CS422
Database systems

Today: Execution models for distributed computing – 1st generation

Data-Intensive Applications and Systems (DIAS) Laboratory
École Polytechnique Fédérale de Lausanne

Previous weeks

- Storage
- Query operators and optimization
- OLAP

<u>Common challenge</u>

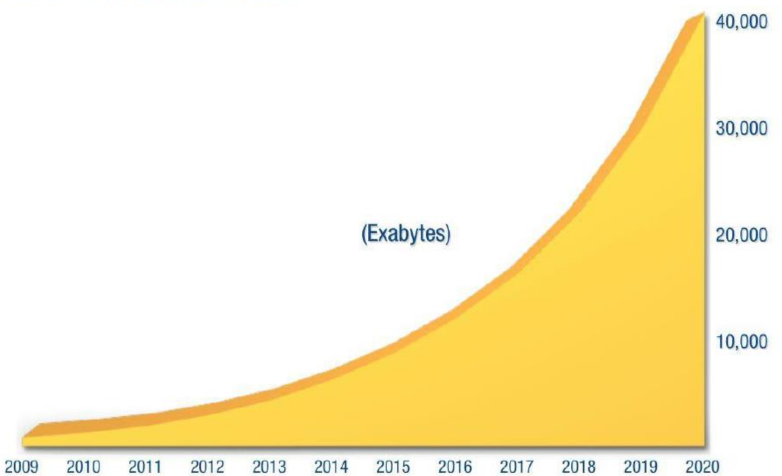
how to handle the ever-increasing data sizes!

Overview

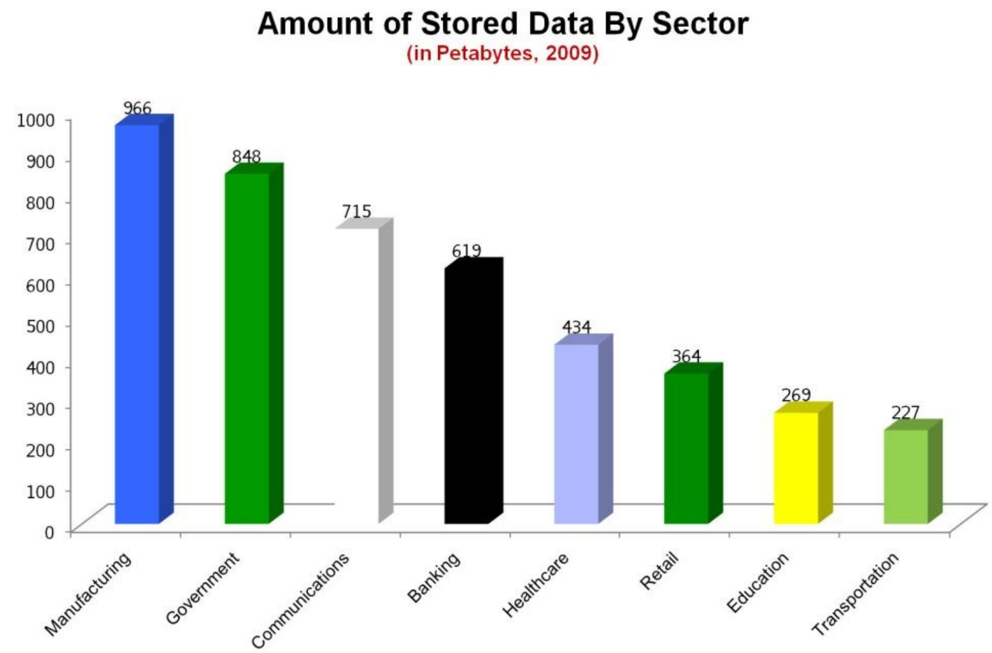
- Big Data
- Introduction to Hadoop & MapReduce
- Architectural choices
- The other side

How big is "all" data?

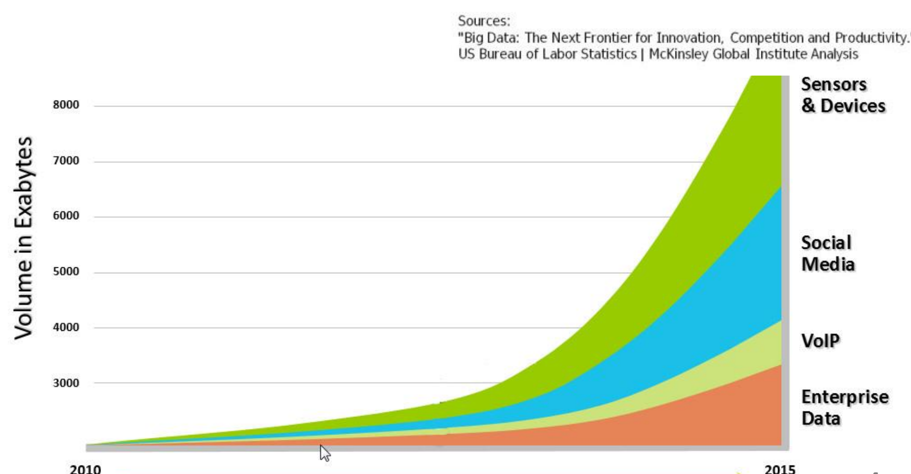
The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020



This IDC graph predicts exponential growth of data from around 3 zettabytes in 2013 to approximately 40 zettabytes by 2020. An exabyte equals 1,000,000,000,000,000,000 bytes and 1,000 exabytes equals one zettabyte. Source: IDC's Digital Universe Study, December 2012, http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.



Decimal		
Value	Metric	
1000	kB	kilobyte
1000^2	MB	megabyte
1000^3	GB	gigabyte
1000^4	TB	terabyte
1000^5	PB	petabyte
1000^6	EB	**exabyte**
1000^7	ZB	zettabyte
1000^8	YB	yottabyte



http://www.tech-dynamics.com/wp-content/uploads/2014/02/BigDataChart.png

Big Data

The three (plus two) Vs: Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization

- Volume: The quantity of generated and stored data.
- Velocity: The speed at which the data is generated and processed.
- Variety: The type and nature of the data.
- Variability: Inconsistency of the data set.
- Veracity: The quality of captured data.

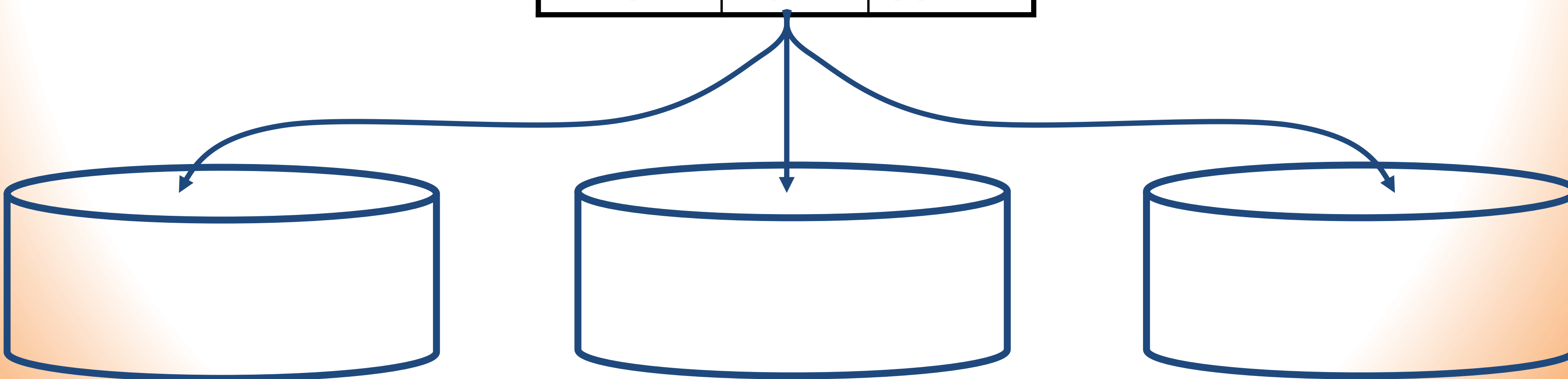
The Gamma Parallel Database Machine (DeWitt, UW Madison)

- Motivation
- Physical System Designs
- **Data Clustering**
- Failure Management
- Query Processing
- Evaluation and Results
- Bonus : Current Progress, Hadoop, Clustera.

