arm

Morello

Architectural feature deployment in the real world

Arnaud de Grandmaison

March 30th, 2023

Who has rounded-corner windows at home ?

Why planes don’t have squared-corner windows ?

Buffer overflows

- Buffer overflows and memory safety issues have had a way too long history !
  - First documented buffer overflow dating from 1972 [Wikipedia]

- 51 years later … where are we standing ?
  - Not much has changed
  - Both Google and Microsoft recognize that 70% of the security issues in their products involve / start with a memory safety issue
  - What was a minor annoyance in the 70s is now a financial drain and a huge security / privacy issue
  - The boiling frog metaphor illustrates pretty well the issue. We need to jump out of the pan !

(Note: no frog has actually been hurt or boiled)

Memory safety overview

- Temporal aspect :
  - Use-after-free example:

```
int *tab = (int *) malloc(10 * sizeof(int));
… // do things
free(tab);
… // do other things
tab[0] = 1234; // What happens here ?
```

- Spatial aspect:
  - Buffer overflow example:

```
int a;
int tab[10];
int b;
tab[-1] = 1234; // What happens here ?
tab[10] = 4567; // What happens here ?
```

Question: any difference with the above code if data are located in stack or memory ?

MPU / MMU overview

- Both are specialized hardware unit designed to help with memory management

- MPU: Memory Protection Unit
  - Assign rights (read, write, execute, …) to range of memory locations
  - Check loads & store at execution time, raise an exception in case of error
  - Can be used to protect one task / process from another, most useful to catch programming errors.
  - Limited number of descriptors, coarse granularity

- MMU: Memory Management Unit
  - Memory Protection
  - Virtual memory management: performs address translation from virtual addresses to physical addresses
  - Used by operating systems (Linux, Windows, MacOS, …)
  - Protection at the page granularity, e.g. 4k

Capabilities to the rescue ! (maybe)

- Capabilities are essentially fat pointers, i.e. pointers with extra information
- They allow enforcing memory safety at runtime
- They exist since the very beginning of the computer industry, and have been in use for quite some time actually
  - CAP computer at the Cambridge University Computer Lab (1970)
  - System/38 from IBM (1978)
- … until they were set aside by segmentation / pagination-based memory management which was way easier and cheaper to implement back then.

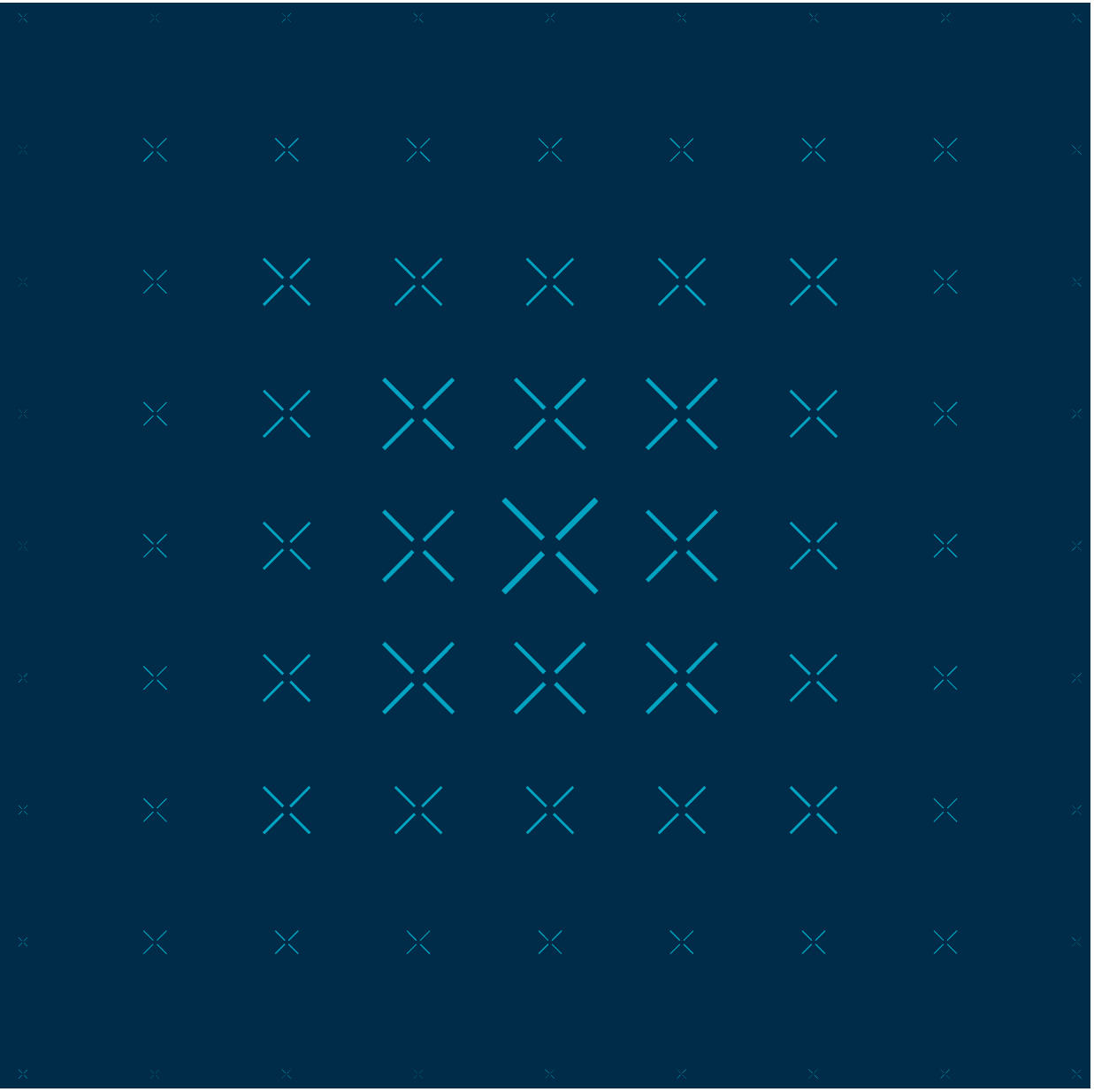
Why capabilities ?

