



OWASP
BeNeLux Days
Conference
2023

Using WebAssembly to run,
extend, and secure your
application

Niels Tanis



VERACODE

Who am I?

- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
 - Enjoying Rust!
- Microsoft MVP - Developer Technologies

VERACODE



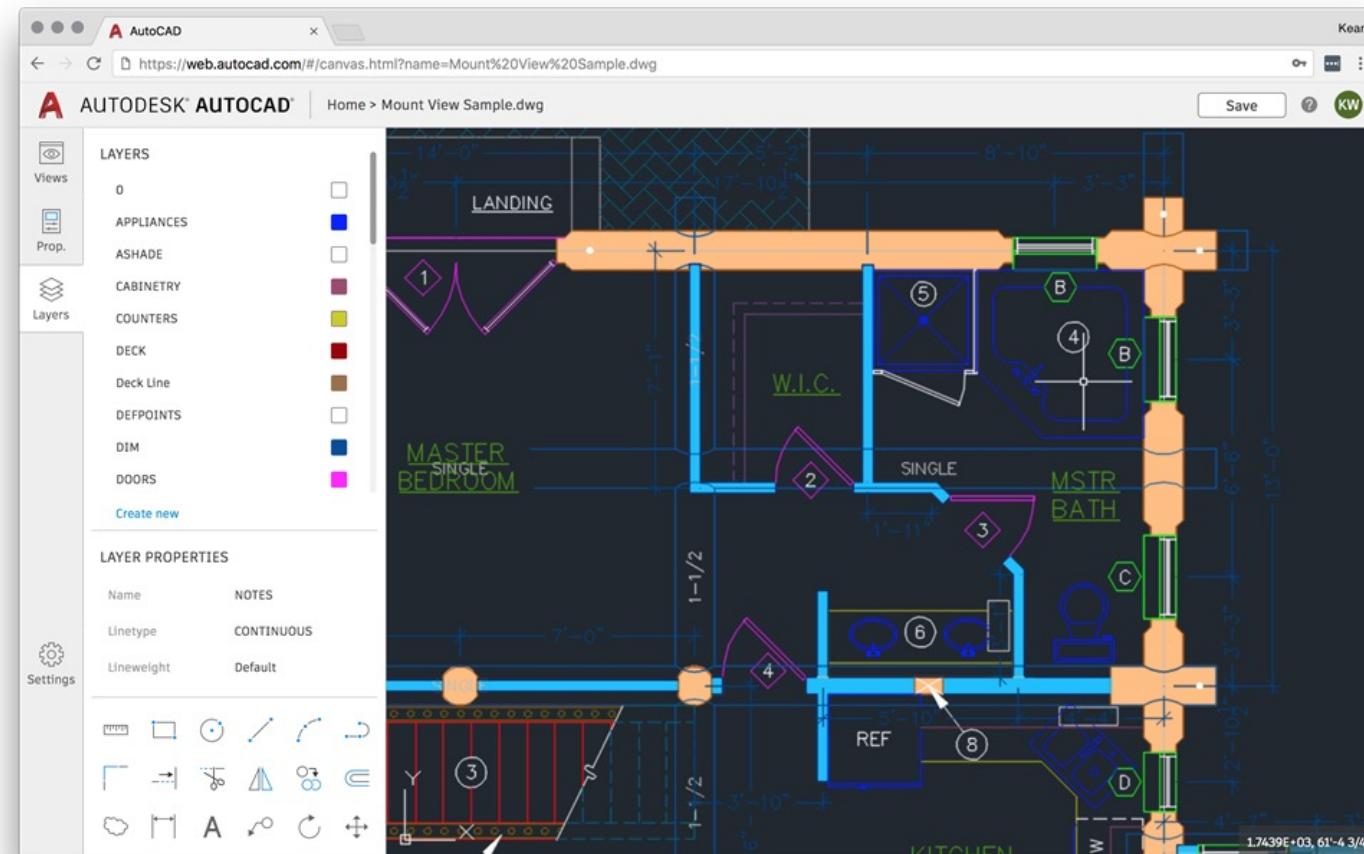
@nielstanis@infosec.exchange

WebAssembly

The screenshot shows a web browser window displaying the [WebAssembly High-Level Goals](https://webassembly.org/docs/high-level-goals/) page. The page features a purple header with the WebAssembly logo and navigation links for Overview, Getting Started, Specs, Future features, Community, and FAQ. A green banner at the top states "WebAssembly 1.0 has shipped in 4 major browser engines." followed by icons for Firefox, Chrome, Opera, and Edge, with a "Learn more" link. The main content area is titled "WebAssembly High-Level Goals" and lists three main goals:

1. Define a [portable](#), size- and load-time-efficient [binary format](#) to serve as a compilation target which can be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including [mobile](#) and [IoT](#).
2. Specify and implement incrementally:
 - a [Minimum Viable Product \(MVP\)](#) for the standard with roughly the same functionality as [asm.js](#), primarily aimed at [C/C++](#);
 - [additional features](#), initially focused on key features like [threads](#), [zero cost exceptions](#), and [SIMD](#), followed by additional features prioritized by feedback and experience, including support for languages other than C/C++.
3. Design to execute within and integrate well with the [existing Web platform](#):
 - maintain the versionless, [feature-tested](#) and backwards-compatible evolution story of the Web;

WebAssembly - AutoCAD



 @nielstanis@infosec.exchange

WebAssembly - SDK's

A screenshot of a Medium article page. The title is "Introducing the Disney+ Application Development Kit (ADK)". It was written by Mike Hanley on September 8, 2021, and has a 10-minute read time. The article discusses the Disney+ Application Development Kit (ADK) and its benefits. The interface includes a "Follow" button for the author, social sharing icons (Twitter, Facebook, LinkedIn), and a "Get started" button.

A screenshot of a Medium article page. The title is "How Prime Video updates its app for more than 8,000 device types". It was written by Alexandru Ene on January 27, 2022. The article discusses the switch to WebAssembly and its impact on stability and speed. The interface includes a "Subscribe" button, a search bar, and a "Share" button.

Agenda

- Introduction
- WebAssembly 101
- Using WebAssembly to ...
 - ... run your application
 - ... extend your application
 - .. secure your application
- Conclusion
- Q&A

WebAssembly Design

- **Be fast, efficient, and portable**

- Executed in near-native speed across different platforms

- **Be readable and debuggable**

- In low-level bytecode but also human readable

- **Keep secure**

- Run on sandboxed execution environment

- **Don't break the web**

- Ensure backwards compatibility



WEBASSEMBLY

WebAssembly

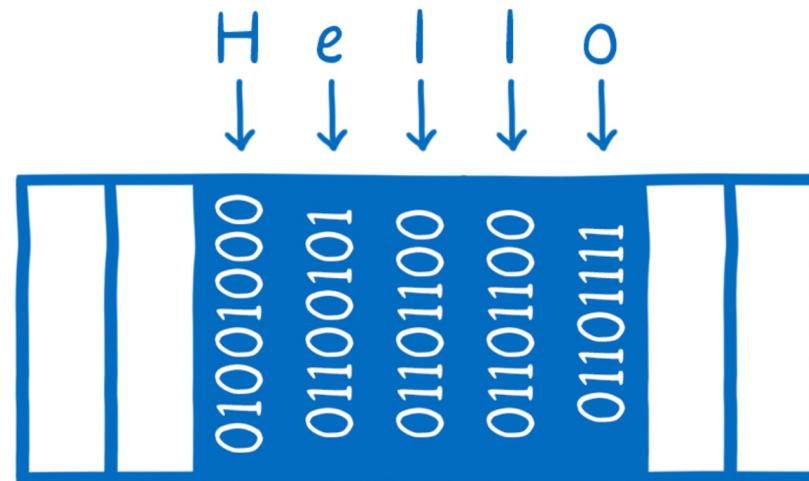
- Binary instruction format for stack-based virtual machine similar to .NET CLR running MSIL or JVM running bytecode
- Designed as a portable compilation target
- The security model of WebAssembly:
 - Protect users from buggy or malicious modules
 - Provide developers with useful primitives and mitigations for developing safe applications



WEBASSEMBLY

WebAssembly Memory

- Isolated per WASM module
- A contiguous, mutable array of uninterpreted bytes

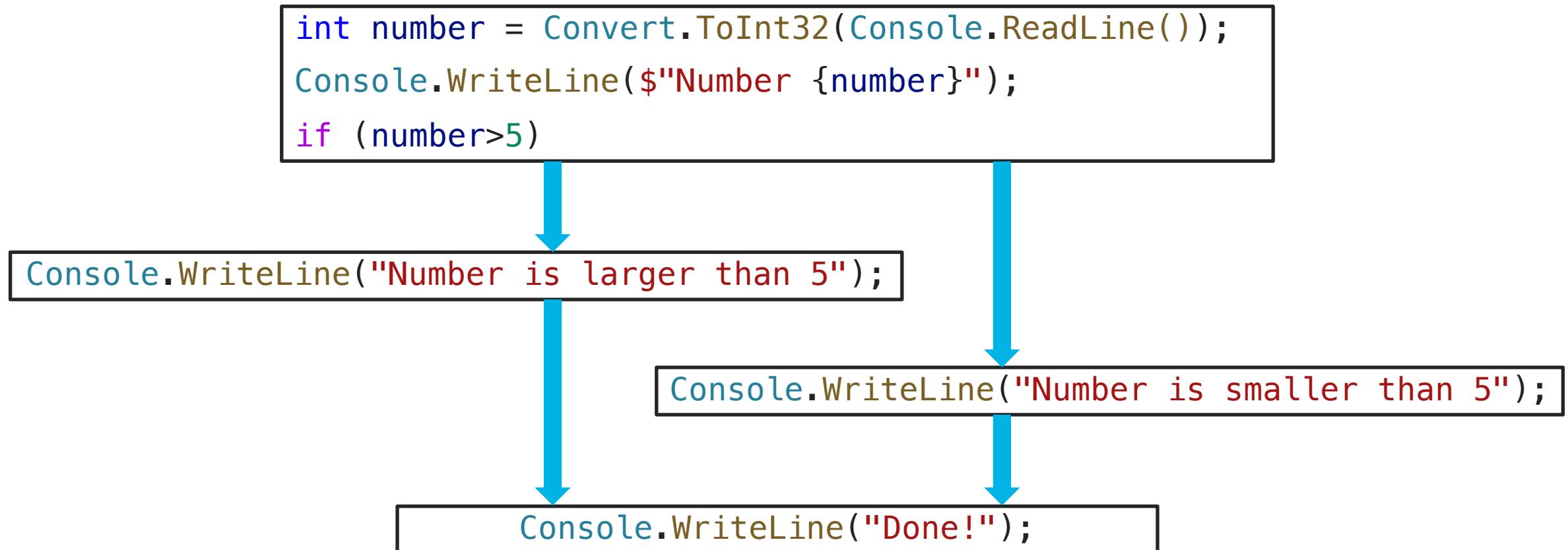


WebAssembly Control-Flow Integrity

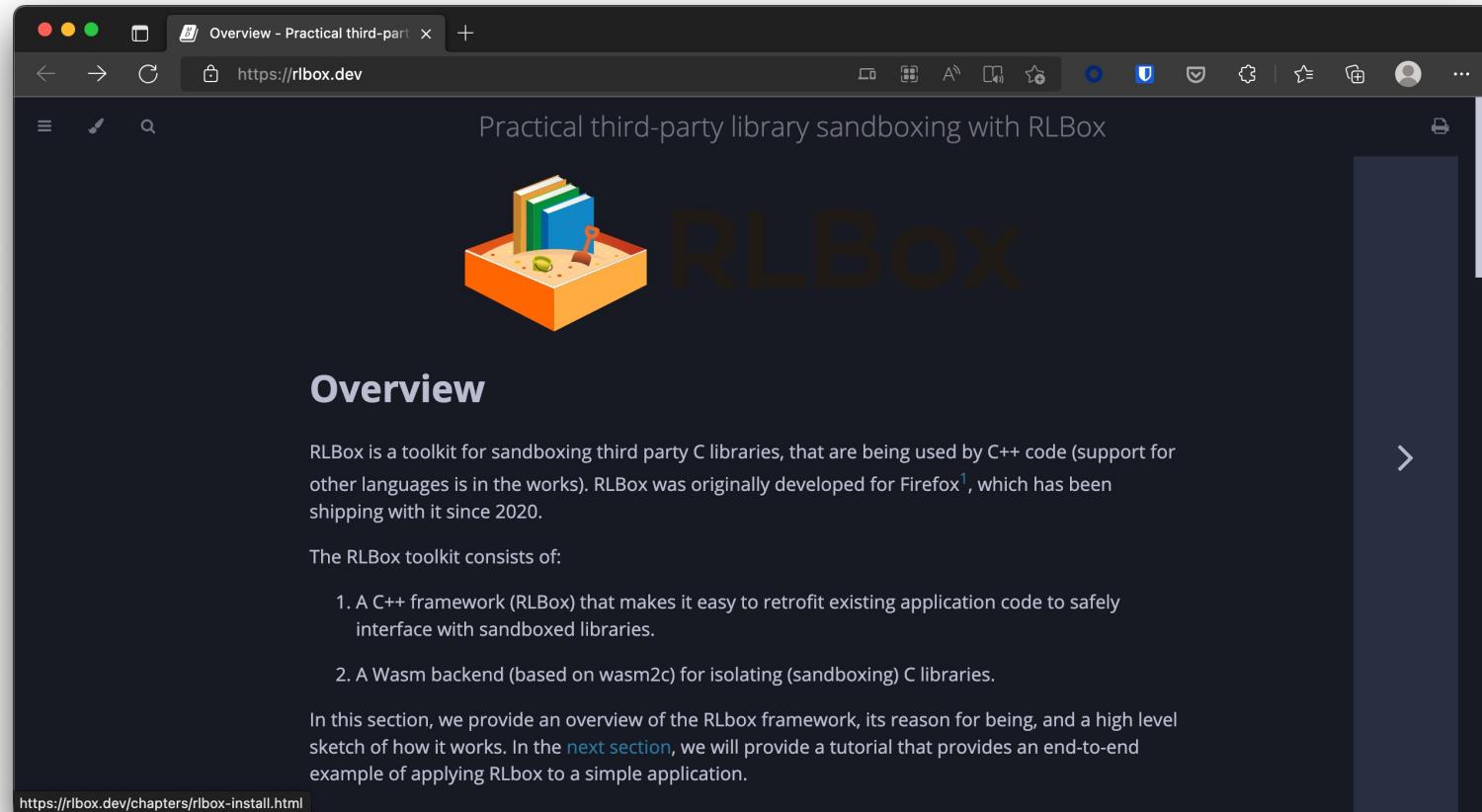
```
int number = Convert.ToInt32(Console.ReadLine());
Console.WriteLine($"Number {number}");
if (number>5)
{
    Console.WriteLine("Number is larger than 5");
}
else
{
    Console.WriteLine("Number is smaller than 5");
}
Console.WriteLine("Done!");
```