No Shared Data ⇒ Declustering

name	age	salary
Bob	20	10K
Shideh	18	35K
Ted	50	60K
Kevin	62	120K
Angela	55	140K
Mike	45	90K

Logical View



Physical View

No Shared Data → Declustering

- Spreading data between disks :
  - Attribute-less partitioning
    - Random
    - Round-Robin
  - Single Attribute Schemes
    - Hash De-clustering
    - Range De-clustering
  - Multiple Attributes schemes possible
    - MAGIC, BERD etc

No Shared Data → Declustering

- Spreading data between disks :
  - Attribute-less partitioning
    - Random
    - Round-Robin
  - **Single Attribute Schemes**
    - **Hash De-clustering**
    - **Range De-clustering**
  - Multiple Attributes schemes possible
    - MAGIC, BERD etc

Hash Declustering

salary is the partitioning attribute.

name	age	salary
Bob	20	10K
Shideh	18	35K
Ted	50	60K
Kevin	62	120K
Angela	55	140K
Mike	45	90K

salary % 3

name	age	salary
Ted	50	60K
Kevin	62	120K

name	age	salary
Bob	20	10K
Mike	45	90K

name	age	salary
Shideh	18	35K
Angela	55	140K

Hash Declustering

- Selections with equality predicates referencing the partitioning attribute are directed to a single node:
  - Retrieve Emp where salary = 60K

```
SELECT *
FROM   Emp
WHERE salary=60K
```

- Equality predicates referencing a non-partitioning attribute and range predicates are directed to all nodes:
  - Retrieve Emp where age = 20
  - Retrieve Emp where salary < 20K

```
SELECT *
FROM   Emp
WHERE salary<20K
```

Range Declustering

salary is the partitioning attribute.

name	age	salary
Bob	20	10K
Shideh	18	35K
Ted	50	60K
Kevin	62	120K
Angela	55	140K
Mike	45	90K

0-50K

name	age	salary
Bob	20	10K
Shideh	18	35K

51K-100K

name	age	salary
Ted	50	60K
Mike	45	90K

101K-∞

name	age	salary
Kevin	62	120K
Angela	55	140K

Range Declustering

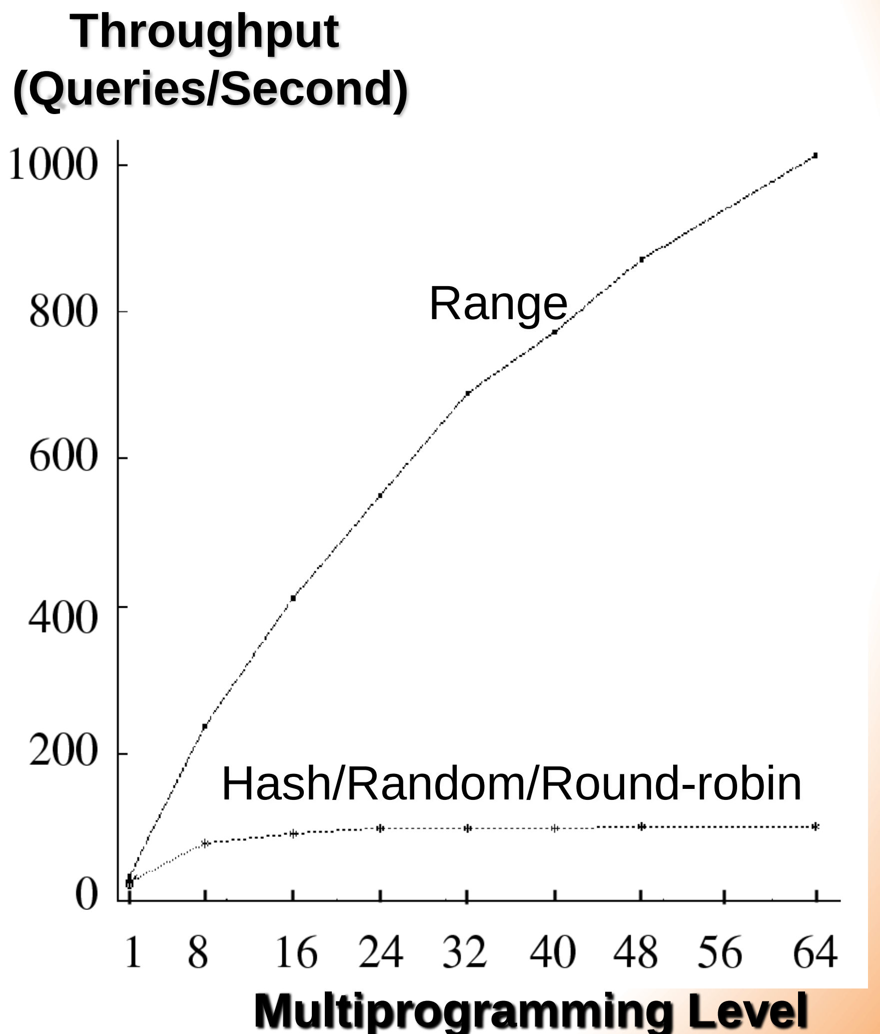
- Equality and range predicates referencing the partitioning attribute are directed to a subset of nodes:
  - Retrieve Emp where salary = 60K
  - Retrieve Emp where salary < 20K

  In the example, both queries are directed to one node.

- Predicates referencing a non-partitioning attribute are directed to all nodes.

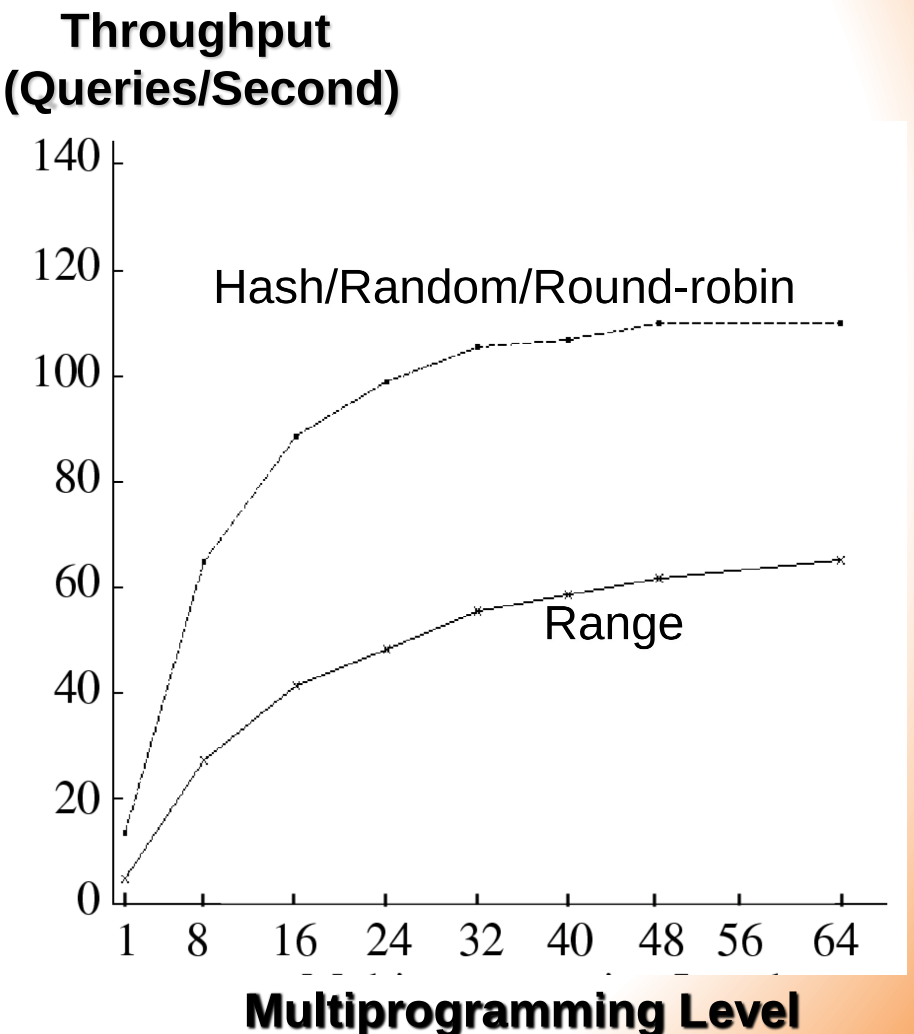
Declustering Tradeoffs

- Range selection predicate using a clustered B+-tree
- 0.01% selectivity (10 records)



Declustering Tradeoffs

- Range selection predicate using a clustered B+-tree
- 1% selectivity (1000 records)



Declustering Tradeoffs

- Why the difference ?

Declustering Tradeoffs

- Why the difference ?

  - When selection was small, Range spread out load and was ideal

  - When selection increased, Range caused high workload on one/few nodes while Hash spread out load

The Gamma Parallel Database Machine (DeWitt, UW Madison)

- Motivation
- Physical System Designs
- Data Clustering
- **Failure Management**
- Query Processing
- Evaluation and Results
- Bonus : Current Progress, Hadoop, Clustera..

Failure Management

- Key Questions
  - Robustness (How much damage recoverable?)
  - Availability (How likely? Hot Recoverable?)
  - MTTR (Mean Time To Recovery)
- Consider two declustering schemes
  - Interleaved Declustering (TeraData)
  - Chained Declustering (Gamma DBM)

Interleaved Declustering

- A partitioned table has a primary and a backup copy.
- The primary copy is constructed using one of the partitioning techniques.
- The secondary copy is constructed by:
  - Dividing the nodes into clusters (cluster size 4 here),
  - Partition a primary fragment (R0) across the remaining nodes of the cluster: 1, 2, and 3. Realizing r0.0, r0.1, and r0.2.

	cluster 0				cluster 1			
Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy		r0.0	r0.1	r0.2		r4.0	r4.1	r4.2
	r1.2		r1.0	r1.1	r5.2		r5.0	r5.1
	r2.1	r2.2		r2.0	r6.1	r6.2		r6.0
	r3.0	r3.1	r3.2		r7.0	r7.1	r7.2	

Interleaved Declustering (Cluster Size = 4)

Interleaved Declustering

- On failure, query load re-directed to backup nodes in cluster
- MTTR :
  - replace node
  - reconstruct failed primary from backups
  - reconstruct backups stored in failed node

- Second failure before this can cause unavailability
- Large cluster size improves failure load balancing but increases risk of data being unavailable

	cluster 0				cluster 1			
Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy		r0.0	r0.1	r0.2		r4.0	r4.1	r4.2
	r1.2		r1.0	r1.1	r5.2		r5.0	r5.1
	r2.1	r2.2		r2.0	r6.1	r6.2		r6.0
	r3.0	r3.1	r3.2		r7.0	r7.1	r7.2	

Interleaved Declustering (Cluster Size = 4)

Chained Declustering (Gamma)

- Nodes are divided into disjoint groups called relation clusters.
- A relation is assigned to one relation cluster and its records are declustered across the nodes of that relation cluster using a partitioning strategy (Range, Hash).
- Given a primary fragment Ri, its backup copy is assigned to node (i+1) mod M (M is the number of nodes in the relation cluster).

