大数据管理
Big Data Management

张海腾

htzhang@ecust.edu.cn

第五章 NOSQL数据库

- 5.1 NoSQL简介
- 5.2 NoSQL兴起的原因
- 5.3 NoSQL与关系数据库的比较
- 5.4 NoSQL的四大类型
- 5.5 NoSQL的三大基石
- 5.6 从NoSQL到NewSQL数据库
- 5.7 补充：键值数据库Redis
- 5.8 本章小结

5.1 NoSQL简介

传统的关系数据库

- 支持结构化数据存储和管理
- 以完善的关系代数理论作为基础
- 支持事务ACID四性
  - 原子性（Atomicity）
  - 一致性（Consistency）
  - 隔离性（Isolation）
  - 持久性（Durability）
- 借助于索引机制可以实现高效的查询

5.1 NoSQL简介

概念演变 ⟶ **Not only SQL**

最初表示“反SQL”运动
用新型的非关系数据库取代关系数据库

现在表示关系和非关系型数据库各有优缺点
彼此都无法互相取代

- **通常，NoSQL数据库具有以下几个特点：**

1、灵活的可扩展性

传统的关系型数据库由于自身设计机理的原因，在面对数据库负载大规模增加时，往往需要通过升级硬件来实现“纵向扩展”。NoSQL天生具备良好的水平扩展能力。

5.1 NoSQL简介

2、灵活的数据模型

关系数据库具有规范的定义，遵守各种严格的约束条件。这种做法虽然保证了业务系统对数据一致性的需求，但是过于死板的数据模型，也意味着无法满足各种新兴的业务需求。相反，NoSQL数据库采用键/值、列族等非关系模型，允许在一个数据元素里存储不同类型的数据。

3、与云计算紧密融合

云计算具有很好的水平扩展能力，可以根据资源使用情况进行自由伸缩，各种资源可以动态加人或退出，NoSQL数据库可以凭借自身良好的横向扩展能力，充分自由利用云计算基础设施，很好地融人到云计算环境中，构建基于NoSQL的云数据库服务。

5.1 NoSQL简介

- 现在已经有很多公司使用了NoSQL数据库：
  - Google
  - Facebook
  - Mozilla
  - Adobe
  - Foursquare-是一家基于用户地理位置信息（LBS）的手机服务网站
  - LinkedIn-领英（全球最大职业社交网站）
  - Digg-美国新闻聚合网站
  - McGraw-Hill Education
  - Vermont Public Radio
  - 百度、腾讯、阿里、新浪、美团、华为……

5.2 NoSQL兴起的原因

1、关系数据库已经无法满足Web2.0的需求，主要表现在以下几个方面：

（1）无法满足海量数据的管理需求(10亿条记录的查询效率低下）

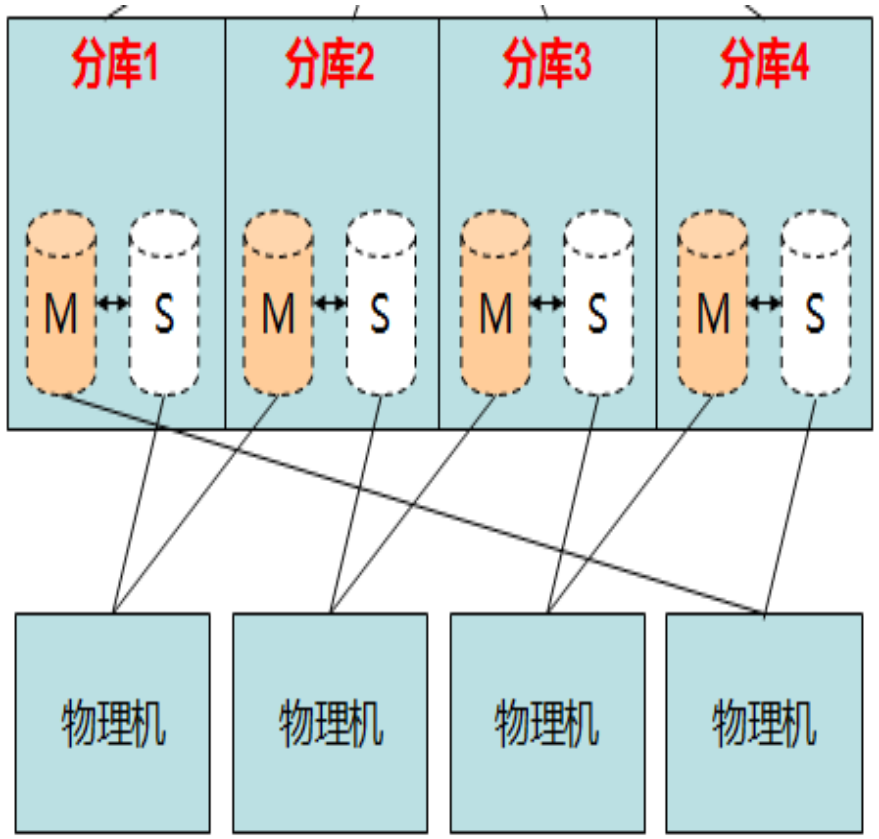
（2）无法满足数据高并发的需求（网站每秒上万次的请求）

（3）无法满足高可扩展性和高可用性的需求（短时间内迅速提升性能应对突发需求)

5.2 NoSQL兴起的原因

MySQL集群是否可以完全解决问题？

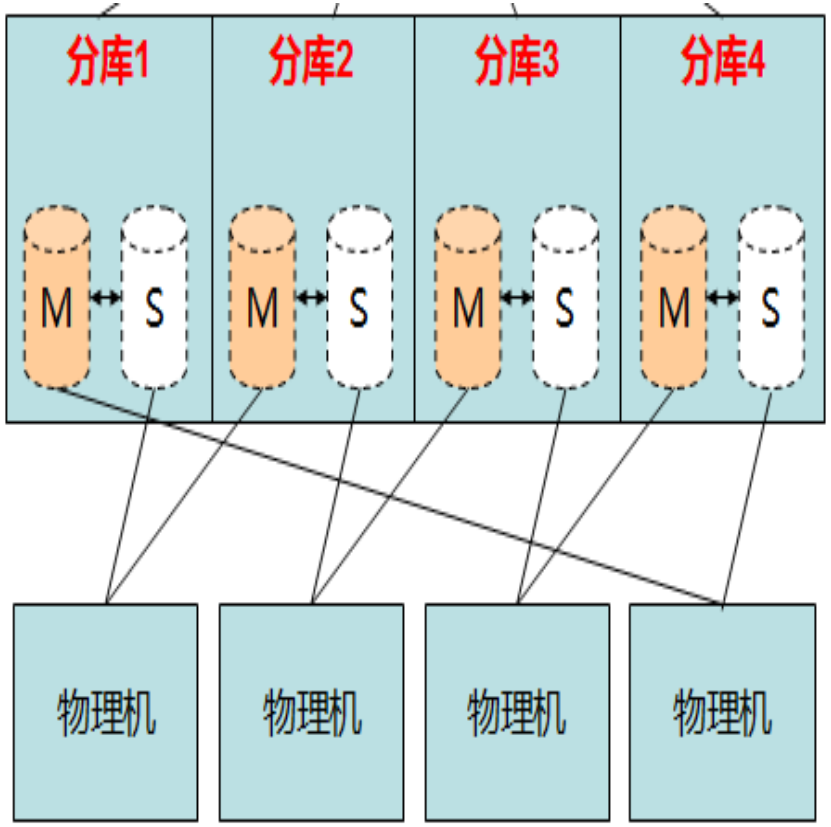
- **复杂性：** 部署、管理、配置很复杂
- **数据库复制：** MySQL主备之间采用复制方式，只能是异步复制，当主库压力较大时可能产生较大延迟，主备切换可能会丢失最后一部分更新事务，这时往往需要人工介入，备份和恢复不方便



5.2 NoSQL兴起的原因

◘MySQL集群是否可以完全解决问题？

- **扩容问题：**如果系统压力过大需要增加新的机器，这个过程涉及数据重新划分，整个过程比较复杂，且容易出错
- **动态数据迁移问题：**如果某个数据库组压力过大，需要将其中部分数据迁移出去，迁移过程需要总控节点整体协调，以及数据库节点的配合。这个过程很难做到自动化



5.2 NoSQL兴起的原因

2、“One size fits all”模式很难适用于截然不同的业务场景：

- 关系模型作为统一的数据模型既被用于数据分析，也被用于在线业务。但这两者一个强调高吞吐，一个强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的
- Hadoop针对数据分析(批处理)；MongoDB、Redis等是针对在线业务，两者都抛弃了关系模型

5.2 NoSQL兴起的原因

3、关系数据库的关键特性包括完善的事务机制和高效的查询机制。但是，这两个关键特性，到了Web2.0时代却成了鸡肋，主要表现在以下几个方面：

（1）Web2.0网站系统通常不要求严格的数据库事务（例如，微博网站，如果一个用户发布微博的过程中出现错误，可以直接丢弃该信息，而不像RDBMS那样执行复杂的回滚操作，这样并不会给用户造成什么损失。而且，数据库事务通常有一套复杂的实现机制来保证数据库一致性，需要大量系统开销。）

5.2 NoSQL兴起的原因

（2）Web2.0并不要求严格的读写实时性（对于RDBMS，一旦有一条数据成功插入数据库，就可以立即被查询到，如银行。但是，对于Web2.0来说，不需要这种实时读写的要求，例如：粉丝数。）

（3）Web2.0通常不包含大量复杂的SQL查询（Web2.0网站在设计时就已经尽量减少复杂的多表连接操作，只采用单表的主键查询。）

5.3 NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	• RDBMS有关系代数理论作为基础 • NoSQL没有统一的理论基础
数据规模	大	超大	• RDBMS很难实现横向扩展，纵向扩展的空间也比较有限，性能会随着数据规模的增大而降低 • NoSQL可以很容易通过添加更多设备来支持更大规模的数据
数据库模式	固定	灵活	• RDBMS需要定义数据库模式，严格遵守数据定义和相关约束条件 • NoSQL不存在数据库模式，可以自由灵活定义并存储各种不同类型的数据

5.3 NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
查询效率	快	可以实现高效的简单查询，但是不具备高度结构化查询等特性，复杂查询的性能不尽人意	• RDBMS借助于索引机制可以实现快速查询（包括记录查询和范围查询） • 很多NoSQL数据库没有面向复杂查询的索引，虽然NoSQL可以使用MapReduce来加速查询，但是，在复杂查询方面的性能仍然不如RDBMS
一致性	强一致性	弱一致性	• RDBMS严格遵守事务ACID模型，可以保证事务强一致性； • 很多NoSQL数据库放松了对事务ACID四性的要求，而是遵守BASE模型，只能保证最终一致性

5.3 NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
数据完整性	容易实现	很难实现	• 任何一个RDBMS都可以很容易实现数据完整性，比如通过主键或者非空约束来实现实体完整性，通过主键、外键来实现参照完整性，通过约束或者触发器来实现用户自定义完整性；但是，在NoSQL数据库却无法实现
扩展性	一般	好	• RDBMS很难实现横向扩展，纵向扩展的空间也比较有限 • NoSQL在设计之初就充分考虑了横向扩展的需求，可以很容易通过添加廉价设备实现扩展

5.3 NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
可用性	好	很好	• RDBMS在任何时候都以保证数据一致性为优先目标，其次才是优化系统性能，随着数据规模的增大，RDBMS为了保证严格的一致性，只能提供相对较弱的可用性；
标准化	是	否	• RDBMS已经标准化（SQL） • NoSQL 还没有行业标准，不同的NoSQL数据库都有自己的查询语言，很难规范应用程序接口； • StoneBraker认为：NoSQL缺乏统一查询语言，将会拖慢NoSQL发展

5.3 NoSQL与关系数据库的比较

比较标准	RDBMS	NoSQL	备注
技术支持	高	低	• RDBMS经过几十年的发展，已经非常成熟，Oracle等大型厂商都可以提供很好的技术支持 • NoSQL在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持
可维护性	复杂	复杂	• RDBMS需要专门的数据库管理员(DBA)维护 • NoSQL数据库虽然没有DBMS复杂，也难以维护

5.3 NoSQL与关系数据库的比较

□总结:

（1）关系数据库

- **优势：**以完善的关系代数理论作为基础，有严格的标准，支持事务ACID四性，借助索引机制可以实现高效的查询，技术成熟，有专业公司的技术支持
- **劣势：**可扩展性较差，无法较好支持海量数据存储，数据模型过于死板、无法较好支持Web2.0应用，事务机制影响了系统的整体性能等

5.3 NoSQL与关系数据库的比较

（2）NoSQL数据库

- **优势：**高并发读写、速度快（如HBase）；可以支持超大规模数据存储，灵活的数据模型可以很好地支持Web2.0应用，具有强大的横向扩展能力等
- **劣势：**缺乏数学理论基础，复杂查询性能不高，大都不能实现事务强一致性，很难实现数据完整性，技术尚不成熟，缺乏专业团队的技术支持，维护较困难等

