Co vám může nabídnout Rust?

Kuba Beránek

github.com/kobzol

Kuba Beránek

github.com/kobzol

Kuba Beránek

- PhD student, teacher @ VSB-TUO

github.com/kobzol

Kuba Beránek

- PhD student, teacher @ VSB-TUO
- Researcher @ IT4Innovations

github.com/kobzol

Kuba Beránek

- PhD student, teacher @ VSB-TUO
- Researcher @ IT4Innovations
- HPC, distributed systems, code optimization, machine learning,...

github.com/kobzol

Kuba Beránek

- PhD student, teacher @ VSB-TUO
- Researcher @ IT4Innovations
- HPC, distributed systems, code optimization, machine learning,...
- Rust project contributor

Compiler performance working group

Improving rustc compilation performance (build times)

Members

Mark Rousskov

GitHub: Mark-Simulacrum

Team leader

Jakub Beránek

GitHub: Kobzol

Rémy Rakic

GitHub: lqd

Michael Woerister

GitHub: michaelwoerister

Nicholas Nethercote

GitHub: nnethercote

Felix Klock

GitHub: pnkfelix

Infrastructure team

Managing the infrastructure supporting the Rust project itself, including CI, releases, bots, and metrics

Members

Jan David Nose

GitHub: jdno

Team leader

Jake Goulding

GitHub: shepmaster

Team leader

kennytm

GitHub: kennytm

Jakub Beránek

GitHub: Kobzol

Mark Rousskov

GitHub: Mark-Simulacrum

Pietro Albini

GitHub: pietroalbini

Rust in a nutshell

Rust in a nutshell

- Static + strong typing

Rust in a nutshell

- Static + strong typing
- OOP + FP paradigms

Rust in a nutshell

- Static + strong typing
- OOP + FP paradigms
- Syntax based on C

Rust in a nutshell

- Static + strong typing
- OOP + FP paradigms
- Syntax based on C
- Compiled to native code

Rust in a nutshell

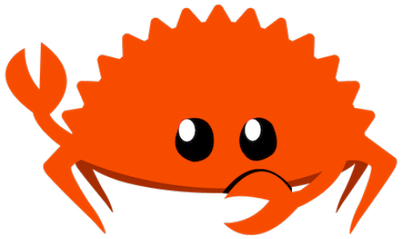
- Static + strong typing
- OOP + FP paradigms
- Syntax based on C
- Compiled to native code
- No GC, minimal runtime

Rust in a nutshell

- Static + strong typing
- OOP + FP paradigms
- Syntax based on C
- Compiled to native code
- No GC, minimal runtime
- Integrated package manager

Rust in a nutshell

- Static + strong typing
- OOP + FP paradigms
- Syntax based on C
- Compiled to native code
- No GC, minimal runtime
- Integrated package manager
- Friendly community



```rust
fn main() {
    println!("Hello TechMeetup!");
}
```

Rust history and testimonials

2006

Started as a personal project
by Graydon Hoare (@Mozilla)

2006

Started as a personal project
by Graydon Hoare (@Mozilla)

> “
> I think I named it after fungi…
> that is "over-engineered for survival".
> Graydon Hoare ”

Project Servo

Technology from the past
come to save the future
from itself

Mozilla Annual Summit, July 2010
<graydon@mozilla.com>

Project Servo

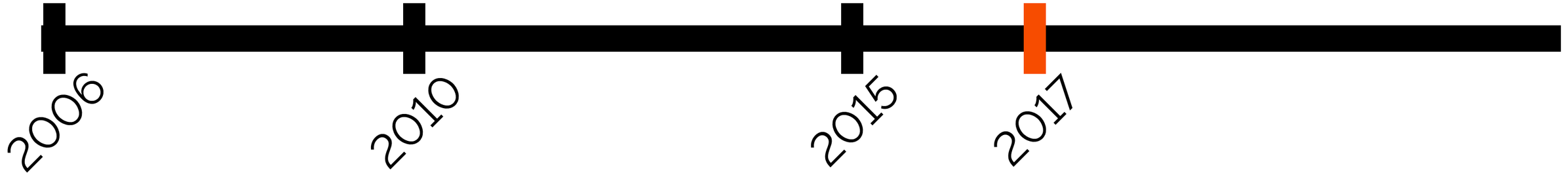
Technology from the past
come to save the future
from itself

Mozilla Annual Summit, July 2010
<graydon@mozilla.com>

Rust 1.0 released 🎉

Rust 1.0 released 🎉

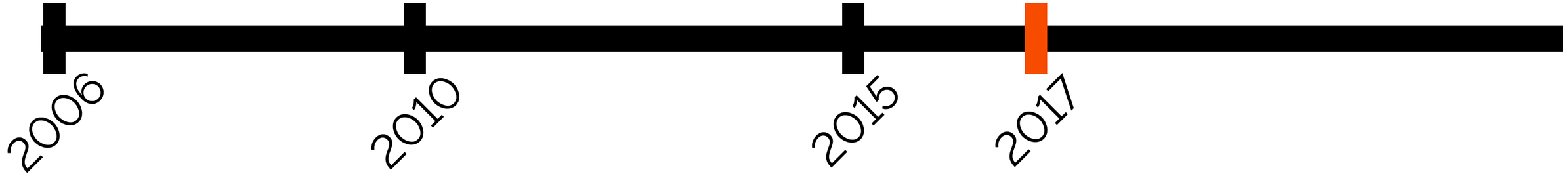
Strong backwards-compatibility promise



Inside a super fast CSS engine: Quantum CSS (aka Stylo)

By Lin Clark

Posted on August 22, 2017 in Code Cartoons, CSS, and Featured Article

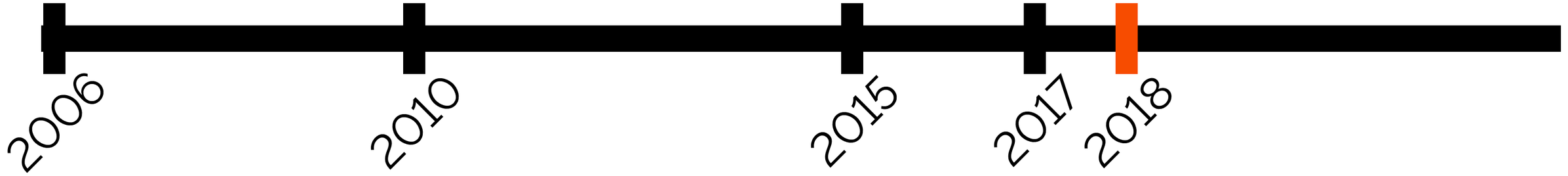


Posted by u/[deleted] 7 years ago

283

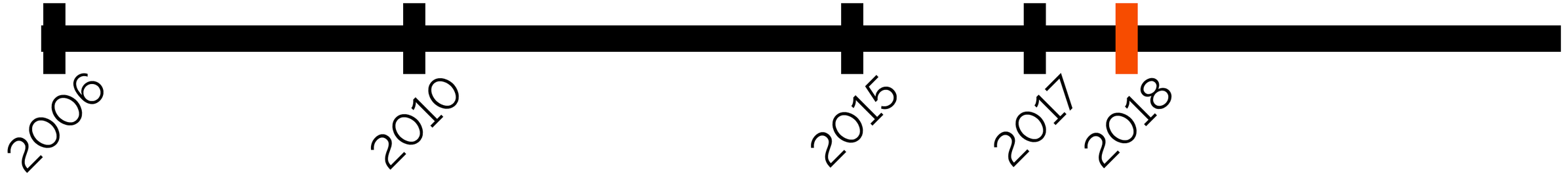
Visual Studio Code's new ripgrep-powered search has been released!

code.visualstudio.com/update...



Announcing Rust 1.31 and Rust 2018

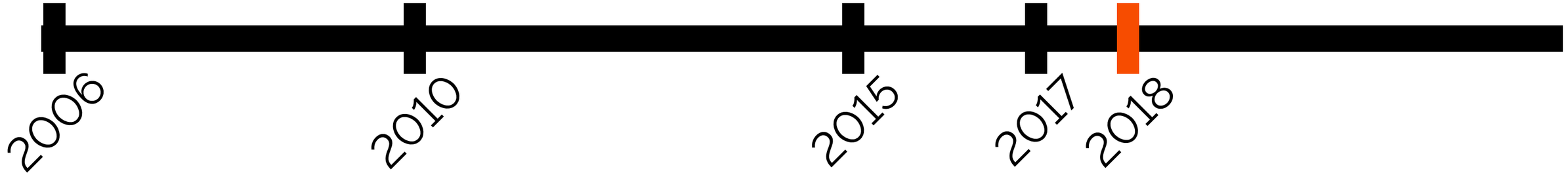
Dec. 6, 2018 · The Rust Core Team



Introducing Firecracker, a New Virtualization Technology and Open Source Project for Running Multi-Tenant Container Workloads

Posted On: Nov 26, 2018

Today, Amazon Web Services (AWS) is announcing Firecracker, new virtualization and open source technology that enables service owners to operate secure multi-tenant container-based services by combining the speed, resource efficiency, and performance enabled

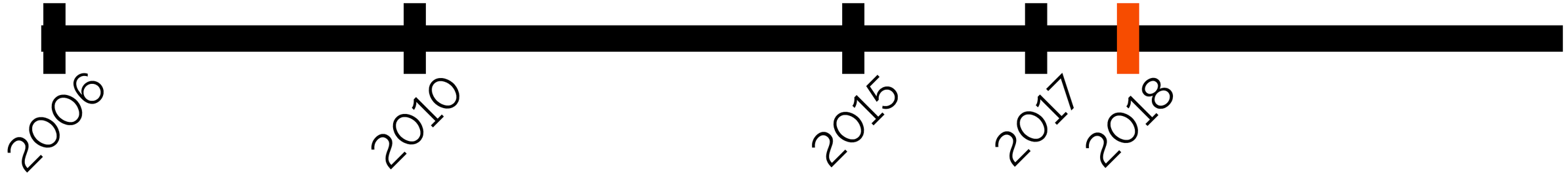


Serverless Rust with Cloudflare Workers

10/16/2018

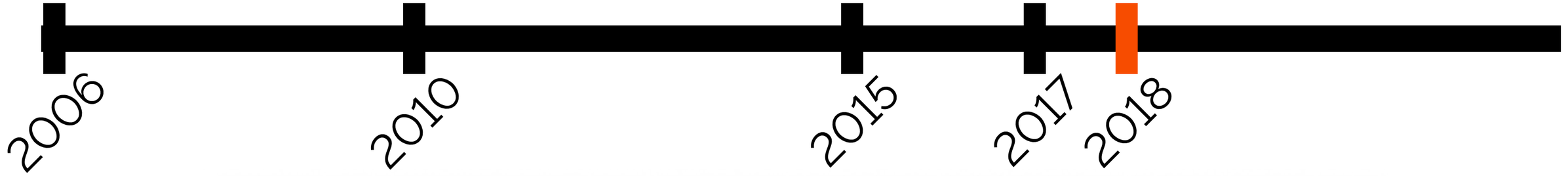
CLOUDFLARE

Steven Pack

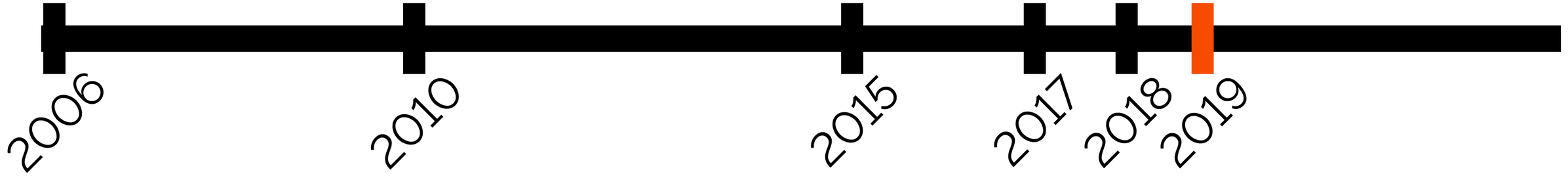


How Mozilla’s new language dramatically improved our server-side performance

Like building state-of-the-art web apps? Come work at Figma!



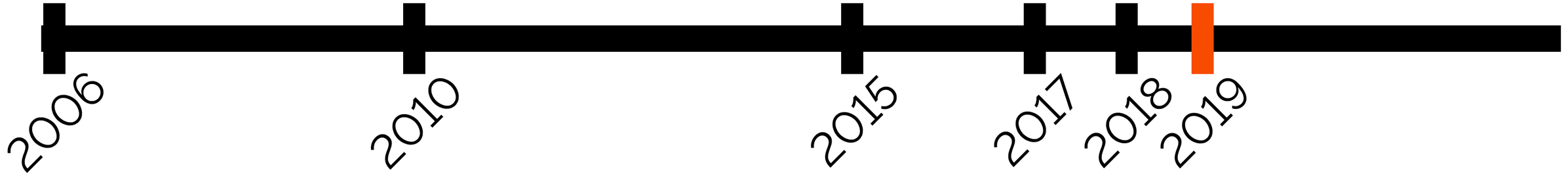
Metric	Old server		New server	Improvement
Peak average per-worker memory usage	4.2gb	→	1.1gb	3.8x smaller
Peak average per-machine CPU usage	24%	→	4%	6x smaller
Peak average file serve time	2s	→	0.2s	10x faster
Peak worst-case save time	82s	→	5s	16.4x faster



Rust Case Study:

Community makes Rust an easy choice for npm

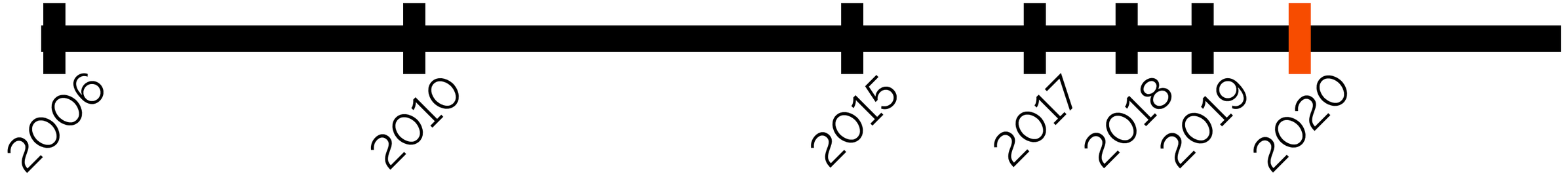
The npm Registry uses Rust for its CPU-bound bottlenecks

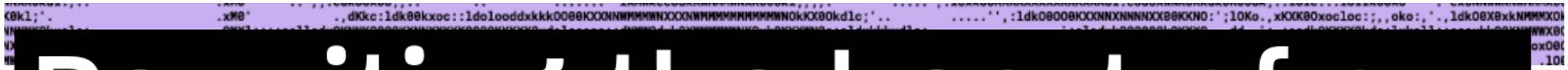


Rust Case Study:

> “
> The good news for the npm team is that the **Rust** service has been **running** for more than one year **in production without a single alert**.
> This is in stark contrast to the usual experience of deploying a Node.js service…
>
> npm ”

The npm Registry uses Rust for its CPU-bound bottlenecks

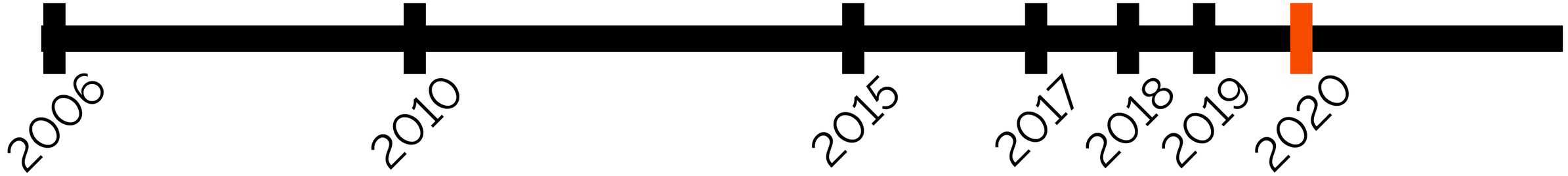




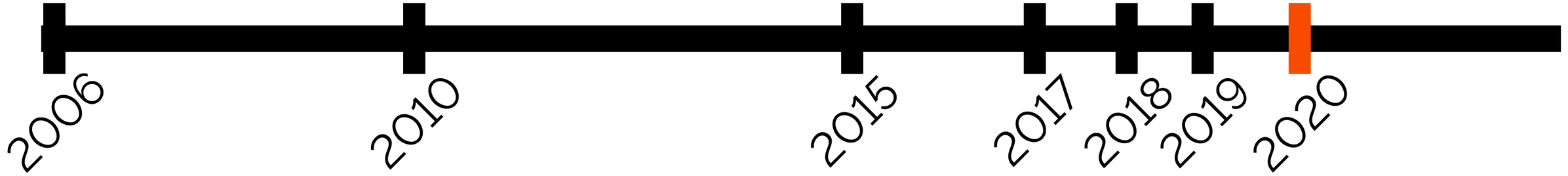
Rewriting the heart of our sync engine

// By Sujay Jayakar • Mar 09, 2020

Dropbox



> “
> …**betting on Rust was one of the best decisions we made**.
> More than performance, its ergonomics and focus on correctness has helped us tame sync’s complexity.
>
> Dropbox ”

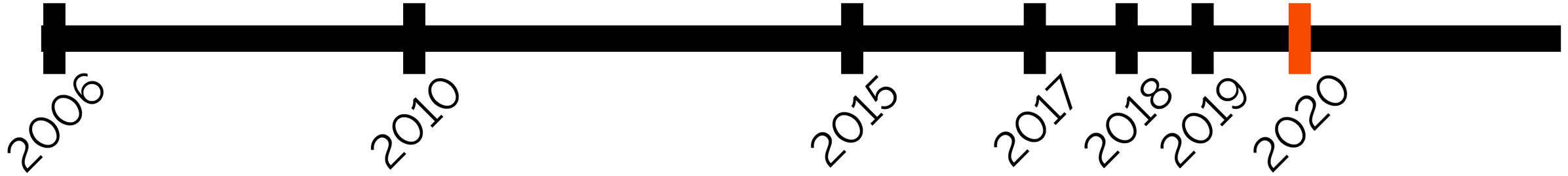


ENGINEERING & DEVELOPERS

WHY DISCORD IS SWITCHING FROM GO TO RUST

Jesse Howarth
February 4, 2020

Rust is becoming a first class language in a variety of domains. At Discord, we've seen success with Rust on the client side and server side. For example, we use it on the client side for our video encoding pipeline for Go Live and on the server side for Elixir NIFs.

