

解密 MoE：将 Golang 嵌入 Envoy (C++)

蚂蚁集团技术专家 / 朱德江

- 开源爱好者
- 十余年网关研发
- OpenResty 老司机 (NGINX + LuaJIT)
- MOSN 核心成员
- 玩过 DSL 编译器
- 对 LuaJIT、Go 有一些研究



公众号

大纲

- 一. 网关的发展趋势
- 二. Envoy 与 Golang 的融合
- 三. cgo 优化
- 四. MoE 架构的现状与未来

MOSN 是什么

主要使用 Go 语言开发的云原生网络代理平台



支付宝新技术最佳演练场

2019

Service Mesh 搭建的全球最大金融级云原生集群首次登上大促舞台，保障支付宝和淘宝双端联动。支付宝技术出海赋能，6个基于支付宝技术的海外钱包首次服务全球电商。

通过融合计算引擎，协同流、图、并行计算、机器学习等不同计算模式，在支付过程中提供秒级智能决策能力。

随着图计算在花呗、蚂蚁森林等场景中大规模上线，图数据库Gea-base突破万亿边，在线图分析百亿关系特征计算秒级响应。

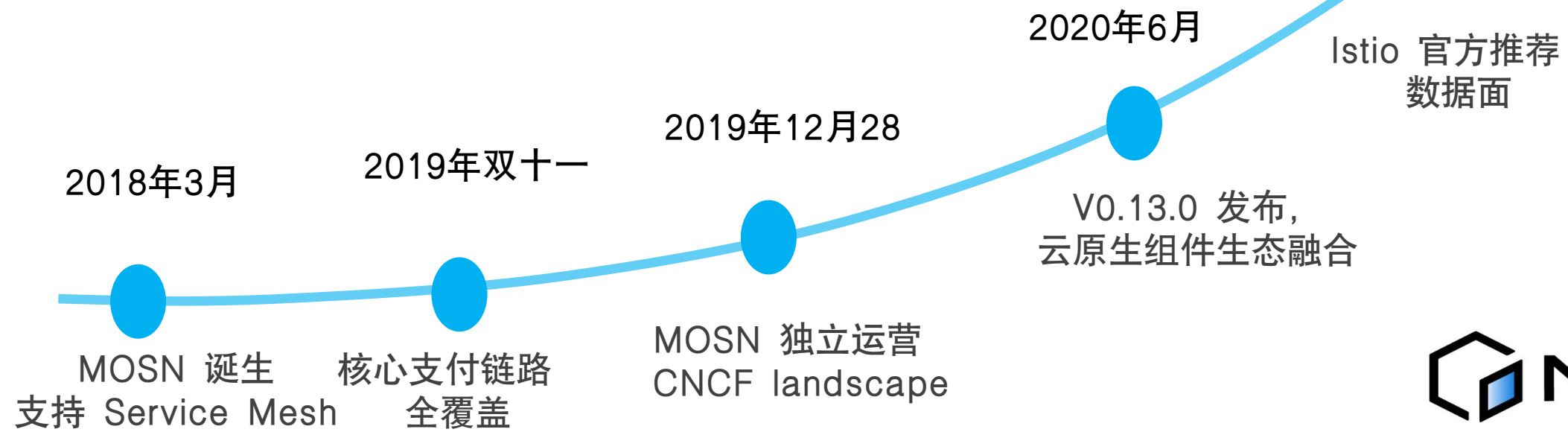
2018

区块链技术第一次全面参赛，百余个国家和地区、1.5亿跨境商品的原产地溯源不再是难题。

生物支付首次成为双11主流支付方式，每10笔支付就有6笔，标志着密码正退出历史舞台，中国的生物支付时代来临。

支持双11的核心技术100%对外开放，帮助金融机构进行数字化转型，更加专注自身业务创新，创造属于自己的“双11”。

基于新计算框架的融合引擎首次上线花呗场景，计算模式正式开始发生改变。



ISTIO / BLOG / 2020 POSTS

2020 Posts

Blog posts for 2020.

Using MOSN with Istio: an alternative data plane

July 28, 2020

An alternative sidecar proxy for Istio.



网关的发展历程



Web Server

LAMP

1995



反向代理

c10k

2004



Service Mesh

微服务 & 云原生

2016



两大变化

Service Mesh

微服务催生的东西向流量场景

服务发现，限流，熔断

可观测性，安全性

云原生

标准化

组件化

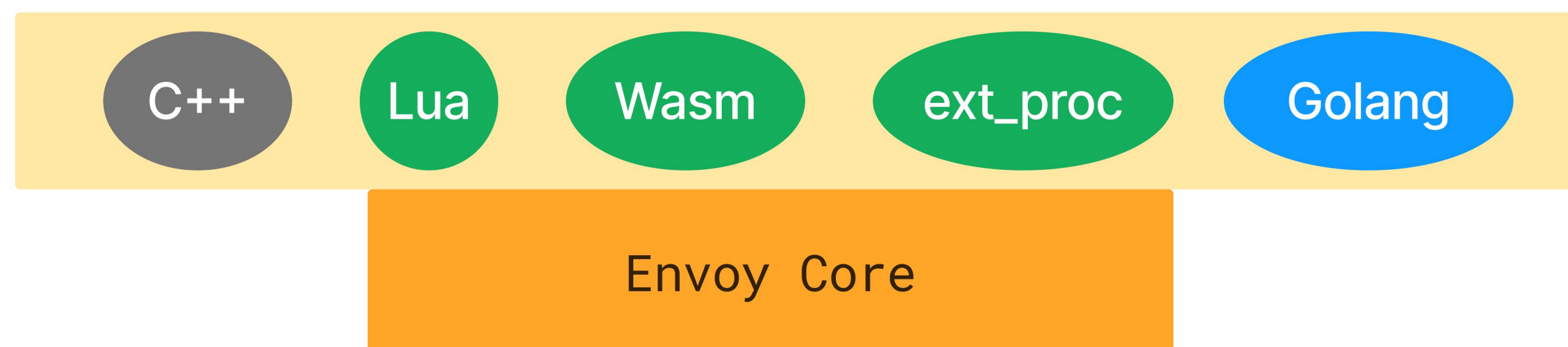
k8s Gateway API

两层架构

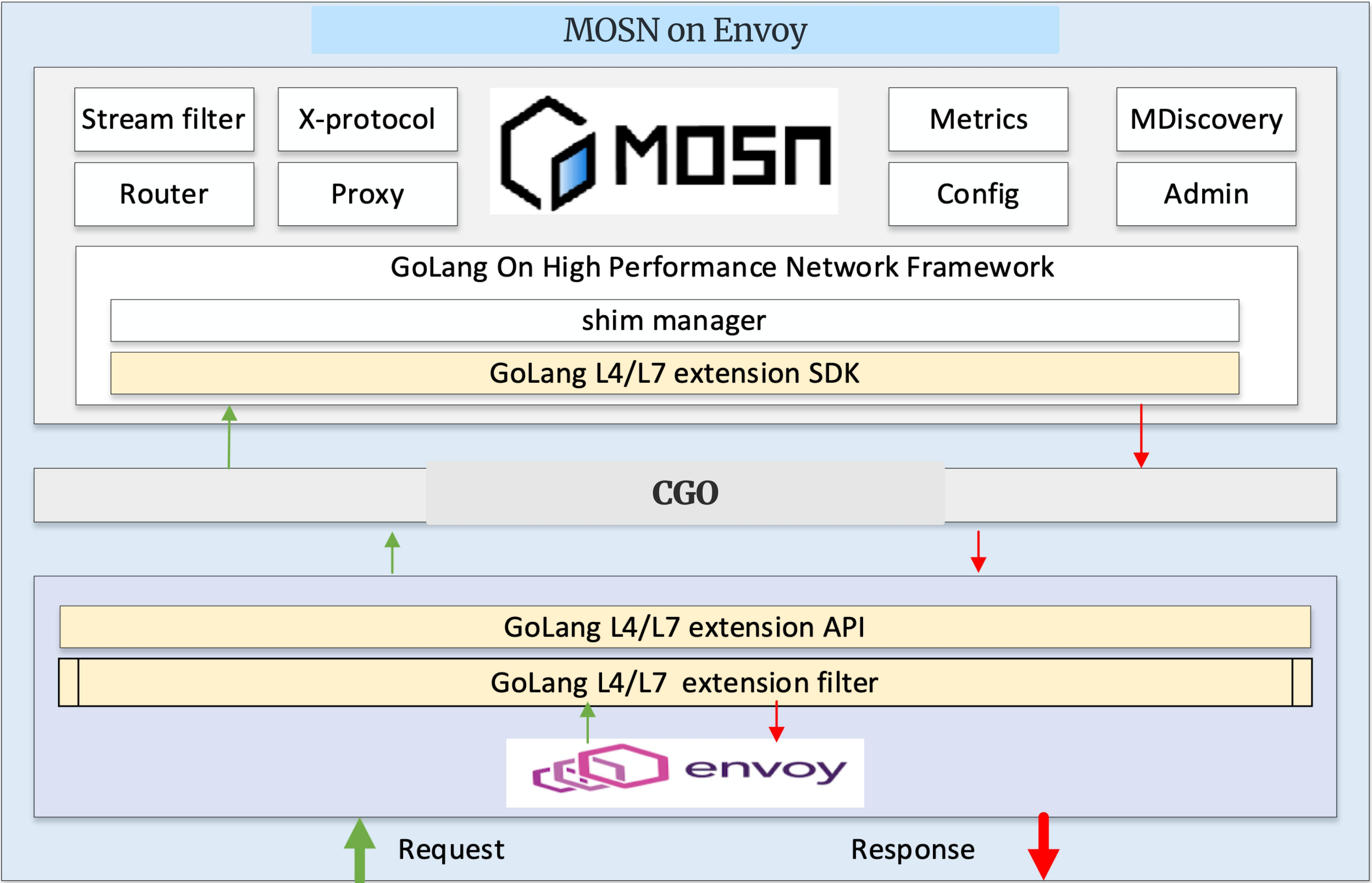
- 通用能力下沉基础设施，网关是很合适的载体
- 对于可扩展性要求越来越高

扩展系统： 满足大量的扩展需求

Core: 核心框架 + 基础能力



MoE 框架



取长补短 & 生态红利

- Envoy 视角：
标准的 L4/L7 扩展
享受 Golang 的生态红利
- MOSN 视角：
更换底层网络库
复用 Envoy 的能力

为什么是 C++ 和 Golang

Envoy 为什么选择 C++

底层语言

C++ 的生态红利

Rust?

