Cpp2Rust: Automatic Translation of C++ to Safe Rust

Lucian Popescu, Francisco Gouveia, Henrique Preto, João Silveira, Dmytro Hrybenko*, José Fragoso Santos, Nuno Lopes

University Of Lisbon & *Google

~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

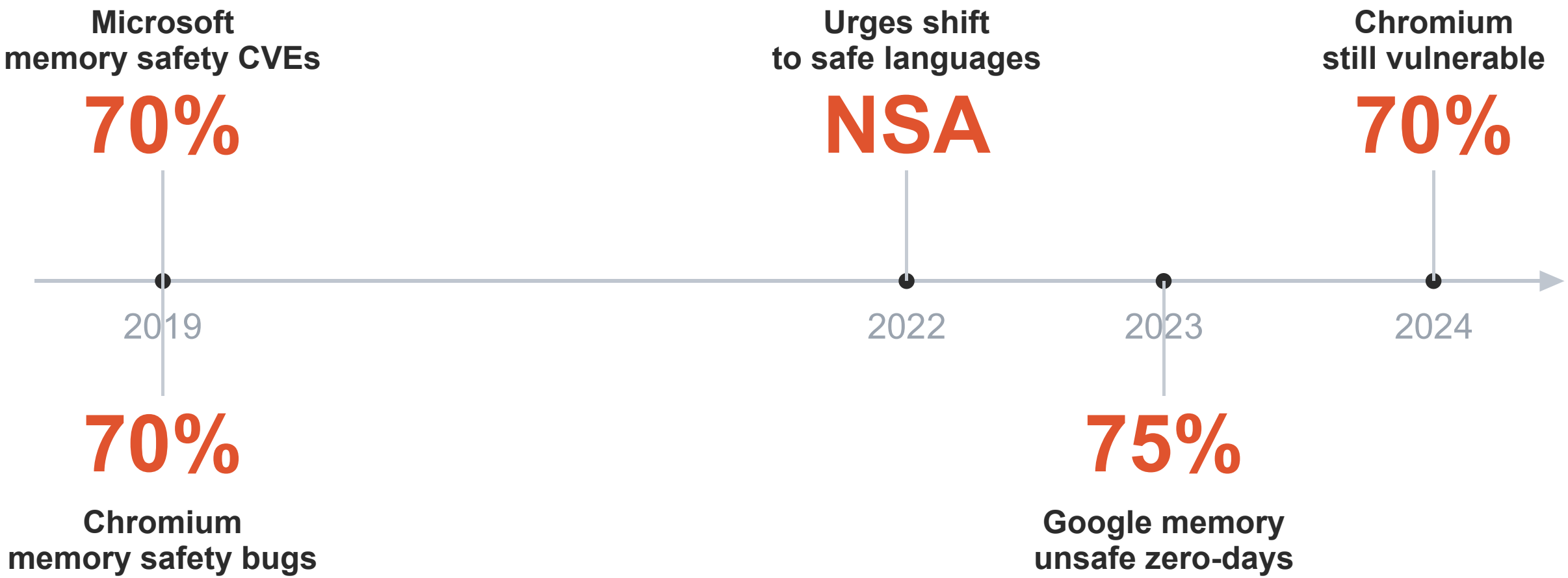
While many experienced programmers can write correct systems-level code, it's clear that no matter the amount of mitigations put in place, it is