5.3 NoSQL与关系数据库的比较

（3）关系数据库和NoSQL数据库各有优缺点，彼此无法取代

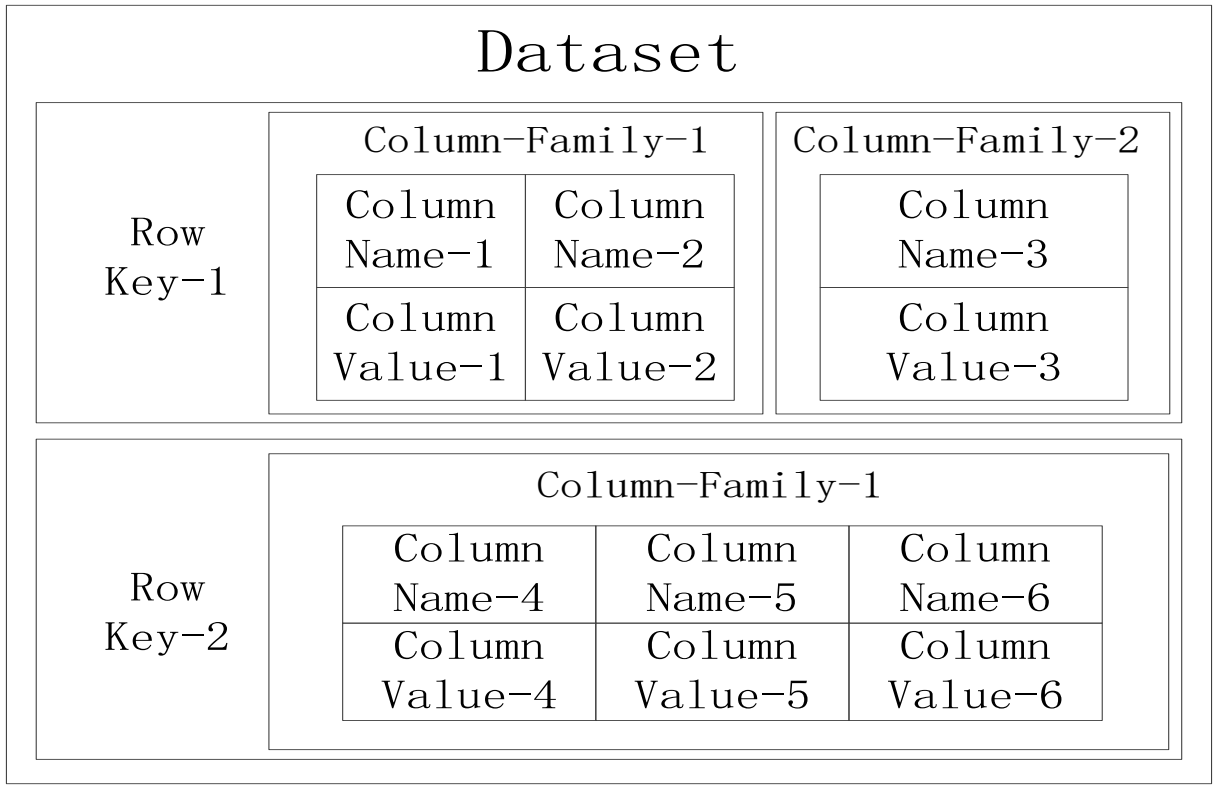
- **关系数据库应用场景：**电信、银行等领域的关键业务系统，需要保证强事务一致性
- **NoSQL数据库应用场景：**互联网企业、传统企业的非关键业务（如数据分析）
- **采用混合架构案例：**亚马逊公司就使用不同类型的数据库来支撑它的电子商务应用
  - 对于“购物篮”这种临时性数据，采用键值存储会更加高效
  - 当前的产品和订单信息则适合存放在关系数据库中
  - 大量的历史订单信息则适合保存在类似MongoDB的文档数据库中

5.4 NoSQL的四大类型

- NoSQL数据库虽然数量众多，但是，归结起来，典型的NoSQL数据库通常包括**键值数据库、列族数据库、文档数据库和图形数据库**

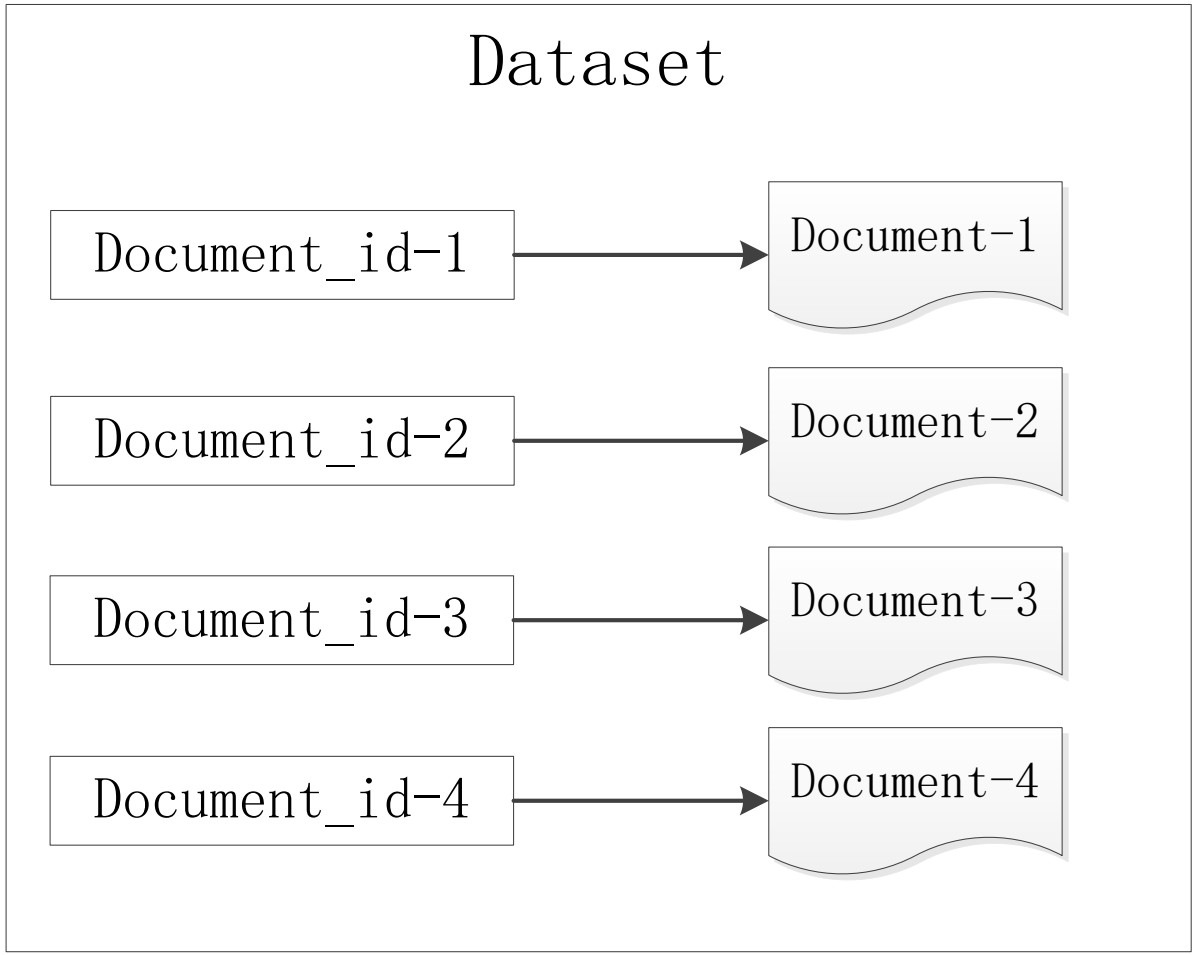
Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

键值数据库

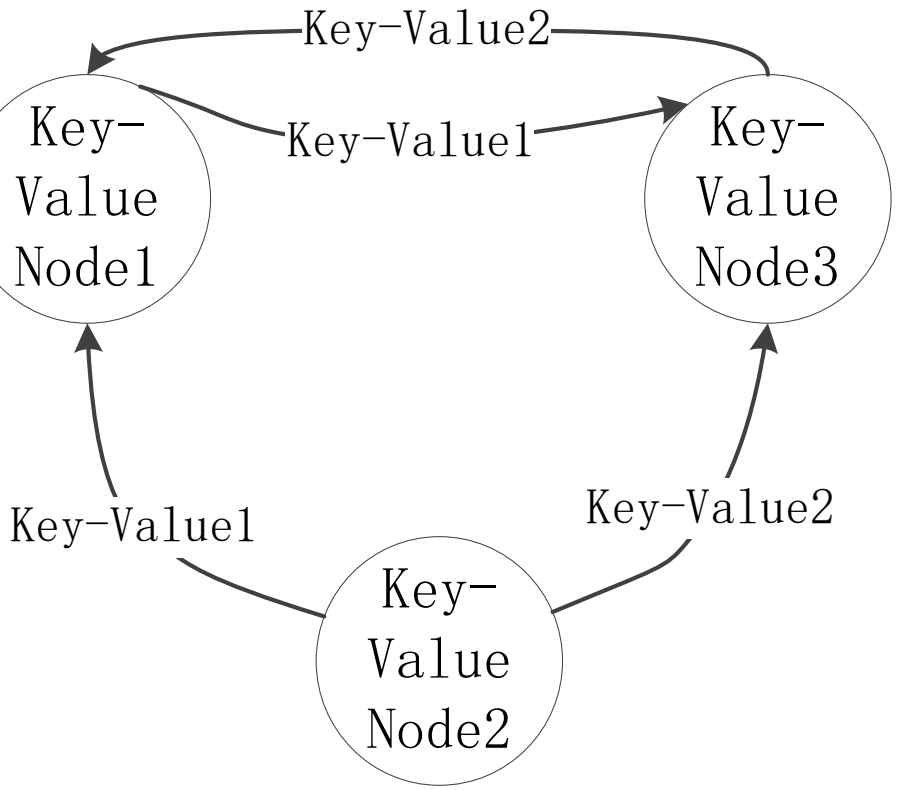


列族数据库

5.4 NoSQL的四大类型



文档数据库



图形数据库

5.4 NoSQL的四大类型

文档数据库	图数据库
 Couchbase MarkLogic mongoDB	Neo4j InfiniteGraph The Distributed Graph Database
键值数据库	**列族数据库**
redis amazon DynamoDB AEROSPIKE riak	accumulo HYPERTABLE INC Cassandra APACHE HBASE

5.4.1 键值数据库

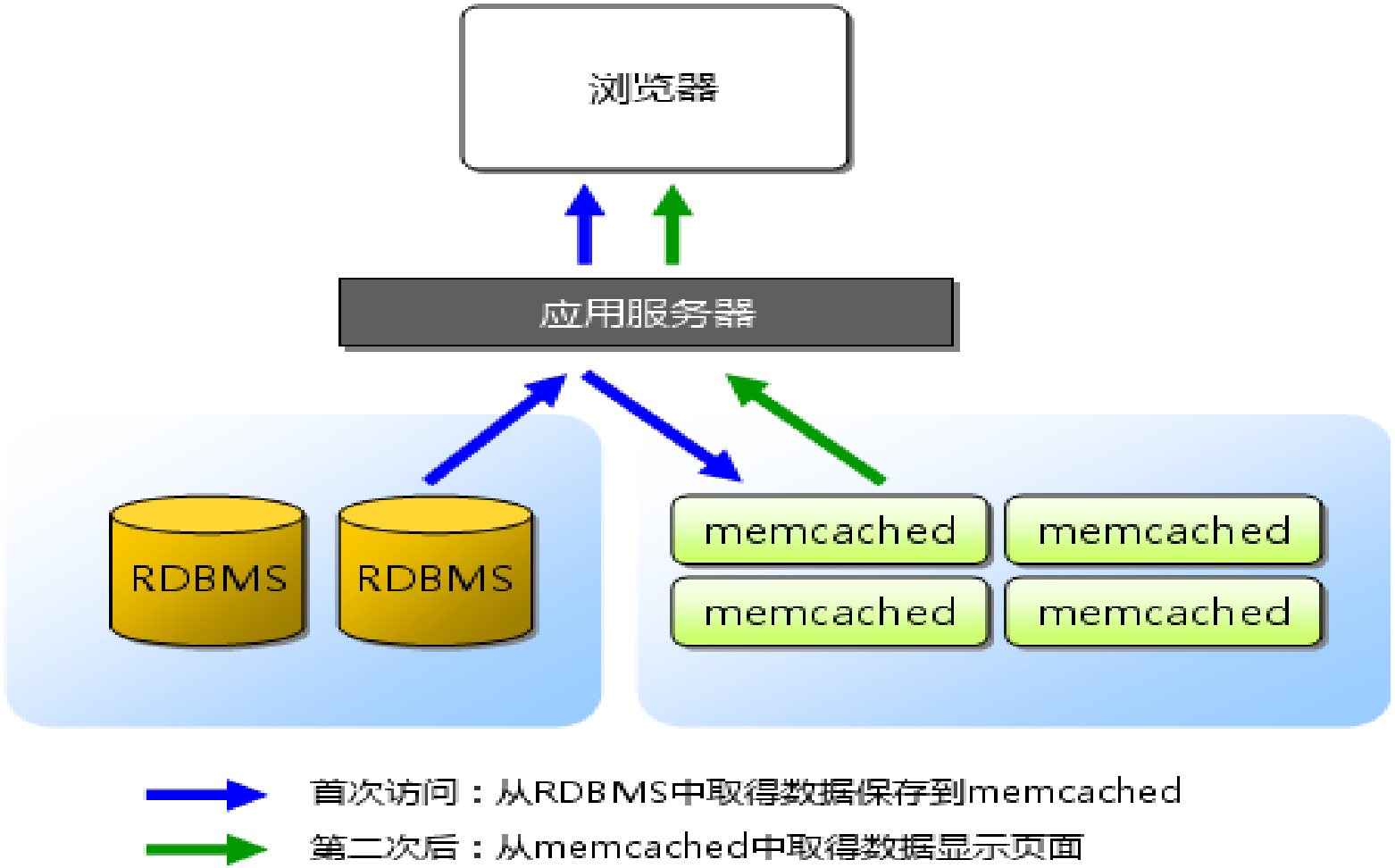
相关产品	Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
数据模型	• 键/值对 • 键是一个字符串对象 • 值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等
典型应用	• 涉及频繁读写、拥有简单数据模型的应用 • 内容缓存，比如会话、配置文件、参数、购物车等 • 存储配置和用户数据信息的移动应用
优点	扩展性好，灵活性好，大量写操作时性能高
缺点	无法存储结构化信息，条件查询效率较低

