Are you a Polyglot Developer?

What I've learn using 10 different Programming Languages

2023, Jaehong Jung

Jaehong Jung

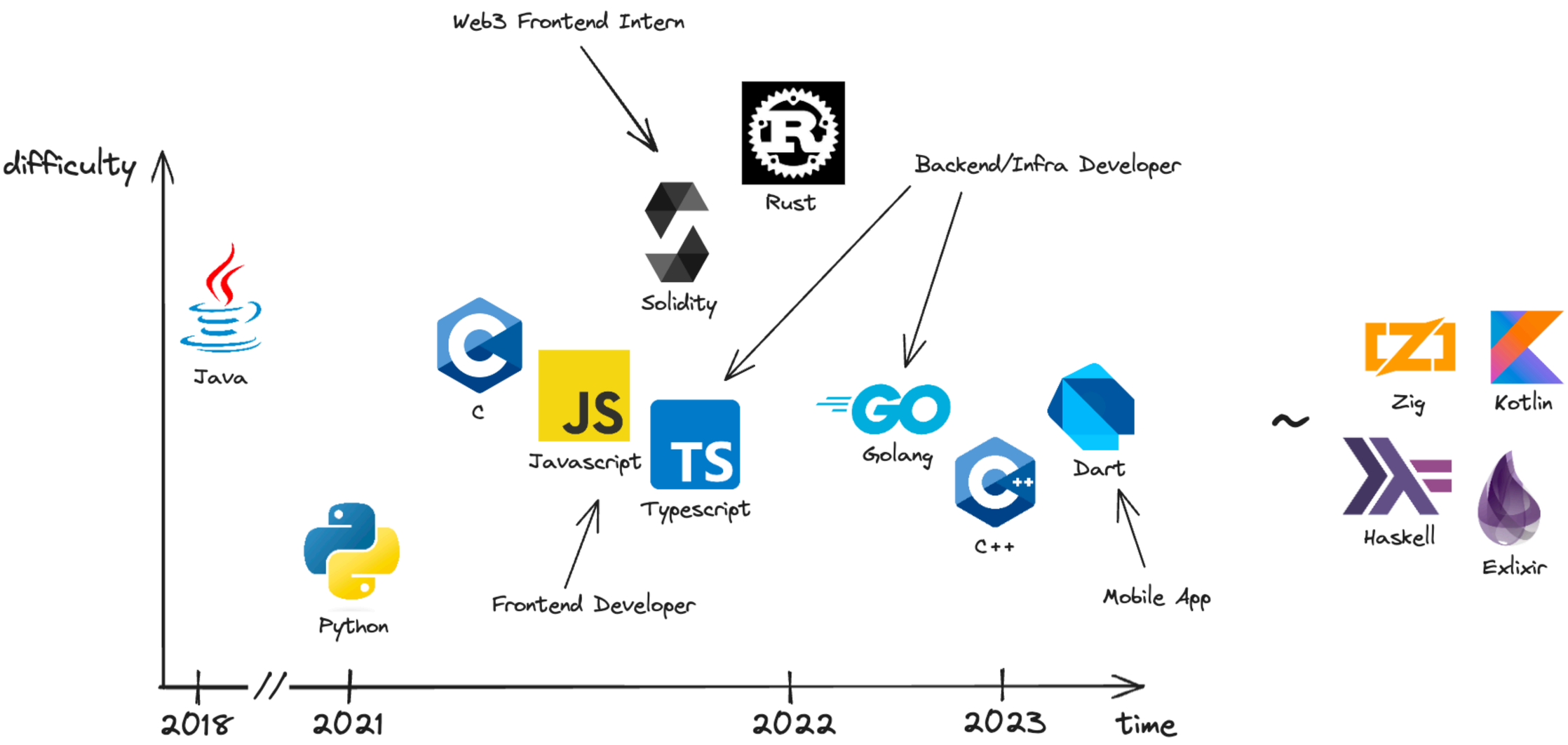
21 GIST

- Blockchain Developer Intern (2021)
  PiLab Tech, Bifrost Network
- Tech Lead (2021.09 ~ 2023.06)
  (주) 시고르자브종 from GIST
- Backend/Cloud Engineer (2023.04 ~)
  Skrr, 2nd place at App Store at peak
- Research Intern (2023.06 ~)
  KAIST NetS&P, Research on Bitcoin and SCION Network

- Leader, WING (2021~2023.06)
- Mentor, GDSC X GIST (2022 ~ 2023)
- GSA-Infoteam (2021 ~)

“What is a Polyglot Programmer And Why You Should Become One”

What I've gone through



1. Purpose of this Language
2. Characteristics and Syntax
3. Documentation and Ecosystem
4. Limits and Weaknesses
5. Real-world projects

Python

Lets you **work quickly** and **integrate systems** more effectively

Purpose of this Language

Python

by Guido van Rossum (1989)

Executable Pseudocode

1. Readability and Simplicity
2. General-Purpose
3. Rapid Development
4. Integration
5. Extensibility
6. Community Support

Python

```python
def factorial(x):
    return 1 if x == 0 else x * factorial(x - 1)
```

C++

```cpp
int factorial(int x) {
    return (x == 0) ? 1 : x * factorial(x - 1);
}
```

PEP 20

The Zen of Python

```
Last login: Tue Aug 15 14:38:35 on ttys000
~ python3
Python 3.11.2 (main, Apr 24 2023, 17:12:56) [Clang 14.0.0 (clang-1400.0.29.202)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Characteristics and Syntax

Python

```python
def triangle_area(width, height):
    return width * height / 2

area = triangle_area(30, 10)
print(area)
```

```python
number = 358

rem = rev = 0
while number >= 1:
    rem = number % 10
    rev = rev * 10 + rem
    number = number // 10

print(rev)
```

```python
class Person:
    eyes = 2
    nose = 1
    mouth = 1

    def eat(self):
        print('yum...')
    def sleep(self):
        print('zzz...')
    def talk(self):
        print('blah...')

