

大数据时代的知识图谱数据管理

彭鹏

hnu16pp@hnu.edu.cn

-
- 而现有RDF数据图规模越来越大，它们中很多已经超出现有单机系统的处理能力



1亿条知识



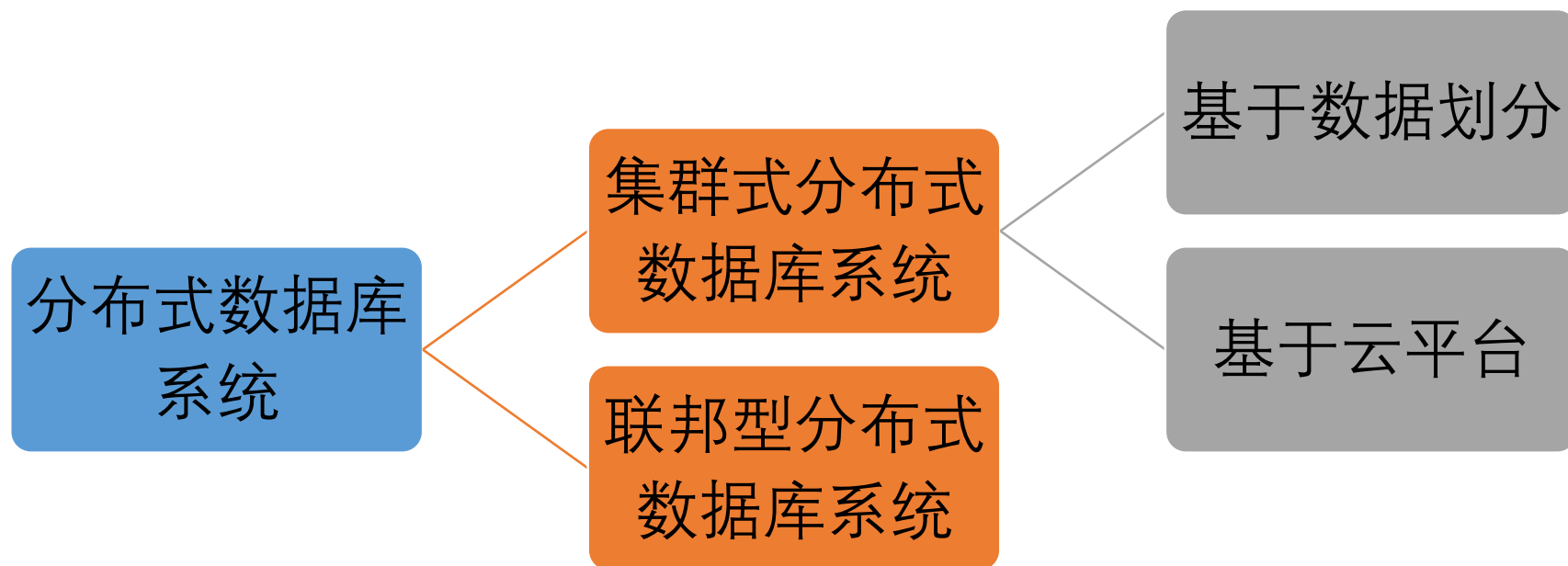
24亿条知识



30亿条知识

利用分布式数据管理技术进行SPARQL查询处理日益紧迫！

分布式数据库系统



集群式分布式知识图谱管理方法

- 基于云平台的分布式知识图谱管理方法
 - 基于Hadoop的分布式知识图谱管理方法
 - 基于其他云平台的分布式知识图谱管理方法
- 基于数据划分的分布式知识图谱管理方法
 - 基于粗化（coarsening）的知识图谱划分方法
 - 基于局部模式的知识图谱划分方法

基于云平台的分布式知识图谱管理方法

- 这类方法都是利用已有云平台存储系统进行知识图谱数据的存储，并利用已有云平台上成熟的任务处理模式处理知识图谱数据任务
- 被最多地利用的云平台系统就是Hadoop；其他的云平台系统如Trinity、Spark等也有相关应用

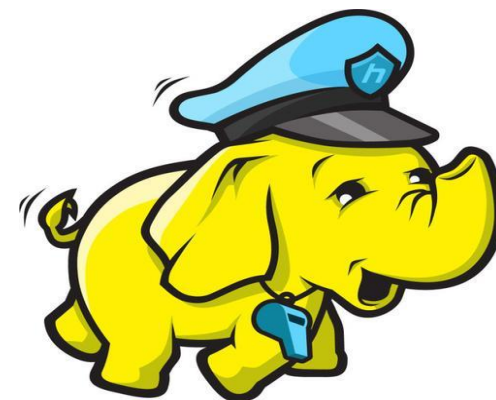
Hadoop概述

- Hadoop是一个开源的、可靠的、可扩展的分布式并行计算框架
- 主要组成:分布式文件系统HDFS和MapReduce算法执行
- 作者：Doug Cutting
- 语言：Java，支持多种编程语言，如:Python、C++
- 系统平台：Linux



Hadoop起源

- Hadoop是Google的集群系统的开源实现
 - Google集群系统：GFS(Google File System)、MapReduce、BigTable
 - Hadoop主要由HDFS(Hadoop Distributed File System Hadoop分布式文件系统)、MapReduce和HBase组成
- Hadoop的初衷是为了解决 Nutch 的海量数据爬取和存储的需要
- Hadoop于2005年秋天作为 Lucene的子项目Nutch的一部分正式引入Apache基金会。
- 名称起源: Doug Cutting儿子的黄色大象玩具的名字



Hadoop优势

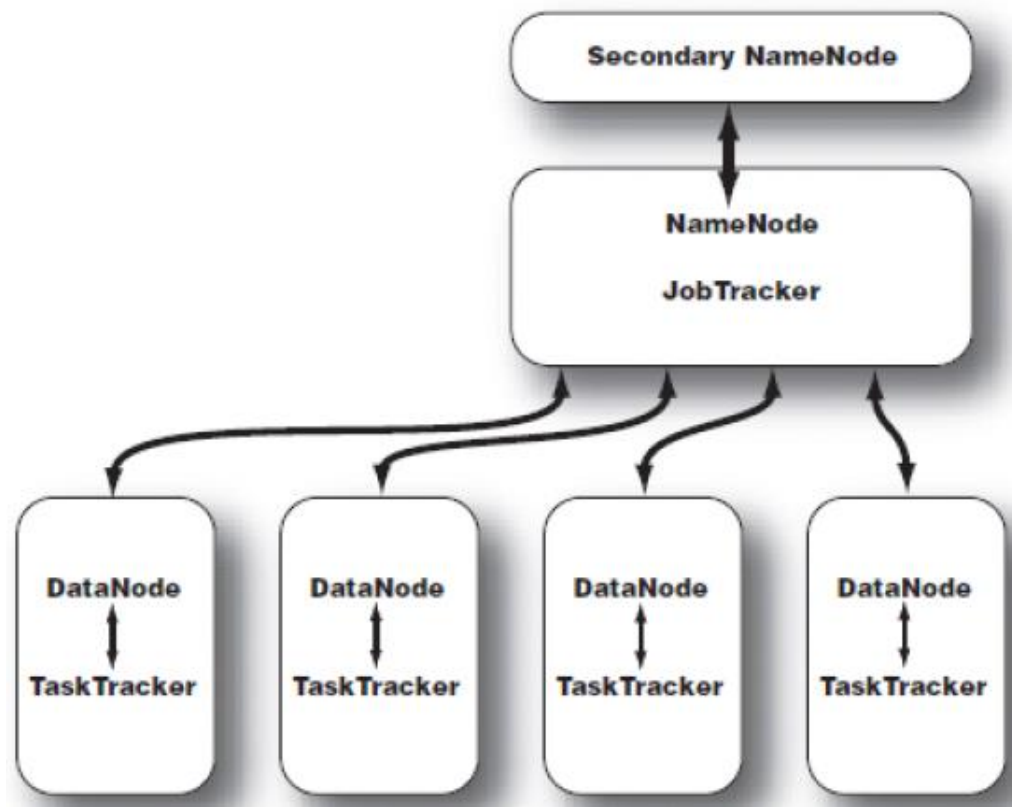
- **可扩展**：不论是存储的可扩展还是计算的可扩展都是Hadoop的设计根本；
- **经济**：它在通常可用的计算机集群间分配数据和处理，这些集群可以被计入数以千计的节点当中
- **高效**：通过分配数据，Hadoop能够在存放数据的节点之间平行的处理它们，因此其处理速度非常快。
- **可信**：Hadoop能够自动保存数据的多份副本，并且能够自动地将失败的任务重新分配

Hadoop概述

- Hadoop是一个分布式系统基础架构，是一个能够对大量数据进行分布式处理的软件框架，由Apache基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序，充分利用集群的的威力高速运算和存储。

Hadoop框架中最核心的设计就是：**MapReduce**和**HDFS**。

Hadoop架构



Hadoop分布式文件系统（HDFS）

- Hadoop也跟其他云计算项目有共同点和目标：实现海量数据的计算。而进行海量计算需要一个稳定的、安全的数据容器，于是就有了Hadoop分布式文件系统（HDFS）

MapReduce分布式计算

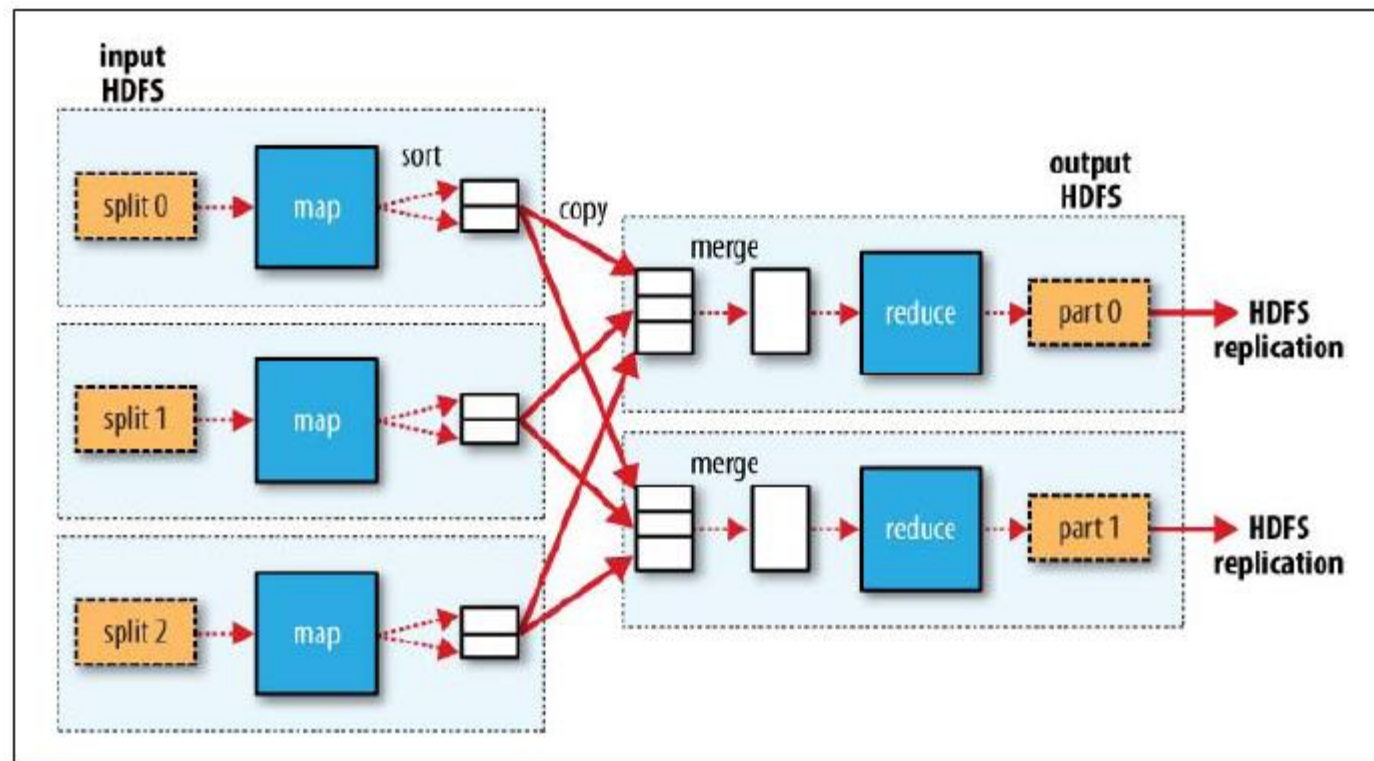
- Hadoop MapReduce是一种简单易用的软件框架，可以开发出运行在由上千个商用机器组成的大型机器上，并以一种可靠容错的方式并行处理的数据集（太字节级的数据）

MapReduce



- 对于任务处理的两步：
- 一个MapReduce作业（Job）通常会把输入集切分成若干独立的数据块，由Map任务（Task）以完全并行的方式处理它们
- MapReduce框架会先排序map任务的输出，然后把结果输入到reduce任务

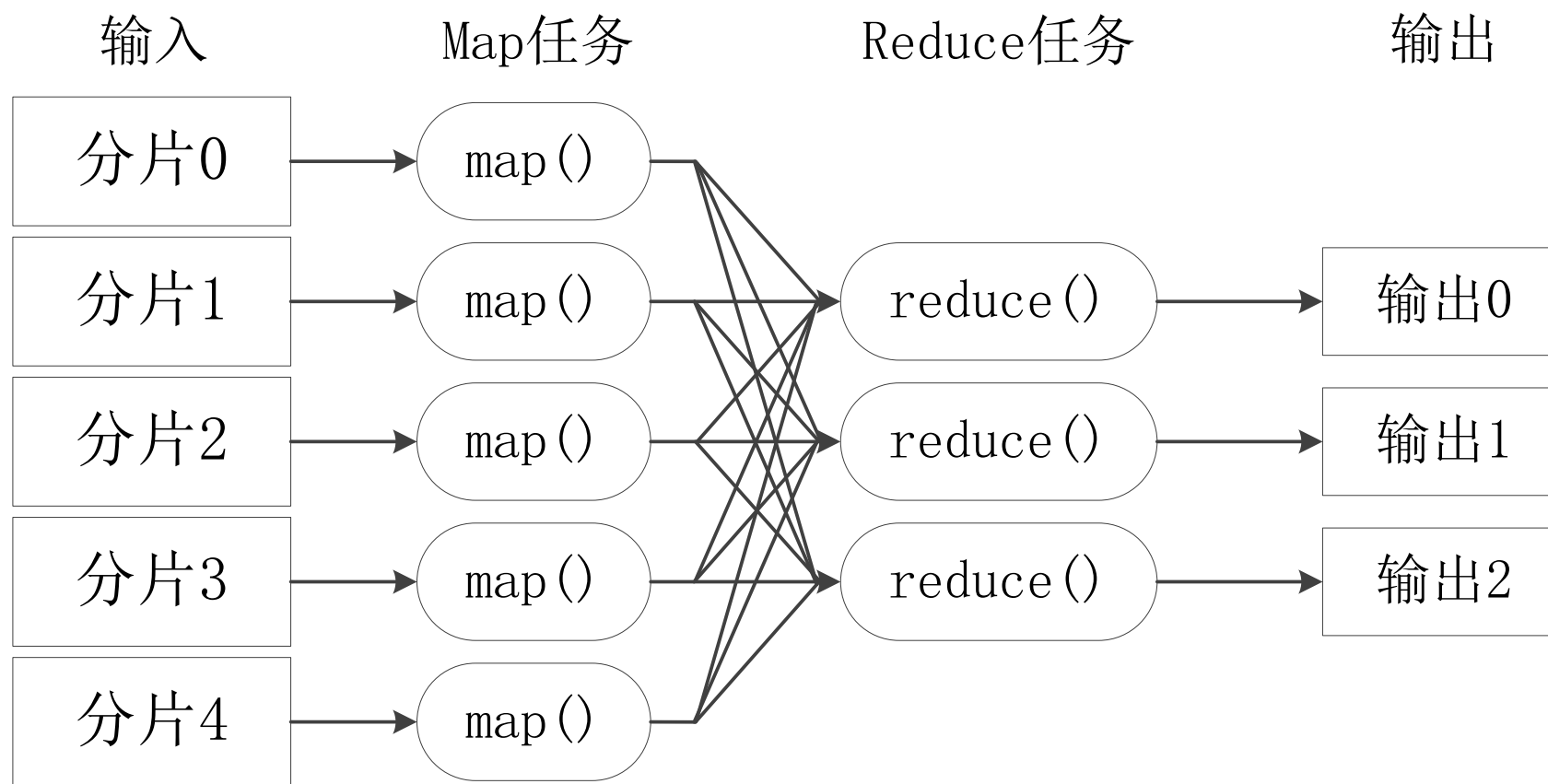
MapReduce处理



MapReduce模型简介

- MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce
- MapReduce采用“分而治之”策略，一个存储在分布式文件系统的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理
- MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销
- MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave。Master上运行JobTracker，Slave上运行TaskTracker

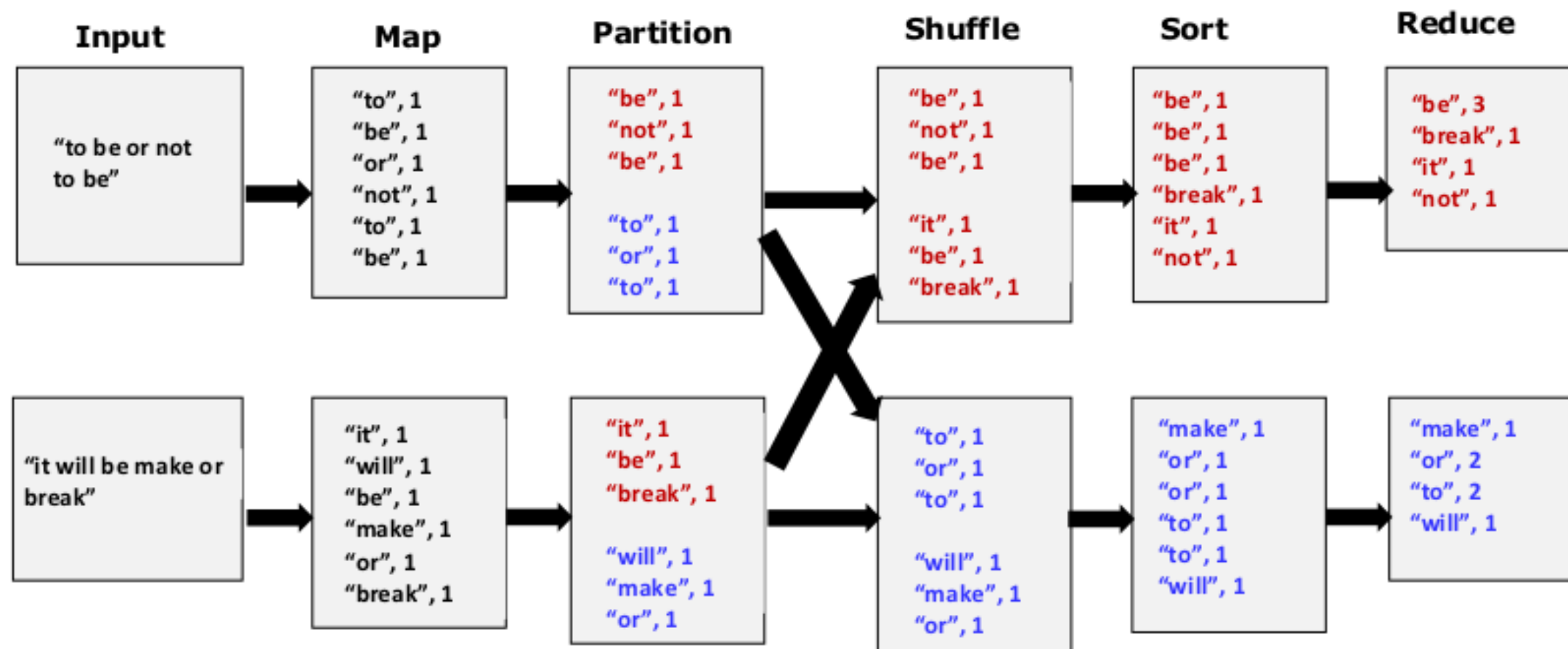
MapReduce计算框架



Map和Reduce函数

函数	输入	输出	说明
Map	$\langle k_1, v_1 \rangle$ 如: $\langle \text{行号}, \text{"a b c"} \rangle$	$\text{List}(\langle k_2, v_2 \rangle)$ 如: $\langle \text{"a"}, 1 \rangle$ $\langle \text{"b"}, 1 \rangle$ $\langle \text{"c"}, 1 \rangle$	1.将小数据集进一步解析成一批 $\langle \text{key}, \text{value} \rangle$ 对, 输入Map函数中进行 处理 2.每一个输入的 $\langle k_1, v_1 \rangle$ 会输出一批 $\langle k_2, v_2 \rangle$ 。 $\langle k_2, v_2 \rangle$ 是计算的中间结果
Reduce	$\langle k_2, \text{List}(v_2) \rangle$ 如: $\langle \text{"a"}, \langle 1, 1, 1 \rangle \rangle$	$\langle k_3, v_3 \rangle$ $\langle \text{"a"}, 3 \rangle$	输入的中间结果 $\langle k_2, \text{List}(v_2) \rangle$ 中的 $\text{List}(v_2)$ 表示是一批属于同一个 k_2 的 value

MapReduce举例



基于Hadoop的分布式知识图谱管理方法

- 这一类方法又可细分为两类
 - 基于三元组的方法，如SHARD
 - 基于图的方法，如EAGRE

SHARD

- SHARD以知识图谱数据中的主体为核心进行数据划分，与一个主体相关的所有三元组被聚集起来并被存储为HDFS 文件中的一行



Aristotle	influencedBy	Plato
Aristotle	mainInterest	Ethics
Aristotle	name	"Aristotle"
Aristotle	placeOfDeath	Chalcis