- The capability concept has stood the test of time
- They are relatively easy to formalize
- CHERI sketches a plausible path to deployment
- A number of parties, including UK's NCSC and industry players are showing interest
- Alternatives are looking more speculative, or solve less of the problem or require more software and/or hardware resources

Lecture outline

- A CHERI overview
- The (very) big Morello picture
- A sketch of Morello architecture
- A higher-level view on Cheri / Morello
- Open questions on Morello
- Morello resources
- Misc
- Conclusion

CHERI overview



About CHERI

- CHERI: **C**apability **H**ardware **E**nhanced **R**ISC **I**nstructions

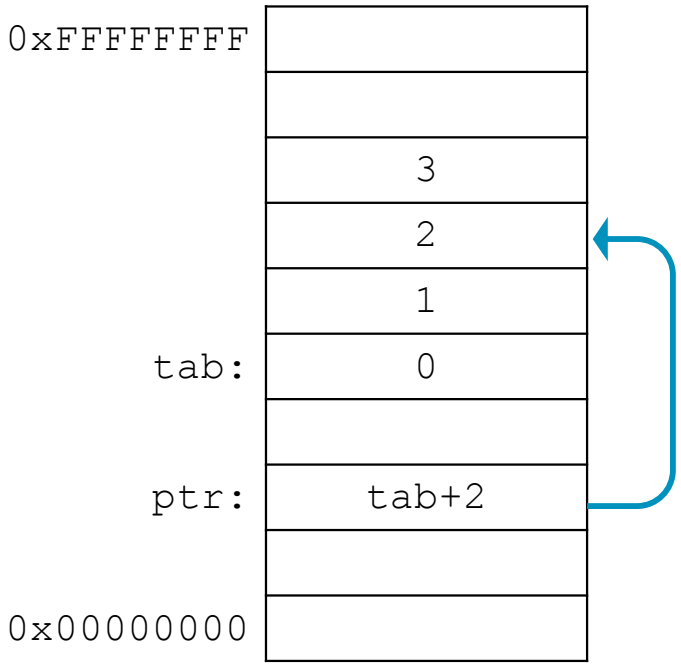
- Started ~ 12 years ago as part of project CTSRD (pronounced "*custard*") :
  - **C**lean Slate **T**rustworthy **S**ecure **R**esearch and **D**evelopment
  - **Goal : Rethinking the hardware-software interface for security**
  - A DARPA-funded project, part of DARPA CRASH programme, with Google's support
  - A joint research project of the Cambridge University Computer Laboratory and SRI International

Memory capability basics

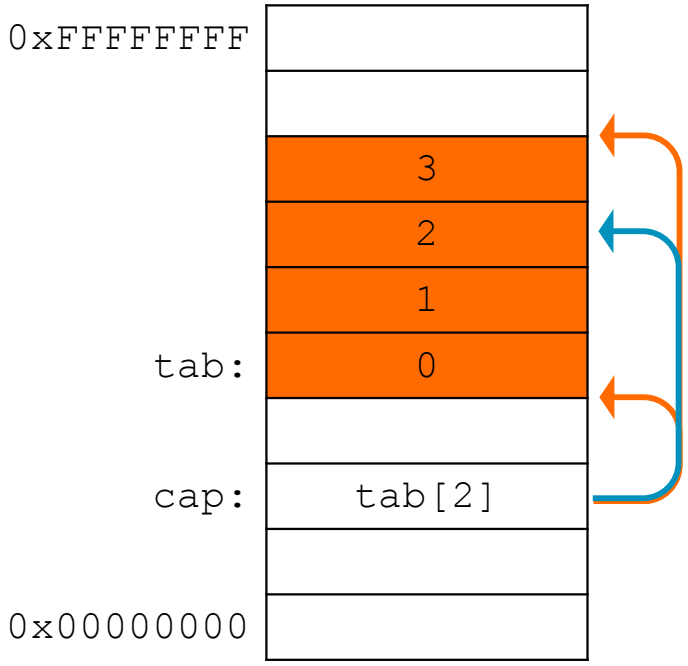
- Capabilities are used wherever a pointer / reference is used
- Capabilities include:
  - Base, pointer, and size (or limit)
  - Rights to the data pointed to : read / write / execute / …
  - Rights to the capability itself
- Instructions manipulating capabilities can only *reduce* their range and permissions (a.k.a *monotonicity*)
- Capabilities cannot be forged (a.k.a *unforgeability*)
  - Capabilities are protected by a fragile tag which is cleared when the location is written by a non-capability instruction
  - Only valid capabilities (i.e. tag is set) can be used by the capability instructions and load/store
- Capabilities do not replace MMU and paged memory
  - They go on top of it
  - Provide fine grained access policy for code and data

