



OWASP 2023
GLOBAL
AppSec | WASHINGTON
DC
OCT 30 - NOV 3

Using WebAssembly to run, extend, and secure your application

Niels Tanis
Security Researcher



Who am I?

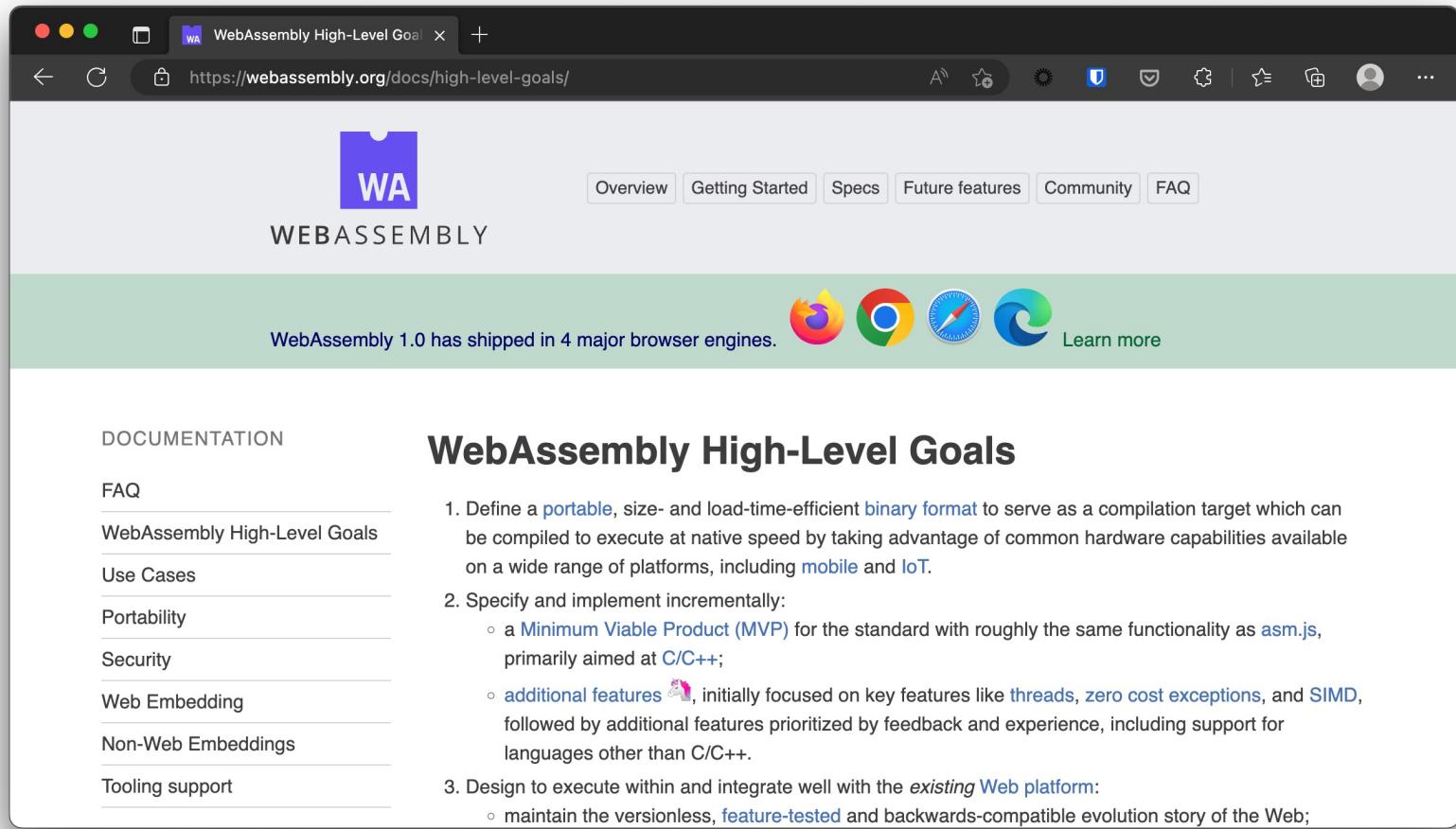
- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
 - Enjoying Rust!
- Microsoft MVP - Developer Technologies



VERACODE

@nielstanis@infosec.exchange

WebAssembly



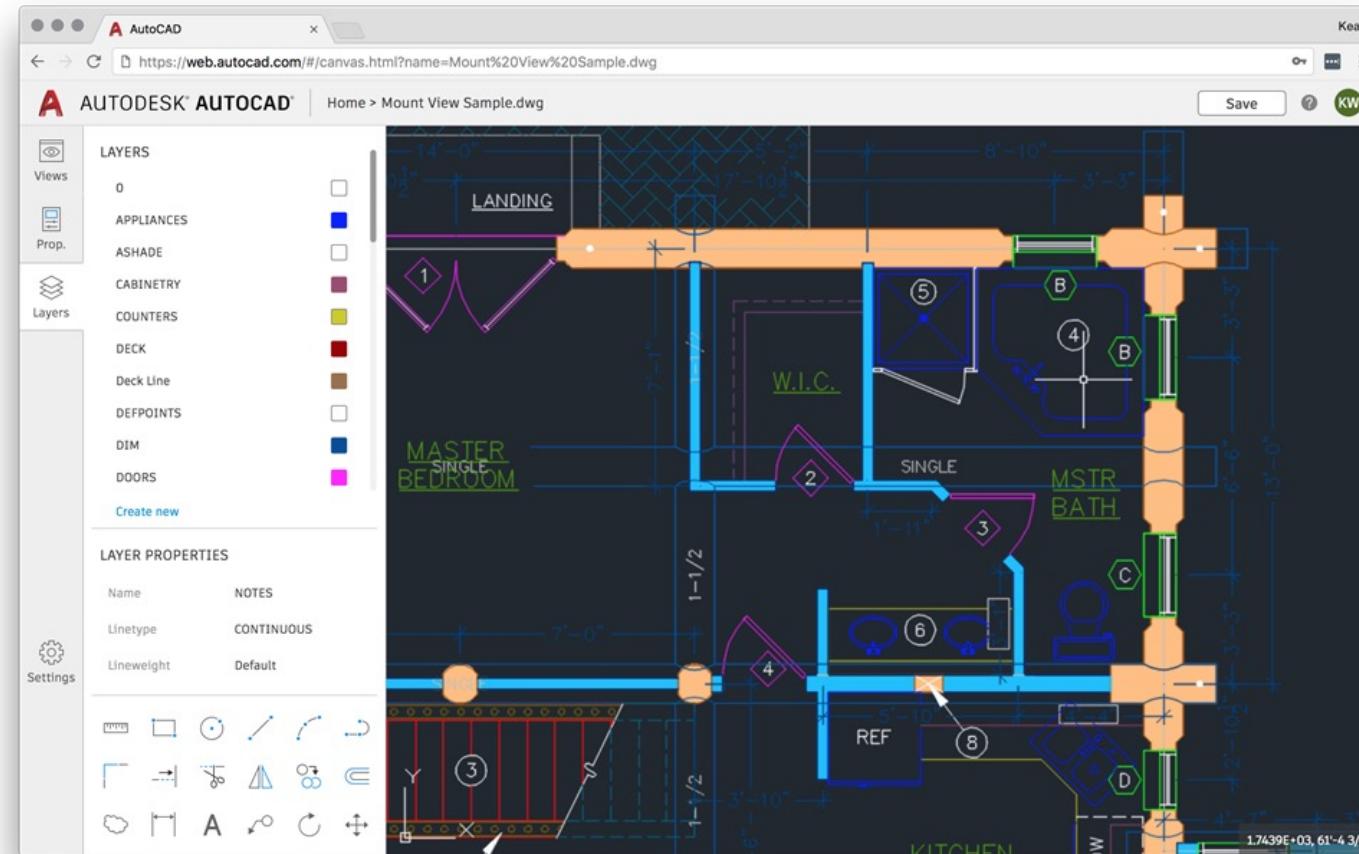
The screenshot shows a web browser window displaying the [WebAssembly High-Level Goals](https://webassembly.org/docs/high-level-goals/) page. The page has a dark header with the WebAssembly logo and navigation links for Overview, Getting Started, Specs, Future features, Community, and FAQ. Below the header, a green banner 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 numbered 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;



@nielstanis@infosec.exchange

WebAssembly - AutoCAD



WebAssembly - SDK's

A screenshot of a Medium article page. The title is "Introducing the Disney+ Application Development Kit (ADK)". It was published by Mike Hanley on Sep 8, 2021. The article has 415 upvotes and 4 comments. The content discusses the Disney+ Application Development Kit (ADK) and its benefits.

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 published by Alexandru Ene on January 27, 2022. The article discusses how Prime Video uses WebAssembly to update its app for thousands of device types. It includes a section on "CLOUD AND SYSTEMS".

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

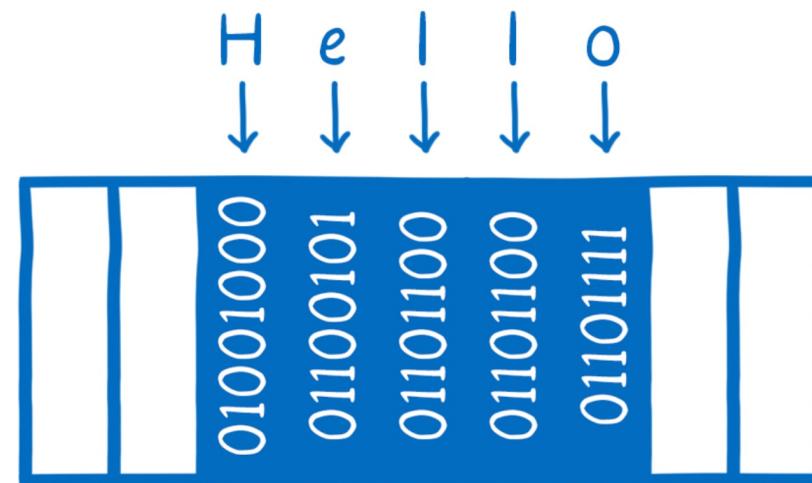
- Binary instruction format for stack-based virtual machine similar to .NET CLR running MSIL or JVM running bytecode
- Designed as a portable compilation target
- The security model of WebAssembly:
 - Protect users from buggy or malicious modules
 - Provide developers with useful primitives and mitigations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

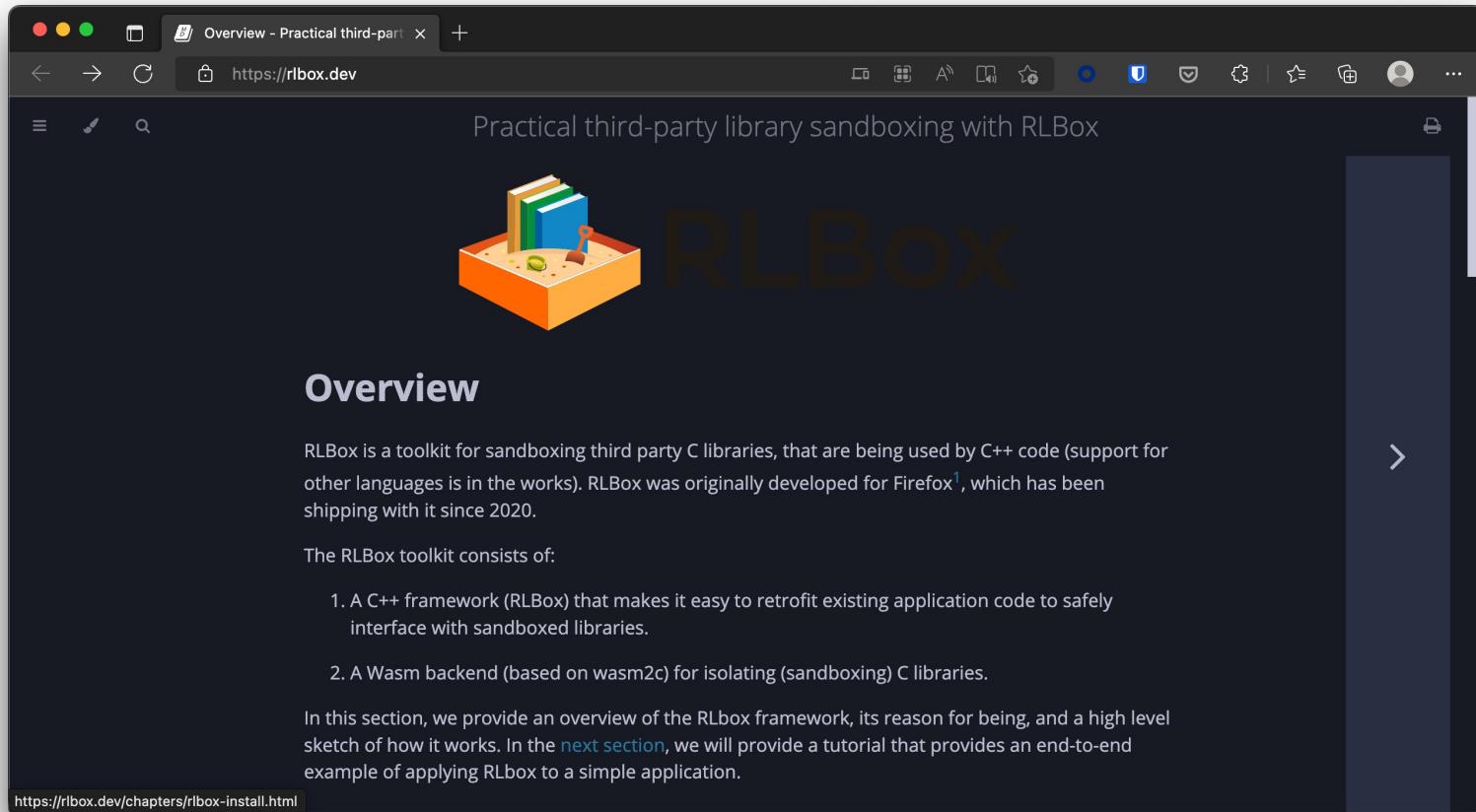
```
int number = Convert.ToInt32(Console.ReadLine());
Console.WriteLine($"Number {number}");
if (number>5)
```

```
Console.WriteLine("Number is larger than 5");
```

