

Structured data management on MapReduce

Ioana Manolescu

INRIA

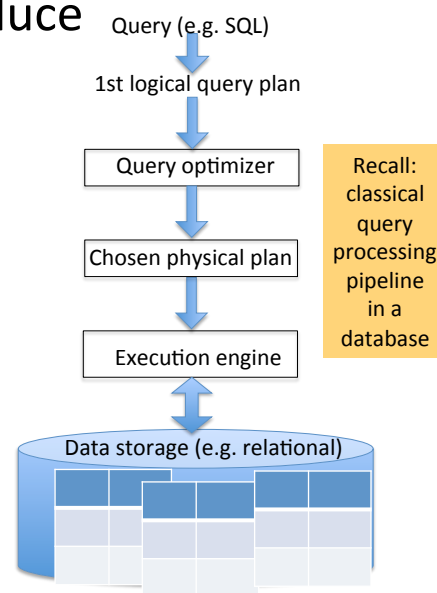
ioana.manolescu@inria.fr

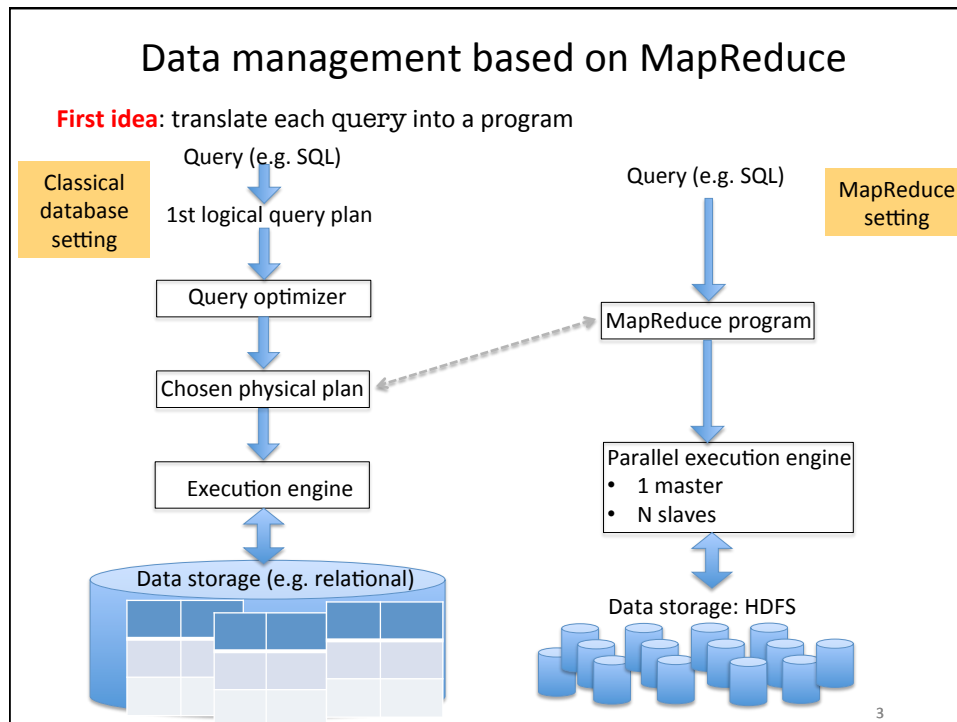
<http://pages.saclay.inria.fr/ioana.manolescu>



Data management based on MapReduce

- Idea: use MapReduce as a low-level programming layer
 - To take advantage of the parallelism
 - In order to implement highly scalable query processors
- Recall: DBMS query processing stages

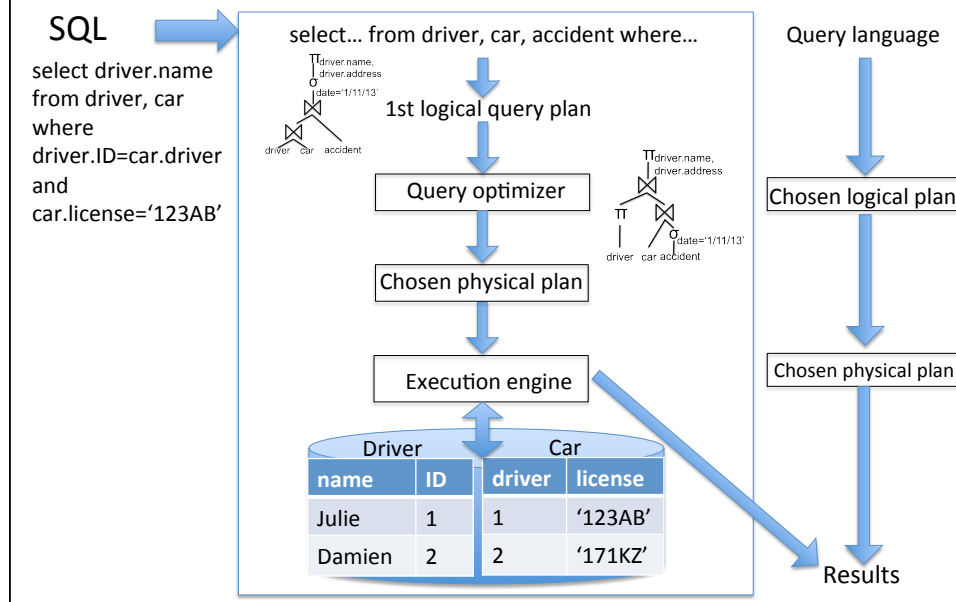




Implementing queries through MapReduce

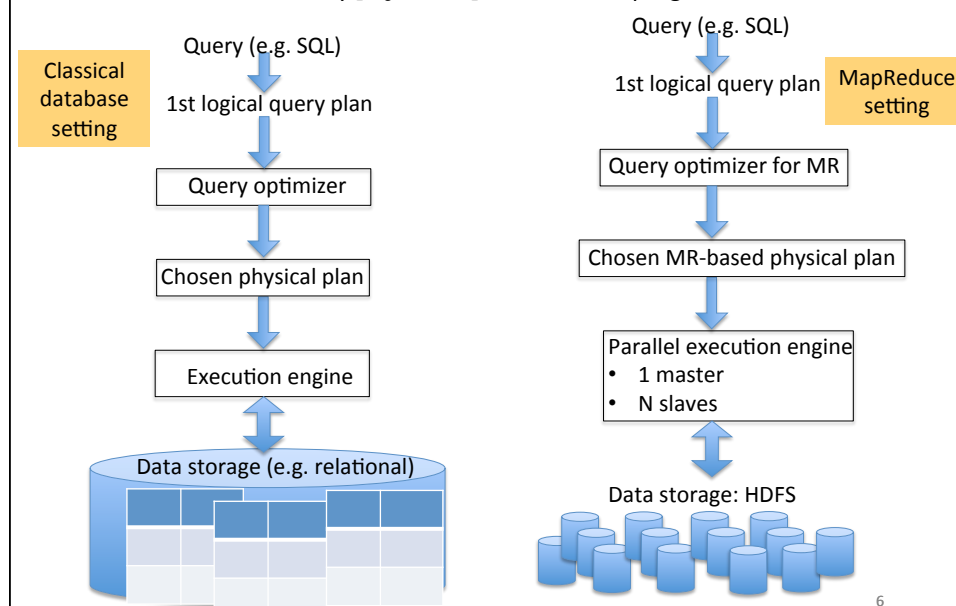
- `SELECT MONTH(c.start_date), COUNT(*)
FROM customer c
GROUP BY MONTH(c.start_date)`
- `SELECT c.name, o.total
FROM customer c, order o
WHERE c.id=o.cid`
- `SELECT c.name, SUM(o.total)
FROM customer c, order o
WHERE c.id=o.cid
GROUP BY c.name`

Recall: query processing stages in a DBMS



Data management based on MapReduce

Second idea: translate every physical operator into a program



Implementing physical operators on MapReduce

- **To avoid writing code for each query!**
- If each operator is a (small) MapReduce program, we can evaluate queries by composing such small programs
- The optimizer can then chose the best MR physical operators and their orders (just like in the traditional setting)
- Translate:
 - Unary operators (σ and π)
 - Binary operators (mostly: \bowtie on equality, i.e. equijoin)
 - N-ary operators (complex join expressions)

7

Implementing unary operators on MapReduce

- Selection ($\sigma_{\text{pred}} (R)$):
 - Split the R input tuples over all the nodes
 - **Map:**
 - foreach t which satisfies pred in the input partition
 - Output ($\text{hn}(\text{t.toString}()), \text{t}$); // hn fonction de hash
 - **Reduce:**
 - Concatenate all the inputs

What values should hn take?

8

Implementing unary operators on MapReduce

- Projection ($\pi_{\text{cols}}(R)$):
 - Split R tuples across all nodes
 - **Map:**

```
foreach t
  output (hn(t),  $\pi_{\text{cols}}(t)$ )
```
 - **Reduce:**
 - Concatenate all the inputs
- Better idea?

9

Recall: physical operators for binary joins (classical DBMS scenario)

Example: equi-join ($R.a=S.b$)

Nested loops join: $O(|R| \times |S|)$

```
foreach t1 in R{
  foreach t2 in S {
    if t1.a = t2.b then output (t1 || t2)
  }
}
```

Merge join: // requires sorted inputs $O(|R| + |S|)$

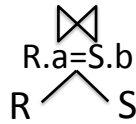
```
repeat{
  while (!aligned) { advance R or S };
  while (aligned) { copy R into topR, S into topS };
  output topR x topS;
} until (endOf(R) or endOf(S));
```

Hash join: // builds a hash table in memory $O(|R| + |S|)$

```
While (!endOf(R)) { t ← R.next; put(hash(t.a), t); }
While (!endOf(S)) { t ← S.next;
  matchingR = get(hash(S.b));
  output(matchingR x t);
}
```

Also:
 Block nested loops join
 Index nested loops join
 Hybrid hash join
 Hash groups / teams
 ...

Implementing equi-joins on MapReduce (1)



Repartition join [Blanas 2010] (~symetric hash)

Mapper:

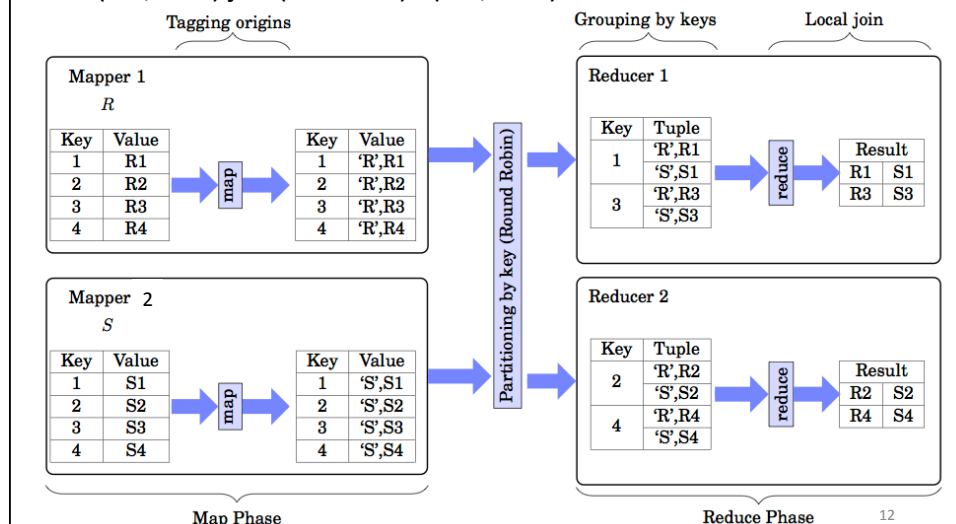
- Output (t.a, («R», t)) for each t in R
- Output (t.b, («S», t)) for each t in S

Reducer:

- Foreach input key k
 - Res_k = set of all R tuples on k \times set of all S tuples on k
- Output Res_k

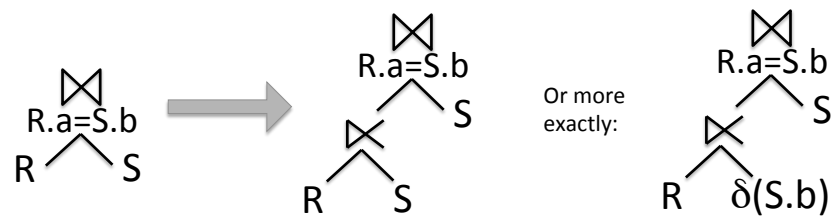
Implementing equi-joins on MapReduce (1) Repartition join

- $R(rID, rVal) \text{ join}(rID = sID) S(sID, sVal)$



Implementing equi-joins on MapReduce (2)

- **Semijoin-based MapReduce join**
- Based on the classical semijoin optimization technique:
 - $R \text{ join } S = (R \text{ semijoin } S) \text{ join } S$

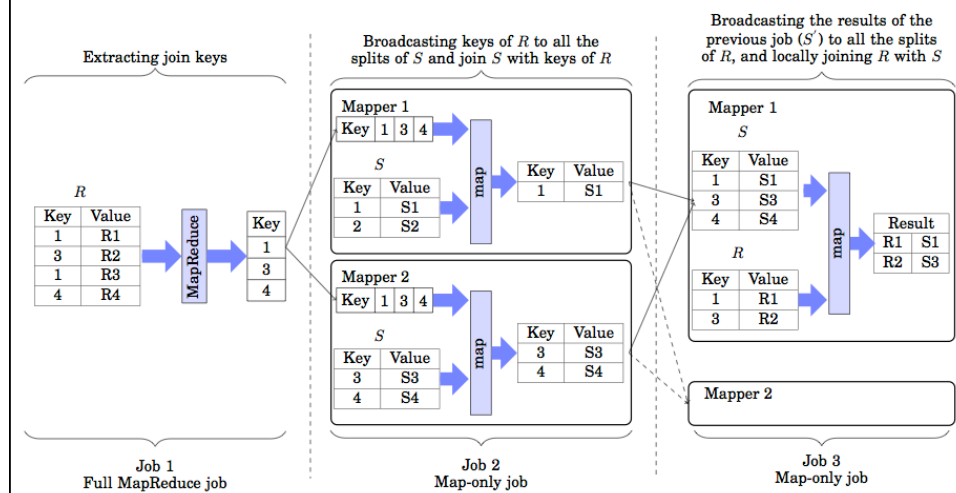


- Useful in distributed settings to reduce transfers: *if the distinct $S.b$ values are smaller than the non-matching R tuples*
- Symmetrical alternative: $R \text{ join } S = R \text{ join } (S \text{ semijoin } R)$

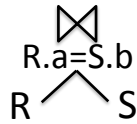
13

Implementing equi-joins on MapReduce (2)

- **Semijoin-based MapReduce join**



Implementing equi-joins on MapReduce (3)



Broadcast (map-only) MapReduce join [Blanas2010]

If $|R| \ll |S|$, broadcast R to all nodes!

- Example: S is a *log* data collection (e.g. log table)
- R is a *reference* table e.g. with user names, countries, age, ...
- Facebook: 6 TB of new log data/day

Map: Join a partition of S with R.

Reduce: nothing (« map-only join »)

15

Implementing equi-joins on MapReduce (4)

- Trojan Join [Dittrich 2010]
- A Map task is sufficient for the join if relations are already **co-partitioned** by the join key
 - The slice of R with a given join key is already next to the slice of S with the same join key
 - This can be achieved by a MapReduce job similar to repartition join but which builds co-partitions at the end



- Useful when the joins can be known in advance (e.g. keys – foreign keys)

16

Implementing binary equi-joins in MapReduce

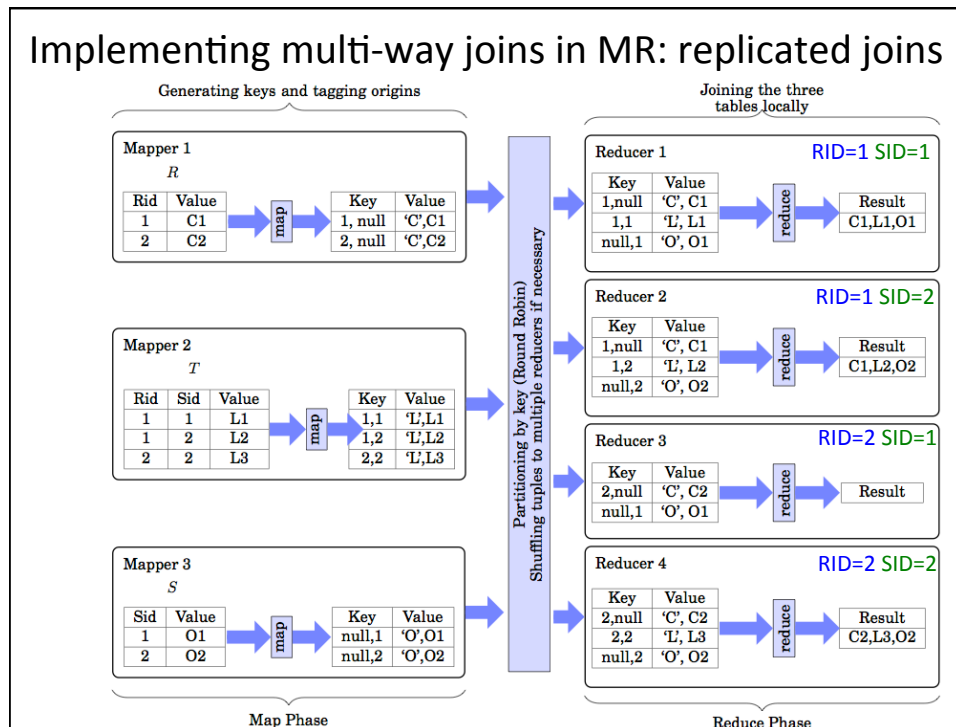
Algorithm	+	-
Repartition Join	Most general	Not always the most efficient
Semijoin-based Join	Efficient when semijoin is selective (has small results)	Requires several jobs, one must first do the semi-join
Broadcast Join	Map-only	One table must be very small
Trojan Join	Map-only	The relations should be co-partitioned

17

Implementing n-ary (« multiway ») join expressions in MapReduce

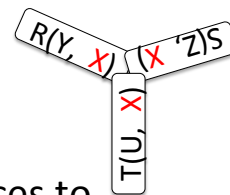
- $R(RID, C) \text{ join } T(RID, SID, O) \text{ join } S(SID, L)$
- « Mega » operator for the whole join expression?...
- Three relations, two join attributes (RID and SID)
- We split the $SIDs$ into Ns groups and the $RIDs$ in Nr groups. Assume $Nr \times Ns$ reducers available.
- Hash T tuples according to a composite key made of the two attributes. Each T tuple goes to one reducer.
- Hash R and S tuples on partial keys ($RID, null$) and ($null, SID$)
- Distribute R and S tuples to each reducer where the non-null component matches (potentially multiple times!)

