



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



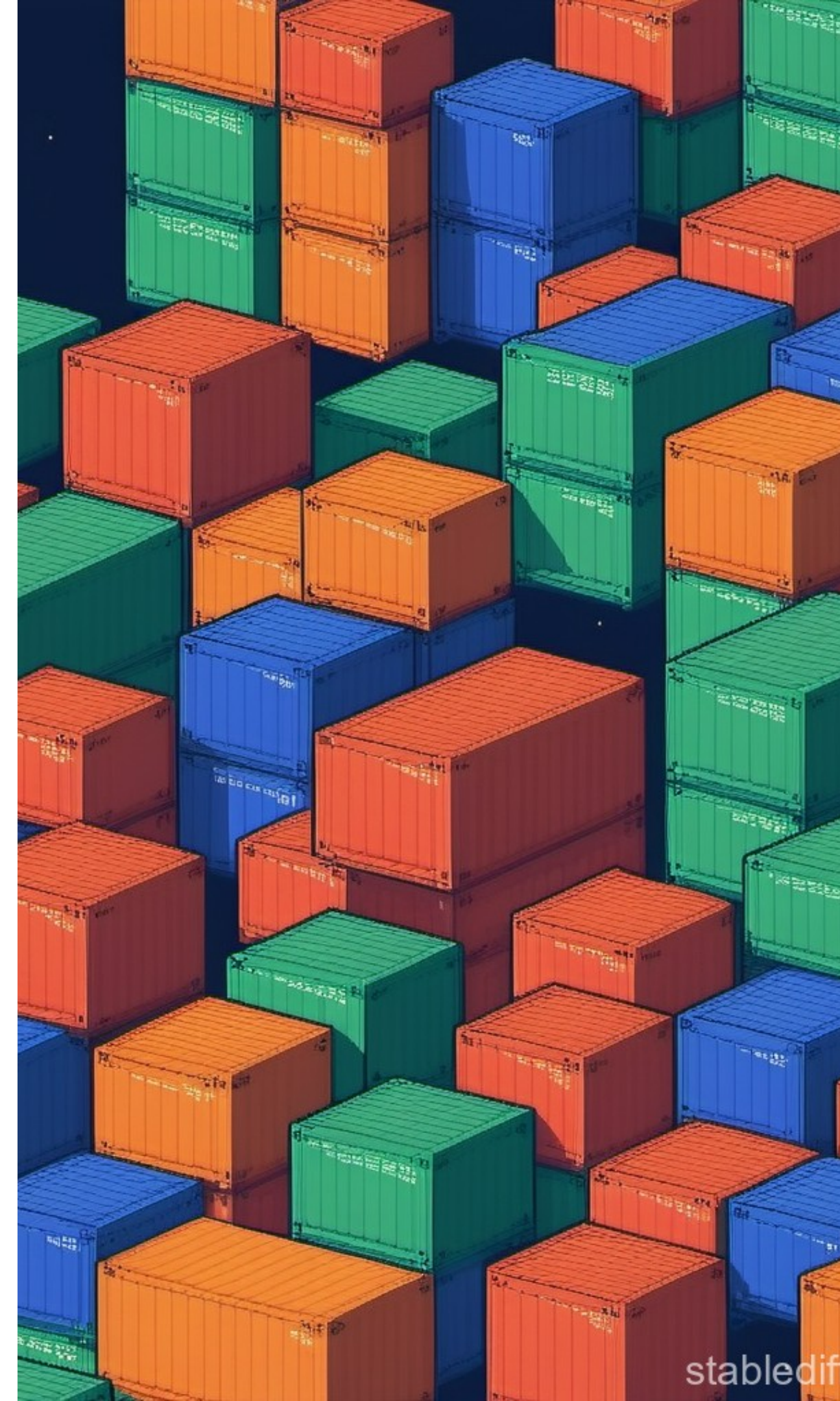
Sarus-slurm

a SPANK slurm Plugin written in Zig

Fawzi Mohamed

Containers

- Give user control of the user space
- install any software
- architecture and kernel remain the same
- various levels of isolation and control
- can be completely rootless (unprivileged), the user does not acquire new
- large ecosystem of tools
- tries to make execution reproducible by freezing the userspace tools and libraries
- mount namespace
- chroot (user has control on the whole filesystem layout)
- uenv ~> container with a "golden image"
- container can give the user more freedom/control
- if one uses that freedom things are not necessarily optimized for the specific hardware
- hooks help in making hardware optimization accessible to more containers



