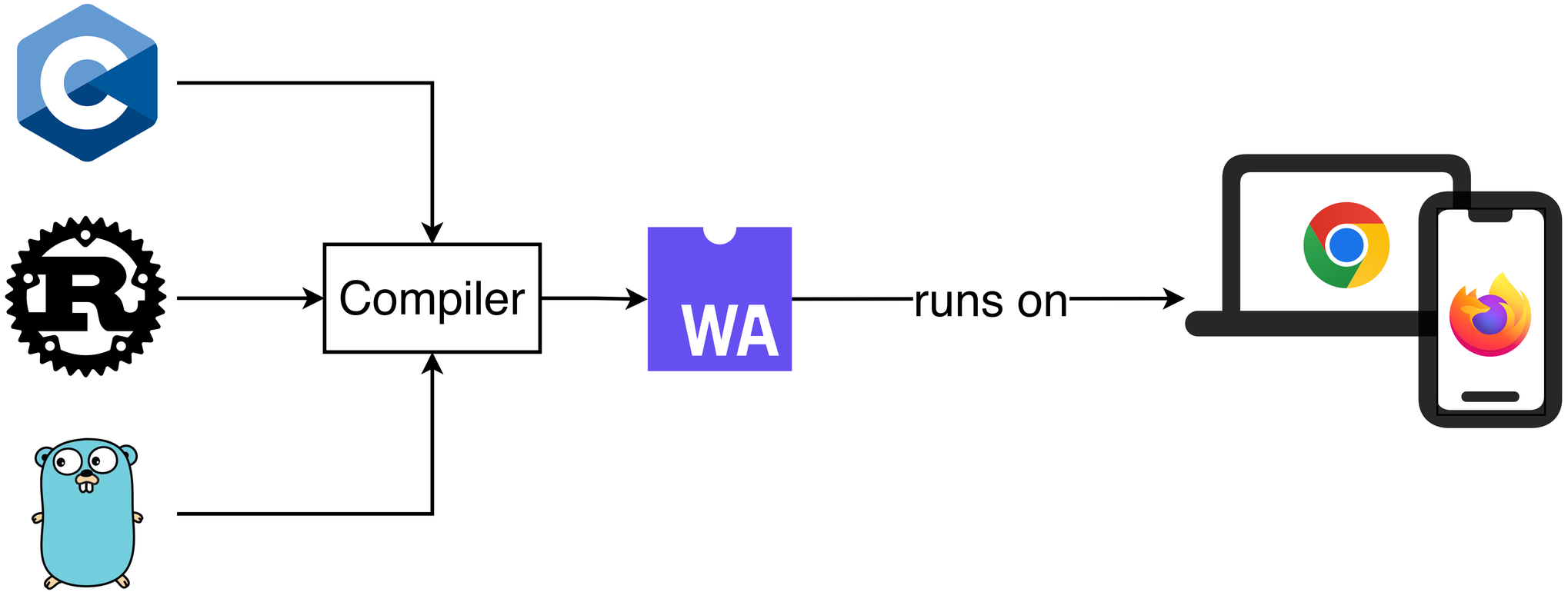
Hardware-Assisted Memory Safety for WebAssembly

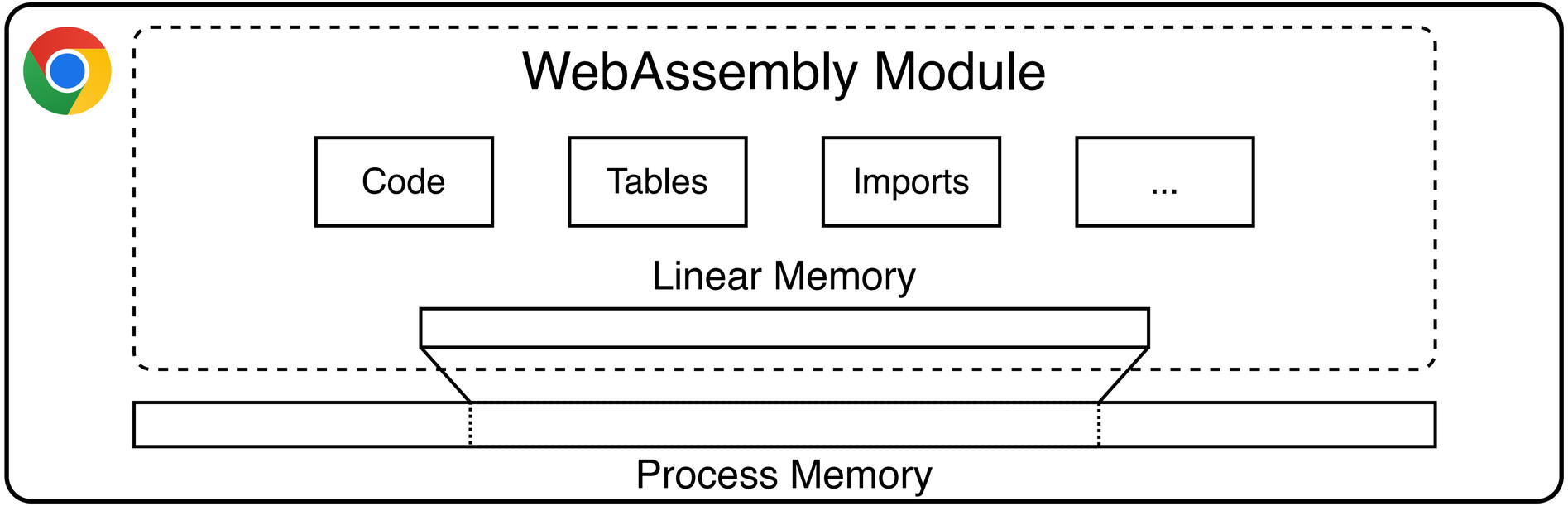
Technical University of Munich
Chair of Computer Systems

Martin Fink <martin.fink@cit.tum.de>

What is WebAssembly?

