Bringing the power of C++ to the web

Krzysztof Paprocki
Meeting C++
Berlin, 16.11.2019

How many websites were existing in 1991?

How many websites were existing in 1991?

1

- Today there are more than 1 B websites
- 400 M are active
- 4.33 billion people are using internet (56% of the population)
- The “Next billion users” are coming

History of speeding up and enriching the web

- ActiveX
- Flash
- NaCl
- PNaCl
- asm.js
- WebAssembly

WebAssembly

- WebAssembly (abbreviated Wasm) is a binary instruction format (and language) for a stack-based virtual machine.
- Binary code – compiled/executed by the host (client)
- Designed to be close to the bare metal and therefore efficient
- Has also human readable text form (possible to write WASM manually, but not recommended)
- Developed by community and consortium

Not only Web and not Assembly

WebAssembly

Not only Web and not Assembly

~~Web~~Assembly

Not only Web and not Assembly

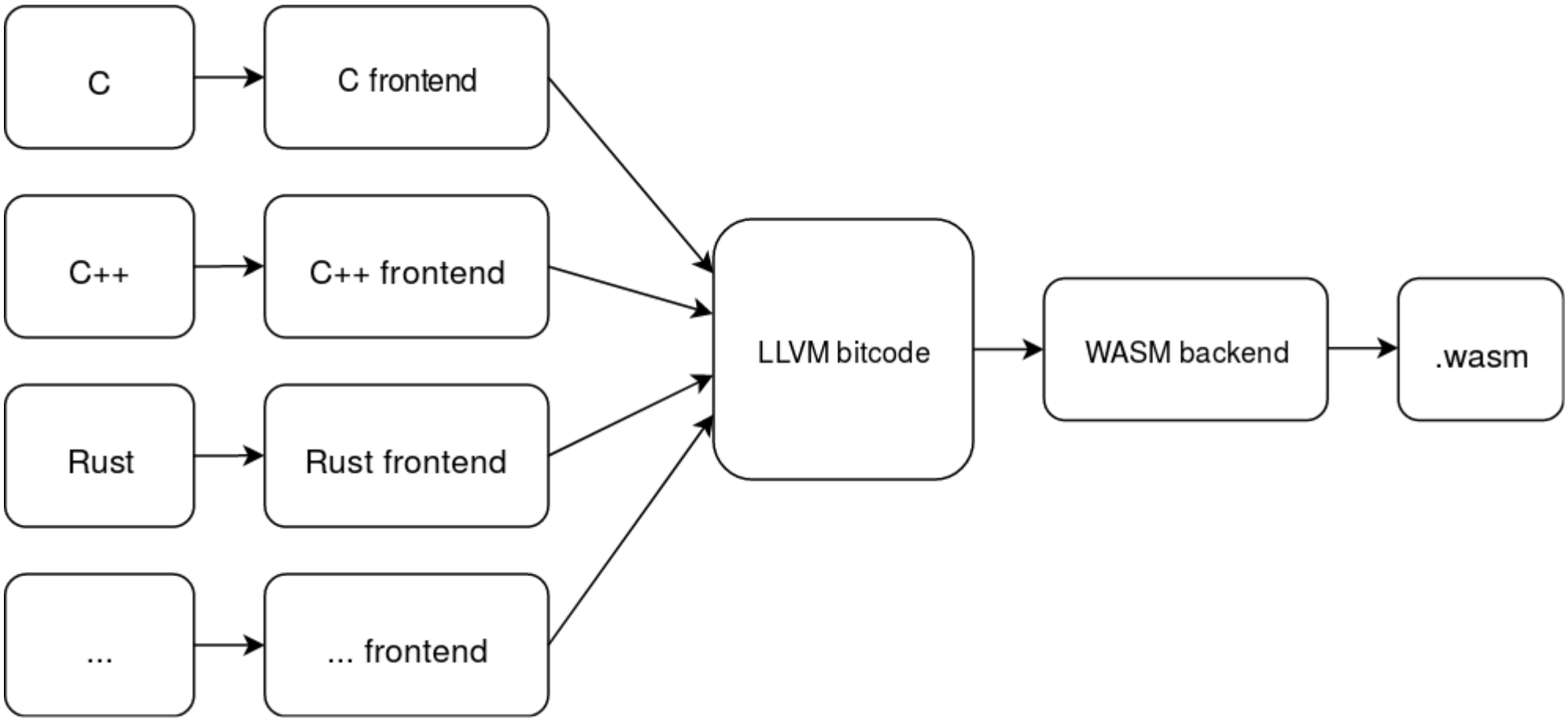
~~WebAssembly~~

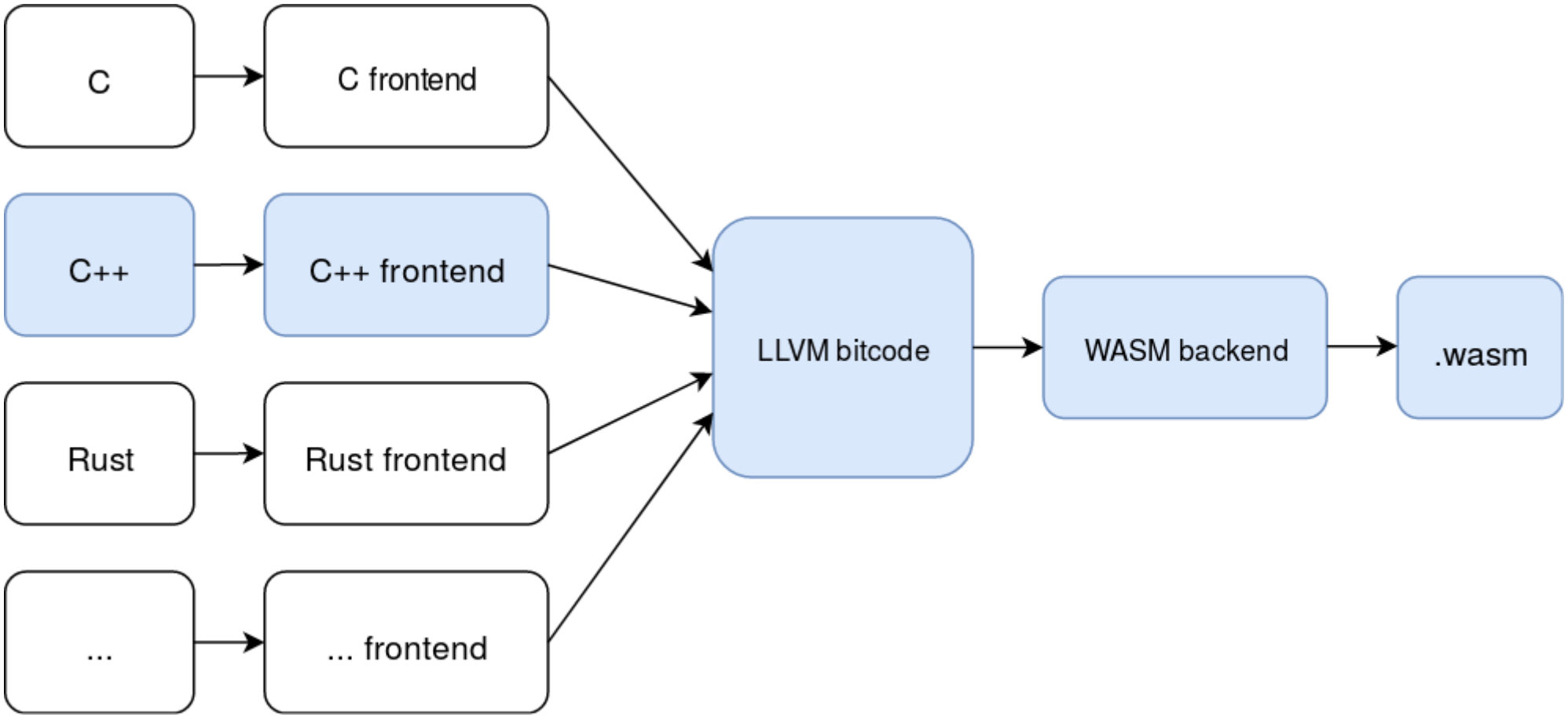
WebAssembly - philosophy

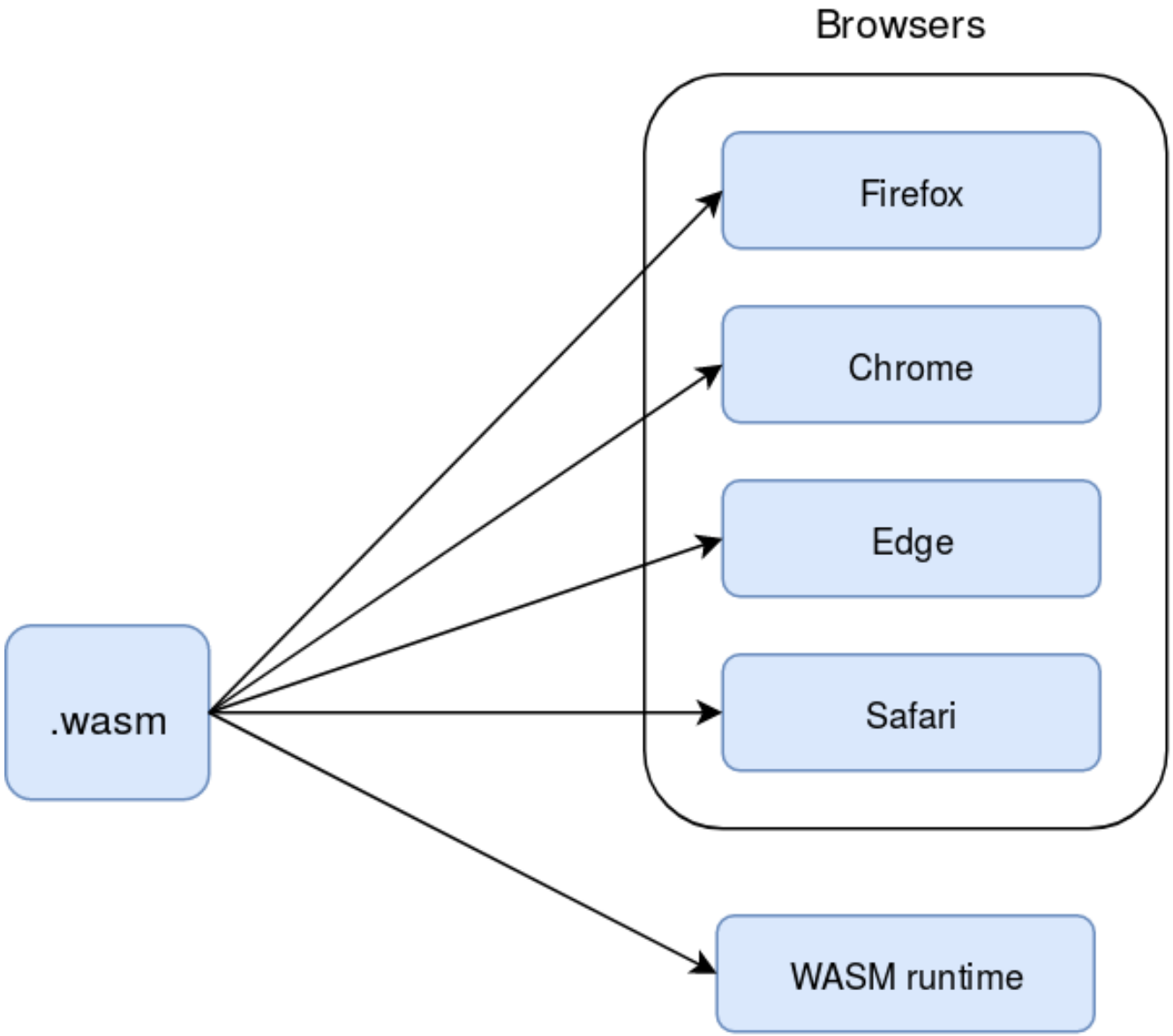
- “Write once, run anywhere”, Sun Microsystems

