Golang 高性能实战

github.com/rfyiamcool

xiaorui.cc

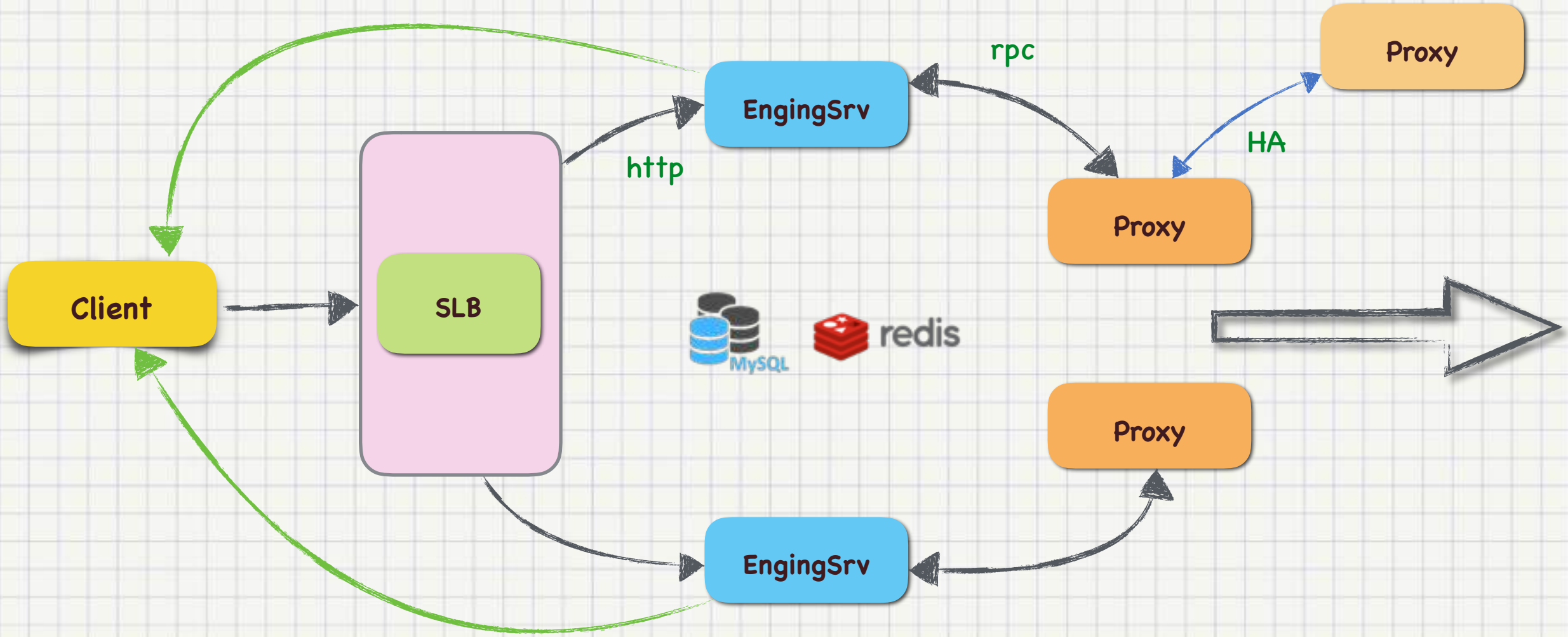
CDN 刷新系统 是一个高性能、可扩展的分布式系统.

支持全网刷新及预缓存业务.

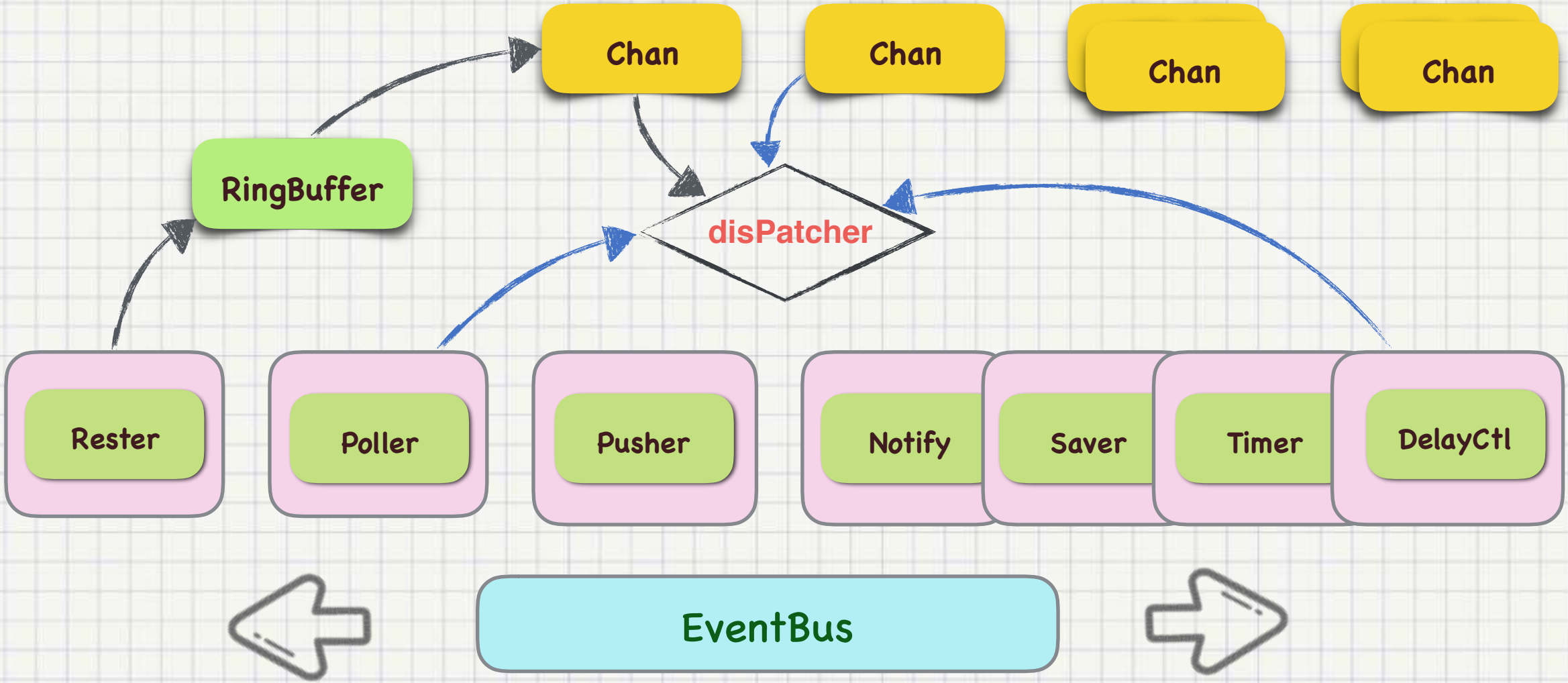
Contents
目录

架构及框架设计

架构设计


Client
SLB
http
EngingSrv
EngingSrv
rpc
Proxy
HA
Proxy
Proxy
MySQL
redis

框架设计


Chan
Chan
Chan
Chan
RingBuffer
disPatcher
Rester
Poller
Pusher
Notify
Saver
Timer
DelayCtl
EventBus

框架设计

- 模块之间解耦
- 复用性强
- 方便监控
- 分布式扩展

- EventBus
  - 控制功能worker池的start / stop

- 读写高的场景
  - 使用ringbuffer避免chan的锁消耗

技术选型

语言: Golang

框架: Gin

压缩: Snappy

序列化: Json-iterator

ORM: Gorm

客户端: imroc/req

日志: Logrus

配置: Toml

缓存: Redis

持久化: Mysql

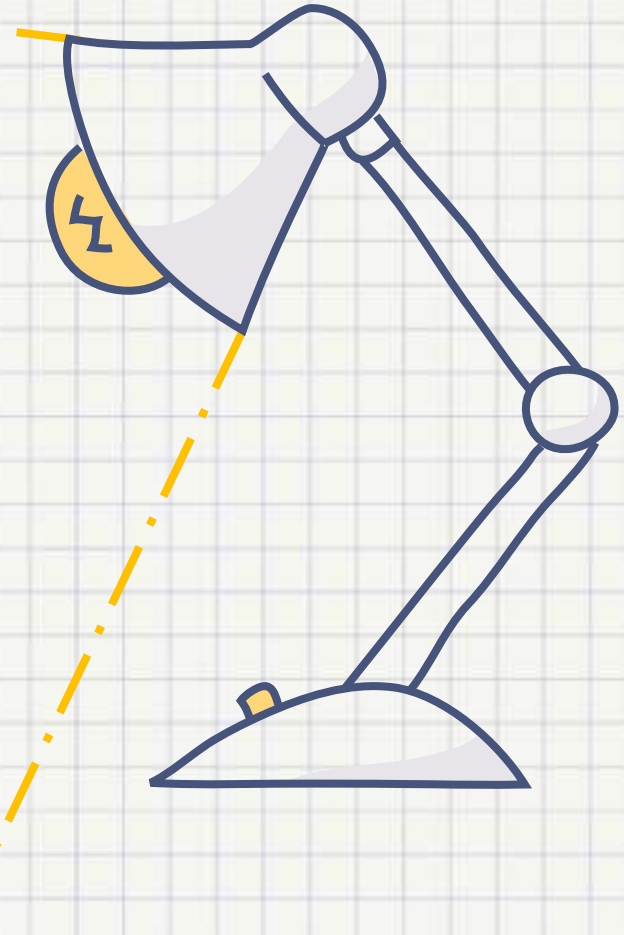
设计调优

基本优化

提前预估size, make 之

临时的map, slice，可采用sync.pool

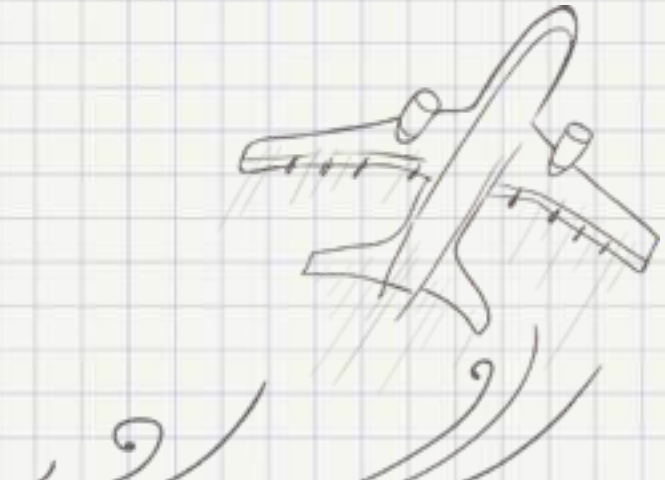
大于32kb，也可使用 sync.pool



基本优化

- 不要滥用time.After，多用事件通知
- 不要滥用goroutine, 减少gc压力
- 不要滥用锁, 引起runtime上下文切换

- 使用unsafe装换类型
- 不要总是[]byte、String转换.



基本优化

- 减少使用reflect反射使用
- 减少不必要的defer调用
- 使用atomic实现无锁逻辑

业务调优

- 减少网络io的消耗
  - 批量接口
  - 缩减网络调用链
  - 使用连接池
- 缓存内存化
  - 减少同步逻辑
  - 使用压缩，节省带宽

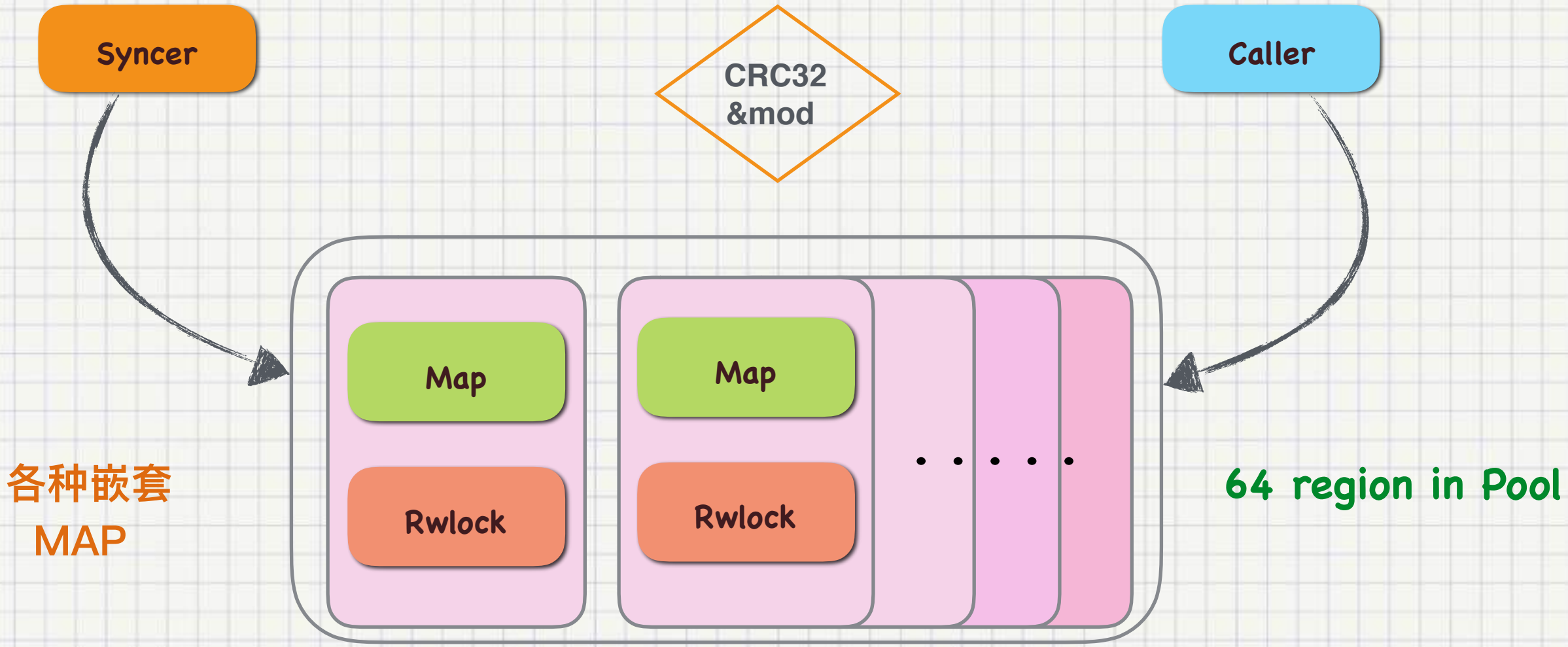
通信协议更换

```
~$ nc -l 8811 -k
POST /v1/agent/task HTTP/1.1
Host: 127.0.0.1:8811
User-Agent: Go-http-client/1.1
Content-Length: 329
Accept: application/json
Accesskey: f0365d34721de99e6f7e9cb5ccda1258
Compress: snappy
Timestamp: 1522114624
Token: 9114b782f14b7428849c78b4a528c5d618f64af8
Accept-Encoding: gzip

{"task_id":"1522114624","proxy_ipd27.0.0.1","due_timestamp":51,
sub=0   <22511092","activ)":0,"sco null},z.jpg"  z  ]},"is_delay":false}
```

HTTP vs RPC

```
nc -l 8811 -k
post_task access_key f0365d34721de99 0cb5ccda1258 callback_api 127.0.0.1 token 9114b782f14b7428849
5d618f64af8 task_type refresh proxy_ip ip task_id 1522114624 hosts_tasks cacheip url http:// com/up
/24/15/kk.webp scope storage_zjnb c01.i01 force subtask_id 10000000022511092 due_timestamp
```