5.4.1 键值数据库

不适用情形	• 不是通过键而是通过值来查：键值数据库根本没有通过值查询的途径 • 需要存储数据之间的关系：在键值数据库中，不能通过两个或两个以上的键来关联数据 • 需要事务的支持：在一些键值数据库中，产生故障时，不可以回滚
使用者	百度云数据库（Redis）、GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram （Redis）、Youtube（Memcached）、Wikipedia（Memcached）

5.4.1 键值数据库

- 键值数据库成为**理想的缓冲层解决方案**
- Redis有时候会被人们称为“强化版的Memcached”支持**持久化、数据恢复、更多数据类型**



5.4.2 列族数据库

相关产品	BigTable、HBase、Cassandra、HadoopDB、GreenPlum、PNUTS
数据模型	列族
典型应用	• 分布式数据存储与管理 • 数据在地理上分布于多个数据中心的应用程序 • 可以容忍副本中存在短期不一致情况的应用程序 • 拥有动态字段的应用程序 • 拥有潜在大量数据的应用程序，大到几百TB的数据

5.4.2 列族数据库

优点	查找速度快，可扩展性强，容易进行分布式扩展，复杂性低
缺点	功能较少，大都不支持强事务一致性
不适用情形	需要ACID事务支持的情形，Cassandra等产品就不适用
使用者	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）

5.4.3 文档数据库

- “文档”其实是一个数据记录，这个记录能够对包含的数据类型和内容进行“自我描述”。
- XML文档、HTML文档和JSON 文档就属于这一类。
- SequoiaDB就是使用JSON格式的文档数据库，它的存储的数据是这样的：

```
{
“ID”:1,
“NAME”: “SequoiaDB”,
“Tel”: {
      “Office”: “123123”, “Mobile”: “132132132”
    }
“Addr”: “China,GZ”
}
```

5.4.3 文档数据库

- 数据是不规则的，每一条记录包含了所有的有关“SequoiaDB”的信息而没有任何外部的引用，这条记录就是“自包含”的
- 这使得记录很容易完全移动到其他服务器，因为这条记录的所有信息都包含在里面了，不需要考虑还有信息在别的表没有一起迁移走
- 同时，因为在移动过程中，只有被移动的那一条记录（文档）需要操作，而不像关系型中每个有关联的表都需要锁住来保证一致性，这样一来ACID的保证就会变得更快速，读写的速度也会有很大的提升

5.4.3 文档数据库

相关产品	MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
数据模型	键/值～值（value）是版本化的文档
典型应用	• 存储、索引并管理面向文档的数据或者类似的半结构化数据 • 比如，用于后台具有大量读写操作的网站、使用JSON数据结构的应用、使用嵌套结构等非规范化数据的应用程序

5.4.3 文档数据库

优点	性能好（高并发），灵活性高，复杂性低，数据结构灵活 提供嵌入式文档功能，将经常查询的数据存储在同一个文档中 既可以根据键来构建索引，也可以根据内容构建索引
缺点	缺乏统一的查询语法
不适用情形	在不同的文档上添加事务。文档数据库并不支持文档间的事务，如果对这方面有需求则不应该选用这个解决方案
使用者	百度云数据库（MongoDB）、SAP （MongoDB）、Codecademy （MongoDB）、Foursquare （MongoDB）、NBC News （RavenDB）

5.4.4 图数据库

相关产品	Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB
数据模型	图结构
典型应用	专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题
优点	灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱
缺点	复杂性高，只能支持一定的数据规模
使用者	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

5.4.5 不同类型数据库分析比较

中国菜刀

MySQL

瑞士军刀

MongoDB

大象兵

HBase

金箍棒

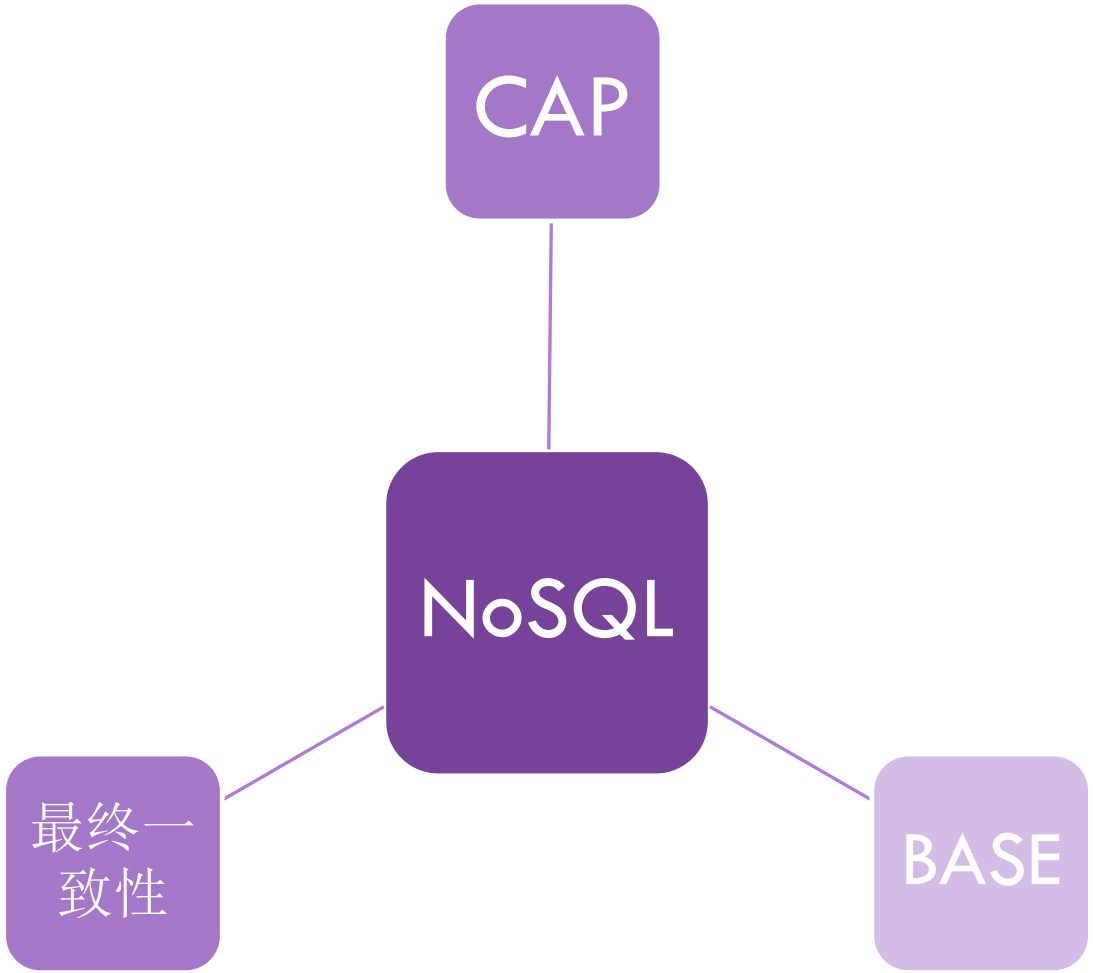
Redis

- **MySQL产生年代较早，而且随着LAMP大潮得以成熟。尽管其没有什么大的改进，但是新兴的互联网使用的最多的数据库**
- **MongoDB是个新生事物，提供更灵活的数据模型、异步提交、地理位置索引等五花八门的功能**

5.4.5 不同类型数据库分析比较

- **HBase是个“仗势欺人”的大象兵。依仗着Hadoop的生态环境，可以有很好的扩展性。但是就像大象兵一样，使用者需要养一头大象(Hadoop)，才能驱使它**
- **Redis是键值存储的代表，功能最简单。提供随机数据存储。就像一根棒子一样，没有多余的构造。但是也正是因此，它的伸缩性特别好。就像悟空手里的金箍棒，大可捅破天，小能缩成针**

5.5 NoSQL的三大基石

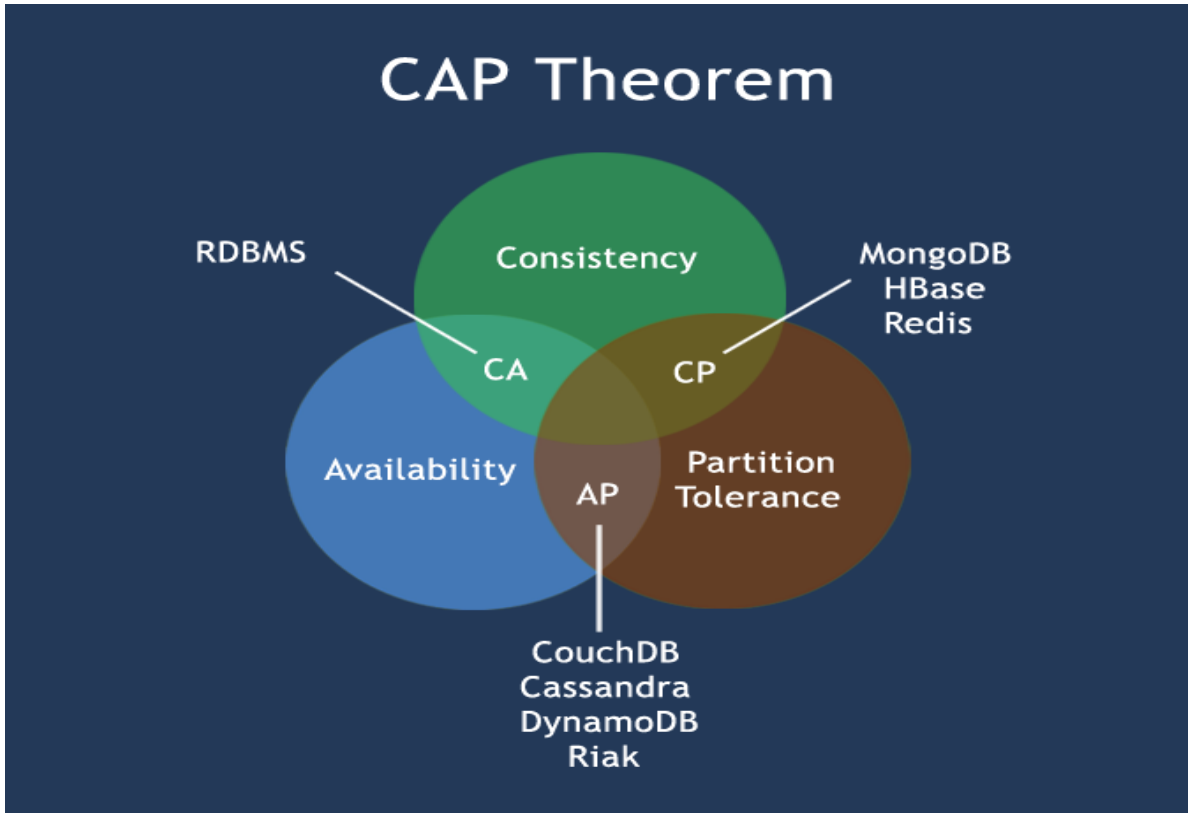


5.5.1 CAP

- **C（Consistency）：强一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多个节点的数据是一致的，或者说，所有节点在同一时间具有相同的数据**
- **A（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；**
- **P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。**

