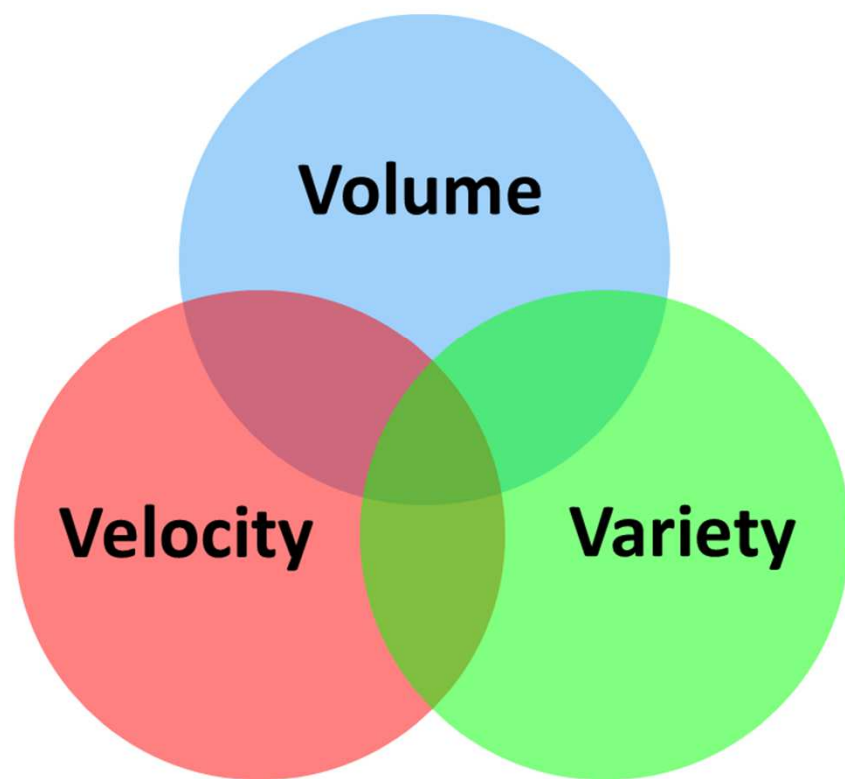


大数据系统与大规模数据分析

大数据存储系统 (3)



陈世敏

中科院计算所
计算机体系结构
国家重点实验室

©2015-2020 陈世敏

作业时间安排

周次	内容	作业
第4周, 3/11	大数据存储系统1: 基础, 文件系统, HDFS	作业1布置
第5周, 3/18	大数据存储系统2: 键值系统	
第6周, 3/25	大数据存储系统3: 图存储, document store	
第7周, 4/1	大数据运算系统1: MapReduce, 图计算系统	作业1提交 作业2布置
第8周, 4/8	大数据运算系统2: 图计算系统, MR+SQL	
第9周, 4/15	大数据运算系统3: 内存计算系统	大作业布置
第10周, 4/22	最邻近搜索和位置敏感 (LHS) 哈希算法	作业2提交
第11周, 4/29	数据空间的维度约化	
第12周, 5/6	推荐系统	作业3
第13周, 5/13	流数据采样与估计、流数据过滤与分析	
第14周, 5/20	教育大数据的建模与分析	
第15周, 5/27	期末考试	
第16周, 6/3	大作业验收报告	大作业验收



作业1安排

- 成绩：占总成绩10%

- 时间

- ☐ 发布：2020/3/11(Wed)前
- ☐ 上交：**2020/4/1(Wed)**，北京时间 6:59pm（共3周）
- ☐ 在sep课程网站>作业>作业1中提交
- ☐ 晚交
 - 最晚：2020/4/8(Wed)，北京时间 6:59pm，将扣除20%成绩
 - 之后不再接收，作业1成绩为0

- 抄袭：课程总分为0！

作业提交的格式

- 文件命名

- 组号_学号_hw1.java

- 例如：0_201618013229032_hw1.java

- 注意：上述文件名没有空格；不能上传rar或zip文件

- 程序中Java class的名字必须为

- Hw1GrpX，其中X为组号，注意大小写

- 例如：Hw1Grp0

- 自动检查程序会根据学号自动寻找对应的文件，重新命名为Hw1GrpX.java、编译、执行

- 如果名称不正确，将无法找到或不能执行，就成绩=0

大数据存储系统（1）

- 分布式系统基本概念
 - 网络与协议
 - 通信方式
 - 分布式系统类型、故障类型、CAP
- 分布式文件系统
 - NFS
 - AFS
- Google File System和HDFS

大数据存储系统（2）

- Key-Value Store

- ☐ Dynamo
- ☐ Bigtable / Hbase
- ☐ Cassandra
- ☐ RocksDB

- Distributed Coordination: ZooKeeper

- ☐ 概念
- ☐ 数据模型和API
- ☐ 基本原理
- ☐ 应用举例

Outline

- Document Store
 - 树状结构数据模型
 - JSON
 - Google Protocol Buffers
 - MongoDB
 - API and Query Model
 - Architecture
- 图存储系统(Graph Database)

JSON

- JavaScript Object Notation

- 是一个低成本的数据交换格式
- 是Javascript程序语言标准(1999年)的子集

- JSON对应于程序语言中的结构与数组

- 举例：

```
{
  "id" : 131234, "name" : "张飞", "major" : "计算机", "year": 2013, "course" :
  [ {"course name" : "体系结构", "year" : "2014", "grade" : 85},
    {"course name" : "操作系统", "year" : "2014", "grade" : 90},
    {"course name" : "英语", "year" : "2013", "grade" : 87} ],
  "address": {"州" : "幽州",
              "郡" : "涿郡",
              "街道" : "张家庄",
              "邮编" : "072750"}
}
```


JSON 格式定义

- Value

- 基础类型: string, number, true/false, null
- Object
- Array

- Object

{“key₁” : value₁, ... , “key_n” : value_n}

- Array

[value₁ , ... , value_n]

JSON举例

- 我们仔细分析一下这个例子：

“key” : string **“key” : number**

```
{“id” : 131234, “name” : “张飞”, “major” : “计算机”, “year” : 2013,
  “course” :
    [ {“course name” : “体系结构”, “year” : “2014”, “grade” : 85},
      {“course name” : “操作系统”, “year” : “2014”, “grade” : 90},
      {“course name” : “英语”, “year” : “2013”, “grade” : 87} ],
  “address” : {“州” : “幽州”,
               “郡” : “涿郡”,
               “街道” : “张家庄”,
               “邮编” : “072750”}
}
```

整体是一个 object

JSON举例

- 我们仔细分析一下这个例子：

```
{“id” : 131234, “name” : “张飞”, “major” : “计算机”, “year”: 2013,  
  “course” :
```

```
[ {“course name” : “体系结构”, “year” : “2014”, “grade” : 85},  
  {“course name” : “操作系统”, “year” : “2014”, “grade” : 90},  
  {“course name” : “英语”, “year” : “2013”, “grade” : 87} ],
```

```
  “address”: {“州” : “幽州”,  
              “郡” : “涿郡”,  
              “街道” : “张家庄”,  
              “邮编” : “072750”}
```

```
}
```

嵌套的Array,
Array的每个元素是一个object

JSON举例

- 我们仔细分析一下这个例子：

```
{“id” : 131234, “name” : “张飞”, “major” : “计算机”, “year”: 2013,  
  “course” :  
    [ {“course name” : “体系结构”, “year” : “2014”, “grade” : 85},  
      {“course name” : “操作系统”, “year” : “2014”, “grade” : 90},  
      {“course name” : “英语”, “year” : “2013”, “grade” : 87} ],  
  “address”: {“州” : “幽州”,  
              “郡” : “涿郡”,  
              “街道” : “张家庄”,  
              “邮编” : “072750”}  
}
```

嵌套的object

JSON举例

- 我们仔细分析一下这个例子：

```
{“id” : 131234, “name” : “张飞”, “major” : “计算机”, “year”: 2013,  
  “course” :  
    [ {“course name” : “体系结构”, “year” : “2014”, “grade” : 85},  
      {“course name” : “操作系统”, “year” : “2014”, “grade” : 90},  
      {“course name” : “英语”, “year” : “2013”, “grade” : 87} ],  
  “address”: {“州” : “幽州”,  
              “郡” : “涿郡”,  
              “街道” : “张家庄”,  
              “邮编” : “072750”}  
}
```

☞ JSON的数据类型定义是完全动态的，一个JSON记录本身自定义了自己的类型，不需要事先声明schema

Google Protocol Buffers

- Google推出，最初用于实现网络协议，所以叫protocol buffers
- 可以用于表达程序设计语言中的结构和数组
 - Google提供了一套软件库，可以把Protocol buffers的记录压缩编码，和解压缩解码，用于网络传输和存储
- 要求先定义类型，然后才可以表达数据

Google Protocol Buffers 举例

r₁

DocId: 10

Links

- Forward: 20
- Forward: 40
- Forward: 60

Name

- Language
 - Code: 'en-us'
 - Country: 'us'
- Language
 - Code: 'en'
- Url: 'http://A'

Name

- Url: 'http://B'

Name

- Language
 - Code: 'en-gb'
 - Country: 'gb'

```
message Document {  
  required int64 DocId;  
  optional group Links {  
    repeated int64 Backward;  
    repeated int64 Forward; }  
  repeated group Name {  
    repeated group Language {  
      required string Code;  
      optional string Country; }  
    optional string Url; } }
```

