PV281: Programování v Rustu

Obsah

Organizační informace

Interaktivní osnova

Primárním zdrojem informací během semestru je předmětová Interaktivní osnova.

Přednášky

2h týdně, vždy v pondělí v 18–20 hod.

Přednášející:
Lukáš Grolig, Stanislav Zeman, Marek Mišík, Marek Vrbka, Tomáš Sedláček, Adam Valt

Během přednášky projdeme teorii a základní příklady.

Přednášky záznamy přednášek nejsou standardně dostupné.

Stream přednášek pouze po předchozím oznámení na YT nebo Twitchi.

Cvičení

2h týdně, celkem 12 cvičení

Cvičící: **Petr Wehrenberg, Marek Mišík, Ondřej Hrdlička, Marek Vrbka, Petr Kadlec, Lukáš Chudíček, Miloš Hegr, Otakar Hirš**

Na cvičení si zopakujete probranou látku a budete vypracovávat příklady s pomocí cvičících.

Poděkování

Ondrovi Hrdličkovi a ostatním za pomoc s organizací předmětu.

Organizační informace, diskuze k Rustu, pomoc s úkoly či projekty, hledání týmu, …

https://discord.gg/8yvsZu9ej4

Hodnocení

Hodnocení předmětu

Maximum bodů: **100**
Minimum pro úspěšné zakončení předmětu: **75**

Hodnocení

Cvičení

- Možnost získat bonusové body
- Účast není povinná, ALE
  za aktivní účast získáte **možnost opravit si odevzdání iterace**
  (toto pravidlo platí od 3. týdne semestru včetně)

Hodnocení

Iterace

- Celkem **10 iterací**
- Každá za **maximálně 5 bodů**
- Každá z nich vás seznámí s konktrétní funkcionalitou Rustu
- Většinou automaticky testované, opravující kontroluje kvalitu kódu

- V základu máte 1 pokus na odevzdání
- Po code review můžete, ale nemusíte využít možnost opravy získanou na základě aktivní účasti na cvičení

Hodnocení

Týmový projekt

- **50 bodů** za týmový projekt zakončený obhajobou ve zkouškovém

- **4členné**, výjimečně 3členné **týmy** – nikdy ne méně ani ne více
- Týmy lze vytvářet **napříč seminárními skupinami**.
- Složení týmu lze měnit do 10. týdne semestru (včetně).

- Zadání projektu jsou volná, očekáváme kreativitu týmu při vlastním dodefinování zadání.
- Zadání plánujeme zveřejnit ve **3. týdnu** semestru.

Hodnocení

Bonusové body

- Na přednášce za aktivitu
- Na cvičeních za aktivitu
- Za dobrovolné, bonusové podčásti iterací
- Za špičkové zpracování úkolů/projektu a výrazné překročení rozsahu

Očekávané znalosti?

- programování v C, programovaní v managovaném jazyce (C# nebo Java), programování ve funkcionálním jazyce
- práce s databází - aktivní znalost SQL a návrhu databáze
- základní znalosti kolem REST rozhraní - endpointy, entity, znalost HTTP
- znalost webobých technologií na úrovni PB138

Co za semestr probereme?

Probíraná látka

- Úvod - historie, hlavní (ne)výhody Rustu, založení projektu
- Syntaxe jazyka a jeho specifika
- CLI aplikace a práce se soubory, obsluha chybových stavů
- Iterátory a datové struktury
- Paralelní programování, strukturování projektu
- Asynchronní programování

Probíraná látka

- Práce s databází
- RESTové služby v Actix
- Server-side aplikace v Actix a HTMX
- Desktopové aplikace v Tauri
- gRPC protokol
- Unsafe Rust, makra a foreign function interface