WebAssembly - philosophy

- Compile once, run anywhere
- Not only about C++


C
C frontend
C++
C++ frontend
Rust
Rust frontend
...
... frontend
LLVM bitcode
WASM backend
.wasm


C
C frontend
C++
C++ frontend
Rust
Rust frontend
...
... frontend
LLVM bitcode
WASM backend
.wasm


Browsers
Firefox
Chrome
Edge
Safari
.wasm
WASM runtime

Solomon Hykes
@solomonstre

Follow

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!

WebAssembly System Interface (WASI)

- WebAssembly run outside the browser
- In a secure manner (sandboxed)
- Abstraction middle-ware to access OS
- Efficient and safe run-time bridge between WASM and OS

Applications

New code:

- Distributed computing systems, Distributed Ledger Technologies (DLT)
  - (more) deterministic systems (Ethereum eWASM)
  - Web3, global computer (e.g. Dfinity)
- Big web apps (e.g. Figma)
- Web games

Applications

Existing code:

- Existing libraries (OpenCV)
- Existing native applications/frameworks (AutoCAD, Qt)
- Game engines (Unity, Unreal)

Applications

Mobile:

PWA (Progressive Web Apps)

WebAssembly - properties

- Portability, flexibility, speed
- it is not for small modules, but designed to work well with heavy weight web apps (eg. Figma, OpenCV)
- Significant memory consumption (64kB pages)
- Enables “serverless” architecture
- **Determinism and predictability**

Formats

- .wasm is binary
- .wat (WebAssembly Text)

Example: add

C++

```
int add(int a, int b)
{
    return a + b;
}
```

WAT

```
(module
  (func $add (param $0 i32)
        (param $1 i32) (result i32)
    get_local $0
    get_local $1
    i32.add)
  (export "add" (func $add))
)
```

```
0000000: 0061 736d  ; WASM_BINARY_MAGIC
0000004: 0100 0000  ; WASM_BINARY_VERSION
; section "Type" (1)
0000008: 01         ; section code
0000009: 00         ; section size (guess)
000000a: 01         ; num types
; type 0
000000b: 60         ; func
000000c: 02         ; num params
000000d: 7f         ; i32
000000e: 7f         ; i32
000000f: 01         ; num results
0000010: 7f         ; i32
0000009: 07         ; FIXUP section size
; section "Function" (3)
0000011: 03         ; section code
0000012: 00         ; section size (guess)
0000013: 01         ; num functions
0000014: 00         ; function 0 signature
index
0000012: 02         ; FIXUP section size
```

```
; section "Export" (7)
0000015: 07          ; section code
0000016: 00          ; section size (guess)
0000017: 01          ; num exports
0000018: 03          ; string length
0000019: 6164 64       add  ; export name
000001c: 00          ; export kind
000001d: 00          ; export func index
0000016: 07          ; FIXUP section size
; section "Code" (10)
000001e: 0a          ; section code
000001f: 00          ; section size (guess)
0000020: 01          ; num functions
; function body 0
0000021: 00          ; func body size (guess)
0000022: 00          ; local decl count
0000023: 20          ; local.get
0000024: 00          ; local index
0000025: 20          ; local.get
0000026: 01          ; local index
0000027: 6a          ; i32.add
0000028: 0b          ; end
0000021: 07          ; FIXUP func body size
000001f: 09          ; FIXUP section size
```

types

- i32: 32-bit integer
- i64: 64-bit integer
- f32: 32-bit float
- f64: 64-bit float