定义了数据类型

r₂

DocId: 20

Links

- Backward: 10
- Backward: 30
- Forward: 80

Name

- Url: 'http://C'

数据记录

图来源: "Dremel: Interactive Analysis of Web-Scale Datasets". PVLDB 3(1): 330-339 (2010)

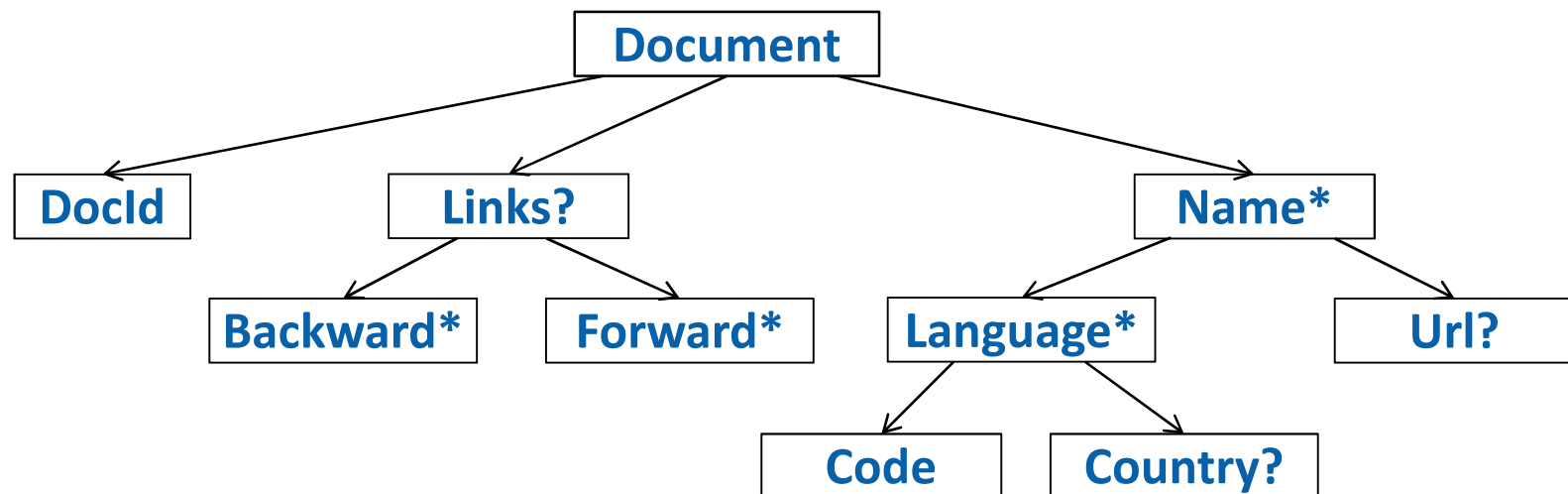
Google Protocol Buffers 类型举例

```
message Document {  
  required int64 DocId;  
  optional group Links {  
    repeated int64 Backward;  
    repeated int64 Forward; }  
  repeated group Name {  
    repeated group Language {  
      required string Code;  
      optional string Country; }  
    optional string Url; }}  
}
```

把嵌套关系表达为一课语法树

?: optional

*: repeated



JSON vs. Google Protocol Buffers

• 相同点

- 都可以表达程序设计语言中的结构和数组
- 嵌套：JSON object, PB message/group
- 数组：JSON array, PB repeated
- 缺值的情况：JSON记录实际上没有规定一定要有什么域，PB optional

• 不同点

- 数据类型：PB要求事先声明，JSON不需要
- 实际使用情况不太一样
 - JSON数据设计的初衷是文本的
 - 但是MongoDB支持一种BSON的二进制编码方式
 - PB从设计开始就是二进制编码的，用于编码协议

JSON vs. XML

- 都是半结构化表示
- 10年前，XML刚出现时，期望很高
 - 但是XML非常heavy-weight
 - XML文件需要格式定义，XML的各种tag也使其比较大
- JSON比较light-weight
 - 自定义格式，简单的key-value形式
 - 已经逐渐为大量的应用所采用

其它相关数据格式/系统

- Apache Avro: 一种半结构化数据格式
- Apache Thrift
 - 基于类似的理念, 实现多语言的互相RPC调用

Document Store



- Document store
 - JSON是基本数据类型，存储为BSON二进制表示
- 基于C++实现
- 名词
 - Database ~ 关系型中的数据库概念
 - Collection ~ 关系型中的table概念
 - Document ~ 关系型中的记录概念
- 一个database包含多个collections, 每个collection包含多个documents
 - document < 16MB

MongoDB

- 安装完毕后，主要有3个可执行程序
 - ❑ mongod - The database process.
 - ❑ mongos - Sharding controller.
 - ❑ mongo - The database shell (uses interactive javascript).
- 下面的例子都是基于mongo

MongoDB vs. SQL

- SQL的简单功能在MongoDB中都有对应的功能
- 下面我们以一个具体的例子来说明

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
student_id: 131234, name: “张飞”, major: “计算机”, year: 2013}
```

注意：mongodb中key的双引号省略了

MongoDB的语法知道一下，考试不要求会写，但需要读懂

SQL create table

MongoDB

```
db.student.insert({  
  student_id: 131234,  
  name: “张飞”,  
  major: “计算机”,  
  year: 2013  
})
```

SQL

```
create table student (  
  student_id integer,  
  name varchar(20),  
  major varchar(20),  
  year integer  
);
```

JSON不定义类型，多以没有表定义的方法，
第一个插入生成collection

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
 student_id: 131234, name: “张飞”, major: “计算机”, year: 2013}
```

SQL insert

MongoDB

```
db.student.insert({  
  student_id : 131234,  
  name : “张飞”,  
  major : “计算机”,  
  year: 2013  
})
```

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
  student_id : 131234, name : “张飞” , major : “计算机” , year: 2013}
```

SQL

```
insert into student  
values (131234, “张飞”, “计算  
机”, 2013);
```


SQL select

MongoDB

```
db.student.find(  
  { major : “计算机” },  
  { name : 1, year:1, _id:0 }  
)
```

SQL

```
select name, year  
from student  
where major = “计算机”;
```

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
 student_id : 131234, name : “张飞” , major : “计算机” , year: 2013}
```

SQL group by aggregation

MongoDB

```
db.student.aggregate( [  
  {$match : { major : “计算机” }},  
  {$group : { _id:{year: “$year”},  
              cnt: {$sum: 1}}  
])
```

SQL

```
select year, count(*) as cnt  
from student  
where major = “计算机”  
group by year;
```

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
 student_id : 131234, name : “张飞” , major : “计算机” , year: 2013}
```

SQL group by aggregation

MongoDB 另一种实现

```
db.student.group({  
  key: { year: 1 },  
  cond: { major: "计算机" },  
  $reduce:  
    function(curr, result){  
      result.total += 1;  
    },  
  initial: {total: 0}  
})
```

```
{_id: ObjectId("509a8fb2f3f4948bd2f983a0"),  
  student_id : 131234, name : "张飞" , major : "计算机" , year: 2013}
```

SQL