Probíraná látka mimo Rust

- Budeme hodně pracovat s Gitem a GitLabem
- Budeme psát čisté SQL

Studijní materiály

Web

The Rust Programming Language
Rust By Example

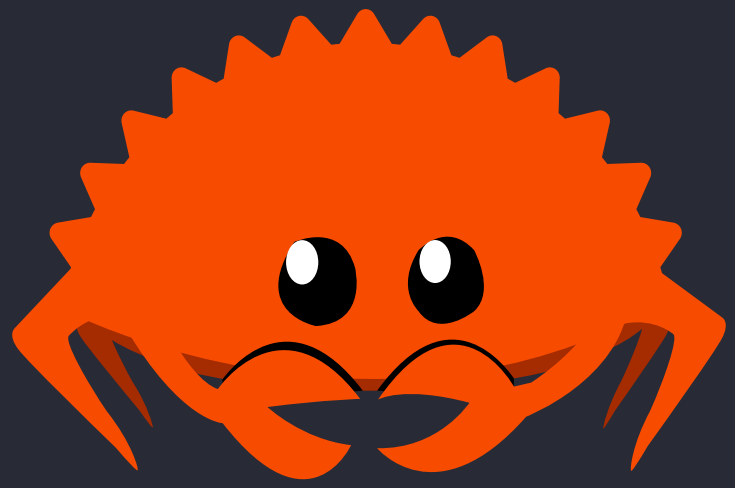
Literatura

Programming Rust: Fast, Safe Systems Development, 2nd Edition

Pokročilá literatura

Rust for Rustaceans: Idiomatic Programming for Experienced Developers

O Rustu a jeho historii

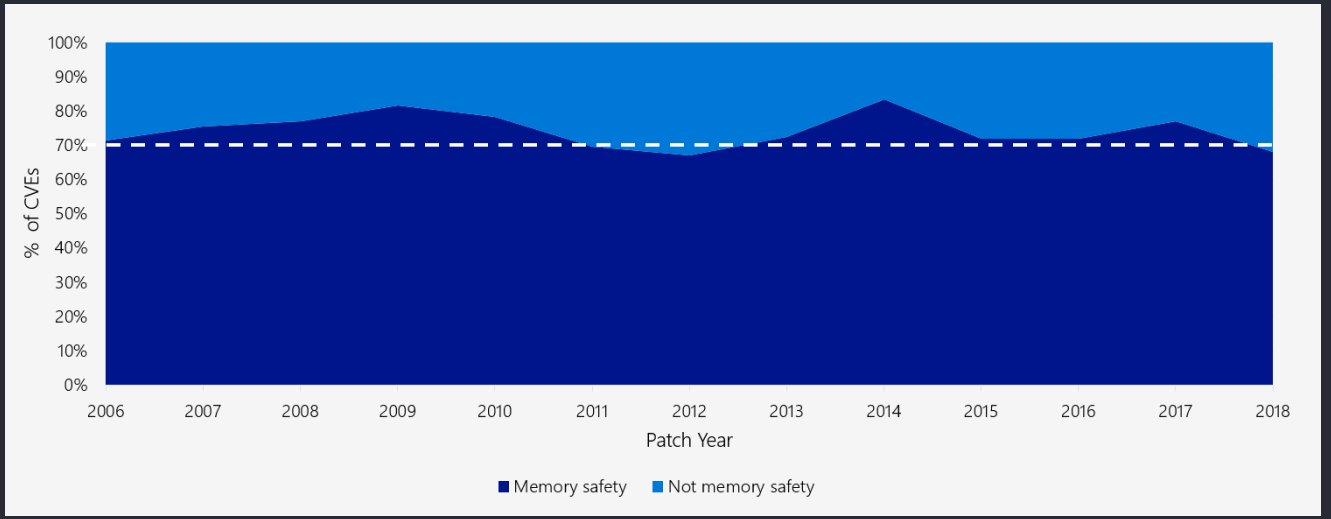


Seznamte se:

Krab Ferris

Historie

- Rust vznikl v Mozille
- Cílem bylo vytvořit jazyk bez garbage kolekce s bezpečnou prací s pamětí



70% bezpečnostních děr v Microsoftu bylo spojených s prací s pamětí.

https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

Rust Foundation

Mozilla ale dál nezvládala sama rozvíjet Rust. To vedlo ke vzniku Rust Foundation v roce 2020.

Zakládajícími členy se stali:

