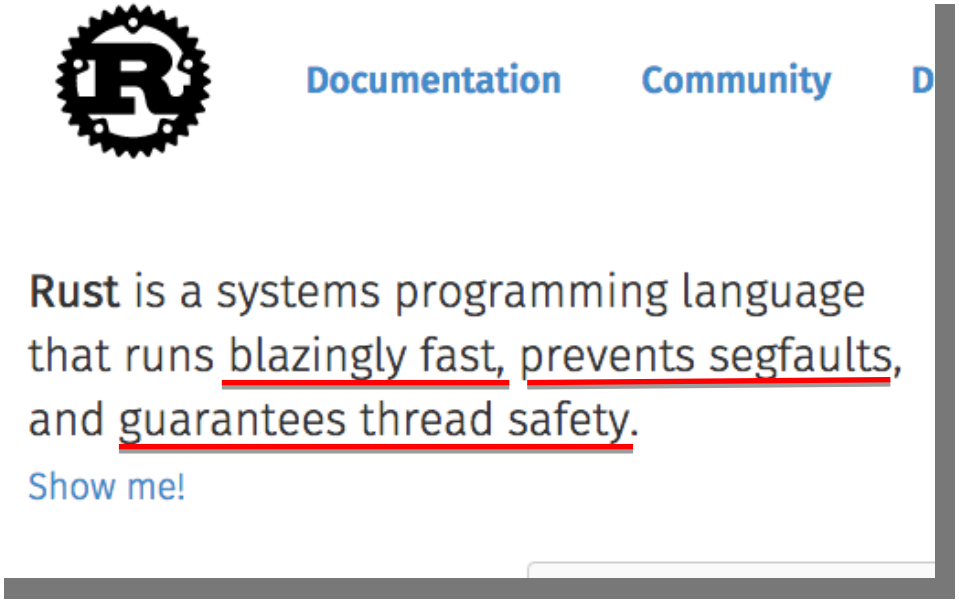
Australian National University

Implementing High Performance GC in Rust

Yi Lin, ANU

Supervisory Panel: Steve Blackburn (Chair), ANU
Michael Norrish, ANU/NICTA
Tony Hosking, ANU/NICTA

WHY RUST?



Garbage Collector

- ☑ performance critical
- ☑ low-level raw memory
- ☑ thread parallelism
- ☑ thread concurrency

Rust

- Started by Mozilla for next-gen web renderer
- ‘rustc’ is implemented in Rust
- Some concepts from ‘CCured’
- Formalized semantics: Petina, RustBelt

Rust's safety guarantees

- no data race
- no dangling/null pointer
- memory safety w/o manually free or GC

…

mostly at compile time!

Rust 101: Ownership

```
let foo = Foo::new();

let foo2 = foo;

foo.access();
```

Rust 101: Reference Borrowing

```
let foo = Foo::new();

let foo_ref = &foo;

foo.access();
```

Rust 101: Reference Borrowing

```
let mut foo = Foo::new();

let foo_ref_mut = &mut foo;

foo.update();
```

