



IT大咖说
知识分享平台

Go微服务实践

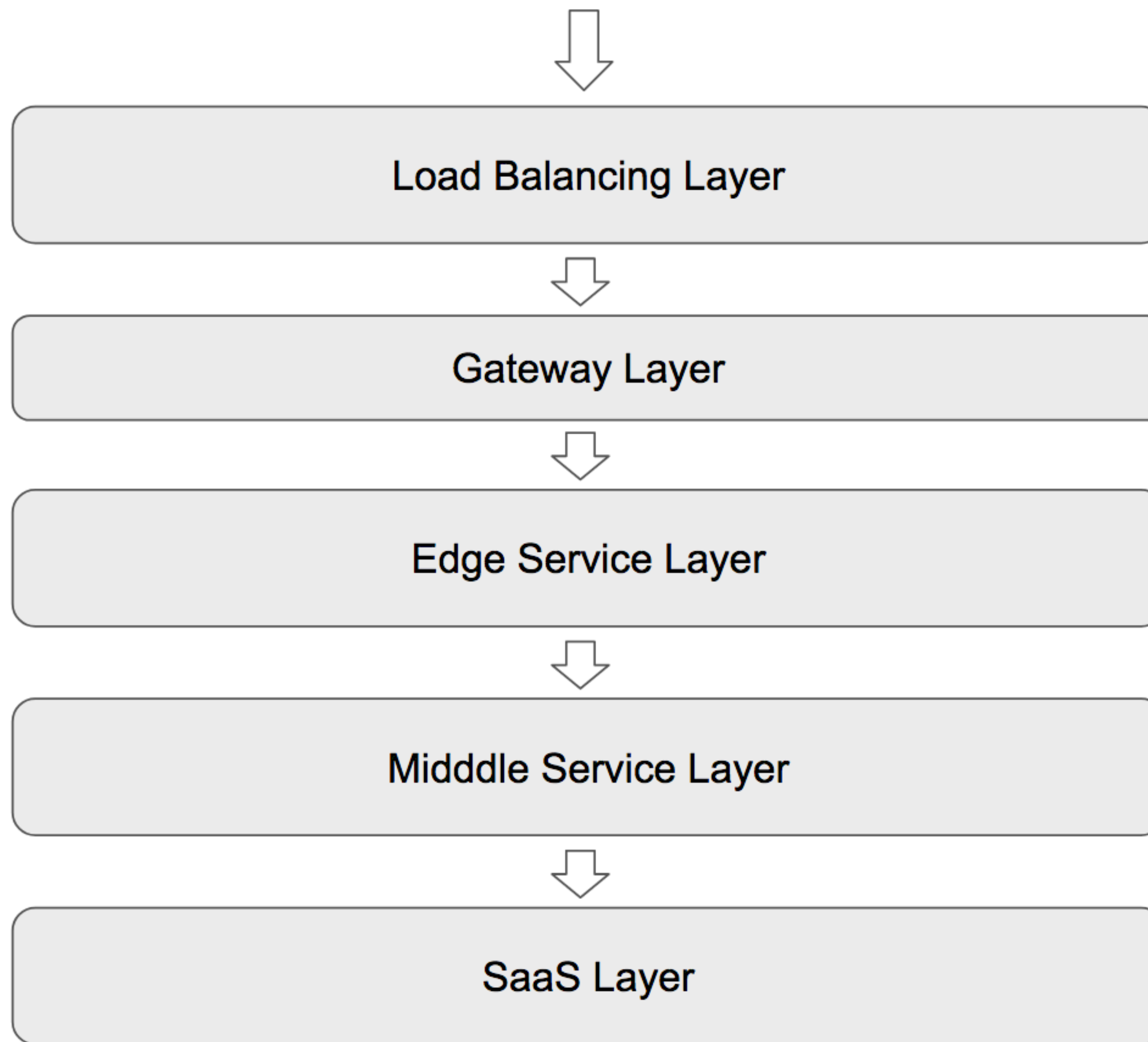
项超

提纲



IT大咖说
知识分享平台

- 微服务基础组件
- 微服务的命名
- 负载均衡
- 超时控制
- RPC抽象与中间件
- 一些探索和思考



基础组件



微服务框架

动态配置管理

服务注册与发现

服务治理

负载均衡

分布式跟踪

.....

