Porting Rust to Morello

A safe software layer for a safe hardware layer

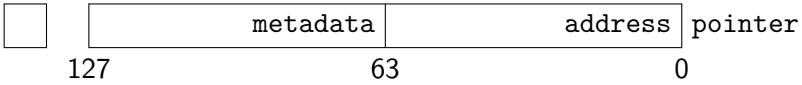
Sarah Harris, Simon Cooksey, Michael Vollmer,
Mark Batty

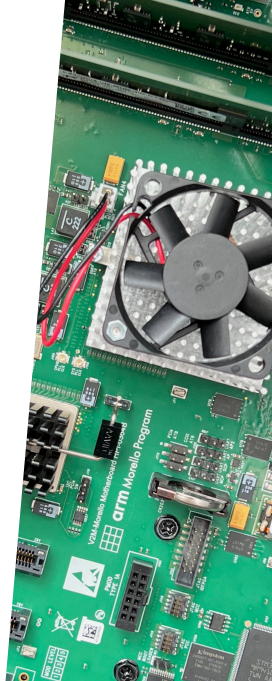
University of Kent

April 2025

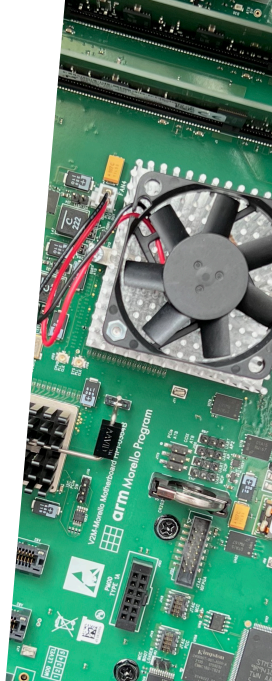
Capability machine

- Morello has hardware pointer provenance
- Pointers have an address (as usual), but also some metadata including a bounds and permissions flags.
- There is a transparent hardware managed validity bit which prevents pointer spoofing.





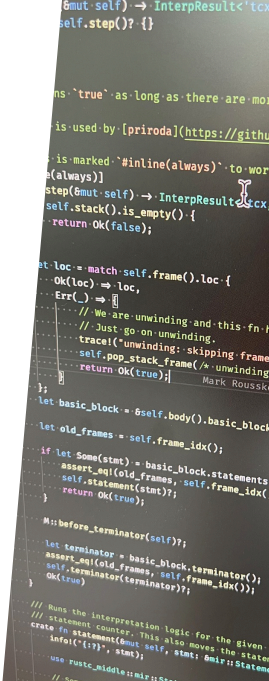
Morello prototype



Rust

- Rust is designed to be a safe systems programming language.
- The compiler (mostly) statically verifies that memory safety issues like use-after-free, and buffer overruns cannot happen.

```
fn main() {
  let mut x : [u8; 8] = [0; 8];
  x[9] = 1;
}
```

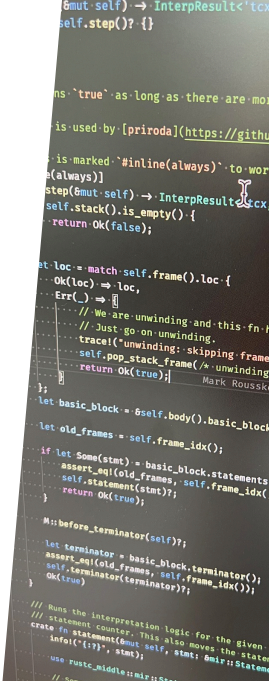


Rust

- Rust is designed to be a safe systems programming language.
- The compiler (mostly) statically verifies that memory safety issues like use-after-free, and buffer overruns cannot happen.

```
fn main() {
  let mut x : [u8; 8] = [0; 8];
  x[9] = 1;
}
```

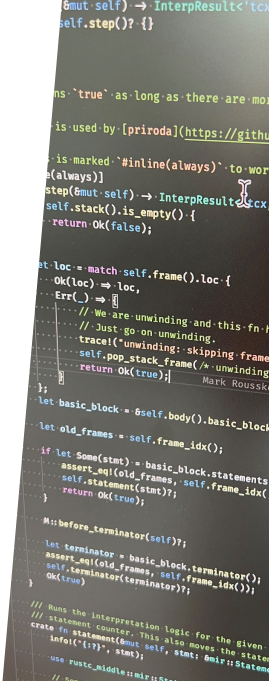
```
$ rustc ./main.rs -o oob-compile
error: this operation will panic at runtime
--> src/main.rs:3:5
  |
3 |     x[9] = 1;
  |     ^^^^ index out of bounds: the length is 8 but the index is 9
  |
```



