That's a Tough Call!
Studying the Challenges of Call Graph Construction for WebAssembly

Daniel Lehmann[+], Michelle Thalakottur[*], Frank Tip[*], Michael Pradel[+]

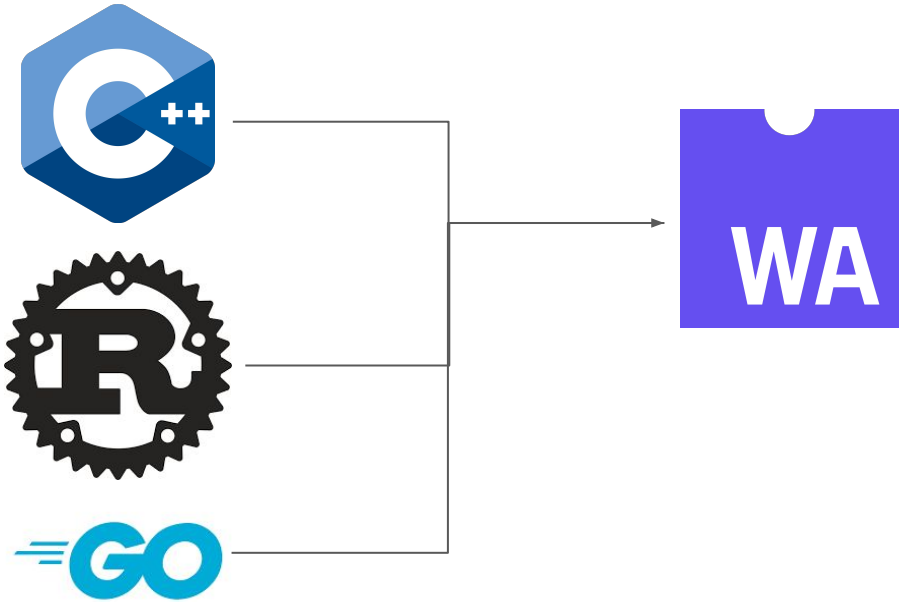
[*] Northeastern University
Khoury College of Computer Sciences

