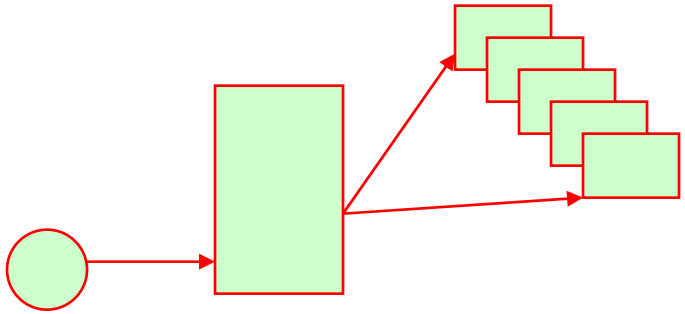
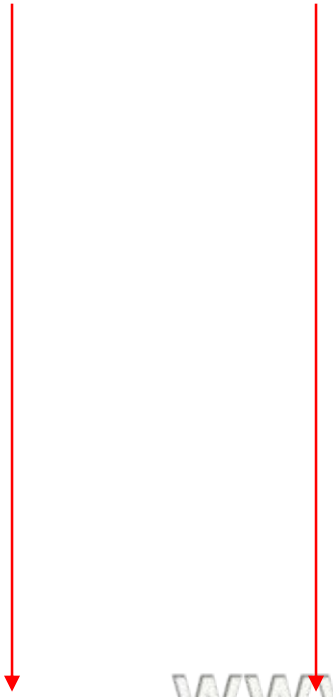
马哥教育

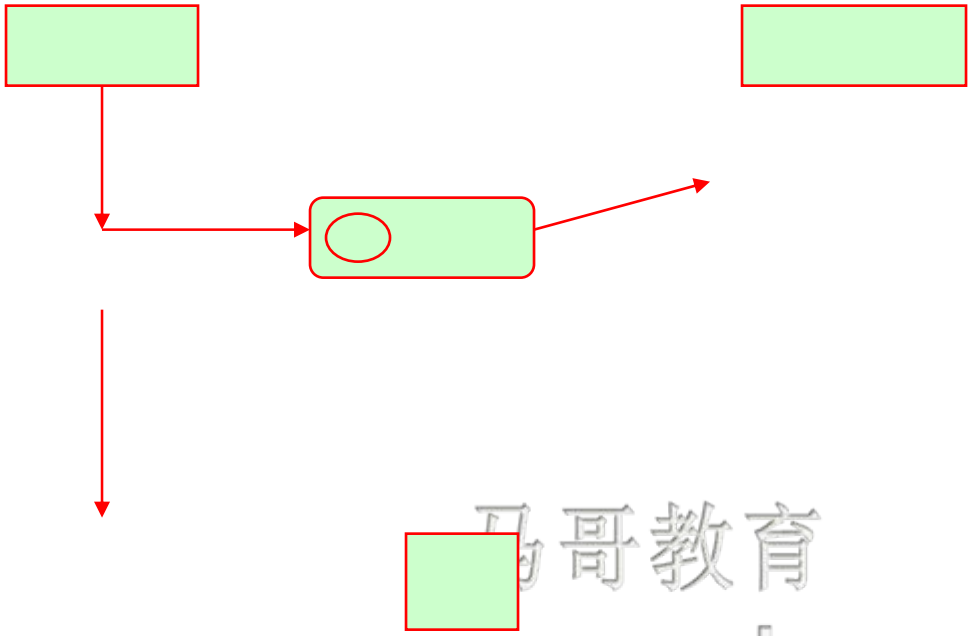
分布式系统基础

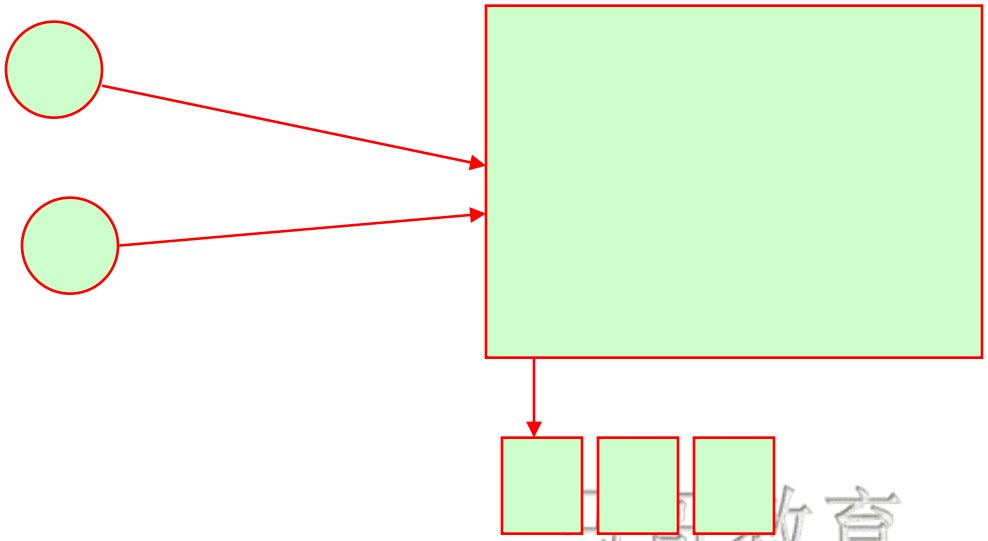
主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