Container first

- Make the usage of container more transparent and seamless
- To the user it should look as much as possible like the normal usage of a cluster
- but with more control on his environment
 - well reproducible (prescriptive)
 - customizable to take advantage of existing containers, test new releases, and use new containers rebuilt from scratch
 - can have usual home/scratch folder with their data
- EDF: Environment Definition File
 - our solution to avoid repeating lot of options
- A [toml](#) file that defines
 - image or a docker file
 - mount points,
 - environment variables
 - annotations
- still in evolution

sample.toml

```
mounts = ["$SCRATCH:$SCRATCH"]
entrypoint = false
[image]
dockerfile=""
FROM nvcr.io/nvidia/pytorch:22.12-py3

# to avoid interaction with apt-get
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y \
    --allow-downgrades --allow-change-held-packages \
    --no-install-recommends \
    build-essential \
    automake \
    autoconf \
    libtool \
    wget \
    libpmi2-0-dev \
    ca-certificates \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

```
RUN wget -q -O nccl-tests-2.13.6.tar.gz \
    https://github.com/NVIDIA/nccl-tests/archive/refs/\
tags/v2.13.6.tar.gz \
    && tar xf nccl-tests-2.13.6.tar.gz \
    && cd nccl-tests-2.13.6 \
    && MPI=1 MPI_HOME=/opt/hpcx/ompi make -j$(nproc) \
    && cd .. \
    && rm -rf nccl-tests-2.13.6.tar.gz

""