Performance – asm.js vs WASM

- On the long run WASM can be twice as fast as asm.js (for single-threaded code)

Performance – asm.js vs WASM

- On the long run WASM can be twice as fast as asm.js (for single-threaded code)

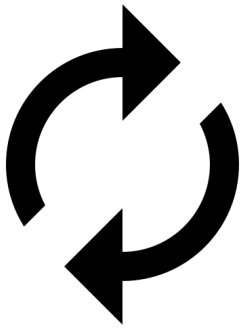
- But it does not matter;) (in most cases)

Performance – asm.js vs WASM

- On the long run WASM can be twice as fast as asm.js (for single-threaded code)
- But it does not matter;) (in most cases)
- Threads and SIMD are the game changers

Performance - JavaScript

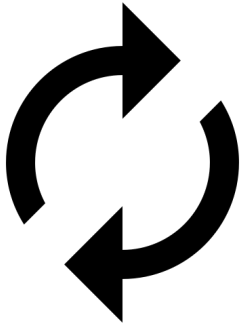
Ignition interpreter



TurboFan optimizing compiler

Performance - JavaScript

Ignition interpreter



TurboFan optimizing compiler

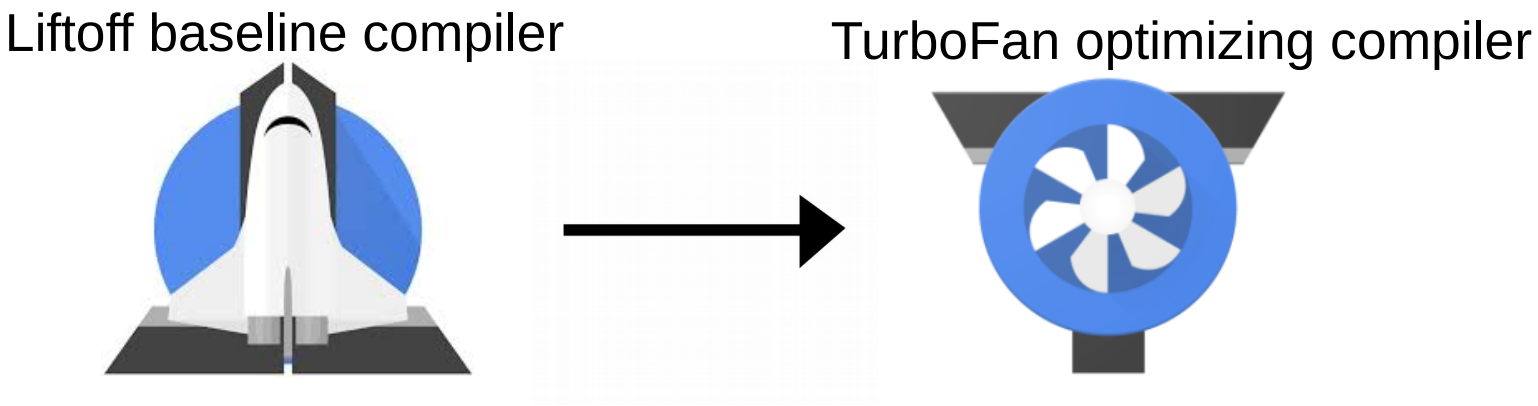
- JS is fast when optimized by TurboFan, slow when interpreted by Ignition
- It cannot be predicted when TurboFan kicks in
- Overall performance is good, but standard deviation may be big

Performance - WebAssembly

Liftoff baseline compiler

TurboFan optimizing compiler

Performance - WebAssembly



- Standard deviation is small
- Once optimized stays optimized
- deliver **predictable performance**

Performance – further optimizations

- Cold start vs caching
- Streaming compilation

Get it running

add.c

```
int add(int a, int b)
{
    return a + b;
}
```

```
$ emcc add.c -O1 -s SIDE_MODULE=1 -o add.wasm
```