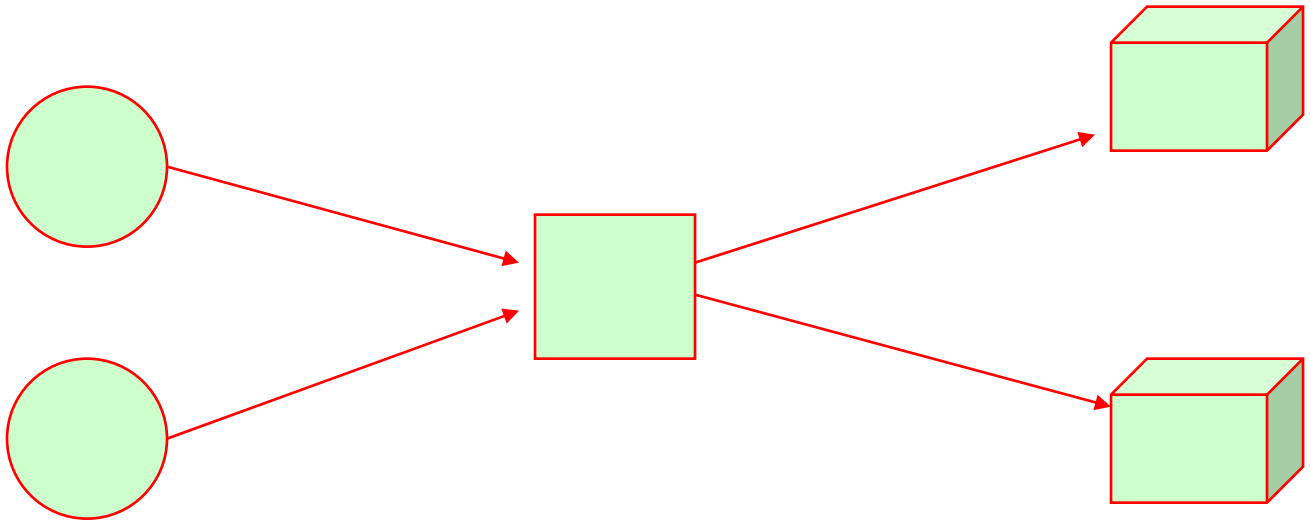


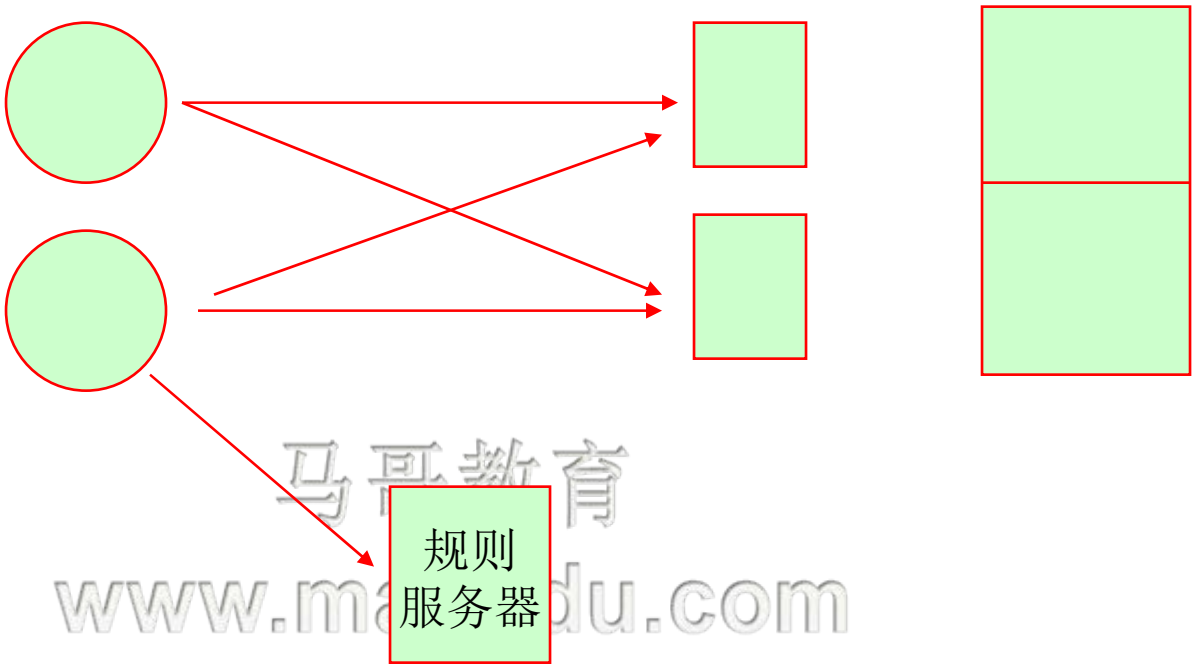


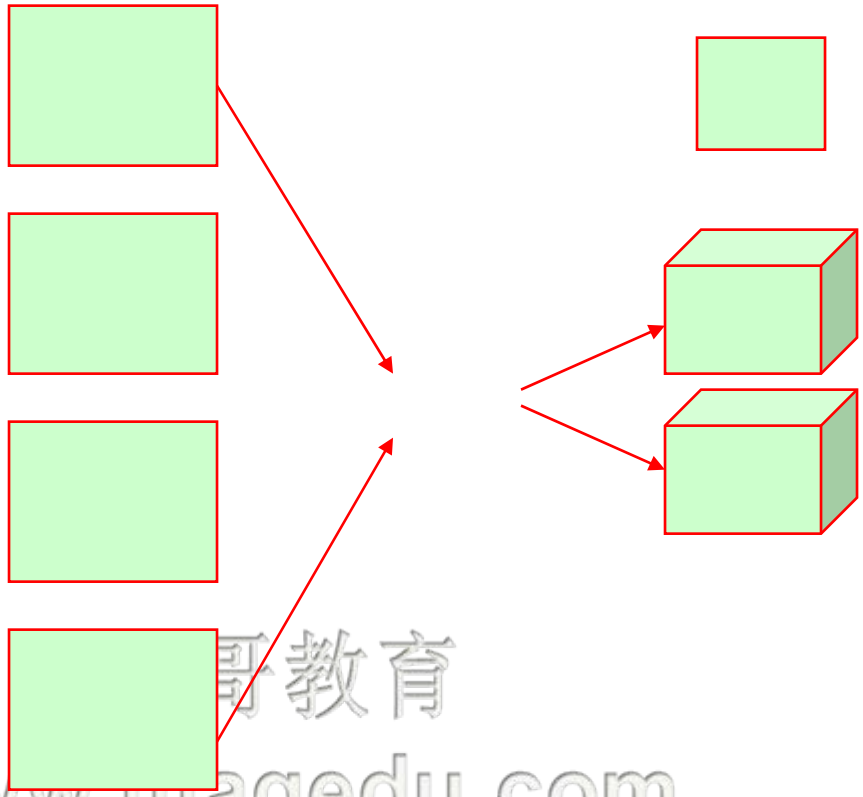
❖ COW：Copy On Write

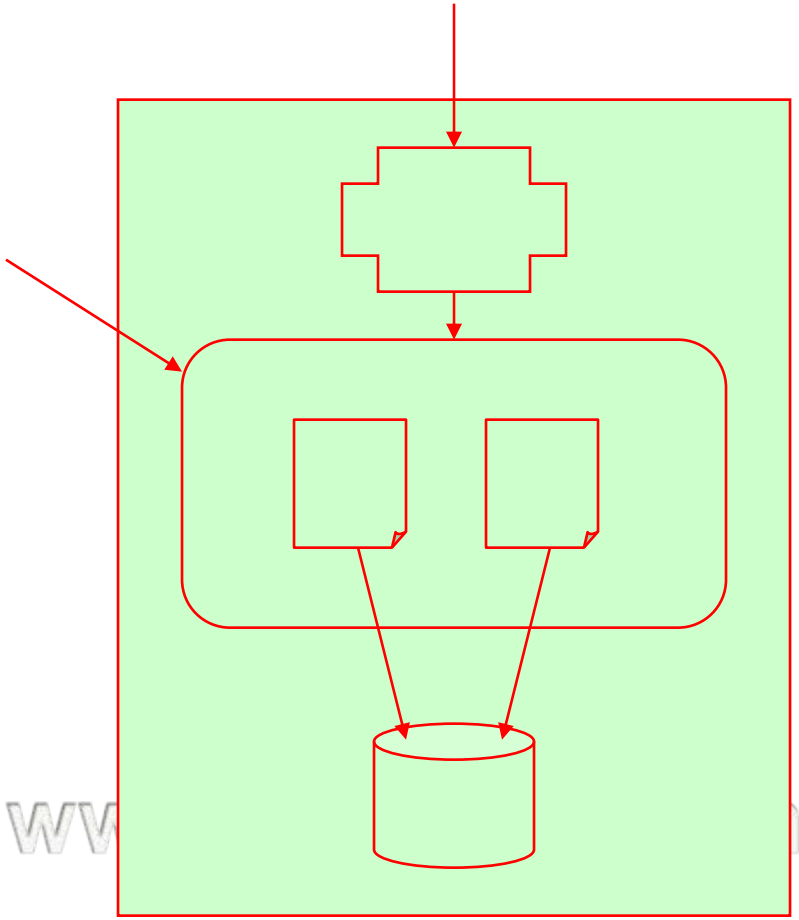




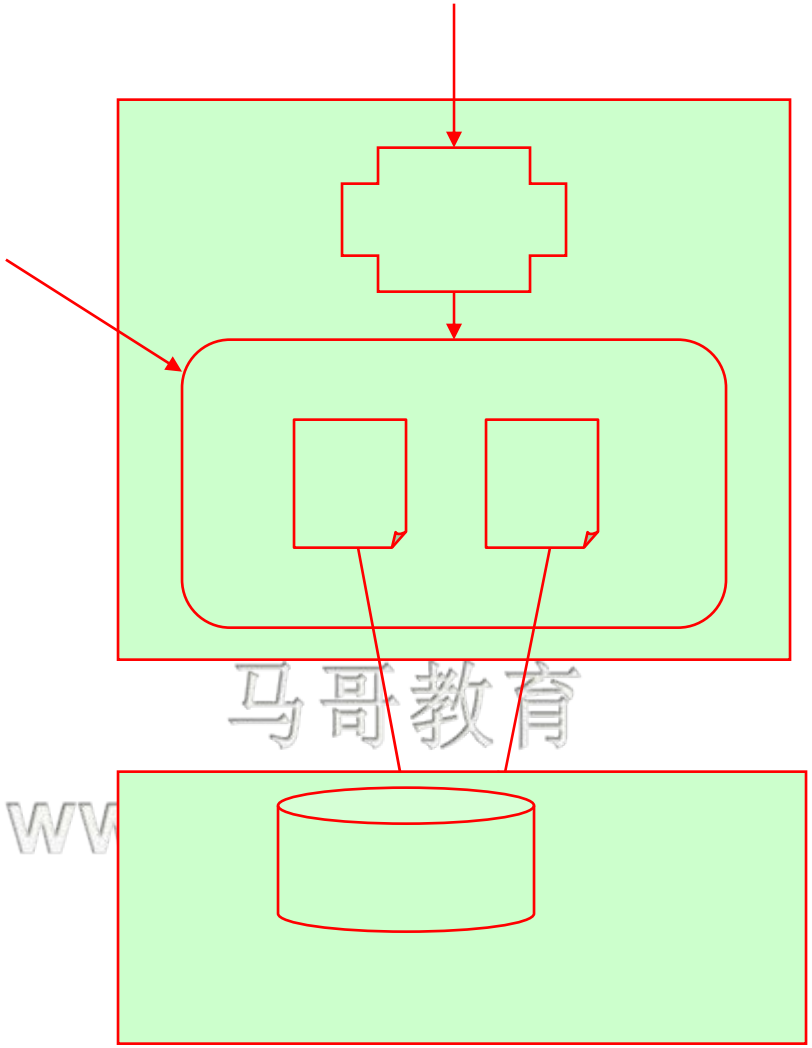




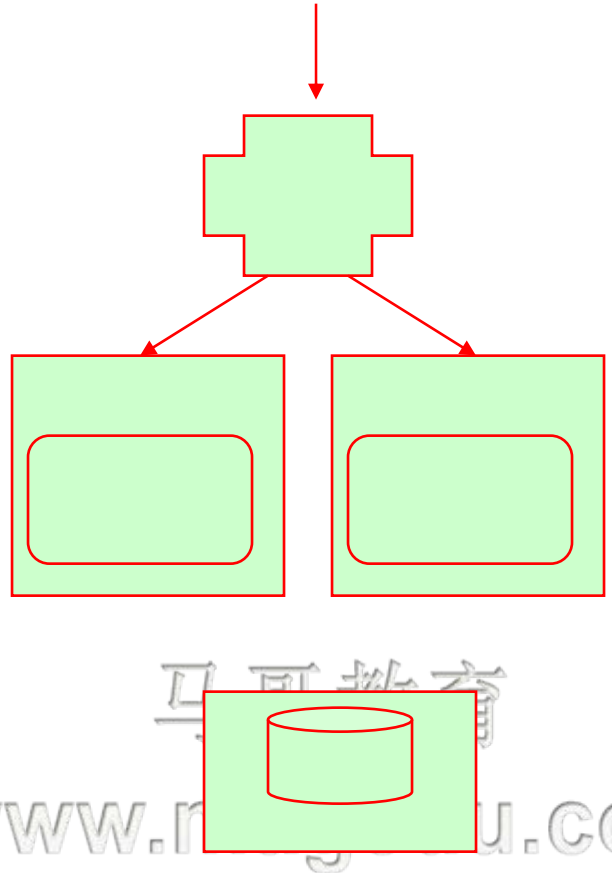




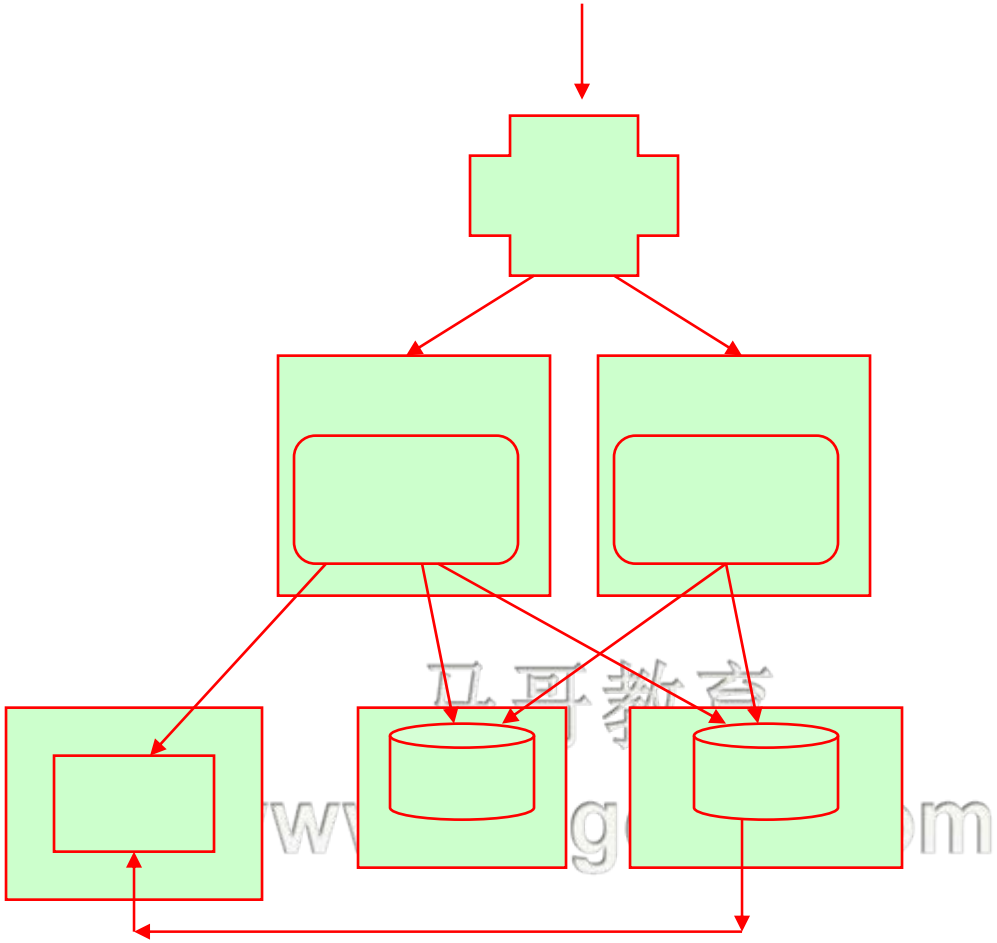
应用和数据库分离



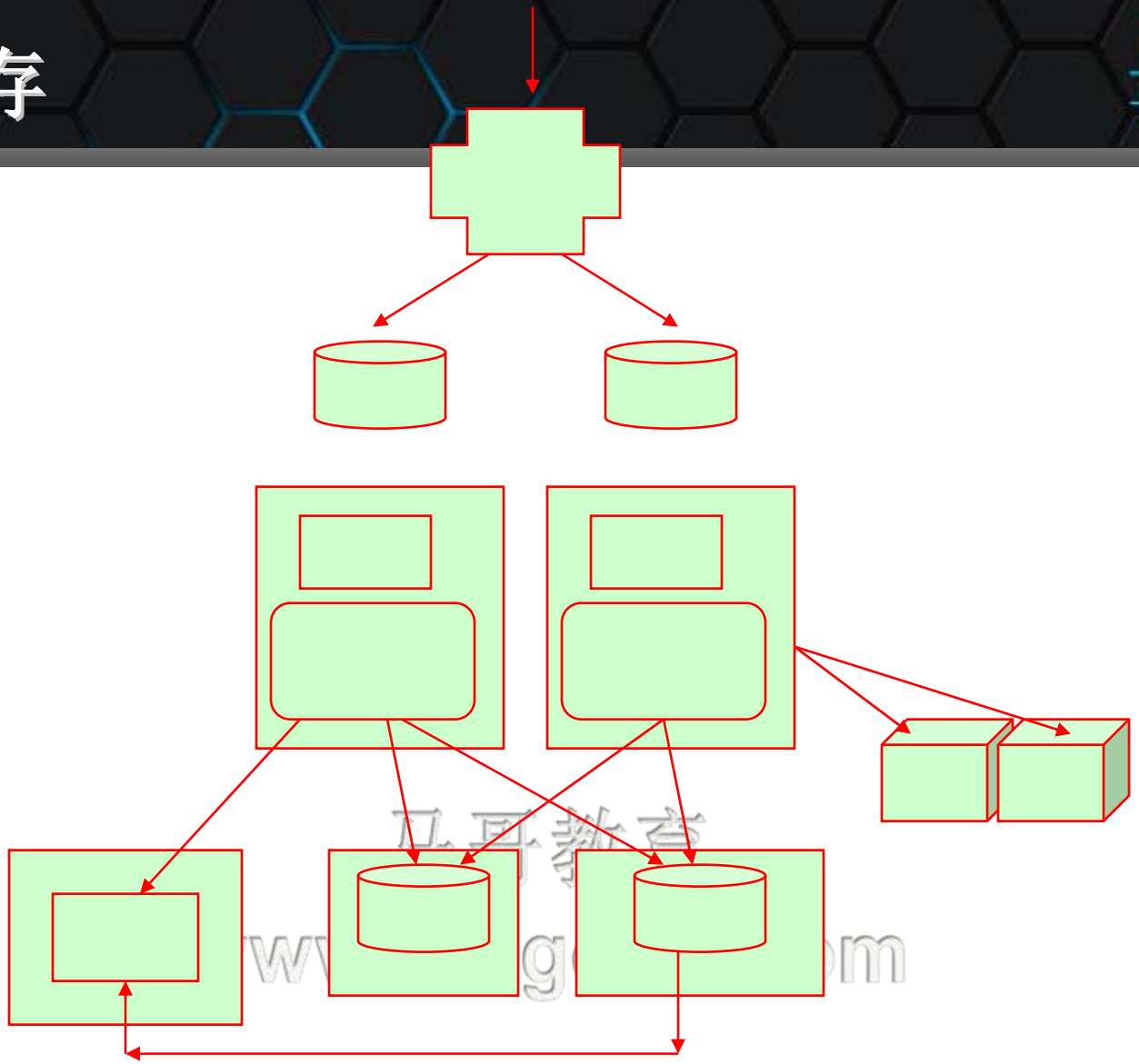
应用服务器负载均衡



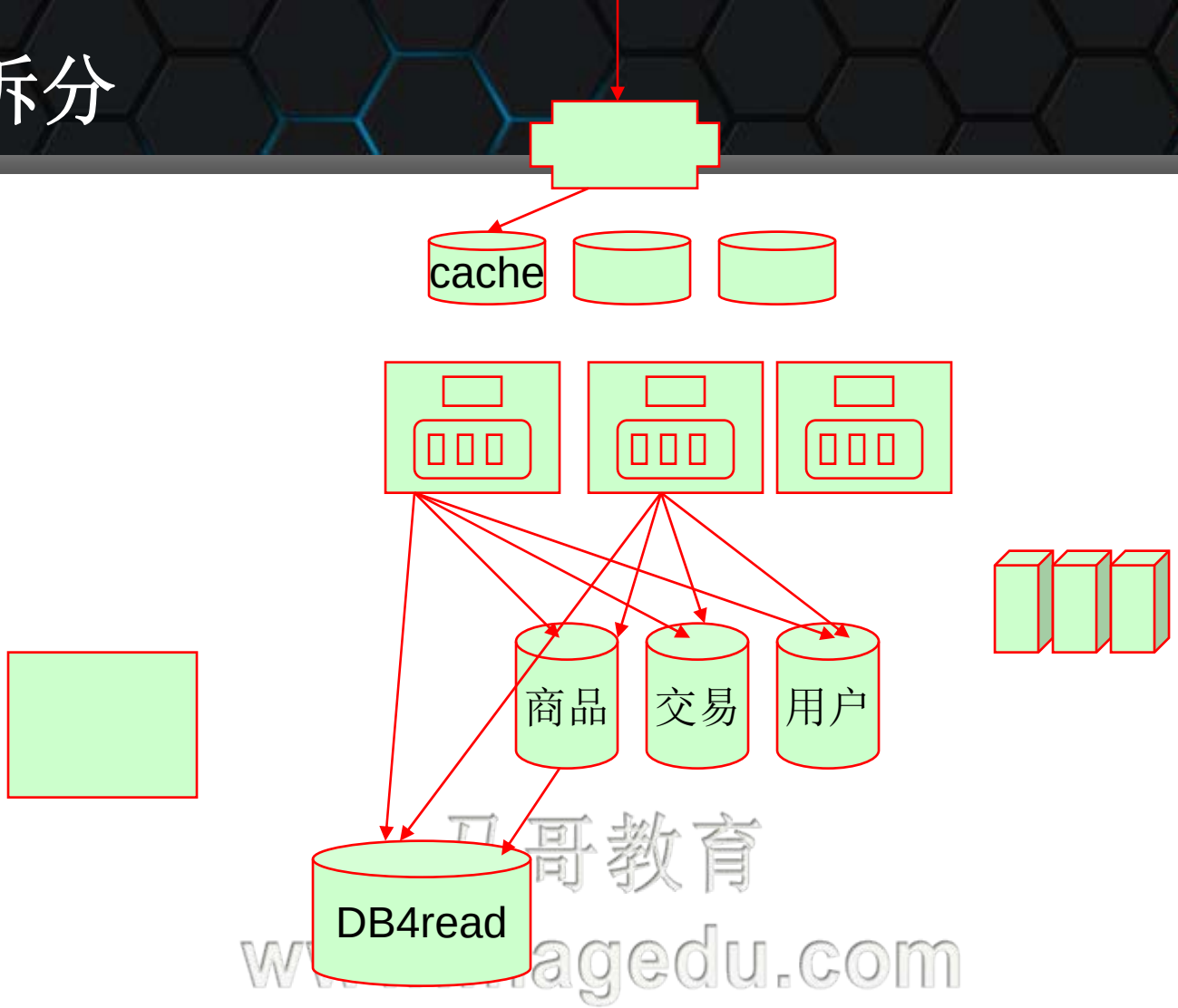
引入搜索引擎实现全文搜索



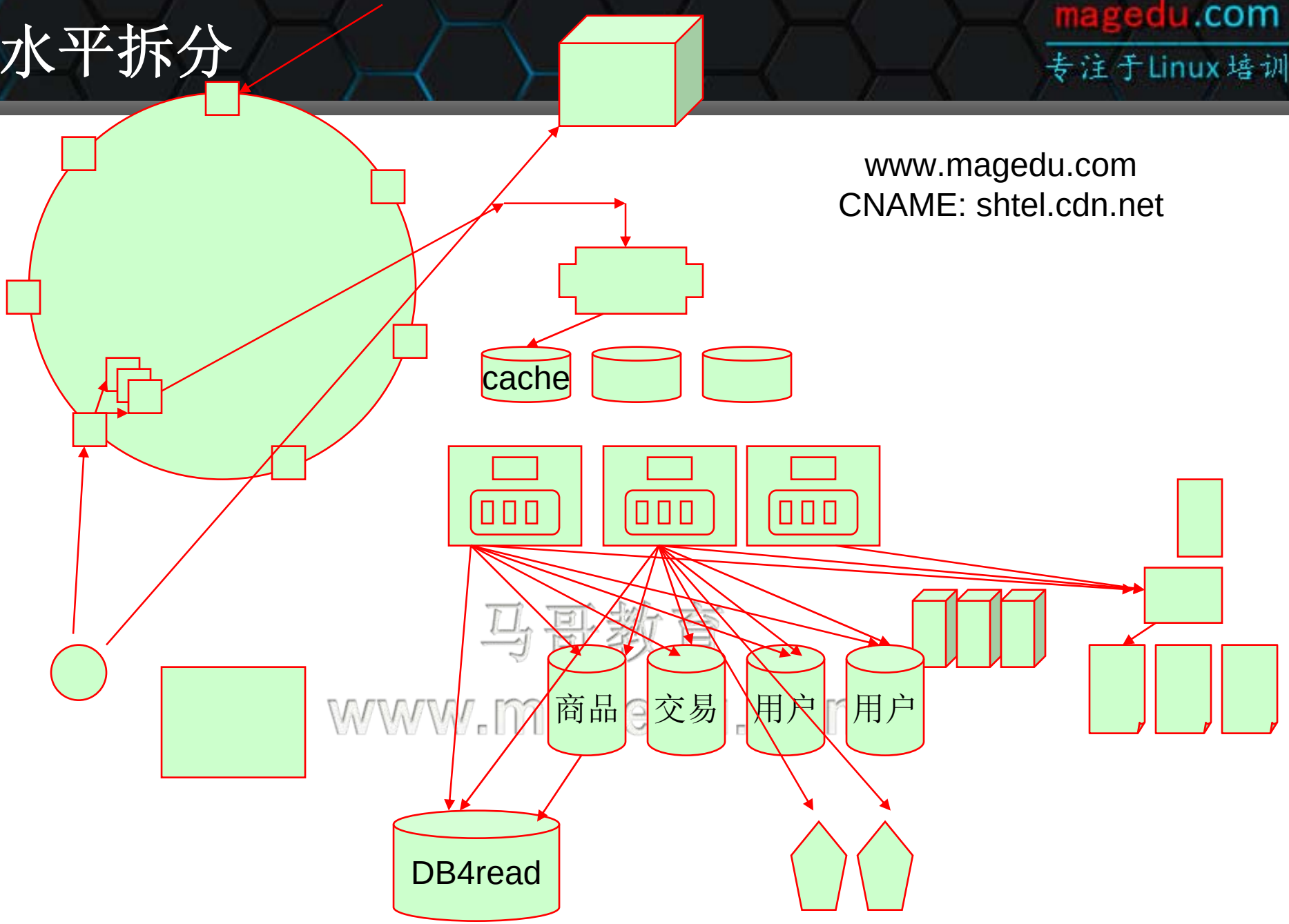
引入缓存



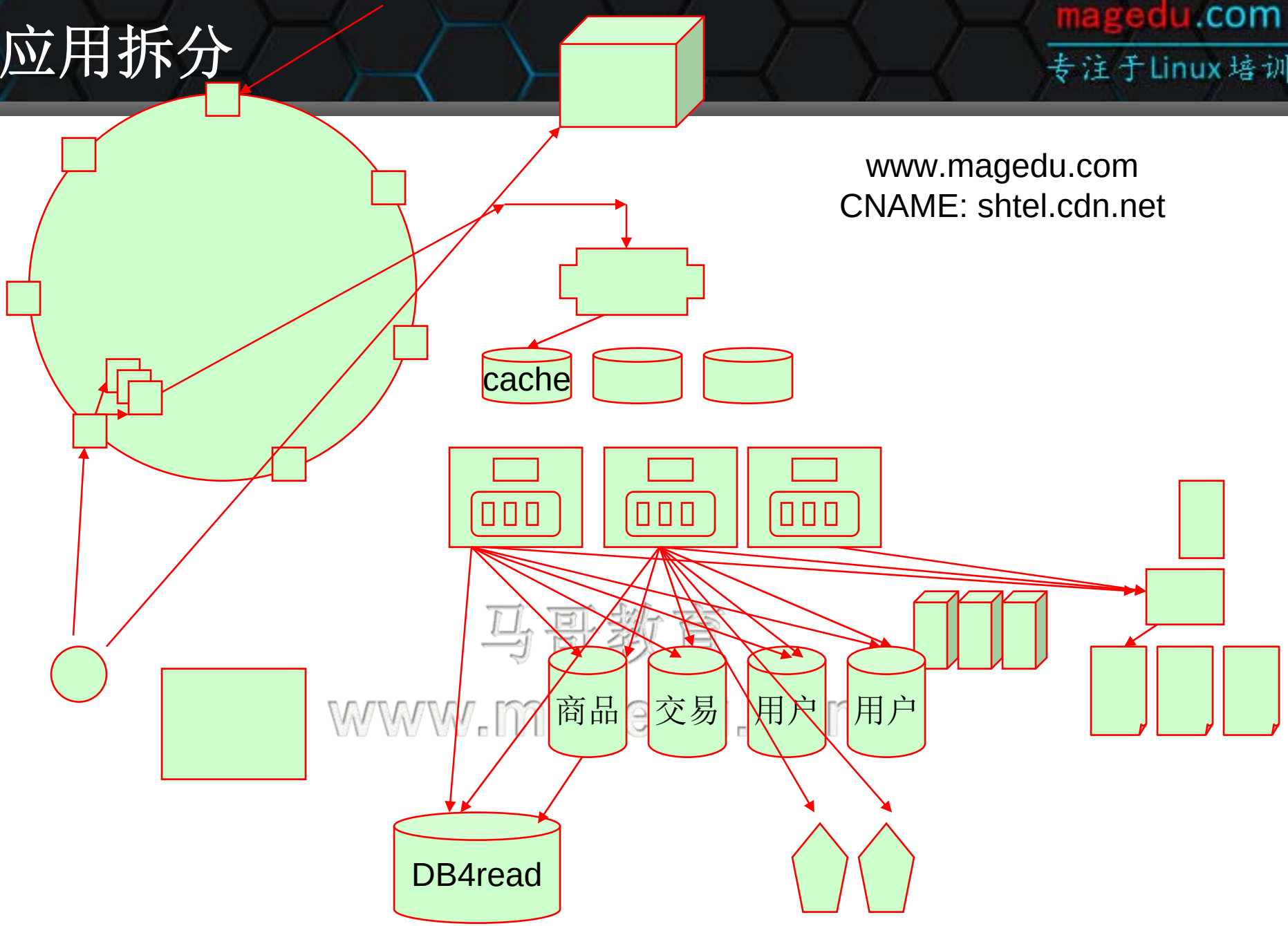
垂直拆分

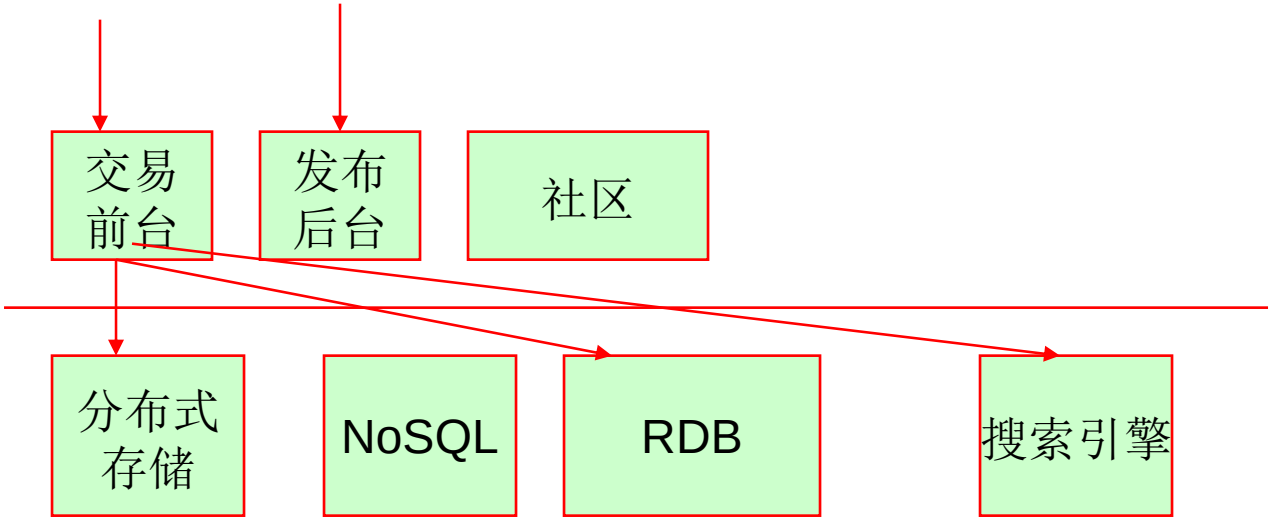


水平拆分

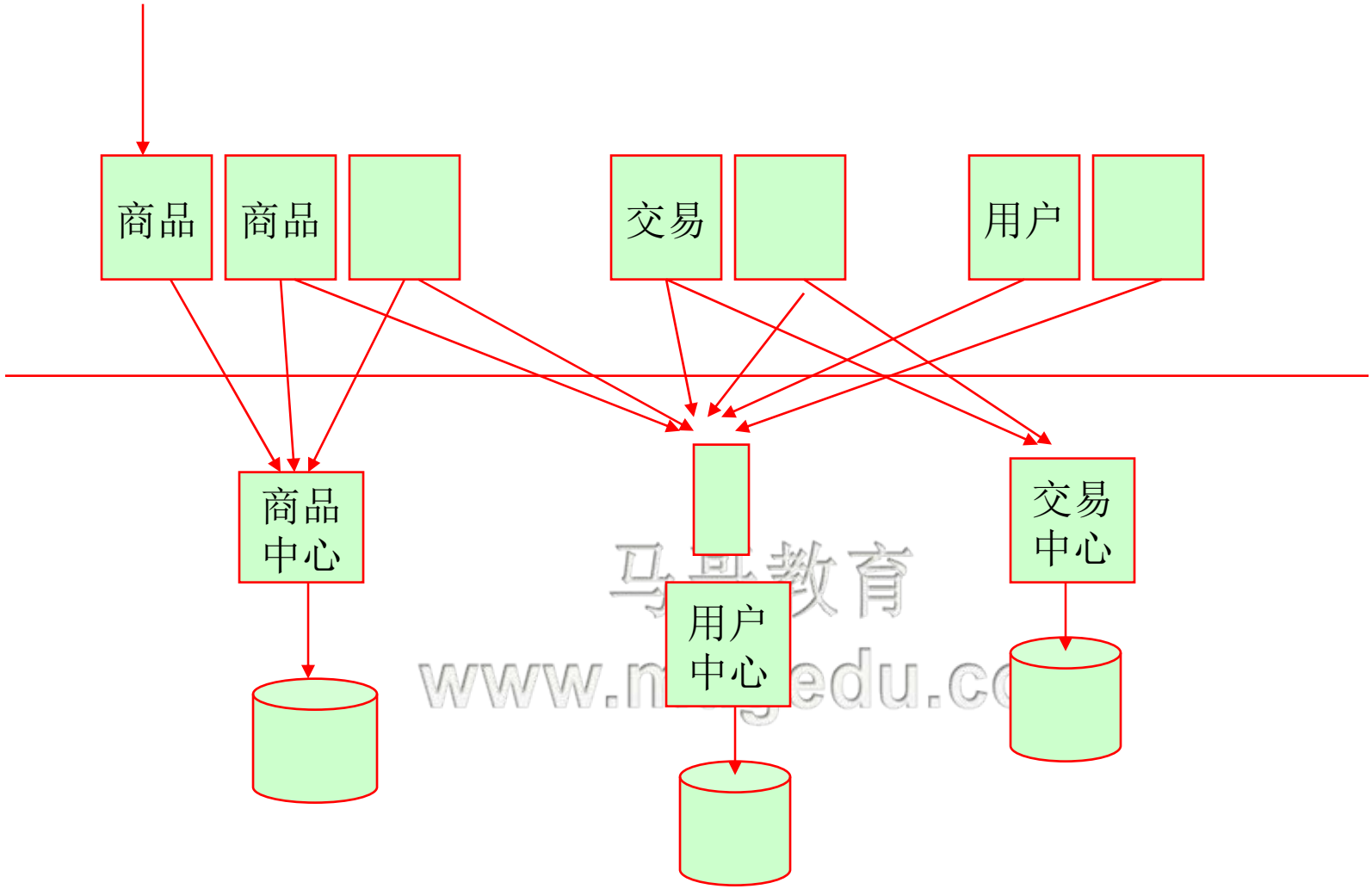


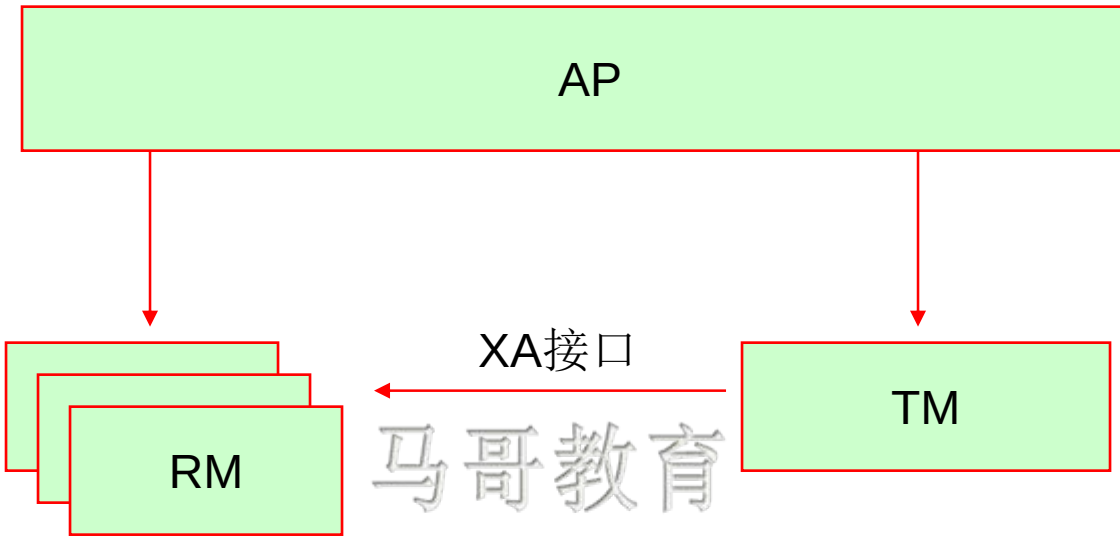
应用拆分

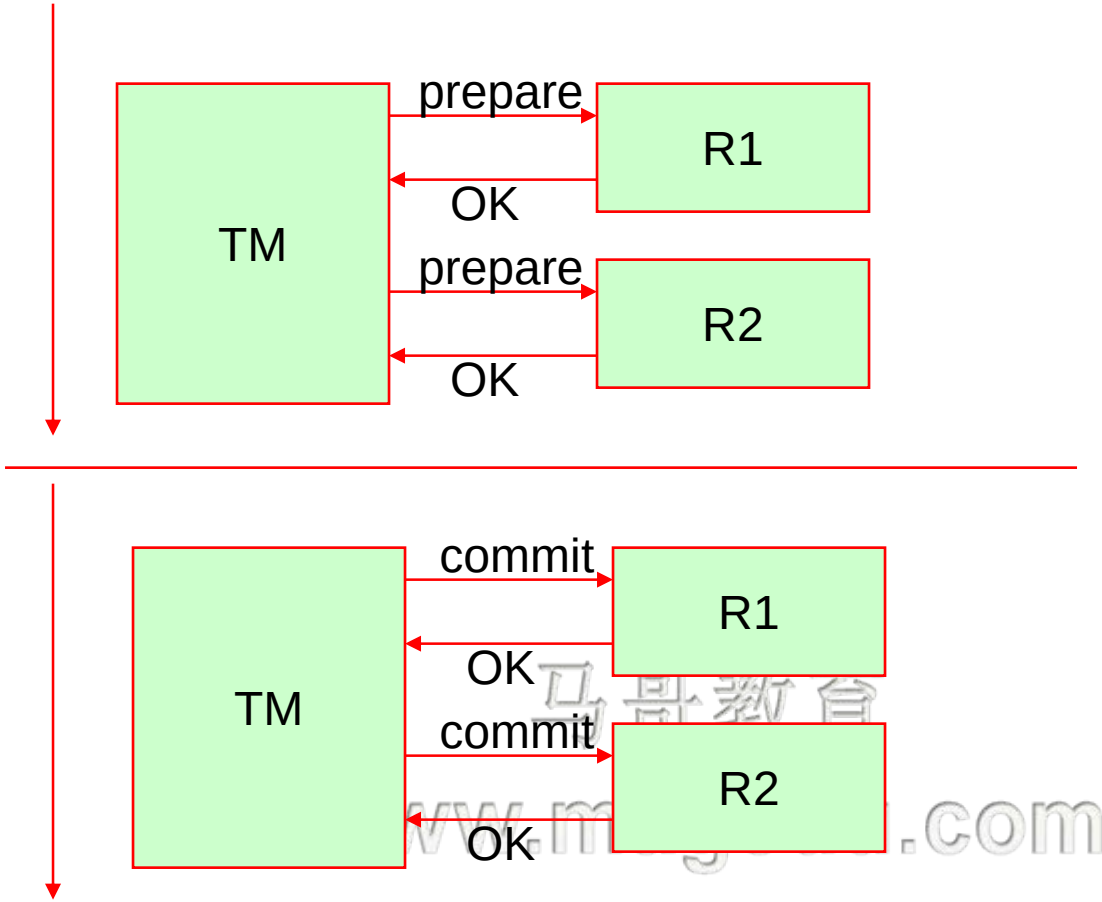