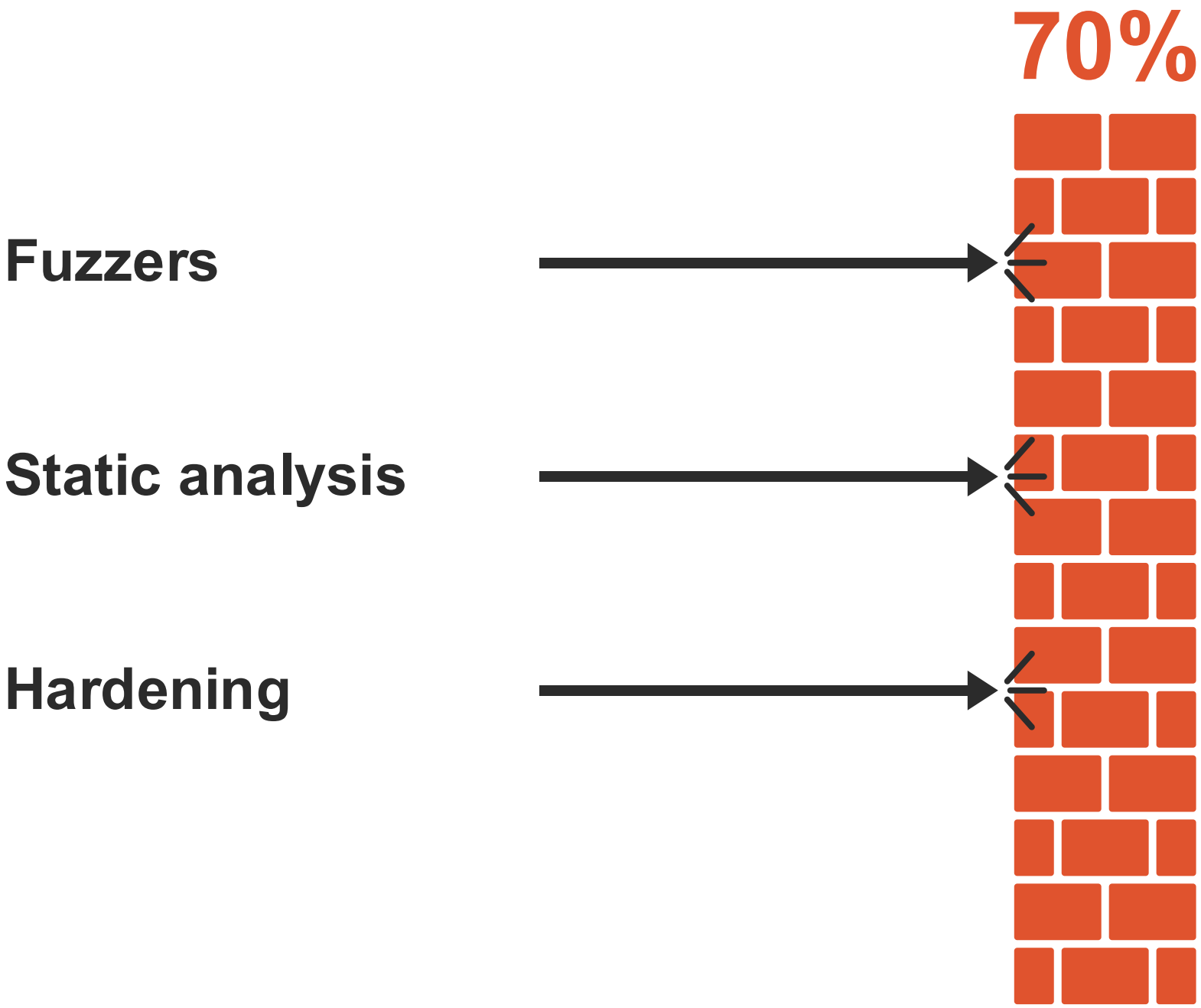
Chromium > Chromium Security >

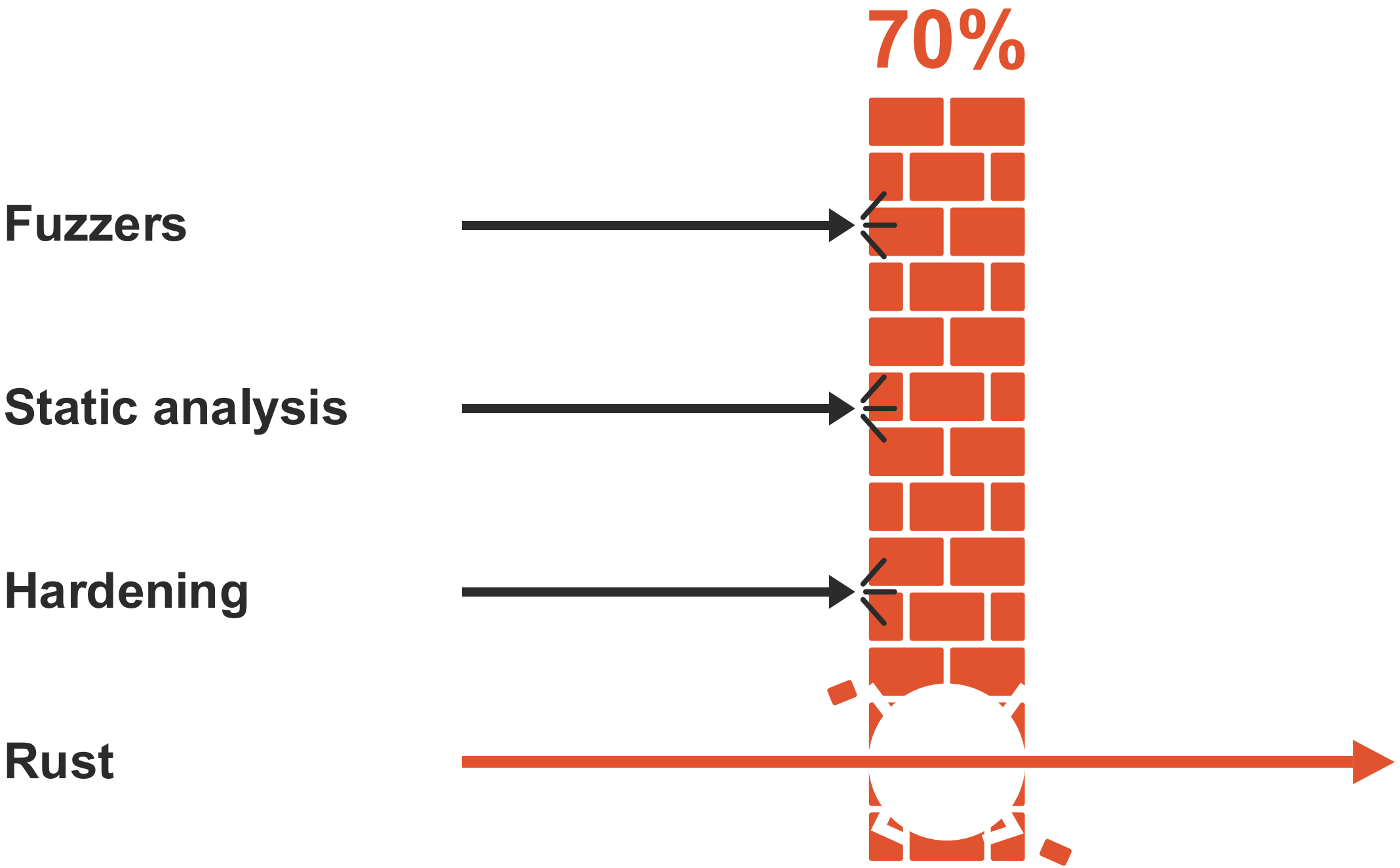
Memory safety

The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.

Memory safety stays the dominant root cause




70%
Fuzzers
Static analysis
Hardening


70%
Fuzzers
Static analysis
Hardening
Rust

Manual ports don’t scale

woff2: font compression @ Google

C++ -> Rust

4K

lines of code

20

days to port

20

new bugs

We need automatic rewrites!

C2Rust (existing approach)

```
int x = 10;
int *p = &x;
int *q = &x;
*p = 20;
*q = 30;
return x;
```

mechanical translation to **unsafe** Rust

```
let mut x: i32 = 10;
let p: *mut i32 = &mut x;
let q: *mut i32 = &mut x;
unsafe { *p = 20; }
unsafe { *q = 30; }
return x;
```

C2Rust fallbacks to `unsafe` because it cannot statically prove aliasing

Cpp2Rust (our contribution)

```
int x = 10;
int *p = &x;
int *q = &x;
*p = 20;
*q = 30;
return x;
```

Safe mechanical translation

```
let x: Value<i32> =
  Rc::new(RefCell::new(10));
let p: Ptr<i32> = x.as_pointer();
let q: Ptr<i32> = x.as_pointer();
*p.deref() = 20;
*q.deref() = 30;
return *x.borrow();
```

type Value<T> = Rc<RefCell<T>>

Rc: multiple aliases
RefCell: mutation through aliases

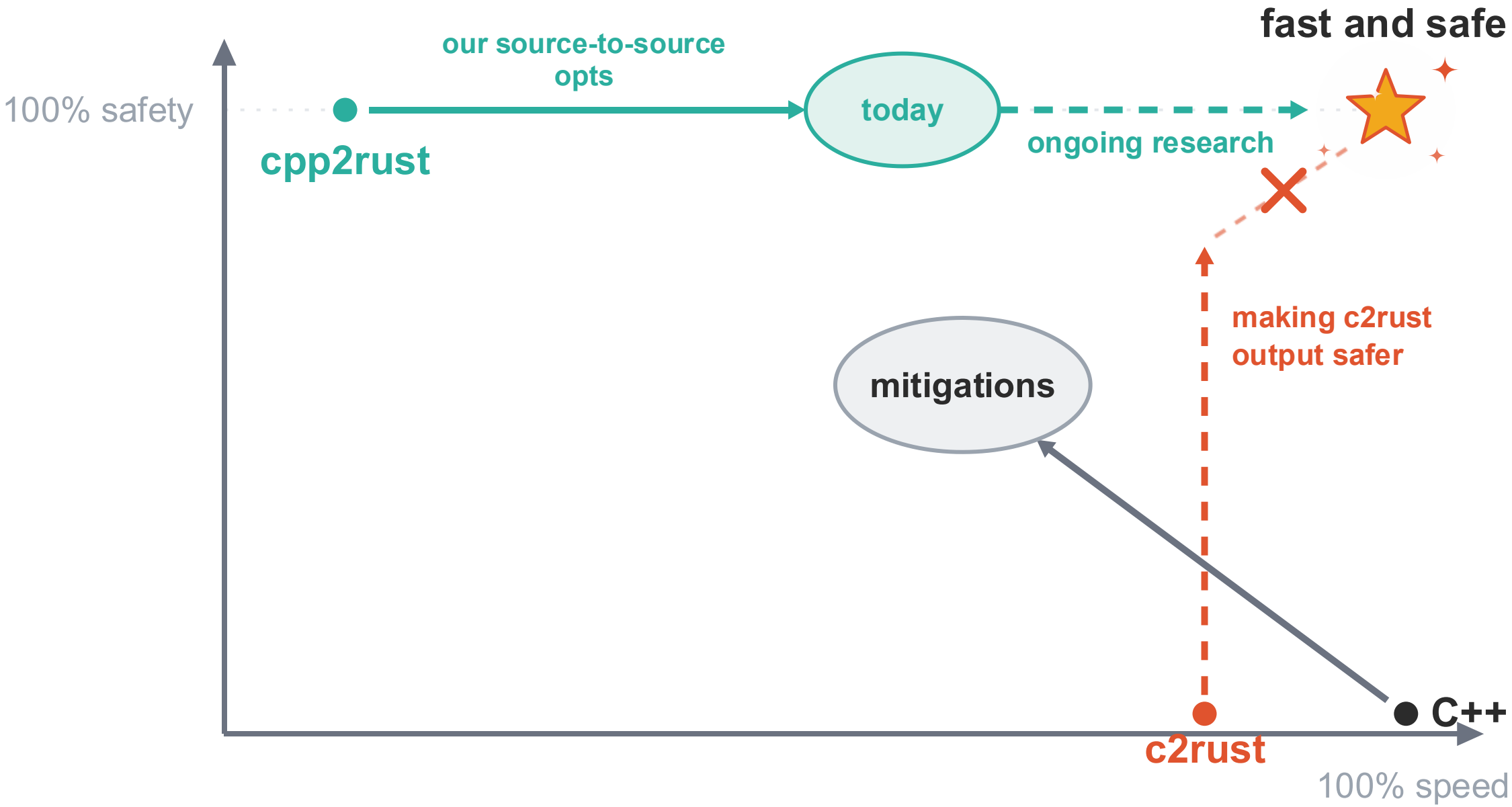
```
struct Ptr {
 weak: Weak<RefCell<T>>,
 offset: usize
}
```

