



从头搭建Nginx静态资源服务

Nginx是什么？

- **Web场景的高性能解决方案**

- 静态资源服务：高效的磁盘IO处理能力
 - CDN
- 七层/四层负载均衡：高效的网络协议处理能力
 - 正向代理
 - 反向代理
- 整合Lua语言的负载均衡：丰富而完善的生态
 - API网关
 - Waf防火墙

Nginx有什么特点？

- 高性能
 - 高并发（C10M）
 - 低时延
- 稳定
 - mainline/stable版本
 - master/worker进程结构
- 开放的生态
 - BSD License: [Tengine](#)
 - 拥有极高自由度的C模块: [Openresty](#)、[Kong](#)

应当如何获取、安装Nginx?

- 放弃定制化能力
 - docker镜像
 - apt-get/yum安装
 - 直接下载编译好的二进制文件
- 定制化Nginx/Openresty
 - nginx.org/nginx-cn.net下载源代码
 - 下载第三方模块源代码
 - configure/make/make install

如何确定该选择哪个Nginx版本？

- CHANGES
 - bugfix
 - feature
 - change
 - security
- 你的Nginx用的是哪个版本？
 - `nginx -v`

为什么需要定制化?

- 默认configure定制后的Nginx，没有哪些功能?
 - 不支持http2
 - 不支持TCP/UDP协议负载均衡
 - 不支持stub_status性能监控
 - 不支持TLS/SSL安全协议
 -
- Nginx为什么要设计编译时的定制化功能?
 - 提升性能
 - 减小可执行文件的体积
 - 有些功能，依赖许多软件，准备环境较复杂
 - 强大的自定义功能，包括编译、运行时各种路径、参数的指定

怎样编译、安装、启动Nginx?

1. 认识Nginx源代码目录
2. configure定制Nginx的编译、运行环境
3. make编译nginx
4. make install安装
5. 启动nginx

configure的用法

- 你的Nginx用了哪些configure选项？

- `nginx -V`

- **configure步骤**

- 解析configure参数，生成编译参数：`auto/options`
 - 第三方模块通过`--add-module`参数会添加模块至`NGX_ADDONS`变量
 - 针对不同操作系统、体系架构、编译器，选择特性（例如linux中的`epoll`或者windows中的`ioep`）及生成相应编译参数
 - 根据所有模块生成`ngx_modules.c`及`makefile`
 - 在屏幕上显示configure执行结果：`auto/summary`

nginx.conf语法格式

- 指令以;符号结尾，以空格分离参数
- 指令块包裹在{}中
- #是注释
- \$是变量
- 各指令以不同方式支持正则表达式
- include可以读入新文件，方便维护

配置参数的单位

- 时间

- ms: 毫秒
- s: 秒
- m: 分钟
- h: 小时
- d: 天
- w: 周
- M: 月
- y: 年

- 空间

- k/K: KB
- m/M: MB
- g/G: GB

```
http {  
    include      mime.types;  
    upstream thwp {  
        server 127.0.0.1:8000;  
    }  
  
    server {  
        listen 443 http2;  
        #Nginx配置语法  
        limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;  
        location ~* \.(gif|jpg|jpeg)$ {  
            proxy_cache my_cache;  
            expires 3m; proxy_cache_key $host$uri$is_args$args;  
            proxy_cache_valid 200 304 302 1d;  
            proxy_pass http://thwp;  
        }  
    }  
}
```

nginx.conf在哪里？

- `nginx -c`: 命令行指定配置文件路径
- `configure --prefix`: 编译时指定
- `configure --conf-path`: 编译时指定

如何测试语法格式？

- `nginx -t/-T`
 - 错误级别
 - 错误描述
 - 错误行数

如何让vim “有颜色” 的显示nginx.conf?

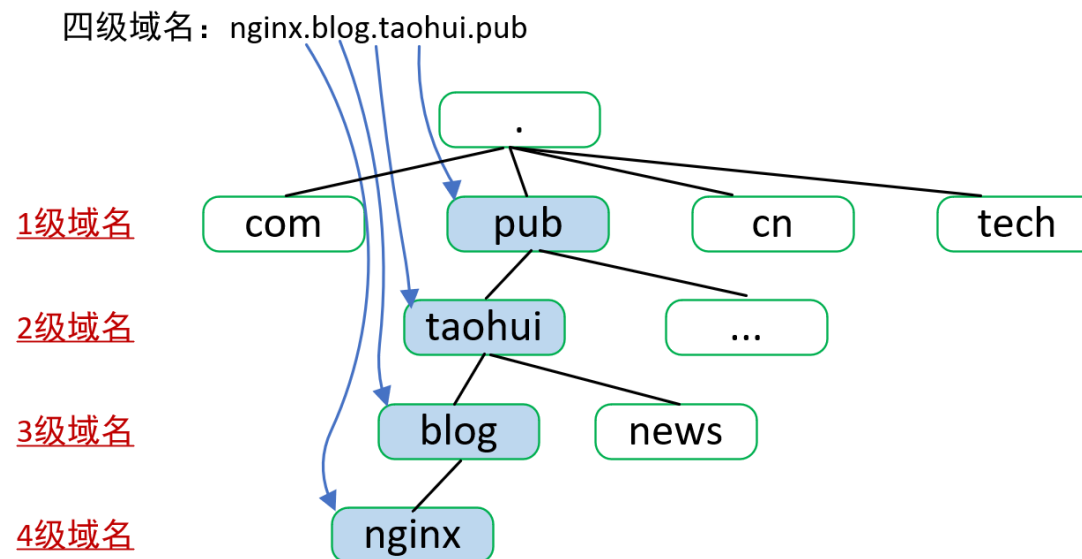
- 复制contrib/vim/*至 ~/.vim/

如何学习每个官方指令的用法？

- <http://www.nginx.org>
 - 寻找到指令
 - 指令索引
 - 模块
 - 查看用法
 - Syntax语法
 - Default默认值
 - Context上下文
 - 描述

server与虚拟主机

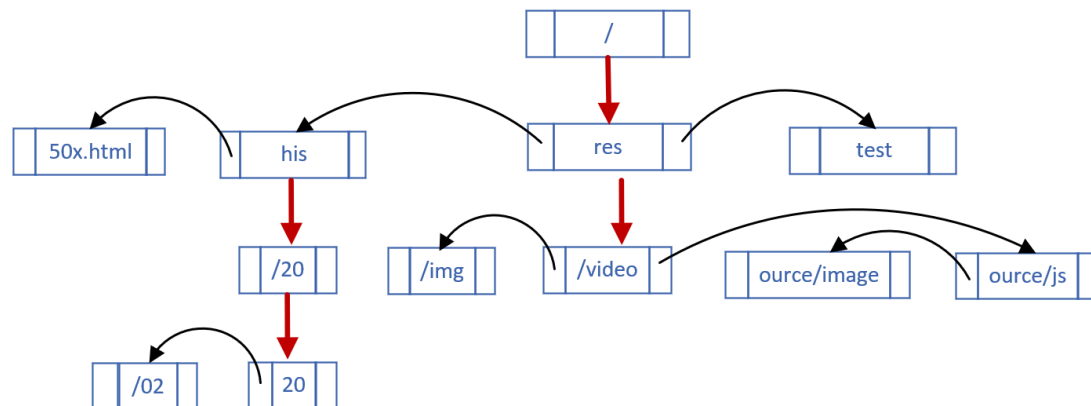
- 多级域名匹配
- server_name
 - 精确匹配
 - 前缀通配符匹配
 - 后缀通配符匹配
 - 正则表达式匹配