服务化





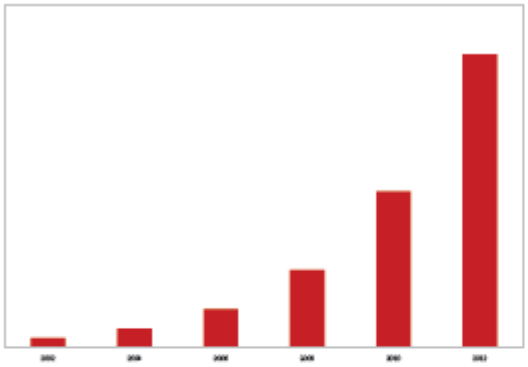


马哥教育

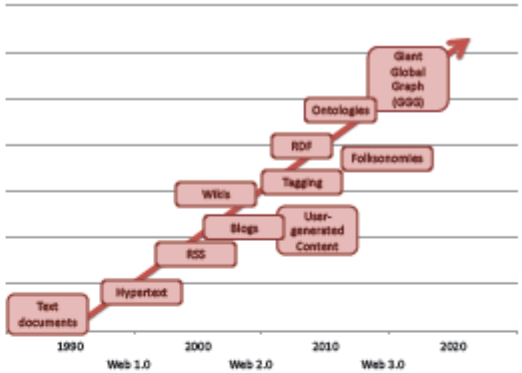
Distributed FS

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

数据存储的趋势

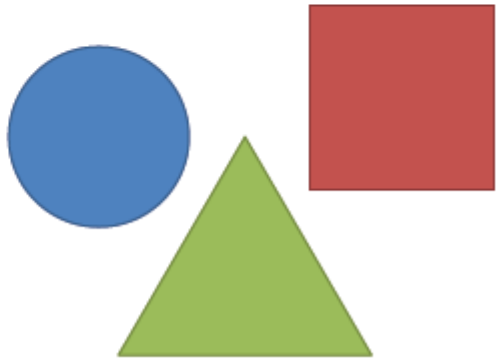


数据增长



信息连接

并发性



结构多样性

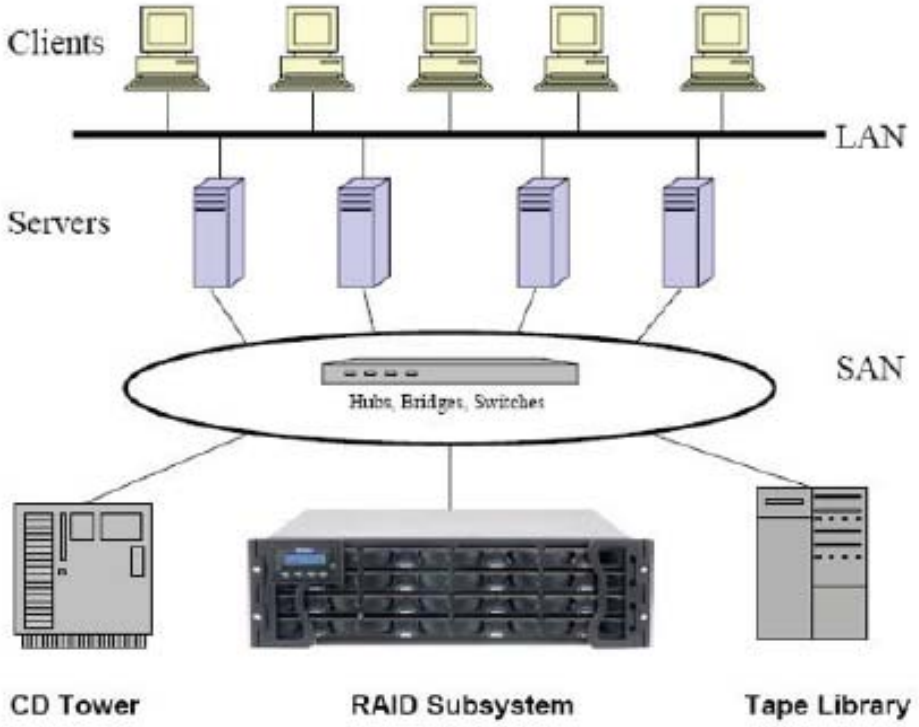
大数据带来的挑战

- 数据采集
- 数据存储
- 数据搜索
- 数据共享
- 数据传输
- 数据分析
- 数据可视化

大数据如何存储？

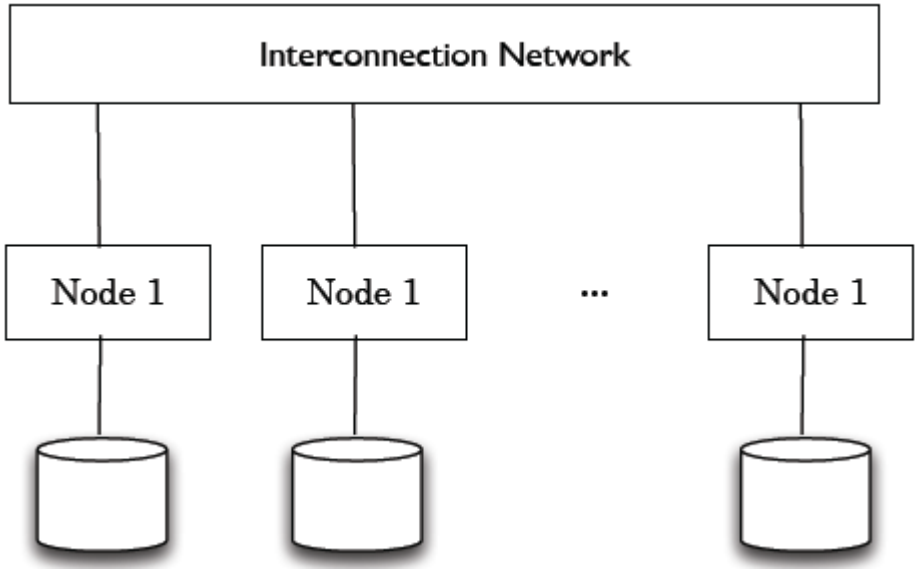
传统存储

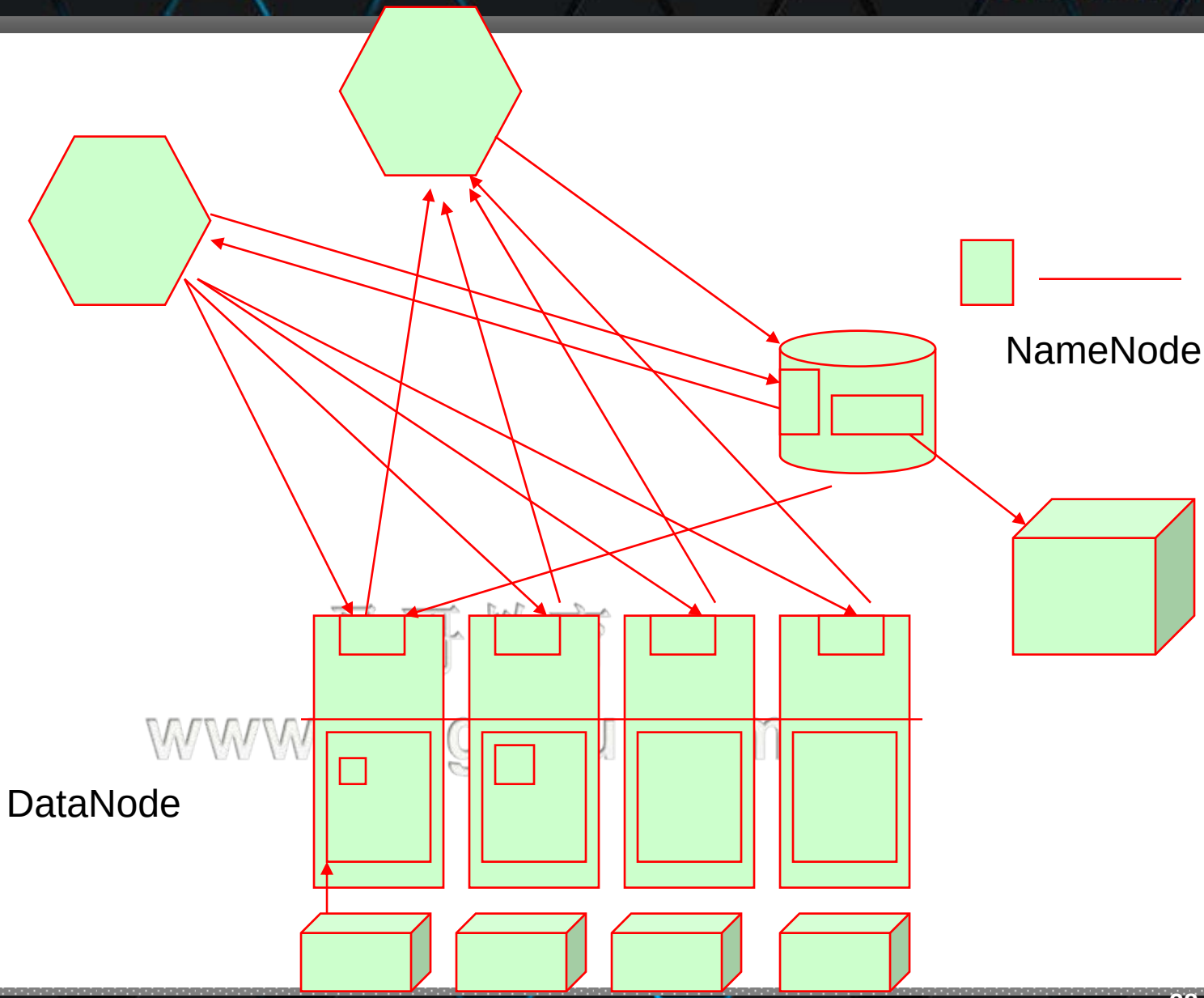
- 问题
  - 纵向扩展受阵列空间限制
  - 横向扩展受交换设备限制
  - 节点受文件系统限制



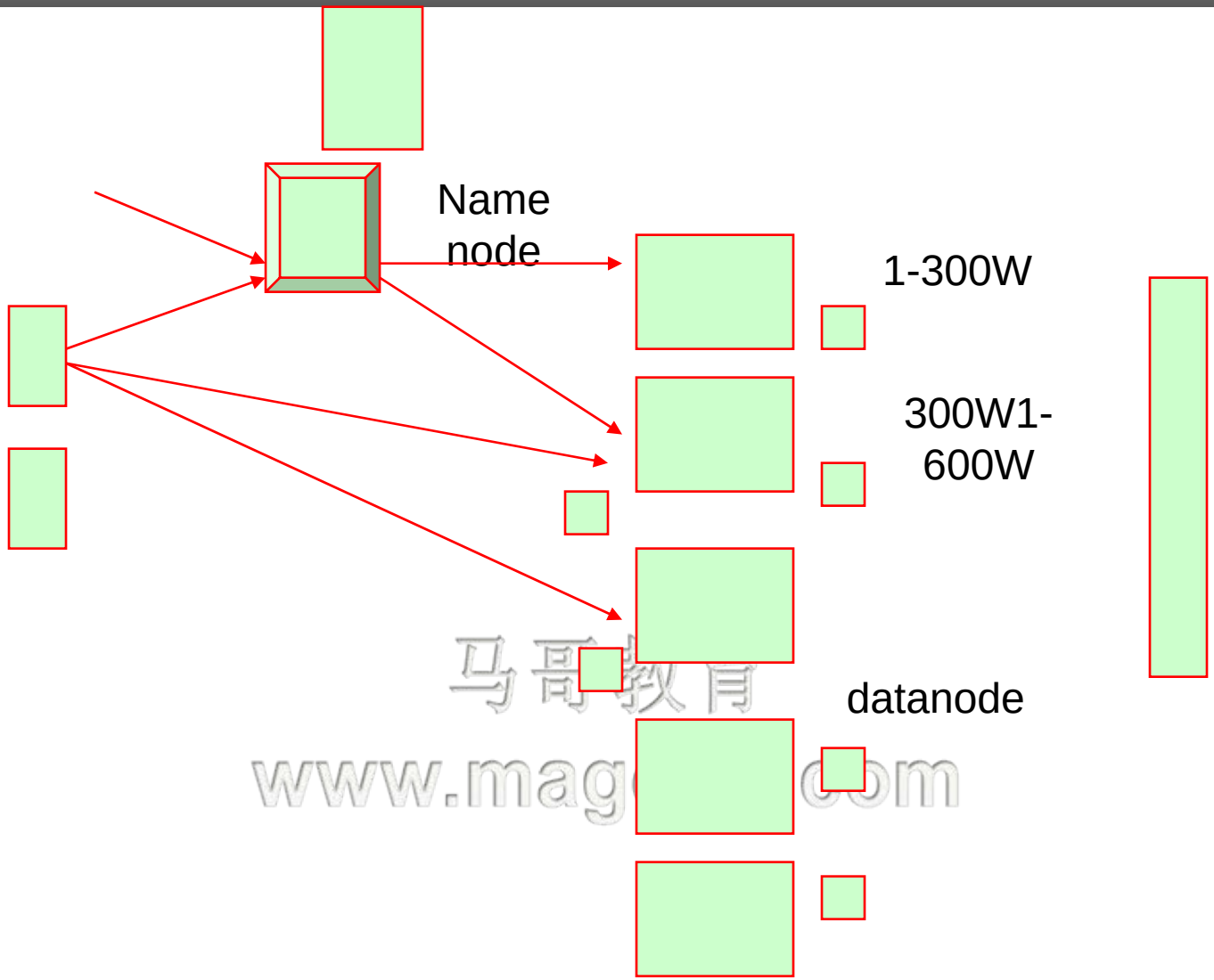
分布式存储

- 挑战
  - 节点间通信
  - 数据存储
  - 数据空间平衡
  - 容错
  - 文件系统支持




NameNode
DataNode

- 10



分布式文件系统设计目标

- 访问透明
- 位置透明
- 并发透明
- 失效透明
- 硬件透明
- 可扩展性
- 复制透明
- 迁移透明

Big File Systems?