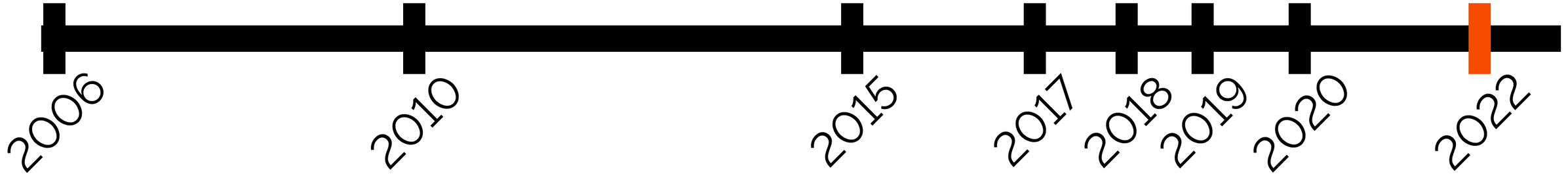


ENGINEERING & DEVELOPERS

> “
> Even with just basic optimization, Rust was able to outperform the hyper hand-tuned Go version.
> …**we were able to beat Go on every single performance metric**.
> Discord ”

February 4, 2020

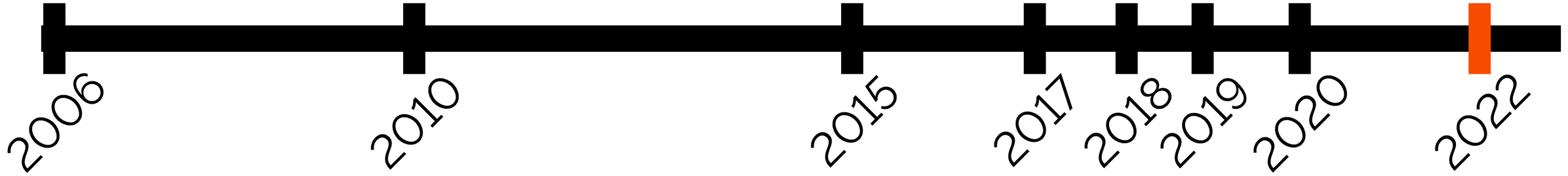
encoding pipeline for Go Live and on the server side for Elixir NIFs.



Amazon CloudFront now supports HTTP/3 powered by QUIC

Posted On: Aug 15, 2022

Amazon CloudFront now supports HTTP version 3 (HTTP/3) requests over QUIC for end user connections. HTTP/3 uses QUIC, a user datagram protocol (UDP) based, stream-multiplexed, secure transport protocol that combines and improves upon the capabilities of existing transmission control protocol (TCP), TLS, and HTTP/2. HTTP/3 offers several benefits over previous HTTP versions, including faster response times and enhanced security.



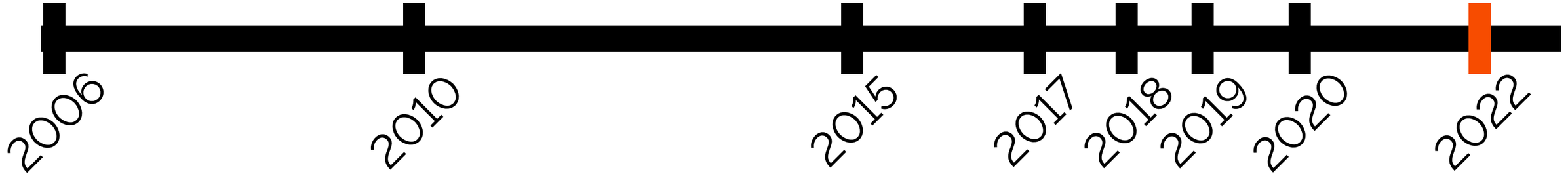
Am
pov

Posted

Amazon
HTTP/3
that co
HTTP/2
enhanced security.

nections.
protocol
TLS, and
nes and

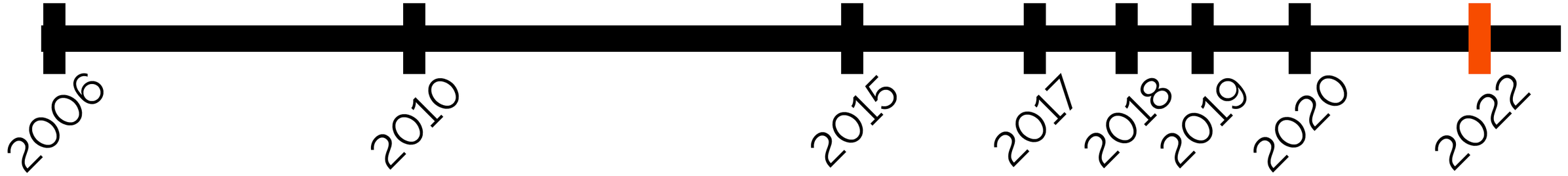
> It is written in Rust, so it reaps some of its benefits such as performance, thread and memory-safety.
>
> AWS



Meta Adds Rust to Its Portfolio of Internally-Supported Languages

Wednesday, social media giant Meta announced that it is now officially using Rust as a server side programming language.

Jul 28th, 2022 12:05pm by Jessica Wachtel



Met
Inte

Wednesday
programmi

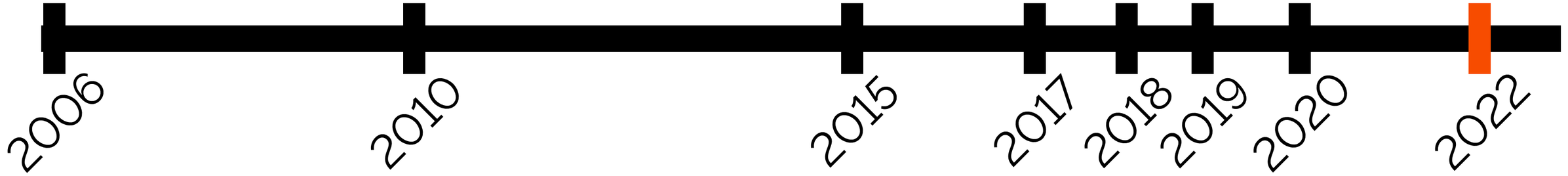
Jul 28th, 2022

of

s

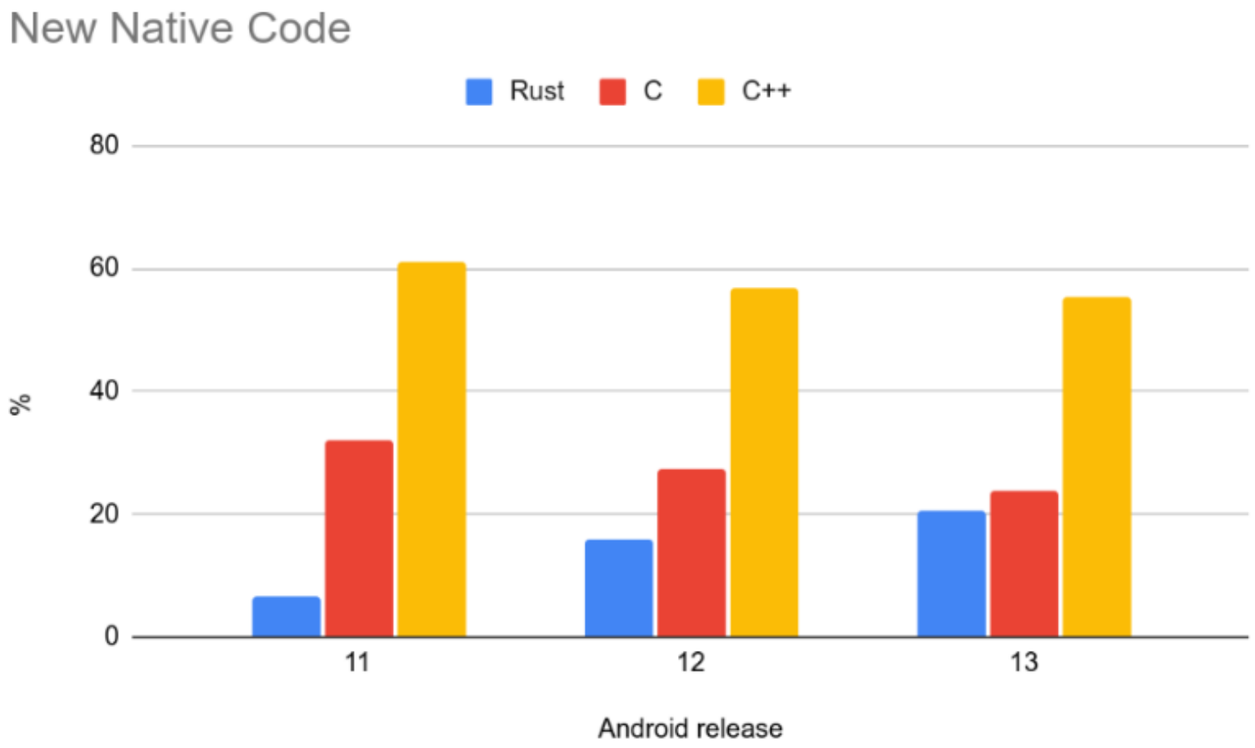
server side

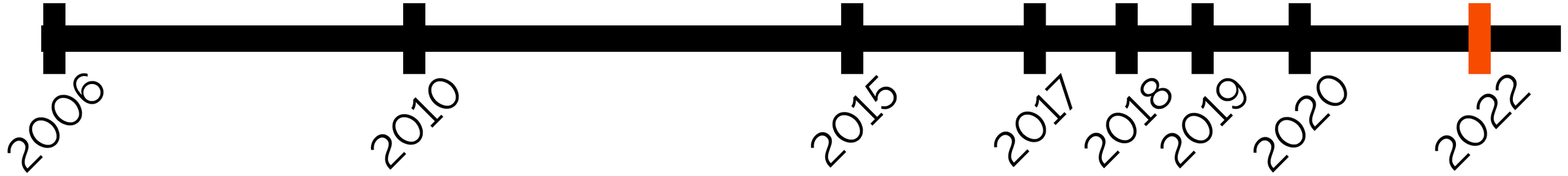
> “
> …we’re committing to Rust long-term and welcome early adopters.
>
> Meta ”



Memory Safe Languages in Android 13

December 1, 2022

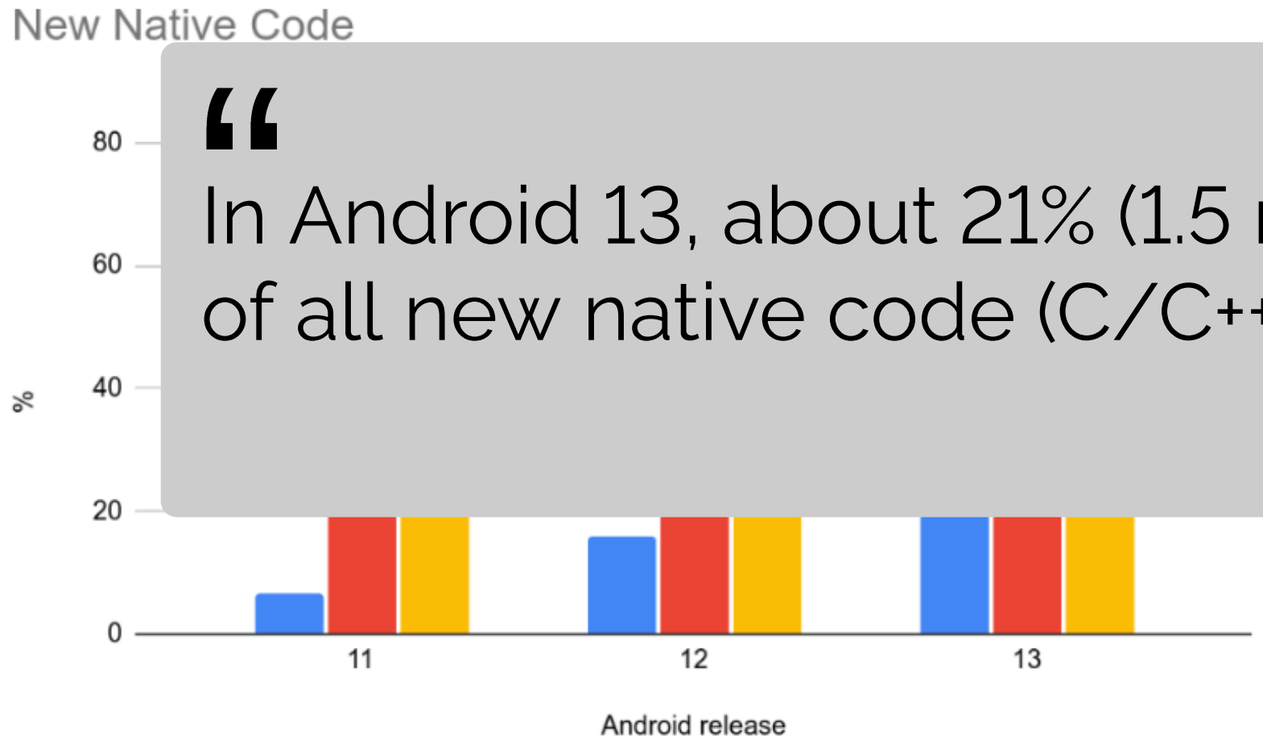


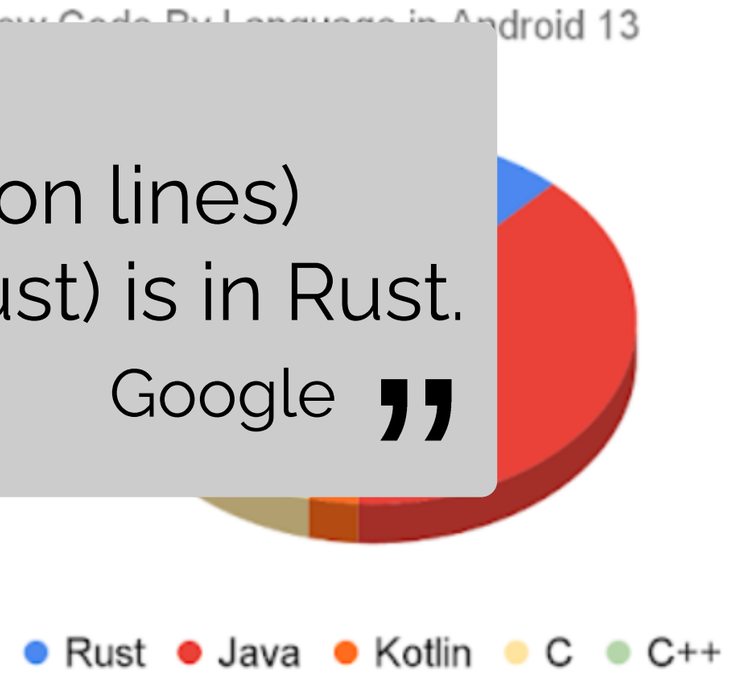


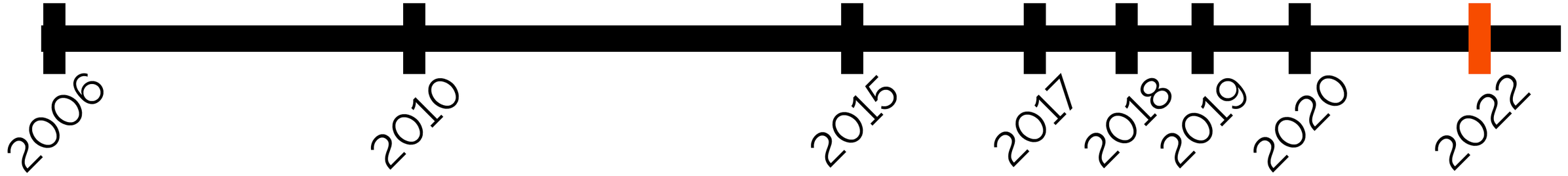
Memory Safe Languages in Android 13

December 1, 2022

> “In Android 13, about 21% (1.5 million lines) of all new native code (C/C++/Rust) is in Rust.”
>
> Google



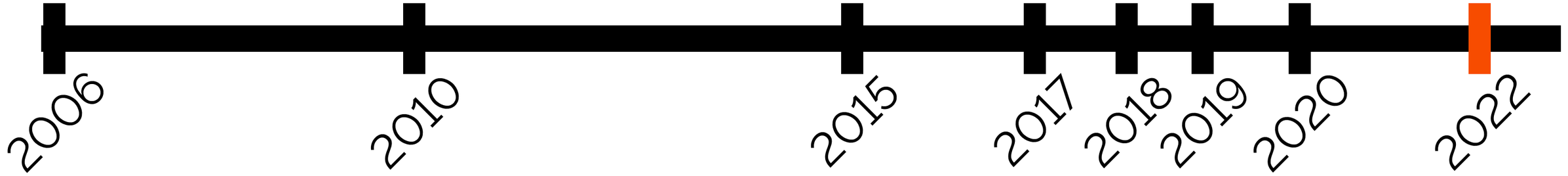




Rust Programming Language To Land in Linux Kernel 6.1

By Ian Evenden published October 06, 2022

Linux will support the Rust programming language in its kernel from version 6.1.

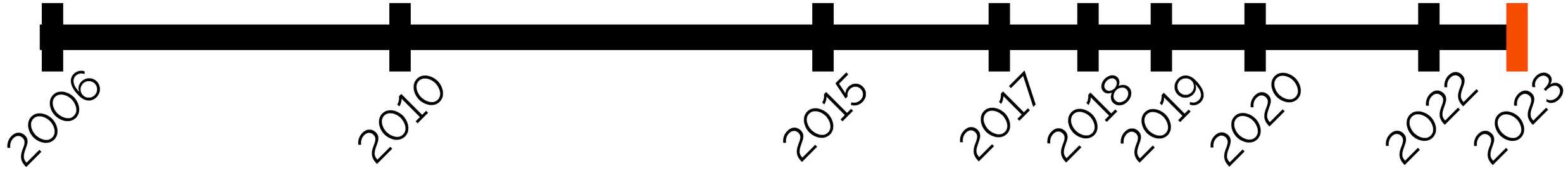


Rust Programming Language To Land in Linux Ker

> “
> …on the whole, I don't hate it.
> Linus Torvalds ”

By Ian Evenden publis

Linux will supp ge in its kernel from version 6.1.