class Student(Person):
    def study(self):
        print('write...')
```

Documentation and Ecosystem

Python

python™

pip
Installation

Poetry

NumPy

pandas

python™
virtualenv

TensorFlow

ANACONDA®

PyTorch

django

Selenium

Flask
web development,
one drop at a time

FastAPI

“Limits and Weaknesses”

Python

“Slow and Unsafe”

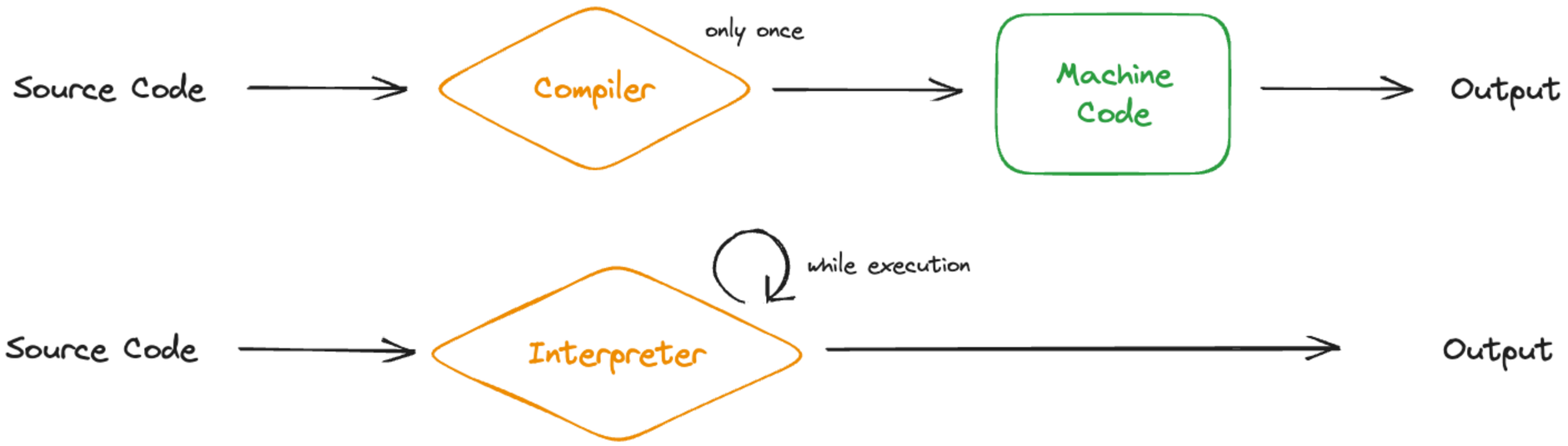
Limits and Weaknesses: Slow

Python

1. **Dynamically Typed** vs Statically Typed
2. **Interpreted Language** vs Compiled Language
3. **Garbage Collection**
4. **GIL** (Global Interpretor Lock)

Interpreter vs Compiler

Python



Garbage Collection

Python

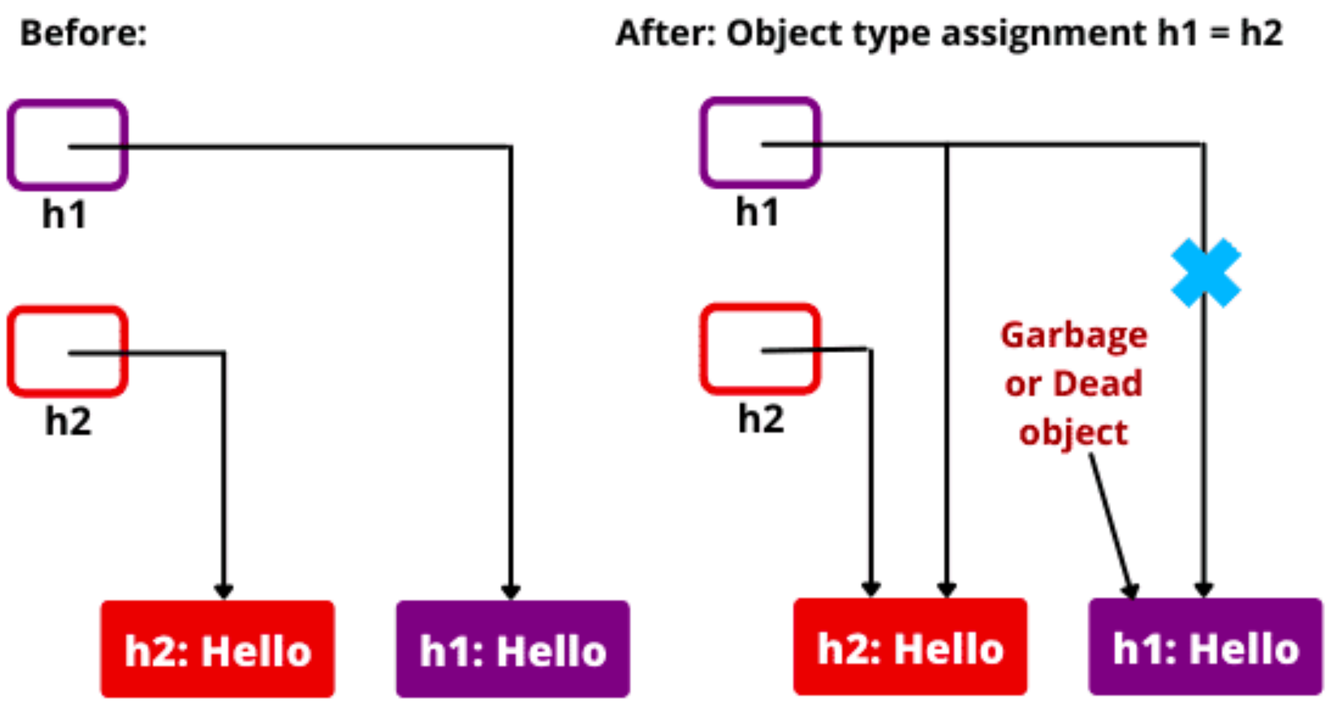


Fig: Reference variable h2 is copied in h1

Garbage Collection

Python

```cpp
int main() {
    // Create a dynamic instance of MyClass
    MyClass* obj1 = new MyClass("Object 1");

    // Assign obj1 to another pointer
    MyClass* obj2 = obj1;

    // Delete obj1
    delete obj1;

    // Clean up memory by deleting obj2
    delete obj2;

    return 0;
}
```

```python
class MyClass:
    def __init__(self, name):
        self.name = name

# Create an instance of MyClass
obj1 = MyClass("Object 1")

# Assign obj1 to another variable
obj2 = obj1

# Delete obj1
del obj1
```

Limits and Weaknesses: Unsafe

Python

Not Type-strict

Changing Variable Types

```python
x = 5          # x is an integer
print(type(x))

x = "hello"    # Now x is a string
print(type(x))
```

Lists with Mixed Types

```python
mixed_list = [1, "two", 3.0, [4, 5], {"six": 6}]
for item in mixed_list:
    print(type(item))
```

Limits and Weaknesses: Unsafe

Python

Not Type-strict

Function Arguments

```python
def add(a, b):
    return a + b

print(add(5, 3))        # 8
print(add("hi", "bye")) # "hibye"
```

Returning Different Types