URL与location

- location的匹配优先级

- 精确匹配
- 正则表达式匹配
- 最长前缀匹配



搭建静态资源服务

- URL与路径的映射
 - root指令：映射完整的URL路径
 - alias指令：只映射location后的URL路径
- 文件后缀名与content-type头部的映射
 - types
 - default_type text/plain
 - types_hash_bucket_size 64
 - types_hash_max_size 1024

提供HTML/JSON等目录服务

- autoindex on;
- autoindex_exact_size on;
- autoindex_format **html** | **xml** | **json** | **jsonp**;
- autoindex_localtime on;

index默认首页的处理

- index index.html;

连续/的合并

- `merge_slashes on;`

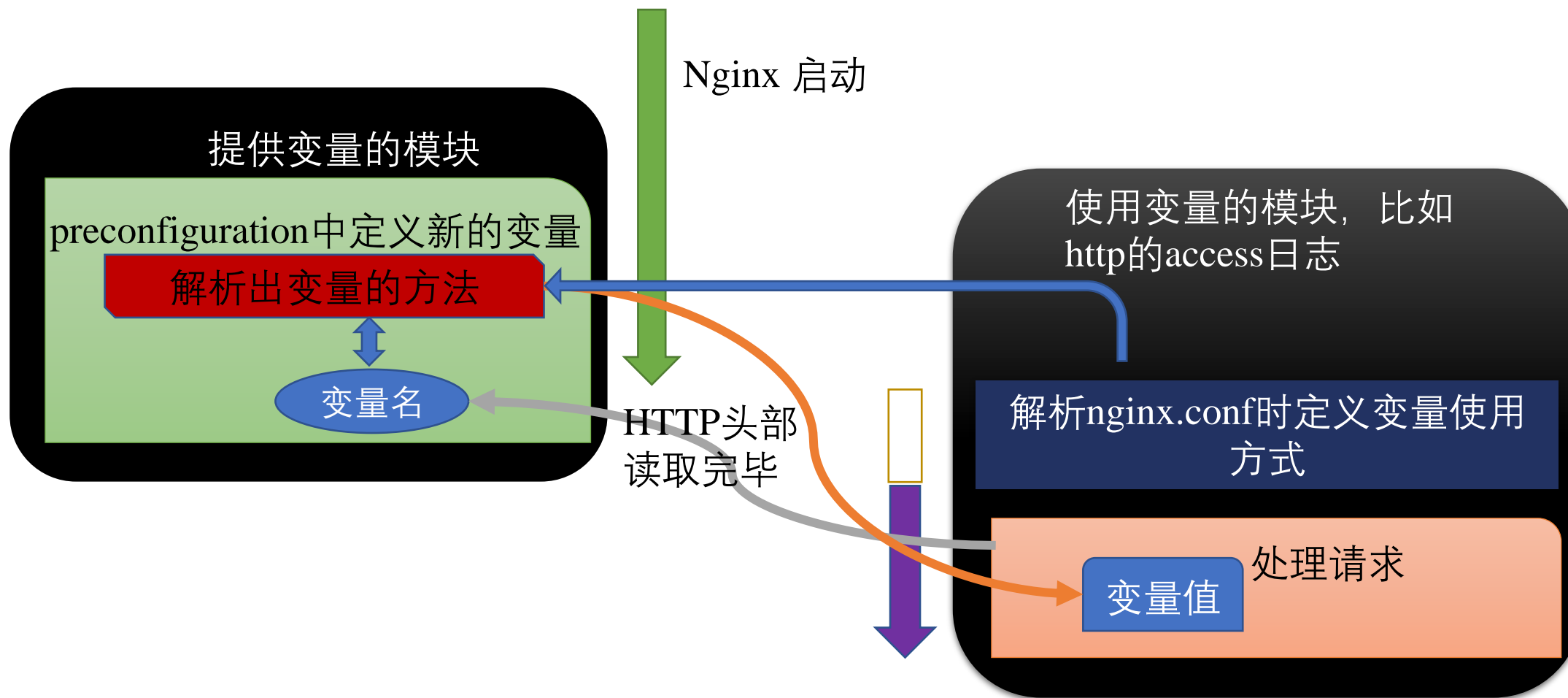
如何定位映射失败问题？

- 最有效率的方式：error.log
- 403
- 404

通过Nginx变量找到映射路径

- `$document_root`: 文件目录路径
- `$realpath_root`: 替换软链接的文件目录路径
- `$request_filename`: 含有文件名及扩展名的路径

惰性变量



access_log

Syntax: **log_format** *name* [escape=default|json|none] *string* ...;

Default: log_format **combined** "...";

Context: [http](#)

Syntax: **access_log** *path* [*format* [buffer=*size*] [gzip[=*level*]] [flush=*time*] [if=*condition*]];
access_log off;

Default: access_log logs/access.log combined;

Context: http, server, location, if in location, limit_except

命令行分析：ngxtop

- <https://github.com/lebinh/ngxtop>
- 安装：pip install ngxtop

可视化分析：goaccess

- <https://goaccess.io/get-started>
- 安装： `yum/apt-get install goaccess`
- 命令行
 - `goaccess access.log -c`
- 生成实时页面
 - `goaccess access.log -o /var/www/html/report.html --log-format=COMBINED --real-time-html`

可视化分析: visitors

- <http://www.hping.org/visitors/>
- 生成HTML页面
 - `./visitors -A -m 30 /usr/local/nginx/logs/access.log -o html > /usr/local/nginx/html/visitor_report.html`
- 生成追踪访问图
 - `./visitors /root/thaccess.log --prefix http://www.taohui.pub -V > /usr/local/nginx/html/visitor_graph.dot`
 - `dot /usr/local/nginx/html/visitor_graph.dot -Tpng > /usr/local/nginx/html/visitor_graph.png`

