API Days Paris

Zacaria Chtatar - December 2023

https://havesome-rust-apidays.surge.sh

Forget TypeScript

Forget TypeScript
Choose Rust

Forget TypeScript
Choose Rust
to build

Forget TypeScript

Choose Rust

to build

Robust, Fast and Cheap APIs

Some context

2009

JS lives in the browser

Until Nodejs

V8 engine, modules system & NPM

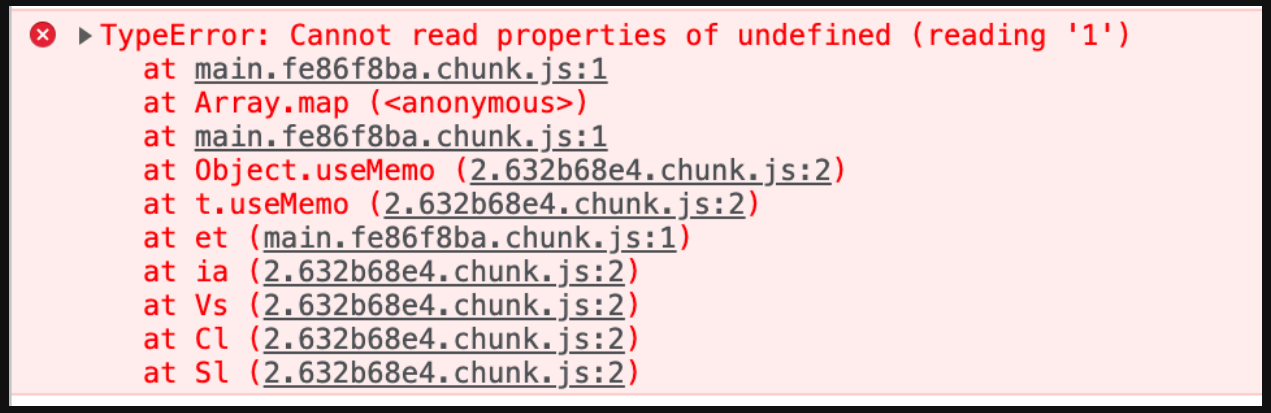
- modules
- boom of easy to reuse code
- fullstack JS : Easy to start with, hard to master

Fullstack

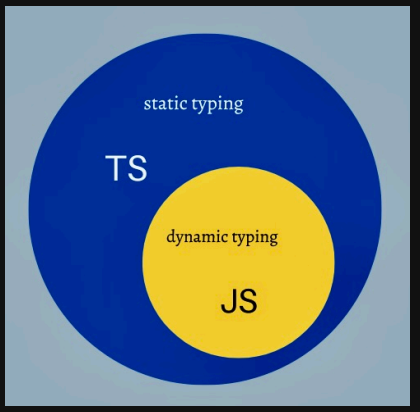
- paradigm clash between static and dynamic
- JS doesn't help you follow strict API interfaces

Fullstack

- paradigm clash between static and dynamic
- JS doesn't help you follow strict API interfaces



TypeScript



Benefits:

- IDE Developer experience
- OOP patterns
- compiler checks
- type system

=> better management of big codebase

Pain points

- does not save you from dealing with JS
- adds types management
- adds static layer onto dynamic layer

Pain points

- does not save you from dealing with JS
- adds types management
- adds static layer onto dynamic layer

JSDoc answers to the precise problem of hinting types without getting in your way

New stakes, new needs

Stakes	Needs
worldwide scale	scalability
privacy	security
market competition	functionality
environment	computation time
	memory footprint
human lives	safety

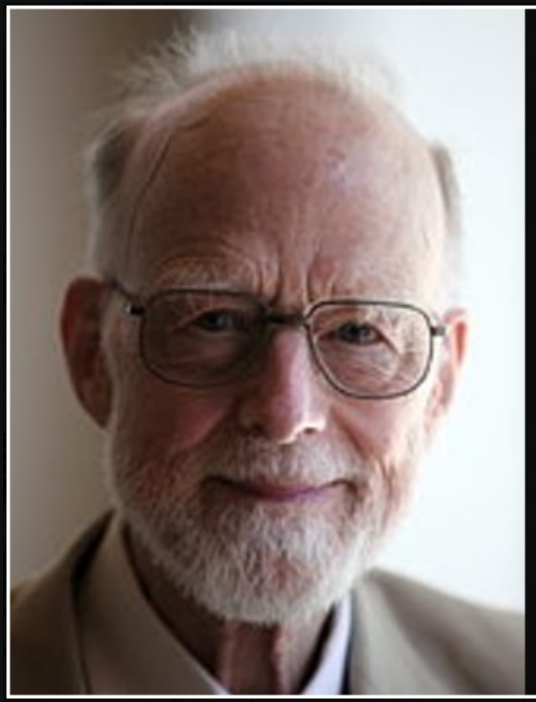
Stakes	Needs
worldwide scale	scalability
privacy	security
market competition	functionality
environment	computation time
	memory footprint
human lives	safety

TypeScript is not enough

Introducing Rust

Fast, Reliable, Productive: pick three

Stable



I call it my billion-dollar mistake. It was the invention of the null reference in 1965.