WebAssembly Control-Flow Integrity

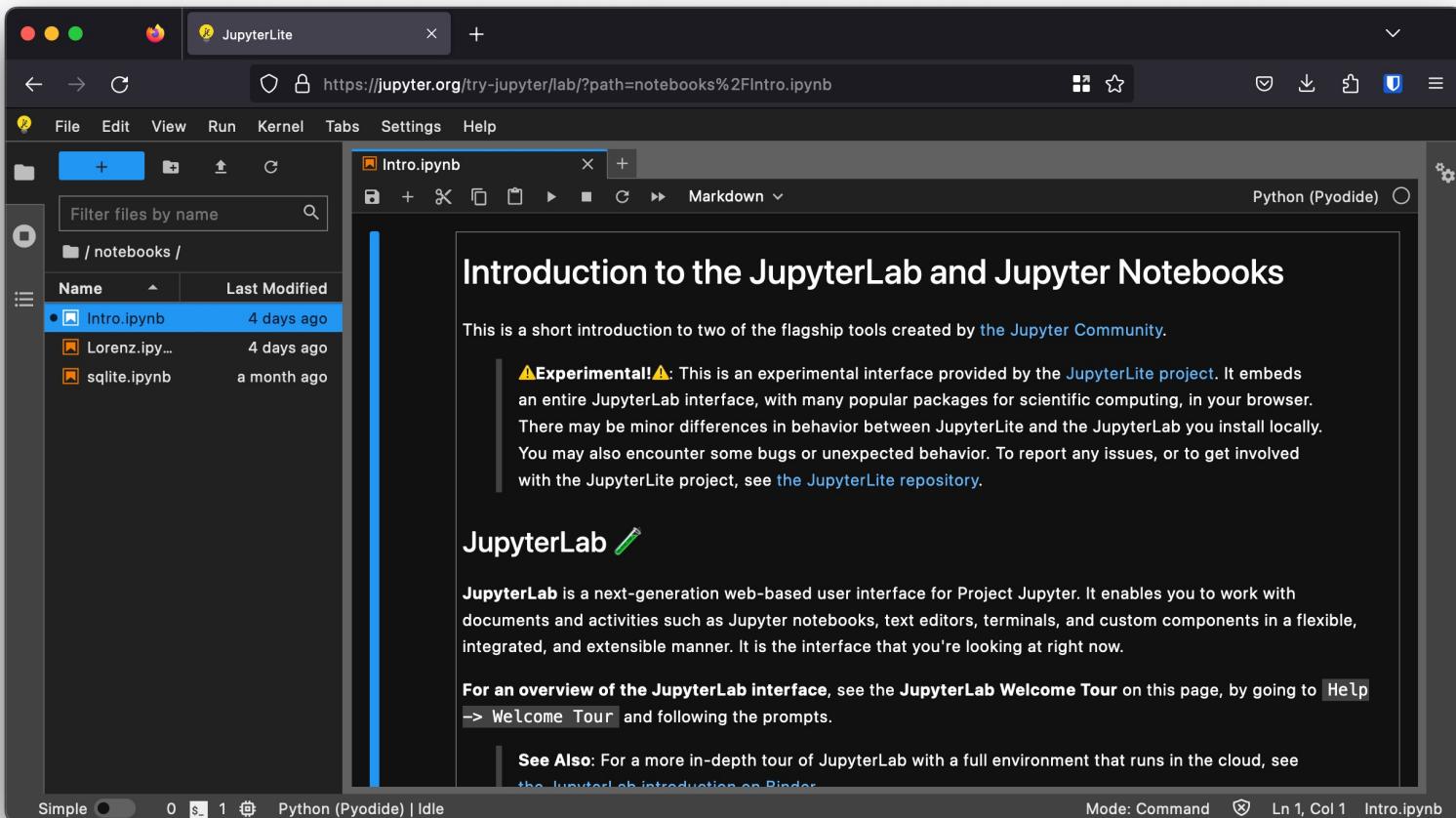


FireFox RLBox

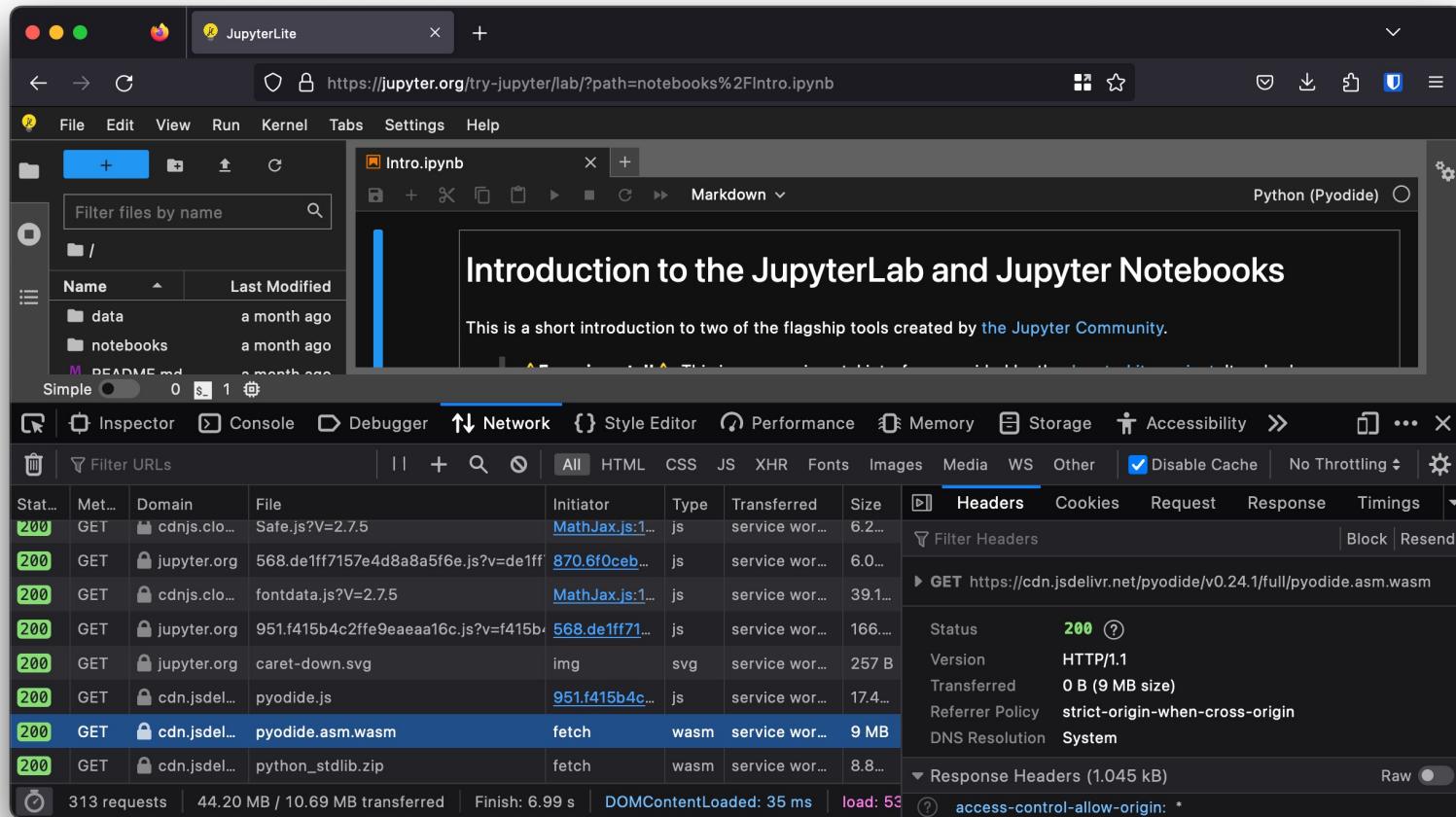


The screenshot shows a Firefox browser window with a dark theme. The title bar says "Overview - Practical third-part" and the address bar shows "https://rlbox.dev". The main content area has a dark background with a central logo featuring three books in a sandcastle-like setup with a shovel. To the right of the logo is the word "RLBox" in large, bold, white letters. Below the logo, the word "Overview" is centered in a large, bold, white font. The text below "Overview" describes RLBox as a toolkit for sandboxing third-party C libraries. It mentions support for C++ and other languages, its development for Firefox since 2020, and its components: a C++ framework and a Wasm backend. A link at the bottom left points to "https://rlbox.dev/chapters/rlbox-install.html".

