# 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

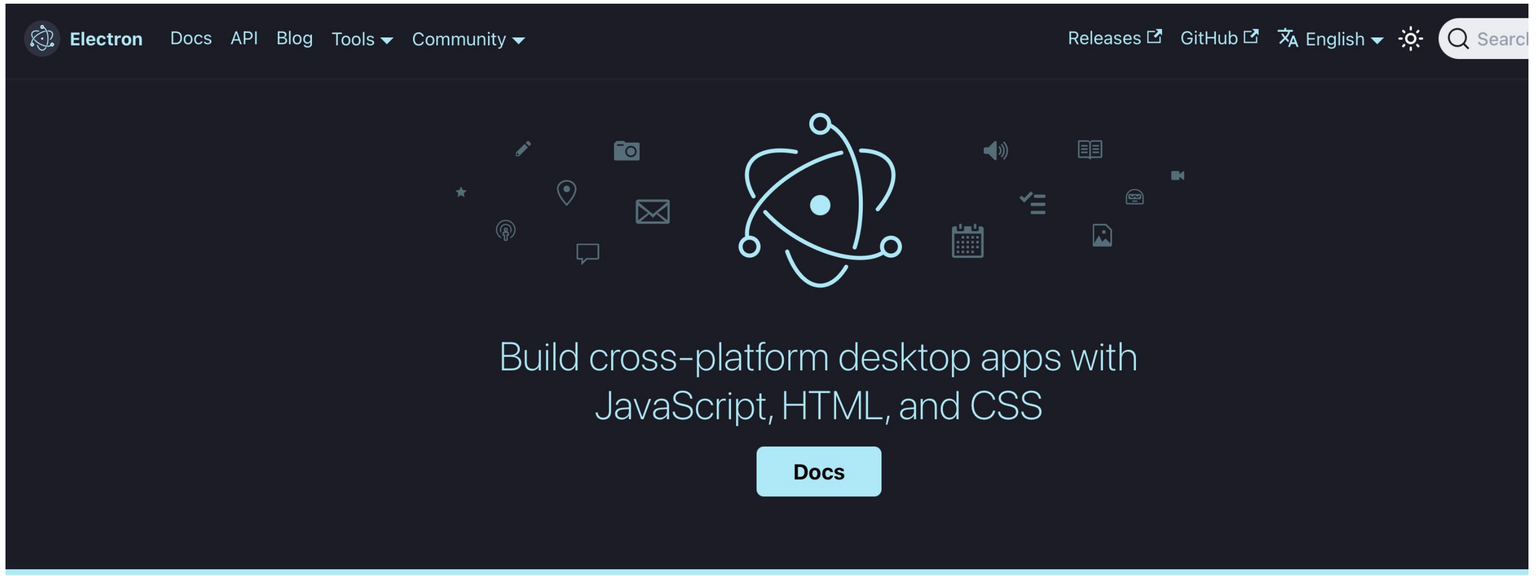
→ Easy & Cheap to Operate

## Developer Experience

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

→ Easy & Cheap to Troubleshoot

# Case in point: Electron




**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.

2013

INITIAL RELEASE

847,354

WEEKLY DOWNLOADS

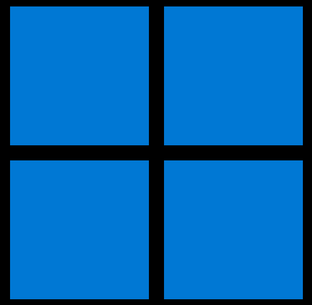
115,000

STARS ON GITHUB

# 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

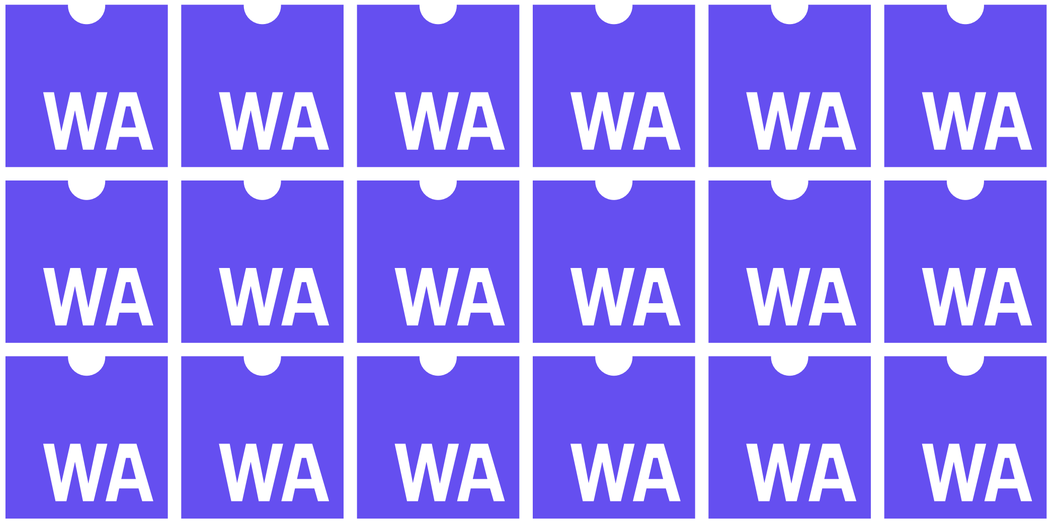
What can we do?