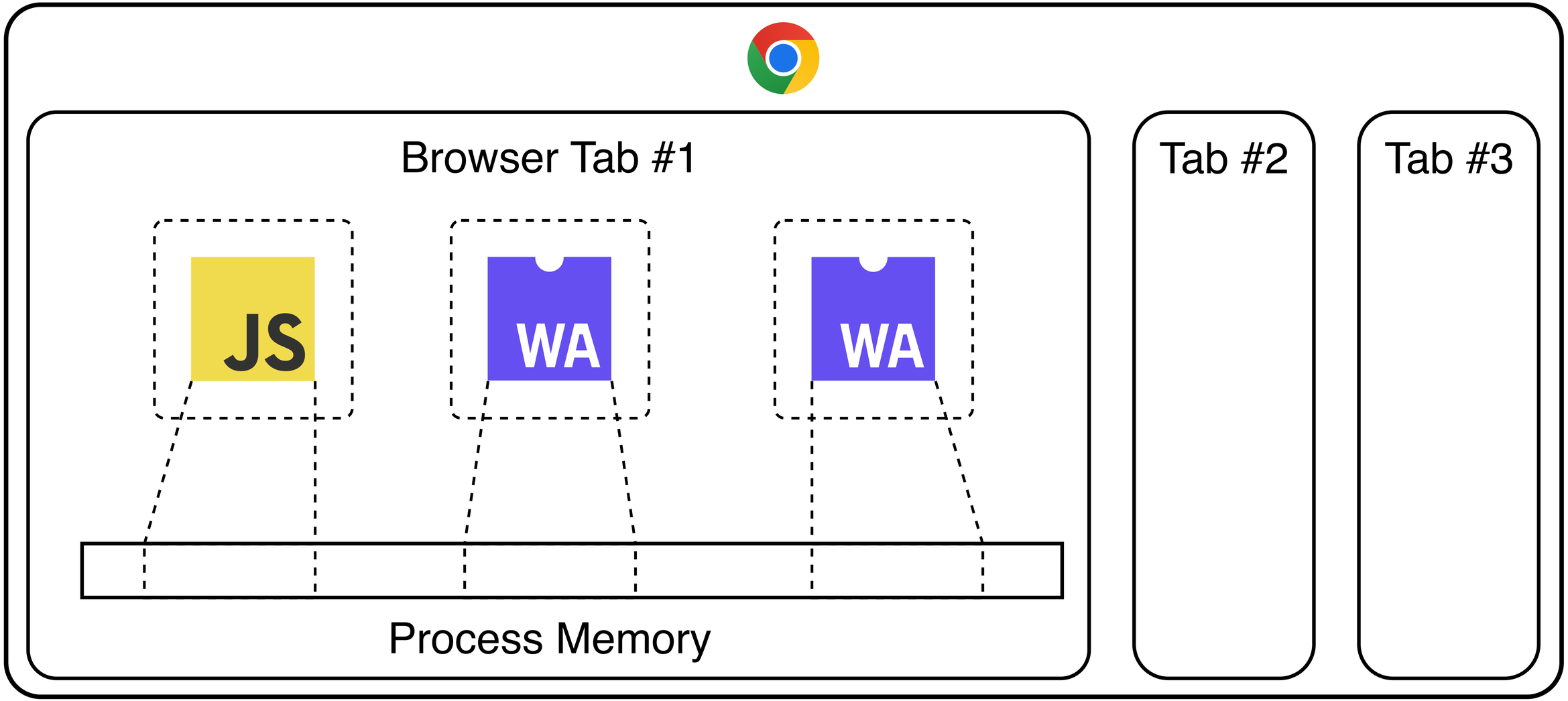


How does it work?



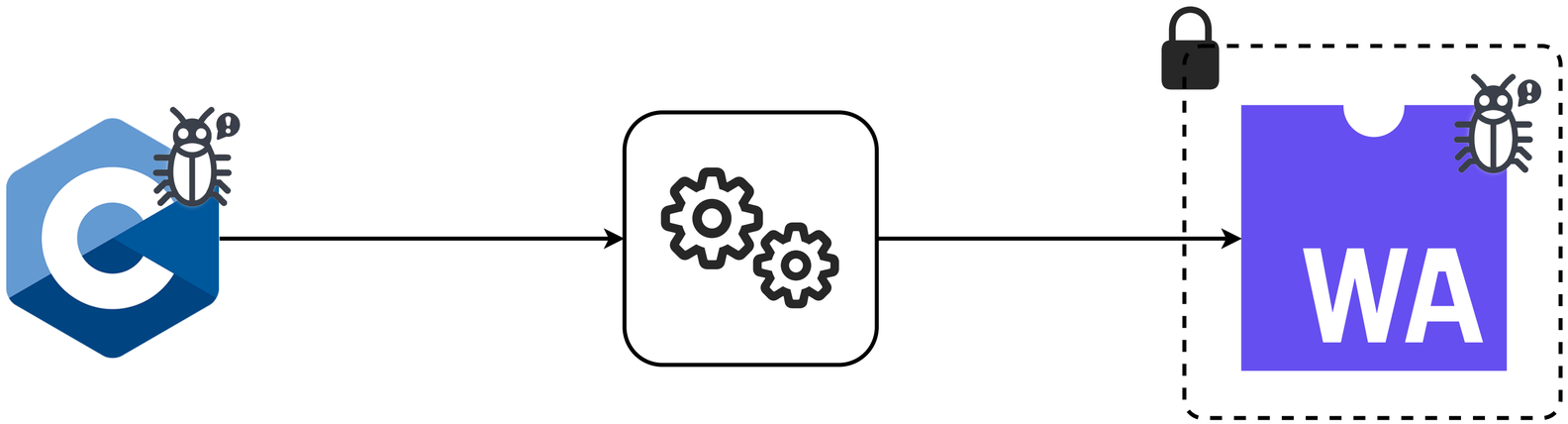
- Stack-based virtual machine
- Linear memory model
- Independent from source language and target architecture

Sandboxing in WebAssembly



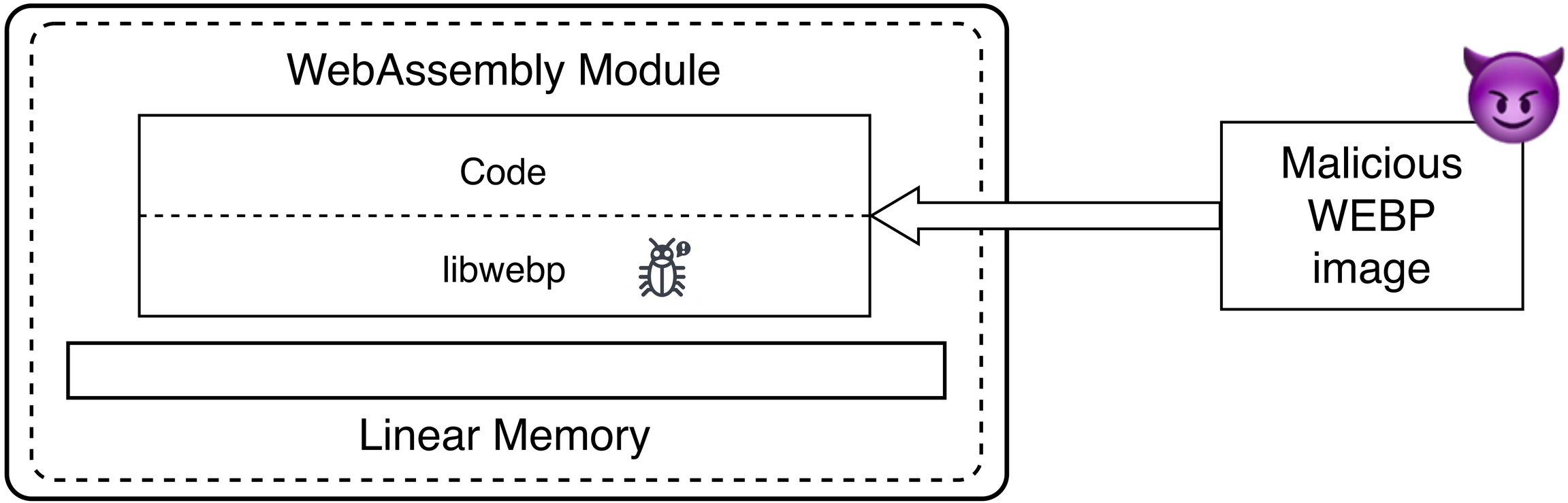
WebAssembly

- Language- and target-agnostic
- Efficient compilation
- Sandboxed execution in and outside of browsers
- Does **not** prevent classical memory safety issues