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

Chained Declustering (Relation Cluster Size = 8)

Chained Declustering (Gamma)

- During normal operation:
  - Read requests are directed to the fragments of primary copy,
  - Write requests update both primary and backup copies.

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

Chained Declustering (Relation Cluster Size = 8)

Chained Declustering (Gamma)

- **In presence of failure:**
  - Both primary and backup fragments are used for read operations,
    - Objective: Balance the load and avoid bottlenecks!
  - Write requests update both primary and backup copies.

- Note:
  - Load of R1 (on node 1) is pushed to node 2 in its entirety.
  - A fraction of read request from each node is pushed to the others for a 1/8 load increase attributed to node 1's failure.

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	~~R1~~	R2	R3	R4	R5	R6	R7
Backup Copy	r7	~~r0~~	r1	r2	r3	r4	r5	r6

Chained Declustering (Relation Cluster Size = 8)

Chained Declustering (Gamma)

- MTTR involves:
  - Replace node 1 with a new node,
  - Reconstruct R1 (from r1 on node 2) on node 1,
  - Reconstruct backup copy of R0 (i.e., r0) on node 1.
- Note:
  - Once Node 1 becomes operational, primary copies are used to process read requests.

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	~~R1~~	R2	R3	R4	R5	R6	R7
Backup Copy	r7	~~r0~~	r1	r2	r3	r4	r5	r6

Chained Declustering (Relation Cluster Size = 8)

Chained Declustering (Gamma)

- Any two node failures in a relation cluster does not result in data un-availability.

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	~~R1~~	R2	R3	~~R4~~	R5	R6	R7
Backup Copy	r7	~~r0~~	r1	r2	~~r3~~	r4	r5	r6

Chained Declustering (Relation Cluster Size = 8)

- Two adjacent nodes must fail in order for data to become unavailable

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	~~R6~~	~~R7~~
Backup Copy	r7	r0	r1	r2	r3	r4	~~r5~~	~~r6~~

Chained Declustering (Relation Cluster Size = 8)

Big Data Landscape 2016 (Version 2.0)

Last Updated 2/12/2016

FIRSTMARK

Mainstream Big Data models

How to store, manage and process Big Data by harnessing large clusters of commodity nodes

- MapReduce family: simpler, more constrained

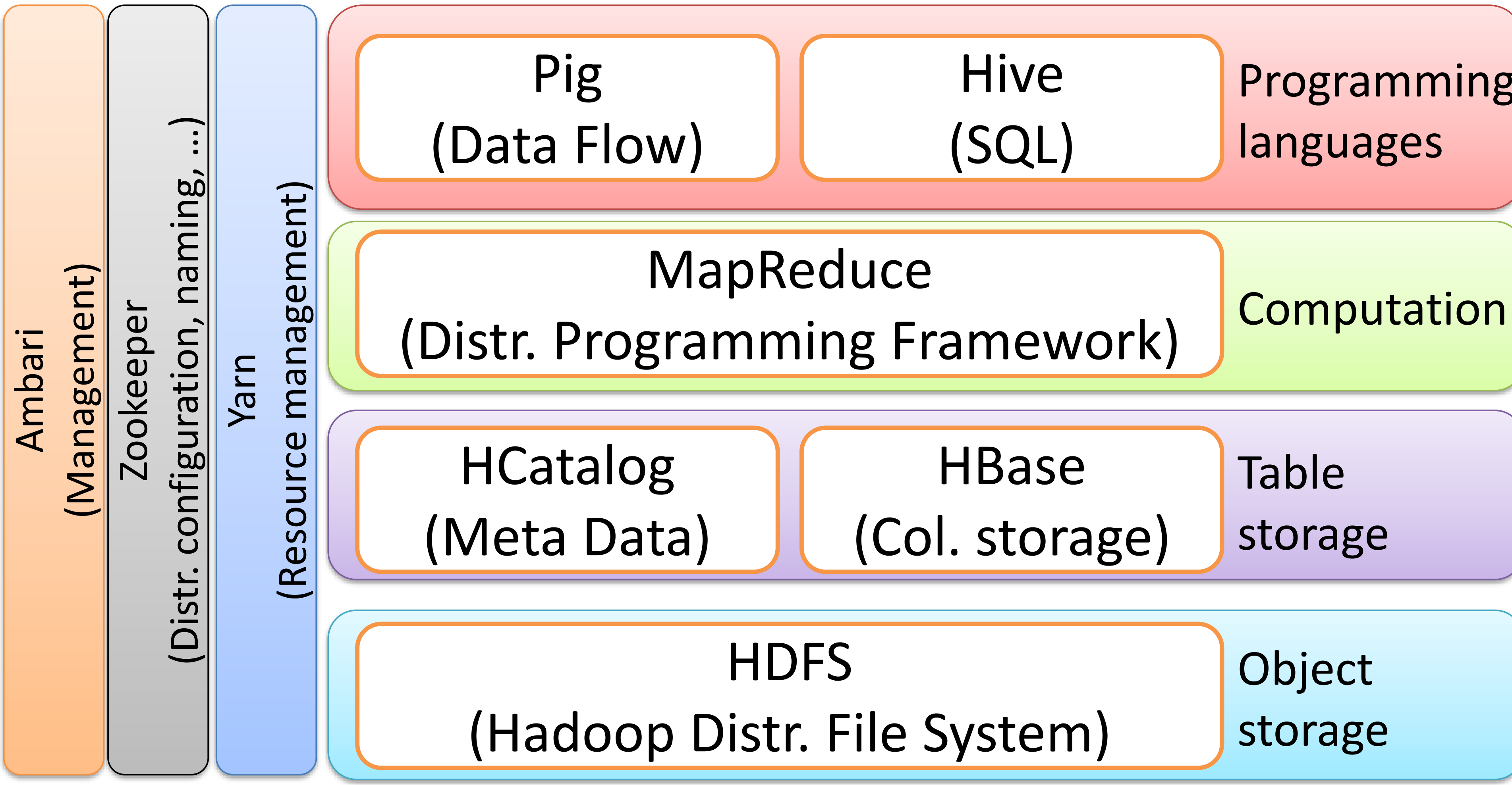
HadoopDB

- 2nd generation: enables more complex processing & data, optimization opportunities

Google Pregel

Microsoft Dryad

The Hadoop ecosystem



Architectural choices of Hadoop

- Storage layer
  - Hadoop HDFS
- Programming model & exec. engine
- Scheduling
- Optimizations
- Fault tolerance

Storage layer

Desiderata

- Scalability: Handle the ever-increasing data sizes
- Efficiency: Fast accesses to data
- Simplicity: Hide complexity from the developers
- Fault-tolerance: Failures do not lead to loss of data

HDFS abstractions

- HDFS provides the illusion of a centralized file system

```
$ hdfs dfs -cat hdfs://nn1.example.com/abc
$ hdfs dfs -copyFromLocal abc hdfs://nn1.example.com/abc
$ hdfs dfs -copyToLocal hdfs://nn1.example.com/abc abc
```

…

- Developers are **NOT** reading from/writing to files explicitly. HDFS handles IO transparently.
  - E.g., MapReduce operates on file splits/partitions, which are received as inputs of Map/Reduce

```
public void map(LongWritable key, Text value,
                                    Context con) {
      …
      con.write(outputKey,outputValue);
}
```

HDFS & MapReduce

- HDFS leveraged MapReduce execution engine for
  - Saving all input & intermediary files and final results
  - Robust communication across the cluster nodes
  - Replication in the context of fault tolerance & load balancing