切割日志

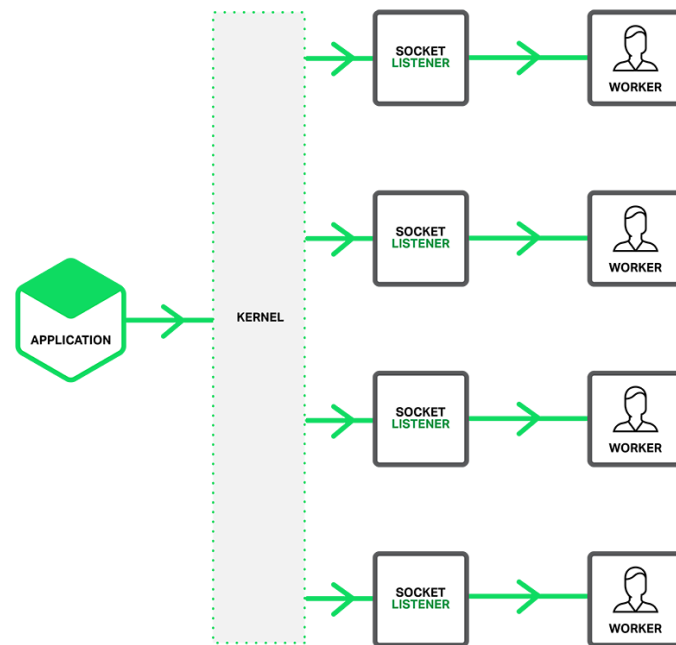
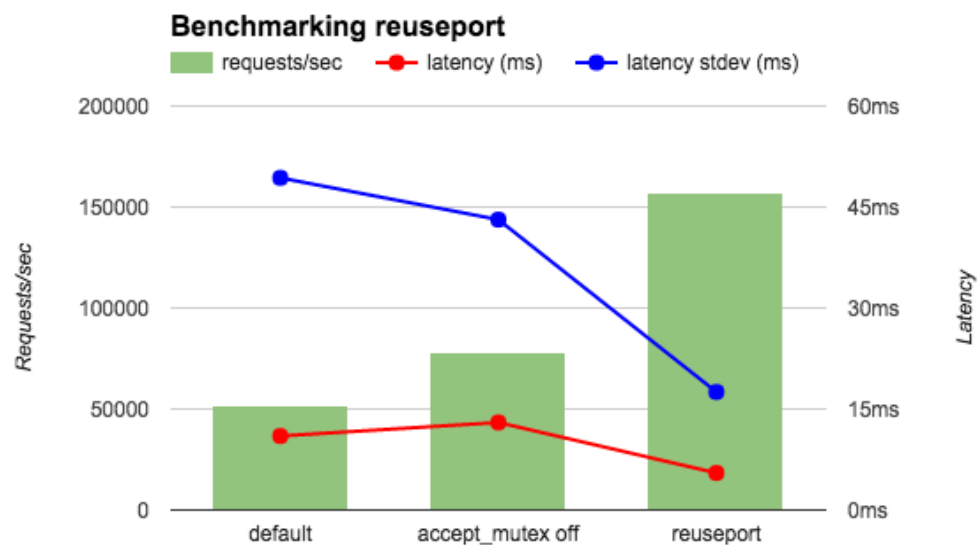
- 切割日志
 - `nginx -s reopen`
 - USR1信号: `kill -USR1 nginx_master_pid`
- 定时备份日志
 - `crontab`
 - `mv`移动并重新命名日志
 - `reopen`切割日志

Nginx的进程架构

- master_process **on** | **off**;
- worker_processes **number** | **auto**;
 - worker_cpu_affinity auto [cpumask];

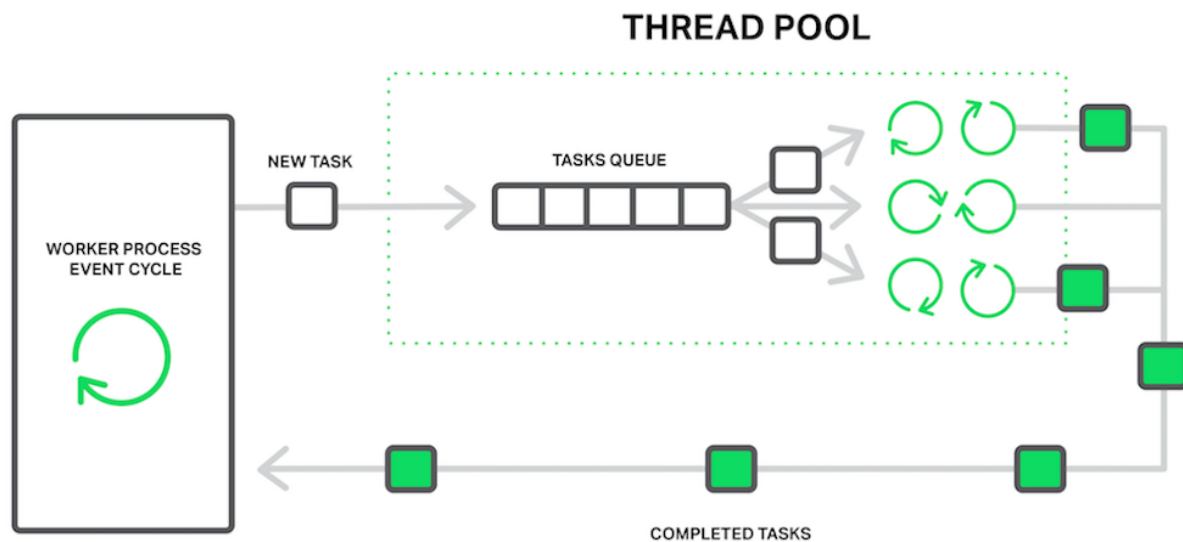


worker间分流请求的方式



Nginx的线程架构

- `thread_pool name threads=number [max_queue=number];`

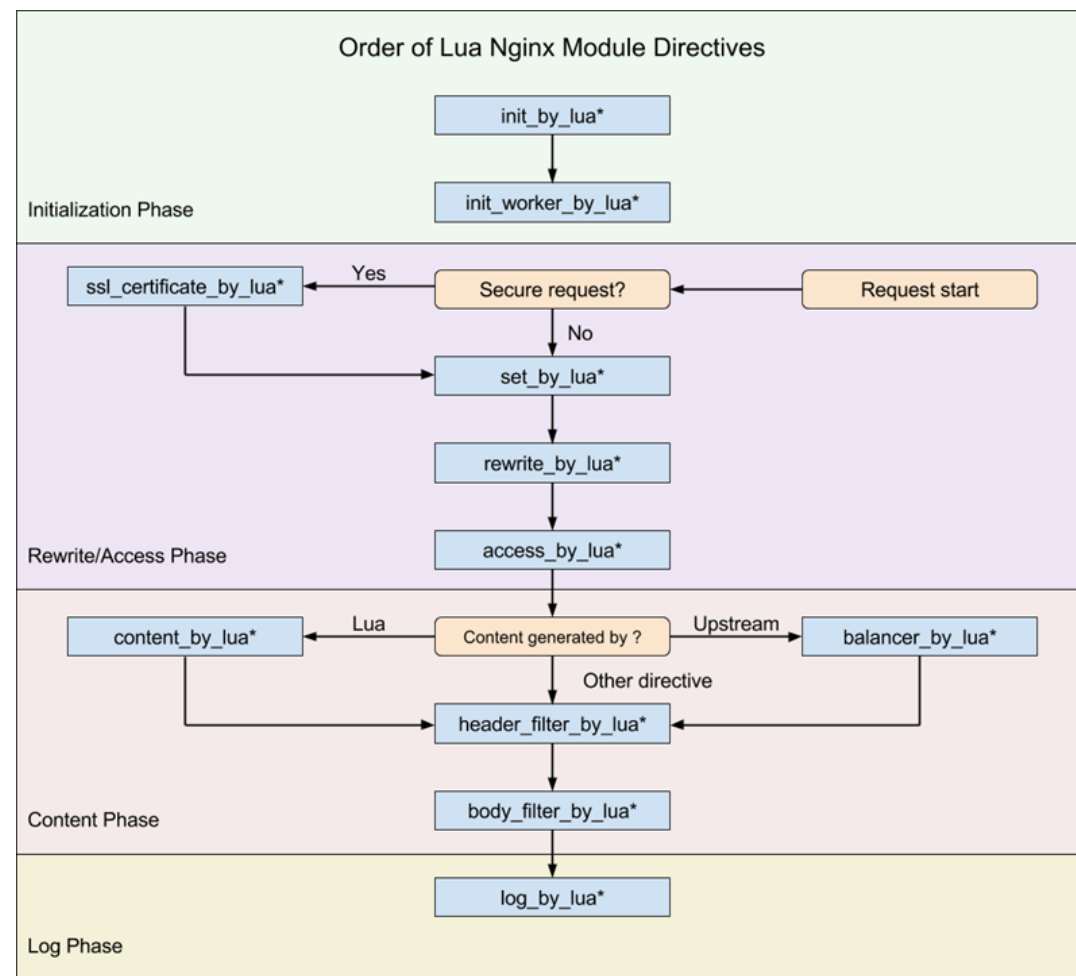


为何要采用这种进程架构？

- 为什么不是单进程单线程？
 - 多核架构下提升并行性
- 为什么不是多线程？
 - 进程的容错能力更高
 - 事件驱动+非阻塞异步开发模式
- 需要master进程吗？
 - 父子进程方便监控（CHLD信号）、权限控制（USER/GROUP）、升级
- cache loader/manage进程有什么用？
 - 将cache功能分离
- 为什么又要引入多线程？
 - read函数阻塞时，使用线程池提高并发

master与worker初始化的异同

- worker进程还是master进程?
 - 读取并由各模块解析自己感兴趣的配置指令
 - 创建共享内存
 - 打开资源文件
 - 监听端口



通讯方式

- master与shell间采用信号通讯
- master与worker进程间采用socket pair通讯
- worker进程间采用共享内存通讯
 - 块内存管理：slab
 - 同步方式：自旋锁
- 线程间直接通讯

