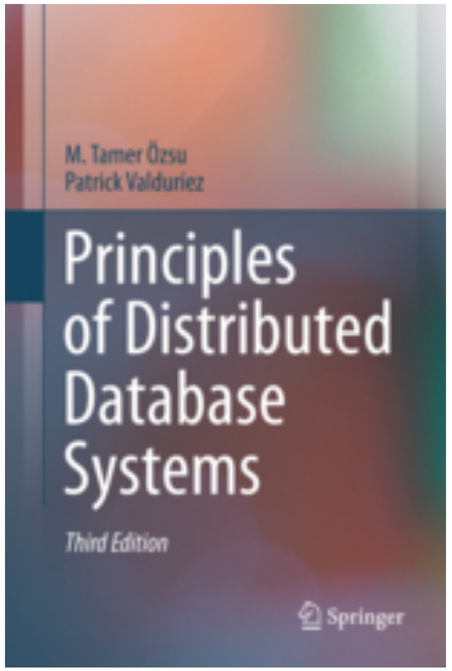
Distributed and Parallel Data Processing

Patrick Valduriez

INRIA, Montpellier

Outline

- Parallel data processing
- Parallel architectures
- Parallel techniques
- Case study: Google Search
- Database machines
- MapReduce
- Google File System
- Apache Hadoop


M. Tamer Özsu
Patrick Valduriez
Principles of Distributed Database Systems
Third Edition
Springer

When Big Data goes bad

FORTUNE

November 5, 2013: 1:00 PM ET

Recommend 32

How the models underlying today's supercomputing prowess are costing us its success.

By Joshua Klein

When Big Data goes bad – 1

- Excerpts:

  Peter Lawrence's The Making of a Fly, a classic book in developmental biology, was listed on Amazon.com as having 17 copies for sale: 15 used from $35.54, and two new from $23,698,655.93 (plus $3.99 shipping).

  What had happened was that two automated programs, one run by seller "bordeebook" and one by seller "profnath," were engaged in an iterative and incremental bidding war. Once a day profnath would raise their price to 0.9983 times bordeebook's listed price. Several hours later, bordeebook would increase their price to 1.270589 times profnath's latest amount.

When Big Data goes bad - 1

- Excerpts:

Peter Lawrence's The Making of a Fly, a classic book in developmental biology, was listed on Amazon.com as having 17 copies for sale: 15 used from $35.54, and two new from

Problem: over simplified models, but reality is complex!

profnath would raise their price to 0.9983 times bordeebook's listed price. Several hours later, bordeebook would increase their price to 1.270589 times profnath's latest amount.

When Big Data goes bad – 2

- Excerpts:

One t-shirt seller on Amazon.co.uk put up a shirt for sale emblazoned with the statement, "Keep Calm and Rape a Lot."

But Solid Gold Bomb, the company that made the shirt, wasn't necessarily aware that it was even selling it. Solid Gold Bomb's business isn't in artfully designing T-shirts. Instead, it writes code that takes libraries of words that slot into popular phrases (such as "Keep Calm and Carry On," which enjoyed a brief mimetic popularity online) to make derivations that get dropped onto a template of a T-shirt and automatically get posted as an Amazon item for sale.

Their mistake was overlooking a single word in a list of 4,000 or so others.

When Big Data goes bad – 2

- Excerpts:

One t-shirt seller on Amazon.co.uk put up a shirt for sale emblazoned with the statement, "Keep Calm and Rape a Lot."

Problem: context-independent model, but context does matter!

"Keep Calm and Carry On," which enjoyed a brief mimetic popularity online) to make derivations that get dropped onto a template of a T-shirt and automatically get posted as an Amazon item for sale.

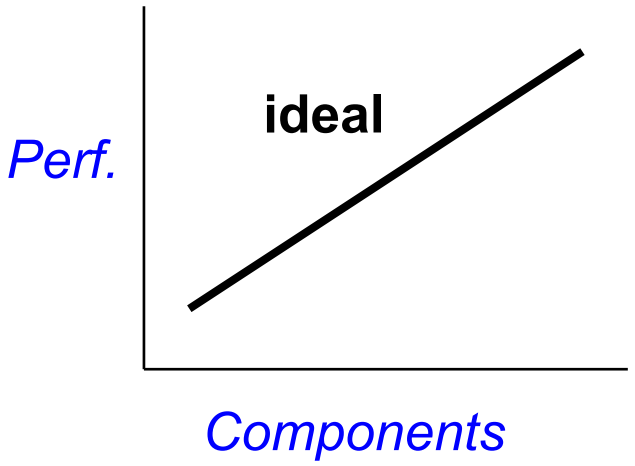
Their mistake was overlooking a single word in a list of 4,000 or so others.

The solution to big data processing!

- Exploit a massively parallel computer
  - A computer that interconnects lots of CPUs, RAM and disk units
- To obtain
  - *High performance* through data-based parallelism
    - High throughput for OLTP loads
    - Low response time for OLAP queries
  - *High availability* and reliability through data replication
  - *Extensibility* of the architecture

Extensibility

- Ideal: linear speed-up
  - Increase in performance and proportional increase of the system components (CPU, memory, disk)
    - For a constant database size and load



Speed-up limits

- Hardware/software
  - As we add more resources, arbitration conflicts increase
    - E.g. Access to the bus by processors
- Application
  - Only part of a program can be parallelized
  - Recall: Amdahl's law that gives the maximum speed-up
    - *Seq* = fraction of code that cannot be parallelized

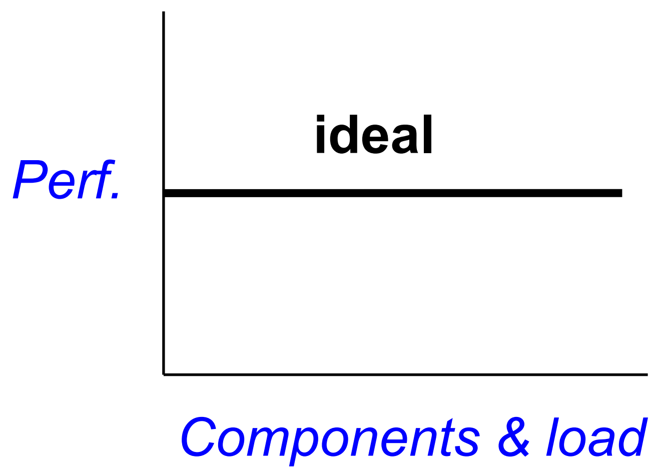
$$\frac{1}{Seq + \frac{1 - Seq}{NbProc}}$$

Examples

- Seq=0, NbProc=4 => speed-up= 4
- Seq=30%, NbProc=4 => speed-up= 2,1
- Seq=30%, NbProc=8 => speed-up= 2,5

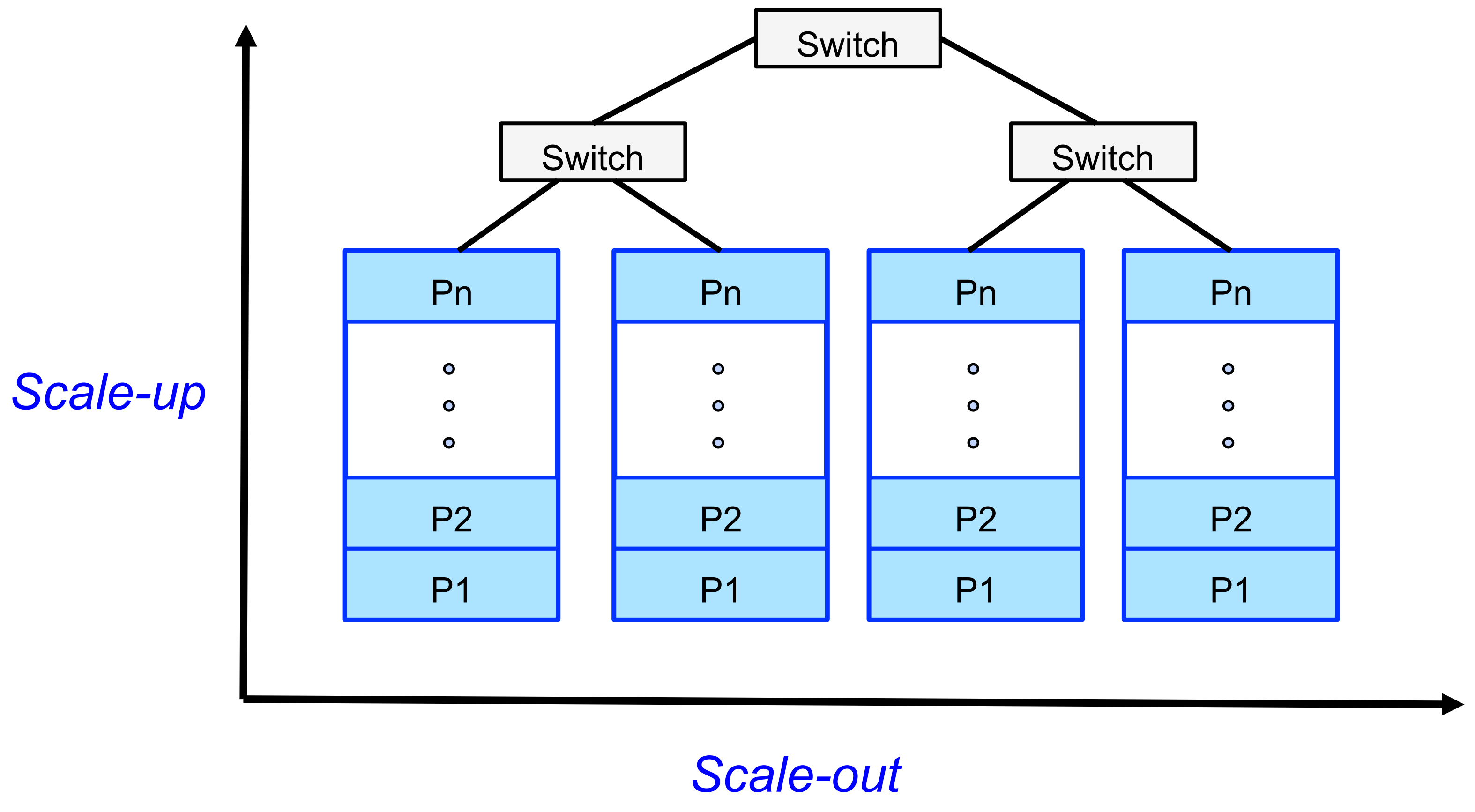
Scalability

- Ideal: linear scale-up
  - Sustained performance for a linear increase of database size and load, and proportional increase of components



