{ fib:JS }

从钻木取火到航空母舰

— fiber 引擎的四次重构

@响马

大家好，我是响马

什么是 fibjs?

- 服务器端 JavaScript 开发
- 基于 Google v8 引擎构建
- 完全非阻塞，fiber 驱动，拒绝回调，类似 goroutine
- CommonJS 模块系统
- 100,000+ 行 c/c++ 代码，充分挖掘多核性能
- 38 个基本模块，覆盖常见服务器应用场景

{ fib:JS }

来看一段常用的数据库操作

nodejs

```
connection.query(sql, function(err, rows, fields) {
    if (err)
        throw err;

    if (rows[0].type === "something")
        connection.query(sql1, function(err, rows, fields) {
            if (err)
                throw err;

            console.log('The solution is: ', rows[0].solution);
        });
    else
        console.log('The solution is: ', rows[0].solution);
});
```

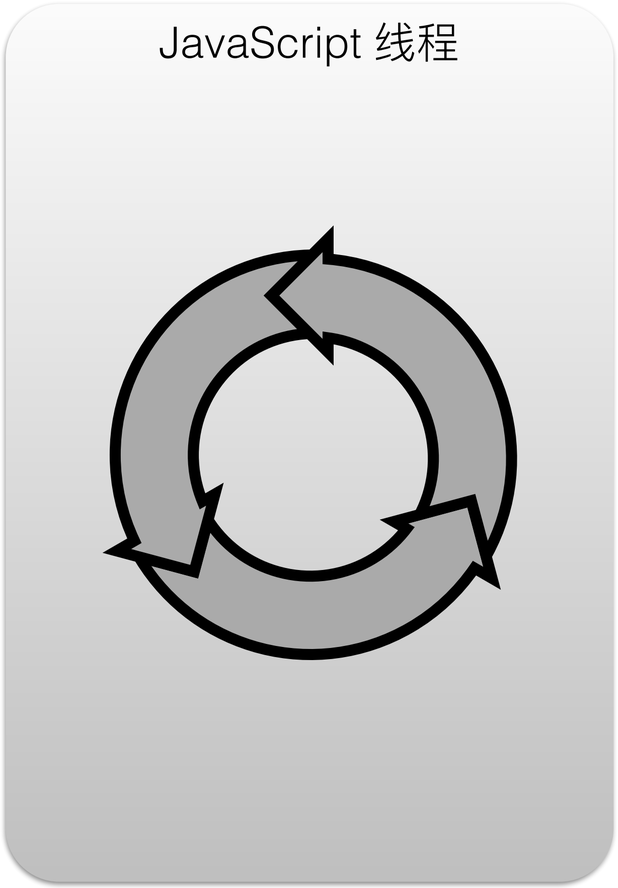
fibjs

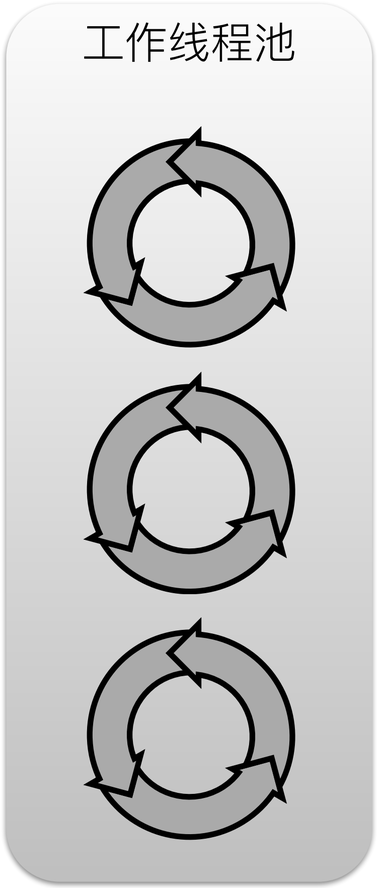
```
var rows = connection.execute(sql);
if (rows[0].type === "something")
    rows = connection.execute(sql1);
console.log('The solution is: ', rows[0].solution);
```

{ fib:JS }

fibs 是如何做到的?

