High Performance? RIIR? A Rust Logging Crate from Scratch

High Performance Tricks in Rust Logging Crate

Asuna

PLCT Lab

2025-07

Outline

1. Me

2. Motivations

3. Implementations

4. Inspirations

Outline

1. Me

2. Motivations

3. Implementations

4. Inspirations

1.1 Me

Hi, I’m Asuna 🍓

1.1 Me

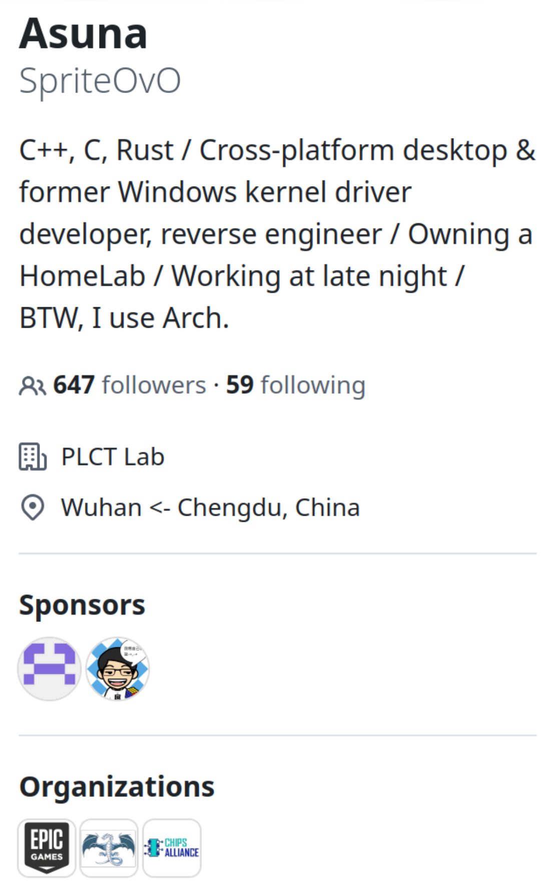
Hi, I’m Asuna 🍓

1.1 Me

Hi, I’m Asuna 🍓 a.k.a. A 宝 on Telegram

1.1 Me

@SpriteOvO
It’s also me, on GitHub



Asuna
SpriteOvO

C++, C, Rust / Cross-platform desktop & former Windows kernel driver developer, reverse engineer / Owning a HomeLab / Working at late night / BTW, I use Arch.

647 followers · **59** following

PLCT Lab

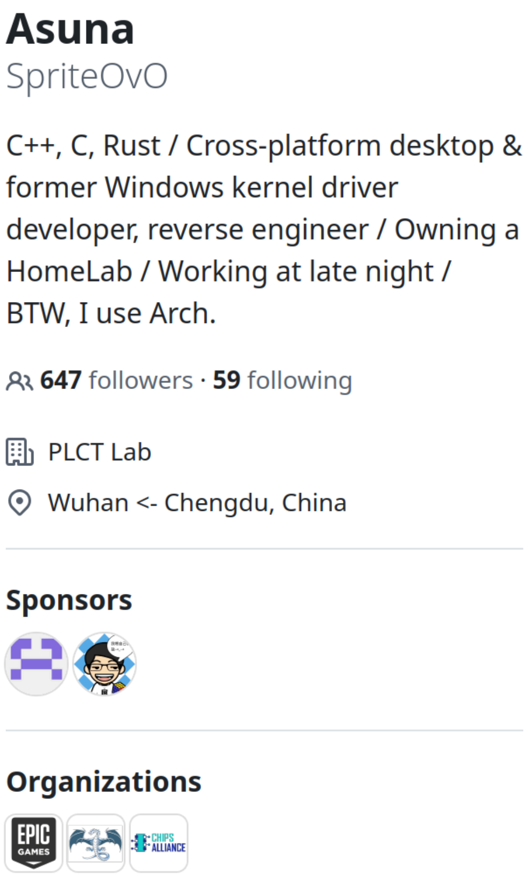
Wuhan <- Chengdu, China

Sponsors

Organizations

1.1 Me

@SpriteOvO
It's also me, on GitHub



Asuna
SpriteOvO

C++, C, Rust / Cross-platform desktop & former Windows kernel driver developer, reverse engineer / Owning a HomeLab / Working at late night / BTW, I use Arch.

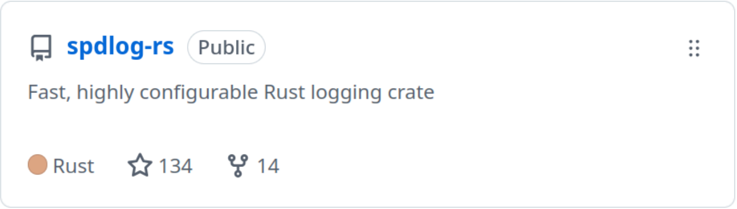
647 followers · **59** following

PLCT Lab

Wuhan <- Chengdu, China

Sponsors

Organizations



spdlog-rs Public

Fast, highly configurable Rust logging crate

● Rust ☆ 134 14

Outline

1. Me

2. **Motivations**

3. Implementations

4. Inspirations

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

in C++ (Stream operators & Input/Output manipulators)

```cpp
#include <iostream>
std::cout << "magic number: 0x" << std::uppercase << std::hex << magic << '\n';
std::cerr << "error occured: " << message << '\n';
```

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

in C++ (Stream operators & Input/Output manipulators)

```cpp
#include <iostream>
std::cout << "magic number: 0x" << std::uppercase << std::hex << magic << '\n';
std::cerr << "error occured: " << message << '\n';
```

Drawbacks

- Basic, but only basic.
- Poor flexibility:
  - just 2 severity levels - `stdout` & `stderr`,
  - file stream based,
- Ugly syntax.

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

in C++ (Stream operators & Input/Output manipulators)

```cpp
#include <iostream>
std::cout << "magic number: 0x" << std::uppercase << std::hex << magic << '\n';
std::cerr << "error occured: " << message << '\n';
```

Drawbacks

- Basic, but only basic.
- Poor flexibility:
  - just 2 severity levels - `stdout` & `stderr`,
  - file stream based,
- Ugly syntax.

Let's just output a floating point

```cpp
std::cout << "flout: " << std::fixed << std::showpos
<< std::setw(8) << std::setprecision(2) << f <<
std::endl;
```

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

in C++ (Stream operators & Input/Output manipulators)

```cpp
#include <iostream>
std::cout << "magic number: 0x" << std::uppercase << std::hex << magic << '\n';
std::cerr << "error occured: " << message << '\n';
```

Drawbacks

- Basic, but only basic.
- Poor flexibility:
  - just 2 severity levels - `stdout` & `stderr`,
  - file stream based,
- Ugly syntax.

2.1 Simplest Logging

in C (Format specifiers)

```c
#include <stdio.h>
printf("magic number: 0x%X\n", magic);
fprintf(stderr, "error occured: %s\n", message);
```

in C++ (Stream operators & Input/Output manipulators)

```cpp
#include <iostream>
std::cout << "magic number: 0x" << std::uppercase << std::hex << magic << '\n';
std::cerr << "error occured: " << message << '\n';
```

Drawbacks

- Basic, but only basic.
- Poor flexibility:
  - just 2 severity levels - `stdout` & `stderr`,
  - file stream based,
- ~~Ugly syntax.~~ Okay, looks better now.

in C++ (Python-like format, since C++23 standard)

```cpp
#include <print>
std::println("magic number: 0x{:X}", magic);
std::println(stderr, "error occured: {}", message);
```

2.2 Logging Libraries in C++

2.2 Logging Libraries in C++

`gabime/spdlog` - Fast C++ logging library.

Features

- Very fast.
- Formatting syntax powered by `fmt` library.
- Various logging targets.
- Asynchronous, multi/single threaded.
- ...

Basic usage

```cpp
#include <spdlog/spdlog.h>
spdlog::info("magic number: 0x{:X}", magic);
spdlog::error("error occured: {}", message);
spdlog::get("custom")->info("multiple loggers");
```

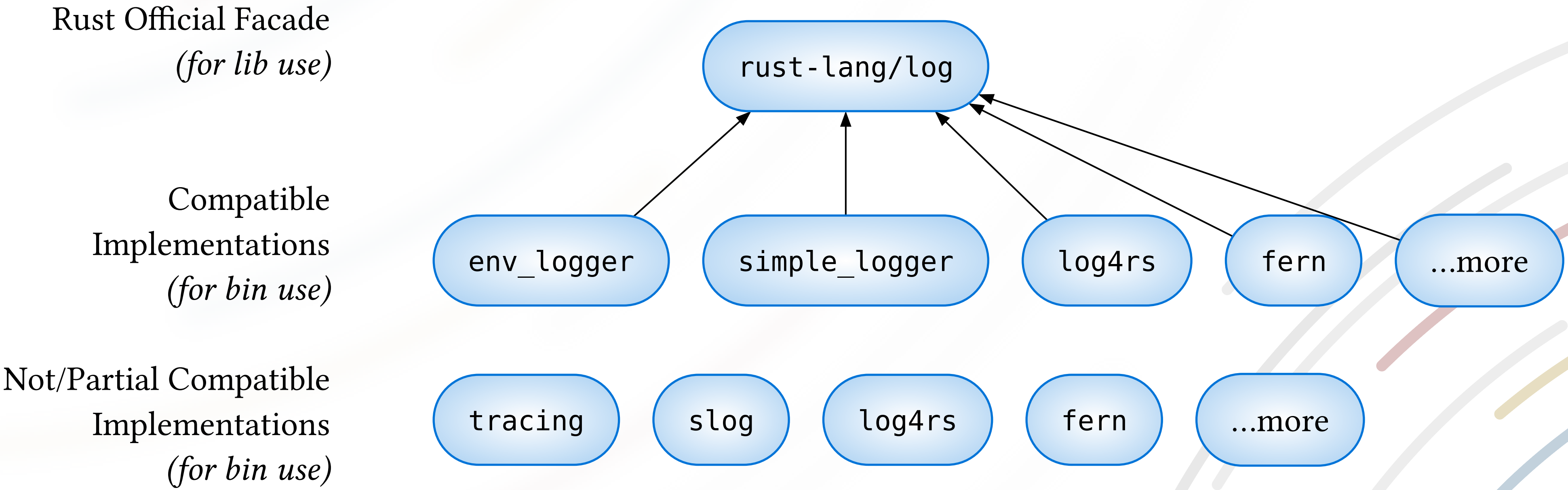
Rich & various configurable targets

```cpp
// Create a daily logger
auto logger = spdlog::daily_logger_mt("daily", "logs/daily.txt", 2, 30);
// Create a rotating by size logger
auto logger = spdlog::rotating_logger_mt("rotating", "logs/rotating.txt", 1048576 * 5, 3);
// Asynchronous logger
auto logger = spdlog::basic_logger_mt<spdlog::async_factory>("async", "logs/async.txt");
// ... more
```

2.3 Logging Crates Ecosystem in Rust

2.3 Logging Crates Ecosystem in Rust

Compatability with `rust-lang/log` crate



Outline

1. Me
2. Motivations
3. **Implementations**
4. Inspirations

3.1 Challenges

Appearance?

Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

3.1 Challenges

Appearance?

Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

Appearance is not a problem really

3.1 Challenges

Appearance?

Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

Appearance is not a problem really, but

- **Feature-rich**
  Outputs to `stdout`/`stderr`, file, rotating files (at a specific time or by file size), OS-native (`journald` on Linux, `windbg` on Windows), etc.

3.1 Challenges

Appearance?

Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

Appearance is not a problem really, but

- **Feature-rich**

  Outputs to `stdout`/`stderr`, file, rotating files (at a specific time or by file size), OS-native (`journald` on Linux, `windbg` on Windows), etc.

- **Highly configurable**

  Users should be able to freely select the logging targets they need, and each feature should be configurable.

3.1 Challenges

Appearance?

Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

Appearance is not a problem really, but

- **Feature-rich**
  Outputs to `stdout`/`stderr`, file, rotating files (at a specific time or by file size), OS-native (`journald` on Linux, `windbg` on Windows), etc.
- **Highly configurable**
  Users should be able to freely select the logging targets they need, and each feature should be configurable.
- **Extensible**
  More than just built-in features, users should be able to implement their own logging targets.

3.1 Challenges

Appearance?

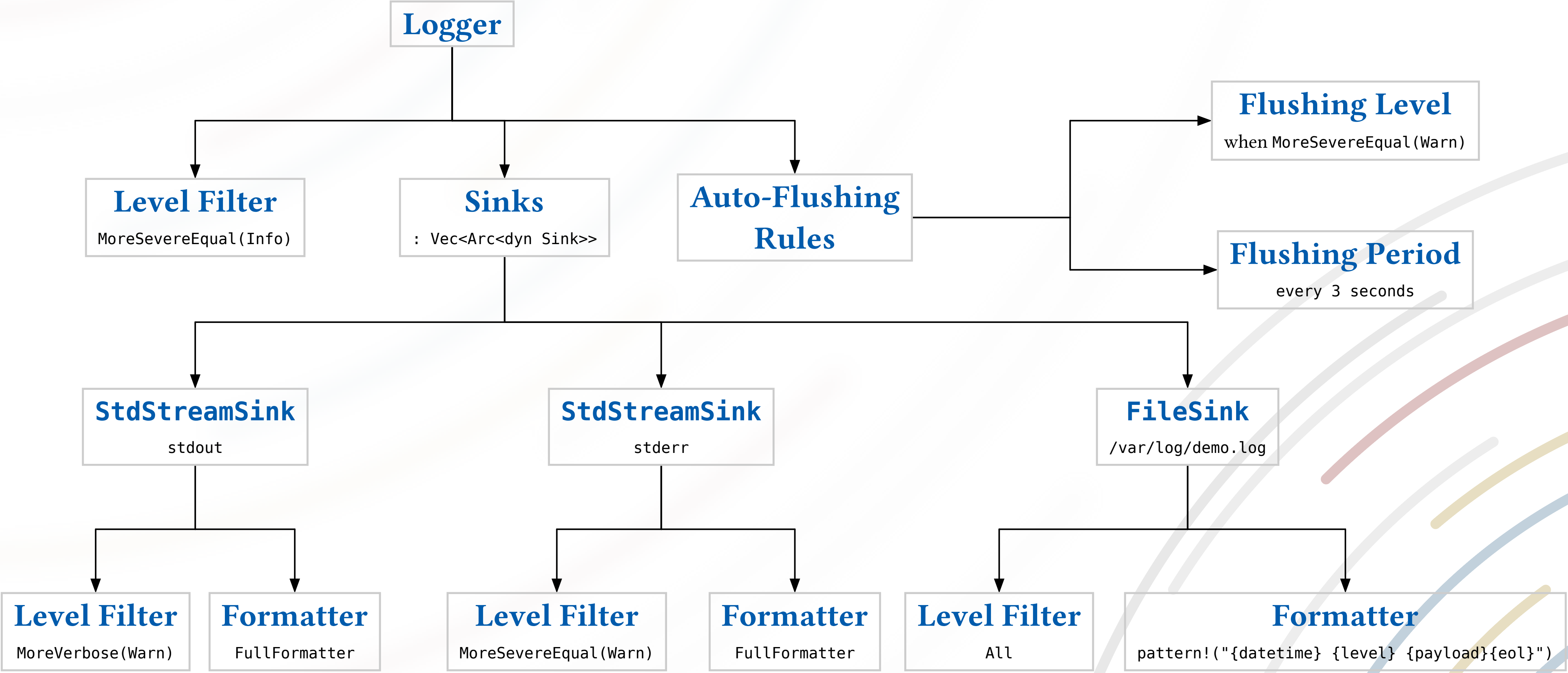
Logs are look like...

```
[2025-06-19 06:39:14.366] [info] [main.cpp:5] hello, world!
[2025-06-19 06:39:14.366] [error] [main.cpp:6] oops! something went wrong!
```

Appearance is not a problem really, but

- **Feature-rich**
  Outputs to `stdout`/`stderr`, file, rotating files (at a specific time or by file size), OS-native (`journald` on Linux, `windbg` on Windows), etc.
- **Highly configurable**
  Users should be able to freely select the logging targets they need, and each feature should be configurable.
- **Extensible**
  More than just built-in features, users should be able to implement their own logging targets.
- **and... Fast** 🚀 🚀 🚀

3.2 Architecture



3.3 Logger

rust-lang/log: Initialize logger at start-up

```rust
fn main() -> Result {
    let my_logger = Box::new(SimpleLogger::new());
    log::set_boxed_logger(my_logger)?;

    info!("hello, world!"); // will be handled by `SimpleLogger`
}
```

- **Singleton**

  Only one logger can be set and used, and cannot be replaced at runtime.

3.3 Logger

spdlog-rs: No need to initialize, and non-singleton

```
fn main() -> Result {
  let my_logger = Logger::builder() /* ... */ .build();
  spdlog::set_default_logger(my_logger)?;

  info!("hello, world!"); // will be handled by `my_logger`

  let another = Logger::builder() /* ... */ .build();
  info!(logger: another, "hello, world!"); // will be handled by `another`
}
```

3.4 File Writes

3.4 File Writes

Writing to a file…

using OS calls in C

```
int fd = open("/tmp/demo.log", O_RDWR | O_CREAT | O_APPEND, 0666);
write(fd, message, strlen(message));
close(fd);
```

using `libc` in C

```
FILE *fp = fopen("/tmp/demo.log", "a+");
fwrite(message, sizeof(message[0]), strlen(message), fp);
fclose(fp);
```

3.4 File Writes

Writing to a file...

using OS calls in C

```c
int fd = open("/tmp/demo.log", O_RDWR | O_CREAT | O_APPEND, 0666);
write(fd, message, strlen(message));
close(fd);
```

using `libc` in C

```c
FILE *fp = fopen("/tmp/demo.log", "a+");
fwrite(message, sizeof(message[0]), strlen(message), fp);
fclose(fp);
```

using STL in C++

```cpp
std::ofstream ofs{"/tmp/demo.log", std::ios::app};
ofs << message;
```

3.4 File Writes

Writing to a file in Rust

```rust
use std::fs::File;

let mut f = File::options().append(true).open("/tmp/demo.log")?;
f.write_all(message.as_bytes())?;
```

`std::fs::File` is unbuffered, all operations invoke syscalls directly.

3.4 File Writes

Writing to a file in Rust

```rust
use std::fs::File;

let mut f = File::options().append(true).open("/tmp/demo.log")?;
f.write_all(message.as_bytes())?;
```

`std::fs::File` is unbuffered, all operations invoke syscalls directly.

Writing to a file in Rust with buffering

```rust
use std::{fs::File, io::BufWriter};

let mut f = BufWriter::new(File::options().append(true).open("/tmp/demo.log")?);
f.write_all(message.as_bytes())?;
```

Wrap `std::fs::File` in `std::io::BufWriter` for buffering in userspace.

