MapReduce and Hadoop

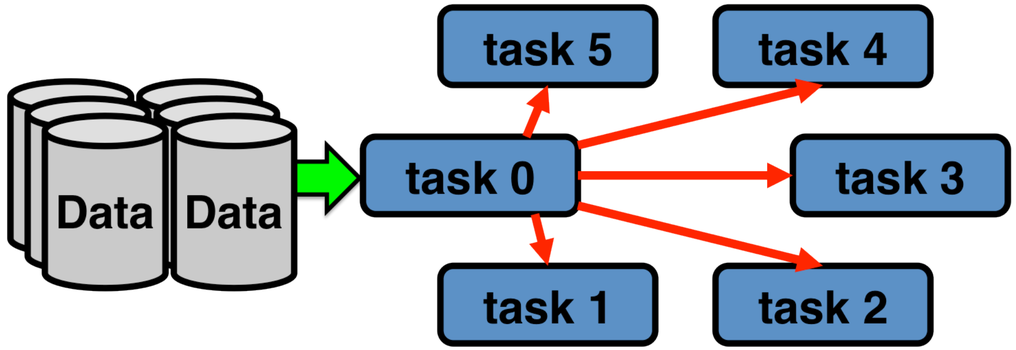
Debapriyo Majumdar

Indian Statistical Institute

debapriyo@isical.ac.in

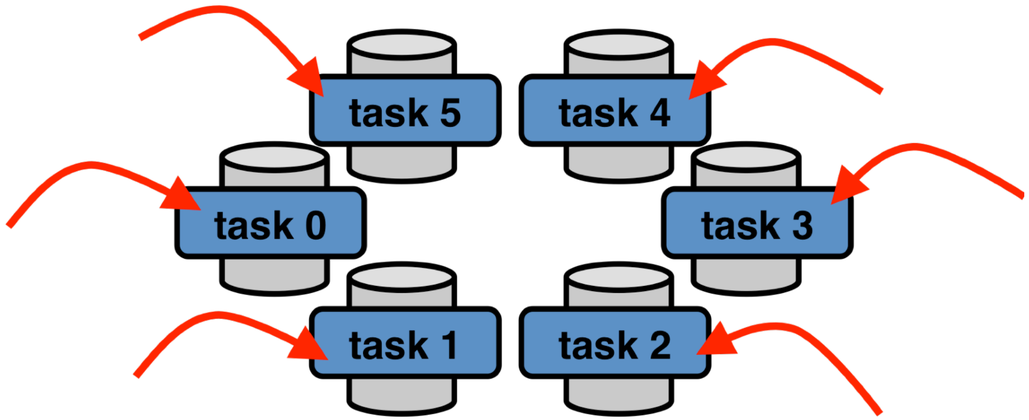
Computer(s) and Data

- Modern data mining: process immense amount of data quickly
- Exploit parallelism



Traditional parallelism

Bring data to the computer

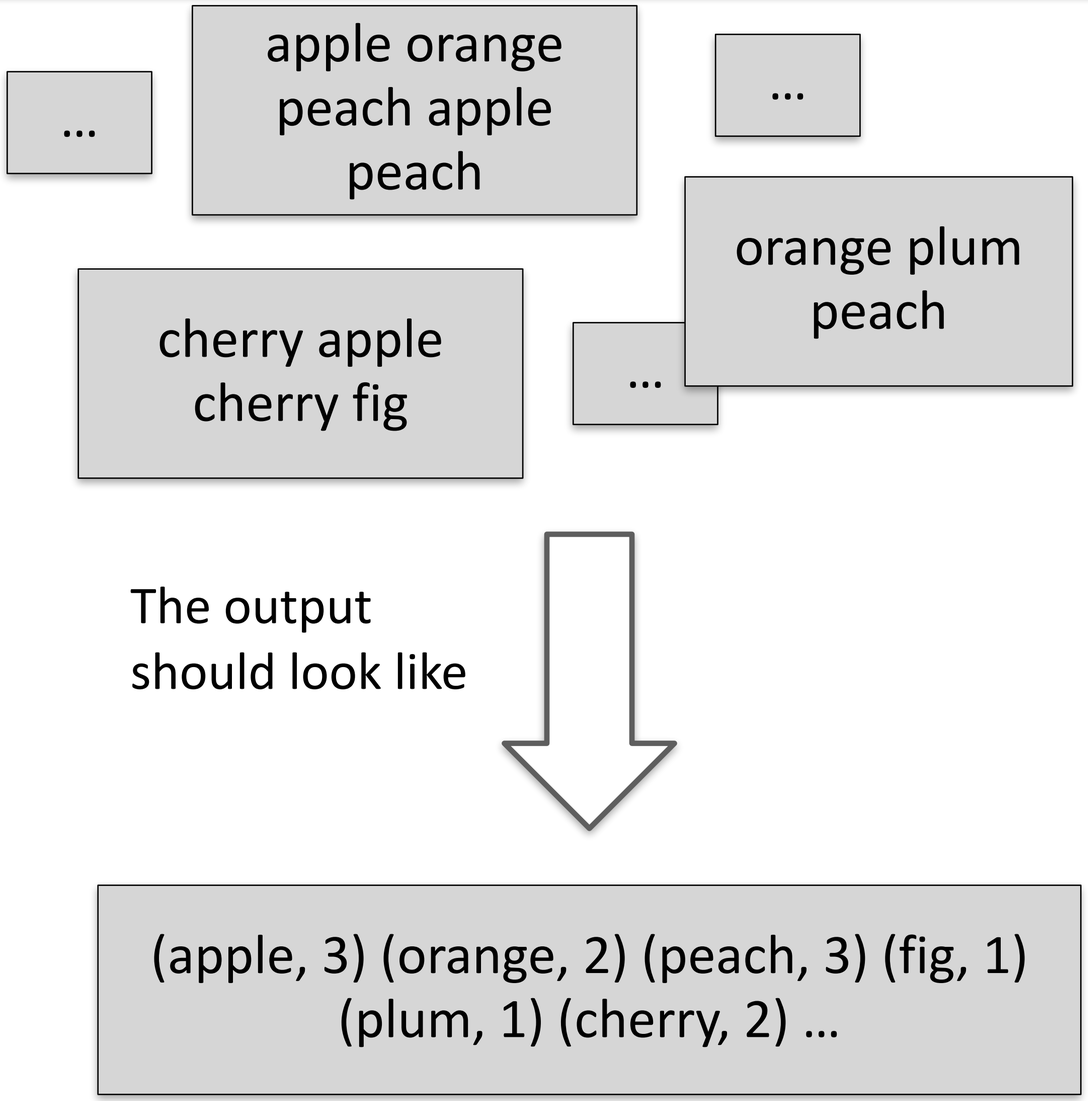


MapReduce

Compute where the data is present

Pictures courtesy: Glenn K. Lockwood, glennklockwood.com

The “Hello World” problem of Big Data



- Many documents, say millions, billions
  - Each with several words
- Goal: word count
  - Count total number of occurrences for each word

Normally, how would we do it?