为什么要命名



- 服务名
 - 唯一表示一个Service
 - 命名符合规范，便于当作标识符来使用
- RPC过程命名
 - 三元组 (From, To, Method)
 - 五元组 (From, **FromCluster**, To, **ToCluster**, Method)

Thrift中的实践



IT大咖说
知识分享平台

```
struct ExampleRequest {  
    1: string Name,  
    2: i64 ID64  
    3: i32 ID32  
    255: base.Base Base,  
}
```

```
struct ExampleResponse {  
    1: string Resp,  
    255: base.BaseResp BaseResp,  
}
```

```
service ExampleService {  
    ExampleResponse Visit(1: ExampleRequest req)  
}
```

```
struct Base {  
    1: string LogID = "",  
    2: string Caller = "",  
    3: optional map<string,string> Extra,  
}
```

服务治理中的实践



IT大咖说
知识分享平台

服务降级

```
/from/fromCluster/to/toCluster/method/degrade
```

流量切换

```
/from/fromCluster/to/toCluster/method/traffic
```

动态配置

```
/from/fromCluster/to/toCluster/method/config
```

负载均衡



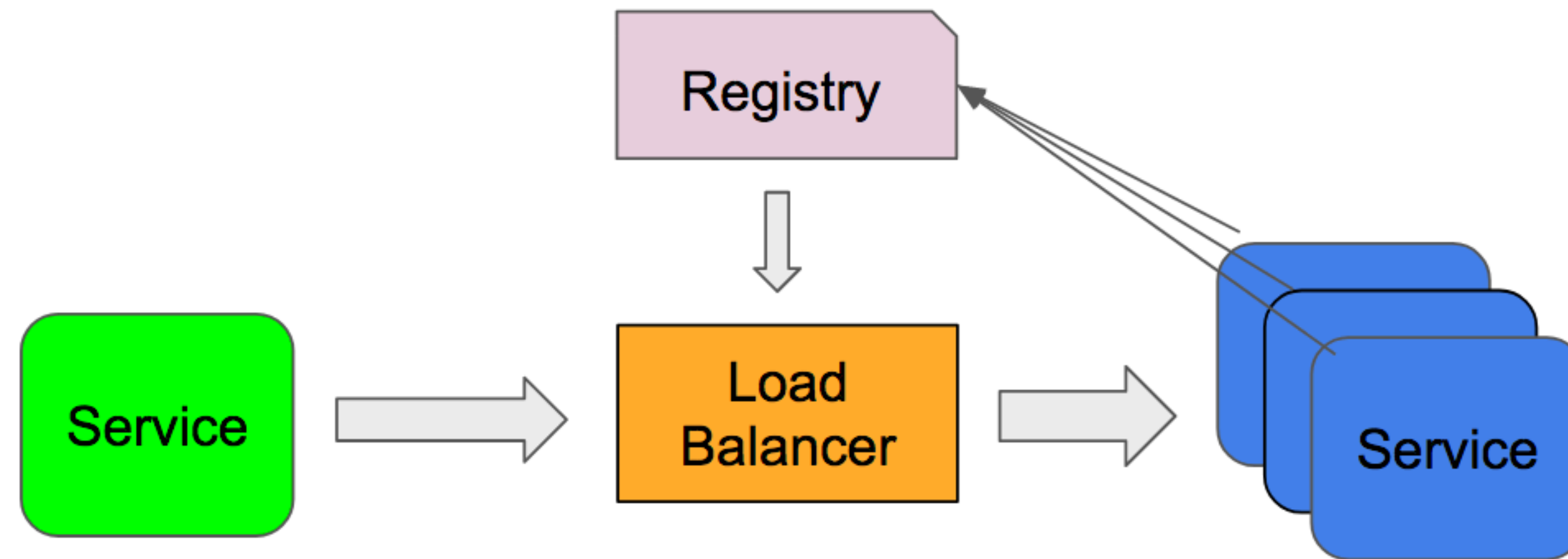