3.4 File Writes

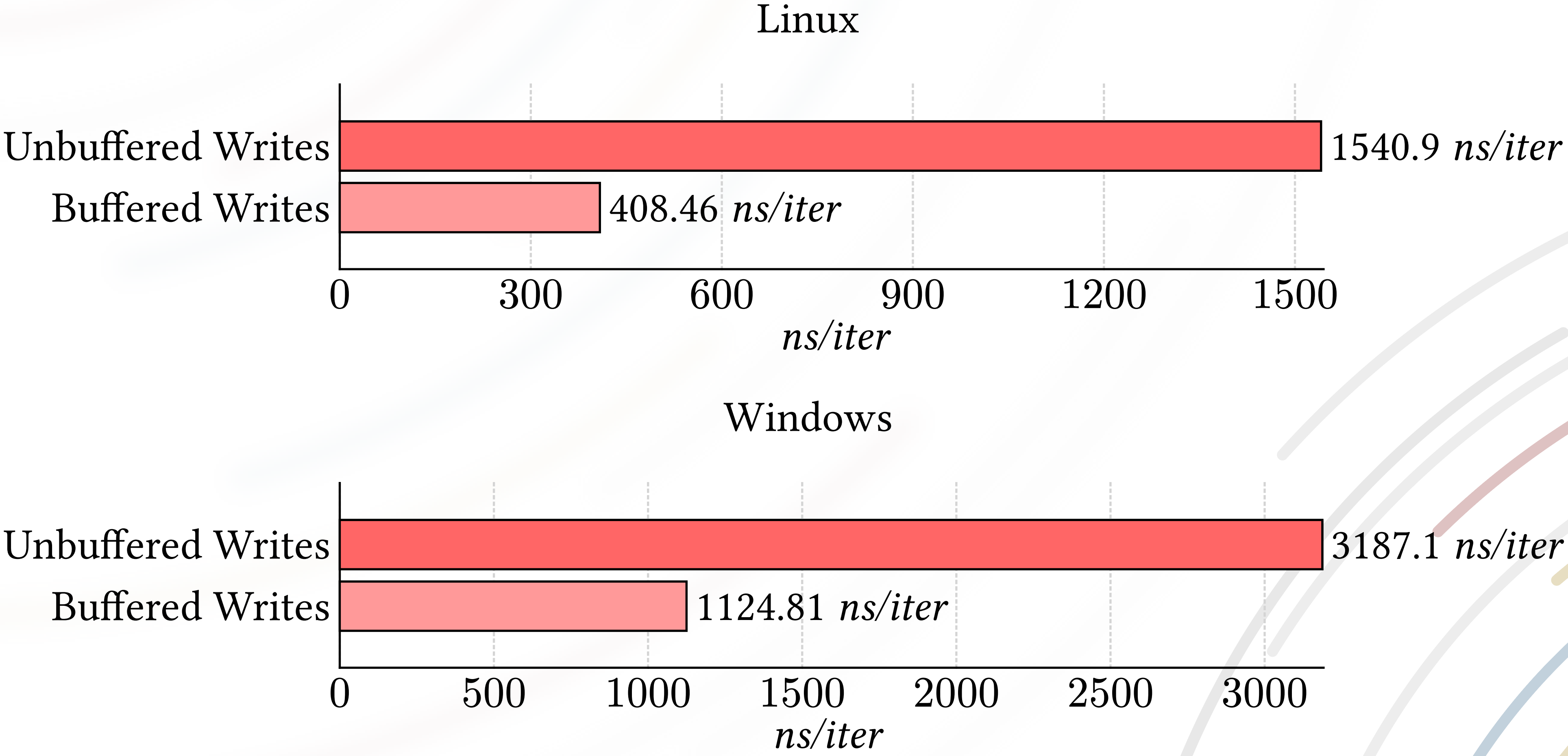
Unbuffered

- OS API in C, `std::fs::File` in Rust
- `open`, `read`, `write`, `close`, etc.
- Directly interacting with OS – manipulated via File Descriptor (`int`)

Buffered

- `libc` API in C, `BufWriter<File>` in Rust, buffered in userspace
- `fopen`, `fread`, `fwrite`, `fclose`, etc.
- Indirectly through a user-space encapsulation – manipulated via `FILE` pointer (`FILE *`)
- Syscalls will be called when the buffer is full, or when flushing is explicitly called.

3.4 File Writes



3.4 File Writes

3.4 File Writes

Implementation of `FileSink` (simplified)

```rust
pub struct FileSink {
    formatter: Box<dyn Formatter>,
    file: BufWriter<File>,
}

impl Sink for FileSink {
    fn log(&self, record: &Record) -> Result<()> {
        let mut string_buf = StringBuf::new();
        self.formatter.format(record, &mut string_buf, &mut ctx)?;

        self.file.write_all(string_buf.as_bytes())?;
    }

    fn flush(&self) -> Result<()> {
        self.file.flush()?;
    }
}
```

3.5 String Buffer

C: Allocating memory, stack or heap

```c
void use(void *buffer);

void on_stack(void) {
    uint8_t buffer[100];
    use(&buffer);
}

void on_heap(void) {
    uint8_t *buffer = malloc(100);
    use(buffer);
    free(buffer);
}
```

3.5 String Buffer

C: Allocating memory, stack or heap

```c
void use(void *buffer);

void on_stack(void) {
    uint8_t buffer[100];
    use(&buffer);
}

void on_heap(void) {
    uint8_t *buffer = malloc(100);
    use(buffer);
    free(buffer);
}
```

gcc -O2 compiles as

```
on_stack:
        sub     rsp, 120
        mov     rdi, rsp
        call    use
        add     rsp, 120
        ret
on_heap:
        push    rbx
        mov     edi, 100
        call    malloc
        mov     rbx, rax
        mov     rdi, rax
        call    use
        mov     rdi, rbx
        pop     rbx
        jmp     free
```

3.5 String Buffer

C: Allocating memory, stack or heap

```c
void use(void *buffer);

void on_stack(void) {
    uint8_t buffer[100];
    use(&buffer);
}

void on_heap(void) {
    uint8_t *buffer = malloc(100);
    use(buffer);
    free(buffer);
}
```

gcc -02 compiles as

```asm
on_stack:
        sub     rsp, 120
        mov     rdi, rsp
        call    use
        add     rsp, 120
        ret

on_heap:
        push    rbx
        mov     edi, 100
        call    malloc
        mov     rbx, rax
        mov     rdi, rax
        call    use
        mov     rdi, rbx
        pop     rbx
        jmp     free
```

3.5 String Buffer

Linux: in `glibc`

```
int __pthread_create_2_1 (
  pthread_t *newthread,
  const pthread_attr_t *attr,
  void *(*start_routine) (void *),
  void *arg)
{
  void *stackaddr = NULL;
  size_t stacksize = 0;

  /* ... */
  struct pthread *pd = NULL;
  int err = allocate_stack (
    iattr, &pd, &stackaddr, &stacksize);

  int retval = 0;
  if (__glibc_unlikely (err != 0)) {
    retval = err == ENOMEM ? EAGAIN : err;
    goto out;
  }
  /* ... */
}
```

Windows: in `kernel32.dll`

```
HANDLE WINAPI CreateRemoteThread(
    IN HANDLE hProcess,
    IN LPSECURITY_ATTRIBUTES lpThreadAttributes,
    IN DWORD dwStackSize,
    IN LPTHREAD_START_ROUTINE lpStartAddress,
    IN LPVOID lpParameter,
    IN DWORD dwCreationFlags,
    OUT LPDWORD lpThreadId)
{
    INITIAL_TEB InitialTeb;
    /* ... */
    Status = BaseCreateStack(
        hProcess,
        (dwCreationFlags & /* ... */) ? 0 : dwStackSize,
        (dwCreationFlags & /* ... */) ? dwStackSize : 0,
        &InitialTeb);
    if (!NT_SUCCESS(Status)) {
        BaseSetLastNTError(Status);
        return NULL;
    }
    /* ... */
}
```

3.5 String Buffer

From the perspective of

- **Performance of allocating**
  Stack is cheap, heap is expensive.
- **Capacity & Applicability**
  Stack is limited, heap is almost unlimited.

3.5 String Buffer

`spdlog`: Used the class provided by `fmt` library

```
template <class T>
class fmt::buffer {
    T *ptr;
    size_t size;
    // ...
};

template <class T, size_t SIZE>
class fmt::basic_memory_buffer : public fmt::buffer {
    T store[SIZE];

    basic_memory_buffer() : buffer(store, SIZE) { /* ... */ }
    ~basic_memory_buffer() {
      if (ptr != store) {
        delete[] ptr;
      }
    }
    // ...
};

using spdlog::memory_buf_t = fmt::basic_memory_buffer<char, 250>;
```

`fmt::buffer`

- a buffer view;
- only stores the pointer and size.

`fmt::basic_memory_buffer`

- inherited from `fmt::buffer`;
- owns and manages the buffer;
- contains an array on the stack as the initial buffer;
- reallocates on heap if usage exceeds `SIZE`.

`spdlog::memory_buf_t`

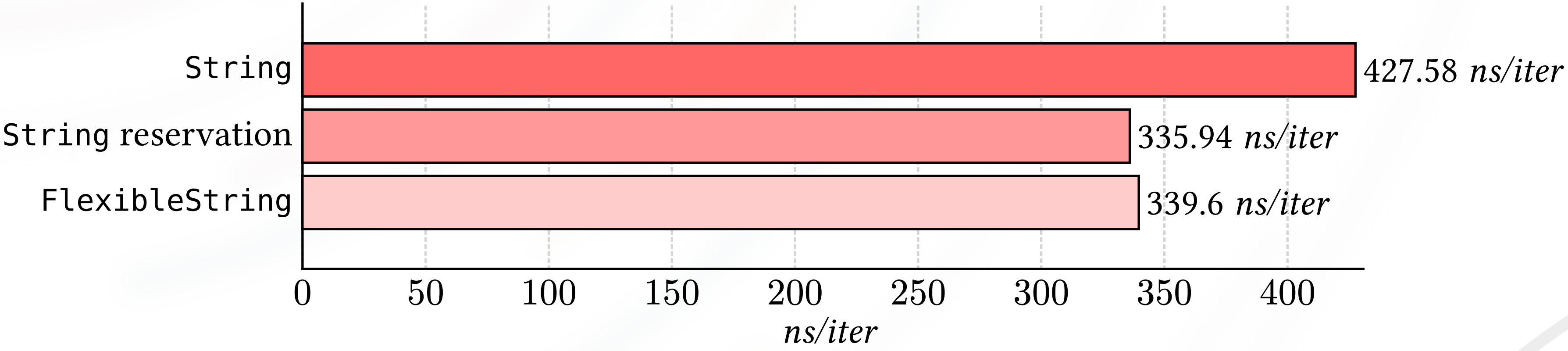
- a type alias;
- uses `char` as the buffer base type and specifies the fixed length of the array as 250;
- used as a string.

