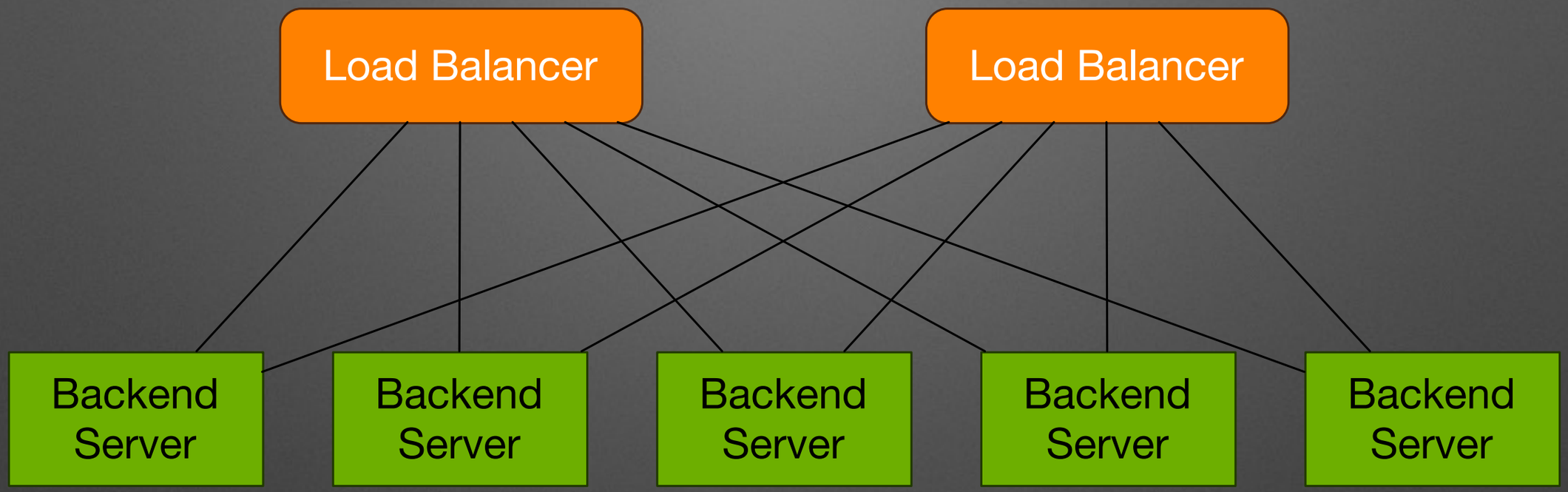
分布式系统架构设计思路

李雨来 @ SpeedyCloud

SFDC SegmentFault Developer Conference

为什么要使用分布式架构?

一台服务器处理不了这么大量的请求！



负载均衡

Nginx，HAProxy，LVS各自的优势与坑

负载均衡选型

- HAProxy

- Nginx

- LVS（ip_vs）

HAProxy和Nginx

- 7层负载均衡
- 支持后端健康探测
  - 可支持TCP探测和HTTP探测
- 通过HTTP Keepalive可以收敛后端服务器连接数

```
[root@centos7 builder]# netstat -a | grep icap | grep TIME_WAIT
tcp        0      0 localhost:icap          localhost:42341         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42363         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42409         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42407         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42325         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42343         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42350         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42146         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42236         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42372         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42362         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42340         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42417         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42339         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42405         TIME_WAIT
tcp        0      0 localhost:icap          localhost:42342         TIME_WAIT
[root@centos7 builder]#
```

