研究生课程

云计算技术原理

Cloud Computing: Principles and Technologies

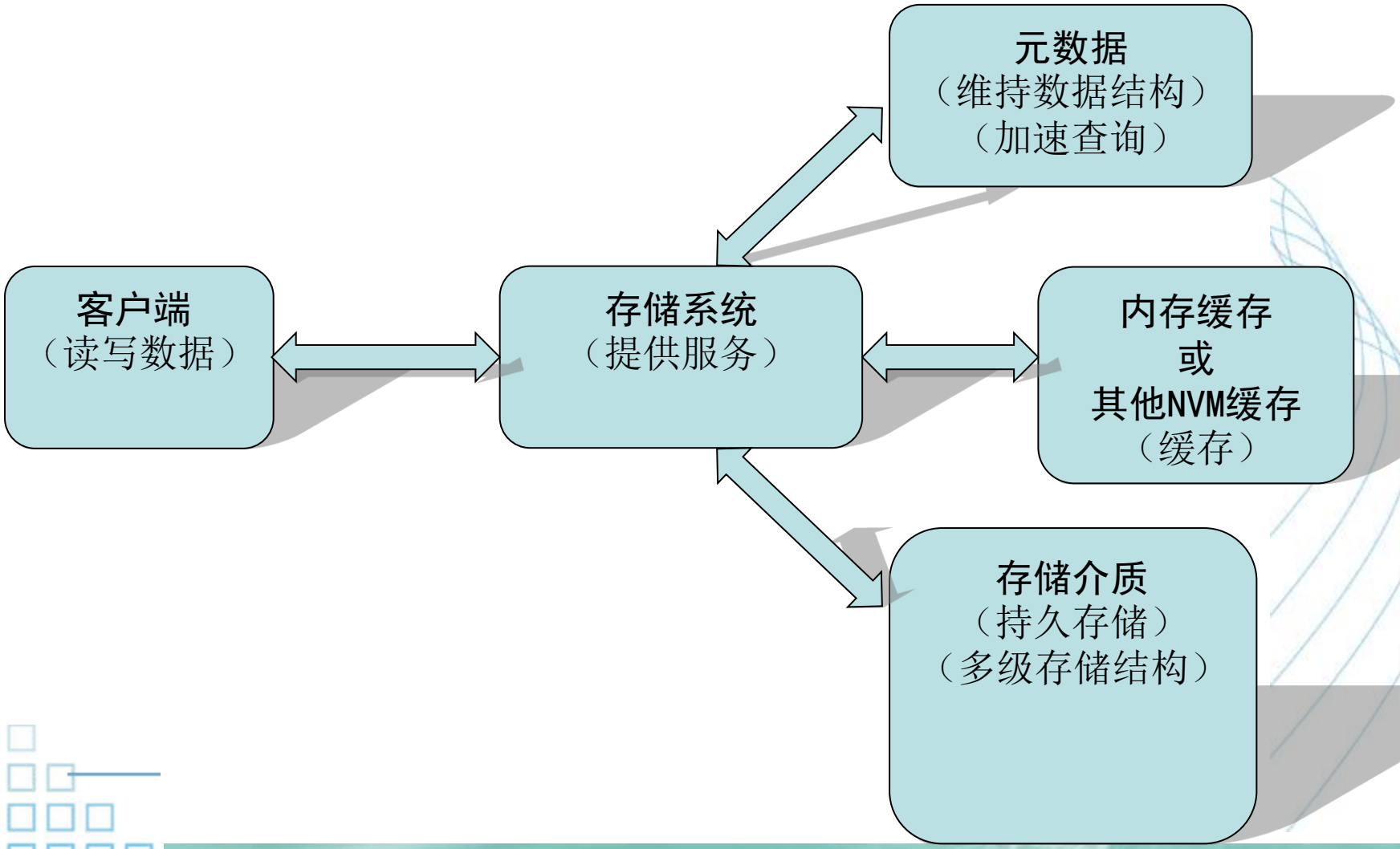
教学组：**胡春明**,沃天宇,**林学练**,李博
hucm@act.buaa.edu.cn

2021年3月23日下午14:00-15:35

存储系统：概述

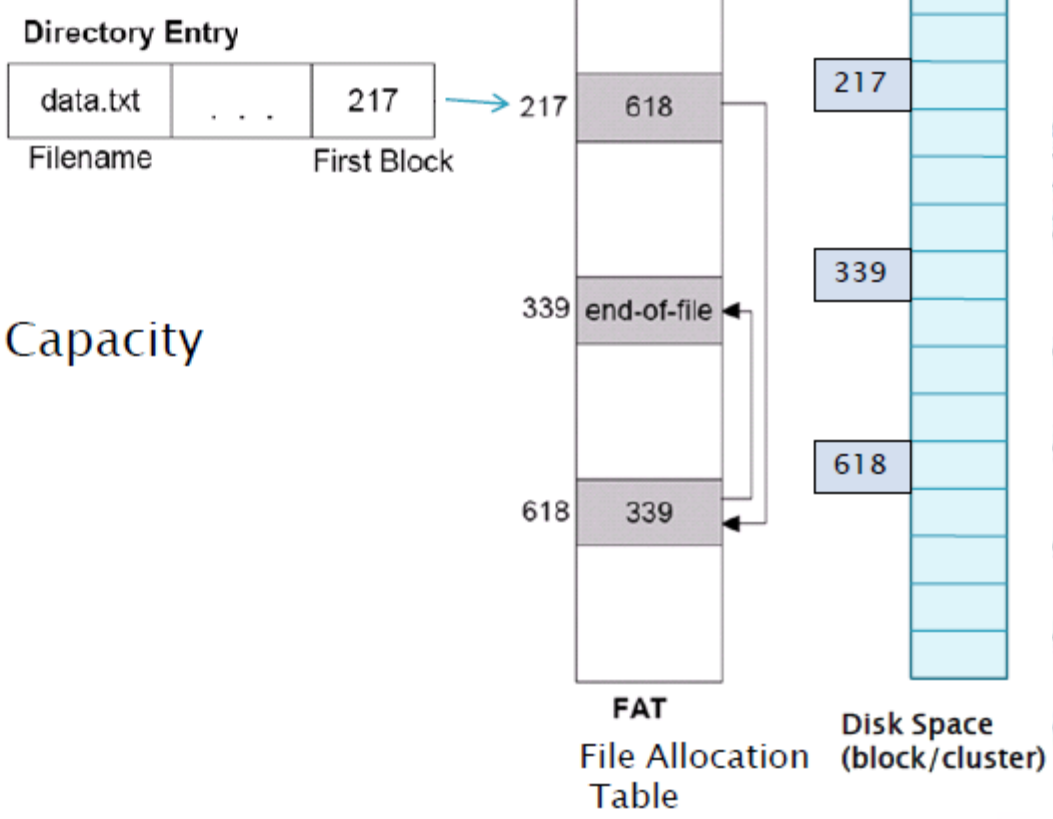
- 存储系统（数据存取系统）
  - 数据模型（Data Model）
  - 资源管理（介质管理、元数据管理）
  - 数据存取（读/写的处理、查询处理）
- 好与不好的评价
  - 一致性、可用性（可靠性）
  - 性能： capacity，IOPS、Throughput、bandwidth，latency，concurrency， …
  - 扩展性（横向、纵向）

存储系统：一个抽象理解



一个例子：文件系统

- Objective
  - ① Data Modal and Query
  - ② Storage Device
  - ③ Metadata
  - ④ *Consistency*
  - ⑤ *Availability*
  - ⑥ Performance and Capacity
  - ⑦ *Scalability*

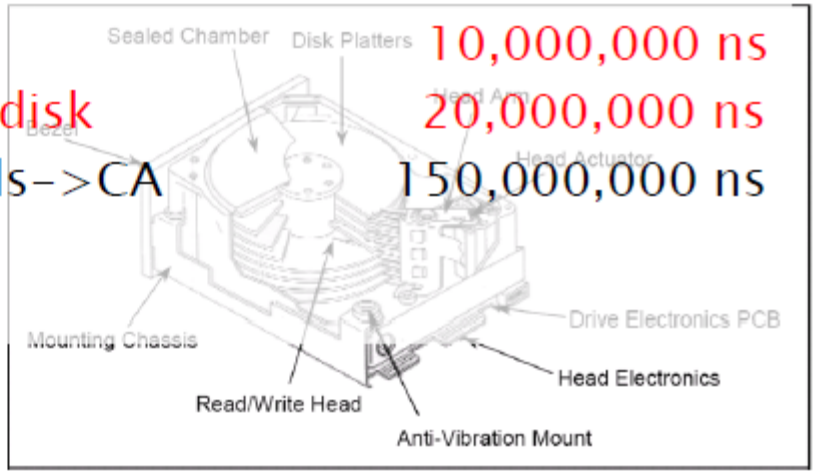


文件系统的优化

- 容量：
  - 不够大？
- 读写性能与延迟：
  - 理解造成性能延迟的原因
  - 局部性与缓存
  - 利用数据本身的特点？
- 并发访问能力：

文件系统的优化

L1 cache reference	0.5 ns
L2 cache reference	7 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same data center	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns



文件系统的优化（1）

- 优化方法：
  - 机械磁盘：减少寻道开销
    - 磁盘整理（顺序读 vs 随机读）
    - 元数据的存储（读文件=读元数据+读数据）
    - 设计和管理元数据，减少文件请求的磁盘操作

文件系统的优化（2）

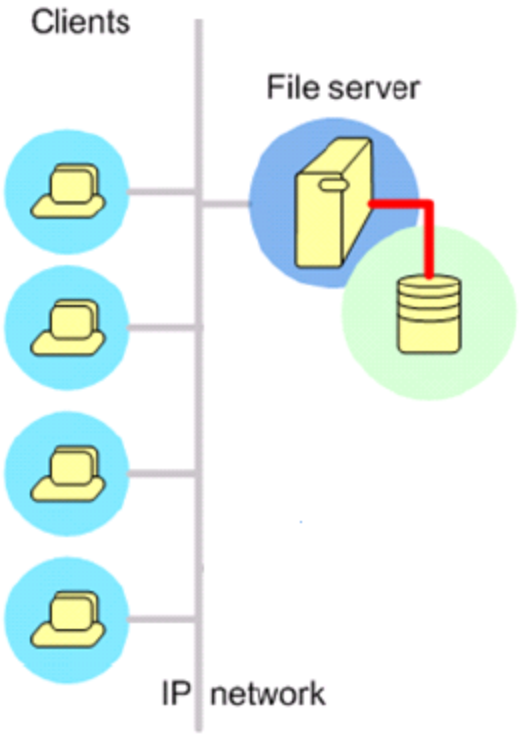
- 优化方法：
  - 机械磁盘：减少寻道开销
    - 磁盘整理（顺序读 vs 随机读）
    - 元数据的存储（读文件=读元数据+读数据）
    - 设计和管理元数据，减少文件请求的磁盘操作
  - 利用更快的介质做缓存
    - 内存? SSD磁盘；缓存数据和元数据
    - 小心处理某些操作（Log/Journaling）

文件系统的优化（3）

- 优化方法：
  - 机械磁盘：减少寻道开销
    - 磁盘整理（顺序读 vs 随机读）
    - 元数据的存储（读文件=读元数据+读数据）
    - 设计和管理元数据，减少文件请求的磁盘操作
  - 利用更快的介质做缓存
    - 内存? SSD磁盘；缓存数据和元数据
    - 小心处理某些操作（Log/Journaling）
  - 减少元数据？
    - 使用大块、变长块

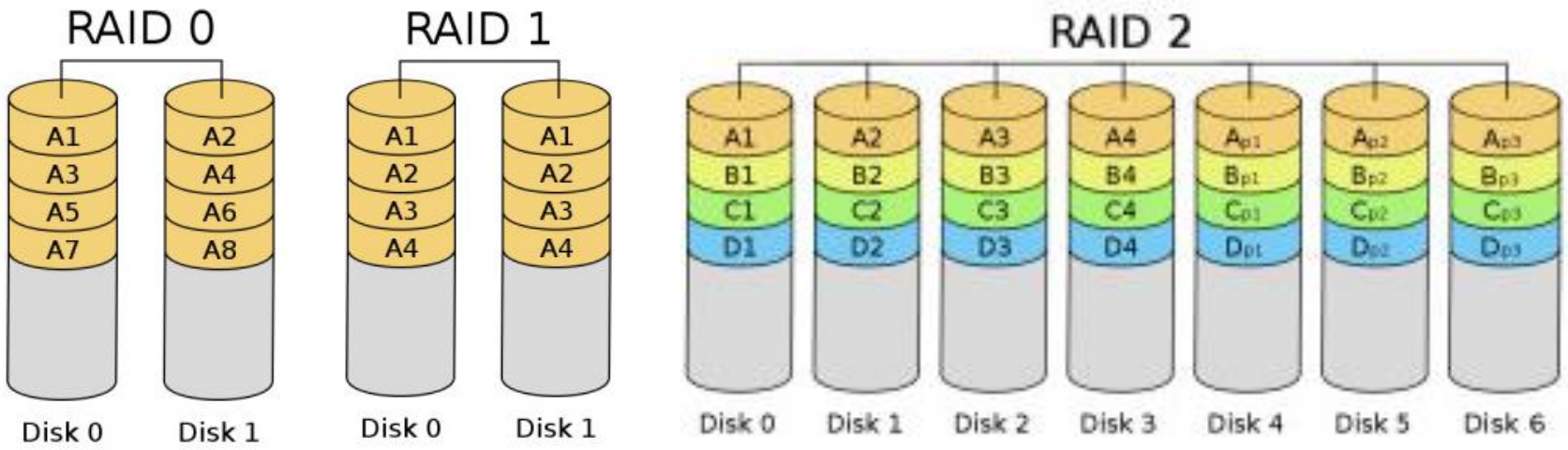
扩展方向：集中—分布

- 松耦合文件系统
  - Objective
    - Capacity
    - Sharing
  - Methods
    - Shared file system
  - Challenges
    - location transparency (dynamically maps file names to storage sites) – Mount
    - Cache – Consistency
  - Systems
    - AFS(Andrew File System), CMU,1982
    - NFS(Network File System), SUN,1985



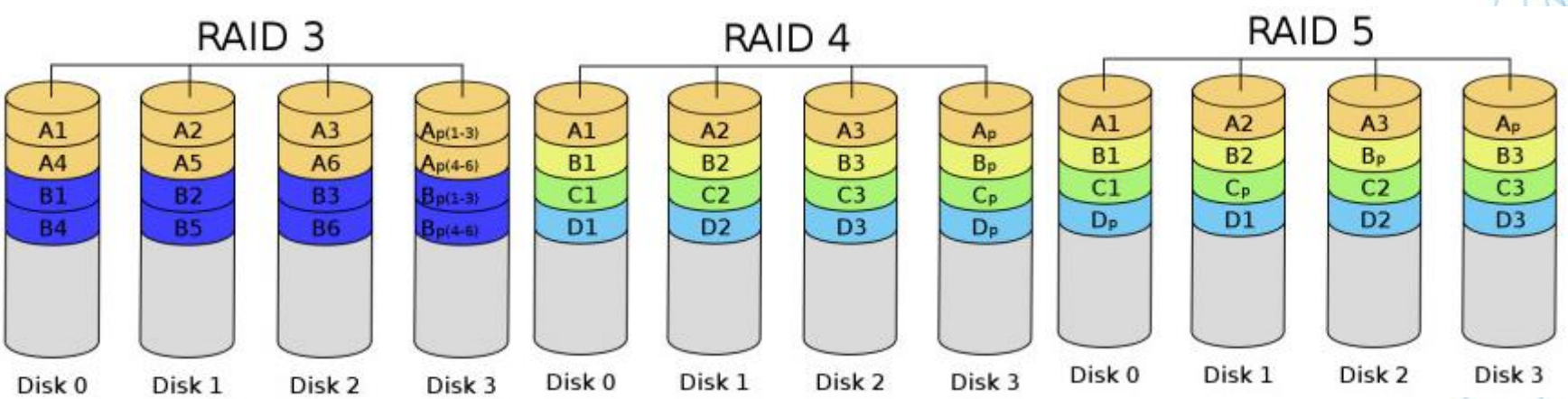
扩展方向：虚拟磁盘

- 昂贵磁盘 vs RAID：独立磁盘的冗余阵列
  - N块盘，通过RAID控制器，虚拟成一块盘
  - EDAP：Extended Data Availability and Protection



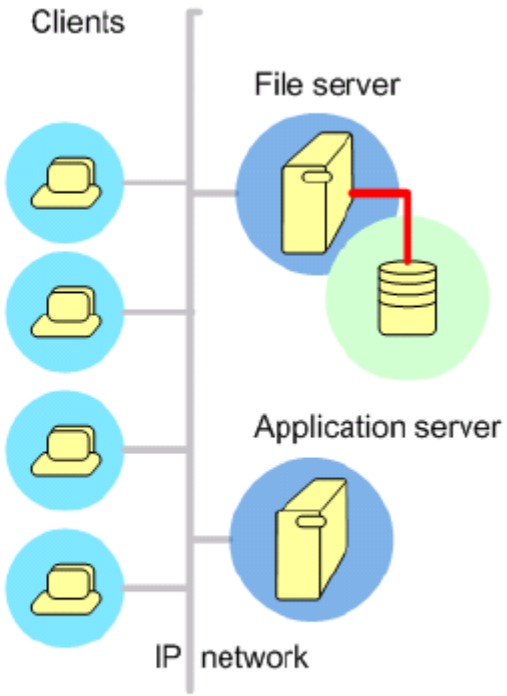
扩展方向：虚拟磁盘

- 昂贵磁盘 vs RAID：独立磁盘的冗余阵列
  - N块盘，通过RAID控制器，虚拟成一块盘
  - EDAP：Extended Data Availability and Protection



扩展方向：集中－分布

- 网络附加存储（NAS）
  - Is hard disk storage that is set up with its own network address(IP address) rather than being attached to the department computer that is serving applications to a network's workstation users.
  - **NAS** software can usually handle a number of network protocols, such as NFS, ICFS, etc.



