



Using WebAssembly to run, extend, and secure your .NET/Java/Python/Go/... application

Niels Tanis

0101
0101

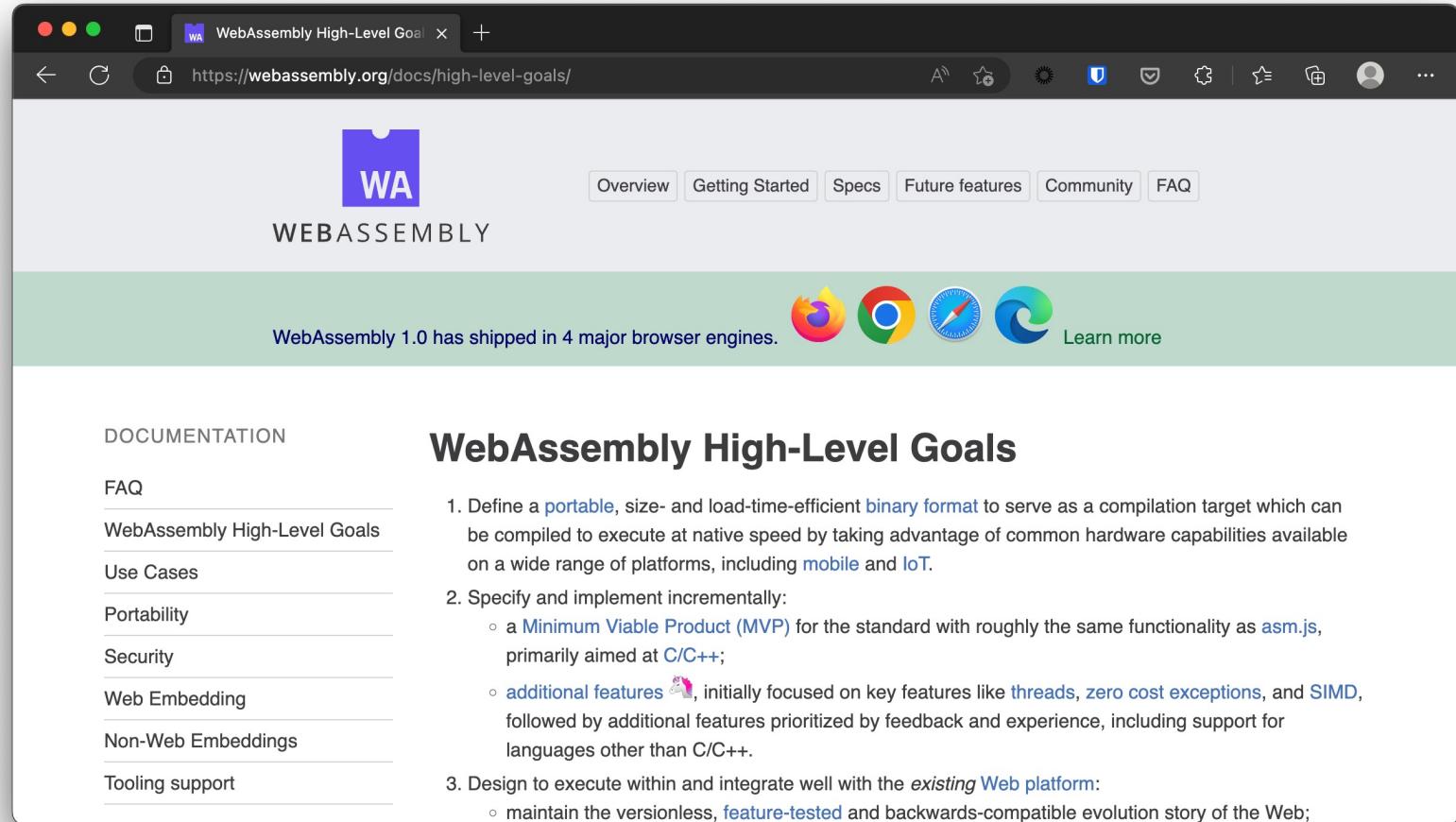
Who am I?

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



0101
0101

WebAssembly

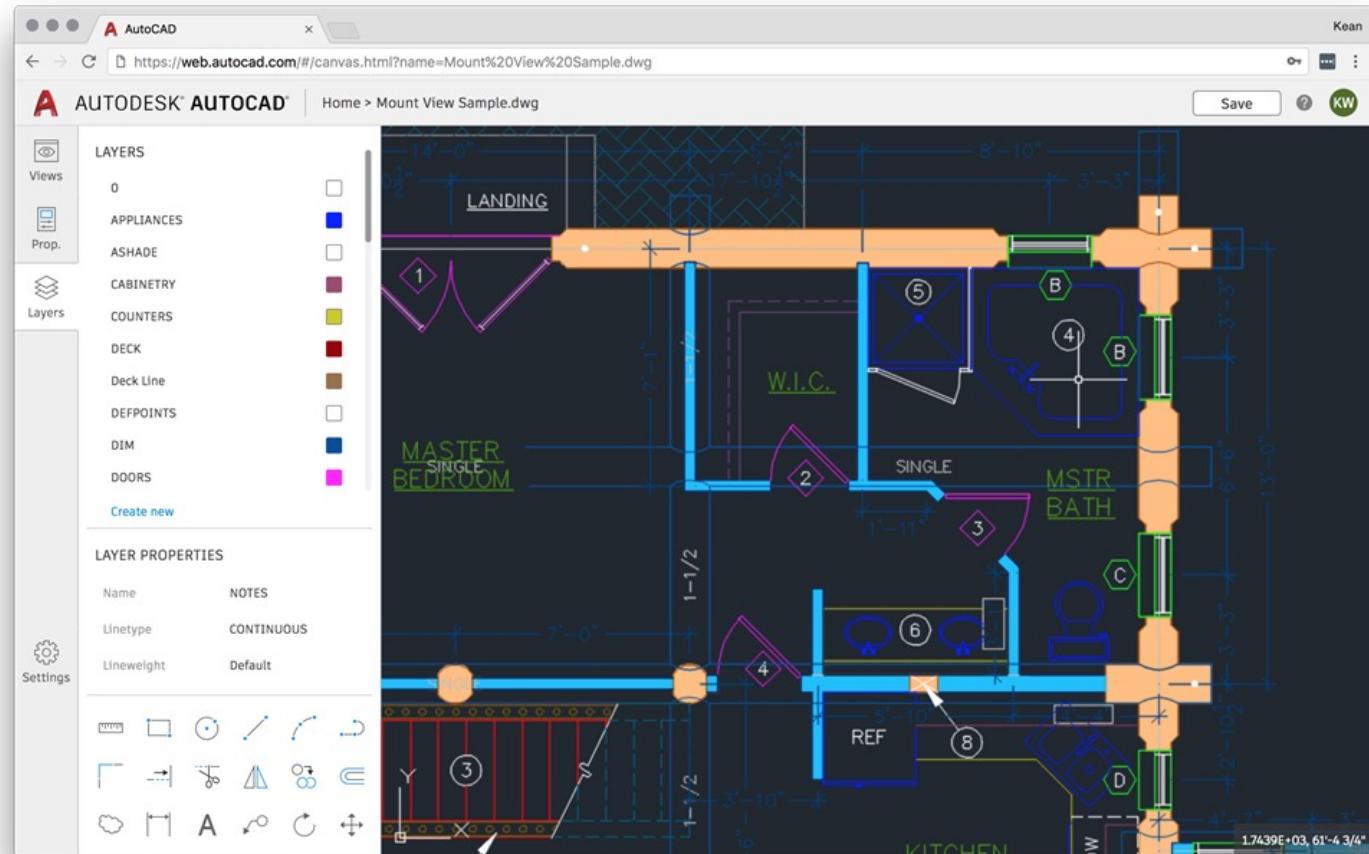


The screenshot shows a web browser window displaying the "WebAssembly High-Level Goals" page from the official website at <https://webassembly.org/docs/high-level-goals/>. The page features a purple header with the "WA" logo and the word "WEBASSEMBLY". Below the header, a green banner states "WebAssembly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



WebAssembly - AutoCAD



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.



Agenda

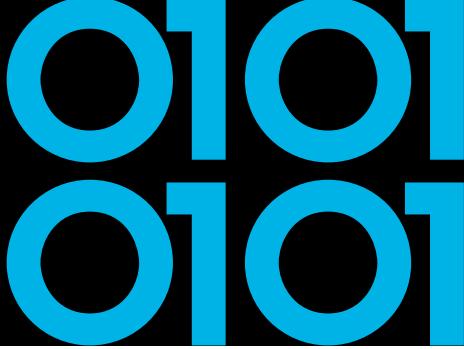
- Introduction
- WebAssembly 101
- Running .NET on WebAssembly
- Extending .NET with WebAssembly
- Securing .NET with WebAssembly
- Conclusion
- Q&A



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

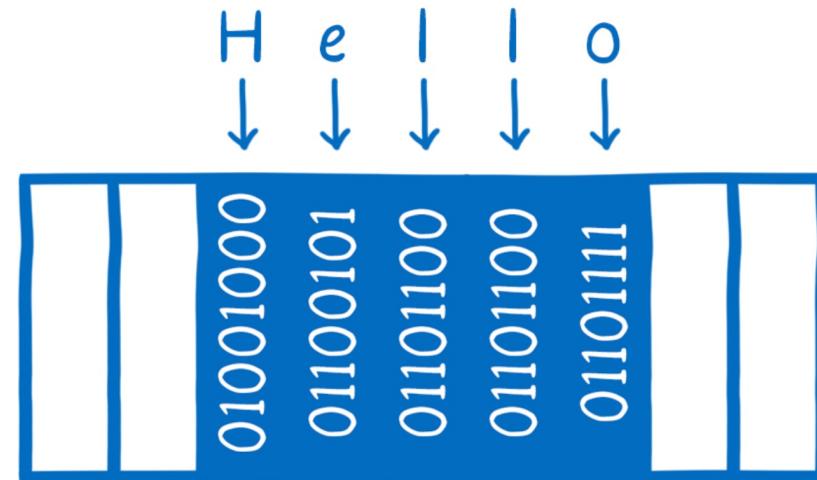
- Binary instruction format for stack-based virtual machine similar to .NET running MSIL
- Designed as a portable compilation target
- The security model of WebAssembly:
 - Protect users from buggy or malicious modules
 - Provide developers with useful primitives and mitigations for developing safe applications



