WebAssembly 在 Node.js 中的应用

傅晓嵩

目录
Contents

什么是 Assembly

C

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

Compiler

ASM.js

```asm
"use asm";
function add(x, y) {
  x = x | 0;
  y = y | 0;
  return x + y | 0;
}
```

WASM

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 00 61 73 6D 01 00 00 00 00 0C 06 64 79 6C 69 6E  .asm.......dylin
00000010: 6B 80 80 C0 02 00 01 0A 02 60 02 7F 7F 01 7F 60  k..@.....`.....`
00000020: 00 00 02 41 04 03 65 6E 76 0A 6D 65 6D 6F 72 79  ...A..env.memory
00000030: 42 61 73 65 03 7F 00 03 65 6E 76 06 6D 65 6D 6F  Base....env.memo
00000040: 72 79 02 00 80 02 03 65 6E 76 05 74 61 62 6C 65  ry.....env.table
00000050: 01 70 00 00 03 65 6E 76 09 74 61 62 6C 65 42 61  .p...env.tableBa
00000060: 73 65 03 7F 00 03 04 03 00 01 01 06 0B 02 7F 01  se..............
00000070: 41 00 0B 7F 01 41 00 0B 07 2B 03 04 5F 61 64 64  A....A...+.._add
00000080: 00 00 12 5F 5F 70 6F 73 74 5F 69 6E 73 74 61 6E  ..__post_instan
00000090: 74 69 61 74 65 00 02 0B 72 75 6E 50 6F 73 74 53  tiate...runPostS
000000a0: 65 74 73 00 01 09 01 00 0A 20 03 07 00 20 01 20  ets.............
000000b0: 00 6A 0B 03 00 01 0B 12 00 23 00 24 02 23 02 41  .j.......#.$.#.A
000000c0: 80 80 C0 02 6A 24 03 10 01 0B                    ..@.j$....
```

WAST