3.5 String Buffer

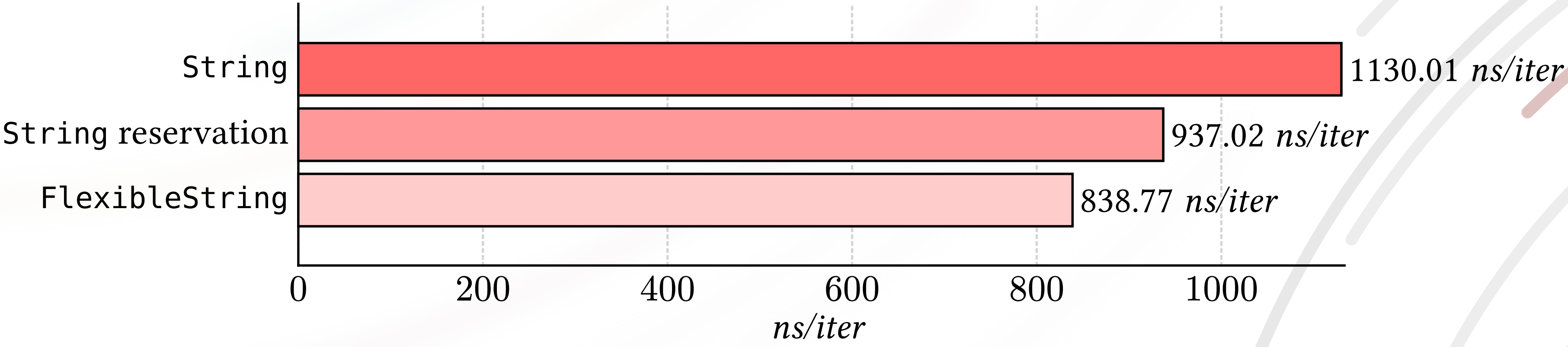
spdlog-rs: We published flexible-string crate

```rust
enum FlexibleString<const CAPACITY: usize> {
    Stack(StackString<CAPACITY>),
    Heap(String),
}

struct StackString<const CAPACITY: usize> {
    data: [MaybeUninit<u8>; CAPACITY],
    len: usize,
}

impl FlexibleString {
    // ... reimplement std String methods
}
```

3.5 String Buffer



↑ Linux ↓ Windows



3.6 Date Time

UTC Date Time

$$\underset{\text{Year}}{\boxed{2025}} - \underset{\text{Month}}{\boxed{07}} - \underset{\text{Day}}{\boxed{26}} \quad \underset{\text{Hour}}{\boxed{03}} : \underset{\text{Minute}}{\boxed{45}} : \underset{\text{Second}}{\boxed{14}} \,.\, \underset{\text{Millisecond}}{\boxed{191}}$$

3.6 Date Time

UTC Date Time

2025	-	07	-	26	03	:	45	:	14	.	191
Year		Month		Day	Hour		Minute		Second		Millisecond

Local Date Time

e.g. China Standard Time (CST)

2025	-	07	-	26	11	:	45	:	14	.	191	+08:00
Year		Month		Day	Hour		Minute		Second		Millisecond	Timezone Offset

3.6 Date Time

When you Google “C get local time”,
this is the first StackOverflow answer, and also what the Google AI tells you.

```c
#include <time.h>

time_t now = time(NULL);
struct tm *local = localtime(&now);

char *display = asctime(local);
printf("now: %s\n", display);
```

3.6 Date Time

When you Google “C get local time”,
this is the first StackOverflow answer, and also what the Google AI tells you.

```c
#include <time.h>

time_t now = time(NULL);
struct tm *local = localtime(&now);

char *display = asctime(local);
printf("now: %s\n", display);
```

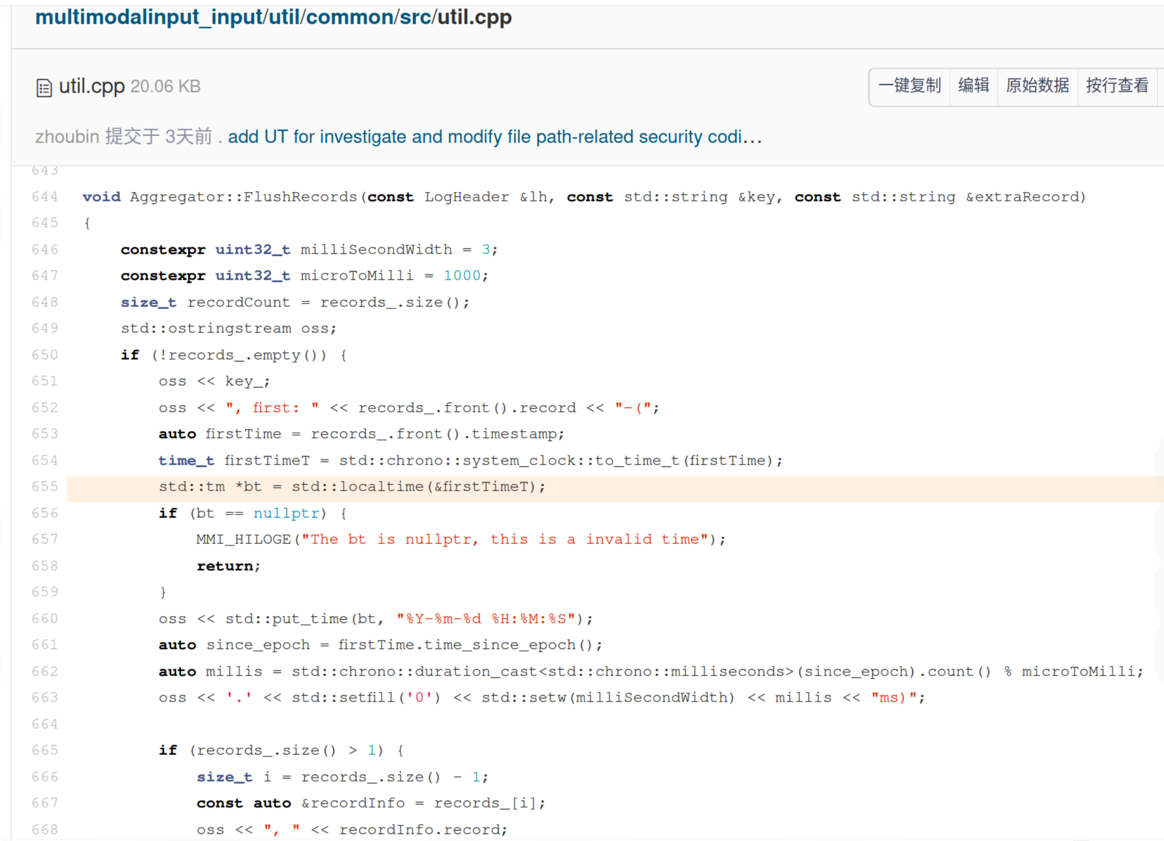
Did you notice anything wrong?

3.6 Date Time

Implementation of `localtime` and `asctime` in `musl` library

```c
struct tm *localtime(const time_t *t)
{
  static struct tm tm;
  return __localtime_r(t, &tm);
}

char *asctime(const struct tm *tm)
{
  static char buf[26];
  return __asctime_r(tm, buf);
}
```

3.6 Date Time



multimodalinput_input/util/common/src/util.cpp

util.cpp 20.06 KB

zhoubin 提交于 3天前 . add UT for investigate and modify file path-related security codi...

```cpp
void Aggregator::FlushRecords(const LogHeader &lh, const std::string &key, const std::string &extraRecord)
{
    constexpr uint32_t milliSecondWidth = 3;
    constexpr uint32_t microToMilli = 1000;
    size_t recordCount = records_.size();
    std::ostringstream oss;
    if (!records_.empty()) {
        oss << key_;
        oss << ", first: " << records_.front().record << "-(";
        auto firstTime = records_.front().timestamp;
        time_t firstTimeT = std::chrono::system_clock::to_time_t(firstTime);
        std::tm *bt = std::localtime(&firstTimeT);
        if (bt == nullptr) {
            MMI_HILOGE("The bt is nullptr, this is a invalid time");
            return;
        }
        oss << std::put_time(bt, "%Y-%m-%d %H:%M:%S");
        auto since_epoch = firstTime.time_since_epoch();
        auto millis = std::chrono::duration_cast<std::chrono::milliseconds>(since_epoch).count() % microToMilli;
        oss << '.' << std::setfill('0') << std::setw(milliSecondWidth) << millis << "ms)";

        if (records_.size() > 1) {
            size_t i = records_.size() - 1;
            const auto &recordInfo = records_[i];
            oss << ", " << recordInfo.record;
```

Figure 1: Unsafe `localtime` function is used in some 开放和谐 OS

3.6 Date Time

In practice, we should avoid using such unsafe API

```c
time_t now = time(NULL);

struct tm local;
localtime_s(&now, &local);

char display[128];
asctime_s(display, sizeof(display), &local);
printf("now: %s\n", display);
```

[1]

[2]

3.6 Date Time

In practice, we should avoid using such unsafe API

```c
time_t now = time(NULL);

struct tm local;
localtime_s(&now, &local);

char display[128];
asctime_s(display, sizeof(display), &local);
printf("now: %s\n", display);
```

… Are we doing things correctly now?

[1]

[2]

3.6 Date Time

In practice, we should avoid using such unsafe API

```
time_t now = time(NULL);

struct tm local;
localtime_s(&now, &local);

char display[128];
asctime_s(display, sizeof(display), &local);
printf("now: %s\n", display);
```

… Are we doing things correctly now? – Still no.

- _s suffixed API are C standard but optionally implemented, for some reason[1] GCC/Clang has chosen not to implement some of them.
- GCC/Clang provides _r suffixed versions (e.g. `asctime_r`, `ctime_r`).
- Among the time-related _r suffixed functions, some are C standard and some are POSIX-only.

[1] StackOverflow Question/79710594: Are `asctime_r` and `ctime_r` standard in C?

[2]

3.6 Date Time

In practice, we should avoid using such unsafe API

```c
time_t now = time(NULL);

struct tm local;
localtime_s(&now, &local);

char display[128];
asctime_s(display, sizeof(display), &local);
printf("now: %s\n", display);
```

... Are we doing things correctly now? – Still no.

- _s suffixed API are C standard but optionally implemented, for some reason[1] GCC/Clang has chosen not to implement some of them.
- GCC/Clang provides _r suffixed versions (e.g. `asctime_r`, `ctime_r`).
- Among the time-related _r suffixed functions, some are C standard and some are POSIX-only.

Okay, okay, the date time API part of the C standard is just a *mess*.
If you really need to work with date time in C, please go ICU – not Intensive Care Unit, it's ICU4C[2] library.

[1] StackOverflow Question/79710594: Are `asctime_r` and `ctime_r` standard in C?

[2] unicode-org/icu

3.6 Date Time

Fortunately, today we have more modern date time API in C++ and Rust.

C++: `std::chrono`, since C++20 standard

```cpp
#include <chrono>
using namespace std::chrono;

auto now = system_clock::now();
auto local = current_zone()->to_local(now);
```

Rust: `std::time` + `chrono` crate

```rust
use chrono::prelude::*;

let now = std::time::SystemTime::now();
let local: DateTime<Local> = now.into();
```

3.6 Date Time

Fortunately, today we have more modern date time API in C++ and Rust.

C++: `std::chrono`, since C++20 standard

```cpp
#include <chrono>
using namespace std::chrono;

auto now = system_clock::now();
auto local = current_zone()->to_local(now);
```

Rust: `std::time` + `chrono` crate

```rust
use chrono::prelude::*;

let now = std::time::SystemTime::now();
let local: DateTime<Local> = now.into();
```

Problem again, the conversion from UTC to Local is very slow.

