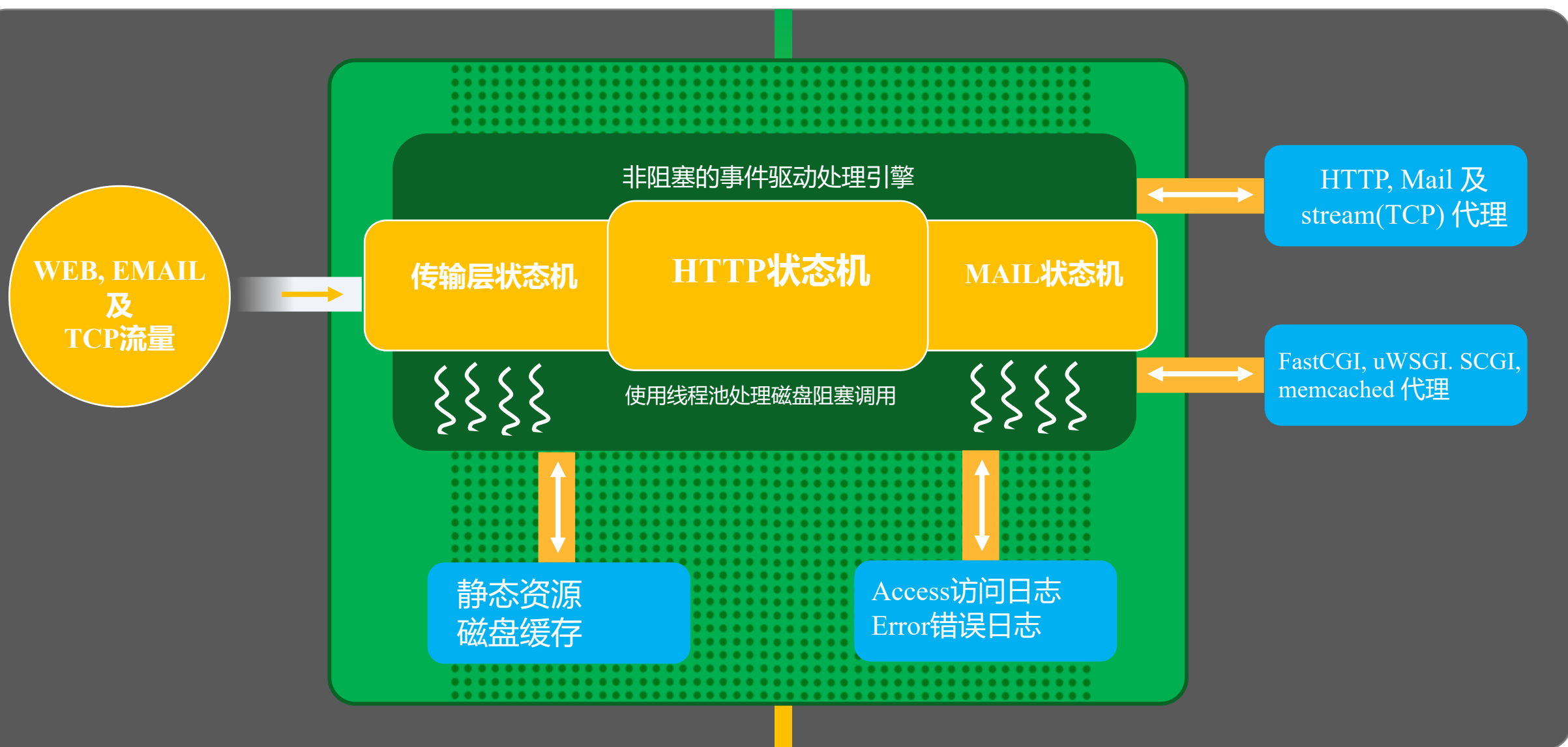


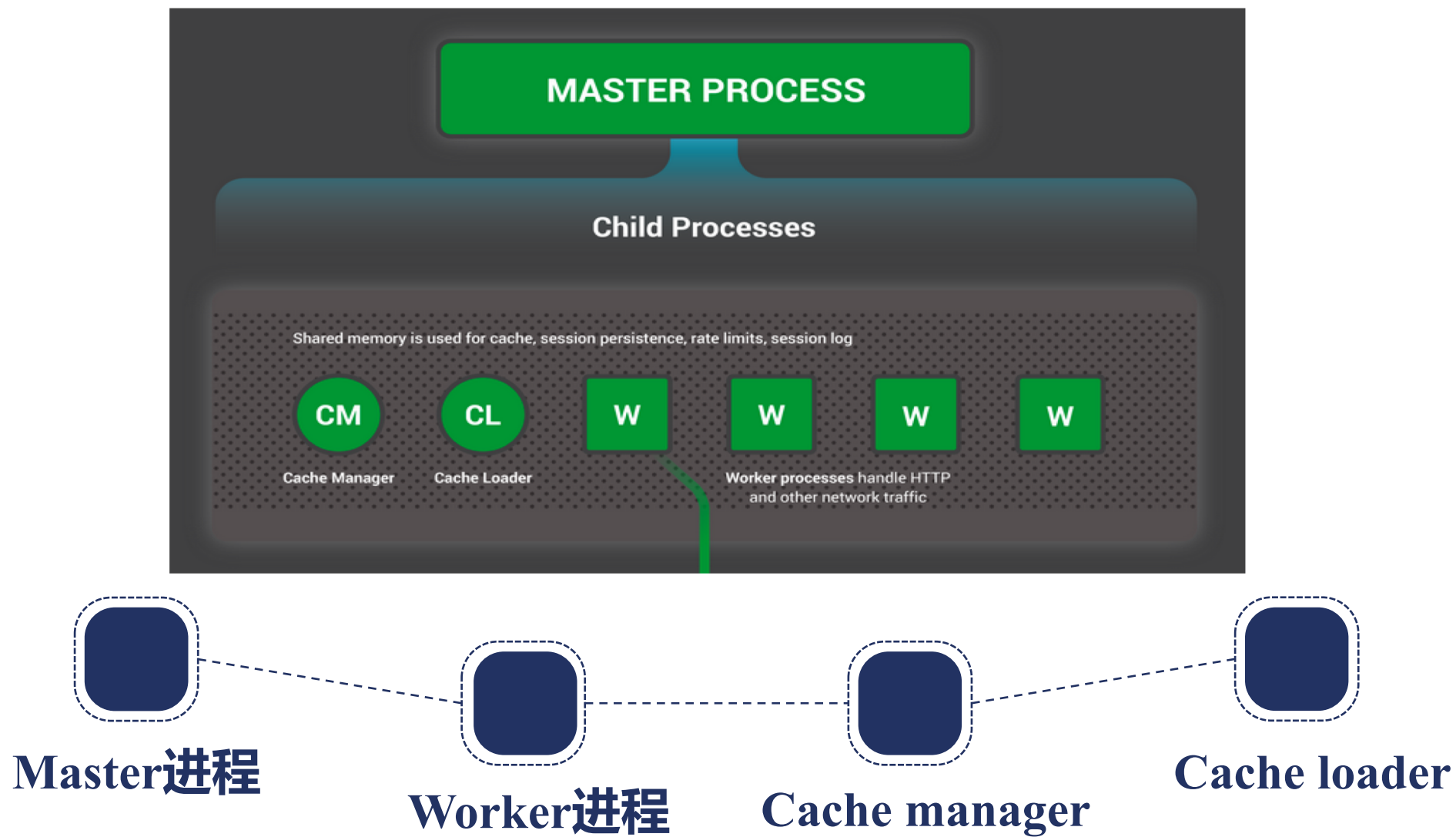


Ngix 架构基础

Nginx请求处理流程



Nginx进程结构



Nginx进程管理：信号

Master进程

- 监控worker进程
 - CHLD
- 管理worker进程
- 接收信号
 - TERM, INT
 - QUIT
 - HUP
 - USR1
 - **USR2**
 - **WINCH**

Worker进程

- 接收信号
 - TERM, INT
 - QUIT
 - USR1
 - WINCH

nginx命令行

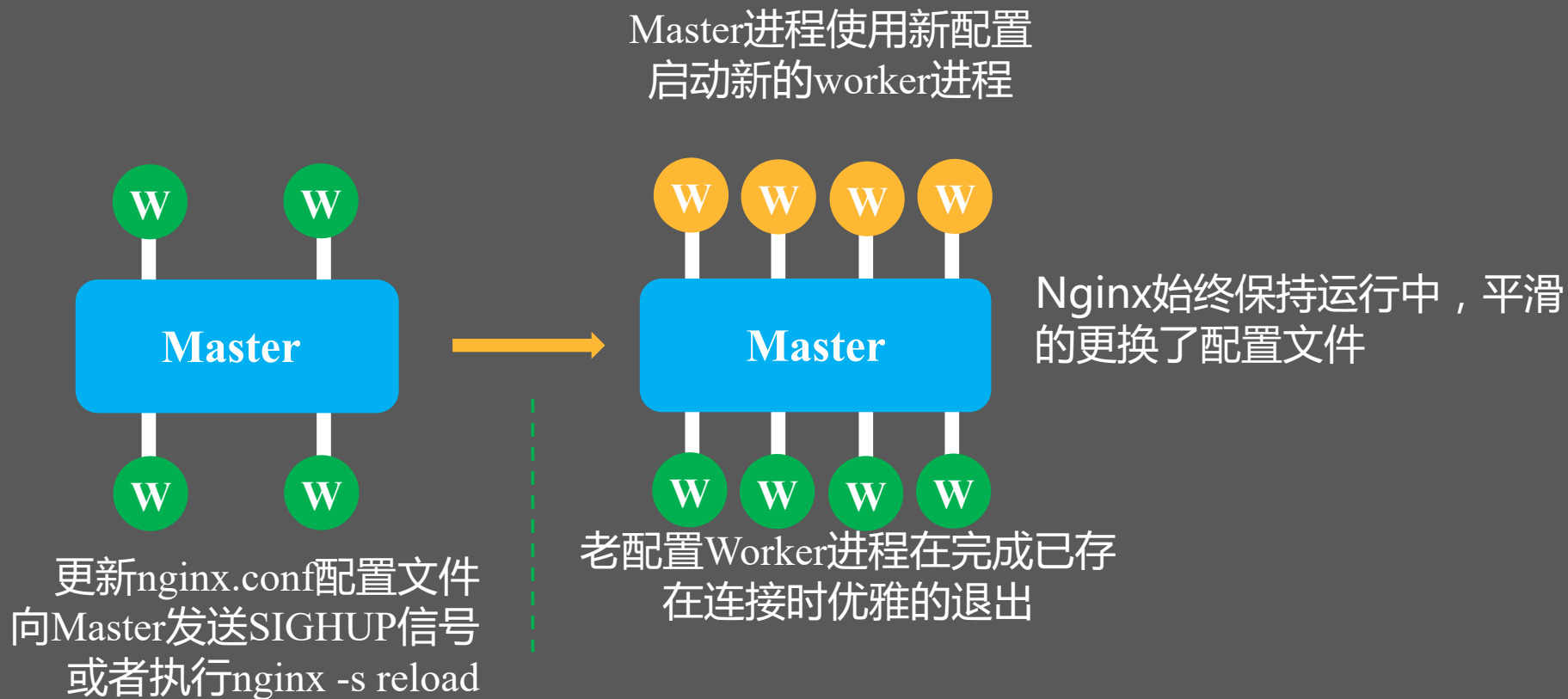
- reload : HUP
- reopen : USR1
- stop : TERM
- quit : QUIT