全局任务状态

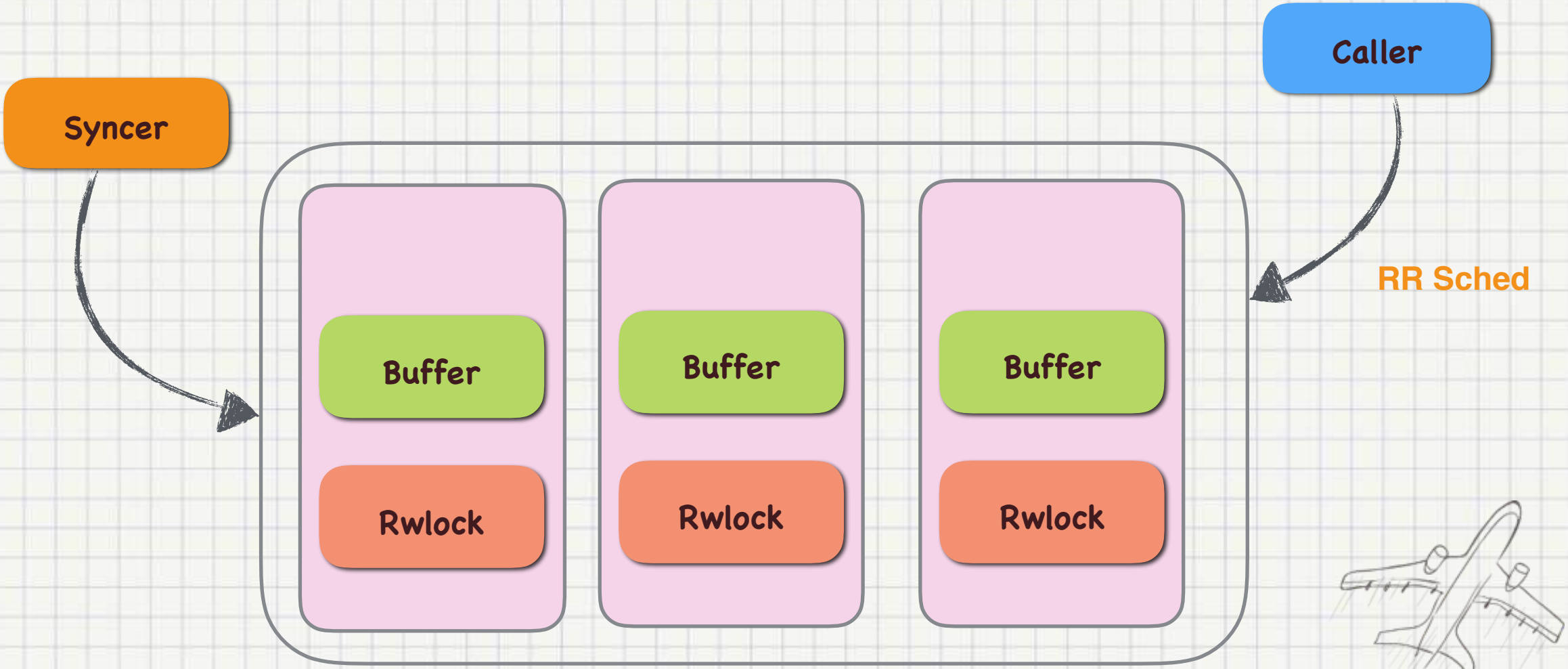

Syncer
CRC32
&mod
Caller
Map
Rwlock
Map
Rwlock
各种嵌套
MAP
64 region in Pool

全局任务状态-code

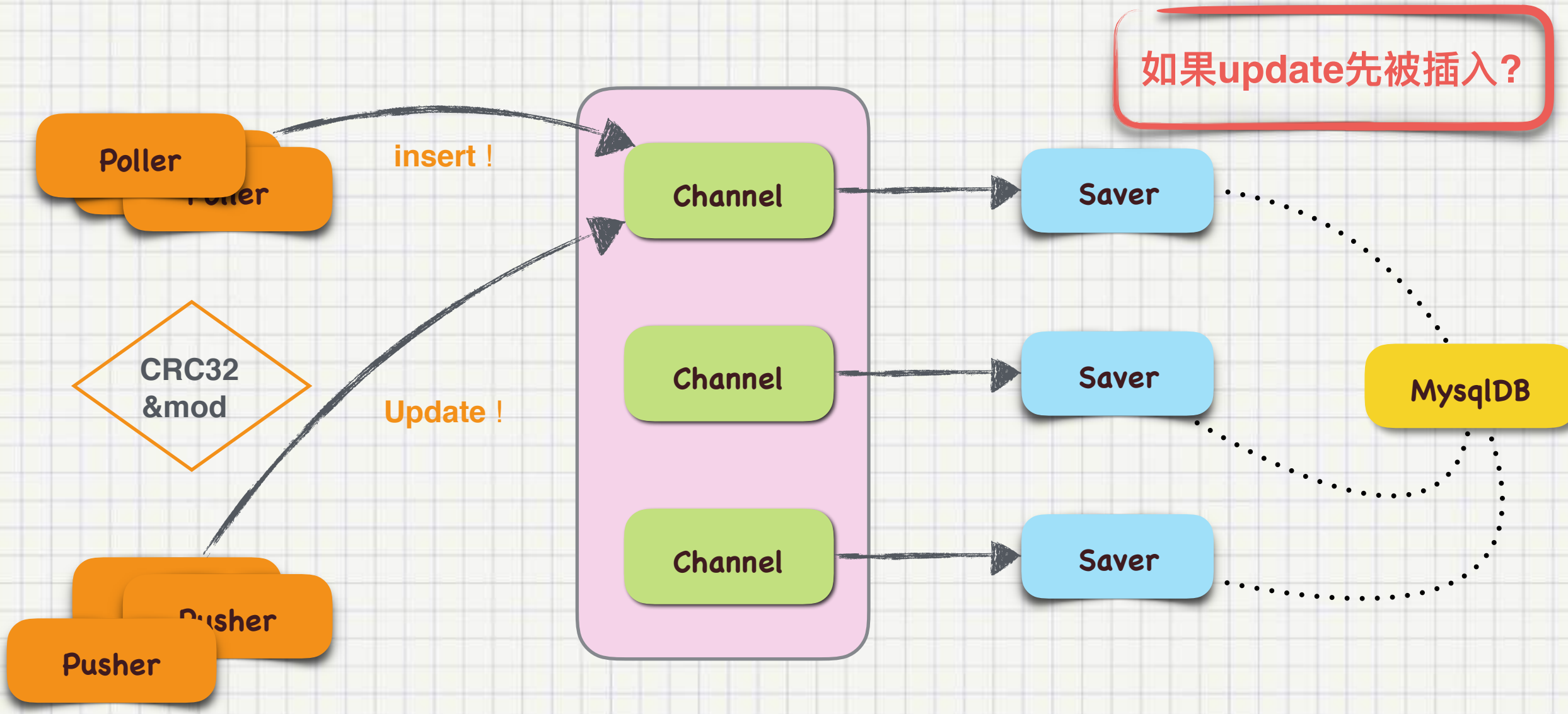

```go
var (
    TaskMapRegionPool = [PreAllocRegion]*TaskMapRegion{}
)

type TaskMapRegion struct {
    map[string]*DetailTaskStats
    *sync.RWMutex
}

type DetailTaskStats struct {
    TaskId       string
    TaskType     string
    WorkingLayer string
    Active       bool // 是否还需要执行?
    Counter      int  // 计数器
    Total        int  // 任务总数
    DueTimeStamp time.Time
    CreatedAt    time.Time
    UpdatedAt    time.Time
    Subtasks     map[string]map[string]string
}
```

```go
func GetOneMapRegion(taskId string) *TaskMapRegion {
    idx := utils.HashToInt(taskId) % PreAllocRegion
    return TaskMapRegionPool[idx]
}

func ExistInRegionPool(taskID string) bool {
    reg := GetOneMapRegion(taskID)
    reg.AcquireR()
    defer reg.ReleaseR()

    _, ok := reg.HasTaskID(taskID)
    return ok
}
```

缓存多副本


Caller
Syncer
RR Sched
Buffer
Rwlock
Buffer
Rwlock
Buffer
Rwlock

操作顺序排队

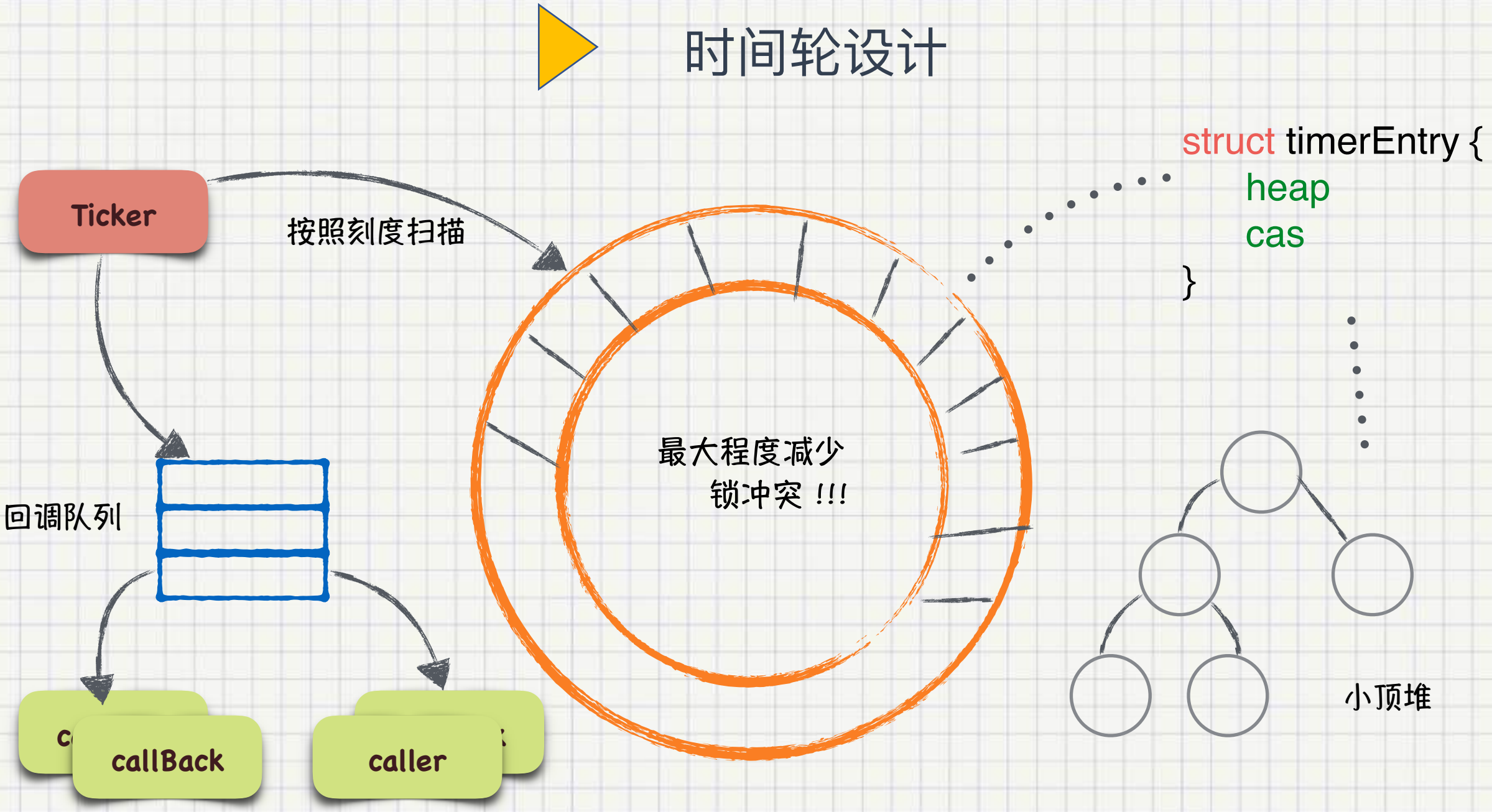

如果update先被插入？
Poller
insert！
Channel
Saver
CRC32
&mod
Update！
Channel
Saver
MysqlDB
Channel
Saver
Pusher

操作顺序排队-CODE

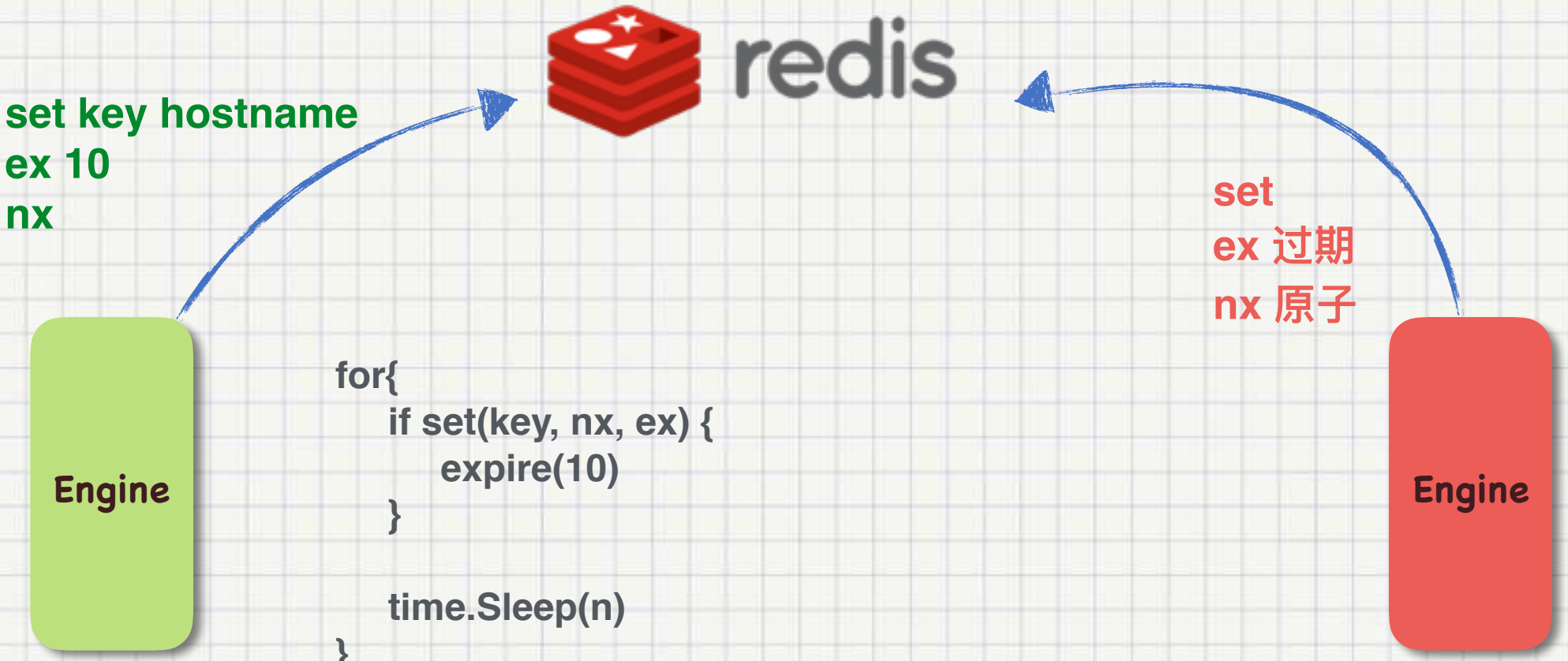
```go
var (
    preSaveNum    int
    saveQueuePool []chan *saveAction
)

func getOneSaveQueue(taskId string) chan *saveAction {
    idx := utils.HashToInt(taskId) % preSaveNum // range 0 <-> (preSaveNum - 1)
    return saveQueuePool[idx]
}

func pushOneSaveQueue(taskId string, data *saveAction) bool {
    queue := getOneSaveQueue(taskId)
    select {
    case queue <- data:
        return true
    default:
        // log
        time.Sleep(1 * time.Second)
        return false
    }
}

func GetSaverLength() int {
    c := 0
    for _, queue := range saveQueuePool {
        c += len(queue)
    }
    return c
}
```

```go
// 消费队列
func (s *saveHandler) popQueue() (*saveAction, bool) {
    var outDto *saveAction
    var ok bool

    select {
    case outDto = <-saveQueuePool[s.slot]:
        ok = true
    case <-EngineWaitGroup.ExitQ:
        return outDto, false
    }

    return outDto, ok
}

// 分段的槽位需要跟协程数对应
func initSaveQueuePool(size int) {
    var defOneSize = 100

    preSaveNum = conf.Gconf.Engine.SaverWorkerNum
    saveQueuePool = make([]chan *saveAction, preSaveNum, preSaveNum)

    oneSize := size / preSaveNum
    if oneSize < defOneSize {
        oneSize = defOneSize
    }
```