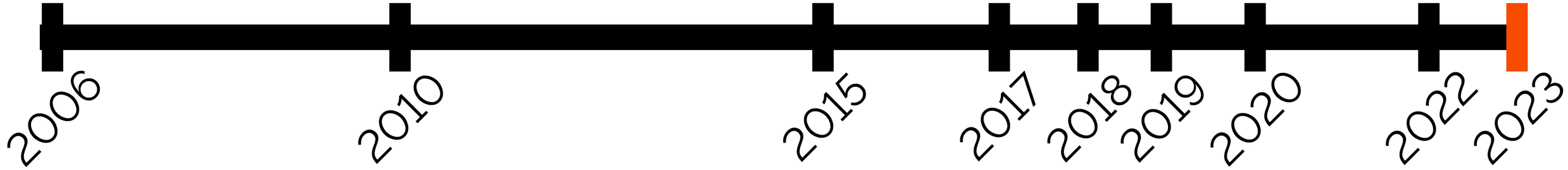


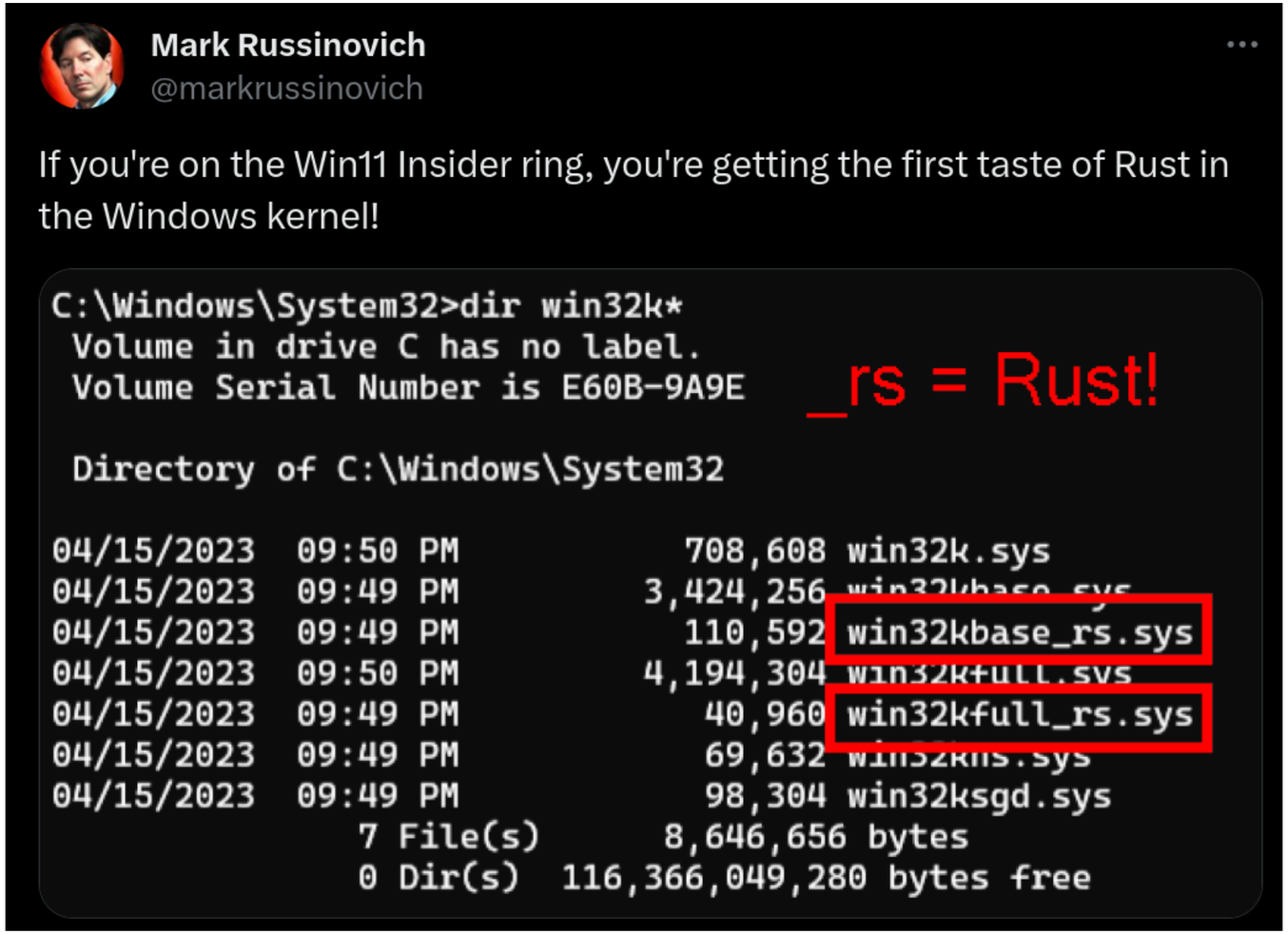
GitHub built a new search engine for code 'from scratch' in Rust

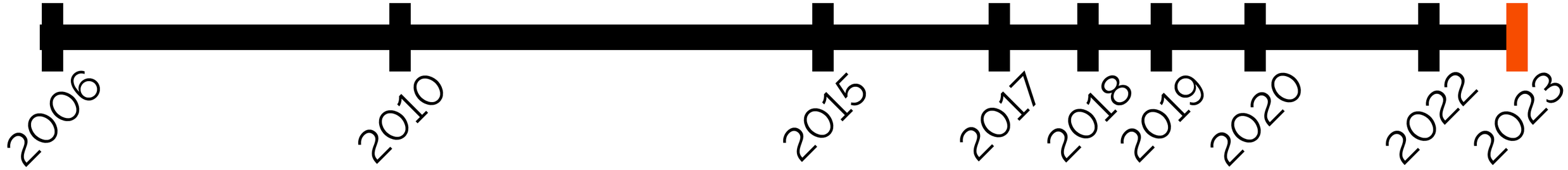
GitHub built a new code-focused search engine in Rust because popular text search engines couldn't scale enough.

Written by **Liam Tung,** Contributing Writer

Feb. 9, 2023 at 3:24 a.m. PT


2006
2010
2015
2017
2018
2019
2020
2022
2023

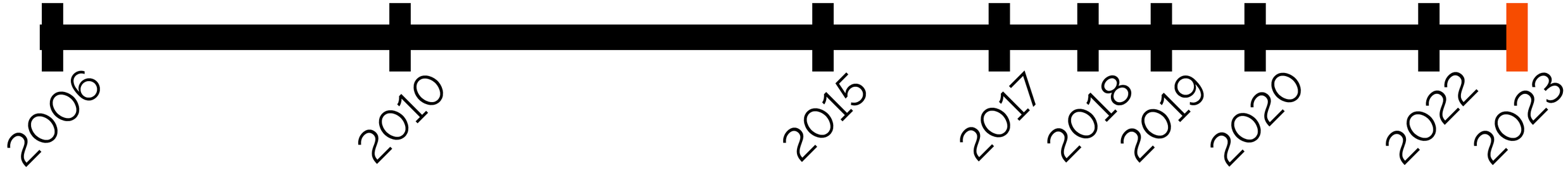

Mark Russinovich
@markrussinovich
If you're on the Win11 Insider ring, you're getting the first taste of Rust in the Windows kernel!
C:\Windows\System32>dir win32k*
Volume in drive C has no label.
Volume Serial Number is E60B-9A9E
_rs = Rust!
Directory of C:\Windows\System32
04/15/2023 09:50 PM 708,608 win32k.sys
04/15/2023 09:49 PM 3,424,256
04/15/2023 09:49 PM 110,592 win32kbase_rs.sys
04/15/2023 09:50 PM 4,194,304
04/15/2023 09:49 PM 40,960 win32kfull_rs.sys
04/15/2023 09:49 PM 69,632
04/15/2023 09:49 PM 98,304 win32ksgd.sys
7 File(s) 8,646,656 bytes
0 Dir(s) 116,366,049,280 bytes free



Mark Russinovich
@markrussinovich

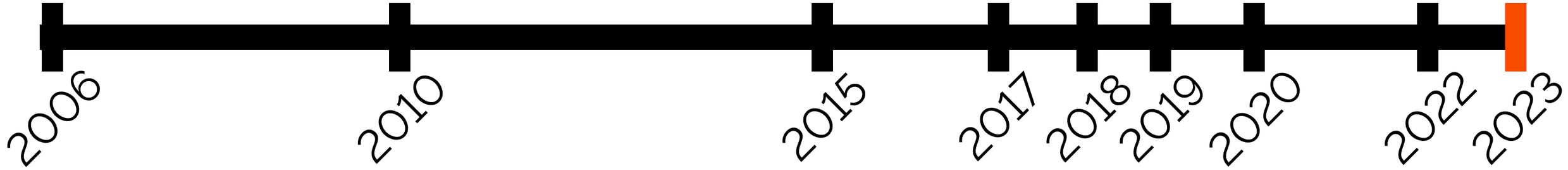
> “Speaking of languages, it's time to halt starting any new projects in C/C++ and **use Rust for those scenarios where a non-GC language is required**. For the sake of security and reliability, the industry should declare those languages as deprecated.
>
> Mark Russinovich ”

```
7 File(s)      8,646,656 bytes
0 Dir(s)  116,366,049,280 bytes free
```



Microsoft posts ‘early stages’ code for developing Windows drivers in Rust

By **Tim Anderson** - September 25, 2023

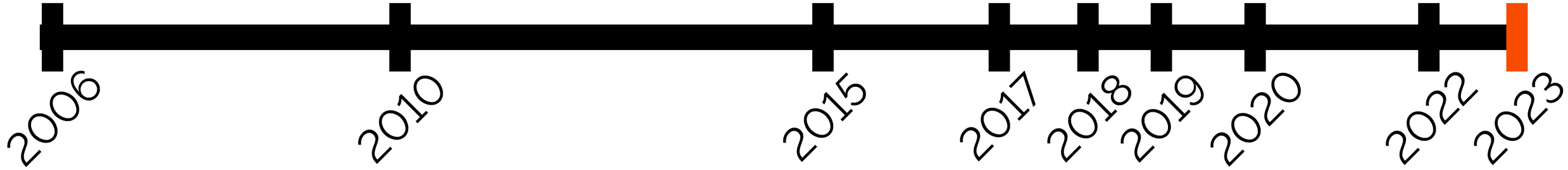


Introducing the Azure Quantum Development Kit Preview

Bill Ticehurst

September 19th, 2023 | 6 | 7

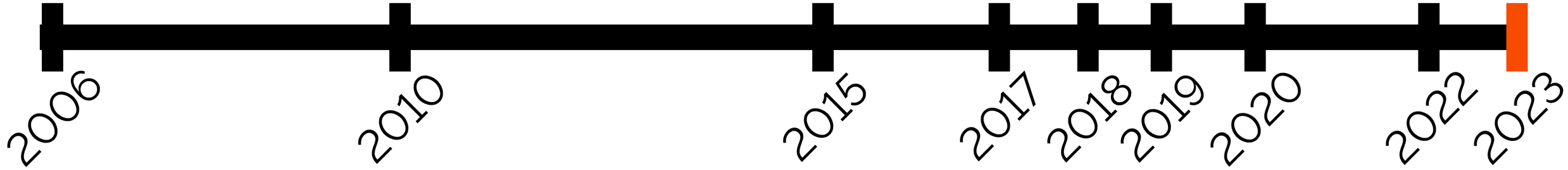
100x faster, 100x smaller, and it runs in the browser!



Ferrocene Safety-Critical Rust Compiler Code Published

Written by Michael Larabel in Programming on 5 October 2023 at 05:15 PM EDT.
35 Comments

Ferrous Systems has made available open-source code for Ferrocene, their Rust compiler focused on safety-critical and mission-critical environments. The Ferrocene compiler is being made available under Apache 2.0 or MIT licensing.



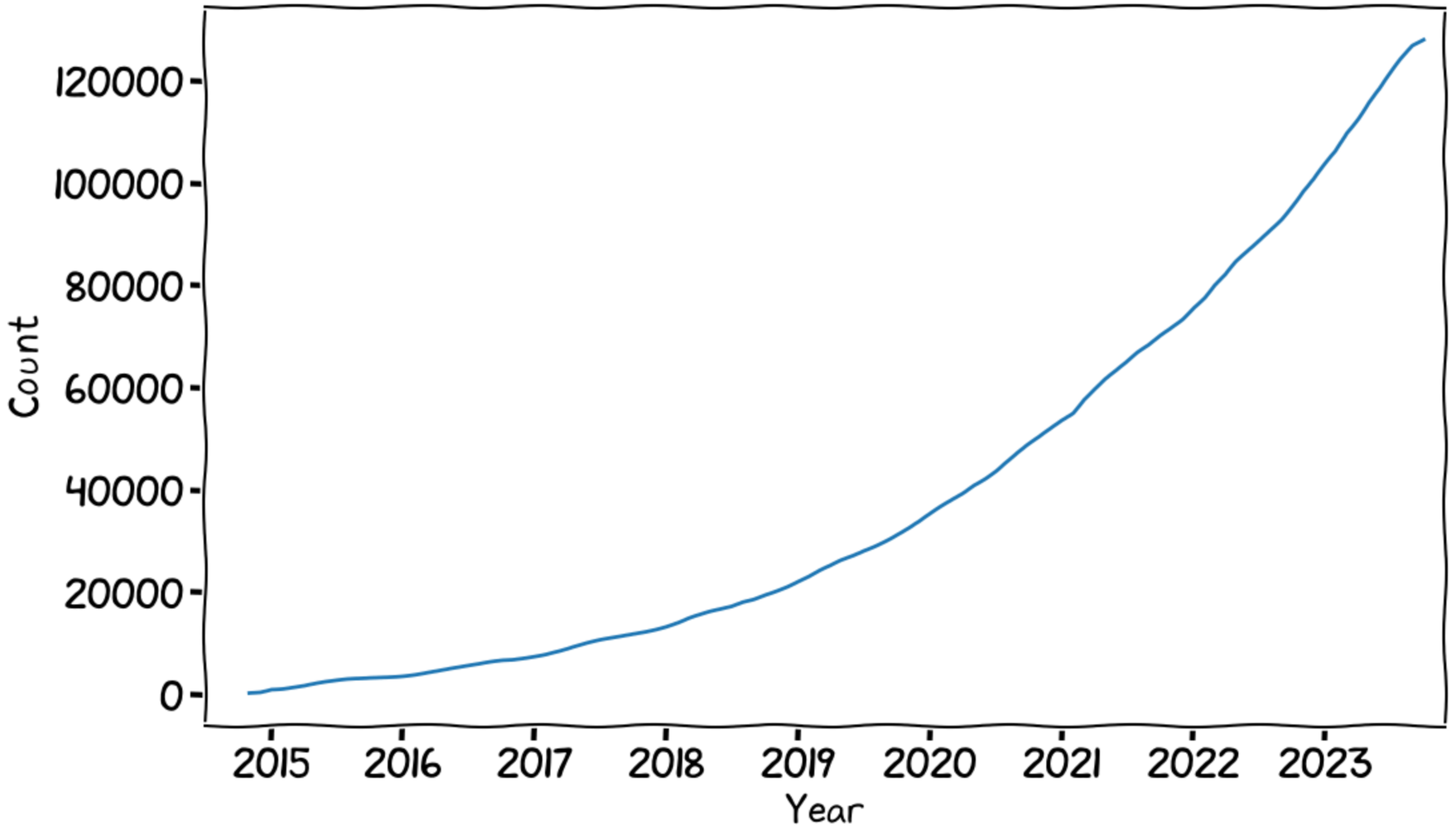
Posted by u/Chadshinshin32 4 months ago

833

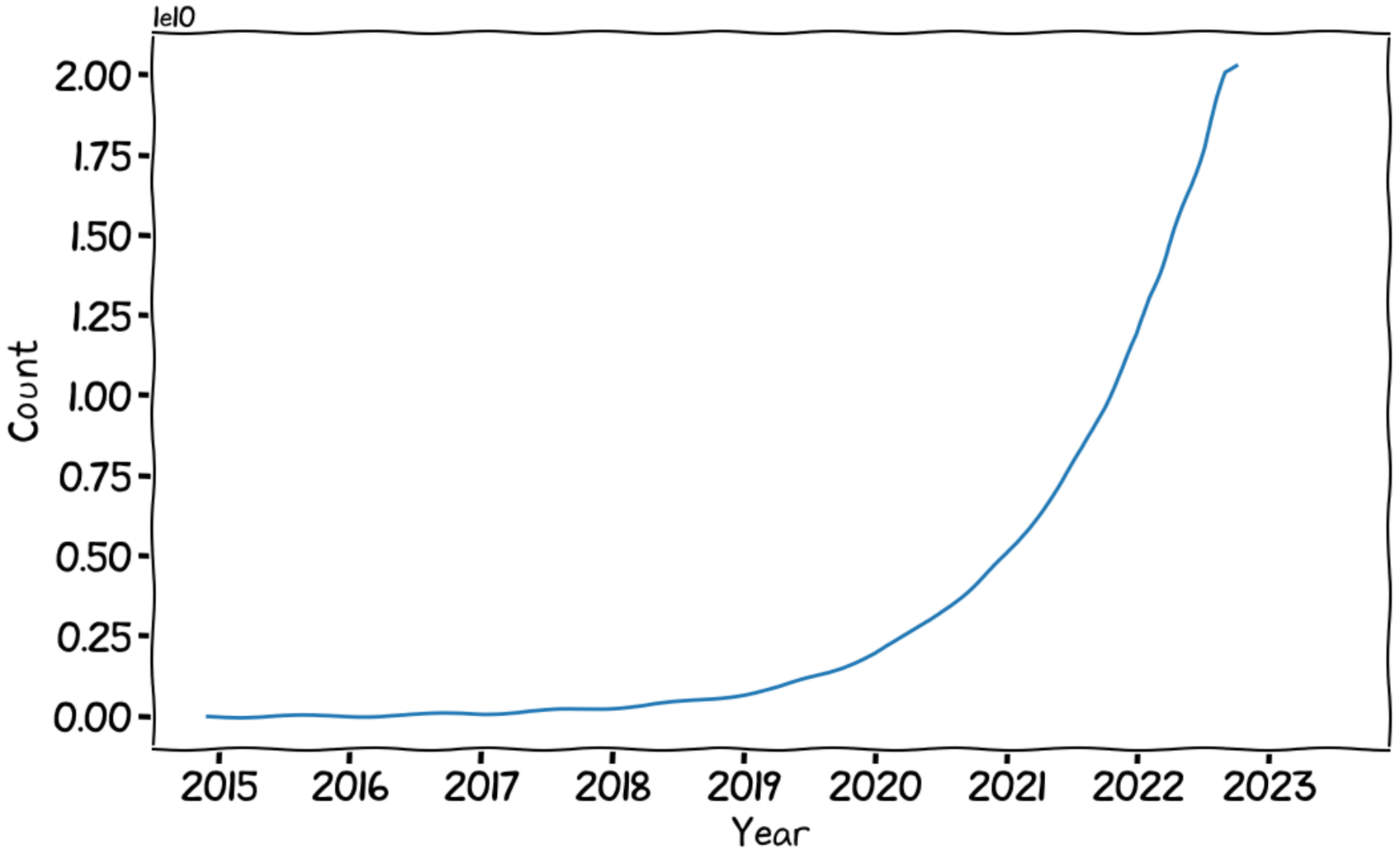
2023 Stack Overflow Survey: Rust is the most admired programming language, making it the most loved language for 8 years in a row