reload流程

- 01 向master进程发送HUP信号（ reload命令 ）
- 02 master进程校验配置语法是否正确
- 03 master进程打开新的监听端口
- 04 master进程用新配置启动新的worker子进程
- 05 master进程向老worker子进程发送QUIT信号
- 06 老worker进程关闭监听句柄，处理完当前连接后结束进程

reload流程

不停机载入新配置

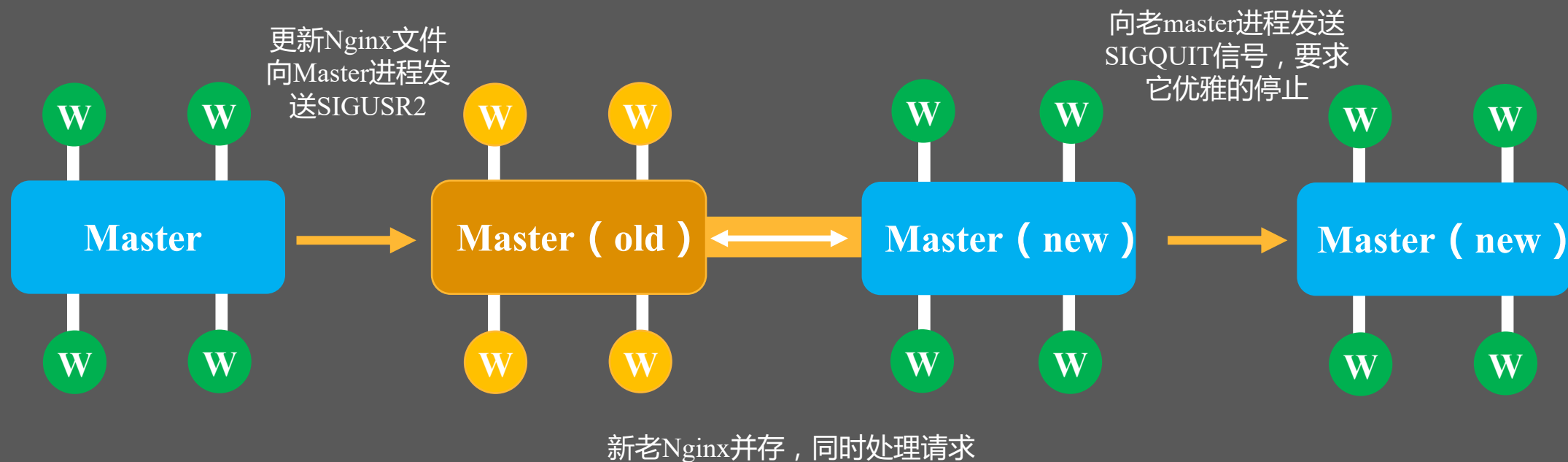


热升级流程

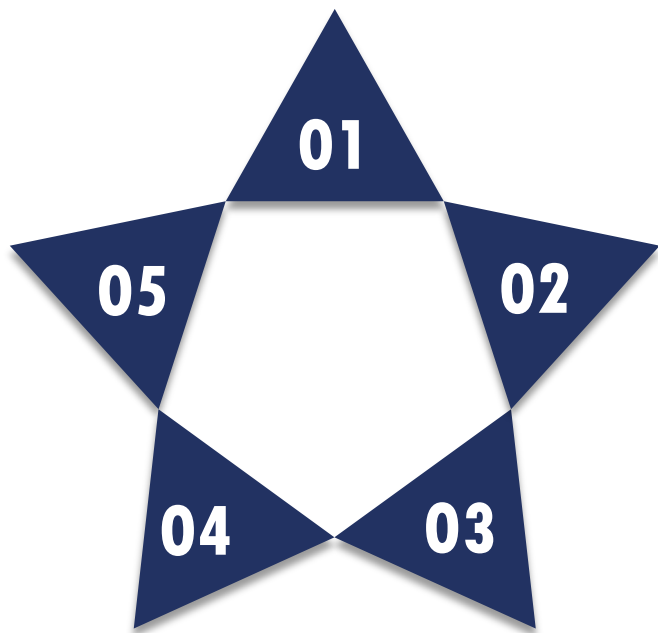
- 01 将旧Nginx文件换成新Nginx文件（注意备份）
- 02 向master进程发送USR2信号
- 03 master进程修改pid文件名，加后缀.oldbin
- 04 master进程用新Nginx文件启动新master进程
- 05 向老master进程发送WINCH信号，关闭老worker
- 06 回滚：向老master发送HUP，向新master发送QUIT