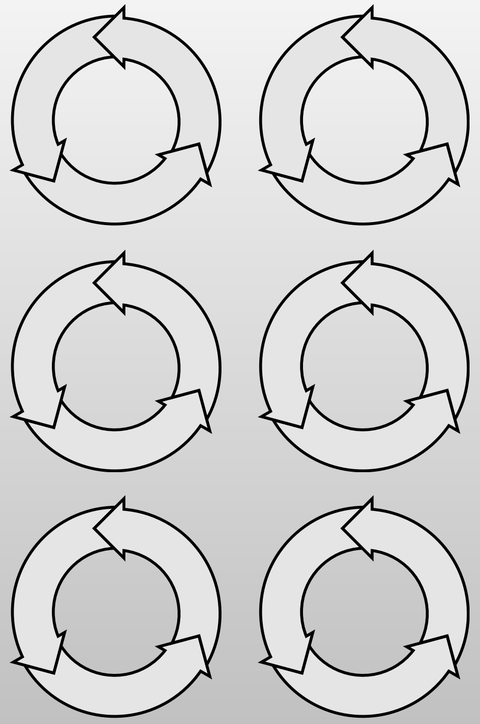
nodejs 有两种类型的线程



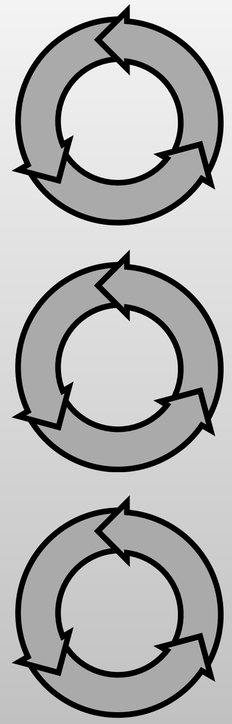


fibjs 有三种类型的线程

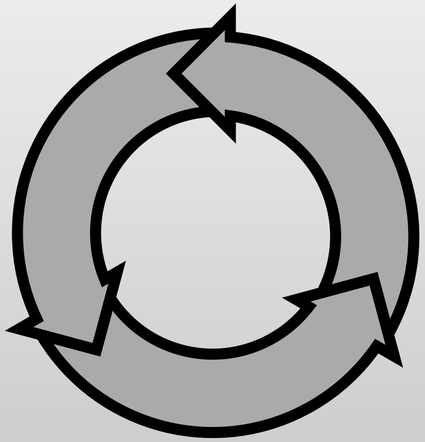
JavaScript 线程



工作线程池

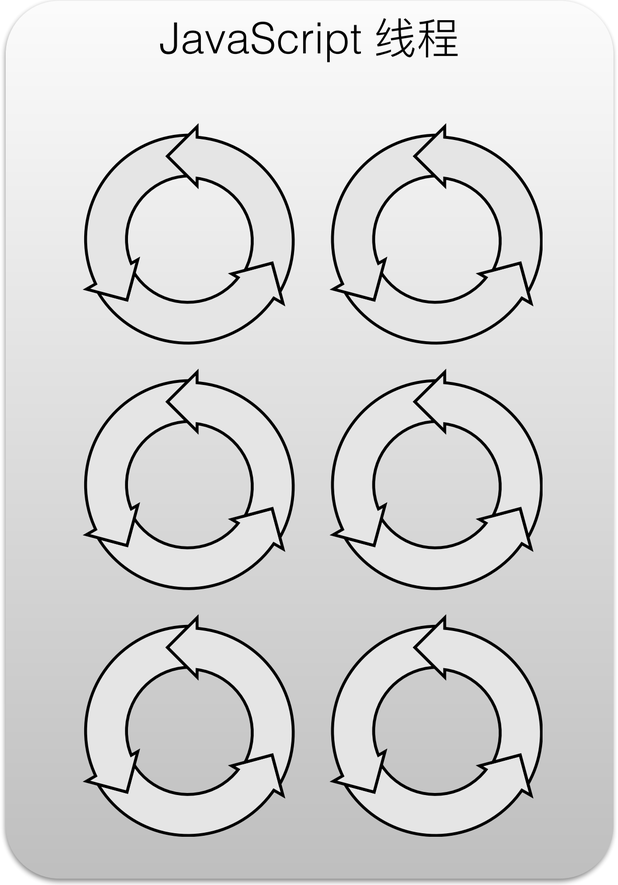


异步 io 线程



{ fib:JS }

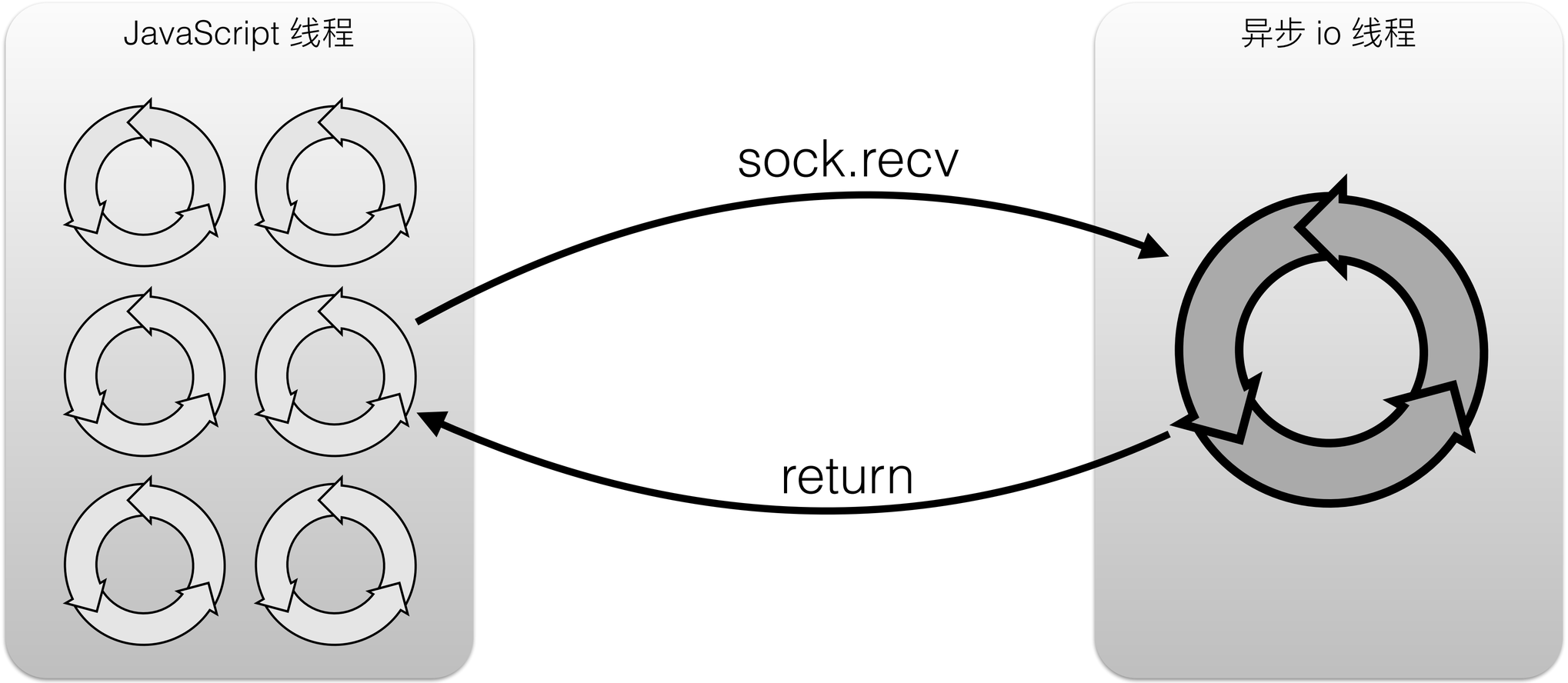
JavaScript 线程是 fibjs 的主线程



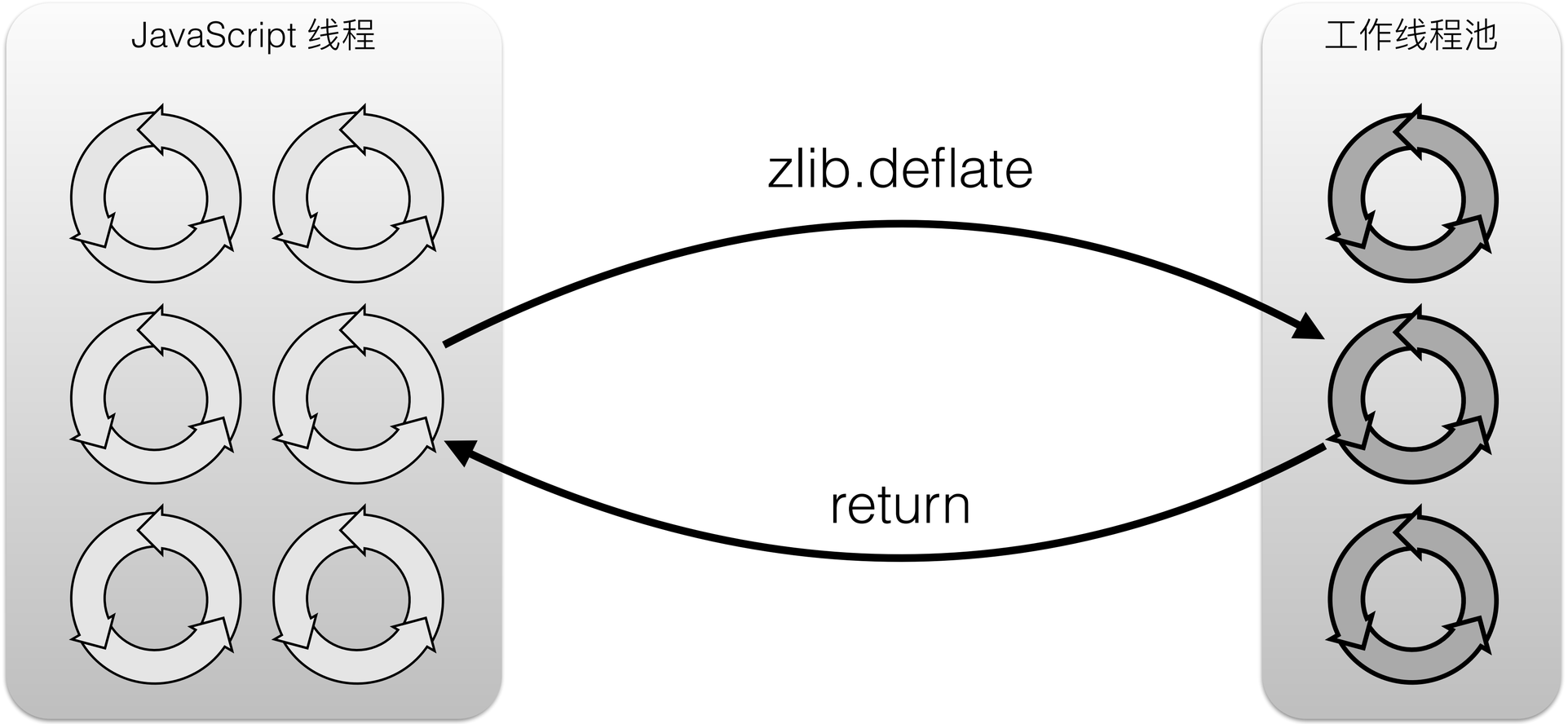
- 一个 fibjs 进程只有一个 JavaScript 线程
- JavaScript 线程内会运行多个 fiber
- JavaScript 代码在 fiber 内运行
- 同一时刻只会有一个 fiber 激活
- 当前 fiber 休眠时其它 fiber 才会恢复
- fiber 不释放就会把 JavaScript 塞住

