



Golang 高性能实战

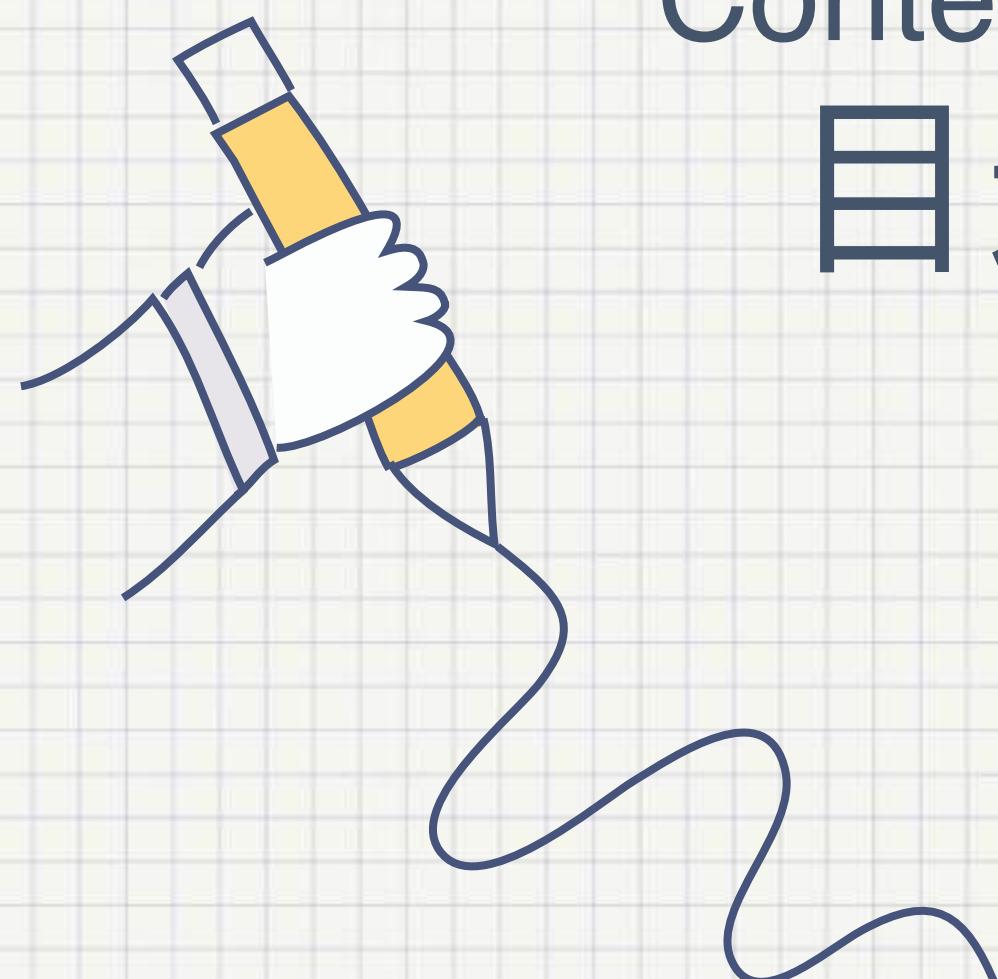
github.com/rfyiamcool
xiaorui.cc



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.

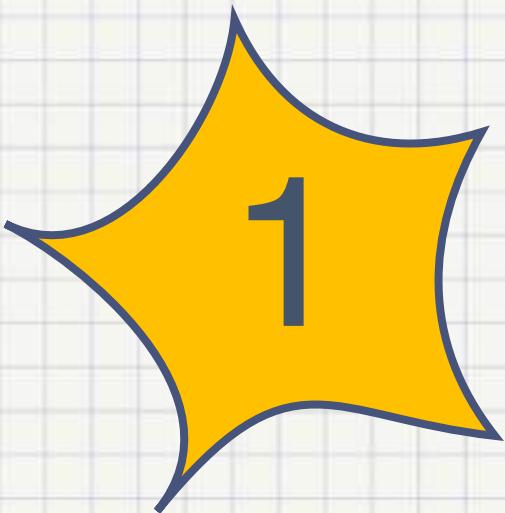




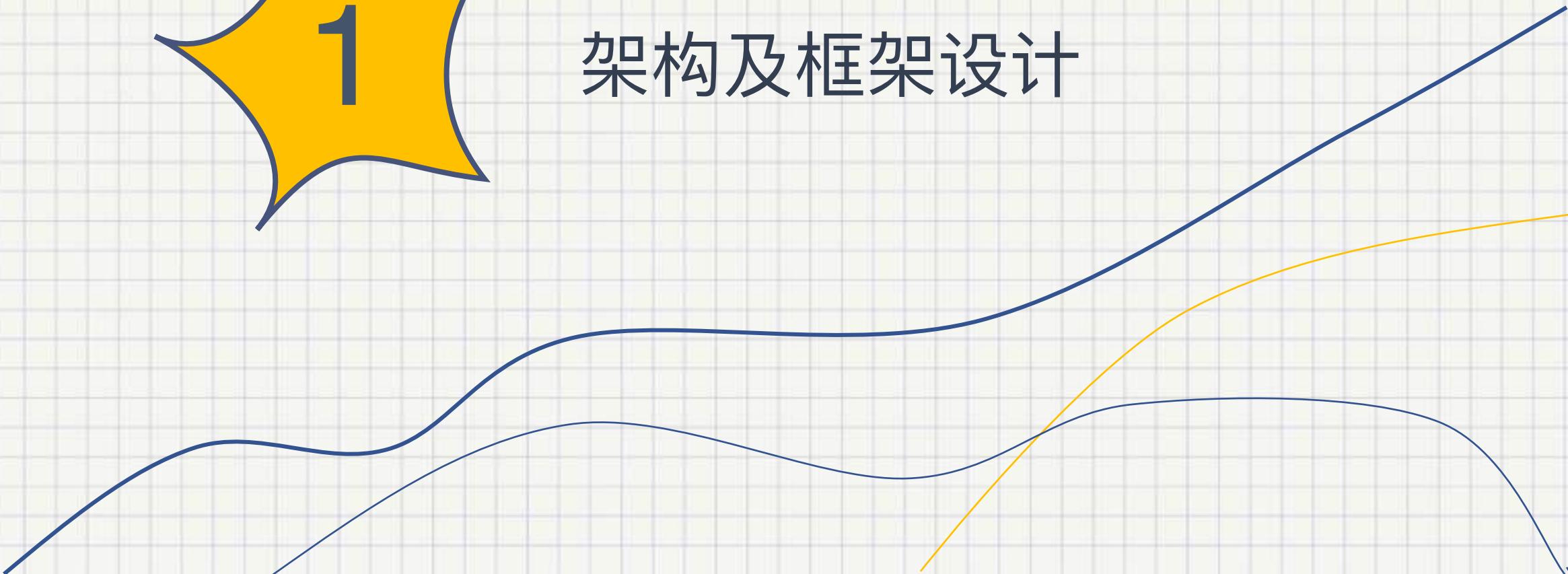
Contents

目录

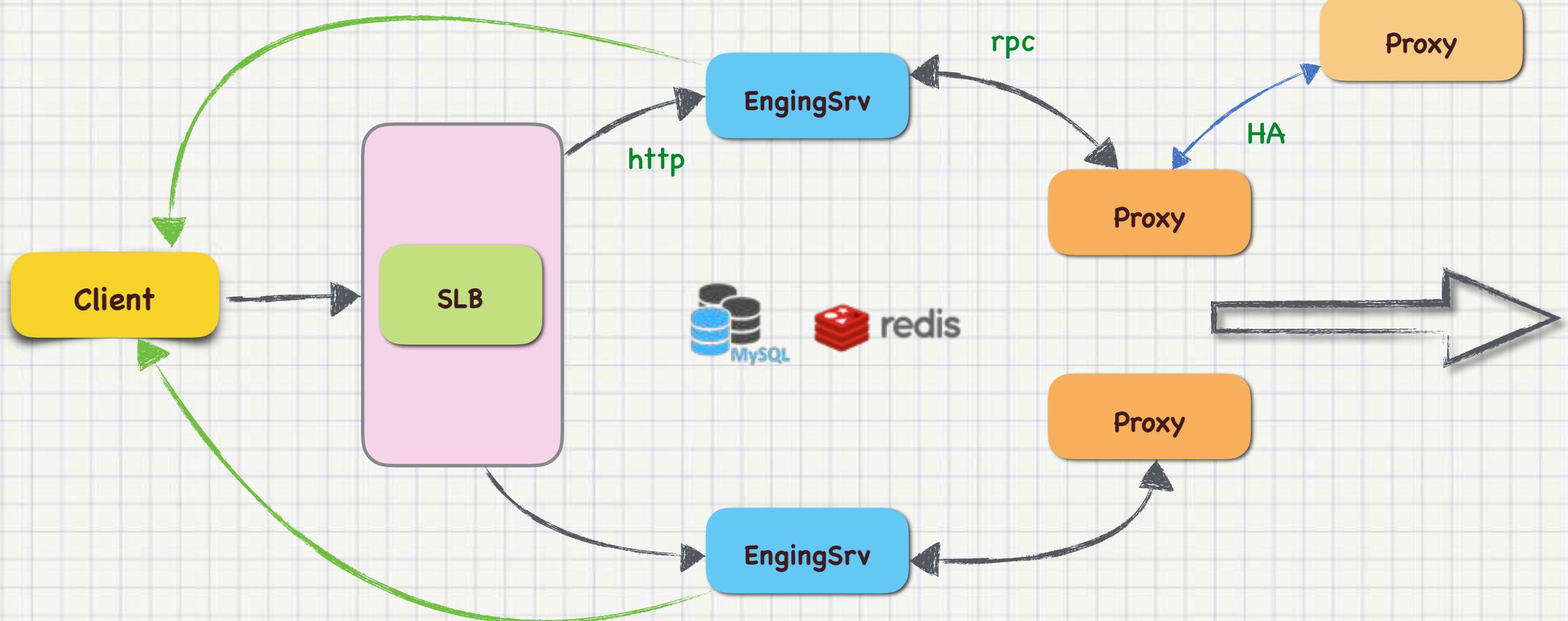
- 1 架构及框架
- 2 调优
- 3 深度排雷
- 4 上线部署
- 5 总结