热升级流程

不停机更新Nginx二进制文件



worker进程：优雅的关闭



01 设置定时器

`worker_shutdown_timeout`

02 关闭监听句柄

03 关闭空闲连接

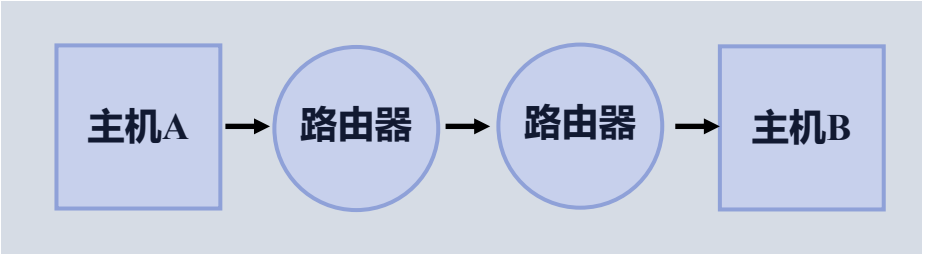
04 在循环中等待全部连接关闭

05 退出进程

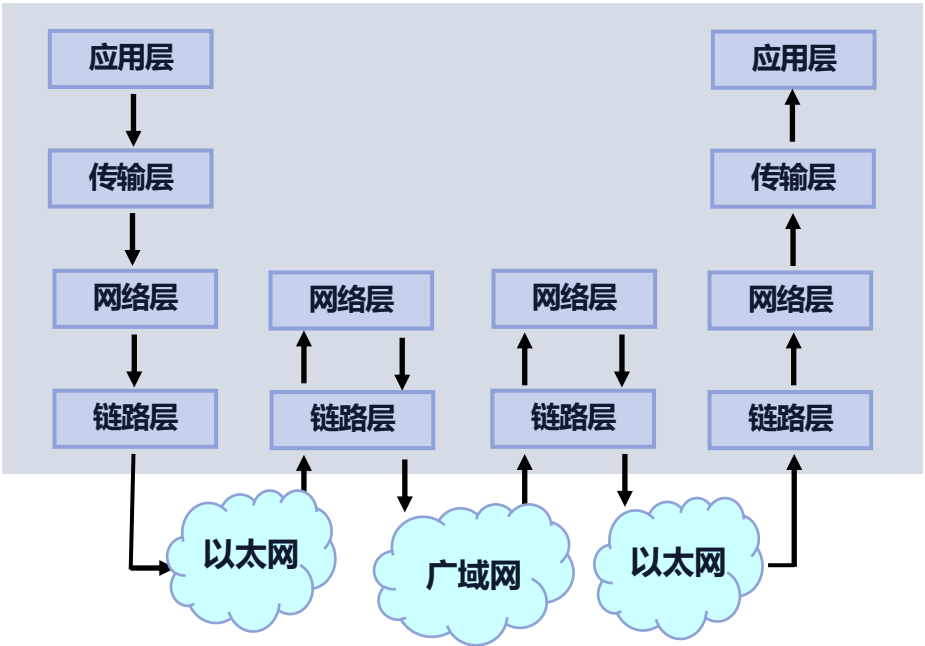
网络传输



网络拓扑

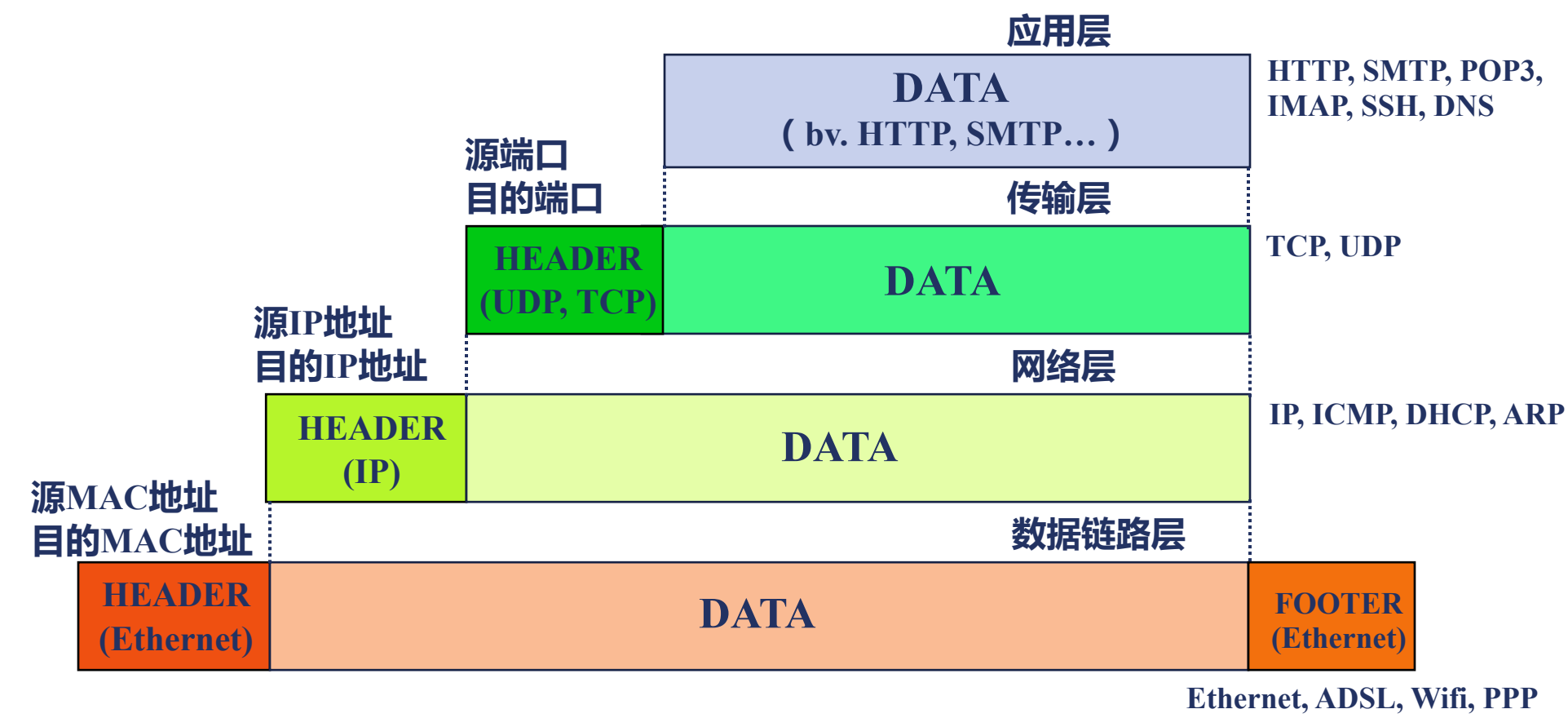


数据流

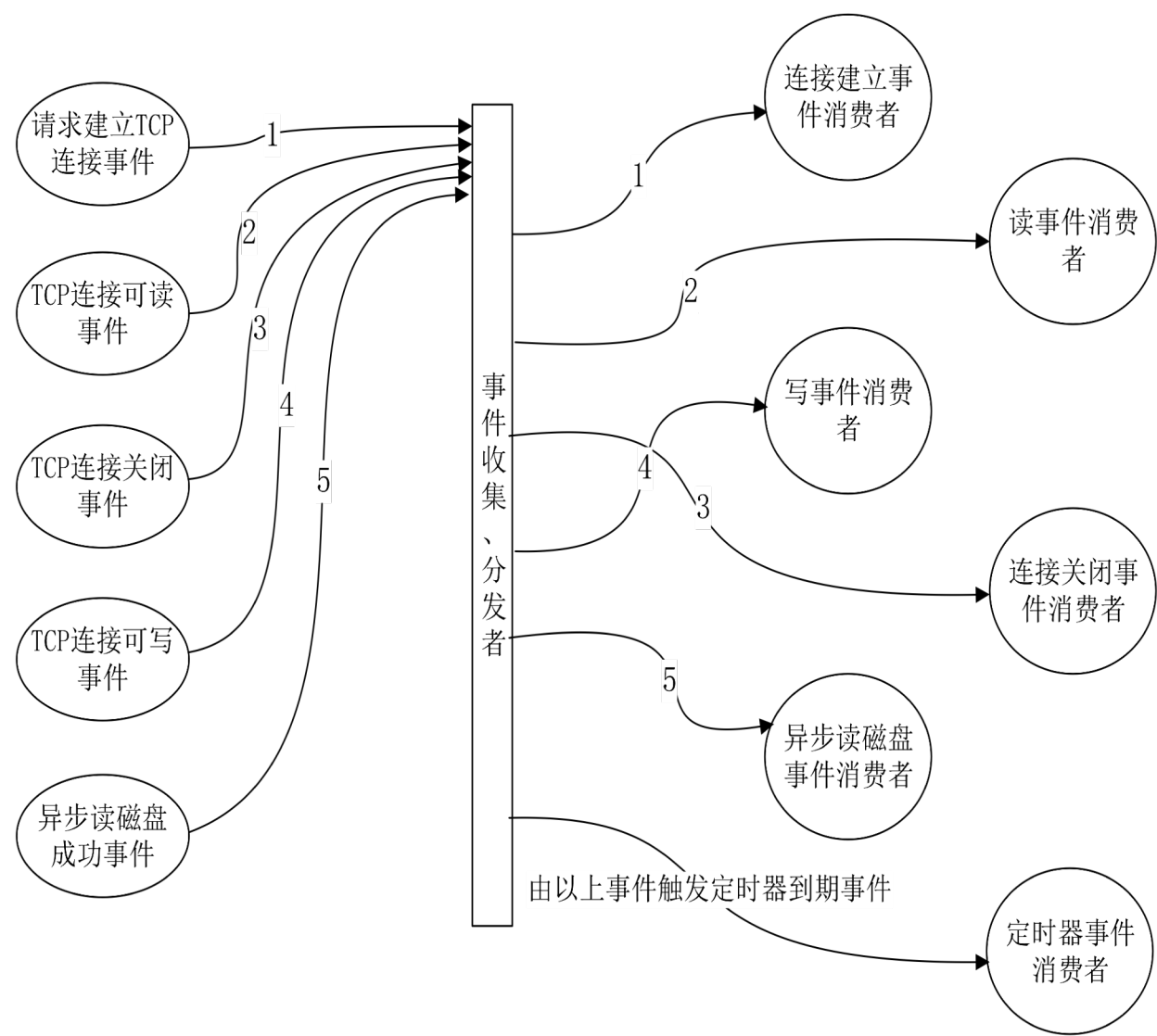


TCP流与报文

TCP/IP协议层级



TCP协议与非阻塞接口



读事件

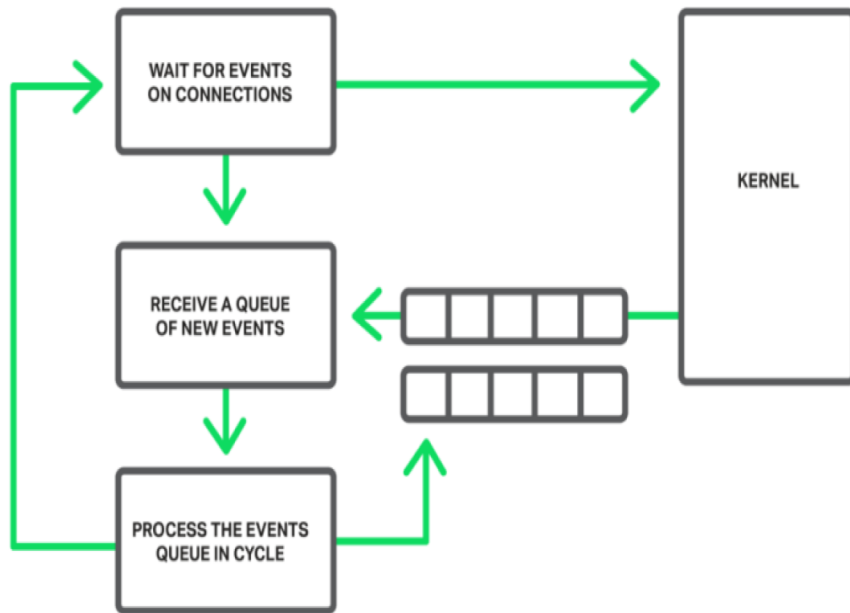
- Accept建立连接
- Read读消息

写事件

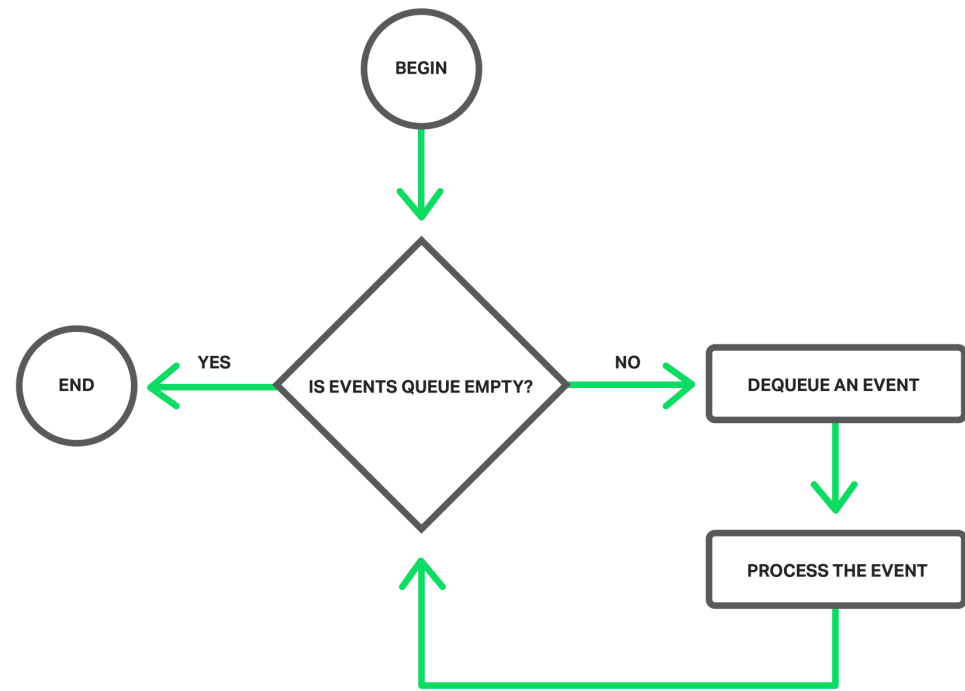
- Write写消息

Nginx事件循环

NGINX EVENT LOOP

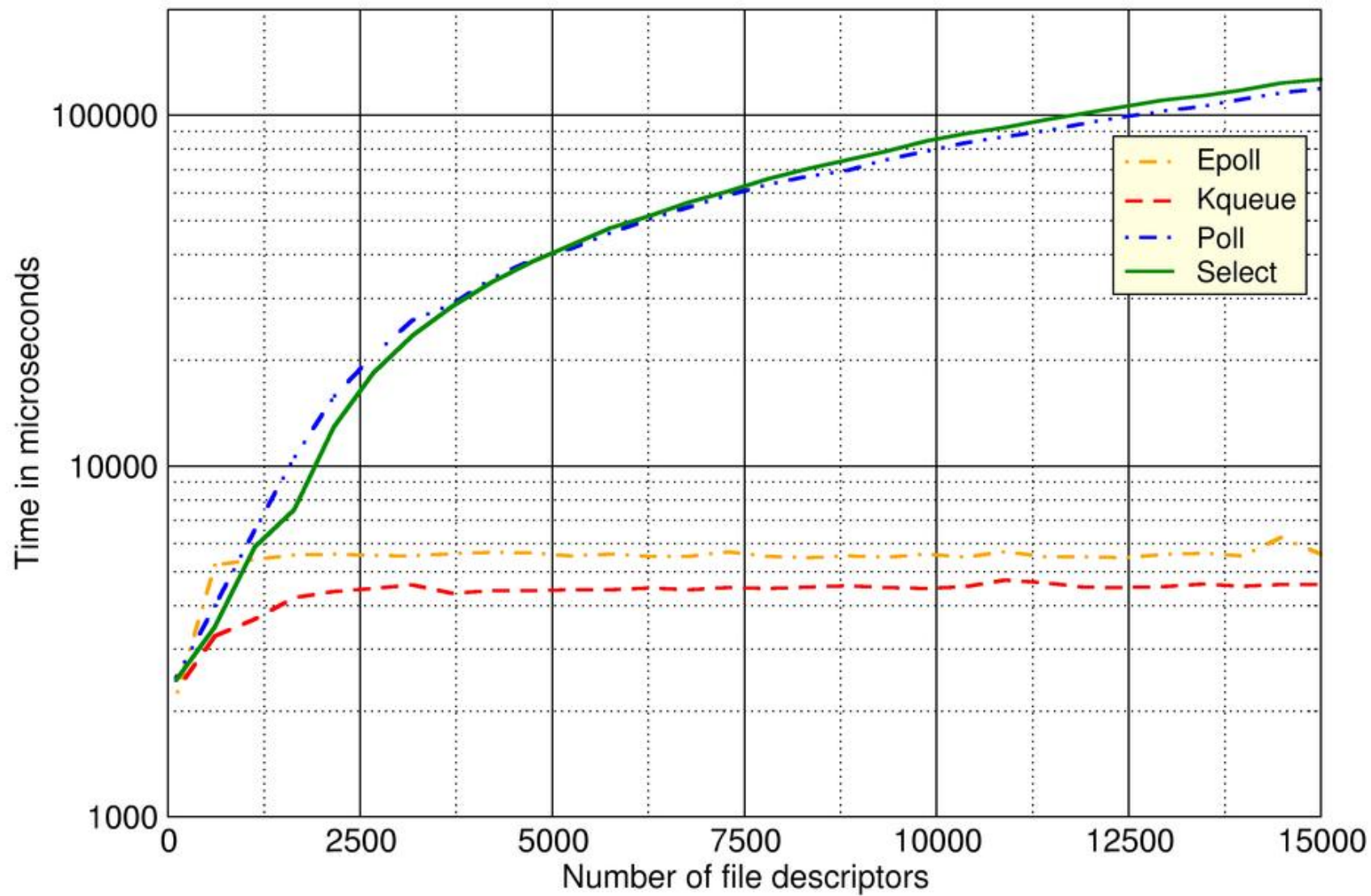


THE EVENTS QUEUE PROCESSING CYCLE



Libevent Benchmark

100 Active Connections and 1000 Writes



epoll

前提

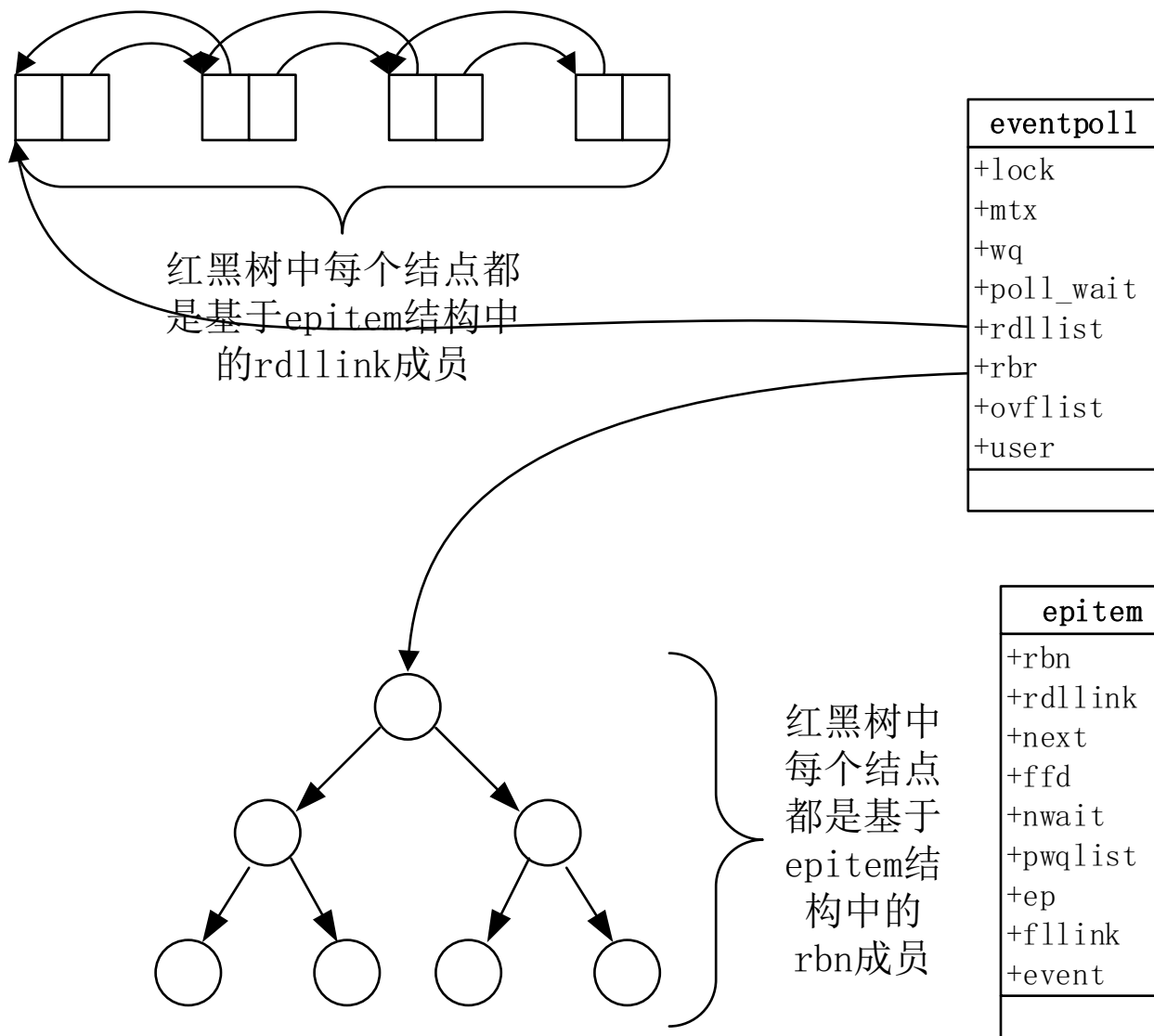
- 高并发连接中，每次处理的活跃连接数量占比很小

实现

- 红黑树
- 链表

使用

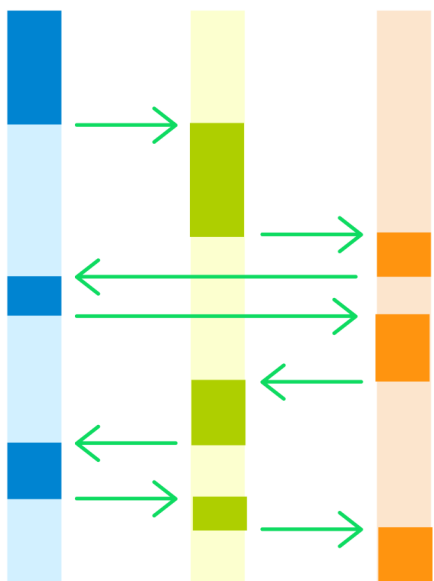
- 创建
- 操作：添加/修改/删除