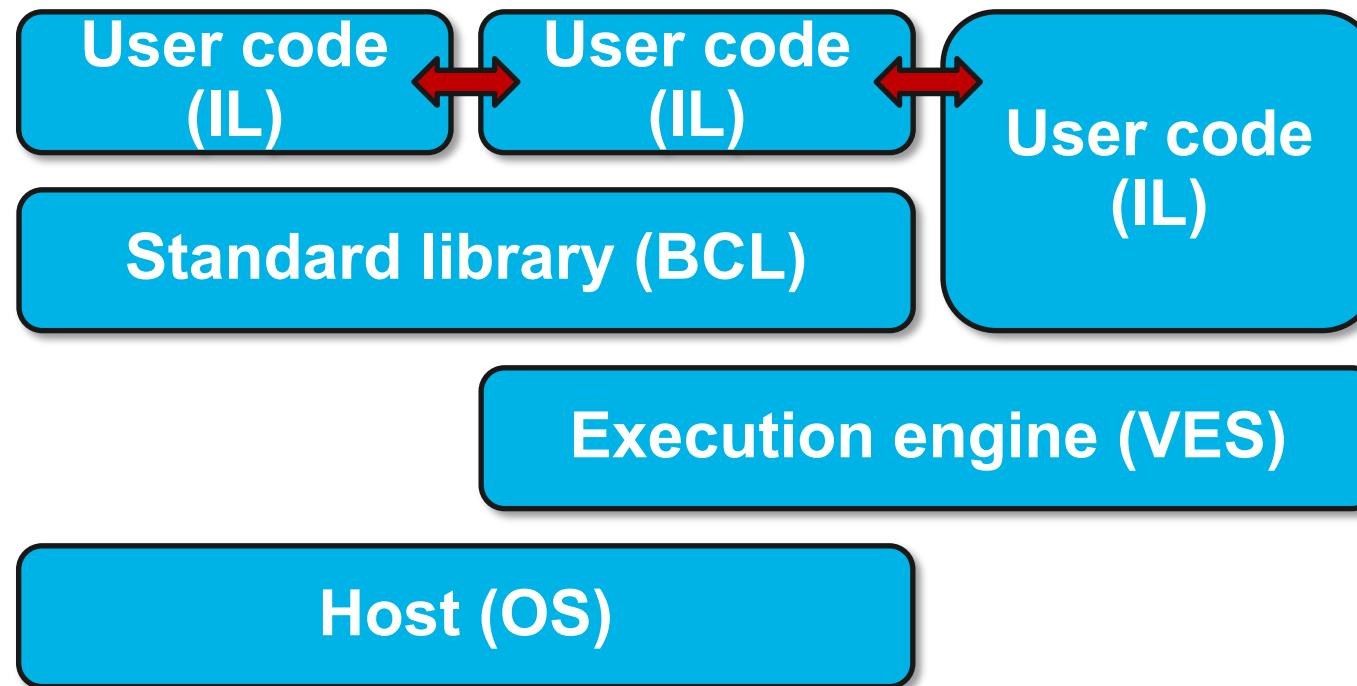
Python on WASM



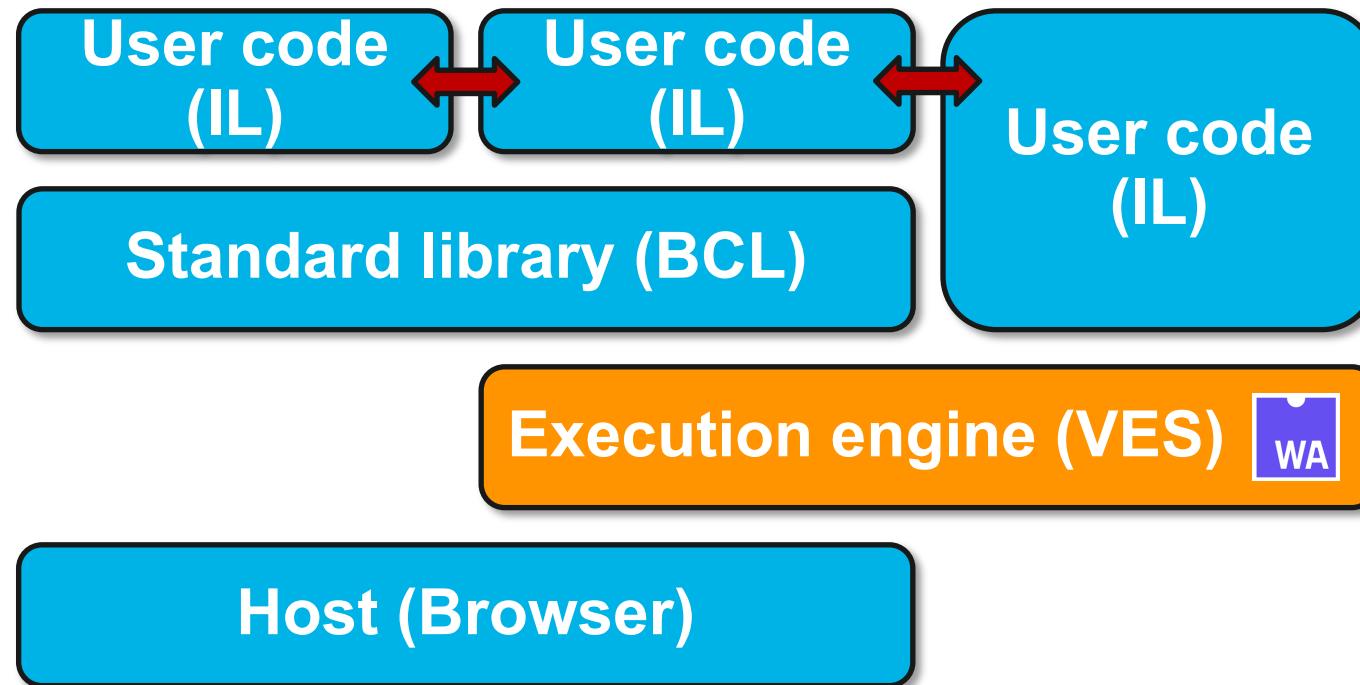
Python on WASM



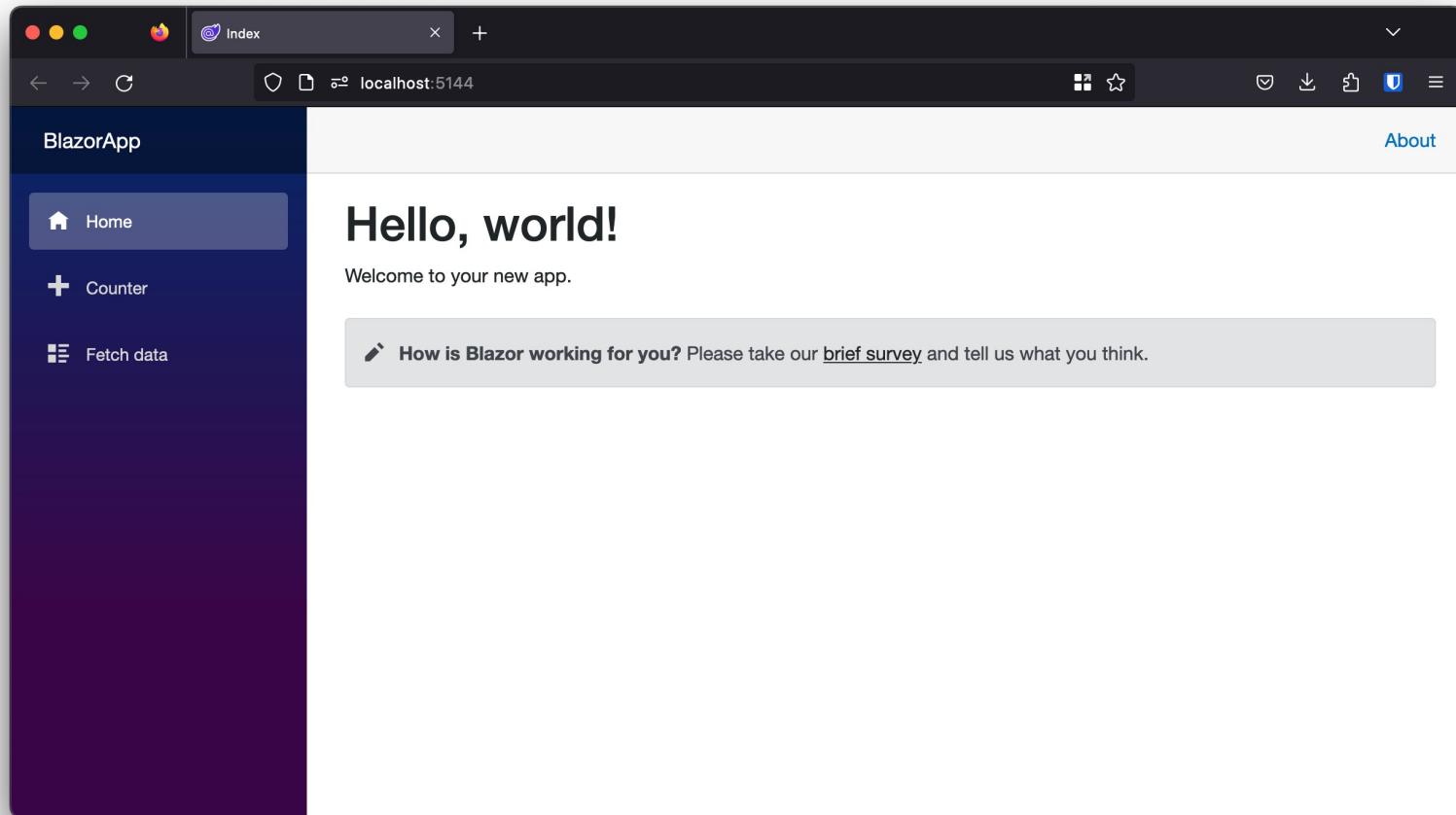
Running .NET on WebAssembly



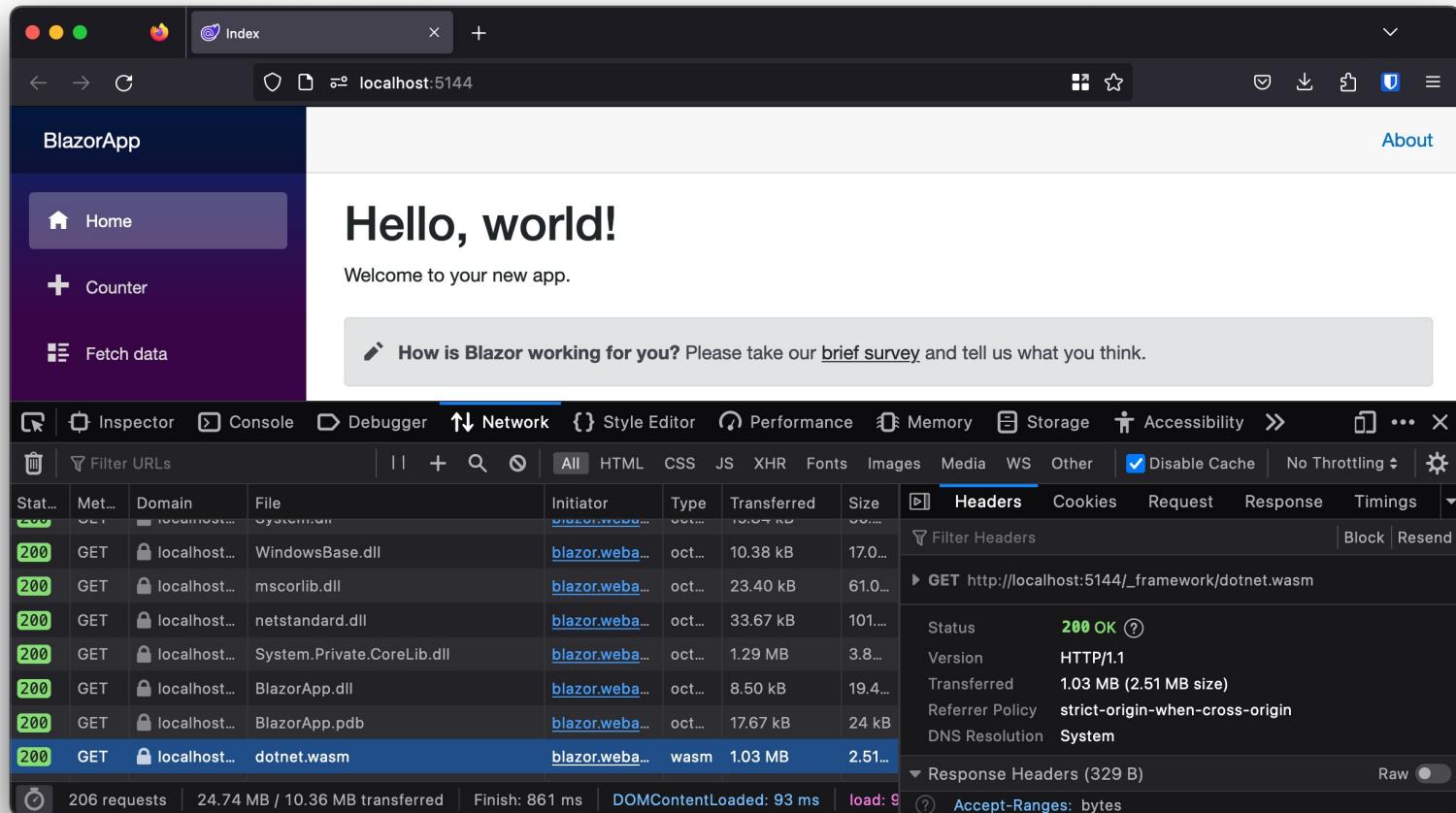
Running .NET on WebAssembly



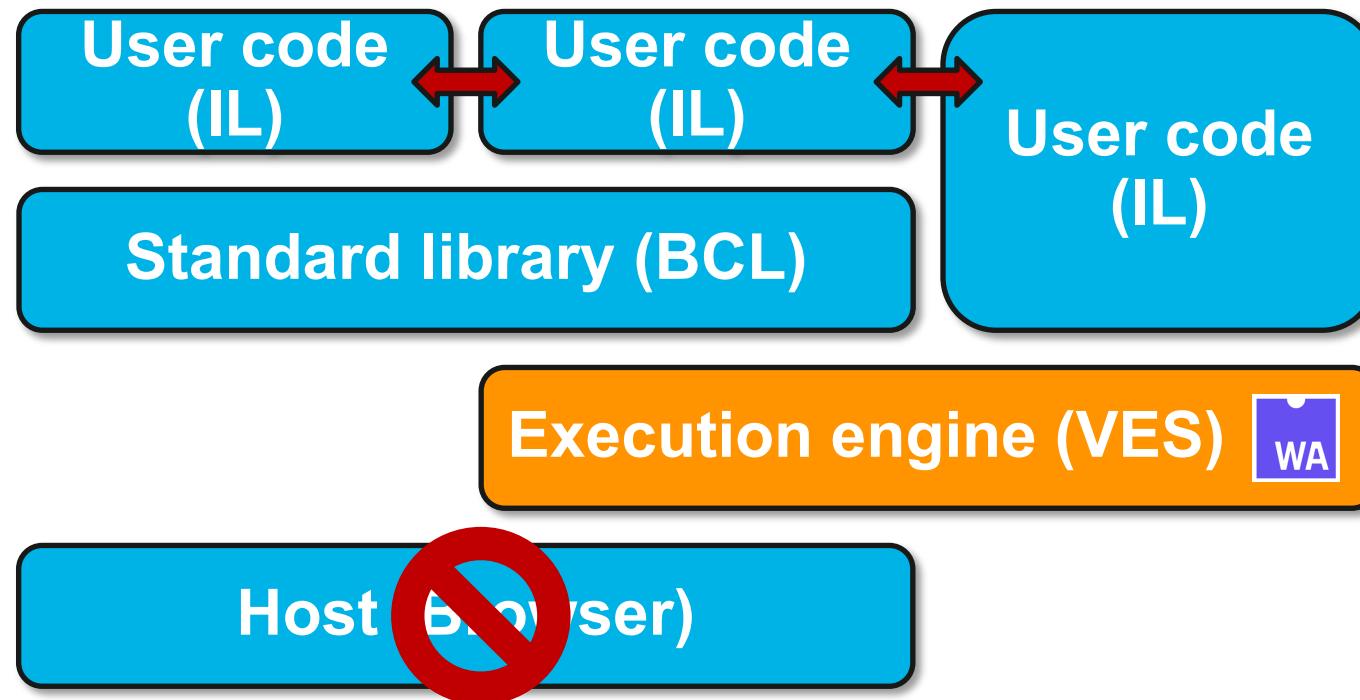
Blazor WebAssembly



Blazor WebAssembly



Running .NET on WebAssembly



WebAssembly System Interface WASI

- Introduced in March 2019 by Bytecode Alliance
- WasmTime implementation as reference
- POSIX inspired, engine-independent, non-Web system-oriented API for WebAssembly

WebAssembly System Interface WASI

- Strong sandbox with Capability Based Security
- Right now, supports e.g. FileSystem actions
- Future support for sockets and other system resources.
- Anyone familiar with .NET Standard? ☺

Docker vs WASM & WASI

A screenshot of a Twitter web client window. The URL in the address bar is <https://twitter.com/s...>. The main content is a tweet from **Solomon Hykes** (@solomonstre) titled "Collectie". The tweet text is:

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

Below the tweet is a reply from **Lin Clark** (@linclark) dated 27 mrt. 2019:

WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

hacks.mozilla.org/2019/03/standa...

[Deze collectie weergeven](#)

At the bottom of the tweet card, it says "9:39 p.m. · 27 mrt. 2019 · Twitter Web Client".

Docker vs WASM & WASI

Solomon Hykes op Twitter: <https://twitter.com/solomonstre>

Solomon Hykes @solomonstre

“So will wasm replace Docker?” No, but imagine a future where Docker runs linux containers, windows containers and wasm containers side by side. Over time wasm might become the most popular container type. Docker will love them all equally, and run it all :)

Tweet vertalen

Solomon Hykes @solomonstre · 27 mrt. 2019

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task! twitter.com/linclark/status/1105404411000000000

Deze collectie weergeven

4:50 a.m. · 28 mrt. 2019 · Twitter Web App

56 Retweets 5 Geciteerde Tweets 165 Vind-ik-leuks

Docker & WASM

The screenshot shows a web browser window with the title "Introducing the Docker+Wasm Technical Preview". The main content features the Docker+Wasm logo and the heading "Introducing the Docker+Wasm Technical Preview". Below the heading is a bio for Michael Irwin, a photo of him, and the date "Oct 24 2022". A note at the bottom states: "The Technical Preview of Docker+Wasm is now available! Wasm has been producing a lot of buzz recently, and this feature will make it easier for you to quickly build applications targeting Wasm runtimes."

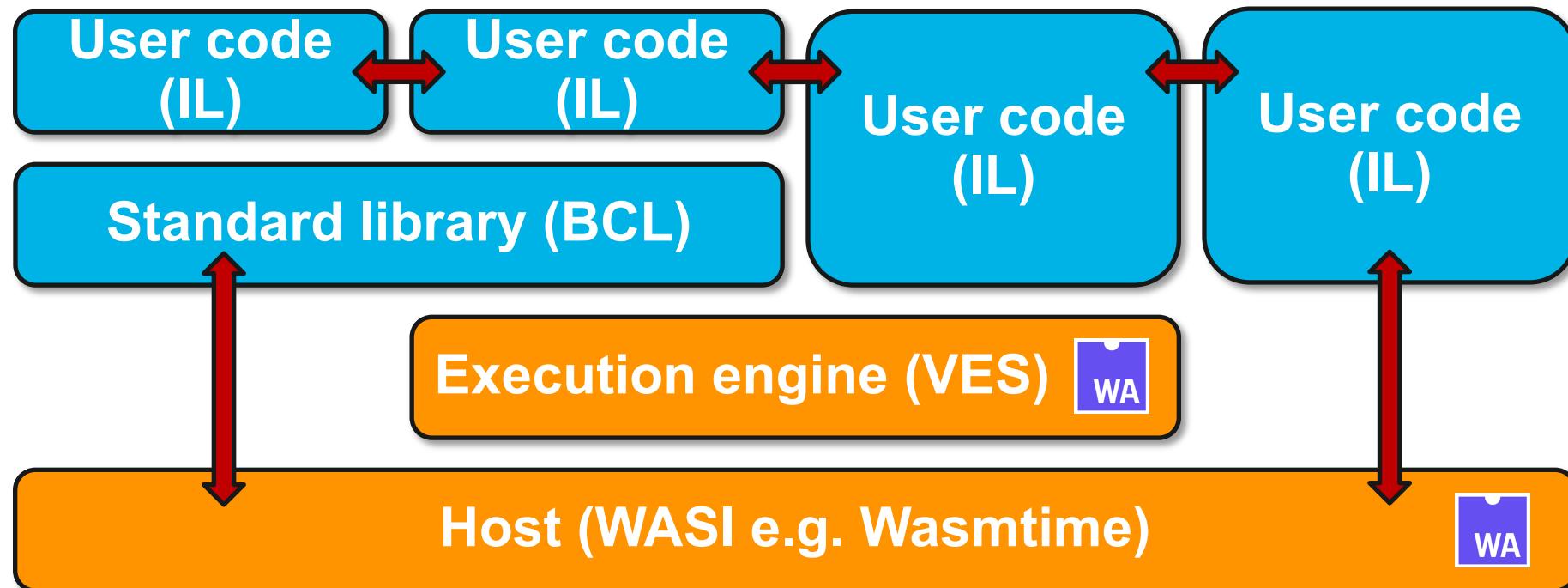
The screenshot shows a web browser window with the same title as the first screenshot. It displays a diagram illustrating the Docker+Wasm architecture. At the top is the "Docker Engine", which interacts with a "containerd" component. Below "containerd" are three separate container instances. Each instance contains a "containerd-shim" layer, which runs "runc", which in turn manages a "Container process". The third container instance also includes a "containerd-wasm-shim" layer, which runs "wasmedge", managing a "Wasm Module". Arrows indicate the flow of control from the Docker Engine down through the layers to the final application processes.