Read through documents

Document 1

apple

orange

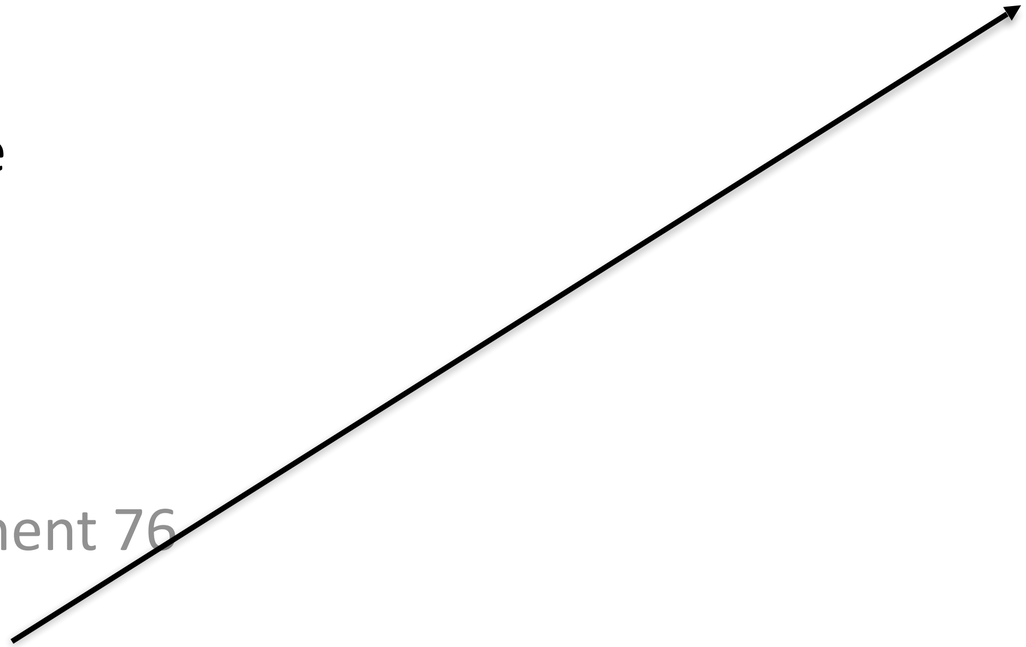
peach

…

…

Document 76

apple



Update counters in memory

(apple, 1)

(orange, 1)

(peach, 1)

Normally, how would we do it?

Read through documents

Document 1

apple

orange

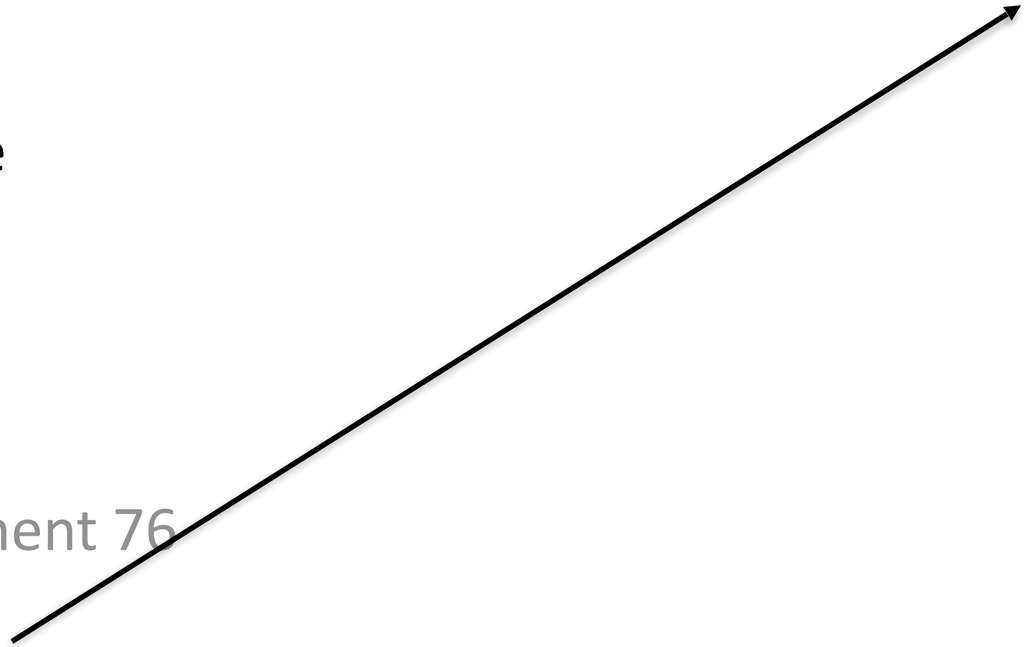
peach

…

…

Document 76

apple



Update counters in memory

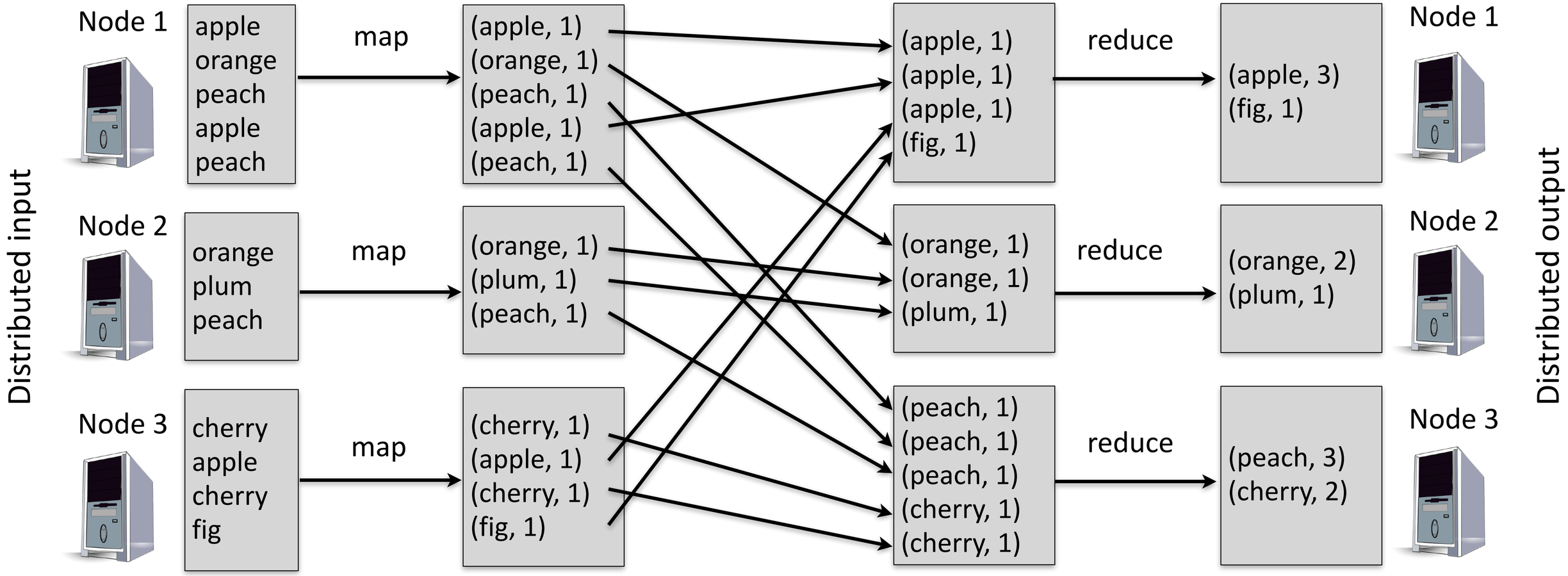
(apple, **2**)