survey.stackoverflow.co/2023/

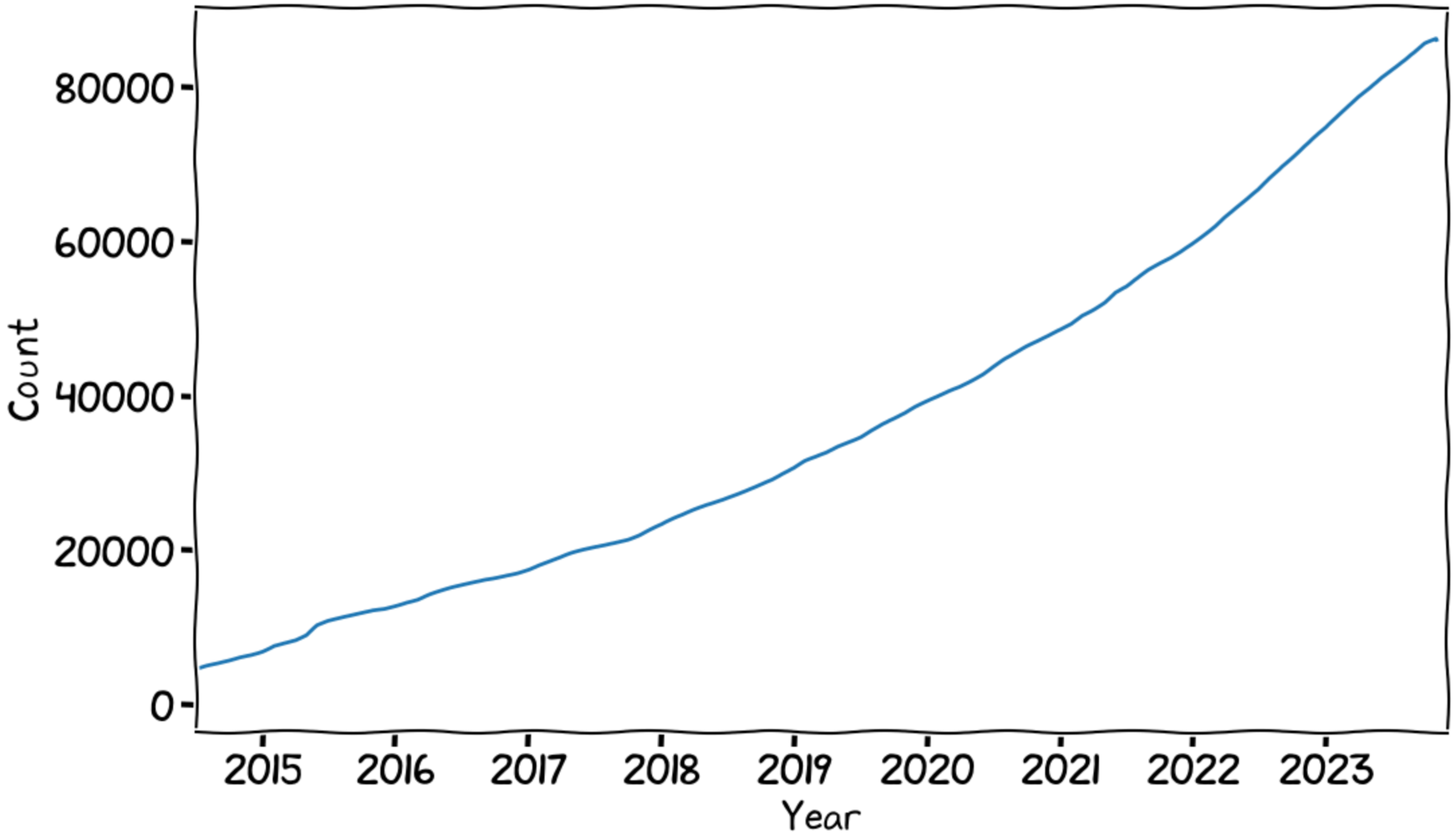
Crate count (120k+)



Crate downloads (20B+)

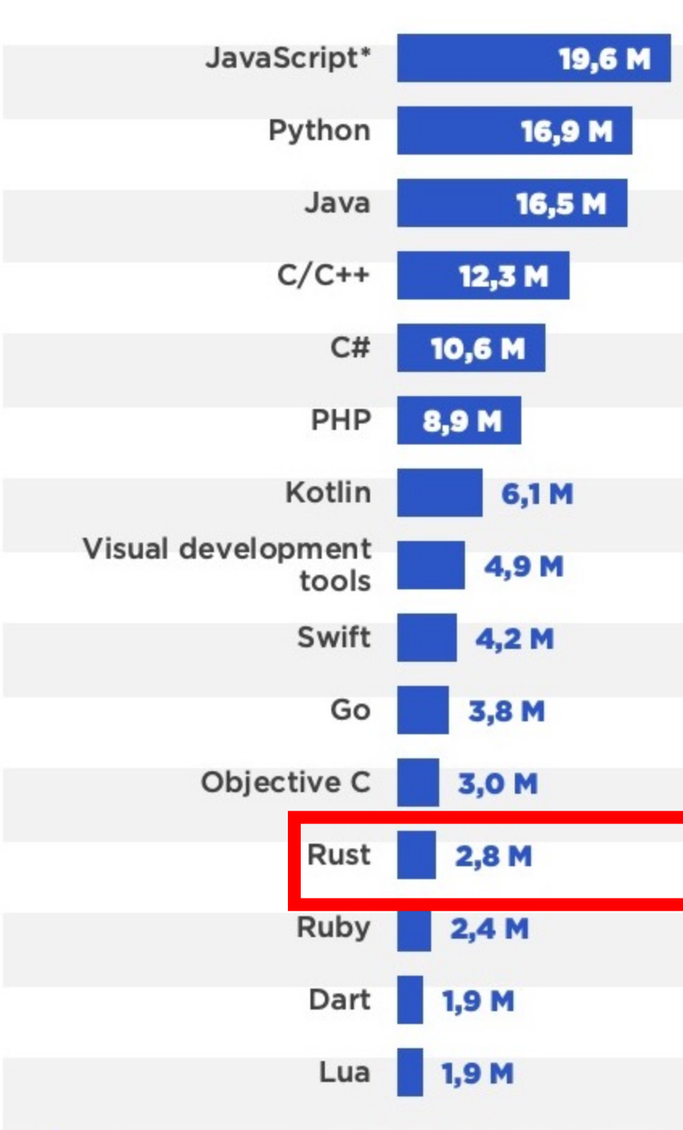


GitHub stars (80k+)



Size of programming language communities in Q3 2022

Active software developers, globally, in millions



	Most popular in	Least popular in
JavaScript*	Apps for 3rd-party ecosystems, Cloud	DS/ML/AI, Embedded
Python	DS/ML/AI, IoT apps	Web, Mobile
Java	Cloud, Desktop	Web, DS/ML/AI
C/C++	Embedded, IoT apps	Cloud, Web
C#	Desktop, Games	DS/ML/AI, IoT devices
PHP	Web, Cloud	Mobile, DS/ML/AI
Kotlin	Mobile, AR/VR	Games, DS/ML/AI
Visual development tools	AR/VR, Games	Embedded, Cloud
Swift	Mobile, AR/VR	Embedded, Cloud
Go	Apps for 3rd-party ecosystems, Cloud	Mobile, DS/ML/AI
Objective C	AR/VR, IoT devices	Desktop, Apps for 3rd-party ecosystems
Rust	AR/VR, IoT apps	Mobile, Web
Ruby	IoT devices, Apps for 3rd-party ecosystems	Embedded, Web
Dart	Mobile, Apps for 3rd-party ecosystems	Web
Lua	IoT devices, AR/VR	Mobile, Embedded

/DATA

** JavaScript includes CoffeeScript and TypeScript*

Rust

A language empowering everyone
to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Reliability

(Memory) Safety

Absence of memory errors

Memory errors

Memory errors

- Null pointer dereference

Memory errors

- Null pointer dereference
- Double-free

Memory errors

- Null pointer dereference
- Double-free
- Use-after-free

Memory errors

- Null pointer dereference
- Double-free
- Use-after-free
- Out-of-bounds access

Memory errors

- Null pointer dereference
- Double-free
- Use-after-free
- Out-of-bounds access
- Iterator invalidation

Memory errors

- Null pointer dereference
- Double-free
- Use-after-free
- Out-of-bounds access
- Iterator invalidation
- …

Memory errors

- Null pointer dereference
- Double-free
- Use-after-free
- Out-of-bounds access
- Iterator invalidation
- …

Caused by undefined behaviour (UB)

Memory errors

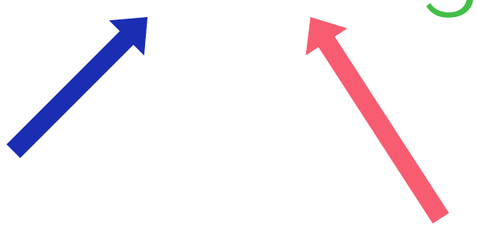
- Null pointer dereference
- Double-free
- Use-after-free
- Out-of-bounds access
- Iterator invalidation
- …

Caused by undefined behaviour (UB)

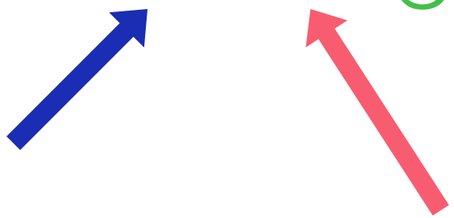
Rust: avoid UB at all costs

Memory errors happen when
aliasing is combined with mutability

Memory errors happen when
aliasing is combined with mutability



Memory errors happen when
aliasing is combined with mUtABil1ty



C++ memory error example

```
std::vector<int> vec = { 1, 2, 3 };
```

C++ memory error example

Aliasing ✓

```
std::vector<int> vec = { 1, 2, 3 };
int& p = vec[0];

std::cout << p << std::endl;
```

C++ memory error example

Mutability ✓

```
std::vector<int> vec = { 1, 2, 3 };

vec.push_back(4);
```

C++ memory error example

Aliasing & Mutability

```
std::vector<int> vec = { 1, 2, 3 };
int& p = vec[0];
vec.push_back(4);
std::cout << p << std::endl;
```

What to do?

Rust's solution

You can mutate

Rust's solution

You can mutate ||

Rust's solution

You can mutate || alias

Rust's solution

You can mutate || alias

But not both at the same time (w.r.t. a single variable)

Rust's solution

You can mutate || alias

But not both at the same time (w.r.t. a single variable)

Rust enforces this at compile time using its type system

Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

Chrome: 70% of all security bugs are memory safety issues

Google software engineers are looking into ways of eliminating memory management-related bugs from Chrome.

Android

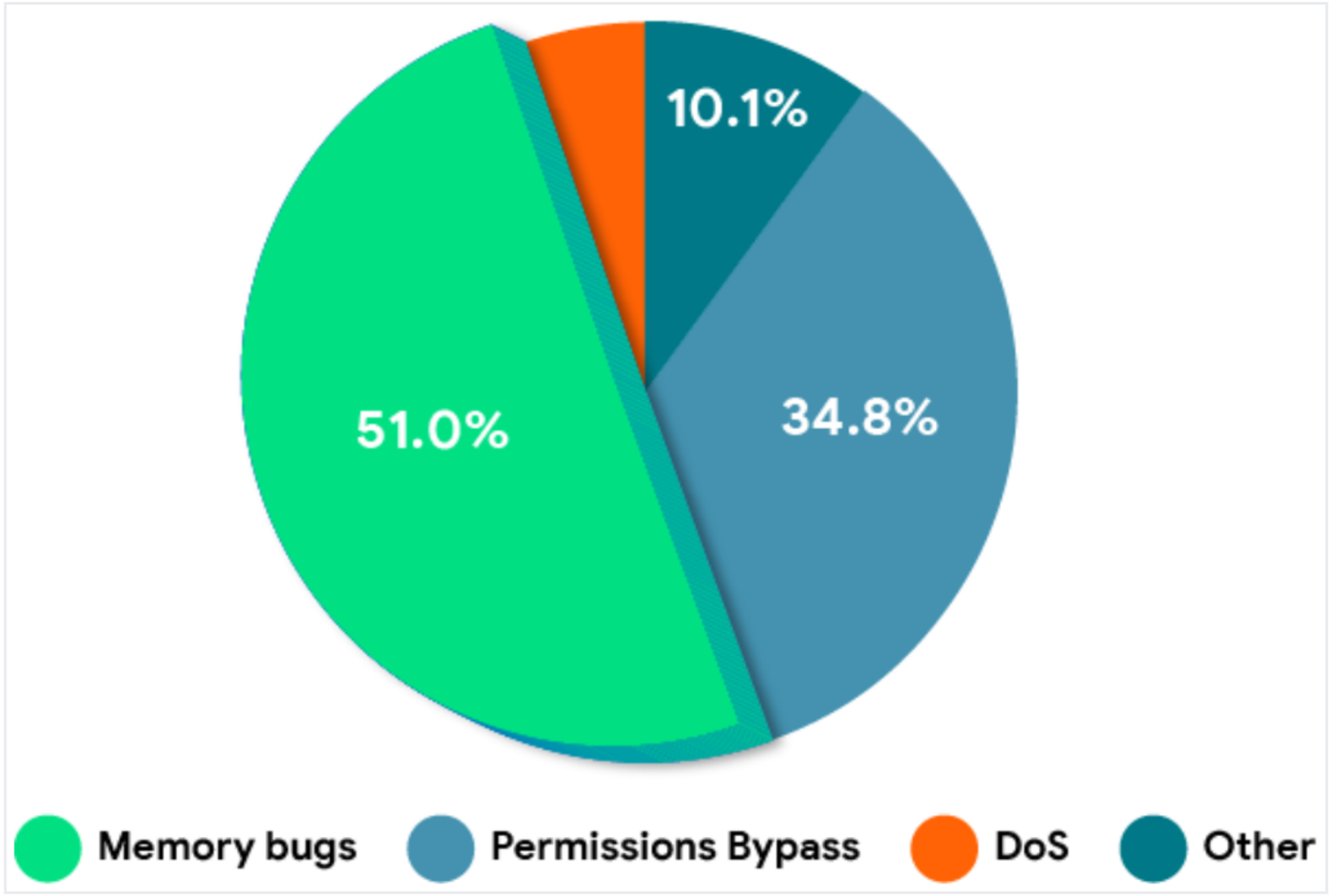


Figure 2: Memory safety bugs contribution to Android vulnerabilities

Reliability

NSA to developers: Think about switching from C and C++ to a memory safe programming language

For many developers, that could mean a shift towards C#, Go, Java, Ruby, Rust, and Swift.

Reliability

A Safer High Performance AV1 Decoder

Josh Aas

Mar 9, 2023

Reliability

Bringing Memory Safety to sudo and su

Josh Aas

Apr 26, 2023

Reliability

A Memory Safe Implementation of the Network Time Protocol

Folkert de Vries

Oct 11, 2022

Reliability

Announcing Hickory DNS

Benjamin Fry

Oct 5, 2023

Reliability

UB in Java

`Iterator::remove`

> **The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress** in any way other than by calling this method, unless an overriding class has specified a concurrent modification policy.
>
> Java docs

UB in Python

`for` cycle

> “
> There is a subtlety when the **sequence is being modified by the loop** (this can only occur for mutable sequences, e.g. lists).
> …**This can lead to nasty bugs**…
>
> Python docs ”

Reliability

```
let mut items = vec![1, 2, 3];
for item in items.iter() {
    items.push(1);
}
```

Reliability

```rust
let mut items = vec![1, 2, 3];
for item in items.iter() {
    items.push(1);
}
```

```
error[E0502]: cannot borrow `items` as mutable because it is also borrowed as immutable
 --> src/lib.rs:4:9
  |
3 |     for item in items.iter() {
  |                 ------------
  |                 |
  |                 immutable borrow occurs here
  |                 immutable borrow later used here
4 |         items.push(1);
  |         ^^^^^^^^^^^^^ mutable borrow occurs here
  |
```

Soundness

API that cannot be misused

Ownership

```
file.close();
```

Ownership

```
file.close();

…
let data = file.read();
```

Ownership

```
file.close();

…
let data = file.read();
```

```
error[E0382]: borrow of moved value: `file`
  --> src/lib.rs:18:16
   |
14 |     let mut file = File;
   |         -------- move occurs because `file` has type `ownership::File`, which does not i
mplement the `Copy` trait
...
17 |     file.close();
   |          ------- `file` moved due to this method call
18 |     let data = file.read();
   |                ^^^^^^^^^^^ value borrowed here after move
   |
note: `ownership::File::close` takes ownership of the receiver `self`, which moves `file`
  --> src/lib.rs:11:18
   |
```

Sum types ("enums on steroids")

```
enum PrintJob {




}
```

Sum types ("enums on steroids")

```
enum PrintJob {
    Created {
        source: Document
    },
}
```

Sum types ("enums on steroids")

```
enum PrintJob {
    Created {
        source: Document
    },
    Finished {
        printer: Printer,
        printed_pages: u32
    },




}
```

Sum types ("enums on steroids")

```
enum PrintJob {
    Created {
        source: Document
    },
    Finished {
        printer: Printer,
        printed_pages: u32
    },
    Failed {
        printer: Printer,
        error: PrintError
    }
}
```

Pattern matching

```
let printer = match job {
    PrintJob::Created  { .. } => None,
    PrintJob::Finished { printer, .. } |
    PrintJob::Failed   { printer, .. } => Some(printer)
};
```

No implicit `null`

No implicit `null`

```
fn max(items: &[u32]) ->
```

No implicit `null`

```
fn max(items: &[u32]) -> Option<usize>
```

No implicit `null`

```
fn max(items: &[u32]) -> Option<usize>
```

```
enum Option<T> {
    Some(T),
    None
}
```

No implicit `null`

```
fn max(items: &[u32]) -> Option<usize>
```

```
enum Option<T> {
    Some(T),
    None
}
```

```
match max(items) {
    Some(value) => ...,
    None => println!("No maximum found")
}
```

Forces you to think about possible errors

Forces you to think about possible errors

```
match read_file("src.txt") {
    Ok(content) => write_file("dst.txt", content),
    Err(Error::PermissionDenied) => eprintln!("Permission denied"),
    Err(Error::NotFound) => eprintln!("File not found"),
    Err(e) => eprintln!("Unknown error has occurred: {e:?}")
}
```

Forces you to think about possible errors

```
match read_file("src.txt") {
    Ok(content) => write_file("dst.txt", content),
    Err(Error::PermissionDenied) => eprintln!("Permission denied"),
    Err(Error::NotFound) => eprintln!("File not found"),
    Err(e) => eprintln!("Unknown error has occurred: {e:?}")
}
```

But tries to make it ergonomic

```
let content = read_file("src.txt")?;
write_file("dst.txt", content)?;
```

Fearless concurrency

Safe(r) mutexes

```
let value = Mutex::new(vec![1, 2]);
```

Safe(r) mutexes

```
let value = Mutex::new(vec![1, 2]);
value.lock().push(3);
```

Compile-time data race detection

```
let mut items = vec![1, 2];
```

Compile-time data race detection

```
let mut items = vec![1, 2];

std::thread::spawn(|| items.push(3));
```

Compile-time data race detection

```
let mut items = vec![1, 2];

std::thread::spawn(|| items.push(3));
items.push(4);
```

Compile-time data race detection

```
let mut items = vec![1, 2];

std::thread::spawn(|| items.push(3));
items.push(4);
```

Compile-time error!

