Rust는 지구를 구할 수 있을까요?

October 13, 2023

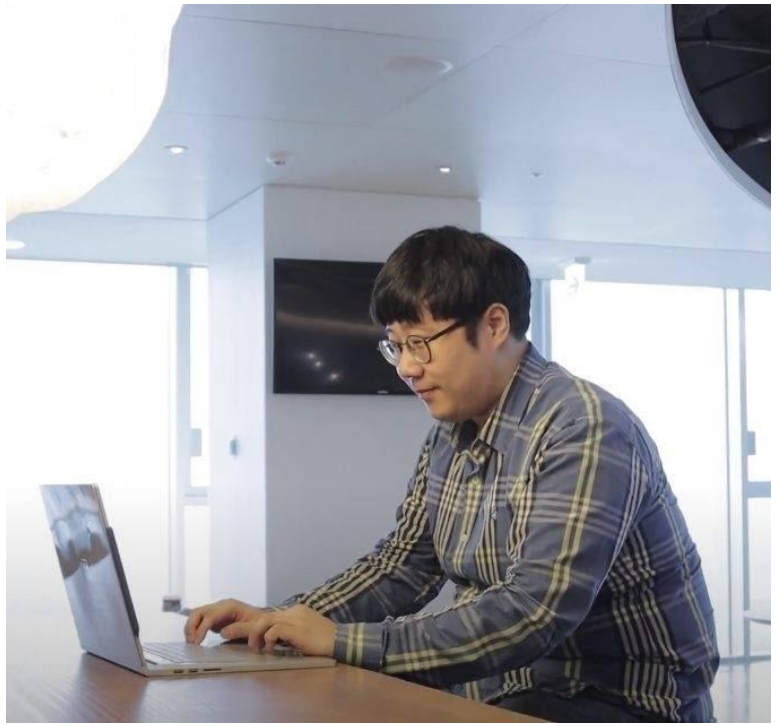
CLOUD NATIVE
COMPUTING FOUNDATION

발표자 소개

- 옥찬호 (Chris Ohk)
  - (현) Momenti Engine Engineer
  - (전) Nexon Korea Game Programmer
  - Microsoft Developer Technologies MVP
  - C++ Korea Founder & Administrator
  - Reinforcement Learning KR Administrator