— Tony Hoare —

AZ QUOTES

enums

```
struct FakeCat {
    alive: bool,
    hungry: bool,
}

let zombie = FakeCat { alive: false, hungry: true }; // ???
```

enums

```
struct FakeCat {
    alive: bool,
    hungry: bool,
}

let zombie = FakeCat { alive: false, hungry: true }; // ???
```

Rust makes it easy to make invalid state unrepresentable

enums

```
struct FakeCat {
    alive: bool,
    hungry: bool,
}

let zombie = FakeCat { alive: false, hungry: true }; // ???
```

Rust makes it easy to make invalid state unrepresentable

```
enum RealCat {
    Alive { hungry: bool }, // enums can contain structs
    Dead,
}

let cat = RealCat::Alive { hungry: true };
let dead_cat = RealCat::Dead;
```

Option enum

```
// full code of the solution to the billion dollar mistake
// included in the standard library
enum Option<T> {
    None,
    Some(T),
}
```

```
let item: Option<Item> = get_item();

match item {
    Some(item) => map_item(item),
    None => handle_none_case(), // does not compile if omitted
}
```

What is wrong with this function ?

```
function getConfig(path: string): string {
  return fs.readFileSync(path);
}
```

What is wrong with this function ?

```
function getConfig(path: string): string {
  return fs.readFileSync(path);
}
```

There is no hint that readFileSync can throw an error under some conditions

Result enum

```rust
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```rust
fn get_config(path: &str) -> Result<String, io::Error> {
    fs::read_to_string(path)
}

// That's a shortcut, don't send me to prod !
fn dirty_get_config(path: &str) -> String {
    fs::read_to_string(path).unwrap() // panics in case of error
}
```

Result enum

```rust
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```rust
fn get_config(path: &str) -> Result<String, io::Error> {
    fs::read_to_string(path)
}

// That's a shortcut, don't send me to prod !
fn dirty_get_config(path: &str) -> String {
    fs::read_to_string(path).unwrap() // panics in case of error
}
```

We need to clarify what can go wrong

Result enum

```rust
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

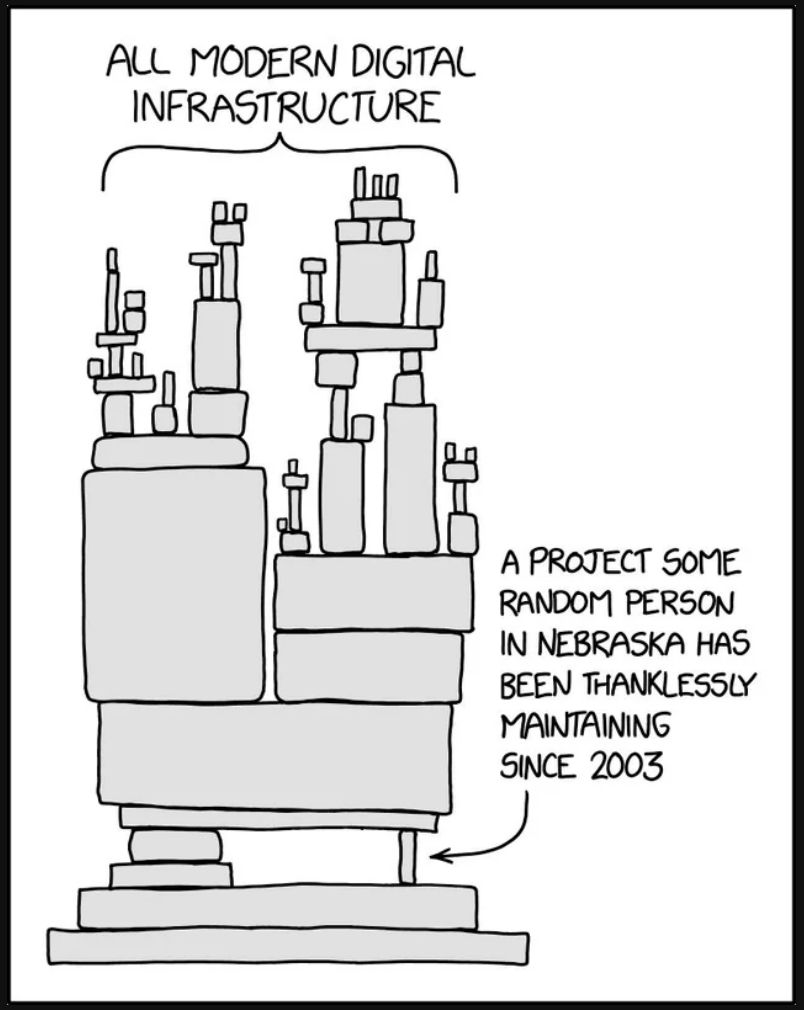
```rust
fn get_config(path: &str) -> Result<String, io::Error> {
    fs::read_to_string(path)
}

