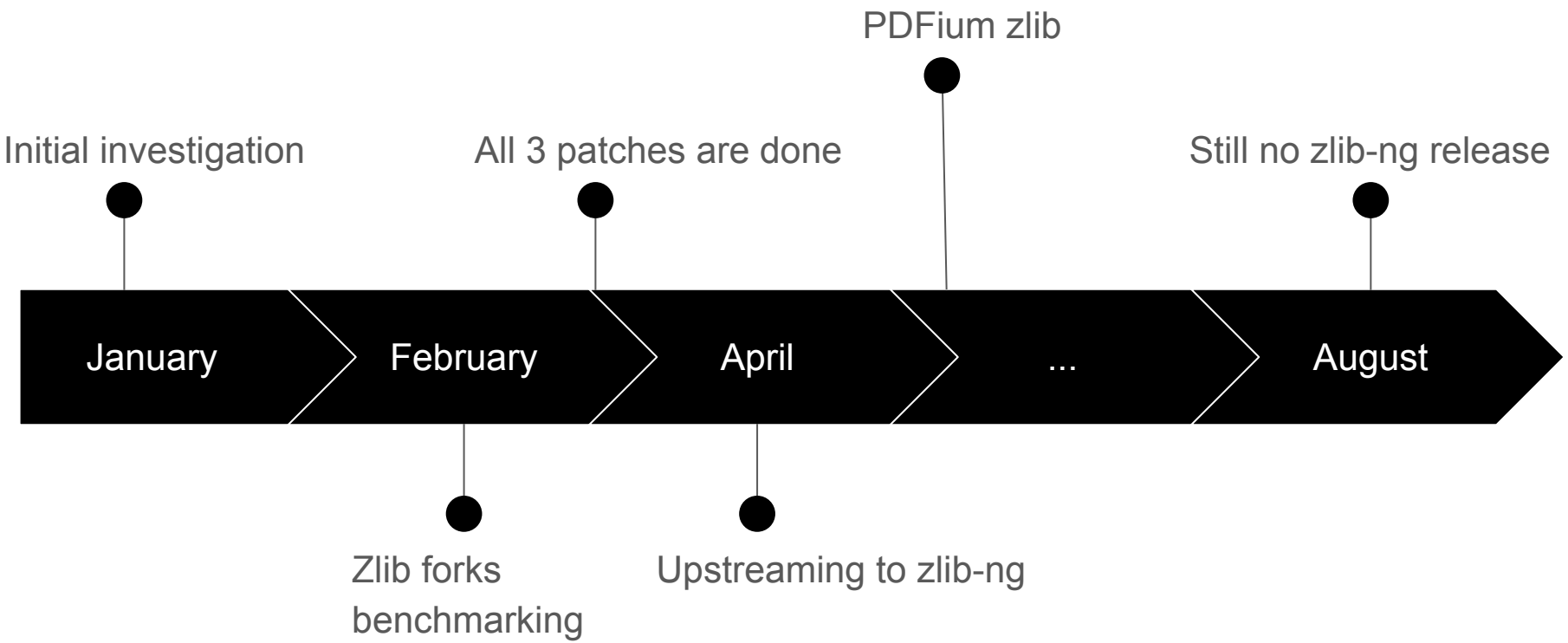
- **arm** cores can benefit **a lot** from NEON optimizations.
- Performance gains of 2 generations of silicon.
- It pays off to work in a lower software layer (e.g. zlib/libpng).

Happy end? Not yet...