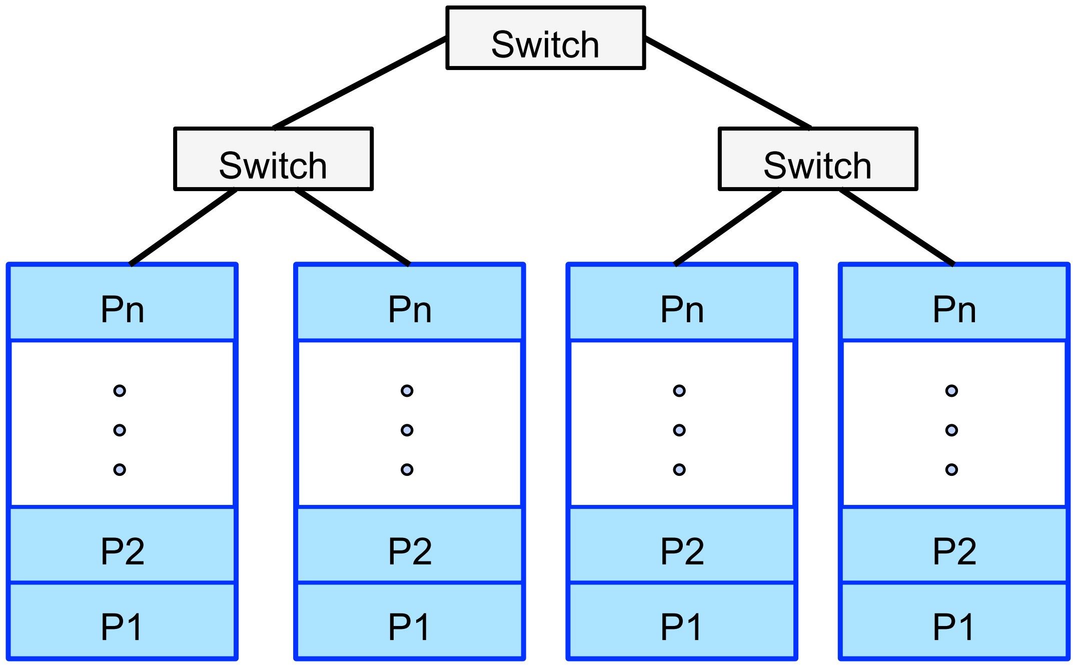
Vertical vs Horizontal Scaleup

- Typically in a computer cluster



Cluster Architecture

- Collection of computers connected by a network
  - High-speed switch-based bus
    - Infiniband, Fibre Channel, etc.
  - Each computer has its own address space
  - Distributed programming through message-passing



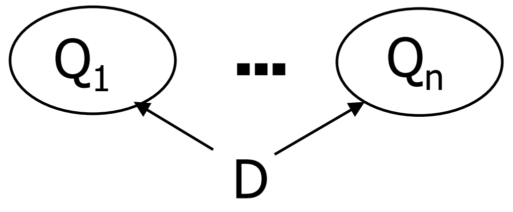
The Progress of Packaging

The early days

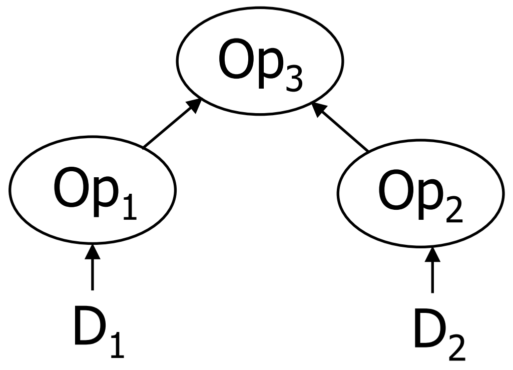
Nowdays

Data-based Parallelism

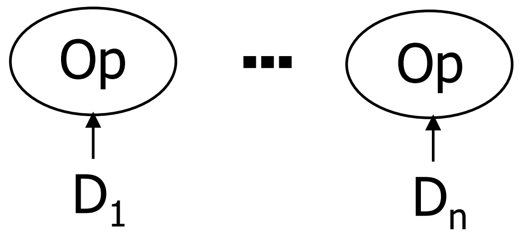
- Inter-query
  - Different queries on the same data
  - For concurrent queries



- Inter-operation
  - Different operations of the same query on different data
  - For complex queries



- Intra-operation
  - The same operation on different data
  - For large queries



Parallel Architectures for Data Management

- Three main alternatives, depending on how processors, memory and disk are interconnected
  - Shared-memory computer
  - Shared-disk cluster
  - Shared-nothing cluster

Shared-memory Computer

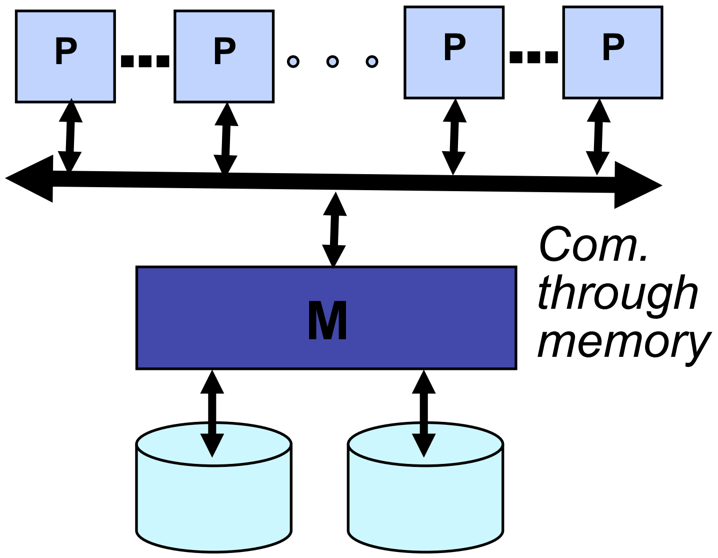
- All memory and disk are shared
  - Symmetric Multiprocessor (SMP)
  - Non Uniform Memory Architecture (NUMA)
    - Examples: IBM Numascale, HP Proliant, Data General NUMALiiNE, Bull Novascale

+ Simple pour apps

+ Load balancing

+ Fast com.

- Limited extensibility, cost



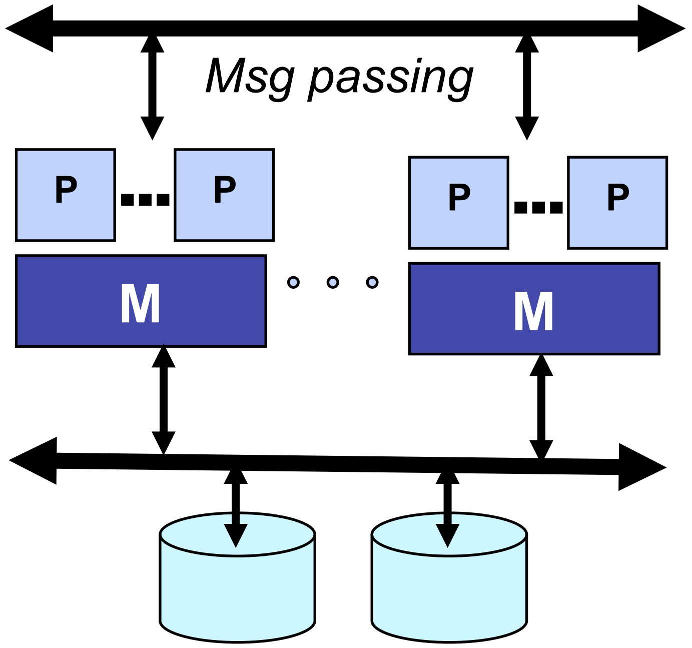
For write-intensive workloads (OLTP), expensive for big data

Shared-disk (SD) Cluster

- Disk is shared, memory is private
  - High-speed bus to interconnect memory and disk (bloc level)
    - Infiniband, Fibre Channel
  - Needs distributed lock manager (DLM) for cache coherence
  - Exemples
    - Oracle RAC et Exadata
    - IBM PowerHA

+ Simple for apps, extensibility

- Complex DLM, cost



For write-intensive workloads or big data

Shared-nothing (SN) Cluster

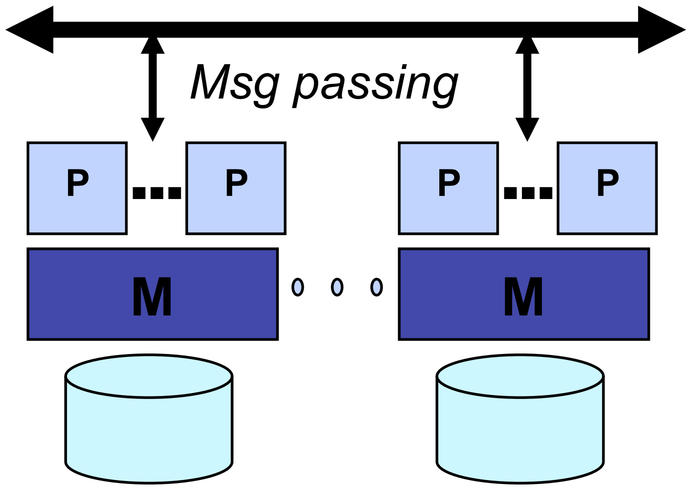
No sharing of either memory or disk across nodes