// That's a shortcut, don't send me to prod !
fn dirty_get_config(path: &str) -> String {
    fs::read_to_string(path).unwrap() // panics in case of error
}
```

We need to clarify what can go wrong

It's even better when it's embedded in the language

2016 : Do you remember the left-pad incident ?



Source

crates.io

- no crate (package) unpublish
- can disable crate only for new projects

Linux: The Kernel

Linux: The Kernel

- attract young devs

Linux: The Kernel

- attract young devs
- 2/3 of vunerabilities come from memory management

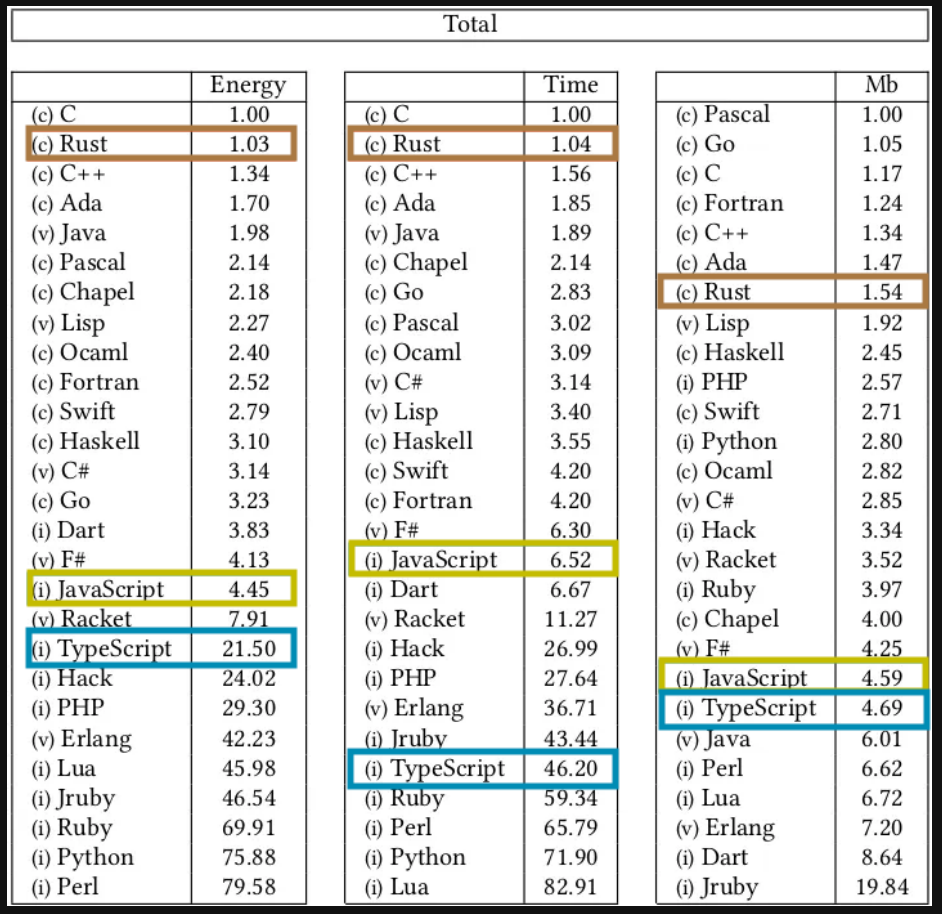
Linux: The Kernel

- attract young devs
- 2/3 of vunerabilities come from memory management
- Kernel is in C and Assembly

Linux: The Kernel

- attract young devs
- 2/3 of vunerabilities come from memory management
- Kernel is in C and Assembly
- Linus Torvalds : C++ 🤦

Fast



Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

2017 : Energy efficiency accross programing languages

Github: Code Search index

45 million repos

28 TB of unique content

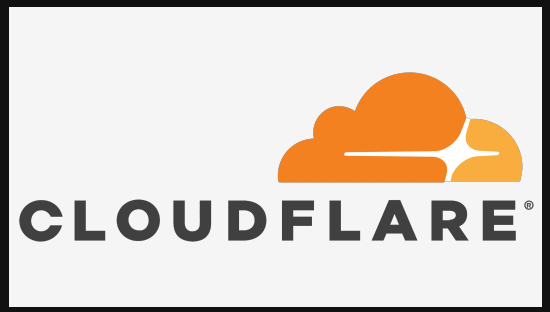
Github: Code Search index

45 million repos

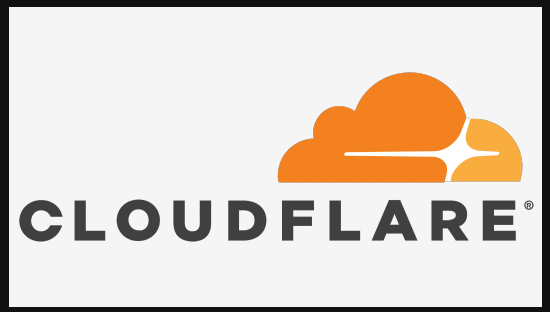
28 TB of unique content

- several months with Elasticsearch
- 36h with Rust and Kafka
- 640 queries /s

Cloudflare: HTTP proxy

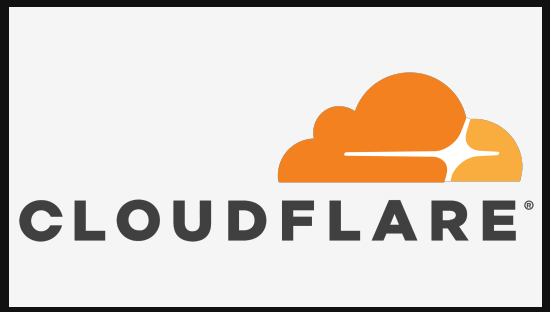


Cloudflare: HTTP proxy



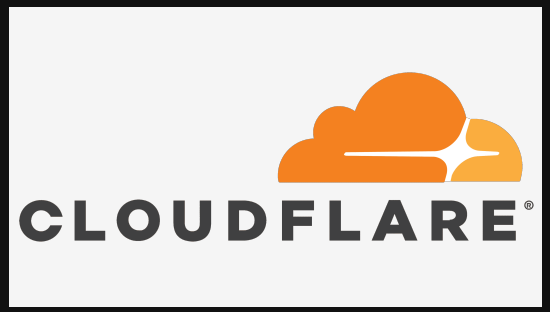
- nginx not fast enough 🤯

Cloudflare: HTTP proxy



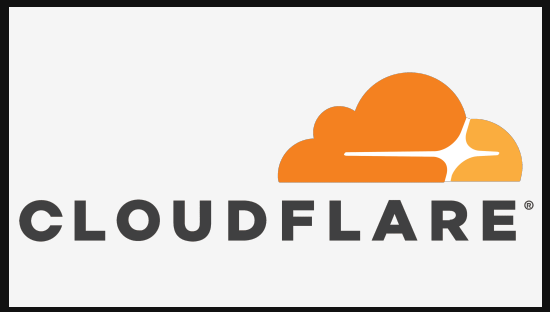
- nginx not fast enough 🤯
