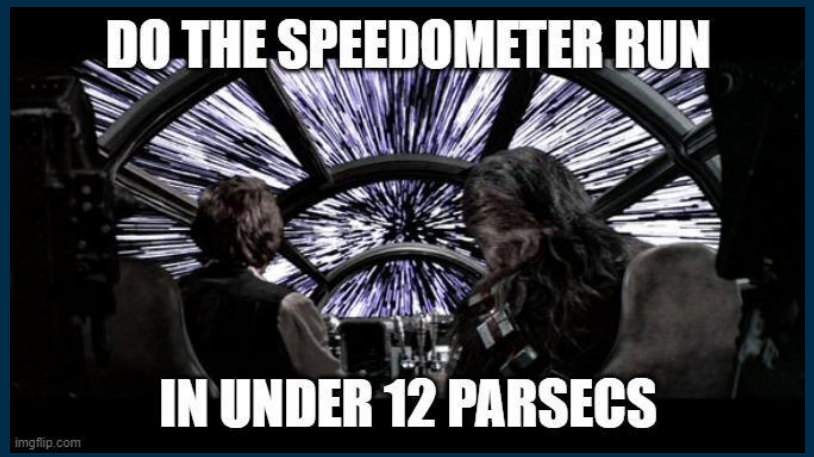
What’s next for Armv9
BlinkOn 16

Android & Browser Enablement Team

Richard Townsend / Jonathan Wright
04/05/2022

Our mission 🚀

Ensure Arm's the best platform for using the web



arm

The Team

JW

Jonathan Wright

libjpeg-turbo OWNER / SIMD expert

RT

Richard Townsend

base/, blink/, build/

AK

André Kempe

PAC, BTI, MTE

C-x C-c

AC

Adenilson Cavalcanti

zlib OWNER

Ctrl + Z

ST

Salomé Thirot

Performance

Rotated to another Arm team (for now)

IR

Ian Rickards

Technology Manager

DK

Daniel Kiss

Tech Lead

JM

Jessica Morgan

Project Manager

richard.townsend@arm.com

arm

Super-fast recap of Armv9

Pointer Authentication (PAC)

- Mitigates stack smashing attacks (return-oriented programming)
- Prevents returning to the wrong place

Branch Target Identification (BTI)

- Mitigates jump-oriented programming

Memory Tagging Extension (MTE)

- Detects memory safety issues: use-after-free, out-of-bounds reads/writes
- Will remain a debug option for now