mut

Rust 101: Lifetimes

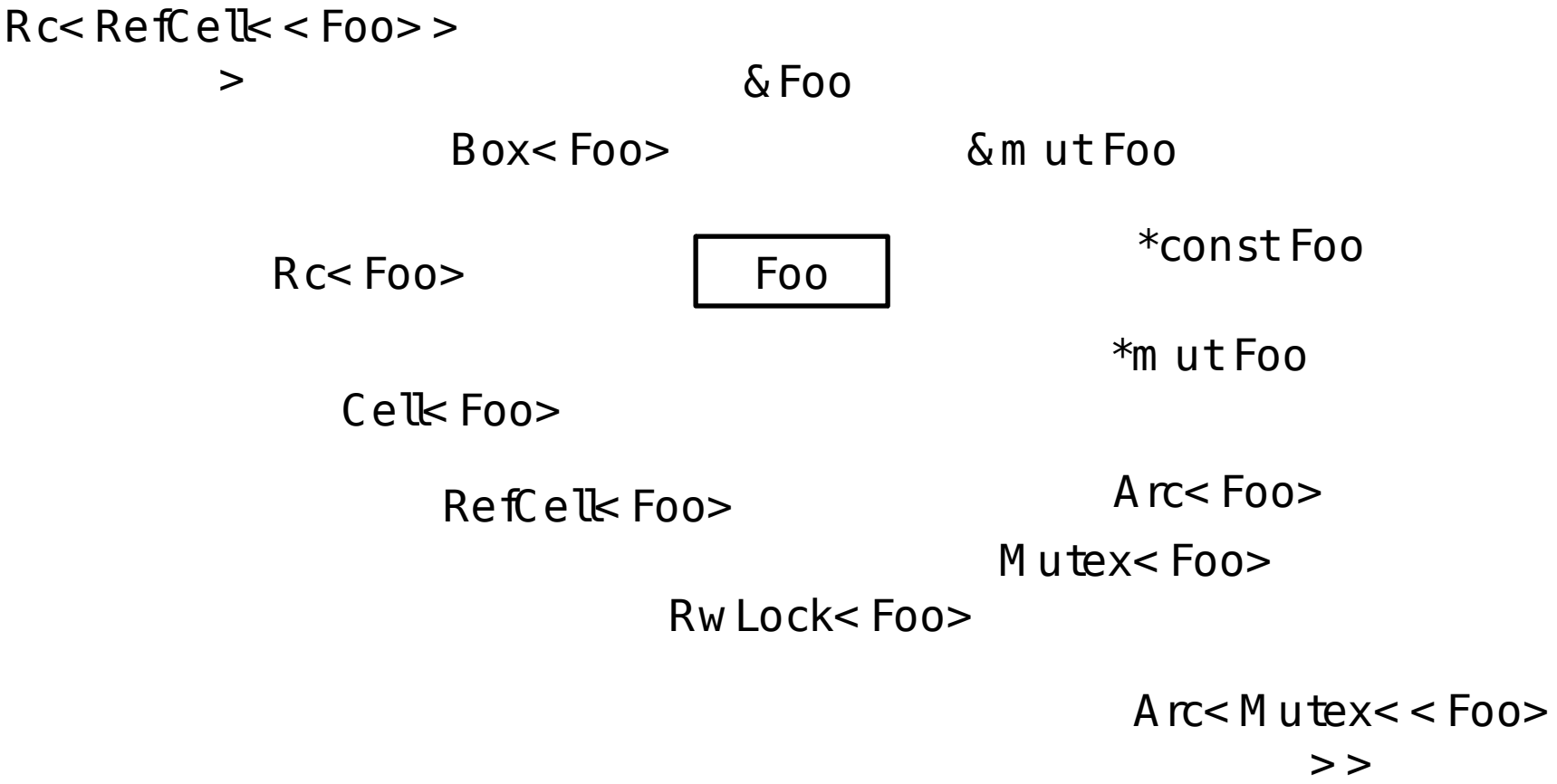
```
let mut foo = Foo::new();

{
  let foo_ref_mut = &mut foo;
}

foo.update();
```

mut

Rust 101: Data Guarantees (Wrappers)