3.6 Date Time

UTC Date Time

2025	-	07	-	26		03	:	45	:	14	.	191
Year		Month		Day		Hour		Minute		Second		Millisecond

Local Date Time

e.g. China Standard Time (CST)

2025	-	07	-	26		11	:	45	:	14	.	191	+08:00
Year		Month		Day		Hour		Minute		Second		Millisecond	Timezone Offset

3.6 Date Time

UTC Date Time

2025	-	07	-	26	03	:	45	:	14	.	191
Year		Month		Day	Hour		Minute		Second		Millisecond

Local Date Time

e.g. China Standard Time (CST)

2025	-	07	-	26	11	:	45	:	14	.	191	+08:00
Year		Month		Day	Hour		Minute		Second		Millisecond	Timezone Offset

3.6 Date Time

UTC Date Time

2025	-	07	-	26		03	:	45	:	14	.	191
Year		Month		Day		Hour		Minute		Second		Millisecond

Local Date Time

e.g. China Standard Time (CST)

2025	-	07	-	26		11	:	45	:	14	.	191		+08:00
Year		Month		Day		Hour		Minute		Second		Millisecond		Timezone Offset

Solution:

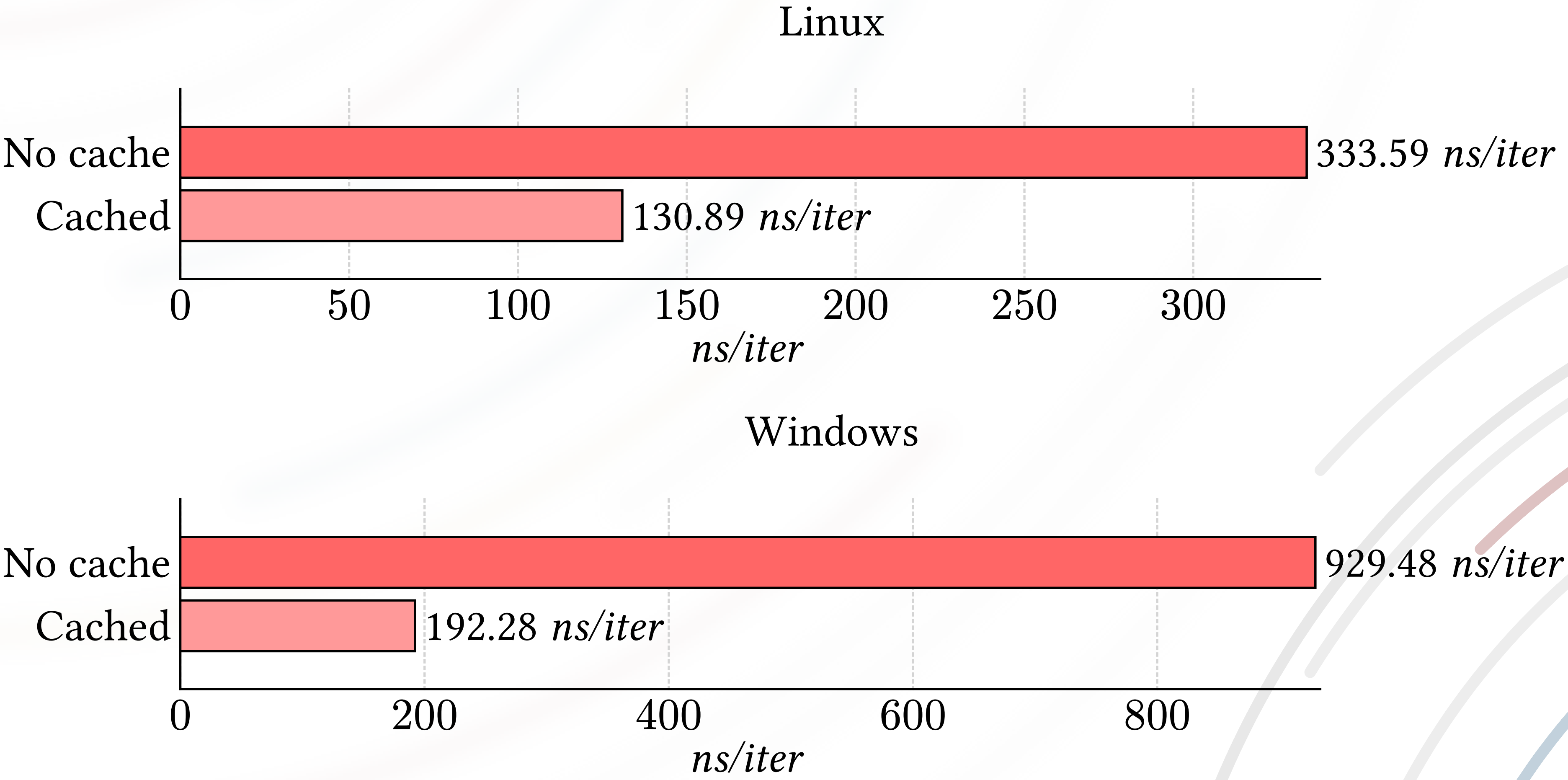
Cache is your friend.

3.6 Date Time

Cache parts before “second” of Local, combined with “millisecond” of UTC

```
struct TimeCacher {
    last_secs: u64,
    cached_local: DateTime<Local>,
}

impl TimeCacher {
    fn get(&mut self, utc: SystemTime) -> (YearMonthDayHourMinSec, Ms) {
        let since_epoch = utc.duration_since(UNIX_EPOCH);
        if since_epoch.as_secs() != self.last_secs {
            self.cached_local = utc.into();
            self.last_secs = since_epoch.as_secs();
        }

        (self.cached_local.year_month_day_hour_min_sec(), utc.subsec_millis())
    }
}
```

3.6 Date Time


Linux
No cache
Cached
333.59 ns/iter
130.89 ns/iter
0
50
100
150
200
250
300
ns/iter
Windows
No cache
Cached
929.48 ns/iter
192.28 ns/iter
0
200
400
600
800
ns/iter

Outline

1. Me

2. Motivations

3. Implementations

4. **Inspirations**

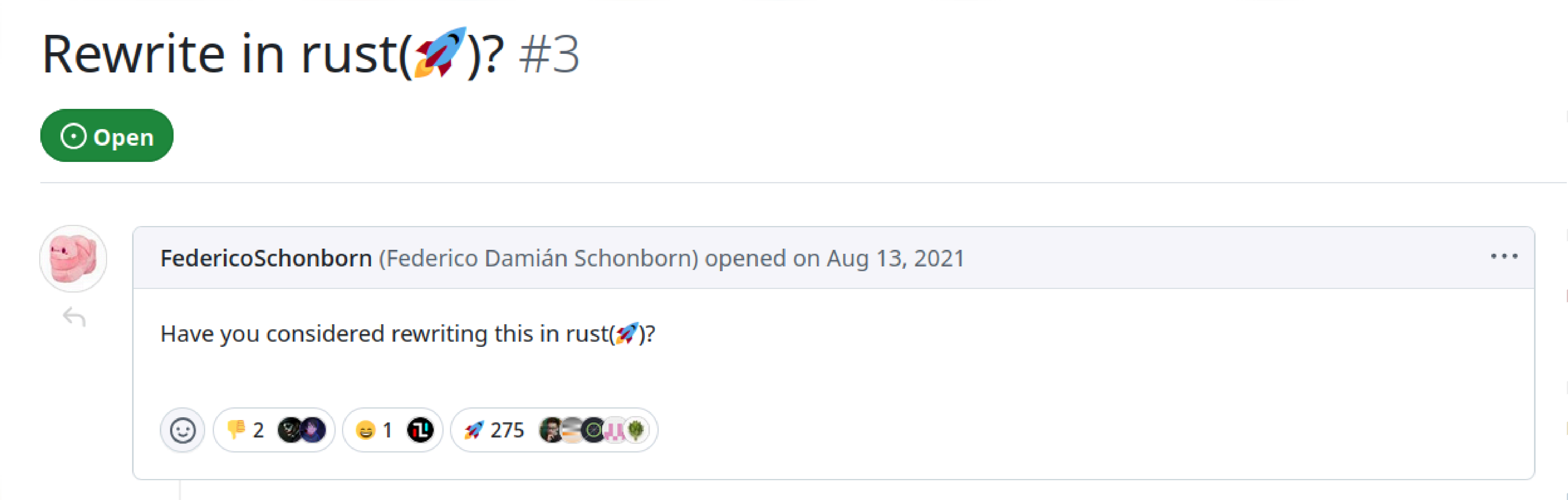
4.1 RIIR

4.1 RIIR

thing.c
Do it.
Just do it.
But why?
thing.rs
memory-safe 🦀 blazing fast 🦀
(only 10% slower than thing.c)
configurable 🦀 rewritten in Rust
🦀 only 372 dependencies 🦀

Figure 2: Meme

4.1 RIIR



Rewrite in rust(🚀)? #3

Open

FedericoSchonborn (Federico Damián Schonborn) opened on Aug 13, 2021

Have you considered rewriting this in rust(🚀)?