- No need for DLM
- But needs data partitioning
- Examples
  - DB2 DPF, SQL Server Parallel DW, Teradata, MySQLcluster
  - Google search, NoSQL

+ Extensibility, cost

- Complex tuning

- Updates, distributed transactions



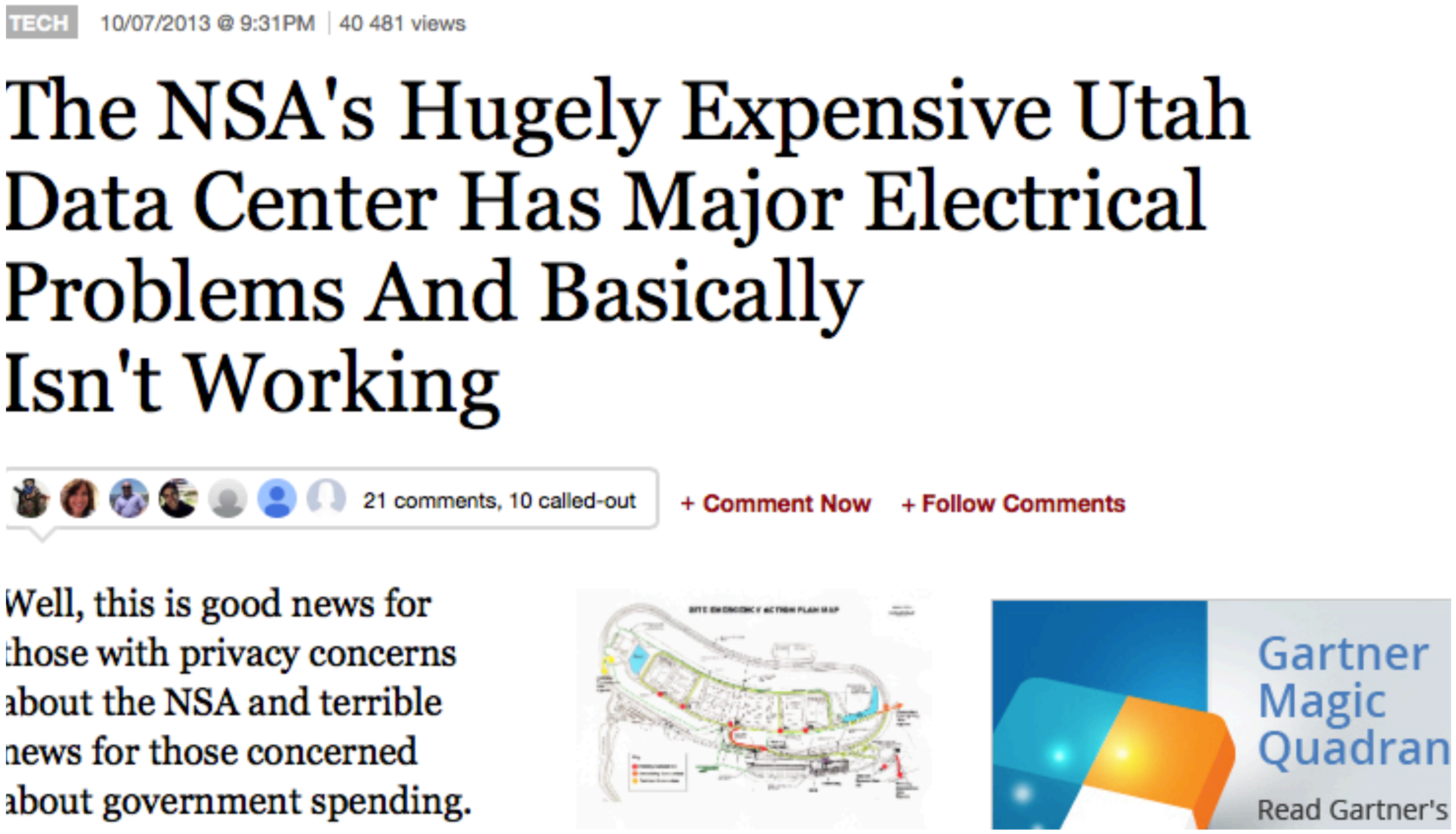
Perfect match for OLAP and big data (read intensive)

SD versus SN

- SD
  - Simple to manage (adding disks)
  - Disk high-speed bus
  - Good scalability
    - Some very good
      - ex. Exadata database machine
  - Good for OLTP (update-intensive)

- SN
  - More complex (partitioning, tuning)
  - Excellent performance/ cost ratio
  - High scalability (scale out)
  - Good for OLAP and big data (read-intensive)

When a Big Data Center Goes Bad



TECH 10/07/2013 @ 9:31PM | 40 481 views

The NSA's Hugely Expensive Utah Data Center Has Major Electrical Problems And Basically Isn't Working

21 comments, 10 called-out + Comment Now + Follow Comments

Well, this is good news for those with privacy concerns about the NSA and terrible news for those concerned about government spending.

When a Big Data Center Goes Bad

- The NSA's Hugely Expensive Utah Data Center Has Major Electrical Problems And Basically Isn't Working. *Forbes*, 2013.
- Extraits:

Well, this is good news for those with privacy concerns about the NSA and terrible news for those concerned about government spending. The National Security Agency’s new billion-dollar-plus data center in Bluffdale, Utah was supposed to go online in September, but the Wall Street Journal’s Siobhan Gorman reports that it has major electrical problems and that the facility known as “the country’s biggest spy center” is presently nearly unusable.

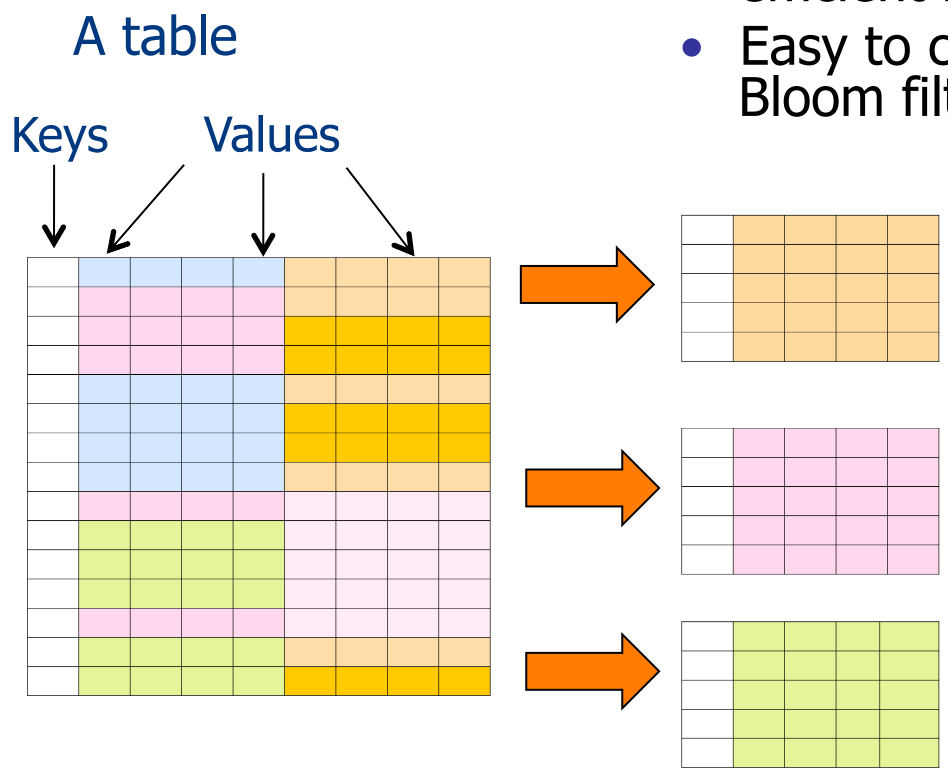
…..

“The problem, and we all know it, is that they put the appliances too close together,” a person familar with the database construction told FORBES, describing the arcs as creating “kill zones.” “They used wiring that’s not adequate to the task. We all talked about the fact that it wasn’t going to work.”