- hard to customize in C

Cloudflare: HTTP proxy



- nginx not fast enough 🤯
- hard to customize in C
- allows to share connections between threads

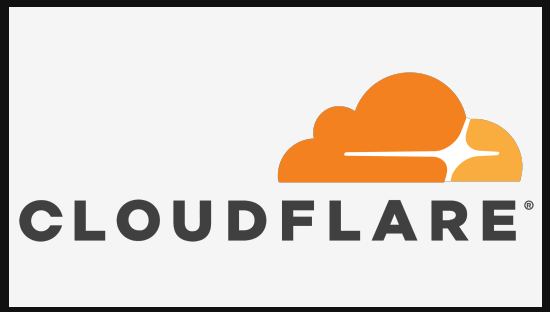
Cloudflare: HTTP proxy



- nginx not fast enough 🤯
- hard to customize in C
- allows to share connections between threads

= 160x less connections to the origins

Cloudflare: HTTP proxy



- nginx not fast enough 🤯
- hard to customize in C
- allows to share connections between threads

= 160x less connections to the origins

= 434 years less handshakes per day

Discord: Message read service

Discord: Message read service

- cache of a few billion entries

Discord: Message read service

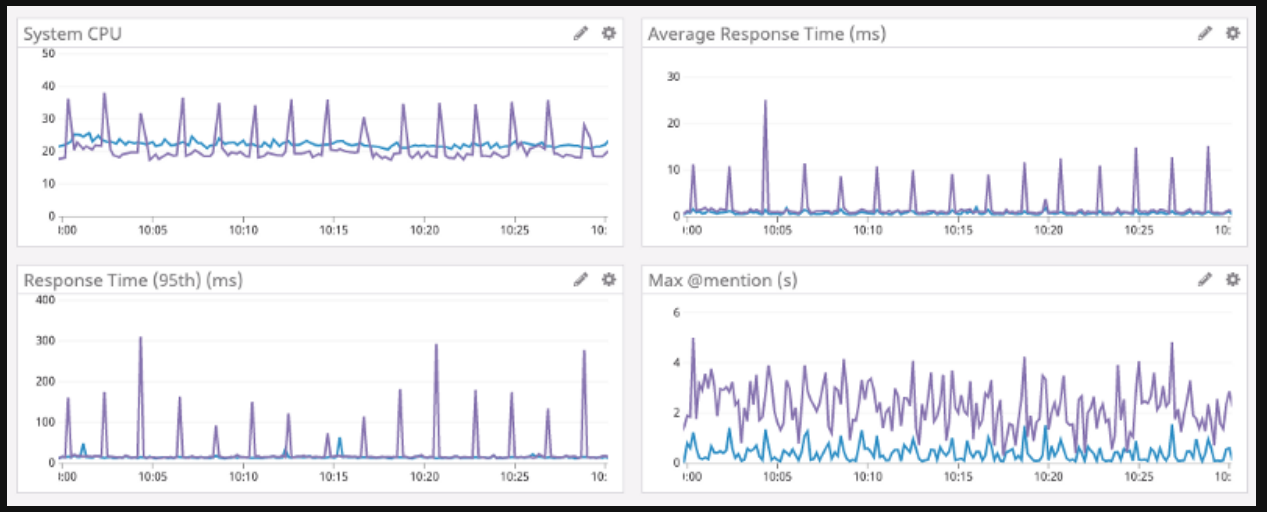
- cache of a few billion entries
- every connection, message sent and read...

Discord: Message read service

- cache of a few billion entries
- every connection, message sent and read...
- latences every 2 minutes because of Go Garbage Collector

Discord: Message read service

- cache of a few billion entries
- every connection, message sent and read...
- latences every 2 minutes because of Go Garbage Collector



Cheap

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

- cpu & memory
- less bugs
- learning curve
- less cases to test

Attractive

Attractive

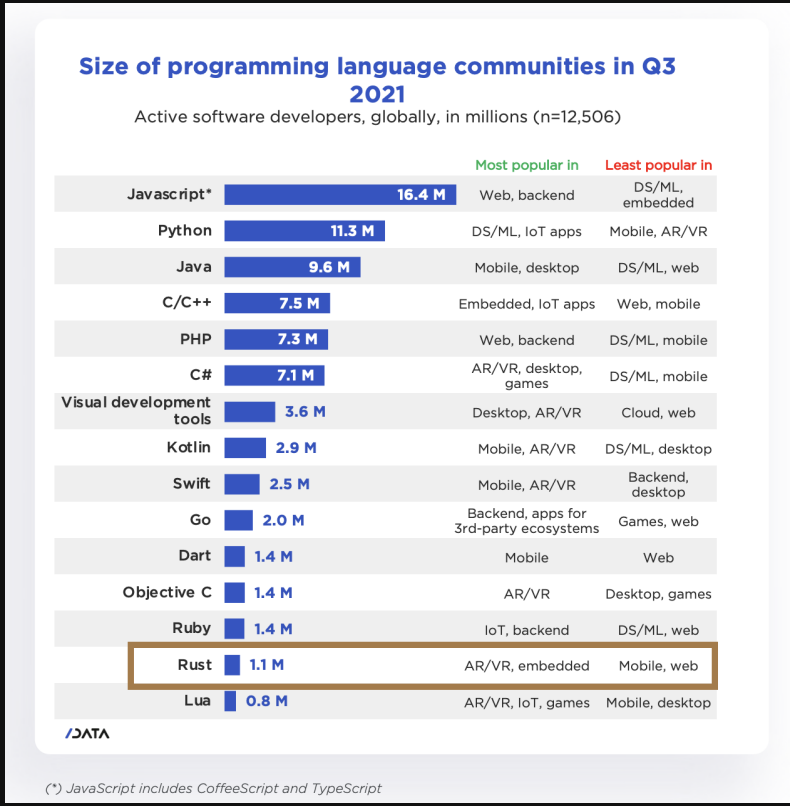
Most admired language according to StackOverflow for 8 years !

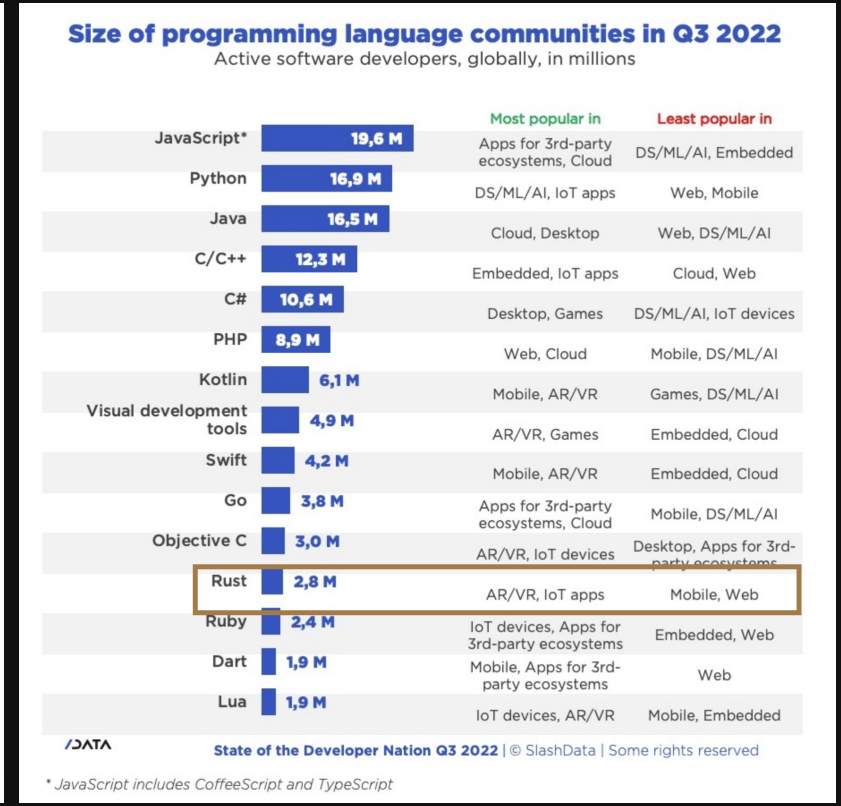
Attractive

Most admired language according to StackOverflow for 8 years !