SVE2

- New vector extension with predicated lanes

arm

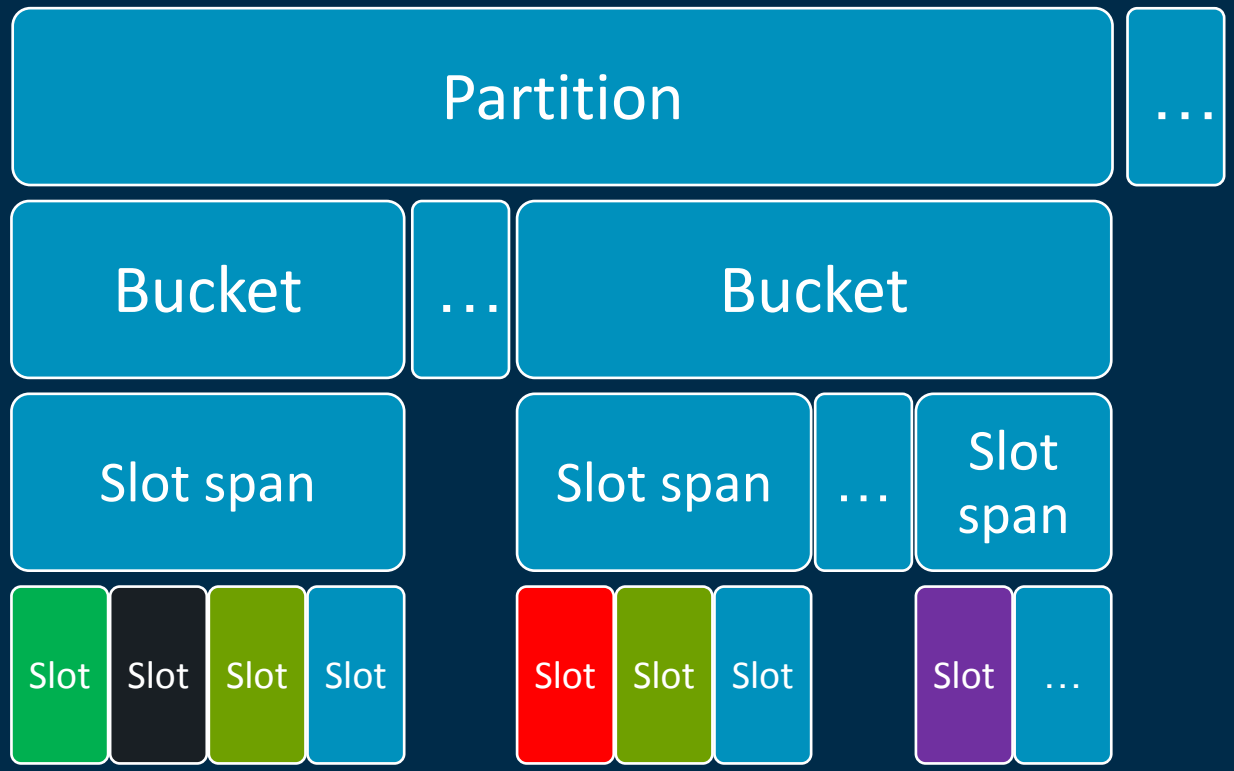
MTE

Memory Tagging Extension

Helps us find and fix memory safety problems earlier in development

```
ptr = 0xf00000000007290
```

- MTE assigns a tag to each pointer and checks if it matches
  - Immediately (sync mode), deferred (async)
- Helps detect use-after-free and out-of-bounds reads and writes
- Lots of interesting ideas on usage



Current thoughts on MTE deployment

- MTE will remain a developer-only/debugging option for at least this year
  - Arm will support Chromium's MTE until it's more widely enabled
- We will investigate synchronous mode for security-critical components (browser, networking, IPC threads)
- Implementing debug/crash-reporting enhancements to make MTE faults more actionable
- Implementing internal test infrastructure
  - Have some development boards with MTE available
- Use-cases this year are QA/validation
  - Focus on fuzzing high-risk areas like audio/video/image codecs

Control flow integrity

Control flow integrity: pointer authentication

Pointer authentication is completely enabled in M102

```c
void log_something(int argument) {
    fprintf(stderr, "something: %d\n", argument);
}
```

Add cryptographically-derived top bits onto LR

Attacker must forge the PAC bits (or find a signing gadget) to take control

Authenticate LR

```
log_something(int):
        paciasp
        adrp    x8, :got:stderr
        mov     w2, w0
        adrp    x1, .L.str
        add     x1, x1, :lo12:.L.str
        ldr     x8, [x8, :got_lo12:stderr]
        ldr     x8, [x8]
        mov     x0, x8
        autiasp
        b       fprintf
```

Control flow integrity: branch target identification

BTI hardens indirect branches taken from a register

```
void do_something(int message, void* payload) {
    switch(message) {
    case 0:
        some_function_1(payload);
    case 1:
        some_function_2(payload);
    case 2:
        some_function_3(payload);
    case 3:
        some_function_4(payload);
    case 4:
        some_function_5(payload);
    case 5:
        some_function_6(payload);
    }
}
```

```
do_something(int, void*):
        cmp     w0, #5
        b.hi    .LBB1_8
        paciasp
        stp     x29, x30, [sp, #-32]!
        str     x19, [sp, #16]
        mov     x29, sp
        mov     w8, w0
        adrp    x9, .LJTI1_0
        add     x9, x9, :lo12:.LJTI1_0
        mov     x19, x1
        adr     x10, .LBB1_2
        ldrb    w11, [x9, x8]
        add     x10, x10, x11, lsl #2
        br      x10
.LBB1_2:
        mov     x0, x19
        bl      some_function_1(void*)
.LBB1_3:
        mov     x0, x19
        bl      some_function_2(void*)
.LBB1_4:
        …
.LBB1_7:
        mov     x0, x19
        ldr     x19, [sp, #16]
        ldp     x29, x30, [sp], #32
        autiasp
        b       some_function_6(void*)
.LBB1_8:
        ret
```

Control flow integrity: branch target identification

BTI hardens indirect branches taken from a register

Can jump here and instantly sign our link register (signing gadget)

Skipped the range check, can now start reading arbitrary memory

If the right value's read, we can go straight to return and defeat PAC

```
do_something(int, void*):
        cmp     w0, #5
        b.hi    .LBB1_8
        paciasp
        stp     x29, x30, [sp, #-32]!
        str     x19, [sp, #16]
        mov     x29, sp
        mov     w8, w0
        adrp    x9, .LJTI1_0
        add     x9, x9, :lo12:.LJTI1_0
        mov     x19, x1
        adr     x10, .LBB1_2
        ldrb    w11, [x9, x8]
        add     x10, x10, x11, lsl #2
        br      x10
.LBB1_2:
        mov     x0, x19
        bl      some_function_1(void*)
.LBB1_3:
        mov     x0, x19
        bl      some_function_2(void*)
.LBB1_4:
        ...
.LBB1_7:
        mov     x0, x19
        ldr     x19, [sp, #16]
        ldp     x29, x30, [sp], #32
        autiasp
        b       some_function_6(void*)
.LBB1_8:
        ret
```

