Precision Tuning within a Memory-Safe Programming Language

Gabriele Magnani
gabriele.magnani@polimi.t

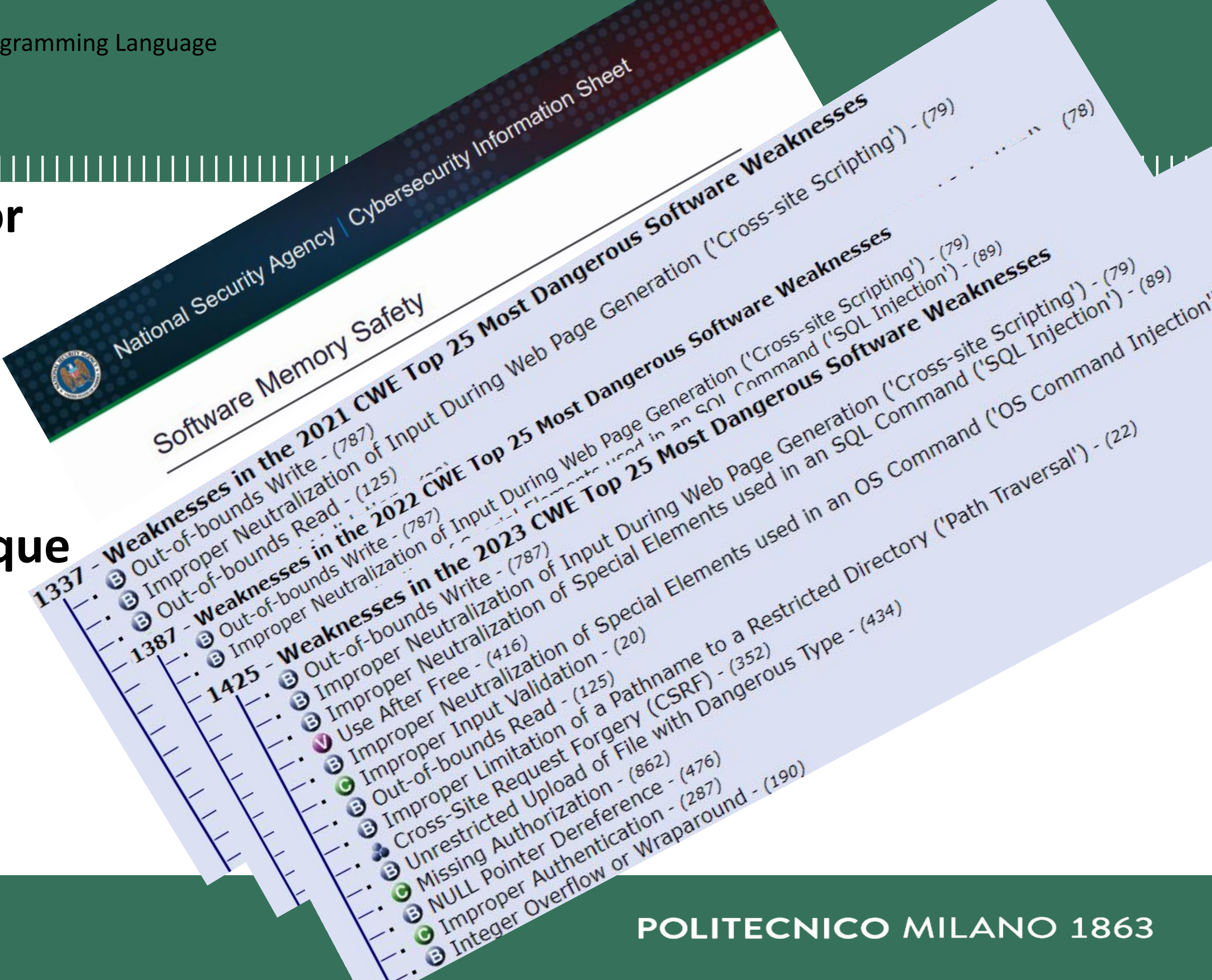
POLITECNICO MILANO 1863

Common Memory Error

1. Buffer overflow
2. Buffer over-read
3. Race condition
4. Use after free
5. Null pointer dereference

Memory Safety Technique

1. Garbage Collection
2. Array Bounds
3. Programmer annotations
4. Runtime checks


National Security Agency | Cybersecurity Information Sheet

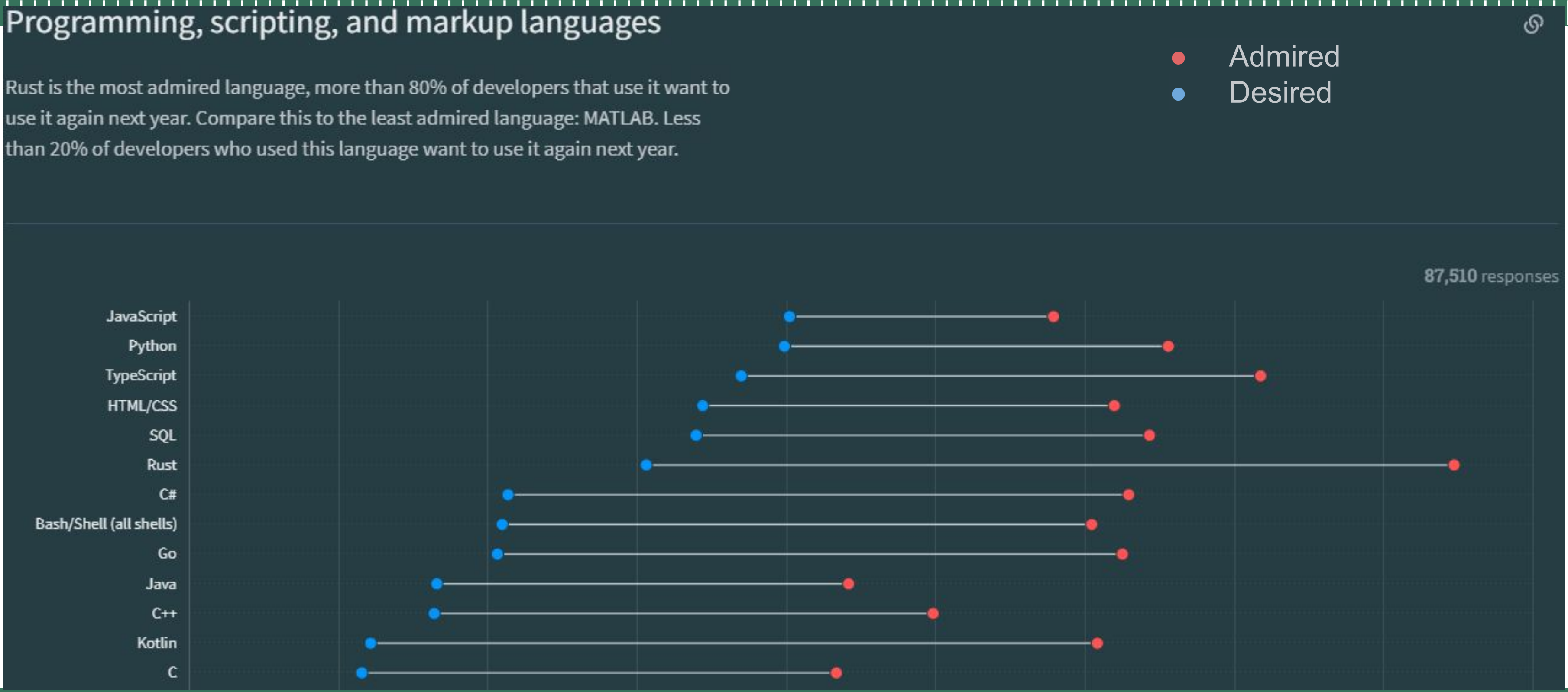
Software Memory Safety

1337 - Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses
- Out-of-bounds Write - (787)
- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - (79)
- Out-of-bounds Read - (125)
- (78)

1387 - Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses
- Out-of-bounds Write - (787)
- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - (79)
- SQL Command ('SQL Injection') - (89)

1425 - Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses
- Out-of-bounds Write - (787)
- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - (79)
- Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - (89)
- Use After Free - (416)
- Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'
- Improper Input Validation - (20)
- Out-of-bounds Read - (125)
- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - (22)
- Cross-Site Request Forgery (CSRF) - (352)
- Unrestricted Upload of File with Dangerous Type - (434)
- Missing Authorization - (862)
- NULL Pointer Dereference - (476)
- Improper Authentication - (287)
- Integer Overflow or Wraparound - (190)