utilForever@gmail.com

utilForever

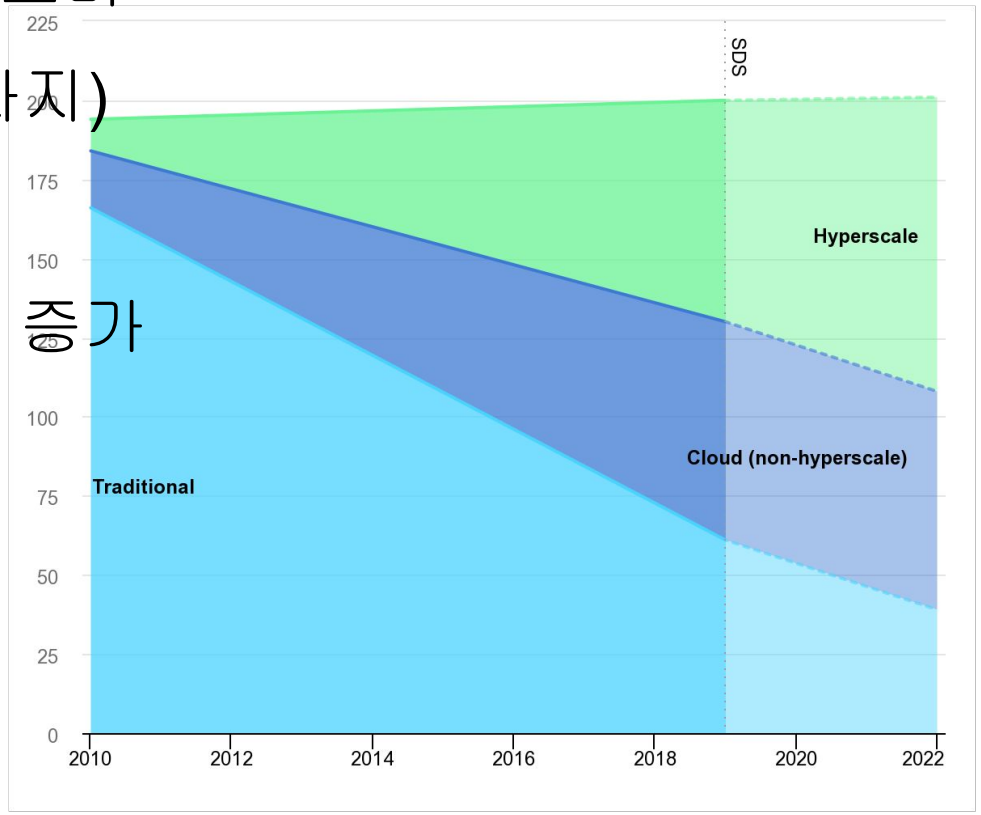


목차

- 클라우드에서의 에너지 사용
- 프로그래밍 언어의 에너지 효율성
- Rust 언어 사용 사례
- Rust 언어 소개
- Rust는 지구를 구할 수 있을까요?

클라우드에서의 에너지 사용

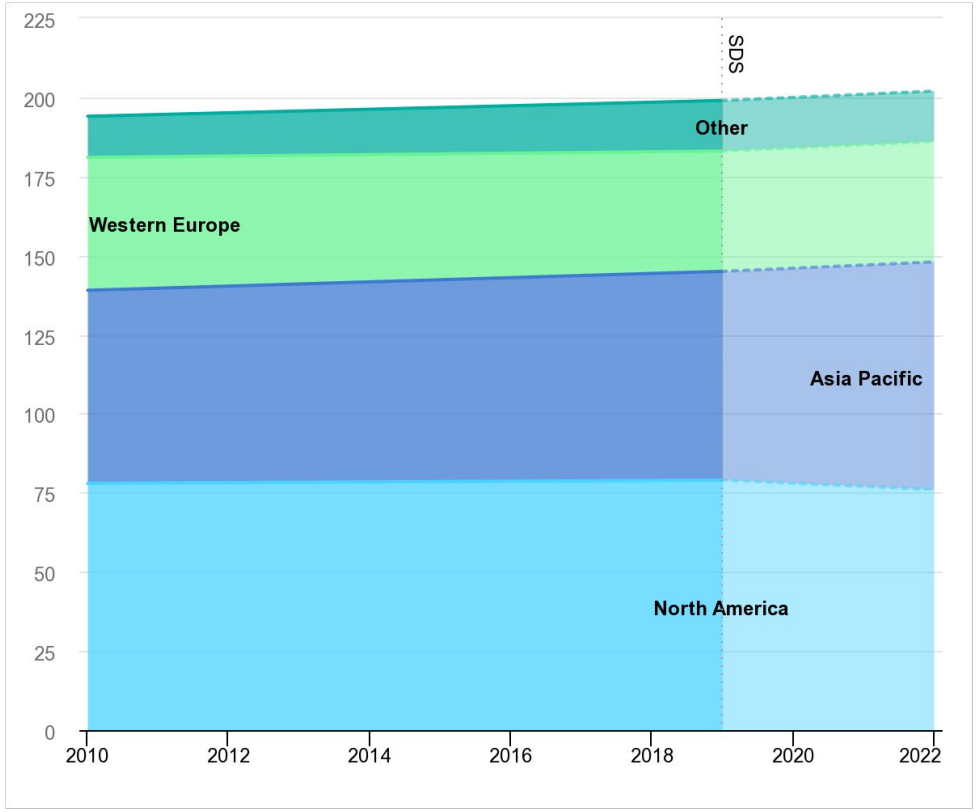
- 전세계적으로 연간 약 200테라와트의 시간을 소비
  (지구에서 소비되는 모든 에너지의 약 1%를 차지)
- 기존 데이터 센터의 비율은 줄고,
  클라우드/하이퍼스케일 데이터 센터의 비율은 증가
- 하지만 전체 에너지 사용량은 거의 비슷
- 이는 클라우드/하이퍼스케일 데이터 센터의
  에너지 효율성 개선을 꾸준히 한 결과



IEA, Global data centre energy demand by region, 2010-2022, IEA, Paris https://www.iea.org/data-and-statistics/charts/global-data-centre-energy-demand-by-region-2010-2022, IEA. Licence: CC BY 4.0

클라우드에서의 에너지 사용

- 대륙별로 살펴보자면, 아시아 태평양에서의 에너지 사용량이 꾸준히 증가하는 추세



IEA, Global data centre energy demand by region, 2010-2022, IEA, Paris https://www.iea.org/data-and-statistics/charts/global-data-centre-energy-demand-by-region-2010-2022, IEA. Licence: CC BY 4.0