# Make apps great again

k23

## WebAssembly (WASM)

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



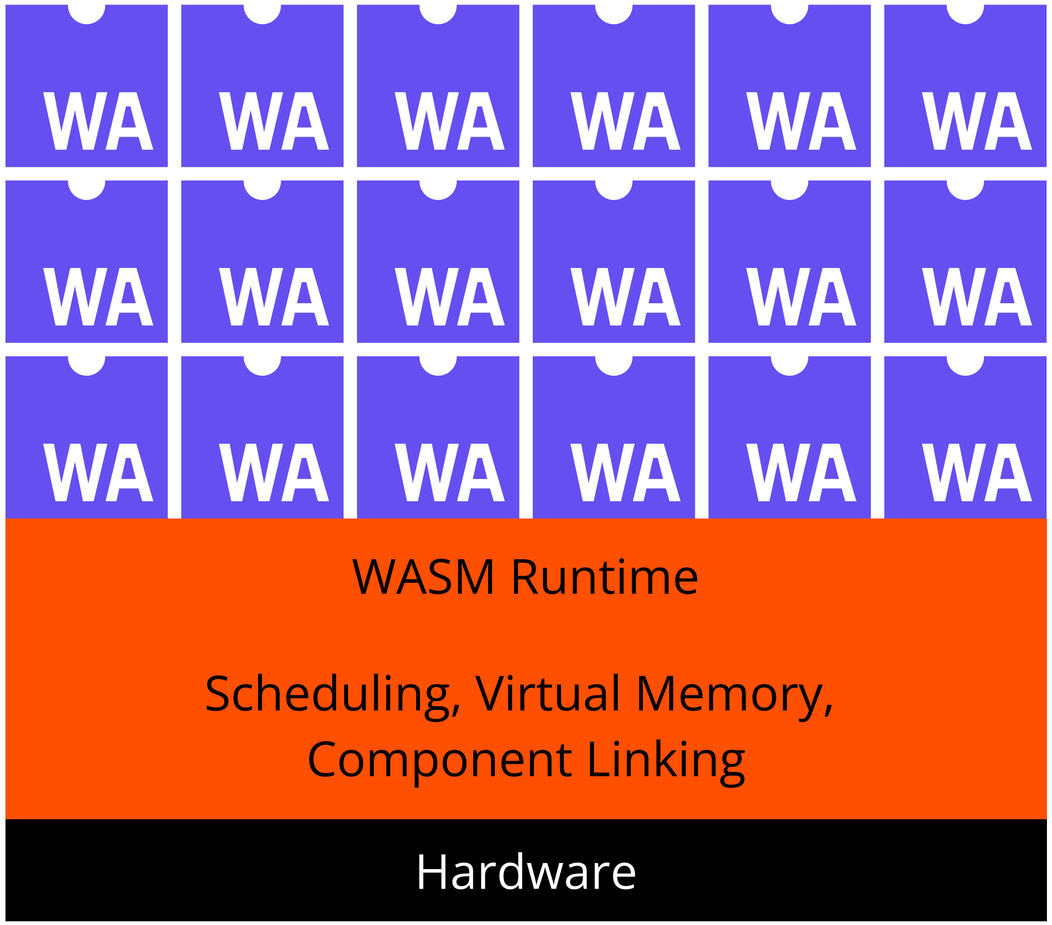
k23

## WebAssembly (WASM)

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

