Introduction to Hadoop

ROYAL INSTITUTE OF TECHNOLOGY

ID2210

Jim Dowling

Large Scale Distributed Computing

- In #Nodes
  - BitTorrent (millions)
  - Peer-to-Peer
- In #Instructions/sec
  - Teraflops, Petaflops, Exascale
  - Super-Computing

- In #Bytes stored
  - Facebook: 300+ Petabytes (April 2014)*
  - Hadoop
- In #Bytes processed/time
  - Google processed 24 petabytes of data per day in 2013
  - Colossus, Spanner, BigQuery, BigTable, Borg, Omega, ..

*http://www.adweek.com/socialtimes/orcfile/434041

Where does Big Data Come From?

• On-line services	PBs per day
• Scientific instruments	PBs per minute
• Whole genome sequencing	250 GB per person
• Internet-of-Things	Will be lots!

ROYAL INSTITUTE OF TECHNOLOGY

What is Big Data?

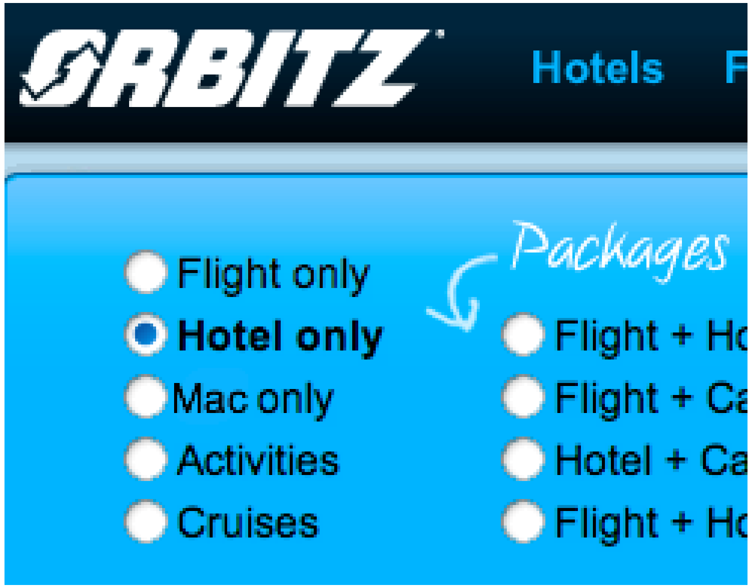
Small Data

Big Data

Why is Big Data "hot"?

- Companies like Google and Facebook have shown how to **extract value from Big Data**

Orbitz looks for higher prices from Safari users [WSJ'12]



Why is Big Data "hot"?

- Big Data helped Obama win the 2012 election through **data-driven decision making***

Data said: middle-aged females like contests, dinners and celebrity

*http://swampland.time.com/2012/11/07/inside-the-secret-world-of-quants-and-data-crunchers-who-helped-obama-win/

KTH VETENSKAP OCH KONST
ROYAL INSTITUTE OF TECHNOLOGY

Why is Big Data Important in Science?

- In a wide array of academic fields, the ability to effectively process data is superseding other more classical modes of research.

"More data trumps better algorithms"*

*"The Unreasonable Effectiveness of Data" [Halevey et al 09]

4 Vs of Big Data

- Volume
- Velocity
- Variety
- Veracity/Variability/Value

A quick historical tour of data systems


In the
Beginning

Batch Sequential Processing

Scan → Sort

IBM 082 Punch Card Sorter

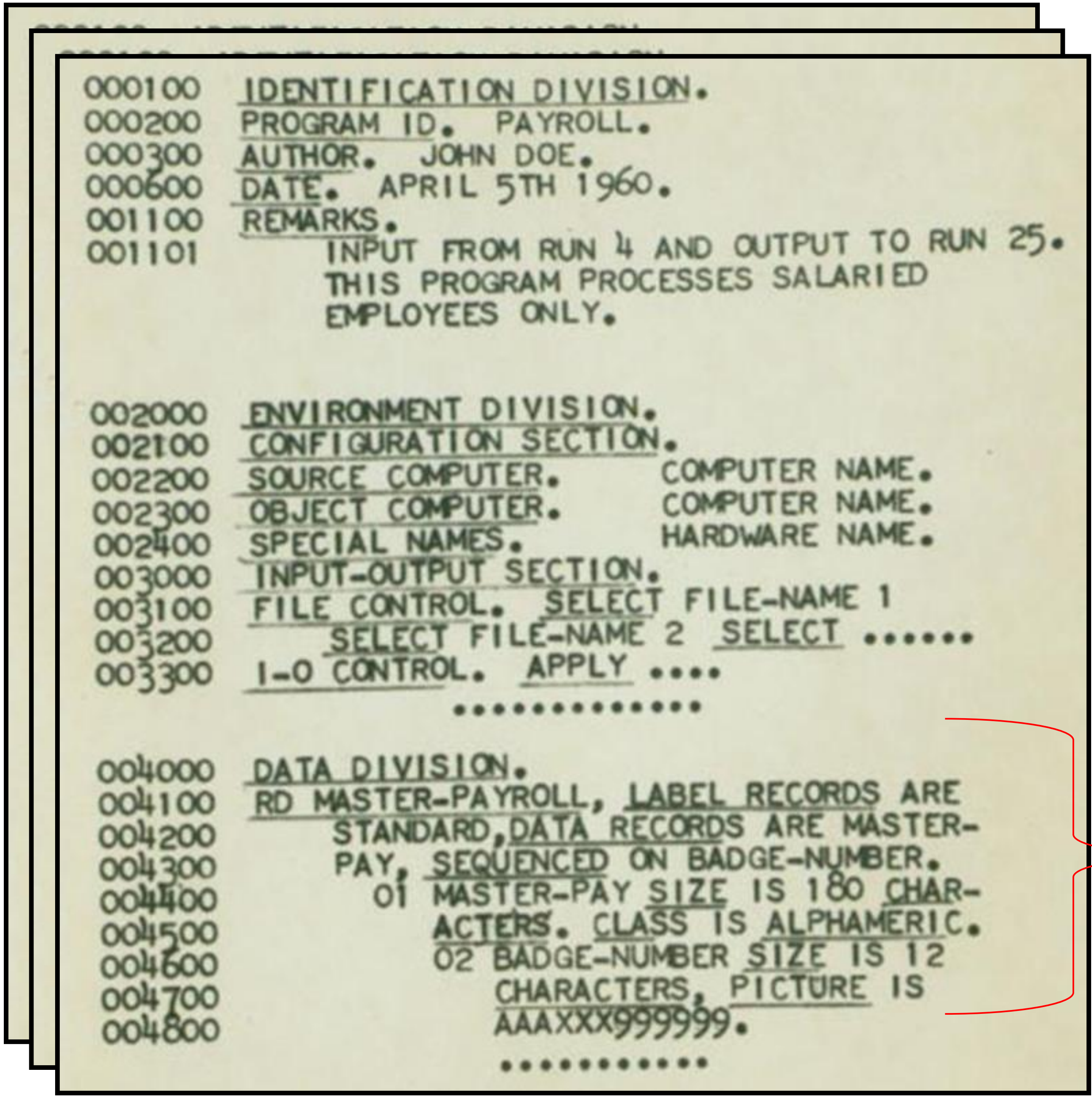
No Fault Tolerance ☺

1960s

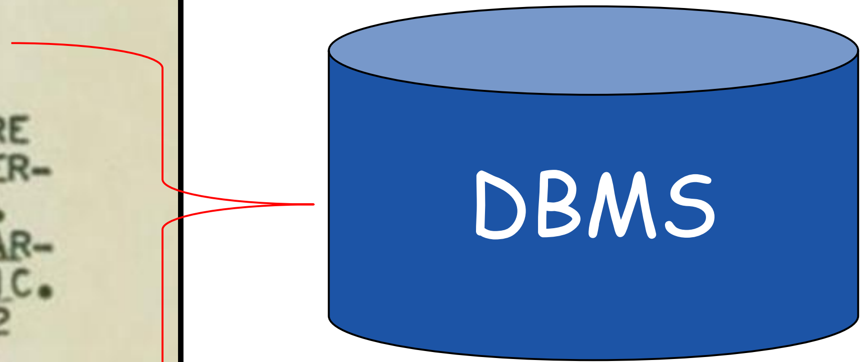

EVA
Colgate
Makaala
KEIKI
© 2001 National Geographic Society. All rights reserved.
nationalgeographic.com

First Database Management Systems

COBOL



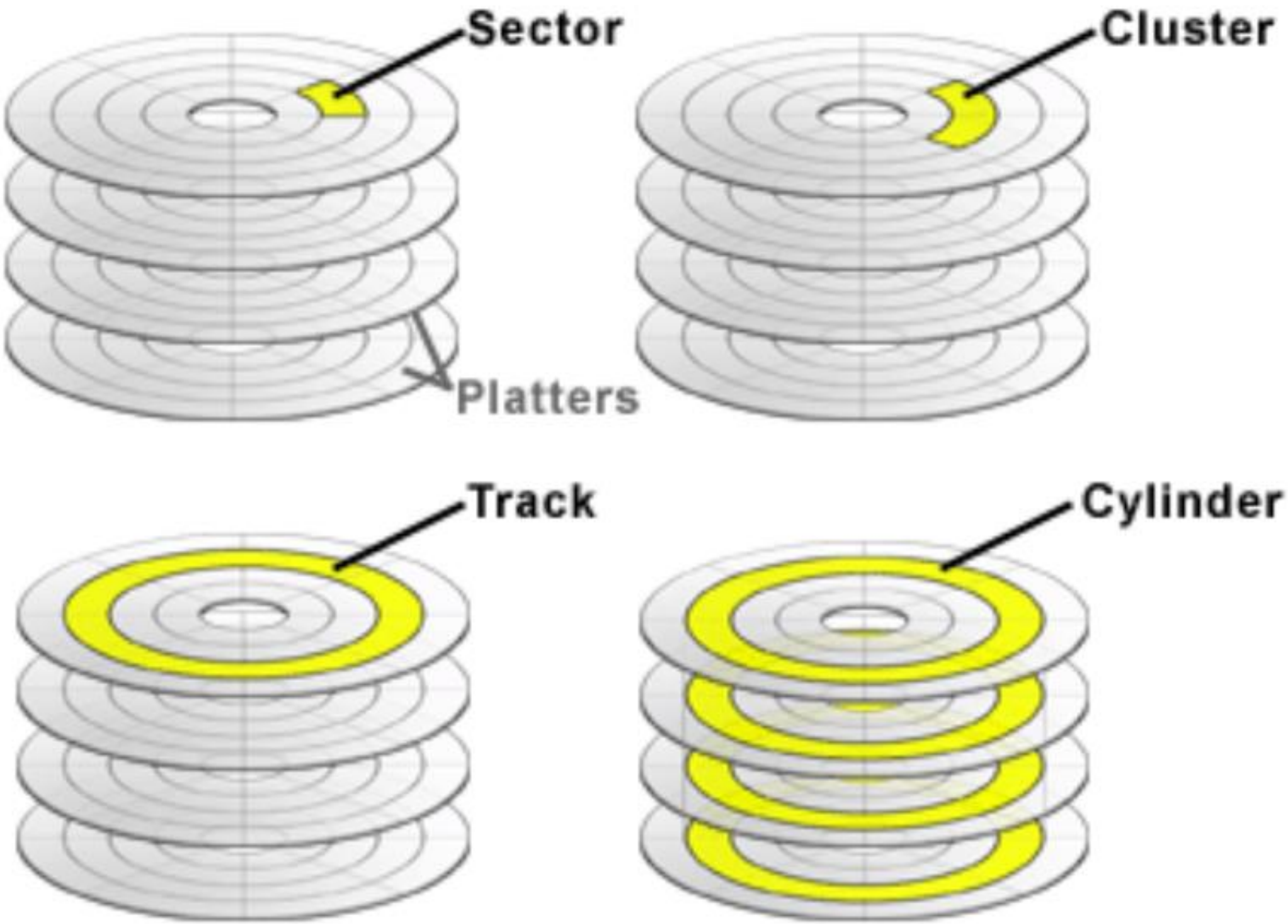
```
000100  IDENTIFICATION DIVISION.
000200  PROGRAM ID.  PAYROLL.
000300  AUTHOR.  JOHN DOE.
000600  DATE.  APRIL 5TH 1960.
001100  REMARKS.
001101        INPUT FROM RUN 4 AND OUTPUT TO RUN 25.
              THIS PROGRAM PROCESSES SALARIED
              EMPLOYEES ONLY.

002000  ENVIRONMENT DIVISION.
002100  CONFIGURATION SECTION.
002200  SOURCE COMPUTER.        COMPUTER NAME.
002300  OBJECT COMPUTER.        COMPUTER NAME.
002400  SPECIAL NAMES.          HARDWARE NAME.
003000  INPUT-OUTPUT SECTION.
003100  FILE CONTROL.  SELECT FILE-NAME 1
003200        SELECT FILE-NAME 2  SELECT ......
003300  I-O CONTROL.  APPLY ....
                  .............

004000  DATA DIVISION.
004100  RD MASTER-PAYROLL, LABEL RECORDS ARE
004200        STANDARD, DATA RECORDS ARE MASTER-
004300        PAY, SEQUENCED ON BADGE-NUMBER.
004400          01 MASTER-PAY SIZE IS 180 CHAR-
004500             ACTERS. CLASS IS ALPHAMERIC.
004600             02 BADGE-NUMBER SIZE IS 12
004700                CHARACTERS, PICTURE IS
004800                AAAXXX999999.
                  ...........
```