1

Scalable

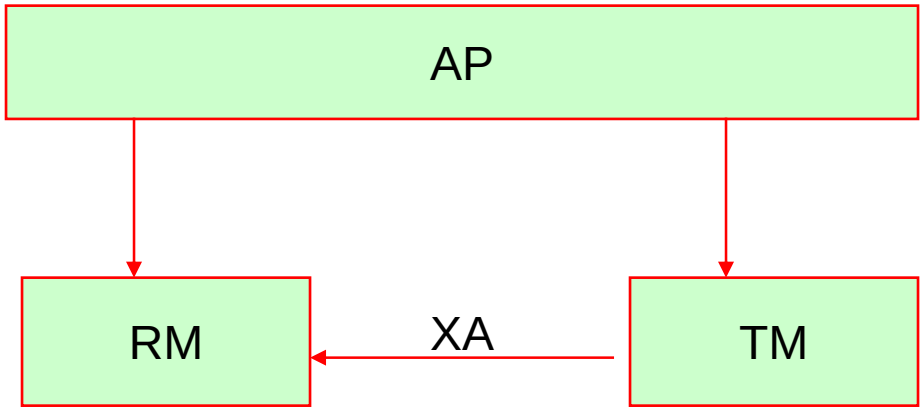
Looking at storage and serving infrastructures

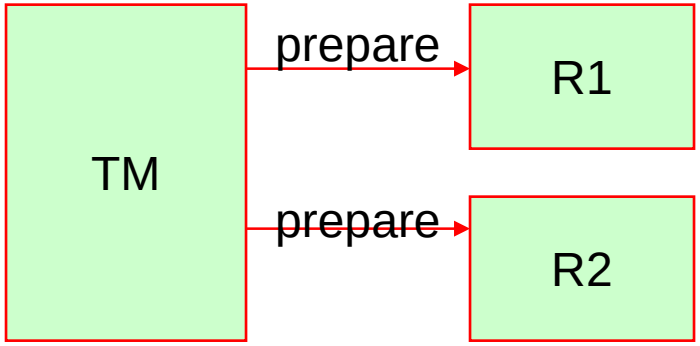
2 Reliable

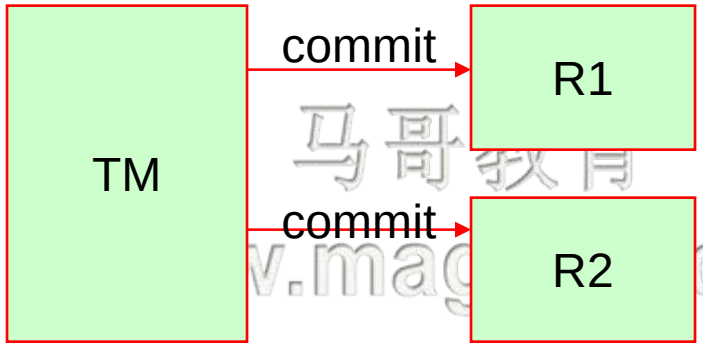
Looking at redundancy, failure rates, on the fly changes

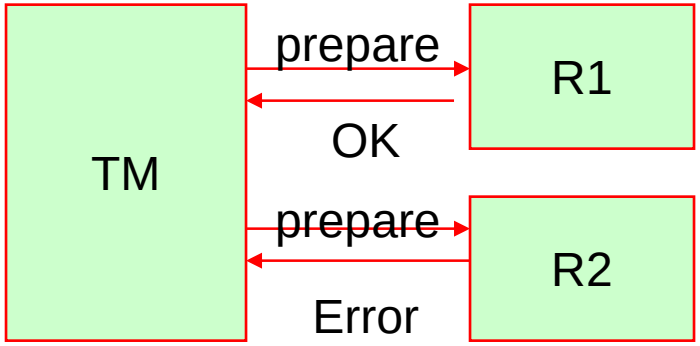
3 Cheap

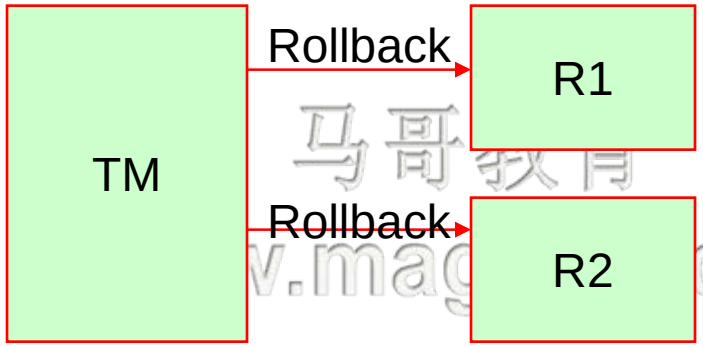
Looking at upfront costs, TCO and lifetimes









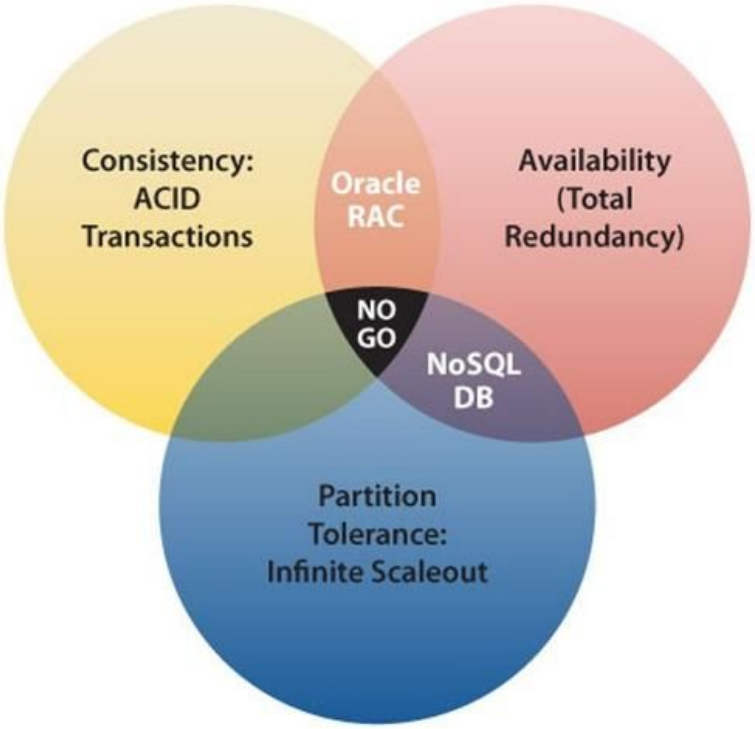


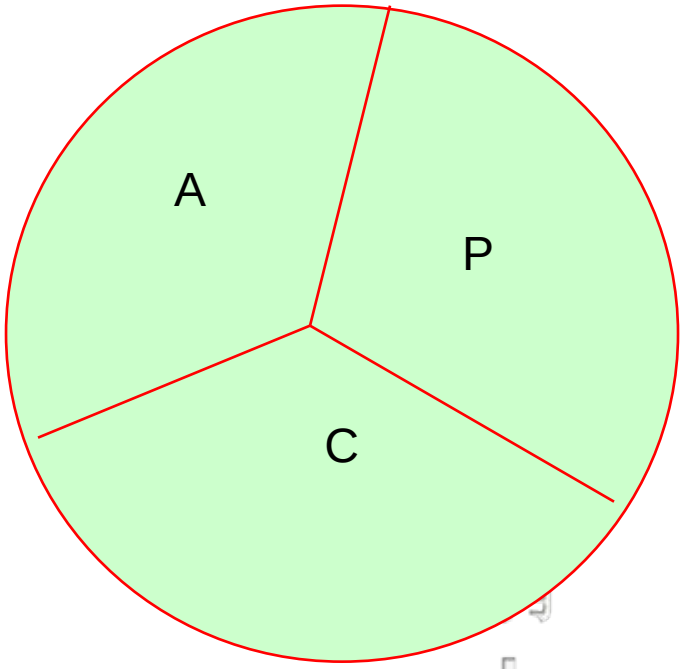
马哥教育

CAP理论

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

- C: Consistency
  - 任何一个读操作总是能够读取之前完成的写操作
- A: Availability
  - 可用性(指的是快速获取数据)
  - 每一次操作总是能够在确定的时间返回
- P: Tolerance of network Partition
  - 分区容错性(分布式)
  - 在出现网络分区的情况下，仍然能够满足一致性和可用性





鱼与熊掌？

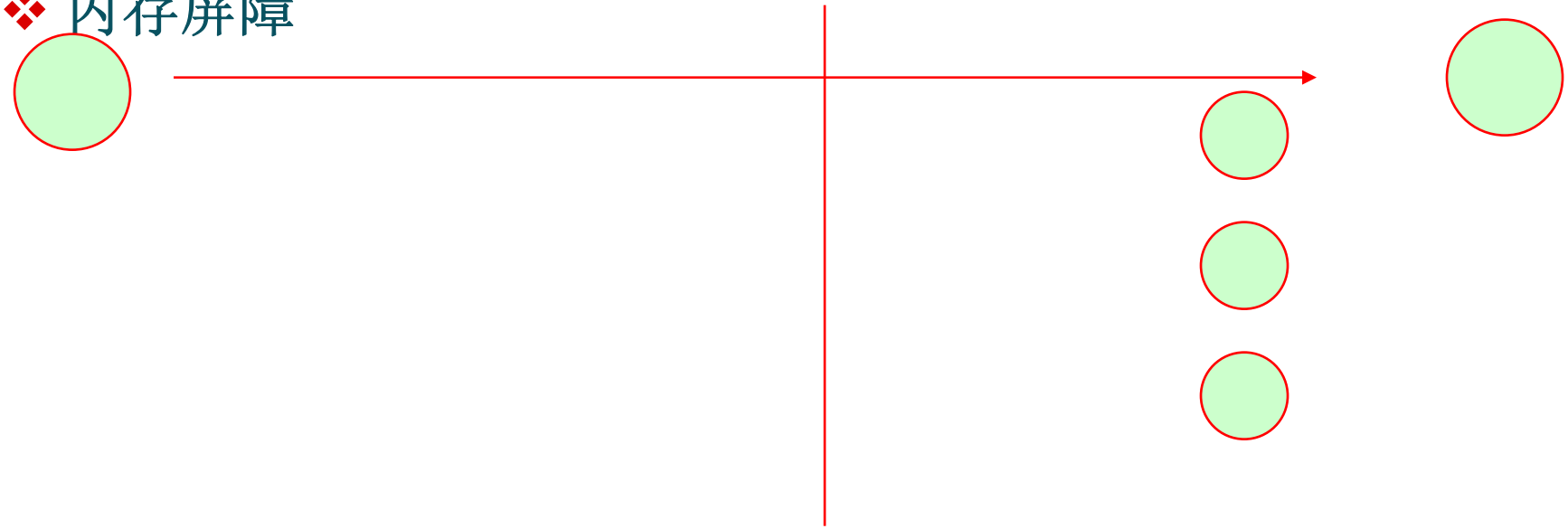
- 10年前，Eric Brewer教授指出了著名的CAP理论，后来Seth Gilbert和 Nancy lynch两人证明了CAP理论的正确性
- CAP理论：一个分布式系统不可能满足一致性，可用性和分区容错性这三个需求，最多只能同时满足两个
- 熊掌与鱼不可兼得
  - 关注一致性，就需要处理因为系统不可用而导致的写操作失败的情况
  - 关注可用性，应该知道系统的read操作可能不能精确的读取到write操作写入的最新值
    - CA：传统关系数据库
    - AP：key-value数据库
- 对大型网站，可用性与分区容忍性优先级要高于数据一致性，因此一般会尽量朝着A、P的方向设计，然后通过其它手段保证对于一致性的商务需求
- 不同数据对于一致性的要求是不同的，例如
  - 用户评论可以容忍相对较长时间的不一致，其较少会影响交易和用户体验
  - 而产品价格数据则是非常敏感的，通常不能容忍超过10秒的价格不一致
- CAP理论的证明：Brewer's CAP Theorem

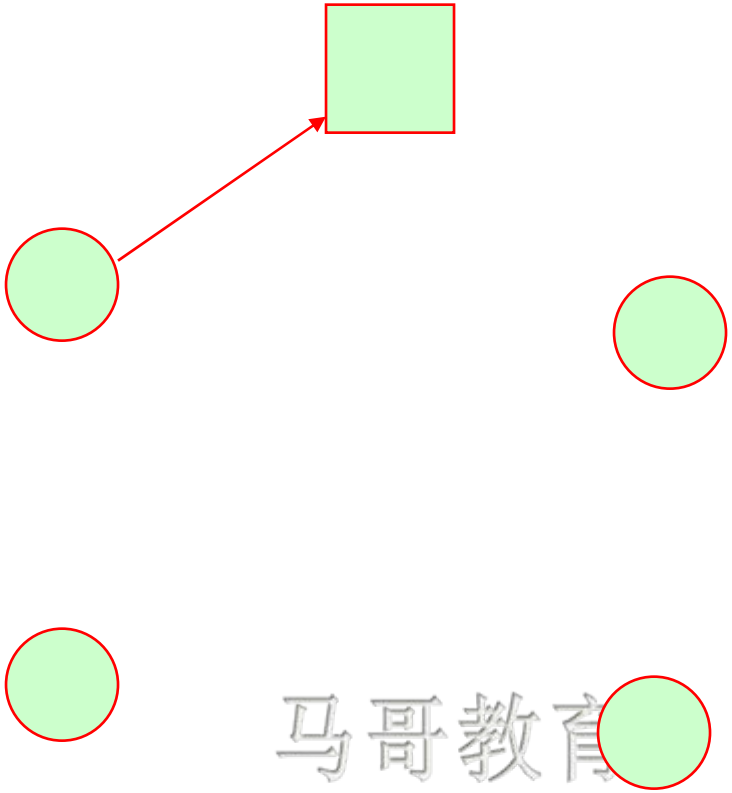
一致性

- 强一致性
  - ACID
  - 在单机环境中，强一致性可以由数据库的事务来保证
  - 在多机环境中，强一致性很难做到
  - 分布式事务：性能太差，在互联网的应用中不适合
- 弱一致性（包括最终一致性）
  - 通过提交处理的半同步、半异步或全异步，取得最终一致性效果
  - 最终一致性使得数据的提交具有延时性，而在一定范围的延时性范围内(比如一秒)，应用的可用性是正常的
- http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

同步调用

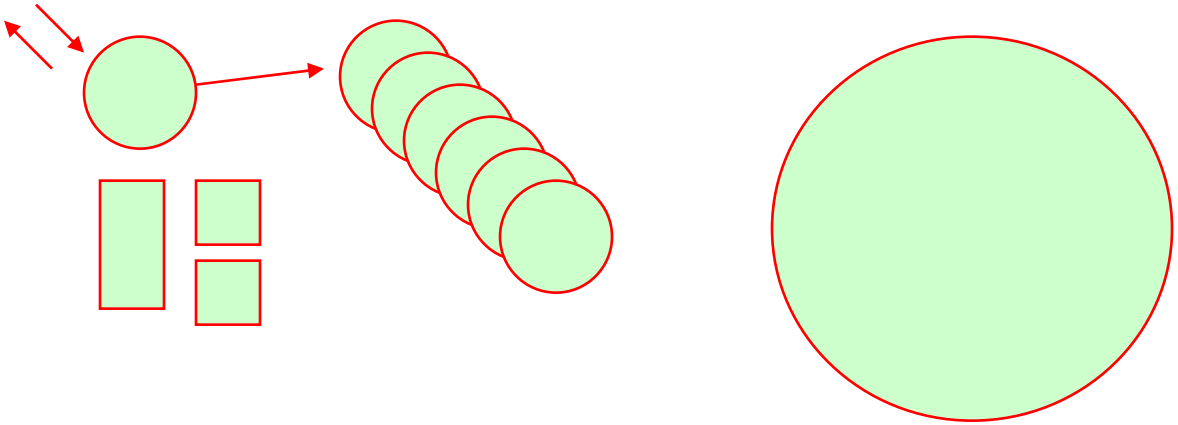
- 内存屏障





一致性应用场景

- 为了更好的描述客户端一致性，我们通过以下的场景来进行，这个场景包括三个组成部分：
  - 存储系统
    - 存储系统可以理解为一个黑盒子，它为我们提供了可用性和持久性的保证
  - ProcessA
    - ProcessA主要实现向存储系统write和read操作
  - ProcessB和ProcessC
    - ProcessB和C是独立于A，并且B和C也相互独立，它们同时也实现对存储系统的write和read操作



不同程度的一致性处理方式

- 强一致性
  - 强一致性（即时一致性）假如A先写入一个值到存储系统，存储系统保证后续的A,B,C的读操作都返回最新值
- 弱一致性
  - 假如A先写入了一个值到存储系统，存储系统不能保证后续A,B,C的读取操作能够读到最新值
  - 这种情况下有一个"不一致性窗口"的概念，它特指从A写入值，到后续操作A,B,C读取到最新值这一段时间。
- 最终一致性
  - 最终一致性是弱一致性的一种特例
  - 假如A首先write了一个值到存储系统，存储系统保证如果在A,B,C后续读取之前没有其他写操作更新同样的值的话，最终所有的读取操作都会读取到A写入的最新的值

- 这种情况下，如果没有失败发生的话，"不一致性窗口"的大小依赖以下的几个因素：
  - 交互延迟
  - 系统的负载
  - 复制架构中replica的个数（可以理解为master/slave模式中，slave的个数）
- 在最终一致性方面最出名的应该是DNS系统
  - 但更新一个域名的IP以后，根据配置策略以及缓存控制策略的不同，最终所有的客户都可以看到最新的域名和IP的映射

其他一致性变体的处理方式

- Causal consistency(因果一致性)
  - 如果Process A通知Process B它已经更新了数据，那么Process B的后续读取操作则读取A写入的最新值，而与A没有因果关系的C则可以最终一致性
- Read-your-writes consistency(过程一致性)
  - 如果Process A写入了最新的值，那么Process A的后续操作都会读取到最新值。但是其它用户可能要过一会才可以看到
- Session consistency(会话一致性)
  - 此种一致性要求客户端和存储系统交互的整个会话阶段保证Read-your-writes consistency
  - Hibernate的session提供的一致性保证就属于此种一致性
- Monotonic read consistency(简单读一致性)
  - 此种一致性要求如果Process A已经读取了对象的某个值，那么后续操作将不会读取到更早的值
- Monotonic write consistency(简单写一致性)
  - 此种一致性保证系统会序列化执行一个Process中的所有写操作

服务器一致性

- 概念
  - N：节点的个数
  - W：更新的时候需要确认已经被更新的节点个数
  - R：读数据的时候读取数据的节点个数

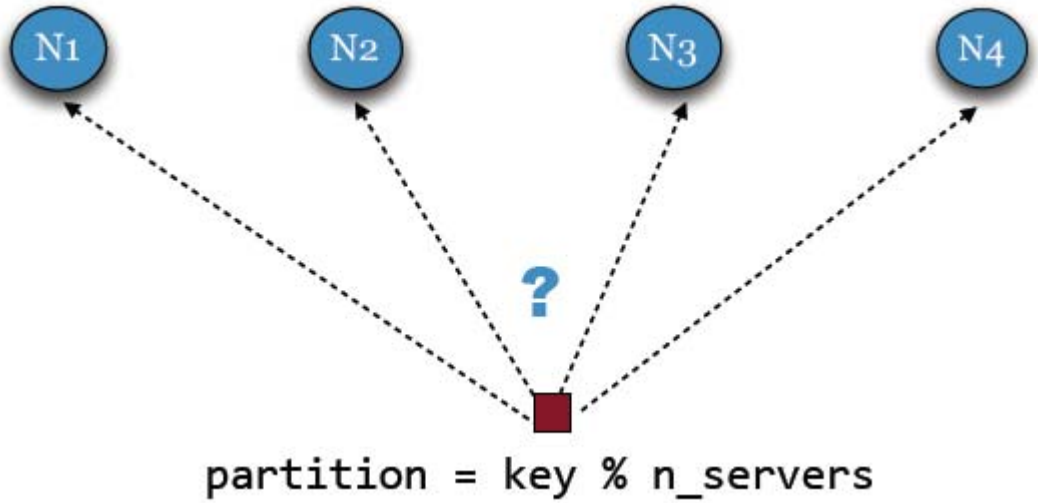
W + R > N ⟶ 强一致性(通常 N=3,W=R=2)