18



Particular case of multi-way joins: star joins on MapReduce

- Same join attribute in all relations:
 $R(x, y) \text{ join } S(x, z) \text{ join } T(x, u)$



- If N reducers are available, it suffices to partition the space of x values in N
- Then co-partition $R, S, T \rightarrow$ map-only join

Query optimization for MapReduce

- Given a query over relations R_1, R_2, \dots, R_n , how to translate it into a MapReduce program?
 - Use **one replicated join**. Pbm: the space of composite join keys ($Att_1|Att_2|\dots|Att_k$) is limited by the number of reducers \rightarrow may shuffle some tuples to many reducers.
 - Use **n-1 binary joins**
 - Use **n-ary (multiway) joins only**: CliqueSquare (next)

21

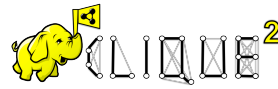
CliqueSquare: flat plans for massively parallel RDF queries

- **Focus**: Build massively parallel **flat** plans for RDF queries by exploiting **n-ary (star)** equality joins.
- **Contributions**:
 - Novel algorithms for exploring the **search space** of logical plans by relying on **n-ary** equality joins
 - **Formal guarantees** regarding the **flatness** of the plans generated by our algorithms
 - Implementation & experimental evaluation on top of **MapReduce**

[Goasdoué, Kaoudi, Manolescu, Quiané, Zampetakis15]

22

CliqueSquare plan



1. Logical query optimization
2. Physical data organization
3. From logical to physical query plans to MapReduce execution

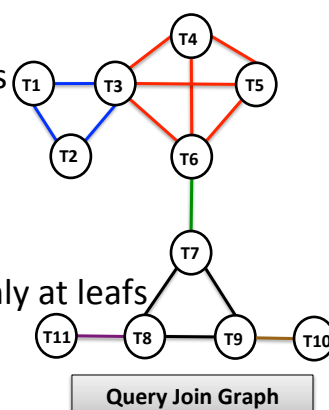
Publication, code at:

<https://team.inria.fr/oak/projects/cliquesquare/>

23

Query optimization overview

- Left deep plans with binary joins
- Left deep plans with n-ary joins
- Bushy plans with binary joins
- Bushy plans with n-ary joins only at leafs
- Bushy plans with n-ary joins



24

Query plans on MapReduce

– Left deep plans with binary joins:

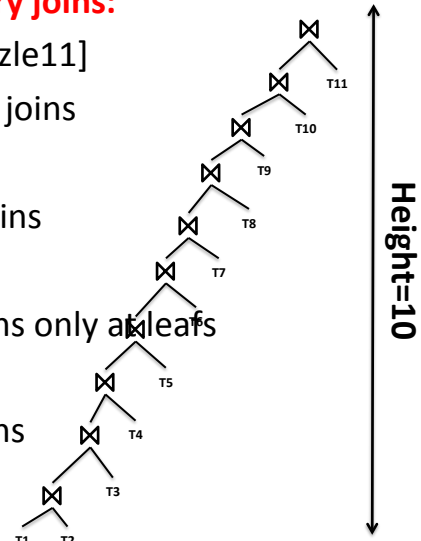
[Olston08][Rohloff10][Schatzle11]

– Left deep plans with n-ary joins

– Bushy plans with binary joins

– Bushy plans with n-ary joins only at leafs

– Bushy plans with n-ary joins



25

Query optimization overview

– Left deep plans with binary joins

[Olston08][Rohloff10][Schatzle11]

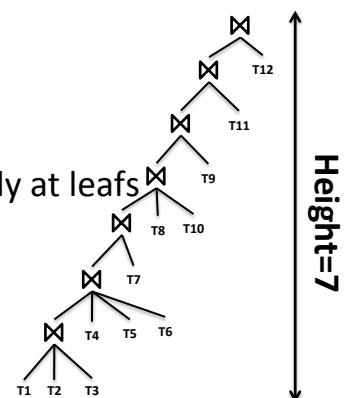
– Left deep plans with n-ary joins:

[Papailiou13]

– Bushy plans with binary joins

– Bushy plans with n-ary joins only at leafs

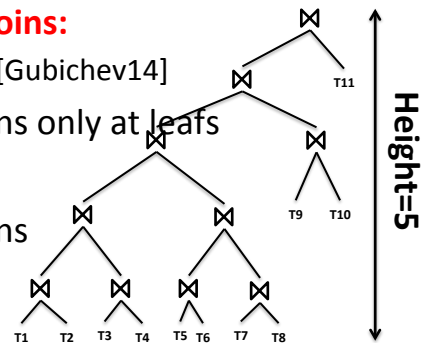
– Bushy plans with n-ary joins



26

Query optimization overview

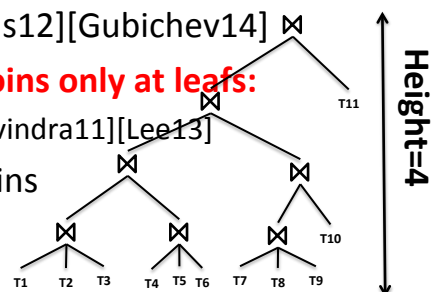
- Left deep plans with binary joins
[Olston08][Rohloff10][Schatzle11]
- Left deep plans with n-ary joins
[Papailiou13]
- **Bushy plans with binary joins:**
[Neumann10][Tsialiamanis12][Gubichev14]
- Bushy plans with n-ary joins only at leaves
- Bushy plans with n-ary joins



27

Query optimization overview

- Left deep plans with binary joins
[Olston08][Rohloff10][Schatzle11]
- Left deep plans with n-ary joins
[Papailiou13]
- Bushy plans with binary joins
[Neumann10][Tsialiamanis12][Gubichev14]
- **Bushy plans with n-ary joins only at leaves:**
[Wu11][Kim11][Huang11][Ravindra11][Lee13]
- Bushy plans with n-ary joins



28

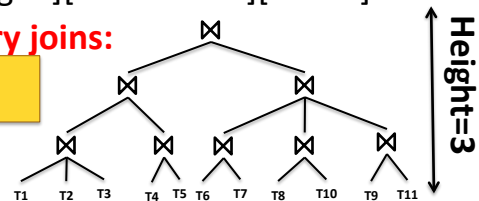
Query optimization overview

- Left deep plans with binary joins
[Olston08][Rohloff10][Schatzle11]
- Left deep plans with n-ary joins
[Papailiou13]
- Bushy plans with binary joins
[Neumann10][Tsialiamanis12][Gubichev14]
- Bushy plans with n-ary joins only at leafs
[Wu11][Kim11][Huang11][Ravindra11][Lee13]

– **Bushy plans with n-ary joins:**

[Husain11]

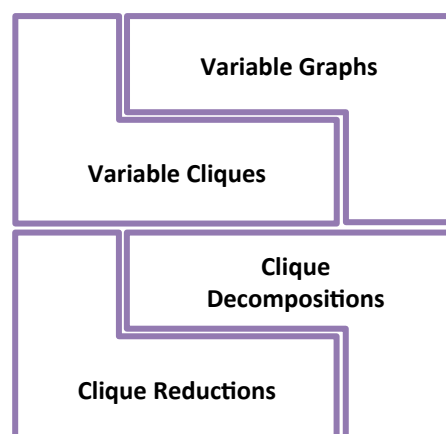
CliqueSquare
[Goasdoué15]



29



CliqueSquare optimization algorithm: Components



30

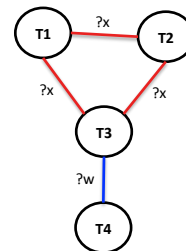
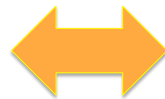


CliqueSquare algorithm: Variable Graphs

- Represent incoming queries and intermediary relations

```
SELECT ?x ?y
WHERE {
  T1: ?x takesCourse ?y .
  T2: ?x member ?z .
  T3: ?w advisor ?x .
  T4: ?w name ?u .}
```

Query



Variable graph

31

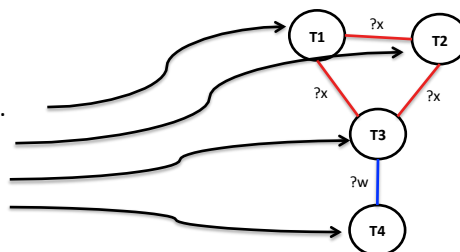


CliqueSquare algorithm: Variable Graphs

- Represent incoming queries and intermediary relations

```
SELECT ?x ?y
WHERE {
  T1: ?x takesCourse ?y .
  T2: ?x member ?z .
  T3: ?w advisor ?x .
  T4: ?w name ?u .}
```

Query



Variable graph

Each **triple pattern** corresponds to a **node** in the graph

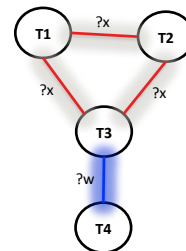
32



CliqueSquare algorithm: Variable Graphs

- Represent incoming queries and intermediary relations

SELECT ?x ?y
WHERE {
T1: ?x takesCourse ?y .
T2: ?x member ?z .
T3: ?w advisor ?x .
T4: ?w name ?u .}



Query

Variable graph

Nodes are connected with an **edge** if they share a **variable**

33

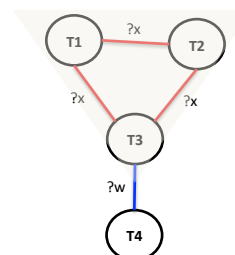


CliqueSquare algorithm: Variable Cliques

- Model n-ary joins among relations

SELECT ?x ?y
WHERE {
T1: ?x takesCourse ?y .
T2: ?x member ?z .
T3: ?w advisor ?x .
T4: ?w name ?u .}

**Maximal
Clique of '?x'**



Query

Variable graph

A **variable clique** is a set of nodes which are connected to each other with the **same edge**

34



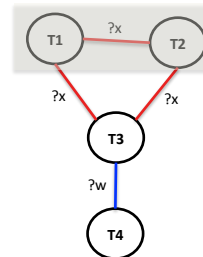
CLIQUE²

CliqueSquare algorithm: Variable Cliques

- Model n-ary joins among relations

```
SELECT ?x ?y
WHERE {
  T1: ?x takesCourse ?y .
  T2: ?x member ?z .
  T3: ?w advisor ?x .
  T4: ?w name ?u .}
```

Partial
Clique of '?x'



Query

Variable graph

A **variable clique** is a set of nodes which are connected to each other with the **same edge**

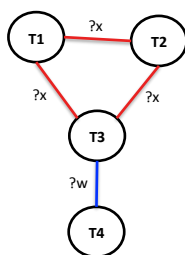
35



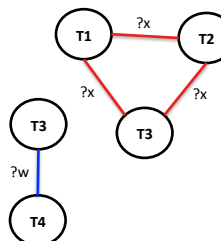
CLIQUE²

CliqueSquare algorithm: Clique Decompositions

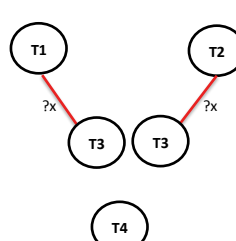
- Correspond to identifying partial results to be joined



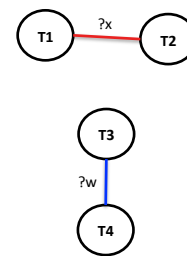
Variable graph



Decomposition 1



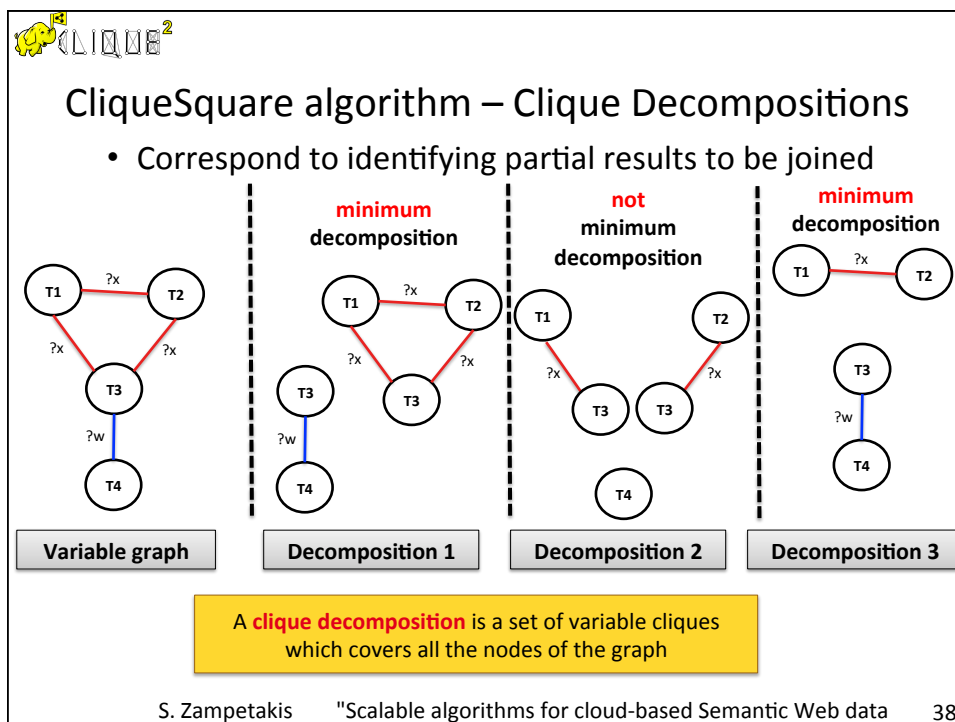
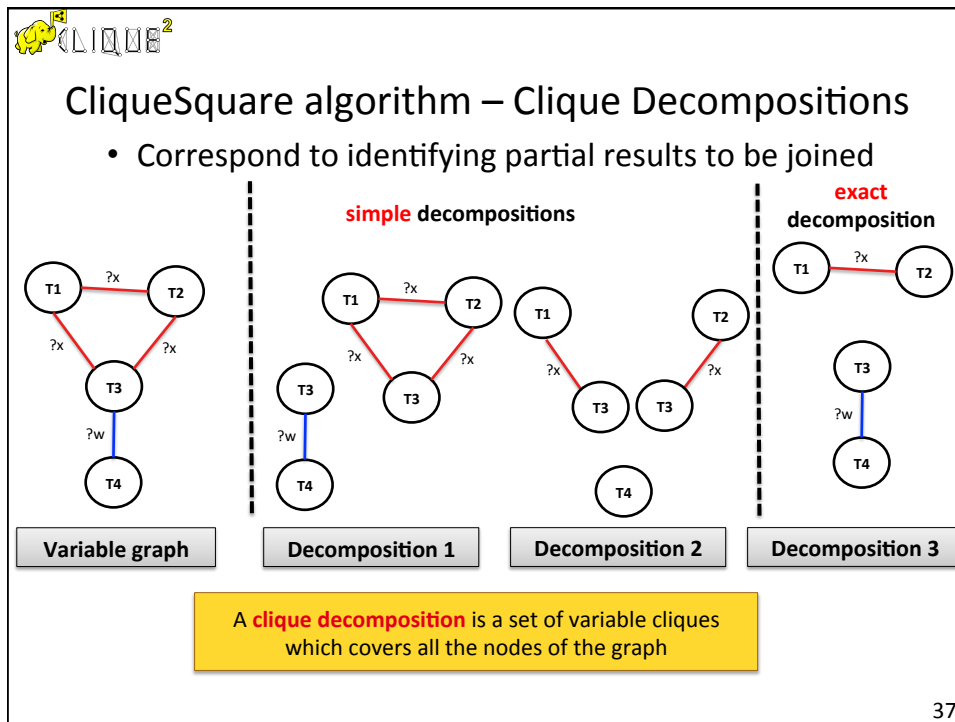
Decomposition 2



