

# Co vám může nabídnout Rust?

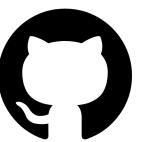
Kuba Beránek



[github.com/kobzol](https://github.com/kobzol)

**Kuba Beránek**



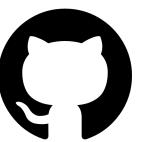


[github.com/kobzol](https://github.com/kobzol)

# Kuba Beránek

- PhD student @ VSB-TUO





[github.com/kobzol](https://github.com/kobzol)

# Kuba Beránek

- PhD student @ VSB-TUO
- Researcher @ IT4Innovations





[github.com/kobzol](https://github.com/kobzol)

# Kuba Beránek

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





[github.com/kobzol](https://github.com/kobzol)

# Kuba Beránek

- PhD student @ 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



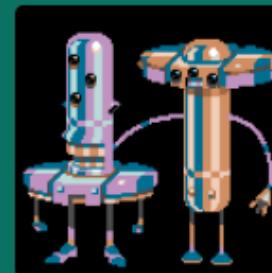
**Jakub Beránek**

GitHub: [Kobzol](#)



**Rémy Rakic**

GitHub: [lqd](#)



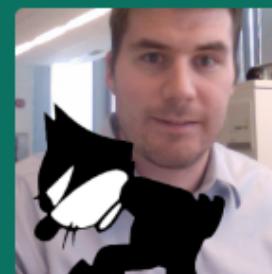
**Michael Woerister**

GitHub: [michaelwoerister](#)



**Nicholas Nethercote**

GitHub: [nnethercote](#)



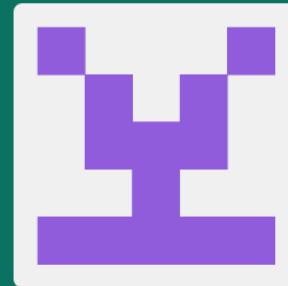
**Felix Klock**

GitHub: [pnkfelix](#)

# Parallel rustc working group

Making parallel compilation the default for rustc

## Members



**Camille Gillot**

GitHub: [cjgillot](#)

Team leader



**Jakub Beránek**

GitHub: [Kobzol](#)



**Sparrow Li**

GitHub: [SparrowLii](#)



**bjorn3**

GitHub: [bjorn3](#)

nek

woerister

[woerister](#)

**Nicholas  
Nethercote**

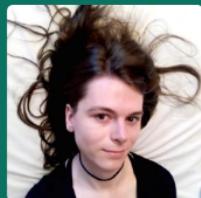
GitHub: [nnethercote](#)

# Parallel Rust working group

## Binary size working group

Improving the binary size of Rust programs and libraries

### Members



**Mara Bos**

GitHub: [m-ou-se](#)

Team leader



**David Wood**

GitHub: [davidtwco](#)



**Jakub Beránek**

GitHub: [Kobzol](#)



**Nicholas Nethercote**

GitHub: [nnethercote](#)



**Jubilee**

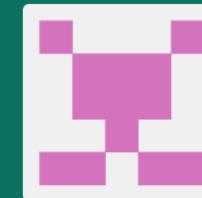
GitHub: [workingjubilee](#)



**Thom Chiovoloni**

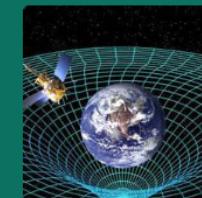
GitHub: [thomcc](#)

Team leader



**h1467792822**

GitHub: [h1467792822](#)



**Gary Guo**

GitHub: [nbdd0121](#)



**Wayne Wu**

GitHub: [wain303009](#)

nek

perister  
woerister

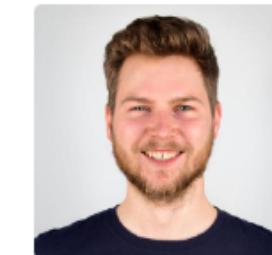
# Parallel Rust working group

## Binary size working group

### 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

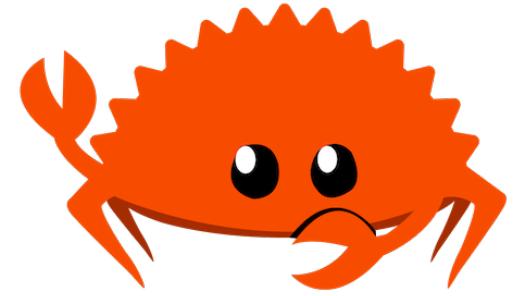
- Static + strong typing
- OOP + FP paradigms
- Syntax based on C
- Compiled to native code
- No GC, minimal runtime