Figure 3: Meme

4.2 What is RIIR?

So, RIIR stands for “Rewrite it in Rust”, because Rust is

- 🔥 Blazing~~(ly)~~ Fast 🚀 🚀 🚀
- ✅ Memory Safe ✅
- 🦀 Minimal and Configurable 🦀

4.2 What is RIIR?

So, RIIR stands for “Rewrite it in Rust”, because Rust is

🔥 Blazing~~(ly)~~ Fast 🚀 🚀 🚀

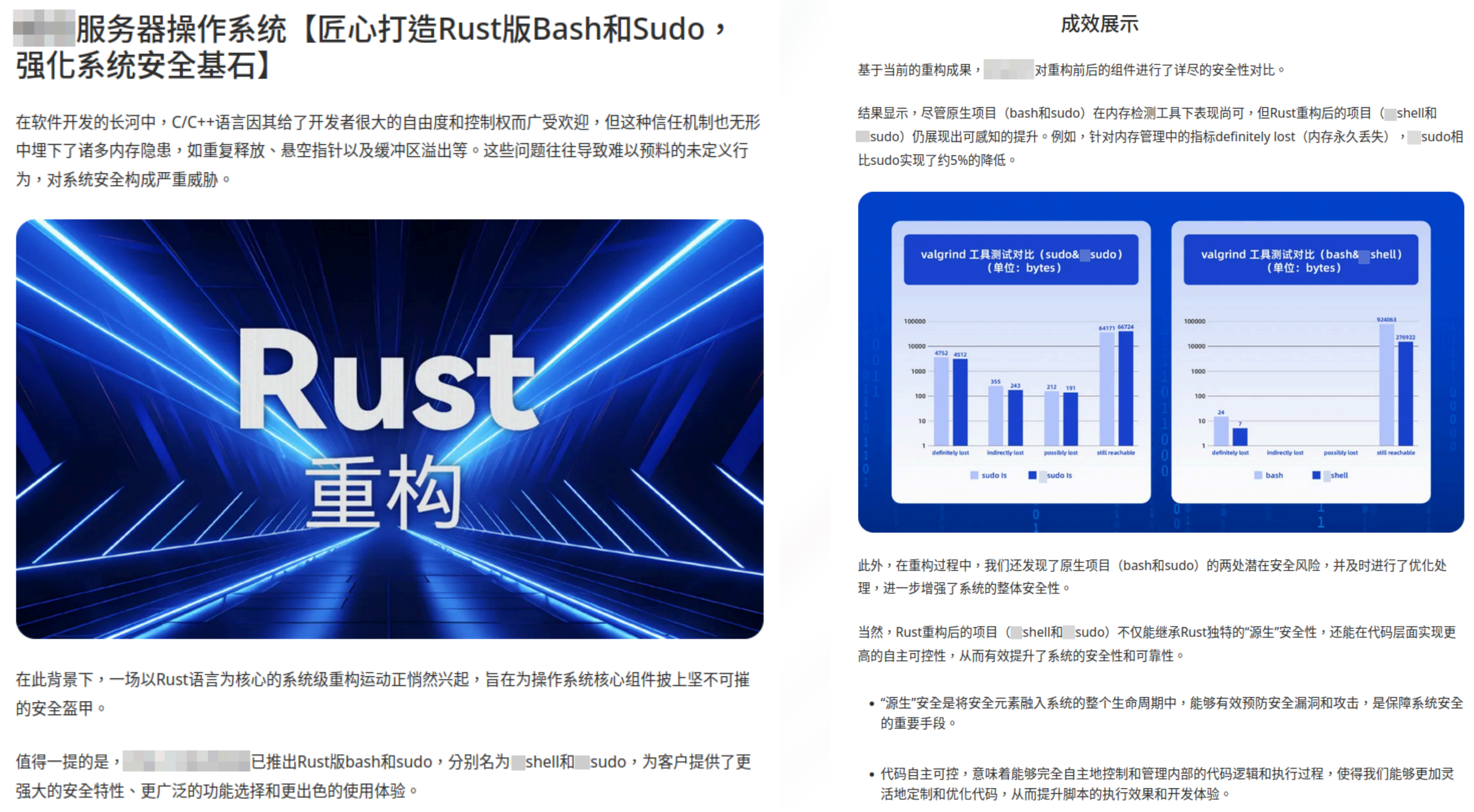
✅ Memory Safe ✅

🦀 Minimal and Configurable 🦀

… Is it? ~~Anyway, meme says so. 🦀~~

4.3 Let’s see some examples

4.3 Let’s see some examples



[redacted]服务器操作系统【匠心打造Rust版Bash和Sudo，强化系统安全基石】

在软件开发的长河中，C/C++语言因其给了开发者很大的自由度和控制权而广受欢迎，但这种信任机制也无形中埋下了诸多内存隐患，如重复释放、悬空指针以及缓冲区溢出等。这些问题往往导致难以预料的未定义行为，对系统安全构成严重威胁。

在此背景下，一场以Rust语言为核心的系统级重构运动正悄然兴起，旨在为操作系统核心组件披上坚不可摧的安全盔甲。

值得一提的是，[redacted]已推出Rust版bash和sudo，分别名为[redacted]shell和[redacted]sudo，为客户提供了更强大的安全特性、更广泛的功能选择和更出色的使用体验。

成效展示

基于当前的重构成果，[redacted]对重构前后的组件进行了详尽的安全性对比。

结果显示，尽管原生项目（bash和sudo）在内存检测工具下表现尚可，但Rust重构后的项目（[redacted]shell和[redacted]sudo）仍展现出可感知的提升。例如，针对内存管理中的指标definitely lost（内存永久丢失），[redacted]sudo相比sudo实现了约5%的降低。

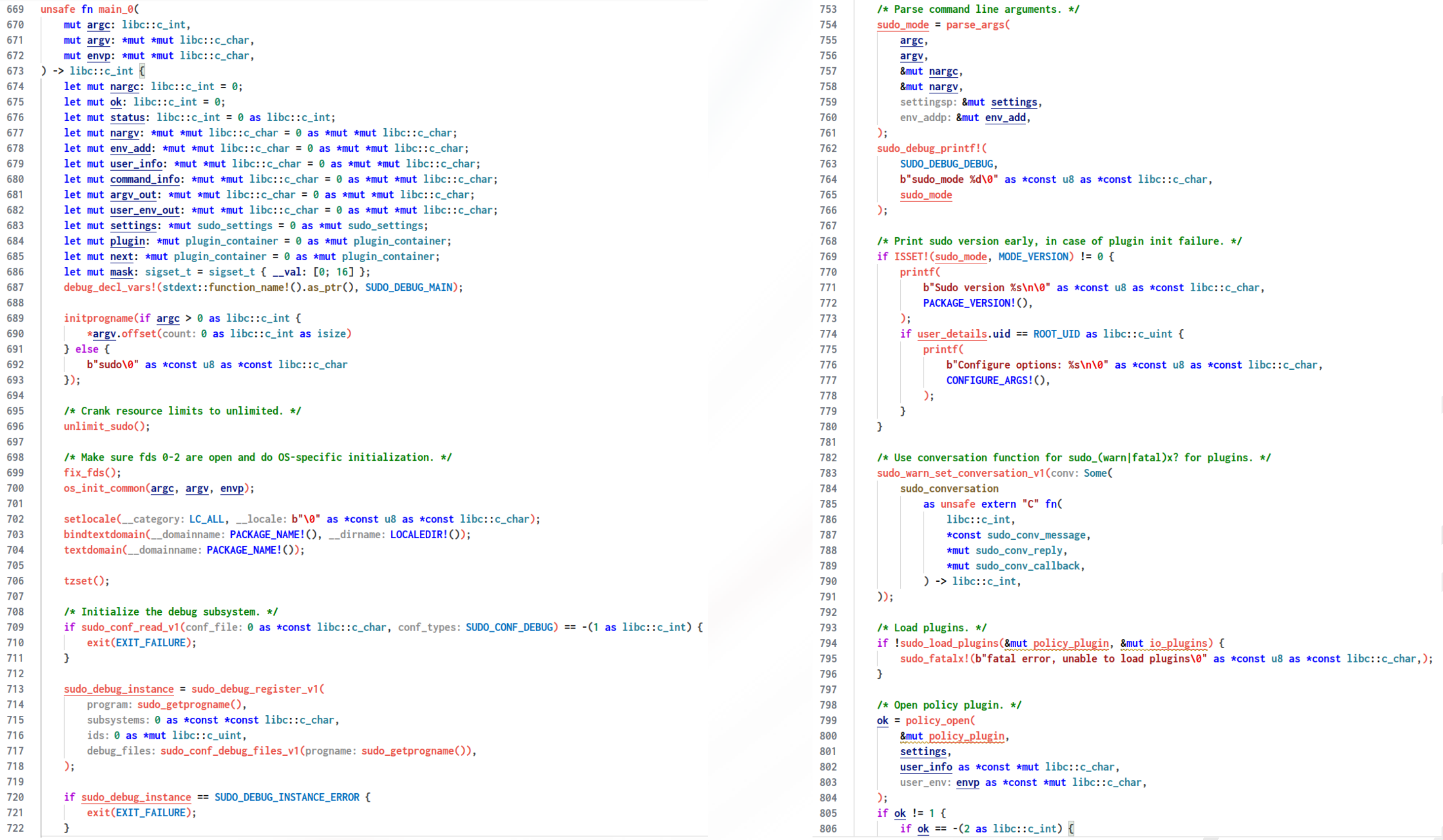
此外，在重构过程中，我们还发现了原生项目（bash和sudo）的两处潜在安全风险，并及时进行了优化处理，进一步增强了系统的整体安全性。

当然，Rust重构后的项目（[redacted]shell和[redacted]sudo）不仅能继承Rust独特的“源生”安全性，还能在代码层面实现更高的自主可控性，从而有效提升了系统的安全性和可靠性。

- “源生”安全是将安全元素融入系统的整个生命周期中，能够有效预防安全漏洞和攻击，是保障系统安全的重要手段。
- 代码自主可控，意味着能够完全自主地控制和管理内部的代码逻辑和执行过程，使得我们能够更加灵活地定制和优化代码，从而提升脚本的执行效果和开发体验。