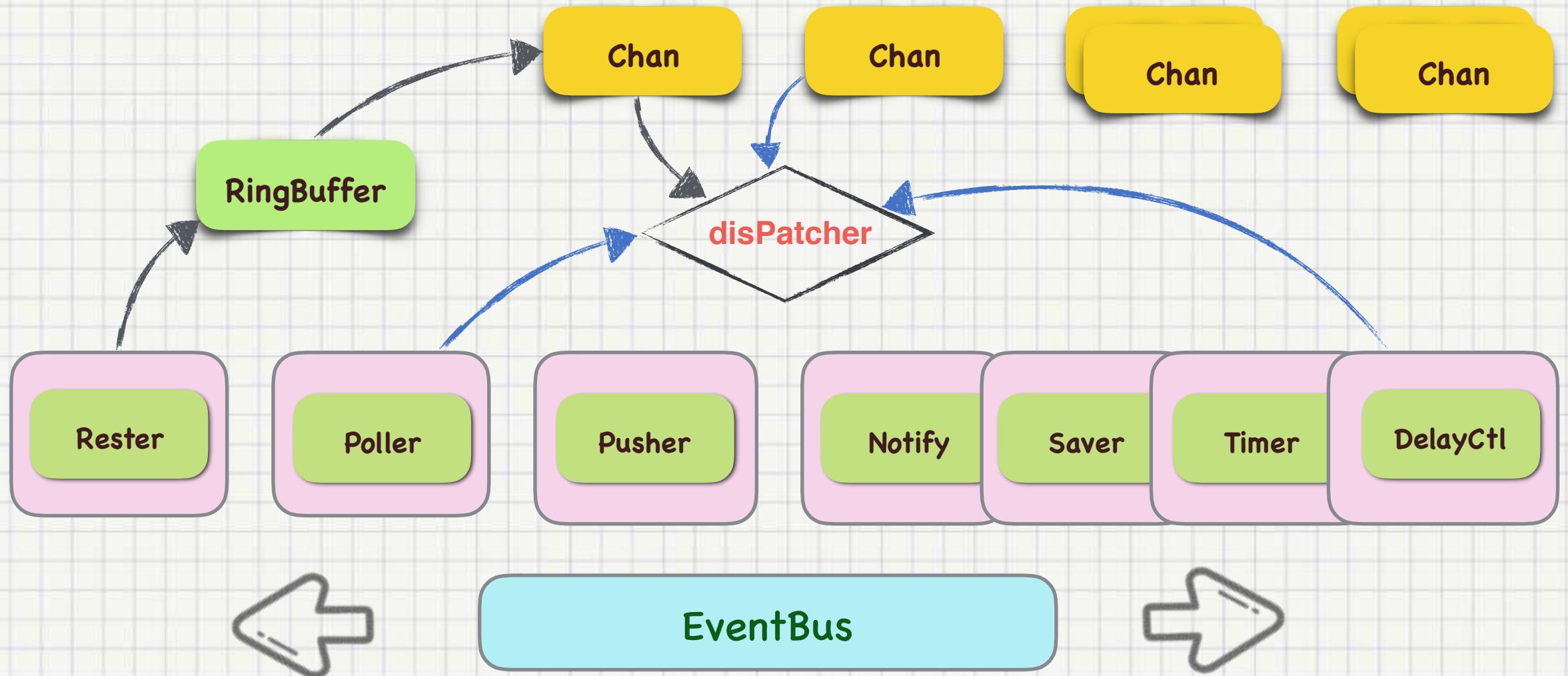
架构及框架设计



架构设计



框架设计





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



日志: Logrus



框架: Gin



配置: Toml



压缩: Snappy

缓存: Redis



序列化: Json-iterator

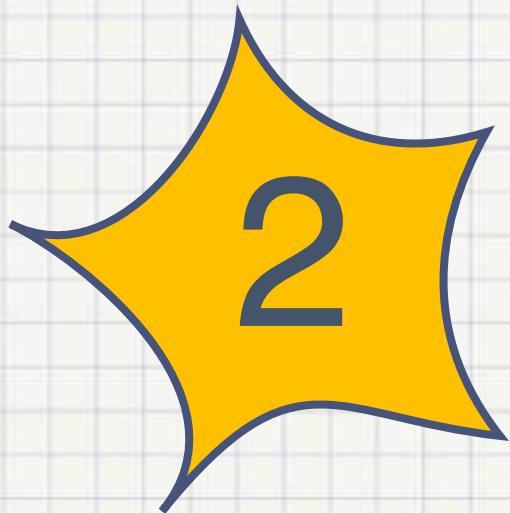
持久化: Mysql



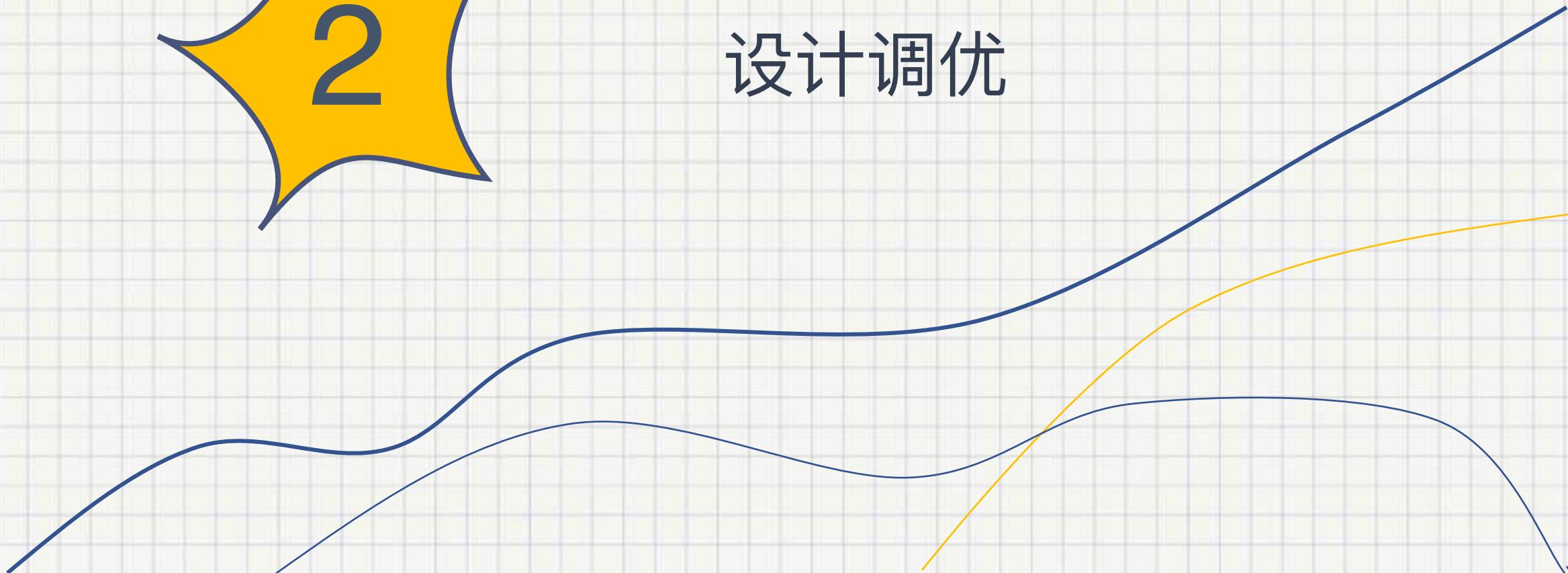
ORM: Gorm

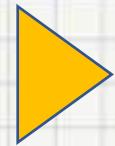


客户端: imroc/req



设计调优

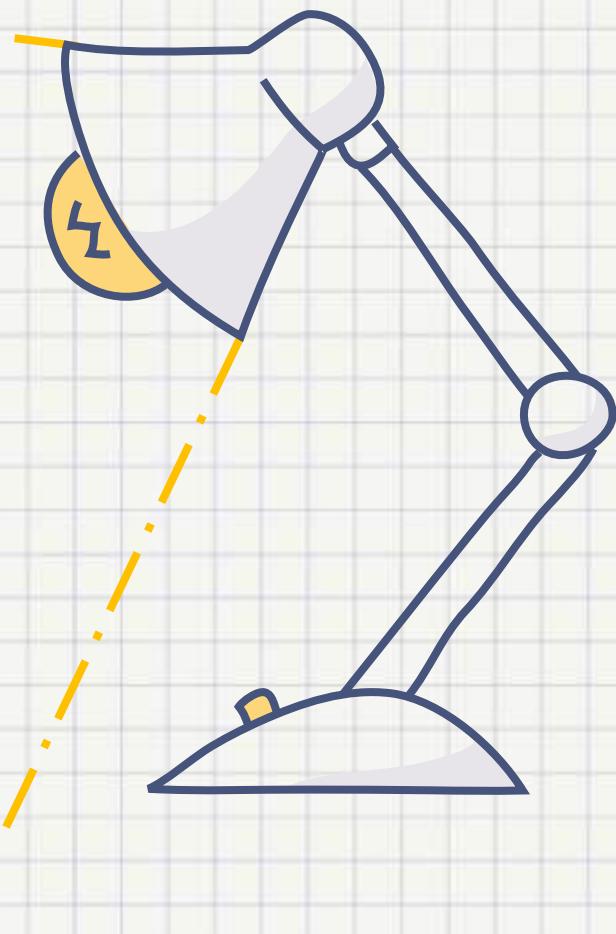


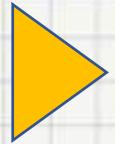


基本优化



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





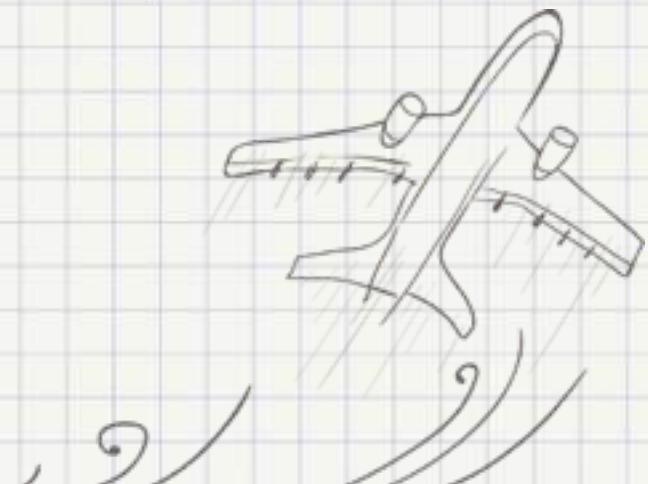
基本优化

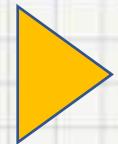


不要滥用time.After, 多用事件通知
不要滥用goroutine, 减少gc压力
不要滥用锁, 引起runtime上下文切换



使用unsafe装换类型
不要总是[]byte、String转换.





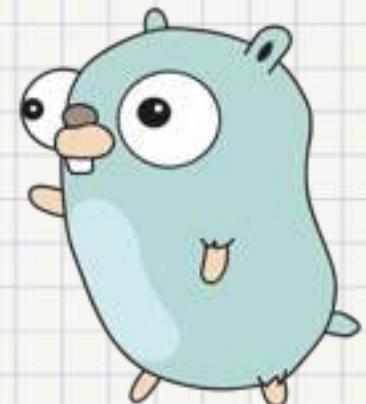
基本优化



- 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

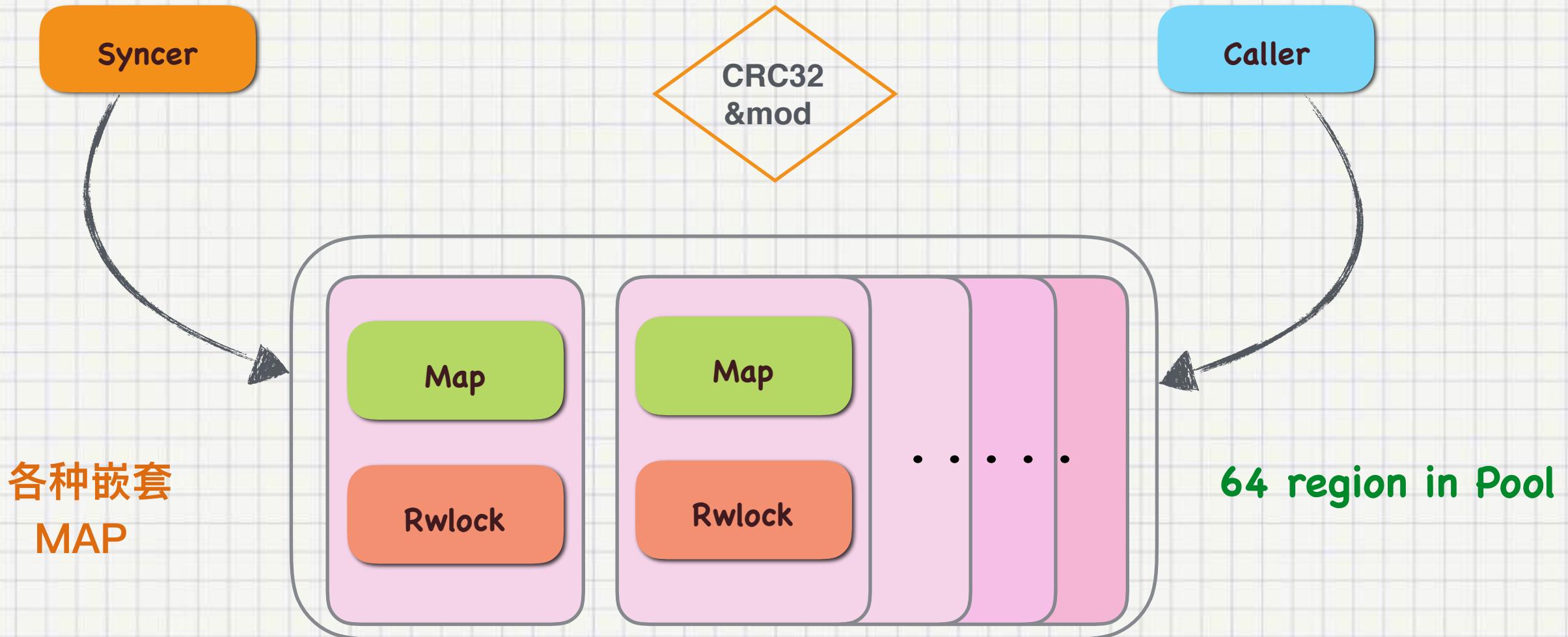
```
$ nc -l 8811 -k
POST /v1/agent/task HTTP/1.1
Host: 127.0.0.1:8811
User-Agent: Go-http-client/1.1
Content-Length: 329
Accept: application/json
Accesskey: f0365d34721de99e6f7e9cb5ccda1258
Compress: snappy
Timestamp: 1522114624
Token: 9114b782f14b7428849c78b4a528c5d618f64af8
Accept-Encoding: gzip

{"task_id": "1522114624", "proxy_ip": "27.0.0.1", "due_timestamp": 51, "callback_api": "http://127.0.0.1:8811/v1/agent/callback", "type": "refresh", "callback_token": "9114b782f14b7428849c5d618f64af8", "sub": 0, "active": 0, "scope": null, "storage_zjnb": "c01.i01", "force": false, "subtask_id": "1000000022511092", "due_timestamp": 51}
```

HTTP vs RPC

```
$ nc -l 8811 -k
post_task access_key f0365d34721de99e6f7e9cb5ccda1258 callback_api 127.0.0.1 token 9114b782f14b7428849c5d618f64af8 task_type refresh proxy_ip ip task_id 1522114624 hosts_tasks cacheip url http://127.0.0.1:8811/v1/agent/callback force subtask_id 1000000022511092 due_timestamp 51
```

全局任务状态





全局任务状态-code



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

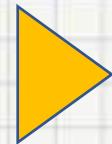
type TaskMapRegion struct {
    map[string]*DetailTaskStats
    *sync.RWMutex
}