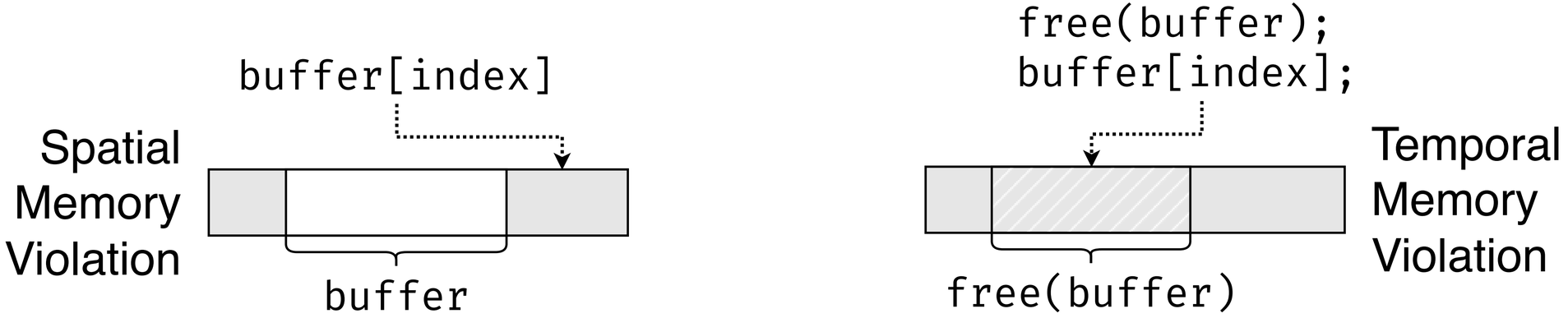


Security Issue: A Real-World Vulnerability

- CVE-2023-4863: Heap buffer overflow in libwebp (memory safety issue)
- Buggy library can be exploited



Memory Safety Issues



Chromium

- **70%** of vulnerabilities are memory safety bugs [1]

Microsoft

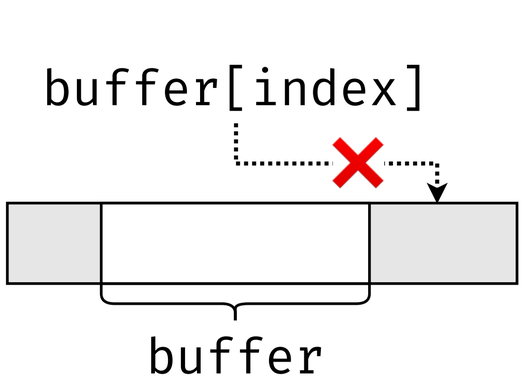
- **70%** of vulnerabilities in security patches are memory safety violations [2]

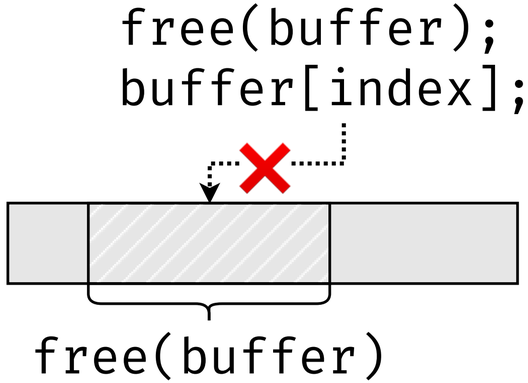
Android

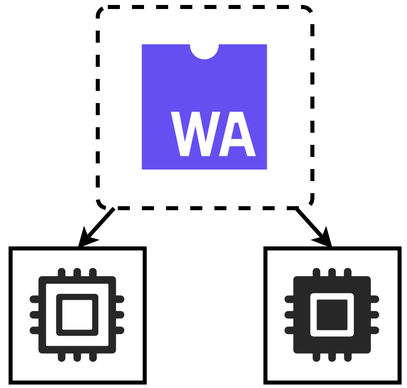
- **75%** of vulnerabilities are memory safety issues [3]

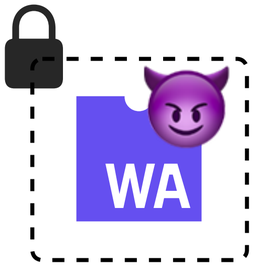
[1] Chromium project: https://www.chromium.org/Home/chromium-security/memory-safety
[2] Microsoft: https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/
[3] Android: https://security.googleblog.com/2019/05/queue-hardening-enhancements.html

Memory Safety for WebAssembly









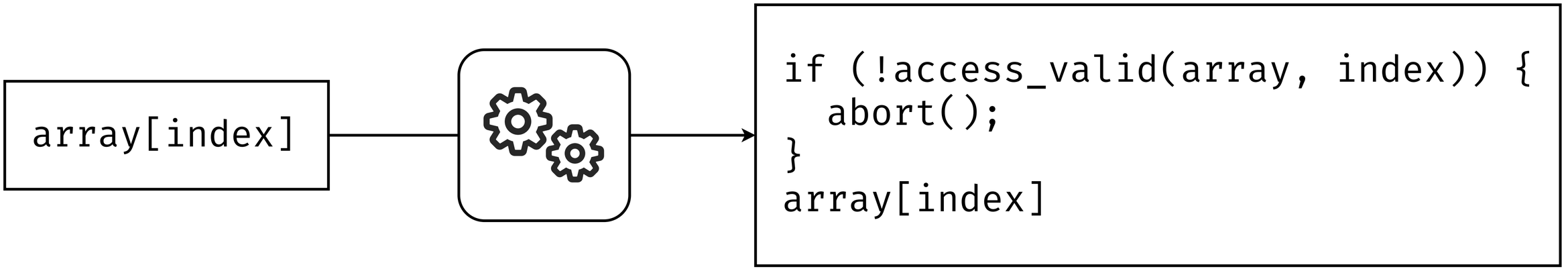
Goals

- **Memory Safety**: spatial and temporal
- **Transparency**: no modification to existing code
- **Portability**: hardware-independent abstraction
- **Security**: WebAssembly modules might be adversarial