扩展方向：集中－分布

- 共享存储的文件系统
  - Related words: Cluster File System, Parallel File System, Storage Area Network(SAN)
  - Objectives
    - Capacity
    - Performance
  - Methods
    - Clustered & Shared storage
  - Systems:
    - Petal + Frangipani
    - GPFS(General Parallel File System)
    - GFS(Global File System）

扩展方向：集中－分布

- DEC Petal/Frangipani
  - 设计目标：无中心，高可用，全局负载均衡
  - 分布式锁服务（Paxos及实现）

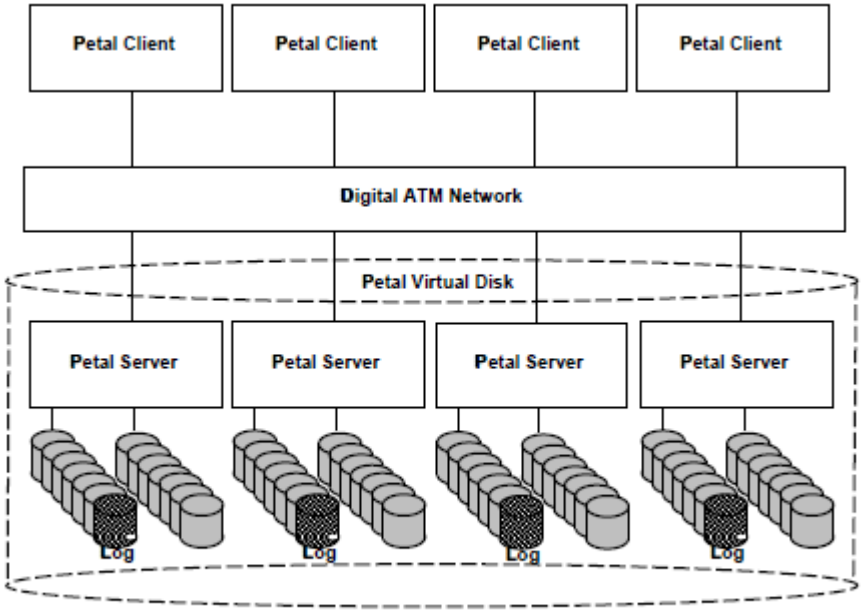


Figure 6: Petal Prototype

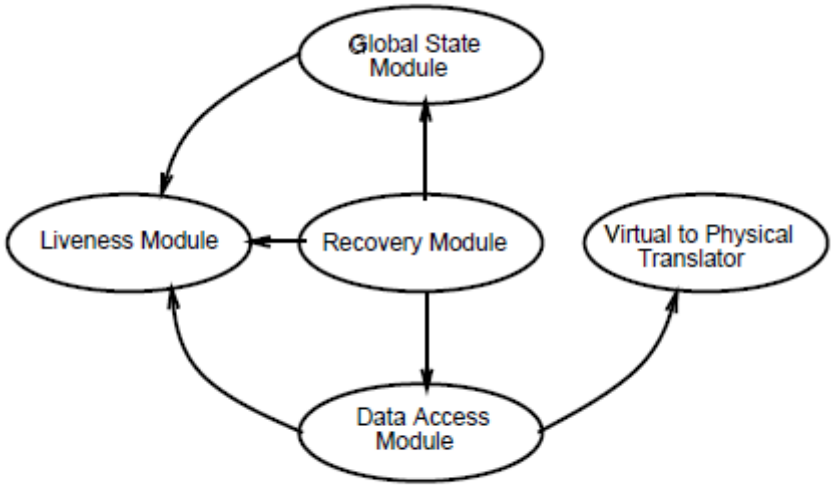
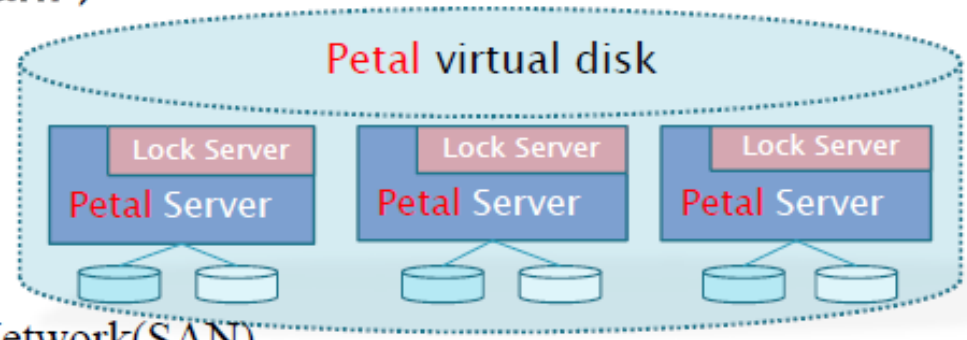


Figure 3: Petal Server Modules

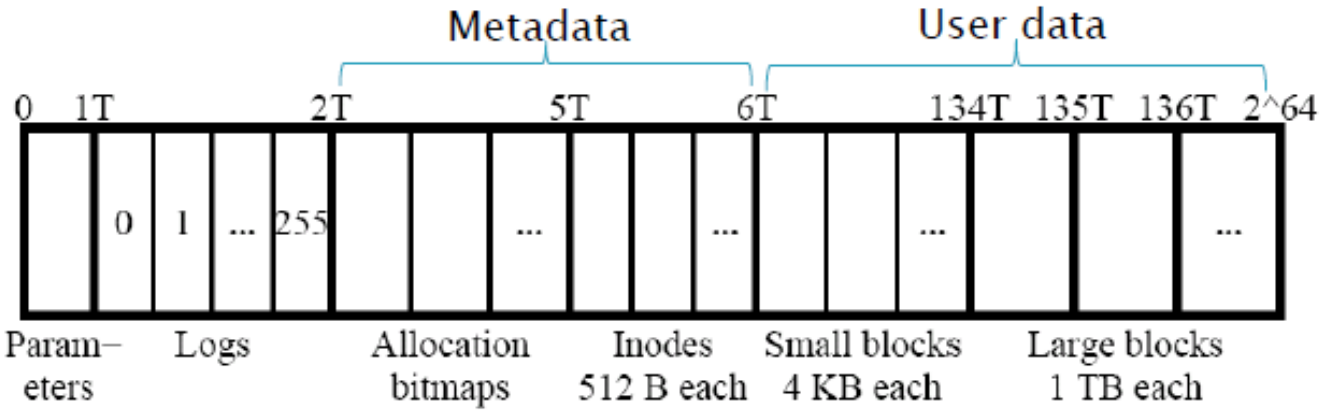
例子：DEC Petal/Frangipani

Frangipani: A Scalable Distributed File System , DEC, 1997

- Introduce a layered system design model
  - File System (**Frangipani**)
  - Virtual Disk (Petal)
  - Physical Disks

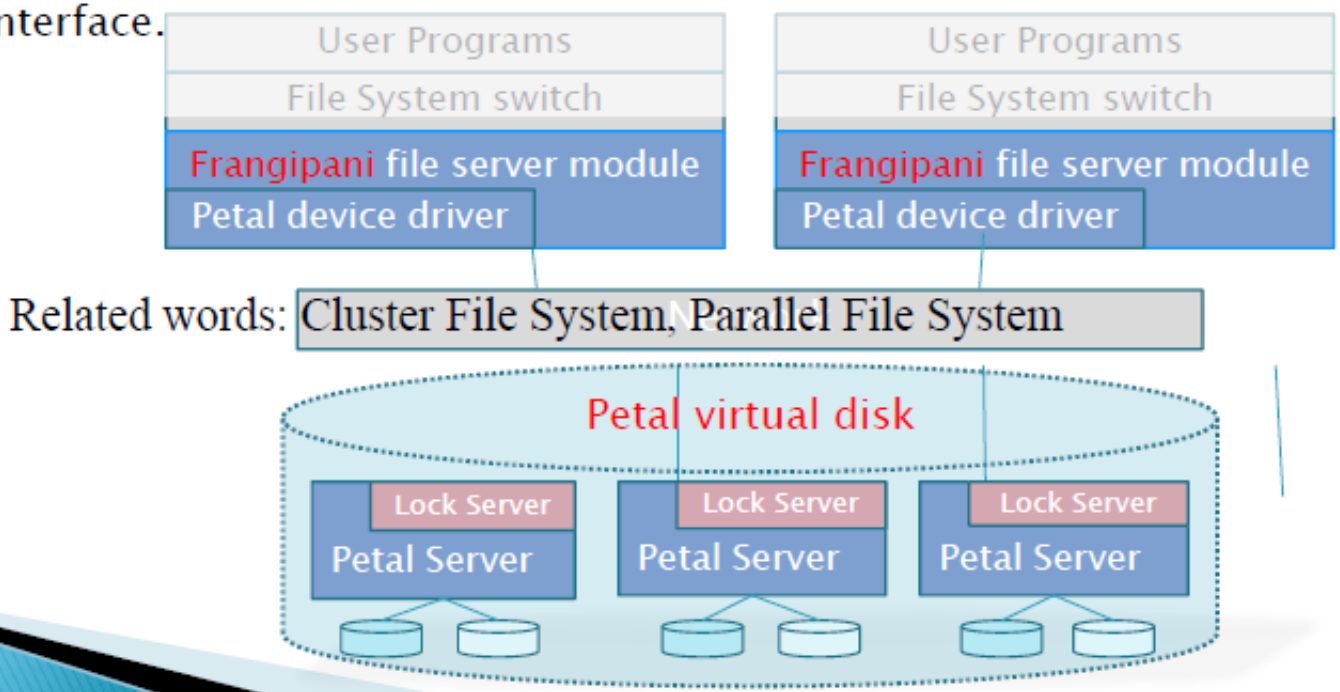


Related words: Storage Area Network(SAN)



例子：DEC Petal/Frangipani

- **Frangipani** : Behaves like a local file system
  - Multiple machines cooperatively manage a Petal disk
  - Users on any machine see a consistent view of data
  - The Frangipani file server module on each machine runs within the operating system kernel.
    - User programs access Frangipani through the standard operating system call interface.



例子：分布式文件系统

The Google File System, Sanjay Ghemawat, et al., sosp 2003

- Background
  - Google stores dozens of copies of the entire Web!
  - PB data, a few million files, span from 100 MB to Multi-GB
  - More than 15,000 commodity-class PC's.
  - Multiple clusters distributed worldwide.

  - Thousands of queries served per second.
  - One query consumes 10's of billions of CPU cycles.
  - One query reads 100's of MB of data.

  - Large streaming reads and small random reads
  - Concurrently append to the same data file

Typical first year for a new cluster:

(Jeff Dean, Google Fellow)

- ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
- ~1 PDU (Power Distribution Unit) failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
- ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
- ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- ~5 racks go wonky (40-80 machines see 50% packet loss)
- ~1 network rewiring (rolling ~5% of machines down over 2-day span)
- ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
- ~12 router reloads (takes out DNS and external vips for a couple minutes)
- ~3 router failures (have to immediately pull traffic for an hour)
- ~dozens of minor 30-second blips for dns
- ~1000 individual machine failures
- ~thousands of hard drive failures
- slow disks, bad memory, misconfigured machines, etc.

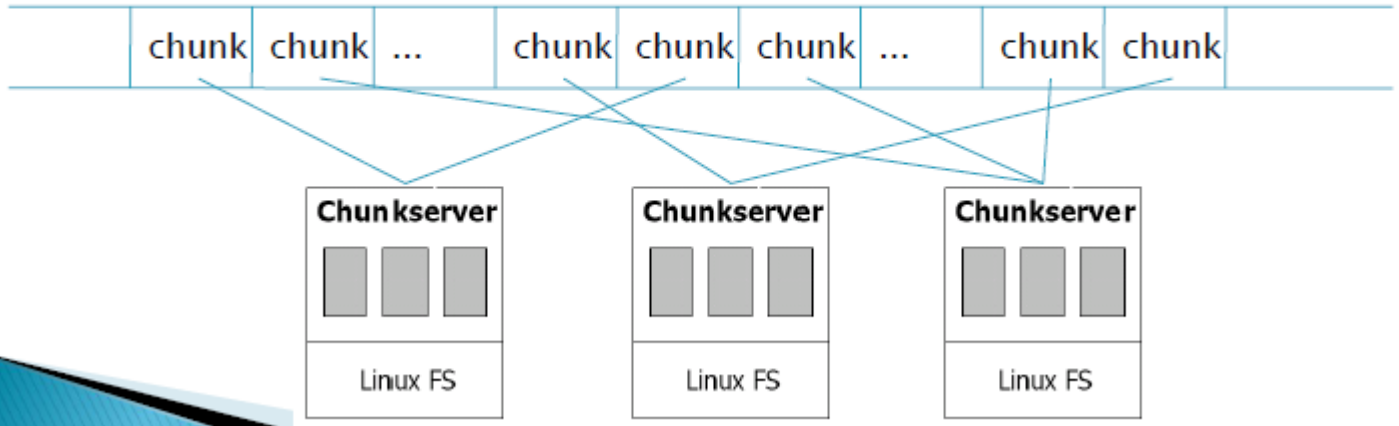
例子：分布式文件系统

- 设计出发点
  - Files are huge by traditional standards. Multi-GB files are common
    - Standard I/O assumptions (e.g. block size) have to be re-examined.
  - Component failures are the norm rather than the exception
    - Fault-tolerance and auto-recovery need to be built into the system.
  - Record appends are the prevalent form of writing.

数据模型和查询

- 文件系统抽象
  - 文件按照层次组织
  - 常见操作
    - *Create, delete, open, close, read, and write*
    - *Snapshot, record append*

Chunk replicated across multiple chunkservers



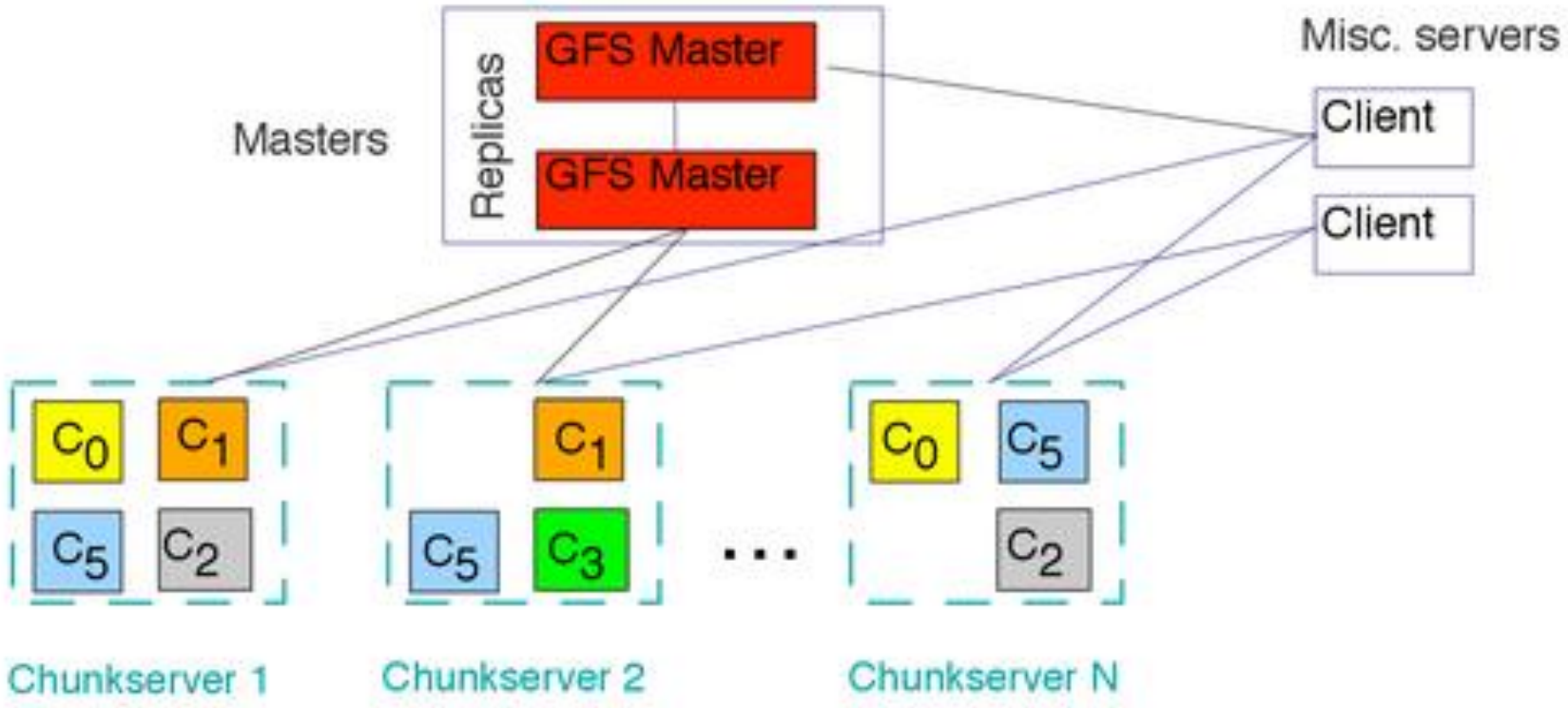
元数据和映射

- All metadata is stored in a single process running on a separate machine, called Master
  - File namespace
    - A few Million files
  - File to Chunk mappings
    - ~10 million chunks
  - Chunk location information
    - 64 bytes for each Chunk
  - Access control information
  - Chunk version numbers