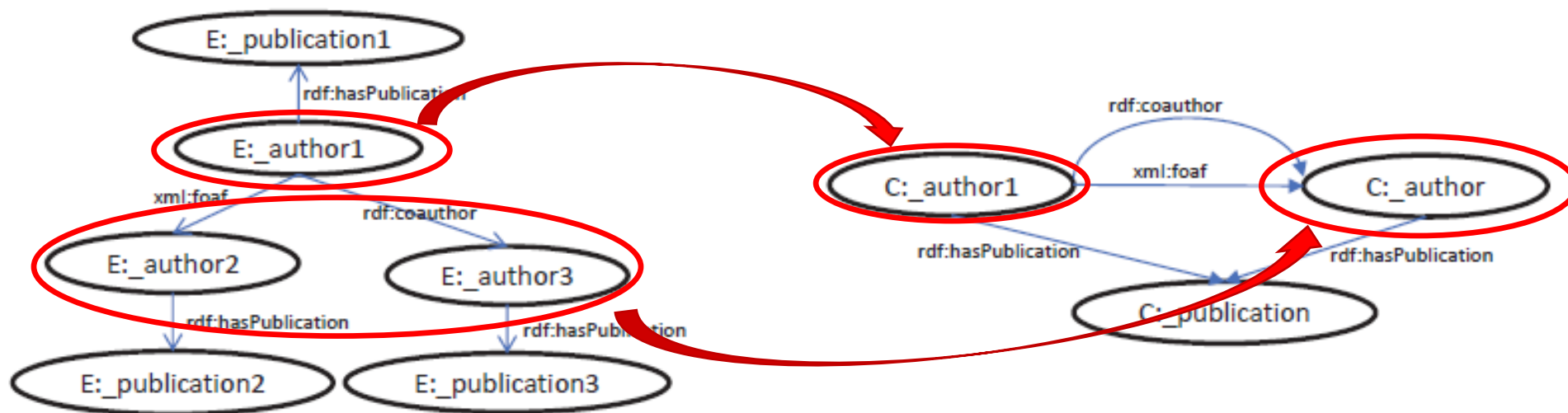
Aristotle influencedBy Plato mainInterest Ethics name "Aristotle" placeOfDeath Chalcis



HDFS

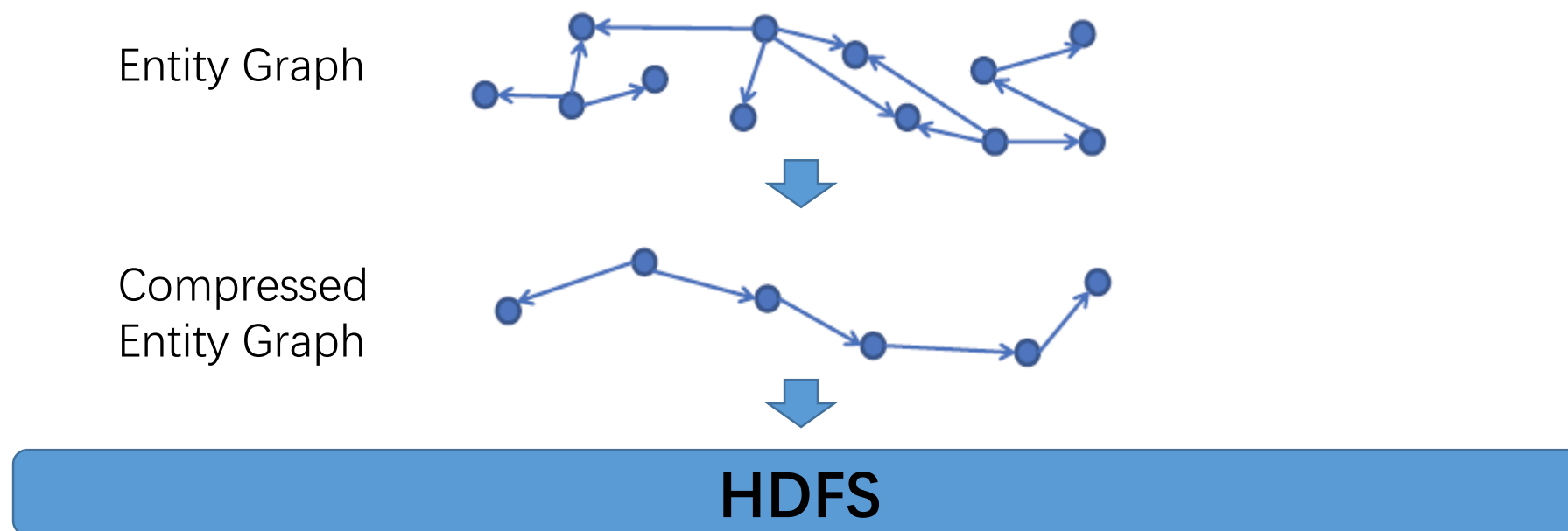
EAGRE

- EAGRE 将知识图谱中相似的实体聚类成一个实体类，进而形成一个压缩实体图



EAGRE

- 上述压缩实体图存储在内存中，并利用METIS 进行图划分，然后根据划分结果将知识图谱中实体以及相应三元组放到不同机器上

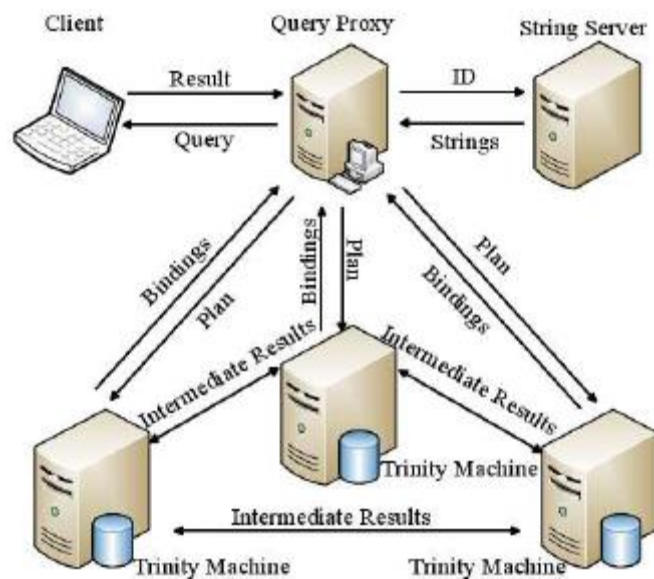


基于其他云平台的分布式知识图谱管理方法

- 基于HBase 的H2RDF
- 基于Trinity 的Trinity.RDF
- 基于Spark 的S2RDF

Trinity.RDF

- Trinity是微软开发的一个分布式内存的图数据管理系统
- 知识图谱可以直接以图的方式存在Trinity上

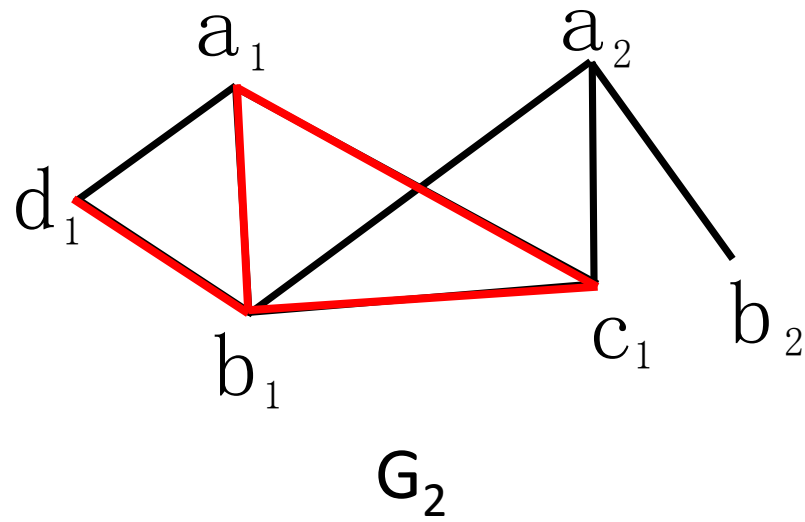
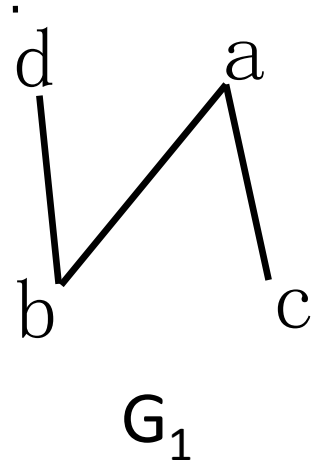


K. Zeng, J. Yang, H. Wang, B. Shao and Z. Wang, A Distributed Graph Engine for Web Scale RDF Data, PVLDB 6 (4), 265–276 (2013).

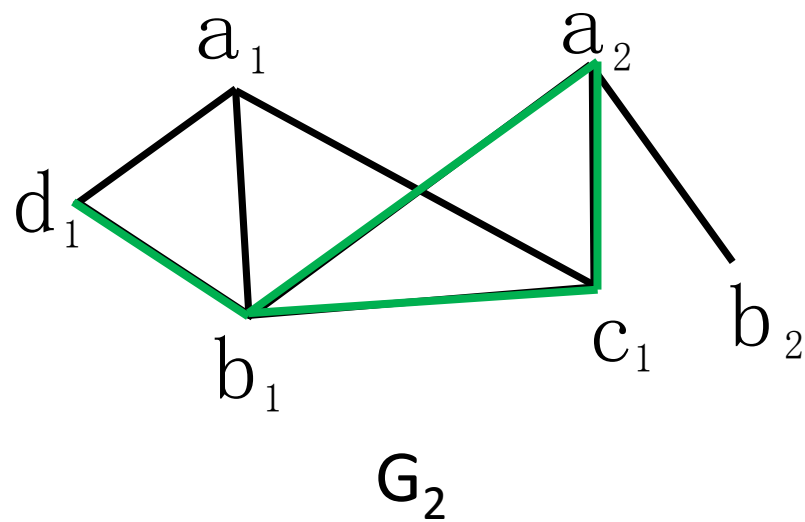
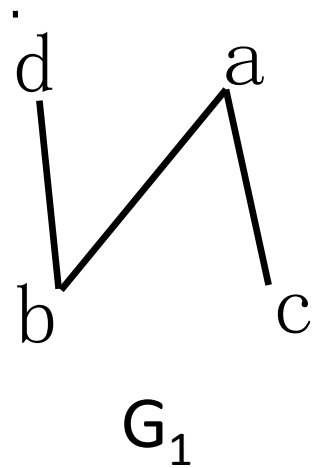
查询处理流程

1. 将图查询分解为小的基本单元
2. 并行地在数据图中匹配这些基本单元
3. 联结所得结果

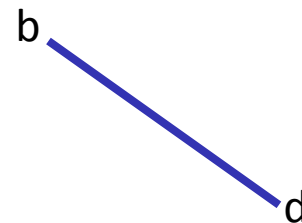
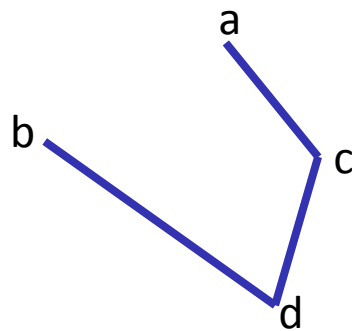
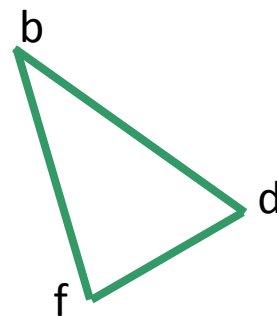
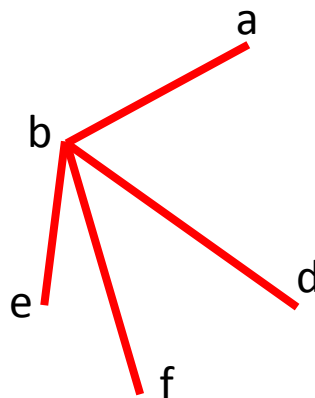
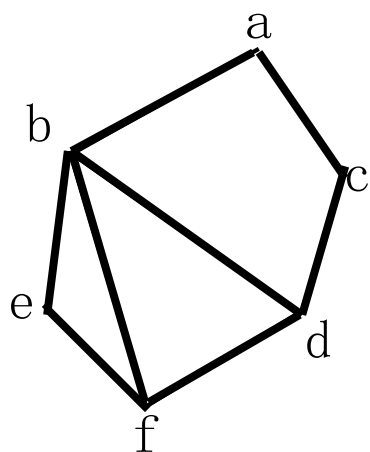
子图匹配



子图匹配

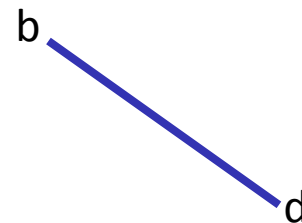
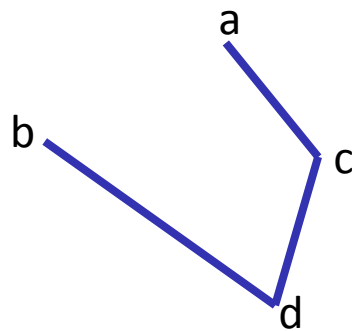
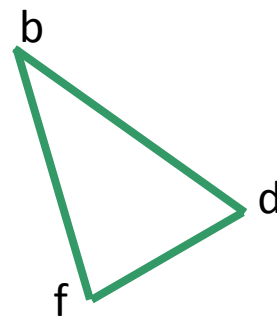
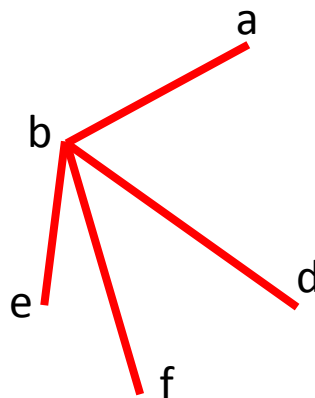
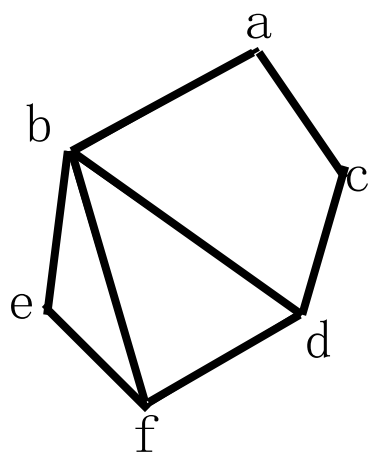


分布式子图匹配的基本单元



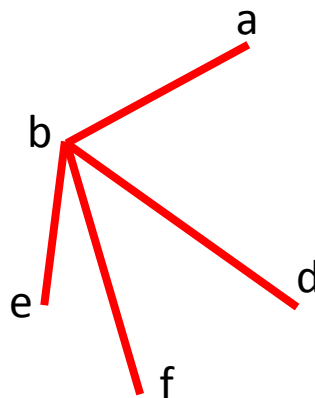
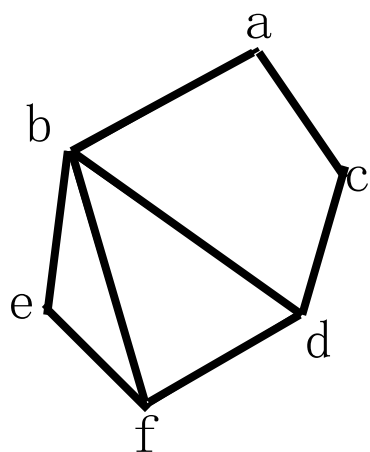
作为基本单元，哪一个才是最优的？

分布式子图匹配的基本单元



作为基本单元，哪一个才是最优的？

分布式子图匹配的基本单元

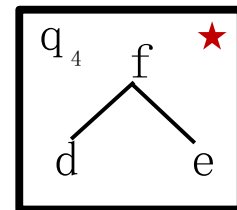
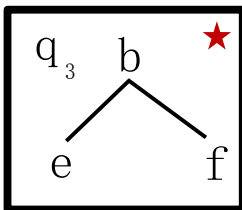
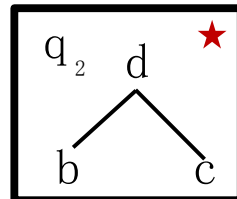
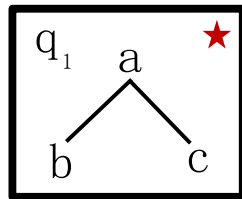
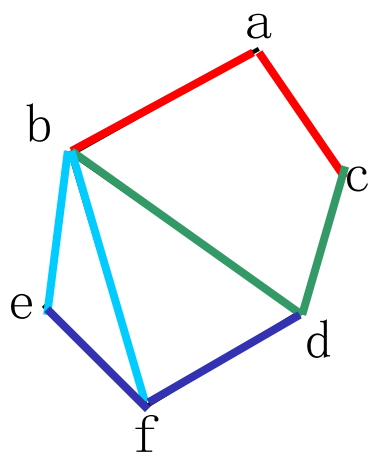


Twig

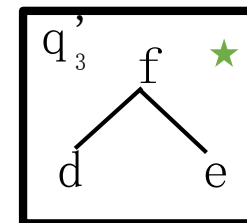
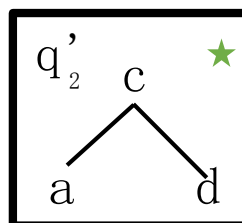
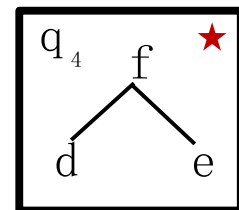
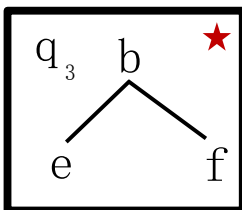
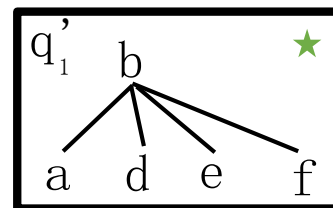
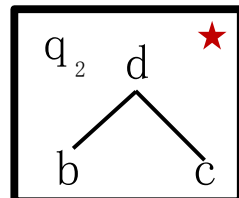
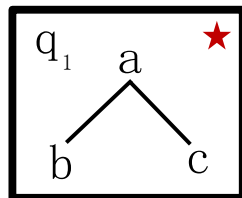
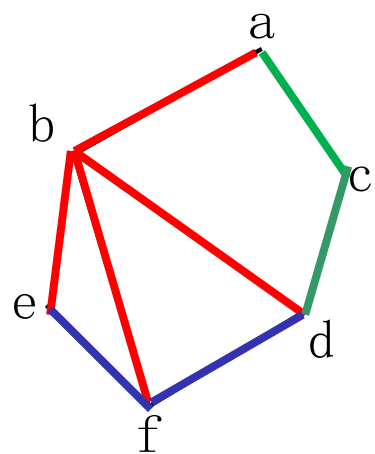
- 容易拆解
- 高度为一
- 最多一次跨网络访问