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**

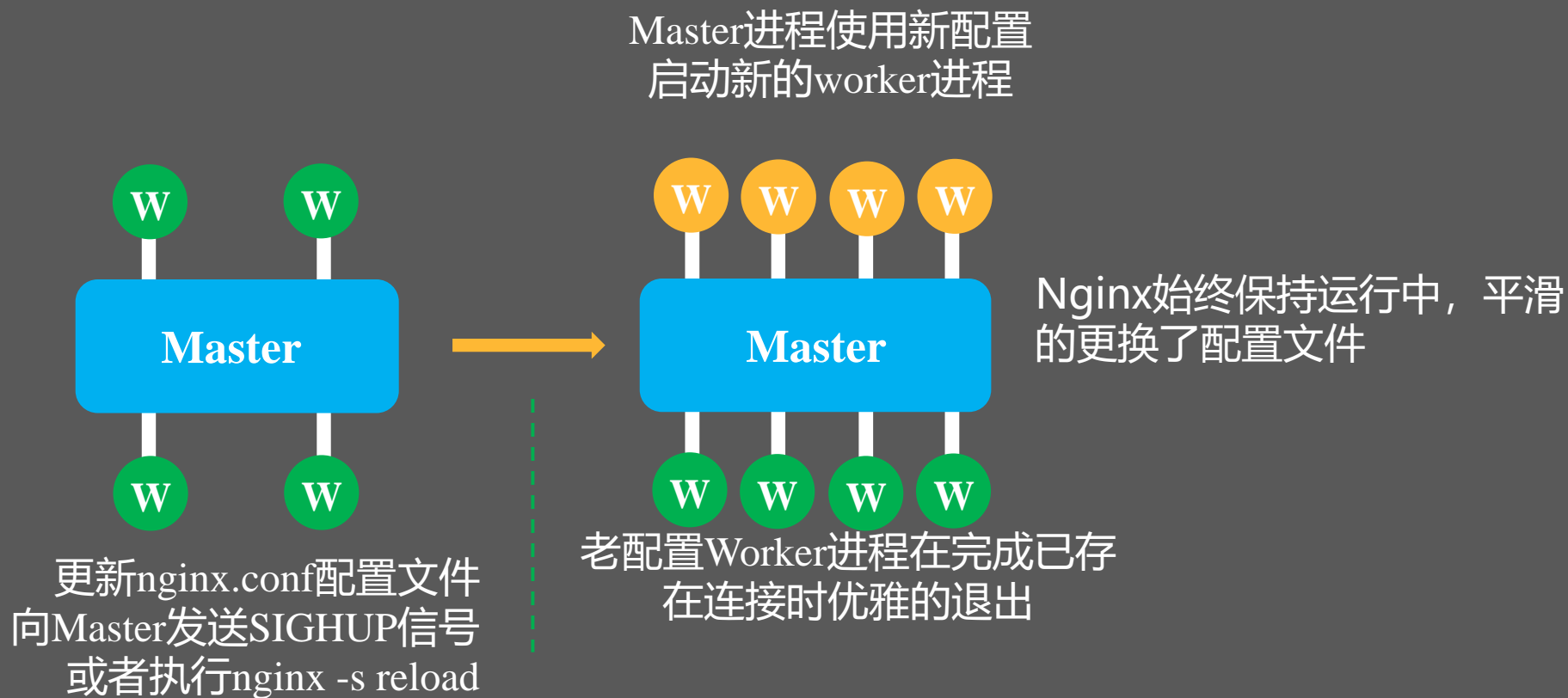
Worker进程

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

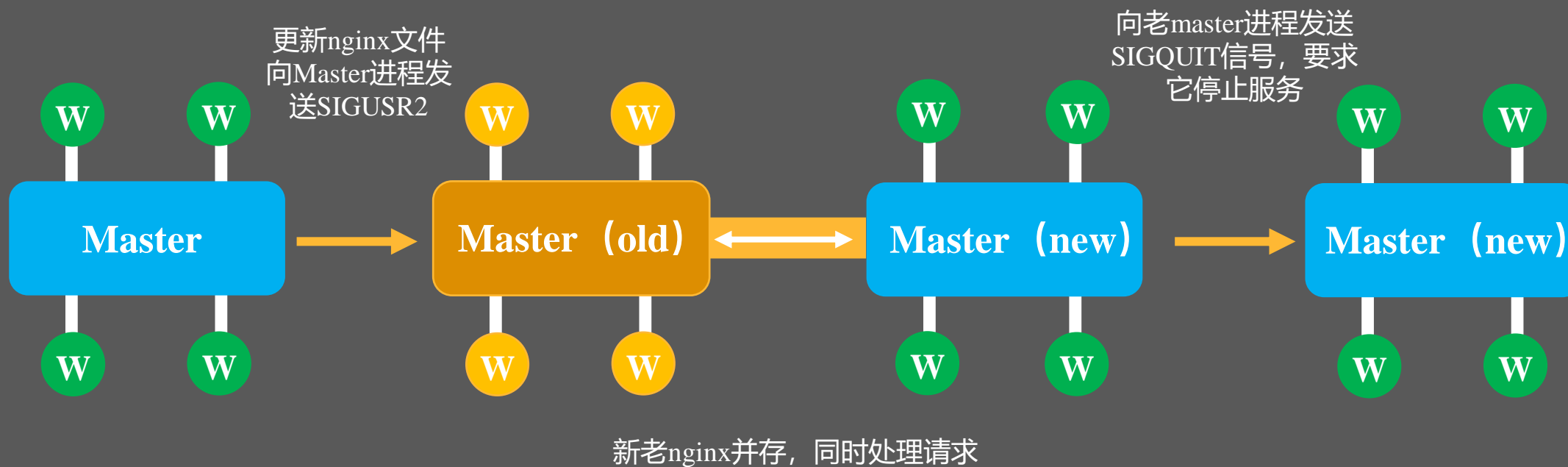
nginx命令行

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

reload



热升级



优雅关闭的问题

- 长连接
 - TCP
 - HTTP KeepAlive
- `worker_shutdown_timeout` **time**;

共享内存的应用（1）

- 负载均衡策略的生效范围：基于内存还是共享内存？
 - zone **name** [**size**];
- 四层、七层并发连接限制
 - limit_conn **zone** **number**;
 - limit_conn_zone **key** zone=**name**:**size**;
- 请求处理速度的控制
 - limit_req zone=**name** [burst=**number**] [**nodelay** | delay=**number**];
 - limit_req_zone **key** zone=**name**:**size** rate=**rate** [**sync**];

