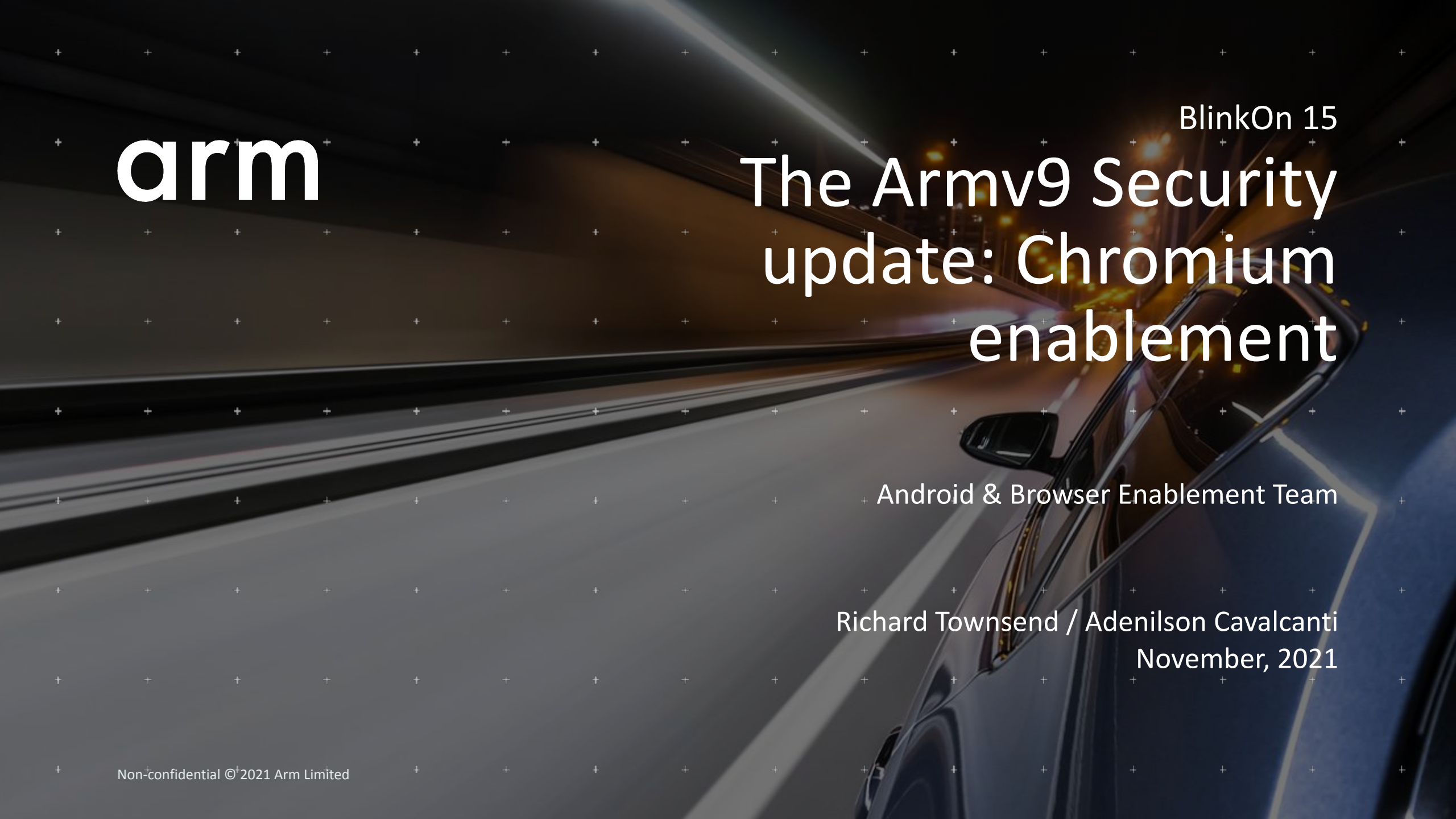


arm

BlinkOn 15

The Armv9 Security update: Chromium enablement

Android & Browser Enablement Team

Richard Townsend / Adenilson Cavalcanti

November, 2021

The team

A

Adenilson Cavalcanti

zlib OWNER, string / hashing / compression expert

JW

Jonathan Wright

libjpeg-turbo OWNER / Image/video expert

R

Richard Townsend

base/, blink/, build/

A

André Kempe New!