[+] University of Stuttgart
Germany

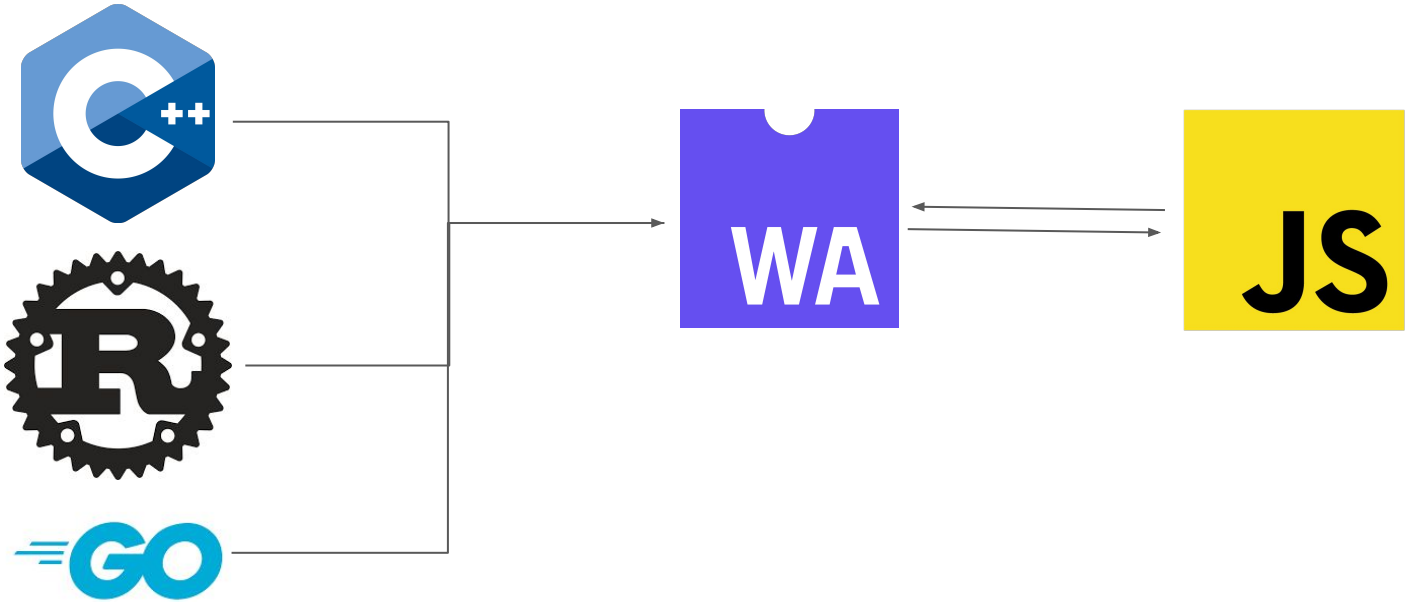
WebAssembly

Fast
Compact
Portable

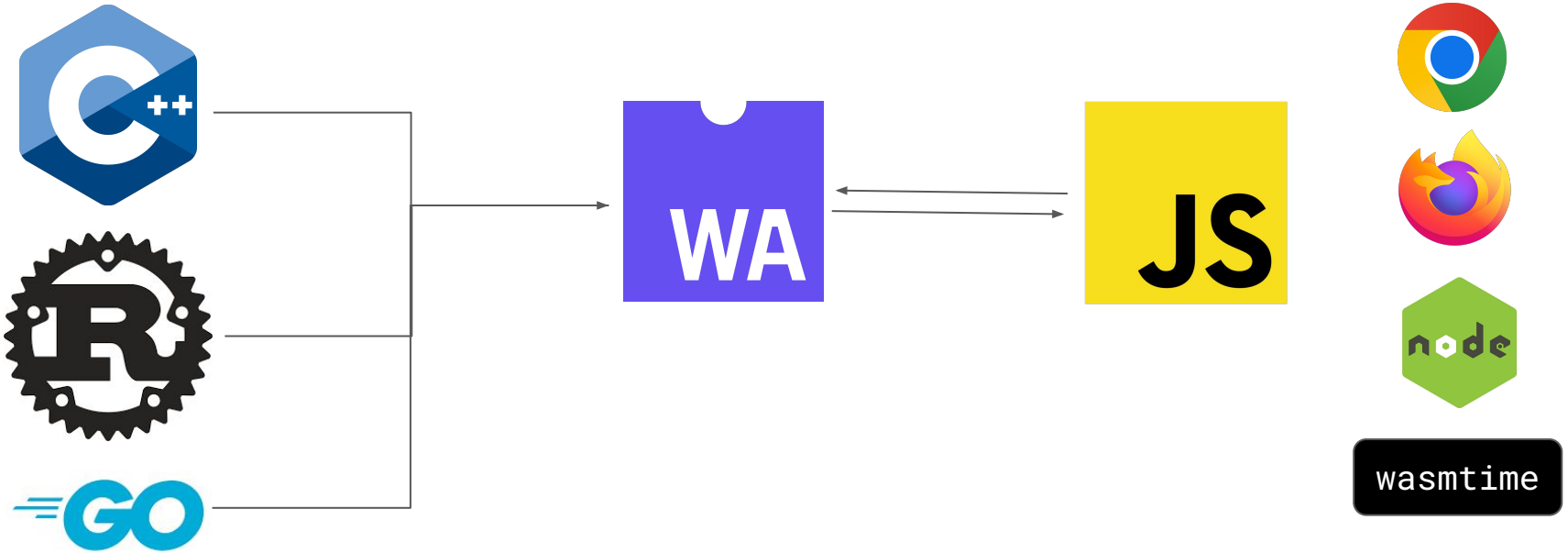
WebAssembly



WebAssembly



WebAssembly



Call Graph Analysis is Important!