## Microkernel

- Security & Sandboxing
- Naturally follows from WASM



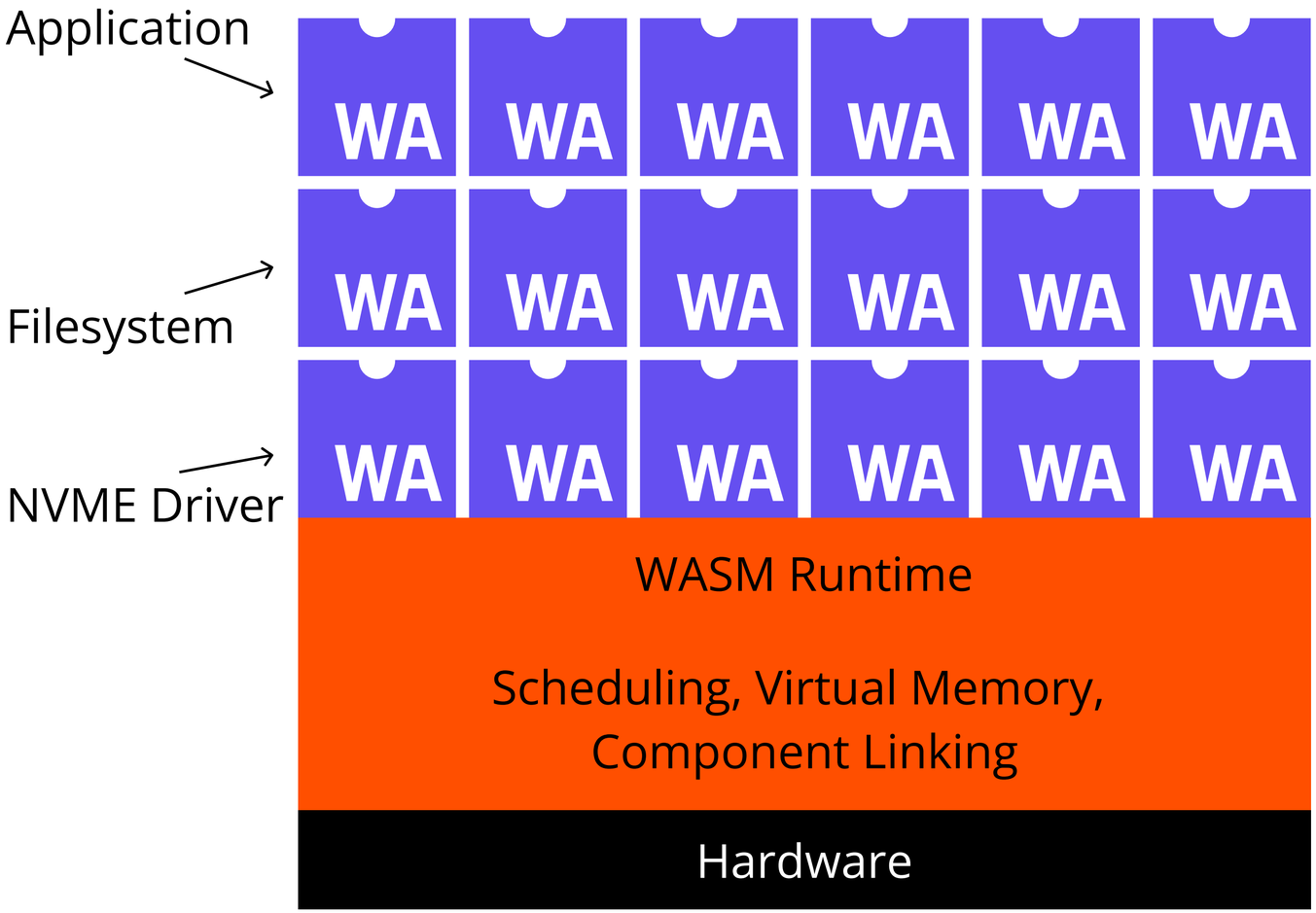
k23

## WebAssembly (WASM)

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