Performance

Memory Safety for WebAssembly

Software-Based Approach: Deterministic Bounds Checking



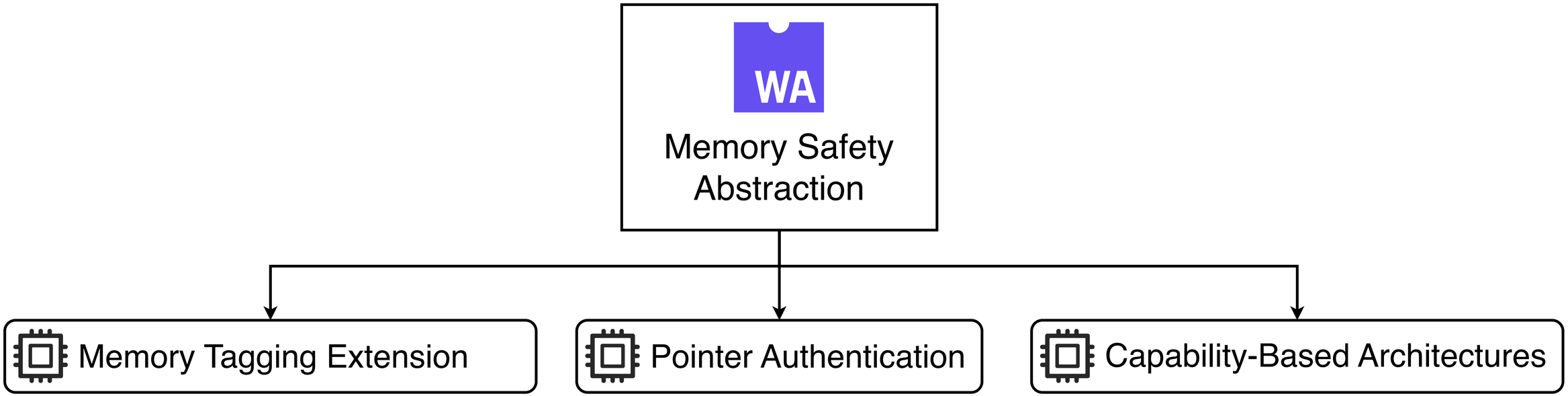
- Significant overheads prevent production deployment
- **Address Sanitizer**: Average slowdown of **73%** [4]

[4] Serebryany, Konstantin, et al. "AddressSanitizer: A fast address sanity checker." 2012 USENIX annual technical conference (USENIX ATC 12). 2012

Hardware Assisted Memory Safety

Hardware Extensions to the Rescue!

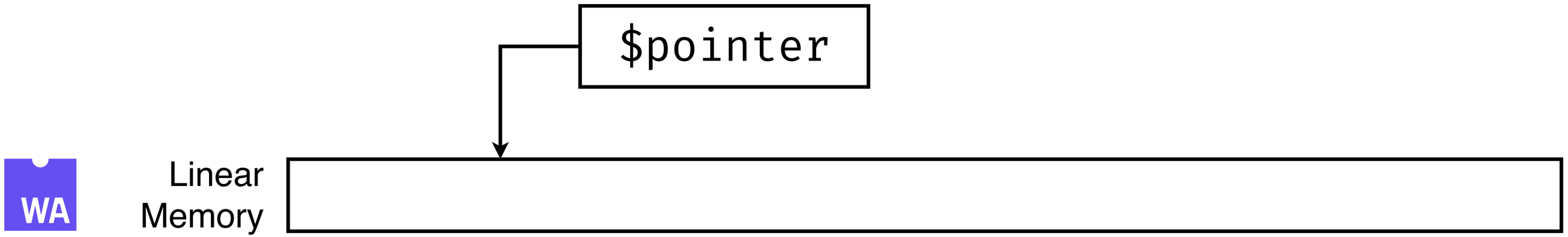
- Explore **different hardware extensions** with different tradeoffs
- **General** design



Hardware Assisted Memory Safety

Hardware-adaptable abstraction

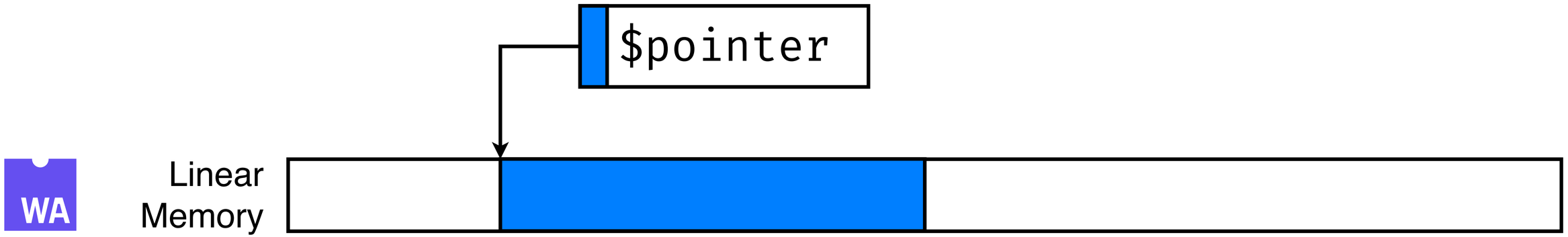
```
char *pointer = malloc(32);
```



Hardware Assisted Memory Safety

Memory Segments and Tagged Pointers

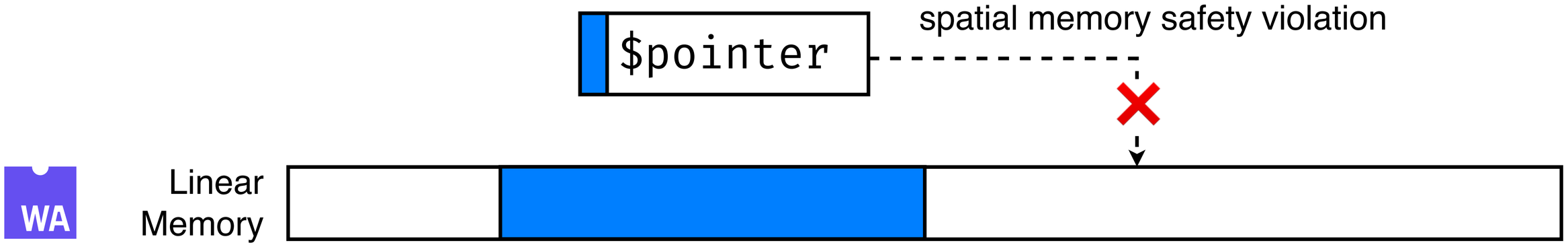
```
char *pointer = malloc(32);
```



Hardware Assisted Memory Safety

Spatial Memory Safety Violation

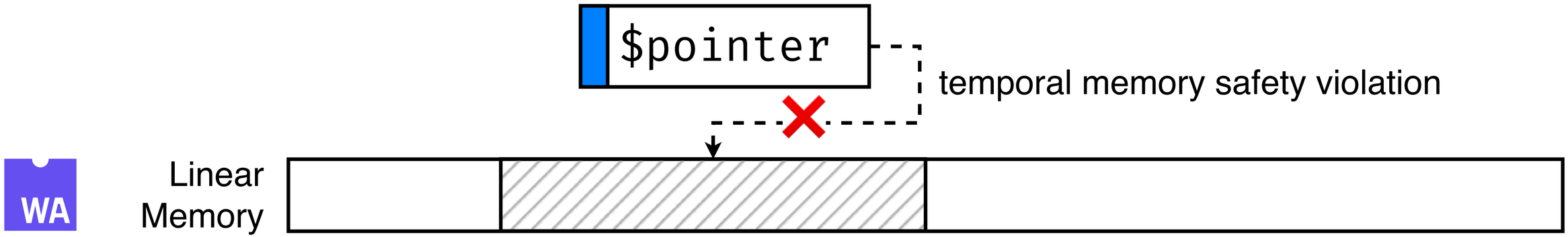
```
char *pointer = malloc(32);
pointer[40];
```



Hardware Assisted Memory Safety

Temporal Memory Safety Violation

```
char *pointer = malloc(32);
free(pointer);
pointer[24];
```

