

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

Niels Tanis



VERACODE

0101
0101

Who am I?

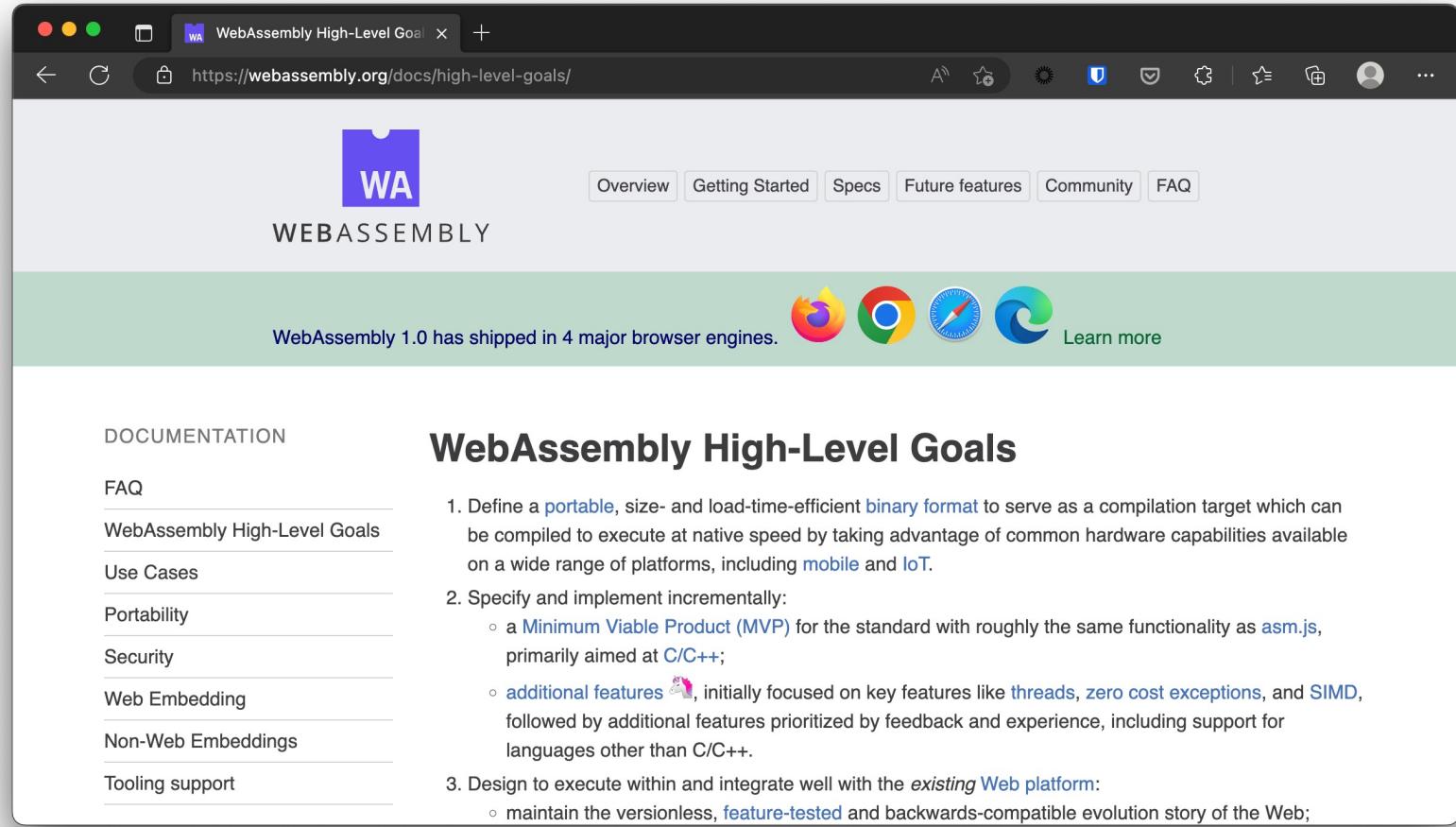
- Niels Tanis
- Sr. Principal Security Researcher
 - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
 - Research on static analysis for .NET apps
- Microsoft MVP - Developer Technologies



@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:

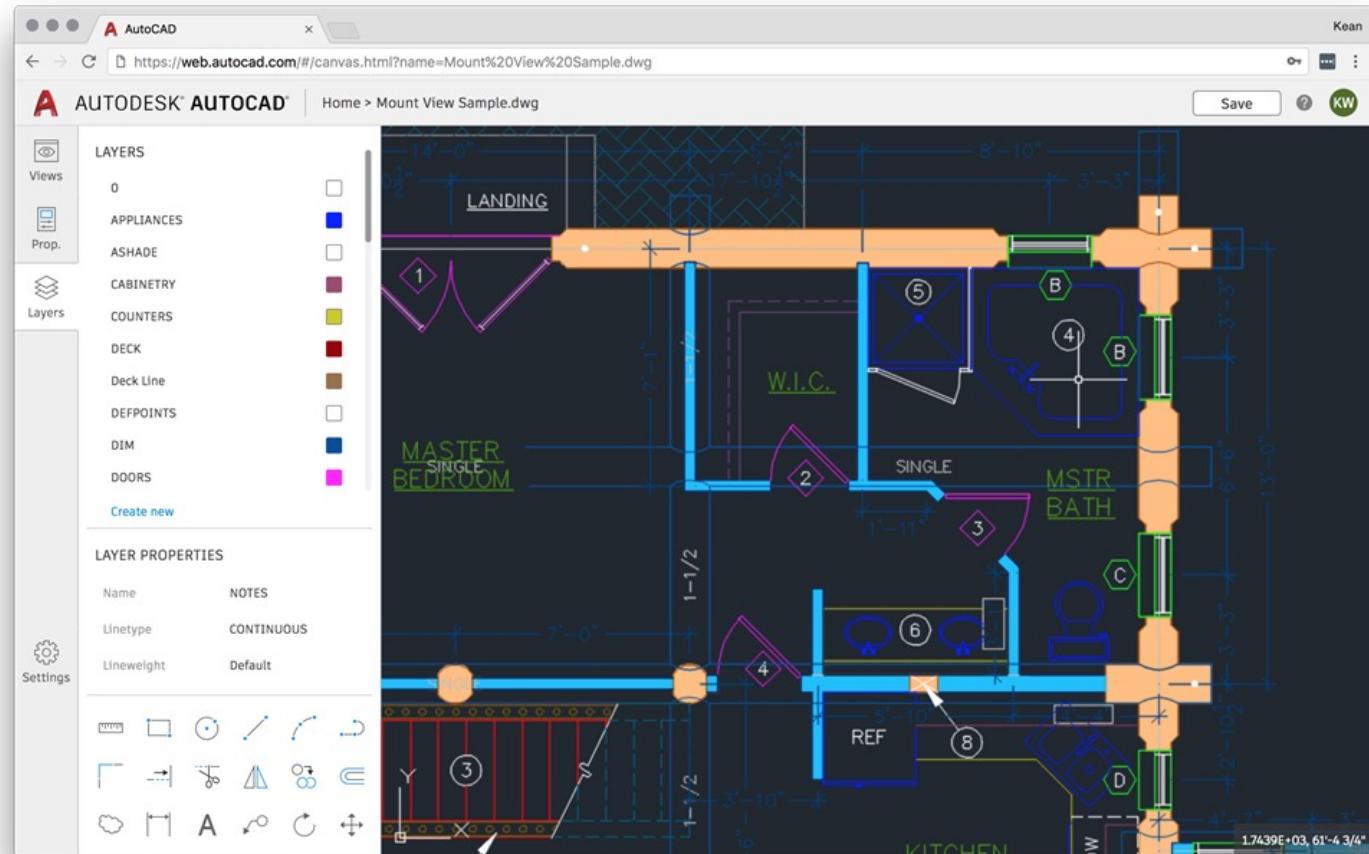
1. Define a portable, size- and load-time-efficient binary format to serve as a compilation target which can be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT.
2. Specify and implement incrementally: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



 @nielstanis@infosec.exchange

0101
0101

WebAssembly - AutoCAD



@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 part of the "CLOUD AND SYSTEMS" section under the "amazon | science" header. It discusses the switch to WebAssembly and its benefits. The content mentions Prime Video's delivery to millions of customers on various devices.



@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

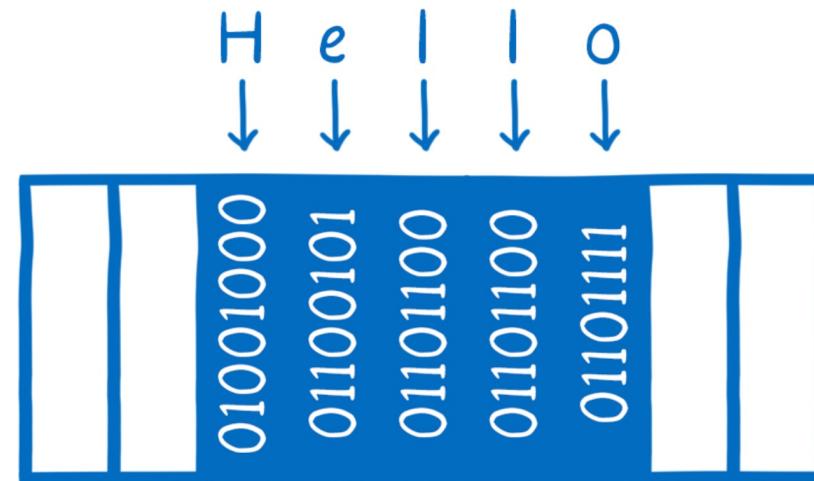
- Binary instruction format for stack-based virtual machine similar to .NET running MSIL
- Designed as a portable compilation target
- The security model of WebAssembly:
 - Protect users from buggy or malicious modules
 - Provide developers with useful primitives and mitigations for developing safe applications