# 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



```
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>](mailto:graydon@mozilla.com)

2006

2010

# Project Servo



Technology from the past  
come to save the future  
from itself

Mozilla Annual Summit, July 2010  
[<graydon@mozilla.com>](mailto:graydon@mozilla.com)



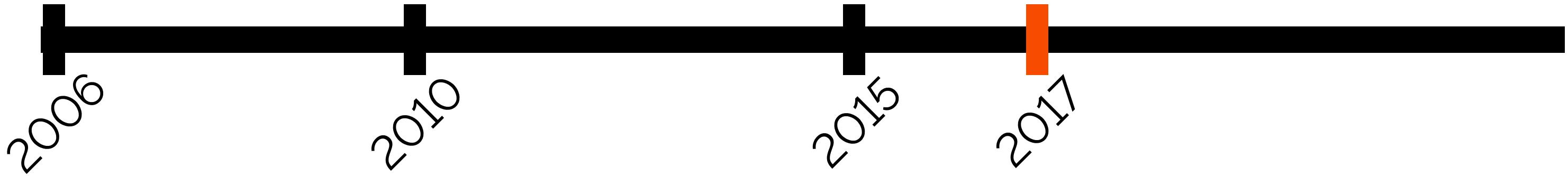
Rust 1.0 released





Rust 1.0 released 

Strong backwards-compatibility promise

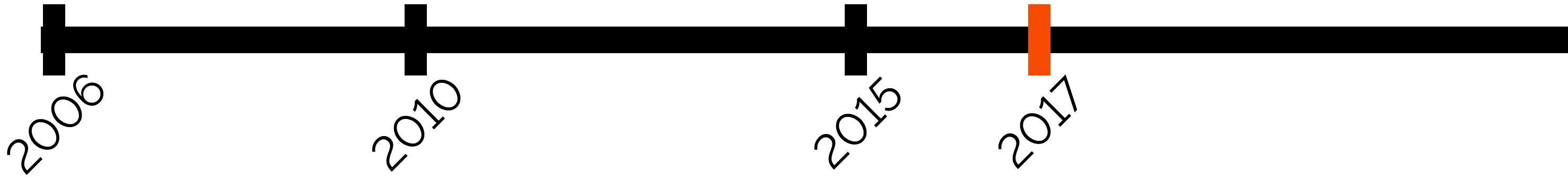


# Inside a super fast CSS engine: Quantum CSS (aka Stylo)



By [Lin Clark](#)

Posted on August 22, 2017 in [Code Cartoons](#), [CSS](#), and [Featured Article](#)



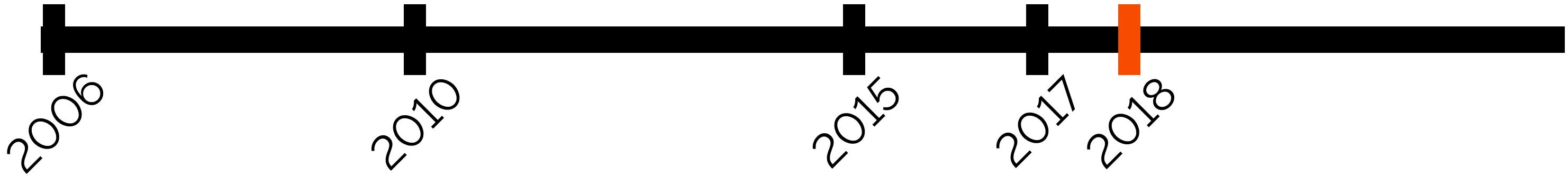
Posted by u/[deleted] 7 years ago

283

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



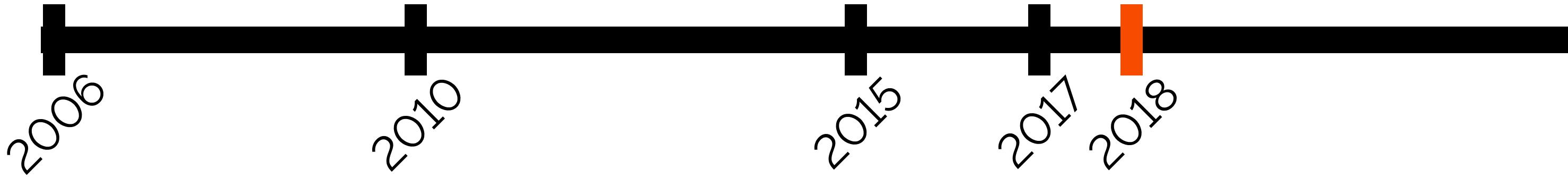
[code.visualstudio.com/update...](http://code.visualstudio.com/update...) ↗



# Announcing Rust 1.31 and Rust 2018

---

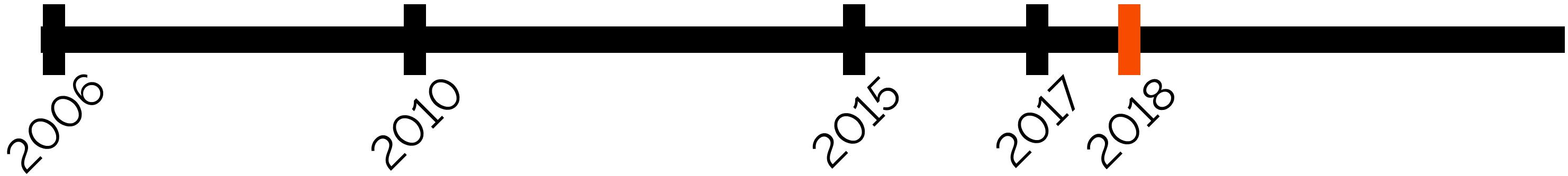
Dec. 6, 2018 · The Rust Core Team



## 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

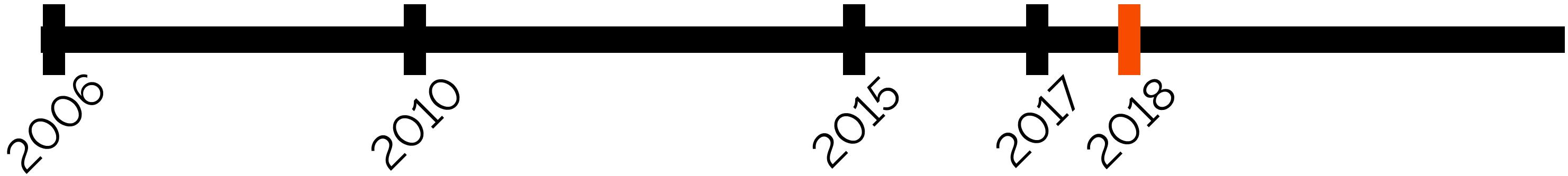


# Serverless Rust with Cloudflare Workers

10/16/2018

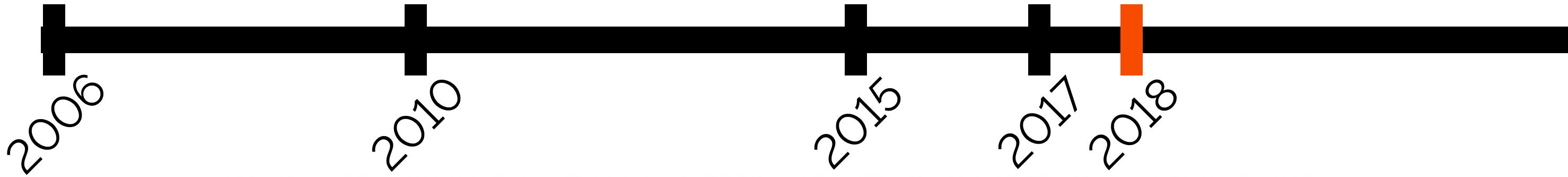


Steven Pack

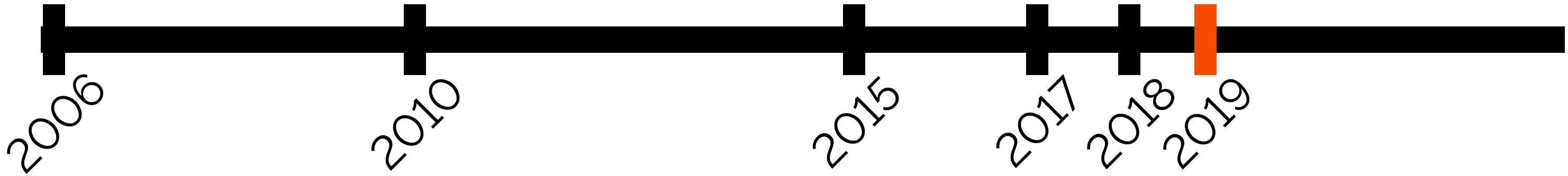


How Mozilla's new language dramatically improved our server-side performance

*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

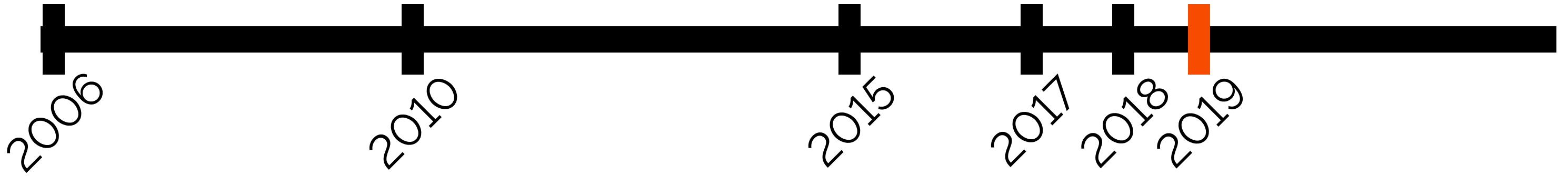


Rust Case Study:

# Community makes Rust an easy choice for npm

---

The npm Registry uses Rust for its CPU-bound bottlenecks



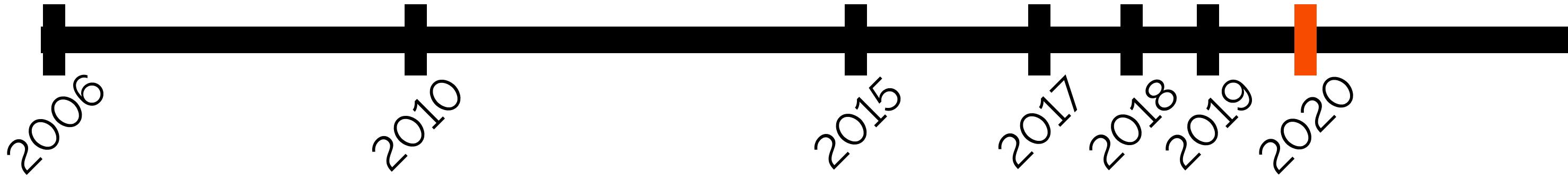
## Rust Case Study:

“

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 ”

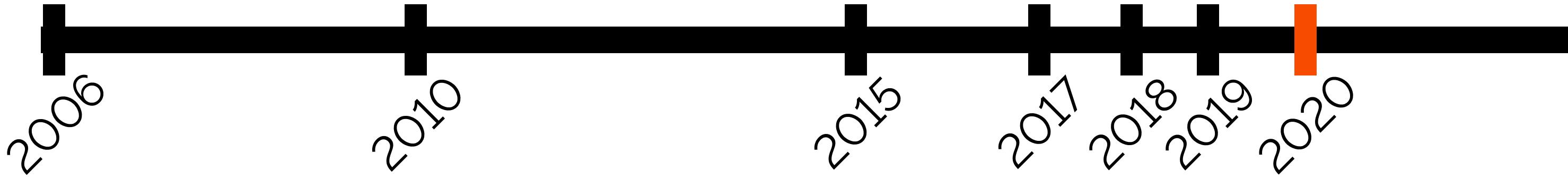
THE NPM REGISTRY USES RUST FOR ITS CPU-BOUND BOTTLENECKS



# Rewriting the heart of our sync engine

// By Sujay Jayakar • Mar 09, 2020

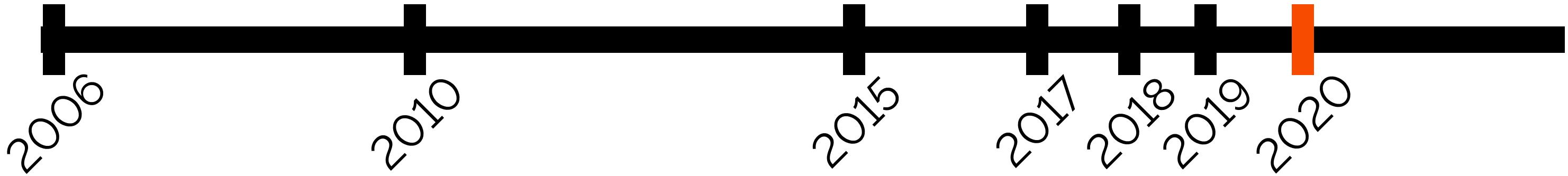




“

**..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 ,”



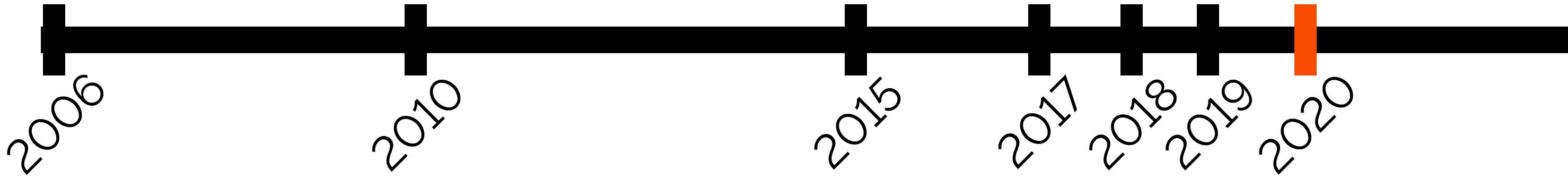