[annotations]
com.hooks.aws_ofi_nccl.enabled = "true"
com.hooks.aws_ofi_nccl.variant = "cuda11"
```

Workload manager

- gives the user a way to specify the hardware resources required for his job
- it is a natural place to specify the environment one wants to use
- use an EDF to define the environment
 - pull image only once per job, (or build it)
 - possibly cache it in the parallel filesystem
 - give a predictable shared environment (mounts, env, annotations)
 - cleanup all resources at the end
- to be able to make the usage of the container seamless
 - we need various configuration points in a job lifetime:
 - setup, run and cleanup) of a job

Slurm Integration

- [Slurm](#) is the work load manager we use
- it can be extended with C based plugins ([SPANK](#))

challenges

- C interface
- loaded and executed in various contexts
- difficult to test for failures, corner cases, special configurations of slurm
 - for example missing \$USER in epilogue
- dependencies of plugin are more difficult to handle in a container

SCHEDMD

ENHANCED BY Google



ABOUT

OVERVIEW

RELEASE NOTES

USING

DOCUMENTATION

FAQ

PUBLICATIONS

INSTALLING

DOWNLOAD

RELATED SOFTWARE

INSTALLATION GUIDE

GETTING HELP

MAILING LISTS

SPANK

Section: Slurm Component (8)
Updated: Slurm Component
[Index](#)

NAME §

SPANK - Slurm Plug-in Architecture for Node and job (K)control

DESCRIPTION §

This manual briefly describes the capabilities of the Slurm Plug-in Architecture for Node and job Control (**SPANK**) as well as the **SPANK** configuration file: (By default: **plugstack.conf**.)

SPANK provides a very generic interface for stackable plug-ins which may be used to dynamically modify the job launch code in Slurm. **SPANK** plugins may be built without access to Slurm source code. They need only

Pyxis => Sarus-slurm

- PoC done with Pyxis and enroot
- pyxis issues
 - written in C
 - not ours
 - no unit tests just integration tests
 - difficult to debug
- migrate toward something we own and that we can evolve
 - "sarus-slurm" (provisional name)

Goals

- less error prone language than C
- just our solution
- make it more more flexible? Pluggable interface with
 - createContainer (master node)
 - startContainer (1x per node)
 - execTask (nTaskPerNode)
 - stopContainer (1x per node)
 - destroy/cleanupContainer (master node)



Current PoC

- pyxis drop in replacement
 - find all hidden magic
 - check that everything still works
- pluggable interface
 - support multiple backends
- error trace when failing
- mock test
- no extra dependencies
- language with less footguns than C
- git.cscs.ch/fmohamed/sarus-slurm

Zig

- is a nice small language, quite suited to low level programming, that aims at replacing C.
- expose the the low level primitives and simplify their usage
 - does not hide very much
- debug your application, not your language
 - keep the language simple
- stay in the same space as C
 - encourage reuse of and from C (and C++)



[Download](#) [Learn](#) [News](#) [Zig Software Foundation](#) [Source Code](#) [Join a Community](#)

Zig is a general-purpose programming language and toolchain for maintaining **robust**, **optimal** and **reusable** software.

GET STARTED

Latest Release: **0.11.0**

[Documentation](#)

[Changes](#)

⚡ A Simple Language

Focus on debugging your application rather than debugging your programming language knowledge.

- No hidden control flow.
- No hidden memory allocations.
- No preprocessor, no macros.

⚡ Comptime

A fresh approach to metaprogramming based on compile-time code execution and lazy evaluation.

- Call any function at compile-time.
- Manipulate types as values without runtime overhead.
- Comptime emulates the target architecture.

⚡ Maintain it with Zig

Incrementally improve your C/C++/Zig codebase.

- Use Zig as a zero-dependency, drop-in C/C++ compiler that supports cross-compilation out-of-the-box.
- Leverage `zig build` to create a consistent development environment across all platforms.
- Add a Zig compilation unit to C/C++ projects; cross-language LTO is enabled by default.

[In-depth overview](#)

[More code samples](#)

```
const std = @import("std");
const parseInt = std.fmt.parseInt;

test "parse integers" {
    const input = "123 67 89,99";
    const ally = std.testing.allocator;

    var list = std.ArrayList(u32).init(ally);
    // Ensure the list is freed at scope exit.
    // Try commenting out this line!
    defer list.deinit();

    var it = std.mem.tokenizeAny(u8, input, " ,");
    while (it.next()) |num| {
        const n = try parseInt(u32, num, 10);
        try list.append(n);
    }

    const expected = [_]u32{ 123, 67, 89, 99 };

    for (expected, list.items) |exp, actual| {
        try std.testing.expectEqual(exp, actual);
    }
}
```