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
}
```

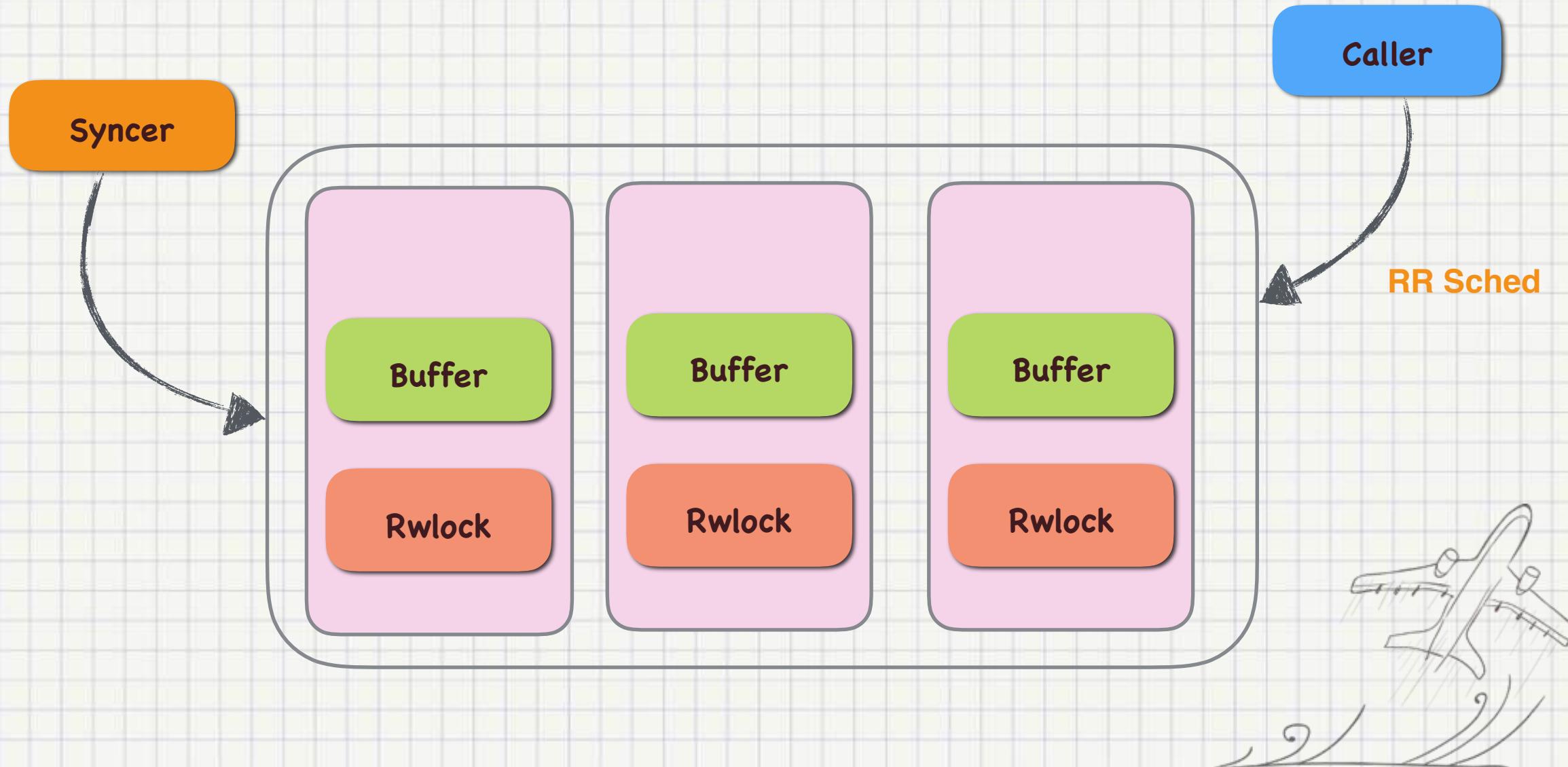
```
func GetOneMapRegion(taskId string) *TaskMapRegion {
    idx := utils.HashToInt(taskId) % PreAllocRegion
    return TaskMapRegionPool[idx]
}

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

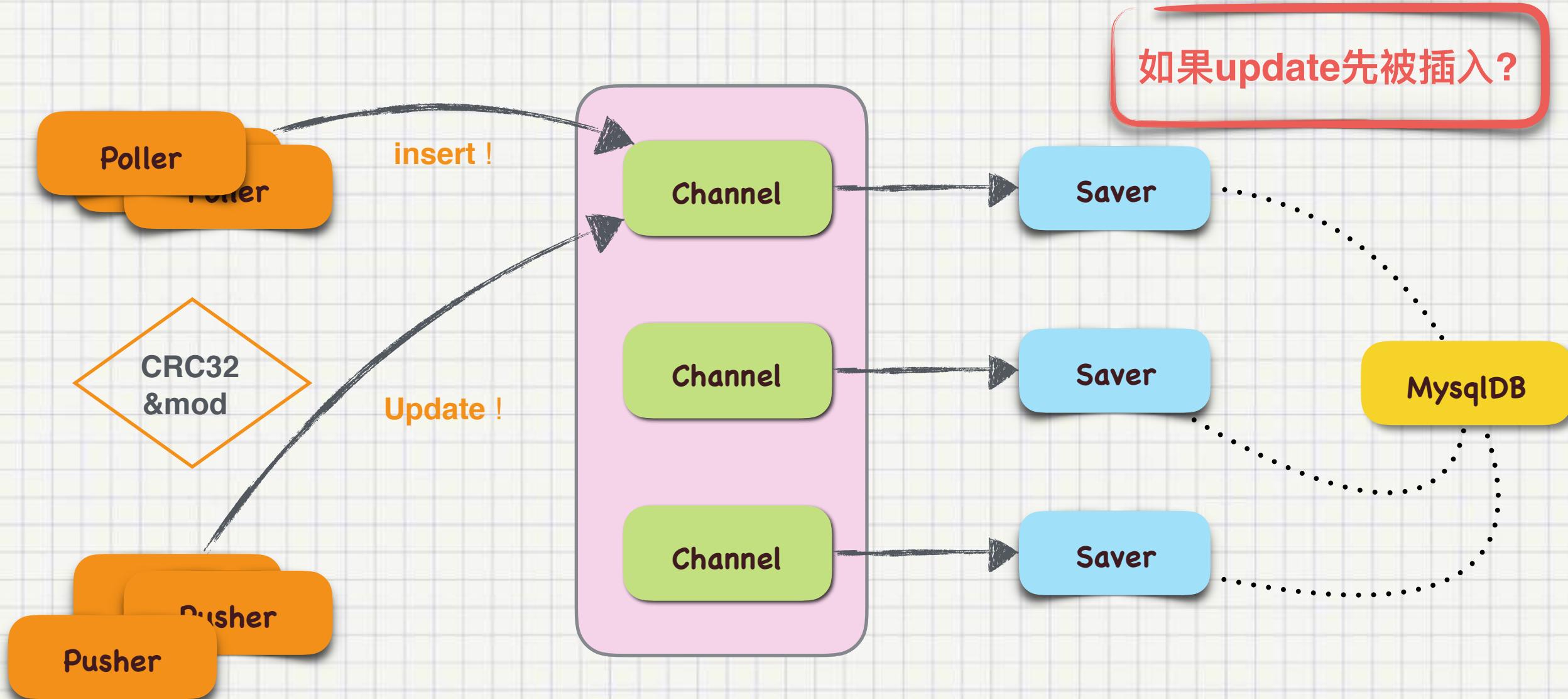
    _, ok := reg.HasTaskID(taskID)
    return ok
}
```



缓存多副本



操作顺序排队





操作顺序排队-CODE

```
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
}
```

```
// 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
    }

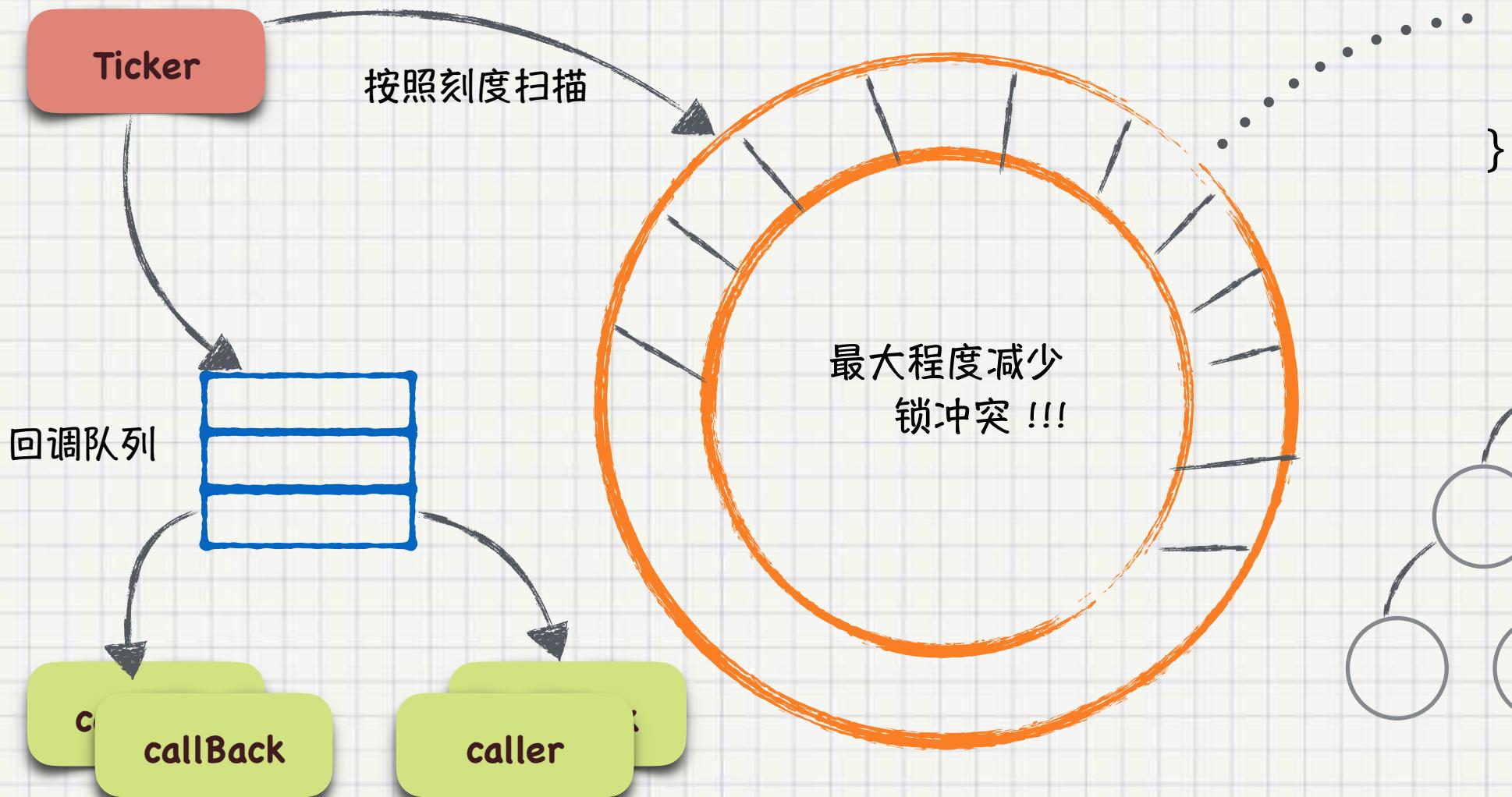
    return outDto, ok
}

// 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



```
struct timerEntry {  
    heap  
    cas  
};
```

小顶堆



分布式选主

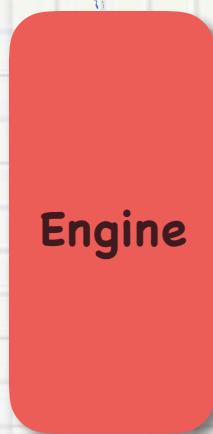
set key hostname
ex 10
nx



```
for{
    if set(key, nx, ex) {
        expire(10)
    }
    time.Sleep(n)
}
```