作为基本单元，哪一个才是最优的？

查询分解



查询分解

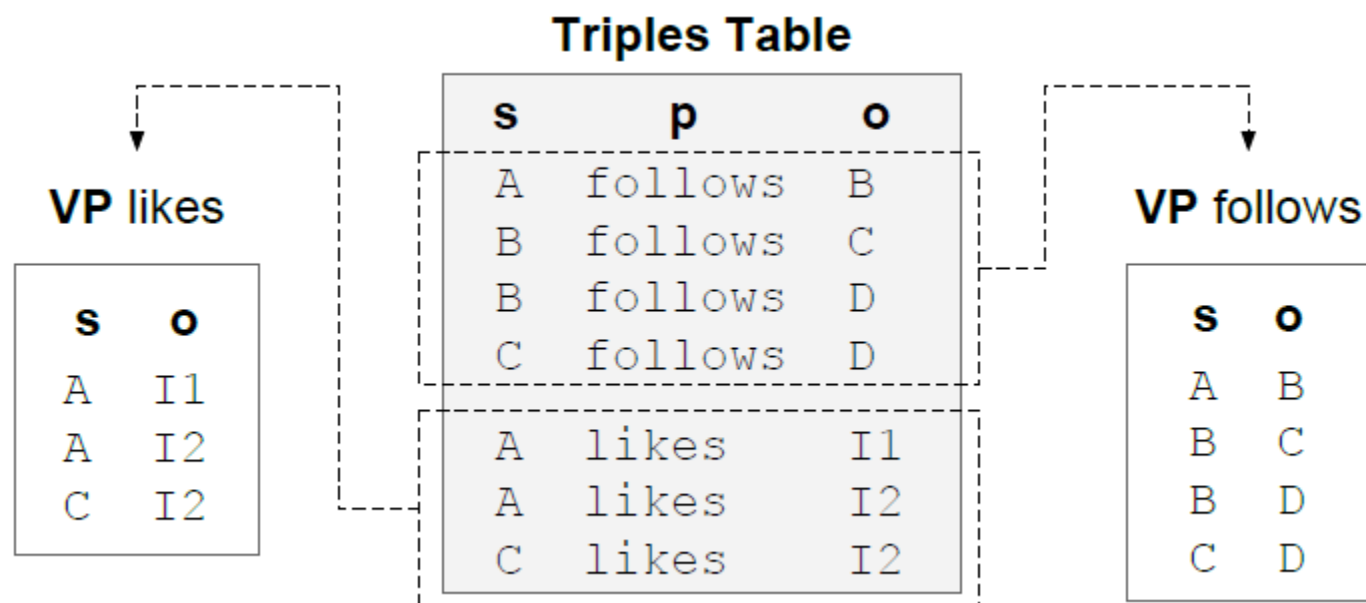


查询优化问题

- 查询分解方案生成
- 子查询执行顺序优化
- 查询结果联结顺序优化

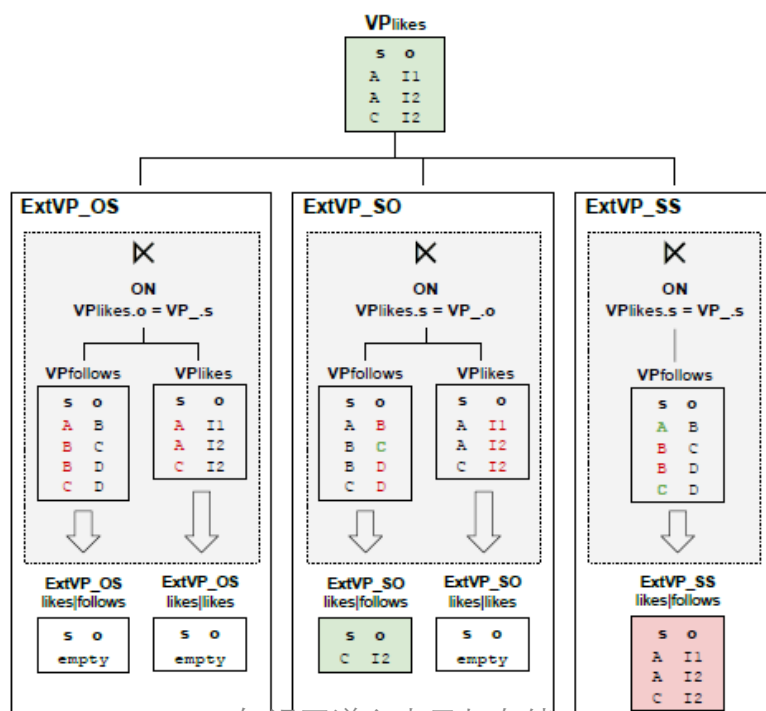
S2RDF

- S2RDF利用Spark的关系数据库接口进行知识图谱数据管理
- S2RDF利用垂直划分对知识图谱数据进行划分



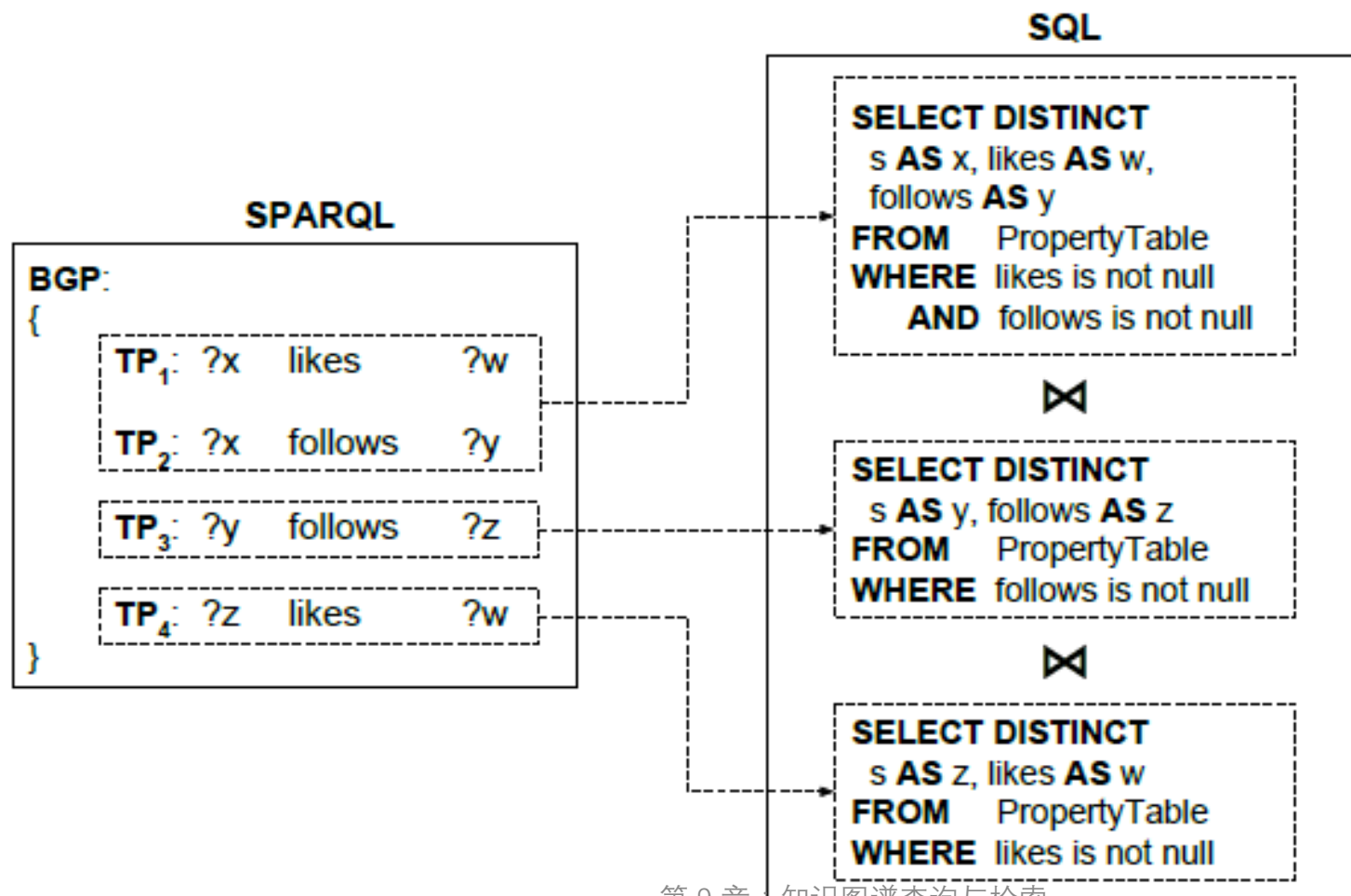
S2RDF

- 在基本垂直划分基础上，S2RDF物化了部分垂直划分数据表之间的连接结果并也存储在关系数据表



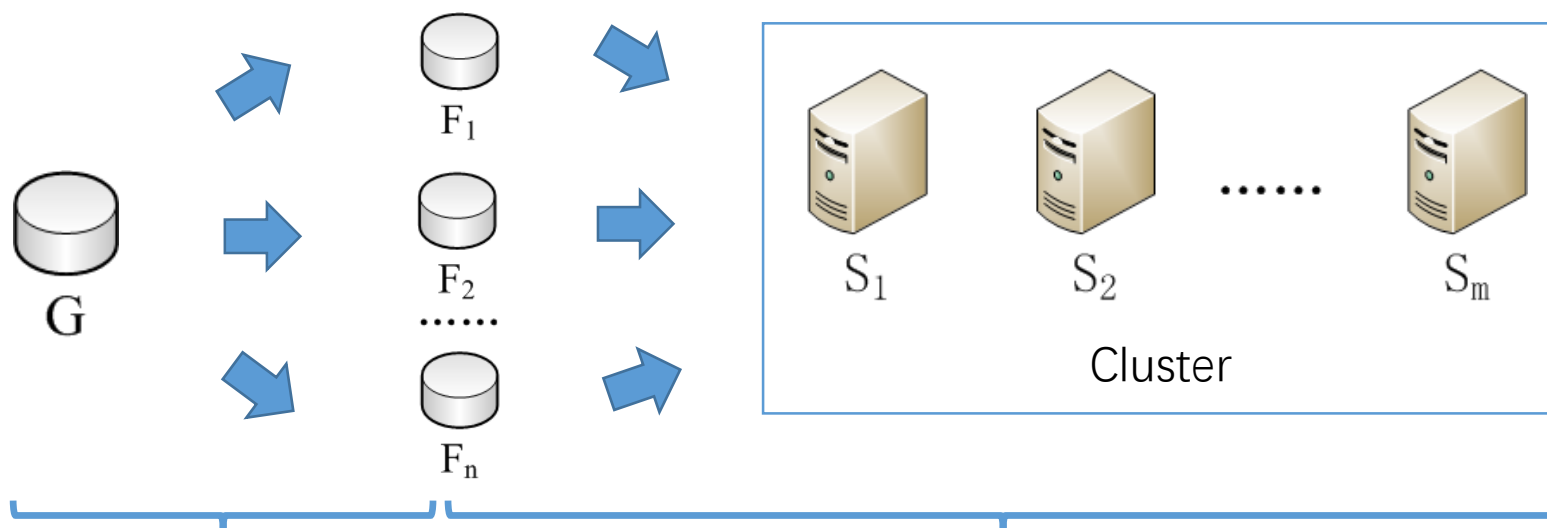
S2RDF查询处理

- 查询分解成基于垂直划分的子查询，并转化成SQL



基于数据划分的分布式知识图谱管理方法

- 第一步，系统要将数据按照一定的算法划分成若干分片；第二步，系统将划分出来的分片分配到不同机器上去
- 这些方法相互之间主要的不同在于数据划分方式不同



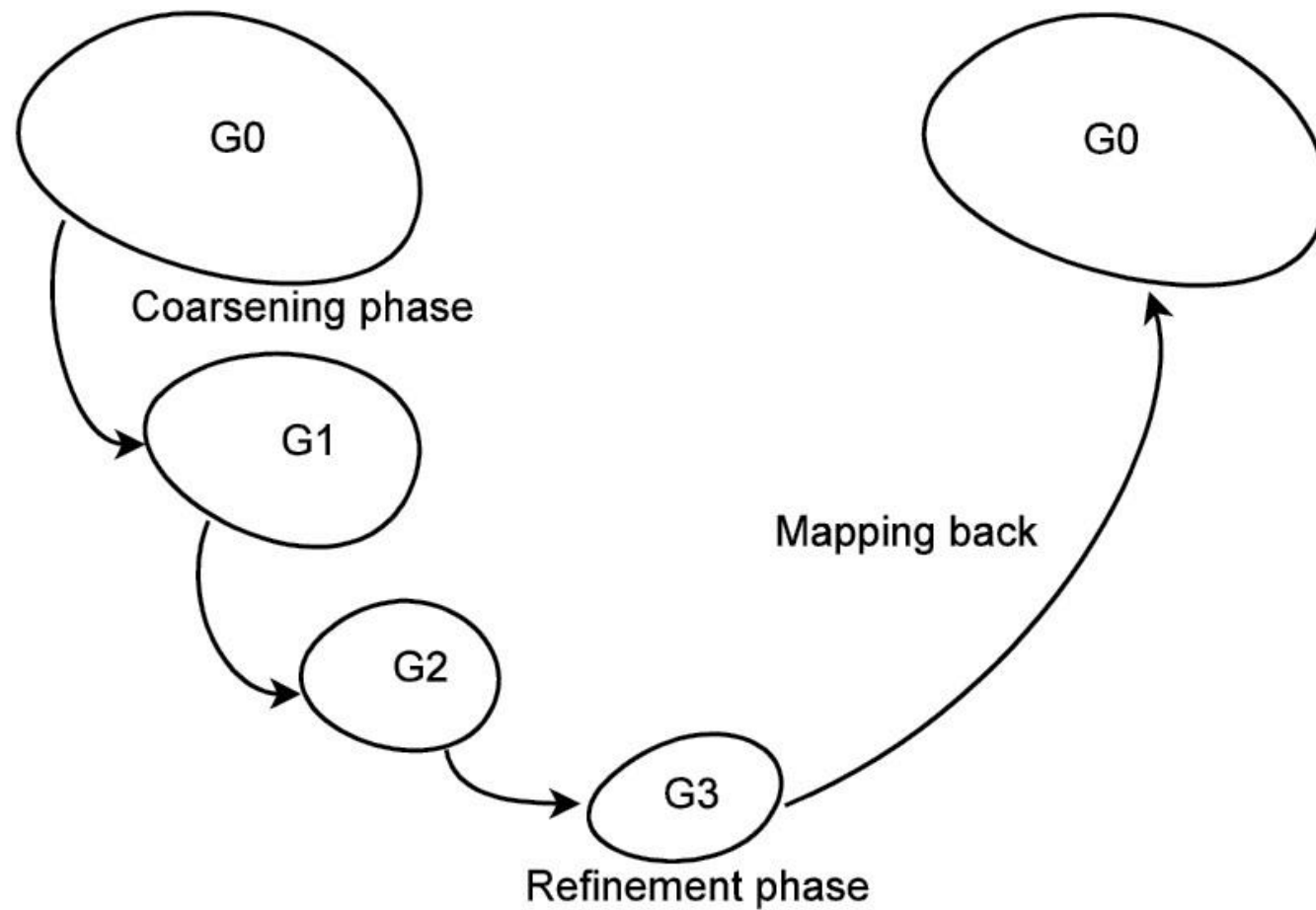
方法分类

- 基于粗化 (coarsening) 的划分方法, 如Huang et al.
- 基于局部模式的划分方法, 如SHAPE

基于粗化的划分方法

- 这类方法主要思想在于大的知识图谱中一些点迭代地合并起来形成能被处理的小图。然后对这个小图调用现有方法来进行划分，最后将对小图的划分结果回溯成大图的分
- 这个挑选点进行合并的过程称之为**粗化 (coarsening)**
- 现有方法主要区别在于粗化方式不一样

基于粗化的划分



Huang et al.

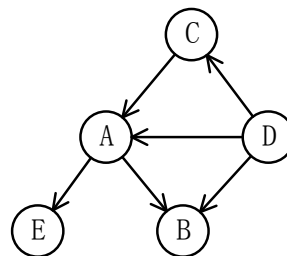
- 这个系统主要思想是将现有知识图谱数据图根据其结构信息进行水平划分
- 它对知识图谱的划分使用现有成熟工具METIS来实现。划分出来每个RDF 数据块对应一个数据分片
- 在此过程中，为了提高数据的局部性，每个块除了保存自身所有的点之外，还保存了它们的 n 步邻居以内所有点和边的副本

基于局部模式的划分方法

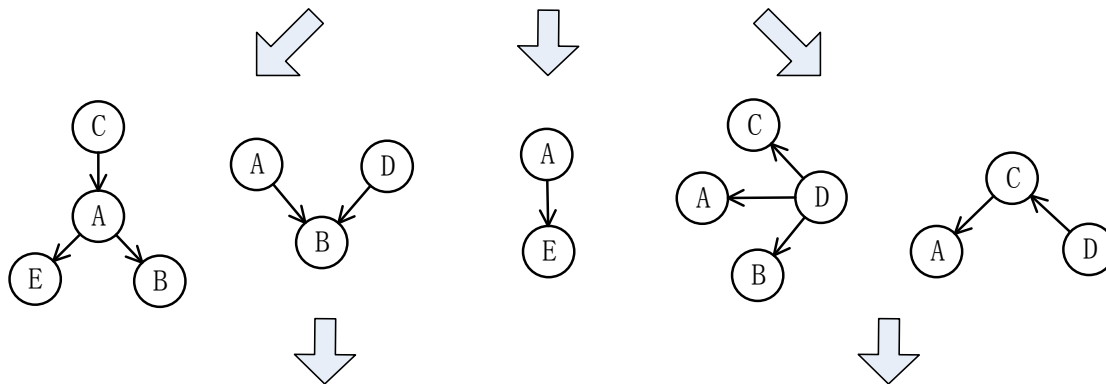
- 这类方法首先定义出若干模式，然后找出图上满足这些模式的子图，最后将这些子图按照模式划分成不同的分片
- 这类方法的关键在于如何定义模式

基于局部模式的划分方法

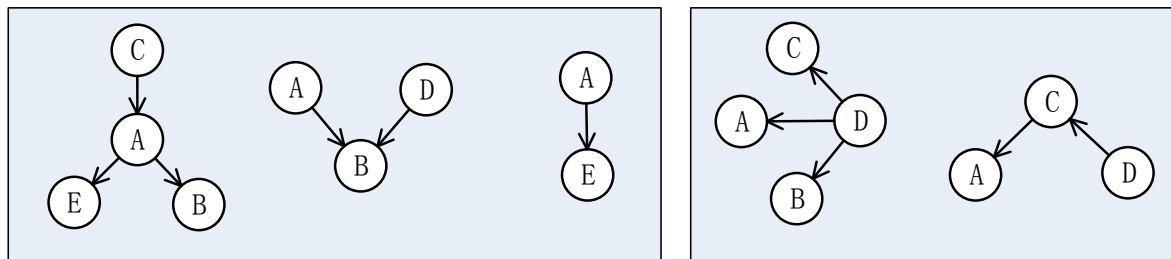
Original Graph



Patterns



Partitions

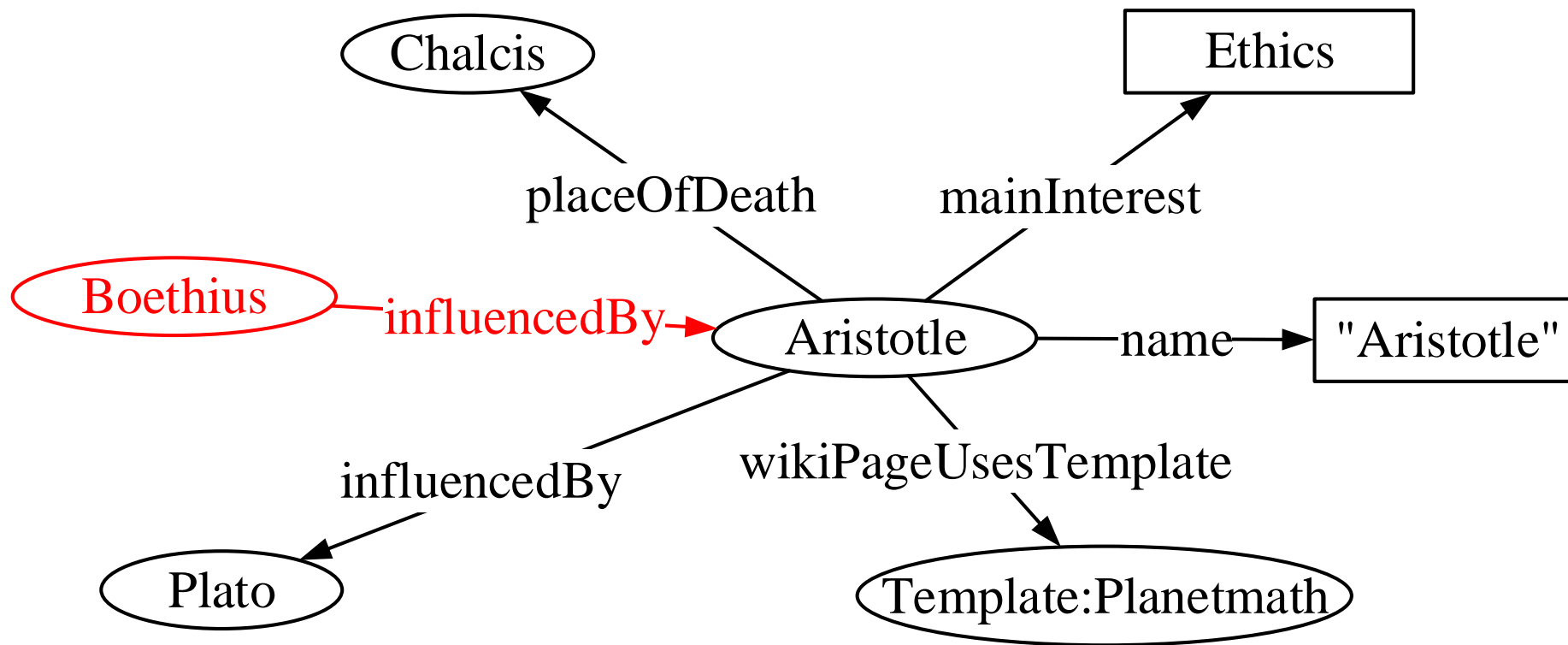


SHAPE



- SHAPE 中，在进行知识图谱数据划分的时候，SHAPE 划分的基本单元为一个三元组群 (Triple Group)
- 所谓三元组群，其实就是图上的星状结构
- SHAPE 提出了三种三元组群：只含中心点出边的三元组群、只含中心点入边的三元组群、含中心点出边与入边的三元组群