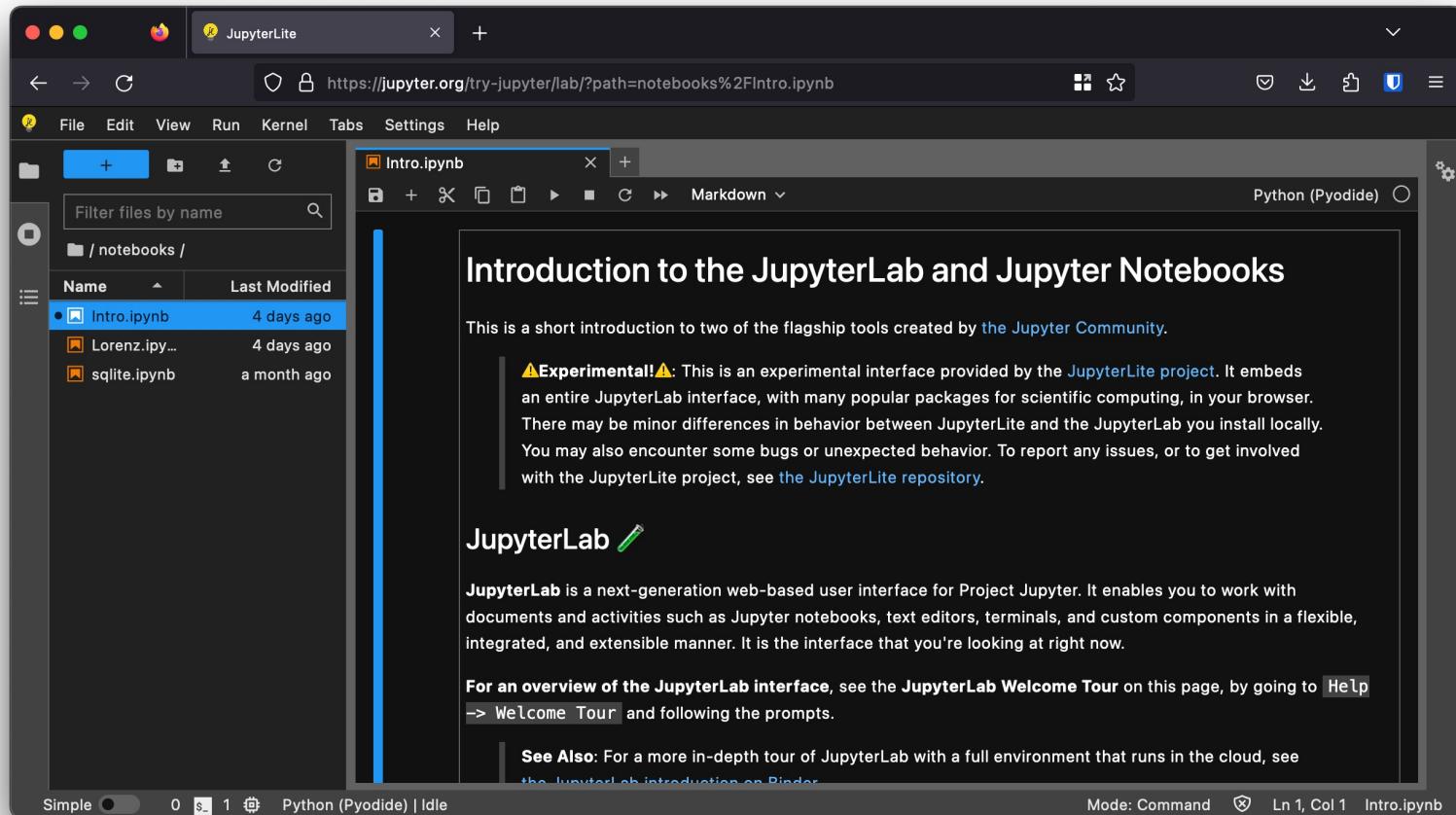
```
Console.WriteLine("Number is smaller than 5");
```

```
Console.WriteLine("Done!");
```