共享内存的应用（2）

- HTTP缓存

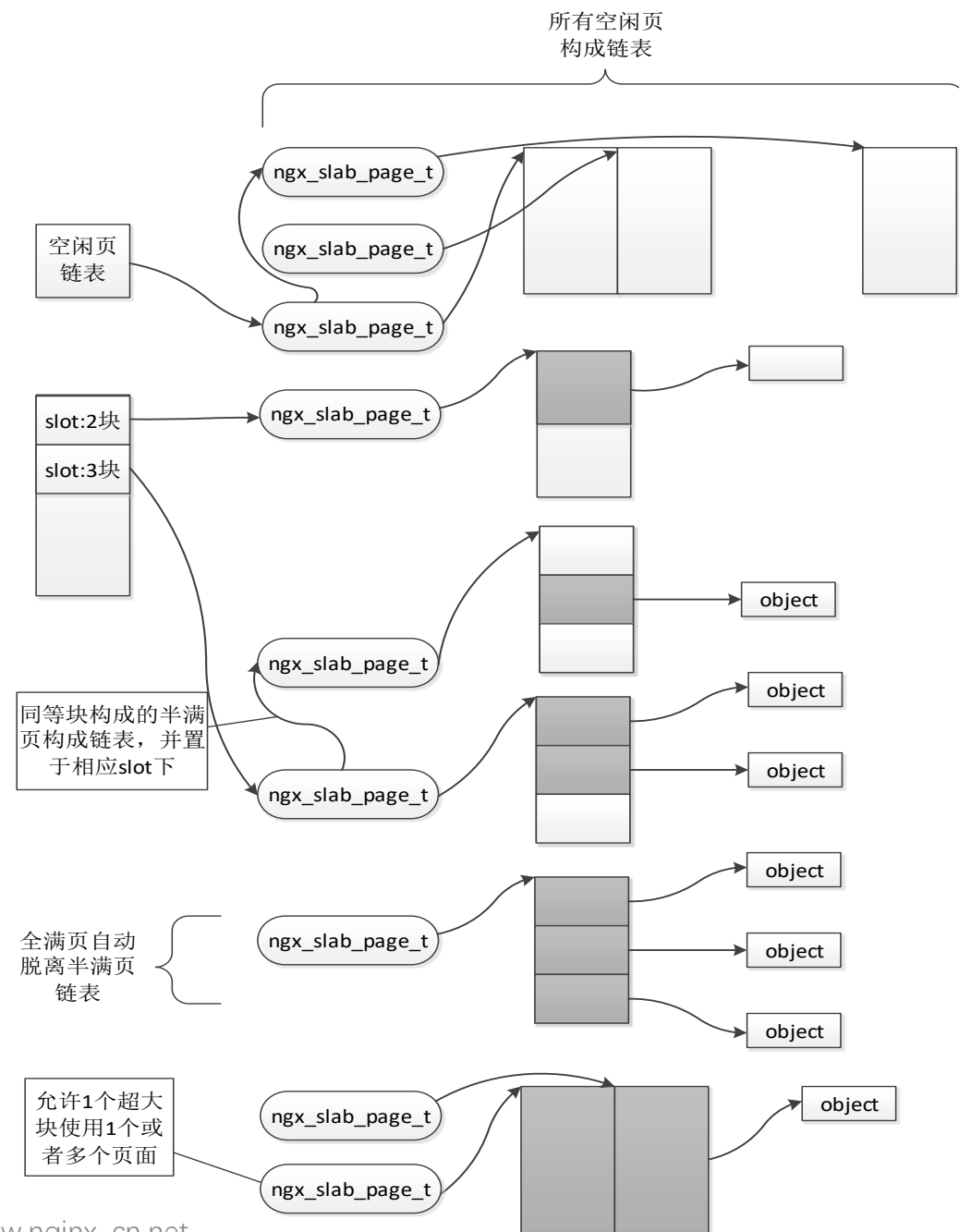
- `proxy/fastcgi/scgi/uwsgi_cache zone;`
- `proxy/fastcgi/scgi/uwsgi_cache_path path [levels=levels]
[use_temp_path=on|off] keys_zone=name:size [inactive=time]
[max_size=size] [manager_files=number] [manager_sleep=time]
[manager_threshold=time] [loader_files=number] [loader_sleep=time]
[loader_threshold=time] [purger=on|off] [purger_files=number]
[purger_sleep=time] [purger_threshold=time];`

- TLS/SSL session缓存

- `ssl_session_cache off | none | [builtin[:size]] [shared:name:size];`

slab

- 两倍内存消耗
- 适合小对象
- 避免碎片
- 避免重复初始化



共享内存的监控: slab_stat

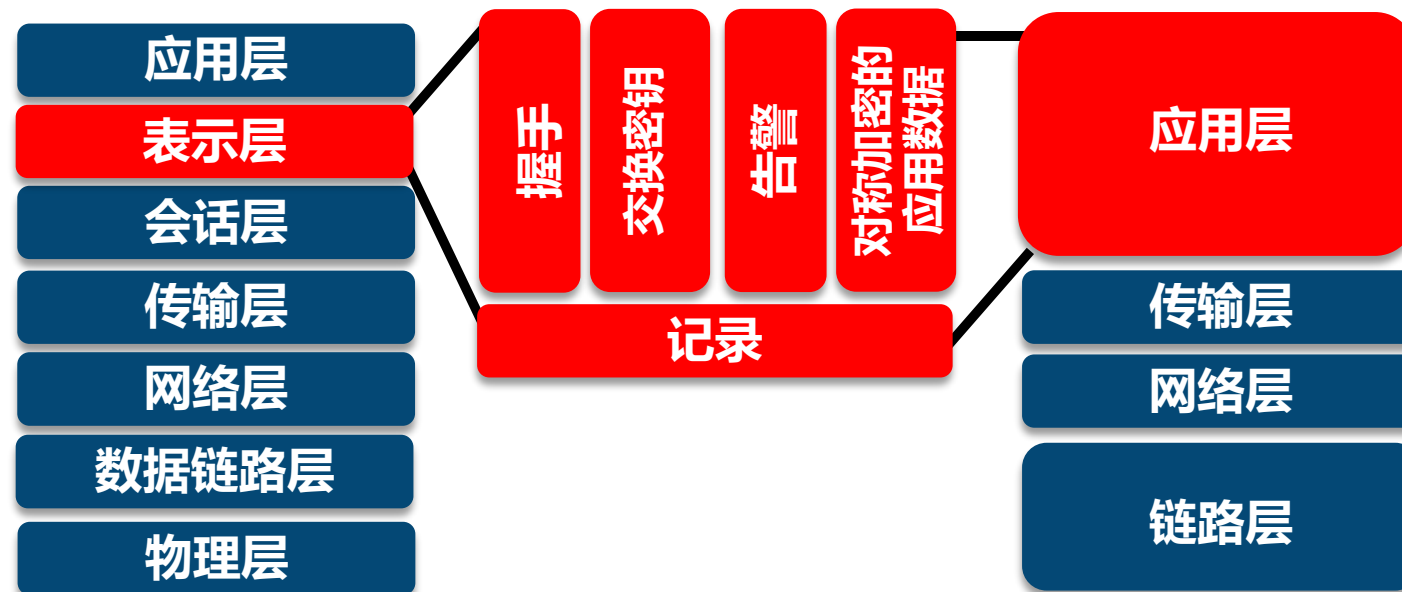
```
$ curl http://localhost:80/slab_stat
* shared memory: one
total:      102400(KB) free:      101792(KB) size:      4(KB)
pages:      101792(KB) start:000000003496000 end:000000009800000
slot:        8(Bytes) total:      0 used:      0 reqs:      0 fails:      0
slot:       16(Bytes) total:      0 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
```

