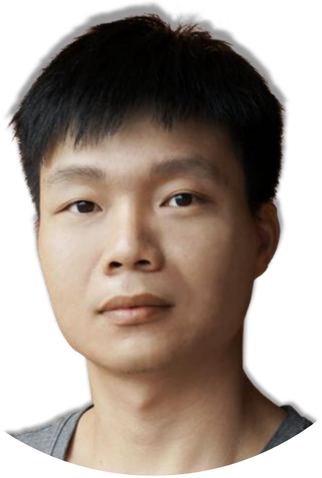
WEB CONF
<腾 讯 前 端 技 术 大 会/>

基于WASM的文件上传加速实践

个人介绍



程海斌@腾讯微云、腾讯文档

商业化前端负责人

2017年开始关注WebAssembly

内容大纲

- 上传应用介绍
- WASM核心原理
- WASM加速实践

上传应用介绍

上传是web应用的基础能力





.....

上传应用介绍

简单


上传图片

复杂


2/3 项任务进行中
2.58 MB/s
取消全部
cmake-3.16.0-rc1-Darwin-x86_64.dmg.zip
34.17 MB 上传
2.58 MB/s
未命名的副本.mov.zip
18.88 MB 上传
googlechrome.dmg.zip
66.24 MB 上传
准备上传

上传应用介绍

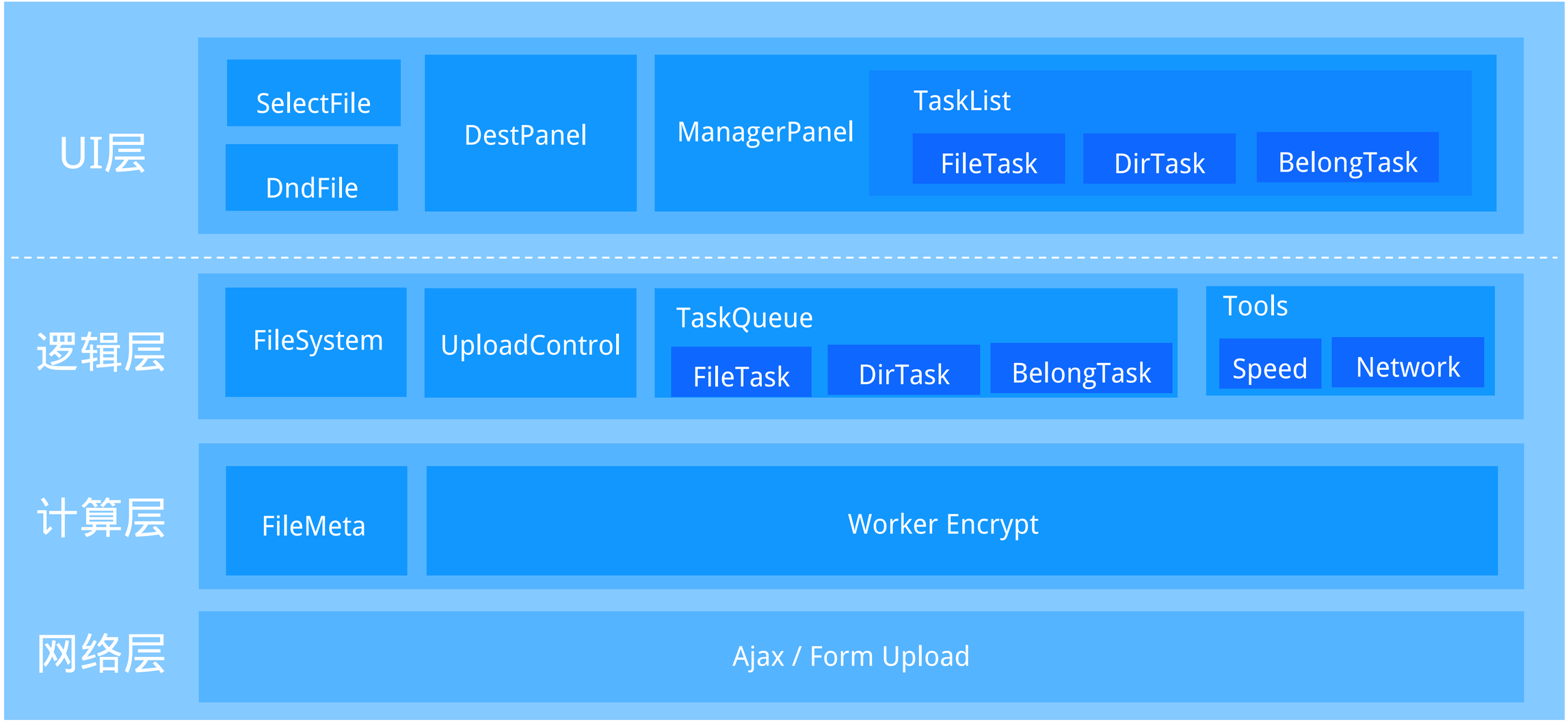
简单

- Form表单
- XHR Upload

复杂

- 分片Blob
- 多通道并行
- 完整性校验
- 秒传、断点续传
- 大文件、大批量
- 异常处理

微云上传前端架构


UI层
SelectFile
DndFile
DestPanel
ManagerPanel
TaskList
FileTask
DirTask
BelongTask
逻辑层
FileSystem
UploadControl
TaskQueue
FileTask
DirTask
BelongTask
Tools
Speed
Network
计算层
FileMeta
Worker Encrypt
网络层
Ajax / Form Upload

上传应用介绍

关键：计算文件的数字签名（SHA1）

WASM是什么

C/C++

Rust

GO

TypeScript

...

IR → WASM → Browser / Out of Browser / Into other Languages

高效

快速

安全

标准

可移值

WASM MVP

- 标准Web API

  WebAssembly.Module、 WebAssembly.compile、 WebAssembly. Instantiate …

- 模块结构

  types、funcs、mems、tables、data、imports、exports…

- 二进制格式、文本格式

  LEB128编码、S文本表达式

- 指令、语法规则

  local.get、local.set、i32.add、i32.load、i32.store …，基于堆栈机执行

- 模块检查、执行

  检查通过、优化编译、转机器码执行

https://webassembly.github.io/spec/core/index.html

一个简单例子 – C源码

编译：emcc –O3 add.c -o add.wasm

```c
#include <emscripten/emscripten.h>

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

一个简单例子 - wat

wasm2wat add.wasm

```
(module
  (type (;0;) (func (param i32 i32) (result i32)))
  (func (;0;) (type 0) (param i32 i32) (result i32)
    local.get 0
    local.get 1
    i32.add)
  (export "_add" (func 0)))
```

wasm-objdump add.wasm -x

```
Section Details:

Type[1]:
 - type[0] (i32, i32) -> i32
Function[1]:
 - func[0] sig=0 <_add>
Export[1]:
 - func[0] <_add> -> "_add"
Code[1]:
 - func[0] size=7 <_add>
```

一个简单例子 - 二进制

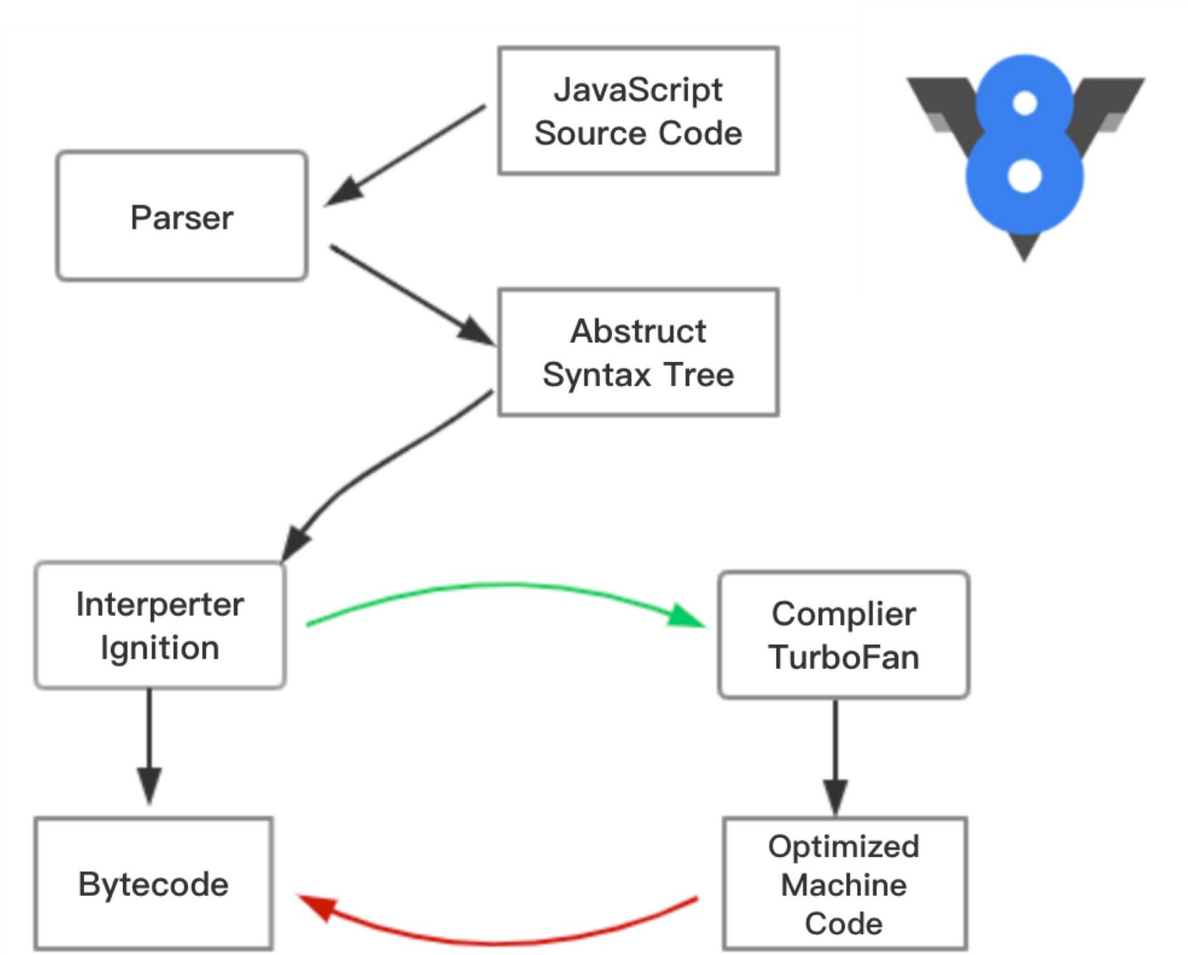
```
HIBINCHENG-MB1:add2 hibincheng$ hexdump -C add.wasm
00000000  00 61 73 6d 01 00 00 00  01 87 80 80 80 00 01 60  |.asm...........`|
00000010  02 7f 7f 01 7f 03 82 80  80 80 00 01 00 04 84 80  |................|
00000020  80 80 00 01 70 00 00 05  83 80 80 80 00 01 00 01  |....p...........|
00000030  06 81 80 80 80 00 00 07  90 80 80 80 00 02 06 6d  |...............m|
00000040  65 6d 6f 72 79 02 00 03  61 64 64 00 00 0a 8d 80  |emory...add.....|
00000050  80 80 00 01 87 80 80 80  00 00 20 01 20 00 6a 0b  |.......... . .j.|
00000060
```

一个简单例子 – JS api调用

```
<script>
  fetch('./add.wasm').then((response) => {
    return response.arrayBuffer()
  }).then((buffer) => {
    return WebAssembly.instantiate(buffer, {})
  }).then((res) => {
    console.log('3+4=', res.instance.exports._add(3, 4))
  })
</script>
```

WASM为什么快

JS执行链路

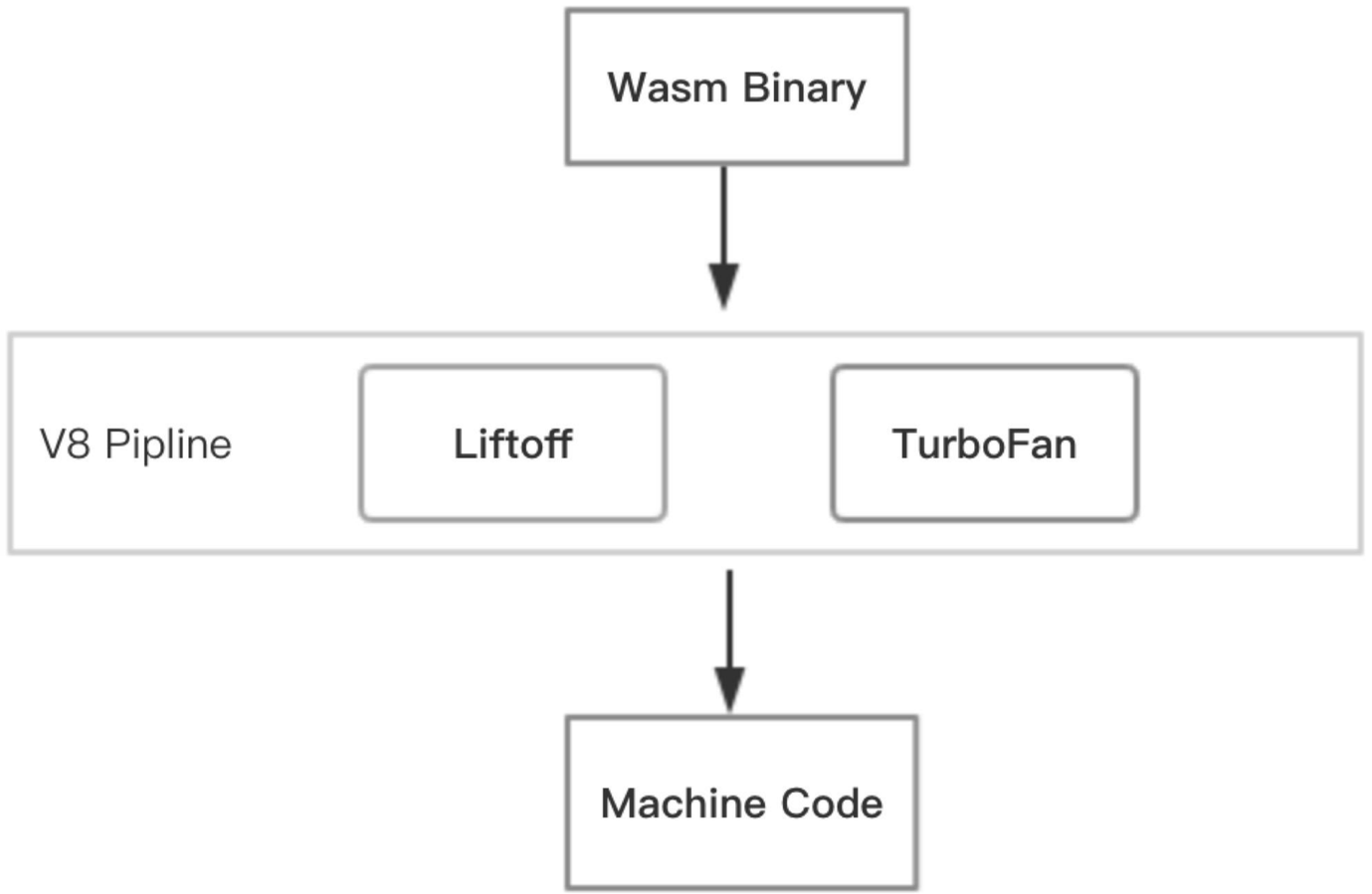