Parallel Techniques

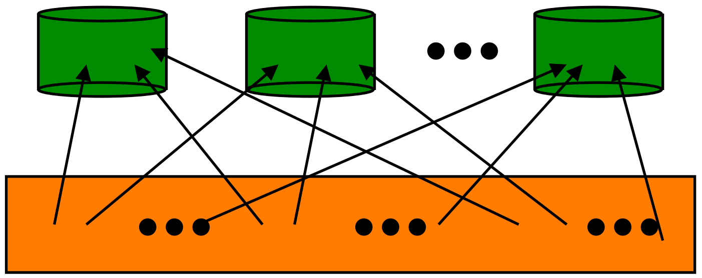
- Big datasets
  - Data partitioning and indexing
    - Problem with skewed data distributions
  - Disk is very slow (100*K* times slower than RAM)
    - Exploit RAM data structures and compression
    - Exploit SSD (read 10-100 times faster than disk)
- Query parallelization and optimization
  - Automatic if the query language is declarative (e.g. SQL)
  - Parallel algorithms for algebraic operators
    - Select is easy, Join is difficult
  - Programmer-assisted otherwise (e.g. MapReduce)
- Transaction support
  - Hard: need for distributed transactions (distributed locks and 2PC)
    - NoSQL systems don't provide transactions
- Fault-tolerance and availability
  - With many nodes (e.g. several thousand), node failure is the norm, not the exception
    - Exploit replication and failover techniques

Data Partitioning



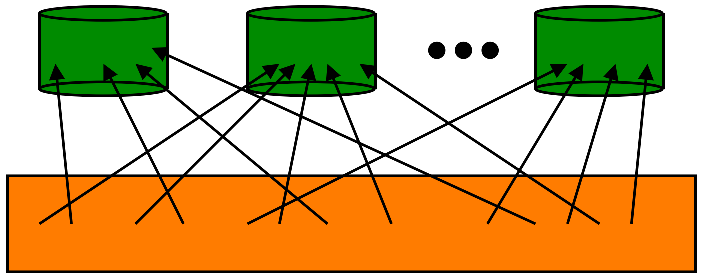
- Vertical partitioning
  - Base Basis for column stores (e.g. MonetDB, Vertica): efficient for OLAP queries
  - Easy to compress, e.g. using Bloom filters
- Horizontal partitioning (sharding)
  - Shards can be stored (and replicated) at different nodes

Sharding Schemes



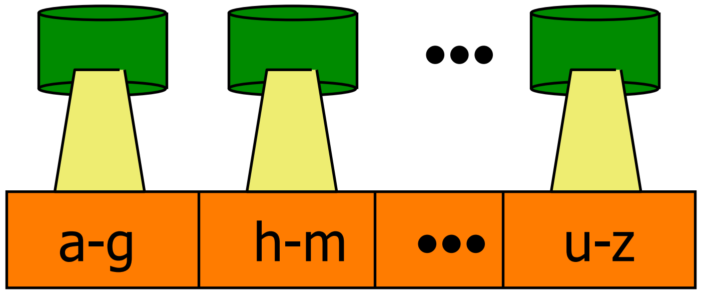
Round-Robin

- *i*th row to node (*i mod n*)
- perfect balancing
- but full scan only



Hashing

- (k,v) to node h(k)
- exact-match queries
- but problem with skew

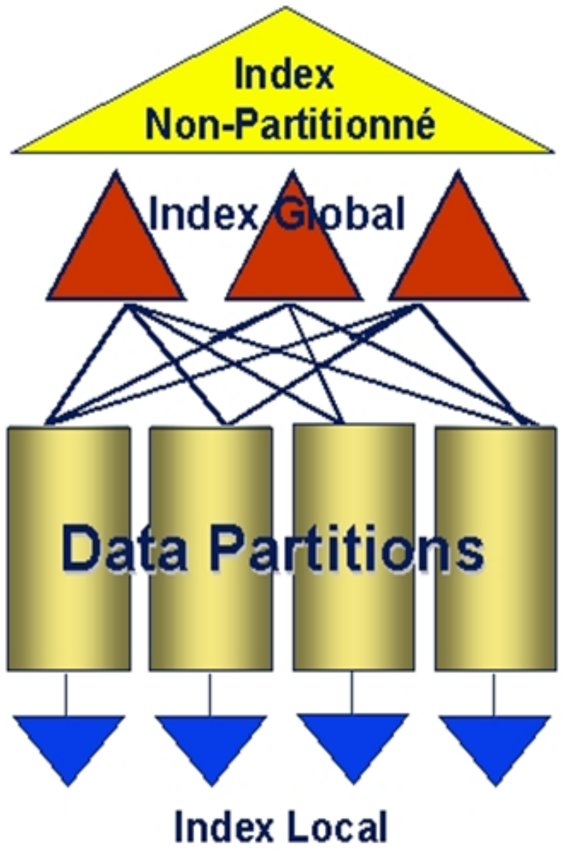


Range

- (k,v) to node that holds k's interval
- exact-match and range queries
- deals with skew

Indexing

- Functions
  - Secondary index or inverted file
- Two levels
  - Global index
    - Index *(attribute, listof (shard#, keys)*
  - Local index
    - Index *(key, value)*



Replication

- Mirror disk
  - Improves availability and performance
  - Load balancing problem in case of node failure

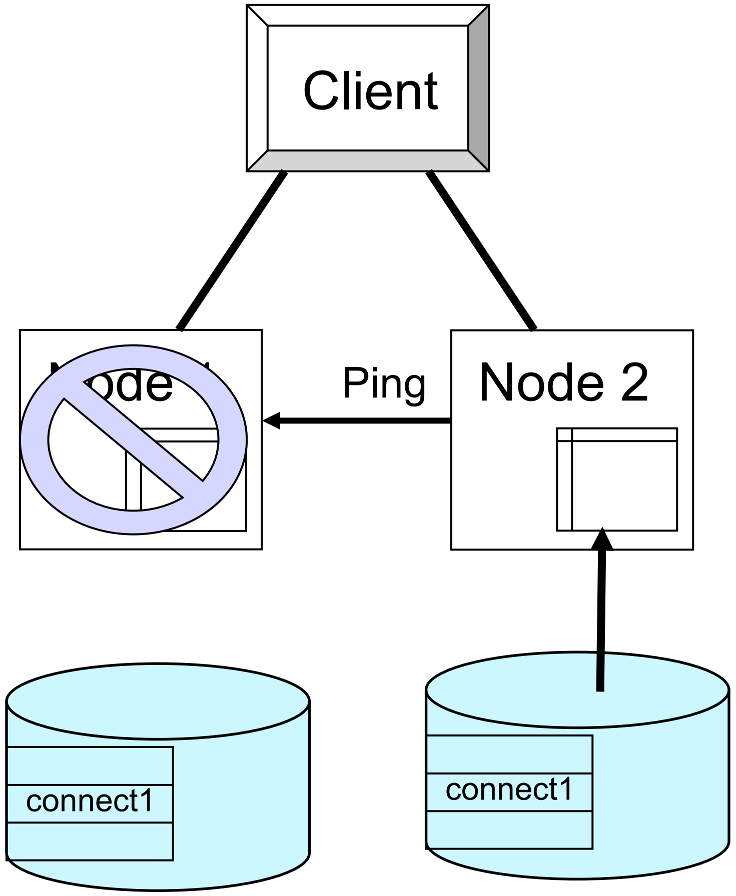
Node	1	2	3	4
Table	R_1	R_2	R_3	R_4
R_1		R_1	R_1	
R_2			R_2	R_2
R_3	R_3			R_3
R_4	R_4	R_4		

- Chained partitioning (Teradata)
  - Better load balancing
  - More complex

Node	1	2	3	4
Table	R_1	R_2	R_3	R_4
R_1		r_{12}	r_{13}	r_{14}
R_2	r_{21}		r_{23}	r_{24}
R_3	r_{31}	r_{32}		r_{34}
R_4	r_{41}	r_{42}	r_{43}	

Replication and Failover

- Replication
  - The basis for fault-tolerance and availability
  - Have several copies of each shard
- Failover
  - On a node failure, another node detects and recovers the node's tasks

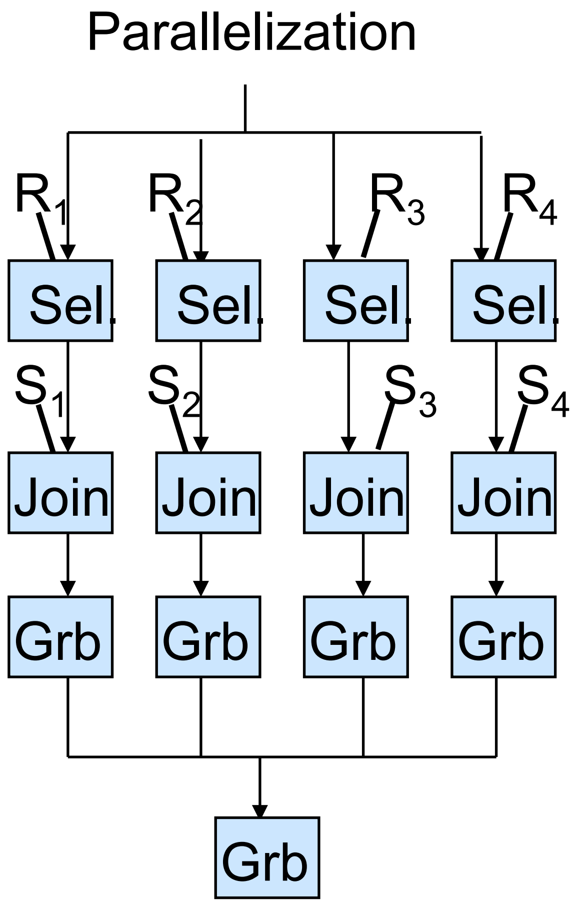


Parallel Query Processing

1. Query parallelization
   - Produces an optimized parallel execution plan, with operators
   - Based on partitioning, replication, indexing