|

compile only methods from add.c and nothing else

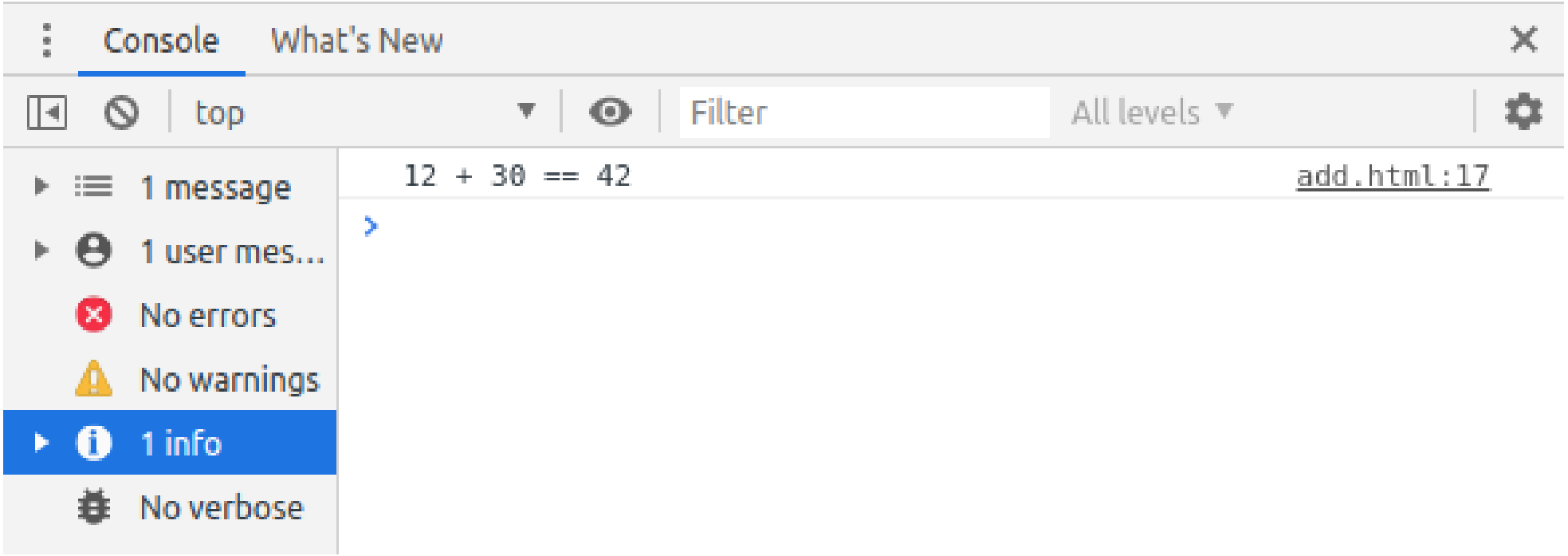
```
$ emcc add.c -O1 -s SIDE_MODULE=1 \
    -s "EXPORTED_FUNCTIONS=['add']" -o add.wasm
```

```
$ emcc add.c -O1 -s SIDE_MODULE=1 \
    -s "EXPORTED_FUNCTIONS=['add']" -o add.wasm
```

- Embind can help

```
<html>
<head>
  <script>
    WebAssembly.instantiateStreaming(fetch('add.wasm'))
    .then(obj => {
      var add = obj.instance.exports.add;
      var arg0 = 12;
      var arg1 = 30;
      console.log(`${arg0} + ${arg1} == ${add(arg0, arg1)}`)
    });
  </script>
</head>
<body>
</body>
</html>
```

```
$ emrun --no_browser --port 8080 .
```


Console
What's New
top
Filter
All levels
1 message
1 user mes...
No errors
No warnings
1 info
No verbose
12 + 30 == 42
add.html:17
>

Get it running

- It is not friction-free
- One can use available tools like Embind
- Name mangling is the challenge

Threads

- Experimental in many browsers
- Using web workers and SharedArrayBuffer
- Often disabled because of Spectre side channel attack

Experiments

78.0.3904.70

WARNING: EXPERIMENTAL FEATURES AHEAD! By enabling these features, you could lose browser data or compromise your security or privacy. Enabled features apply to all users of this browser.

Available | Unavailable

Experimental WebAssembly

Enable web pages to use experimental WebAssembly features. – Mac, Windows, Linux, Chrome OS, Android