```
let sum = 0
let numList = [1, 2, 3, 4, 5]
function add(a, b) {
    return a + b
}
numList.forEach((num) => {
    sum = add(sum, num)
})
add(sum, '6')
```

去优化

WASM为什么快

wasm执行链路



WASM与JS交互

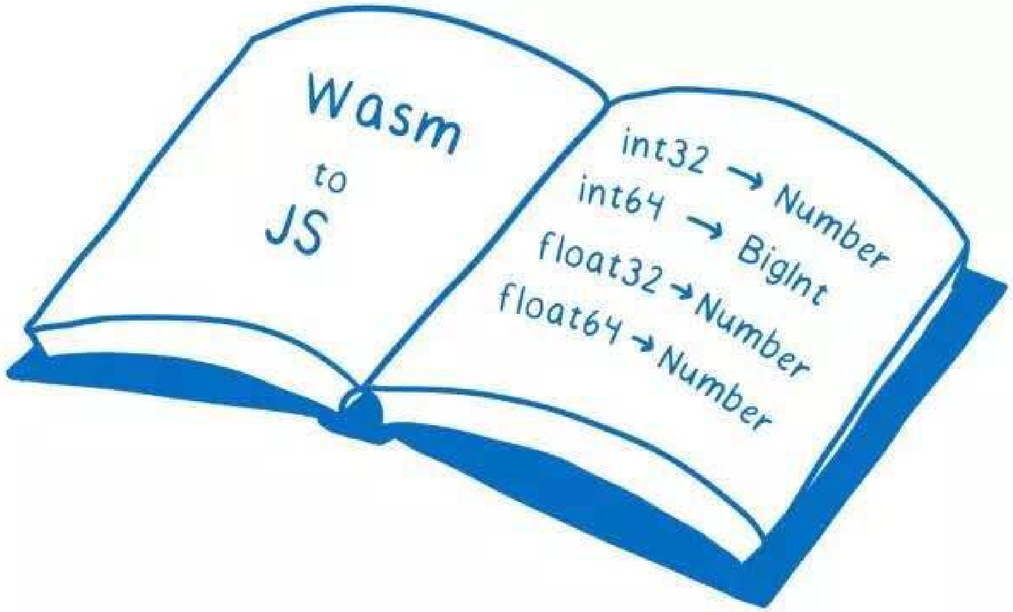
importObject导入到wasm

```
const instance = await WebAssembly.instantiate(module, importObject)
instance.exports._add(3, 4)
```

exports导出供js使用

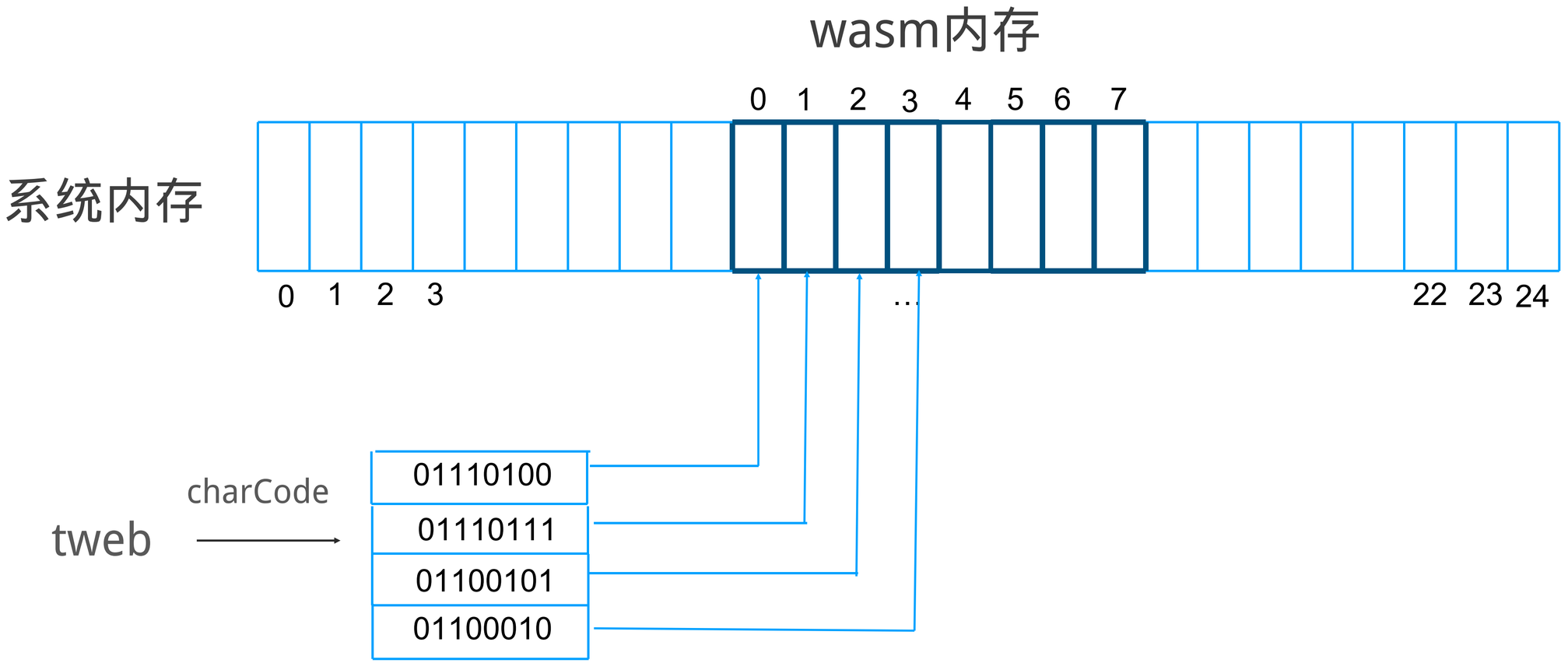
WASM与JS数据传递

目前只支持number传递



WASM与JS数据传递

借助共享线性内存传递 - ArrayBuffer



WASM与JS数据传递

js胶水代码实现

```
let wasmMod = res.instance.exports
let heapu8 = new Uint8Array(wasmMod.memory.buffer)
let str = 'hello tweb'
let ptr = wasmMod.stackAlloc()
for(let i = 0; i < str.length; i++) {
  heapu8[ptr + i] = str.charCodeAt(i)
}
heapu8[ptr+str.length + 1] = 0
wasmMod._sayHello(ptr)
wasmMod.free(ptr)
```

```
Uint8Array(16777216)
    5244446: 0
    5244447: 0
    5244448: 104
    5244449: 101
    5244450: 108
    5244451: 108
    5244452: 111
    5244453: 32
    5244454: 116
    5244455: 119
    5244456: 101
    5244457: 98
    5244458: 0
    5244459: 0
```

使用Emscripten构建应用

Html5.h
File system
Pthread
Liner Memory
WebIDL
...

使用Emscripten WebIDL binding

tweb.cc