ENGINEERING & DEVELOPERS

# 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

“

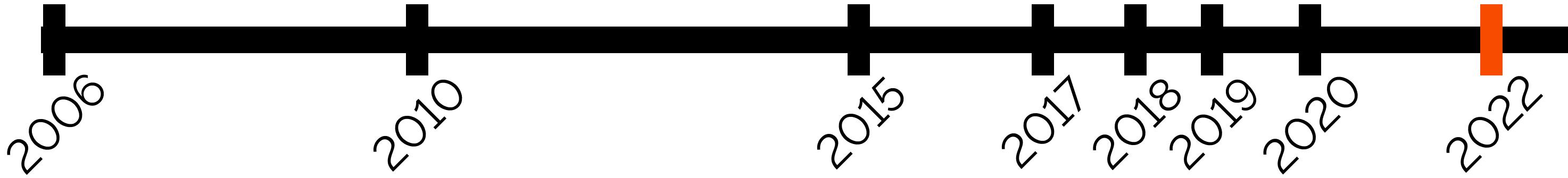
Even with just basic optimization, Rust was able to outperform the hyper hand-tuned Go version.

**...we were able to beat Go on every single performance metric.**

Discord ,”

February 4, 2020

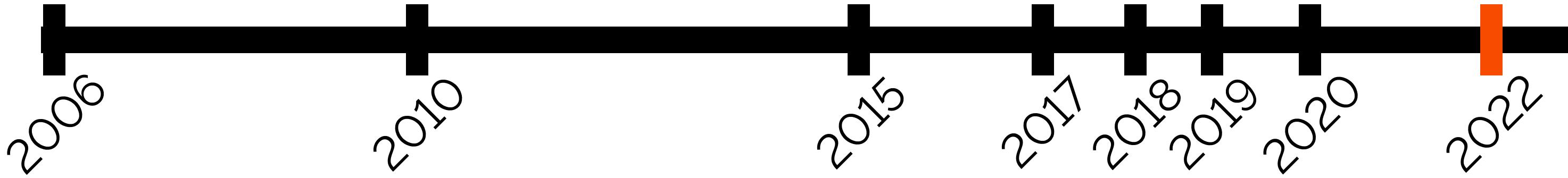
encoding pipeline for Go Live and on the server side for [Elixir NIFs](#).



## 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.



Amazon POV

Posted

Amazon  
HTTP/3

that con-

HTTP/2

enhanced security.

“

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

AWS

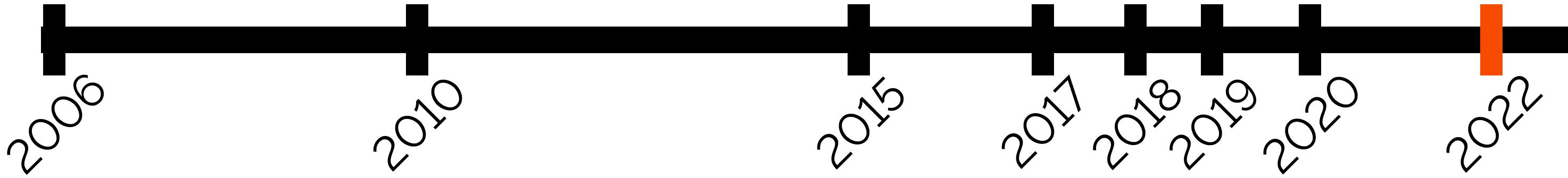
”

connections.

protocol

TLS, and

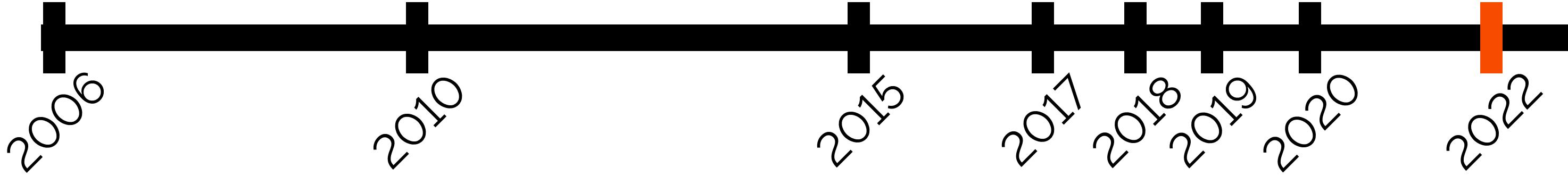
names and



# 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](#)



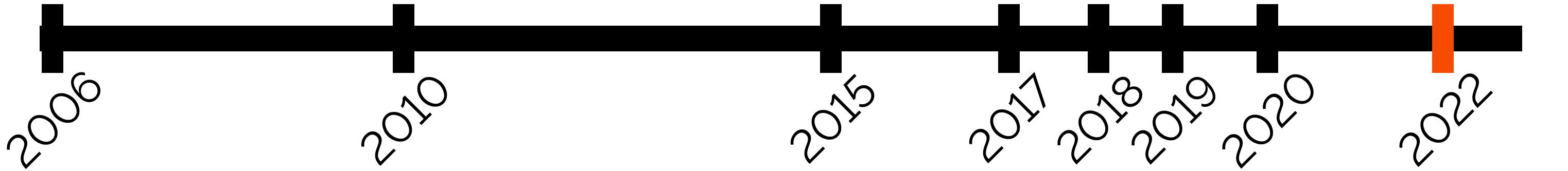
Meta “...we're committing to Rust long-term; and welcome early adopters.”

Wednesday  
programming

Jul 28th, 2022

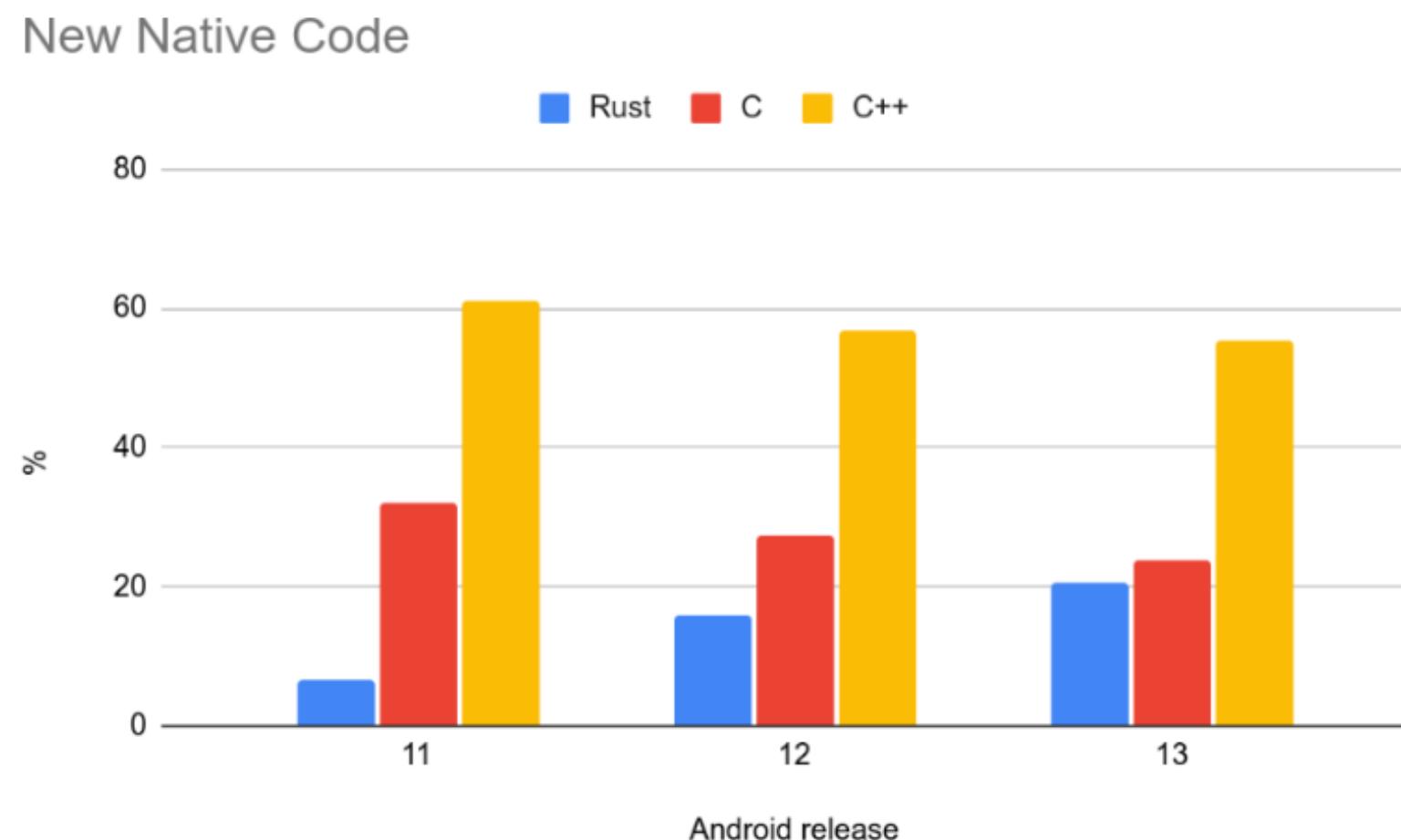
Meta ,”

server side

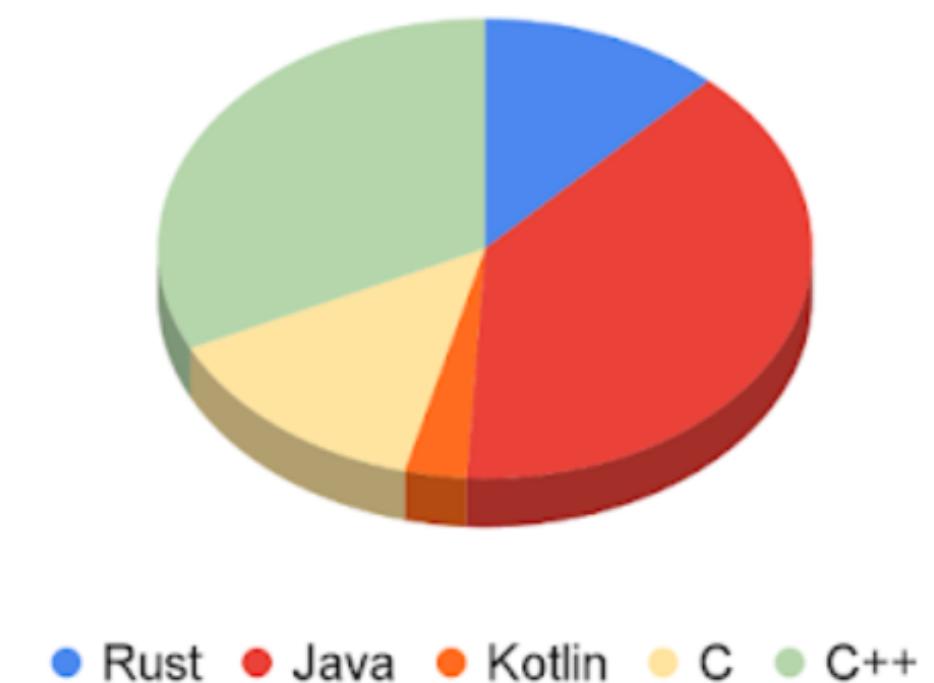


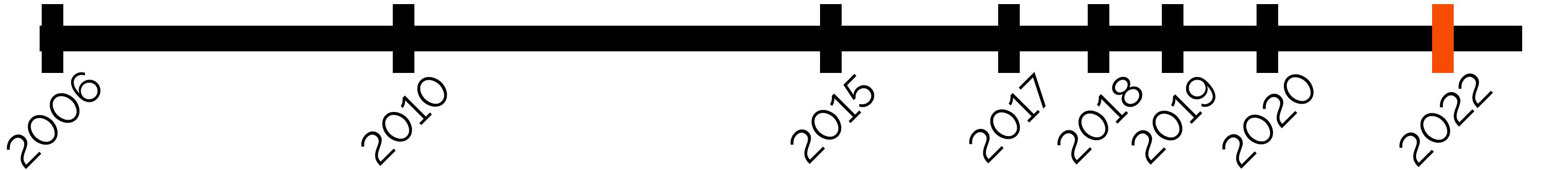
## Memory Safe Languages in Android 13

December 1, 2022



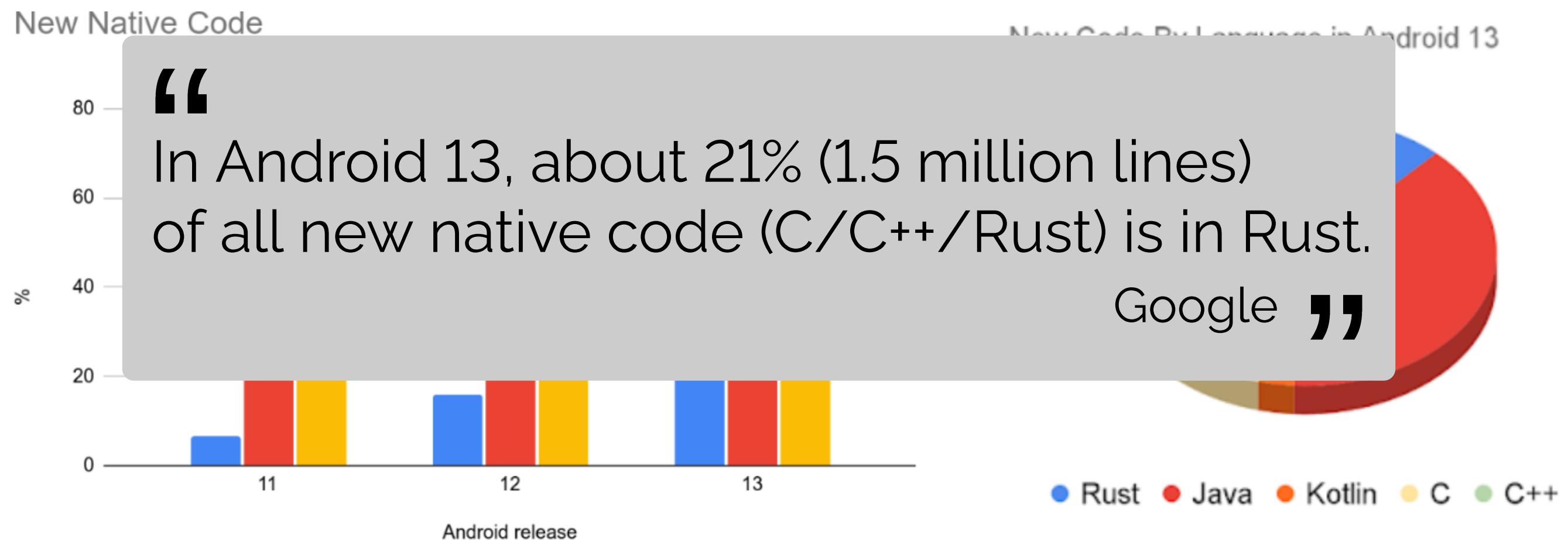
New Code By Language in Android 13

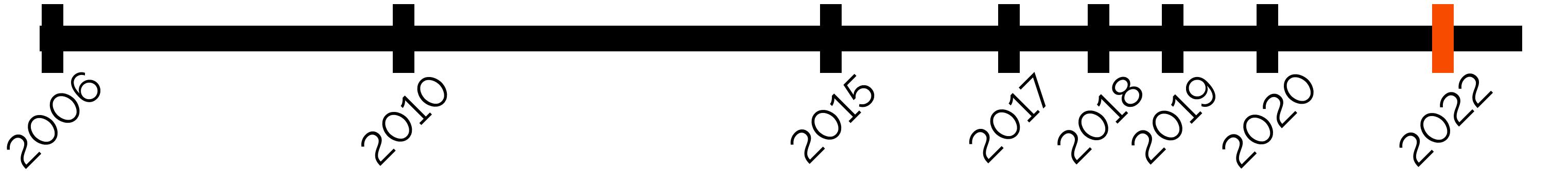




## Memory Safe Languages in Android 13

December 1, 2022

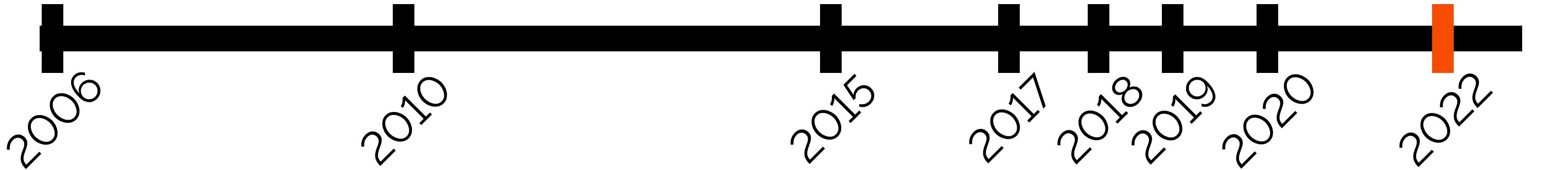




# Rust Programming Language To Land in Linux Kernel 6.1

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 Kernel

By Ian Evenden published

Linux will support

from version 6.1.

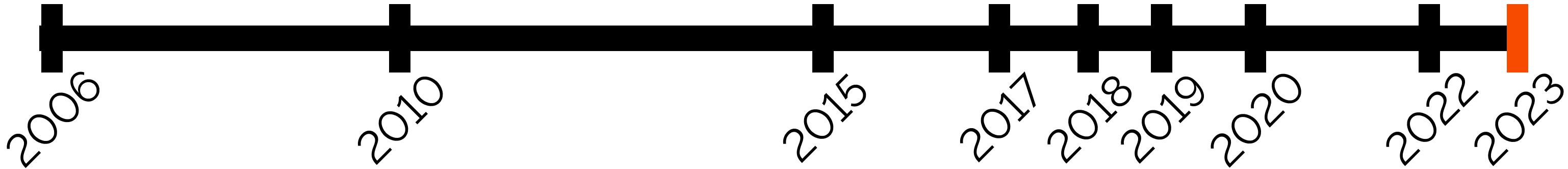
“

...on the whole, I don't hate it.

Linus Torvalds

”

change in its kernel



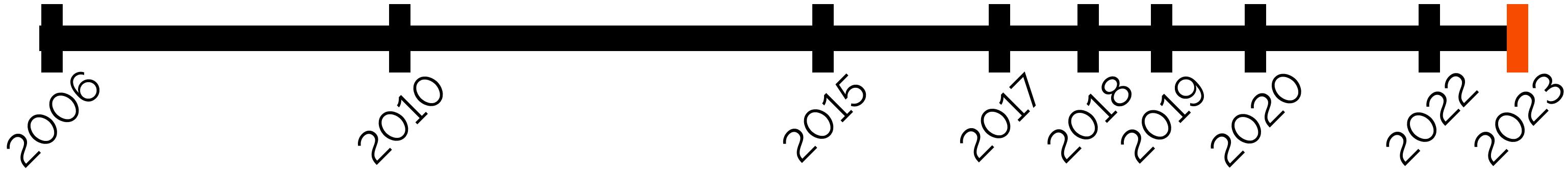
# GitHub built a new search engine for code 'from scratch' in Rust

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



 **Mark Russinovich**  
@markrussinovich

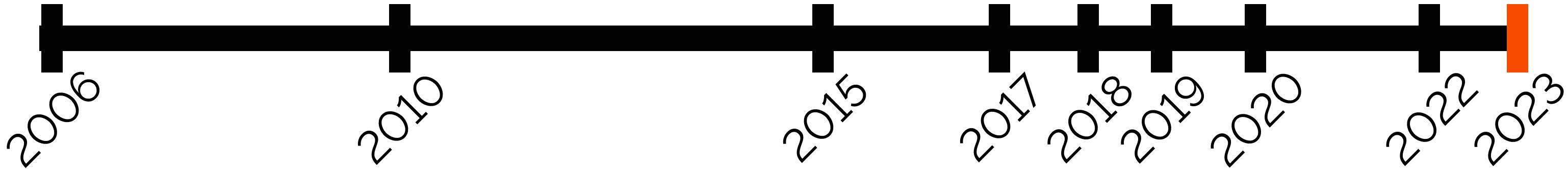
If you're on the Win11 Insider ring, you're getting the first taste of Rust in the Windows kernel!