{ fib:JS }

socket 操作会委托给异步 io 线程处理

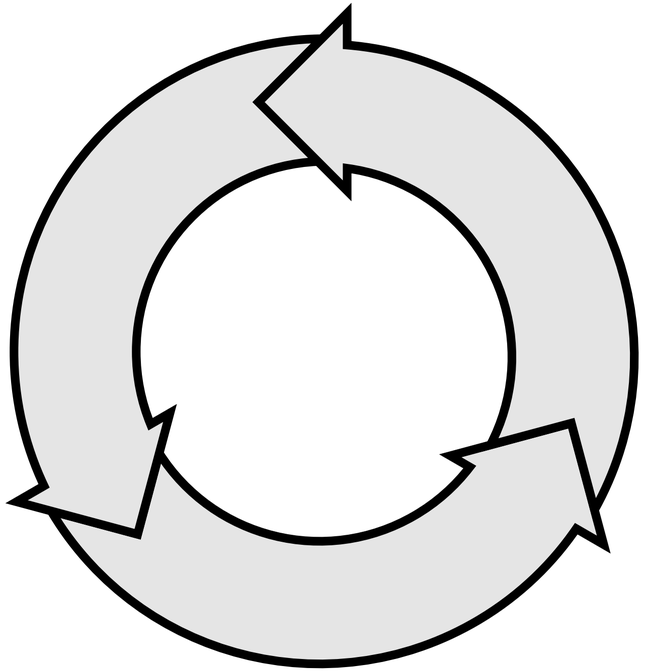


工作线程池分担了大量的繁重运算



{ fib:JS }

最终避免 fiber 阻塞导致 JavaScript 被挂起



{ fib:JS }

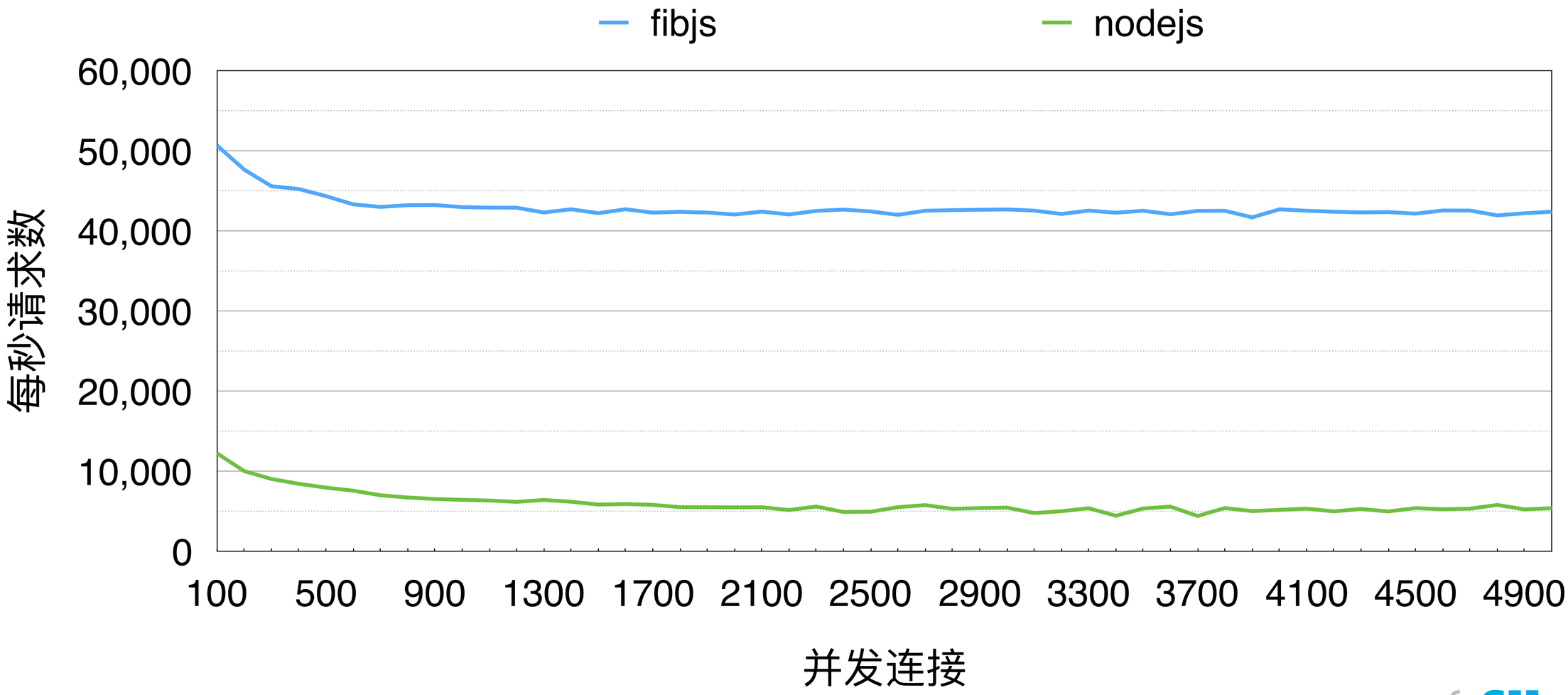
fibjs 的并发：轻量的用户空间线程 fiber

- 操作系统线程的切换成本昂贵
- fiber 是在应用级的线程系统
- 运行现场完整保护，对应用开发透明
- 可以直接使用的编程逻辑，包括 try/catch
- 基于堆栈切换现场，模块调用和返回更高效
- 非抢先，无需内存级锁，并发逻辑简单