```python
def get_value(flag):
    if flag:
        return "hello"
    else:
        return 12345

print(type(get_value(True)))  # <class 'str'>
print(type(get_value(False))) # <class 'int'>
```

Real-world Projects

Python

1. Web Crawler
2. CNN, RNN, Seq2seq, GAN, Autoencoder
3. Backend API Server (Django, FastAPI)
4. Simple script

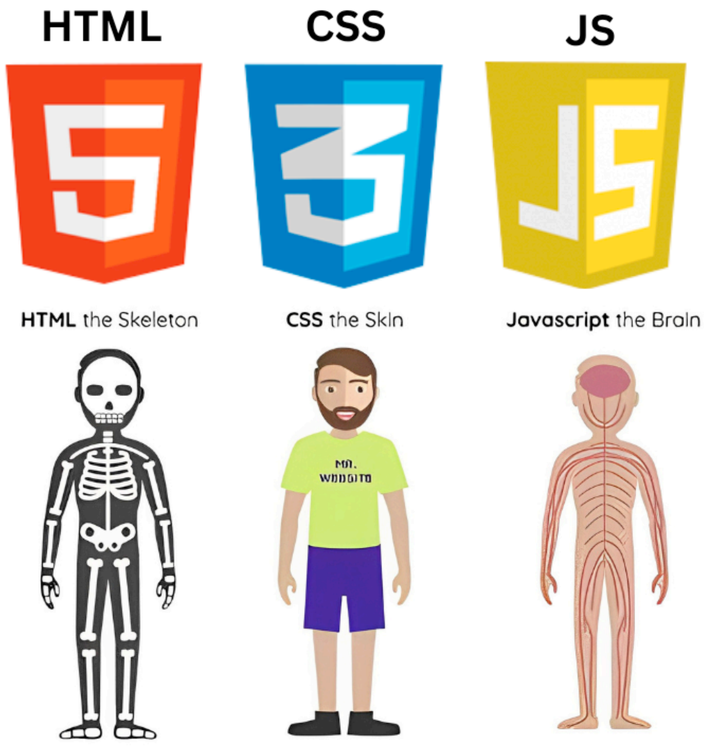
Javascript

Lightweight interpreted programming language with **first-class functions**.
Most well-known as the scripting language **for Web pages**.

Purpose of this Language

Javascript

by Brendan Eich (1995)

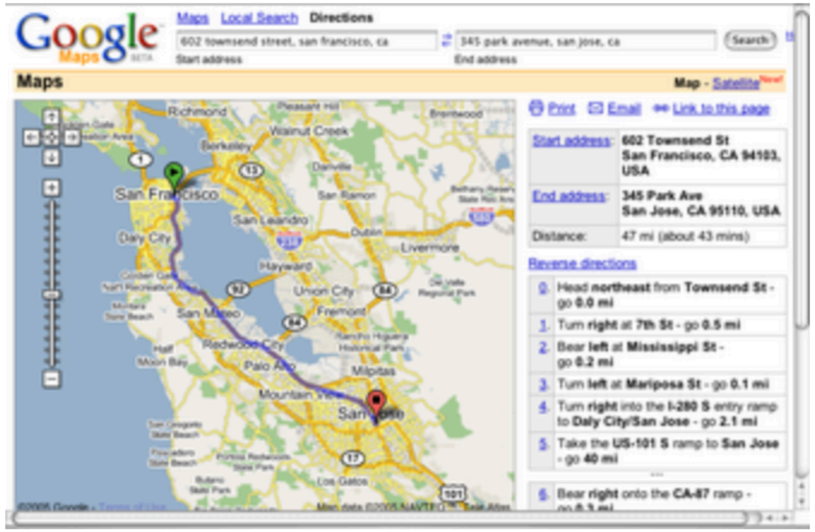


버전	출시년도	특징
ES1	1997	초판
ES2	1998	ISO/IEC 16262 국제 표준과 동일한 규격을 적용
ES3	1999	정규 표현식, try...catch 예외 처리
ES5	2009	HTML5와 함께 출현한 표준안. JSON, strict mode, 접근자 프로퍼티(getter, setter), 향상된 배열 조작 기능(forEach, map, filter, reduce, some, every)
ES6 (ECMAScript 2015)	2015	let, const, class, 화살표 함수, 템플릿 리터럴, 디스트럭처링 할당, spread 문법, rest 파라미터, Symbol, Promise, Map/Set, iterator/generator, module import/export
ES7 (ECMAScript 2016)	2016	지수(**) 연산자, Array.prototype.includes, String.prototype.includes
ES8 (ECMAScript 2017)	2017	async/await, Object 정적 메소드(Object.values, Object.entries, Object.getOwnPropertyDescriptors)
ES9 (ECMAScript 2018)	2018	Object Rest/Spread 프로퍼티

Purpose of this Language

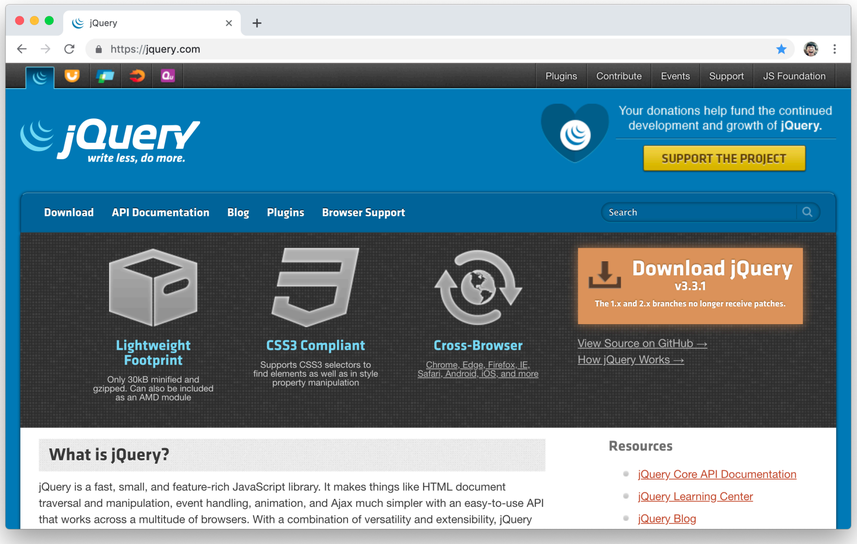
Javascript

AJAX (2005)



Google Maps Beta

jQuery (2006)



Purpose of this Language

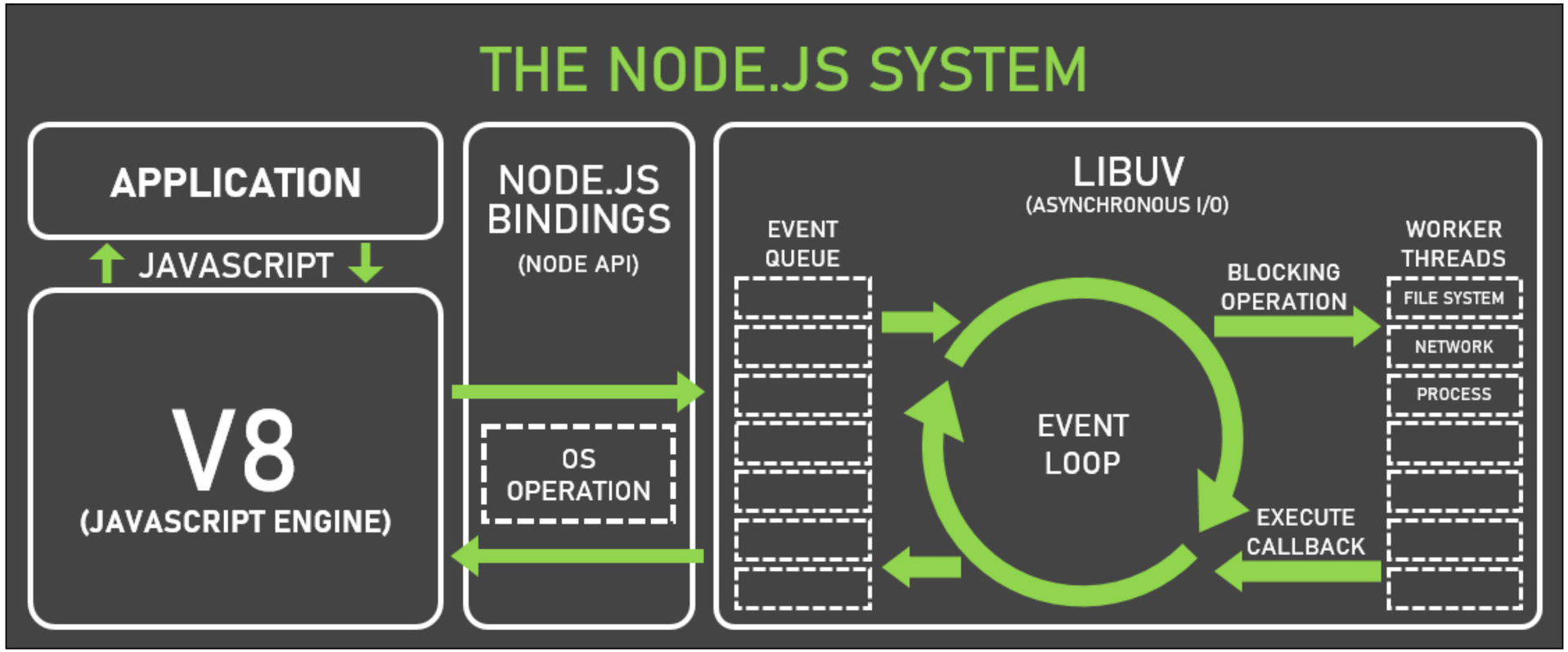
Javascript

V8 Engine (2008)

Node.js (2009)

Characteristics and Syntax

Javascript



Characteristics and Syntax

Javascript

