

Fast and safe web services with  
axum

---

# About me

- Tim Eggert
- Working at a Startup called drant as a Staff Engineer & Security Officer
- Recently moved to Baldenhain
- **Looking for a cozy place to talk about tech frequently!**

Me trying to make friends



# Goal

- Broad overview about Rust and it's current usage/applications
- Detailed example to illustrate some features
- Encourage audience to try it out
- Provide resources for continued research

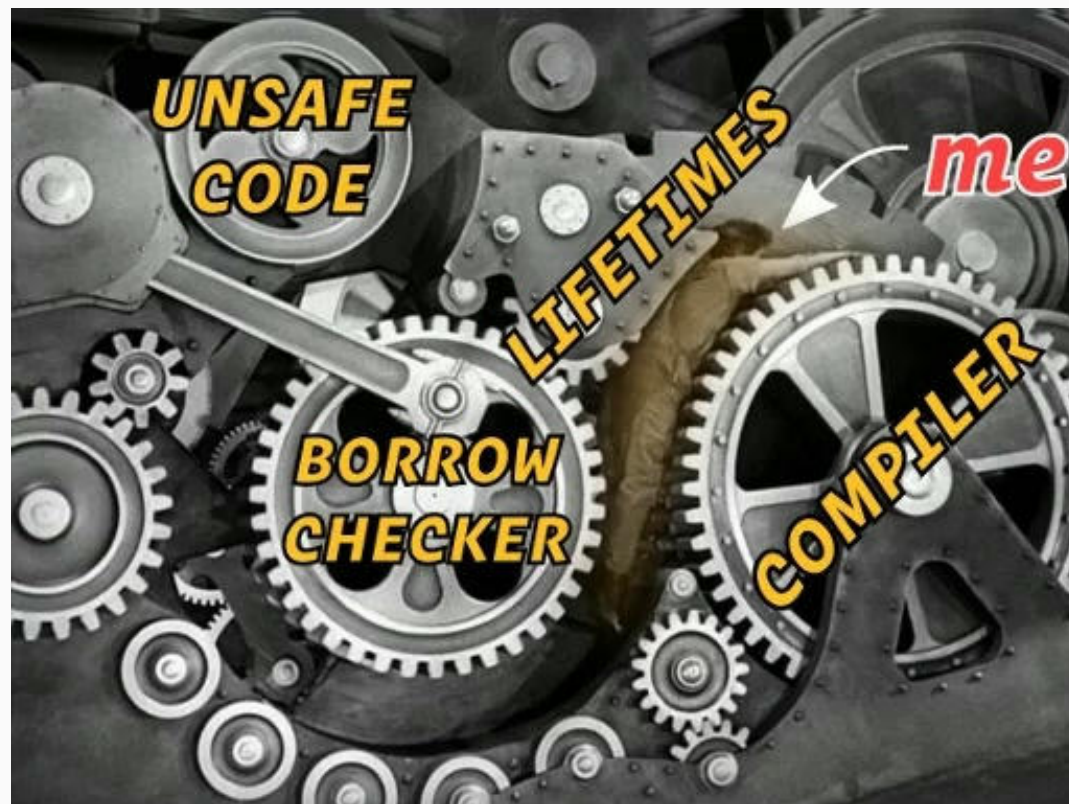


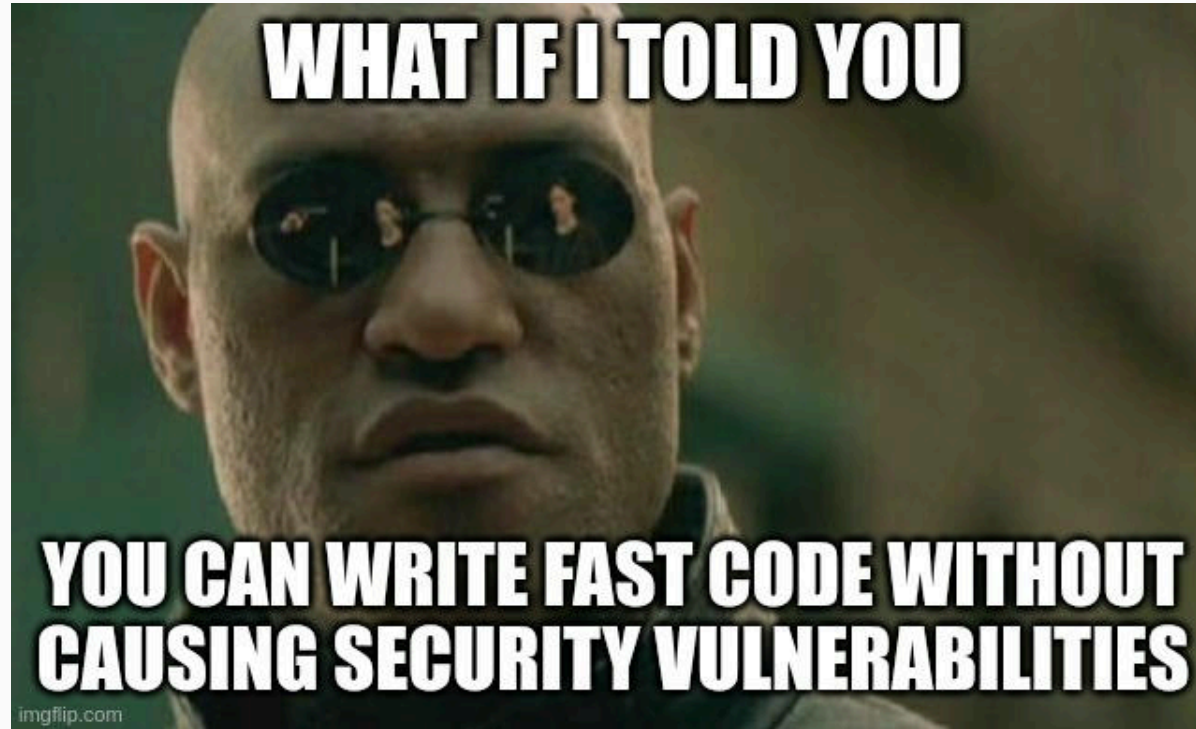
# My way to Rust

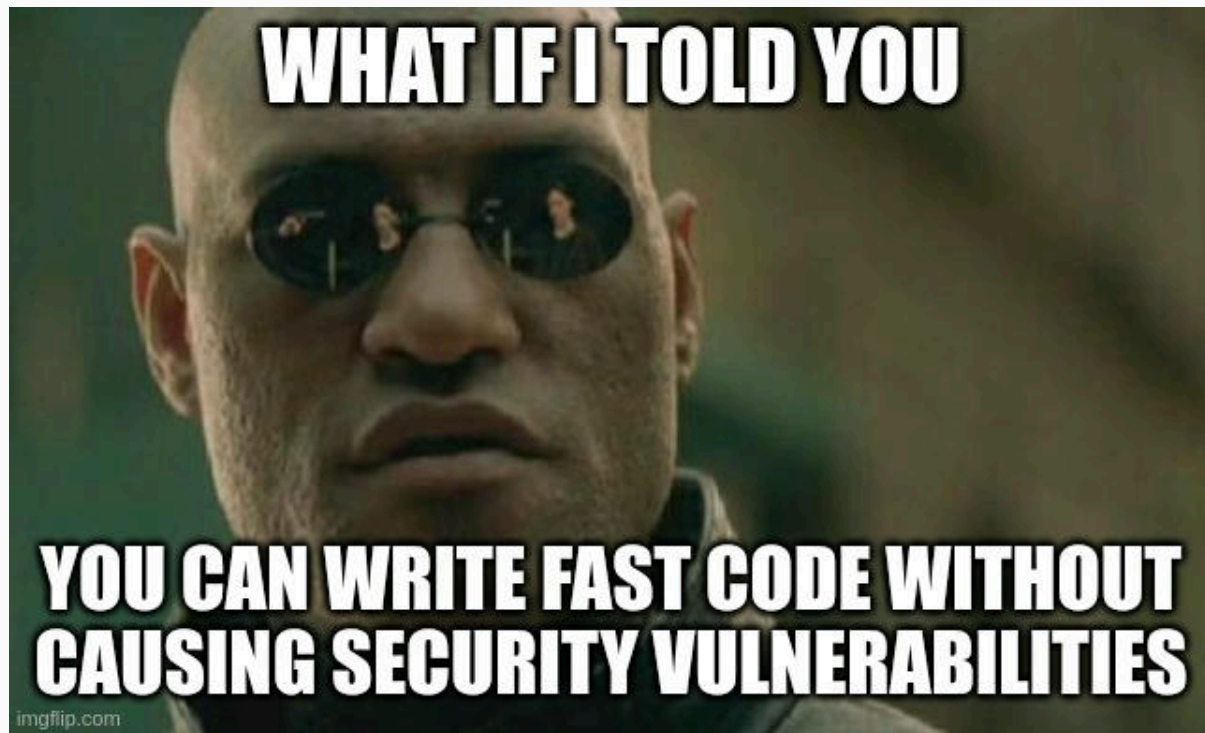
- Started programming with dynamic languages (PHP, later Python)
- Sometimes Java, but too verbose...
- Some C++ 11 with type inference