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");
```

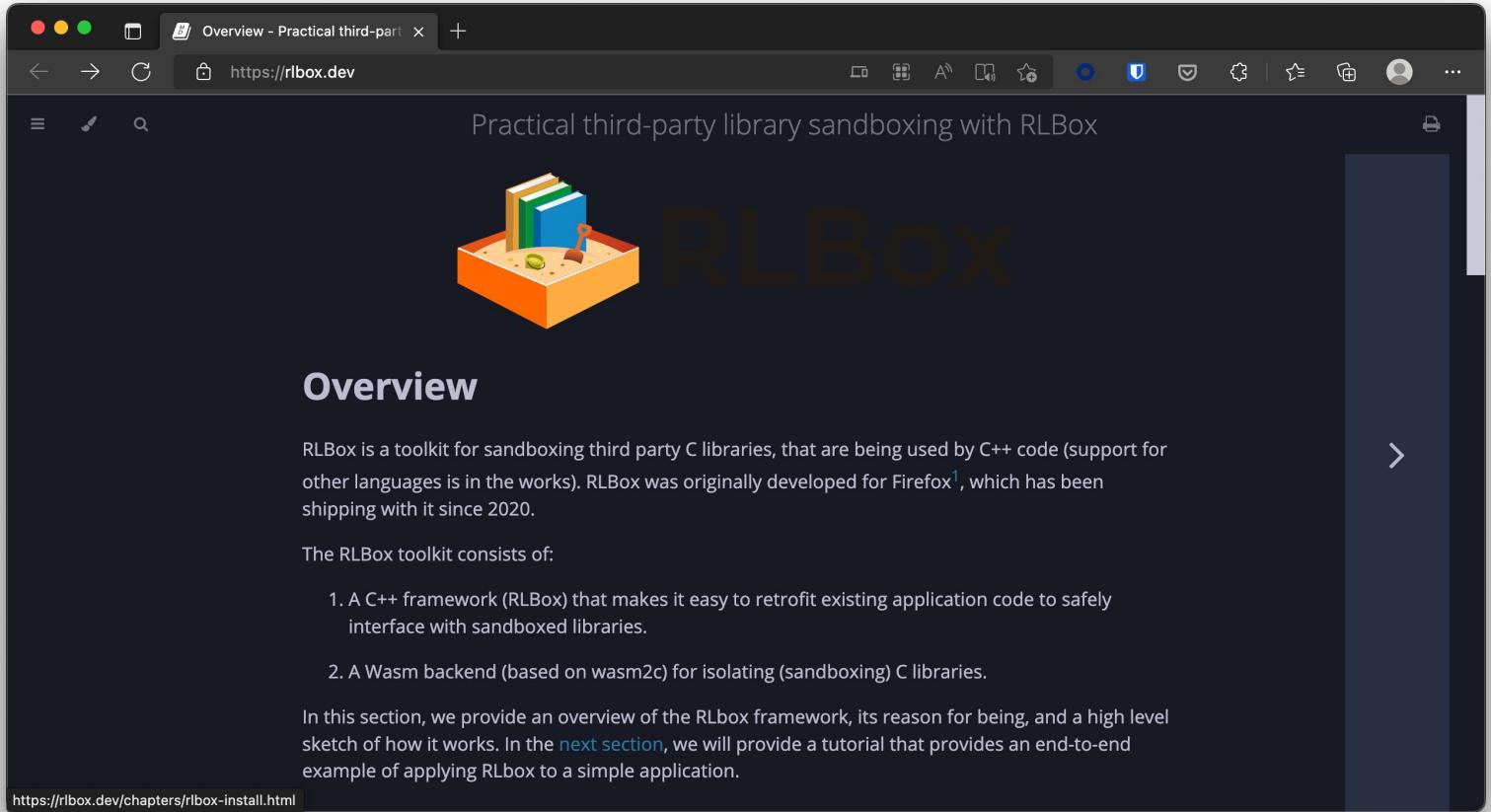
```
Console.WriteLine("Number is smaller than 5");
```

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



0101
0101

FireFox RLBox



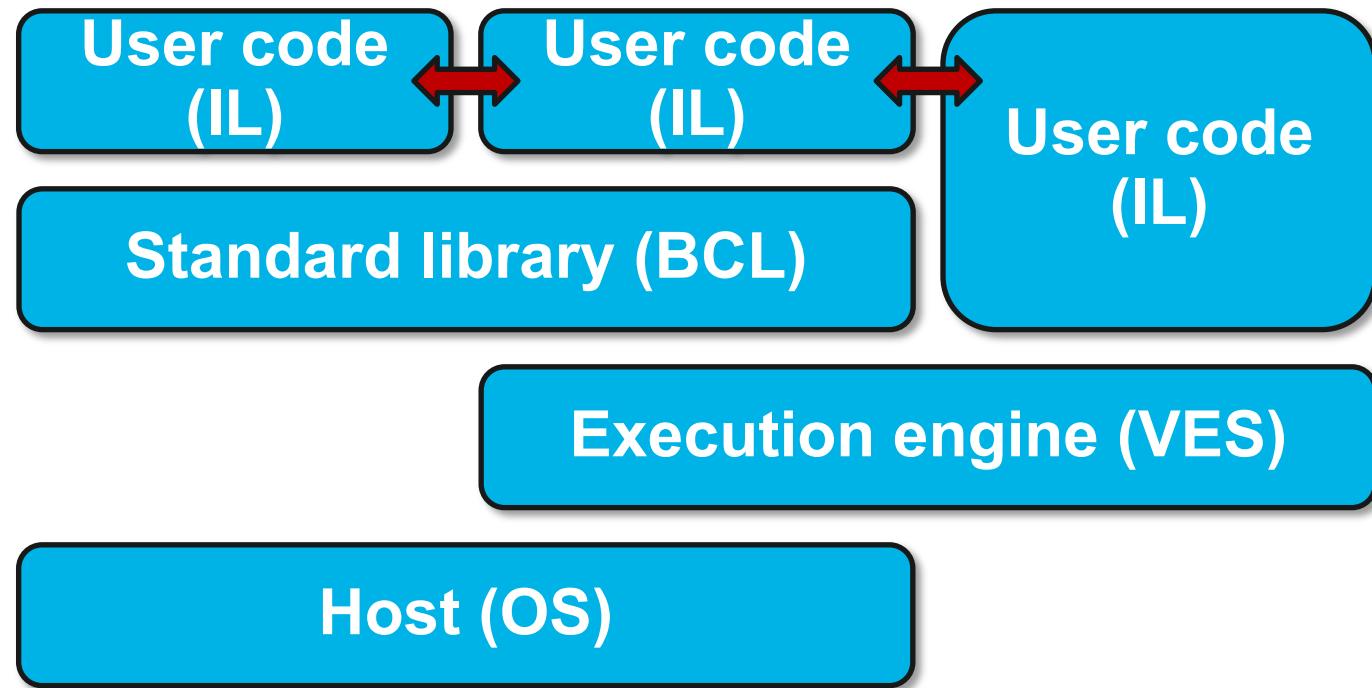
The screenshot shows a Firefox browser window with a dark theme. The title bar says "Overview - Practical third-part" and the address bar shows "https://rlbox.dev". The main content area has a dark background with a large orange sandcastle icon containing three books and a shovel. To the right of the icon, the word "RLBox" is written in large, bold, dark letters. Below the icon, the word "Overview" is centered in a 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 leads to "https://rlbox.dev/chapters/rlbox-install.html".



 @nielstanis@infosec.exchange

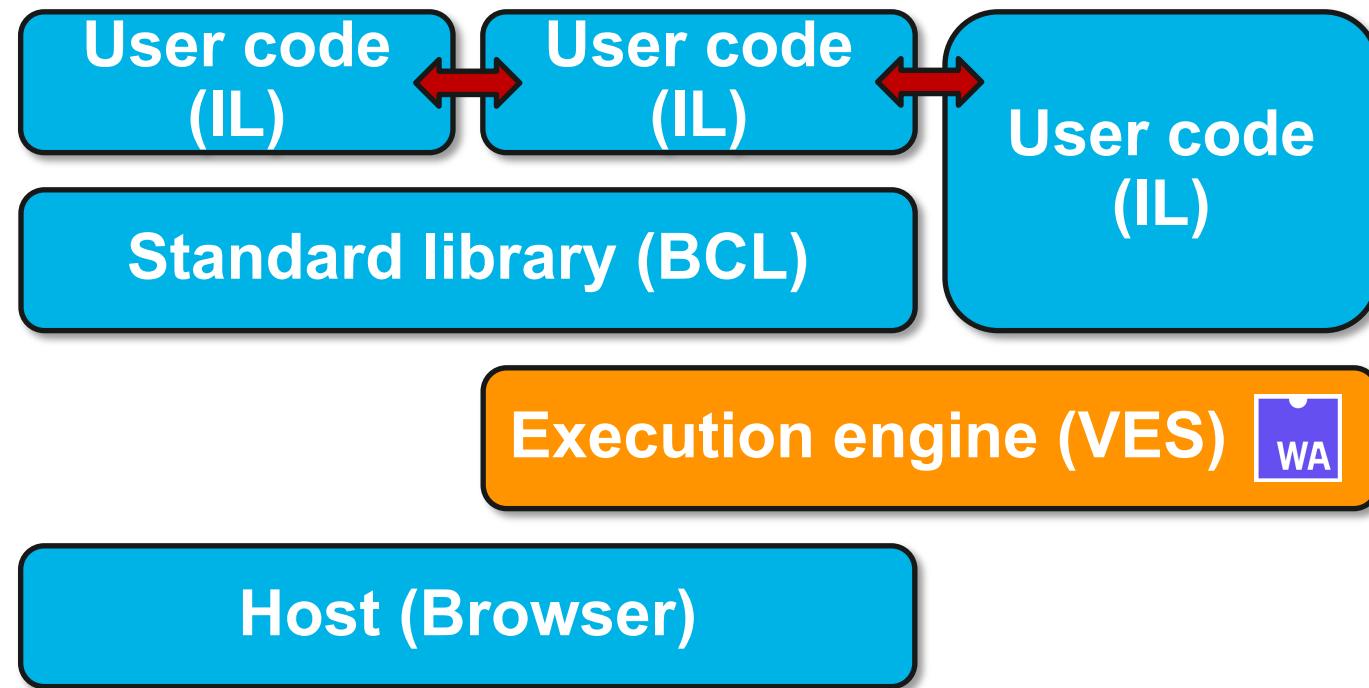


Running .NET on WebAssembly



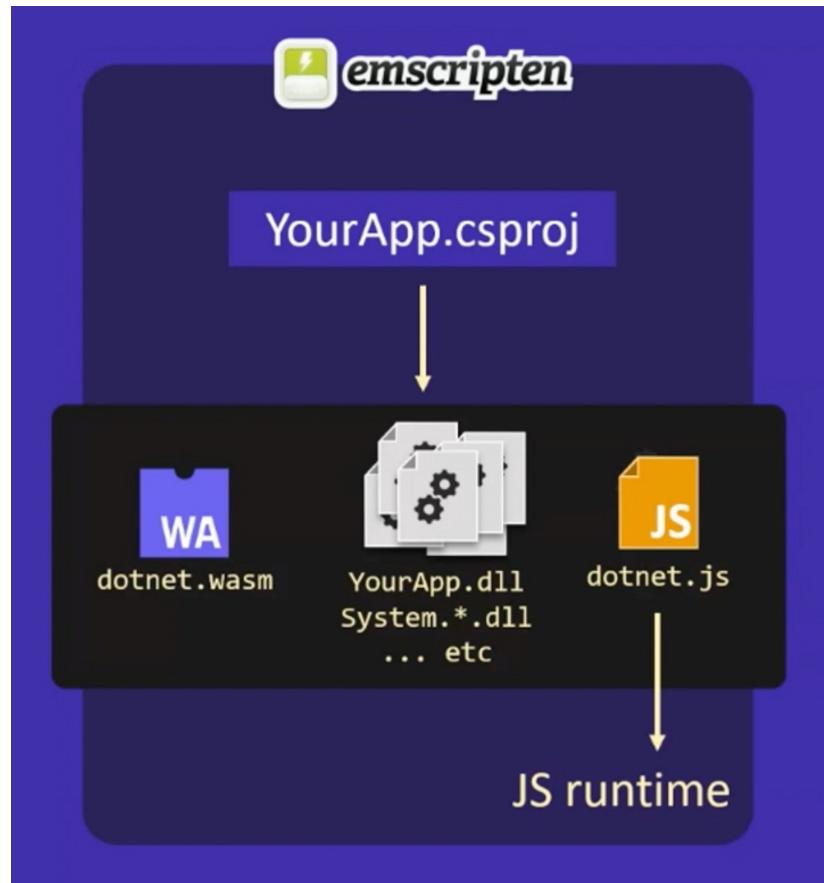


Running .NET on WebAssembly