Další známé firmy využívající Rust

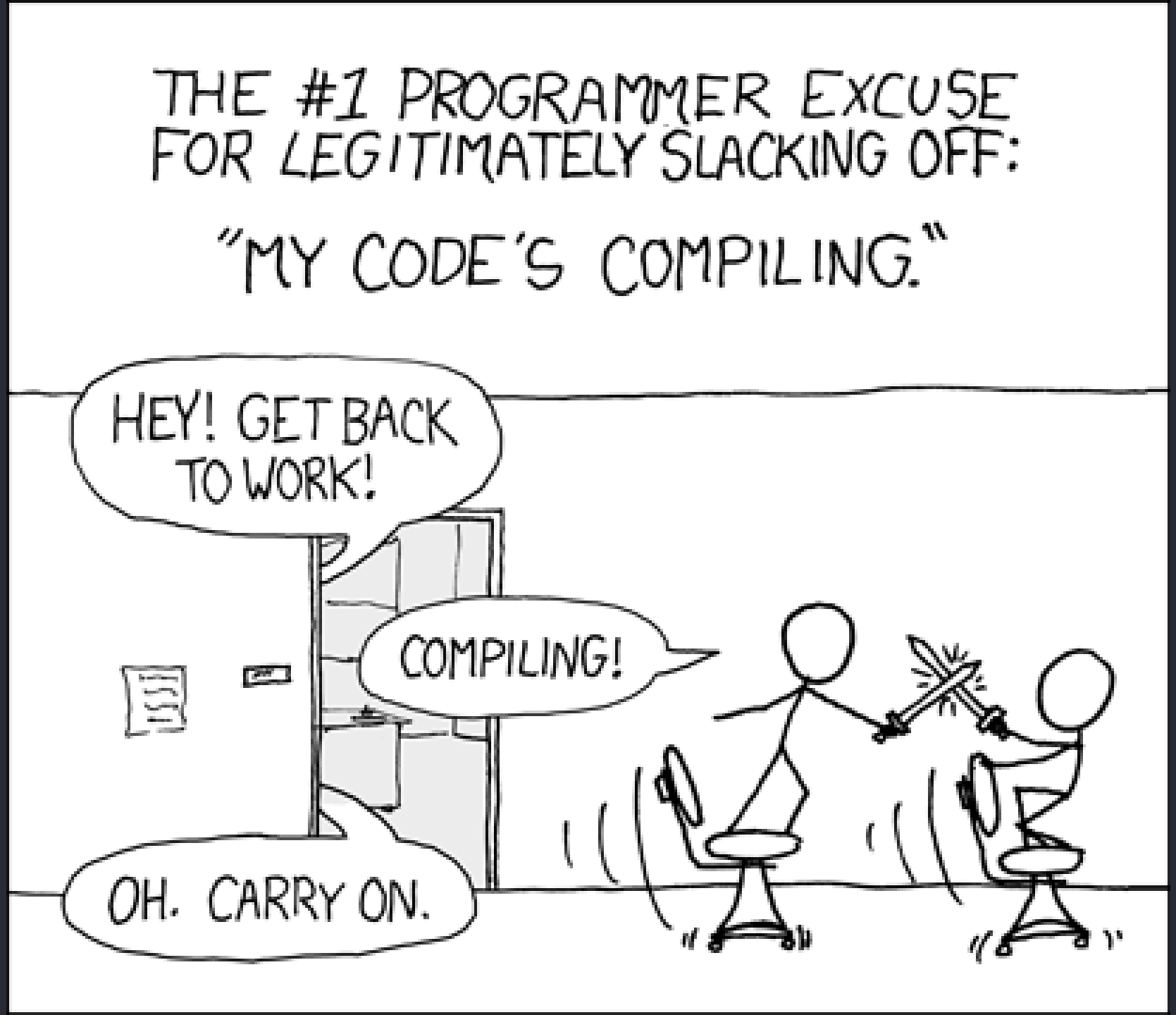
Tento seznam se už hodně natáhl, ale můžeme jmenovat:
1Password, Apple, Canonical, Cloudflare, Discord, Dropbox, Figma, Facebook, System76, OVH

Hlavní výhody Rustu

Bezpečnost

V jazycích typu C vznikají problémy s manuální správou pamětí jako dangling pointer, dvojité uvolnění aj.
V moderním C++ je spousta věcí řešena technikami jako je RAII nebo smartpointy.
Rozdílem je, že Rustu toto nemusí nehlídat člověk, ale řeší překladač.

Cenou za to je doba překladu.