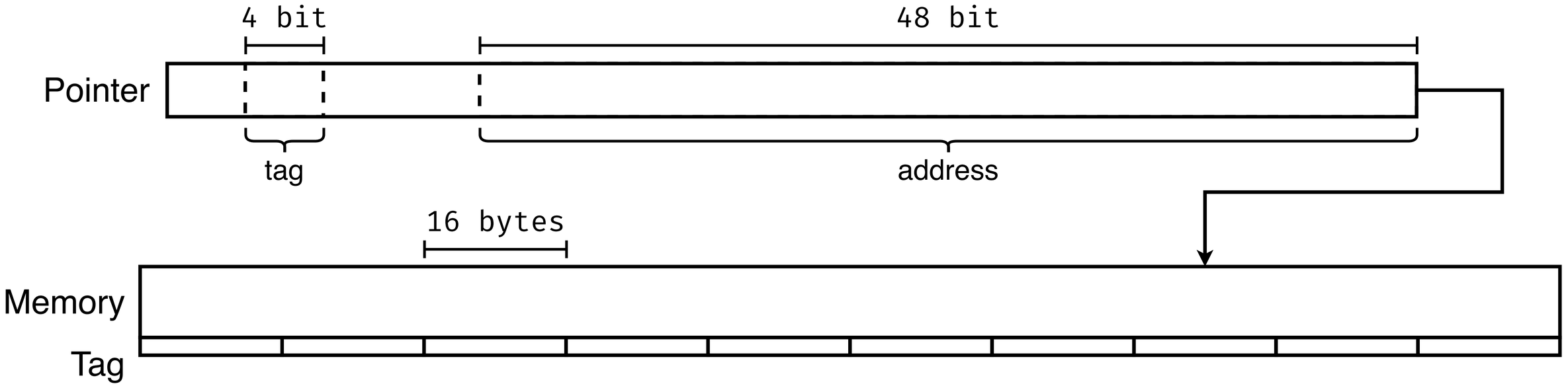


Challenges

- **Tagged pointer representation**: Metadata-carrying pointers that work on all platforms
  - Opaque and adaptable pointer representation
- **Transparency**: Correct and transparent compilation from unmodified source code regardless of underlying hardware platform
  - LLVM sanitizer passes
  - Modification of standard library
- **Performance**: Eliminate software bounds checks
  - Rely on hardware extensions (MTE, CHERI, PAC, …)

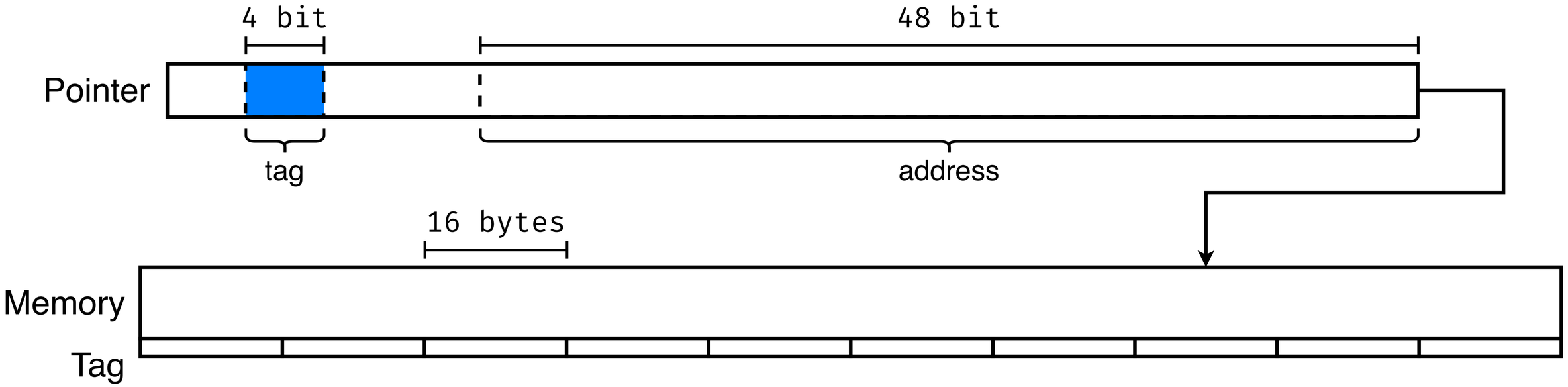
ARM Memory Tagging Extension (MTE)

- 4 bit tag in unused address bits
- 16 byte granularity
- Tag mismatch is caught by hardware



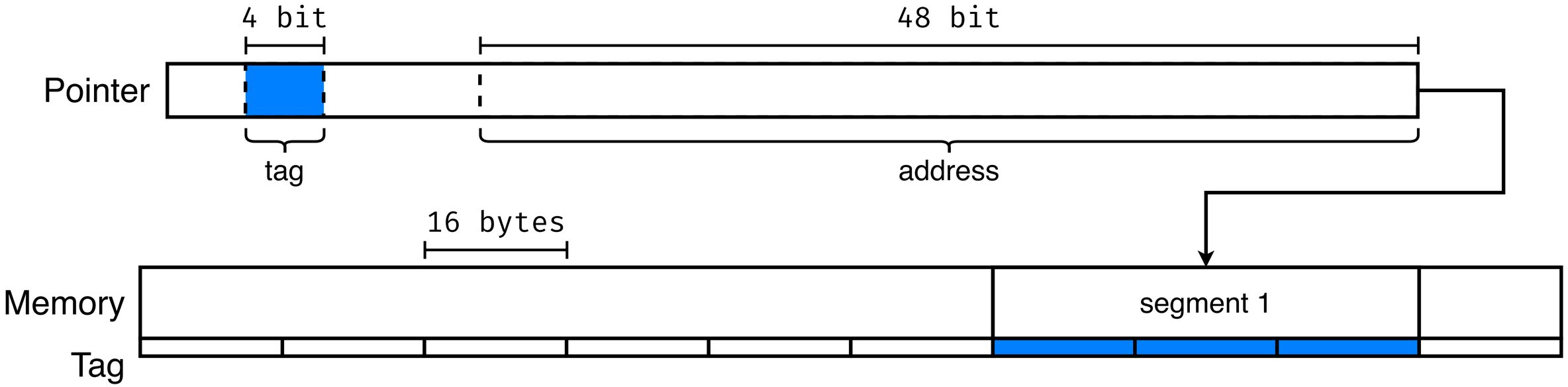
ARM Memory Tagging Extension (MTE)

- 4 bit tag in unused address bits
- 16 byte granularity
- Tag mismatch is caught by hardware