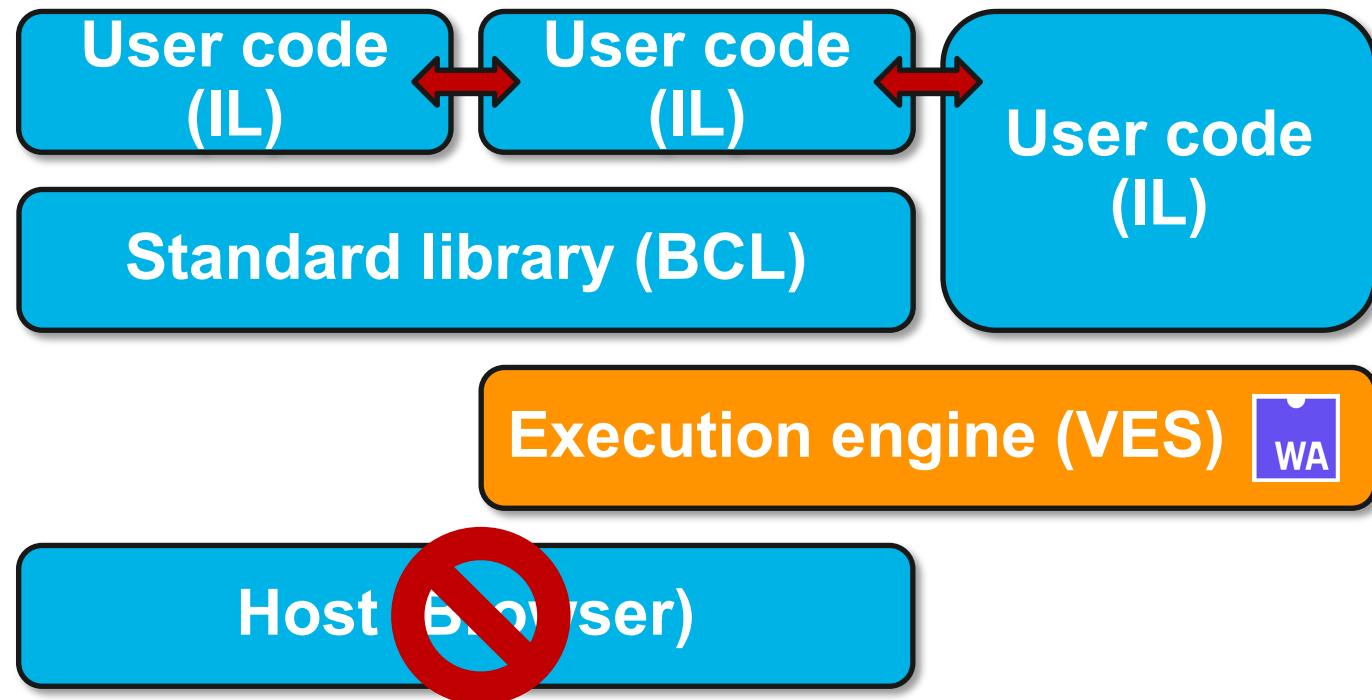
0101
0101

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 recall .NET Standard? ☺



0101
0101

Docker vs WASM & WASI

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

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, there 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](#)

The Twitter interface shows a sidebar with various icons for navigation.



0101
0101

Docker vs WASM & WASI

Solomon Hykes op Twitter: "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 :)

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...](https://twitter.com/linclark/status/110541444000000000)

Deze collectie weergeven

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

56 Retweets 5 Geciteerde Tweets 165 Vind-ik-leuks



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 heading "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 text block 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 runtime environments: "runc" (which runs "Container process"), "runc" (which runs "Container process"), and "wasmedge" (which runs a "Wasm Module"). Each runtime environment is preceded by a "containerd-shim" layer. Arrows indicate the flow of control from the Docker Engine down through containerd and its shims to the final runtime and application layers.

Let's look at an example!

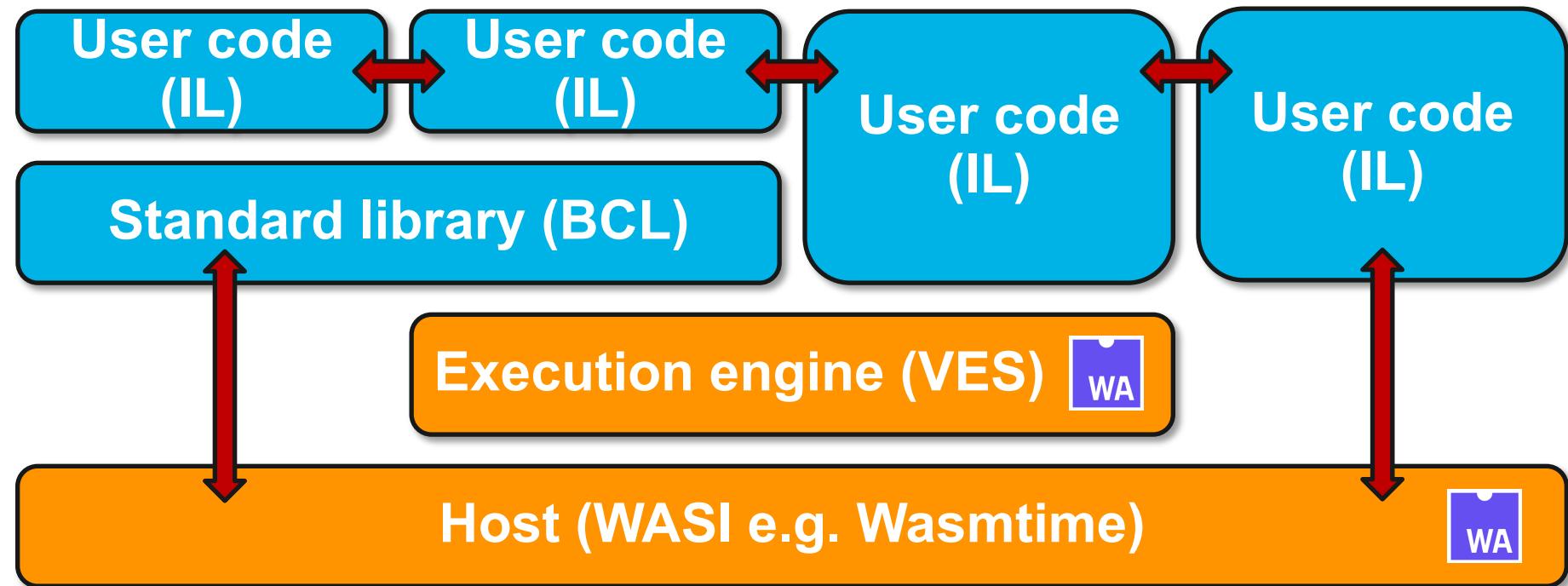
After installing the preview, we can run the following command to start an example Wasm application:

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



WebAssembly System Interface WASI

0101
0101



0101
0101

Experimental WASI SDK for .NET



Experimental WASI SDK for .NET

0101
0101

The screenshot shows a GitHub issue page for the 'dotnet/runtime' repository. The issue is titled 'WASI support tracking #65895'. It is marked as 'Open' and has 14 of 21 tasks completed. SteveSandersonMS opened the issue on Feb 25, 2022, with 5 comments. The issue description states: "This issue is