- Core of many inter-procedural static analyses!
  - Detect unused code that can be removed from a binary (debloating)
  - Reverse engineering binaries
  - Optimizations

This Work: Call Graphs in Wasm

- ☐ Identify 12 challenges for sound and precise static analysis
  - ☐ Prevalence study of these challenges over the WasmBench dataset (8461 binaries)
- ☐ Evaluation of 4 real world static analysis tools
  - ☐ 24 microbenchmarks
  - ☐ 10 real WebAssembly libraries

Challenges Overview

Indirect Calls:

- Table Indirection
- Table Index Value
- Table Initial State
- Table Mutation

Program Representation:

- Function Indices
- Program Structure

Host Environment:

- Host Callbacks
- Entry Points

Memory:

- Memory Management
- Memory Mutable

Types: Low Level Types

Source Languages: Multi PL

Challenges Overview

Indirect Calls:

- Table Indirection
- Table Index Value
- Table Initial State
- Table Mutation

Program Representation:

- Function Indices
- Program Structure

Host Environment:

- Host Callbacks
- Entry Points

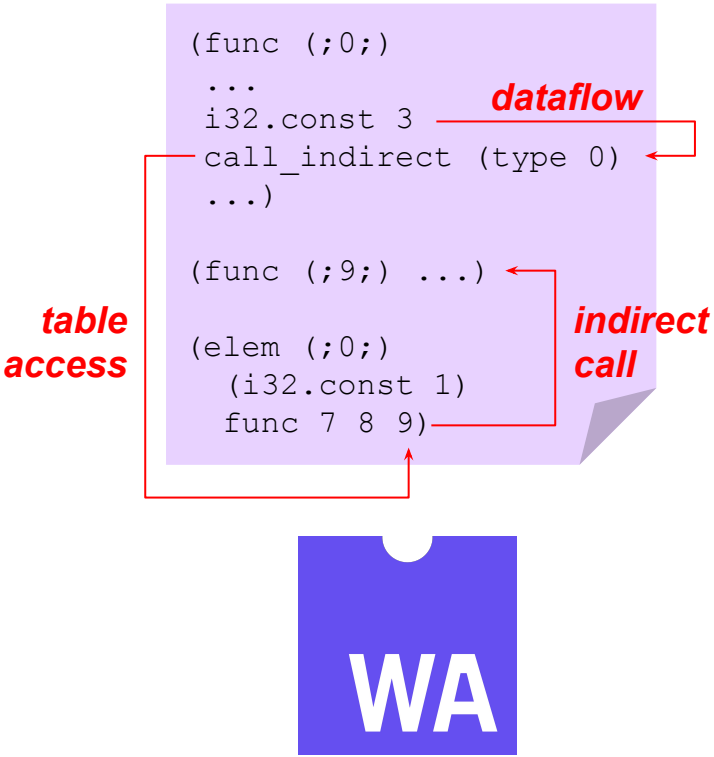
Memory:

- Memory Management
- Memory Mutable

Types: Low Level Types

Source Languages: Multi PL

Challenge: Indirect Calls



Challenge: Indirect Calls

```
(func (;0;)
 local.get 0
 i32.load
 call_indirect (type 0)
 ...)

(func (;9;) ...)

(elem (;0;)
  (i32.const 1)
  func 7 8 9)
```

Challenge: Indirect Calls

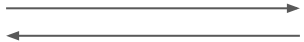
```
(func (;0;)
 local.get 0
 i32.load
 call_indirect (type 0)
 ...)

(func (;9;) ...)

(elem (;0;)
  (global.get 0)
  func 7 8 9)
```

