Optimizing Chromium image decoding for arm

An **inflated** story

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

Optimizing Chromium image decoding for arm

Or 3 gens of silicon thanks to NEON

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

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.

!

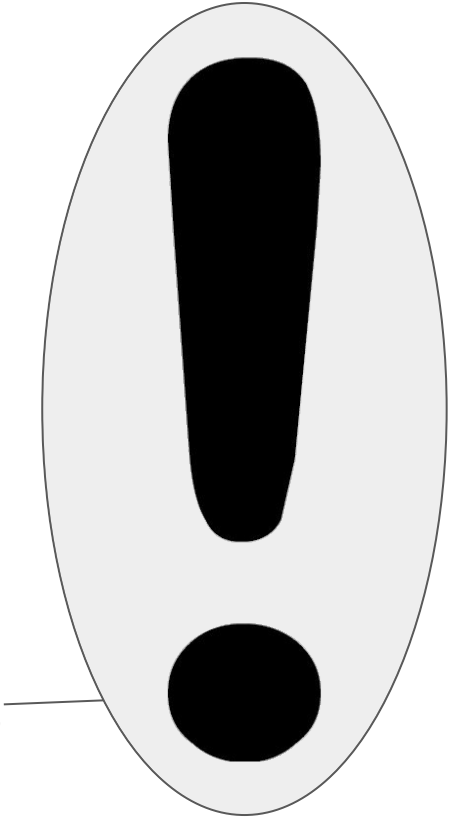
What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.

Text

Image

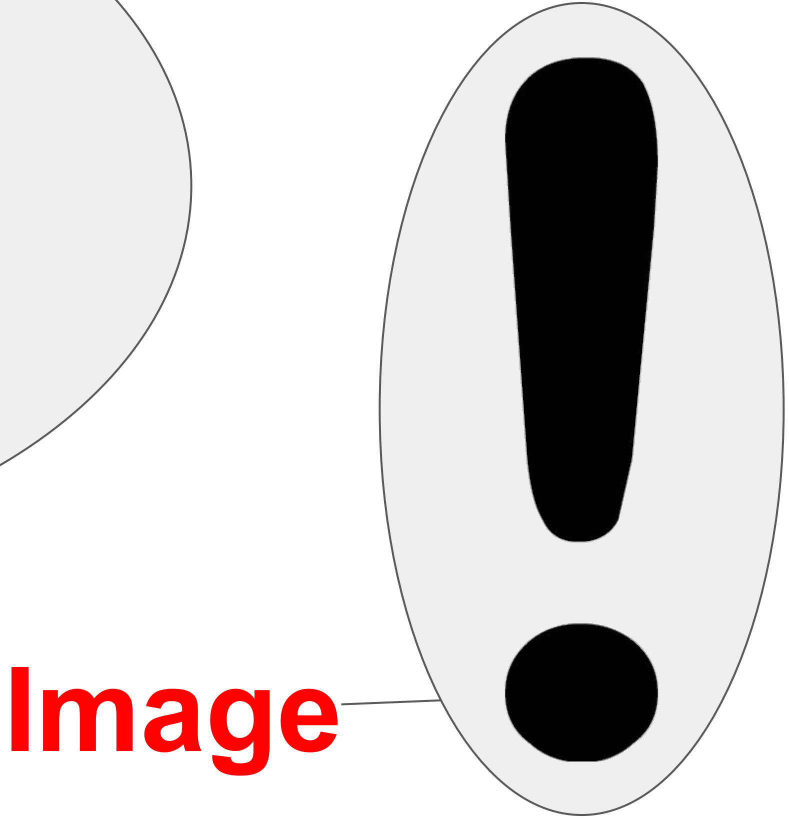


What to optimize in Chromium

Bulk of content still is:

- Text.
- Images.