(orange, 1)

(peach, 1)

Word frequency using MapReduce

Given a collection of documents, count the number of times each word occurs in the collection

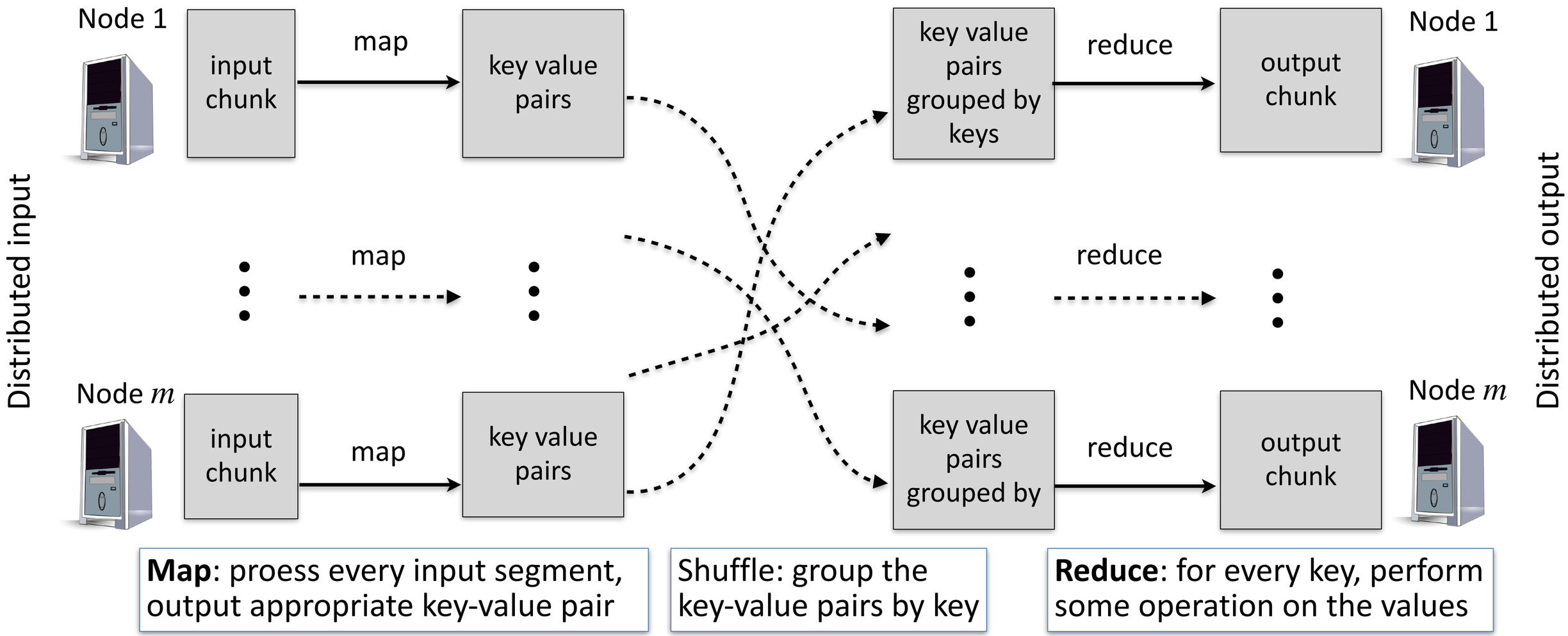


Map: for every word w, output the key-value pair $(w,1)$

Shuffle: group the key-value pairs by key

Reduce: for every key, sum the values

The MapReduce Paradigm



The user (programmer) needs to write the **map** and the **reduce** functions

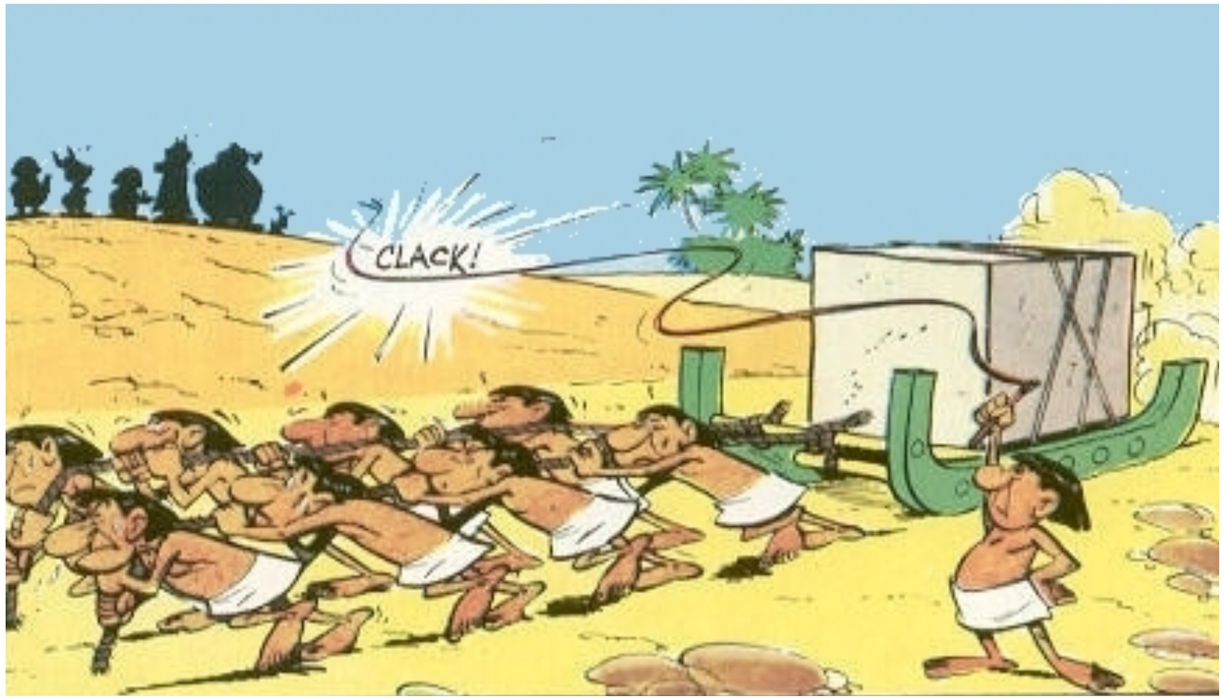
Apache Hadoop

An open source MapReduce framework

Hadoop

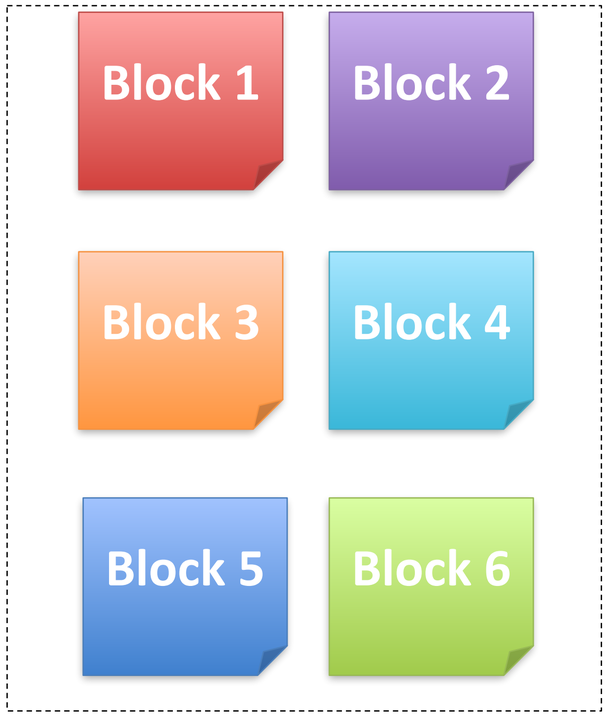