Only place where there is **more** devs available than offers

Growing community





Features

- static types
- compiled
- no GC
- compiler developed in Rust
- low and high level : zero cost abstraction
- no manual memory management : Ownership & Borrow checker

Features

- static types
- compiled
- no GC
- compiler developed in Rust
- low and high level : zero cost abstraction
- no manual memory management : Ownership & Borrow checker

=> There is no blackbox between you and the machine

Features

- static types
- compiled
- no GC
- compiler developed in Rust
- low and high level : zero cost abstraction
- no manual memory management : Ownership & Borrow checker

=> There is no blackbox between you and the machine

=> Better predictability

Features

- static types
- compiled
- no GC
- compiler developed in Rust
- low and high level : zero cost abstraction
- no manual memory management : Ownership & Borrow checker

=> There is no blackbox between you and the machine

=> Better predictability

=> Awesome developer experience

Ownership rules

Ownership rules

- Only one variable owns data at a time

Ownership rules

- Only one variable owns data at a time
- Multiple readers or one writer

Ownership rules

- Only one variable owns data at a time
- Multiple readers or one writer

=> Memory is freed as soon as variable is out of scope

Ownership rules

- Only one variable owns data at a time
- Multiple readers or one writer

=> Memory is freed as soon as variable is out of scope

It's like a fundamental problem has been solved

The compiler ❤️