## Microkernel

- Security & Sandboxing
- Naturally follows from WASM



k23

## WebAssembly (WASM)

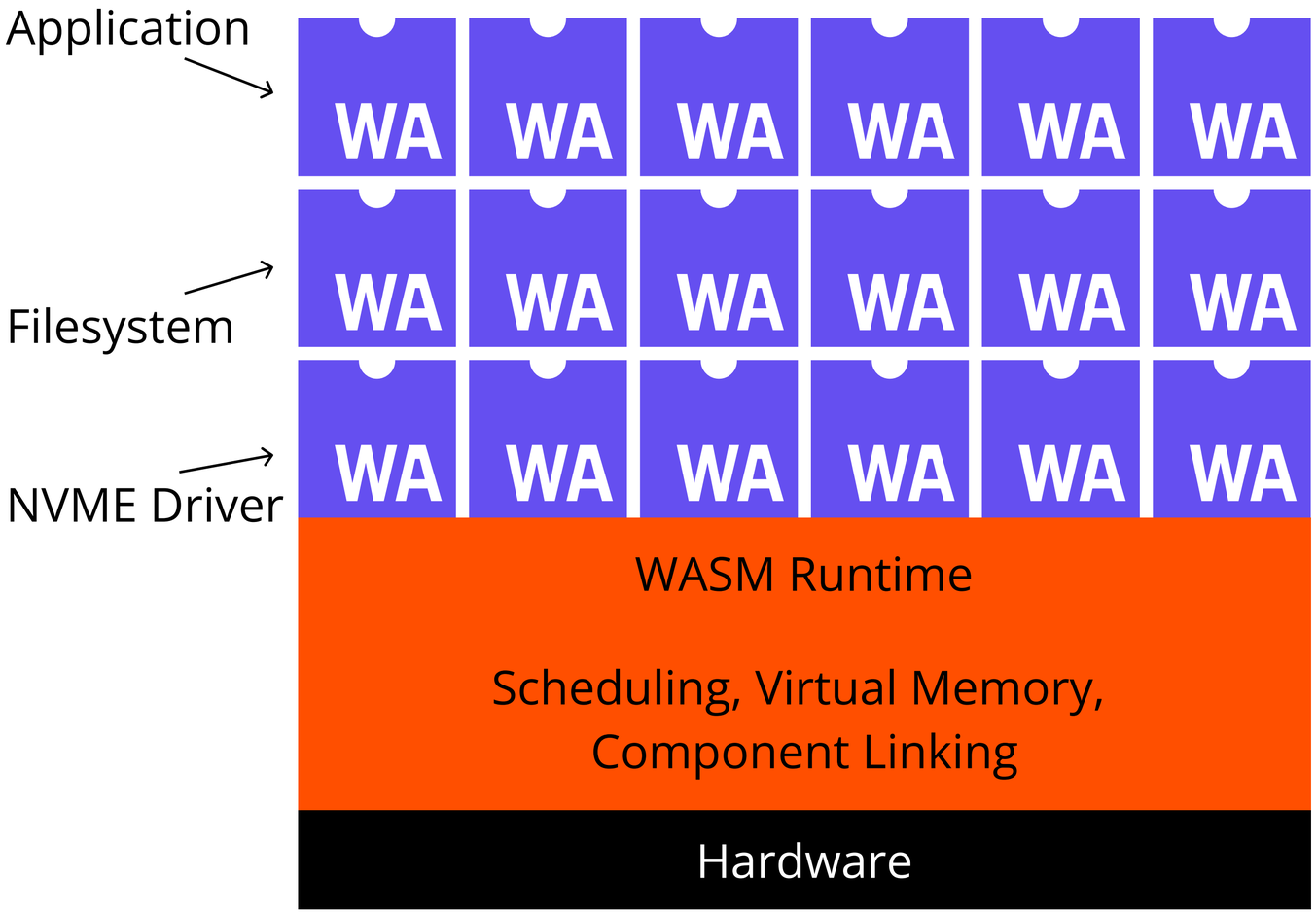
- Strong security, Permissions, Statically analyzable
- Just-In-Time Compilation