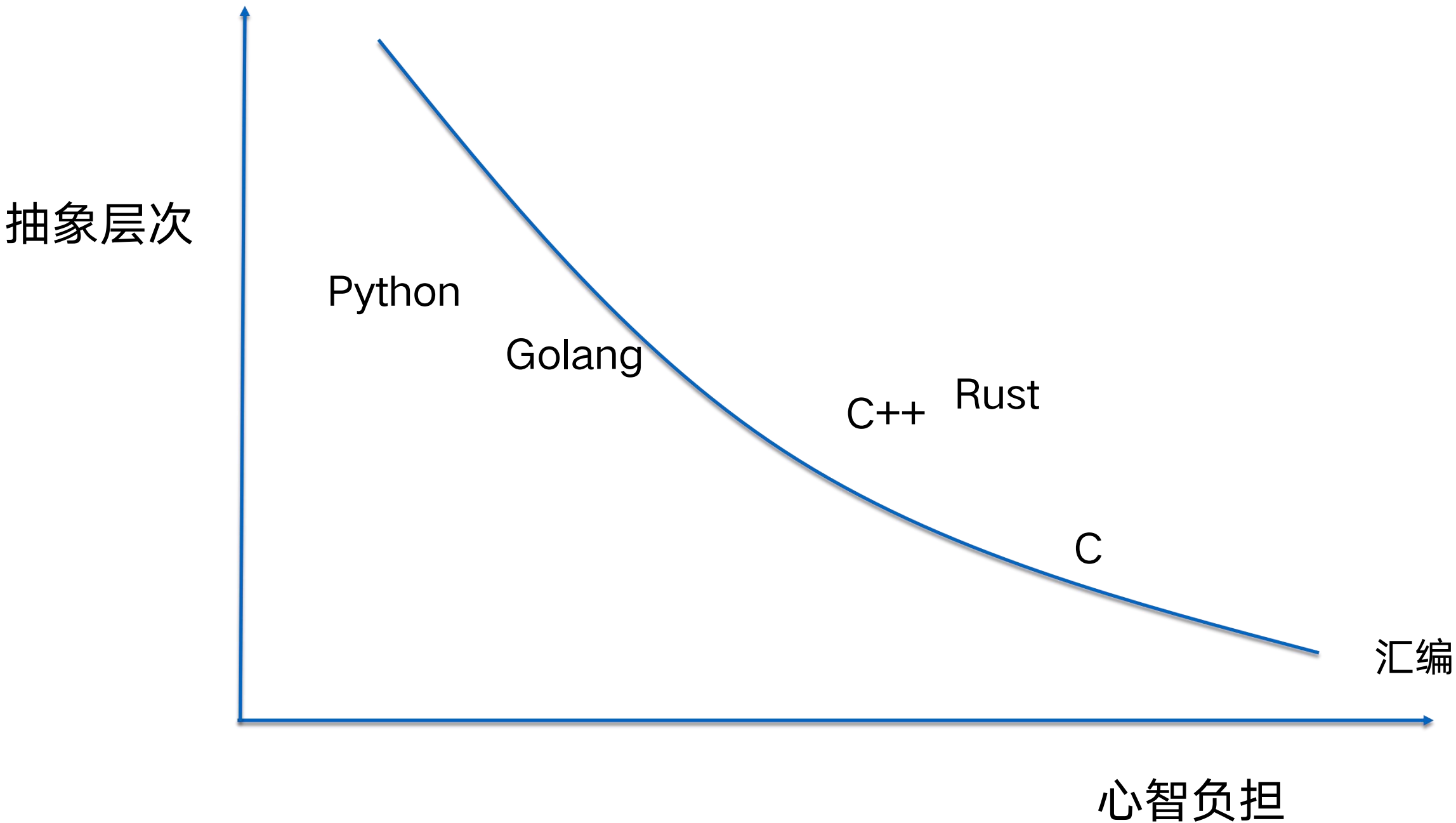
MoE 为什么选择 Golang

研发效能

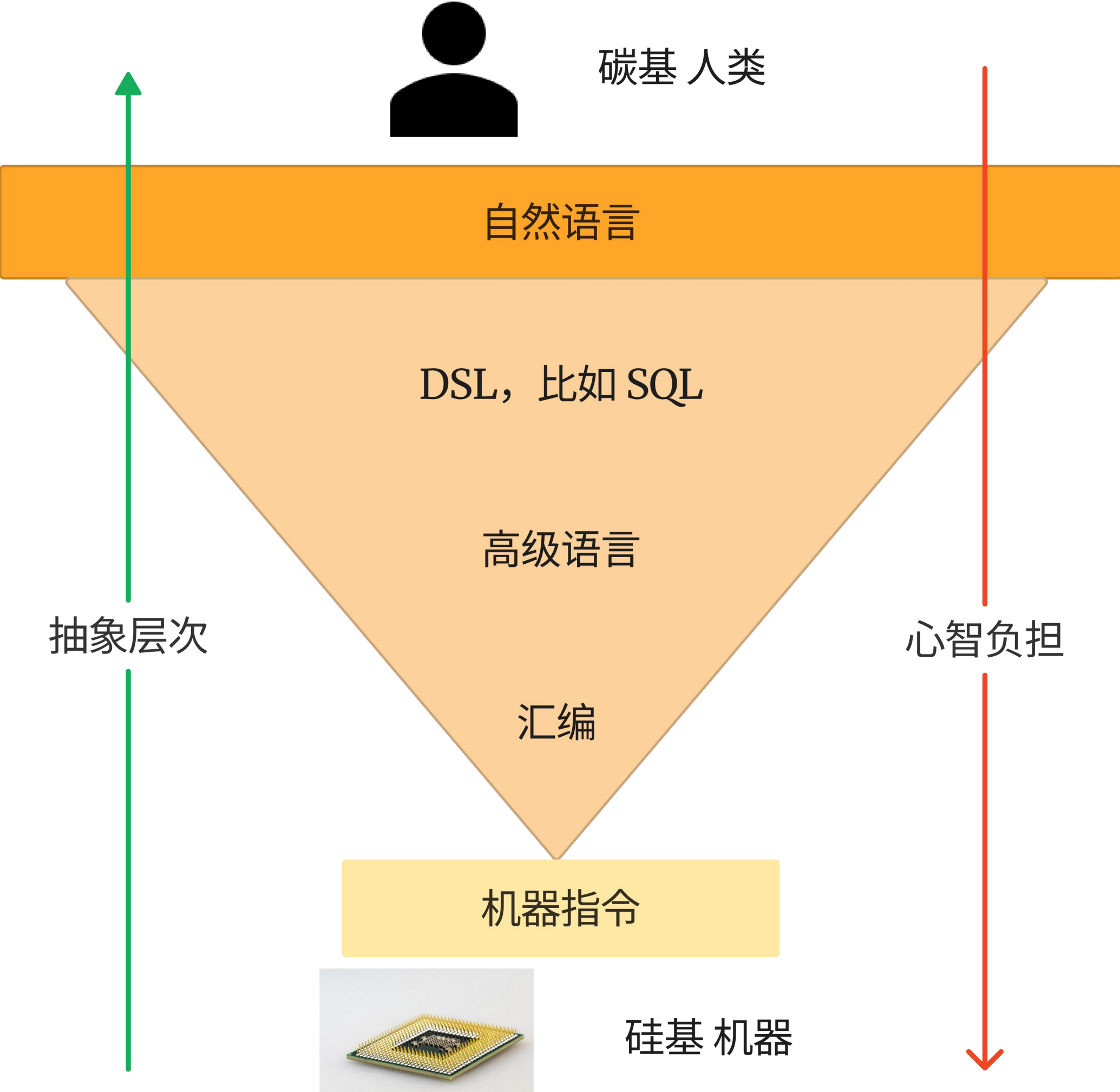
良好的生态，上手门槛低

Wasm? Lua?