#enable-experimental-webassembly-features

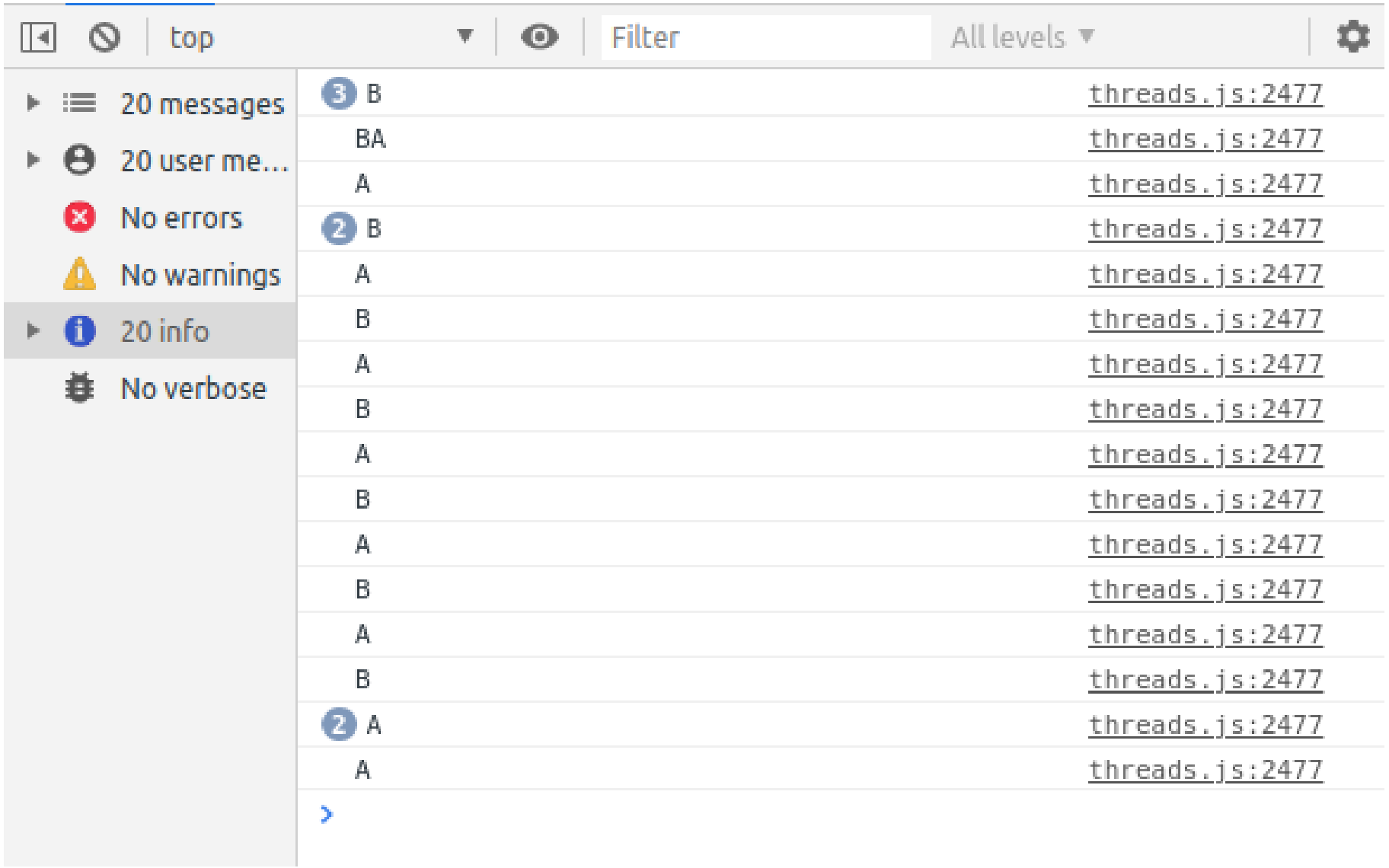
Enabled

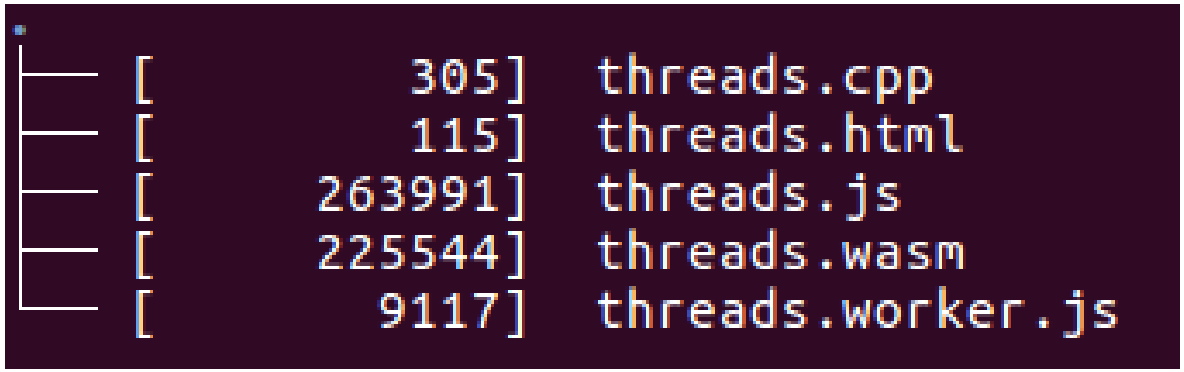
```
#include <iostream>
#include <thread>
#include <string>

void do_smth(const std::string& id) {
  for(int i = 0; i < 10; ++i) {
    std::cout << id << std::endl;
  }
}

int main() {
  std::thread first (do_smth, "A");
  std::thread second (do_smth, "B");

  first.join();
  second.join();

  return 0;
}
```

```
$ em++ -O1 -std=c++17 -s USE_PTHREADS=1 \
   -s PTHREAD_POOL_SIZE=2 -o threads.js threads.cpp
```

```
$ cat threads.html

<html>
  <title>Threads</title>
  <body>
    <script src="threads.js"></script>
  </body>
</html>
```