to track known issues in the early WASI-enabled runtime builds. These need to be resolved in order to have a proper supportable WASI-ready release." Below the description, there are two main sections of tasks:

- Add `wasi-wasm` RID
 - [wasi] Add new RID for WASI #77780
 - [API Proposal]: `OperatingSystem.IsWasi()` #78389 API change
- Build `System.Native` using WASI SDK instead of using the Emscripten-built binary.
 - MSbuild files for it and generate `wasm_m2n_invoke.g.h`
 - Enable p/invoke to all APIs exposed by `System.Native`. Currently there's a small hardcoded list.
 - [wasi] compile with `libs.native` #79046

On the right side of the issue page, there are sections for 'Assignees' (pavelsavara, thaystg, mkhamoyan), 'Labels' (arch-wasm, area-Build-mono, os-wasi, tracking), and 'Projects' (None yet).



@nielstanis@infosec.exchange



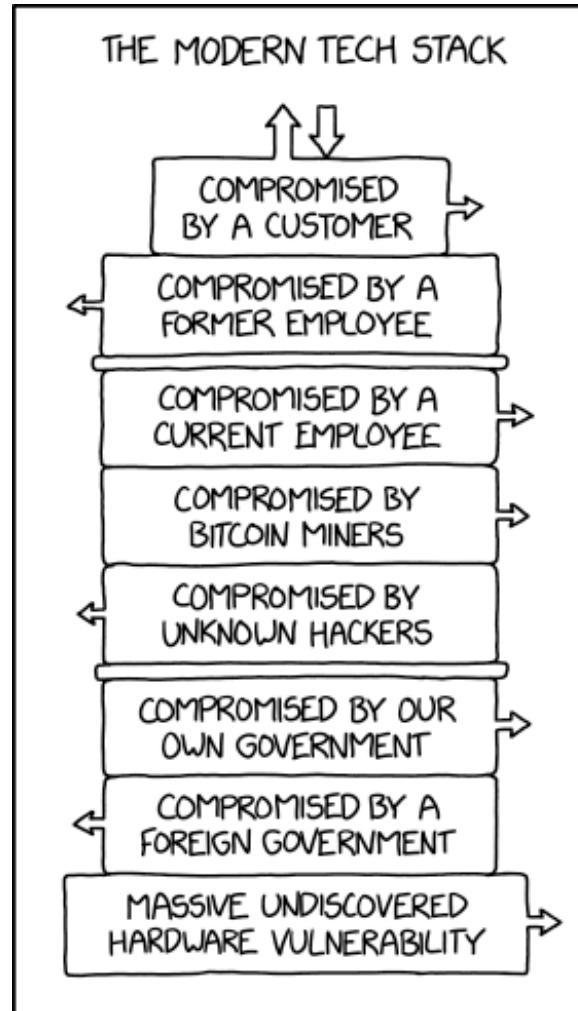
Extending .NET with WASM

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



0101
0101