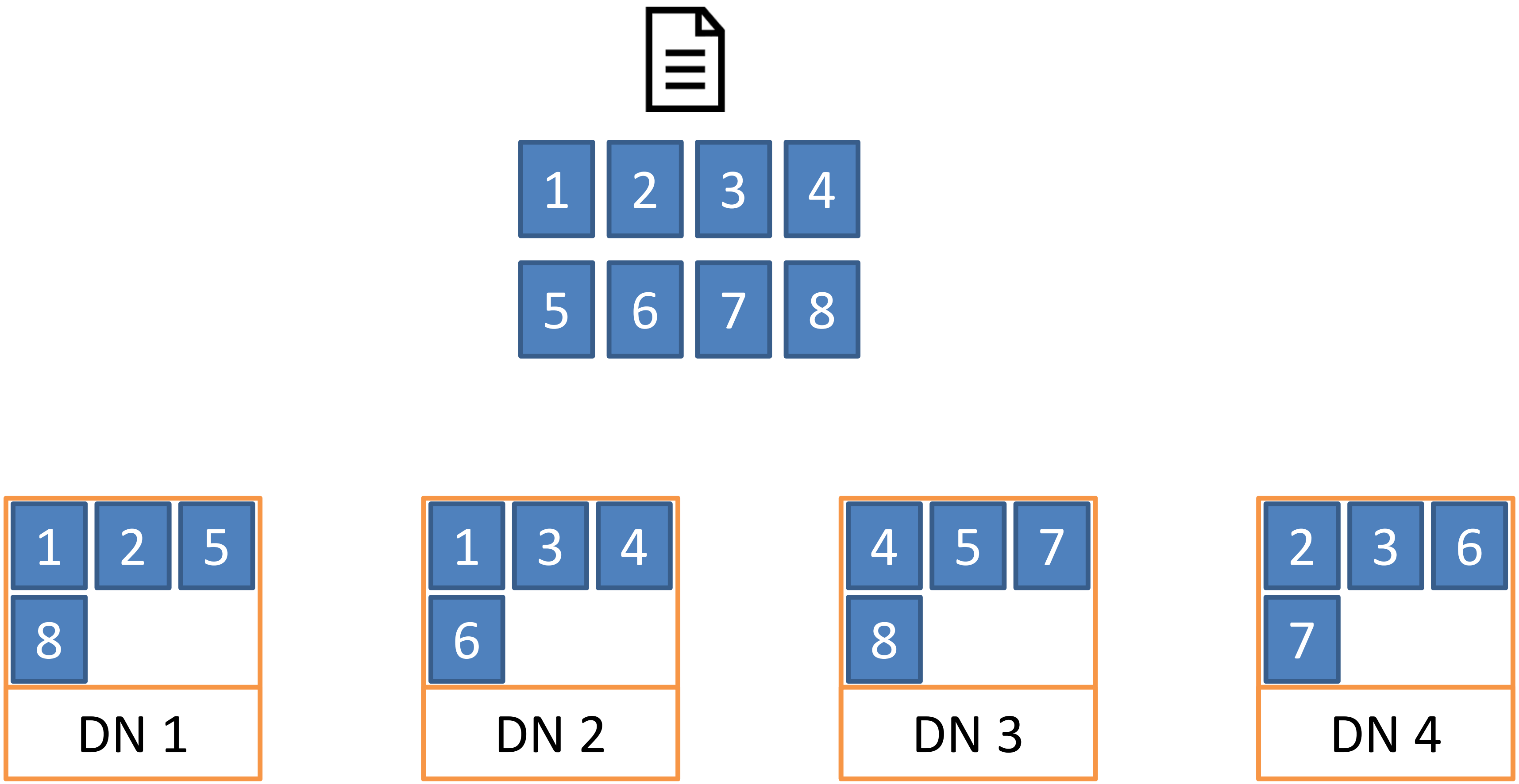
HDFS replication

- Files partitioned into blocks (typically 64 MB)
- Blocks distributed and replicated across nodes

- Three types of nodes in HDFS
  - Name nodes: Keep the locations of blocks
  - Secondary name nodes: backup nodes
  - Data nodes: keep the actual blocks

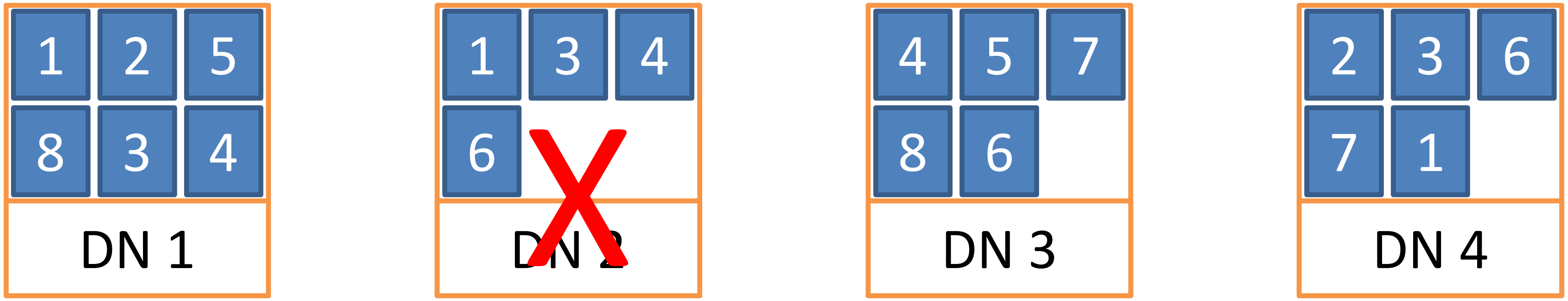
HDFS replication (2)

- Files partitioned into blocks (default: 64 Mb each)
- Blocks replicated across different nodes



HDFS failures

- Detect failed data nodes with heartbeats
  - On failure, name node removes the failed data nodes from the index
  - Lost partitions are re-replicated to the remaining data nodes



HDFS evaluation

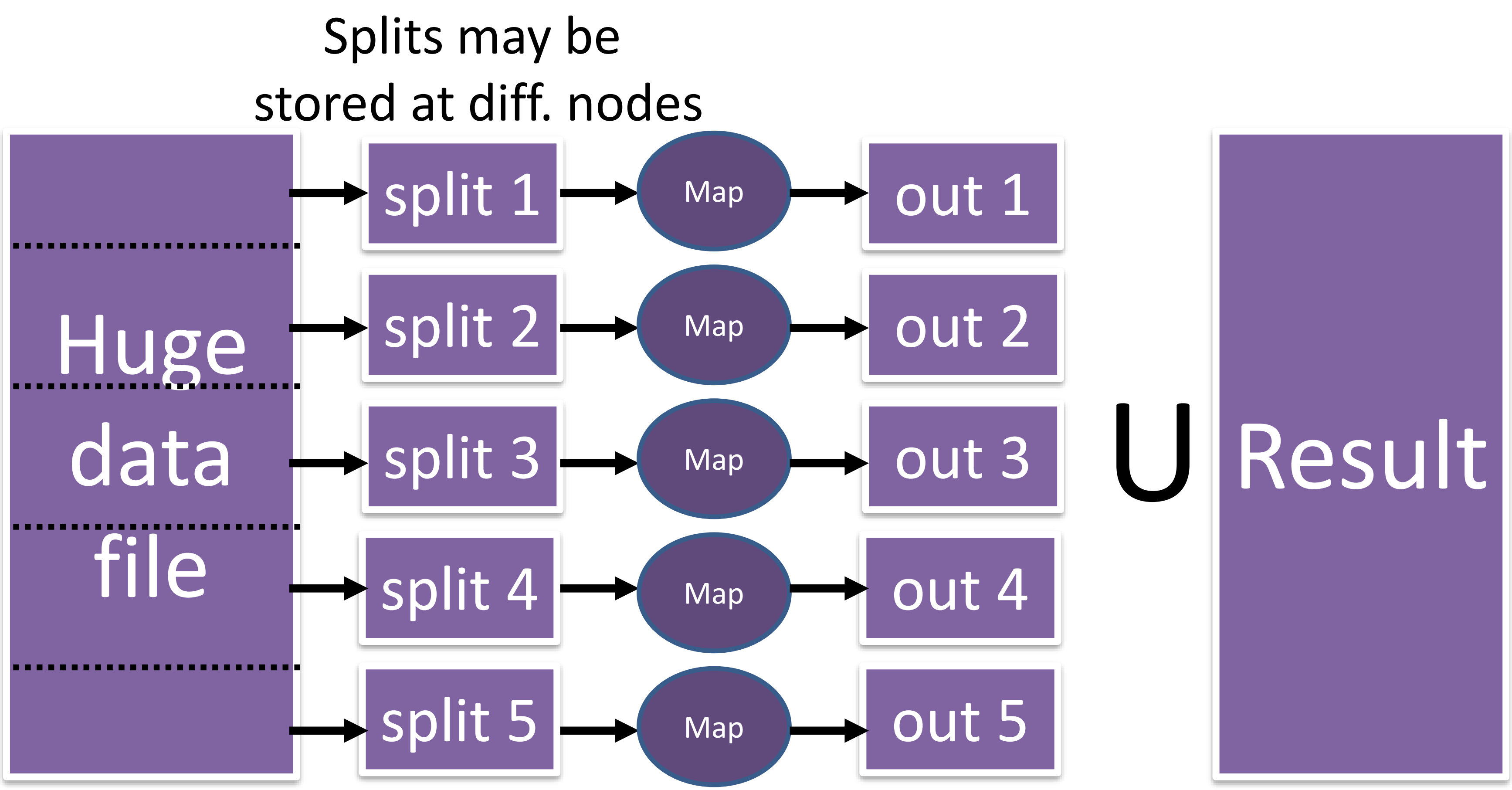
- Scalability: Handle the ever-increasing data sizes?
  - Just add more data nodes
- Efficiency: Fast accesses to data?
  - Everything read from HDD -- still requires I/O
- Simplicity: Hide complexity from the developers?
  - Developer does not need to know where each block is stored
- Fault-tolerance: Failures lead to loss of data?
  - Administrator can control replication
  - If failures are not widespread, no data is lost

Architectural choices of Hadoop

- Storage layer
- Programming model & exec. engine
  - The vision
  - MapReduce
- Scheduling
- Optimizations
- Fault tolerance

The vision…

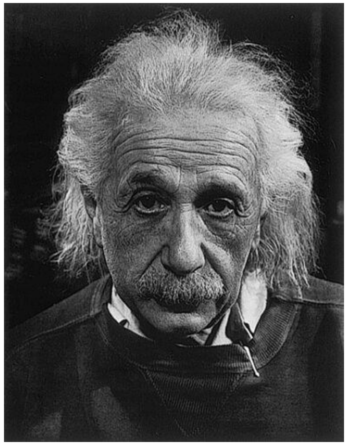
Sample function: convert all text to upper case



The vision (2)