프로그래밍 언어의 에너지 효율성

- 여러 프로그래밍 언어들의 에너지 소비,
  성능, 메모리 사용 사이의 상관관계를 조사
- C와 Rust가 다른 언어보다 더 효율적이라는 결과

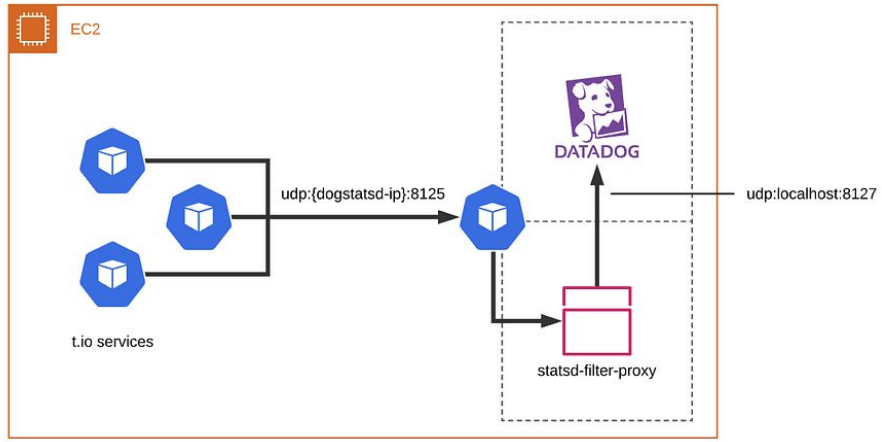
Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Pereira, Rui, et al. "Energy efficiency across programming languages: how do energy, time, and memory relate?." Proceedings of the 10th ACM SIGPLAN international conference on software language engineering. 2017.

Rust 언어 사용 사례

- Tenable.io
  - Datadog의 커스텀 메트릭을 사용하기 위해 수백만 개의 지표를 전송
  - 플랫폼이 성장함에 따라 레거시 앱에서 전송한 수많은 메트릭 더 이상 사용되지 않는다는 사실을 발견
  - 이 문제를 해결하기 위해 불필요한 측정 항목을 필터링하기로
  - 필터는 Node.js로 작성되었는데, 처음에는 괜찮았지만 시간이 지나면서 성능 문제가 발생하기 시작
  - 성능 지표를 분석하고 필터를 보다 효율적인 언어, 즉 Rust로 다시 작성하기로 결정

