SYSTEMS · TAUGHT WITH RUST

# Systems Programming in Rust

Ownership and lifetimes as a memory model, not syntax to appease. You write concurrent code the borrow checker signs off on, profile it, and ship a binary that starts instantly.

Intermediate | 5 phases | 25 sessions | 38 contact hours

## Syllabus

5 phases, 25 sessions. Each phase ends in something you have built.

### 1 Ownership as a Memory Model

Read ownership as a description of where memory lives, not a set of rules to argue with.

5 sessions
$72 alone

**1.1 Stack, Heap, and Move**

Where a value actually lives, what a move costs at runtime, and why Copy is not an optimization.

FREE PREVIEW

**1.2 The Borrow Rules**

Shared versus exclusive references, aliasing XOR mutation, and the errors that mean restructure.

**1.3 Lifetimes Are Regions**

Annotations describe facts the compiler already knows, and 'static does not mean forever.

**1.4 Choosing a Smart Pointer**

Box, Rc, RefCell, and Cow, with the runtime cost of interior mutability stated out loud.

**1.5 Cargo, Clippy, Tests**

cargo check, clippy pedantic, rustfmt, workspaces, and reading a rustc error all the way down.

By the end: **A linked data structure ported from C that compiles with zero clones added to appease the compiler.**

## 2 The Type System Does the Work

Delete whole classes of bug with types instead of writing runtime checks for them.

5 sessions
$72 alone

| | | |
|---|---|---|
| 2.1 | **Enums and Exhaustive Match** | Sum types, the absence of null, and making the illegal state something the code cannot express. |
| 2.2 | **Traits, Not Interfaces** | Static dispatch, coherence and the orphan rule, blanket impls, and what a dyn vtable costs. |
| 2.3 | **Generics and Monomorphization** | Bounds, where clauses, and the compile time and binary size that generics quietly buy. |
| 2.4 | **Errors That Compose** | thiserror 2.0 for libraries, anyhow for binaries, the ? operator, and From conversions. |
| 2.5 | **Collections and Iterators** | Vec capacity, HashMap versus BTreeMap, and iterator chains that do not allocate per step. |

By the end: **A library crate with a documented public API, a typed error enum, and no unwrap outside tests.**

## 3 Concurrency Without Data Races

Write multithreaded code where the compiler, not code review, rules out the data race.

5 sessions
$72 alone

| | | |
|---|---|---|
| 3.1 | **Send, Sync, and Threads** | Two auto traits, what they promise, and why the compiler rejects your thread before it runs. |
| 3.2 | **Channels and Ownership Transfer** | std mpsc and crossbeam, moving data instead of sharing it, and what closing a channel means. |
| 3.3 | **Shared State, Measured** | Mutex, RwLock, and atomics, with Arc<Mutex<_>> treated as a decision rather than a reflex. |
| 3.4 | **Rayon and Data Parallelism** | par_iter over a real workload, and the cases where the parallel version is measurably slower. |
| 3.5 | **Deadlocks and Data Races** | Lock ordering, poisoning, and loom exploring the interleavings your tests never happen to hit. |

By the end: **A parallel log processor with benchmarks showing where the parallel version wins and where it loses.**

## 4 Async Rust, Honestly

Understand why async is harder than threads, then write a service that survives being cancelled.

5 sessions
$72 alone

| | | |
|---|---|---|
| 4.1 | **Futures Are State Machines** | poll, wakers, and the fact that an async fn does nothing at all until something drives it. |
| 4.2 | **Tokio's Runtime** | Tasks versus threads, spawn and its 'static bound, and where blocking work actually has to go. |
| 4.3 | **Async Traits, Actually** | async fn in traits since 1.75, still not dyn compatible, and what async-trait boxes for you. |
| 4.4 | **Cancellation and Select** | Dropping a future mid await, timeouts, and the lock guard you must never hold across await. |
| 4.5 | **An HTTP Service** | axum 0.8 on Tokio with serde, reqwest, tracing spans, and shutdown that drains in flight work. |

By the end: **An async HTTP service with structured tracing, timeouts, and a graceful shutdown that is load tested.**

## 5 Unsafe, Fast, and Shipped

Find the real bottleneck, cross the safety boundary on purpose, and hand someone an artifact.

5 sessions
$72 alone

| | | |
|---|---|---|
| 5.1 | **Measure Before Optimizing** | One samply profile, one criterion benchmark, one bottleneck you did not predict, one measured fix. |
| 5.2 | **Unsafe Is a Contract** | Raw pointers, the undefined behavior list, and Miri catching what your tests silently pass. |
| 5.3 | **FFI in Both Directions** | bindgen and cxx against a C library, then PyO3 exposing your crate to a Python caller. |
| 5.4 | **Testing That Finds Bugs** | cargo nextest, proptest for invariants, insta snapshots, and cargo-fuzz pointed at the parser. |
| 5.5 | **Release Builds and WASM** | Release profile, LTO, cross compilation, and the same core shipped through wasm-bindgen. |

By the end: **A native CLI binary and a WASM module built from one core crate, with before and after benchmark numbers.**

## Tools you will use

Rust 1.97, Rust 2024 edition | Cargo, Clippy, rustfmt | Tokio 1.x | axum 0.8 and reqwest | serde 1.0, thiserror 2.0, anyhow 1.0 | clap 4.5 | criterion and divan | samply and cargo-flamegraph | Miri, cargo-nextest, proptest, loom | wasm-bindgen and wasm-pack

## What you will build

1. **A Crate Worth Publishing**
   Typed errors, documented API, no unwrap outside tests
2. **The Parallel Log Cruncher**
   Threads, channels, and rayon against a measured baseline
3. **One Core, Two Targets**
   The same engine as a native CLI and a WASM module, profiled

## Registration

| | |
|---|---|
| Phase 1: Ownership as a Memory Model | $72 |
| Phase 2: The Type System Does the Work | $72 |
| Phase 3: Concurrency Without Data Races | $72 |
| Phase 4: Async Rust, Honestly | $72 |
| Phase 5: Unsafe, Fast, and Shipped | $72 |
| **Whole track, all 5 phases** | **$300** |

Buying every phase separately costs $360, so the whole track saves $60.

No payment is taken online. Registering creates an order and payment instructions are sent with it. Access opens once payment is confirmed.