Hadoop

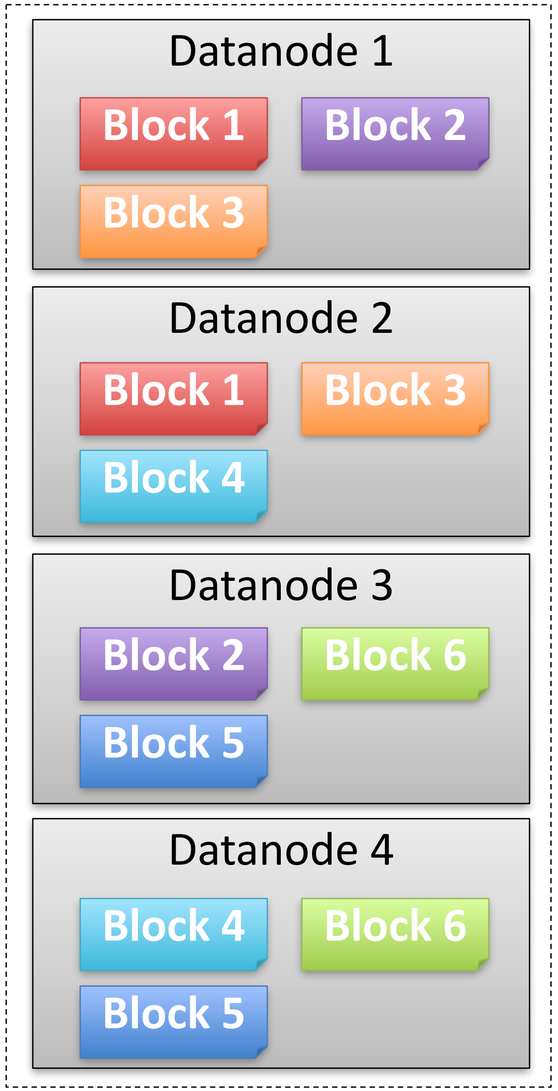
- Two main components
  - **Hadoop Distributed File System** (HDFS): to store data
  - **MapReduce engine**: to process data
- Master – slave architecture using commodity servers



- The HDFS
  - Master: Namenode
  - Slave: Datanode
- MapReduce
  - Master: JobTracker
  - Slave: TaskTracker

HDFS: Blocks





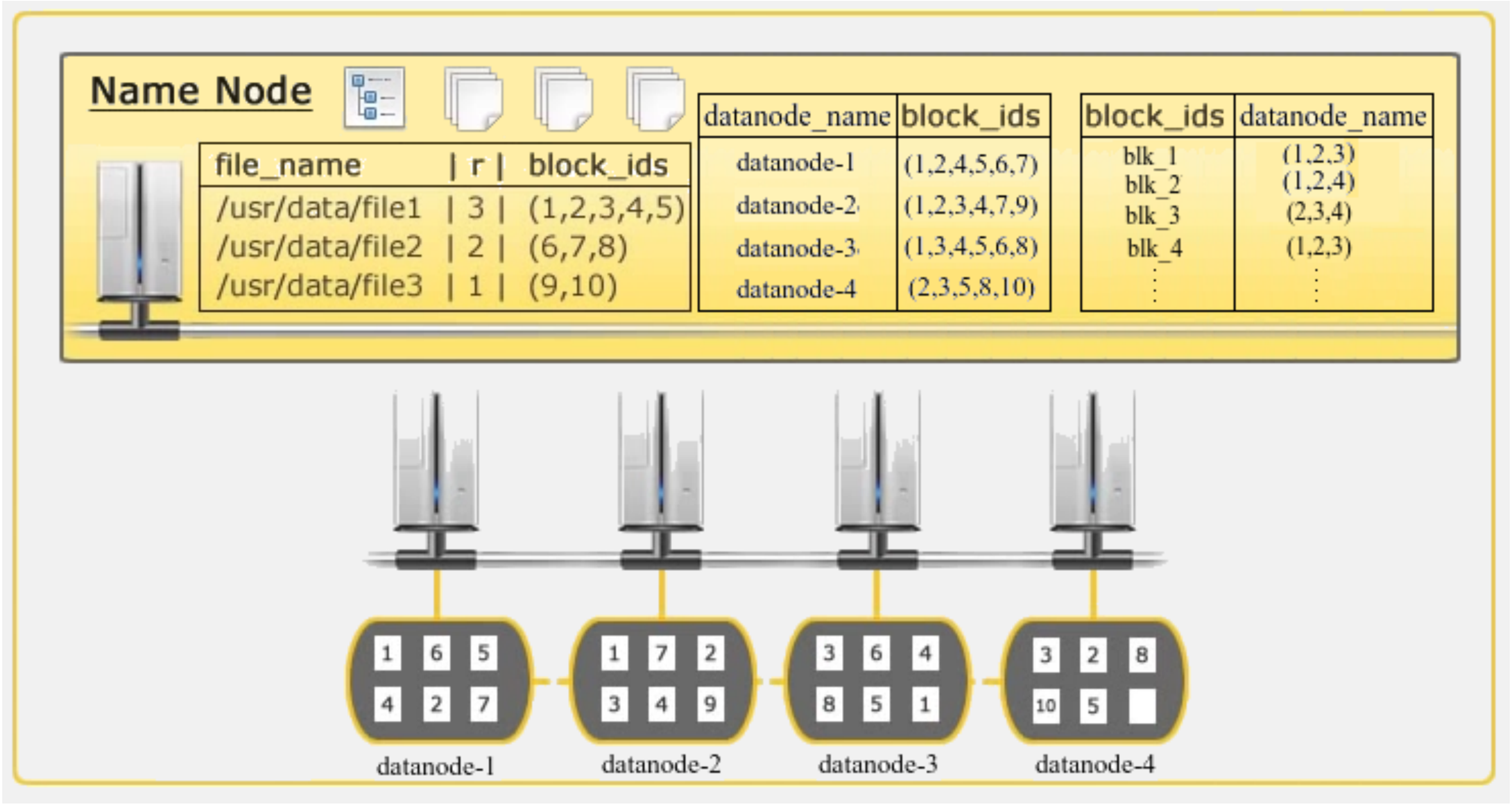
- Runs on top of existing filesystem
- Blocks are 64MB (128MB recommended)
- Single file can be > any single disk
- POSIX based permissions
- Fault tolerant