Text



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

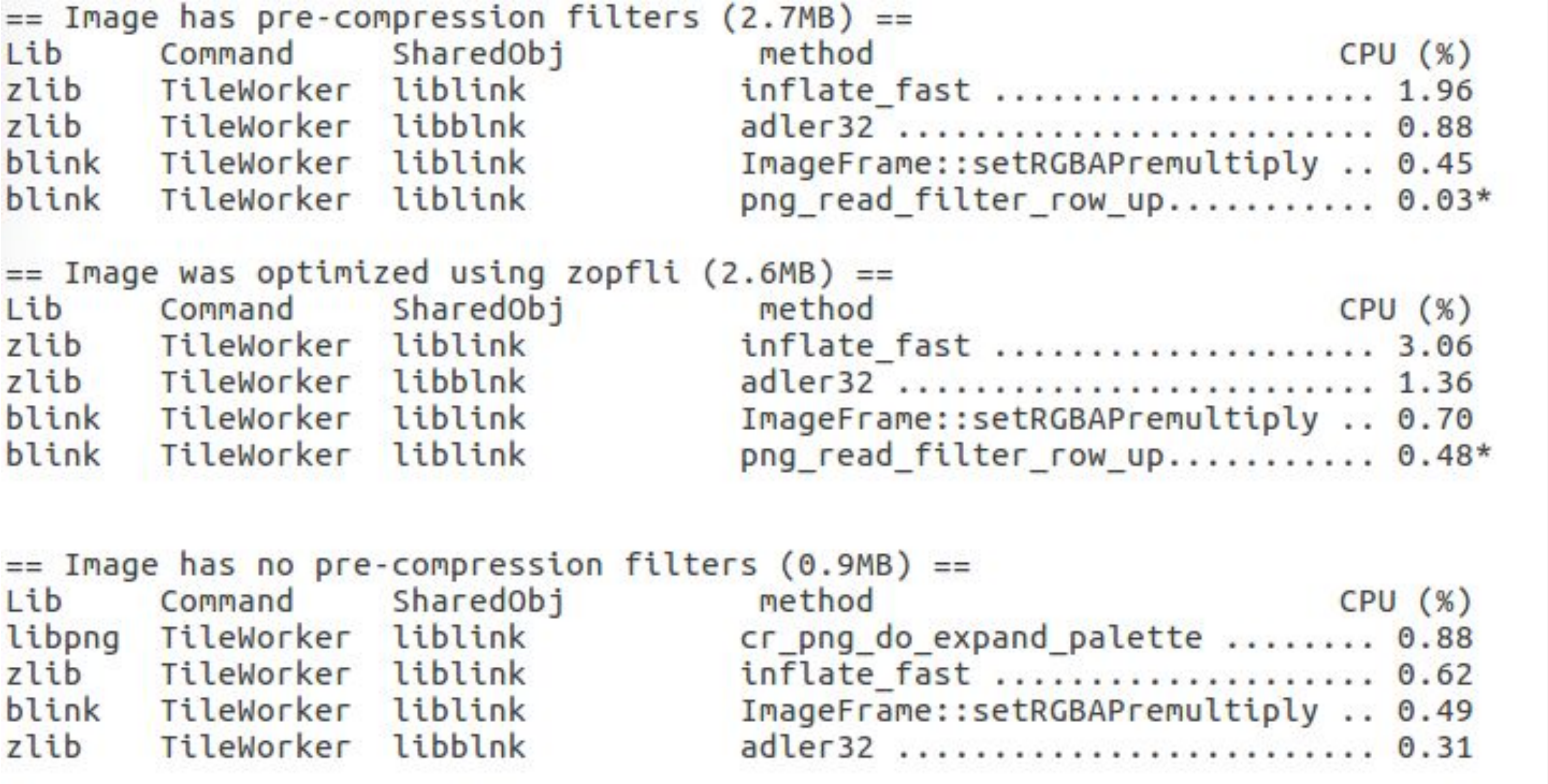
Parrots are not created equal

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
```

Candidates

- Inflate_fast: **zlib**.
- Adler32: **zlib**.
- ImageFrame: Blink.
- png_do_expand_palette: libpng.
- Crc32: **zlib**.

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

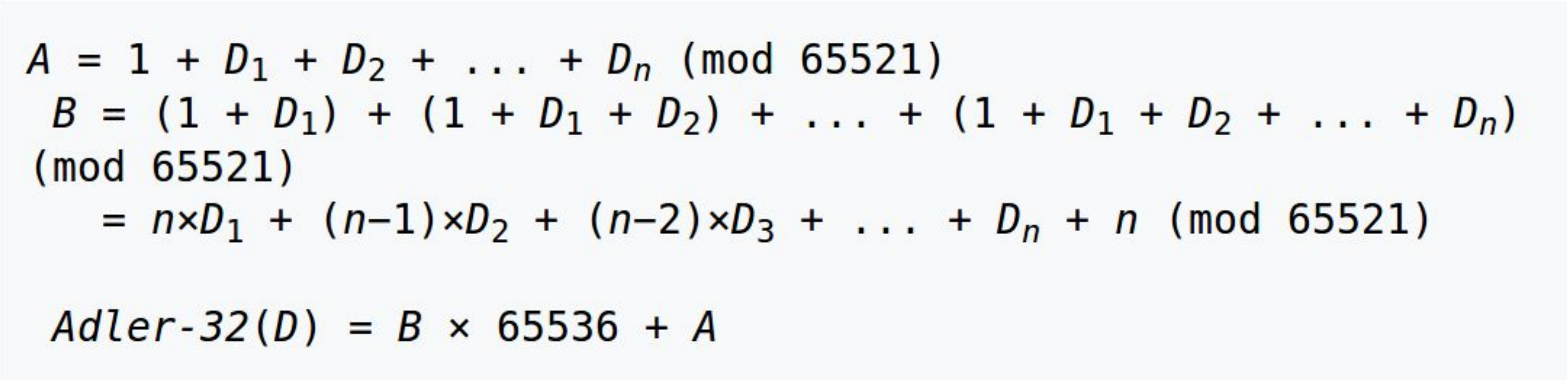
Lacks any optimizations for ARM CPUs.

Problem statement

Identify potential optimization candidates and verify positive effects in Chromium.

Implementation

Adler-32



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

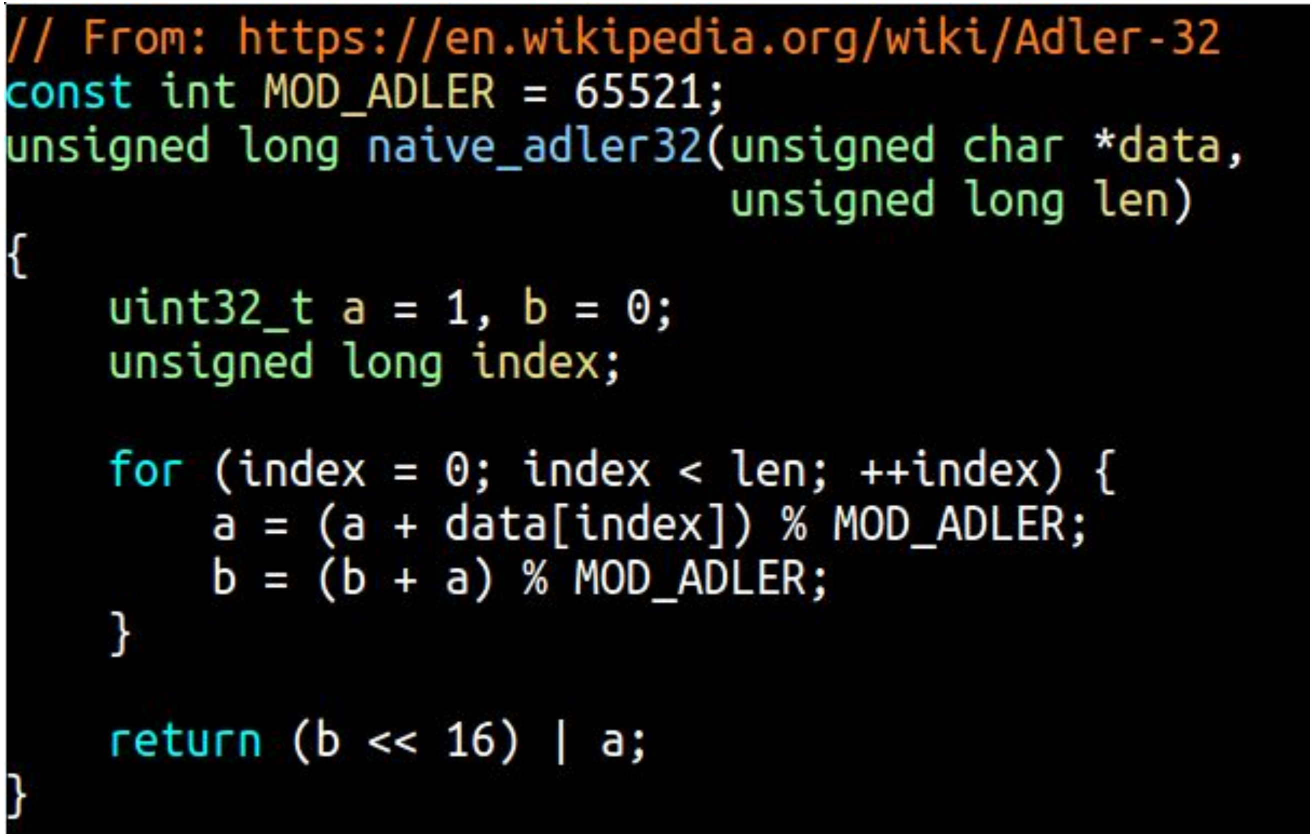
$$B = (1 + D_1) + (1 + D_1 + D_2) + ... + (1 + D_1 + D_2 + ... + D_n) \pmod{65521}$$

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

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

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

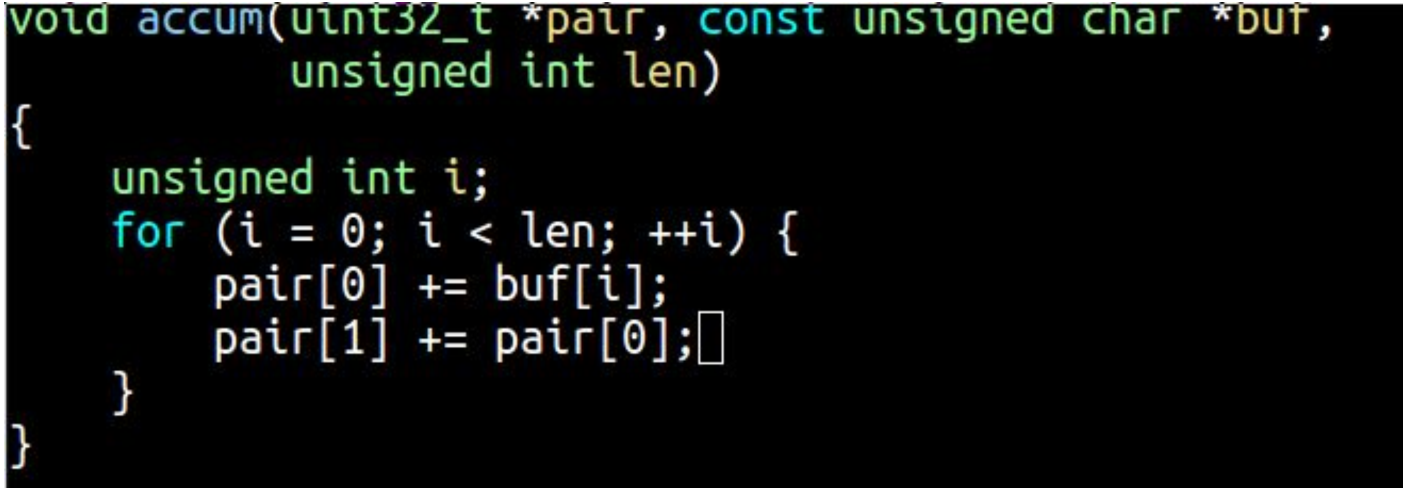
Adler-32: simplistic implementation



