Query processing on raw files

Vítor Uwe Reus

Outline

1. Introduction
2. Adaptive Indexing
3. Hybrid MapReduce
4. NoDB
5. Summary

Outline

1. Introduction
2. Adaptive Indexing
3. Hybrid MapReduce
4. NoDB
5. Summary

Raw Files

Information Storing

Sometimes human-readable, open format

Not physically optimized for querying

Might be useful in some cases

Big Data

Traditional DBMS may not be a good option

Internet-scale business

Scientific data

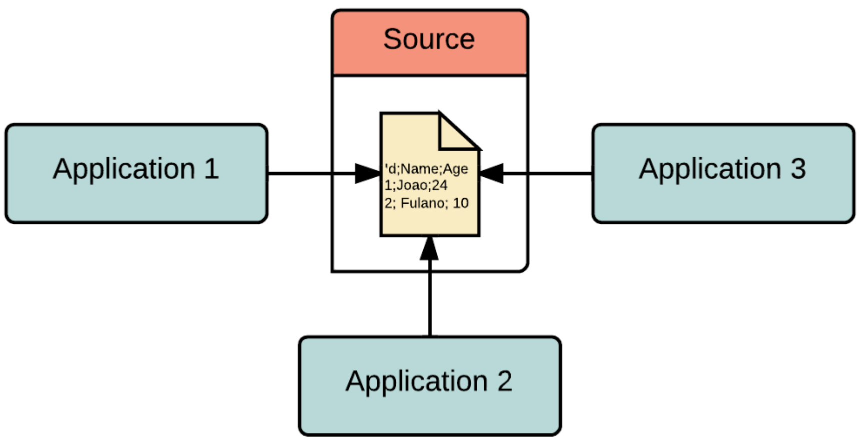
The fourth paradigm

For scientific discovery

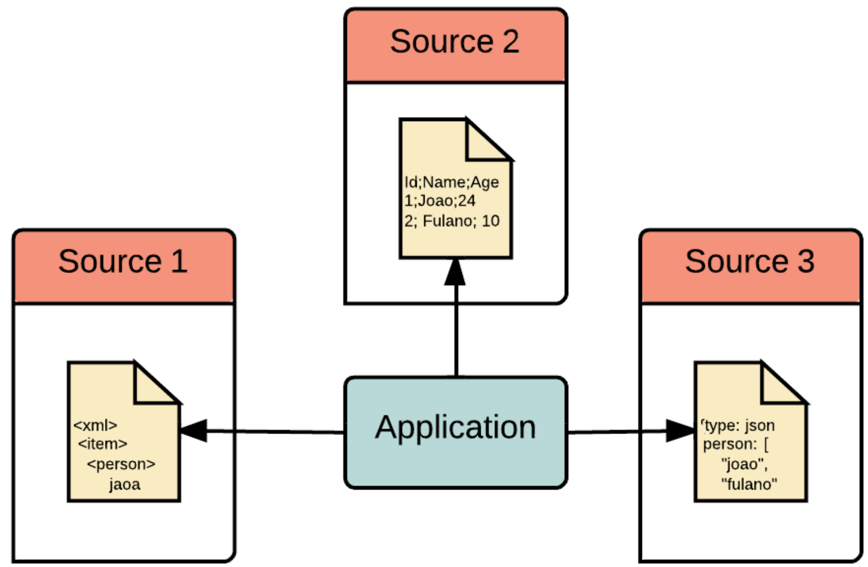
- Experimental
- Theoretical
- Computational (simulations)
- **Data driven**

Interoperability

Interoperability



Information interoperability



Application interoperability

Human-sourced Information

How to query raw files?

State of the art

Raw file as storage

A-priori loading

Raw file parsing

AWK

Oracle external table

MySQL CSV engine

MapReduce

Read entire data all times

No indexing features

A-priori loading

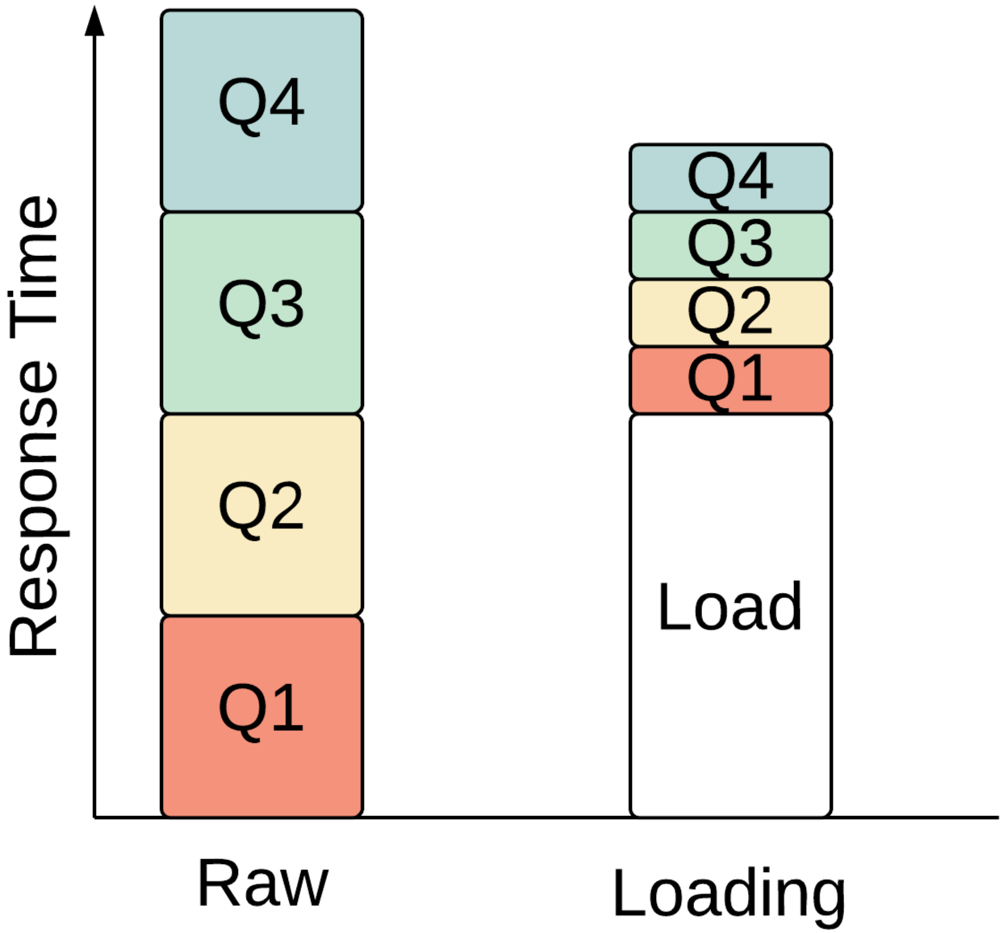
Load into a DBMS and then query
- Benefit from indexes

Time

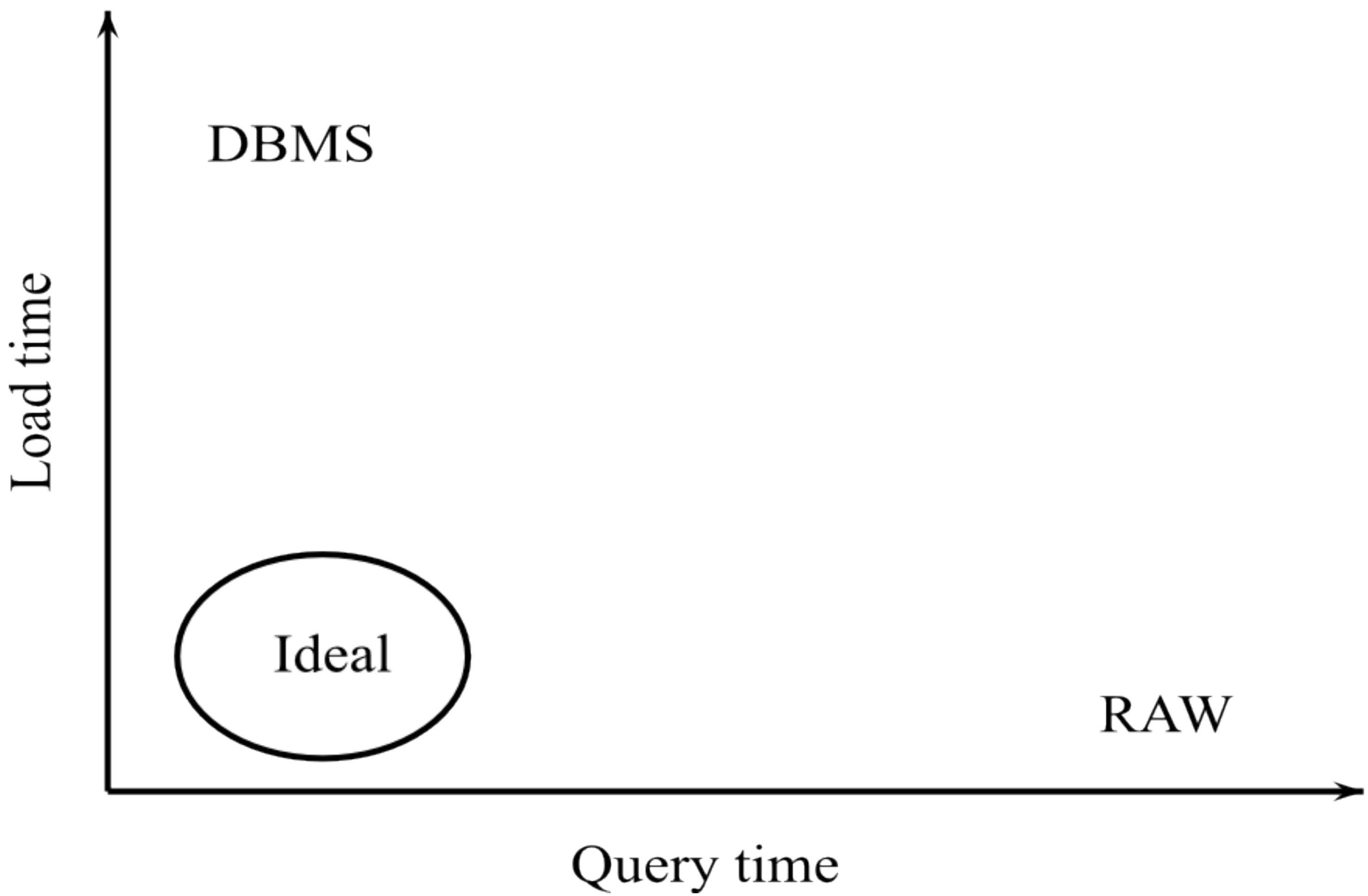
Labor intensive
- Loading scripts, schemas