时间轮设计

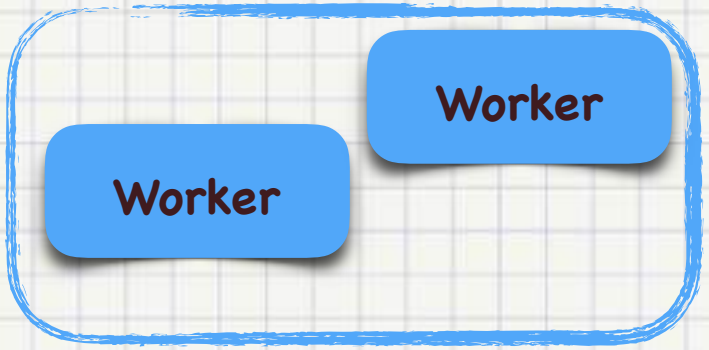

Ticker
按照刻度扫描
回调队列
callBack
caller
最大程度减少
锁冲突 !!!
struct timerEntry {
heap
cas
}
小顶堆

分布式选主


redis
set key hostname
ex 10
nx
set
ex 过期
nx 原子
Engine
Engine
for{
if set(key, nx, ex) {
expire(10)
}
time.Sleep(n)
}

区分快慢队列

Fast Queue



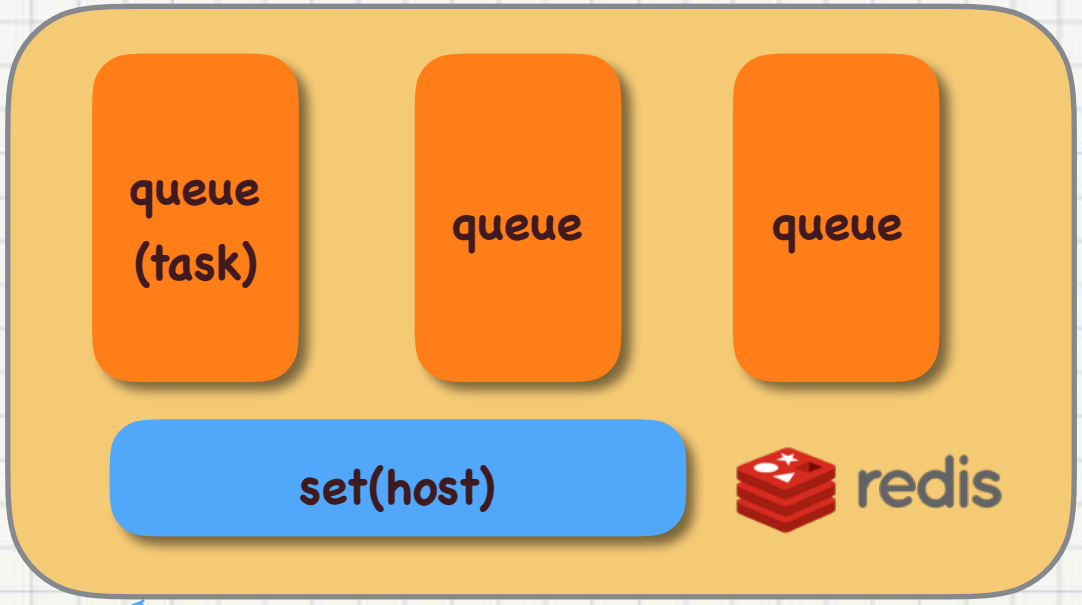
- 避免单一队列因慢任务阻塞
- 排队执行慢任务，减少后端压力

提高吞吐！

Slow Queue

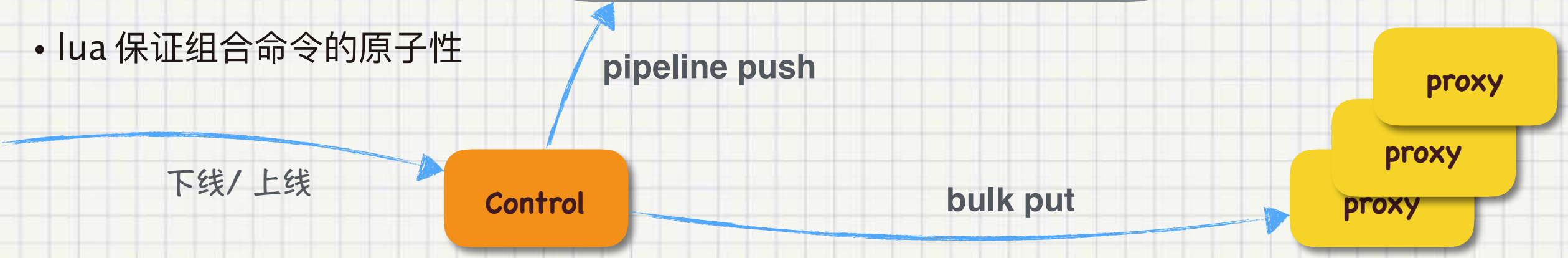
```
SCRIPT_POP = `
local host = KEYS[1]
local queue = KEYS[2]
local delay_set = KEYS[3]
local res = redis.call("lpop", queue)
if res == ""
then
    local v = redis.call("llen", queue)
    if v == 0
    then
        redis.call("SPOP", delay_set, host)
    end
    return res
end
return res
`
```

设备下线延迟队列



吞吐提升
30%

- redis pipeline 减少RTT消耗
- lua 保证组合命令的原子性



排查死锁技巧

Caller

Acquie()

Caller

Release()

输出当前所有协程的Stack

对比锁逻辑

OR

方便调试, 抽象锁方法

加入 代码行 及 随机id

acquie加入, release删除.

单一的就是死锁了

排查OOM

追踪内存泄露点 OR 增长点

fatal error: runtime: out of memory

runtime stack:

runtime.throw(0x665297, 0x16)

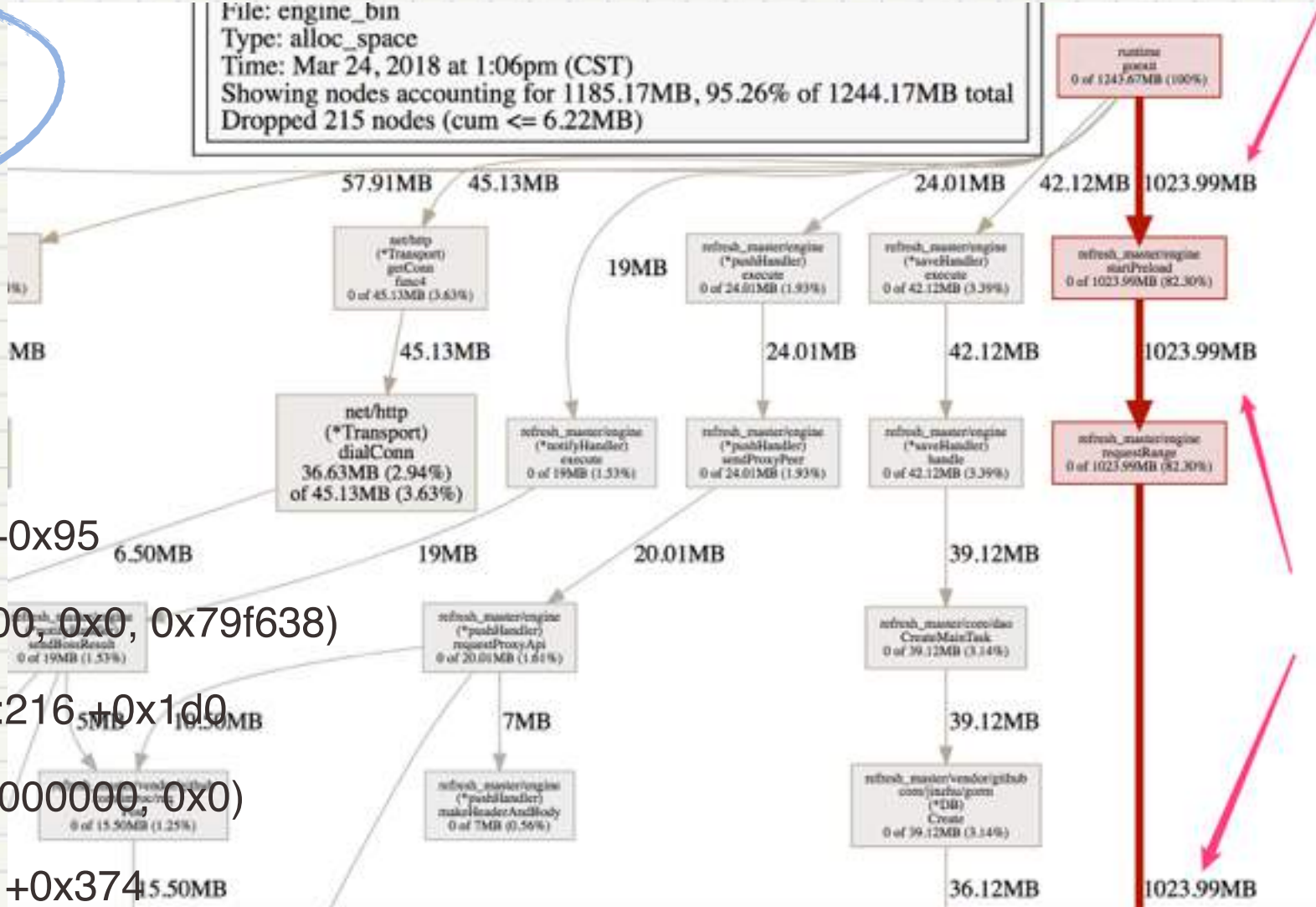
/usr/local/go/src/runtime/panic.go:596 +0x95

runtime.sysMap(0xc4a0200000, 0x80000000, 0x0, 0x79f638)

/usr/local/go/src/runtime/mem_linux.go:216 +0x1d0

runtime.(*mheap).sysAlloc(0x786b20, 0x80000000, 0x0)

/usr/local/go/src/runtime/malloc.go:440 +0x374



开放stats状态接口

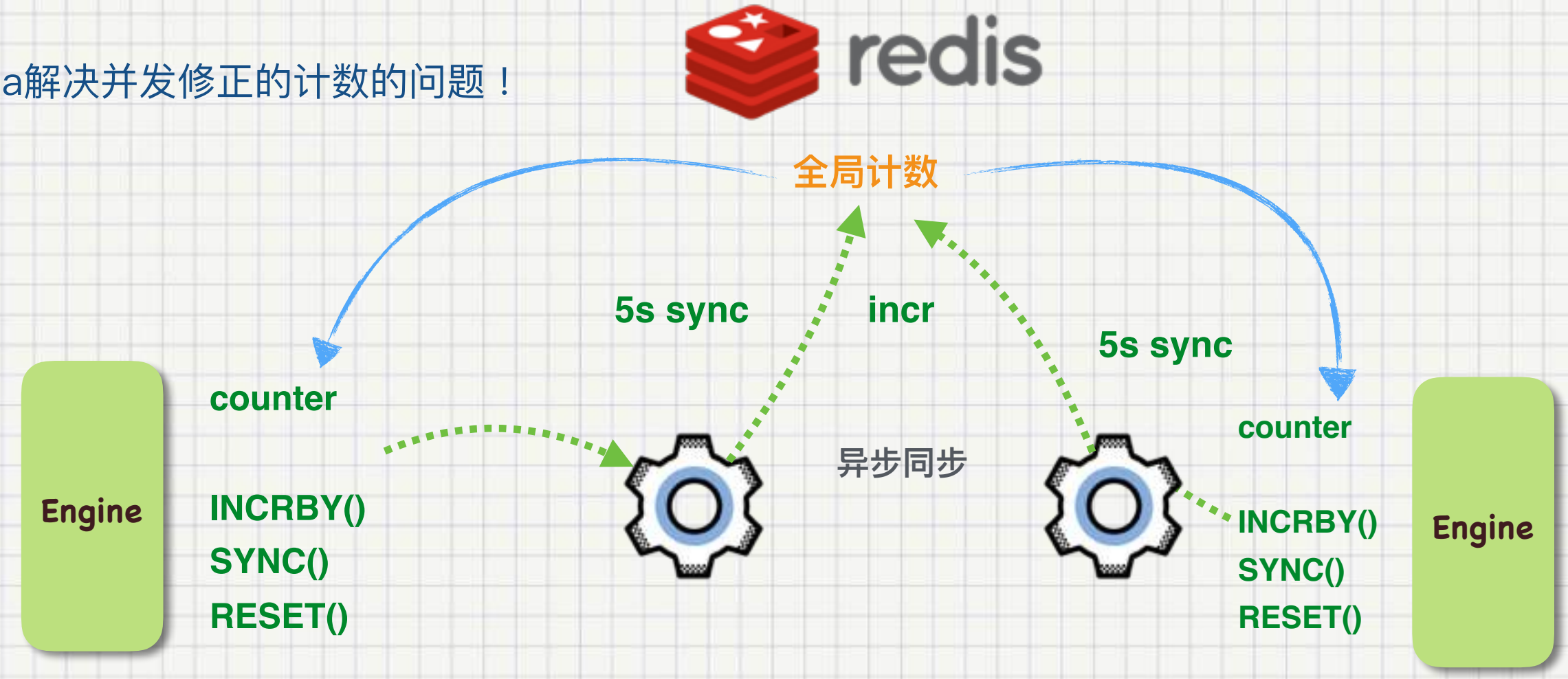

```
{
    "alert_queue": 30,
    "input_task_queue": 400,
    "notify_queue": 1092,
    "pushing_queue": 311,
    "reporting_queue": 612,
    "saver_queue": 3197,
    "tasks_in_cache": 13189,
    "timer_tasks": 13189,
    "timestamp": 1521682815
```



```
    "qps_proxy": {
        "15m": 9031,
        "1m": 10221,
        "5m": 11005,
        "avg": 91110,
        "count": 897811
    },
    "qps_boss": {
        "15m": 5113,
        "1m": 5200,
        "5m": 5300,
        "avg": 5001,
        "count": 297811
    },
}
```

加入分布式限频

lua解决并发修正的计数的问题！



深度排雷

SystemTool

- perf top
- lsof、netstat、sar
- go tool pprof
- strace
- iostat、vmstat、top
- wrk

Established 暴增问题

```
task_disp 14138 root  130u  IPv4 739219083      0t0  TCP 10.3[redacted]:17170->[redacted] (ESTABLISHED)
task_disp 14138 root  132u  IPv4 739219051      0t0  TCP 10.3[redacted]:17106->[redacted] (ESTABLISHED)
task_disp 14138 root  136u  IPv4 739219055      0t0  TCP 10.3[redacted]:17114->[redacted] (ESTABLISHED)
task_disp 14138 root  137u  IPv4 739219056      0t0  TCP 10.3[redacted]:17116->[redacted] (ESTABLISHED)
task_disp 14138 root  138u  IPv4 739219057      0t0  TCP 10.3[redacted]:17118->[redacted] (ESTABLISHED)
task_disp 14138 root  139u  IPv4 739219058      0t0  TCP 10.3[redacted]:17120->[redacted] (ESTABLISHED)
task_disp 14138 root  140u  IPv4 739219059      0t0  TCP 10.3[redacted]:17122->[redacted] (ESTABLISHED)
task_disp 14138 root  141u  IPv4 739219060      0t0  TCP 10.3[redacted]:17124->[redacted] (ESTABLISHED)
```

```
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