```
// 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;
}
```

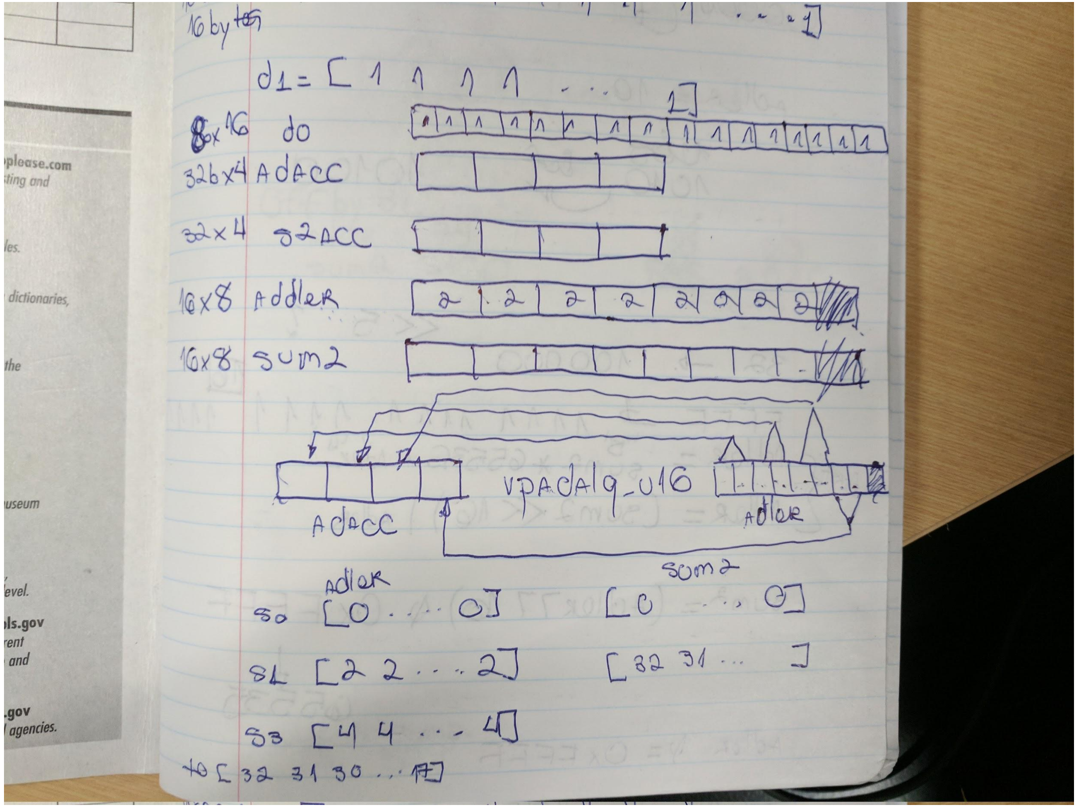
Problems

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

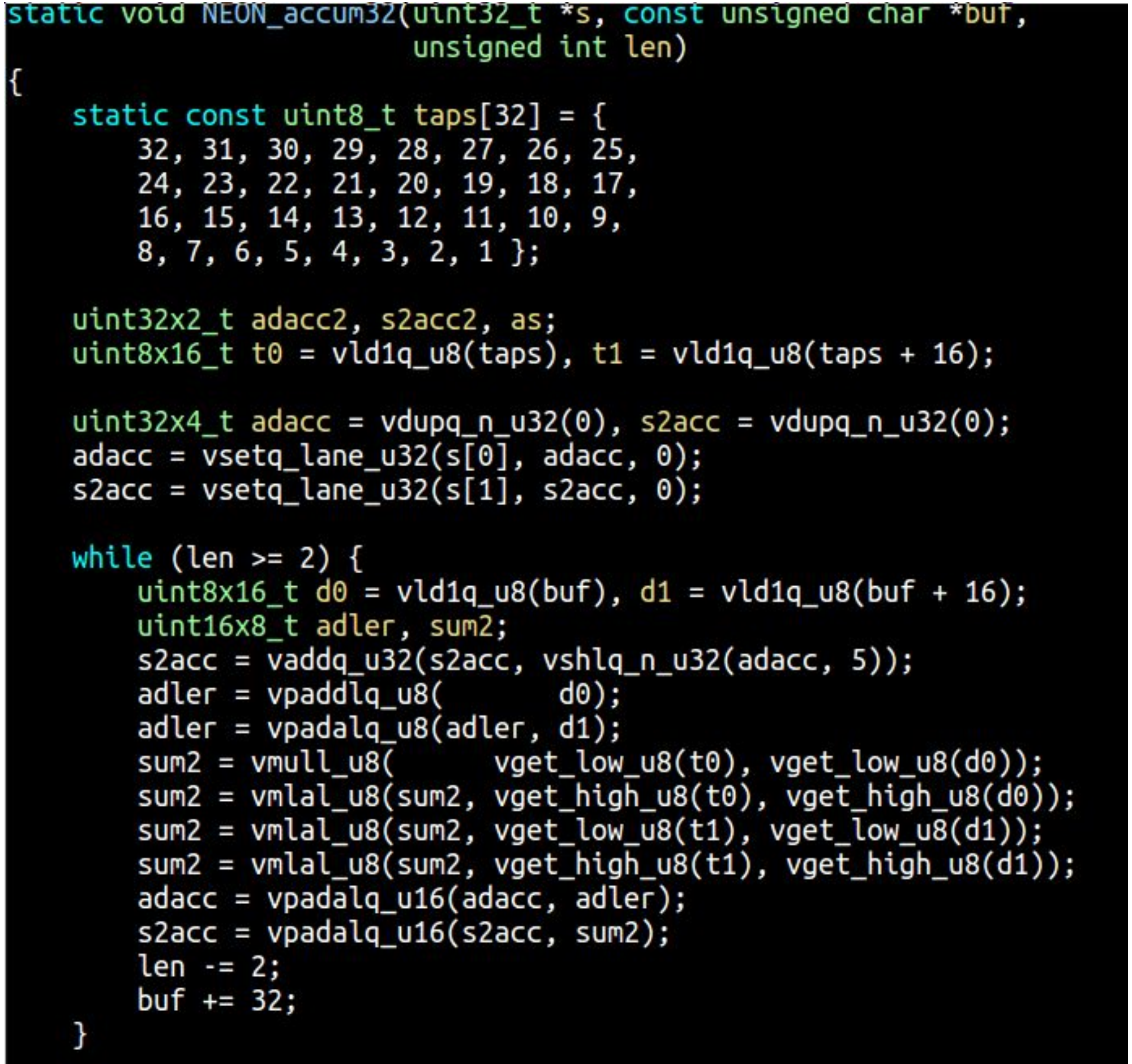