~1P byte data
~1G bytes metadata in master

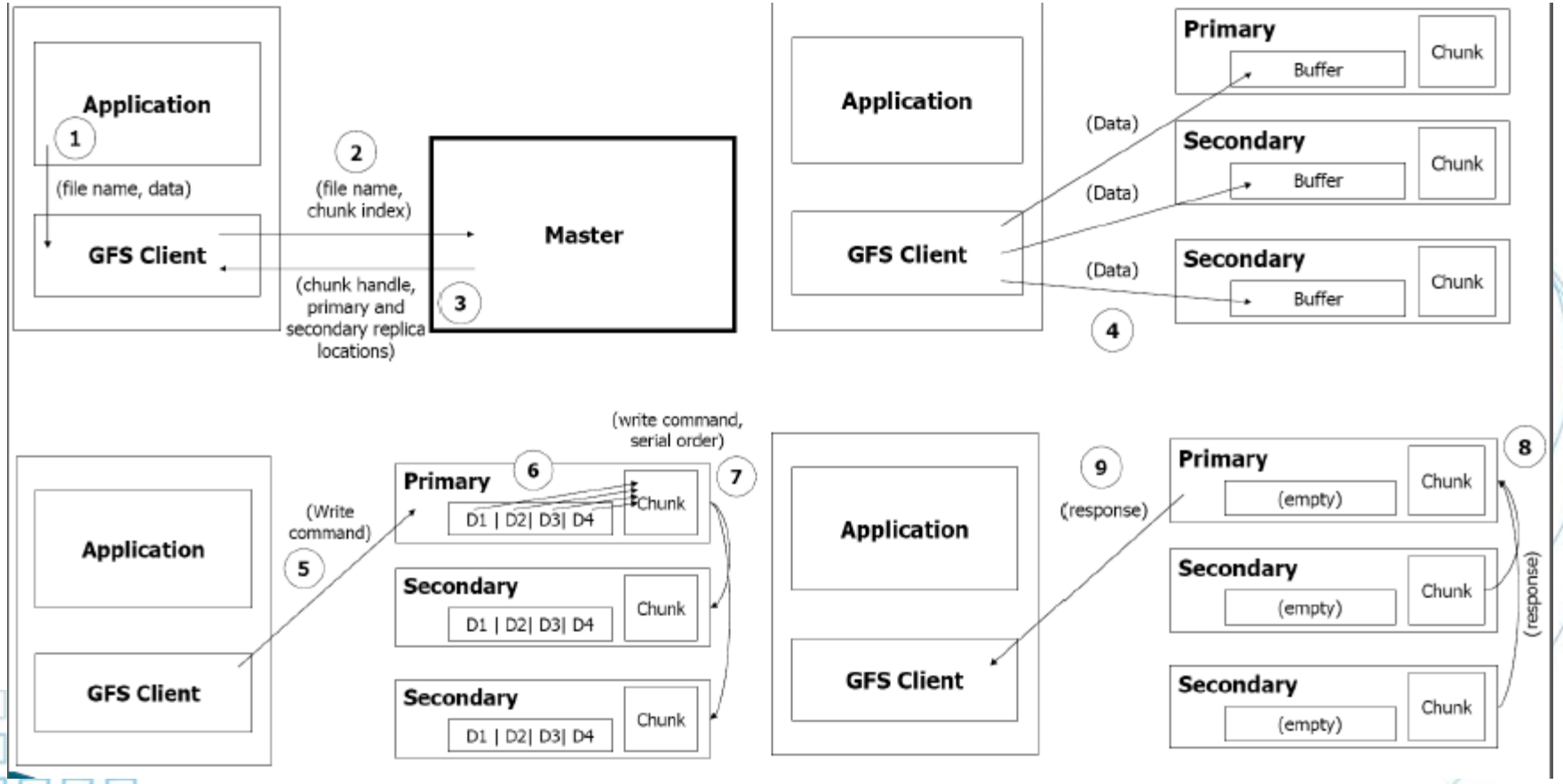
例子：分布式文件系统

- GFS, HDFS



例子：分布式文件系统

- GFS中的写操作



GFS的性能

- Real data (Raw data, Derived Data), applications and users on Internet
  - Data scale: PB+
  - Number of Storage Device: 1K+
- Customization("Google applications and GFS should be co-designed")
  - Special design of Data Model, Metadata, Data read, write and process
  - High Availability (so Reliability) and High Performance
    - Chunk Replication
  - Relaxed consistency model

Hadoop File System (HDFS)

Source: Hadoop Wiki, September 2009

HDFS能做什么？

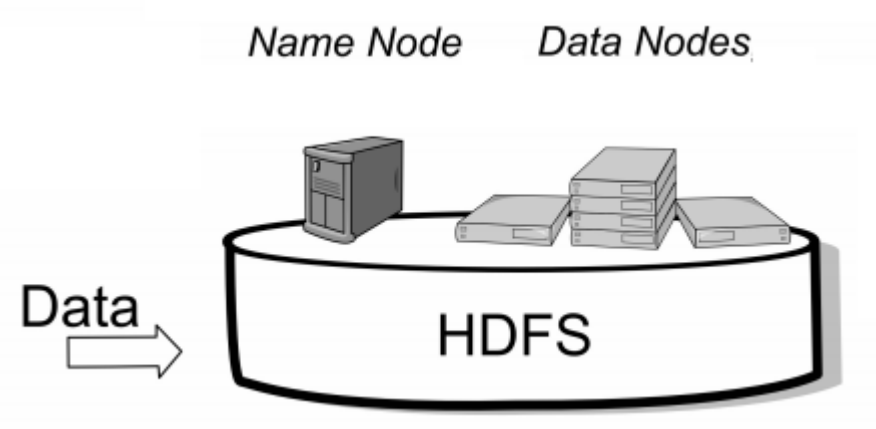
- 存储并管理PB级数据
- 处理非结构化数据
- 注重数据处理的吞吐量（latency不敏感）
- 应用模式为：write-once-read-many存取模式

HDFS不适合做什么？

- 存储小文件 (不建议使用)
- 大量的随机读 (不建议使用)
- 需要对文件的修改 (不支持)

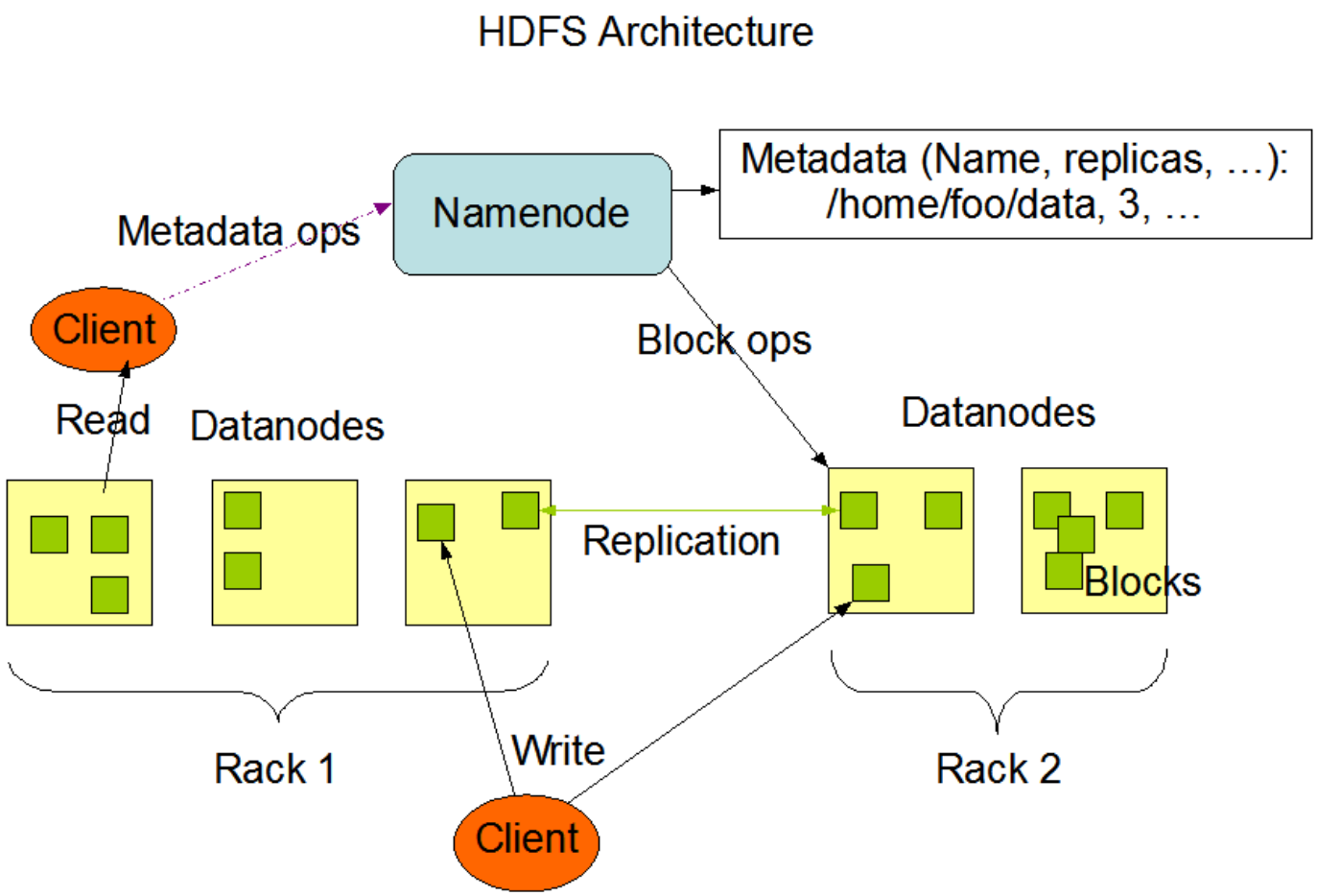
- 不支持文件并发写入
- 不支持文件修改

HDFS主要组件的功能



NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block，datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

系统架构



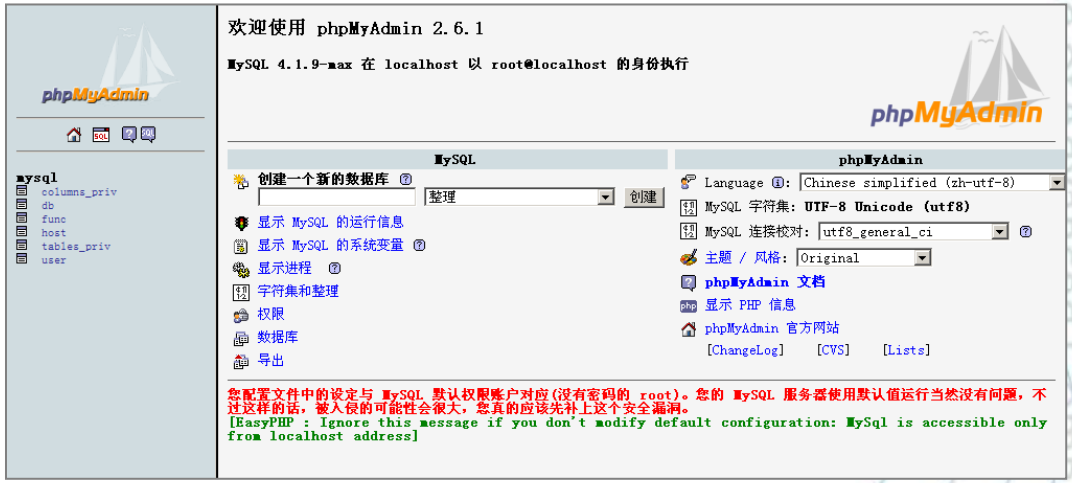
扩展2：数据的抽象

- 文件：一种数据的抽象

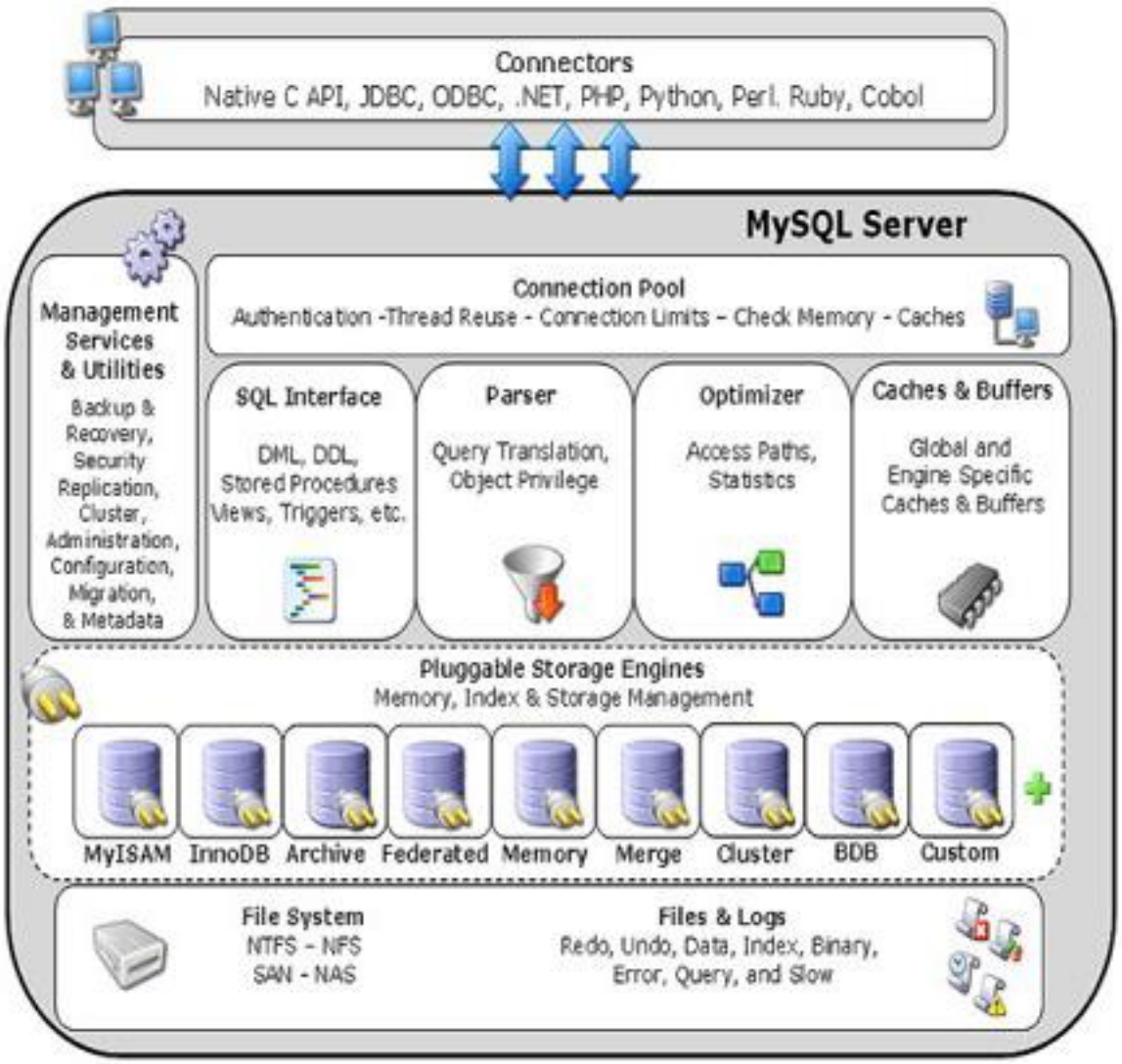
- 其他扩展？时代特征？
  - 存结构化数据的文件系统？RDB
  - 存小文件（微博图片）的文件系统？
  - 存“对象”的文件系统？
  - 存“名值对”的文件系统
  - ……

数据库系统

- 常见的数据库系统
  - 商业软件
    - MS SQL Server
    - IBM DB2
    - Oracle
  - 开源软件
    - mySQL http://www.mysql.com
    - postgreSQL http://www.postgresql.org/
    - phpMyAdmin 图形化的mySQL管理界面