```
TIME_WAIT 52
CLOSE_WAIT 5011
FIN_WAIT2 12
ESTABLISHED 19303
LAST_ACK 25
```

Established 暴增问题

net/http 连接池的维度是 http.Transport.

问题代码出在， 不断的实例化 http.Transport

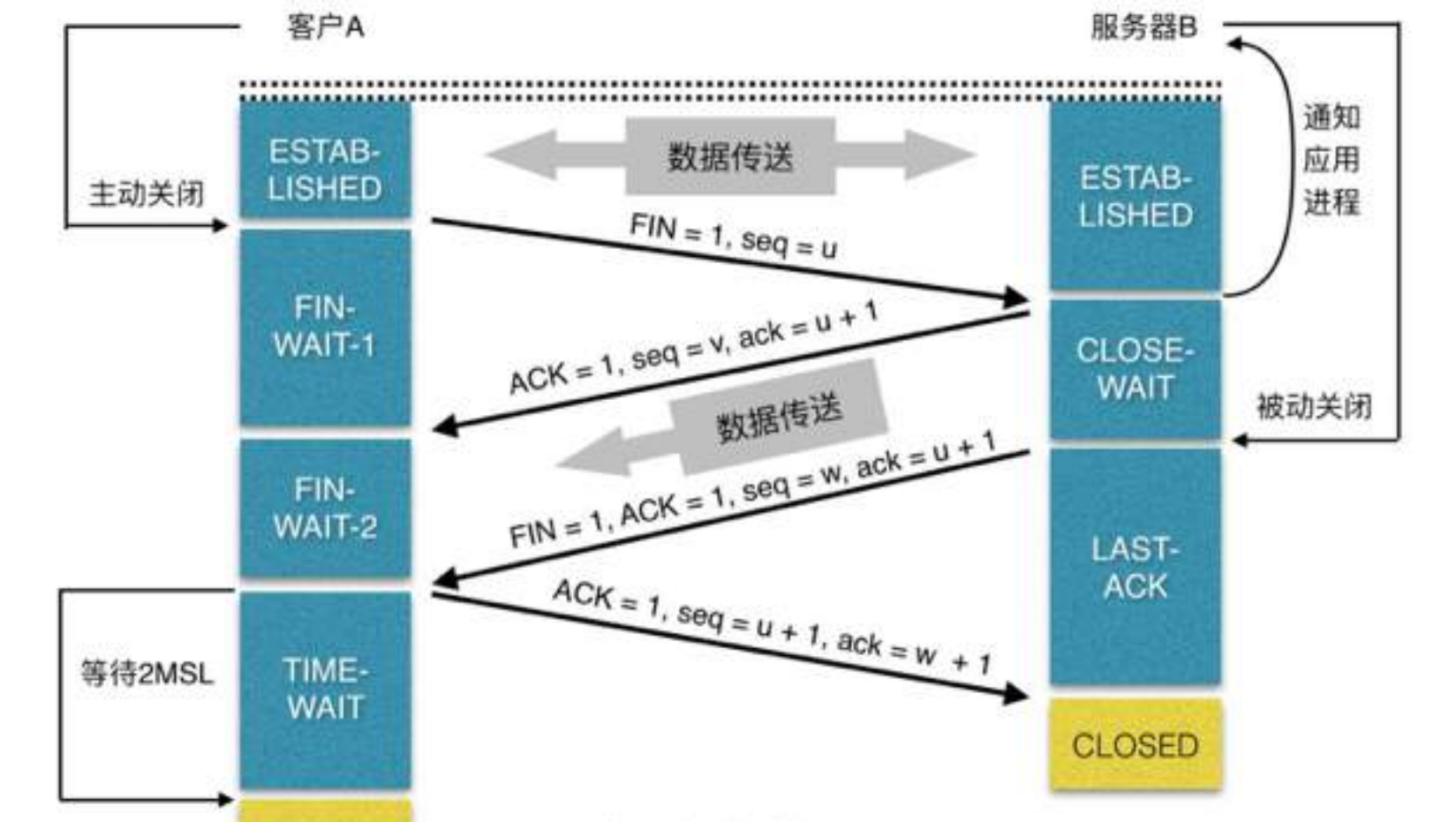
连接池引起的TimeWait暴增

TimeWait 暴增到 3.5W ...

TimeWait 暴增到 5W ...

TimeWait 暴增到 3W ...

TimeWait 暴增到 4W ...



连接池引起的TimeWait暴增

TimeWait 暴增到 3.5W ...

TimeWait 暴增到 5W ...

TimeWait 暴增到 3W ...

TimeWait 暴增到 4W ...

```
http.DefaultTransport.(*http.Transport).MaxIdleConnsPerHost = 200

const DefaultMaxIdleConnsPerHost = 2
t := &http.Transport{
    ...
    DialContext: (&net.Dialer{
        ....
    }).DialContext,
    MaxIdleConnsPerHost:   200,
    MaxIdleConns:          1000,
    ...
```

系统线程暴涨

```
  wsec/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
113328.00   809.49   145.55  602.14    0.00  602.14   7.14 100.00
    0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
    0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

  wsec/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
111480.00   995.36   132.68 1184.48    0.00 1184.48   8.93 100.00
    0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
    0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

profiles:

0 block
398 goroutine
760 heap
0 mutex
1009 threadcreate

full goroutine stack dump

系统线程暴涨

```
strace -fp pid -s 512
```

```
write(8, ……
mmap(NULL, 10489856, PROT_READ|PROT_...
mprotect(0x7f04d1e6d000, 4096,...
clone(child_stack=0x7f04d286cff0,
...
clone(child_stack=0x7f04d286cff0,
```

- 线程不会被回收
- 每个线程8M内存
- 增加调度消耗
- cpu cache miss
- more

```
lsof -P -p pid -n
```

```
refresh_master 2184 root    8u   REG   ... refresh_master/logs/http.log
```

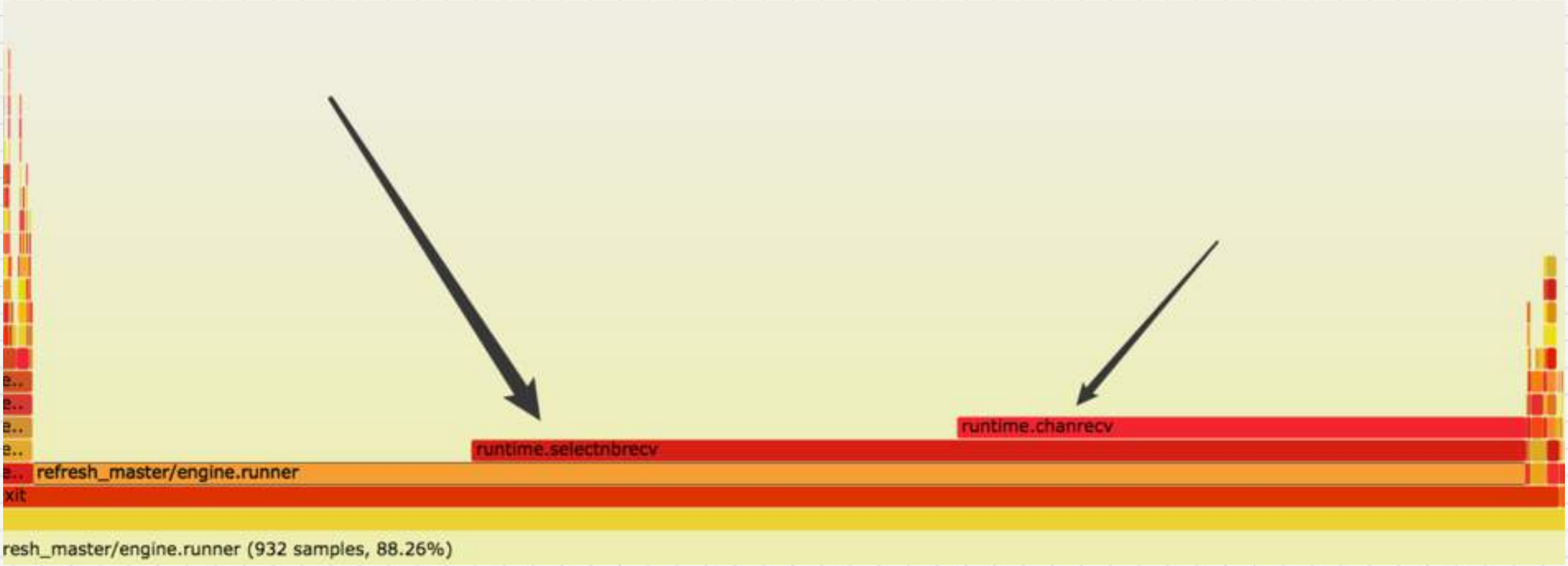
忙轮询

```
...
for {
    select {
        case res = <-dataQueue:
            retrun res
        case <-done:
            return res
        default:
            // do something
    }
    return res
}
...
```

```
Samples: 78K of event 'cpu-clock', Event count (approx.): 121561151512
Overhead  Shared Object       Symbol
  49.09%  k                   [.] runtime.chanrecv
  25.58%  k                   [.] runtime.selectnbrecv
  17.98%  k                   [.] main.runner
   1.01%  [kernel]            [k] finish_task_switch
```

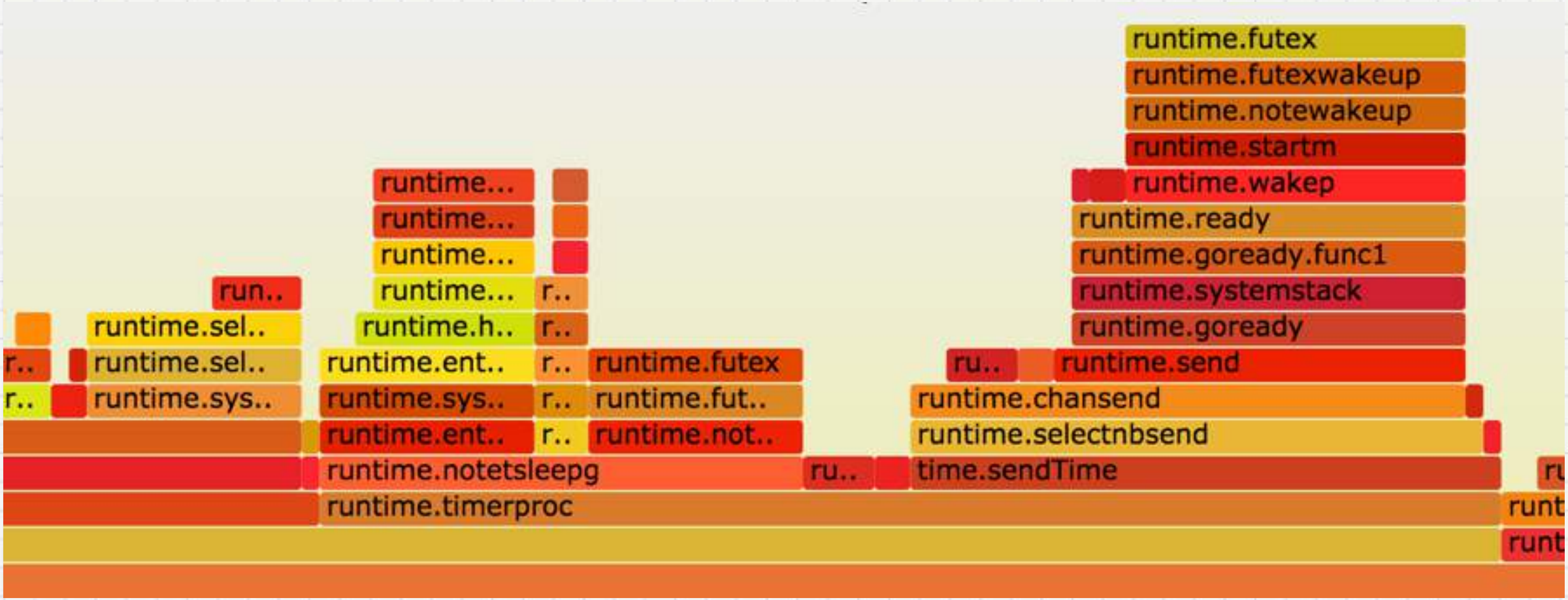
- perf top 可以看出 select chan奇高
- strace 几乎无调用, 单纯cpu指令
- top 直接 100%

忙轮询


runtime.chanrecv
runtime.selectnbrecv
refresh_master/engine.runner
xit
resh_master/engine.runner (932 samples, 88.26%)

定时器 | 断续器

空跑 1000个协程，sleep 100ms的火焰图表现



定时器 | 断续器

futex、pselect6 syscall 出奇的高！

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
26587 root      20   0  126m 8296 2884 S  6.0  0.2   0:00.20 test_select
19649 root      20   0  502m 197m 5164 S  4.3  5.2  27:21.21 agent_bin
   12 root      20   0     0    0    0 S  0.3  0.0  41:01.90 events/1
 1309 root      20   0  584m  11m 1524 S  0.3  0.3  64:44.50 rsyslogd
13931 root      20   0  241m  11m 5032 S  0.3  0.3   0:30.21 atm_master
```

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 58.28   21.095623        3007      7016       898 futex
 40.99   14.837569          40    369096           pselect6
  0.73    0.263777           0    790515           clock_gettime
  0.00    0.000290           0       848           sched_yield
  0.00    0.000000           0         2         1 restart_syscall
```