TLS (Transport Layer Security) /SSL(Secure Sockets Layer) 发展

SSL/TLS 通用模型

ISO/OSI 模型

TCP/IP 模型



1995

SSL3.0

1999

TLS1.0

2006

TLS1.1

2008

TLS1.2

2018

TLS1.3

TLS 安全密码套件解读



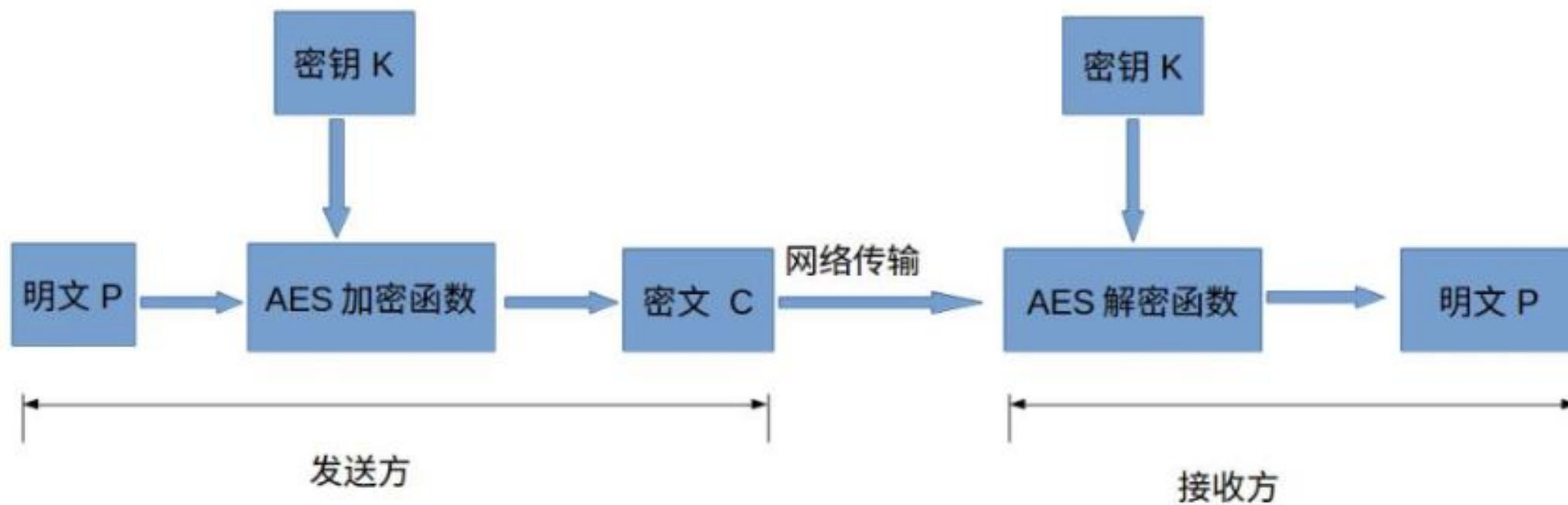
密钥交换算法

身份验证算法

对称加密算法、强度、工作模式

签名hash算法