Confidence

Confidence

- "If it compiles, it works"

Confidence

- "If it compiles, it works"
- Easier code review

Confidence

- "If it compiles, it works"
- Easier code review
- Better discipline in other languages

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Zero-cost abstractions

Zero-cost abstractions

> “What you don’t use, you don’t pay for.
> What you do use, you couldn’t hand code any better.”
>
> Bjarne Stroustrup

No GC

Performance

(LLVM) optimizations

```
pub fn dot_product(x: &[u32], y: &[u32]) -> u32 {
    x.iter().zip(y).map(|(&a, &b)| a * b).sum::<u32>()
}
```

Performance

(LLVM) optimizations

```rust
pub fn dot_product(x: &[u32], y: &[u32]) -> u32 {
    x.iter().zip(y).map(|(&a, &b)| a * b).sum::<u32>()
}
```

```
vmovdqu ymm4, ymmword ptr [rdx + 4*rax]
vmovdqu ymm6, ymmword ptr [rdx + 4*rax + 64]
vmovdqu ymm5, ymmword ptr [rdx + 4*rax + 32]
vmovdqu ymm7, ymmword ptr [rdx + 4*rax + 96]
vpmulld ymm8, ymm5, ymmword ptr [rdi + 4*rax + 32]
vpmulld ymm4, ymm4, ymmword ptr [rdi + 4*rax]
vpmulld ymm6, ymm6, ymmword ptr [rdi + 4*rax + 64]
vpmulld ymm5, ymm7, ymmword ptr [rdi + 4*rax + 96]
add     rax, 32
vpaddd  ymm0, ymm4, ymm0
vpaddd  ymm1, ymm8, ymm1
```

Parallelization

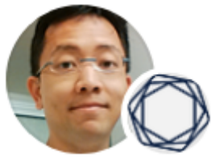
```
fn sum_of_squares(input: &[i32]) -> i32 {
    input.iter()
         .map(|&i| i * i)
         .sum()
}
```

Parallelization

```
fn sum_of_squares(input: &[i32]) -> i32 {
    input.par_iter()
         .map(|&i| i * i)
         .sum()
}
```

Performance

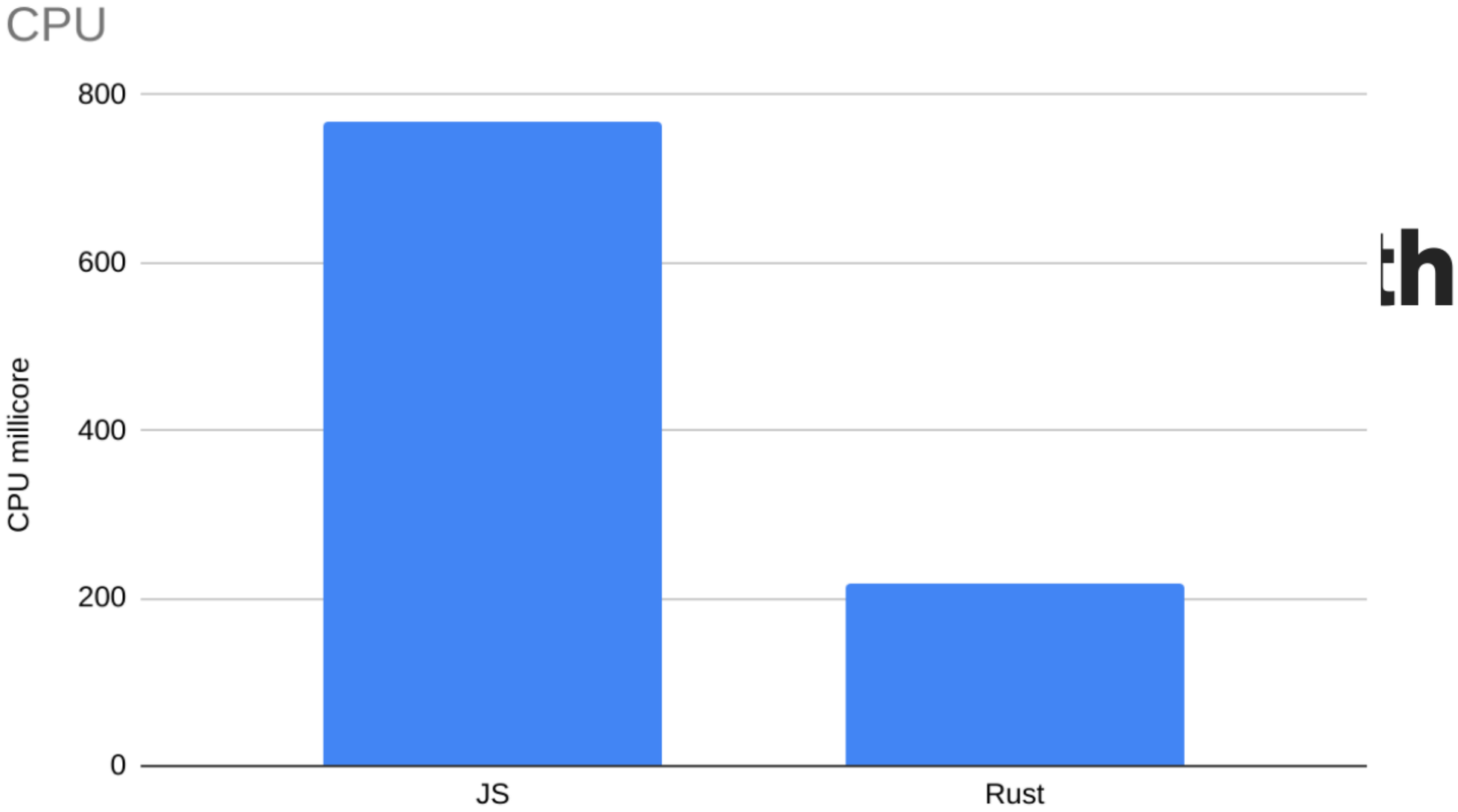
Optimizing 700 CPUs Away With Rust



Alan Ning · Follow

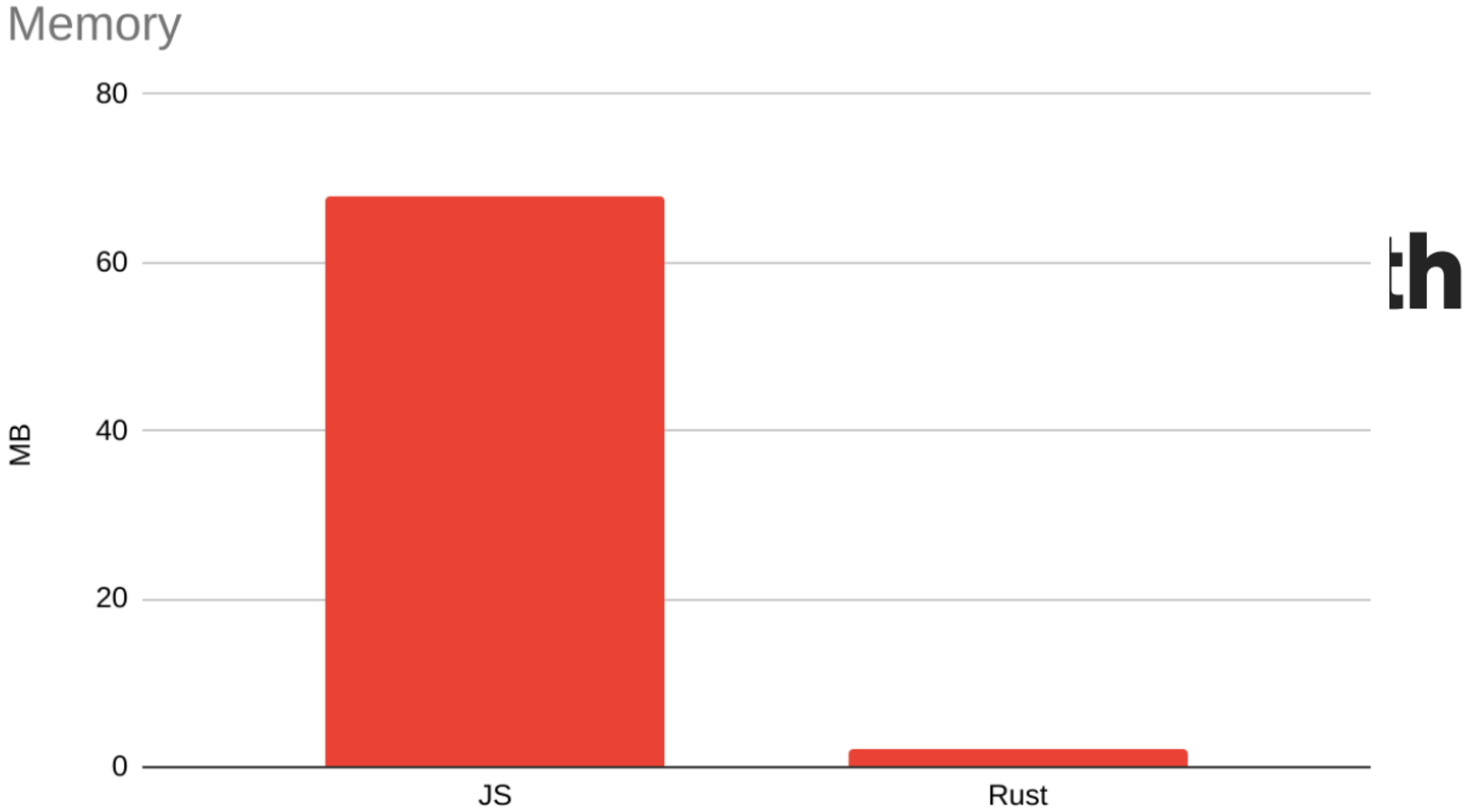
Published in Tenable TechBlog · 3 min read · May 6, 2021

Performance



Per pod, average CPU reduced from 800m to 200m core

Performance



Per pod, average memory reduced from 70MB to 5MB.

“
With this small change, we were able
to optimize away over 700 CPU and 300GB of memory.
This was all implemented, tested and deployed in two weeks.

Tenable ”

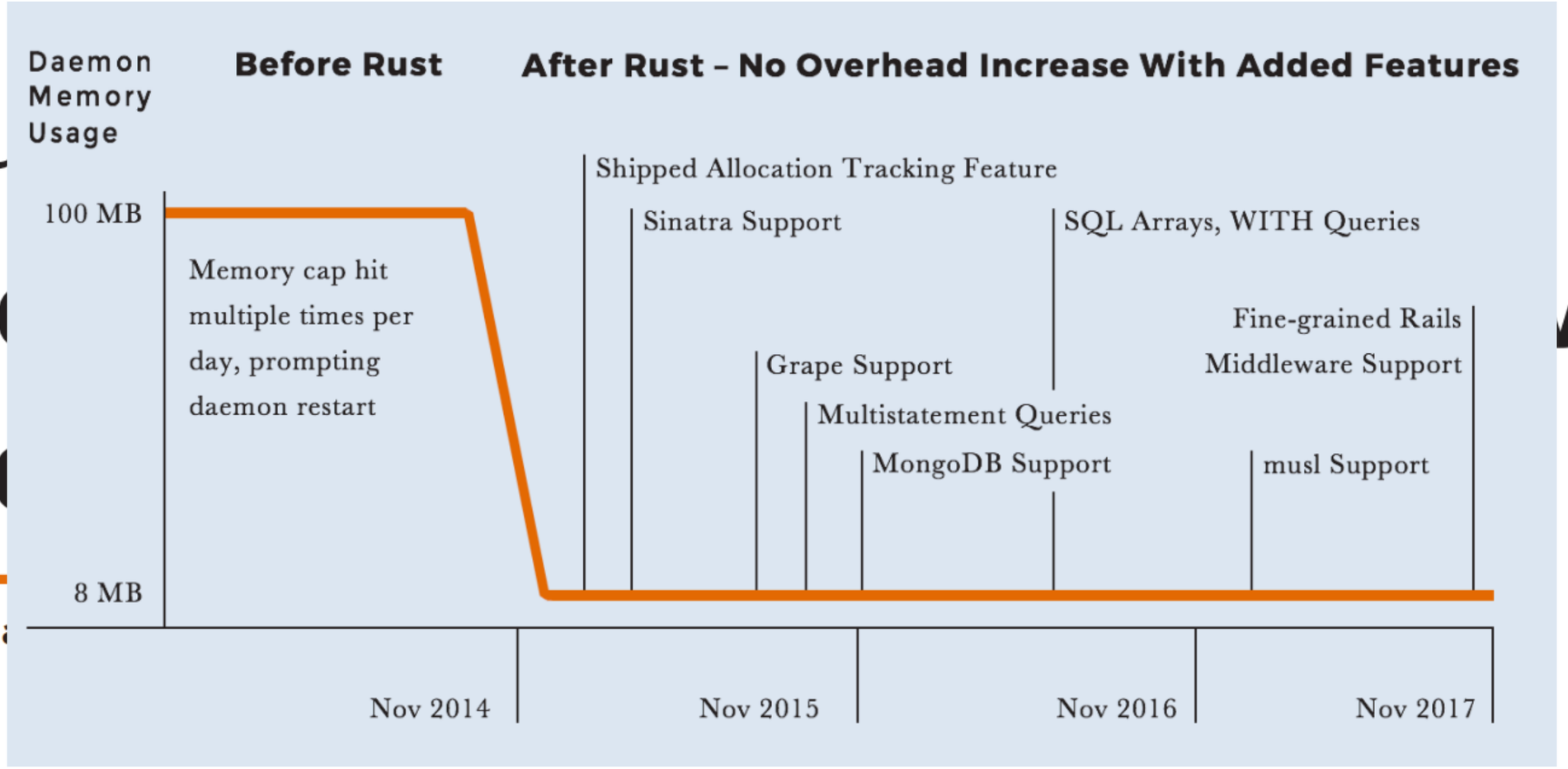
Performance

Rust Case Study:

How Rust is Tilde's Competitive Advantage

The analytics startup innovates safely with the help of Rust

Performance



Performance

ENGINEERING & DEVELOPERS

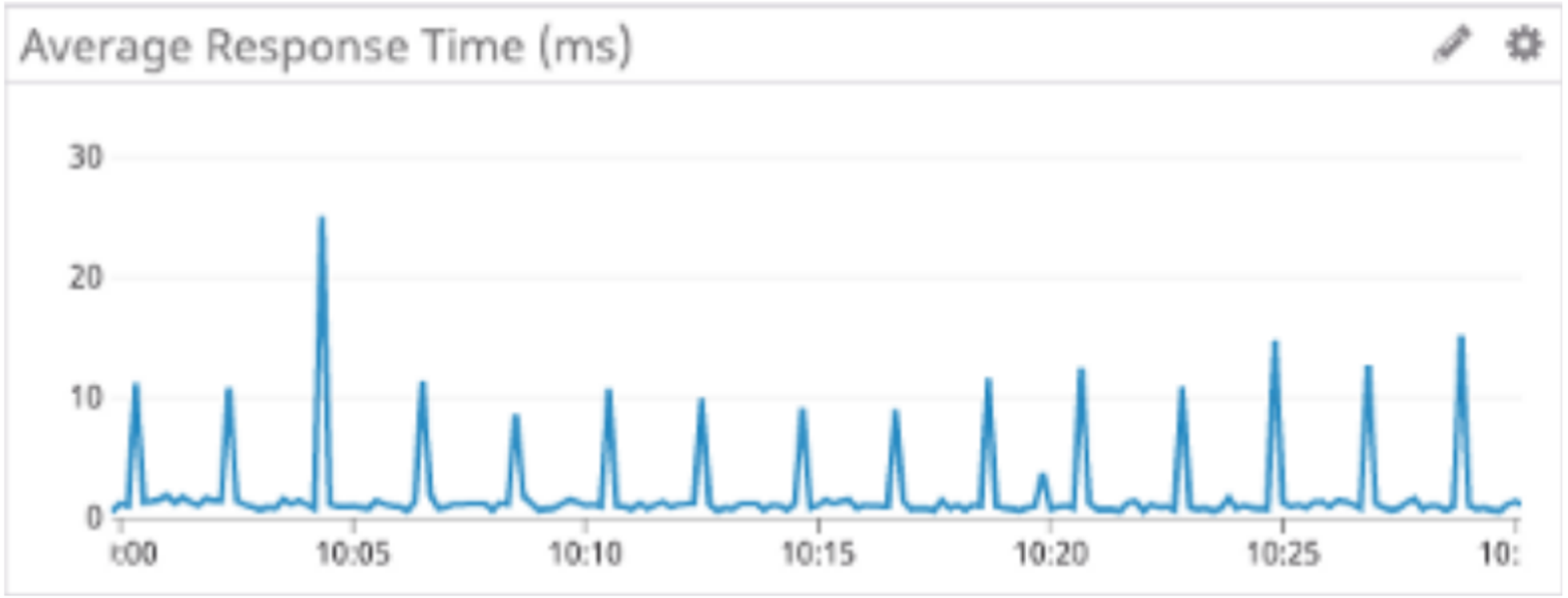
WHY DISCORD IS SWITCHING FROM GO TO RUST