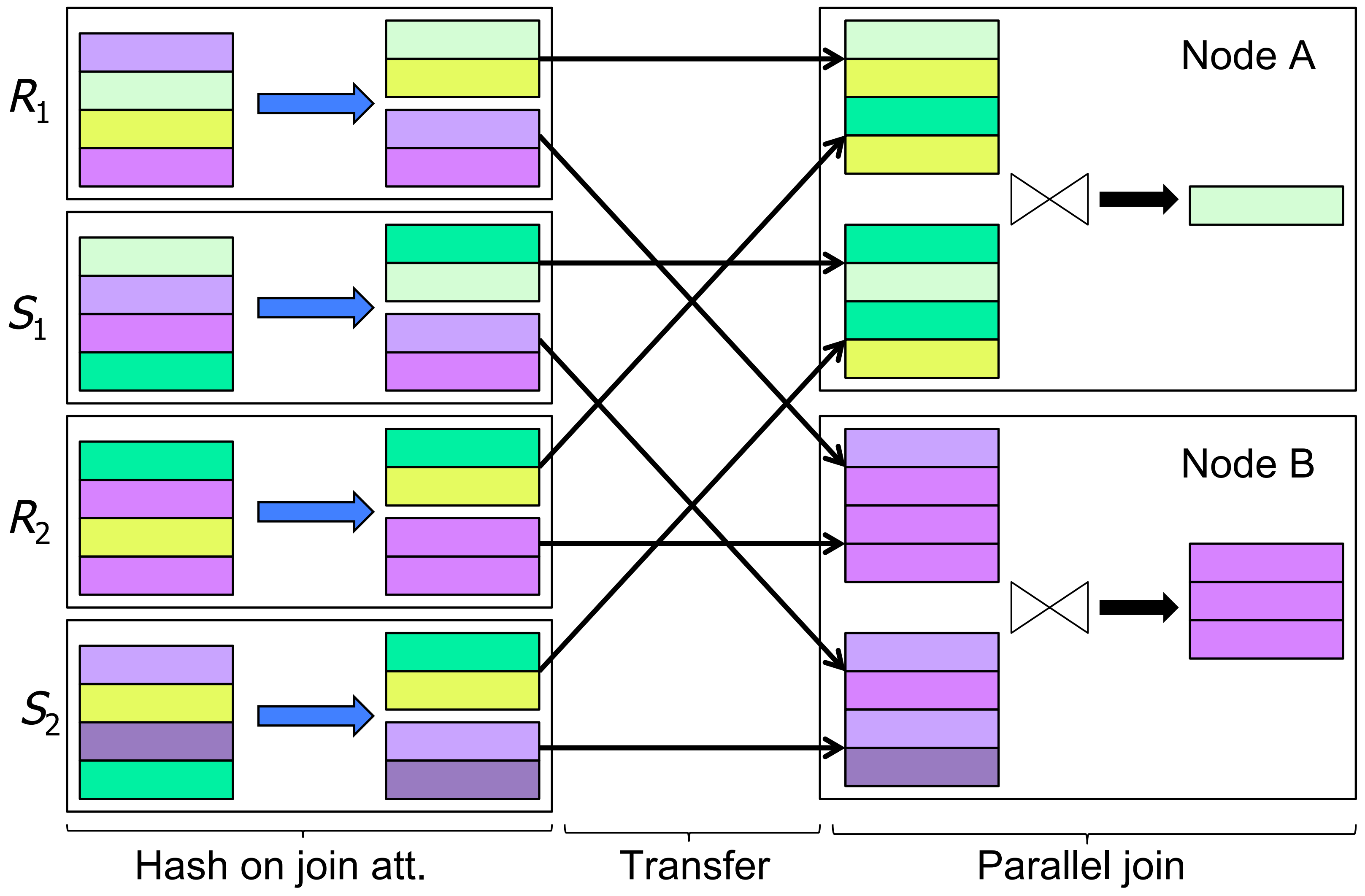
2. Parallel execution
   - Relies on parallel main memory algorithms for operators
   - Use of hashed-based join algorithms
   - Adaptive degree of partitioning to deal with skew

Select … from R,S
where …group by…



Parallel hash join algorithm

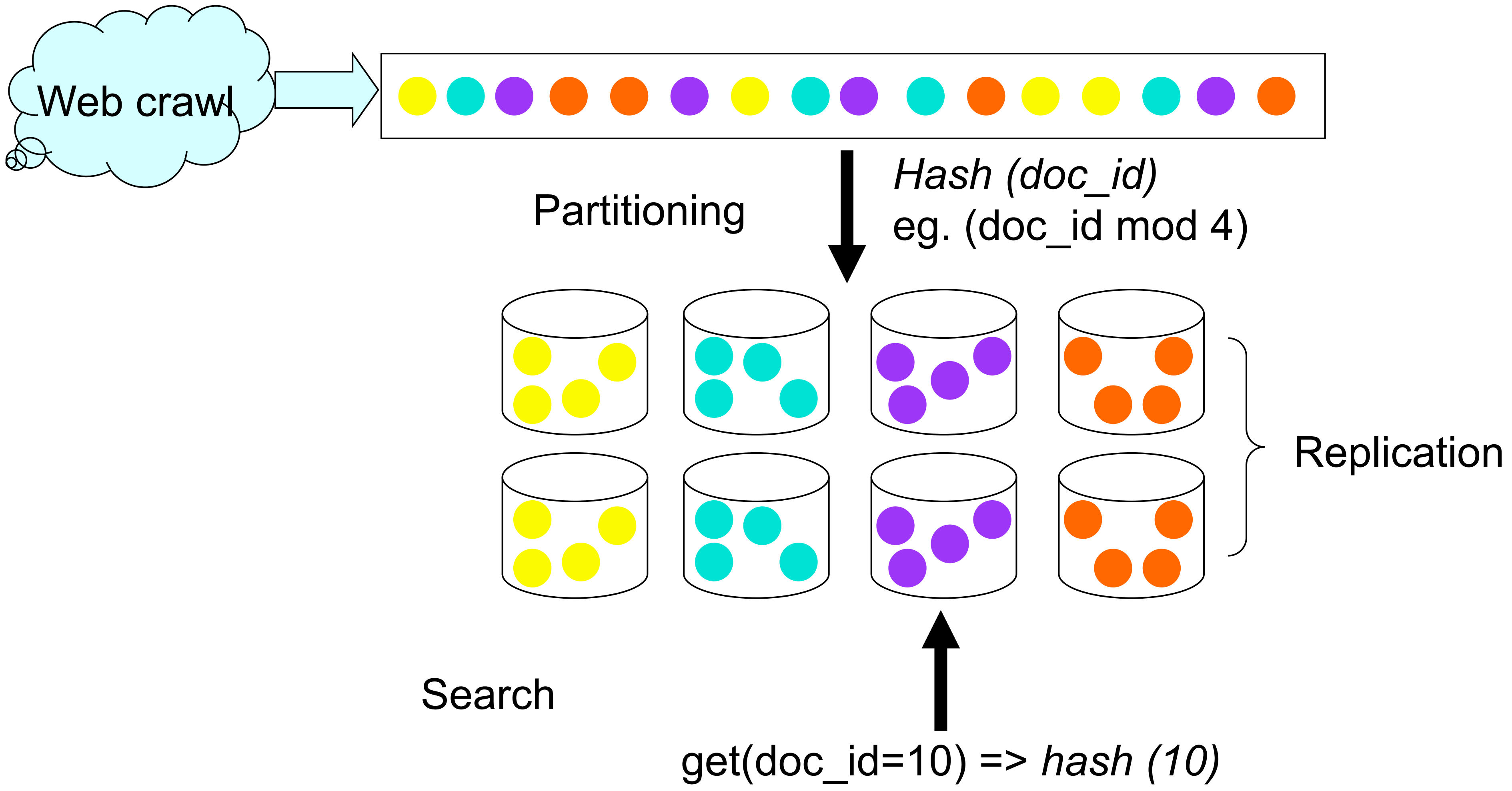
Objective: compute $R \bowtie S = \bigcup_{i=1}^{n} R_i \bowtie S_i$ with n nodes



Case Study: Google Search

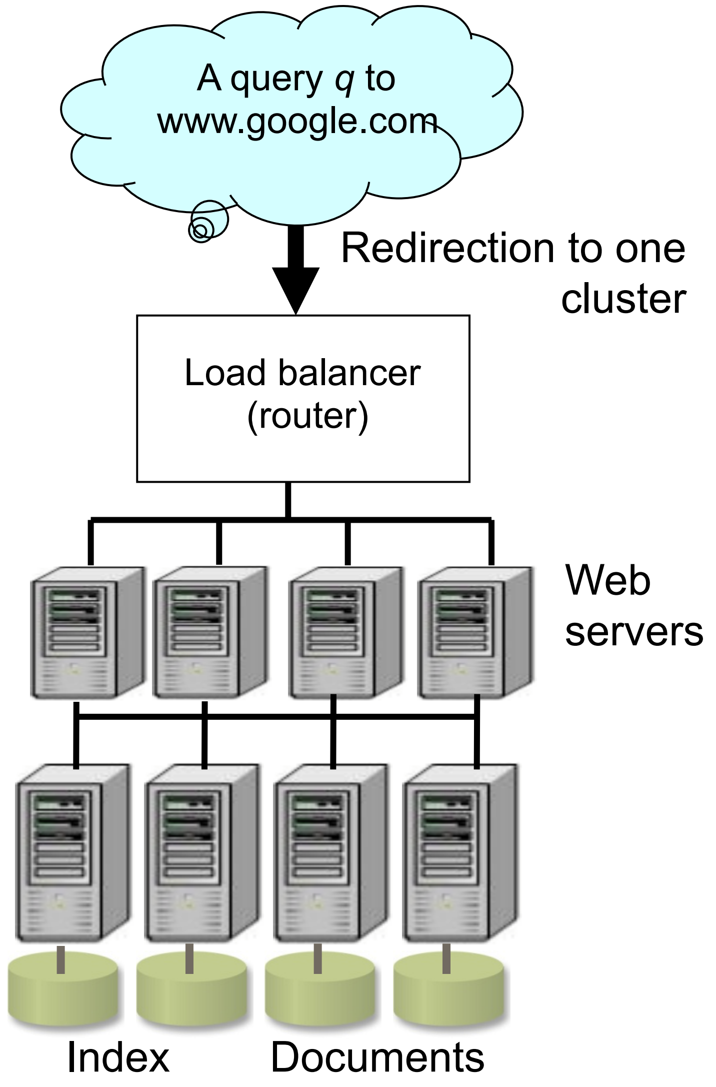
- Massive data distribution and replication in multiple clusters
  - Massive parallelism
  - Documents index: <*keyword, list of doc_ids*>
  - Data-intensive application
- Some numbers (estimation)
  - Billions of search queries per day
  - Tens of data centers in the world, each with
    - A SN cluster with a copy of the web
      - Several petabytes (billions of pages)
    - Total estimated to several millions server nodes