```
select year, count(*) as cnt  
from student  
where major = "计算机"  
group by year;
```

其它运算相关

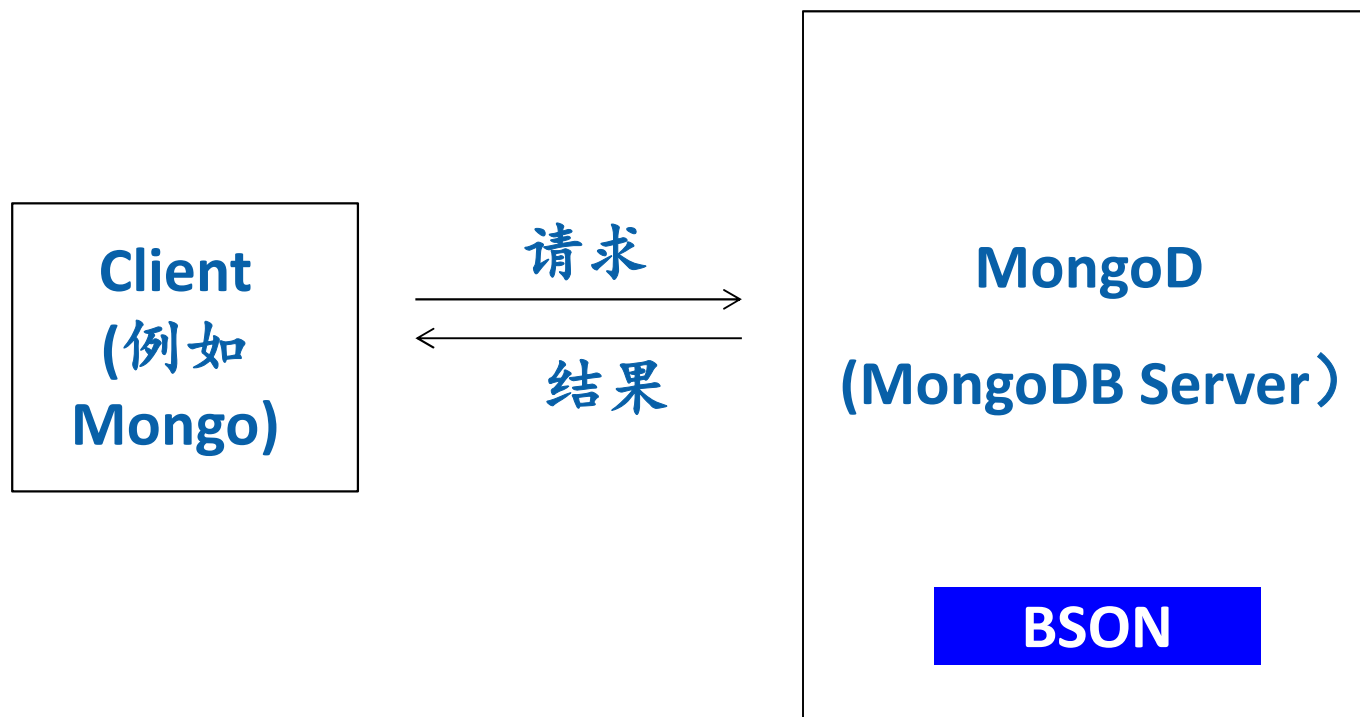
- 不支持Join

- 两个Collection之间不能通过join连接起来
- 想法
 - 相互关联的数据可以放在一个document中，嵌套表达

- 可以定义Index

- Btree
- Node size: 8KB

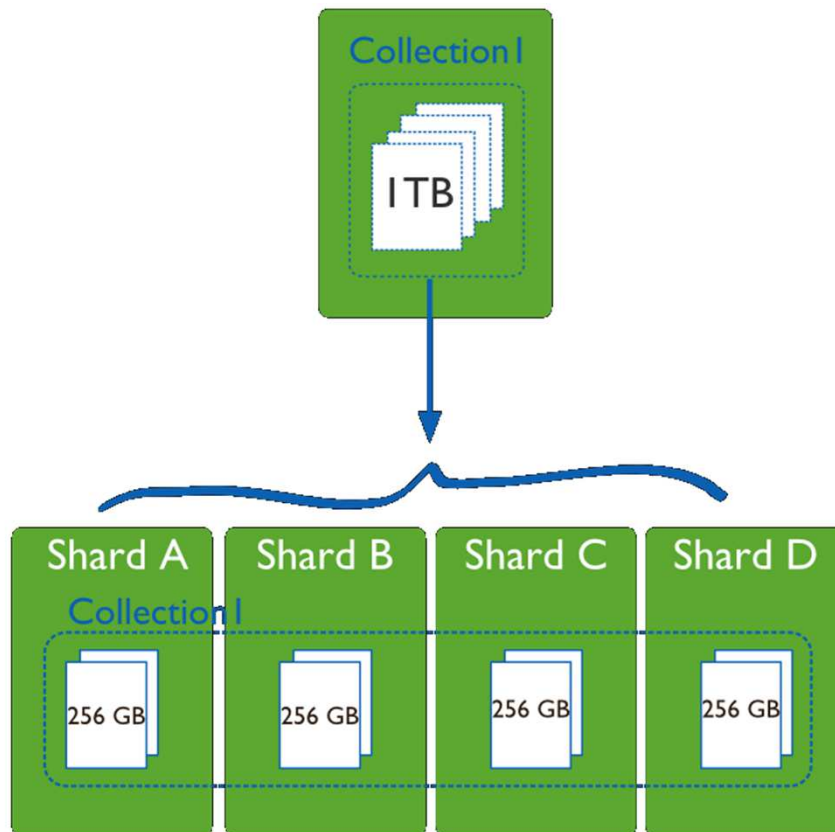
单机MongoDB



- 与RDBMS非常相似，采用Operator Tree进行查询执行
- BSON：一种JSON二进制表示
 - 行式，key都是字符串，value根据具体的类型存储二进制值
- 后端可选存储引擎：WiredTiger（默认）等

分布式MongoDB

- Sharding = horizontal partitioning

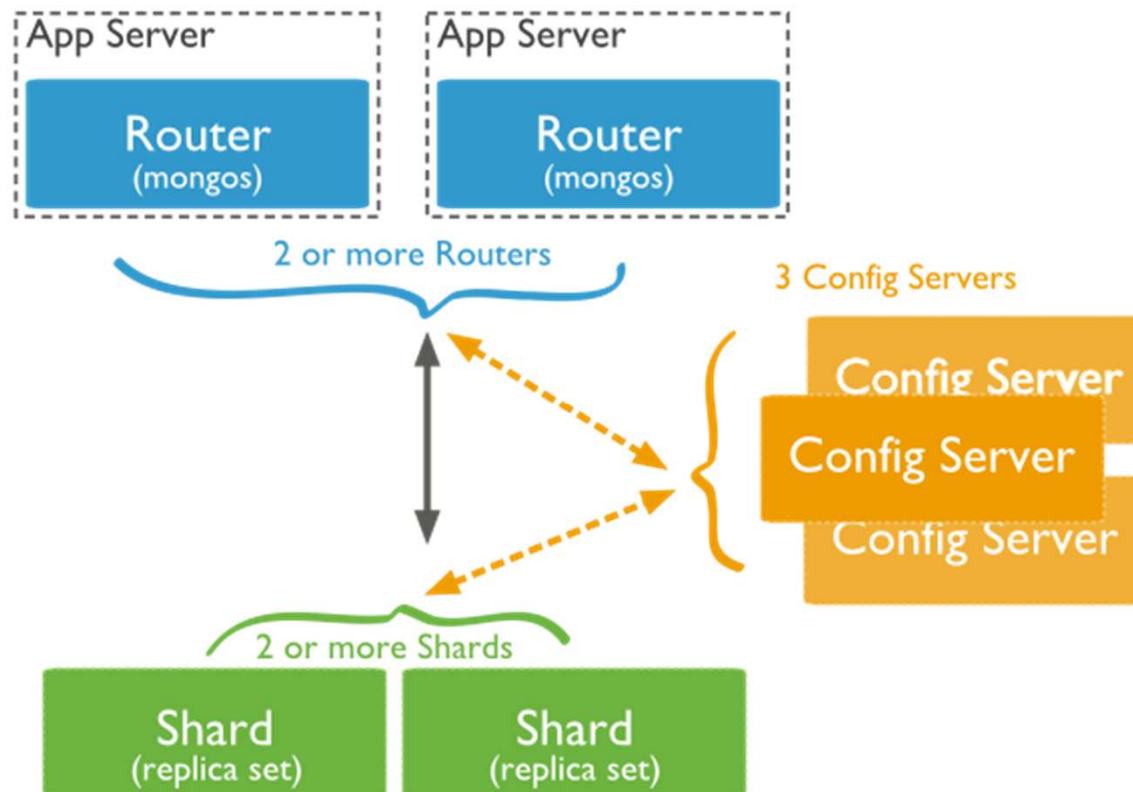


MongoDB Manual

- Shard key
- range partitioning
- hash partitioning

分布式MongoDB

- Sharding = horizontal partitioning



MongoDB Manual

- 没有Join操作
- 选择、投影、分组统计等可以在多个Shard上分布式执行
- 如果过滤条件中包括shard key, mongos会选择符合条件的shard(s)执行

ACID?

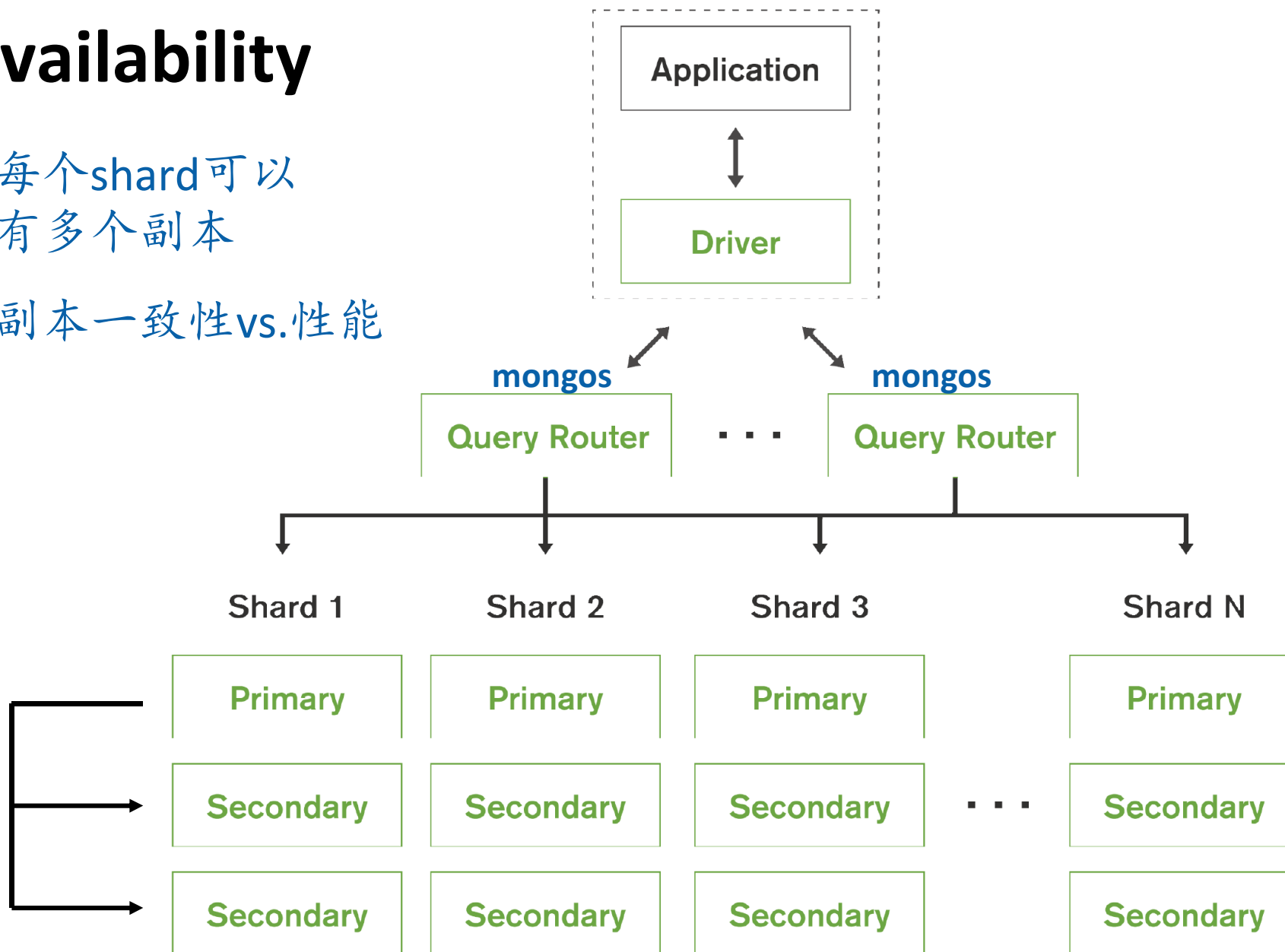
- 只保证单个document（记录）修改的一致性
 - 没有transaction概念
 - 多个记录的修改一致性没有保证
- Concurrency control
 - 根据存储引擎，具体支持Document-level 或 collection-level
 - WiredTiger支持document-level并发控制
- Journaling
 - 即logging
 - ☞ Write concern

Write Concern

- 什么时候认为写完成了？
- MongoDB有多种不同的Write Concern等级
 - Unacknowledged: 写请求发送了，就认为完成
 - Acknowledged: MongoDB应答了收到写请求，就认为完成
 - Journalled: MongoDB把写请求记录在硬盘上的日志中，认为完成
- 显然，前两种并不能保证掉电后写请求仍然有效！
 - Journal会每隔一段时间写入硬盘，这个时间间隔可能是60秒！
 - 这期间发生的数据修改可能丢失！
 - 是性能 vs. 可靠性的折中方案
 - 在配置使用时，需要小心

Availability

- 每个shard可以有多个副本
- 副本一致性vs.性能



Outline

- Document Store
- 图存储系统(Graph Database)
 - 图数据模型
 - Neo4j
 - JanusGraph
 - RDF和Sparql

图(Graph)的概念

- $G=(V, E)$

- V : 顶点(Vertex)的集合
- E : 边(Edge)的集合
 - 边 $e=(u,v)$, $u \in V$, $v \in V$

- 有向图 (directed graph)