W=N,R=1 ⟶ 最佳读

W=1,R=N ⟶ 最佳写

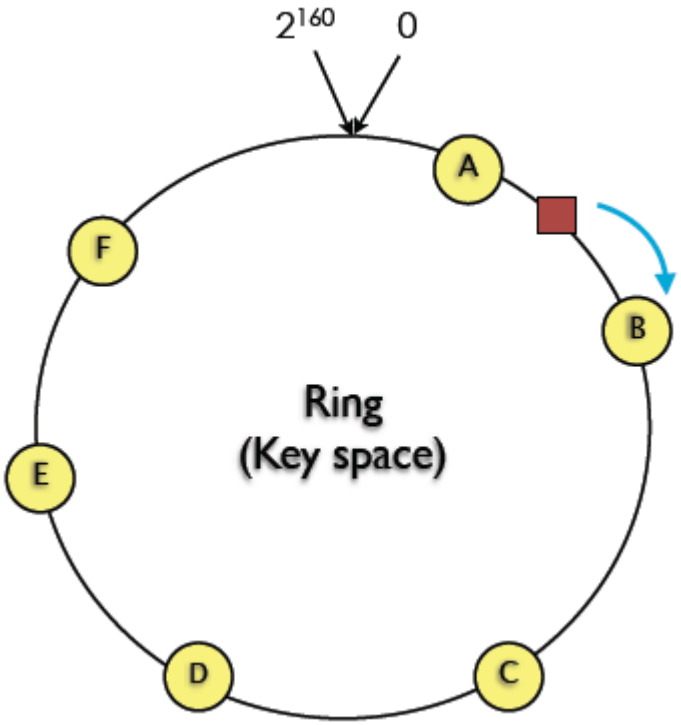
W + R ≤ N ⟶ 弱一致性

基于模的哈希



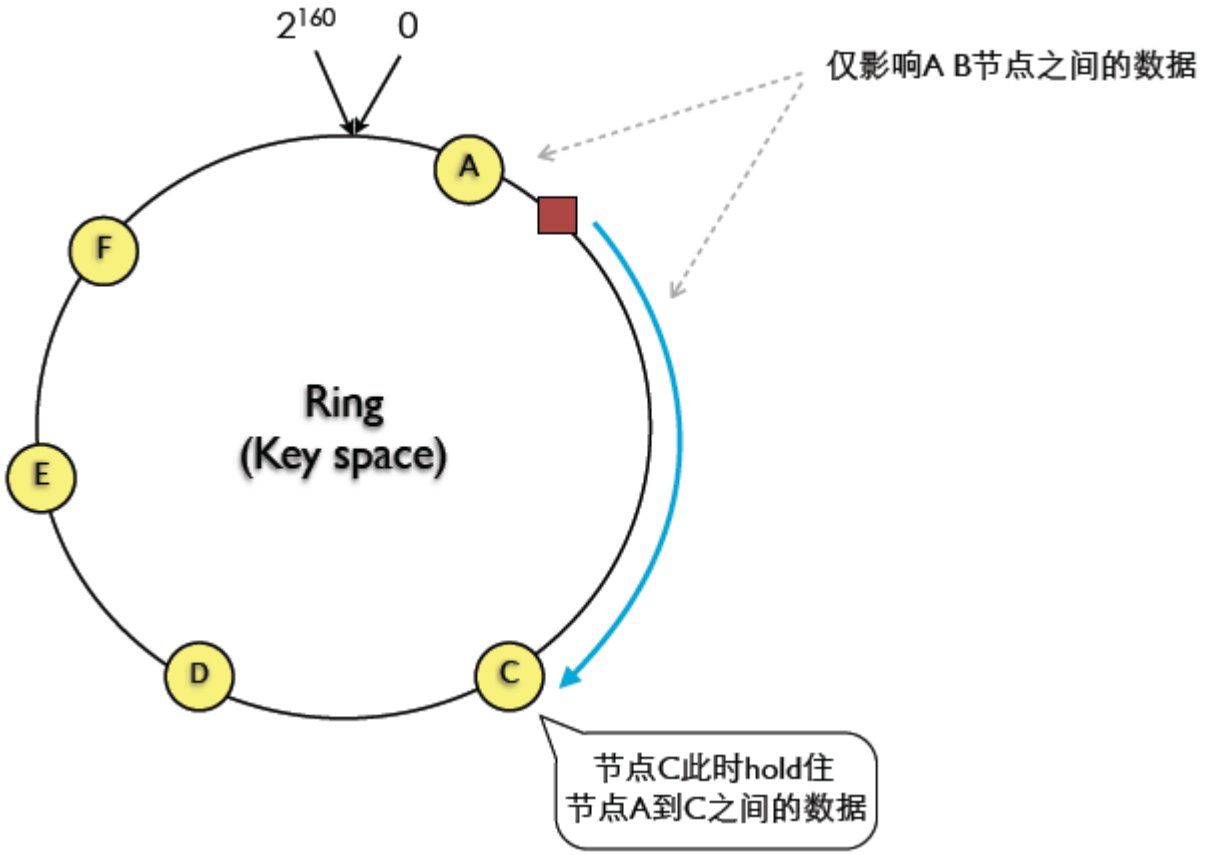
当*n_servers*改变时，重新计算所有的资源
(i.e 节点改变时，资源需要全部重新分布)

一致性哈希

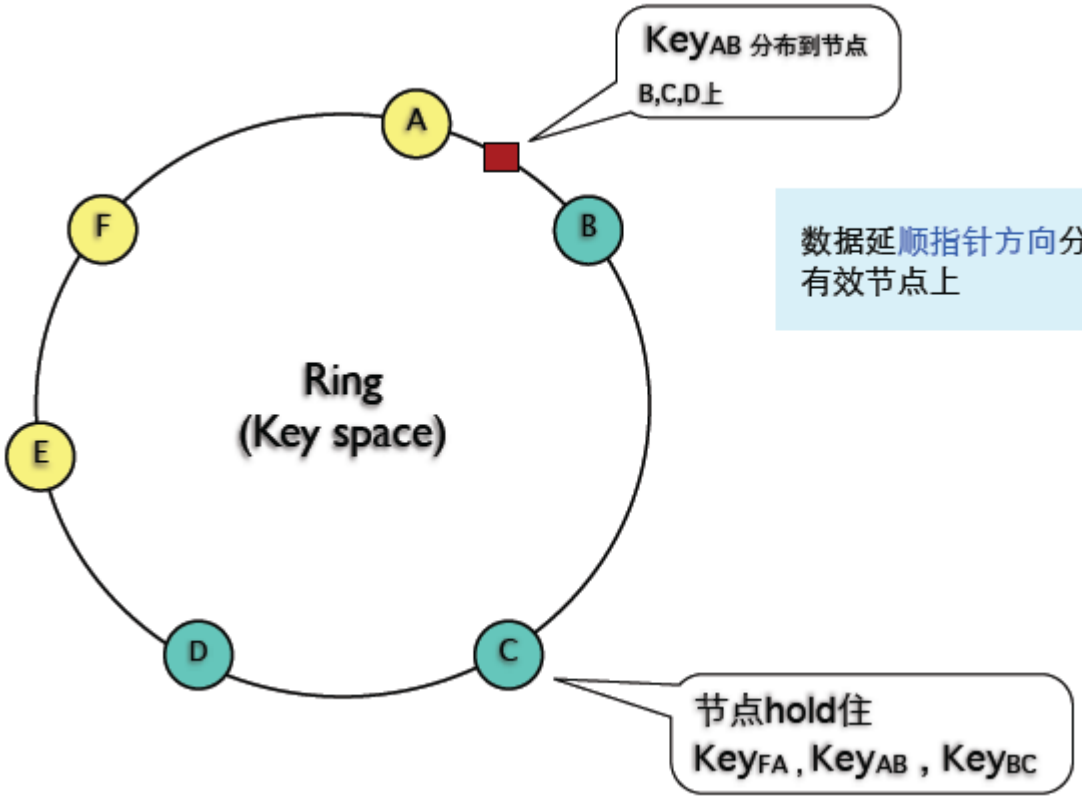


- 创建一个针对数据以及节点的hash函数
  - idx = hash(key)
- 协调器：延顺指针方向的第一个有效节点

一致性哈希：节点改变

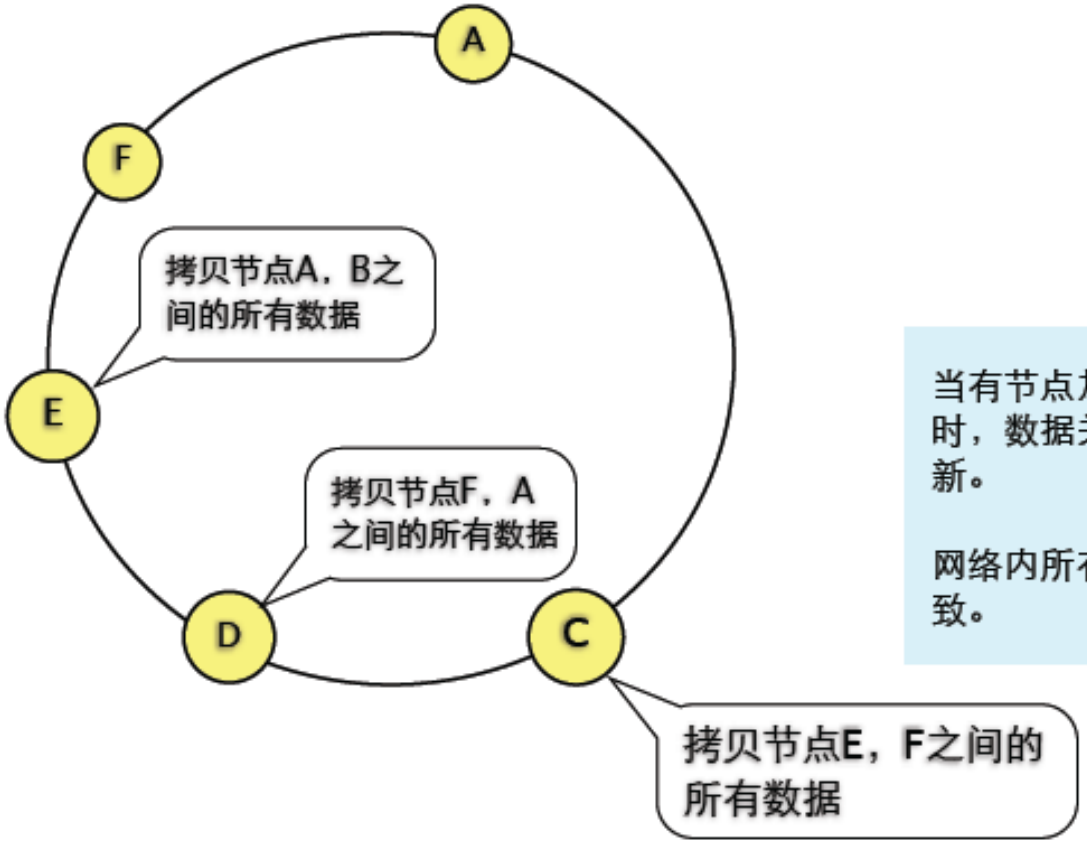

2^{160}
0
仅影响A B节点之间的数据
A
F
Ring
(Key space)
E
D
C
节点C此时hold住
节点A到C之间的数据

一致性哈希：复制



数据延顺指针方向分布到N -1
有效节点上

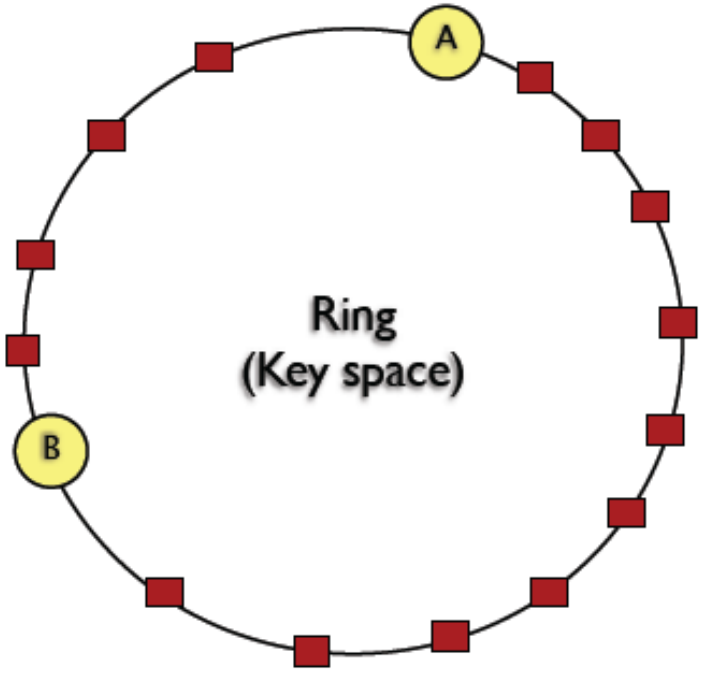
一致性哈希：节点改变



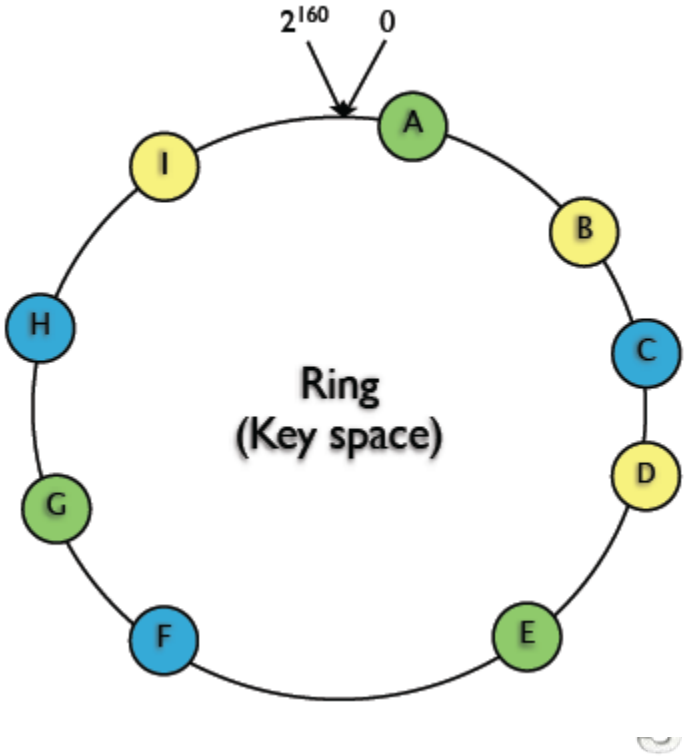
当有节点加入或者离开网络时，数据关系以及副本需要更新。

网络内所有副本数据保持一致。

一致性哈希：数据倾斜



一致性哈希：虚拟节点

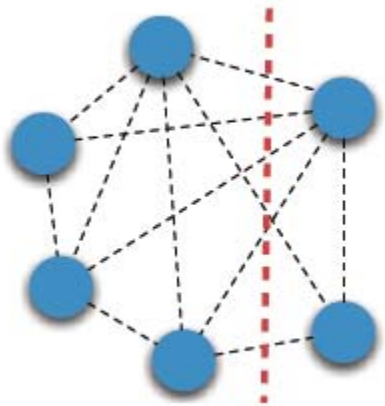


每个物理节点生成随机tokens，然后根据token值分区

- Node 1: tokens A, E, G
- Node 2: tokens C, F, H
- Node 3: tokens B, D, I

分区容错性-可用性

- "The network will be allowed to lose arbitrarily many messages sent from one node to another" [...]
- "For a distributed system to be continuously availability,every request received by non-failing node in the system must result in a response"



HIGH LATENCY

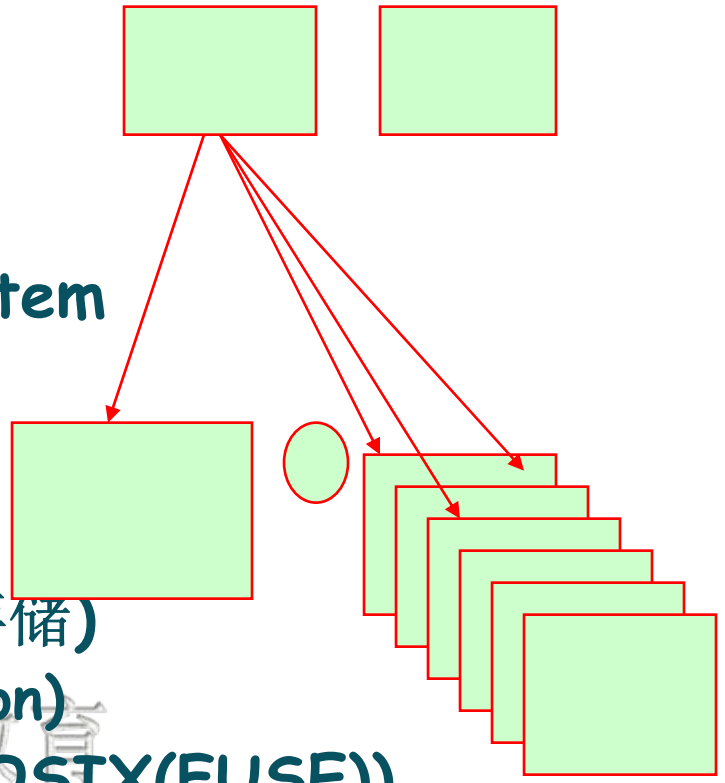
≈

NETWORK PARTITION

http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html

分布式文件系统

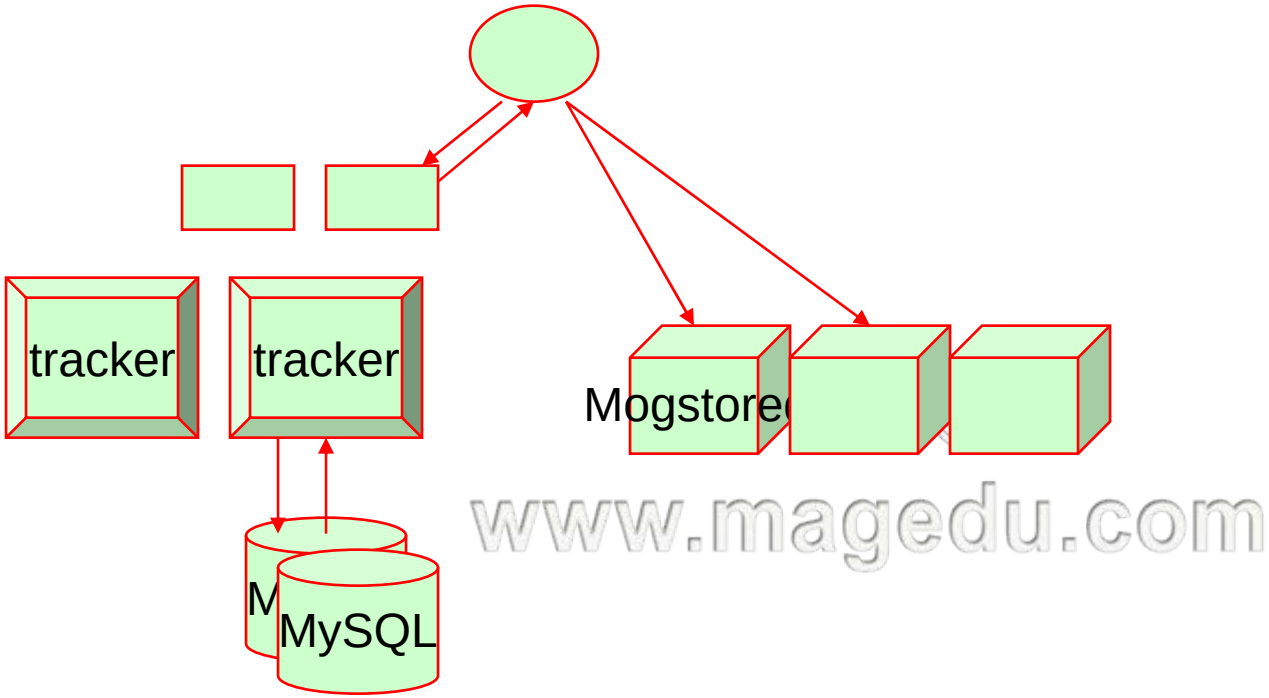
- Google File System
  - MapReduce：编程模型
- TFS：Taobao Filesystem
- Hadoop Distributed Filesystem
- LiveJournal
  - memcached
  - perlbal
- Mogile Filesystem (分布式存储)
  - API (php, java, perl, python)
- Moose Filesystem (MFS, POSIX(FUSE))
- Lustre, HPC
- Ceph
- GlusterFS