Trusted Computing - XKCD 2166



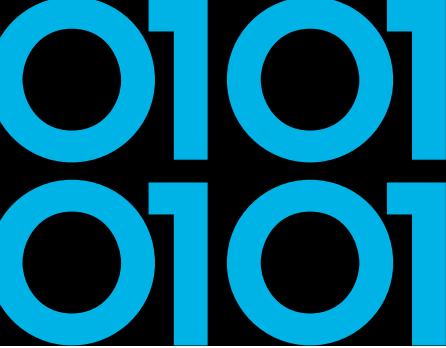
Enarx

0101
0101

The screenshot shows a dark-themed web browser window displaying the Enarx homepage at <https://enarx.dev>. The page features a large title "Enarx" and subtitle "Confidential Computing with WebAssembly". It includes two prominent buttons: "Download" (green) and "Try Enarx" (blue). Below these buttons is a large white icon of a padlock. The text "100% Open Source" is displayed above a dark banner at the bottom. The banner contains the text: "Enarx is the leading open source framework for running applications in TEEs (Trusted Execution Environments). It's part of the Confidential Computing Consortium from the Linux Foundation." The browser's address bar shows the URL, and the top bar includes standard navigation and search icons.

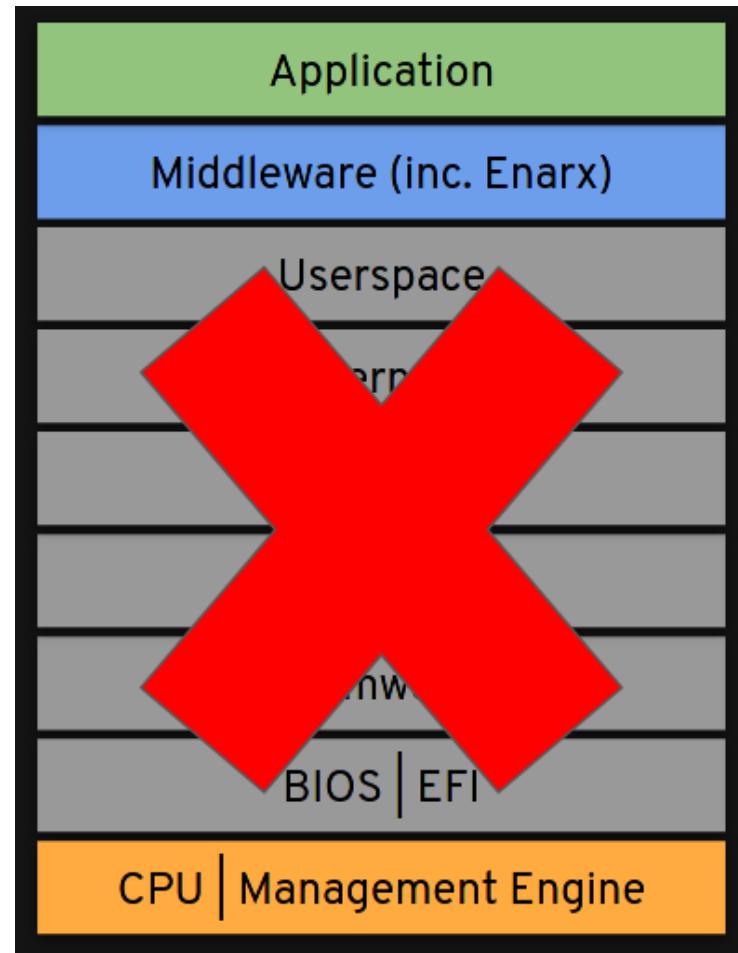


@nielstanis@infosec.exchange



Enarx Threat Model

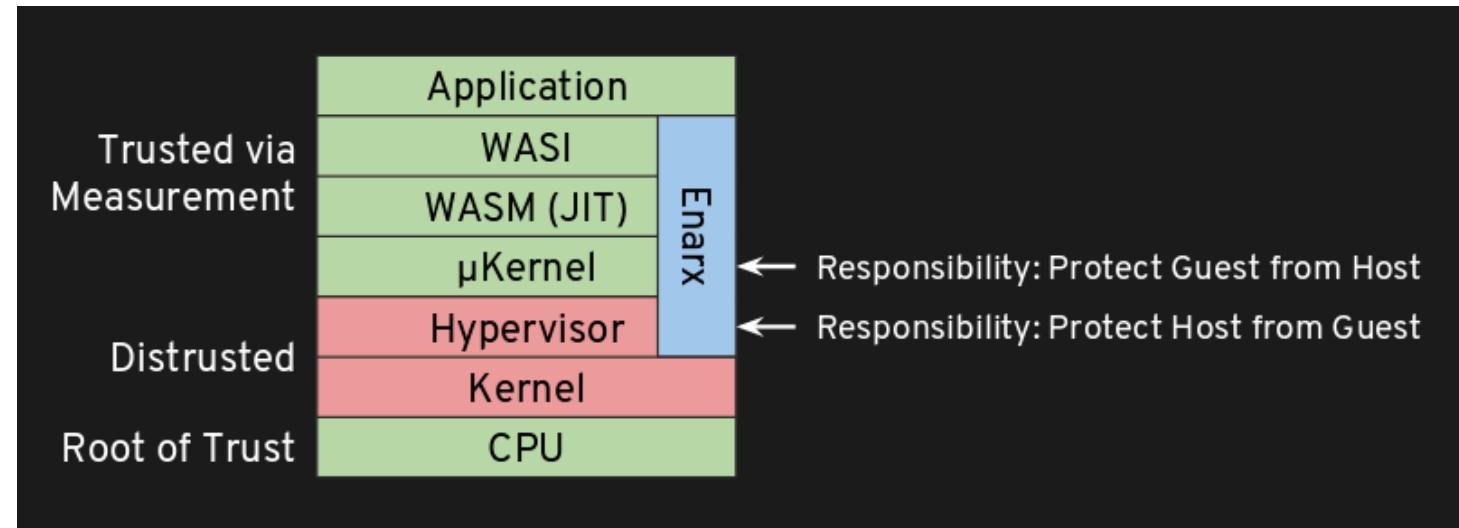
- Don't trust the host
- Don't trust the host owner