Partitioning Algorithm



Processing of a Google Query

1. Assignment of *q* to a web server
   - Controls the parallel execution and formats the result in HTML
2. Access to index based on *q*'s keywords
   - Produces a list of *doc_ids* sorted by relevance (PageRank algorithm)
3. Access to the documents of the list
   - Produces a summary per document



Parallel Database Machine

- A DBMS on a parallel computer
  - Combination of hardware/software dedicated to data management
    - High-speed interconnexion network
      - Infiniband, Fibre channel
    - Large main memory (RAM) and in memory techniques
    - Flash memory as cache
    - Solid State Disk
    - Multicore CPU/GPU

Main Systems

Vendor	Product	Archi.	Remarks
EMC	GreenPlum	SN	Hybrid SQL/MapReduce, based on PostgreSQL
HP	Vertica	SN	Column store
IBM	DB2 Pure Scale DB2 Database Partitioning Feature PureData System for Analytics	SD SN	Scalable POWERparallel (SP) Cluster Linux Acquisition of Netezza
Microsoft	SQL Server SQL Server PDW	SD SN	Windows only
Oracle	Real Application Cluster Exadata Database machine MySQL	SD SD SN	Portability OSS on cluster Linux
ParAccel	ParAccel Analytic Database	SN	Column store
SAP	High-Performance Analytic Appliance (HANA)	SN	In memory, column store
Teradata	Teradata Database Aster	SN SN	Unix and Windows Hybrid SQL/MapReduce

MapReduce

- A framework for big data analysis
  - Invented par Google
    - Written in C++
    - Proprietary (and protected by software patents)
- For unstructured, schemaless data
  - SQL or Xquery too heavy
- Implemented on GFS on very large clusters
  - Thousands of nodes
  - Automatic partitioning and parallelization
  - The basis for popular implementations
    - Hadoop (Apache), Hadoop++, Amazon MapReduce, etc.

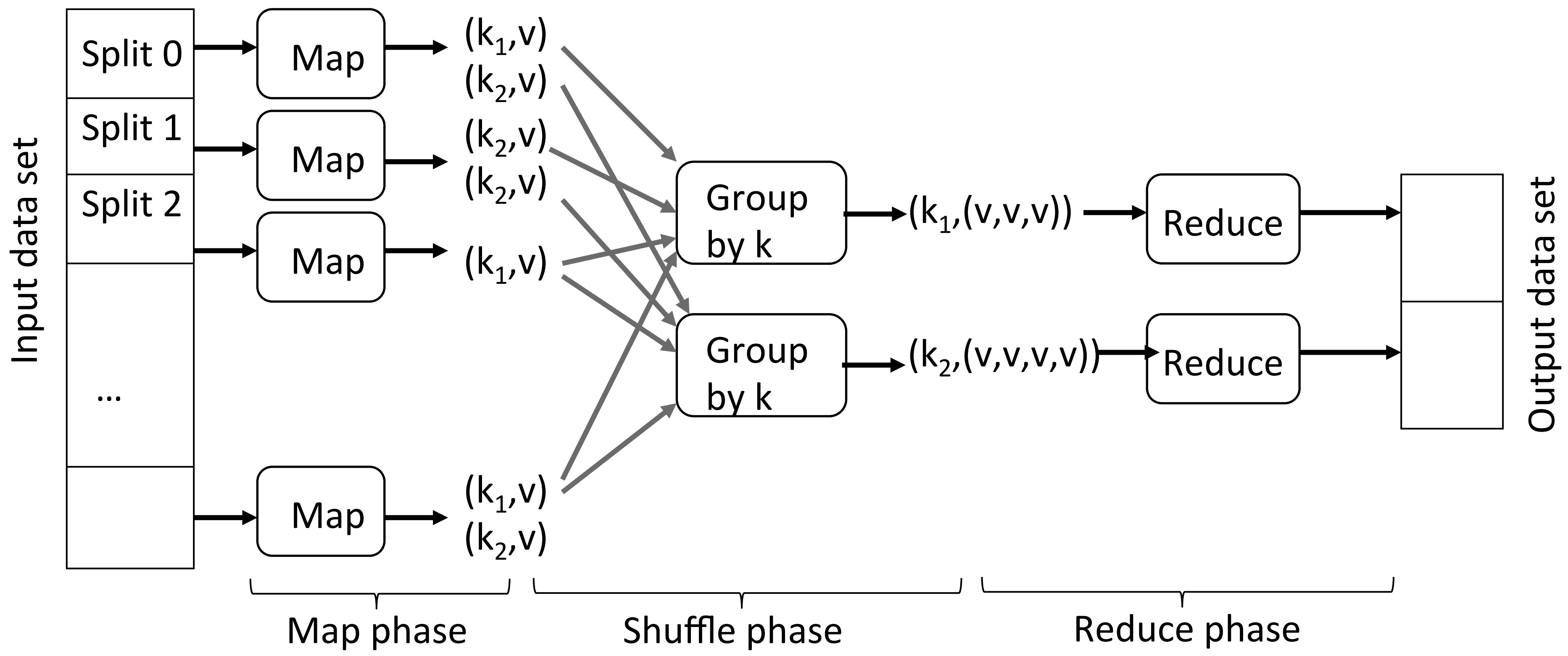
Programming Model

- Data structured as (key, value) pairs
  - E.g. (doc-id, content), (word, count), etc.
- The programmer provides the code of two functions :
  1. Map (key, value) -> list(ikey, ivalue)
     - To perform the same work in parallel on partitioned data
  2. Reduce (ikey, list(ivalue)) –> list(ikey, fvalue)
     - To aggregate the data processed by Map
- Parallel processing of Map and Reduce
  - Data partitioning
  - Fault-tolerance
  - Scheduling of disk accesses
  - Monitoring

MapReduce Typical Usages

- Counting the numbers of some words in a set of docs
- Distributed grep: text pattern matching
- Counting URL access frequencies in Web logs
- Computing a reverse Web-link graph
- Computing the term-vectors (summarizing the most important words) in a set of documents
- Computing an inverted index for a set of documents
- Distributed sorting

MapReduce Processing



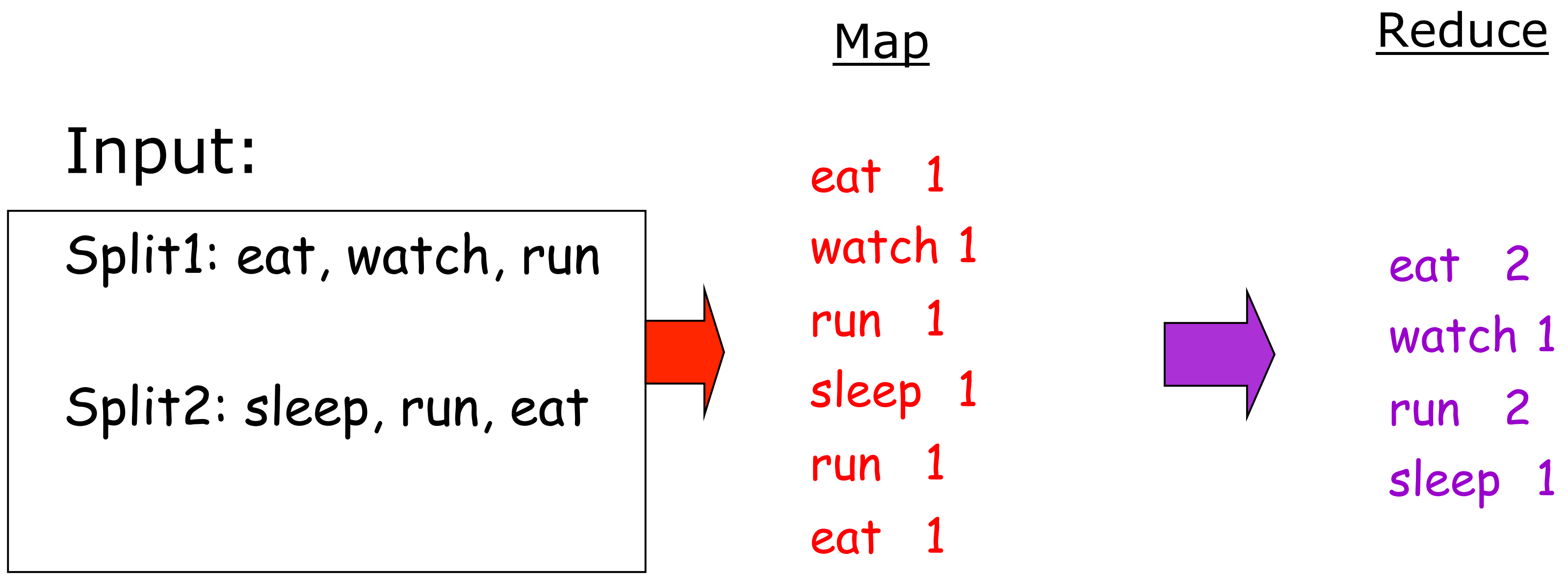
- Simple programming model
  - Key-value data storage
  - Hash-based data partitioning

Ex1: word count in a text

```
Map (key, value):
// key: file name; value: content (of a part of) a file
        for each word w in value
                EmitIntermediate (w, 1)

Reduce (key, values):
// key: a word; values: a list of 1
        result = 0
        for each value v in values
                result += v;
        Emit (key, result)
```

Ex1: illustration



Ex2: size of a web server

- Let a big file containing metadata on the size of a collection of web pages
  - Lines of the form (Server, Page URL, page size, …)
- For each server, compute the total size of the pages
  - I.e. the size of the pages of all URLs in the server