TIME_WAIT

TCP连接的TIME_WAIT

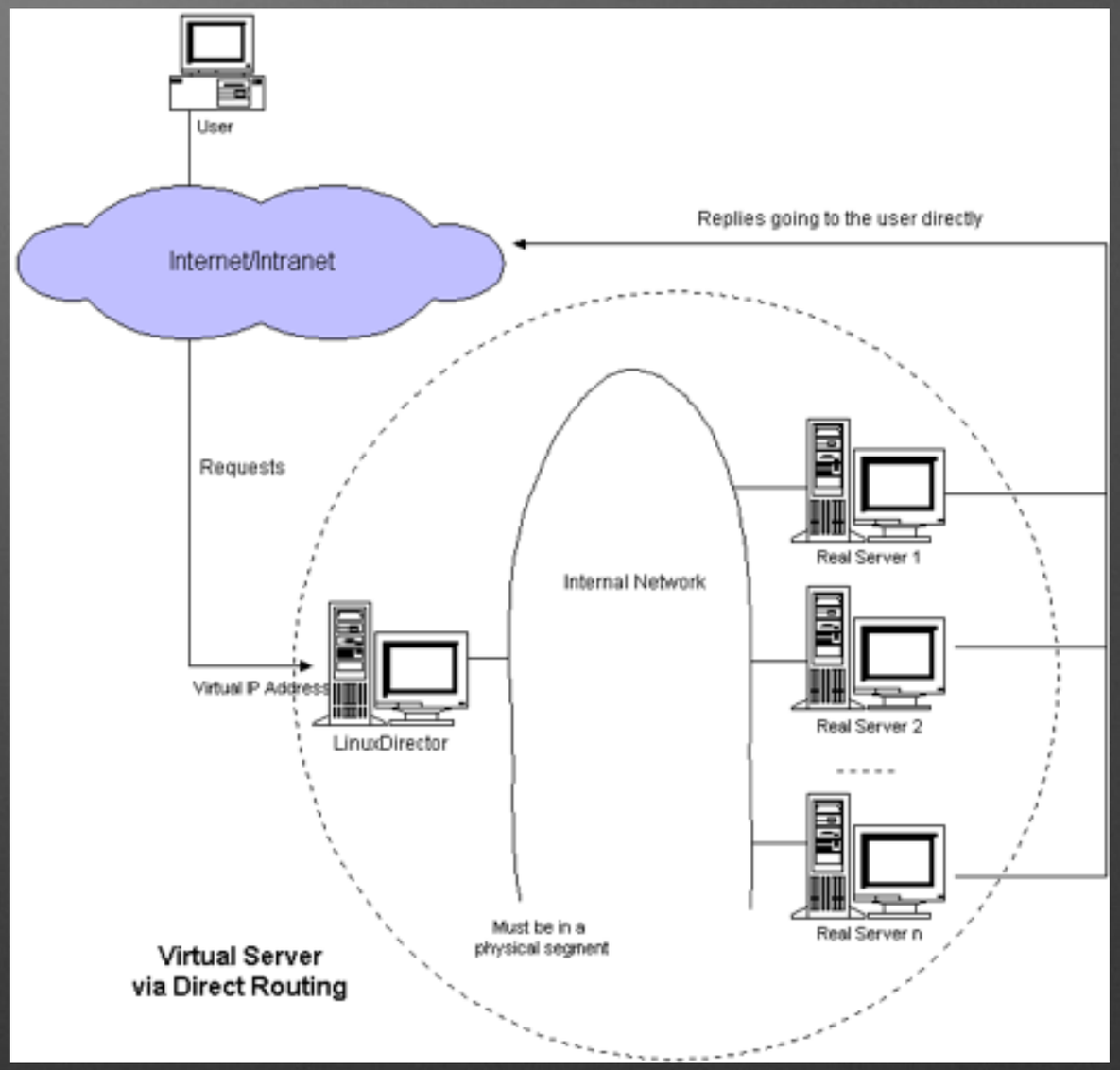
- 服务器作为客户端向外发起TCP连接的时候也要占用端口号
- cat /proc/sys/net/ipv4/ip_local_port_range
  - Linux默认值：32768　61000 = 28232个端口号
- cat /proc/sys/net/ipv4/tcp_fin_timeout
  - Linux默认值：60
- (61000 - 32768) / 60 = 470