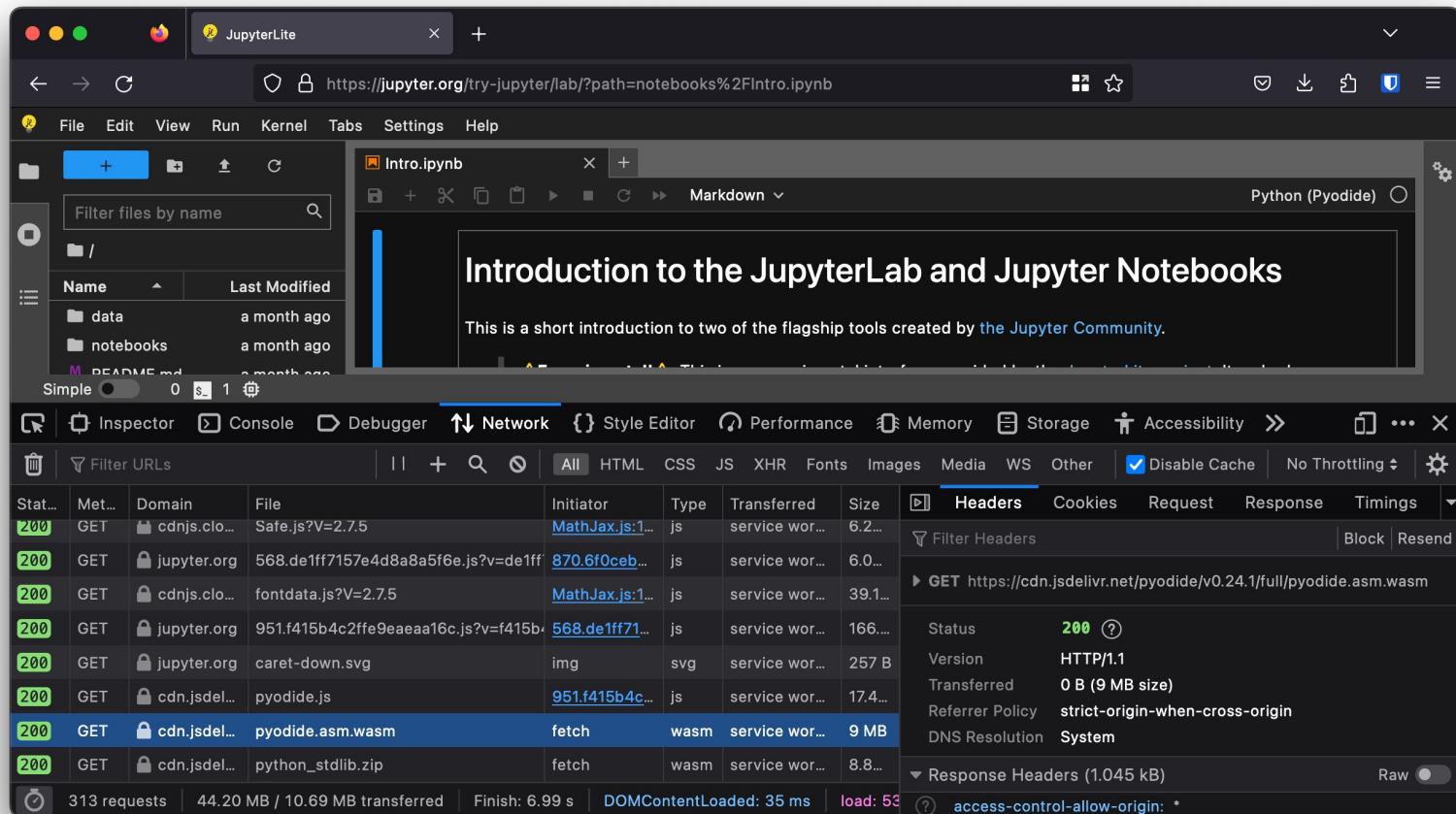
FireFox RLBox



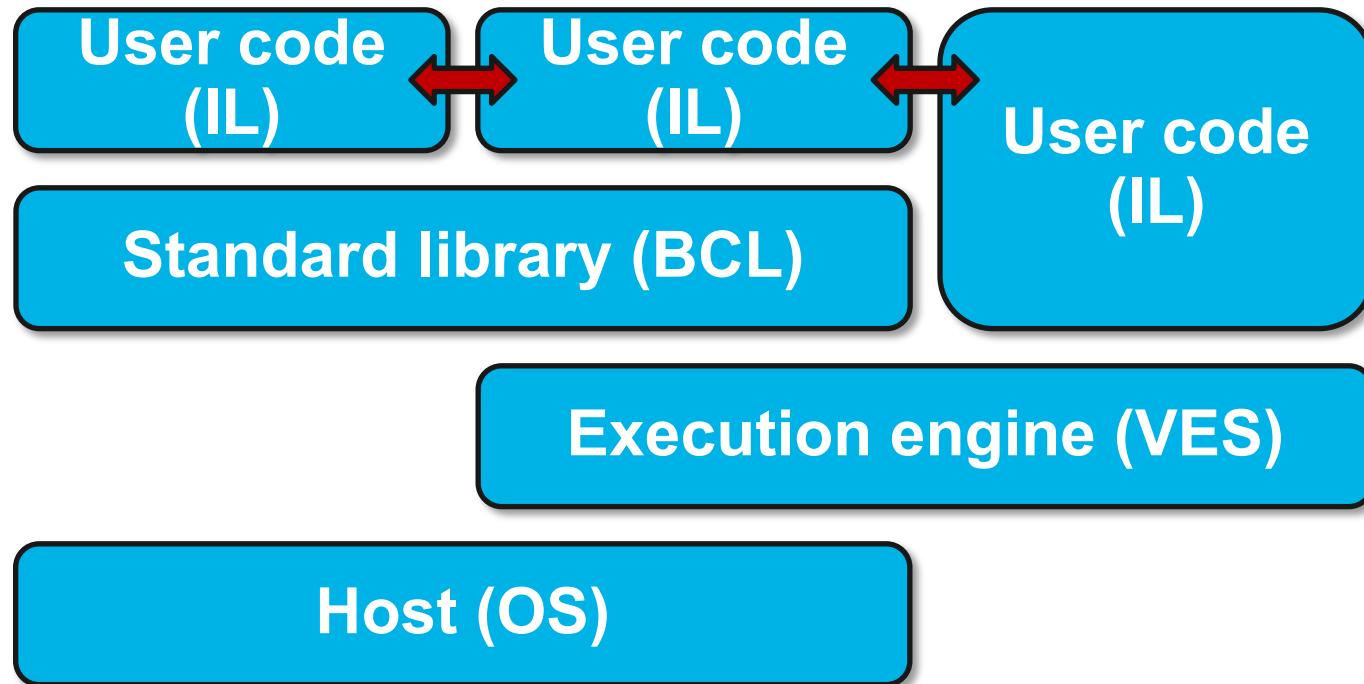
Python on WASM



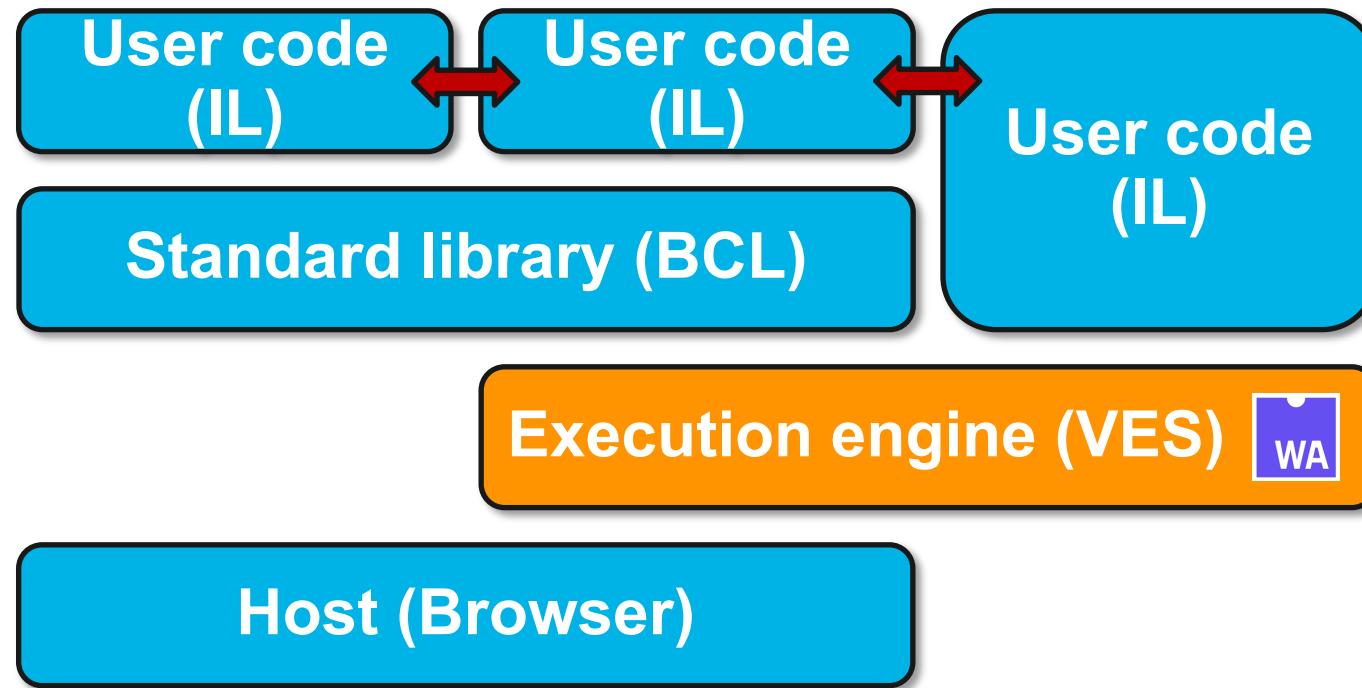
Python on WASM



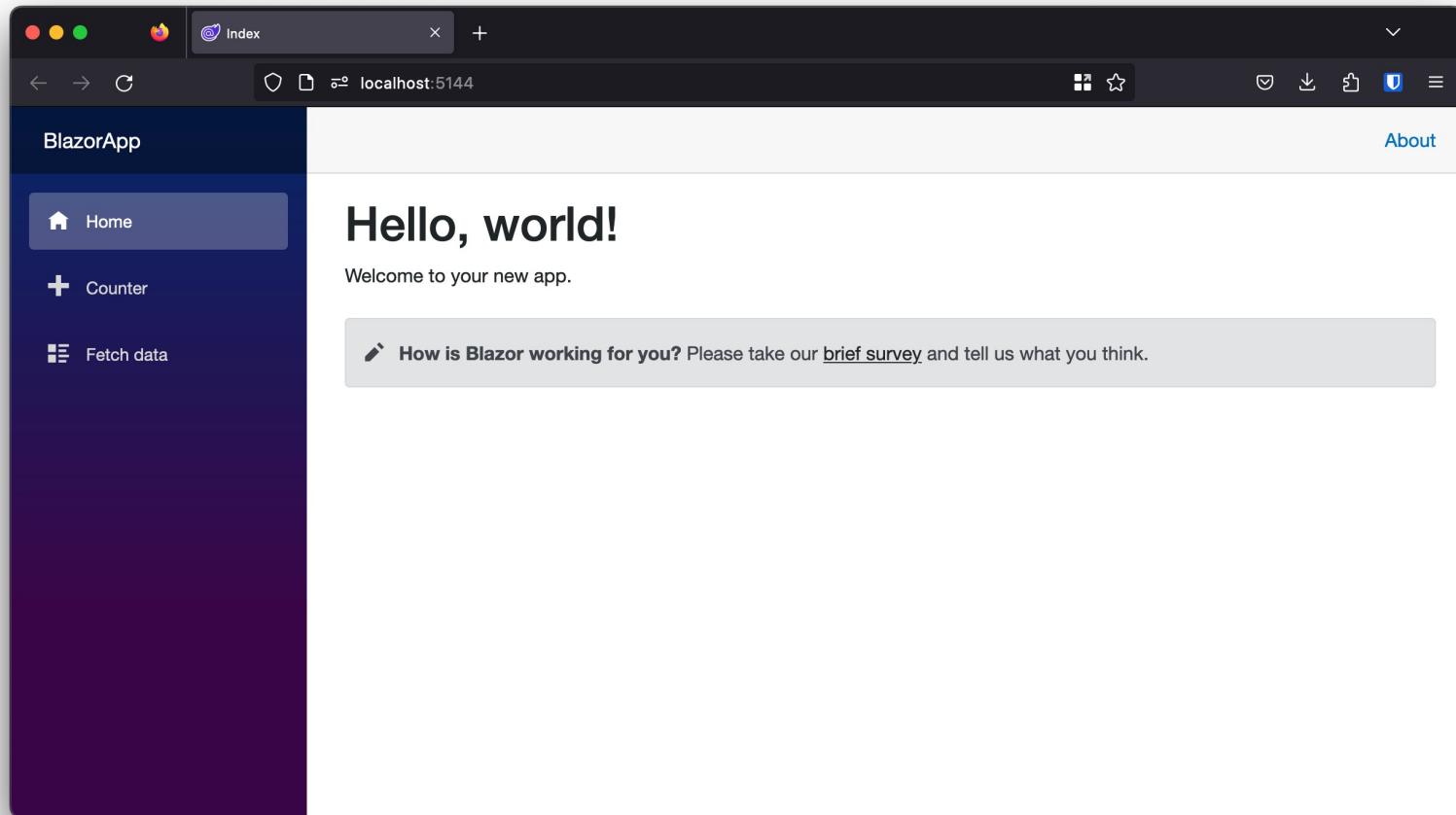
Running .NET on WebAssembly



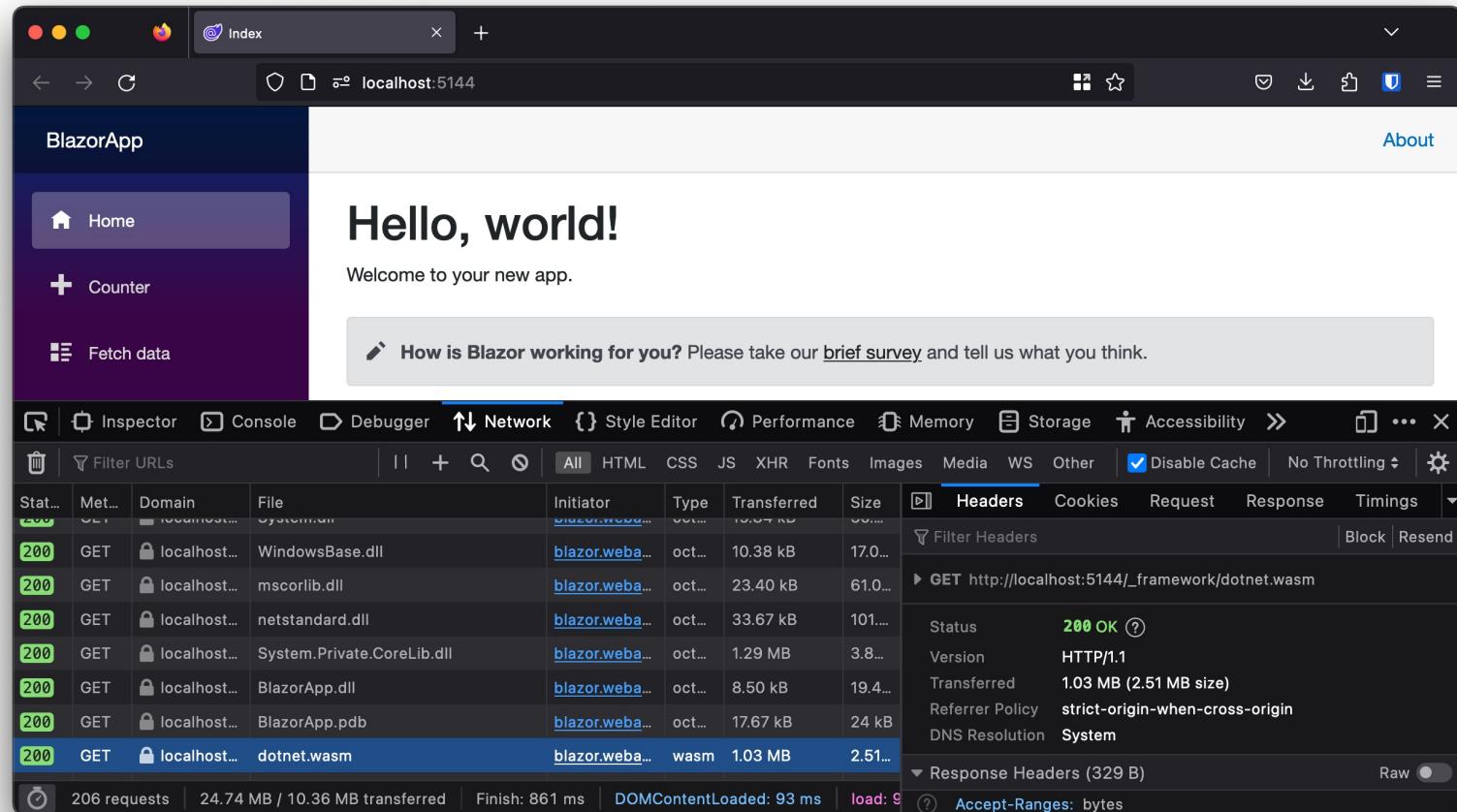
Running .NET on WebAssembly



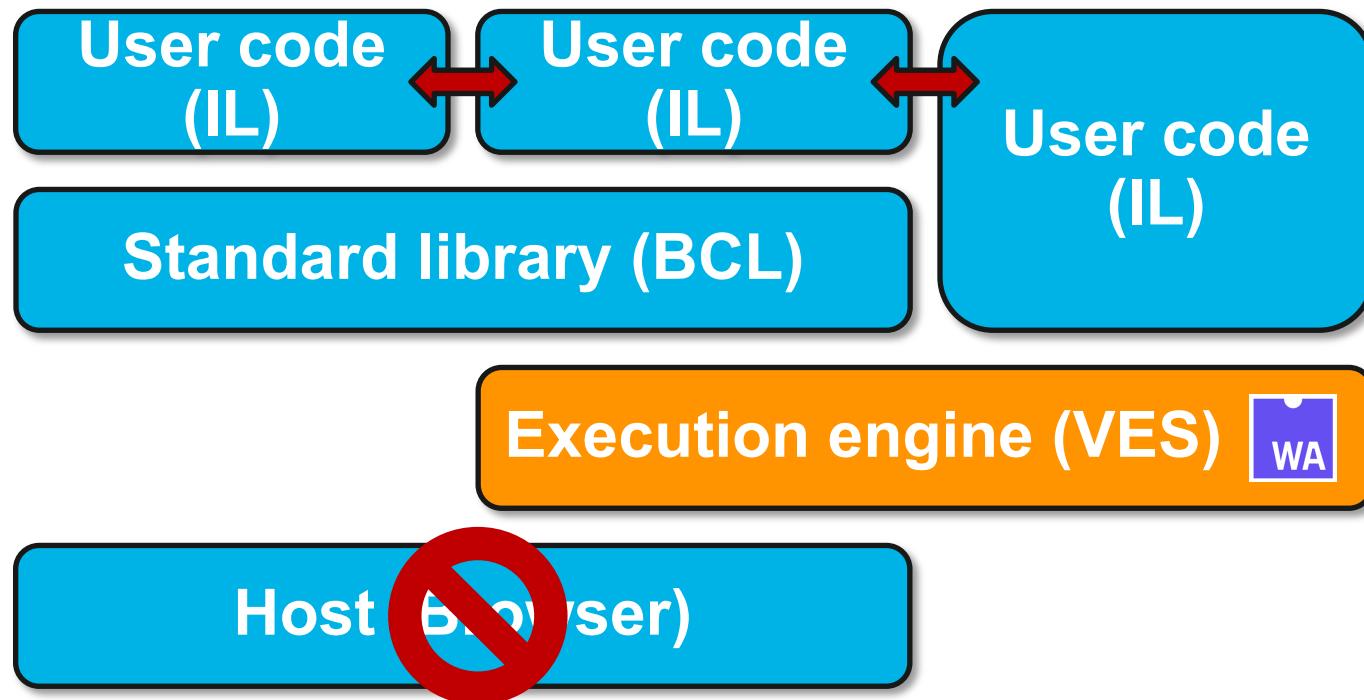
Blazor WebAssembly



Blazor WebAssembly



Running .NET on WebAssembly



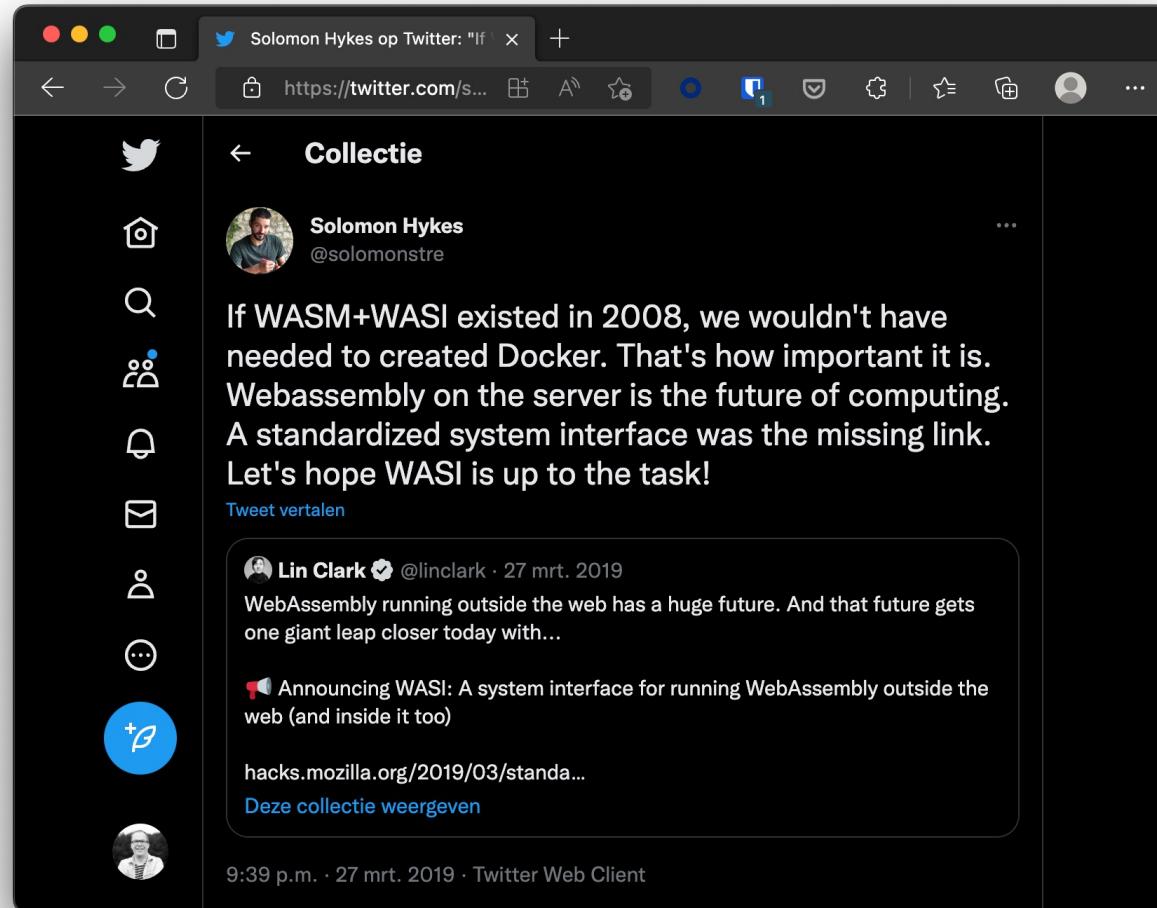
WebAssembly System Interface WASI

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

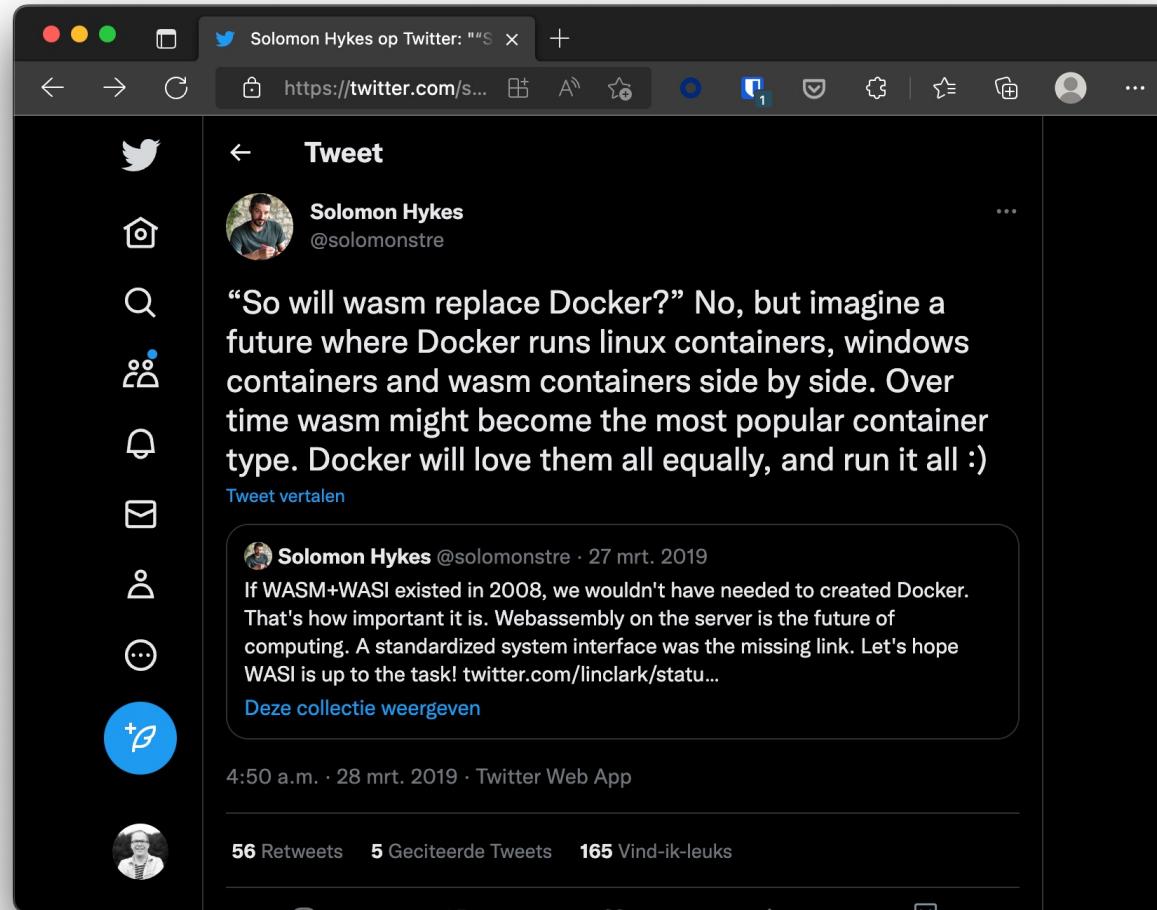
WebAssembly System Interface WASI

- 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



Docker vs WASM & WASI



Docker & WASM

The screenshot shows a web browser window with the title "Introducing the Docker+Wasm Technical Preview". The header bar includes the Docker logo and the text "Wasm is a fast, light alternative to Linux containers — try it out today in the Docker+Wasm Technical Preview". Below the header, the main content features the heading "Introducing the Docker+Wasm Technical Preview" in large, bold, dark blue font. A small profile picture of Michael Irwin is on the left, followed by his name "MICHAEL IRWIN" and the date "Oct 24 2022". At the bottom, there is a paragraph about the Docker+Wasm Technical Preview being available and producing buzz.