Rust

- Rust is designed to be a safe systems programming language.
- The compiler (mostly) statically verifies that memory safety issues like use-after-free, and buffer overruns cannot happen.
- Rust is designed to be used in places where C/C++ is used.
- Rust has an escape keyword **`unsafe`**.

```
fn main() {
    let mut x : [u8; 8] = [0; 8];
    unsafe {
        *x.get_unchecked_mut(9) = 1;
    }
}
```

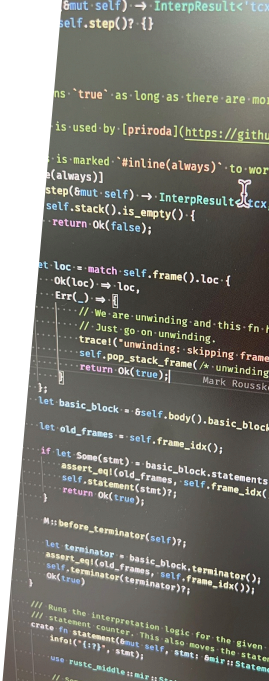


Rust

- Rust is designed to be a safe systems programming language.
- The compiler (mostly) statically verifies that memory safety issues like use-after-free, and buffer overruns cannot happen.
- Rust is designed to be used in places where C/C++ is used.
- Rust has an escape keyword **unsafe**.

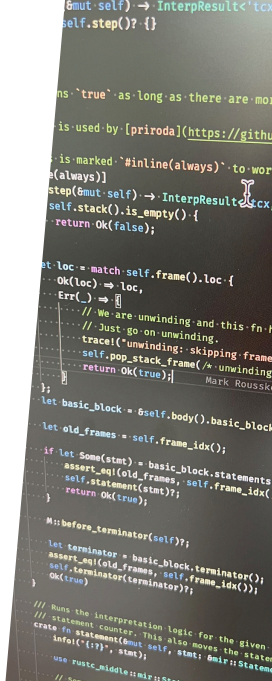
```
fn main() {
    let mut x : [u8; 8] = [0; 8];
    unsafe {
        *x.get_unchecked_mut(9) = 1;
    }
}
```

```
$ ./oob-runtime
Segmentation fault
```



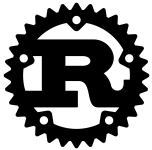
Why port Rust to Morello?

- The guarantees of capabilities complement the guarantees of Rust
- Rust provides compile-time guarantees for `safe` code
- Capabilities provide run-time guarantees for **`unsafe`** code

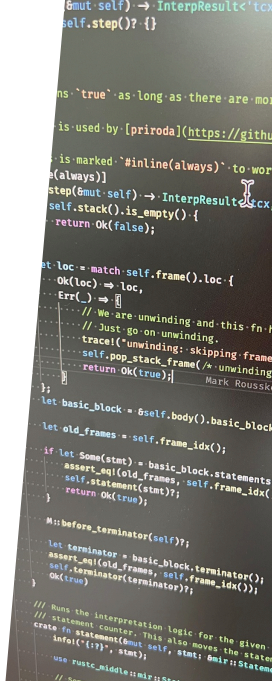


Why port Rust to Morello?

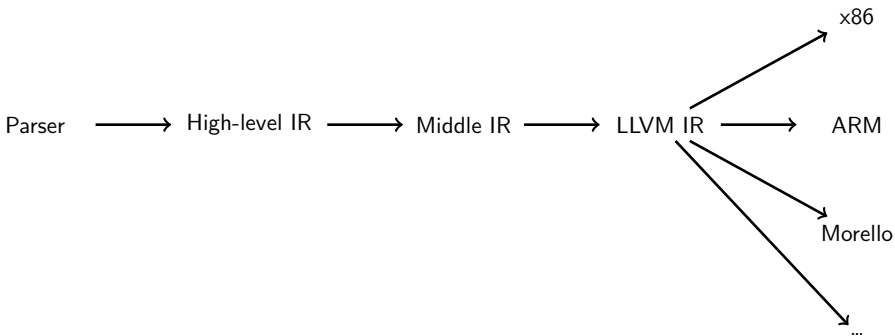
- The guarantees of capabilities complement the guarantees of Rust
- Rust provides compile-time guarantees for `safe` code
- Capabilities provide run-time guarantees for **`unsafe`** code