Let's look at an example!

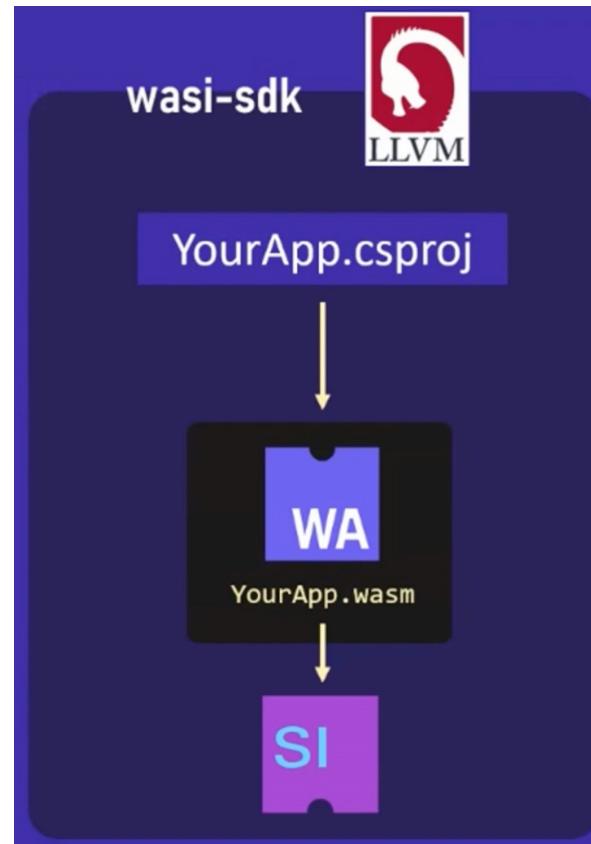
After installing the preview, we can run the following command to start an example Wasm application:

```
docker run -dp 8080:8080 --name=wasm-example --runtime=io.containerd.wasmedge.v1 --platform=wasi/wasm32 michaelirwin244/wasm-example
```

WebAssembly System Interface WASI



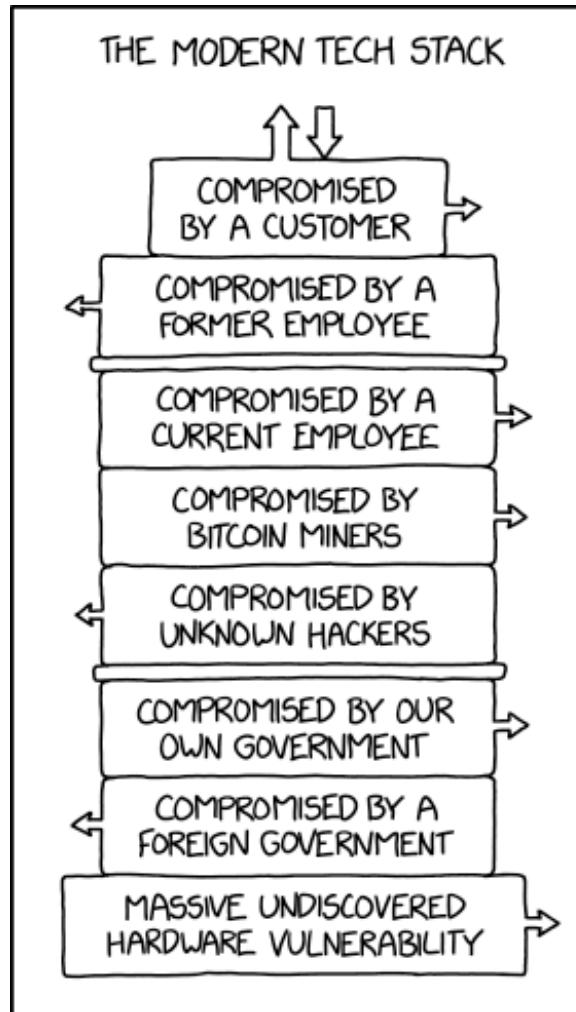
Experimental WASI SDK for .NET



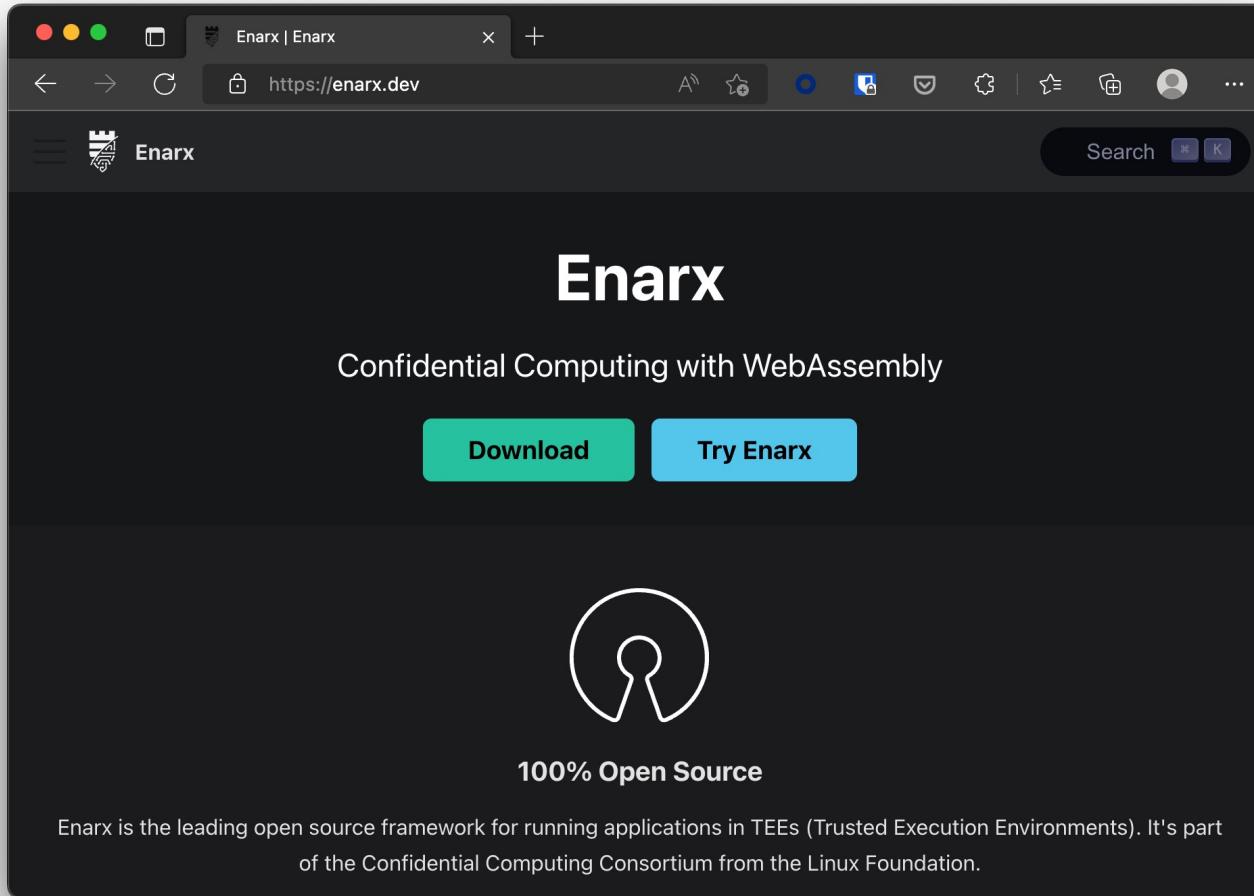
Extending .NET with WASM

- WasmTime.NET NuGet package
- Can run WASM inside of any .NET application
- Extend with Rust based WASM module
- Limit capabilities
- Demo time!

Trusted Computing - XKCD 2166

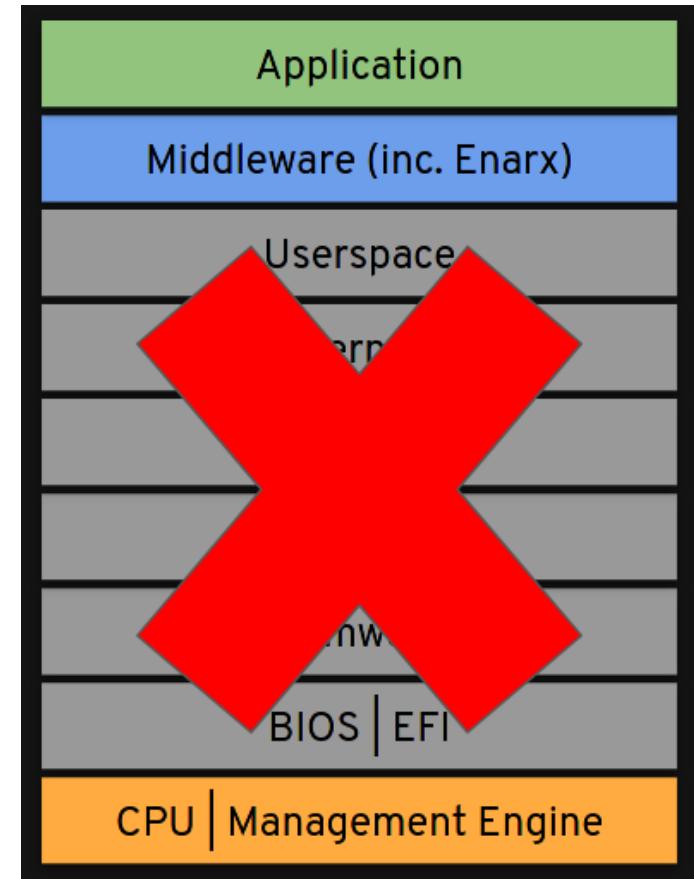


Enarx



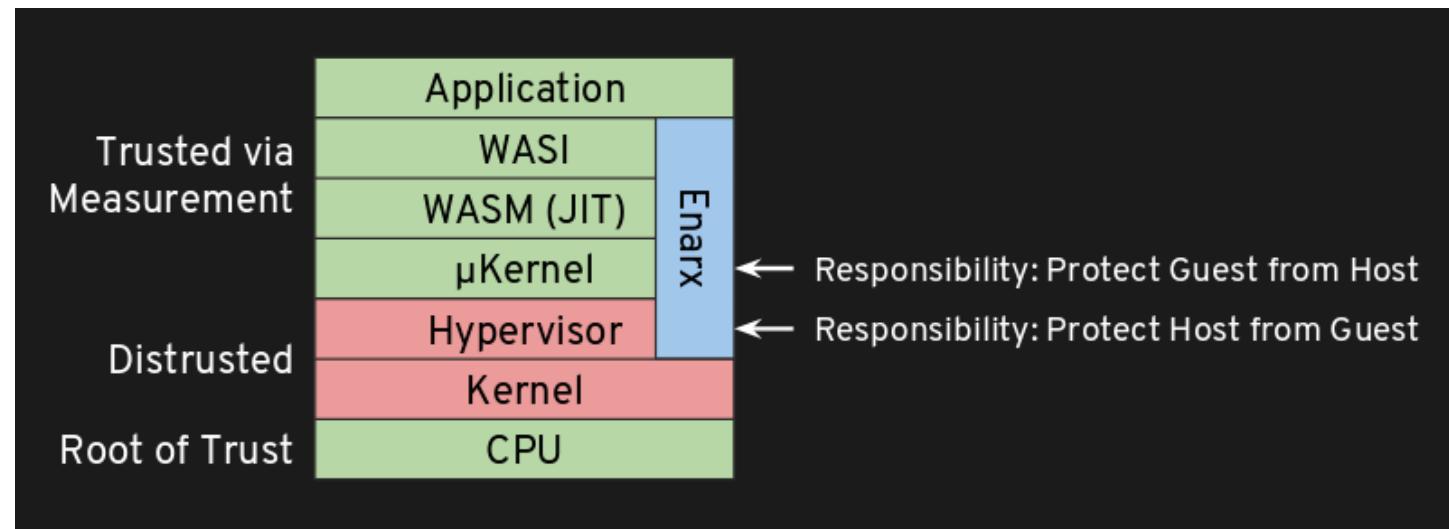
Enarx Threat Model

- Don't trust the host
- Don't trust the host owner
- Don't trust the host operator
- Hardware cryptographically verified
- Software audited and cryptographically verified