```cpp
class TWeb{
public:
    char* title = "tencent web";
    int size = 1024;
    char* sayHello() {
        return "hello tweb";
    }
};
```

tweb.idl

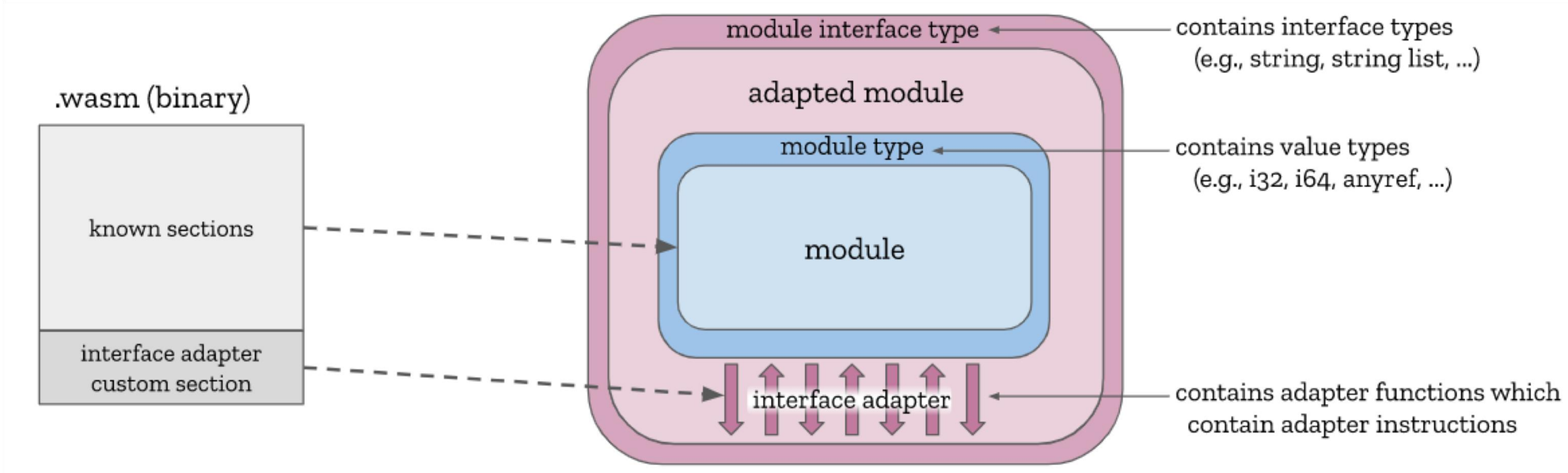
```
interface TWeb {
    void TWeb();
    attribute DOMString title;
    attribute long size;
    DOMString sayHello();
};
```

tweb.html

```html
<script>
  Module.onRuntimeInitialized = function() {
      var tweb = new Module.TWeb()
      console.log(tweb.title, tweb.size, tweb.sayHello())
  }
</script>
```

WASM Interface Types(提案)

Wasm与宿主环境或其它模块交互，不同类型之间转换



WASM Interface Types(提案)

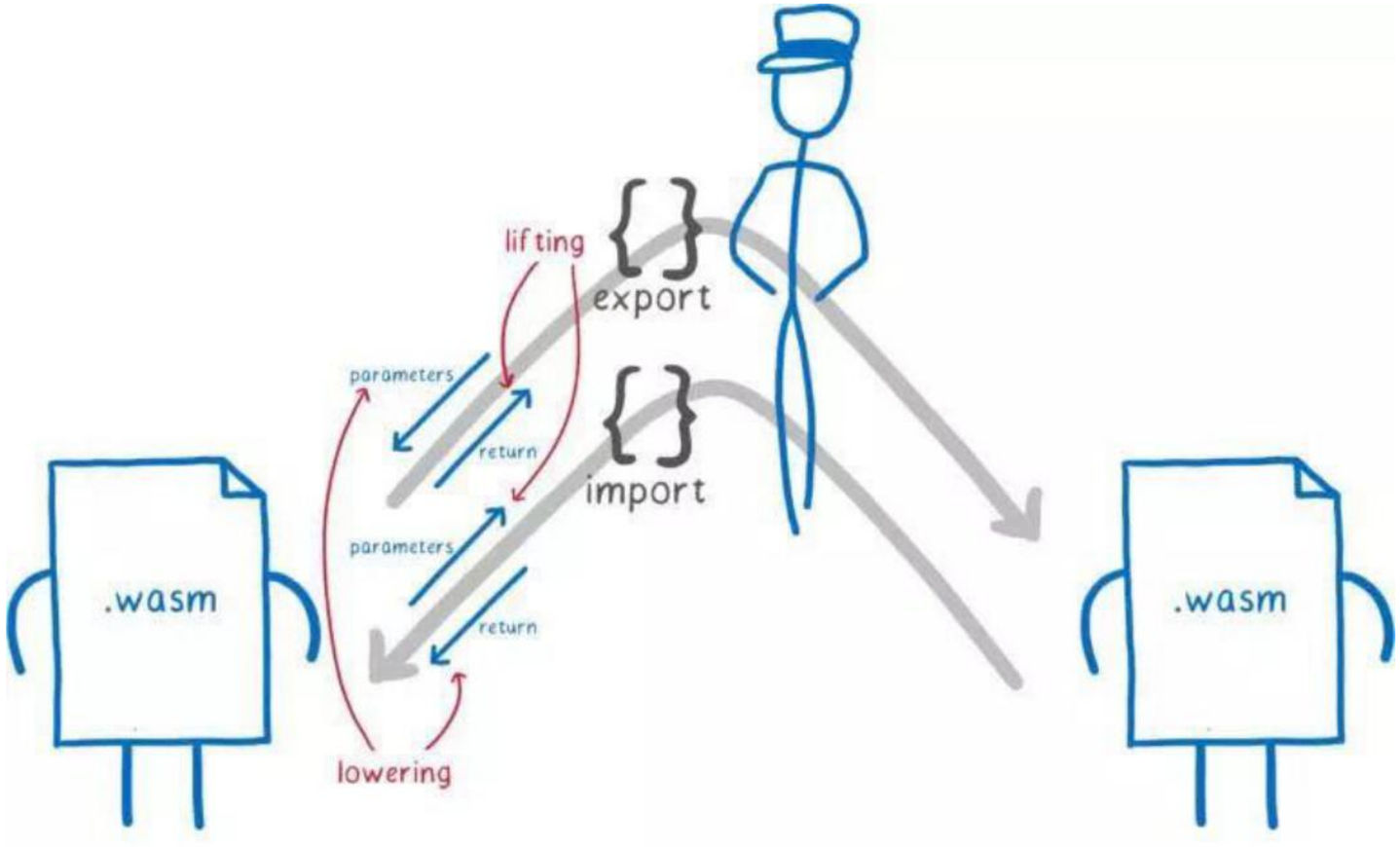
定义更多指令，引擎内完成类型转换

```
(module
  (memory (export "mem") 1)
  (func (import "" "log_") (param i32 i32))
  ...
  (@interface func $log (import "" "log")
  (@interface implement (import "" "log_"
    arg.get $ptr
    arg.get $len
    memory-to-string "mem"
    call-import $log
  )
)
```