```rust
fn say(message: String) {
    println!("{}", message);
}

fn main() {
    let message = String::from("hey");
    say(message);
    say(message);
}
```

```
error[E0382]: use of moved value: `message`
  --> src/main.rs:12:9
   |
10 |     let message = String::from("hey");
   |         ------- move occurs because `message` has type `String`, which does
 not implement the `Copy` trait
11 |     say(message);
   |         ------- value moved here
12 |     say(message);
   |         ^^^^^^^ value used here after move
   |
note: consider changing this parameter type in function `say` to borrow instead
if owning the value isn't necessary
  --> src/main.rs:5:17
   |
5  | fn say(message: String) {
   |    ---          ^^^^^^ this parameter takes ownership of the value
   |    |
   |    in this function
help: consider cloning the value if the performance cost is acceptable
   |
11 |     say(message.clone());
   |                ++++++++

For more information about this error, try `rustc --explain E0382`.
error: could not compile `learn-dyn` (bin "learn-dyn") due to previous error
```

Tools

- cargo test : Integration Tests, Unit Ttests
- cargo fmt
- cargo bench
- clippy : lint
- bacon : reload
- rust-analyzer : IDE developer experience

Cargo doc stays up to date

```rust
/// Formats the sum of two numbers as a string.
///
/// # Examples
///
/// ```
/// let result = mycrate::sum_as_string(5, 10);
/// assert_eq!(result, "15");
/// ```
pub fn sum_as_string(a: i32, b: i32) -> String { (a + b).to_string() }
```

```
cargo doc --open
```

Cargo doc stays up to date

```rust
/// Formats the sum of two numbers as a string.
///
/// # Examples
///
/// ```
/// let result = mycrate::sum_as_string(5, 10);
/// assert_eq!(result, "15");
/// ```
pub fn sum_as_string(a: i32, b: i32) -> String { (a + b).to_string() }
```