```
void accum(uint32_t *pair, const unsigned char *buf,
           unsigned int len)
{
    unsigned int i;
    for (i = 0; i < len; ++i) {
        pair[0] += buf[i];
        pair[1] += pair[0];
    }
}
```

Highly technical drawing (Jan 2017)



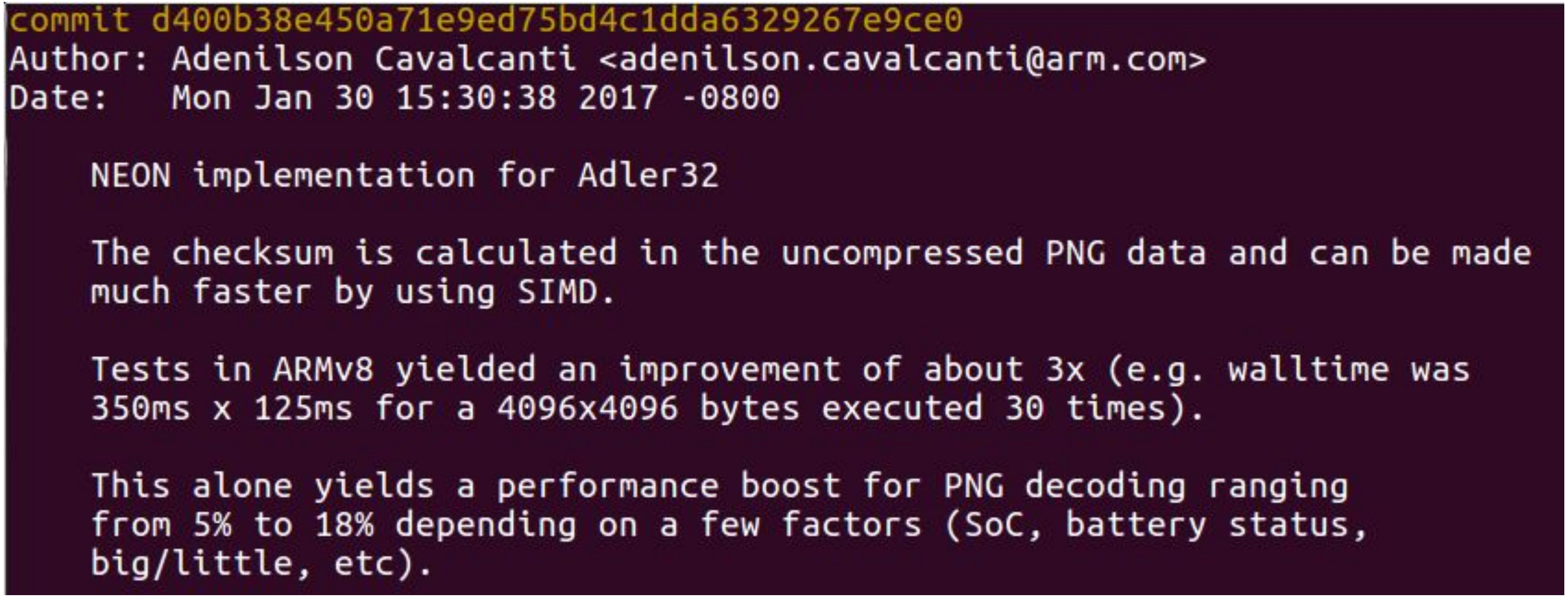
‘Taps’ to the rescue



```
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

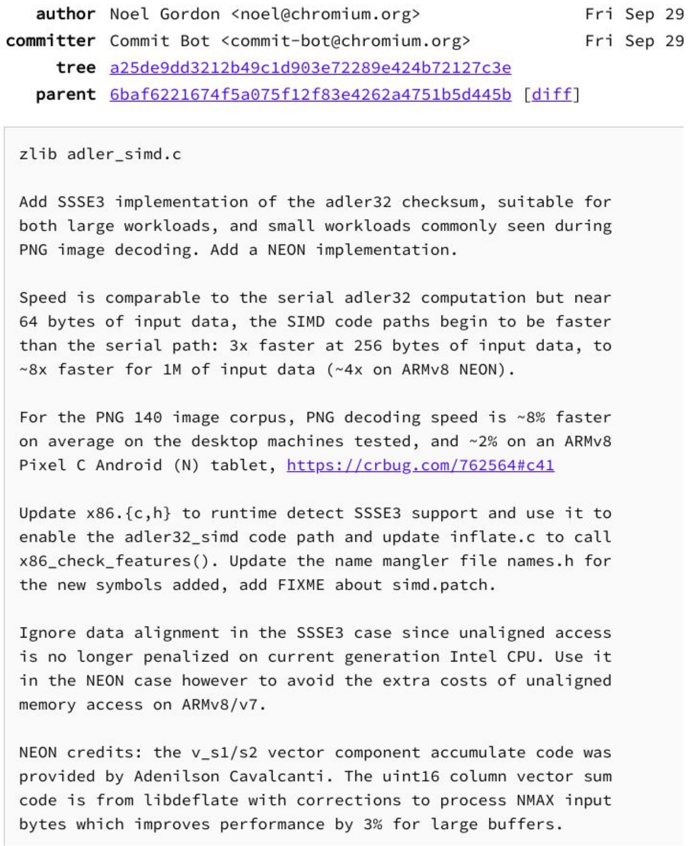
Happy end! Up to 12% performance gain in PNG



```
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

Shipping on M63: Intel got some love too!