Hierarchical and Network Database Management Systems

You had to know what data you want, and how to find it

Early DBMS' were Disk-Aware



Codd's Relational Model

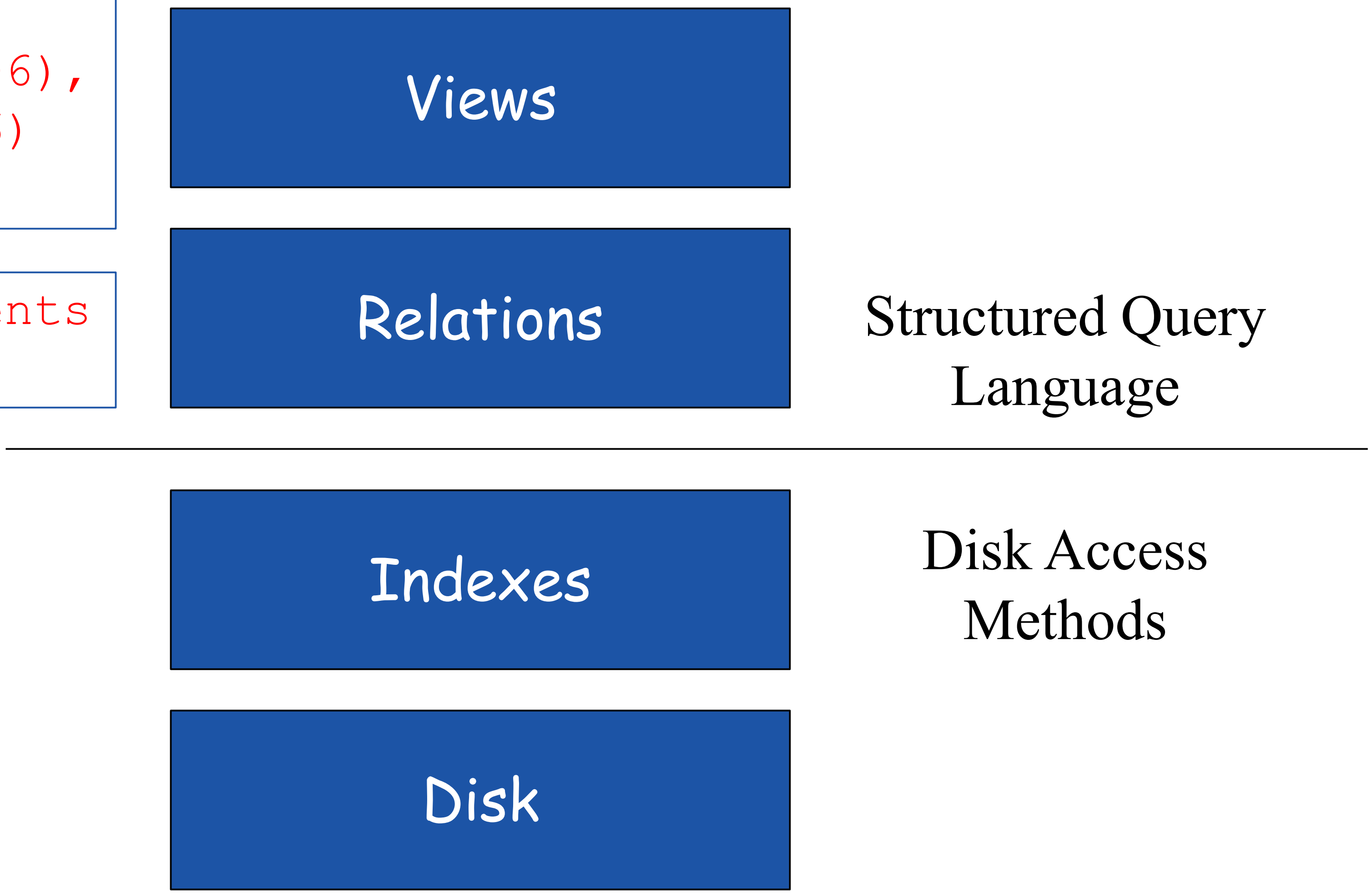
Just tell me
the data you want,
the system will
find it.

SystemR

```sql
CREATE TABLE Students(
id INT PRIMARY_KEY,
firstname VARCHAR(96),
lastname VARCHAR(96)
);
```

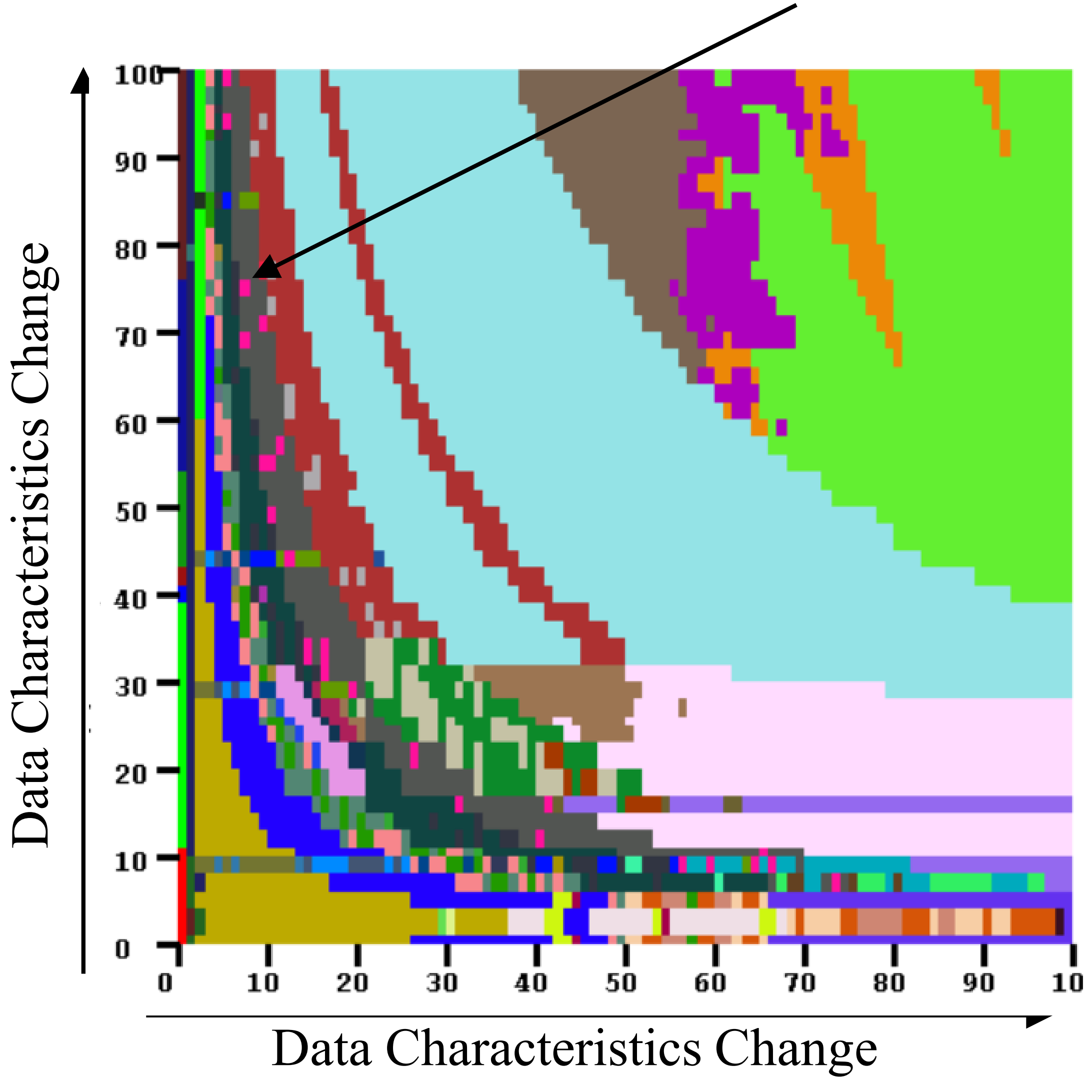
```sql
SELECT * FROM Students
WHERE id > 10;
```

?



Finding the Data using a Query Optimizer

Each color represents a program in this plan diagram



- Each program produces the same result for the Query.
- Each program has *different performance characteristics* depending on changes in the data characteristics

What if I have lots of Concurrent Queries?