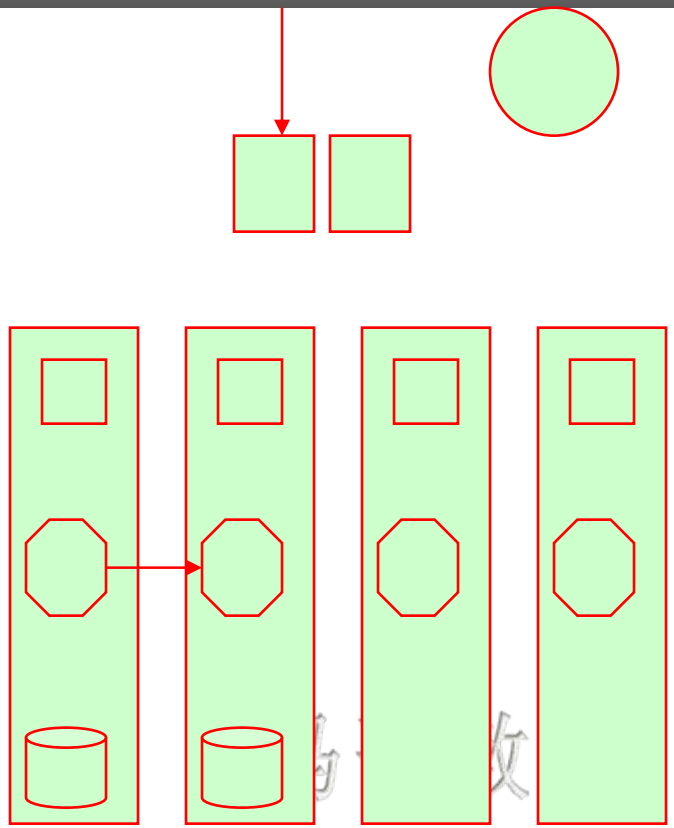


马哥教育

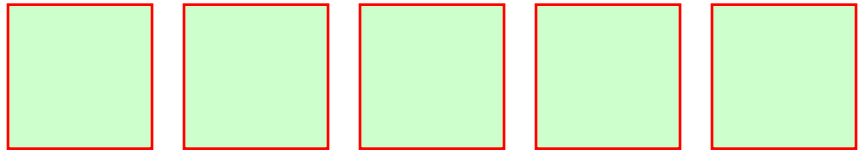
MogileFS

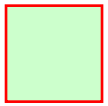
主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

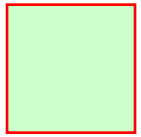




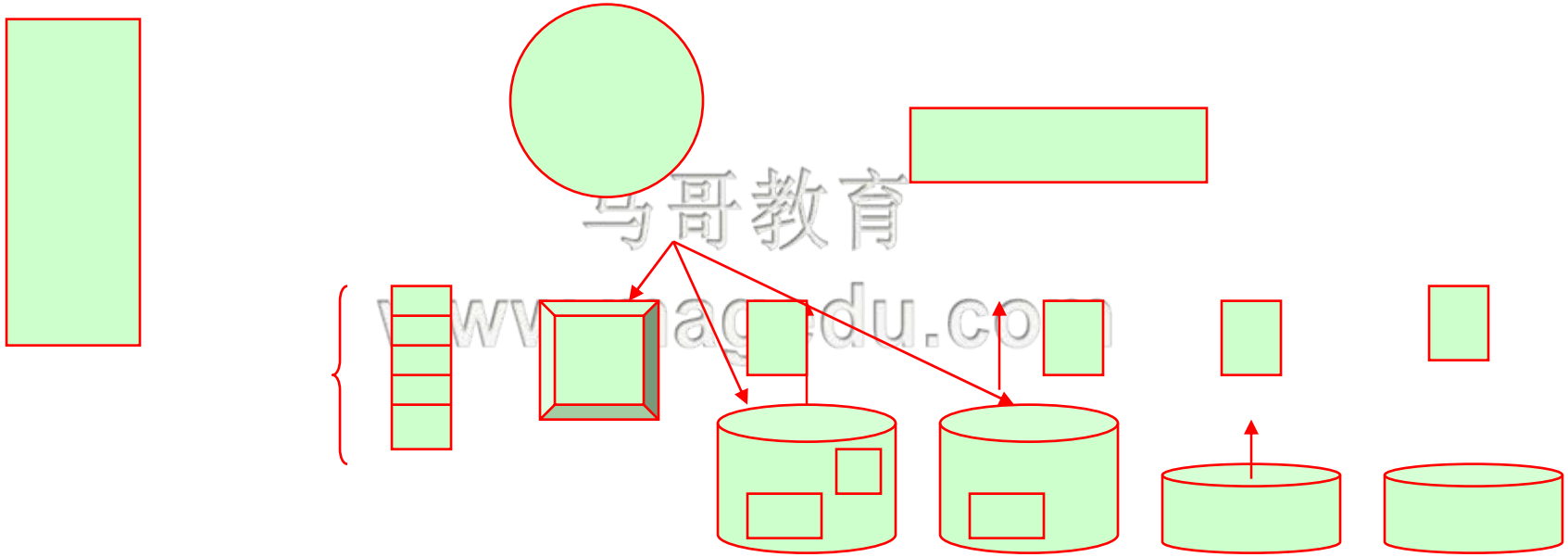
w,1
R,1



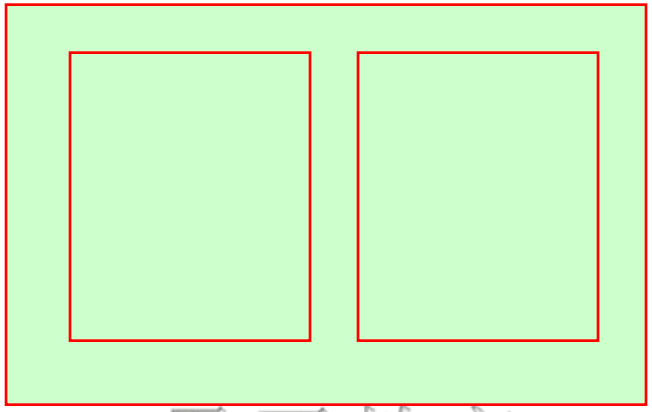




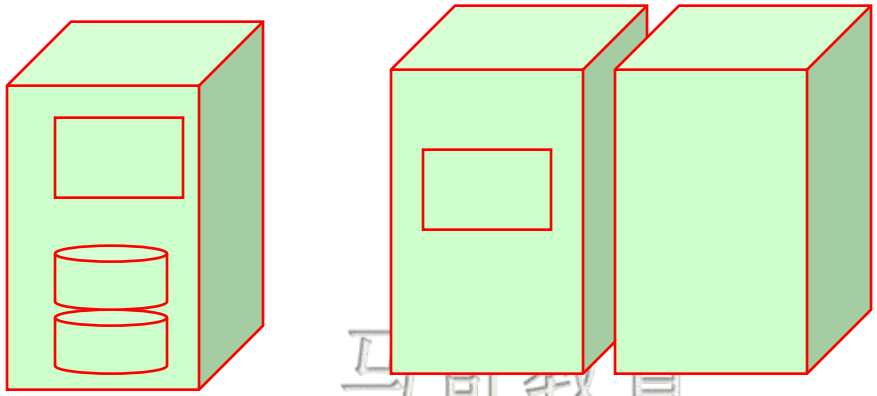
2T: 64M



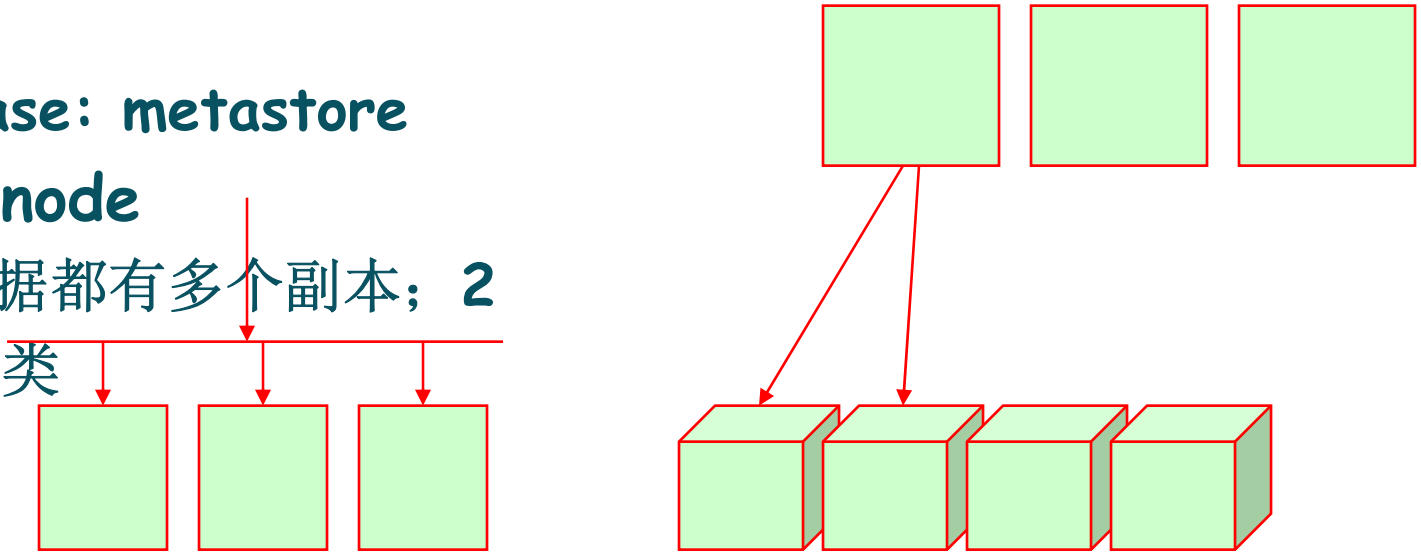
- www.a.com/a/b.jpg
- www.b.org/a/b.jpg

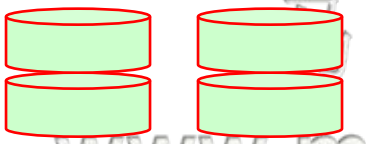


- MogileFS
  - mogilefsd：tracker进程
  - mogstored：storage进程
  - perlbal进程



- tracker
  - database：metastore
- storage node
  - 每个数据都有多个副本：2
  - class：类





mogifsd

mogstored

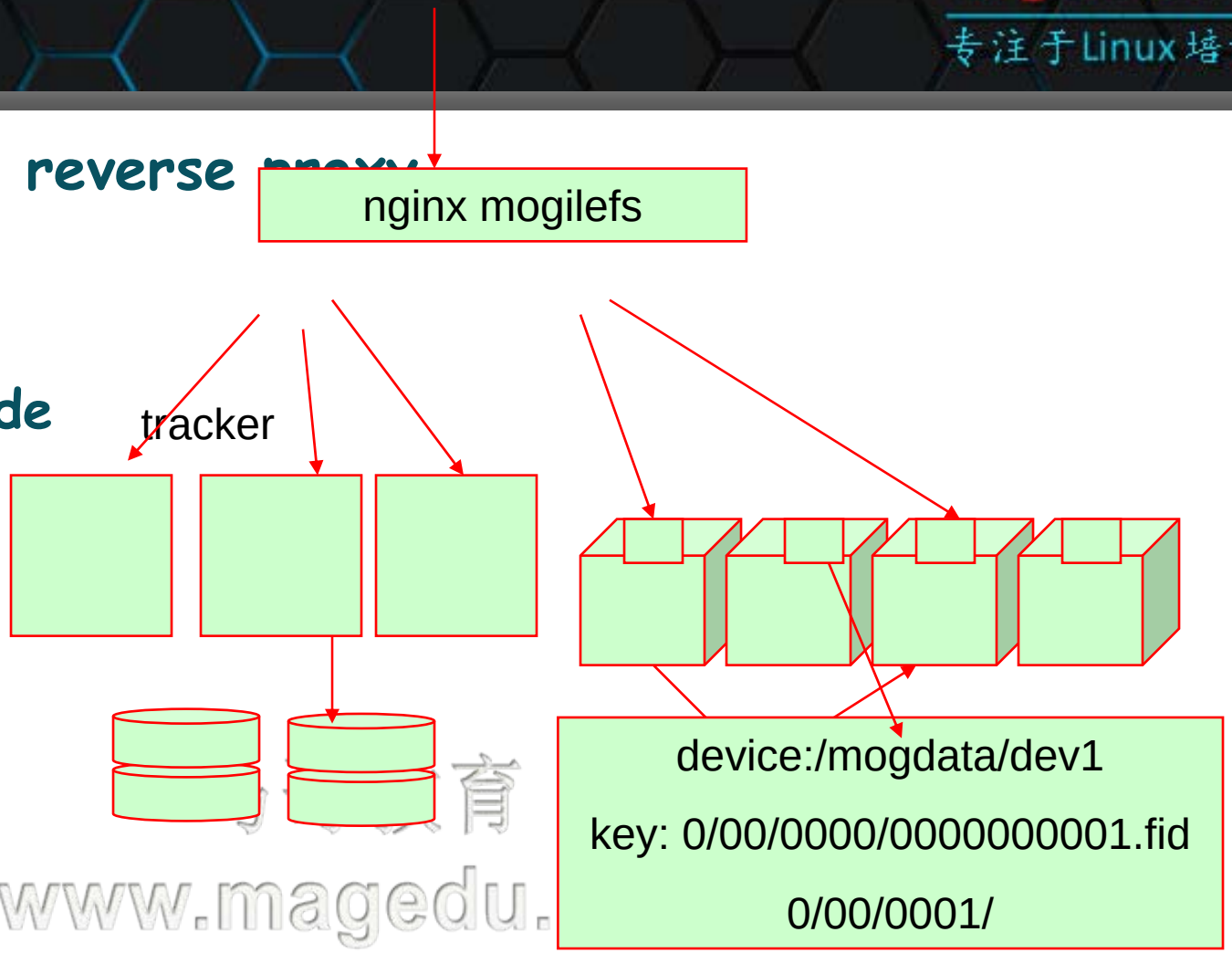
7500

7501

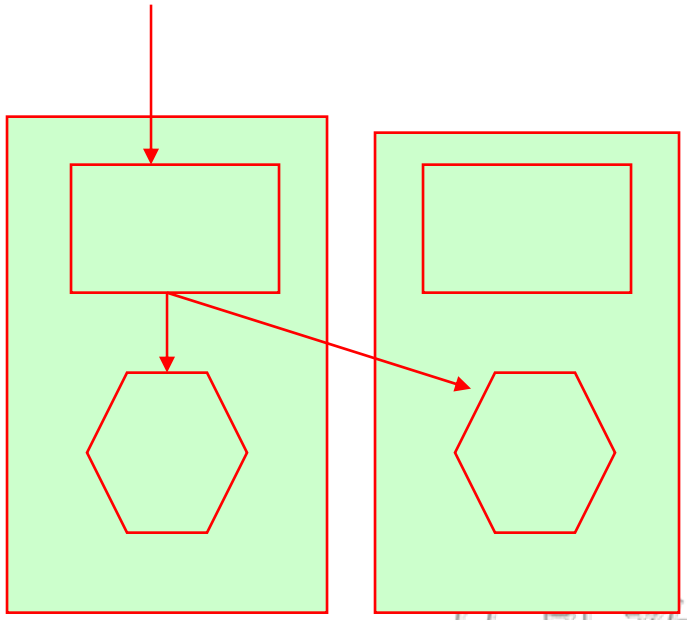
perlbal:

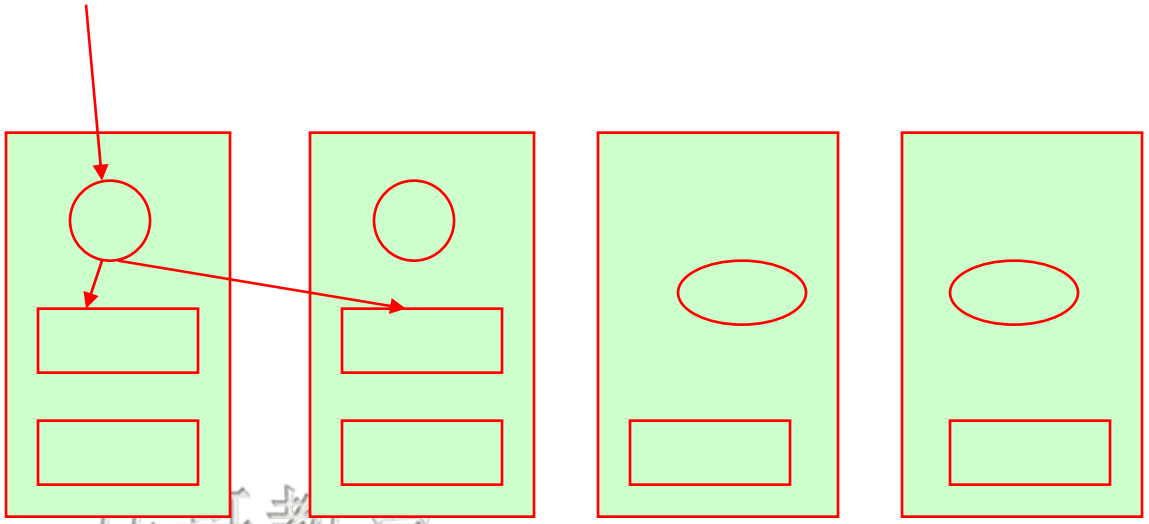
http://

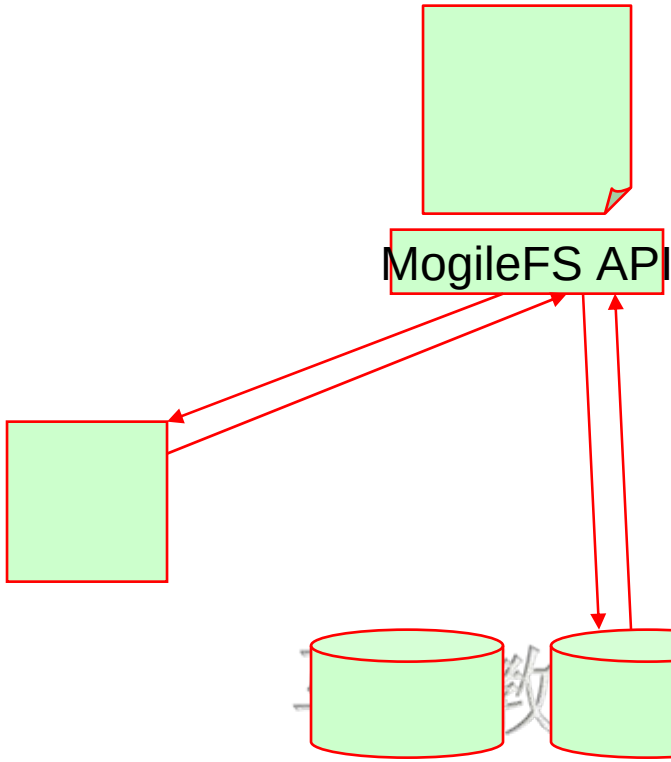
- pe[redacted]l: web reverse proxy
- MogileFS
  - tracker
  - storage node
  - database