## Microkernel

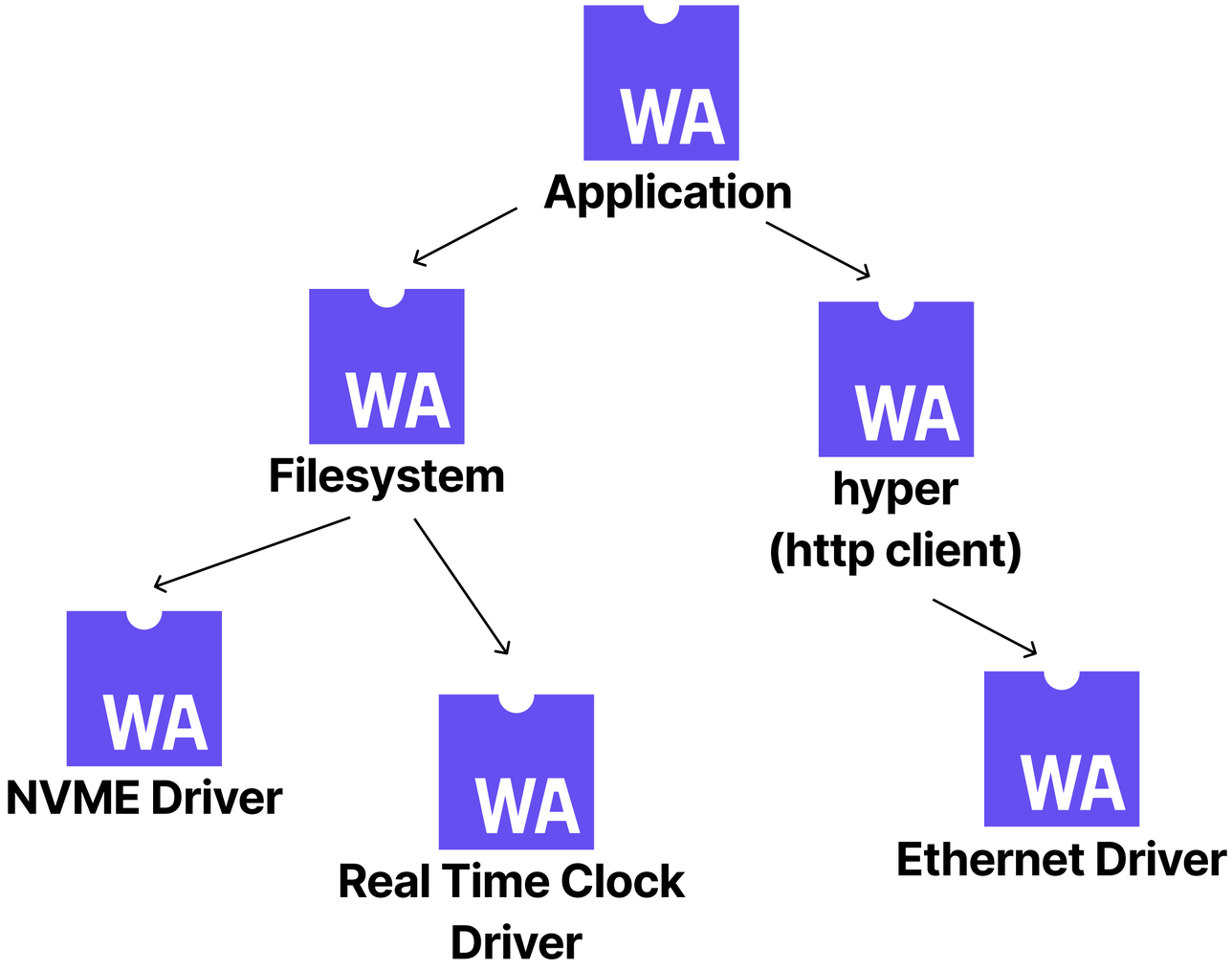
- Security & Sandboxing
- Naturally follows from WASM