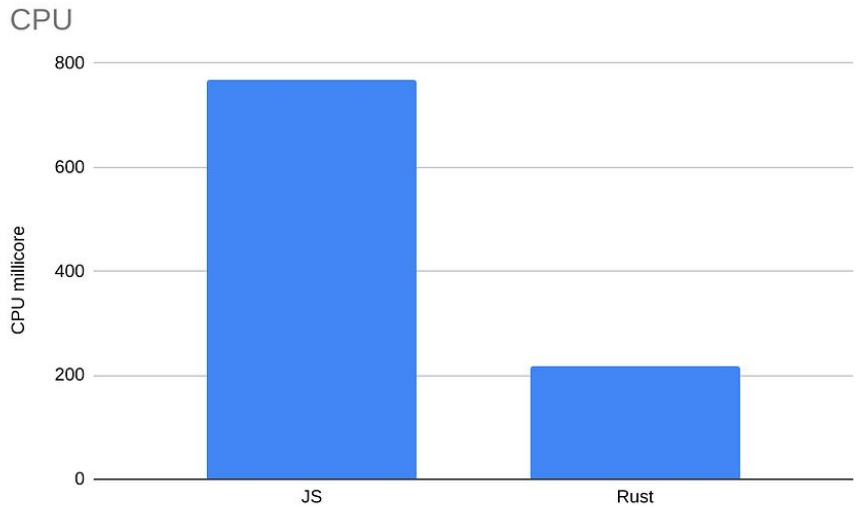


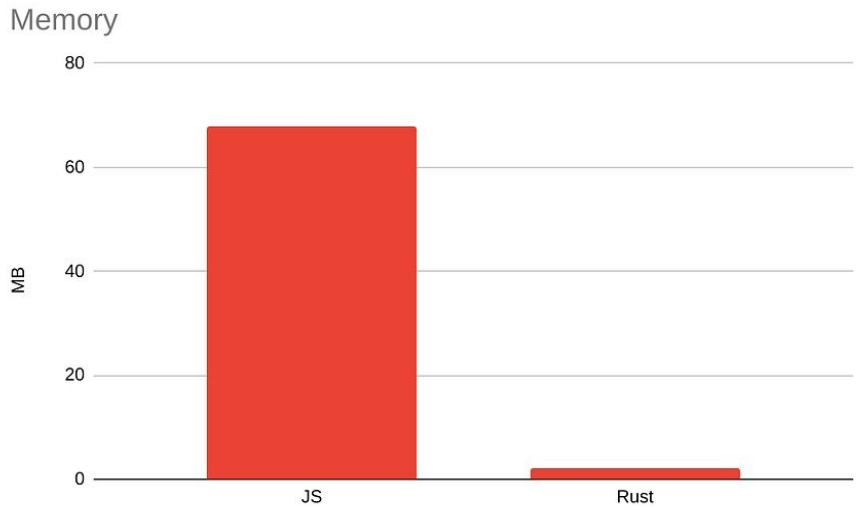
https://medium.com/tenable-techblog/optimizing-700-cpus-away-with-rust-dc7a000dbdb2

Rust 언어 사용 사례

- Tenable.io
  - 그 결과, 프로덕션 환경에서 CPU 사용량이 75%, 메모리 사용량이 95% 감소

    (CPU : 800ms -> 200ms, Memory : 70MB -> 5MB)

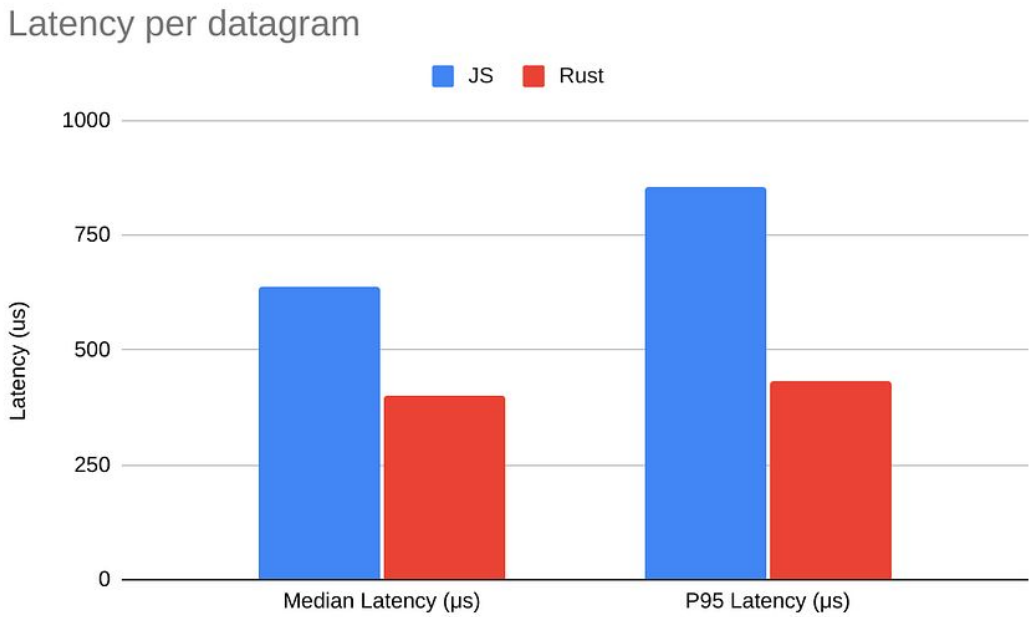




https://medium.com/tenable-techblog/optimizing-700-cpus-away-with-rust-dc7a000dbdb2