PAC and BTI

S

Salomé Thirot New!

PAC and BTI

I

Ian Rickards

Technology Manager

D

Daniel Kiss

Tech Lead

JM

Jessica Morgan New!

Project Manager

arm

Our mission

Ensure Arm’s the best platform for browsing the web

arm

Over the past year

- Revitalised HTMLDocumentParser is in field trials
  - See lightning talk for more
- Some zlib compression research
- Steadily adding PAC/BTI/MTE support to Chrome
  - MTE support merged into the partition allocator
  - Chromium + most dependencies are ready for PAC + BTI

AKA Branch Target Identification

Reminder about PAC/BTI

AKA Pointer authentication (PAuth)

- `bti c` and `bti j` are *landing pads*
  - bti c = must be called in a function-like way
  - Also have `bti j` for indirect jumps
- `pacia` is a pointer signing (PAC) instruction
  - "sign this address with the A-key"
  - There's also a B-key (currently unused in Chromium)
  - `retaa` is the "return and authenticate" instruction
    - Basically, makes sure that what we're going back to is what we originally expected
    - In Chrome, you'll see an `autiasp` or `hint #29` instruction
- Together, PAC/BTI strengthen control flow integrity, making it harder to start sandbox attacks via ROP + JOP

```c
#include <stdio.h>

void do_something_PAC(int with) {
    fprintf(stderr, "%d\n", with);
}

int main(int argc, char **argv) {
    do_something_PAC(argc);
    return 0;
}
```

```
do_something(int):
        bti     c
        adrp    x8, stderr
        adrp    x1, .L.str
        mov     w2, w0
        add     x1, x1, :lo12:.L.str
        ldr     x8, [x8, :lo12:stderr]
        mov     x0, x8
        b       fprintf
main:
        bti     c
        pacia   x30, sp
        stp     x29, x30, [sp, #-16]!
        mov     x29, sp
        bl      do_something(int)
        mov     w0, wzr
        ldp     x29, x30, [sp], #16

.L.str:
        .asciz  "%d\n"
```

PAC and BTI reinforce each other

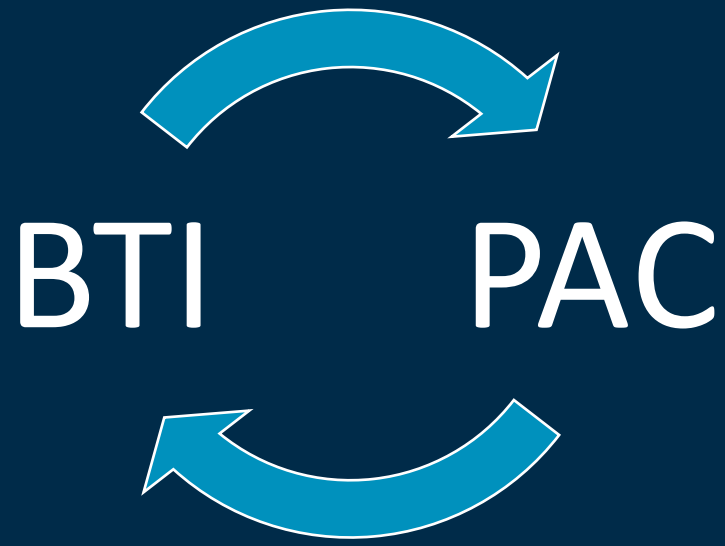
We can deploy PAC without BTI…

but much of the work is shared
PAC interoperates with non-PAC code (e.g. prebuilts), but to get complete coverage we need to convert everything

Every .o file in the codebase must support BTI for it to work, meaning we need:

- NDK r23
- third_party/ dependencies
- compiler-rt/libunwind migration
- compiler-rt recompile

… which (co-incidentally) are all the things we need for strong PAC coverage too



arm

One year ago (BlinkOn13), we presented this slide

Deploying PAC & BTI

PAC is easy (-mbranch-protection=pac-ret), BTI is a bit more difficult

PAC: mostly a compiler flag

- 700 kB APK size increase (95MB → 96 MB)
- Small difference on existing phones (0% to -2% depending on how hot is the code)
- Next steps:
  - Official build investigation
  - Pinpoint runs
  - Ship for AArch64 Linux
  - Ship for AArch64 Android
- Hope to ship next year

