



# Using WebAssembly to run, extend, and secure your .NET application

# Niels Tanis

0101  
0101

# 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

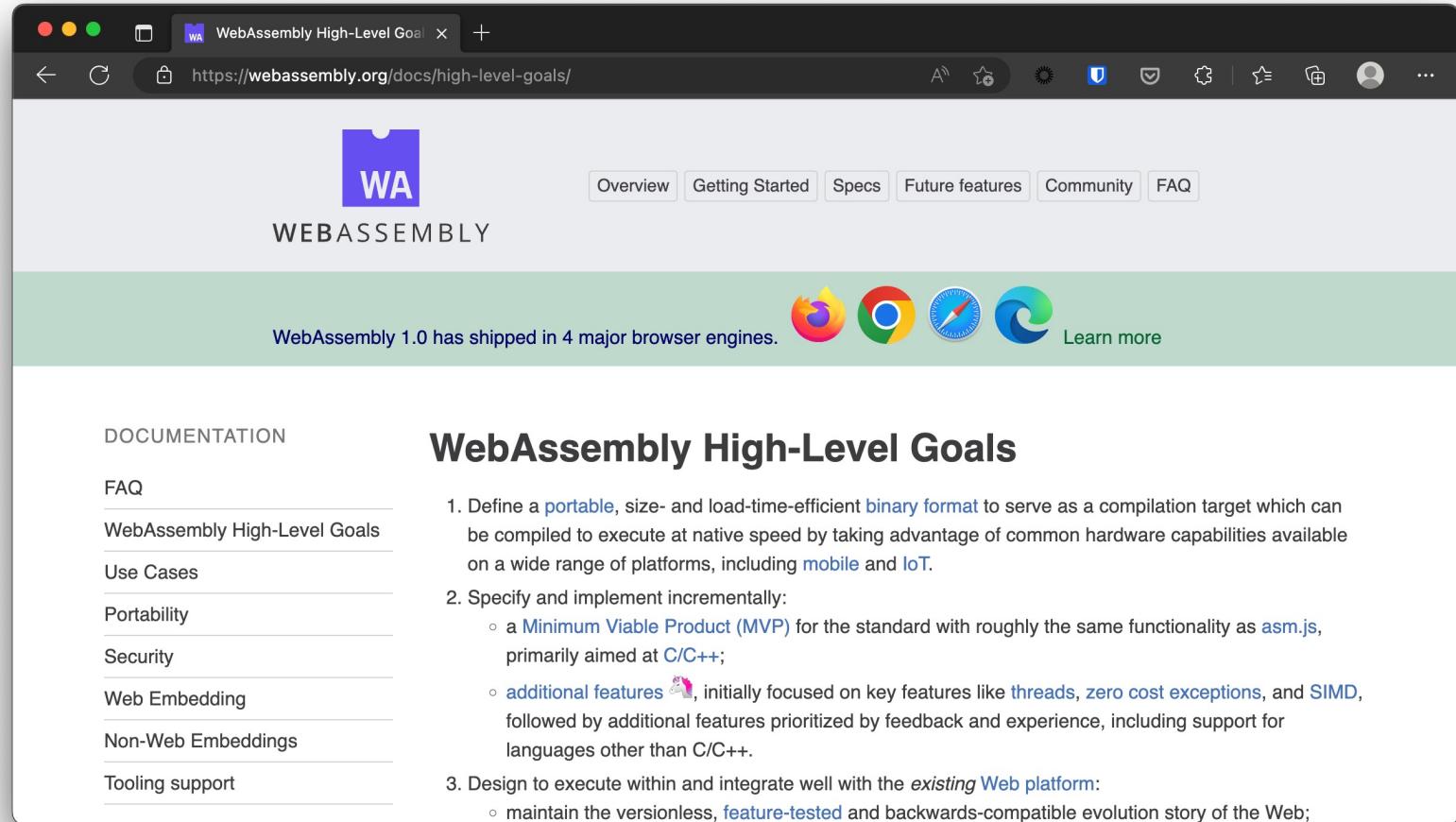


SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly



The screenshot shows a web browser window displaying the "WebAssembly High-Level Goals" page from the official website at <https://webassembly.org/docs/high-level-goals/>. The page features a purple header with the "WA" logo and the word "WEBASSEMBLY". 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:

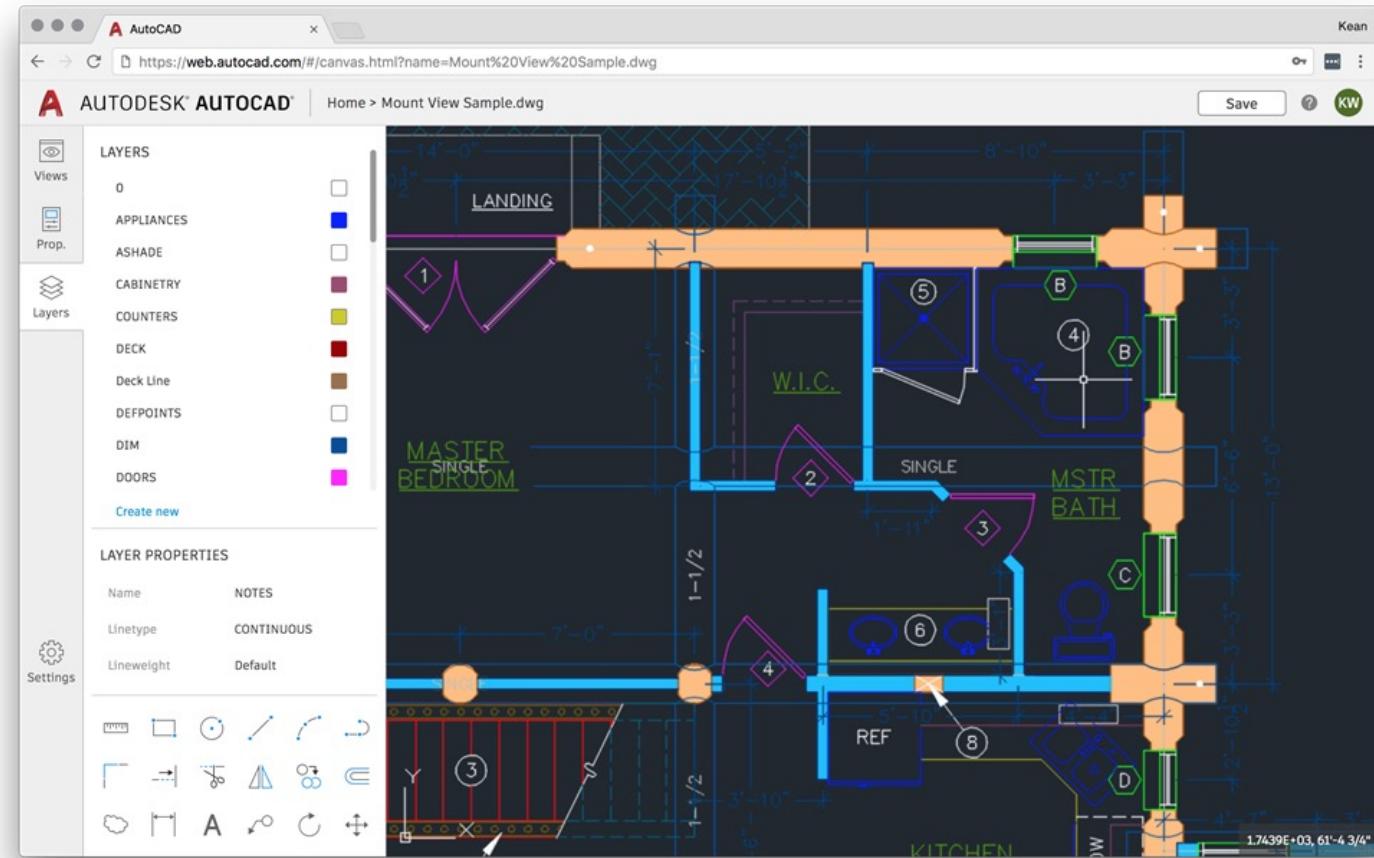
1. Define a portable, size- and load-time-efficient binary format to serve as a compilation target which can be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT.
2. Specify and implement incrementally:
  - 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;

SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly - AutoCAD



SWETUGG

 @nielstanis@infosec.exchange

0101  
0101

# WebAssembly - SDK's

A screenshot of a Medium article page. The title is "Introducing the Disney+ Application Development Kit (ADK)" by Mike Hanley. The article was published on Sep 8, 2021, and has a 10 min read time. It features a profile picture of Mike Hanley, social sharing icons (Twitter, Facebook, LinkedIn), and a "Get started" button. 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" by Alexandru Ene. The article is categorized under "CLOUD AND SYSTEMS". It discusses the switch to WebAssembly and its benefits. The content mentions Prime Video's delivery to millions of customers on various devices.

SWETUGG

@nielstanis@infosec.exchange



# Agenda

- Introduction
- WebAssembly 101
- Running .NET on WebAssembly
- Extending .NET with WebAssembly
- Securing .NET with WebAssembly
- 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