```
(module
  (memory (export "mem") 1)
  (func (export "malloc") (param i32) (result i32) ...)
  (func (export "free") (param i32) ...)
  (func (export "log_") (param i32 i32) ...)
  (@interface func (export "log") (param $str string)
    arg.get $str
    string-to-memory "mem" "malloc"
    call-export "log_"
  )
)
```

WASM Interface Types(提案)

Interface adapter



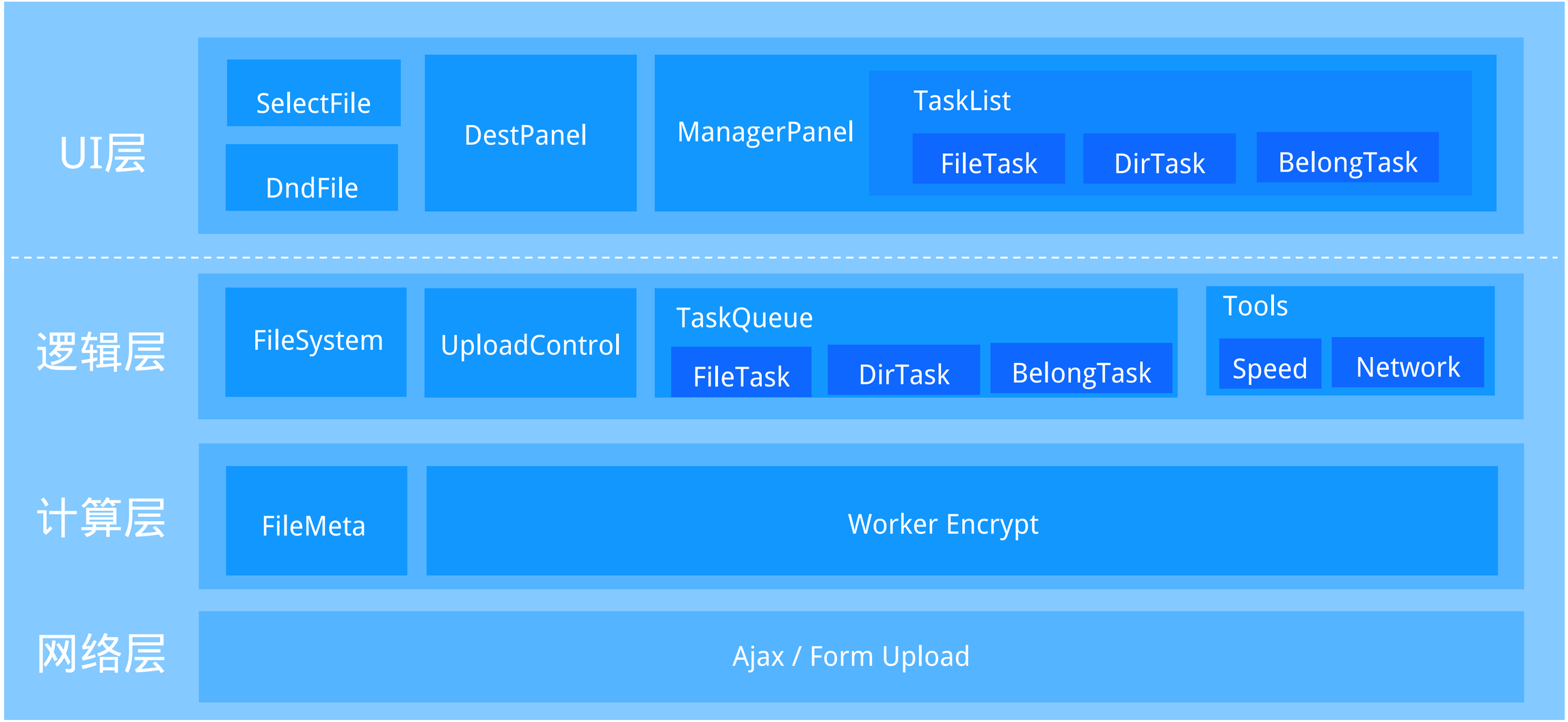
WASM Interface Types(提案)

期待更优雅的调用

```
const {instance} =
  await WebAssembly.instantiateStreaming(
    fetch('qrcode.wasm')
  )