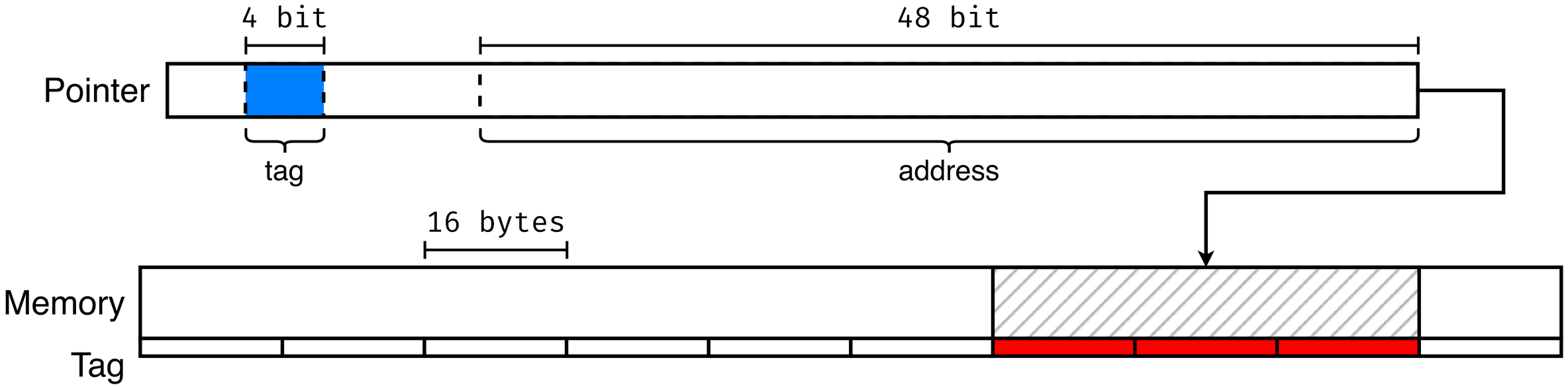
ARM Memory Tagging Extension (MTE)

- 4 bit tag in unused address bits
- 16 byte granularity
- Tag mismatch is caught by hardware



ARM Memory Tagging Extension (MTE)

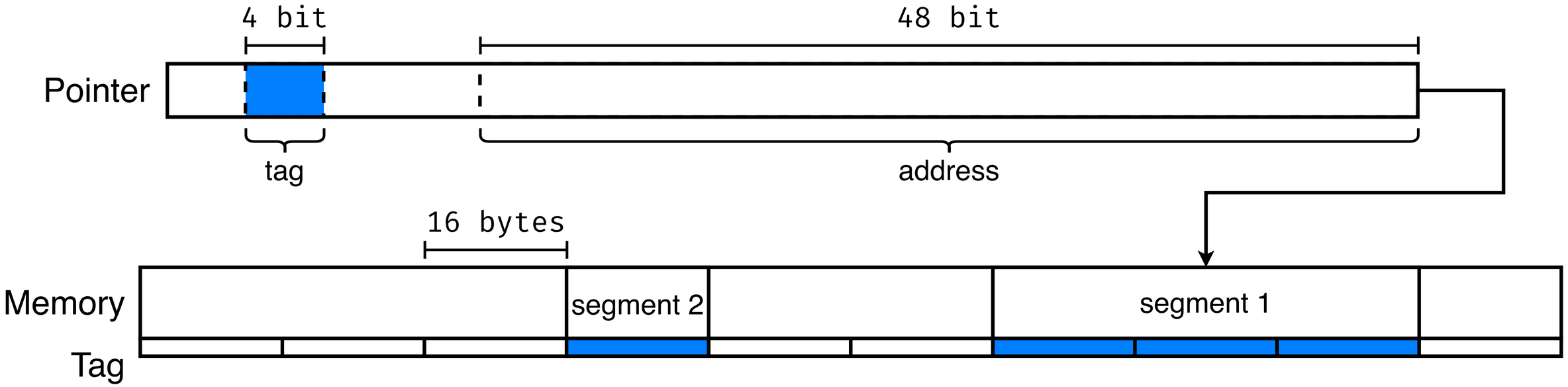
- 4 bit tag in unused address bits
- 16 byte granularity
- Tag mismatch is caught by hardware



ARM Memory Tagging Extension (MTE)