Jesse Howarth
February 4, 2020

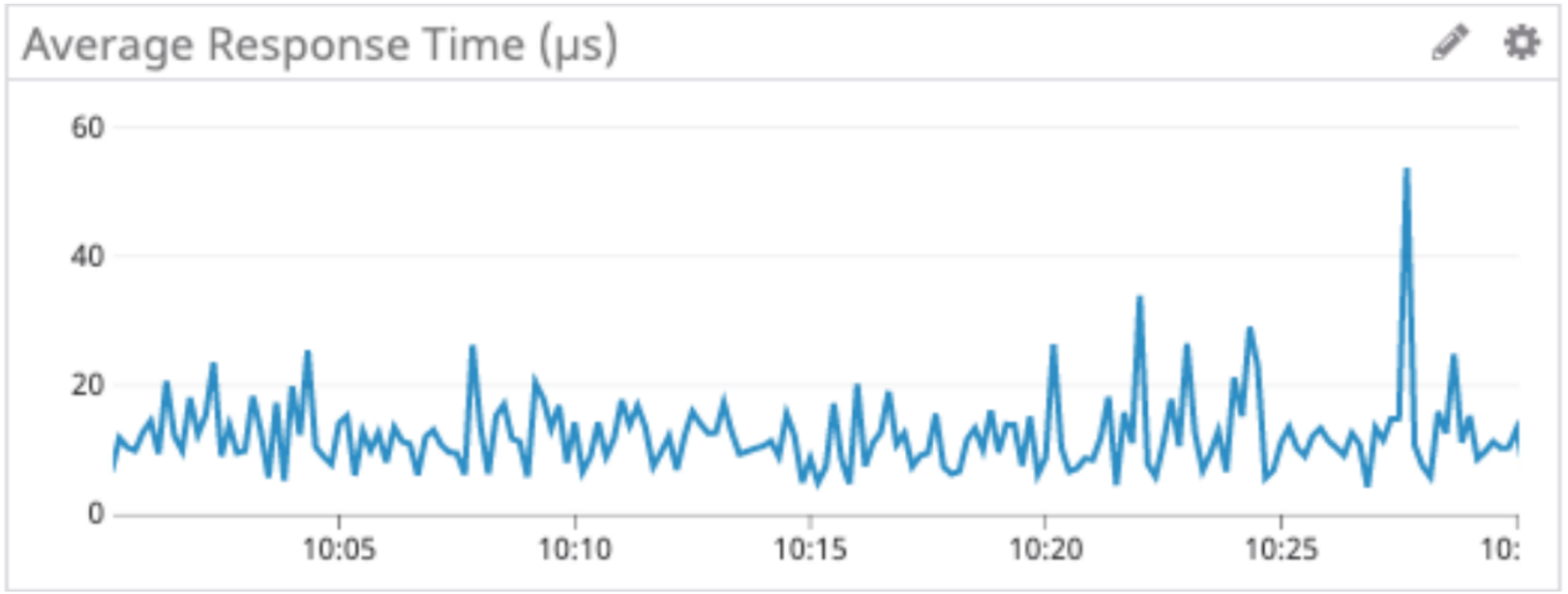
Rust is becoming a first class language in a variety of domains. At Discord, we've seen success with Rust on the client side and server side. For example, we use it on the client side for our video encoding pipeline for Go Live and on the server side for Elixir NIFs.

Performance

Go



Rust

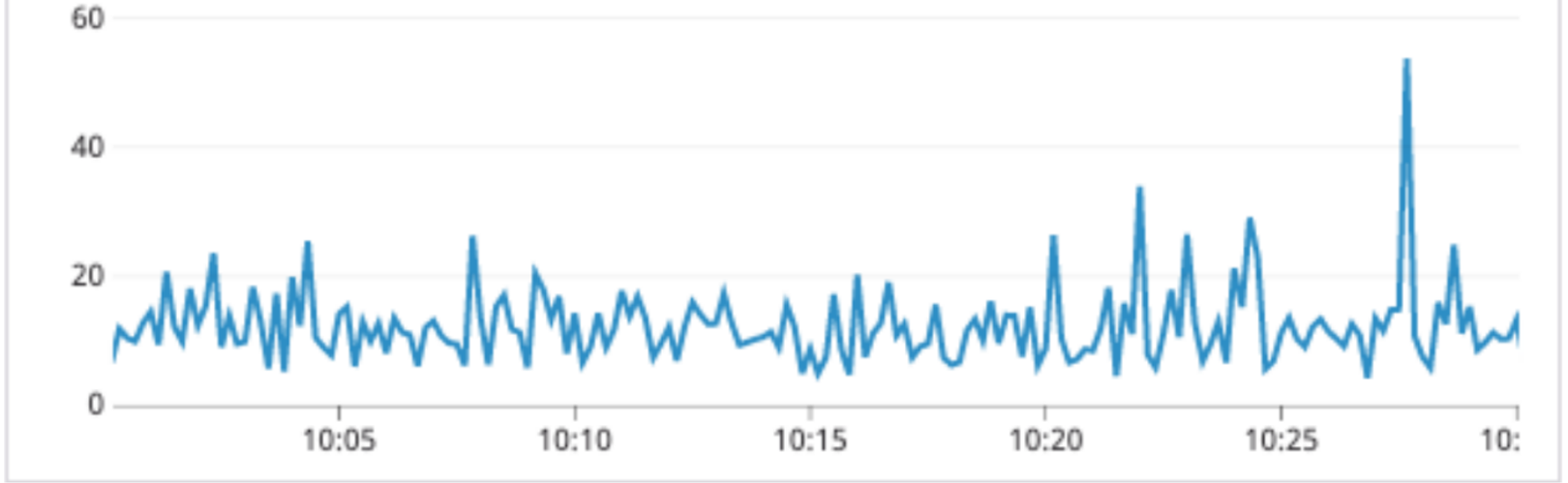


Performance

Go

> “
> Notice the average time is now measured in microseconds.
>
> Discord ”

Rust



Performance

Daniele Frasca 🇮🇹🇩🇪🇪🇺
@dfrasca80

Follow

After the Rust talk at @awsugnl, I have run new load testing. To get alike perf
Rust 256MB for 2.5B invocations:
- is 41% cheaper than #NodeJS at 1GB
- is 34% cheaper than #golang at 1GB
- is 17% cheaper than #golang at 512MB
- Faster between 8% and 25% (warm)
#serverless #aws

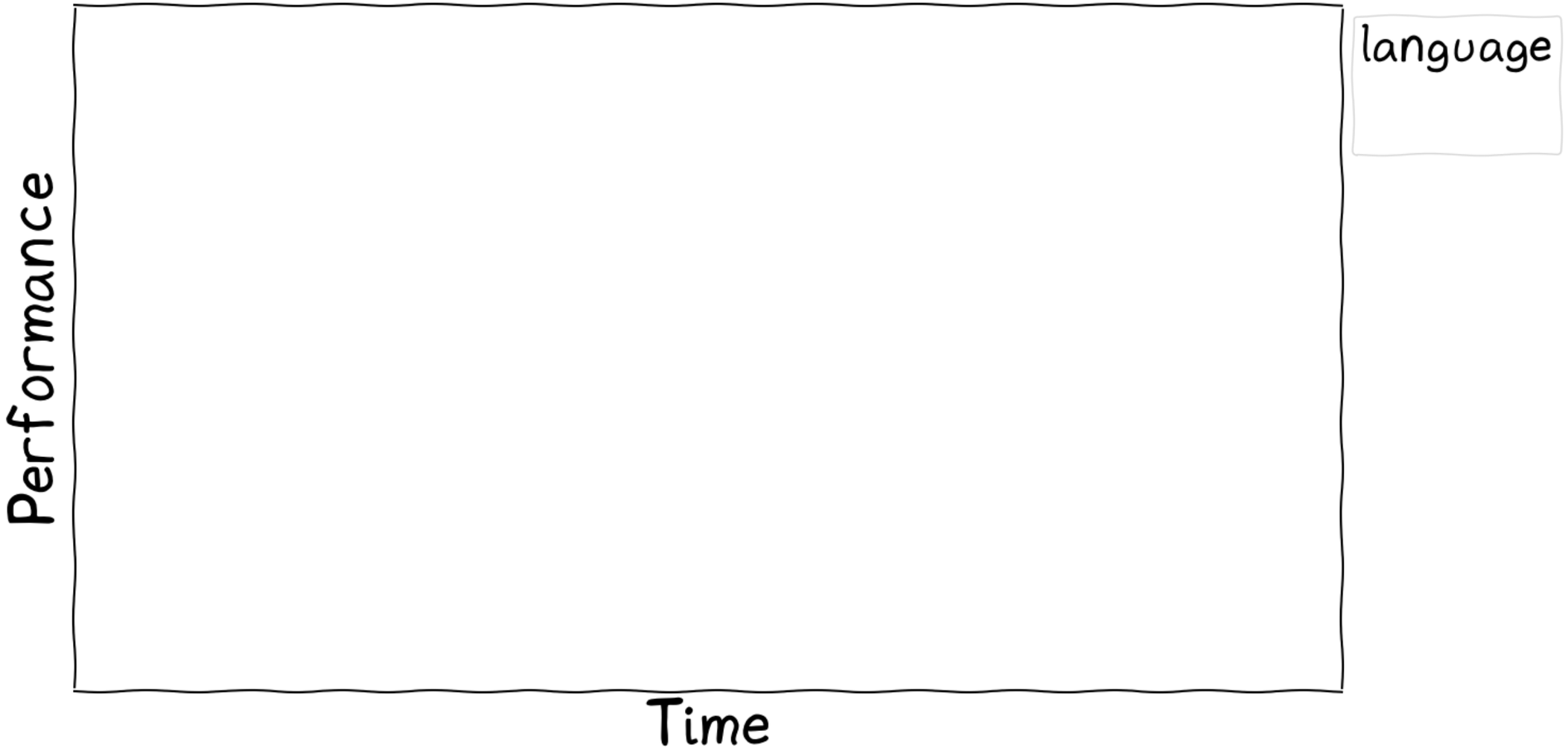
Performance

My Node.js is a bit Rusty

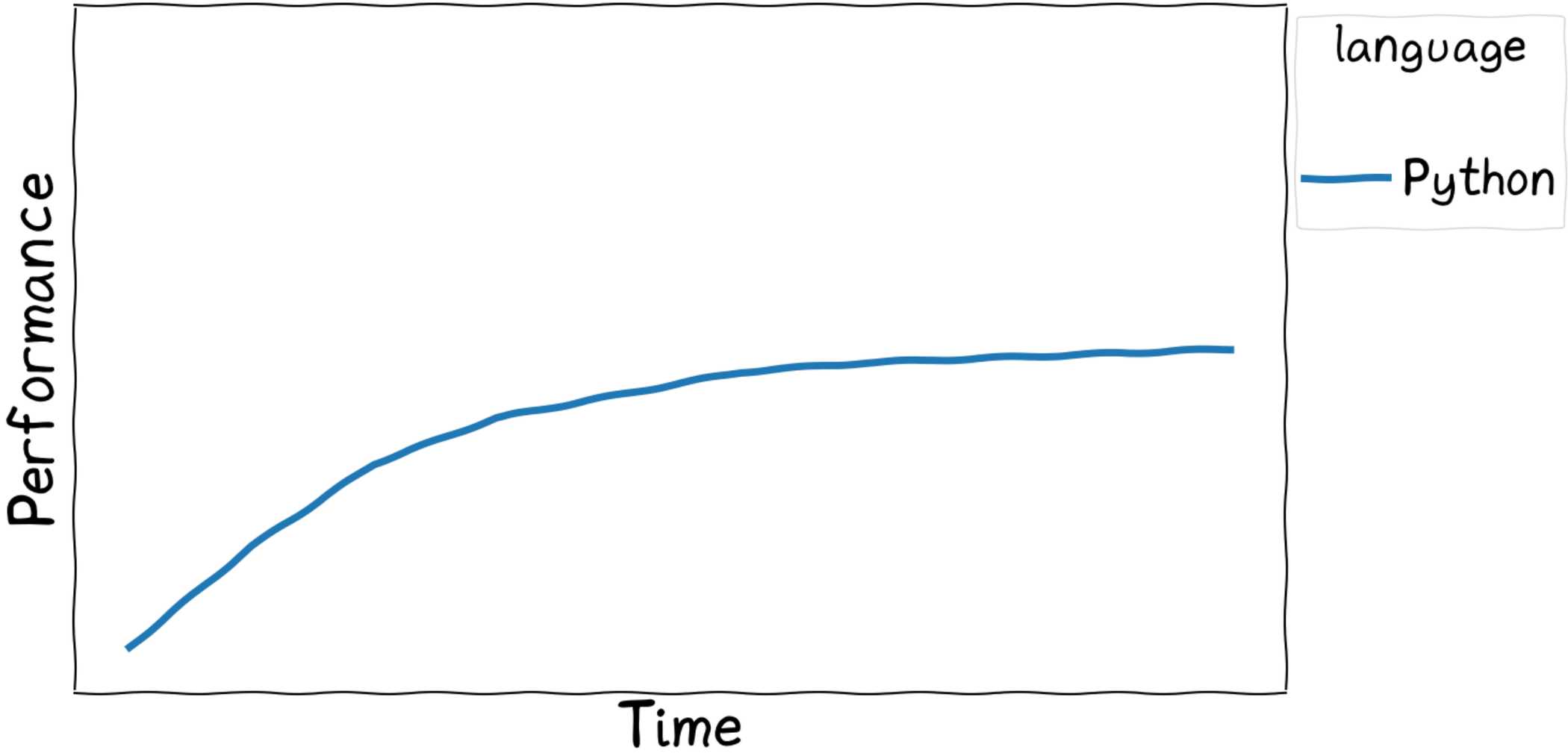
Replacing an internal Node.js module with a native Rust module made a x25 perf boost. Let's understand why.

September 4, 2023.