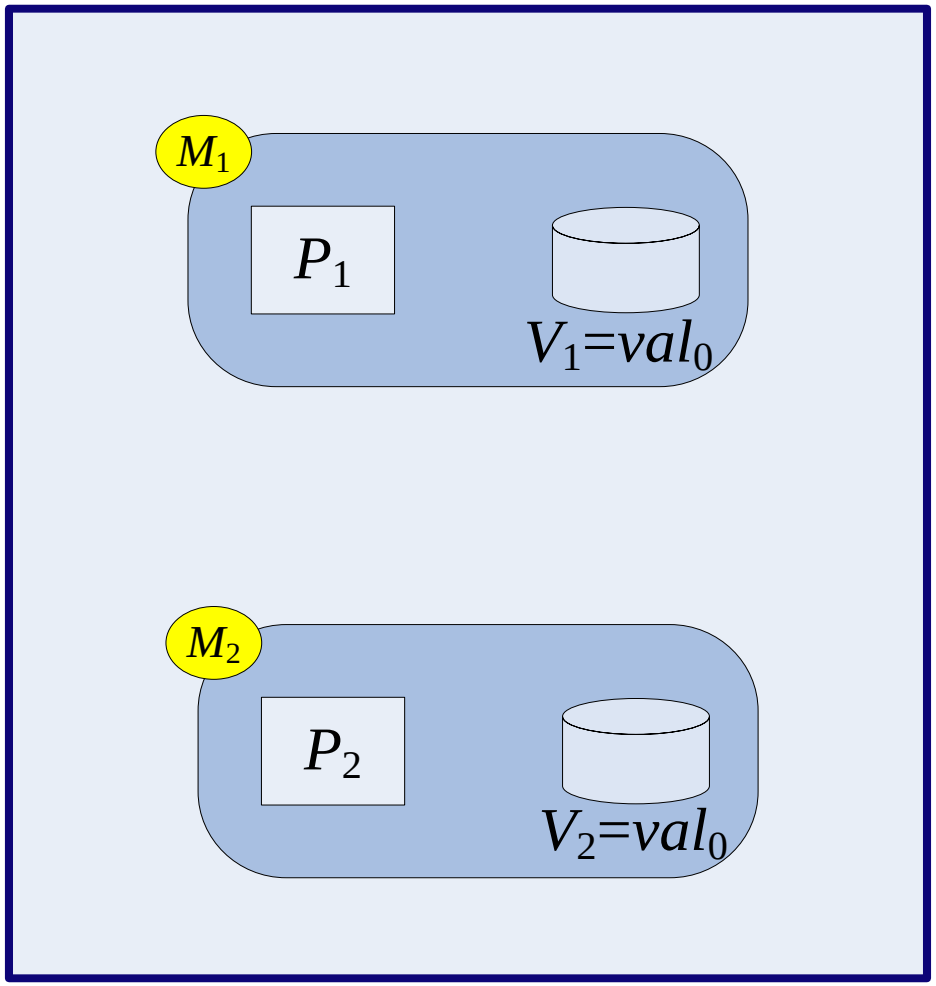
5.5.1 CAP

- CAP理论告诉我们，一个分布式系统不可能同时满足强一致性、可用性和分区容忍性（分区容错）这三个需求，最多只能同时满足其中两个，“鱼和熊掌不可兼得”。



5.5.1 CAP

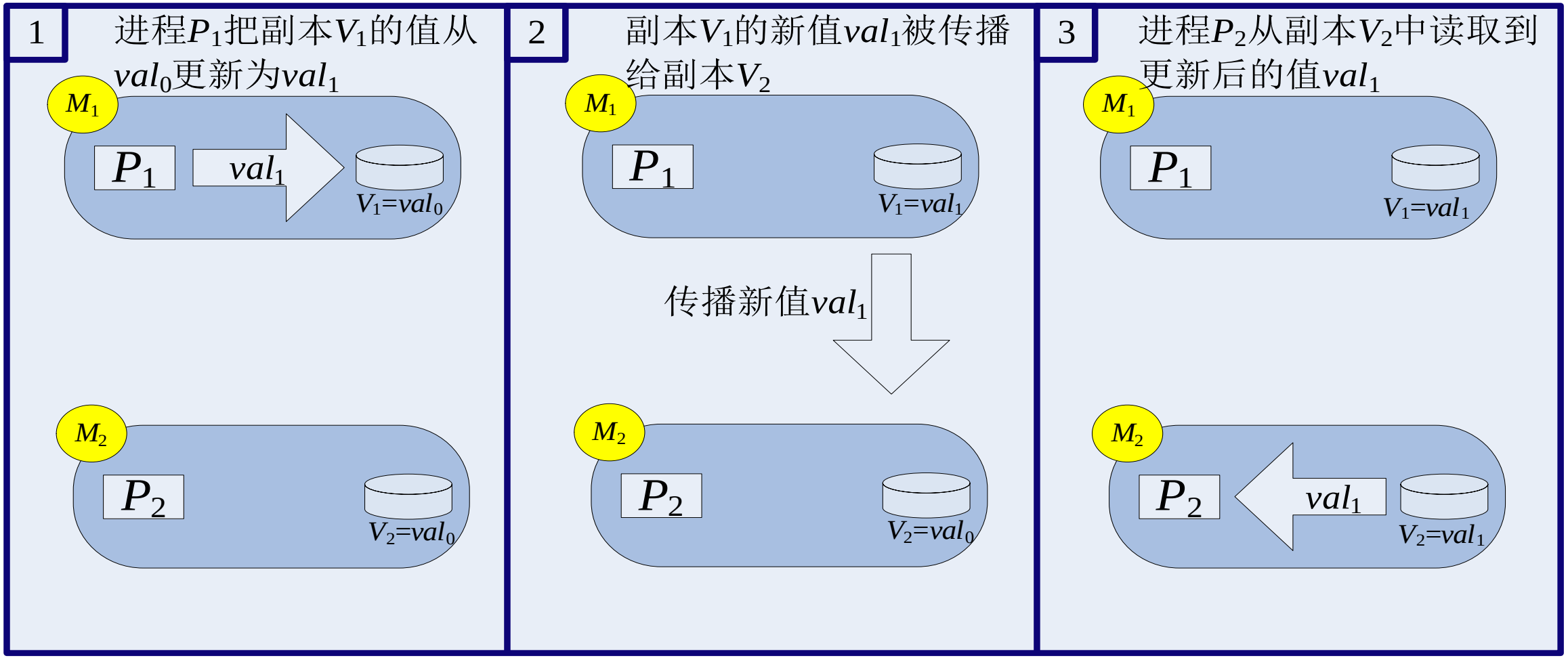
- 牺牲强一致性（C）换取可用性（A）的实例：



（a）初始状态

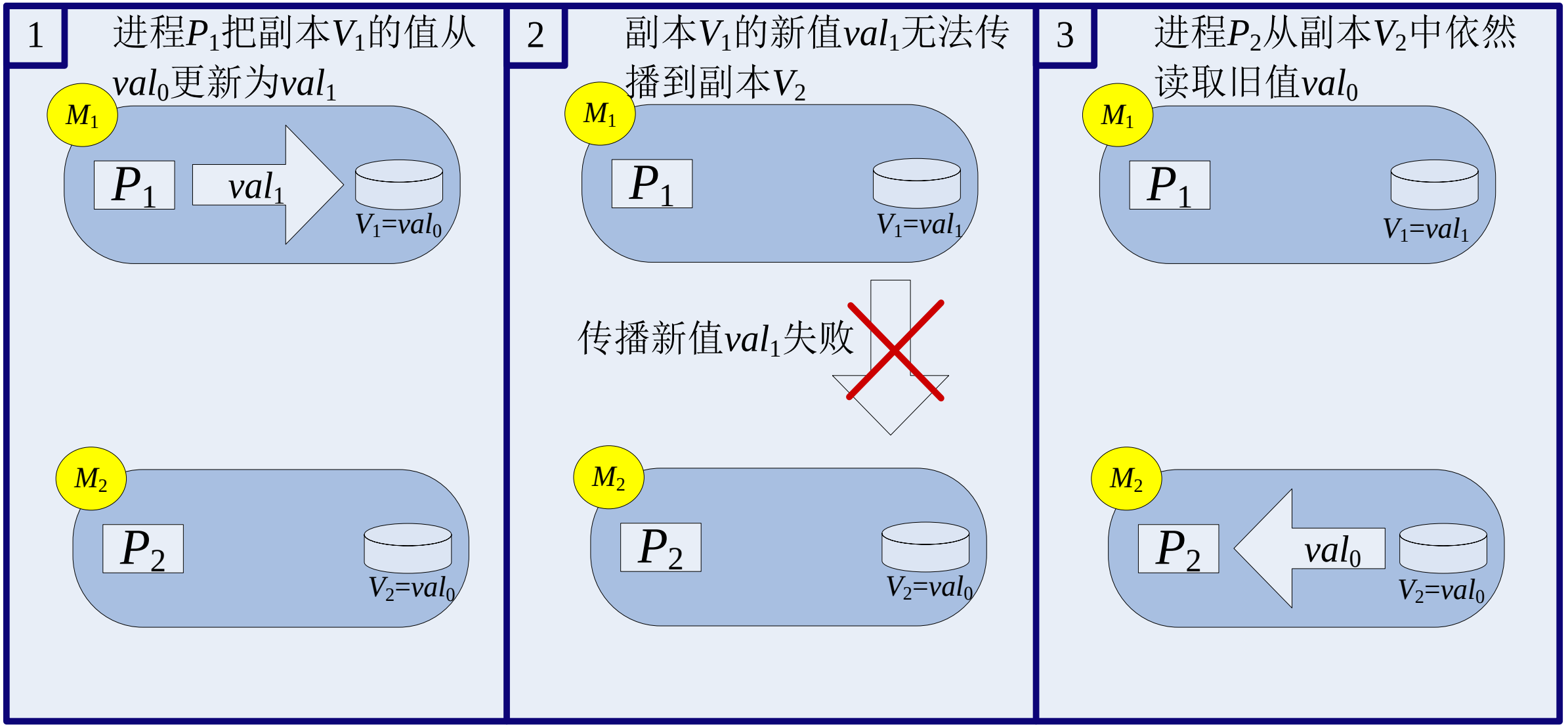
5.5.1 CAP

- 牺牲强一致性（C）换取可用性（A）的实例：



（b）正常执行过程

5.5.1 CAP



(c) 更新传播失败时的执行过程

5.5.1 CAP

当处理CAP的问题时，可以有几个明显的选择：

- **CA：也就是强调强一致性（C）和可用性（A），放弃分区容忍性（P），**最简单的做法是把所有与事务相关的内容都放到同一台机器上。很显然，这种做法会严重影响系统的可扩展性。传统的关系数据库（MySQL、SQL Server和PostgreSQL），都采用了这种设计原则，因此，扩展性都比较差
- **CP：也就是强调强一致性（C）和分区容忍性（P），放弃可用性（A），**当出现网络分区的情况时，受影响的服务需要等待数据一致，因此在等待期间就无法对外提供服务
- **AP：也就是强调可用性（A）和分区容忍性（P），放弃强一致性（C），**允许系统返回不一致的数据

5.5.1 CAP

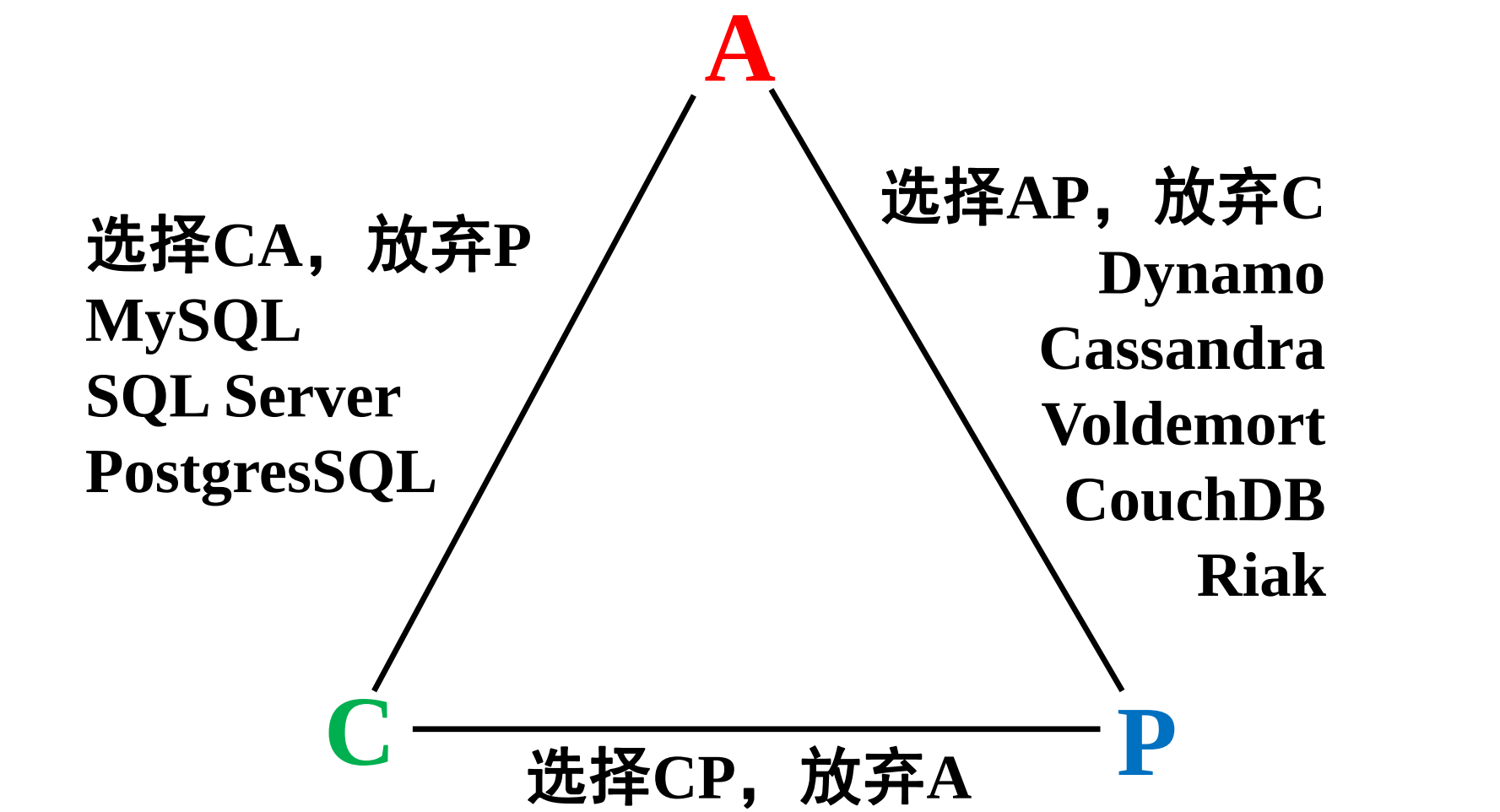


图5-4 不同产品在CAP理论下的不同设计原则

5.5.2 BASE

- BASE和ACID

ACID	BASE
原子性(Atomicity)	基本可用 (Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durability)	