```
cargo doc --open
```

Cargo doc stays up to date

```rust
/// Formats the sum of two numbers as a string.
///
/// # Examples
///
/// ```
/// let result = mycrate::sum_as_string(5, 10);
/// assert_eq!(result, "15");
/// ```
pub fn sum_as_string(a: i32, b: i32) -> String { (a + b).to_string() }
```

```
cargo doc --open
```

serde_json
1.0.108

All Items

Modules
Macros
Structs
Enums
Functions
Type Aliases

Crates
serde_json

Crate serde_json

source · [-]

Serde JSON

JSON is a ubiquitous open-standard format that uses human-readable text to transmit data objects consisting of key-value pairs.

```
{
    "name": "John Doe",
    "age": 43,
    "address": {
        "street": "10 Downing Street",
        "city": "London"
    },
    "phones": [
        "+44 1234567",
        "+44 2345678"
    ]
}
```

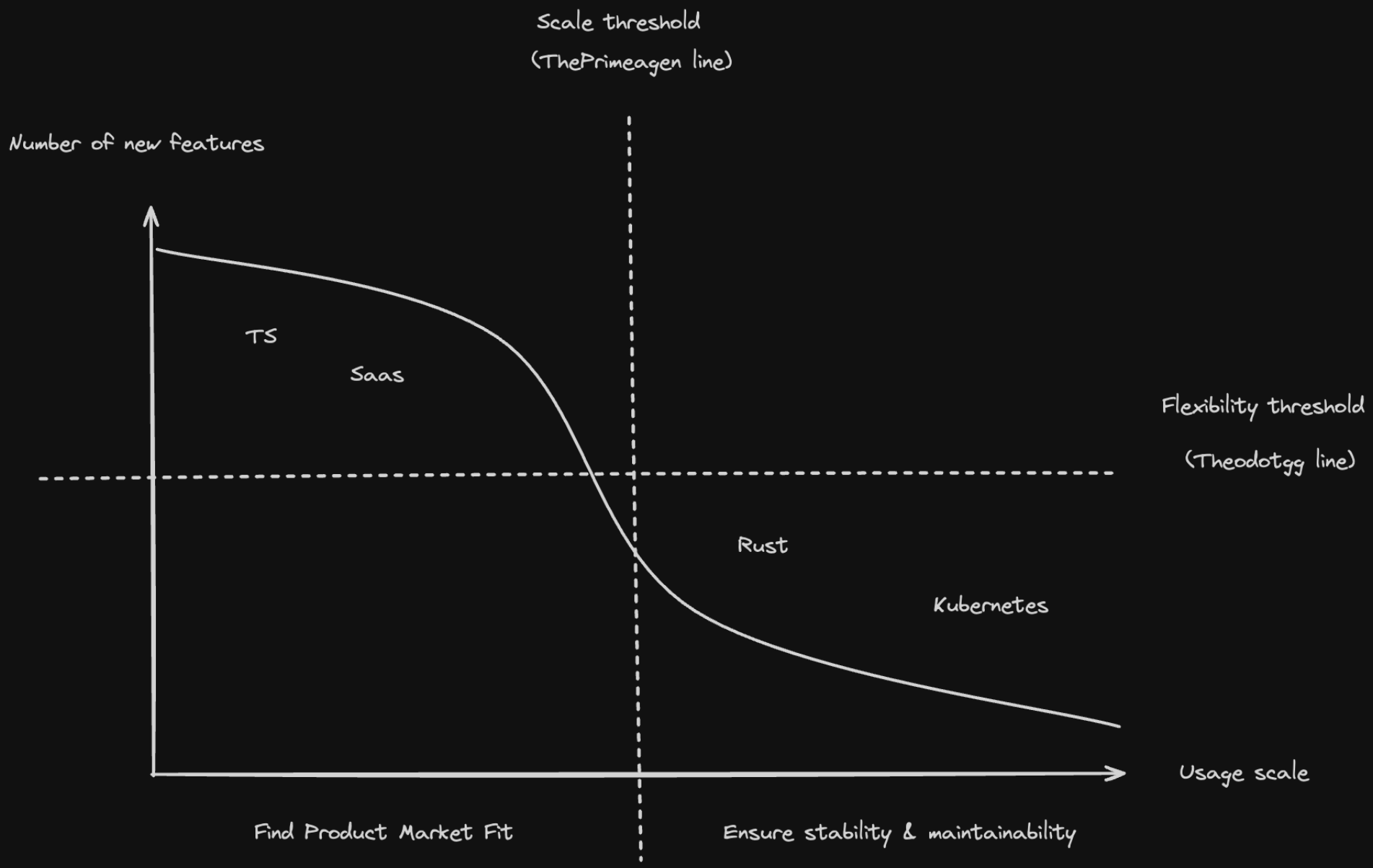
There are three common ways that you might find yourself needing to work with JSON data in Rust.

- **As text data.** An unprocessed string of JSON data that you receive on an HTTP endpoint, read from a file, or prepare to send to a remote server.
- **As an untyped or loosely typed representation.** Maybe you want to check that some JSON data is valid before

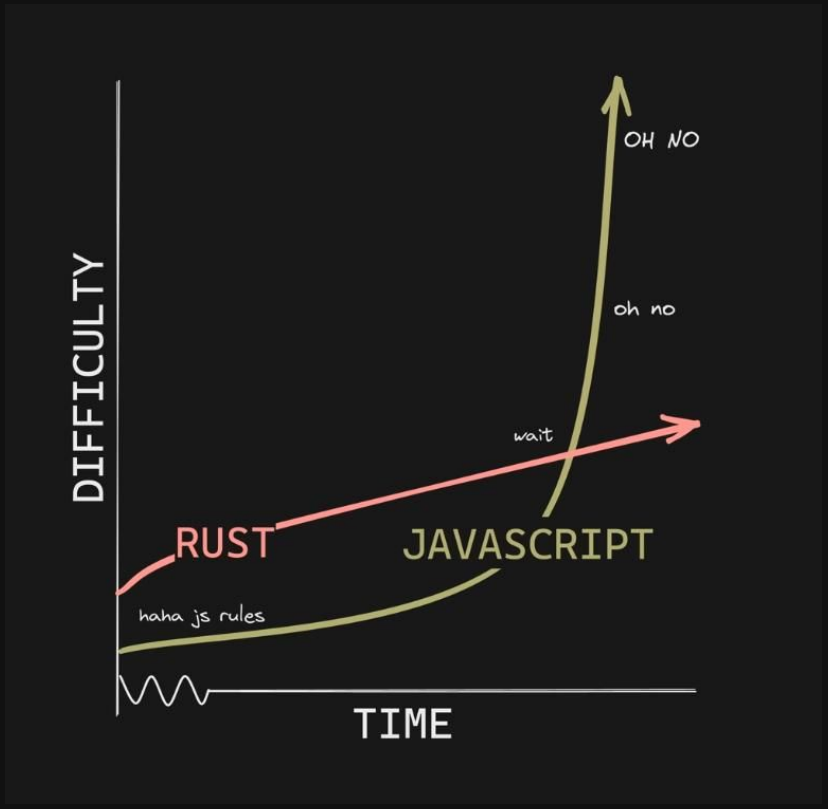
Possible struggles

- projects move slower
- embrace the paradigm
- takes 3 to 6 months to become productive
- build time