Control flow integrity: branch target identification

BTI hardens indirect branches taken from a register

Now get a bunch of `bti c` and `bti j` instructions…

`bti c` = only accessible as a function entry point

e.g. `blr x2` …

`bti j` = only accessible as a indirect branch target

e.g. `br x10`

`paciasp` = only accessible as a indirect branch target from x16/x17

(we call these *landing pads*)

… which means we can no longer jump to arbitrary points in the function

```
do_something(int, void*):
        bti     c
        cmp     w0, #5
        b.hi    .LBB1_8
        paciasp
        stp     x29, x30, [sp, #-32]!
        str     x19, [sp, #16]
        mov     x29, sp
        mov     w8, w0
        adrp    x9, .LJTI1_0
        add     x9, x9, :lo12:.LJTI1_0
        mov     x19, x1
        adr     x10, .LBB1_2
        ldrb    w11, [x9, x8]
        add     x10, x10, x11, lsl #2
        br      x10
.LBB1_2:
        bti     j
        mov     x0, x19
        bl      some_function_1(void*)
.LBB1_3:
…
.LBB1_7:
        bti     j
        mov     x0, x19
        ldr     x19, [sp, #16]
        ldp     x29, x30, [sp], #32
        autiasp
        b       some_function_6(void*)
.LBB1_8:
        ret
```