More complicated: the word-count problem

- Huge file → extract frequencies of words
- Example



Einstein once said…

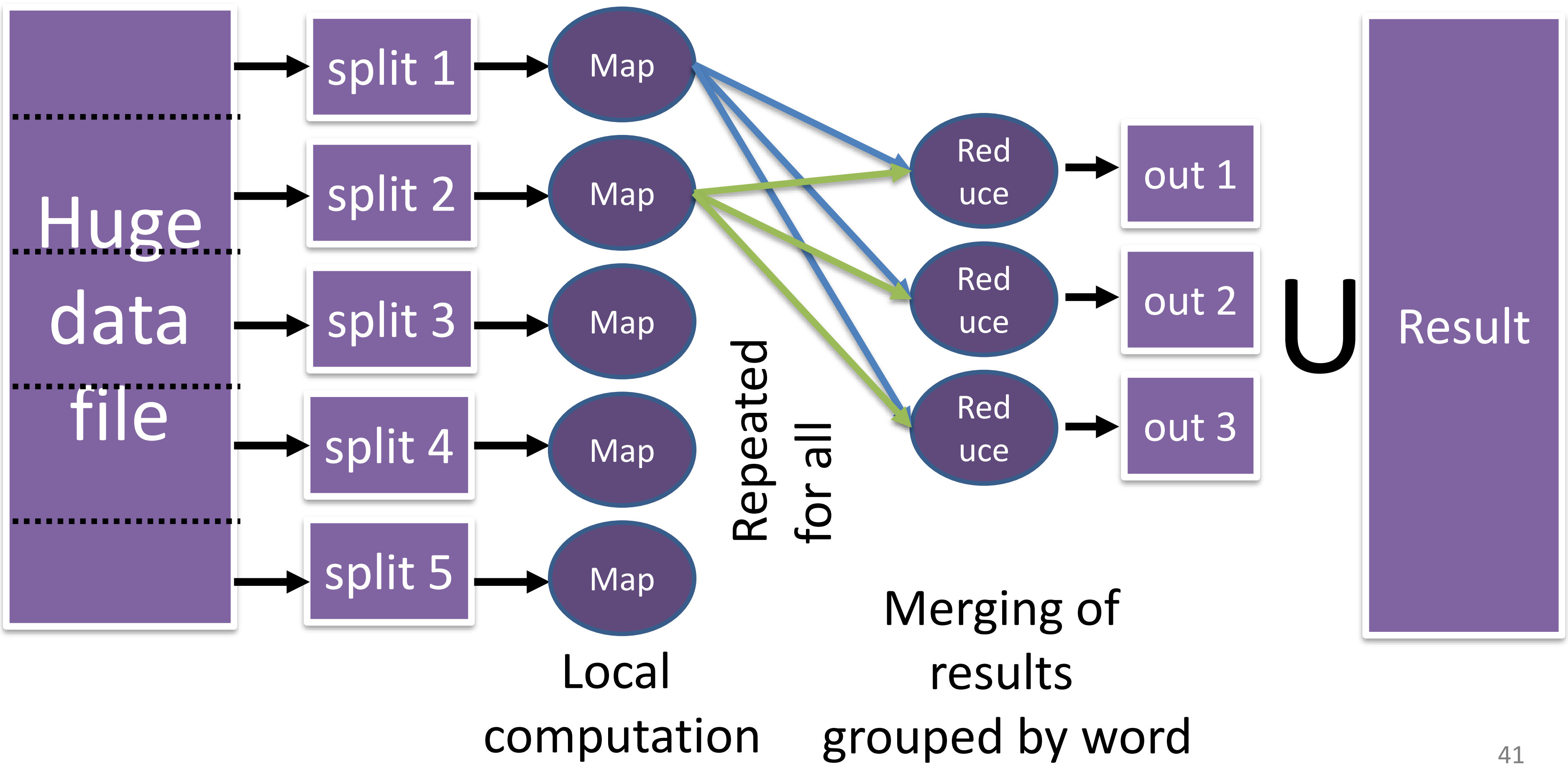
Logic will get you from A to B.
Imagination will take you everywhere.

Extracted frequencies:

- <Logic,1>, <will,2>, <get,1>, <you,2>, …

The vision (3)

Sample application: the word-count example



MapReduce programming model

- Data model: everything is a <key,value> pair
- Programming model - two core functions
  - Map(key,value): Invoked for every split of the input data. Value corresponds to the split.
  - Reduce(key,list(values)): Invoked for every unique key emitted by Map. List(values) corresponds to all values emitted from ALL mappers for this key.

→ parallelism and deployment handled by the system

MapReduce programming model (2)

- The word-count problem
  - Input: Text file, broken in splits
  - Output: Frequency of each word observed in the file
  - Map(key,value): value: a split of the text file

```
for each word in value
    emit pair <word,+1>
```

  - Reduce(key,list(values)): Key: word, values: list of (+1’s)

```
count=0
for each value in list(values)
    count+=value
emit pair<key,count>
```

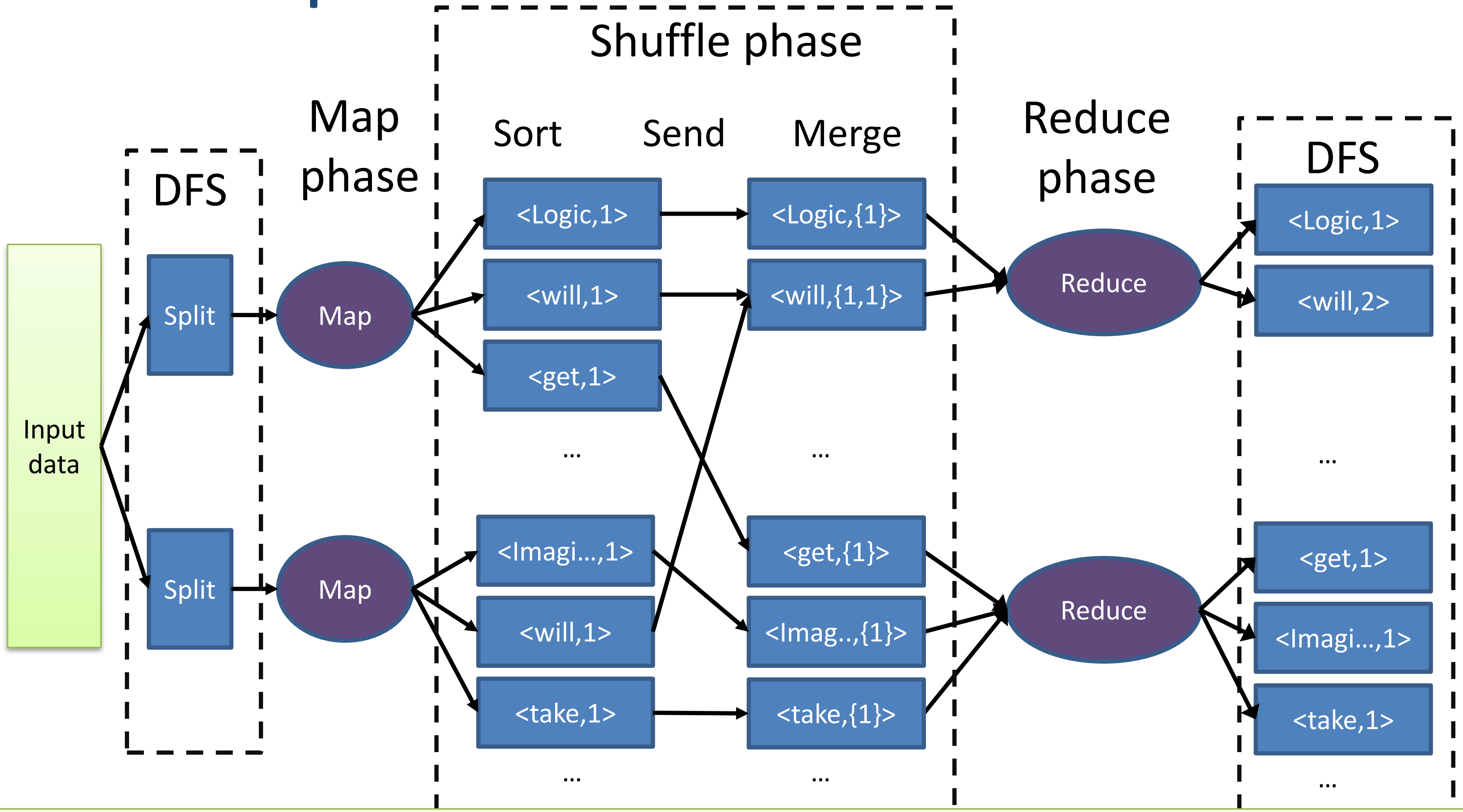
Sample code for word-count

```
private Text word = new Text();
public void map(Object key, Text value, Context context){
 StringTokenizer itr = new StringTokenizer(value.toString());
 while (itr.hasMoreTokens()) {
   word.set(itr.nextToken());
   context.write(word, new IntWritable(1));
 }
---
public void reduce(Text key, Iterable<IntWritable>values,
                   Context context) {
 int sum = 0;
 for (IntWritable val : values) {
   sum += val.get();
 }
 result.set(sum);
 context.write(key, result);
}
```

one item in the list

Only relevant code is shown here. Full example online

MapReduce – under the hood



Intermediary results are saved as regular files in HDFS. Is this good?

MapReduce – under the hood (2)

- Shuffling consumes network and causes delays
  - Idle time for CPUs
- Progressive shuffling
- Phases partially overlap: reducers start before maps fully complete

Is it possible that a reducer completes before all mappers/shuffling complete?