TCP连接的TIME_WAIT

- 调优方法
  - echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
  - echo 15000 65000 > /proc/sys/net/ipv4/ip_local_port_range
- 有风险调优方法
  - echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
  - echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse

LVS

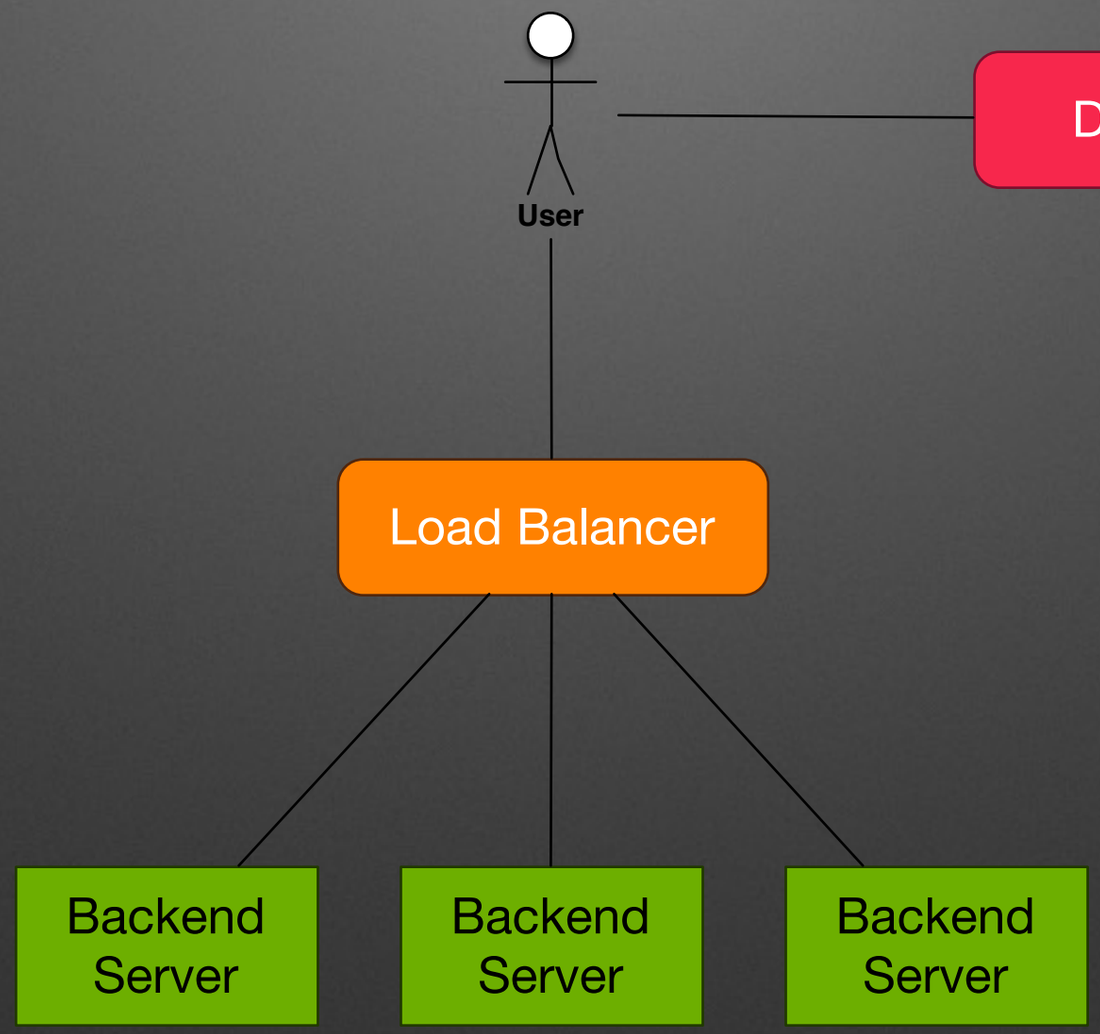
- 4层负载均衡
- ip_vs内核模块
- ip_vs负载均衡模式
  - NAT
  - TUN (IP Tunneling)
  - DR
- 实现的核心思想：MAC地址欺骗