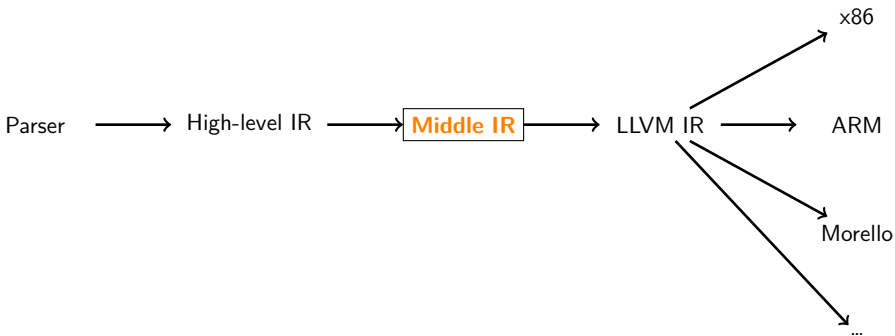
Digital Security by Design



The Rust Compiler



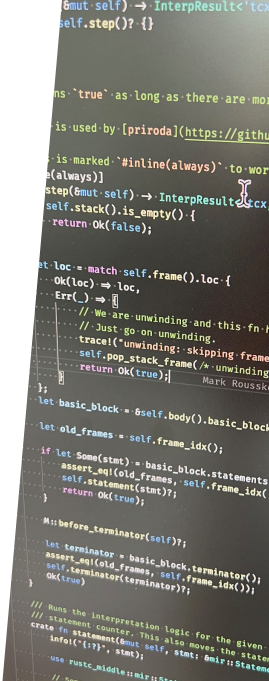
The Rust Compiler



Compiler changes — plumbing

The first task is hooking Rust up with Morello LLVM.

- We added a target, and set the appropriate options
- We hooked up Morello clang as the linker for the Rust compiler
- We extended the Rust target options to allow us to describe object layout differences...

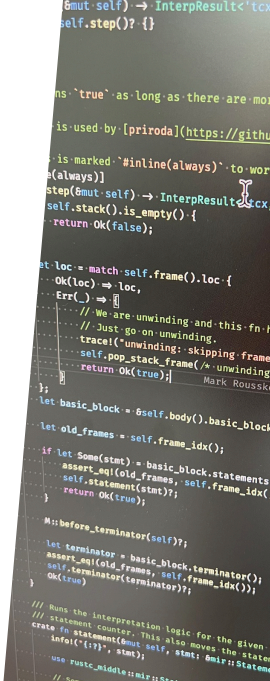


Compiler changes — object layout

Object layout differences, you say?

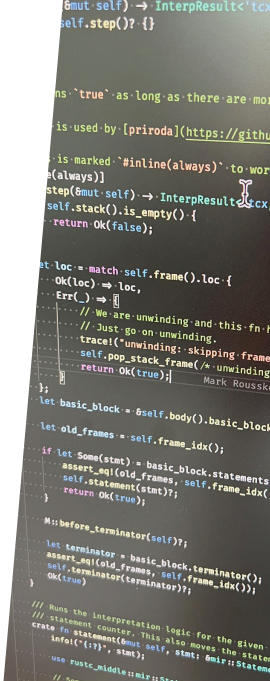
- **`usize`** is a type which must represent the whole range of addresses a pointer can dereference.
- It is used for array indexing, and array bounds.
- We don't want **`usize`** to be 128 bits, memory isn't 128 bit on Morello[†].
- We break the equality between **`usize`** and pointer size instead.

[†]This approach was explored by Nicholas Sim in his Masters Thesis.



Compiler changes — constant evaluation

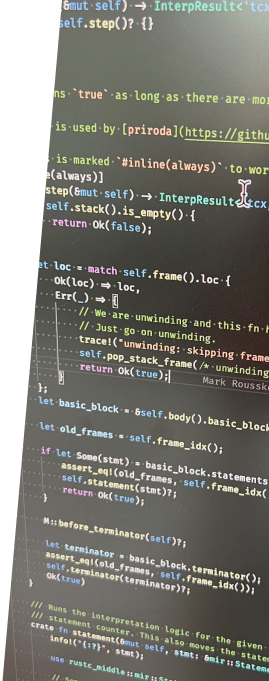
- Rust's IR is interpreted within the compiler to do constant evaluation.
- If it attempts to read uninitialised data that's considered an error.
- We cannot initialise the metadata of these pointers at compile time, so we had to patch up that divide.