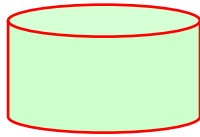


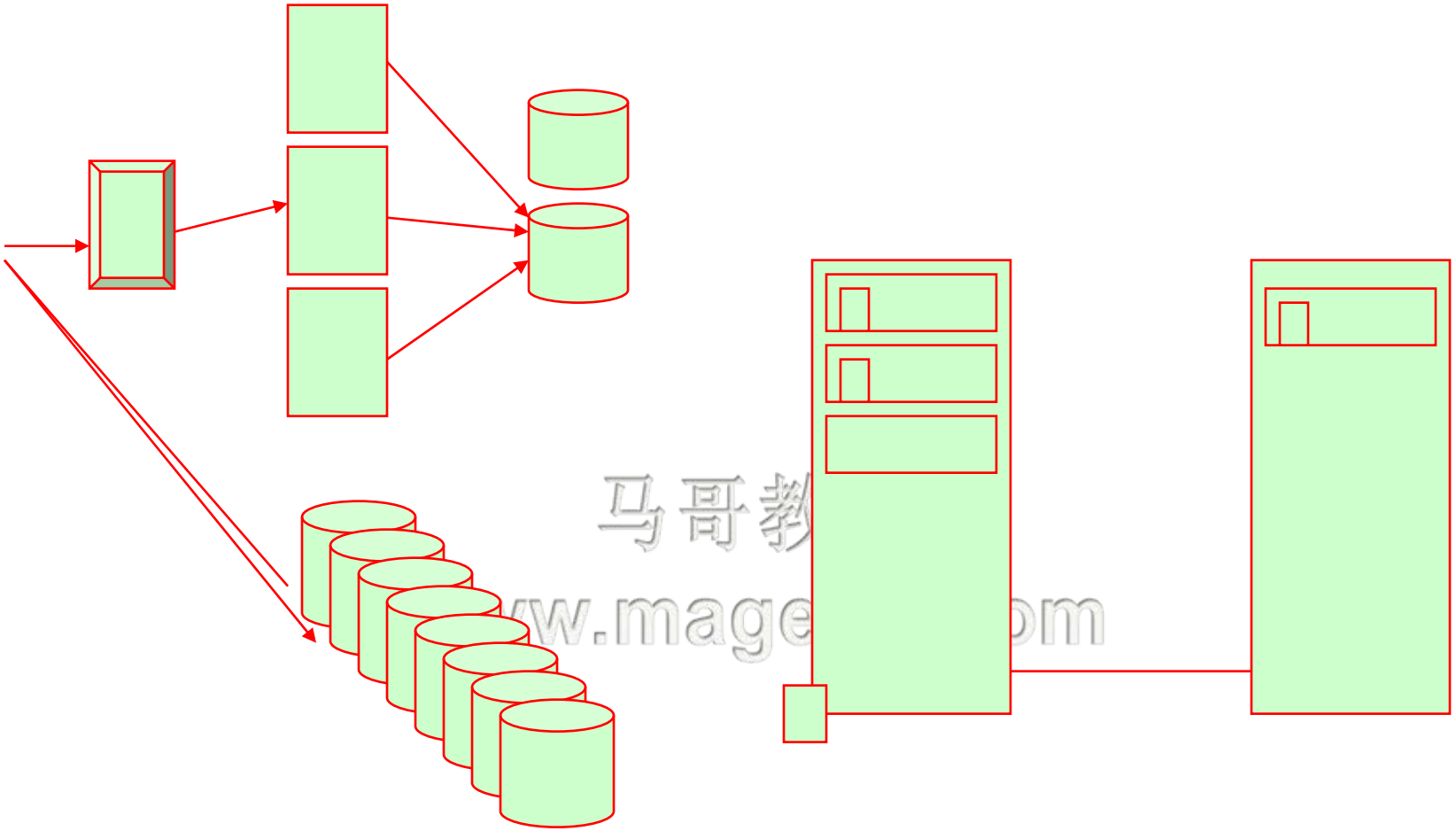
/files/fstab → 0/00/0000/0000000000006.fid



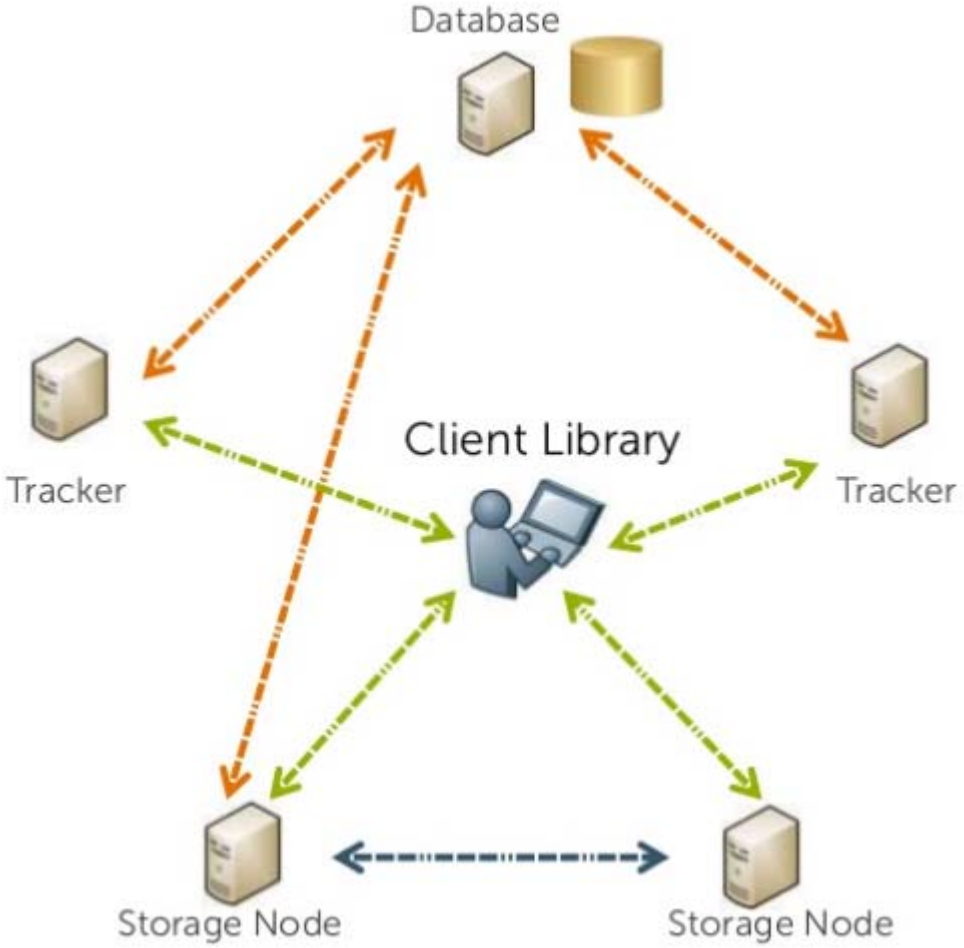


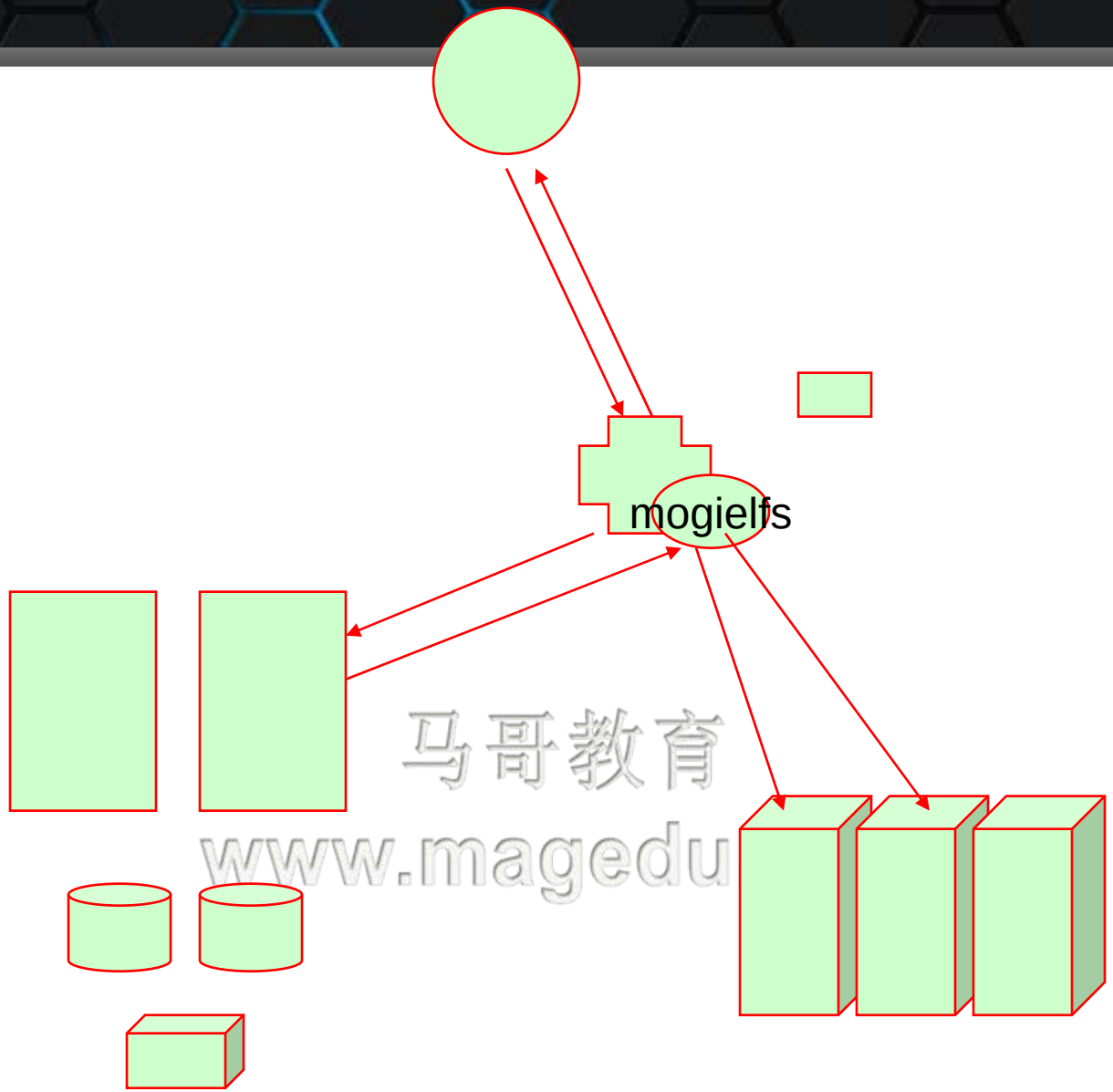






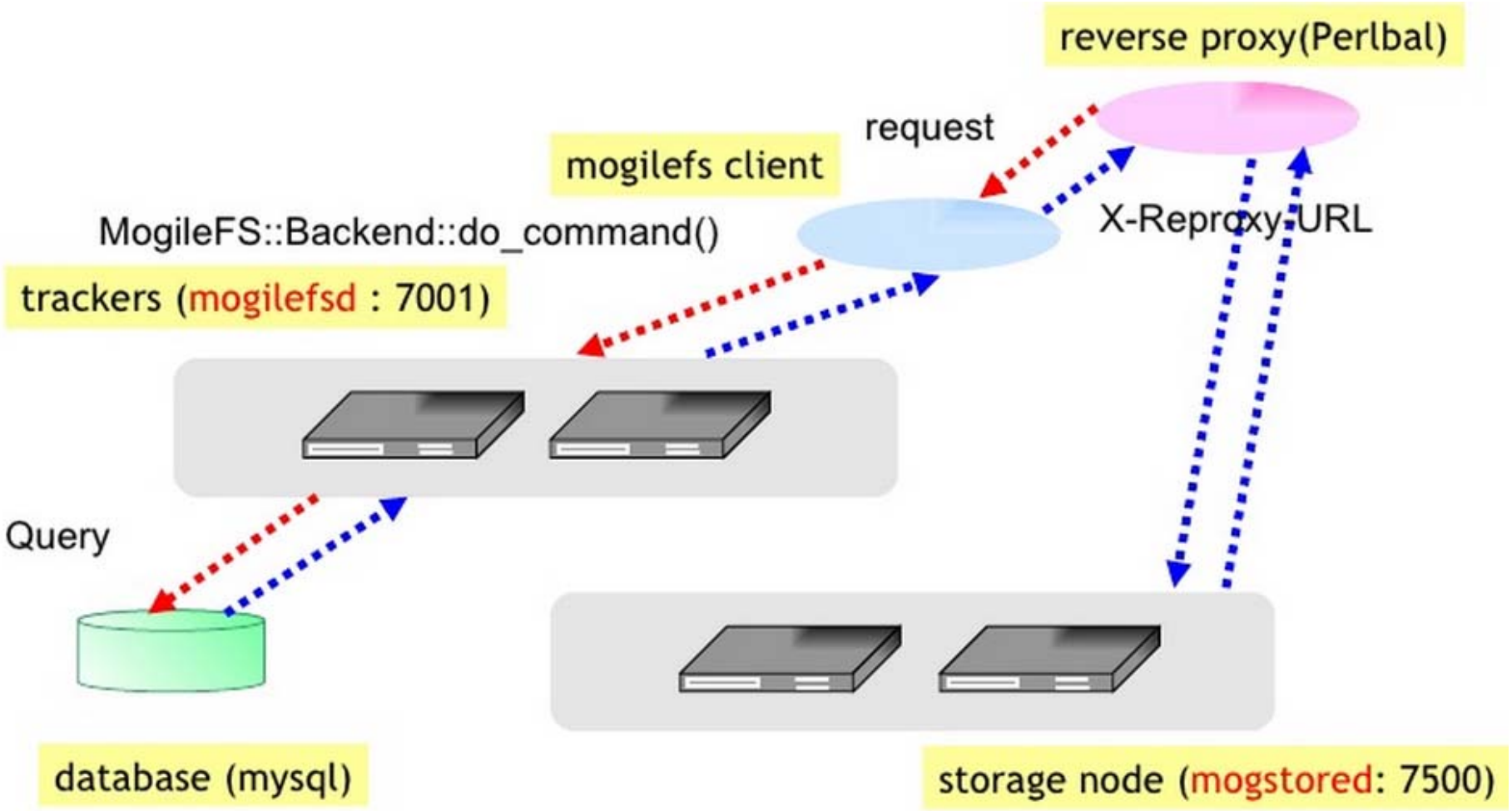
MogileFS Architecture Overview





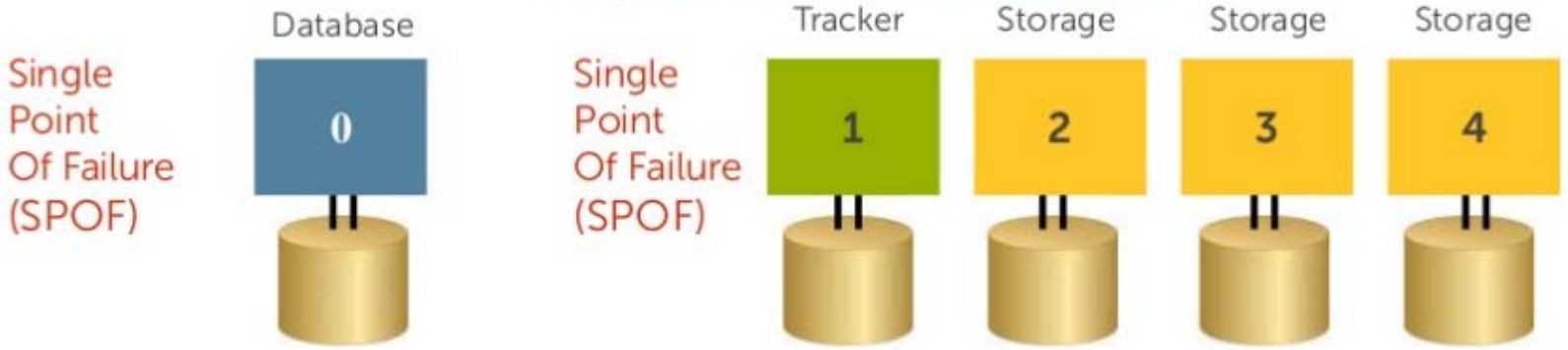
Features

- 应用层：无需特殊核心组件
- 无单点失败
- 自动文件复制
- 传输中立，无特殊协议
- 简单命名空间
- Shared-Nothing
- non-RAID
- 不能追加写、随机写
- Tracker Client传输(mogilefsd)，管理数据复制、删除、查询、修复以及监控
- 数据通过HTTP/WebDAV服务上传到 Storage node(mogstored)
- MySQL 存储MogileFS 元数据(命名空间、位置)

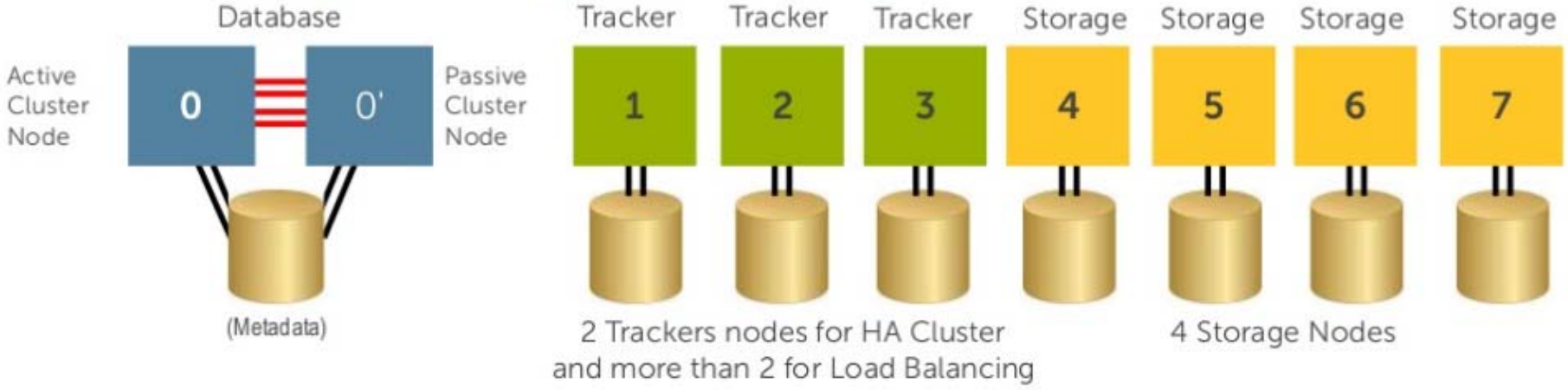

reverse proxy(Perlbal)
request
mogilefs client
MogileFS::Backend::do_command()
X-Reproxy-URL
trackers (mogilefsd : 7001)
Query
database (mysql)
storage node (mogstored: 7500)