SWETUGG

@nielstanis@infosec.exchange



# 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

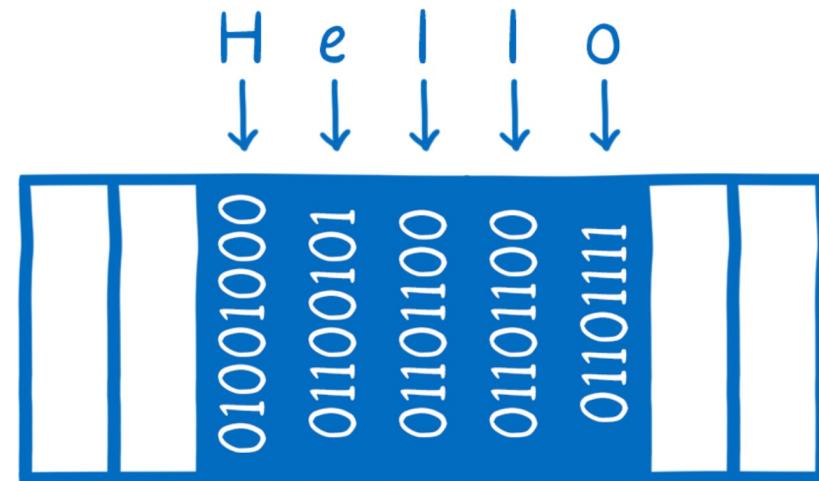


WEBASSEMBLY

0101  
0101

# 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!");
```

0101  
0101

# WebAssembly Control-Flow Integrity

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

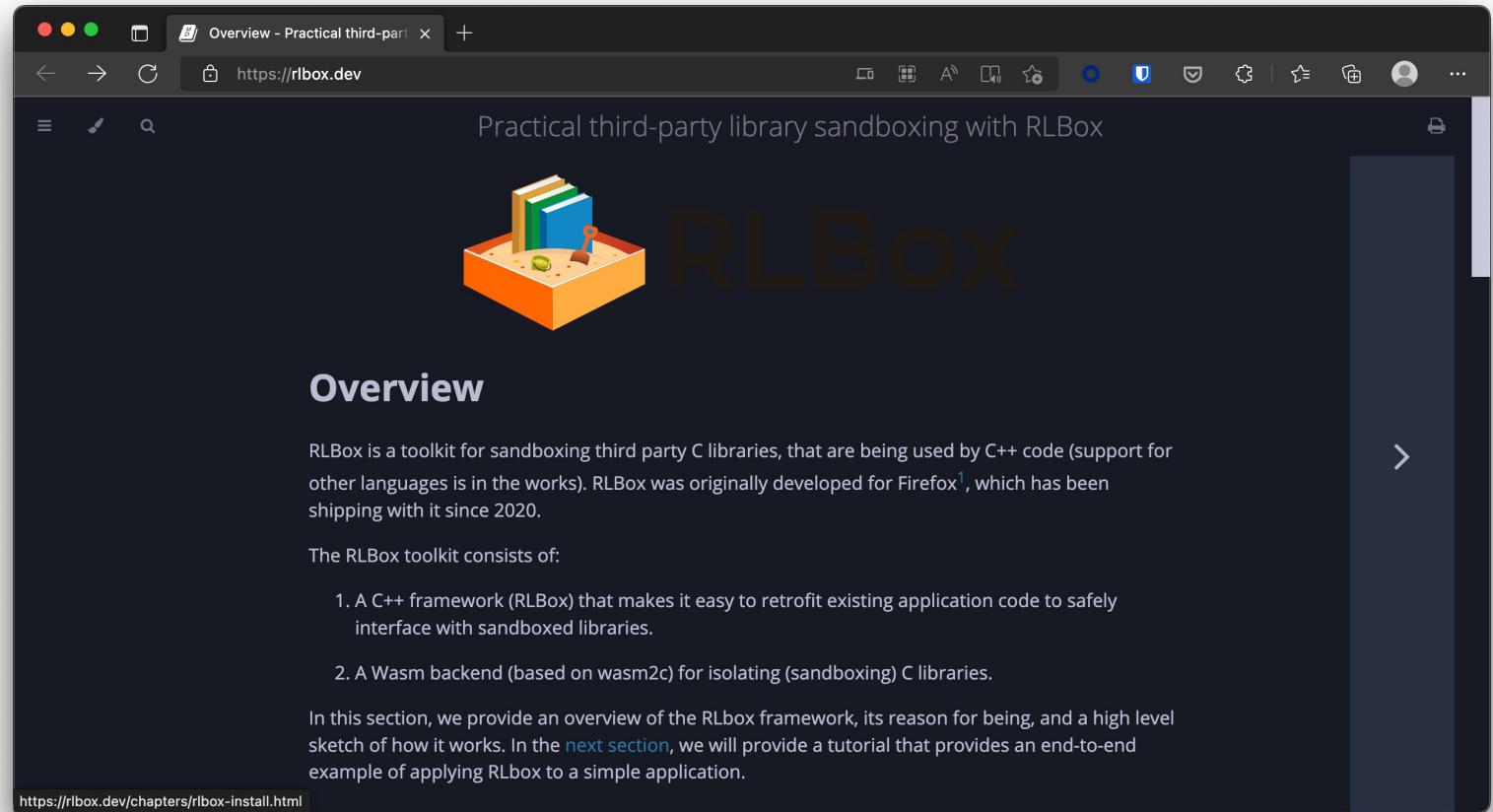
```
Console.WriteLine("Number is larger than 5");
```

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

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

0101  
0101

# FireFox RLBox



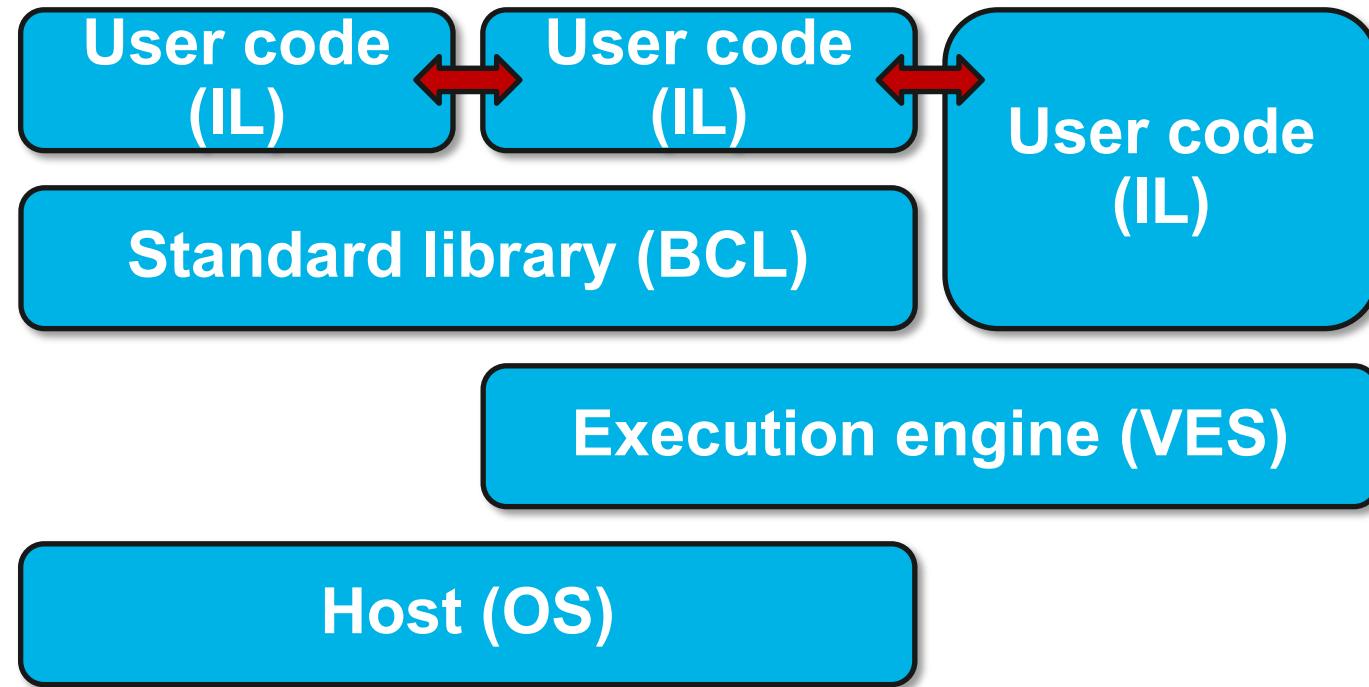
The screenshot shows a Firefox browser window with a dark theme. The title bar reads "Overview - Practical third-part" and the address bar shows "https://rlbox.dev". The main content area has a dark background with a central logo featuring three books in a sandcastle-like setup with a shovel. To the right of the logo, the word "RLBox" is written 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 of the page points to "rlbox.dev/chapters/rlbox-install.html".

SWETUGG

@nielstanis@infosec.exchange

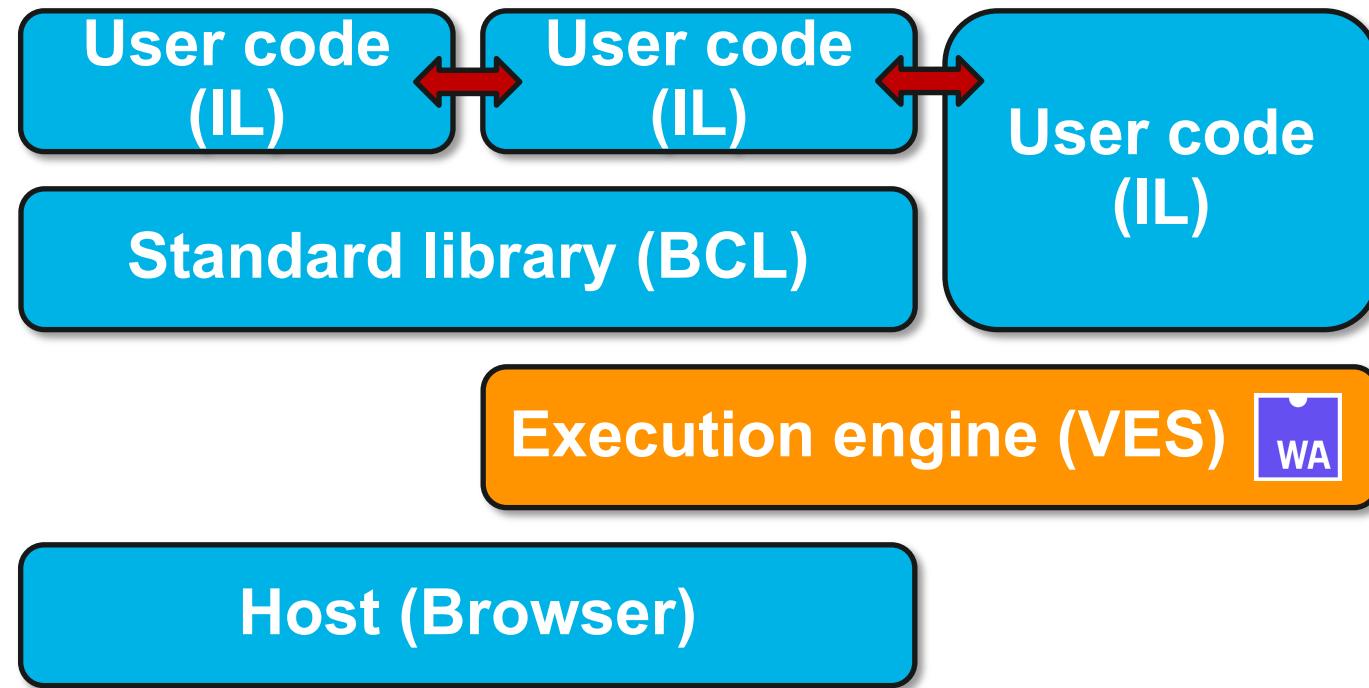


# Running .NET on WebAssembly



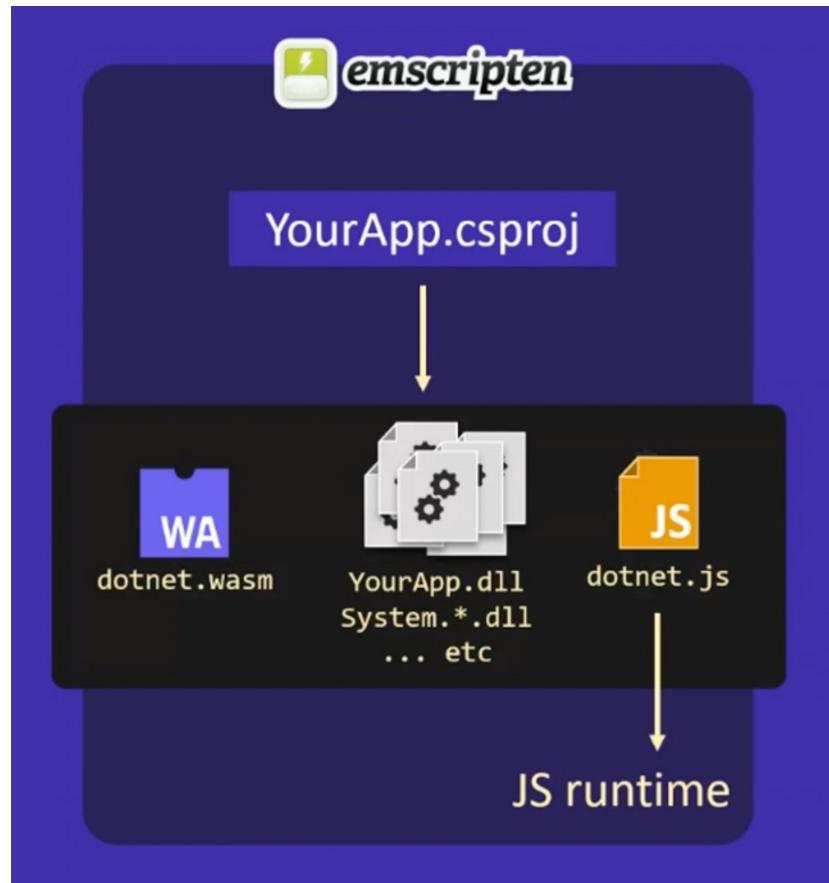


# Running .NET on WebAssembly



0101  
0101

# Blazor WebAssembly

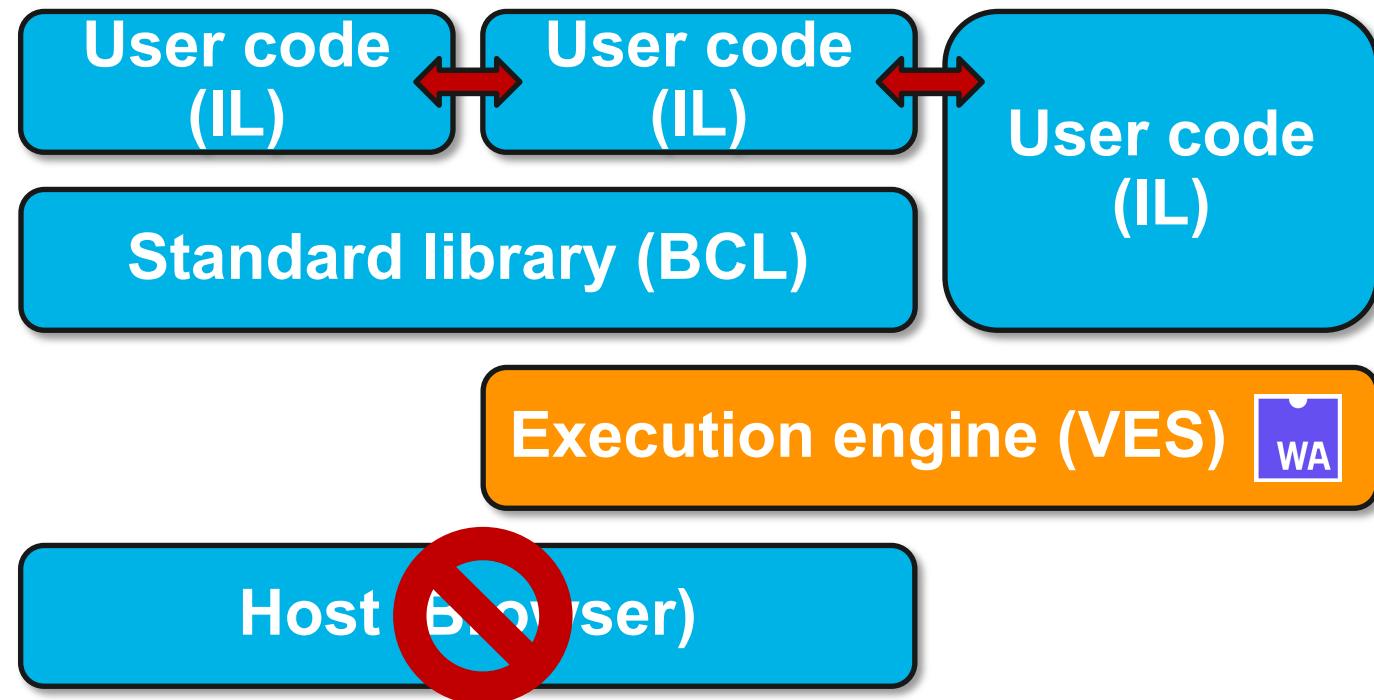


SWETUGG

@nielstanis@infosec.exchange



# 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

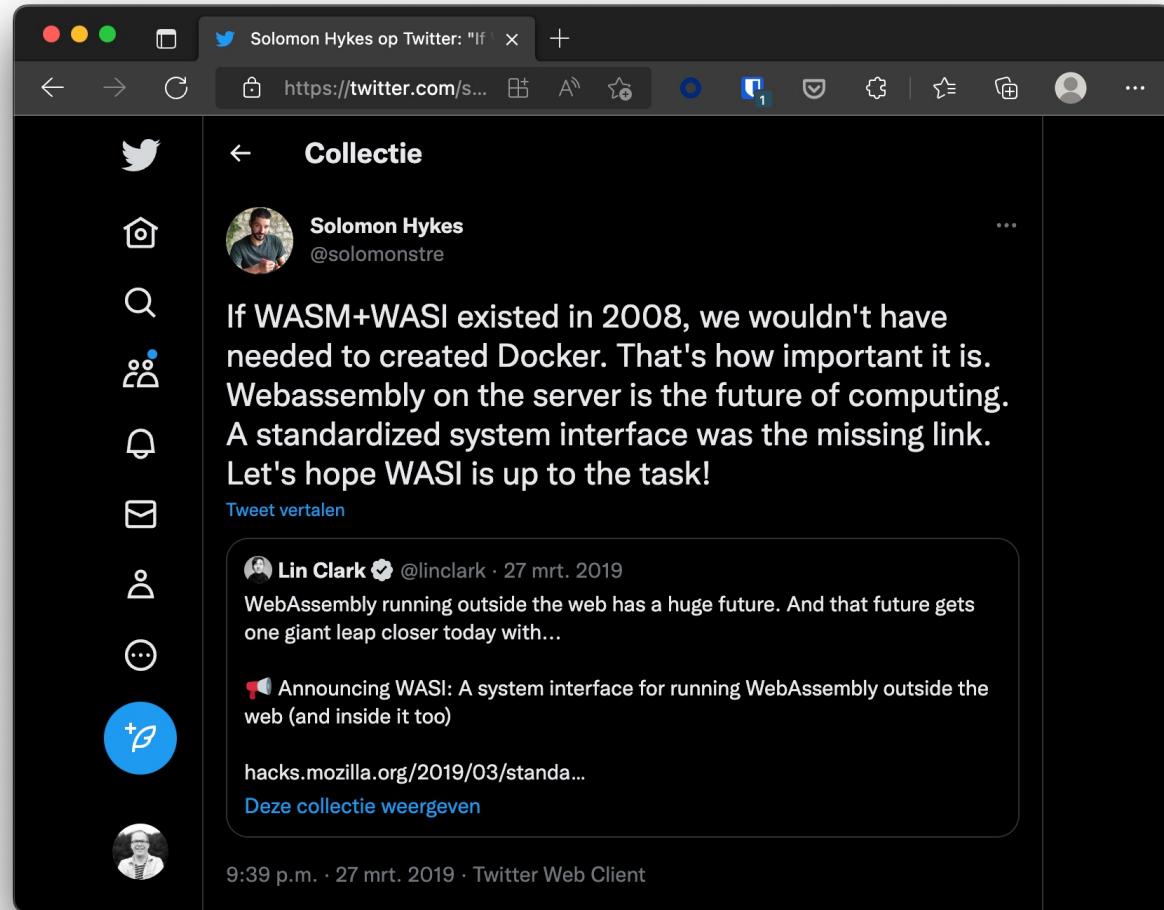
# WebAssembly System Interface WASI



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

0101  
0101

# Docker vs WASM & WASI



SWETUGG

 @nielstanis@infosec.exchange

0101  
0101

# Docker vs WASM & WASI

Solomon Hykes op Twitter: <https://twitter.com/solomonstre/status/11091000000000000>

**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/11091000000000000](https://twitter.com/linclark/status/11091000000000000)

Deze collectie weergeven

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

56 Retweets 5 Geciteerde Tweets 165 Vind-ik-leuks

SWETUGG

 @nielstanis@infosec.exchange

# Docker & WASM

0101  
0101