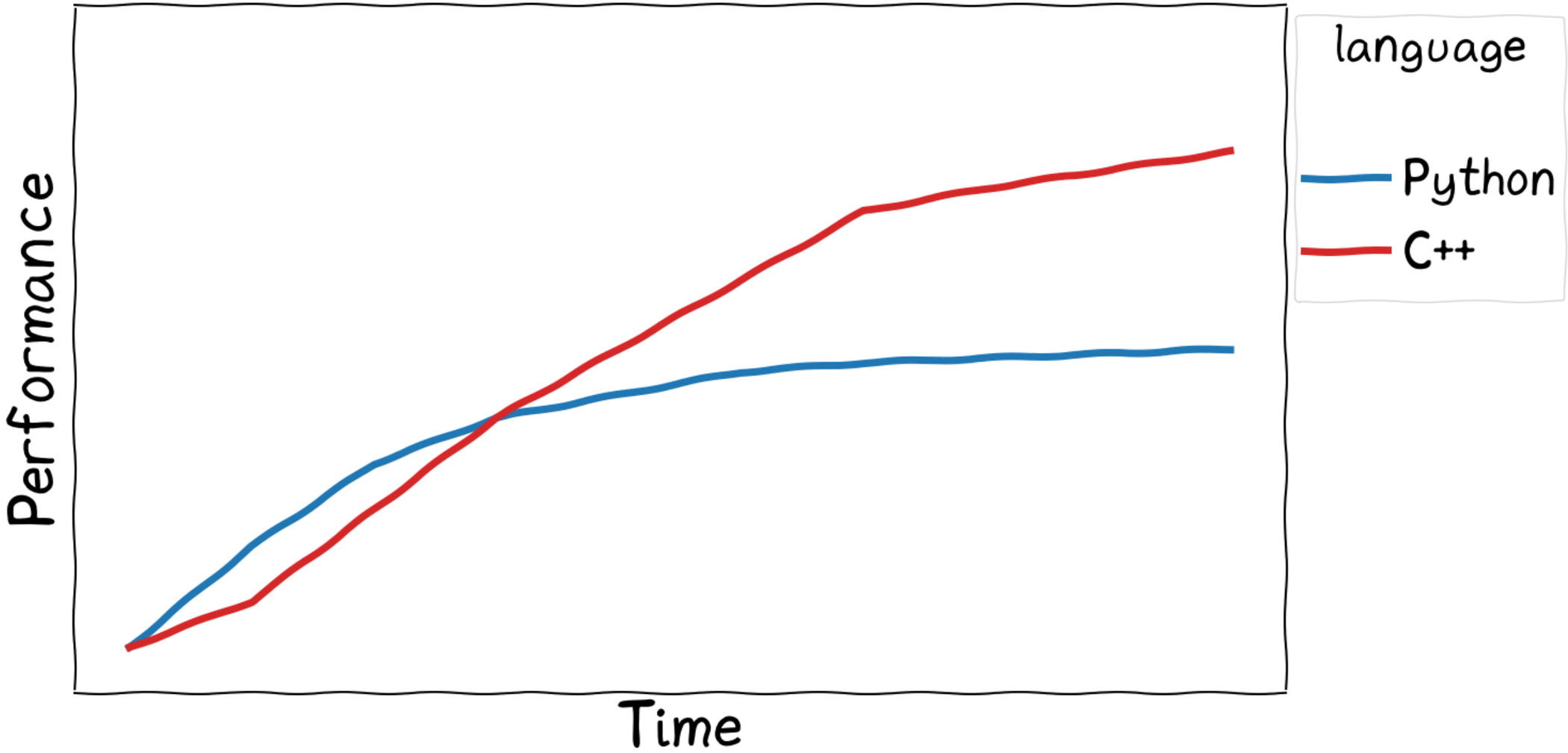
Time to performance

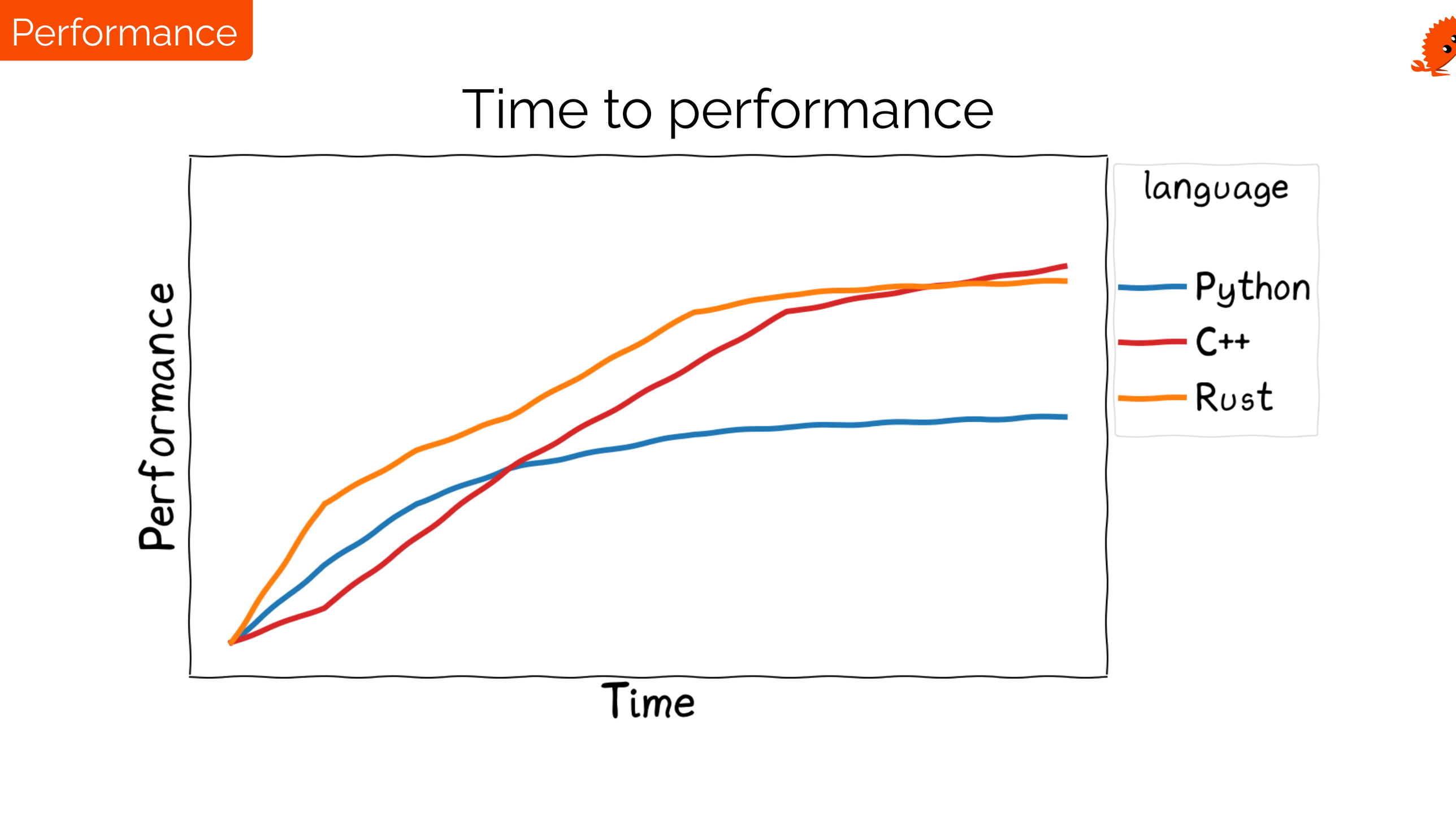


Time to performance



Time to performance





Time to performance

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Cargo

Integrated package manager

Type check

```
$ cargo check
```

Build

```
$ cargo build
```

Build

```
$ cargo build
```

```
> ls target/debug/
helloworld
```

Ergonomics

Run

```
$ cargo run
```

Ergonomics

```rust
#[test]
fn test_add() {
    assert_eq!(add(1, 2), 3);
}
```

+

Test

```
$ cargo test
```

Ergonomics

```rust
#[test]
fn test_add() {
    assert_eq!(add(1, 2), 3);
}
```

+

Test

```
$ cargo test
```

```
running 1 test
test test_add ... ok
```

Ergonomics

```rust
#[bench]
fn bench_add(b: &mut Bencher) {
    b.iter(|| add(1, 2));
}
```

+

Benchmark

```
$ cargo bench
```

Ergonomics

```rust
#[bench]
fn bench_add(b: &mut Bencher) {
    b.iter(|| add(1, 2));
}
```

\+

Benchmark

```
$ cargo bench
```

```
running 1 test
test bench_factorial ... bench:          17 ns/iter (+/- 9)
```

Format code

```
$ cargo fmt
```

Ergonomics

```rust
/// This function does something
fn my_func(a: u32) -> u32 {
    a + 1
}
```

+

Build documentation

```
$ cargo doc
```

Implementations

[–] `impl String` source

[–] `pub const fn new() -> String` const: 1.39.0 · source

Creates a new empty `String`.

Given that the `String` is empty, this will not allocate any initial buffer. While that means that this initial operation is very inexpensive, it may cause excessive allocation later when you add data. If you have an idea of how much data the `String` will hold, consider the `with_capacity` method to prevent excessive re-allocation.

Examples

```
let s = String::new();
```

[–] `pub fn with_capacity(capacity: usize) -> String` source

Creates a new empty `String` with at least the specified capacity.

`String`s have an internal buffer to hold their data. The capacity is the length of that buffer, and can be queried with the `capacity` method. This method creates an empty `String`, but one with an initial buffer that can hold at least `capacity` bytes. This is useful when you may be appending a bunch of data to the `String`, reducing the number of reallocations it needs to do.

If the given capacity is `0`, no allocation will occur, and this method is identical to the `new` method.

Ergonomics

Library (crate) documentation

`https://docs.rs/<crate-name>`

Custom commands

```
$ cargo semver-checks
```

Ergonomics

```
    Starting 8 checks, version 0.1.0 -> 0.1.0 (no change)
        PASS [   0.000s]       major        enum_missing
        FAIL [   0.000s]       major        enum_variant_added
        PASS [   0.000s]       major        enum_variant_missing
        PASS [   0.000s]       major        struct_marked_non_exhaustive
        PASS [   0.000s]       major        struct_missing
        PASS [   0.000s]       major        struct_pub_field_missing
        PASS [   0.000s]       major        unit_struct_changed_kind
        PASS [   0.000s]       major        variant_marked_non_exhaustive
     Summary [   0.003s] 8 checks run: 7 passed, 1 failed, 0 skipped

--- failure enum_variant_added: enum variant added on exhaustive enum ---

Description:
A publicly-visible enum without #[non_exhaustive] has a new variant.
        ref: https://doc.rust-lang.org/cargo/reference/semver.html#enum-variant-new
       impl: https://github.com/obi1kenobi/cargo-semver-check/tree/v0.4.1/src/queries/enum_va

Failed in:
  variant EnumWithNewVariant:NewVariant in src/test_cases/enum_variant_added.rs:5

       Final [   0.003s] semver requires new major version: 1 major and 0 minor checks failed
```

Cargo.toml

```
[package]
name = "hello_world"
version = "0.1.0"

[dependencies]
json = "1.0"
```

Ergonomics

Cargo.toml

```
[package]
name = "hello_world"
version = "0.1.0"

[dependencies]
json = "1.0"
```

main.rs

```
use json::parse;

fn main() {
    parse(...);
}
```

Ergonomics

Cargo.toml

```
[package]
name = "hello_world"
version = "0.1.0"

[dependencies]
json = "1.0"
```

main.rs

```
use json::parse;

fn main() {
    parse(...);
}
```

120k+ crates available on crates.io

Error messages

```
error[E0616]: field `len` of struct `Vec` is private
 --> src/main.rs:2:14
  |
2 |     if items.len > 5 {
  |              ^^^ private field
  |
help: a method `len` also exists, call it with parentheses
  |
2 |     if items.len() > 5 {
  |                  ++
  |
```

Error messages

```
error[E0499]: cannot borrow `items` as mutable more than once at a
time
  --> src/main.rs:40:13
   |
38 |     for item in items.iter_mut() {
   |                 ---------------
   |                 |
   |                 first mutable borrow occurs here
   |                 first borrow later used here
39 |         if *item == 2 {
40 |             items.remove(0);
   |             ^^^^^^^^^^^^^^^ second mutable borrow occurs here
   |
```

Error messages

```
error[E0382]: borrow of moved value: `file`
  --> src/main.rs:25:5
   |
23 | fn fn3(mut file: File) {
   |        -------- move occurs because `file` has type `File`, wh
ich does not implement the `Copy` trait
24 |     file.close();
   |          ------- `file` moved due to this method call
25 |     file.read();
   |     ^^^^^^^^^^^ value borrowed here after move
   |
note: `File::close` takes ownership of the receiver `self`, which m
oves `file`
```

Error messages

```
error[E0597]: `value` does not live long enough
  --> src/main.rs:32:21
   |
31 |         let value = 5;
   |             ----- binding `value` declared here
32 |         reference = &value;
   |                     ^^^^^^ borrowed value does not live long e
nough
33 |     }
   |     - `value` dropped here while still borrowed
34 |     println!("{reference}");
   |                ----------- borrow later used here
```

Error messages

```
error[E0277]: `Rc<i32>` cannot be shared between threads safely
   --> src/main.rs:49:24
    |
49  |         std::thread::spawn(|| {
    |  _____-------------------_^
    | |     |
    | |     required by a bound introduced by this call
50  | |         *val += 1
51  | |     });
    | |_____^ `Rc<i32>` cannot be shared between threads safely
    |
    = help: the trait `Sync` is not implemented for `Rc<i32>`
    = note: required for `&Rc<i32>` to implement `Send`
```

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

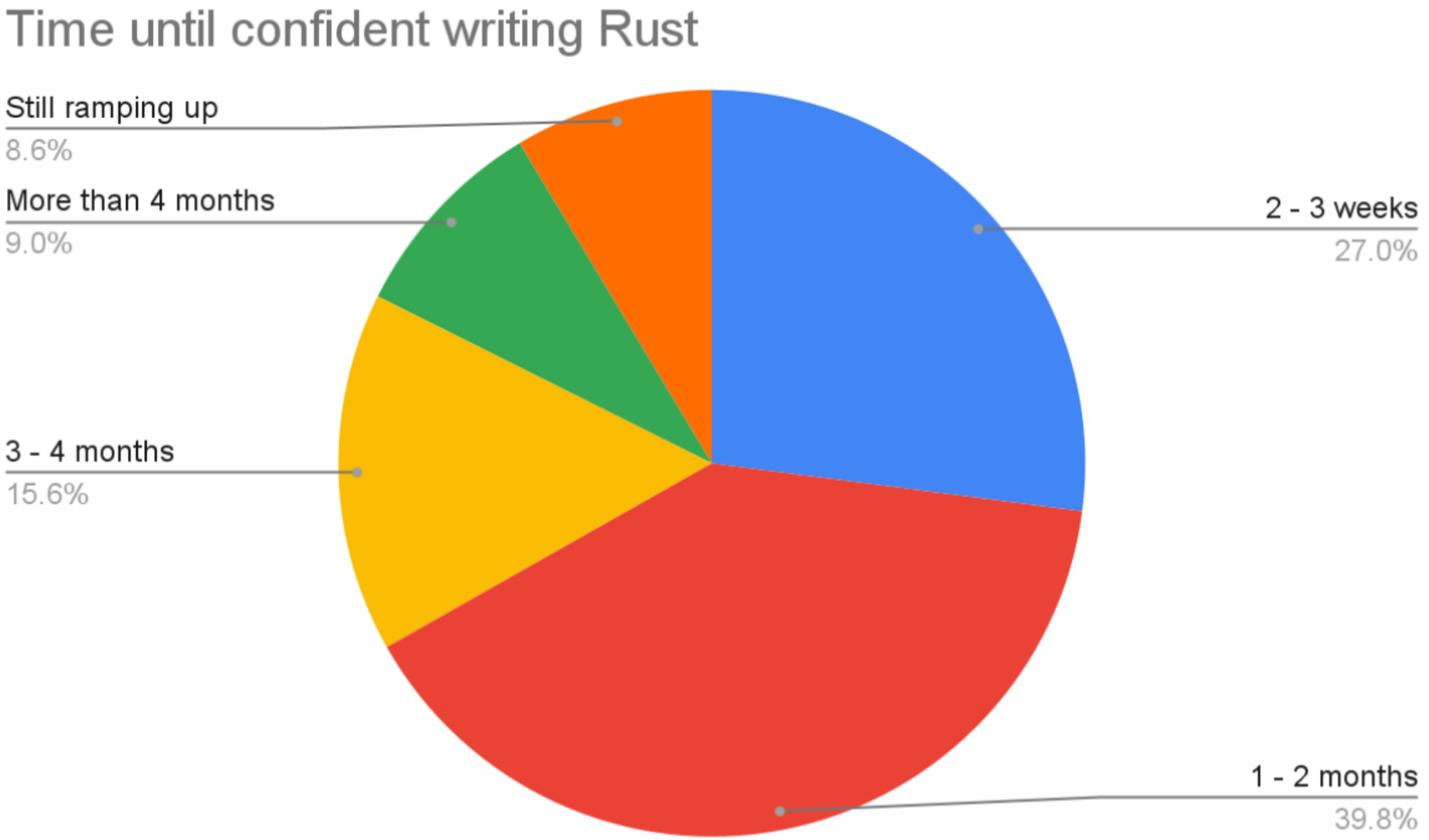
Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Why not Rust?

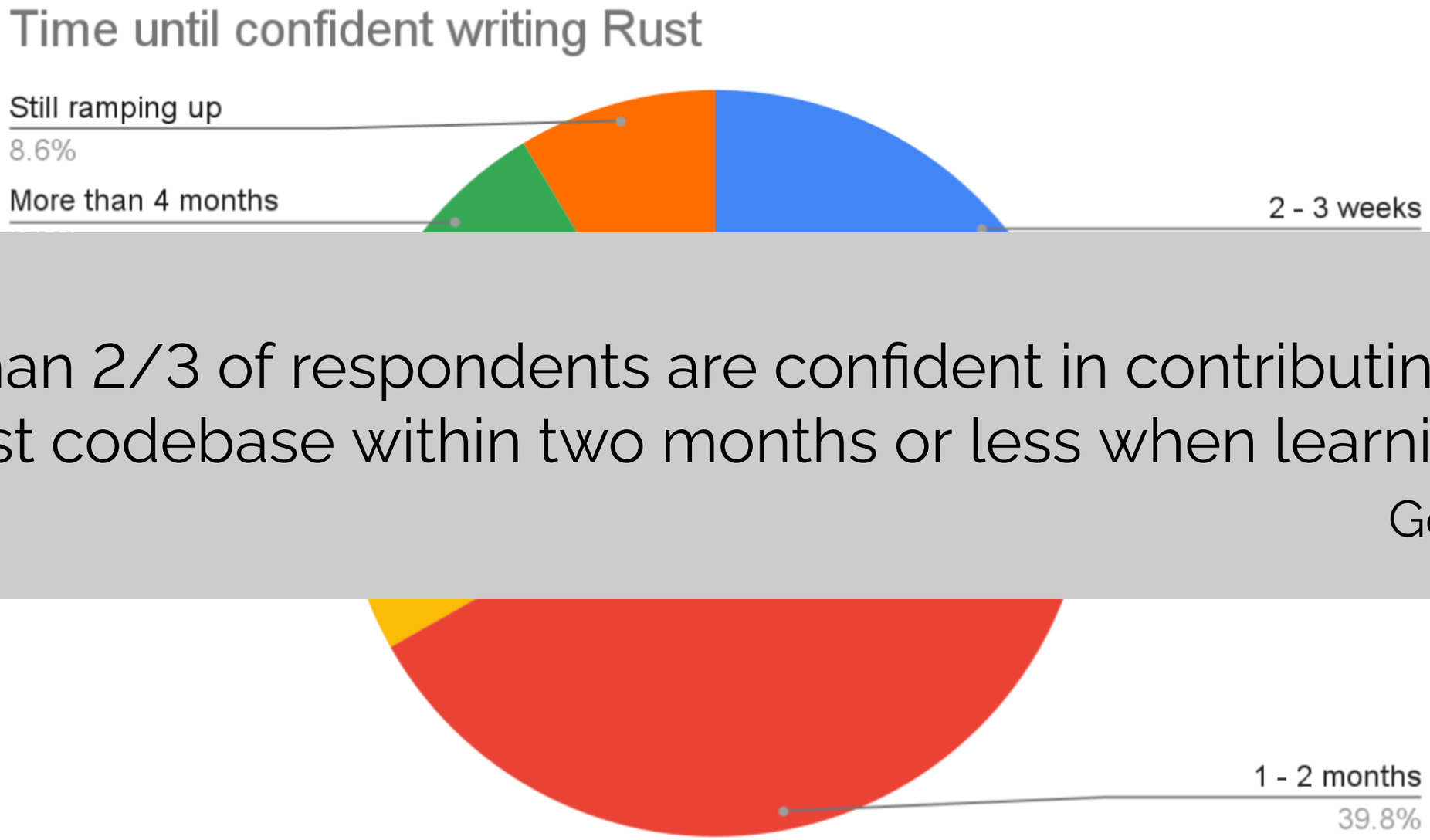
Why not Rust?

- Learning curve

Why not Rust?


Time until confident writing Rust
Still ramping up
8.6%
More than 4 months
9.0%
3 - 4 months
15.6%
2 - 3 weeks
27.0%
1 - 2 months
39.8%

Why not Rust?



> “More than 2/3 of respondents are confident in contributing to a Rust codebase within two months or less when learning Rust.”
>
> Google

Why not Rust?

