# WebAssembly: Mechanisation, Security, and Concurrency

Conrad Watt

University of Cambridge

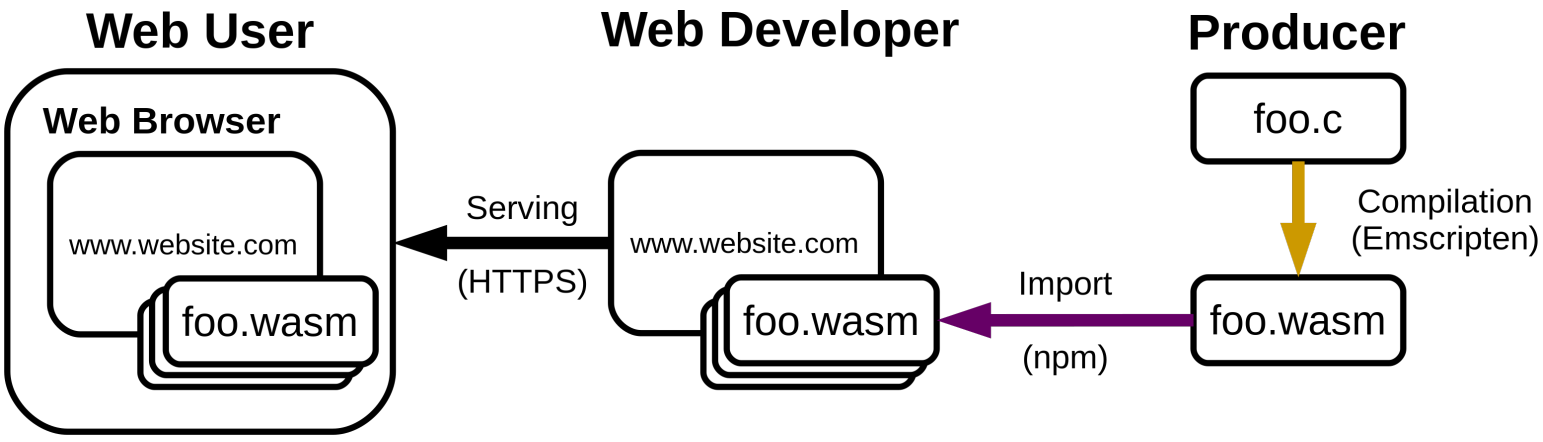
Verified Software Workshop 2019

## A brief history of WebAssembly (Wasm)

- A low-level bytecode, supported by all major browsers.
- A "compilation target for the Web".
- Has a principled formal specification.


WEBASSEMBLY

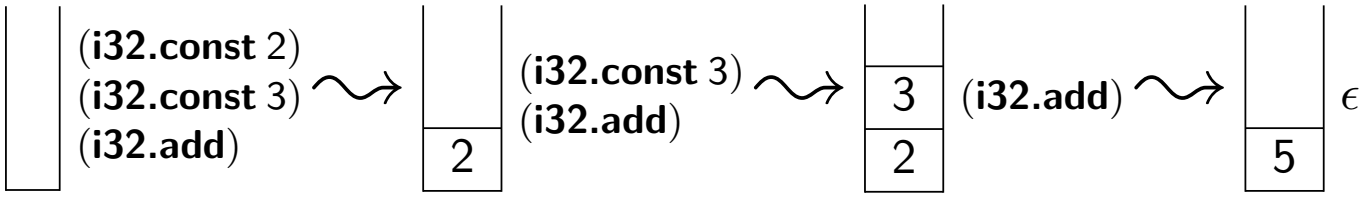
# How WebAssembly can be Used




- Wasm bytecode files are packaged and distributed without their original source files.

## WebAssembly is Stack-Based

- WebAssembly is specified using a small-step formal semantics.
- WebAssembly programs must be *validated* (type-checked) before they can be run.
- Only well-typed programs may be executed.