妄议语言生态位



- 抽象能力
- 开发者生态
- 生态位，适合自己的领域



二. Envoy 与 Golang 的融合

挑战：将 Golang 当做嵌入式语言

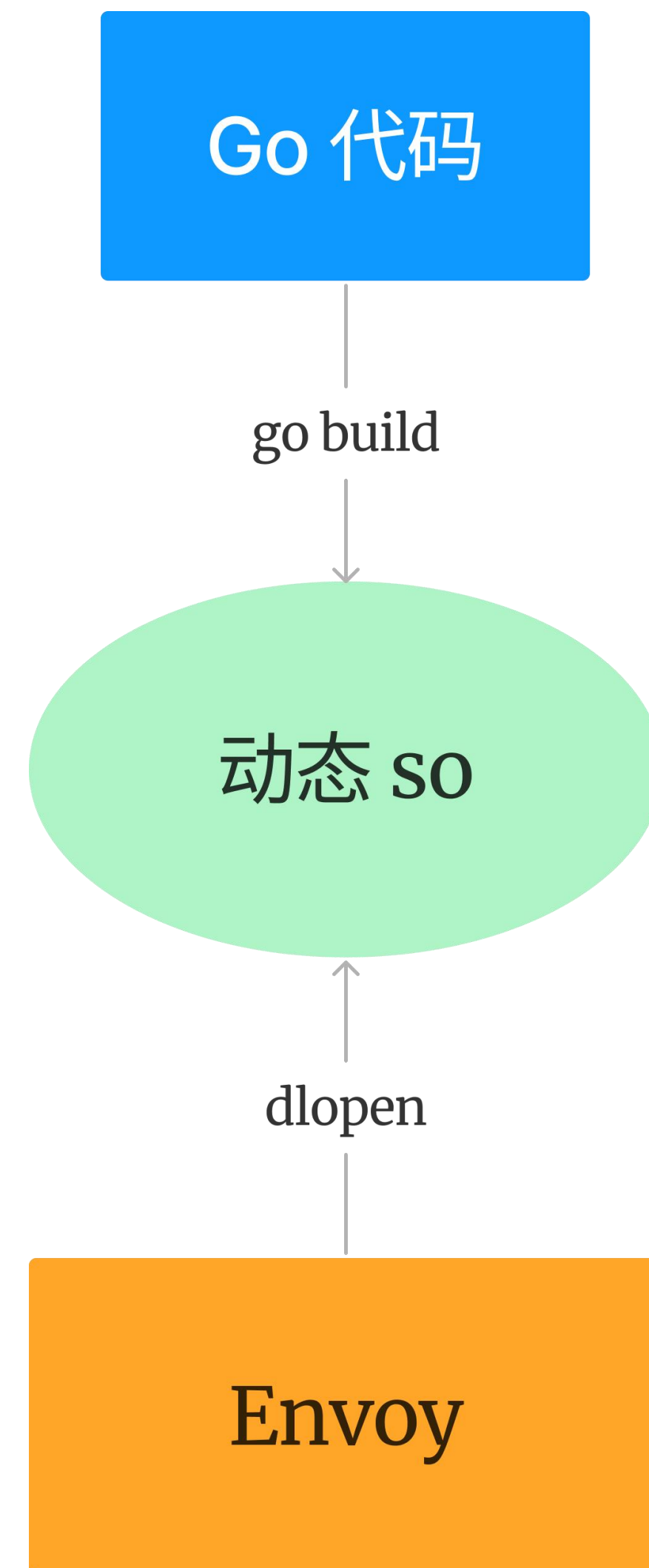
选择：保持用户低心智负担

- 内存安全
- 并发安全
- 沙箱安全

举个例子 - basic auth

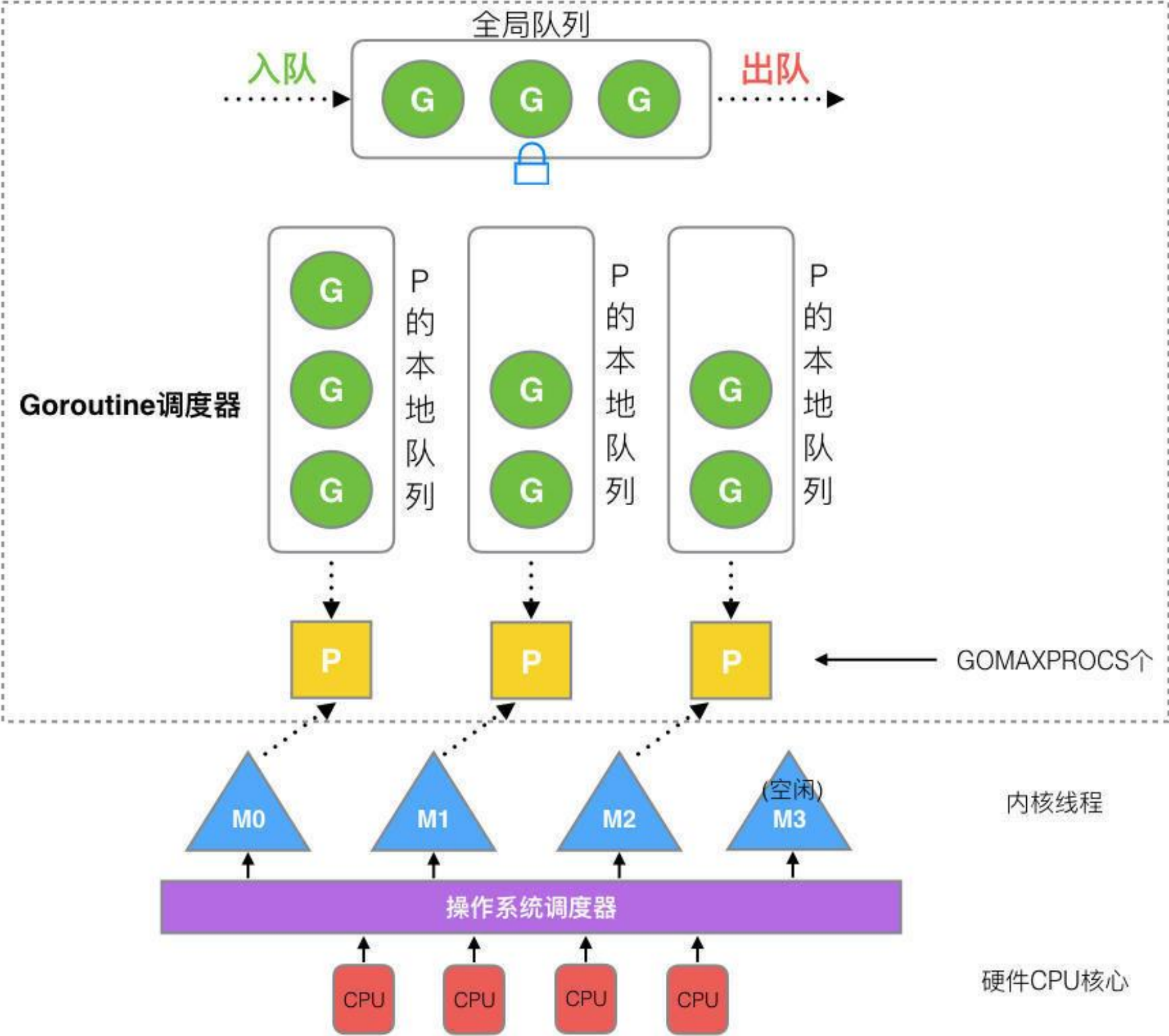
```
func (f *filter) verify(header api.RequestHeaderMap) (bool, string) {
    auth, ok := header.Get("authorization")
    if !ok {
        return false, "no Authorization"
    }
    username, password, ok := parseBasicAuth(auth)
    if !ok {
        return false, "invalid Authorization format"
    }
    if f.config.username == username && f.config.password == password {
        return true, ""
    }
    return false, "invalid username or password"
}

func (f *filter) DecodeHeaders(header api.RequestHeaderMap, endStream bool) api.StatusType {
    if ok, msg := f.verify(header); !ok {
        // TODO: set the WWW-Authenticate response header
        f.callbacks.SendLocalReply(401, msg, map[string]string{}, 0, "bad-request")
        return api.LocalReply
    }
    return api.Continue
}
```



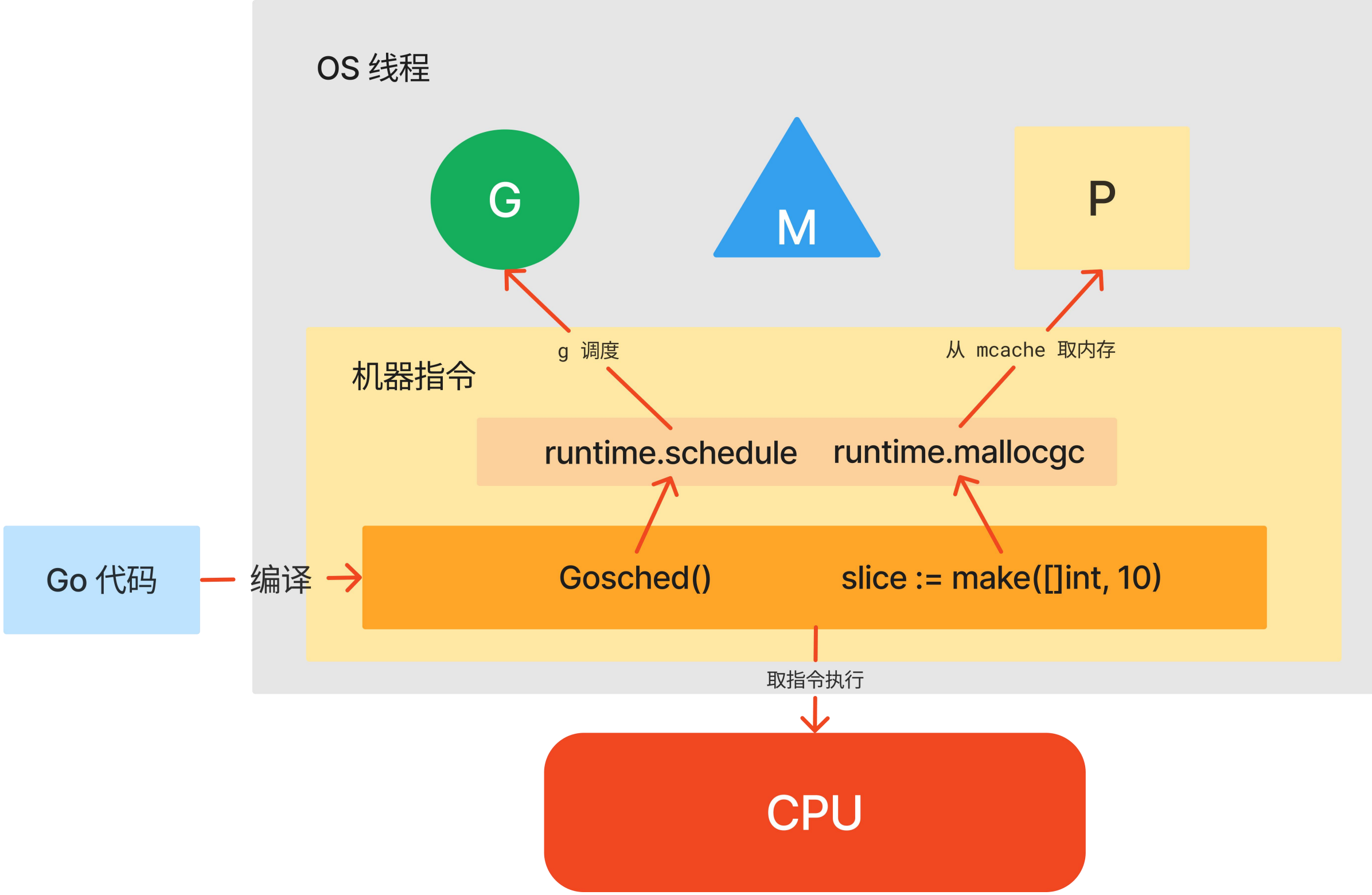
Golang 的 GMP 模型

全局视角



来源: <https://learnku.com/articles/41728>

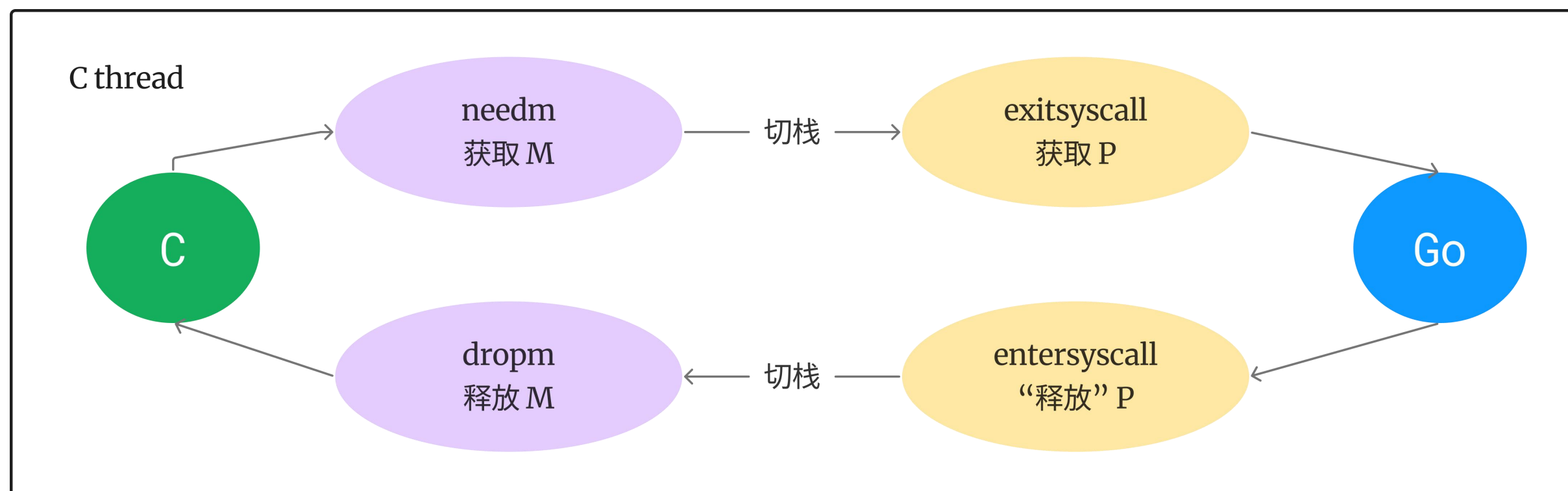
线程视角



OS 线程 + GMP 环境, 执行 Go 函数

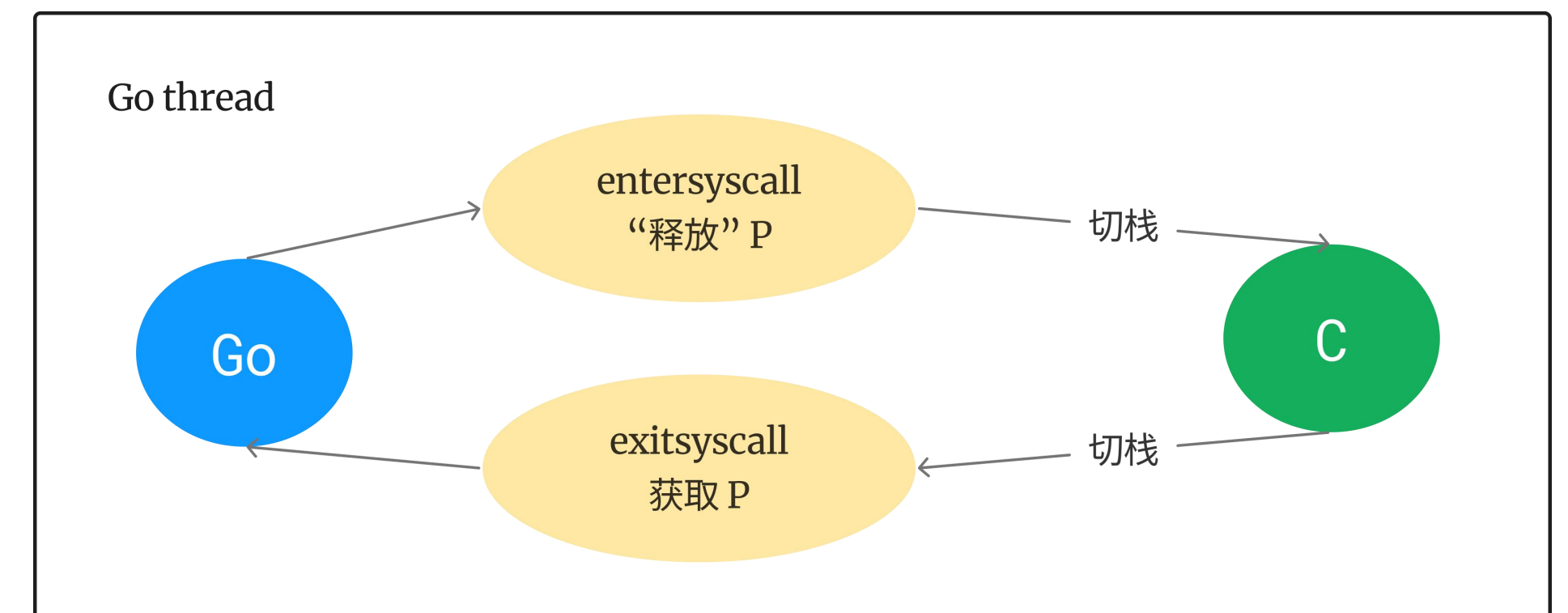
cgo 的两个执行方向

C 调 Go



- Extra M (虚拟的 M)
- 每个 Extra M, 预先“绑定”一个 Goroutine
- 从 c 进入 Go, 获取 P, 从 Go 返回 c, 释放 P