5.5.2 BASE

□一个数据库事务具有ACID四性：

- A（Atomicity）：原子性，是指事务必须是原子工作单元，对于其数据修改，要么全都执行，要么全都不执行
- C（Consistency）：一致性，是指事务在完成时，必须使所有的数据都保持一致状态
- I（Isolation）：隔离性，是指由并发事务所做的修改必须与任何其它并发事务所做的修改隔离
- D（Durability）：持久性，是指事务完成之后，它对于系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持

5.5.2 BASE

- **BASE的基本含义是基本可用（Basically Availible）、软状态（Soft-state）和最终一致性（Eventual consistency）：**

- **基本可用：是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用，也就是允许分区失败的情形出现**
- **软状态：“软状态（Soft-state）”是与“硬状态（Hard-state）”相对应的一种提法。数据库保存的数据是“硬状态”时，可以保证数据一致性，即保证数据一直是正确的。“软状态”是指状态可以有一段时间不同步，具有一定的滞后性**

5.5.2 BASE

- **最终一致性：**一致性包括强一致性和弱一致性，二者的主要区别在于高并发的数据访问操作下，后续操作是否能够获取最新的数据。对于强一致性而言，当执行完一次更新操作后，后续的其他读操作就可以保证读到更新后的最新数据；反之，如果不能保证后续访问读到的都是更新后的最新数据，那么就是弱一致性。而最终一致性只不过是弱一致性的一种特例，允许后续的访问操作可以暂时读不到更新后的数据，但是经过一段时间之后，必须最终读到更新后的数据。

5.5.2 BASE

- 最常见的**实现最终一致性的系统是DNS（域名系统）**。一个域名更新操作根据配置的形式被分发出去，并结合有过期机制的缓存；最终所有的客户端可以看到最新的值。
- **说明：**BASE原则通过牺牲强一致性来获取高可用性；
- 根据目前的NoSQL发展来看，尽管绝大多数NoSQL系统采用了BASE原则，但是正在逐步提供局部的ACID的特性，**即全局保证BASE原则，局部支持ACID**，吸收二者好处，在两者之间建立平衡。

5.5.3 最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：

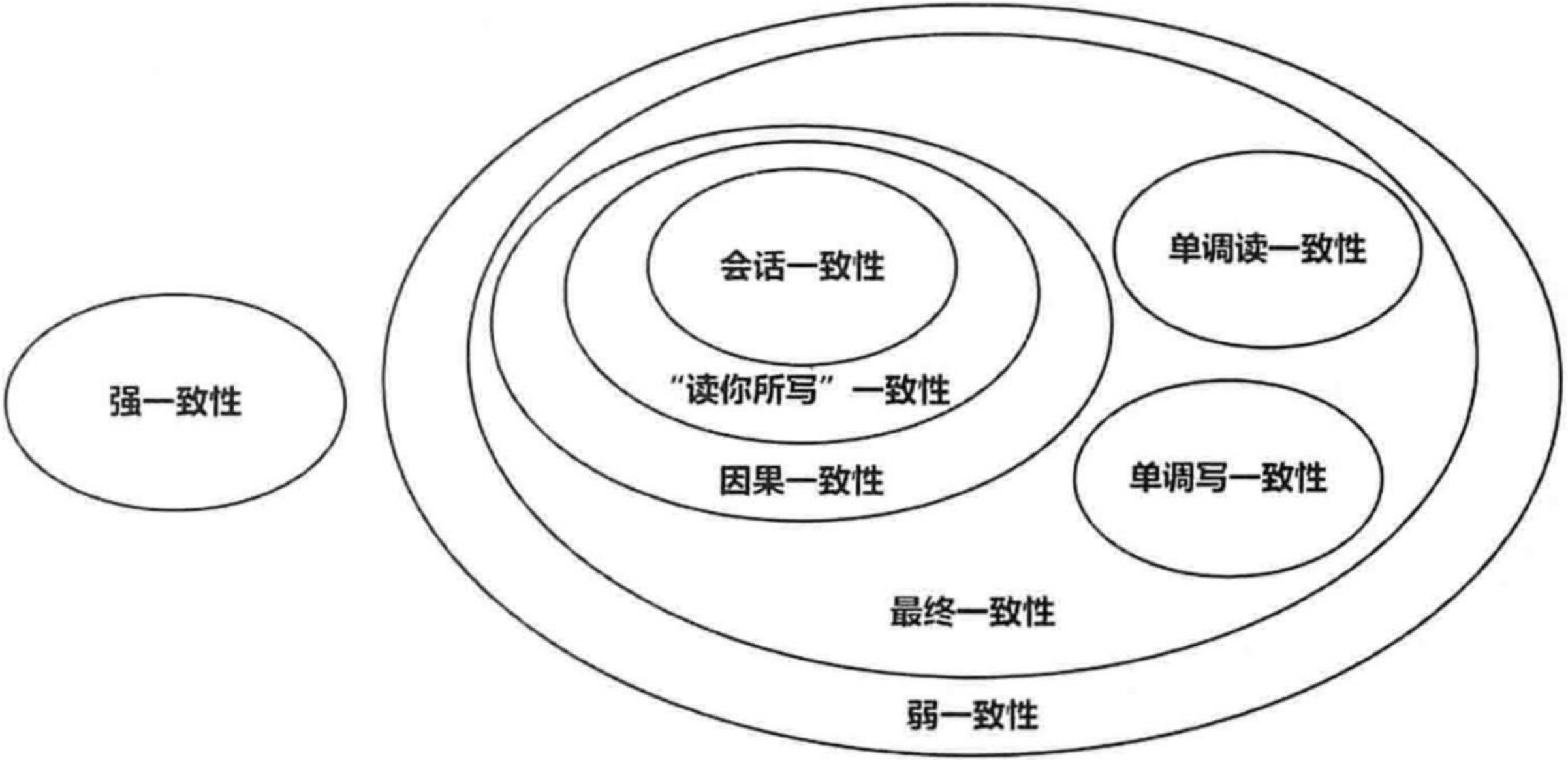
  - **因果一致性（Causal Consistency）：** 如果进程A通知进程B它已更新了一个数据项，那么进程B的后续访问将获得A写入的最新值。而与进程A无因果关系的进程C的访问，仍然遵守一般的最终一致性规则

  - **“读己之所写”（READ-YOUR-WRITES）一致性：** 可以视为因果一致性的一个特例。当进程A自己执行一个更新操作之后，它自己总是可以访问到更新过的值，绝不会看到旧值

5.5.3 最终一致性

- 最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：
  - **会话（Session）一致性：**因果一致性的一个特例。它把访问存储系统的进程放到会话（Session）的上下文中，只要会话还存在，系统就保证“读己之所写”一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统保证不会延续到新的会话
  - **单调（Monotonic）读一致性：**如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回在那个值之前的值
  - **单调写一致性：**系统保证来自同一个进程的写操作顺序执行。系统必须保证这种程度的一致性，否则就非常难以编程了

一致性模型关系图



5.5.3 最终一致性

□如何实现各种类型的一致性？

对于分布式数据系统：

N — 数据复制的份数

W — 更新数据时需要保证写完成的节点数

R — 读取数据的时候需要读取的节点数

- 如果W+R>N，写的节点和读的节点重叠，则是强一致性。例如对于典型的一主一备同步复制的关系型数据库，N=2,W=2,R=1，则不管读的是主库还是备库的数据，都是一致的。一般设定是W＋R = N+1，这是保证强一致性的最小设定

5.5.3 最终一致性

□如何实现各种类型的一致性？

对于分布式数据系统：

N — 数据复制的份数

W — 更新数据时需要保证写完成的节点数

R — 读取数据的时候需要读取的节点数

- 如果W+R<=N，则是弱一致性。例如对于一主一备异步复制的关系型数据库，N=2,W=1,R=1，则如果读的是备库，就可能无法读取主库已经更新过的数据，所以是弱一致性。

5.5.3 最终一致性

- 对于分布式系统，为了保证高可用性，**一般设置N>=3**。不同的N,W,R组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景。
- 如果N=W,R=1，任何一个写节点失效，都会导致写失败，因此可用性会降低，但是由于数据分布的N个节点是同步写入的，因此可以保证强一致性。
- 实例：HBase是借助其底层的HDFS来实现其数据冗余备份的。HDFS采用的就是强一致性保证。在数据没有完全同步到N个节点前，写操作是不会返回成功的。也就是说它的W＝N，而读操作只需要读到一个值即可，也就是说它R＝1。

5.6 从NoSQL到NewSQL数据库

- **NoSQL数据库的优点：**
  - 提供了良好的扩展性和灵活性
  - 较好地满足了Web2.0应用的需求
- **NoSQL数据库的不足之处：**
  - 采用非关系数据模型，不支持高度结构化查询
  - 查询效率低，不支持事务ACID四性
- **NewSQL数据库：各种新的可扩展、高性能数据库的简称**
  - 具有NoSQL对海量数据的存储管理能力
  - 保持了传统数据库支持ACID和sql等特性

5.6 从NoSQL到NewSQL数据库

- NewSQL数据库种类多，内部结构差异大,两个显著的共同特点：
  - 都支持关系数据模型
  - 都使用SQL作为主要的接口
- NewSQL数据库的代表：Google的Spanner
  - 可扩展、多版本、全球分布式并且支持同步复制的数据库
  - 无锁读事务
  - 原子模式修改
  - 读历史数据无阻塞