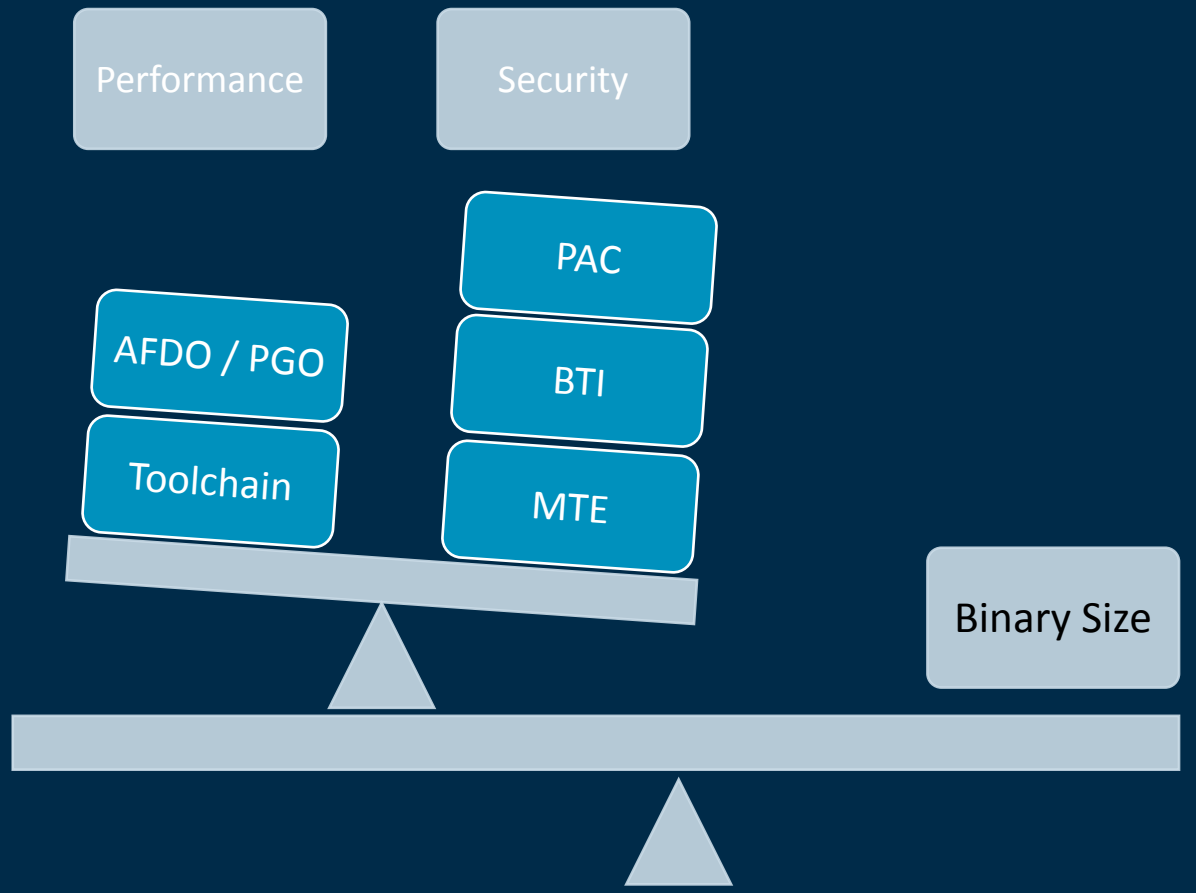
Since BlinkOn 15…

- ✔ Hardened FFMPEG's landing pads
  - ✔ Added support for pointer authentication as well as branch target identification
- ✔ Pointer authentication for C++ has shipped to stable (M101)
- ✔ Pointer authentication for V8 is in M102
- ✔ Branch target identification is starting in M104 (V8), finishing in M105 (C++)
- ✔ We've got devices!
  - ✔ OnePlus 10 Pro (Snapdragon 8 Gen 1)
- ✔ Built out a testing infrastructure for PAC + BTI

arm

Performance

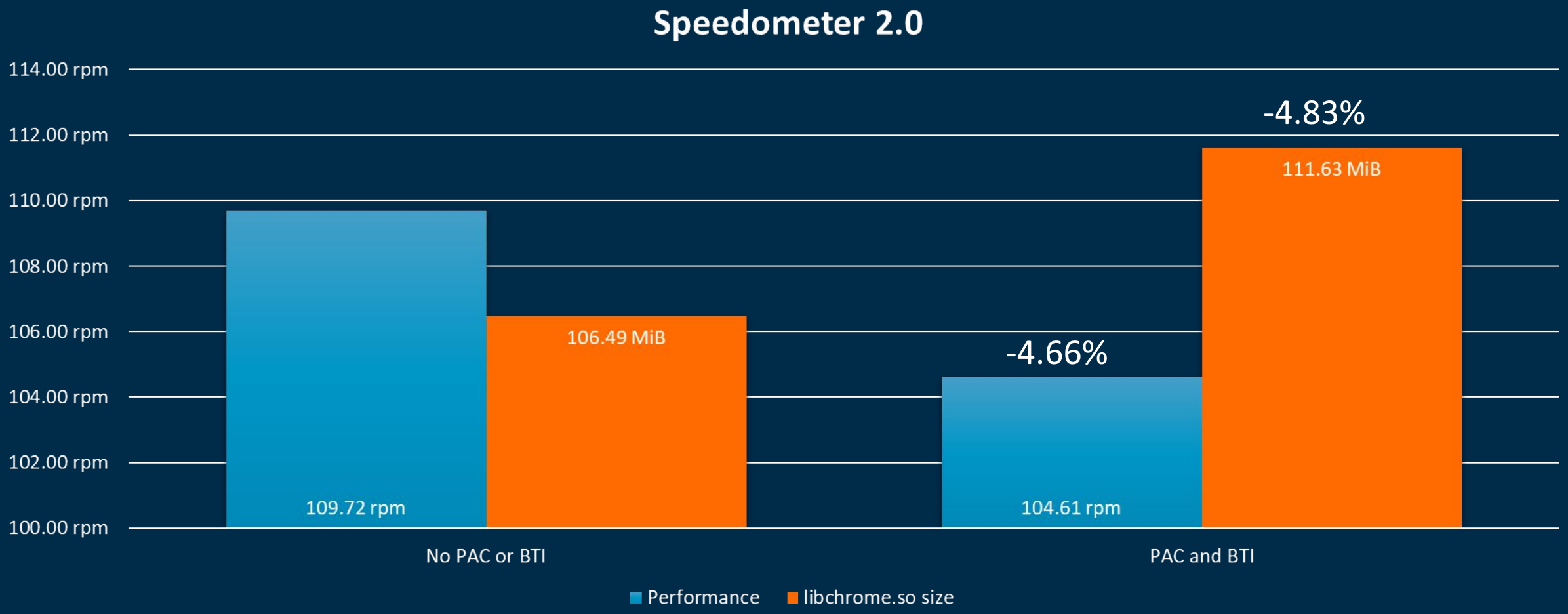
Striking a balance



We want the browser to be as secure as possible,
but also as fast and small as possible