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Hlavní výhody Rustu

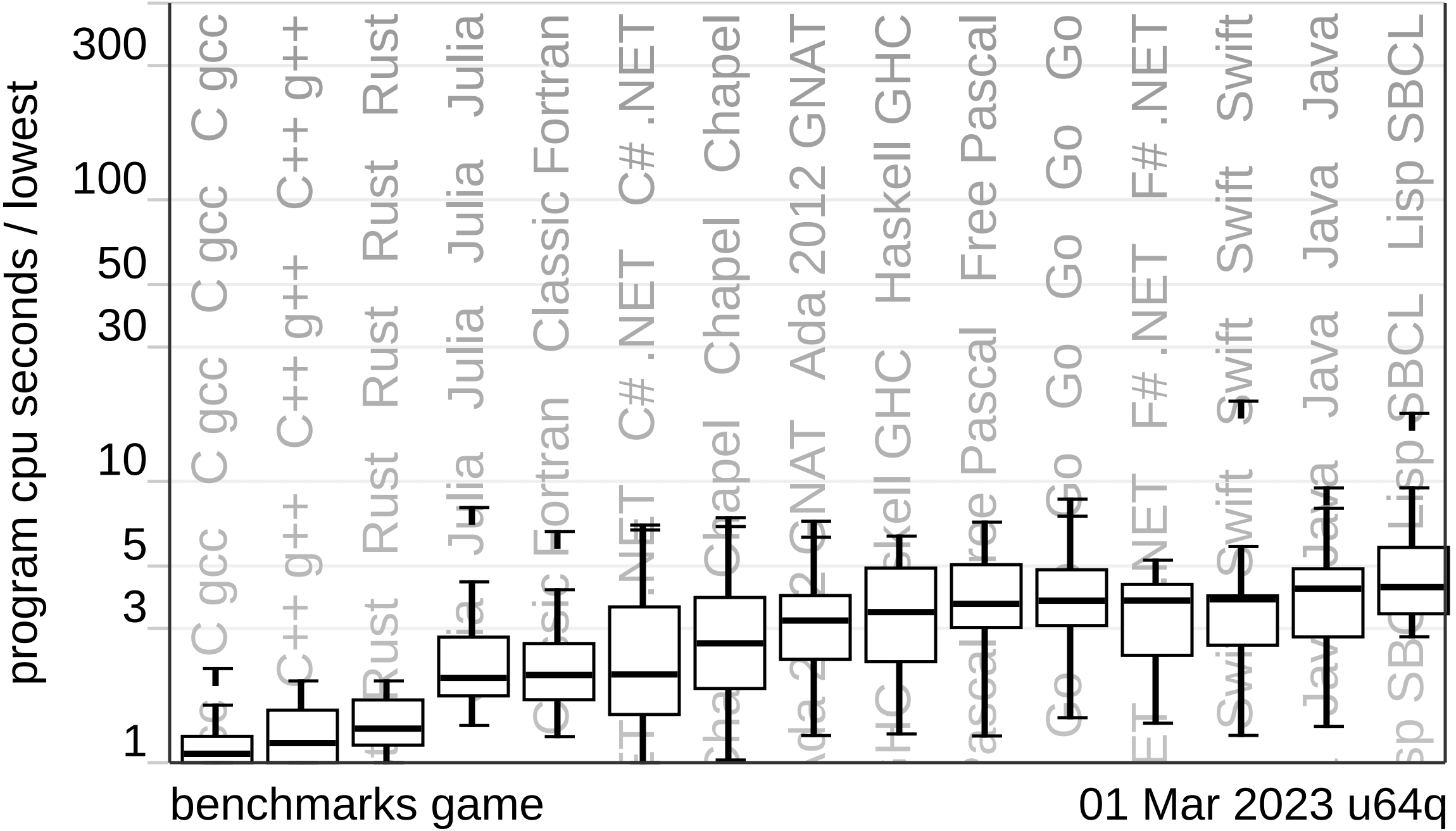
Rychlost

Prakticky všechny dnešní jazyky jsou pomalejší než C a C++.
Je to kvůli abstrakcím, garbage kolekci aj.
Výsledný program běžící v Rustu je na stejné úrovni v rychlosti běhu jako C/C++.

To máme díky:

- **zero cost abstrakci**
- **správě paměti během kompilace**

How many times more CPU seconds?



Srovnání Rustu s C

reverse-complement

source	secs	mem	gz	busy	cpu load
Rust	**0.45**	498,964	3040	0.77	25% 23% 100% 25%
C gcc	0.86	712,208	820	1.27	99% 28% 1% 19%

binary-trees

source	secs	mem	gz	busy	cpu load
Rust	**1.09**	198,728	765	3.90	87% 98% 88% 86%
C gcc	1.54	168,832	809	4.35	60% 67% 57% 100%