SHAPE

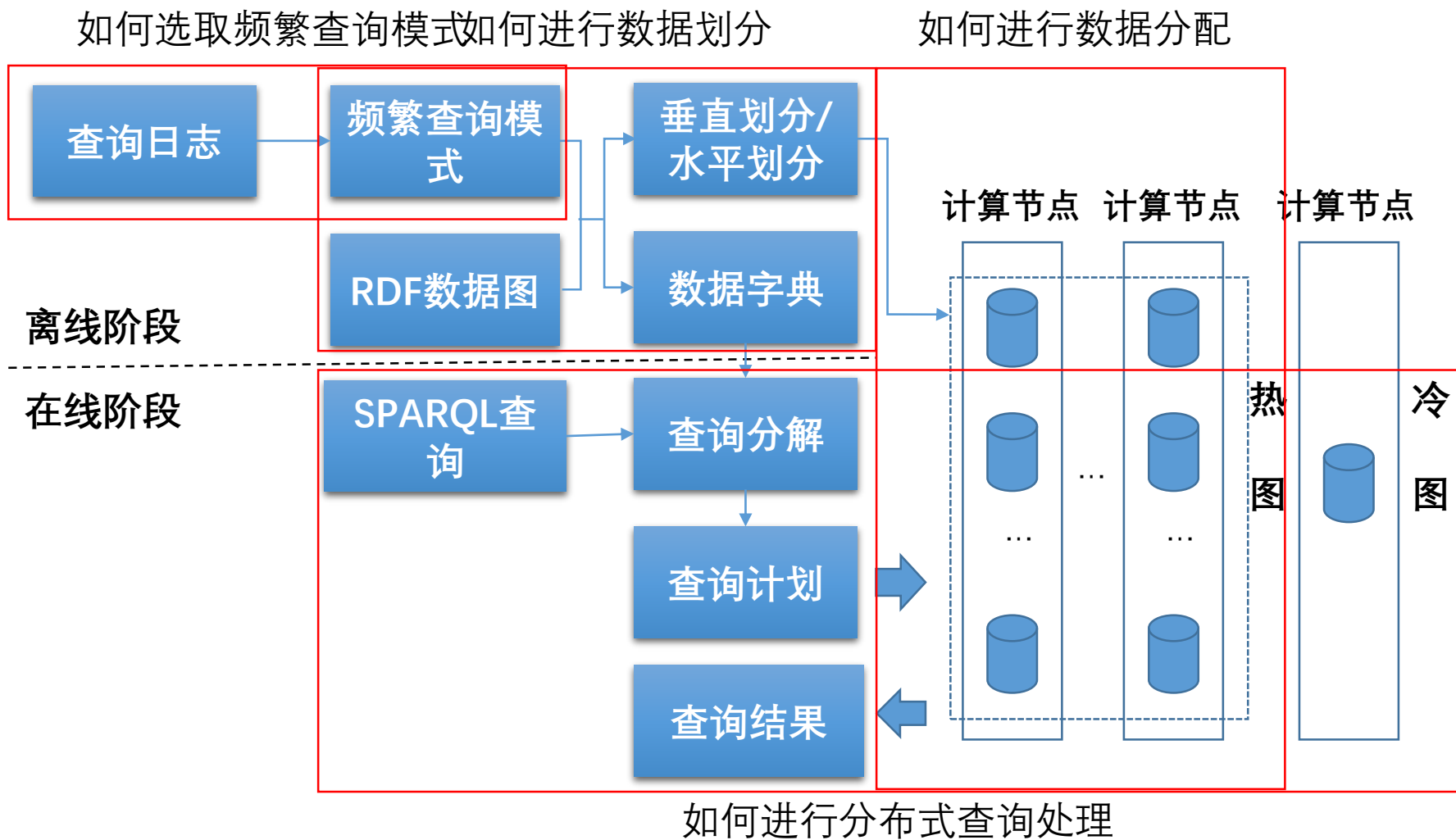


- 黑色部分是只含中心点出边的三元组群；红色部分是只含中心点入边的三元组群；整个就是含中心点出边与入边的三元组群

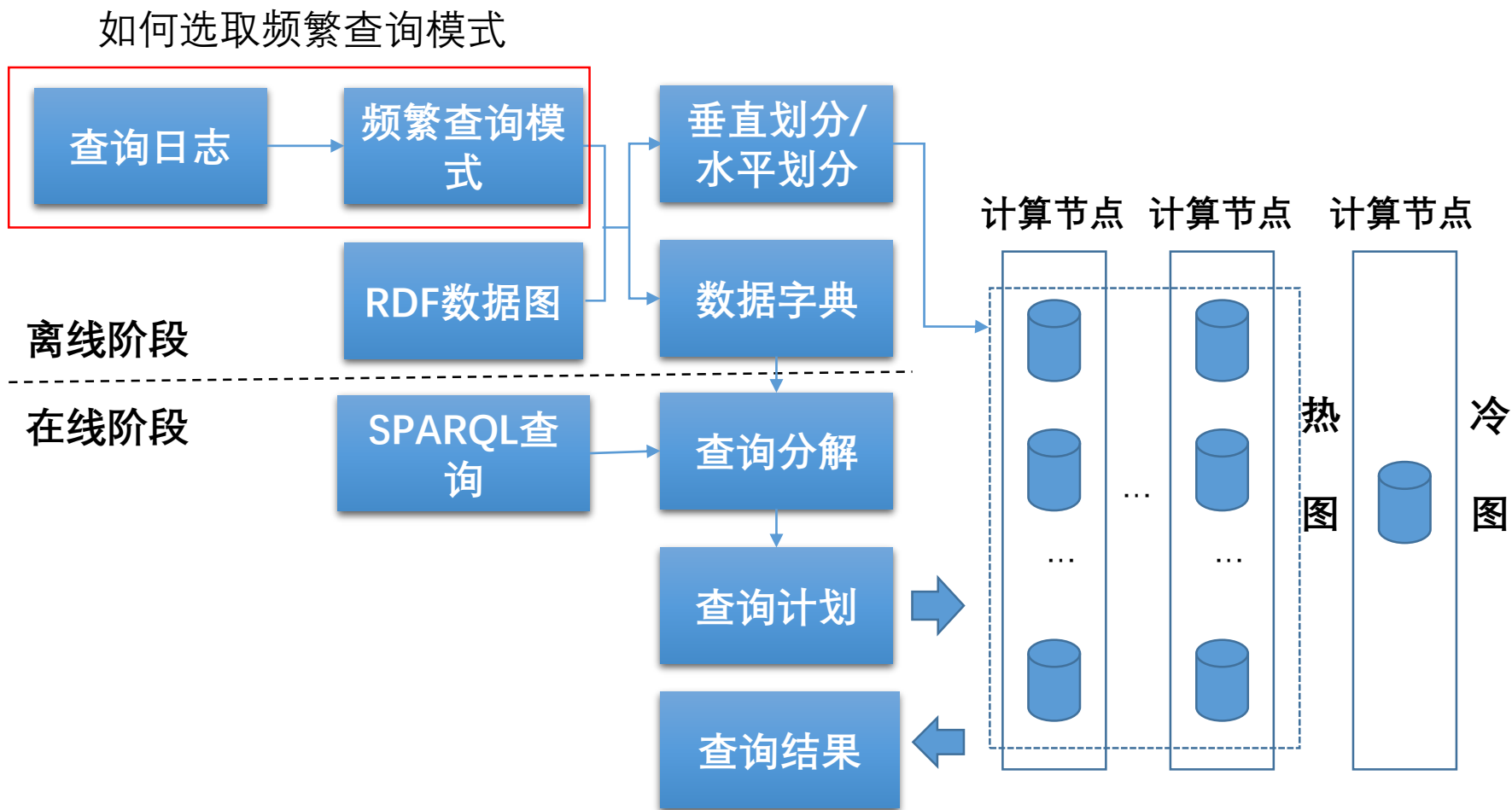
基于数据划分的分布式知识图谱管理方法

- Peng Peng, Lei Zou, Lei Chen, Dongyan Zhao: Query Workload-based RDF Graph Fragmentation and Allocation. EDBT 2016: 377-388
- Peng Peng, Lei Zou, Lei Chen, Dongyan Zhao: Adaptive Distributed RDF Graph Fragmentation and Allocation based on Query Workload. IEEE Trans. Knowl. Data Eng. 31(4): 670-685 (2019)

方法框架



方法框架

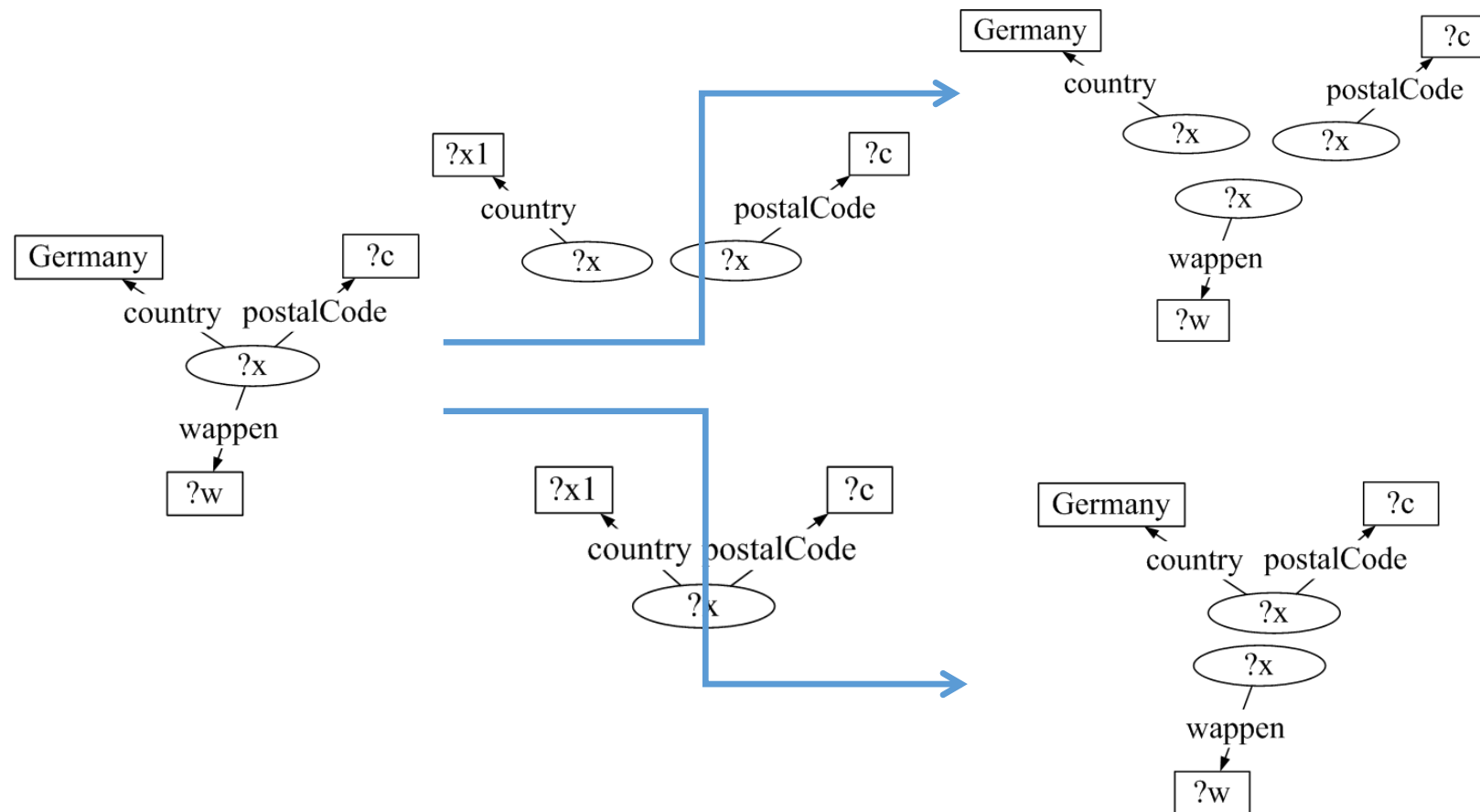



频繁查询模式

- 一个查询模式 p 的 **读取频率** 记为查询日志中以 p 为子图的查询的数量
- 如果一个查询模式 p 的读取频率大于一个用户事先给定的阈值 minSup , 那么是 p 就被视为**频繁查询模式**

频繁查询模式选取

- 越大的频繁查询模式越应该被选来进行数据划分





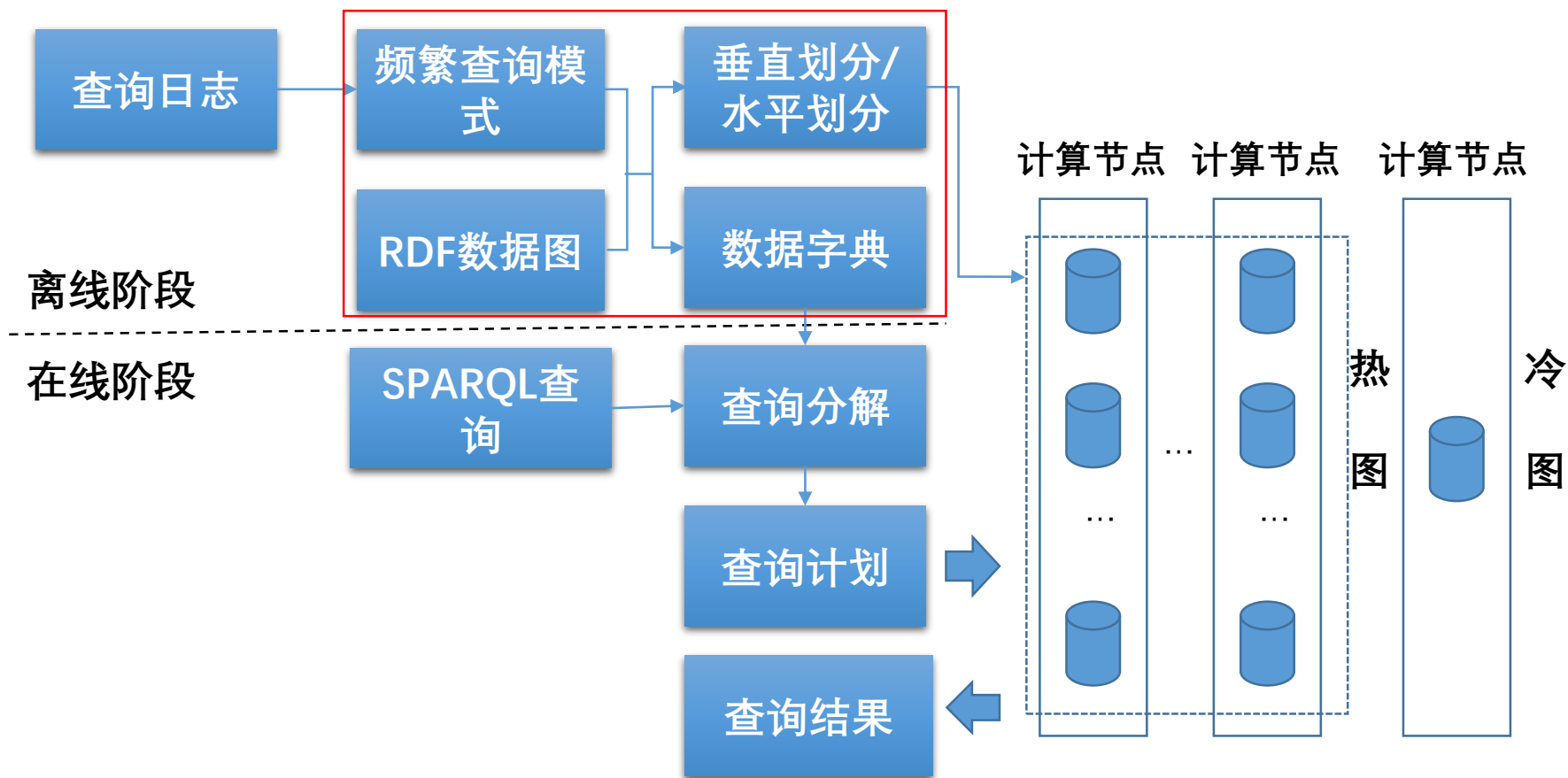
- 选取合适的频繁查询模式这个问题可以形式化为在满足存储空间限制的情况下找出选取收益最大的频繁查询模式集合

- 这个问题是NP-hard^[1], 于是本文提出一个基于贪心算法的启发式算法来解决这个问题。这个算法保证了数据完整性同时近似率

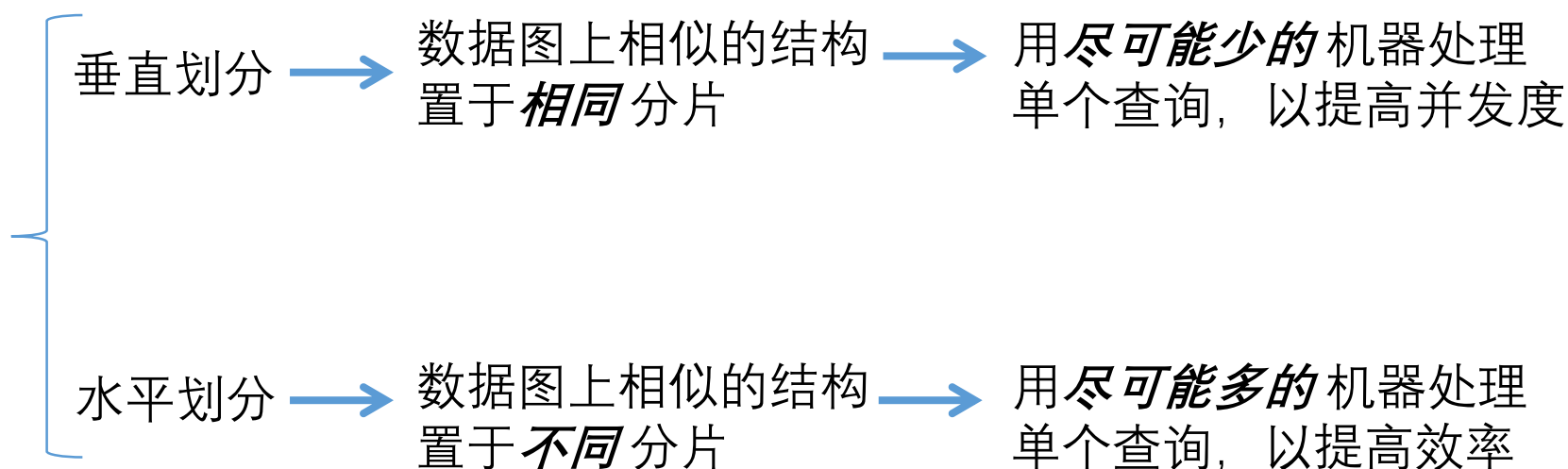
为 $\min\{\frac{1}{\max_{p \in P} |E(p)|}, \frac{1}{2}(1 - \frac{1}{e})\}$

方法框架

如何进行数据划分

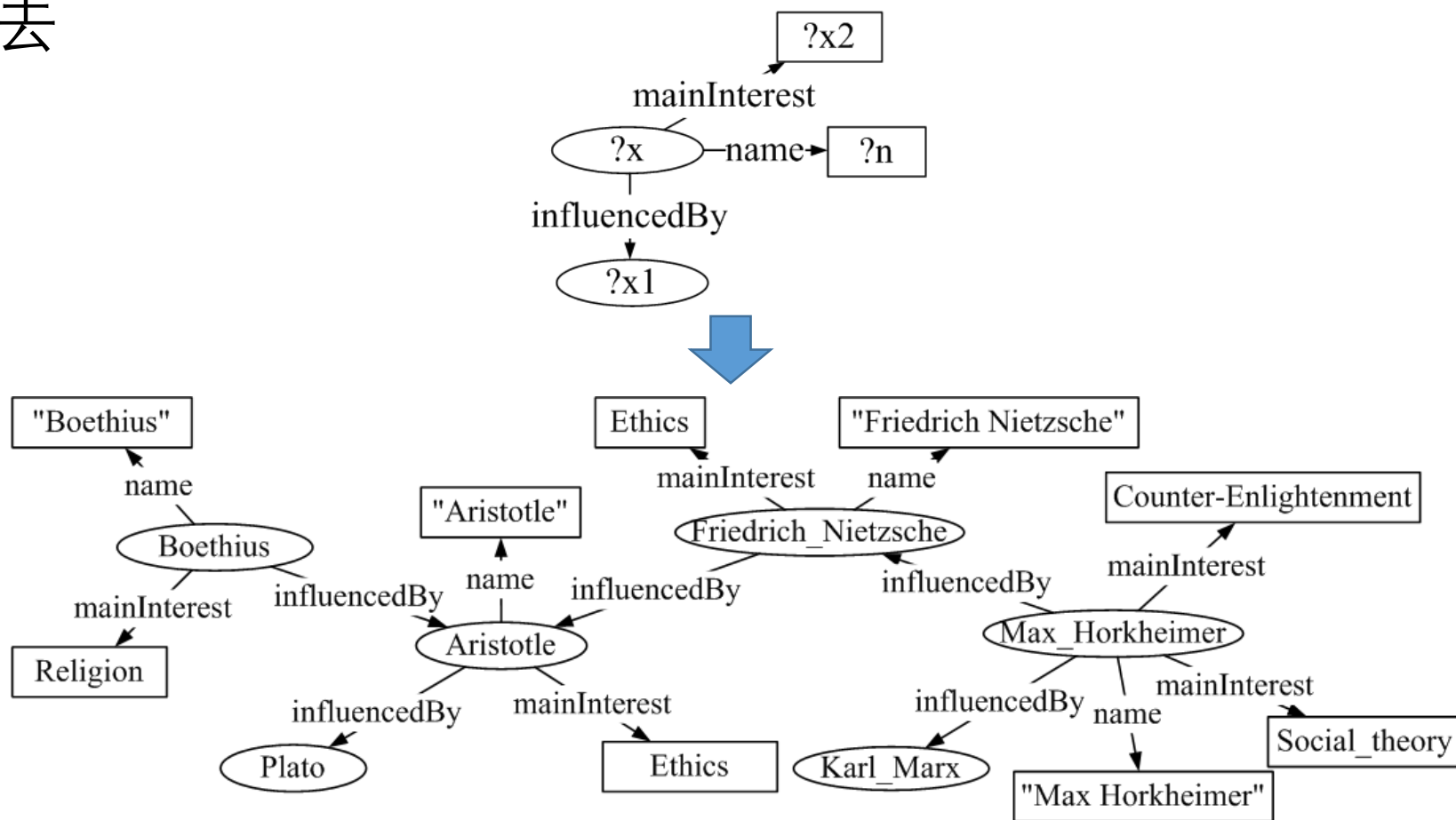


数据划分



垂直划分

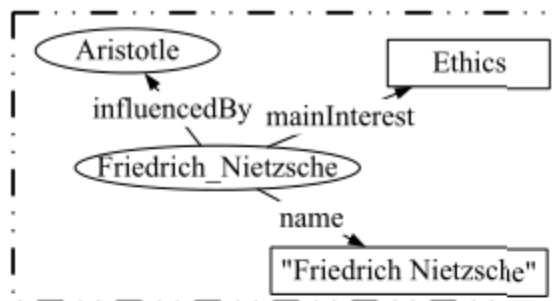
- RDF 数据图上所有同态于相同频繁查询模式的匹配被划分到相同分片中去



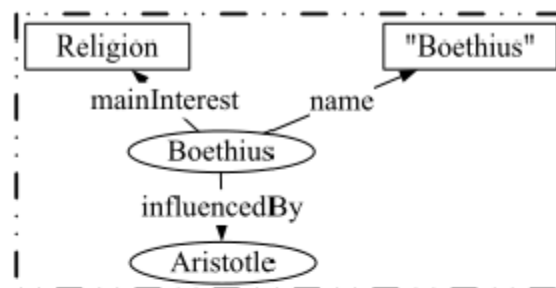
水平划分

- RDF 数据图上所有同态于相同频繁查询模式的匹配被划分到不同分片中去
- 我们扩展分布式关系数据库中的“简单谓词”和“小项谓词”的概念^[2]来进行 RDF 数据的水平划分