## Stack Typing

Typing program fragments:

| | | | |
|---|---|---|---|
| (**i32.const** 2) | (**i32.add**) | (**i32.const** 2)<br>(**i32.const** 3)<br>(**i32.add**) | (**f64.const** 0)<br>(**i32.const** 3)<br>(**i32.add**) |
| $[] \to [\mathsf{i32}]$ | $[\mathsf{i32}, \mathsf{i32}] \to [\mathsf{i32}]$ | $[] \to [\mathsf{i32}]$ | $\bot$ |

Some structural typing rules:

$$\frac{e^* : t_a^* \to t_b^*}{e^* : t^*; t_a^* \to t^*; t_b^*} \qquad \frac{e_1^* : t_a^* \to t_b^* \qquad e_2^* : t_b^* \to t_c^*}{e_1^*; e_2^* : t_a^* \to t_c^*}$$

# WebAssembly type system

**Progress**

For any validated program P that has not terminated with a result, there exists P' such that P reduces to P'

**Preservation**

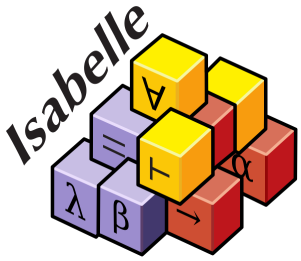
If a program P is validated with a type ts, any program obtained by reducing P to P' can also be validated with type ts.

These properties together guarantee syntactic type soundness.[1]

[1] A.K. Wright and M. Felleisen. "A Syntactic Approach to Type Soundness". In: *Information and Computation* 115.1 (1994). ISSN: 0890-5401.

# Mechanisation

- An unambiguous formal specification and an unambiguous correctness condition.
- Perfect for mechanisation!
- ∼11,000 lines of Isabelle/HOL.[2]
- Found several errors in the draft specification.
- Also included:
  - Verified sound and complete type-checking algorithm.
  - Verified sound run-time interpreter.




[2] Conrad Watt. "Mechanising and Verifying the WebAssembly Specification". In: *Certified Programs and Proofs (CPP 2018)*.

# Mechanisation

**Wasm Logic**

A separation logic for WebAssembly.

Petar Maksimović*

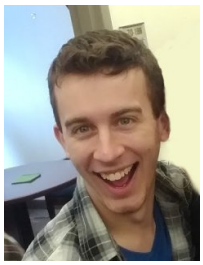
Neel Krishnaswami†

Philippa Gardner*

Imperial College London*/Cambridge†

**CT-Wasm**

Secure information flow type system.



John Renner

Natalie Popescu

Sunjay Cauligi

Deian Stefan

UC San Diego

## Wasm Logic[3]

- WebAssembly's stack is very static.
- The type system guarantees a precise structure.
- A program logic should mirror/take advantage of this structure.

[3] Conrad Watt, Petar Maksimovic, Neelakantan R. Krishnaswami, and Philippa Gardner. "A Program Logic for First-Order Encapsulated WebAssembly". In: *European Conference on Object-Oriented Programming (ECOOP 2019)*.

# Wasm Proof Rules - Control

Notice how closely these proof rules follow the typing rules!

$$\frac{t^m; labs \vdash e^* : t^n \to t^m}{labs \vdash (\textbf{block}\ (t^n \to t^m)\ e^*\ \textbf{end}) : t^n \to t^m}\ \text{block typing}$$

$$\frac{Q_m\ ; L \vdash \{P_n\}\ e^*\ \{Q_m\}}{L \vdash \{P_n\}\ \textbf{block}\ (t^n \to t^m)\ e^*\ \textbf{end}\ \{Q_m\}}\ [\text{block}]$$

---

$$\frac{labs!k = t^*}{labs \vdash (\textbf{br}\ k) : t^* \to t^*}\ \text{br typing} \qquad \frac{L!k = P}{L \vdash \{P\}\ \textbf{br}\ k\ \{Q\}}\ [\text{br}]$$

# Wasm Proof Rules - Control

Wasm's **loop** opcode works like **block**, except executing **br** *restarts* the loop, like a continue statement.

$$\frac{t_a^*; labs \vdash e^* : t_a^* \to t_b^*}{labs \vdash (\textbf{loop}\ (t_a^* \to t_b^*)\ e^*\ \textbf{end}) : t_a^* \to t_b^*}\ \text{loop typing}$$

$$\frac{P_n\ ; L \vdash \{P_n\}\ e^*\ \{Q_m\}}{L \vdash \{P_n\}\ \textbf{loop}\ (t^n \to t^m)\ e^*\ \textbf{end}\ \{Q_m\}}\ [\text{loop}]$$

## CT-Wasm[4]

- WebAssembly's type system is very simple and static.
- We can easily add additional security annotations.
- Key insight - best practice cryptographic algorithms already obey a course-grained "type system" - *constant time* principles.

[4] Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi, and Deian Stefan. "CT-Wasm: Type-Driven Secure Cryptography for the Web Ecosystem". In: *Principles of Programming Languages (POPL 2019)*.

## CT-Wasm

- We turn violations of these principles into type errors.
- Security properties fully mechanised.