Cost of PAC & BTI

PAC + BTI cost about 5% on Speedometer 2.0



arm

Performance optimization strategy

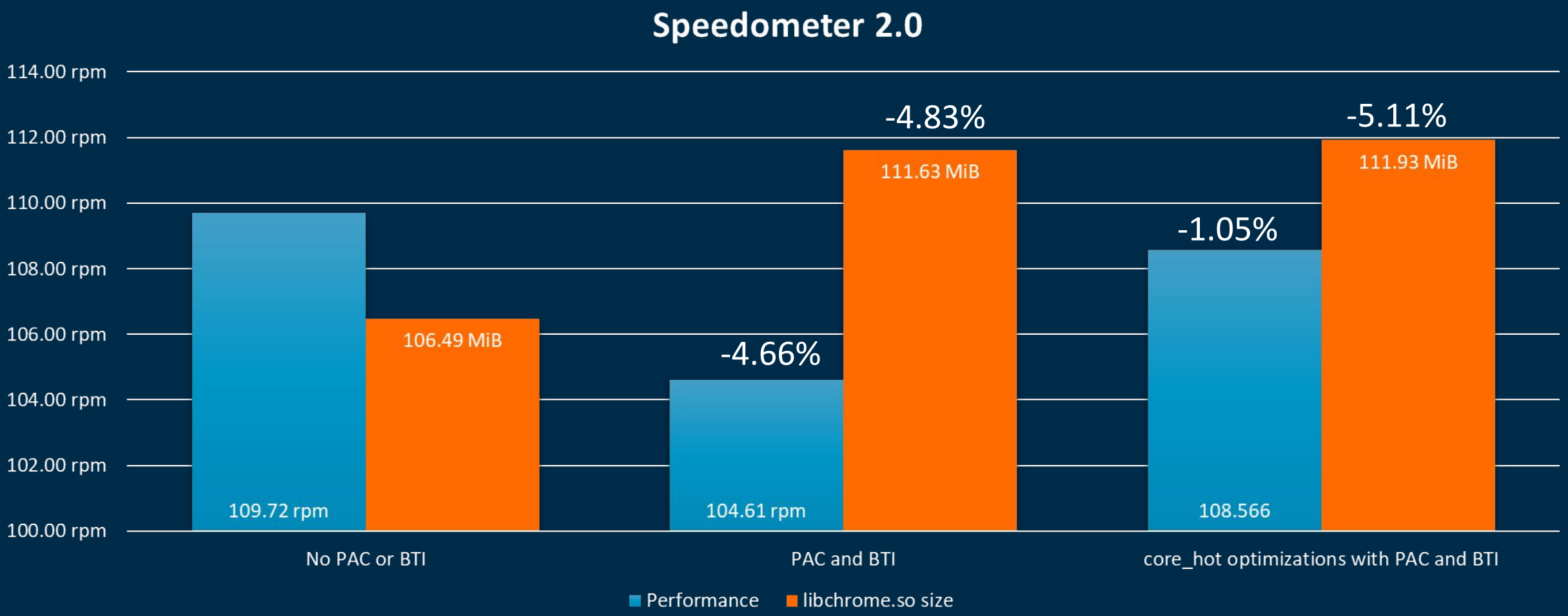
Strategy: improve optimization of critical components

AKA “soft PGO”

- Salomé worked on improving the optimization level in blink/
  - core_hot component contains HTML parser, CSS tokenizer, some layout stuff
  - All built with the highest level of optimization
    - For some reason, optimize_max is not the highest level of optimization
- Upgraded optimization from optimize_max -> optimize_speed
  - … without adding too much binary size
- Technique
  1. Profile the browser to pick out the hottest functions
  2. Include the right .cc files in the core_hot set
  3. If the binary size increases too much, split the .cc file up into a –hot variant
- Quite time-consuming
  - Lots of trade-offs / benchmarking / profiling required
  - Landed in early May