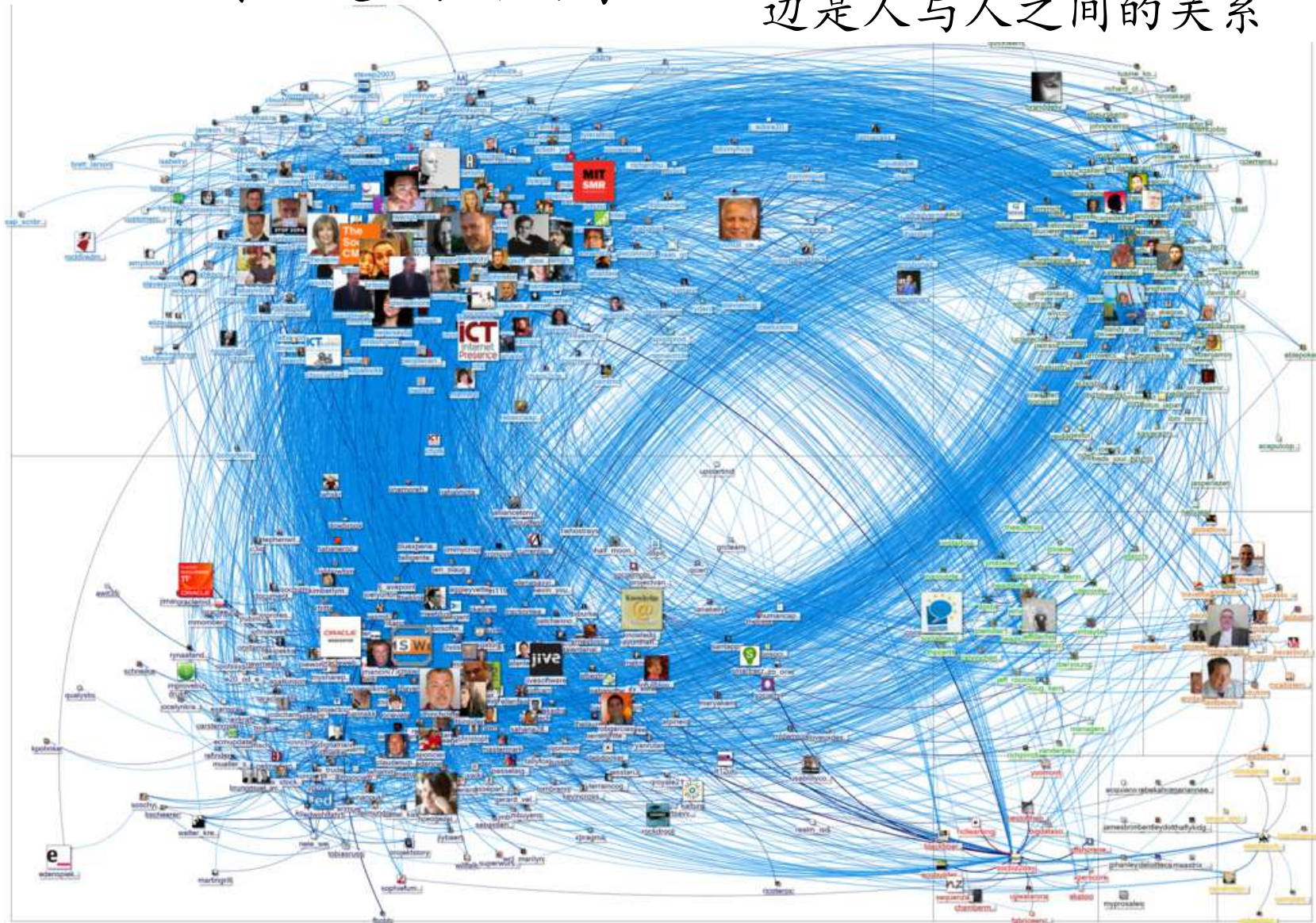
- 边有方向

- 无向图

- 边没有方向
- 可以用有向图表达无向图: 每条无向边 \rightarrow 2条有向边

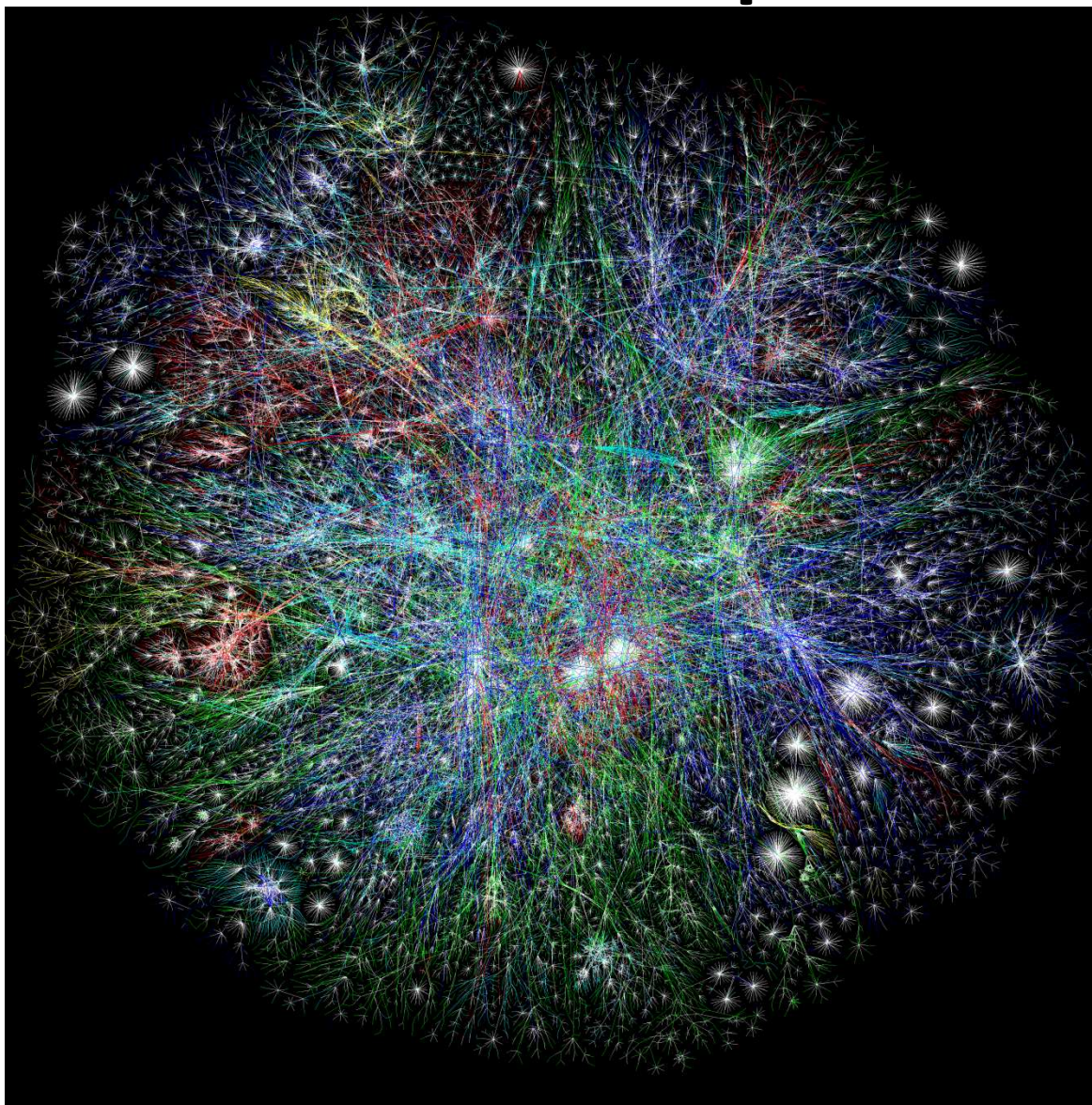
Twitter社交网络图

顶点是人，
边是人与人之间的关系



Created with NodeXL (<http://nodexl.codeplex.com>) from the Social Media Research Foundation (<http://www.smrfoundation.org>)

Internet Connection Map

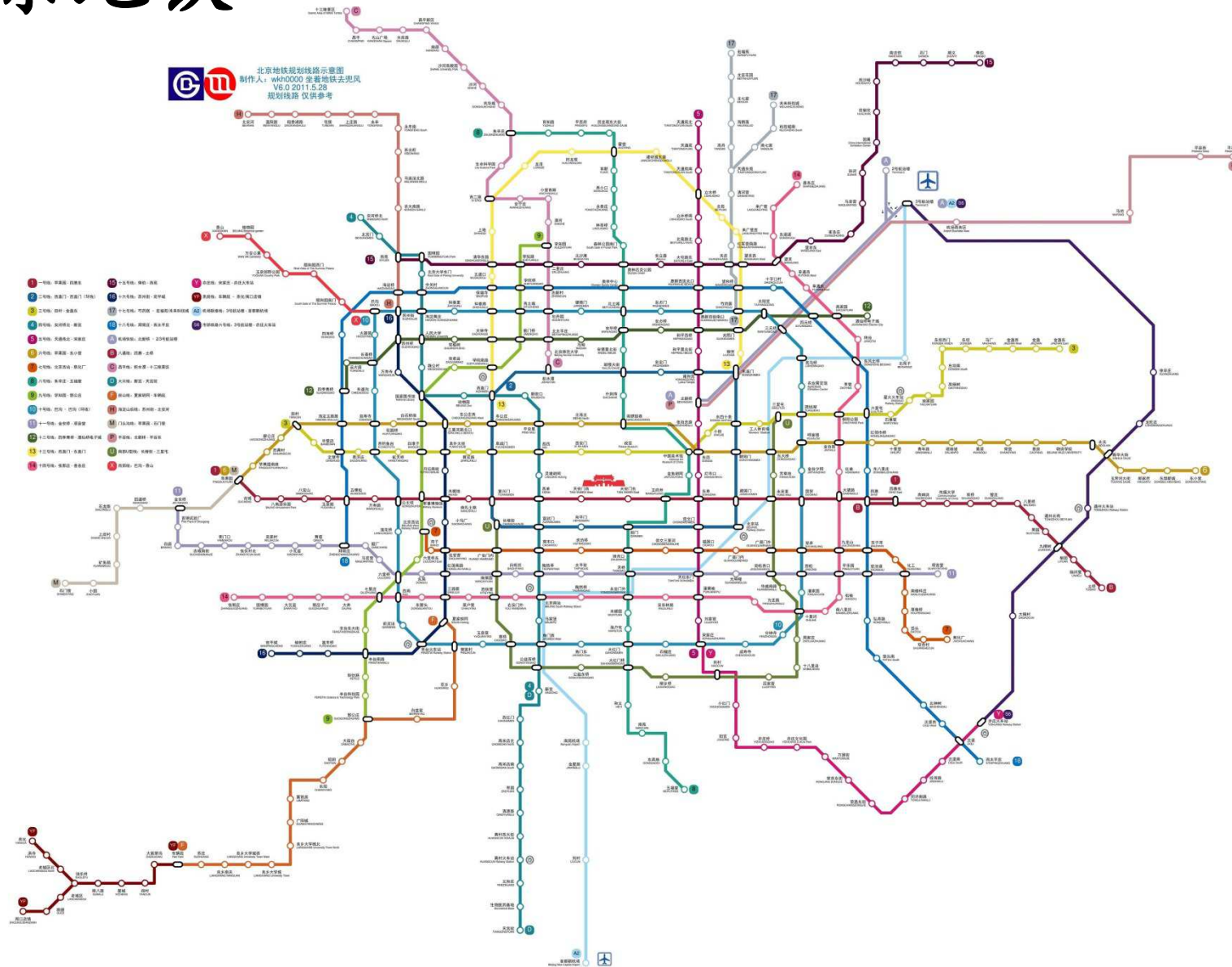


铁路网



中华人民共和国铁道部编制

北京地铁



与图相关的系统

- 图数据存储系统
 - 存储图顶点和边
 - 提供顶点和边的查询
- 图运算系统
 - 以图为基础运行大规模算法
- 本节课关注：图数据存储系统

Neo4j



- Native graph database

- 采用自定义的结构在本地硬盘存储图，而不是存在数据库关系型表中

- 开源Java实现

- Neo4j存储

- 顶点：称为node
 - 边：称为relationship
 - 顶点和边上可以存储多个key-value值：称为property

- Neo4j使用

- Cypher: Declarative query language
 - Traversal: Embedded Java lib

Cypher的语法知道一下，考试
不要求会写，但需要读懂

Node in Cypher

`(nodename:type, {property_key:value, key:value})`

例子：

`(张飞)`

`(张飞:Student)`

`(张飞:Student, {name: “张飞”, major : “计算机”,
year: 2013})`

`(体系结构:Course, {name: “体系结构”})`

Relationship in Cypher

-[name:type, {property_key:value, key:value}]->

例子:

-[:Takecourse]->

-[:Takecourse,{year:2014, grade:85}]->

Create in Cypher

CREATE (张飞:Student, {name: “张飞”, major : “计算机”,
year: 2013})

CREATE (体系结构:Course, {name: “体系结构”})

CREATE
(张飞)-[:Takecourse,{year:2014, grade:85}]->(体系结构)



Match in Cypher

- 给定一个图的模板，找到所有匹配的子图

MATCH (x {name:"张飞"}) **RETURN** x

找具有name:"张飞"属性的顶点

MATCH (x:Student)-[:Takecourse]->(:Course {name:"体系结构"})

RETURN x

找到所有选修体系结构课程的学生顶点

MATCH

({name:"数据库"})<-[:Takecourse]-(x)-[:Takecourse]->({name:"体系结构"})

RETURN x

找到既选修体系结构又选修数据库的顶点

Match in Cypher