{ fib:JS }

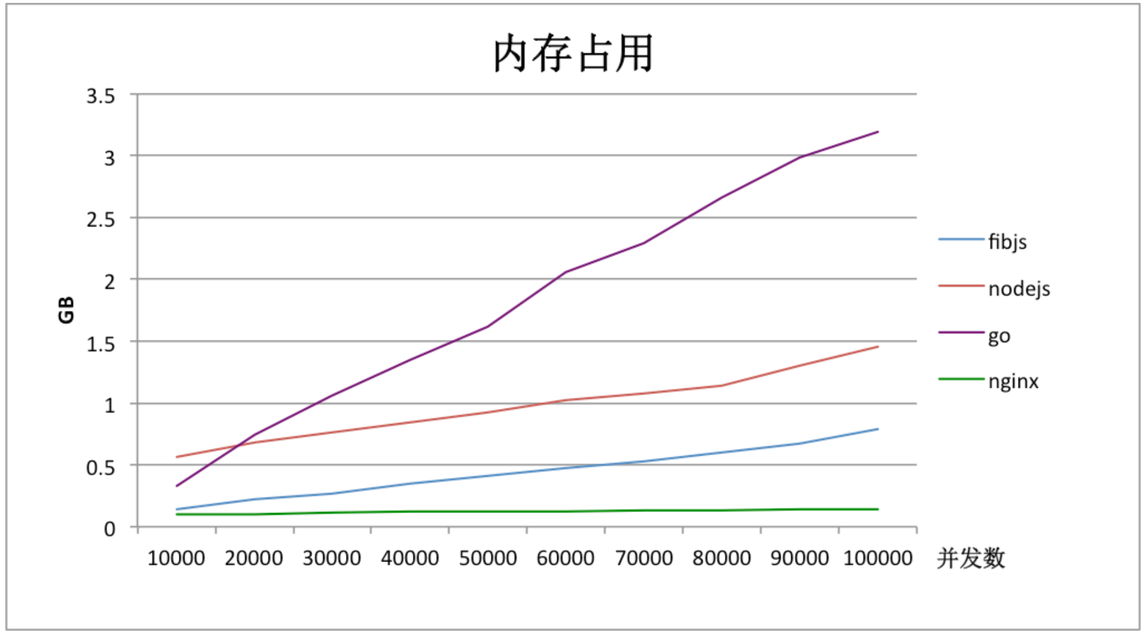
以上是最初的 fiber 模型

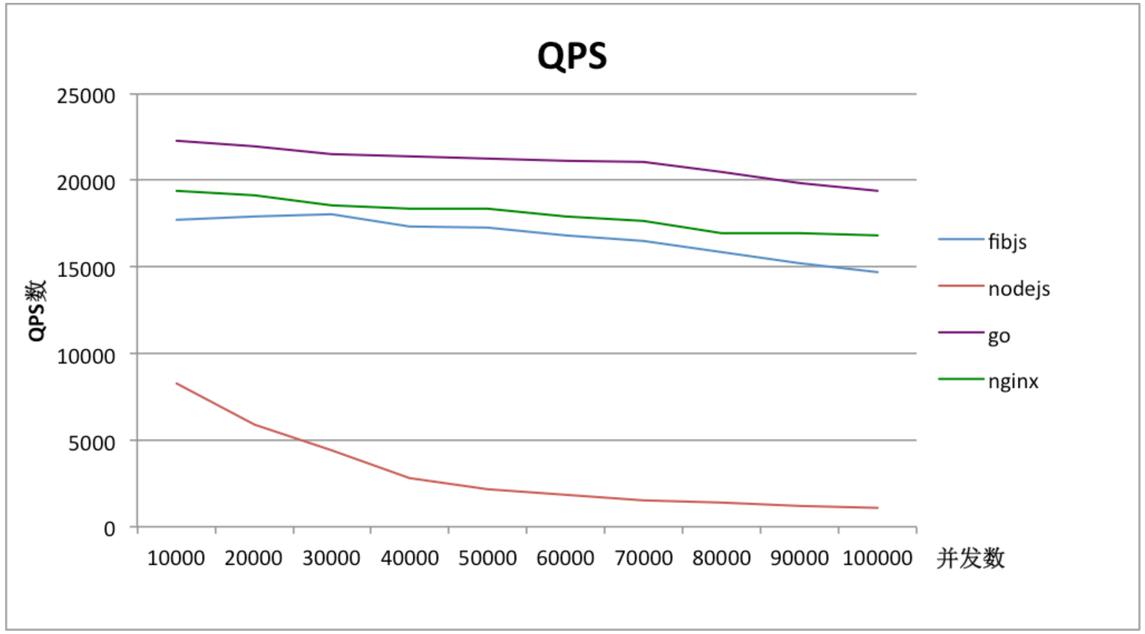
这是当时的 web 服务器基准测试



{ fib:JS }

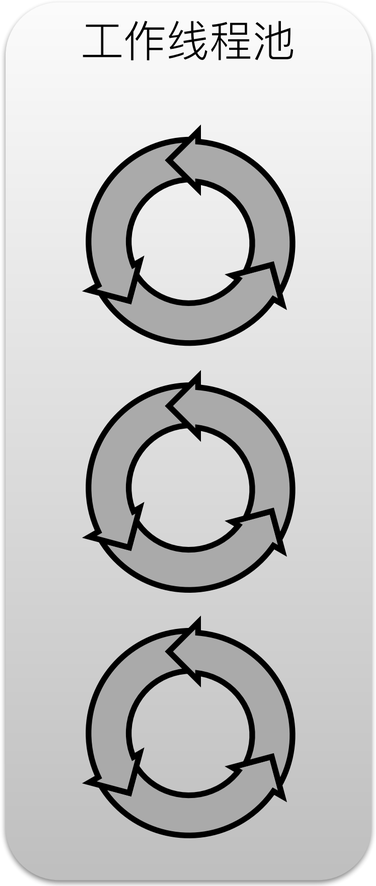
单机 10 万连接的超强压力测试





{ fib:JS }

第一次重构：工作线程 fiber 化

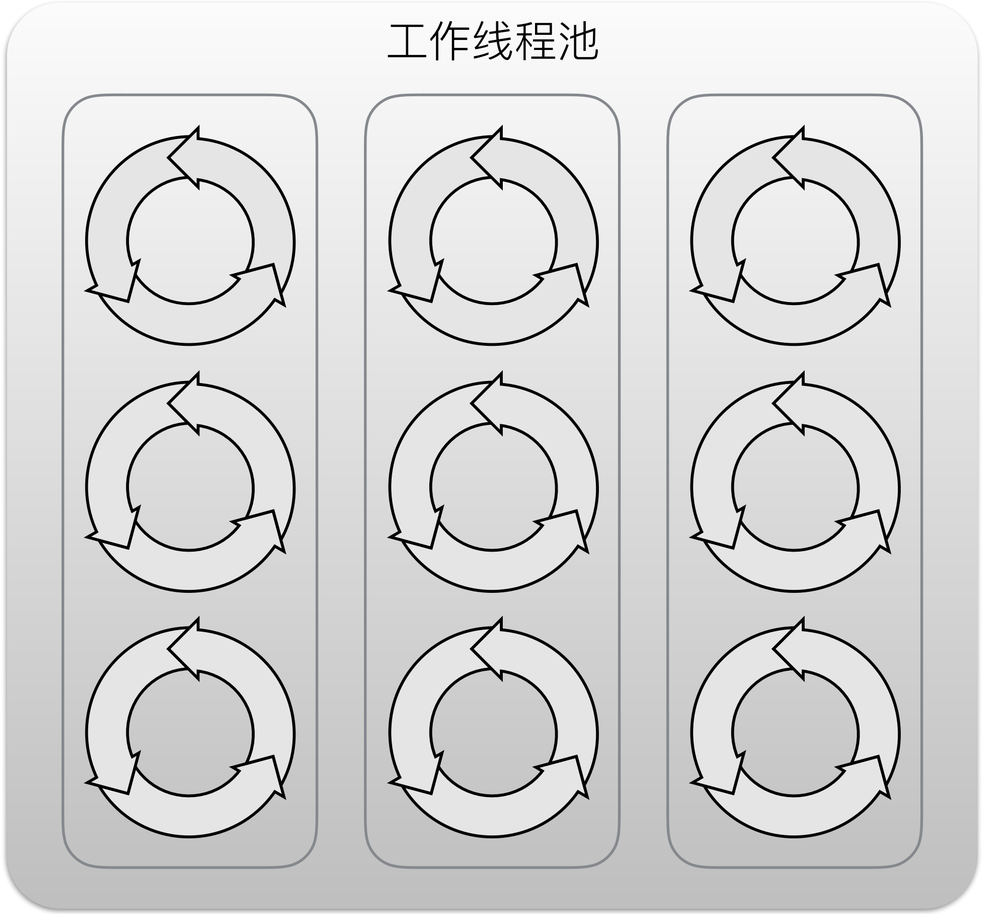


问题：

- 每一项任务都需要独占一个线程
- 任务挂起时，线程被闲置
- 任务空闲时，线程池回收成本高
- 线程数量会在任务增加时剧增
- 全异步实现逻辑复杂，复用性差

{ fib:JS }