```
C:\Windows\System32>dir win32k*
Volume in drive C has no label.
Volume Serial Number is E60B-9A9E

Directory of C:\Windows\System32

04/15/2023  09:50 PM           708,608 win32k.sys
04/15/2023  09:49 PM          3,424,256 win32kbase.sys
04/15/2023  09:49 PM          110,592 win32kbase_rs.sys
04/15/2023  09:50 PM          4,194,304 win32kfull.svs
04/15/2023  09:49 PM          40,960 win32kfull_rs.sys
04/15/2023  09:49 PM          69,632 win32krnl.sys
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
```

\_rs = Rust!



Mark Russinovich  
@markrussinovich

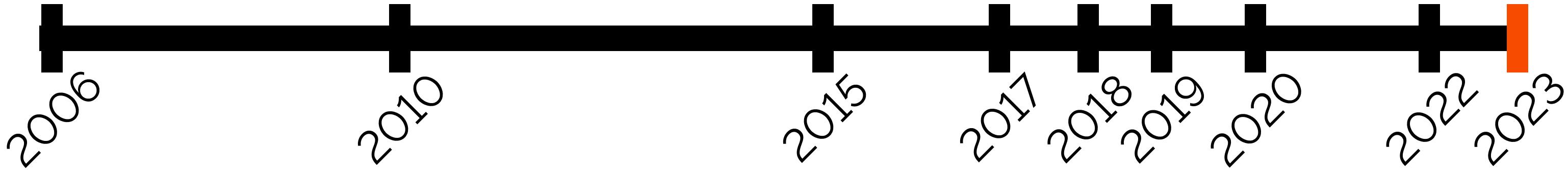
...

“

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.

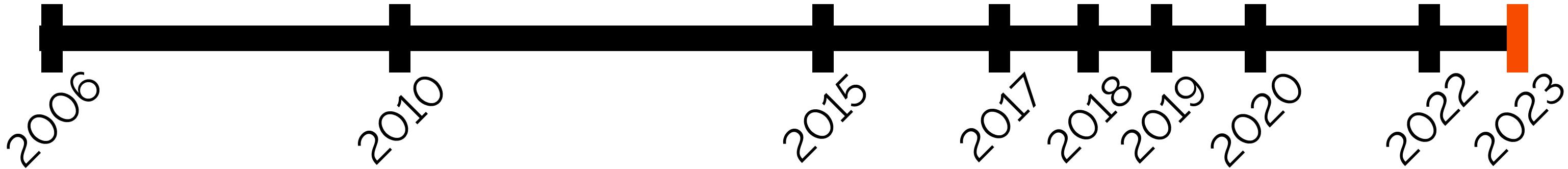
Mark Russinovich ,”

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

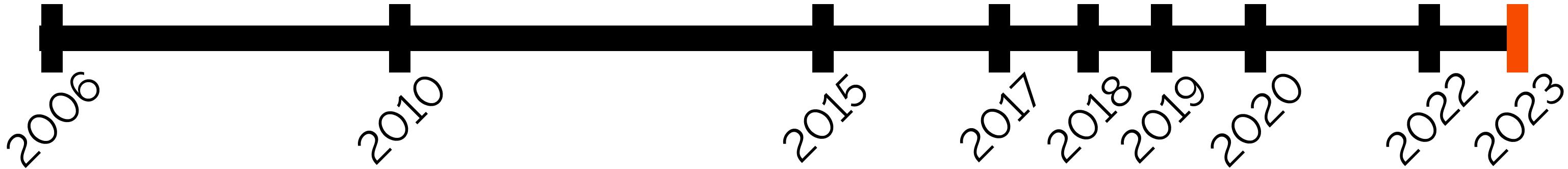


Bill Ticehurst

---

September 19th, 2023 | 6 | 7

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

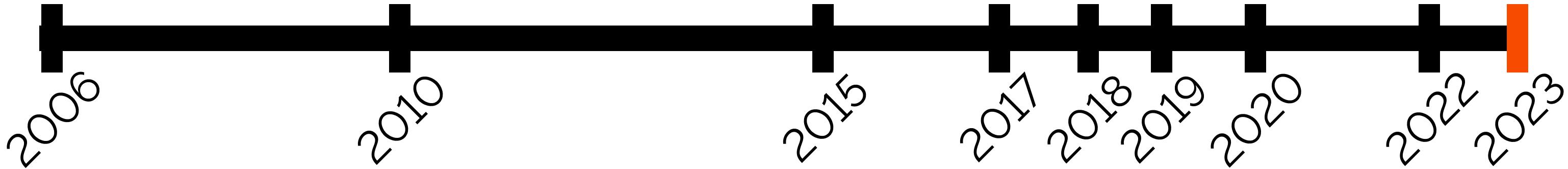


# Ferrocene Safety-Critical Rust Compiler Code Published

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

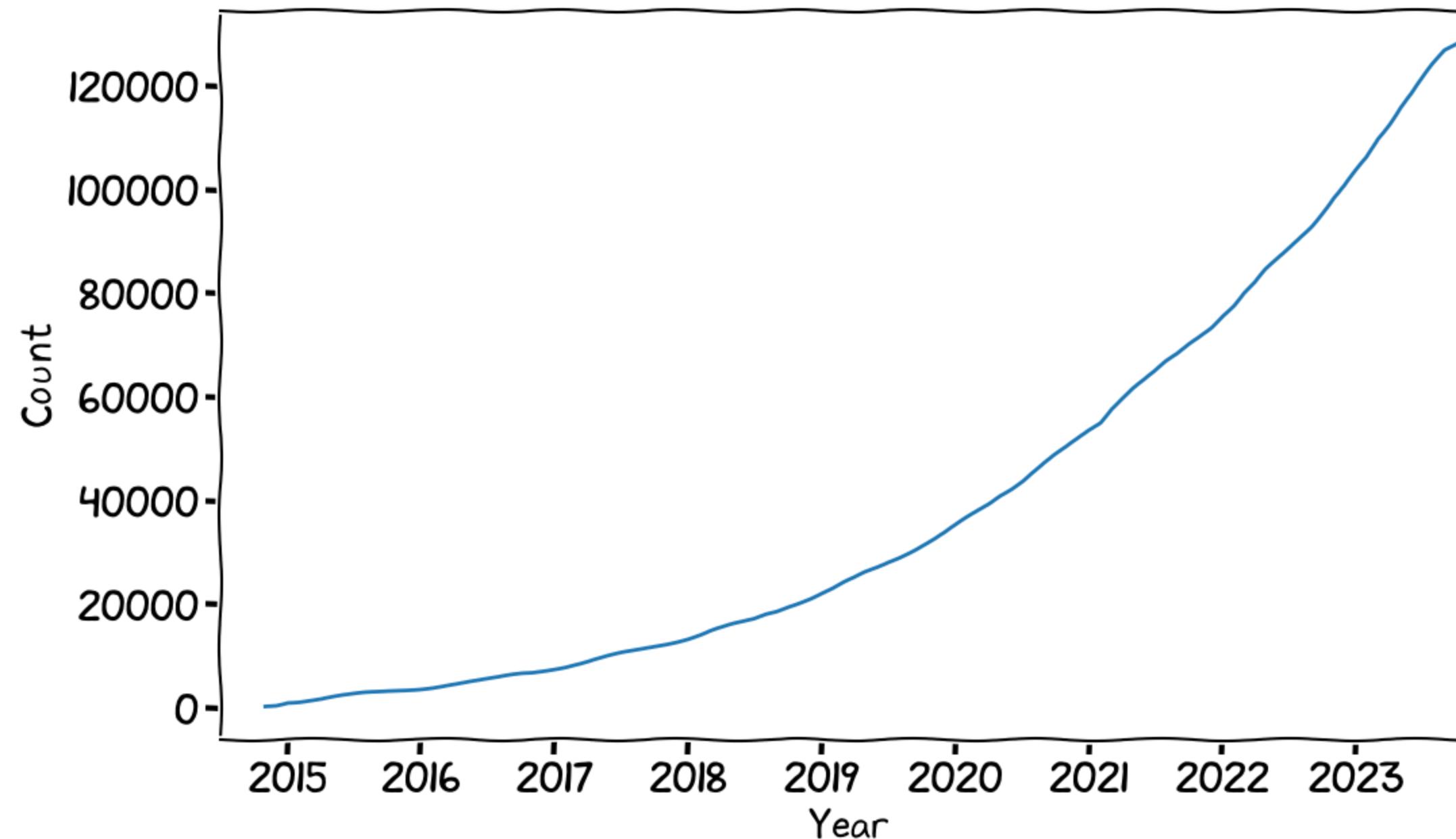
833

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

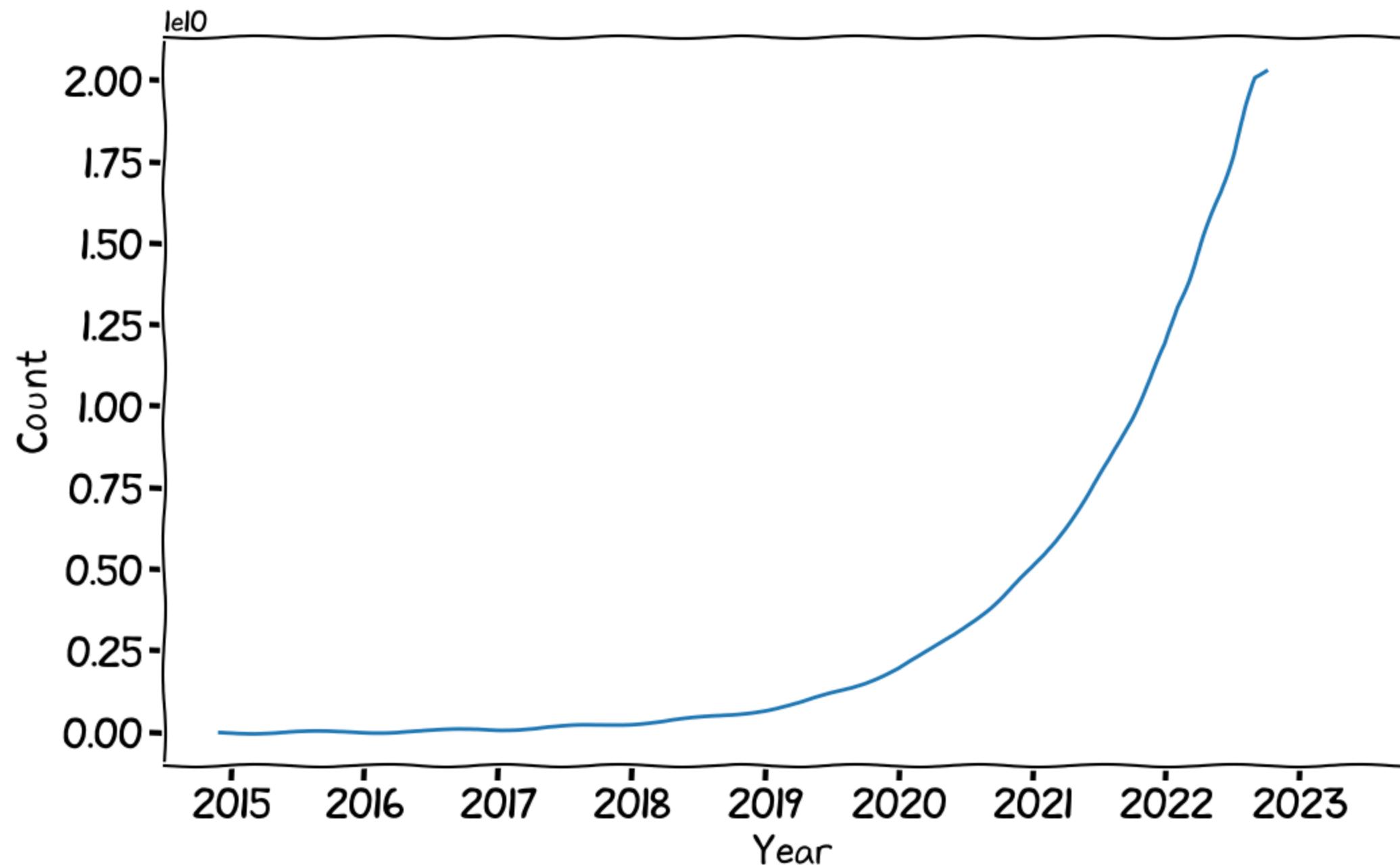


[survey.stackoverflow.co/2023/](https://survey.stackoverflow.co/2023/) ↗

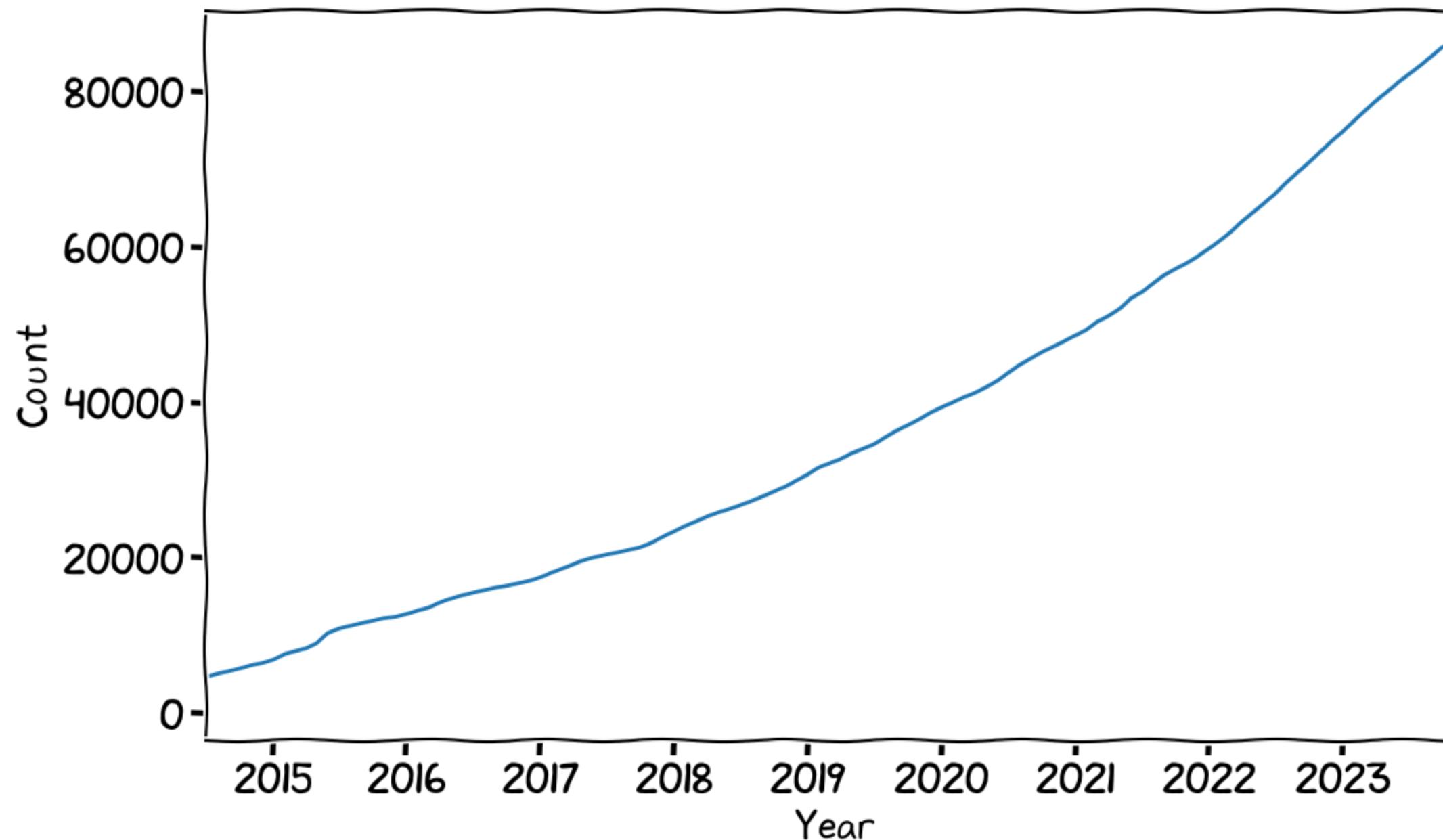
# Crate count (120k+)



# Crate downloads (20B+)

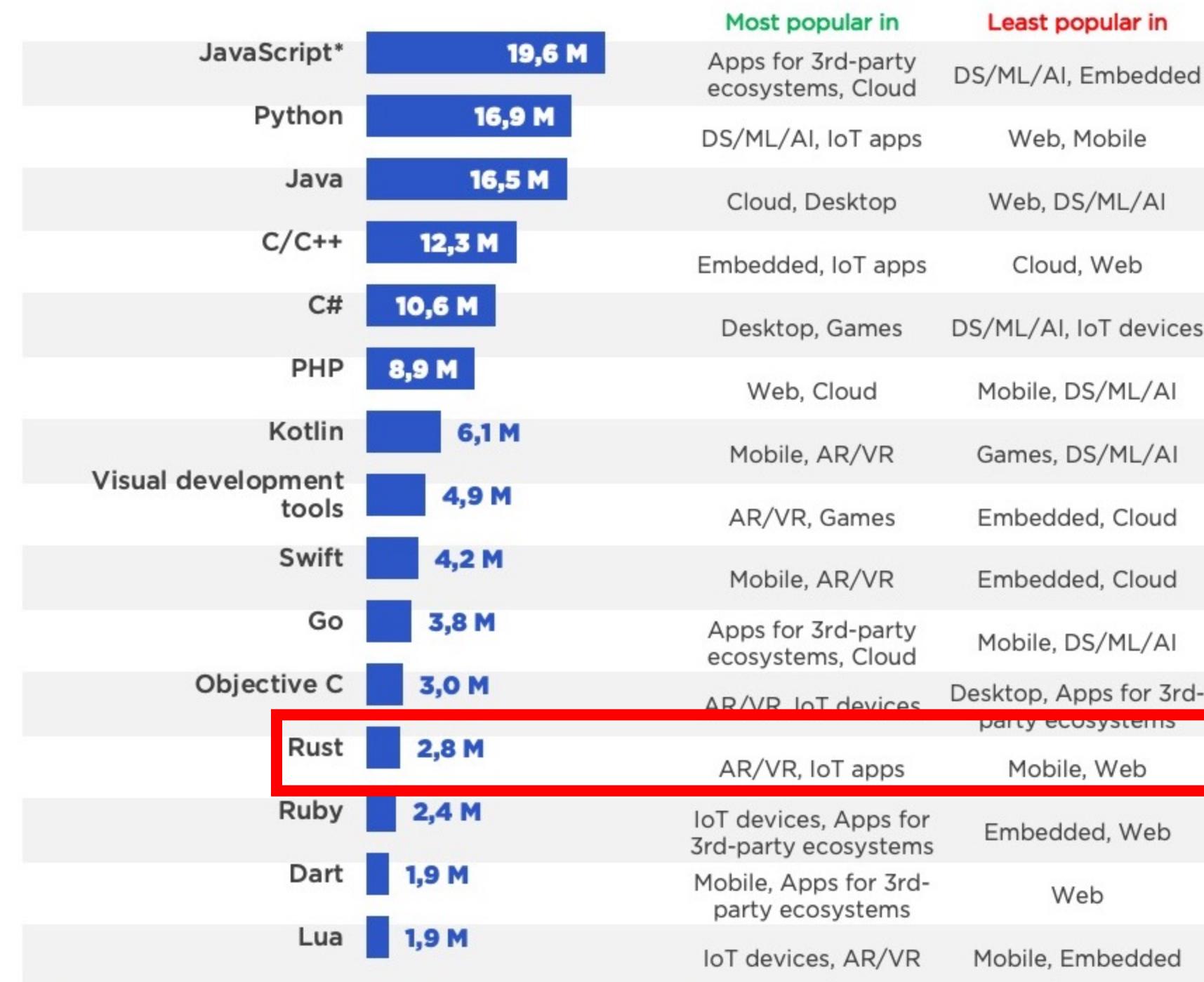


# GitHub stars (80k+)



# Size of programming language communities in Q3 2022

Active software developers, globally, in millions





# 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.



# (Memory) Safety



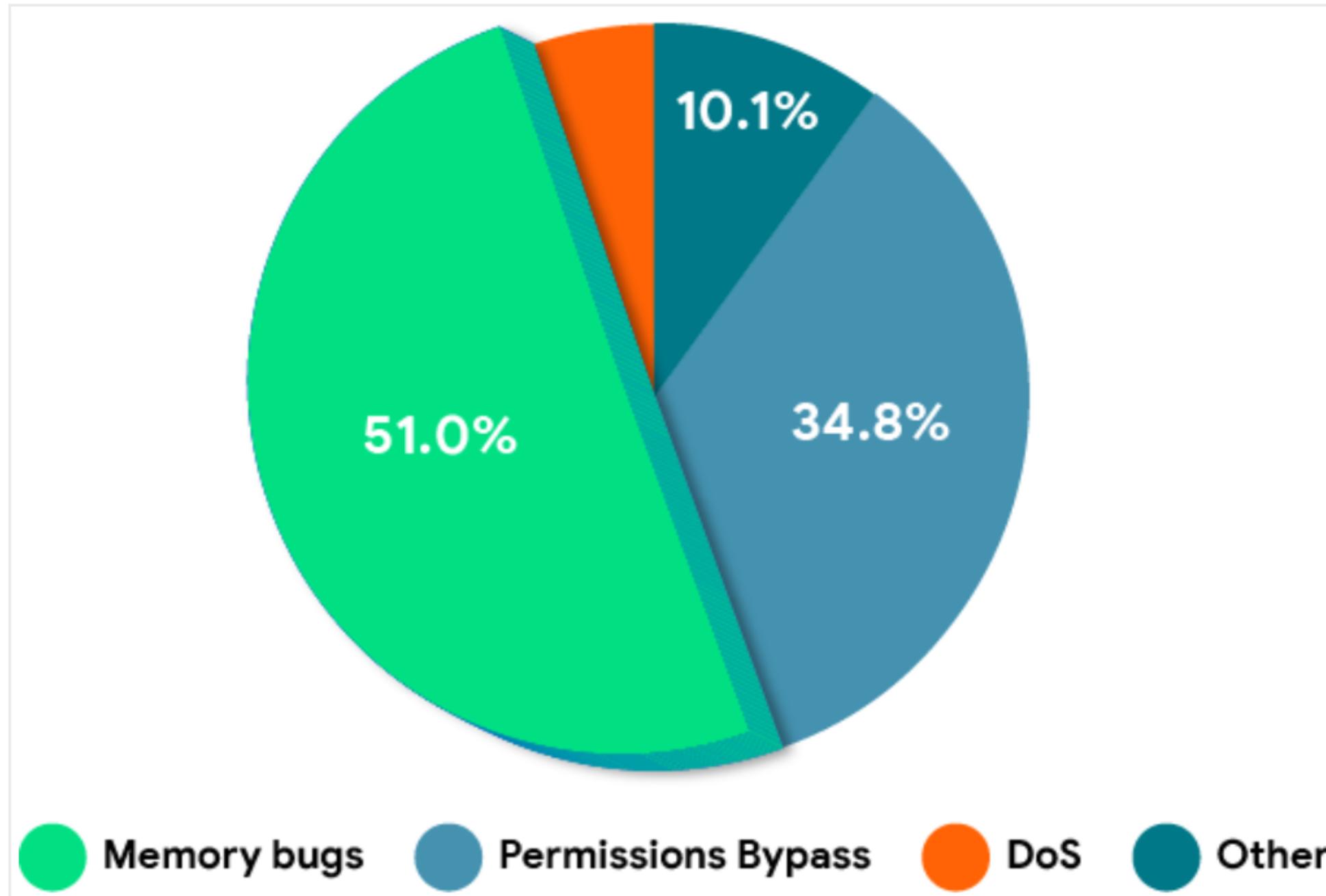
# **Microsoft: 70 percent of all security bugs are memory safety issues**

**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.**



**Figure 2:** Memory safety bugs contribution to Android vulnerabilities



## **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.**



Caused by undefined behaviour (UB)



## Caused by undefined behaviour (UB)

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



## Caused by undefined behaviour (UB)

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

Rust: avoid UB at all costs



Memory errors arise when  
aliasing is combined with mutability



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



## Aliasing ✓

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

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



## Mutability ✓