instance.exports.encode('helloworld', canvas, {
  version: 2,
  size: 100,
  level: 8
})
```

WASM加速实践

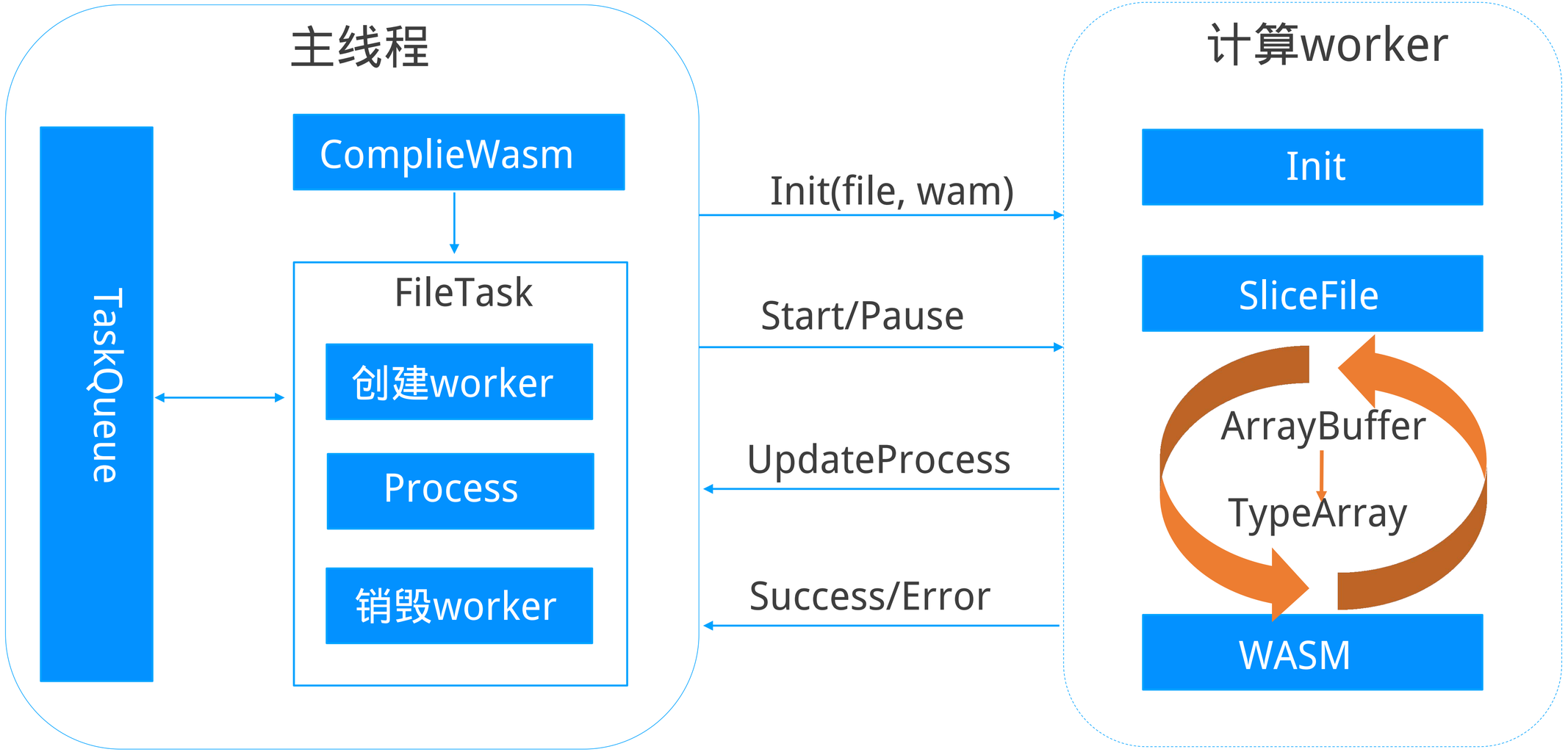

UI层
SelectFile
DndFile
DestPanel
ManagerPanel
TaskList
FileTask
DirTask
BelongTask
逻辑层
FileSystem
UploadControl
TaskQueue
FileTask
DirTask
BelongTask
Tools
Speed
Network
计算层
FileMeta
Worker Encrypt
网络层
Ajax / Form Upload

WASM加速实践


UI层
SelectFile
DndFile
DestPanel
ManagerPanel
TaskList
FileTask
DirTask
BelongTask
逻辑层
FileSystem
UploadControl
TaskQueue
FileTask
DirTask
BelongTask