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.

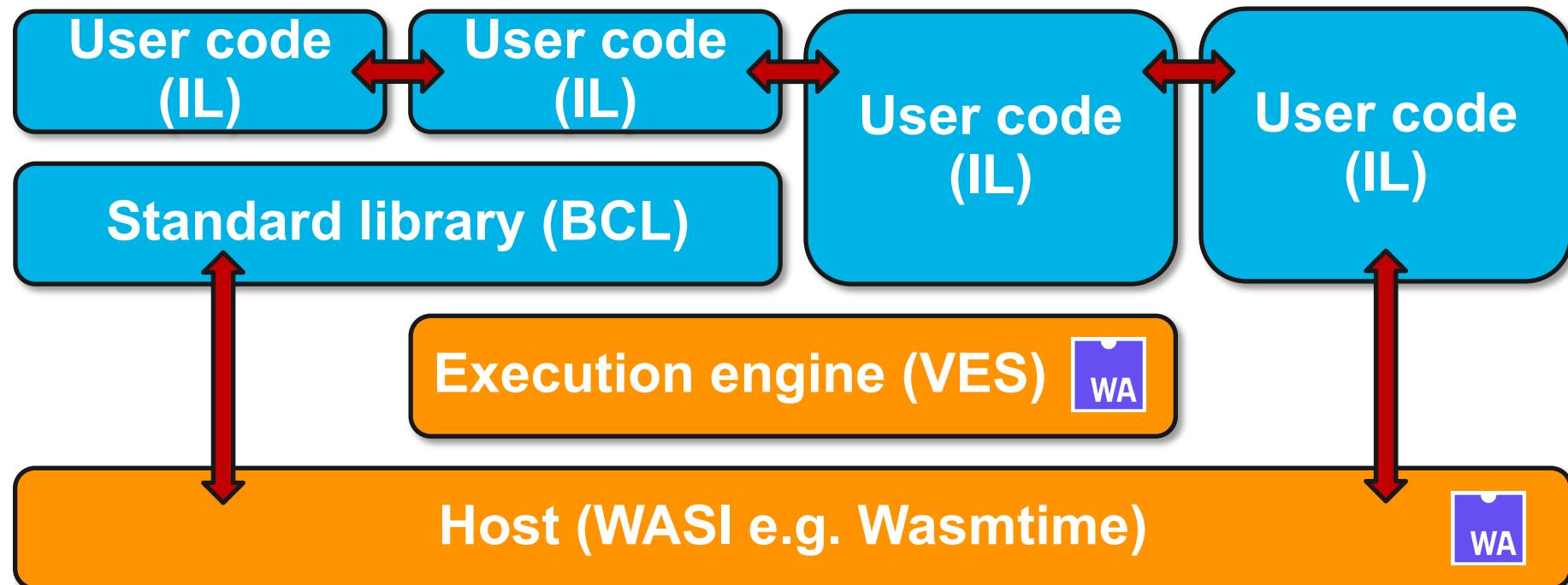
The screenshot shows a web browser window with the same title as the first screenshot. The main content area contains a diagram illustrating the Docker Engine architecture for Docker+Wasm. The diagram shows the Docker Engine at the top, which interacts with a central "containerd" component. Three separate "containerd-shim" components are shown, each managing a "runc" process. The middle "containerd-shim" also manages a "wasmedge" process, which in turn runs a "Wasm Module". Arrows indicate the flow of communication between the Docker Engine, containerd, containerd-shim, runc, wasmedge, and Wasm Module.