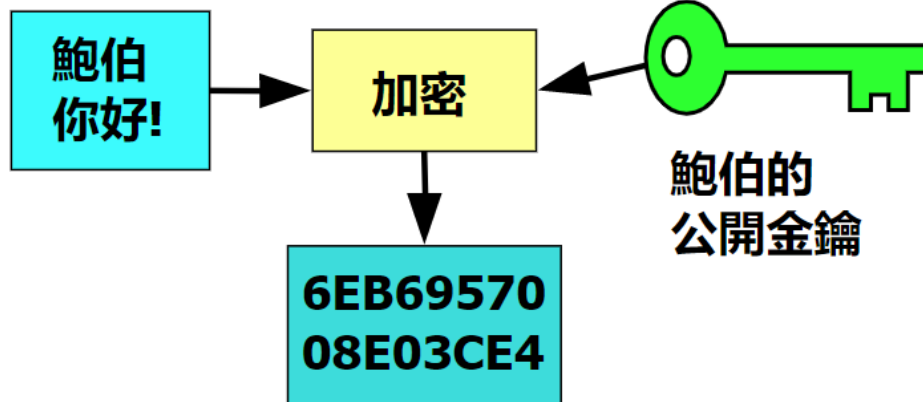
对称加密



非对称加密

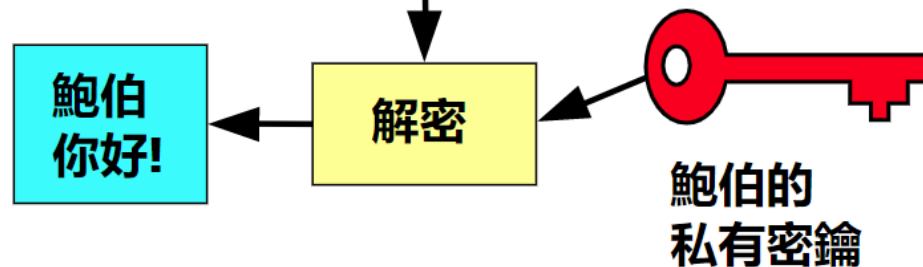
- 身份验证
- 加密

愛麗斯



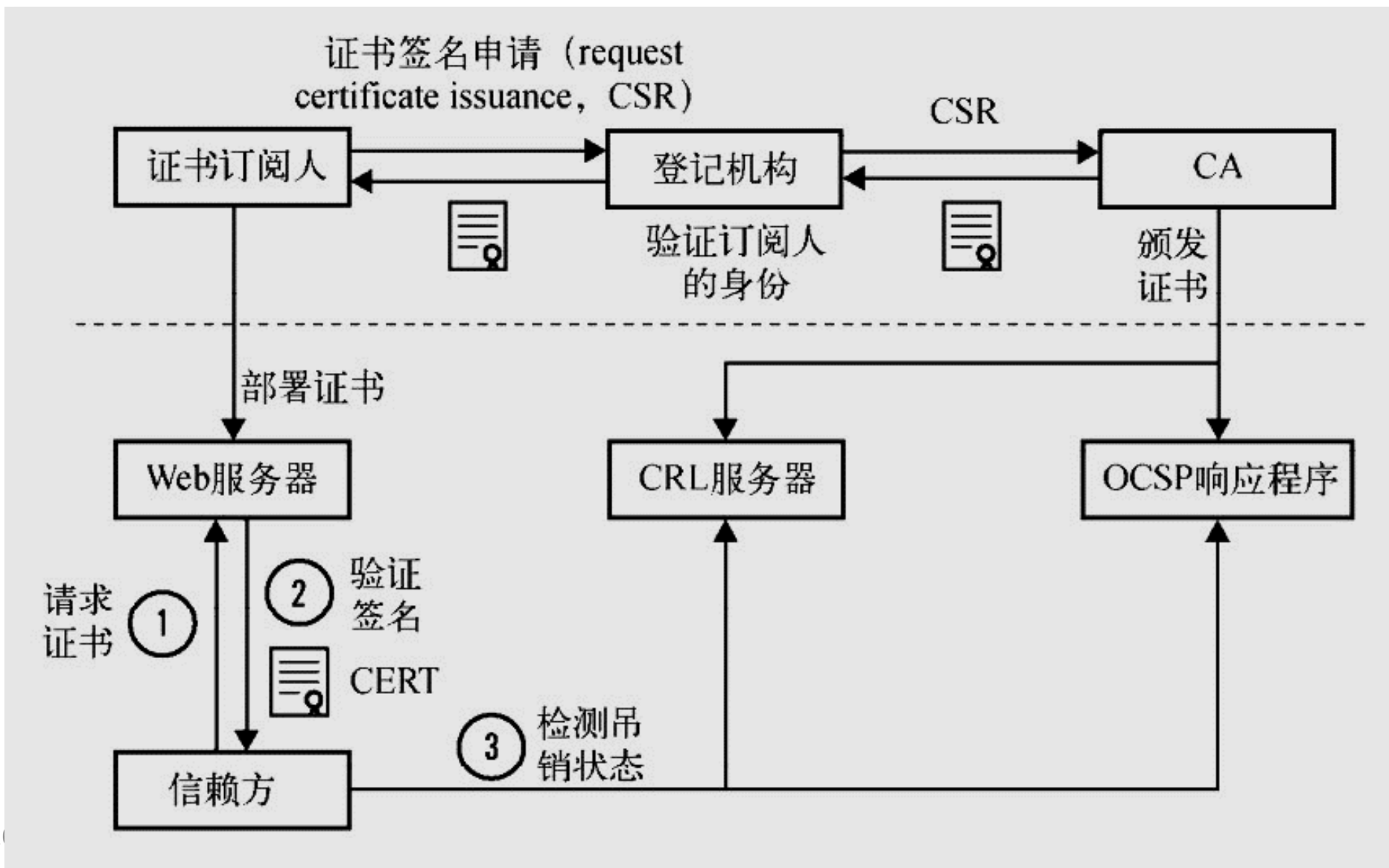
鮑伯的
公開金鑰

鮑伯



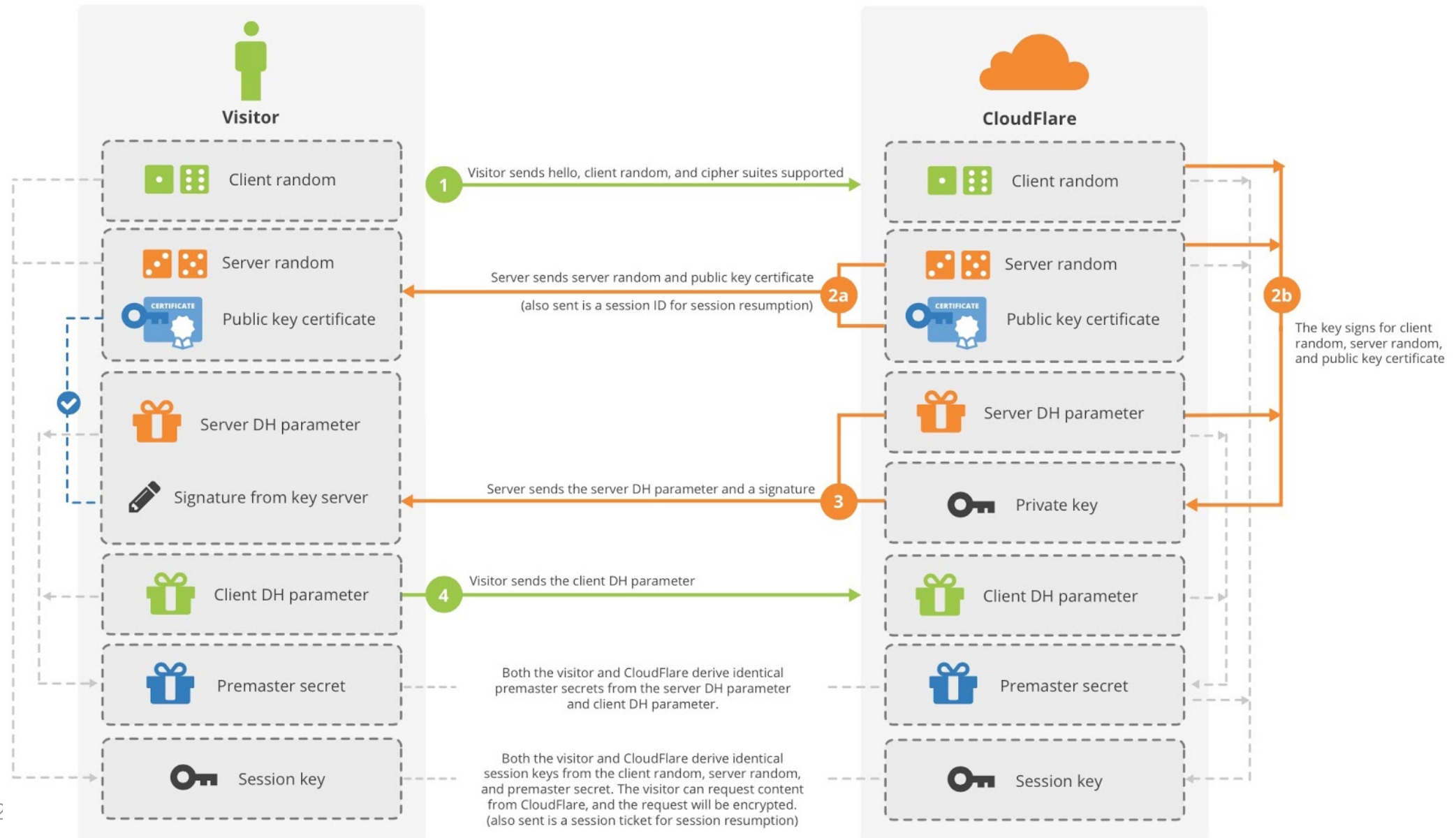
鮑伯的
私有密鑰

PKI公钥基础设施



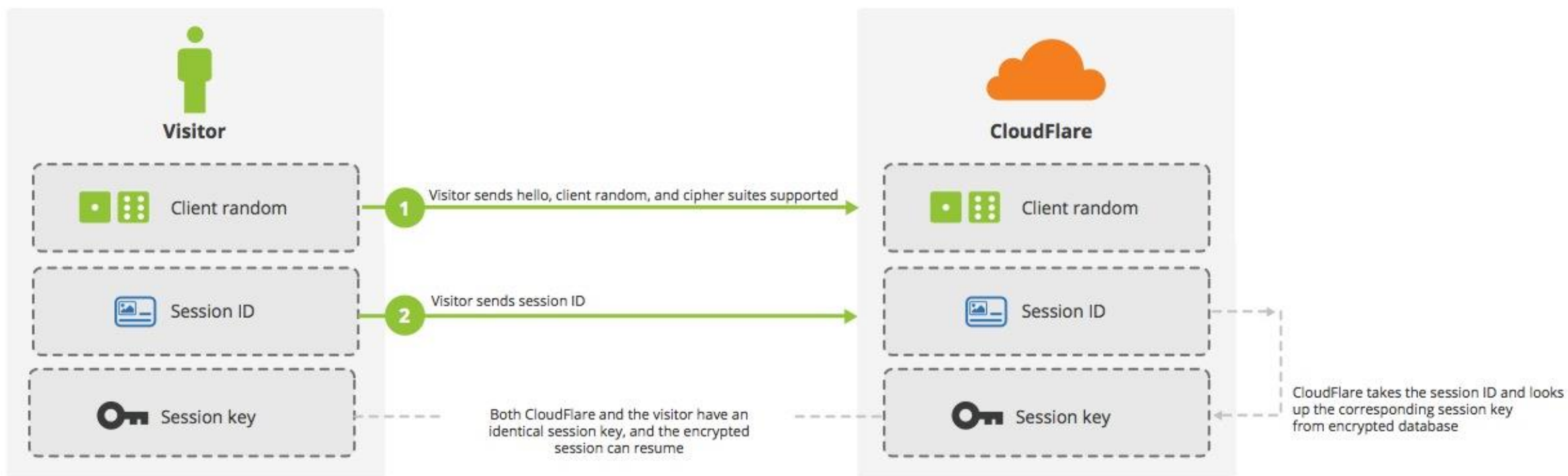
SSL Handshake (Diffie-Hellman)