Decomposition 3

A **clique decomposition** is a set of variable cliques which covers all the nodes of the graph

36



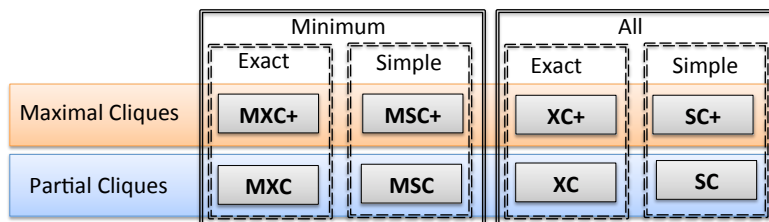


CliqueSquare algorithm – Clique Decompositions

- Correspond to identifying partial results to be joined

The clique decomposition is **not unique**

- We identify **eight** decomposition alternatives based on:
 - Type of variable cliques: maximal or partial
 - Type of cover: simple or exact
 - Cover size: minimum or not

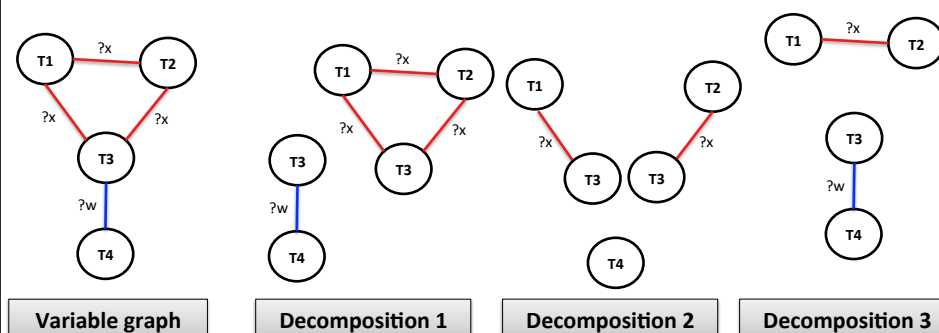


39



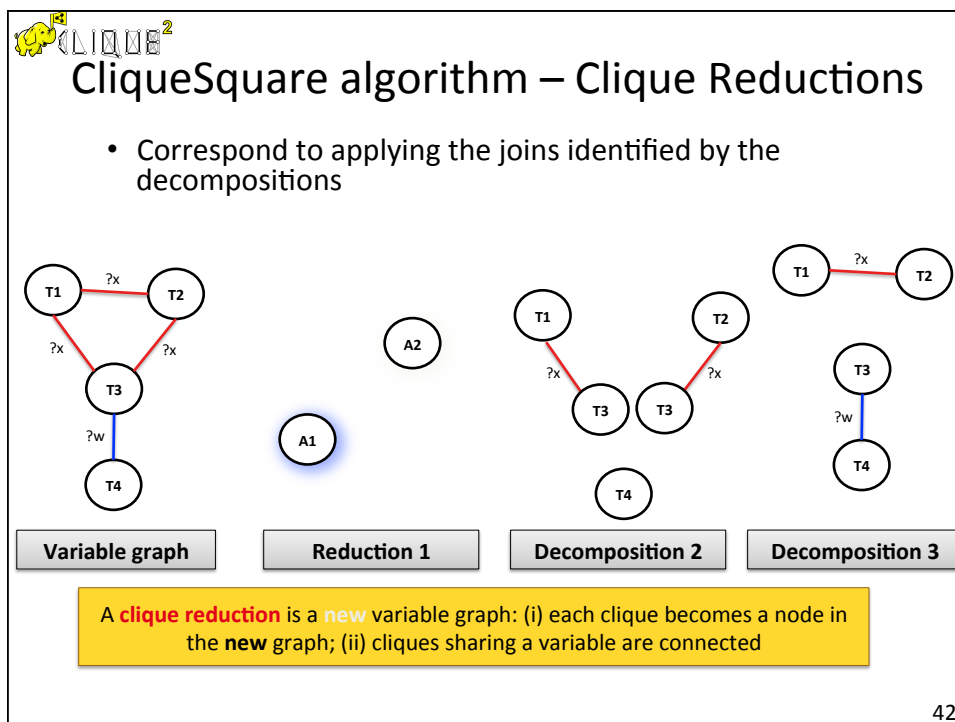
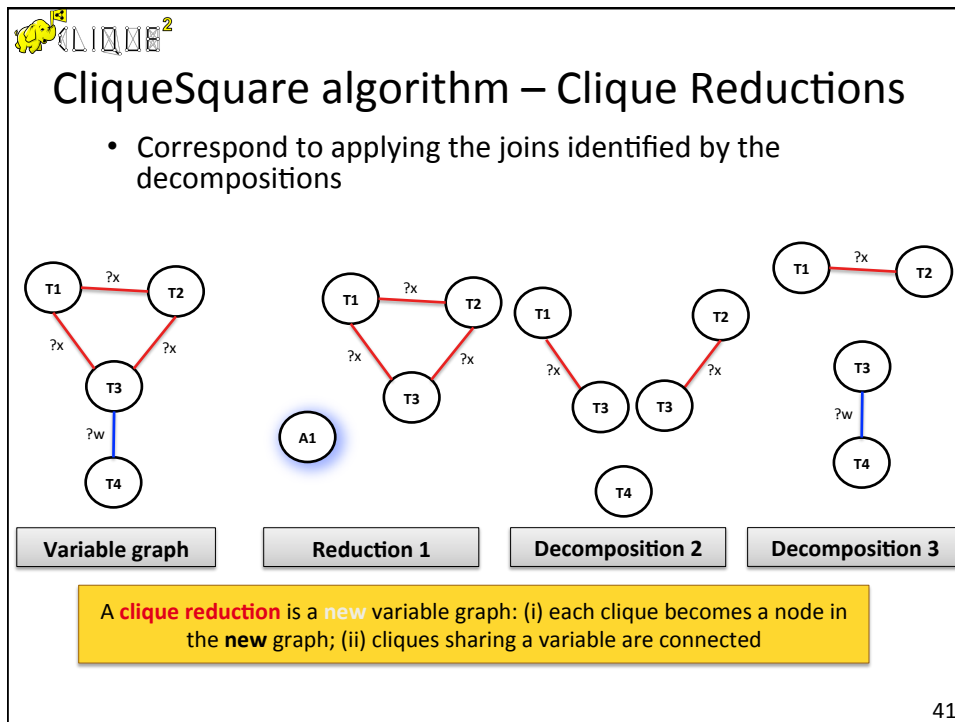
CliqueSquare algorithm – Clique Reductions

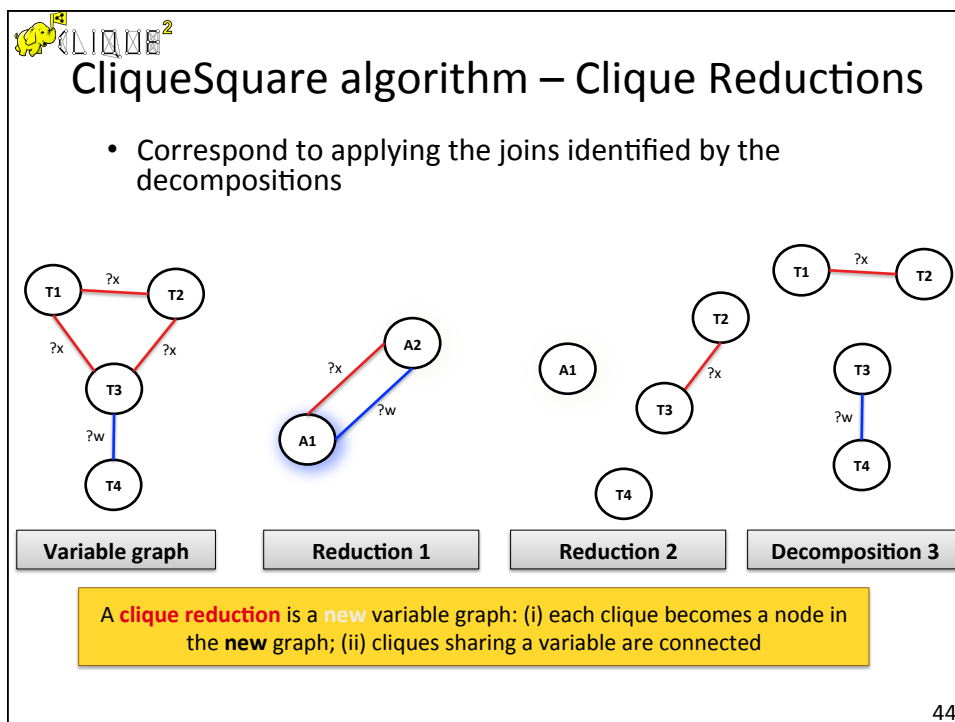
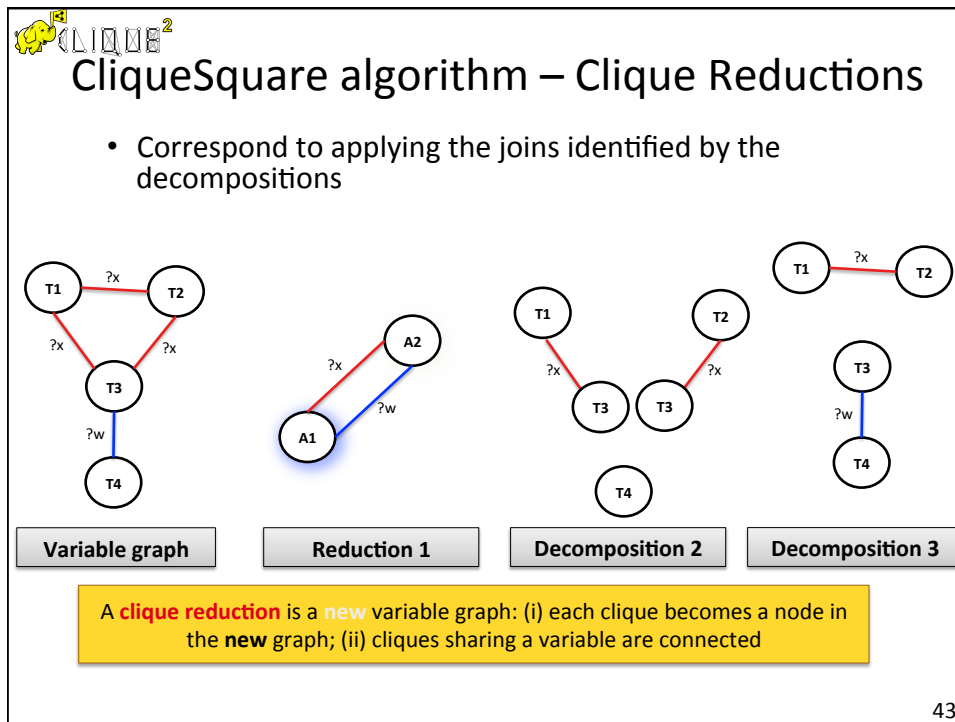
- Correspond to applying the joins identified by the decompositions

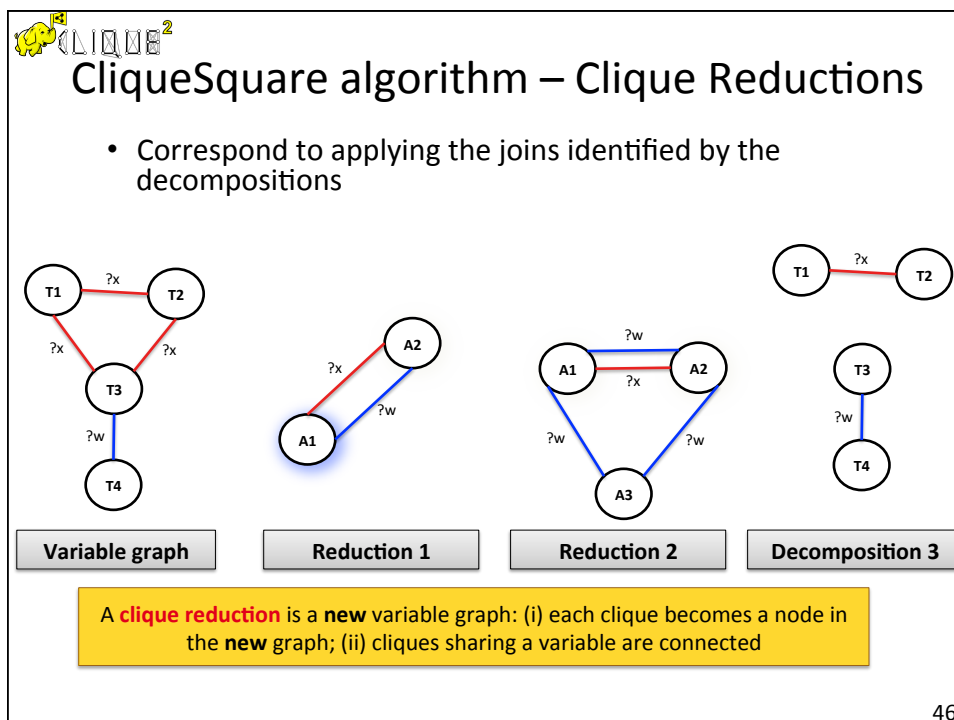
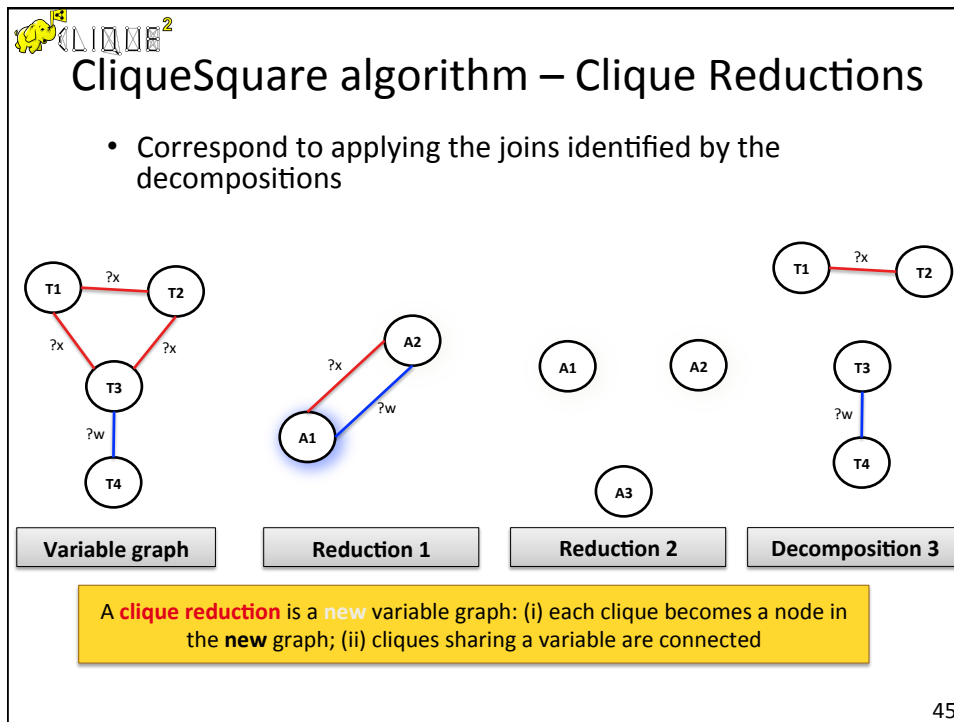