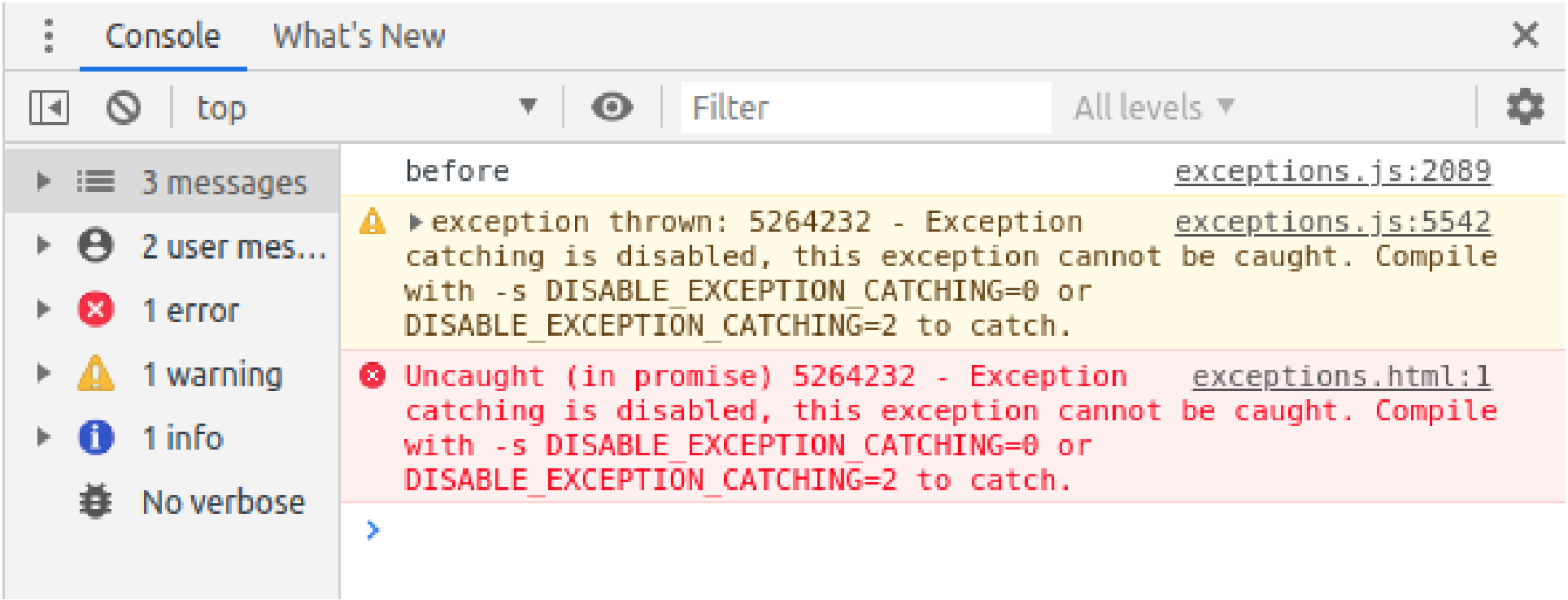
```
$ emrun --no_browser --port 8080 .
```


top
Filter
All levels
20 messages
20 user me…
No errors
No warnings
20 info
No verbose
3 B threads.js:2477
BA threads.js:2477
A threads.js:2477
2 B threads.js:2477
A threads.js:2477
B threads.js:2477
A threads.js:2477
B threads.js:2477
A threads.js:2477
B threads.js:2477
A threads.js:2477
B threads.js:2477
A threads.js:2477
B threads.js:2477
2 A threads.js:2477
A threads.js:2477
>


[305] threads.cpp
[115] threads.html
[263991] threads.js
[225544] threads.wasm
[9117] threads.worker.js

Exceptions

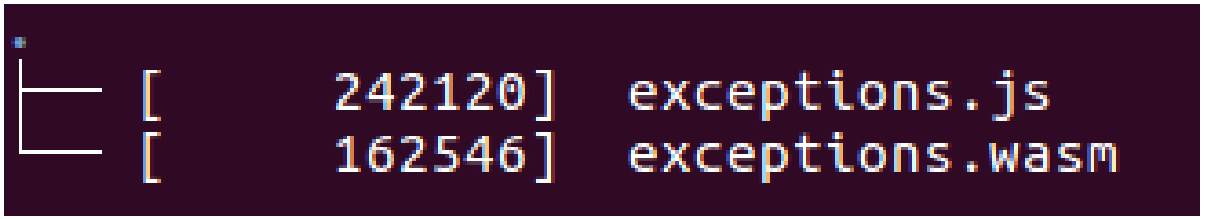
- Compiler transforms throw to abort() by default
- Program will run, but terminates on throw


Console
What's New
top
Filter
All levels
3 messages
2 user mes...
1 error
1 warning
1 info
No verbose
before
exceptions.js:2089
exception thrown: 5264232 - Exception catching is disabled, this exception cannot be caught. Compile with -s DISABLE_EXCEPTION_CATCHING=0 or DISABLE_EXCEPTION_CATCHING=2 to catch.
exceptions.js:5542
Uncaught (in promise) 5264232 - Exception catching is disabled, this exception cannot be caught. Compile with -s DISABLE_EXCEPTION_CATCHING=0 or DISABLE_EXCEPTION_CATCHING=2 to catch.
exceptions.html:1

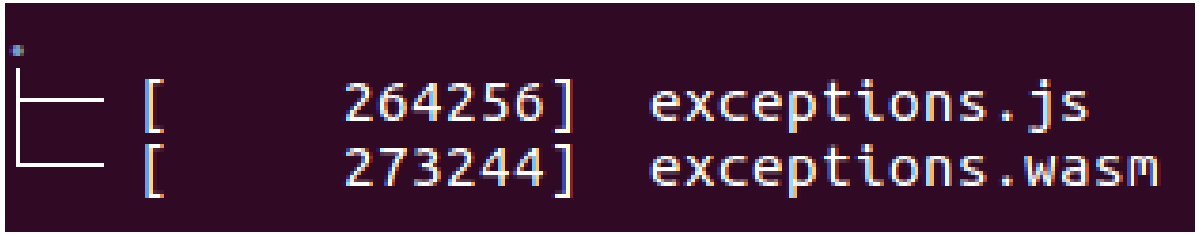
Enable exceptions

```
$ em++ -s DISABLE_EXCEPTION_CATCHING=0 -o exceptions.js \
    exceptions.cpp
```