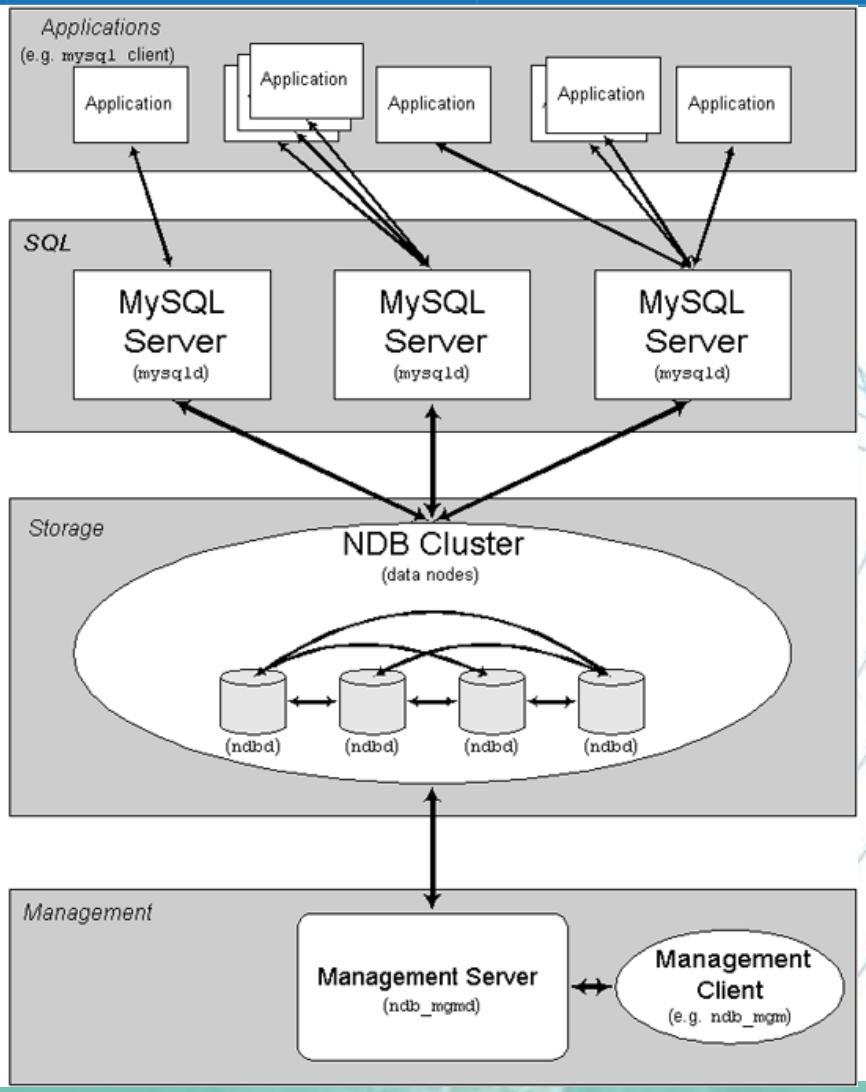


MySQL


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, Cobol
MySQL Server
Management Services & Utilities
Backup & Recovery, Security Replication, Cluster, Administration, Configuration, Migration, & Metadata
Connection Pool
Authentication -Thread Reuse - Connection Limits - Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Archive
Federated
Memory
Merge
Cluster
BDB
Custom
File System
NTFS - NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

数据库集群

- 配置数据库集群



对象存储

- Objectives
  - Availability
  - Scalability
- Method:
  - Object Storage: 其核心思想为通过在存储设备（磁盘）上加上CPU，使其具有一定的处理能力而实现自我管理、提供基于网络的数据访问
- Systems
  - Panasas, PVFS(03), Lustre(03,07), Ceph(06)

对象存储

- 对象：一个逻辑存储单元
  - Flat name space (vs. 层次结构)
  - 自包含（data & metadata）
  - 类似文件的操作

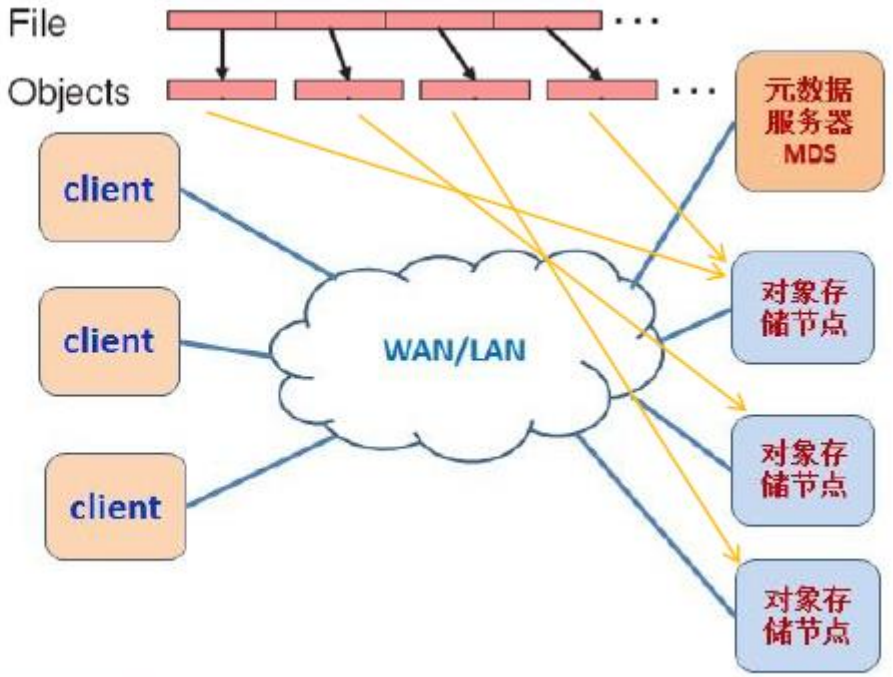
VS Shared Storage File System

顺序的存储空间–> 对象空间

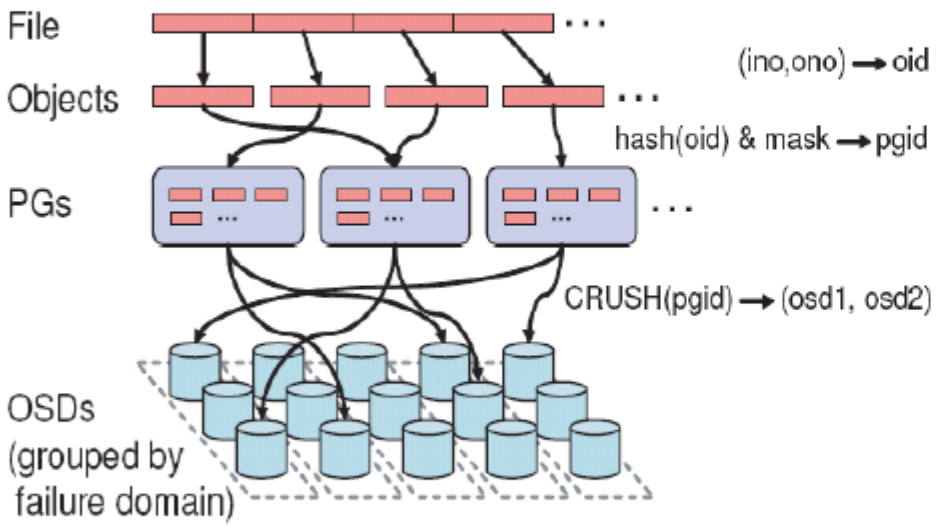
内核态–>用户态

紧耦合结构–>松耦合结构

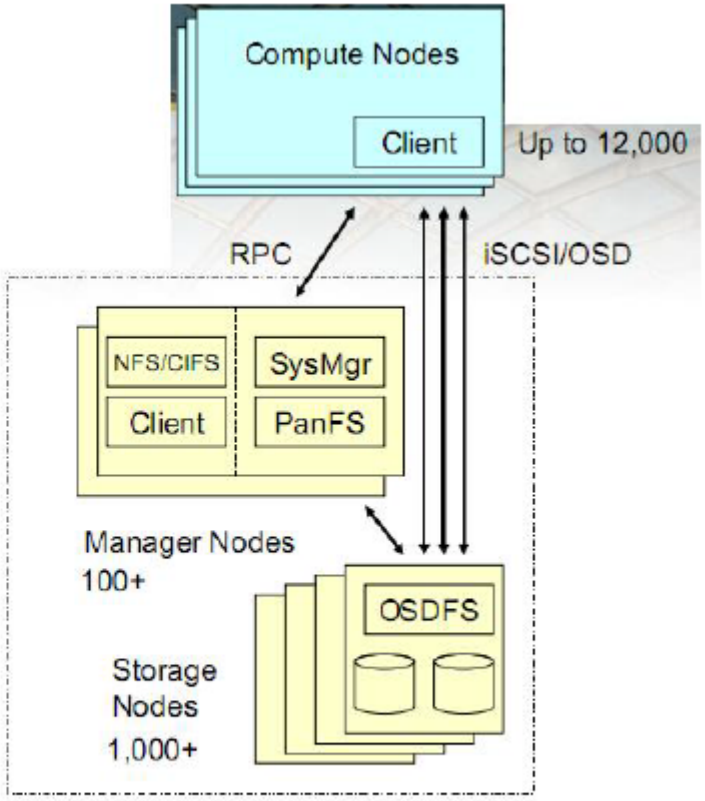
垂直扩展–>水平扩展



集中式设计 vs 分布式设计



Ceph(OSDI 06)
(Decentralization)



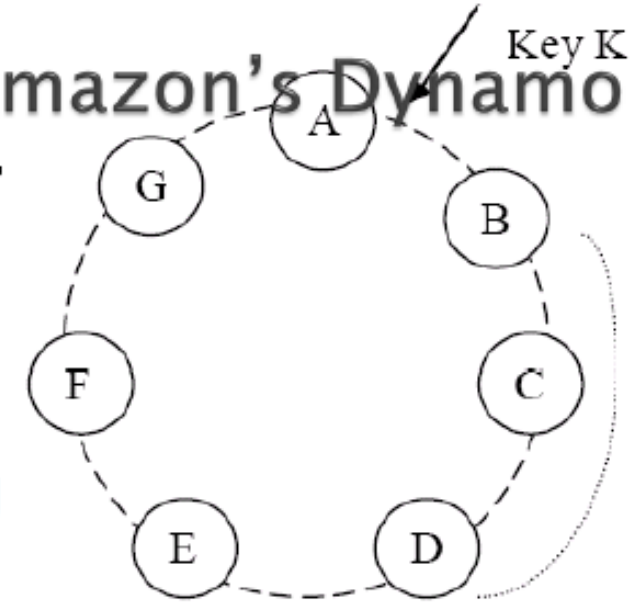
Panasas(08)
(Centralization)

K-V 存储：Dynamo

Dynamo: Amazon's Highly Available Key-value Store,
Giuseppe DeCandia, et al., sosp 2007

- Background
  - Storage for Amazon's world-wide e-commerce platform
  - Such as best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog
- Data Model and Query:
  - Key-value pairs: Data is stored as binary objects (i.e., blobs) identified by unique keys.
  - *simple read and write operations to a data item that* is uniquely identified by a key.
    - get(*key*)
    - put(*key, context, object*)

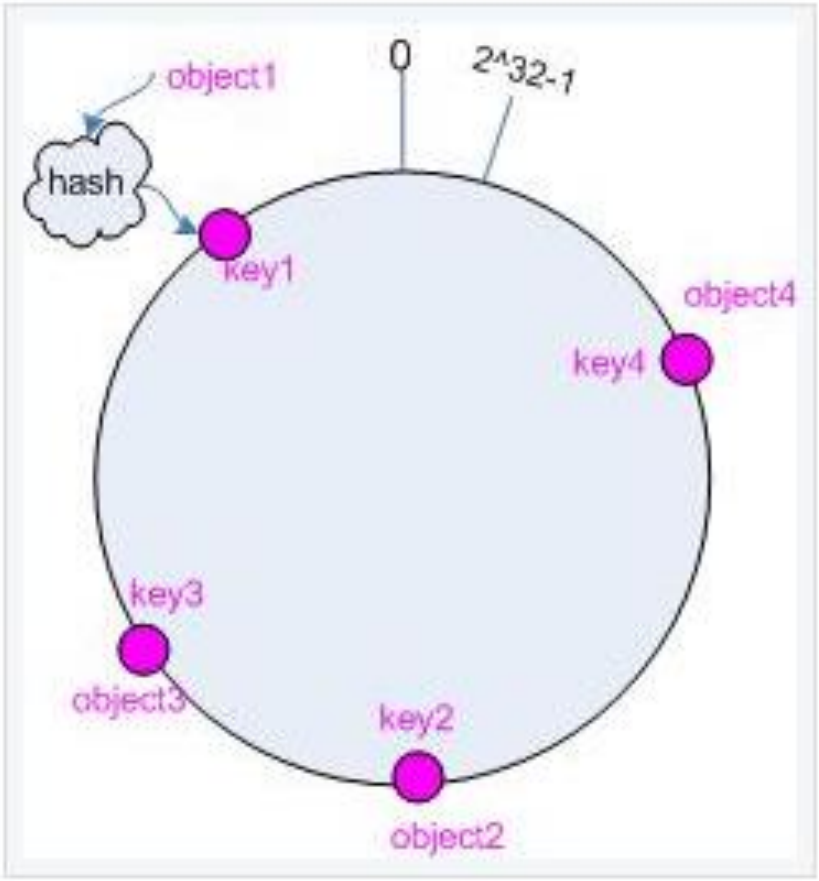
(1) Key Value Store – Amazon's Dynamo



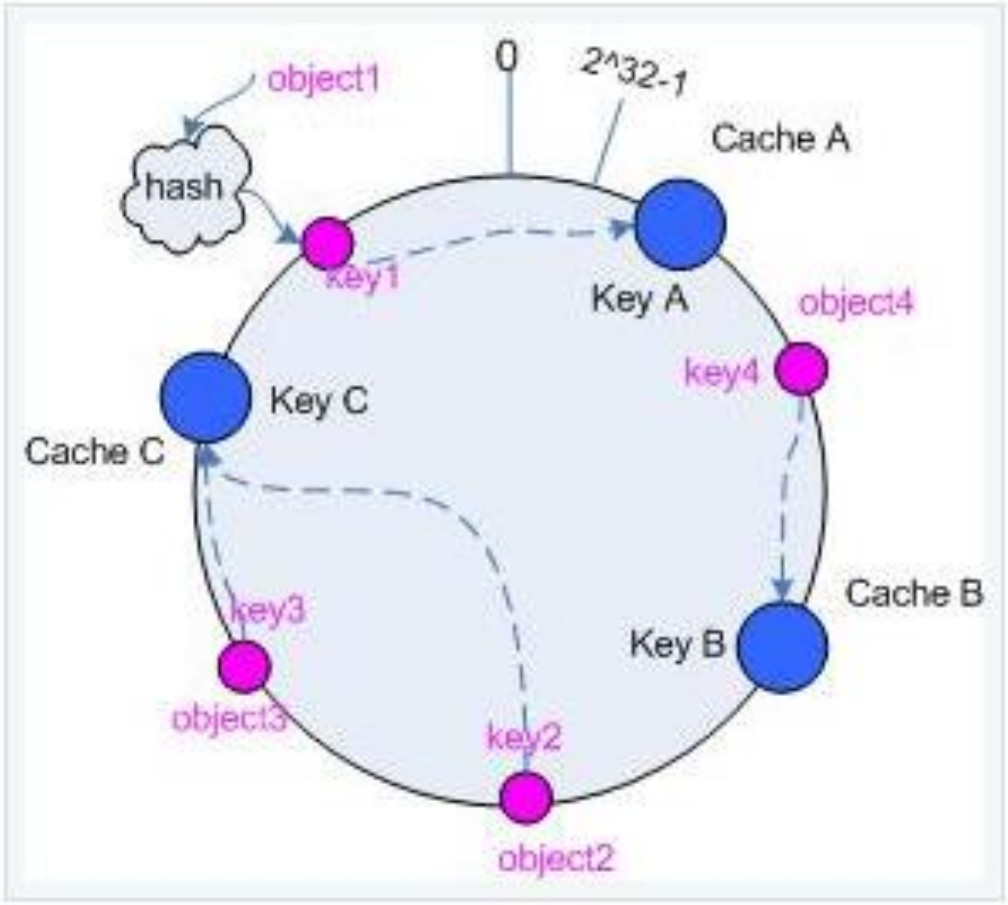
- Data Store and mapping
  - Consistent Hashing:
    - the output range of a hash function is treated as a fixed circular space or "ring" (i.e. the largest hash value wraps around to the smallest hash value).
  - Virtual Nodes
    - Each node in the system is assigned a random value within this space which represents its "position" on the ring.
    - Each node gets assigned to multiple points in the ring.
      - node B replicates the key *k* at nodes C and D in addition to storing it locally.
      - Node D will store the keys that fall in ranges (A, B], (B, C], and (C, D].