Standard library changes

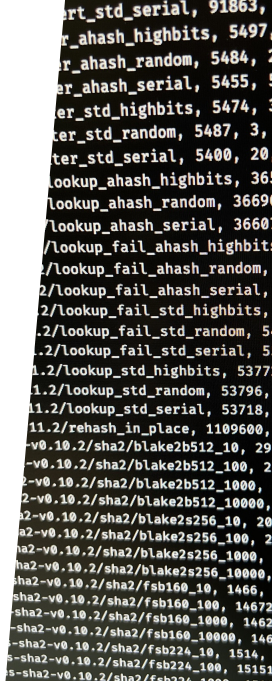
- The worst so far has been in a concurrency library which casts pointers to/from integers to tag them with metadata in the lower bits.
- Some bits of the FFI needed some tweaks, integer types being replaced with pointer types.

```
pub unsafe fn cast_from_usize(signal_ptr: usize) -> SignalToken {
    SignalToken { inner: mem::transmute(signal_ptr) }
}
```



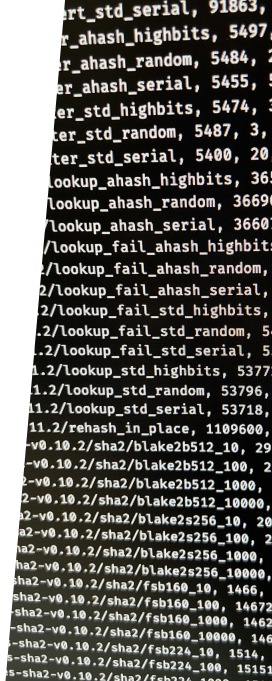
Measuring the performance of bounds checking

- It is interesting to understand what cost there is to Rust's dynamic bounds checking and how it relates to the always-on bounds checking in Morello.
- We have implemented a flag on the Rust compiler, `-C drop_bounds_checks`, which prevents the compiler from emitting software bounds checks. We call this version of the language Rust_{DBC}.



Measuring the performance of bounds checking

- We have picked 19 suites from the `crates.io` repository, which in total contain 872 benchmarks.
- These crates have 108k lines of Rust, of which 1k is `unsafe`. This does not include the dependencies.
- The benchmarks are run many times using the standard `cargo bench` command, and results are aggregated.

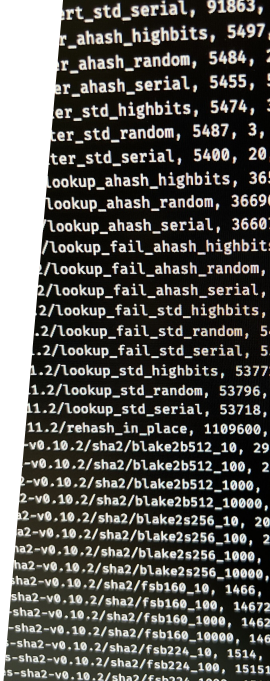


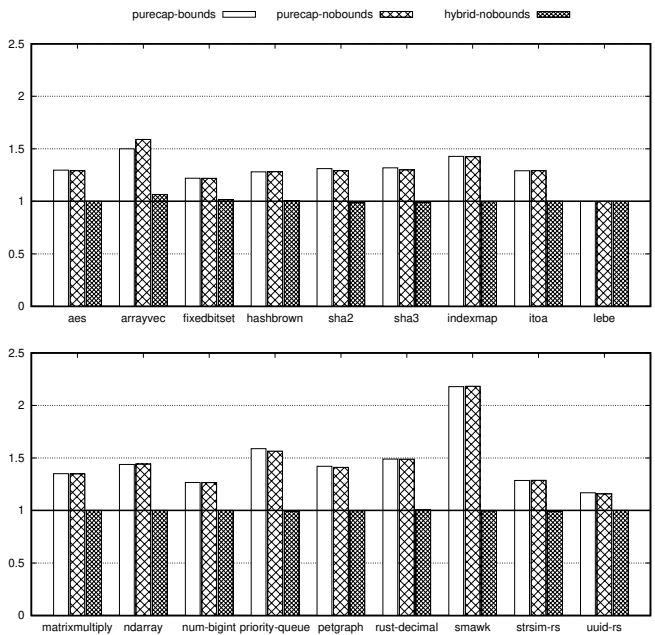
Measuring the performance of bounds checking

- We have picked 19 suites from the `crates.io` repository, which in total contain 872 benchmarks.
- These crates have 108k lines of Rust, of which 1k is **`unsafe`**. This does not include the dependencies.
- The benchmarks are run many times using the standard `cargo bench` command, and results are aggregated.