```
(module
 (import "env" "memory" (memory $0 256 256))
 (import "env" "table" (table 0 0 anyfunc))
 (import "env" "memoryBase" (global $memoryBas
 (import "env" "tableBase" (global $tableBase
 (export "add" (func $add))
 (func $add (param $x i32) (param $y i32) (re
  (return
   (i32.add
    (get_local $x)
    (get_local $y)
   )
  )
 )
)
```

Javascript Engine （V8）

编译工具链

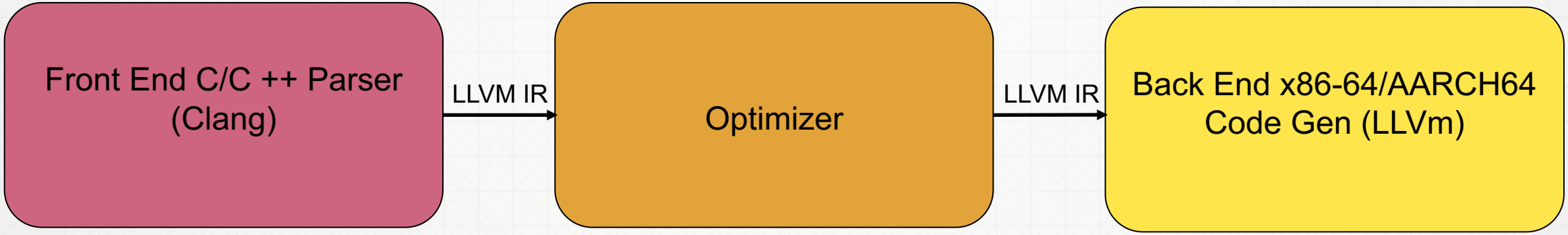
- **LLVM**
  source code -> IR -> target code

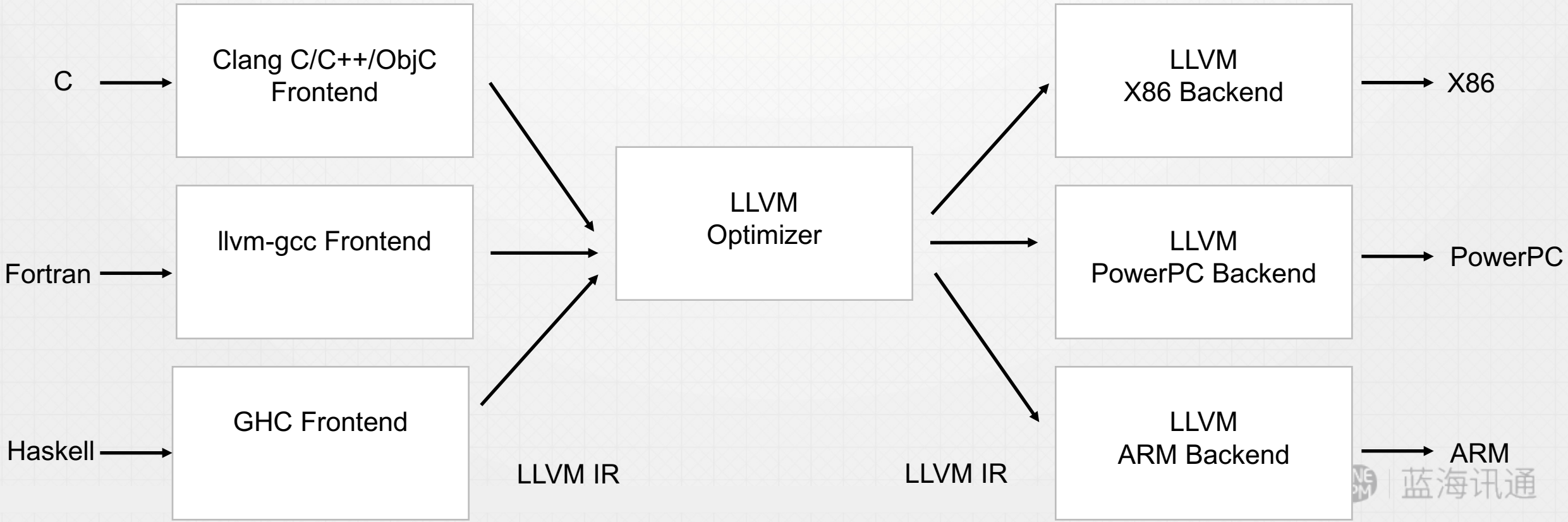
- **emscripten**
  C/C++ -> LLVM -> asm.js
  -> binaryen -> wasm

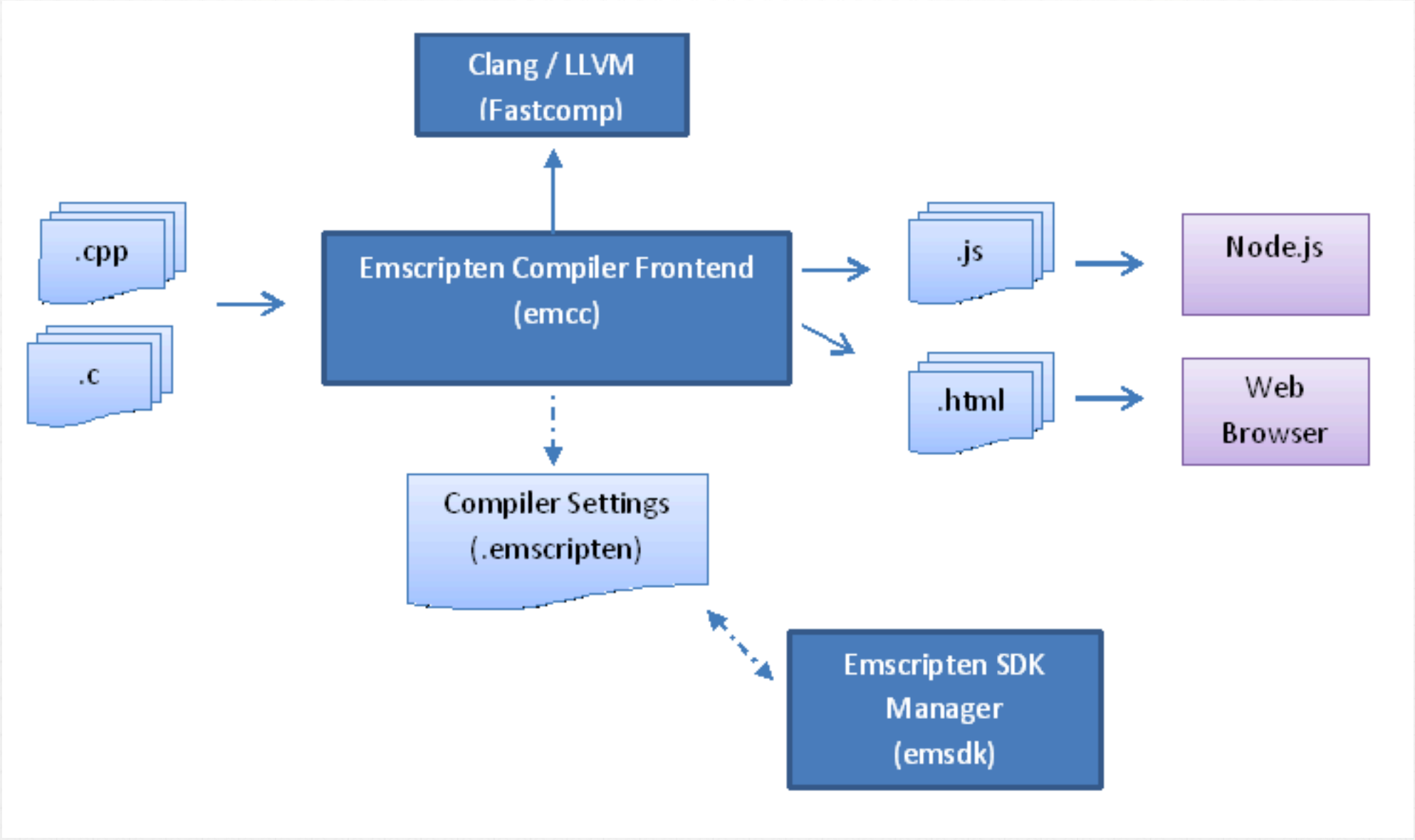
- **binaryen**
  llvm ast / asm.js -> wasm

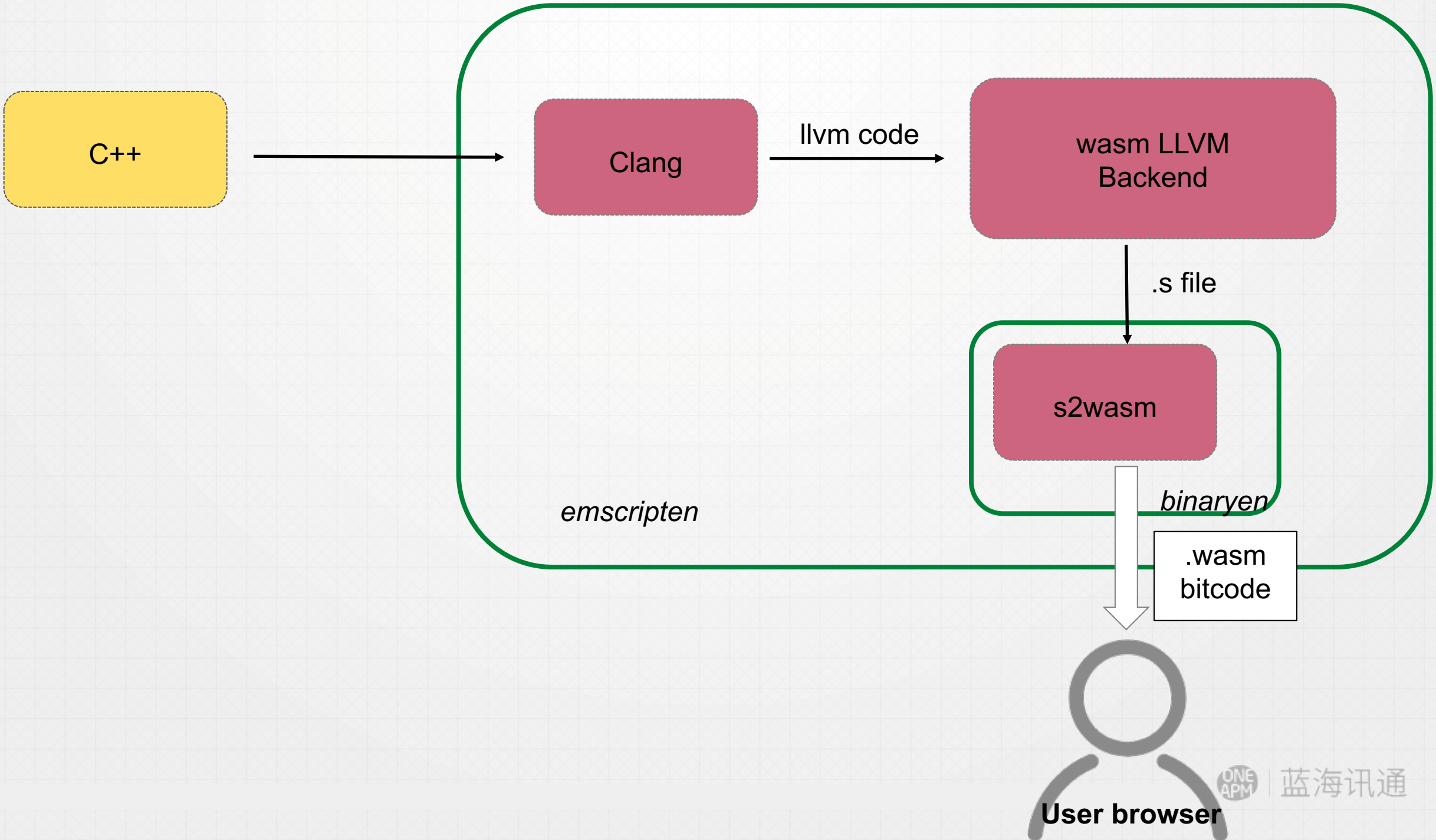