Visual intuition : Pointer vs. Capability

```
int tab[4] = {1, 2, 3, 4};
int *ptr = &tab[2];
```




```
int tab[4] = {1, 2, 3, 4};
int *cap = &tab[2];
```



arm

A hardware perspective on CHERI architecture

- CHERI restricts access to memory and system resources *within* a virtual address space
  - Originally (1960s, 1970s), capabilities did control access to physical memory
- CHERI replaces virtual addresses with memory capabilities that comprise
  - A virtual-address pointer component
  - A meta-data component that encodes
    - (Compressed) bounds (base and limit) on the pointer
    - Permissions to use the capability in certain ways (e.g. mutable/immutable)
    - Permissions to use the object identified by the capability in certain ways (e.g. R, W, X)
- CHERI can be embedded in any modern 64-bit host ISA
  - MIPS, Arm's A64, RISC-V, x86-64…
  - See CHERI architecture
- CHERI mostly affects the load & store part of its host ISA
  - Some additional instructions operate on capabilities themselves

arm

A software perspective on CHERI architecture

- CHERI gives *spatial safety* to programs written in memory-**un**safe languages (e.g. C/C++)
  - And good hooks for adding *temporal safety* at a cost similar to garbage collection…
  - And weak *control-flow integrity (CFI)* …
    - Similar to A64 with PAC + AUT (reverse CFI) + BTI (forward) or x86 with shadow stack and landing pads
- CHERI supports fine-grain, recursive delegation of access privileges
  - Accesses checked by hardware at hardware speed
- CHERI supports secure compartments *within* a virtual address space
  - A lighter weight alternative to compartmentalizing with OS processes
  - Compartments are a vital tool to resist security exploits
    (c.f. Thomas Dullien's *Weird machines, exploitability, and provable unexploitability* –
    his assessment implies a need for many, fine-grained compartments…)
- CHERI has formal ISA semantics
  - Formalization of *architectural security properties* is a work in progress

The (very) big Morello picture

In brief, *Morello* is…

- A specific variety of cherry
- An *instruction-set architecture* (ISA) derived from Arm's A64 and Cambridge Computer Laboratory's *Capability Hardware Enhanced RISC Instructions* (CHERI) [CHERI architecture]
- A Mobile/Server-class ASIC containing multiple Morello-enhanced Arm CPUs
  - Derived from Arm's Neoverse™ N1 derived from Cortex® A-76
- A development board containing the Morello ASIC
  - Derived from an existing, non-public, Arm development board
  - The board has the resources to boot Android and act as a low-end server

- A UK government funded project under the *Digital Security by Design* [DSbD] umbrella
  - Approximately £70m committed by government
  - More then £100m in kind committed by Morello project industrial partners

The *Morello* project will…

- Fund creating a Mobile/Server-class ASIC and 500-1,000 development boards
  - The number of boards depends on test-chip yield, funding, and component cost (all variable)
- Support evaluation of deployment options and priorities by Arm's industrial partners
  - Public support at the DSbD launch from both Microsoft and Google
- Support evaluation of different software and hardware implementation options
  - **For example**: the Morello board supports two ways to tag memory, one appropriate for Mobile (no ECC on DRAM) and one appropriate for Server (with ECC on DRAM)
- Support a broad academic research program
  - From *Computer Science* to *Social Science…*
  - From theory/proof/formal to empirical studies of large-scale software…
  - Hardware and software…

arm

Morello is not …

- Morello is not the final architecture implementation:
  - It will be the **ONLY** implementation of this prototype architecture
- Morello has ***NO COMMITMENT*** to forward / backward compatibility
  - But successful concepts are expected to become part of an ARM architecture extension and/or CHERI

arm

Why are we doing this ?

- Our aim is to:
  - Break the cyclic dependency between software and hardware…
  - Understand the cost of implementation, deployment and use of these new concepts
  - Get useful feedback before committing any variant of it to CHERI or the Arm Architecture

- Because we need:
  - Answers to performance questions for a wide range of different usage models
  - Compelling examples of Capabilities offering a security / performance improvements
    - Backed up by “Red-teams” having attacked the system and demonstrated security of the system
    - Compelling in comparison with existing deployed state of the art exploit mitigations
  - Understanding of how different languages and run-times can use capabilities
    - Not just C and C++, but also Javascript, Java, Rust, …
  - Far better understanding of how fine-grained compartmentalisation can be used and supported
  - A showcase to encourage other architectures to adopt the same concepts
  - Experience of the SoC hardware to implement systems based on the CHERI concepts