# My way to Rust

- Started programming with dynamic languages (PHP, later Python)
- Sometimes Java, but too verbose...
- Some C++ 11 with type inference
- Started my Rust journey in 2018 (freetime only)
  - Steep learning curve
  - Lot's of headaches with the Memory model







“A language empowering everyone to build reliable and efficient software.”

— <https://www.rust-lang.org>

**Disclaimer!!!**

**I am a fan of Rust, there might be a bias towards it which I cannot justify rationally...**



# Rust Intro: Typical language features

- Compiled, statically typed language
- Generics
- Async
- Macros
- Multi-Threading
- Concurrency & Thread Safety
- Zero Cost Abstractions
- Minimal Runtime
- Inline Assembly Code

# Rust Intro: Novel / exotic language features

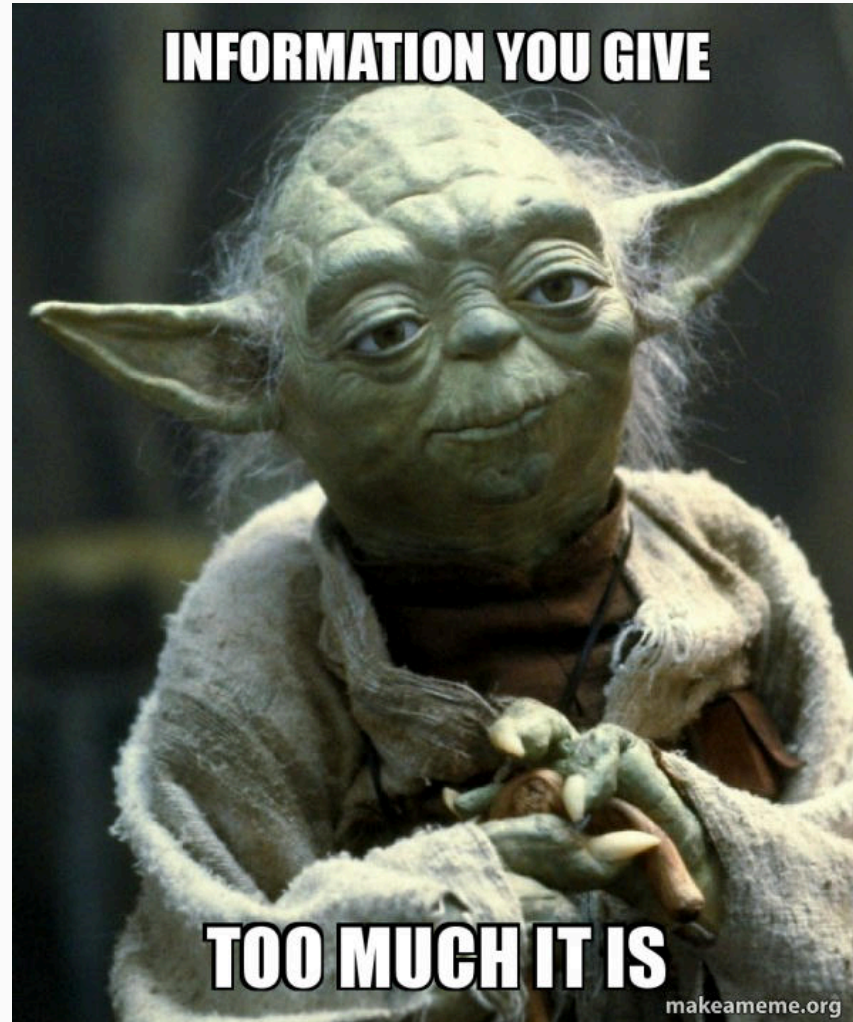
- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default