- 获取句柄
- 关闭



请求切换

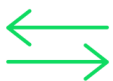
TRADITIONAL SERVER

PROCESS 1 PROCESS 2 PROCESS 3



NGINX WORKER

PROCESS



TASK SWITCHES



PROCESSING REQUEST 1



PROCESSING REQUEST 2



PROCESSING REQUEST 3

一线程仅处理一连接

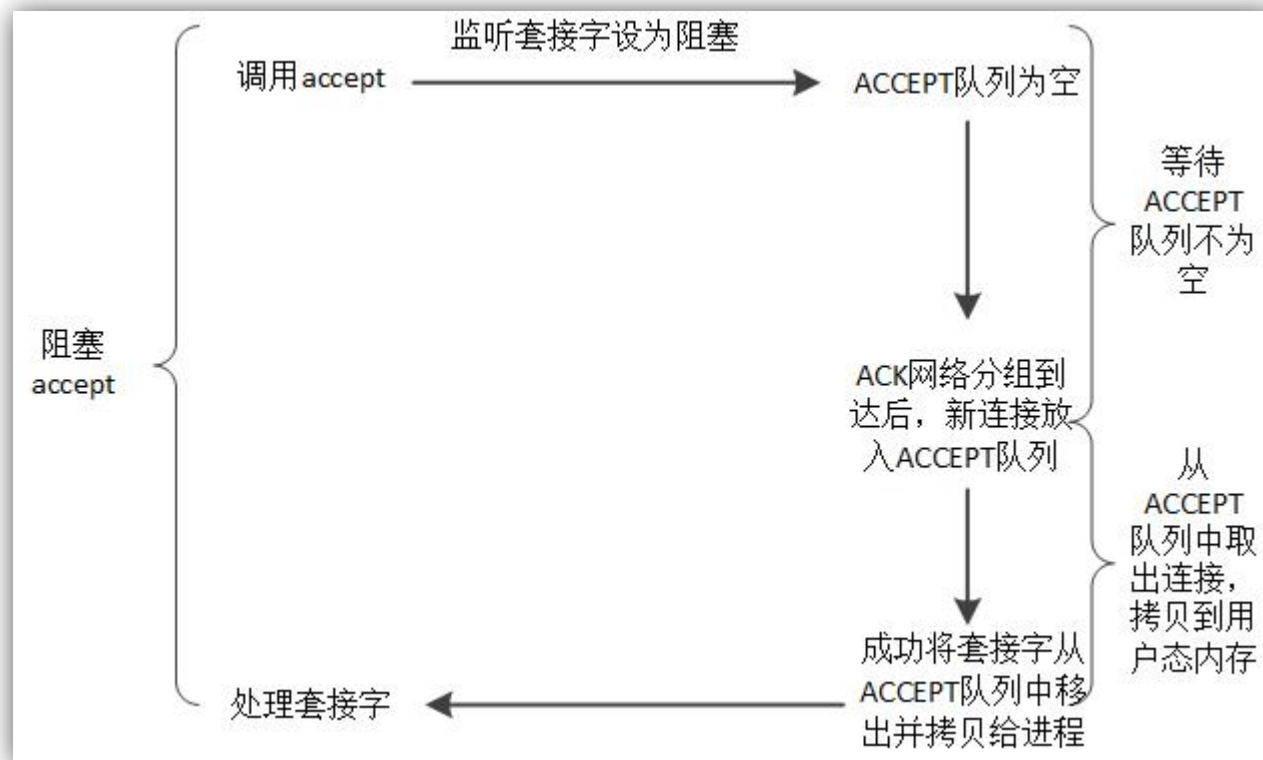
- 不做连接切换
- 依赖OS的进程调度实现并发

一线程同时处理多连接

- 用户态代码完成连接切换
- 尽量减少OS进程切换

阻塞调用

以accept为例



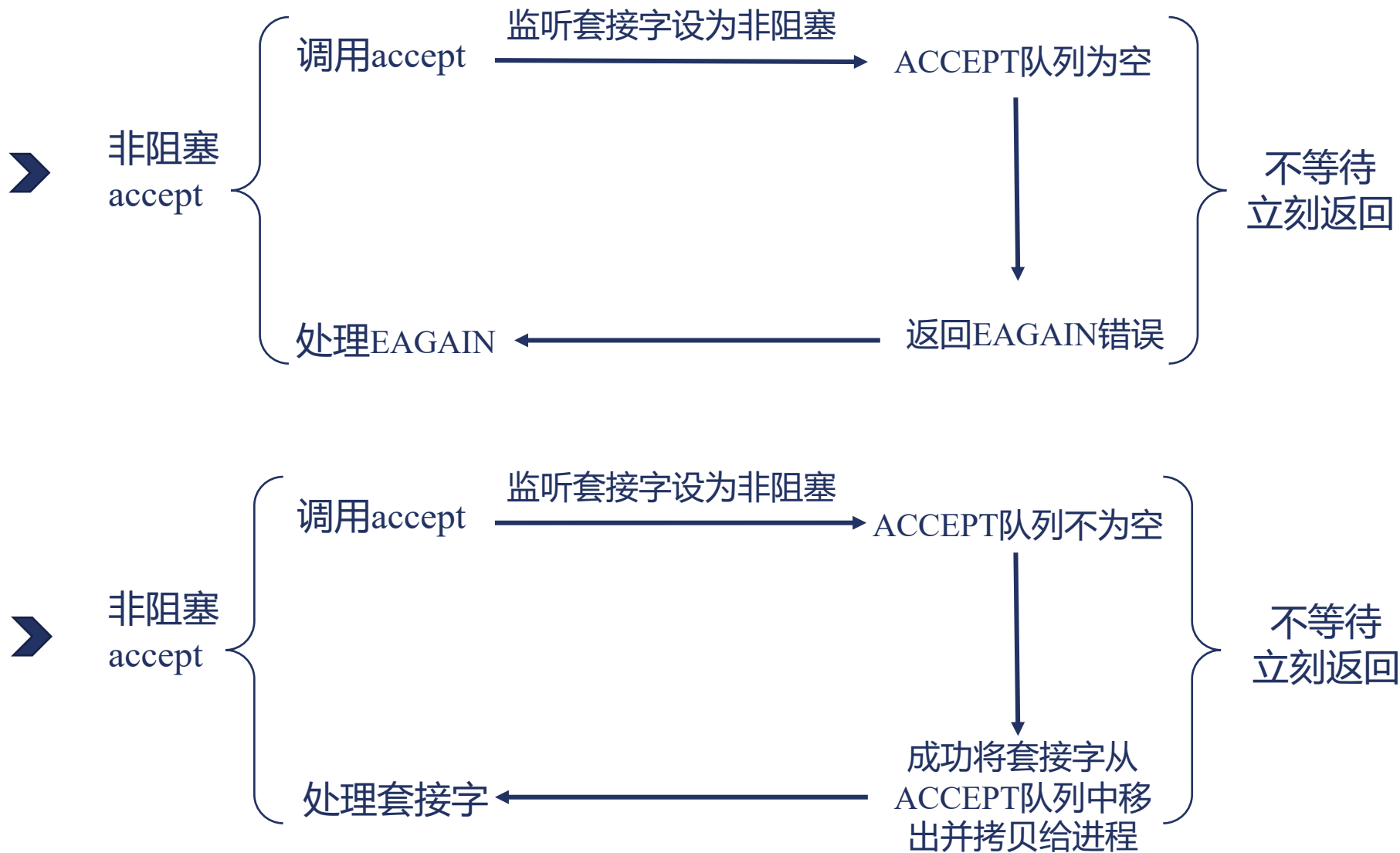
**BLOCKING
OPERATION**



产生进程主动切换

非阻塞调用

由你的代码决定是否切换新任务



非阻塞调用下的同步与异步

Openresty的同步调用代码

```
local client = redis:new()
client:set_timeout(30000)
local ok,err = client:connect(ip,port)
if not ok then
    ngx.say("failed: ",err)
    return
end
```

这是标准的异步调用

```
rc = ngx_http_read_request_body ( r, ngx_http_upstream_init) ;
if ( rc >= NGX_HTTP_SPECIAL_RESPONSE) {
    return rc ;
}
```