HDFS: Namenode and Datanode

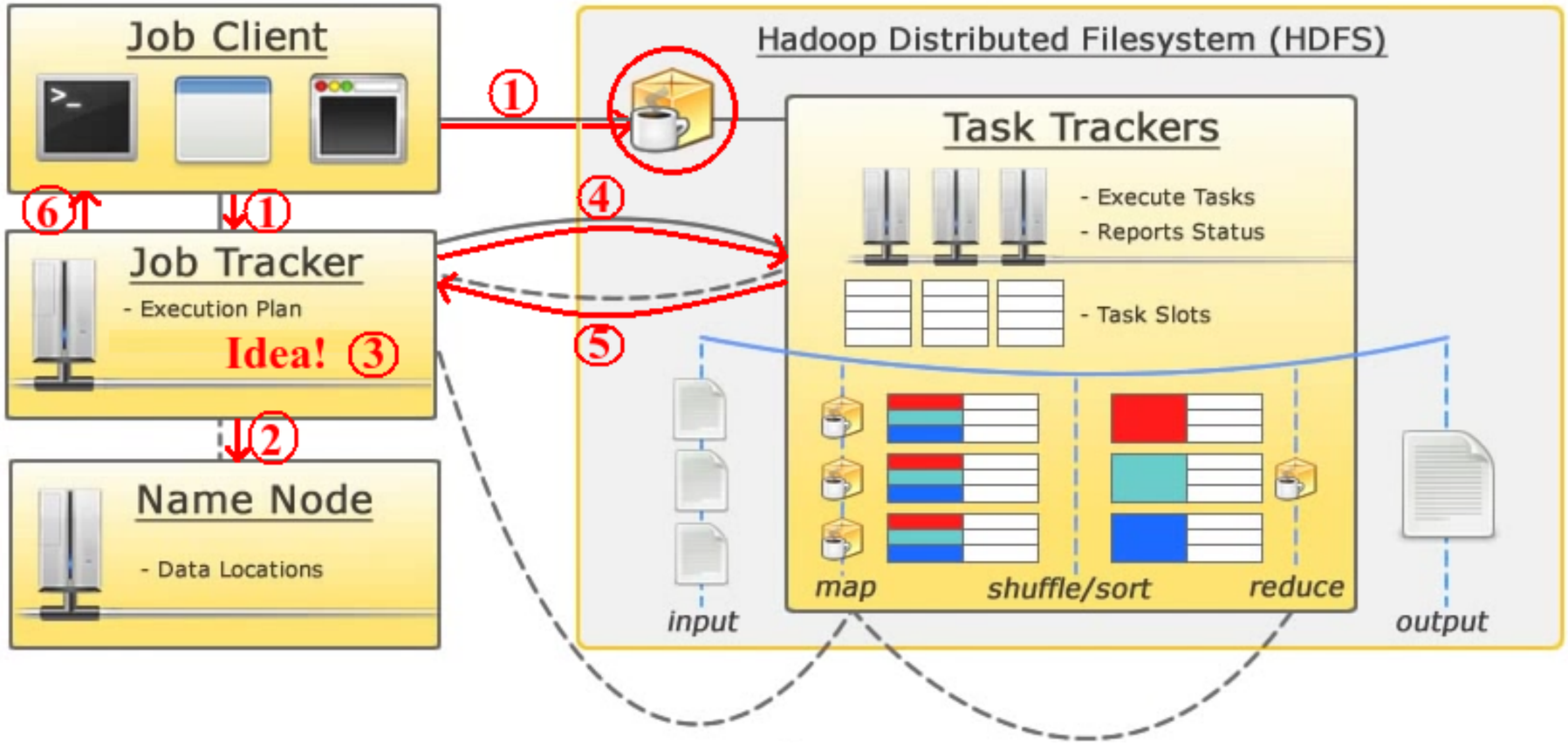
- Namenode
  - Only one per Hadoop Cluster
  - Manages the filesystem namespace
  - The filesystem tree
  - An edit log
  - For each block block i, the datanode(s) in which block i is saved
  - All the blocks residing in each datanode
- Secondary Namenode
  - Backup namenode

- Datanode
  - Many per Hadoop cluster
  - Controls block operations
  - Physically puts the block in the nodes
  - Does the physical replication

HDFS: an example



MapReduce: JobTracker and TaskTracker



1. JobClient submits job to JobTracker; Binary copied into HDFS
2. JobTracker talks to Namenode
3. JobTracker creates execution plan
4. JobTracker submits work to TaskTrackers
5. TaskTrackers report progress via heartbeat
6. JobTracker updates status