- Data Integrity using Transactions*

ACID

Atomicity Consistency Isolation Durability

*Jim Gray, ”The Transaction Concept: Virtues and Limitation”

In the 1990s
Data Read Rates Increased Dramatically

Distribute within a Data Center

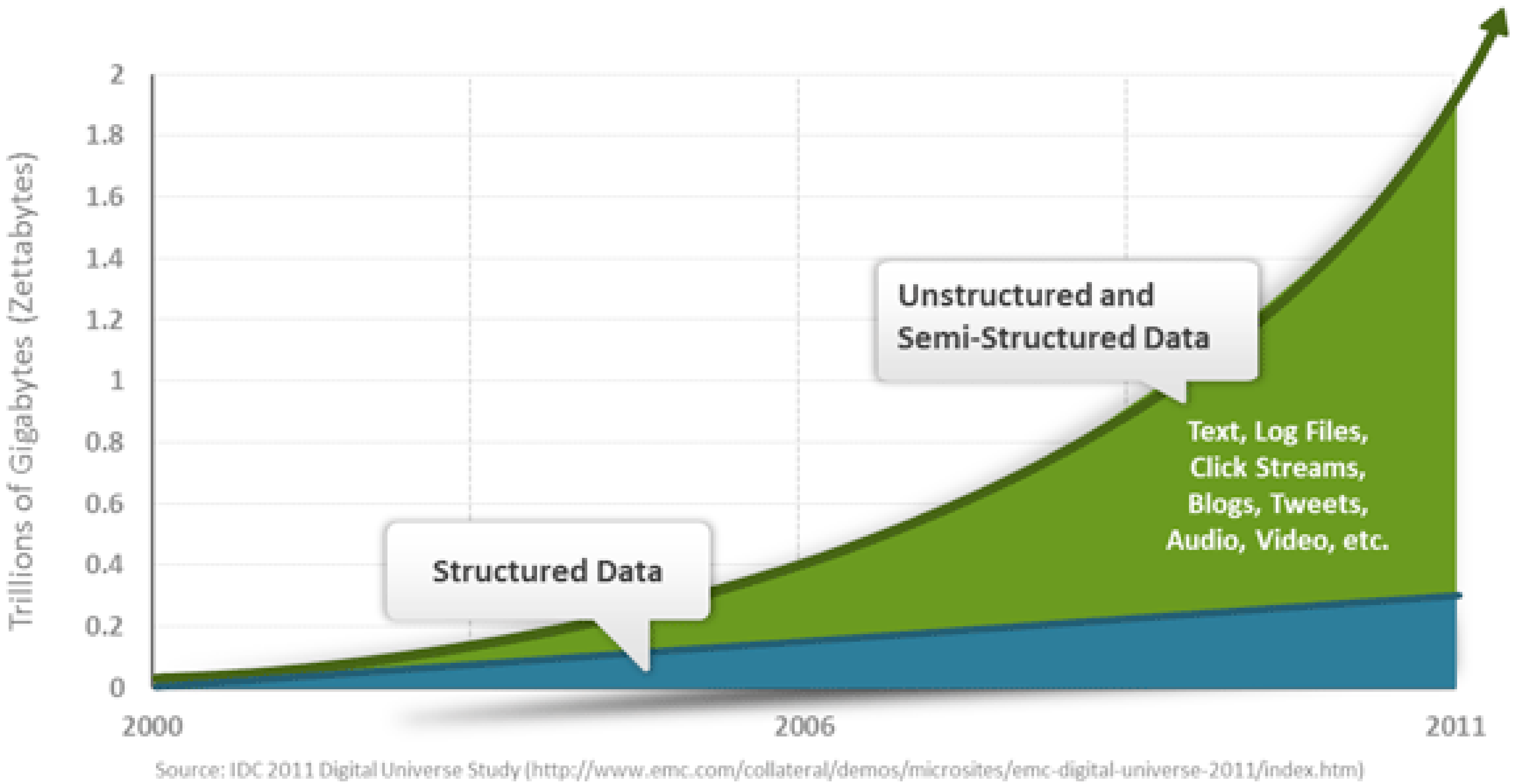
Master-Slave Replication

Data-location awareness is back:
Clients read from slaves, write to master.
Possibility of reading stale data.

In the 2000s

Data Write Rates Increased Dramatically

Unstructured Data explodes



Source: IDC whitepaper. As the Economy contracts, the Digital Universe Explodes. 2009

Key-Value stores don’t do Big Data yet. Existing Big Data systems currently only work for a single Data Centre.*

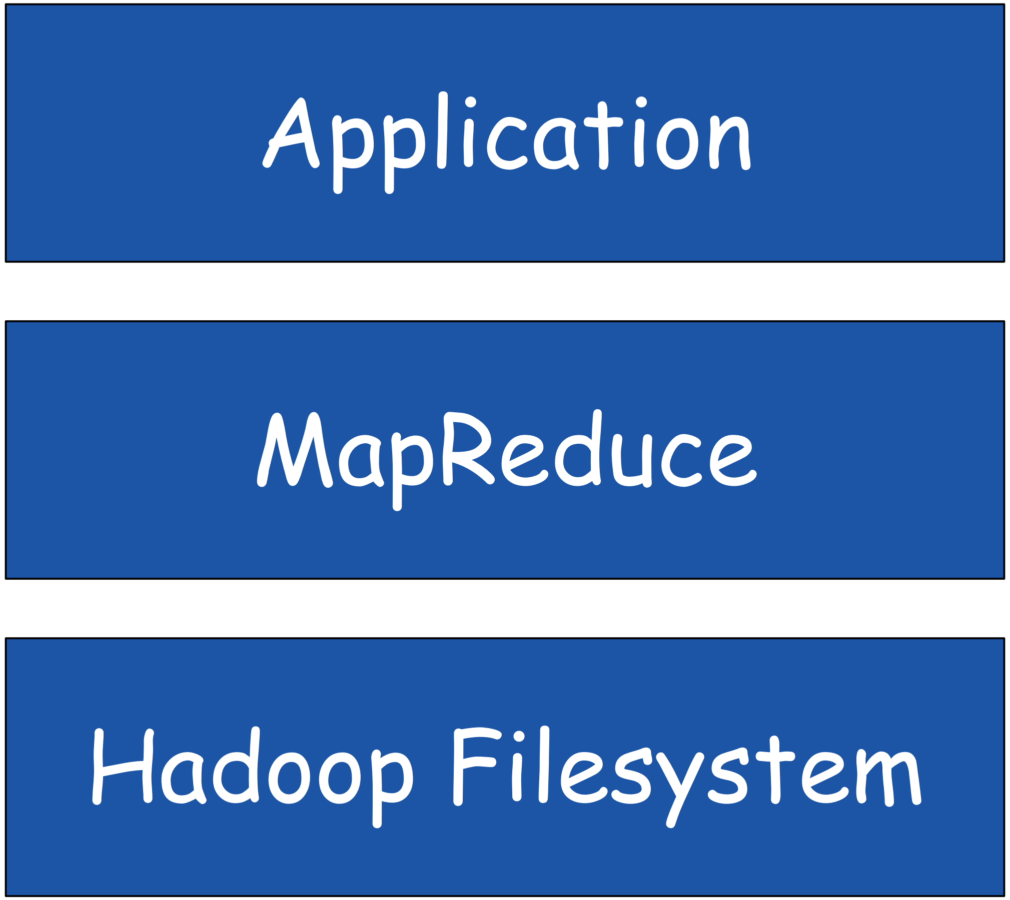
*The usual Google Exception applies

Storage and Processing of Big Data

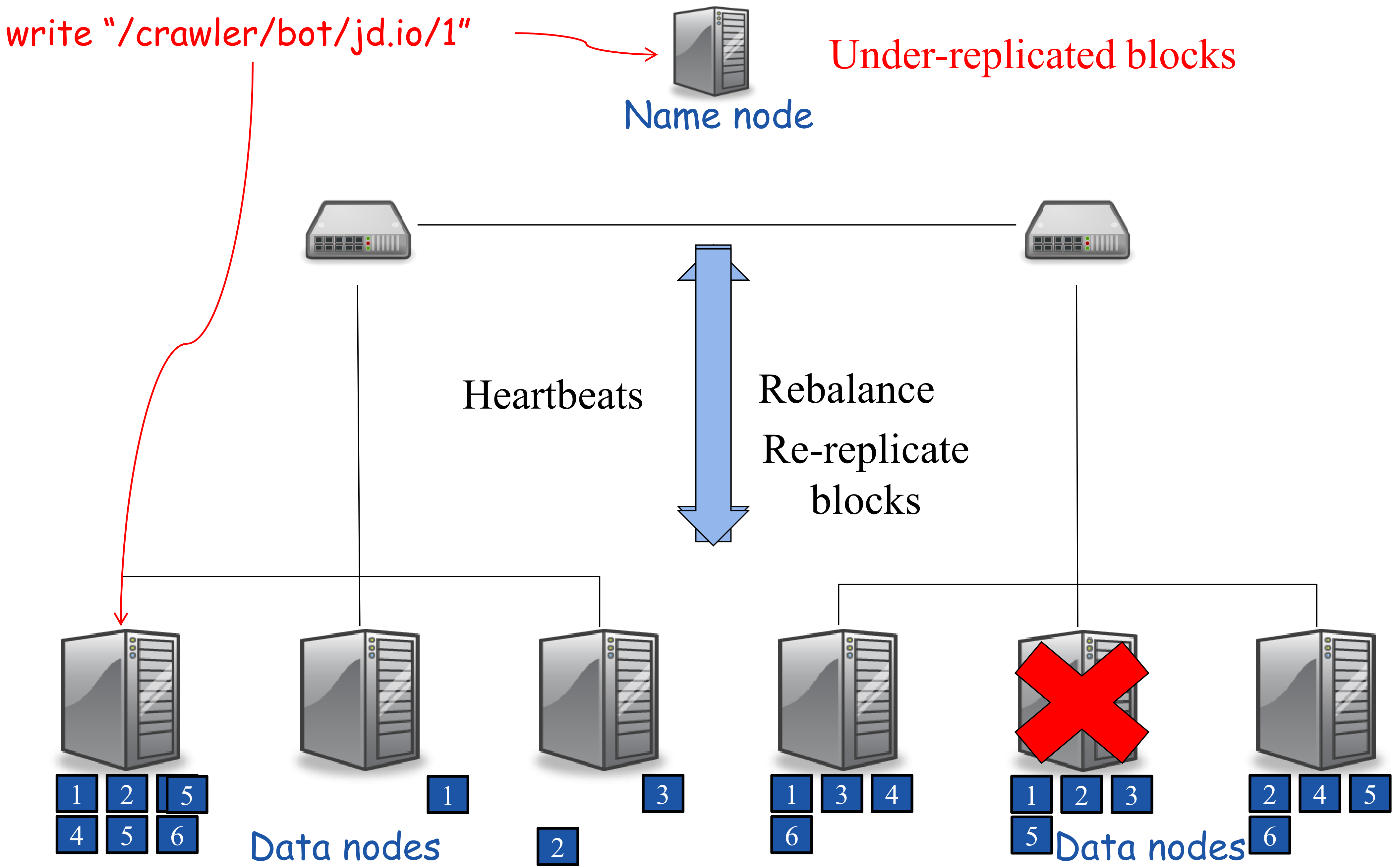
What is Apache Hadoop?