Precision Tuning within a Memory-Safe Programming Language

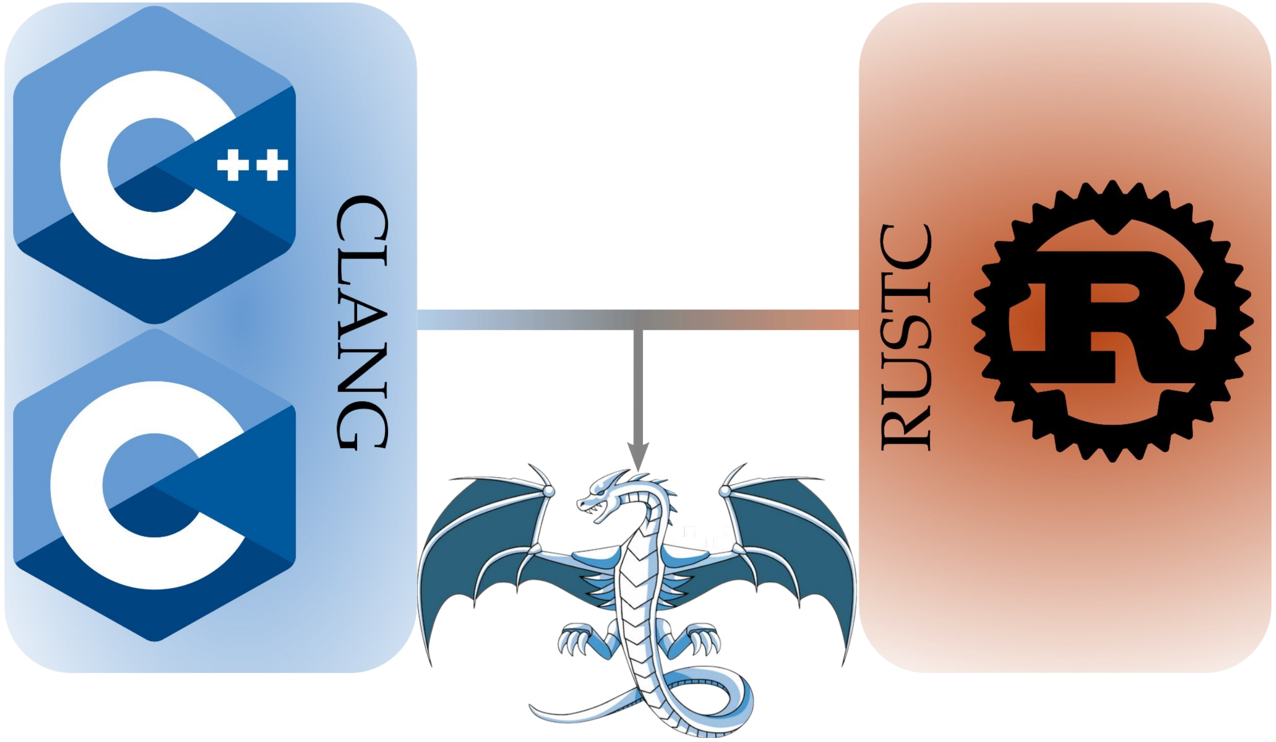


Programming, scripting, and markup languages

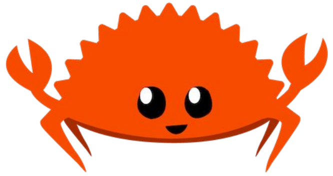
Rust is the most admired language, more than 80% of developers that use it want to use it again next year. Compare this to the least admired language: MATLAB. Less than 20% of developers who used this language want to use it again next year.