Srovnání Rustu s C

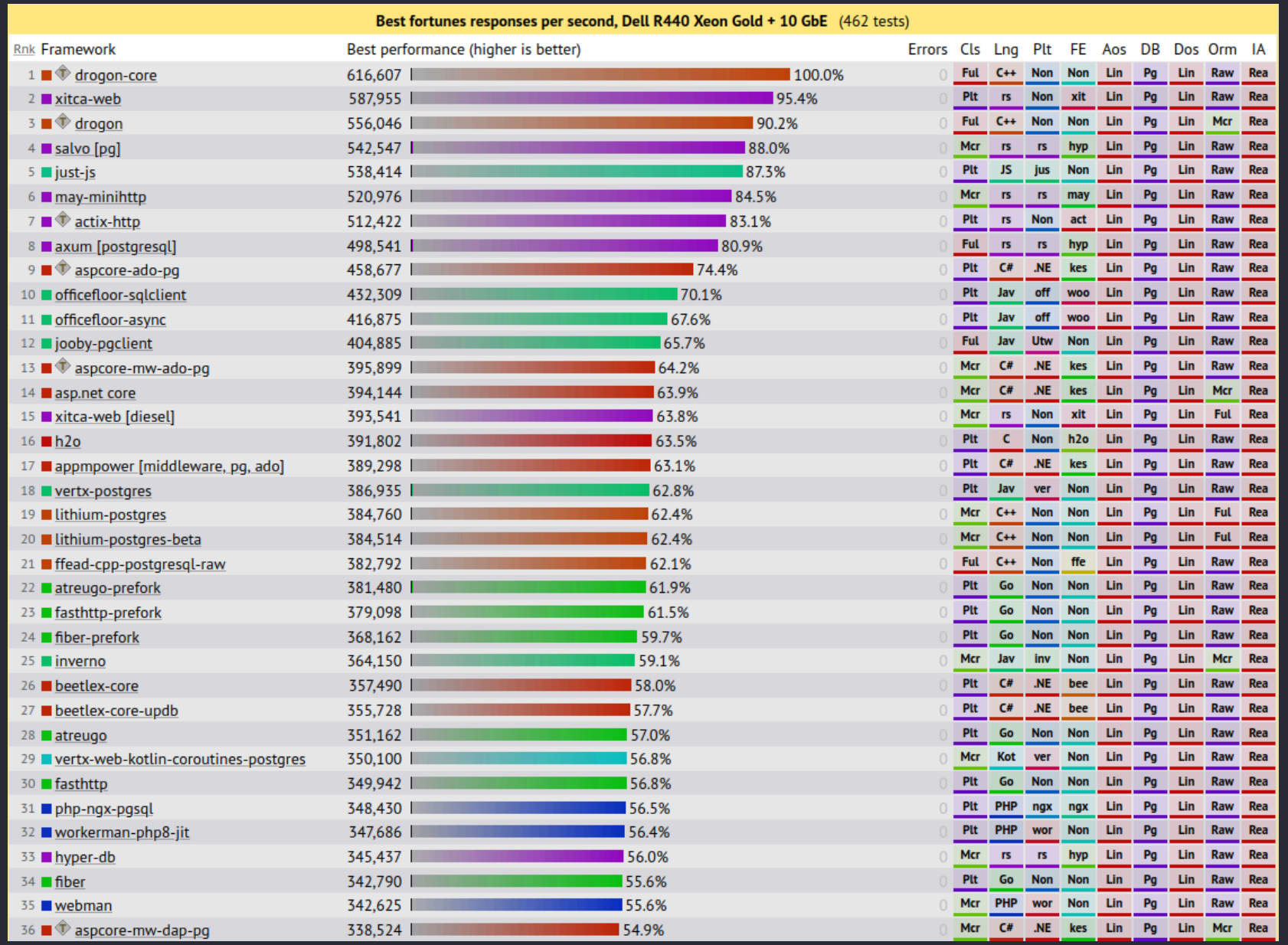
mandelbrot

source	secs	mem	gz	busy	cpu load
Rust	**0.93**	32,676	763	3.70	100% 99% 100% 100%
C gcc	1.27	31,792	1135	5.08	100% 100% 99% 100%

regex-redux

source	secs	mem	gz	busy	cpu load
Rust	**0.77**	147,524	2458	1.99	54% 59% 91% 54%
C gcc	0.80	152,172	1397	2.01	52% 99% 48% 53%

Porovnání frameworků ve Fortunes



Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (462 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	drogon-core	616,607 100.0%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
2	xitca-web	587,955 95.4%	0	Plt	rs	Non	xit	Lin	Pg	Lin	Raw	Rea
3	drogon	556,046 90.2%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
4	salvo [pg]	542,547 88.0%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
5	just-js	538,414 87.3%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
6	may-minihttp	520,976 84.5%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
7	actix-http	512,422 83.1%	0	Plt	rs	Non	act	Lin	Pg	Lin	Raw	Rea
8	axum [postgresql]	498,541 80.9%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
9	aspcore-ado-pg	458,677 74.4%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
10	officefloor-sqlclient	432,309 70.1%	0	Plt	Jav	off	woo	Lin	Pg	Lin	Raw	Rea
11	officefloor-async	416,875 67.6%	0	Plt	Jav	off	woo	Lin	Pg	Lin	Raw	Rea
12	jooby-pgclient	404,885 65.7%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
13	aspcore-mw-ado-pg	395,899 64.2%	0	Mcr	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
14	asp.net core	394,144 63.9%	0	Mcr	C#	.NE	kes	Lin	Pg	Lin	Mcr	Rea
15	xitca-web [diesel]	393,541 63.8%	0	Mcr	rs	Non	xit	Lin	Pg	Lin	Ful	Rea
16	h2o	391,802 63.5%	0	Plt	C	Non	h2o	Lin	Pg	Lin	Raw	Rea
17	appmpower [middleware, pg, ado]	389,298 63.1%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
18	vertx-postgres	386,935 62.8%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
19	lithium-postgres	384,760 62.4%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
20	lithium-postgres-beta	384,514 62.4%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
21	ffead-cpp-postgresql-raw	382,792 62.1%	0	Ful	C++	Non	ffe	Lin	Pg	Lin	Raw	Rea
22	atreugo-prefork	381,480 61.9%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
23	fasthttp-prefork	379,098 61.5%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
24	fiber-prefork	368,162 59.7%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
25	inverno	364,150 59.1%	0	Mcr	Jav	inv	Non	Lin	Pg	Lin	Mcr	Rea
26	beetlex-core	357,490 58.0%	0	Plt	C#	.NE	bee	Lin	Pg	Lin	Raw	Rea
27	beetlex-core-updb	355,728 57.7%	0	Plt	C#	.NE	bee	Lin	Pg	Lin	Raw	Rea
28	atreugo	351,162 57.0%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
29	vertx-web-kotlin-coroutines-postgres	350,100 56.8%	0	Mcr	Kot	ver	Non	Lin	Pg	Lin	Raw	Rea
30	fasthttp	349,942 56.8%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
31	php-ngx-pgsql	348,430 56.5%	0	Plt	PHP	ngx	ngx	Lin	Pg	Lin	Raw	Rea
32	workerman-php8-jit	347,686 56.4%	0	Plt	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
33	hyper-db	345,437 56.0%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
34	fiber	342,790 55.6%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
35	webman	342,625 55.6%	0	Mcr	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
36	aspcore-mw-dap-pg	338,524 54.9%	0	Mcr	C#	.NE	kes	Lin	Pg	Lin	Mcr	Rea