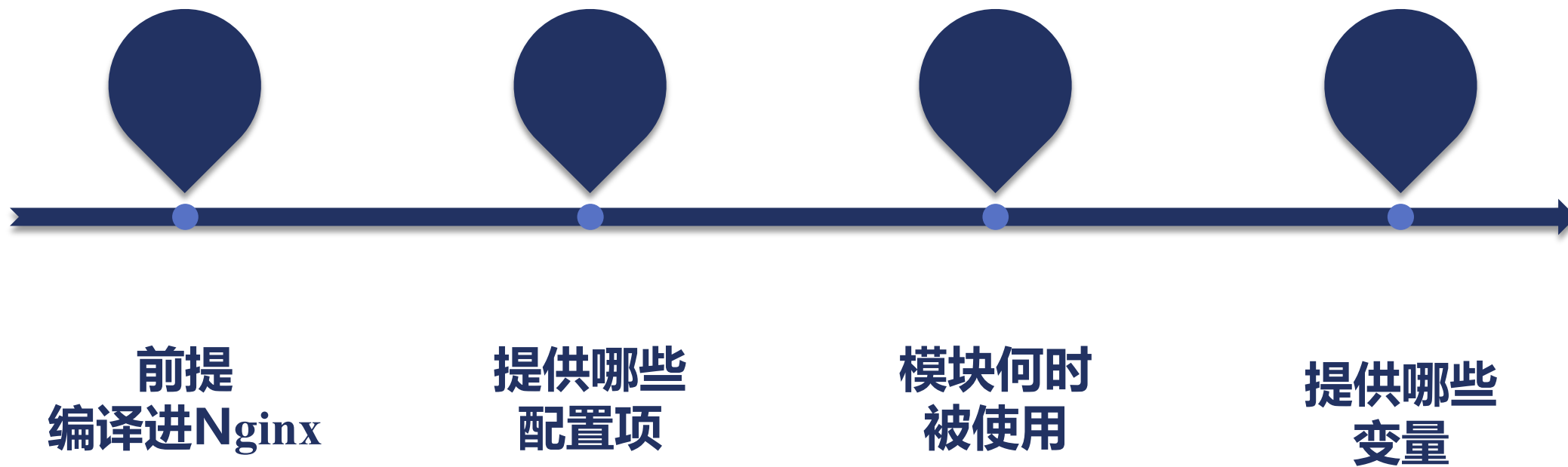
这个方法执行完时调用post_handler异步方法

```
ngx_int_t
ngx_http_read_client_request_body (ngx_http_request_t * r,
    ngx_http_client_body_handler_pt post_handler)
```

最终读取完body后调用 ngx_http_upstream_init方法

```
void
ngx_http_upstream_init (ngx_http_request_t * r)
{
```

Nginx 模块



Nginx 模块

内聚

抽象

- 配置
- 启停回调方法
- 子模块抽象
 - http
 - event
 - mail
 - stream

ngx_core_module_t
+name
+create_conf
+init_conf

ngx_http_module_t
+preconfiguration
+postconfiguration
+create_main_conf
+init_main_conf
+create_srv_conf
+merge_srv_conf
+create_loc_conf
+merge_loc_conf

ngx_event_module_t
+name
+create_conf
+init_conf
+add
+del
+enable
+disable
+add_conn
+del_conn
+process_changes
+process_events
+init
+done

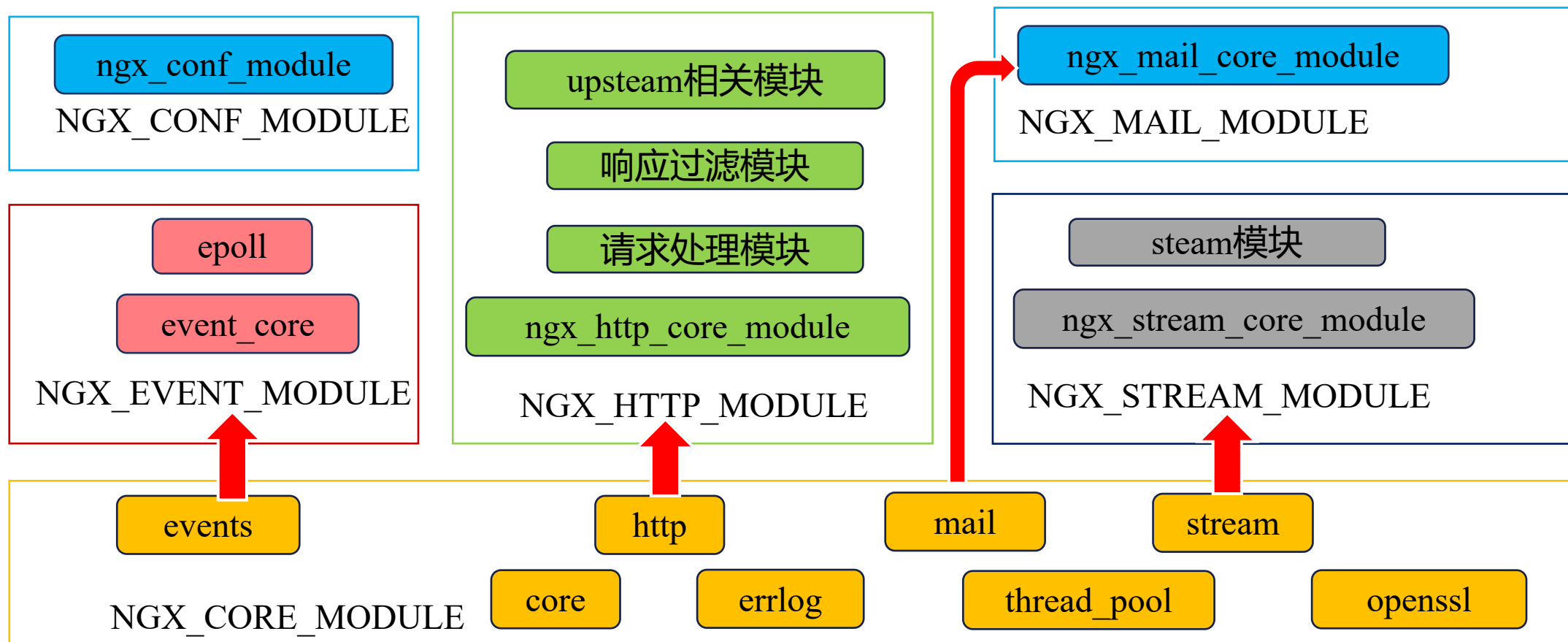
ngx_mail_conf_ctx_t
+main_conf
+srv_conf