Cpp2Rust generates safe Rust by shifting mutability and ownership to runtime

Language tradeoffs

Language tradeoffs

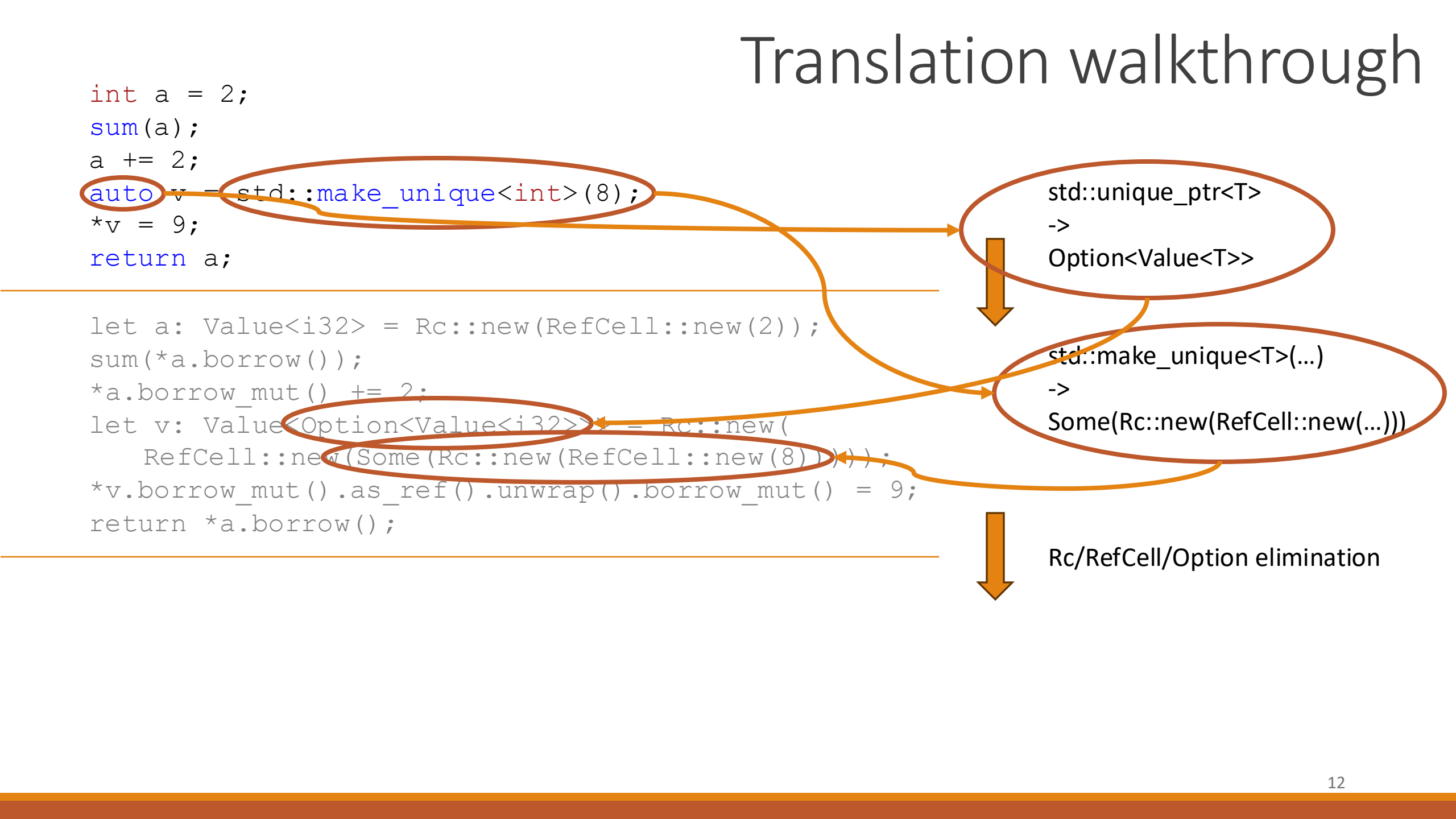
Language tradeoffs



Translation walkthrough

```
int a = 2;
sum(a);
a += 2;
auto v = std::make_unique<int>(8);
*v = 9;
return a;
```

```
let a: Value<i32> = Rc::new(RefCell::new(2));
sum(*a.borrow());
*a.borrow_mut() += 2;
let v: Value<Option<Value<i32>>> = Rc::new(
    RefCell::new(Some(Rc::new(RefCell::new(8))));
*v.borrow_mut().as_ref().unwrap().borrow_mut() = 9;
return *a.borrow();
```



Translation walkthrough

```
int a = 2;
sum(a);
a += 2;
auto v = std::make_unique<int>(8);
*v = 9;
return a;
```

std::unique_ptr<T>
->
Option<Value<T>>

```
let a: Value<i32> = Rc::new(RefCell::new(2));
sum(*a.borrow());
*a.borrow_mut() += 2;
let v: Value<Option<Value<i32>>> = Rc::new(
    RefCell::new(Some(Rc::new(RefCell::new(8)))));
*v.borrow_mut().as_ref().unwrap().borrow_mut() = 9;
return *a.borrow();
```

std::make_unique<T>(…)
->
Some(Rc::new(RefCell::new(…)))