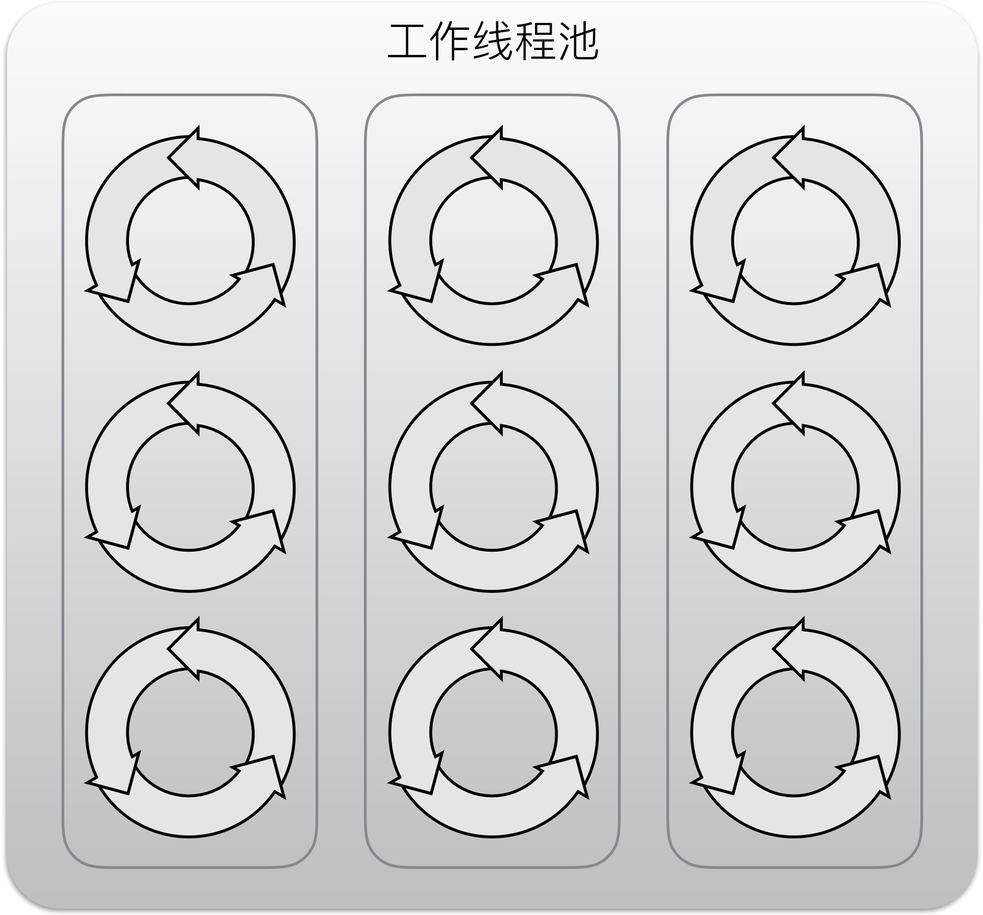
方案：fiber 线程池



- 每个工作线程为一个 fiber 容器
- 工作线程由多个 fiber 线程组成
- fiber 随时创建，用完销毁
- 可执行 js fiber 线程的部分业务

{ fib:JS }

第二次重构：fiber 多线程化

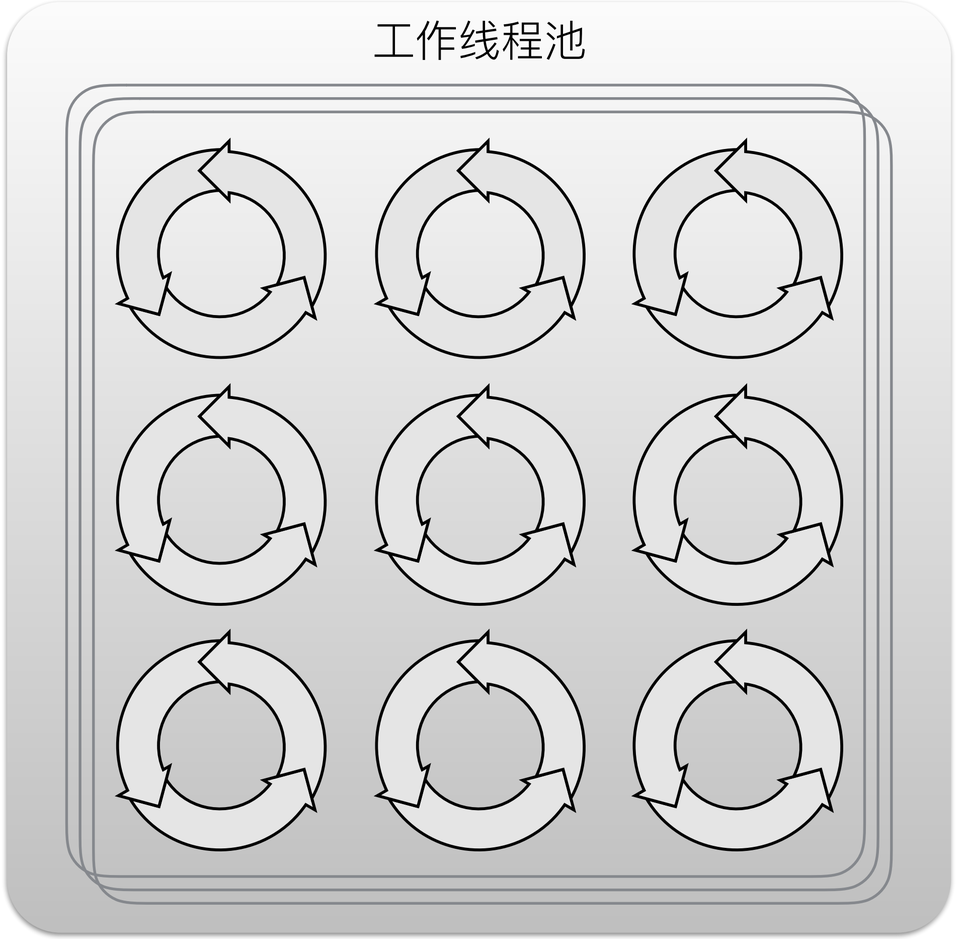


问题：

- fiber 忙，同线程其它任务挂起
- 多个 fiber 线程，管理复杂
- 线程有忙有闲，利用率低
- 线程数依然较多，调度效率低

{ fib:JS }

方案：多线程 fiber 调度



- fiber 容器有多个线程同时运行
- fiber 可由任意线程执行
- fiber 唤醒后可被切换至其它线程
- 容器内的线程按需创建
- 最大线程数不超过 cpu 核数