# Rust Intro: Novel / exotic language features

- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default
- Ecosystem at hand with cargo: Build, test, release, format, lint, manage dependencies (so-called crates)
- [Many different build targets](#)
  - Embedded devices like ESP32
  - Major CPU architectures: x86, ARM,
  - Multiple platforms: Linux, Mac, Windows, Android, Web Assembly, ...

# Rust Intro: Novel / exotic language features

- Ownership with Borrow Checker
- Memory Safety without Garbage Collection
- Pattern Matching + Advanced Enums
- Traits + Trait Objects (Structs, but no Classes; Composition over Inheritance)
- Option (no nulls pointers!) and Result Types
- Immutable by Default
- Ecosystem at hand with cargo: Build, test, release, format, lint, manage dependencies (so-called crates)
- [Many different build targets](#)
  - Embedded devices like ESP32
  - Major CPU architectures: x86, ARM,
  - Multiple platforms: Linux, Mac, Windows, Android, Web Assembly, ...
- Widely adopted own documentation standard
- Enterprise features for stability and maintainability (Editions)



# Rust Intro: Origins

- Brainchild of Graydon Hoare started in 2006
- Sponsored (2009) + officially adopted (2010) by Mozilla
- Decision at Mozilla: Build new Browser Engine (Servo) from scratch based on Rust
- Rust 1.0 was released (2015)
- Adoption by bigger companies in 2020 (Amazon, Google, Microsoft)
  - Rust Foundation was created in 2021
  - Big Corps hired core engineers (Ex-Mozillians) and also long standing community members

- [Android](#)
- AWS Lambda (via [Firecracker](#))
- Discord, Dropbox, Cloudflare backend systems
- Mozilla Servo (browser engine)

- [Android](#)
  - AWS Lambda (via [Firecracker](#))
  - Discord, Dropbox, Cloudflare backend systems
  - Mozilla Servo (browser engine)
  - Databases (Meilisearch, Qdrant, ...)
  - CLI Tools , Editors, Terminal Emulators, Shells, Language tooling (ruff, uv, ...)
  - Deno (nodejs runtime competitor)
  - Operating Systems (Redox OS, Linux Kernel)
  - Cryptocurrency projects
- ... and [many many more](#)





# Demo Time: Setup / Installation

- Install Rust toolchain via [rustup.rs](https://rustup.rs):

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



- Initiate a new project

```
cargo init hello_world
```

```
cd hello_world
```

```
cargo run
```



# Demo Time: Hello World

```
1 fn main() {  
2     println!("Hello World!");  
3 }
```



- Famous choices: Actix Web, Rocket, Warp, Axum, Rouille, Tide, ...
- [Framework Comparison](#)
- For benchmark results, see the [Tech Empower Web Framework Benchmarks](#)

# ToDo Example

## Initialize project

```
cargo init todos  
cd todos
```

 Shell

## Install dependencies

```
cargo add axum  
cargo add tokio --features full  
cargo add serde --features derive  
cargo add uuid --features v4 --features serde  
cargo add serde_json
```

 Shell

# ToDo Example

## Static Axum Server



```
1 use axum::{response::IntoResponse, routing::get, Router};
2
3 #[tokio::main]
4 async fn main() {
5     let app = Router::new().route("/todos", get(list_todos));
6     println!("Server running on http://127.0.0.1:3000");
7     let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
8     axum::serve(listener, app).await.unwrap();
9 }
```

# ToDo Example

## Static Axum Server



```
1 use axum::{response::IntoResponse, routing::get, Router};
2
3 #[tokio::main]
4 async fn main() {
5     let app = Router::new().route("/todos", get(list_todos));
6     println!("Server running on http://127.0.0.1:3000");
7     let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
8     axum::serve(listener, app).await.unwrap();
9 }
```