Rc/RefCell/Option elimination

Translation walkthrough

```
int a = 2;
sum(a);
a += 2;
auto v = std::make_unique<int>(8);
*v = 9;
return a;
```

```
let a: RefCell<i32> = RefCell::new(2);
sum(*a.borrow());
*a.borrow_mut() += 2;
let v: RefCell<Option<Value<i32>>> =
   RefCell::new(Some(Rc::new(RefCell::new(8))));
*v.borrow_mut().as_ref().unwrap().borrow_mut() = 9;
return *a.borrow();
```

std::unique_ptr<T>
->
Option<Value<T>>

std::make_unique<T>(…)
->
Some(Rc::new(RefCell::new(…)))

Rc/RefCell/Option elimination

Translation walkthrough

```
int a = 2;
sum(a);
a += 2;
auto v = std::make_unique<int>(8);
*v = 9;
return a;
```

std::unique_ptr<T>
->
Option<Value<T>>

```
let mut a: i32 = 2;
sum(a);
a += 2;
let v: Option<Value<i32>> =
    Some(Rc::new(RefCell::new(8)));
*v.as_ref().unwrap().borrow_mut() = 9;
return a;
```

std::make_unique<T>(…)
->
Some(Rc::new(RefCell::new(…)))

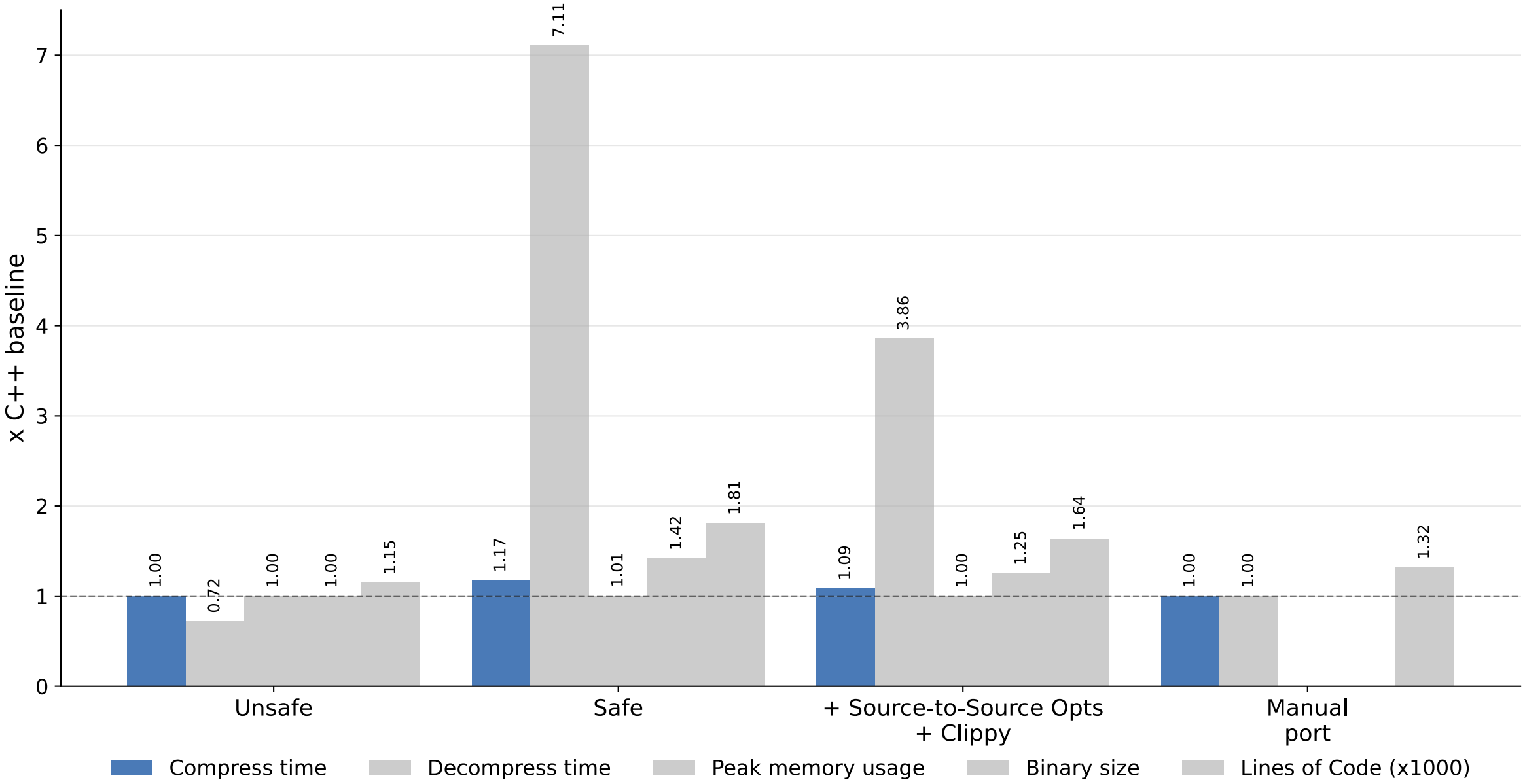
Rc/RefCell/Option elimination

```
let mut a: i32 = 2;
sum(a);
a += 2;
let mut v: i32 = 8;
v = 9;
return a;
```