Data duplication
- Big data
- Versioning

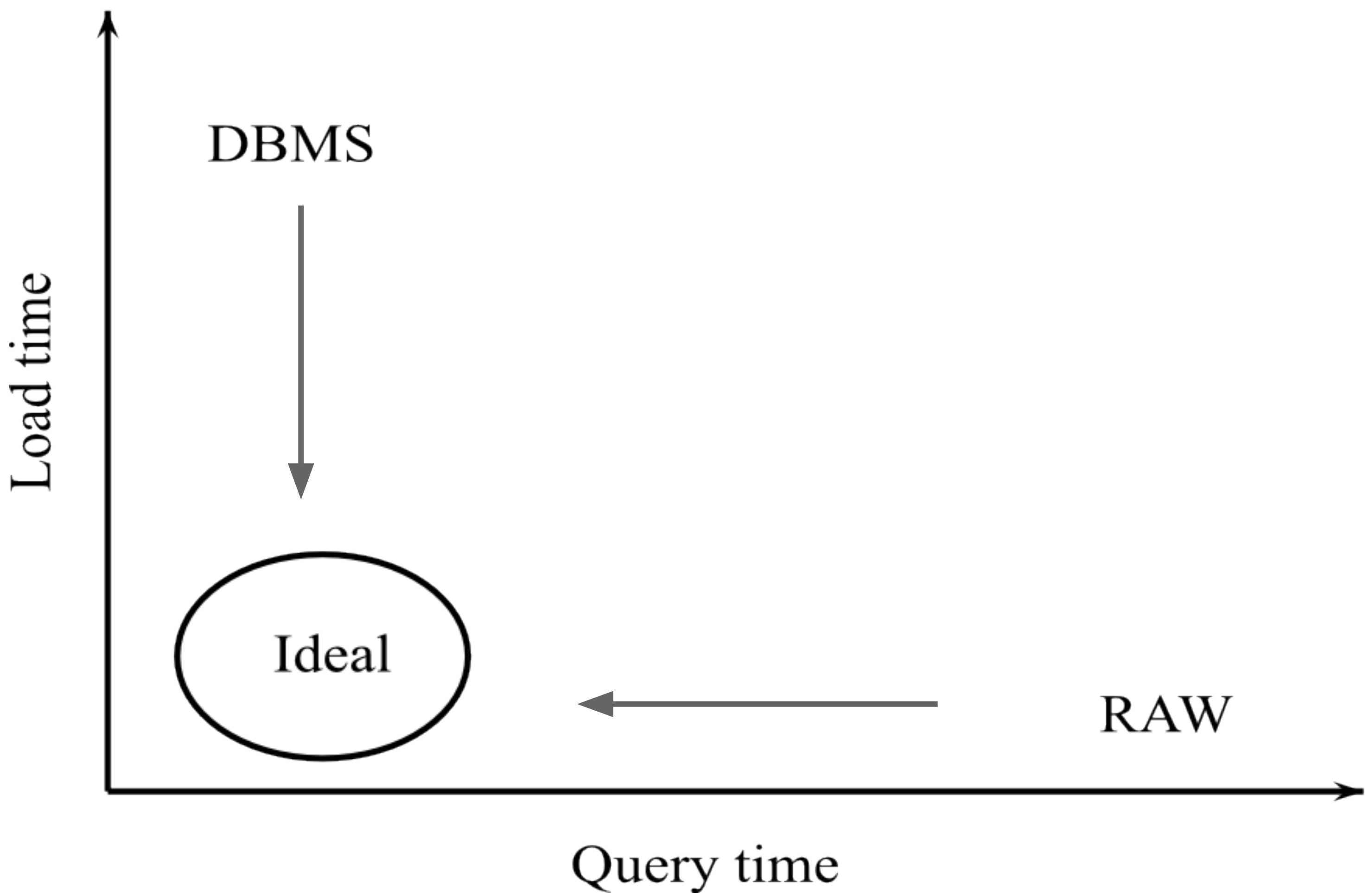
Workload behavior



Load time vs Query time



Hybrid querying techniques



Outline

1. Introduction
2. Adaptive Indexing
3. Hybrid MapReduce
4. NoDB
5. Summary

Adaptive indexing

Automatic tuning based on workload

Keep an auxiliary structure

Can benefit raw file parsing

Database Cracking

Adaptive Merging

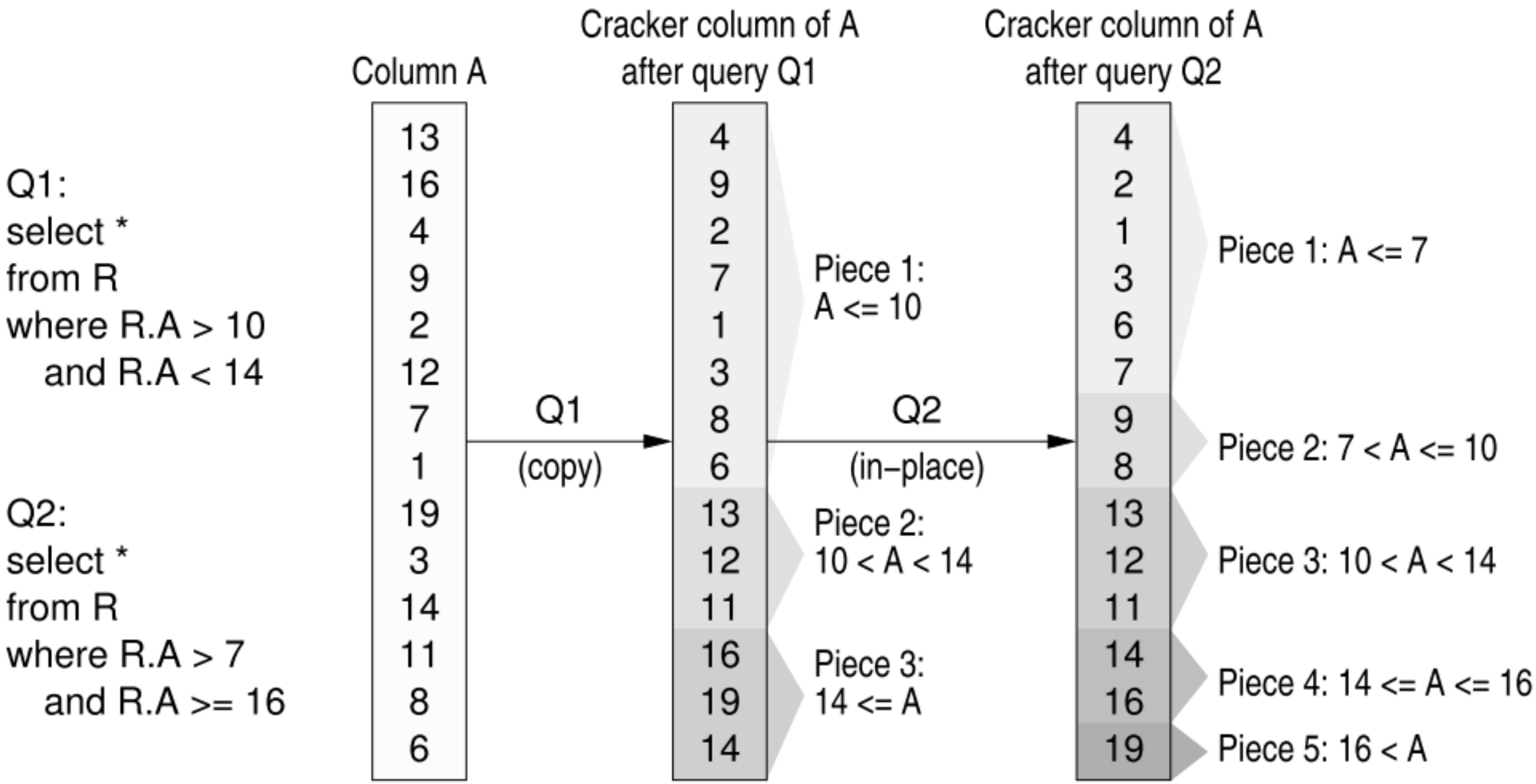
Database cracking

Physical reorganization of columns

Implemented on MonetDB

- A column store, but can be generalized (raw)

Database Cracking



Cracking a column

Database Cracking

Column A $\rightarrow$ Copy to cracker column A_{CRK}

AVL tree indexing

Refinement

Tuple reconstruction

Fast if columns are in same order

Cracking compromises original positions

Cracker columns: Value selection

Original columns: Tuple reconstruction

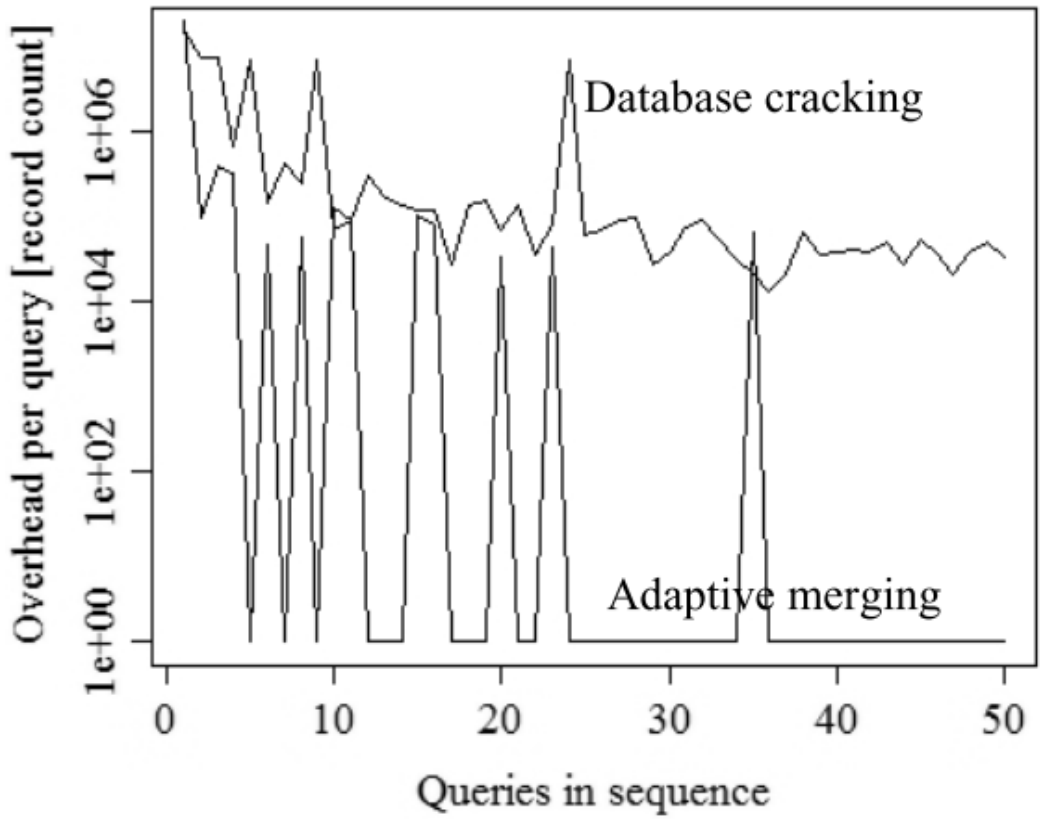
Adaptive merging

Incremental index creation as in cracking

Partitioned B-trees

Focus on merging instead of partitioning

Merging vs cracking



Typical result of merging compared to cracking

*In this case, all queries focus on the same 10^6 keys in the center of the domain

Merging vs cracking

	Cracking	Merging
Converge	Stable	Faster
Storage	AVL	B-Tree
Data is	Partitioned	...and Sorted
as in..	Quick Sort	Merge Sort