5.6 从NoSQL到NewSQL数据库

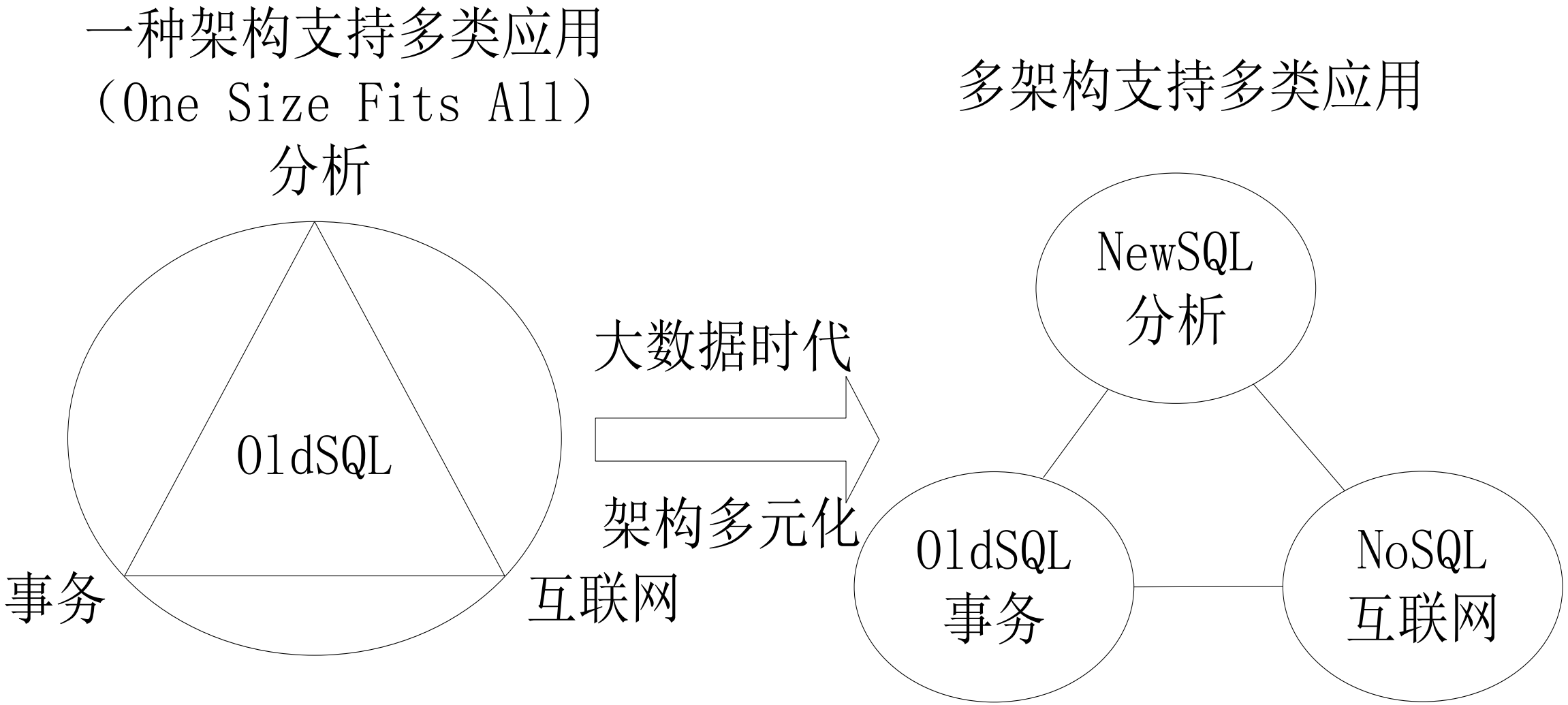


图5-5 大数据引发数据处理架构变革

5.6 从NoSQL到NewSQL数据库

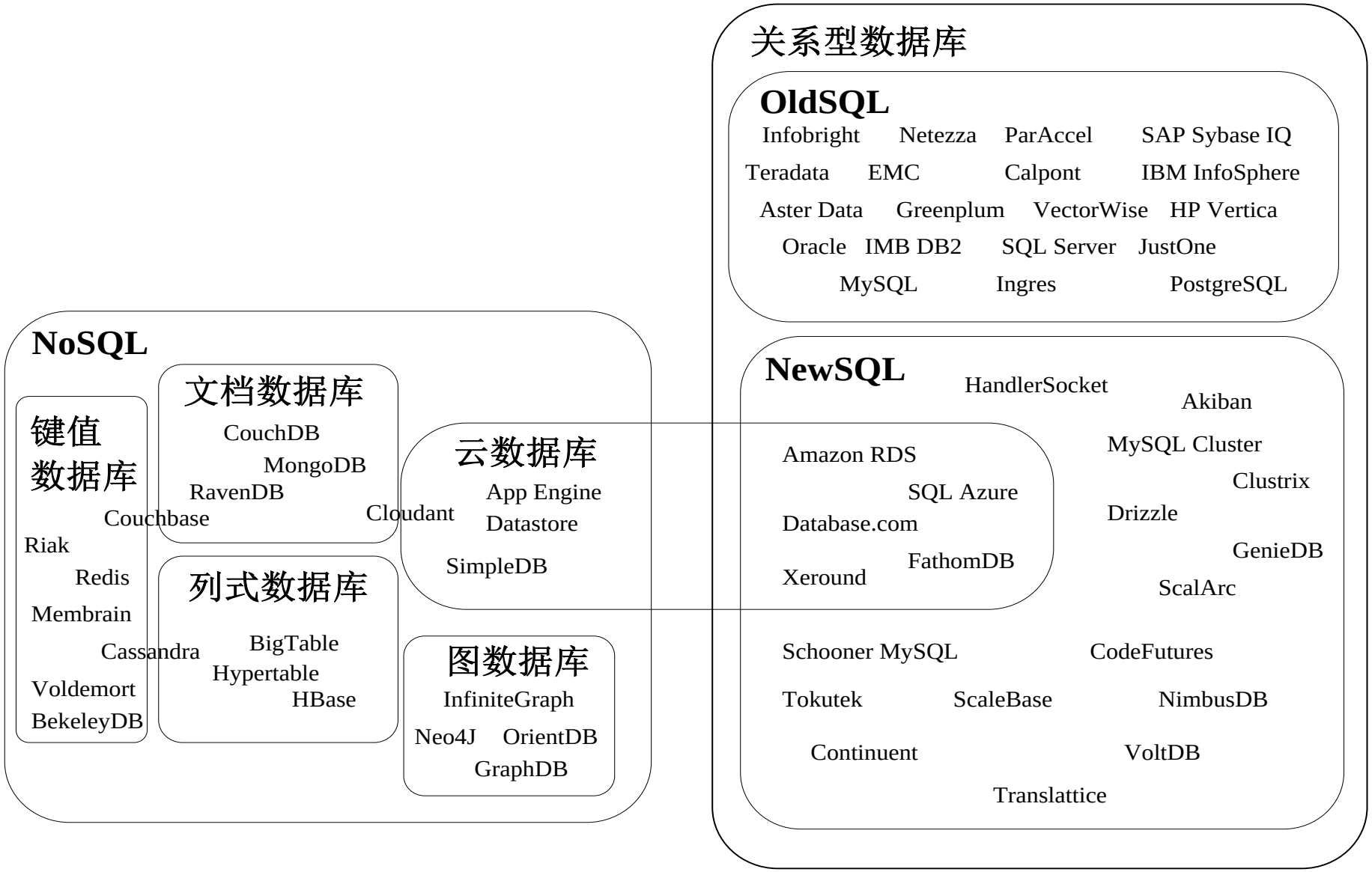


图5-6 关系数据库、NoSQL和NewSQL数据库产品分类图

5.7 键值数据库Redis

5.7.1 Redis的诞生

5.7.2 Redis简介

5.7.3 Redis安装和使用

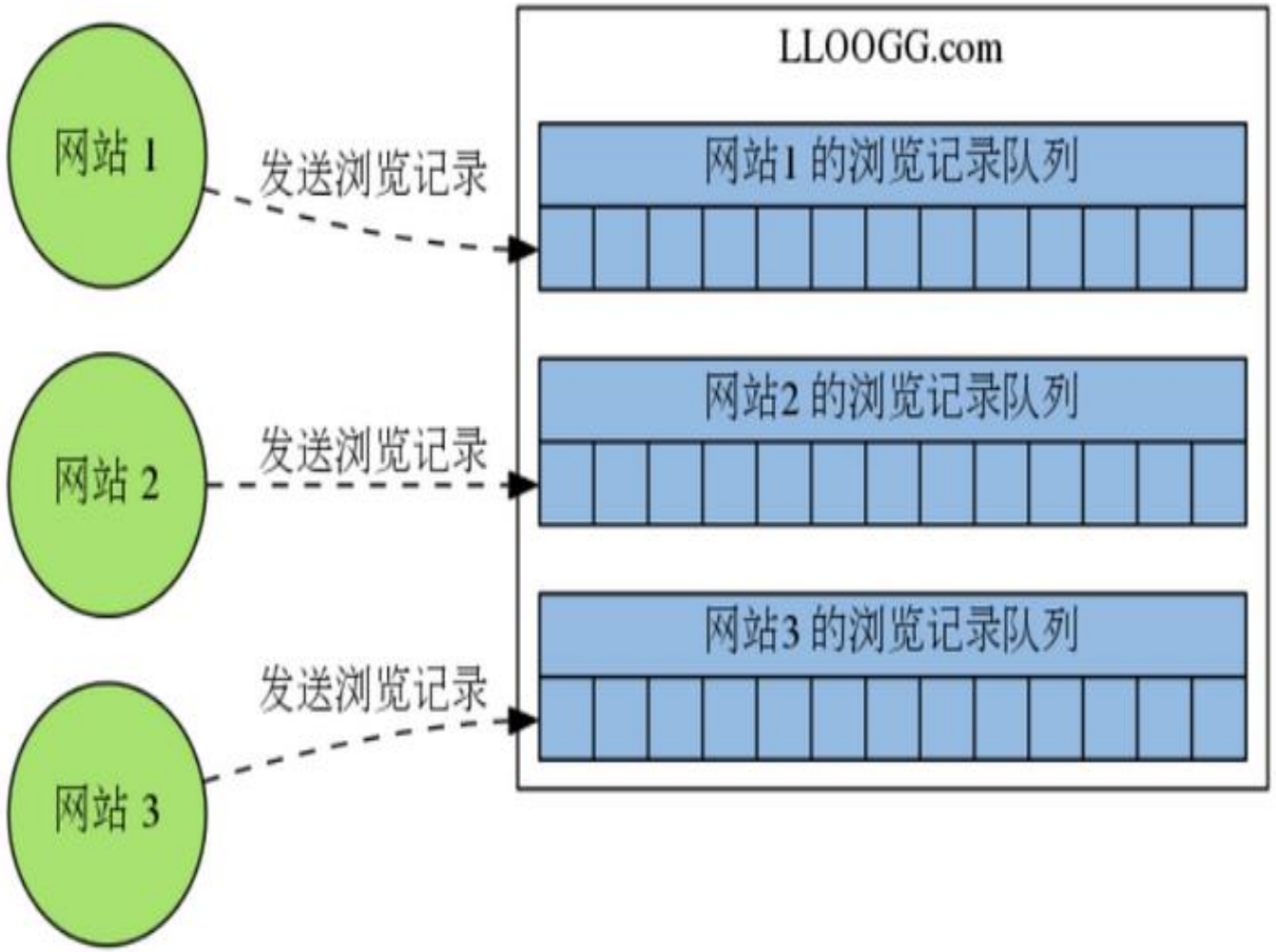
5.7.1 Redis的诞生

- **Redis之父：Salvatore Sanfilippo（antirez）**
  - 2007 年他和朋友共同创建了 LLOOGG. com，并因为解决这个网站的负载问题而在 2009 年发明了Redis
  - LLOOGG.com是一个访客信息追踪网站，网站可以通过 JavaScript 脚本，将访客的 IP 地址、所属国家、阅览器信息、被访问页面的地址等数据传送给 LLOOGG.com 。然后 LLOOGG.com 会将这些浏览数据通过 web 页面实时地展示给用户， 并储存起最新的 5 至 10,000 条浏览记录以便进行查阅。

5.7.1 Redis的诞生

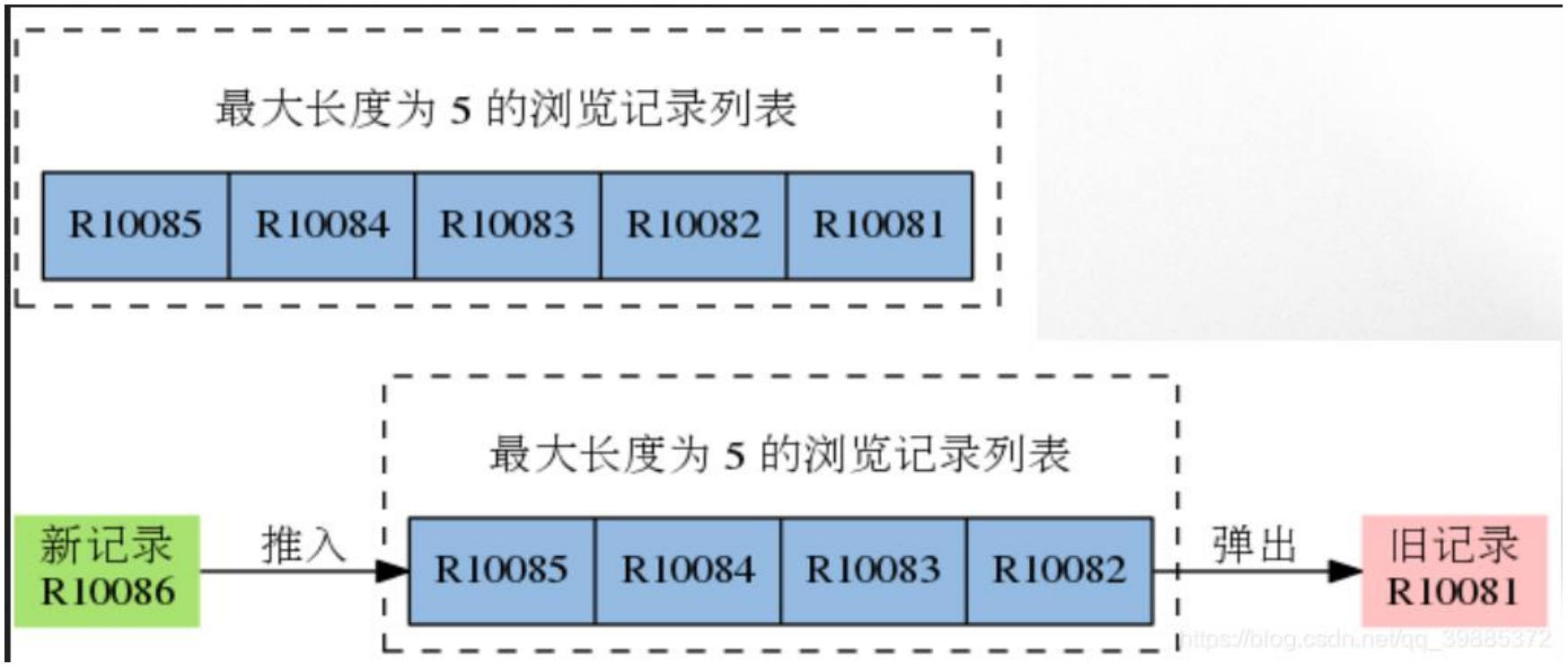
□LLOOGG.com 的运作方式

- 为了记录每个被追踪网站的浏览信息， LLOOGG.com 需要为每个被追踪的网站创建一个列表（list），每个列表需要根据用户的设置，储存最新的 5 至 10,000 条浏览记录。



5.7.1 Redis的诞生

- 每当某个被追踪的网站新增一条 浏览记录时， LLOOGG.com 就会将这条新的浏览记录推入（push）到与该网站相对应的列表里面，当列表的长度超过用户指定的最大长度时，程序每向列表推入一条新的记录，就需要从列表中弹出（pop）一条最旧的记录。



5.7.1 Redis的诞生

□LLOOGG.com 的负载问题

- 随着 LLOOGG.com 的用户越来越多， LLOOGG.com 要维护的列表数量也越来越多，要执行的推入和弹出操作也越来越多。
- LLOOGG.com 当时使用 MySQL 数据库，而 MySQL 每次执行推入和弹出操作都要进行硬盘写入和读取，程序的性能严重受制于硬盘 I/O 。 最终，LLOOGG.com 所使用的 MySQL 再也没办法在当时的 VPS上处理新增的大量负载，
- 因为 LLOOGG.com 当时还没有找到盈利模式，所以 为了尽量节约开支，antirez 没有选择直接升级 LLOOGG.com 所使用的 VPS ， 而是打算另寻办法，在现有硬件的基础上，通过提升列表操作的性能来解决 负载问题

5.7.1 Redis的诞生

□Redis 的诞生

- 为了在不升级 VPS 的前提下，解决 LLOOGG.com 的负载问题，antirez 决定自己写一个具有列表结构的内存数据库原型（prototype）。
- 这个数据库原型支持 O(1) 复杂的推入和弹出操作，并且将数据储存在内存而不是硬盘，所以程序的性能不会受到硬盘 I/O 限制，可以以极快的速度 执行针对列表的推入和弹出操作。
- 经过实验，这个原型的确可以在不升 级 VPS 的前提下，解决 LLOOGG.com 当时的负载问题。
- 于是 antirez 使用 C 语言重写了这个内存数据库，并给它加上了持久化功能， Redis 就此诞生！