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



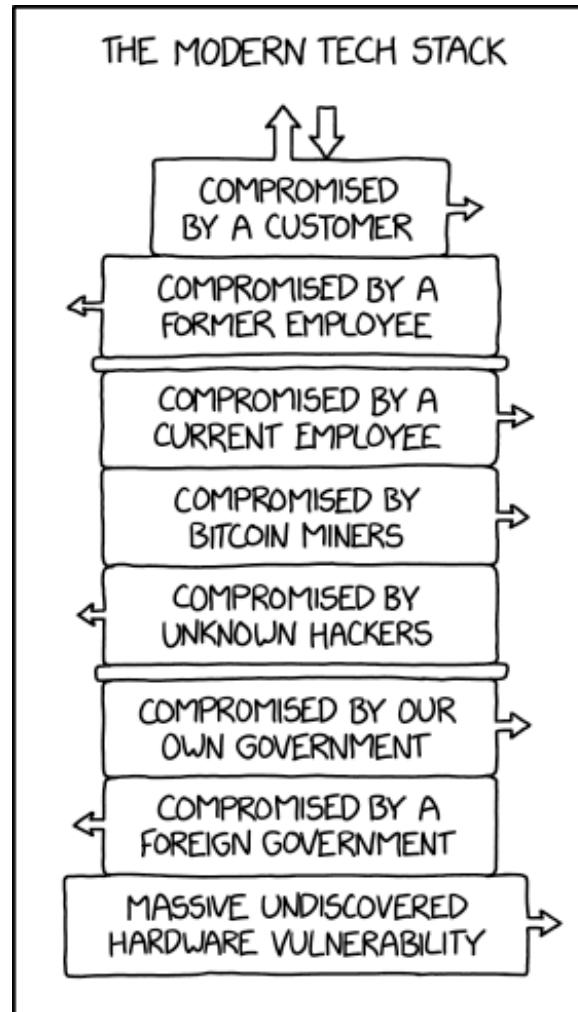
Experimental WASI SDK for .NET



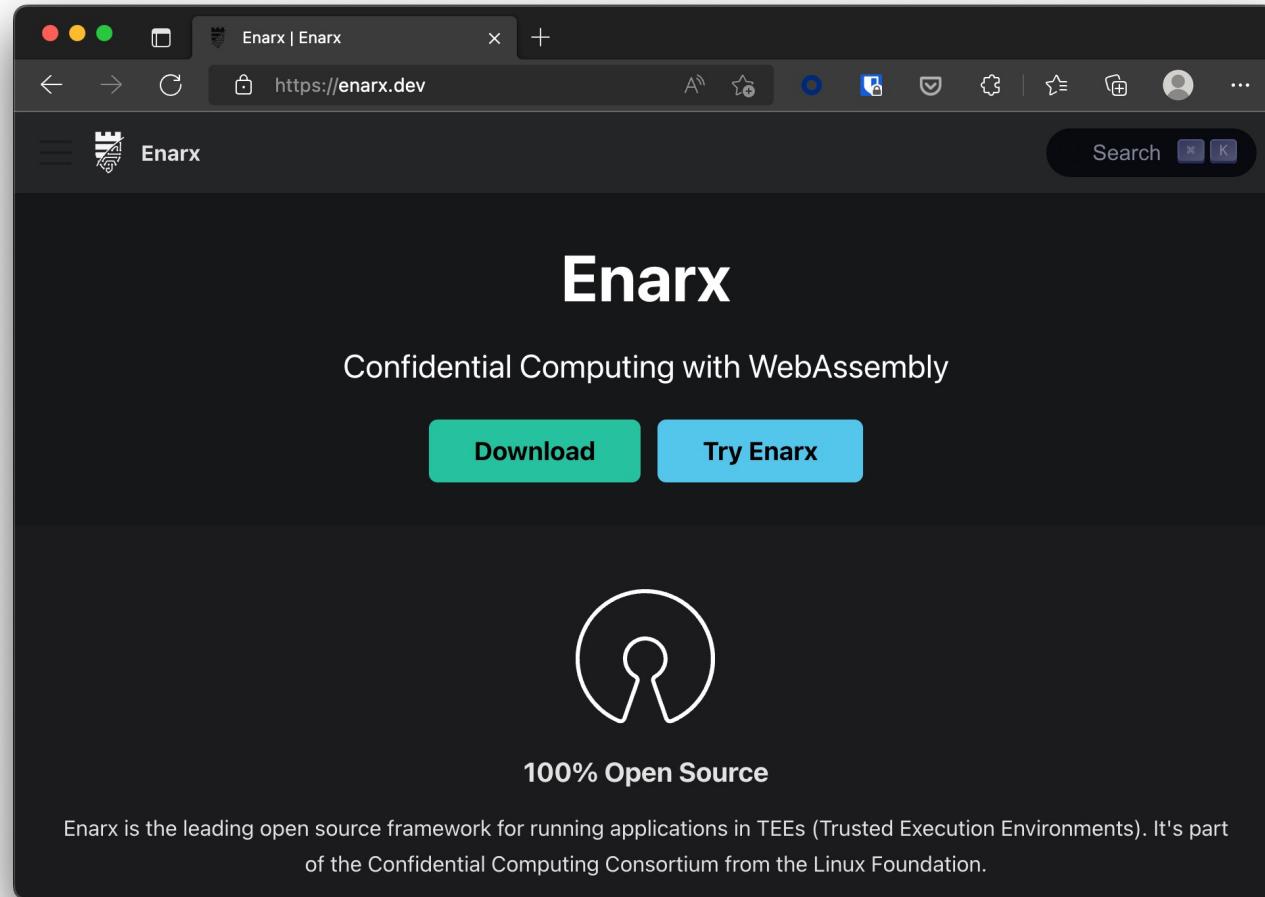
Extending .NET with WASM

- 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

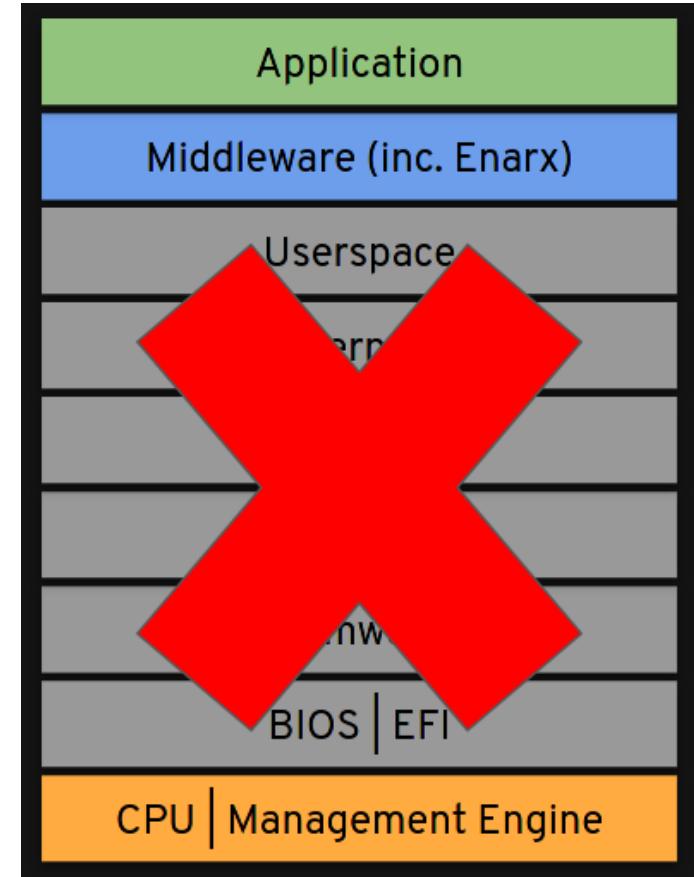


Enarx



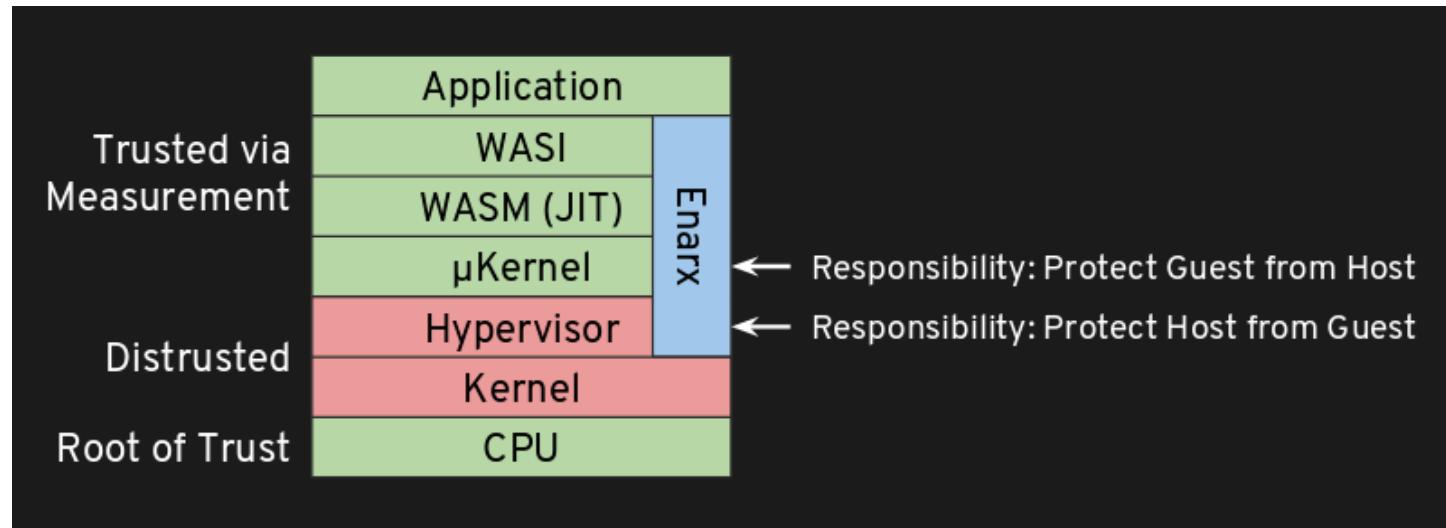
Enarx Threat Model

- 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

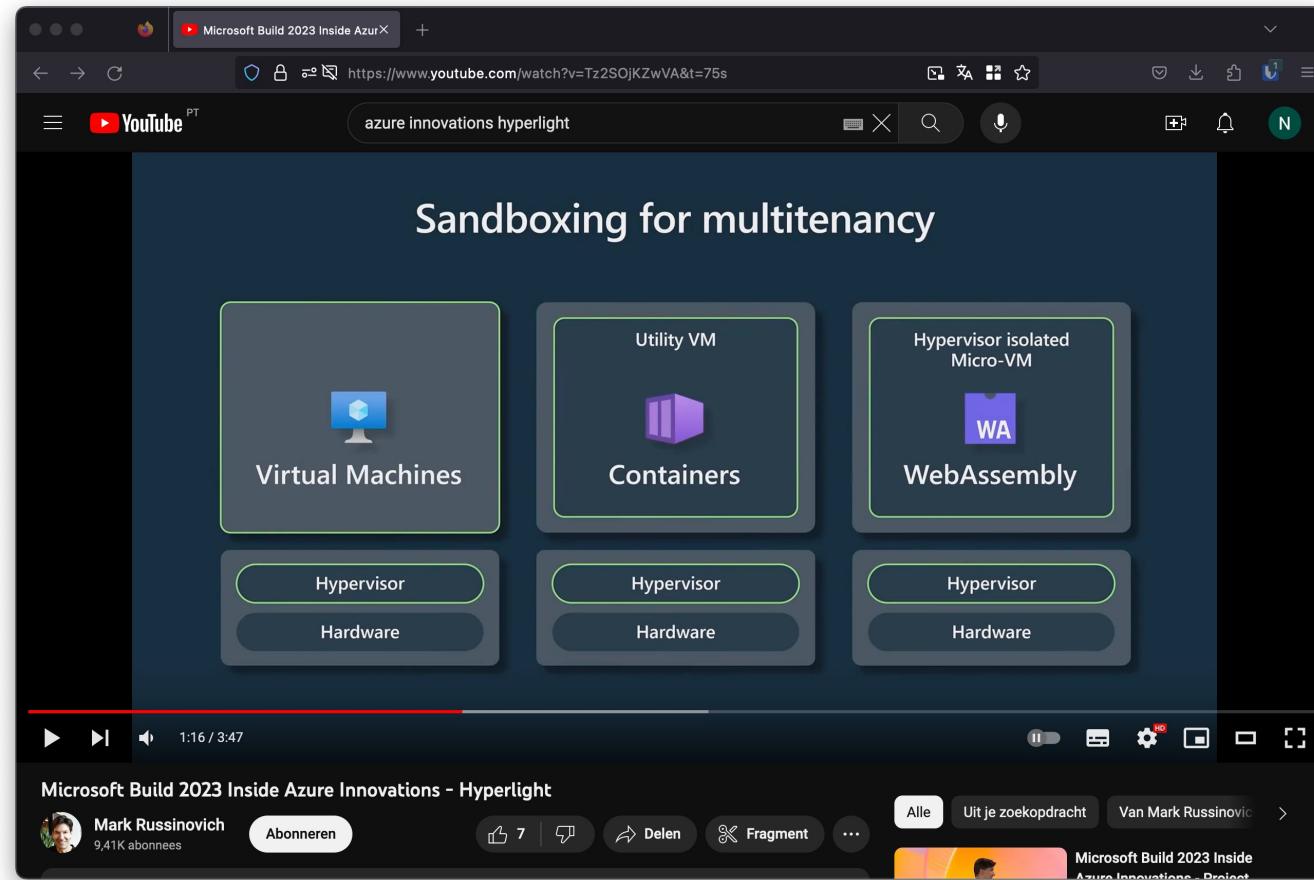


Enarx

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

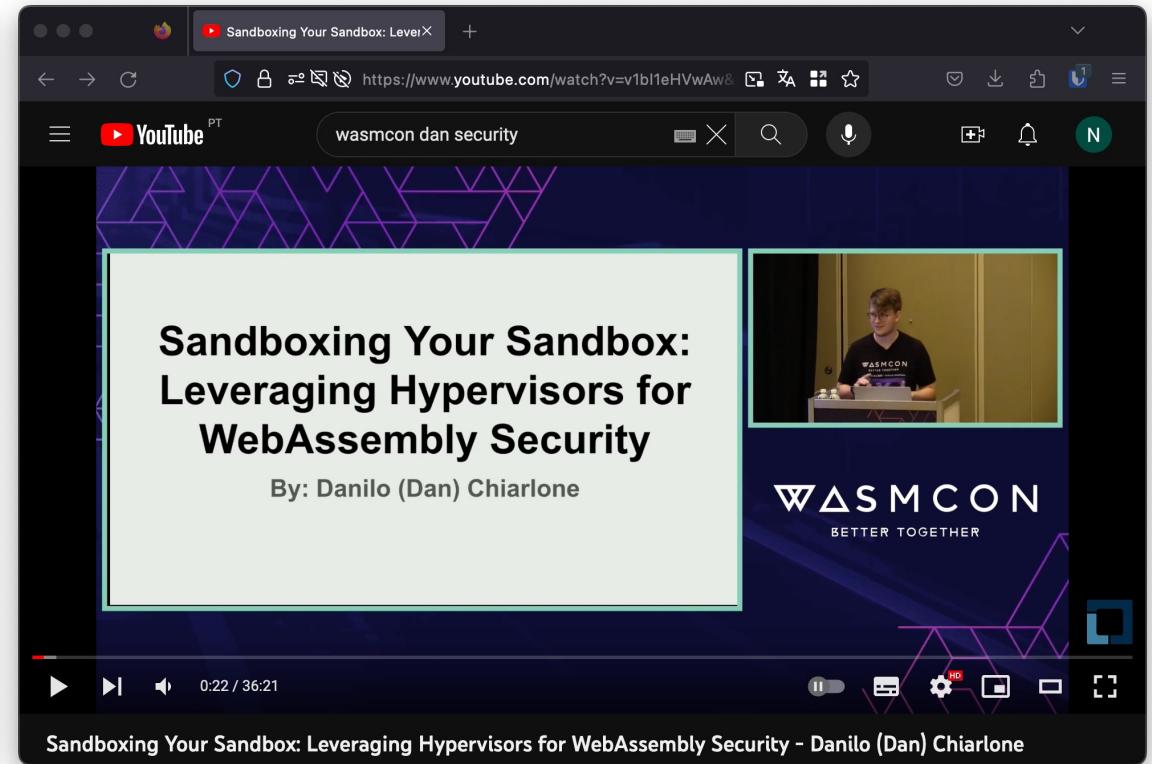
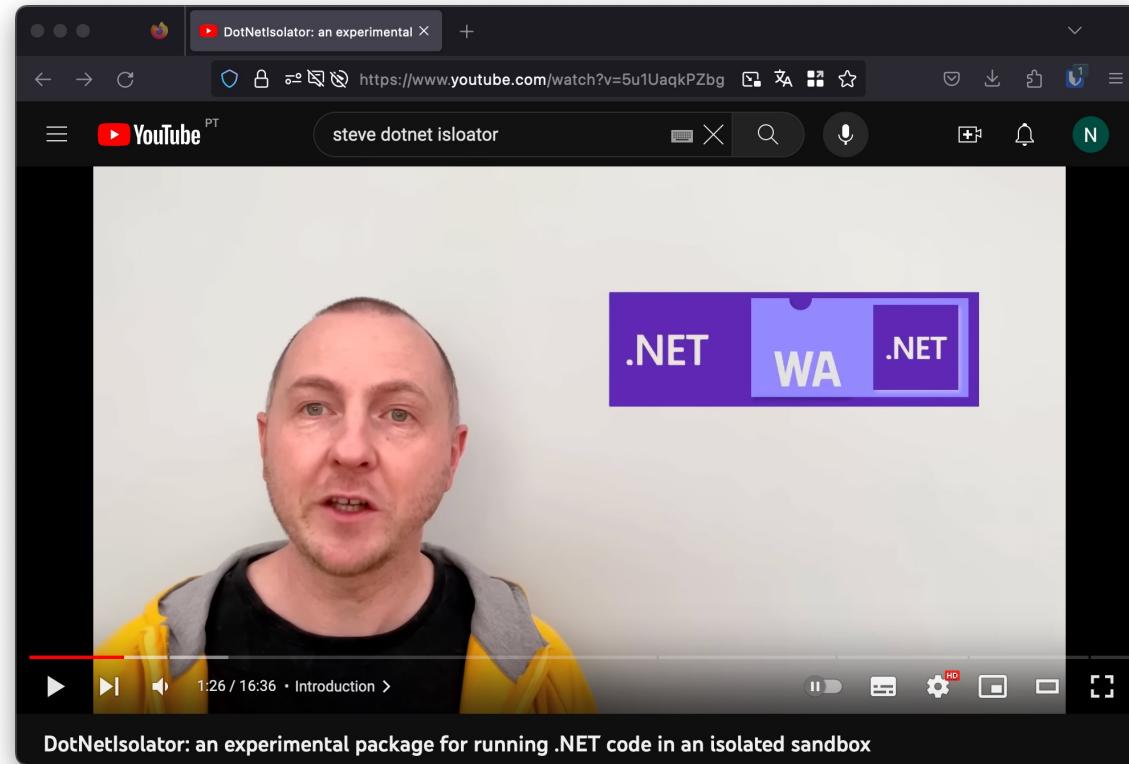


Project Hyperlight



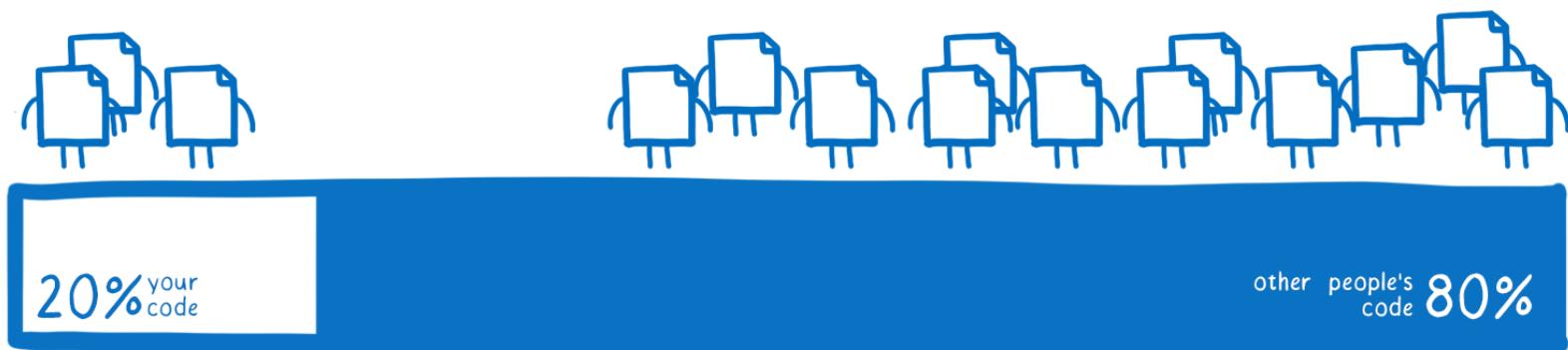
@nielstanis@infosec.exchange

DotNetIsolator & Project Hyperlight

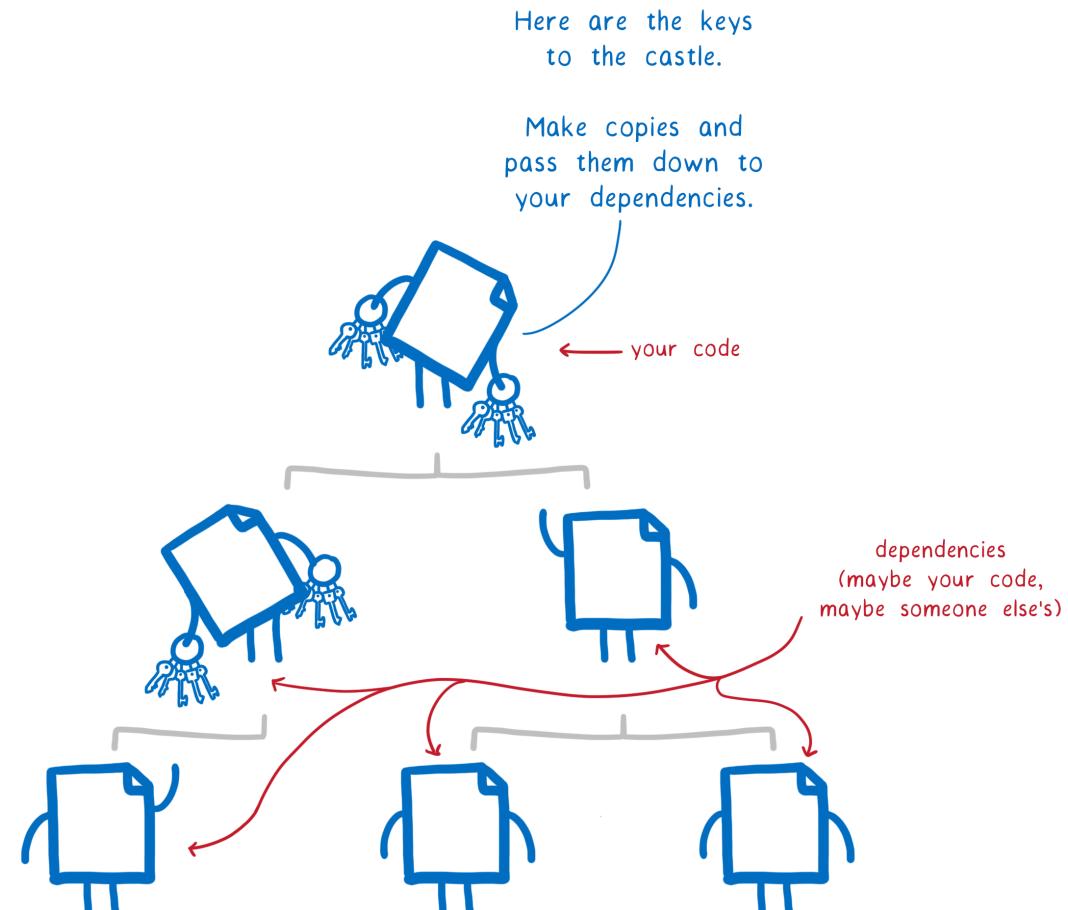


WASM - What's next?