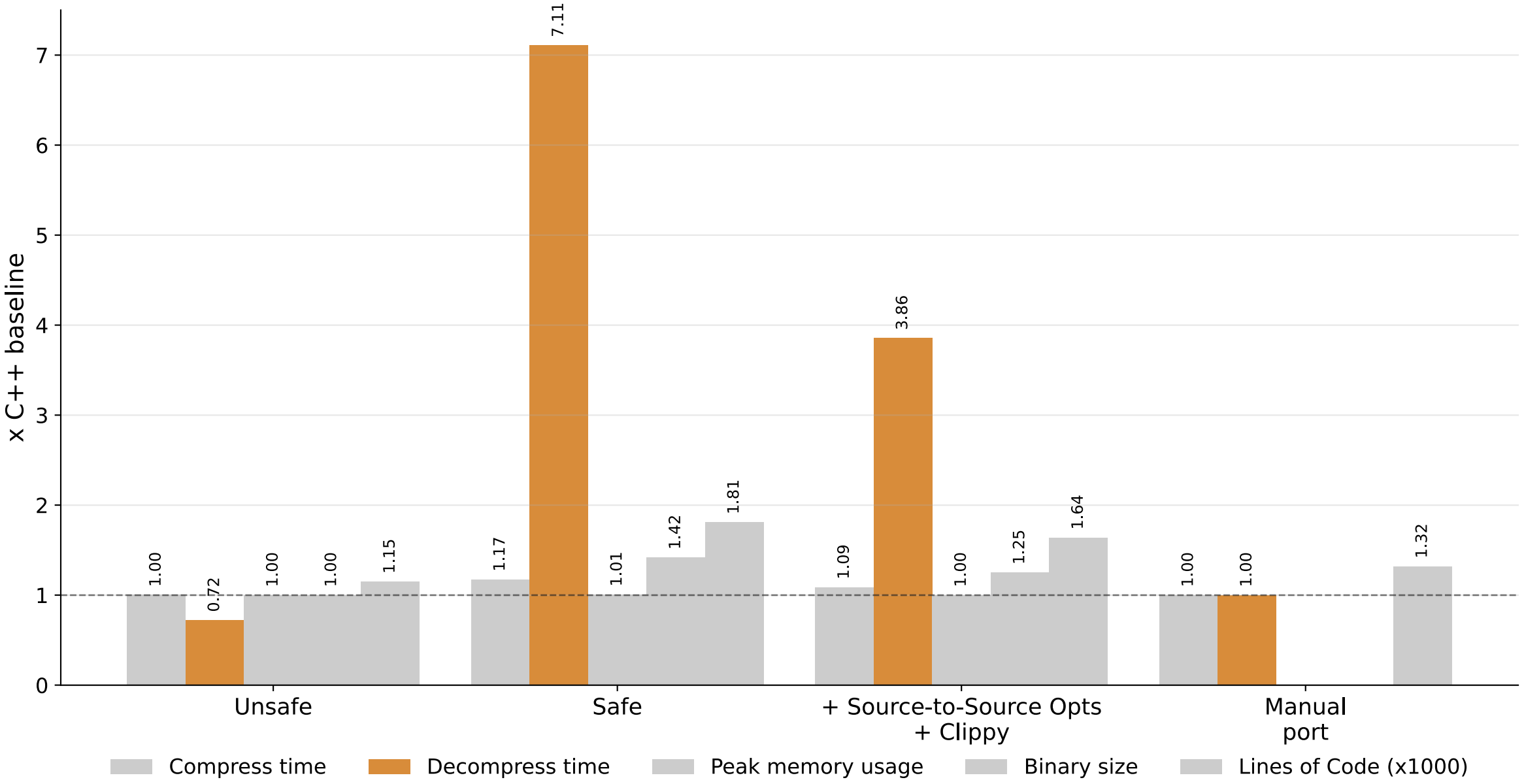
Evaluation

- Translated to safe Rust: woff2 (font compression), brunsli (JPEG repacker). 13K LoC total
- C++ features: STL (map, vector, unique_ptr), OOP, void*, lambdas
- We compare the perf of C++ vs Cpp2Rust unsafe vs Cpp2Rust Safe