Tools
Speed
Network
WASM用武之地
高性能计算
Sha库复用
计算层
FileMeta
Worker Encrypt
网络层
Axios / Form Upload

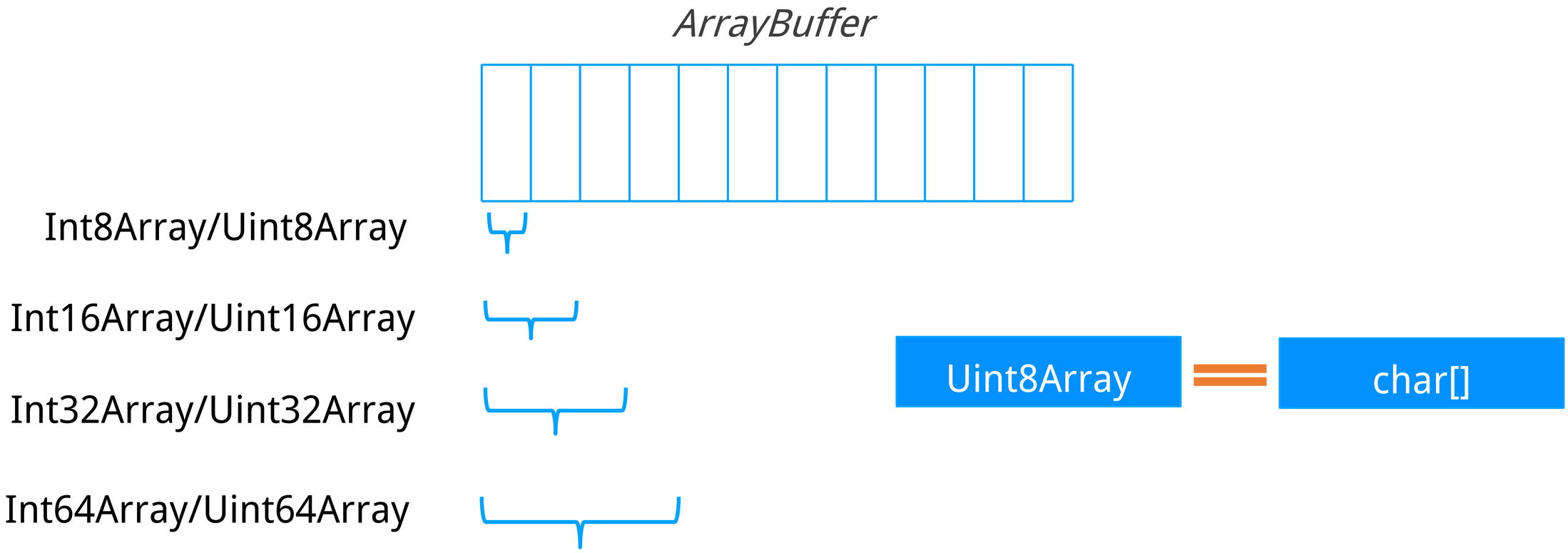
WASM加速实践

计算模型



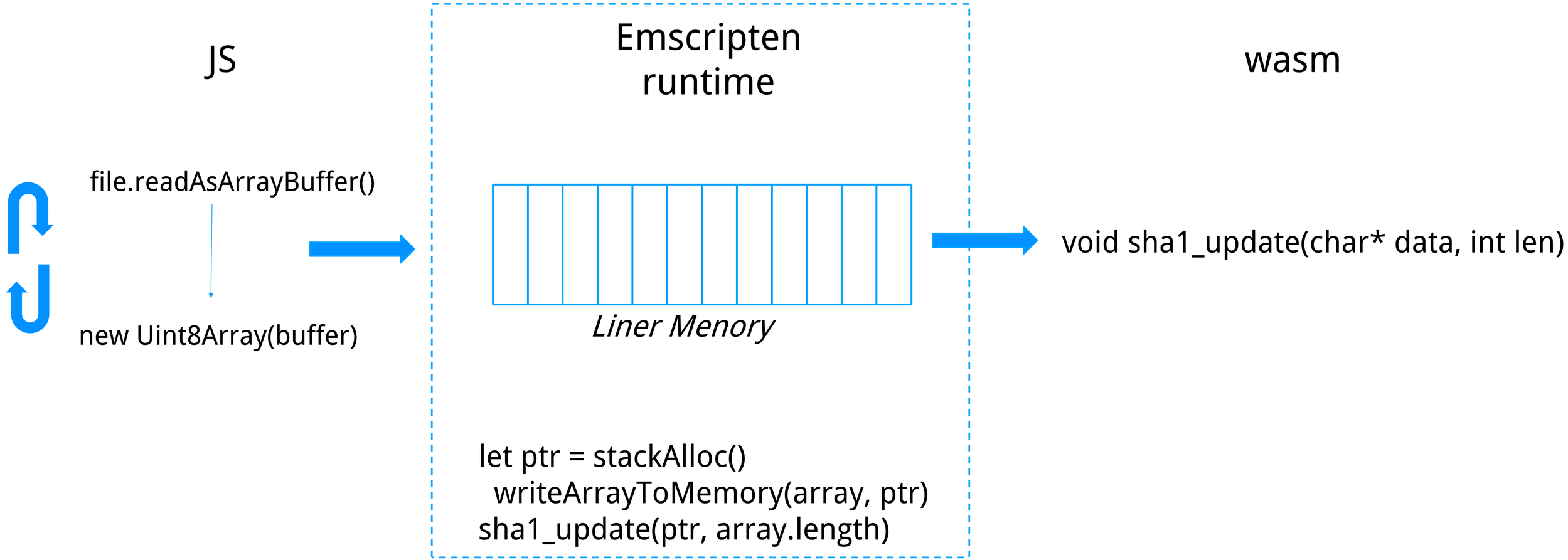
WASM加速实践

认识TypeArray



WASM加速实践

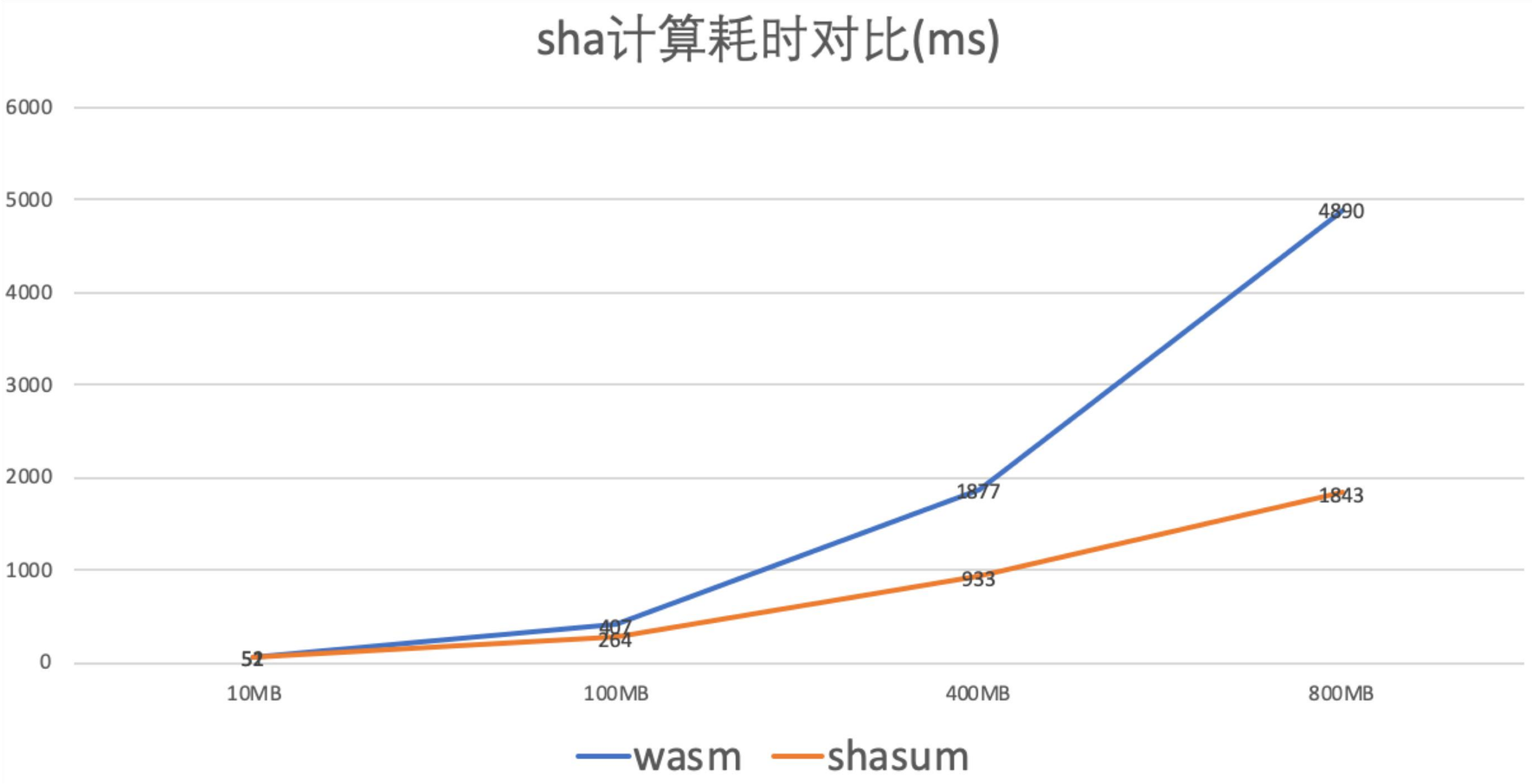
大文件基于分片计算



WASM加速实践

测试机：

MacBook Pro i5



WASM加速实践

Emscripten runtime 中数据转换耗时分析

	1B	1KB	10MB	100MB
arrayToC	0.08ms	0.09ms	7ms	90ms
stringToC	0.18ms	0.33ms	85ms	830ms

WASM加速实践

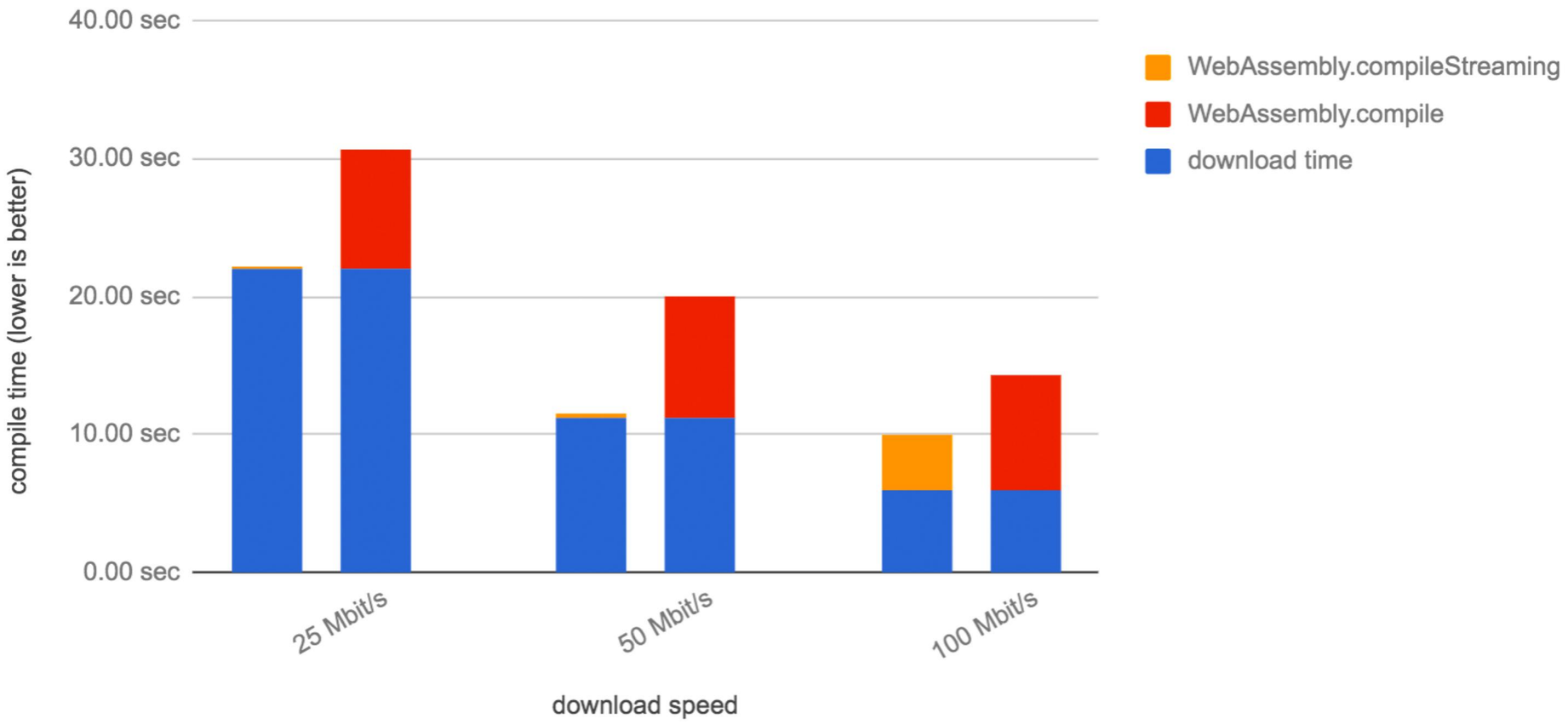
- Wasm纯计算速度能接近native
- 数据传递占总耗时的 1/3