```
std::vector<int> vec = { 1, 2, 3 };  
vec.push_back(4);
```



## 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?



imgflip.com

JAKE-CLARK.TUMBLR



## 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



# A Safer High Performance AV1 Decoder

Josh Aas

Mar 9, 2023



# **Bringing Memory Safety to sudo and su**

Josh Aas

Apr 26, 2023



# A Memory Safe Implementation of the Network Time Protocol

Folkert de Vries

Oct 11, 2022



# **Announcing Hickory DNS**

Benjamin Fry  
Oct 5, 2023



Rustls is a modern TLS library written in Rust.



# 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 ,”



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



```
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
```



## Correctness

Design APIs 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 implement 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<u32>
```



## 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 opt_value {  
    Some(value) => ...,  
    None => println!("Value is missing")  
}
```



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")  
}
```



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")  
}
```

But tries to make it ergonomic

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



## 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





## (LLVM) optimizations

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



## (LLVM) optimizations

```
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  
vpadddd ymm0, ymm4, ymm0  
vpadddd 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()  
}
```



# Optimizing 700 CPUs Away With Rust



Alan Ning · Follow

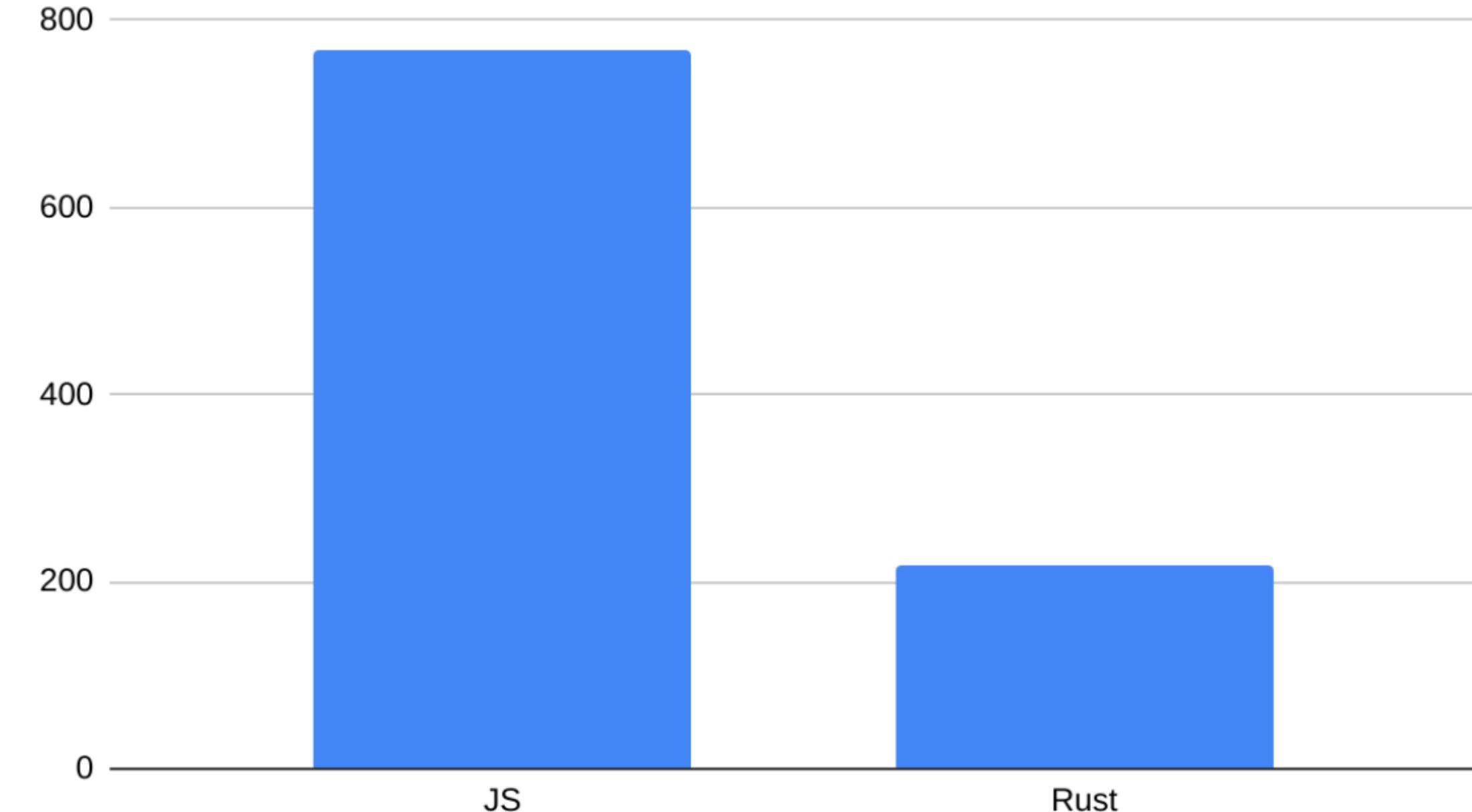
 Published in Tenable TechBlog · 3 min read · May 6, 2021



CPU

O

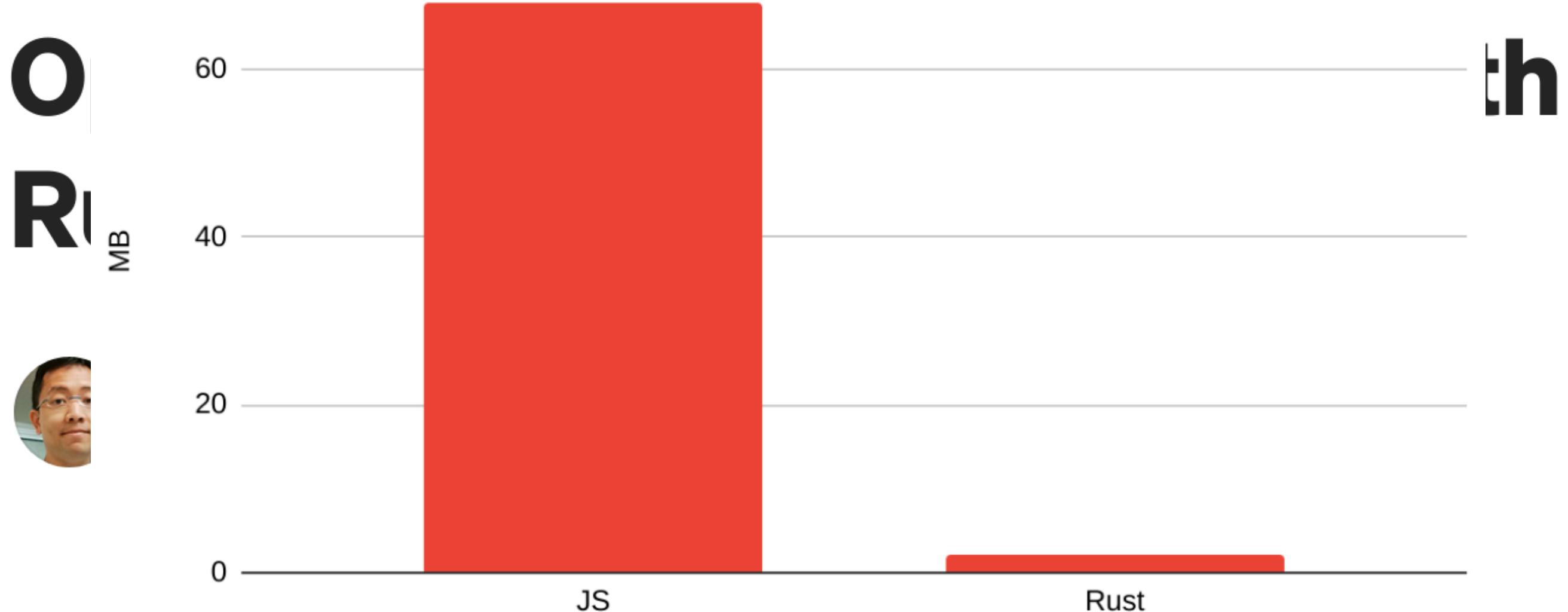
R



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



## Memory



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 ”

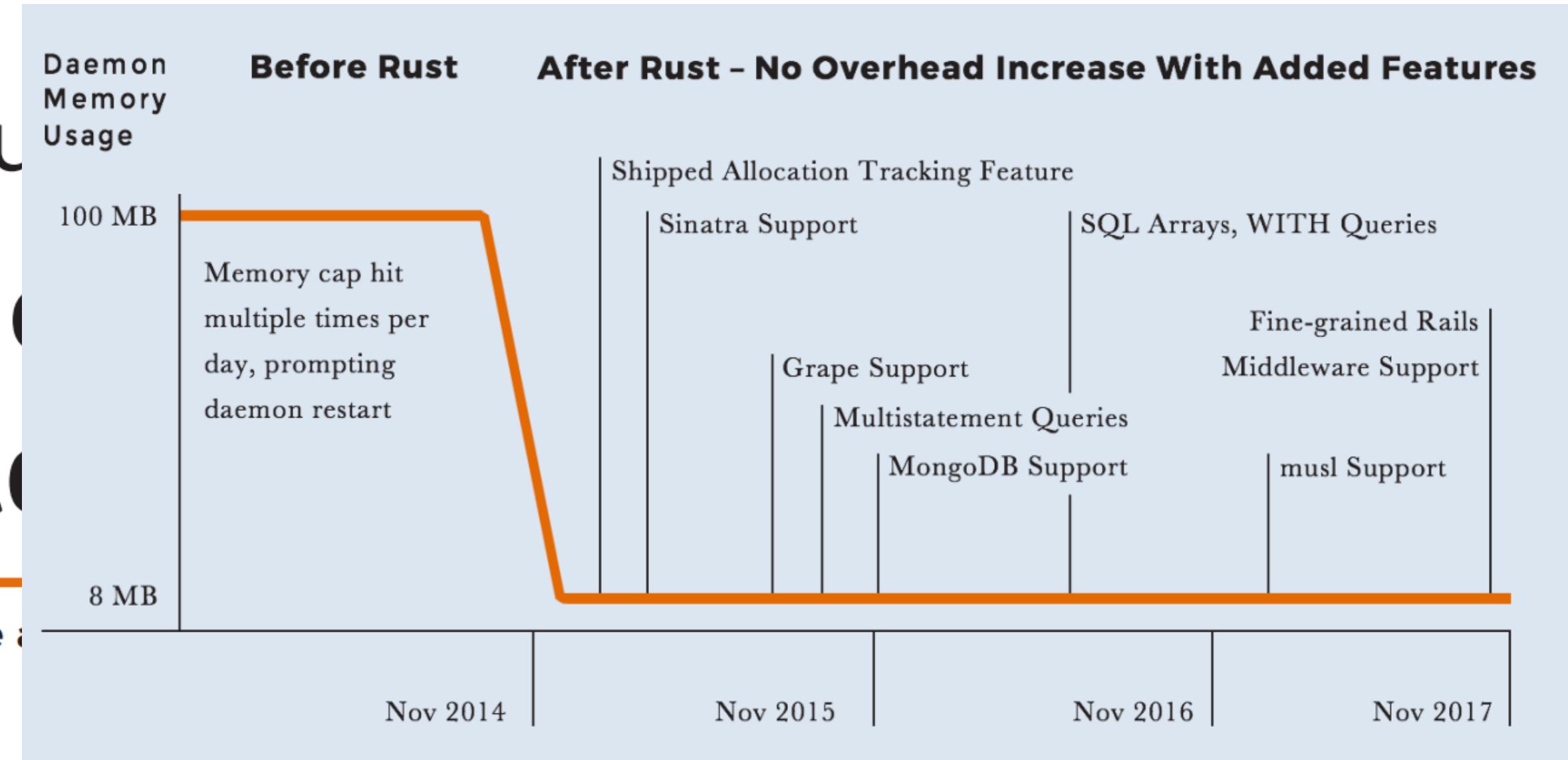


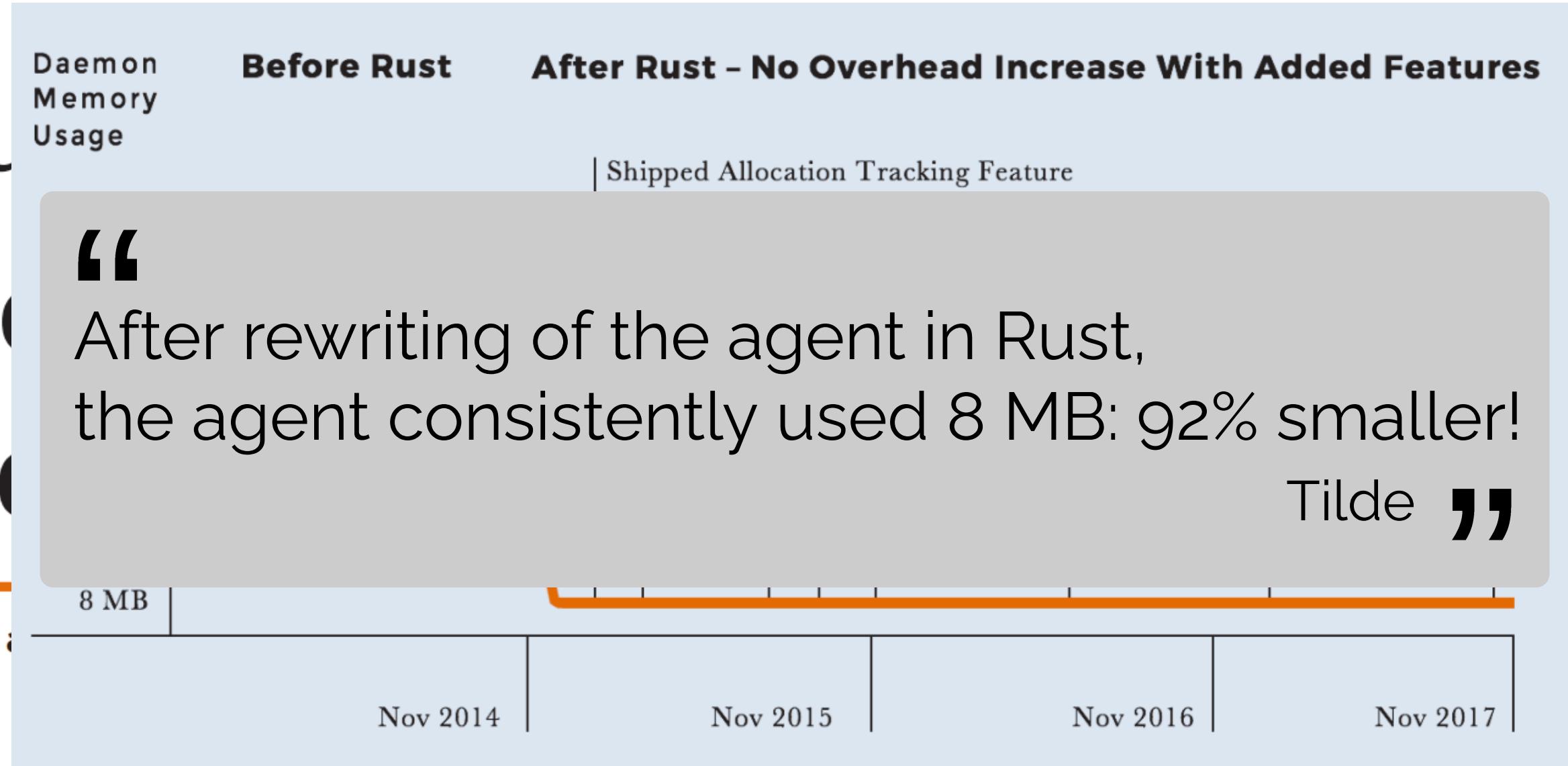
Rust Case Study:

# **How Rust is Tilde's Competitive Advantage**

---

**The analytics startup innovates safely with the help of Rust**







ENGINEERING & DEVELOPERS

# WHY DISCORD IS SWITCHING FROM GO TO RUST

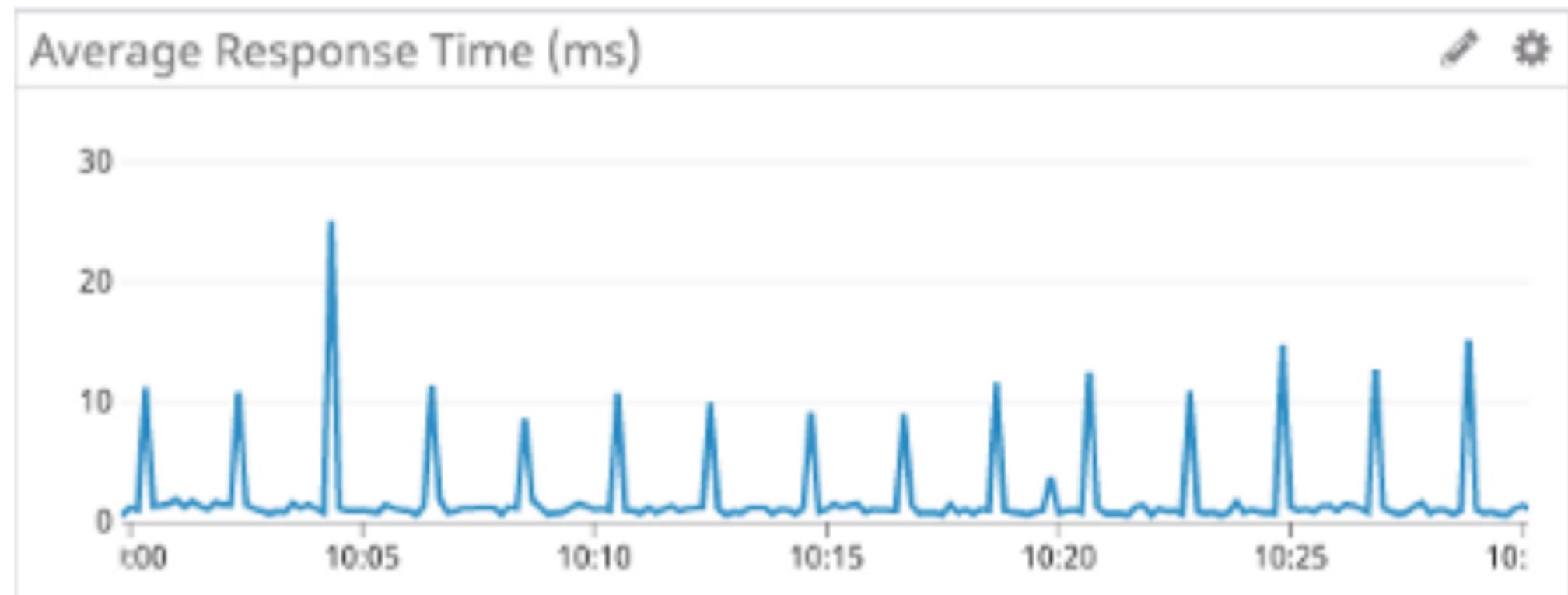


Jesse Howarth  
February 4, 2020

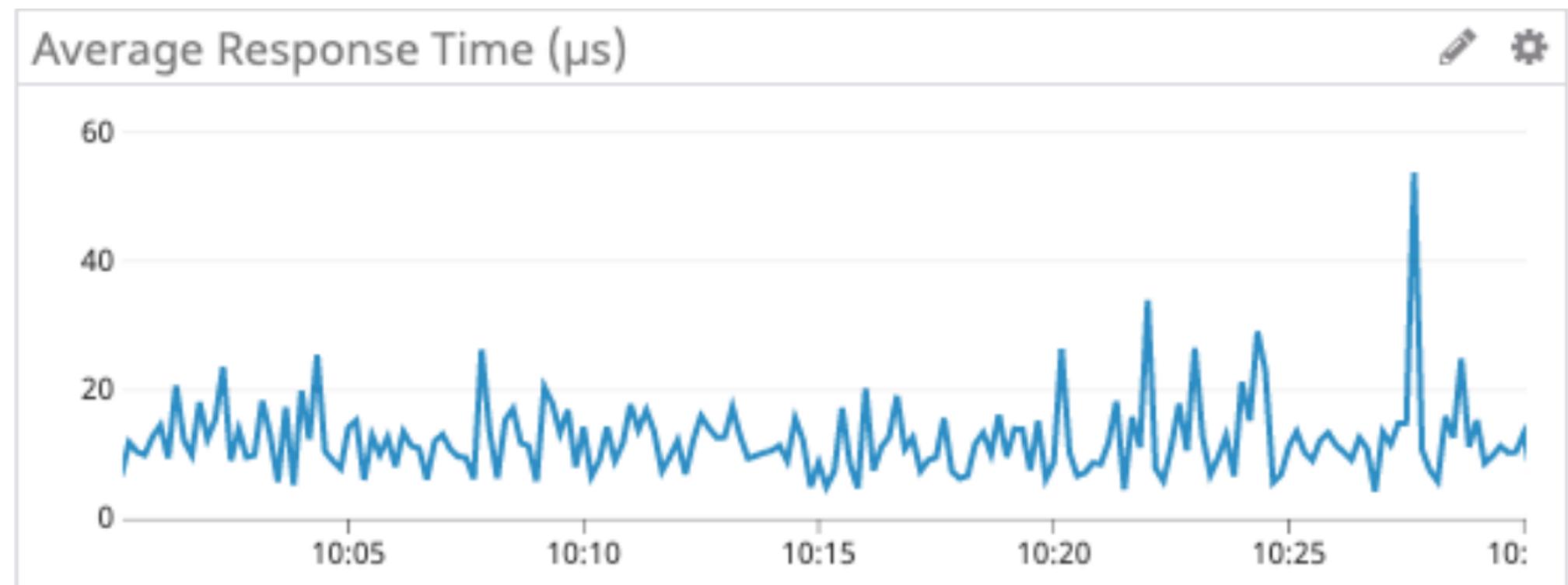
Rust is becoming a first class language in a variety of domains. At Discord, we've seen success with Rust on the client side and server side. For example, we use it on the client side for our video encoding pipeline for Go Live and on the server side for [Elixir NIFs](#).



Go



Rust





Go

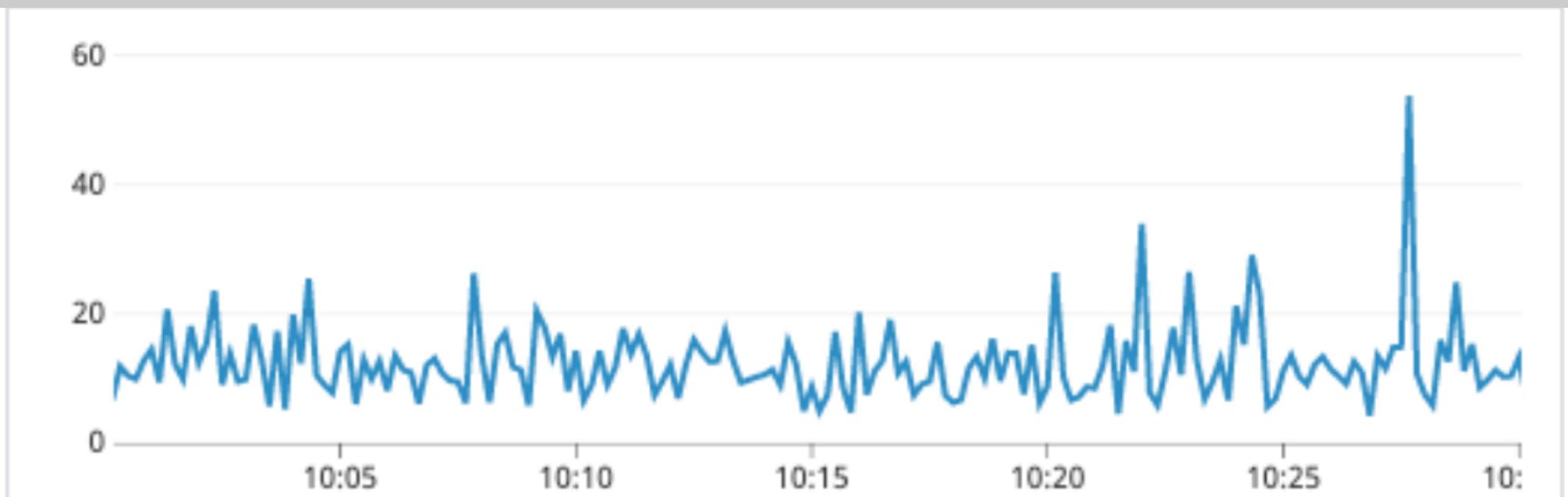


“

Notice the average time is now measured in microseconds.

Discord ,”

Rust





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](#)



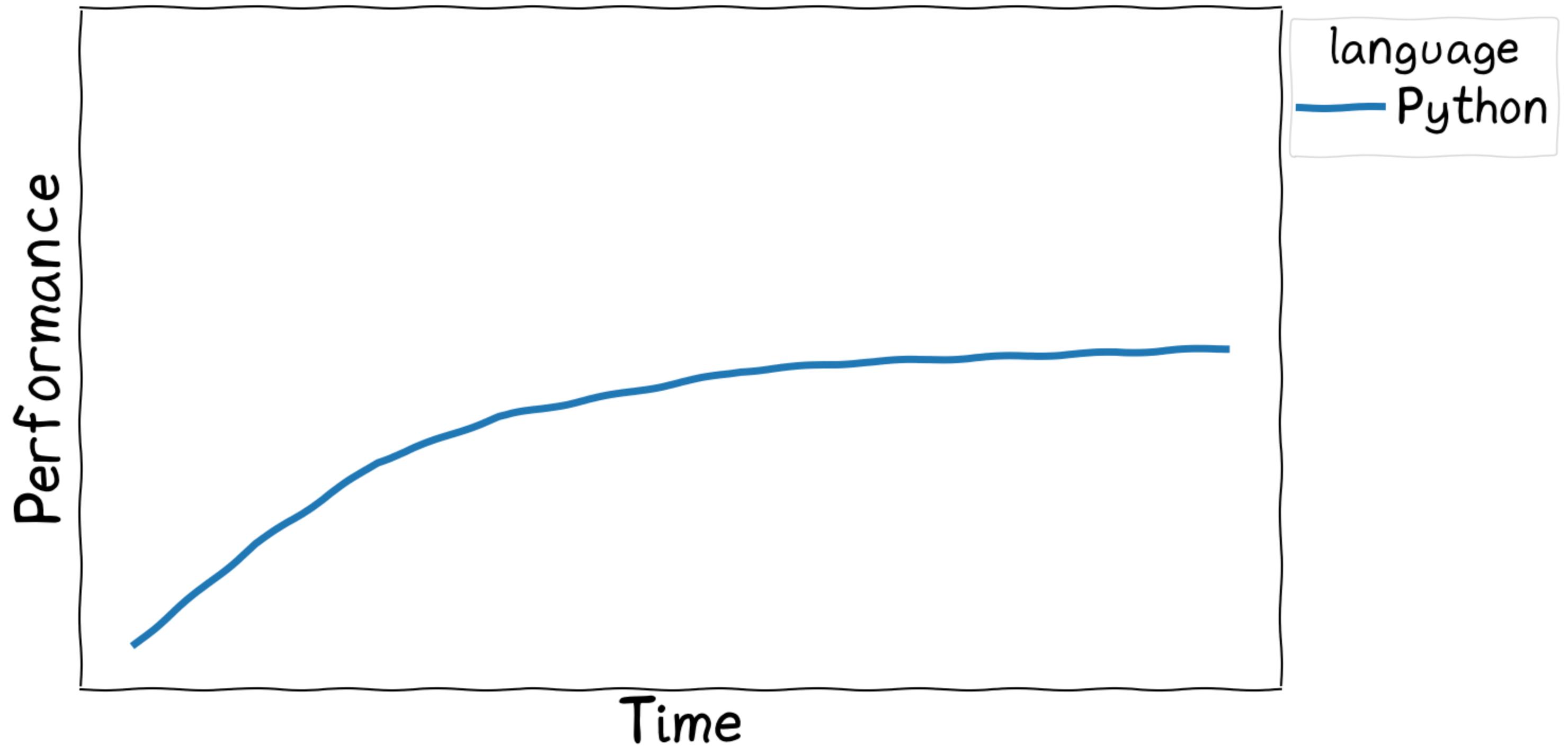
# My Node.js is a bit Rusty

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

September 4, 2023.

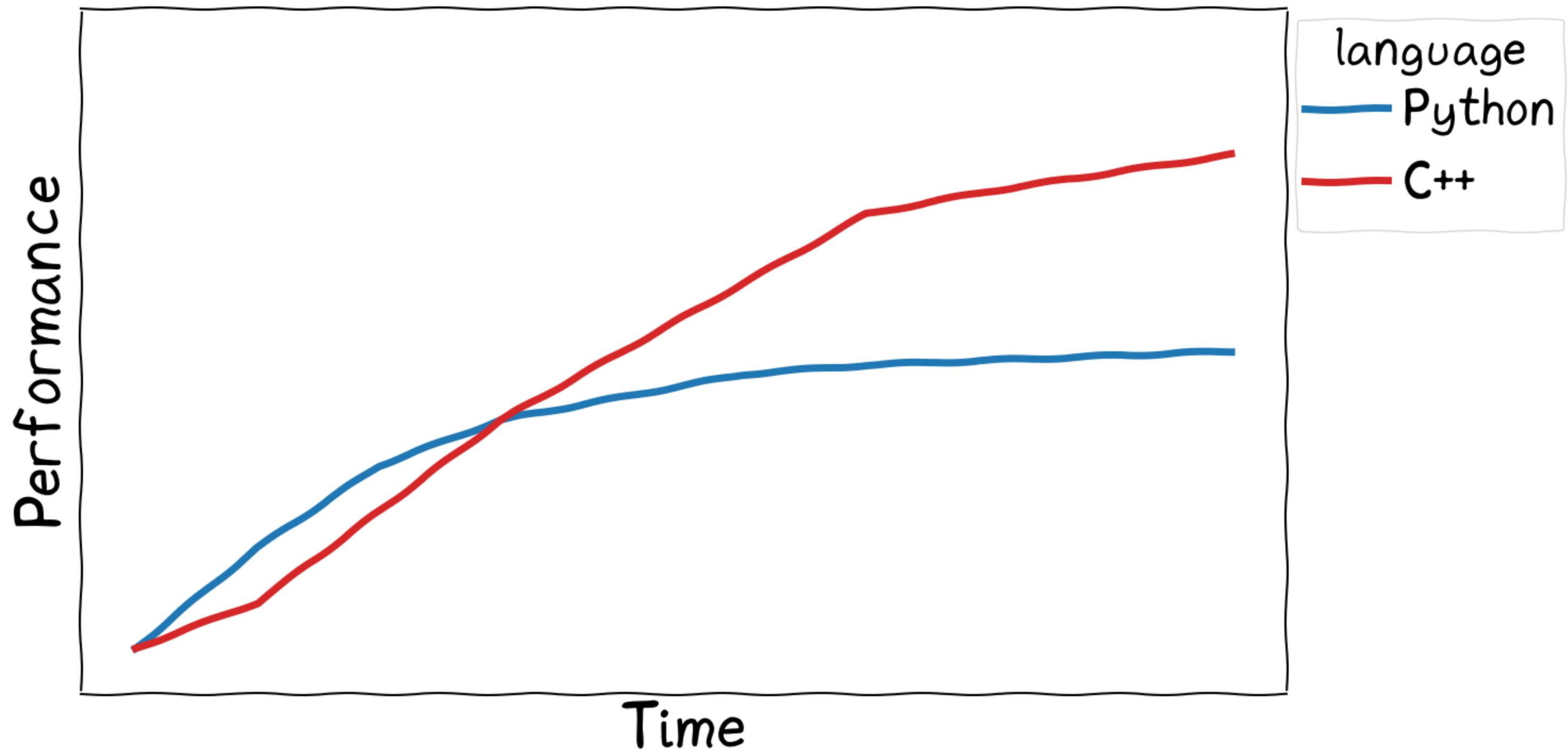


# Time to performance



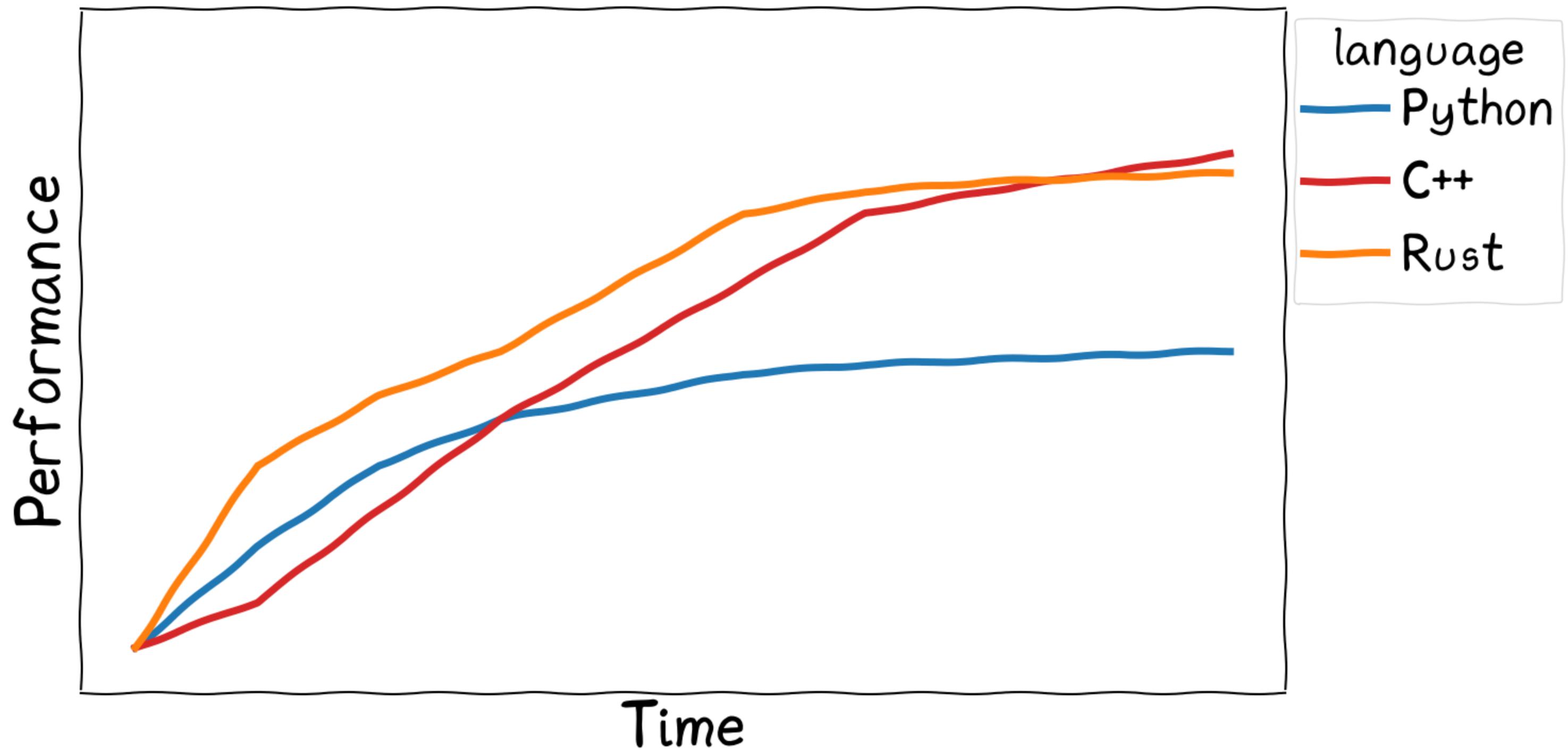


# Time to performance





# Time to performance





# 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
```