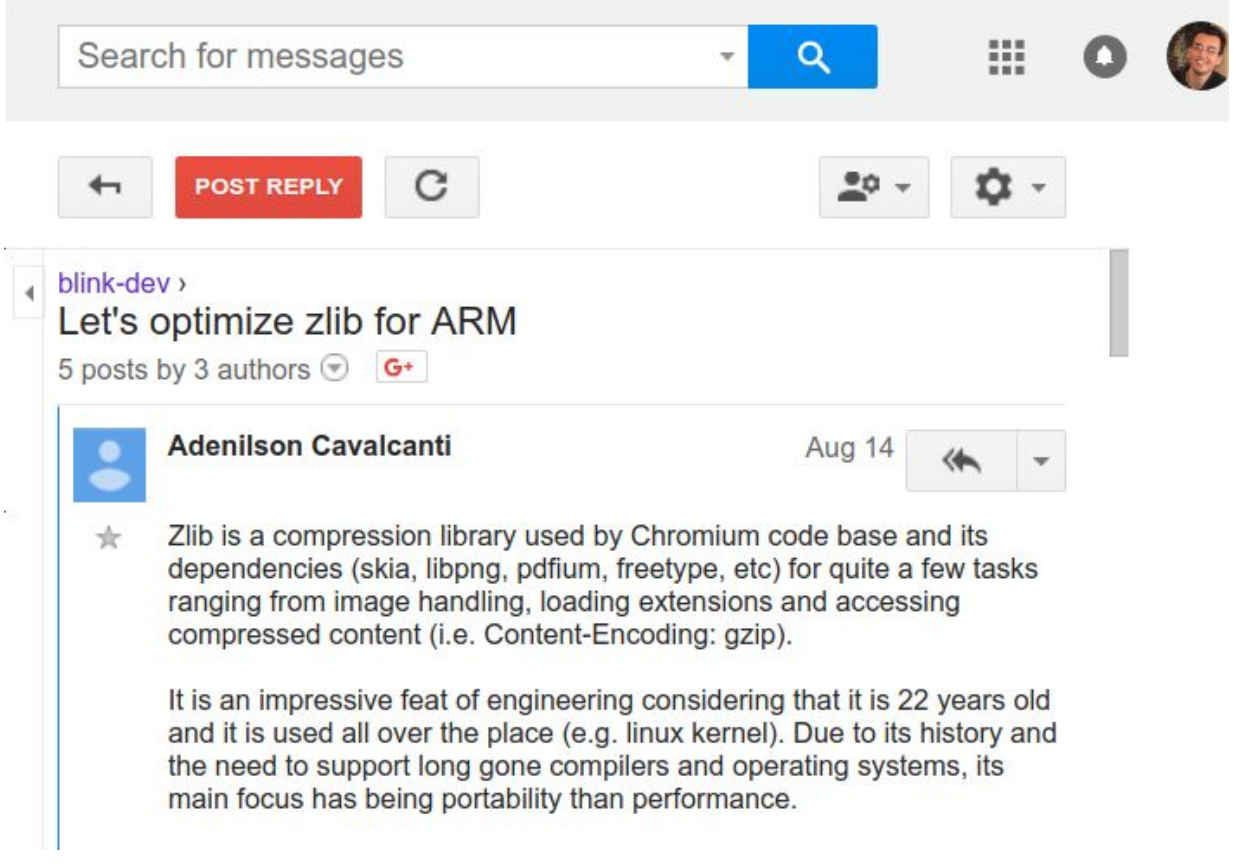
- Requested to perform a study comparing zlibs forks.
- Upstream ARM optimizations.
- Move Chromium to a new/better maintained zlib.

Happy end? Not yet...

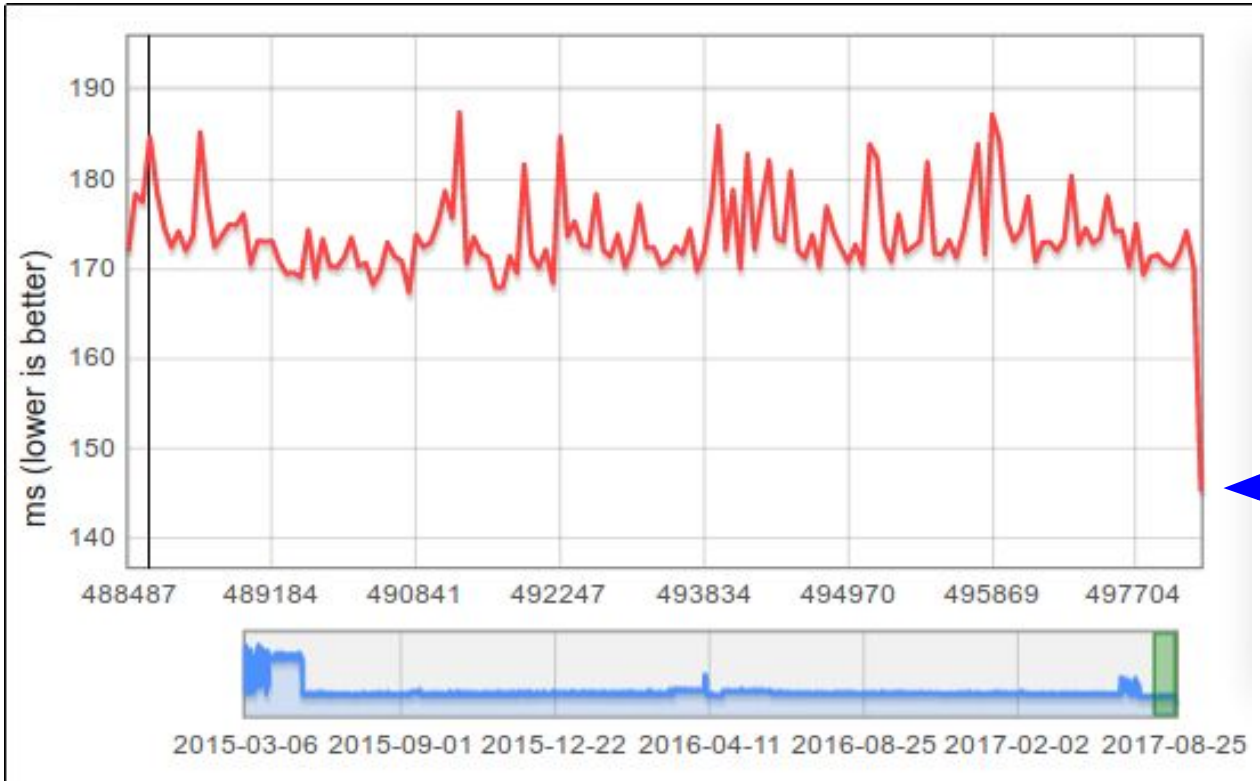
- Requested to perform a study comparing zlibs forks. Done!
  - https://goo.gl/ZUoy96
- Upstream ARM optimizations. Done!
  - https://github.com/Dead2/zlib-ng/commit/ec02ecf104e1d3f1836a908a359f20aa93494df5
- Move Chromium to a new/actively maintained zlib.
  - Upgraded/moved PDFium to Chromium's zlib.
  - Zlib-ng **didn't release** a stable release.