- Huge data sets and large files
  - Gigabytes files, petabyte data sets
  - Scales to thousands of nodes on commodity hardware
- No Schema Required
  - Data can be just copied in, extract required columns later
- Fault tolerant
- Network topology-aware, Data Location-Aware
- Optimized for analytics: high-throughput file access

Hadoop (version 1)



HDFS: Hadoop Filesystem

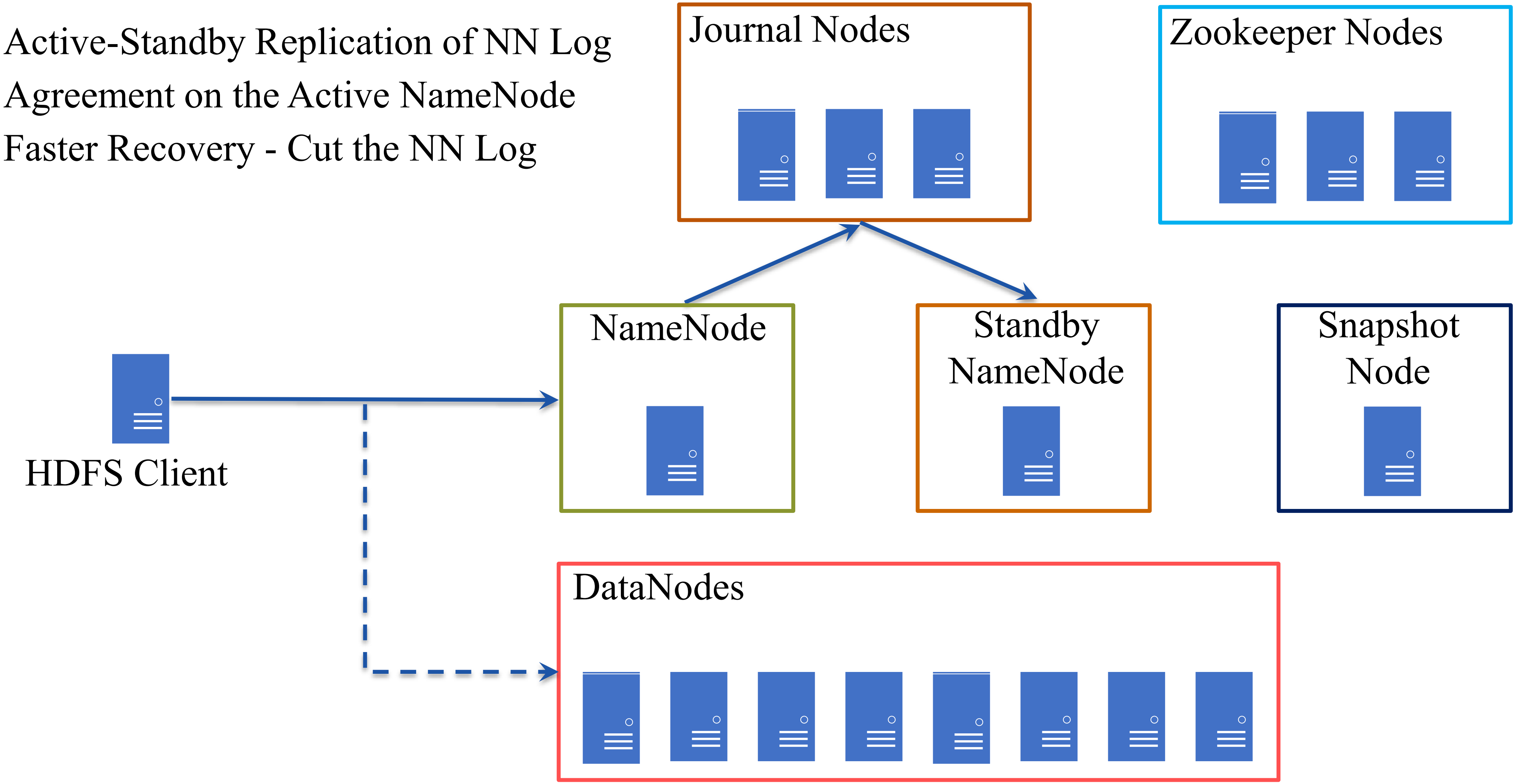


HDFS v2 Architecture

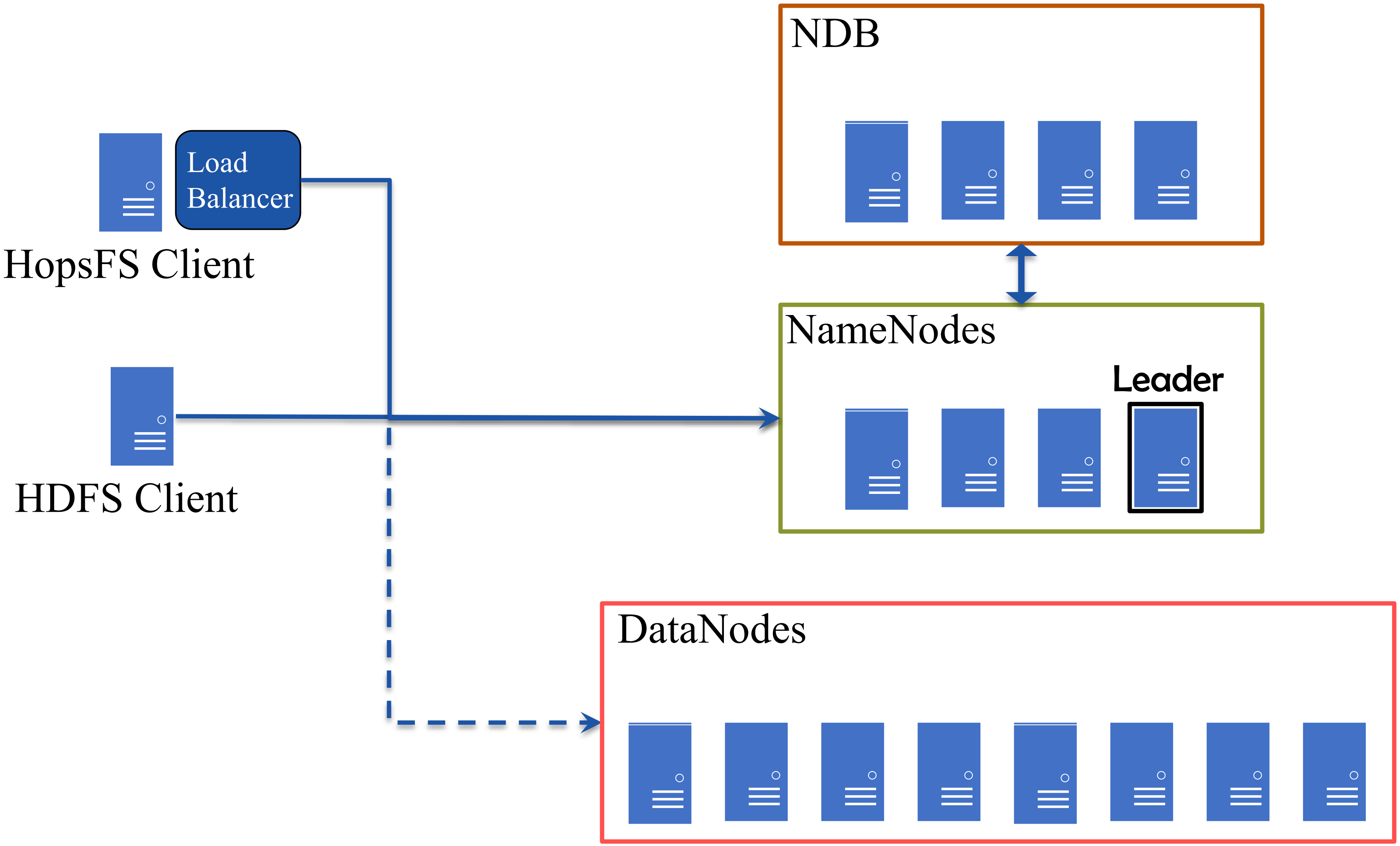
Active-Standby Replication of NN Log

Agreement on the Active NameNode

Faster Recovery - Cut the NN Log

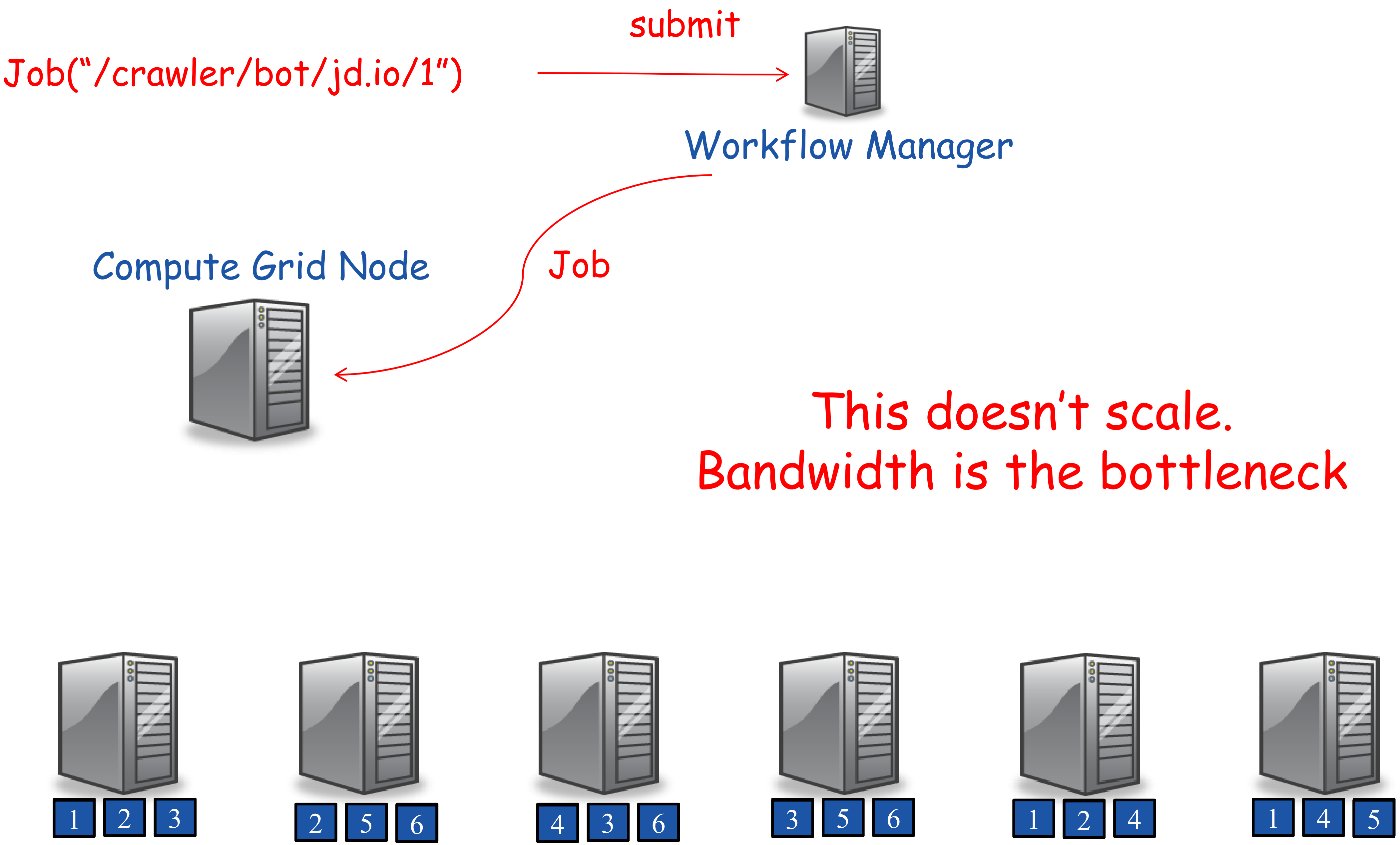


HopsFS Architecture

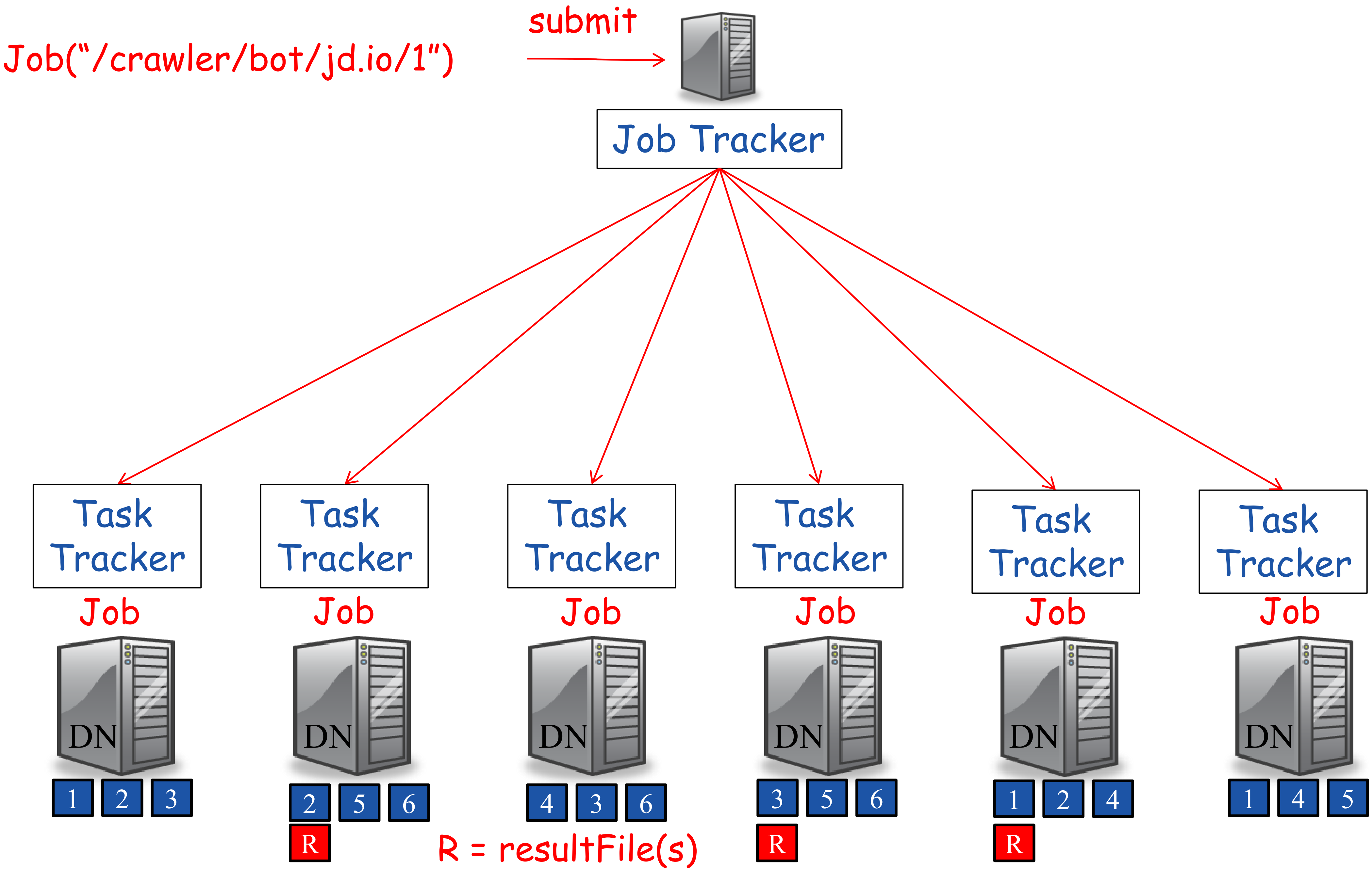


Processing Big Data

Big Data Processing with No Data Locality



MapReduce – Data Locality



MapReduce*

1. Programming Paradigm

2. Processing Pipeline (moving computation to data)

*Dean et al, OSDI'04

MapReduce Programming Paradigm

```
map(record) ->
          {(key_i, value_i), .., (key_l, value_l)}

reduce((key_i, {value_k, .., value_y}) -> output
```

MapReduce Programming Paradigm

- Also found in:

Functional programming languages

MongoDB

Cassandra

KTH VETENSKAP OCH KONST

ROYAL INSTITUTE OF TECHNOLOGY