● Admired
● Desired

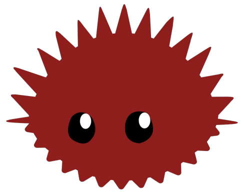
87,510 responses


CLANG
RUSTC

Rust Borrow checker

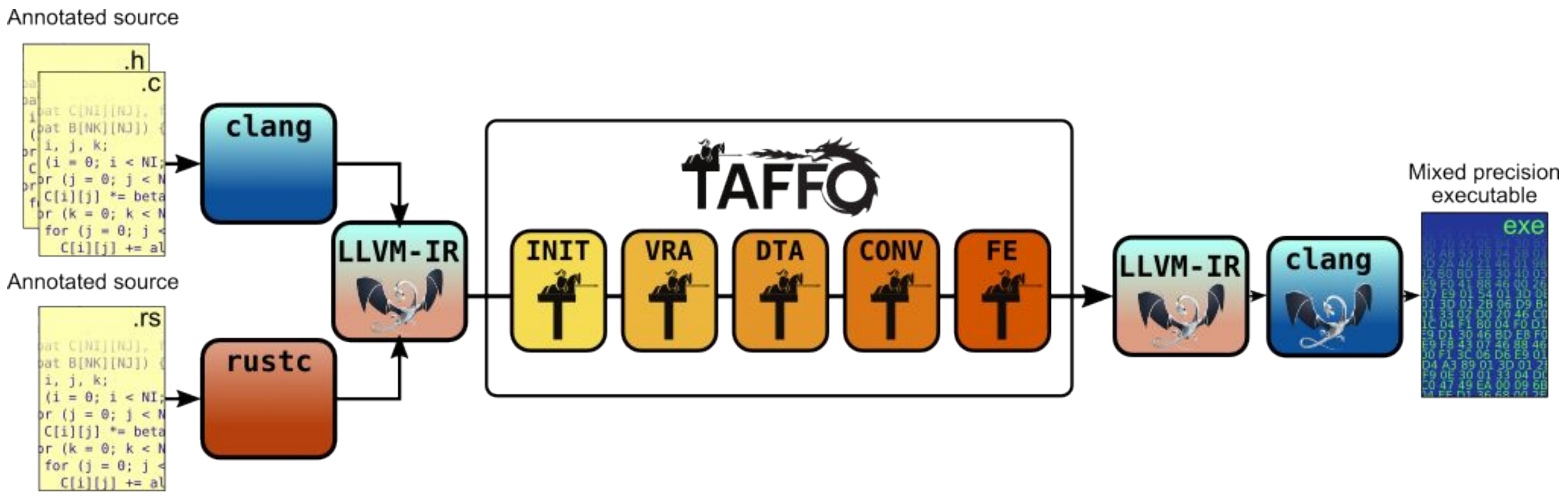


- All variables are initialized before use.
- Can't move the same value twice.
- Can't move a value while it is borrowed.
- Can't access a value while it is mutably borrowed (except through the reference).
- Can't mutate a value while it is immutably borrowed.



Rust Unsafe

- Dereference a raw pointer
- **Call an unsafe function or method**
- Access or modify a mutable static variable
- Implement an unsafe trait
- Access fields of unions


Annotated source
.h
.c
clang
Annotated source
.rs
rustc
LLVM-IR
TAFFO
INIT
VRA
DTA
CONV
FE
LLVM-IR
clang
Mixed precision executable
exe

TAFFO ANNOTATION

```c
int main(int argc, char *argv[]) {
float  __attribute__((annotate("scalar(range(-16384, 16384) final)"))) x_var[i][j];
…
}
```

```llvm
call void @llvm.var.annotation(ptr %x_var, ptr @.str, ptr @.str.1, i32 9, ptr null)
```

TAFFO ANNOTATION

```
fn main() {
annotate!(let mut x_var = [0_f32 ; I * J], "scalar(range(-16384, 16384) final)");
…
}
```

```
fn main() {
static mut range: &'static str = "scalar(range(-16384, 16384) final)";
static mut name: &'static str = "2mm.rs";
let mut x_var = [0_f32; I * J];
unsafe {
    var_annotation(&mut x_var as *mut _ as *mut i8, &mut range as *mut _ as *mut i8,
                           &mut name as *mut _ as *mut i8,  9 );
};}
```

TAFFO ANNOTATION

```
fn main() {
static mut range: &'static str = "scalar(range(-16384, 16384) final)";
static mut name: &'static str = "2mm.rs";
let mut x_var = [0_f32; I * J];
unsafe {
    var_annotation(&mut x_var as *mut _ as *mut i8, &mut range as *mut _ as *mut i8,
                        &mut name as *mut _ as *mut i8,  2 );
};}
```

```
call void @llvm.var.annotation(ptr %x_var, ptr @.str, ptr @.str.1, i32 9, ptr null)
```

