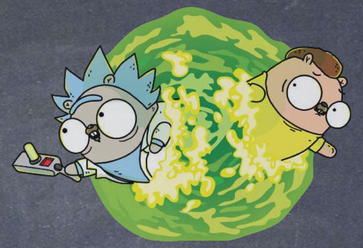


构建分布式任务系统

- xiaorui.cc

- github.com/rfyiamcool

1

出师有名

shark ?

- kubernetes
  - deploy k8s cluster
  - manage k8s state
  - backup k8s config
  - migrate k8s zone
- manage node like ansible
  - template + scripts
  - execute shell
  - custom module
  - custom plugin

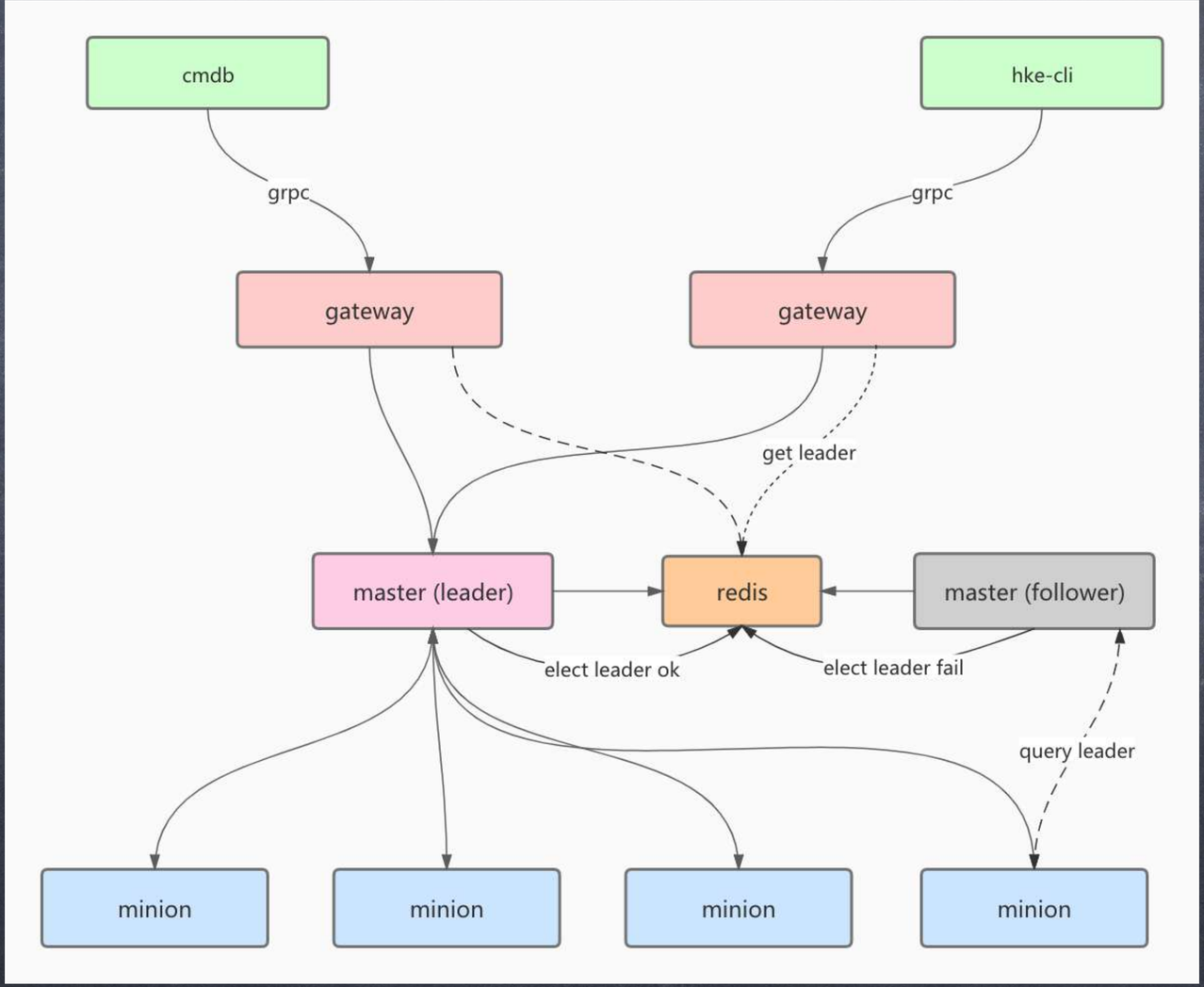
shark ?

- standard
  - os init
  - base init
  - extend init
  - cloud init
  - ...
- deploy & manage server
  - kafka, rabbitmq
  - rocketmq cluster
  - redis, mysql, mongodb cluster
  - ...
- more feature

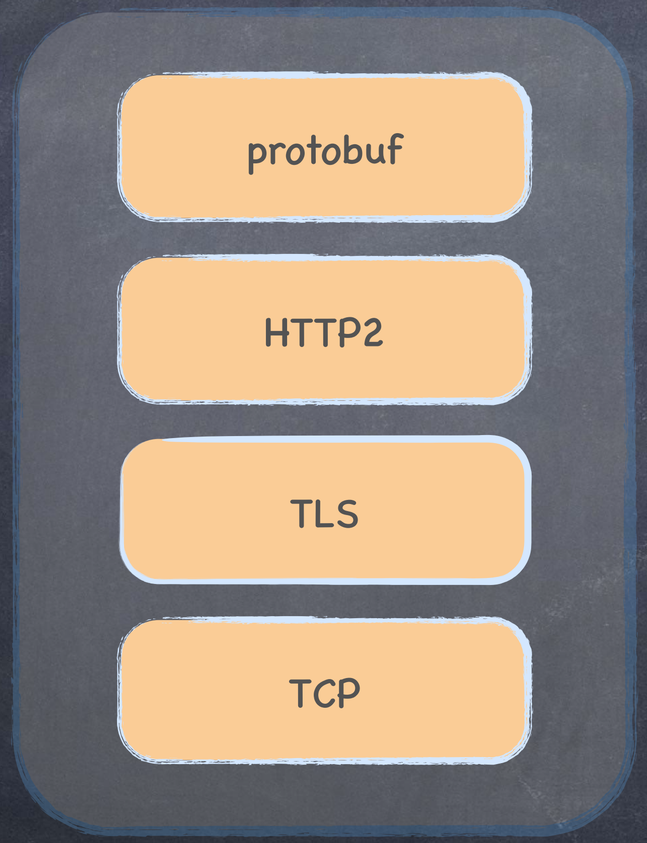
2

架构设计

架构设计


cmdb
hke-cli
grpc
grpc
gateway
gateway
get leader
master (leader)
redis
master (follower)
elect leader ok
elect leader fail
query leader
minion
minion
minion
minion