LVS

- 对后端服务器没有直接的TCP连接，避免了TIME_WAIT问题
- TUN和DR模式非常适合小请求大响应的业务，响应的发送被分散到后端服务器直接对外
- NAT模式受限于服务器的带宽和包转发能力
- TUN和DR模式受限于入口IP服务器的上行流量处理能力

一个网络请求的过程

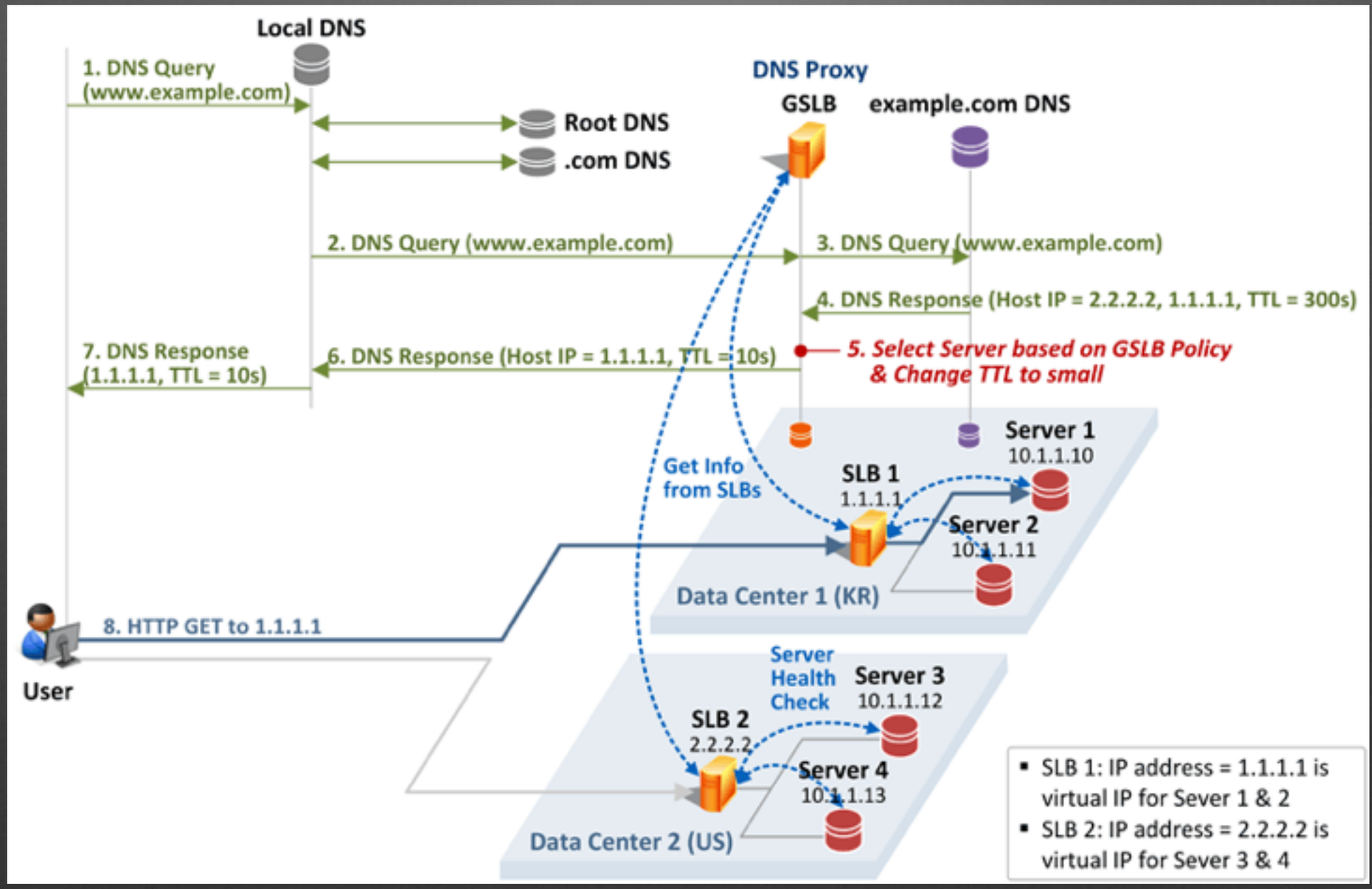

User
DNS
Load Balancer
Backend Server
Backend Server
Backend Server

智能DNS与多A记录

```
; <<>> DiG 9.8.3-P1 <<>> www.kernel.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20781
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.kernel.org.                        IN      A

;; ANSWER SECTION:
www.kernel.org.         600     IN      CNAME   pub.all.kernel.org.
pub.all.kernel.org.     600     IN      A       199.204.44.194
pub.all.kernel.org.     600     IN      A       149.20.4.69
pub.all.kernel.org.     600     IN      A       198.145.20.140

;; Query time: 482 msec
;; SERVER: 114.114.114.114#53(114.114.114.114)
;; WHEN: Mon Nov 28 15:57:23 2016
;; MSG SIZE  rcvd: 102
```