HDFS & MapReduce

- Master-slave architecture
  - Namenodes, JobTrackers, TaskTrackers

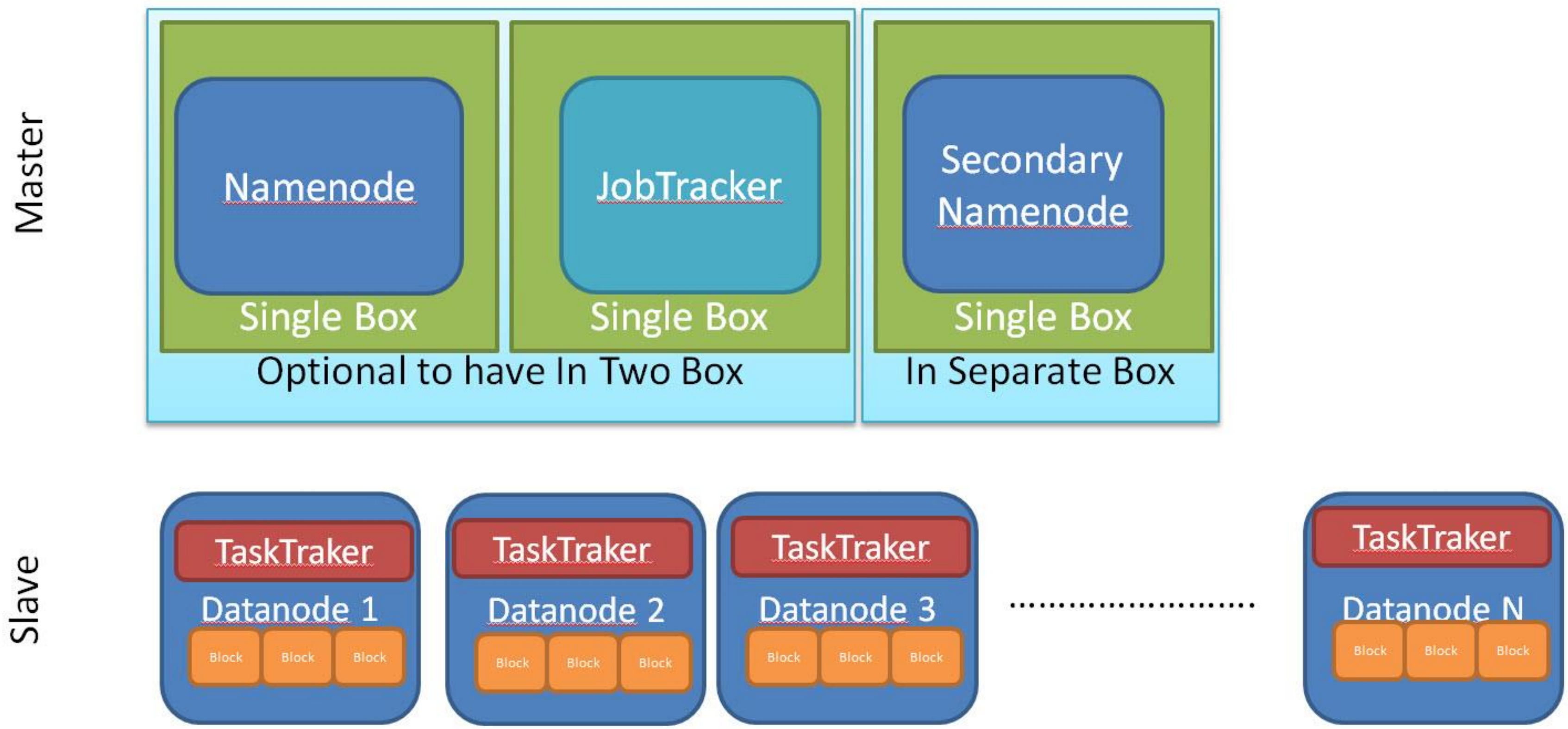


Figure from
https://blogs.msdn.microsoft.com/avkashchauhan/2012/02/24/master-slave-architecture-in-hadoop/

MapReduce is not a panacea

- MapReduce simple but weak for some reqs.
  - Cannot define complex processes
  - Batch mode, acyclic, not iterative
  - Everything file-based, no distributed memory
  - Procedural → difficult to optimize

- Examples
  - Iterative processes, e.g., clustering
  - Real-time answers, e.g., streams
  - Graph queries, e.g., shortest path


OTHERSIDE
RED HOT CHILI PEPPERS

Architectural choices of Hadoop

- Storage layer
- Programming model & exec. engine
- Scheduling
  - Data locality
  - Different schedulers
- Optimizations
- Fault tolerance

Scheduling

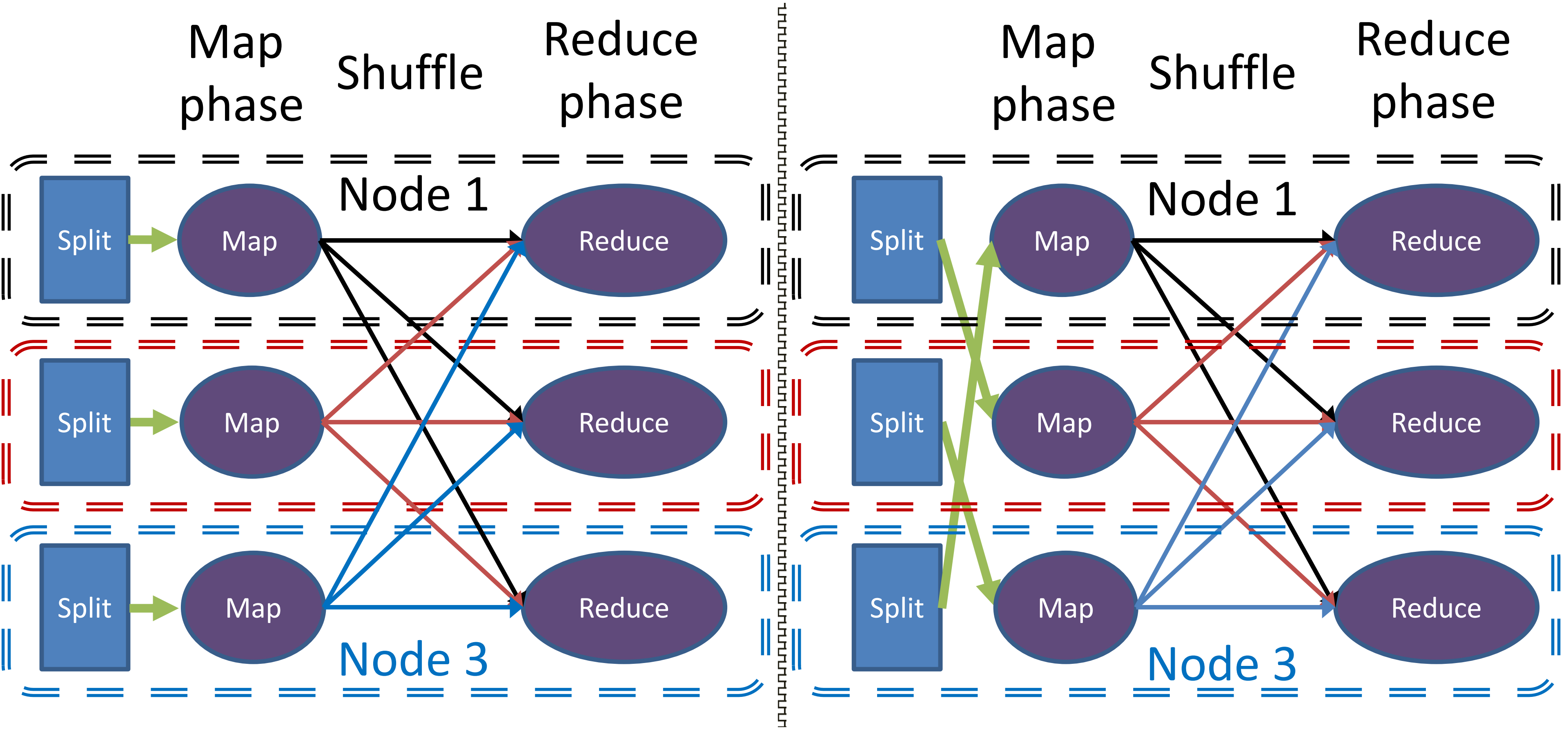
Decisions to be taken

- When to execute tasks
- Where to execute tasks
- How many resources to devote on each task
- Multi-tenant platforms: how to share resources

Primary goal

- Maximize data locality: move code to the data
- Three options for shipping data
  - (a) Node-local, (b) Rack-local, (c) Different rack