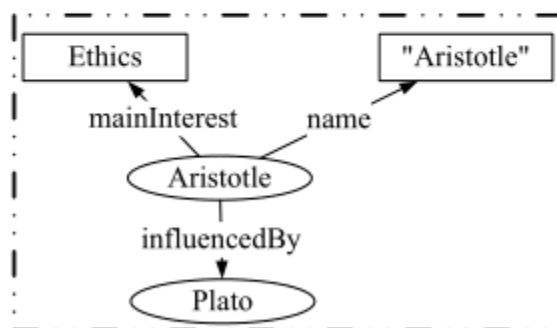
水平划分示例



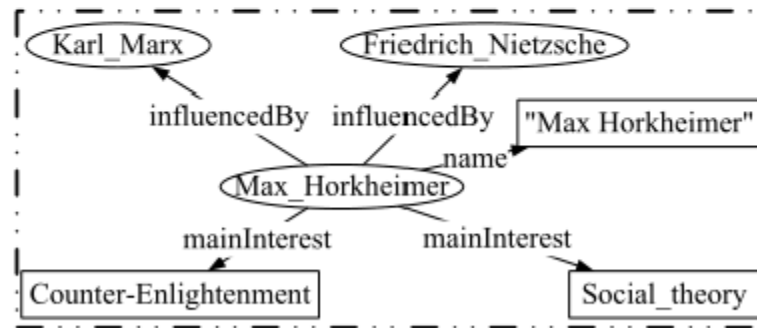
(a) Example Horizontal Fragment Generated from mp_1



(b) Example Horizontal Fragment Generated from mp_2

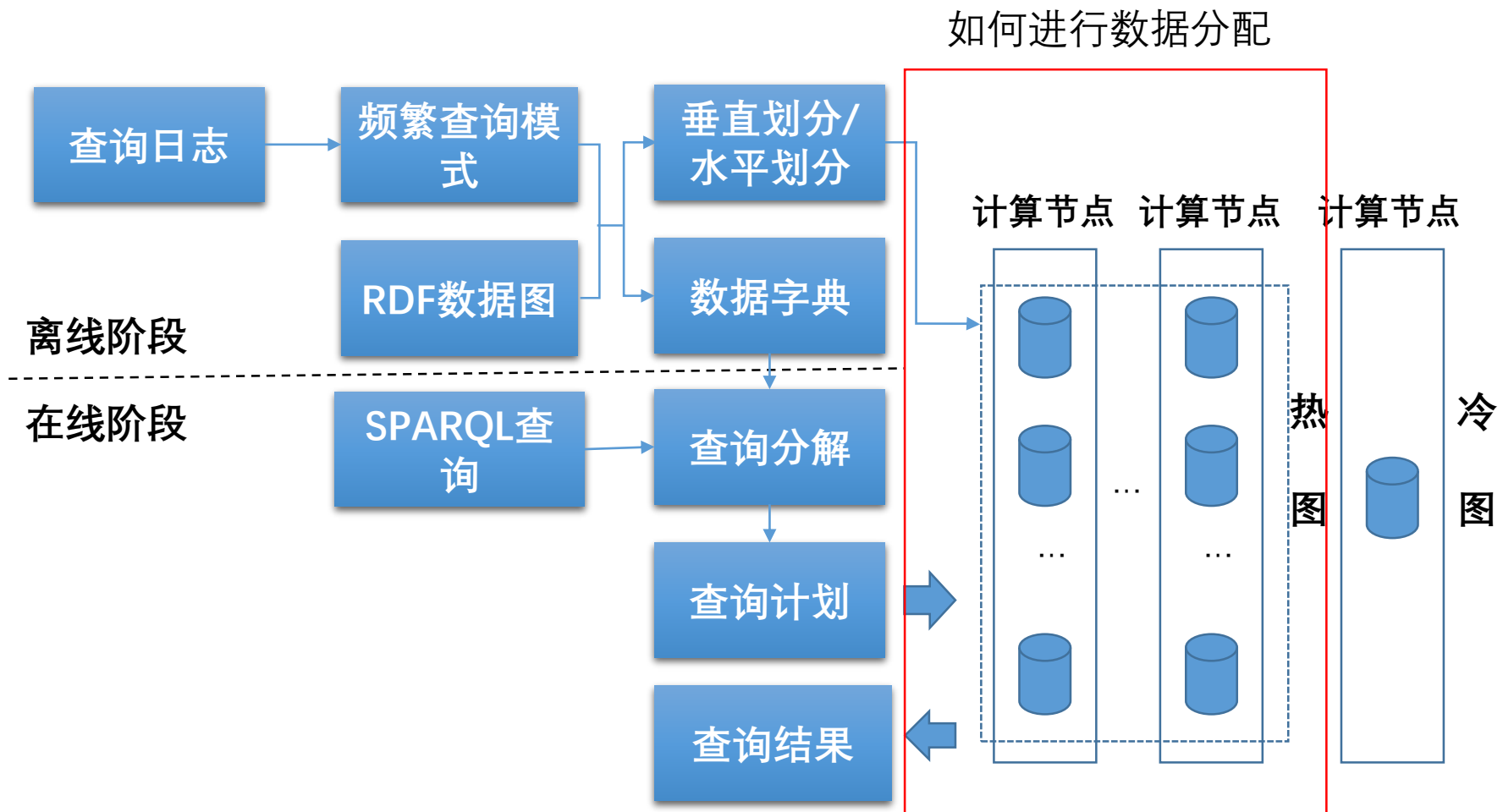


(c) Example Horizontal Fragment Generated from mp_3



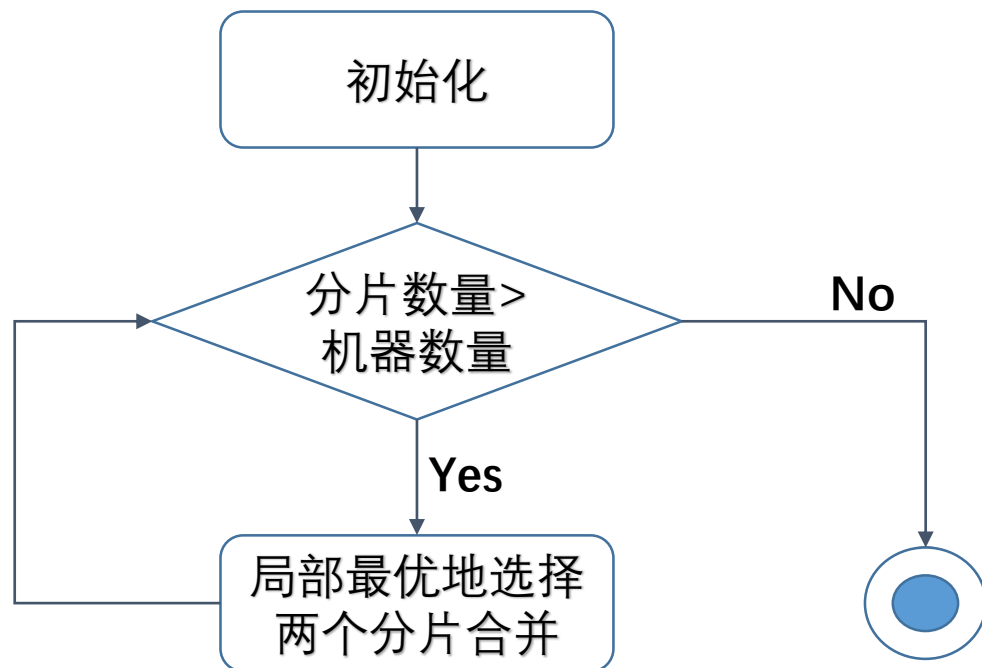
(d) Example Horizontal Fragment Generated from mp_4

方法框架

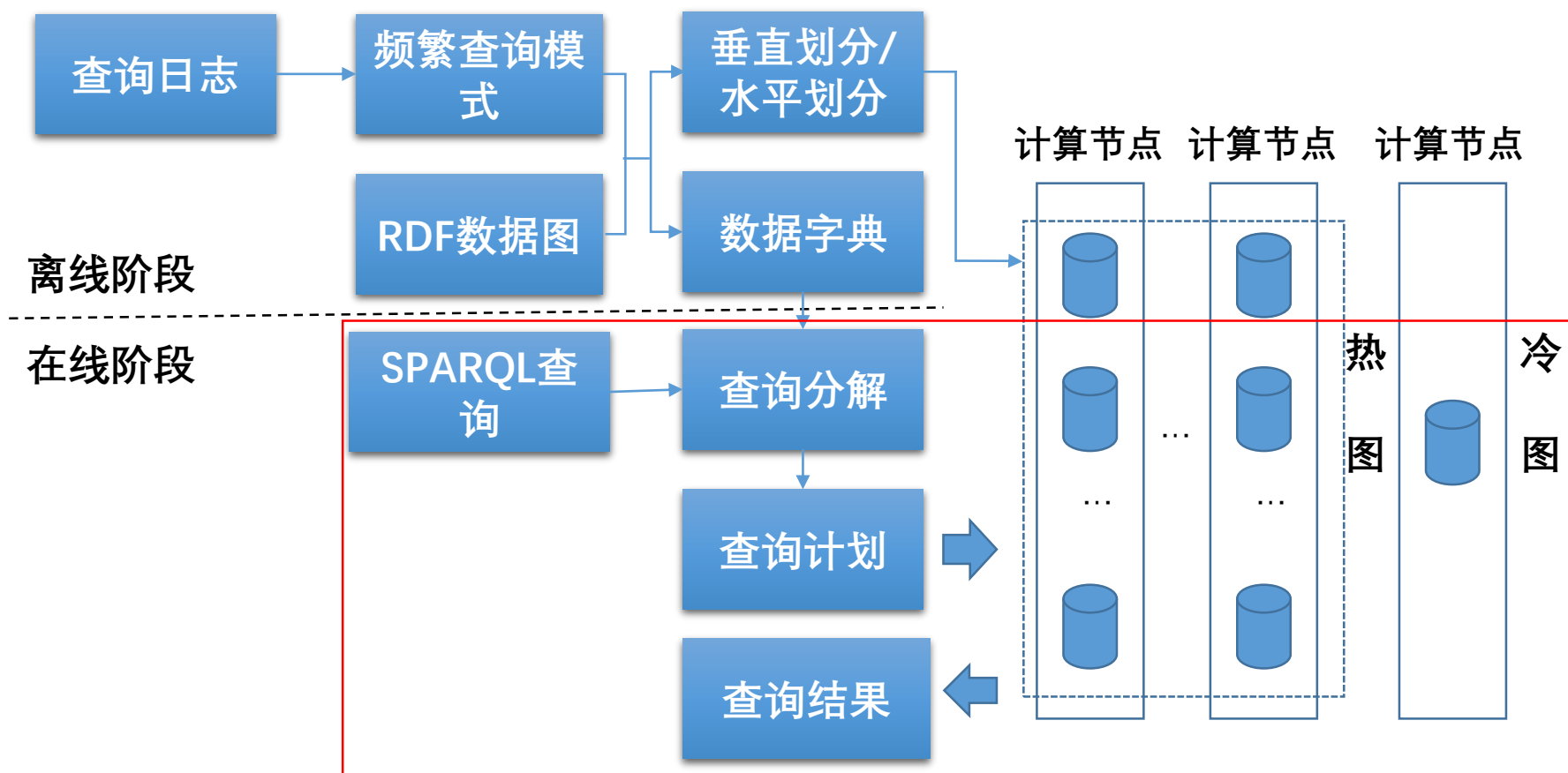


数据分配

- 经常被同一个查询所涉及的两个分片应该被分配在相同机器



方法框架



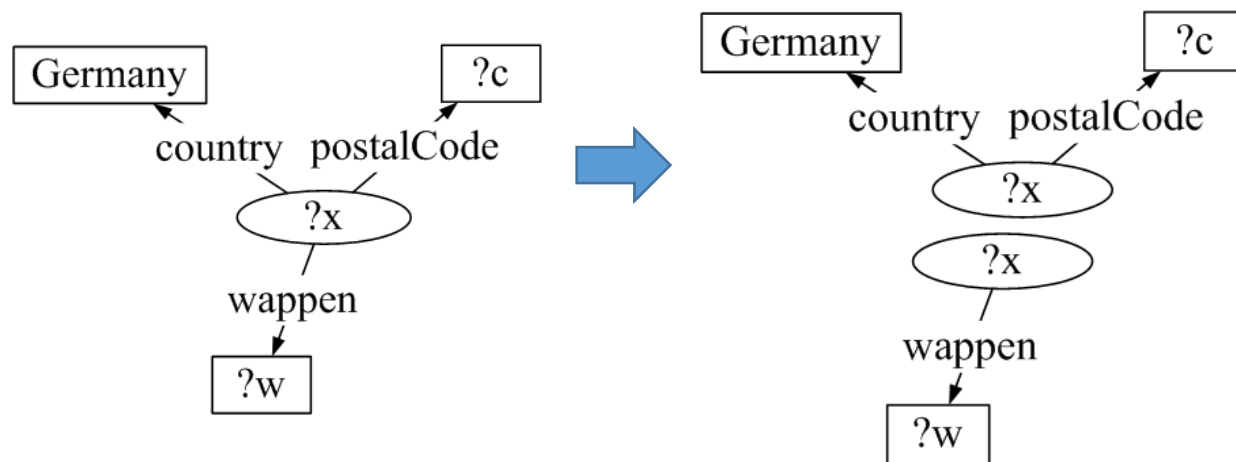
如何进行分布式查询处理

数据字典

- 关于数据划分与分配的元数据也需要维护在系统中以支持分布式查询处理
 - 数据分片的定义
 - 数据分片的大小
 - 数据分片的读取频率
 -

查询分解

- 当用户输入新的查询，系统首先将查询图分解成若干同态于频繁查询模式的子查询



查询优化与执行

- 查询分解之后，我们扩展经典的确定连接顺序的算法 System-R^[3] 来找出最优查询执行计划
- 然后，每个子查询首先在各个机器上执行并求得匹配
- 最后，这些匹配根据上述查询执行计划通过连接操作拼装出最终匹配

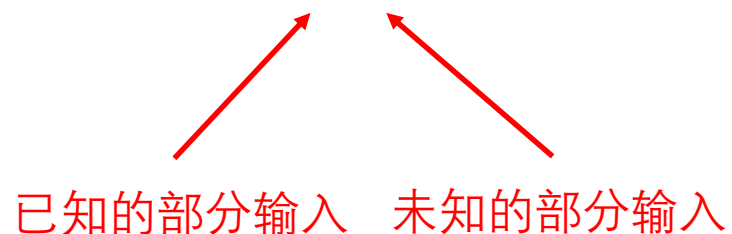
gStore^D



- Peng Peng, Lei Zou, M. Tamer Özsu, Lei Chen, Dongyan Zhao: Processing SPARQL queries over distributed RDF graphs. VLDB J. 25(2): 243-268 (2016)
- Peng Peng, Lei Zou, Runyu Guan: Accelerating Partial Evaluation in Distributed SPARQL Query Evaluation. ICDE 2019: 112-123