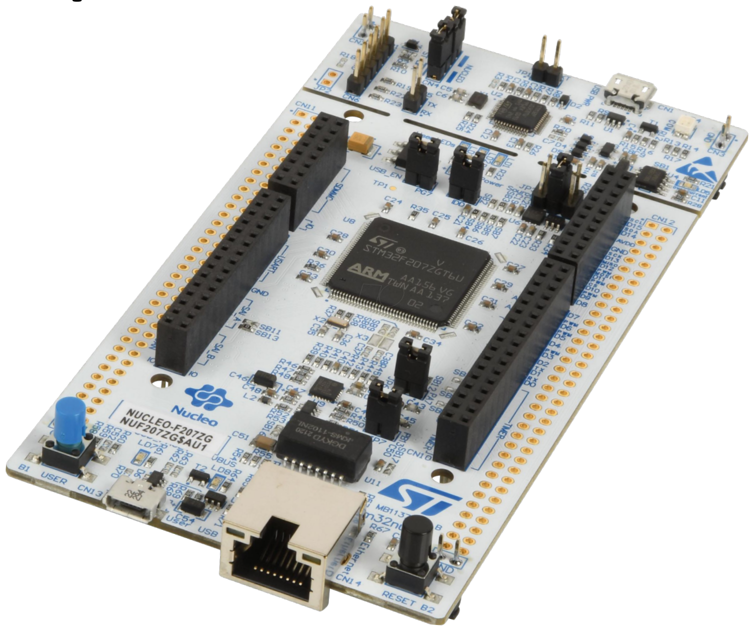
Soundness

- **unsafe**
  - Non-code-generating intrinsic

- **TAFFO**
  - Doesn't Increase the size of the elements of an array.
  - Doesn't change the size of a dynamic memory allocation.

Experimental Setup

- STM32F207ZG
  - ARM® Cortex®-M3 32-bit RISC 120 MHz
  - Flash memory 1 Mbyte
  - RAM 128 KBytes
  - No FPU
- Clang 15.0.7
- Rustc 1.64.0
  - -Z mir-opt-level
- Opt 15.0.7
- PolyBench 4.2.1