Porovnání frameworků ve Fortunes

331	express-mysql	32,016	5.2%	0	Mcr	JS	njs	Non	Lin	My	Lin	Ful	Rea
332	sinatra-sequel	31,506	5.1%	0	Mcr	Rby	Rac	Pum	Lin	My	Lin	Ful	Rea
333	sinatra-sequel-postgres	31,386	5.1%	0	Mcr	Rby	Rac	Pum	Lin	Pg	Lin	Ful	Rea
334	aspcore-mono-mvc-pg	31,246	5.1%	0	Ful	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
335	starlite	30,918	5.0%	0	Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
336	racket	30,657	5.0%	0	Mcr	rac	rac	rac	Lin	Pg	Lin	Raw	Rea
337	sinatra-sequel-postgres-unicorn-mri	30,615	5.0%	0	Mcr	Rby	Rac	Uni	Lin	Pg	Lin	Ful	Rea
338	slim	30,026	4.9%	0	Mcr	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
339	hot-mysql	29,642	4.8%	67	Plt	Grv	Jty	Jty	Lin	My	Lin	Raw	Rea
340	ktor-jasync	29,479	4.8%	0	Ful	Kot	Non	Non	Lin	Pg	Lin	Raw	Rea
341	php-eloquent	27,869	4.5%	0	Plt	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
342	elixir-plug-ecto	27,308	4.4%	0	Mcr	Eli	bea	cow	Lin	Pg	Lin	Ful	Rea
343	wildfly-ee	26,395	4.3%	0	Ful	Jav	Svt	Wil	Lin	My	Lin	Ful	Rea
344	sinatra-sequel-postgres-passenger-mri	26,352	4.3%	0	Mcr	Rby	Rac	Pas	Lin	Pg	Lin	Ful	Rea
345	sinatra-sequel-passenger-mri	25,930	4.2%	0	Mcr	Rby	Rac	Pas	Lin	My	Lin	Ful	Rea
346	trillium	25,564	4.1%	0	Mcr	rs	Non	Non	Lin	Pg	Lin	Ful	Rea
347	codeigniter	24,904	4.0%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
348	fat-free-raw	24,093	3.9%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
349	warp-rust	23,250	3.8%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
350	revenj	22,891	3.7%	0	Ful	C#	Non	Non	Lin	Pg	Lin	Ful	Rea
351	spring	22,478	3.6%	0	Ful	Jav	tom	Non	Lin	Pg	Lin	Mcr	Rea
352	quart-uvicorn	22,466	3.6%	0	Mcr	Py	Non	uvi	Lin	Pg	Lin	Raw	Rea
353	cppcms-postgres	21,945	3.6%	0	Plt	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
354	spring-mongo	21,771	3.5%	0	Ful	Jav	tom	Non	Lin	Mo	Lin	Ful	Rea
355	falcon	21,743	3.5%	0	Mcr	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
356	falcon [meinheld-orjson]	21,557	3.5%	0	Mcr	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
357	yii2	21,317	3.5%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
358	ninja-standalone	21,185	3.4%	0	Ful	Jav	Jty	Non	Lin	My	Lin	Ful	Rea
359	cppcms	20,916	3.4%	0	Plt	C++	Non	Non	Lin	My	Lin	Raw	Rea
360	spring-jpa	20,768	3.4%	0	Ful	Jav	tom	Non	Lin	Pg	Lin	Ful	Rea
361	phalcon-mongodb	20,602	3.3%	0	Ful	PHP	fpm	ngx	Lin	Mo	Lin	Raw	Rea
362	ktor-jetty	20,539	3.3%	0	Mcr	Kot	Jty	Non	Lin	Pg	Lin	Raw	Rea
363	sinatra-sequel-postgres-torquebox-jruby	20,522	3.3%	0	Mcr	Rby	Rac	Tor	Lin	Pg	Lin	Ful	Rea
364	tornado-py3-uvloop	20,332	3.3%	0	Plt	Py	Non	Tor	Lin	Pg	Lin	Raw	Rea
365	phalcon	20,098	3.3%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
366	sinatra-sequel-torquebox-jruby	19,452	3.2%	0	Mcr	Rby	Rac	Tor	Lin	My	Lin	Ful	Rea
367	go-mgo	19,377	3.1%	0	Plt	Go	Non	Non	Lin	Mo	Lin	Raw	Rea
368	duct-httpkit	18,563	3.0%	0	Mcr	Clj	Rin	Non	Lin	Pg	Lin	Raw	Rea
369	fat-free	18,562	3.0%	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
370	dropwizard-mongodb	18,150	2.9%	0	Ful	Jav	JAX	Jty	Lin	Mo	Lin	Ful	Rea
371	lumen-swoole	17,005	2.8%	0	Mcr	PHP	swo	Non	Lin	My	Lin	Ful	Rea
372	grails	16,993	2.8%	0	Ful	Grv	Svt	Non	Lin	My	Lin	Ful	Rea

Konkurence

Při paralelním programování často dochází k všemožným problémům. Rust díky síle překladače (rozumějte statickým kontrolám), pravidlům a konvencím v jazyce poskytuje podstatně větší jistotu.