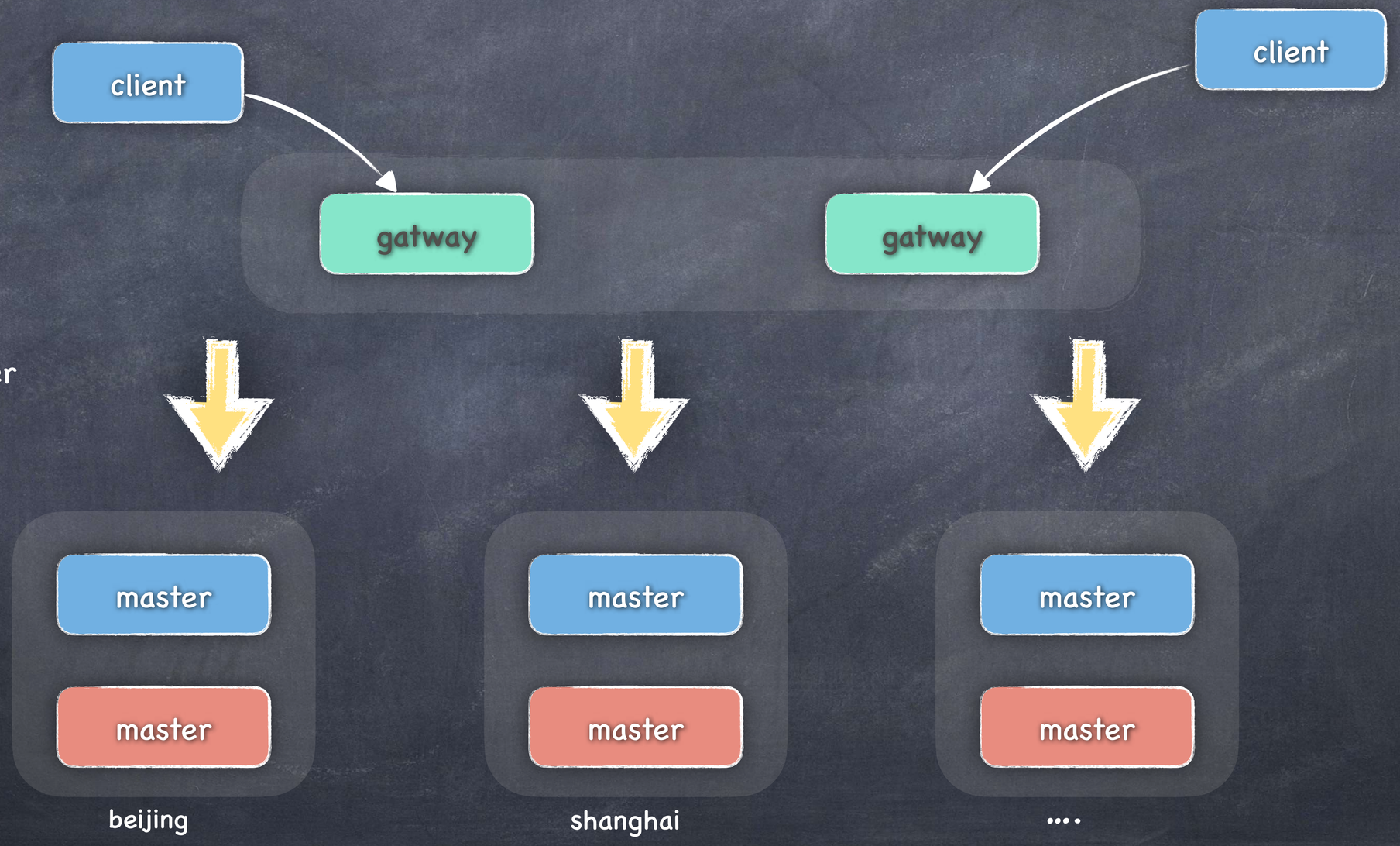
grpc



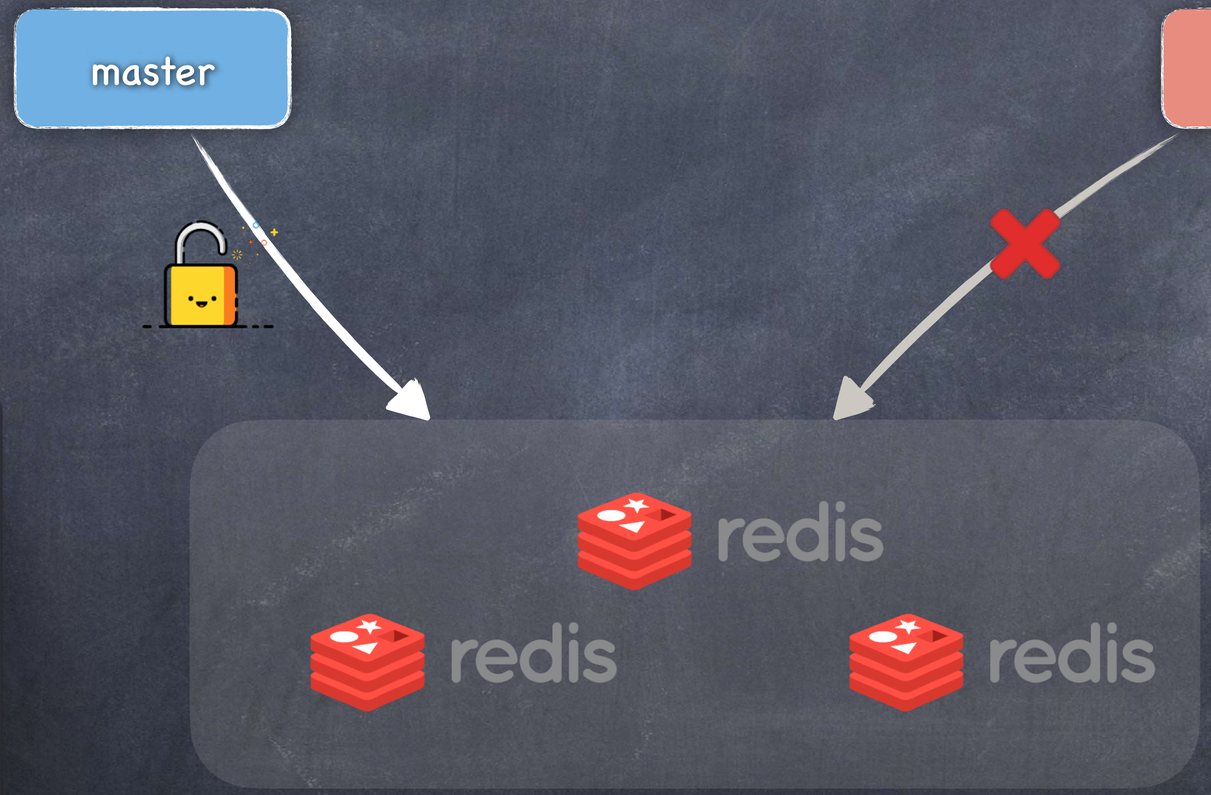
- 优点
  - 支持各类语言
  - 基于http2兼容好
  - 支持bidi全双工通信模式
  - protobuf
    - 高性能序列化
    - tag及数据压缩
    - 类型约束