Example: Building a Web Search Index

```
map(url, doc) ->
            {(term_i, url),(term_m, url)}

reduce((term,{url_k,..,url_y}) ->
            (term, (posting list of url, count))
```

Example: Building a Web Search Index

```
map( (“jd.io”, “A hipster website with news”))
->
{
     emit(“a”, “jd.io”),
     emit(“hipster”, “jd.io”),
     emit(“website”, “jd.io”),
     emit(“with”, “jd.io”),
     emit(“news”, “jd.io”)
}
```

Example: Building a Web Search Index

```
map( (“hn.io”, “Hacker hipster news”))
->
{
        emit(“hacker”, “hn.io”),
        emit(“hipster”, “hn.io”),
        emit(“news”, “hn.io”)
}
```

Example: Building a Web Search Index

```
reduce( “hipster”, { “jd.io”, “hn.io” }) ->
      ( “hipster”, ([“jd.io”, “hn.io”], 2))
```

ROYAL INSTITUTE OF TECHNOLOGY

Example: Building a Web Search Index

```
reduce( “website”, { “jd.io”}) ->
      ( “website”, ([“jd.io”], 1))
```

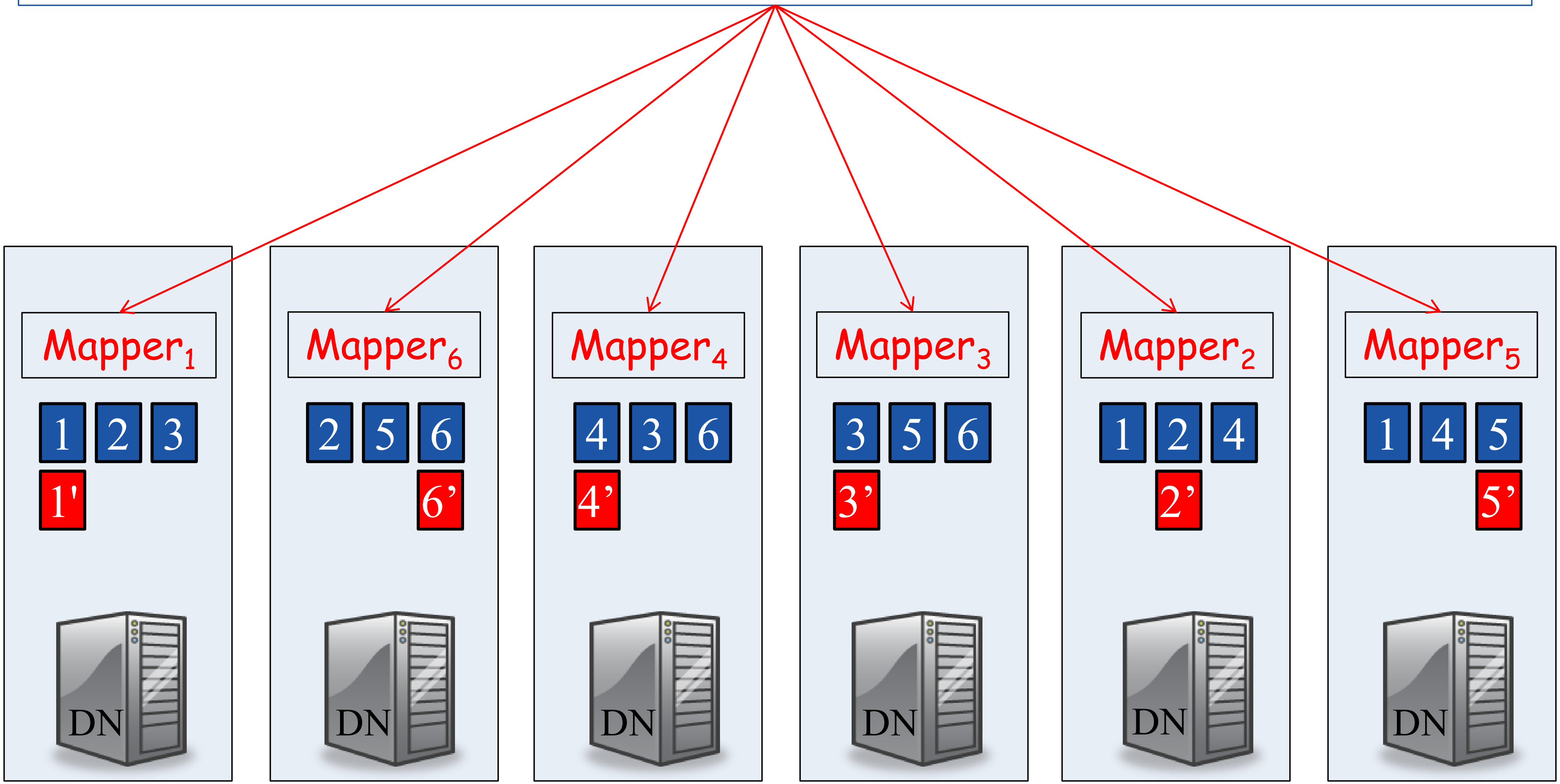
Example: Building a Web Search Index

```
reduce( “news”, { “jd.io”, “hn.io” }) ->
       ( “news”, ([“jd.io”, “hn.io”], 2))
```

Map Phase

MapReduce

$\texttt{map(url, doc) -> \{(term}_{\texttt{i}}\texttt{, url),(term}_{\texttt{l}}\texttt{, url)\}}$