BTI: also mostly a compiler flag

Current status since BlinkOn 13

Adjustment: ship PAC+BTI as one technology (due to V8, libunwind, clang_rt dependency)

PAC: mostly a compiler flag

Done

- 700 kB APK size increase (95MB → 96 MB)
- Small difference on existing phones (0% to -2% depending on how hot is the code)
- Next steps:
  - ~~Official build investigation~~
  - ~~Pinpoint runs~~
  - ~~Ship for AArch64 Linux~~
  - ~~Ship for AArch64 Android~~

WIP

- Hope to ~~ship~~ complete ~~next~~ late this year

BTI: also mostly a compiler flag

- But… have to make the whole libchrome.so BTI-capable — In progress
  - Adding landing pads to ffmpeg/libdav1d assembly code (use intrinsics!) — Done
  - Remove / rebuild prebuilt binaries
    - crt_begin, libgvr etc — Done (mostly)
- Expect similar performance / code size impact

What have we done?

- We have a new flag: **`arm_control_flow_integrity="standard"`**
  - Sets `v8_control_flow_integrity=true` for runtime BTI protection
  - Adds `-mbranch-protection=standard` to all .cc files
  - Eventually, we hope it'll become the default for all Armv8 builds
    - (loudly complain when non-BTI compliant code's checked in)
- We've made changes to libdav1d, breakpad, libunwind
  - Adding BTI landing pads to assembly files
  - Adding missing "I support PAC + BTI" sections to assembly files
- libgvr_shim recompiled
- We've moved to LLVM project's compiler-rt and libunwind on Android
  - libgcc 4.9 was too old to properly understand PAC + BTI stack unwinding
- We are moving to the latest NDK (r23) for crtbegin.o and friends
  - PAC + BTI is available in Android 12

Still to do…

- Update the compiler toolchain package to include compiler-rt with PAC + BTI
  - Blocked on NDK changes
- Finalize crashpad support
- Upstream and downstream the ffmpeg landing pads
  - Undergoing review (update: merged! 🎉)
- Still confident that anyone could build a PAC + BTI Chromium from ToT (Top of Tree) by the end of this year
  - Still some questions about how to deploy (i.e. official Chrome build)
- PAC could be made even stronger in future (by signing things like virtual function pointers)
  - Exploring the implications of this

The threat landscape for M95 (95.0.4638.69)

High CVE-2021-37997: **Use after free in Sign-in**
High CVE-2021-37998: Use after free in Garbage Collection
High CVE-2021-3799: Insufficient data validation in New Tab Page
High CVE-2021-3800: Insufficient validation of untrusted input in Intents
High *CVE-2021-38001: Type confusion in V8*
High CVE-2021-38002: **Use after free in Web Transport**
High CVE-2021-38003: Inappropriate implementation in V8
Medium CVE-2021-38004: Insufficient policy enforcement in autofill

https://chromereleases.googleblog.com/2021/10/stable-channel-update-for-desktop_28.html

arm

The threat landscape for M95 (95.0.4638.54)

High CVE-2021-37981: Heap buffer overflow in Skia

High CVE-2021-37982: Use after free in Incognito

High CVE-2021-37983: Use after free in Dev Tools

High CVE-2021-37984: Heap buffer overflow in PDFium

High CVE-2021-37985: Use after free in V8

Medium CVE-2021-37986: Heap buffer overflow in Settings

Medium CVE-2021-37987: Use after free in Network APIs

Medium CVE-2021-37988: Use after free in Profiles

Medium CVE-2021-37989: Inappropriate implementation in Blink

Medium CVE-2021-37990: Inappropriate implementation in WebView

Medium CVE-2021-37991: Race in V8

Medium CVE-2021-37992: Out of bounds read in WebAudio

Medium CVE-2021-37993: Use after free in PDF Accessibility

Medium CVE-2021-37996: Insufficient validation of untrusted input in Downloads

The threat landscape for M95

Out of 22 Medium/High CVEs publicly disclosed in M95, **13** were related in memory safety

- 9 use after free (7 targeted by heap tagging)
- 4 spatial safety problems

We're reasonably confident that MTE can find memory bugs – it's already found one:

Fix use-after free in CoordinatorImplTest. (pcc)

https://chromium-review.googlesource.com/c/chromium/src/+/3050835

arm

What is MTE?