The screenshot shows a web browser window with the title "Introducing the Docker+Wasm Technical Preview". The page features a purple header bar with the text "Wasm is a fast, light alternative to Linux containers — try it out today in the Docker+Wasm Technical Preview". Below this is the Docker+Wasm logo. The main content area has a dark blue background with the text "Introducing the Docker+Wasm Technical Preview" in large white font. At the bottom left is a circular profile picture of Michael Irwin, followed by his name "MICHAEL IRWIN" and the date "Oct 24 2022". A small 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 Engine architecture for Docker+Wasm. The diagram shows the "Docker Engine" at the top, which interacts with a "containerd" component. The "containerd" component oversees three separate container instances. Each instance consists of a "containerd-shim" layer, a "runc" layer, and a "Container process". The third instance is specifically labeled with "wasmedge" and a "Wasm Module". Arrows indicate the flow of communication between these layers.

**Let's look at an example!**

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

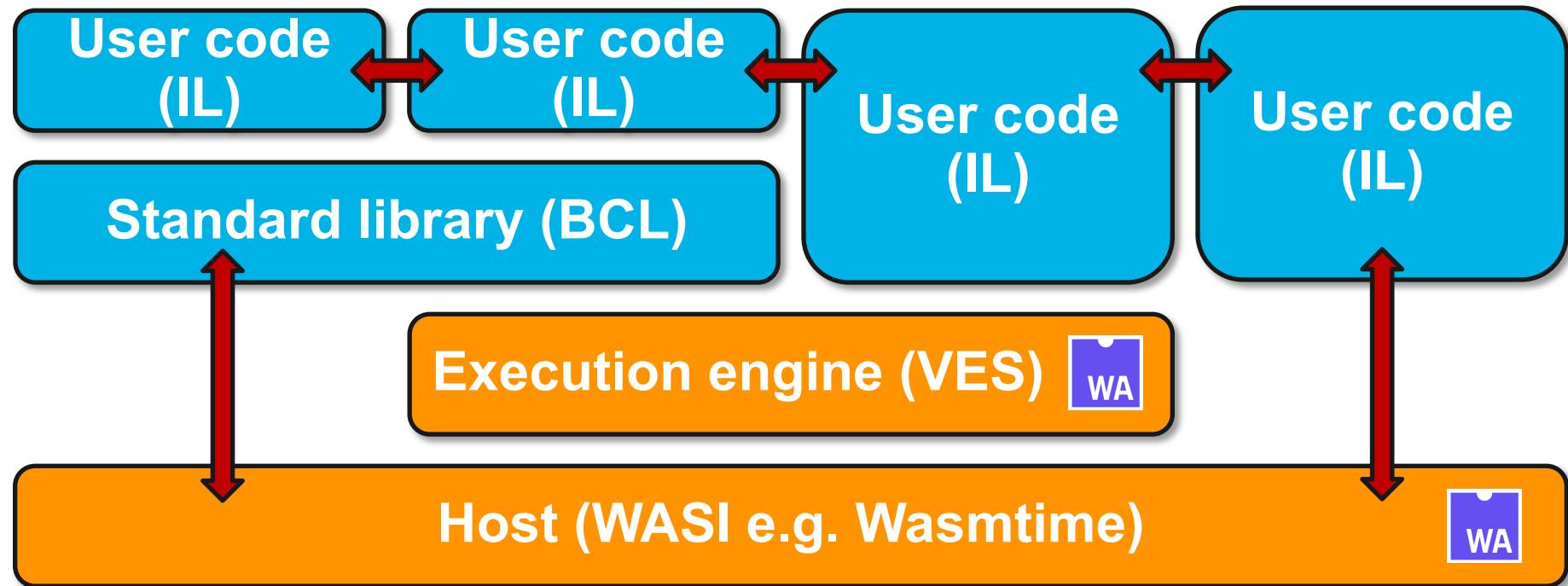
```
docker run -dp 8080:8080 --name=wasm-example --runtime=io.containerd.wasmedge.v1 --platform=wasi/wasm32 michaelirwin244/wasm-example
```

SWETUGG

 @nielstanis@infosec.exchange



# WebAssembly System Interface WASI



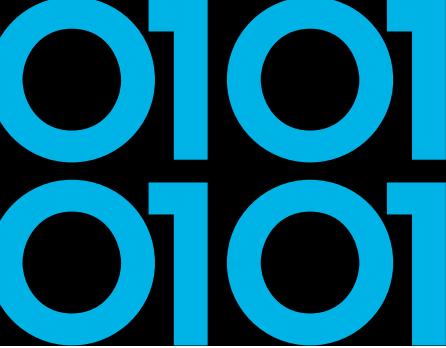


# Experimental WASI SDK for .NET



SWETUGG

@nielstanis@infosec.exchange



# .NET 8 WASI-Experimental

The screenshot shows a web browser window with the URL <https://devblogs.microsoft.com/dotnet/extending-web-assembly-to-the-cloud/>. The page title is "Extending WebAssembly to the Cloud". The main content is titled "wasi-experimental workload". It explains that .NET 8 includes a new workload called "wasi-experimental" which builds on Wasm functionality used by Blazor, extending it to run in "wasmtime" and invoke WASI interfaces. It notes that while not fully developed, it already provides useful functionality.

Let's move on from theory to demonstrating the new capabilities.

After installing the [.NET 8 SDK](#), you can install the "wasi-experimental" workload.