```
   author Noel Gordon <noel@chromium.org>            Fri Sep 29
committer Commit Bot <commit-bot@chromium.org>       Fri Sep 29
     tree a25de9dd3212b49c1d903e72289e424b72127c3e
   parent 6baf6221674f5a075f12f83e4262a4751b5d445b [diff]
```

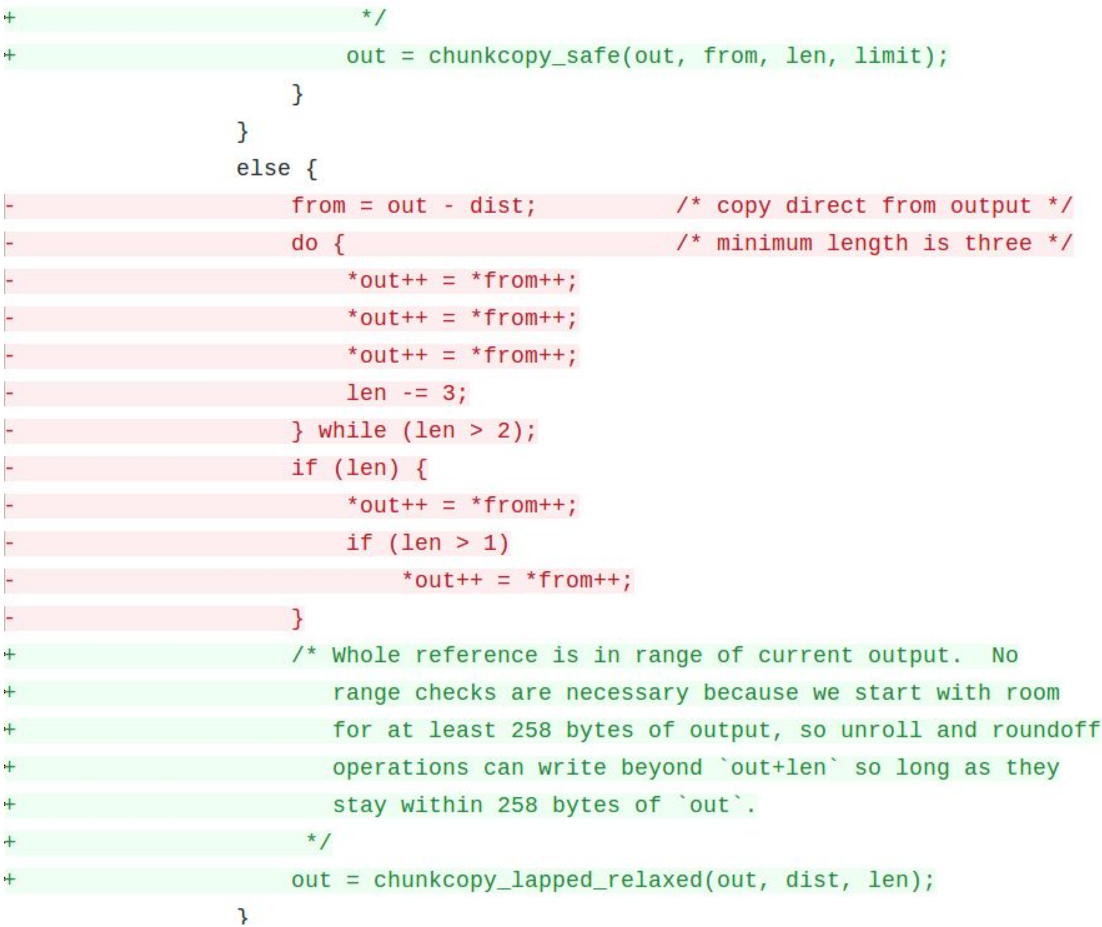
```
zlib adler_simd.c

Add SSSE3 implementation of the adler32 checksum, suitable for
both large workloads, and small workloads commonly seen during
PNG image decoding. Add a NEON implementation.

Speed is comparable to the serial adler32 computation but near
64 bytes of input data, the SIMD code paths begin to be faster
than the serial path: 3x faster at 256 bytes of input data, to
~8x faster for 1M of input data (~4x on ARMv8 NEON).

For the PNG 140 image corpus, PNG decoding speed is ~8% faster
on average on the desktop machines tested, and ~2% on an ARMv8
Pixel C Android (N) tablet, https://crbug.com/762564#c41

Update x86.{c,h} to runtime detect SSSE3 support and use it to
enable the adler32_simd code path and update inflate.c to call
x86_check_features(). Update the name mangler file names.h for
the new symbols added, add FIXME about simd.patch.

Ignore data alignment in the SSSE3 case since unaligned access
is no longer penalized on current generation Intel CPU. Use it
in the NEON case however to avoid the extra costs of unaligned
memory access on ARMv8/v7.

NEON credits: the v_s1/s2 vector component accumulate code was
provided by Adenilson Cavalcanti. The uint16 column vector sum
code is from libdeflate with corrections to process NMAX input
bytes which improves performance by 3% for large buffers.
```

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.
- Average **20% faster**!
- Shipping on M62.

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