Figure 4: Some company has refactored sudo and bash with Rust

4.3 Let’s see some examples

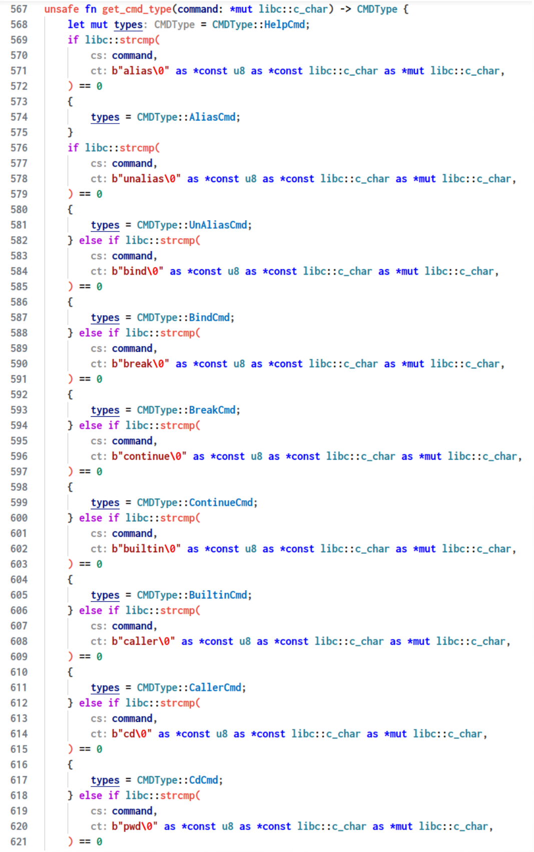


```rust
unsafe fn main_0(
    mut argc: libc::c_int,
    mut argv: *mut *mut libc::c_char,
    mut envp: *mut *mut libc::c_char,
) -> libc::c_int {
    let mut nargc: libc::c_int = 0;
    let mut ok: libc::c_int = 0;
    let mut status: libc::c_int = 0 as libc::c_int;
    let mut nargv: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut env_add: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut user_info: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut command_info: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut argv_out: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut user_env_out: *mut *mut libc::c_char = 0 as *mut *mut libc::c_char;
    let mut settings: *mut sudo_settings = 0 as *mut sudo_settings;
    let mut plugin: *mut plugin_container = 0 as *mut plugin_container;
    let mut next: *mut plugin_container = 0 as *mut plugin_container;
    let mut mask: sigset_t = sigset_t { __val: [0; 16] };
    debug_decl_vars!(stdext::function_name!().as_ptr(), SUDO_DEBUG_MAIN);

    initprogname(if argc > 0 as libc::c_int {
        *argv.offset(count: 0 as libc::c_int as isize)
    } else {
        b"sudo\0" as *const u8 as *const libc::c_char
    });

    /* Crank resource limits to unlimited. */
    unlimit_sudo();

    /* Make sure fds 0-2 are open and do OS-specific initialization. */
    fix_fds();
    os_init_common(argc, argv, envp);

    setlocale(__category: LC_ALL, __locale: b"\0" as *const u8 as *const libc::c_char);
    bindtextdomain(__domainname: PACKAGE_NAME!(), __dirname: LOCALEDIR!());
    textdomain(__domainname: PACKAGE_NAME!());

    tzset();

    /* Initialize the debug subsystem. */
    if sudo_conf_read_v1(conf_file: 0 as *const libc::c_char, conf_types: SUDO_CONF_DEBUG) == -(1 as libc::c_int) {
        exit(EXIT_FAILURE);
    }

    sudo_debug_instance = sudo_debug_register_v1(
        program: sudo_getprogname(),
        subsystems: 0 as *const *const libc::c_char,
        ids: 0 as *mut libc::c_uint,
        debug_files: sudo_conf_debug_files_v1(progname: sudo_getprogname()),
    );

    if sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR {
        exit(EXIT_FAILURE);
    }
```

```rust
    /* Parse command line arguments. */
    sudo_mode = parse_args(
        argc,
        argv,
        &mut nargc,
        &mut nargv,
        settingsp: &mut settings,
        env_addp: &mut env_add,
    );
    sudo_debug_printf!(
        SUDO_DEBUG_DEBUG,
        b"sudo_mode %d\0" as *const u8 as *const libc::c_char,
        sudo_mode
    );

    /* Print sudo version early, in case of plugin init failure. */
    if ISSET!(sudo_mode, MODE_VERSION) != 0 {
        printf(
            b"Sudo version %s\n\0" as *const u8 as *const libc::c_char,
            PACKAGE_VERSION!(),
        );
        if user_details.uid == ROOT_UID as libc::c_uint {
            printf(
                b"Configure options: %s\n\0" as *const u8 as *const libc::c_char,
                CONFIGURE_ARGS!(),
            );
        }
    }

    /* Use conversation function for sudo_(warn|fatal)x? for plugins. */
    sudo_warn_set_conversation_v1(conv: Some(
        sudo_conversation
            as unsafe extern "C" fn(
                libc::c_int,
                *const sudo_conv_message,
                *mut sudo_conv_reply,
                *mut sudo_conv_callback,
            ) -> libc::c_int,
    ));

    /* Load plugins. */
    if !sudo_load_plugins(&mut policy_plugin, &mut io_plugins) {
        sudo_fatalx!(b"fatal error, unable to load plugins\0" as *const u8 as *const libc::c_char,);
    }

    /* Open policy plugin. */
    ok = policy_open(
        &mut policy_plugin,
        settings,
        user_info as *const *mut libc::c_char,
        user_env: envp as *const *mut libc::c_char,
    );
    if ok != 1 {
        if ok == -(2 as libc::c_int) {
```

Figure 5: Their refactored Rust sudo code

4.3 Let’s see some examples



```
unsafe fn get_cmd_type(command: *mut libc::c_char) -> CMDType {
    let mut types: CMDType = CMDType::HelpCmd;
    if libc::strcmp(
        cs: command,
        ct: b"alias\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::AliasCmd;
    }
    if libc::strcmp(
        cs: command,
        ct: b"unalias\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::UnAliasCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"bind\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::BindCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"break\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::BreakCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"continue\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::ContinueCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"builtin\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::BuiltinCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"caller\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::CallerCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"cd\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
    {
        types = CMDType::CdCmd;
    } else if libc::strcmp(
        cs: command,
        ct: b"pwd\0" as *const u8 as *const libc::c_char as *mut libc::c_char,
    ) == 0
```

Figure 6: Their refactored Rust `bash` code

4.3 Let’s see some examples

4.3 Let's see some examples

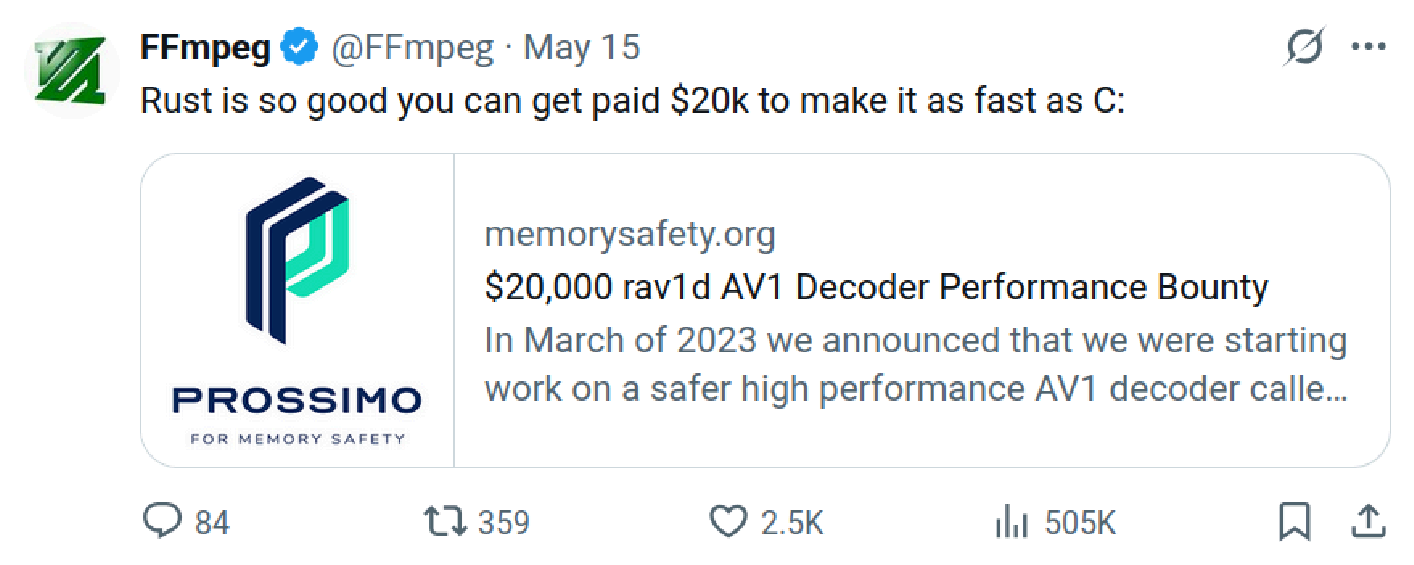
`dav1d`

is a open-source AV1 cross-platform decoder, written in C.

`rav1d`

is a Rust port of `dav1d`.

4.3 Let's see some examples



FFmpeg @FFmpeg · May 15

Rust is so good you can get paid $20k to make it as fast as C:

memorysafety.org

$20,000 rav1d AV1 Decoder Performance Bounty

In March of 2023 we announced that we were starting work on a safer high performance AV1 decoder calle...

Figure 8: FFmpeg comments on `rav1d`'s bounty on Twitter

4.3 Let's see some examples

4.3 Let’s see some examples

`grep`

is a command for searching strings for lines that match a pattern.

`ripgrep`

is a tool for the same purpose, written in Rust.

4.3 Let’s see some examples

`grep`

is a command for searching strings for lines that match a pattern.

`ripgrep`

is a tool for the same purpose, written in Rust.

ripgrep is faster than {grep, ag, git grep, ucg, pt, sift}

4.3 Let's see some examples

`grep`

is a command for searching strings for lines that match a pattern.

`ripgrep`

is a tool for the same purpose, written in Rust.

"ripgrep is faster than {grep, ag, git grep, ucg, pt, sift}"

is a blog by Andrew Garland explaining the optimizations made in `ripgrep`.

4.4 What Rust does

Similar to C/C++

Rust compiles to native binary files, by using LLVM as the backend.

4.4 What Rust does

Similar to C/C++

Rust compiles to native binary files, by using LLVM as the backend.

There is no free lunch

as well as free performance boosts and safety increases.

4.4 What Rust does

Similar to C/C++

Rust compiles to native binary files, by using LLVM as the backend.

There is no free lunch

as well as free performance boosts and safety increases.

Fast and safe

not because the program is written in Rust.

4.4 What Rust does

- **Less historical baggage**
  More modern and ergonomic language syntax and standard library.

- **Less pain for debugging**
  Same cognitive load, but narrowing the pain of debugging to the learning and development stages.

- **More opportunities**
  Rust is not perfect, but it is developing rapidly and actively embracing change.