Ex2: pseudo-code

```
Map (key, value):
// key: file name; value: file lines
   for each line L(Server, Page url, Page size, …) in value
        EmitIntermediate (Server, page size);

Reduce (key, values):
// key: a server name; values: a list of page sizes
   result = 0;
   for each size s in values:
        result += s;
   Emit (key, result);
```

Ex3: reverse web links

- Let a big set of web pages
- For each page *p* in the set
  - Find the set of pages that refer to *p*
- Ex. if in pages p_1 and p_2, there are links to page *q*, then we have:
  - Sources(q) = $\{p_1, p_2, \ldots\}$

Ex3. pseudo-code

```
map(key, value):
// key: a web page URL; value: content of the page
   for each link to a target URL t in value
         EmitIntermediate (t, {key});

reduce(key, values):
// key: a URL; values: a list of URLs referencing key
   src_set = {};
   for each value v in values
         if v $\notin$ src_set then
             src_set = src_set + v;
   Emit(key, src_set);
```

Ex.4: group by

EMP (ENAME, TITLE, CITY)

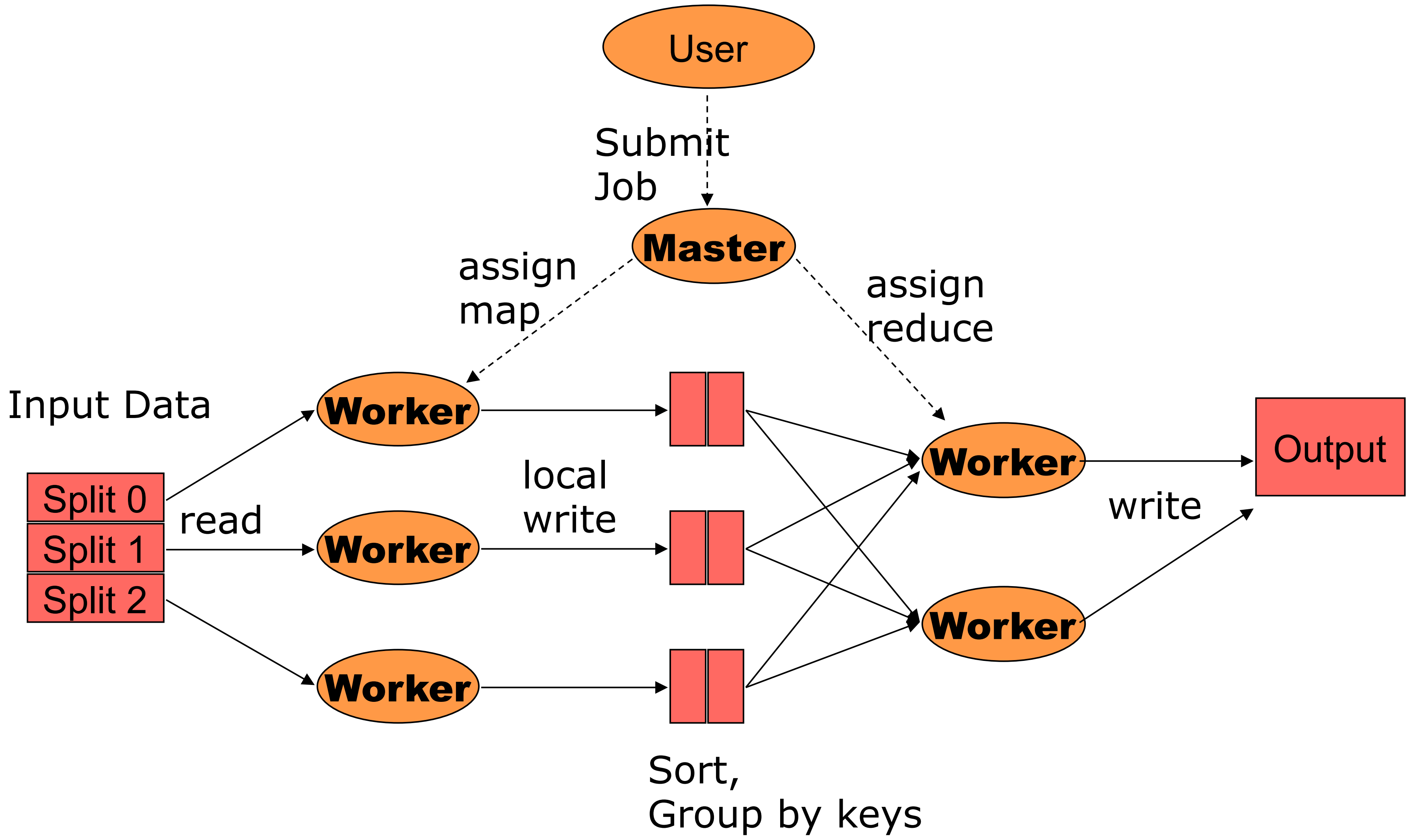
Query: for each city, return the number of employees whose name is "Smith"

```
SELECT CITY, COUNT(*)
FROM EMP
WHERE  ENAME LIKE "\%Martin"
GROUP BY CITY
```

```
Map (Input (TID,emp), Output: (CITY,1))
// TID: tuple identifier, emp: one row of EMP
        if emp.ENAME like "%Martin"
            EmitIntermediate (CITY,1)

Reduce (Input (CITY,list(1)), Output: (CITY,SUM(list(1)))
        Emit (CITY,SUM(1*))
```

MapReduce Architecture



Task Scheduling

- Dynamic approach
  - State of a task: inactive, active, terminated
  - Inactive tasks are activated as *worker* nodes become available
    - They are assigned to the workers that are closest to input data
      - Eg. Local disk or same rack, to reduce inter-node transfers
  - When a task ends, it sends to the master the addresses and sizes of intermediate data
  - When all the Map tasks have terminated, the Reduce tasks start

Fault-tolerance

- Fault-tolerance is fine-grain and well suited for large jobs
- Input and output data are stored in GFS
  - Already provides high fault-tolerance
- All intermediate data is written to disk
  - Helps checkpointing Map operations, and thus provides tolerance from soft failures
- If one Map node or Reduce node fails during execution (hard failure)
  - The tasks are made eligible by the master for scheduling onto other nodes
  - It may also be necessary to re-execute completed Map tasks, since the input data on the failed node disk is inaccessible

Google File System (GFS)

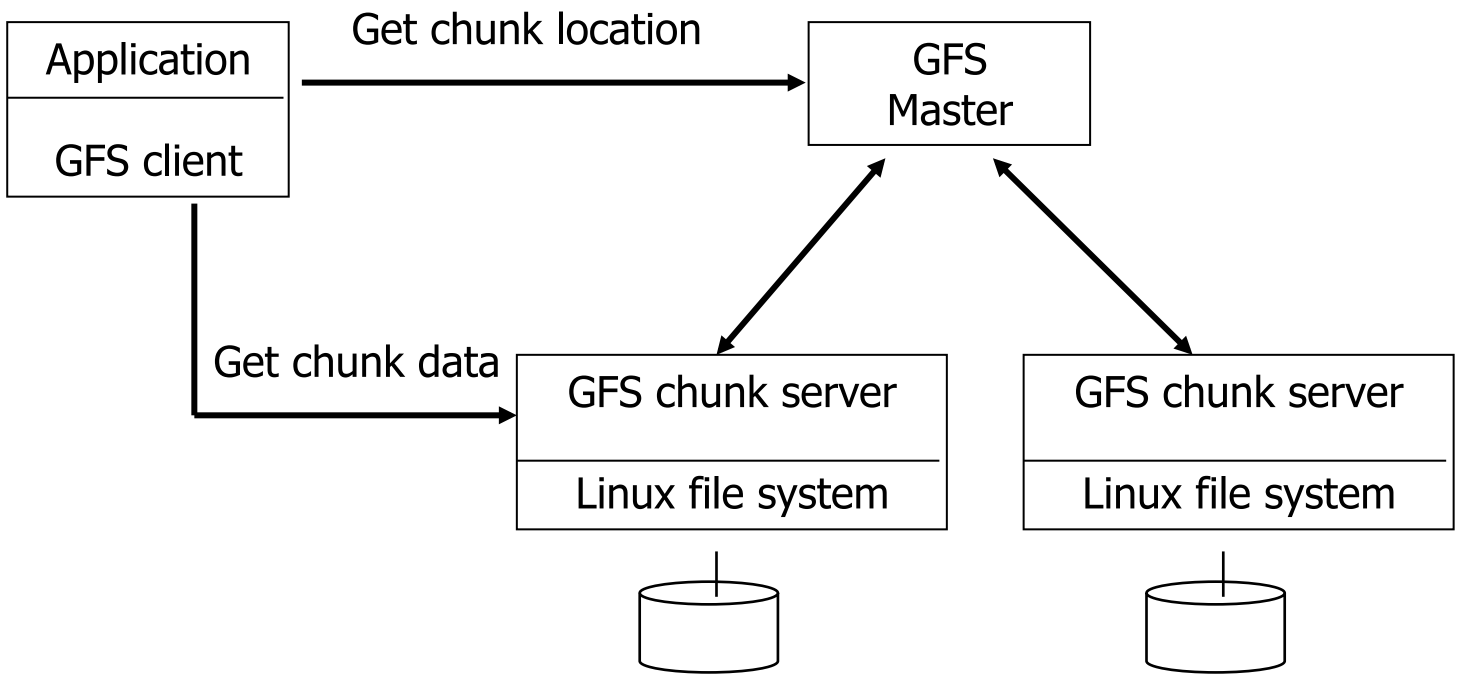
- Used by many Google applications
  - Search engine, Bigtable, Mapreduce, etc.
- The basis for popular Open Source implementations
  - Hadoop HDFS (Apache & Yahoo)
- Optimized for specific needs
  - Shared-nothing cluster of thousand nodes, built from inexpensive harware => node failure is the norm!
  - Very large files, of typically several GB, containing many objects such as web documents
  - Mostly read and append (random updates are rare)
    - Large reads of bulk data (e.g. 1 MB) and small random reads (e.g. 1 KB)
    - Append operations are also large and there may be many concurrent clients that append the same file
    - High throughput (for bulk data) more important than low latency