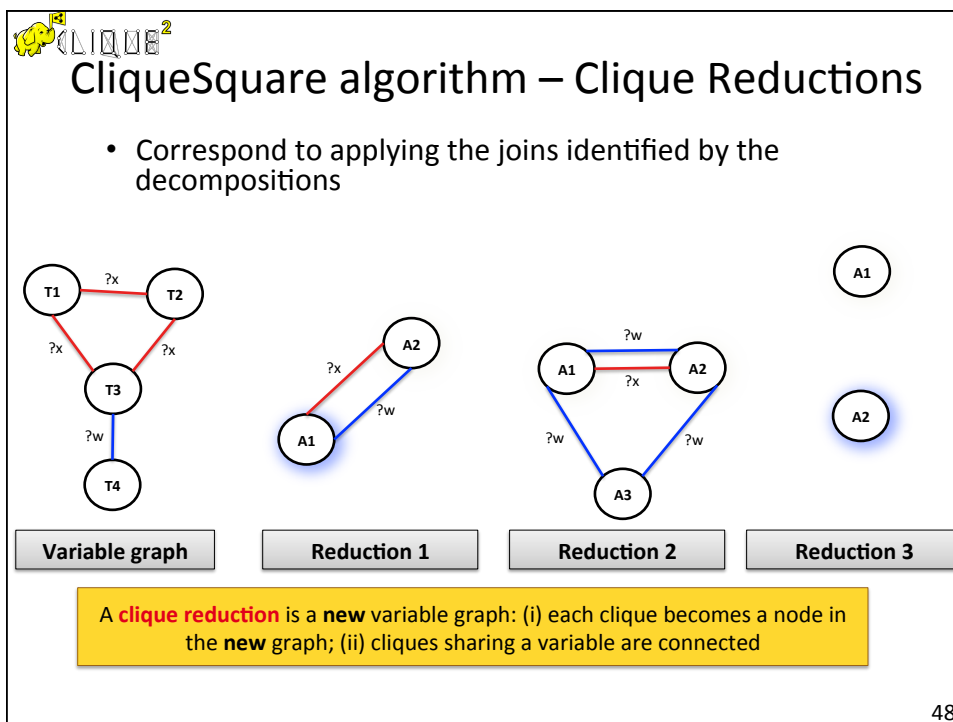
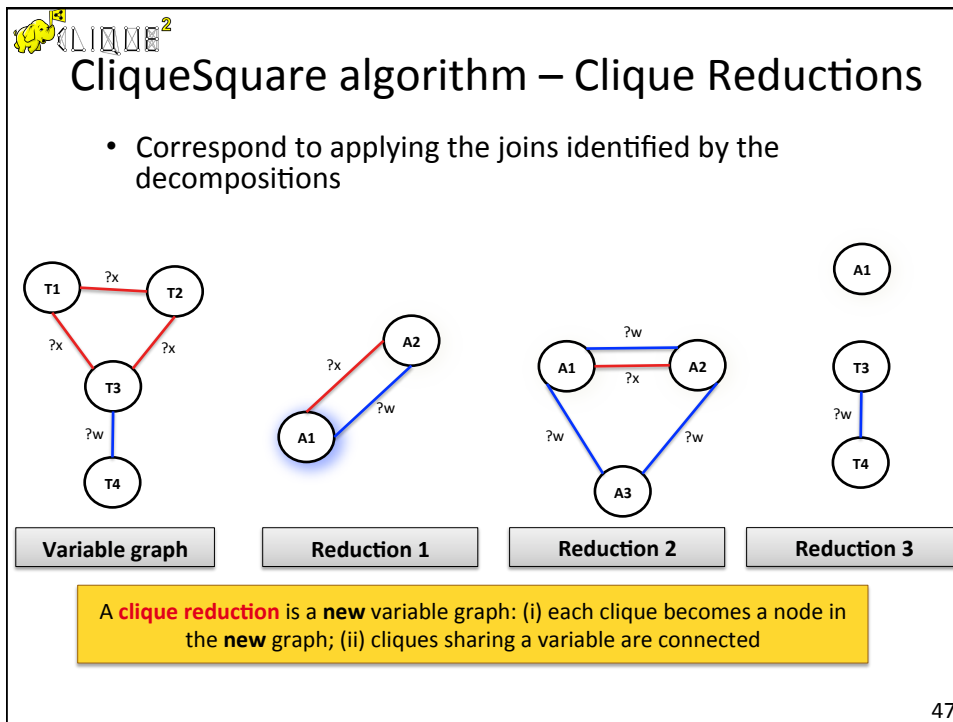
A **clique reduction** is a **new** variable graph: (i) each clique becomes a node in the **new** graph; (ii) cliques sharing a variable are connected


40



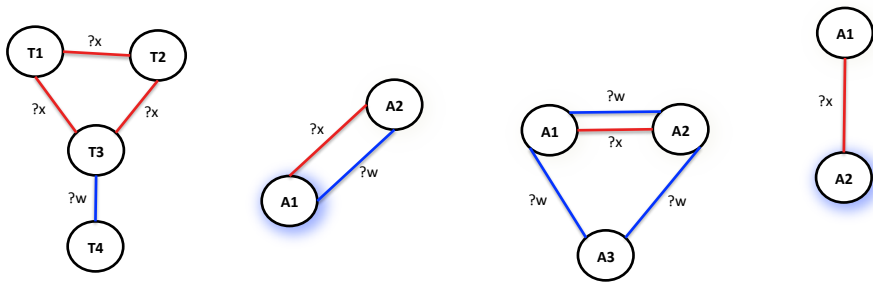






 **CliqueSquare algorithm – Clique Reductions**


- Correspond to applying the joins identified by the decompositions



Variable graph **Reduction 1** **Reduction 2** **Reduction 3**

A **clique reduction** is a **new** variable graph: (i) each clique becomes a node in the **new** graph; (ii) cliques sharing a variable are connected

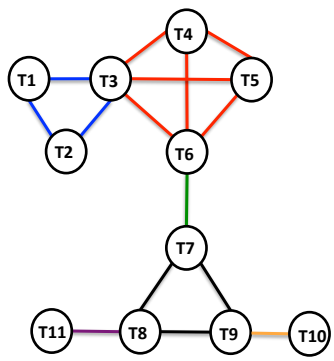
49

 **CliqueSquare algorithm - Example**

- Create the variable graph for the given query

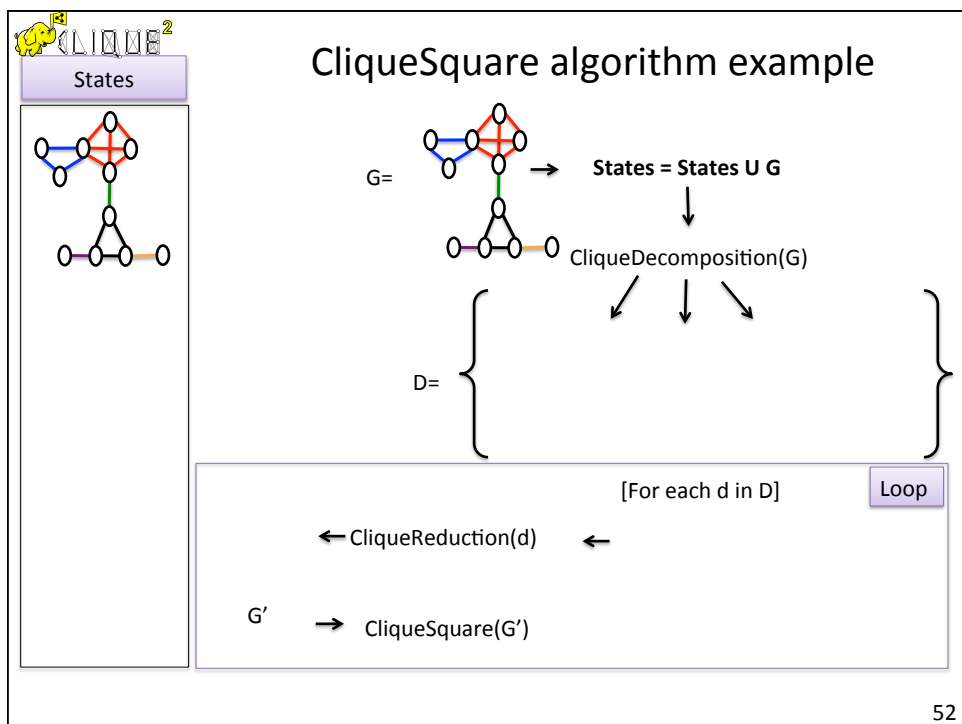
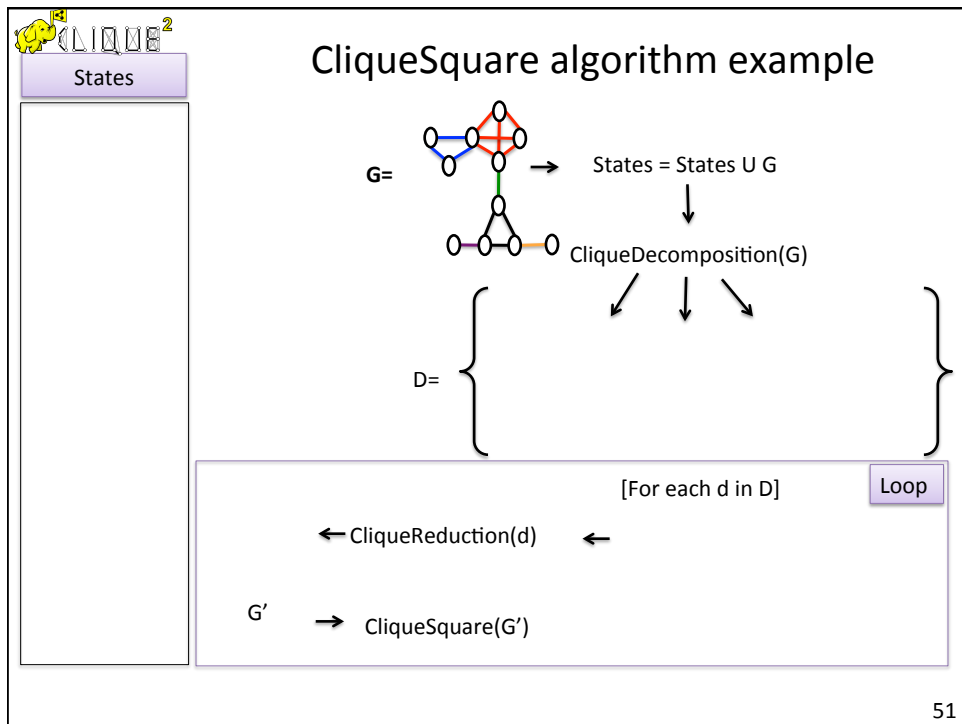
```

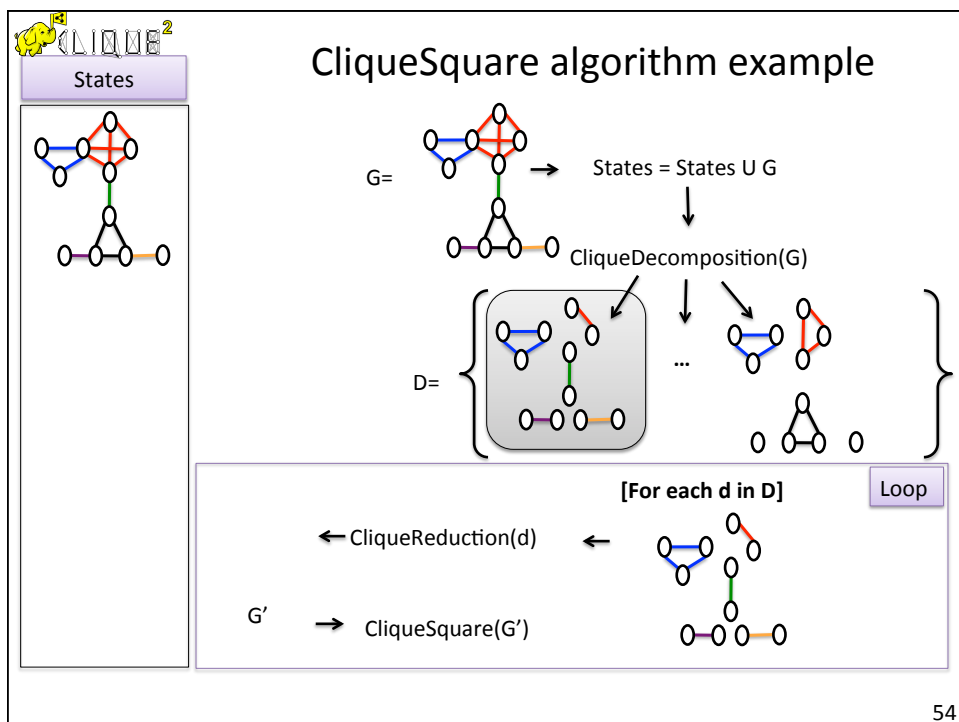
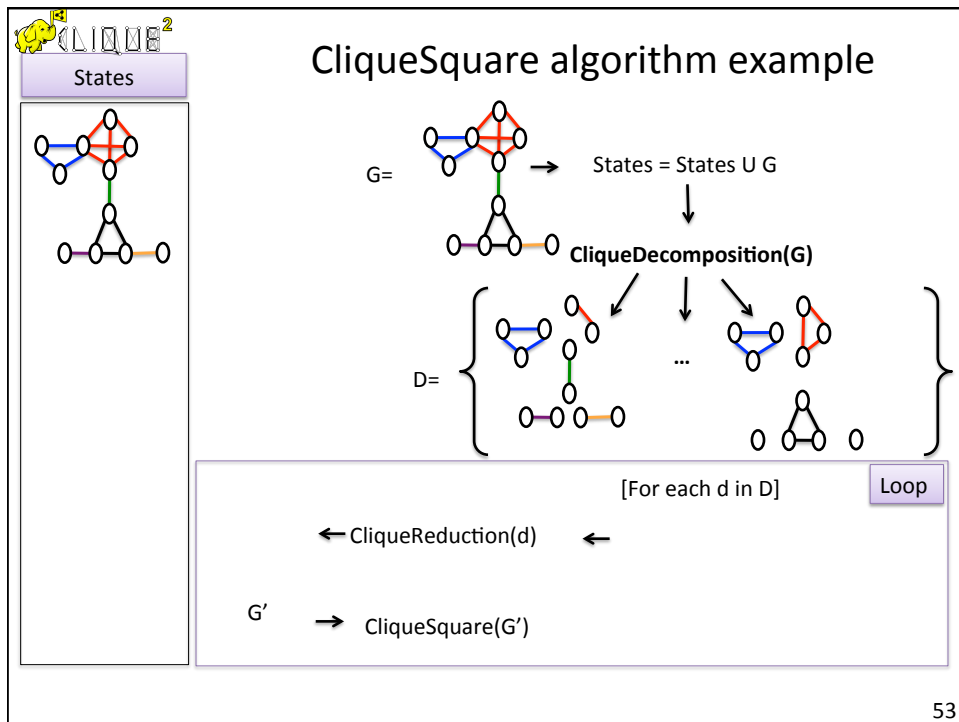
SELECT ?x ?y
WHERE {
  T1: ?w :prop1 <C1> .
  T2: ?w :prop2 <C2> .
  T3: ?w :prop3 ?x .
  T4: ?x :prop4 <C3> .
  T5: ?x :prop5 <C4> .
  T6: ?x :prop6 ?z .
  T7: ?z :prop7 ?f .
  T8: ?f :prop8 ?y .
  T9: ?f :prop9 ?h .
  T10: <C5> :prop10 ?h .
  T11: ?y :prop11 <C6> .}
  
```

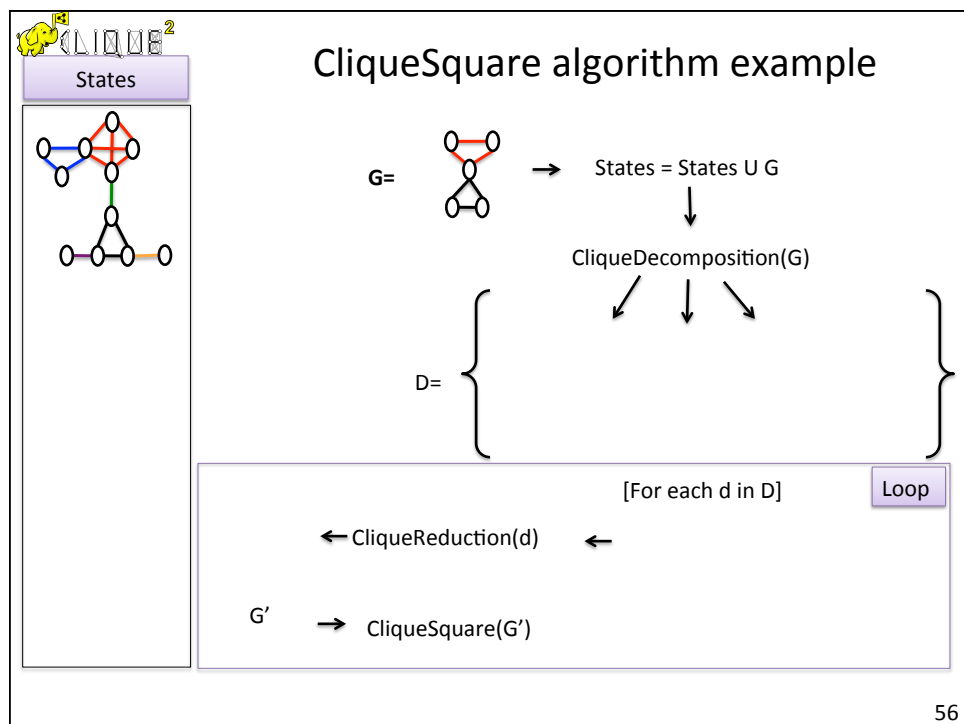
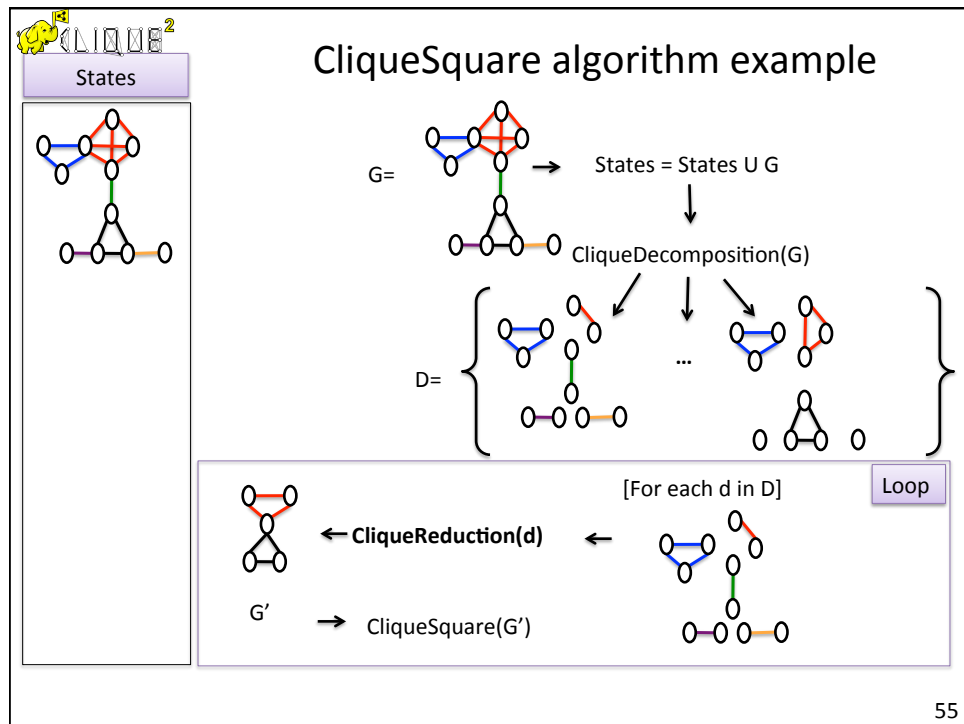