```javascript
function greet(name) {
    return "Hello, " + name + "!";
}

let message = greet("Alice");

if (x > y) {
    console.log("x is greater than y");
} else if (x < y) {
    console.log("x is less than y");
} else {
    console.log("x is equal to y");
}
```

```javascript
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
    greet: function() {
        return "Hello, " + this.firstName + " " + this.lastName;
    }
};

console.log(person.greet());

let fruits = ["apple", "banana", "cherry"];
fruits.push("date"); // Adds "date" to the end
let firstFruit = fruits[0]; // "apple"
```

Characteristics and Syntax

Javascript

1. **First-Class Functions**
2. **Event-Driven** & **Single-Threaded**
3. Object-oriented
4. Dynamic Typed & Interpreted Language
5. Garbage Collection

Documentation and Ecosystem

Javascript

webpack

npm

yarn

Deno

recoil

tailwindcss

React Query

fastify

express

koa

next generation web framework for node.js

nest

Vue.js

React

Remix

NEXT.JS

Limits and Weaknesses

Javascript

1. **Global Variables**
2. **Type Coercion**
3. Inconsistencies **across Browsers**
4. **Single-Threaded**
5. **undefined**

Limits and Weaknesses: Global Variables

Javascript

```javascript
function setGlobalVar() {
    globalVar = "I'm a global variable";
}