# ToDo Example

## Static Axum Server



```
1 use axum::{response::IntoResponse, routing::get, Router};
2
3 #[tokio::main]
4 async fn main() {
5     let app = Router::new().route("/todos", get(list_todos));
6     println!("Server running on http://127.0.0.1:3000");
7     let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
8     axum::serve(listener, app).await.unwrap();
9 }
```



# ToDo Example

## Static Axum Server



```
1 use axum::{response::IntoResponse, routing::get, Router};
2
3 #[tokio::main]
4 async fn main() {
5     let app = Router::new().route("/todos", get(list_todos));
6     println!("Server running on http://127.0.0.1:3000");
7     let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
8     axum::serve(listener, app).await.unwrap();
9 }
```

# ToDo Example

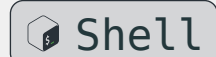
## handler function to list Todos



```
1 async fn list_todos() -> impl IntoResponse {  
2     // Just return a static string for now  
3     return "Todos";  
4 }
```

# ToDo Example

```
cargo run --example 01_static
```



```
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.03s
```

```
Running `target/debug/examples/01_static`
```

```
Server running on http://127.0.0.1:3000
```

# ToDo Example

## Add a TodoItem type / struct



```
1 use serde::{Deserialize, Serialize};
2 use uuid::Uuid;
3
4 #[derive(Serialize, Deserialize, Clone)]
5 struct TodoItem {
6     id: Uuid,
7     title: String,
8     completed: bool,
9 }
```

# ToDo Example

## Return static TodoItem as JSON



```
1  use axum::Json;
2
3  async fn list_todos() -> Json<Vec<TodoItem>> {
4      let todo = TodoItem {
5          id: Uuid::new_v4(),
6          title: "First Todo".into(),
7          completed: false,
8      };
9      return Json(vec![todo]);
10 }
```

# ToDo Example

## Add shared state



```
1 use tokio::sync::RwLock;  
2  
3 #[derive(Default)]  
4 struct AppState {  
5     todos: RwLock<Vec<TodoItem>>,  
6 }
```

# ToDo Example

## Add shared state



```
1  use std::sync::Arc;
2
3  #[tokio::main]
4  async fn main() {
5      let state = Arc::new(AppState::default());
6      let app = Router::new()
7          .route("/todos", get(list_todos))
8          .with_state(state);
9      ...
10     axum::serve(listener, app).await.unwrap();
11 }
```

# ToDo Example

## Use AppState in list handler



```
1 use axum::extract::State;  
2  
3 async fn list_todos(State(state): State<Arc<AppState>>) -> Json<Vec<TodoItem>> {  
4     let todos = state.todos.read().await.clone();  
5     return Json(todos);  
6 }
```



# ToDo Example

Add POST /todos



```
1  use std::sync::Arc;
2
3  #[tokio::main]
4  async fn main() {
5      let state = Arc::new(AppState::default());
6      let app = Router::new()
7          .route("/todos", get(list_todos).post(create_todo))
8          .with_state(state);
9      ...
10     axum::serve(listener, app).await.unwrap();
11 }
```

# ToDo Example

Add a request type for TodoItem



```
1 #[derive(Serialize, Deserialize, Clone)]
2 struct TodoItemCreateRequest {
3     title: String,
4     completed: bool,
5 }
```

# ToDo Example

## Add create\_todo handler function



```
1  async fn create_todo(  
2      State(state): State<Arc<AppState>>,  
3      Json(payload): Json<TodoItemCreateRequest>,  
4  ) -> Json<TodoItem> {  
5      let mut todos = state.todos.write().await;  
6      let todo = TodoItem {  
7          id: Uuid::new_v4(),  
8          title: payload.title,  
9          completed: payload.completed,  
10     };  
11     todos.push(todo.clone());  
12     Json(todo)  
13 }
```