Rust 언어 사용 사례

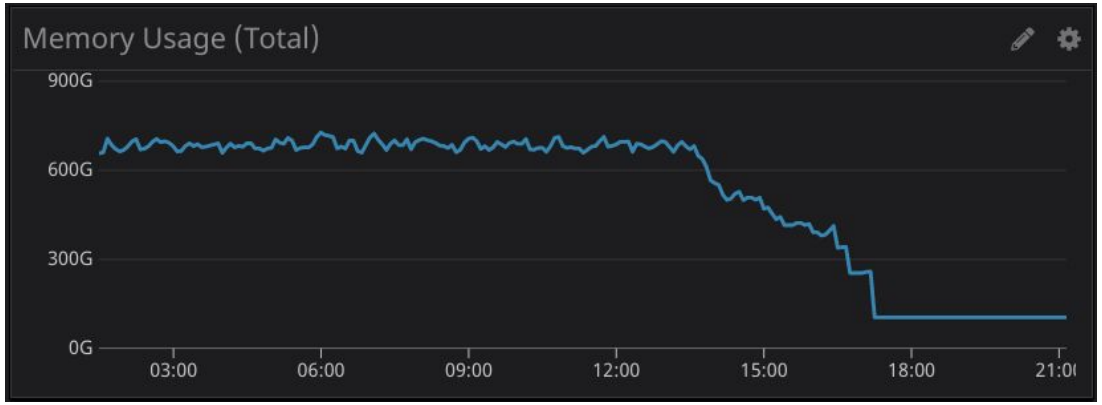
- Tenable.io
  - 또한 패킷 당 대기 시간이 **50%** 이상 감소



https://medium.com/tenable-techblog/optimizing-700-cpus-away-with-rust-dc7a000dbdb2

Rust 언어 사용 사례

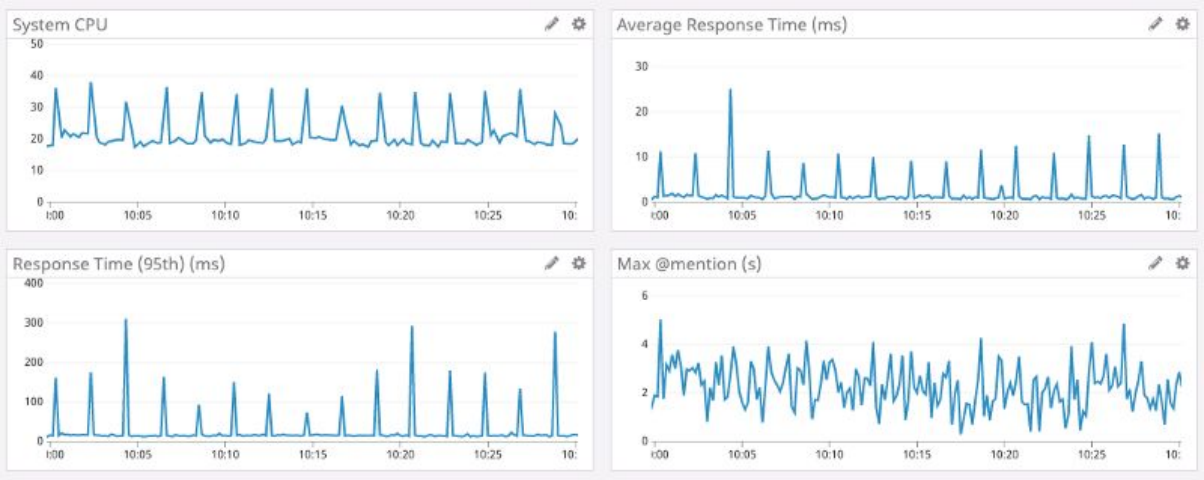
- Tenable.io
  - 새로운 필터가 배포된 후 CPU/메모리 사용량이 급격히 감소되었음을 확인



https://medium.com/tenable-techblog/optimizing-700-cpus-away-with-rust-dc7a000dbdb2

