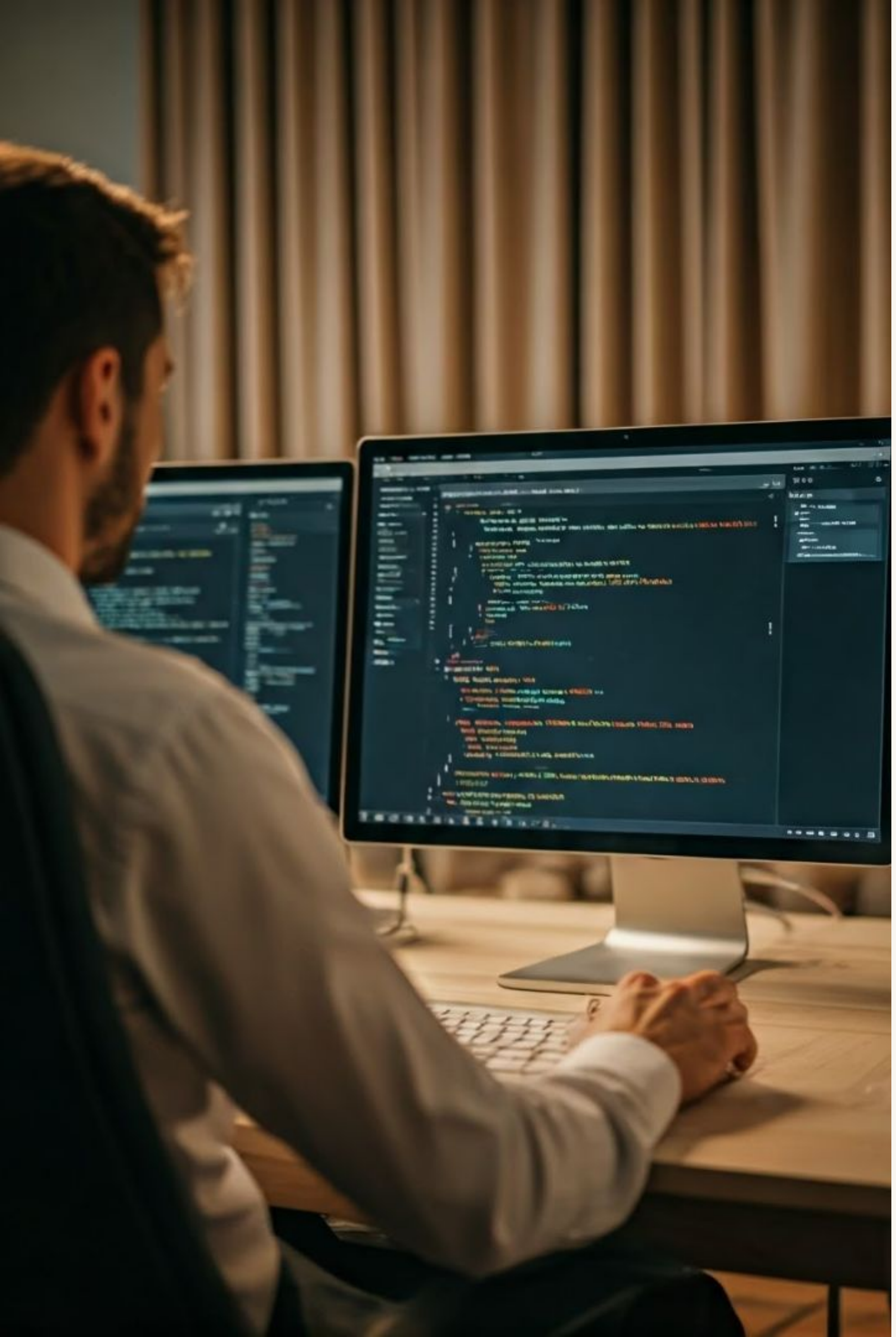


A Little Bit About Rust For Web Developers

Discovering Rust and how its shaping new ways to write programs

By: Muhammed Towfik Jemal

What is Rust?

Modern Systems Programming

A language built for performance and safety without garbage collector and runtime.