5.7.2 Redis简介

- Redis(REmote DIctionary Server)即远程字典服务
- 基于键值对（key-value）非关系型的NoSQL内存数据库
- 提供了Python、Ruby、PHP客户端，使用很方便
- 支持存储的值（value）的数据结构：string、hash、list、set、zset（有序集合）、Bitmap、Geo、HyperLogLog
- 这些数据结构都支持push/pop、add/remove以及取交集、并集和差集等丰富的操作（原子性的操作）
- 支持各种不同方式的排序
- Redis中的数据都是缓存在内存中的，周期性把更新的数据写入磁盘
- 实现主从（master-slave）同步

5.7.2 Redis简介

□为什么要使用Redis数据库

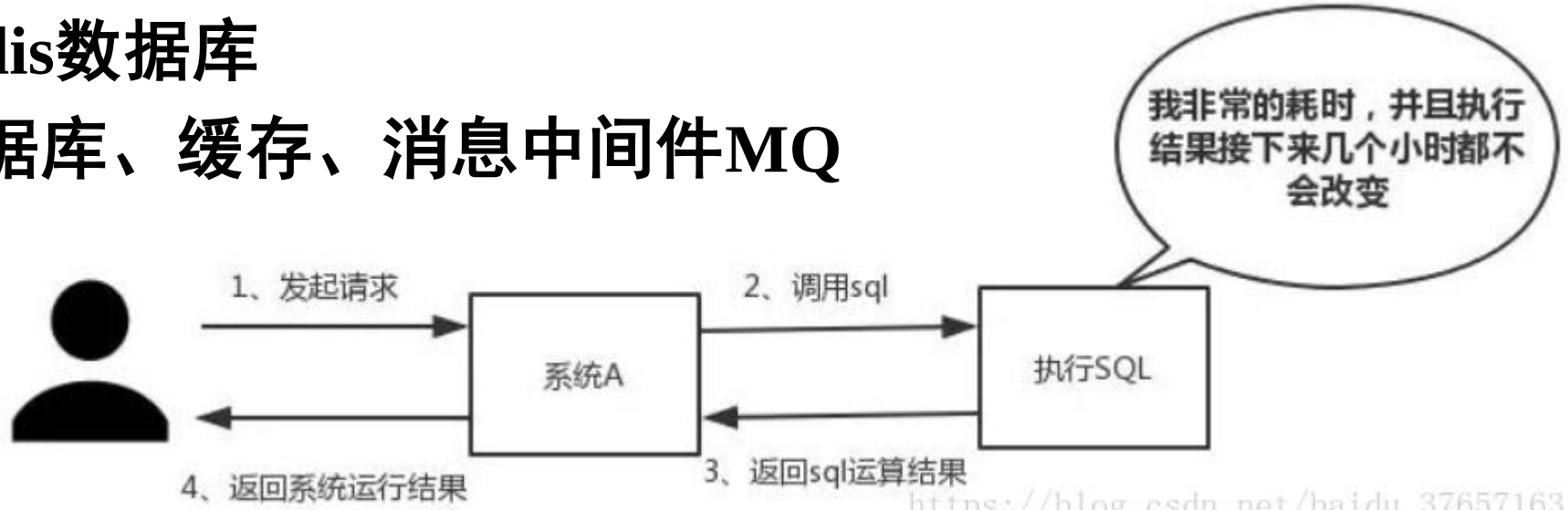
➢80%的系统瓶颈主要出现在数据库一侧

• 海量并发下，网络、磁盘I/O开销会导致数据库性能出现瓶颈

• 海量数据下，数据查询可能需要关联上千张表、遍历数千万的数据，要花费几分钟

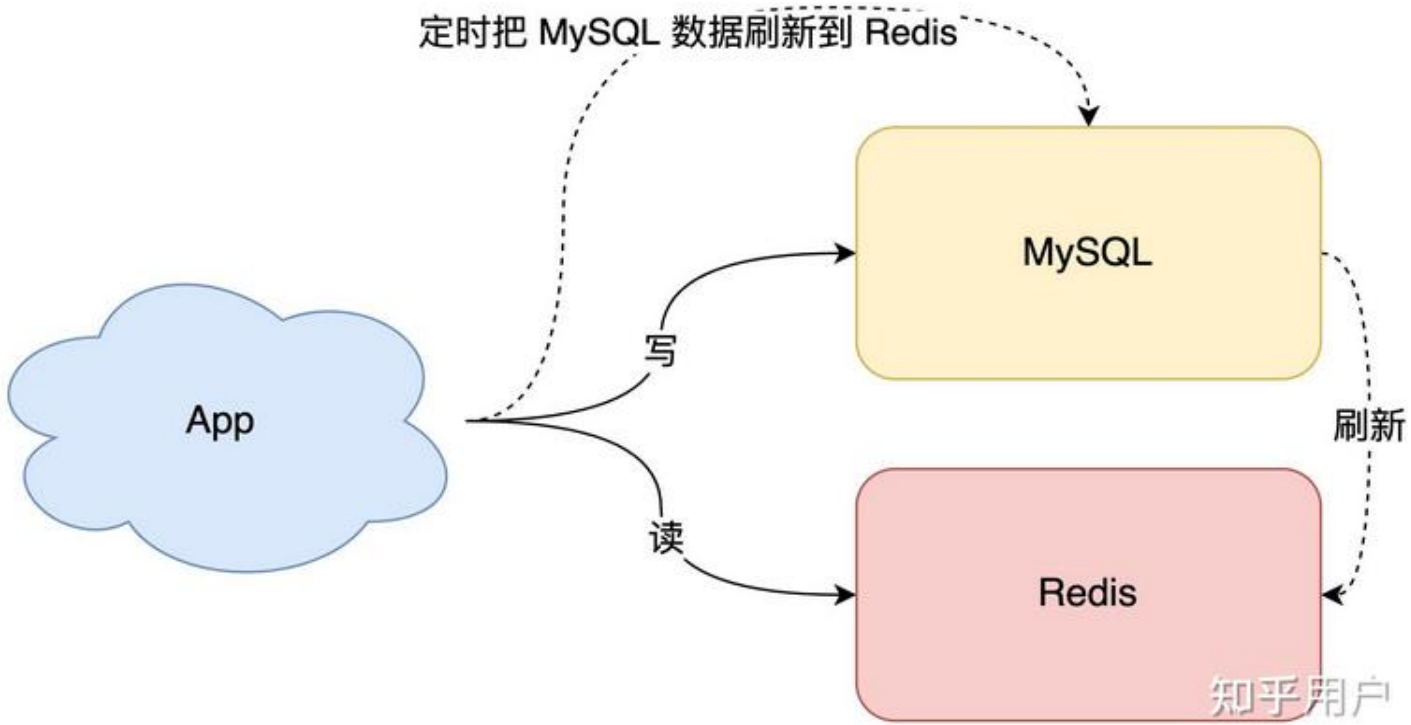
➢为了减少数据库压力，提高访问速度，需要用到读写速度更快的高性能缓存框架，如Redis数据库

➢Redis可以用作数据库、缓存、消息中间件MQ



5.7.2 Redis简介

（1）性能：在碰到需要执行耗时特别久，且结果不频繁变动的SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。



（2）并发：在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用Redis做一个缓冲操作，让请求先访问到Redis，而不是直接访问数据库。

5.7.2 Redis简介

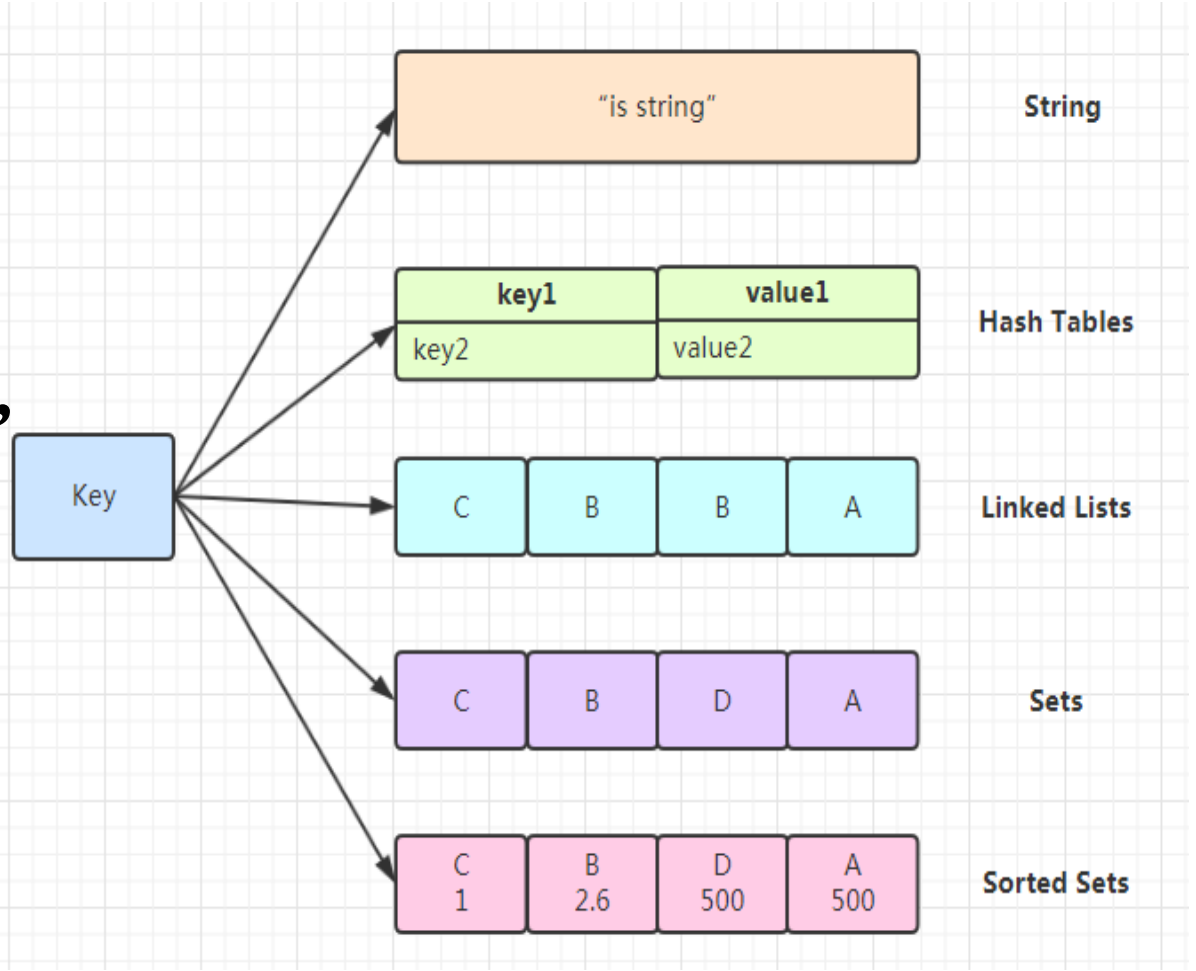
Redis的9种数据结构

- **5种普通的数据结构：**
  - String、Hash、List、Set、Zset
- **3种高级的数据结构：**
  - Bitmap、Geo、HyperLogLog
- **Redis5.0引入的全新数据结构：**
  - Stream：Stream就是Redis实现的内存版kafka。通过Redis源码中对stream的定义可知，stream底层的数据结构是radix tree（基数树，几乎就是传统的二叉树）