set
ex 过期
nx 原子



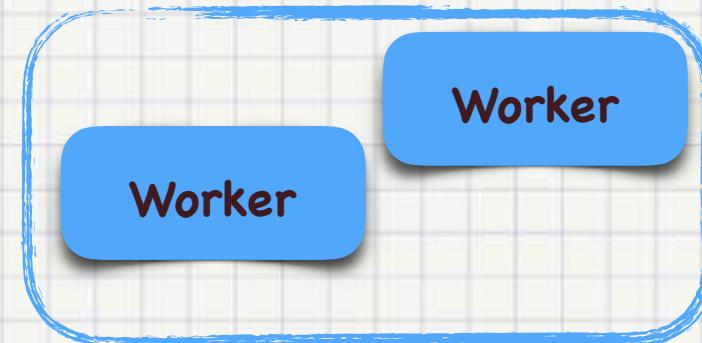


区分快慢队列

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

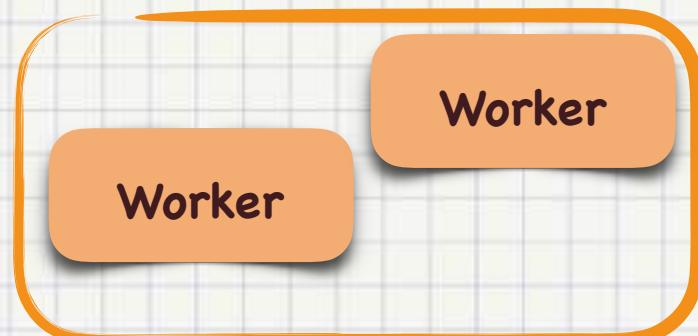
Slow Queue



提高吞吐！

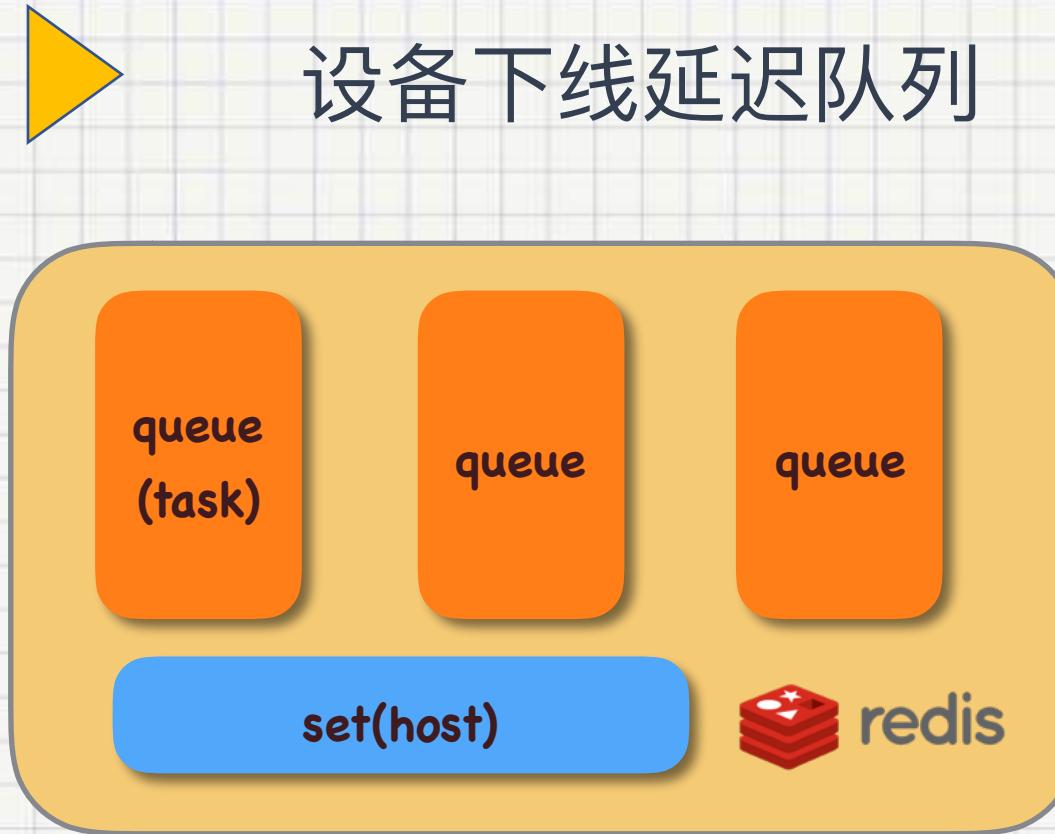
Worker

Worker



```
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
```

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



设备下线延迟队列

吞吐提升
30%

下线/上线

Control

bulk put

proxy
proxy
proxy



排查死锁技巧

Caller

Acquie()



方便调试, 抽象锁方法.

Caller

Release()



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

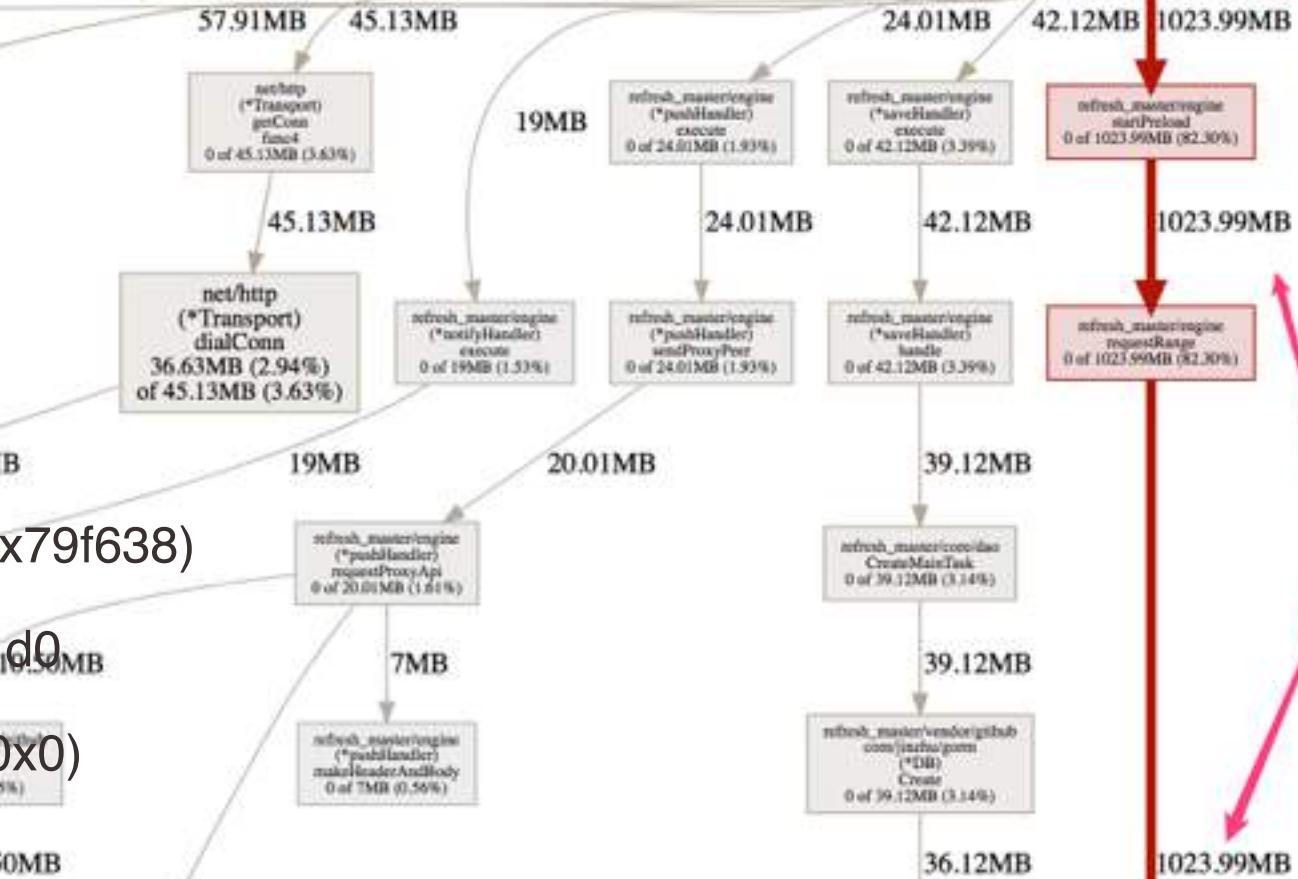
runtime.sysMap(0xc4a0200000, 0x80000000, 0x0, 0x79f638)

/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

File: engine_bin
Type: alloc_space
Time: Mar 24, 2018 at 1:06pm (CST)
Showing nodes accounting for 1185.17MB, 95.26% of 1244.17MB total
Dropped 215 nodes (cum <= 6.22MB)





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！

Engine

counter
INCRBY()
SYNC()
RESET()

5s sync

全局计数

incr

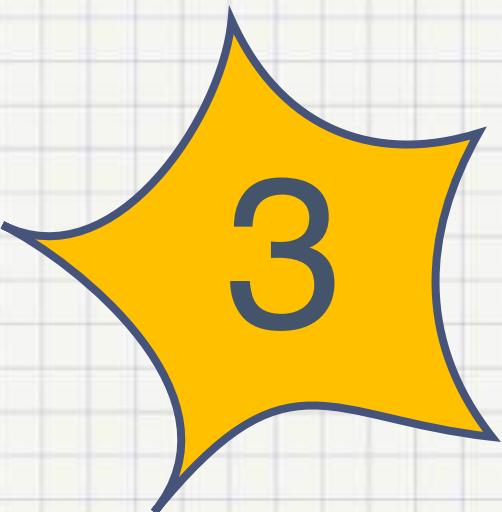
异步同步

5s sync

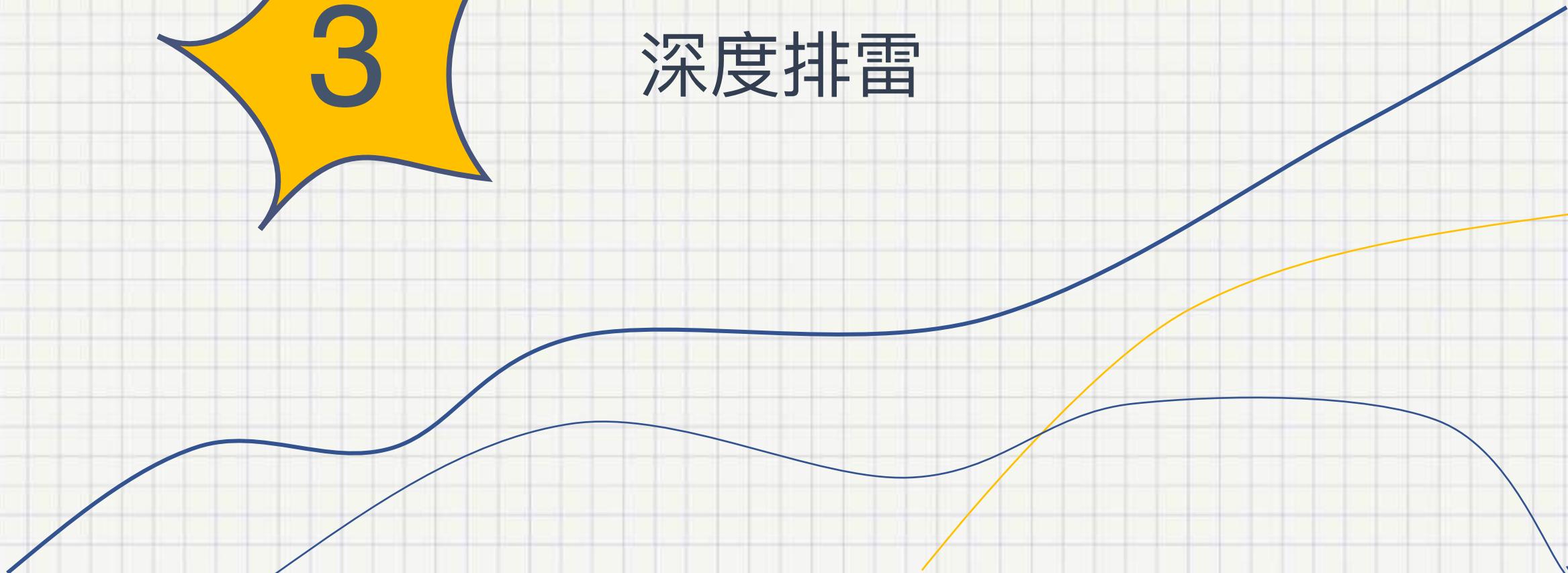
counter

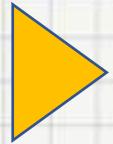
INCRBY()
SYNC()
RESET()