```
dotnet workload install wasi-experimental
```

Note: This command may require admin permissions, for example with `sudo` on Linux and macOS.

You also need to install `wasmtime` to run the Wasm code you are soon going to produce.

Try a simple example with the "wasi-console" template.

```
$ dotnet new wasiconsole -o wasiconsole  
$ cd wasiconsole  
$ cat Program.cs  
using System;  
  
Console.WriteLine("Hello, WASI Console!");
```

SWETUGG

@nielstanis@infosec.exchange

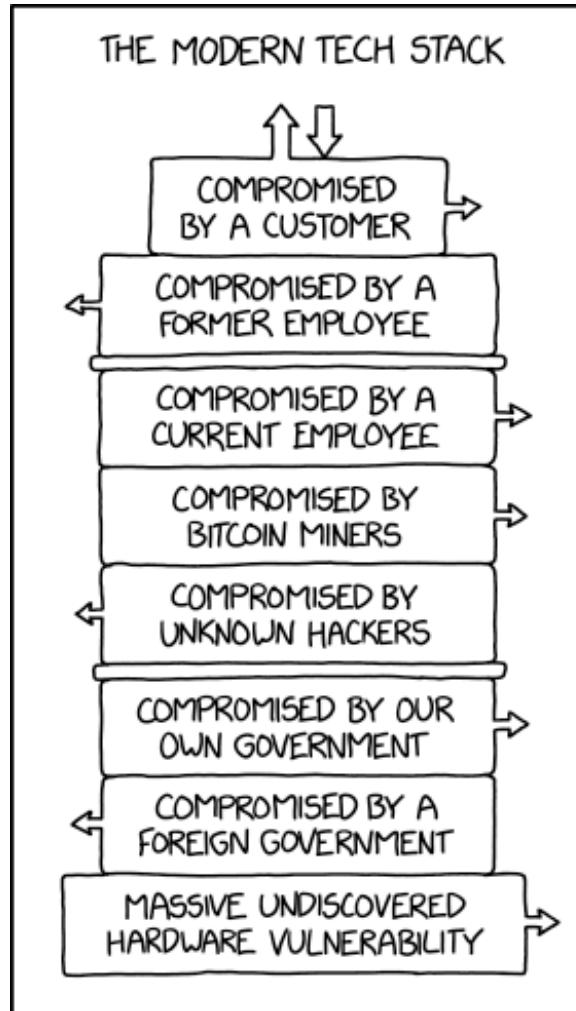


# Extending .NET with WASM

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

0101  
0101

# Trusted Computing - XKCD 2166

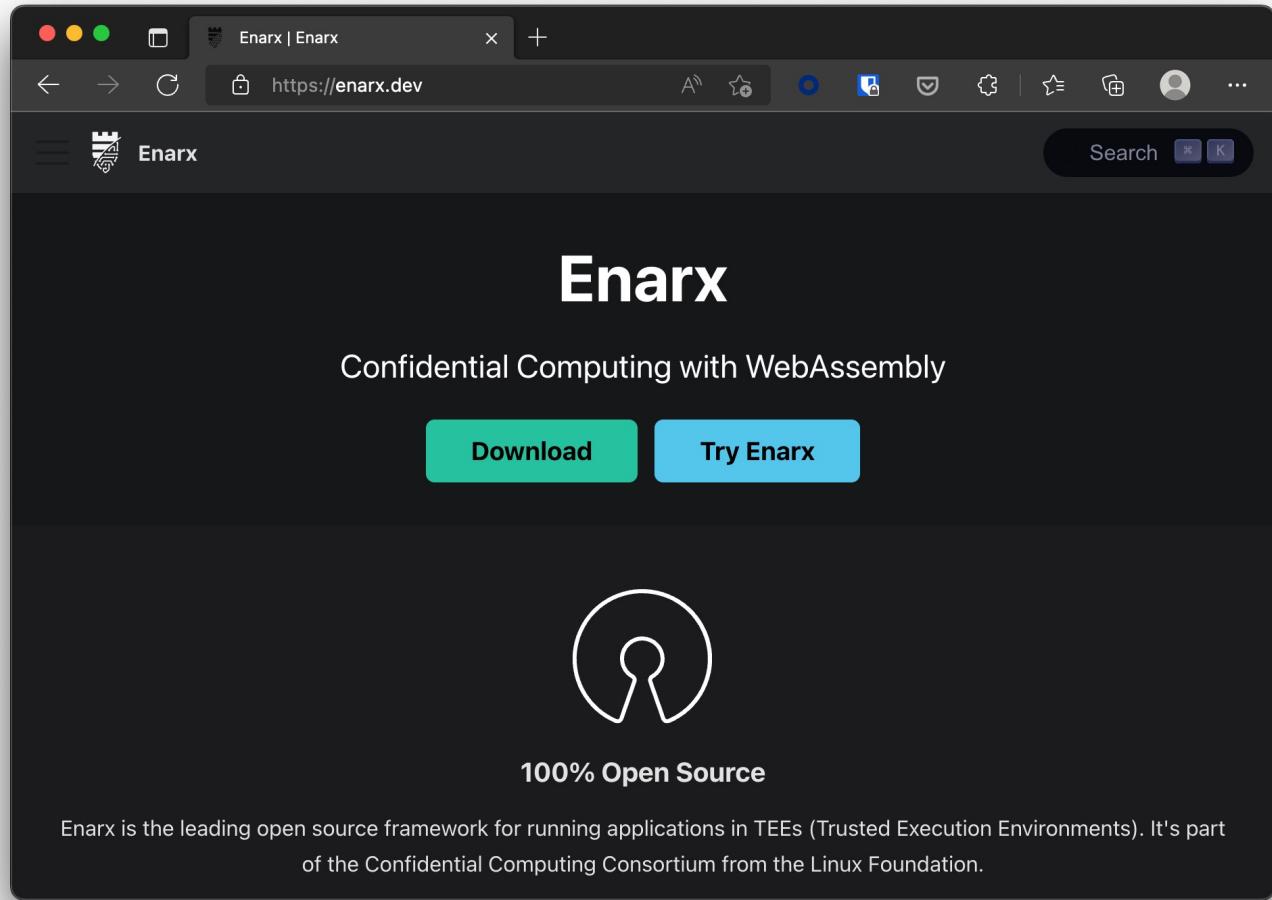


SWETUGG

@nielstanis@infosec.exchange

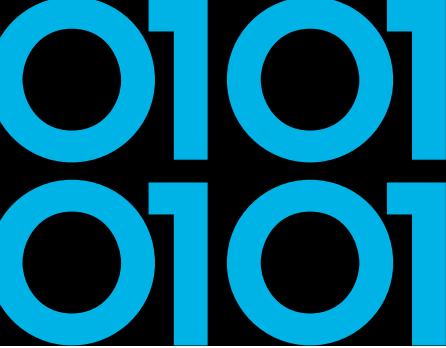
0101  
0101

# Enarx



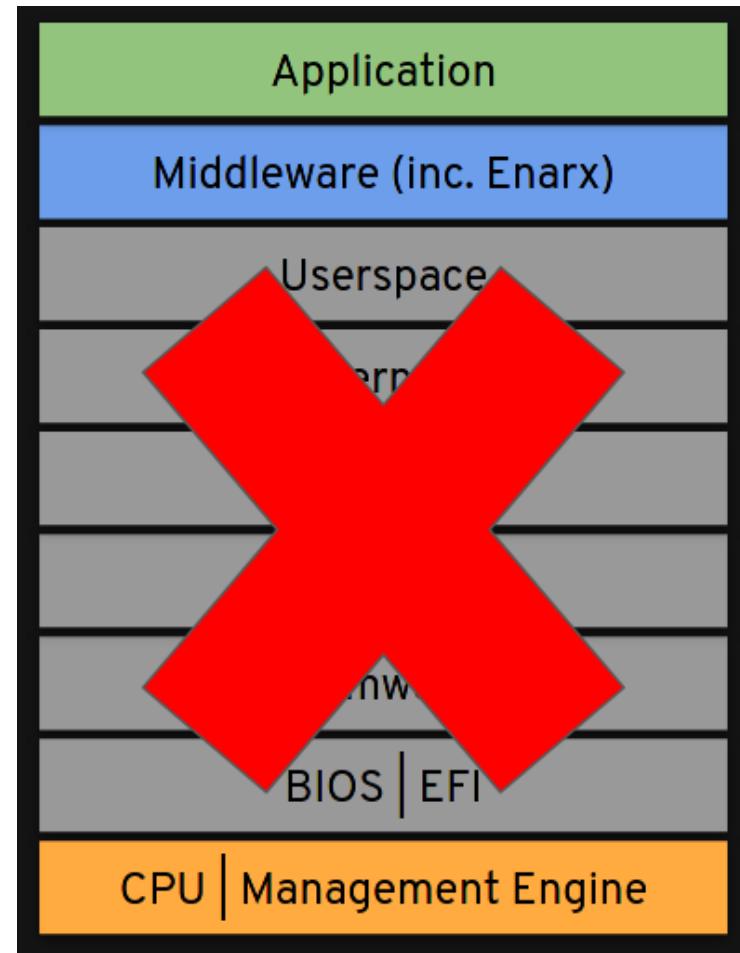
SWETUGG

@nielstanis@infosec.exchange



# 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

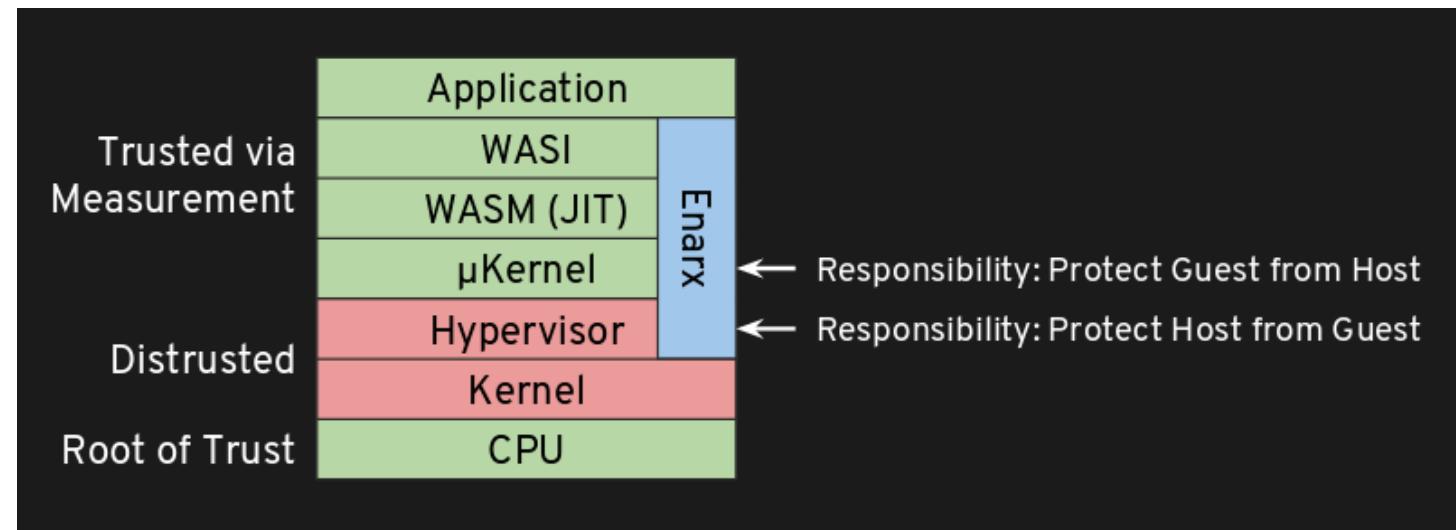


SWETUGG



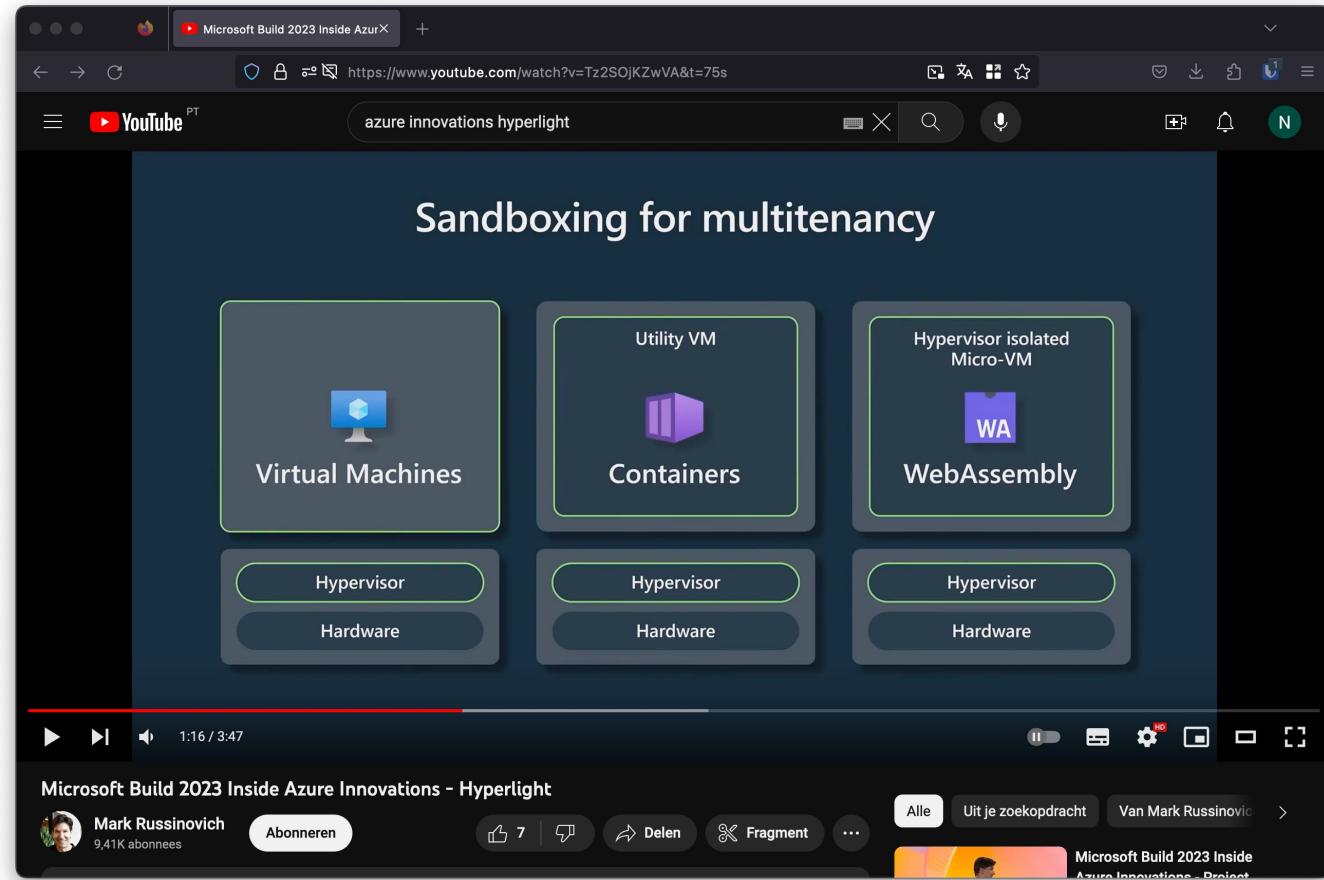
# Enarx

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



0101  
0101

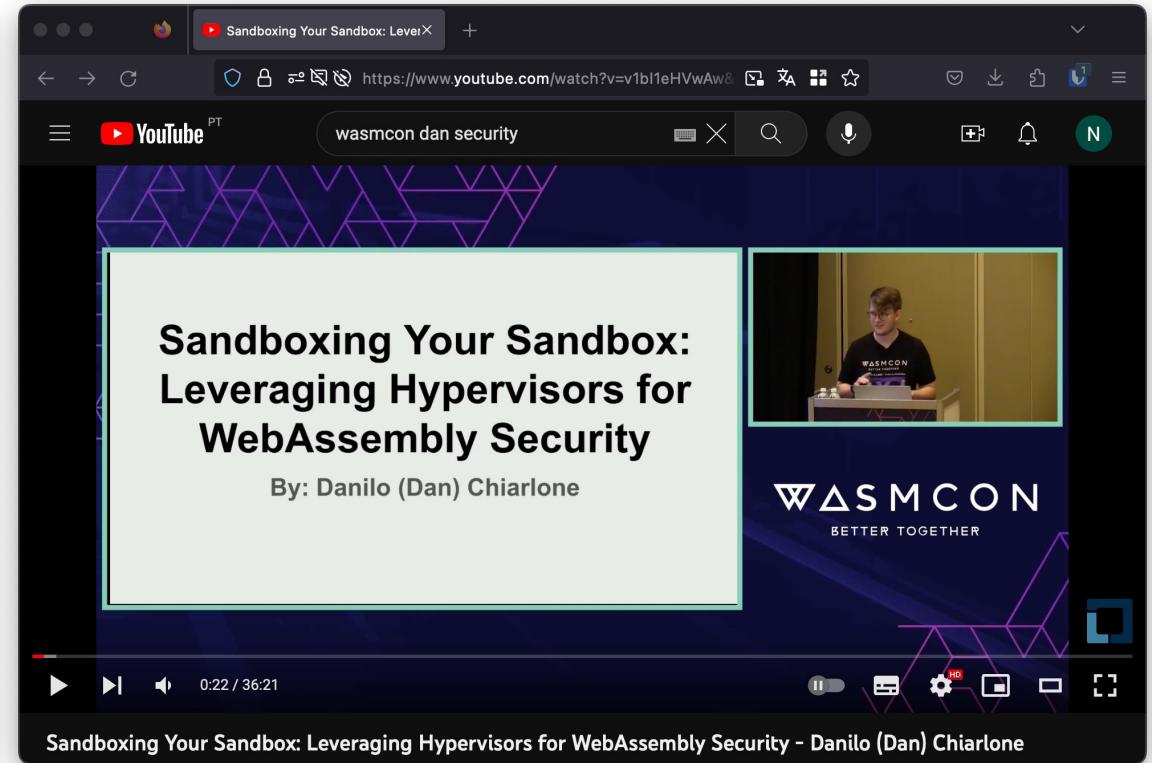
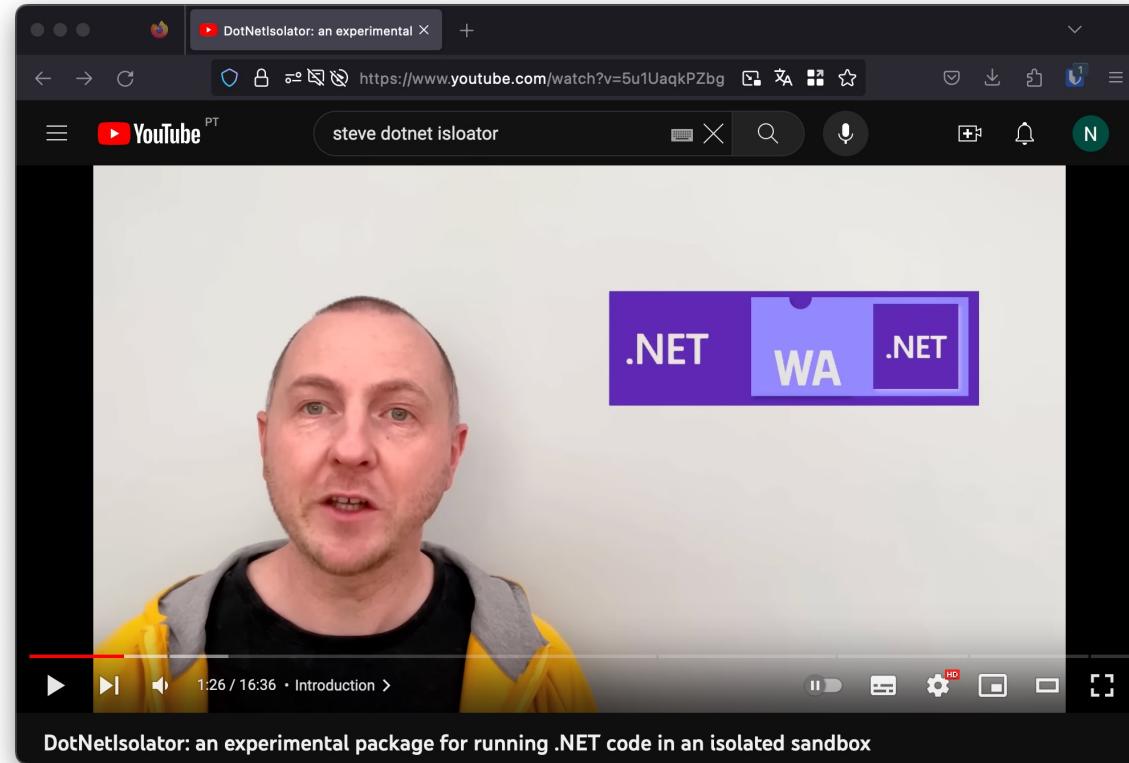
# Project Hyperlight



SWETUGG

@nielstanis@infosec.exchange