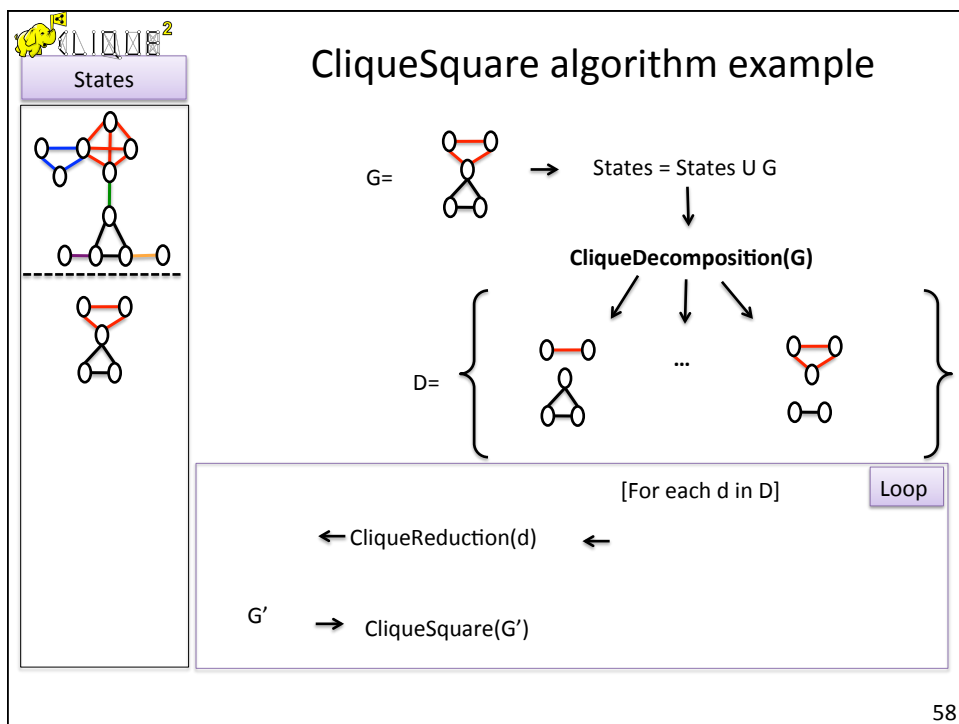
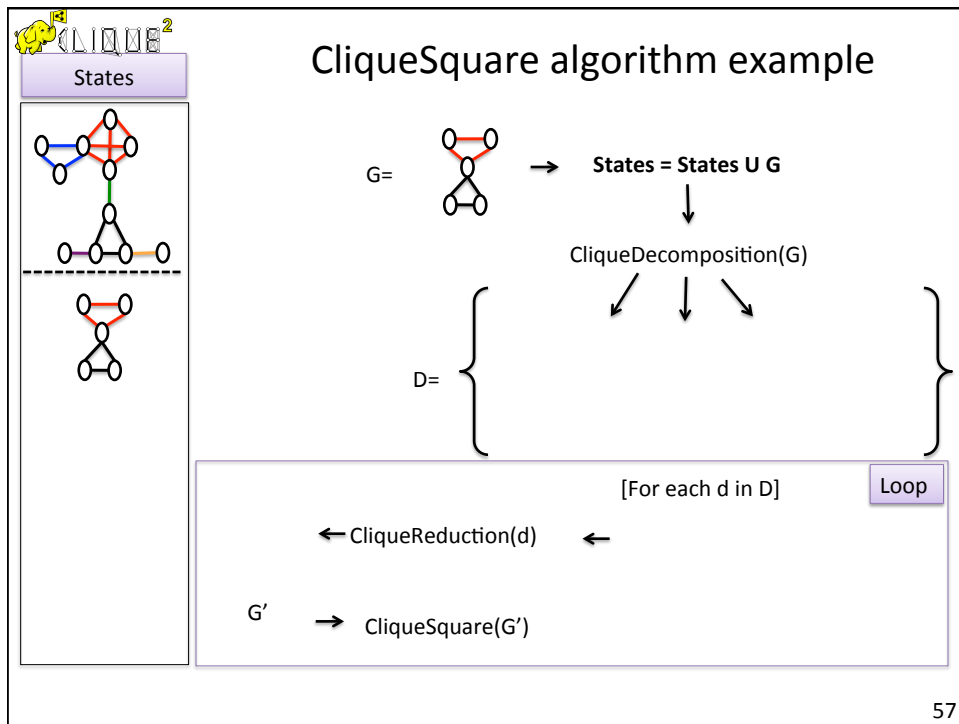
Query **Variable Graph**

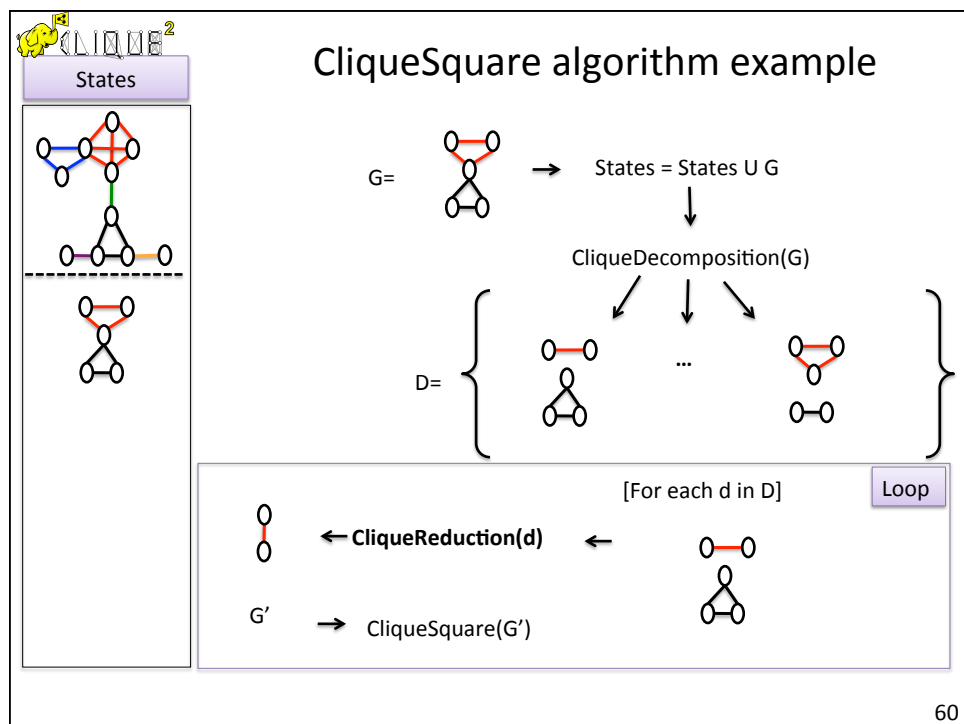
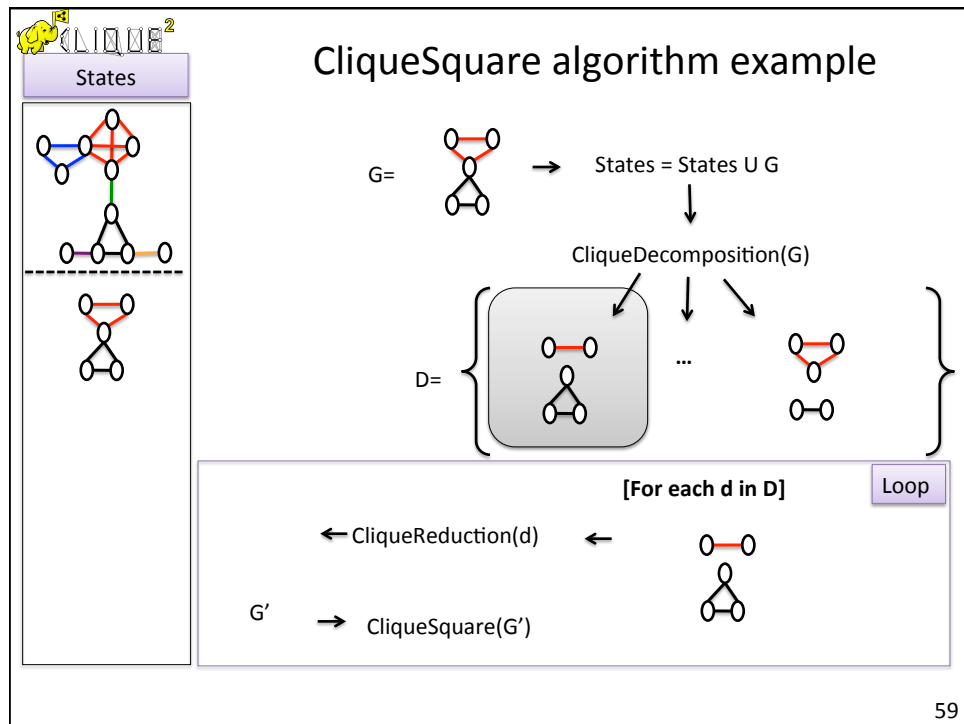
50

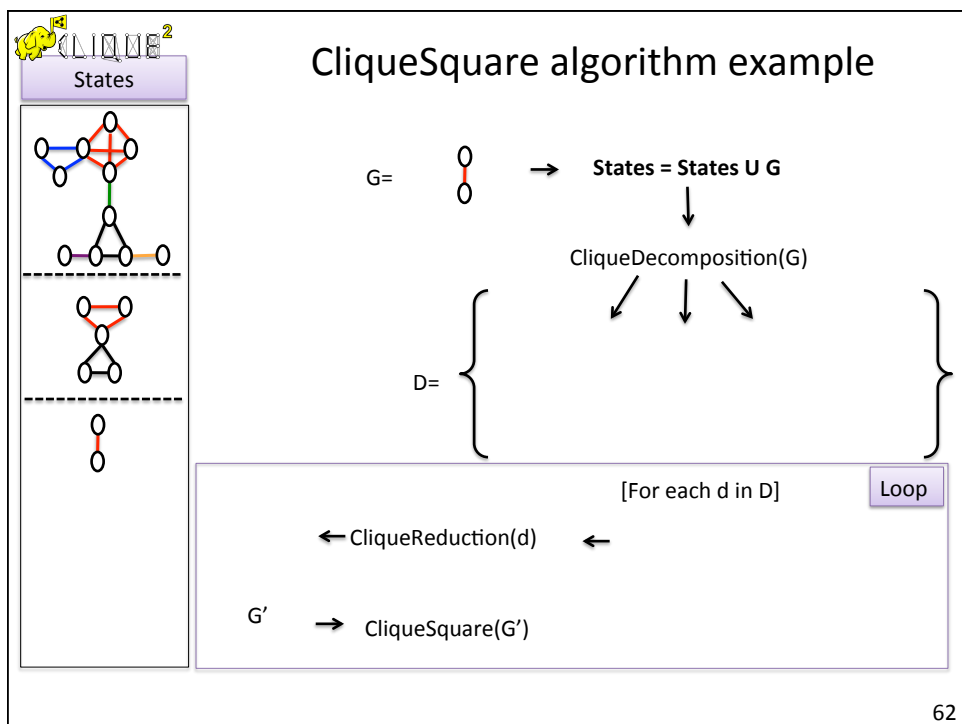
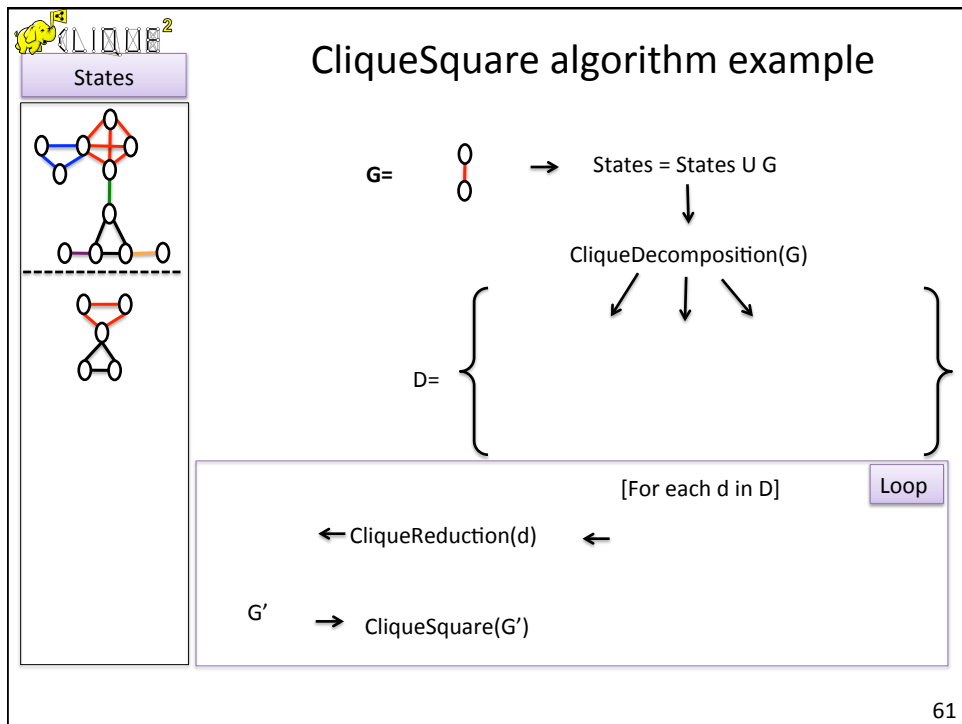