- work with external libraries
- ecosystem calmer than JS

Project lifetime



In other languages simple things are easy and complex things are possible, in Rust simple things are possible and complex things are EASY.



Get started as dev

- Rust book - reference
- Rust by example - condensed reference
- Rustlings - exercises
- A half-hour to learn Rust - quick walkthrough
- Comprehensive Rust by Google - complete walkthrough
- Noboilerplate - quick videos ❤️
- Code to the moon - longer videos
- Let's get rusty- Youtube & paid bootcamp - not sponsored
- Awesome Rust - keywords discovery
- Roadmap - keywords discovery

Get started as manager

Get started as manager

- find dev interested in Rust: there are a lot

Get started as manager

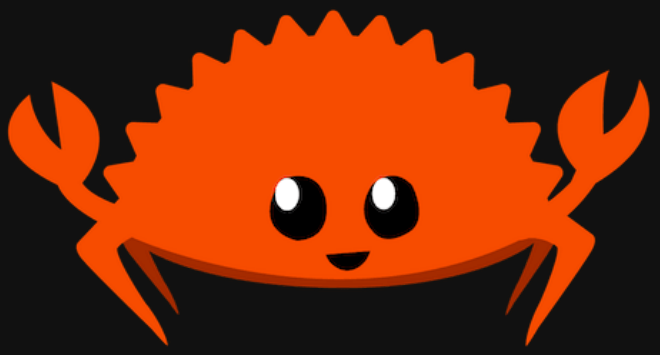
- find dev interested in Rust: there are a lot
- start with simple projects:
  - CLI
  - lambdas
  - microservice
  - network app
  - devops tools

Get started as manager

- find dev interested in Rust: there are a lot
- start with simple projects:
  - CLI
  - lambdas
  - microservice
  - network app
  - devops tools

Make the world a safer, faster and sustainable place

Thank you



Slides : https://havesome-rust-apidays.surge.sh

𝕏 @ChtatarZacaria - 🌍 havesomecode.io

Q&A

Governance

Rust Project

Rust Foundation

Release every 6 weeks

Backward compatibility

Breaking changes are opt-in thanks to editions