- 多条A记录可以把请求分散到多个IP地址

CDN的启发



- 通过GSLB调度用户的请求

CDN的启发

- 通过调度系统分散用户请求
- 基于元数据调度
- 基于Hash算法调度

降低单次请求的时间

- 大部分情况下影响一个请求的响应时间最大的热点在IO
- 缓存是公认的提升响应速度的一个方法
- 通过APM软件监控程序瓶颈点

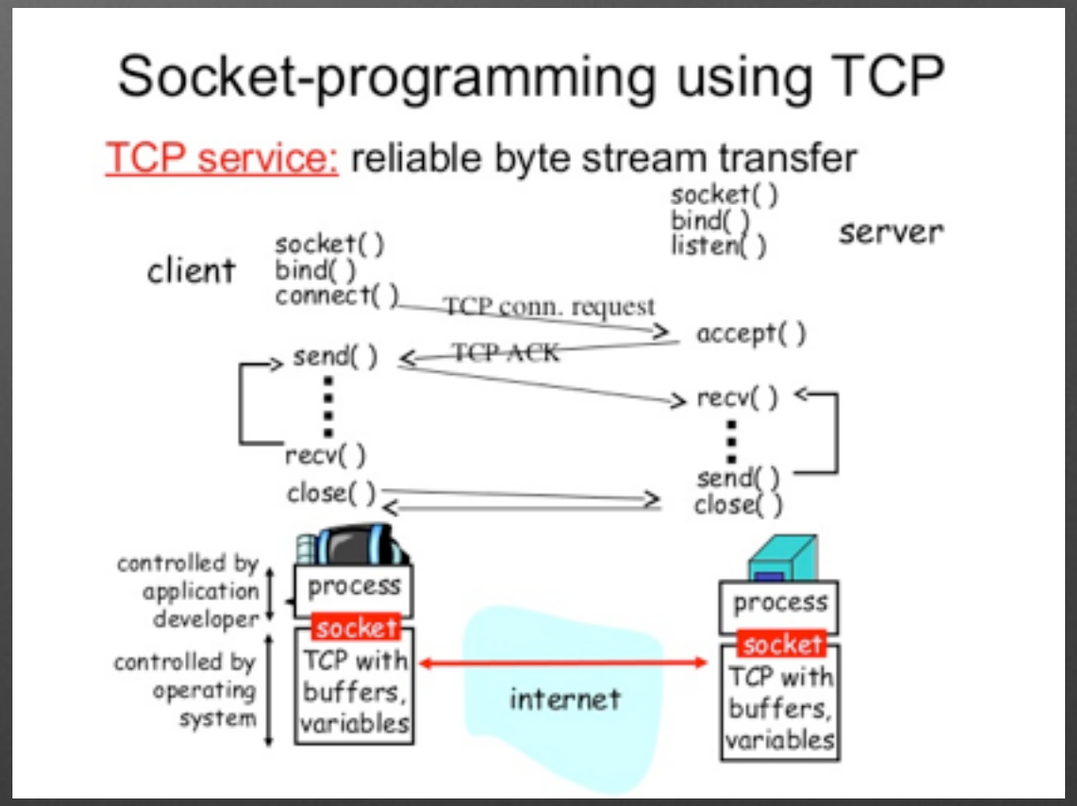
横向扩展与纵向扩展

- 横向扩展：适合无状态服务（Web Server）