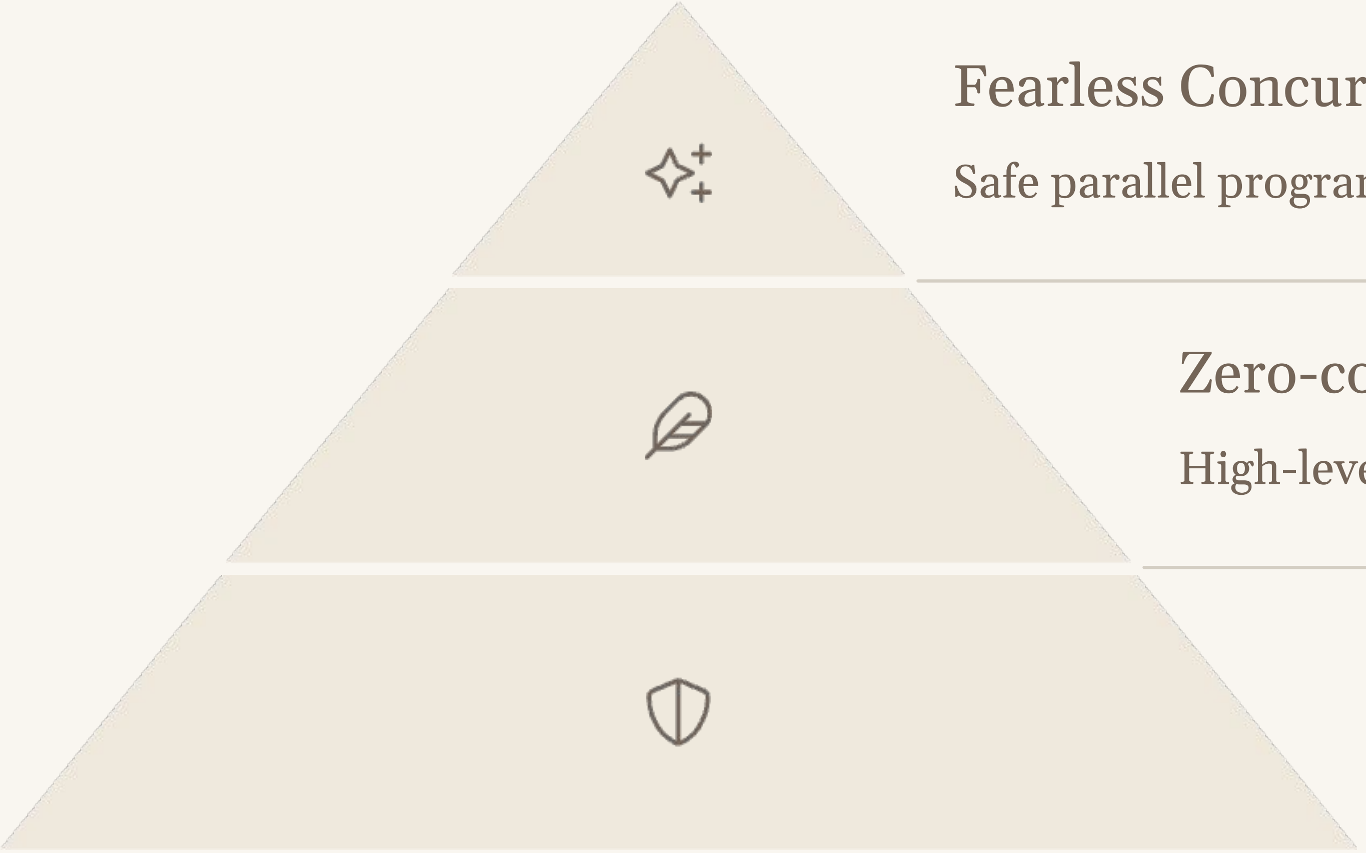
Memory Safety

Prevents common bugs and security issues at compile time.

Runs Everywhere

Can be run on Web, Embedded System and Operating Systems.

Rust's Philosophy and Goals



Fearless Concurrency

Safe parallel programming without data races

Zero-cost Abstractions

High-level code with low-level performance

Memory Safety

Catch errors at compile time, not runtime

Rust is Already in Your Tools

SWC

Powers Next.js, replacing Babel with 20x faster compilation.

Turbopack

Vercel's Webpack replacement with massive speed improvements.

esbuild & Parcel

Modern bundlers inspired by or built with Rust.