setGlobalVar();
console.log(globalVar);  // Outputs: "I'm a global variable"
```

Limits and Weaknesses: undefined

Javascript

Non-zero value

0

null

undefined

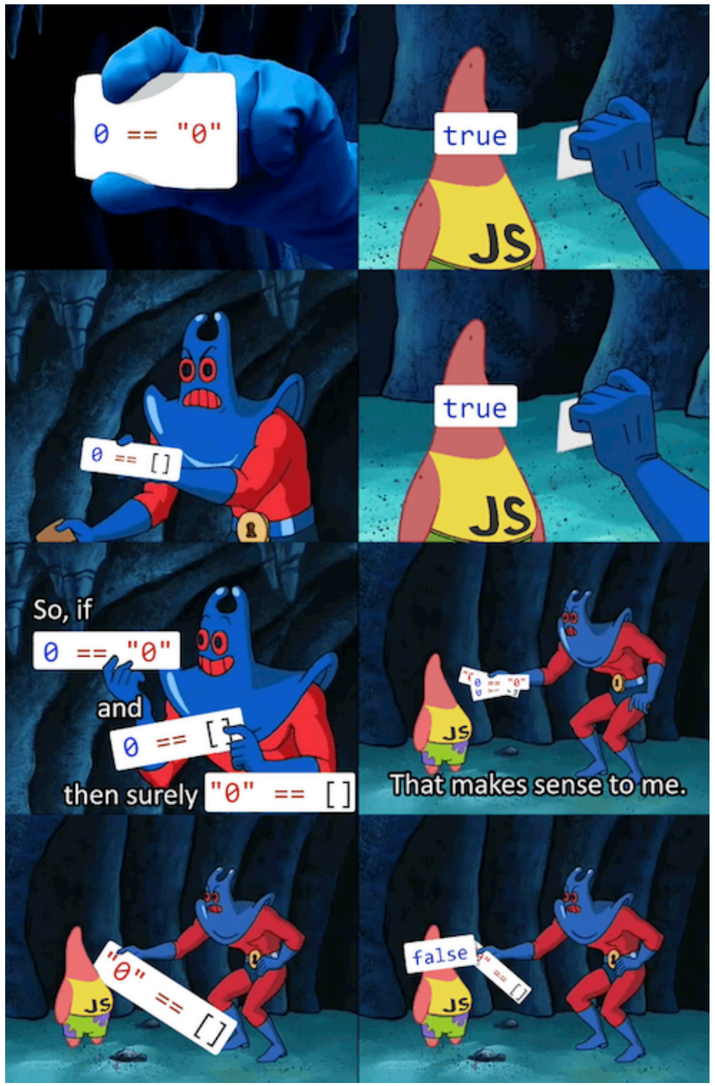
Limits and Weaknesses: Type Coercion

Javascript

Console | What's New

top | Filter

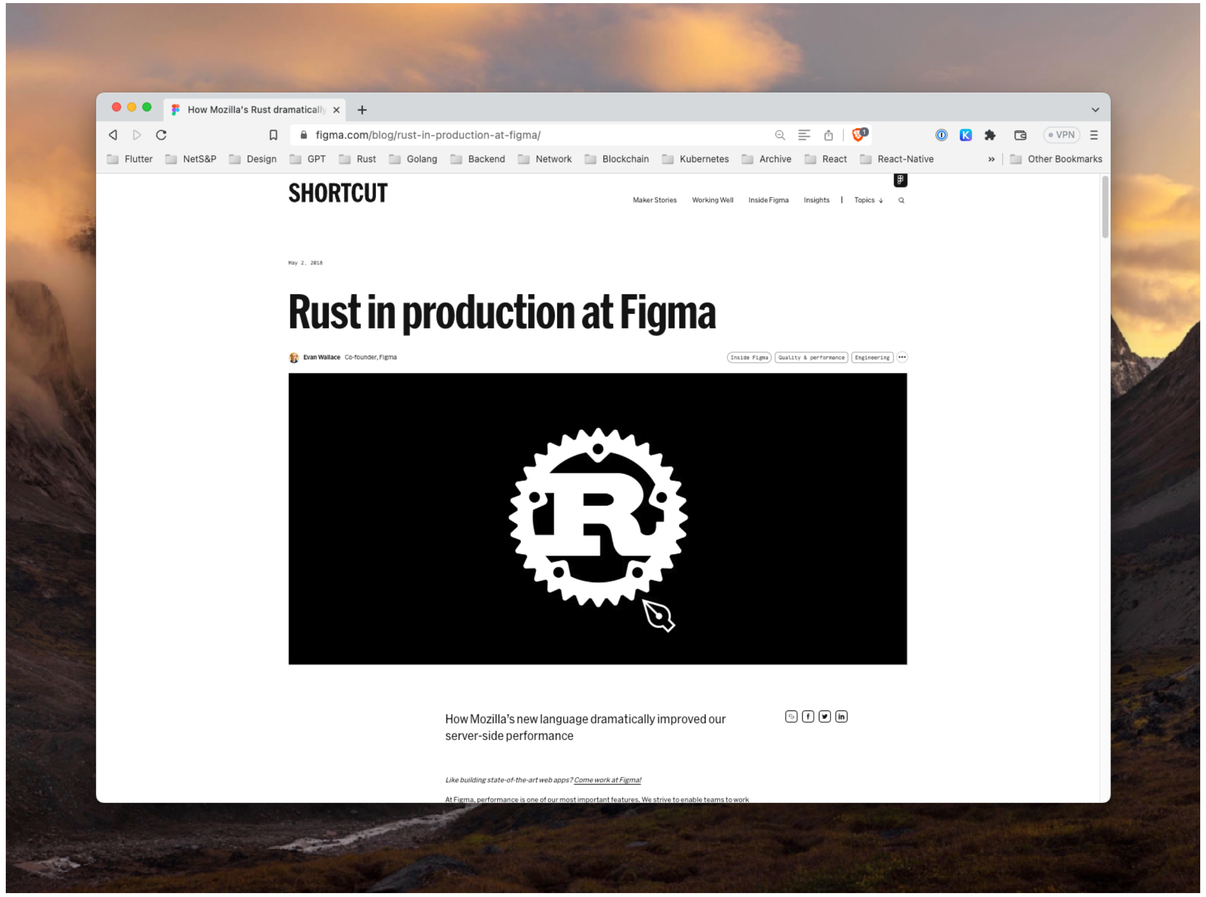
```
> 2 + 2
<· 4
> "2" + "2"
<· "22"
> 2 + 2 - 2
<· 2
> "2" + "2" - "2"
<· 20
```



Limits and Weaknesses: Single-Threaded

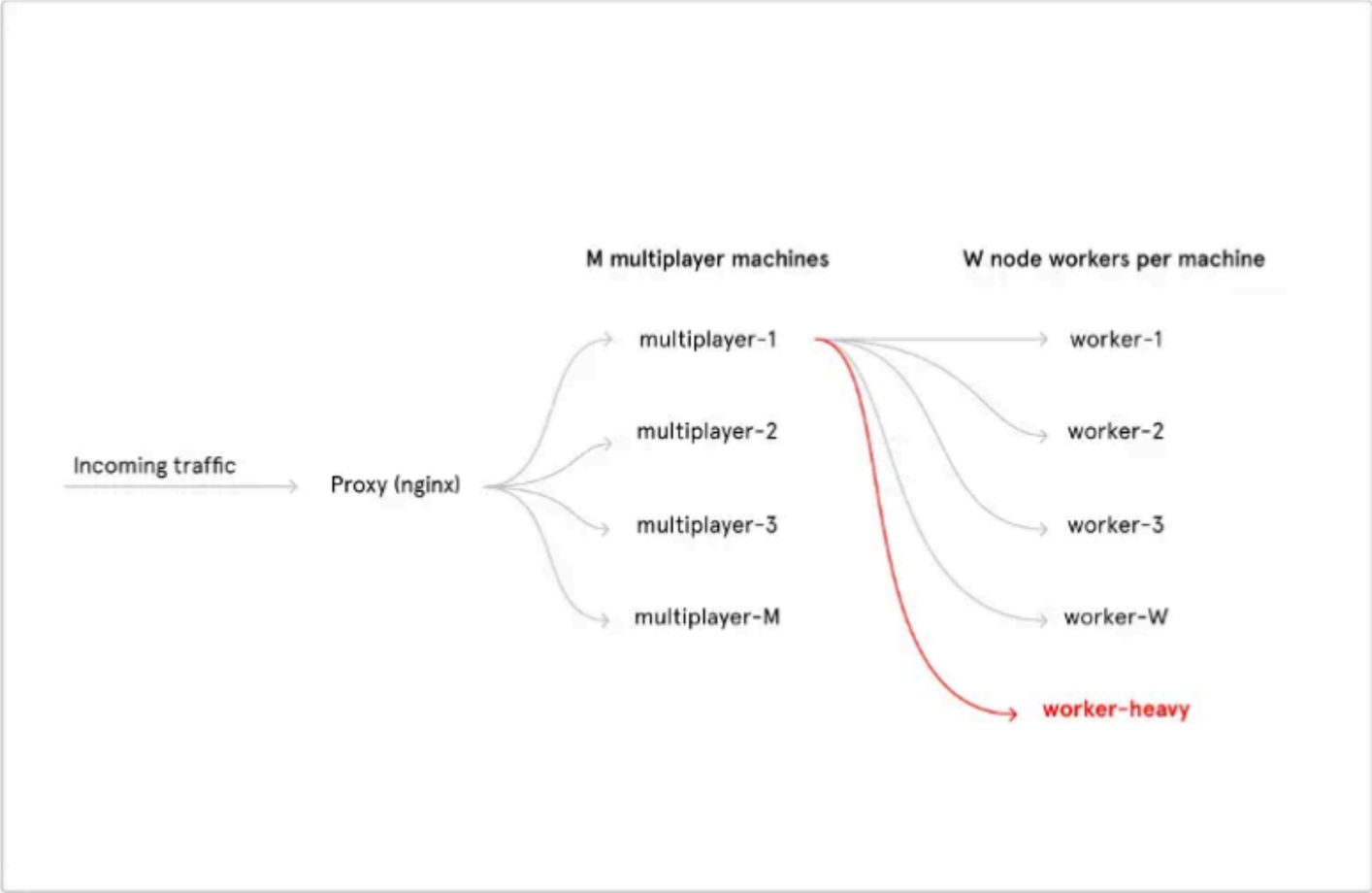
Javascript

Figma