A sketch of Morello architecture

Morello capability format

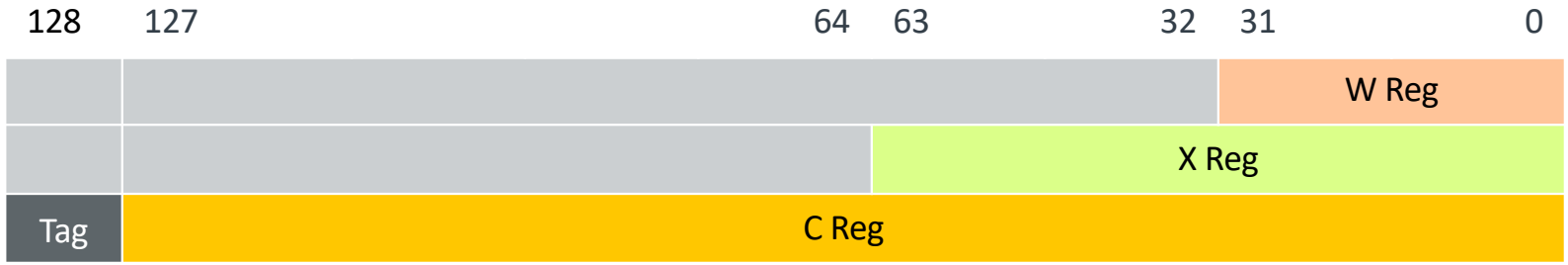
- The Morello capability format is similar *but not identical* to the CHERI-concentrate format.

- Morello capabilities have compressed bounds:
  - Vanilla CHERI capabilities are 256(+1) bits fat
    - A reasonable choice research-wise from CUCL as it allows easy exploration
    - From an industrial point of view, we believe this is way too large to be deployed
  - Morello capabilities are 128(+1) bits fat
    - Achieved using a pointer bound compression technique similar to floating point encoding

arm

Capabilities in storage and on buses

Capabilities in registers and on buses



Capabilities in memory

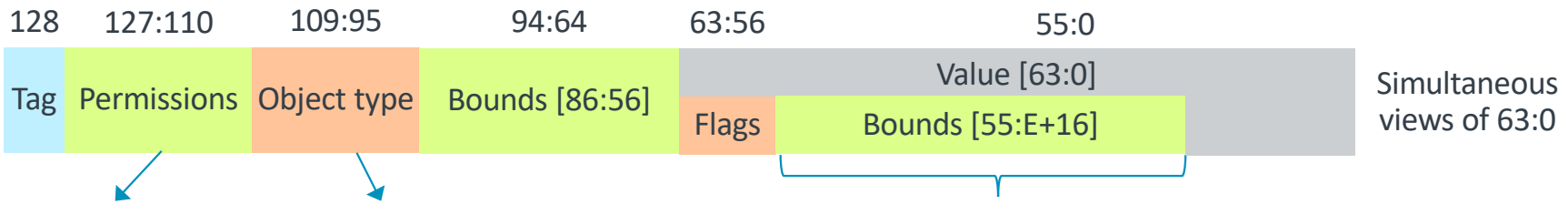
- 16-byte aligned
- A 1-bit tag is stored separately (in separate *tag memory,* or using the ECC code on the 128 bits)

Tag is *fragile*

- Preserved by specific instructions, cleared or ignored by any other access to the location
- De-referencing a capability with no tag causes a machine exception (capability fault)

arm

A Morello capability in detail

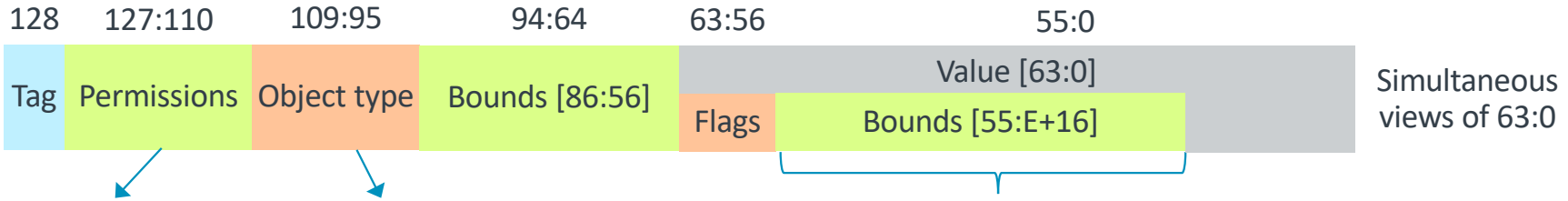


Permission
Load
Store
Execute
LoadCap
StoreCap
StoreLocalCap
Seal
Unseal
System
BranchSealedPair
CompartmentID
MutableLoad
User[4]