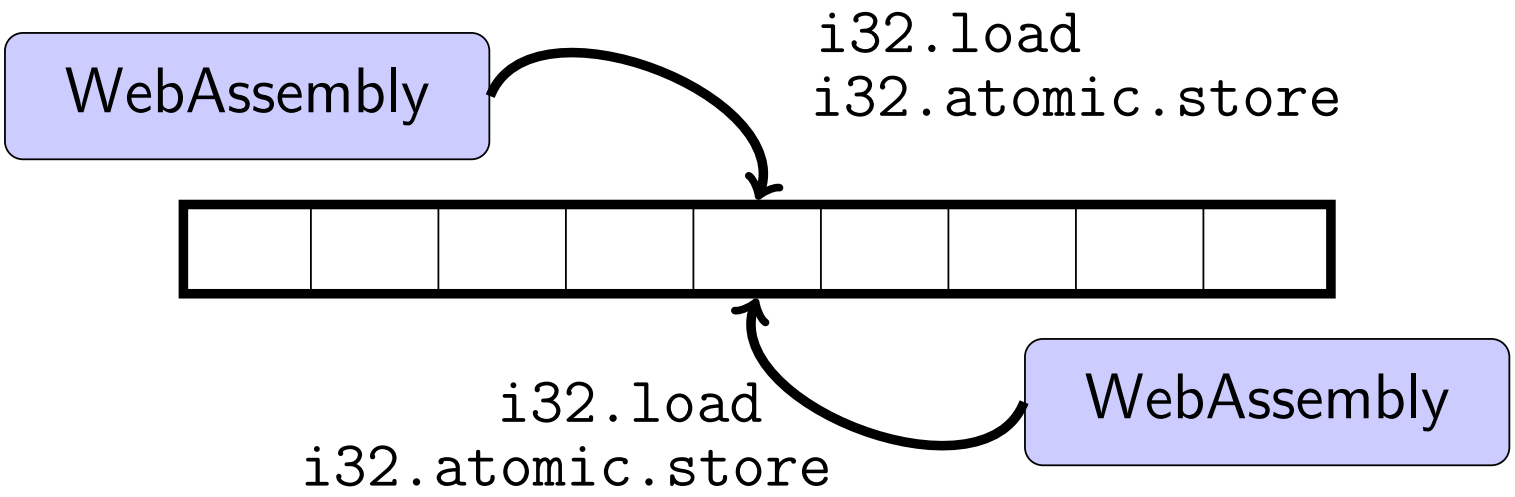
| | | |
|---|---|---|
| **s32.const** 2 | **s32.const** 2<br>**i32.const** 3<br>**i32.add** | **s32.const** 2<br>**br_if** |
| [] → [s32] | ⊥ | ⊥ |

## Relaxed Memory

Guillaume Barbier (ENS Rennes)
Stephen Dolan (University of Cambridge)
Shaked Flur (University of Cambridge)
Shu-yu Guo (Google / Bloomberg LP)
Jean Pichon-Pharabod (University of Cambridge)
Anton Podkopaev (HSE / MPI-SWS)
Christopher Pulte (University of Cambridge)
Andreas Rossberg (Dfinity Stiftung)