# ToDo Example

Add DELETE /todo/{id}



```
1  use axum::routing::delete;
2
3  #[tokio::main]
4  async fn main() {
5      let state = Arc::new(AppState::default());
6      let app = Router::new()
7          .route("/todos", get(list_todos).post(create_todo))
8          .route("/todos/{id}", delete(delete_todo))
9          .with_state(state);
10     ...
11     axum::serve(listener, app).await.unwrap();
12 }
```

# ToDo Example

Add DELETE /todo/{id}


 Rust

```
1 use axum::{extract::Path, http::StatusCode};
2
3 async fn delete_todo(
4     Path(id): Path<Uuid>,
5     State(state): State<Arc<AppState>>
6 ) -> StatusCode {
7     let mut todos = state.todos.write().await;
8     if let Some(pos) = todos.iter().position(|todo| todo.id == id) {
9         todos.remove(pos);
10        StatusCode::NO_CONTENT
11    } else {
12        StatusCode::NOT_FOUND
13    }
14 }
```

# ToDo Example

## Building a release version

```
$> cargo build --release
```

 Shell

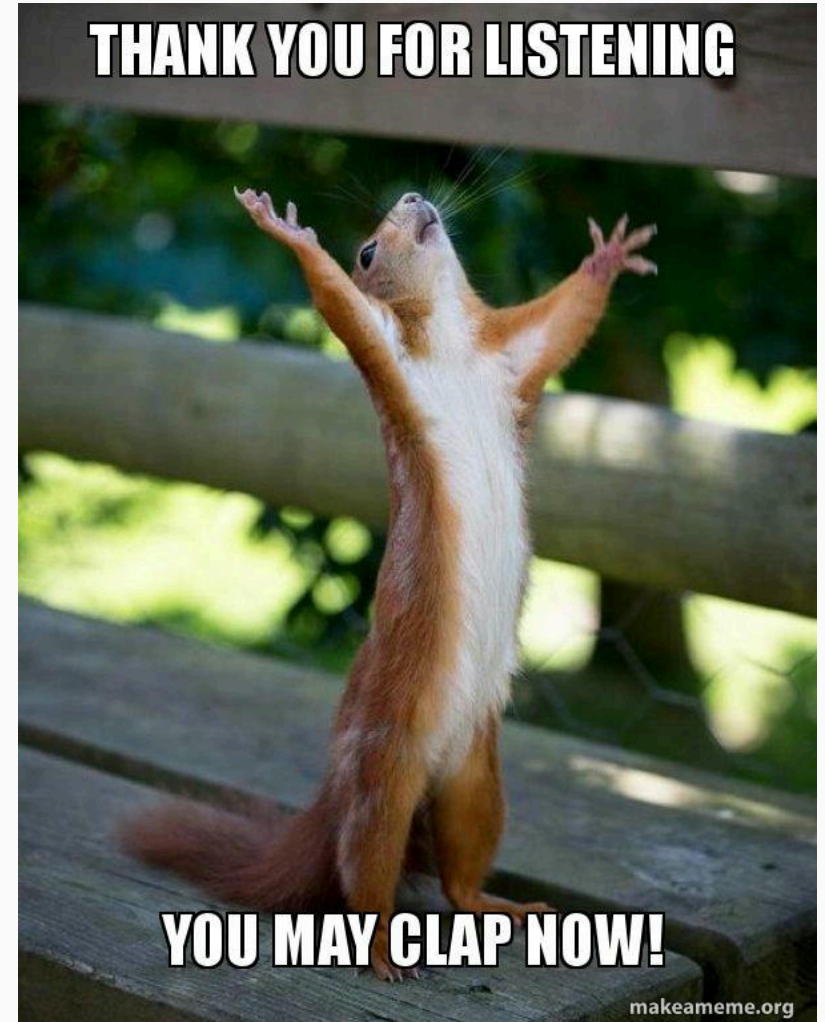
```
$> ls -lah ./target/release/todos
```

```
-rwxr-xr-x@ 1 tim  staff   1.8M Jan 11 00:33 ./target/release/todos*
```

**Questions now or later while socializing :)**

Thanks

Thank you for listening  
and a big Thanks to our  
organizers!





- Weekly newsletter: “This week in Rust”
- Wide Editor support: (RustRover, VSCode, Emacs, Vim, Zed, ...)
- Deep Dives: Crust of Rust
- Books: Which ones?