# DotNetIsolator & Project Hyperlight



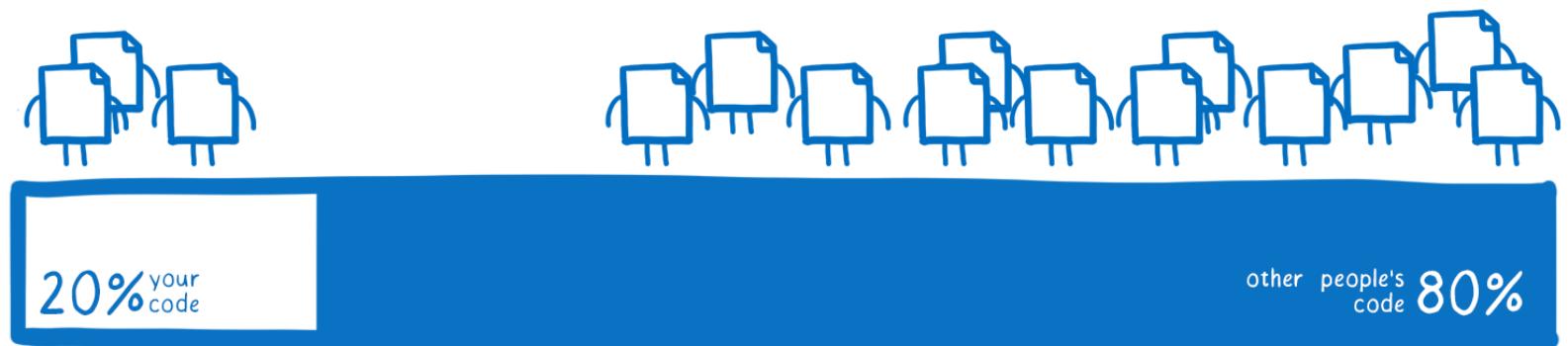
SWETUGG

@nielstanis@infosec.exchange

0101  
0101

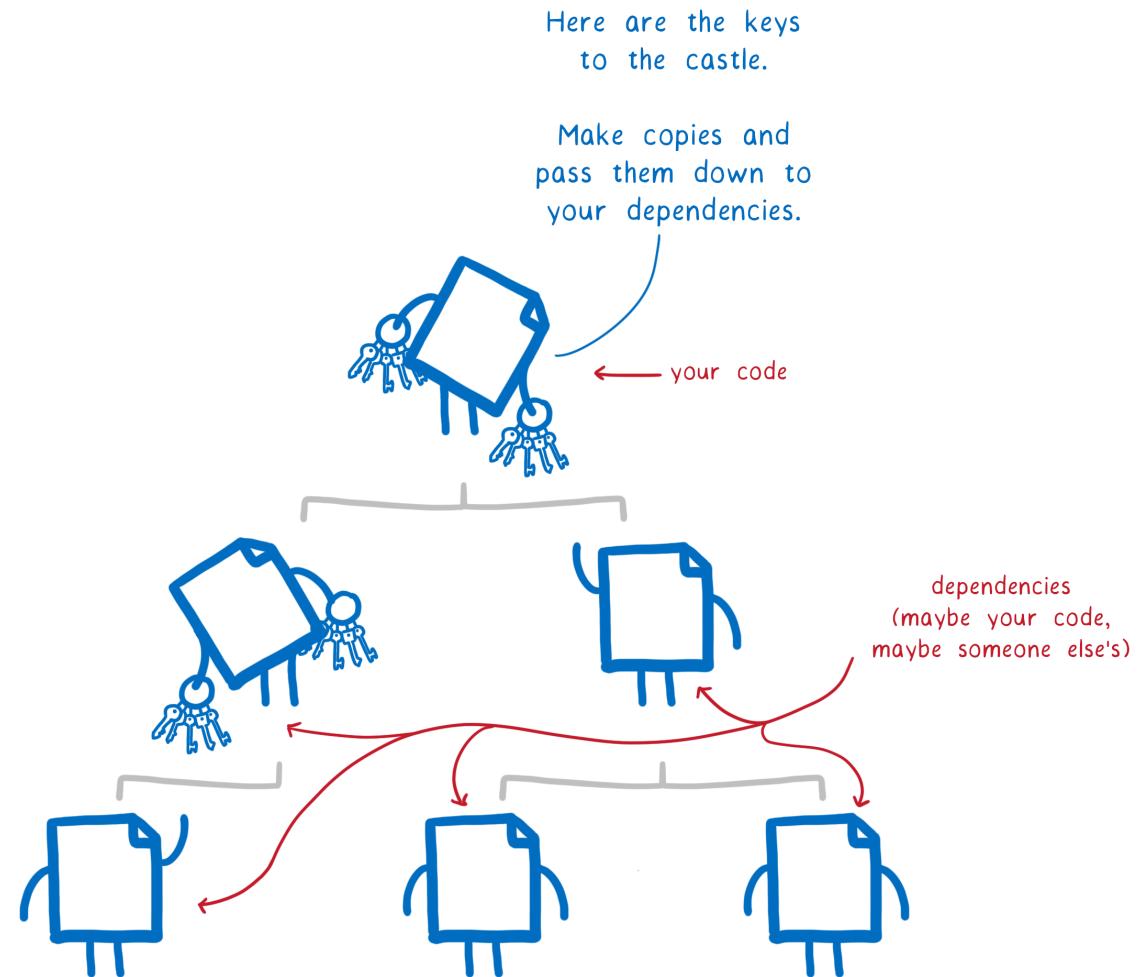
# WASM - What's next?

composition of an  
average code base



0101  
0101

# Dependencies

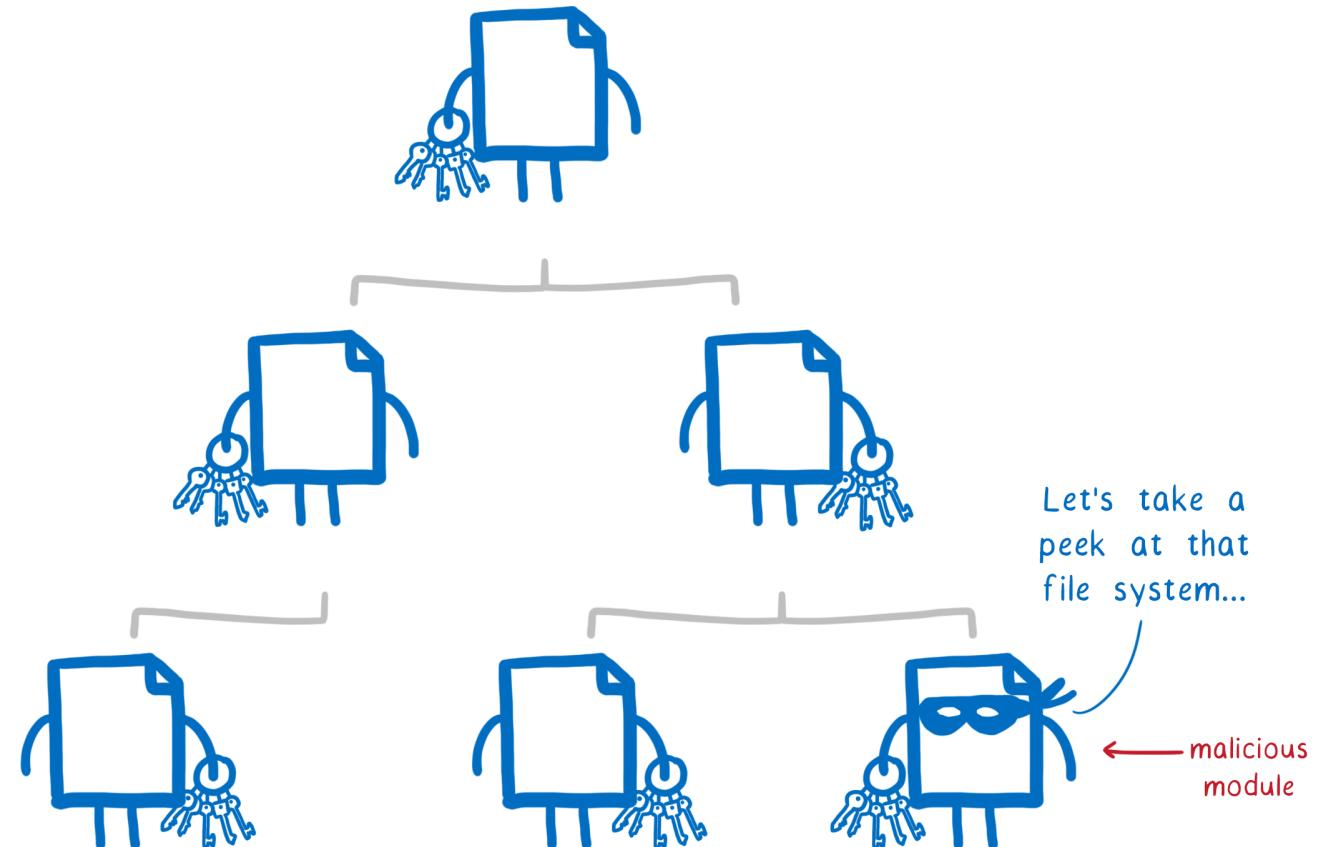


SWETUGG

nis@infosec.exchange

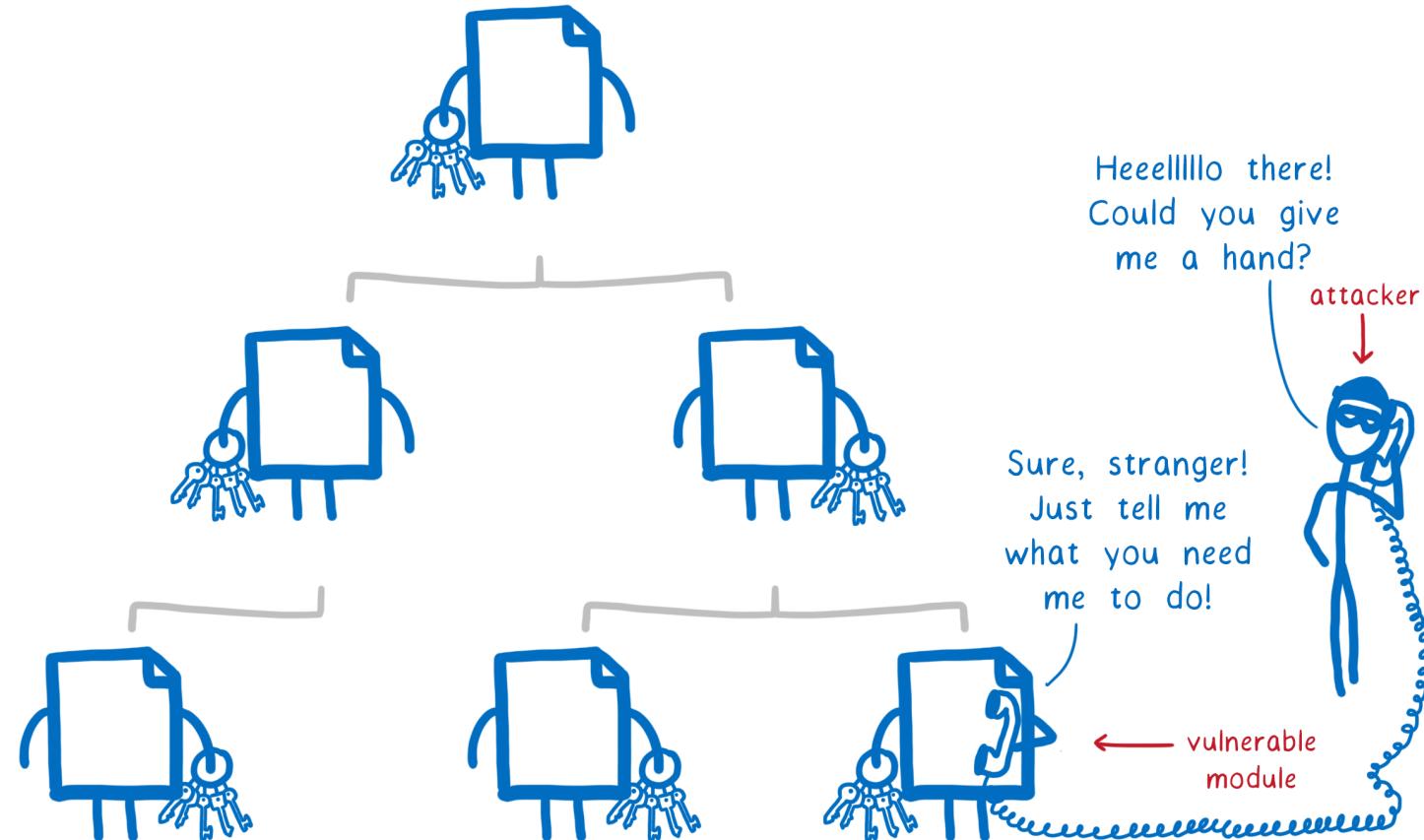
0101  
0101

# Malicious module



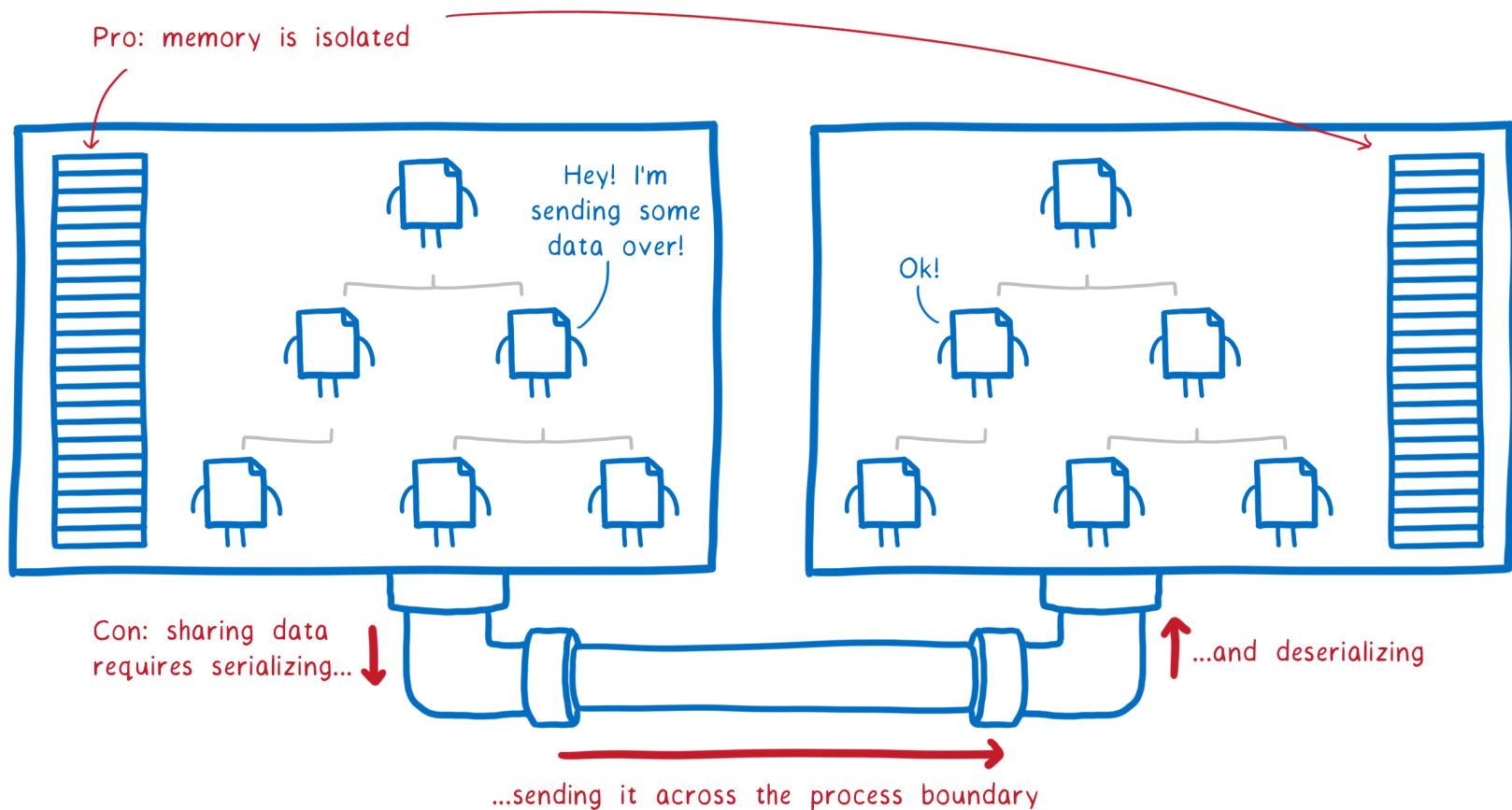
0101  
0101

# Vulnerable module



0101  
0101

# Process Isolation

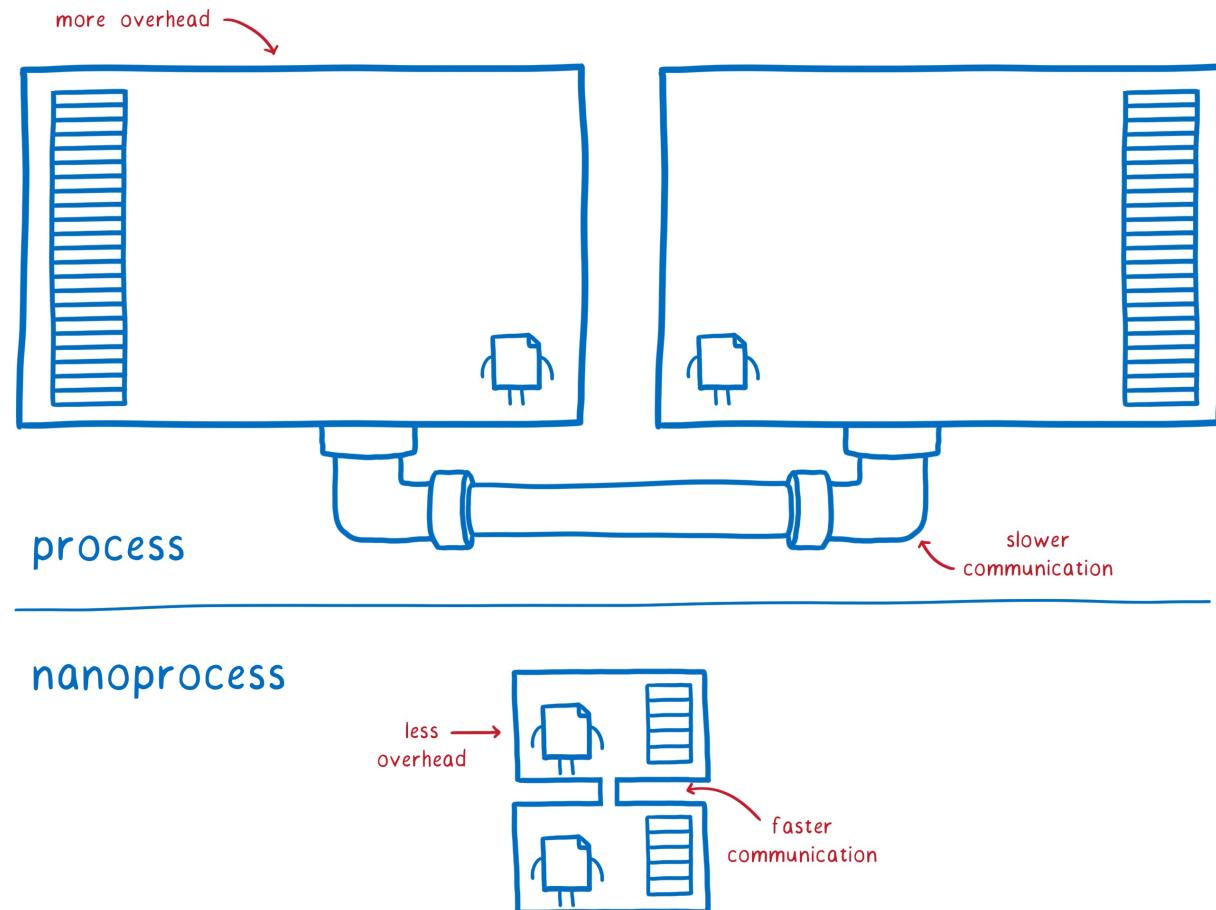


SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly Nano-Process



\* not drawn to scale

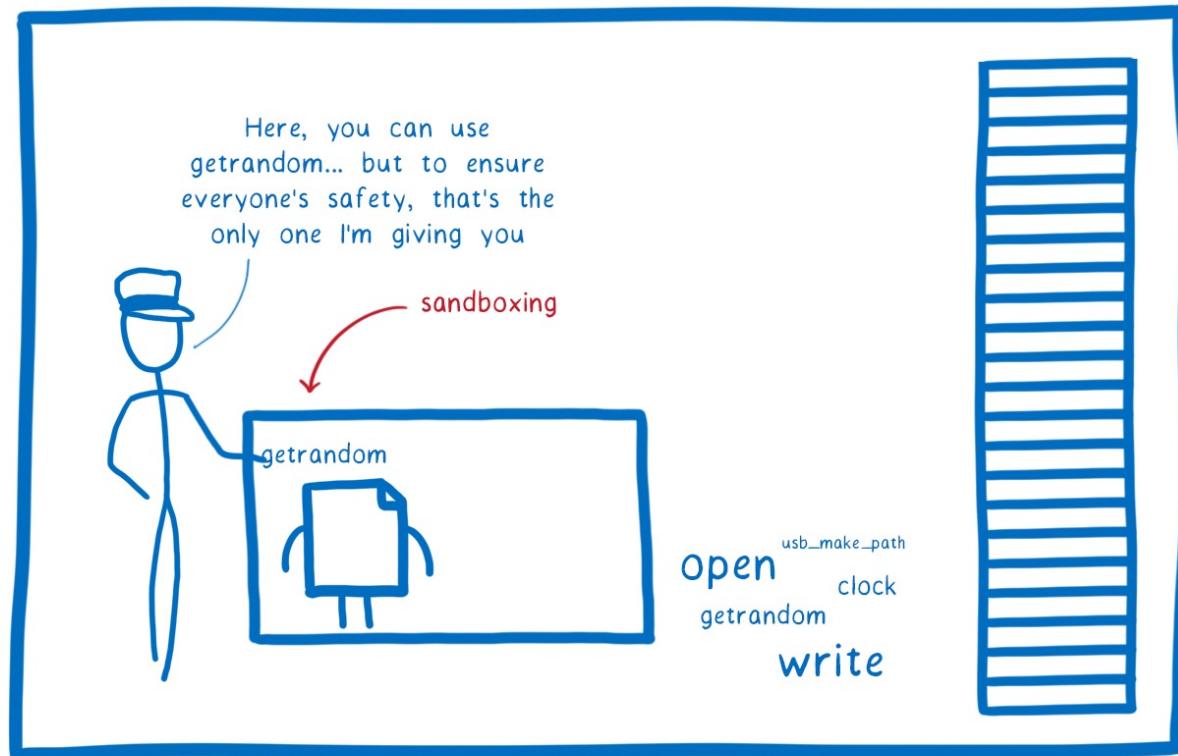
SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly Nano-Process

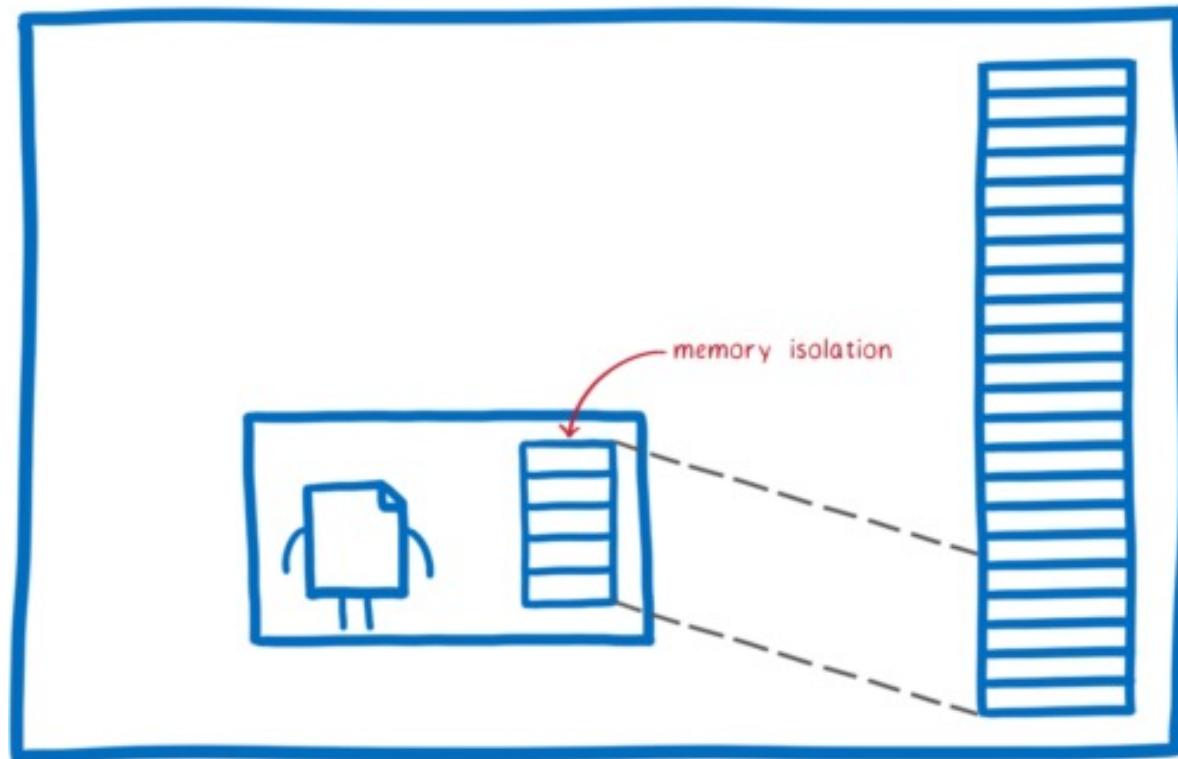
## 1. Sandboxing



0101  
0101

# WebAssembly Nano-Process

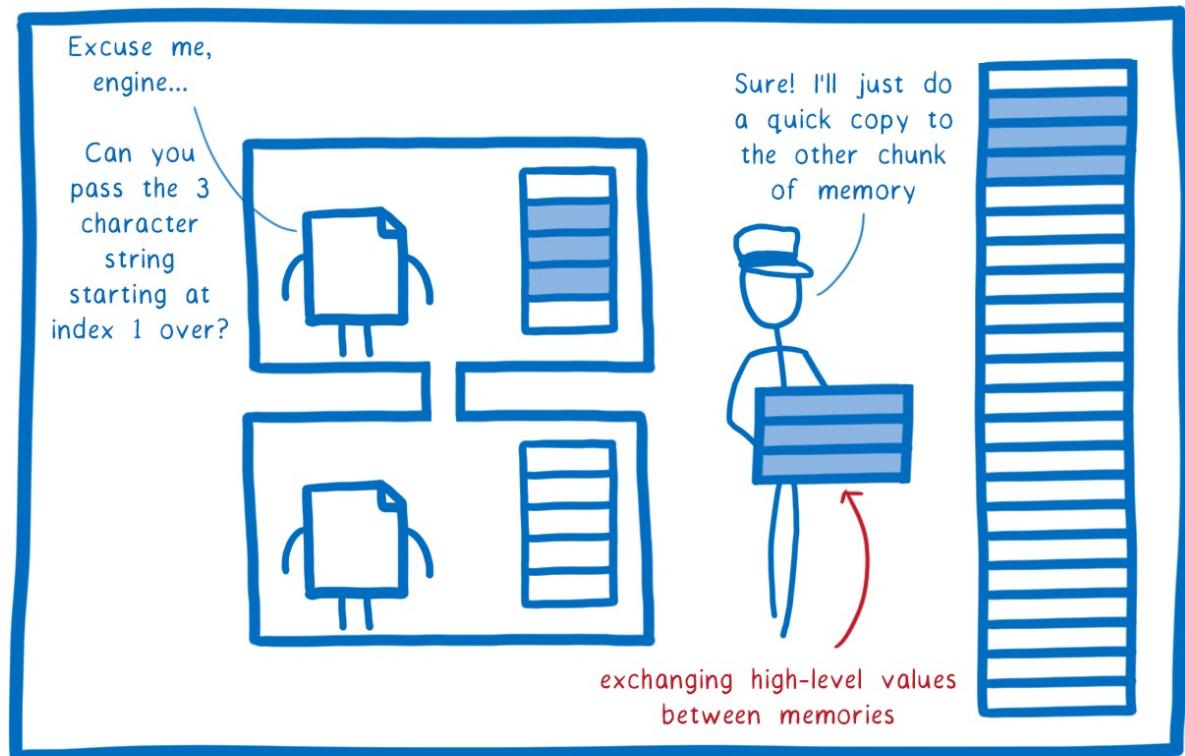
## 2. Memory model



0101  
0101

# WebAssembly Nano-Process

## 3. Interface Types



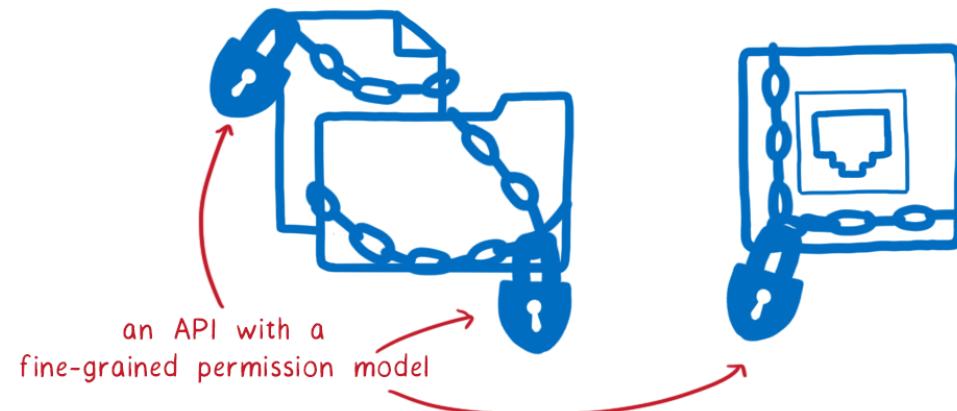
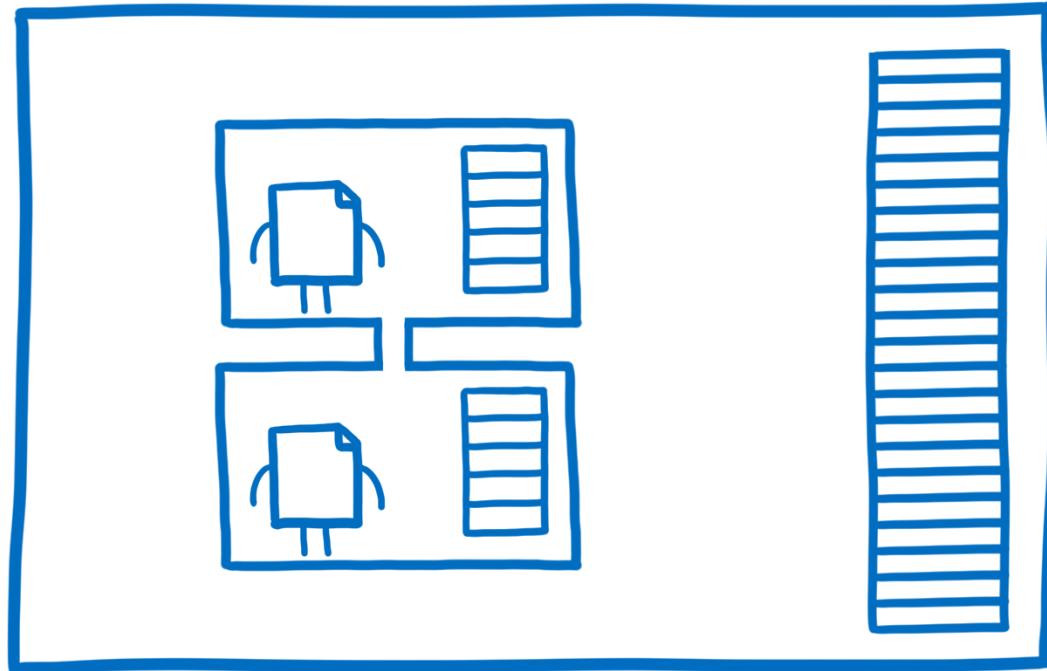
SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly Nano-Process

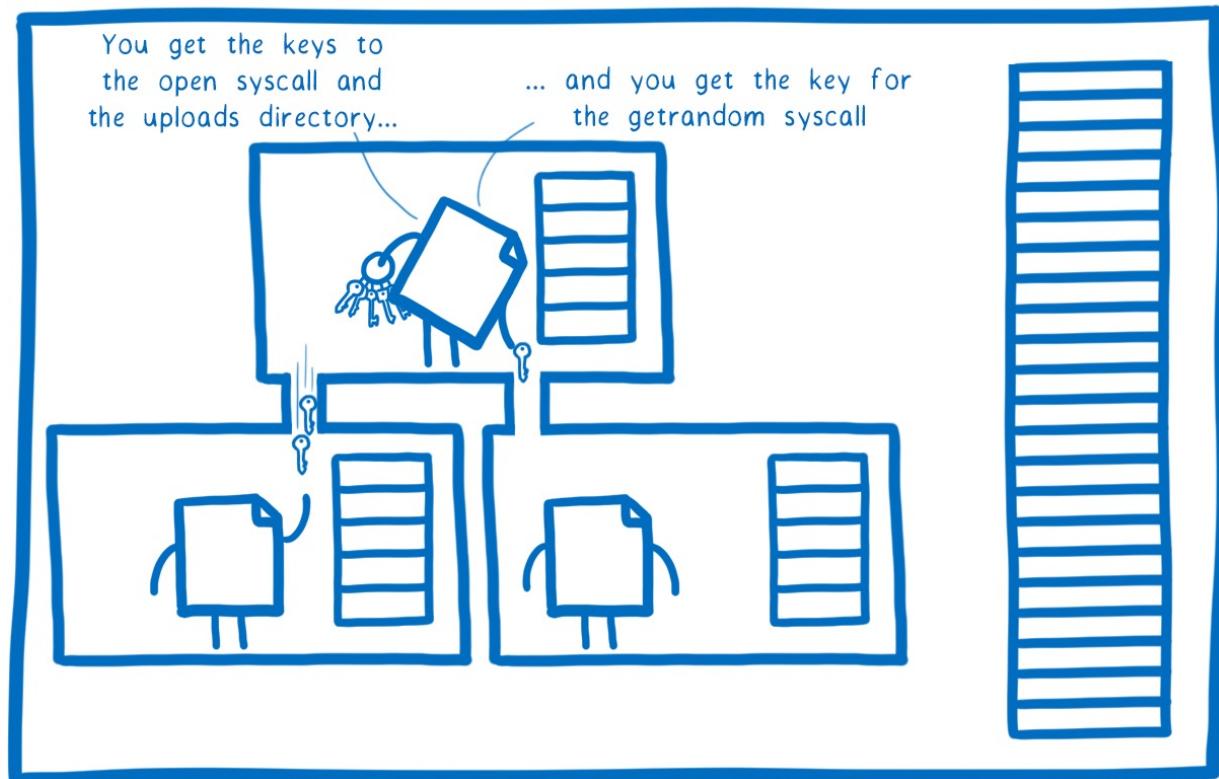
## 4. WebAssembly System Interface



0101  