## 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

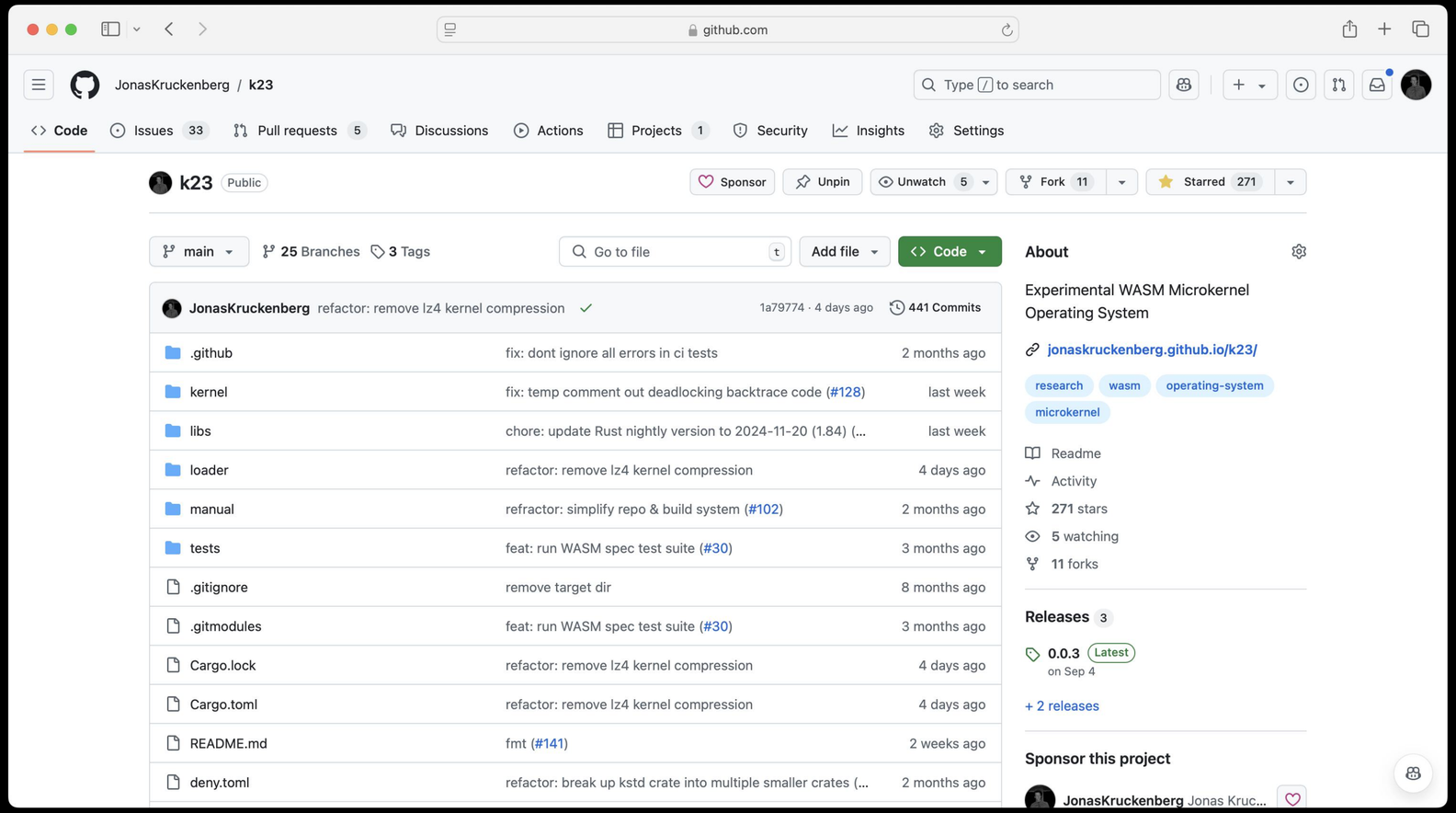
- Hardware access, language agnostic
- No hardware restrictions

## 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




https://github.com/JonasKruckenberg/k23

# The Plan


Today

# The Plan

Temporary Goal

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

Today

# The Plan

Phase I *WASM*

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

Today

Temporary Goal

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

# The Plan

Phase I *WASM*

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

Today

Phase II *Concurrency*

Cooperative Scheduler, Async Rust, SMP support and WASM threads proposal.
**Running multiple programs**

Temporary Goal

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

# The Plan

Today

Phase I *WASM*

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

Phase II *Concurrency*

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

# The Plan

Phase I *WASM*

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

Today

Phase II *Concurrency*

Cooperative Scheduler, Async Rust, SMP support and WASM threads proposal.
**Running multiple programs**

Phase II 1/2 *Kotlin*

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

Phase III Component Model

Implement WASM component model, Package Registry, and Dynamic Linking.
**Running 3rd party programs**

Temporary Goal

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

yay!