Go 调 C



- 不需要 GMP
- “释放” P 是为了避免浪费

内存安全

Envoy

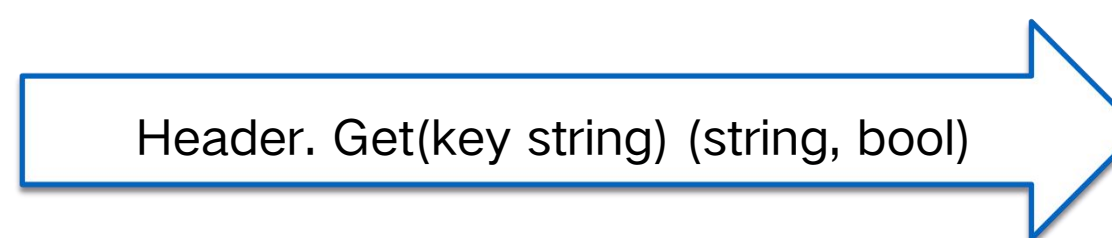
基于请求生命周期的内存管理



Golang

自带 GC

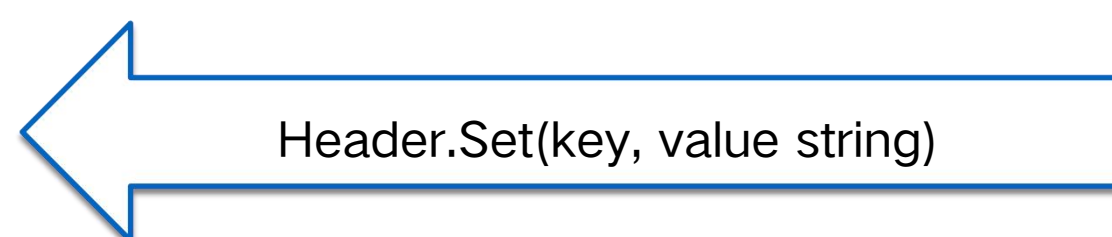
② C++ 复制内存到 Go



① Go 侧预先分配内存空间

① C++ 返回 Go 之前，引用是安全

② 返回之后继续使用，需要复制



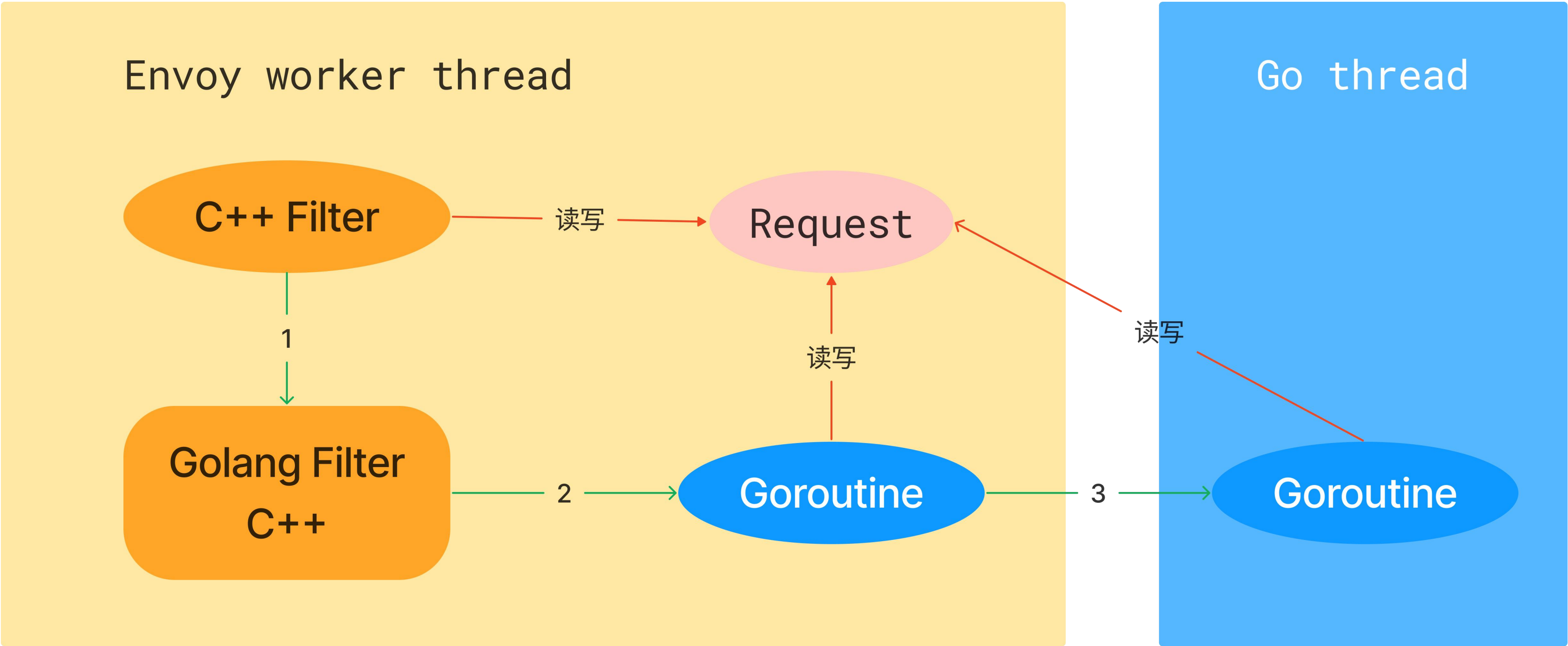
栈引用是可靠的

为什么会有并发

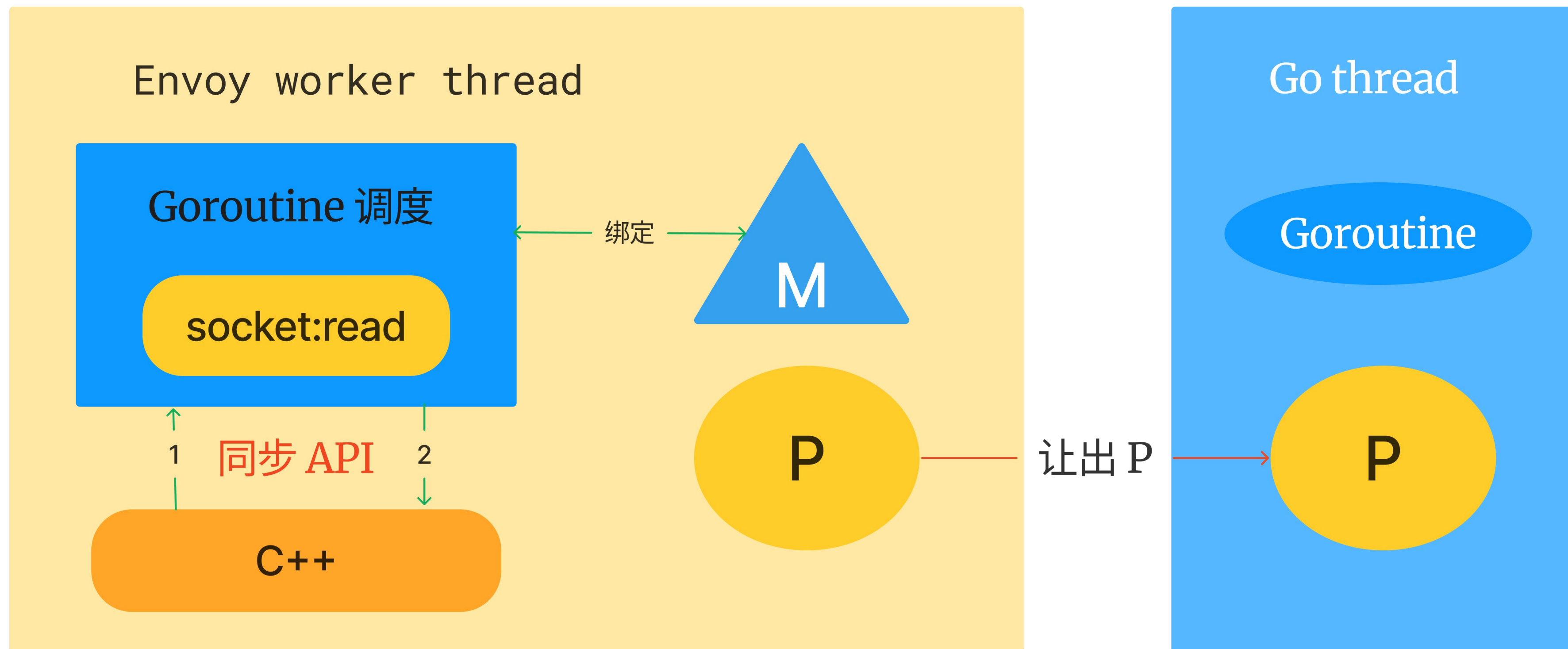
两类线程



Goroutine 两种运行形态