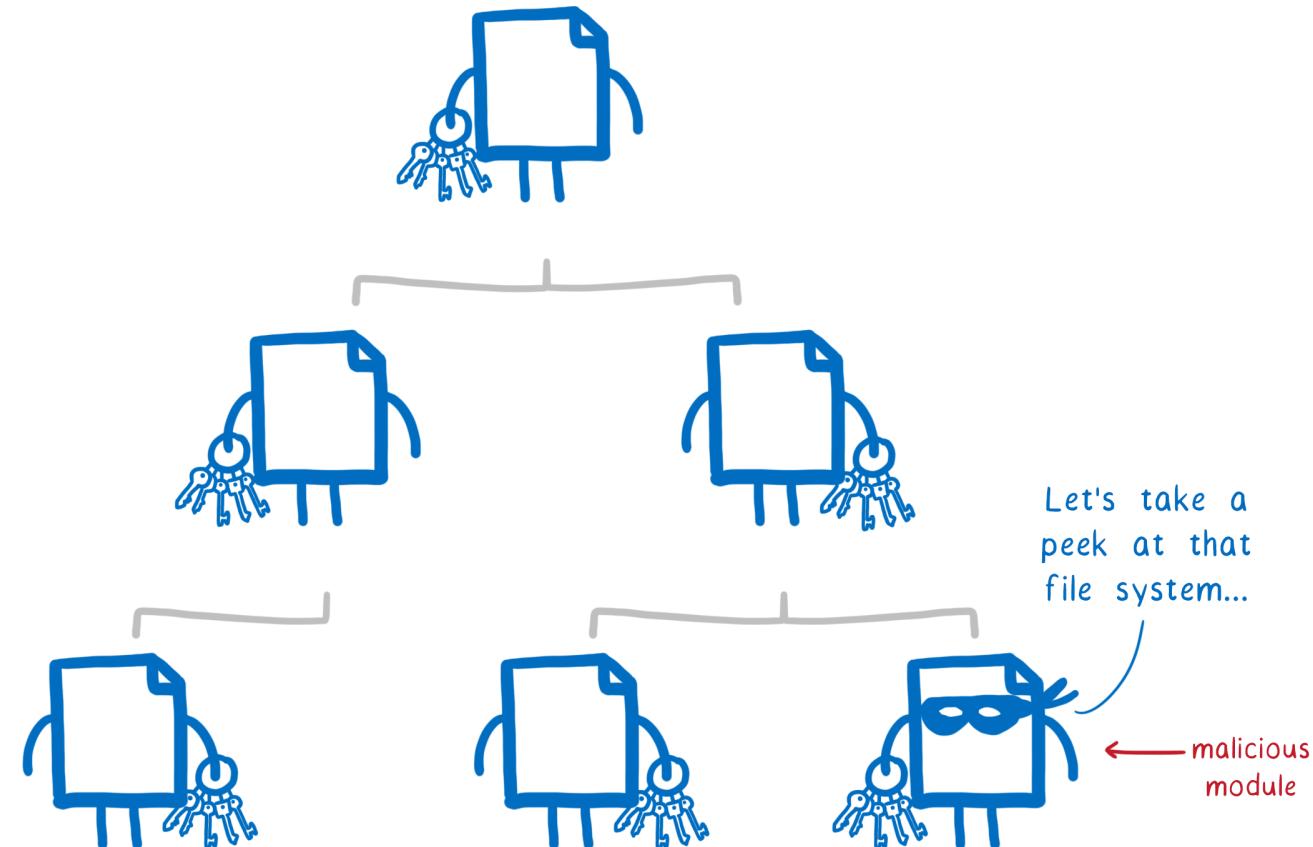
composition of an
average code base



Dependencies

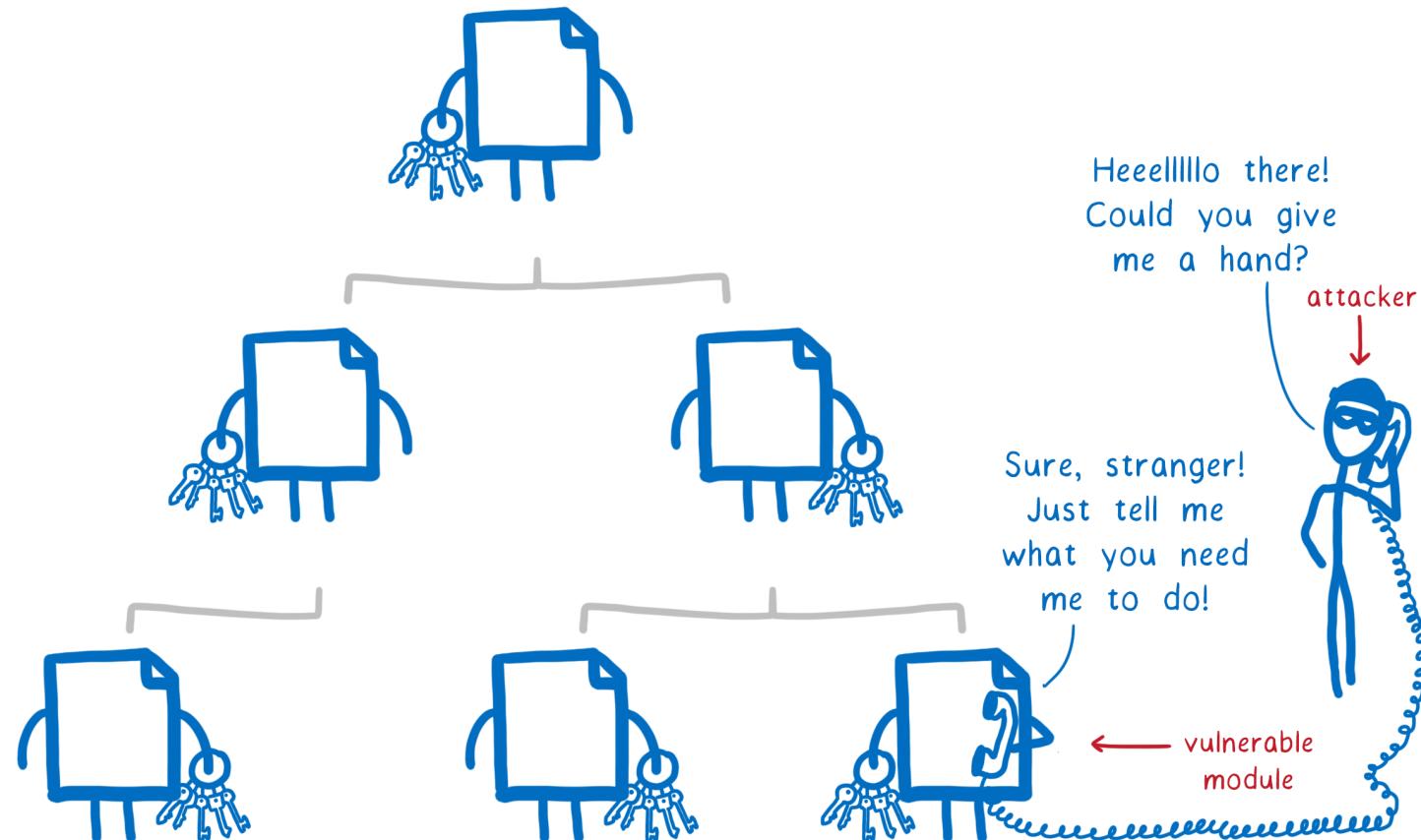


Malicious module



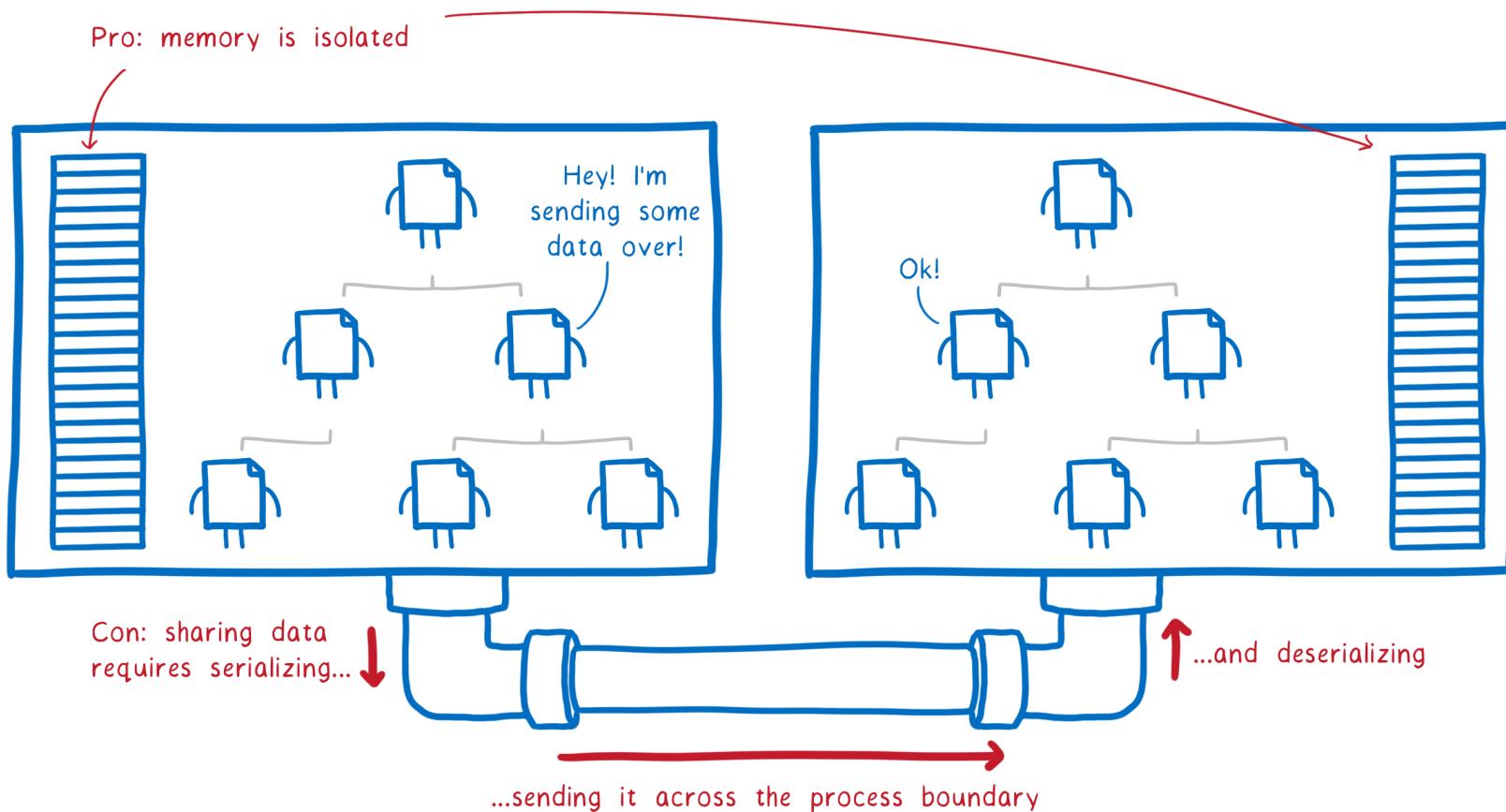
@nielstanis@infosec.exchange

Vulnerable module

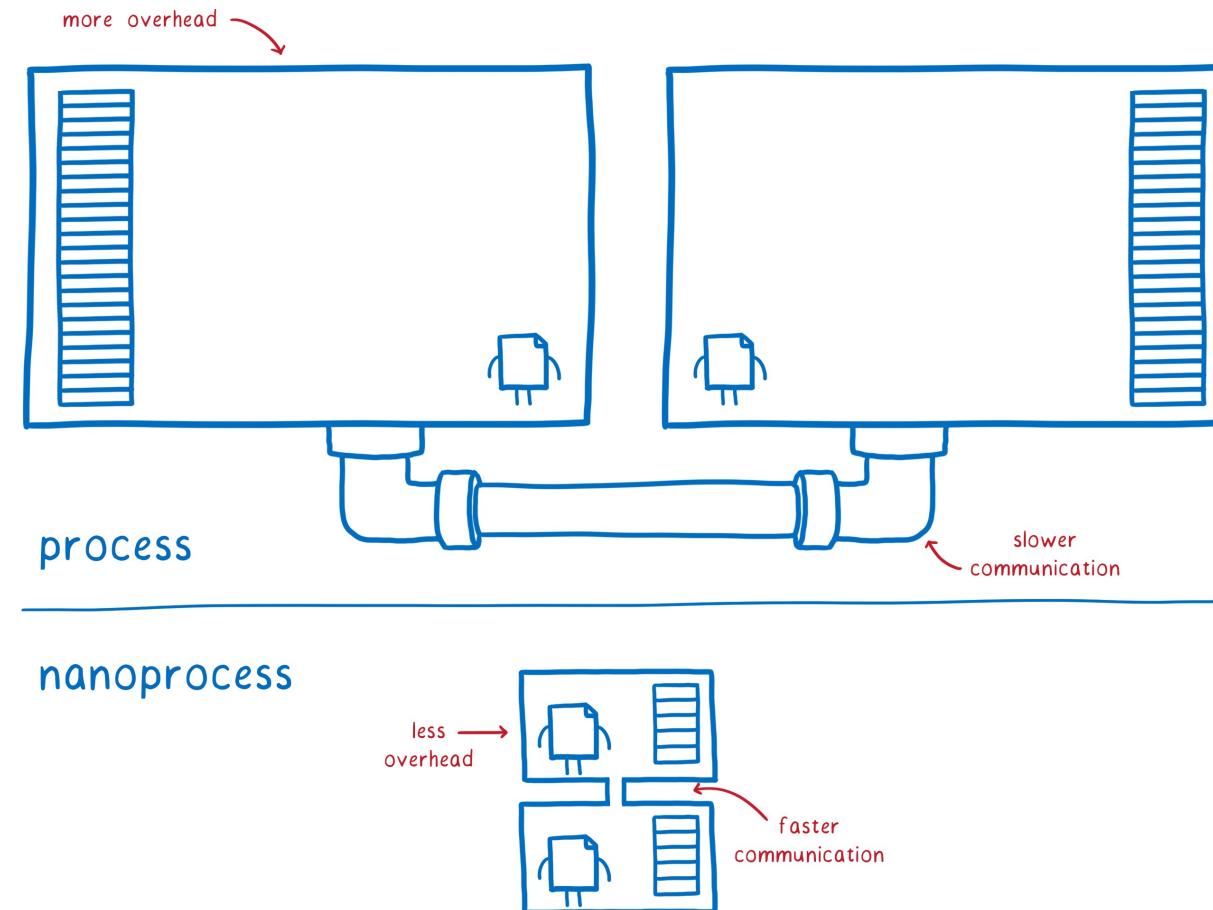


@nielstanis@infosec.exchange

Process Isolation



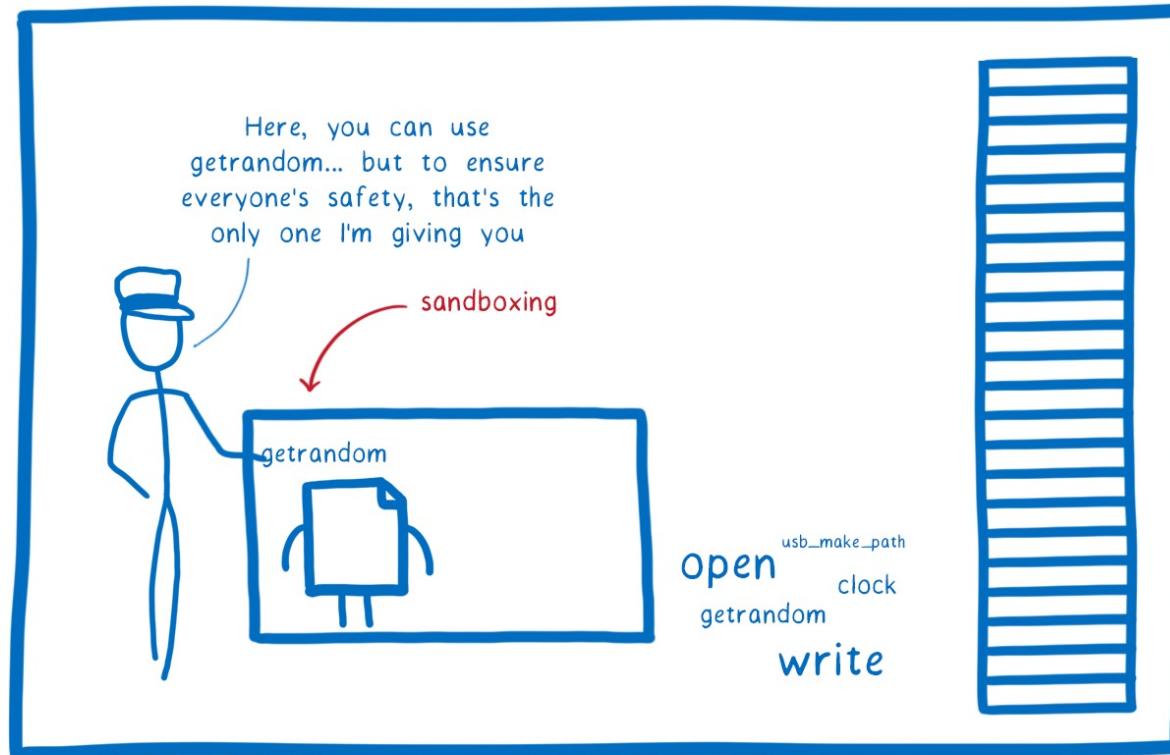
WebAssembly Nano-Process



* not drawn to scale

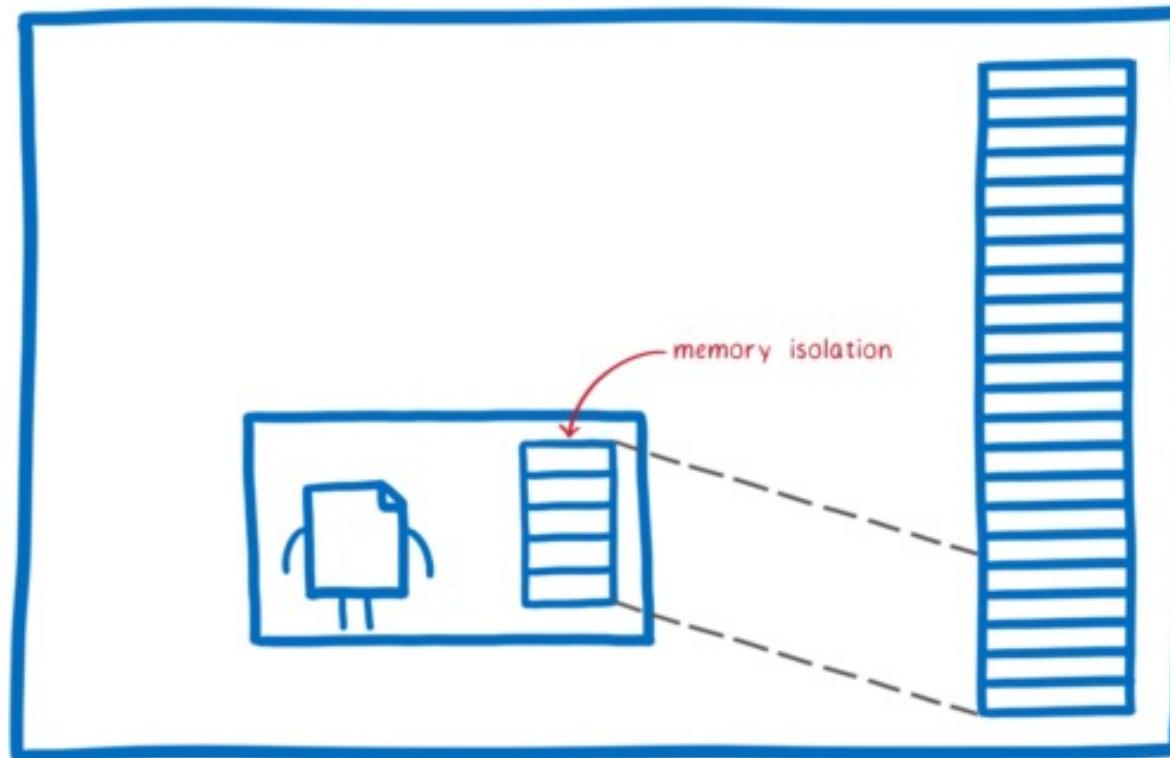
WebAssembly Nano-Process

1. Sandboxing



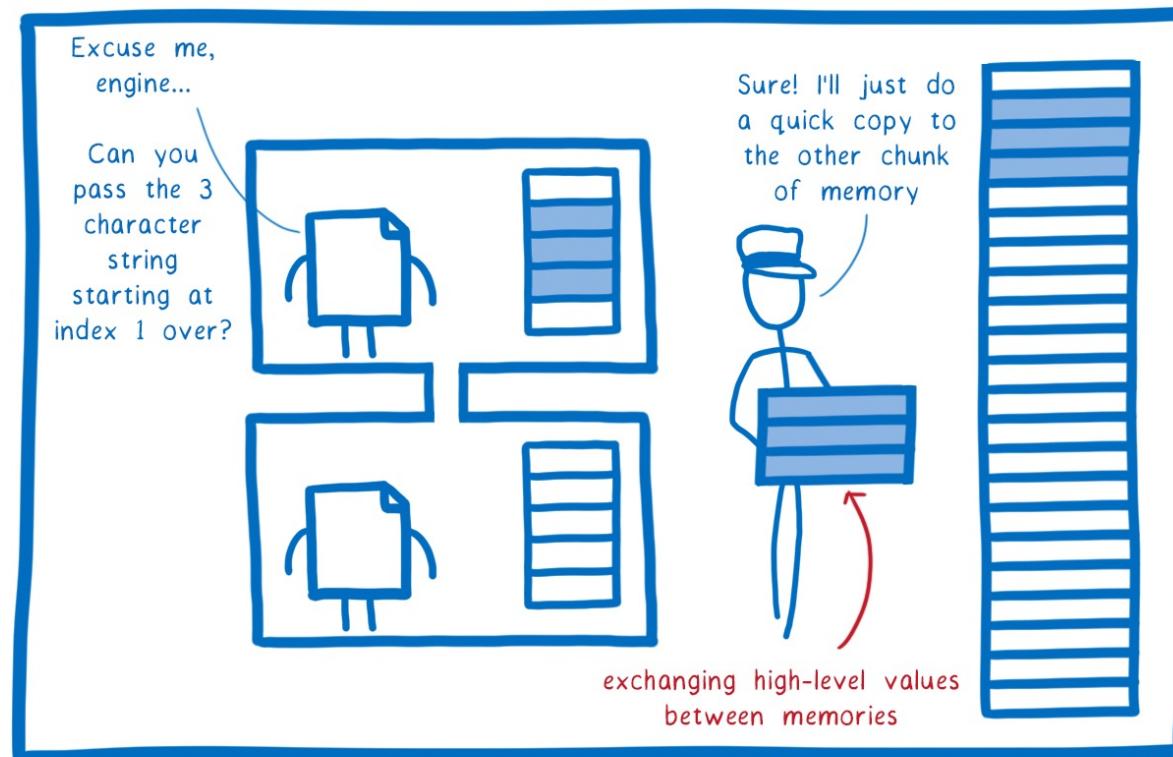
WebAssembly Nano-Process

2. Memory model



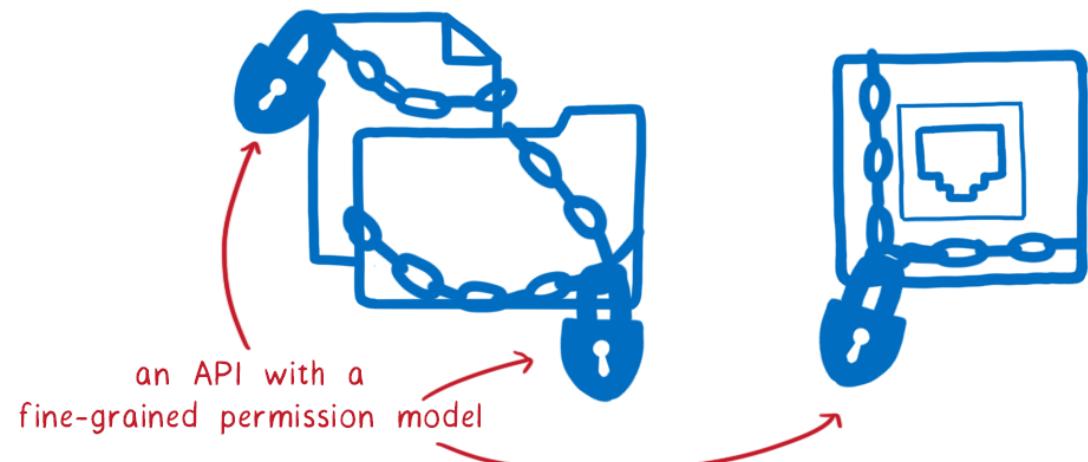
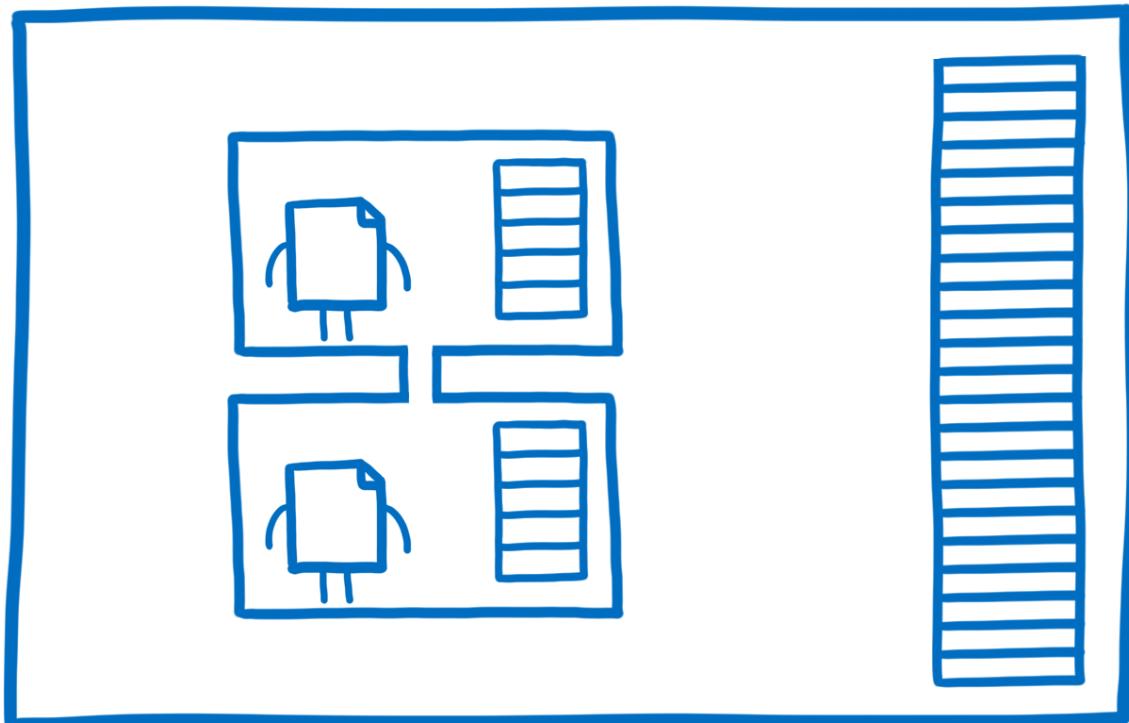
WebAssembly Nano-Process