Outline

1. Introduction
2. Adaptive Indexing
3. Hybrid MapReduce
4. NoDB
5. Summary

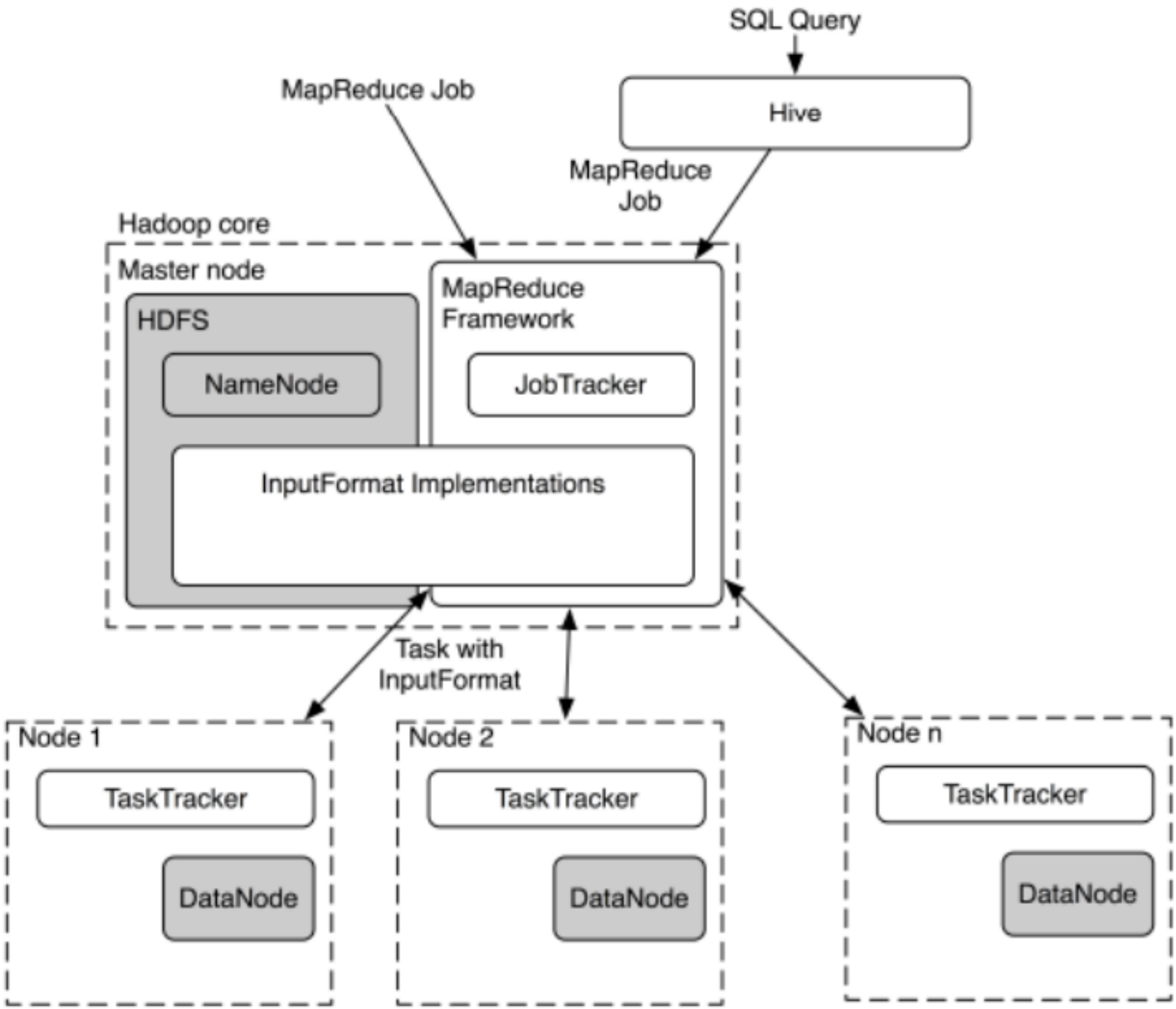
Hybrid MapReduce

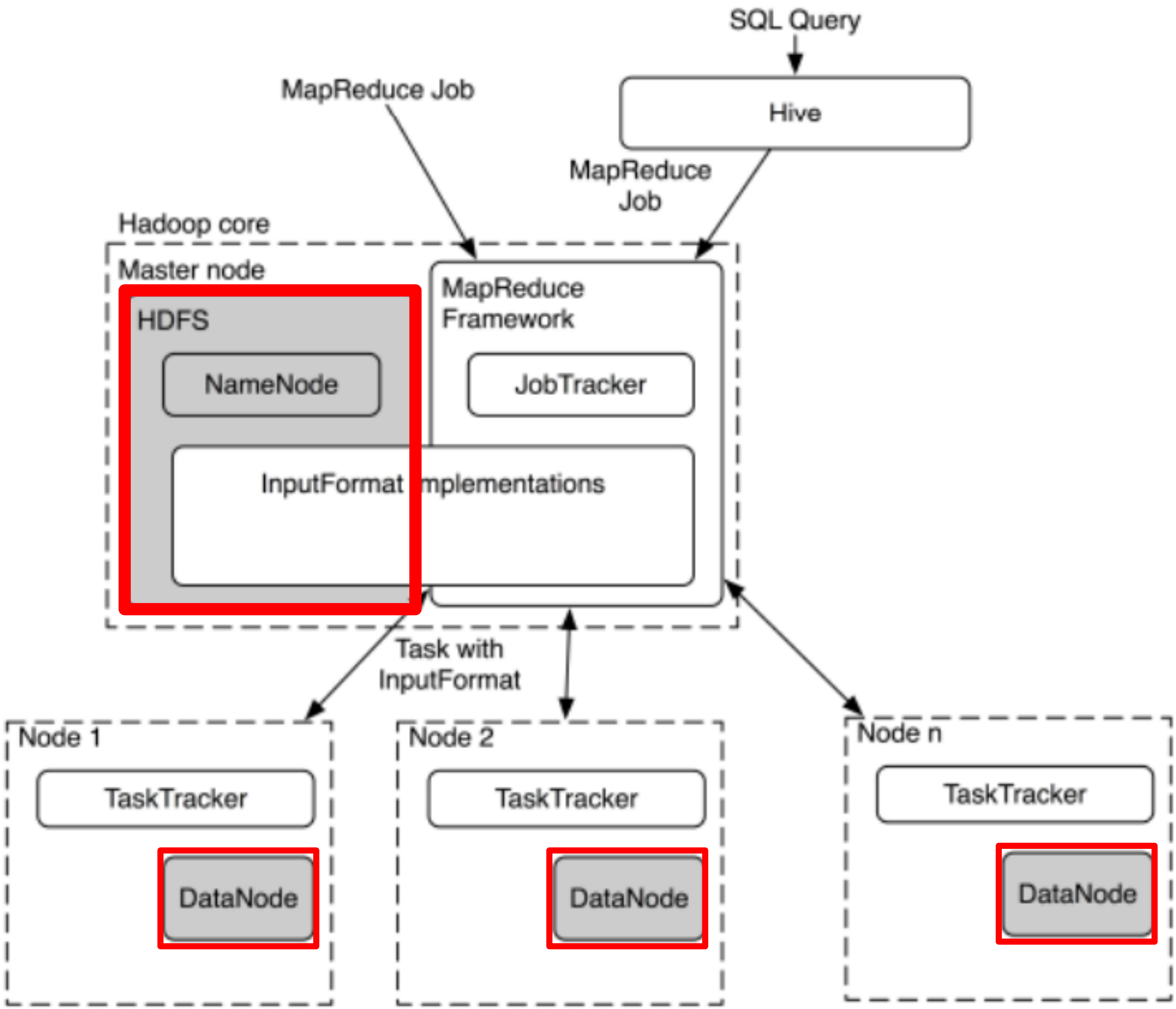
	Scalability*	High Performance**
MapReduce	✓	✗
Parallel Databases	✗	✓
What is needed	✓	✓

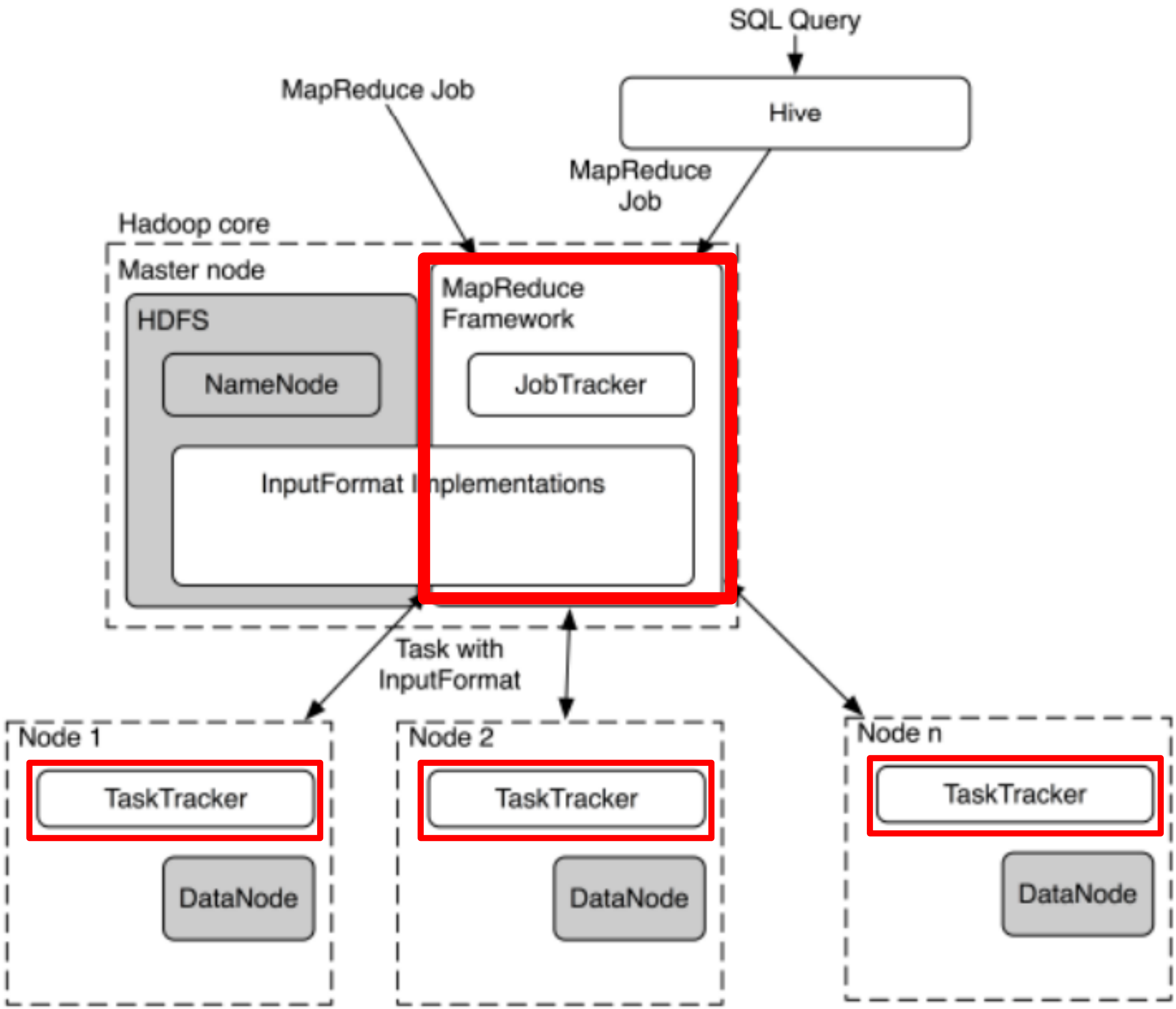
** 1000s of nodes*
*** Queries on structured data*