ngx_module_t
+ctx_index
+index
+spare0-3
+version
+ctx
+commands : ngx_command_t
+type
+spare_hook0-7
+init_master
+init_module
+init_process
+init_thread
+exit_thread
+exit_process
+exit_master

ngx_command_t
+name
+type
+set
+conf
+offset
+post

每种模块
将具体化
ctx上下文

模块分类

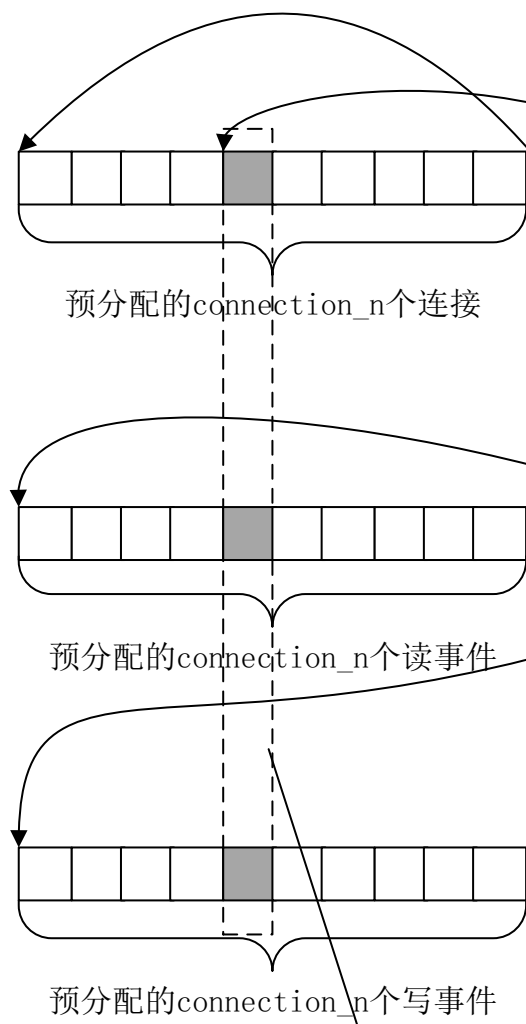


连接池

构成

对下游客户端的连接

对上游服务器的连接



connections指向的连接池中，每个连接所需要的读写事件都以相同的数组序号对应着read_events、write_events读写事件数组，所以相同序号下这三个数组中的元素是配合使用的

ngx_cycle_t
<div>+conf_ctx</div> <div>+pool</div> <div>+log</div> <div>+new_log</div> <div>+files</div> <div>+free_connections</div> <div>+free_connection_n</div> <div>+reusable_connections_queue : ngx_queue_s</div> <div>+listening : ngx_array_t</div> <div>+pathes : ngx_array_t</div> <div>+open_files : ngx_list_t</div> <div>+shared_memory : ngx_list_t</div> <div>+connection_n</div> <div>+files_n</div> <div>+connections</div> <div>+read_events</div> <div>+write_events</div> <div>+old_cycle</div> <div>+conf_file</div> <div>+conf_param</div> <div>+conf_prefix</div> <div>+prefix</div> <div>+lock_file</div> <div>+hostname</div> <div>+ngx_master_process_cycle()</div> <div>+ngx_single_process_cycle()</div> <div>+ngx_start_worker_processes()</div> <div>+ngx_start_cache_manager_processes()</div> <div>+ngx_pass_open_channel()</div> <div>+ngx_signal_worker_processes()</div> <div>+ngx_reap_children()</div> <div>+ngx_master_process_exit()</div> <div>+ngx_worker_process_cycle()</div> <div>+ngx_worker_process_init()</div> <div>+ngx_worker_process_exit()</div> <div>+ngx_cache_manager_process_cycle()</div> <div>+ngx_process_events_and_timers()</div>

核心数据结构

```
struct ngx_event_s {  
    void          *data;  
  
    unsigned       instance:1;  
    unsigned       timeout:1;  
    unsigned       timer_set:1;  
    unsigned       available:1;  
  
    ngx_event_handler_pt handler;  
    ngx_uint_t     index;  
    ngx_log_t      *log;  
    ngx_rbtreenode_t timer;  
    ngx_queue_t    queue;  
    ... ..  
};
```

96 字节

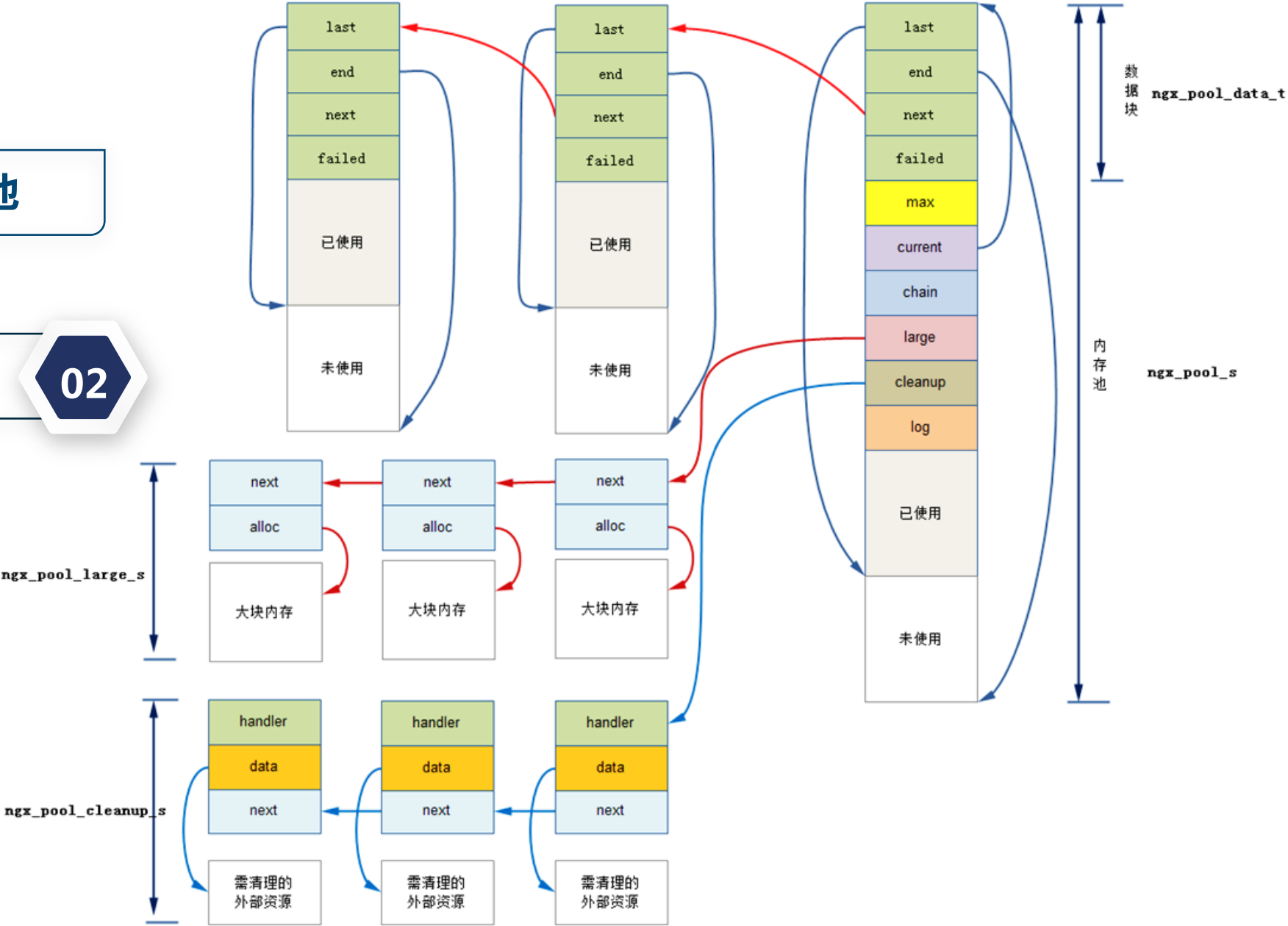
字节 232

```
struct ngx_connection_s {  
    void          *data;  
  
    ngx_event_t    *read;  
    ngx_event_t    *write; 读写事件  
  
    ngx_socket_t    fd;  
  
    ngx_recv_pt     recv;  
    ngx_send_pt     send; 抽象解耦OS底层方法  
    off_t           sent;    <= bytes_sent变量  
  
    ngx_log_t      *log;  
    ngx_pool_t      *pool; <= 初始 connection_pool_size配置  
  
    int             type;  
  
    struct sockaddr *sockaddr;  
    socklen_t       socklen;  
    ngx_str_t       addr_text;  
    ngx_str_t       proxy_protocol_addr;  
    in_port_t       proxy_protocol_port;  
    ngx_buf_t       *buffer;  
    ngx_queue_t     queue;  
  
};
```