```
MATCH (x:Student)-[:Takecourse]->(c:Course {name:"体系结构"})  
WHERE x.year=2013  
RETURN x
```

找到所有选修体系结构课程的学生顶点,
要求学生是2013年入学的

```
MATCH (x:Student)-[:Takecourse]->(c:Course {name:"体系结构"})  
WHERE NOT (x)-[:Takecourse]->(c:Course {name:"操作系统"})  
RETURN x
```

找到所有选修体系结构课程的学生顶点,
要求学生没有选修过操作系统

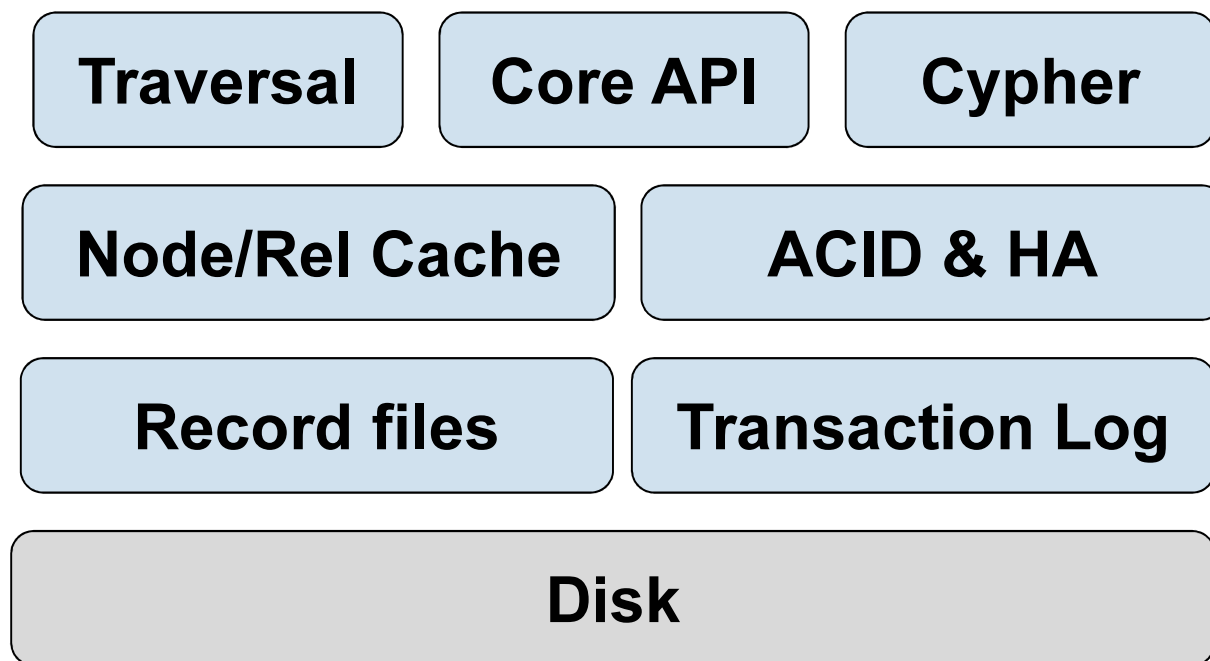
Match in Cypher

- 更多的使用

- $(a)-[*]->(b)$
有路径从a到b

- $(a)-[*3..5]->(b)$
有路径从a到b，路径最短3步，最长5步

Neo4j 系统结构



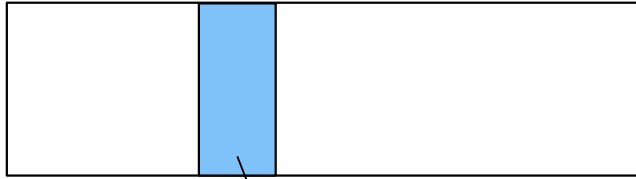
参照：“An Overview of neo4j Internals” presentation,
www.slideshare.net/thobe/an-overview-of-neo4j-internals

文件存储

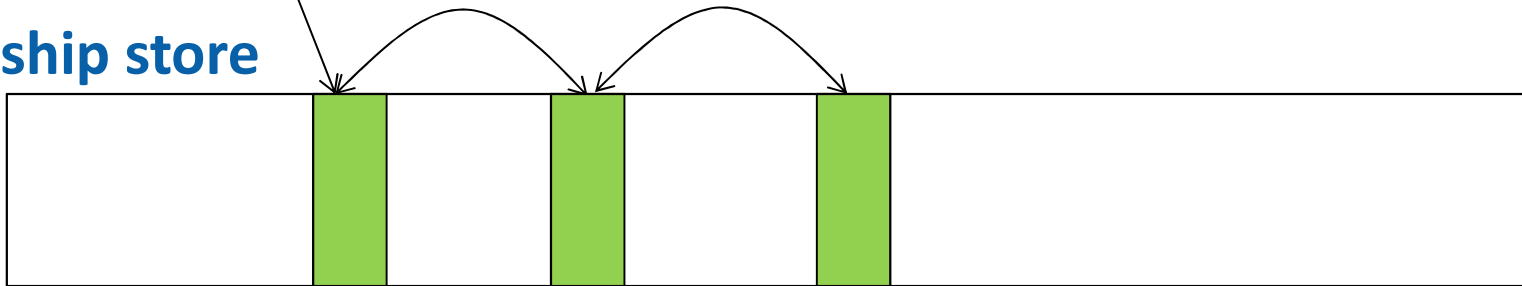
- Node store文件, Relationship store文件, Property store文件
 - 分别存储所有node, relationship, property的文件
 - unique id
 - 每个node, relationship, property都有unique id
- Relationship双向链表
 - 同一个node的relationship形成一个双向链表
 - 链接指针为relationship id
 - 相应的node中记录链表的第一个relation id
- Property单向链表
 - 同一个node/relationship的property形成一个单向链表
 - 链接指针为next property id
 - 链表第一个property id存在对应的node/relationship中

Node和Relationship

Node store文件



Relationship store
文件



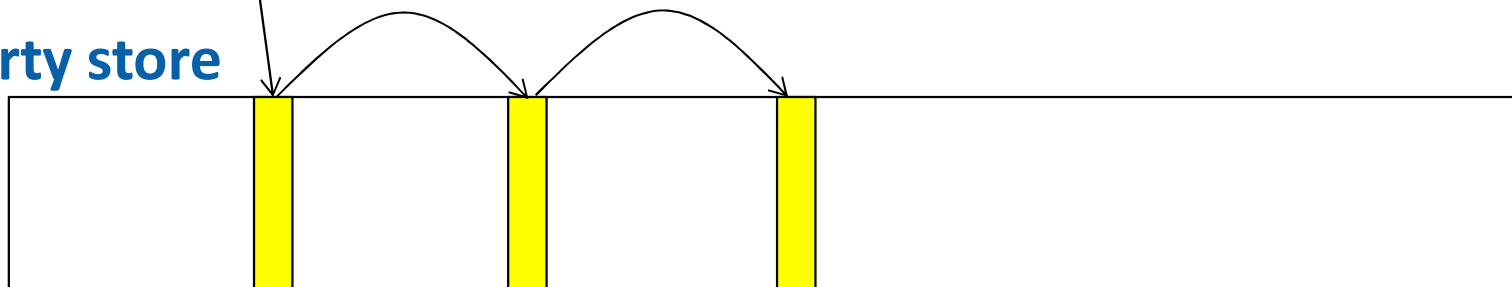
- 同一个node的relationship形成一个双向链表，链接指针为relationship id
- 相应的node中记录链表的第一个relation id

Node和Property

Node store文件



Property store
文件



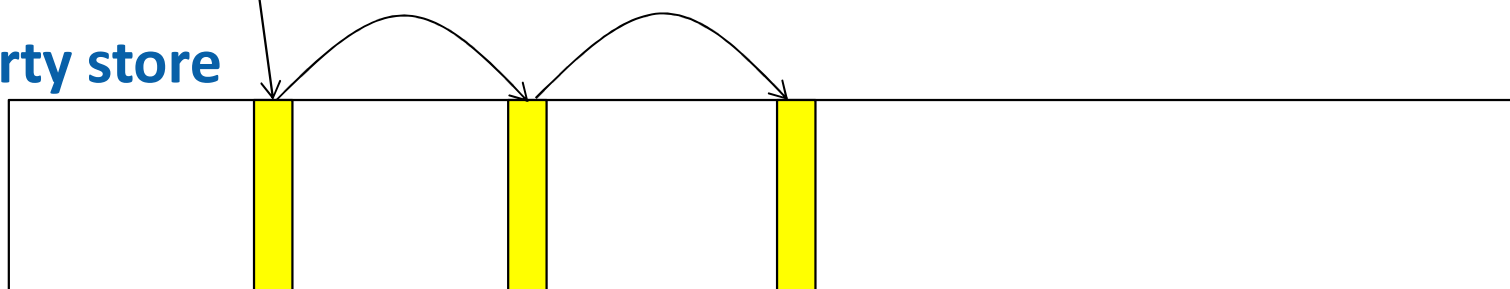
- 每个顶点node可能有多个property
- 同一个node的所有property形成一个单向链表，指针为next property id

Relationship和Property

Relationship store文件



Property store
文件



- 每个边relationship可能有多个property
- 同一个relationship的所有property形成一个单向链表，指针为next property id

内存缓冲区

- OS的文件系统有缓冲池
 - 文件page的缓冲
- Node/Rel cache
 - Neo4j对node, relationship, property缓冲
 - 内部的数据结构是以node和relationship为主要单位的
 - Property以key-value形式附加在node/relationship上

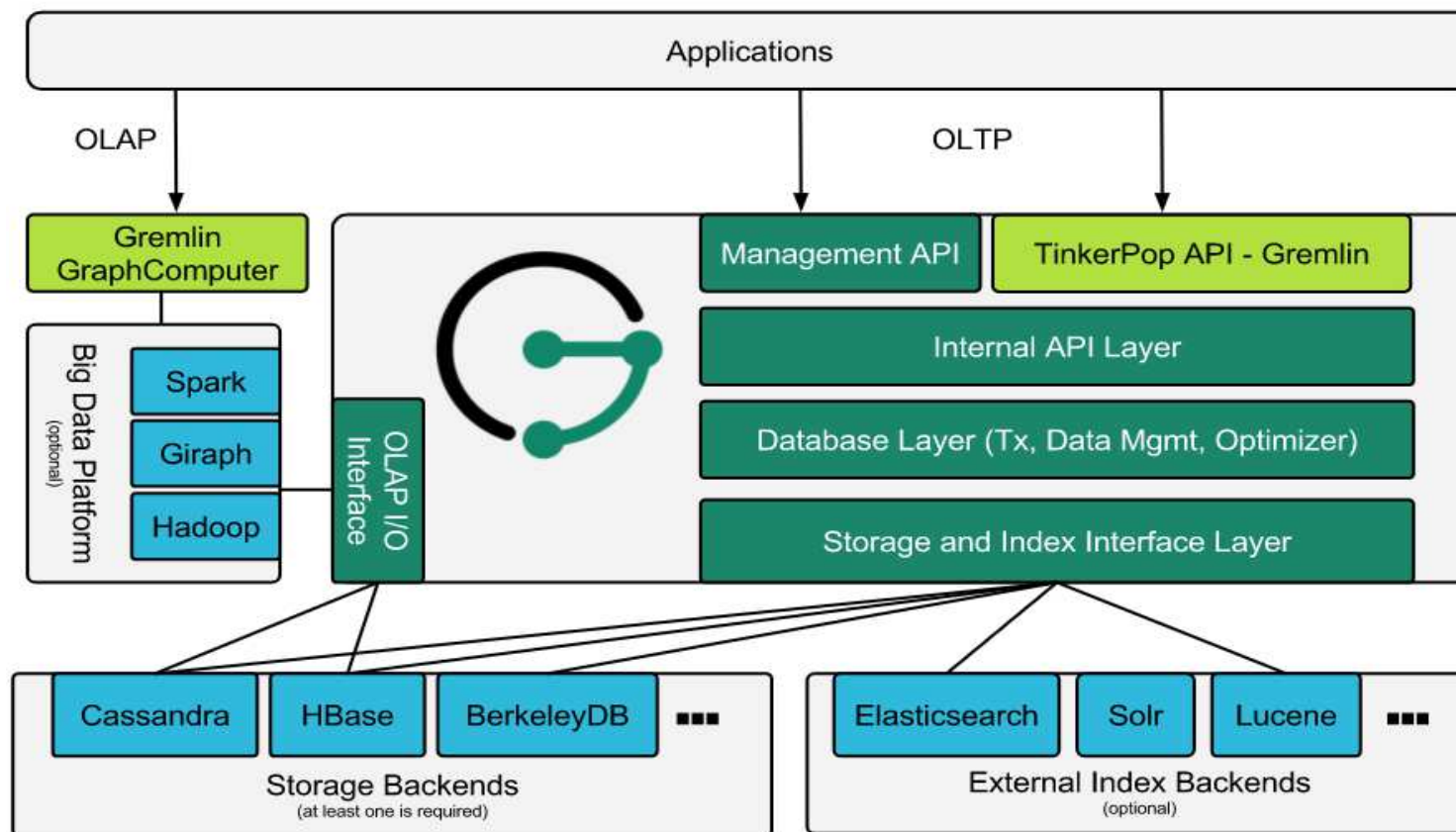
ACID

- 定义了transaction概念
 - 在embedded Java中显式使用
 - 在cypher中隐式使用
- 采用类似snapshot isolation的机制
 - 一个transaction的写先保存起来，直至transaction.finish()时，才真正尝试修改数据
 - 可能commit / rollback
- 采用Transaction log

Availability

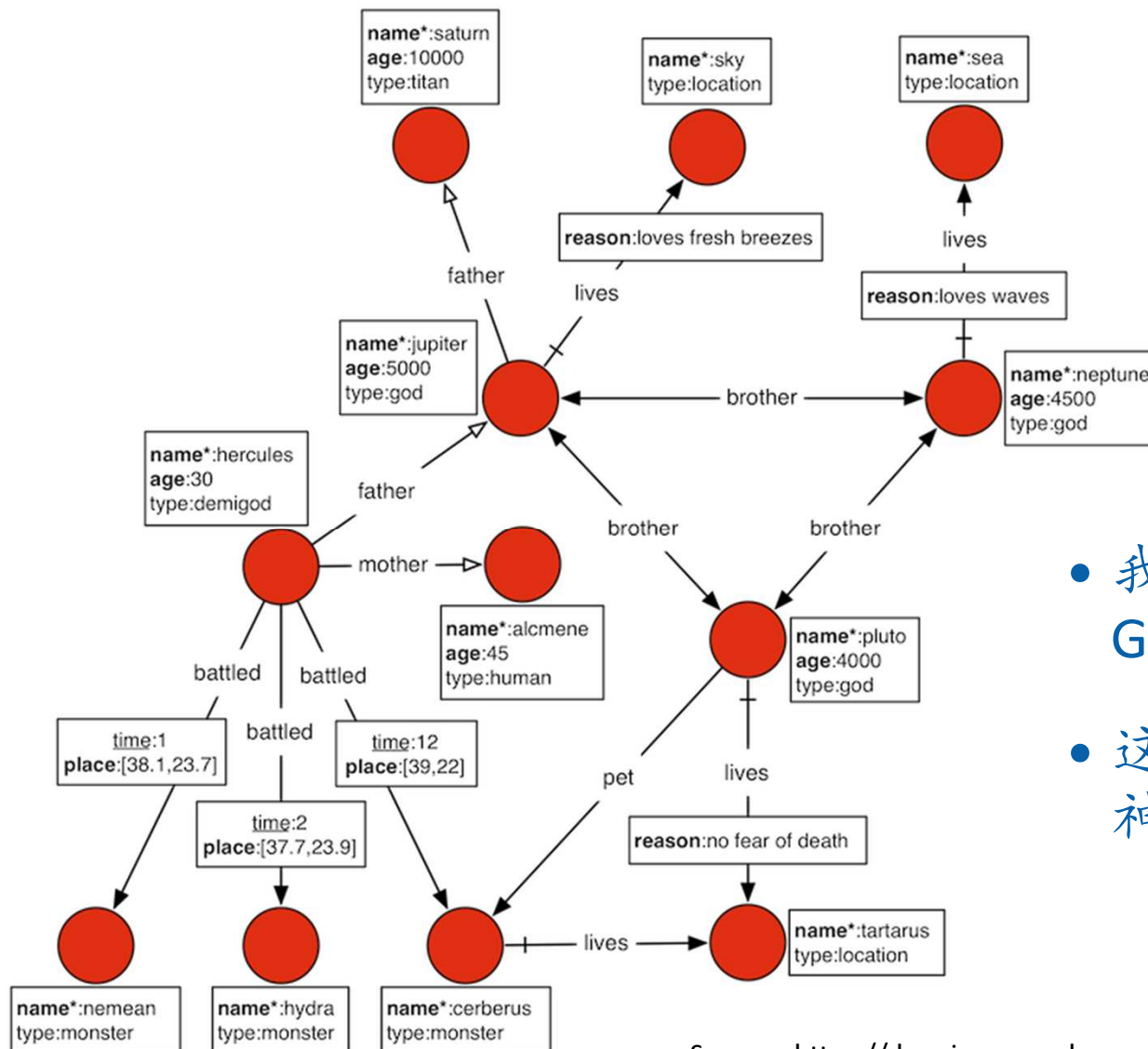
- HA: High Availability的缩写
- 采用多副本
 - 主副本把transaction log发送到从副本
 - 从副本replay log从而执行同样的操作

JanusGraph: 分布式图数据库



后端使用Key-Value Store来存储图数据

JanusGraph使用Gremlin查询语言

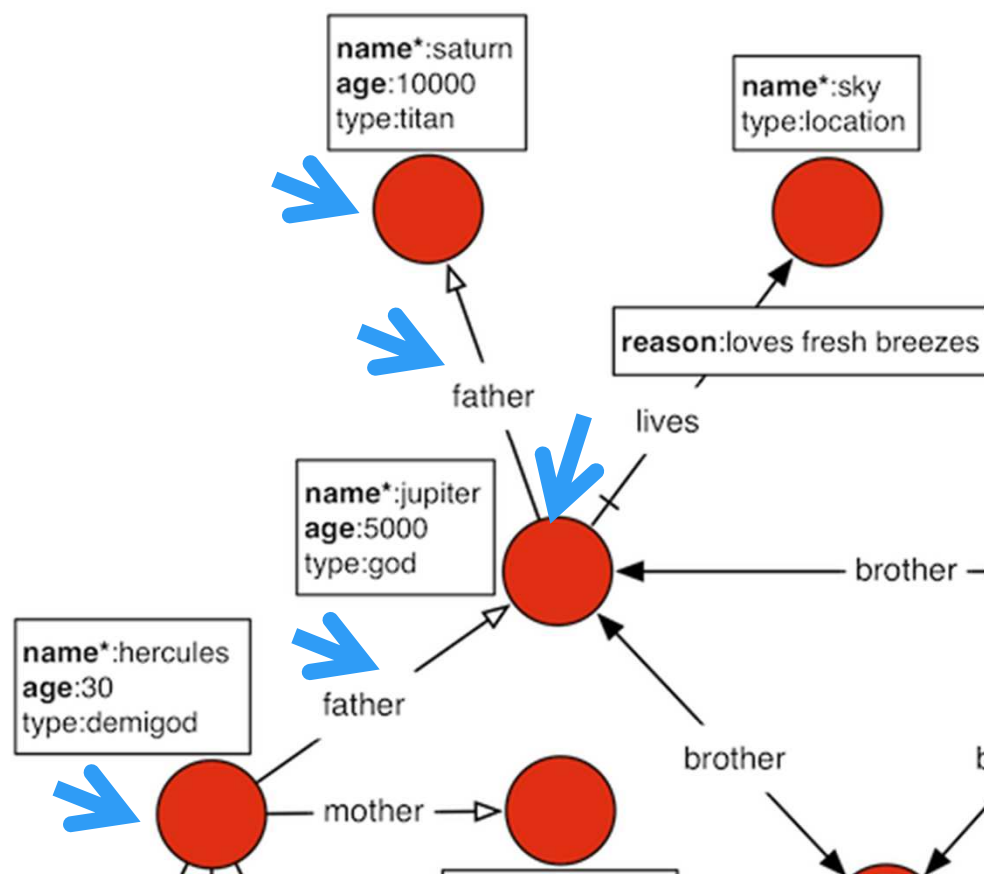


Source: <https://docs.janusgraph.org>

- 我们举例说明 Gremlin
- 这是一个图，表示神话关系（罗马名）

Gremlin 查询举例