Limits and Weaknesses: Single-Threaded

Javascript

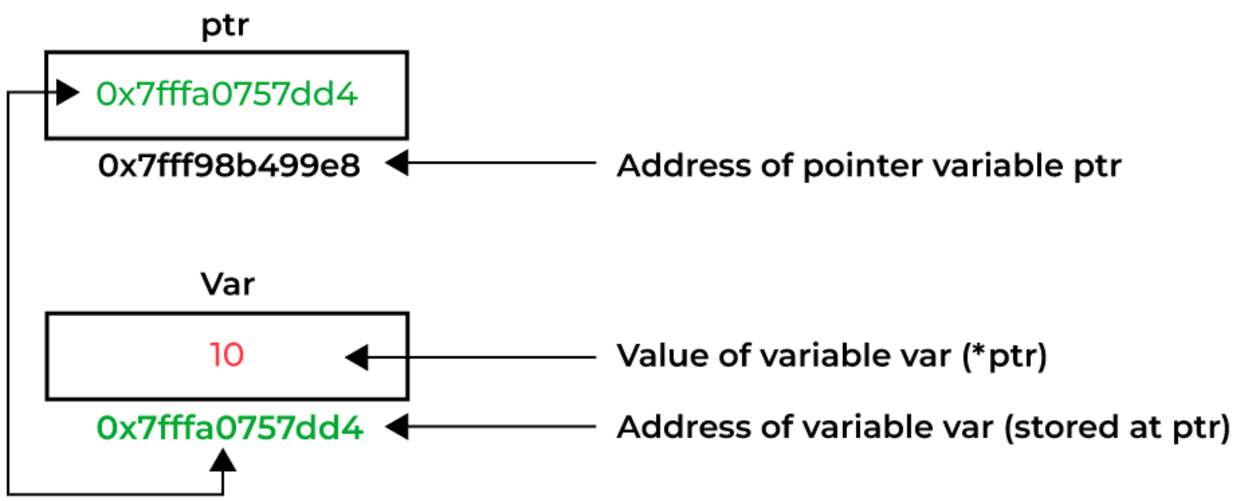


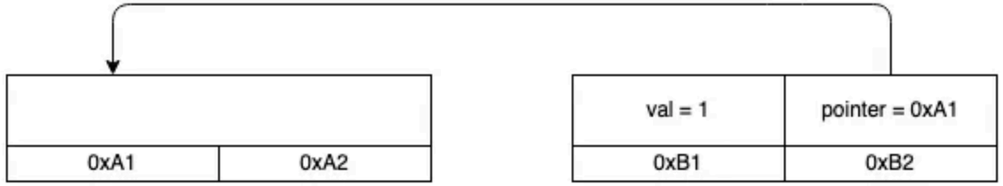
Real-world Projects

Javascript

1. Frontend WebPage (React, NextJS)
2. Mobile App (React Native)
3. Backend API Server (NestJS)

Low-level Languages





Golang

Open source programming language supported **by Google** that makes it simple to build secure, scalable systems with built-in **concurrency** and a **robust** standard library

Real-world Projects

Golang

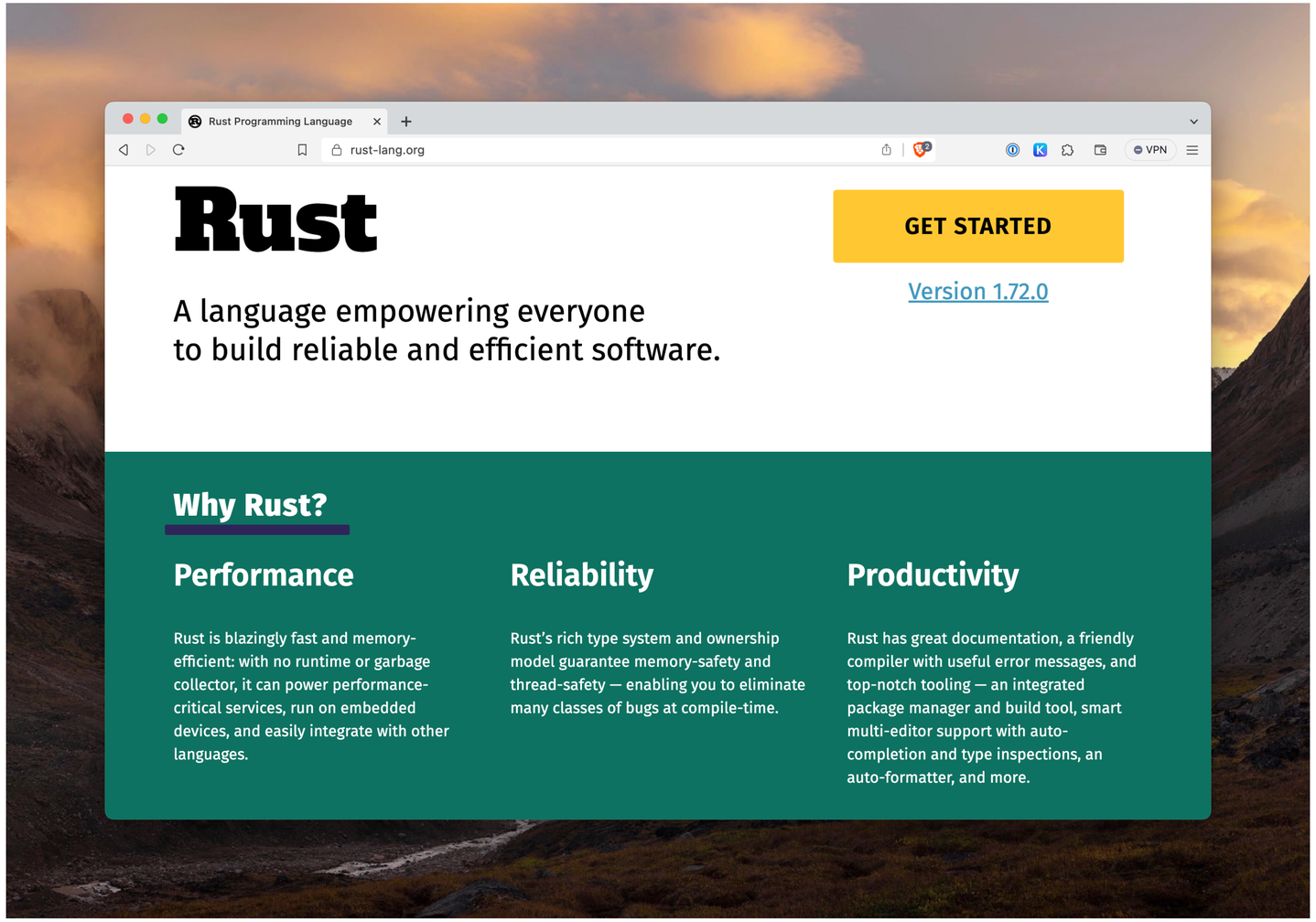