gateway

* dispatch zone
* redirect leader master
* wrap logic



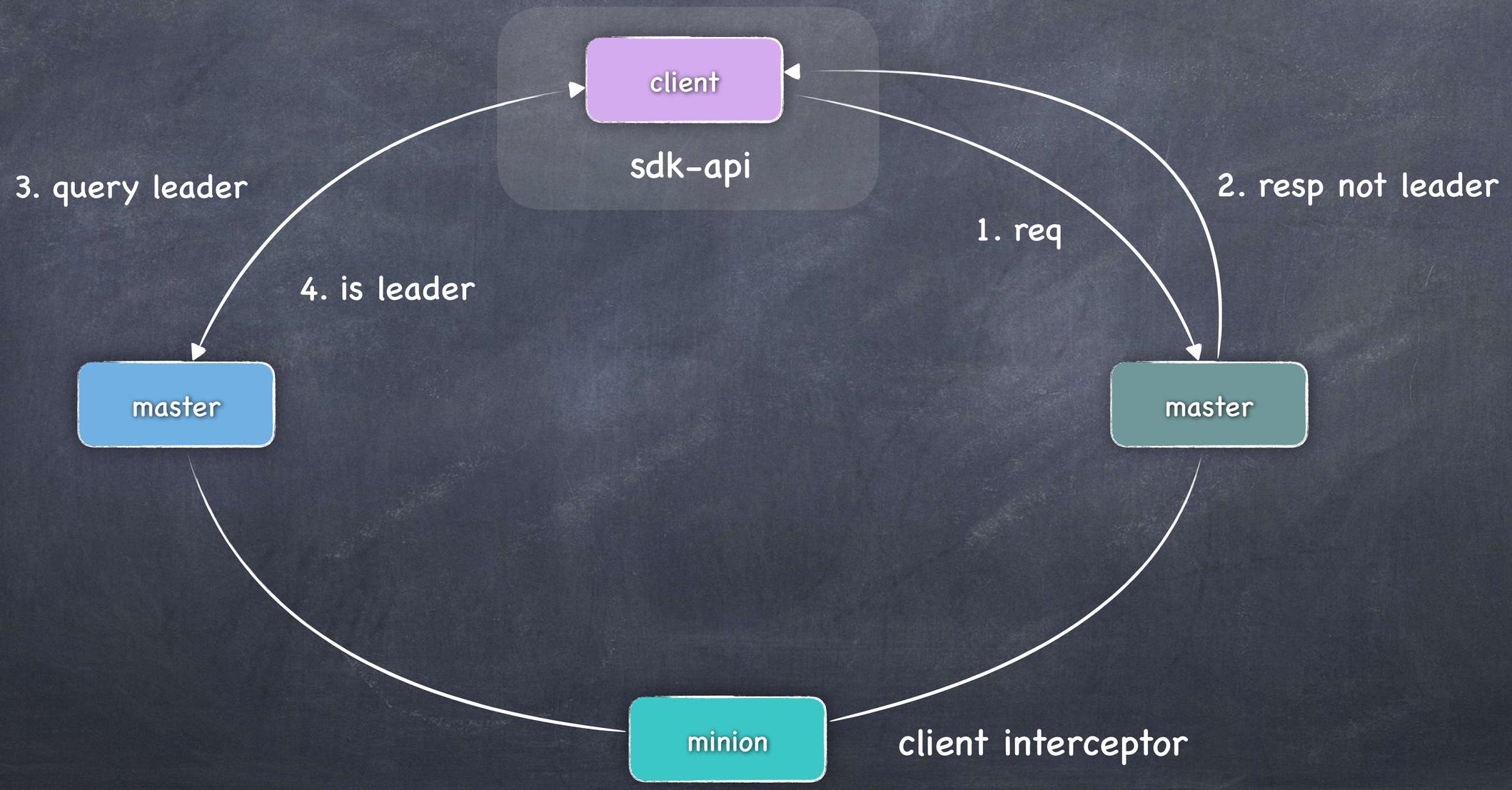
leader election



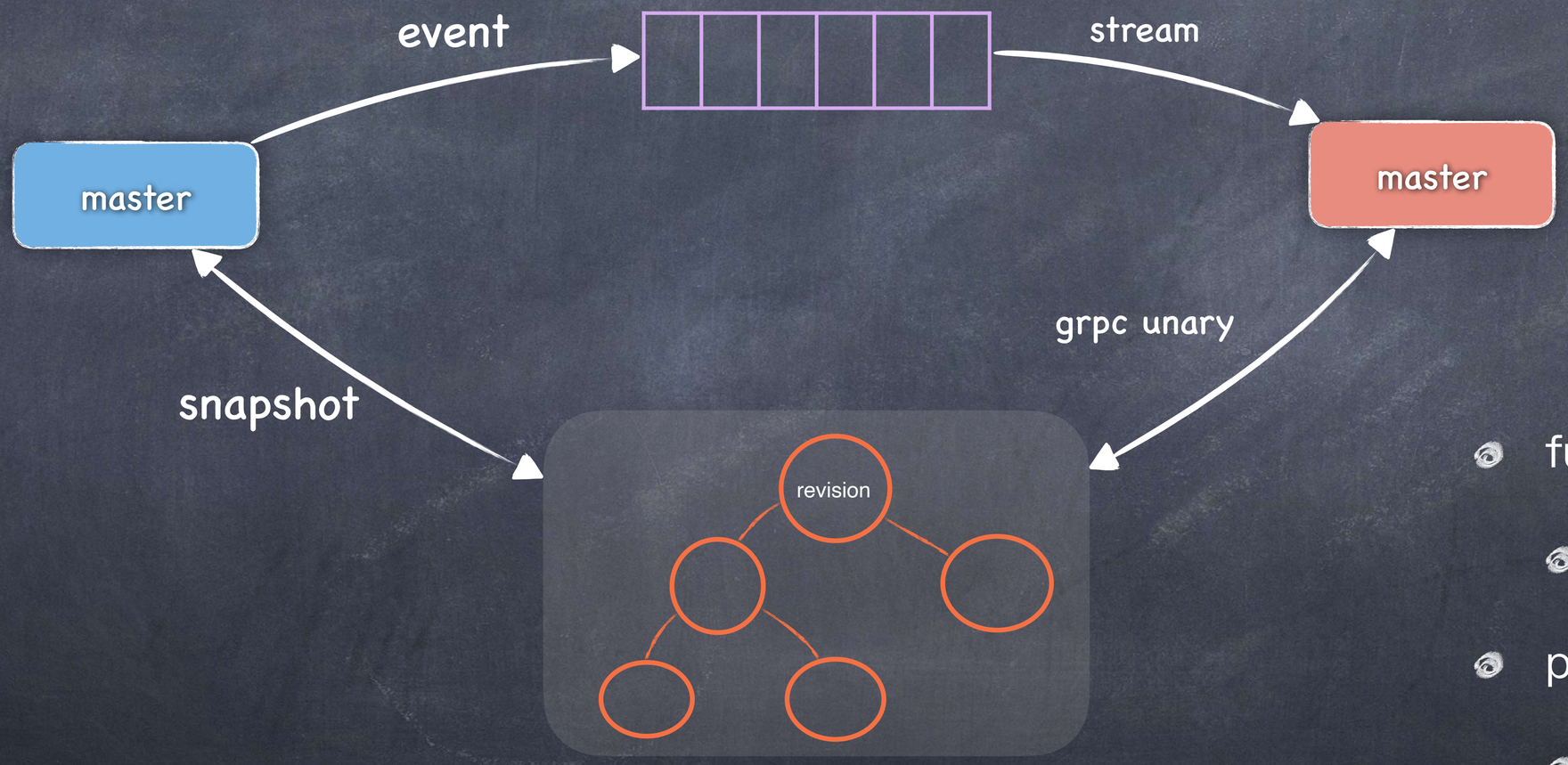

```
func() {
    if acqiure() {
        for {
            ttl(N)
            time.sleep(N)
        }
    }
}
```

- redis + lua
- acqiure
- release
- ttl

interceptor



sync replica



- fullsync
  - 全量数据
- psync
  - 增量数据

grace upgrade

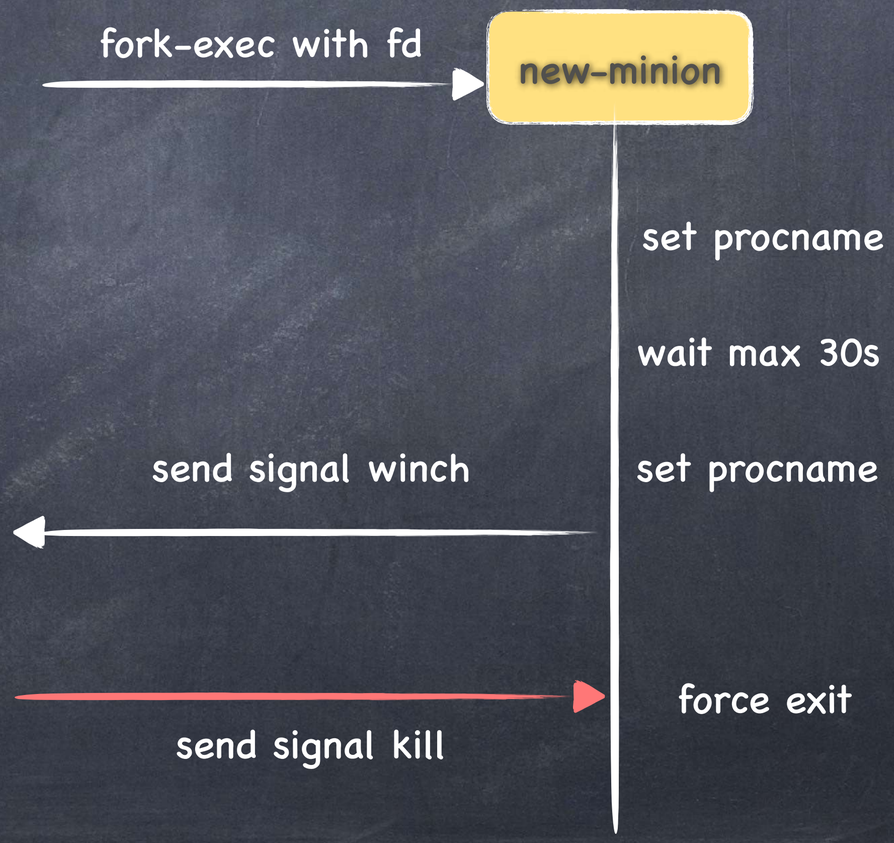
golang没有原生的fork和exec syscall !!!

1. 询问是否可升级？
2. 下载升级文件
3. 检验升级文件md5、version及试运行自检
4. fork + exec 新文件携带listen fd
5. 主进程加入超时事件并监听winch信号
6. 子进程在服务运行30s后，向主进程触发winch信号
7. 主进程收到winch触发grpc grace shutdown()

- watch sig winch
- check if pid is running ?
- add timeout event

grace shutdown

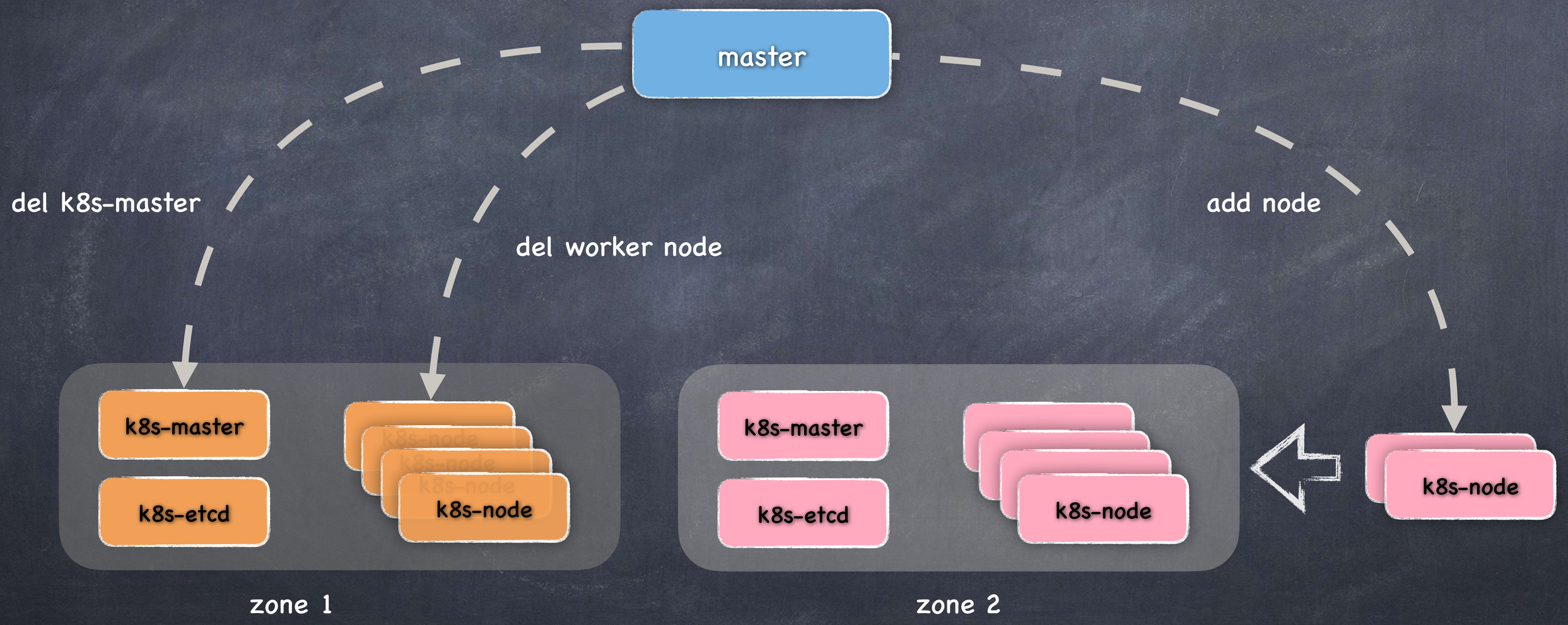
if rase timeout ?