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!");
```



WebAssembly Control-Flow Integrity

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

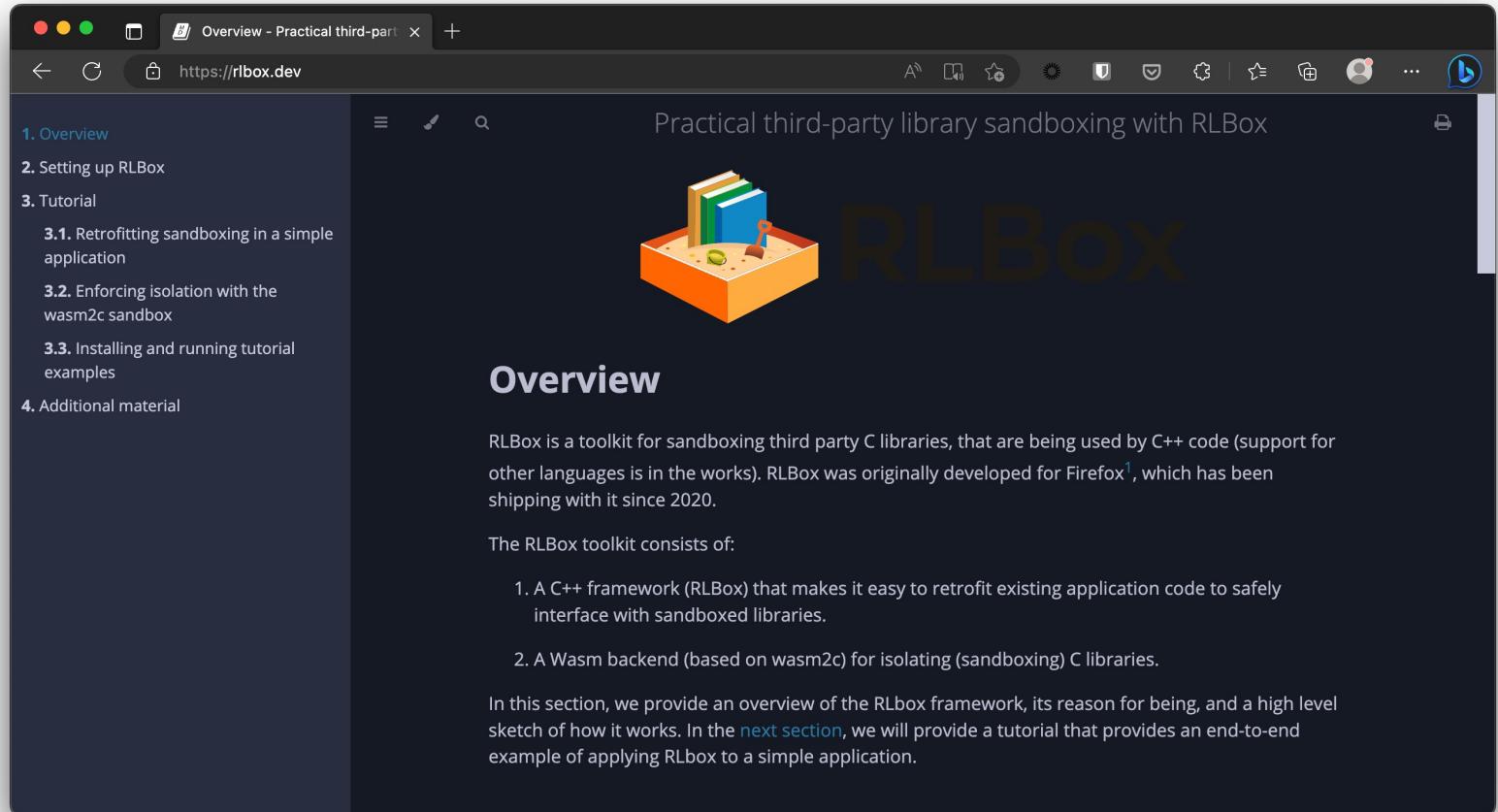
```
Console.WriteLine("Number is larger than 5");
```

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

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

0101
0101

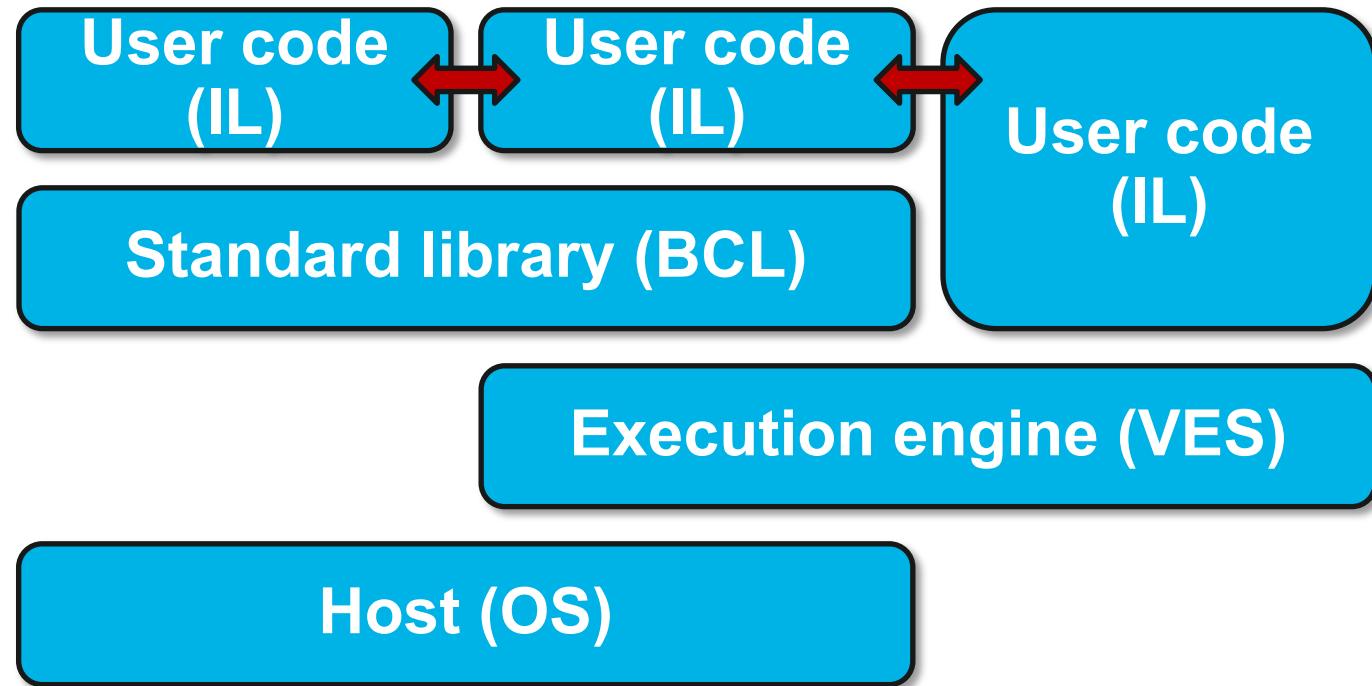
FireFox RLBox



The screenshot shows a Firefox browser window with the URL <https://rlbox.dev>. The page title is "Overview - Practical third-party library sandboxing with RLBox". The main content area features a large "RLBox" logo with a small illustration of books in a sandcastle. Below the logo, the word "Overview" is prominently displayed. The text explains that RLBox is a toolkit for sandboxing third party C libraries used by C++ code, originally developed for Firefox and shipped since 2020. It lists the toolkit's components: a C++ framework and a Wasm backend. A sidebar on the left contains a navigation menu with sections: 1. Overview, 2. Setting up RLBox, 3. Tutorial (which is expanded to show 3.1, 3.2, and 3.3), and 4. Additional material.

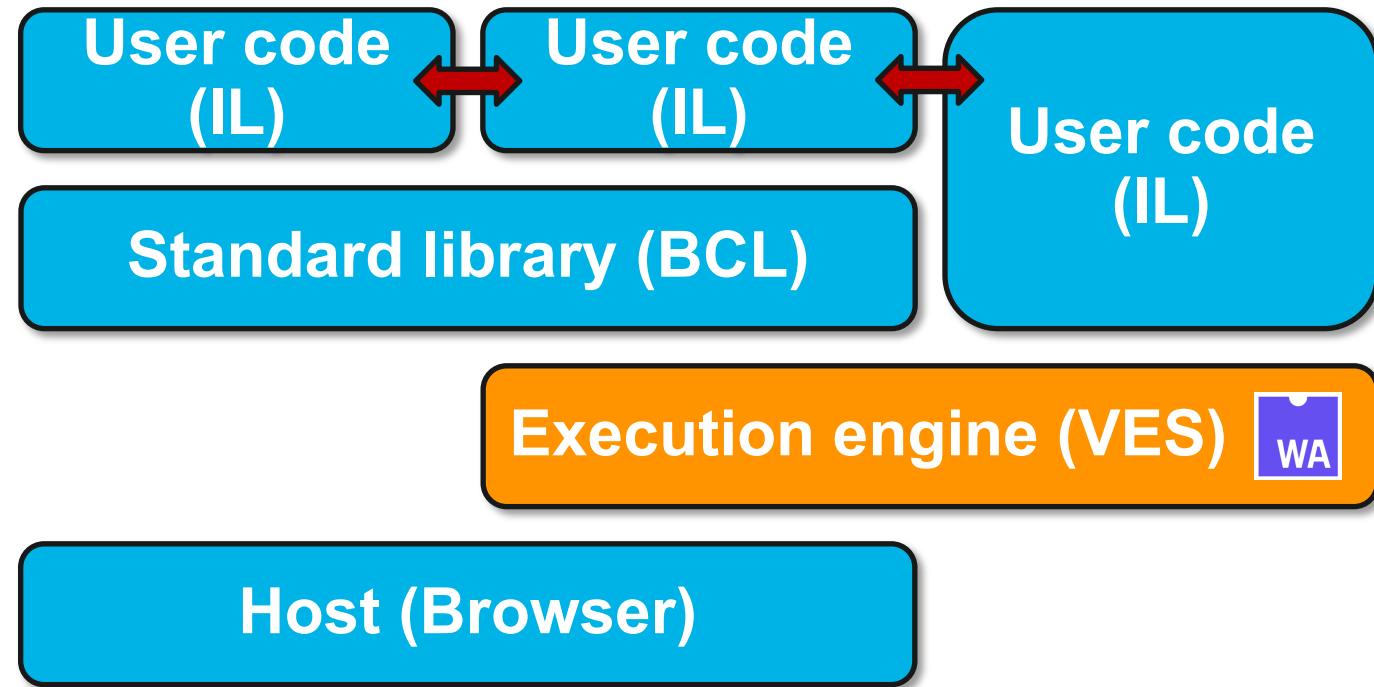


Running .NET on WebAssembly



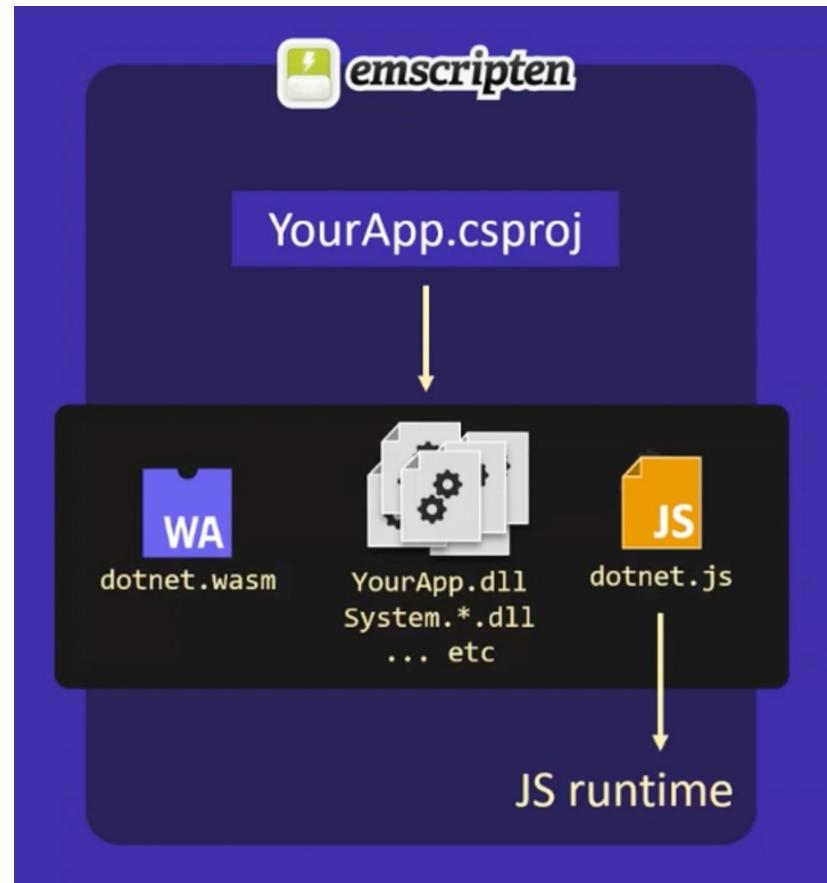