IT大咖说
知识分享平台

- 集中式负载均衡
- 进程内负载均衡
- 独立进程负载均衡

集中式负载均衡



- 实现简单
- 单点
- 增加延迟
- 性能瓶颈

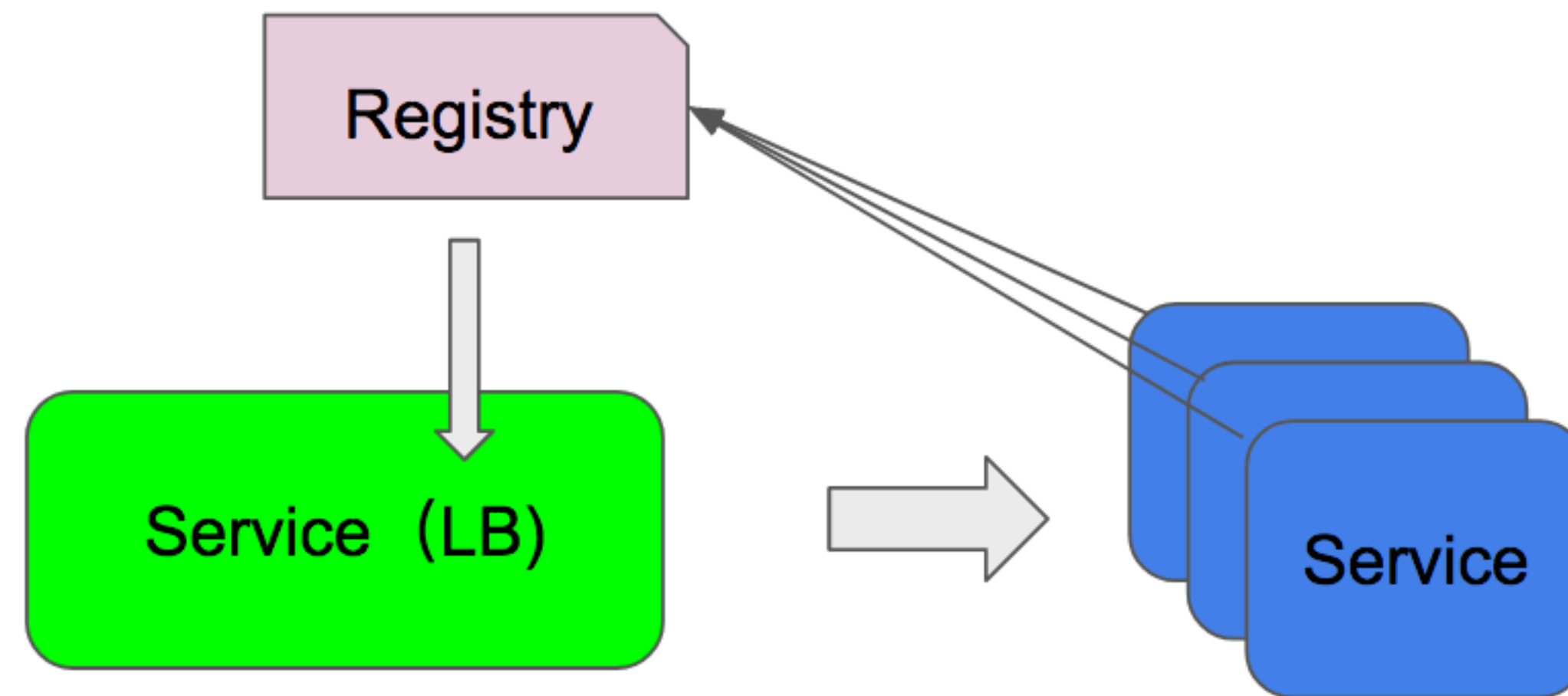


进程内负载均衡



- 无单点
- 无额外延迟
- 无性能瓶颈

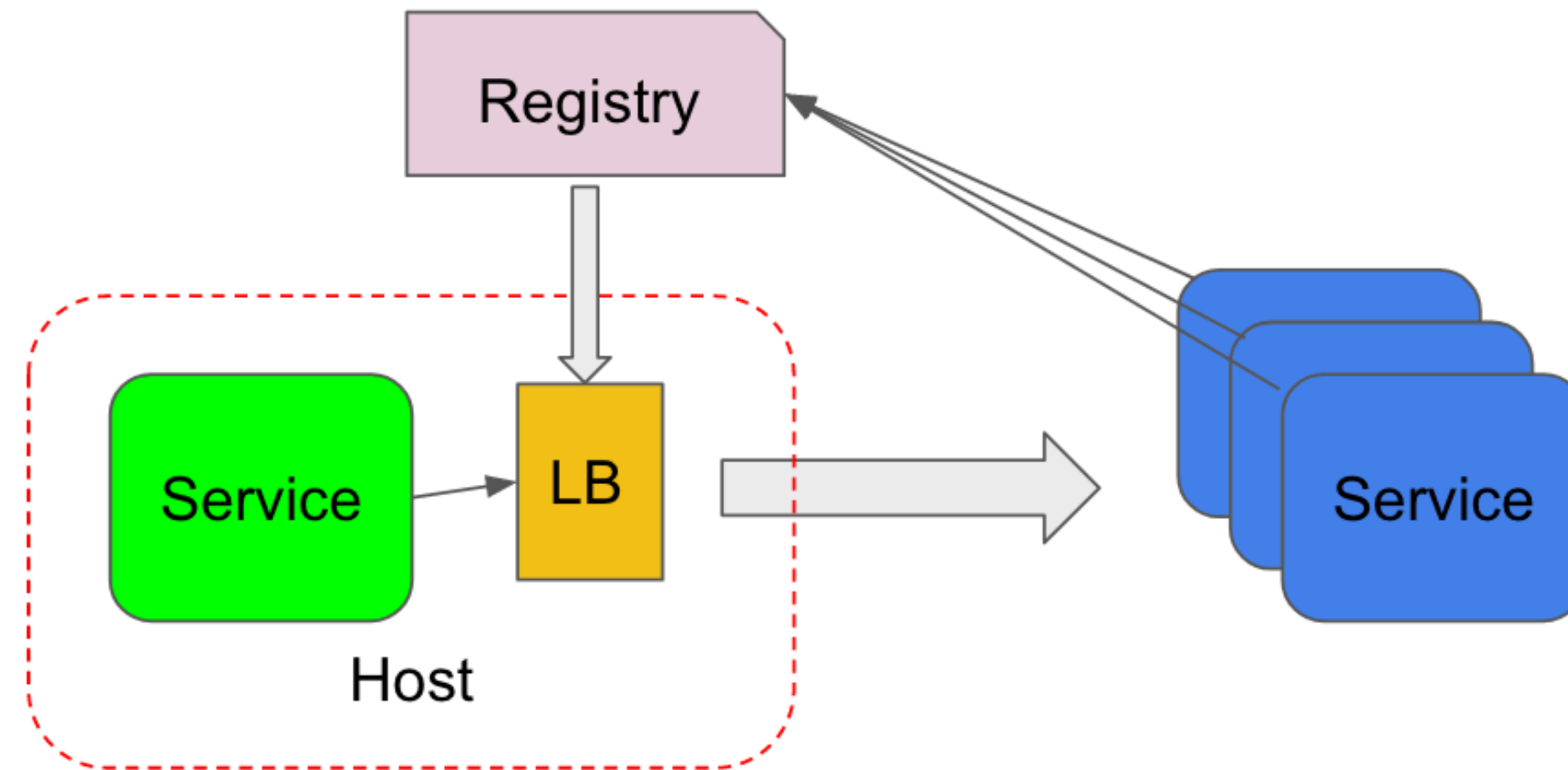
- 每个语言需要实现一个Lib
- 升级代价高



独立进程负载均衡



- 无单点
- 无额外延迟
- 无性能瓶颈
- 每个语言需要实现一个Lib
- 升级代价高