定时器 | 断续器

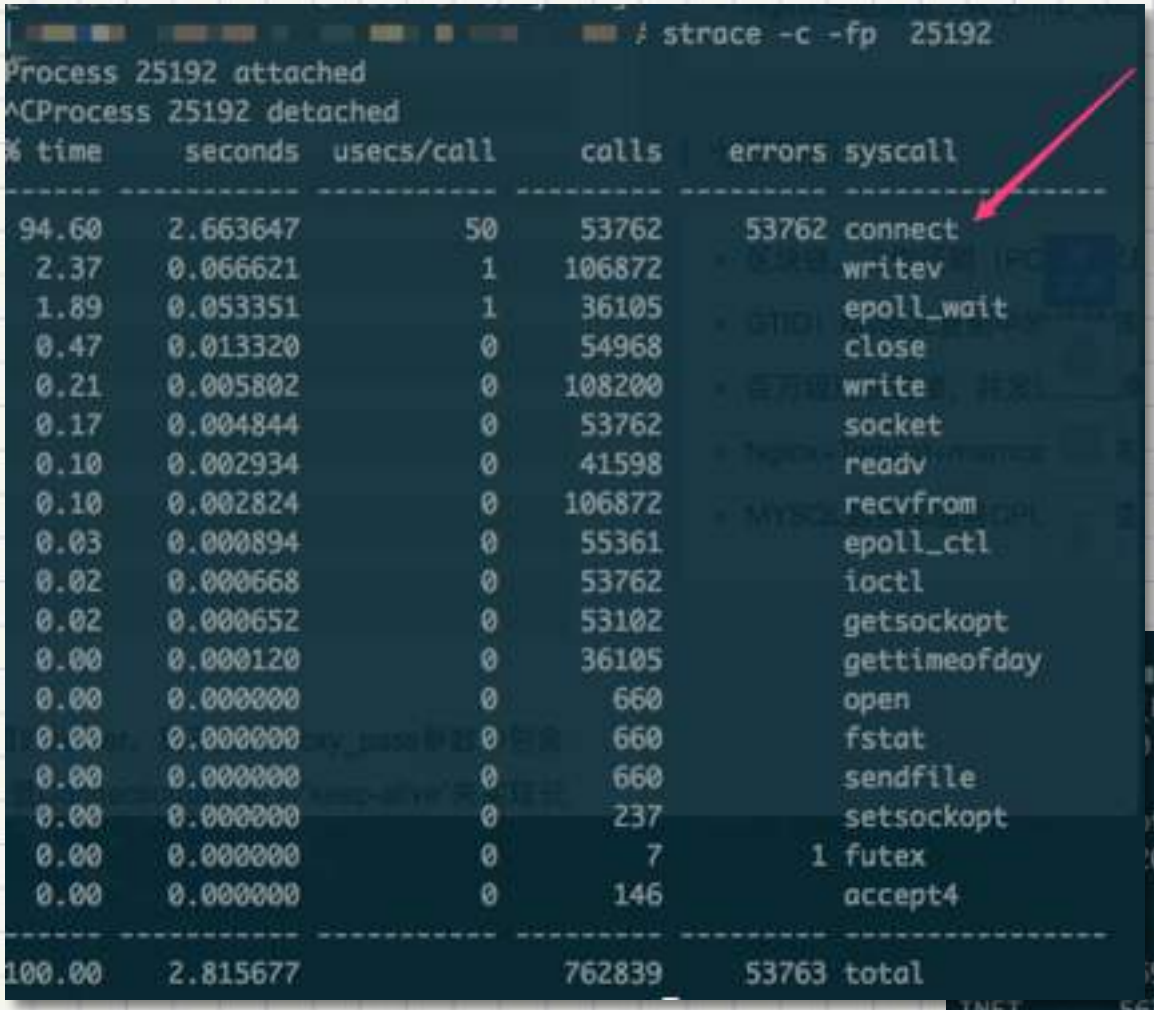
time.After

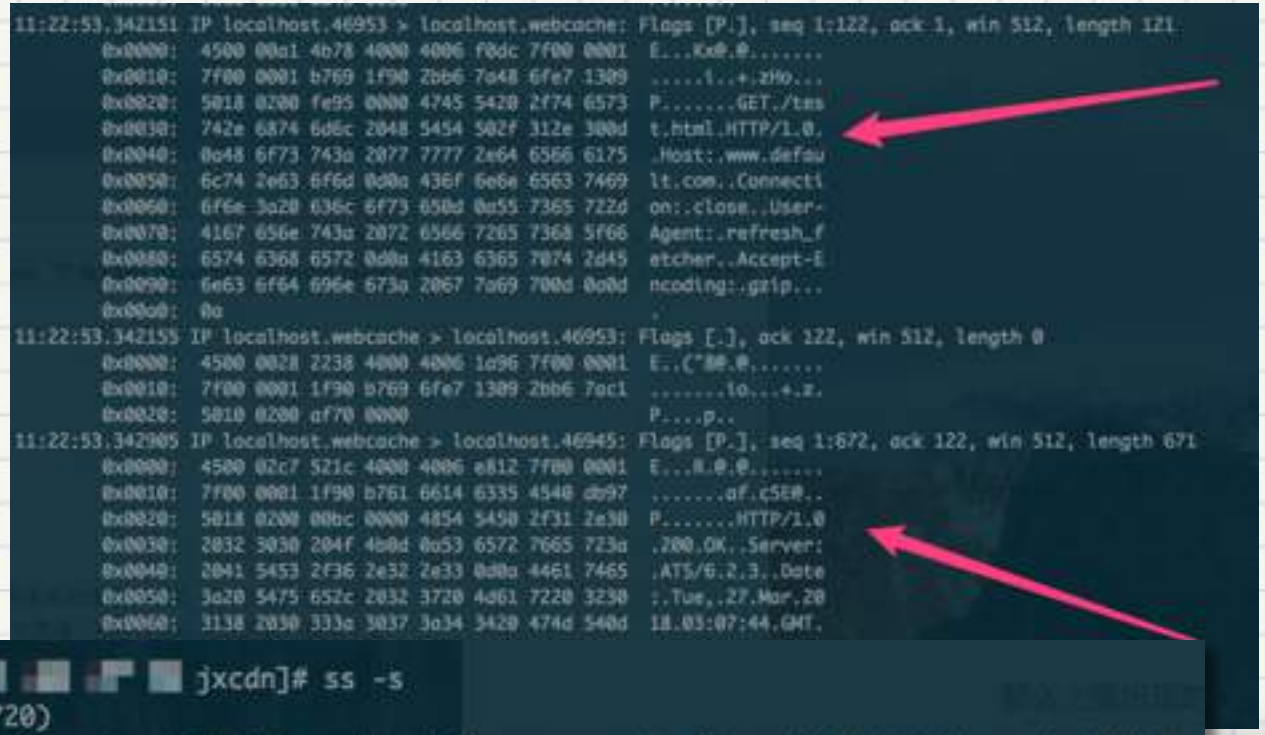
time.NewTicker

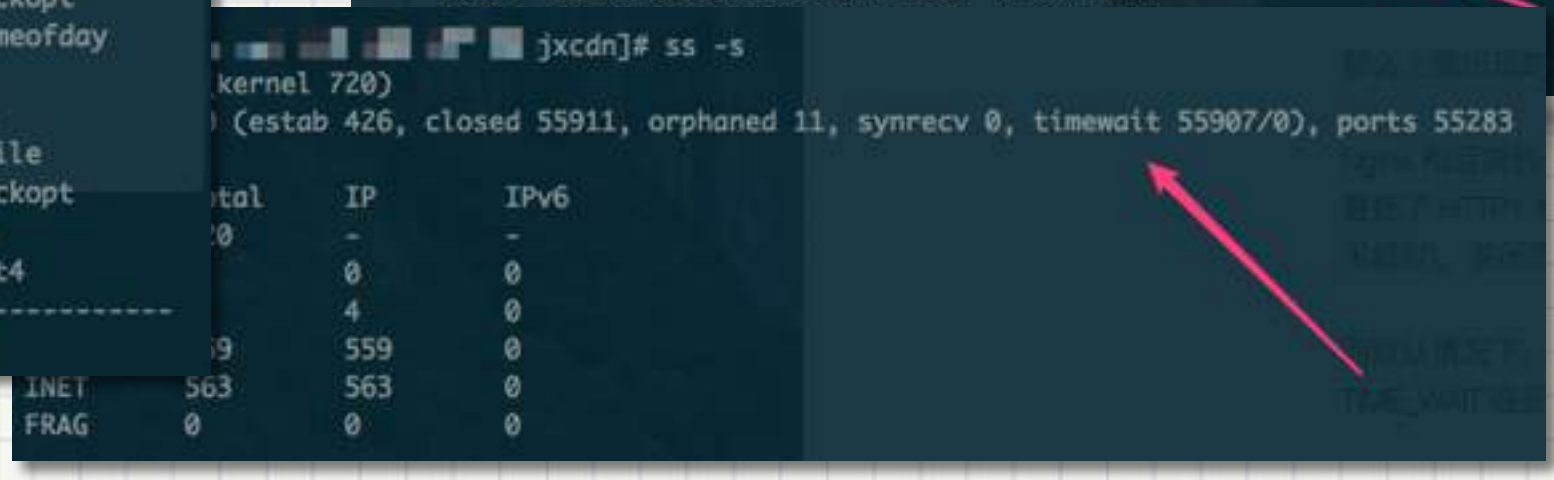
解决方法:

- 自定义定时器
- 减少select的超时逻辑
- sleep 时间周期放大

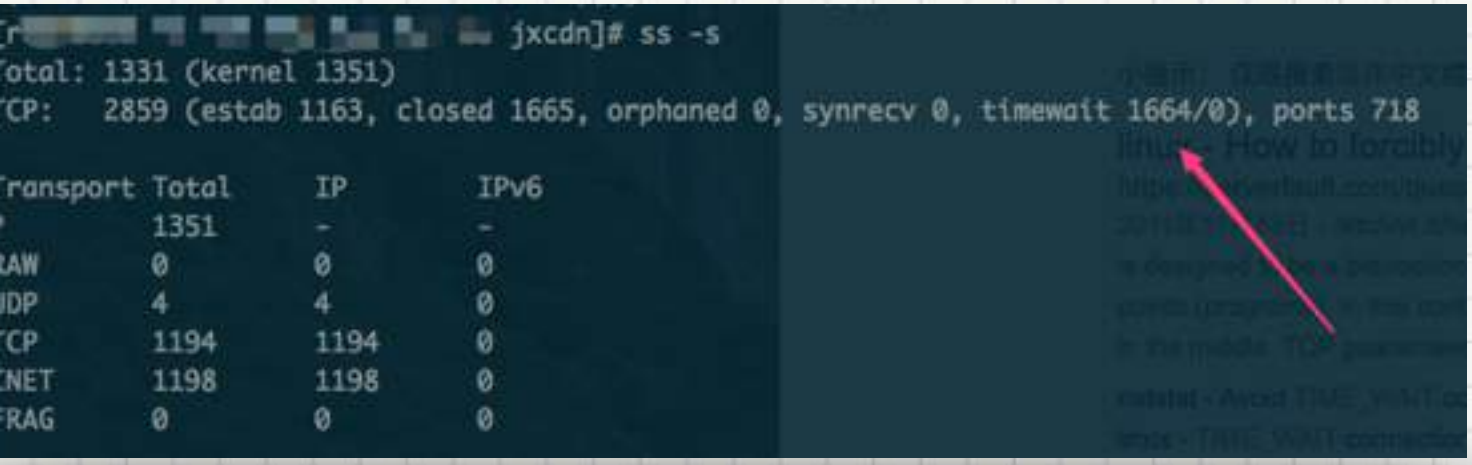
Nginx的time_wait问题

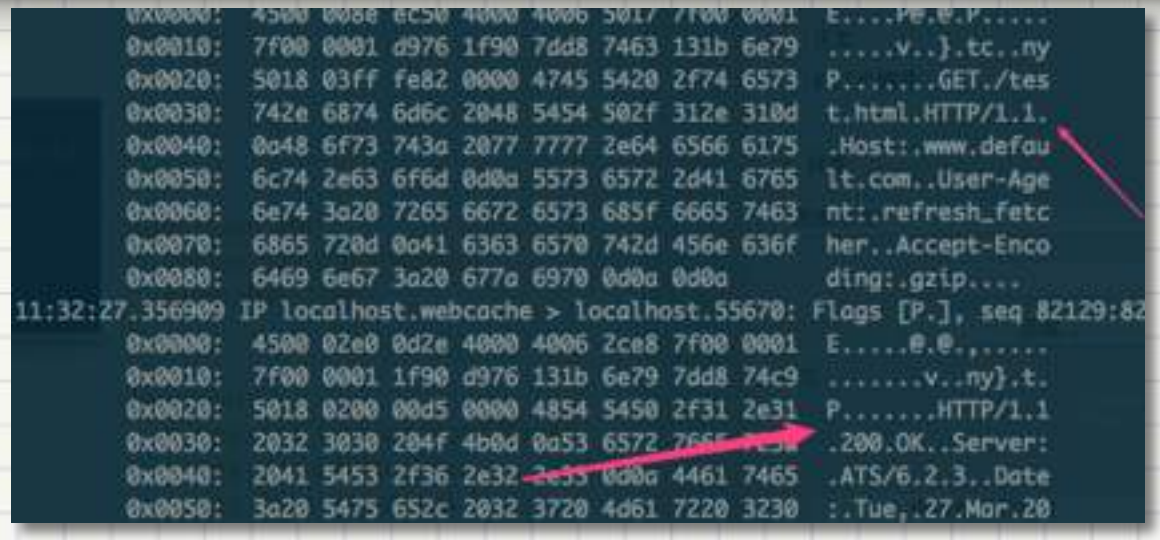






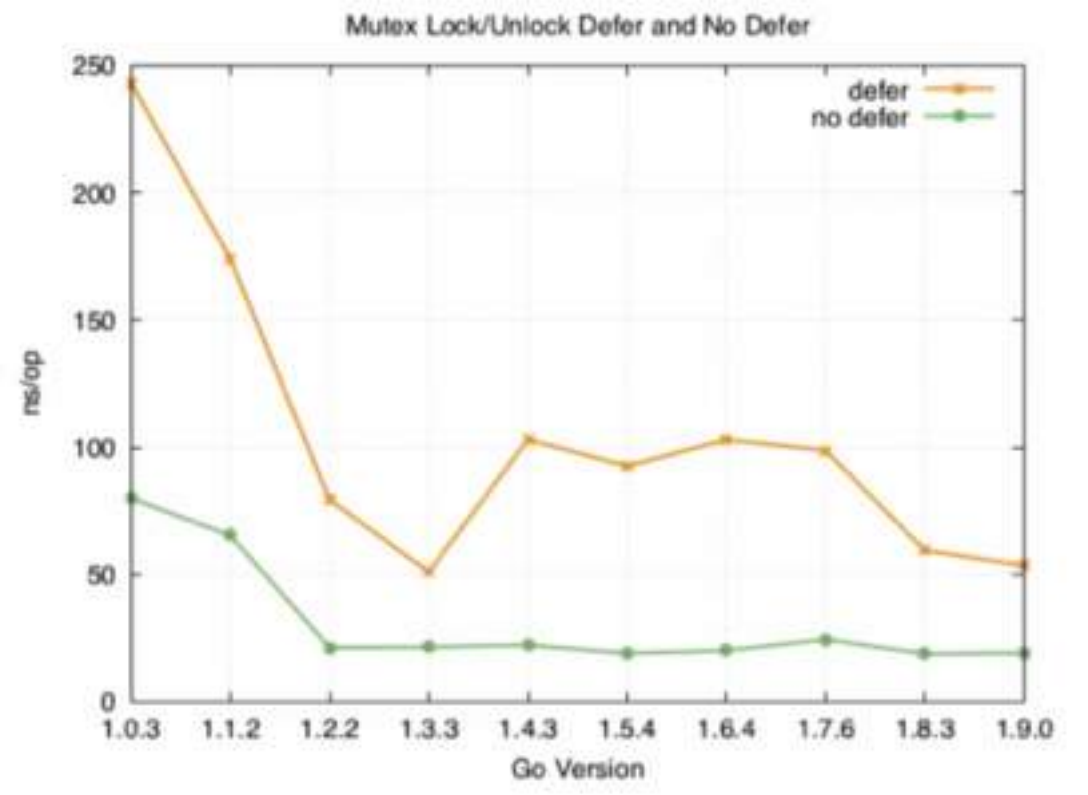
Nginx的time_wait问题






```
upstream ats{
    server 127.0.0.1:8080;
    ...
    keepalive 100;
}

location / {
    proxy_set_header Host $host;
    proxy_pass http://ats;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
```

defer的消耗



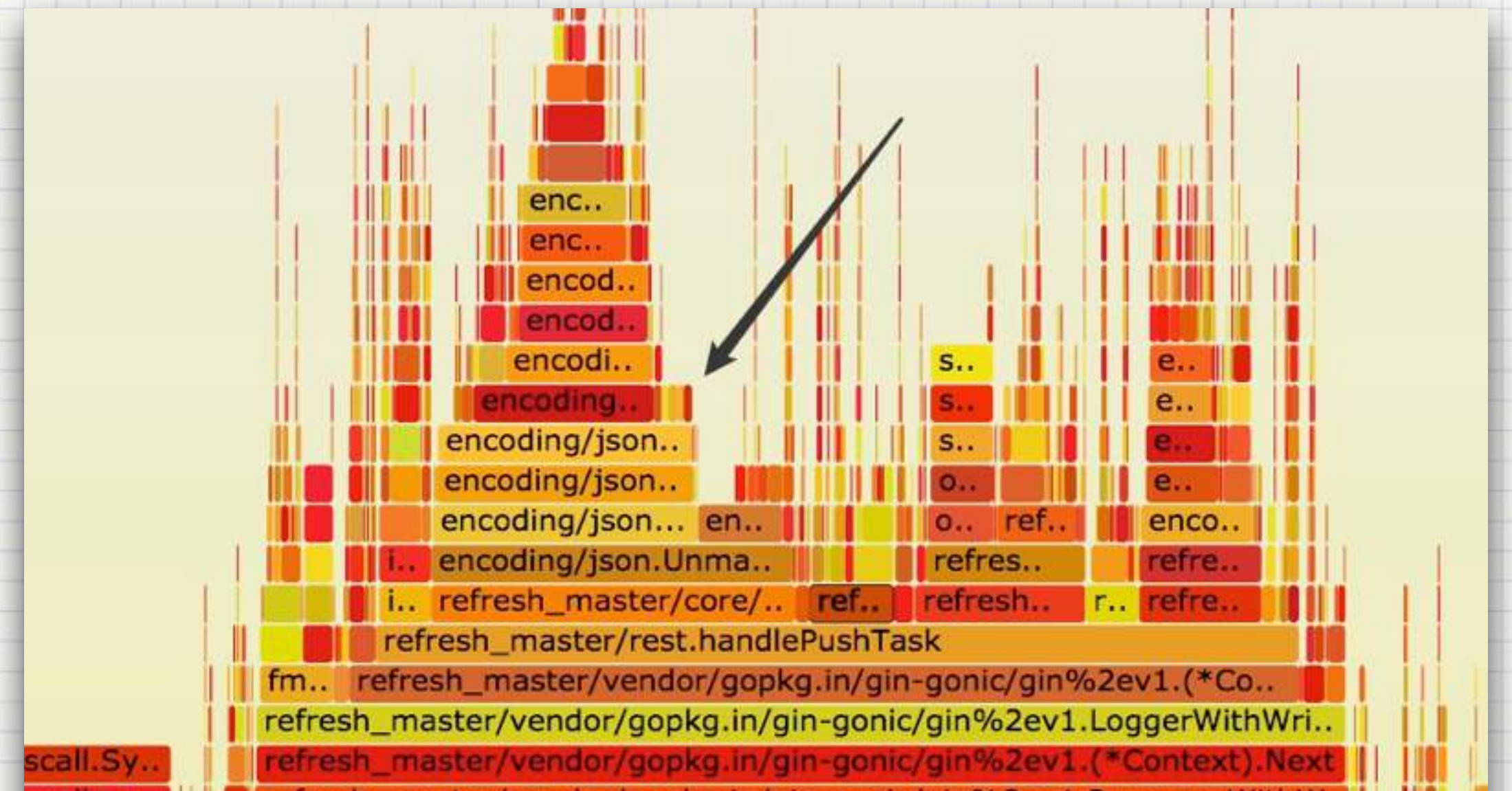
汇编:

call runtime.deferreturn,
add xx SP
return

go 1.10

```
[ruifengyun@xiaorui locker (engine ✗)]$ go test -bench=. -benchmem -run=none
goos: darwin
goarch: amd64
pkg: refresh_master/locker
BenchmarkDeferLock-4            30000000                67.1 ns/op             0 B/op
BenchmarkNoDeferLock-4          100000000               20.9 ns/op             0 B/op
```

Json性能问题


enc..
enc..
encod..
encod..
encodi..
encoding..
encoding/json..
encoding/json..
encoding/json...
encoding/json.Unma..
refresh_master/core/..
refresh_master/rest.handlePushTask
refresh_master/vendor/gopkg.in/gin-gonic/gin%2ev1.(*Co..
refresh_master/vendor/gopkg.in/gin-gonic/gin%2ev1.LoggerWithWri..
refresh_master/vendor/gopkg.in/gin-gonic/gin%2ev1.(*Context).Next
scall.Sy..
s..
s..
s..
o..
o..
ref..
refres..
refresh..
e..
e..
e..
e..
enco..
refre..
refre..
r..
en..
i..
i..
ref..
fm..