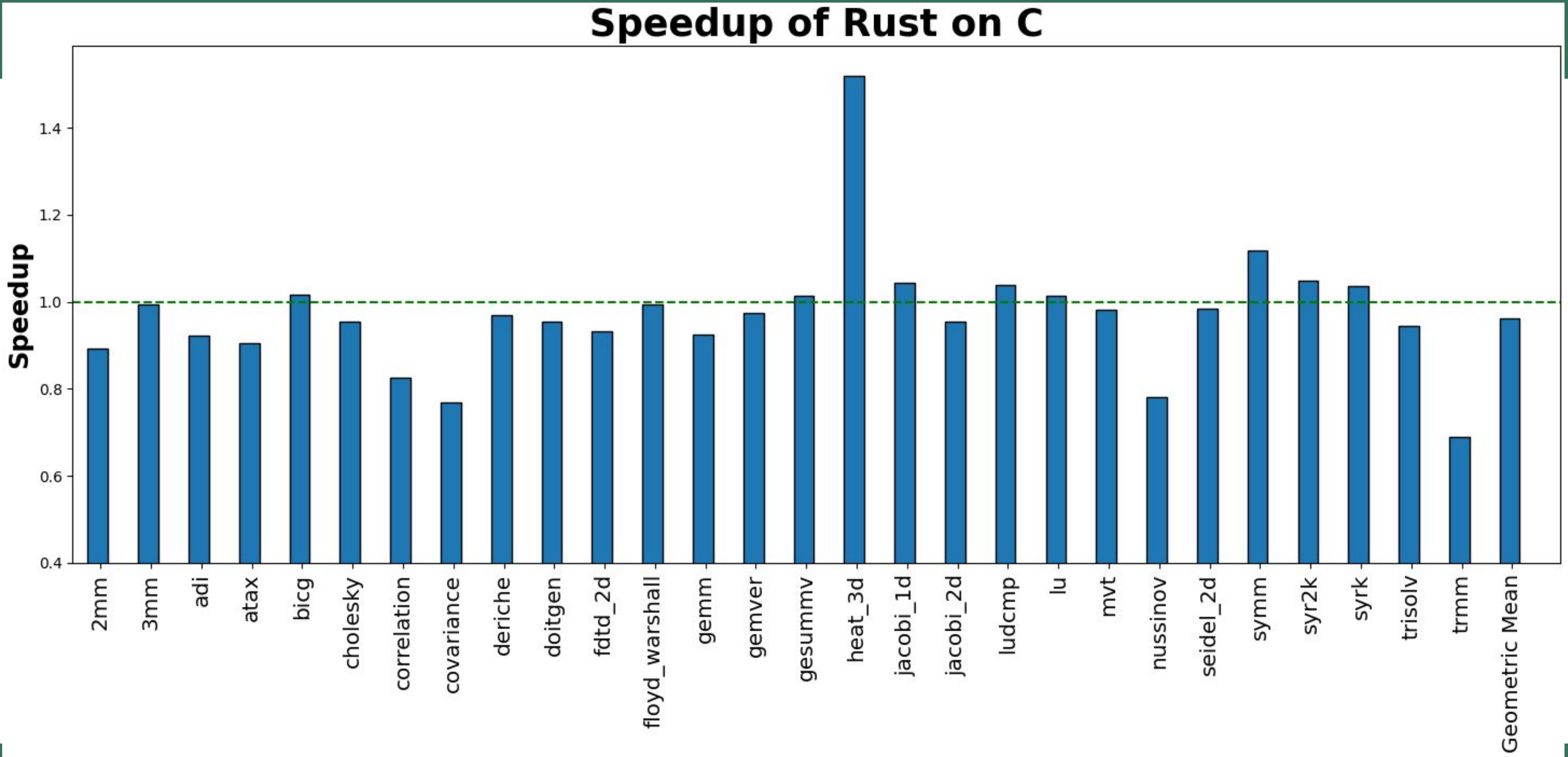
PolyBench-rs Unsafe Code

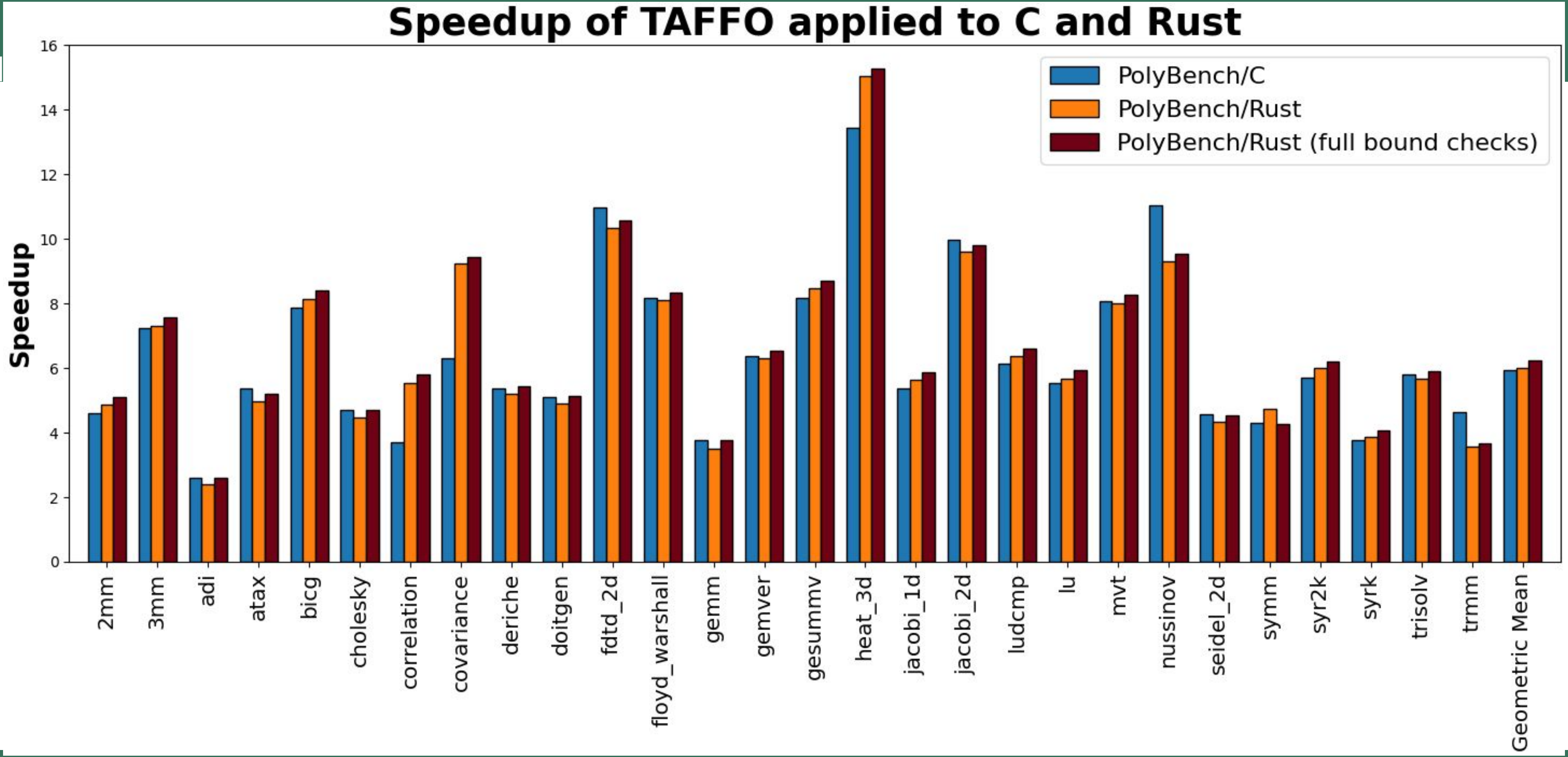
```
unsafe fn kernel_gemm
<const NI: usize, const NJ: usize, const NK: usize>(
    ni: usize,
    nj: usize,
    nk: usize,
    alpha: DataType,
    beta: DataType,
    C: &mut Array2D<DataType, NI, NJ>,
    A: &Array2D<DataType, NI, NK>,
    B: &Array2D<DataType, NK, NJ>,
)
```