分布式缓存实现:一致性哈希

- 考虑环形空间



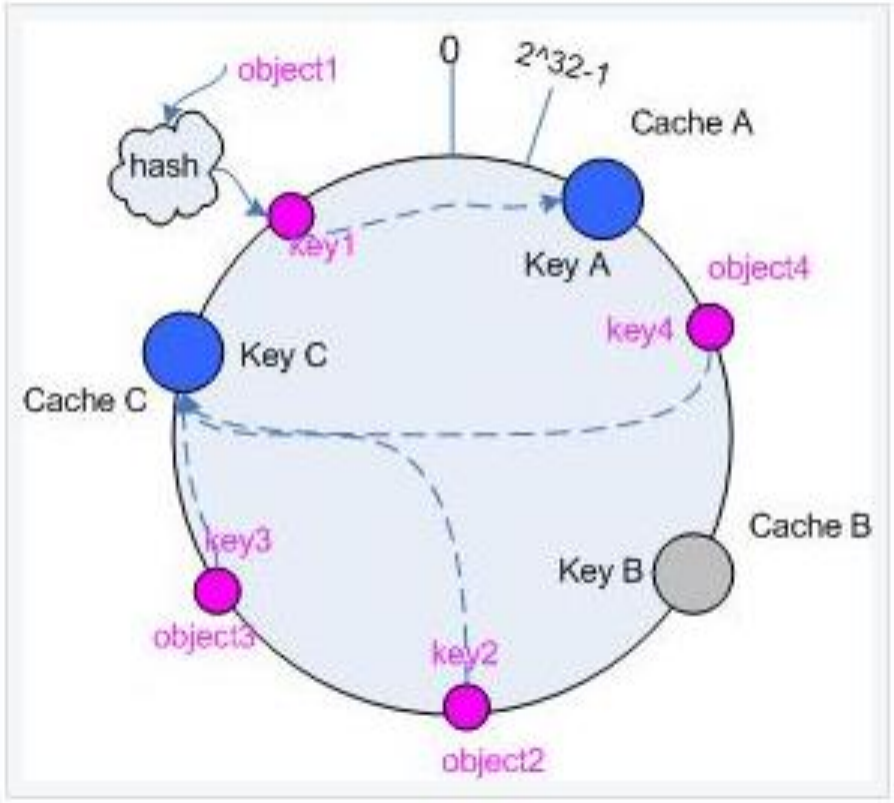
分布式缓存实现:一致性哈希

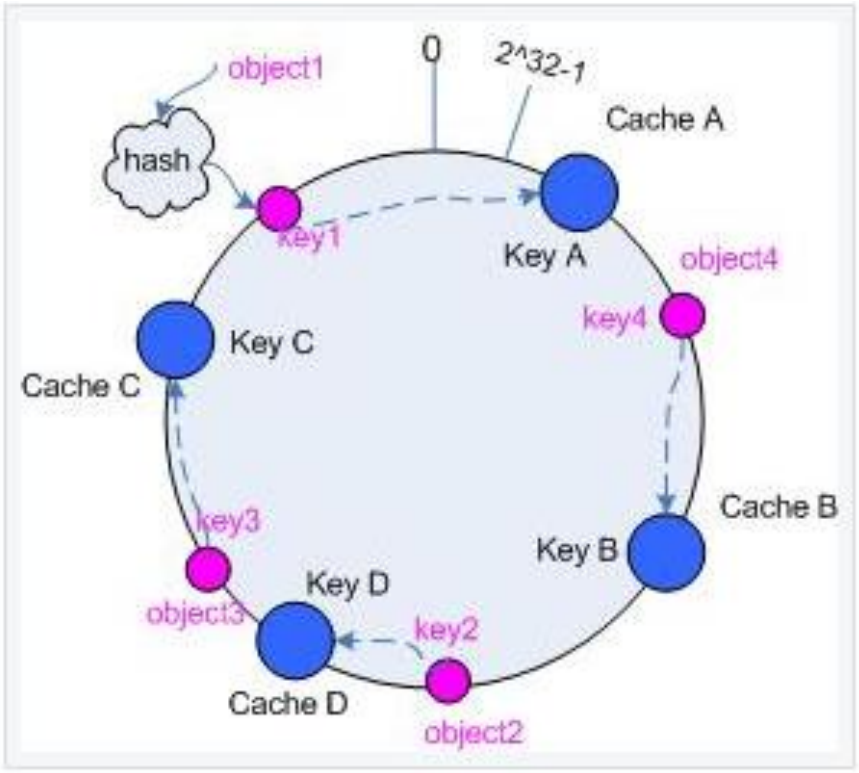



```
1       hash(cache A) = key A;
2       … …
3       hash(cache C) = key C;
```

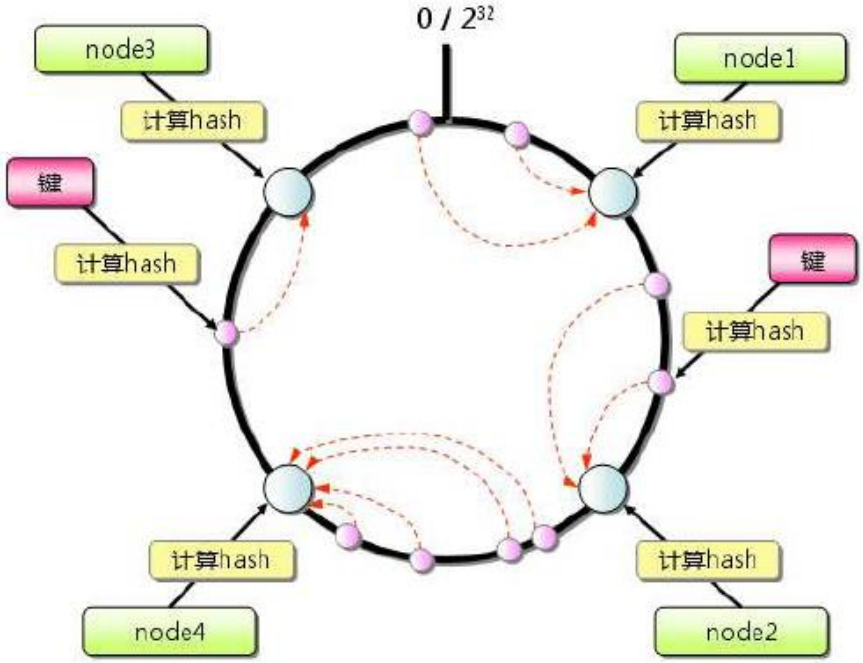
分布式缓存实现:一致性哈希

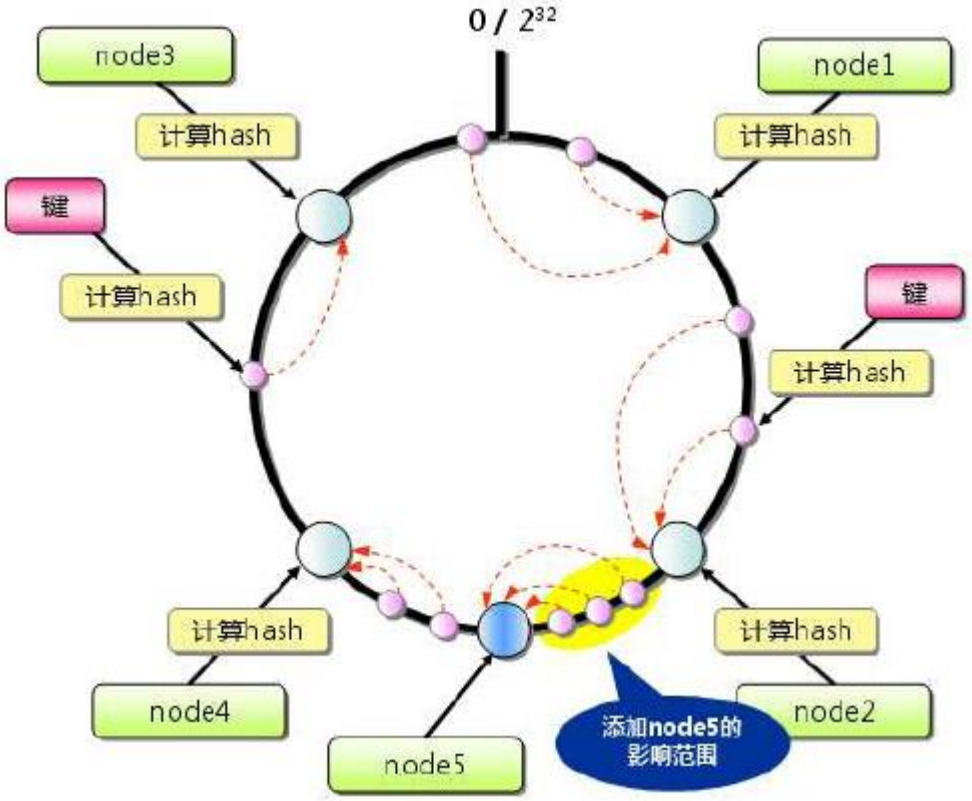
- 考虑节点失效/增加节点





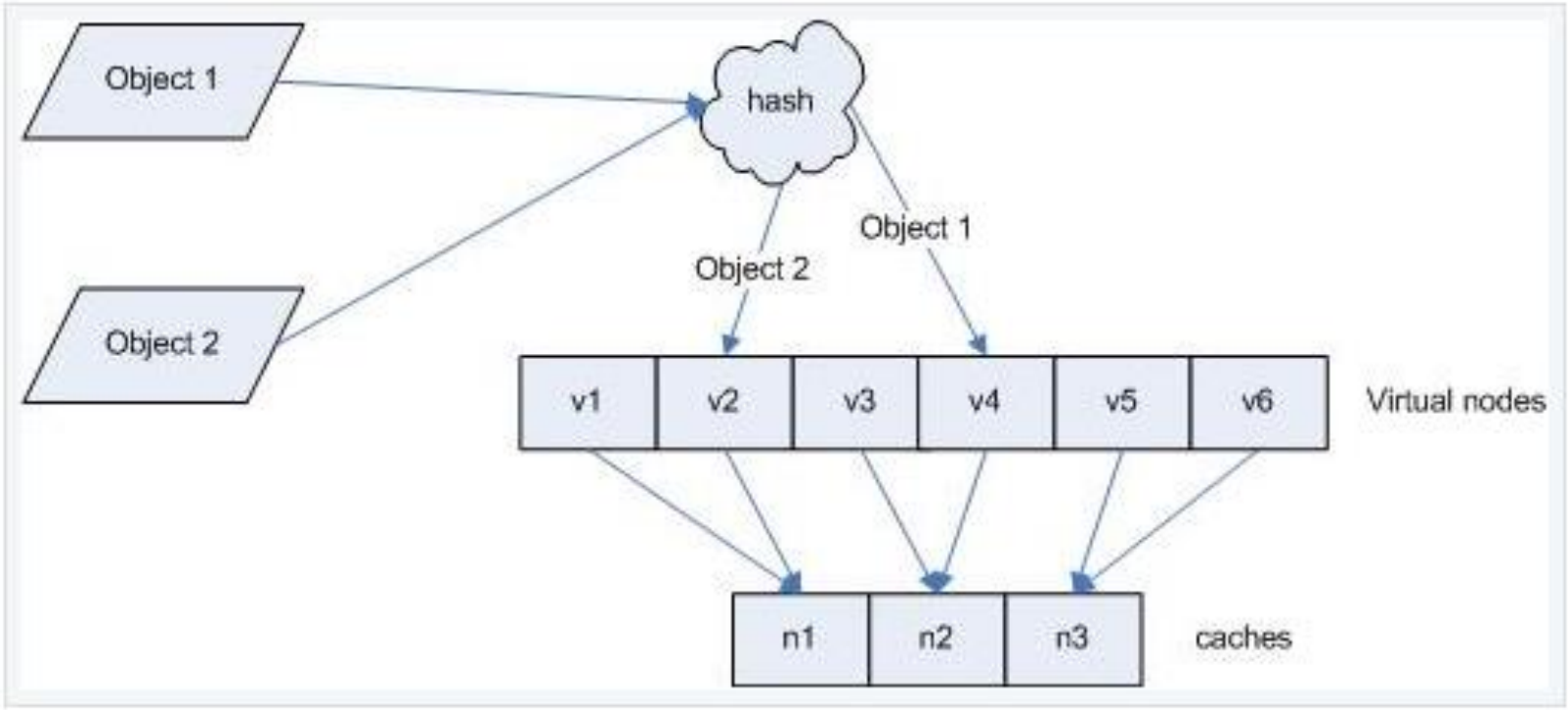
分布式缓存实现:一致性哈希



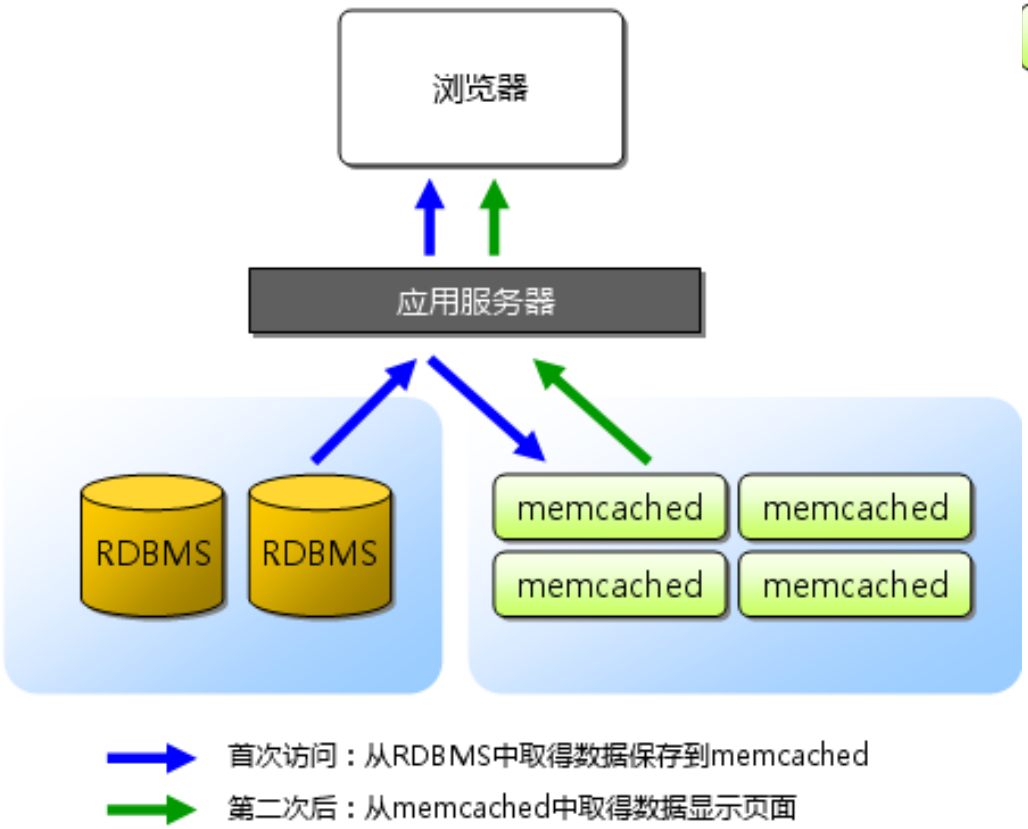


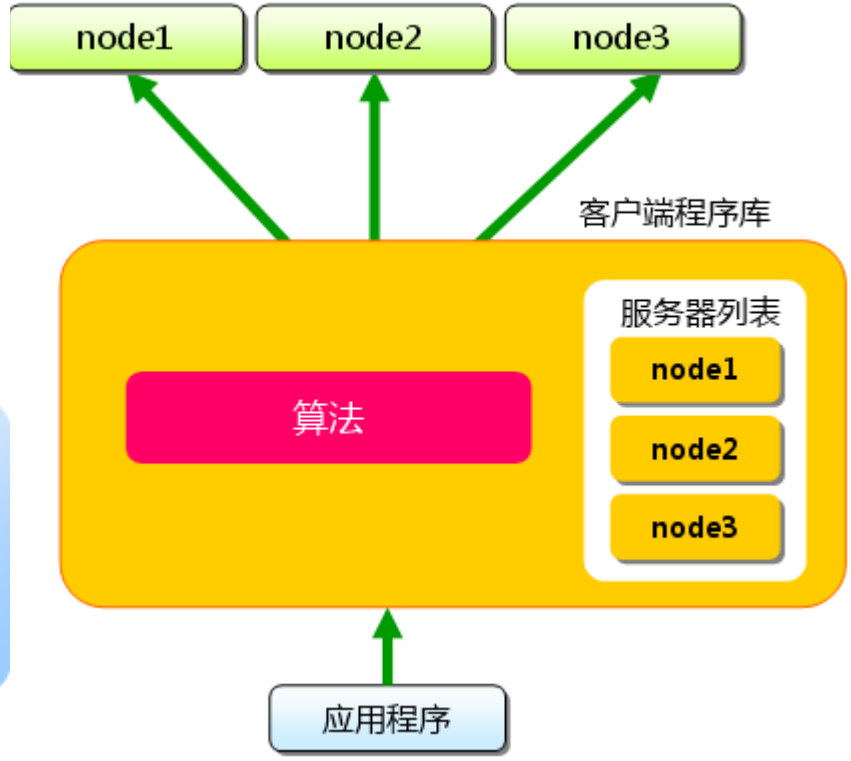
分布式缓存实现:一致性哈希

- 考虑平衡性（Balancing）
  - 引入虚拟节点（Virtual nodes）



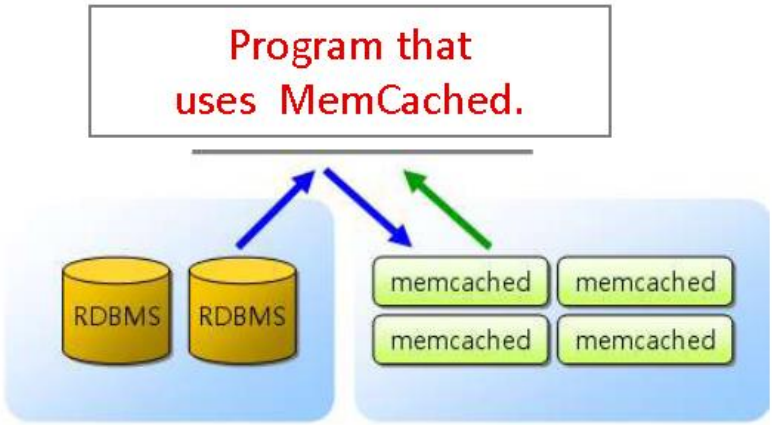
memcached


浏览器
应用服务器
RDBMS
RDBMS
memcached
memcached
memcached
memcached
首次访问：从RDBMS中取得数据保存到memcached
第二次后：从memcached中取得数据显示页面


node1
node2
node3
客户端程序库
算法
服务器列表
node1
node2
node3
应用程序

分布式缓存: MemCached

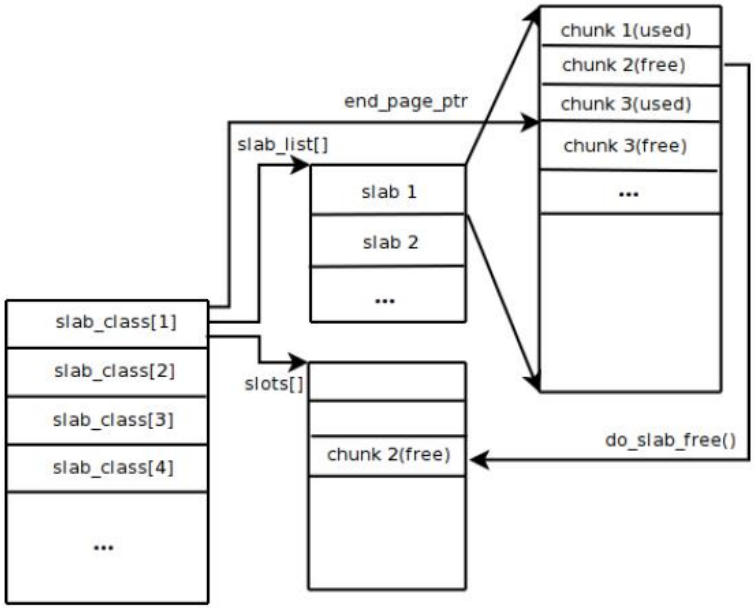
Usage of MemCached.