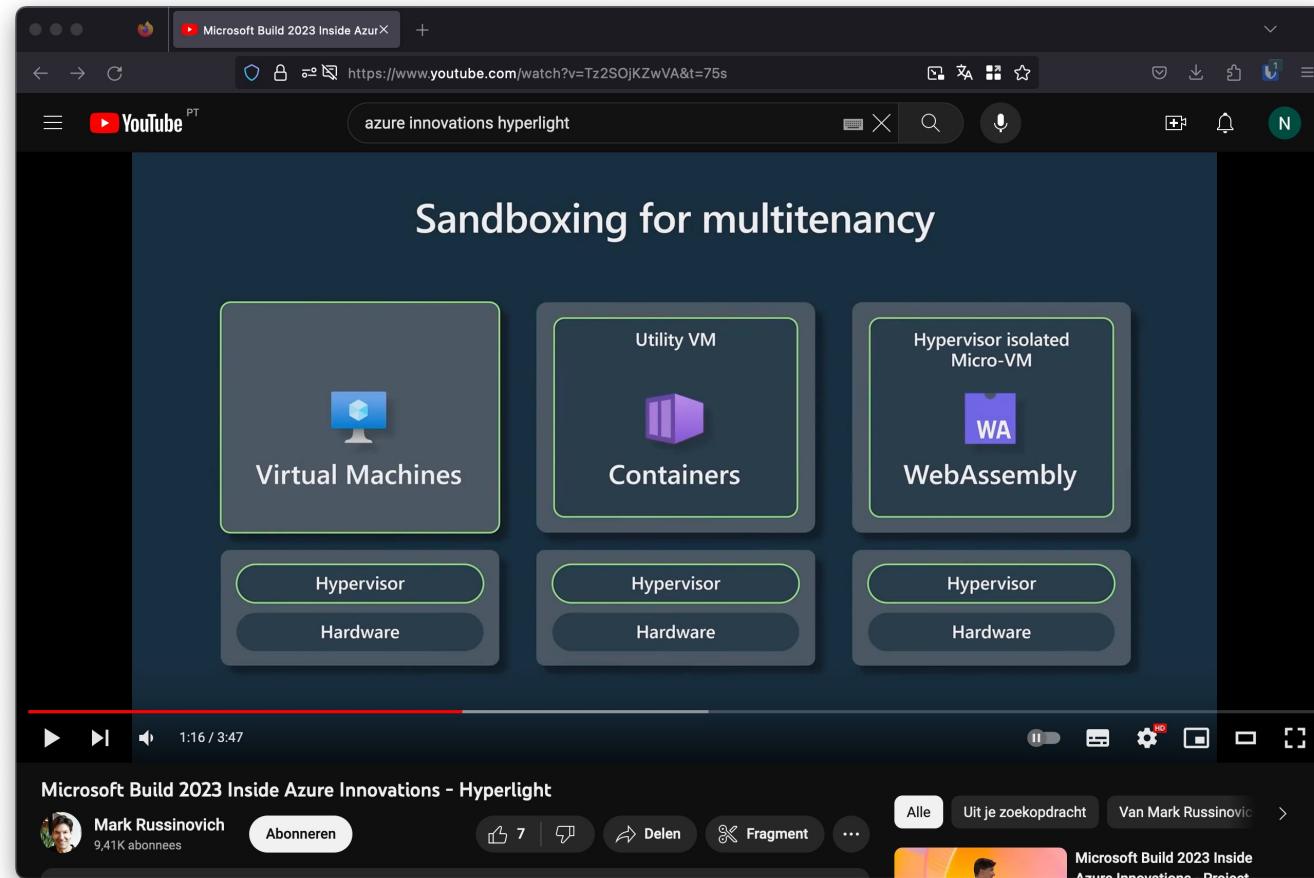


Enarx

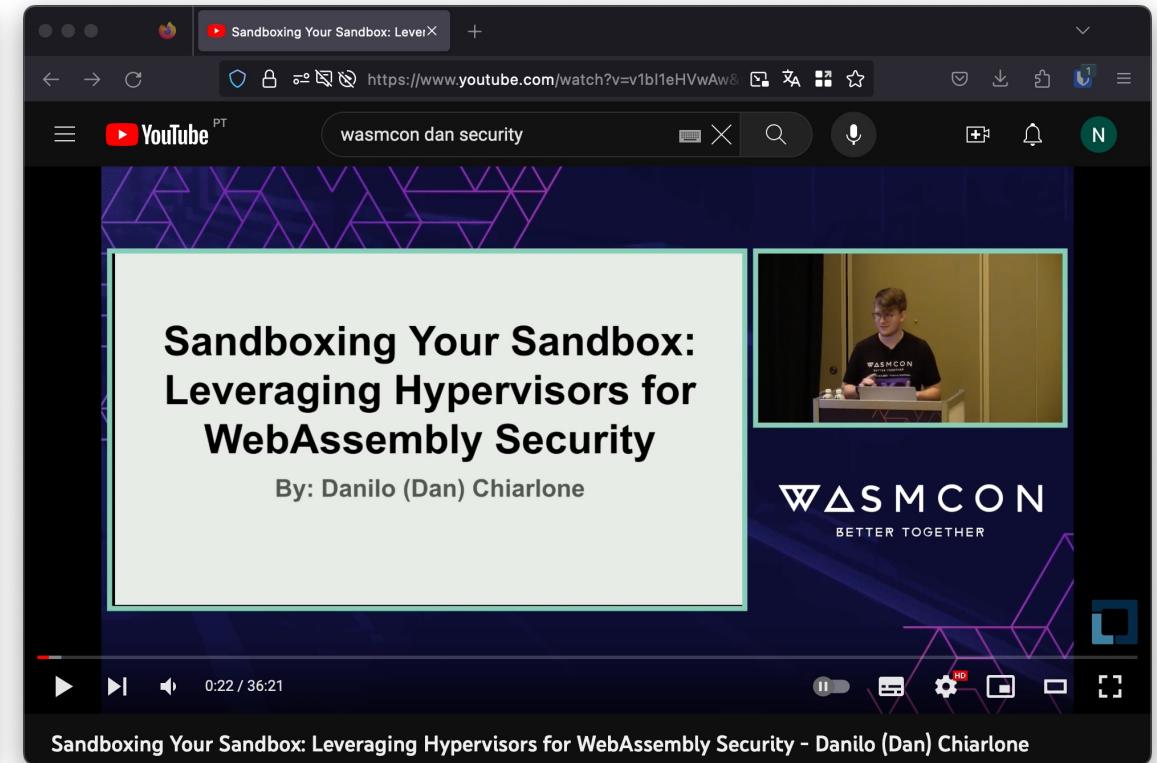
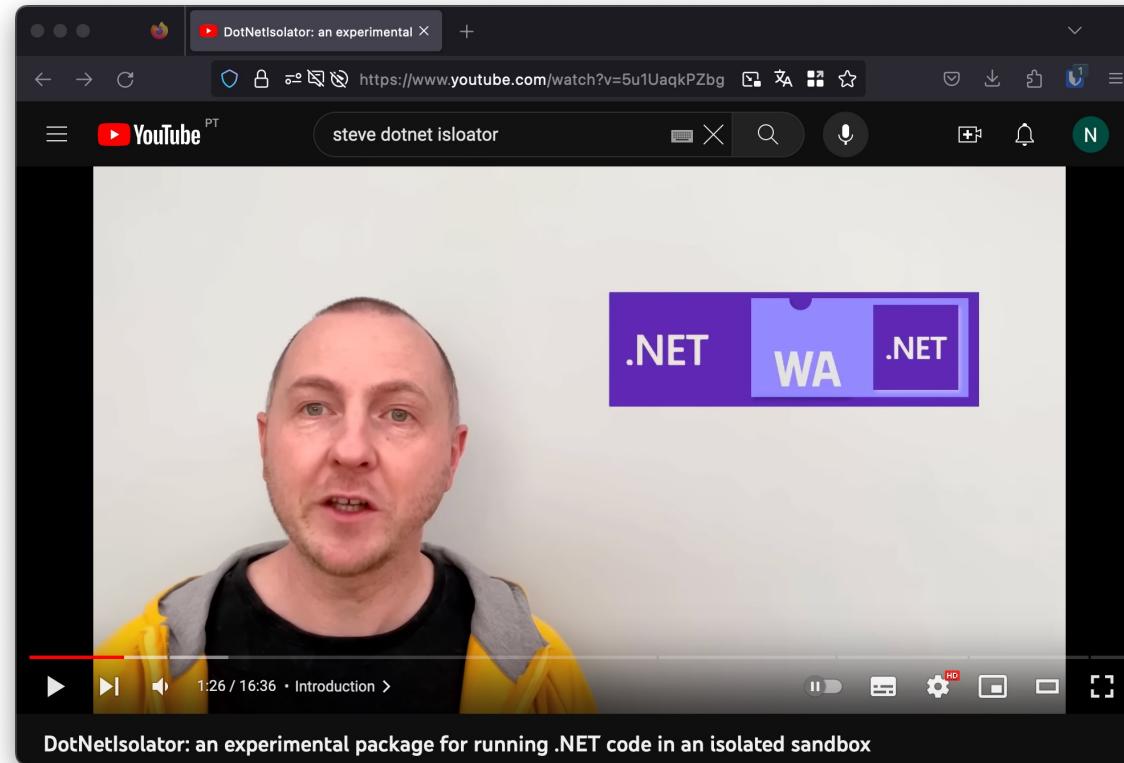
- Leverages Trusted Execution Environment (TEE) direct on processor
 - AMD's SEV, Intel's SGX and IBM's PEF
- Attestation of hardware and Enarx runtime



Project Hyperlight

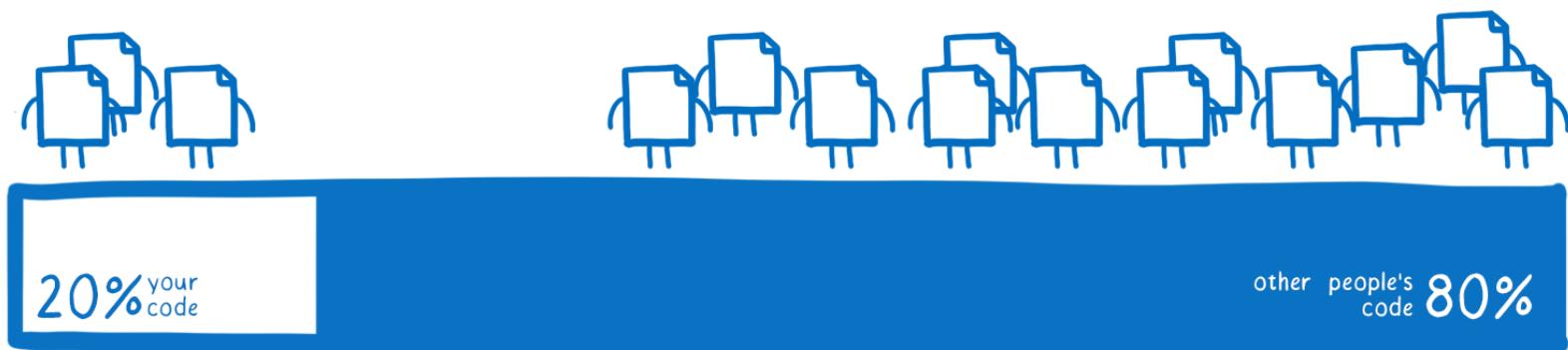


DotNetIsolator & Project Hyperlight

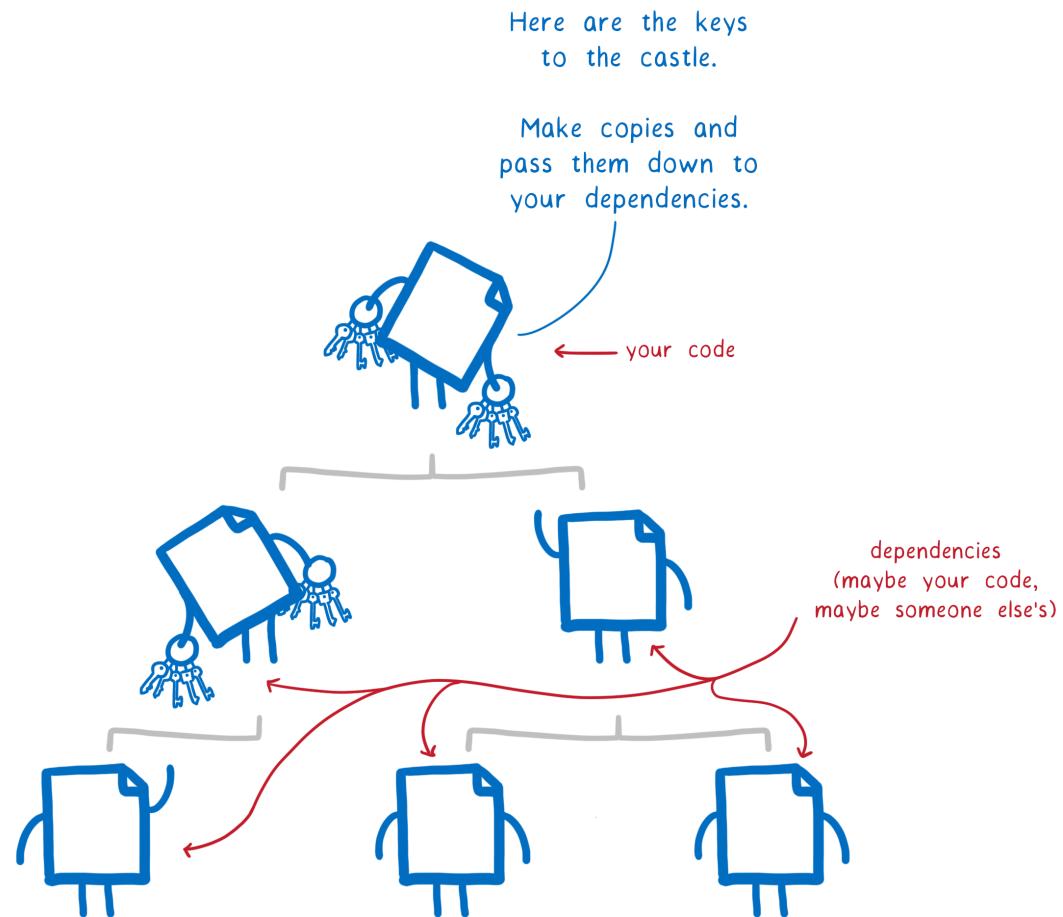


WASM - What's next?