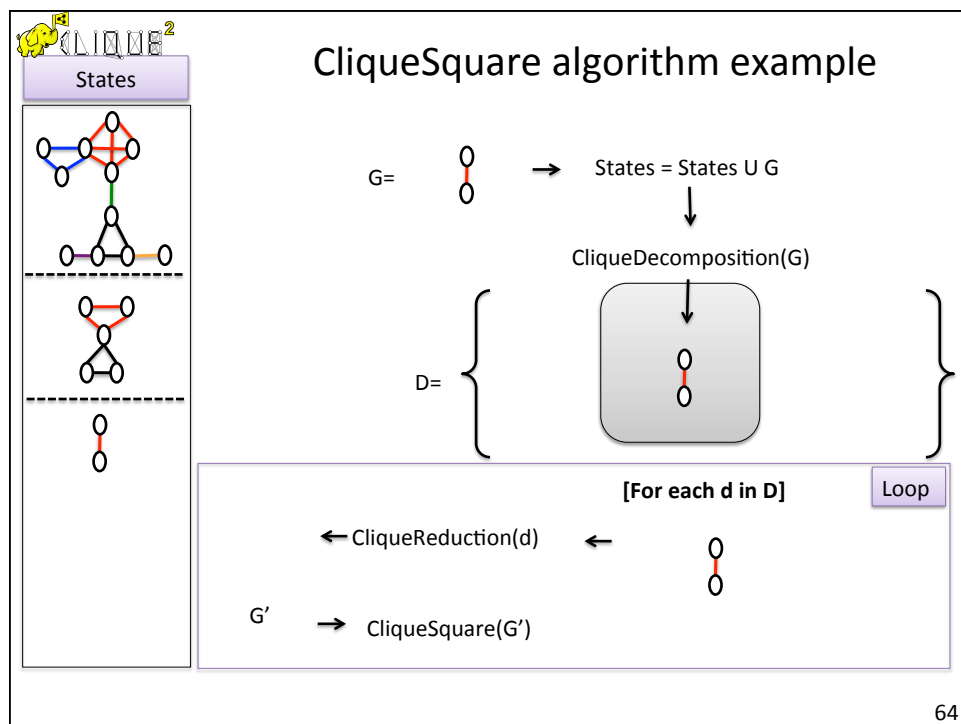
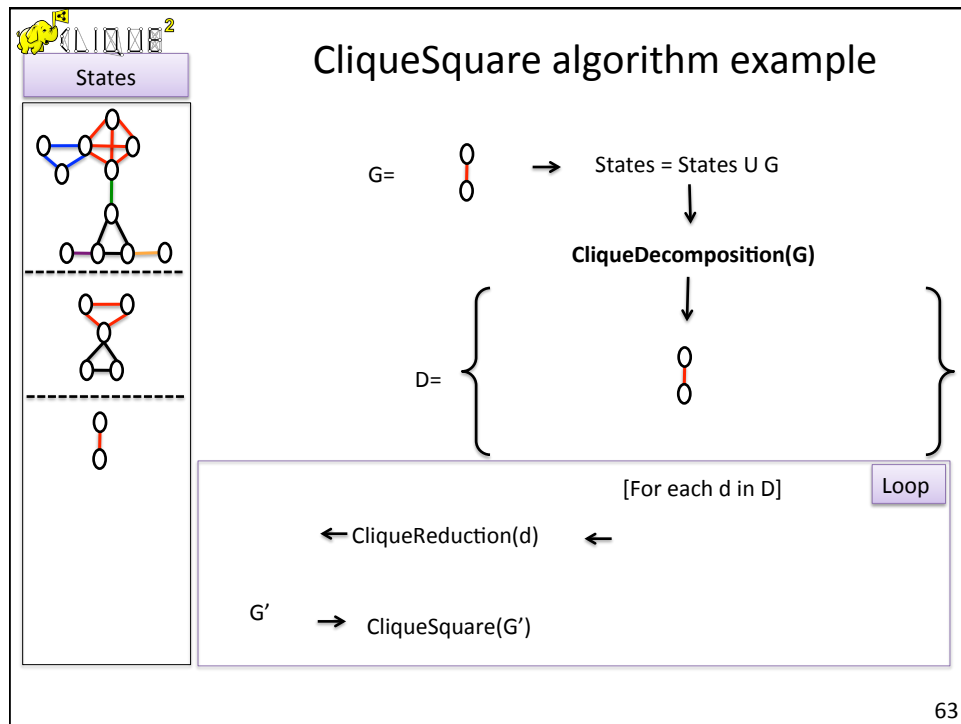


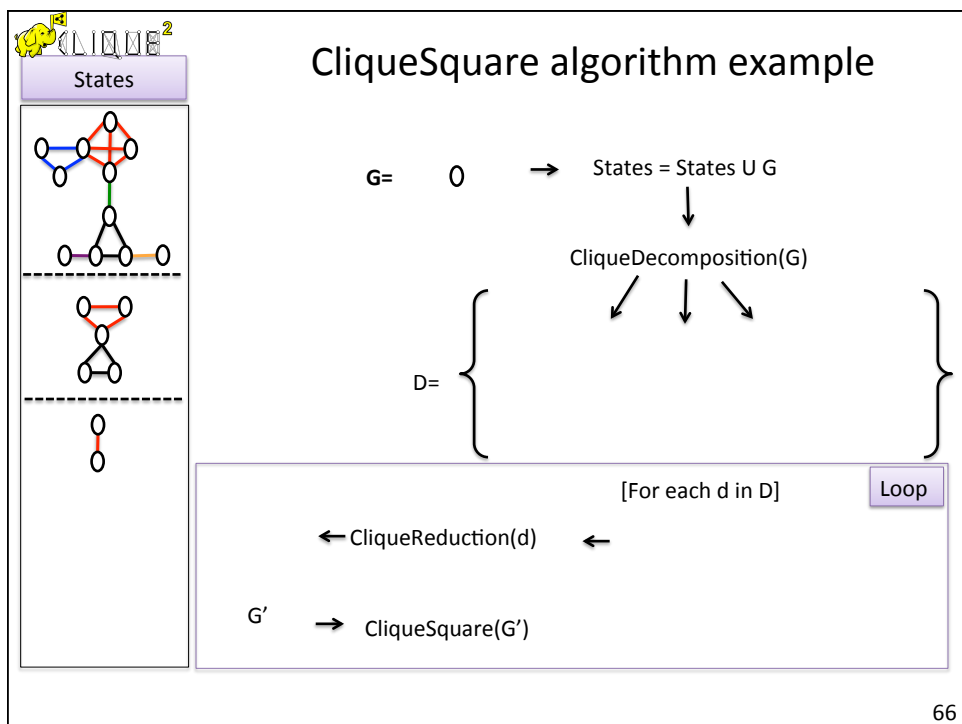
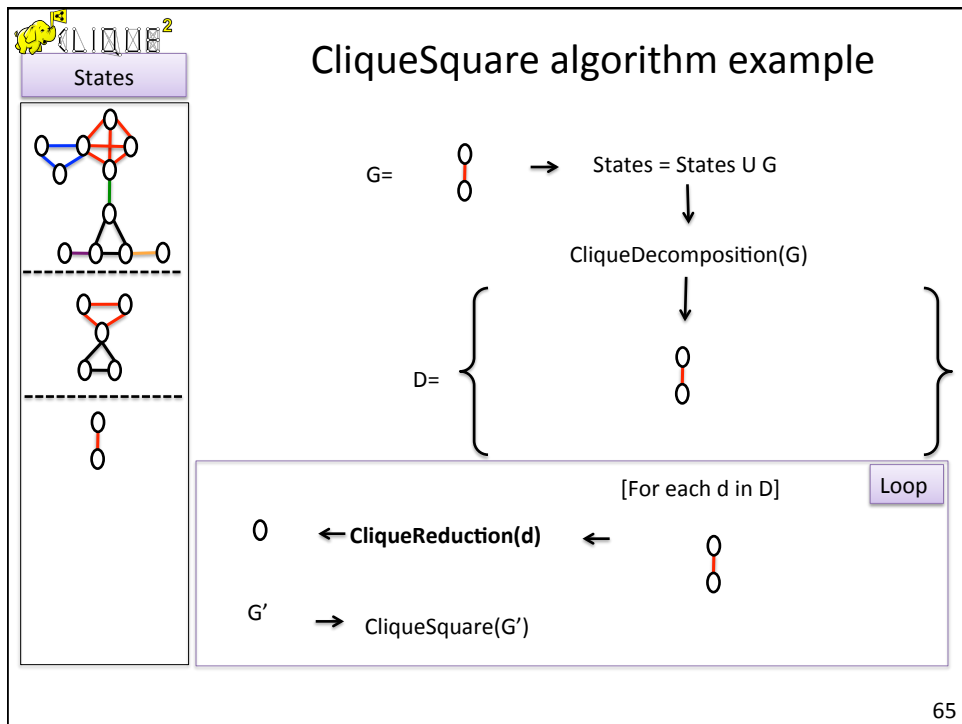


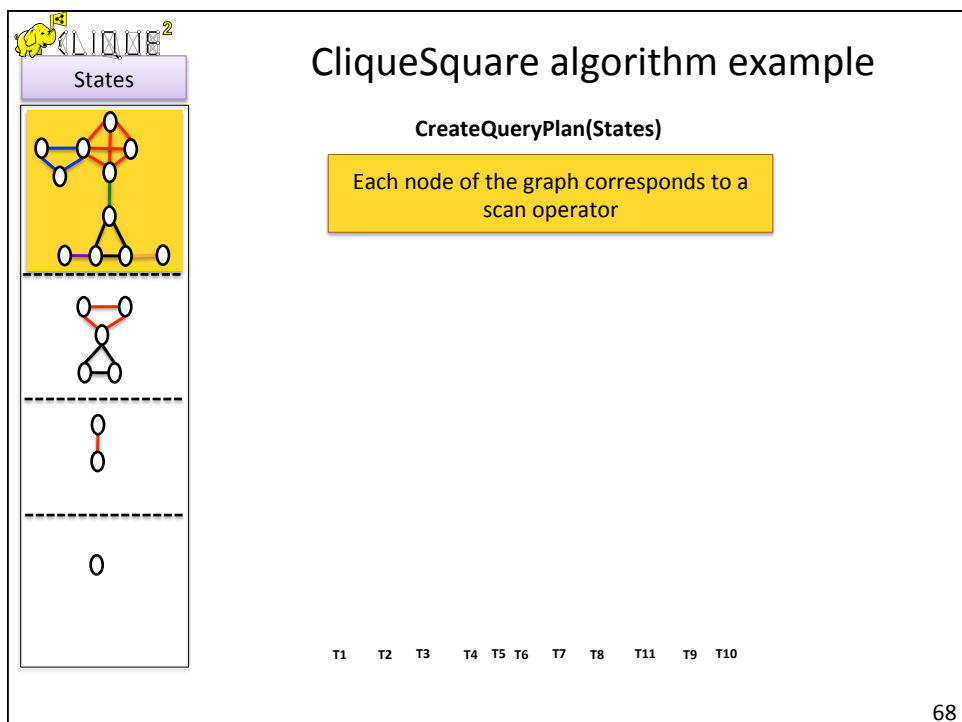
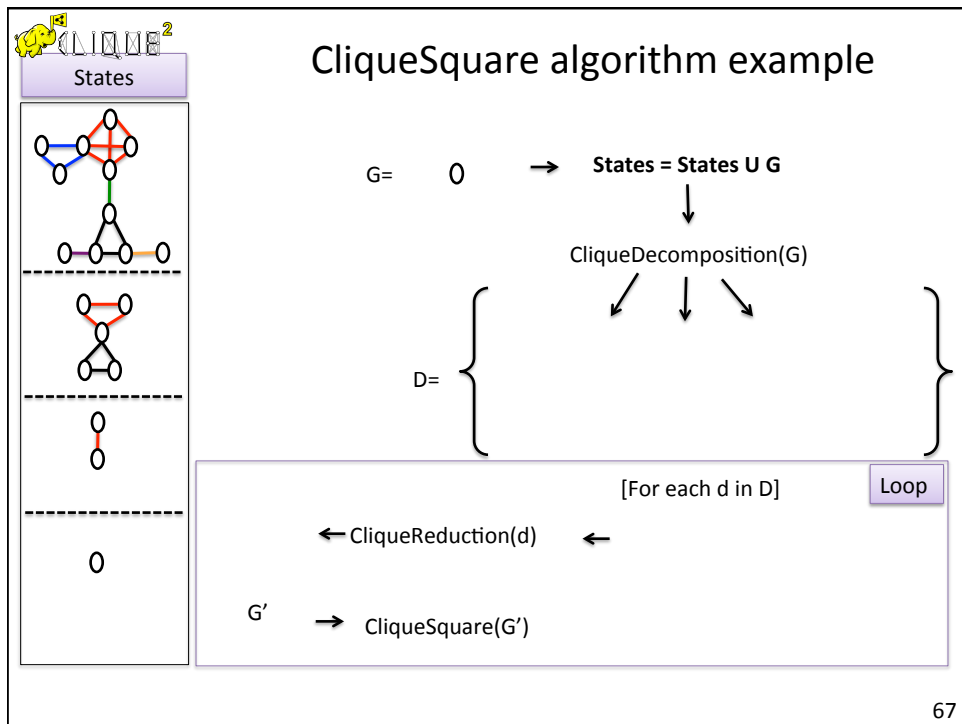







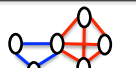





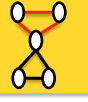





States









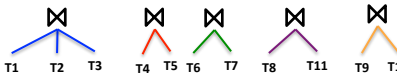
0

CliqueSquare algorithm example


CreateQueryPlan(States)

Each **node** of the graph corresponds to a **clique** of nodes of the previous graph.


Introduce the **join** operators.





69




States









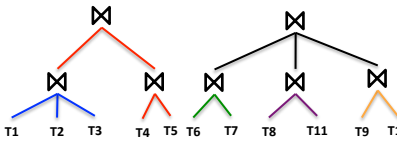
0

CliqueSquare algorithm example


CreateQueryPlan(States)

Each **node** of the graph corresponds to a **clique** of nodes of the previous graph.

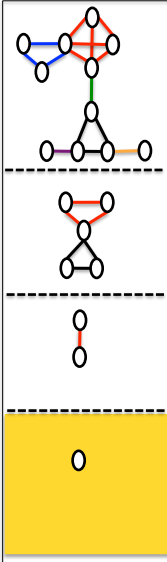
Introduce the **join** operators.



70



States

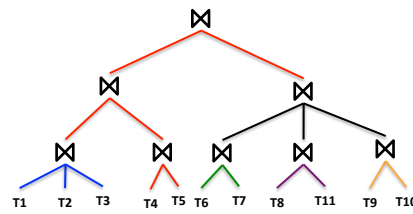


CliqueSquare algorithm example


CreateQueryPlan(States)

Each **node** of the graph corresponds to a **clique** of nodes of the previous graph.

Introduce the **join** operators.

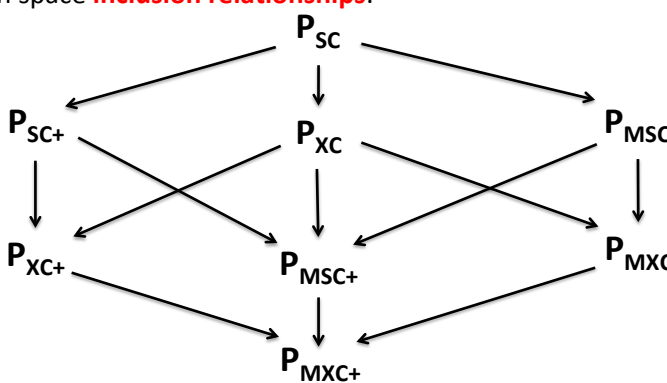


71



CliqueSquare algorithm – Logical plans

- The choice of the decomposition method (m) affects the production of plans - **Plan space** (P_m)
- 8 CliqueSquare variants with 8 plan spaces
- Plan space **inclusion relationships**:



72



CliqueSquare algorithm – Logical plans

- **Height-Optimal (Flat)** plans: having the least **height**
- Decomposition variants are categorized based on height optimality into:

- **HO-Complete:** **all** height optimal plans for a query q

SC

- **HO-Partial:** at **least one** height optimal plan for a query q

MSC+

SC+

MSC

- **HO-Lossy:** possibly **no** height optimal plan for a query q

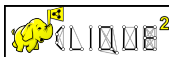
MXC

MXC+

XC+

XC

73



CliqueSquare algorithm – Logical plans

- **Height-Optimal (Flat)** plans: having the least **height**
- Decomposition variants are categorized based on height optimality into:

- **HO-Complete:** **all** height optimal plans for a query q

SC

- **HO-Partial:** at **least one** height optimal plan for a query q

MSC+

SC+

MSC

- **HO-Lossy:** possibly **no** height optimal plan for a query q

MXC

MXC+

XC+

XC

74



Data organization within CliqueSquare

1. Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db

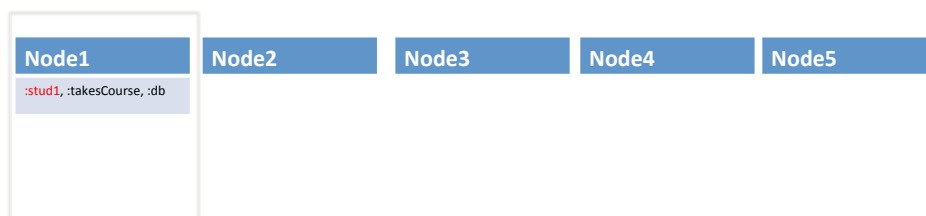


Data organization within CliqueSquare

1. Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db

Hash triple by **subject**



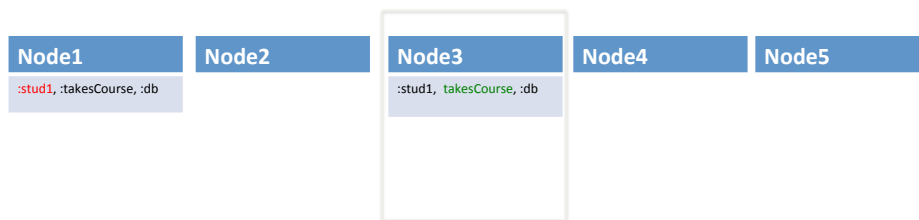


Data organization within CliqueSquare

1. Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db

Hash triple by **property**

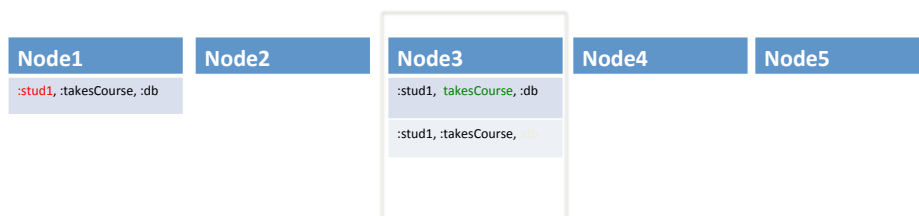


Data organization within CliqueSquare

1. Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db

Hash triple by **object**

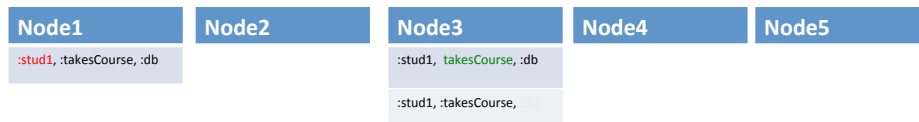




Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db

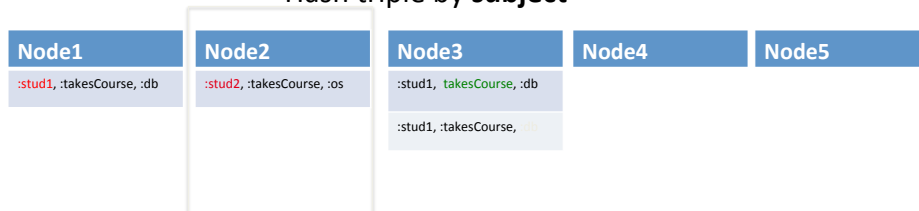


Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os

Hash triple by **subject**



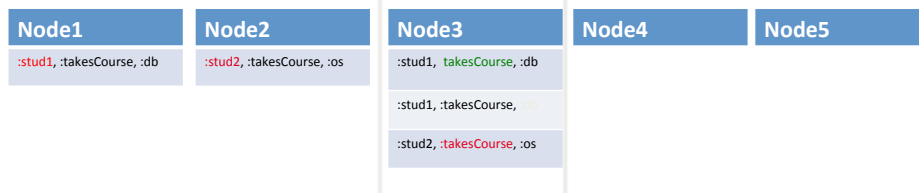


Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os

Hash triple by **property**

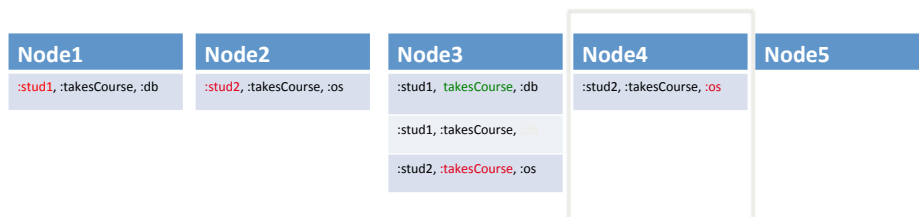


Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os

Hash triple by **object**





Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os
t3	:prof1	:advisor	:stud1
t4	:prof2	:advisor	:stud2