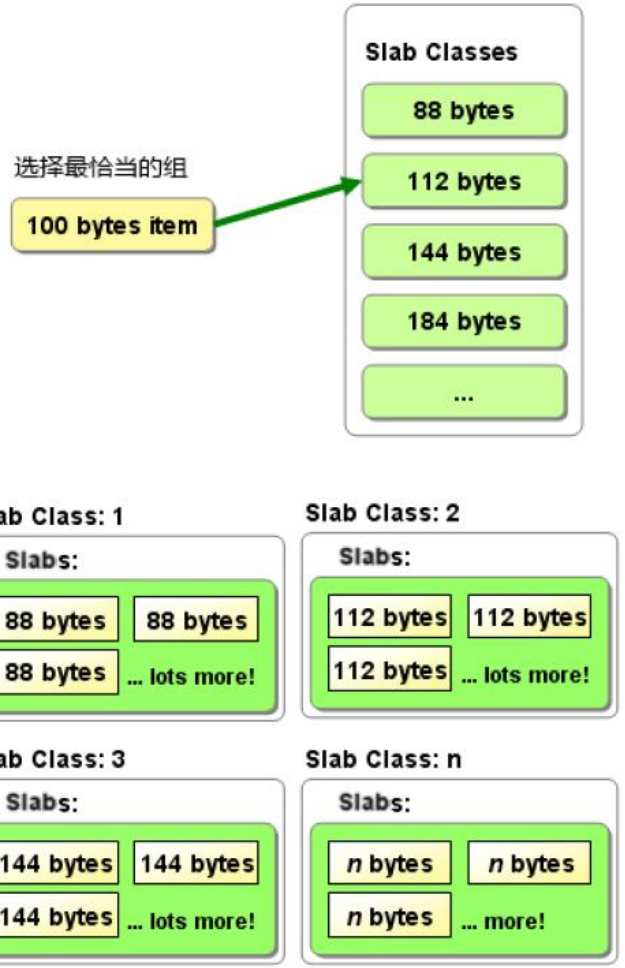



```
function get_foo(int userid) {
   /* first try the cache */
   data = memcached_fetch("userrow:" + userid);
   if (!data) {
      /* not found : request database */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* then store in cache until next get */
      memcached_add("userrow:" + userid, data);
   }
   return data;
}
```

```
function update_foo(int userid, string dbUpdateString) {
  /* first update database */
   result = db_execute(dbUpdateString);
   if (result) {
      /* database update successful : fetch data to be stored in cache */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* last line could also look like data = createDataFromDBString(dbUpdateString); */
      /* then store in cache until next get */
      memcached_set("userrow:" + userid, data);
   }
```

分布式缓存实现: MemCached





- Before caching an item,
  find the slab with the appropriate size,
  equal or a little bigger than item.

简化的RDB：Yahoo PNUTS

- PNUTS: Yahoo's Hosted Data Serving Platform, VLDB 2008

- Yahoo的应用需求：Web应用
  - 用户数据库：上亿活跃用户
  - Web会话状态：大量并发Session

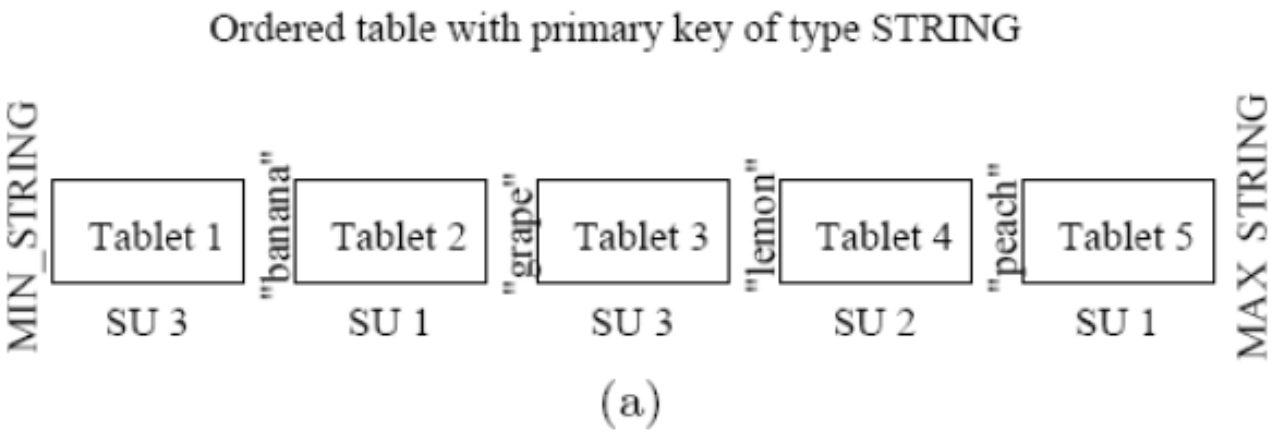
  - 社交应用：在用户间建立连接和信息共享
  - 大量的异构数据：小文件、邮件附件、图片、视频

简化的RDB：Yahoo PNUTS

- 数据模型：
  - “简单”的关系模型
  - Table抽象，并增加blob支持附件存储
  - Schema Free：可随时增加新的列
  - 支持对表的基本操作
  - 更新和删除操作需要指定主键（Selection，Projection）
  - 查询读/写一个记录，或数量较少的一组记录

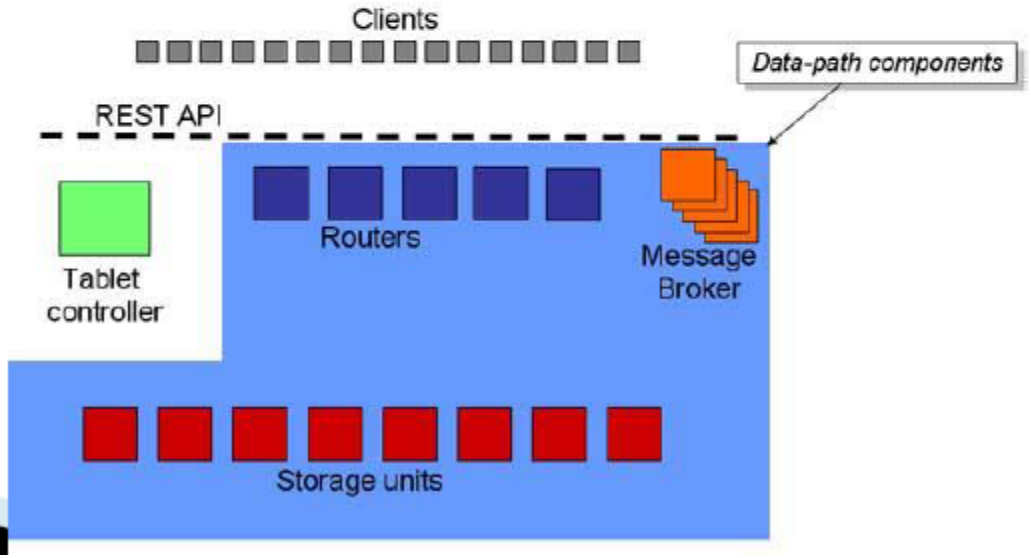
简化的RDB：Yahoo PNUTS

- Data Storage
  - Ordered table
    - table is divided into intervals, and each interval corresponds to one tablet
    - tablet is stored in storage unit.
      - For ordered tables, uses MySQL with InnoDB



简化的RDB：Yahoo PNUTS

- System and metadata
  - The tablet controller owns the mapping from table to units
    - The mapping is similar to a very large root node of a B+ tree.
  - Routers periodically poll the tablet controller to get any changes to the mapping, contain only a cached copy of the interval mapping.
    - determine which tablet contains the record
    - determine which storage unit has that tablet.



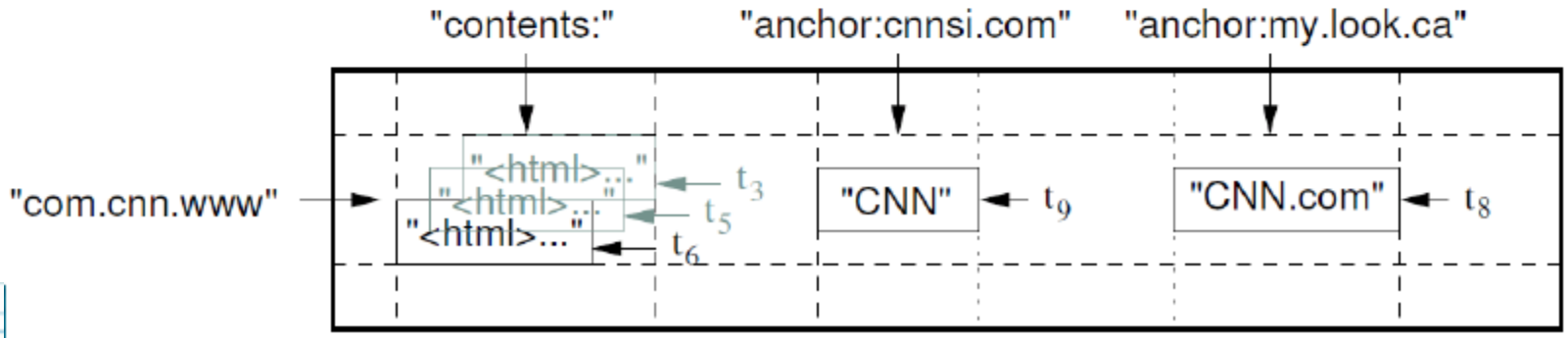
列存储 – Google Big Table

A Distributed Storage System for Structured Data , Fay Chang, et al., OSDI '06

- Motivation
  - Lots of (semi-)structured data at Google
    - URLs: Contents, crawl metadata, links, anchors, pagerank, …
    - Per-user data: User preference settings, recent queries/search results, …
    - Geographic locations: Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
  - Scale is large
    - – billions of URLs, many versions/page (~20K/version)
    - – Hundreds of millions of users, thousands of q/sec
    - – 100TB+ of satellite image data

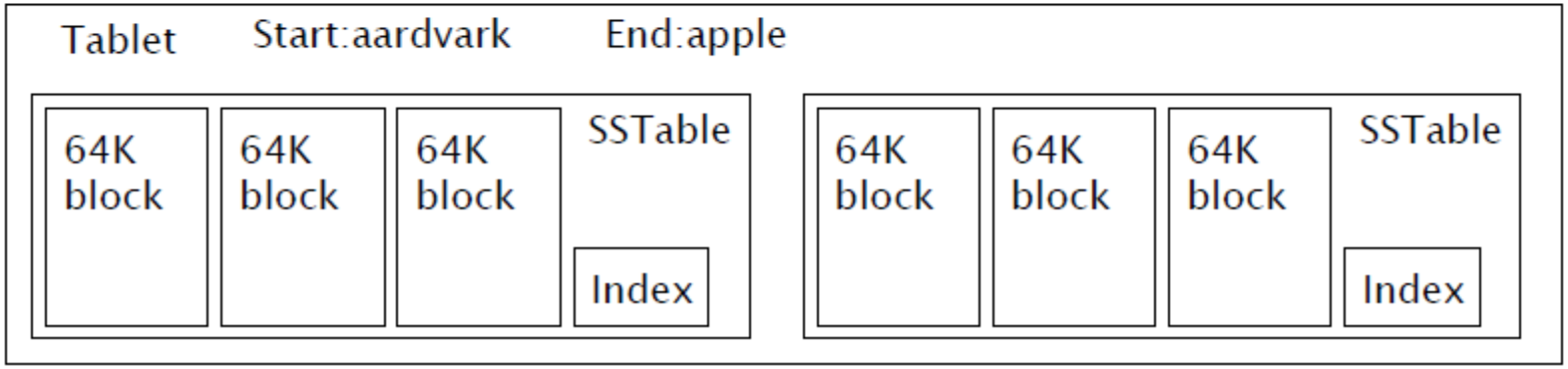
列存储 – Google Big Table

- Data model : A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map.
  - A Column Family is a NoSQL object that contains columns of related data.
    - It is a tuple (pair) that consists of a key-value pair, where the key is mapped to a value that is a set of columns.
    - analogy with relational databases, a column family is as a "table", each key-value pair being a "row".
  - Each Column is a tuple (triplet) consisting of a column name, a value, and a timestamp.[



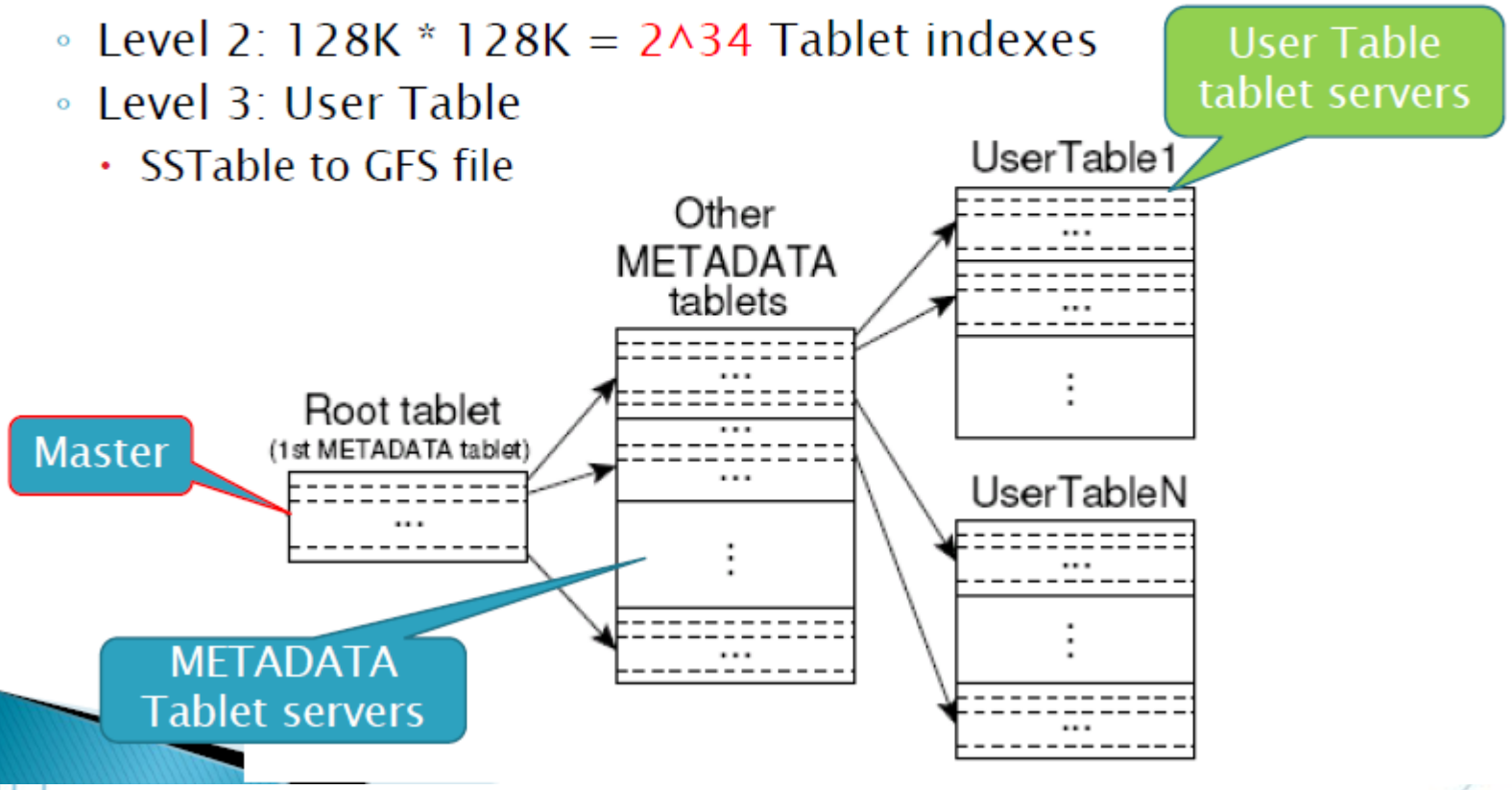
列存储 – Google Big Table

- Data Store
  - Tablets: Each table consists of a set of tablets,
    - each tablet contains all data associated with a row range.
    - Approximately 100–200 MB in size by default.
  - Tablet is built out of multiple SSTables
    - SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable).
    - Bigtable uses the distributed Google File System (GFS) to store log and data files (SSTable files).