Cost of PAC & BTI

core_hot optimizations get us about 4% back

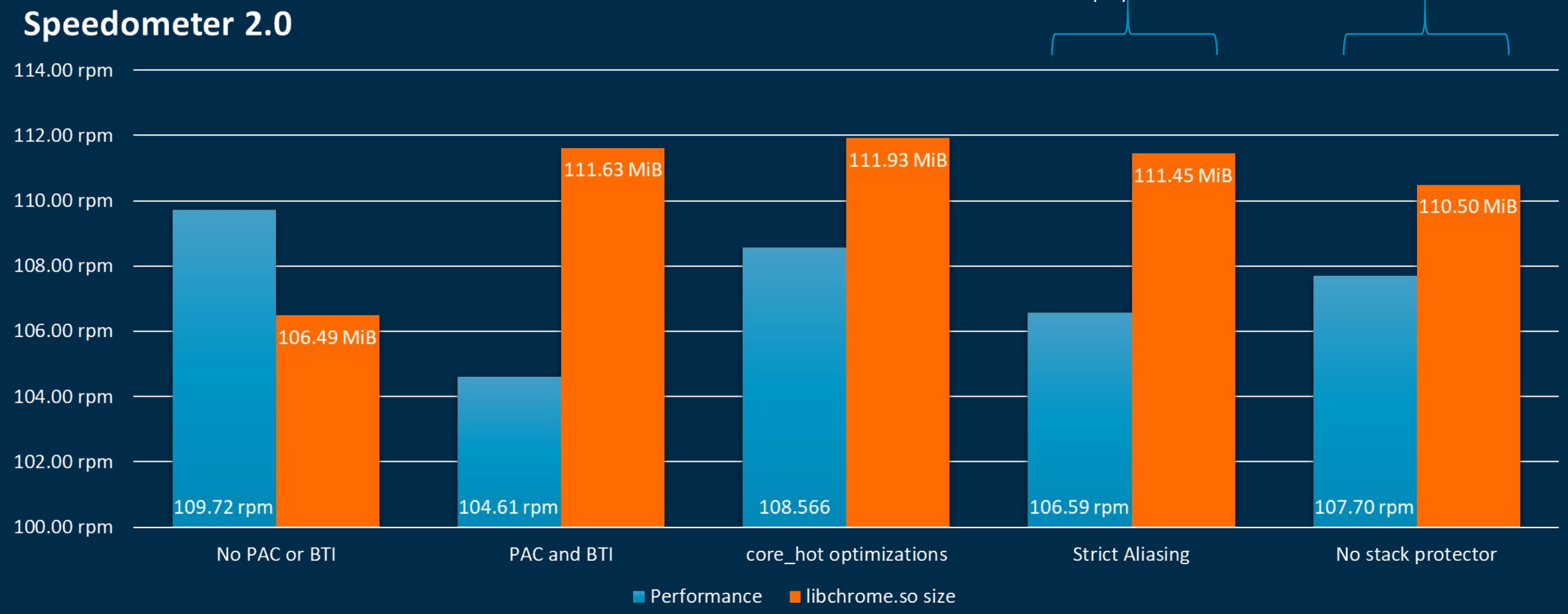


More speculative improvements

Probably in the 1-2% range

-fstrict-aliasing could be deployable

Only deployable as a streamed Armv9-only build with PAC/BTI



Strategy: improve PGO/AFDO

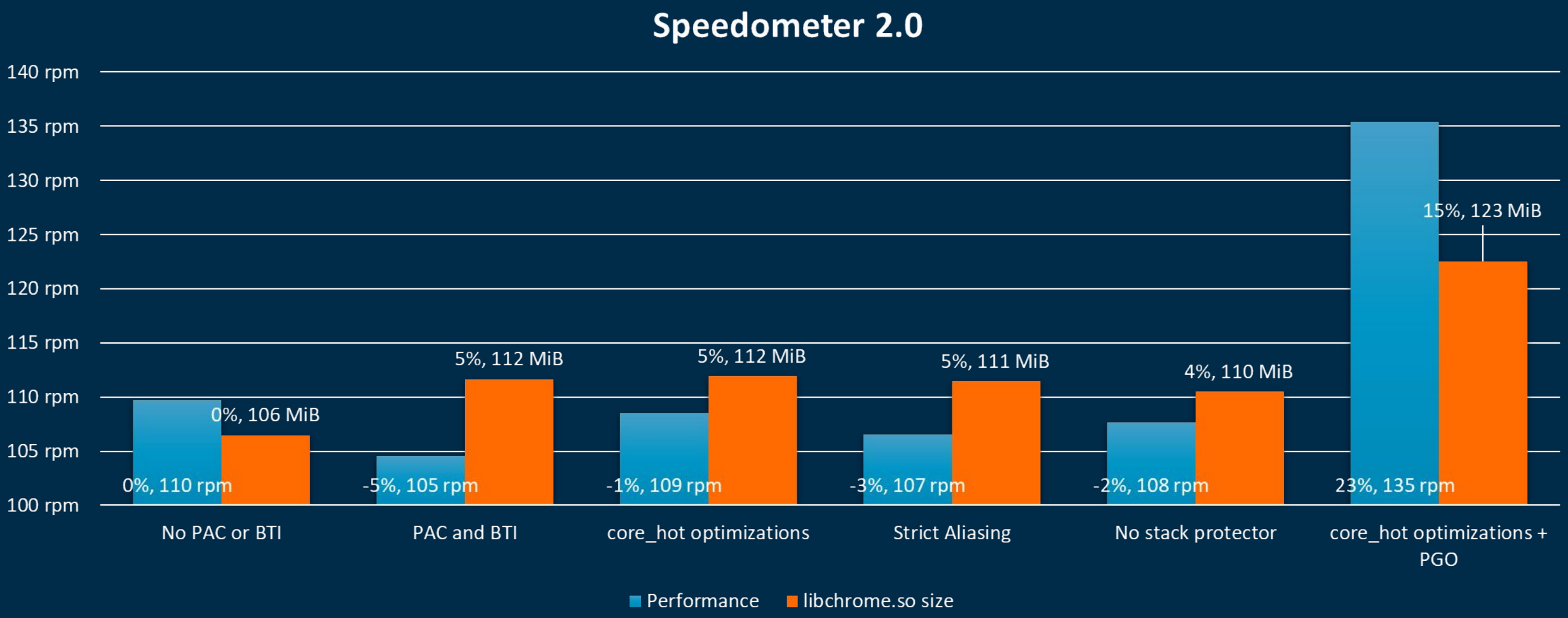
	PGO	AFDO
AKA	Profile-guided optimization	Automatic Feedback-directed optimization
Source	Instrumented Chrome binary	Released Chrome binary
Real user workloads?	No	Yes
Source architecture	Arm	x64 (perhaps Armv7 in future), ChromeOS
Age	Days	Weeks-months