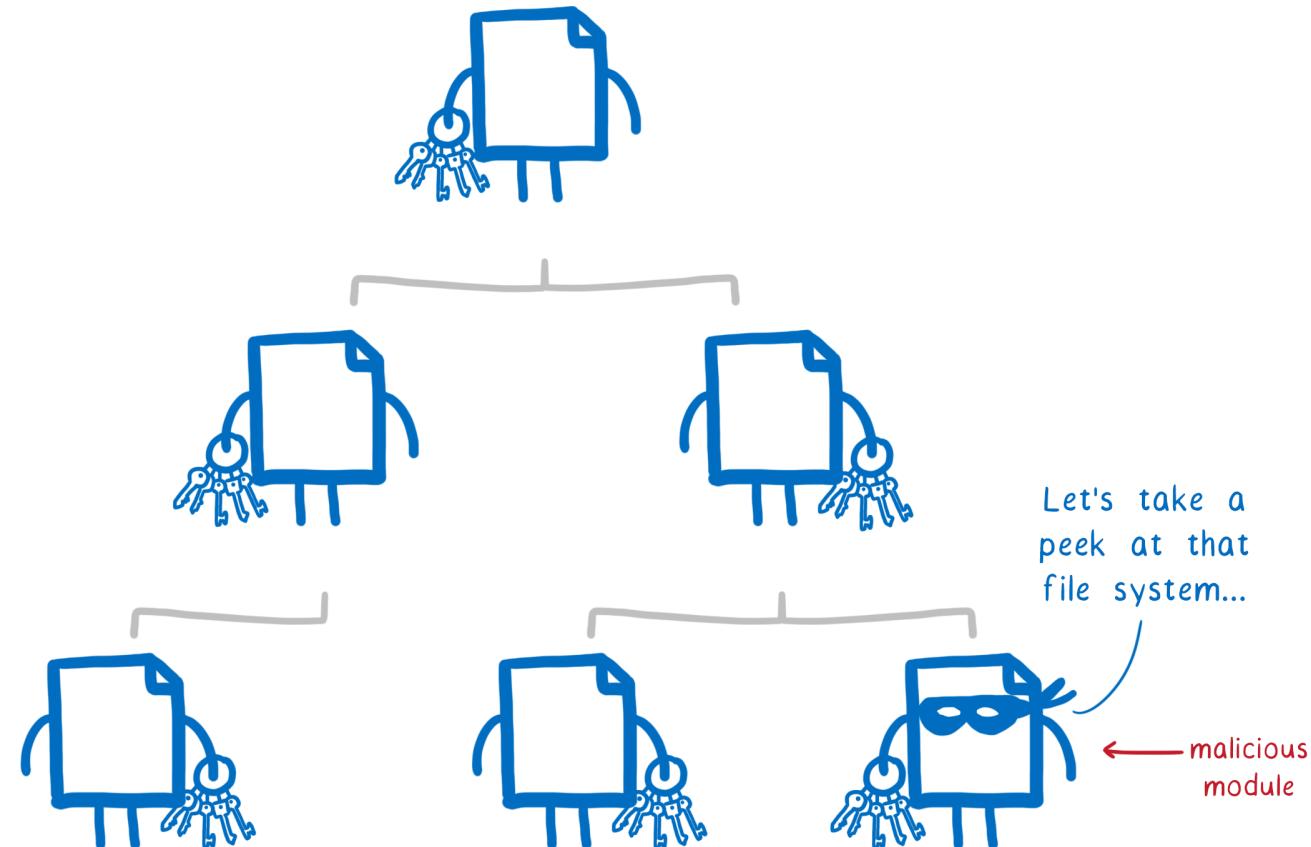
composition of an
average code base



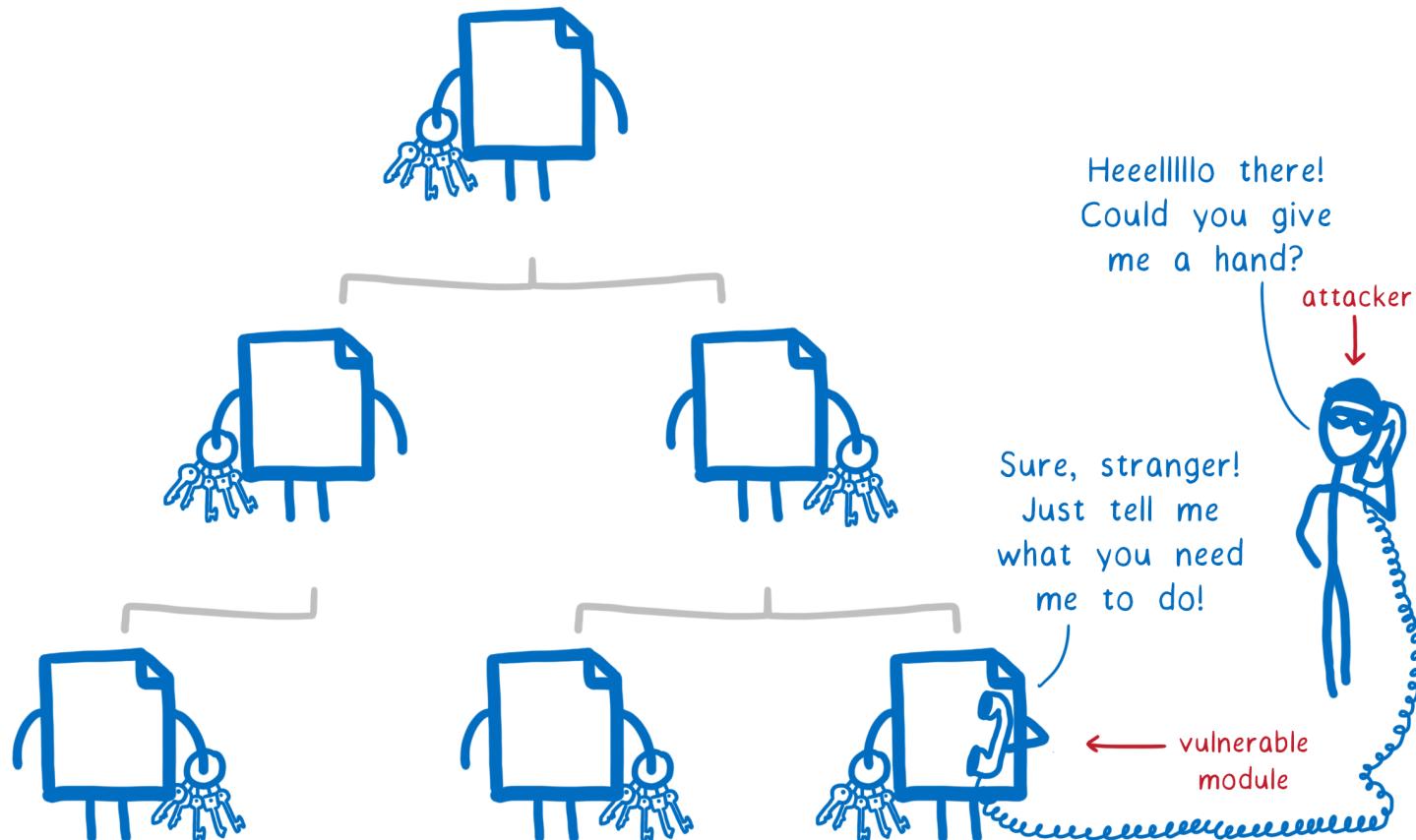
Dependencies



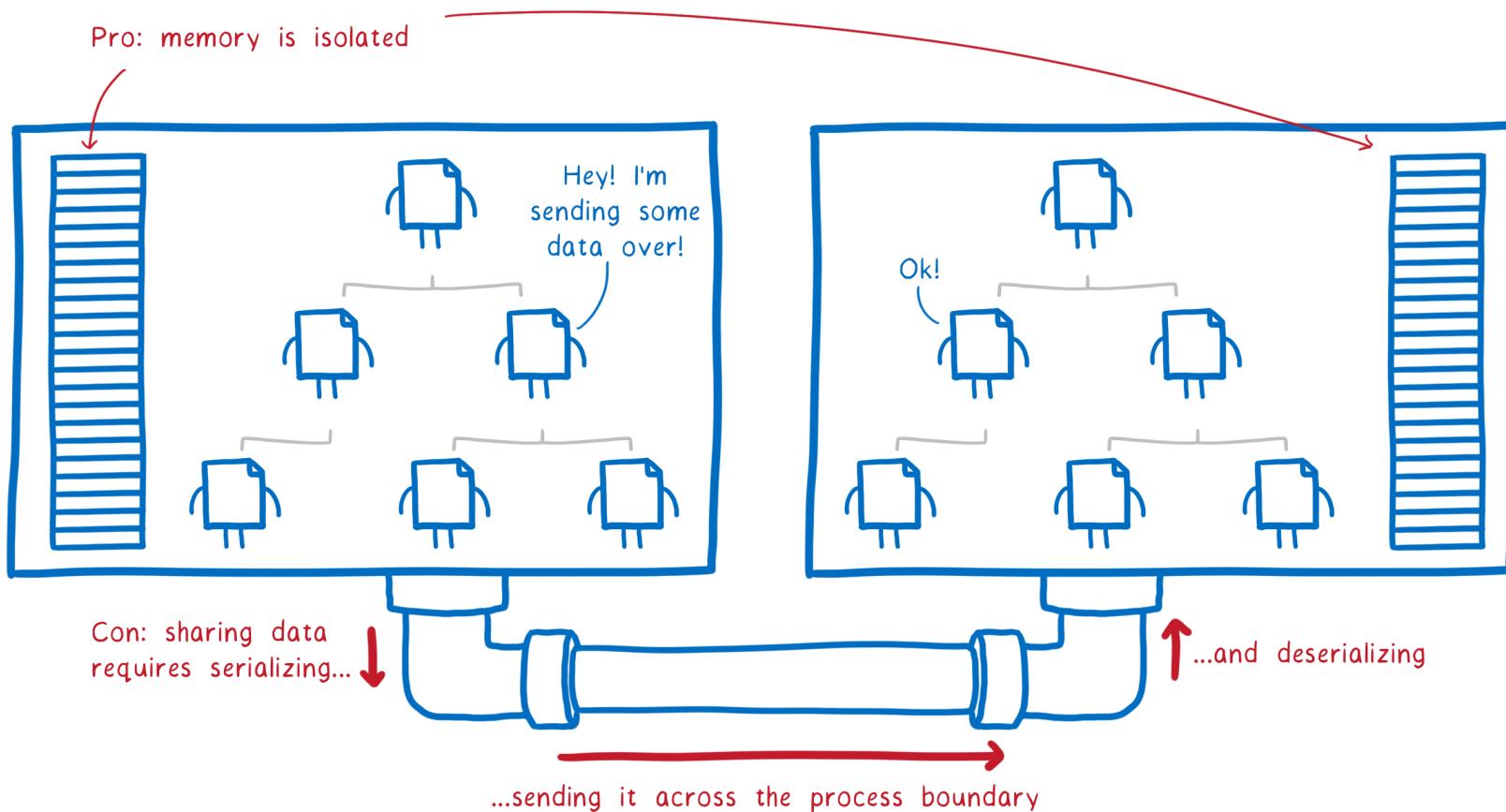
Malicious module



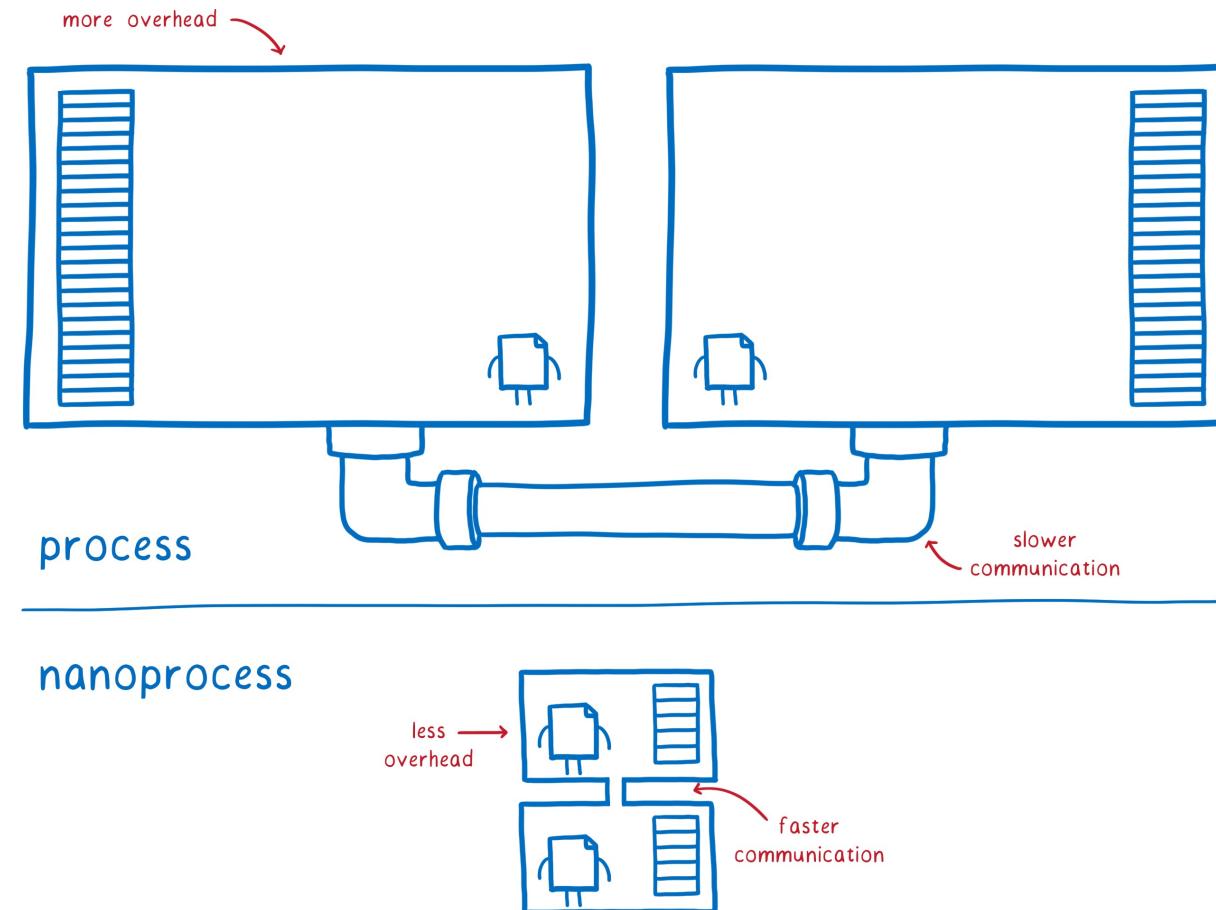
Vulnerable module



Process Isolation

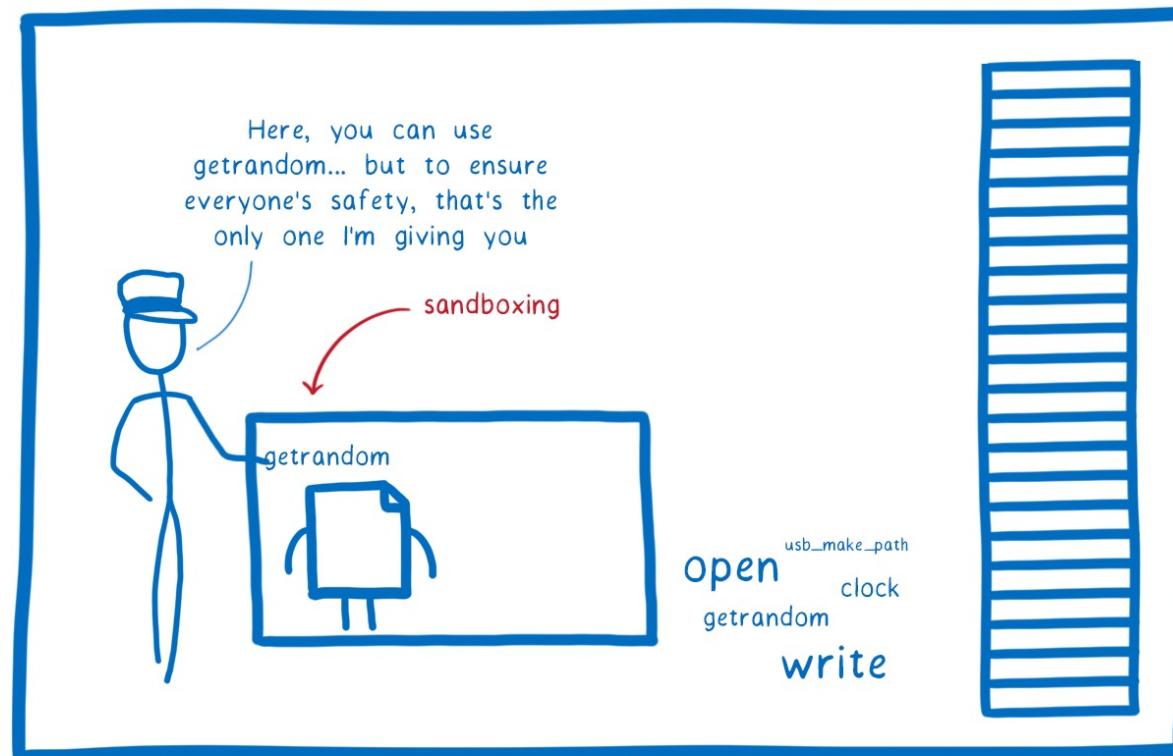