Node1	Node2	Node3	Node4	Node5
:stud1, :takesCourse, :db	:stud2, :takesCourse, :os	:stud1, :takesCourse, :db :stud1, :takesCourse, :os :stud2, :takesCourse, :os	:stud2, :takesCourse, :os	



Data organization within CliqueSquare

- Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os
t3	:prof1	:advisor	:stud1
t4	:prof2	:advisor	:stud2

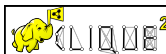
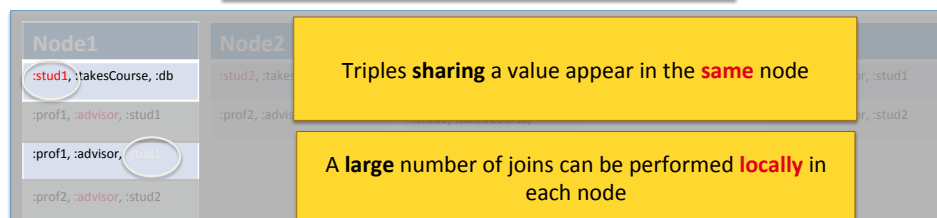
Node1	Node2	Node3	Node4	Node5
:stud1, :takesCourse, :db :prof1, :advisor, :stud1 :prof1, :advisor, :stud1 :prof2, :advisor, :stud2	:stud2, :takesCourse, :os :prof2, :advisor, :stud2	:stud1, :takesCourse, :db :stud1, :takesCourse, :os :stud2, :takesCourse, :os	:stud2, :takesCourse, :os	:prof1, :advisor, :stud1 :prof2, :advisor, :stud2



Data organization within CliqueSquare

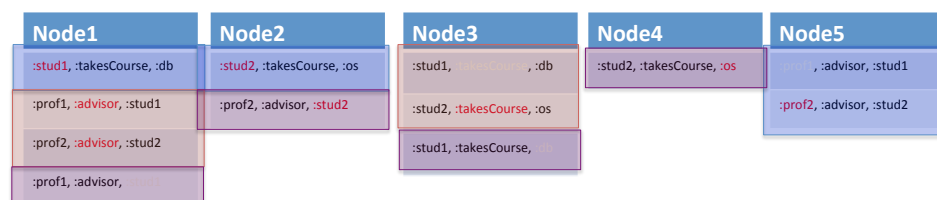
1. Each triple is partitioned based on its **subject**, **property** & **object**

tid	Subject	Property	Object
t1	:stud1	:takesCourse	:db
t2	:stud2	:takesCourse	:os
t3	:prof1	:advisor	:stud1
t4	:prof2	:advisor	:stud2



Data organization within CliqueSquare

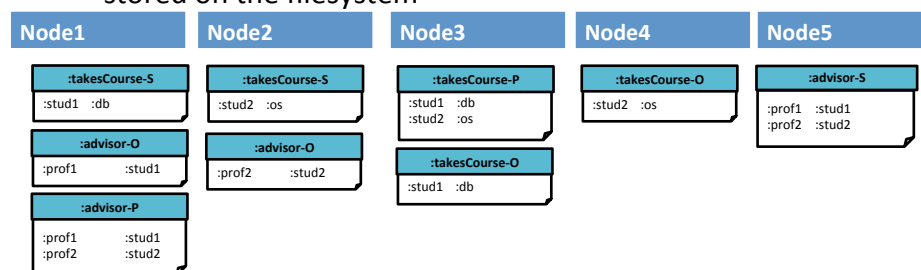
1. Each triple is partitioned based on its **subject**, **property** & **object**
2. The triples are grouped into **3** partitions (**subject**, **property**, **object**) based on the **partition attribute**





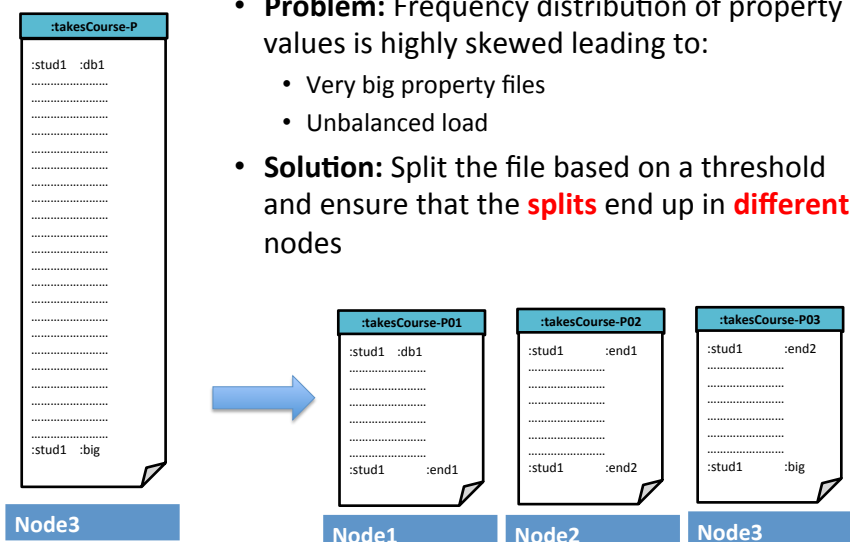
Data organization within CliqueSquare

1. Each triple is partitioned based on its **subject**, **property** & **object**
2. The triples are grouped into **3** partitions (subject, property, object) based on the **partition attribute**
3. The triples inside **each** partition are grouped by **property** and stored on the filesystem



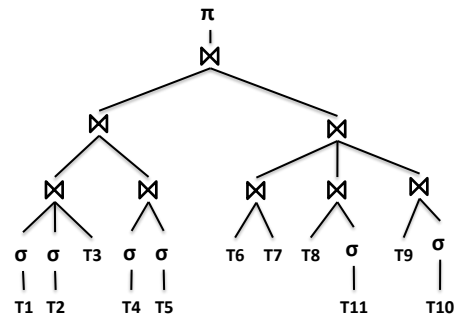
Handling property skew

- **Problem:** Frequency distribution of property values is highly skewed leading to:
 - Very big property files
 - Unbalanced load
- **Solution:** Split the file based on a threshold and ensure that the **splits** end up in **different** nodes





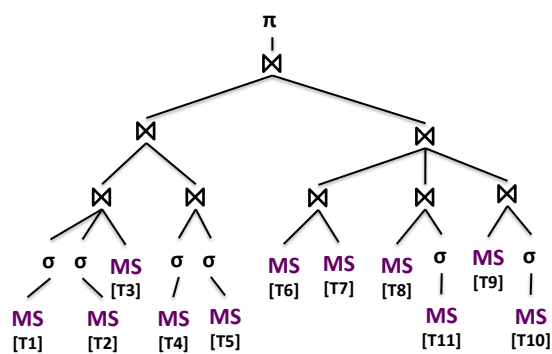
From logical plan to physical plans



89



Logical plan → physical plan

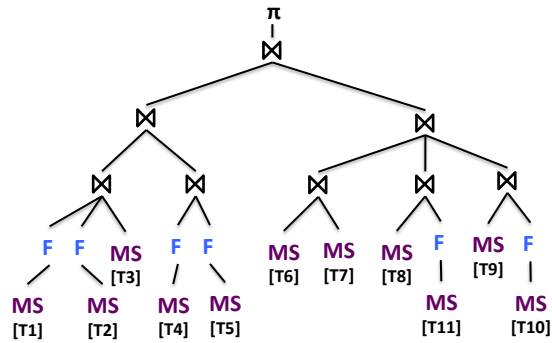


- Reading the triples from HDFS requires a Map Scan (**MS**) operator

90



Logical plan \rightarrow Physical plan

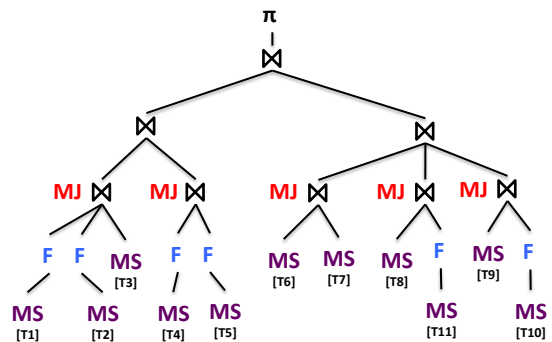


- Logical selections (σ) are translated to physical selections (F)

91

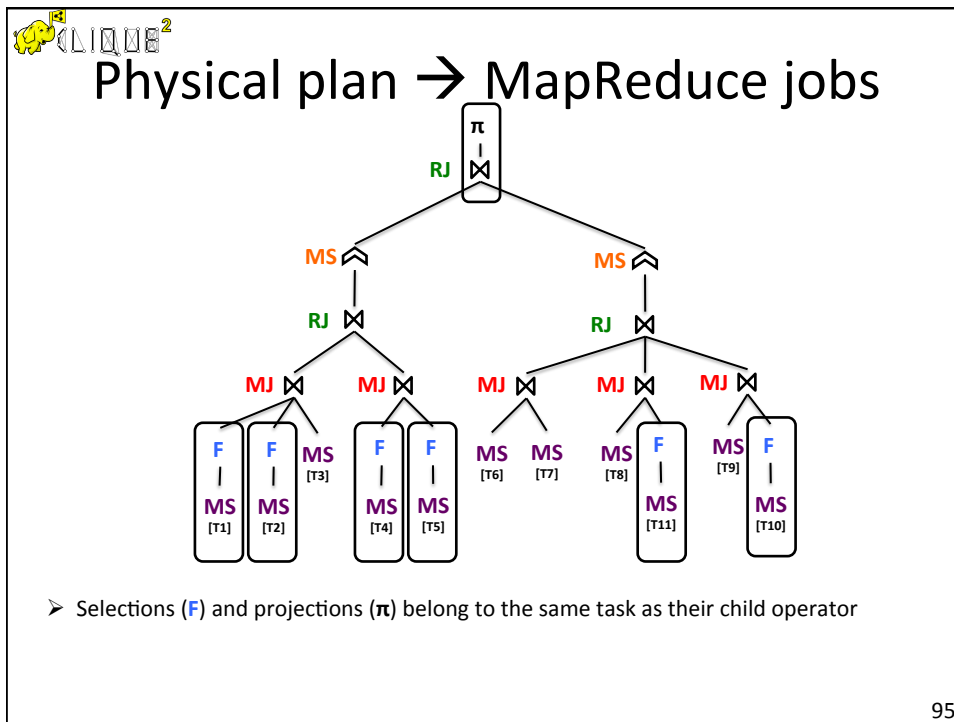


Logical plan \rightarrow Physical plan

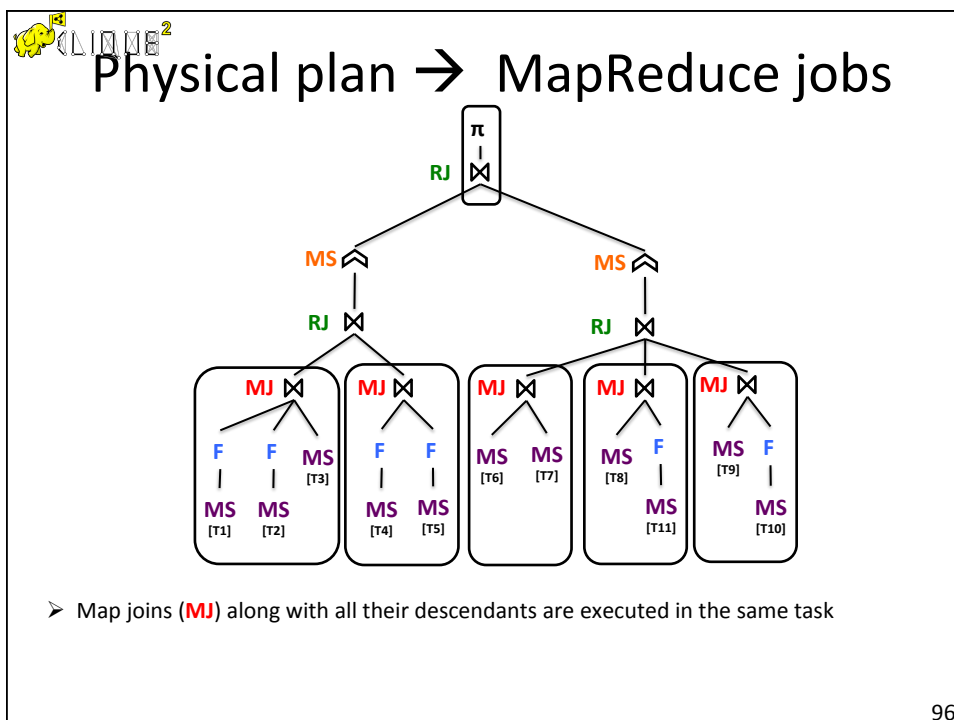


- First level joins are translated to Map side joins (MJ) taking advantage of the data partitioning