Both techniques give the compiler a lot more information on how to optimize

- e.g. which if-statement branches are often taken
- e.g. which functions tend to be called together
- e.g. which loops to unroll
- e.g. which functions should be inlined

Impact of profile-guided optimization

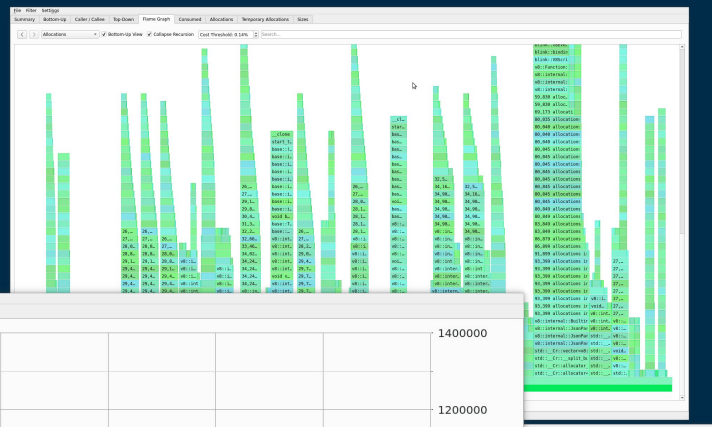
PGO massively improves performance on this particular workload