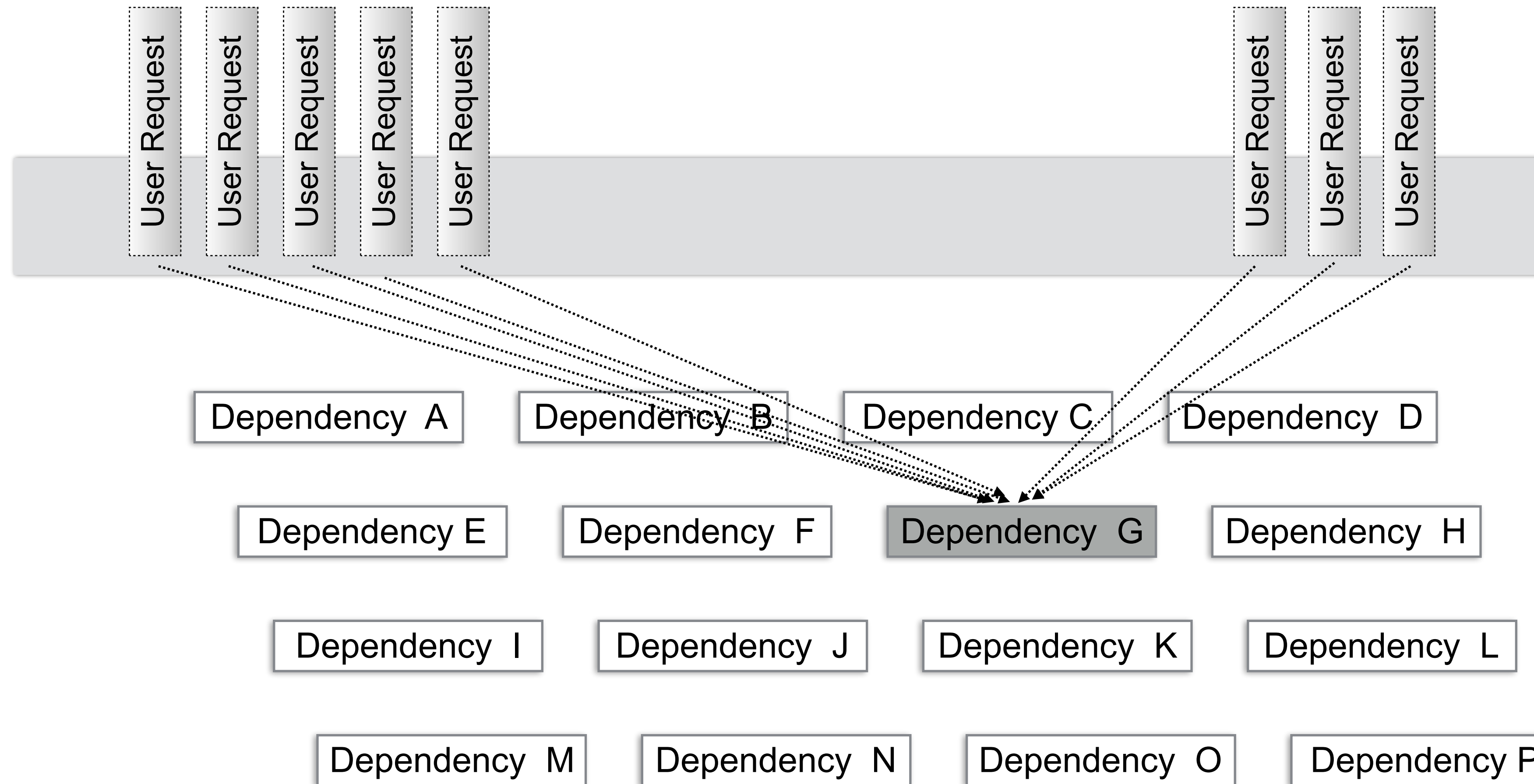
负载均衡的选择



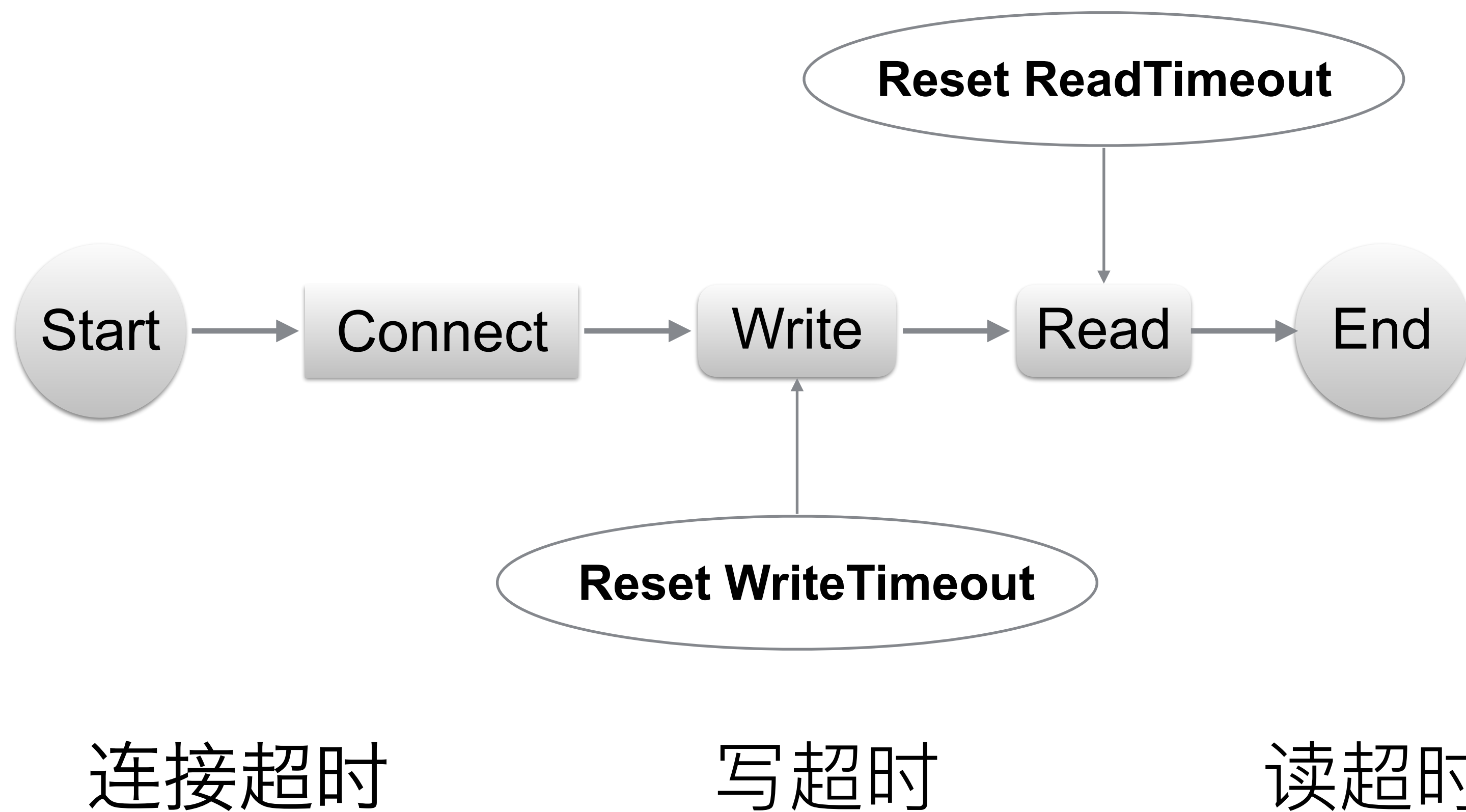
进程内LB

- Thrift协议没有现有的中心化代理服务，自行开发代价高，且由于Thrift协议是基于流的协议，代理的性能成本很高
- 像所有集中式LB一样，存在单点问题，微服务之间的调用本身是网状的，变成星状之后，单点造成的风险过大，一旦LB出问题，影响严重
- 性能问题，类似Thrift这种二进制协议其本身设计就考量了性能要求，集中式LB必然是应用层的LB，增加的网络延迟也将是不可接收的