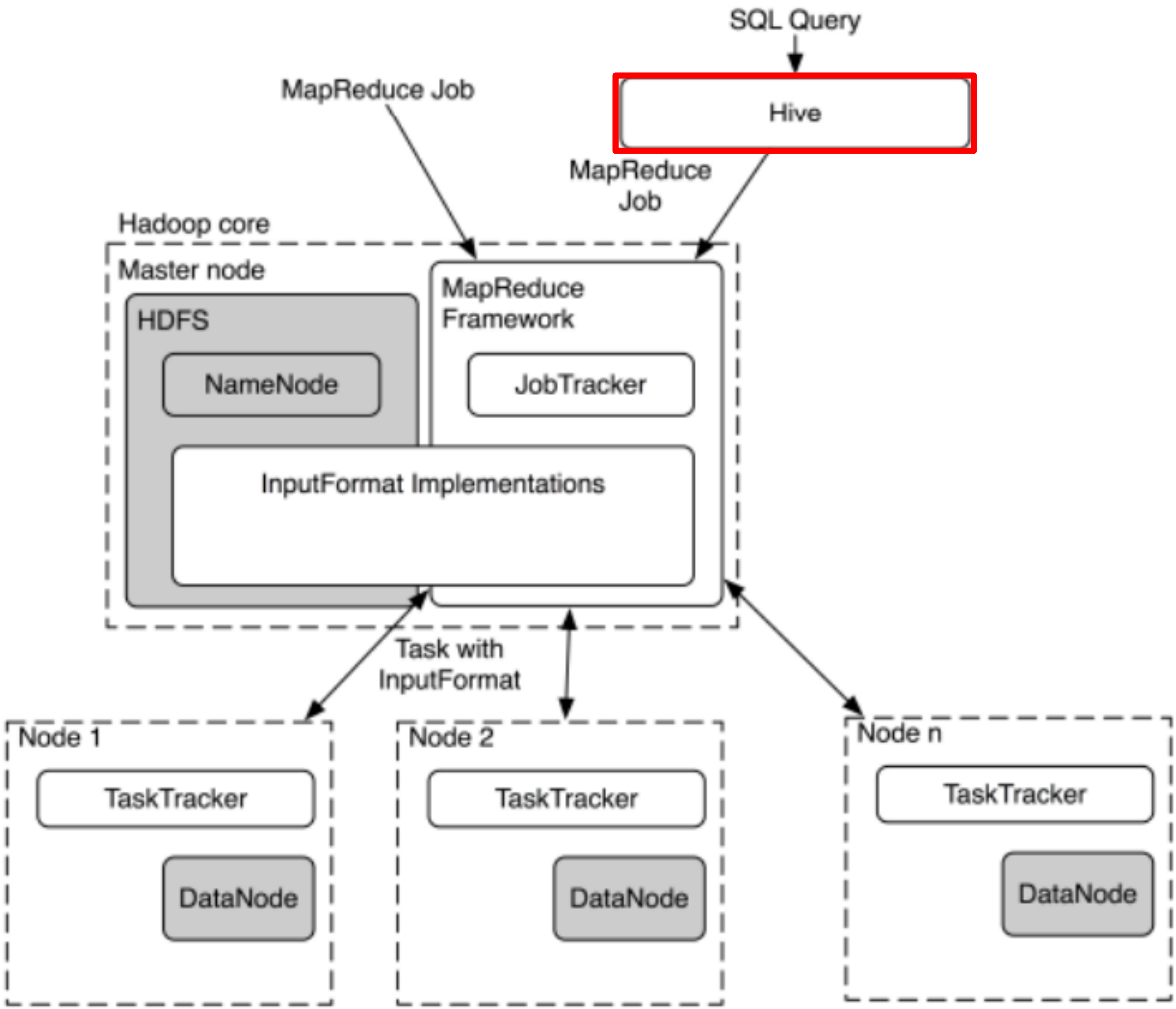
HadoopDB

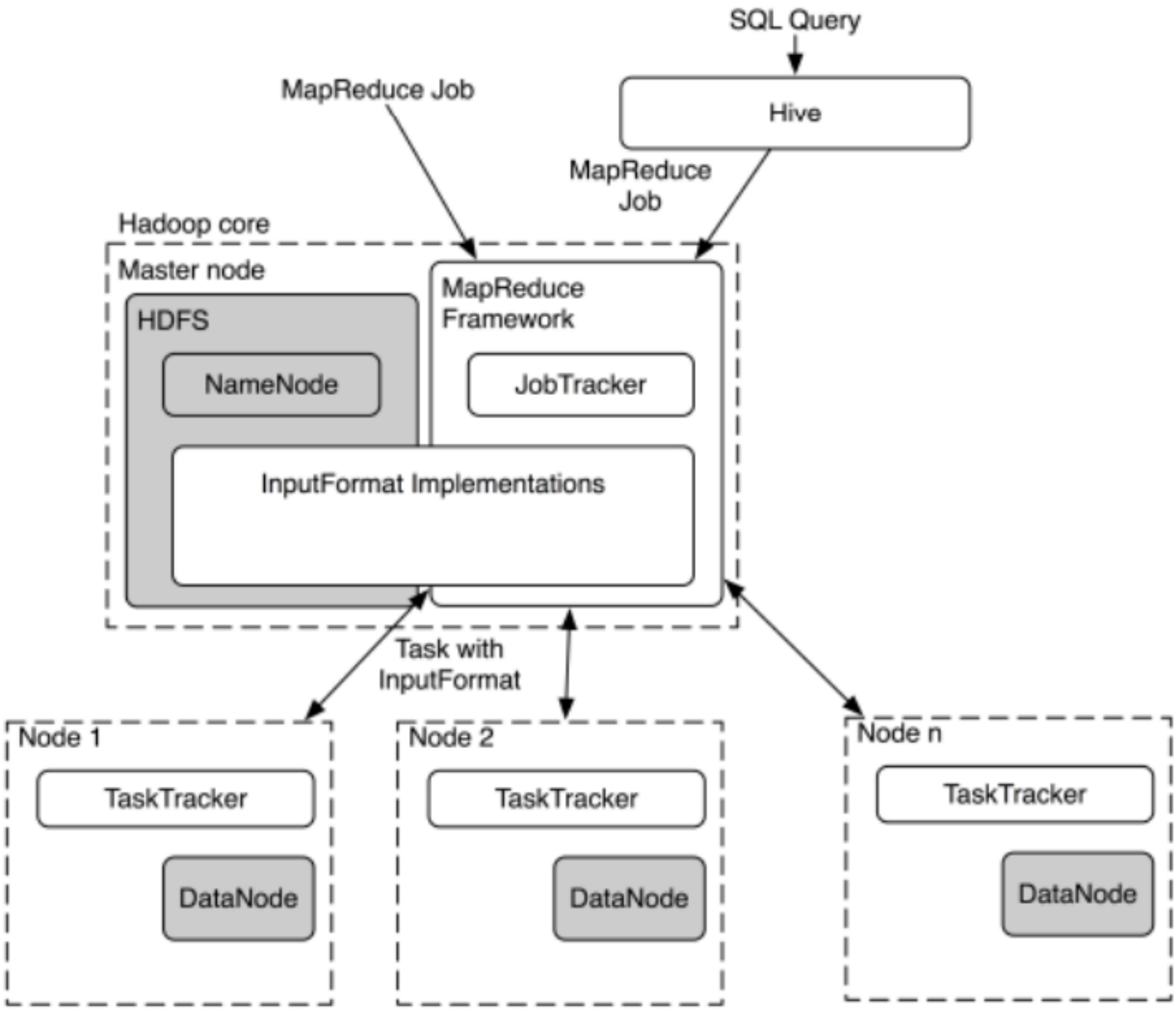
MapReduce using a DBMS instead of HDFS

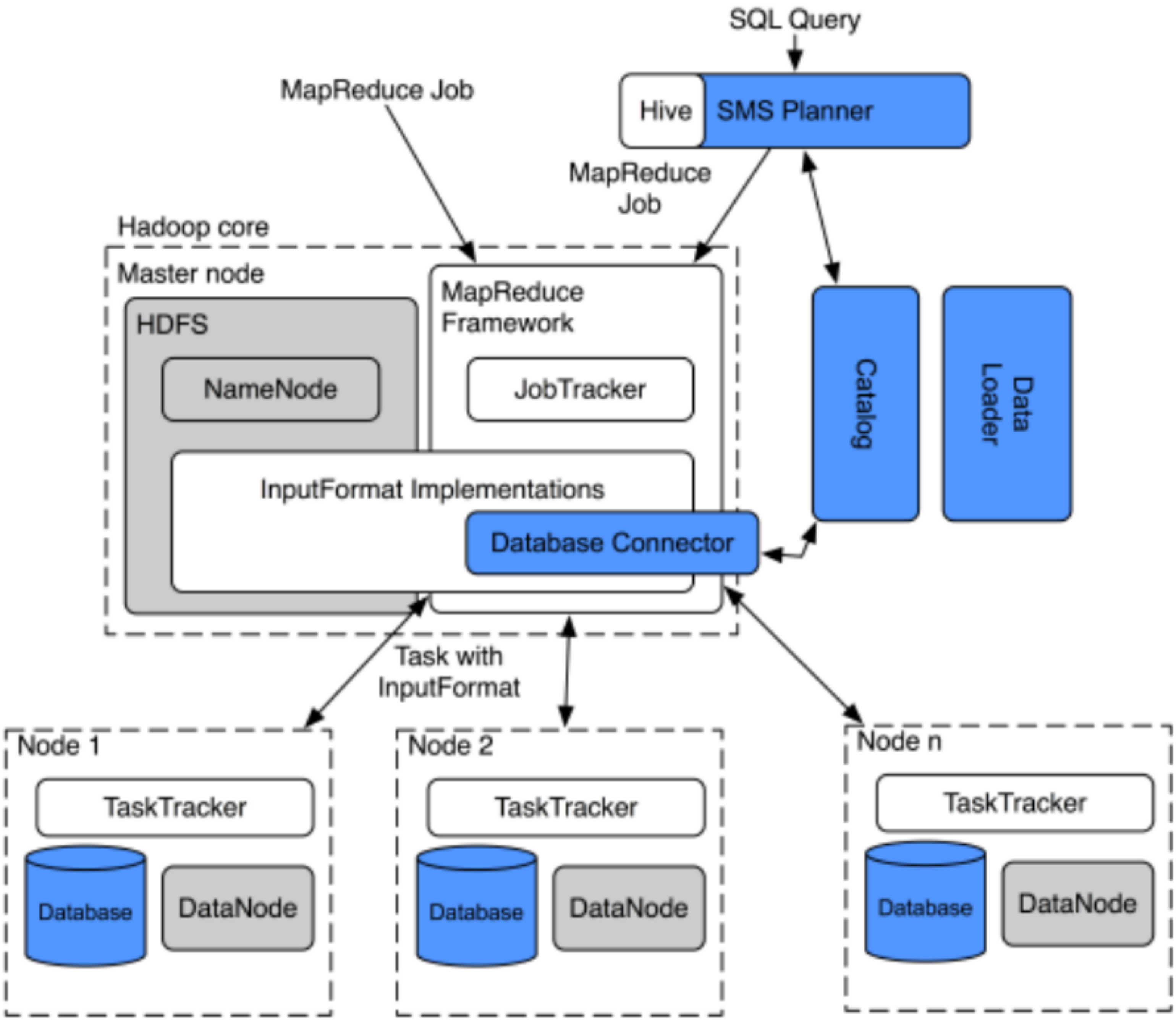

SQL Query
Hive
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Task with
InputFormat
Node 1
TaskTracker
DataNode
Node 2
TaskTracker
DataNode
Node n
TaskTracker
DataNode


SQL Query
Hive
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Task with
InputFormat
Node 1
TaskTracker
DataNode
Node 2
TaskTracker
DataNode
Node n
TaskTracker
DataNode

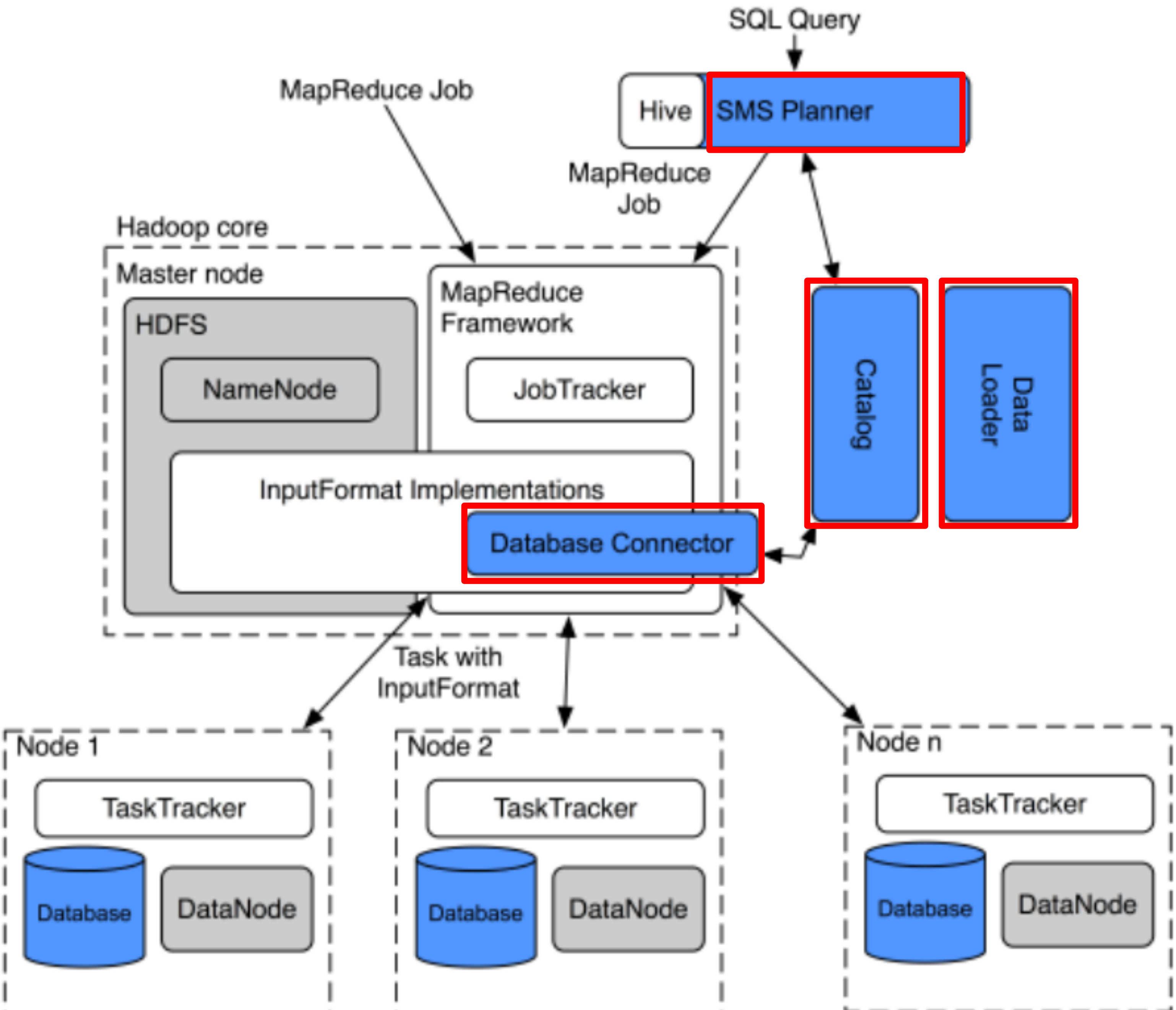

SQL Query
MapReduce Job
Hive
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Task with
InputFormat
Node 1
TaskTracker
DataNode
Node 2
TaskTracker
DataNode
Node n
TaskTracker
DataNode

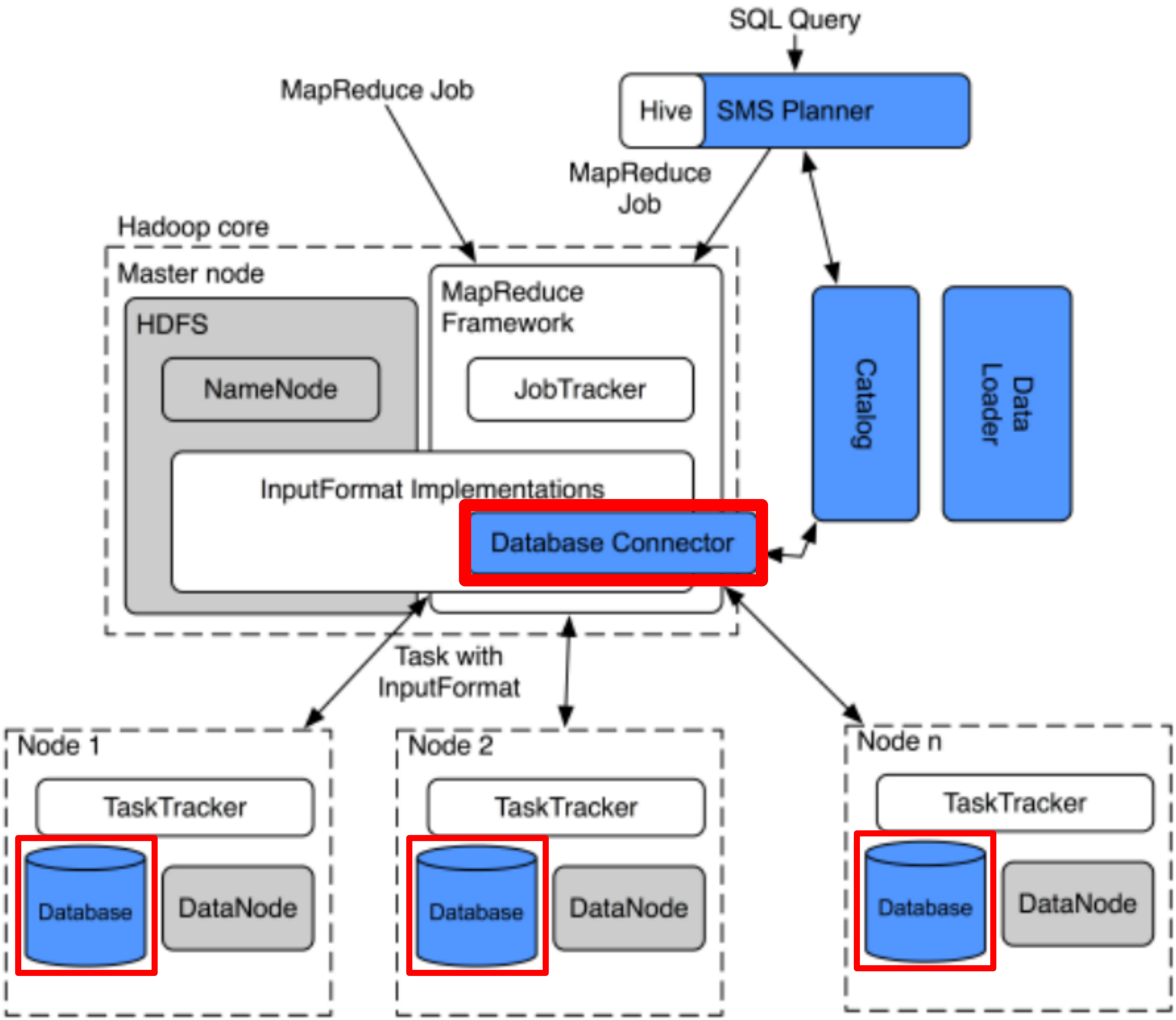

SQL Query
Hive
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Task with
InputFormat
Node 1
TaskTracker
DataNode
Node 2
TaskTracker
DataNode
Node n
TaskTracker
DataNode