WRITING GC IN RUST

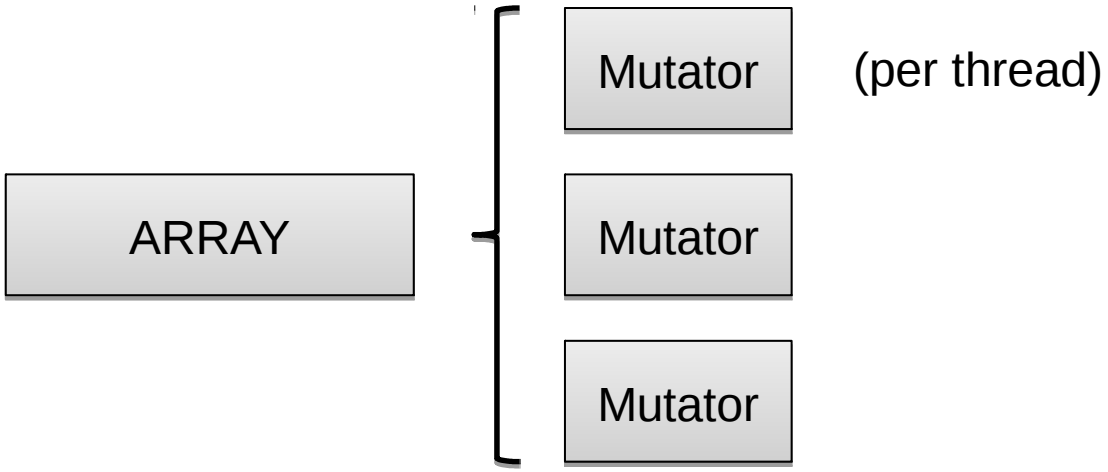
Ownership Transfer

Box<Block>

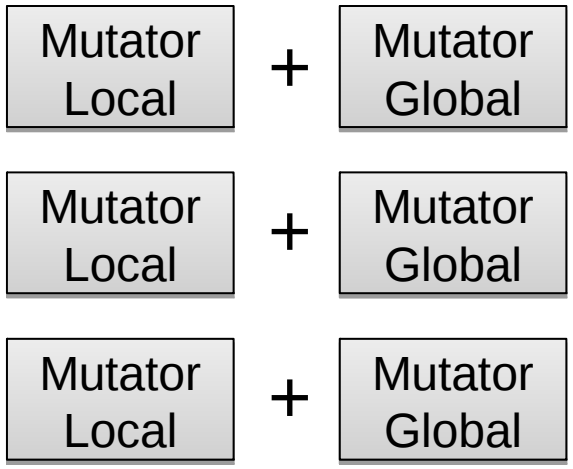
```
struct MutatorLocal{
 …
 block :Option<Box<Block>>
}
```

```
struct Space {
 …
 usable_blocks:LinkedList<Box<Block>>,
 used_blocks :LinkedList<Box<Block>>
}
```