Engine



深度排雷





SystemTool



perf top



lsof、netstat、sar



go tool pprof



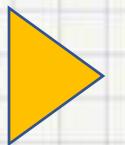
strace



iostat、vmstat、top



wrk



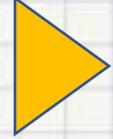
Established 暴增问题

```
task_disp 14138 root 130u IPv4 739219083 0t0 TCP 10.3...:17170->1... (ESTABLISHED)
task_disp 14138 root 132u IPv4 739219051 0t0 TCP 10.3...:17106->1... (ESTABLISHED)
task_disp 14138 root 136u IPv4 739219055 0t0 TCP 10.3...:17114->1... (ESTABLISHED)
task_disp 14138 root 137u IPv4 739219056 0t0 TCP 10.3...:17116->1... (ESTABLISHED)
task_disp 14138 root 138u IPv4 739219057 0t0 TCP 10.3...:17118->1... (ESTABLISHED)
task_disp 14138 root 139u IPv4 739219058 0t0 TCP 10.3...:17120->1... (ESTABLISHED)
task_disp 14138 root 140u IPv4 739219059 0t0 TCP 10.3...:17122->1... (ESTABLISHED)
task_disp 14138 root 141u IPv4 739219060 0t0 TCP 10.3...:17124->1... (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



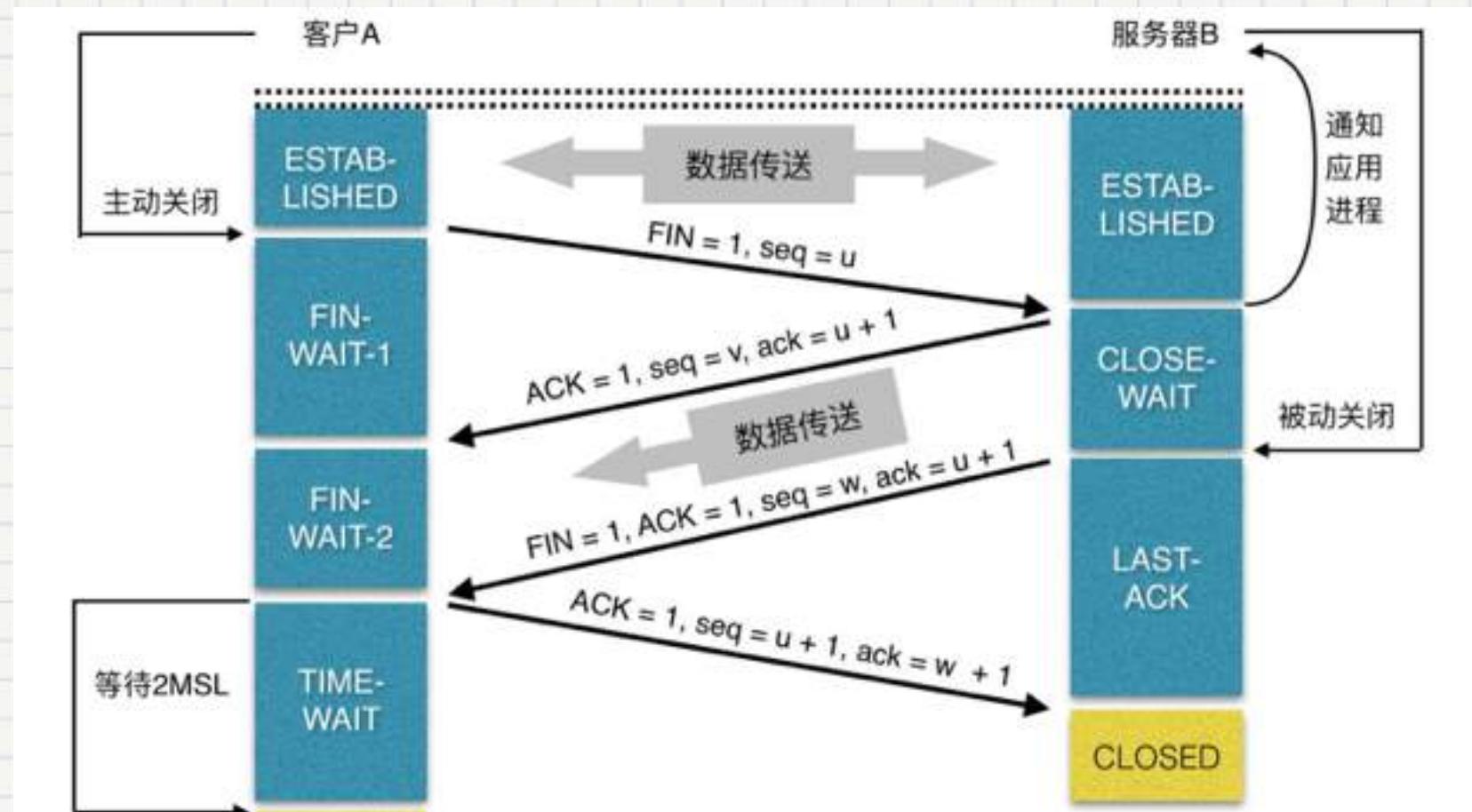
连接池引起的TimeWait暴增

TimeWait 暴增到 3.5W ...

TimeWait 暴增到 3W ...

TimeWait 暴增到 5W ...

TimeWait 暴增到 4W ...





连接池引起的TimeWait暴增

TimeWait 暴增到 3.5W ...

TimeWait 暴增到 3W ...

TimeWait 暴增到 4W ...

TimeWait 暴增到 5W ...

`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

profiles:

0 [block](#)

398 [goroutine](#)

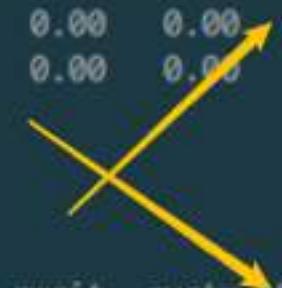
760 [heap](#)

0 [mutex](#)

1009 [threadcreate](#)

[full goroutine stack dump](#)

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
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





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

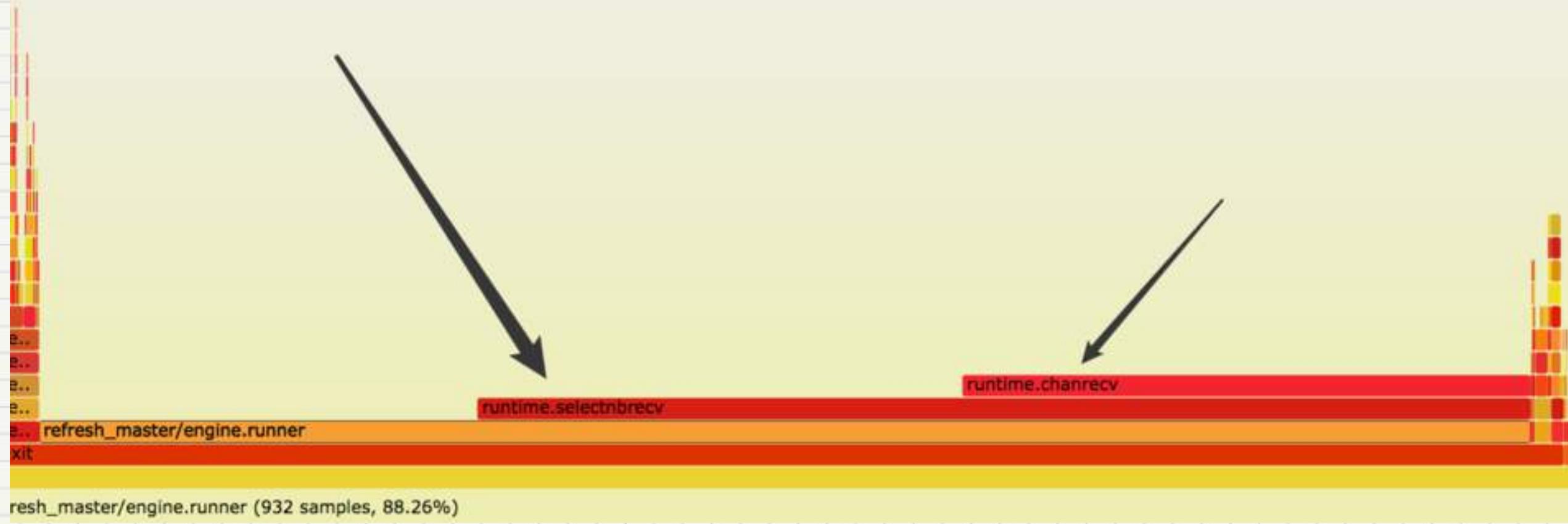
```
...
for {
    select {
        case res = <-dataQueue:
            return res
        case <-done:
            return res
        default:
            // do something
    }
    return res
}
```

Samples: 78K of event 'cpu-clock', Event count (approx.): 12156151512			
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%



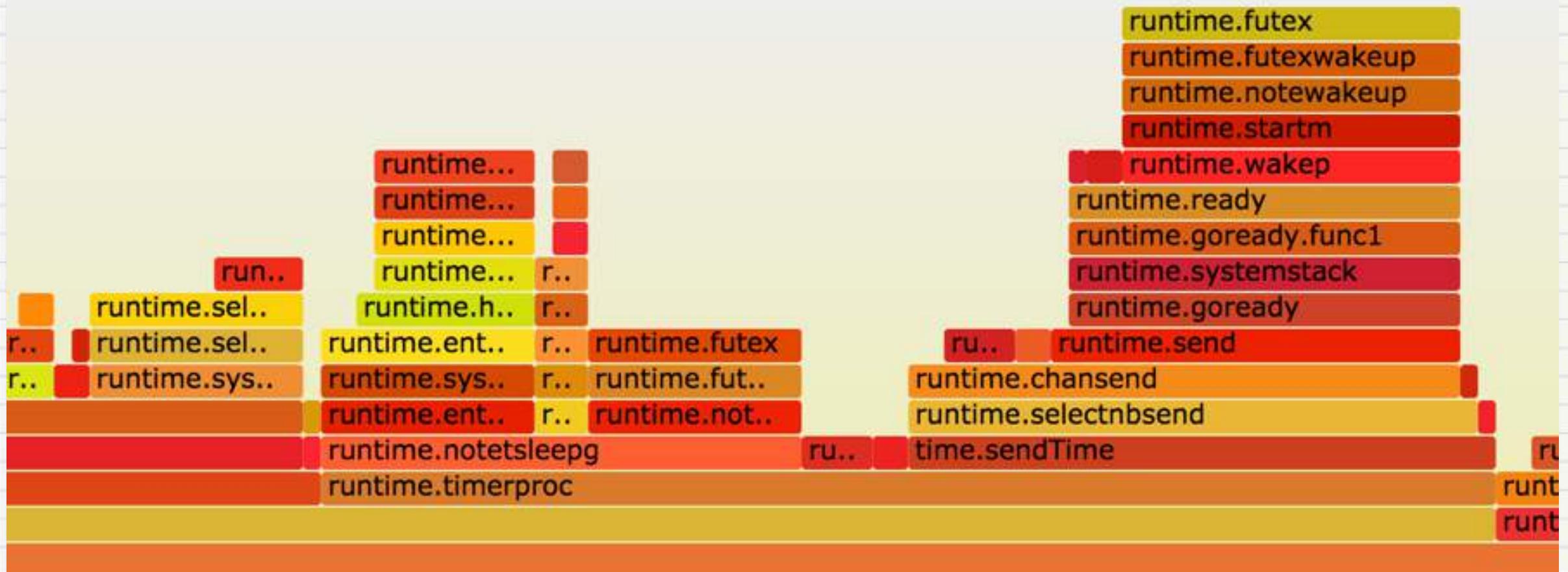
忙轮询





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

Process 27585 detached

% 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

```
Process 25192 attached
^CProcess 25192 detached
% time      seconds   usecs/call  calls  errors syscall
-----+-----+-----+-----+-----+
 94.60    2.663647      50     53762    53762 connect
  2.37    0.066621      1     106872           writev
  1.89    0.053351      1     36105           epoll_wait
  0.47    0.013320      0     54968           close
  0.21    0.005802      0     108200          write
  0.17    0.004844      0     53762           socket
  0.10    0.002934      0     41598           readv
  0.10    0.002824      0     106872          recvfrom
  0.03    0.000894      0     55361           epoll_ctl
  0.02    0.000668      0     53762           ioctl
  0.02    0.000652      0     53102           getsockopt
  0.00    0.000120      0     36105           gettimeofday
  0.00    0.000000      0      660            open
  0.00    0.000000      0      660            fstat
  0.00    0.000000      0      660            sendfile
  0.00    0.000000      0     237            setsockopt
  0.00    0.000000      0       7            futex
  0.00    0.000000      0     146            accept4
-----+-----+-----+-----+-----+
100.00   2.815677    762839    53763 total
                                     9      559      0
                                     INET     563      563      0
                                     FRAG      0       0       0
```

```
11:22:53.342151 IP localhost.46953 > localhost.webcache: Flags [P.], seq 1:122, ack 1, win 512, length 121
0x0000: 4500 00a1 4b78 4000 4006 f0dc 7f00 0001 E...Kx@.0.....
0x0010: 7f00 0001 b769 1f90 2b66 7e48 6fe7 1309 .....i..+zH0...
0x0020: 5818 0200 fe95 0000 4745 5420 2f74 6573 P.....GET./tes
0x0030: 742e 6874 6d6c 2048 5454 502f 312e 300d t.html.HTTP/1.0.
0x0040: 8048 6f73 743a 2077 7777 2e64 6566 6175 P.....GET./tes
0x0050: 6c74 2e63 6f6d 800n 436f 6e6e 6563 7469 Host:.www.defau
0x0060: 6f6e 3a20 636c 6f73 650d 0a55 7365 722d lt.com.Connecti
0x0070: 4167 656e 743a 2072 6566 7265 7368 5f66 on::close..User-
0x0080: 6574 6368 6572 0d0a 4163 6365 7874 2d45 Agent::refresh_
0x0090: 6e63 6f64 696e 673a 2067 7a69 700d 0d0d etcher..Accept-E
0x00a0: 8a ncoding::gzip...
11:22:53.342155 IP localhost.webcache > localhost.46953: Flags [.], ack 122, win 512, length 0
0x0000: 4500 0028 2238 4000 4006 1a96 7f00 0001 E..C*88.0.....
0x0010: 7f00 0001 1f90 b769 6fe7 1309 2b66 7e41 .....io...e.z.
0x0020: 5818 0200 d700 8000 P....p..
11:22:53.342905 IP localhost.webcache > localhost.46945: Flags [P.], seq 1:672, ack 122, win 512, length 671
0x0000: 4500 02c7 521c 4000 4006 e812 7f80 0001 E...R.0.0.....
0x0010: 7f80 0001 1f90 b761 6614 6335 4548 db97 .....af.c5E0..
0x0020: 5818 0200 00bc 0000 4854 5458 2f31 2e30 P.....HTTP/1.0
0x0030: 2832 3030 204f 4b8d 0a53 6572 7665 723a .200.OK..Server:
0x0040: 2041 5453 2f36 2e32 2e33 0d0a 4461 7465 ATS/5.2.3..Date
0x0050: 3a28 5475 652c 2832 3720 4061 7220 3250 ,Tue,,27.Mar.20
0x0060: 3138 2830 333a 3037 3a34 3428 471d 540d 18:05:07:44.GMT.
```