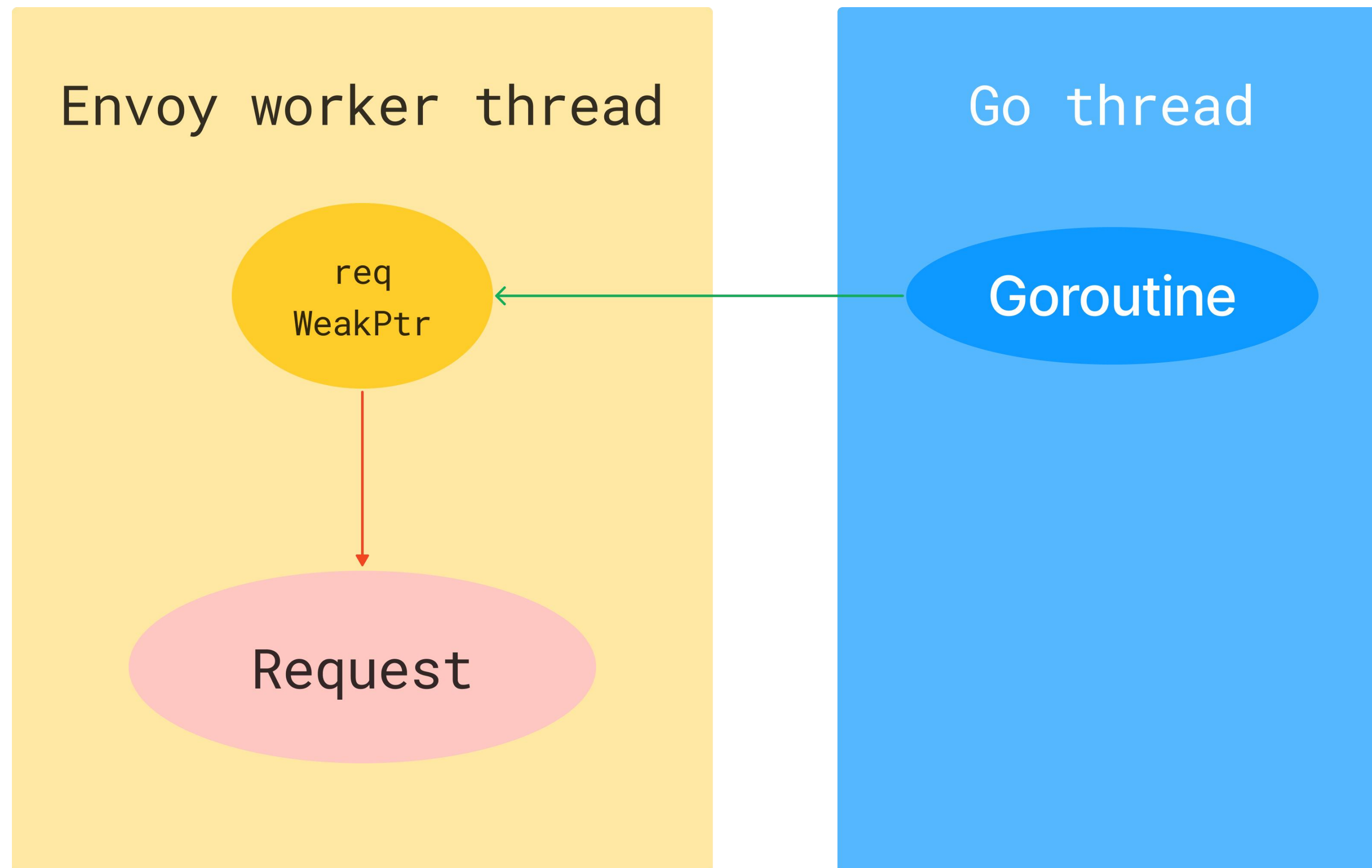
为什么需要 Go thread



- 同步 API, 导致 G 和 extra M 绑定
- 让出 P, Envoy 线程挂起

- 全功能的 Go 语言特性
- GMP 调度不关心线程阻塞

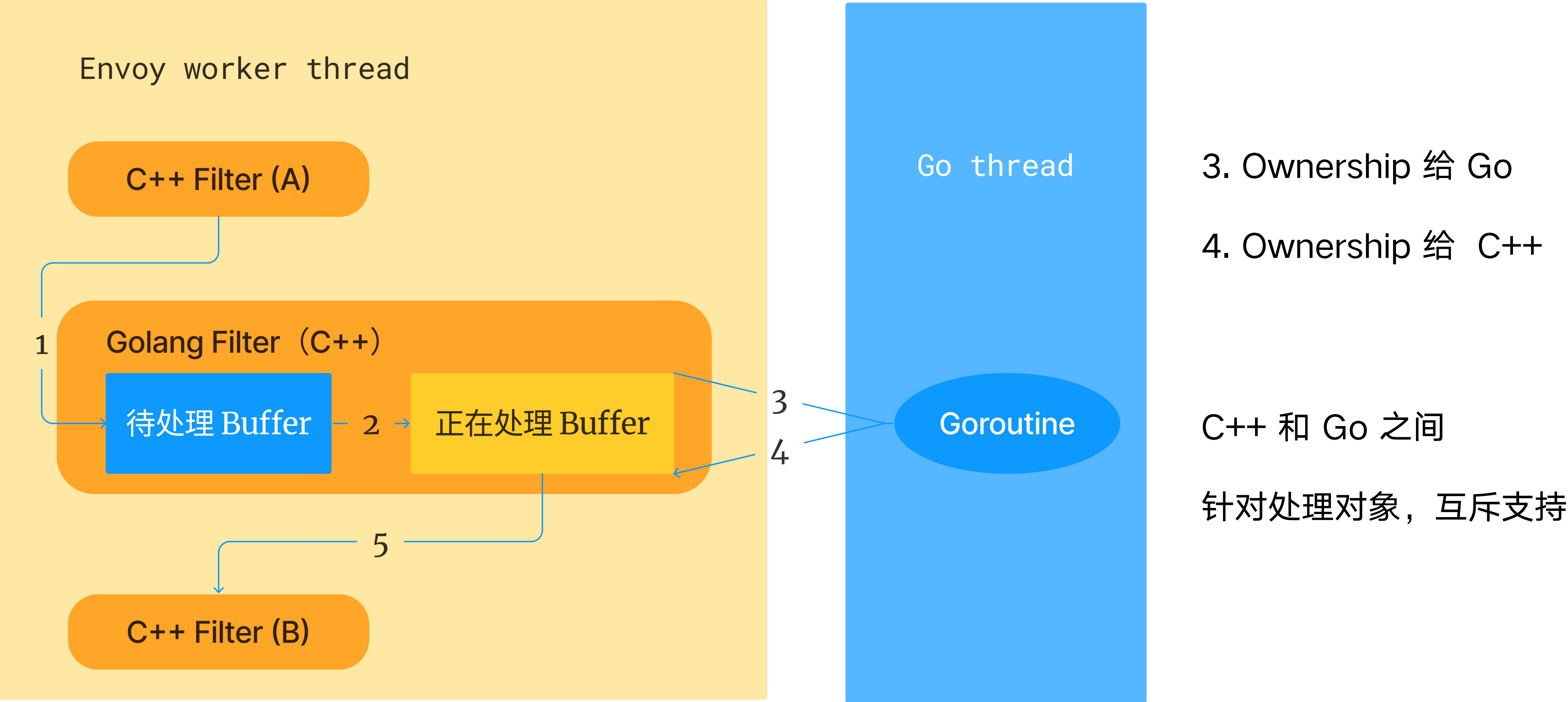
并发安全的实现 - 请求生命周期



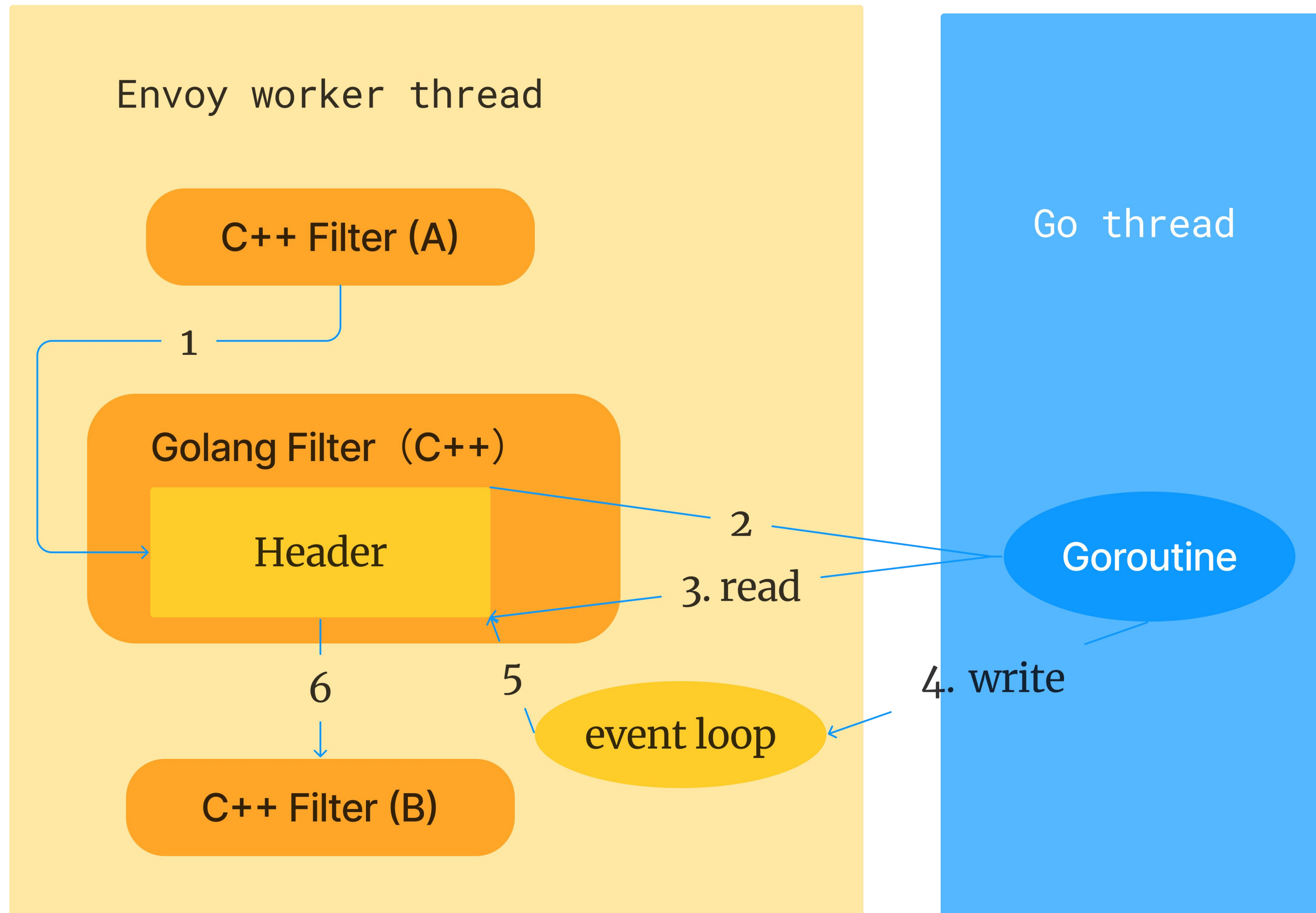
- Go 持有 C++ 内存对象
- weakPtr

```
auto weak_filter = req->weakFilter();  
if (auto filter = weak_filter.lock()) {  
    return f(filter);  
}
```