- 纵向扩展：适合有状态服务（数据库）
- 有状态服务的横向扩展需要在CAP上做权衡
  - HBase： CP，Cassandra：AP ……

一连接一线程与IO多路复用

一连接一线程

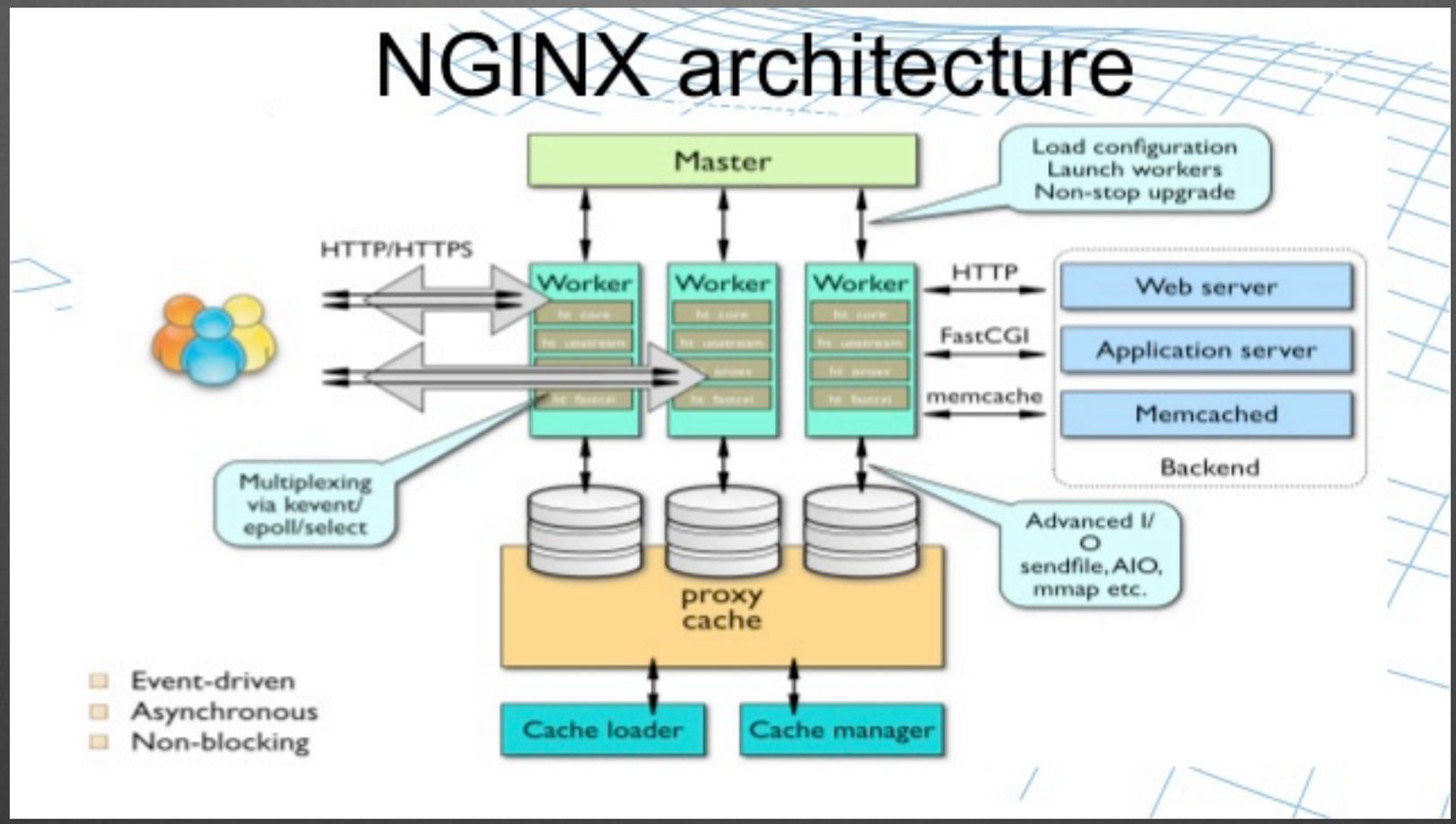
- 顺序编程模型
- recv()，send()，read()，write()调用会出发内核的线程切换
- 请求数太多会导致Context Switch频繁，降低系统处理能力