3

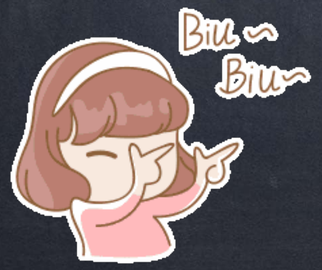
功能设计

K8S


master
del k8s-master
del worker node
add node
k8s-master
k8s-etcd
k8s-node
zone 1
k8s-master
k8s-etcd
k8s-node
zone 2
k8s-node

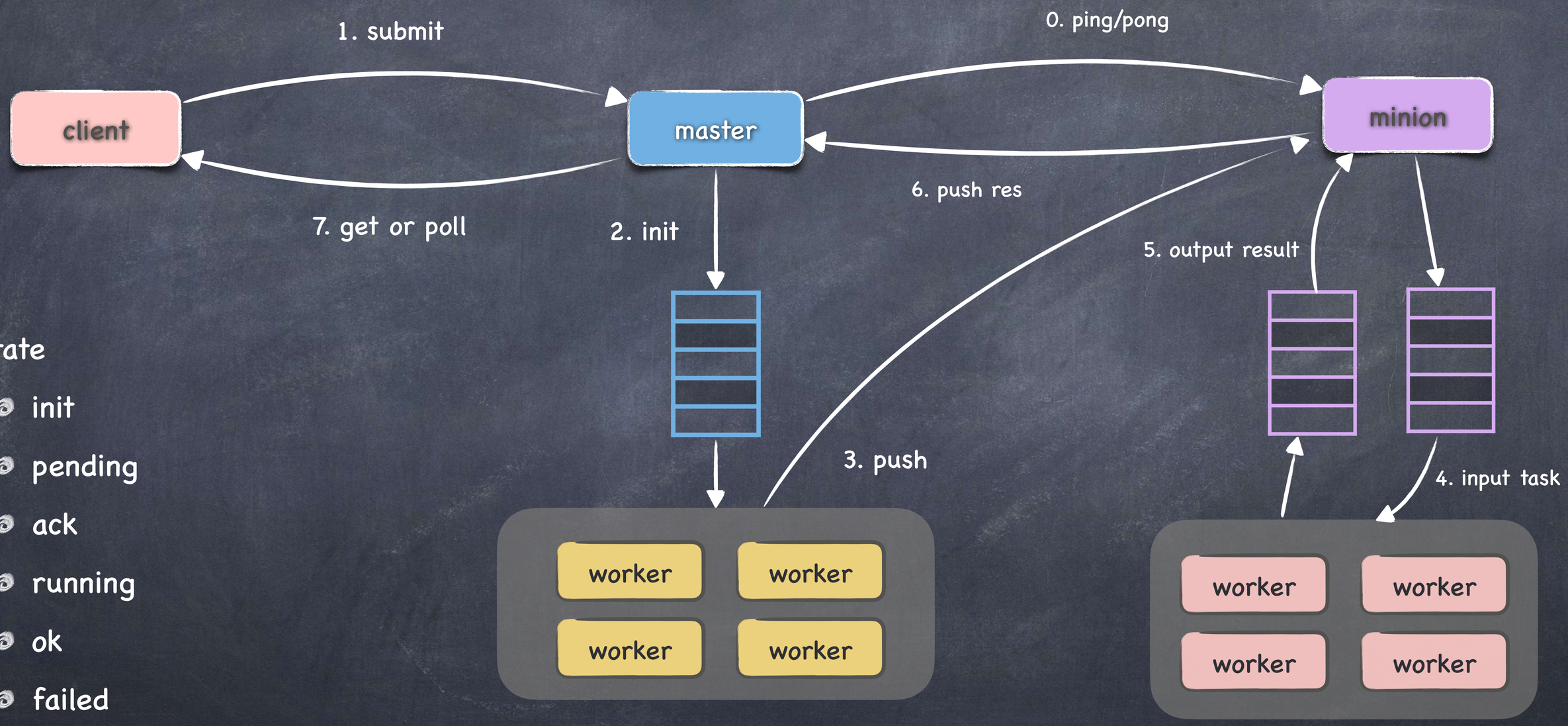
K8S

- 基于rancher rke二次开发
  - 改造力度很大
  - 很大
  - 大
  - .



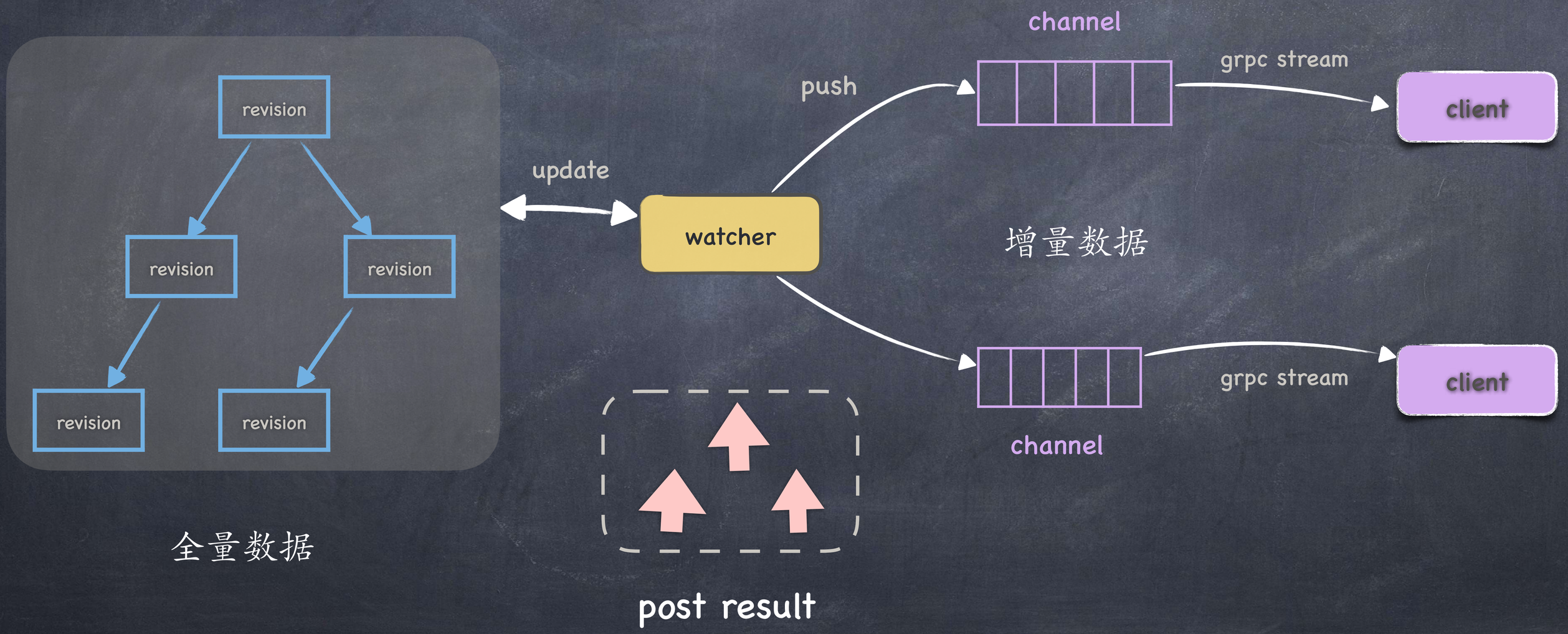
- 加入更多的预先条件检查
- k8s组件使用hyperkube运行
- 可增减k8s master及node节点
- 实时进度流程反馈
- 更详细的参数定制
- 可传入自定义插件
- 自由的镜像版本选择
- …

任务的异步化

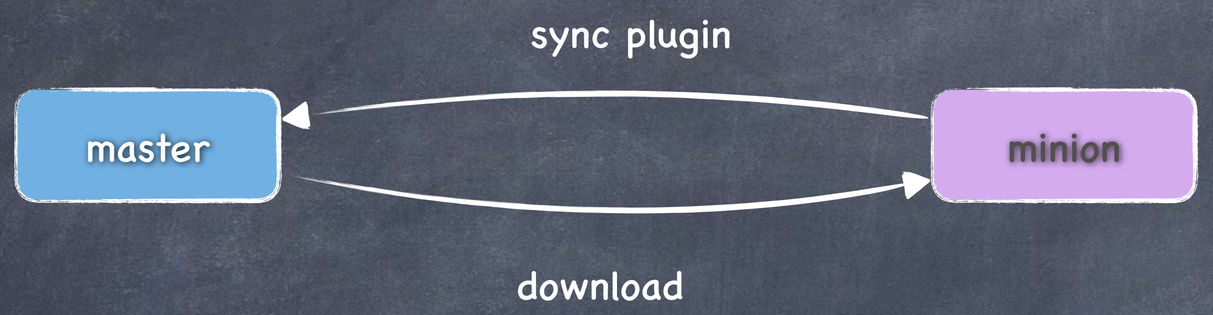


- state
  - init
  - pending
  - ack
  - running
  - ok
  - failed

watch api


revision
revision
revision
revision
revision
全量数据
update
watcher
push
channel
grpc stream
client
增量数据
channel
grpc stream
client
post result