Rust 언어 사용 사례

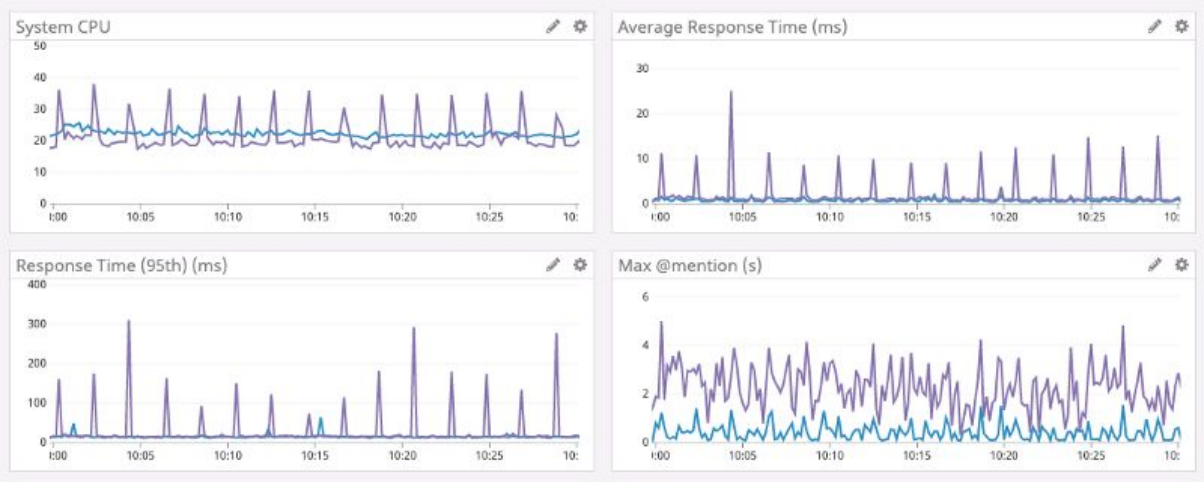
- Discord
  - 어떤 채널과 메시지를 읽었는지 추적하는 서비스 “Read States”
  - 디스코드에 연결할 때, 메시지가 전송될 때, 메시지를 읽을 때 접근
  - Go로 만들었으나 2분마다 GC가 실행되어 스파이크가 발생하는 문제 → 성능 저하



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

Rust 언어 사용 사례

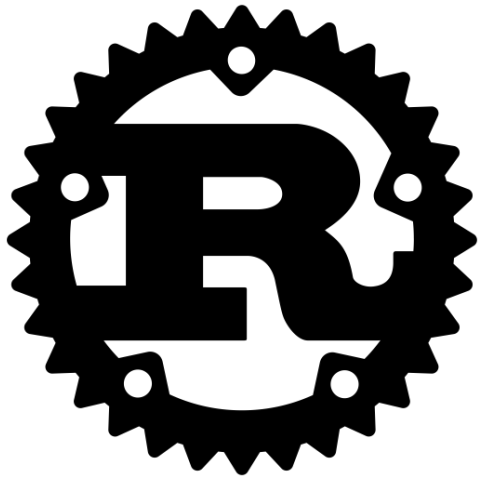
- Discord
  - Rust로 재구현한 결과 성능이 개선됨 (Go는 보라색, Rust는 파란색)
  - 측정 단위가 달라짐 (평균 응답 시간 : $ms \rightarrow \mu s$, 최대 멘션 : $s \rightarrow ms$)



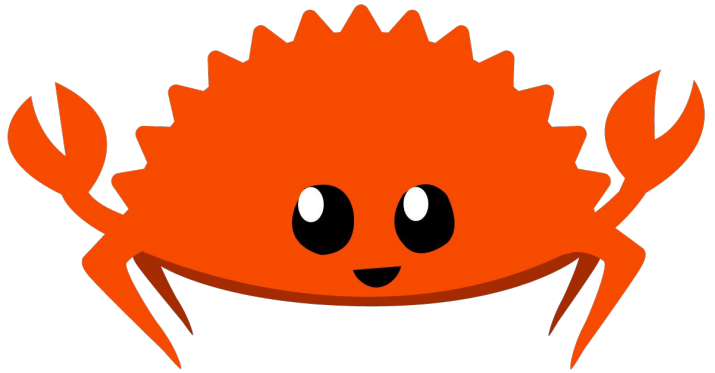
https://discord.com/blog/why-discord-is-switching-from-go-to-rust

Rust 언어란

- https://www.rust-lang.org
- 모질라 재단에서 2010년 7월 7일 처음 발표
- 현재는 러스트 재단으로 독립한 뒤 여러 기업들의 후원을 받아 개발하고 있다.



Rust 공식 로고



Rust 비공식 마스코트 “Ferris”

Rust 언어의 특징

- 안전한 메모리 관리
- 철저한 예외와 오류 관리
- 특이한 열거 시스템
- 트레잇
- 하이지닉 매크로
- 비동기 프로그래밍
- 제네릭

Rust 언어의 핵심, 소유권과 빌림 검사

- Rust에서는 자료형을 인스턴스화해 변수명에 할당(Binding)하면,
  Rust 컴파일러가 전체 생명 주기(Lifetime) 동안 검증할 메모리 리소스를
  생성한다.
- 할당된 변수를 리소스의 소유자(Owner)라고 한다.
- Rust는 범위(Scope)가 끝나는 곳에서 리소스를 소멸하고 할당 해제한다.
  이 소멸과 할당 해제를 의미하는 용어로 drop을 사용한다.
  (C++에서는 Resource Acquisition Is Initialization(RAII)라고 부른다.)