- Don't trust the host operator
- Hardware cryptographically verified
- Software audited and cryptographically verified





Enarx

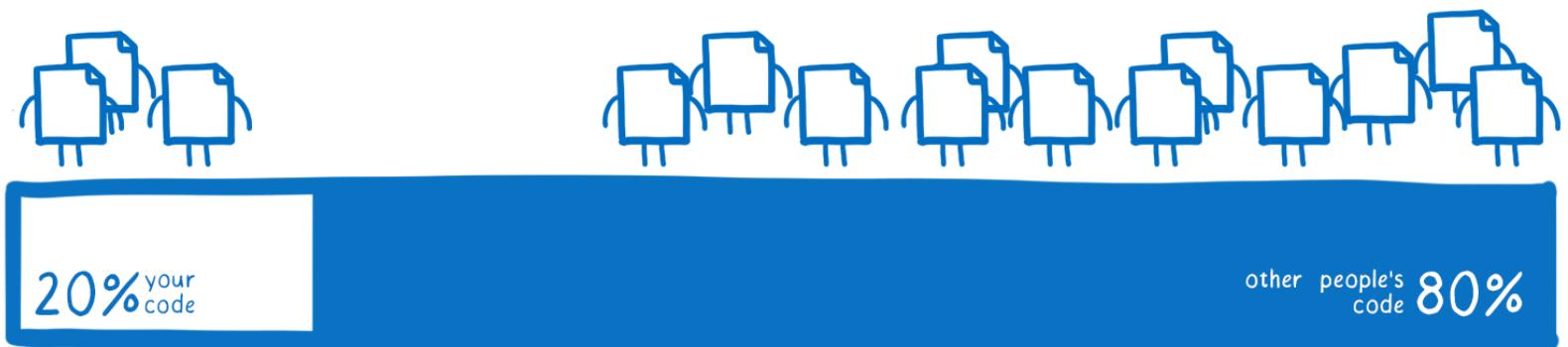
- Leverages Trusted Execution Environment (TEE) direct on processor
 - AMD's SEV, Intel's SGX and IBM's PEF
- Attestation of hardware and Enarx runtime



0101
0101

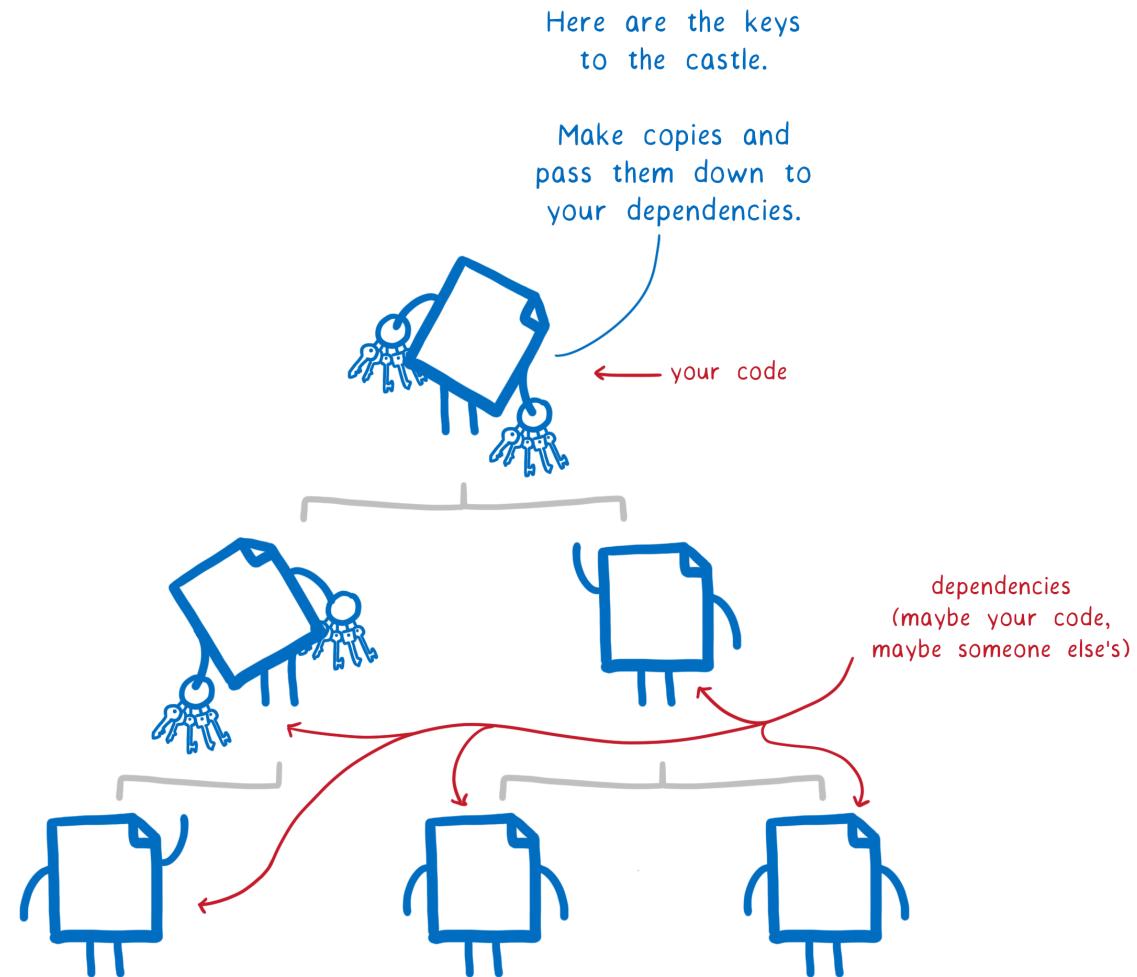
WASM - What's next?

composition of an
average code base



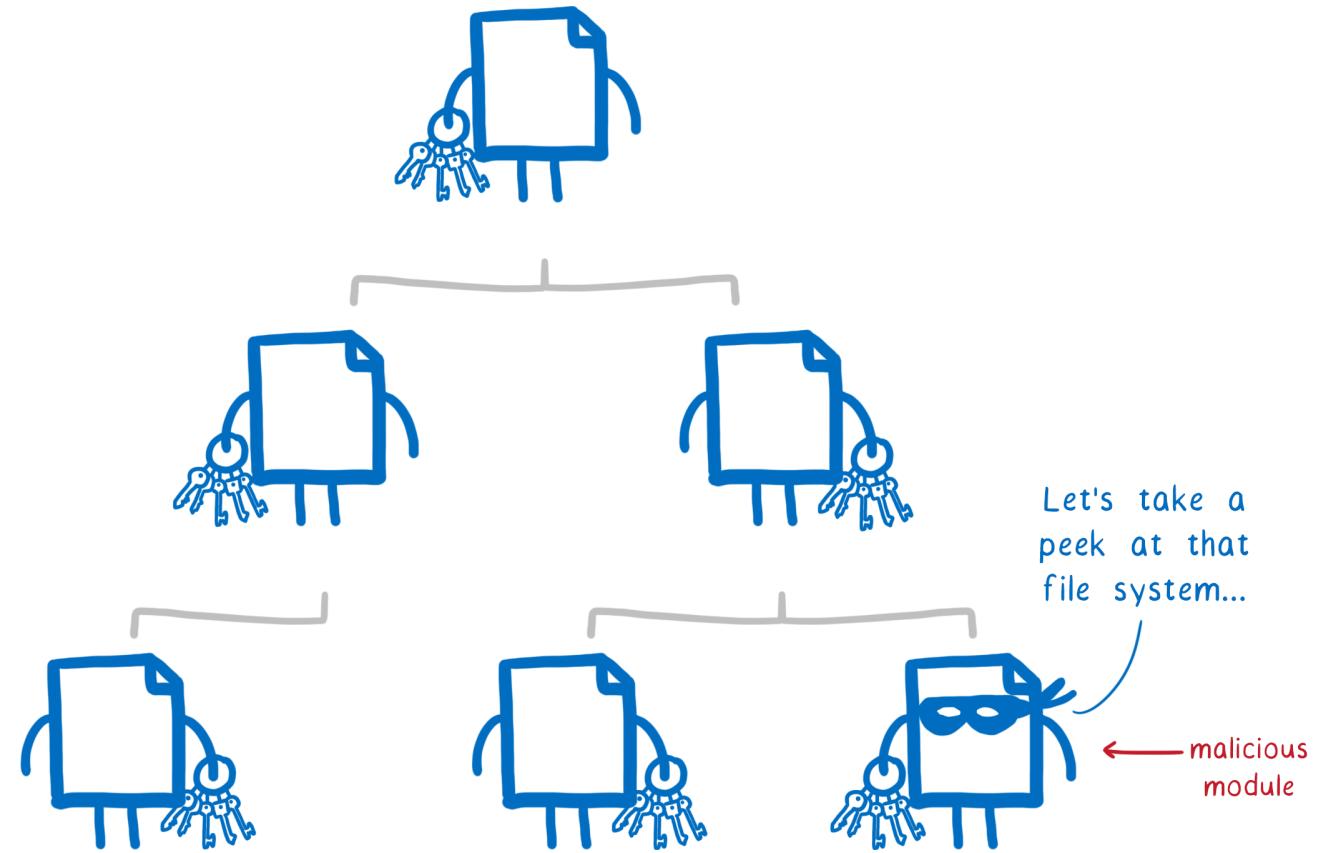
0101
0101

Dependencies



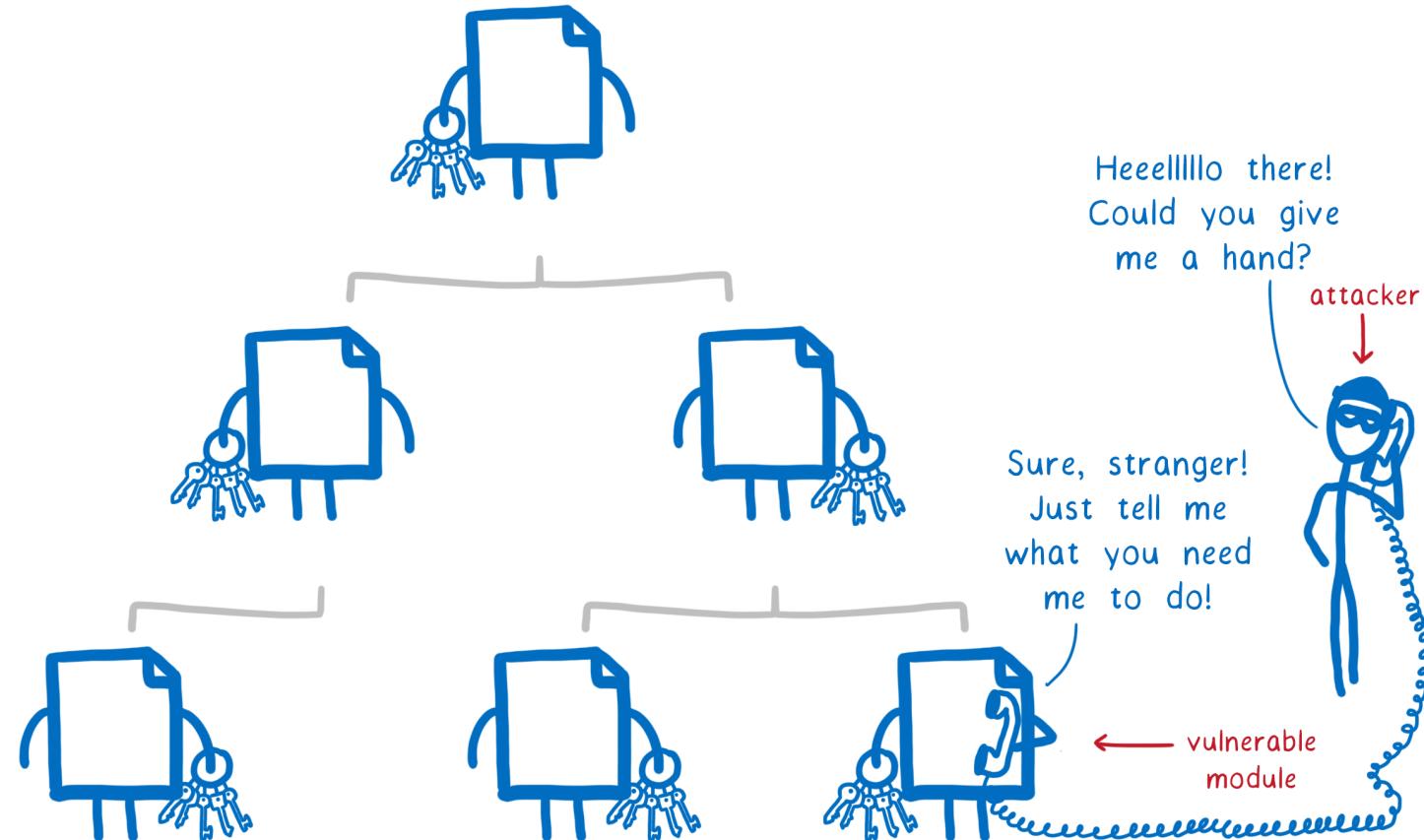
0101
0101

Malicious module



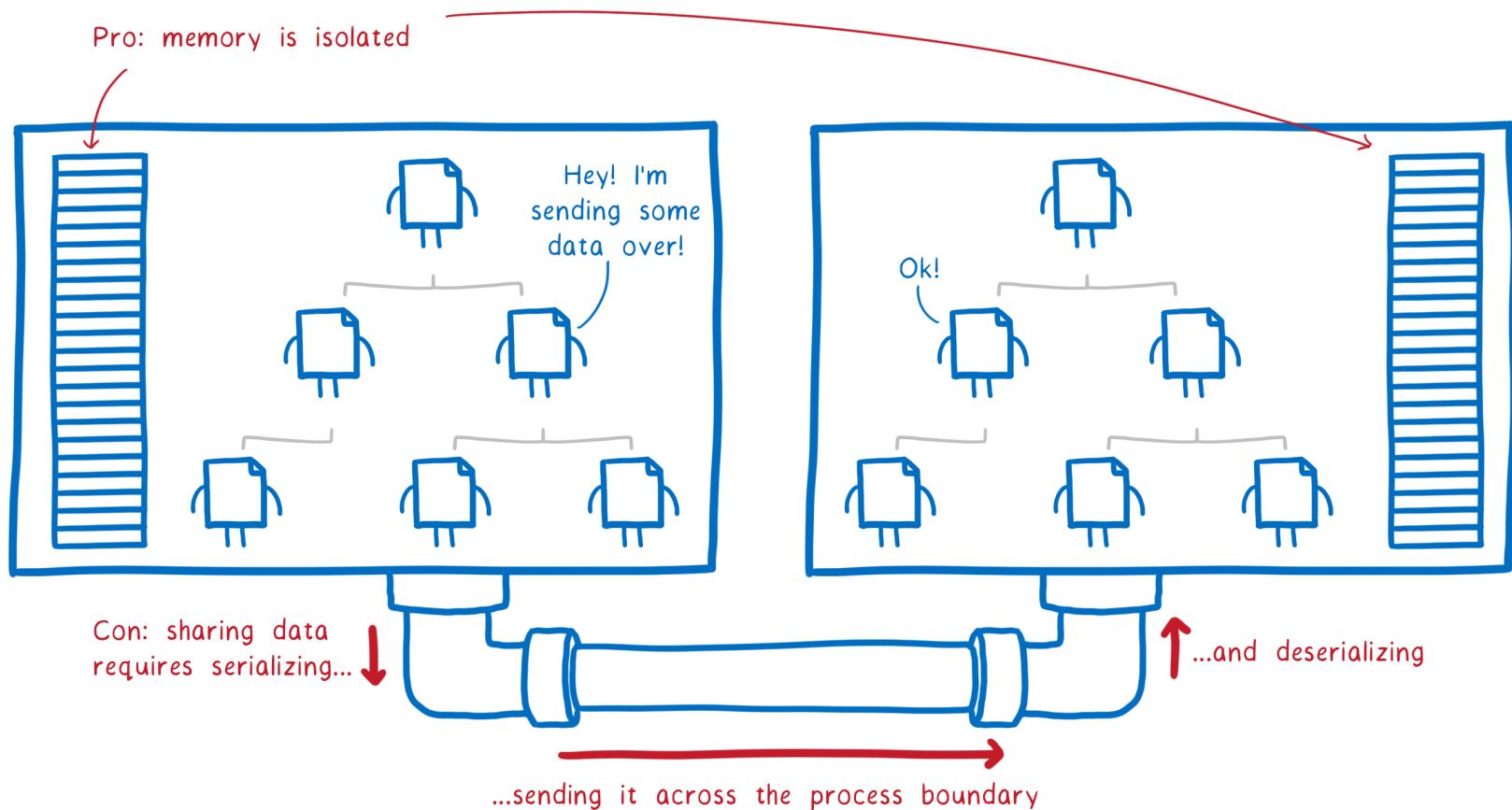
0101
0101

Vulnerable module



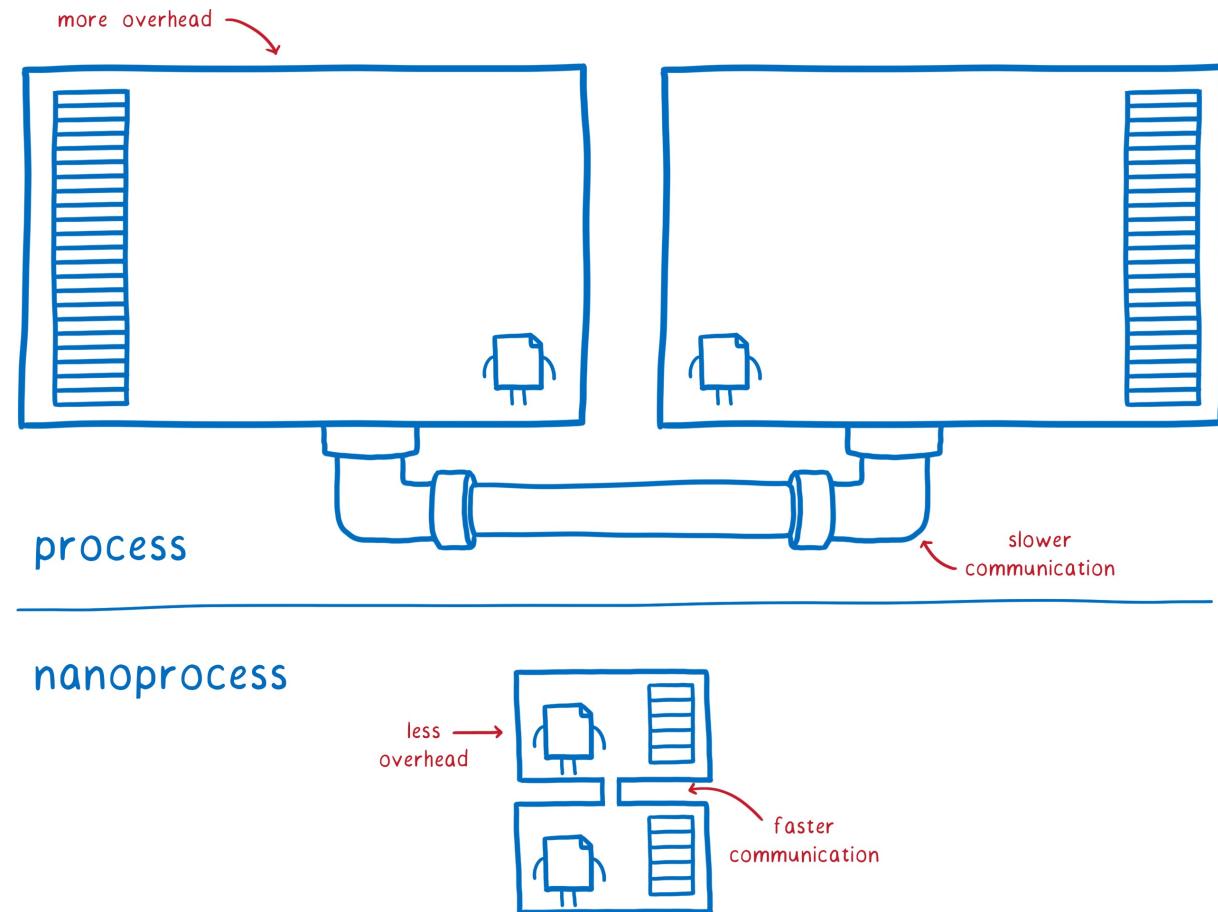
0101
0101

Process Isolation



0101
0101

WebAssembly Nano-Process



* not drawn to scale

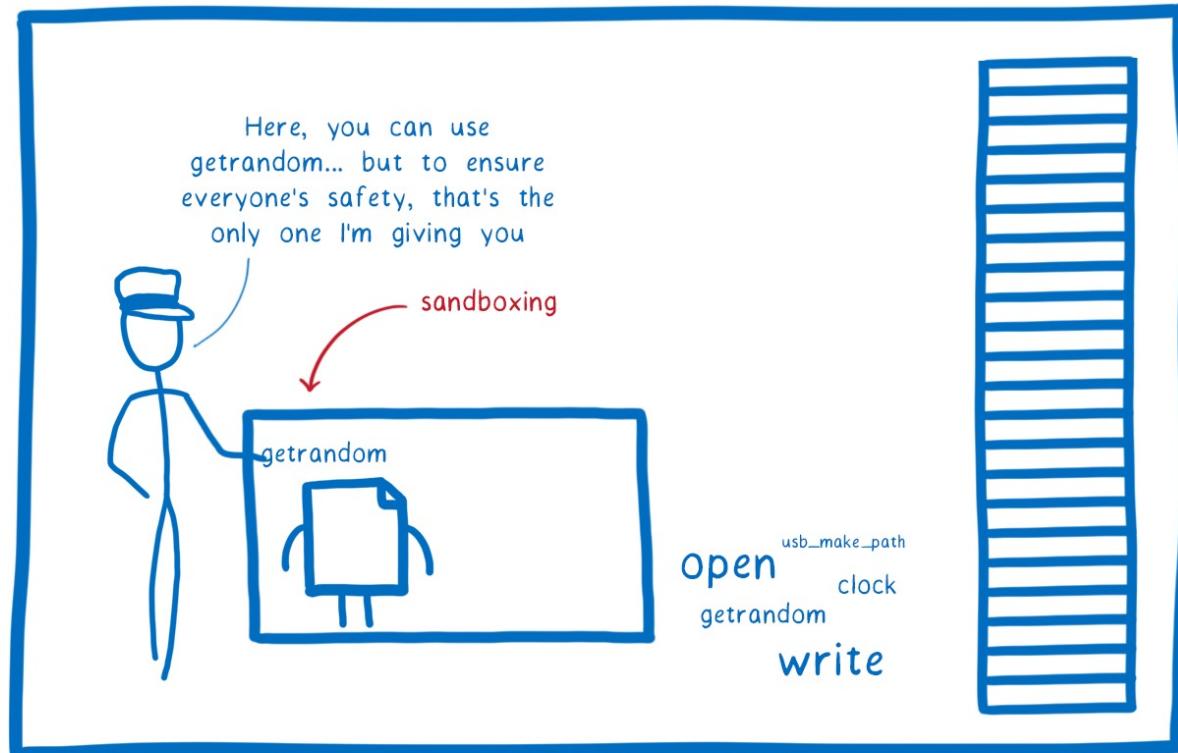


@nielstanis@infosec.exchange



WebAssembly Nano-Process

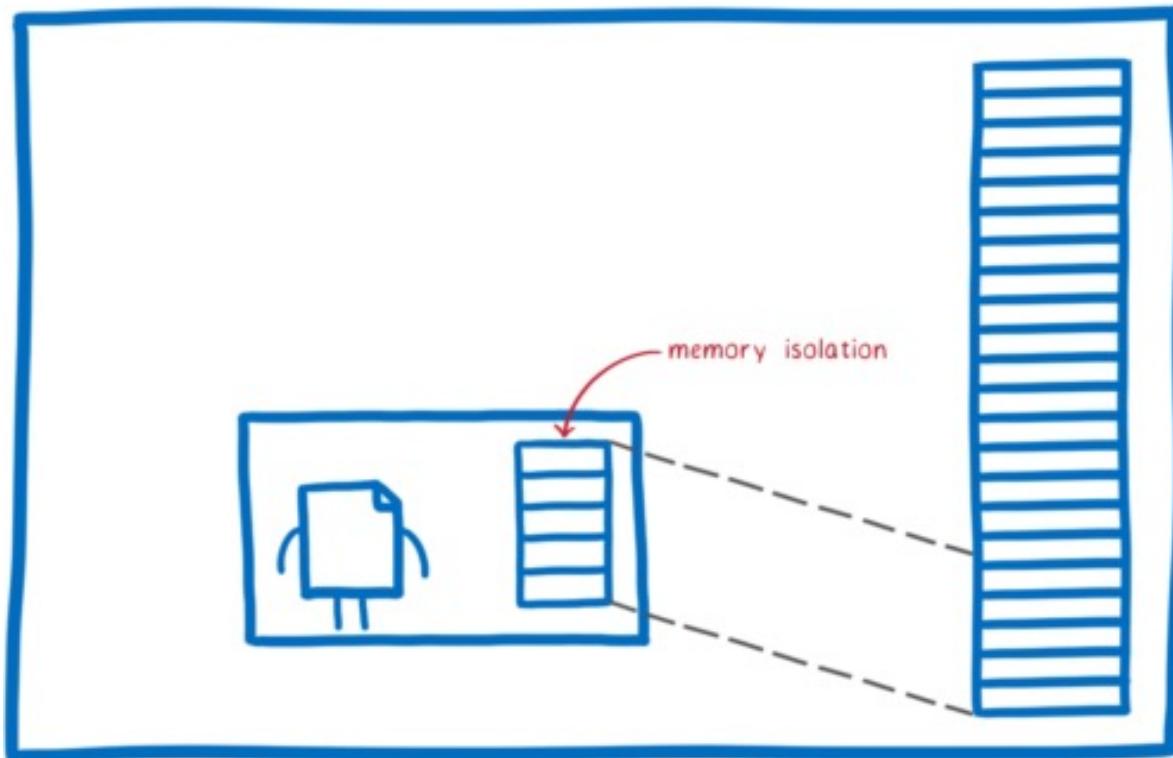
1. Sandboxing



0101
0101

WebAssembly Nano-Process