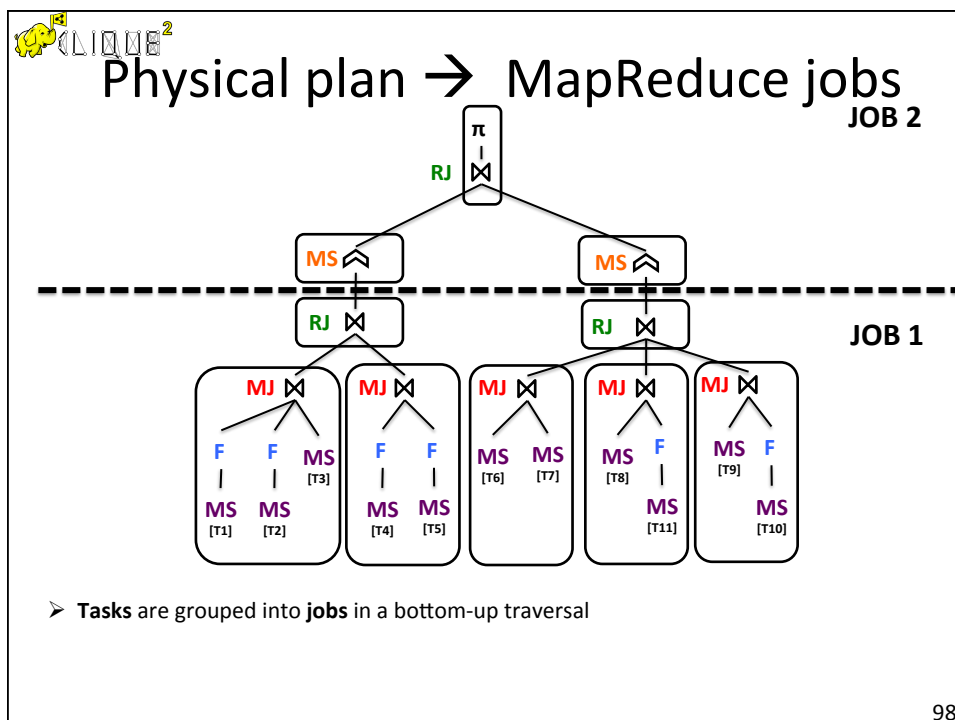
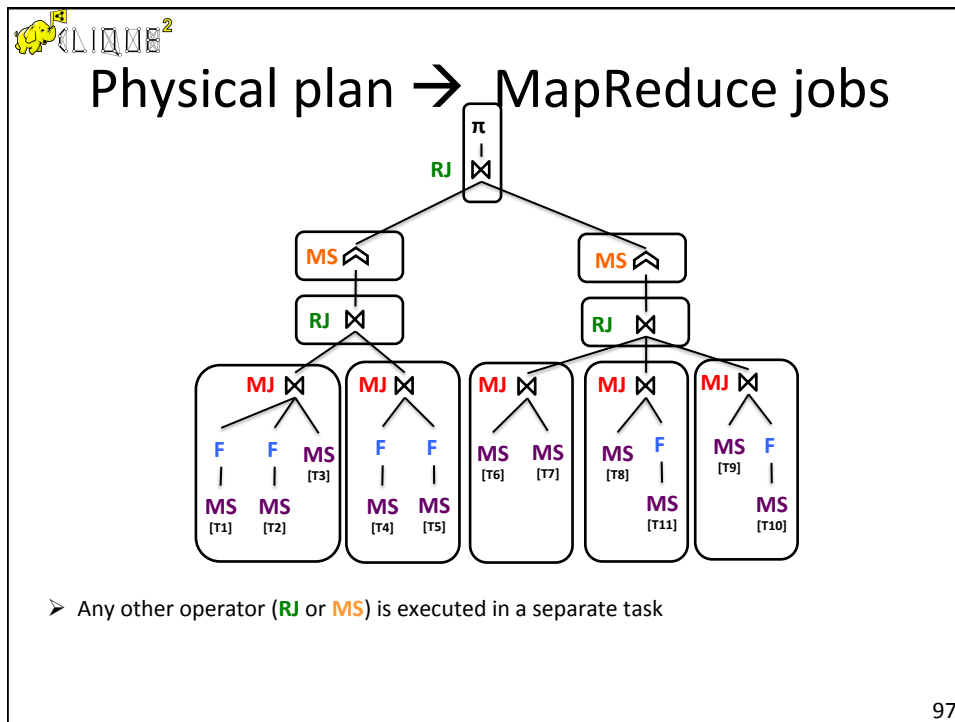
92

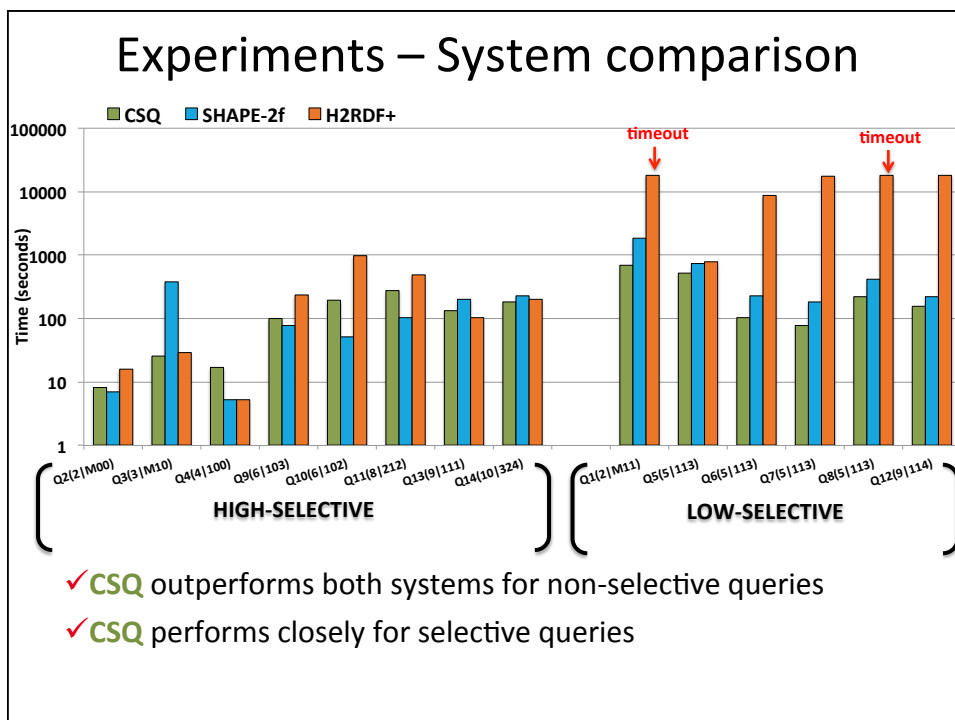
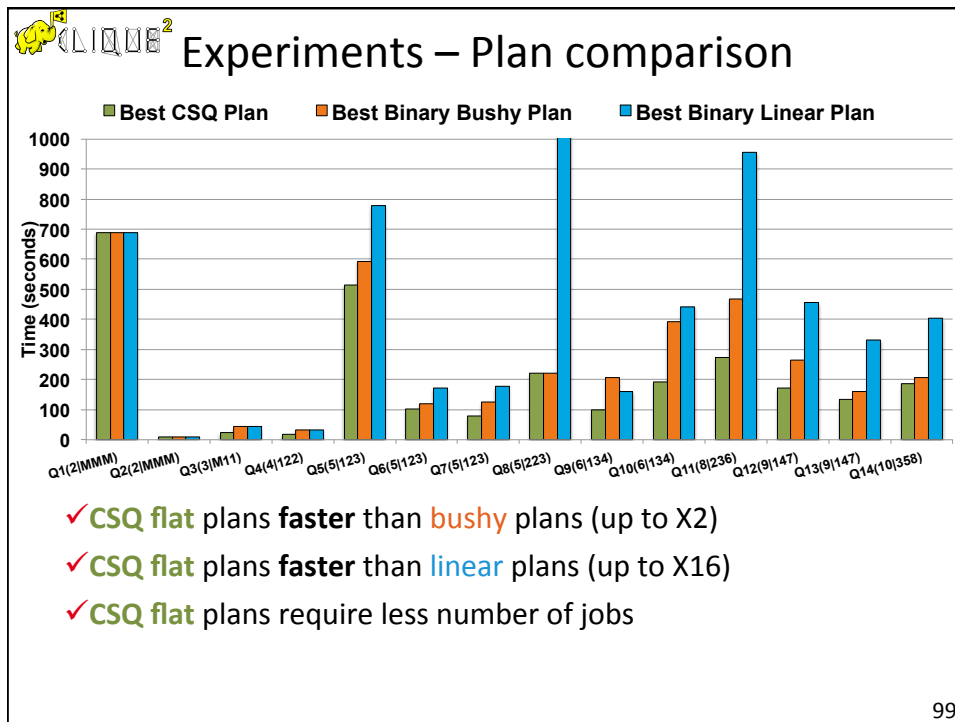


95

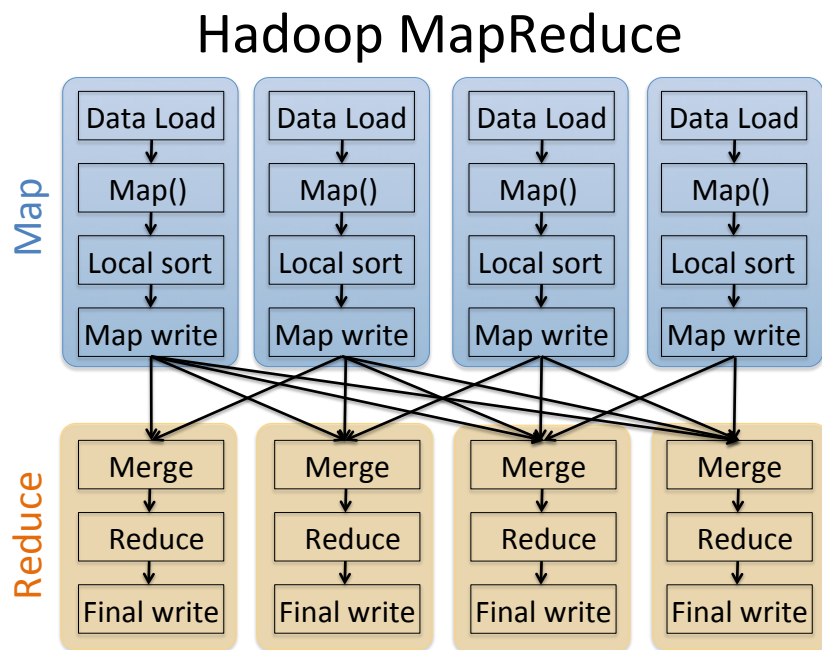


96





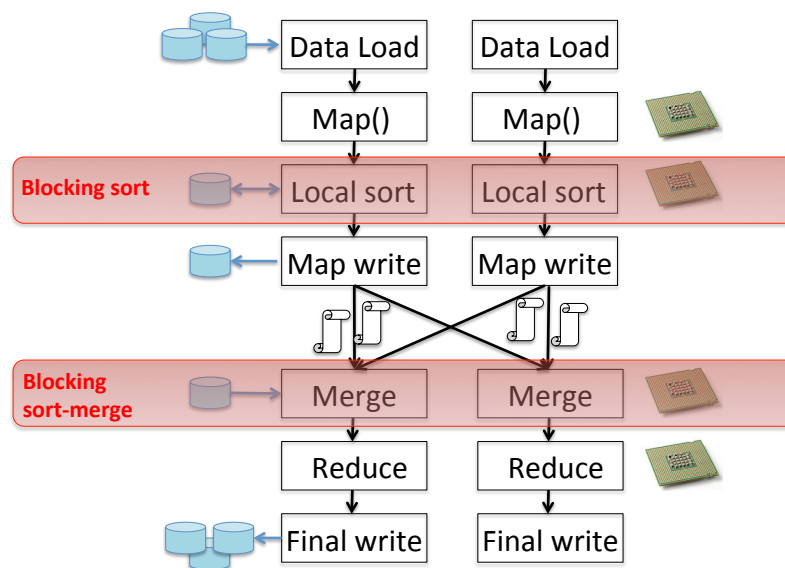
Hadoop: performance issues and solutions



102

Performance problem 1: Idle CPU due to blocking steps

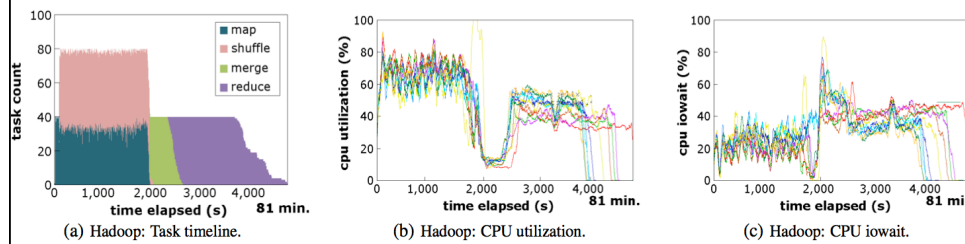
Hadoop resource usage



104

Hadoop performance study

Benchmark from [LMDMcGS2011]



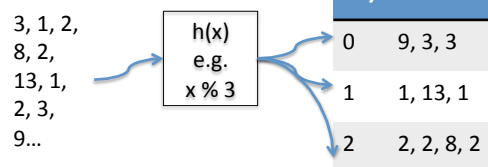
CPU stalls during I/O intensive Merge
Reduce strictly follows Merge

105

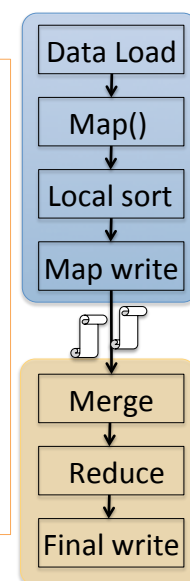
Hash-based algorithms to improve Hadoop performance

Main idea: use non-blocking hash-based algorithms to group items by keys during **Map.LocalSort** and **Reduce.Merge**

Principle of hashing:



- Partitions can be in memory or flushed to disk
- If the Reduce works incrementally, early send

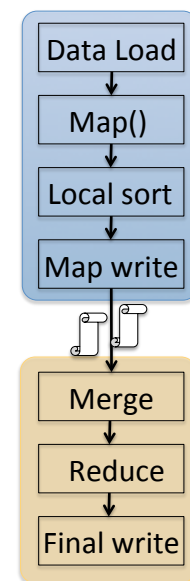


106

Performance problem 2: non-selective data access

Data access in Hadoop

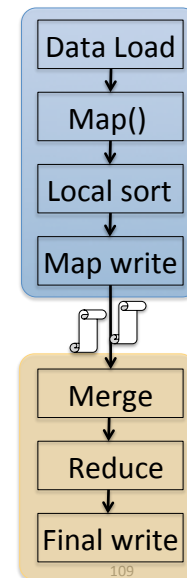
- Basic model: *read all the data*
 - If the tasks are selective, we don't really need to!
- Database indexes? But:
 - Map/Reduce works on top of a **file system** (e.g. Hadoop file system, HDFS)
 - Data is stored only once
 - Hard to foresee all future processing
 - "Exploratory nature" of Hadoop



108

Accelerating data access in Hadoop

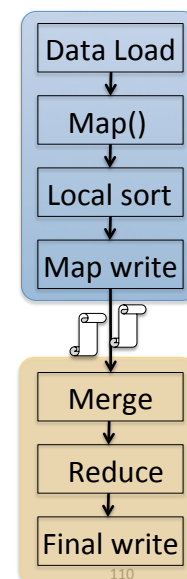
- Idea 1: Hadop++ [JQD2011]
 - Add **header information** to each data split, **summarizing** split attribute values
 - Modify the RecordReader of HDFS, used by the Map().
Make it prune irrelevant splits



109

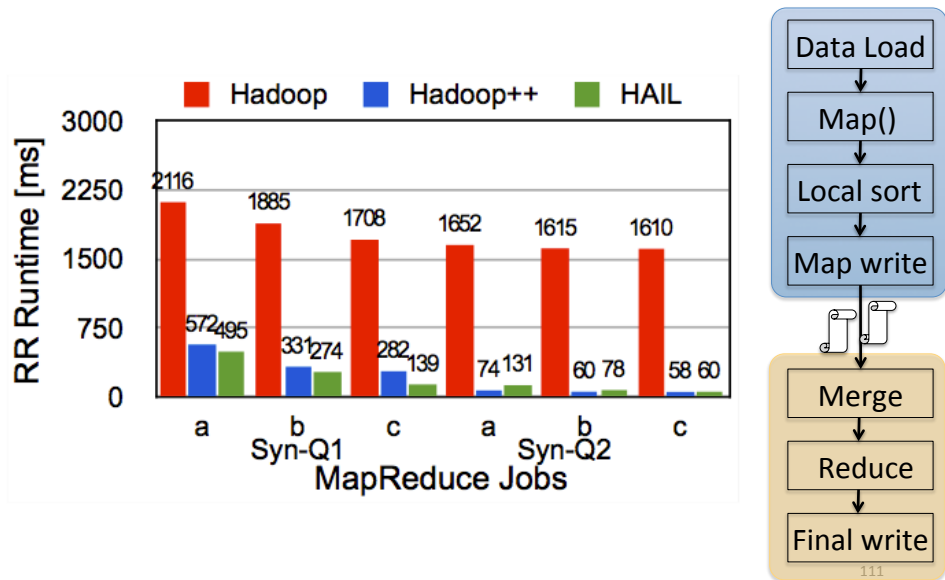
Accelerating data access in Hadoop

- Idea 2: HAIL [DQRSJS12]
 - Each storage node builds an **in-memory, clustered index** of the data in its split
 - There are three copies of each split for reliability →
Build **three different indexes**!
 - Customize RecordReader



110

Hadoop, Hadoop++ and HAIL



Other MapReduce performance issues

- Speed up the *straggler*
 - Try to optimize distribution of data to Mappers and Reducers
 - Data statistics
 - Dynamic re-distribution of straggler's load
- Eliminate the HDFS file writing
 - In-memory variants; loss of reliability; OK for short programs
- Identifying and reusing computations
 - Similar to view-based rewriting
 - At MapReduce level
 - At PigLatin level

References

- [BPERST10] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A Comparison of Join Algorithms for Log Processing in MapReduce," in SIGMOD 2010.
- [LMDMcGS11] Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, Prashant Shenoy. "A Platform for Scalable One-Pass Analytics using MapReduce", ACM SIGMOD 2011
- [DQRSJS] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Stefan Richter, Stefan Schuh, Alekh Jindal, Jorg Schad. "Only Aggressive Elephants are Fast Elephants", VLDB 2012
- [JQD11] A.Jindal, J.-A.Quiané-Ruiz, and J.Dittrich. "Trojan Data Layouts: Right Shoes for a Running Elephant" SOCC, 2011