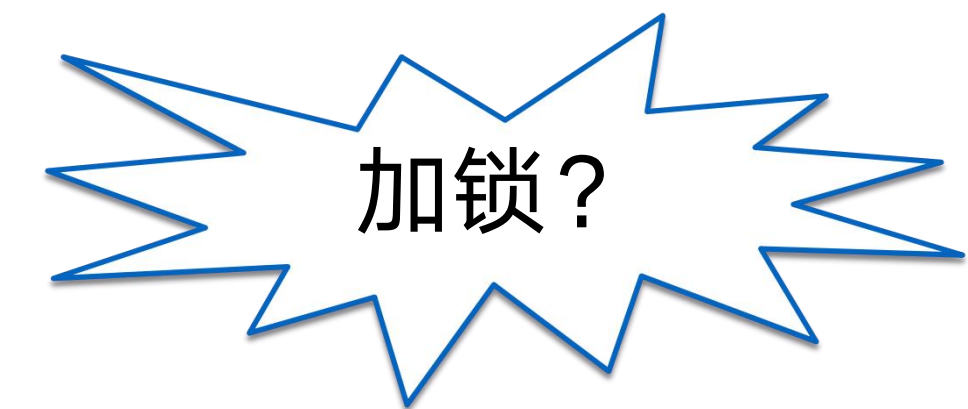
并发安全的实现 - 所有权设计



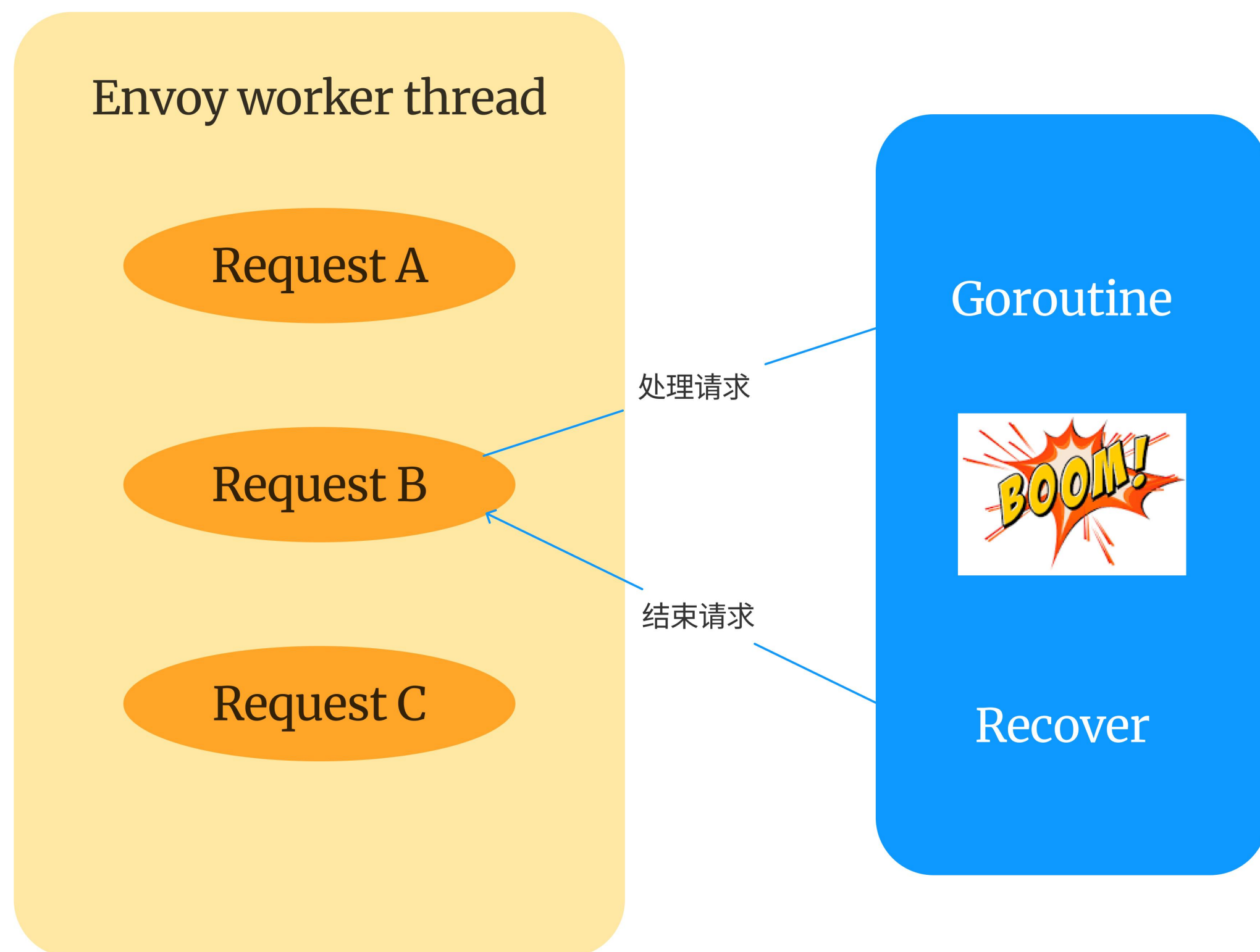
并发安全的实现 - 异步事件调度



3. 读安全
4. 写异步事件
5. Envoy 线程中完成写



沙箱安全



目的：保护 Envoy 宿主，减少爆炸半径

效果：Go 代码异常，只是结束当前请求

方法：通过 Go runtime 提供 recover 来恢复异常

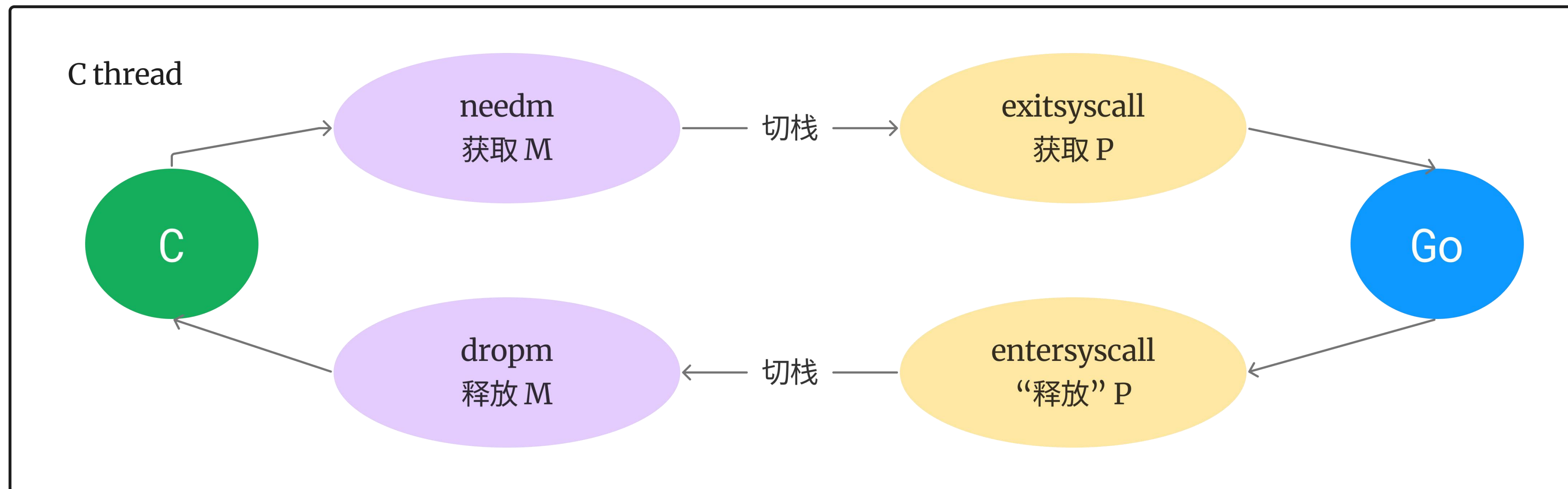
局限：有些异常 recover 恢复不了

未来：通过重新加载来重建 go runtime

三. cgo 优化

- CPU 耗时优化
- Goroutine 调度优化
- GC 优化

C 调 Go 的性能瓶颈



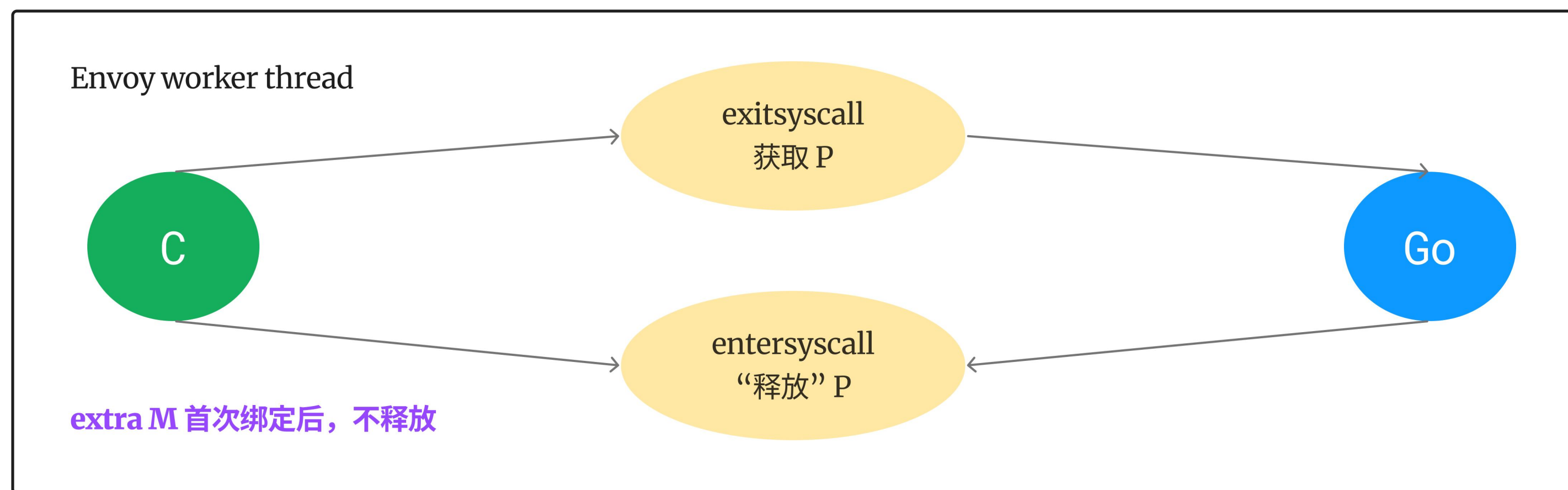
- needm: 获取 extra M, 确保 go 需要的信号没有被屏蔽
- dropm: 释放 extra M, 恢复信号

8 个系统调用

CPU 耗时优化

快了 10 倍

优化前 ~1600ns, 优化后 ~140ns (每次 c=>go)



MoE 整体提升 8%

获取到 extra M 之后不释放, c 线程退出的时候, 再释放 M

<https://go-review.googlesource.com/c/go/+/-/392854>



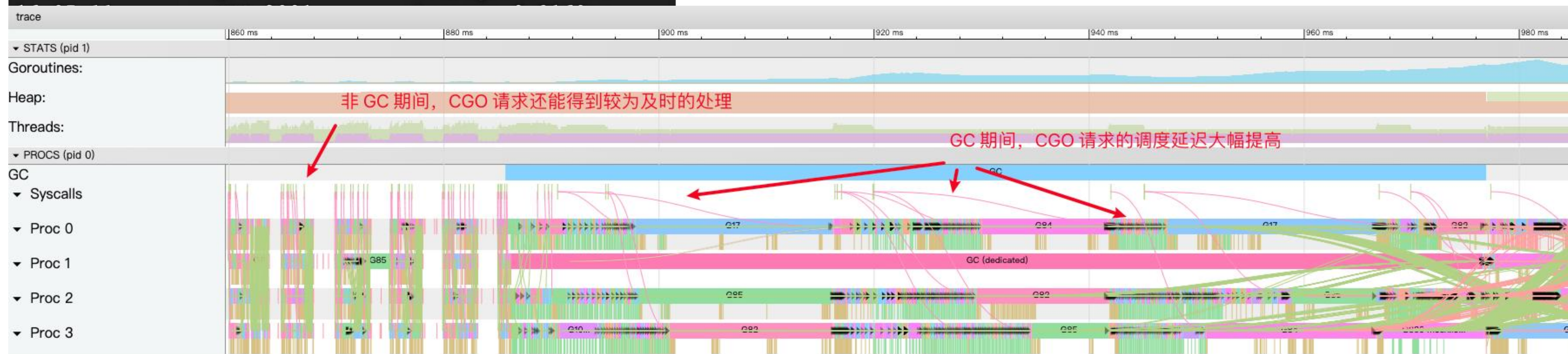
Cherry Mui

Thanks. Let's try again.