WebAssembly Nano-Process



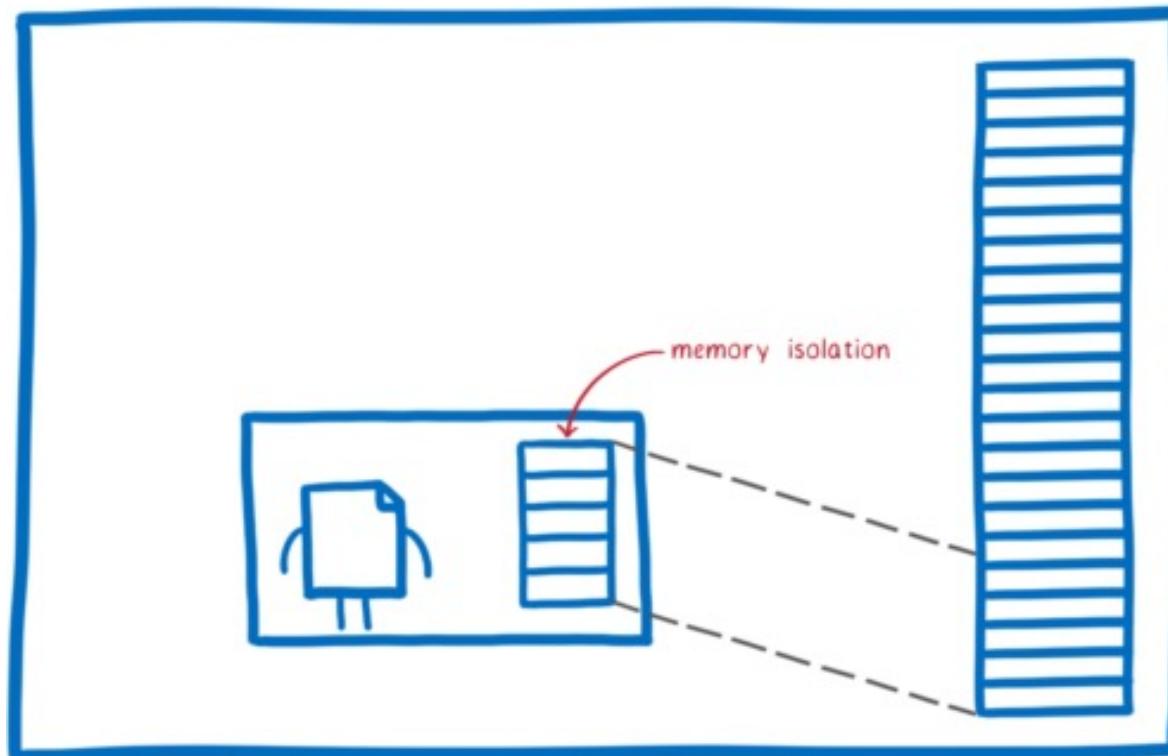
WebAssembly Nano-Process

1. Sandboxing



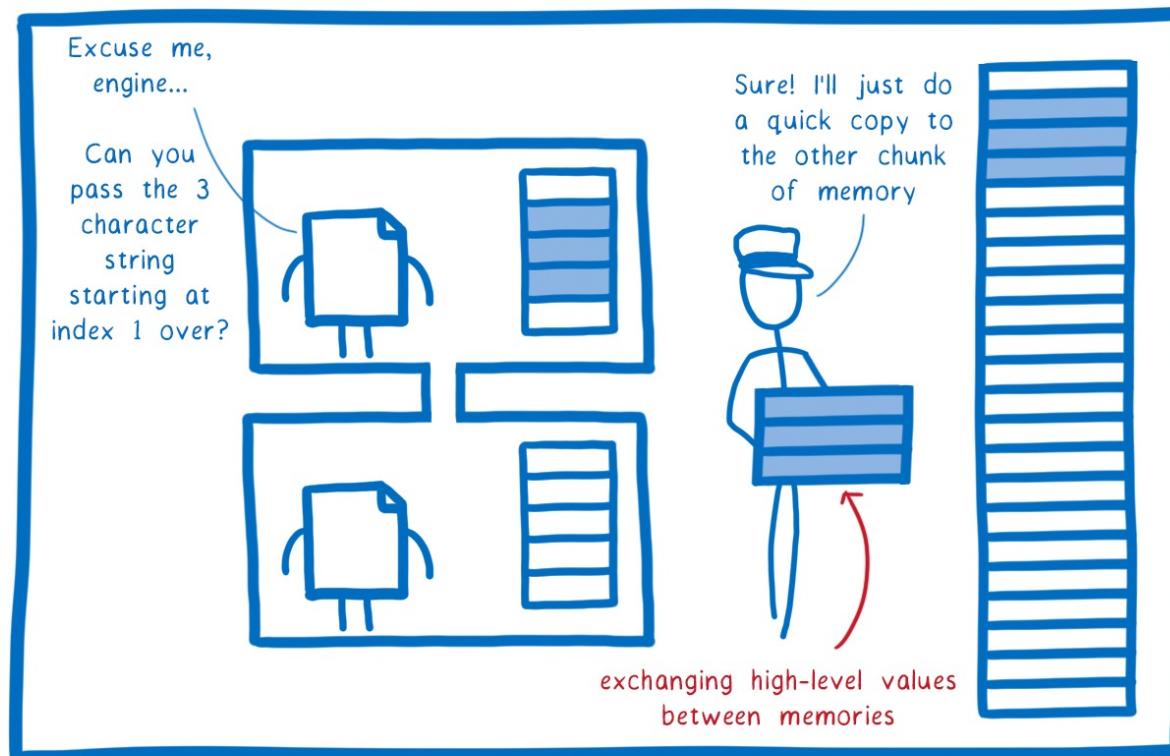
WebAssembly Nano-Process

2. Memory model



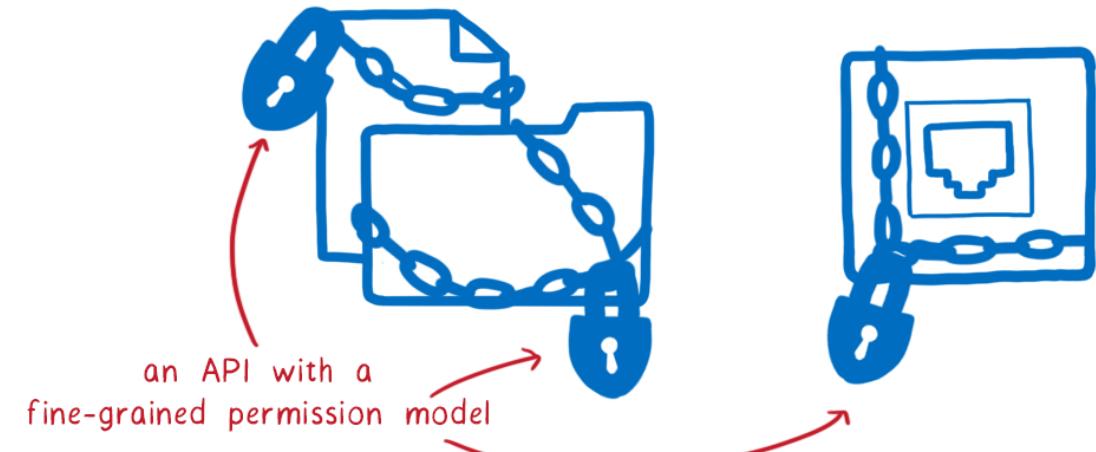
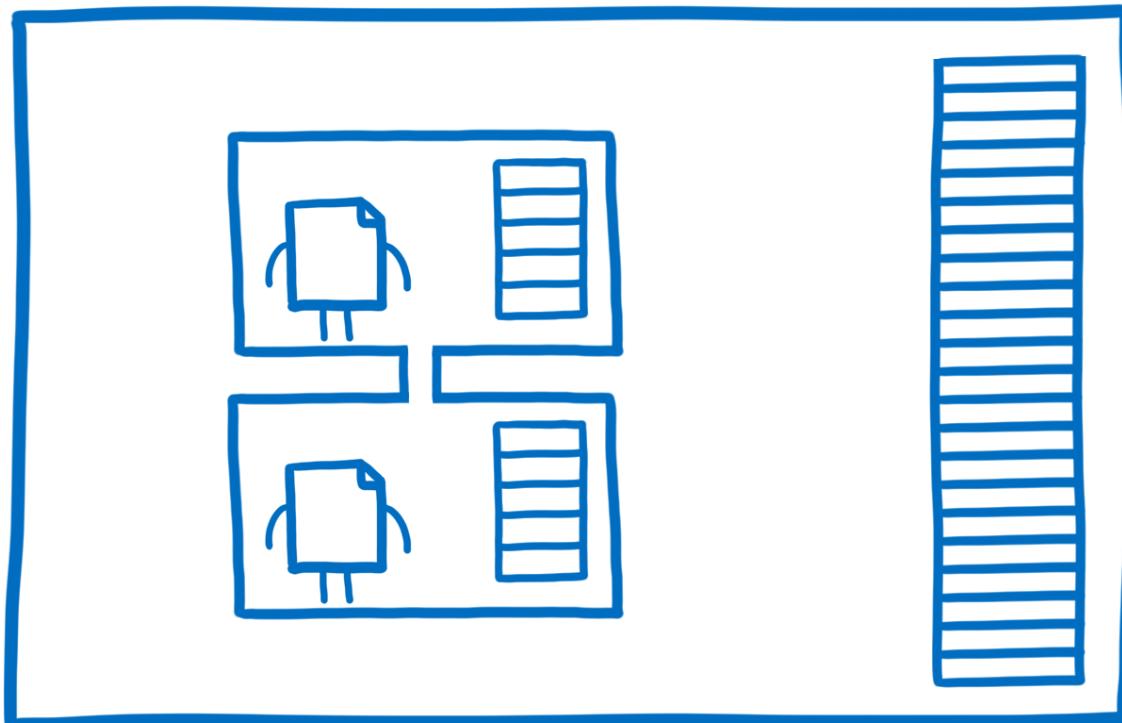
WebAssembly Nano-Process