1. Simple network proxy
2. Backend API server (std net/http)
3. Geth (Go-ethereum implementation)
4. Chat server relayer

Rust

Rust is **blazingly fast and memory-efficient**; with no runtime or garbage collector
With rich type system and ownership model, it guarantee **memory-safety** and **thread-safety**

Purpose of this Language

Rust



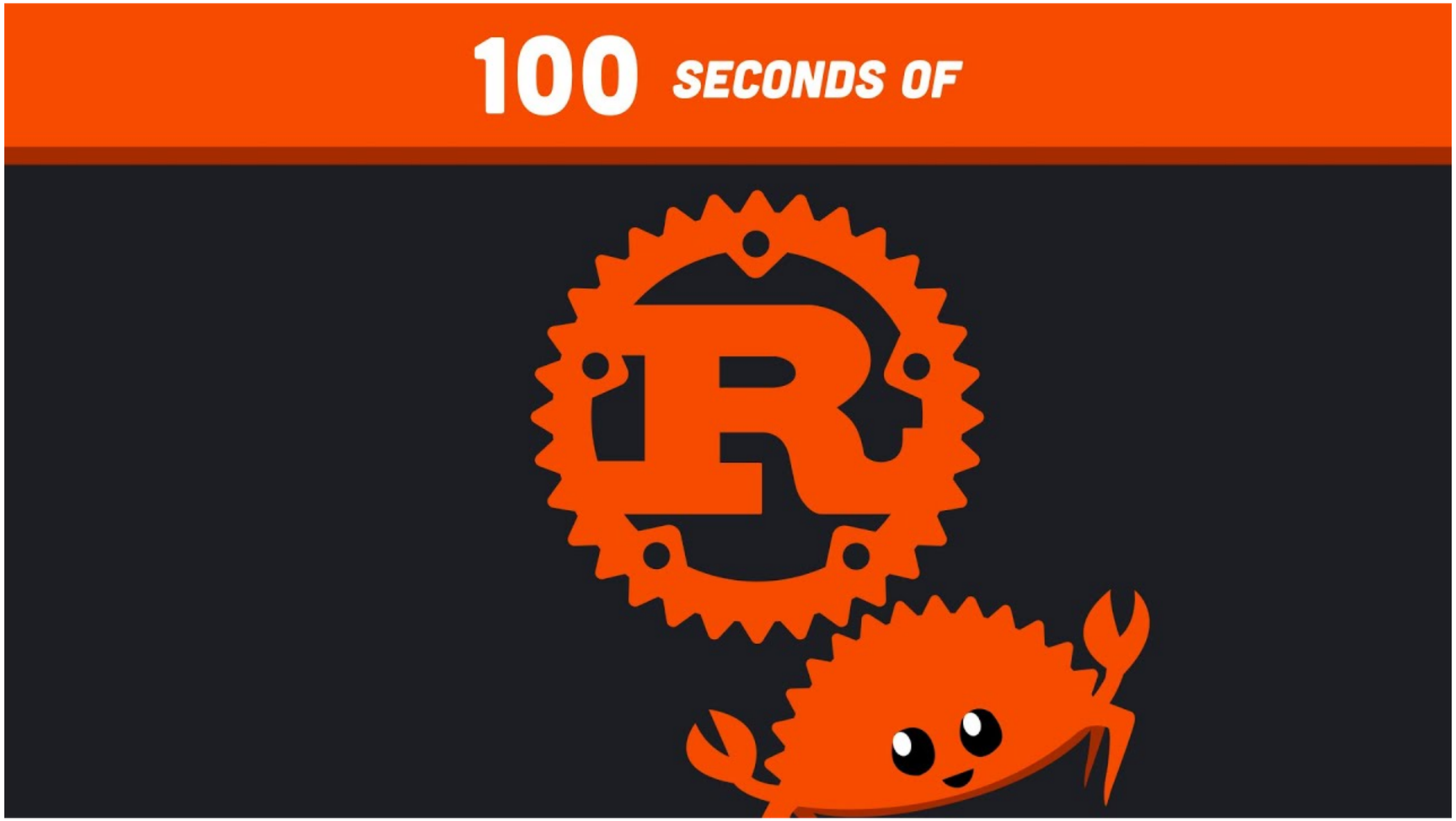
Characteristics and Syntax

Rust

“Ownership and Borrowing”

Characteristics and Syntax

Rust



Characteristics and Syntax

Rust