- **wabt**
  wast <-> wasm

LLVM Toolchain






Clang / LLVM
(Fastcomp)
.cpp
.c
Emscripten Compiler Frontend
(emcc)
.js
Node.js
.html
Web
Browser
Compiler Settings
(.emscripten)
Emscripten SDK
Manager
(emsdk)


C++
Clang
llvm code
wasm LLVM
Backend
.s file
s2wasm
binaryen
emscripten
.wasm
bitcode
User browser

WASM

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 00 61 73 6D 01 00 00 00 00 0C 06 64 79 6C 69 6E  .asm.......dylin
00000010: 6B 80 80 C0 02 00 01 0A 02 60 02 7F 7F 01 7F 60  k..@.....`.....`
00000020: 00 00 02 41 04 03 65 6E 76 0A 6D 65 6D 6F 72 79  ...A..env.memory
00000030: 42 61 73 65 03 7F 00 03 65 6E 76 06 6D 65 6D 6F  Base....env.memo
00000040: 72 79 02 00 80 02 03 65 6E 76 05 74 61 62 6C 65  ry.....env.table
00000050: 01 70 00 00 03 65 6E 76 09 74 61 62 6C 65 42 61  .p...env.tableBa
00000060: 73 65 03 7F 00 03 04 03 00 01 01 06 0B 02 7F 01  se..............
00000070: 41 00 0B 7F 01 41 00 0B 07 2B 03 04 5F 61 64 64  A....A...+.._add
00000080: 00 00 12 5F 5F 70 6F 73 74 5F 69 6E 73 74 61 6E  ..__post_instan
00000090: 74 69 61 74 65 00 02 0B 72 75 6E 50 6F 73 74 53  tiate...runPostS
000000a0: 65 74 73 00 01 09 01 00 0A 20 03 07 00 20 01 20  ets.............
000000b0: 00 6A 0B 03 00 01 0B 12 00 23 00 24 02 23 02 41  .j.......#.$.#.A
000000c0: 80 80 C0 02 6A 24 03 10 01 0B                    ..@.j$....
```