## Relaxed Memory

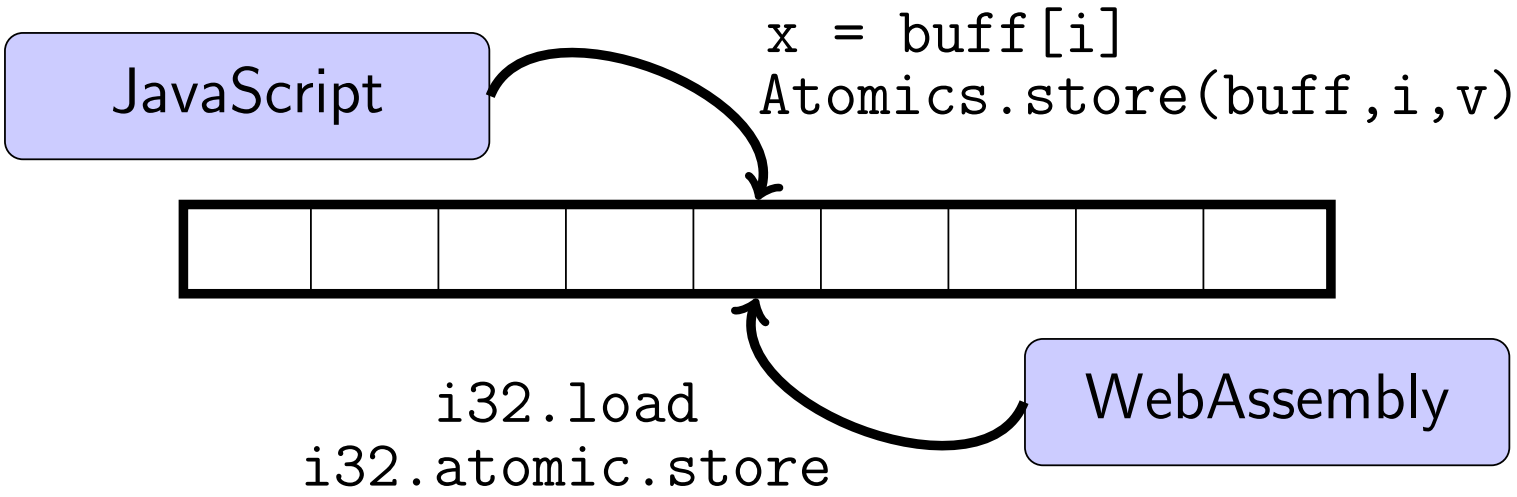
- WebAssembly program can read from and write to a linear buffer of raw bytes.
- Adding threads, these buffers can now be shared.
- Need a relaxed memory model.



## Relaxed Memory

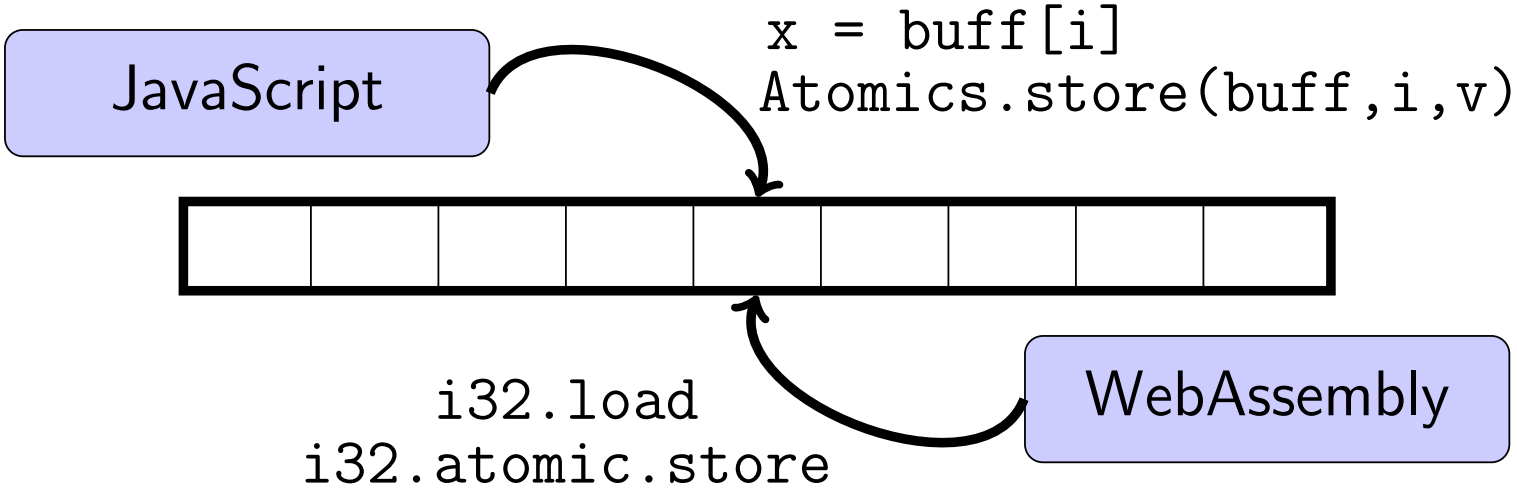
- JavaScript also has threads ("web workers") and shared buffers, even a memory model!

- The WebAssembly memory will be exposed to JavaScript as a shared buffer.