Running .NET on WebAssembly



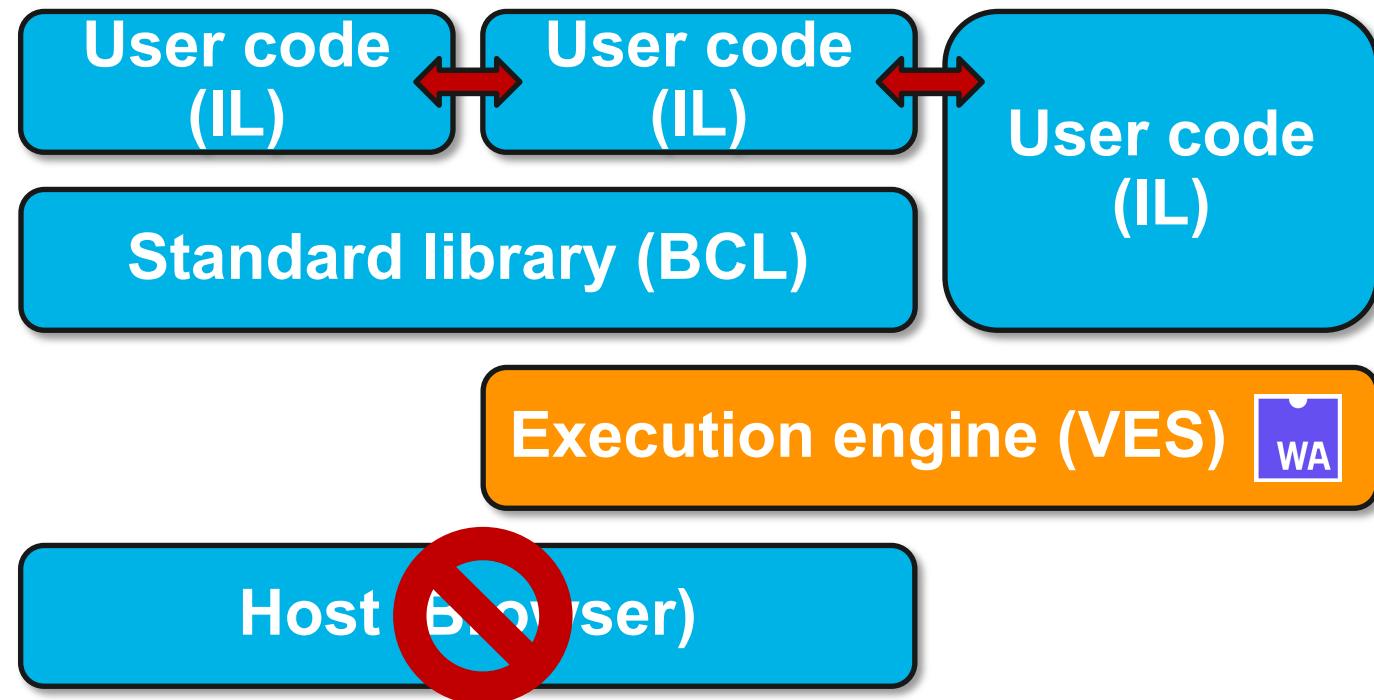
0101
0101

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 mobile interface. The top bar shows the URL <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 are icons for reply, retweet, and like.

0101
0101

Docker vs WASM & WASI

A screenshot of a Twitter browser window showing a tweet from Solomon Hykes (@solomonstre). The tweet discusses the future of containerization, mentioning Docker, WASM, and WASI. The tweet text is as follows:

“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 :)

The tweet was posted on March 27, 2019. A reply below the main tweet reads:

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker.
That's how important it is. WebAssembly is the answer in the future of

0101
0101

Docker & WASM

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 container instances. Each instance consists of a "containerd-shim" layer, a "runc" layer, and a "Container process". The third instance is specifically labeled with "containerd-wasm-shim", "wasmedge", and "Wasm Module". Arrows indicate the flow of communication between these layers.

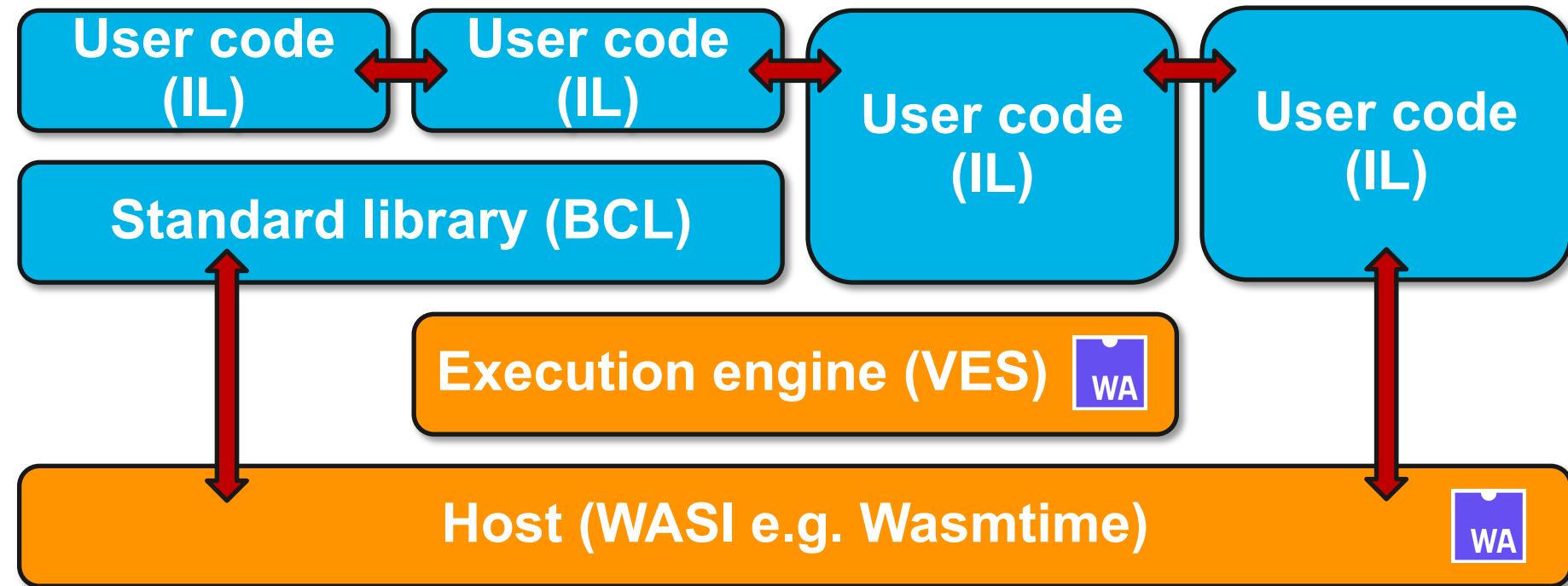
Let's look at an example!

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
```