Object type
0: Capability is *mutable*
Non-0: Capability is *sealed*

Not all combinations of bounds and value can be represented

A Morello capability in detail



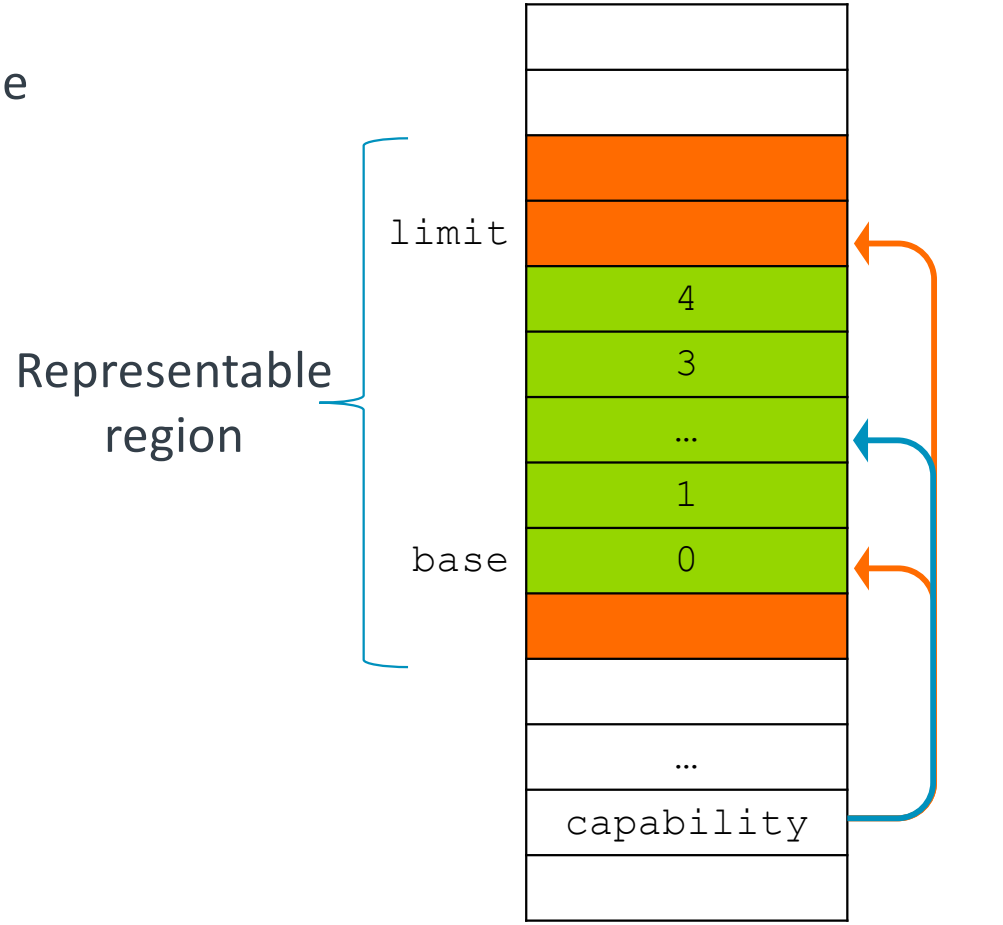
Permission
Load
Store
Execute
LoadCap
StoreCap
StoreLocalCap
Seal
Unseal
System
BranchSealedPair
CompartmentID
MutableLoad
User[4]

Object type
0: Capability is *mutable*
Non-0: Capability is *sealed*

Not all combinations of bounds and value can be represented

Bounds representability

- A capability representable region covers the range of values that are between 12.5% below the base and 25% above the limit.
- A representability check is applied when manipulating a Capability value.



Precise out of bounds is needed !

- Need to support “past-the-end” programming idiom (and all its variations)
  - Idiom largely used in software, by developpers AND compilers
- C example:

```
int tab[10];
for (int* ptr=tab; ptr<tab+10; ptr++) // tab+10 == &tab[10]
   *ptr = 0;
```

- C++ example:

```
vector<int> tab;
for (auto iter = tab.begin(); iter != tab.end(); iter++)
   *iter = 0;
```

- Need to be able to represent / compare to tab+10 / tab.end()
  - But dereferencing tab+10 / tab.end() must be an error !
- How far is precise representation needed ?
  - What about vectorization’s stride width (compiler transformations) ?

But the real world is actually more complex than this…

- What about memory allocation ?
- Weird sizes are often requested by users:

```
char *str = malloc(57);
```

- The memory allocator will probably allocate 64 bytes for your data
  - Even though 57 were requested.
  - Should the bound be set to 57 or 64 ?
- The memory allocator will probably “steal” some of (your) space.
  - Stash its own metadata, often at addresses below &str[0] (glibc)

Bounds compression – the essence of it

- See **CHERI Concentrate: Practical Compressed Capabilities** – Morello is different in detail
- Details are described in DDI0606 / A.k , section 2.5.1 Morello Bounds format

- Not all bounds values are representable, constraining how far *Value* can be taken out of bounds before it is impossible to represent the derived capability

arm

The world is still legacy !