Importance of data locality



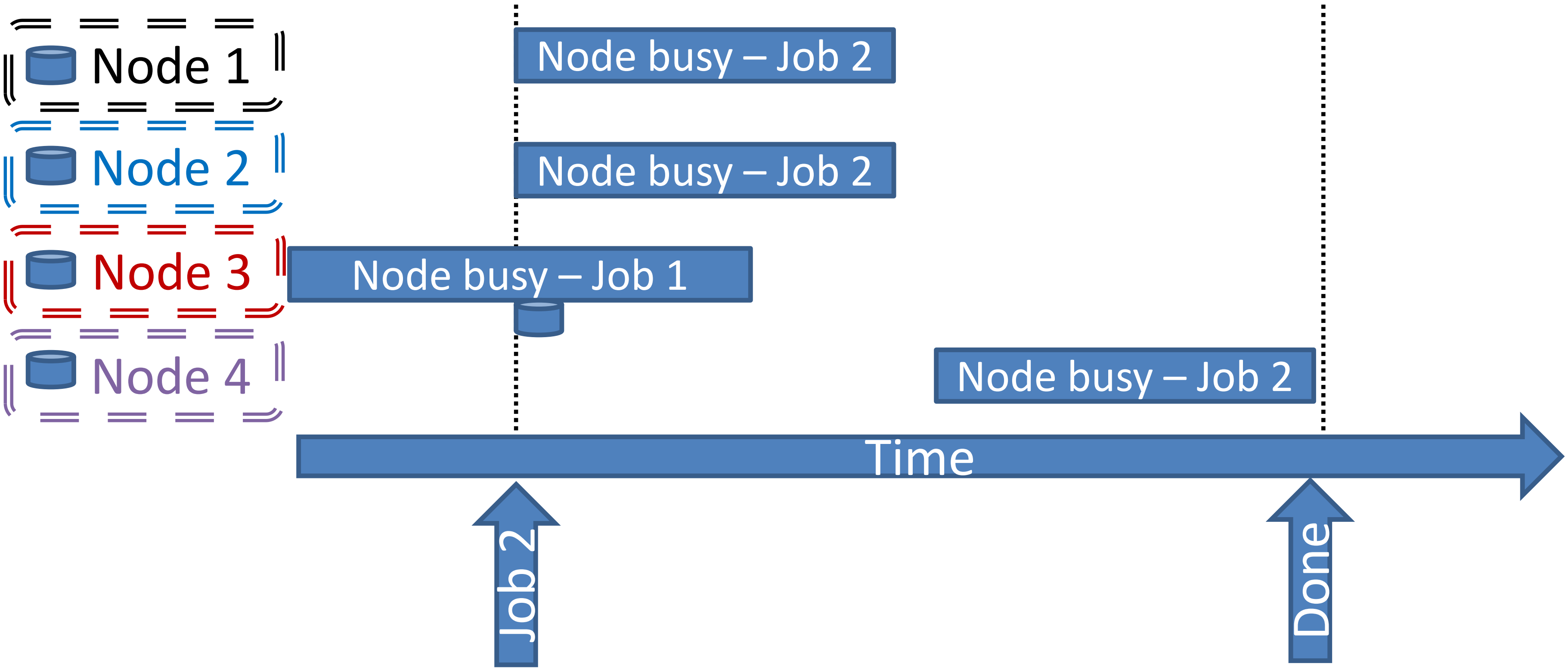
- Double-lined boxes denote node boundaries
  - Data shipping cost usually dominates

Scheduling (2)

- Delay scheduler: further promotes **data locality**
  - Wait for some time until you decide to move data
    - to another node in the same rack
    - to a node in a different rack

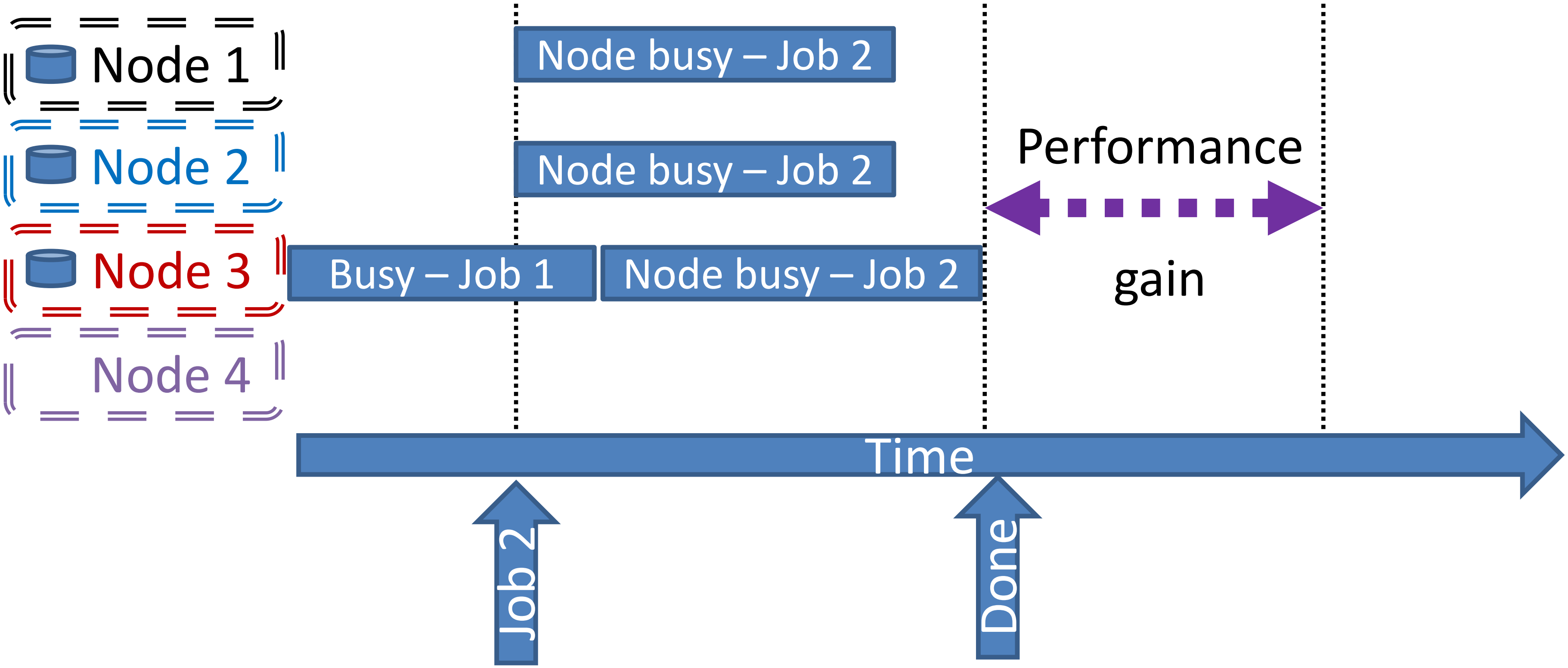
```
If wait < T1, only allow node-local tasks
If T1 < wait < T2, also allow rack-local
If wait > T2, also allow different rack
```

Without delay scheduling – map only



- Move/copy data when node is busy
  - Copying delays **both** job 1 and job 2
  - Network usage, Node 4 also occupied

With delay scheduling – map only



- Wait for a bit until node 3 is freed
  - Job 1 completes sooner, job 2 starts/completes sooner
  - No data moved for map phase
- Challenge: how long to wait

Scheduling (3)

Schedulers for multi-tenant platforms

- FIFO scheduler – naïve first approach
  - Hadoop early scheduler & Spark’s standalone scheduler
  - Jobs are served First-in, first-out
  - Each job exploits all available nodes
- Fair scheduler: fairness on resource usage
  - All jobs get ~ the same time
- Capacity scheduler: fairness among users
  - All users get ~ the same time

Architectural choices of Hadoop

- Storage layer
- Programming model & exec. engine
- Scheduling
- Optimizations
  - Programming model
  - Execution engine
- Fault tolerance

Programming model optimizations

- HadoopToSQL: Static code analysis to identify declarative constructs
  → more optimization opportunities!

```
function map(LogEntry)
   emit pair <LogEntry.Country,+1>

function reduce(Country,list(values))
      count=0
      for each value in list(values)
            count+=value
      emit pair<country,count>
```

```
SELECT Country, Count(*) FROM LogEntry GROUP BY Country
```

Programming model optimizations (2)

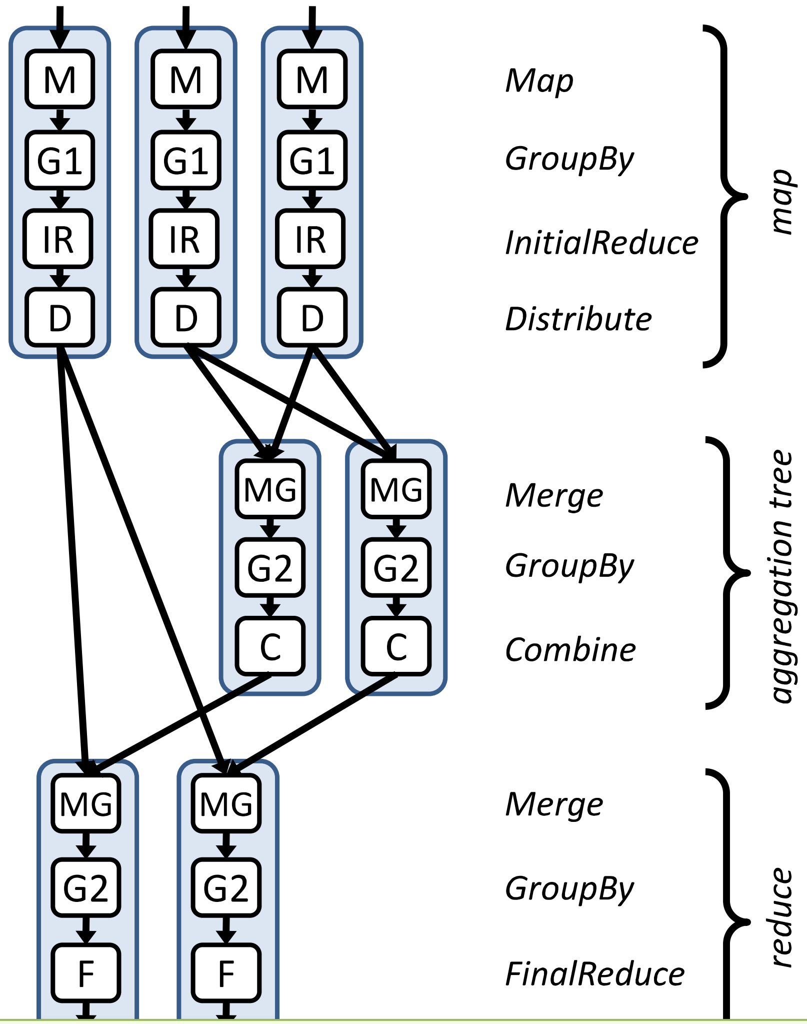
- DryadLINQ: Annotations to discover decomposable UDFs → eager aggregation, less network, less CPU
  - Associative-decomposable function: Function H can be expressed as a composition of two functions I and C, such that:

    $\oplus$: concatenation
    `x1,x2` : subsets

    - `H(x1`$\oplus$`x2) =C(I(x1`$\oplus$`x2)) =C(I(x1)`$\oplus$`I(x2))`
    - I is commutative: `I(x1` $\oplus$ `x2) = I(x2` $\oplus$ `x1)`
    - C is commutative & associative
      `C(x1` $\oplus$ `x2) = C(x2` $\oplus$ `x1)` &
      `C(x1` $\oplus$ `C(x2` $\oplus$ `x3)) = C(C(x1` $\oplus$ `x2)`$\oplus$`x3))`

Decomposable UDFs in DryadLINQ

- Associative-decomposable functions
  - initial reduction at each node
  - combine
  - final reduction



- Eager aggregation
- Less network

Can you rewrite the MR code of slide 38 such that it becomes decomposable?