Json 性能问题

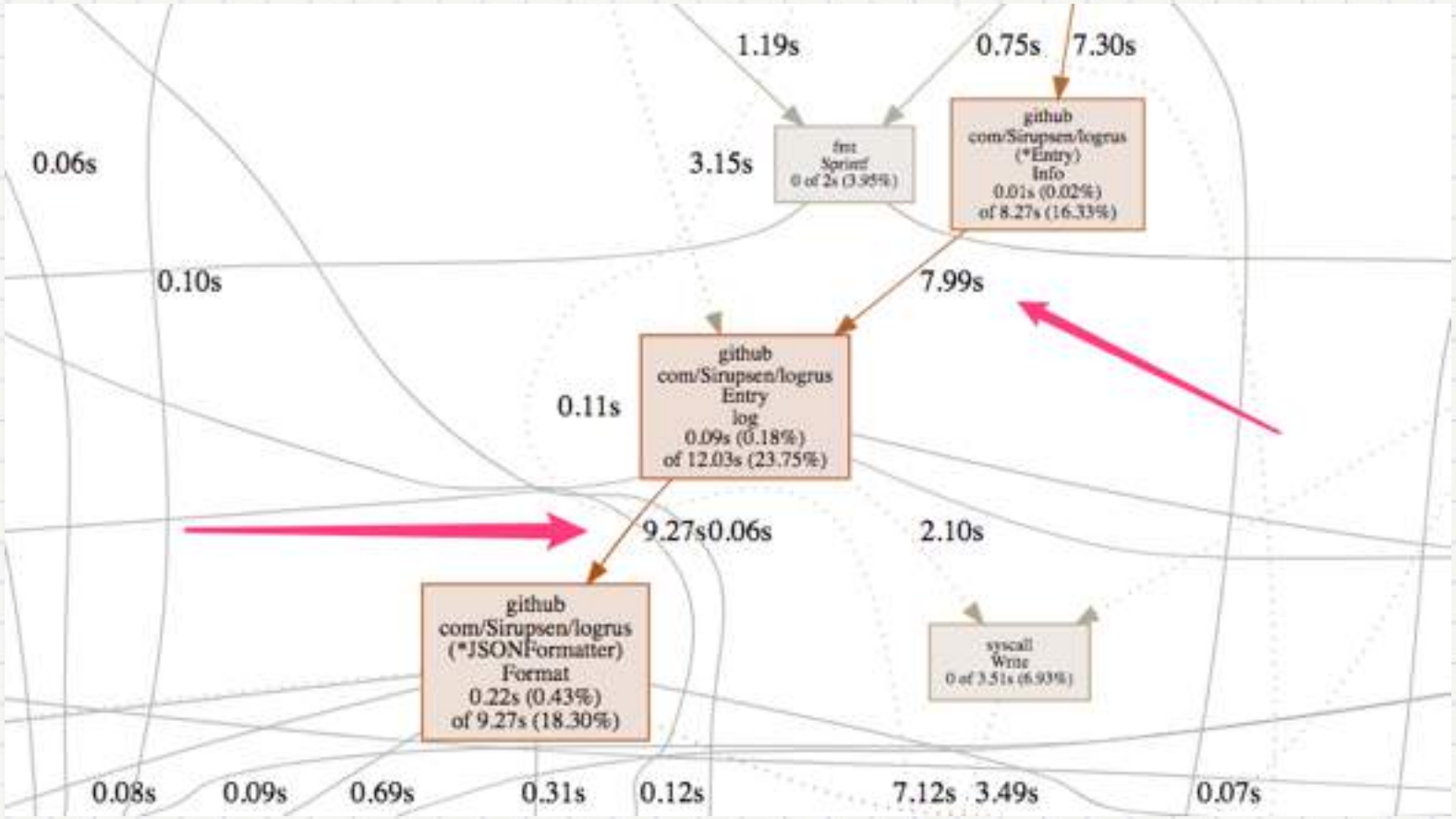
- 固定数据结构
  - encoding/json
- 不固定数据结构
  - buger/jsonparser

go1.10 std > json-iterator

减少反射，减少对象创建 !!!

- 上层代码
  - 直接替换
- 第三方库修改方法
  - 修改Vendor相关库的引用 .
  - 使用Monkey的替换标准库 .

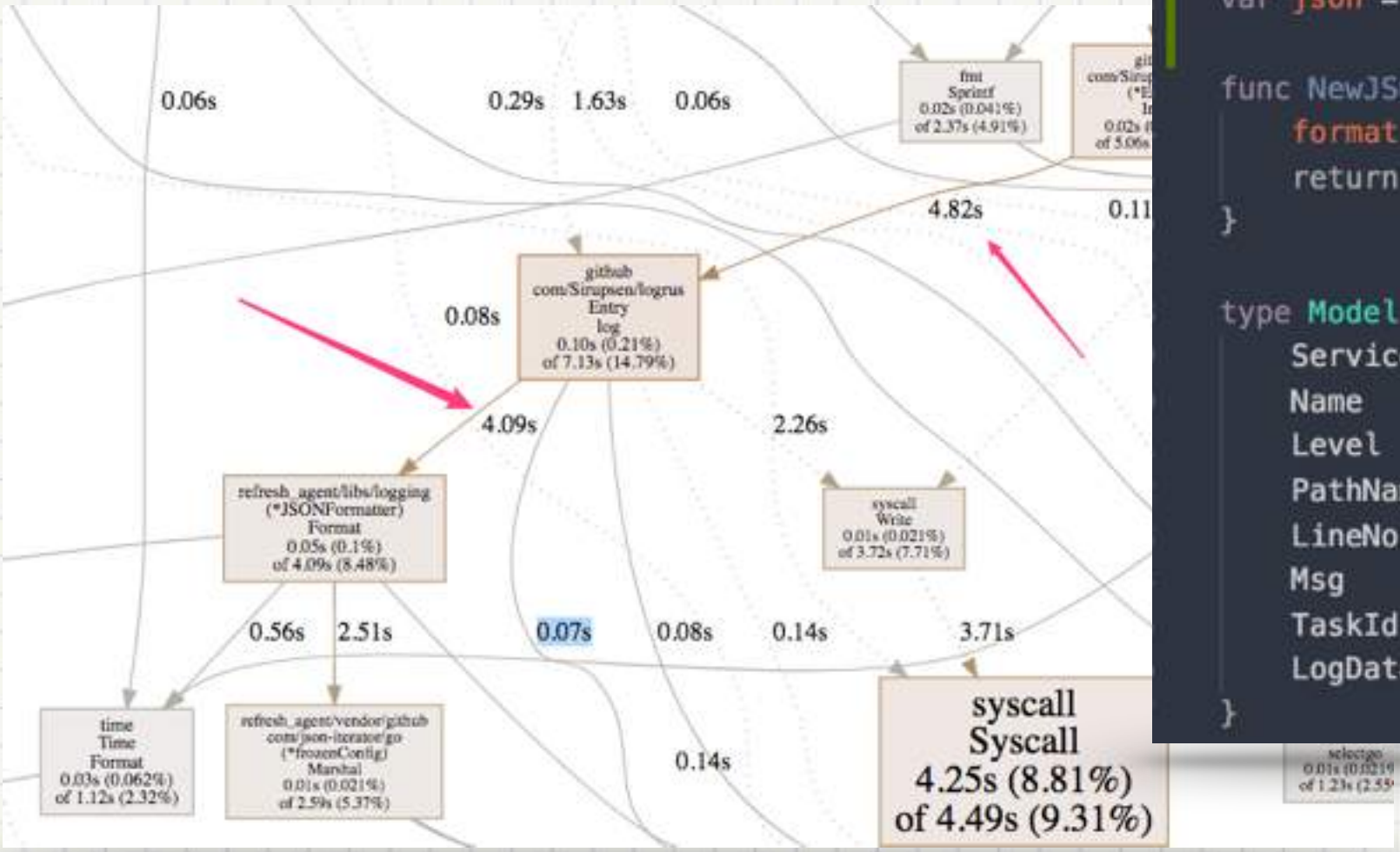
logrus json (map vs struct)



struct > map

cost 9.27s

logrus json (map vs struct)




```go
var json = jsoniter.ConfigCompatibleWithStandardLibrary

func NewJSONFormatter(service string) *JSONFormatter {
    format := JSONFormatter{Service: service}
    return &format
}

type Model struct {
    Service  string `json:"service"`
    Name     string `json:"name"`
    Level    string `json:"level"`
    PathName string `json:"pathname"`
    LineNo   int    `json:"lineno"`
    Msg      string `json:"msg"`
    TaskId   string `json:"task_id"`
    LogData  string `json:"log_date"`
}
```

cost 4.09s

GC 耗时调优

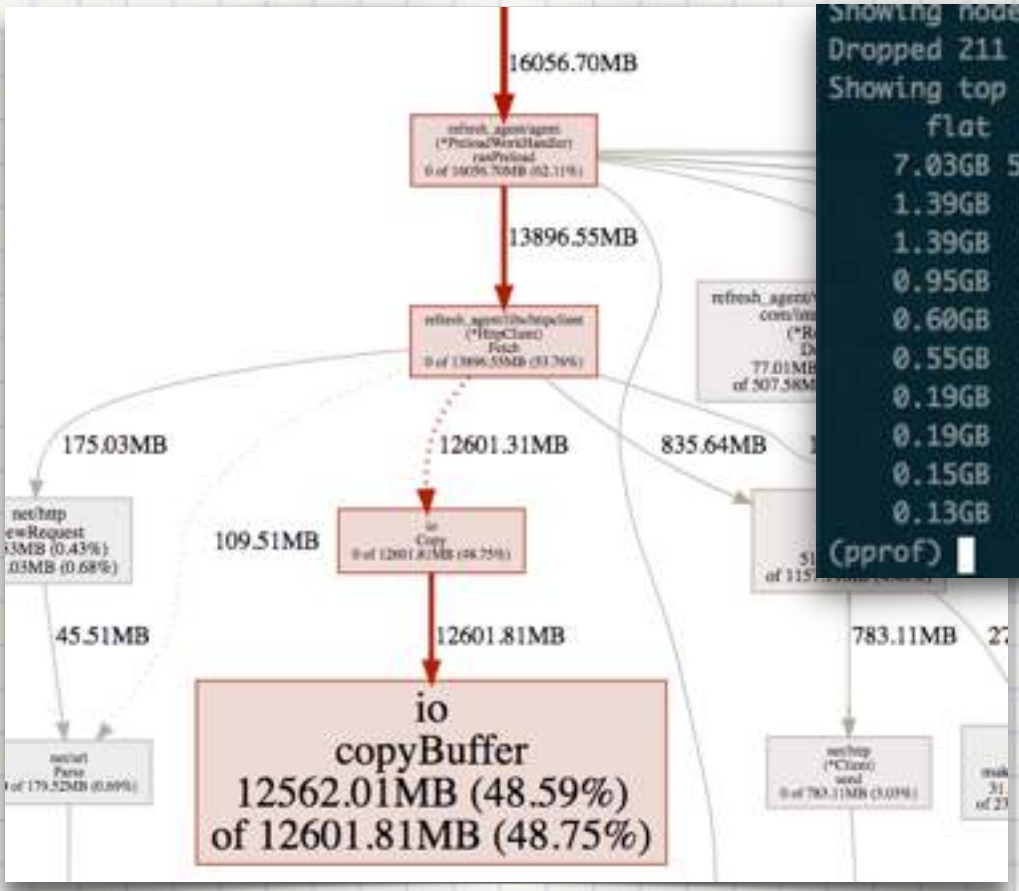
```
scvg3: inuse: 1653, idle: 73, sys: 1727, released: 0, consumed: 1727 (MB)
GC forced
gc 13 @604.438s 0%: 0.024+12+0.013 ms clock, 0.048+0/11/11+0.026 ms cpu, 1->1->1 MB, 4 MB goal, 2 P
gc 326 @597.825s 15%: 0.18+2569+0.075 ms clock, 0.37+976/1287/268+0.15 ms cpu, 1356->1629->935 MB, 1844 MB goal
gc 327 @605.664s 15%: 0.28+2133+0.086 ms clock, 0.56+257/1067/1542+0.17 ms cpu, 1379->1536->823 MB, 1870 MB goal
gc 328 @614.554s 15%: 0.34+1963+0.061 ms clock, 0.69+247/988/1406+0.12 ms cpu, 1344->1483->800 MB, 1646 MB goal
gc 329 @620.163s 15%: 0.23+1971+0.079 ms clock, 0.47+1292/988/170+0.15 ms cpu, 1369->1508->796 MB, 1601 MB goal
gc 330 @624.180s 15%: 0.11+2038+0.090 ms clock, 0.22+1412/1021/0+0.18 ms cpu, 1318->1488->827 MB, 1593 MB goal,
gc 331 @627.806s 15%: 0.13+2302+0.071 ms clock, 0.27+1366/1152/0+0.14 ms cpu, 1300->1522->876 MB, 1654 MB goal,
gc 332 @631.511s 15%: 0.11+2622+0.069 ms clock, 0.22+1211/1312/0+0.13 ms cpu, 1309->1587->930 MB, 1753 MB goal,
gc 333 @635.419s 15%: 0.20+3032+0.99 ms clock, 0.40+1231/1519/0+1.9 ms cpu, 1342->1659->970 MB, 1861 MB goal, 2
gc 334 @639.716s 15%: 0.14+4767+0.090 ms clock, 0.28+1866/2386/0+0.18 ms cpu, 1369->1713->993 MB, 1940 MB goal,
gc 335 @645.690s 15%: 5.8+3756+0.026 ms clock, 11+1305/1881/0+0.052 ms cpu, 1380->1741->1009 MB, 1986 MB goal,
gc 336 @651.168s 16%: 0.22+3906+0.093 ms clock, 0.44+1016/1954/0+0.18 ms cpu, 1399->1801->1067 MB, 2018 MB goal
gc 337 @656.949s 16%: 0.23+4436+0.063 ms clock, 0.47+1035/2218/0+0.12 ms cpu, 1484->1896->1095 MB, 2134 MB goal
```

GODEBUG=gctrace=1

压测环境下，GC占比高达16%

GC的STW时间也会暴到几秒 !!!