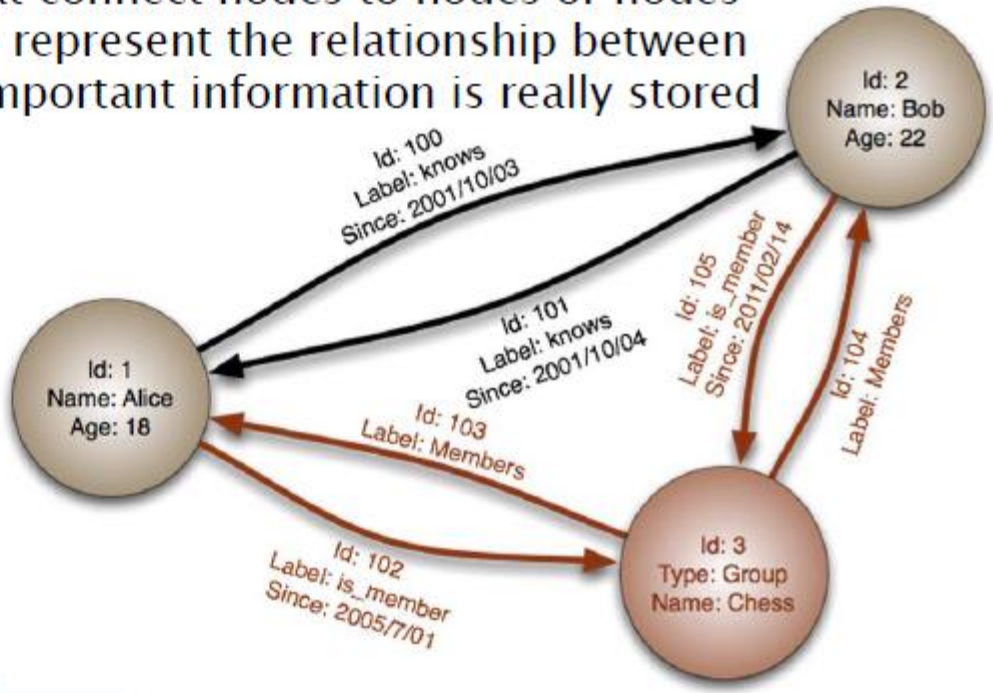
列存储 – Google Big Table

- Mapping & Metadata
  - three-level hierarchy B+ tree
  - Level 1: 128M
  - Level 2: 128K * 128K = 2^34 Tablet indexes
  - Level 3: User Table
    - SSTable to GFS file



图存储、图数据库

- Data Model:
  - Nodes represent entities such as people, businesses, accounts, or any other item you might want to keep track of. Properties are pertinent information that relate to nodes. database.
  - Edges are the lines that connect nodes to nodes or nodes to properties and they represent the relationship between the two. Most of the important information is really stored in the edges.



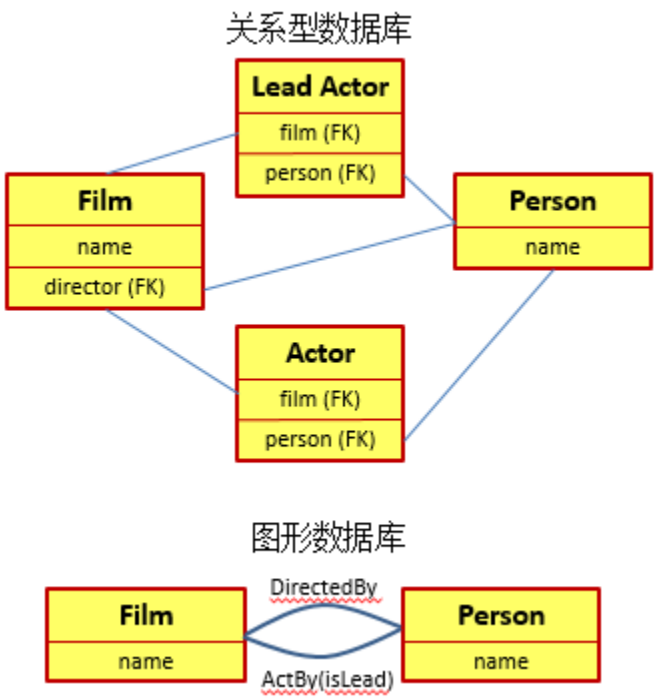
图存储、图数据库

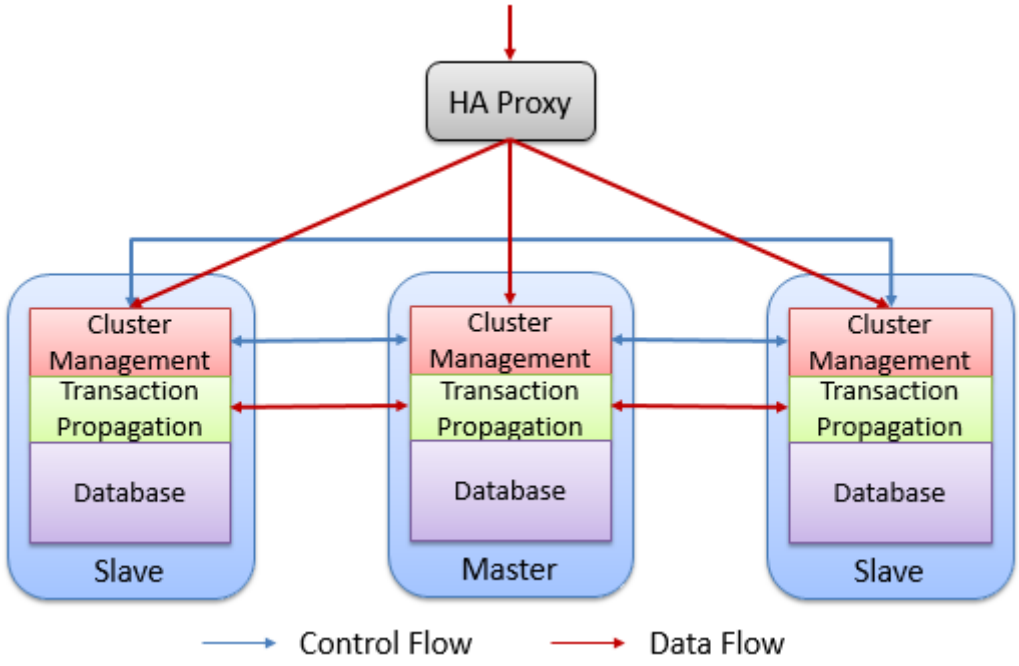
- graph-like queries, for example,
  - shortest path
  - graph's diameter
  - community detection

- Sample Systems
  - Neo4j、FlockDB、Infinite Graph、Titan、Microsoft Trinity、HyperGraphDB、AllegroGraph、Affinity 、OrientDB、DEX、Facebook Open Graph和Google Knowledge Graph

图存储、图数据库

- Neo4j：分布式图数据库（图计算平台）





分布式系统

- A.Tanenbaum: <coordination>
  - 分布式网络的计算机中的组件之间协调动作是通过消息进行通讯。
- G. Coulouris: <availability>
  - 当你知道有一台电脑崩溃，但是你的软件运行从来不会停止。
- Leslie Lamport: <share nothing + message passing>
  - 分布式系统是这样系统：旨在支持应用程序和服务的开发，可以利用物理架构由多个自治的处理元素，不共享主内存，但通过网络发送异步消息合作。

分布式系统

- 目标：扩展单机能力，提供透明、伸缩
  - 使资源可访问（协作，互斥）
  - 提供透明性（Transparent）
    - 访问，位置，迁移，重定位，复制，并发，故障
  - 开放性
    - 互操作，可移植，可扩展
  - 可伸缩性
    - 能够应对负载变化
    - 应对办法：异步通信、分布、缓存及复制

共性的问题

- 分布式节点组织和管理
  - 集中式、有中心（master-slave）、层次式
  - 无中心（Peer-to-Peer）：对等网络
- 任务的分配
  - 请求如何转交，请求状态的保持
- 资源映射
- 一致性
- …

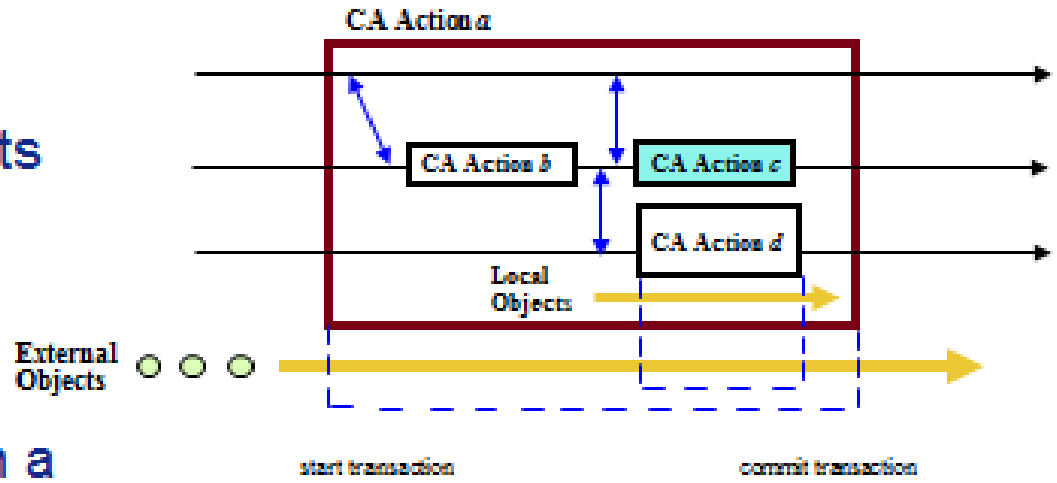
分布式系统的一致性

Definition: Distributed systems consistency is agreement or logical coherence amongst data or states of a system

There are various consistency models and issues in distributed computing

Three examples:

- Replicated data or objects
- Transactions – ACID implemented by locking
- Multi-party agreement on a joint decision or output, e.g.



顺序与一致性

- 一个例子

Initially, x = 5, y = 2

P1

1: x := x + y

2: z := x

P2

1: y := x - y

2: q := y

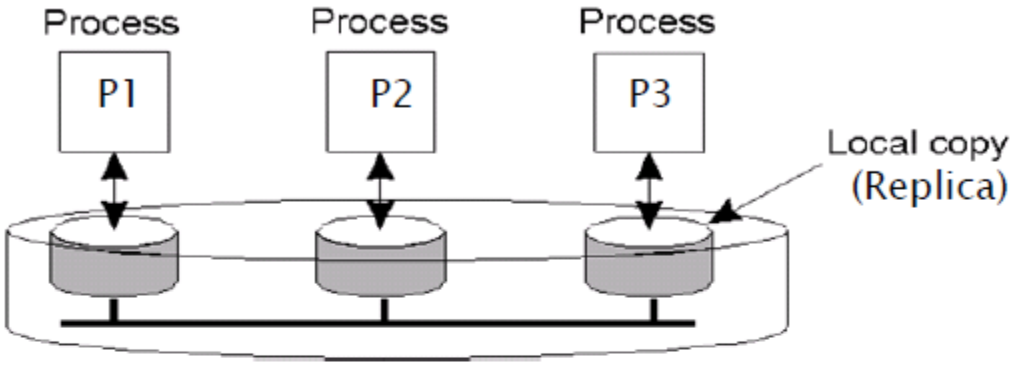
A. {P1(1) || P2(1)} ; {P1(2) || P2(2)} will produce x = z = 7, y = q = 3

B. {P1(1) ; P2(1)} ; {P1(2) || P2(2)} will produce results same as S1

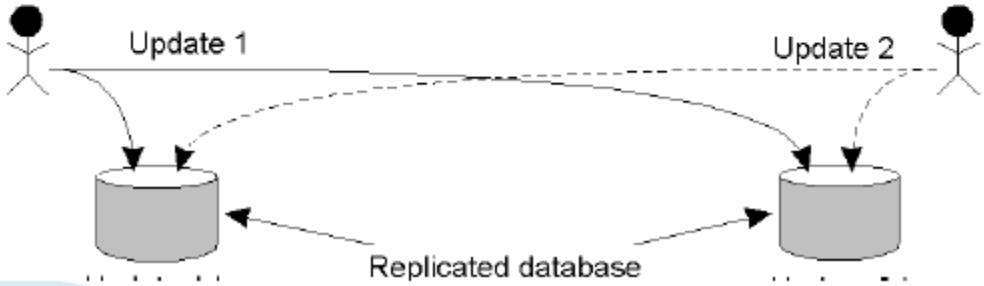
C. {P2(1) ; P1(1)} ; {P1(2) || P2(2)} will produce results same as S2

副本与一致性

- Replicas



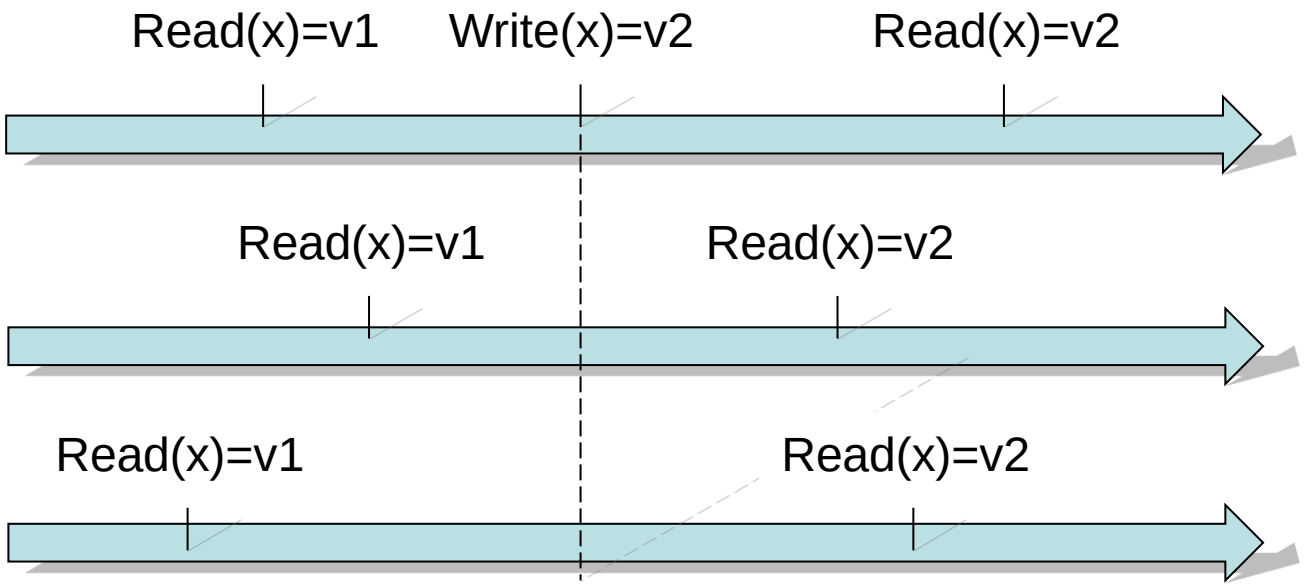
- Consistency Problem
  - Data-centric
  - Client-Centric



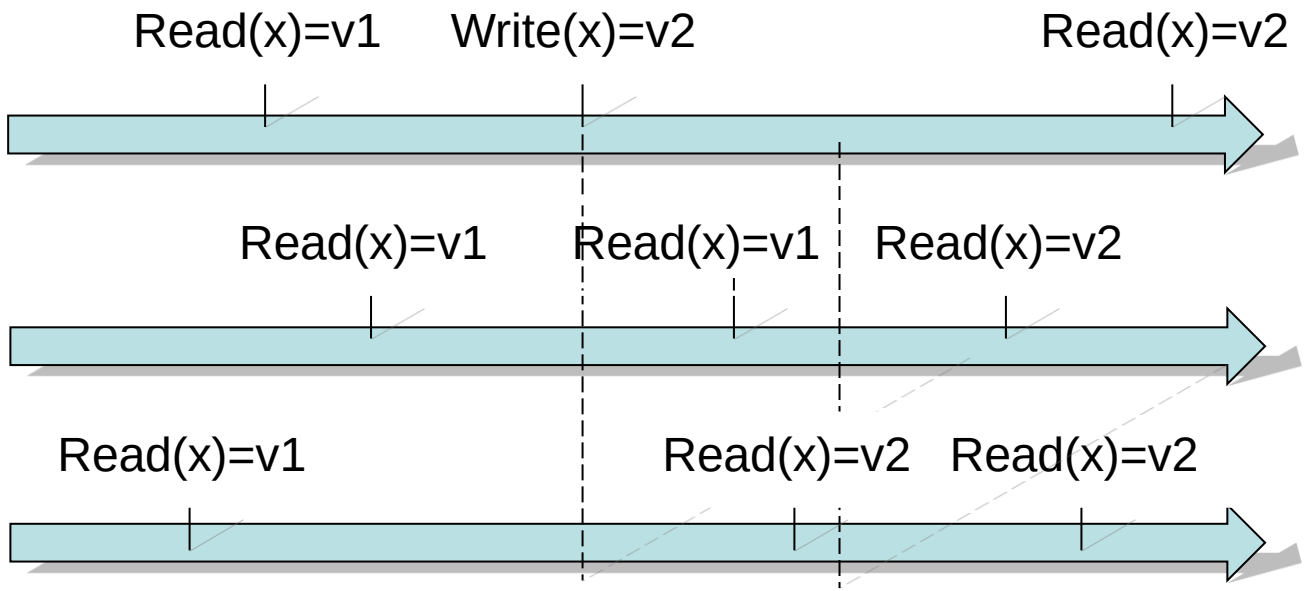
一致性模型

- 本质：程序员和系统（或存储数据的进程）之间的约定
  - 当满足约定，数据状态会保护一致，数据操作的效果是满足预期的
- 不同的一致性模型会有不同的一致性确保（guarantees）和限制（restrictions）