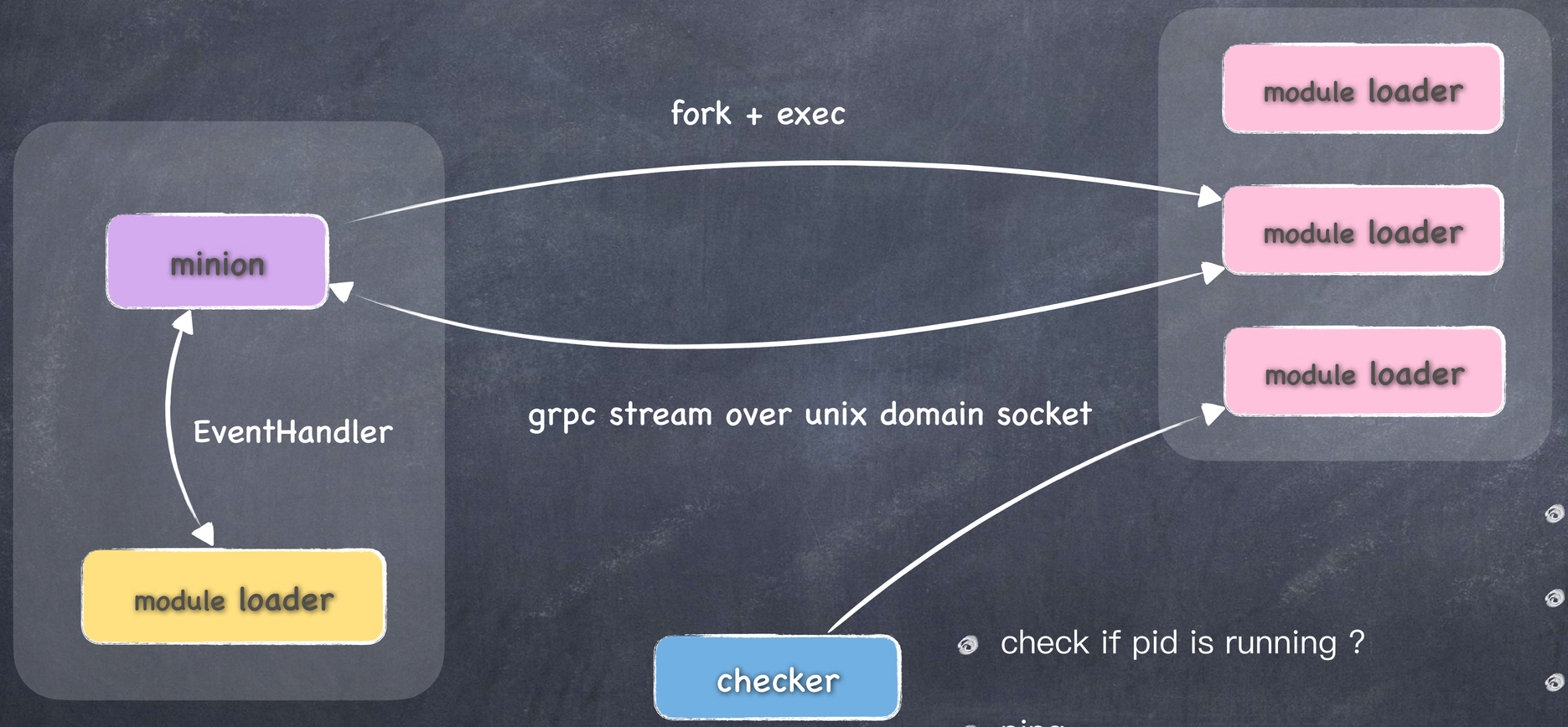
golang plugin



- plugin优点
  - 减少主代码的污染
  - 动态库的热加载
  - 加快编译速度
- 设计
  - 使用子进程运行不安全plugin
  - 对go plugin进行压缩传输
  - 申明多种任务类型的interface{}

```go
switch req.ActionType {
case proto.ActionType_Daemon:
    err = handleDaemon(req, meta)

case proto.ActionType_Once:
    err = handleOnce(req, meta)

case proto.ActionType_Delay:
    err = handleDelay(req, meta)

case proto.ActionType_Ticker:
    err = handleTicker(req, meta)

case proto.ActionType_Crontab:
    err = handleCrontab(req, meta)

case proto.ActionType_Times:
    err = handleRunTimes(req, meta)

default:
    return errUnknownType
```

golang modules



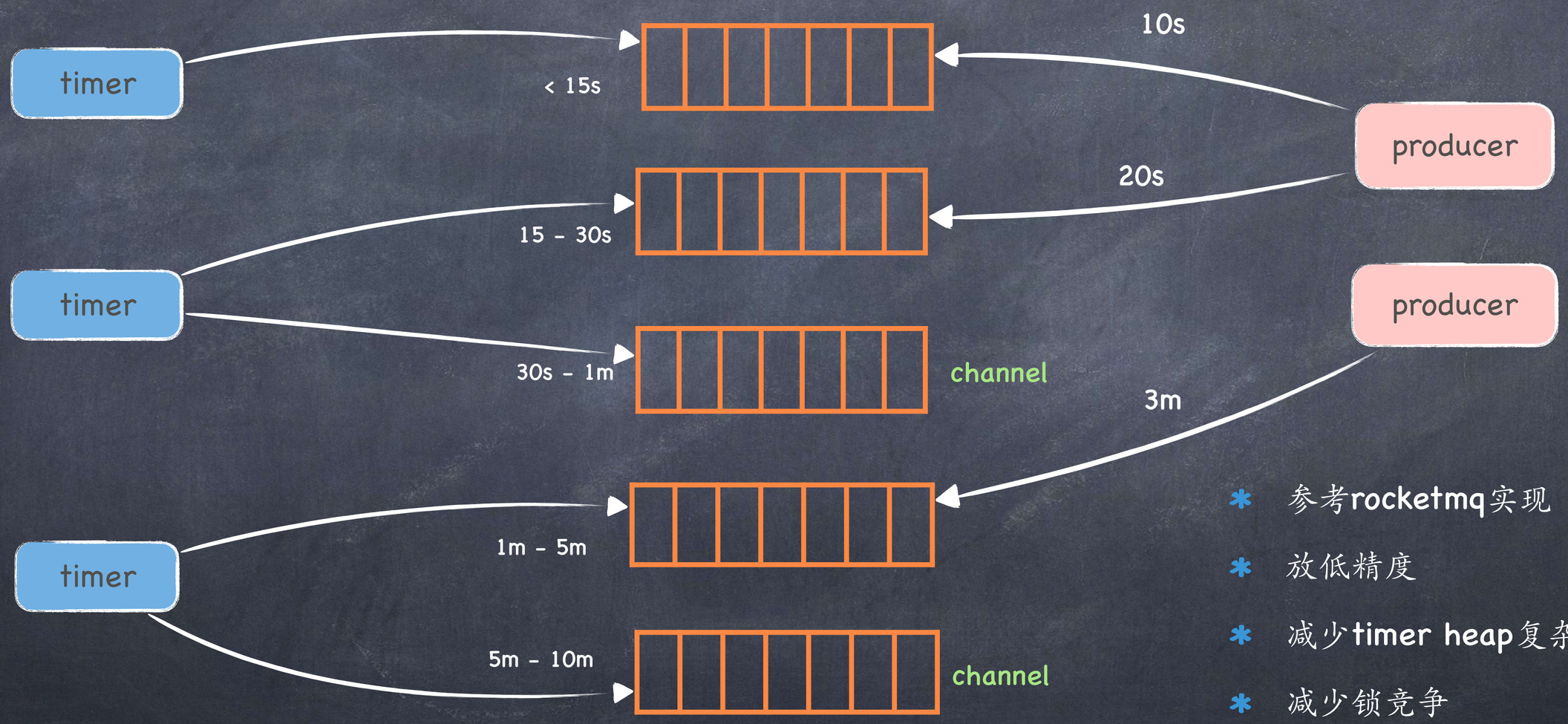
- check if pid is running ?
- ping
- idle timeout

- 各个业务modules实现自注册
- 开放evnetHandler方法
- 运行模式
  - 安全模式
  - 非安全模式

```go
type EventHandler interface {
    GetTaskID() string
    GetReqData() *proto.MinionModuleReq
    GetModule() string
    GetMethod() string
    OK(string) error
    Fail(string) error
    Running(string) error
    NotifyOK(*EventData) error
    NotifyFail(*EventData) error
    NotifyRunning(*EventData) error
    GOGO(func() error) error
    Sleep(time.Duration) error
}
```

```go
func init() {
    modules.Register("example", "test", ModuleTestHandler.Test)
    modules.Register("example", "fast", ModuleTestHandler.Fast)
    ...
}

func (i *ModuleTest) Fast(react modules.EventHandler) error {
    for i := 0; i < 100; i++ {
        react.NotifyRunning(&modules.EventData{
            Mesg: fmt.Sprintf("fast running %d", i),
        })
        time.Sleep(50 * time.Millisecond)
    }

    react.NotifyOK(&modules.EventData{
        Mesg: "fast ok",
    })

    return nil
}
```

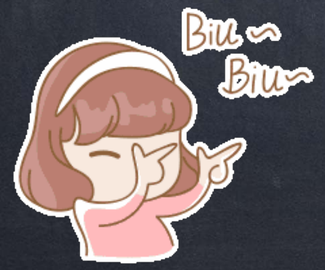
timer



* 参考rocketmq实现
* 放低精度
* 减少timer heap复杂度
* 减少锁竞争

kvstore

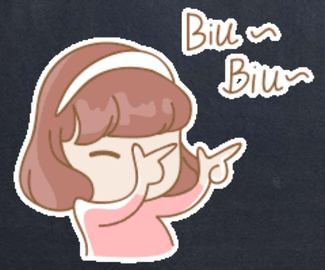
- 使用go-badger实现minion本地存储
- 借鉴redis来设计复杂数据结构
- 简易接口实现
- 由gcworker来定时轮询清理过期数据



- string
  - +keyname,S = expire | length
  - S[keyname] = value
- list
  - +keyname,L = expire | length
  - L[key] = value
- hash
  - +keyname,H = expire | length
  - H[keyname]fieldname = value
- sorted set
  - +key,Z = expire | length
  - Z[keyname]m_member = score
  - Z[keyname]s_score = "member valud"
- ...