Fault Tolerance

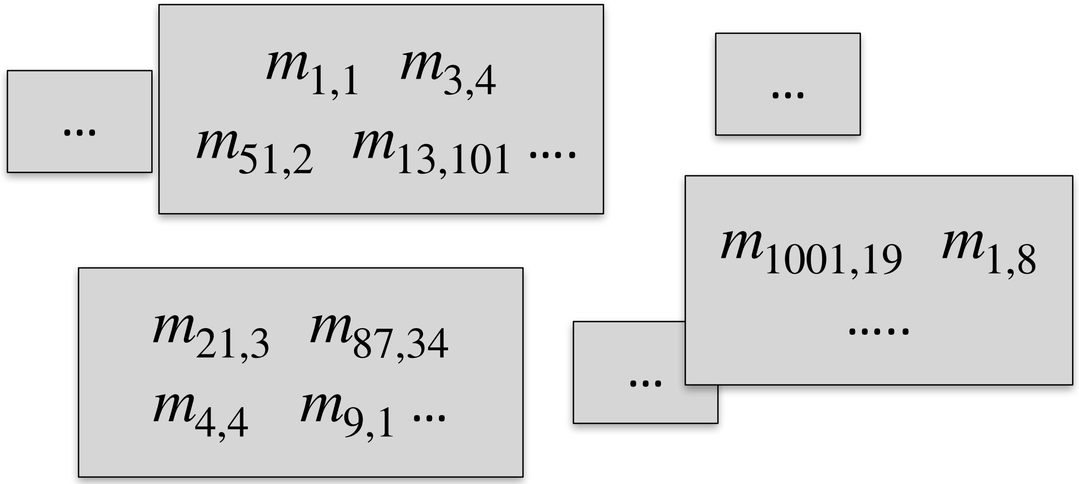
- If the master fails
  - MapReduce would fail, have to restart the entire job
- A map worker node fails
  - Master detects (periodic ping would timeout)
  - All the map tasks for this node have to be restarted
    - Even if the map tasks were done, the output *were* at the node
- A reduce worker fails
  - Master sets the status of its currently executing reduce tasks to idle
  - Reschedule these tasks on another reduce worker

Some algorithms using MapReduce

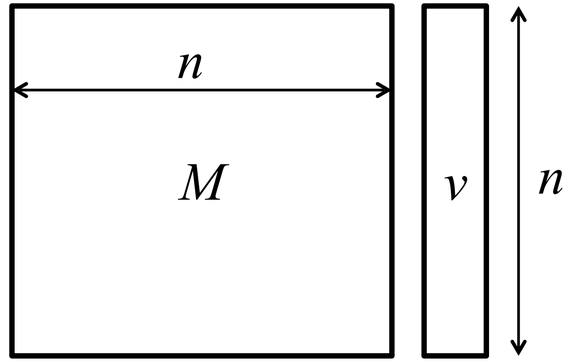
Using MapReduce

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$$Mv = (x_i)$$

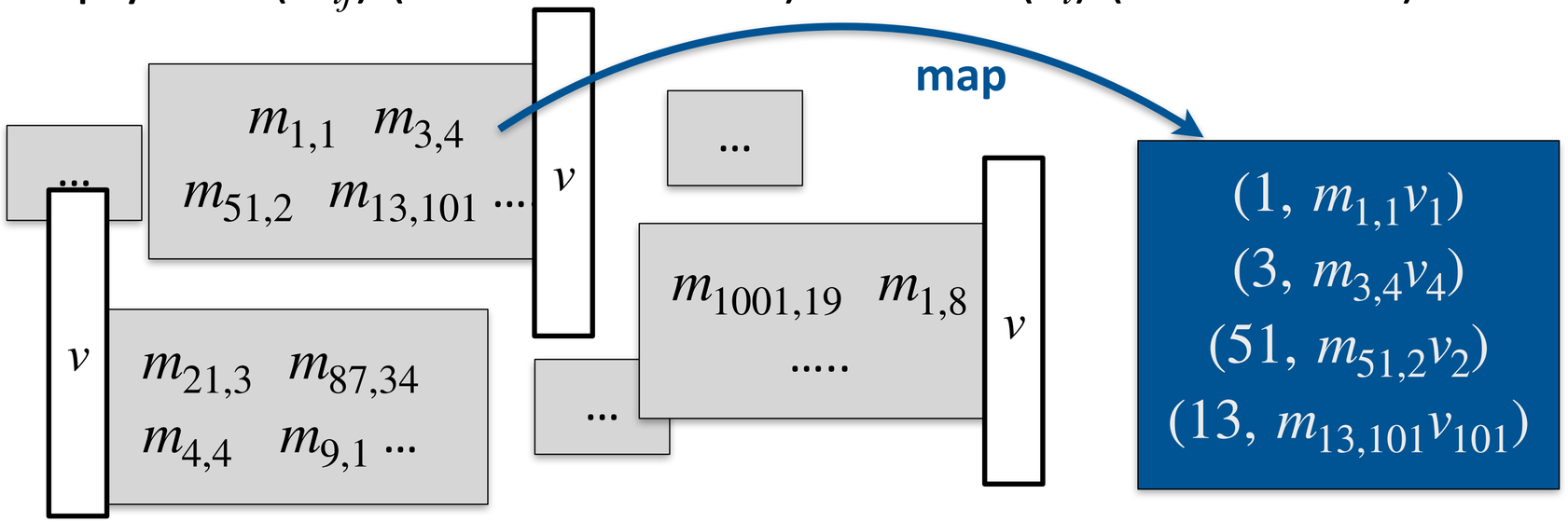


$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

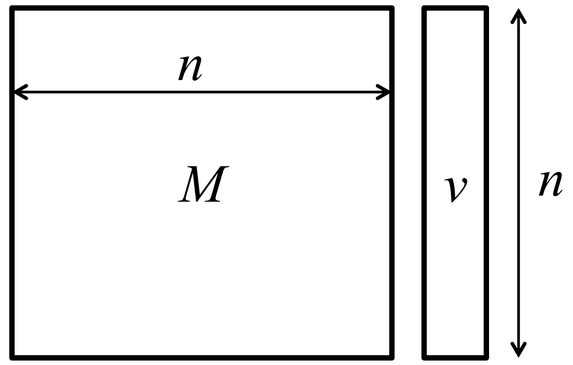
- If $n = 1000$ (small enough), no need of MapReduce!
- When n is very large, how are the entries stored (possibly in a distributed filesystem)?
  - May not be in an order convenient for making any assumption
  - We will not assume anything about the arrangement of the entries
  - Approach: the map should consider each m_{ij}, one at a time

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$$Mv = (x_i)$$



$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Case 1: Large n, M does not fit into main memory, but v does