```
jxcdn]# ss -s
(kernel 720)
(estab 426, closed 55911, orphaned 11, synrecv 0, timewait 55907/0), ports 55283
          total      IP          IPv6
                           0       -        -
                           0       0        -
                           4       0        -
                           0       0        -
                           0       0        -
```



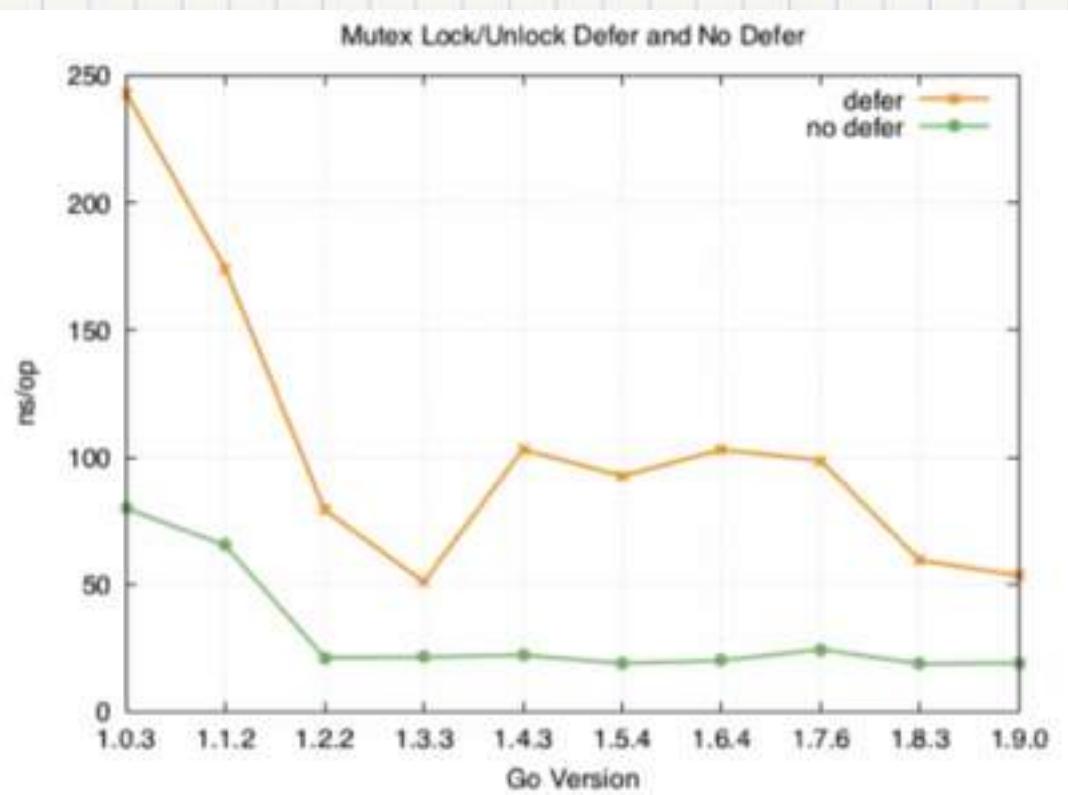
Nginx的time_wait问题

```
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 "";
}
```

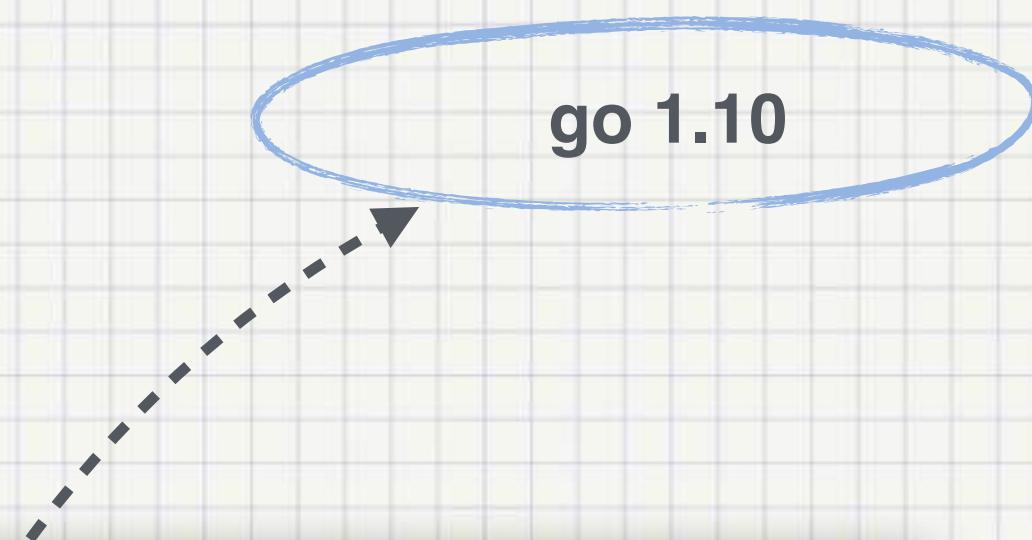


defer的消耗



汇编:

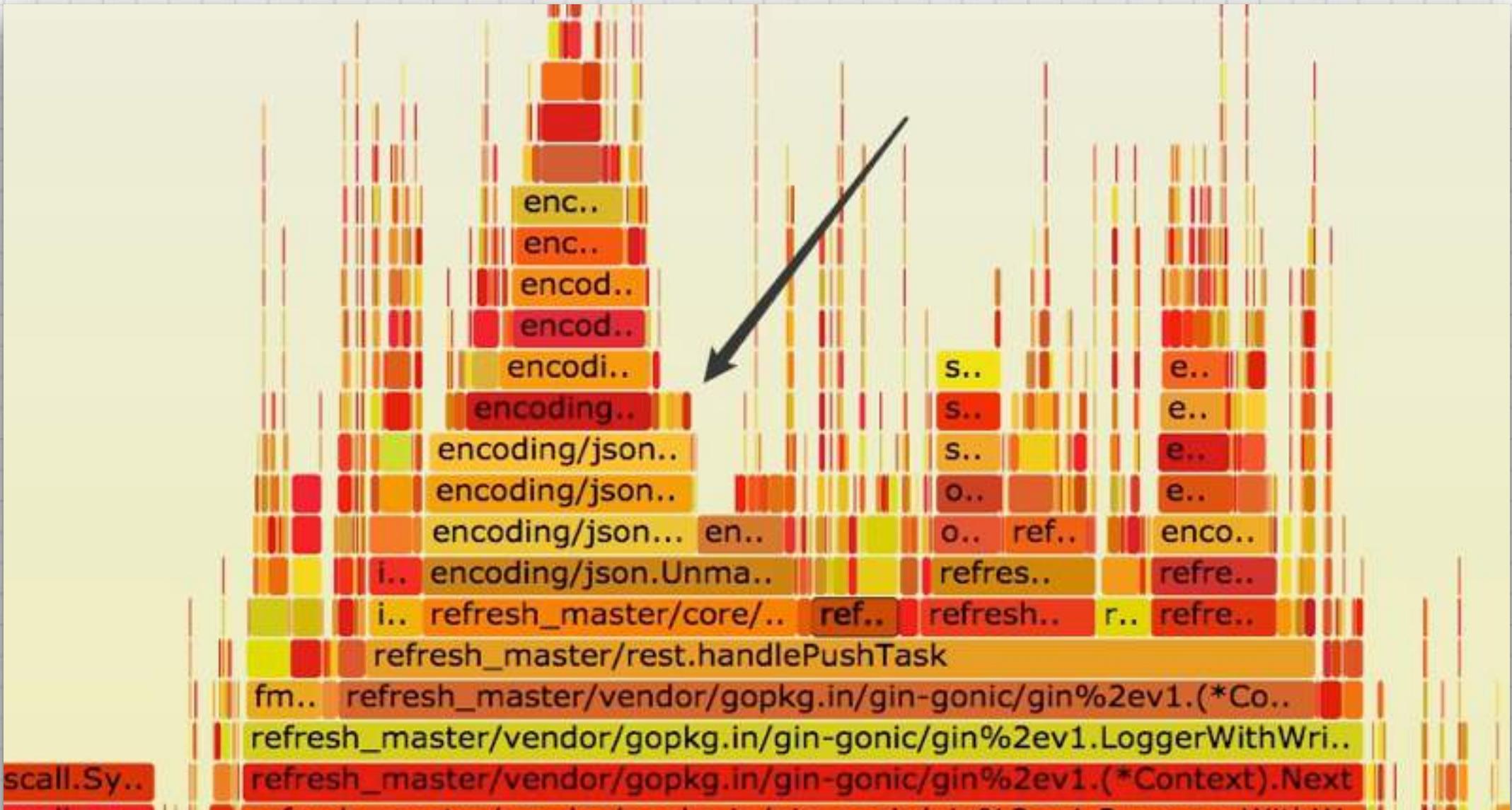
call runtime.deferreturn,
add xx SP
return



```
[ruifengyun@xiaorui locker (engine x)]$ go test -bench=. -benchmem -run=None
goos: darwin
goarch: amd64
pkg: refresh_master/locker
BenchmarkDeferLock-4          3000000000      67.1 ns/op      0 B/op
BenchmarkNoDeferLock-4        10000000000     20.9 ns/op      0 B/op
```



Json性能问题





Json 性能问题

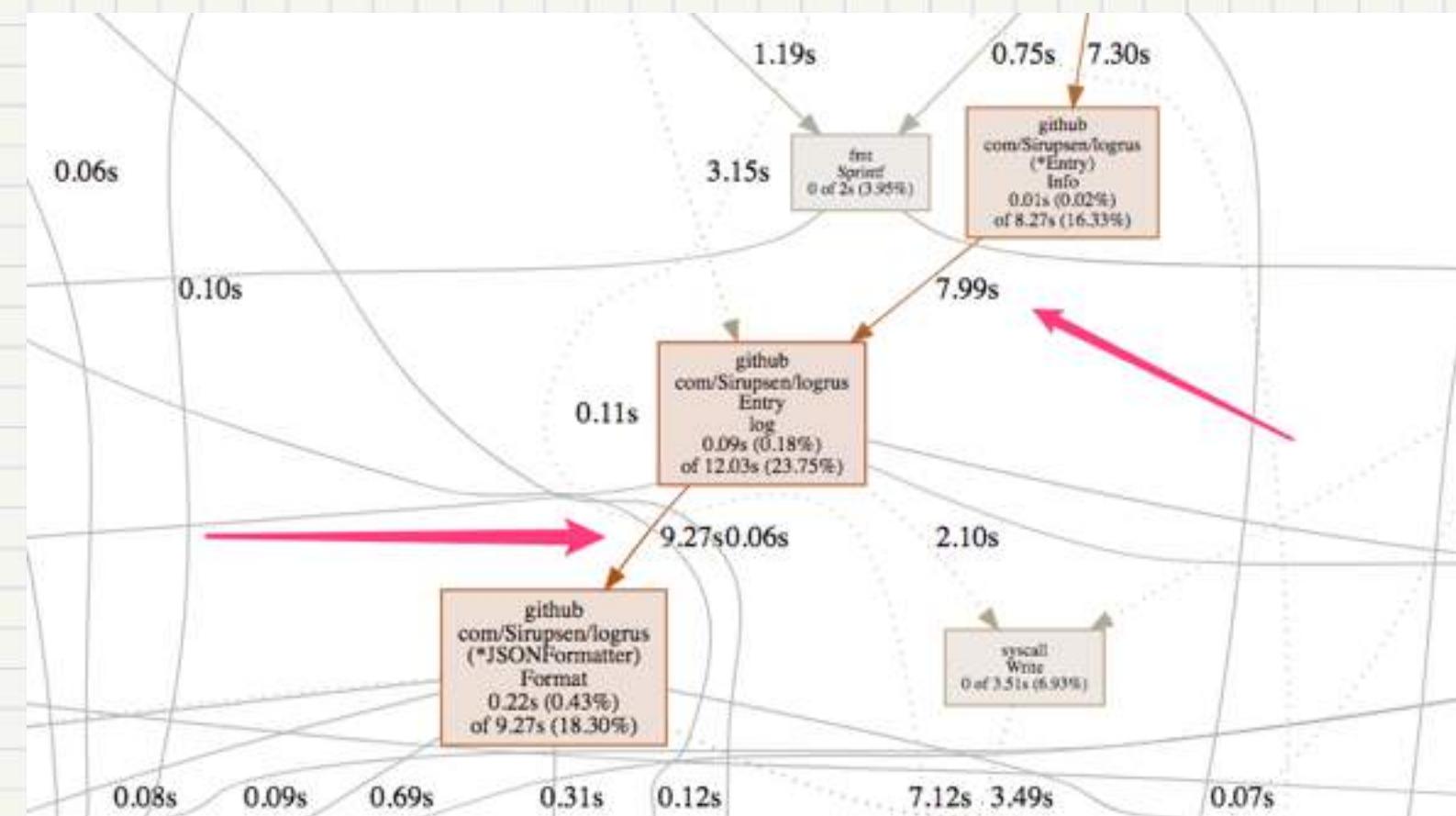
- 固定数据结构
 - *json-iterator/go*
- 不固定数据结构
 - *buger/jsonparser*

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)

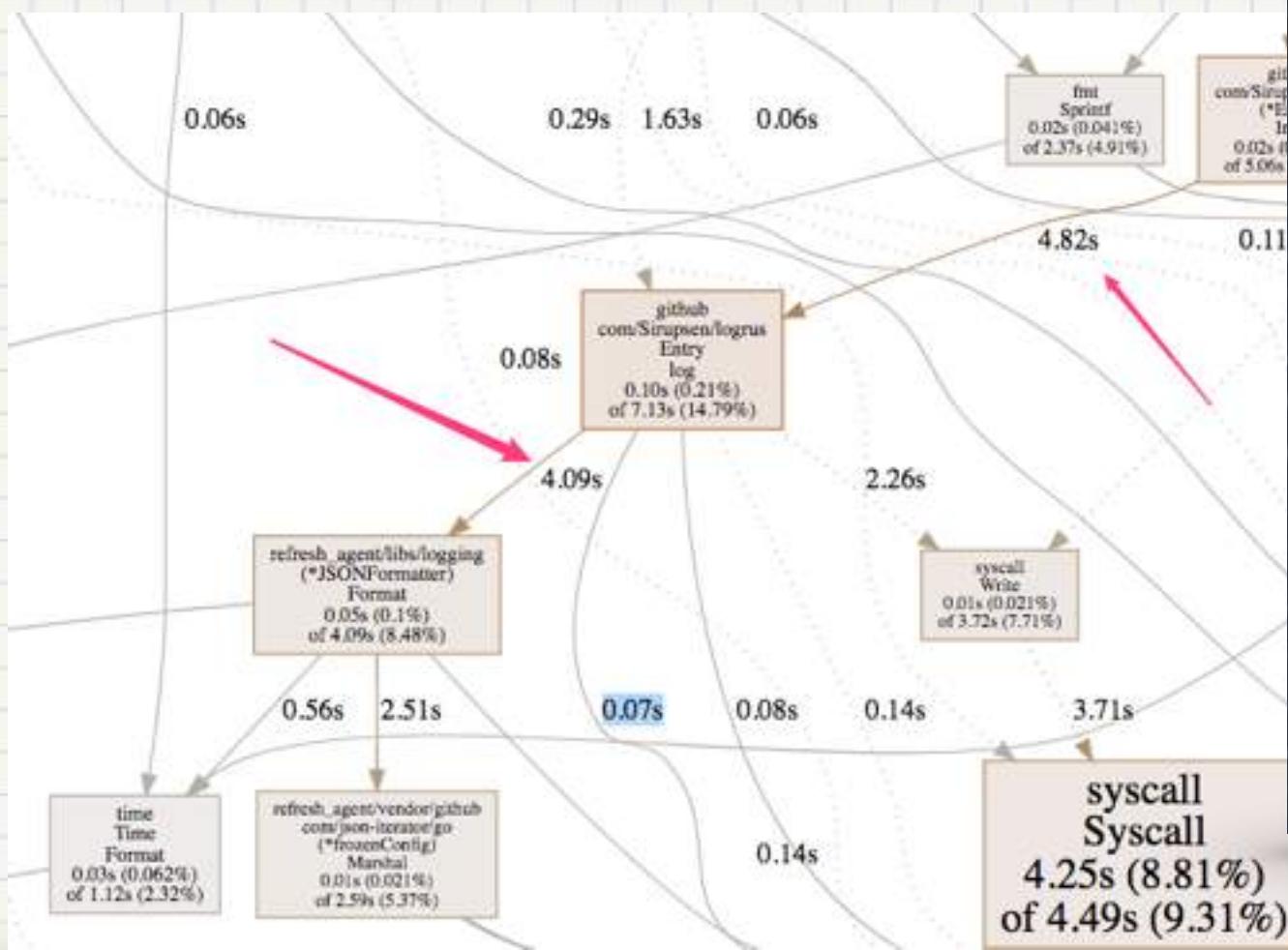


struct > map

cost 9.27s



logrus json (map vs struct)