# The Good

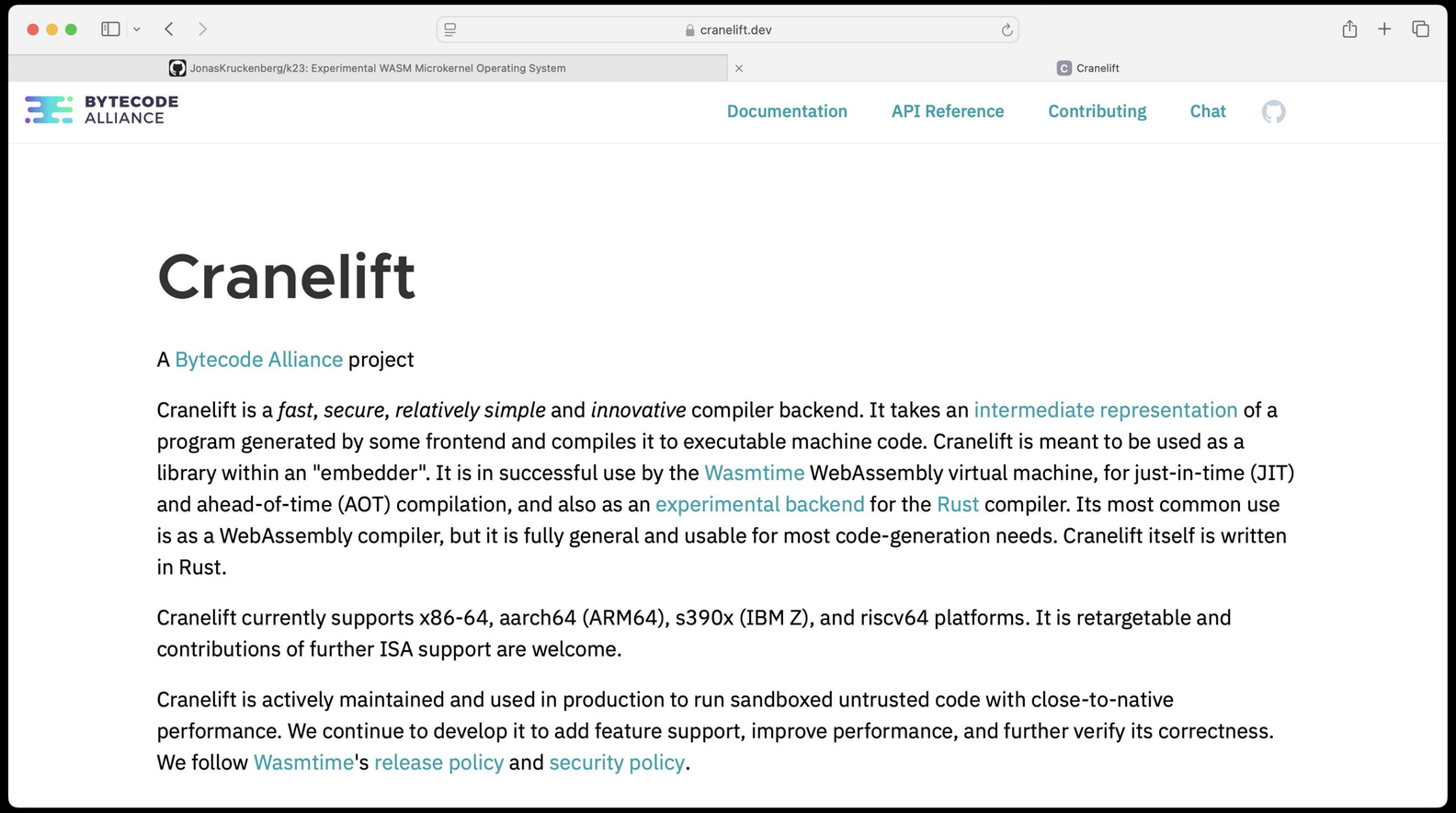
## Superpower Cargo

- 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`




## 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 Kruckenberg

@jonaskruckenberg.de
@unsafe@webtoo.ls
@jonasKruckie