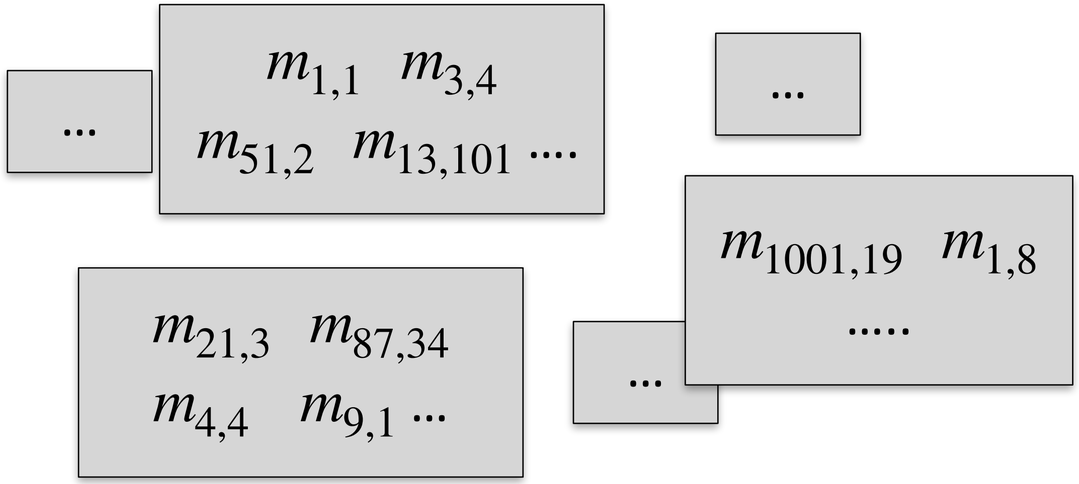
- Since v fits into main memory, all of v can be copied to every map worker
- ***Map***: for each matrix element m_{ij}, emit key value pair $(i,\ m_{ij}v_j)$
- *Shuffle*: groups all $m_{ij}v_j$ values together for the same i
- ***Reduce***: sum $m_{ij}v_j$ for all j for the same i

Need all $m_{ij}v_j$ values together for the same i

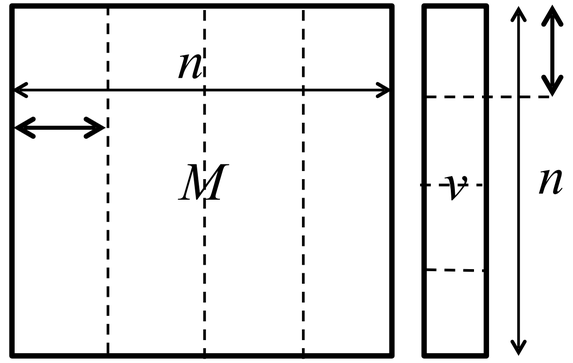
So, i needs to be the **key**

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$$Mv = (x_i)$$



$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Case 2: Very large n, even v does not fit into main memory of the nodes

- For every map, many accesses to disk (for parts of v) required!
- Solution:
  - Partition v so that each partition of v fits into memory
  - Take dot product of one partition of v and the corresponding partition of M
  - *Map* and *reduce* same as before, but several times to complete the computation

Relational Algebra

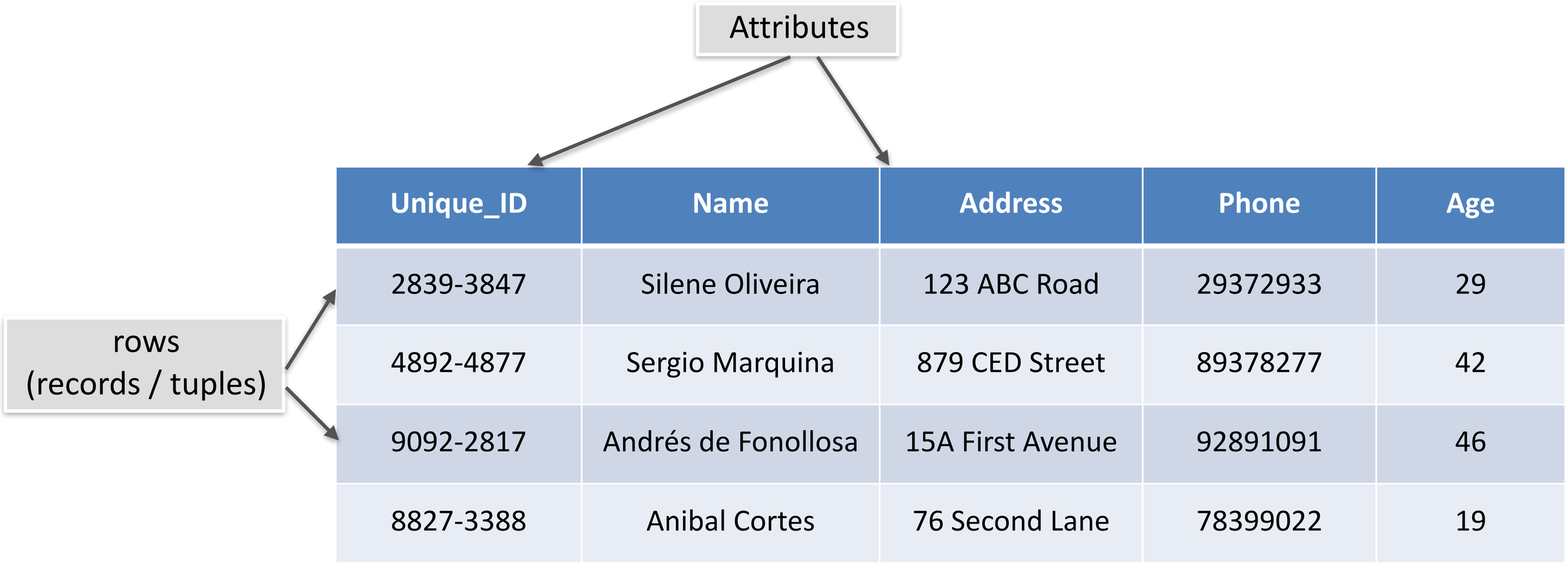
- Relation $R(A_1, A_3, \ldots, A_n)$ is a relation with attributes A_i
- Schema: set of attributes
- Operations
  - Selection on condition C: apply C on each tuple in R, output only those which satisfy C
  - Projection on a subset S of attributes: output the components for the attributes in S
  - Union, Intersection, Join…

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

Example: Links between URLs

URL1	*URL2*
url1	url2
url2	url1
url3	url5
url1	url3

Example: A relation or table **people_data**



Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Selection: example

select * from people_data where **age < 30**

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Projection: example

select **name**, **phone** from people_data

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Name	Phone
Silene Oliveira	29372933
Sergio Marquina	89378277
Andrés de Fonollosa	92891091
Anibal Cortes	78399022

(Inner) Join: example

people_data

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29
4892-4877	Sergio Marquina	879 CED Street	**89378277**	42
9092-2817	Andrés de Fonollosa	15A First Avenue	**92891091**	46
8827-3388	Anibal Cortes	76 Second Lane	**78399022**	19

car_data

Car_number	Phone
XBT 2301	**29372933**
BGA 2891	**78399022**
NMAH 938	**29372933**
MGB 7829	**68279482**

people_data **natural join** car_data

Unique_ID	Name	Address	Phone	Age	Car_number
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29	XBT 2301
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29	NMAH 938
8827-3388	Anibal Cortes	76 Second Lane	**78399022**	19	BGA 2891

Selection using MapReduce

- SQL: select * from R where C
- Trivial using MapReduce
- Map: For each tuple t in R, test if t satisfies condition C. If so, produce the key-value pair $(t, 1)$.
- Reduce: The identity function. It simply passes each key t to the output.