```
gremlin> g.V().has('name', 'hercules').out('father').out('father').values('name')  
=>saturn
```



按照路径表达查询

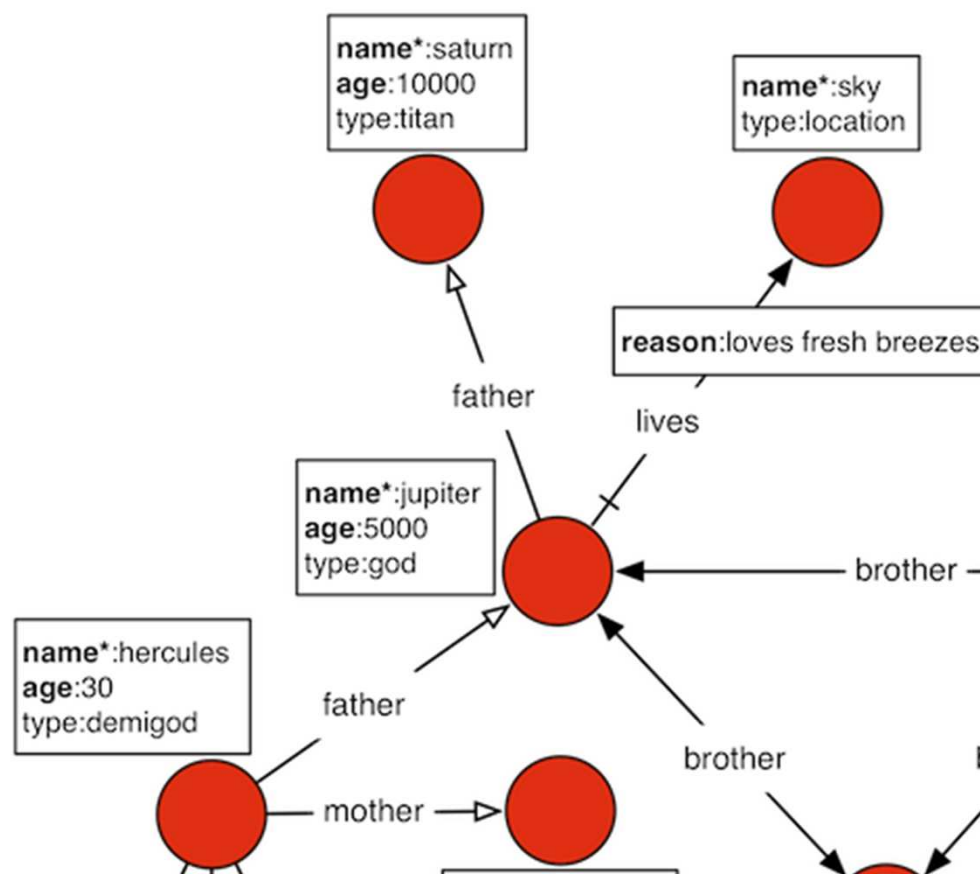
- 上述查询可以分解如下