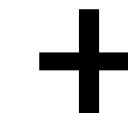


Run

```
$ cargo run
```



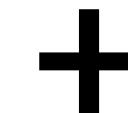
```
#[test]
fn test_add() {
    assert_eq!(add(1, 2), 3);
}
```



Test  
\$ cargo test



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



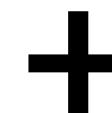
Test

```
$ cargo test
```

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



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

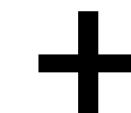


Benchmark

\$ cargo bench



```
#[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)
```

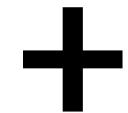


## Format code

```
$ cargo fmt
```



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



Build documentation  
\$ cargo doc



[–] **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.

`Strings` 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.



## Library (crate) documentation

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



## Custom commands

```
$ cargo semver-checks
```



```
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\\_var](https://github.com/obi1kenobi/cargo-semver-check/tree/v0.4.1/src/queries/enum_var)

#### 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"
```



## Cargo.toml

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

[dependencies]
json = "1.0"
```

## main.rs

```
use json::parse;

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





## 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 |
34 |     }
   | - `value` dropped here while still borrowed
35 |     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(|| {
50     |-----^
51     | required by a bound introduced by this call
52     |         *val += 1
53   });
54   ^ `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?



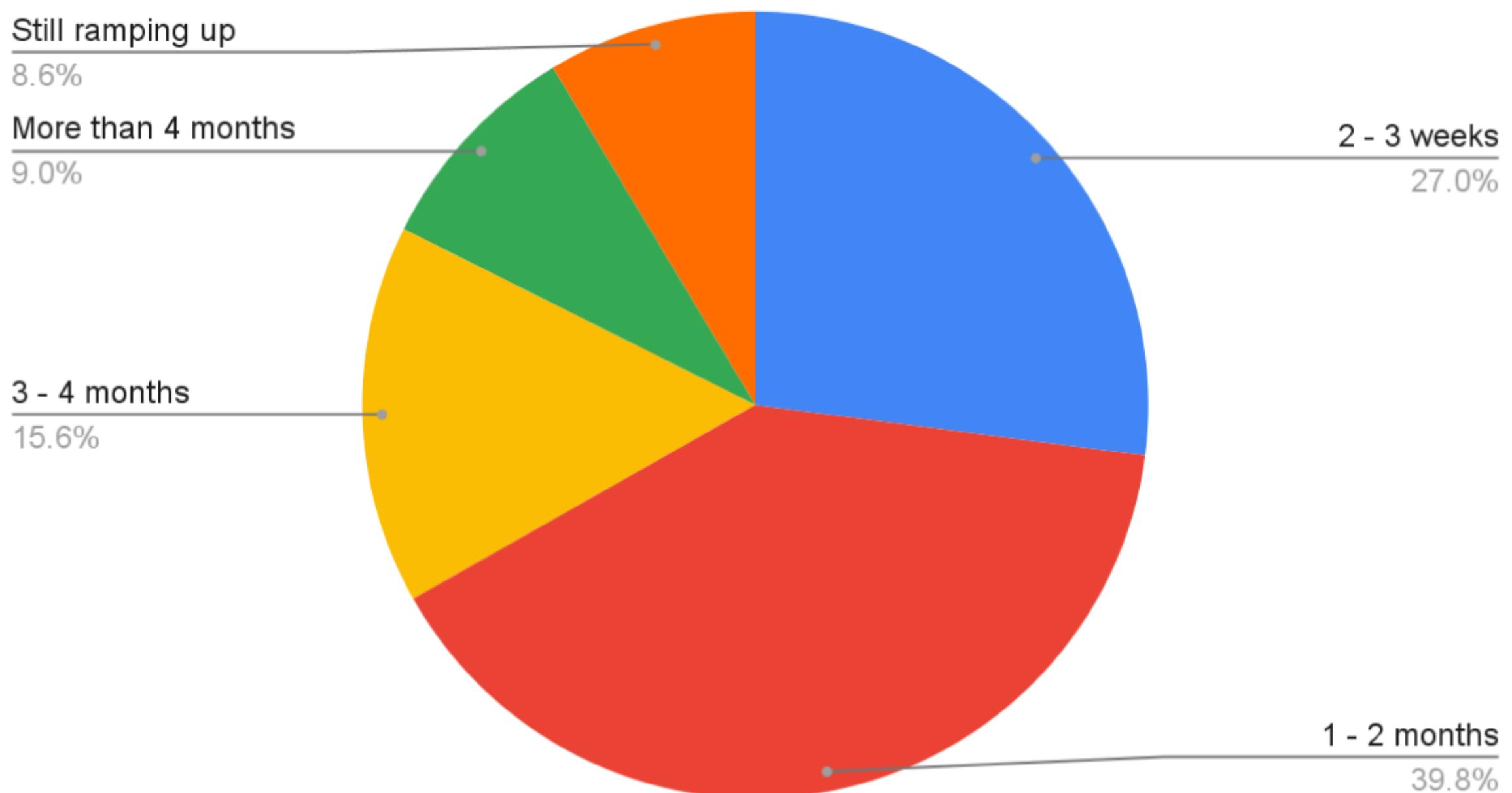
# Why not Rust?