Další výhody

- jazyk nemá dědičnost, místo toho vyžaduje kompozici
- neexistuje `null`
- moderní a pokrokový tooling
  - správce závislostí
  - dokumentování kódu
  - testování

Nevýhody Rustu

- Pomalý překlad
- Velké množství knihoven v ekosystému stojí na jedincích
- Často více psaní než v jiných jazycích

Jak se rozhodovat při výběru jazyka?

Potřebuju jednoúčeloový rychle **naprasený tool**? Python.
Bude to běžet v prohlížeči nebo je to **UI**? JS nebo TS.
Můžu použít **managovaný jazyk**? C# > Golang > Kotlin > Java.

Jinak vyberu **Rust**!

Rustu nahrává i lepší udržovatelnost codebase, rychlost nebo nižší chybovost kódu.

K čemu se dnes Rust primárně používá?

- tooly - CLI nebo integrované
- webové aplikace a služby
- knihovny pro jiné jazyky
- nízkoúrovňové aplikace - ovladače, jádro OS, firmware

Příběh Discordu

Příběh Discordu

> As usual with a garbage collected language the problem was CPU stalls due to garbage collection spikes.
> But in non-GC languages you have to worry about memory fragmentation, especially for long-lived processes.
> When you get that sev 1 bug that happens after two months of flawless execution it will often be a memory allocation failure due to memory fragmentation.
>
> So you end up creating your own memory allocator anyway.

Příběh Discordu

> When we started load testing, we were instantly pleased with the results. The latency of the Rust version was just as good as Go's and had no latency spikes!
>
> Remarkably, we had only put very basic thought into optimization as the Rust version was written.
> Even with just basic optimization, Rust was able to outperform the hyper hand-tuned Go version.

Příběh Discordu

> After a bit of profiling and performance optimizations, we were able to beat Go on every single performance metric. Latency, CPU, and memory were all better in the Rust version.

Příběh Discordu

> Along with performance, Rust has many advantages for an engineering team. For example, its type safety and borrow checker make it very easy to refactor code as product requirements change or new learnings about the language are discovered. Also, the ecosystem and tooling are excellent and have a significant amount of momentum behind them.
>
> Also, our business case for using Go - it's all about saving money.

Příběh Discordu

http://highscalability.com/blog/2020/2/7/stuff-the-internet-says-on-scalability-for-february-7th-2020.html

https://blog.discord.com/why-discord-is-switching-from-go-to-rust-a190bbca2b1f

Úvod do Rustu a jeho syntaxe

Verzování Rustu

`nightly`

vydáváno každodenně

`beta`

vydávána jednou za 6 týdnů

`stable`

vydávána jednou za 6 týdnů (následujících po betě)

Problémy verzování

Dříve nebyly všechny features dostupné ve stabilní verzi. Tvůrci frameworků proto sahali po nightly, a ta musela být používaná i v projektu.

Dnes už to neplatí a použití nightly verze se snažíme vyhnout.

Je vhodné zmínit, že Rust se rychle vyvíjí. Je proto nutné hlídat, které vlastnosti jsou deprekovány, a naopak nově přidány.

Nové verze jazyka jsou testovány na dostupných crates!

Instalace Rustu

Instalaci a aktualizaci Rustu doporučujeme provádět přes `rustup`.

UN*X

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Windows

Stáhnout rustup-init.exe a řídit se pokyny instalátoru.

MacOS

Pokud máte homebrew, tak `brew install rustup`.

Základní nástroje

`rustup` – spravuje verze Rustu

`rustc` – překladač

`cargo` – balíčkovací systém, správce projektu a závislostí

`clippy` – linter

IDE

VS Code

s pluginem `rust-analyzer` - neinstalujte plugin jménem "Rust" (starý, deprekovaný, nahrazený)

Jetbrains RustRover

Jetbrains CLion

s pluginem `Intellij Rust`

Založení nového projektu

Příkazem

```
cargo new nazev_projektu
```

se vytvoří nový projekt:

```
nazev_projektu
+-- Cargo.toml
+-- src
|   +--main.rs
```

Překlad a spuštění

```
cargo build
cargo run
```

Cargo.toml

```toml
[package]
name = "hello_world"
version = "0.1.0"
edition = "2021"
authors = ["Your Name <you@example.com>"]

[dependencies]
rand = { git = "https://github.com/rust-lang-nursery/rand.git" }
```

Cargo.lock

```
# This file is automatically @generated by Cargo. It is not intended for manual editing.

[[package]]
name = "hello_world"
version = "0.1.0"
dependencies = [
 "rand",
]

[[package]]
name = "rand"
version = "0.9.0"
source = "git+https://github.com/rust-lang-nursery/rand.git#f3dd0b885c4597b9617ca79987a0dd899ab29fcb"
dependencies = [
 "libc",
 "rand_chacha",
 "rand_core",
]

# ...
```