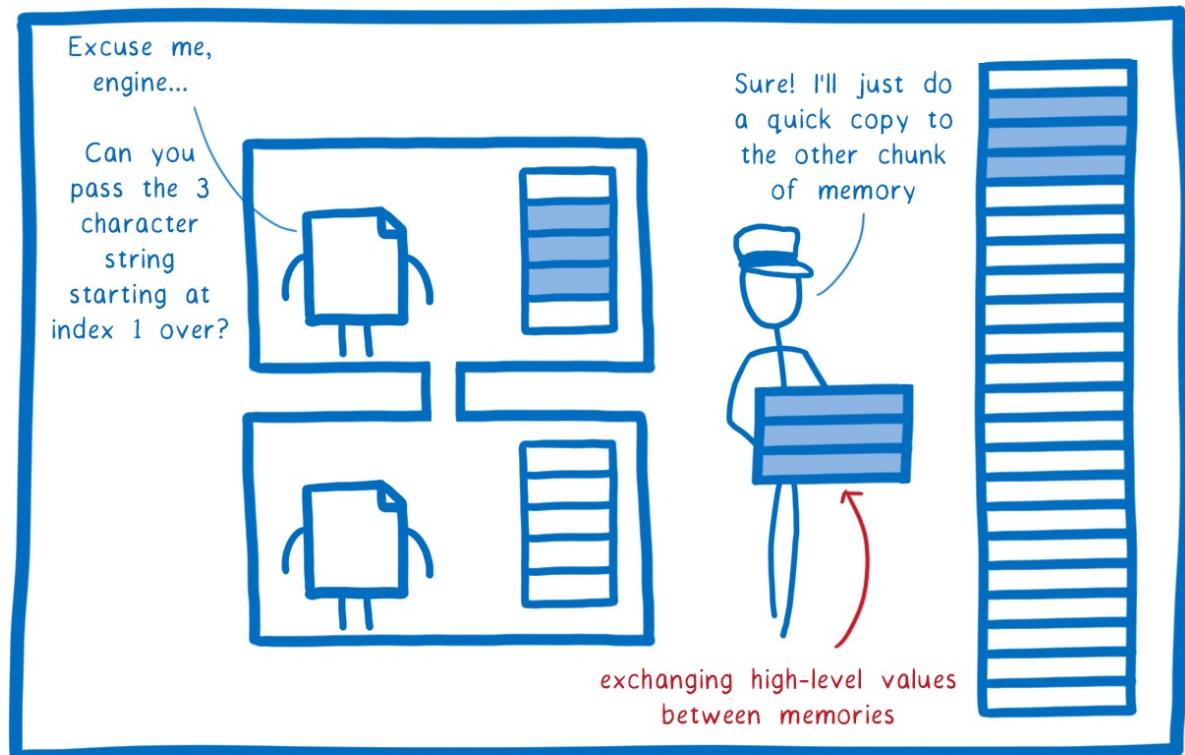
2. Memory model



0101
0101

WebAssembly Nano-Process

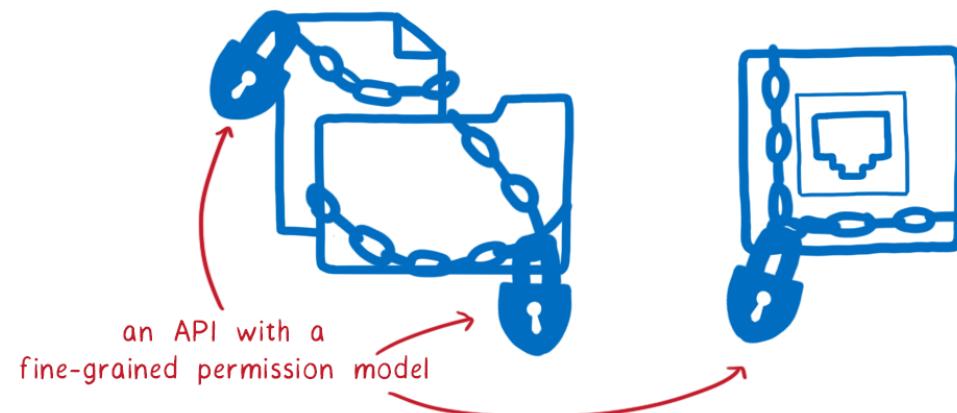
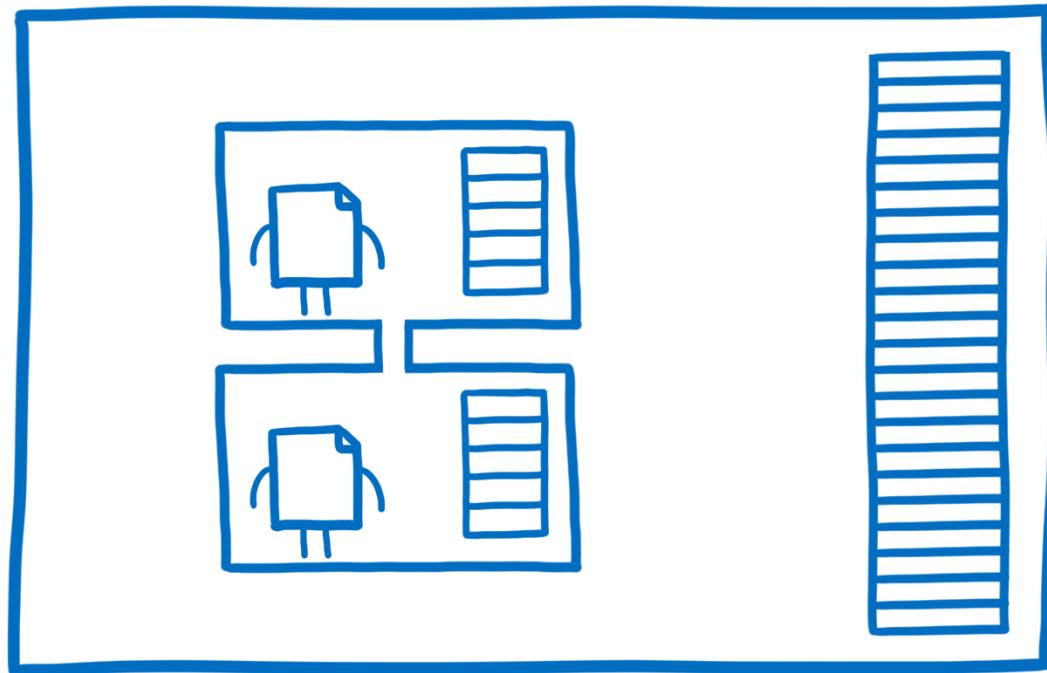
3. Interface Types



0101
0101

WebAssembly Nano-Process

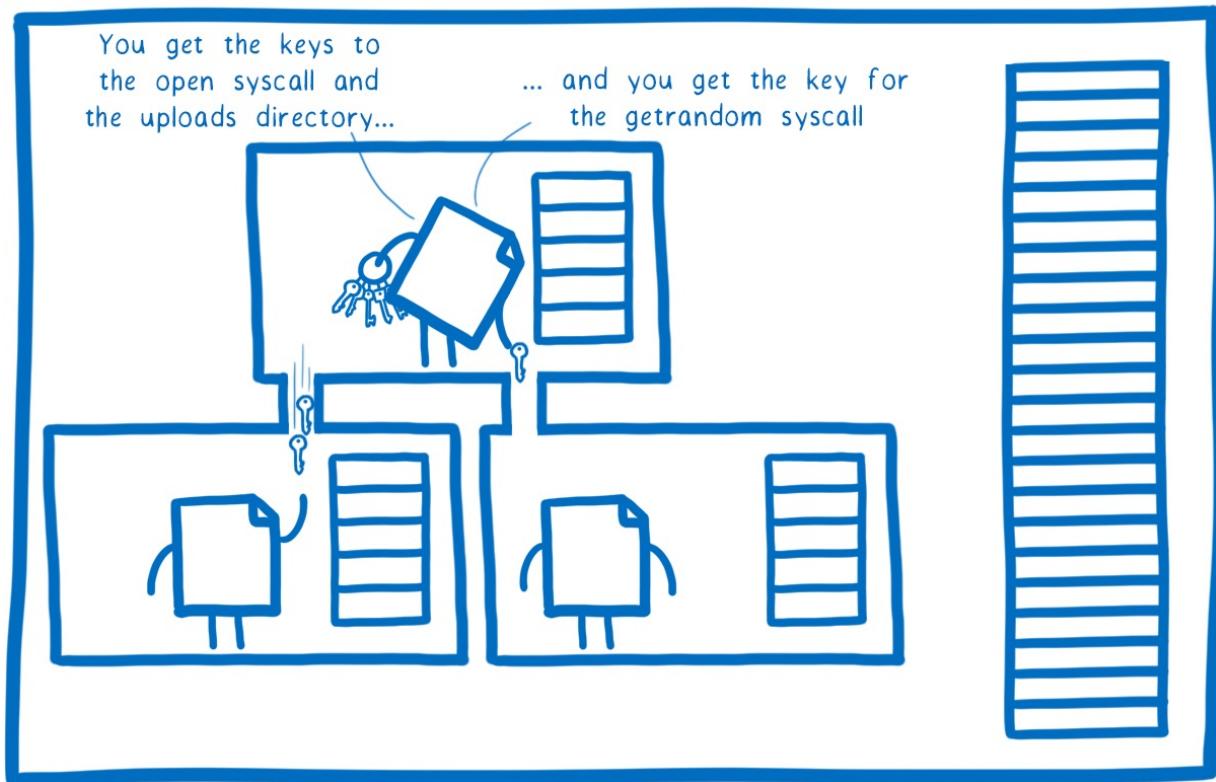
4. WebAssembly System Interface



0101
0101

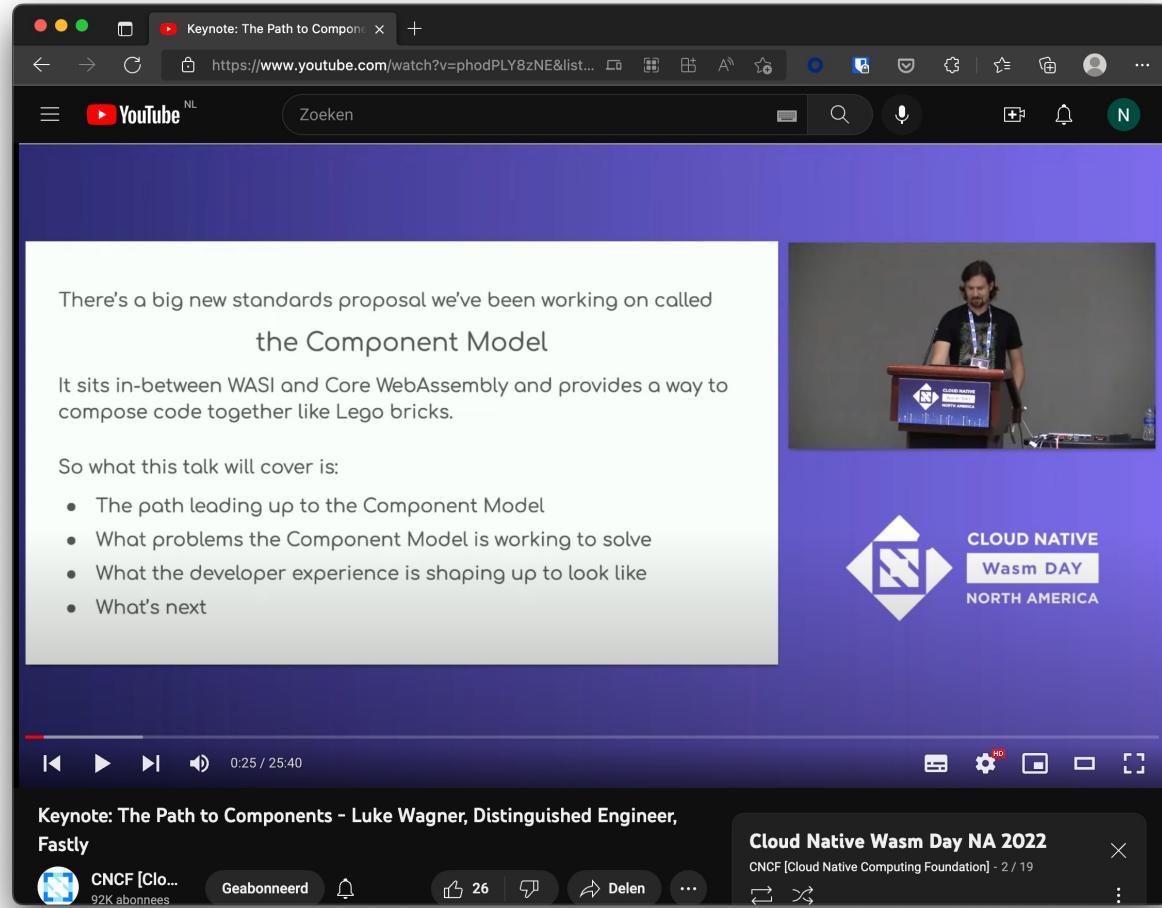
WebAssembly Nano-Process

5. The missing link



0101
0101

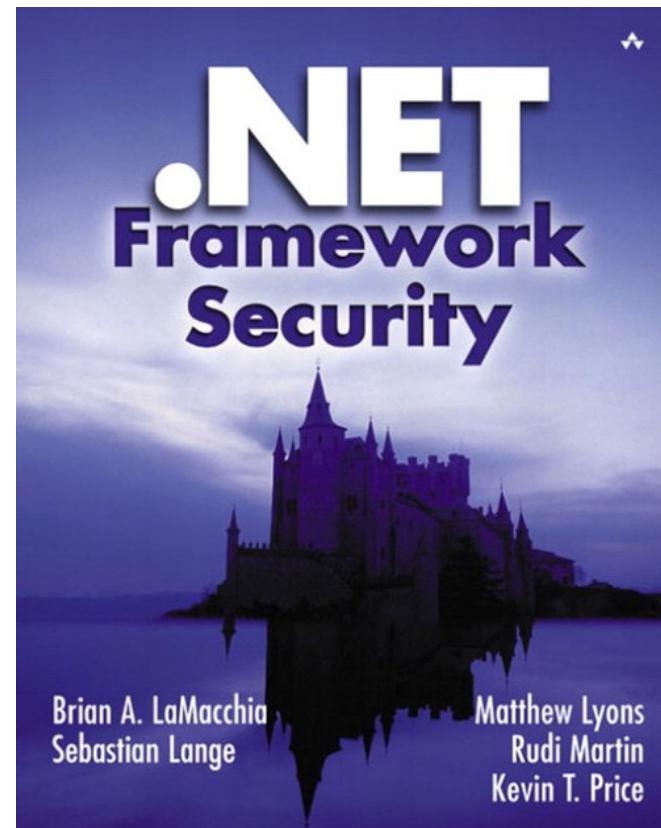
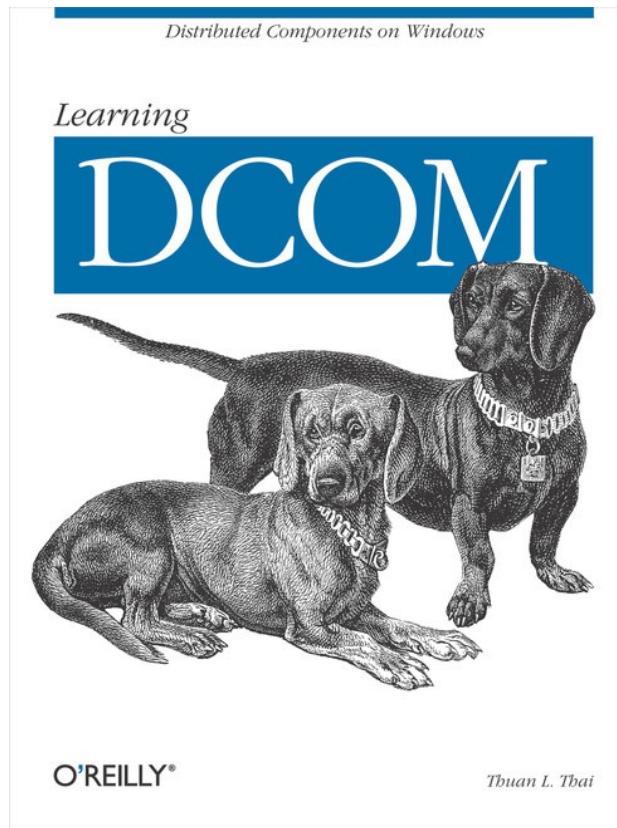
WebAssembly Component Model



@nielstanis@infosec.exchange

0101
0101

Have we seen this before?



@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





Conclusion

- Cloud Native ❤️ WebAssembly
- WebAssembly has a lot of potential to be used to run, extend, and secure your .NET 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



0101
0101

Questions?

- <https://github.com/nielstanis/futuretech2023>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Bedankt! Thank you!