安全

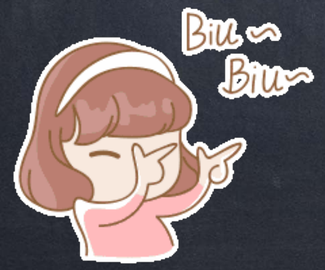
- 正向
  - client -> master -> minion
    - 链路有证书
- 反向
  - minion -> master -> client
    - 另一组证书
- 防止gdb
  - 去除符号及调试信息



- 对client加入额外的token检验
- IP的黑白名单
- auth错误次数超过阈值则进行锁定
- minion token机制
  - minion启动时使用master公钥构建token投递给master
  - master用密钥解析token, 用该token才可访问minion

兜底

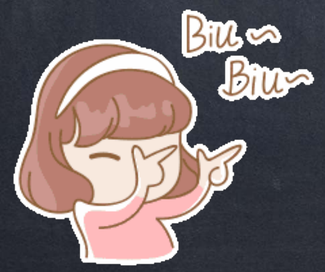
- if minion process crash ?
  - systemd
- if minion node crash ?
  - no way



- if master node crash ?
  - 重新选举并更正节点的状态
  - minion监听选举状态，连接leader节点
- if master logic panic ?
  - wrapper recovery
- if master process crash ?
  - systemd

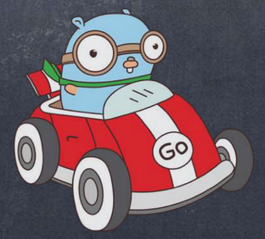
protect

- rate limiter
  - ip
  - service/method
- backoff 退避算法
- failfast 快速失败
- failover 失效转义