强一致性



弱一致性（最终一致性）

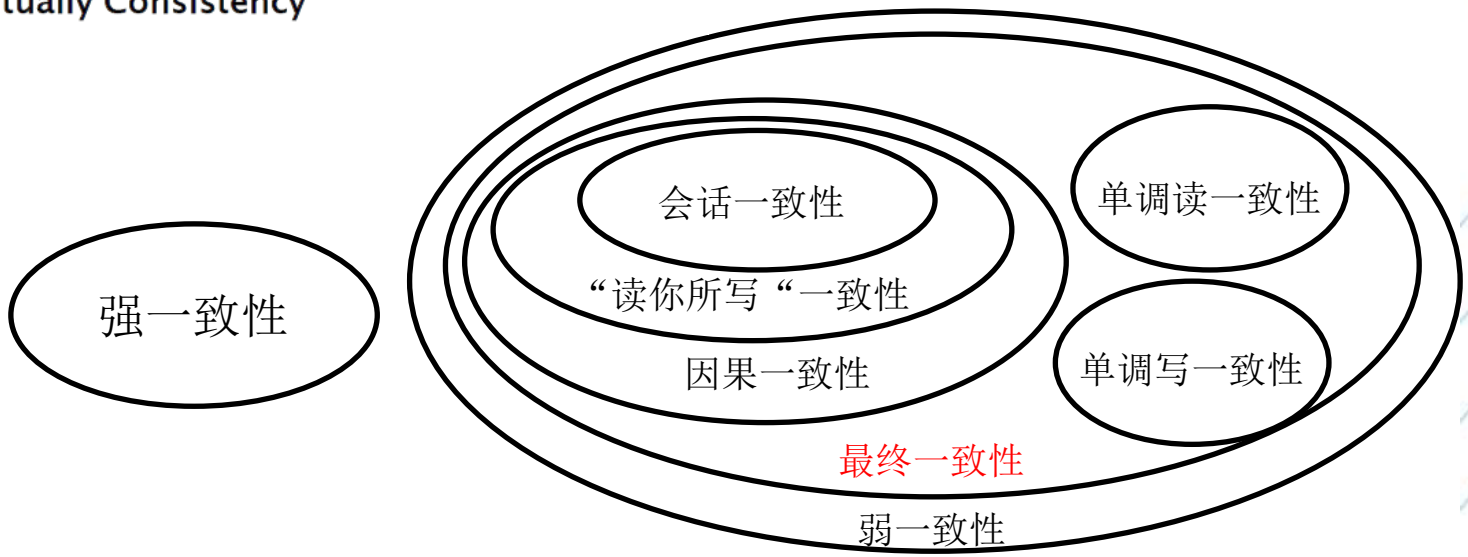


一致性模型

- Data-centric Consistency Models
  - Strict Consistency
  - Sequential Consistency
  - Causal Consistency
  - FIFO Consistency
  - Weak Consistency
  - ……
- Client-Centric Consistency models
  - Monotonic Read Consistency
  - Monotonic Write Consistency
  - Read-your-writes Consistency
  - Writes-follows-reads Consistency
  - Eventually Consistency

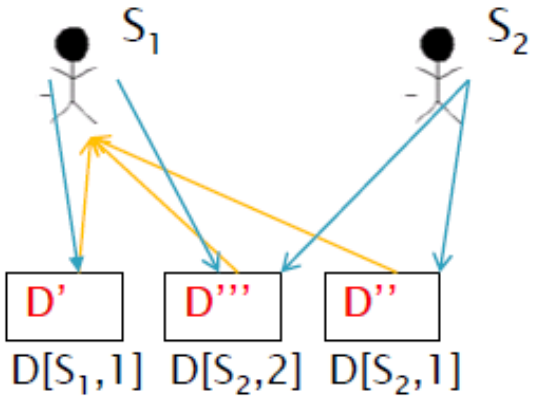
一致性模型

- Data-centric Consistency Models
  - Strict Consistency
  - Sequential Consistency
  - Causal Consistency
  - FIFO Consistency
  - Weak Consistency
  - ……
- Client-Centric Consistency models
  - Monotonic Read Consistency
  - Monotonic Write Consistency
  - Read-your-writes Consistency
  - Writes-follows-reads Consistency
  - Eventually Consistency



放松的一致性模型: R+W>N

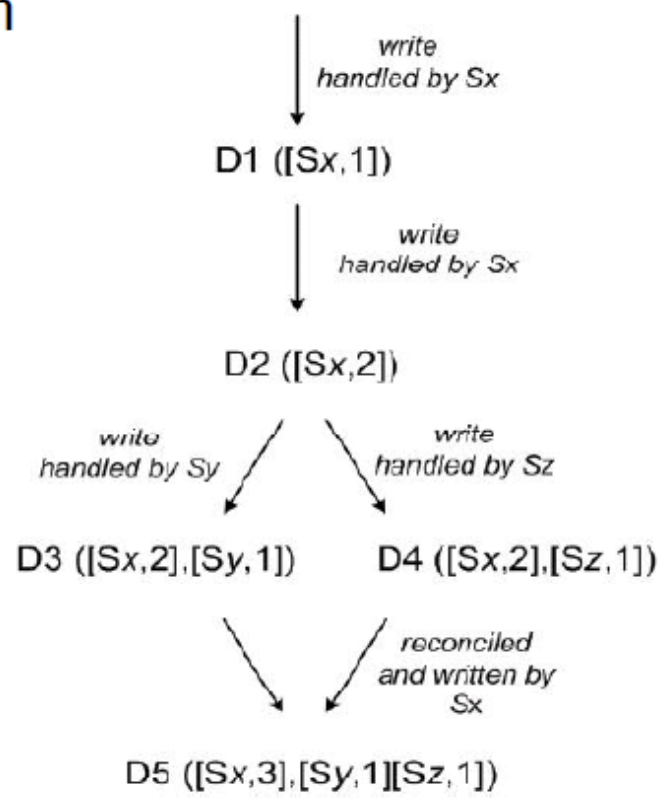
- 保持最终一致性的读写协议
  - "R + W > N" protocol, Such as
    - R=3, W=1, N=3
    - Updates to be propagated to all replicas asynchronously.



- 向量时钟标记数据版本
  - Problem: In {D', D'', D'''}, which is the freshest?
    - Data Versioning by vector clocks, i.e. (node, counter) pairs, Such as $[S_1,1]$, $[S_2,1]$, $[S_2,2]$
    - Propagated and Merge
      - $D[S_2,1]$ "<" $D[S_2,2]$, so D'' is replaced by D'''
      - How about D' and D'''?

放松的一致性模型: R+W>N

- 向量时钟，冲突交给应用层处理
  - version branching–the client must perform the reconciliation

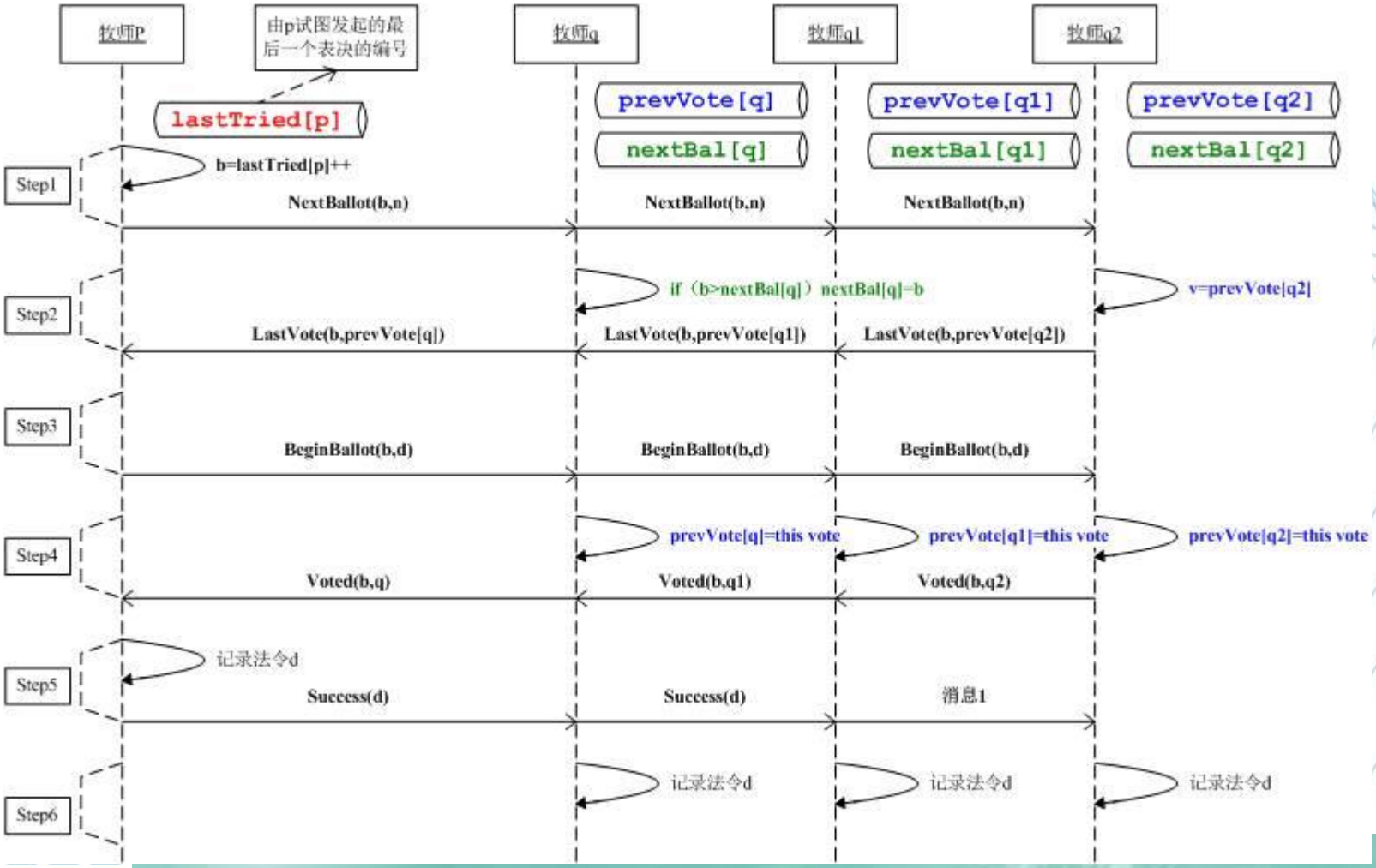


故障与容错

- Fail-Stop
  - 故障检测
  - 故障恢复：冗余
    - 计算中的双机热备
    - 存储中的主从、多副本
  - 避免：分布式一致性协议

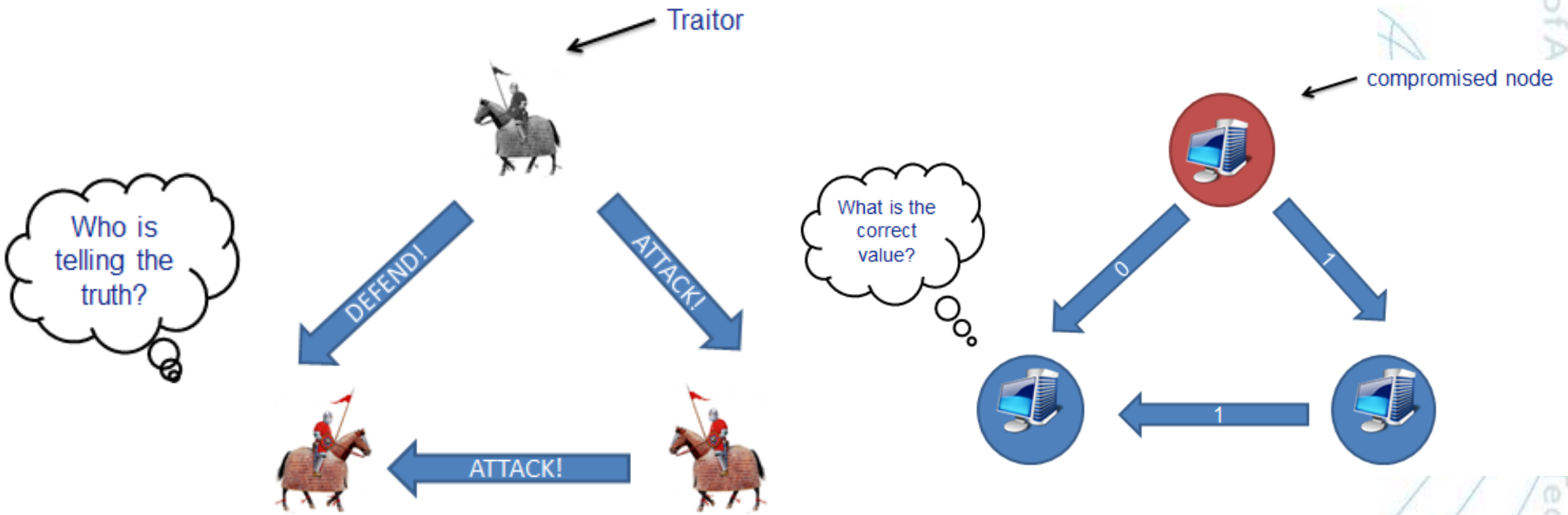
应对单一节点故障: Paxos协议

- 如何确保整个分布式服务器集群形成一致决议

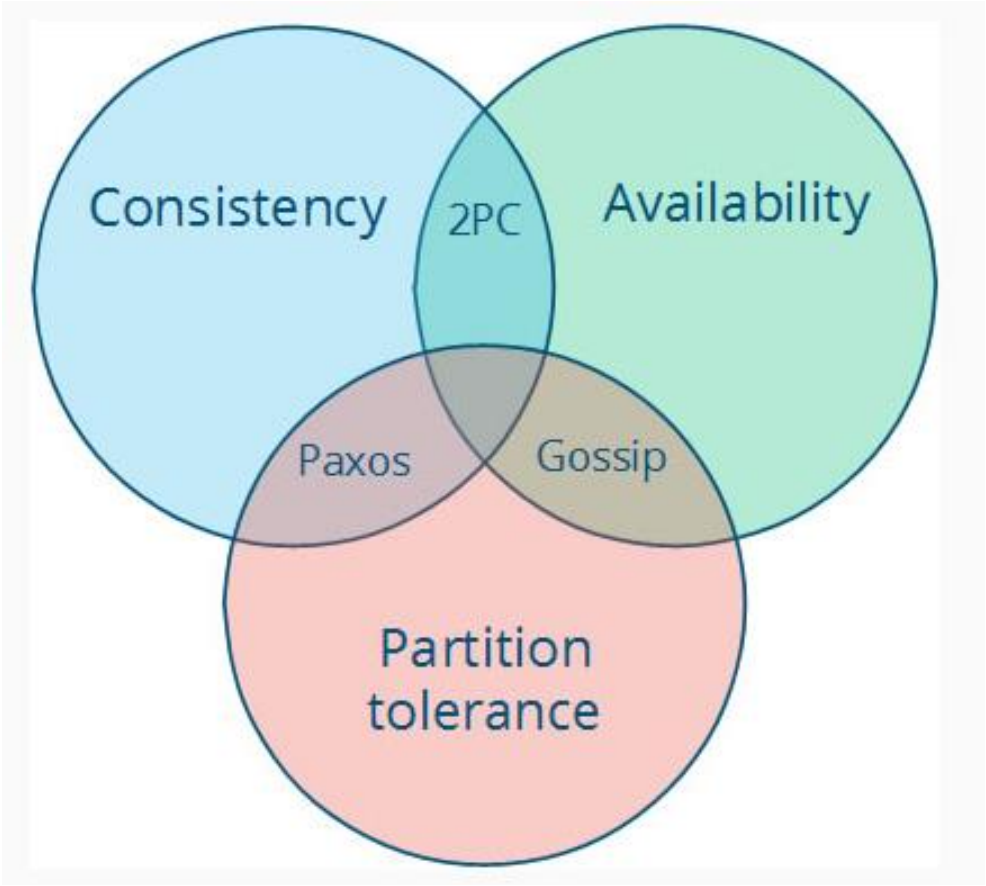


故障与容错

- Byzantine（Malfunction）



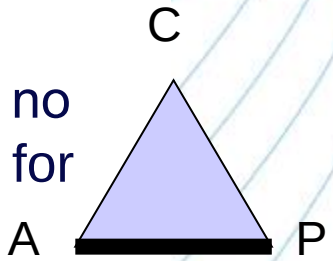
CAP理论



- 2000年，UCBerkeley 的 Eric Brewer提出猜想 —CAP

BASE

- BA: Basic Availability （基本可用）
  - Service availability is the most important attribute to ensure, e.g. a user is allowed to perform a transaction anytime and anywhere, while partition tolerance is supposed to be another basic requirement for any Cloud
- S: Soft State（软状态）
  - A transaction may take much longer time before it actually commits, e.g. a few seconds, thereby leaving the system in a temporary inconsistent or soft state
- E: Eventual Consistency (timed or delayed consistency)
  - The system will reach a consistent state “eventually”, but no guarantee on a specific time, e.g. in practice 3 – 4 hours for handling node crashes



小结

- 分布式存储系统
  - 数据抽象与访问方式
  - 数据存储介质管理
  - 元数据管理
- 不同的抽象方式
  - 文件、对象、关系、“简单”关系、图…
- 共性的部分
  - 性能优化、数据可靠性设计
  - 一致性

谢谢！

办公室：
新主楼G1119，电话：8233 9679
电子邮件：
hucm@buaa.edu.cn