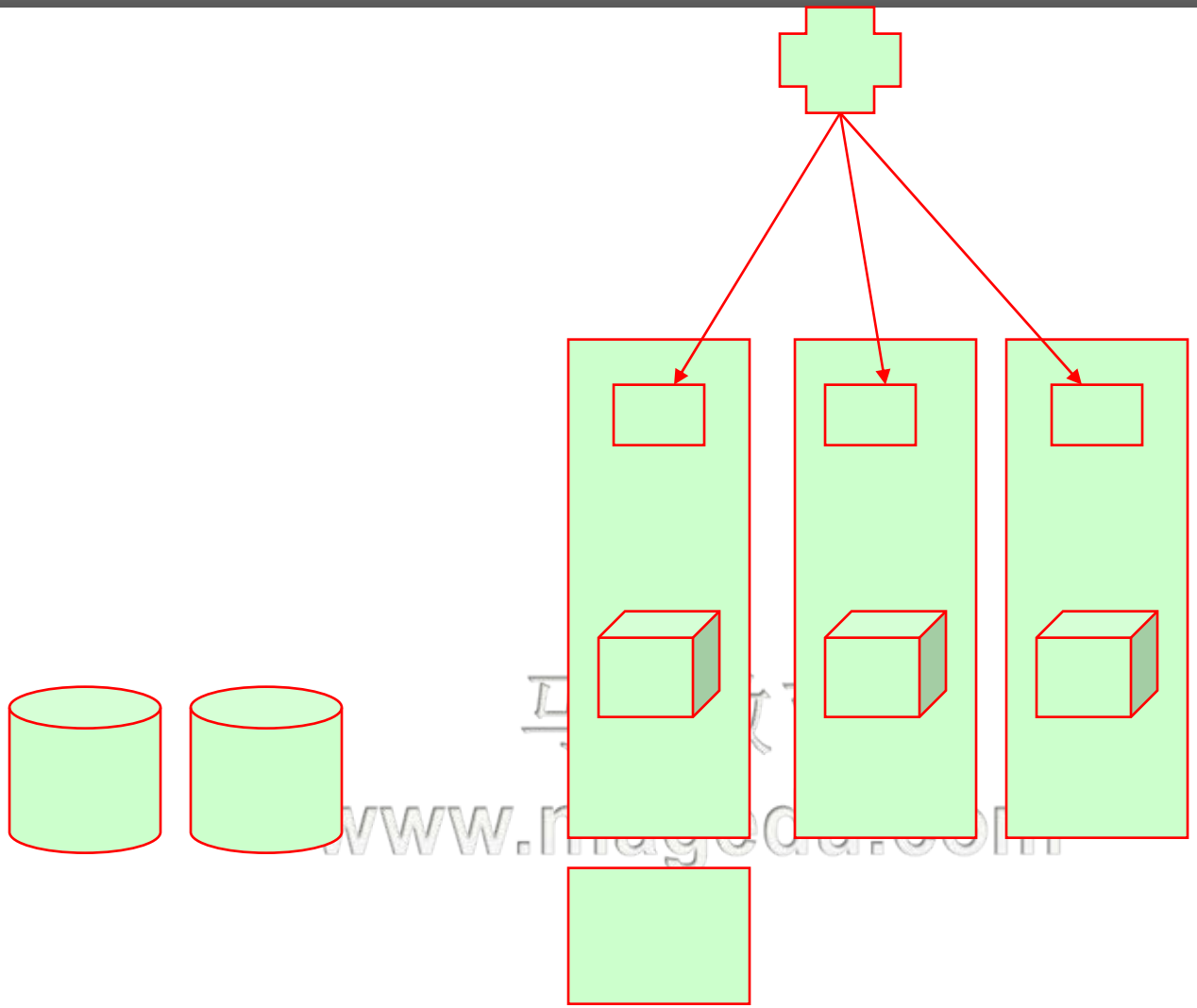
MogileFS High Availability

MogileFS Architecture

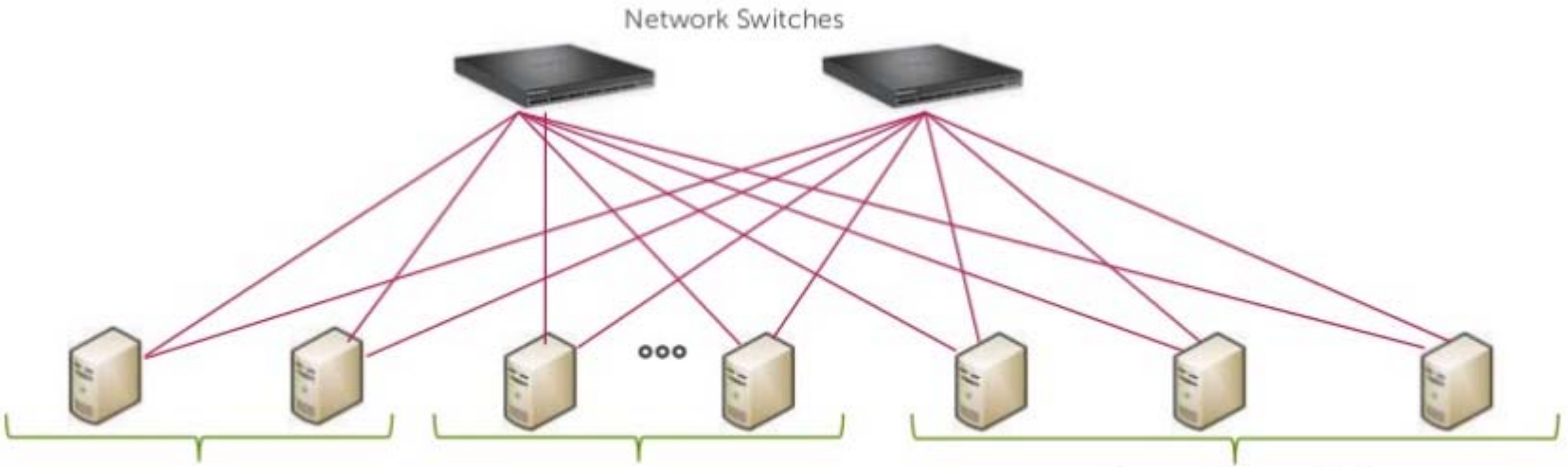


MogileFS High Availability Architecture

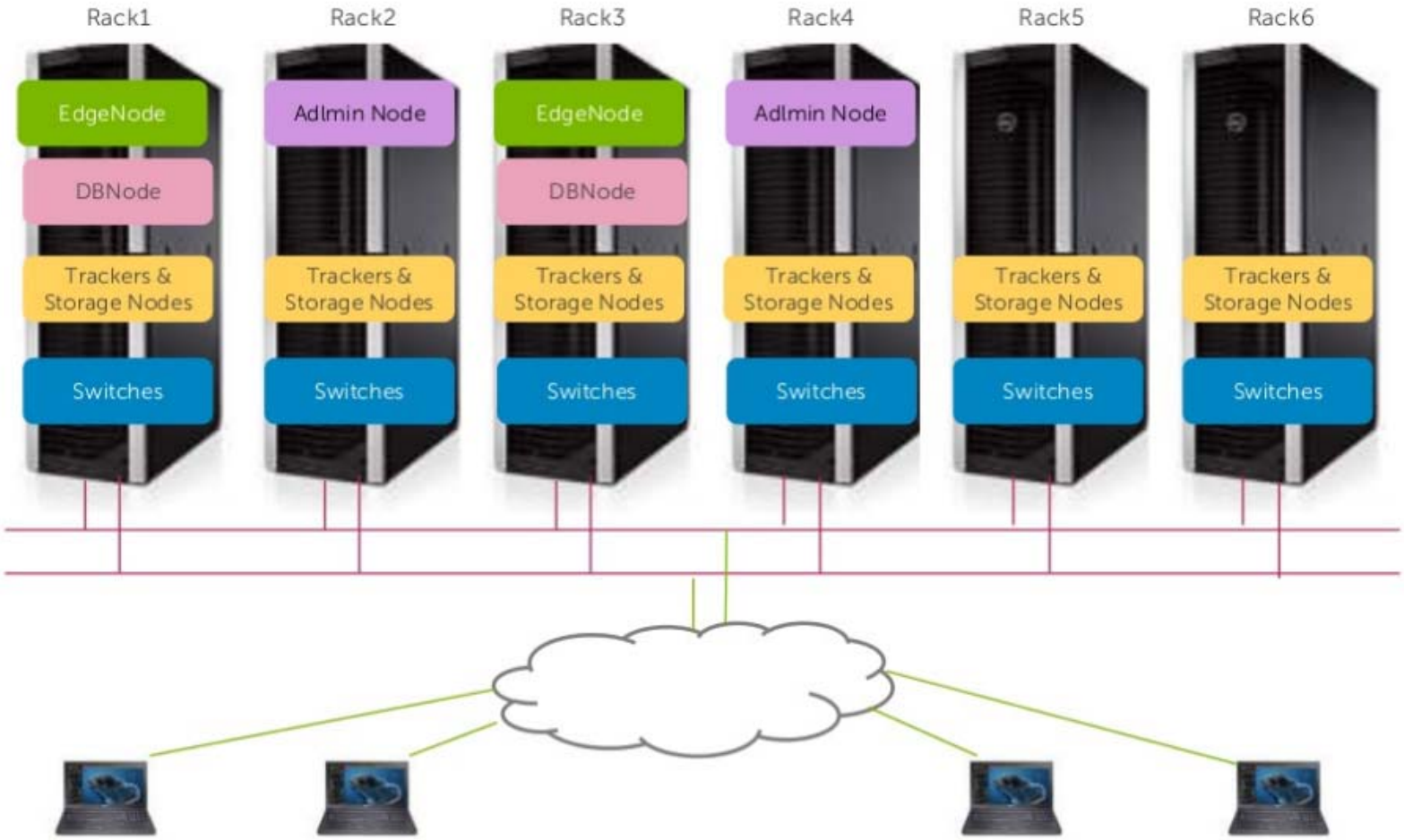




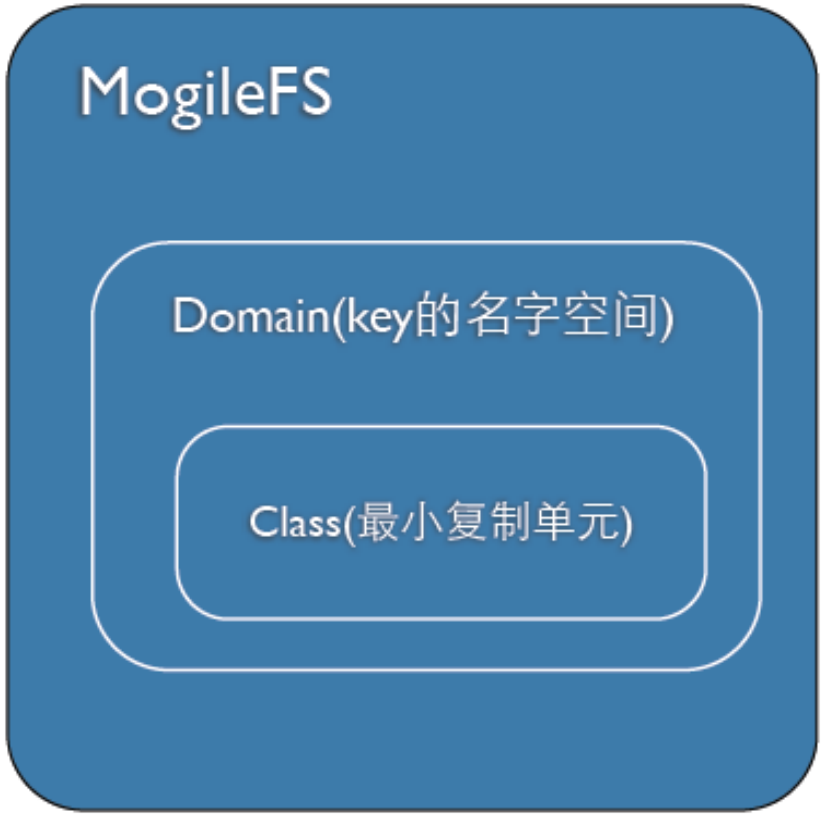
MogileFS Architecture



MogileFS Architecture Overview



- Domain
  - 一个MogileFS可以有多个Domain
  - 用来存放不同文件（大小、类型）
  - 同一个Domain内，key必须唯一
  - 不同Domain内，key可以相同
- Class
  - 文件属性管理
  - 定义文件存储在不同设备上的份数
- Domain + Fid 定位文件

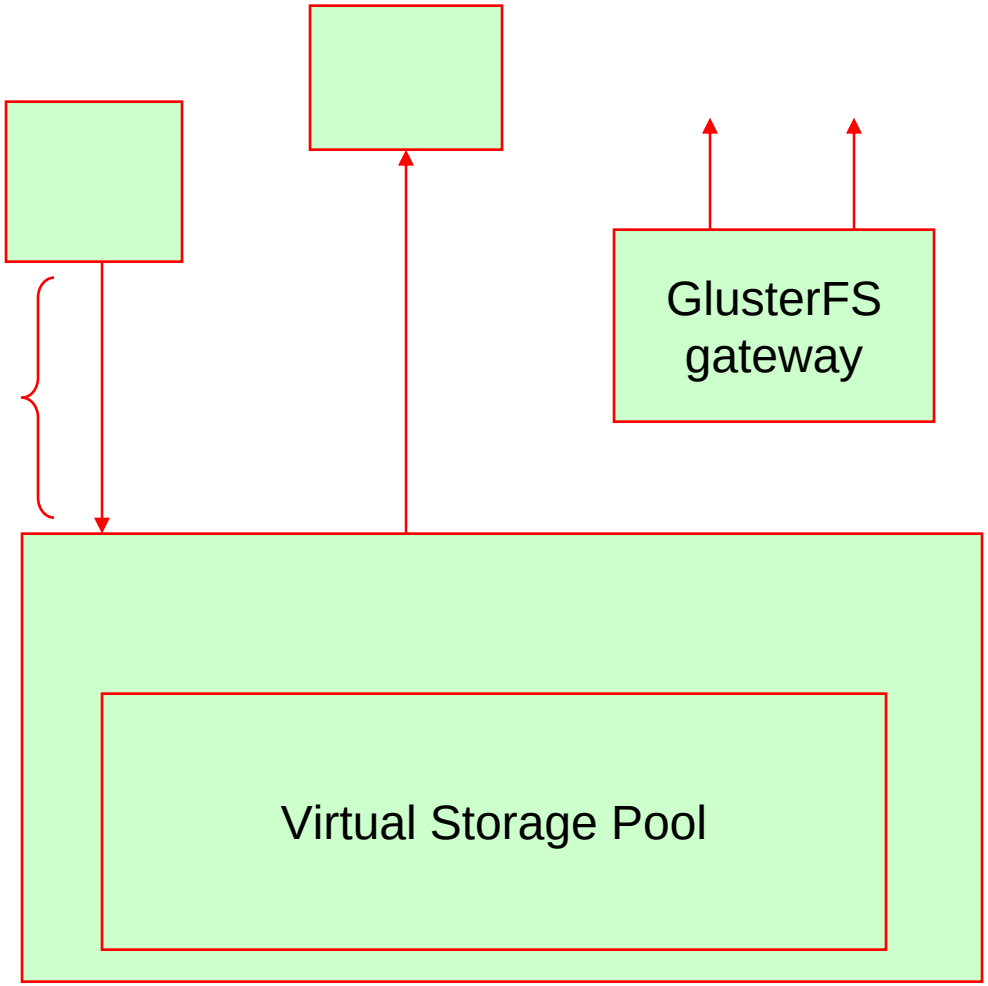


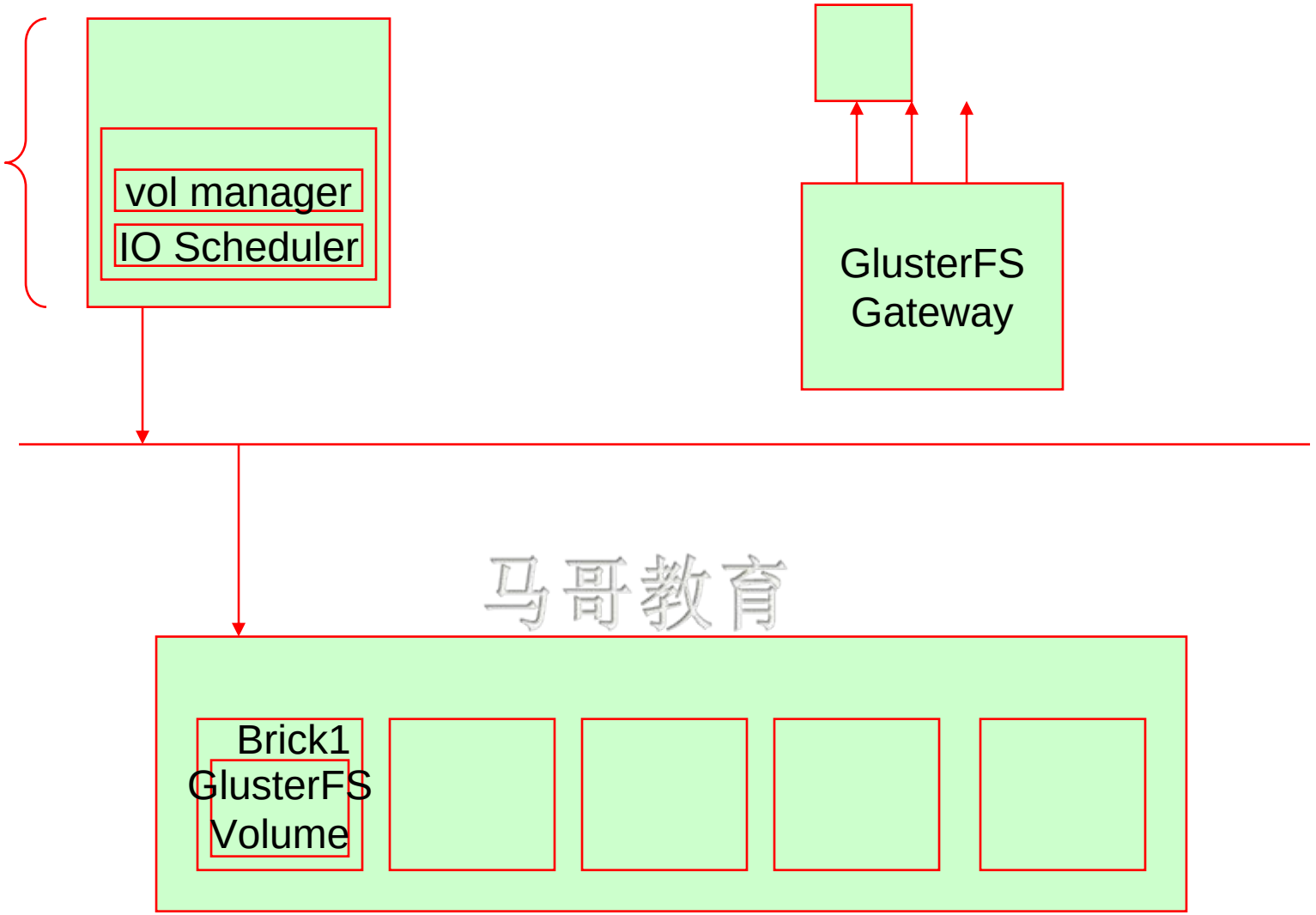
马哥教育

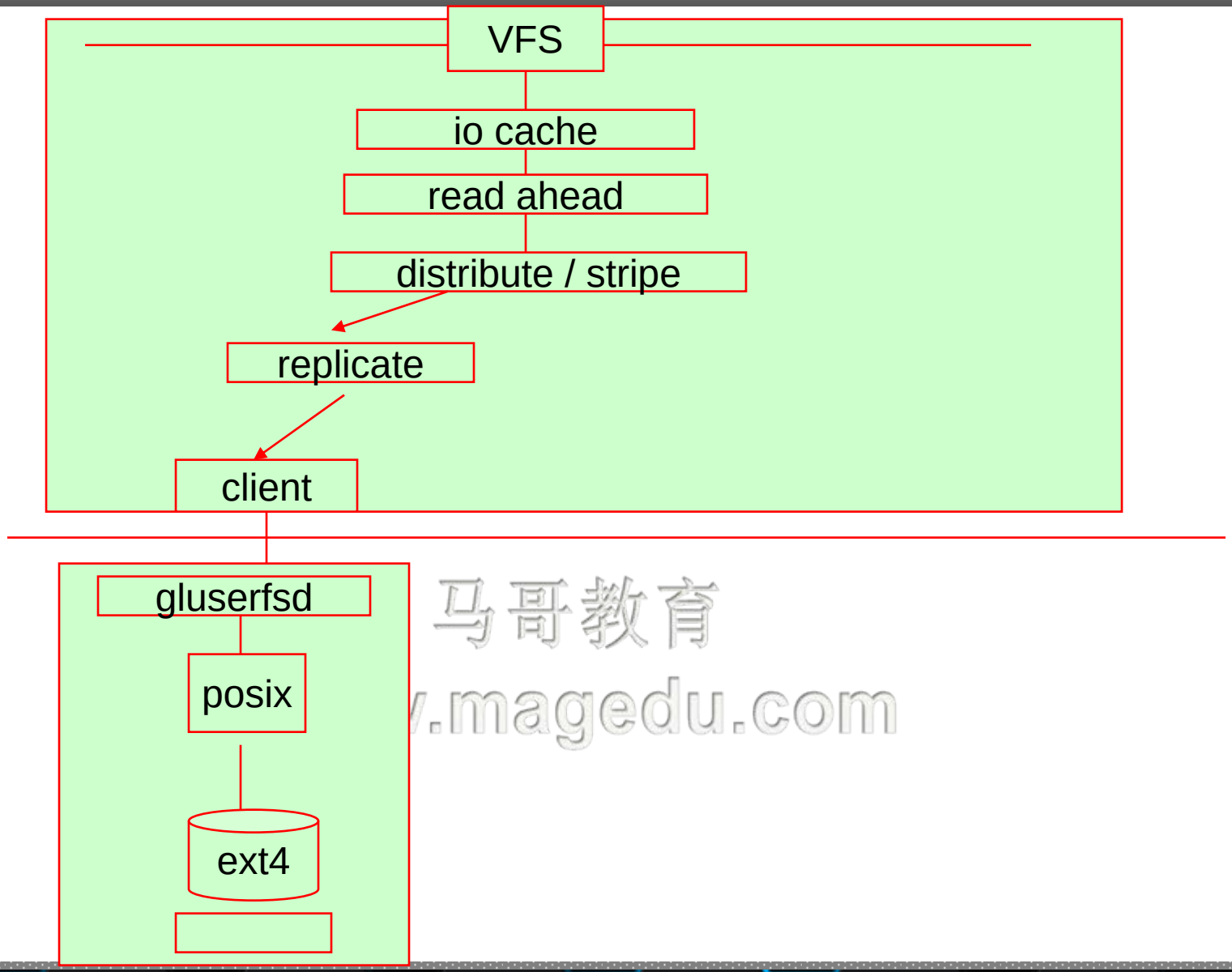
GlusterFS

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

- glusterfs




vol manager
IO Scheduler
GlusterFS
Gateway
马哥教育
Brick1
GlusterFS
Volume



关于马哥教育

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：2813150558，1661815153，113228115
- QQ群：203585050，279599283

马哥教育

Thank You!

项目设计实践

- 5个节点，实现：
  - 负载均衡器
  - 缓存服务器
  - tomcat两服务器；
  - mysql主从；
  - nginx反代用户对图片请求至mogilefs集群；
  - mogilefs集群至少两节点：双mogilefsd和双mogstored；

- 请画出设计图，并说明其存在问题的位置及后续的解决方案；