IO多路复用（IO Multiplexing）

- 非阻塞事件编程模型
- 单线程可处理大量并发连接，同时不需要考虑并发加锁问题
- 对CPU密集型业务支持不是很好，需要突破单线程限制

混合模式



- 一个线程（进程）负责异步accept
- 多个Worker线程（进程）负责处理连接

混合模式

- 可以充分利用多核
- CPU的Context Switch适中
- Linux 3.9 SO_REUSEPORT
  - 多进程同时监听一个端口
  - Kernel负责连接的负载均衡

Server端编程的坑

- 非阻塞IO夹杂阻塞IO调用
  - 后果：整个线程被阻塞IO卡住导致无法处理新的请求

Server端编程的坑

- 针对MySQL，Redis没有使用连接池
  - 后果：
    - 每次请求都重新建连，占用MySQL，Redis一部分CPU处理能力
    - 压力过大的情况下会遇到大量TIME_WAIT问题

Server端编程的坑

- 注意MySQL，Redis，Memcached等服务程序的IO模型
- 大量连接会影响这些服务器程序处理能力
- MySQL是一连接一线程连接数过多会导致MySQL创建大量线程
- Redis，Memcached是单进程单线程异步IO模型，单个Server的处理能力上限与CPU个数无关

为故障设计

故障无处不在

- 没有100%的SLA
  - 网络故障
  - 服务器硬件故障
  - IDC机房故障
  - 不可抗拒力导致的故障
  - 压力太大导致系统处理不过来
  - 系统Bug

容灾的考虑

- IDC方面的考量：“风水”
- 硬件方面的考量：无单点故障
- 软件方面的考量：故障可隔离，快速失败和重试

系统间通信

- 任何通信都有可能失败
- 通信要有超时，并可以自动重试
- 调用保证幂等性
- 服务间通信要有多条路径，避免单点故障

性能问题与Bug

- 系统可拆分（微服务架构）
  - 任何一个API有性能问题时不至于影响整个系统
- 当程序遇到异常或者故障的时候可以自己报告出来

集群内部的协调

- 通过HAProxy，Nginx的后端探测做故障隔离
- ZooKeeper，etcd做集群内部协调和故障切换
- 注意：雪崩效应
  - 设置自动调度的阈值
  - 提供人工介入的方法

架构就是取舍，没有完美的架构，

只有不断演进的架构

谢谢