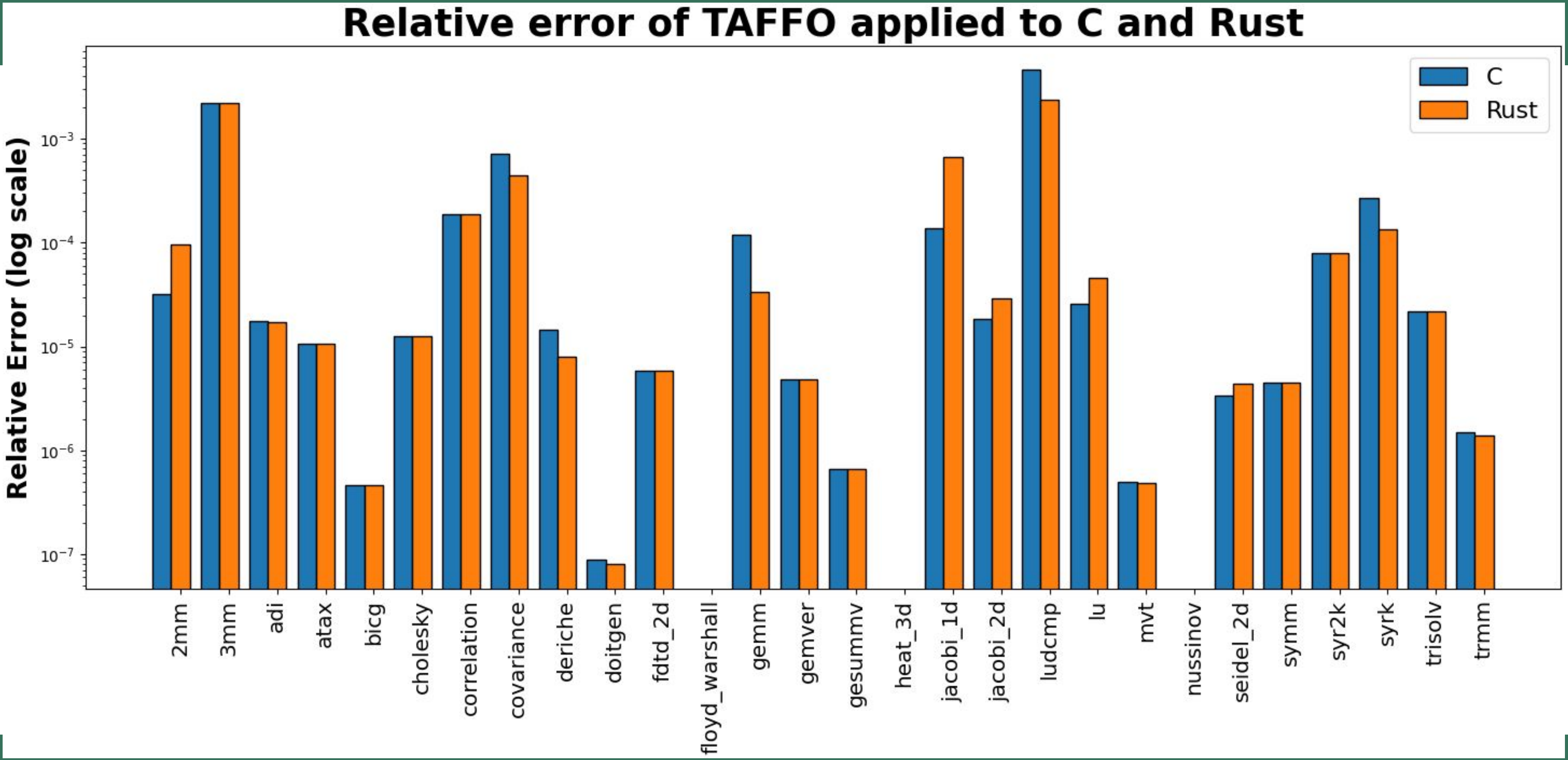
```
fn uninit() -> Box<Self> {
    let layout = std::alloc::Layout::new::<Self>();
    unsafe {
       let raw = std::alloc::alloc(layout) as *mut Self;
       Box::from_raw(raw)
     }
    }
```