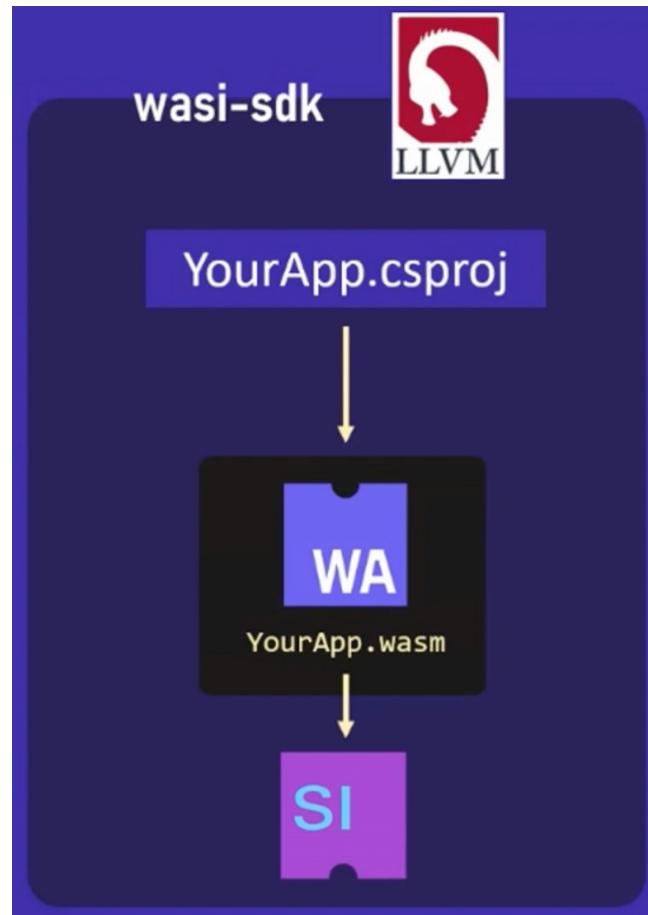


WebAssembly System Interface WASI



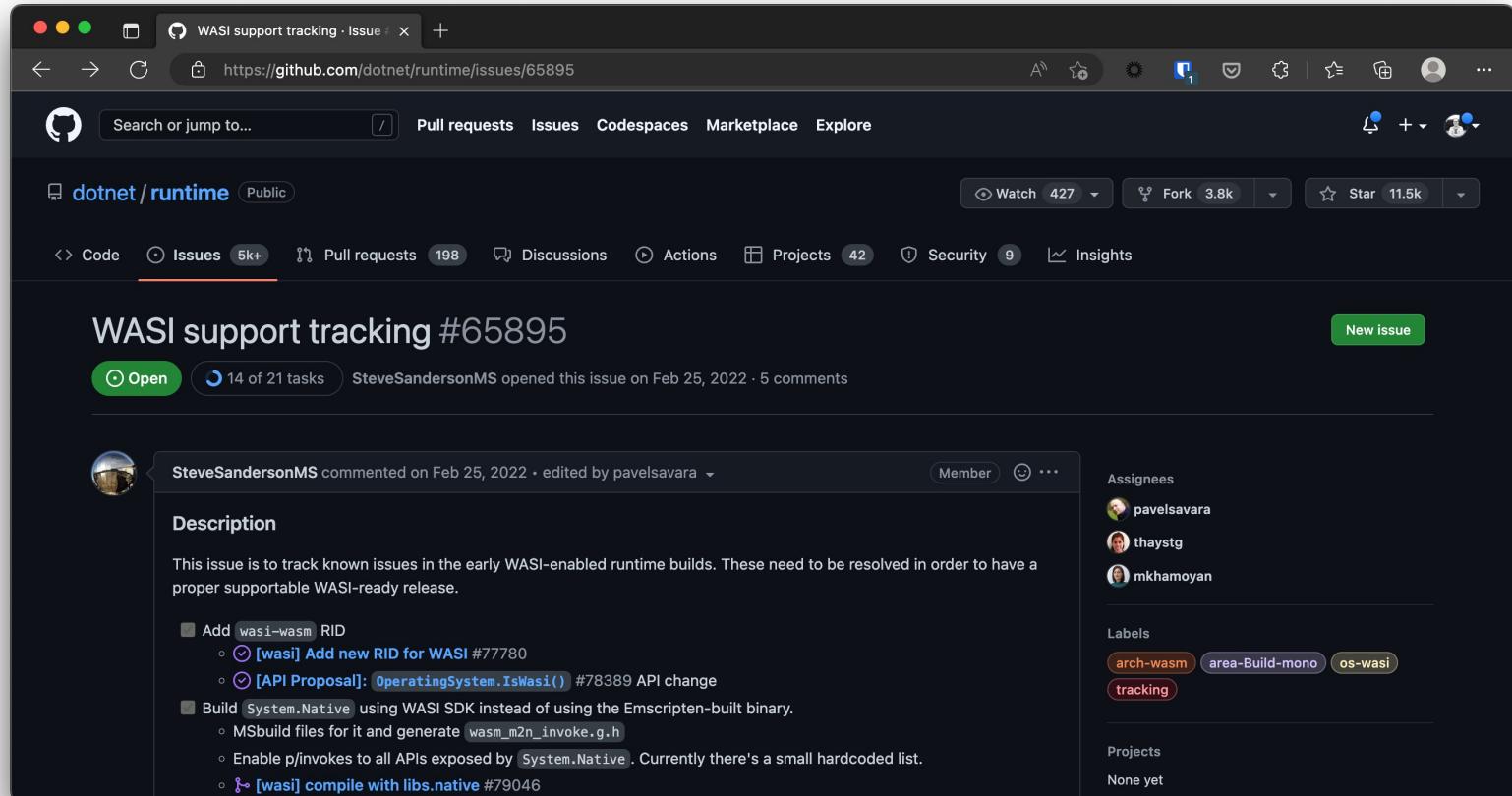
0101
0101

Experimental WASI SDK for .NET



0101
0101

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
- Demo time!

0101
0101

DotNetIsolator

A screenshot of a GitHub repository page for 'DotNetIsolator'. The page shows the README.md file. The title 'DotNetIsolator [EXPERIMENTAL]' is displayed prominently. Below it, the text 'Lets your .NET code run other .NET code in an isolated environment easily.' is visible. A numbered list at the bottom begins with '1. Create as many `IsolatedRuntime` instances as you like.'

SteveSandersonMS / DotNetIsolator Public

Code Issues 5 Pull requests Discussions Actions Projects Security Insights

main DotNetIsolator / README.md

SteveSandersonMS Mechanism for guest to call host ✓ 805563e · last month History

Preview Code Blame 267 lines (186 loc) · 10.8 KB Raw ⌂ ⌄ ⌅ ⌆

DotNetIsolator [EXPERIMENTAL]

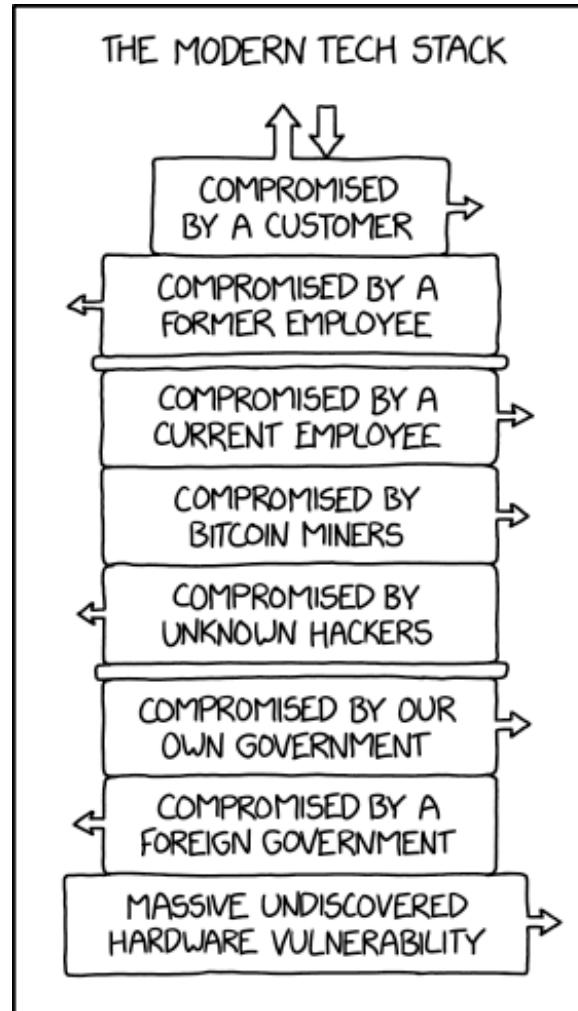
Lets your .NET code run other .NET code in an isolated environment easily.

Basic concept:

1. Create as many `IsolatedRuntime` instances as you like.

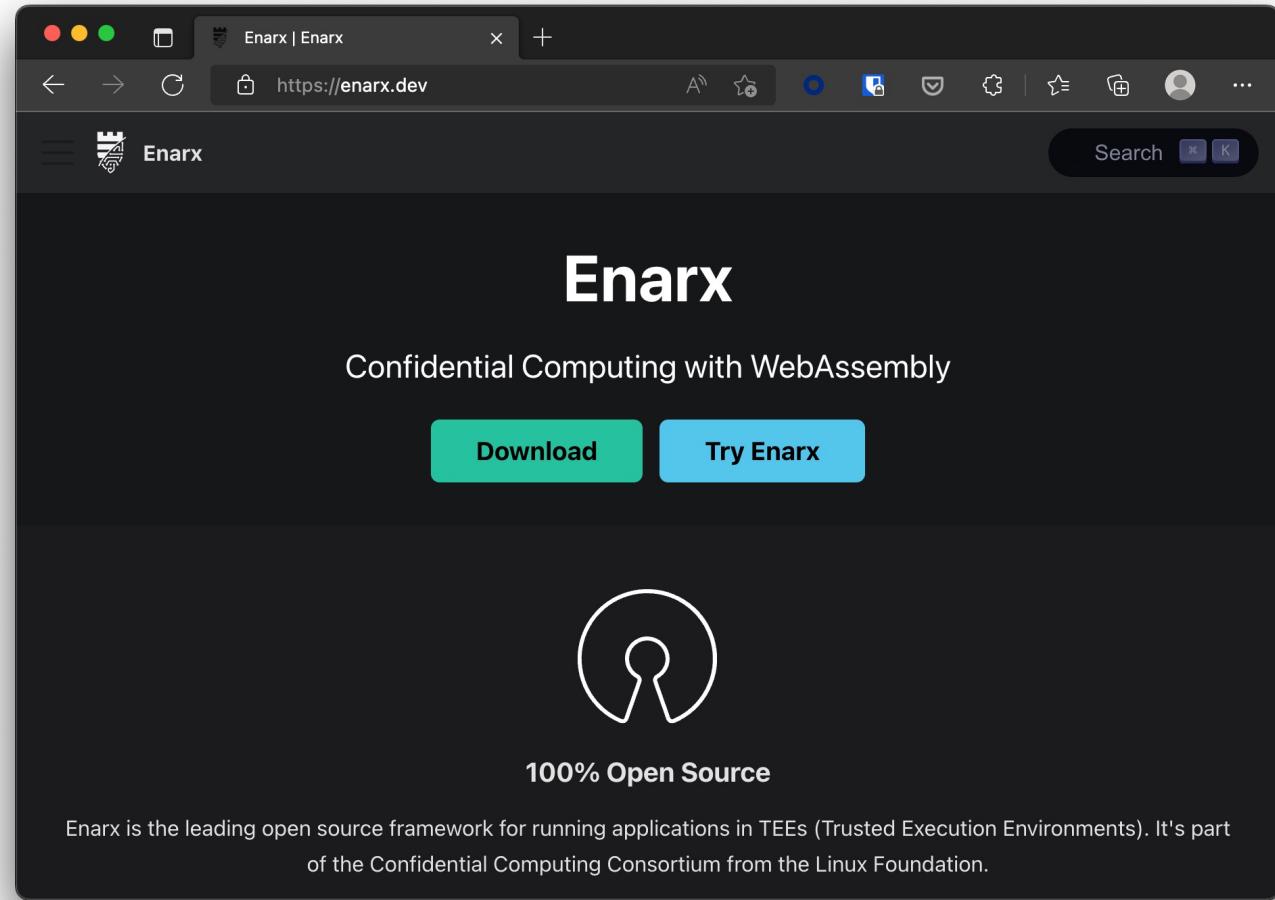
0101
0101

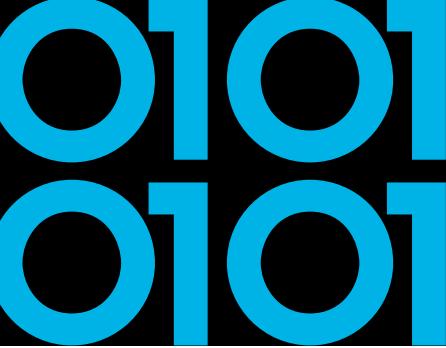
Trusted Computing - XKCD 2166



0101
0101

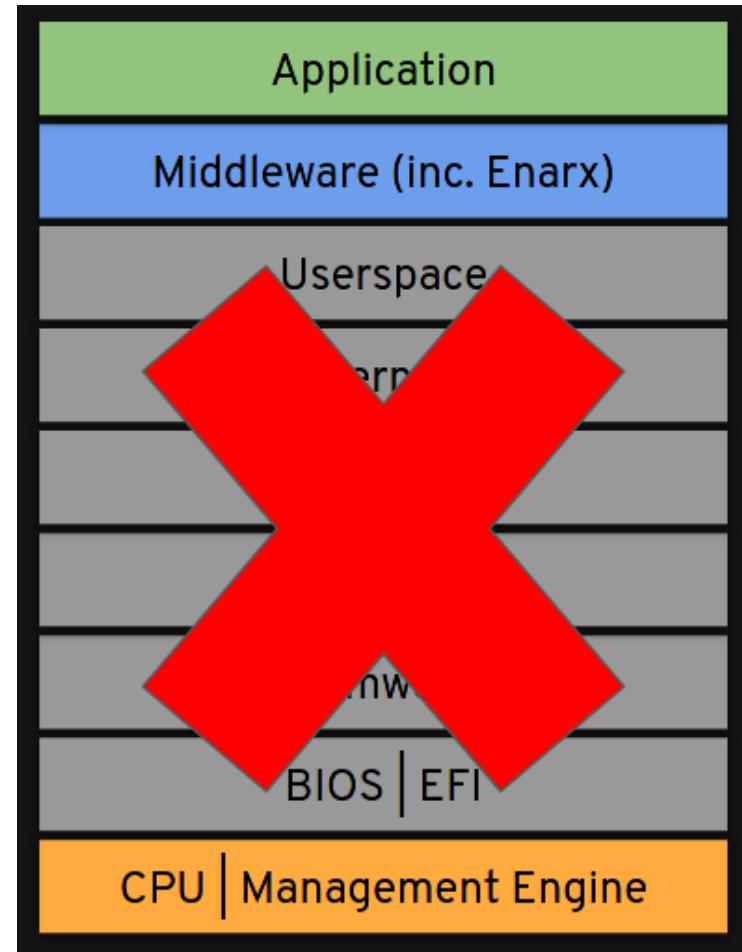
Enarx





Enarx Threat Model

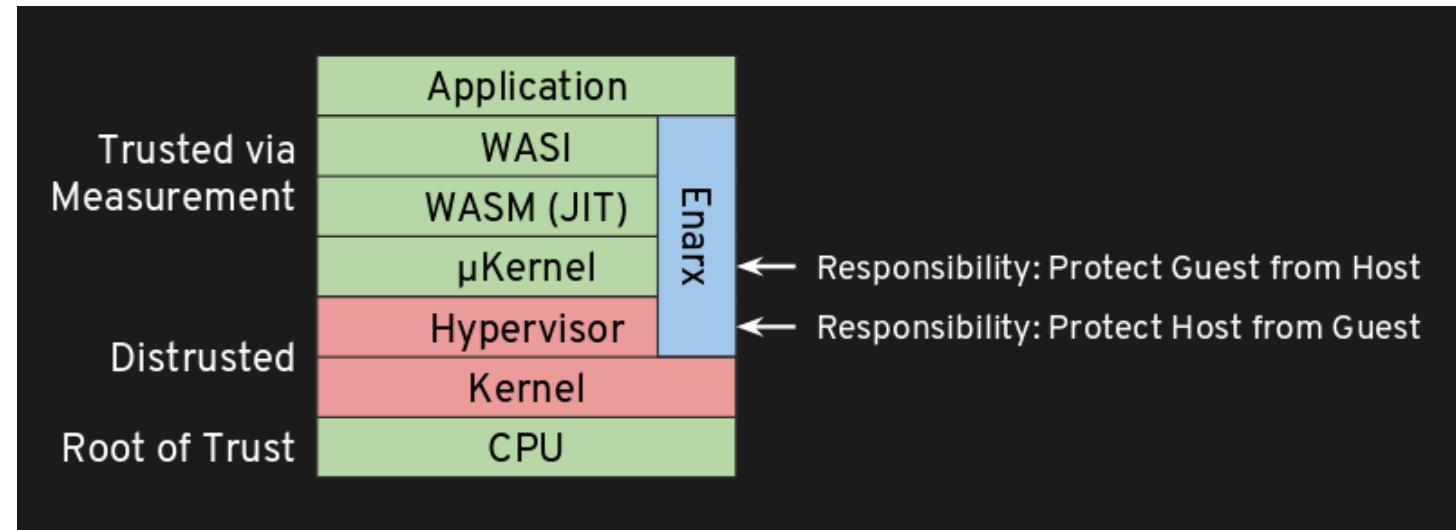
- Don't trust the host
- Don't trust the host owner
- Don't trust the host operator
- Hardware cryptographically verified
- Software audited and cryptographically verified





Enarx

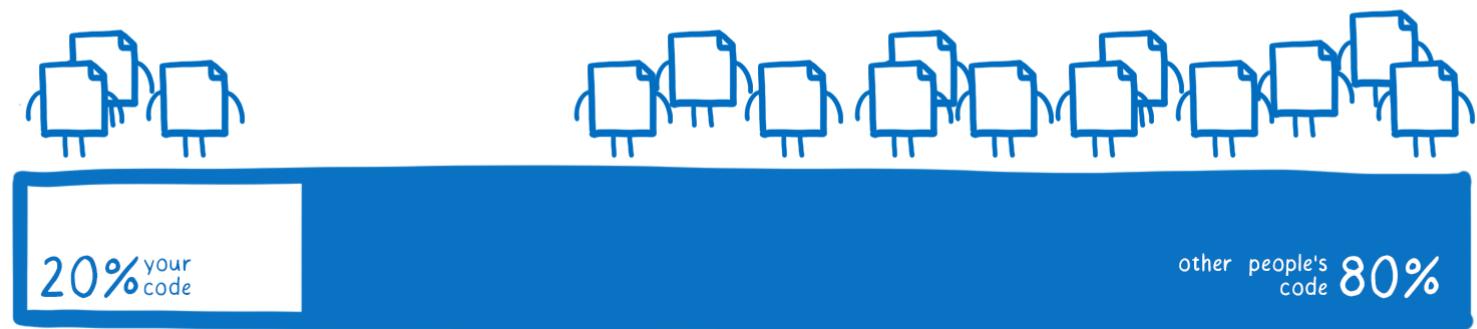
- Leverages Trusted Execution Environment (TEE) direct on processor
 - AMD's SEV, Intel's SGX and IBM's PEF
- Attestation of hardware and Enarx runtime



0101
0101

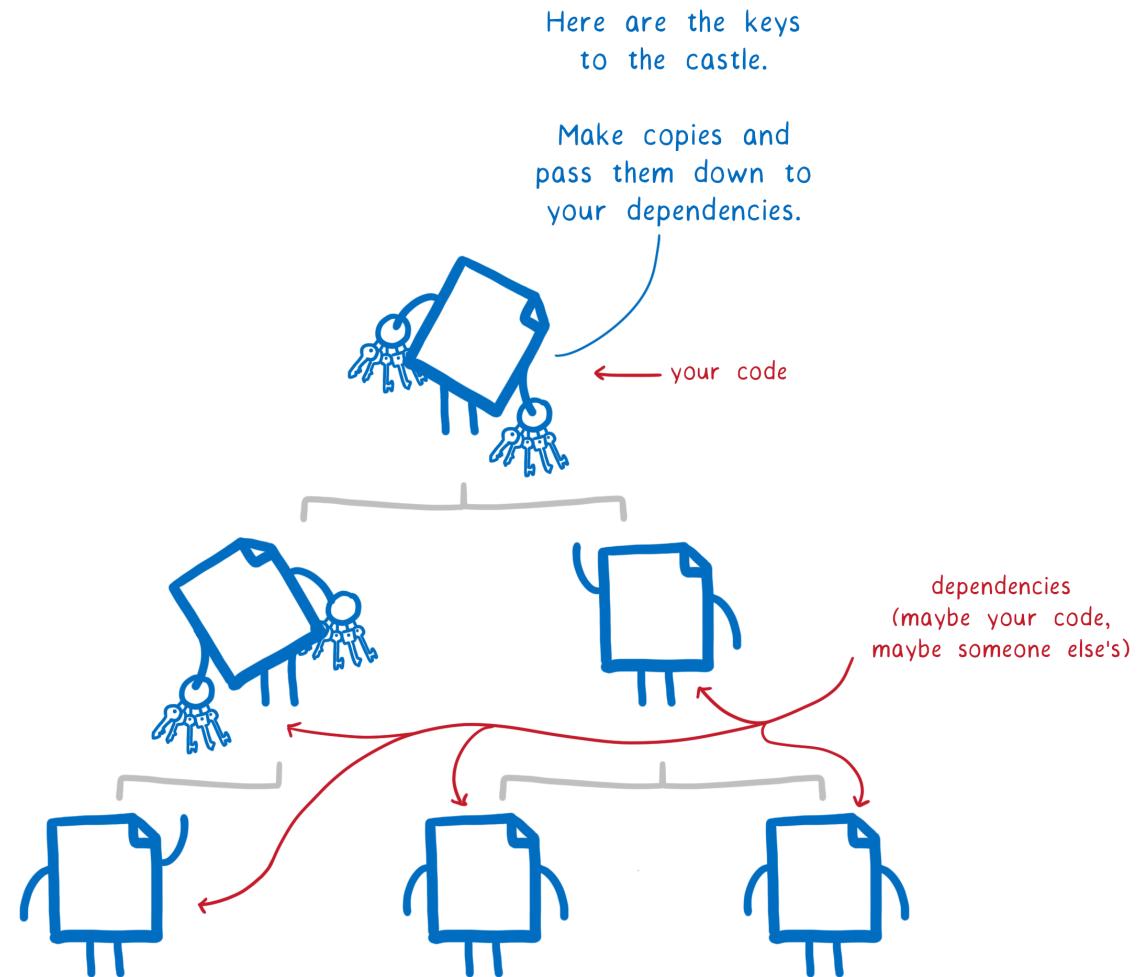
WASM - What's next?

