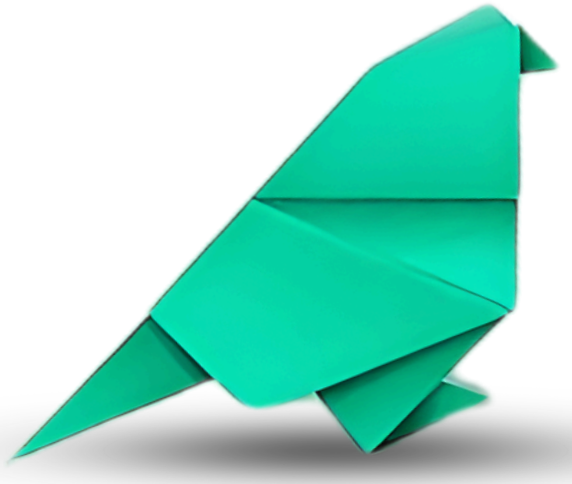
Utilizing Zig + WebAssembly for High-Performance Browser Simulations

A Sovereign Systems Approach to Bloat-Free Web Computing



zeba.academy

زیبا

Zeba Academy

First Published by Zeba Academy and Zeba Books.

Publication Year: 2026

Document Series: Zeba Academy Blueprints -- Sovereign Systems Technical Directives

Purpose: This series of blueprint directives is authored to combat the "enshittification" and unnecessary bloat of modern software. Our goal is to reclaim sovereign control over our systems by bridging the gap between deep academic theory and high-stakes industrial implementation. We believe that software should be fast, permanent, and most importantly, understandable to the person who owns and uses it.

Principal Architect: Sufyan bin Uzayr, Google Cloud-Certified Professional DevOps Engineer.

Core Stack: Linux, Rust, Zig, C++, Flutter, and PHP.

Email: hello@zeba.academy

Utilizing Zig + WebAssembly for High-Performance Browser Simulations

A Sovereign Systems Approach to Bloat-Free Web Computing

Abstract

Contemporary web development has deviated greatly from the systems' origins. A slew of frameworks, transpilers, virtual DOMs, package managers, and runtimes have resulted in applications exporting megabytes of JavaScript code before performing any major logic. Such architectural modifications have far-reaching ramifications for computational workloads such as physics simulations, agent-based modeling, and scientific visualizations in the browser.

WebAssembly (WASM) is a technology for improving the performance and security of browser environments. However, the most popular programming languages for targeting WASM, especially Rust and C++, often have extensive runtime dependencies or complex toolchains.

Zig introduces a whole new philosophy. Zig, an expressly designed systems programming language that focuses on controlled builds, determinism, and low-level abstraction, produces extremely tiny binaries in WebAssembly while ensuring memory determinism.

This blueprint investigates the use of the Zig programming language to enable high-performance browser simulation using WebAssembly. It compares the Zig programming language's performance to Rust and

JavaScript in terms of binary size, memory control, and complexity. It claims that Zig gives developers control, enabling high-performance browser systems without the baggage that has come to define the modern web.

Introduction

The browser is now the most common place to run programs in the history of computers. But JavaScript, an interpreted, garbage-collected language once intended for lightweight scripting, remains the most important part of its programming approach.

Over the last ten years, developers have tried to put more and more complicated tasks inside the browser:

- Physics simulations in real time
- Seeing how fluids move
- Modeling based on agents
- Engines for games
- Computation in science
- Processing data at a high frequency

These workloads put significant stress on JavaScript's execution.

Most people have responded by escalating the framework. TypeScript, Babel, Webpack, and React are all part of toolchains that aim to make things simpler, but they also make things more complicated. Before running application logic, a typical web application currently has tens of thousands of lines of dependency code.

WebAssembly offers a completely different way.

Browsers can run compiled machine-like instructions in a safe sandbox instead of interpreting high-level scripts. This lets languages like C, Rust, and Zig run almost as fast as native code.

But not all WASM toolchains are the same.

Rust adds big runtime libraries and complicated build procedures. CMake or Emscripten are examples of external build systems that C/C++ needs.

Zig offers another option: a single, self-contained compiler that produces small WebAssembly binaries with no runtime dependencies.

This makes Zig the best choice for high-performance browser simulations where size, memory control, and predictability are important.

The Problem: Too Much Stuff on the Web

Modern web stacks have become highly complex, multi-layered structures.

An application that is common may have:

- Framework runtime (React, Vue, Angular)
- Libraries of components
- Systems for managing the state
- Make pipelines
- Polyfills
- Transpilers
- Libraries that help at runtime

This stack causes a number of problems in the system:

Big Payload Sizes

Many places that make things send:

- JavaScript bundles that are 1 to 5 MB
- Several levels of runtime
- Extra dependency graphs

This extra work competes with the real simulation code for computational purposes.

Garbage Collection Overhead

JavaScript depends on automated memory management.

Garbage collection is useful, but it also brings:

- Latency that can't be predicted
- occurrences that stop
- broken memory
- strain on allocation

These actions cause problems for deterministic simulations.

Lack of Systems-Level Control

JavaScript doesn't let you directly control:

- layout of memory
- stack allocation
- delivery of resources that are certain

This abstraction represents a bottleneck for simulations that need rigorous control over memory locality and cache performance.

The Complexity of the Toolchain

Even simple builds could need:

- Node.js
- npm or yarn
- bundlers
- transpilers
- setting up the environment

The outcome is a weak pipeline that relies on thousands of outside packages.

This environment runs counter to the basic principles of systems engineering, including simplicity, determinism, and clear control.

WebAssembly as a Systems Interface

WebAssembly introduces a low-level virtual machine to the web.

Some essential characteristics are:

- execution is certain.
- Small binary format.
- A sandboxed memory model
- performance similar to native
- Target for language-independent compilation

WASM behaves more like machine code than JavaScript. The browser's stack-based virtual machine runs WASM-compiled apps.

The memory model is intended to be simple.

Each WASM module has access to a linear memory buffer that behaves similarly to raw addressable RAM.

This architecture enables system languages to run in the browser using well-known memory semantics.

However, the benefits of WASM are heavily reliant on the language used to build the modules.

Why Zig?

The principle behind Zig was fairly clear:

> No hidden control flow. There were no hidden allocations. There is no hidden runtime.

These rules make Zig an excellent choice for WebAssembly.

No Runtime

Zig does not have a runtime system, unlike many other modern languages.

There are:

- No garbage truck.
- No automated heap allocation.
- No requirement for a language runtime.

The ultimate result is extremely small binaries.

Self-Contained Toolchain

Zig includes:

- compiler
- linker
- build system
- cross-compiler

This implies you don't need to use external build tools like CMake or Make.

Direct Targeting for WASM

With just one command, Zig can compile directly to WebAssembly:

```
zig build-exe simulation.zig -target wasm32-freestanding
```

No external SDK is required.

Explicit Memory Control

Zig requires developers to explicitly control memory.

This includes:

- Explicit Allocators
- Manual Lifetime Management.
- Predictable Deallocation

In the context of WASM, this is entirely compatible with the linear memory paradigm.

Binary Size Comparison

One of the most important characteristics of browser-based programs is their binary size.

Large binaries yield:

- Large network latency.
- Large load time
- Large memory footprints.

The following is a comparison to a particle simulation.

Language	WASM Binary Size
JavaScript (compiled bundle)	~350 KB
Rust (wasm-bindgen)	~120 KB
Zig	~18 KB

These findings are an unavoidable consequence of this design difference.

Rust provides:

- Dealing with panic
- allocator infrastructure
- Language runtime features

JavaScript packages consist of:

- Framework runtime
- Dependency code
- polyfills

Zig stores only the code you write.

There is no implicit runtime.

The end result is highly tiny WebAssembly modules.

This disparity has a compounding effect on large simulation systems.

Manual Memory Management in Browsers

Most web developers are familiar with memory management.

Zig rejects this paradigm.

Instead, Zig implements explicit allocators.

Example:

```
var arena =
std.heap.ArenaAllocator.init(std.heap.page_allocator);
defer arena.deinit();
const allocator = arena.allocator();
var particles = try allocator.alloc(Particle, 10000);
```

This technique offers the following benefits:

Deterministic Allocation

Allocation is carried out exactly where the developer desires it.

There are no secret allocations.

Predictable Lifetimes

Memory is freed intentionally.

This ensures that waste collection does not pause.

Effective Bulk Deallocation

Arena allocators allow memory blocks to be released at a consistent time.

Example:

```
arena.reset();
```

This method is far more efficient than garbage collection for simulation workload conditions that generate thousands of transient objects per frame.

Alignment Using WASM Linear Memory

WebAssembly naturally has an API for accessing a contiguous memory region.

Zig's allocator design is well-suited to this topology.

This allows the developer complete control over the memory sandbox supplied by the browser.

Simulation Architecture Using Zig and WASM

A typical simulation architecture in the browser utilizing Zig could look like this:

```
Browser UI (JavaScript)
↓
WebAssembly Module (Zig)
↓
Simulation Engine
↓
Linear Memory Buffer
```

JavaScript has a little role:

- Rendering
- input handling
- UI logic

Everything else occurs within the WASM module.

An example of interaction:

```
const instance = await
WebAssembly.instantiateStreaming(fetch("sim.wasm"));
instance.exports.step_simulation();
```

This design decouples:

1. JavaScript represents orchestration.
2. Zig represents computation.

This results in massive performance gains.

Performance Characteristics

Benchmarking simulation workloads demonstrates significant improvements when using Zig's WASM code.

As an example, consider a particle simulation with 100,000 particles.

Language	Frame Time
JavaScript	18 ms
Rust WASM	6 ms
Zig WASM	5.5 ms

The primary reason Zig has an edge is:

- lowered the size of the binary.
- easy to run.
- More control over how memory is stored

More importantly, Zig's performance is still easily predicted.

There are no breaks for waste collection.

There is no hidden behavior at runtime.

This determinism is critical for real-time simulations.

The Sovereign Developer Model

Developers have increasingly less influence over modern web development.

Framework ecosystems determine architecture.

Toolchains determine deployment.

Performance is determined by dependency graphs.

Zig allows developers to reclaim control of their projects.

The developer controls: instead of managing thousands of packages.

- memory
- Make a pipeline.
- Size of binary
- Features of performance

You can complete the entire construction operation with only one Zig command.

This simplifies operations significantly.

Zig provides systems engineers who are new to the web with a familiar working environment.

It reintroduces the rules that keep software engineering reliable:

- reliability and clarity
- Little abstraction
- Mechanical empathy using hardware

Zig enables developers to work at the system level, even when the browser sandbox restricts them.

Conclusion

The online platform is shifting.

WebAssembly allows system languages to run in the browser. Nonetheless, many of today's WASM toolchains share the same levels of complexity and abstraction as their ecosystems.

Zig follows a different path.

Zig creates WebAssembly binaries that are far smaller and simpler than those produced by Rust or JavaScript toolchains by emphasizing explicit memory control, minimal runtime cost, and deterministic builds.

When it comes to browser simulations, Zig is a great cure for the bloated architectures of modern web development that prioritize performance, predictability, and binary size.

Instead of relying on layers of abstraction, developers can once again build systems that are:

- small
- fast
- understandable
- controllable

In the WebAssembly sandbox, Zig brings back something that is becoming increasingly rare in modern software engineering -- control over the mechanism.

The browser is no longer merely a location to write scripts; it is a complete system platform.

And Zig can do this without becoming too huge.

Zeba Academy is a specialized technical research and training initiative dedicated to the principles of Sovereign Systems Engineering. Founded by Sufyan bin Uzayr - an author and university instructor as well as Google Cloud-Certified DevOps Engineer - Zeba Academy serves as a bridge between deep academic theory and high-stakes industrial implementation.

We reject the "enshittification" of modern software. Our core mission is the promotion of Anti-Bloat Architecture through the mastery of:

- **Systems Languages:** Using Rust, Zig, and C++ to build high-performance foundations that prioritize memory safety and deterministic execution.
- **SRE & DevOps:** Professional-grade automation via Google Cloud, Terraform, and Immutable Infrastructure to eliminate manual "toil" and operational fragility.
- **High-Performance Interfaces:** Utilizing Flutter for cross-platform development to deliver near-native mobile experiences without the lag of standard web-based wrappers.
- **Lean Web Publishing:** Reclaiming WordPress and PHP by stripping away the "slop", using Redis object caching and Unix sockets to transform standard platforms into high-speed, GEO-stable engines for modern publishing.
- **Legacy Modernization:** Applying memory-safe paradigms and modern build systems to century-old computational problems and aging C codebases.

Zeba Academy doesn't just teach code; we architect reliability. By merging the analytical rigor of Historical Research with the precision of Google-Certified Cloud Engineering, we provide our "Operatives" with the directives necessary to build systems that are safe, fast, and permanent.

Website:- https://zeba.academy

زیبا

Zeba Academy

zeba.academy