4.5 More opportunities

```
[info] [main.cpp] hello, world!
```

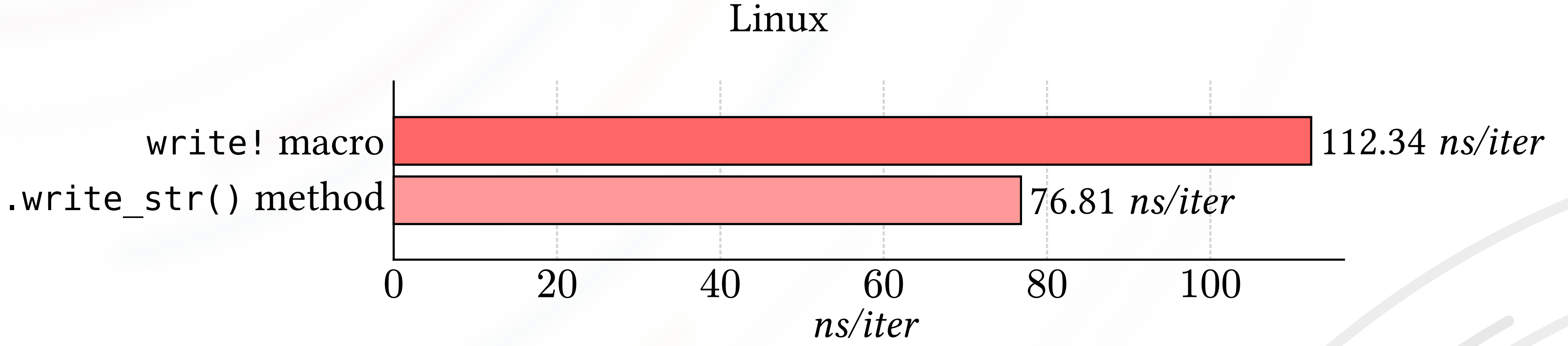
Write buffer via `write!` macro

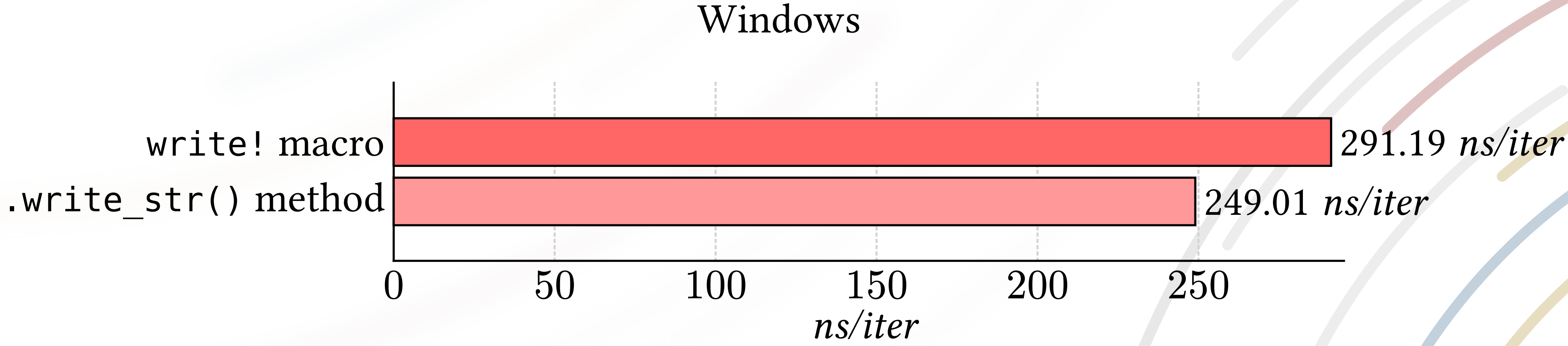
```rust
let mut dest = String::new();
write!(dest, "[{}] [{}] {}", level, source, payload)?;
```

Write buffer via `.write_str()` method

```rust
let mut dest = String::new();
dest.write_str(&level)?;
dest.write_str("] [")?;
dest.write_str(&source)?;
dest.write_str("] ")?;
dest.write_str(&payload)?;
```

4.5 More opportunities





4.5 More opportunities

This is a known issue[1], and someone is working on it.

[1] rust-lang/rust#99012: Tracking issue for improving std::fmt::Arguments and format_args!()

4.5 More opportunities

spdlog-rs feature: Named optional parameter logger

```rust
let my_logger = /* ... */;
info!(logger: my_logger, "hello, world!");
```

This feature has been accepted and merged[1] in the upstream log crate.

[1] rust-lang/log#664: Add an optional logger param

4.5 More opportunities

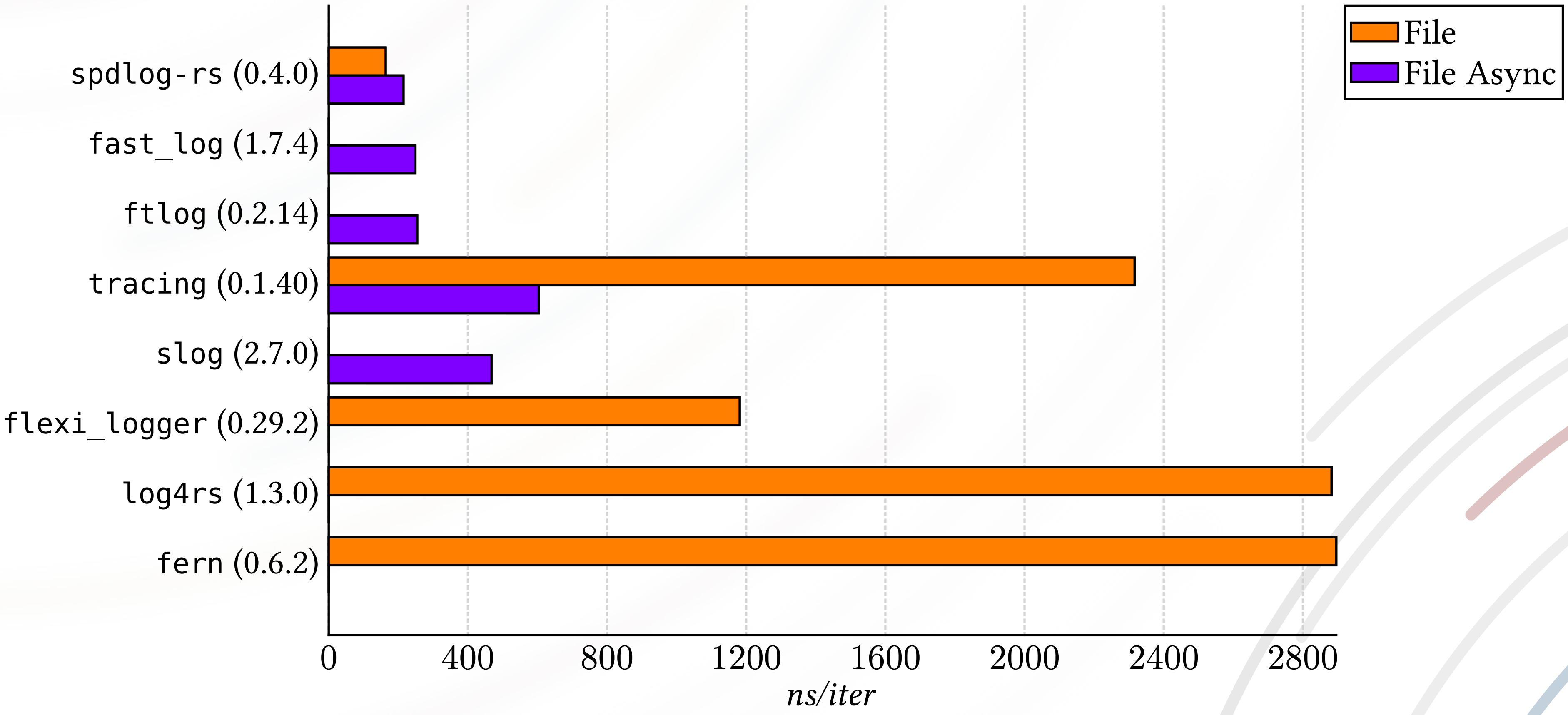
`spdlog-rs` feature: Compile-time pattern formatter

```
let p = pattern!("{datetime} {level} {payload}{eol}");
// The type of `p` is compiled from the literal string.
```

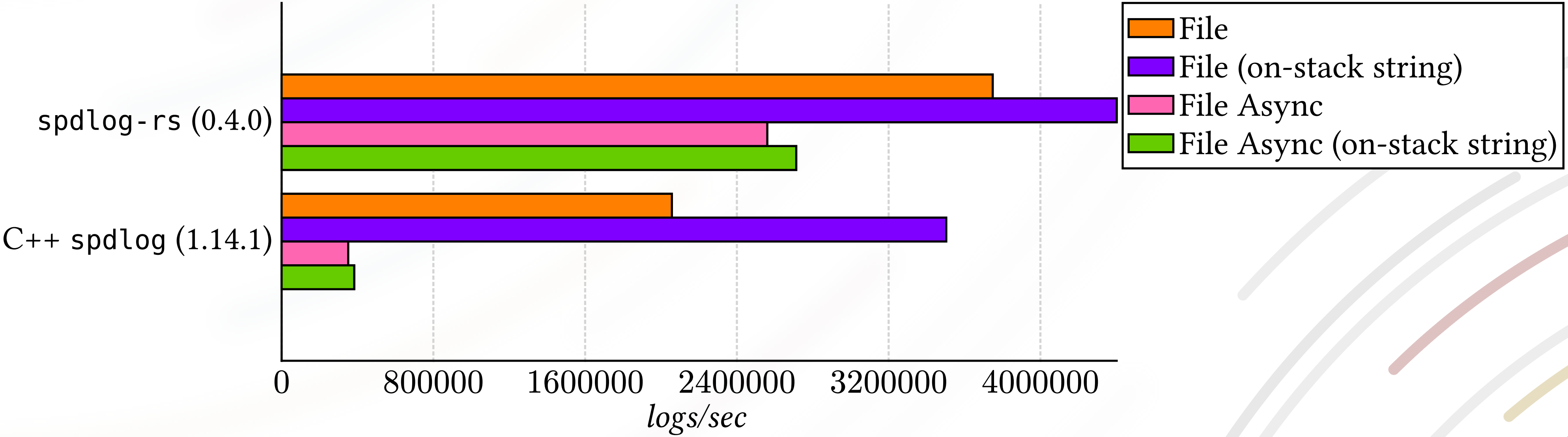
We are proposing this feature to C++ `spdlog` and have opened a PR[1] for the initial implementation.

[1] gabime/spdlog#3404: Support compiling pattern literal strings at compile-time

4.6 Benchmarks



4.6 Benchmarks



4.7 Acknowledgements contributors

Sirui Mu
@Lancern

猫猫
@NotEvenANeko

Nick Clavette
@Nicholas-Clavette

Dylan DPC
@Dylan-DPC

Bobby Powers
@bpowers

Faizaan Pervaiz
@fpervaiz

lioriz
@lioriz

Any question?

We ❤️ Interns

PLCT Lab

Compilers, Runtimes, and Emulators.

We ❤️ Interns

PLCT Lab

Compilers, Runtimes, and Emulators.

Thanks.