3. Interface Types



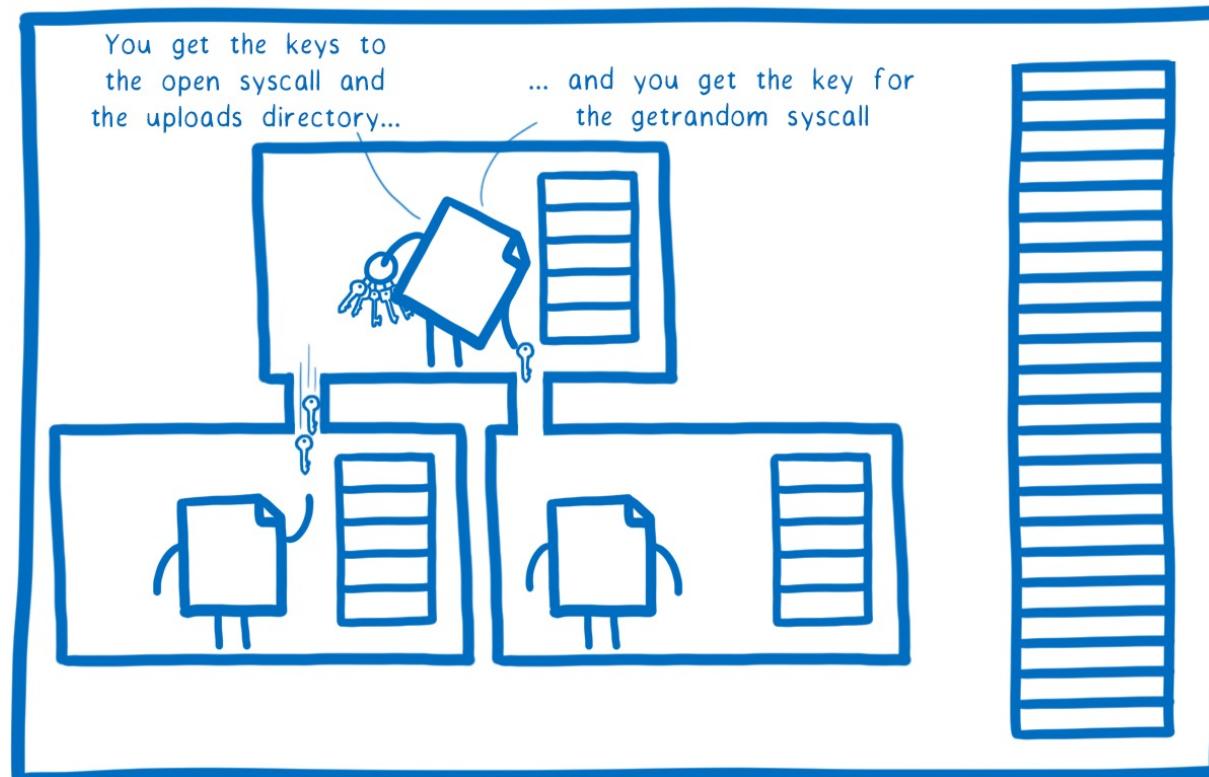
WebAssembly Nano-Process

4. WebAssembly System Interface

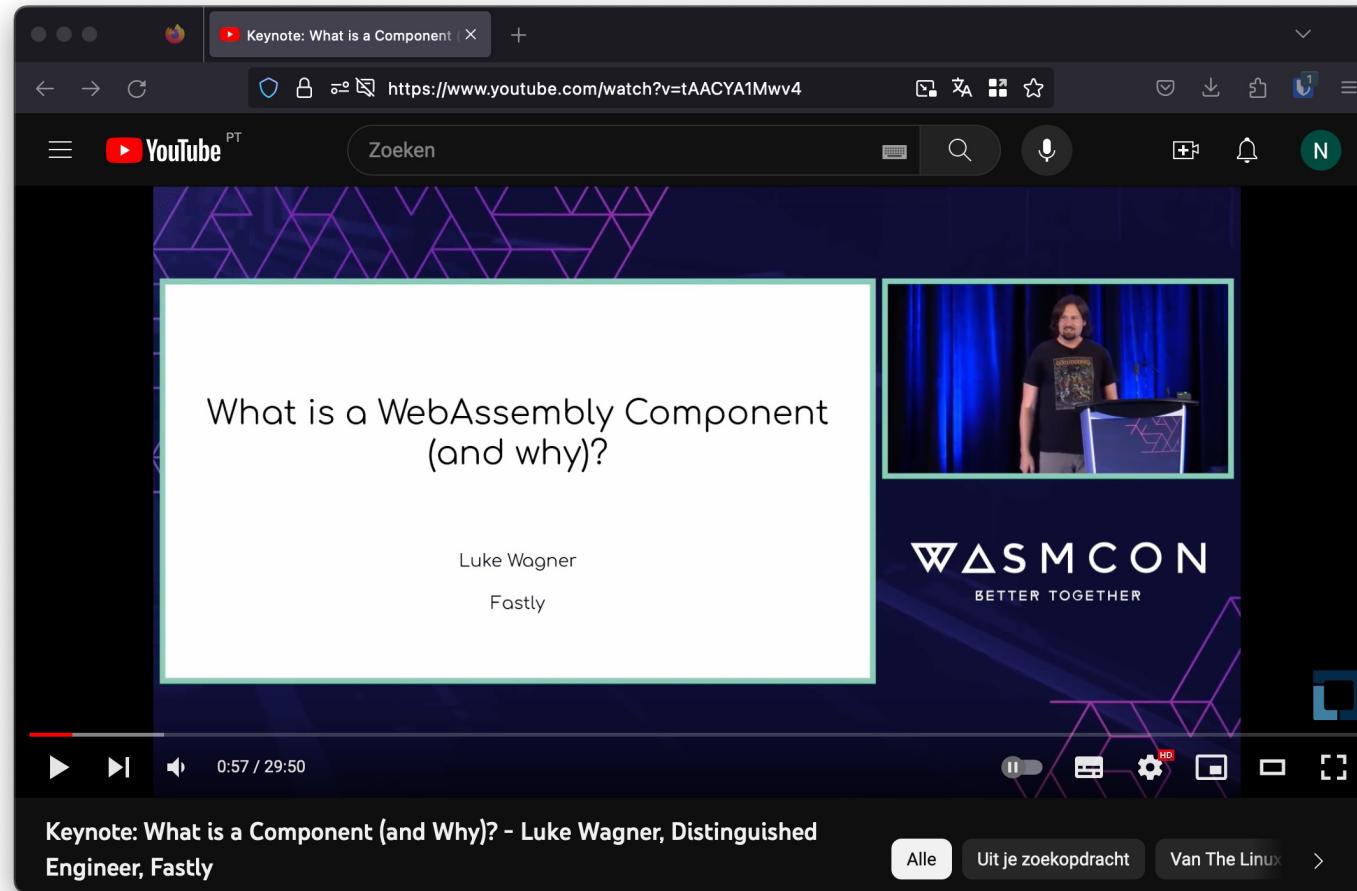


WebAssembly Nano-Process

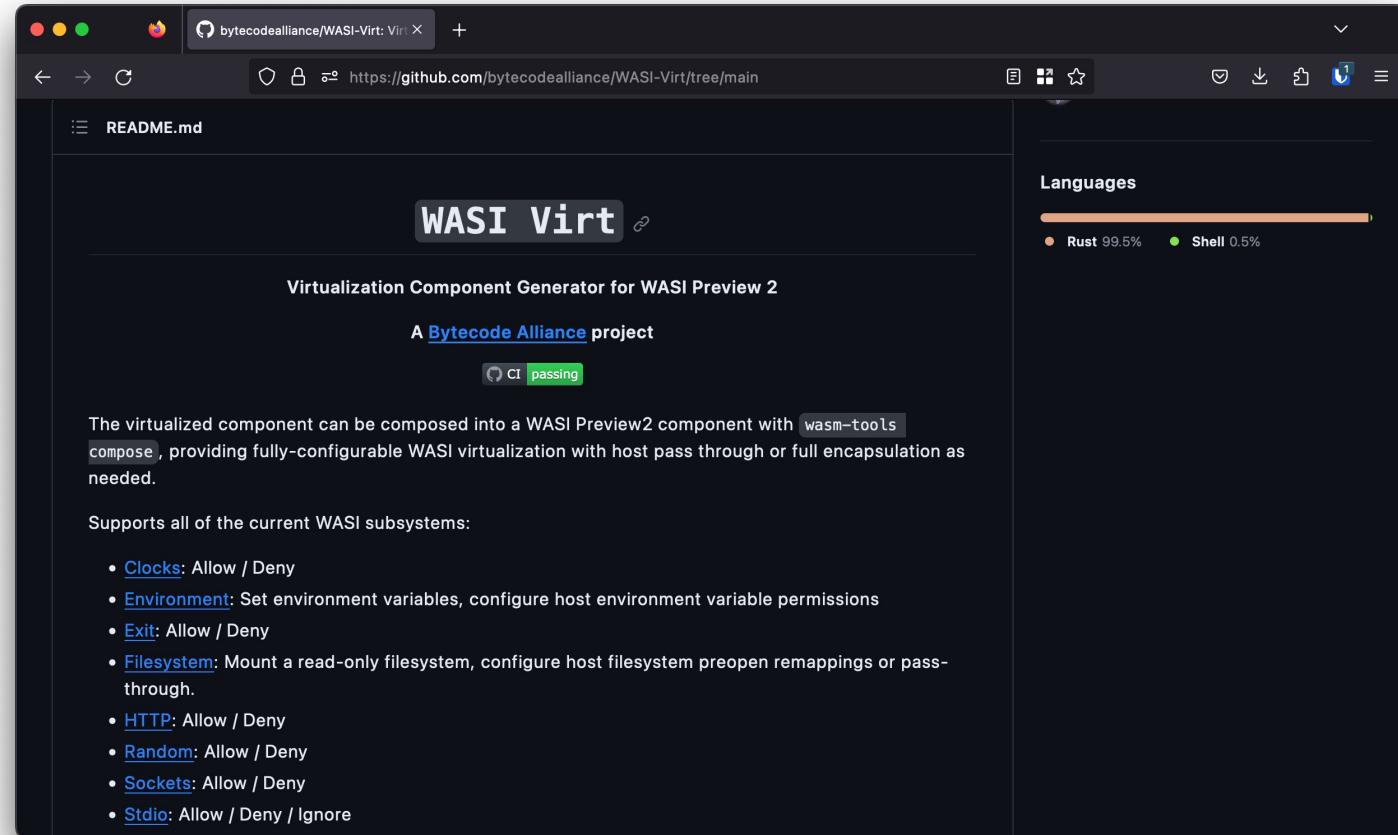
5. The missing link



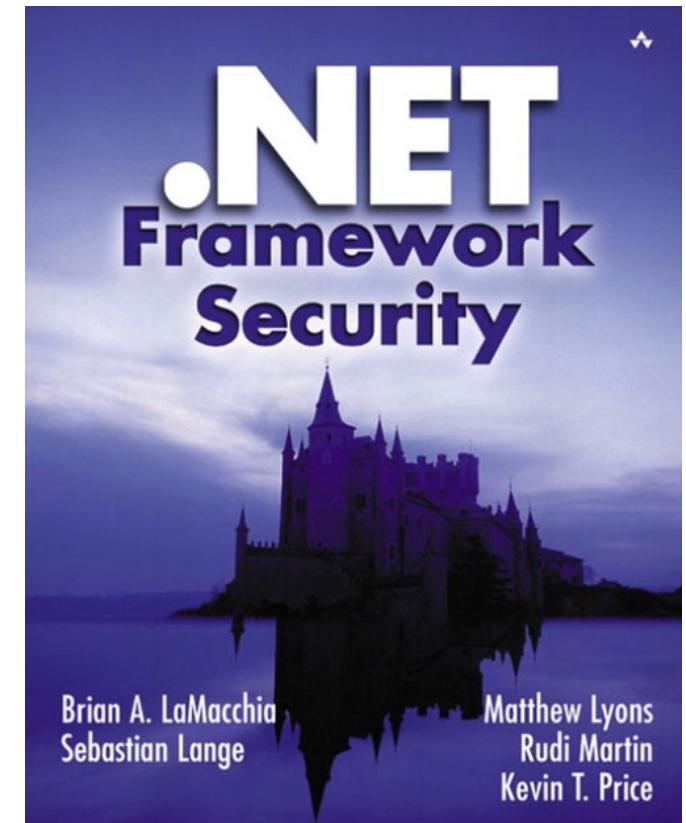
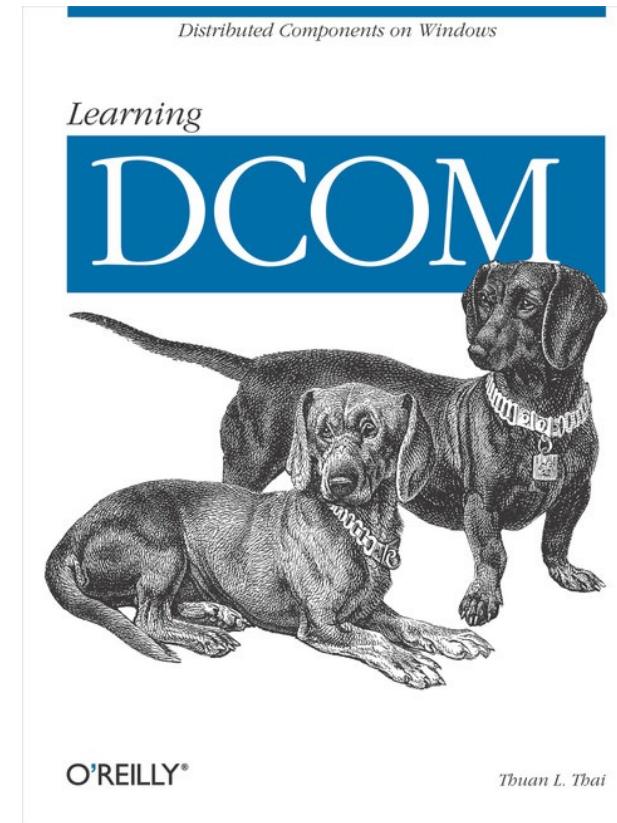
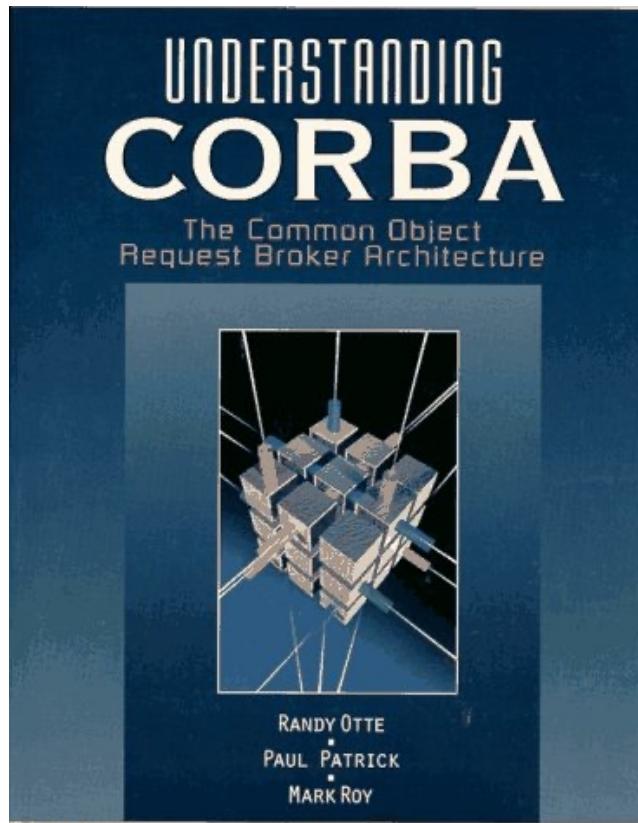
WebAssembly Component Model



WebAssembly Component Model



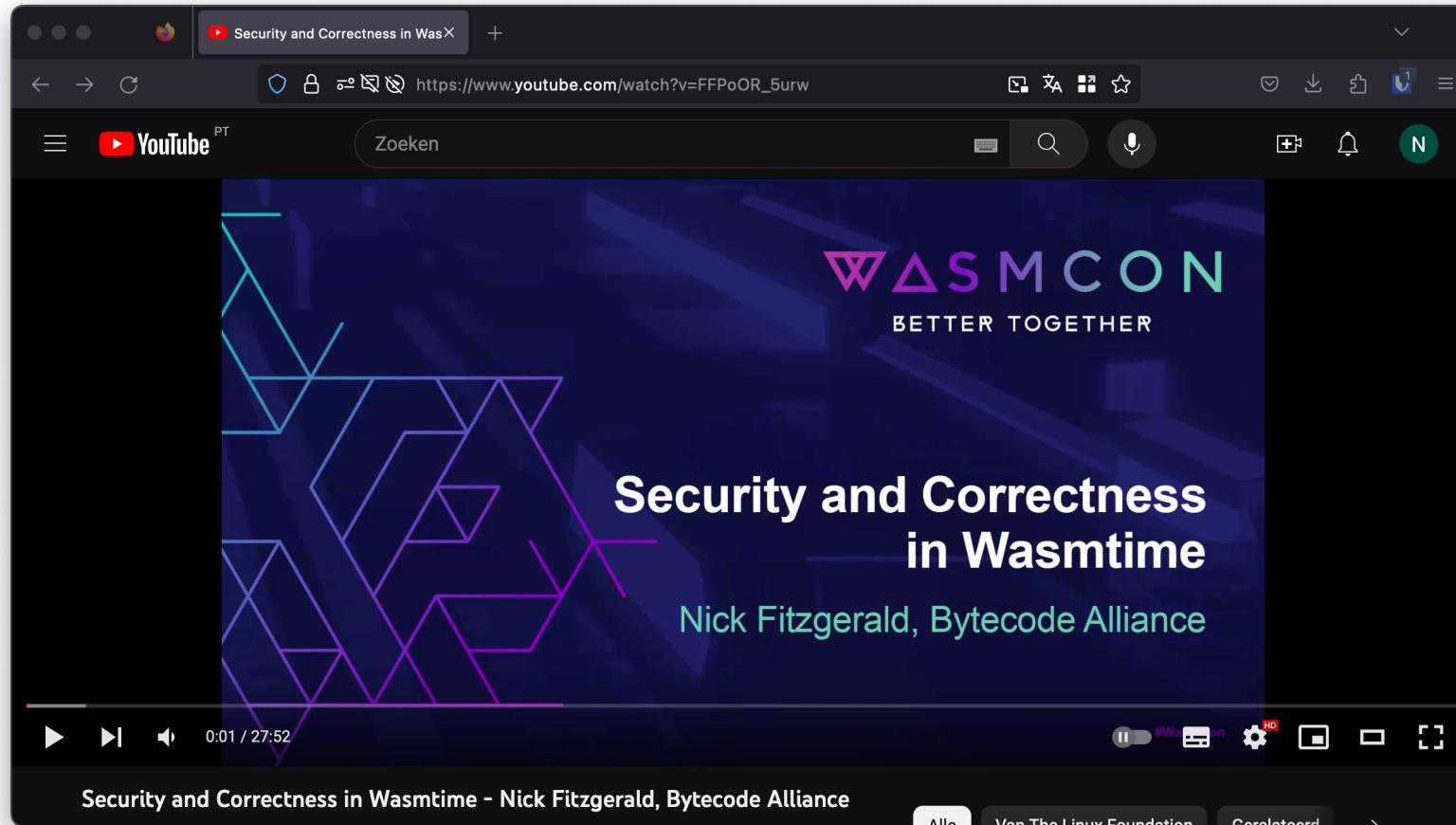
Have we seen this before?



Runtimes and Security

- Most security research published focusses on correctness of WASM runtimes/VM's based on e.g. fuzzing
- Bytecode Alliance Blogpost September 2022 by Nick Fitzgerald:
 - “Security and Correctness in Wasmtime”
 - Written in Rust → Using all it's LangSec features
 - Continues Fuzzing & formal verification
 - Security process & vulnerability disclosure

Runtimes and Security



Conclusion

- Cloud Native ❤️ WebAssembly
- WebAssembly has a lot of potential to be used to run, extend, and secure your applications
- Its as secure as the WebAssembly runtime implementation!
- I like top-down approach Bytecode Alliance is taking in moving forward, feedback will change/influence

Questions?

- <https://github.com/nielstanis/owaspbenelux2023>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Merci! Bedankt!