GC 耗时调优




```
Showing nodes accounting for 12.58GB, 89.54% of 14.05GB total
Dropped 211 nodes (cum <= 0.07GB)
Showing top 10 nodes out of 97
      flat  flat%   sum%        cum   cum%
    7.03GB 50.05% 50.05%     7.04GB 50.09%  io.copyBuffer
    1.39GB  9.92% 59.97%     2.05GB 14.60%  refresh_agent/libs/logging.(*JSONFormatter).Format
    1.39GB  9.88% 69.85%     1.39GB  9.91%  runtime.mapassign
    0.95GB  6.77% 76.62%     0.95GB  6.77%  runtime.rawstringtmp
    0.60GB  4.30% 80.92%     0.63GB  4.48%  refresh_agent/vendor/github.com/json-iterator/go.(*frozenConfig
    0.55GB  3.95% 84.87%     0.55GB  3.95%  runtime.makemap
    0.19GB  1.35% 86.21%     0.19GB  1.35%  reflect.unsafe_New
    0.19GB  1.34% 87.56%     0.19GB  1.34%  runtime.convT2E
    0.15GB  1.03% 88.59%     2.20GB 15.64%  github.com/Sirupsen/logrus.Entry.log
    0.13GB  0.95% 89.54%     0.66GB  4.73%  github.com/Sirupsen/logrus.(*Entry).WithFields
(pprof)
```

通过Heap确定, io.copyBuffer逻辑导致的问题！

业务代码是通过io.Copy来调用的copyBuffer！

GC 耗时调优

```go
//err write /dev/null: bad file descriptor#
out, err := os.OpenFile("/dev/null", os.O_RDWR|os.O_CREATE|os.O_APPEND, 0666)
defer out.Close()
buf := buffPool.Get()
if buf == nil {
    buf = make([]byte, 32*1024)
}
io.CopyBuffer(out, resp.Body, buf.([]byte))
// io.Copy(out, resp.Body)
buffPool.Put(buf)

return err
}
```

```go
// the copy is implemented by calling dst.ReadFrom(src).
func Copy(dst Writer, src Reader) (written int64, err error) {
    return copyBuffer(dst, src, nil)
}

func CopyBuffer(dst Writer, src Reader, buf []byte) (written int64, err error) {
    if buf != nil && len(buf) == 0 {
        panic("empty buffer in io.CopyBuffer")
    }
    return copyBuffer(dst, src, buf)
}

func copyBuffer(dst Writer, src Reader, buf []byte) (written int64, err error) {
    // If the reader has a WriteTo method, use it to do the copy.
    // Avoids an allocation and a copy.
    ...
    if buf == nil {
        buf = make([]byte, size)
    }
    for {
```

当为nil的时候, copyBuffer会new一个对象

使用sync.pool来复用[]byte对象

GC 耗时调优

```
gc 17 @83.363s 1%: 0.075+119+0.083 ms clock, 0.15+134/61/0+0.16 ms cpu, 296->298->57 MB, 308 MB goal, 2 P
gc 18 @97.136s 1%: 0.094+115+0.033 ms clock, 0.18+126/61/0+0.066 ms cpu, 274->276->51 MB, 285 MB goal, 2 P
gc 19 @100.916s 1%: 0.057+119+0.094 ms clock, 0.11+124/70/0+0.18 ms cpu, 249->250->47 MB, 259 MB goal, 2 P
gc 20 @105.609s 1%: 0.091+85+0.046 ms clock, 0.18+12/43/104+0.093 ms cpu, 229->229->42 MB, 239 MB goal, 2 P
gc 21 @114.230s 1%: 0.058+89+0.031 ms clock, 0.11+92/49/0+0.062 ms cpu, 203->205->40 MB, 211 MB goal, 2 P
gc 22 @116.039s 1%: 0.30+67+0.032 ms clock, 0.60+69/36/0+0.065 ms cpu, 192->193->34 MB, 200 MB goal, 2 P
gc 23 @116.949s 1%: 0.19+51+0.11 ms clock, 0.39+50/33/0+0.22 ms cpu, 167->169->31 MB, 174 MB goal, 2 P
gc 24 @117.384s 1%: 0.086+51+0.075 ms clock, 0.17+47/32/0+0.15 ms cpu, 153->155->34 MB, 159 MB goal, 2 P
gc 25 @117.971s 1%: 0.080+48+0.063 ms clock, 0.16+44/30/0+0.12 ms cpu, 163->165->32 MB, 170 MB goal, 2 P
gc 26 @118.747s 1%: 0.13+55+0.081 ms clock, 0.27+56/30/0+0.16 ms cpu, 156->157->32 MB, 162 MB goal, 2 P
gc 27 @119.856s 1%: 0.056+60+0.094 ms clock, 0.11+58/30/0.58+0.18 ms cpu, 158->159->35 MB, 164 MB goal, 2 P
GC forced
```

已经压缩到 1%, gc的次数明显减少！

GC 性能对比

在我们的项目里

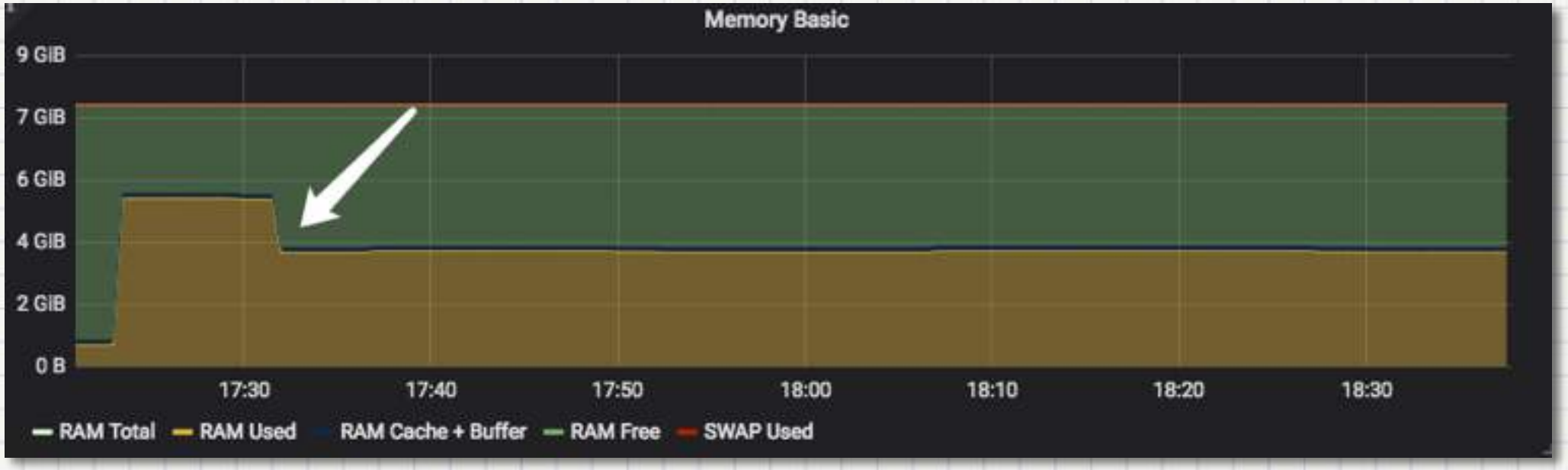
go 1.10 > go 1.9 > go 1.8

sync.pool 效果明显

内存释放不干净

当使用全局map时, 会造成GC已清理，但内存释放不彻底问题?

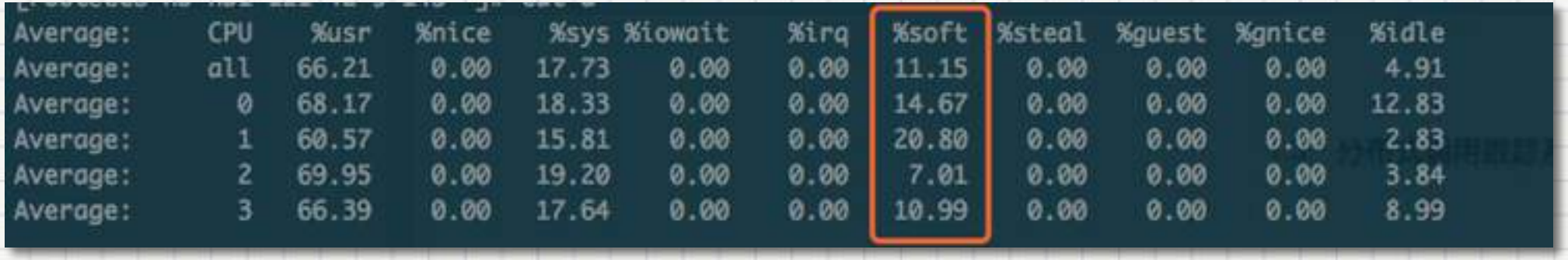
可信号调用debug.FreeOSMemory来释放更多的mspan.



cpu软中断

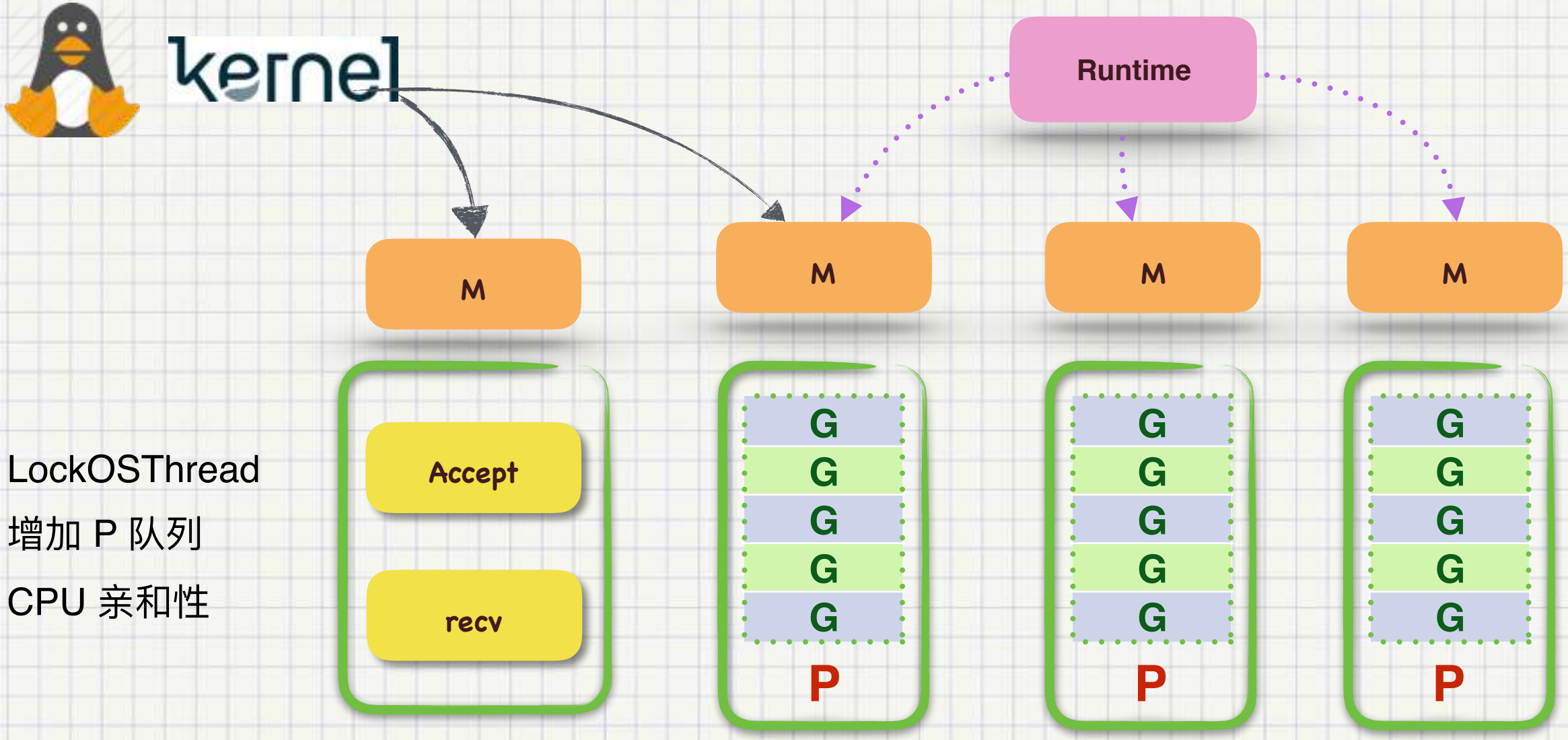
注意软中断打满cpu

- 多队列网卡 + SMP affinity
- 采用RPS/ RFS软件模拟多队列



Average:	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
Average:	all	66.21	0.00	17.73	0.00	0.00	11.15	0.00	0.00	0.00	4.91
Average:	0	68.17	0.00	18.33	0.00	0.00	14.67	0.00	0.00	0.00	12.83
Average:	1	60.57	0.00	15.81	0.00	0.00	20.80	0.00	0.00	0.00	2.83
Average:	2	69.95	0.00	19.20	0.00	0.00	7.01	0.00	0.00	0.00	3.84
Average:	3	66.39	0.00	17.64	0.00	0.00	10.99	0.00	0.00	0.00	8.99

主动介入runtime调度调度

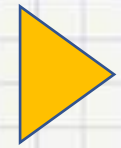


- LockOSThread
- 增加 P 队列
- CPU 亲和性

压测

- 压测工具
  - wrk > ab > golang
- 不要本地环路测试
- 多源压力
- 注意带宽问题
- 注意内核的种种error
- more…

```
Mar 15 16:56:55 xxx kernel: nf_conntrack: table full, dropping packet.
Mar 15 16:56:55 xxx kernel: nf_conntrack: table full, dropping packet.
Mar 15 16:56:55 xxx kernel: nf_conntrack: table full, dropping packet.

Mar 10 13:45:11 xxxx kernel: TCP: time wait bucket table overflow
Mar 10 13:45:11 xxxx kernel: TCP: time wait bucket table overflow
Mar 10 13:45:11 xxxx kernel: TCP: time wait bucket table overflow

more ….
```



Wrk

```
wrk -t8 -c800 -d100s -T10s --script=js --latency http://xxxxx:9181/v1/engine/task
```

```
thread addr: xxxxx
...
  8 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    19.60ms   11.97ms 144.14ms   73.42%
    Req/Sec     23.73k   2784.29     29.85k    79.29%
  Latency Distribution
     50%   17.90ms
     75%   24.92ms
     90%   34.63ms
...
Requests/sec:   23189.11
Transfer/sec:    5.06MB
```