- MTE is Arm's *Memory Tagging Extension*
  - Every 16-byte region of memory (granule) can now have 1 of 16 different *tags*
    - Tags are encoded in the top bits of a pointer, stored in a shadow area of memory
  - Each time we load or store a pointer, we check whether the pointer's top bits match the shadow area
    - Mismatch = eventual or immediate crash (depending on enforcement mode)
  - We have new Armv8.5 instructions for generating + setting tags
- MTE *probabilistically* detects use-after-free and buffer overflows in production software
  - Two main flavours: heap tagging (backwards compatible) and stack tagging (requires Armv8.5 hardware)
    - We've implemented heap tagging
- MTE comes in two modes: sync and async
  - Sync crashes precisely when the fault is detected
  - Async crashes later – higher performance (can switch from async to sync)

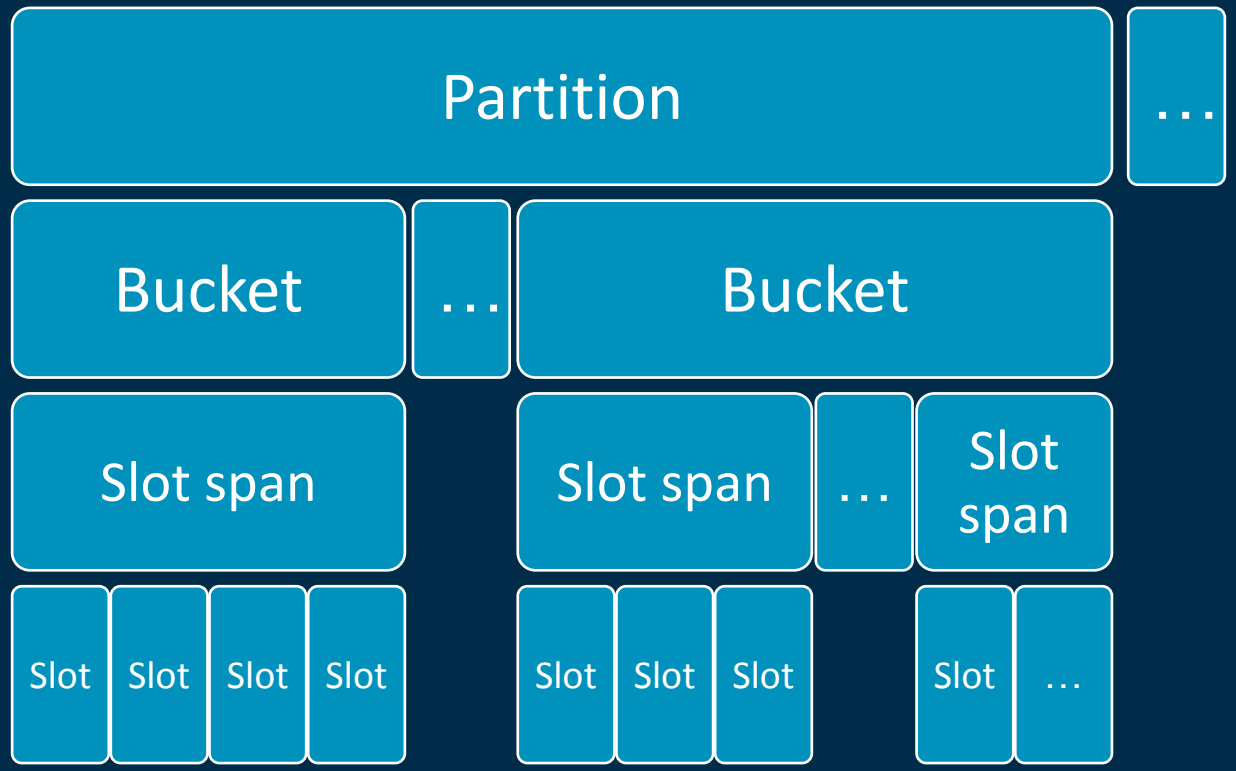
How heap tagging works in the partition allocator

