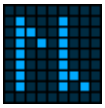

How I think about Zig and Rust

Zig and Rust are the two great systems languages of our time, and discussions of their respective merits generate a lot interest.

But the usual arguments leave me wanting:

- "Rust is safer, but Zig is simpler"
- "Zig is a C replacement, Rust is a C++ replacement"
- "Zig is for super pragmatic people who want to get things done, and Rust is for wizards"[1]

These I think miss the big picture; Zig and Rust come from completely different contexts.

Zig

Zig is firmly a middle level[2] programming language. It is designed to be as expressive as possible in when doing *systems programming* - however it has very little interest in being expressive for *general purpose programming*. It has no:

- operator overloading
- implicit memory allocation
- multi-line comments
- compiler warnings
- closures
- for each loops
- async/await.

Looking at their github issues, the reason usually stated is it makes the language more "complex", or can be done more "simply" using other constructs.

Case in point: most programming languages give you some way to constrain a generic type; Rust's traits, Ocaml's signatures, C#'s interfaces, etc. With Zig you get none of these - everything generic is statically duck-typed with Comptime; Zig's, simple, powerful, and somewhat brutalist solution that tackles the macro and generic problem with a single language construct.

This philosophy carries over to the standard library; Zig's std.os offers functionality only available to Rustaceans in a third party crate. Zig's concept of strings are that they are a slice of bytes that you happen to be able to create with quotation marks, and are dealt with in std.mem. And can you really call yourself a serious systems programming language if you don't have ring buffers in the std lib?

Furthermore, Zig takes C interop very seriously. Most languages have an FFI, some like Rust let you automatically generate the bindings. Zig simply includes a C compiler so you can parse header files and call C code directly - it's basically seamless.

Rust

Rust in contrast, comes from the world of high level languages. It's an attempt to have an expressive language, with a powerful type system, that can nevertheless operate without a runtime, and be deployed where you'd otherwise have to resort to C++.

This is evident in rust's features:

- trait based polymorphism
- pattern matching
- closures
- operator overloading
- monadic error handling
- a macro system more akin to a LISP than to C

Everything Zig leaves out in the name of "simplicity", Rust includes to be more expressive. Even it's much touted "safety" is accomplished through an affine-like type system; sophisticated for any general purpose programming language, let alone one without a garbage collector.

Rust's tooling is far more comfortable and usable than Zigs; the build system is done through a concise configuration file (Zig uses itself to do this imperatively), the test runner output is informative and colourful (Zig's displays nothing), and Rust boasts a large ecosystem of easily installable packages (or "crates" as they call them). Everything a high level language programmer needs to feel welcome is here, and usually to a much higher standard than what they're used to.

What to use?

If I want to do something in the systems space; I want to make linux syscalls, or explicitly control every single memory allocation - Zig is the obvious choice. If I want to be more expressive and high level, but also be able to address systems level concerns without "dropping down" to C - Rust is fantastic for that.

Of course it's likely that one of the language philosophies I've outlined resonates very strongly with you, and one repulses you. So choose what makes you happy.

Ultimately, I hope the question we start to ask ourselves is not "Zig VS Rust", but "Zig *or* Rust". That is to say, they are both such massive improvements over C/C++[3] that the discussion should be which to use today, not which should replace the systems languages of the late 20th century. They're both better in almost every way.

Footnotes

1. I am definitely guilty of this. ↩

2. Here I use the taxonomy that machine code is low level, a garbage collected language is high level, and C is somewhere in-between. ↩

3. Yes I said "C/C++" deliberately. ↩