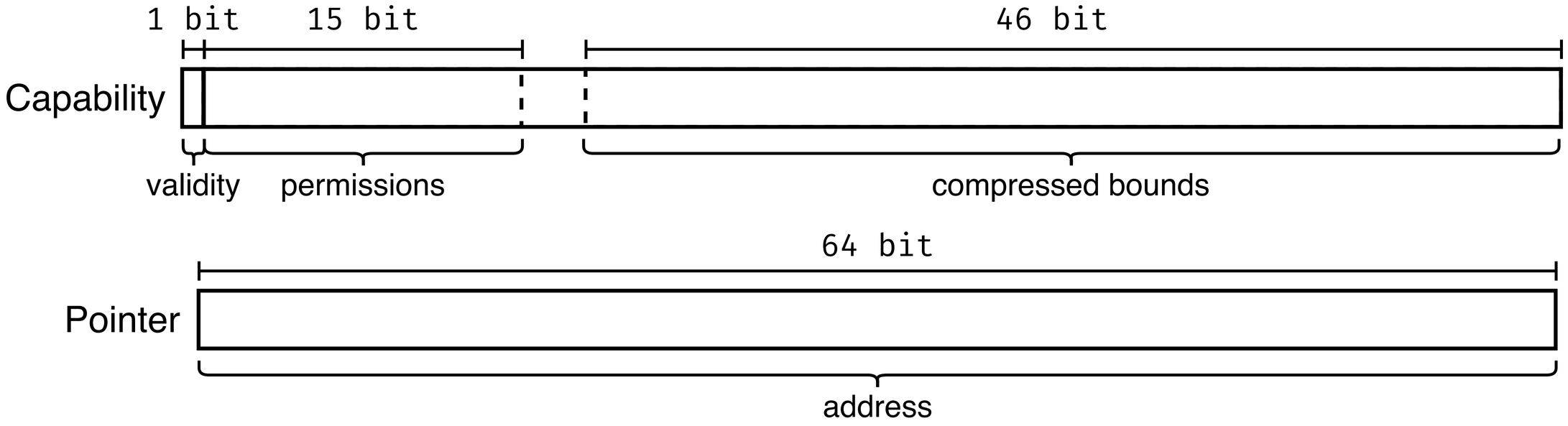
Tradeoffs

- Probabilistic Memory Safety
  - 16 distinct tags → tag collisions



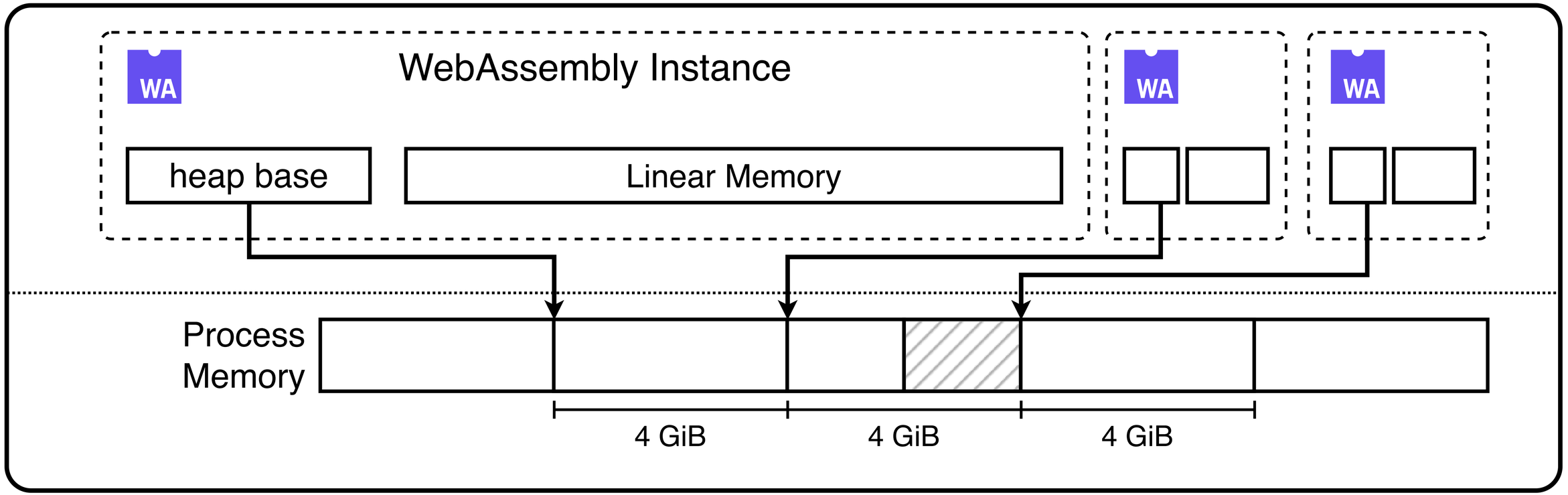
Capability-Based Architecture: CHERI

- 128 + 1 bit pointer
- Deterministic checks
- Research-architecture, implemented in ARM Morello as extension to Armv8-A



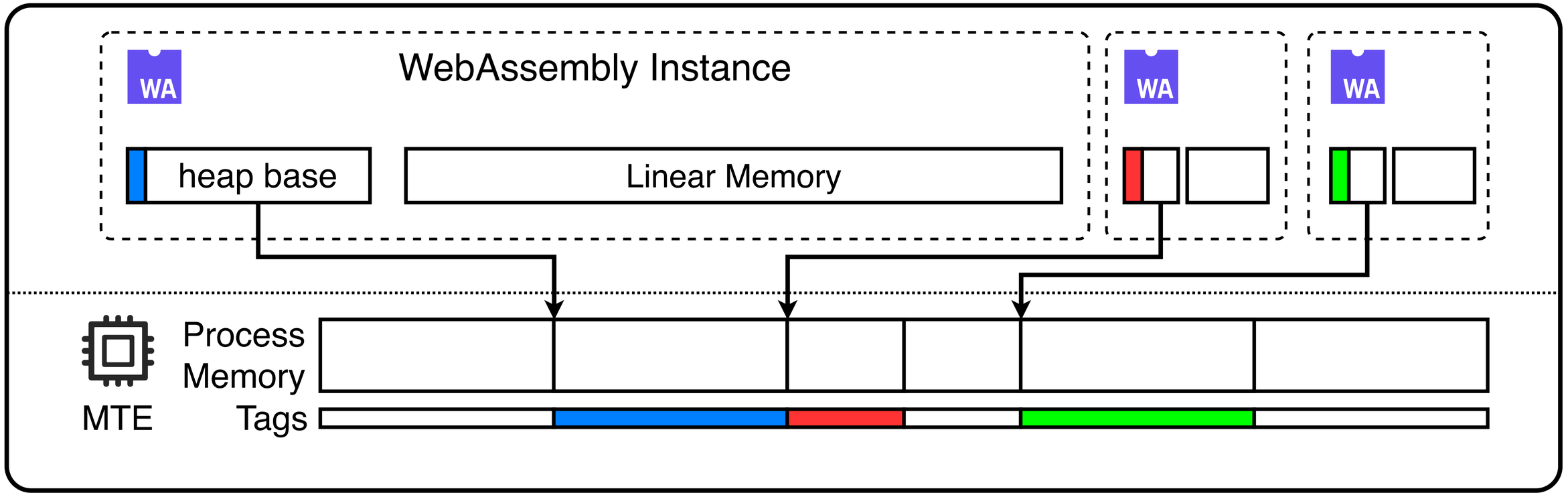
WebAssembly Sandboxing

- Sandboxing using **guard pages**
- Allocate 2^{32} = 4 GiB of virtual memory per sandbox
- Only possible for 32 bit WebAssembly