```
$ zig test parse_integers.zig
1/1 parse_integers.test.parse integers... OK
All 1 tests passed.
```



- very explicit: no hidden control flow (implicit destructor calls, exceptions), and no hidden memory allocations.
 - a bit more verbose
 - simplifies the analysis of what happens looking just at the local context
 - very useful for low level code, and to optimize
- neither preprocessor, nor macros, but a very nice comptime execution:
 - any function can be called at compile time emulating the target architecture
 - compile time functions can manipulate types
 - Compile time is lazy (no unused function is compiled), and types do ducktyping at compile time.
- zig can incrementally build a C/C++/Zig codebase, and easily crosscompile.

Community

- sycl.it Software you can love
- system programming
 - really understanding the low level
 - i.e.: glibc support
- embedded/freestanding as an option
- cross compilation
- quick compilation -> real incremental compilation

Software You Can Love

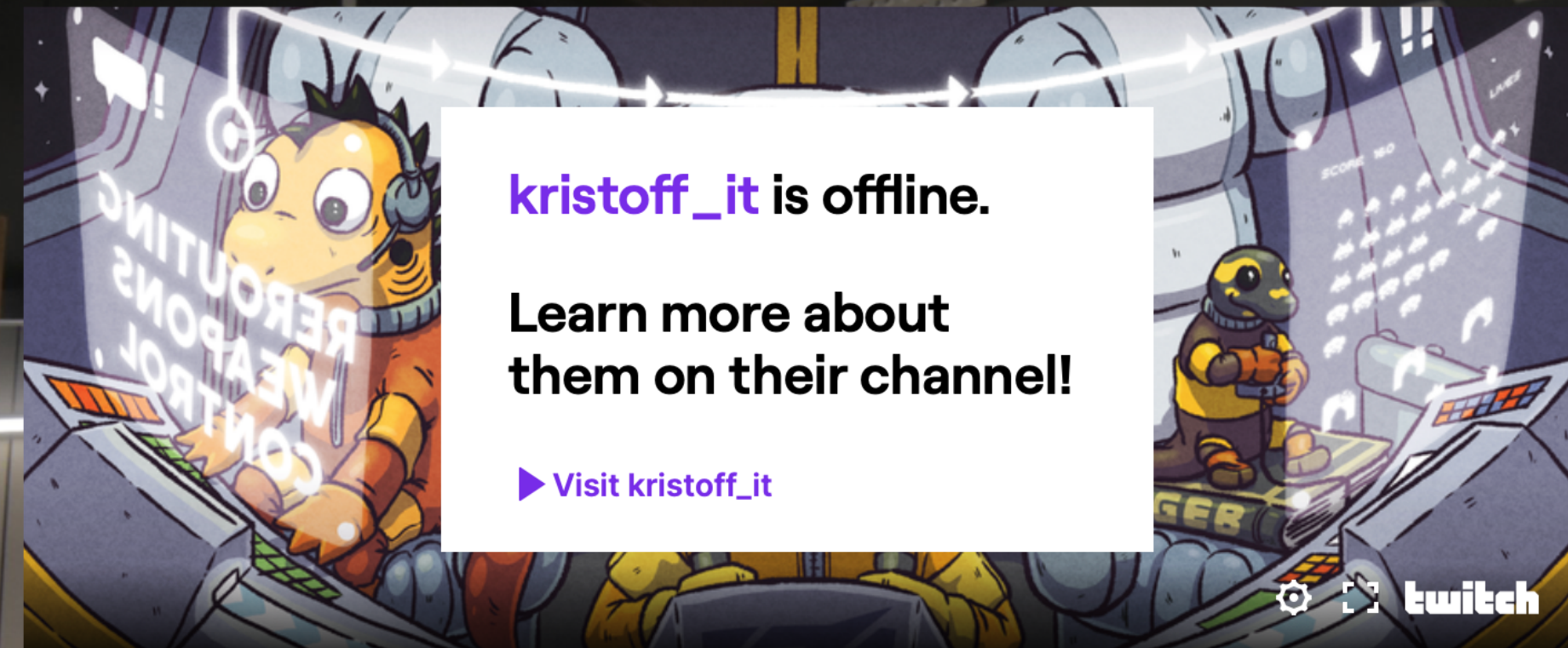
celebrate the art of creating software for humans

[Home](#) • [Location](#) • [Travel FAQs](#) • [Tickets](#) • [Agenda](#) • [Speakers](#) • [About](#)

[YouTube](#) • [Twitch](#) • [Discord](#)

GET NOTIFIED ABOUT THE NEXT EVENT

STREAM WILL GO LIVE ON MAY 16 AND 17 AT 9AM CEST



14-17 May 2024

Milan, Italy



Getting started

- `zig run file.zig`
- `zigtools/zls`
- `zig init-exe -> build.zig`
- zig build system can build zig, C and C++
- `zig build`
- `zig build test`
- <https://ziglang.org/learn/>
 - language
 - standard library

How is Zig

- Order independent top level declarations
- no header files (but h files can be generated)
- type declaration after name
- var/const to declare variables
- const can hold anything, value, functions, struct
- commas also for last element
- fn for functions, first arg can be self (like python)
 - instance.f(x) <=> Class.f(instance,x)
- error handling !type Errors!type
- expressions can return and still assign a value
- defer/errdefer to do cleanup
- unittest
- error traces available on all targets
- explicit allocators