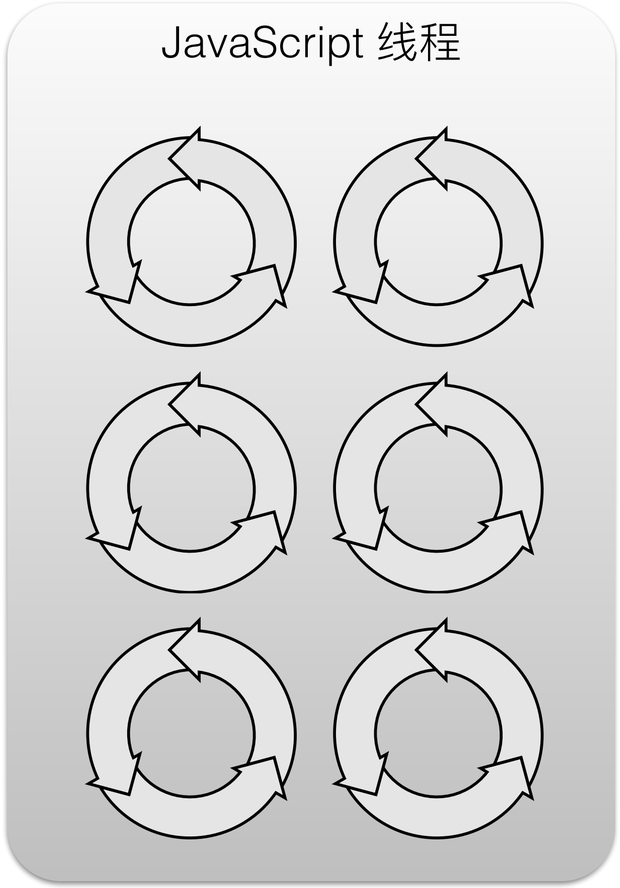
{ fib:JS }

多线程 fiber 调度就像柜台叫号

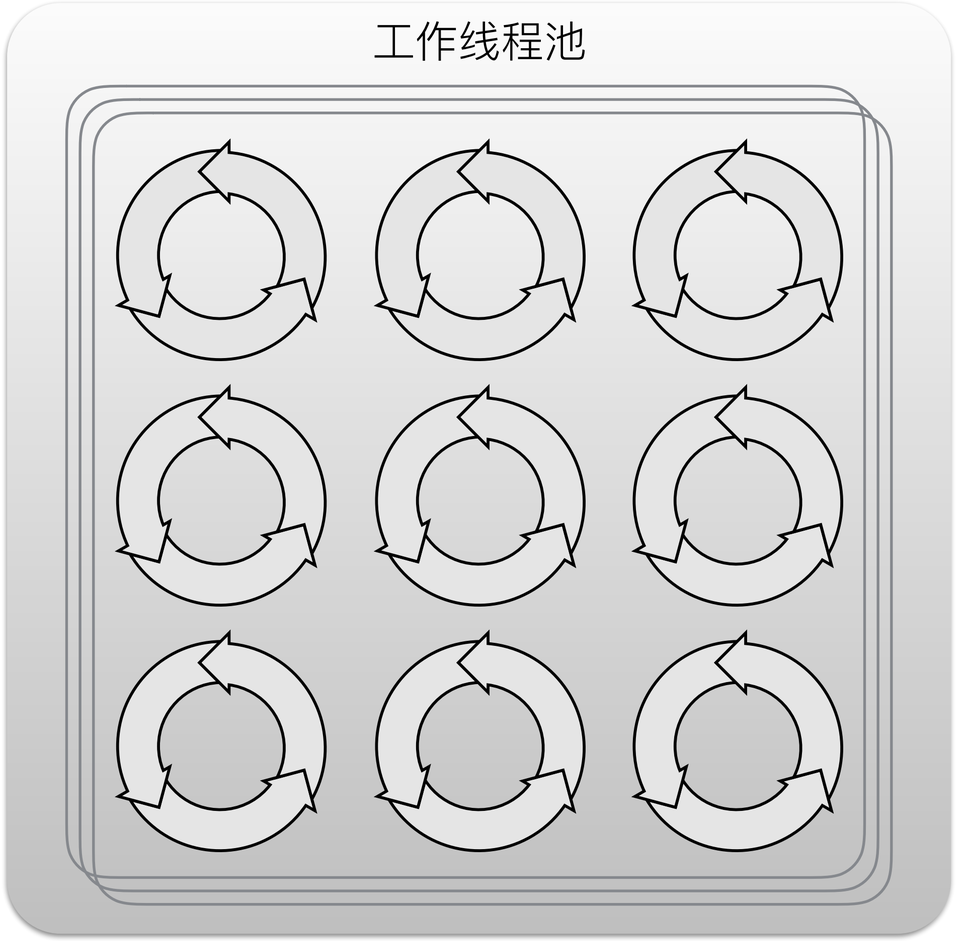
- fiber 领号进入等待队列
- 被叫号，进入线程执行
- 一个线程一次执行一个 fiber
- 下次会被另一个线程执行
- fiber 不关心由哪个线程执行