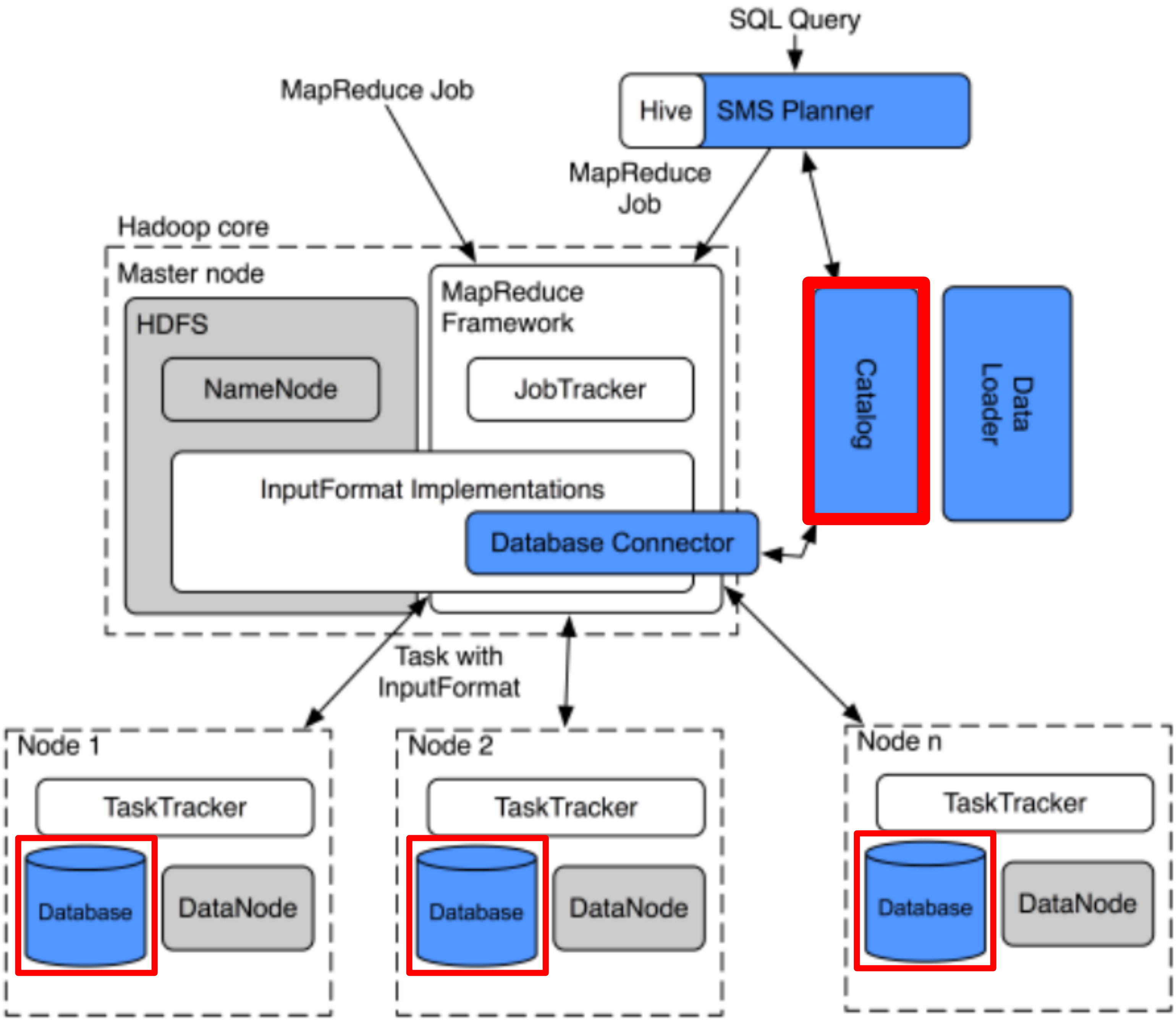

SQL Query
Hive
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Task with
InputFormat
Node 1
TaskTracker
DataNode
Node 2
TaskTracker
DataNode
Node n
TaskTracker
DataNode



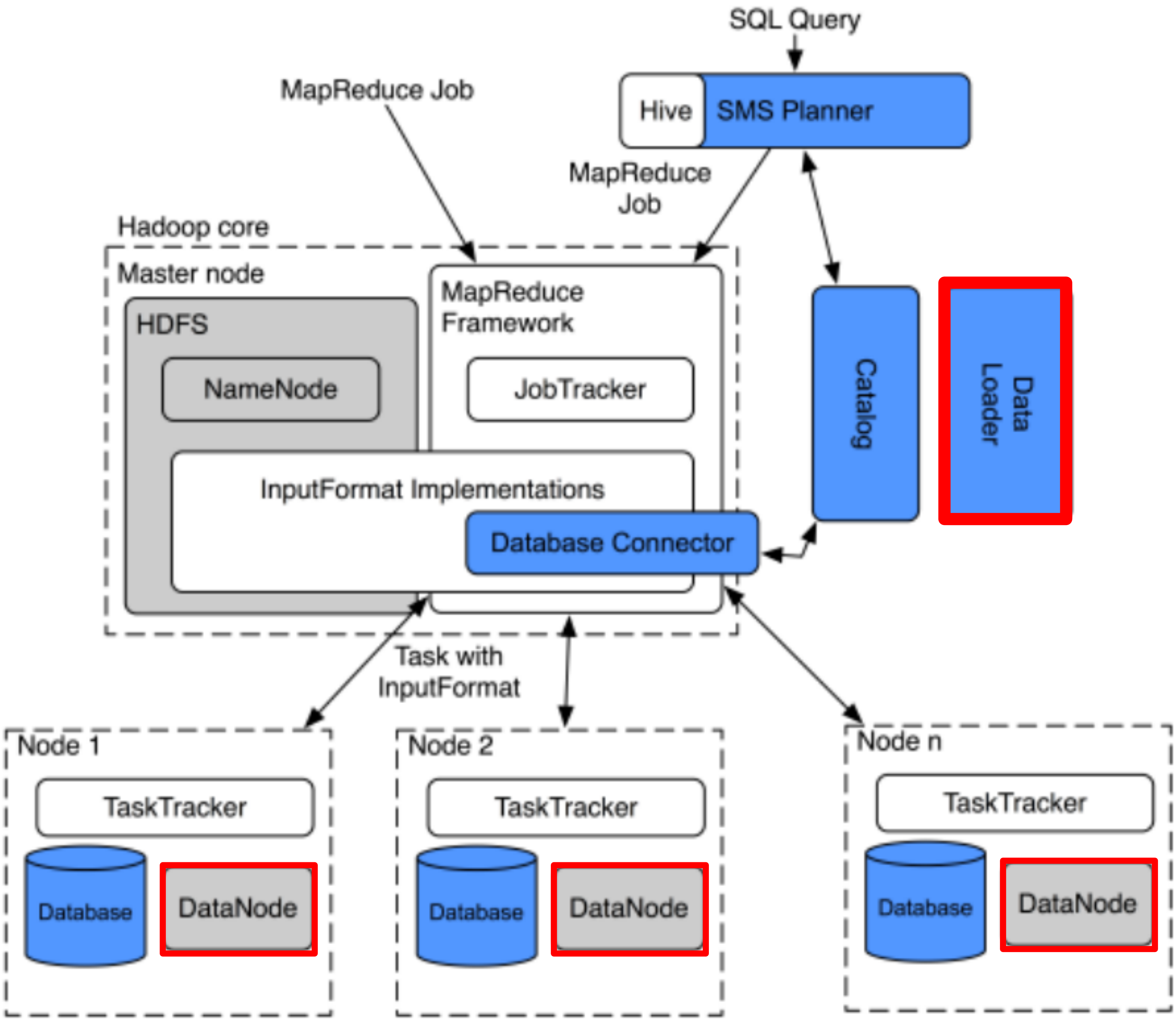
SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
Catalog
Data
Loader
InputFormat Implementations
Database Connector
Task with
InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode

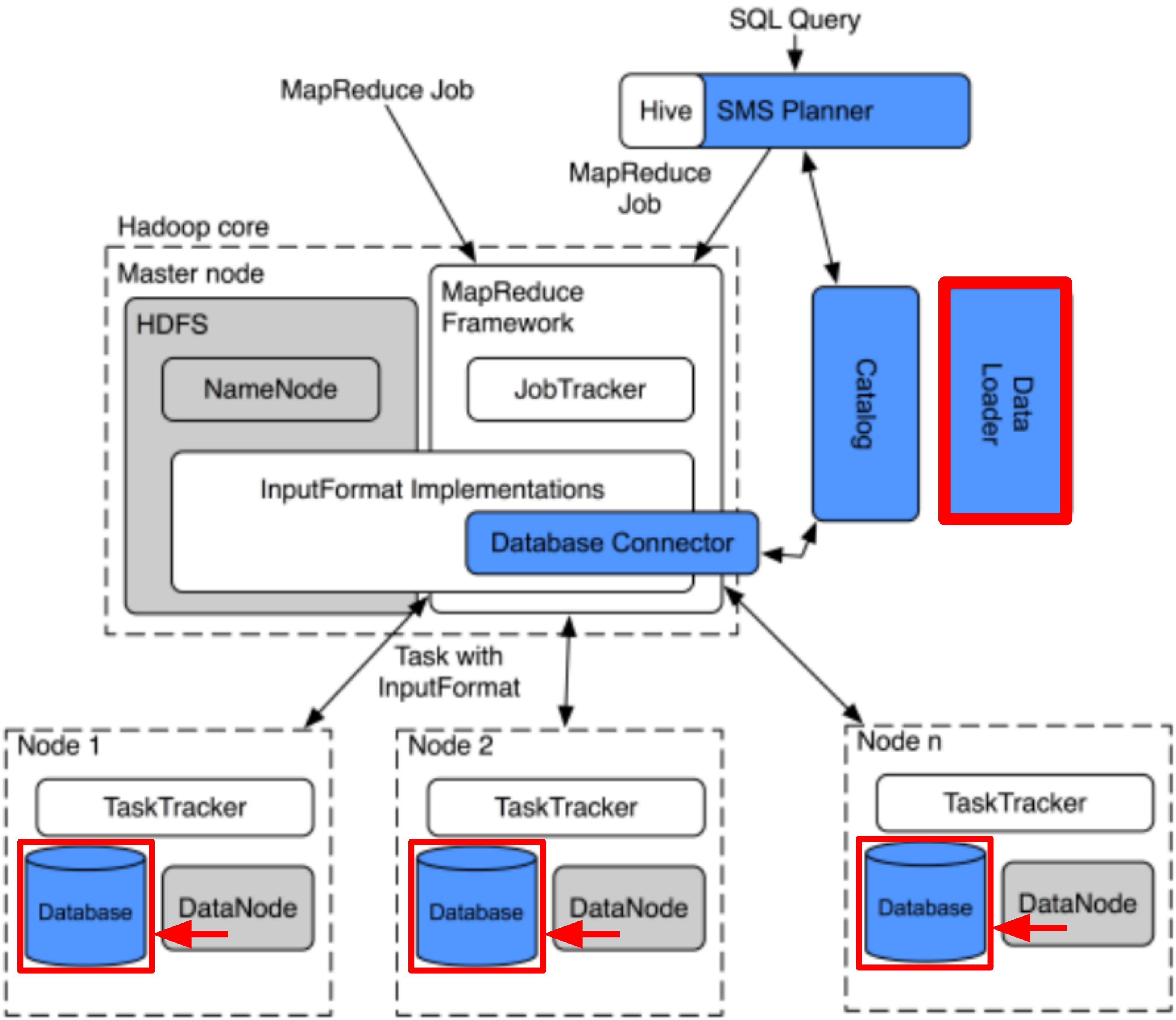

SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Database Connector
Catalog
Data
Loader
Task with
InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode

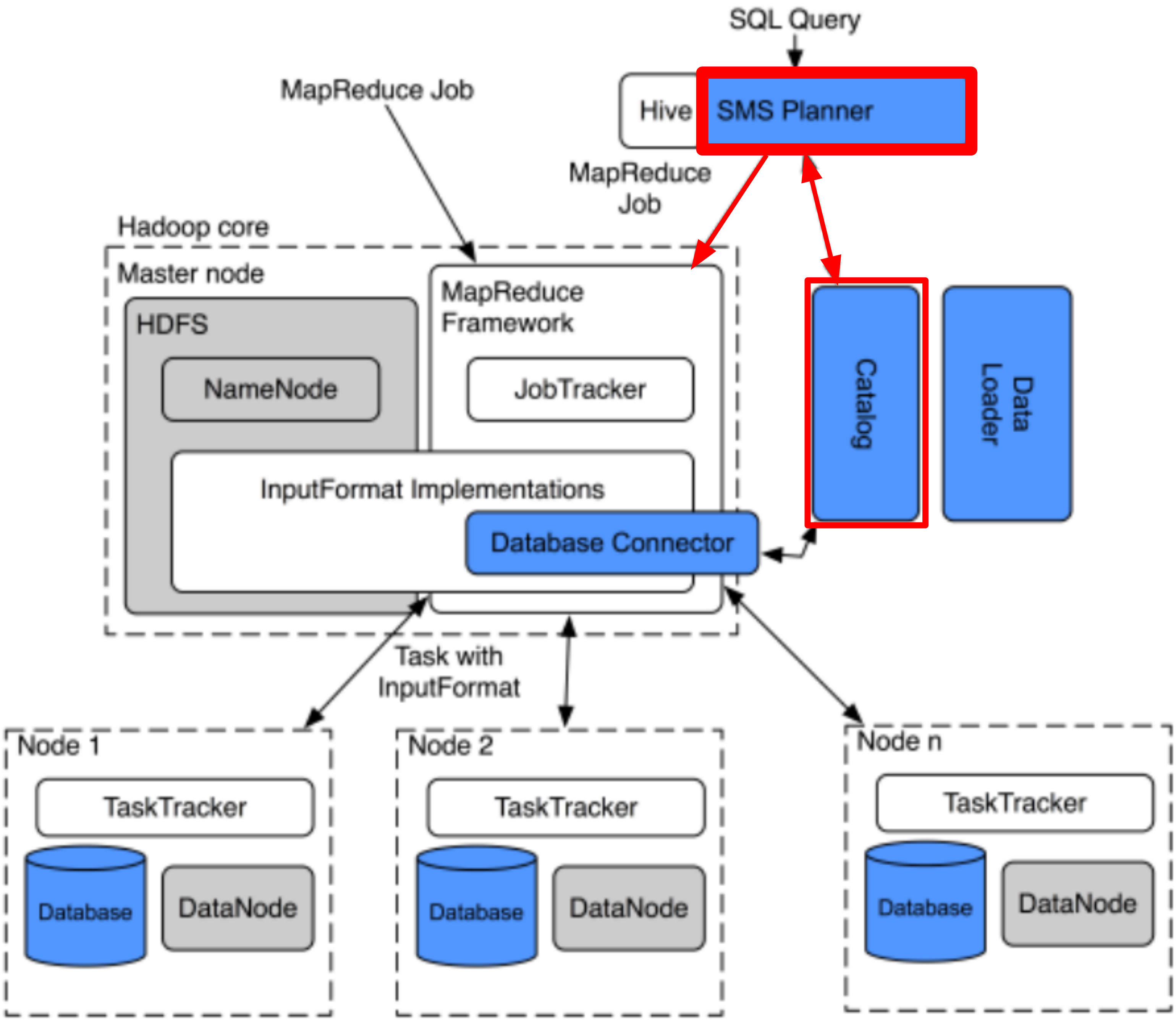

SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Database Connector
Catalog
Data
Loader
Task with
InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode



SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce Job
Hadoop core
Master node
HDFS
NameNode
MapReduce Framework
JobTracker
InputFormat Implementations
Database Connector
Catalog
Data Loader
Task with InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode


SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
Catalog
Data
Loader
InputFormat Implementations
Database Connector
Task with
InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode


SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce
Job
Hadoop core
Master node
HDFS
NameNode
MapReduce
Framework
JobTracker
InputFormat Implementations
Database Connector
Catalog
Data
Loader
Task with
InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode


SQL Query
Hive
SMS Planner
MapReduce Job
MapReduce Job
Hadoop core
Master node
HDFS
NameNode
MapReduce Framework
JobTracker
InputFormat Implementations
Database Connector
Catalog
Data Loader
Task with InputFormat
Node 1
TaskTracker
Database
DataNode
Node 2
TaskTracker
Database
DataNode
Node n
TaskTracker
Database
DataNode

SMS Planner

Hive query processor

1. Convert HiveQL query to AST
2. Get schema from catalog
3. Create a Query Plan
4. Optimize
5. Converted plan to one or more MR Jobs

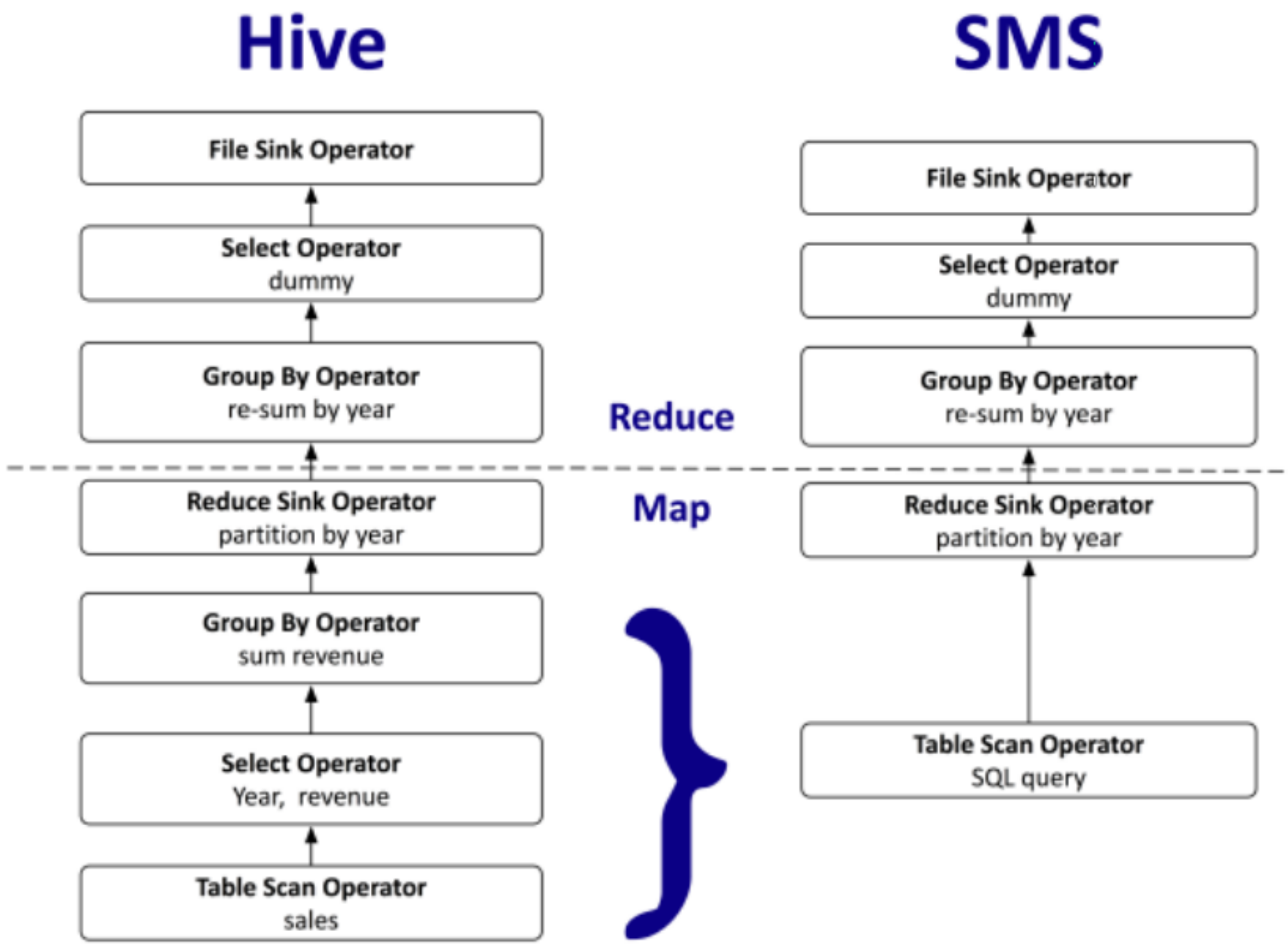
SMS Planner

1. Convert HiveQL query to AST

   Update Catalog with DB information
2. Get schema from catalog
3. Create a Query Plan
4. Optimize

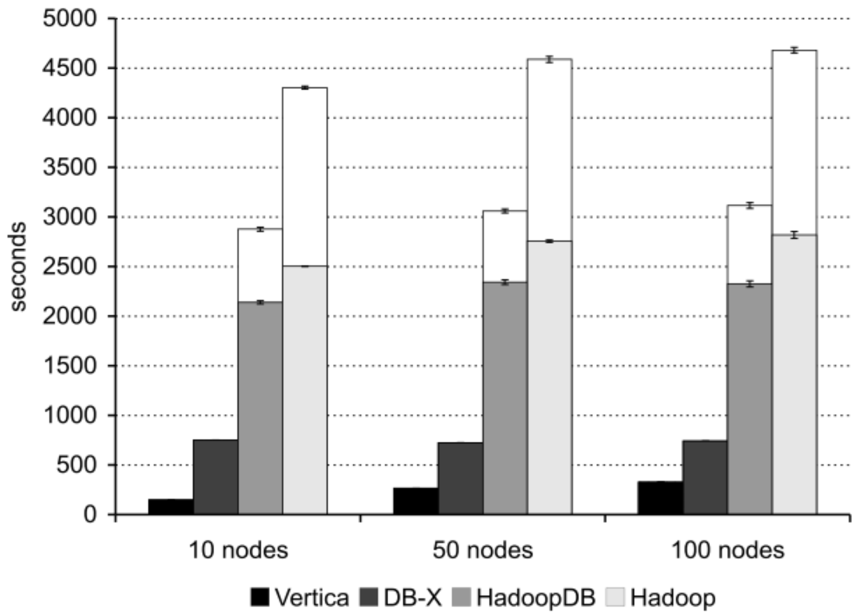
   Reconstruc some SQL to push it to the DB
5. Converted plan to one or more MR Jobs

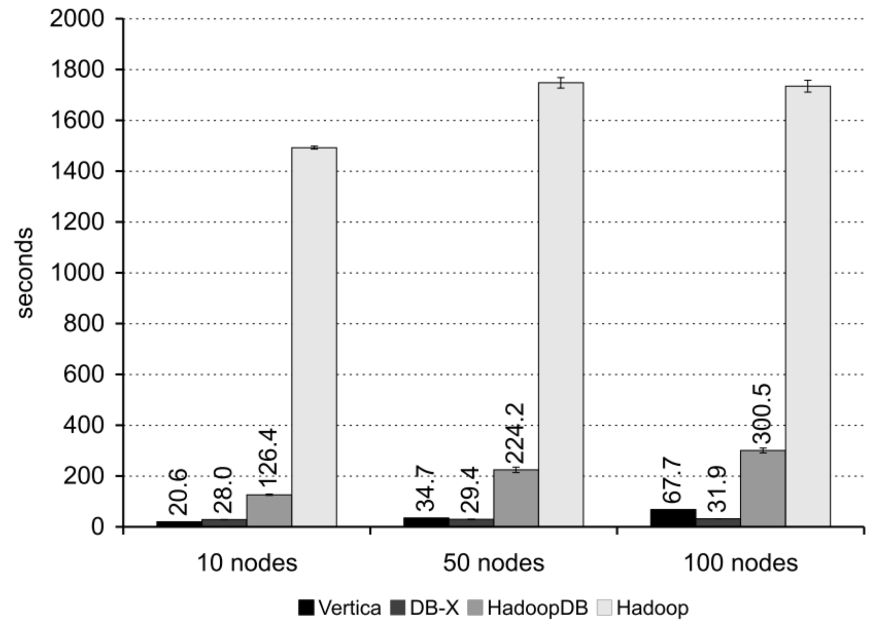
SMS Planner



SELECT YEAR(saleDate), SUM(revenue) FROM sales GROUP BY YEAR(saleDate);