内存池

01 连接内存池

请求内存池 02



Nginx进程间的通讯方式



基础同步工具

- 信号
- 共享内存

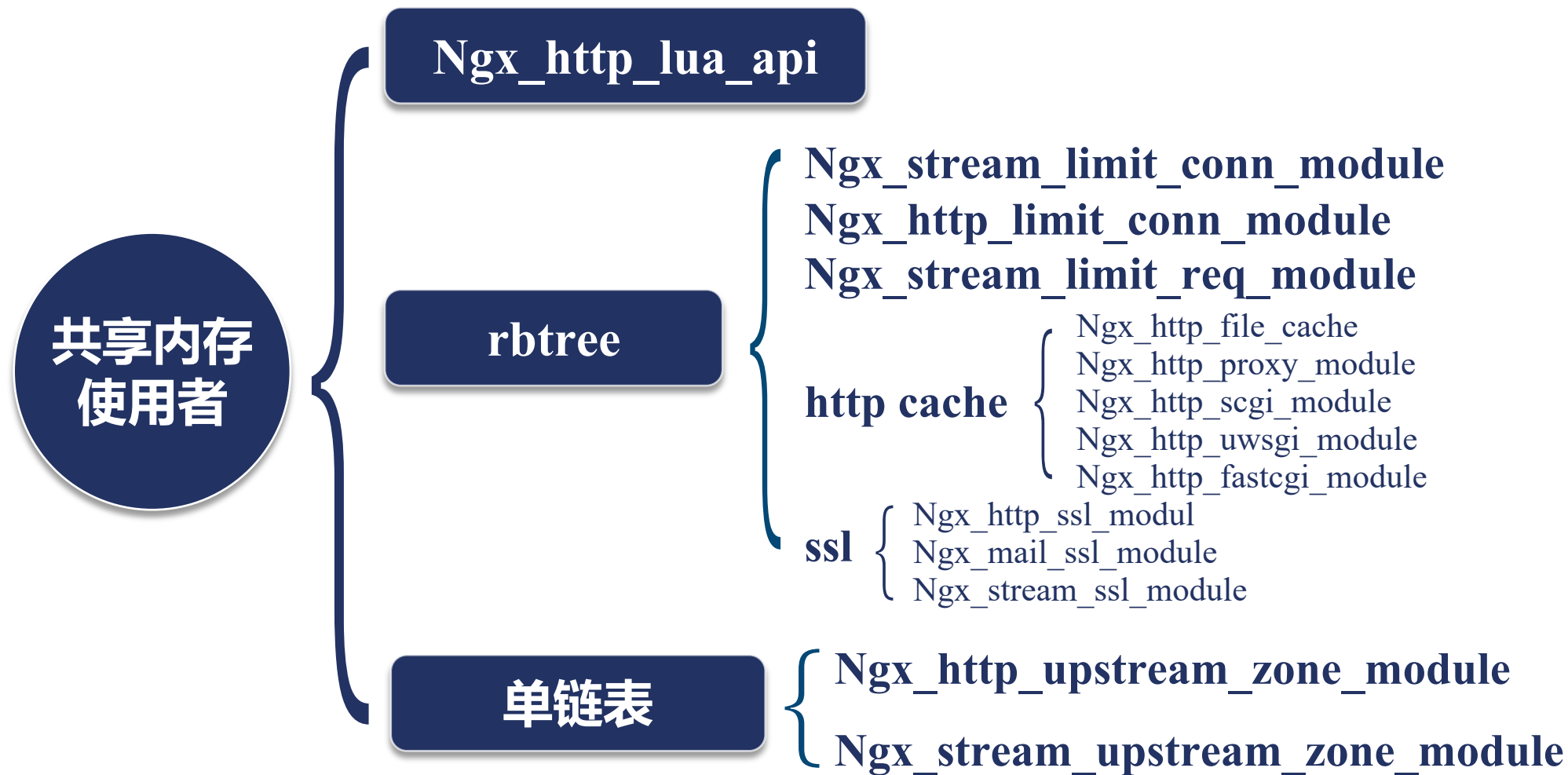


高级通讯方式

- 锁
- Slab内存管理器

共享内存：跨worker进程通讯

哪些官方nginx模块使用了共享内存



OpenResty共享内存代码示例

```
http {  
    lua_shared_dict dogs 10m;  
    server {  
        location /set {  
            content_by_lua_block {  
                local dogs = ngx.shared.dogs  
                dogs:set("Jim", 8)  
                ngx.say("STORED")  
            }  
        }  
        location /get {  
            content_by_lua_block {  
                local dogs = ngx.shared.dogs  
                ngx.say(dogs:get("Jim"))  
            }  
        }  
    }  
}
```

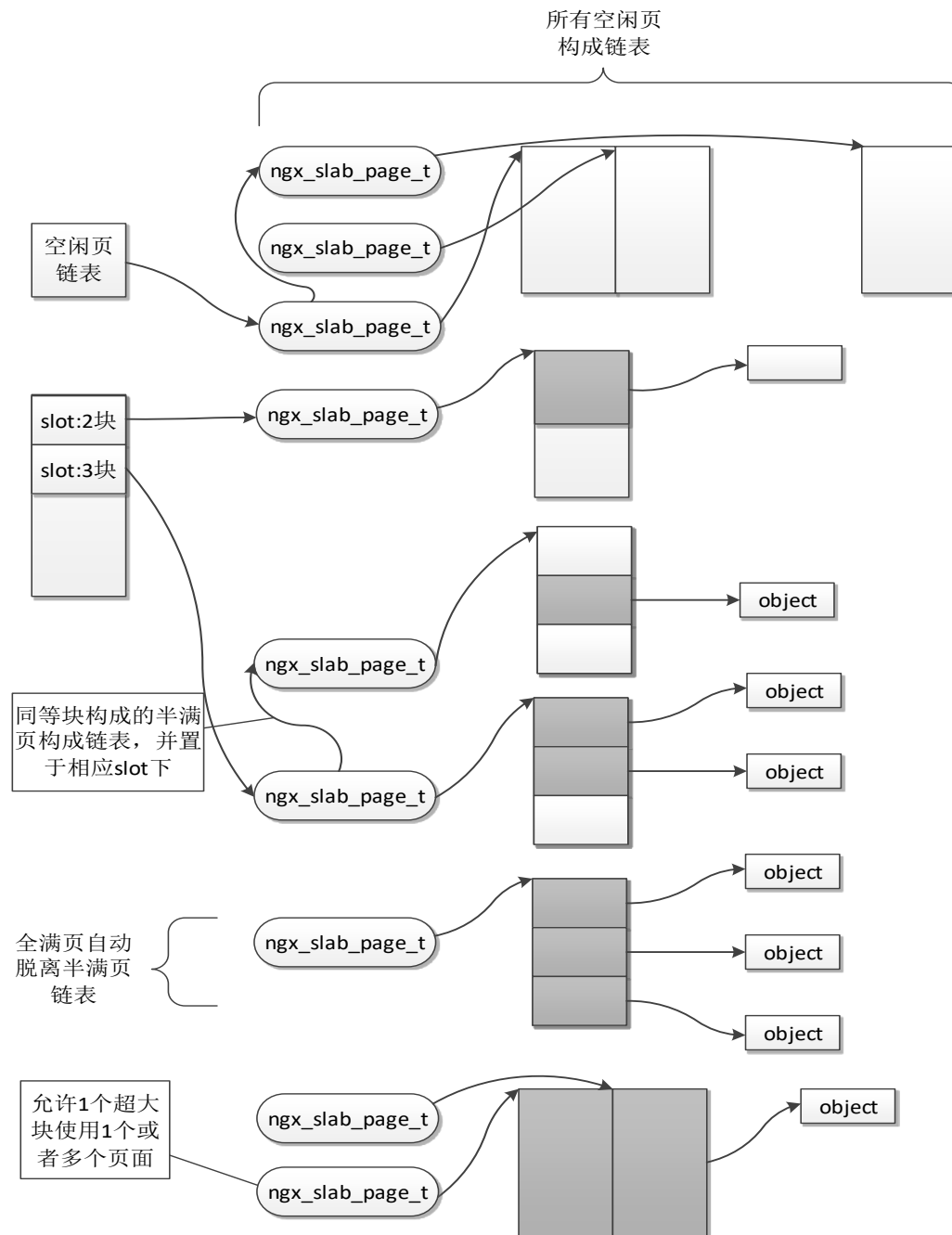

Slab内存管理

Bestfit

- 最多两倍内存消耗

Bestfit

- 适合小对象
- 避免碎片
- 避免重复初始化



ngx_slab_stat : 统计Slab使用状态

```
$ curl http://localhost:80/slab_stat
* shared memory: one
total:      102400(KB) free:      101792(KB) size:      4(KB)
pages:      101792(KB) start:0000000003496000 end:0000000009800000
slot:        8(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:       16(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:       32(Bytes) total:     127 used:      1 reqs:      1 fails:      0
slot:       64(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:      128(Bytes) total:     32 used:      1 reqs:      1 fails:      0
slot:      256(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:      512(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:     1024(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:     2048(Bytes) total:      0 used:      0 reqs:      0 fails:      0
```