```
gremlin> g
==>graphtraversalsource[janusgraph[cql:127.0.0.1], standard]
gremlin> g.V().has('name', 'hercules')
==>v[24]
gremlin> g.V().has('name', 'hercules').out('father')
==>v[16]
gremlin> g.V().has('name', 'hercules').out('father').out('father')
==>v[20]
gremlin> g.V().has('name', 'hercules').out('father').out('father').values('name')
==>saturn
```

Gremlin查询举例

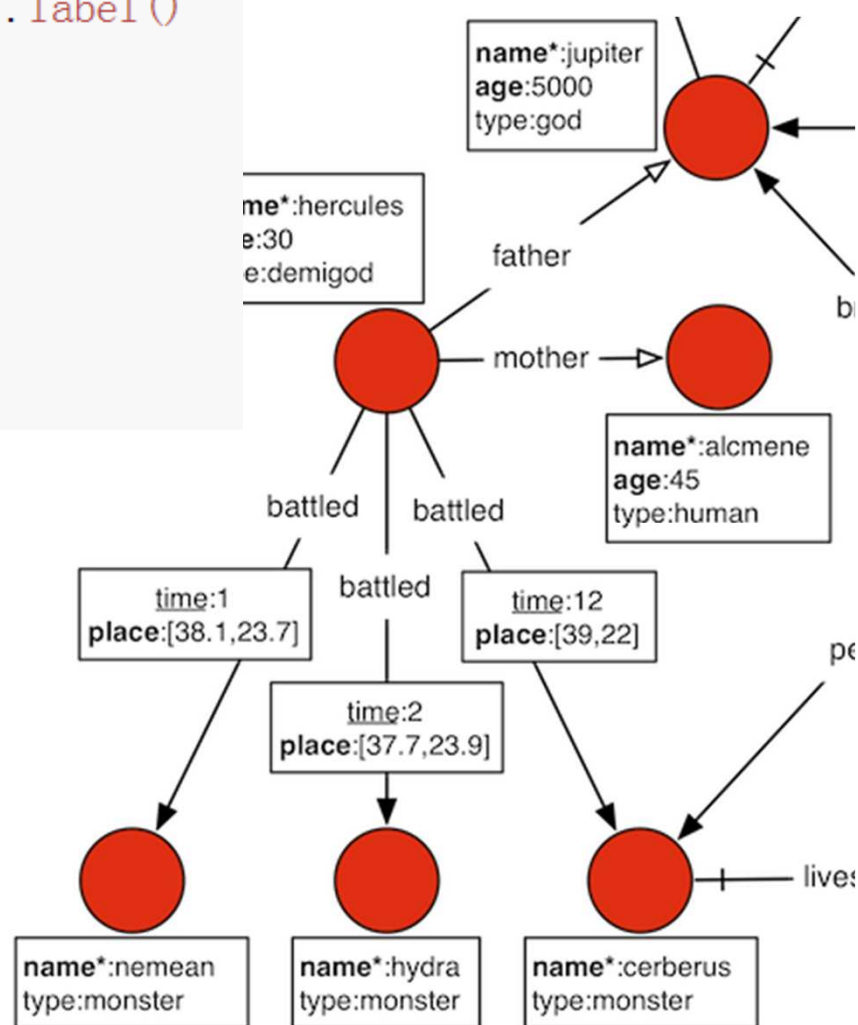
```
gremlin> g.V().has('name', 'hercules').repeat(out('father')).emit().values('name')  
=>jupiter  
=>saturn
```

沿着father出边不断访问下一个顶点



Gremlin 查询举例