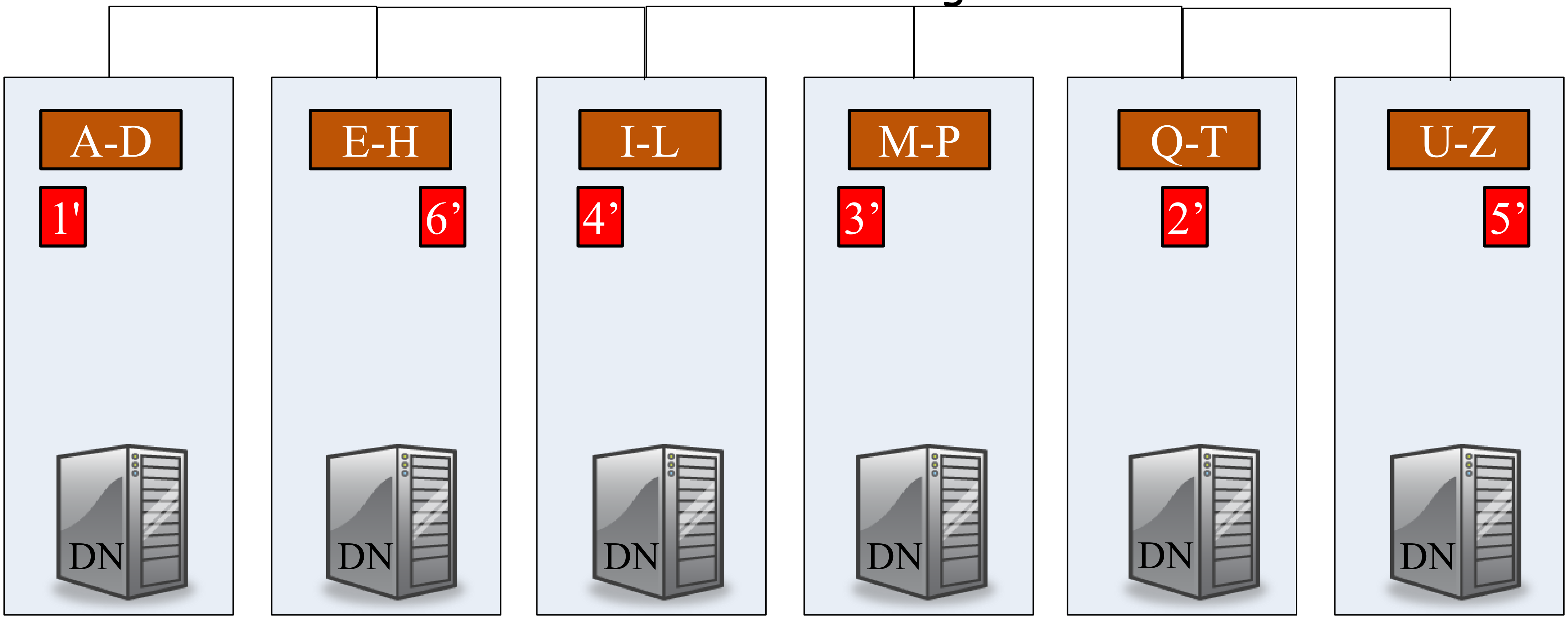


Shuffle Phase

MapReduce

```
group by term
```

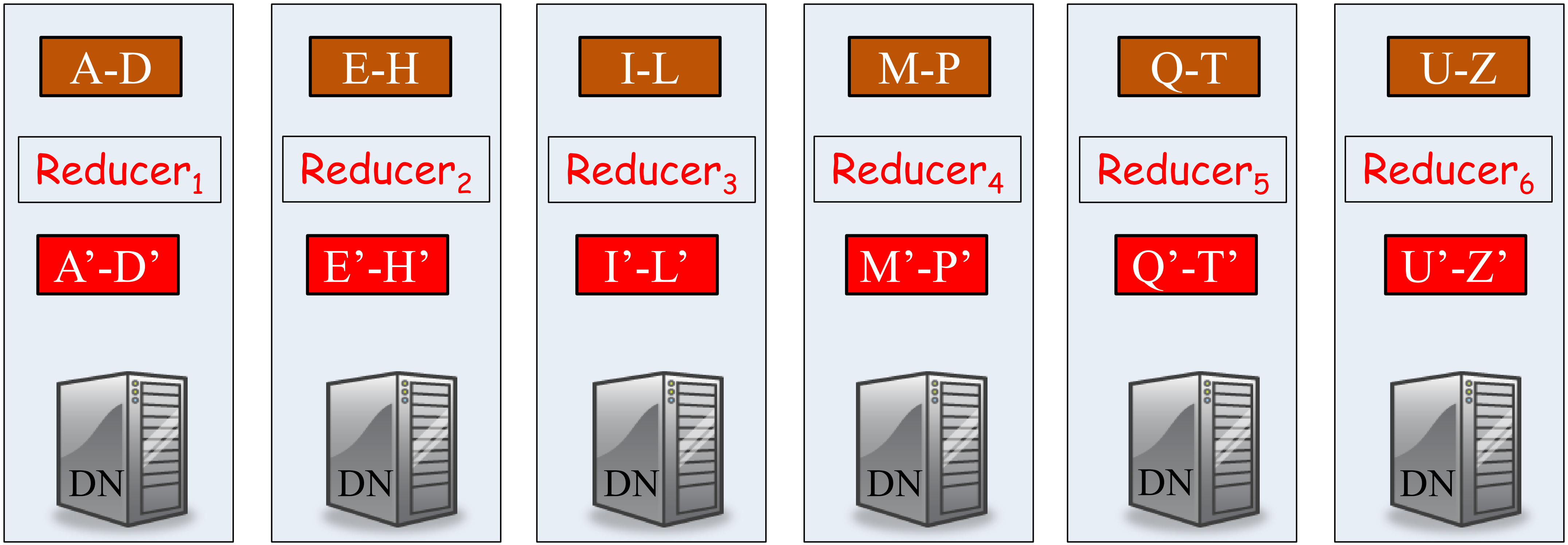
Shuffle over the Network using a Partitioner



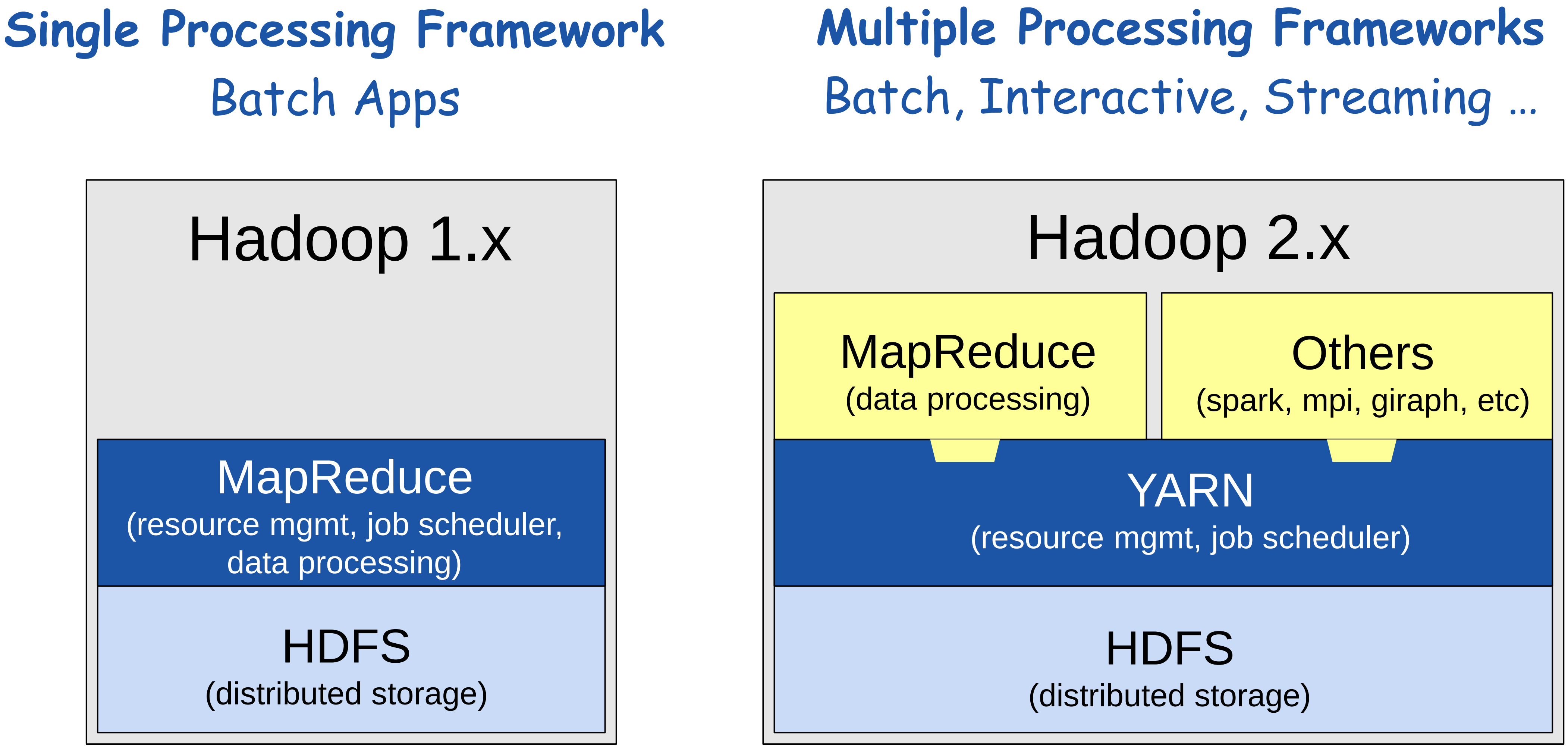
Reduce Phase

MapReduce

reduce((term,{url_k,url_y}) ->
(term, (posting list of url, count))