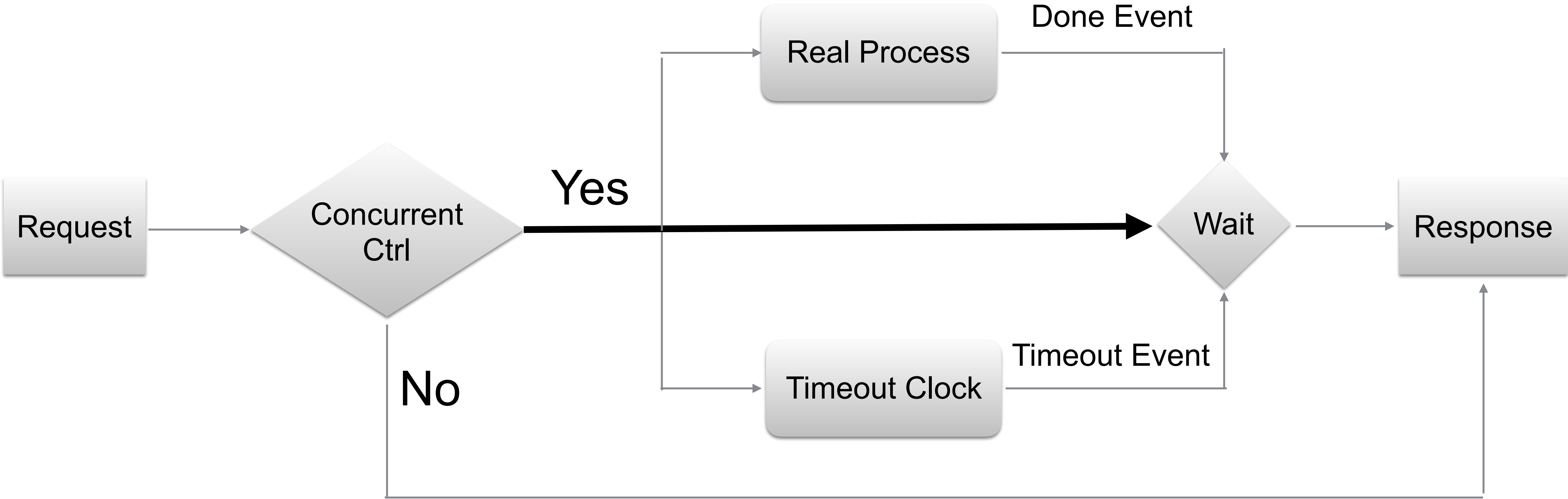
超时控制



网络超时



并发超时控制



RPC抽象与中间件



IT大咖说
知识分享平台

<https://github.com/go-kit/kit>

```
// EndPoint represent one method for calling from remote.  
type EndPoint func(ctx context.Context, req interface{}) (resp interface{}, err error)  
  
// Middleware deal with input EndPoint and output EndPoint  
type Middleware func(EndPoint) EndPoint
```


中间件



IT大咖说
知识分享平台

- 日志
- Trace
- Metrics
- 熔断
- 降级
- 访问控制
- 服务发现
- 连接池
- ...

另一种RPC抽象



- 返回值

```
func(ctx context.Context, req interface{}) (resp interface{}, err error)
```

- 传参数

```
func (ctx context.Context, req interface{}, resp interface{}) error
```

经验和教训



IT大咖说
知识分享平台

- context变量不可随意使用
- 尽早监控协程数，大规模Go服务下，协程暴涨很容易发生
- 谨慎对待Cgo调用
- 锁变量而不要锁过程

一些探索和思考



IT大咖说
知识分享平台

- 寻求更加高效Go语言Thrift代码生成器
- 基于Binary协议实现独立进程的LB模型
- 贯穿调用链的超时控制模型



IT大咖说
知识分享平台

Thanks!