- Without exceptions:



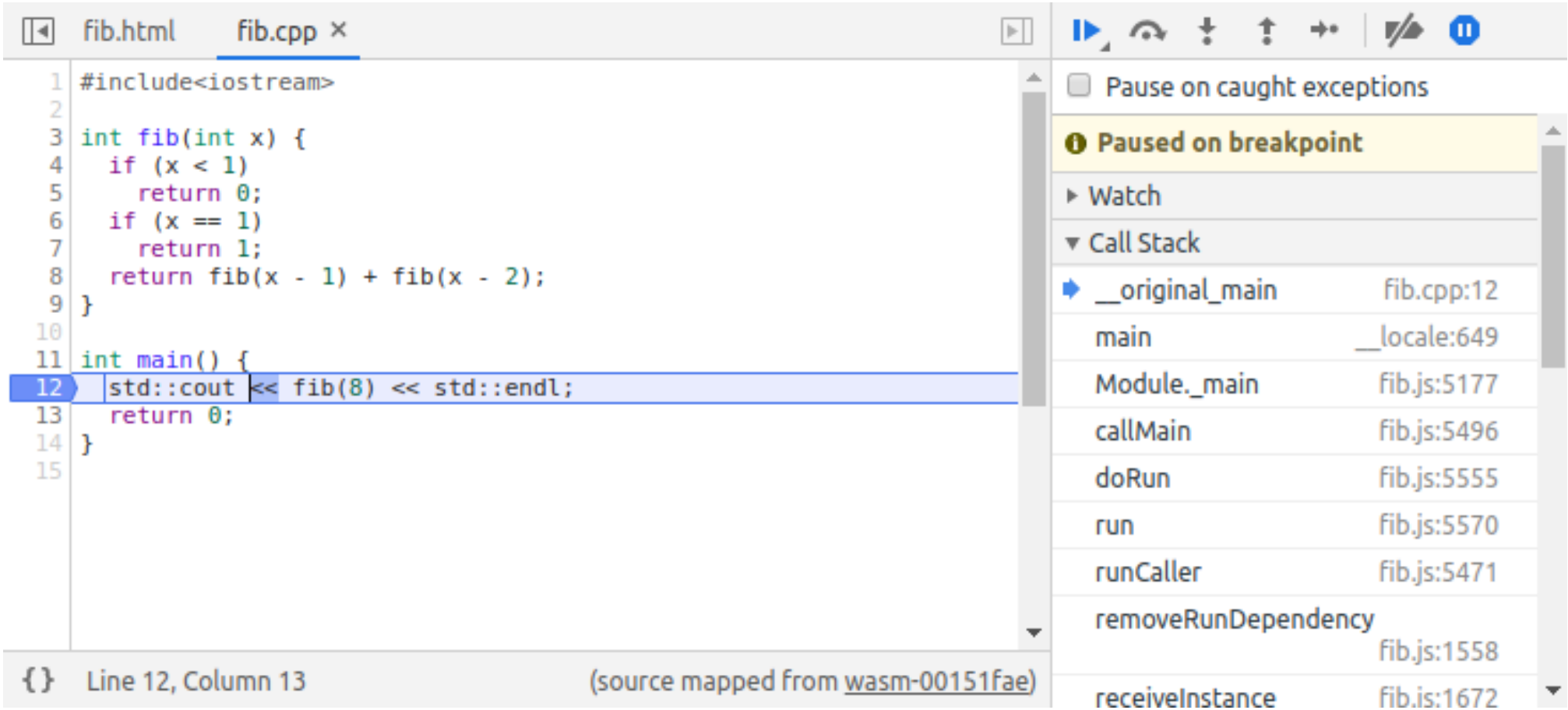
- With exceptions:



Debugging

- Source maps
- Firefox is generally better than Chrome

```
$ em++ fib.cpp -g4 -source-map-base http://localhost:8080 \
  -o fib.js
```


fib.html
fib.cpp
#include<iostream>
int fib(int x) {
if (x < 1)
return 0;
if (x == 1)
return 1;
return fib(x - 1) + fib(x - 2);
}
int main() {
std::cout << fib(8) << std::endl;
return 0;
}
{}
Line 12, Column 13
(source mapped from wasm-00151fae)
Pause on caught exceptions
Paused on breakpoint
Watch
Call Stack
__original_main fib.cpp:12
main __locale:649
Module._main fib.js:5177
callMain fib.js:5496
doRun fib.js:5555
run fib.js:5570
runCaller fib.js:5471
removeRunDependency fib.js:1558
receiveInstance fib.js:1672

Work in progress

- WebAssembly 1.0
  - MVP
- Missing or experimental features
  - Threads
  - Exceptions
  - SIMD support
  - DOM access

Work in progress

- WebAssembly 1.0
  - MVP
- Missing or experimental features
  - Threads
  - Exceptions
  - SIMD support
  - DOM access

- Stay tuned!