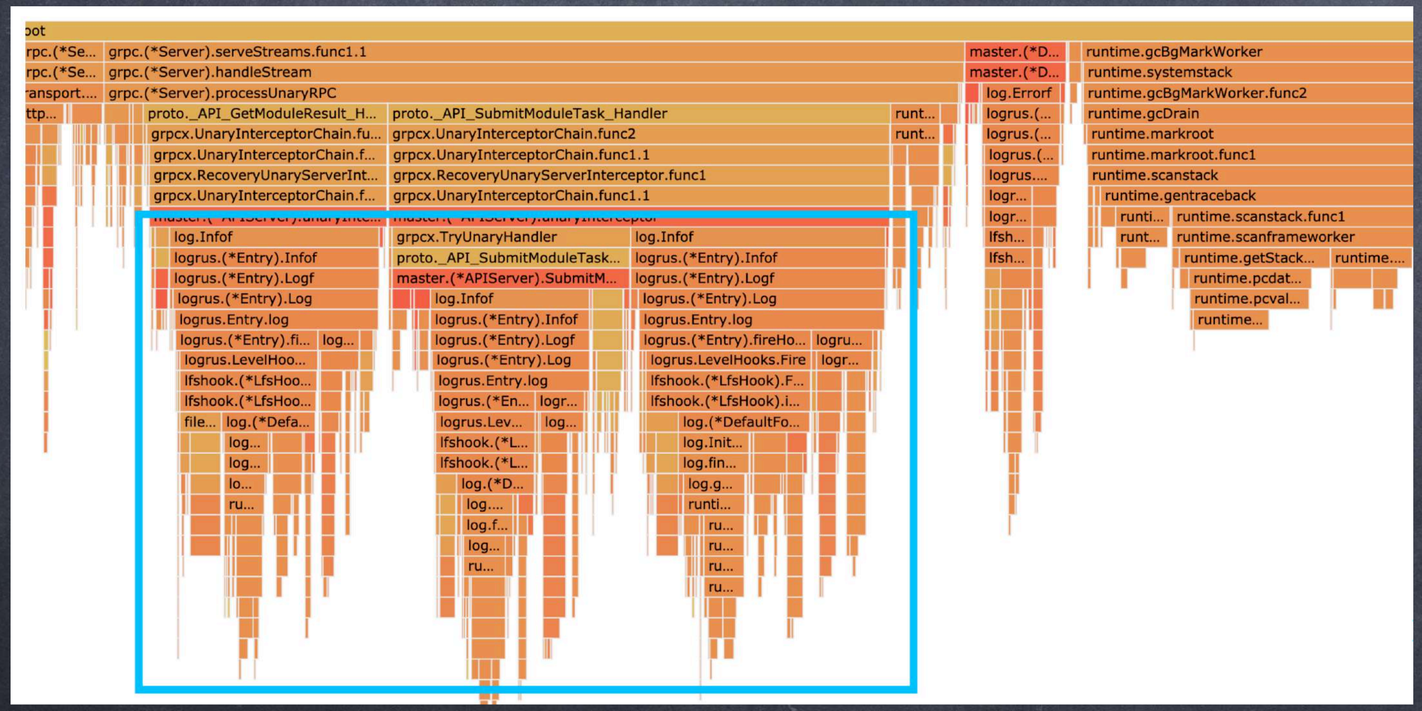


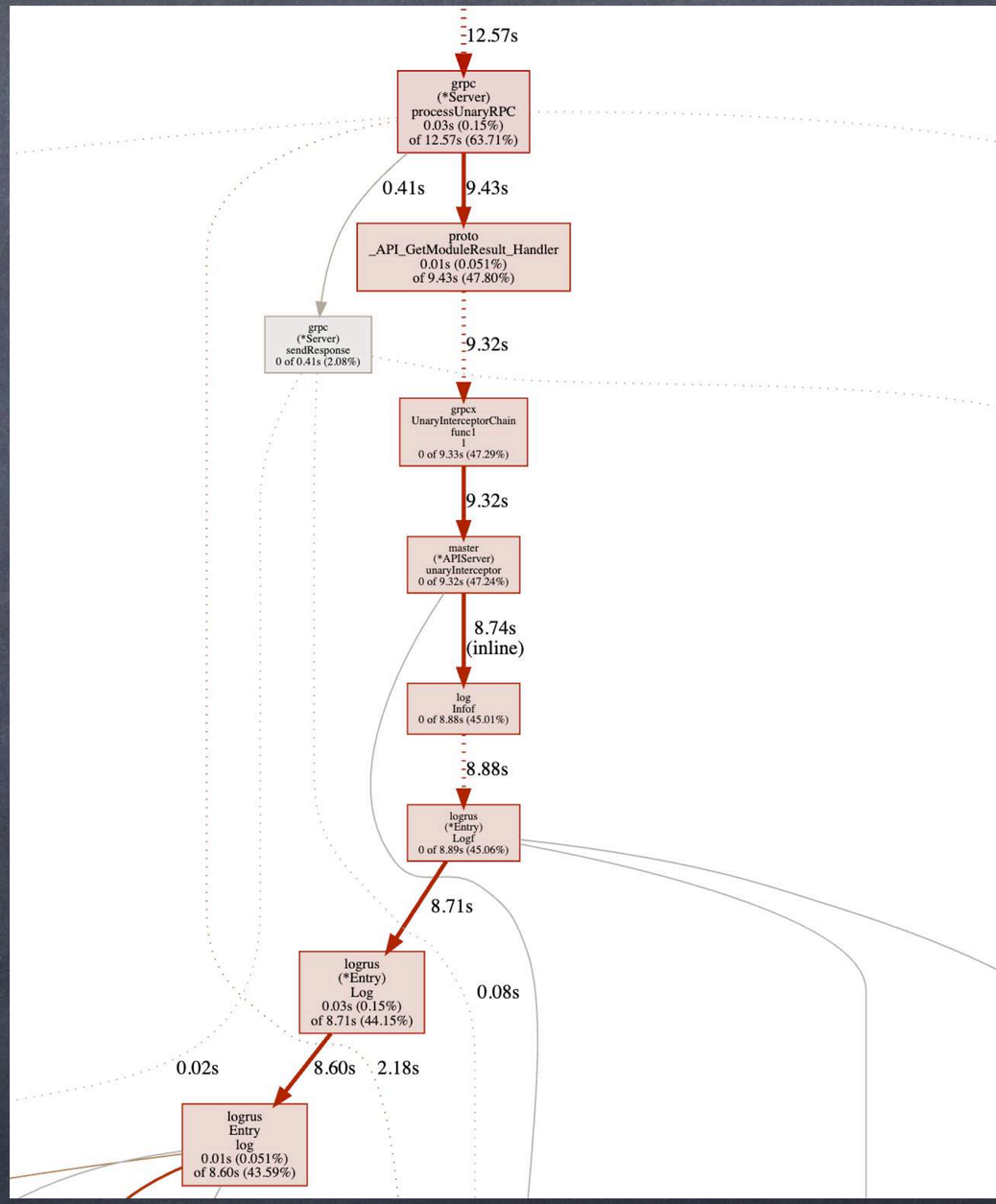
3

optimize

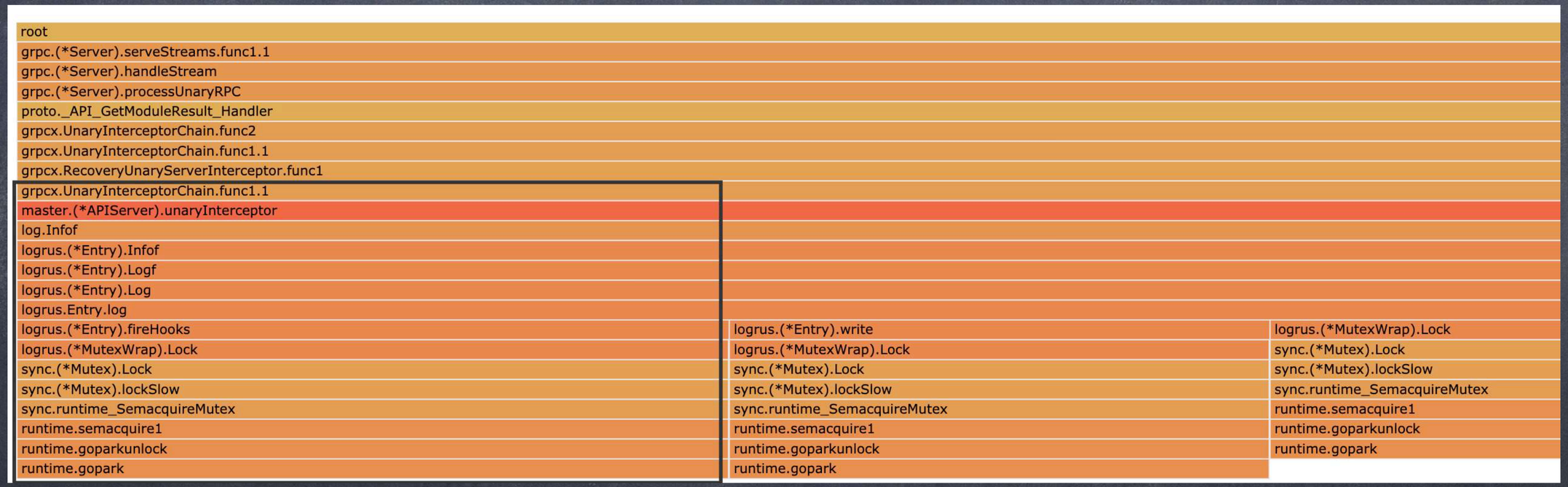


cpu on


grpc.(*Server).serveStreams.func1.1
grpc.(*Server).handleStream
grpc.(*Server).processUnaryRPC
proto._API_GetModuleResult_H...
proto._API_SubmitModuleTask_Handler
grpcx.UnaryInterceptorChain.fu...
grpcx.UnaryInterceptorChain.func2
grpcx.UnaryInterceptorChain.f...
grpcx.UnaryInterceptorChain.func1.1
grpcx.RecoveryUnaryServerInt...
grpcx.RecoveryUnaryServerInterceptor.func1
grpcx.UnaryInterceptorChain.f...
grpcx.UnaryInterceptorChain.func1.1
log.Infof
logrus.(*Entry).Infof
logrus.(*Entry).Logf
logrus.(*Entry).Log
logrus.Entry.log
grpcx.TryUnaryHandler
proto._API_SubmitModuleTask...
master.(*APIServer).SubmitM...
runtime.gcBgMarkWorker
runtime.systemstack
runtime.gcBgMarkWorker.func2
runtime.gcDrain
runtime.markroot
runtime.markroot.func1
runtime.scanstack
runtime.gentraceback
runtime.scanstack.func1
runtime.scanframeworker
runtime.getStack...
runtime.pcdat...
runtime.pcval...
runtime...

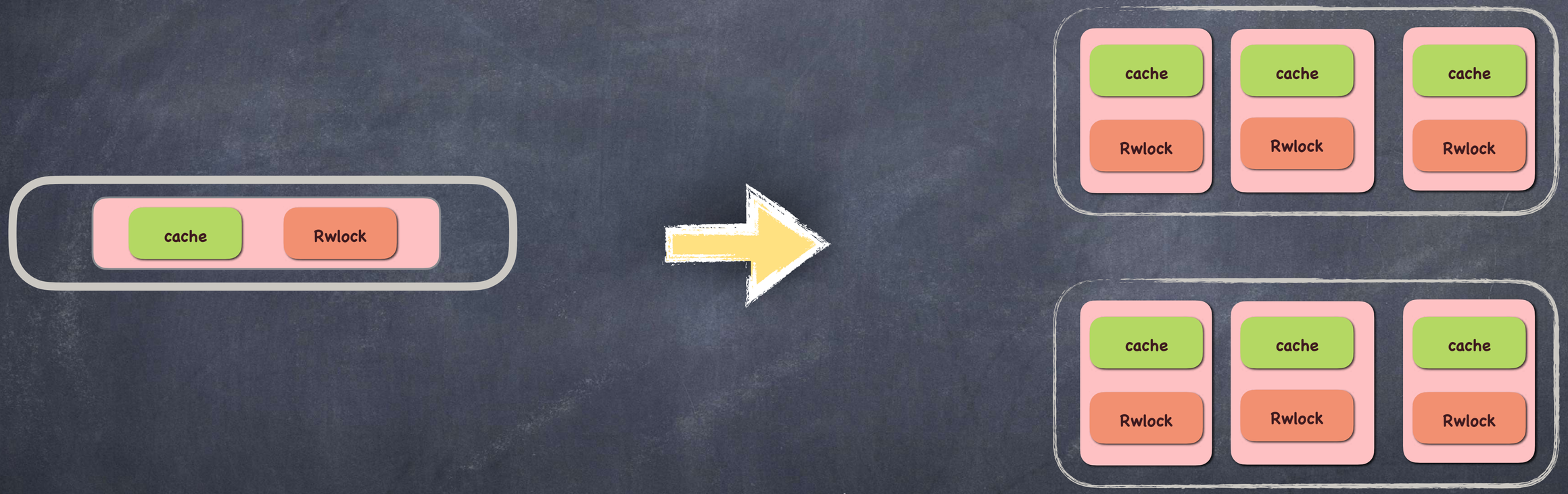

12.57s
grpc (*Server) processUnaryRPC 0.03s (0.15%) of 12.57s (63.71%)
0.41s
9.43s
proto _API_GetModuleResult_Handler 0.01s (0.051%) of 9.43s (47.80%)
grpc (*Server) sendResponse 0 of 0.41s (2.08%)
9.32s
grpcx UnaryInterceptorChain func1 1 0 of 9.33s (47.29%)
9.32s
master (*APIServer) unaryInterceptor 0 of 9.32s (47.24%)
8.74s (inline)
log Infof 0 of 8.88s (45.01%)
8.88s
logrus (*Entry) Logf 0 of 8.89s (45.06%)
8.71s
logrus (*Entry) Log 0.03s (0.15%) of 8.71s (44.15%)
0.08s
0.02s
8.60s
2.18s
logrus Entry log 0.01s (0.051%) of 8.60s (43.59%)