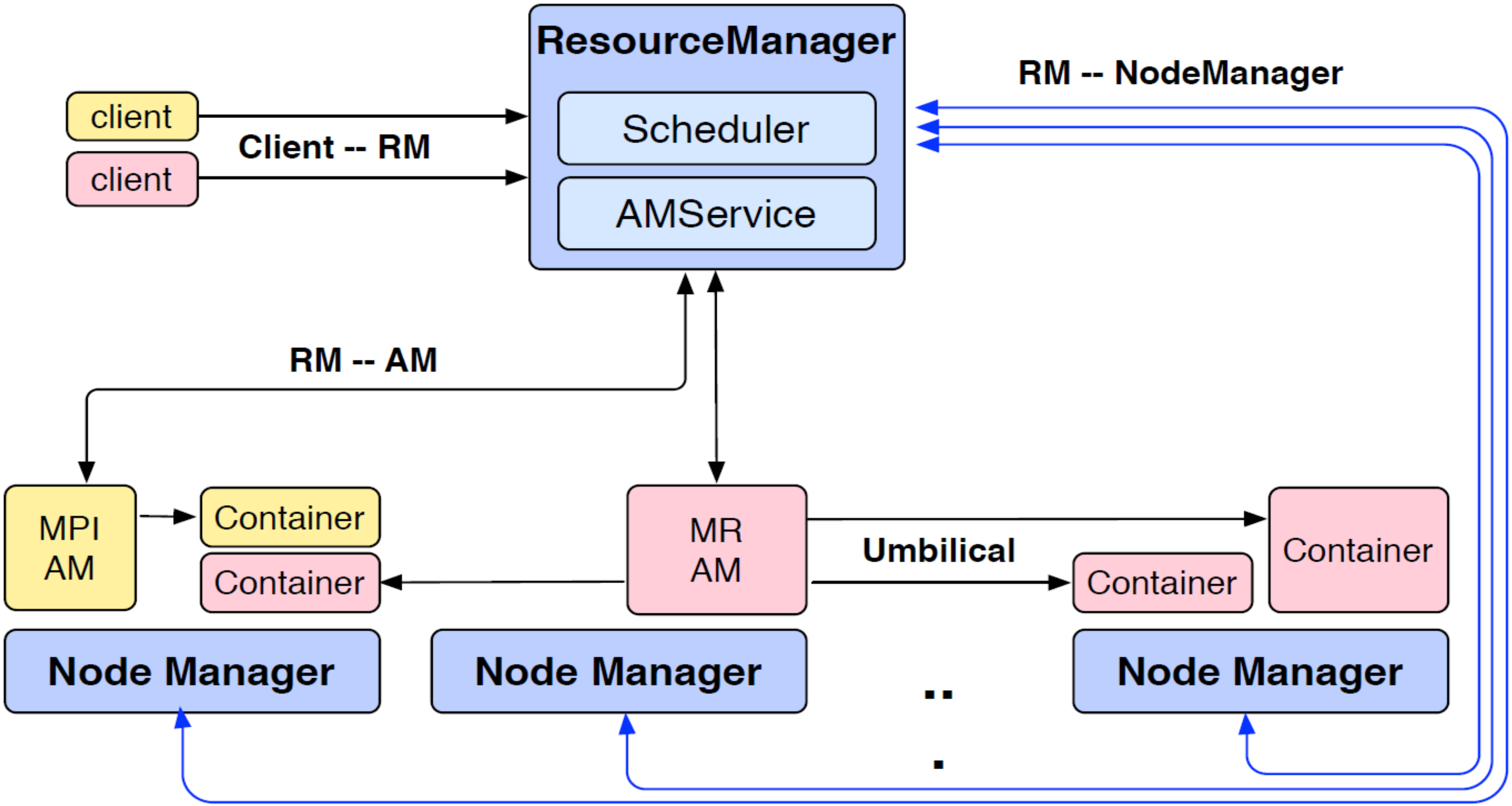


Hadoop 2.x



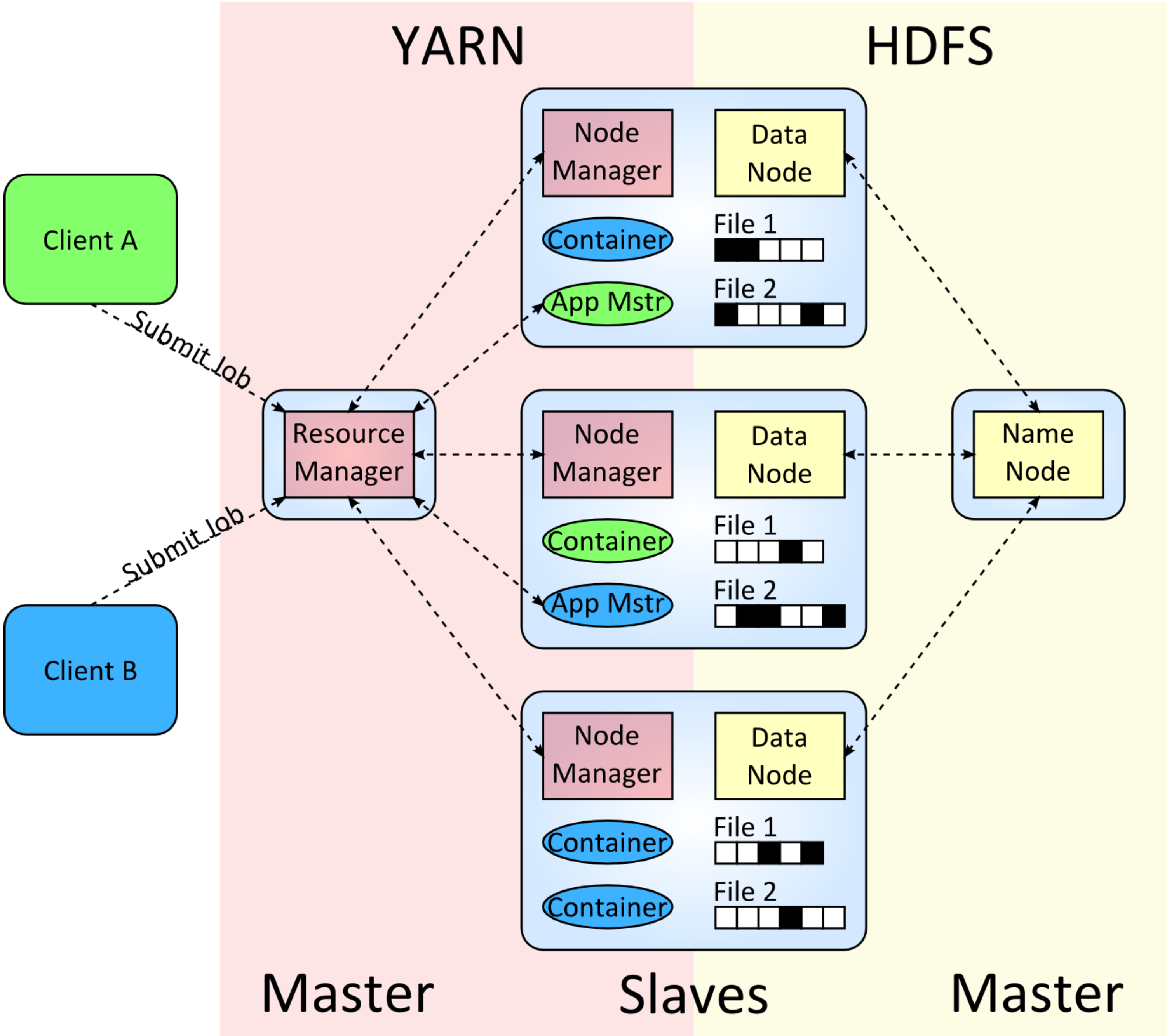
Single Processing Framework
Batch Apps
Multiple Processing Frameworks
Batch, Interactive, Streaming …
Hadoop 1.x
MapReduce
(resource mgmt, job scheduler, data processing)
HDFS
(distributed storage)
Hadoop 2.x
MapReduce
(data processing)
Others
(spark, mpi, giraph, etc)
YARN
(resource mgmt, job scheduler)
HDFS
(distributed storage)

MapReduce and MPI as YARN Applications



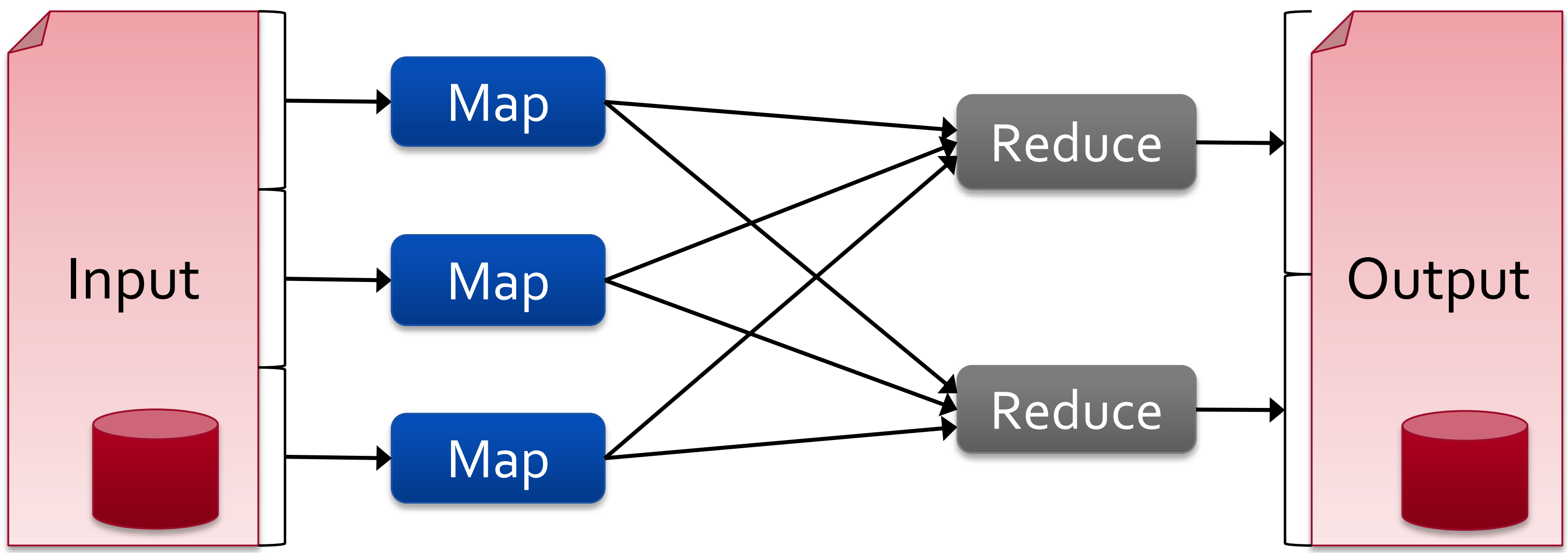
[Murthy et. al, Apache Hadoop YARN: Yet Another Resource Negotiator", SOCC'13]

Data Locality in Hadoop v2



Limitations of MapReduce [Zaharia'11]

- MapReduce is based on an *acyclic data flow* from stable storage to stable storage.
  - Slow writes data to HDFS at every stage in the pipeline
- Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (machine learning, graphs)
  - **Interactive** data mining tools (R, Excel, Python)



Iterative Data Processing Frameworks

```
val input= TextFile(textInput)

val words = input
    .flatMap
      { line => line.split(" ") }

val counts = words
    .groupBy
        { word => word }
    .count()

val output = counts
    .write (wordsOutput,
        RecordDataSinkFormat() )

val plan = new ScalaPlan(Seq(output))
```

Spark
Lightning-Fast Cluster Computing

Flink

Spark – Resiliant Distributed Datasets

- Allow apps to keep working sets in memory for efficient reuse
- Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability

Resilient distributed datasets (RDDs)

- Immutable, partitioned collections of objects
- Created through parallel *transformations* (map, filter, groupBy, join, …) on data in stable storage
- Can be *cached* for efficient reuse

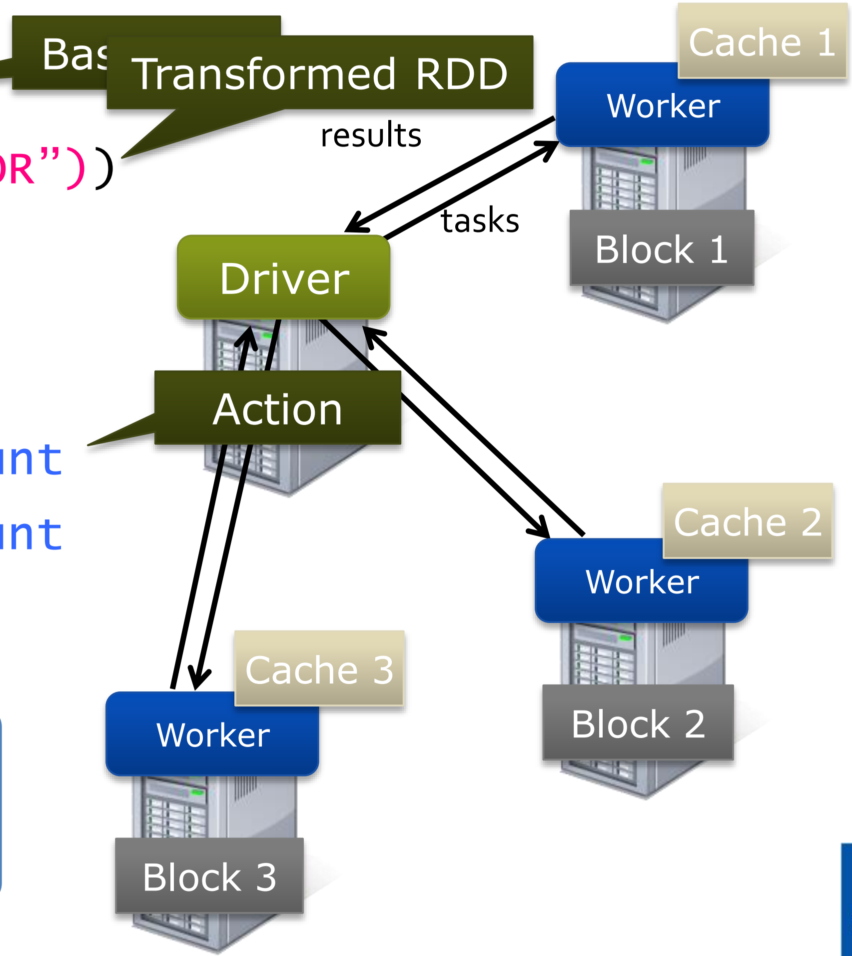
Actions on RDDs

- Count, reduce, collect, save, …

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



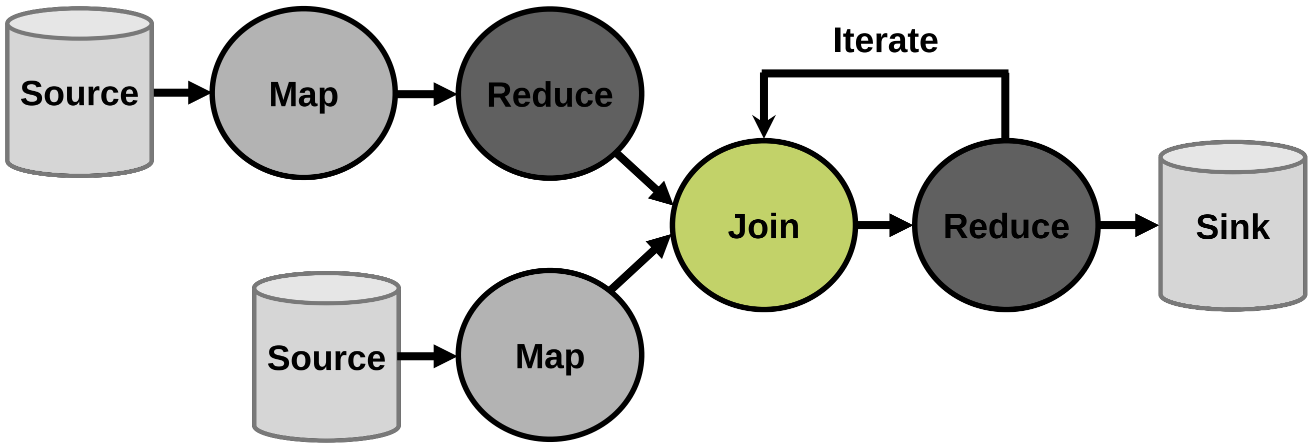
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