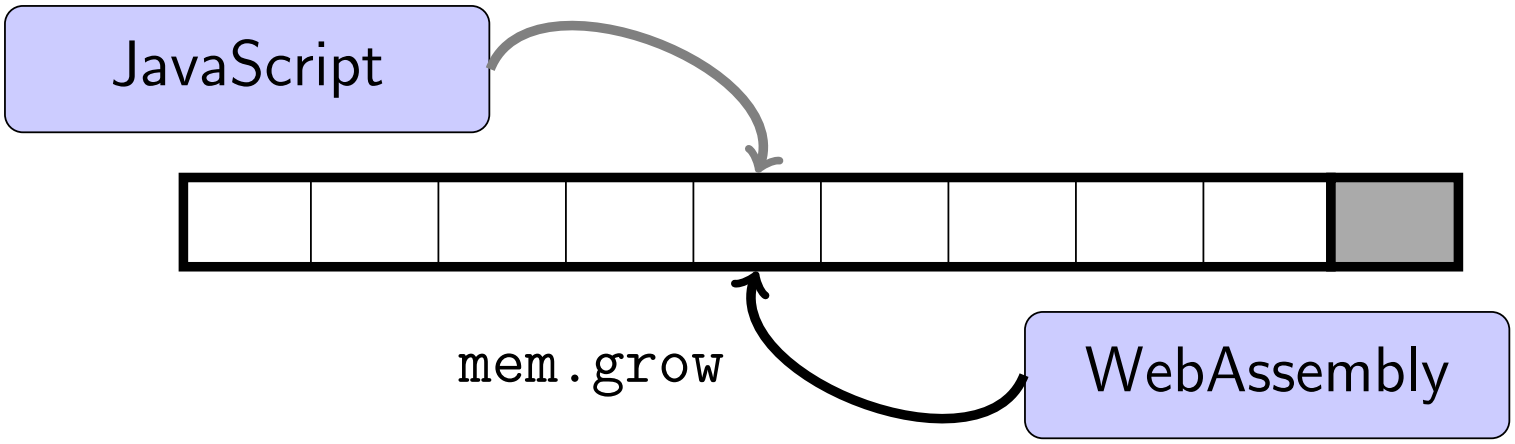
## Relaxed Memory

- Committee: JS/Wasm interop should "just work".
- So a lot of Wasm consistency behaviour is inherited from JS.



## Relaxed Memory

- But Wasm has additional feature - memory growth.
- Now, the size of the memory needs to become part of the axiomatic model.



## Relaxed Memory

- Implementers don't want to guarantee SC bounds-checking behaviour.
- Updates to memory size can create "data" races.[5]


```
store x 42  ||  load y
grow 2      ||  load x
```

[5] Conrad Watt, Andreas Rossberg, and Jean Pichon-Pharabod. "Weakening WebAssembly". In: *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2019)*.

## Relaxed Memory

- We said Wasm follows JS.
- What if the JS model is wrong? Ideally, we fix it.
- JS standards body has been *very* welcoming.
- Shu-yu Guo (Bloomberg LP) has been a great point of contact.

## Relaxed Memory

- Several JS memory model problems discovered.
  - Missing synchronization for wait/wake ops.[6]
  - SC-DRF violation.[7]
  - ARMv8 lda/stl not supported (Stephen Dolan, Cambridge).[8]

[6] Conrad Watt. *Normative: Strengthen Atomics.wait/wake synchronization to the level of other Atomics operations*. Mar. 2018. URL: https://github.com/tc39/ecma262/pull/1127.

[7] Shu-yu Guo. *Normative: Fix memory model so DRF-SC holds*. Nov. 2018. URL: https://github.com/tc39/ecma262/pull/1362.

[8] Shu-yu Guo. *Memory Model Support for ARMv8 LDA/STL*. Jan. 2019. URL: https://docs.google.com/presentation/d/1qif7z-Y8C-nvJM20UNJQzAKJgLN4wmXS_5NN2Wgipb4/edit?usp=sharing.

## SC-DRF violation

```
Atomics.store(v,0,1);  ||  Atomics.store(v,0,2);
                           if (Atomics.load(v,0) === 1) {
                             r = v[0];
                           }
```

- You might think that `r` must always be assigned 1.
- Not so, before our corrections!
- This example, and others, found by model checking (in Alloy).[9]

[9] John Wickerson, Mark Batty, Tyler Sorensen, and George A. Constantinides. "Automatically Comparing Memory Consistency Models". In: *Principles of Programming Languages (POPL 2017)*.

# bit.ly/2M8cZ2v

- Android/Desktop Chrome for best results.
- You will almost certainly observe store buffering (SB) relaxed behaviour.[10]

```
store x 1  ||  store y 1
load y     ||  load x
```

- Do you observe the JavaScript Violation? This is the ARMv8 compilation violation.

[10] https://www.cl.cam.ac.uk/ pes20/ppc-supplemental/test6.pdf

## Thanks for listening!


WEBASSEMBLY