Challenge: Indirect Calls

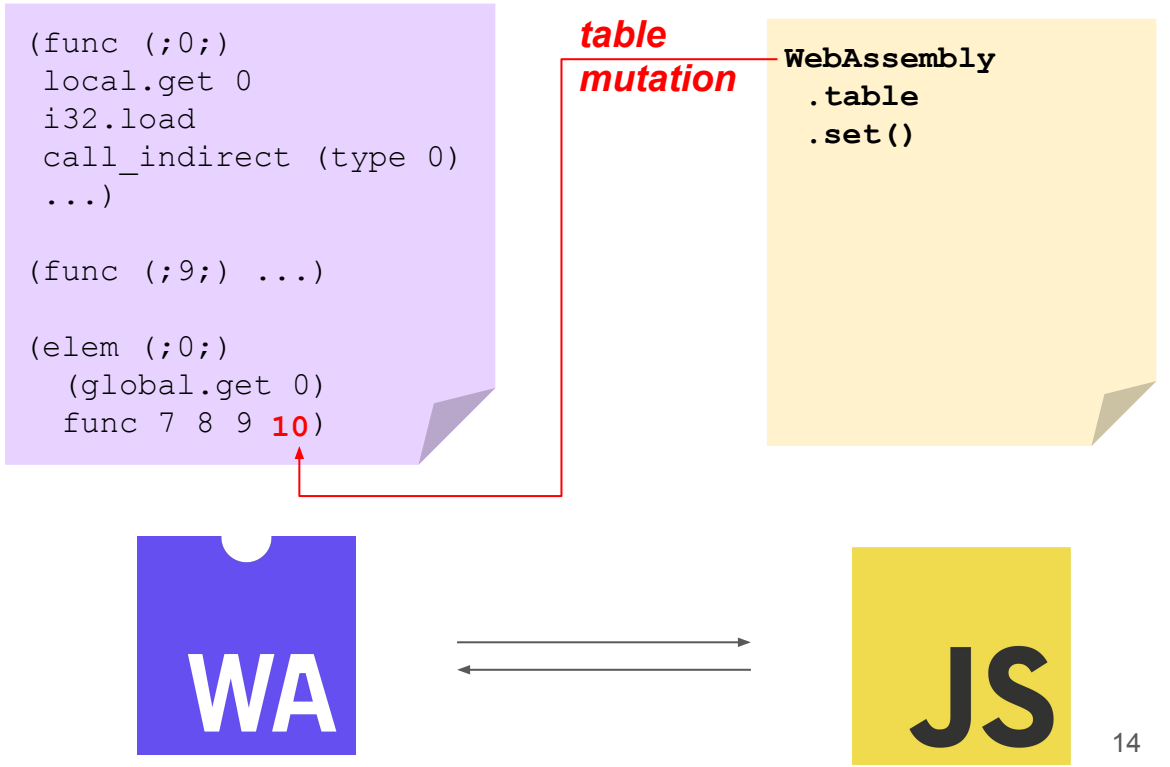
```
(func (;0;)
 local.get 0
 i32.load
 call_indirect (type 0)
 ...)

(func (;9;) ...)

(elem (;0;)
  (global.get 0)
  func 7 8 9)
```

dataflow

```
var x = 10
```



Challenge: Indirect Calls

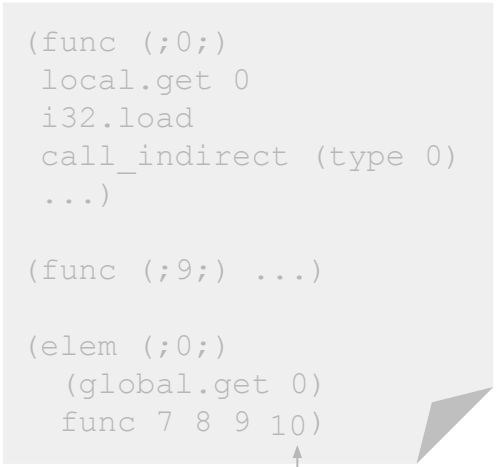


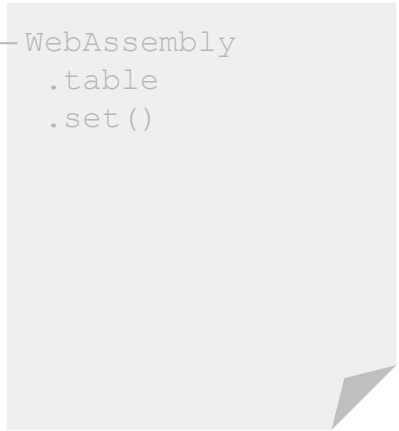
Challenge: Indirect Calls

49% of index instructions involve local variables, 64% involve reading memory.

