WASMine

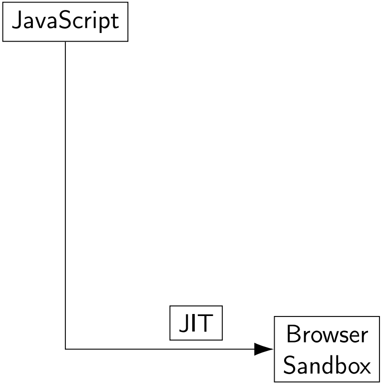
A WebAssembly Runtime with AOT, JIT and Interpreter Backends

Lukas Döllerer, Enrico Fazzi

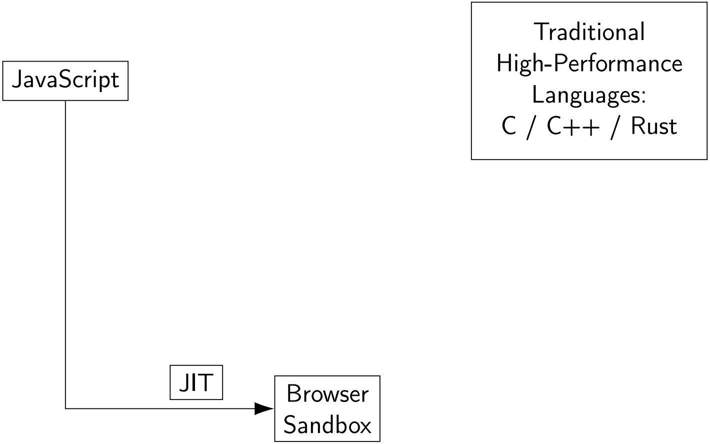
TUM School of Computation, Information, and Technology
Technical University of Munich