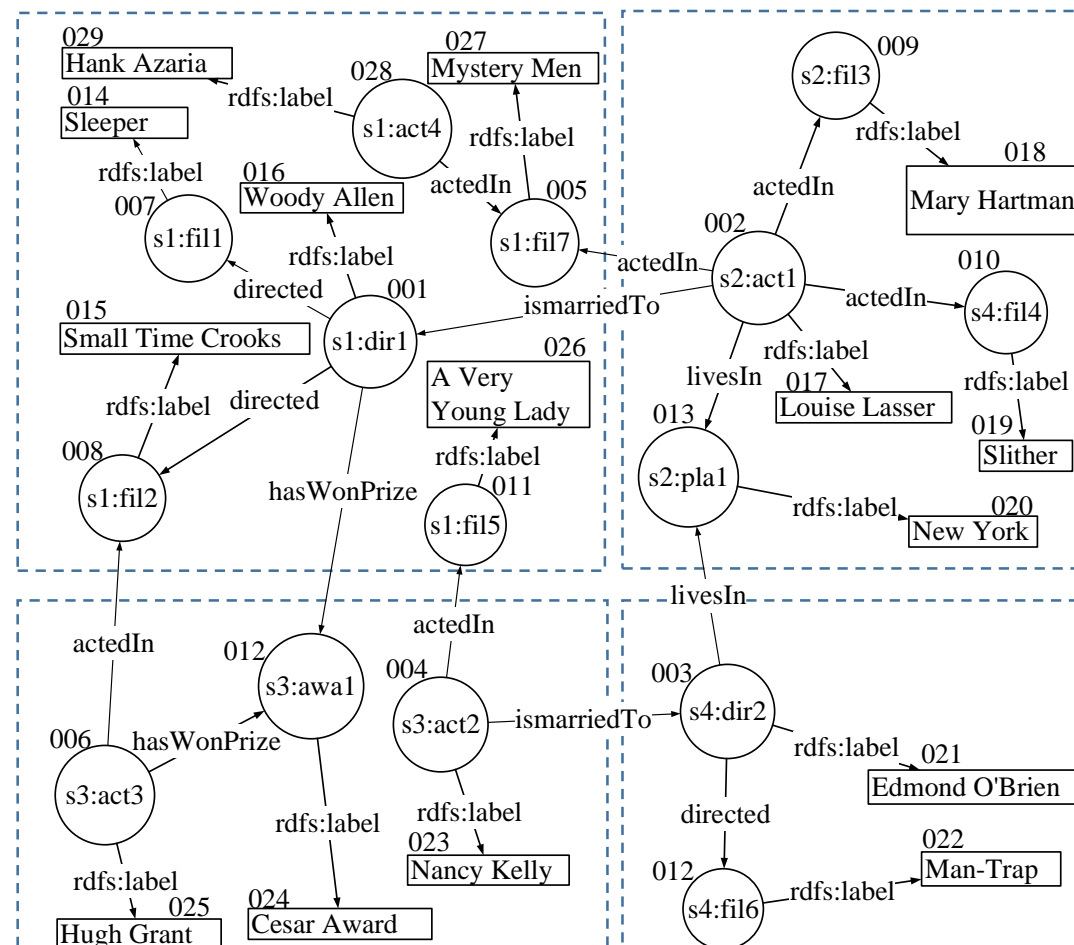
局部计算框架

- 局部计算 (Partial Evaluation) 是一个以及被应用在多个领域的分布式图计算模型



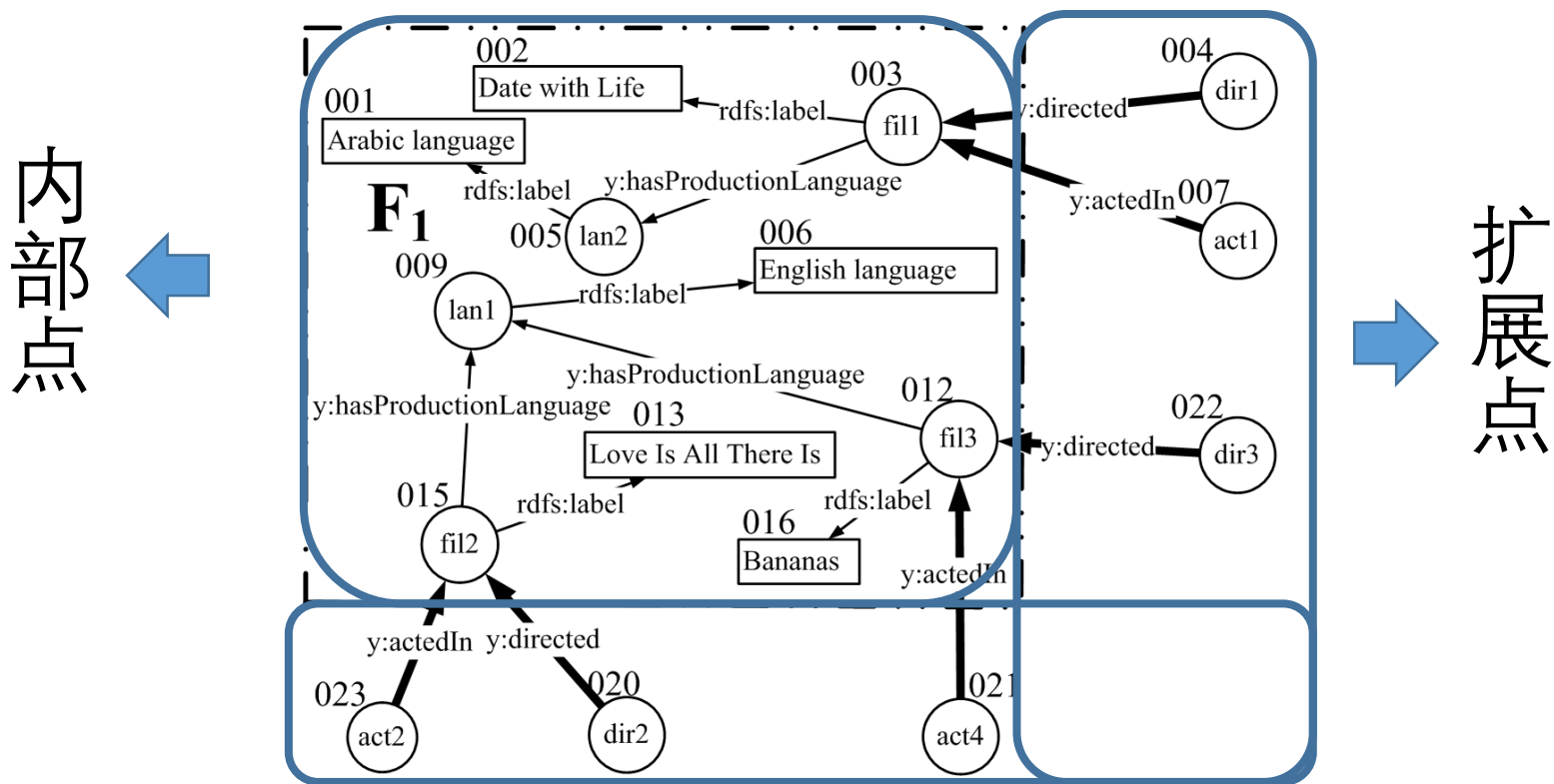
gStore^D

- 实际应用中的 RDF 数据图可能会被预先划分好

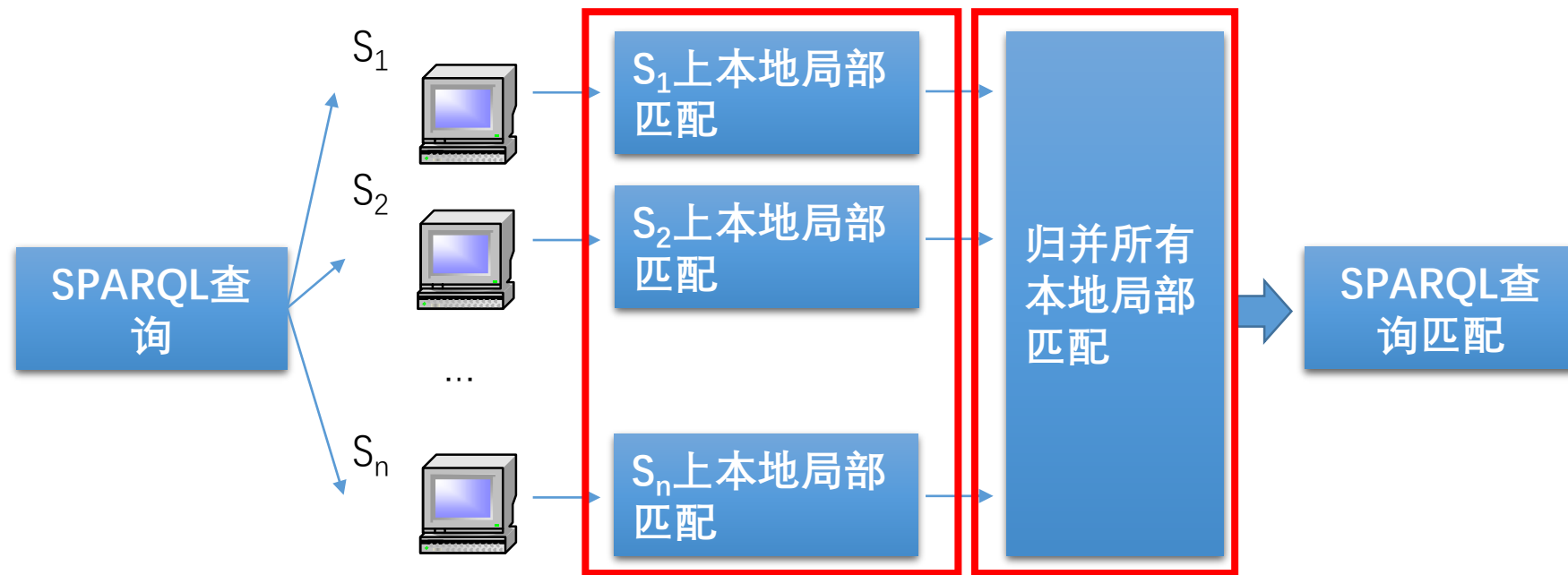


内部点和扩展点

- 对于每个计算节点而言，它不但存储其本身包含的点与边，还存储其相邻一步的点与边

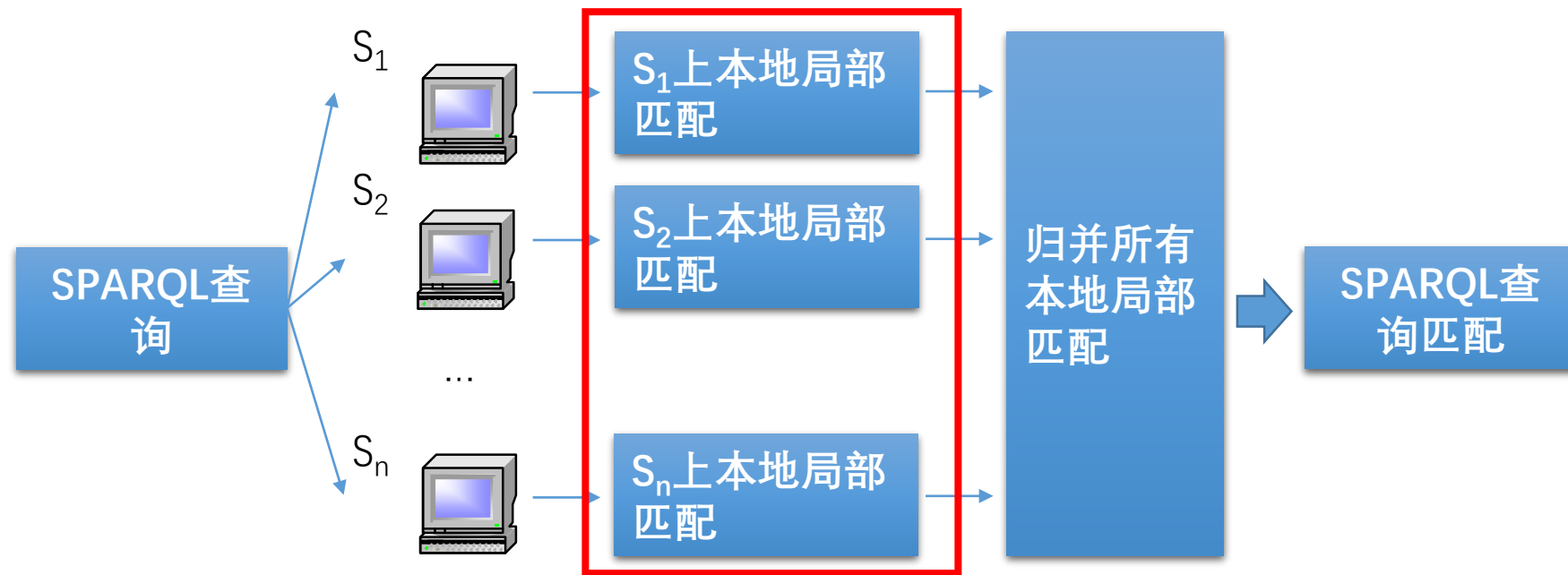


系统框架



如何定义本地局部匹配

系统框架

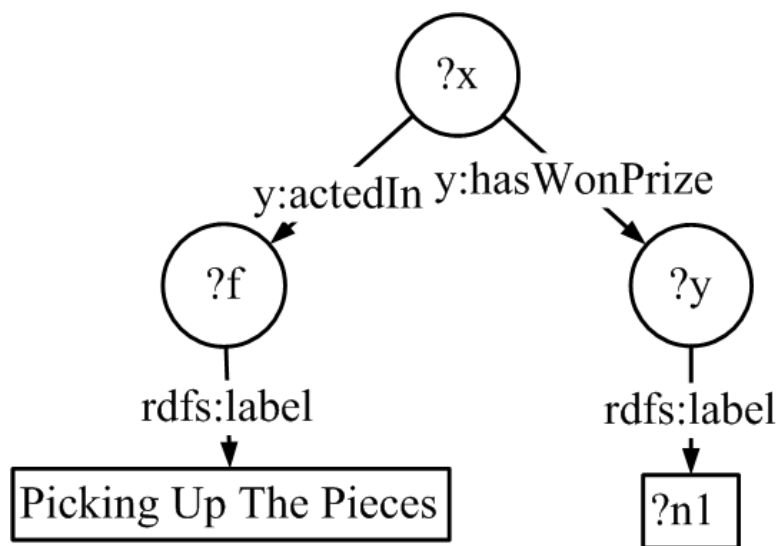


如何定义本地局部匹配

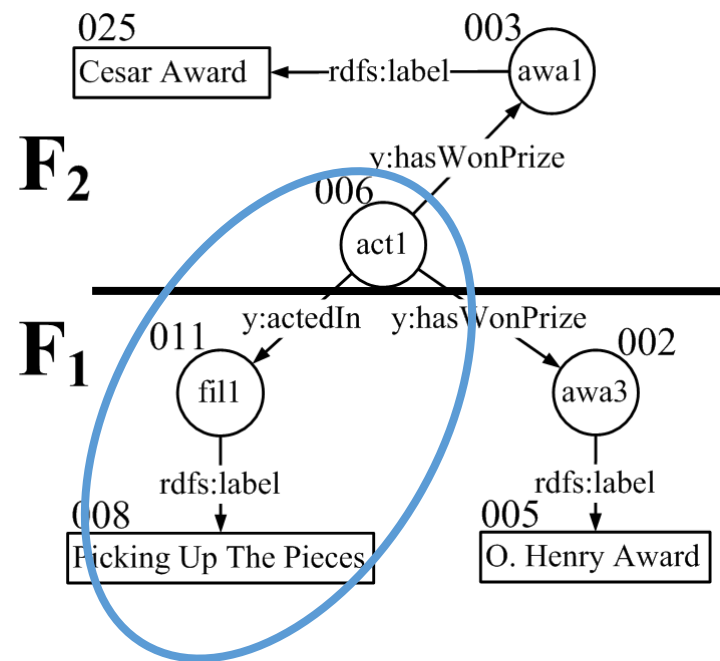
本地局部匹配定义

- 对于SPARQL查询Q而言，其在 S_i 上的本地局部匹配PM满足如下条件：
 - 如果PM[u]是内部点，那么u相邻边全部被匹配；
 - 去掉PM中所有扩展点，PM仍然是连通的


本地局部匹配示例



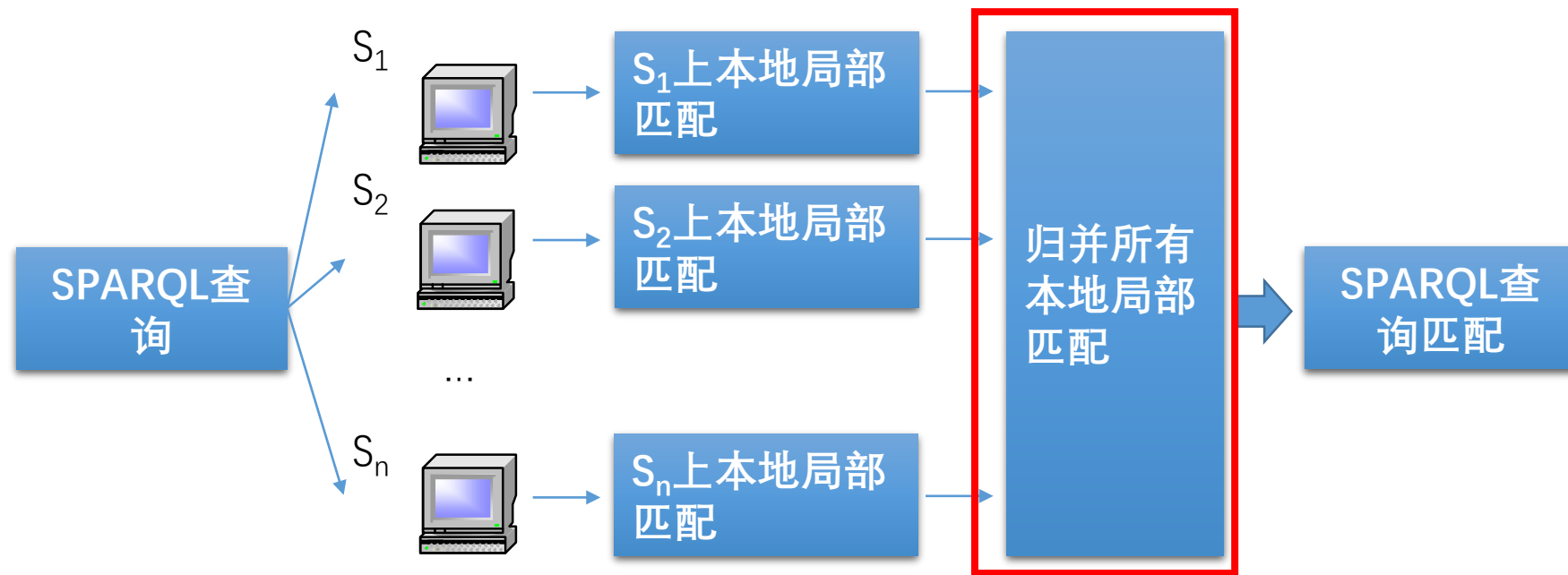
查询示例



本地局部匹配示例

- 
- 可以证明，利用上述定义下所得到的本地局部匹配能保证找到所有最终的SPARQL匹配（正确性），而且中间结果所涉及点和边的数量在上述定义下为最小（最优性）

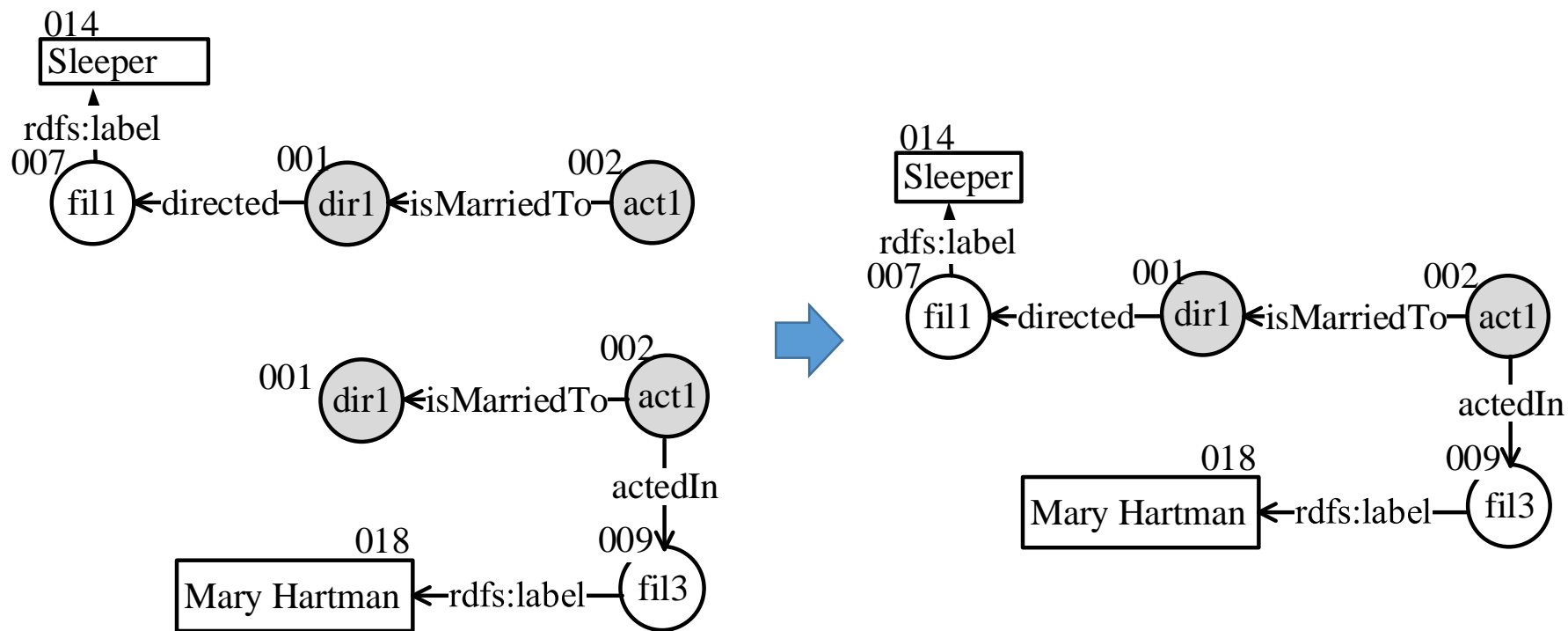
系统框架



怎样归并本地局部匹配

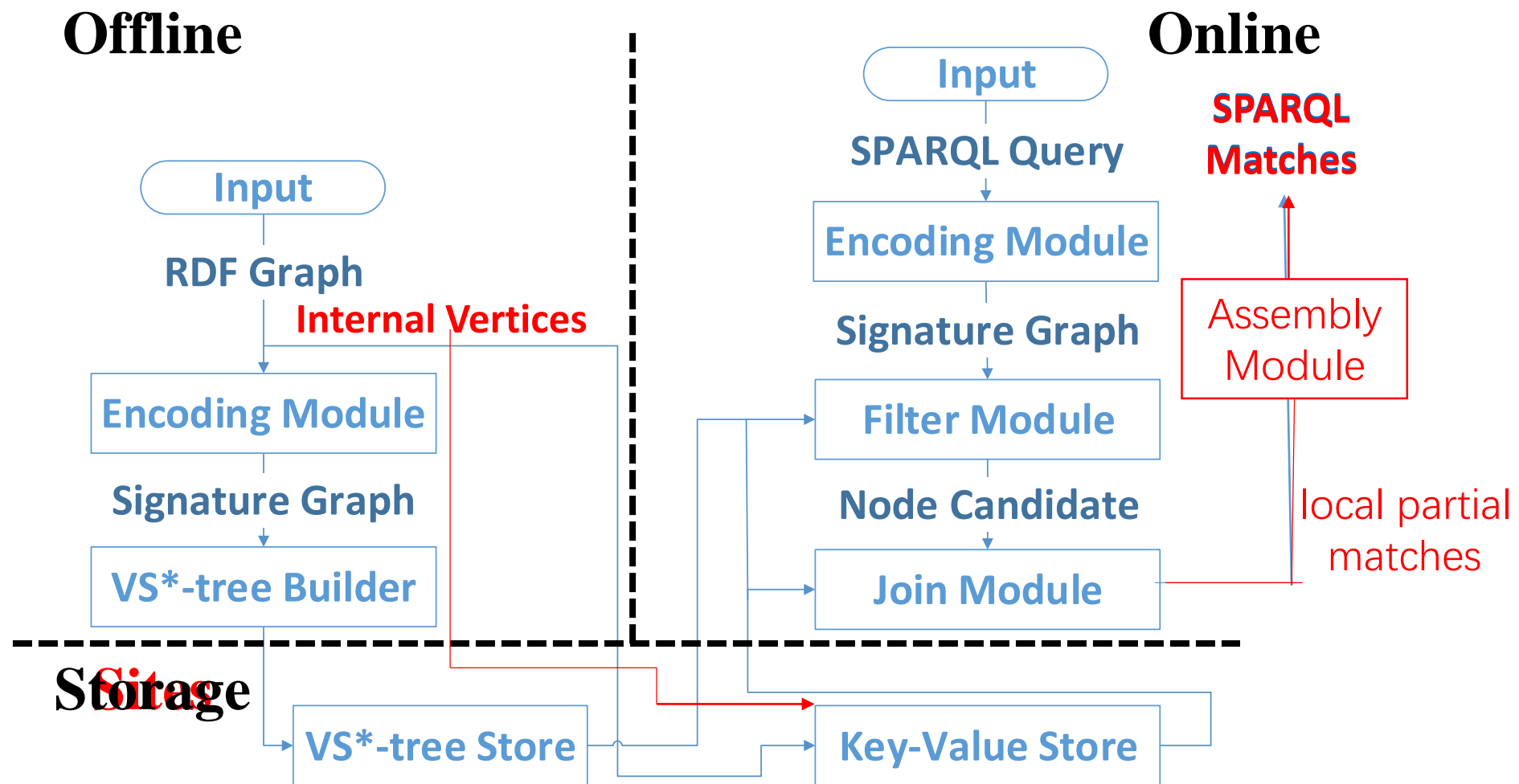
本地局部匹配归并

- 当所有的本地局部匹配得到之后，我们就需要将中间结果归并





- 这里，我们设计了两个归并策略：**集中式归并与分布式归并**

gStore^D系统



gStore^D@Github

 Personal Open source Business Explore Pricing Blog Support This repository Search Sign in Sign up


 **bnu05pp** / **gStoreD** Watch 1 Star 0 Fork 0





Code Issues 0 Pull requests 0 Pulse Graphs

A Distributed RDF Data Management System for Processing SPARQL Queries Over Distributed RDF Graphs