- Most of the world’s codebase is NOT using capabilities
- Compilers can automatically use capabities instead of pointers
  - But most codebase will break
  - Compilers can not easily add extra semantic to the current language
- Rewriting / porting the code to take advantage of capabilities will take time
  - A migration / transition path is required

Morello machine states & instructions

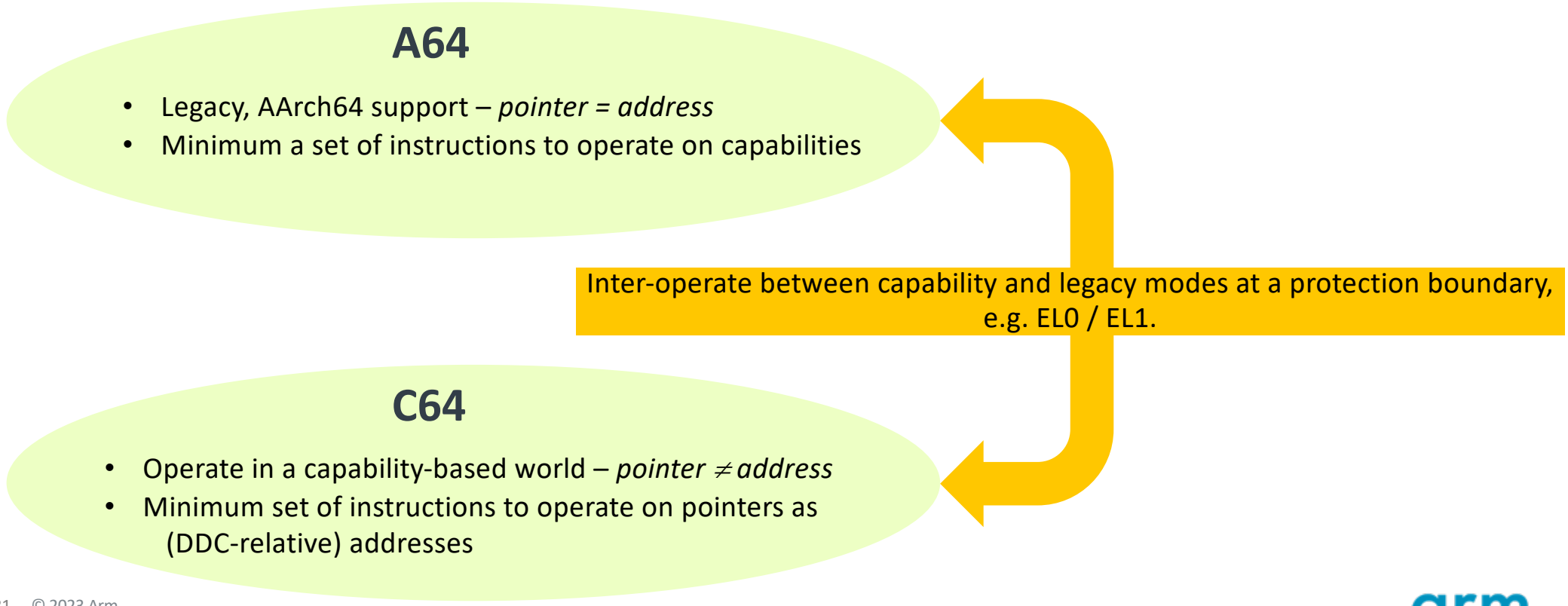
- A64:
  - Aarch64 ISA + minimum set of instructions to operate on capabilities
  - Memory accesses are address-based by default (*pointer = address*)
- C64:
  - Aarch64 ISA + minimum set of instruction to operate on pointers
  - Memory accesses are capability-based by default (*pointer = capability ≠ address*)
  - Address-based memory accesses are interpreted relative to DDC

		A64	C64
AArch64	operations	`add x0, x1, x2`	`add x0, x1, x2`
	memory	`load x0, [x1]`	`load x0, [c1]`
Extension	operations	`add c0, c1, x2`	`add c0, c1, x2`
	memory	`load x0, [c1]`	`load x0, [x1]`
		`load c0, [c1]`	`load c0, [x1]`

arm

Morello machine states & instructions --- intended use

- Same instruction encoding, but “address” interpretation depends on the machine state



A glimpse at some instructions

- Getters:

```
gclen x0, c1              ; Get length
gcperm x0, c1             ; Get permissions
```

- Setters:

```
scperm c1, c0, x2         ; Set permission (reduce only)
scbnds c1, c0, x2         ; Set bounds (reduce only)
```

- Memory accesses:

```
ldr x1, [c0, #8]          ; load
```

- Control flow:

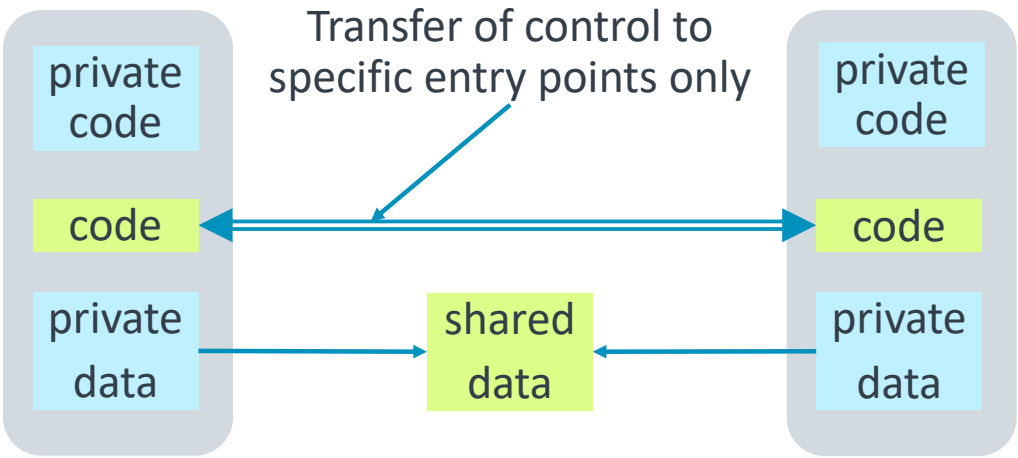
```
bl c0                     ; Branch & Link
```