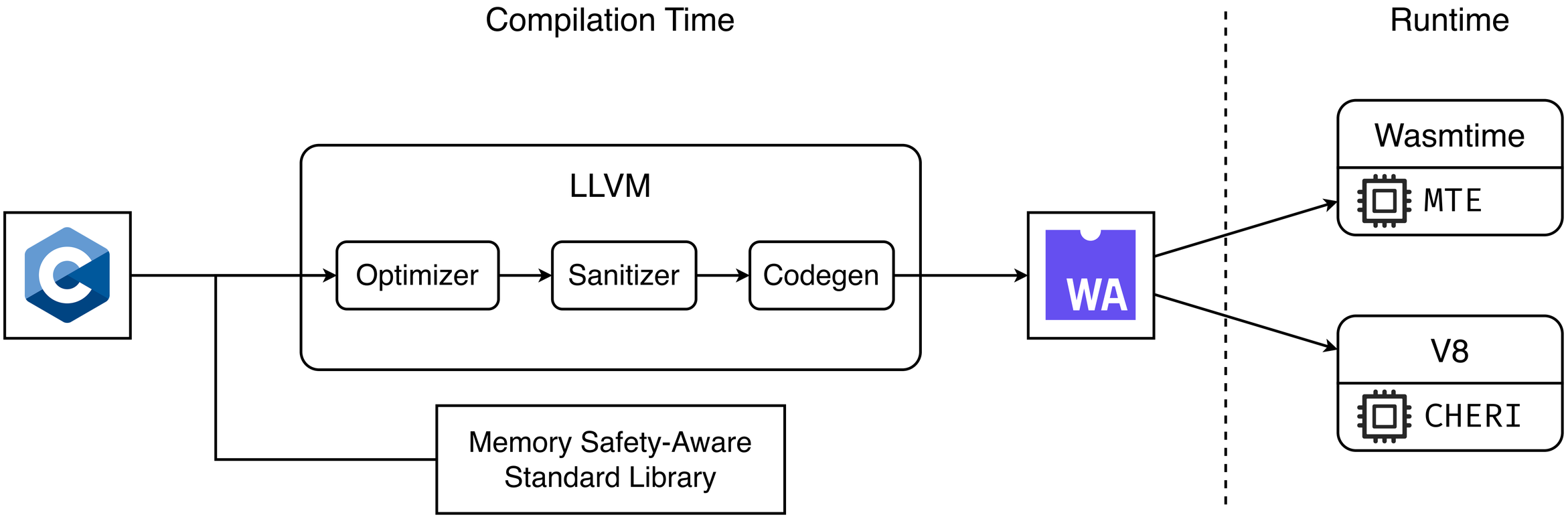


Sandboxing using MTE

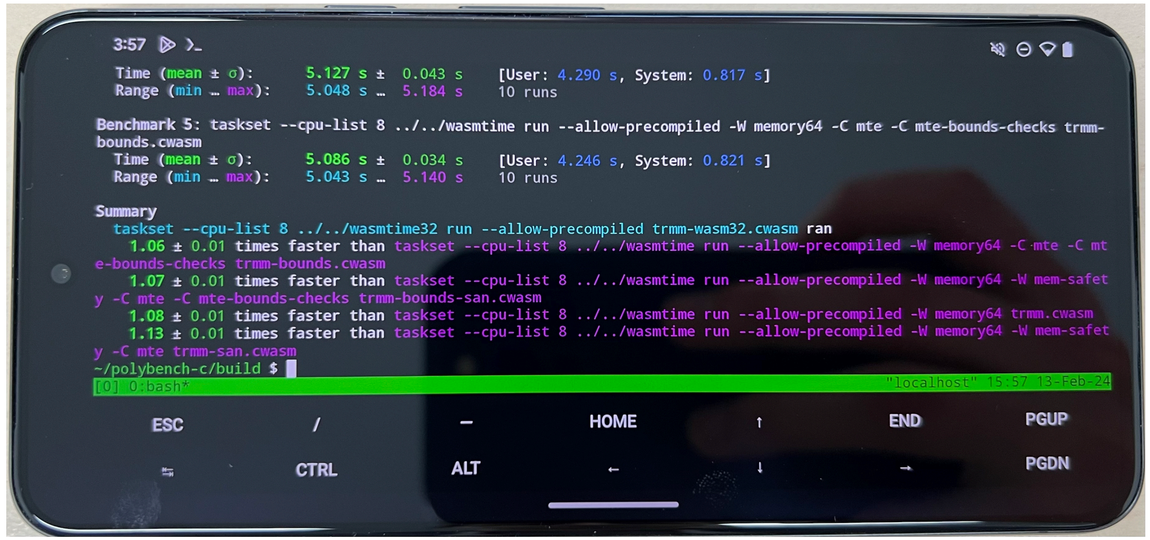
- Assign distinct tag for each sandbox
- Perform access relative to tagged base pointer



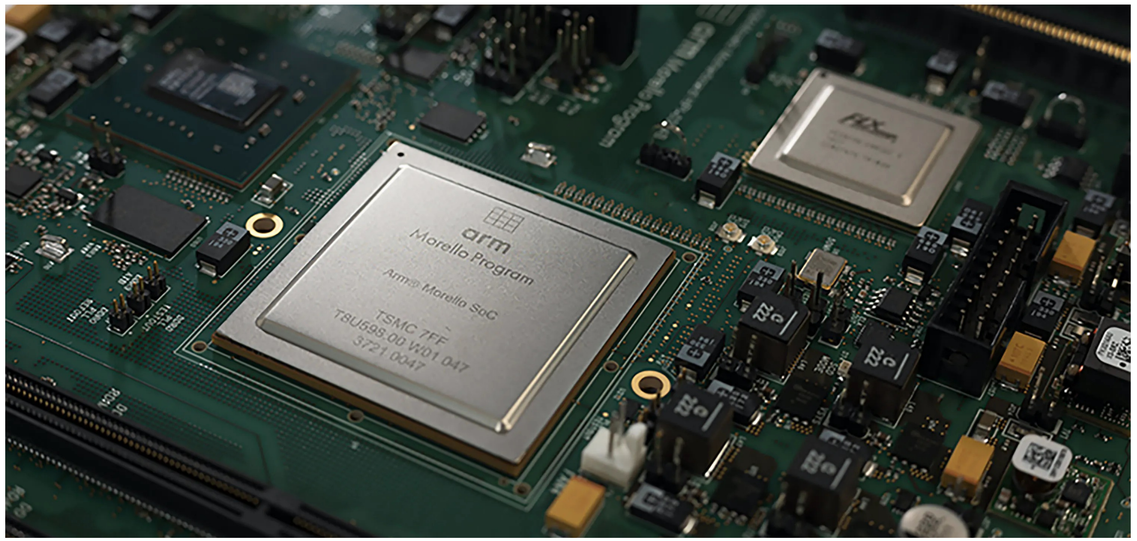
Implementation



Implementation



Pixel 8 with MTE

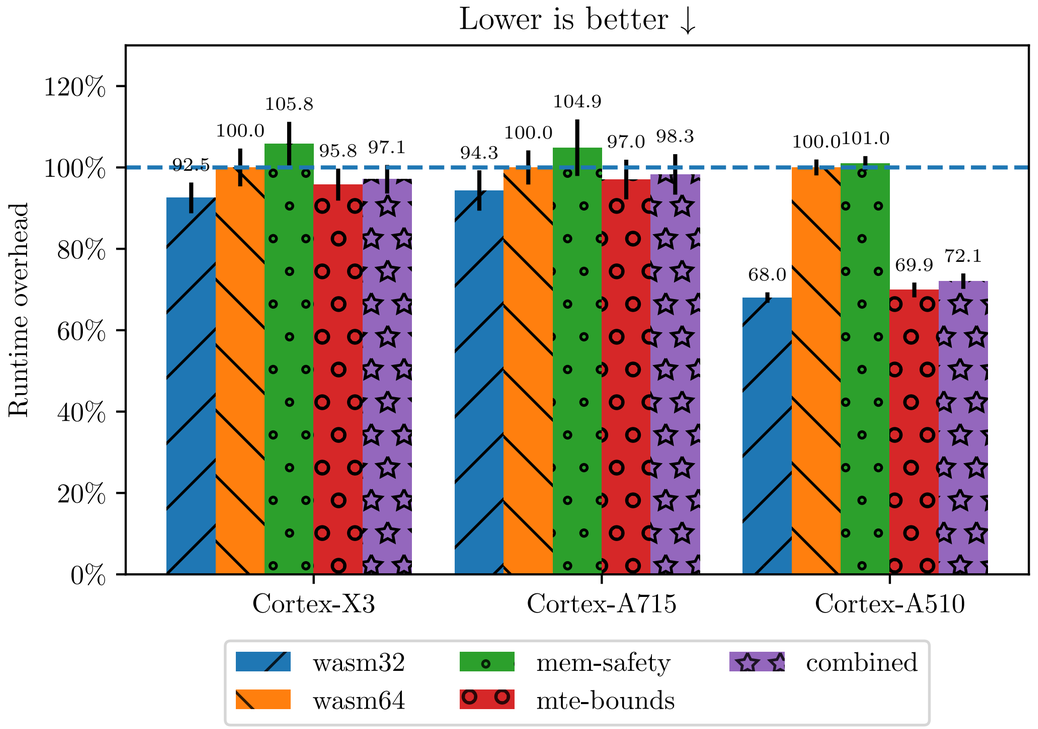


ARM Morello SoC

Results

MTE on Pixel 8

- PolyBench-C
  - Baseline: 64 bit WebAssembly
  - Memory safety: only **1.0% - 5.8% overhead**
  - MTE sandboxing: **3% - 29% speedup**
  - MTE sandboxing + memory safety: **1.7% - 28% speedup**



Summary

- Hardware-adaptable memory safety abstraction for WebAssembly
- Prevention of spatial and temporal memory safety issues
- Low overhead compared to baseline
- Speedup when eliminating software-based sandboxing

Next Steps

- Implementation on CHERI architecture
- Implementation of pointer authentication

https://github.com/TUM-DSE/wasmtime-mte
https://github.com/TUM-DSE/llvm-memsafe-wasm