Design Choices

- Traditional file system interface (create, open, read, write, close, and delete file)
  - Two additional operations: snapshot and record append.
- Relaxed consistency, with atomic record append
  - No need for distributed lock management
  - Up to the application to use techniques such as checkpointing and writing self-validating records
- Single GFS master
  - Maintains file metadata such as namespace, access control information, and data placement information
  - Simple, lightly loaded, fault-tolerant
- Fast recovery and replication strategies

GFS Distributed Architecture

- Files are divided in fixed-size partitions, called *chunks*, of large size, i.e. 64 MB, each replicated at several nodes



Apache Hadoop

- OSS framework for storing and analyzing big data on very large clusters
  - Written in Java
  - Initially created by Yahoo
  - The basis for an major ecosystem
- Modules
  - Hadoop Common: library of codes and utilities
  - Hadoop YARN: resource management in a cluster
  - Hadoop Distributed File System (HDFS): a GFS clone
  - Hadoop MapReduce
- Complementary tools
  - Apache Pig: workflow-style interface
  - Apache Hive: SQL-style interface

MapReduce Assessment

- Advantages
  - Simple for the programmer
  - Parallelization, fault-tolerance, scalability
  - For unstructured data
- A very large community of developpers
  - Adopted by all web giants
    - Google, Facebook, Amazon, etc.
  - And software vendors
    - Oracle, IBM, Microsoft, etc.
      - NB: Microsoft gave up on its Dryad competitor
- Much room for improvement (see MapReduce workshops)
  - Map phase
    - Minimize I/0 cost using indices (Hadoop++)
  - Shuffle phase
    - Minimize data transfers by partitioning data on the same intermediate key
    - Current work in Zenith
  - Reduce phase
    - Exploit fine-grain parallelism of Reduce tasks
    - Current work in Zenith

MapReduce vs Parallel DBMS

- [Pavlo et al. SIGMOD09]: Hadoop MapReduce vs two parallel DBMS, one row-store DBMS and one column-store DBMS
  - Benchmark queries: a grep query, an aggregation query with a group by clause on a Web log, and a complex join of two tables with aggregation and filtering
  - Once the data has been loaded, the DBMS are significantly faster, but loading is much time consuming for the DBMS
  - Suggest that MapReduce is less efficient than DBMS because it performs repetitive format parsing and does not exploit pipelining and indices
- [Dean and Ghemawat, CACM10]
  - Make the difference between the MapReduce model and its implementation which could be well improved, e.g. by exploiting indices
- [Stonebraker et al. CACM10]
  - Argues that MapReduce and parallel DBMS are complementary as MapReduce could be used to extract-transform-load data in a DBMS for more complex OLAP

Some MapReduce Solutions

Vendor	Product	Remarks
Google	MapReduce	Proprietary, C++, on GFS data Python and Java API in AppEngine-MapReduce
Apache Hadoop	MapReduce	OSS, Java, on HDFS data Interfaces: C++, Unix streams for any language
Apache	Pig	Interface for MapReduce jobs
Apache	Hive	SQL / MapReduce interface
Cloudera	Dist. Hadoop	Hadoop services and products
Amazon	Amazon Elastic MapReduce	MapReduce for Amazon cloud
IBM	InfoSphere BigInsights	Bigdata analysis platform including Hadoop
Microsoft	HDInsight	MapReduce platform for Azure cloud
Oracle	Bigdata Appliance	Bigdata analysis platform including Hadoop (Cloudera)

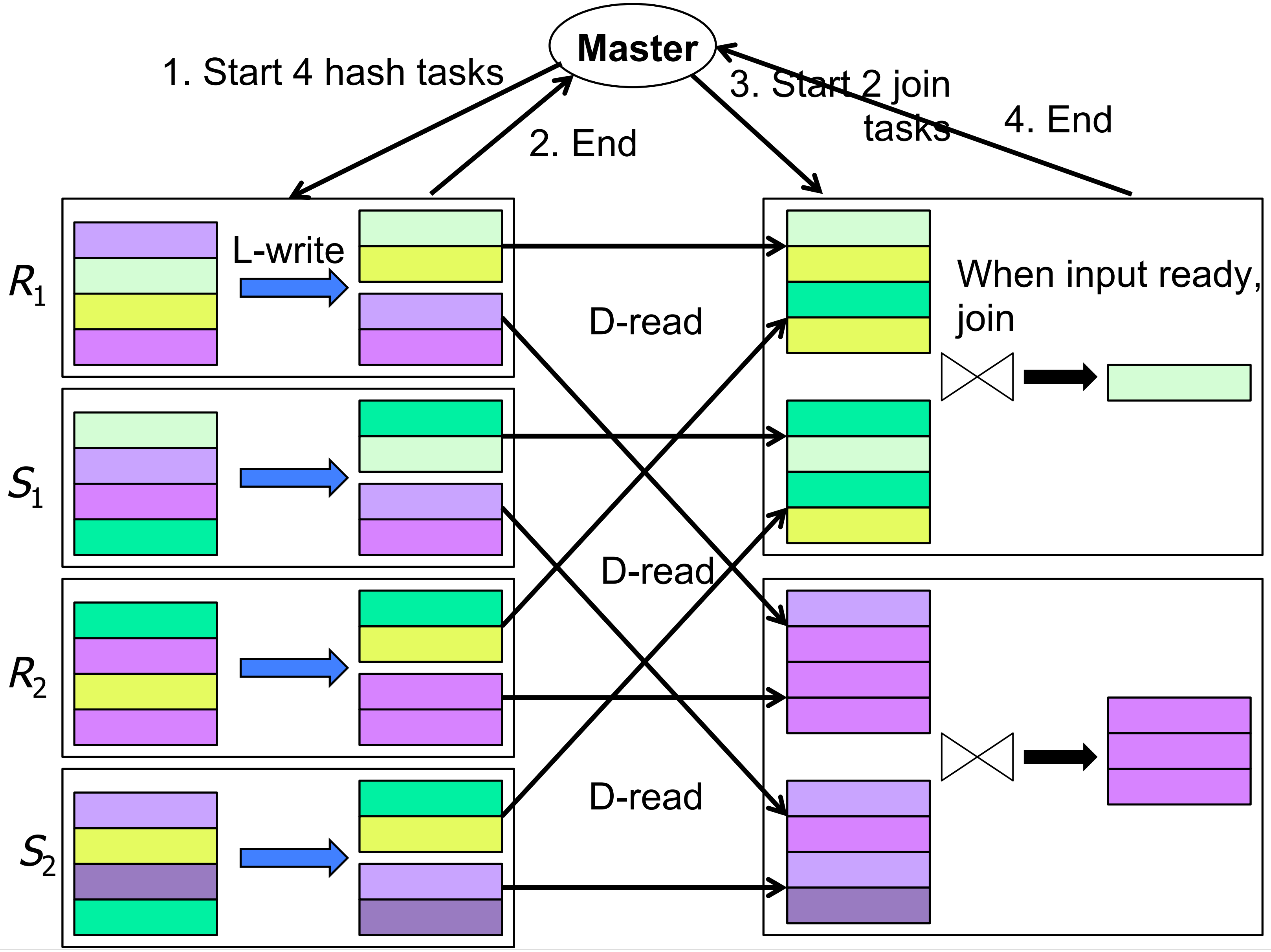
MapReduce Best Practices

- *Big data is not Hadoop only*
- When to use MapReduce?
  - Unstructured data, without precise schema
    - Repetitive structure, easy to partition
  - Batch-type processes and analyses
  - Need for low-cost big processing
  - Strong development expertise, not in databse
- When not to use?
  - Data streams and continuous/incremental processing
  - Real-time analysis, with guaranteed response time
  - Access to shared data, with updates
  - What about structured data?

Exercise 1: Parallel Algorithm Design

- Objective
  - Design an efficient version of the parallel hash-based join algorithm
- Assumptions
  - A parallel shared-nothing cluster
  - Two tables *R* and *S*, partitioned on a number of nodes
    - $R_1, R_2, \ldots R_m$ and $S_1, S_2, \ldots, S_n$
  - Two kinds of tasks that can run at any node
    - Master task: has global information (partitioning, nodes's load, etc) and controls all the workers
    - Worker task: obeys the master
- Interfaces
  - Master-Worker
    - Start a task, with one input buffer and one or more output buffers (for storing partitions)
  - Worker-master
    - Notify master of end of work
- Data transfer between workers (like remote pipes)
  - Write to a distant buffer (at a different worker)
  - Read from a distant buffer
  - D-read and D-write are blocking operations
- Work to do
  - Write pseudo code for Master and Worker's tasks
  - Illustrate with a figure

Exercise 1: Solution



Exercice 1: Solution discussion

- How to improve performance?
  - Pipelining between workers
    - Requires non blocking d-read/d-write
- What can go wrong?
  - Worker failure
    - Requires failure detection and failover

Exercise 2: Map Reduce Design

- Objective
  - Compute the join of 2 tables R and S with MapReduce
- Assumptions
  - R and S contained in a input file
  - Structured records
    - The join key can be accessed
- Work to do
  - Write pseudo code for Map and Reduce

Exercise 2: Solution

```
Map (K: null, V : a row of a split of R or S)
   join key = extract the join column from V
   tagged record = add a tag of either R or S
   EmitIntermediate (join key, tagged record)

Reduce (K: join key, V: rows of R and S having K as join key)
   create buffers BR and BS for R and S, respectively
   for each record t in V do
         append t to one of the buffers according to its tag
         for each pair of records (r, s) in BR × BS do
               Emit (null, new record(r, s))
```