wabt

WAST

```
(module
 (import "env" "memory" (memory $0 256 256))
 (import "env" "table" (table 0 0 anyfunc))
 (import "env" "memoryBase" (global $memoryBas
 (import "env" "tableBase" (global $tableBase
 (export "add" (func $add))
 (func $add (param $x i32) (param $y i32) (res
  (return
   (i32.add
    (get_local $x)
    (get_local $y)
   )
  )
 )
)
```

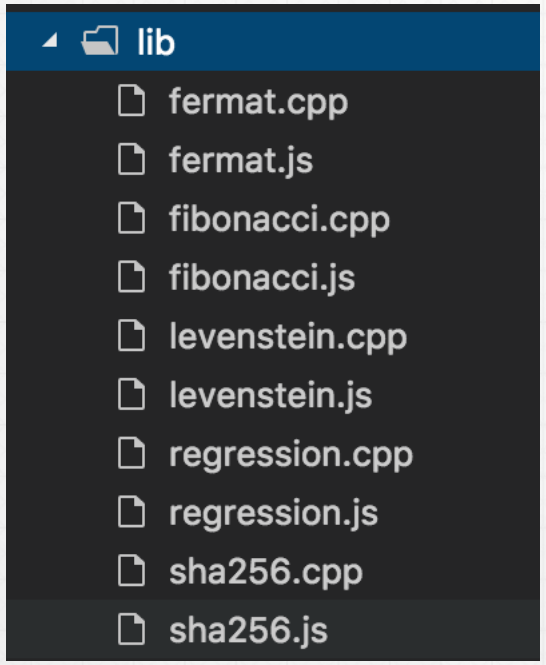
在 Node.js 应用中使用 WebAssembly

```
const fs = require('fs')

const wasmFilePath = './src0/hello.wasm'
const buffer = fs.readFileSync(wasmFilePath)
const importObject = {
  env: {
    memoryBase: 0,
    memory: WebAssembly.Memory({
      initial: 256,
      maximum: 256
    }),
    tableBase: 0,
    table: WebAssembly.Table({
      initial: 0,
      maximum: 0,
      element: 'anyfunc'
    })
  }
}

async function loadWASMAsync(){
  const { module, instance } = await WebAssembly.instantiate(buffer, importObject)
  console.log('1 + 2 = ', instance.exports.add(1, 2))
  return instance.exports
}
function loadWASMSync(){
  const wasmModule = new WebAssembly.Module(buffer)
  const instance = new WebAssembly.Instance(wasmModule, importObject)
  console.log('1 + 2 = ', instance.exports.add(1, 2))
  return instance.exports
}
```

WebAssembly API

- Memory
- Table
- Module
- Instance

与 Node.js C++ Addon 的性能对比



benchmark

```
Levenstein Distance:
   Native x 118,191 ops/sec ±0.94% (83 runs sampled)
   N-API Addon x 228,882 ops/sec ±0.89% (89 runs sampled)
   Web Assembly x 139,091 ops/sec ±3.65% (79 runs sampled)
 Fastest is N-API Addon

Fibonacci:
   Native x 3,158,795 ops/sec ±1.81% (81 runs sampled)
   N-API Addon x 2,731,388 ops/sec ±1.67% (83 runs sampled)
   Web Assembly x 6,615,989 ops/sec ±1.78% (81 runs sampled)
 Fastest is Web Assembly

Fermat Primality Test:
   Native x 1,546,993 ops/sec ±1.03% (83 runs sampled)
   N-API Addon x 1,318,161 ops/sec ±2.49% (79 runs sampled)
   Web Assembly x 2,297,521 ops/sec ±2.99% (76 runs sampled)
 Fastest is Web Assembly

Simple Linear Regression:
   Native x 161,016 ops/sec ±3.50% (76 runs sampled)
   N-API Addon x 3,397 ops/sec ±3.71% (72 runs sampled)
   N-API Addon using TypedArrays x 73,713 ops/sec ±2.58% (75 runs sampled)
   Web Assembly x 22,633 ops/sec ±3.35% (78 runs sampled)
   Web Assembly using TypedArrays x 26,032 ops/sec ±2.24% (77 runs sampled)
 Fastest is Native

SHA256:
   Native x 14,166 ops/sec ±3.12% (78 runs sampled)
   N-API Addon x 63,740 ops/sec ±0.81% (84 runs sampled)
   Web Assembly x 32,916 ops/sec ±0.91% (88 runs sampled)
 Fastest is N-API Addon
```

WebAssembly 在 Node.js 应用中未来可能的使用场景

- 并行编程
- 前后端共享wasm
- AssemblyScript

The End