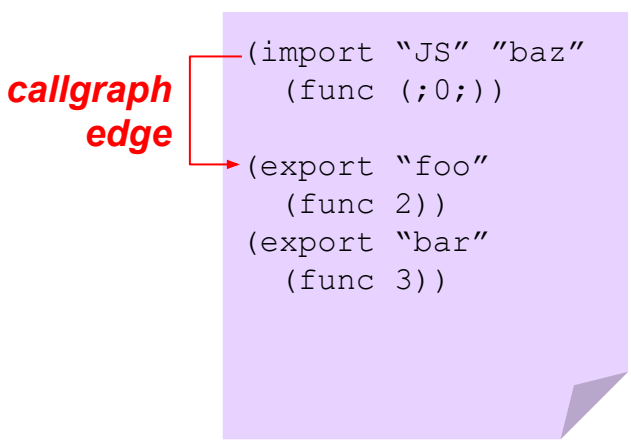
22% of Wasm tables are imported or exported.

12% of Wasm binaries have an imported variable as the table offset.

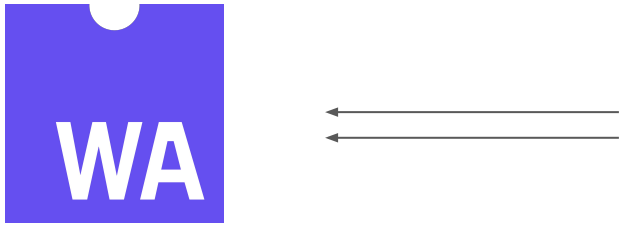




Challenge: Host Environment

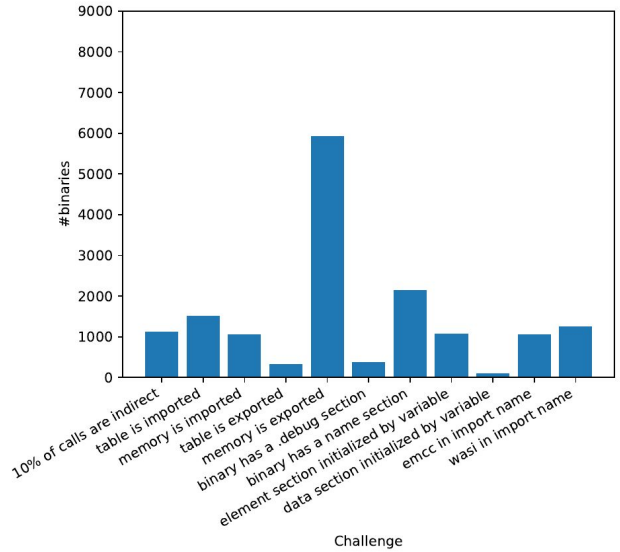


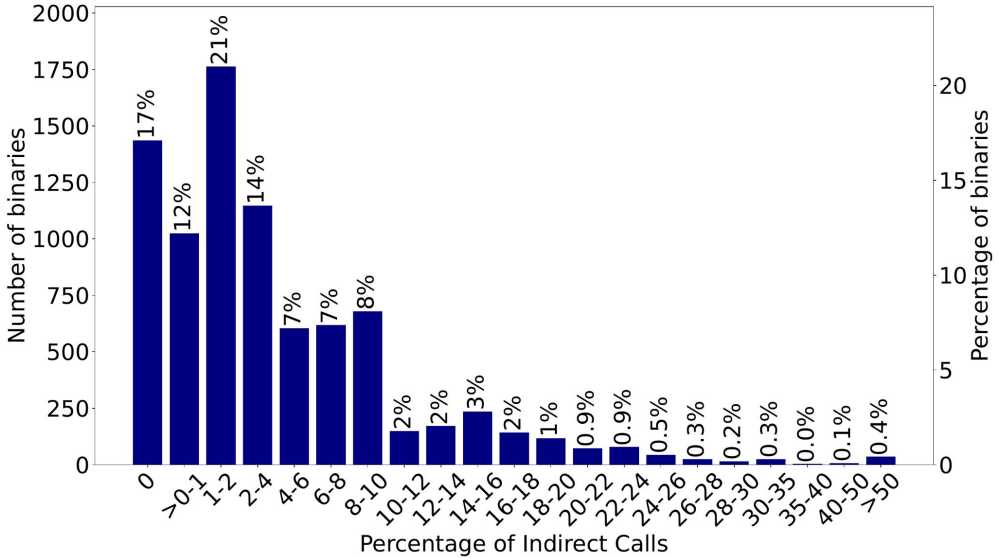

```
baz() {
    WebAssembly
      .exports
      .foo()
 }
```