3. Interface Types



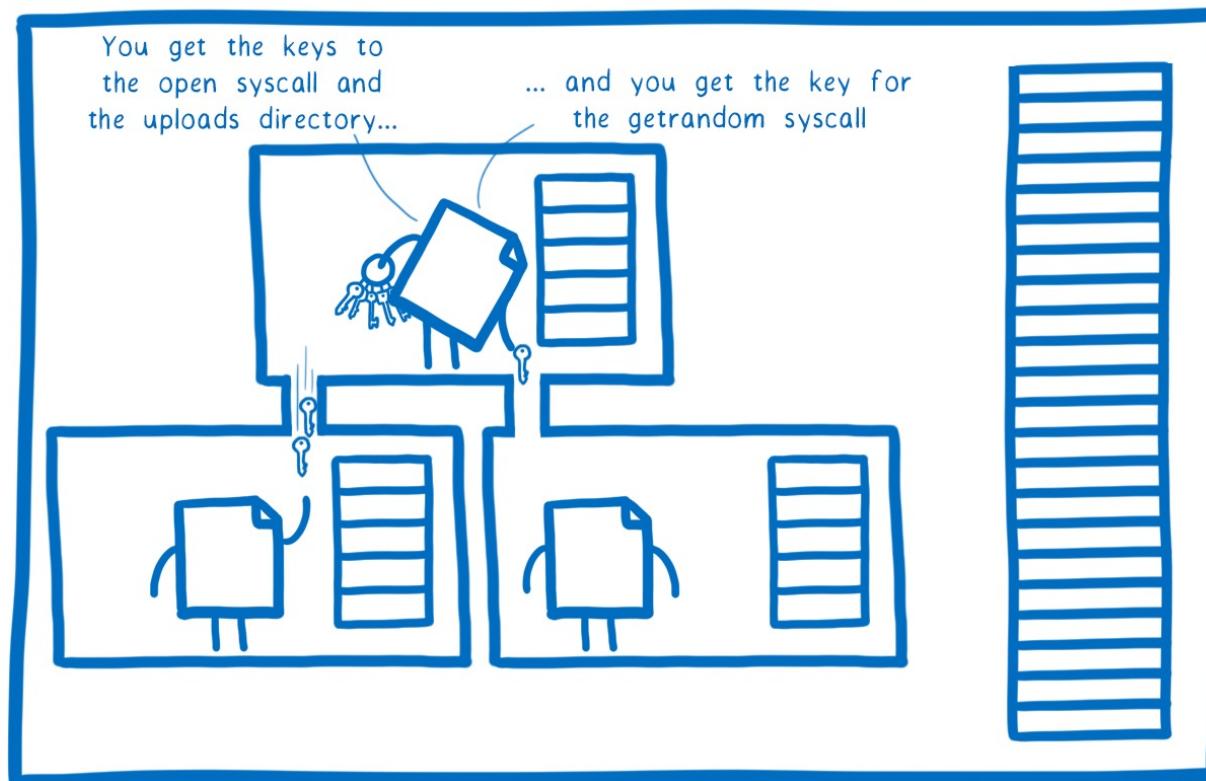
WebAssembly Nano-Process

4. WebAssembly System Interface



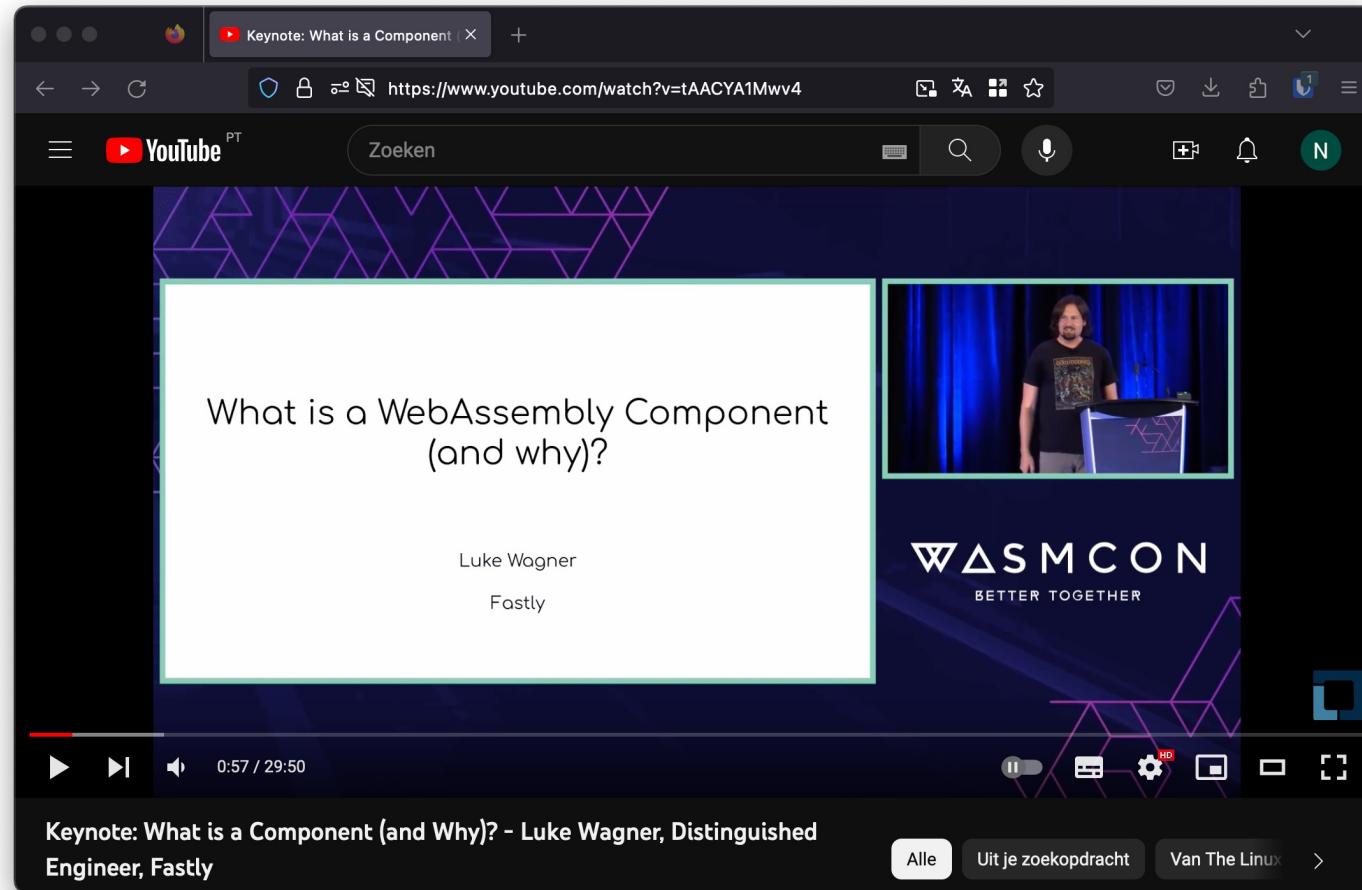
WebAssembly Nano-Process

5. The missing link



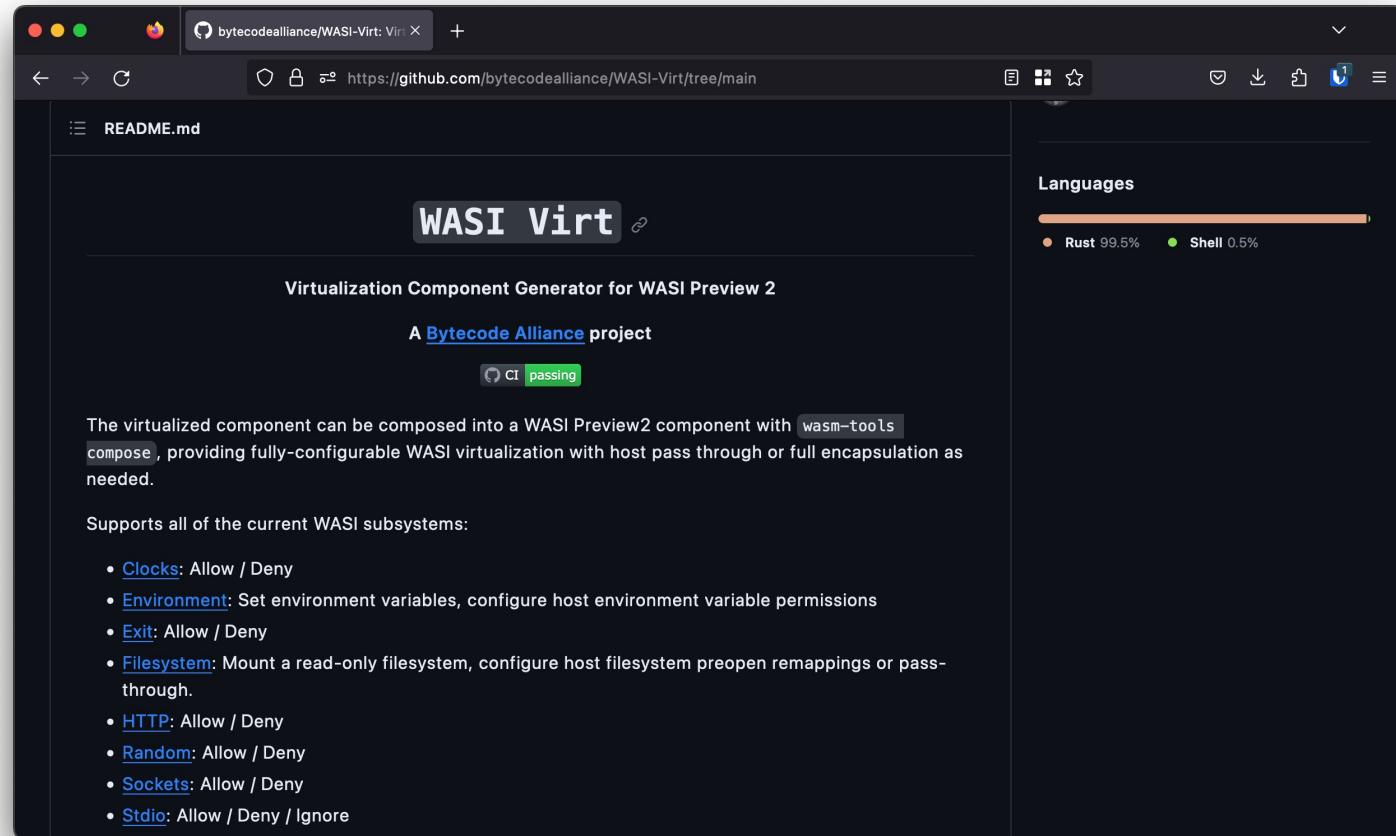
@nielstanis@infosec.exchange

WebAssembly Component Model

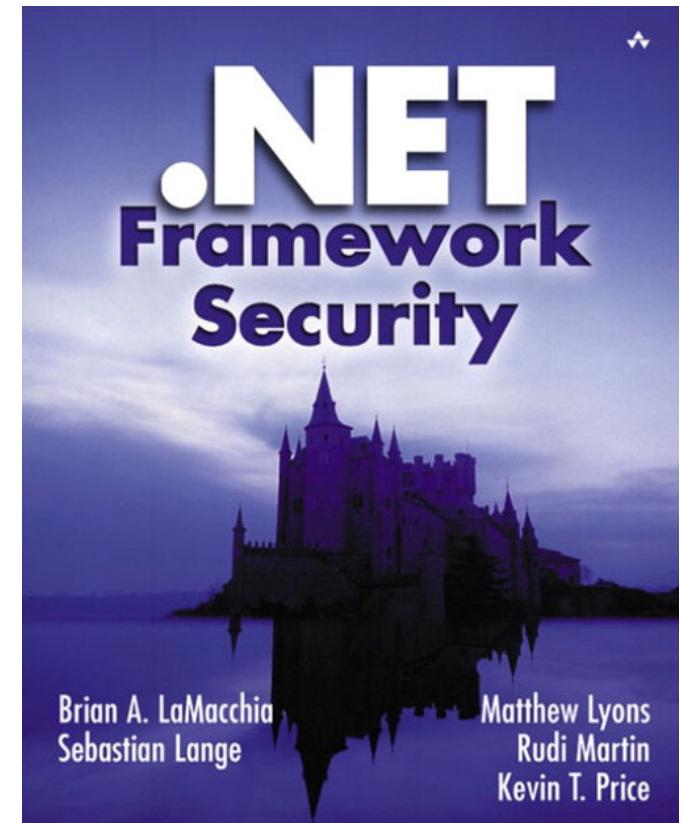
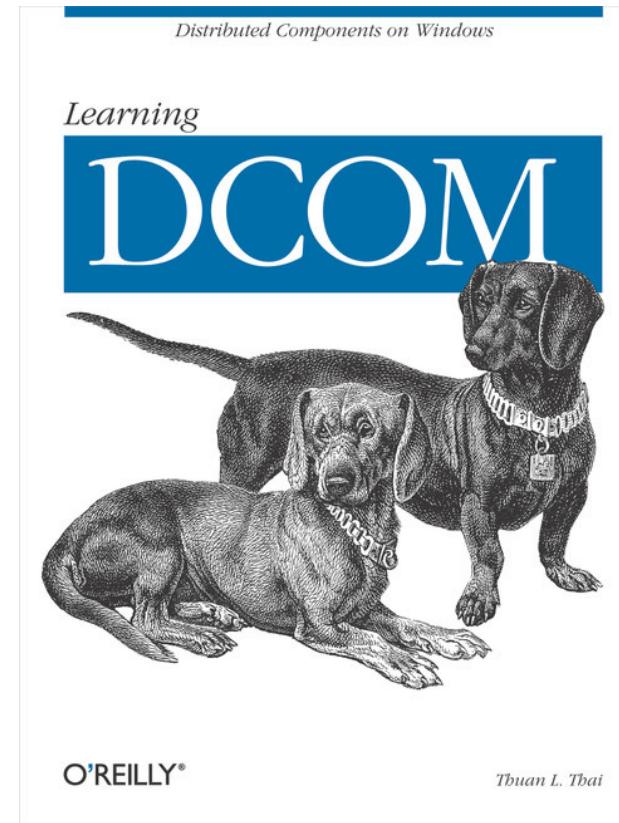
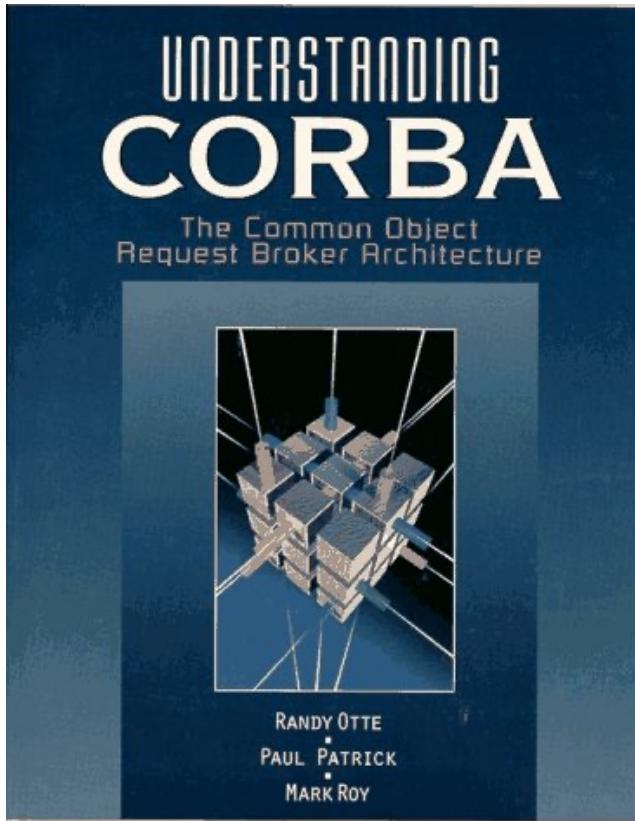


@nielstanis@infosec.exchange

WebAssembly Component Model



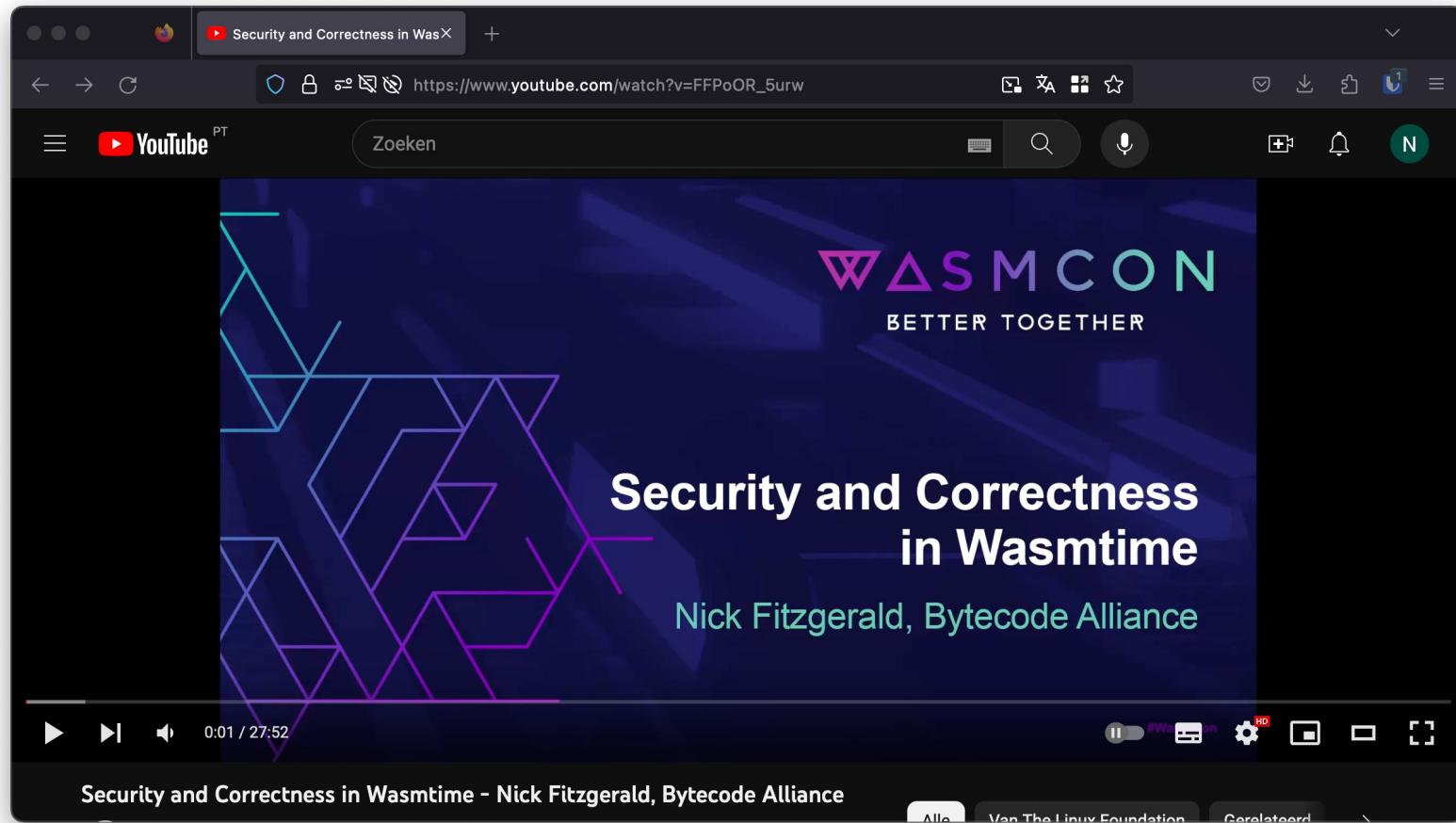
Have we seen this before?



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/appsecdc2023>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Thank you!



OWASP 2023
GLOBAL
AppSec

WASHINGTON
DC
OCT 30 - NOV 3

THANK YOU