Složitější struktura projektu

```
+-- Cargo.lock
+-- Cargo.toml
+-- src/
|   +-- lib.rs <---------------------------------- základní soubor knihovny
|   +-- main.rs <--------------------------------- základní spustitelný soubor
|   +-- bin/ <------------------------------------ veškeré další spustitelné soubory
|       +-- named-executable.rs
|       +-- another-executable.rs
|       +-- multi-file-executable/
|           +-- main.rs
|           +-- some_module.rs
+-- benches/ <---------------------------------- benchmarky
|   +-- large-input.rs
|   +-- multi-file-bench/
|       +-- main.rs
|       +-- bench_module.rs
+-- examples/ <--------------------------------- ukázky kódu
|   +-- simple.rs
|   +-- multi-file-example/
|       +-- main.rs
|       +-- ex_module.rs
+-- tests/ <------------------------------------ integrační testy
    +-- some-integration-tests.rs
    +-- multi-file-test/
        +-- main.rs
        +-- test_module.rs
```

Ukázka kódu z main.rs

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

- Všimněte si, že `main` nevrací hodnotu. Později zjistíte, že může vracet `Result`.
  Pro explicitní návratový kód také existuje funkce `std::process::exit(code: i32)`.
- `println!()` je makro. Poznáme ho podle vykřičníku.

Přidání crate pro CLI aplikace

Závislost na crate `clap` přidáme příkazem `cargo add clap`.

```
use std::path::PathBuf;
use clap::{arg, command, value_parser, ArgAction, Command};

fn main() {
    let matches = command!()
        .arg(arg!([name] "Optional name to operate on"))
        .arg(arg!(
            -d --debug ... "Turn debugging information on"
        ))
        .get_matches();

    // Continued on the next slide...
}
```

Kontrola argumentů

```rust
fn main() {
    // ...continued from the previous slide.

    if let Some(name) = matches.get_one::<String>("name") {
        println!("Value for name: {name}");
    }

    match matches
        .get_one::<u8>("debug")
        .expect("Count's are defaulted")
    {
        0 => println!("Debug mode is off"),
        1 => println!("Debug mode is kind of on"),
        2 => println!("Debug mode is on"),
        _ => println!("Don't be crazy"),
    }
}
```

Základní typy

a řízení toku programu

Celočíselné typy

Velikost	Znaménkový	Neznaménkový
8 bitů	i8	u8
16 bitů	i16	u16
32 bitů	i32	u32
64 bitů	i64	u64
128 bitů	i128	u128
dle architektury	isize	usize

Zápisy literálů

Velikost	Příklad
desítkové	98_222
šestnáctkové	0xff
osmičkové	0o77
binární	0b1111_0000
bajtové	b'A'

S plovoucí řádovou čárkou (IEEE-754)

Velikost	Typ
32 bitů	f32
64 bitů	f64

Výchozí je `f64`.

Definice proměnné

```
let number = 42;
let number: i16 = 42i16;

let mut fp_number: f64 = 3.14f64;
```

Boolovské typy

```
fn main() {
    let t = true;

    let f: bool = false;
}
```

Znakové typy (UTF-8)

```
fn main() {
    let c: char = 'z';
    let z = 'ℤ';
    let heart_eyed_cat = '🏞😻';
}
```

Složené typy

Tuple (n-tice)

```
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);

    // lze použít destructuring:
    let (first, second) = (1, 2.0);
}
```

Pole

```
fn main() {
    let a: [i32; 5] = [1, 2, 3, 4, 5];
    let first = a[0];
    let second = a[1];

    let months = ["January", "February", "March", "April", "May", "June", "July",
                  "August", "September", "October", "November", "December"];
}
```

Ovládání toku programu

Klasický if

```
fn main() {
    let number = 3;

    if number < 5 {
        println!("condition was true");
    } else if number % 3 == 0 {
        println!("number is divisible by 3");
    } else {
        println!("condition was false");
    }
}
```

If jako výraz

```
fn main() {
    let condition = true;
    let number = if condition { 5 } else { 6 };

    println!("The value of number is: {}", number);
}
```

If let

```
let number = Some(7);
let letter: Option<i32> = None;
let emoticon: Option<i32> = None;

if let Some(i) = number {
    println!("Matched {:?}!", i);
}
```

Match

```
fn main() {
    let some_u8_value = 0u8;

    match some_u8_value {
        1 => println!("one"),
        3 => println!("three"),
        5 => println!("five"),
        7 => println!("seven"),
        _ => (),
    }
}
```

Nekonečná smyčka

```
fn main() {
    loop {
        println!("again!");
    }
}
```

While

```
fn main() {
    let mut number = 3;

    while number != 0 {
        println!("{}!", number);

        number -= 1;
    }

    println!("LIFTOFF!!!");
}
```

For

```
fn main() {
    for n in 1..=100 {
        if n % 2 == 0 {
            println!("even");
        } else {
            println!("odd");
        }
    }
}
```

To je pro dnešek vše.

Dotazy?

Děkuji za pozornost