```
+                     */
+                   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);
            }
```

Libpng (Richard Townsend)

- NEON optimization in libpng.
- From **10 to 30% improvement**.
- Depends on png using a palette.
- Shipping on M66.

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

Blink::setRGBAPreMultiply (Jonathan Wright)

- NEON optimization in Blink.
- Around **9% improvement**.
- Shipping on M66.

https://bugs.chromium.org/p/chromium/issues/detail?id=702860

CRC32: zlib

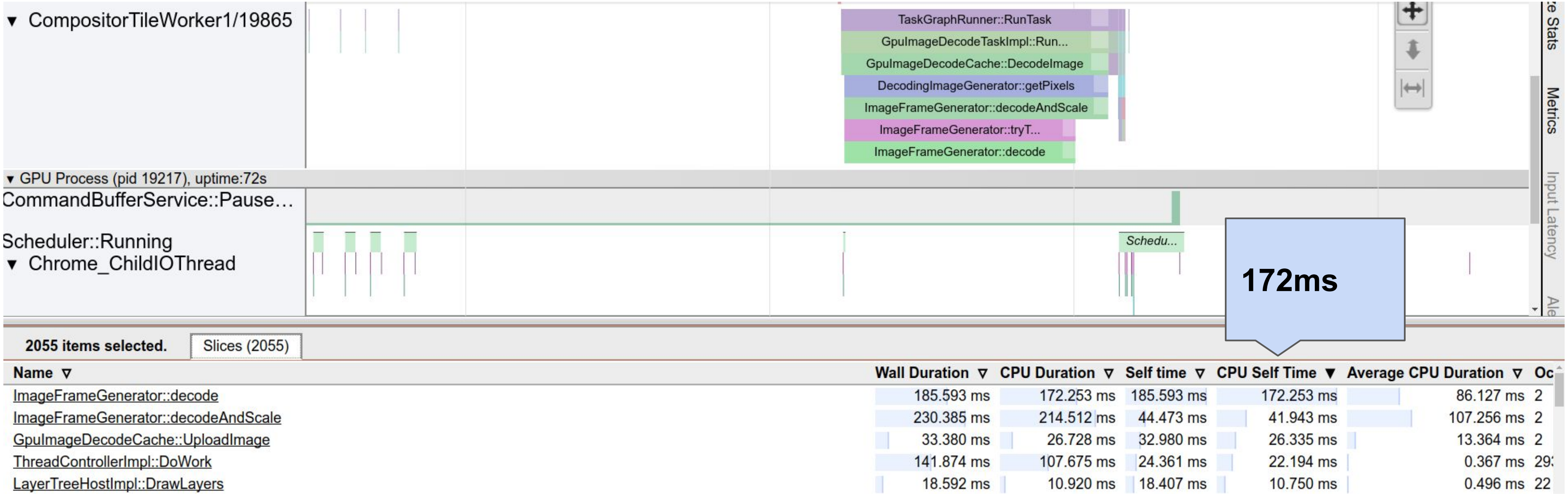
- YMMV on PNGs (from 1 to 5%).
- Remember it is used while **decompressing** web content (29% boost for gzipped content).
- ARMv8-a has a crc32 instruction (from 3 to 10x faster than zlib’s crc32 C code).
- Shipping on M66.

https://bugs.chromium.org/p/chromium/issues/detail?id=709716

Impact

Combined effect of 5 patches

No patches: Chrome M66 trace in a Moto X4


▼ CompositorTileWorker1/19865
TaskGraphRunner::RunTask
GpuImageDecodeTaskImpl::Run...
GpuImageDecodeCache::DecodeImage
DecodingImageGenerator::getPixels
ImageFrameGenerator::decodeAndScale
ImageFrameGenerator::tryT...
ImageFrameGenerator::decode
▼ GPU Process (pid 19217), uptime:72s
CommandBufferService::Pause…
Scheduler::Running
▼ Chrome_ChildIOThread
Schedu...
e Stats
Metrics
Input Latency
Ale

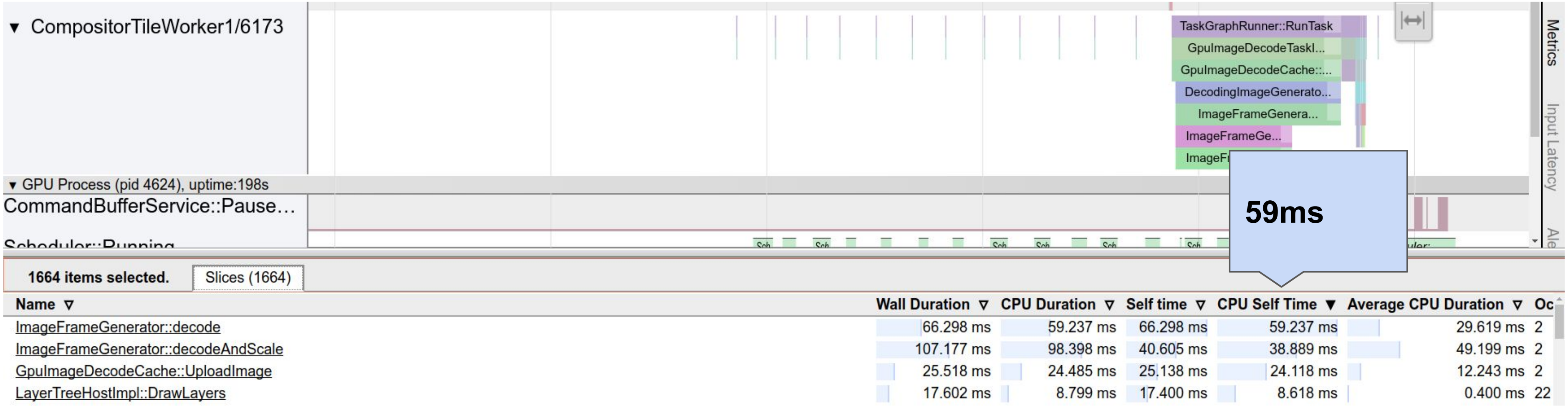