ROYAL INSTITUTE OF TECHNOLOGY

Apache Flink – DataFlow Operators

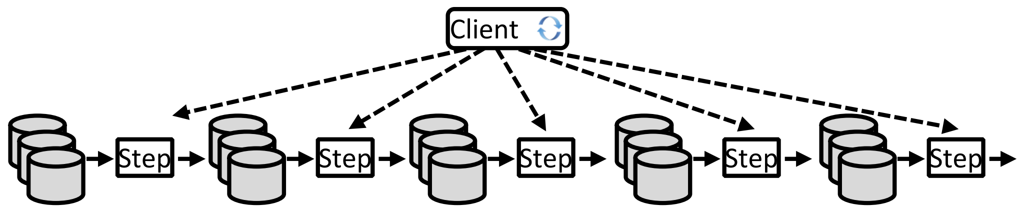
Flink

Map	Iterate	Project
Reduce	Delta Iterate	Aggregate
Join	Filter	Distinct
CoGroup	FlatMap	Vertex Update
Union	GroupReduce	Accumulators

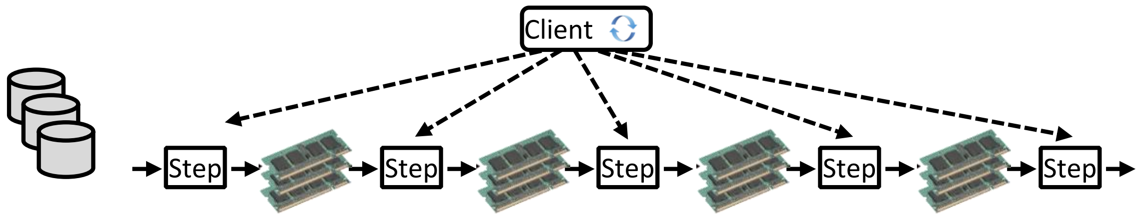


*Alexandrov et al.: "The Stratosphere Platform for Big Data Analytics," VLDB Journal 5/2014

Built-in vs. driver-based looping

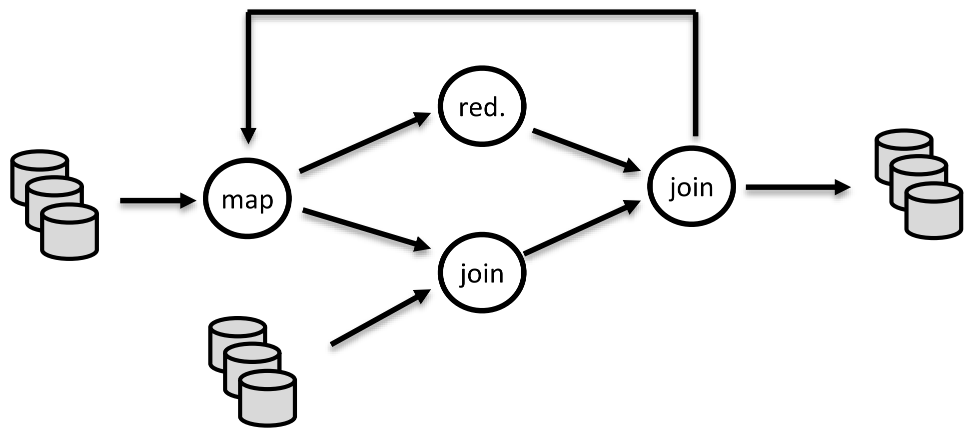


Loop outside the system, in driver program



Iterative program looks like many independent jobs

Flink

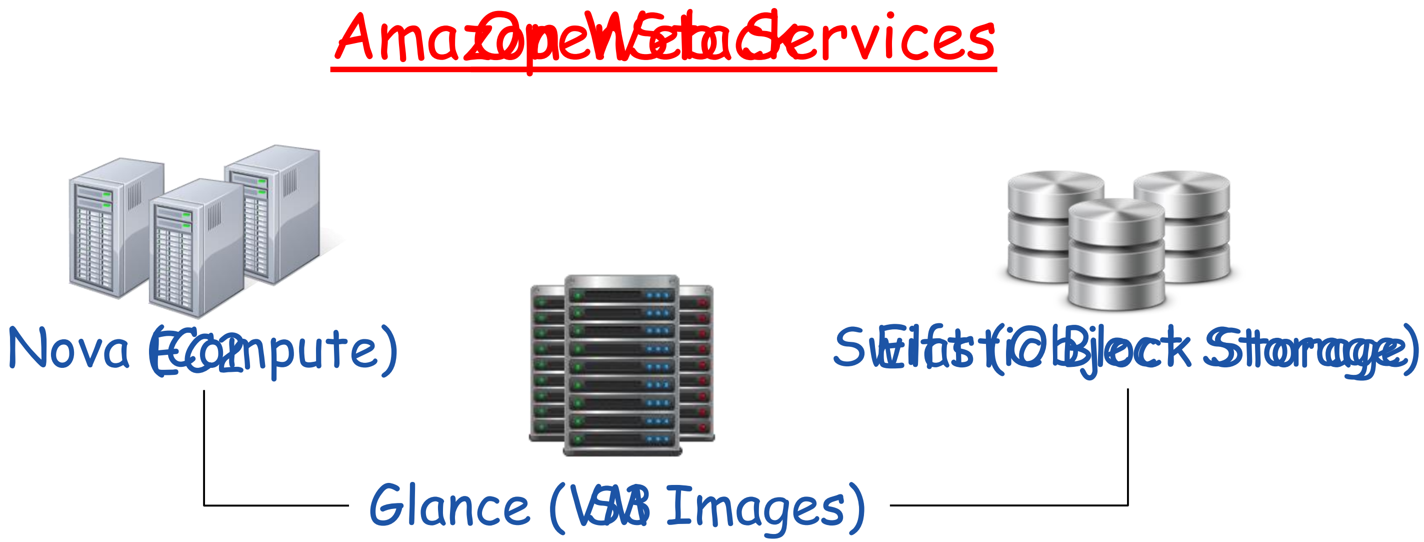


Dataflows with feedback edges

System is iteration-aware, can optimize the job

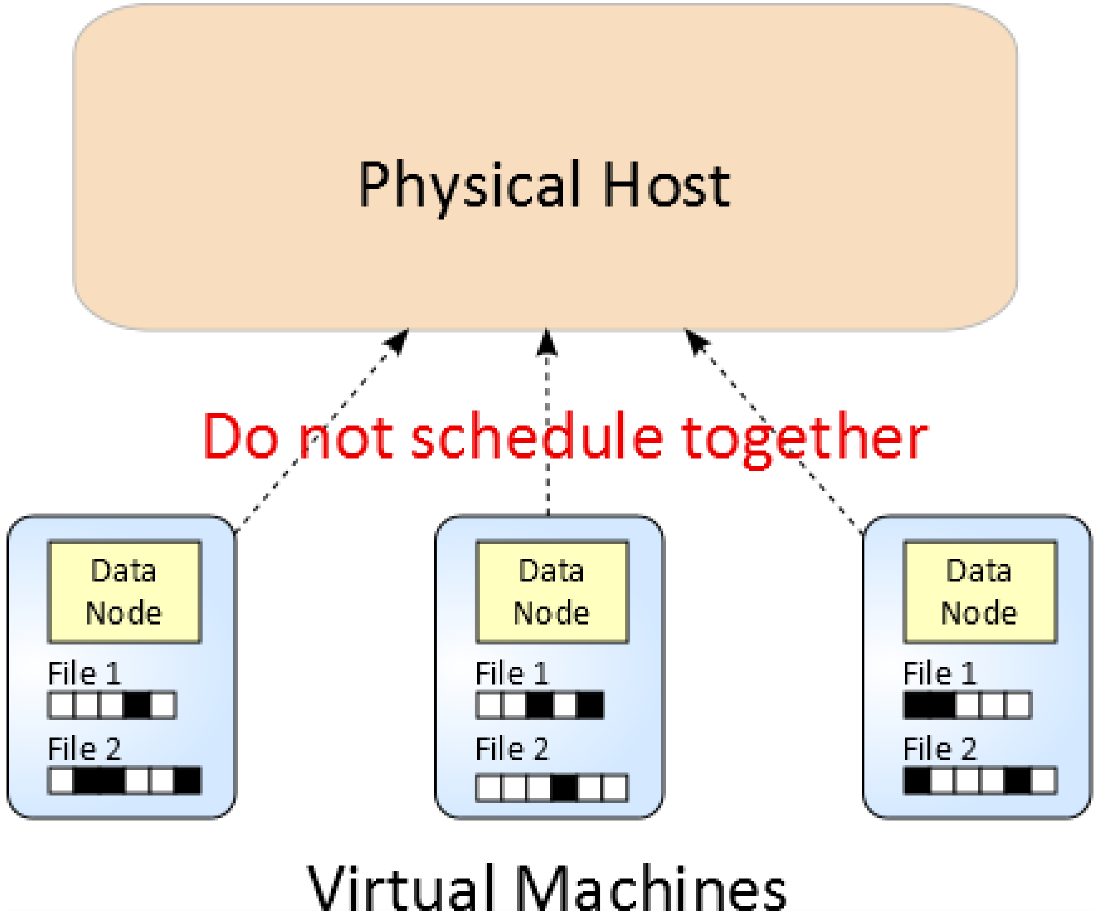
Hadoop on the Cloud

- Cloud Computing traditionally separates storage and computation.



Data Locality for Hadoop on the Cloud

- Cloud hardware configurations should support data locality
- Hadoop's original topology awareness breaks
  - Placement of >1 VM containing block replicas for the same file on the same physical host increases correlated failures
- VMWare introduced a NodeGroup aware topology
  - HADOOP-8468



Conclusions

- Hadoop is the open-source enabling technology for Big Data

- YARN is rapidly becoming the operating system for the Data Center

- Apache Spark and Flink are in-memory processing frameworks for Hadoop

References

- Dean et. Al, “MapReduce: Simplified Data Processing on Large Clusters”, OSDI’04.
- Schvachko, “HDFS Scalability: The limits to growth”, Usenix, :login, April 2010.
- Murthy et al, “Apache Hadoop YARN: Yet Another Resource Negotiator”, SOCC’13.
- “Processing a Trillion Cells per Mouse Click”, VLDB’12