Nginx 容器



数组



链表



队列



哈希表

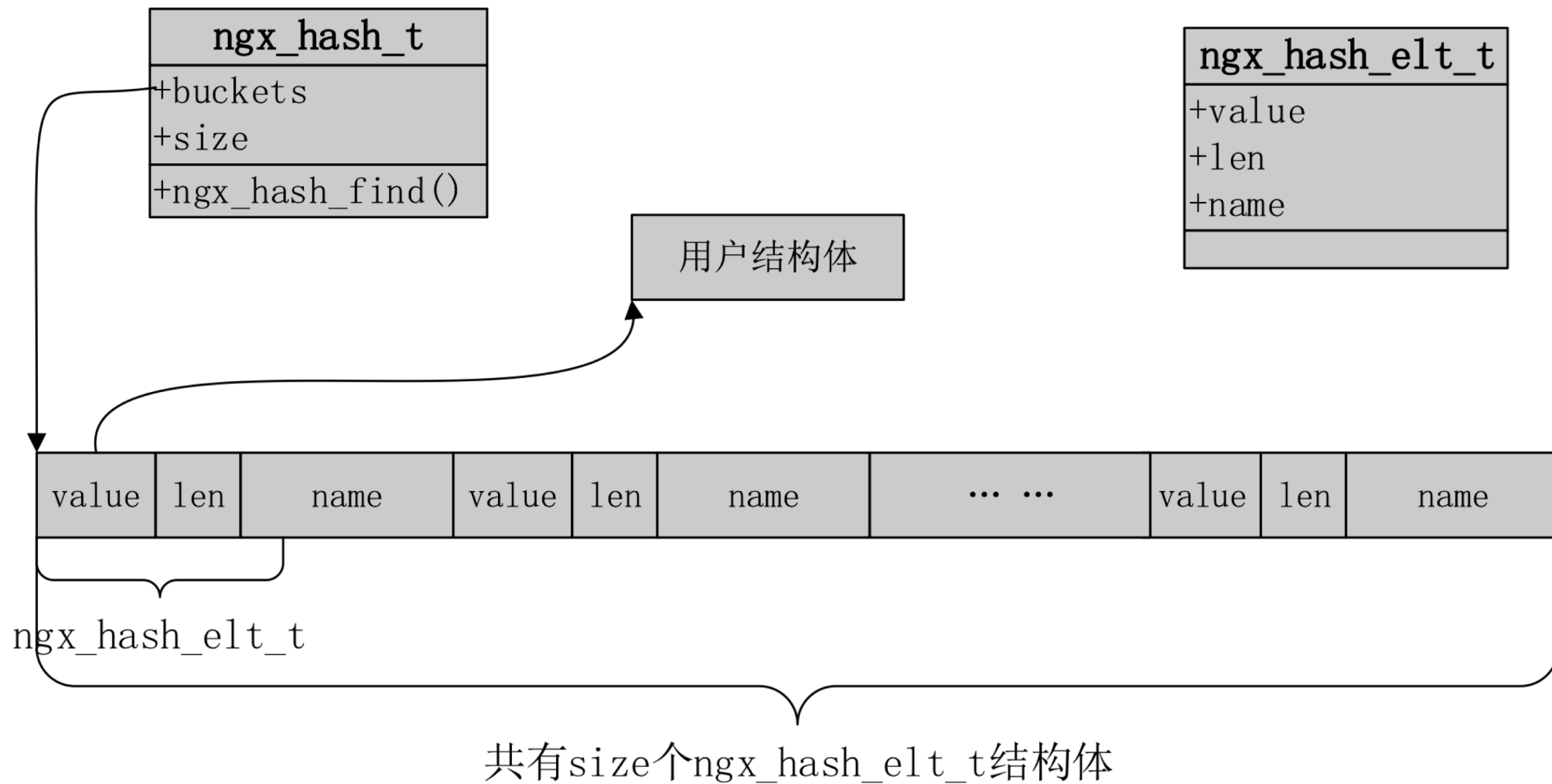


红黑树



基数树

Nginx 哈希表

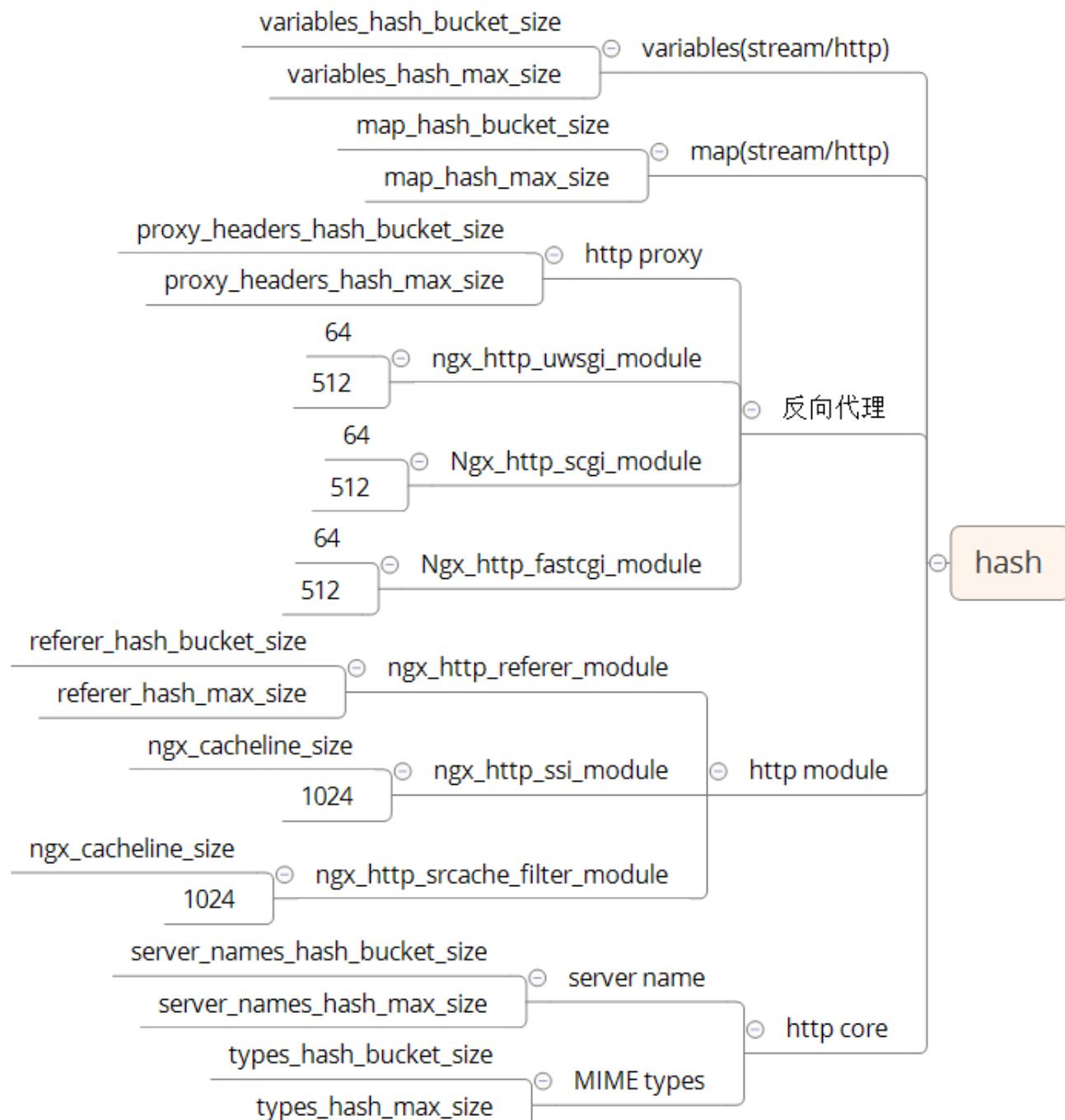


哈希表配置

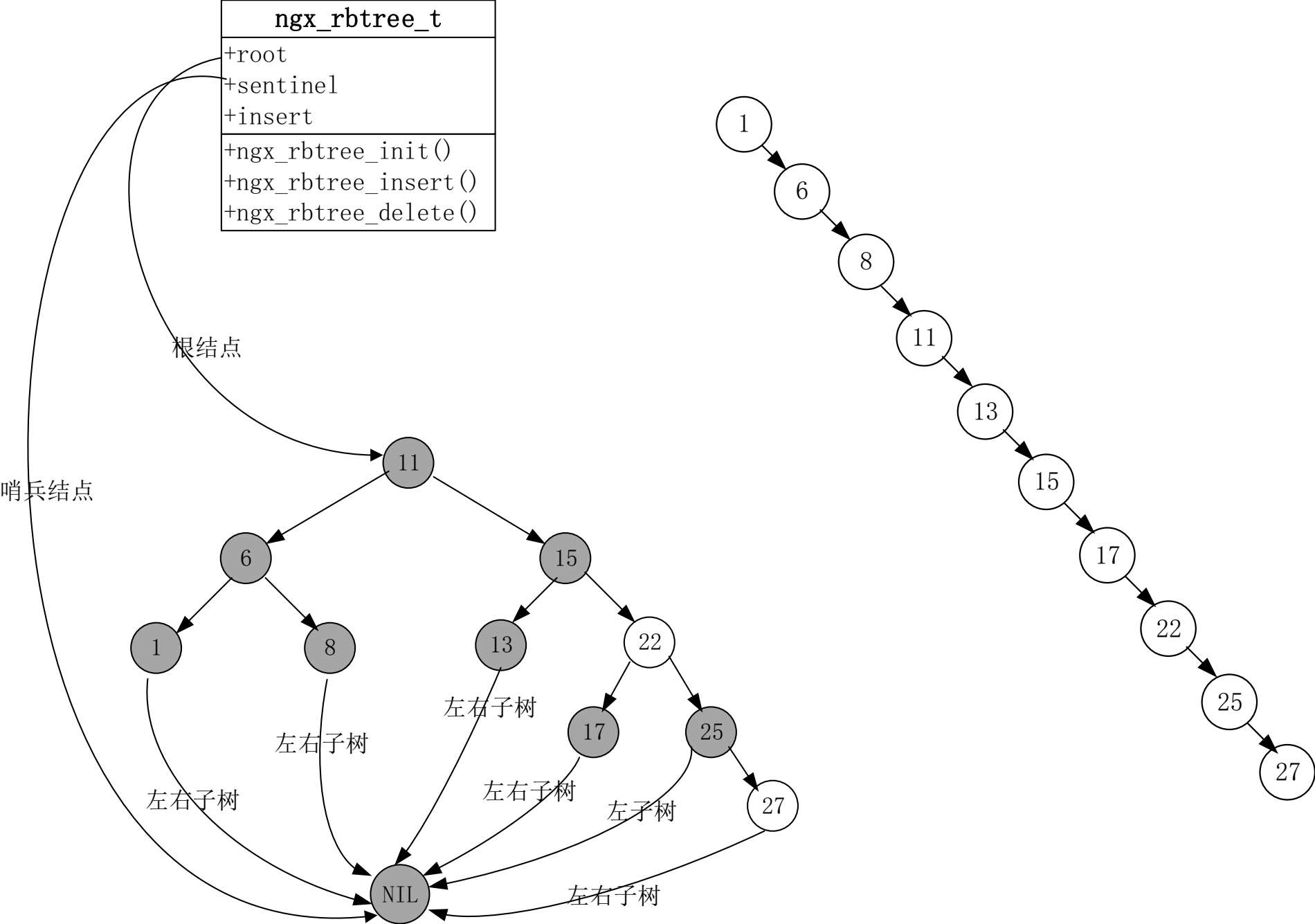
Bucket size

对齐问题

Max size



红黑树



红黑树

自平衡二
叉查找树

高度不会超过2倍 $\log(n)$

增删改查算法复杂度 $O(\log(n))$

遍历复杂度 $O(n)$

红黑树的使用模块

红黑树

ngx_conf_module `config_dump_rbtrees`

ngx_event_timer_rbtrees

Ngx_http_file_cache

Ngx_http_geo_module

Ngx_http_limit_conn_module

Ngx_http_limit_req_module

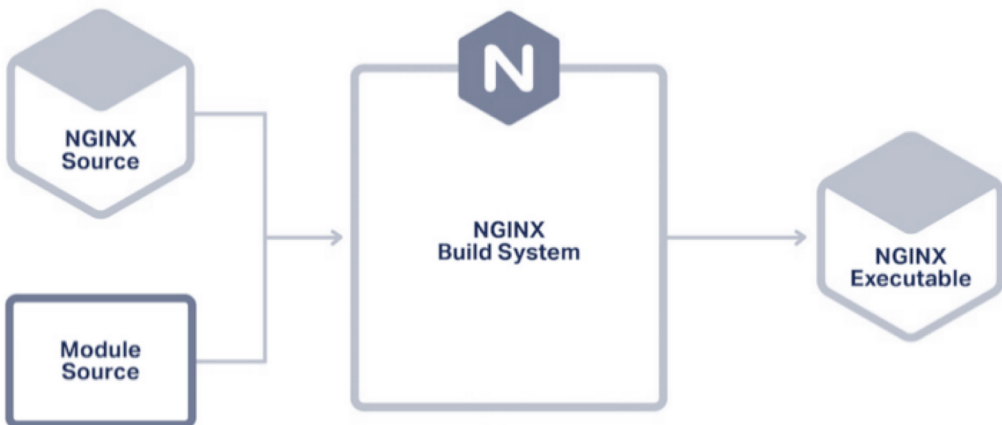
Ngx_http_lua_shdict:ngx.shared.DICT LRU链表性质

resolver `ngx_resolver_t`

Ngx_stream_geo_module

Ngx_stream_limit_conn_module

动态模块-减少编译环节



`load_module modules/nginx_http_image_filter_module.so;`