A higher-level view on Cheri / Morello

Let’s take a step back…

- The tag bit essentially makes a hardware type !
  - The raw data interpretation no longer just depend on the instruction
- Benefits goes beyond memory safety.
  - Transitive closure of the root capabilities allows fine grained compartimentalization
  - In practice, (provably) perfect security is impossible to achieve. If an attacker compromises a compartment, the attacker cannot easily break out of the compartment.

Fine grained protection – Compartments



- Ideally, in any address space in any EL / security state / …
  - And as strong as MMU based compartments (VMs, processes)
- Ideally, cheap to create and destroy compartments
  - Encourage the use of lots of small compartments, reduce the attack surface for all types of exploits
- Ideally, supporting different trust models
  - At least symmetric and asymmetric distrust…

arm

Compartments – the essence of it

- The memory image of a compartment is all the memory transitively reachable from the root capabilities it was started with.

- Transfer of control between compartments requires:
  - Atomically (from the perspective of code executing at this exception level) jumping to an entry (resp. resume) point in the other compartment
  - Swapping the memory context so that the memory from the first compartment is inaccessible (other than transferring control back to the origin, or thru capabilities passed explicitly as arguments) when entering the second compartment.

Architectural challenges and opportunities

- Instructions to transfer control between compartments (security domains) give opportunities to purge or limit speculative state and the opportunity to exploit it
  - [CHERI architecture] states: *In order to achieve compartmentalization, and not simply isolation, CHERI's selective nonmonotonic mechanisms can be used*:
    - exception handling
    - jump-based invocation
  - CHERI also defines a *compartment ID* register that hardware can monitor
  - Whatever the mechanism, and whatever the distrust relationship between compartments, natural, identified points in the instruction stream are needed to tame speculative execution attacks between security domains
    - Without first-class compartments, the only mitigation hook is a *speculation barrier* instruction, resembling those introduced in v8.5 of Arm's 64-bit architecture
  - It remains micro-architecturally challenging to effectively use this architectural opportunity !

arm

What can / could be done better with capabilities ?

- Hardware enforced `const` qualifier:

```
void function(const Object *p)
```

- Give only access to a smaller range of memory:

```
struct S { /* some fields */ };
void f(struct S *p);
struct S *tab = malloc(10 * sizeof(struct S));
for(int i=0; i<10; i++)
    f(&tab[i]);         // f can not access outside the object it was given !
```

- Compartment unsafe libraries / untrusted code:

```
libjpeg_decode(&MyShip, "~/Download/alien.jpeg"); // No one will hear you scream
```

- Compartment safe / trusted code:

```
Err = EverCrypt_AEAD_encrypt(…);
```

- Garbage collection

Compilation and language issues

- C/C++ fundamental assumption that pointer = address = integer
- Pointer provenance
- `*cpy` / `*move` functionality needs extra care to preserve tags
- Code generation tactics (e.g. for managing stack frames) may make material differences to resisting the first essential step in an exploit chain that breaks memory safety
  - This is how 2/3 of today's exploits begin (according to Google and Microsoft)

Experience return ☺

- We have recompiled many large code bases. Most had to be “fixed”…
  - Most of the world’s software has migrated to 64bits pointers, so moving to 128bits pointers will be easy because they have learned the lesson, haven’t they ?
  - Lots of code bases built for CHERI exhibited out of bounds access…
  - People are doing ~~smart~~ horrible things with pointers
  - The good point is … this shows right away that CHERI is useful !

- Some applications play fast and loose with pointer provenance…
  - See, for example, Exploring C Semantics and Pointer Provenance

- The above led to some innovative research into what programmers believe about C
  - See, for example, Into the depths of C: elaborating the de facto standards

Open questions on Morello

Are all problem solved then ?

- No !
- ABI impacts ?
- CHERI allows plain old pointers to coexist with fat pointers, in order to ease the migration… But how to handle that ?
- Linking applications is becoming even more interesting...
- Loading applications and “seeding” the capabilities…
- This all requires changes in the OS / libc / ld.so / debuggers / traces / …
- People distributing full environment (Android, …) need to be involved as they will have to manage a transition. On the other hand, they have a strong incentive to improve security.
- Memory safety is only a part of the security problem --- a significant one though

Open questions

- To take full advantage of capabilities, software must be rewritten
  - Is it still worth rewriting it in C/C++ or should it be done with Rust (for example) ?
  - Can Rust (or its competitors) benefit from capabilities ?
- How to analyze and automate compartmentalization of large existing code bases ?
- (How) do proven components compose ?
  - For example, is SeL4 + CHERI + Morello secure ?
  - What does “secure” mean in this context ?
  - How to prove it ?
- Side channels (timing, EM, power) & fault injection
- Are there single points of failure in the architecture ?
  - That is, failure points independent of micro-architectural implementation
  - Or is it all about the micro-architecture ?