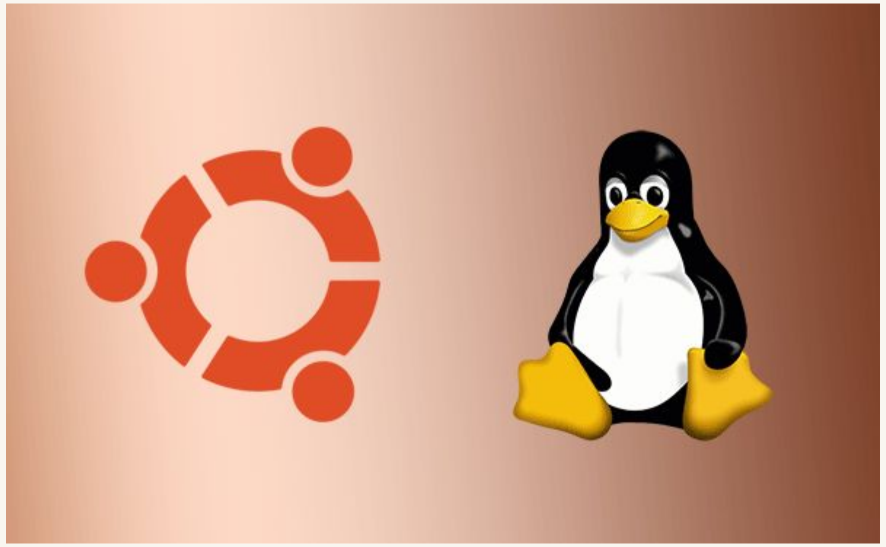
Tech's Using Rust

Cloudflare

Uses for cloudflare workers (with wasm) and Pingora.

Discord

Handles millions of concurrent users with low-latency services.



Linux & Ubuntu

Integrating Rust into critical infrastructure and kernel code.

Memory Handling

Rust's Approach

- Ownership and Borrowing
- No Runtime Overhead
- Explicit Lifetimes:
- Zero-Cost Abstractions (no garbage Collection)

Manual Memory Management (e.g., C, C++) Approach

- Explicit Control
- Risk of Errors
- No Safety Guarantees

Garbage-Collected Languages (e.g., Java, JavaScript, C#) Approach

- Automatic Memory Management
- Non-deterministic Deallocation:
- Simplicity for Developers
- No Compile-Time Safety

Reference Counting (e.g., Swift, Python) Approach

- Automatic Deallocation (via Ref Counting)
- Overhead
- More Flexible than GC

Rust's Borrow Checker

Ownership

Each value has a single owner variable

Borrowing

References allow safe access to data

Validation

Compiler enforces rules at build time

Lifetimes

Track how long references stay valid

OwnerShip

- Each value has one owner
- When the owner goes out of scope, the value is automatically dropped
- You can move ownership to another variable, but only one owner exists at a time.
- You can borrow a value temporarily using references, but Rust enforces rules to prevent data races and invalid memory access.

Borrowing

Mutable

- Allows read/write access to a value.
- Only one mutable borrow is allowed at a time.
- Prevents data races by enforcing exclusive access

Immutable

- Allows read-only access to a value.
- Many immutable borrows are allowed at the same time.
- Useful for functions or scopes that only need to inspect data.

LifeTimes

- Ensure that references are always valid — they prevent dangling references at compile time.
- Rust usually infers lifetimes automatically, but sometimes you must annotate them (e.g., in functions returning references).
- Lifetime annotations (like 'a) tell the compiler how long references must live relative to each other.
- Lifetimes don’t affect program behavior at runtime — they exist only at compile time to ensure memory safety.

Validation

Rust borrow checker validation prevents

- Use-after-free
- Double frees
- Dangling references
- Data races
- Aliasing bugs
- Unsafe mutation
- Using a reference to data that's already been dropped

Developer Experience

Cargo

All-in-one package manager, build tool, and test runner.

- Dependency management
- Consistent project structure
- Built-in testing framework

Crates.io

Rich ecosystem of reusable packages.

- Over 100,000 packages
- Semantic versioning
- Strong security focus

Tooling

First-class developer tools make coding enjoyable.

- Excellent error messages
- IDE integration
- Documentation generators

Pros & Cons

Pros

- Memory Safety Without Garbage Collection
- Performance
- Concurrency and Parallelism
- Strong Type System and Compile-Time Safety
- Cross-Platform and Embedded Development
- Zero-Cost Abstractions
- Growing Ecosystem and Tooling

Cons

- Steep Learning Curve
- Compilation Time
- Limited Ecosystem Compared to More Mature Languages
- Smaller Talent Pool
- Less Mature Tooling for Some Domains

When To Use Rust

1. Performance-Critical Applications
2. Memory Safety Without GC
3. Concurrency and Parallelism
4. Embedded and Low-Level Systems
5. Large-Scale Software with High Safety Requirements

When Not To Use Rust

1. Rapid Development Need
2. Non-Systems or Non-Performance-Critical Workloads
3. Simple CRUD APPS

Why Really Learn Rust ?

(My Take)

1. To Write Performant Code

2. To Have a Memory-First Mindset

3. To Understand Concurrency

4. To Understand Low-Level Programming

5. To Understand JavaScript Tools Written in Rust

Where To Learn?

1. The Rust Book
2. Rustlings Course
3. Rust By Example

Ownership taken. Bugs avoided.
Rust on.

Open For Any Questions