HadoopDB Performance





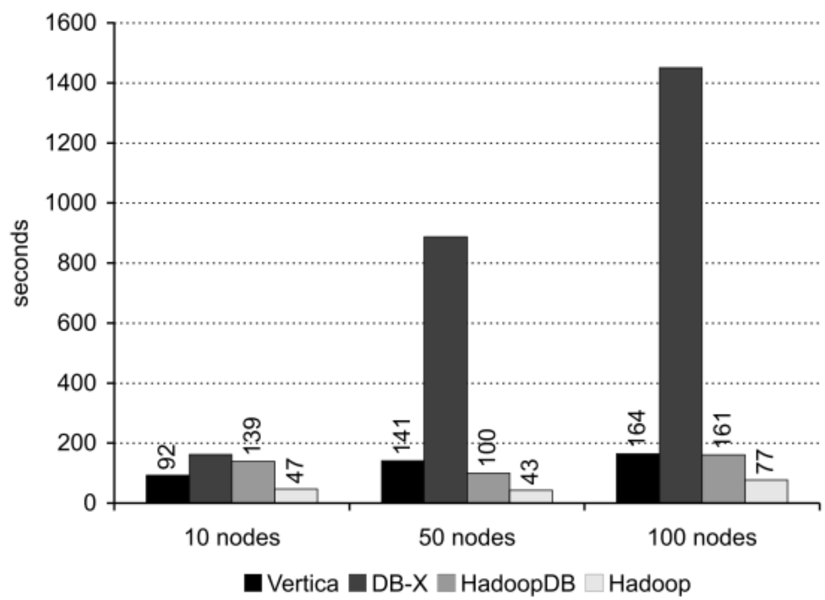
Group By

2,500,000 unique groups
over 20gb of data

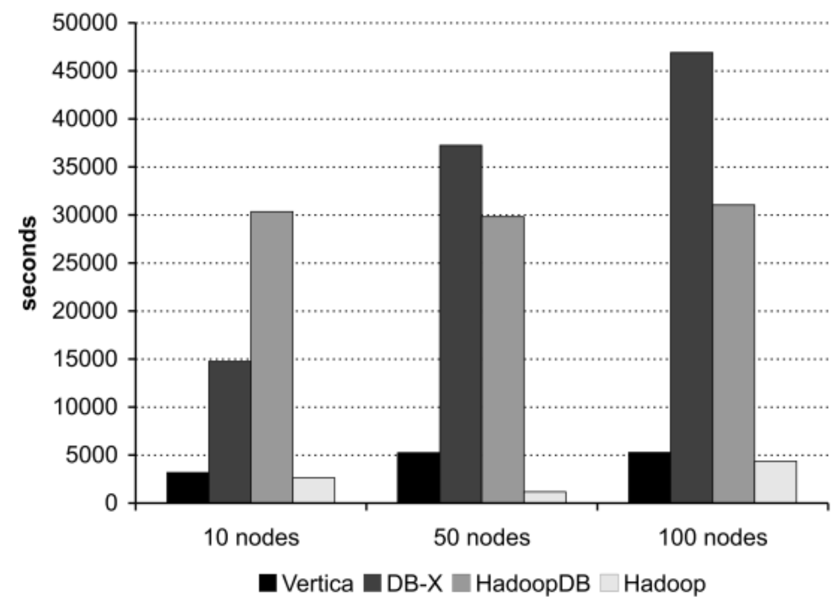
Join

134,000 joined records
over 20gb of date

HadoopDB loading times



Load Grep (0.5GB/node)



Load UserVisits (20GB/node)

HadoopDB

☺ Good performance

☺ Scalable

☺ Fault tolerant

☺ Heterogeneous node compatible

☺ Make any DBMS a distributed system

☹ Data Loader: All *a-priori* loading problems

Invisible loading

Load DBMS with data from Hadoop at run-time

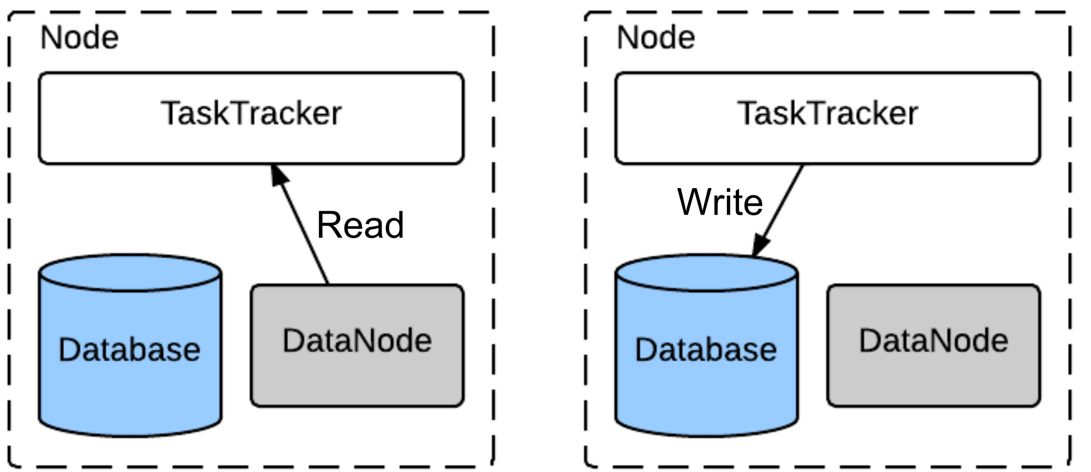
Invisibility objective

- Minimal human effort
- Minimal increase in response time

Use a DBMS as a cache for the raw data

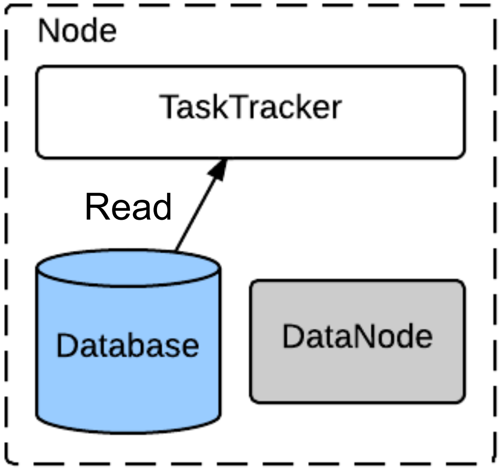
Invisible loading

Use code for tuple parsing and extraction to invisibly load the parsed tuples into a DBMS



Invisible loading

On next data access, it can be read from DBMS



Invisible loading: Parser

Parser extends inputFormat

- `getAttribute(int index)`
- Code for tuple parsing and extraction

Map takes a Parser as input

Invisible loading

☺ Incremental data reorganization

☺ Almost no overhead on MR Jobs

☺ Optimizes future access speeds

☹ Data duplication (No GC)

Outline

1. Introduction

2. Adaptive Indexing

3. Hybrid MapReduce

4. NoDB

5. Summary

NoDB

New DBMS paradigm

Do not require data loading

Maintains feature set of modern DBMS

Replaces physical storage with raw files

PostgresRaw

NoDB Implementation

Replaces TableScan Operator

CSV Files

Optimizations

PostgresRaw Optimizations

Selective...

a. Tokenizing
b. Parsing
c. Tuple formation

Indexing

Auto Tuning

Caching

Statistics

a. Selective tokenizing

111;222;"third";garbage;...

Supposing we want attributes **1** and **3**

We can stop tokenizing at the third

Saves CPU time

b. Selective parsing

111;222;"third";garbage;...

In memory:

111	`6F`	Parsed to int
222	`32 32 32`	Keep as string
"third"	`74 68 69 72 64`	

Also: delayed parsing

c. Selective tuple formation

111;222;"third";garbage;...

```
(111, "third")
```

Final tuple containing only attributes 1 and 3

CPU bound

Indexing

```
Year;  Make;      Model;   Liters
1997;  BMW;       E89;     2,34
2011;  Mercedes;  SLS;     2
```

Looks nice :)

Indexing

NOT :(

```
Year;Make;Model;Liters¶1997;BMW;E89;2,34¶2011;Mercedes;SLS;2
```

Sequentially reading each time is not an option

Solution

- Keep an index of the already used attributes
- Skip file reading to this positions

Indexing

Positional Map

Dynamically created according to queries

Tuple 1		Tuple 2		Tuple 3	
Attribute 1	Attribute 3	Attribute 1	Attribute 3	Attribute 1	Attribute 3
0	10	23	32	41	55

```
Year;Make;Model;Liters¶1997;BMW;E89;2,34¶2011;Mercedes;SLS;2
```

Updates

First case, no positions change

Tuple 1		Tuple 2		Tuple 3	
Attribute 1	Attribute 3	Attribute 1	Attribute 3	Attribute 1	Attribute 3
0	10	23	32	41	55

`Year;Make;Model;Liters¶`**`1989;BBB;CCC;4,44`**`¶2011;Mercedes;SLS;2`

Updates

Second case, positions change.

First option, update index.

Tuple 1		Tuple 2		Tuple 3	
Attribute 1	Attribute 3	Attribute 1	Attribute 3	Attribute 1	Attribute 3
0	10	23	~~32~~ 30 (-2)	~~41~~ 37 (-4)	~~55~~ 51 (-4)

```
Year;Make;Model;Liters¶1989;B;C;4,44¶2011;Mercedes;SLS;2
```

Updates

Second case, positions change.
Second option, throw it partially (or fully) away.

Tuple 1		Tuple 2
Attribute 1	Attribute 3	Attribute 1
0	10	23

Year;Make;Model;Liters¶1**989;B;C;4,44**¶2011;Mercedes;SLS;2

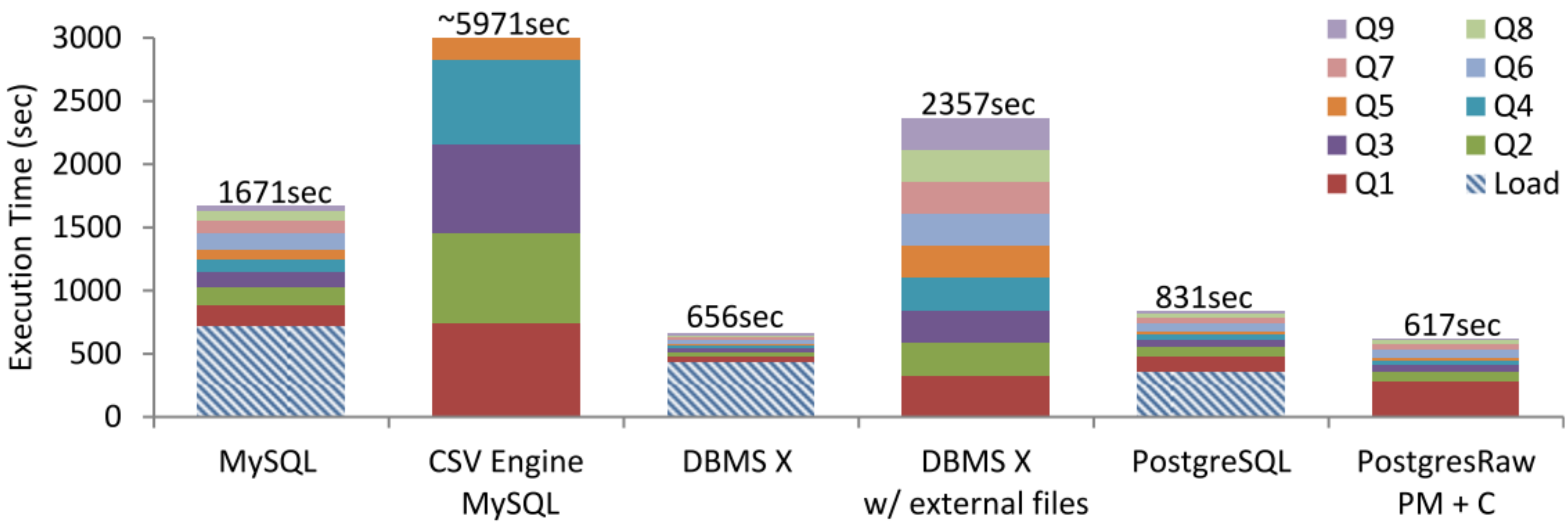
Index will automatically reconstruct itself

Traditional optimizations

Caching

Statistics

NoDB Performance Compared



NoDB

- ☺ Great DBMS + Raw hybrid
- ☺ Competitive performance with traditional DBs
- ☺ Eliminates loading times
- ☺ Queries get faster with time

☹ Updates

Outline

1. Introduction
2. Adaptive Indexing
3. Hybrid MapReduce
4. NoDB
5. Summary

Summary

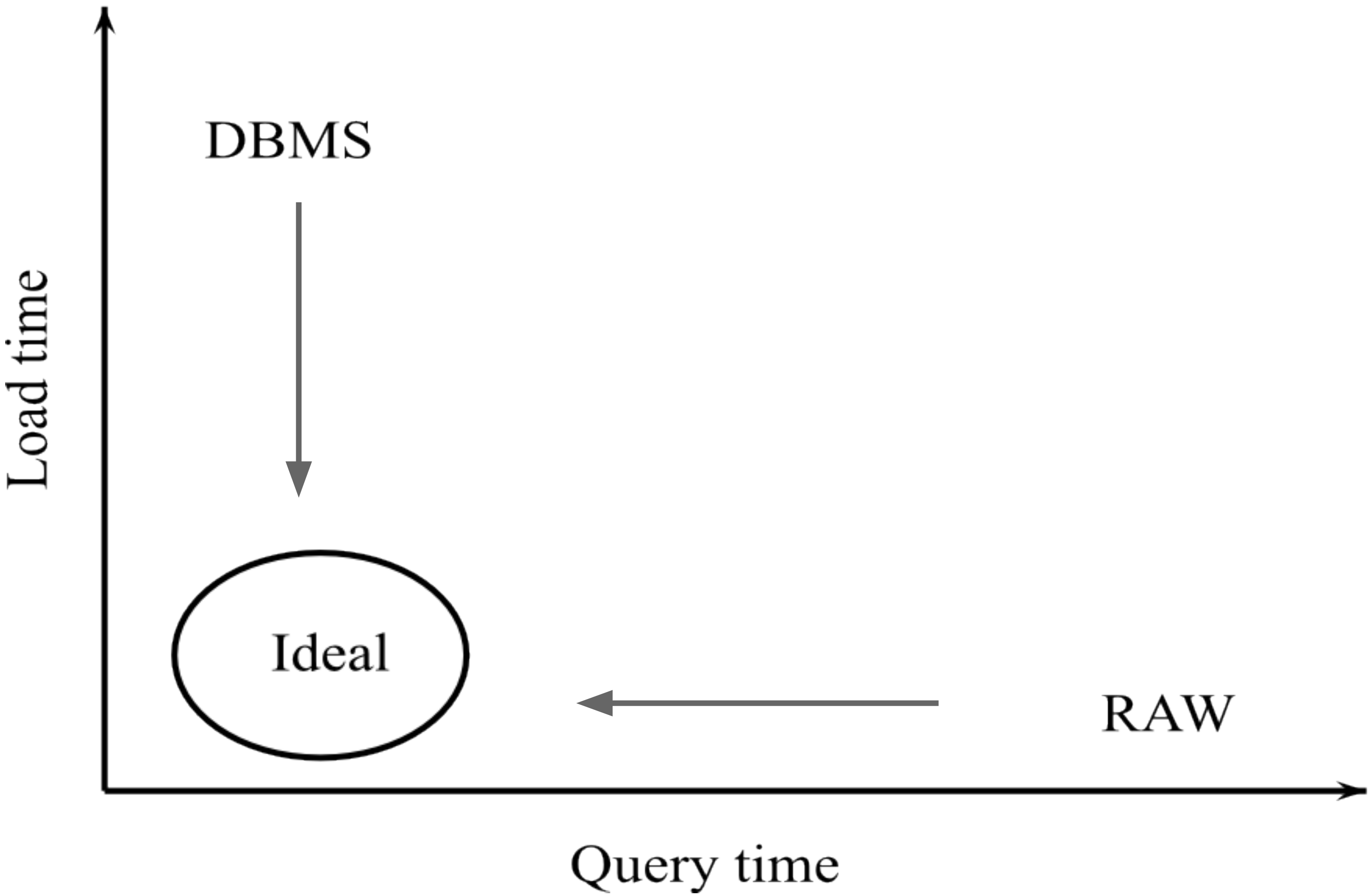
Mature solutions: high load or query time