P99 太高

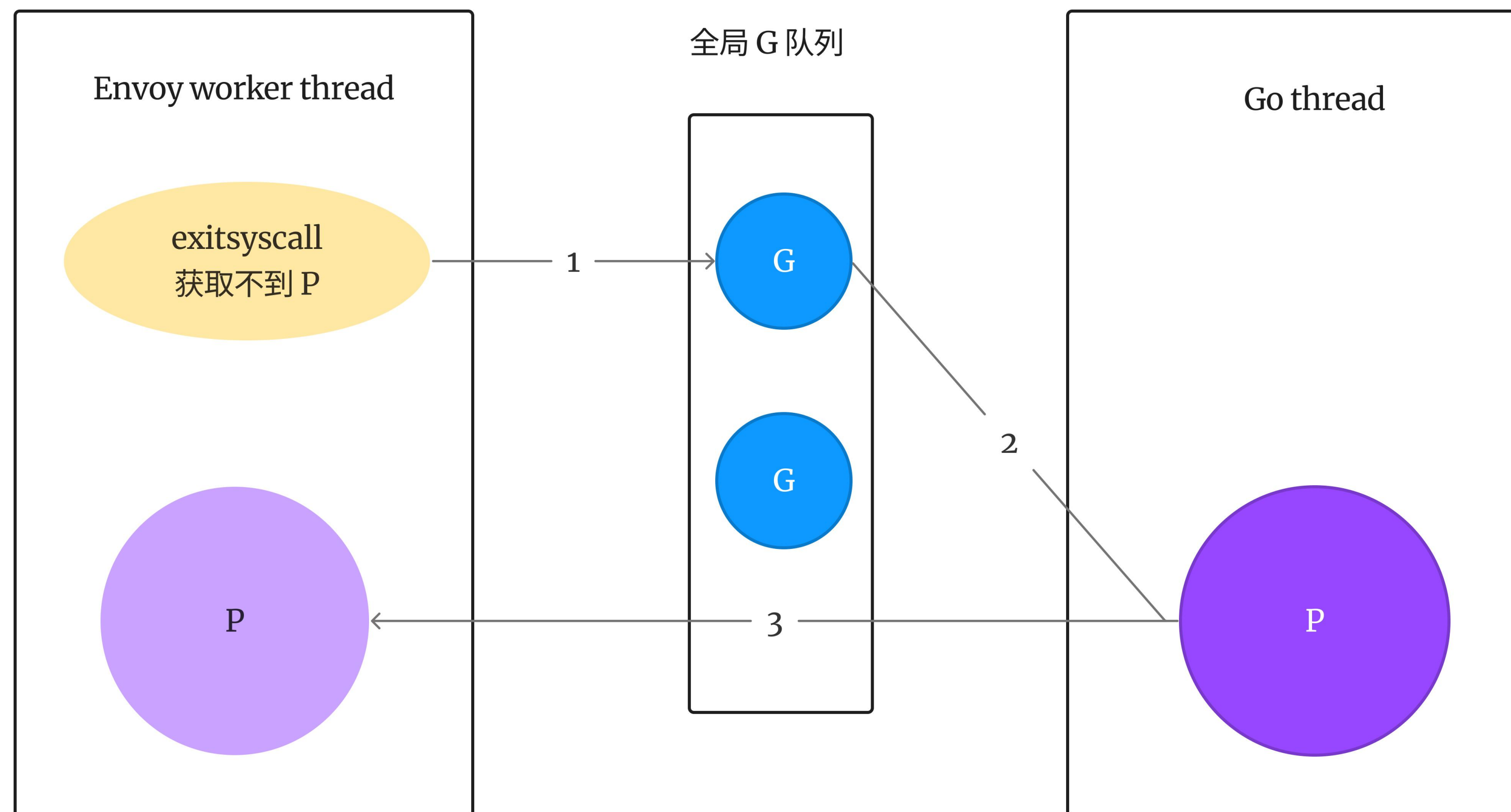
```
-----  
16:05:00      QPS:1997      Average:0.8358ms  
16:05:01      QPS:1981      Average:19.4982ms  
16:05:02      QPS:2007      Average:0.9088ms  
16:05:03      QPS:1998      Average:0.9444ms  
16:05:04      QPS:1994      Average:1.9975ms  
16:05:05      QPS:2001      Average:0.7951ms  
16:05:06      QPS:1998      Average:0.8153ms  
16:05:07      QPS:1992      Average:1.7314ms  
16:05:08      QPS:2000      Average:0.8535ms  
16:05:09      QPS:1996      Average:0.8838ms  
16:05:10      QPS:2001      Average:1.3778ms
```

- 发现 rt 周期性飙升
- 跟 GC 同步出现



cgo trace bugfix: <https://go-review.goglesource.com/c/go/+/429858>

获取不到 P

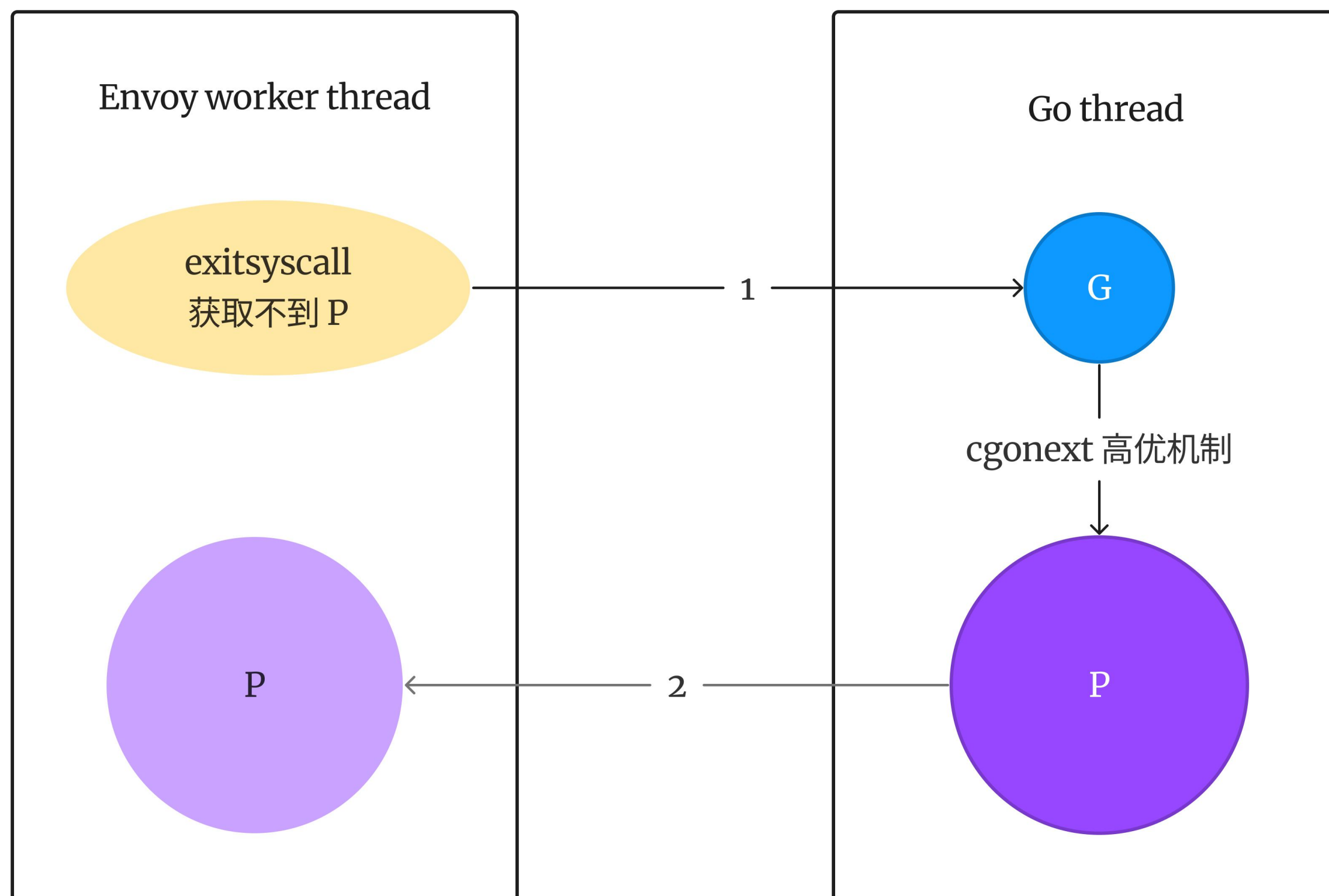


关键点：G 和 M 是绑定关系

1. G 放入全局队列，c 线程挂起
2. 从全局队列获取 G，并执行
3. 让出 P，唤醒 c 线程，Go 线程挂起

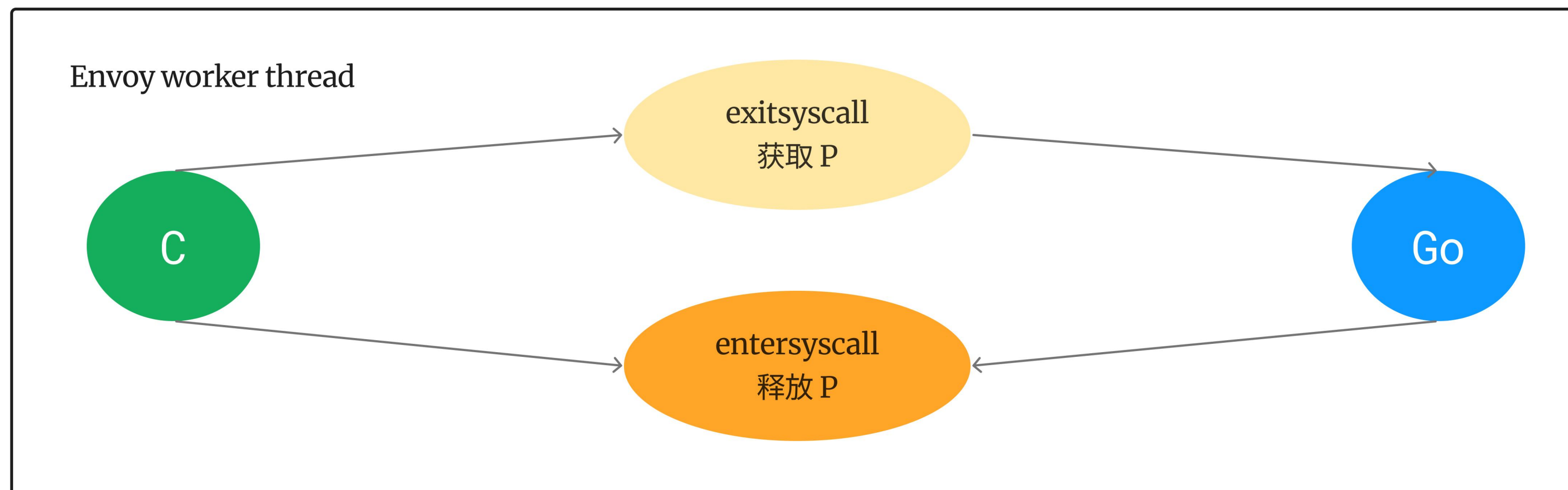
- extra M，资源无限
- P 资源有限

提高 G 的优先级



1. 新增 cgonext, 类似于 runnext
2. 效果比较理想

立即释放 P



P 会保留在 M.oldp，并标注为 `_Psyscall` 状态

- ① 下一次从 C 进入 Go 的时候，优先复用这个 P
- ② 由 sysmon 来 retake，强制释放

Go => C 场景下的优化策略

在 C => Go 则不太成立

提高 P 的利用率

<https://go-review.goglesource.com/c/go/+455418>

extra P 畅想



- ① 不参与常规 Goroutine 调度
- ② 没有本地 g 队列
- ③ 不参与辅助 GC
- ④ GC stop the world?