October 17, 2024

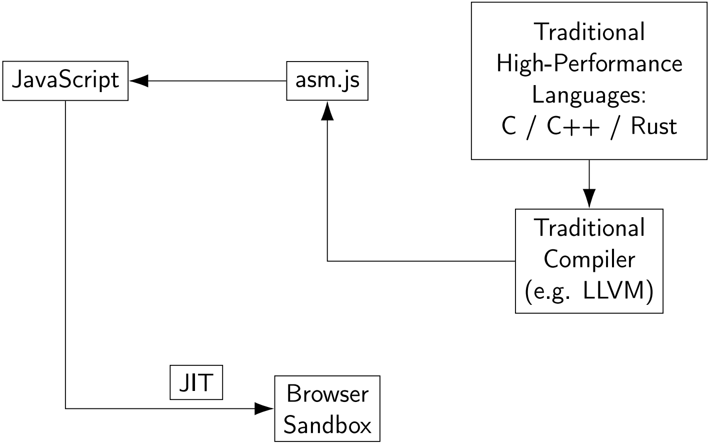
JavaScript, asm.js and WebAssembly



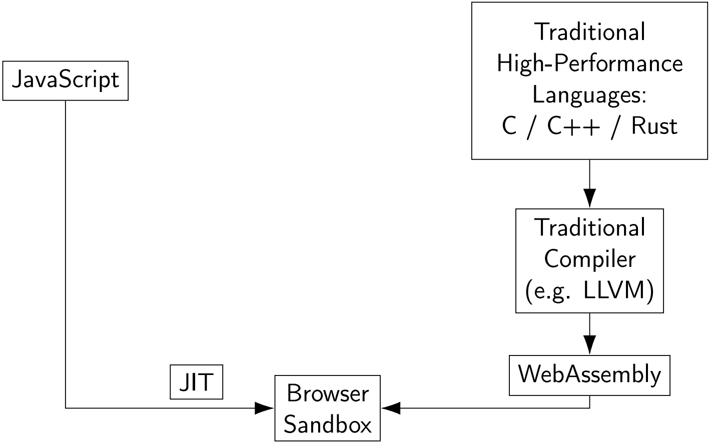
JavaScript, asm.js and WebAssembly



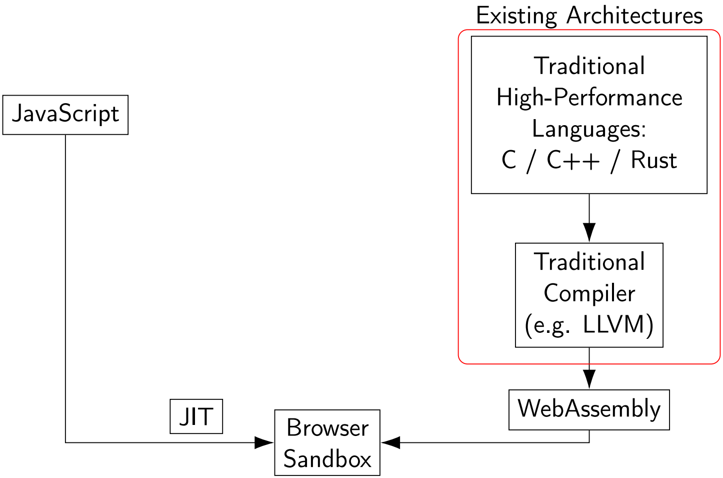
JavaScript, asm.js and WebAssembly



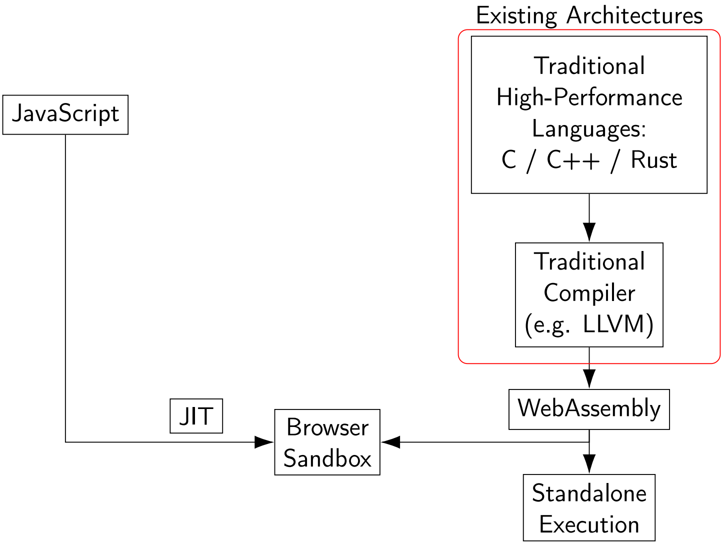
JavaScript, asm.js and WebAssembly



JavaScript, asm.js and WebAssembly



JavaScript, asm.js and WebAssembly



WebAssembly Runtimes

Browser JavaScript Engines:

- Google: v8
- Apple: JavaScriptCore
- Mozilla: SpiderMonkey

Standalone Runtimes:

- Wasmtime
- Wasmer
- WasmEdge
- ...

WebAssembly (Wasm)

- **Portable**: Low system requirements
  - 64 KiB pages
  - 32-bit and 64-bit integers and IEEE 754 floats
  - Memory abstraction: Linear array of 32-bit addressable bytes

WebAssembly (Wasm)

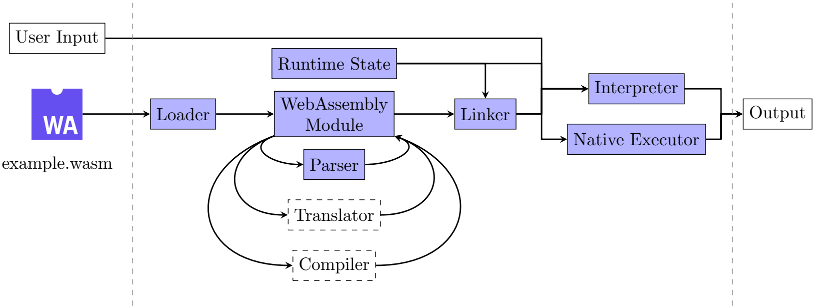
- **Portable**: Low system requirements
  - 64 KiB pages
  - 32-bit and 64-bit integers and IEEE 754 floats
  - Memory abstraction: Linear array of 32-bit addressable bytes
- **Simple**: Minimal instruction set
  - Stack-machine-based instructions
  - Simple arithmetic
  - `load` & `store` for memory accesses
  - `memory.grow` *n* to increase memory size
  - ...