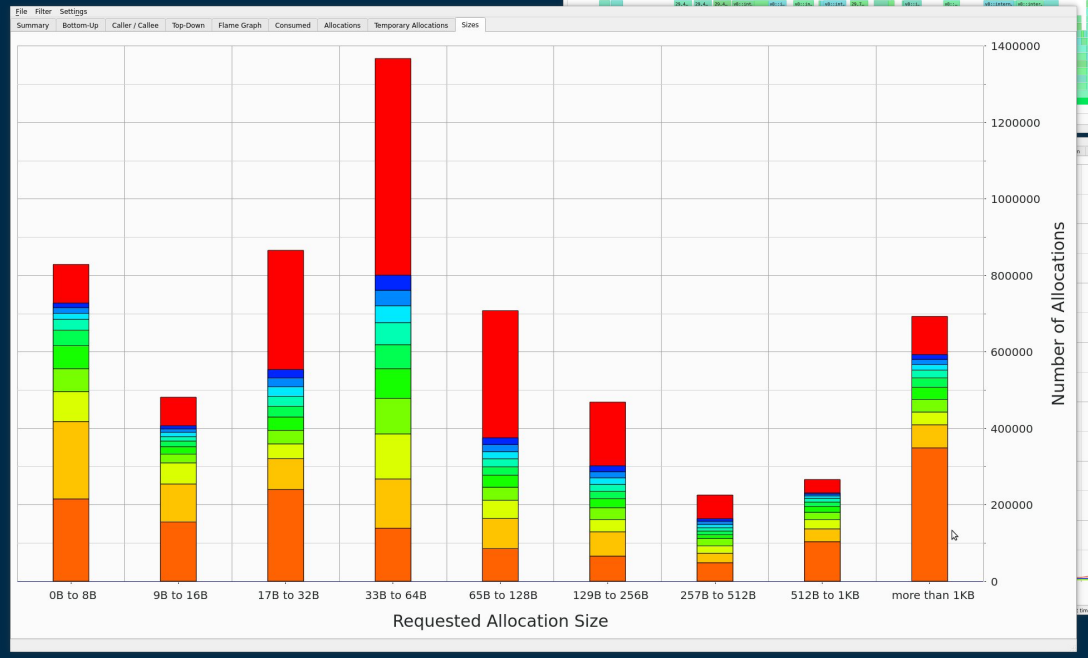


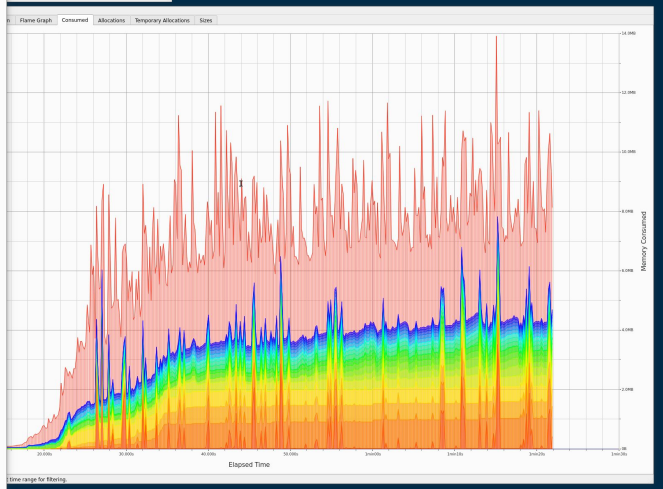
Strategy: reduce unnecessary memory allocations

Interesting results so far from HeapTrack

- Roughly 6 million allocations in Speedometer 2
- 15% of allocations don’t make it outside of their original stack frame
- 16% of malloc() calls request less than 8 bytes

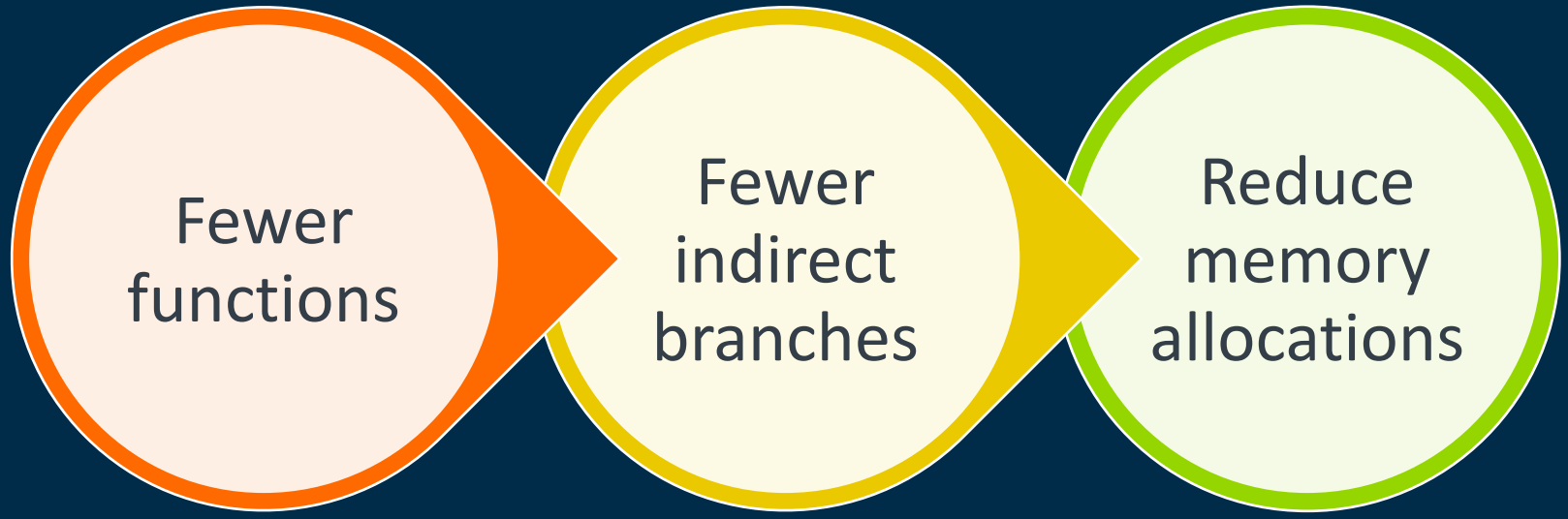






Summary: optimization strategy for Armv9

1) Gather a load of profiles, 2) ????, 3) Profit!



arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה