

# The Coming OS Revolution

k23 - Next-Gen WASM Operating System

---

Jonas Kruckenberg

# Jonas Kruckenberg

## Research engineer

- Originally self-taught & web developer
- Tauri
- TC39 Invited Expert
- ~2 years OS development



01

The OS Revolution

02

Rust!

03

Snacks & Drinks

# Why a new OS?

# BROWSER WARS

# The best platform?

## Low Barrier To Entry

- The Web was easy to develop for
- The peoples GUI



Easy & Cheap to Get Started

## Easy Distribution

- No App Stores, Complex Bundling, Installers or Dependency Issues.
- Instant Availability, Easy to Update



Easy & Cheap to Operate

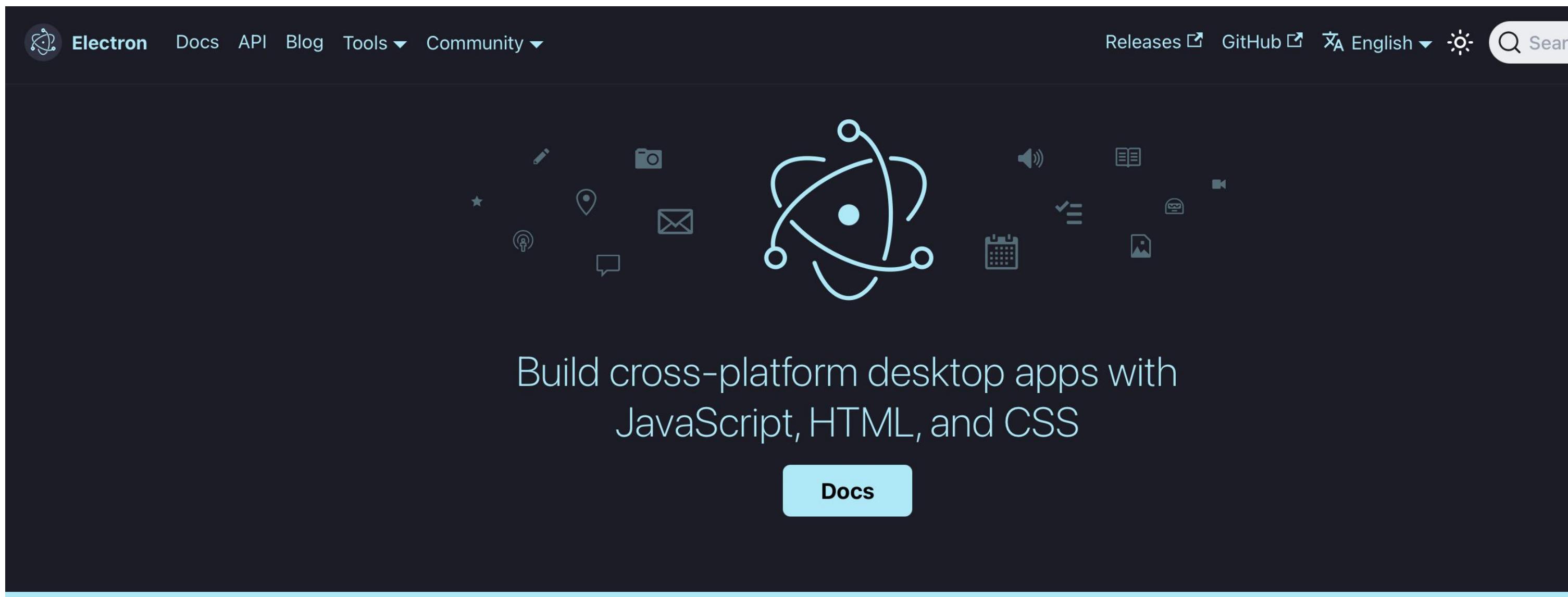
## Developer Experience

- Browser DevTools, 3rd Party Tools, and Frameworks
- High Level of Polish



Easy & Cheap to Troubleshoot

# Case in point: Electron



2013

INITIAL RELEASE

847,354

WEEKLY DOWNLOADS

115,000

STARS ON GITHUB



## Web Technologies

Electron embeds Chromium and Node.js to enable web developers to create desktop applications.



## Cross Platform

Compatible with macOS, Windows, and Linux, Electron apps run on three platforms across all supported



## Open Source

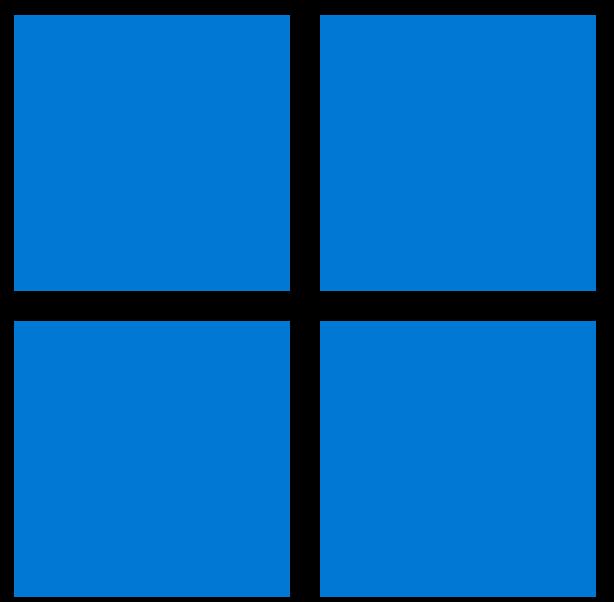
Electron is an open source project maintained by the [OpenJS Foundation](#) and an active community of contributors.

# Operating Systems Today



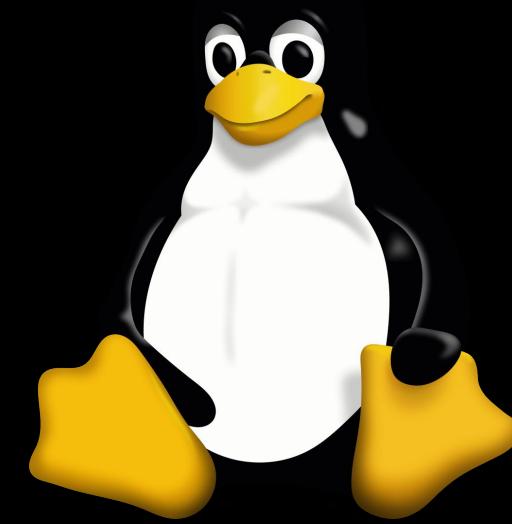
macOS

Building app in spite of Apple  
Increasing hostility  
Still incredibly niche



Windows

3 spywares in a trenchcoat  
Even system settings is web tech  
Abandoned developers



Linux

Well...  
Linux is a hobby not an OS  
Ecosystem is way too fractured

# Issues with the Web

## Too Complicated for Simple Apps

- So much tooling, new framework every day
- Not the peoples GUI anymore

# Issues with the Web

## Too Complicated for Simple Apps

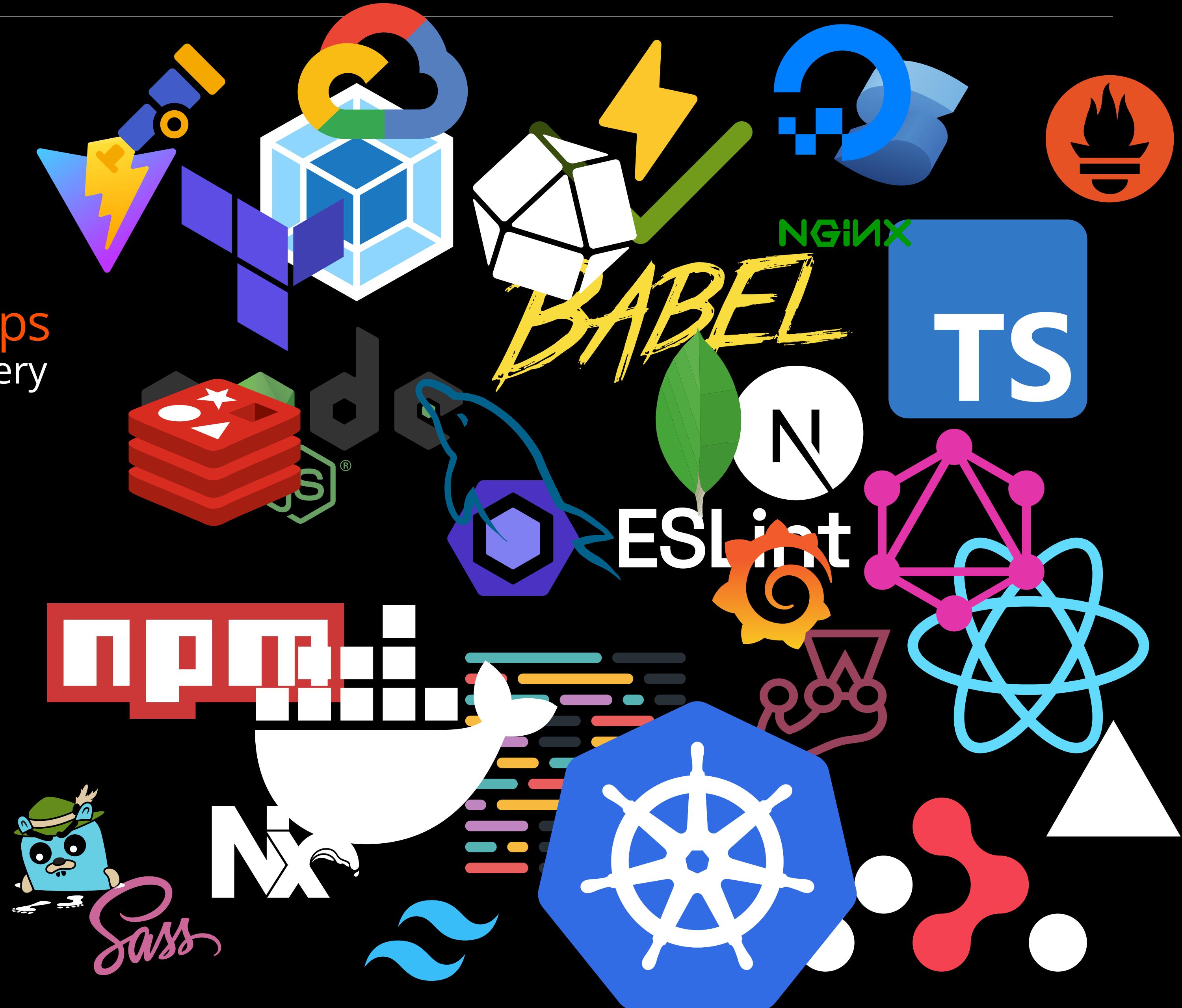
- So much tooling, new framework every day
- Not the peoples GUI anymore



# Issues with the Web

# Too Complicated for Simple Apps

- So much tooling, new framework every day
  - Not the peoples GUI anymore



# Issues with the Web

## Too Complicated for Simple Apps

- So much tooling, new framework every day
- Not the peoples GUI anymore

## Not Powerful Enough for Complex Apps

- Limited access to hardware, resource limits
- JavaScript performance

# Issues with the Web

There is still demand for “desktop” apps

It is getting progressively harder to build them

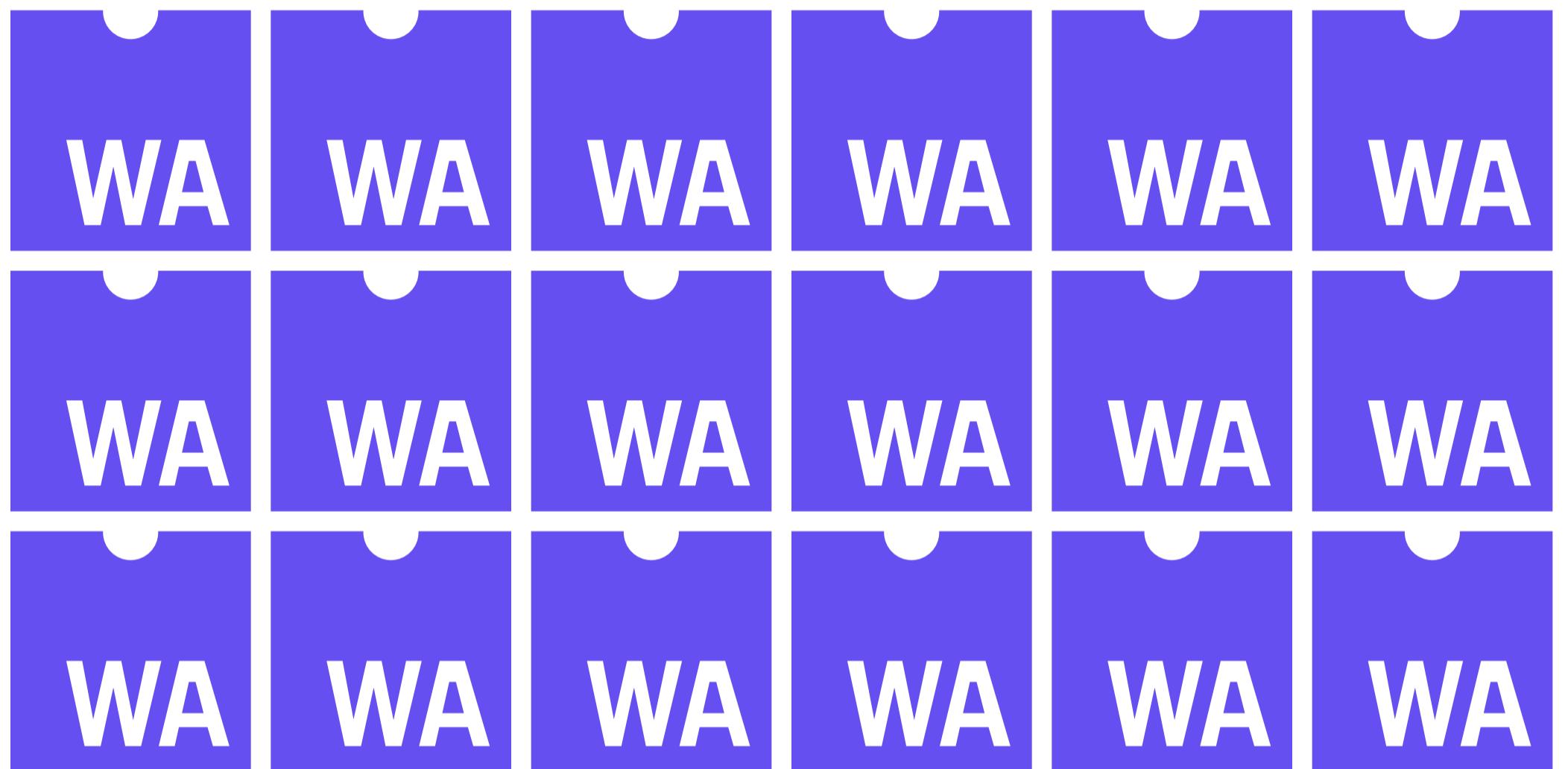
What can we do?

Make apps  
great again

# k23

## WebAssembly (WASM)

- Strong security, Permissions,
- Statically analyzable
- Just-In-Time Compilation



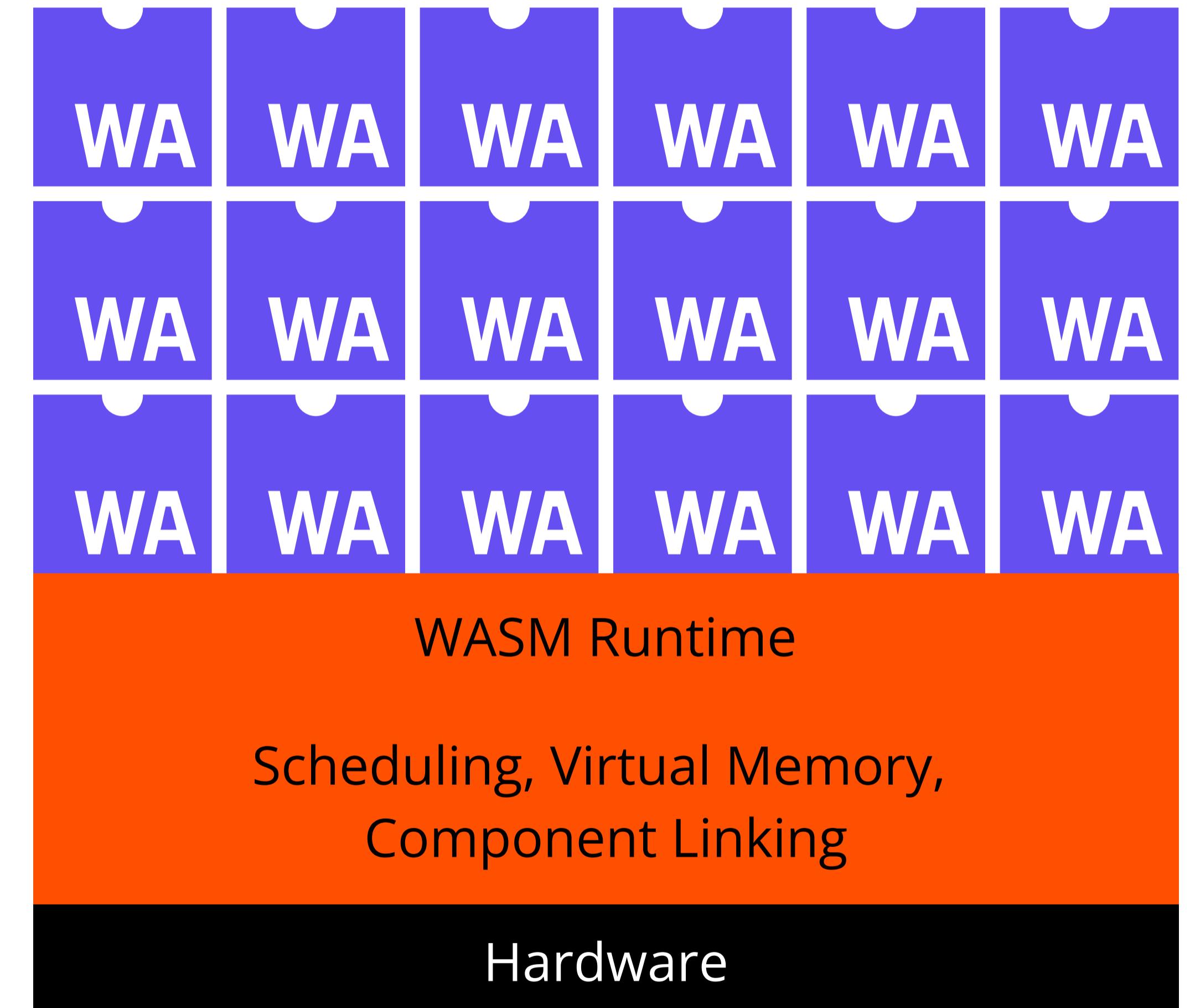
# k23

## WebAssembly (WASM)

- Strong security, Permissions,
- Statically analyzable
- Just-In-Time Compilation

## Microkernel

- Security & Sandboxing
- Naturally follows from WASM



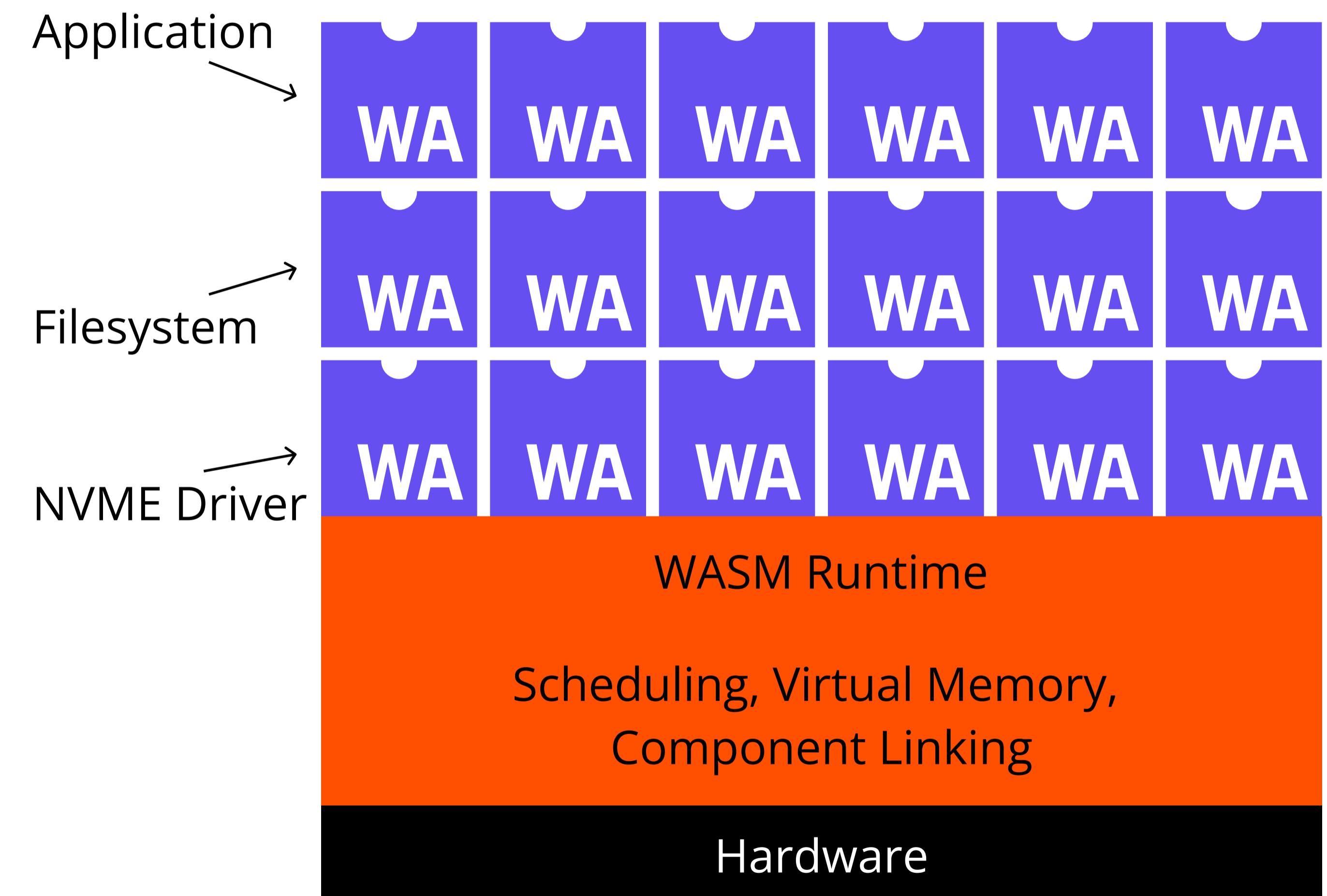
# k23

## WebAssembly (WASM)

- Strong security, Permissions,
- Statically analyzable
- Just-In-Time Compilation

## Microkernel

- Security & Sandboxing
- Naturally follows from WASM



# k23

## WebAssembly (WASM)

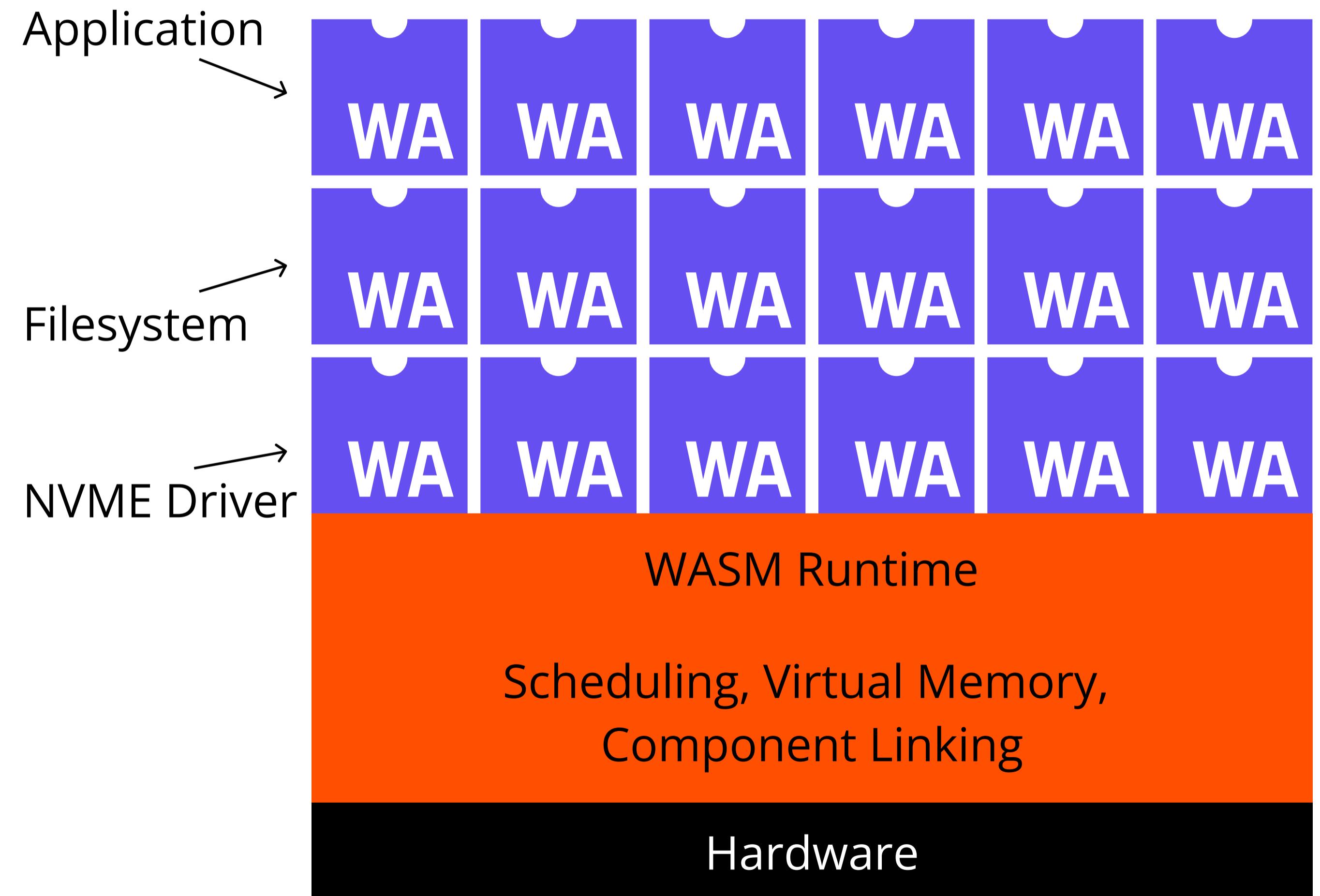
- Strong security, Permissions,
- Statically analyzable
- Just-In-Time Compilation

## Microkernel

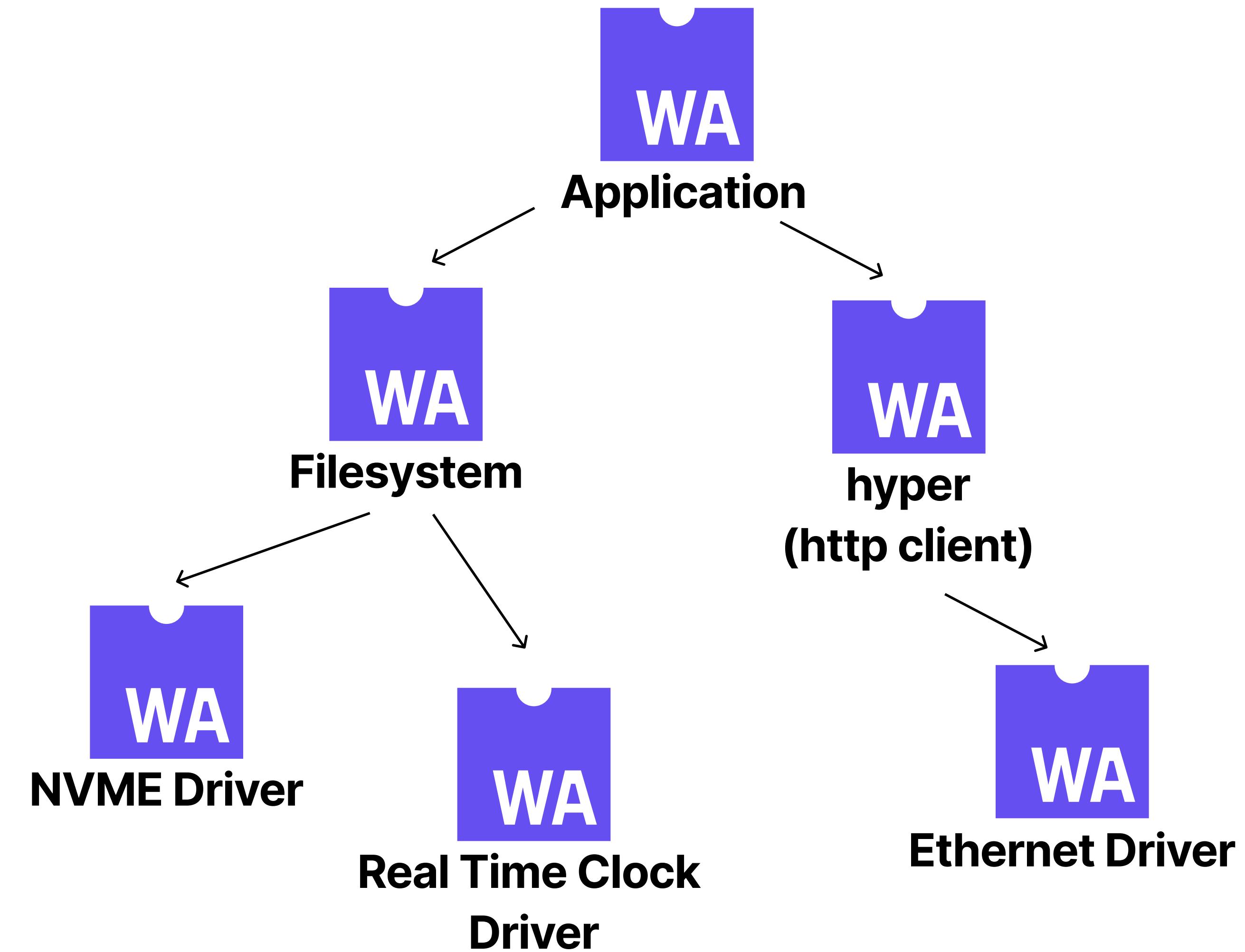
- Security & Sandboxing
- Naturally follows from WASM

## Nix-style Package Management

- Reproducible, declarative and reliable
- No more dependency issues
- Single remote registry



# Package Management



# The best of both worlds

## Performance & Flexibility of Native

- Hardware access, language agnostic
- No hardware restrictions

## Great Experience of the Web

- Great tooling and debugging experience
- Easy distribution, updating and management

Integrating the Compiler and OS  
unlocks a lot of opportunities

# Open Source

The screenshot shows the GitHub repository page for 'k23' owned by 'JonasKruckenberg'. The repository is described as an 'Experimental WASM Microkernel Operating System'. It has 25 branches and 3 tags. The main branch has 441 commits. Key commits include 'refactor: remove lz4 kernel compression' and 'fix: dont ignore all errors in ci tests'. The repository has 271 stars, 5 watchers, and 11 forks. It features three releases, with the latest being version 0.0.3 from Sep 4. A 'Sponsor this project' button is visible at the bottom.

JonasKruckenberg / k23

Type / to search

Code Issues 33 Pull requests 5 Discussions Actions Projects 1 Security Insights Settings

k23 Public

main 25 Branches 3 Tags Go to file Add file Code

JonasKruckenberg refactor: remove lz4 kernel compression 1a79774 · 4 days ago 441 Commits

.github fix: dont ignore all errors in ci tests 2 months ago

kernel fix: temp comment out deadlocking backtrace code (#128) last week

libs chore: update Rust nightly version to 2024-11-20 (1.84) ... last week

loader refactor: remove lz4 kernel compression 4 days ago

manual refractor: simplify repo & build system (#102) 2 months ago

tests feat: run WASM spec test suite (#30) 3 months ago

.gitignore remove target dir 8 months ago

.gitmodules feat: run WASM spec test suite (#30) 3 months ago

Cargo.lock refactor: remove lz4 kernel compression 4 days ago

Cargo.toml refactor: remove lz4 kernel compression 4 days ago

README.md fmt (#141) 2 weeks ago

deny.toml refactor: break up kstd crate into multiple smaller crates (...) 2 months ago

About

Experimental WASM Microkernel Operating System

[jonaskruckenberg.github.io/k23/](https://jonaskruckenberg.github.io/k23/)

research wasm operating-system microkernel

Readme Activity 271 stars 5 watching 11 forks

Releases 3

0.0.3 Latest on Sep 4 + 2 releases

Sponsor this project

JonasKruckenberg Jonas Kruc...

<https://github.com/JonasKruckenberg/k23>

# The Plan



# The Plan

Temporary Goal  
**Backend Services, docker, k8s, deployment**



Today

# The Plan

Phase I WASM

WASM Runtime v2, Virtual  
Memory Management.  
**Running basic programs**

Temporary Goal

**Backend  
Services, docker,  
k8s, deployment**



Today

# The Plan

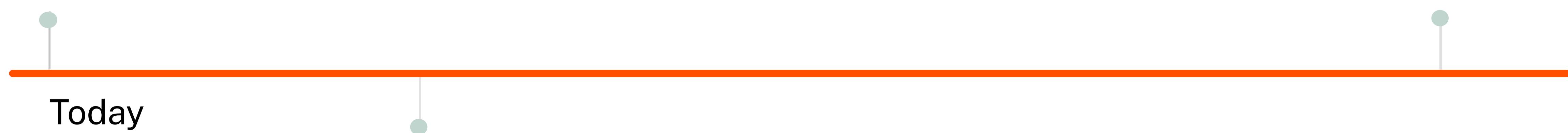
Phase I WASM

WASM Runtime v2, Virtual  
Memory Management.  
**Running basic programs**

Temporary Goal

**Backend**

**Services, docker,  
k8s, deployment**



Today

Phase II Concurrency

Cooperative Scheduler,  
Async Rust, SMP support  
and WASM threads  
proposal.

**Running multiple  
programs**

# The Plan

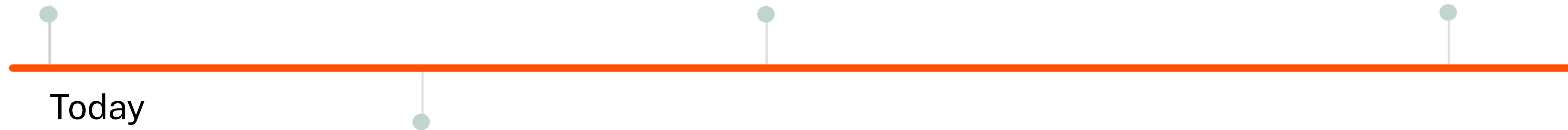
Phase I WASM

WASM Runtime v2, Virtual  
Memory Management.  
**Running basic programs**

Phase II 1/2 *Kotlin*

Implement WASM GC,  
and Exception  
Handling proposals.  
Demonstrate support  
for languages  
**Running complex  
programs**

Temporary Goal  
**Backend Services, docker, k8s, deployment**



Today

Phase II Concurrency

Cooperative Scheduler,  
Async Rust, SMP support  
and WASM threads  
proposal.

**Running multiple  
programs**

# The Plan

Phase I WASM

WASM Runtime v2, Virtual  
Memory Management.

**Running basic programs**

Phase II 1/2 *Kotlin*

Implement WASM GC,  
and Exception

Handling proposals.

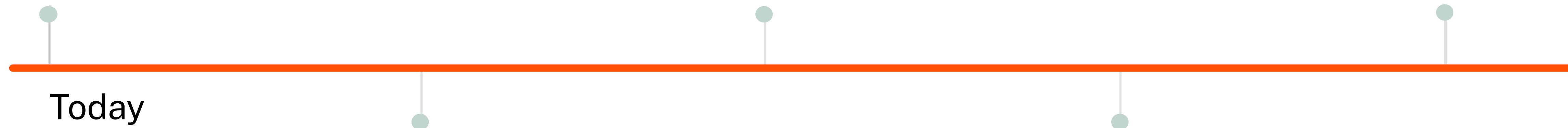
Demonstrate support  
for languages

**Running complex  
programs**

Temporary Goal

**Backend**

**Services, docker,  
k8s, deployment**



Today

Phase II Concurrency

Cooperative Scheduler,  
Async Rust, SMP support  
and WASM threads  
proposal.

**Running multiple  
programs**

Phase III Component Model

Implement WASM  
component model, Package  
Registry, and Dynamic  
Linking.

**Running 3rd party  
programs**





yay!

# The Good

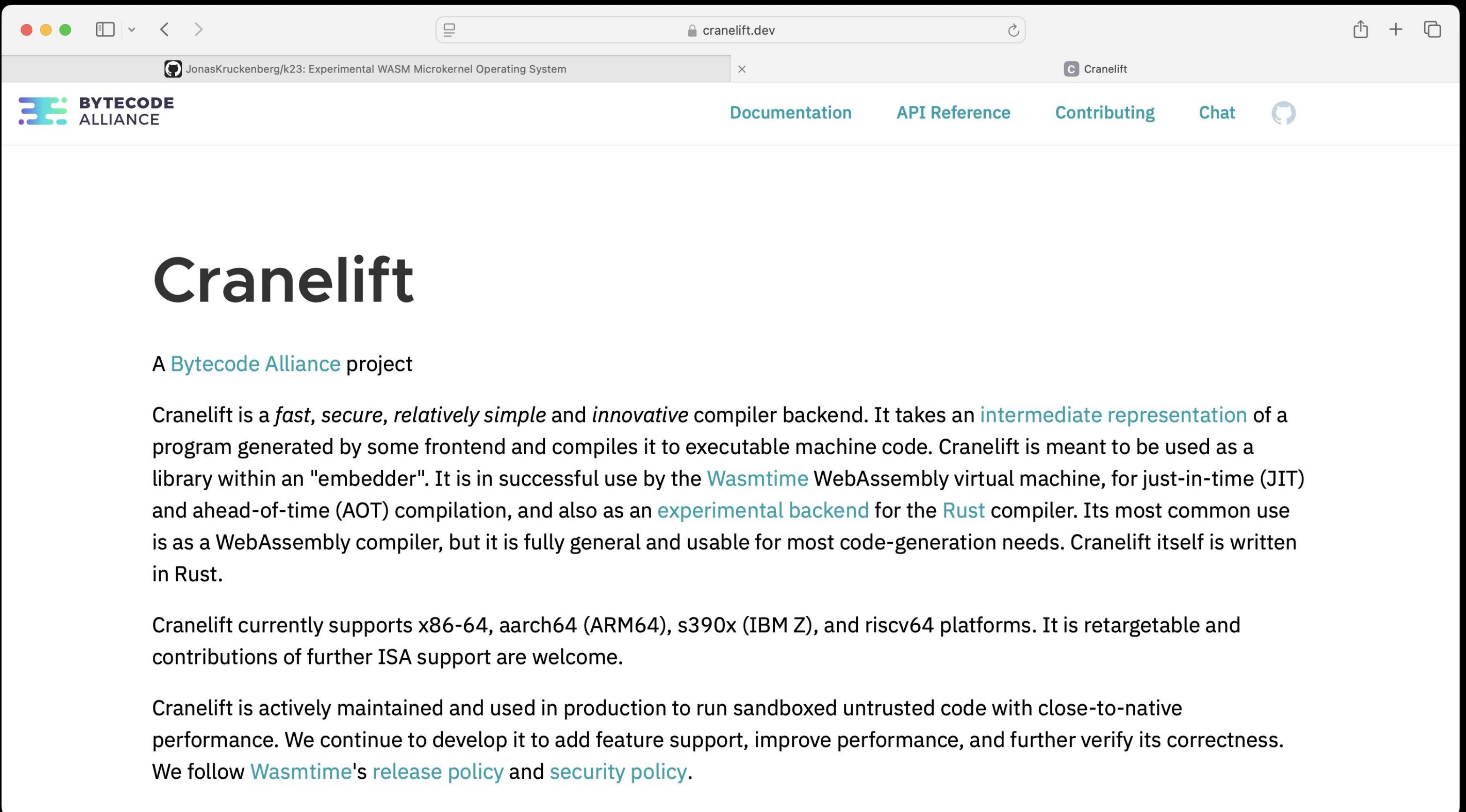
## Superpower Cargo

- Deduplicate effort & Pool resources
- Stronger as a community
- Be productive quickly

## Static Guarantees

- Low-level has very few safety nets
- As much static guarantees about runtime behavior as possible
- **Embedded & low-level can be multithreaded too!**

# Example: cranelift

A screenshot of a web browser displaying the Cranelift project page. The address bar shows 'cranelift.dev'. The page header includes the Bytecode Alliance logo and navigation links for 'Documentation', 'API Reference', 'Contributing', and 'Chat'. The main content features a large heading 'Cranelift', described as a 'Bytecode Alliance project'. It explains Cranelift's purpose as a fast, secure, relatively simple, and innovative compiler backend, mentioning its use in Wasmtime and Rust, and its support for various platforms like x86-64 and aarch64. The page also discusses its active maintenance and adherence to Wasmtime's release and security policies.

## Cranelift

A [Bytecode Alliance](#) project

Cranelift is a *fast, secure, relatively simple* and *innovative* compiler backend. It takes an [intermediate representation](#) of a program generated by some frontend and compiles it to executable machine code. Cranelift is meant to be used as a library within an "embedder". It is in successful use by the [Wasmtime](#) WebAssembly virtual machine, for just-in-time (JIT) and ahead-of-time (AOT) compilation, and also as an [experimental backend](#) for the [Rust](#) compiler. Its most common use is as a WebAssembly compiler, but it is fully general and usable for most code-generation needs. Cranelift itself is written in Rust.

Cranelift currently supports x86-64, aarch64 (ARM64), s390x (IBM Z), and riscv64 platforms. It is retargetable and contributions of further ISA support are welcome.

Cranelift is actively maintained and used in production to run sandboxed untrusted code with close-to-native performance. We continue to develop it to add feature support, improve performance, and further verify its correctness. We follow [Wasmtime's release policy](#) and [security policy](#).

- ## JIT Compiler Backend
- Maintained by Bytecode Alliance
  - Used in Wasmtime
  - Proven and well designed

# The Bad

## Missing Support

- Plenty of nightly features
- Missing features in Cargo (bindeps, build-std)
- no\_std Rust

## Custom Tooling

- Build tooling
- Debuggers & Profiling

# Example: ktest & unwind2

```
new *
#[test]
fn feature() {
    assert!(false);
    |
    let result :Result<(), Box<dyn ...>> = unwind2::catch_unwind(|| {});
    unwind2::begin_panic(Box::new(x: ()).unwrap());
}
```

## Problem

- **libtest** doesn't support `no_std`
- Assertions panic
- `no_std` means **panic-abort**
- **libunwind** is horrific

## Solution

- Custom Test Runner
- Custom **libunwind => unwind2**

# Example: nightly features

```
// To avoid code duplication, the `ktest` crate just depend on the kernel, overriding the `kmain`  
// function with its own test runner.  
#![no_std]  
#![no_main]  
#![allow(internal_features)]  
#![feature(used_with_arg, naked_functions, thread_local, allocator_api)]  
#![feature(panic_can_unwind, std_internals, fmt_internals)]  
  
extern crate alloc;  
  
2 usages  ↳ Jonas Kruckenberg  
pub mod allocator;  
4 usages  ↳ Jonas Kruckenberg
```

## Nightly features

- Many features are stuck in nightly
- Sometimes unstable
- Mostly badly documented
- Compiler crashes

# Example: allocator API

```
9 usages • Jonas Kruckenberg
pub type GuestVec<T> = Vec<T, GuestAllocator>;
new *
fn example() {
    let vec: GuestVec<u8> = GuestVec::new_in(guest_allocator());
}
```

## Allocator API

- New `_in` methods on collections
- Allows differentiating ownership on type-level

# The Coming OS Revolution

Small but growing community  
Learn & have fun!



**DISCORD**

# Snacks & Drinks

Jonas Kruckenber

𝕏 @jonaskruckenber

Ⓜ @unsafe@webtoo.ls

🐦 @jonasKruckie