4 上线部署

打包方法

Engine端

- build
- pack
- repo
- salt
- unpack
- reload

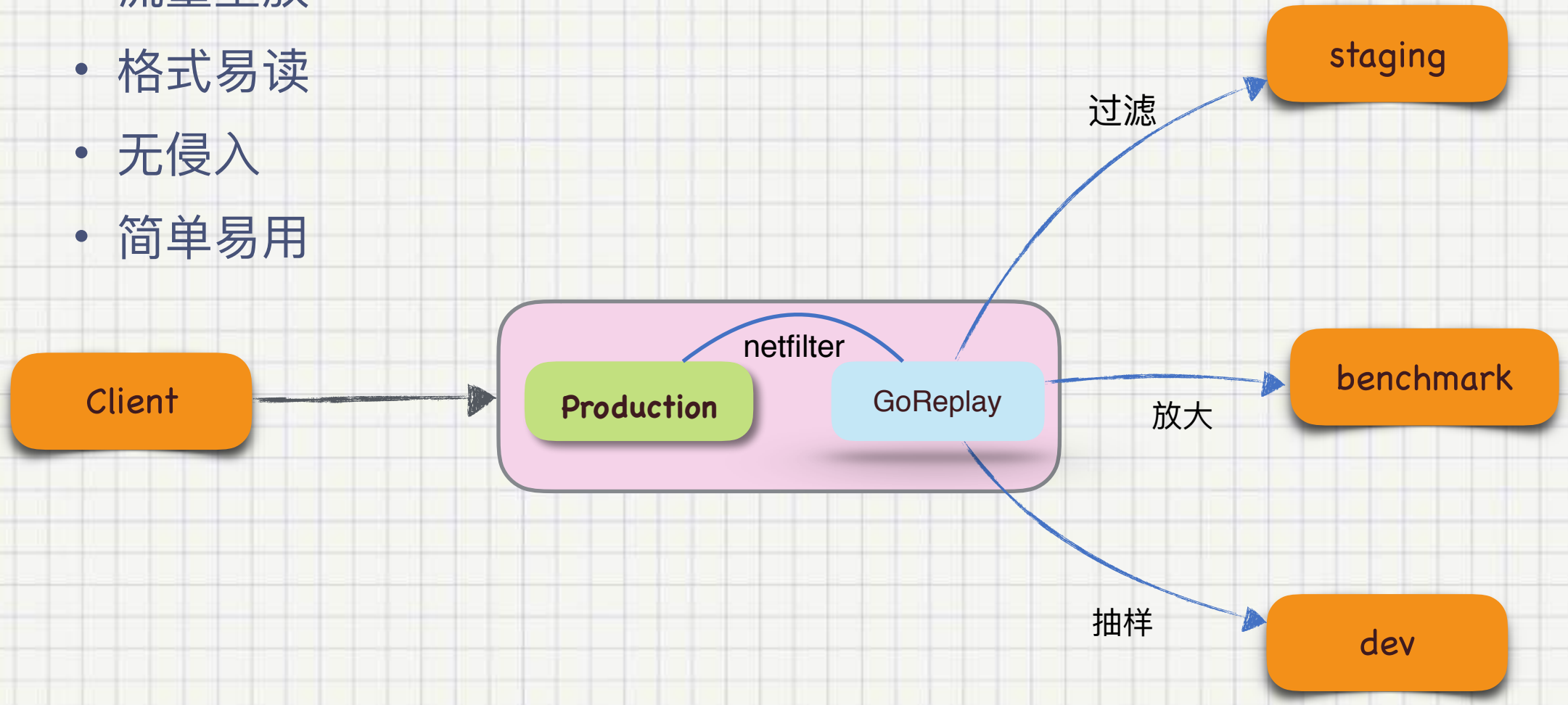
Proxy端，封装RPM包上线 .

```
go build -ldflags "-X main.Version=V0.1.10" main.go
```

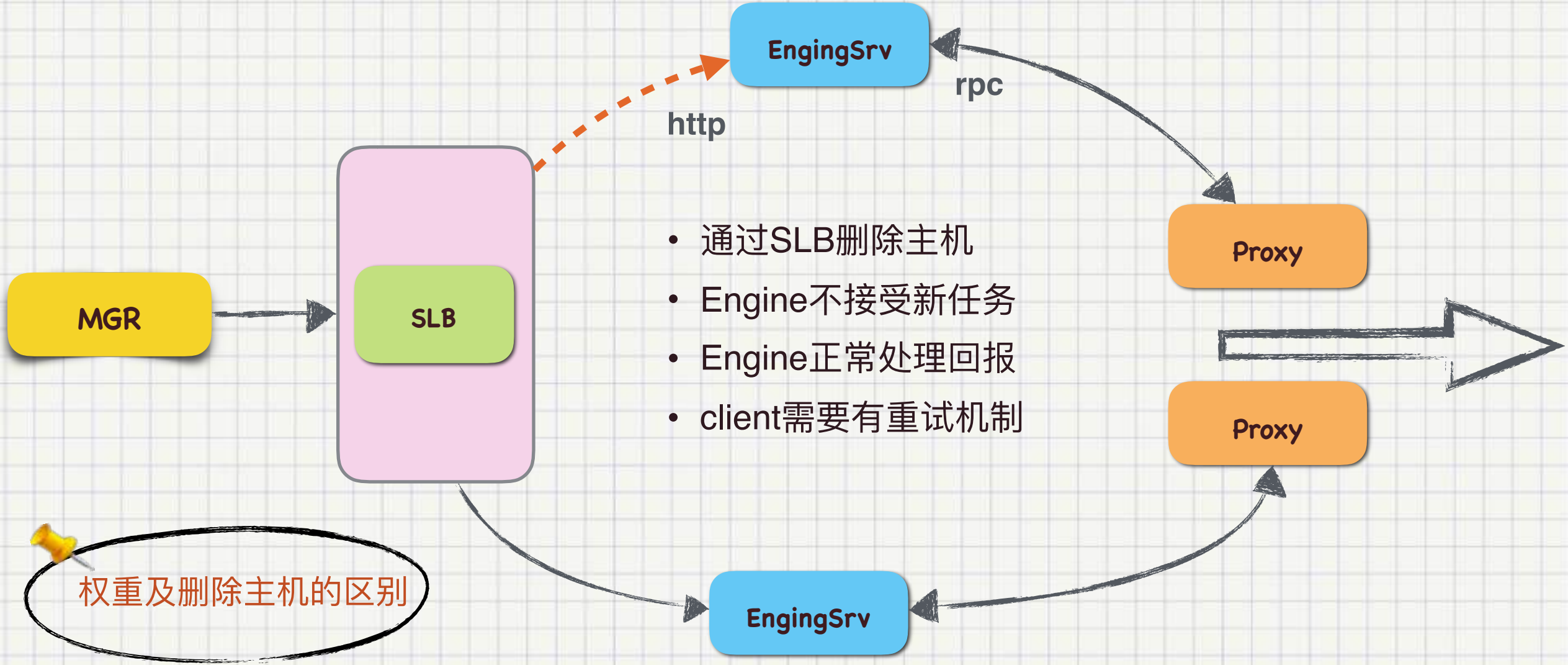
```
var (
    // go build with flag
    Version    = ""
    CommitID   = ""
    CommitDesc = ""

    // force exit timeout
    exitTimeout = 60
)
```

在线流量回放

- 流量重放
- 格式易读
- 无侵入
- 简单易用



灰度平滑迁移-SLB层

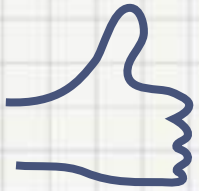


- 通过SLB删除主机
- Engine不接受新任务
- Engine正常处理回报
- client需要有重试机制

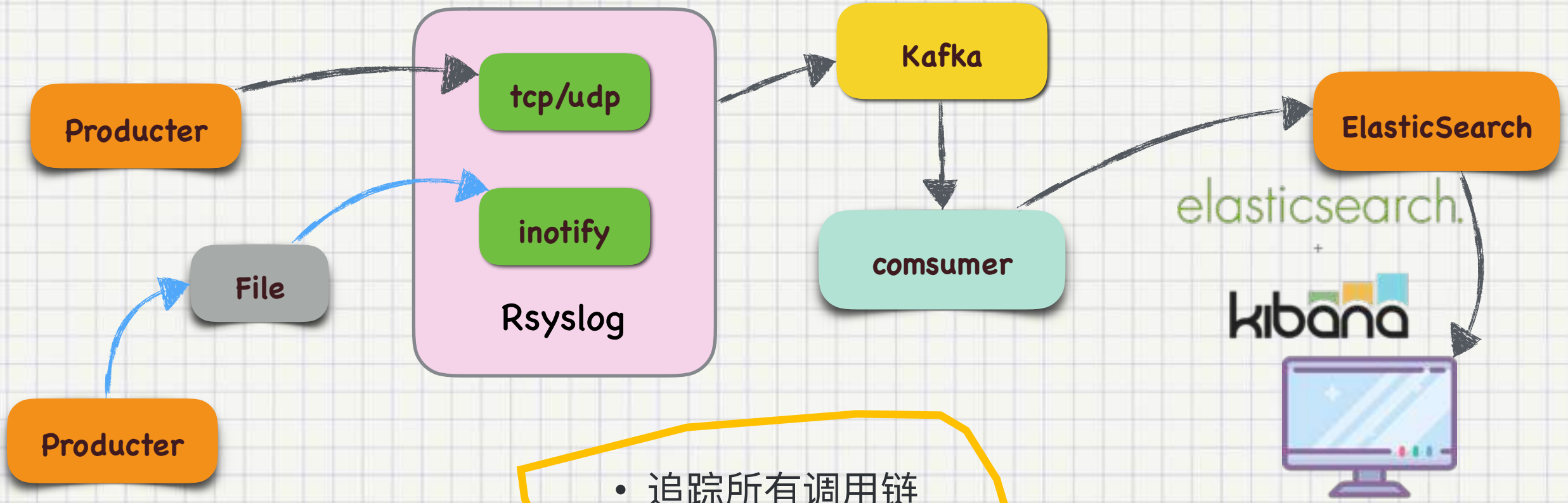
灰度平滑迁移-框架层

- 从调用链最前端开始顺序关闭.
- 持久化的数据要优先保证
- join住所有的业务协程.

```
# 伪代码
recvAndSetExit(signal)
publishSigWorerks()

if globalExited && len(queue) == 0 && pushExit {
        return
}
wait(saveQueueExit)
wait(eventBus)
```



收集分布式日志



- 追踪所有调用链
- 计算耗时
- 排查问题

如何报警通知

服务本身主动去上报异常！

异步去上报！

异常抽样报警！

状态监控页面

模块心跳

Module	Status	Diff	LastTime
poller	✔	2	2018-03-23 14:29:13
reater	✔	2	2018-03-23 14:29:13
scaner	✔	1	2018-03-23 14:29:14
[illegible]	[illegible]	[illegible]	[illegible]

节点心跳

Module	Status	Diff	LastTi
[illegible]	✔	0	2018-0
[illegible]	✔	0	2018-0
[illegible]	✔	0	2018-0
[illegible]	✔	0	2018-0
[illegible]	✔	2	2018-0
[illegible] /70	—	597[illegible]	2018-0
[illegible] /71	—	597[illegible]	2018-0
[illegible]	—	597[illegible]	2018-0
[illegible] 1 [illegible]	—	597[illegible]	2018-0

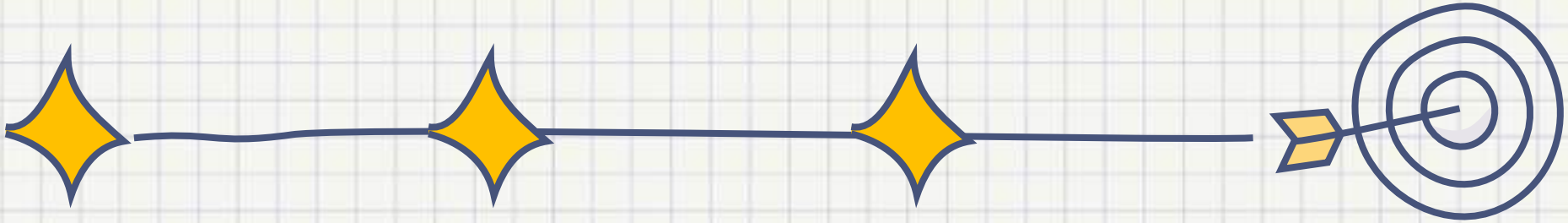
API_ID计数器

API_ID	INIT	PENDING	RUNNIN
2[illegible]	0	0	0
[illegible]	0	0	0
[illegible]	0	0	0
[illegible]	0	0	0
[illegible]	0	0	0
[illegible]	0	0	0
[illegible]	0	0	0

5

总结

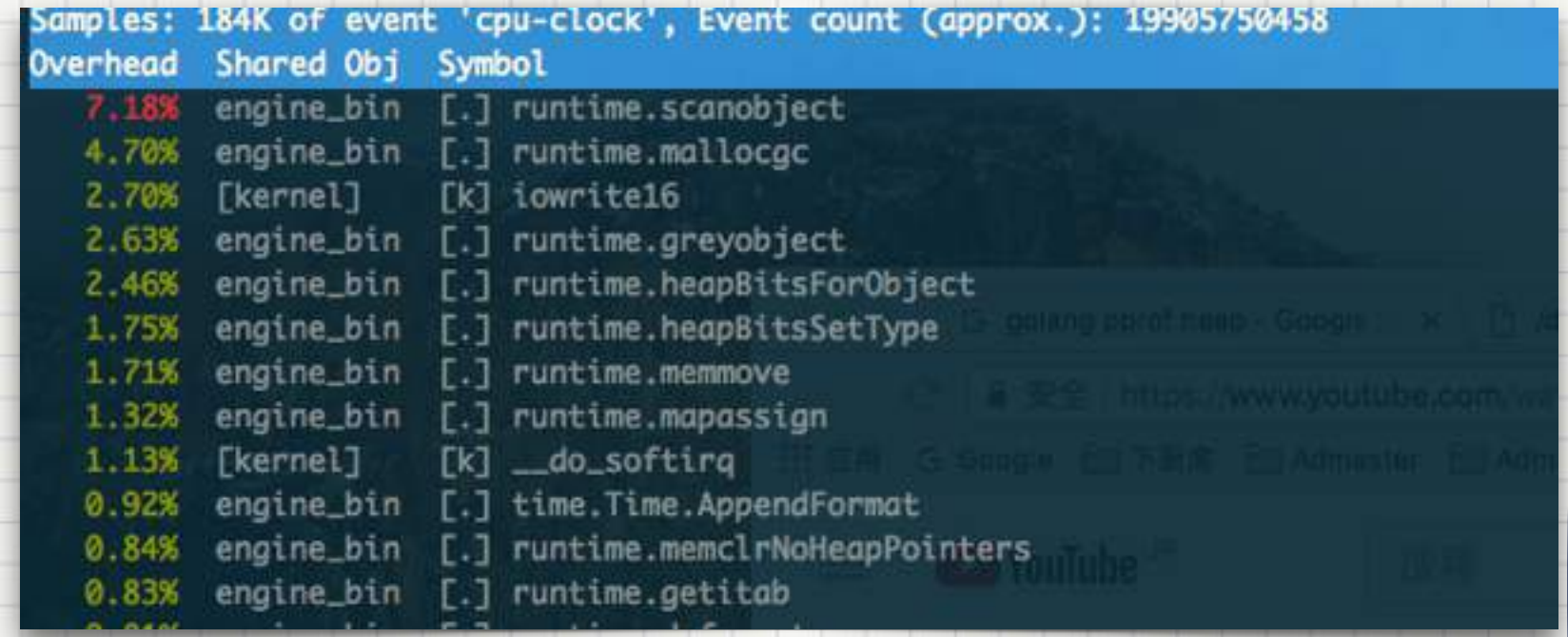
性能演进



1w　　1.5w　　2w　　3w +

QPS

GC 还需优化


Samples: 184K of event 'cpu-clock', Event count (approx.): 19905750458
Overhead Shared Obj Symbol
7.18% engine_bin [.] runtime.scanobject
4.70% engine_bin [.] runtime.mallocgc
2.70% [kernel] [k] iowrite16
2.63% engine_bin [.] runtime.greyobject
2.46% engine_bin [.] runtime.heapBitsForObject
1.75% engine_bin [.] runtime.heapBitsSetType
1.71% engine_bin [.] runtime.memmove
1.32% engine_bin [.] runtime.mapassign
1.13% [kernel] [k] __do_softirq
0.92% engine_bin [.] time.Time.AppendFormat
0.84% engine_bin [.] runtime.memclrNoHeapPointers
0.83% engine_bin [.] runtime.getitab

协程太多

profiles:

0 block

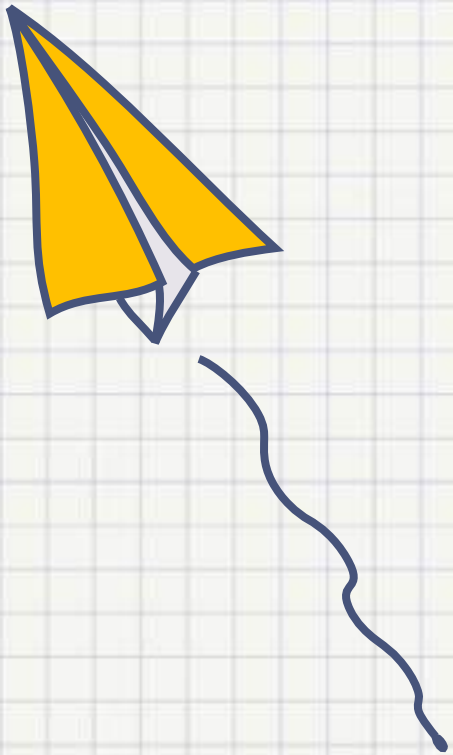
119865 goroutine

1231 heap

0 mutex

12 threadcreate

full goroutine stack dump



Q & A