From `cargo bench` we extract time per iteration of the benchmark, and the run-to-run variance, for each of the four modes under test:

`hashbrown-0.11.2/clone_from_large`				
	Rust		Rust_{DBC}	
	Time/iter	$\pm$	Time/iter	$\pm$
Purecap	15,779	8	15,818	59
Hybrid	15,557	53	15,601	16



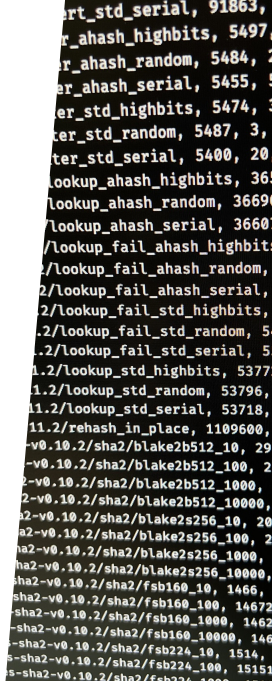

purecap-bounds
purecap-nobounds
hybrid-nobounds
2.5
2
1.5
1
0.5
0
aes
arrayvec
fixedbitset
hashbrown
sha2
sha3
indexmap
itoa
lebe
2.5
2
1.5
1
0.5
0
matrixmultiply
ndarray
num-bigint
priority-queue
petgraph
rust-decimal
smawk
strsim-rs
uuid-rs

Performance results

Overall (by geometric mean) we measured a slow-down on Purecap Morello. We found the cost of Rust's dynamic bounds checking to be extremely low.

The slowdown on Morello appears to be exagerated because branch prediction is less effective on the prototype. The fix can be modelled on current hardware in a special target, and we will re-run these tests.

Rust for Morello: Always-On Memory Safety, Even in Unsafe Code, ECOOP 2023

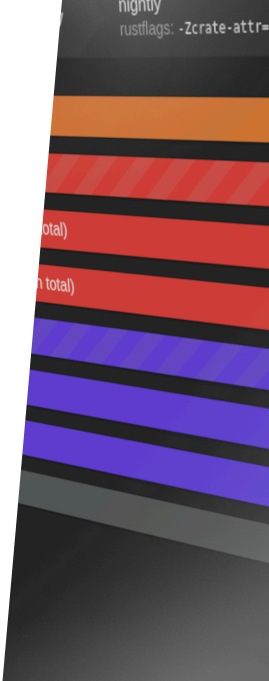


What Effect Could Our Changes Have?

Object layout differences: differentiate `usize` and capability size.

Changing types? Doesn't that break things?

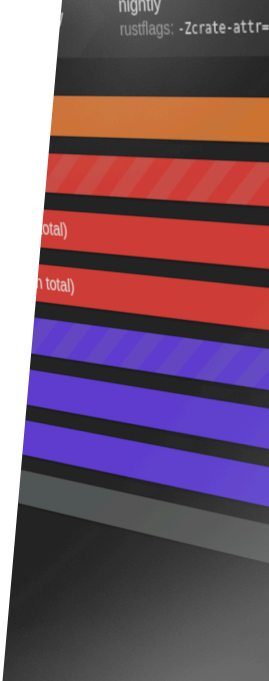
- Well, yes, but we think it will be rare
- Plain pointers are usually unsafe
- `usize` to pointer casts are odd and unidiomatic
- Still, we can do better than, "we think"!



What Effect Could Our Changes Have?

We arranged a *Crater* run with lints to detect misuse (387,225 crates).

- Result: 0.49-0.85% of projects fail the lint
- We should note some limitations, but this looks promising



What Effect Could Our Changes Have?

We arranged a *Crater* run with lints to detect misuse (387,225 crates).

- Result: 0.49-0.85% of projects fail the lint
- We should note some limitations, but this looks promising
- false negatives (`transmute::<usize, *const T>()`)
- false positives (`ALIAS_FOR_ZERO as *const T`)
- we scanned logs for patterns, not foolproof!†
- 37.6% broken anyway, aborted builds can hide problems

†script here: `https://gist.github.com/seharris/fb7606\e0dbddcabbc9702c644372a95b`

Any questions?

- Port of Rust to Morello – Mac and Linux binaries.
- Rust code running on Morello + measured needed changes.
- Baremetal Rust (Michael Vollmer)
- Performance numbers need updating for benchmark target.

`https://github.com/kent-weak-memory/rust`