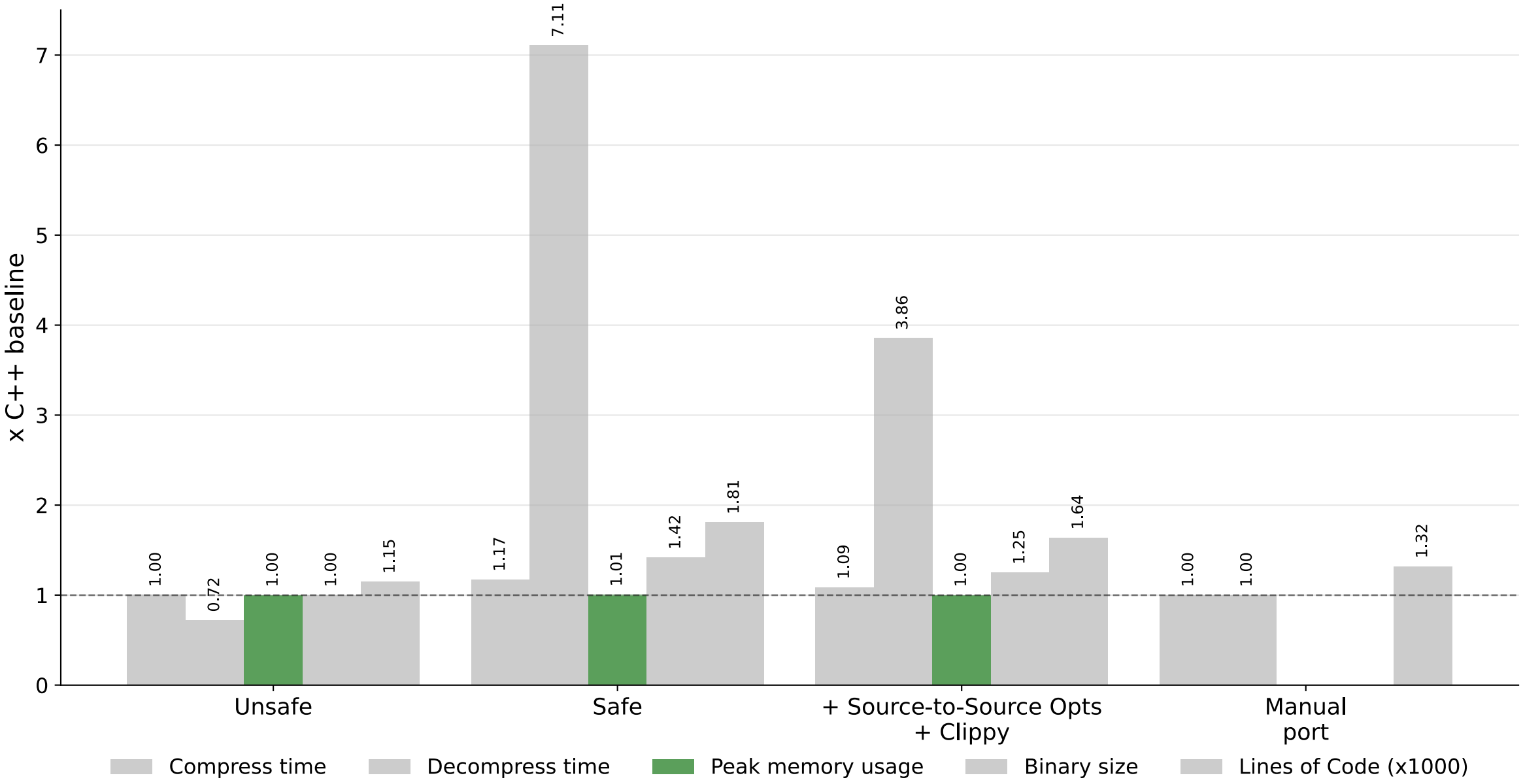
WOFF2: metrics relative to C++



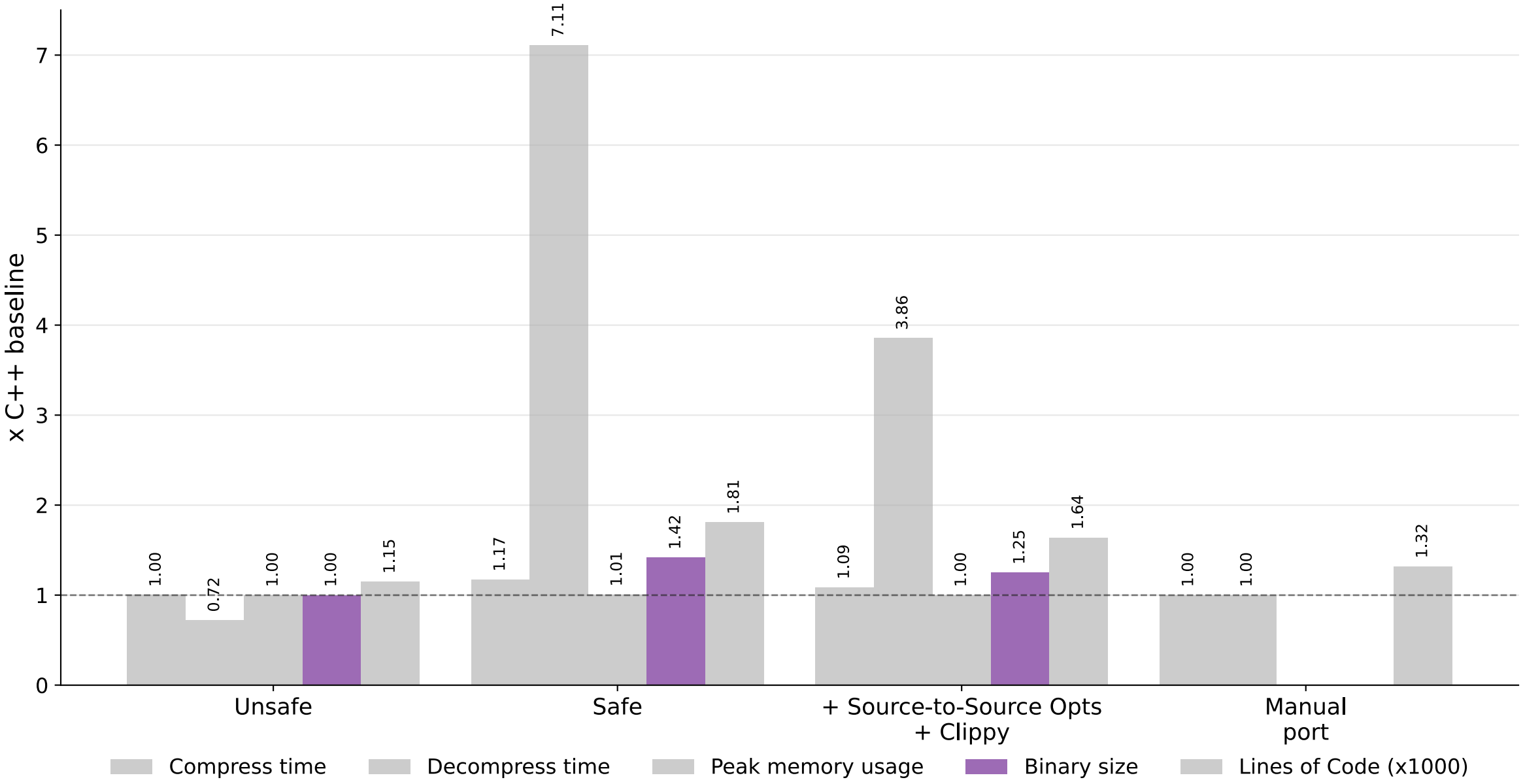
WOFF2: metrics relative to C++



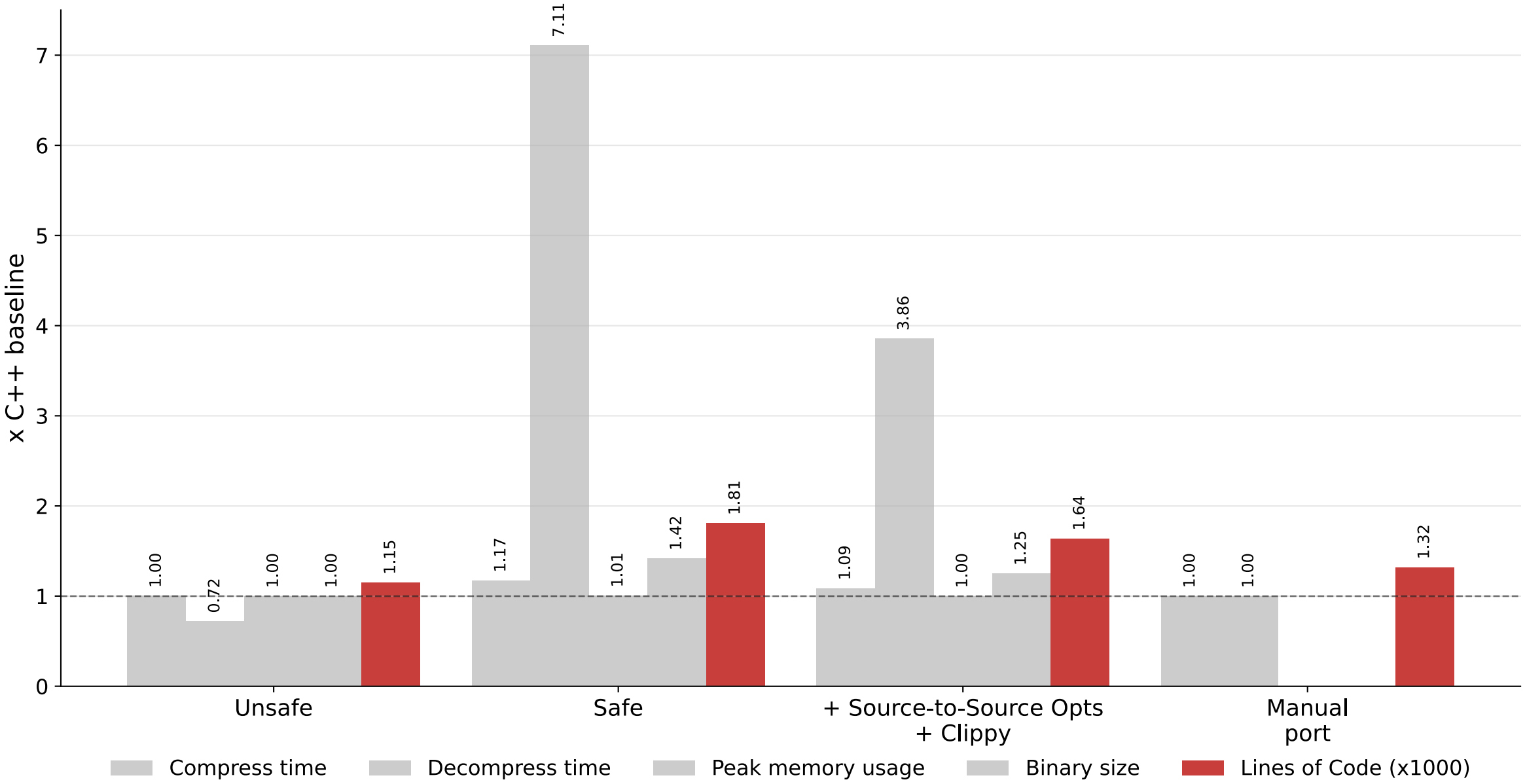
WOFF2: metrics relative to C++