0101

# WebAssembly Nano-Process

## 5. The missing link

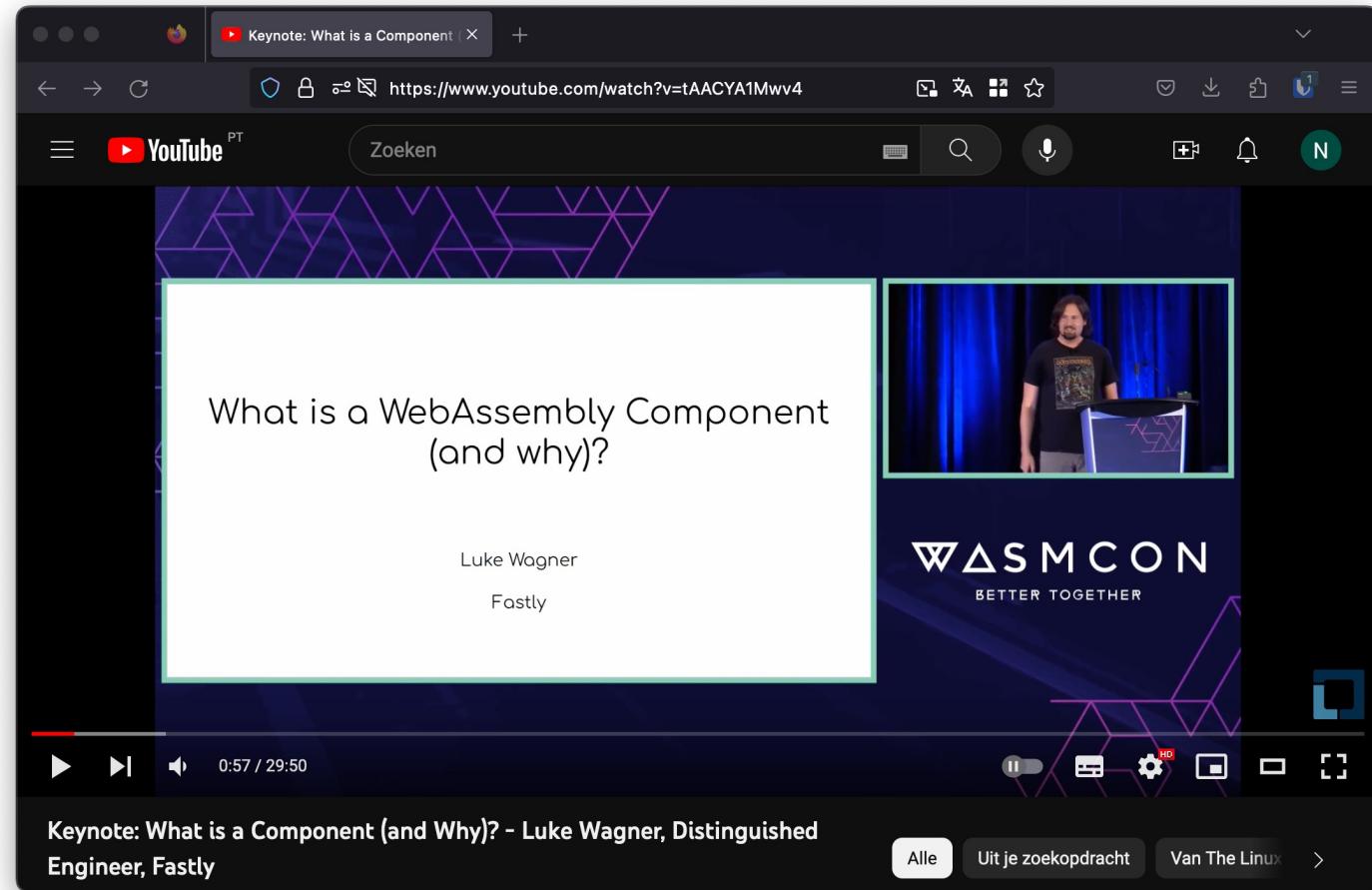


SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WebAssembly Component Model



SWETUGG

@nielstanis@infosec.exchange

0101  
0101

# WASI Preview 2

The screenshot shows a web browser window with the title bar "WASI Preview 2 Launched - sunfishcode". The URL in the address bar is "https://blog.sunfishcode.online/wasi-preview2/". The page content is as follows:

**sunfishcode's blog**  
A blog by sunfishcode

---

## WASI Preview 2 Launched

Posted on January 25, 2024

The WASI Subgroup has just voted to launch WASI Preview 2! This blog post is a brief look at the present, past, and future of WASI.

### The present

The Subgroup voted to launch Preview 2!

This is a major milestone! We made it! At the same time, the journey is only just beginning. But let's talk this moment to step back and look at what this means.

Most immediately, what this means is that the WASI Subgroup officially says that the Preview 2 APIs are stable. There is still a lot more to do, in writing more documentation, more tests, more toolchains, more implementations, and there are a lot more features that we all want to add. This vote today is a milestone along the way, rather than a destination in itself.