cpu off



use fgprof

消减锁粒度

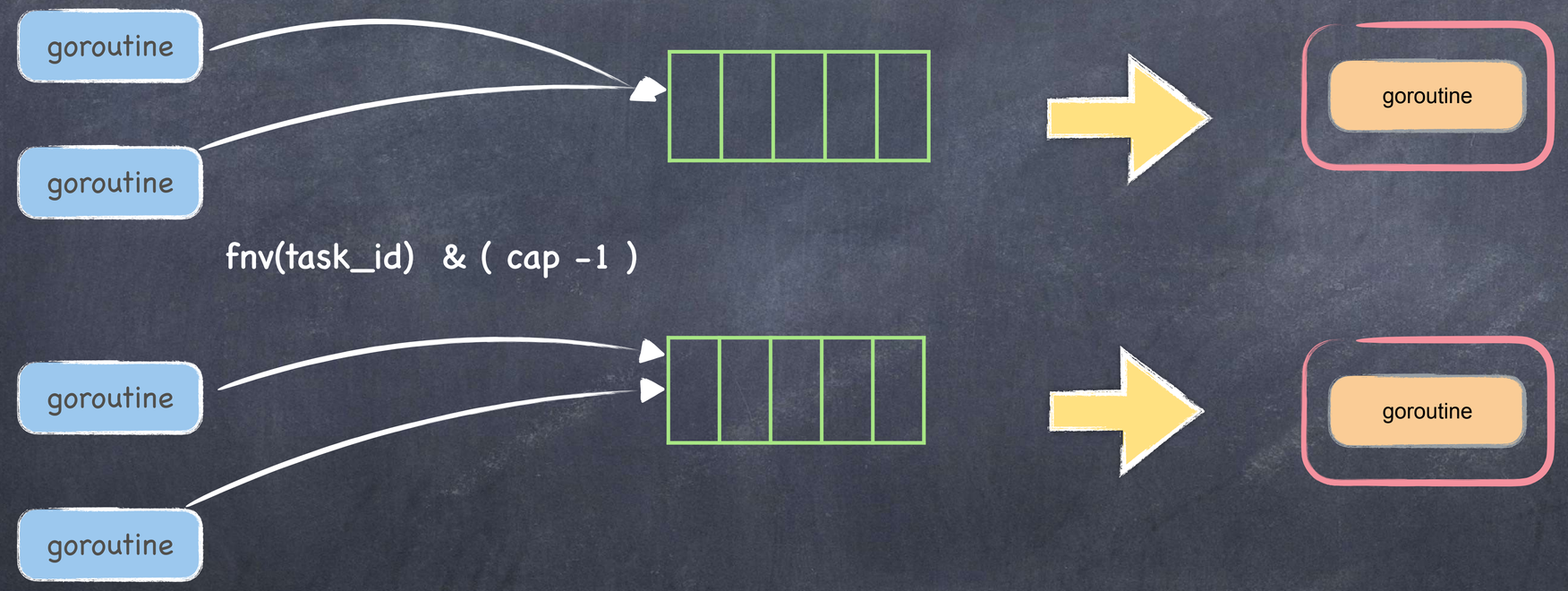


* 减少锁竞争
* 提高并发性

gc optimize

- sync.pool缓存 结果集对象
- sync.pool缓存 bytes.buffer对象
- map及slice采用预分配方案
- ...

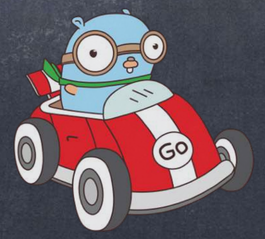
reduce syscall



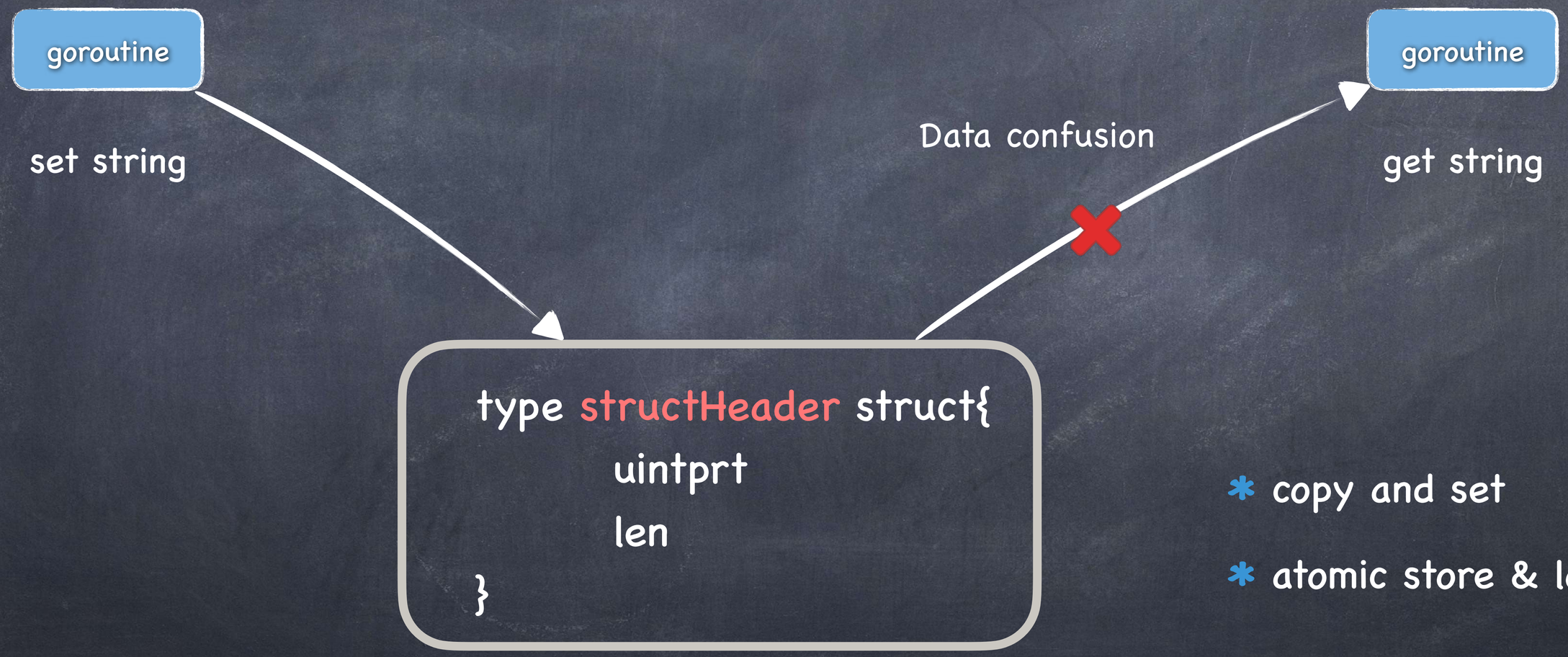
* pipeline
* merge

4

some bug



data race



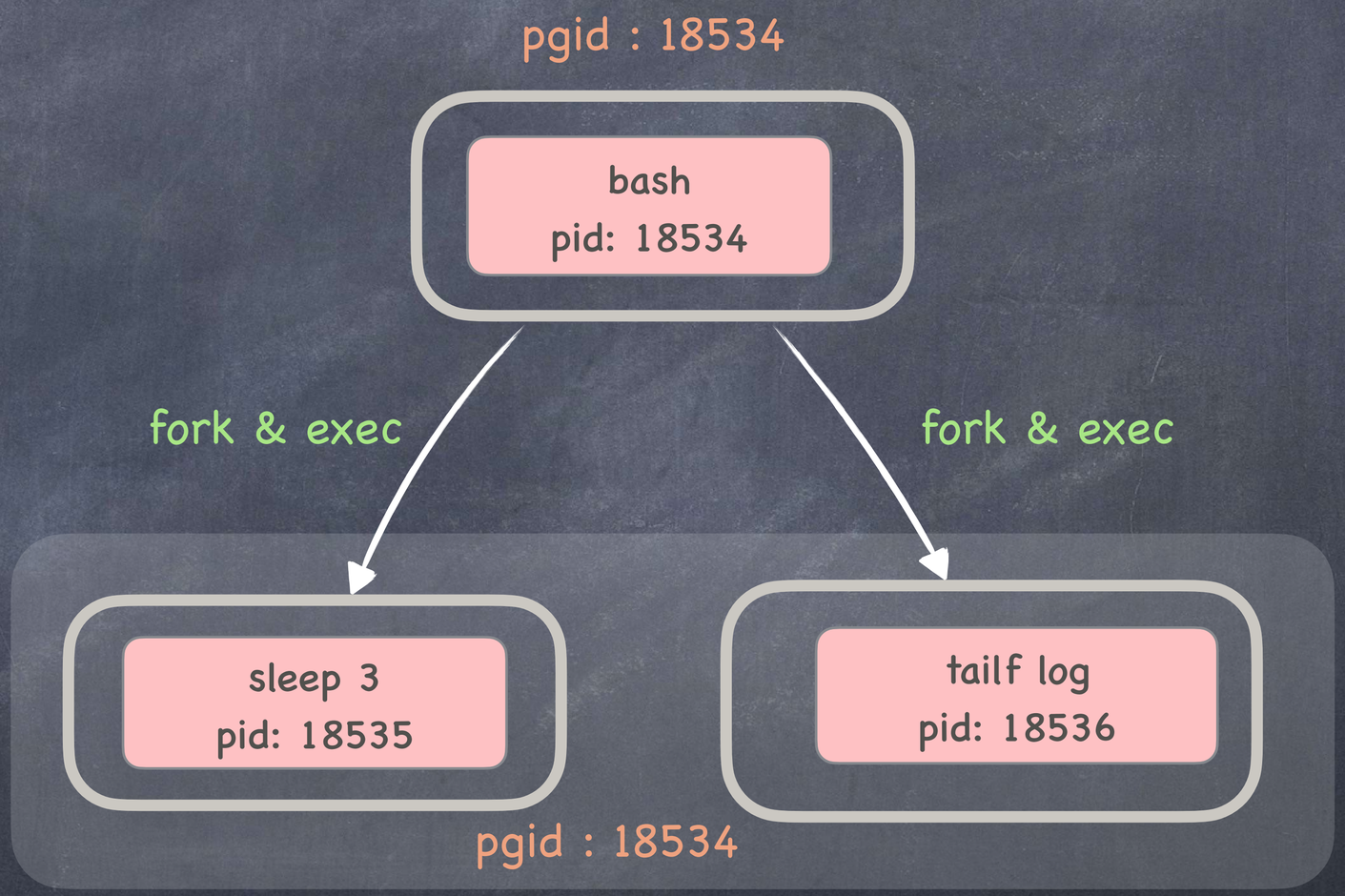
* copy and set
* atomic store & load

shell

* 设置子进程的进程组id
* 杀进程组

golang os/exec

bash -c “sleep 3; tailf log”



shell执行多条命令, kill os/exec pid清理不干净

share

https://github.com/rfyiamcool/go-shell

https://github.com/rfyiamcool/grpcx

https://github.com/rfyiamcool/wmanager

https://github.com/rfyiamcool/node-exporter-plugin

" Q&A "

- xiaorui.cc