```
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"`
}

selectgo
0.01s(0.0239
of 1.23s (2.55
```

cost 4.09s



GC 耗时调优

```
scvg3: inuse: 1653, idle: 73, sys: 1727, released: 0, consumed: 1727 (MB)
GC forced slight incident may touch omain
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
```

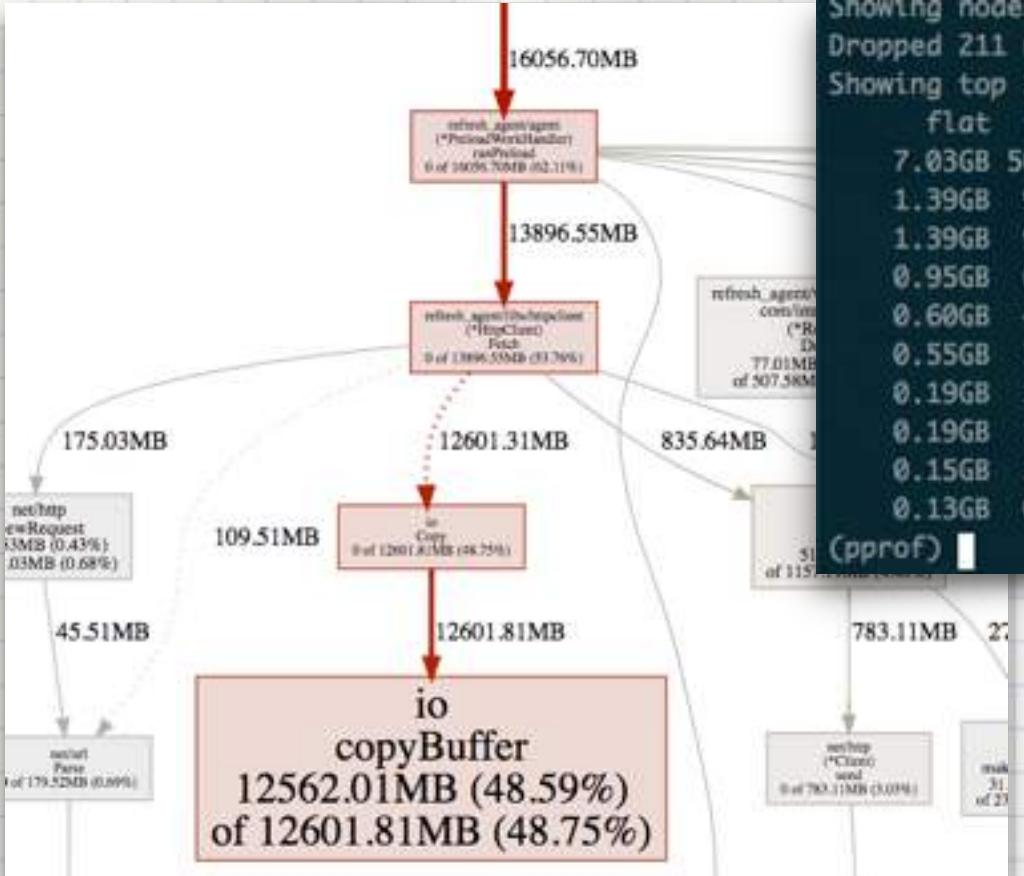
GODEBUG=gctrace=1

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

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

```
//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
}
```

```
// 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

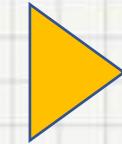
go 1.10 > go 1.9 > go 1.8

sync.Pool 效果明显

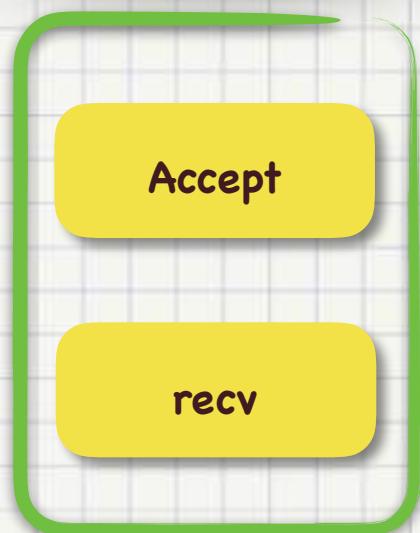
主动介入runtime调度调度



kernel



- 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

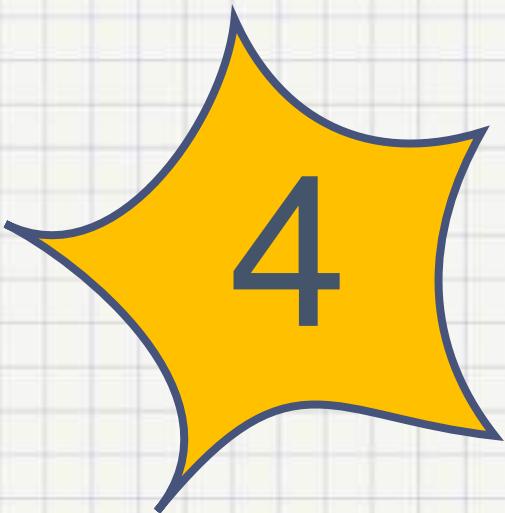
50%	17.90ms
75%	24.92ms
90%	34.63ms

...

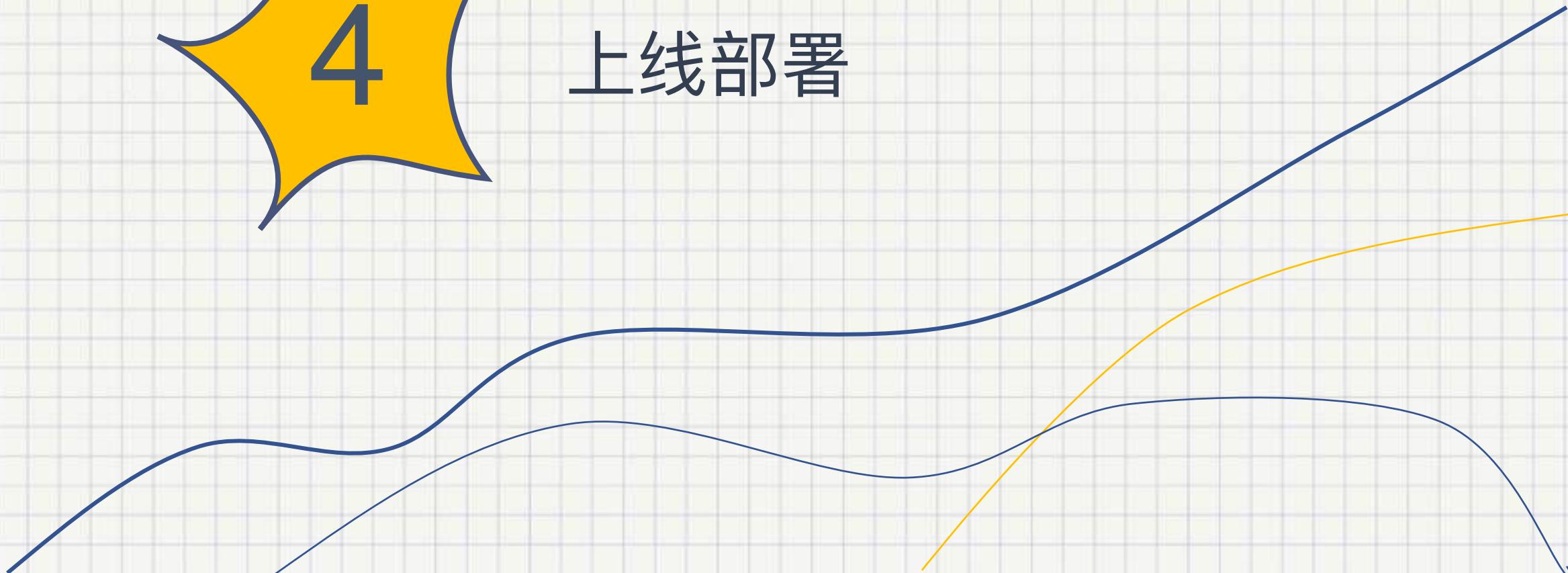
Requests/sec: 23189.11

Transfer/sec: 5.06MB





上线部署





打包方法



Engine端

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

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



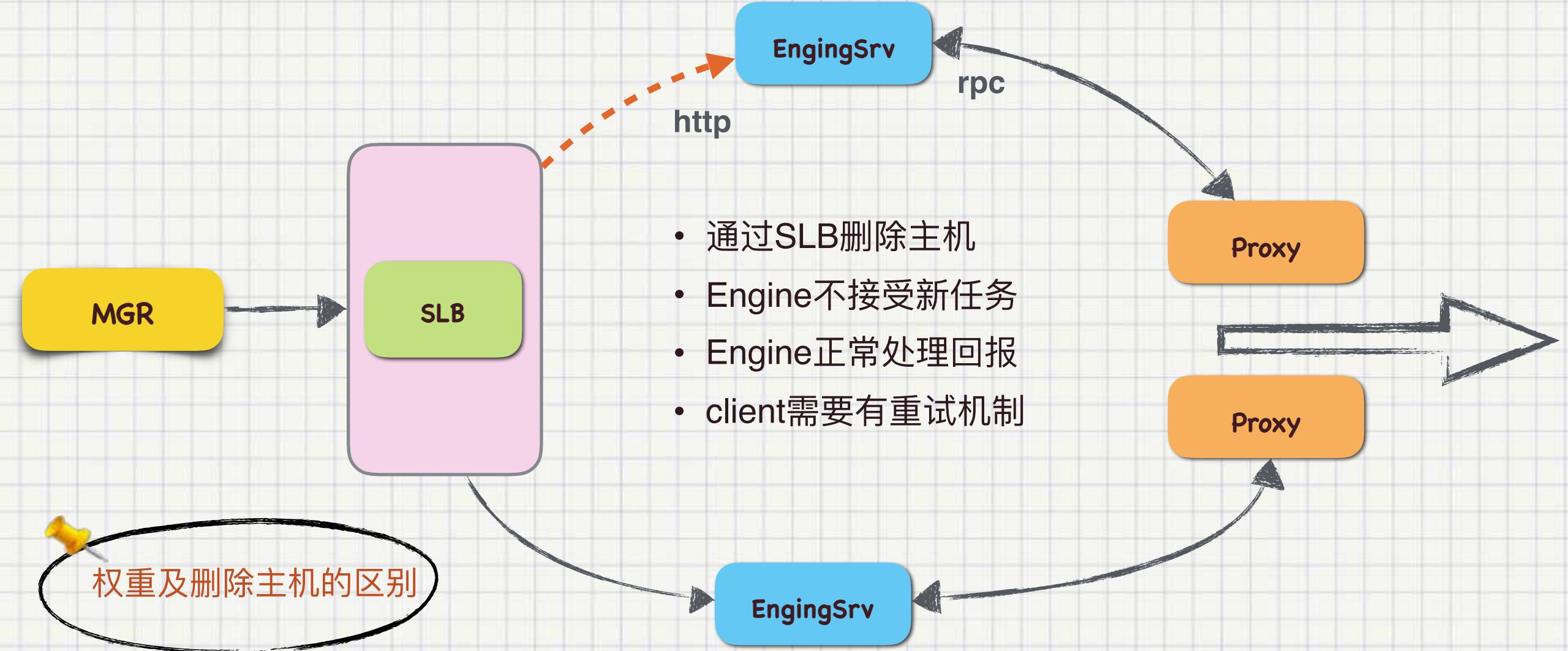
Proxy端, 封装RPM包上线 .