Apologies to bartekn@

Each *partition* is subdivided into buckets holding objects of similar size

Each bucket contains one or more *slot spans*, now backed with PROT_MTE

Upon slot span provisioning, we tag each slot randomly



arm

How heap tagging works in the partition allocator

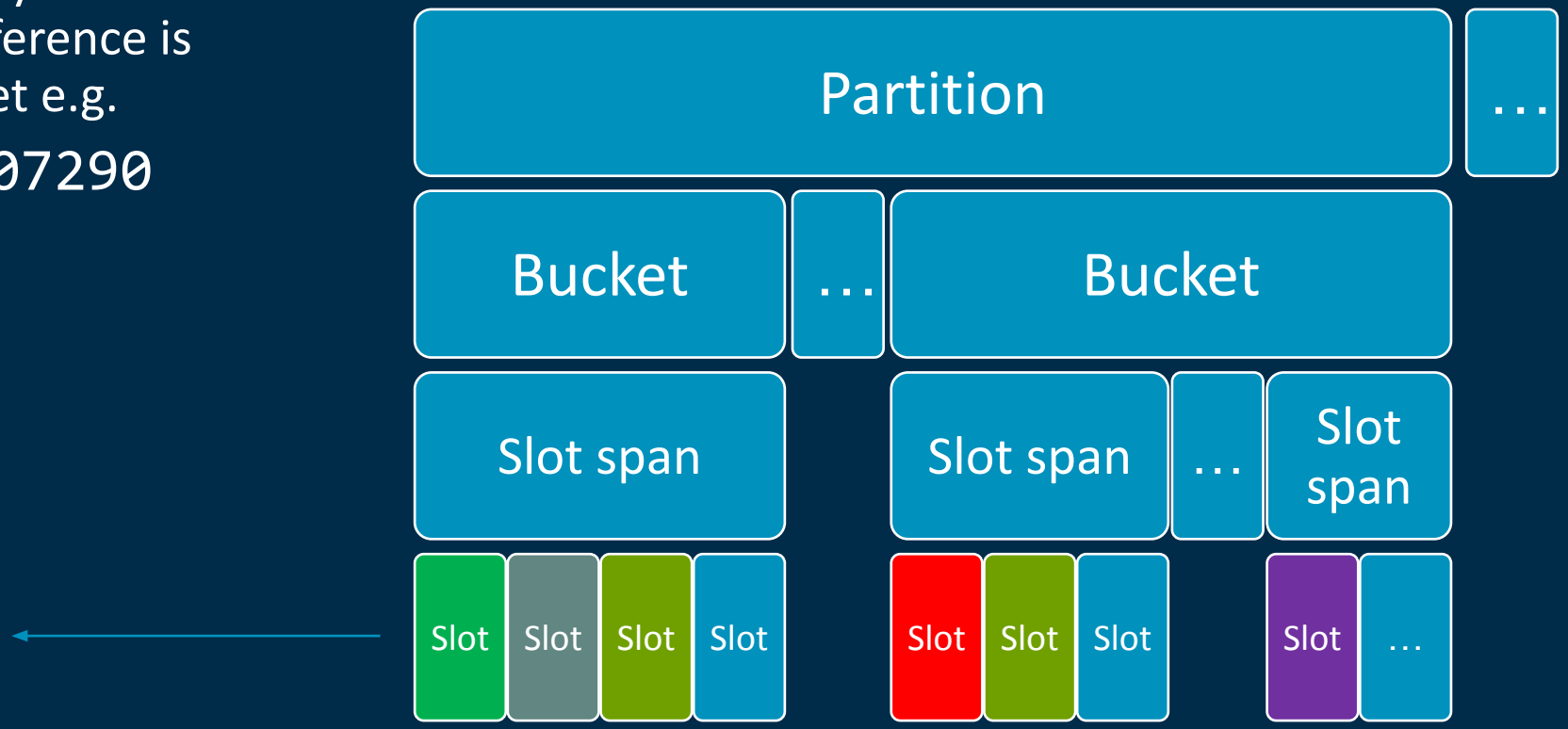
Apologies to bartekn@

Now, we can allocate memory as normal, only discernable difference is that there's now a high bit set e.g.

```
ptr = 0xf000000000007290
```

malloc(…)

PartitionRoot::Alloc(…)