WebAssembly (Wasm)

- **Portable**: Low system requirements
  - 64 KiB pages
  - 32-bit and 64-bit integers and IEEE 754 floats
  - Memory abstraction: Linear array of 32-bit addressable bytes
- **Simple**: Minimal instruction set
  - Stack-machine-based instructions
  - Simple arithmetic
  - `load` & `store` for memory accesses
  - `memory.grow` *n* to increase memory size
  - ...
- **Safe**: Designed for sandboxed execution
  - No direct access to the host system
  - Memory access is bounds checked

WASMine

High-Level ARchitecture



WASMine

Key Optimizations

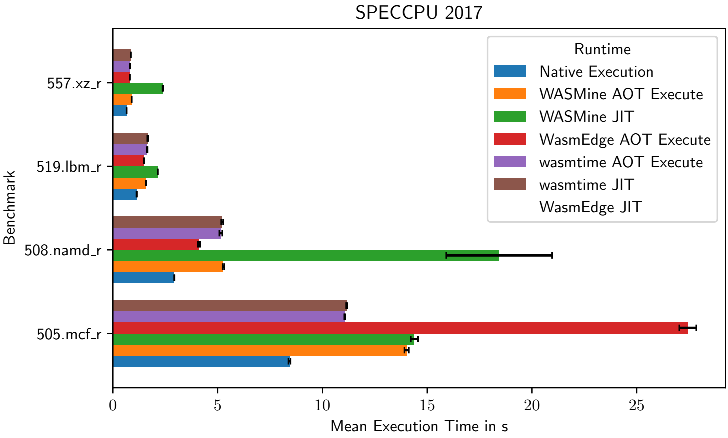
- Numerically encoded Function Types for efficient storage and comparison
- Direct Wasm bytecode to LLVM IR translation (single pass)
- LLVM AOT compilation (+ object file loading)
- Lazy function table population
- Lazy function (code) loading

Benchmarks

- SPEC CPU®,2017
  - Computationally intensive, real-world programs
  - Only the 505.mcf_r, 508.namd_r, 519.lbm_r, 557.xz_r are compilable to WebAssembly (out of the box)
- PolyBench
  - 30 numerical computations, extracted from real-world applications
  - Commonly used for benchmarking compilers
  - Free and Open Source