- No index → High query time
- Load all data → High delay (load time)

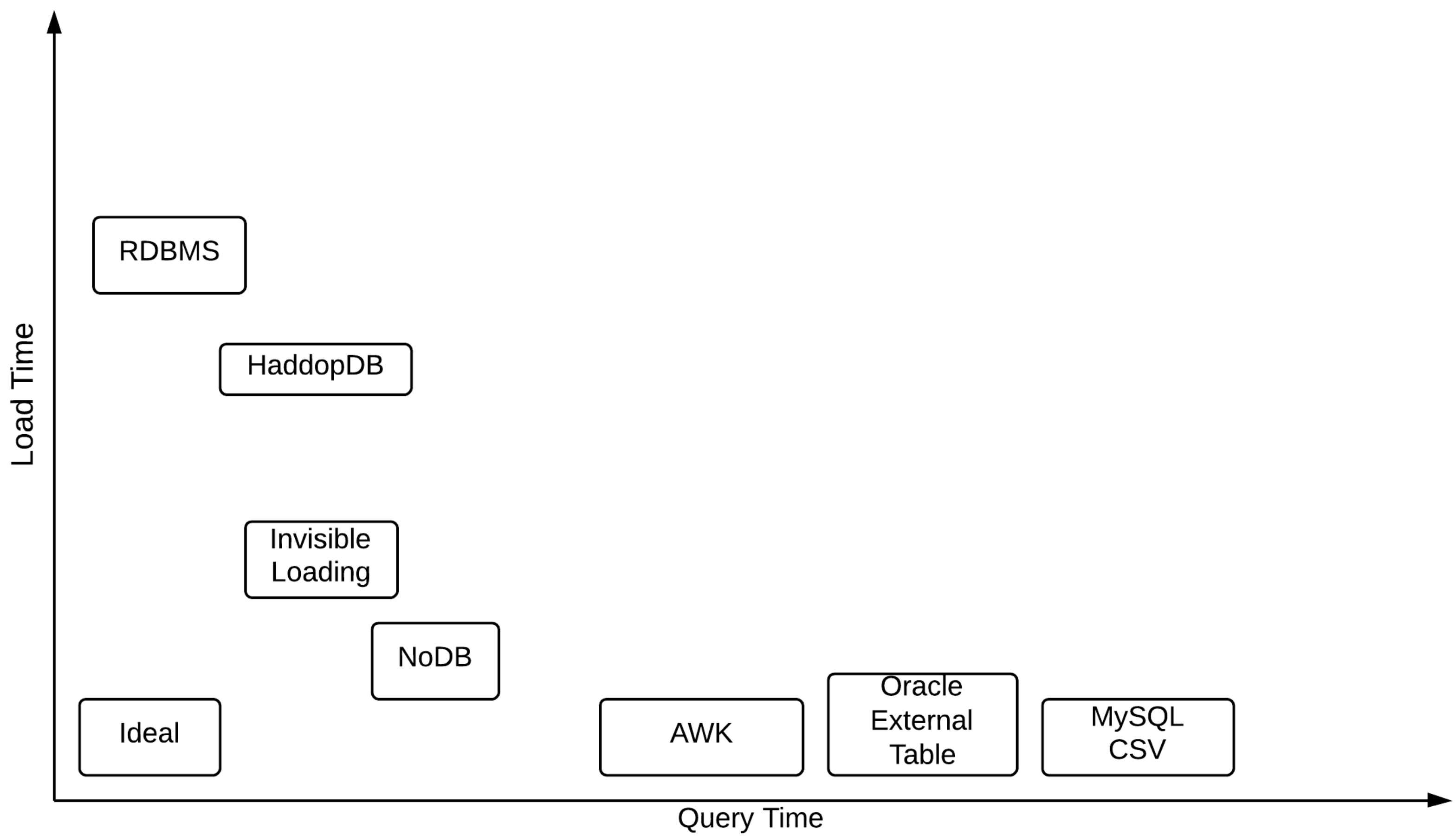
Hybrid solutions

- Bring indexes to in-situ processing
- Adaptive indexing
- HadoopDB
- NoDB

Remember..



Conclusions



References

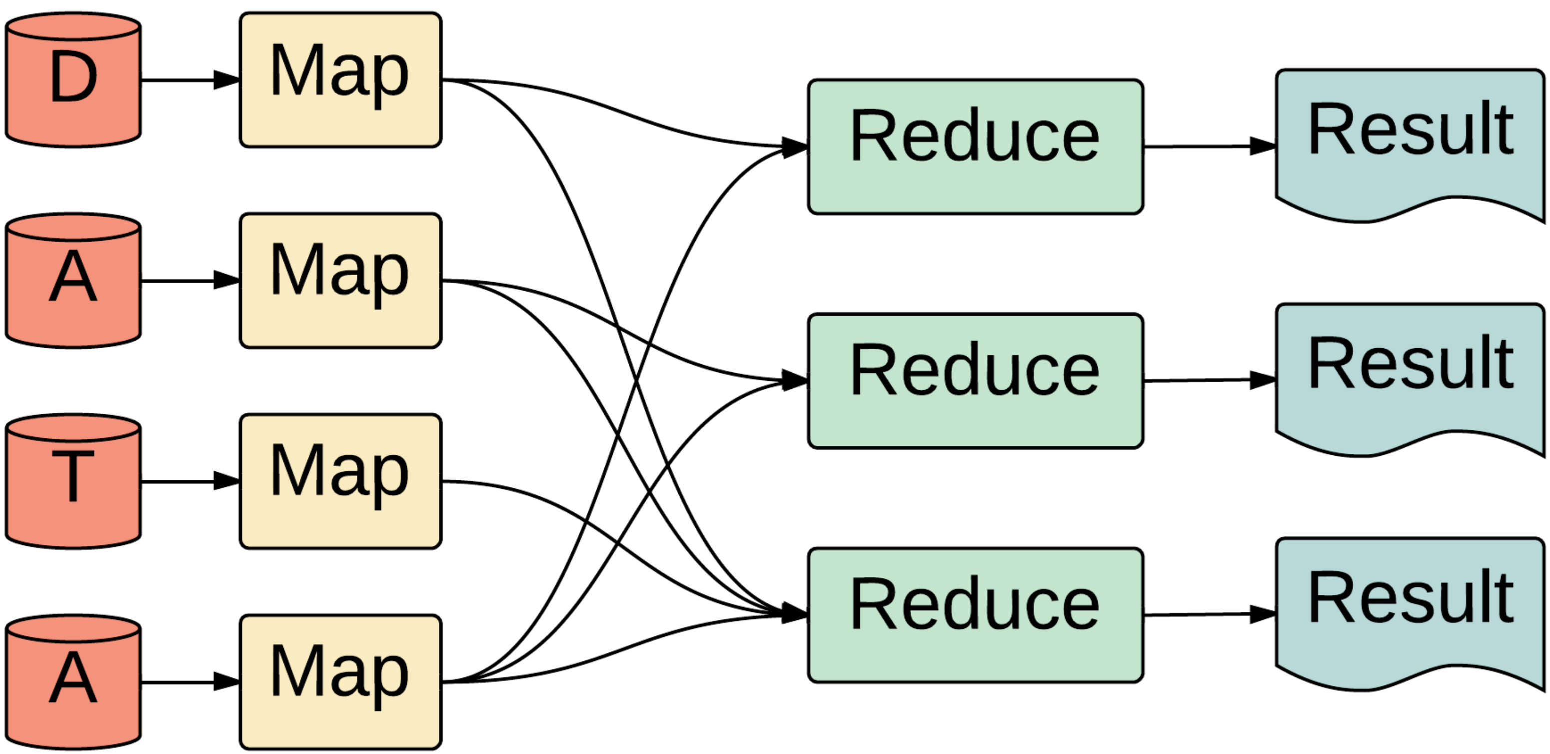
1. Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. **HadoopDB**: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proceedings of the VLDB Endowment,*
2. Azza Abouzied, Daniel J. Abadi, and Avi Silberschatz. **Invisible loading**: access-driven data transfer from raw files into database systems. *Proceedings of the 16th International Conference on Extending Database Technology, pages 1–10, 2013.*
3. Renata Borovica, Stratos Idreos, and Anastasia Ailamaki. **NoDB** : Efficient Query Execution on Raw Data Files *Categories and Subject Descriptors. pages 241–252.*
4. Goetz Graefe and Harumi Kuno. **Adaptive indexing** for relational keys. *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pages 69–74, 2010.*
5. Felix Halim, S Idreos, P Karras, and RHC Yap. Stochastic **database cracking**: Towards robust adaptive indexing in main-memory column-stores. *Proceedings of the VLDB Endowment (PVLDB),*
6. Tony Hey, Stewart Tansley, and Kristin Tolle, editors. **The Fourth Paradigm**: Data-Intensive Scientific Discovery. *Microsoft Research, Redmond, Washington, 2009.*
7. Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. **Here are my data files. here are my queries. where are my results**. *Proceedings of 5th Biennial Conference on Innovative Data Systems Research, pages 57–68, 2011.*
8. Christopher Olston, Benjamin Reed, Ravi Kumar, and Andrew Tomkins. **Pig Latin**: A Not-So-Foreign Language for Data Processing.
9. Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. **Hive -** A Ware-housing Solution Over a Map-Reduce Framework. *PVLDB*

Questions?

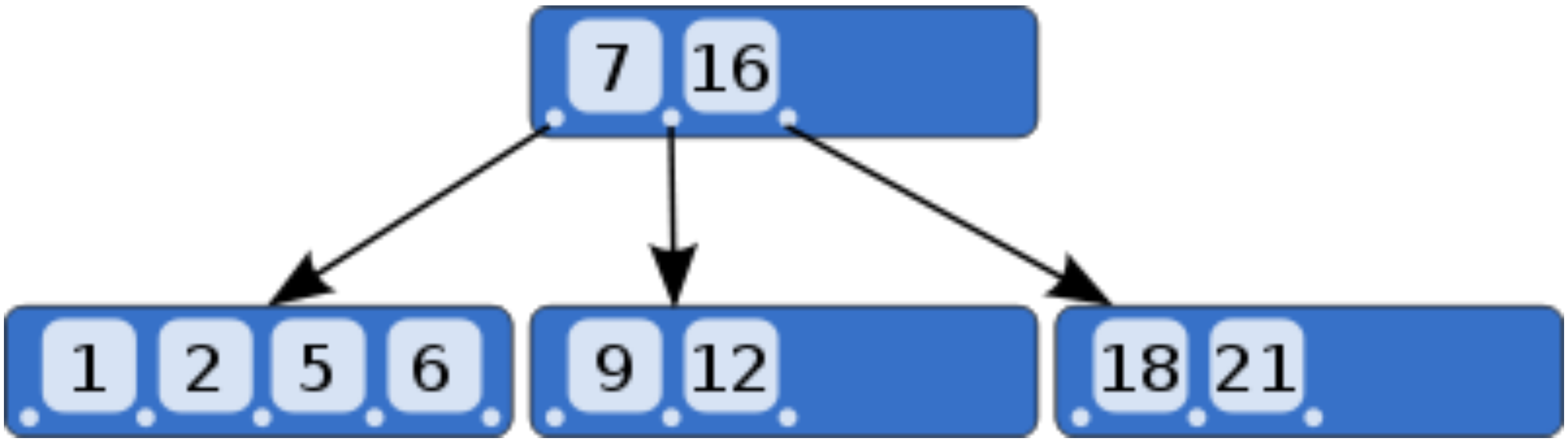
Thank you!

MapReduce

Can be classified as distributed raw file parsing



Adaptive merging



Database Cracking