composition of an
average code base



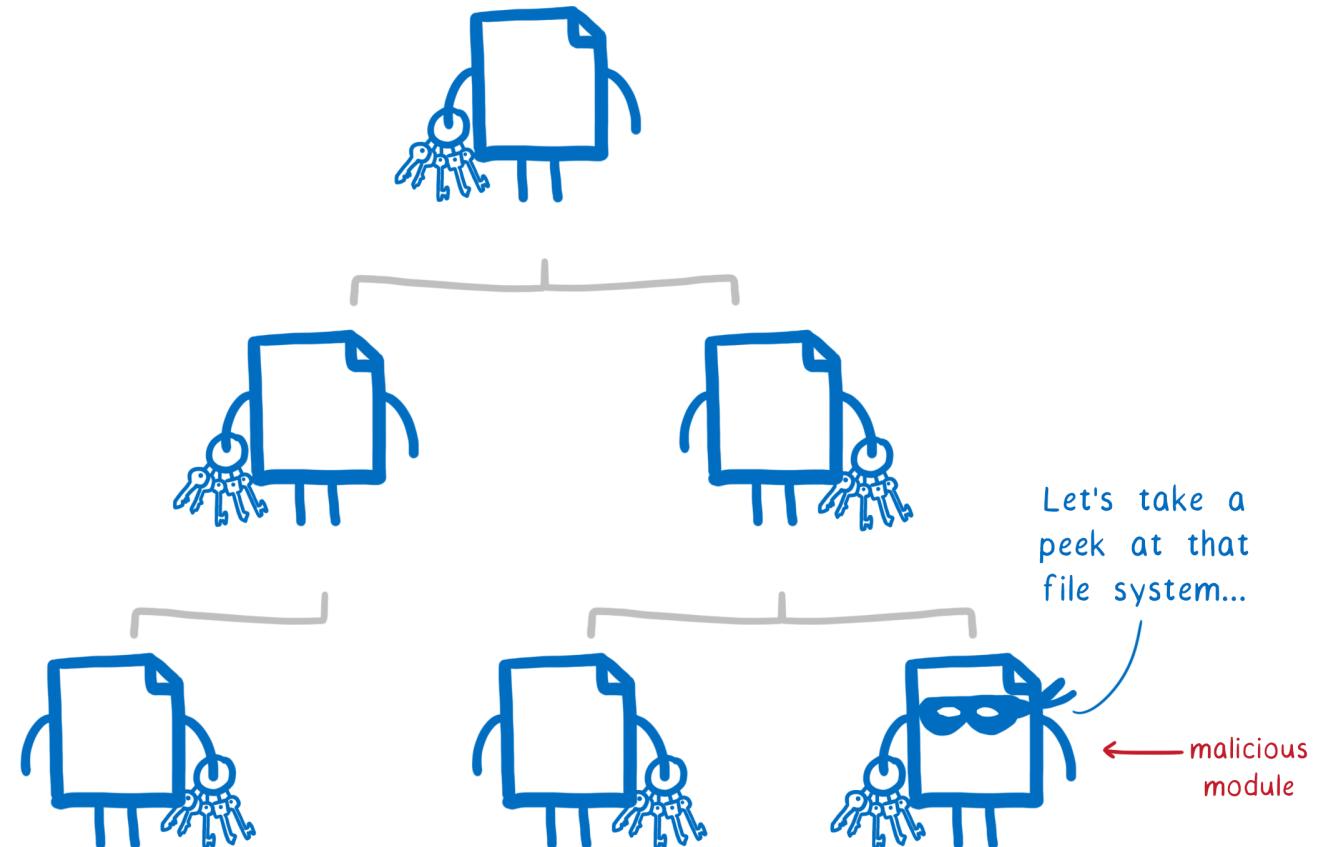
0101
0101

Dependencies



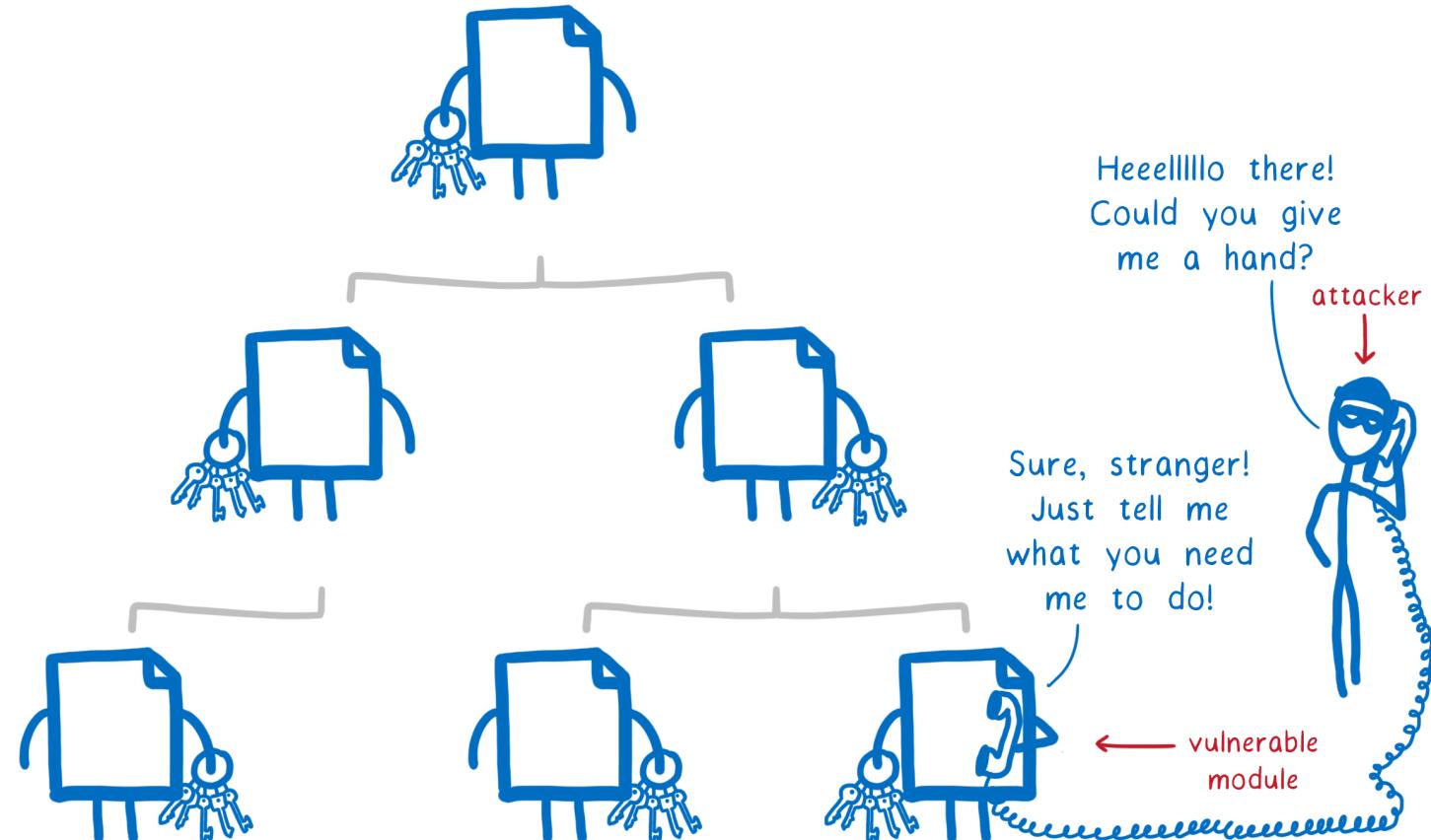
0101
0101

Malicious module



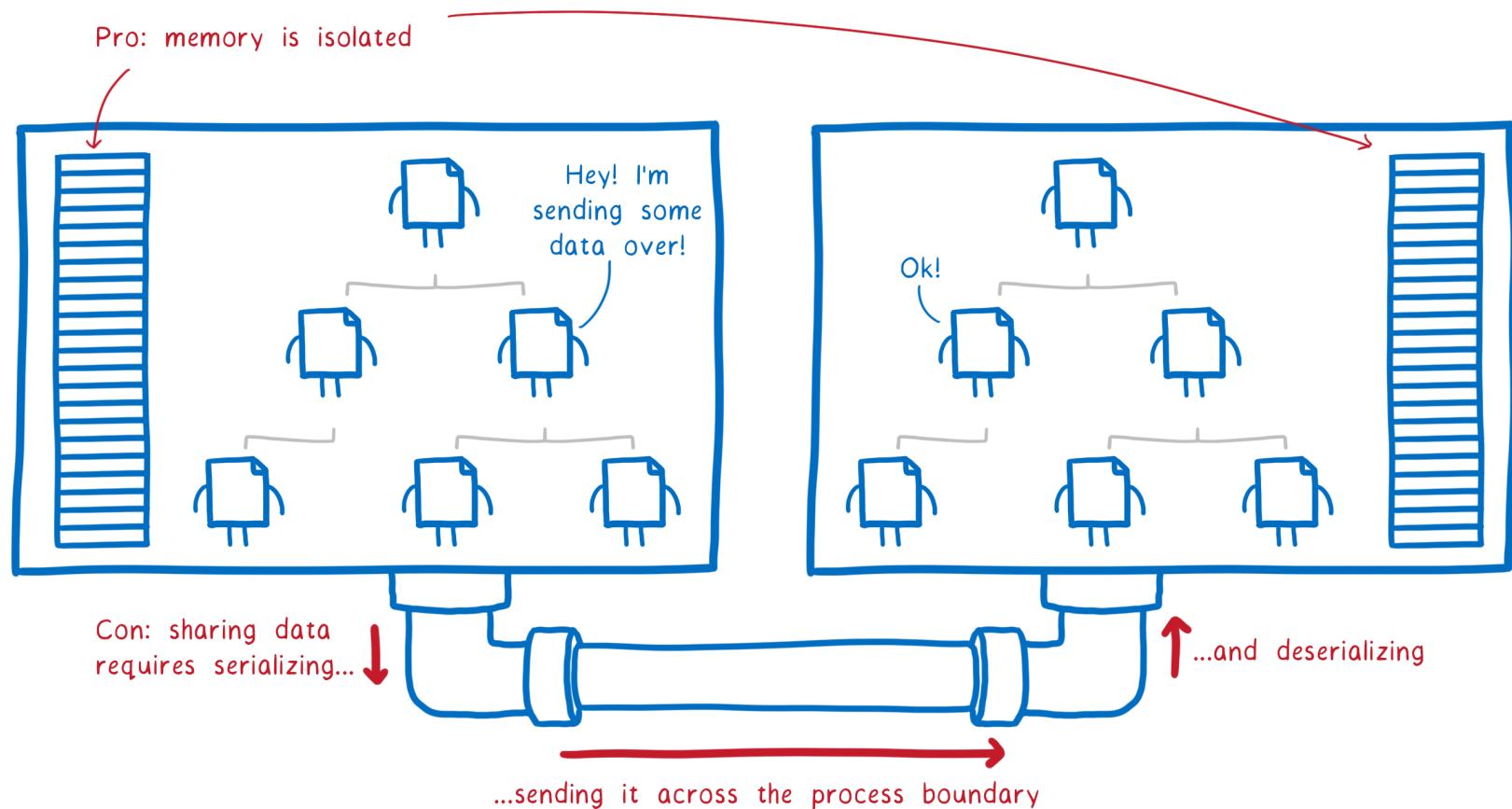
0101
0101

Vulnerable module



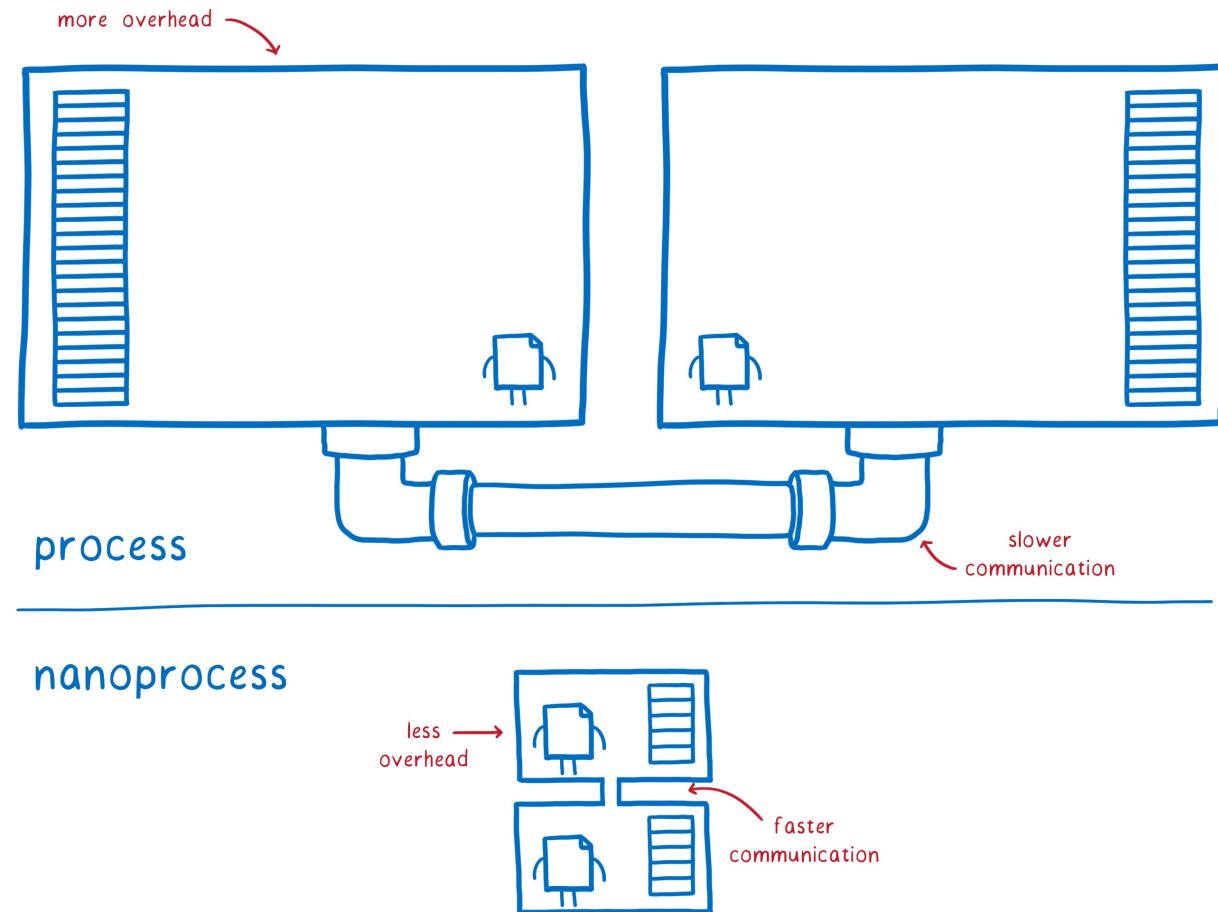
0101
0101

Process Isolation



0101
0101

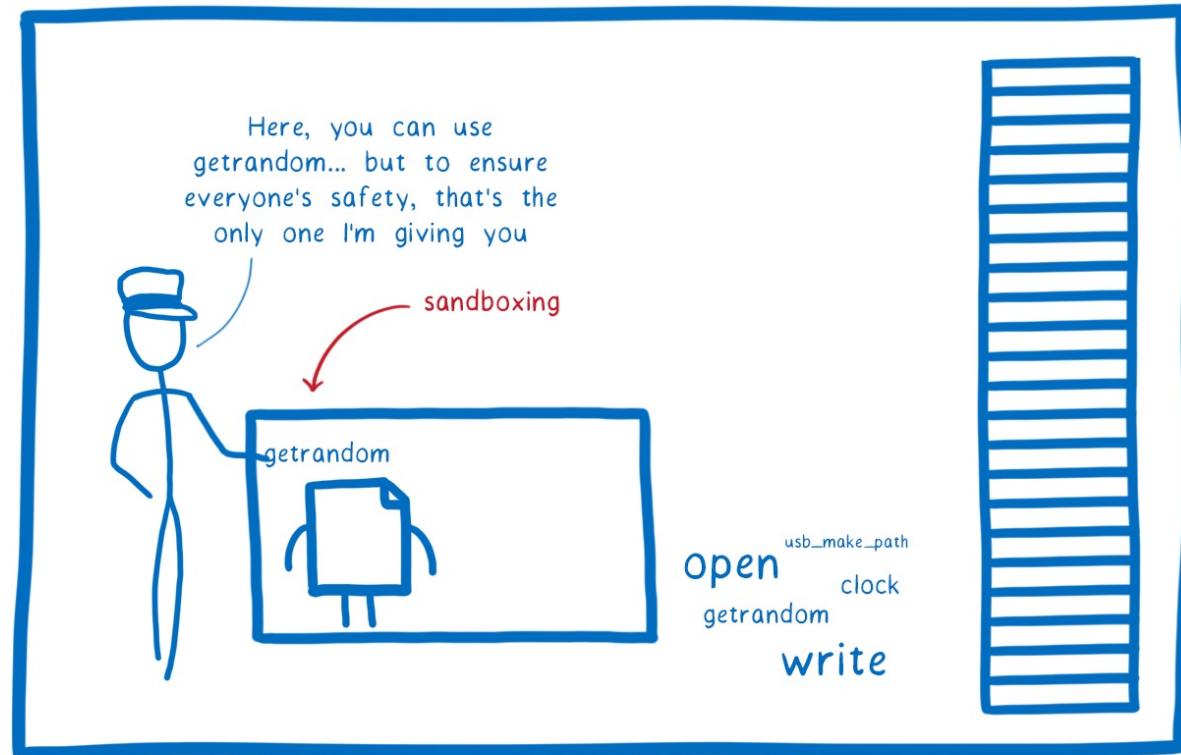
WebAssembly Nano-Process



0101
0101

WebAssembly Nano-Process

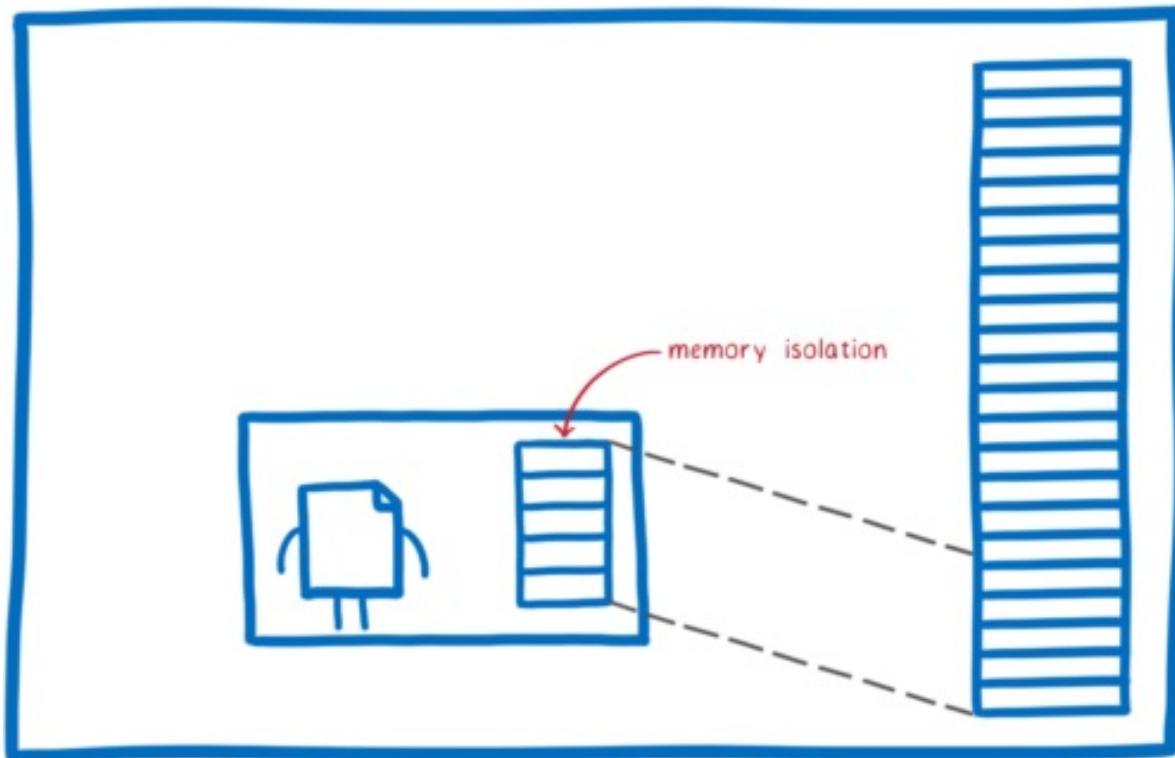
1. Sandboxing



0101
0101

WebAssembly Nano-Process