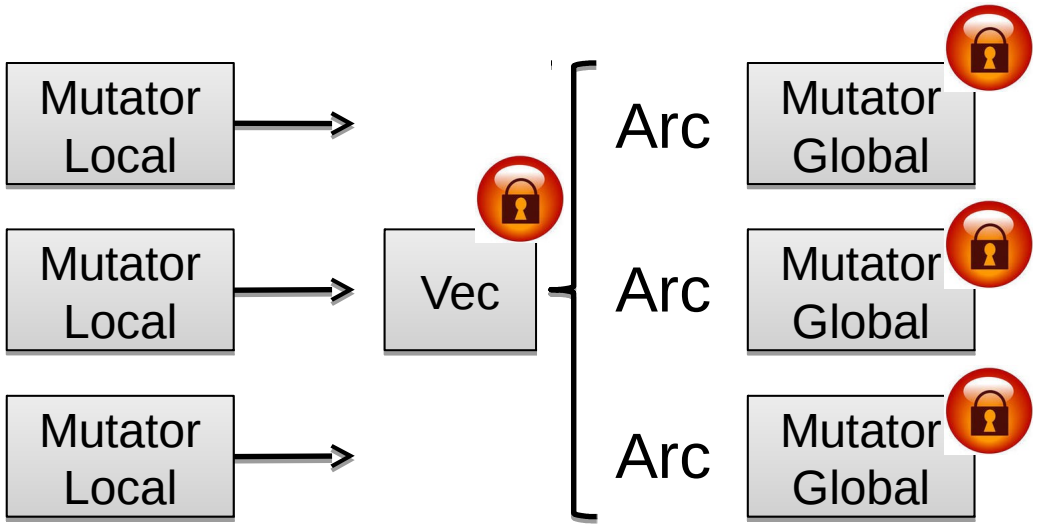
Globally accessible TL data



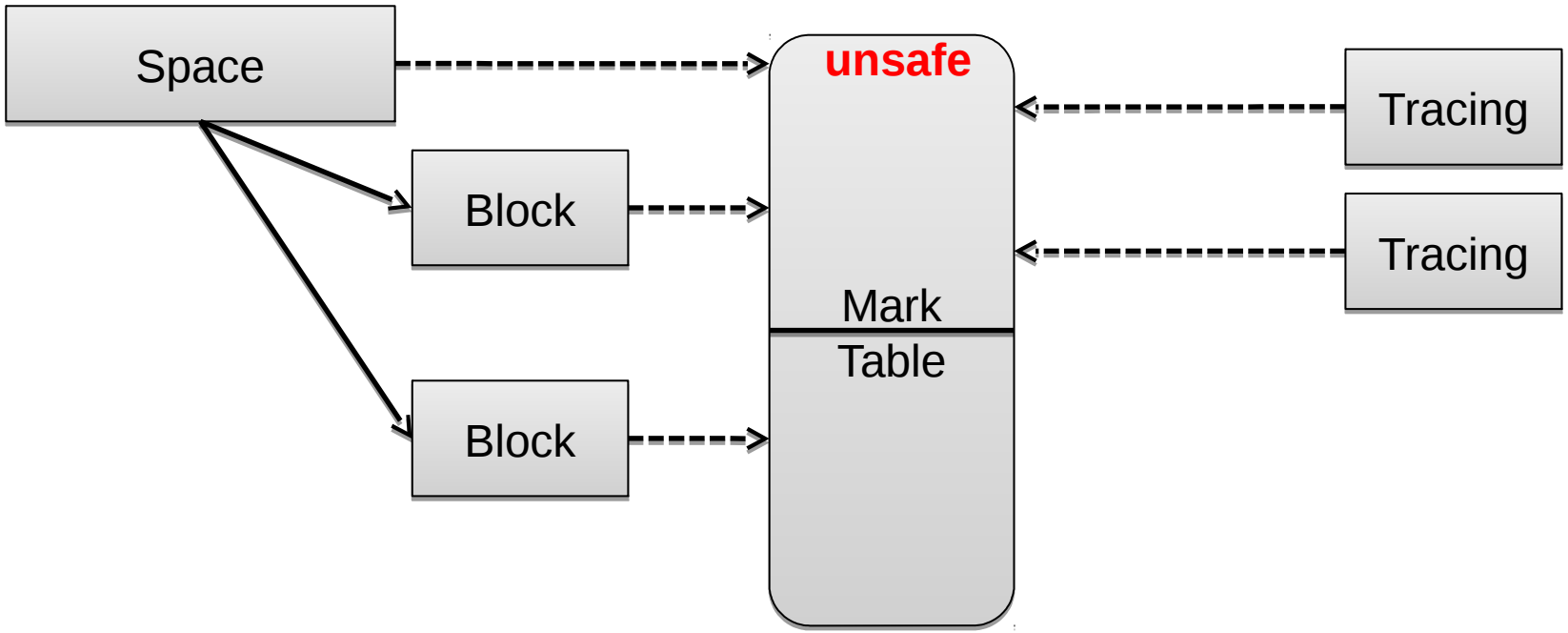
Globally accessible TL data



Globally accessible TL data



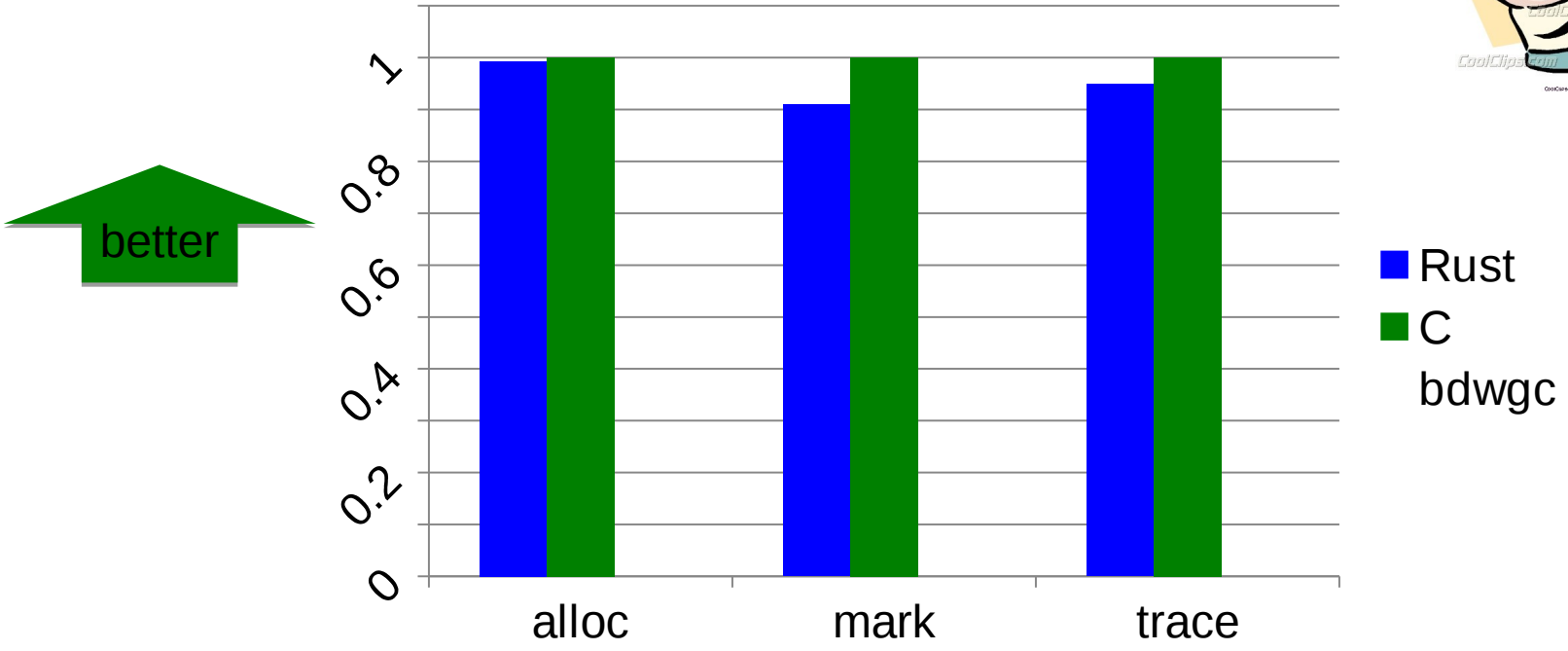
Mark Table (unsafe)



Safe code

- ~98% LOC
- ~2% unsafe code:
  - to access memory through raw pointers
  - to bypass Rust's restricted semantics

Micro Benchmarks



within **10%** of C performance

Conclusion

- GC is not mostly ‘unsafe’ as we may expect.
- Rust can deliver good performance along with safety for GC implementations.

vmmagic [1]

```
//Java

@Unboxed
@RawStorage(
lengthInWords = true, length = 1)
public final class Address {
 public Address plus(int v);
 public Offset sub(Address addr2);
 …
}
```

```
//Rust

#[derive(Copy, Clone)]
pub struct Address(usize);

impl Address {
  #[inline(always)]
  pub fn plus(Self, v: usize) ->
Address{… };

  #[inline(always)]
  pub fn sub(Self, addr2: Address) -> Offset
{… }
}
```

[1] ‘vmmagic’: D. Frampton et al., VEE’09

Globally accessible TL data

```
struct MutatorLocal{
 cursor: Address,
 limit: Address,
 …
 global: Arc<MutatorGlobal>
}

struct MutatorGlobal{
 take_yield: AtomicBool,
}

lazy_static!{
 pub static ref MUTATORS
 : RwLock<Vec<Arc<MutatorGlobal>>>
 = RwLock::new(vec![]);
}
```

Rust

- Some concepts from CCured
- Formal semantics: Petina, RustBelt