6 commits 1 branch 0 releases 1 contributor

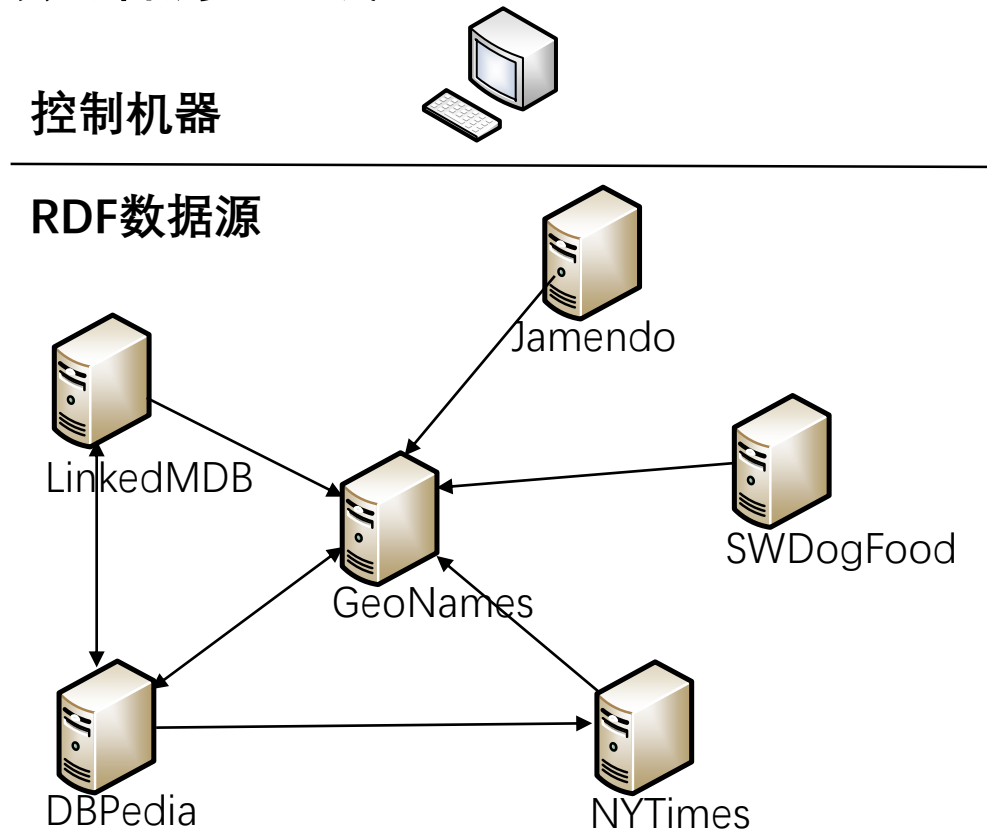
Branch: master New pull request Find file Clone or download

 **bnu05pp** Update README Latest commit 64c4d81 4 minutes ago

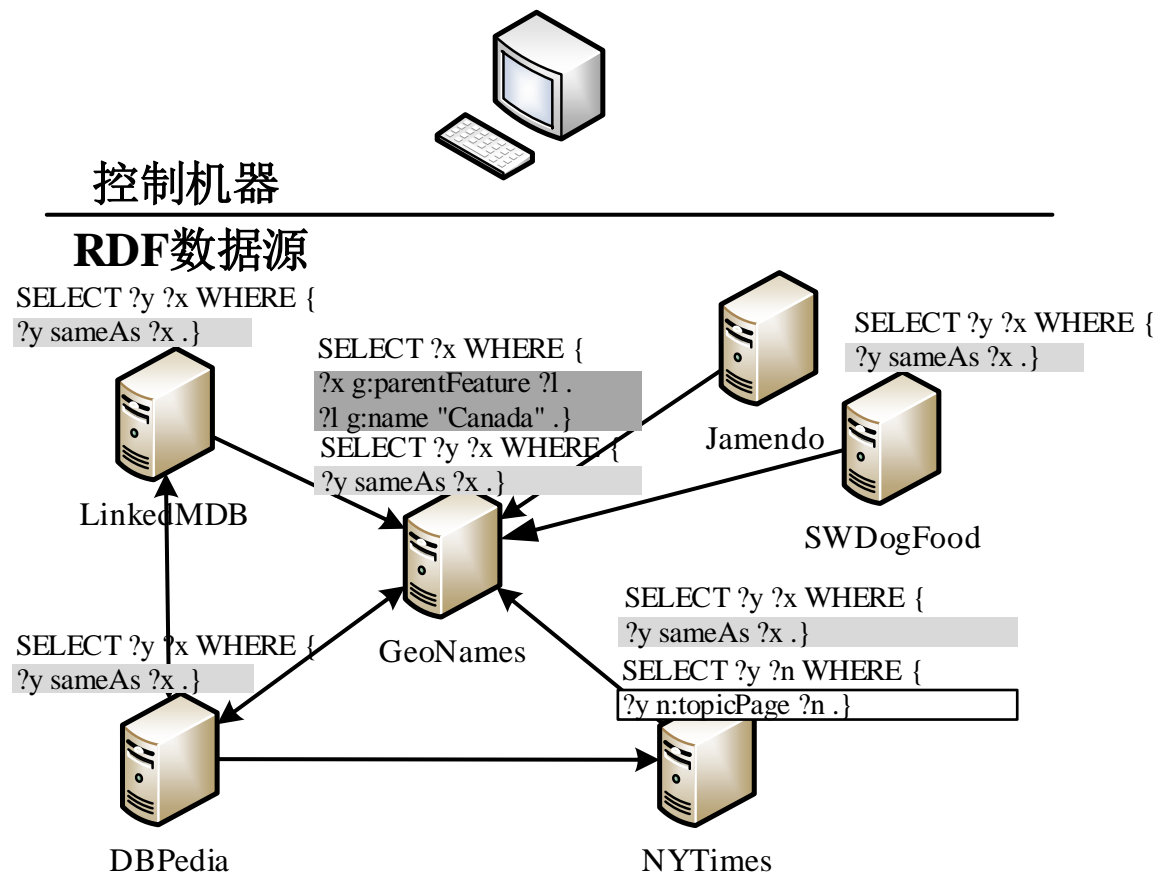
 .debug	Initially upload	21 minutes ago
 .objs	Initially upload	21 minutes ago
 .settings	Initially upload	21 minutes ago
 .tmp	Initially upload	21 minutes ago

联邦型分布式知识图谱管理系统

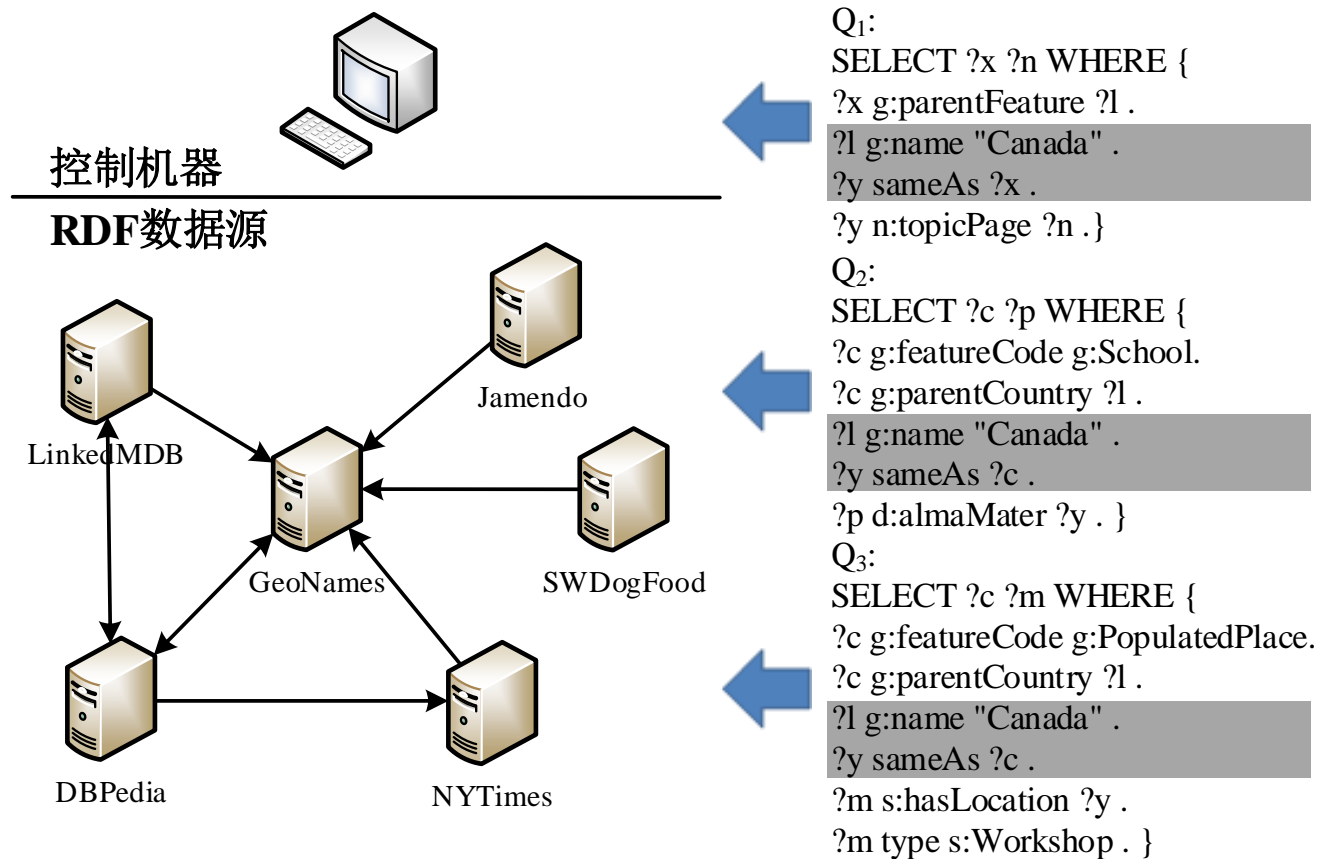
- 所谓联邦型分布式知识图谱管理系统，是由存储在不同“自治”机器上的知识图谱数据源组成



联邦型分布式知识图谱管理系统上的查询处理

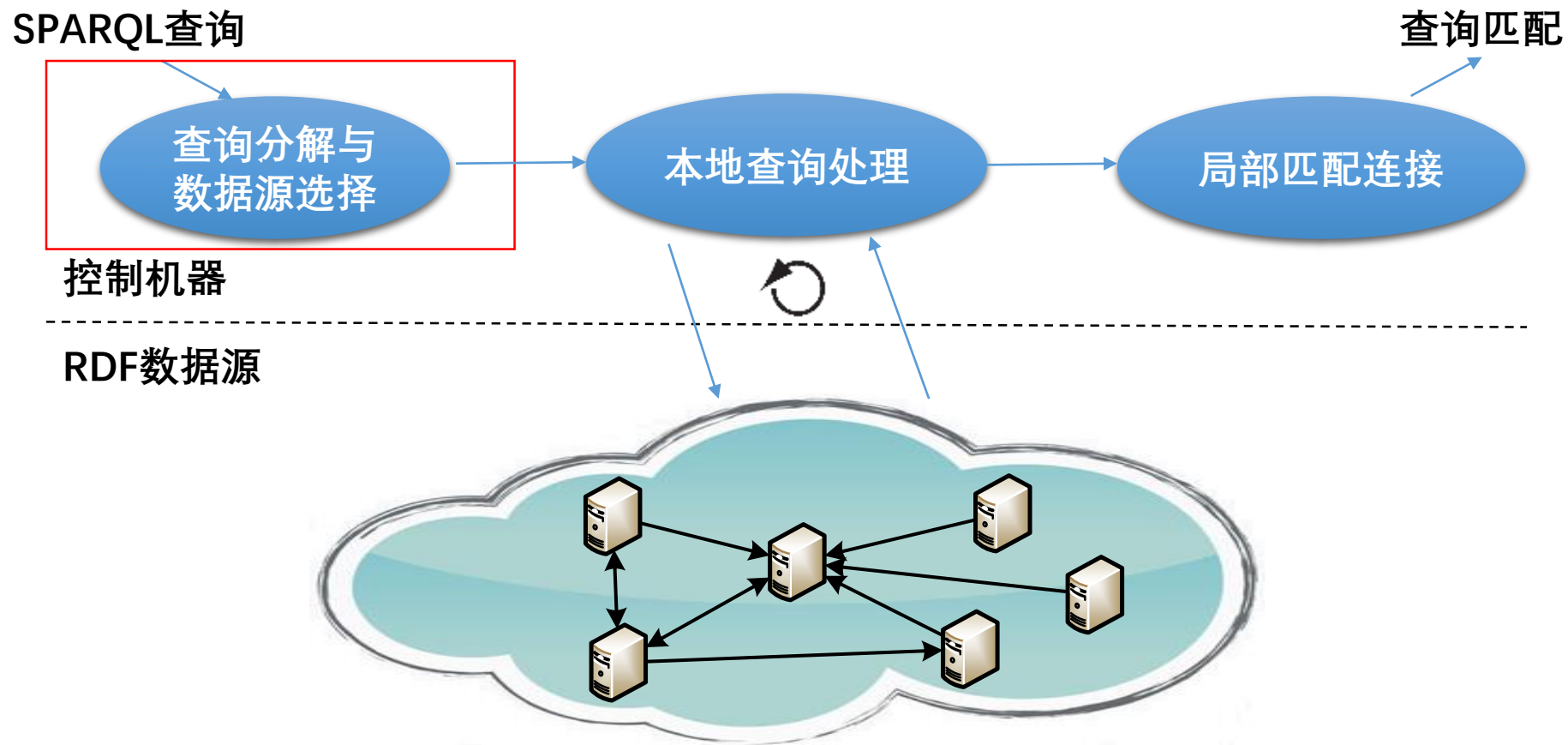


多查询优化



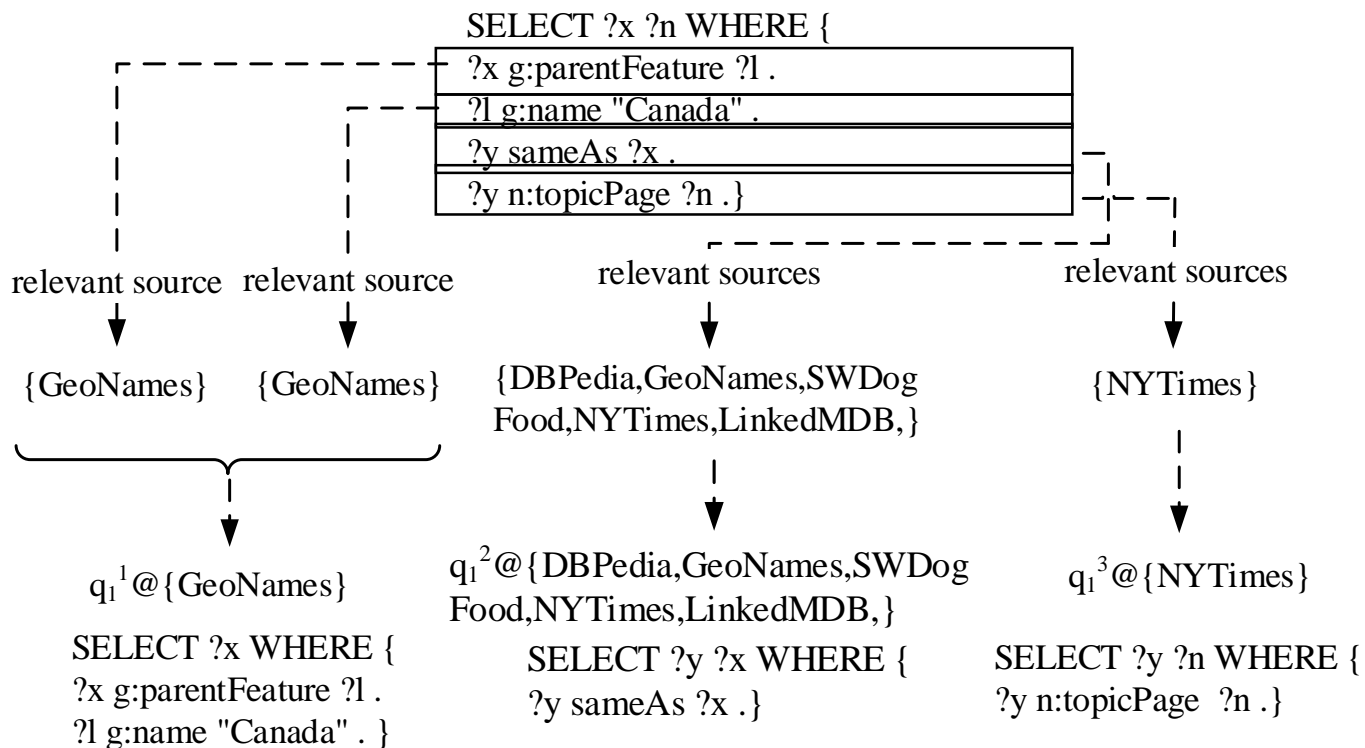
用户输入三个查询，且他们的相同子结构用文字阴影强调出来

系统框架

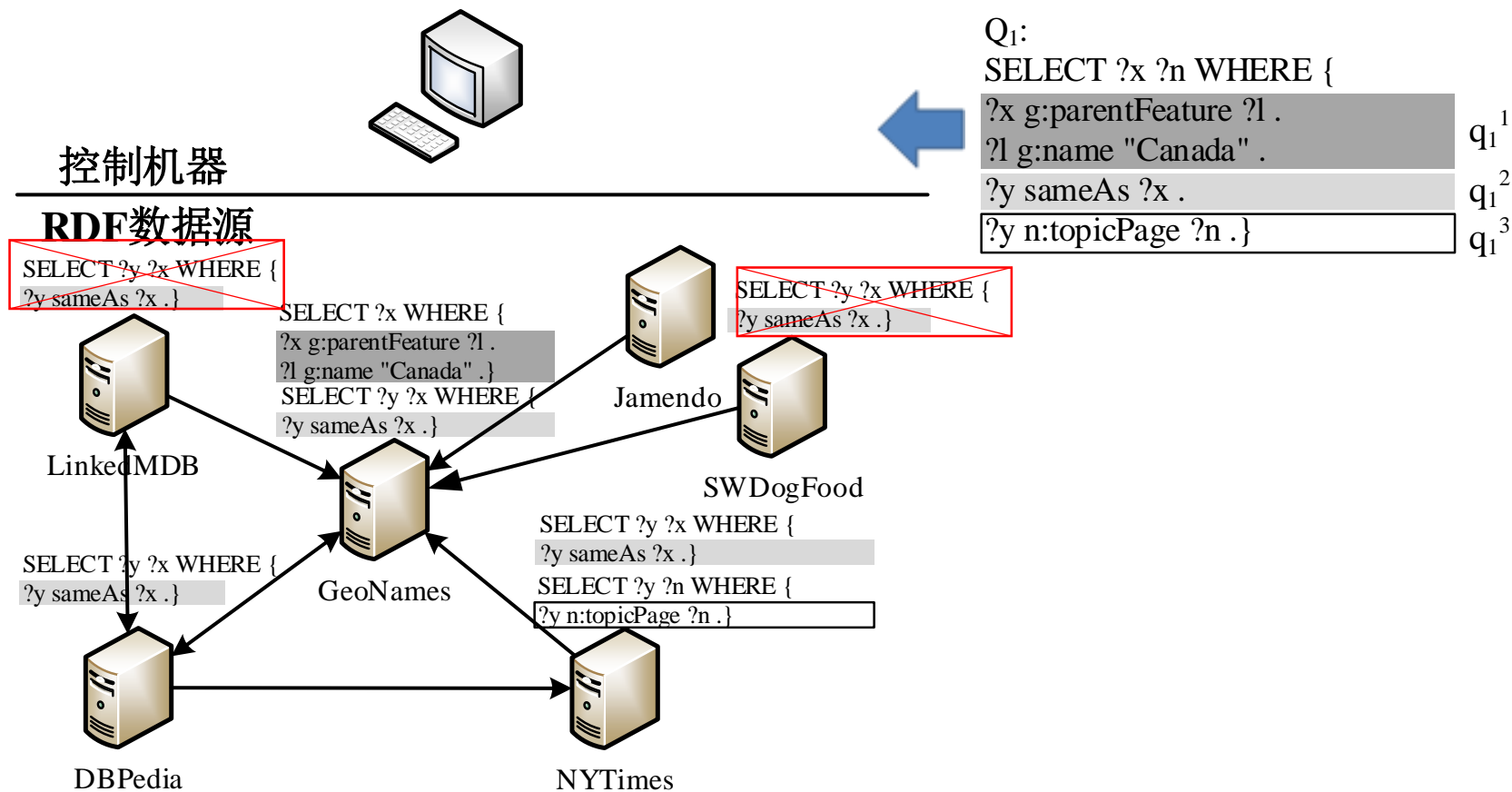


基本的查询分解与数据源选择

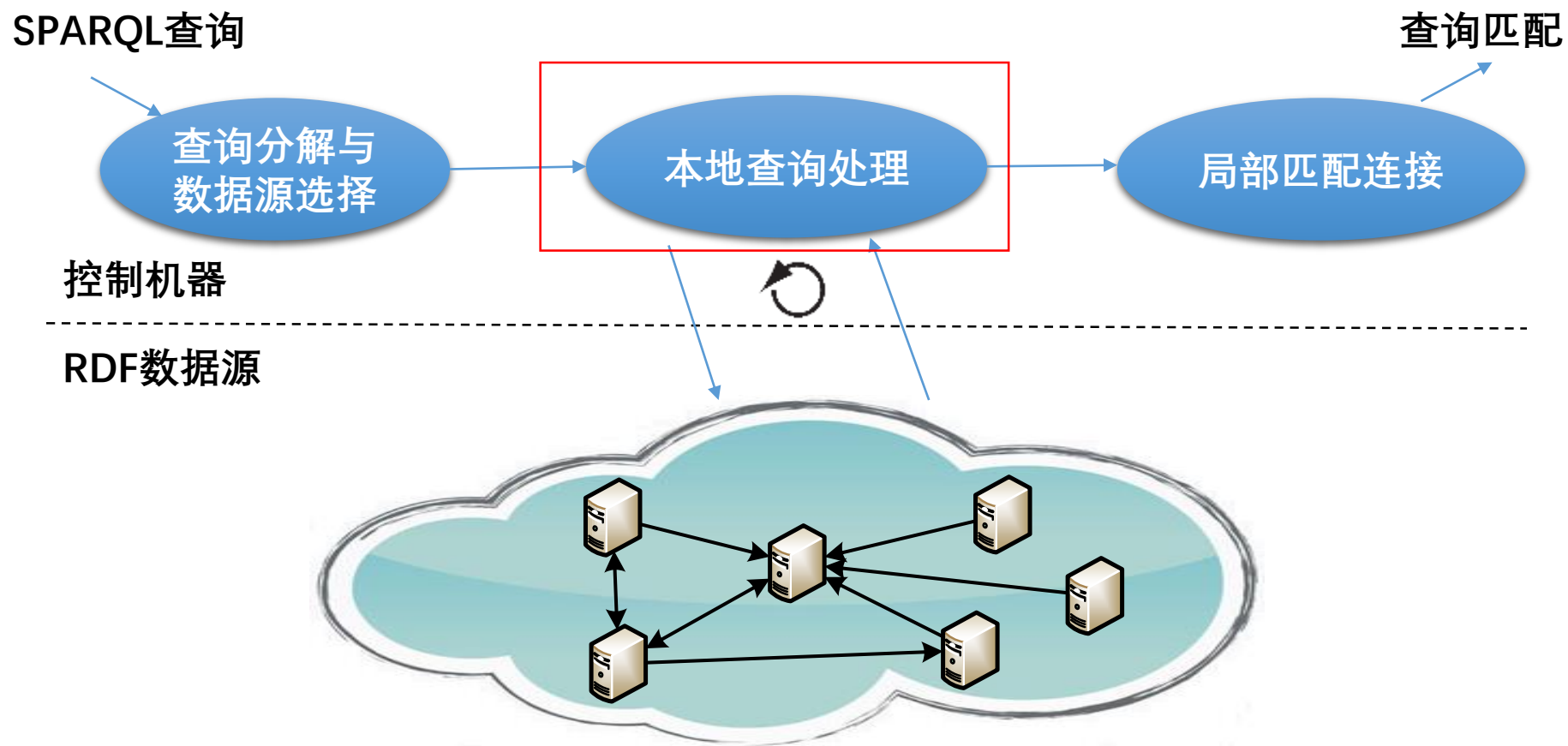
- 根据每个三元组模式在主体、属性和客体三个位置上的值确定所涉及的知识图谱数据源