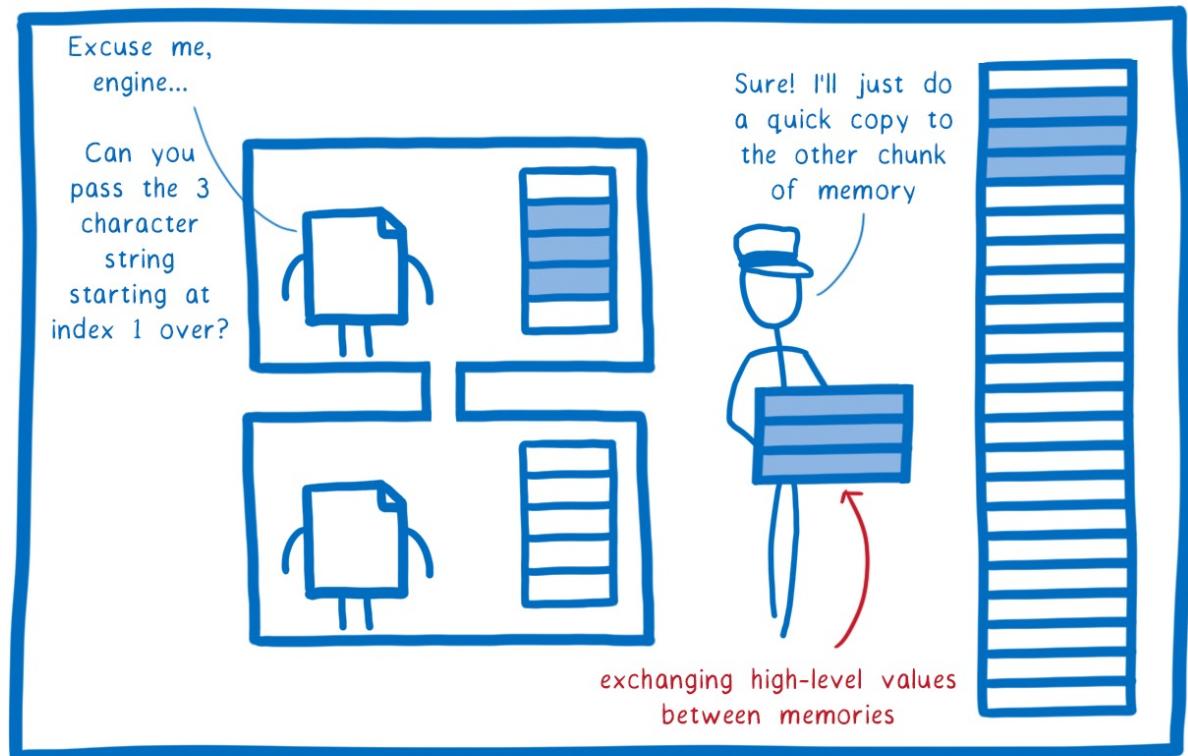
2. Memory model



0101
0101

WebAssembly Nano-Process

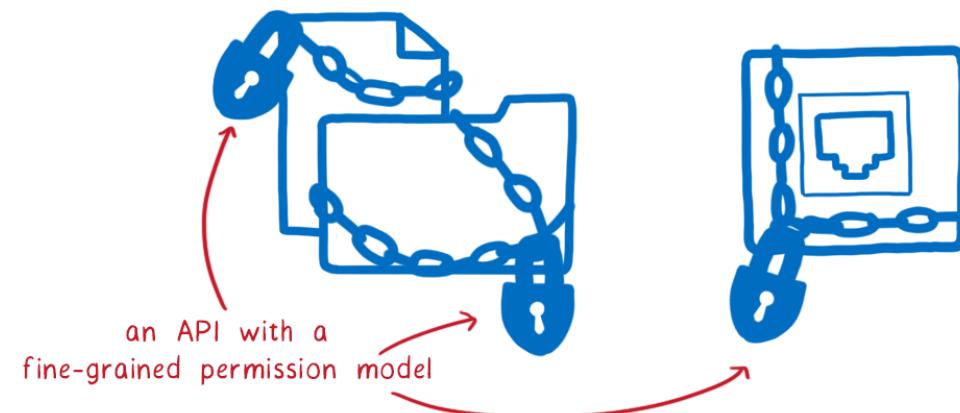
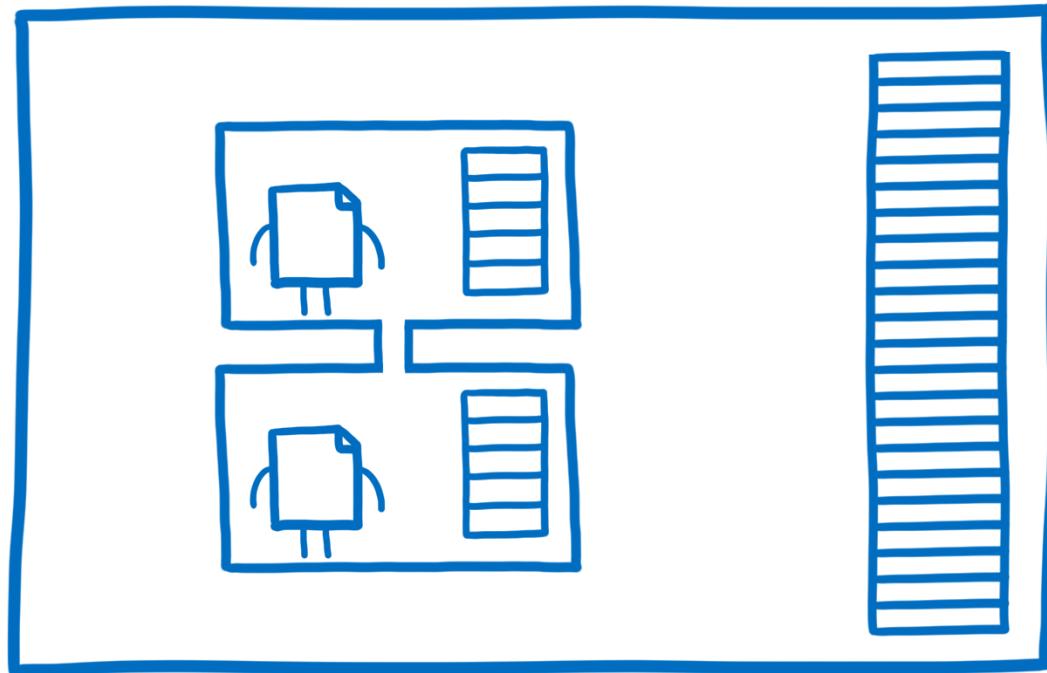
3. Interface Types



0101
0101

WebAssembly Nano-Process

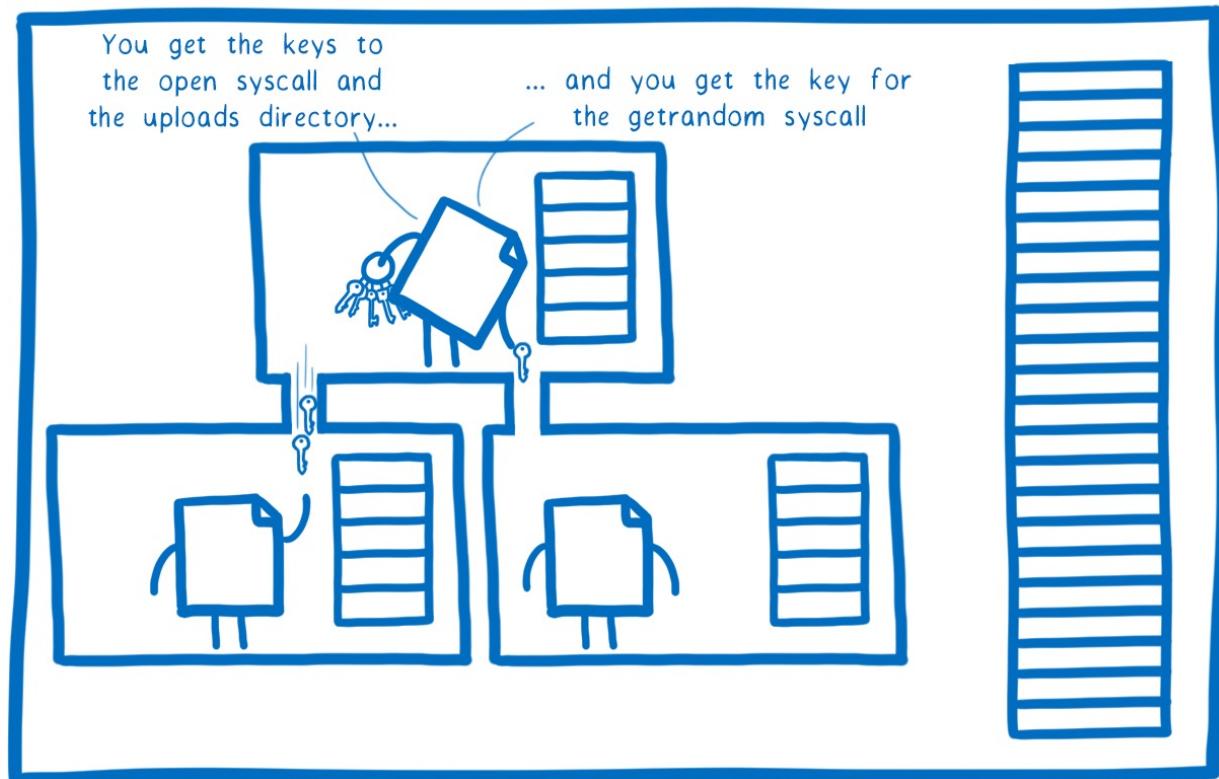
4. WebAssembly System Interface



0101
0101

WebAssembly Nano-Process

5. The missing link



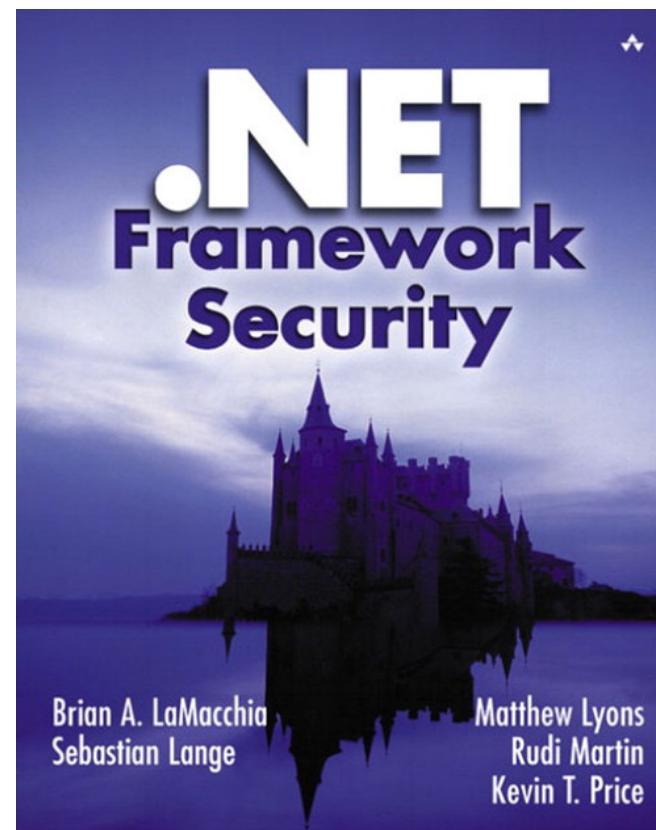
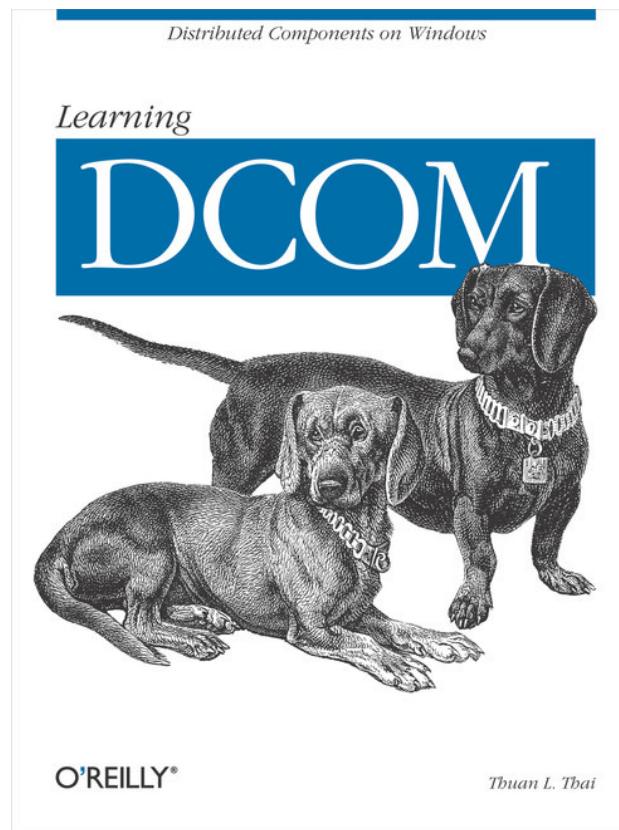


WebAssembly Component Model

- Cloud Native WASM Day EU 2023
 - Evolution of Wasm: Past, Present, Future - Bailey Hayes, Cosmonic
https://youtu.be/6_BRLqxiZPU
 - WASI and the Cloud - Jiaxiao Zhou, Microsoft & Dan Gohman, Fastly
https://youtu.be/5WQRT62V_VU
 - Future of Component Tooling - Peter Huene & Guy Bedford, Fastly
<https://youtu.be/JCIwpc7x4jU>

0101
0101

Have we seen this before?





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/codeeurope2023-wasm>
- ntanis at Veracode.com
- @nielstanis@infosec.exchange
- <https://blog.fennec.dev>
- Dziękuję! Thank you!