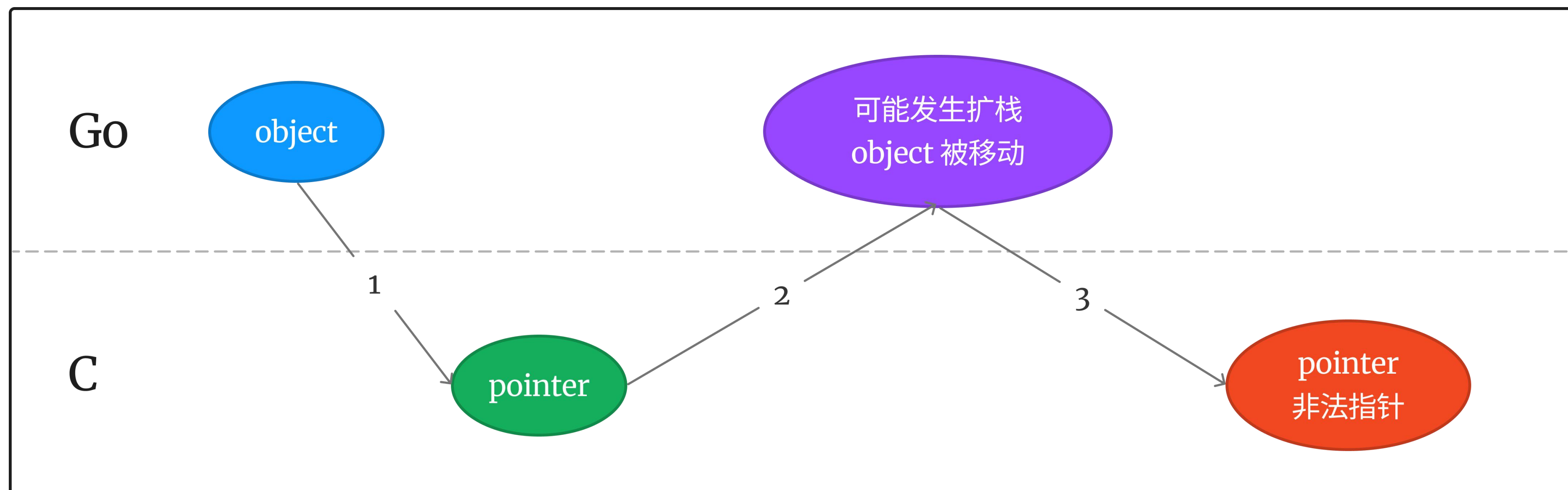
GC 优化

问题：Go 传给 C 的 object **总是** 会被 escape to heap

期望：尽量保留在 stack 上，降低 GC 开销

原因：C 又回调 Go 时，Goroutine 的 stack 可能会移动

```
func (c *httpCImpl) HttpGetStringValue(r unsafe.Pointer, id int) (string, bool) {  
    var value string  
    C.envoyGoFilterHttpGetStringValue(r, C.int(id), unsafe.Pointer(&value))  
    return strings.Clone(value), true  
}
```



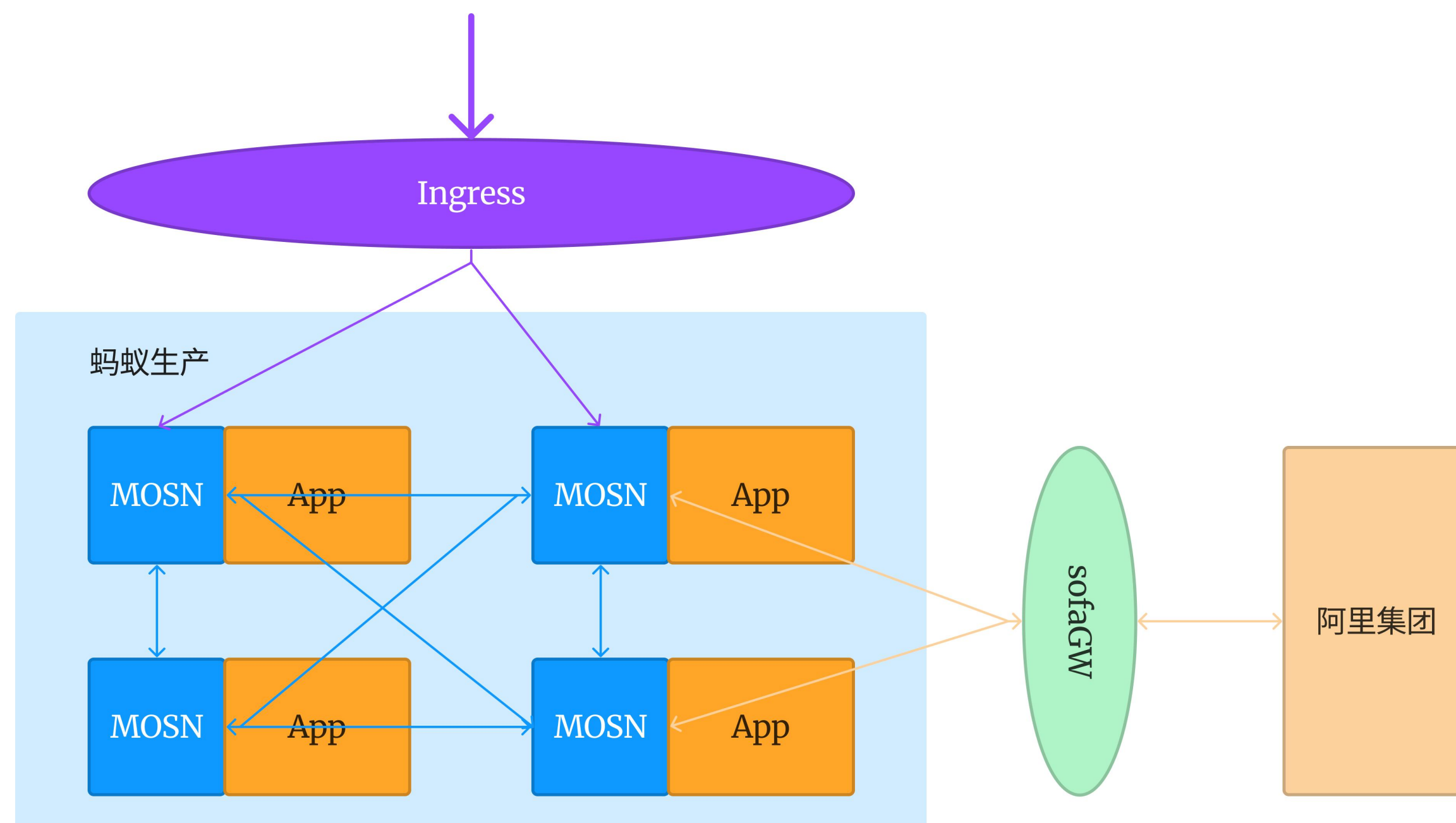
解法：增加 annotations，标记不会回调

<https://go-review.goglesource.com/c/go/+497837>

四. MoE 架构的现状和未来

- 在蚂蚁的落地状况
- 在 Envoy 社区的进展
- 未来基于 MoE 架构的网关产品


MoE 在蚂蚁的落地状况




- ① sofaGW: 最先落地验证
- ② Mesh 2.0: 几十万 Pod 规模
- ③ Ingress: 落地验证中


在 Envoy 社区的进展

- L7 能力逐步完善中
- contrib to standard
- L4 扩展 review 推进中
- Golang plugin hub
- 社区参与越来越活跃


 **Sotiris** 11:10 AM
Hey folks,
I am potentially looking into becoming an early adopter. Want to build a custom envoy filter but the implementation is still TBD.
Golang is nice because the company that I work at is really familiar with Go so it would make maintenance much easier.

 **phlax** 7:15 PM
np - just quite busy so not so much time for blog-writing just now
but happy to review
seems the plugin is steadily growing users/popularity/functionality
s/plugin/filter/

我想在uber这里推动一下envoy的go extension的使用
能不能和你约个时间聊一下?

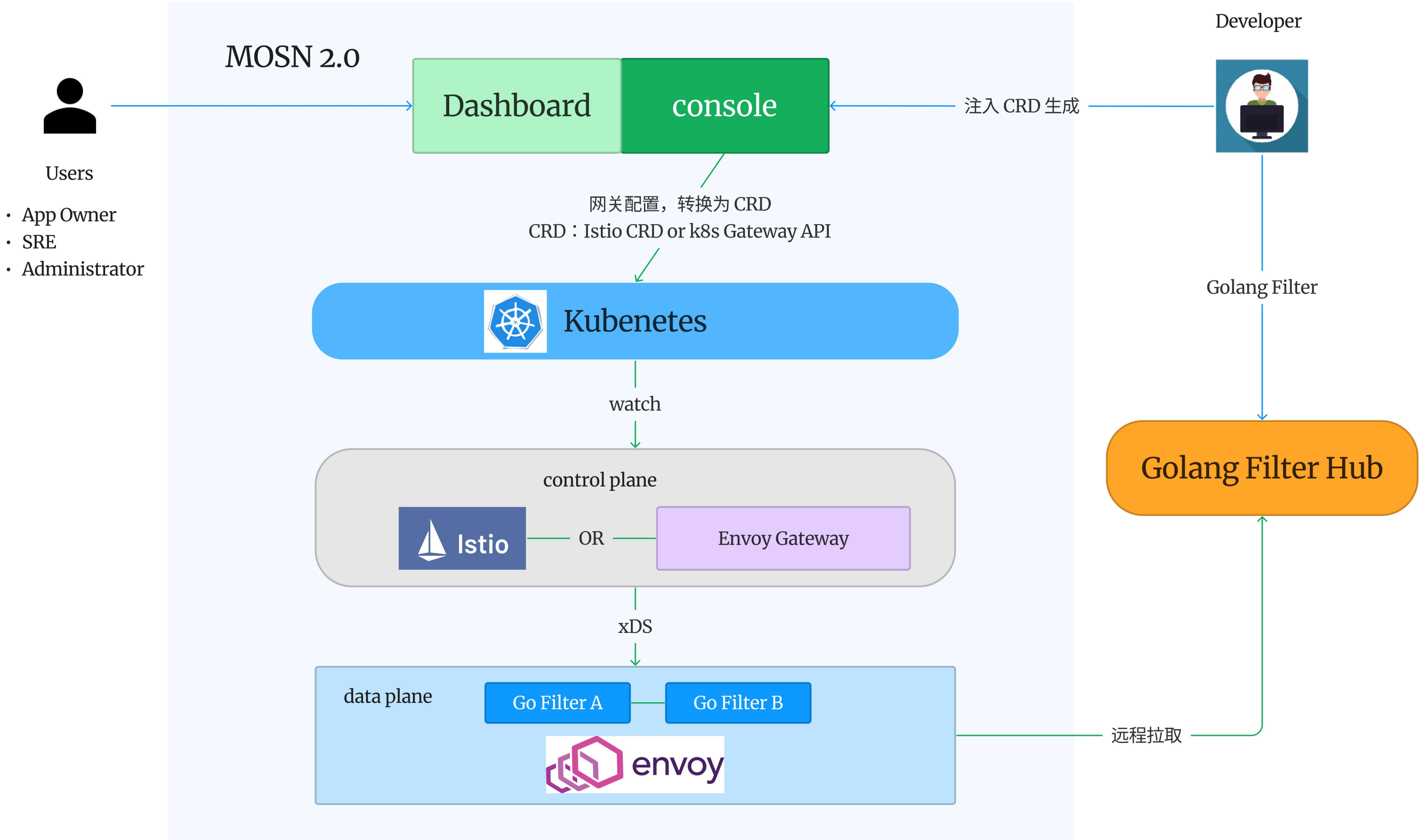
 **istio / proxy** Public
<> Code **Pull requests** 1 Actions Security Insights

[enable http golang filter extension. #4463](#)

 **emissary-ingress / emissary** Public
<> Code Issues 328 **Pull requests** 109 Actions Projects 2 Wiki Security

[envoy: add go-filter protobuf #5057](#)

云原生网关产品



1. 开箱即用
2. 以应用为中心
3. 企业级能力
4. 高度可定制
5. k8s first

总结

- MoE: 需要一个低心智负担的扩展方式
- 三种安全机制
- 优化: 低垂的果实
- MOSN 2.0

MOSN 官网 <http://mosn.io>

MOSN Github <http://github.com/mosn/mosn>

MOSN 开源交流群



钉钉

欢迎技术交流



微信

THANKS

—
软件正在重新定义世界

Software Is Redefining The World