基于知识图谱数据源拓扑关系的查询分解与数据源选择



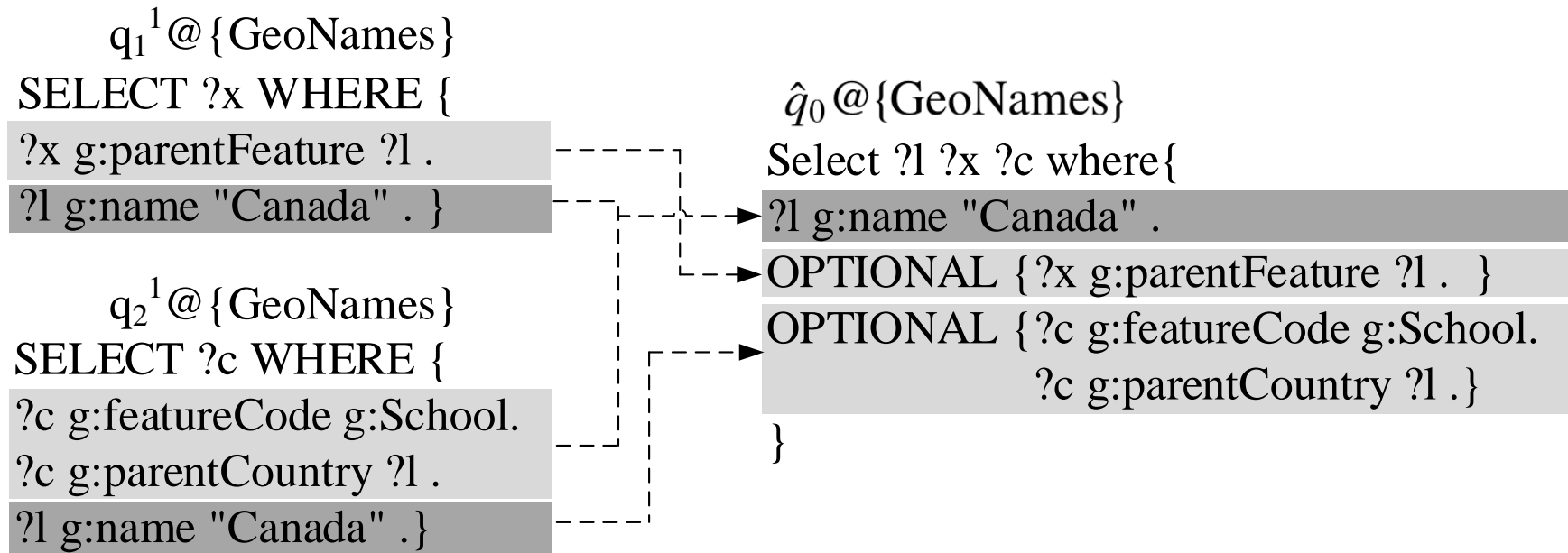
系统框架



本地查询处理

- 在实际应用中，这些本地查询相互间经常共享部分结构。这些共享的结构使得这些本地查询在被处理时能共享一些计算
- 这里，我们提出一个基于**查询重写**的策略来优化每个知识图谱数据源上的多本地查询处理

基于 OPTIONAL 的查询重写规则



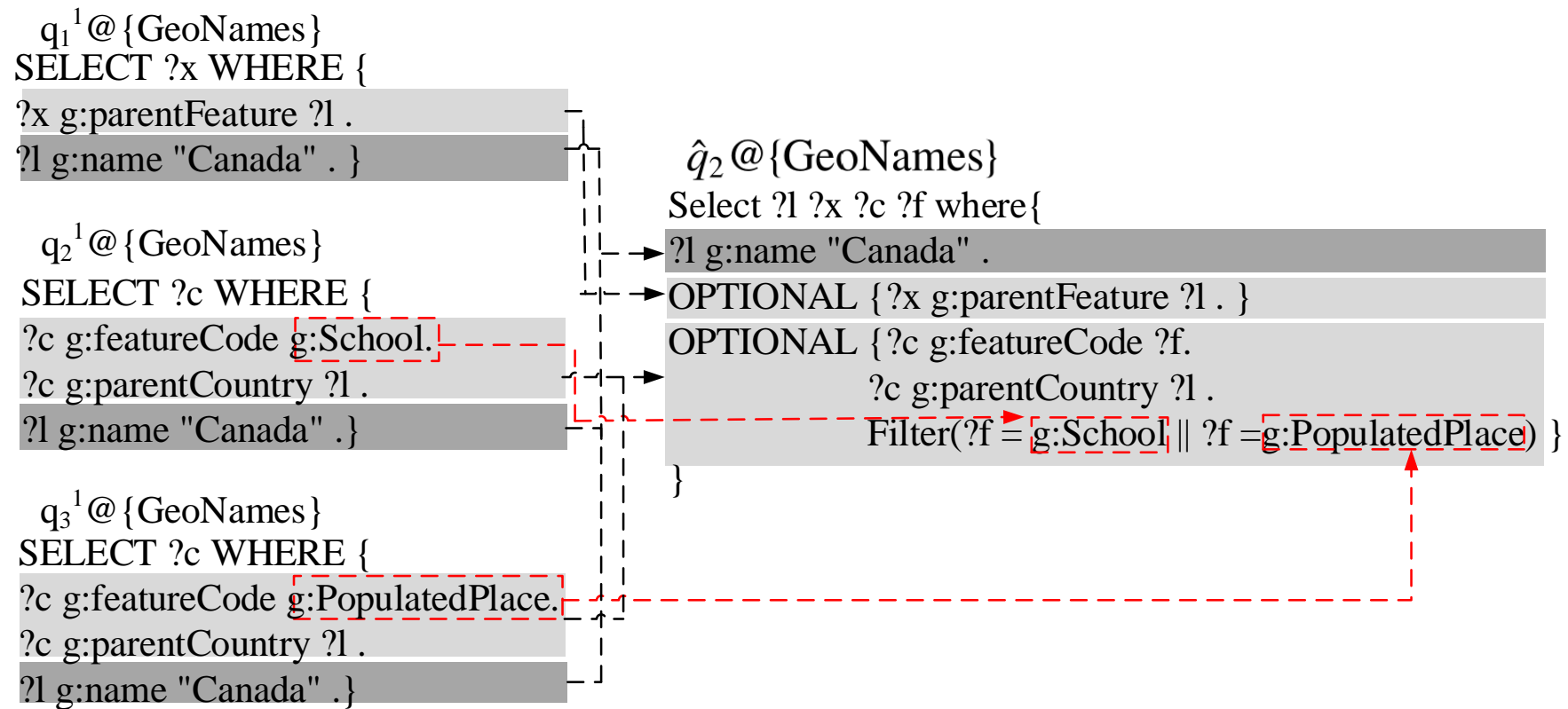
基于 FILTER 的查询重写规则

$q_2^1 @ \{ \text{GeoNames} \}$
SELECT ?c WHERE {
 ?c g:featureCode g:School .
 ?c g:parentCountry ?l .
 ?l g:name "Canada" .}

$q_3^1 @ \{ \text{GeoNames} \}$
SELECT ?c WHERE {
 ?c g:featureCode g:PopulatedPlace .
 ?c g:parentCountry ?l .
 ?l g:name "Canada" .}

$\hat{q}_1 @ \{ \text{GeoNames} \}$
Select ?l ?c ?f where {
 ?c g:featureCode ?f .
 ?c g:parentCountry ?l .
 ?l g:name "Canada" .
 Filter(?f = g:School || ?f = g:PopulatedPlace)
}

混合查询重写规则



查询重写代价模型

- 我们定义一个 SPARQL 查询 Q 的代价如下

$$\text{cost}(Q) = \begin{cases} \min(\text{Sel}(e)), & \text{if } Q \text{ is a BGP and } e \in E(Q) \\ \min\{\text{cost}(Q_1), \text{cost}(Q_2)\}, & \text{if } Q = Q_1 \text{ AND } Q_2 \\ \text{cost}(Q_1) + \text{cost}(Q_2), & \text{if } Q = Q_1 \text{ UNION } Q_2 \\ \text{cost}(Q_1) + \Delta_1, & \text{if } Q = Q_1 \text{ OPT } Q_2 \\ \text{cost}(Q_1) + \Delta_2, & \text{if } Q = Q_1 \text{ FILTER } F \end{cases}$$

其中，根据前人工作可知， Δ_1 和 Δ_2 都是可以忽略的小数值

查询重写目标

- 给定一个本地查询集合，查询重写目标为找出代价最小的重写查询集合

$$\hat{Q} = \{\hat{q}_1, \dots, \hat{q}_m\}$$

- 我们可以证明找出代价最小的重写查询集合是NP-complete^[5]

查询重写过程

First equivalence class

```
q11@{GeoNames}
SELECT ?x WHERE {
  ?x g:parentFeature ?l .
  ?l g:name "Canada" . }
```

Second equivalence class

```
q21@{GeoNames}
SELECT ?c WHERE {
  ?c g:featureCode g:School .
  ?c g:parentCountry ?l .
  ?l g:name "Canada" . }
```

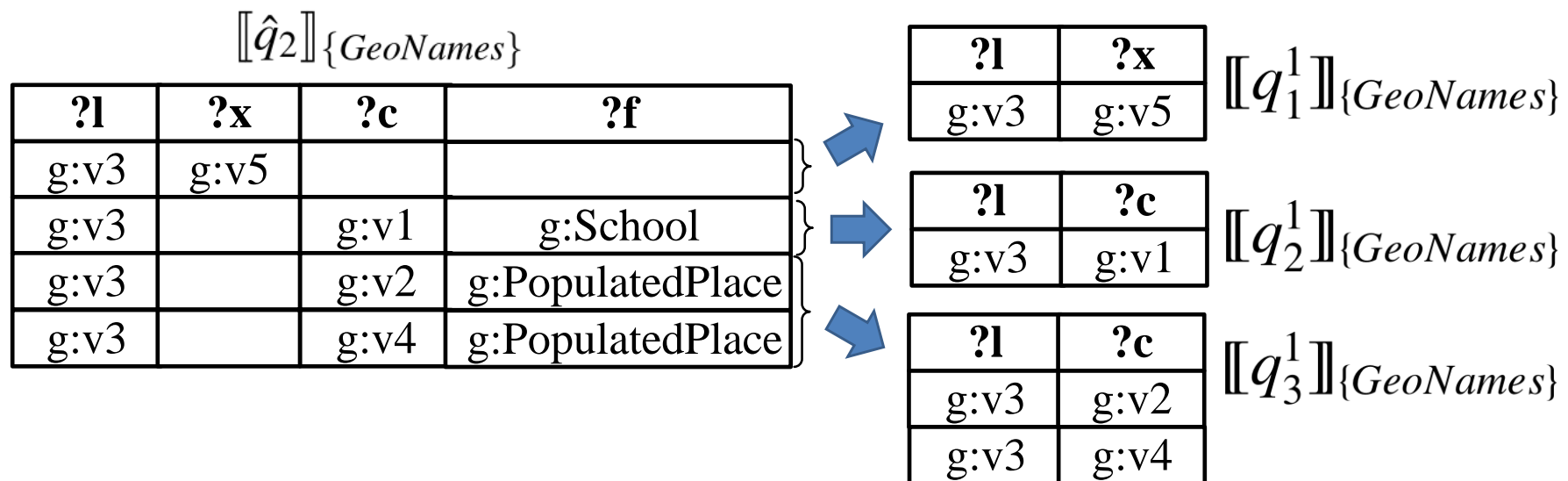
```
q31@{GeoNames}
SELECT ?c WHERE {
  ?c g:featureCode g:PopulatedPlace .
  ?c g:parentCountry ?l .
  ?l g:name "Canada" . }
```

```
q11@{GeoNames}
SELECT ?x WHERE {
  ?x g:parentFeature ?l .
  ?l g:name "Canada" . }
```

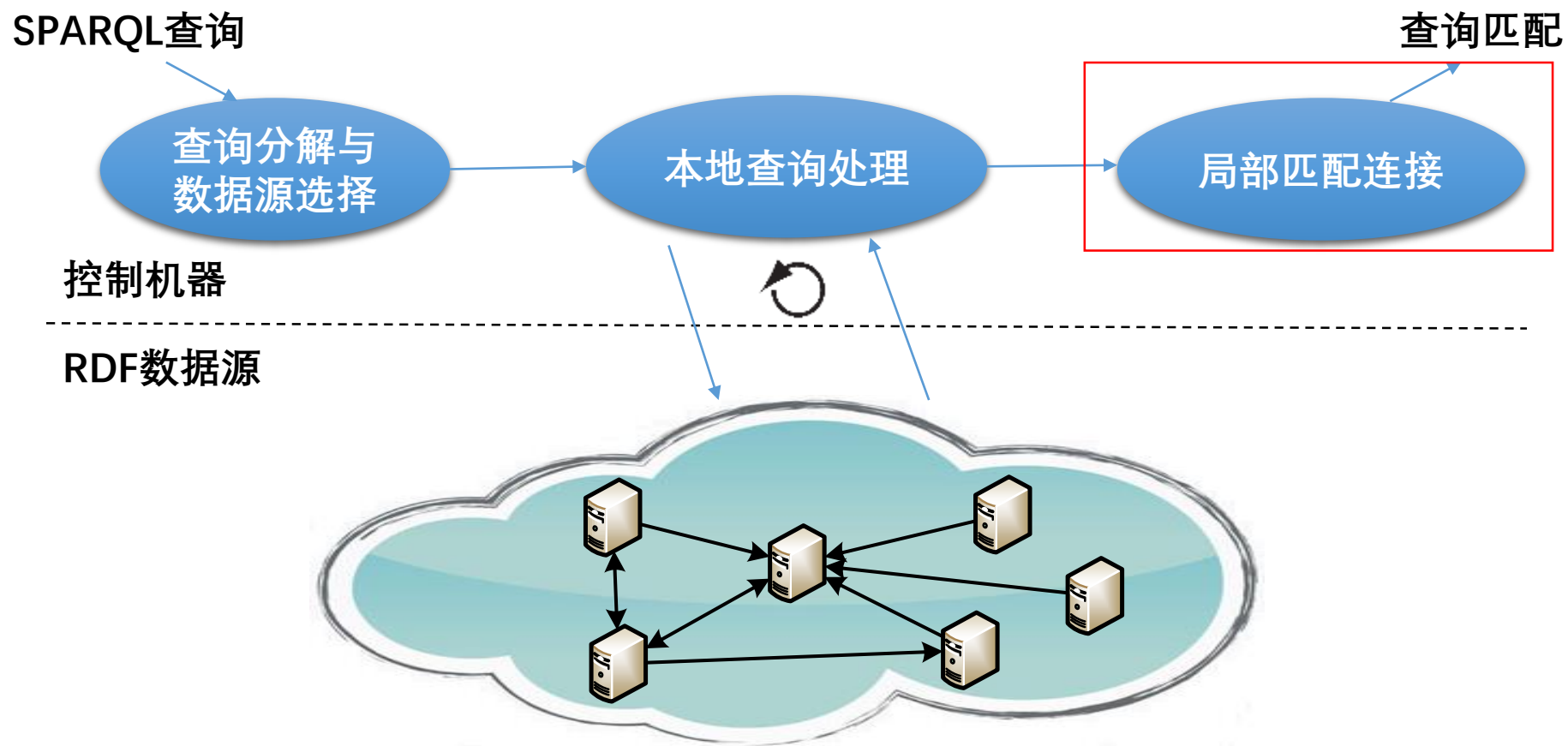
```
q̂1@{GeoNames}
Select ?l ?c ?f where{
  ?l g:name "Canada" .
  ?c g:featureCode ?f .
  ?c g:parentCountry ?l .
  Filter(?f = g:School ||
    ?f = g:PopulatedPlace) }
```

```
q̂2@{GeoNames}
Select ?l ?x ?c ?f where{
  ?l g:name "Canada" .
  OPTIONAL { ?x g:parentFeature ?l . }
  OPTIONAL { ?c g:featureCode ?f .
    ?c g:parentCountry ?l .
    Filter(?f = g:School ||
      ?f = g:PopulatedPlace) } }
```

查询结果后处理

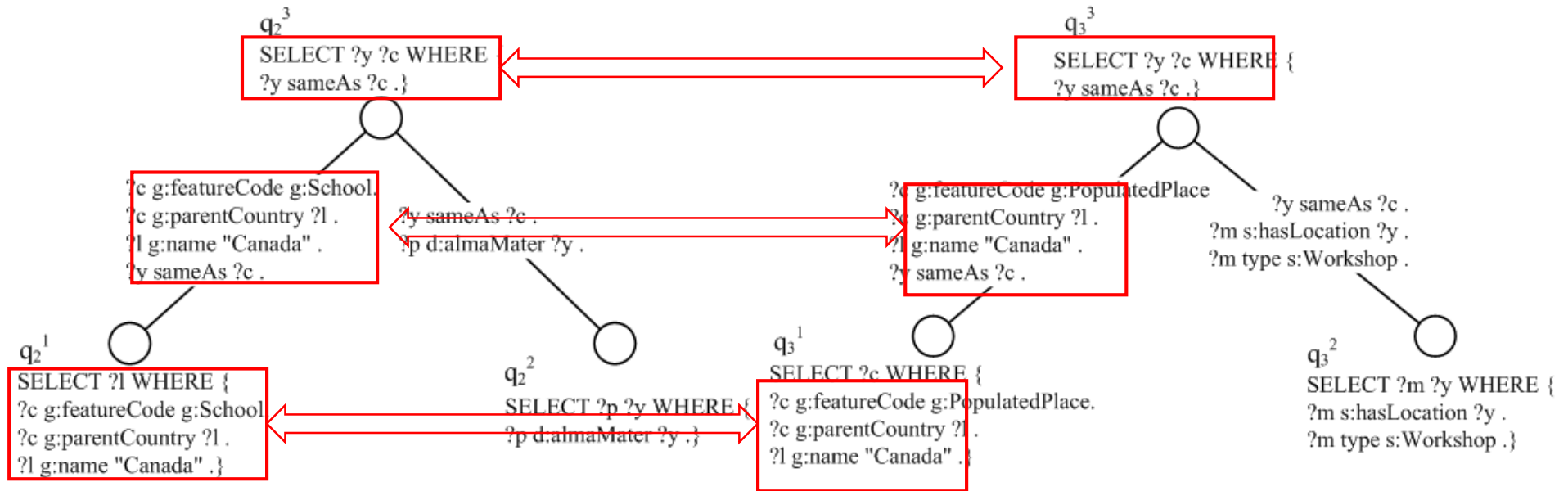


系统框架



局部匹配连接

- 不同查询分解而得的本地查询匹配进行连接时有些连接操作计算是可以相同的
- 处理这些用户输入的相似查询时，我们可以通过**共享相同的连接操作**计算来降低查询整体响应时间



- 形式化地说,

$$\left. \begin{array}{l} \llbracket q_2^1 \rrbracket \bowtie \llbracket q_2^3 \rrbracket \\ \llbracket q_3^1 \rrbracket \bowtie \llbracket q_3^3 \rrbracket \end{array} \right\} \rightarrow (\llbracket q_2^1 \rrbracket \cup \llbracket q_3^1 \rrbracket) \bowtie (\llbracket q_2^3 \rrbracket \cup \llbracket q_3^3 \rrbracket)$$

Thank You!