```
test "test_id_list" {
    const a = std.testing.allocator;
    var idList = IdList.init(a);
    defer idList.deinit();
    _ = try idList.addId("pippo");
    try std.testing.expectEqual(idList.ids.items.len, 1);
}

const IdList = struct {
    allocator: std.mem.Allocator,
    ids: std.ArrayListUnmanaged([:0]const u8) = .{},
    pub fn init(allocator: std.mem.Allocator) IdList {
        return .{ .allocator = allocator, }; }
    pub fn deinit(self: *IdList) void {
        for (self.ids.items) |el| self.allocator.free(el);
        return self.ids.deinit(self.allocator); }
    pub fn addId(self: *IdList, value: []const u8) !usize {
        const newId = self.allocator.allocSentinel(
            u8, value.len, 0) catch |err| { return err; };
        errdefer self.allocator.free(newId);
        for (newId, value) |*target, source|
            target.* = if (source == ' ') return error.InvalidId
                        else source;
        try self.ids.append(self.allocator, newId);
        return self.ids.items.len - 1;
    } };

const std = @import("std");
```

C interoperability

```
pub usingnamespace @cImport({
    @cInclude("slurm/spank.h");
    @cInclude("sys/types.h");
    @cInclude("string.h");
    @cInclude("toml.h");
});
```

```
const std = @import("std");
const sarus = @import("sarus_slurm.zig");
const spank = @import("spank.zig");

// the global context used by the plugin
pub var globalContext = sarus.SarusPluginContext{
    .noContext = void{}
};

// define all the c function callbacks
export fn slurm_spank_init(spnk: spank.spank_t,
    ac: c_int, argv: ?[*][*:0]const u8) spank.slurm_err_t {
    return globalContext.slurm_spank_init(spnk, ac, argv);
}
```

- import header files
- pointers
 - `[*c] i32` : a generic c pointer to 32 bit integers (very under determined)
 - `[*] i32` : a non null pointer to an array of unknown length
 - ``[*:0] i32`: a non null pointer to a null terminated array
 - `* i32` : a non null pointer to a single 32 bit integer
- `?T` : a possibly null Type T (optional)
- `[]i32` : slice (range with `.ptr` and `.len`), should be preferred to bare pointers
- `[128]i32`, `[128:0]i32` : fixed size arrays

sarus-args.zig

```
const std = @import("std");
const zspank = @import("sarus_spank.zig");
const spankErrorToCi = zspank.spankErrorToCi;
pub const SarusSlurmArgs = struct {
    allocator: std.mem.Allocator,
    environment: ?[:0]const u8 = null,
    /// Sets the name or path of the environment to read
    pub fn set_environment(self: *SarusSlurmArgs,
        arg_newVal: ?[]const u8) !void {
        if (self.environment) |edf| self.allocator.free(edf);
        self.environment = null;
        if (arg_newVal) |newVal| {
            const newValCopy = try self.allocator
                .allocSentinel(u8, newVal.len, 0);
            @memcpy(newValCopy, newVal);
            self.environment = newValCopy;
        }
    }
    /// callback for environment option
    pub fn cb_spank_option_environment(arg_val: c_int,
        optarg: [*c]const u8, arg_remote: c_int)
        callconv(.C) c_int {
```

```
    _ = arg_val;
    _ = arg_remote;
    if (optarg == null or optarg?[0] == 0) {
        return spankErrorToCi(zspank.SpankError.BAD_ARG);
    }
    const sarus_args: *SarusSlurmArgs =
        globalArgsPrintErr("--environment") catch |err| {
            return spankErrorToCi(err);
        };
    sarus_args.set_environment(sliceTo(optarg, 0))
        catch |err| {
            return spankErrorToCi(err);
        };
    return 0;
};
test "set_args" {
    const a = std.testing.allocator;
    const expect = std.testing.expect;
    var args: SarusSlurmArgs = SarusSlurmArgs.init(a);
    defer args.deinit();

    try args.set_environment("bla");
    try expect(eql(u8, args.environment orelse "", "bla"));
}
```

Zig?

- Why Not?

- Not yet 1.0
- smaller community

- Why Zig after all

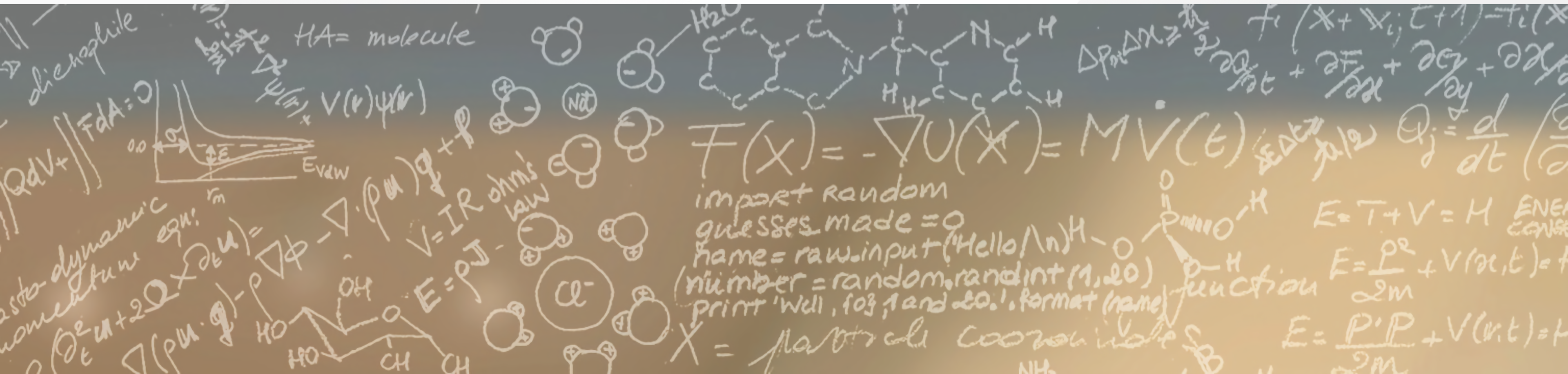
- Error traces simplify debugging
- no dependencies of plugin
- nicer and safer than C
- fast recompile/test
- Everything changes: HW, libraries, OS,...
- following language changes can be part of the development work
- zig fmt took care of `for (it) |el, index| {}` -> `for (it, 0..) |el, index| {}`
- we are not interested in experimental features (async / await)
- some large projects like bun.sh, tigerbeetle.com, [match](https://matcha.zone),... use it already



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Zig Alternatives

Rust

- a better C++
- focused on safety
- large community
- cdylib target can create plugins that should work
- bindings to call C, [bindgen](#) can help generating them
- Stacktraces but no error traces
- large and complex language



[Install](#) [Learn](#) [Playground](#) [Tools](#) [Governance](#) [Community](#) [Blog](#) [English \(en-US\)](#)

Rust

GET STARTED

[Version 1.76.0](#)

A language empowering everyone
to build reliable and efficient software.

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

Build it in Rust

In 2018, the Rust community decided to improve the programming experience for a few distinct domains (see [the 2018 roadmap](#)). For these, you can find many high-quality crates and some awesome guides on how to get started.



Command Line

Whip up a CLI tool quickly with Rust's robust ecosystem. Rust helps you maintain your app with confidence and distribute it with ease.

BUILDING TOOLS



WebAssembly

Use Rust to supercharge your JavaScript, one module at a time. Publish to npm, bundle with webpack, and you're off to the races.

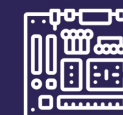
WRITING WEB APPS



Networking

Predictable performance. Tiny resource footprint. Rock-solid reliability. Rust is great for network services.

WORKING ON SERVERS



Embedded

Targeting low-resource devices? Need low-level control without giving up high-level conveniences? Rust has you covered.

STARTING WITH EMBEDDED

FEATURES

[Current ISO C++ status](#)