5.7.2 Redis简介

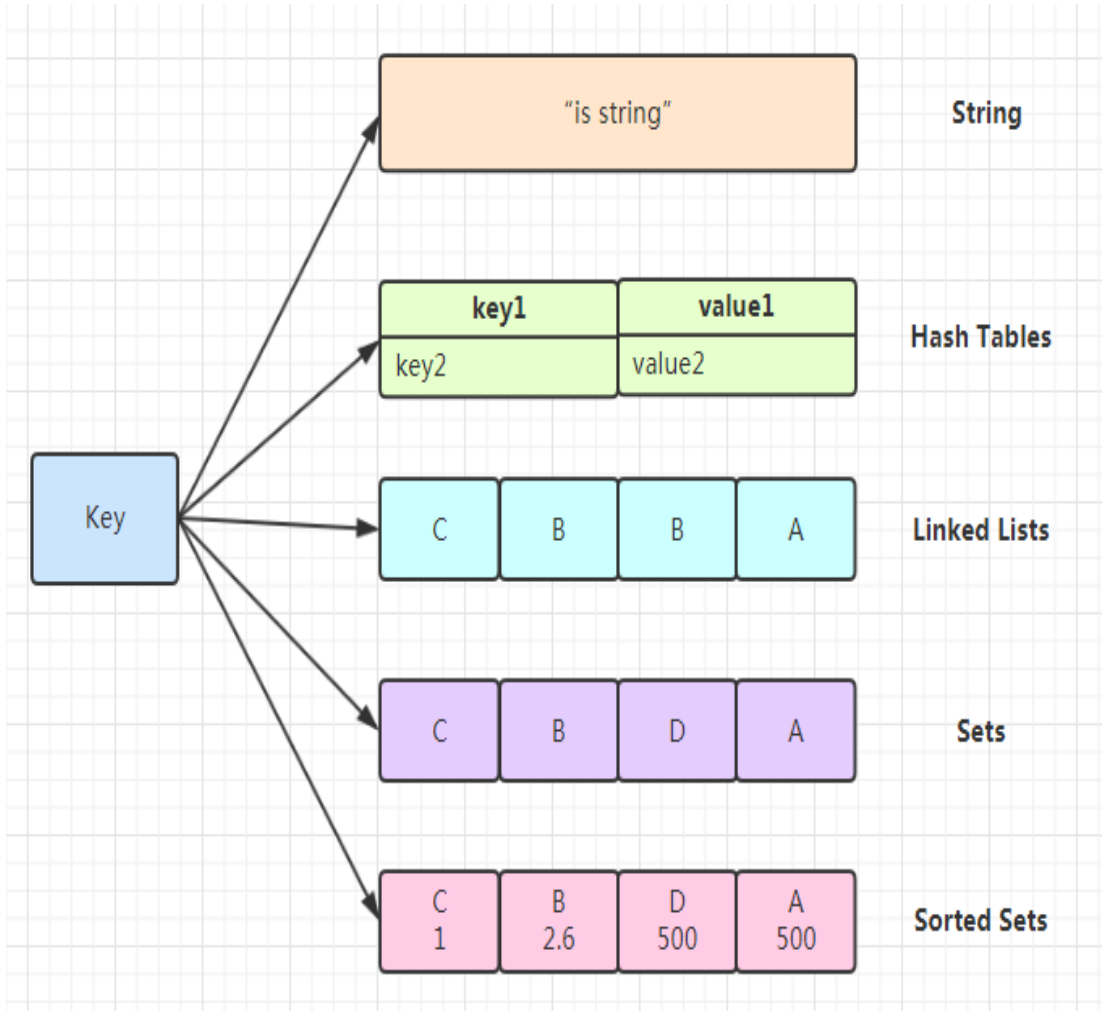
Redis的5种普通的数据结构（外层：从使用者的角度）

- **String：** 是redis中最基本的数据类型，一个key对应一个value。
- **Hash：** 是一个Mapmap，指值本身又是一种键值对结构，如 value={{field1, value1},.....fieldN,valueN}}
- **List：** 就是链表（redis 使用双端链表实现的 List），是有序的，value可以重复，可以通过下标取出对应的value值，左右两边都能进行插入和删除数据。



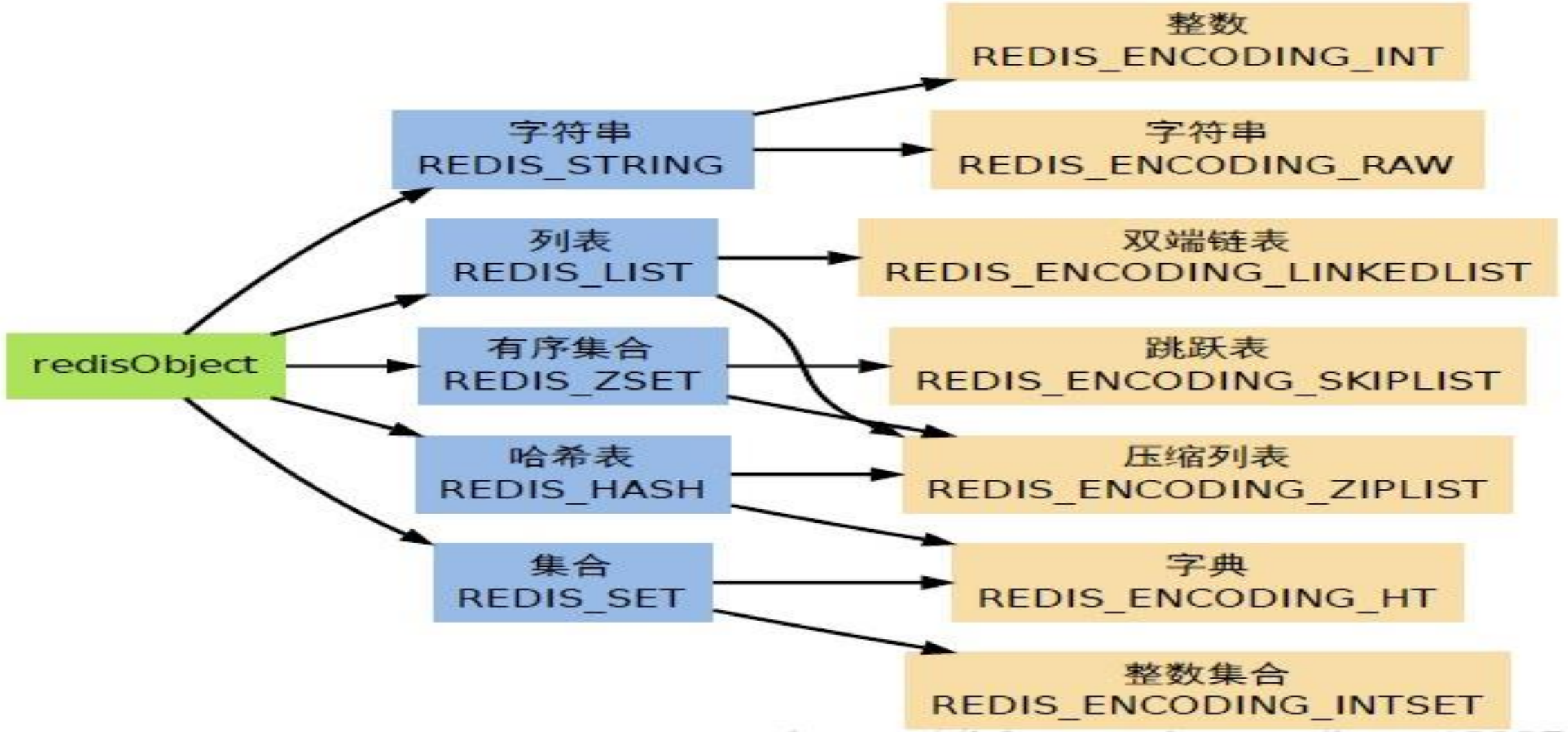
5.7.2 Redis简介

- **Set：**集合类型也是用来保存多个字符串的元素，但和列表不同的是集合中不允许有重复的元素，集合中的元素是无序的，不能通过索引下标获取元素，支持集合间的操作，可以取多个集合取交集、并集、差集。
- **Zset：**有序集合和集合有着必然的联系，保留了集合不能有重复成员的特性，区别是，有序集合中的元素是可以排序的，它给每个元素设置一个分数，作为排序的依据。



5.7.2 Redis简介

- Redis的5种普通的数据结构（底层：内部编码实现）



5.7.2 Redis简介

Redis在互联网公司的应用

- **String：** 缓存、限流、计数器、分布式锁、分布式Session
- **Hash：** 存储用户信息、用户主页访问量、组合查询
- **List：** 微博关注人时间轴列表、简单队列（内部用双向链表实现，向列表两端添加数据，获取列表片段，适合实现新鲜事场景）
- **Set：** 赞、踩、标签、好友关系
- **Zset：** 排行榜

5.7.2 Redis简介

Redis的特点

- 整个数据库统统加载在内存中进行操作，定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作（读：110000次/s，写：81000次/s），是已知性能最快的Key-Value DB。
- Redis支持保存多种数据结构，此外单个value的最大限制是1GB，不像 memcached只能保存1MB的数据，
- 数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。

5.7.2 Redis简介

- **6.0版本之前Redis是单线程的，单线程架构和I/O多路复用模型：**
  - 单线程指的是“其网络IO和键值对读写是由一个线程完成的”
  - Redis作为一个成熟的分布式缓存框架，由很多个模块组成，如网络请求模块、数据操作模块、索引模块、存储模块、高可用集群支撑模块等。
  - Redis中只有网络请求模块和数据操作模块是单线程的。而其它的如持久化存储模块、集群支撑模块等是多线程的。
- **6.0版本引入了多线程：**
  - 网络请求过程采用了多线程，而数据操作模块仍然是单线程处理的，所以依然是并发安全的。

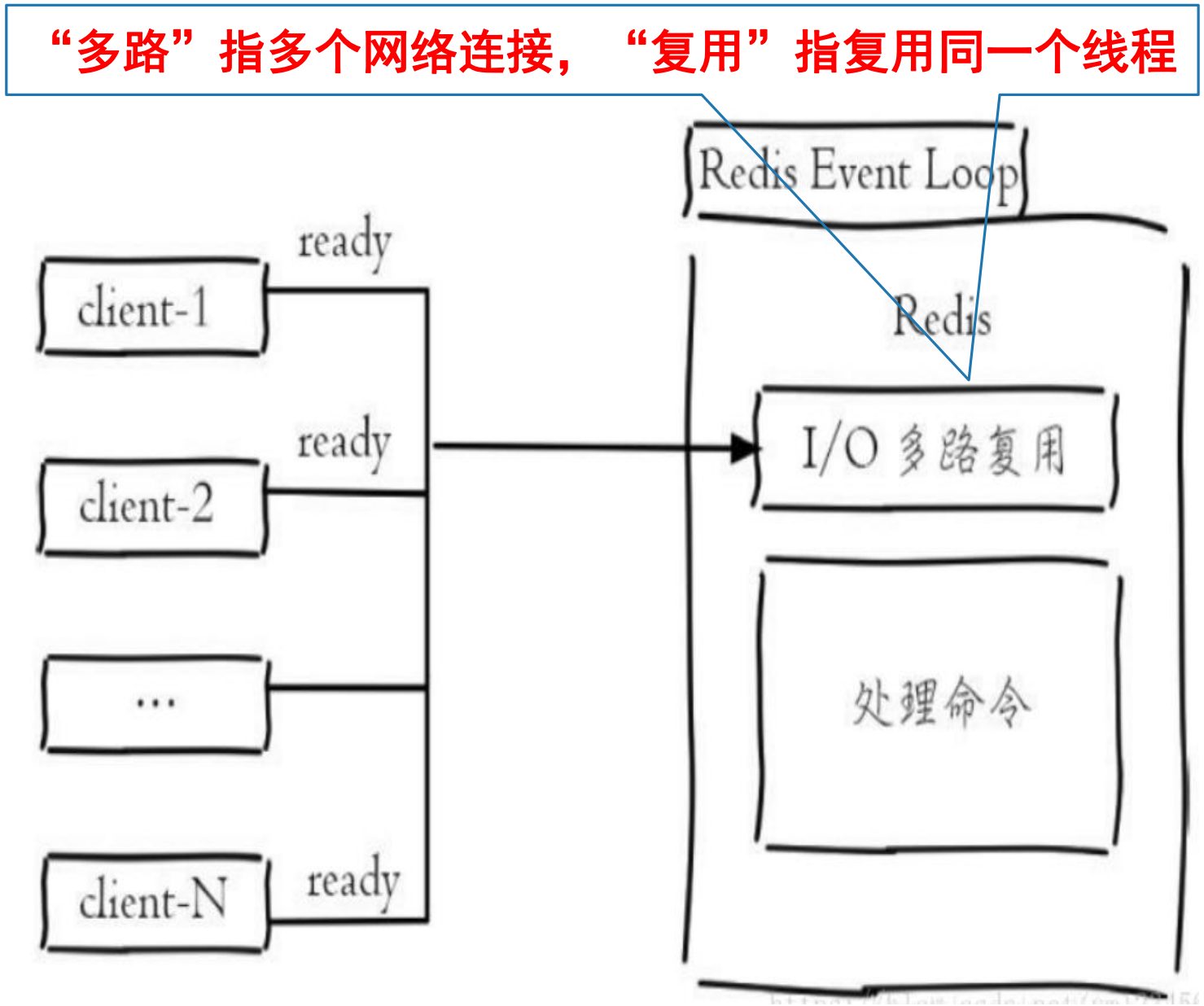
5.7.2 Redis简介

➢ I/O多路复用模型：

- 首先，Redis是跑在单线程中的，所有的操作都是按照顺序线性执行的，但是由于读写操作等待用户输入或输出都是阻塞的，所以 I/O 操作在一般情况下往往不能直接返回，这会导致某一文件的 I/O 阻塞导致整个进程无法对其它客户提供服务，而 I/O 多路复用就是为了解决这个问题而出现的。
- 多路I/O复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

5.7.2 Redis简介

- 采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗），且 Redis 在内存中操作数据的速度非常快，也就是说内存内的操作不会成为影响Redis性能的瓶颈，造就了 Redis 具有很高的吞吐量



5.7.3 Redis安装和使用

- Redis简介
- 安装Redis
- Redis实例演示

https://dblab.xmu.edu.cn/blog/1513/#more-1513

Redis小练习

1、（判断题）Redis是单进程单线程的。

正确。redis利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销

2、为什么Redis需要把所有数据放到内存中？

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度会严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

Redis小练习

3、Redis集群会有写操作丢失吗？为什么？

Redis并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

4、怎么理解Redis事务？

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

本章小结

- 本章介绍了NoSQL数据库的相关知识
- NoSQL数据库较好地满足了大数据时代的各种非结构化数据的存储需求，开始得到越来越广泛的应用。但是，需要指出的是，传统的关系数据库和NoSQL数据库各有所长，彼此都有各自的市场空间，不存在一方完全取代另一方的问题，二者会共同存在，满足不同应用的差异化需求

本章小结

- NoSQL数据库主要包括键值数据库、列族数据库、文档型数据库和图形数据库等四种类型，不同产品都有各自的应用场合。CAP、BASE和最终一致性是NoSQL数据库的三大理论基石，是理解NoSQL数据库的基础
- 介绍了融合传统关系数据库和NoSQL优点的NewSQL数据库
- 介绍了Redis数据库的安装和使用