PDFium zlib
Initial investigation
All 3 patches are done
Still no zlib-ng release
January
February
April
...
August
Zlib forks benchmarking
Upstreaming to zlib-ng

Change of strategy



NEON inffast: featured in M62



https://bugs.chromium.org/p/chromium/issues/detail?id=697280

cronet: NEON != ARMv6

Source: https://xkcd.com/1172/

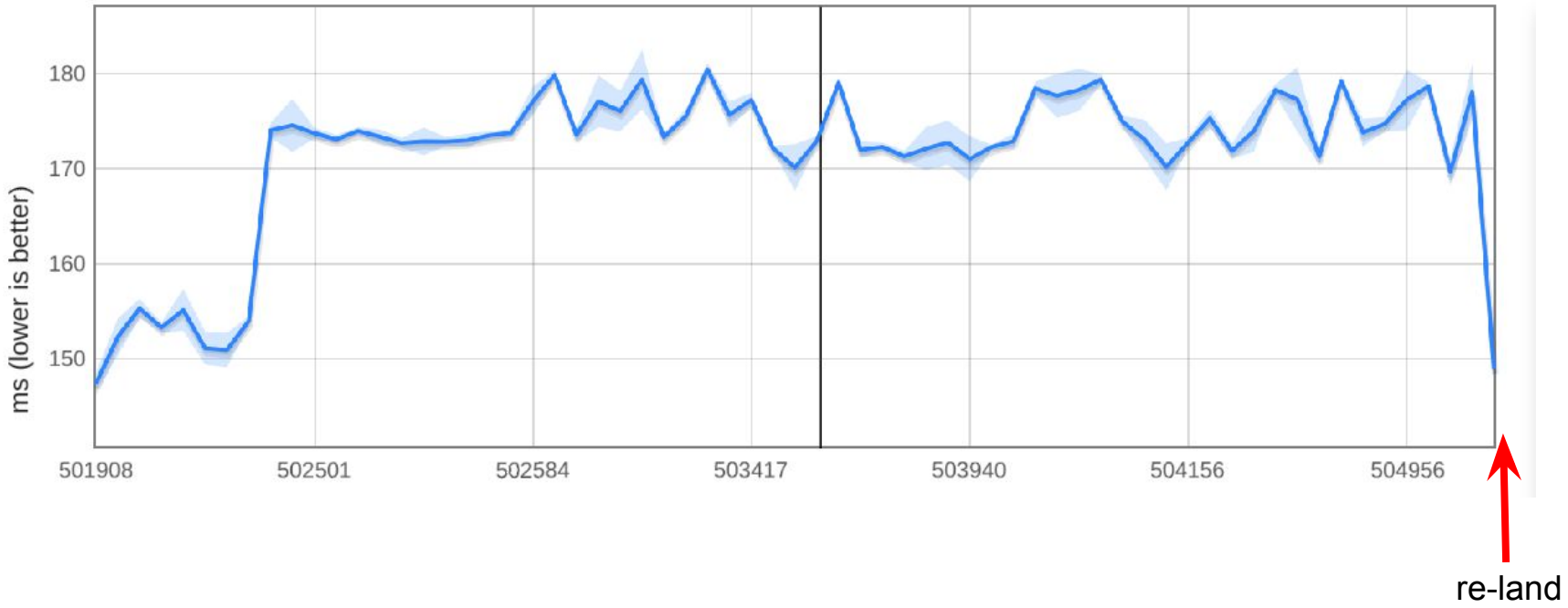
After re-landing… An internal app was broken.