```rust
fn main() {
    let a = String::from("hello");
    print_str(a);
}

fn print_str(text: String) {
    println!("{}", text);
}
```

```rust
fn main() {
    let a = String::from("hello");
    print_str(a);
    println!("{}", a);
}

fn print_str(text: String) {
    println!("{}", text);
}
```

Characteristics and Syntax

Rust

```rust
fn main() {
    let a = String::from("hello");
    print_str(a);
    println!("{}", a);
}

fn print_str(text: String) {
    println!("{}", text);
}
```

```
   Compiling bitcoin v0.1.0 (/Users/jaehong21/Desktop/lab/mock-bitcoin-peer)
error[E0382]: borrow of moved value: `a`
 --> src/main.rs:4:20
  |
2 |     let a = String::from("hello");
  |         - move occurs because `a` has type `String`, which does not implement the `Copy` trait
3 |     print_str(a);
  |               - value moved here
4 |     println!("{}", a);
  |                    ^ value borrowed here after move
  |
```

“Limits and Weaknesses”

Rust

“Difficult and Complex”

Limits and Weaknesses

Rust

Python

```python
async def valid_request(
    auth: Optional[HTTPAuthorizationCredentials] = Depends(get_bearer_token),
) -> str:
    try:
        if auth is None:
            return None
        token = auth.credentials
        payload = jwt.decode(
            token, JWT_SECRET_KEY.encode("utf-8"), algorithms=[ALGORITHM]
        )
        id = payload.get("sub")
        if id is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED, detail="sub not found"
            )
    except JWTError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Could not validate credentials",
        )
    return id
```

Rust

```rust
pub async fn validate_request(
    req: HttpRequest,
    pool: &Pool<Postgres>,
) -> Result<String, HttpResponse> {
    // access_token from request header
    let access_token: String = match req.headers().get("Authorization") {
        Some(token) => match token.to_str() {
            Ok(token) => token.replace("Bearer ", ""),
            Err(_) => {
                return Err(HttpResponse::InternalServerError().json(JsonMessage {
                    msg: "Error parsing header to string".to_string(),
                }));
            }
        },
        None => {
            return Err(HttpResponse::InternalServerError().json(JsonMessage {
                msg: "Authorization field not exist".to_string(),
            }));
        }
    };

    // find user from database
    let user: Option<User> = match validate_jwt(access_token) {
        Ok(sub) => match find_user_by_email(pool, &sub).await {
            Ok(user) => user,
            Err(e) => {
                return Err(HttpResponse::InternalServerError().json(format!("Error: {:?}", e)));
            }
        },
        // JwtError
        Err(e) => {
            return Err(HttpResponse::Unauthorized().json(JsonMessage {
                msg: format!("{:?}", e),
            }));
        }
    };
```

Documentation and Ecosystem

Rust

Rust Foundation

Company
Amazon
Microsoft Corporation
Facebook
Dropbox
Mozilla
Cloudflare
Coursera
Discord
Figma
npm

Microsoft

Linux™

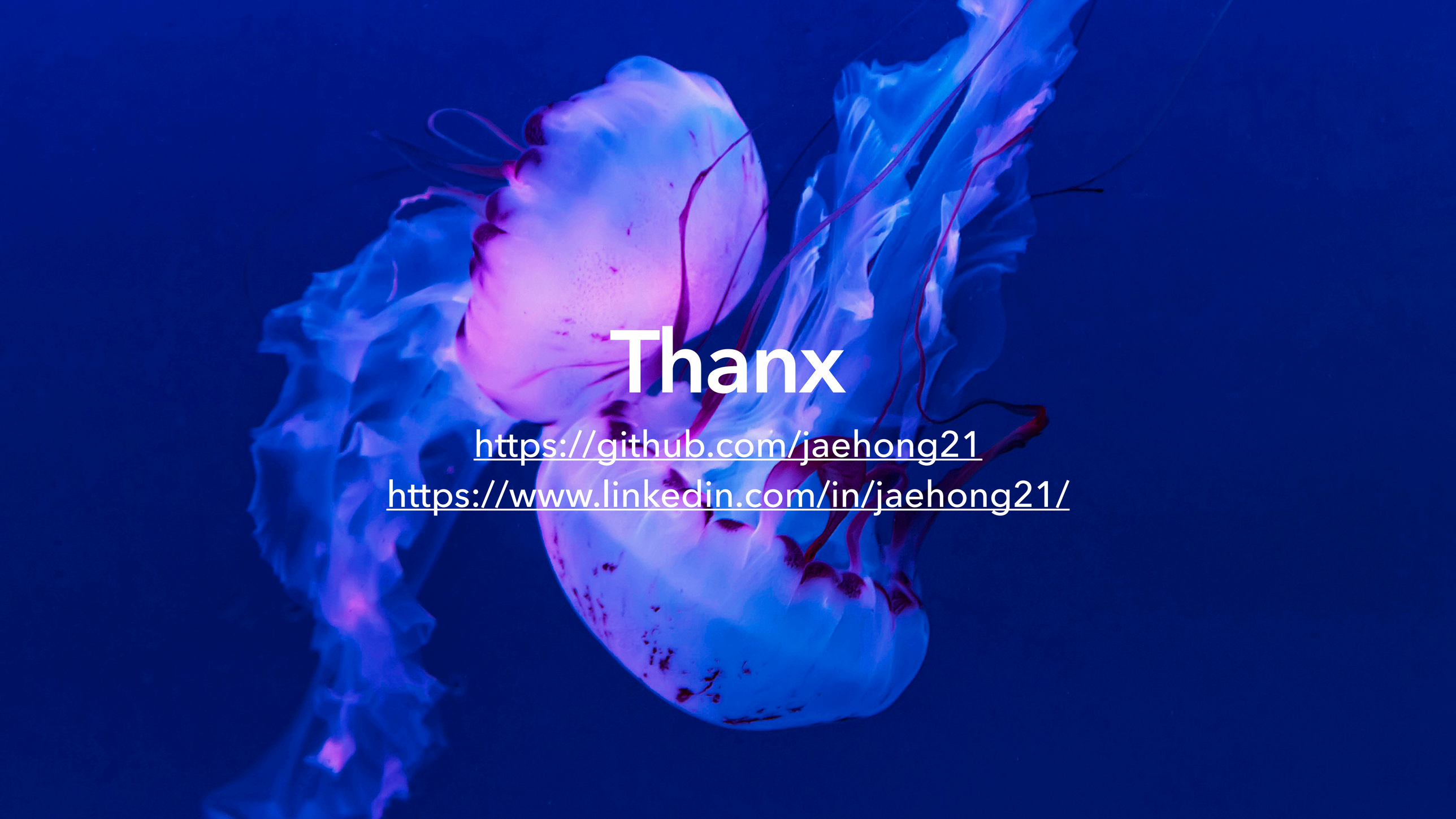
CLOUDFLARE®

SOLANA

Real-world Projects

Rust

1. Backend API server (actix-web)
2. **Mock Bitcoin node** written in Rust



Thanx

https://github.com/jaehong21

https://www.linkedin.com/in/jaehong21/