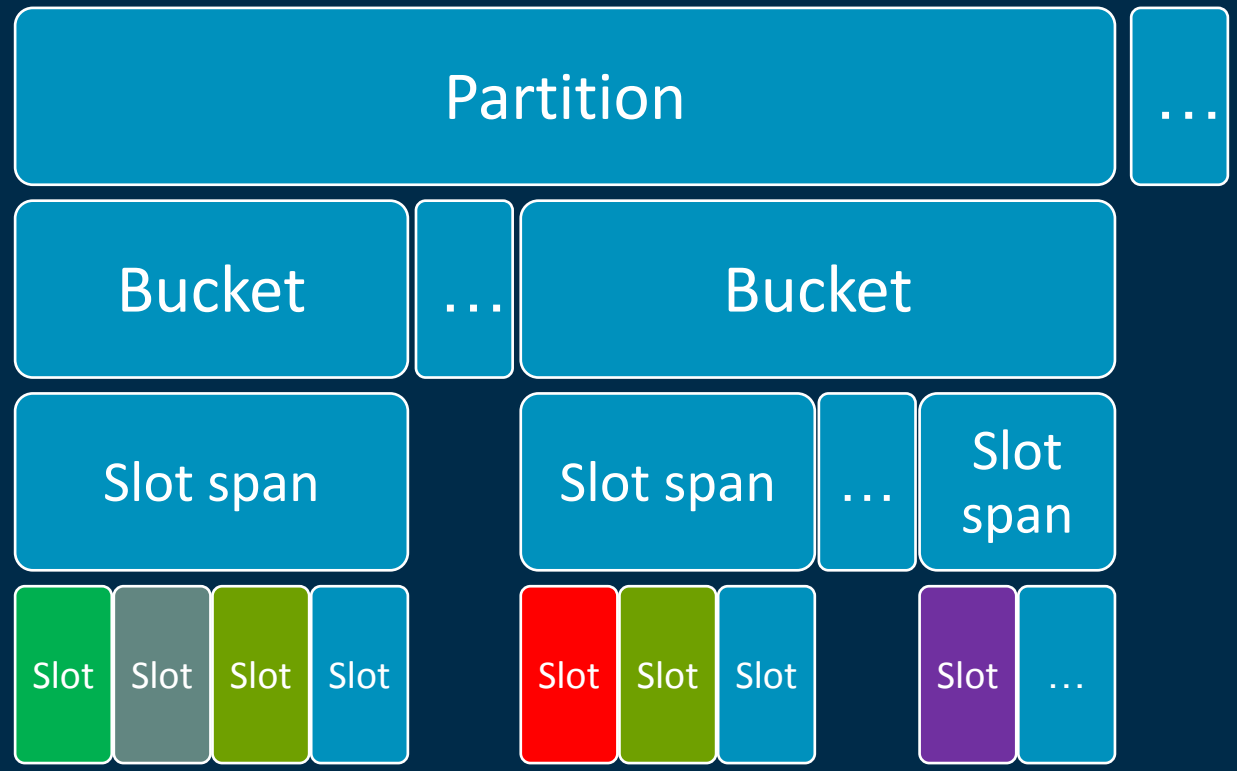
arm

How heap tagging works in the partition allocator

Apologies to bartekn@

When `0xf000000000007290` is freed, we re-tag its slot

`free(0xf000000000007290)`



arm

How heap tagging works in the partition allocator

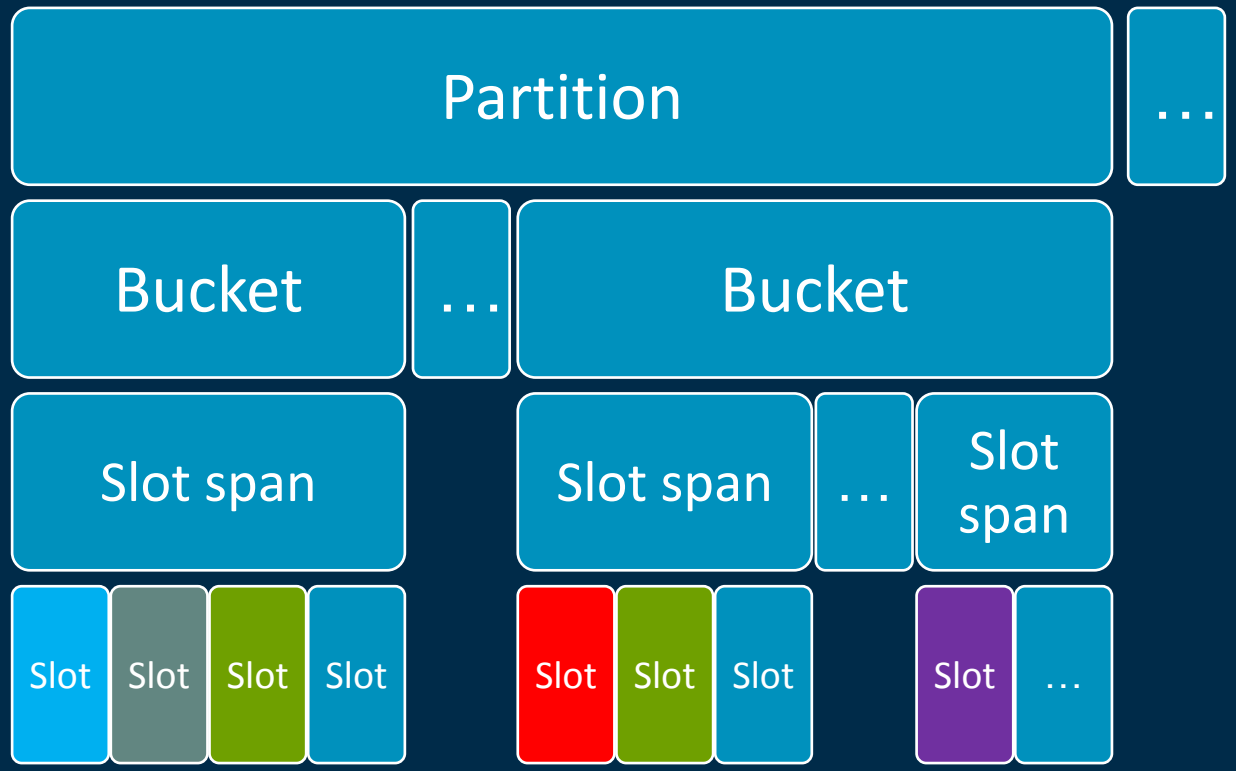
Apologies to bartekn@

Now, `0xf00000000007290` can no longer be used to access that slot

```
free(ptr); // ptr = 0xf00000000007290
*ptr = ‘h’ // Not fine, will crash
```

Same principle applies to overflows – different slots are differently tagged (mostly), so this will also not work most of the time

```
ptr = reinterpret_cast<char*>(malloc(64));
strcpy(ptr, “this is fine”);
// Not fine, will crash
strcpy(ptr, kSomeStringLongerThan64Chars);
```