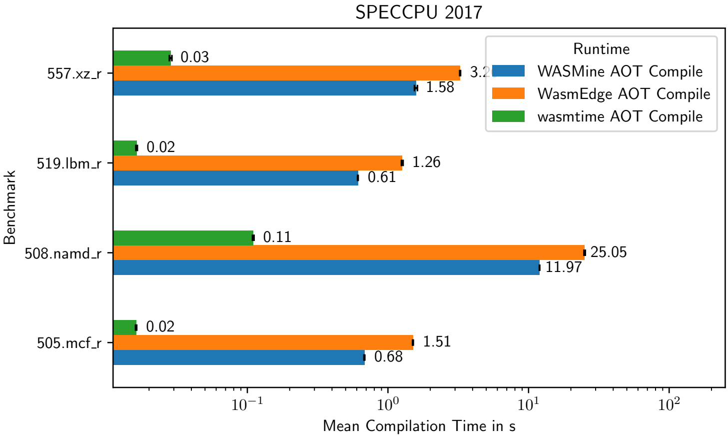
Benchmarks

SPECCPU 2017 Execution Time



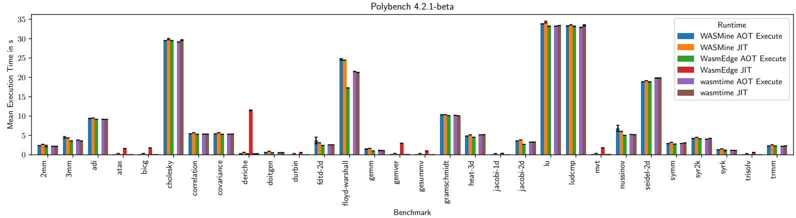
Benchmarks

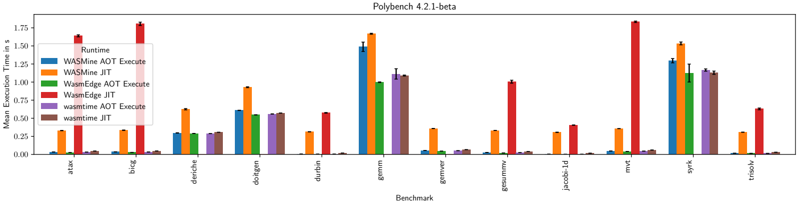
SPECCPU 2017 AOT Compilation Time



Benchmarks

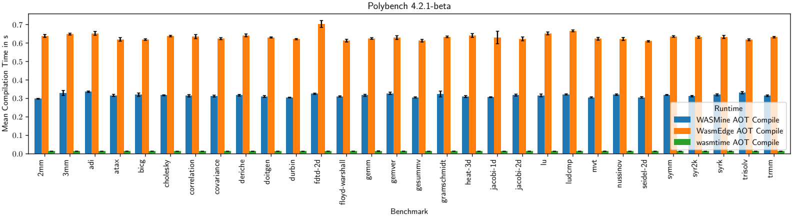
PolyBench Execution Time





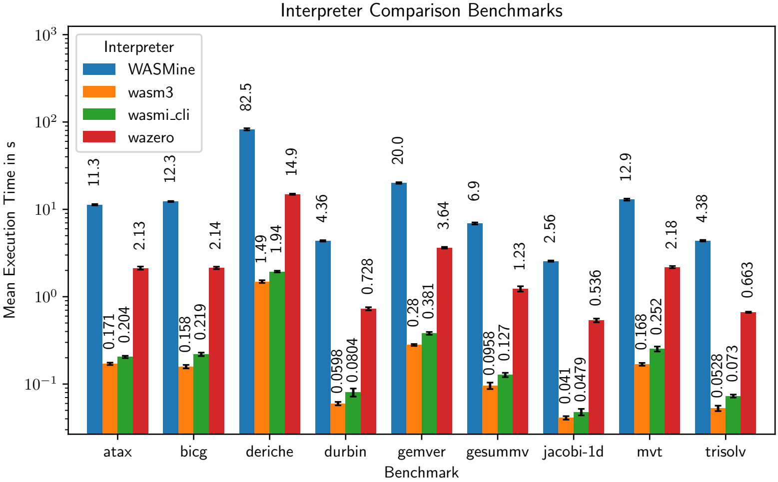
Benchmarks

PolyBench AOT Compilation Time



Benchmarks

Selected PolyBench Interpreter Execution Time



Live Demo!

```c
int main() {
    int i, t1 = 0, t2 = 1, nextTerm, n = 10;

    puts("Fibonacci Series: ");
    for (i = 1; i <= n; ++i) {
        printf("%d", t1);
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;

        if(i < n) {
                printf(", ");
            }
    }
    puts("");
    return 0;
}
```

```
~/.local/share/wasi-sdk/bin/clang ./fibonacci.c -o fibonacci.wasm
```

Live Demo!

LLVM JIT

```
wasm_rt -b llvm run-wasi ./fibonacci.wasm
```

LLVM AOT

```
wasm_rt -b llvm compile ./fibonacci.wasm -o ./fibonacci.cwasm
wasm_rt -b llvm run-wasi ./fibonacci.cwasm
```

Interpreter

```
wasm_rt -b interpreter run-wasi ./fibonacci.wasm
```

Outlook

Future Work

- **Optimizations:** Further optimize parsing, de-/serialization (IR), memory management (caching), and interpreter
- **Testing:** Interpreting an interpreter
- **Documentation:** ~~Improve~~ Write the documentation

Conclusion

What we've done

During this course we have...

- implemented a WebAssembly runtime with AOT, JIT, and Interpreter backends and benchmarked all of them
- evaluated the runtime with the WebAssembly specification tests
- benchmarked our runtime with the SPEC CPU 2017 and PolyBench benchmark suites

Conclusion

What we've learnt

- WebAssembly itself
- Parser design
- LLVM's IR and (C-)API
- Interpreter design
- Cooperation is key