Rust 언어의 핵심, 소유권과 빌림 검사

- 소유자가 함수의 인자로 전달되면, 소유권은 그 함수의 매개 변수로 이동(Move)된다.
- 이동된 이후에는 원래 함수에 있던 변수를 더 이상 사용할 수 없다.

```rust
struct Foo {
    x: i32,
}

fn do_something(f: Foo) {
    println!("{}", f.x);
}

fn main() {
    let foo = Foo { x: 42 };
    do_something(foo);
}
```


Rust 언어의 핵심, 소유권과 빌림 검사

- & 연산자를 통해 참조로 리소스에 대한 접근 권한을 대여할 수 있다.
- 참조도 다른 리소스와 마찬가지로 Drop된다.

```rust
struct Foo {
    x: i32,
}

fn main() {
    let foo = Foo { x: 42 };
    let f = &foo;
    println!("{}", f.x);
}
```


Rust 언어의 핵심, 소유권과 빌림 검사

- &mut 연산자를 통해 리소스에 대해 변경 가능한 접근 권한도 대여할 수 있다.
- 리소스의 소유자는 변경 가능하게 대여된 상태에서 이동되거나 변경될 수 없다.

```rust
struct Foo {
    x: i32,
}

fn do_something(f: Foo) {
    println!("{}", f.x);
}

fn main() {
    let foo = Foo { x: 42 };
    let f = &mut foo;

    // do_something(foo);
    // foo.x = 13;

    f.x = 13;
    println!("{}", foo.x);

    foo.x = 7;
    do_something(foo);
}
```


Rust 언어의 핵심, 소유권과 빌림 검사

- Rust의 참조 규칙
  - 단 하나의 변경 가능한 참조 또는 여러개의 변경 불가능한 참조만 허용하며, 둘 다는 안된다.
  - 참조는 그 소유자보다 더 오래 살 수 없다.

```rust
struct Foo {
    x: i32,
}

fn do_something(f: &mut Foo) {
    f.x += 1;
}

fn main() {
    let foo = Foo { x: 42 };
    do_something(&mut foo);
    do_something(&mut foo);
}
```


쉽게 적용할 수 있는 동시성

```rust
fn sum_odd(nums: &[i64]) -> i64 {
    nums.iter()
        .filter(|val| *val % 2 != 0)
        .sum()
}
```

싱글 스레드
버전

```rust
use rayon::prelude::*;

fn sum_odd(nums: &[i64]) -> i64 {
    nums.par_iter()
        .filter(|val| *val % 2 != 0)
        .sum()
}
```

멀티 스레드
버전

Rust는 지구를 구할 수 있을까요?

Linux 6.1 Officially Adds Support for Rust in the Kernel

LIKE DISCUSS

DEC 20, 2022 • 1 MIN READ

by

Sergio De Simone FOLLOW

After over two years in development, support for using Rust for kernel development has entered a stable Linux release, Linux 6.1, which became available a couple of weeks ago.

Previous to its official release, Rust support has been available in linux-next, the git tree resulting from merging all of the developers and maintainers trees, for over a year. With the stable release, Rust has become the second language officially accepted for Linux kernel development, along with C.

Initial Rust support is just the absolute minimum to get Rust code building in the kernel, say Rust for Linux maintainers. This possibly means that Rust support is not ready yet for prime-time development and that a number of changes at the infrastructure level are to be expected in coming releases. Still, there has been quite some work work going on on a few actual drivers that should become available in the next future. These include a Rust nvme driver, a 9p server, and Apple Silicon GPU drivers.

Rust for Linux is only available on the architectures supported by LLVM/Clang (https://github.com/Rust for Linux/linux/blob/rust/Documentation/rust/arch-support.rst), which is required to compile Rust. Thus, LLVM/Clang must be used to build the whole Linux kernel instead of the more traditional GNU toolchain. This limits supported architecture to a handful, including arm, arm64, x86, powerps, mips, and others. For detailed instructions about building Linux with the appropriate flag for each supported platform, check the official documentation (https://github.com/Rust for Linux/linux/blob/rust/Documentation/kbuild/llvm.rst).