arm

In Chromium: MTE is a tool for detecting bugs

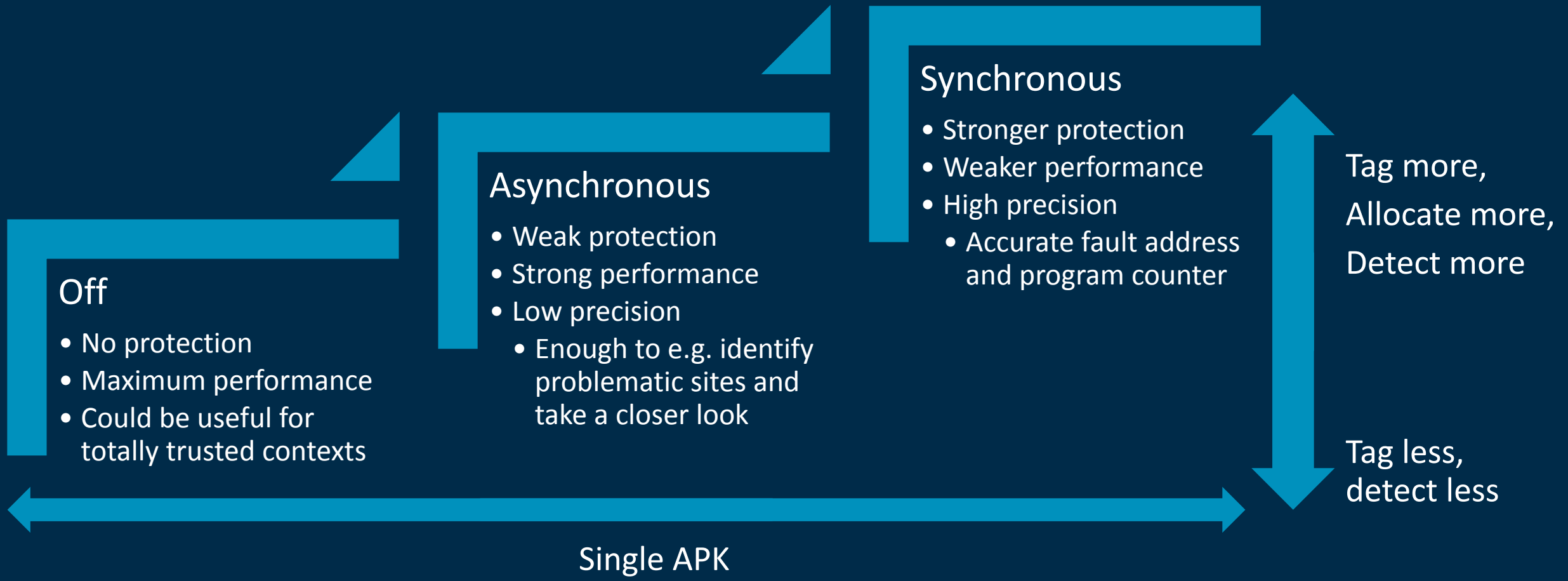
It’s basically a hardware-backed ASAN*

- We’ve taught the partition allocator about MTE such that:
  - Adjacent memory areas are tagged differently… most of the time
    - Therefore, detects buffer overflows most of the time
  - Freed memory areas are always tagged differently
    - Should detect UaF errors very frequently
- Limitations
  - Only objects less than 1500 bytes are tagged at this time
  - Can’t protect objects allocated on the stack or in globals (requires a special Armv8.5 build)
  - Needs additional work to support V8 (requires a special build)
  - Doesn't protect metadata in the partition allocator

* Unlike HWASAN though, it doesn’t need any compiler instrumentation or carve out big chunks of the memory map and all code (even assembly) is covered

Deployment: Precision versus performance

MTE offers a tuneable level of performance, protection and precision at runtime



arm

Deployment modes

	Security	MTE mode now	Later
Unit Tests	Low	Synchronous, ubiquitous	Synchronous, ubiquitous
Browser process	High	Asynchronous, ubiquitous	Synchronous, configurable
Renderer process	Medium	Asynchronous, ubiquitous	Async, configurable

Current plan: unit tests always get synchronous MTE (good for debugging and finding issues quickly)

Must start the browser APK in asynchronous mode for MTE to work correctly

- Can upgrade to stronger synchronous MTE, or switch it off

Planning to opt native unit tests into MTE via AndroidManifest.xml

Planning to opt the browser into async MTE via AndroidManifest.xml and add command line flags for per-process control.

Last year, this was our MTE plan

Experimented in 2021, will ship in 2022

2020

- Design documents
- Partition allocator experiments / prototyping
- Foundations
  - CPU feature detection
  - Low-level functions

2021

- Production-ready partition allocator implementation
- Production-ready foundations
- Android S-based test environment available
- PCScan/DCScan prototyping

When devices available

- Performance tuning
- Experiments with cppgc / other operating systems

Last year, this was our MTE plan

Experimented in 2021, will ship in 2022

2020

- Design documents — Done
- Partition allocator experiments / prototyping — Done
- Foundations — Done
  - CPU feature detection
  - Low-level functions

2021

- Production-ready partition allocator implementation — Done
- Production-ready foundations — Done
- Android S-based test environment available — Yep
- PCScan/~~DCScan~~ BackupRefPtr prototyping — In progress – tests pass
- Debug tools — In progress

When devices available

- Performance tuning
- Experiments with cppgc / other operating systems
- Fuzzing enablement — New
  - Prepare experimental stack tagging builds to find more bugs
- Explore ideas to make PCScan/BackupRefPtr more efficient with MTE — New

arm

Conclusion

- PAC/BTI – strengthen control flow integrity, almost done
- MTE – strengthens memory safety, almost done
- Questions or comments?

arm

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