Execution engine optimizations

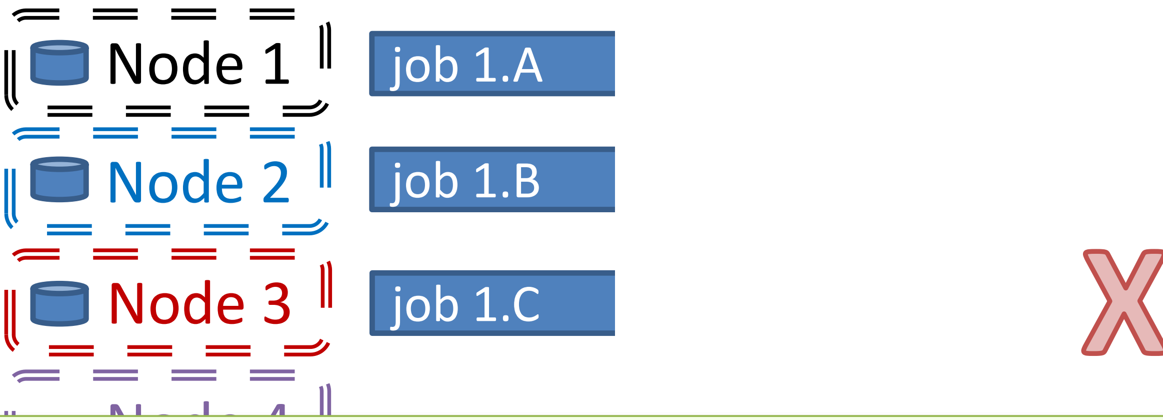
- Data locality
  - Data shipping cost typically dominates
  - Move computation to data

- Run-time/dynamic optimizations
  - Dynamically change number of mappers/reducers
    - Increase mappers at beginning of job
    - Increase reducers at end of the job
  - Load balancing – carefully choose replica
  - Dryad: Change layout to enable partial aggregation and reduce network traffic between racks

Execution engine optimizations (2)

- Handling of stragglers
  - Hardware heterogeneity, data skew, multi-tenancy, failures, …
  - Speculative execution – execute redundant copies of the task on idle nodes → get the o st

Something wrong with node 3



Does speculative execution always help?

Design choices of Big Data systems

- Storage layer
- Programming model & exec. engine
- Scheduling
- Optimizations
- Fault tolerance

Fault tolerance

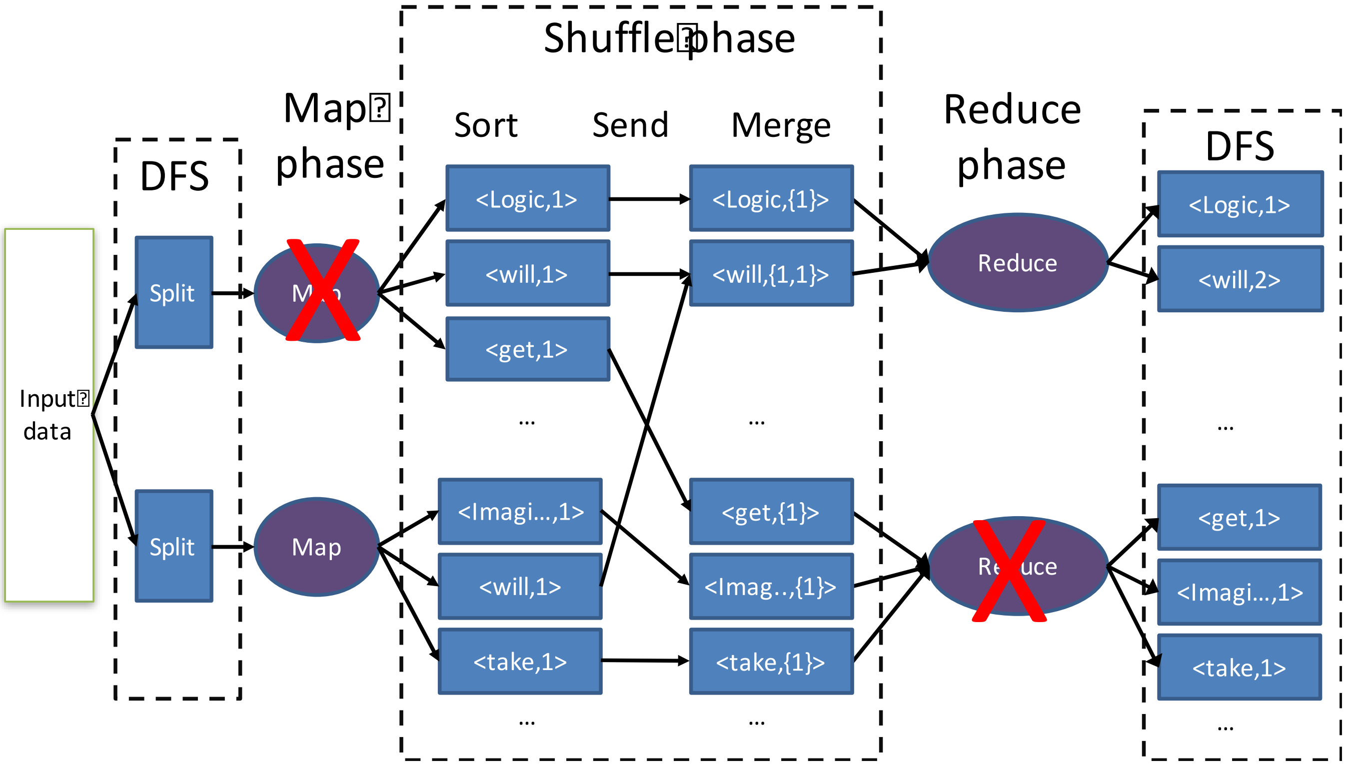
- Hardware/software failures are the rule
  - Heterogeneous hardware
  - Can be low-end, cheap, unstable
  - Network size can be in the order of thousands
  - Data skew
  - Bugs!
- Requirements
  - Data safety
  - Job recovery
    - Minimize effort – recompute as less as possible
    - Mask failures – do not delay the user!

Data safety

- Data replication
  - Full copies
  - Copy data splits
    - In the same rack
    - In different racks
  - Controlled degree of replication

Job recovery in MapReduce

- Intermediary results saved in replicated files!
- Recompute the failed maps or reduces, only on the splits that failed!



EXTRA MATERIAL - SCEPTICISM

MapReduce: A major step backwards

Dimensions to examine

- Programming paradigm
- Implementation
- Novelty
- Features
- Compatibility

Blog post by D. DeWitt and M. Stonebraker, 17 Jan. 2008

Programming paradigm

MapReduce does not support:

- Data schema
  - Best way to protect the data
- Separation of schemas from application
  - MR developer extracts structure by examining the code
  - No system catalogue
- High-level access languages
  - Offer ease of use & optimization opportunities
  - The SQL-Codasyl debate revisited

Implementation

MapReduce is poorly implemented

- Lacks fundamental big data capabilities
  - Brute-force vs indexes
  - Optimizer
- Data skew kills parallelism
- Data interchange severely unoptimized
  - Heavy I/O & network
- Rare that a MapReduce solution will beat an Oracle solution on SQL-like workload

Novelty

MapReduce is not novel

- Techniques are more than 20 years old
  - See parallel databases, distributed joins, data management over shared-nothing clusters, …
- Teradata – commercial scale-out DBMS since the 80s
- Expressivity
  - SQL & stored procedures & user defined aggregates

Features

MapReduce is missing fundamental features

- Bulk loading
- Indexing
- Updates
- Transactions
- Integrity constraints & referential integrity
- Views

Compatibility

MapReduce is incompatible with existing DBMS tools

- Reports
- BI tools
- Data mining tools
- Replication tools
- DB design tools

Summary

- 40+ years of breakthroughs and lessons in databases largely ignored
  - Need schema
  - Need declarative language
- Not so novel
- Still a long way to go for MapReduce to reach maturity of DBMS

Open problems!

- Scaling across multiple distributed clusters
  - Reducing network
  - Deciding on data and task location
- Scheduling and resource sharing
  - Across tasks, users, applications
- Powerful/expressive programming models
- Improving performance & admin. cost
  - Performance of these systems still lacks performance of RDBMS *for comparable hardware*!!!
  - Administration cost also worse

Reading material

- Jeffrey Dean, et. al.: MapReduce: simplified data processing on large clusters. Commun. ACM 2008 https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
- Shivnath Babu, Herodotos Herodotou: Massively Parallel Databases and MapReduce Systems. Foundations and Trends in Databases 5(1): 1-104 (2013). **Section 4**. Available online at http://www.nowpublishers.com/article/Details/DBS-036
- DeWitt and Stonebreaker: MapReduce: A major step backwards. https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html
- Parquet Overview: https://www.slideshare.net/julienledem/parquet-overview