Morello resources

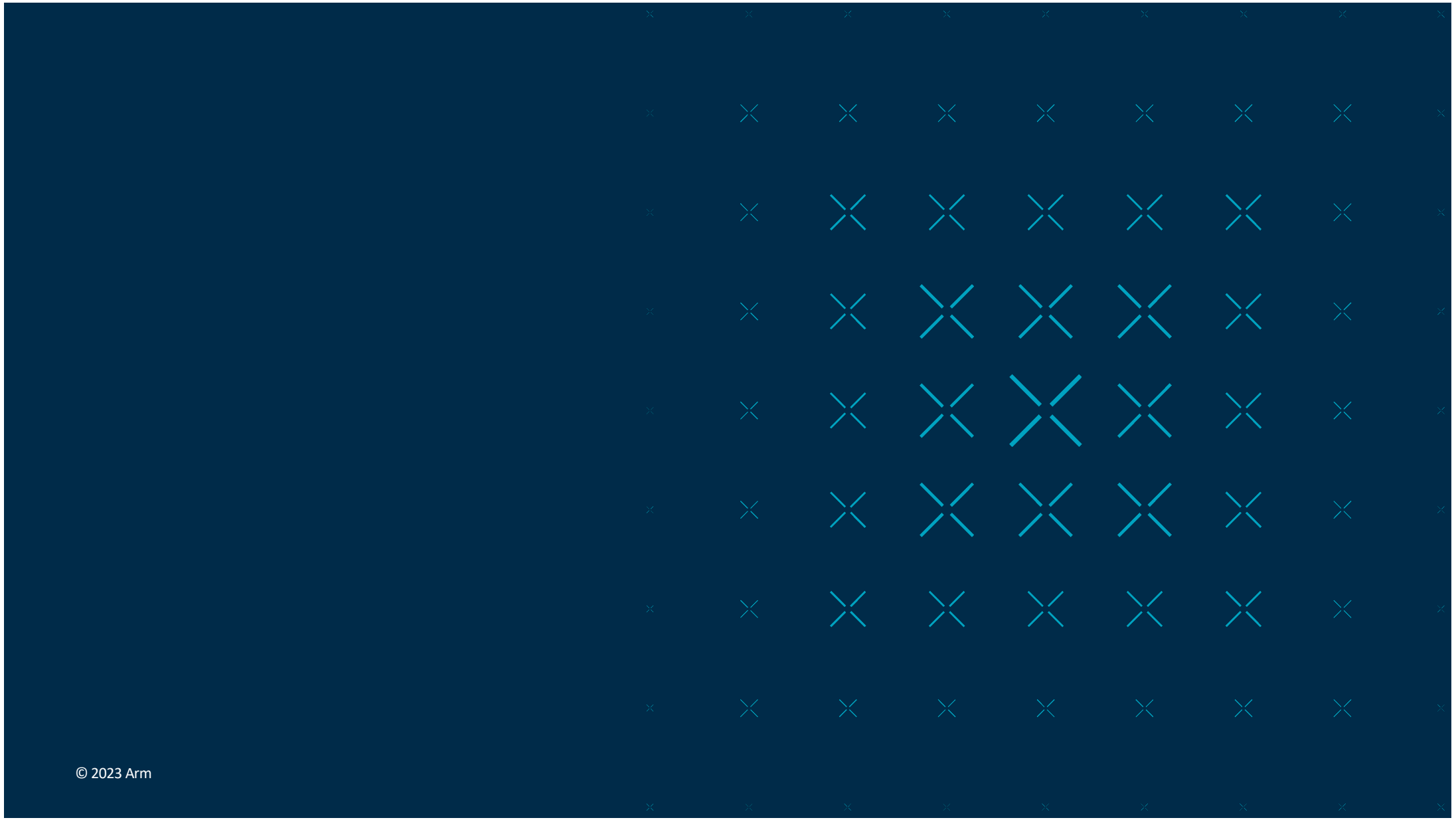
Resources

- https://www.morello-project.org/ Morello open-source software, including core repositories
- https://www.arm.com/architecture/cpu/morello : Architecture specifications, platform model, technical reference manual, …
  - Morello Platform Model (FVP) and Morello Instruction Emulator (IE)
  - Toolchains (compilers, debuggers, runtime libraries): LLVM and GCC
  - Arm Development Studio, Morello Edition
- Available environments:
  - Bare-metal enablement
  - Android / Linux
  - CheriBSD (hosted and maintained by CUCL)
- https://github.com/herd/herdtools7 : Memory Model Tools support

Misc.

Misc notes:

- What's the smallest variety of CHERI? (MSR, Risc-V)
  - https://msrc-blog.microsoft.com/2022/09/06/whats-the-smallest-variety-of-cheri/
- FlexCap: Exploring Hardware Capabilities in Unikernels and Flexible Isolation OSes
  - https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/X015610/1



Conclusion

- Architectural features are useful to help solving a problem
- But deploying them in a complete system is actually the main problem:
  - Cost
  - Performance
  - Debug
  - Language and compiler support
  - OS support
  - Existing code bases
  - New tools

arm

arm

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