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

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

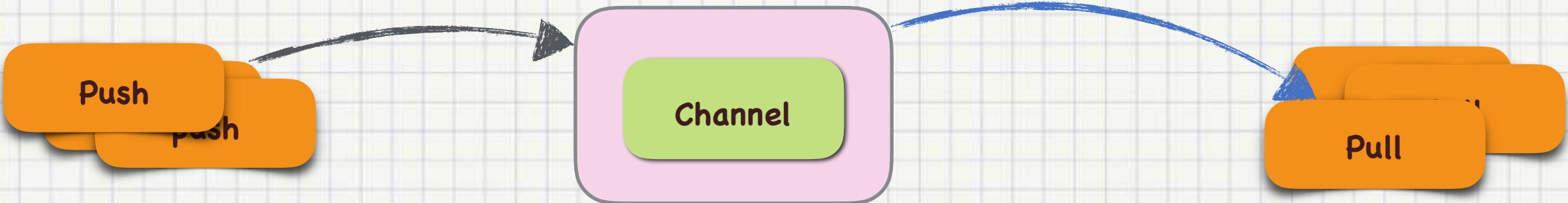


灰度平滑迁移-SLB层





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.

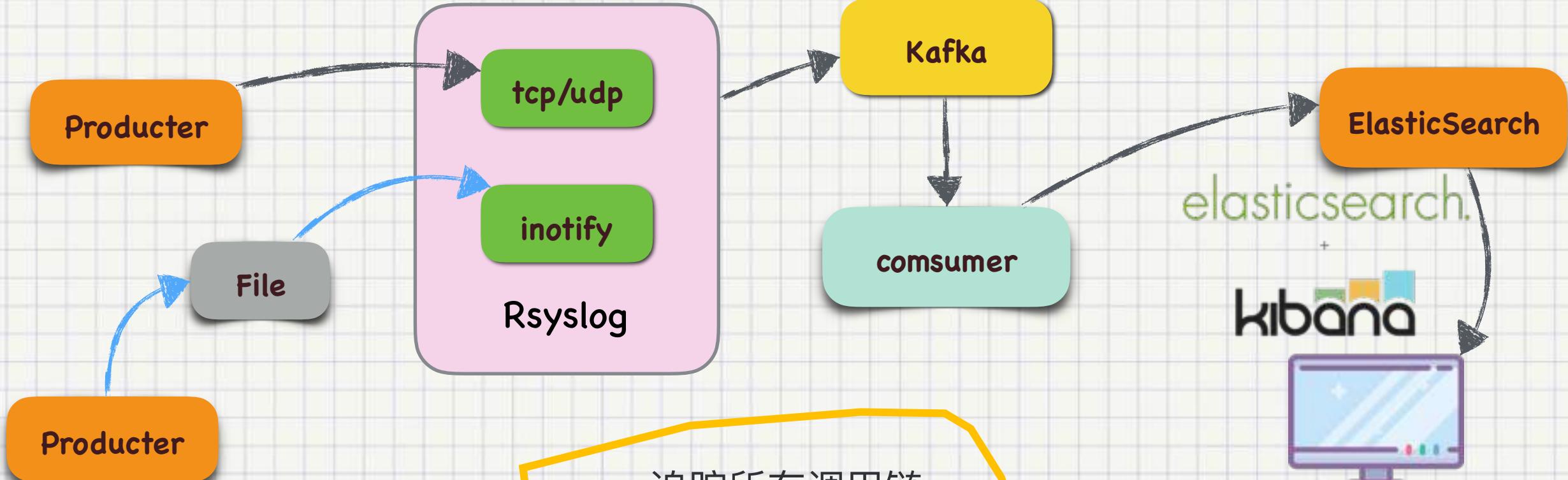
伪代码

```
recvAndSetExit(signal)  
publishSigWorker()
```

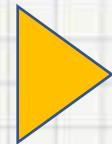
```
if globalExited && len(queue) == 0 && pushExit {  
    return  
}  
wait(saveQueueExit)  
wait(eventBus)
```



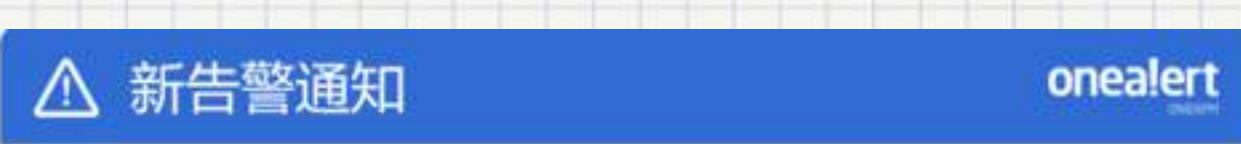
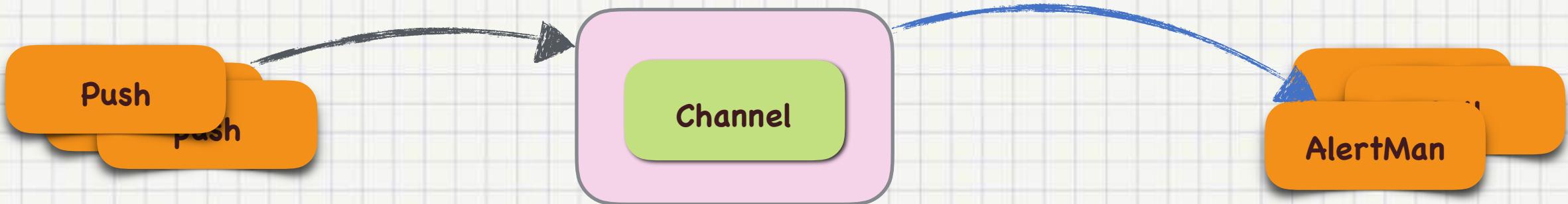
收集分布式日志



- 追踪所有调用链
- 计算耗时
- 排查问题



如何报警通知



提醒 : refresh:engine loss

83

告警对象:

告警内容: {"extra_info":"post to engine failed"}

告警编号: 420

发生时间: 11:04

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
rester	✓	2	2018-03-23 14:29:13
scancer	✓	1	2018-03-23 14:29:14
...



节点心跳

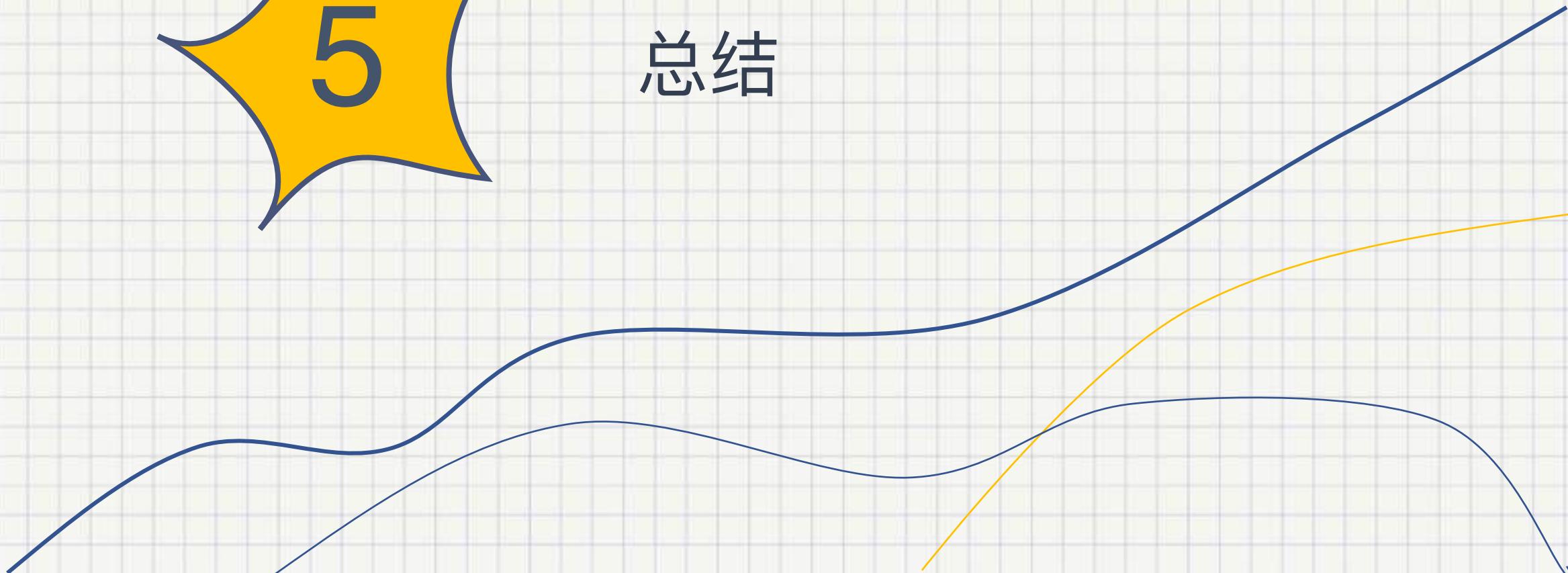
Module	Status	Diff	LastTime
node1	✓	0	2018-0
node2	✓	0	2018-0
node3	✓	0	2018-0
node4	✓	0	2018-0
node5	✓	2	2018-0
node6	---	597	2018-0
node7	---	597	2018-0
node8	---	597	2018-0
node9	---	597	2018-0

API_ID计数器

API_ID	INIT	PENDING	RUNNING
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0

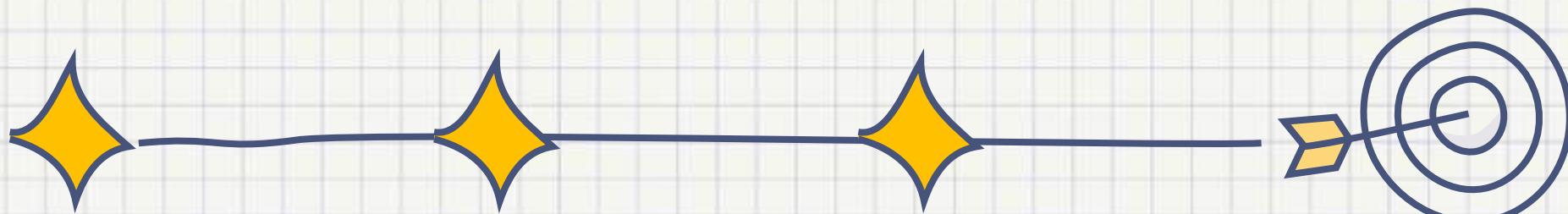


总结





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
0.81%	engine_bin	[.] runtime.gcWriteBarrier



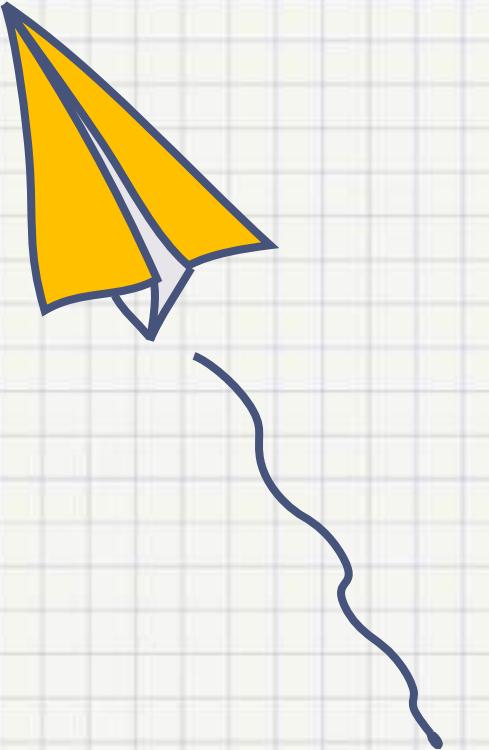
协程太多

profiles:

0	block
119865	goroutine
1231	heap
0	mutex
12	threadcreate

[full goroutine stack dump](#)





Q & A