Source: https://xkcd.com/1172/

Second revert (i.e. revert-revert-revert)

Misha Efimov@Google found the bug in the Java app client last Wednesday (Sep 27th).

Re-re-landed on Thur 28th

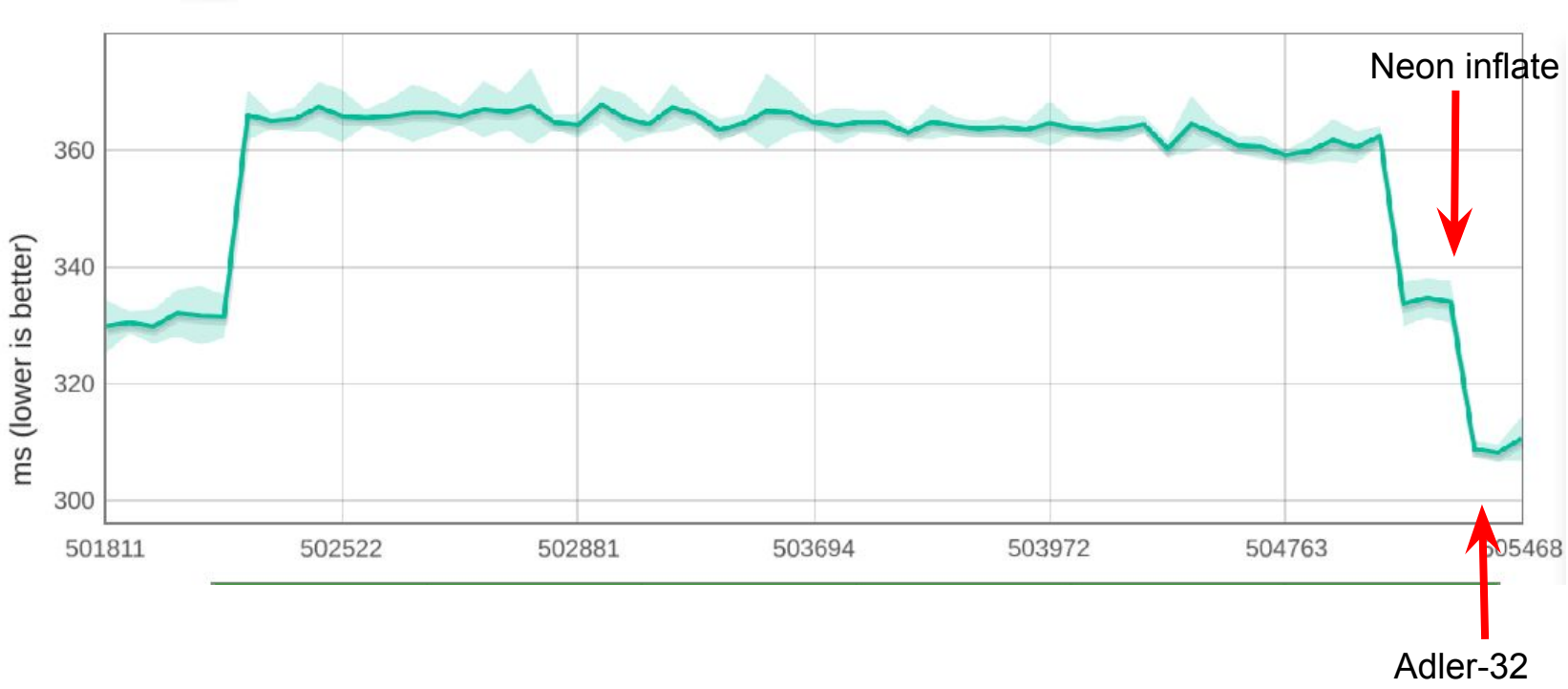


What comes next

- Land Adler-32 optimization* (Noel Gordon@Google implemented the same algorithm for Intel).
- Land the libpng optimization.
- CRC32: Armv8 instruction is about 10x faster.
- Compression comes next.

*Just landed last Friday:
https://chromium-review.googlesource.com/c/chromium/src/+/660019

Adler-32 landed on Fri 29th



https://goo.gl/RTgkGe

What comes next

- Land the libpng optimization.
- CRC32: ARMv8 instruction is about 10x faster.
- Fix infback corner case.
- Compression comes next.

Zlib users should consider migrating to Chromium’s zlib.

Special Thanks

- Igalia for the invite (Xabier Rodriguez Calvar).
- **Arm** for sponsoring the trip.
- Chris Blume@Google.
- Team **Arm@UK**: Dave Rodgman, Matteo Franchin, Richard Townsend, Stephen Kyle.
- Team **Arm@US**: Amaury Leleyzour, Simon Hosie.
- Compiler explorer: https://godbolt.org

Questions?