[Upcoming ISO C++ meetings](#)

[Upcoming C++ conferences](#)

[Compiler conformance status](#)

TAGS

[basics](#)

[intermediate](#)

[advanced](#)

[experimental](#)

UPCOMING EVENTS

ISO C++ committee meeting

March 18-23, Tokyo, Japan

ACCU 2024

April 17-20, Bristol, UK

using std::cpp 2024

April 24-26, Leganes, Spain

C++ Now 2024

May 7-12, Aspen, CO, USA

ISO C++ committee meeting

June 24-29, St. Louis, MO, USA

C++ on Sea

July 2-5, Folkestone, Kent, UK

Sign In / Suggest an Article Register

Get Started! Tour Core Guidelines Super-FAQ Standardization About

News, Status & Discussion about Standard C++

Follow All Posts

The home of Standard C++ on the web — news, status and discussion about the C++ standard on all compilers and platforms.

Recent Highlights

C++23: Allocator Related Changes – Sandor Dargo

By Blog Staff | Mar 13, 2024 01:48 PM

C++20 Concepts Applied – Safe Bitmasks Using Scoped Enums – Andreas Fertig

By Blog Staff | Mar 11, 2024 01:14 PM

Announcing the full ACCU 2024 Conference schedule – ACCU & ShavedYaks

By philsquared | Mar 11, 2024 11:57 AM

Aggregates: C++17 vs. C++20 – Andreas Fertig

By Blog Staff | Mar 9, 2024 01:13 PM

Data Structures and Algorithms with the C++ STL – John Farrier

By John Farrier | Mar 8, 2024 11:19 AM

Recent CppCast Podcasts

Psychology and Starting Out as a Developer

Date: Mon, 11 Mar 2024

Compiler Explorer Revisited

Date: Fri, 23 Feb 2024

Teaching and Training Modern C++

Date: Fri, 9 Feb 2024

Reflection for C++26

Date: Fri, 26 Jan 2024

Recent C++ Weekly Podcasts

s a `for` Loop?

Date: Fri, 1 Mar 2024

s a `do` Loop?

Date: Wed, 28 Feb 2024

Turbocharge Your Build With Mold?

Date: Mon, 26 Feb 2024

re `if`/`else` Statements?

Date: Fri, 23 Feb 2024

Articles & Books

C++23: Allocator Related Changes – Sandor Dargo

By Blog Staff | Mar 13, 2024 01:48 PM

C++20 Concepts Applied – Safe Bitmasks Using Scoped Enums – Andreas Fertig

By Blog Staff | Mar 11, 2024 01:14 PM

Aggregates: C++17 vs. C++20 – Andreas Fertig

By Blog Staff | Mar 9, 2024 01:13 PM

Data Structures and Algorithms with the C++ STL – John Farrier

By John Farrier | Mar 8, 2024 11:19 AM

Using std::expected from C++23 – Bartlomiej Filipek

By Blog Staff | Mar 7, 2024 12:59 PM

Product News

Seastar, ScyllaDB, and C++23

By Jordi Mon Companys | Feb 17, 2024 05:36 AM

PVS-Studio 7.29: Boost smart pointers, plugin for Qt Creator on macOS

By Andrey Karpov | Feb 13, 2024 06:49 AM

mp-units 2.1.0 released – Mateusz Pusz

By Mateusz Pusz | Feb 6, 2024 12:40 PM

CppDepend 2024.1 Released! - Unveiling New Features and Improvements

By CppDepend Team | Feb 4, 2024 06:03 AM

C++

- improves on C
- largest community
- already well known by all
- larger/complex language
- stacktrace possible
- more "hairs" (due to the long history)
- at least libstdc++ dependency

CSCS

17

ETH zürich

Thanks