PolyBench/C	PolyBench-rs

`define void @kernel_gemm([1024 x float]* %C, [1024 x float]* %A, [1024 x float]* %B) {`
 	
`…`
 	
`%0 = getelementptr inbounds [1024 x float], [1024 x float]* %C, i64 %indvars.iv9, i64 %indvars.iv5`
`%1 = load float, float* %arrayidx5,`
`%2 = fmul float %1, 1.200000e+00`
	`define void kernel_gemm(`
	`%"polybench_rs::ndarray::Array2D<f32, 1024_usize, 1024_usize>"* %C,`
	`%"polybench_rs::ndarray::Array2D<f32, 1024_usize, 1024_usize>"* %A,`
	`%"polybench_rs::ndarray::Array2D<f32, 1024_usize, 1024_usize>"* %B) {`
	
	`…`
	
	`%0 = getelementptr inbounds`
	`%"polybench_rs::ndarray::Array2D<f64, 1024_usize, 1024_usize>",`
	`%"polybench_rs::ndarray::Array2D<f64, 1024_usize, 1024_usize>"* %C,`
	`i64 0, i32 0, i64 %iter.sroa.0.091.us.us, i32 0,`
	`i64 %iter2.sroa.0.090.us.us.us`
	`%1 = load float, float* %0, align 8`
	`%2 = fmul float%1, 1.200000e+00`


Speedup of Rust on C
Speedup
1.4
1.2
1.0
0.8
0.6
0.4
2mm
3mm
adi
atax
bicg
cholesky
correlation
covariance
deriche
doitgen
fdtd_2d
floyd_warshall
gemm
gemver
gesummv
heat_3d
jacobi_1d
jacobi_2d
ludcmp
lu
mvt
nussinov
seidel_2d
symm
syr2k
syrk
trisolv
trmm
Geometric Mean


Speedup of TAFFO applied to C and Rust
Speedup
0
2
4
6
8
10
12
14
16
PolyBench/C
PolyBench/Rust
PolyBench/Rust (full bound checks)
2mm
3mm
adi
atax
bicg
cholesky
correlation
covariance
deriche
doitgen
fdtd_2d
floyd_warshall
gemm
gemver
gesummv
heat_3d
jacobi_1d
jacobi_2d
ludcmp
lu
mvt
nussinov
seidel_2d
symm
syr2k
syrk
trisolv
trmm
Geometric Mean


Relative error of TAFFO applied to C and Rust
Relative Error (log scale)
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
C
Rust
2mm
3mm
adi
atax
bicg
cholesky
correlation
covariance
deriche
doitgen
fdtd_2d
floyd_warshall
gemm
gemver
gesummv
heat_3d
jacobi_1d
jacobi_2d
ludcmp
lu
mvt
nussinov
seidel_2d
symm
syr2k
syrk
trisolv
trmm

Conclusion

- Other **LLVM-IR** based languages
    - **Go , Swift**
- **Rust** language extensions for annotation

Thank You!