- Learning curve
  - Functional paradigm

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow
  - Compile -> Run cycle

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow
  - Compile -> Run cycle
  - More development up front, less production bugs

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow
  - Compile -> Run cycle
  - More development up front, less production bugs
- Ecosystem

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow
  - Compile -> Run cycle
  - More development up front, less production bugs
- Ecosystem
  - Symfony/Django/Spring/ASP.NET (?)

Why not Rust?

- Learning curve
  - Functional paradigm
  - No inheritance :-)
- Programming workflow
  - Compile -> Run cycle
  - More development up front, less production bugs
- Ecosystem
  - Symfony/Django/Spring/ASP.NET (?)
- Job Market

Where to try Rust?

Where to try Rust?

- Libraries

Where to try Rust?

- Libraries
- CLI applications

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless
- Tooling, infrastructure

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless
- Tooling, infrastructure
- Data analysis

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless
- Tooling, infrastructure
- Data analysis
- Perf. critical Python/Node.js

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless
- Tooling, infrastructure
- Data analysis
- Perf. critical Python/Node.js
- Embedded

Where to try Rust?

- Libraries
- CLI applications
- Web services (REST, GraphQL, …)
- WebAssembly
- Cloud functions, serverless
- Tooling, infrastructure
- Data analysis
- Perf. critical Python/Node.js
- Embedded
- …

Accelerate Node.js

```rust
use napi_derive::napi;

#[napi]
fn fibonacci(n: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
             fibonacci(n - 2),
    }
}
```

Accelerate Node.js

```rust
use napi_derive::napi;

#[napi]
fn fibonacci(n: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
            fibonacci(n - 2),
    }
}
```

+

```ts
import { fibonacci } from './rusty.js'

console.log(fibonacci(5))
```

Accelerate Python

```rust
use pyo3::pyfunction;

#[pyfunction]
fn fibonacci(a: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
             fibonacci(n - 2),
    }
}
```

Accelerate Python

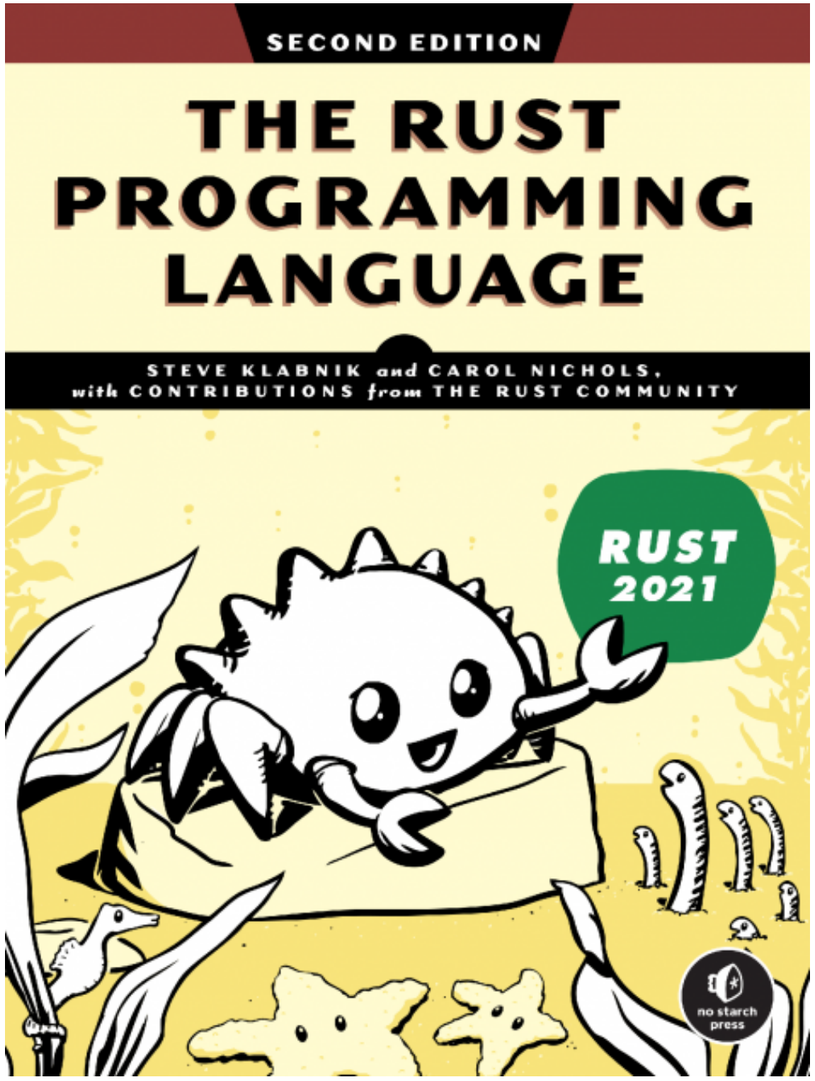
```rust
use pyo3::pyfunction;

#[pyfunction]
fn fibonacci(a: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
             fibonacci(n - 2),
    }
}
```

+

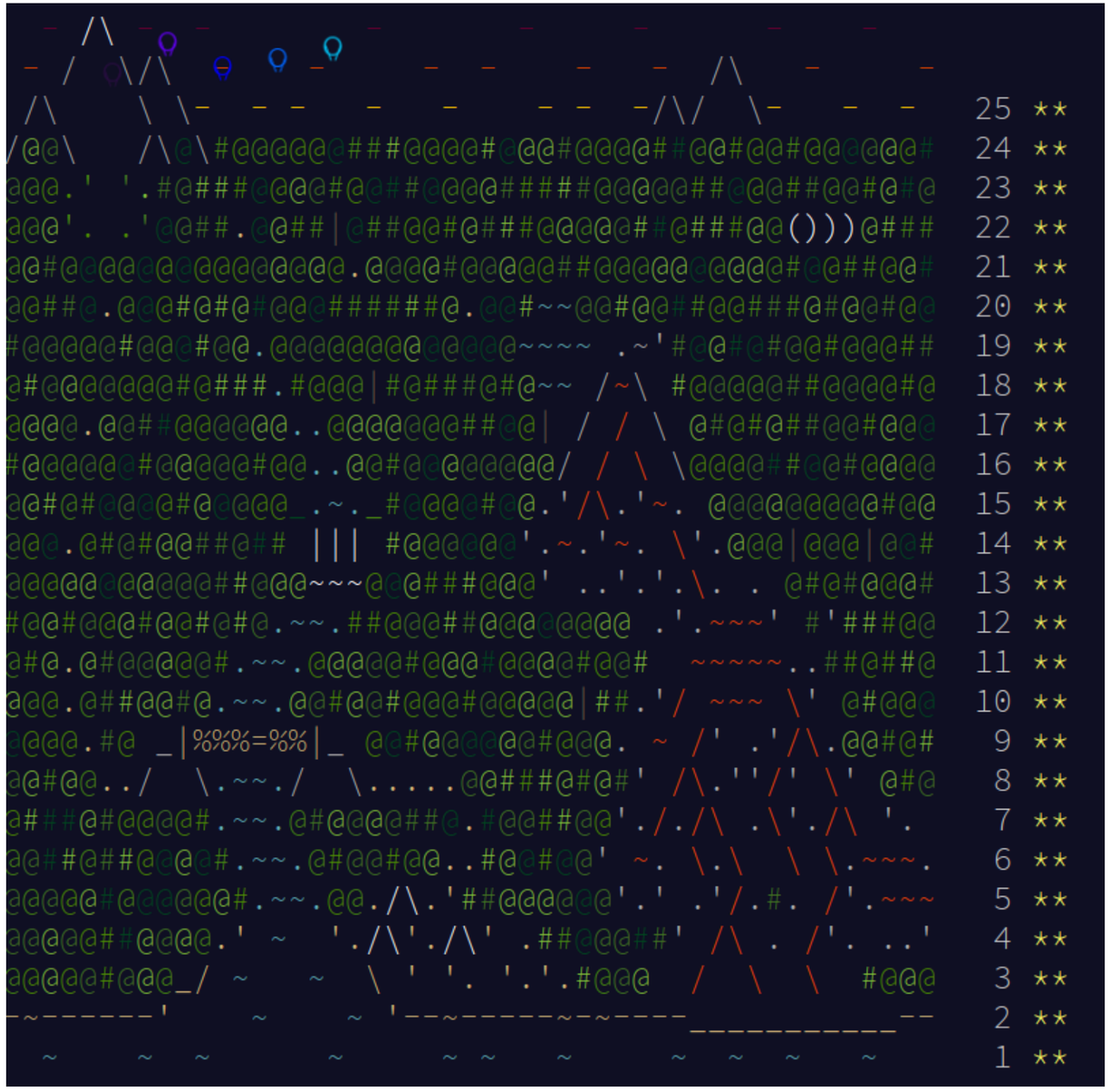
```python
import rusty
print(rusty.fibonacci(5))
```

The Rust Programming Language

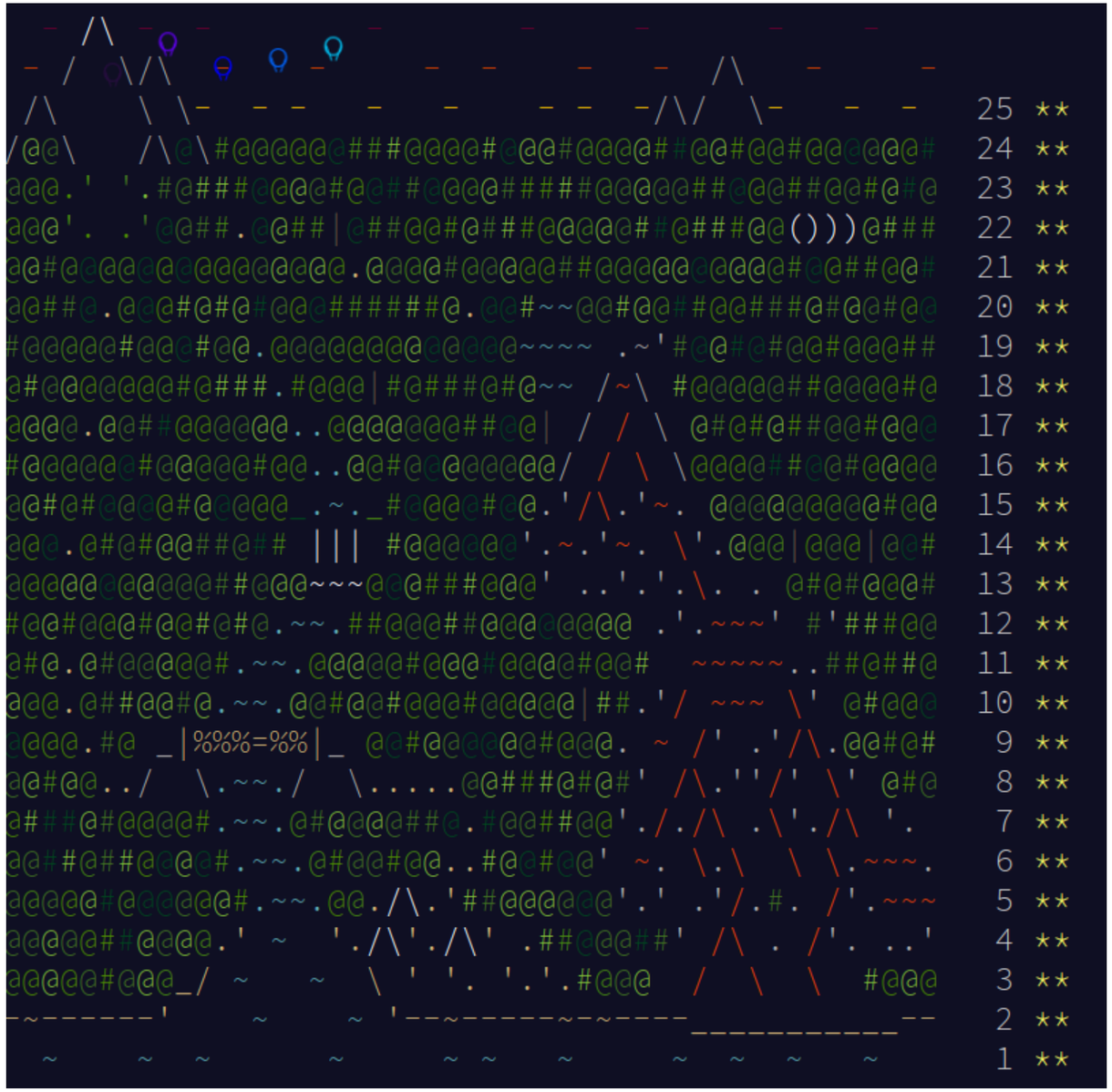


Rust course @ FEI VSB-TUO

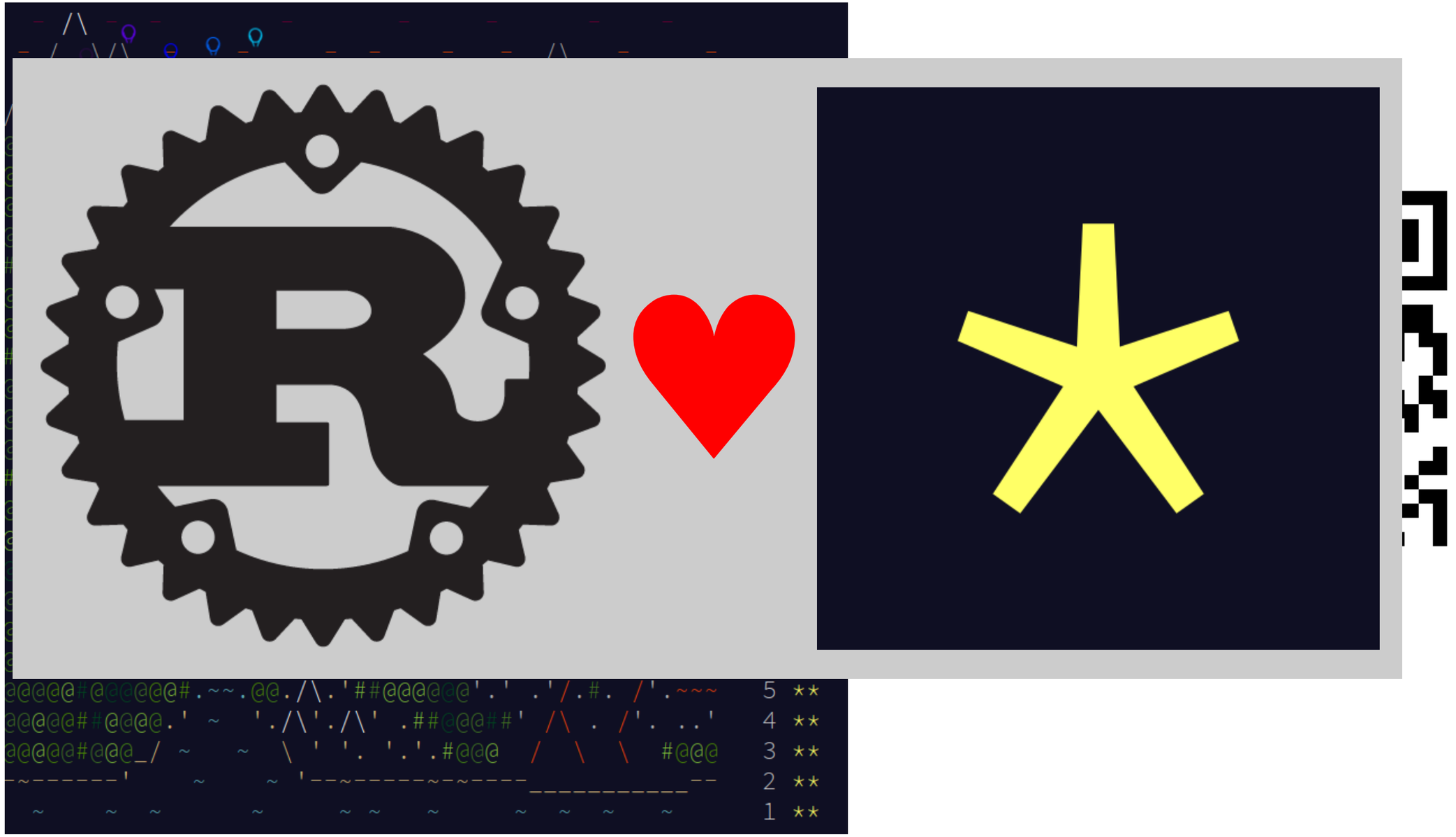
`github.com/kobzol/rust-course-fei`

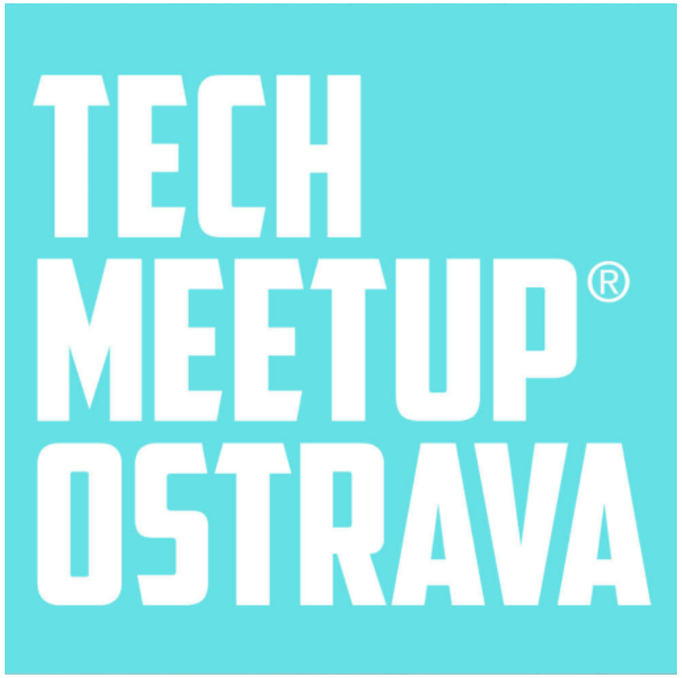

25 **
24 **
23 **
22 **
21 **
20 **
19 **
18 **
17 **
16 **
15 **
14 **
13 **
12 **
11 **
10 **
9 **
8 **
7 **
6 **
5 **
4 **
3 **
2 **
1 **

Advent of Code



Advent of Code




TECH
MEETUP®
OSTRAVA

+

+

Beginning of 2024 (?)
Stay tuned :-)

Děkuji za pozornost

Slidy jsou dostupné zde:

Slidy byly vytvořeny pomocí `github.com/spirali/elsie`