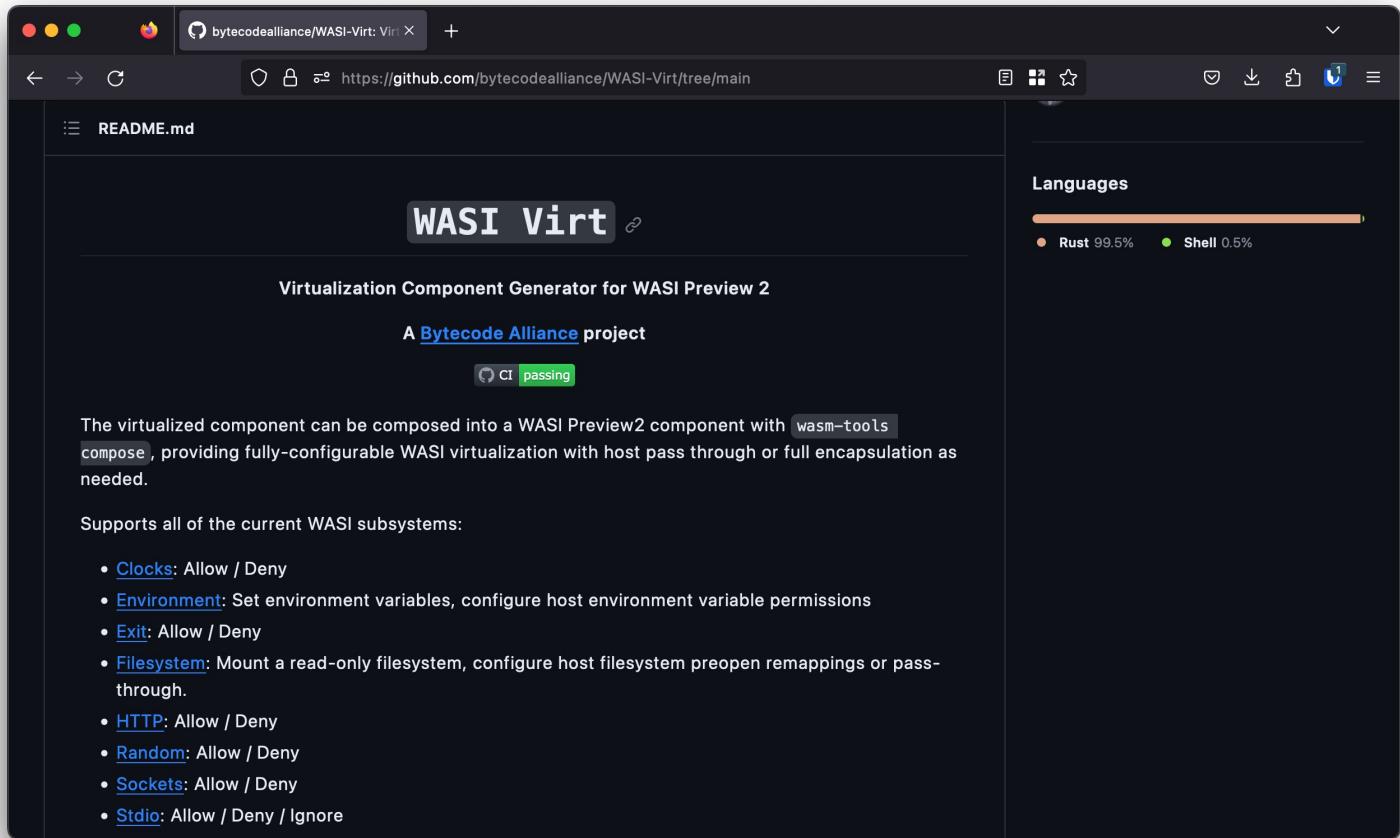
It also means that WASI is now officially based on the Wasm [component model](#), which makes it cross-language and virtualizable. Figuring out what a component model even is, designing it, implementing it, and building APIs using it has been a

SWETUGG

 @nielstanis@infosec.exchange

0101  
0101

# WASI Virt



SWETUGG

@nielstanis@infosec.exchange

# WasmComponent.SDK

0101  
0101

The screenshot shows a GitHub README page for the `WasmComponent.Sdk` repository. The page has a dark theme. At the top, there's a navigation bar with icons for file, search, and repository details. Below that is a header bar with back, forward, refresh, and search buttons, along with a URL bar showing <https://github.com/SteveSandersonMS/wasm-component-sdk/>. The main content area starts with a `README` section, indicated by a tab at the top left. The title `WasmComponent.Sdk` is bolded. A descriptive paragraph follows, stating: "An experimental package to simplify building WASI preview 2 components using .NET, including early support for WIT files." Another paragraph explains the purpose: "The build output is fully AOT compiled and is known to work in recent