- 数据源最好以ArrayBuffer获取
- String需要转换步骤多，性能不好

WASM加速实践

优化初始化路径 – 减少网络加载、 缓存wasm module

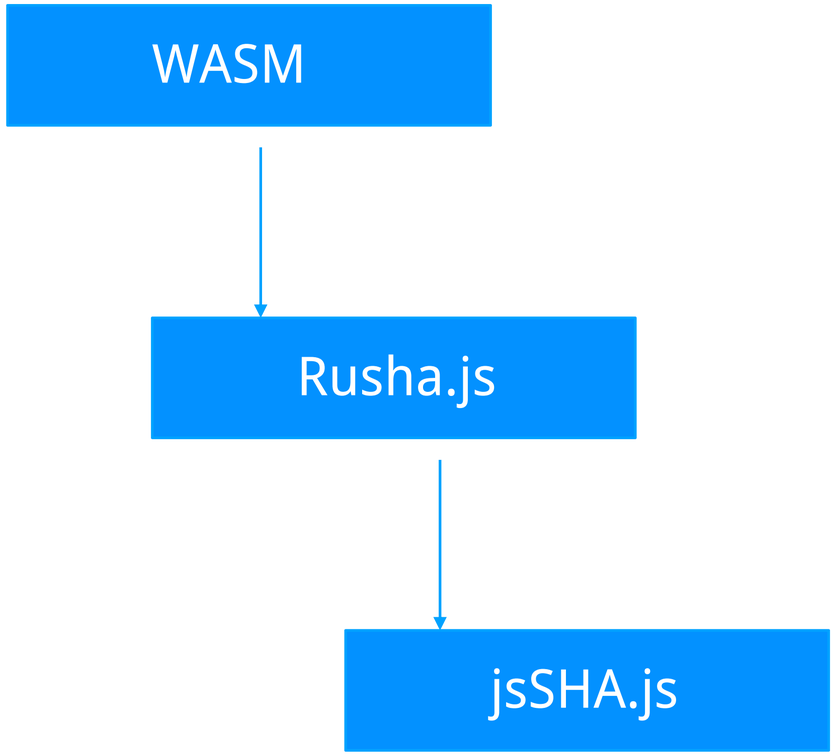
WASM加速实践

通用的流式编译： compileStreaming、 instantiateStreaming

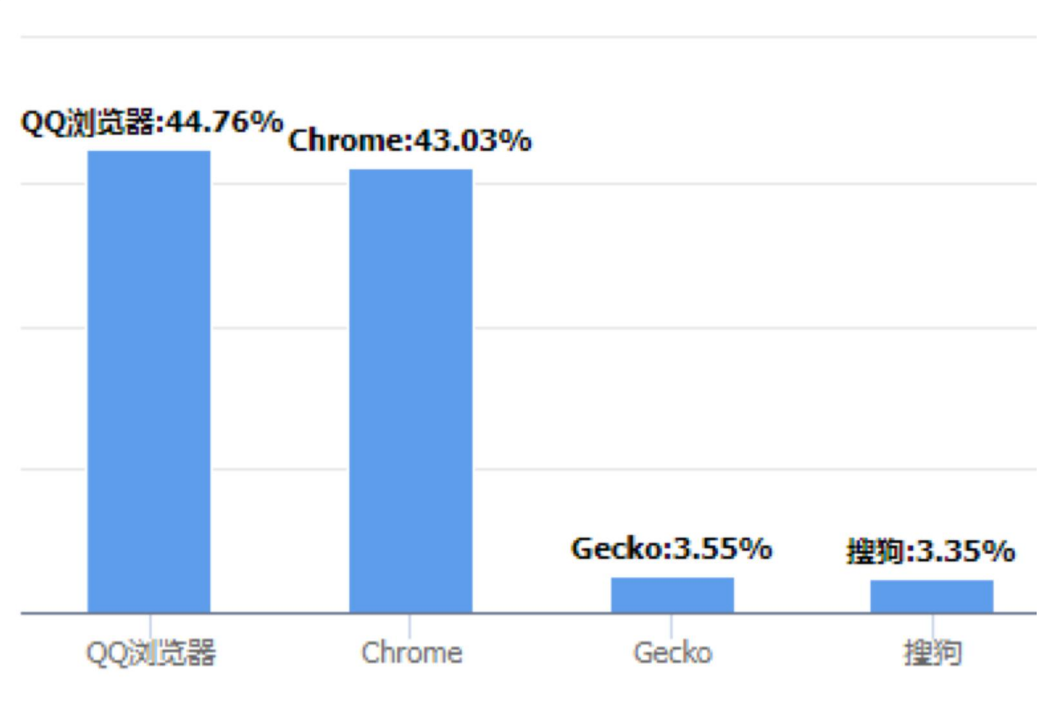


WASM加速实践

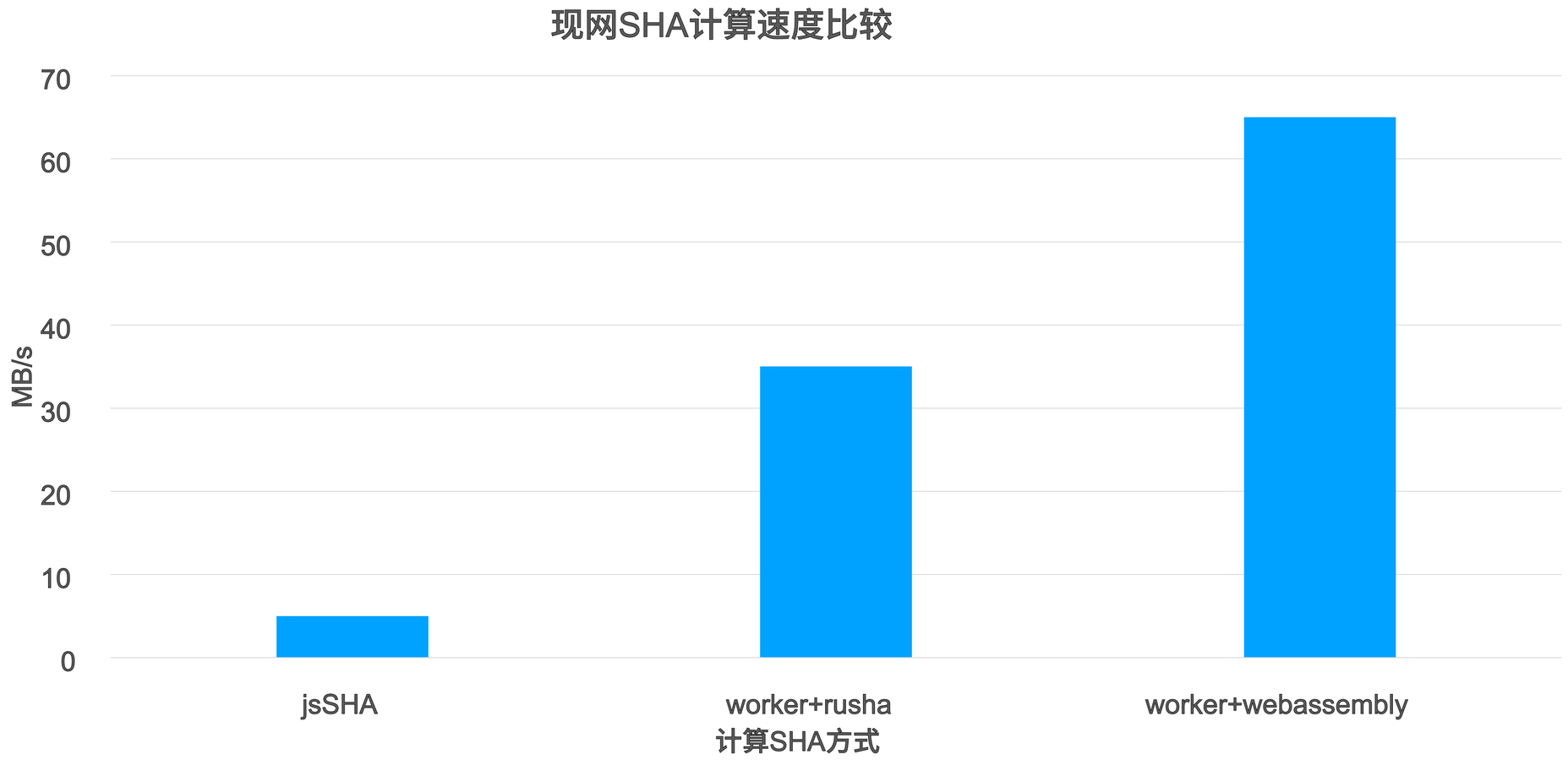
降级方案



使用微云的浏览器分布



WASM加速实践 – 成果


现网SHA计算速度比较
70
60
50
40
30
20
10
0
MB/s
jsSHA
worker+rusha
worker+webassembly
计算SHA方式

WASM加速实践 - 总结

- 找出性能瓶颈运用wasm
- 关注整体优化
- 拥抱wasm，做好降级体验

TWEB CONF

<腾 讯 前 端 技 术 大 会/>

<Thanks/>

主办方 Organizer Tencent TWeb