{ fib:JS }

第三次重构：js & worker 统一调度

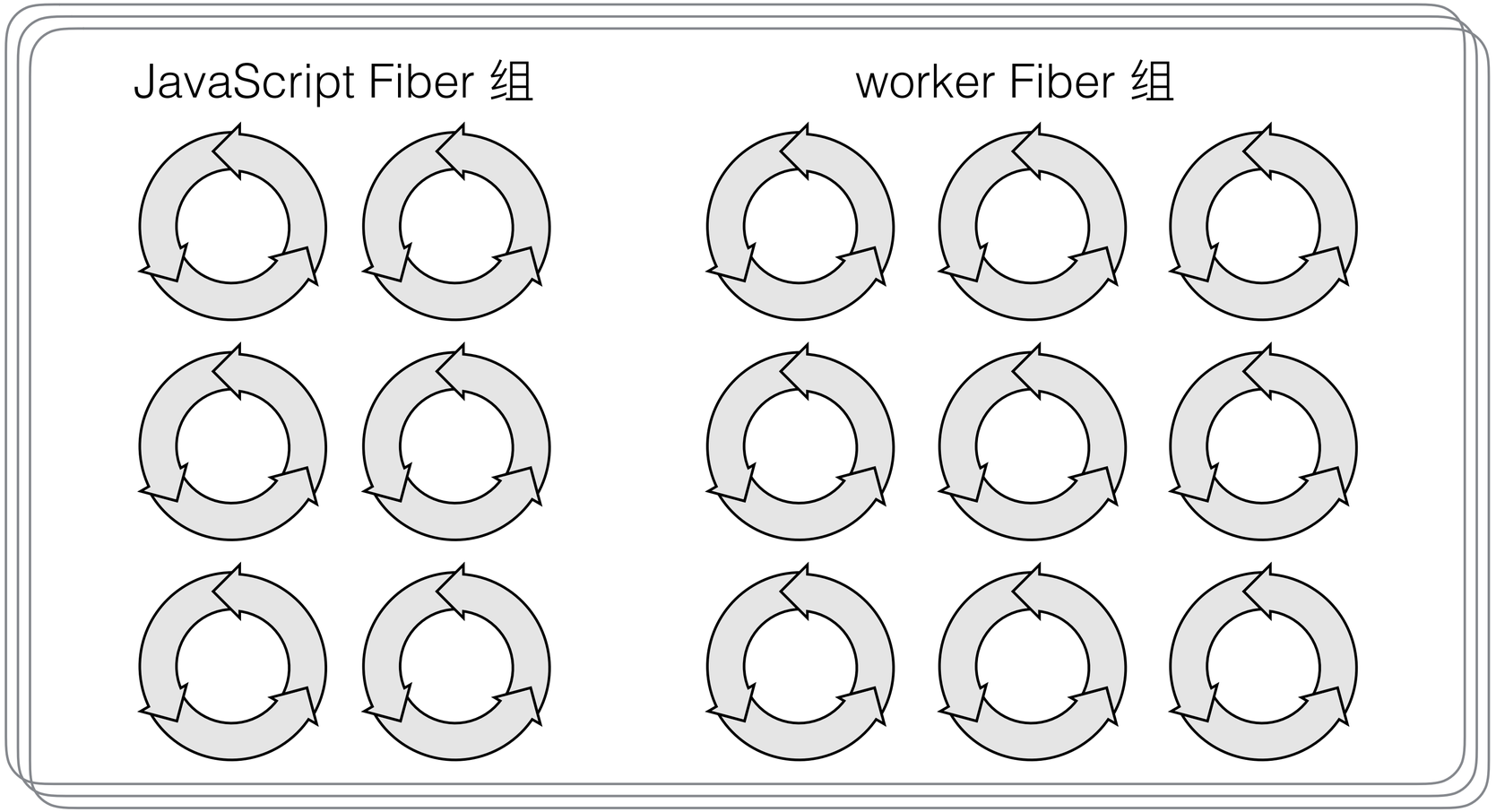


在强大的多线程 fiber 面前，单线程 JS fiber 容器显得多余



{ fib:JS }

方案：统一 fiber 容器

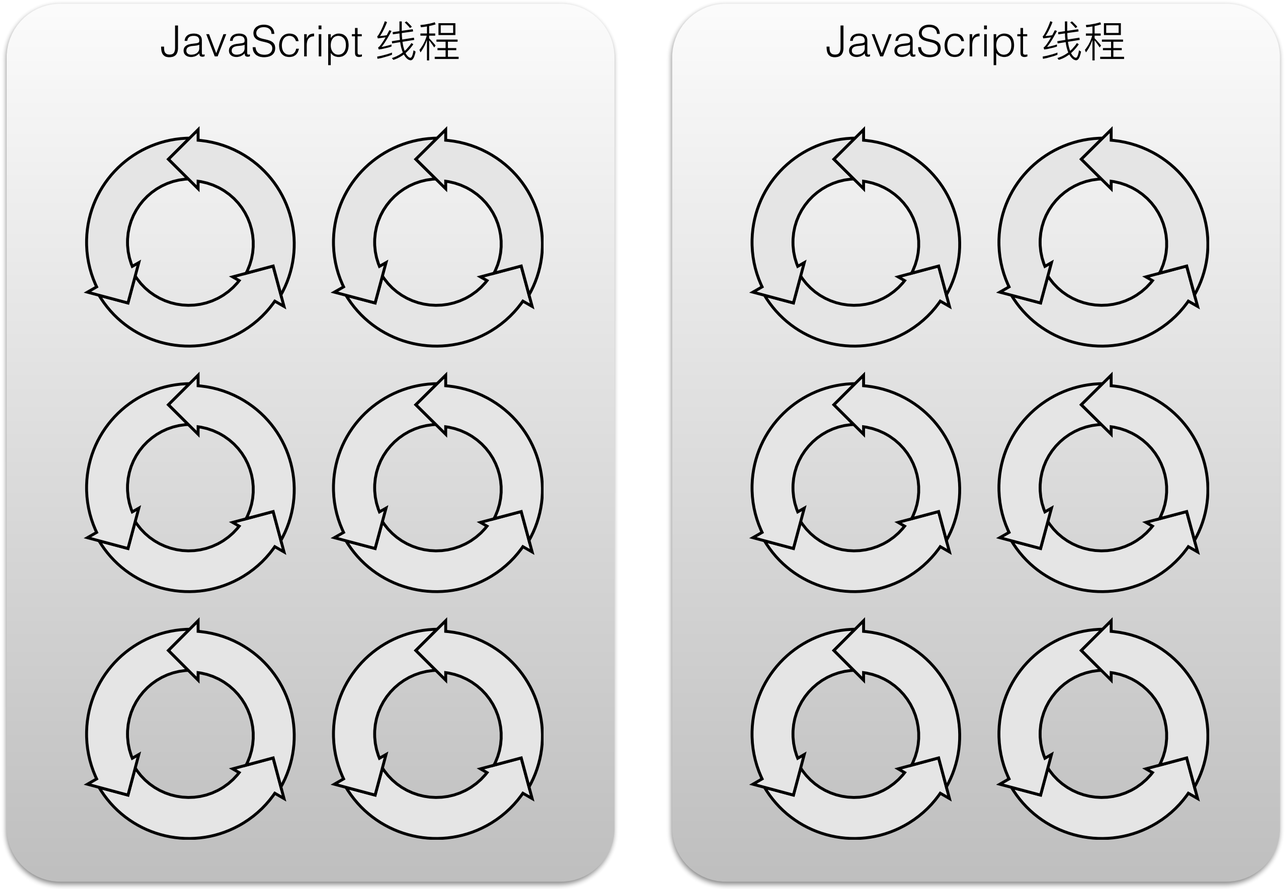


统一容器后，各fiber管理器不再需要关注线程，只需按需创建 fiber 并随时销毁

工作线程池因而退化为后台 fiber 任务

{ fib:JS }

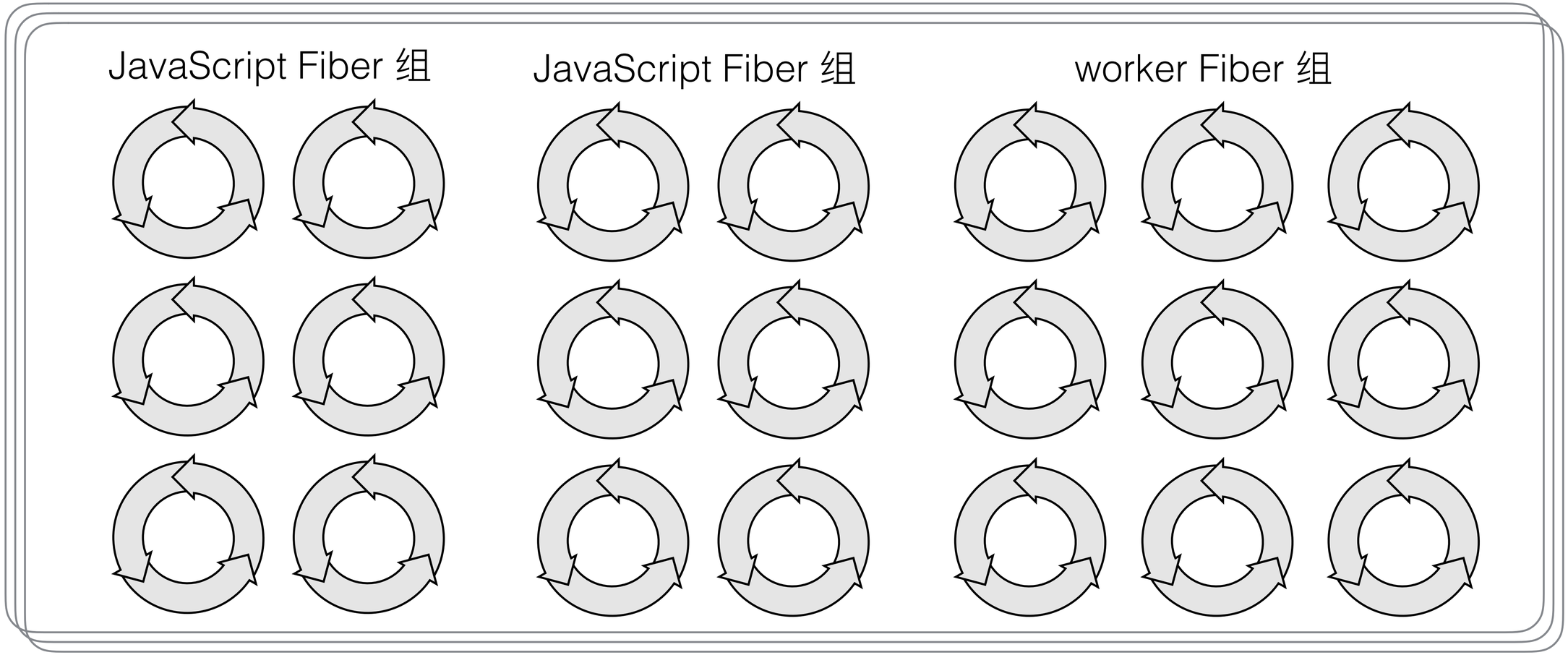
第四次重构：JavaScript vm 多实例



问题：

- 每个实例需要一个线程

方案：乱炖



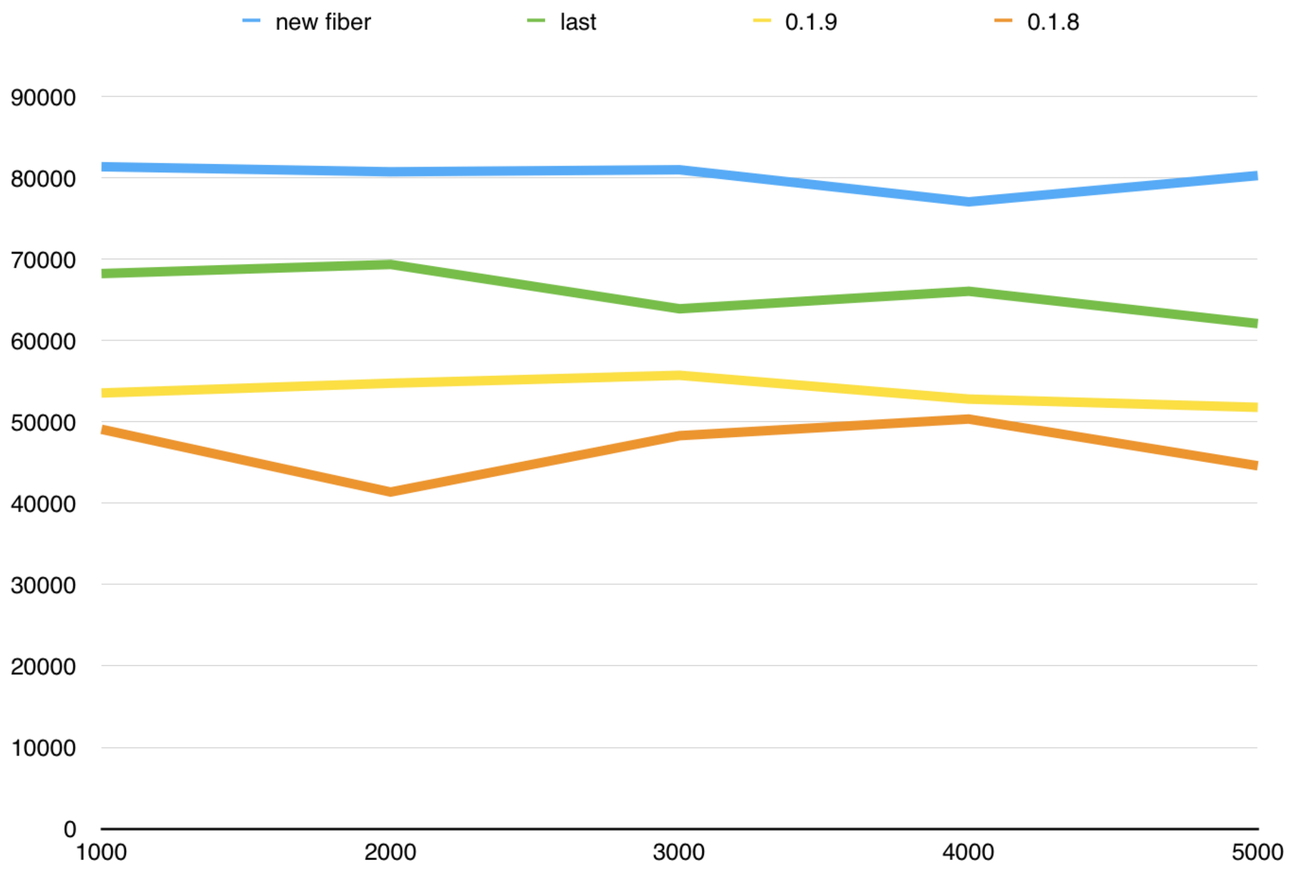
{ fib:JS }

总结

- 多线程全局单一 fiber 容器
- JavaScript 与 worker 统一调度
- 最大效率利用 JavaScript vm 运行时间
- 总线程数与 cpu 核数相同，完全避免线程调度

{ fib:JS }

这是新 fiber 引擎的成果



	new fiber	last	0.1.9	0.1.8
1000	81329.14	68201.37	53528.41	49067.74
2000	80704.12	69327.63	54728.03	41362.64
3000	80956.43	63873.76	55706.83	48284.88
4000	77015.87	66030.38	52780.58	50332.05
5000	80241.38	62057.35	51765.49	44586.94

{ fib:JS }


第三极