https://www.infoq.com/news/2022/12/linux-6-1-rust/

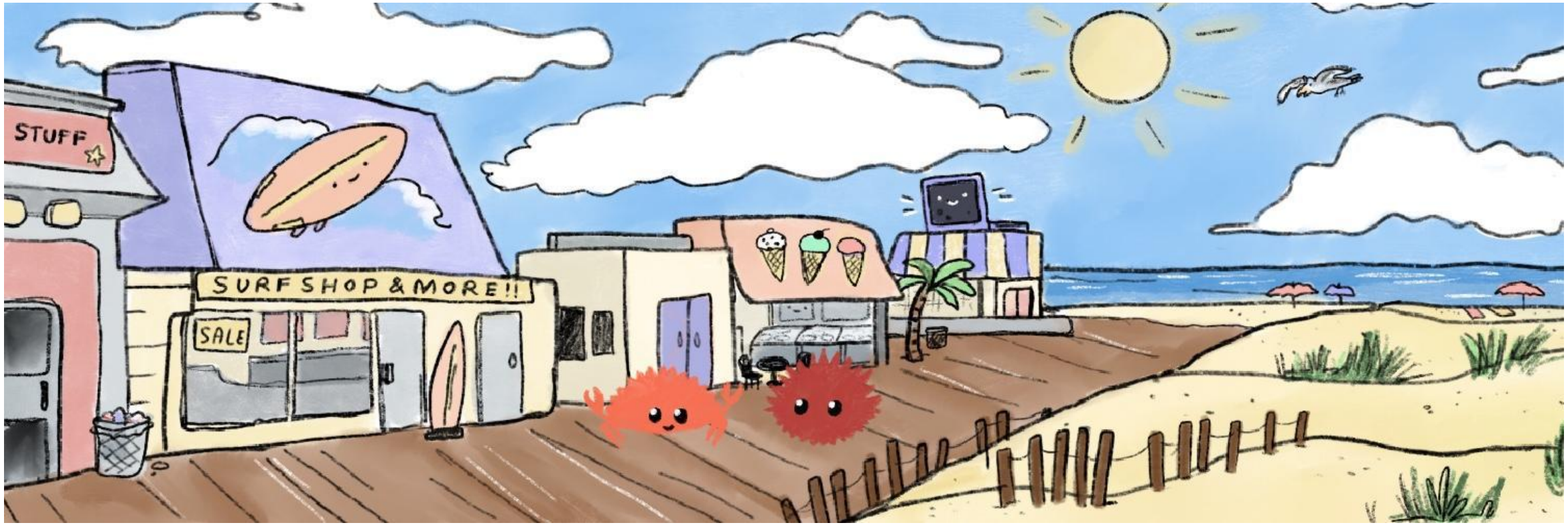
Rust는 지구를 구할 수 있을까요?

https://twitter.com/markrussinovich/status/1656416376125538304

Rust는 지구를 구할 수 있을까요?

- 하드웨어 성능의 발전으로 예전에 비해 소프트웨어의 성능은 조금씩 덜 중요해졌다.
- 하지만 여전히 성능이 중요한 곳이 있다.
  - 임베디드 시스템처럼 하드웨어 성능의 제약이 있는 분야
  - 대규모 시스템처럼 약간의 성능 개선이 많은 변화를 일으킬 수 있는 분야
- 클라우드 분야에서의 성능 개선은 곧 에너지 절약과 비용 절감으로 이어진다. 지구 환경도 지키고 회사에게도 도움을 줄 수 있는 일석이조의 방법이다.
- 또한 안전성도 갖출 수 있다면 더할나위 없이 좋을 것이다. 그렇다면 Rust를 시작할 때다!

Rust는 지구를 구할 수 있을까요?



https://twitter.com/_httprs/status/1394248700424302600

Rust를 공부하고 싶다면?

- 한국어로 번역된 책이 몇 권 있다.

Rust를 공부하고 싶다면?

- 도움이 될 만한 문서들
  - Rust 프로그래밍 언어 공식 문서 : https://doc.rust-lang.org/book/
  - Rust 표준 라이브러리 공식 문서 : https://doc.rust-lang.org/std/index.html
  - Comprehensive Rust by Google : https://google.github.io/comprehensive-rust/ko/
  - Rustlings : https://github.com/rust-lang/rustlings/
  - Rust by Example : https://doc.rust-lang.org/stable/rust-by-example/
  - Rust-101 by Ralf Jung : https://www.ralfj.de/projects/rust-101/main.html
  - Exercism - Rust : https://exercism.org/tracks/rust

감사합니다.

utilForever@gmail.com
https://github.com/utilForever