versions of wasmtime and WAMR." A `Purpose` section is present, with a note about simplifying experimentation and prototyping. A detailed paragraph discusses the complexity of building components without this package, mentioning the need to discover, download, configure, and manually chain together various tools like WIT imports/exports. The final paragraph states that with this package, you can add one NuGet reference and get started quickly. On the right side of the screen, there's a sidebar for the repository owner, `NielsPilgaard`, Niels Pilgaard Grøndahl. It shows his profile picture, a progress bar for his languages (C# at 100%), and a link to his GitHub profile.

SWETUGG

@nielstanis@infosec.exchange

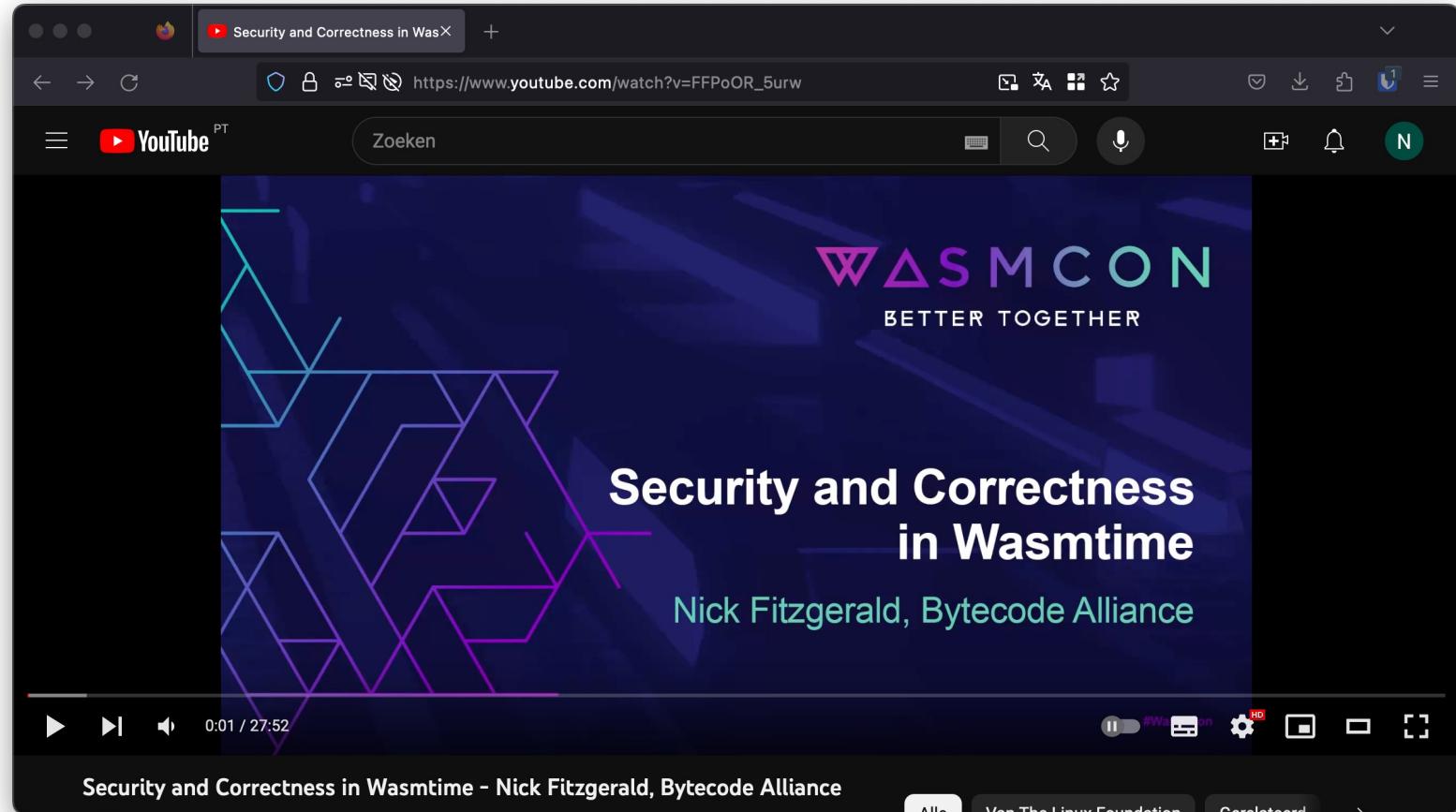


# Runtimes and Security

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

0101  
0101

# Runtimes and Security



SWETUGG

@nielstanis@infosec.exchange



# 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!
- WASI Preview 2 big milestone; now tooling can be implemented!



# Questions?

- <https://github.com/nielstanis/swetugg2024>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Tack! Thank you!

SWETUGG

 [@nielstanis@infosec.exchange](mailto:@nielstanis@infosec.exchange)