- Learning curve



# Why not Rust?

Time until confident writing Rust





# Why not Rust?

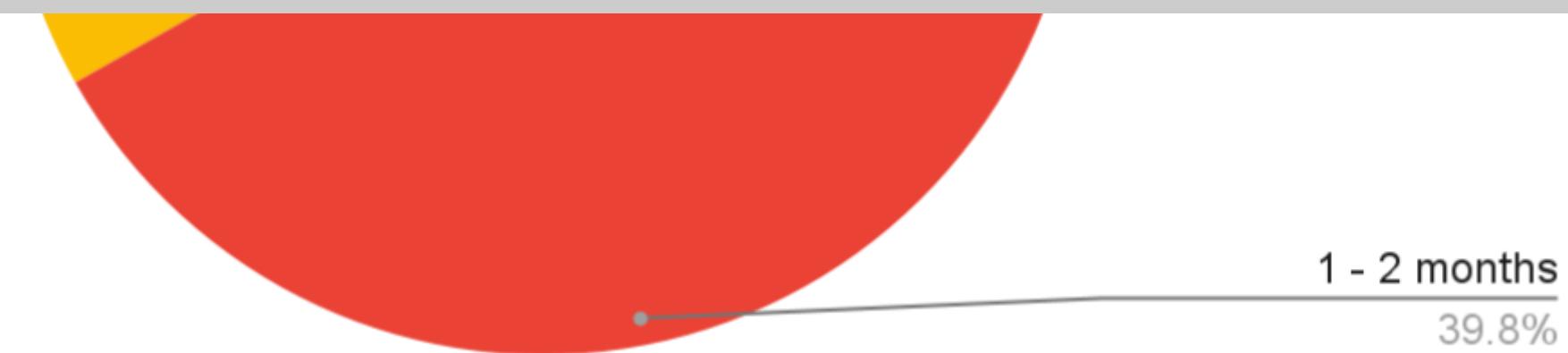
Time until confident writing Rust



“

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

```
use napi_derive::napi;

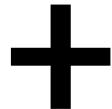
#[napi]
fn fibonacci(n: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
            fibonacci(n - 2),
    }
}
```



# Accelerate Node.js

```
use napi_derive::napi;

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



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

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



# Accelerate Python

```
use pyo3::pyfunction;

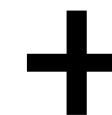
#[pyfunction]
fn fibonacci(a: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
            fibonacci(n - 2),
    }
}
```



# Accelerate Python

```
use pyo3::pyfunction;

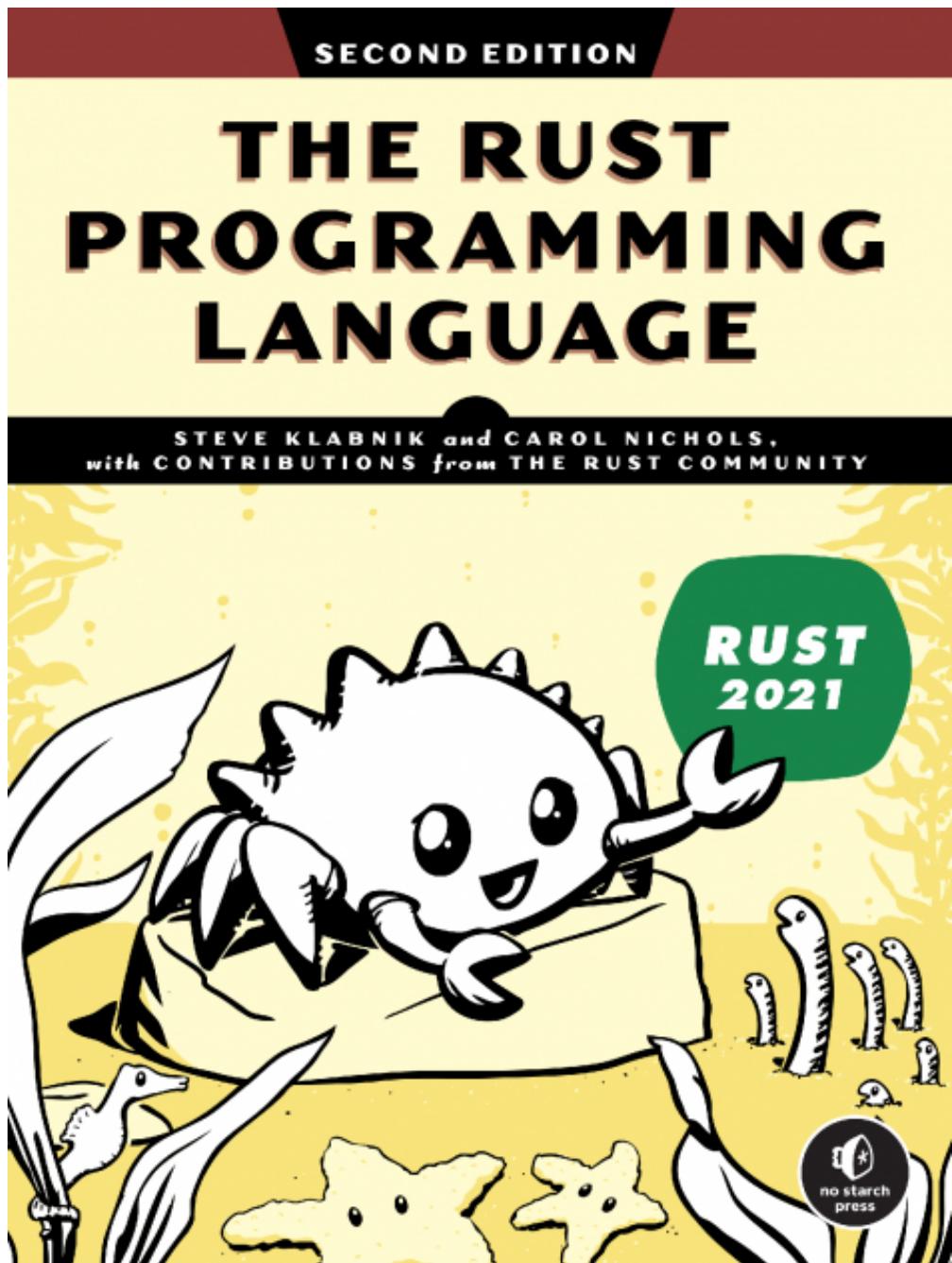
#[pyfunction]
fn fibonacci(a: u32) -> u32 {
    match n {
        1 | 2 => 1,
        _ => fibonacci(n - 1) +
            fibonacci(n - 2),
    }
}
```



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



# The Rust Programming Language





Rust course @ FEI VSB-TUO

[github.com/kobzol/rust-course-fei](https://github.com/kobzol/rust-course-fei)





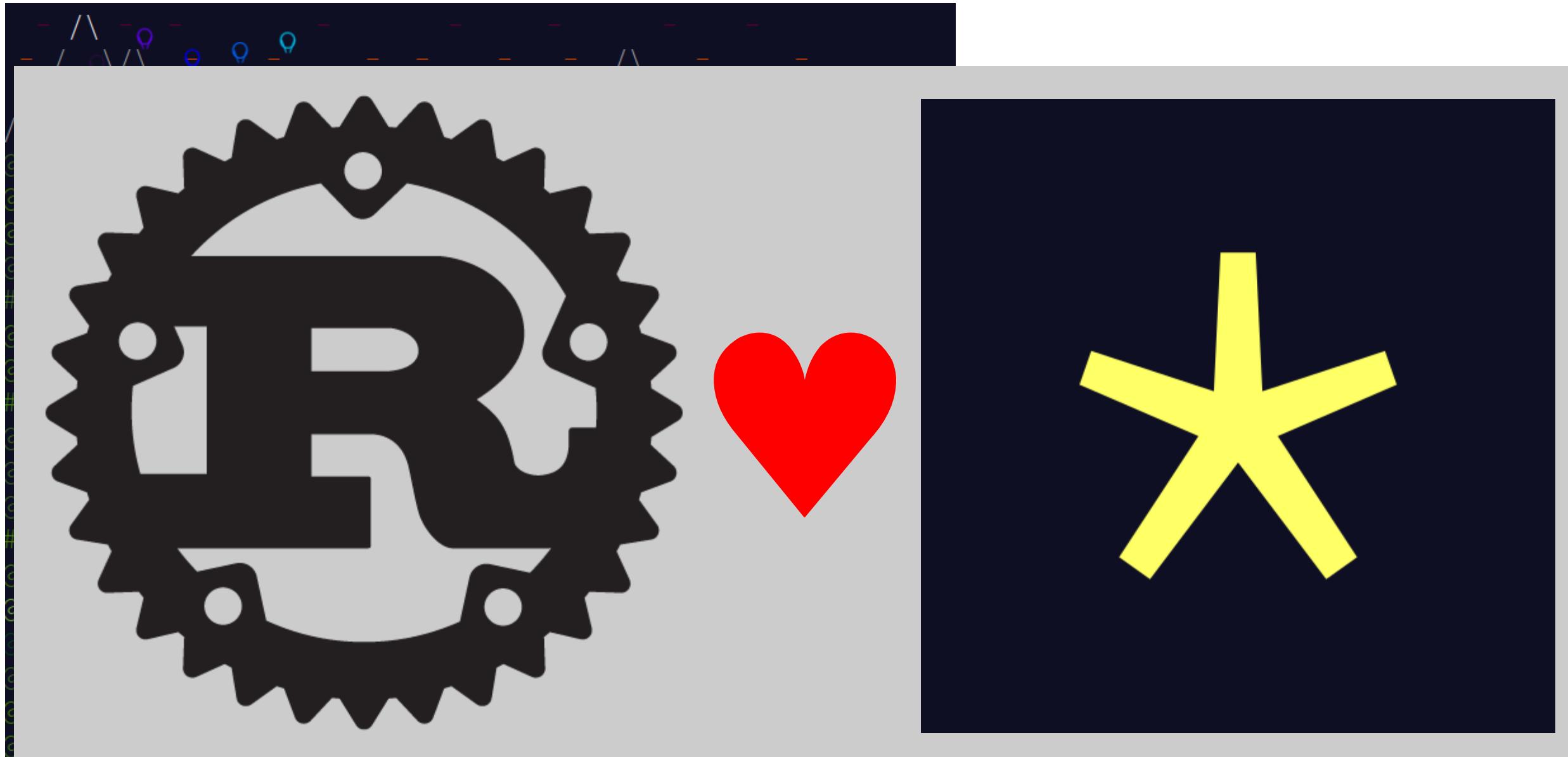


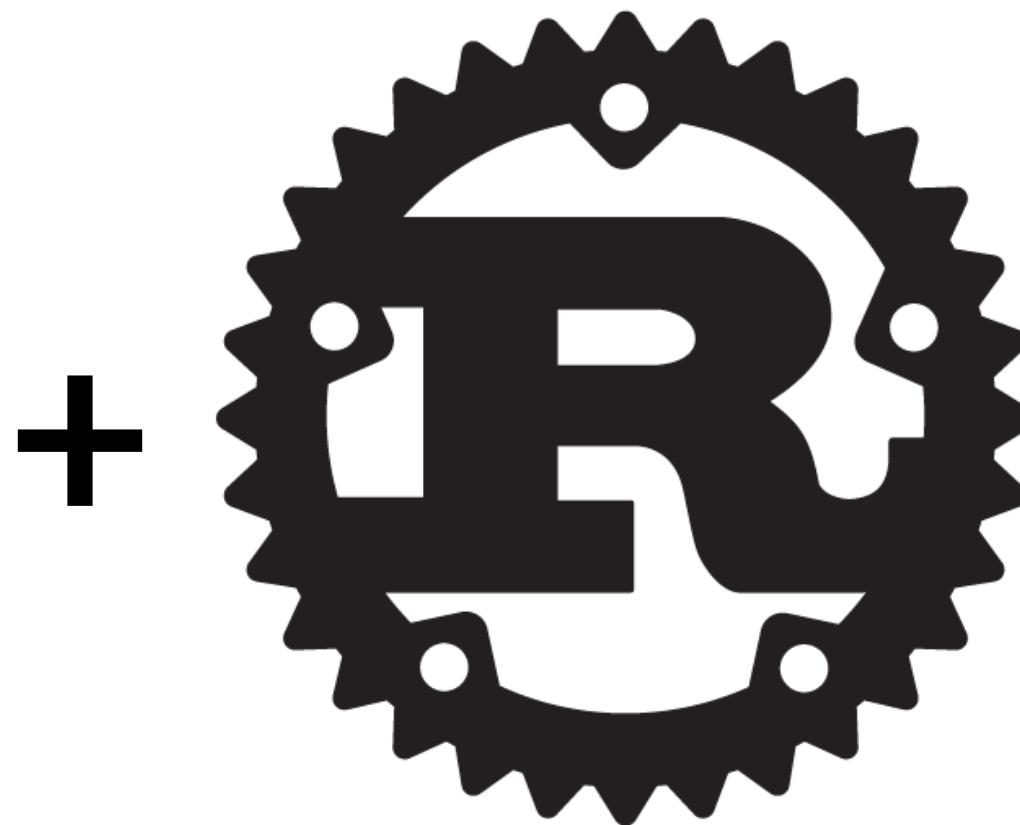
# Advent of Code

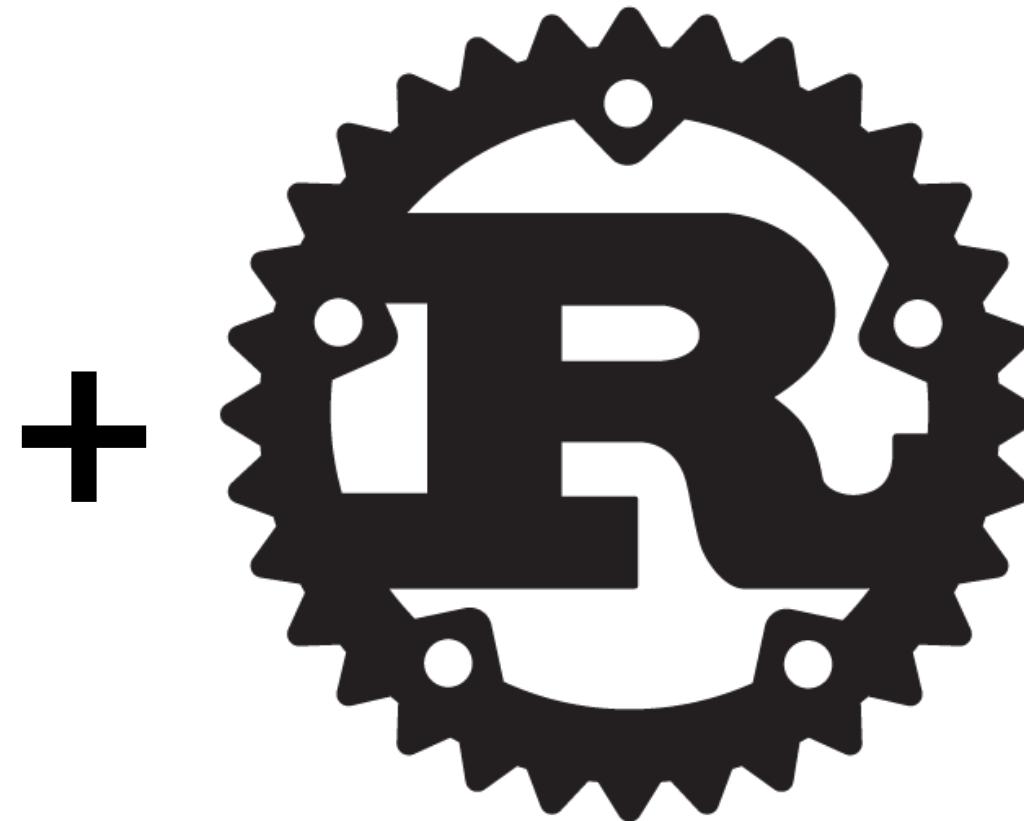
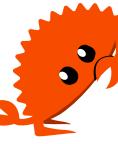




# Advent of Code







Beginning of 2024 (?)  
Stay tuned :-)



# Děkuji za pozornost

Slide jsou dostupné zde:



Slidy byly vytvořeny pomocí [github.com/spirali/elsie](https://github.com/spirali/elsie)