172ms

2055 items selected. Slices (2055)

Name ▽	Wall Duration ▽	CPU Duration ▽	Self time ▽	CPU Self Time ▼	Average CPU Duration ▽	Oc
ImageFrameGenerator::decode	185.593 ms	172.253 ms	185.593 ms	172.253 ms	86.127 ms	2
ImageFrameGenerator::decodeAndScale	230.385 ms	214.512 ms	44.473 ms	41.943 ms	107.256 ms	2
GpuImageDecodeCache::UploadImage	33.380 ms	26.728 ms	32.980 ms	26.335 ms	13.364 ms	2
ThreadControllerImpl::DoWork	141.874 ms	107.675 ms	24.361 ms	22.194 ms	0.367 ms	29
LayerTreeHostImpl::DrawLayers	18.592 ms	10.920 ms	18.407 ms	10.750 ms	0.496 ms	22

172ms spent decoding parrot PNG with palette.

NEON: Chrome M66 trace in a Moto X4



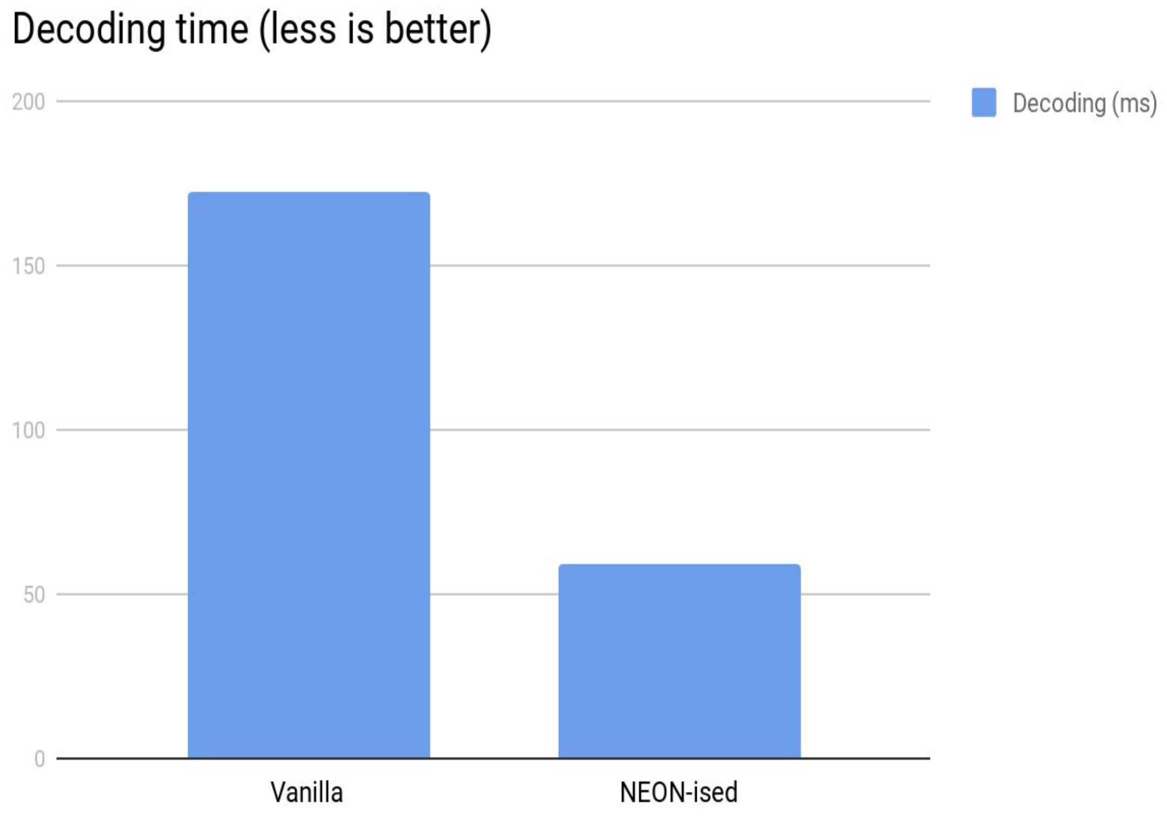
59ms

1664 items selected. Slices (1664)

Name ▽	Wall Duration ▽	CPU Duration ▽	Self time ▽	CPU Self Time ▼	Average CPU Duration ▽	Oc
ImageFrameGenerator::decode	66.298 ms	59.237 ms	66.298 ms	59.237 ms	29.619 ms	2
ImageFrameGenerator::decodeAndScale	107.177 ms	98.398 ms	40.605 ms	38.889 ms	49.199 ms	2
GpuImageDecodeCache::UploadImage	25.518 ms	24.485 ms	25.138 ms	24.118 ms	12.243 ms	2
LayerTreeHostImpl::DrawLayers	17.602 ms	8.799 ms	17.400 ms	8.618 ms	0.400 ms	22

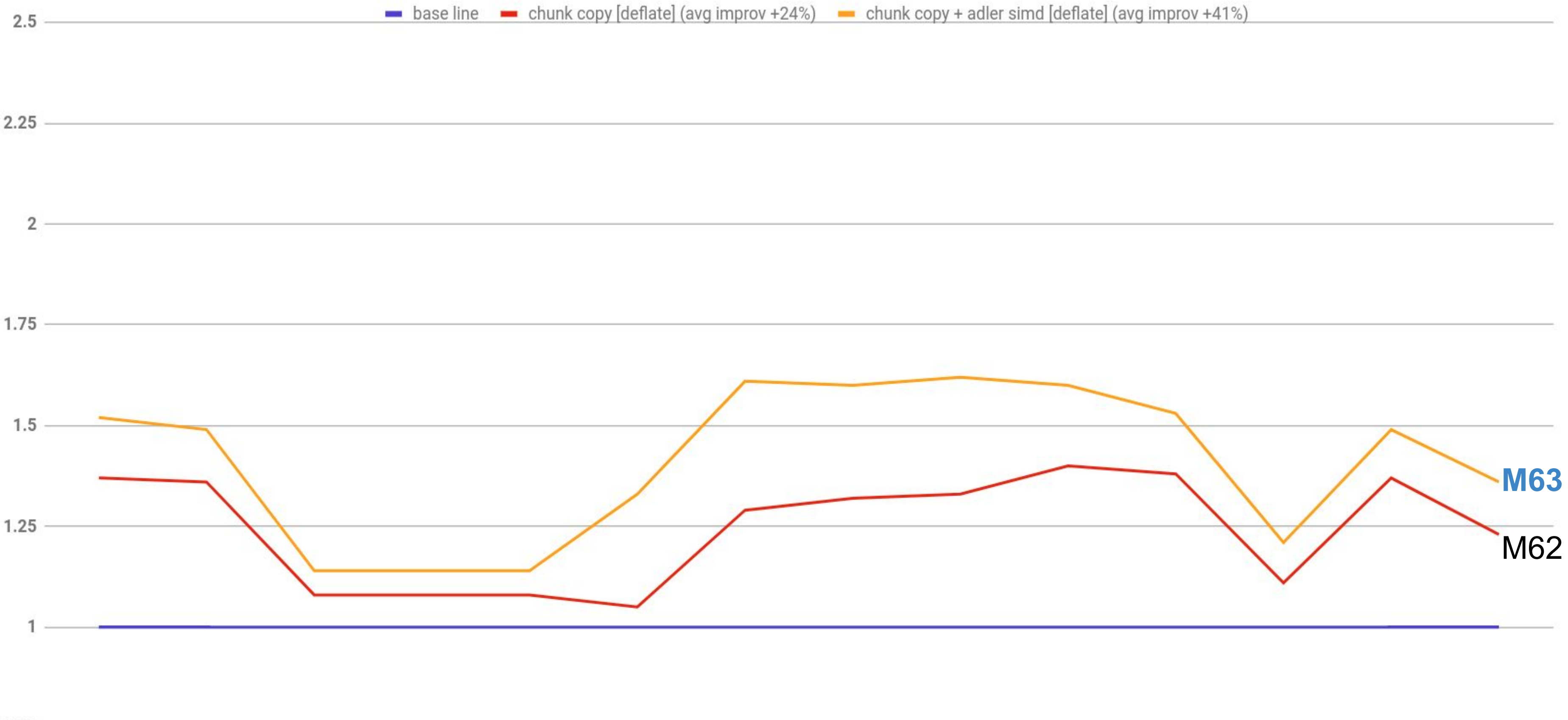
59ms spent decoding parrot PNG with palette.

PNG decoding: 172ms X 59ms (or 2.9x faster)

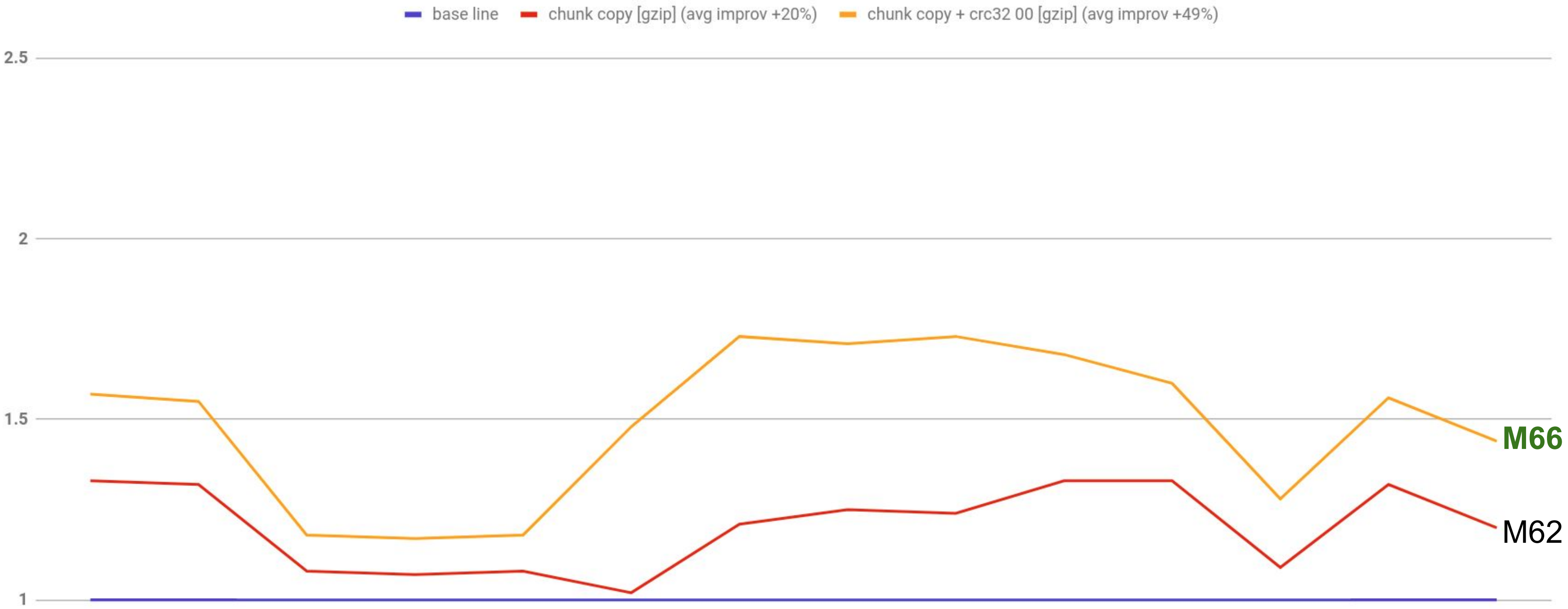