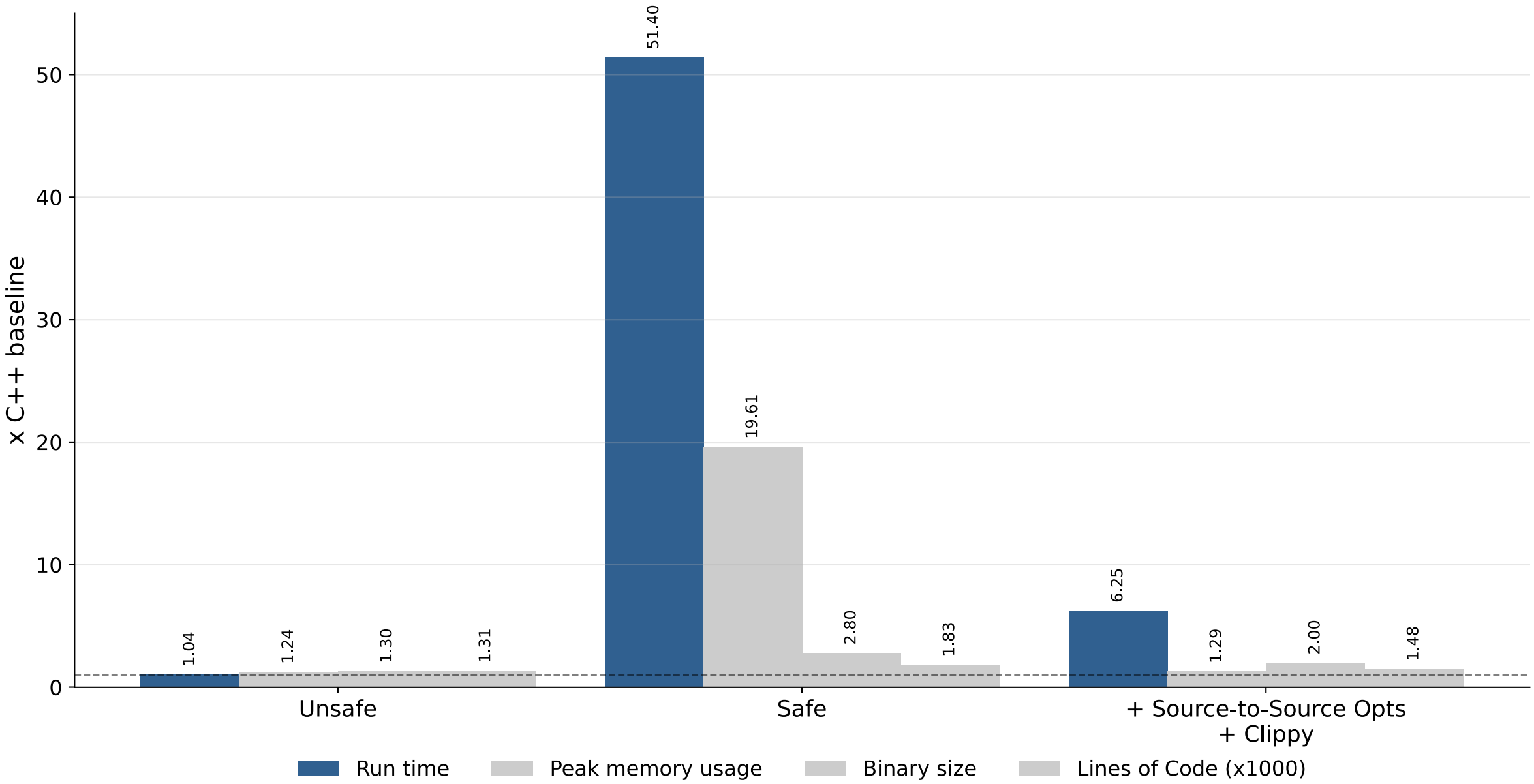
WOFF2: metrics relative to C++



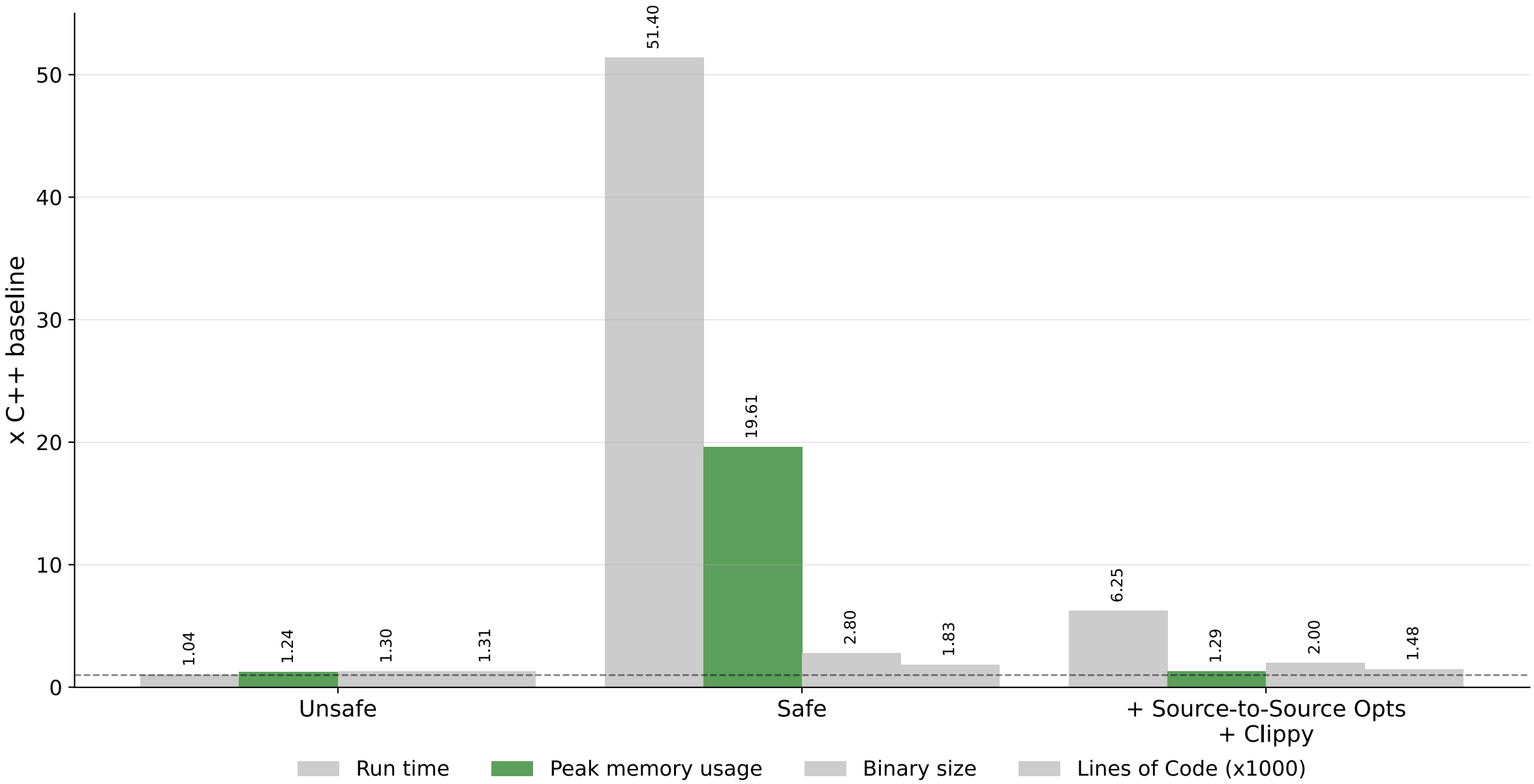
WOFF2: metrics relative to C++



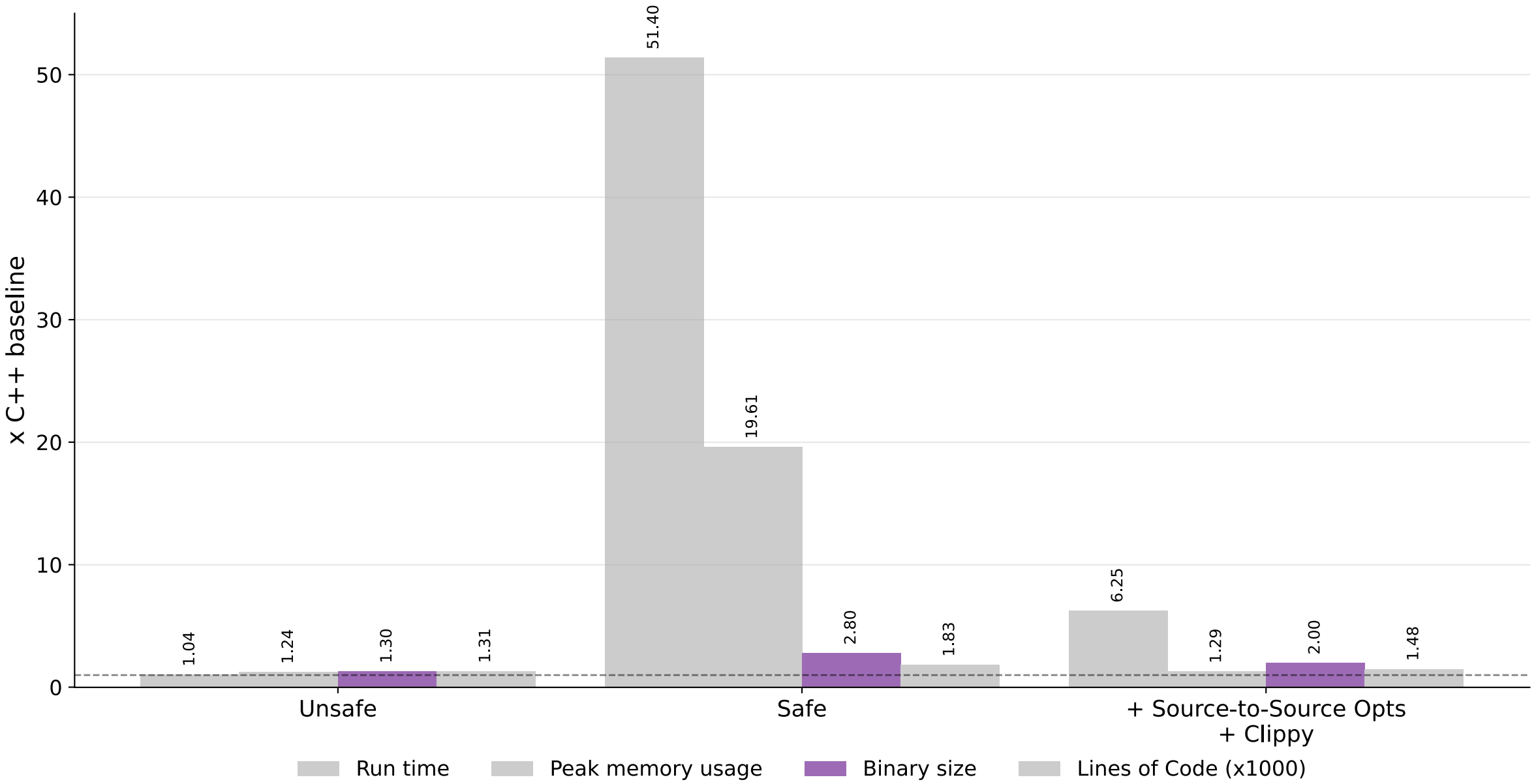
Brunsli: metrics relative to C++



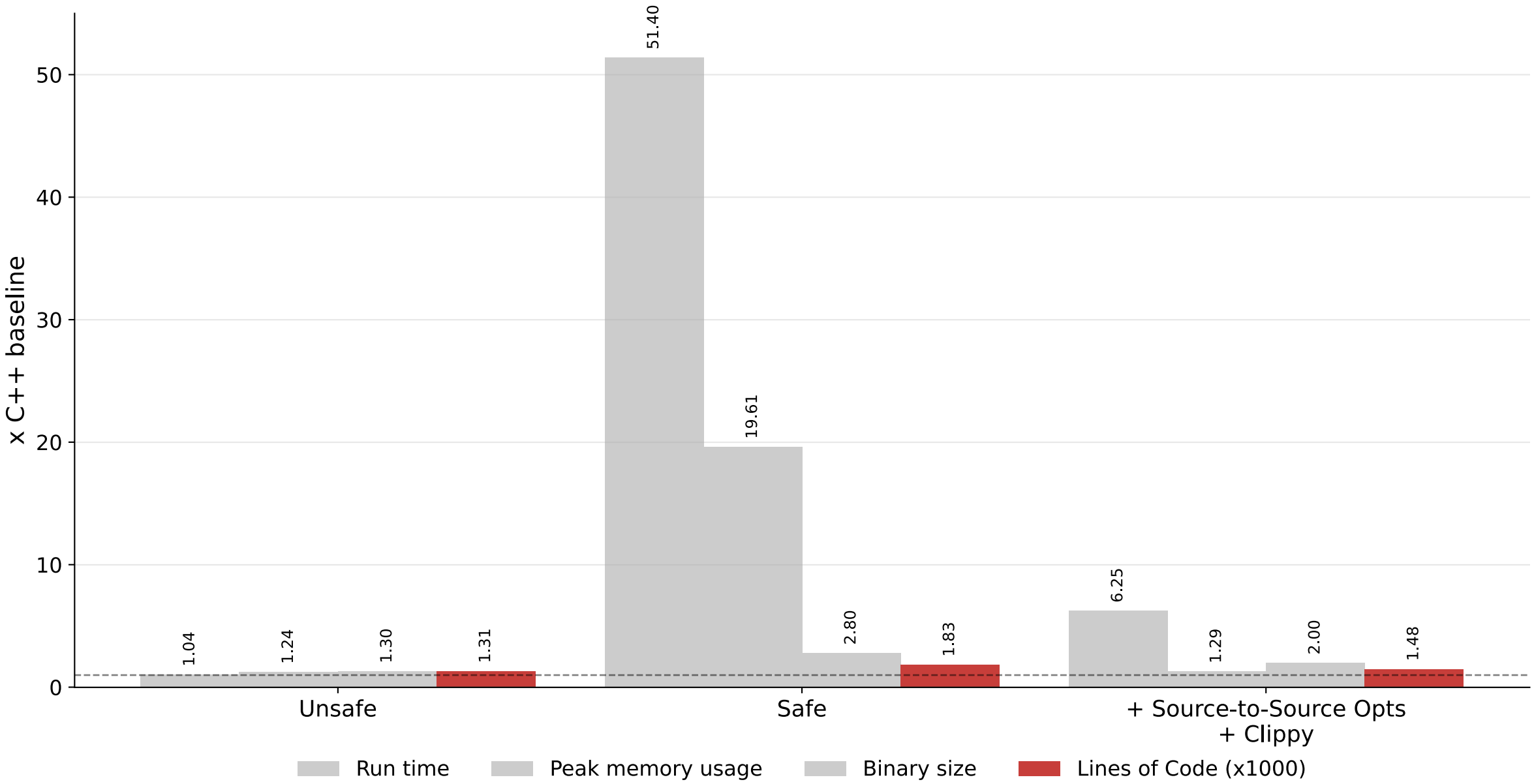
Brunsli: metrics relative to C++



Brunsli: metrics relative to C++



Brunsli: metrics relative to C++



github.com/cpp2rust/cpp2rust

Cpp2Rust: memory-safe Rust, near C++ performance for some applications

Open-source (try it 🫩)