```
gremlin> g.V(hercules).out('father', 'mother').label()  
=>god  
=>human  
gremlin> g.V(hercules).out('battled').label()  
=>monster  
=>monster  
=>monster
```



RDF

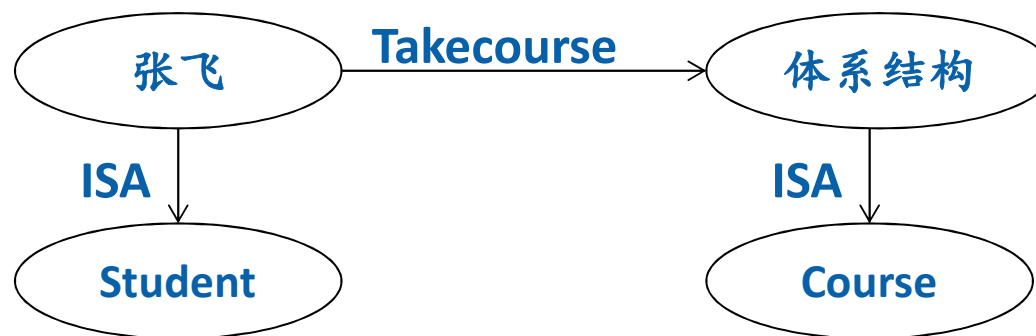
- RDF(Resource Description Framework)
 - W3C标准，广泛用于语义网络
- 每个RDF记录是三元组(subject 主, predicate 谓, object 宾)

□ 例如，

(张飞, Takecourse, 体系结构)

(张飞, ISA, Student)

(体系结构, ISA, Course)



- subject和object都是图的顶点
- predicate表达了边的类型
- 多个RDF三元组表达一张图

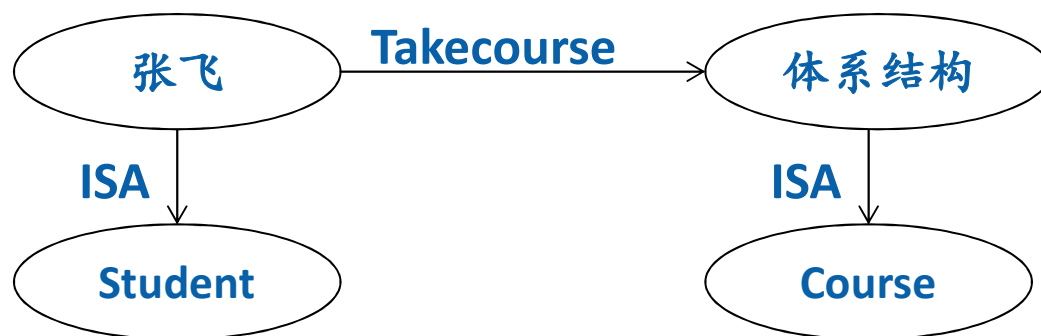
SPARQL

- SPARQL（读作sparkle）：RDF的查询语言

- 类似SQL Select语句
- where语句规定一个子图模板
- ?前缀代表变量，注意“.”

```
SELECT ?s
WHERE { ?s Takecourse 体系结构 .
       ?s ISA Student .
}
```

Select结果：张飞



小结

- Document Store

- 树状结构数据模型
 - JSON
 - Google Protocol Buffers
- MongoDB
 - API and Query Model
 - Architecture

- 图存储系统(Graph Database)

- 图数据模型
- Neo4j
- JanusGraph
- RDF和SPARQL

课后问题

1. 请比较JSON和Protocol Buffers的异同
2. Neo4j是如何在外存存储图数据的？