140PNG corpus (**zlib only**): https://bugs.chromium.org/p/chromium/issues/detail?id=709716#c33

Decompression: avg 1.4x for zlib format

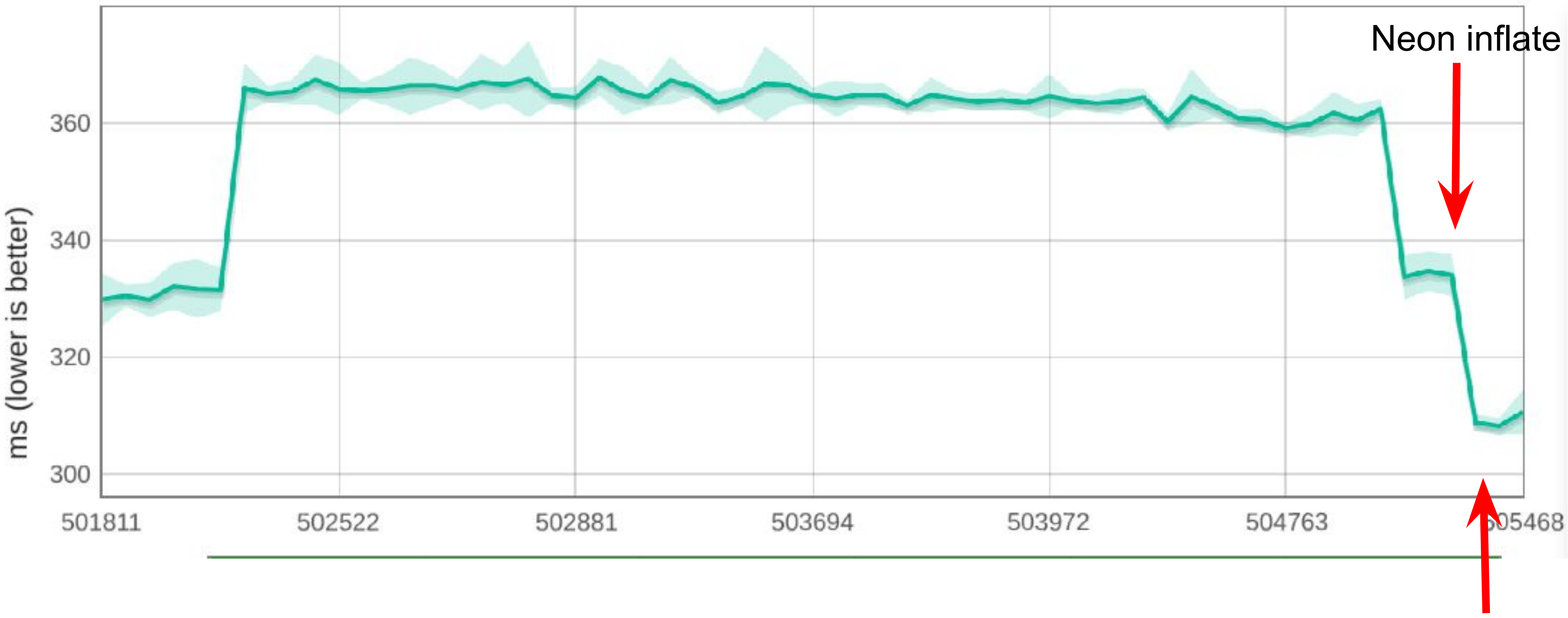


Decompression: avg 1.5x for gzip format



Possible change on compression research? Link:
https://cran.r-project.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf

Google bots confirm the effect of optimizations



Neon inflate

Adler-32

https://goo.gl/RTgkGe

Lessons learned

- **arm** cores can benefit **a lot** from NEON optimizations.
- Performance gains of up to 3 generations* of silicon.
- It pays off to work in a lower software layer (e.g. zlib/libpng).
- Upstreaming can be **really** slow.

What comes next

- Zlib: Compression.
- Port optimizations to Android?

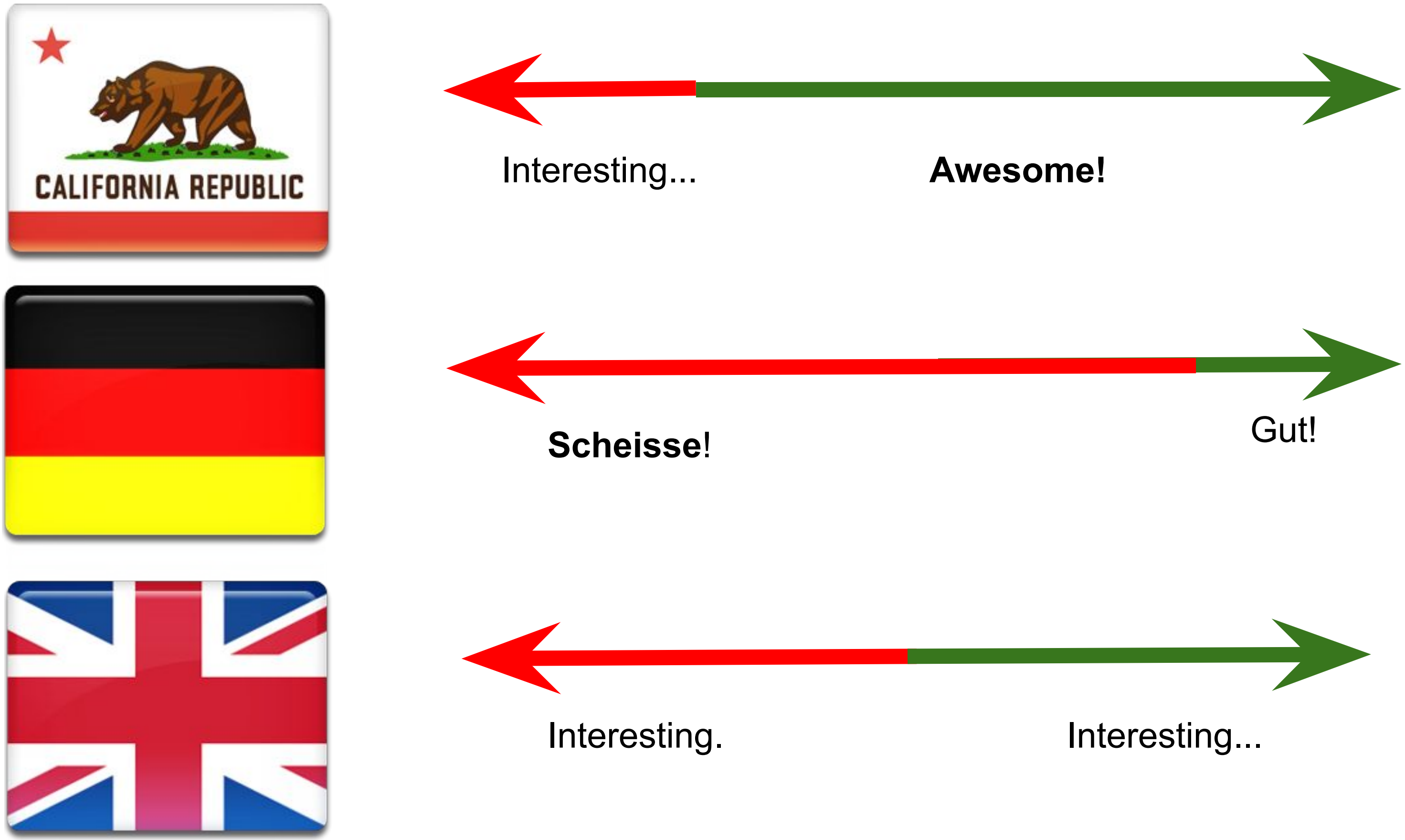
Zlib users should consider migrating to Chromium’s zlib.

Special Thanks

- **Arm** for sponsoring this research.
- Google: Chris Blume (reviews, fuzzers, perf bots, GMeet debugging*), Mike Klein (SIMD), Noel Gordon (tools, Intel port), Leon Scroggins (libpng).
- Team **Arm@UK**: Dave Rodgman, Richard Townsend, Stephen Kyle, Jonathan Wright.
- Team **Arm@US**: Adenilson Cavalcanti (Simon Hosie@google, Amaury Leleyzour@unity).
- Compiler explorer: https://godbolt.org

Questions?

Different kinds of feedback



Interesting... **Awesome!**

Scheisse! Gut!

Interesting. Interesting...