Handshake



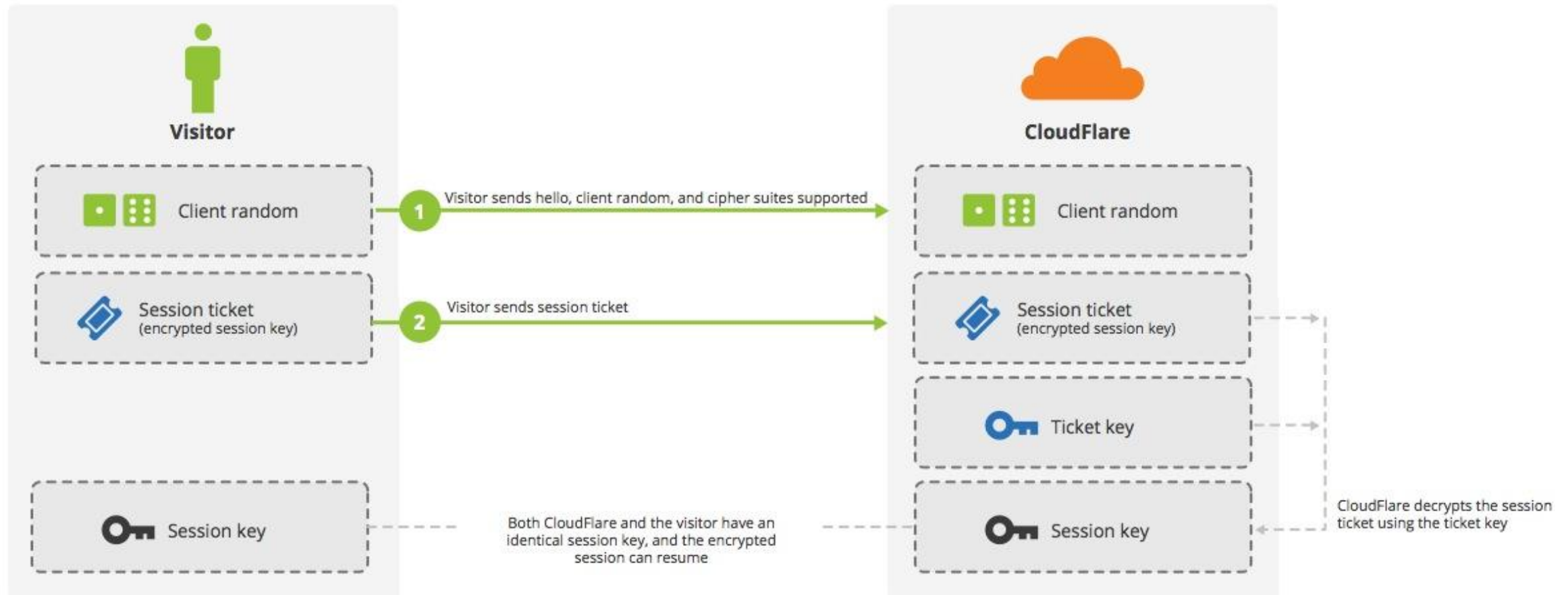
session缓存：以服务器生成的session

Session resume with session ID

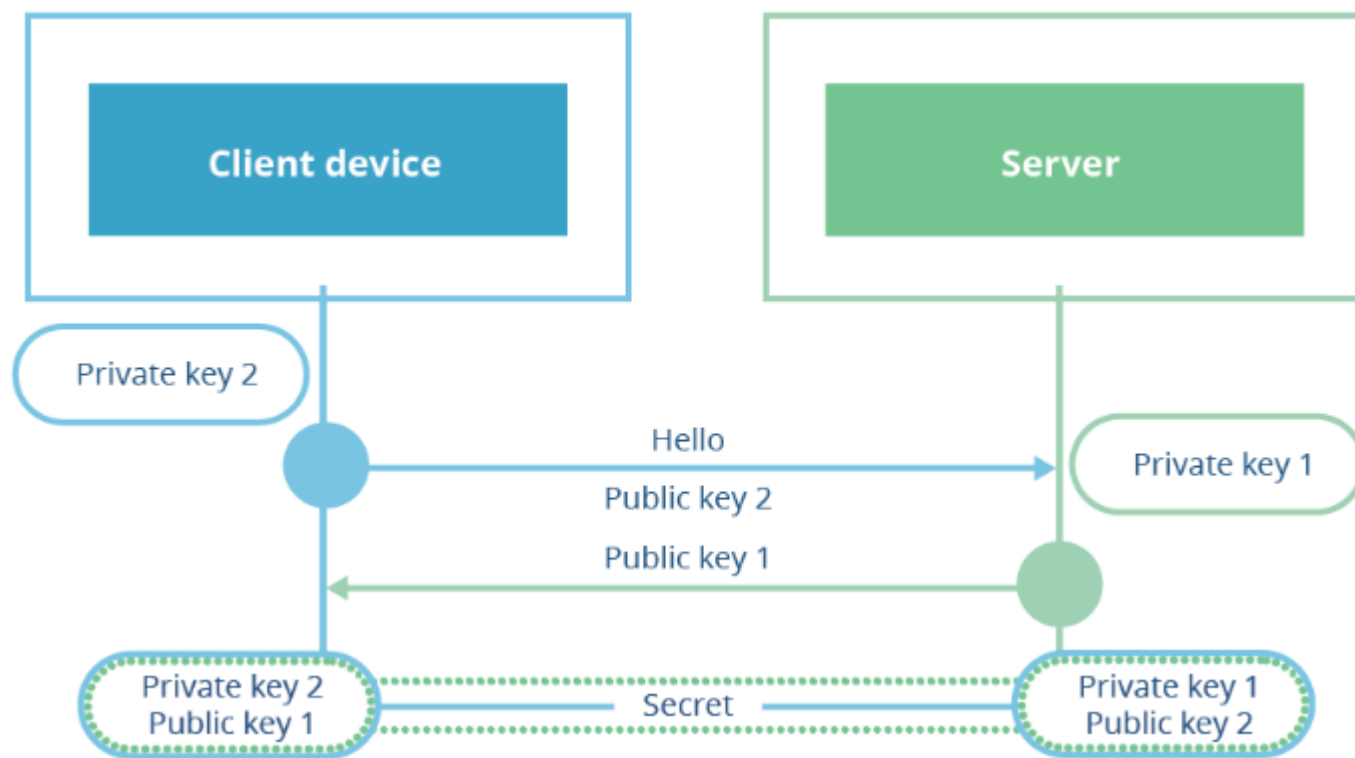


session ticket

Session resume with session ticket



TLS1.3密钥交换



谢谢