Prevalence Study

- Prevalence study of 12 challenges over the WasmBench dataset (8461 binaries)





This Work: Call Graphs in Wasm

- ✔ Identify 12 challenges for sound and precise static analysis
  - ✔ Prevalence study of these challenges over the WasmBench dataset (8461 binaries)
- ☐ Evaluation of 4 real world static analysis tools
  - ☐ 24 microbenchmarks
  - ☐ 10 real WebAssembly libraries

Current Static Analysis Tools

Wassail

Static Analysis Framework
[1]

MetaDCE

Dead Code Elimination tool
[2]

Twiggy

Static Code Size Profiler
[3]

WAVM+ LLVM SA

Wasm -> LLVM IR Wasm VM
[4]

[1] https://github.com/acieroid/wassail
[2] https://github.com/WebAssembly/binaryen/blob/main/src/tools/wasm-metadce.cpp
[3] https://github.com/rustwasm/twiggy
[4] https://wavm.github.io/

Microbenchmarks

- 24 microbenchmarks that cover each challenge.

Microbenchmark	***Soundness***			
	Wassail	WAVM+LLVM	MetaDCE	Twiggy
Functions in exported table are reachable	✗	✗	✗	✔
Functions in imported table are reachable	✗	✗	✔	✗
Table is mutated by host	✔	✔	✗	✔
Table init offset is import from host	💣	✗	💣	✔

✔ Sound ✗ Unsound ⏱️Timeout 💣Crash

Real World Benchmarks: Soundness

- 10 Wasm libraries: sql.js, opencv, graphviz, fonteditor, etc

Real World Benchmarks	***Soundness***			
	Wassail	WAVM+LLVM	MetaDCE	Twiggy
sql.js	✔	✗	✔	✔
opencv	⏱	✗	💣	✗
graphviz	✗	✗	✔	✔
rsa	✔	✗	✔	✔

✔ Sound ✗ Unsound ⏱ Timeout 💣 Crash

Real World Benchmarks: DCE (Dead Code Elimination)

- Most tools are overly conservative

Real World Benchmarks	***Percentage of Functions Removed***			
	Wassail	WAVM+LLVM	MetaDCE	Twiggy
sql.js	**0%**	50%	**0%**	**0%**
opencv	⏰	92%	💣	92%
graphviz	1%	72%	**0%**	**0%**
rsa	1%	28%	**0%**	**0%**

⏰ Timeout 💣 Crash

This Work: Call Graphs in Wasm

- ✔ Identify 12 challenges for sound and precise static analysis
  - ✔ Prevalence study of these challenges over the WasmBench dataset (8461 binaries)
- ✔ Evaluation of 4 real world static analysis tools
  - ✔ 24 microbenchmarks
  - ✔ 10 real WebAssembly libraries

Conclusions

- Recommendations:
  - Tailor your analysis specifically to Wasm.
  - For precision, track data-flow and perform pointer analysis.
  - You have to analyze Wasm's interaction with the host environment.
- We're currently looking at how we can analyze JS applications that use WebAssembly!

Conclusions

- Recommendations:
  - Tailor your analysis specifically to Wasm.
  - For precision, track data-flow and perform pointer analysis.
  - You have to analyze Wasm's interaction with the host environment.
- We're currently looking at how we can analyze JS applications that use WebAssembly!

https://github.com/sola-st/wasm-call-graphs/