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

Union using MapReduce

- Union of two relations R and S
- Suppose R and S have the same schema
- Map tasks are generated from chunks of both R and S
- Map: For each tuple t, produce the key-value pair $(t, 1)$
- Reduce: Only need to remove duplicates
  - For all key t, there would be either one (if the tuple is present in one relation) or two values (if the tuple is present in both)
  - Output t in either case

R

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

S

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyw	1	false
xyz	def	1	false
bcd	def	2	true

Natural join using MapReduce

- Join $R(A, B)$ with $S(B, C)$ on attribute B
- Map:
  - For each tuple $t = (a,b)$ of R, emit key value pair $(b, (R, a))$
  - For each tuple $t = (b,c)$ of S, emit key value pair $(b, (S, c))$
- Reduce:
  - Each key b would be associated with a list of values that are of the form (R, a) or (S, c)
  - Construct all pairs consisting of one with first component R and the other with first component S, say (R,a) and (S,c). The output from this key and value list is a sequence of key-value pairs

R

A	*B*
x	m
y	n
z	p
w	q

S

B	*C*
m	1
p	3
q	4
r	7

$R \bowtie S$

A	*B*	*C*
x	m	1
z	p	3
w	q	4

Grouping and Aggregation using MapReduce

- Group and aggregate on a relation $R(A, B)$ using aggregation function $\gamma(B)$, group by A
- Map:
  - Grouping comes for free with MapReduce, so the attribute to "group by" needs to be the key
  - Map is essentially the identity function
  - For each tuple $t = (a,b)$ of R, emit key value pair (a, b)
- Reduce:
  - Reduce does the aggregation part
  - For the group $\{(a, b_1), \ldots, (a, b_m)\}$ represented by a key a, apply γ to obtain $b_a = \gamma(b_1, \cdots, b_m)$
  - Output (a, b_a)

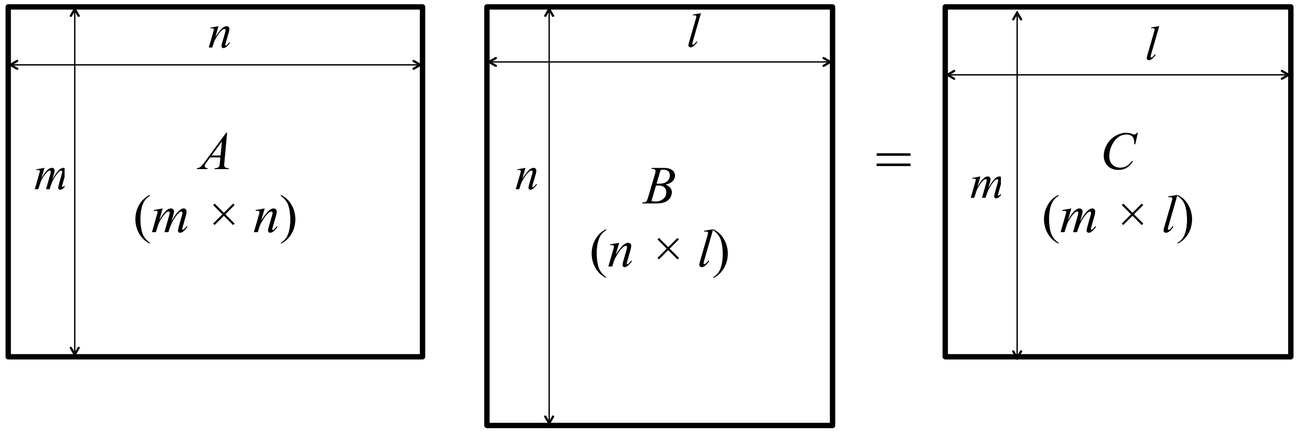
R

A	B
x	2
y	1
z	4
z	1
x	5

```
select A, sum(B)
from R group by A;
```

A	$sum(B)$
x	7
y	1
z	5

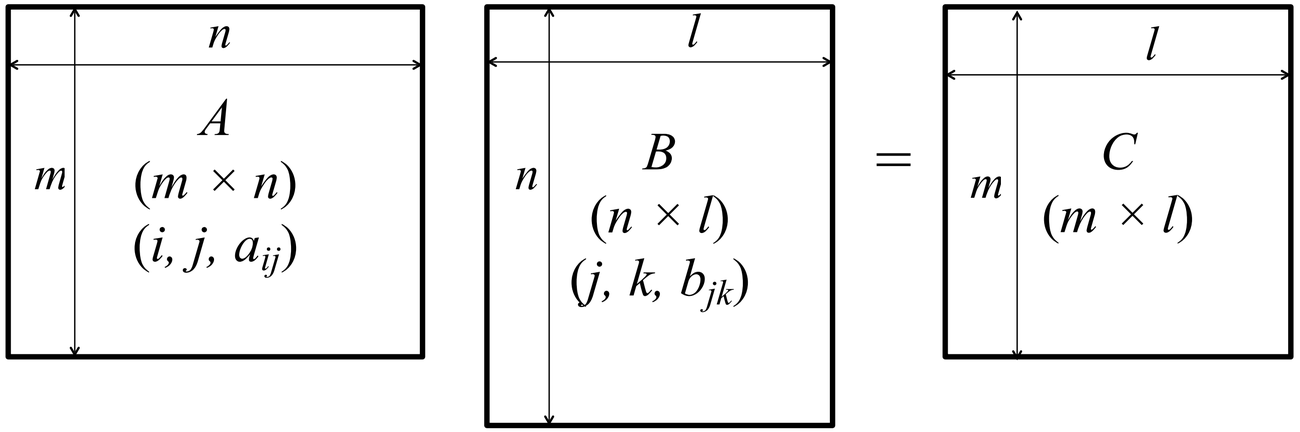
Matrix multiplication using MapReduce



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

- Think of a matrix as a relation with three attributes
- For example matrix A is represented by the relation $A(I, J, V)$
  - For every non-zero entry (i, j, a_{ij}), the row number is the value of I, column number is the value of J, the entry is the value in V
  - Also advantage: usually most large matrices would be sparse, the relation would have less number of entries
- The product is *analogous to* a natural join followed by a grouping with aggregation

Matrix multiplication using MapReduce



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

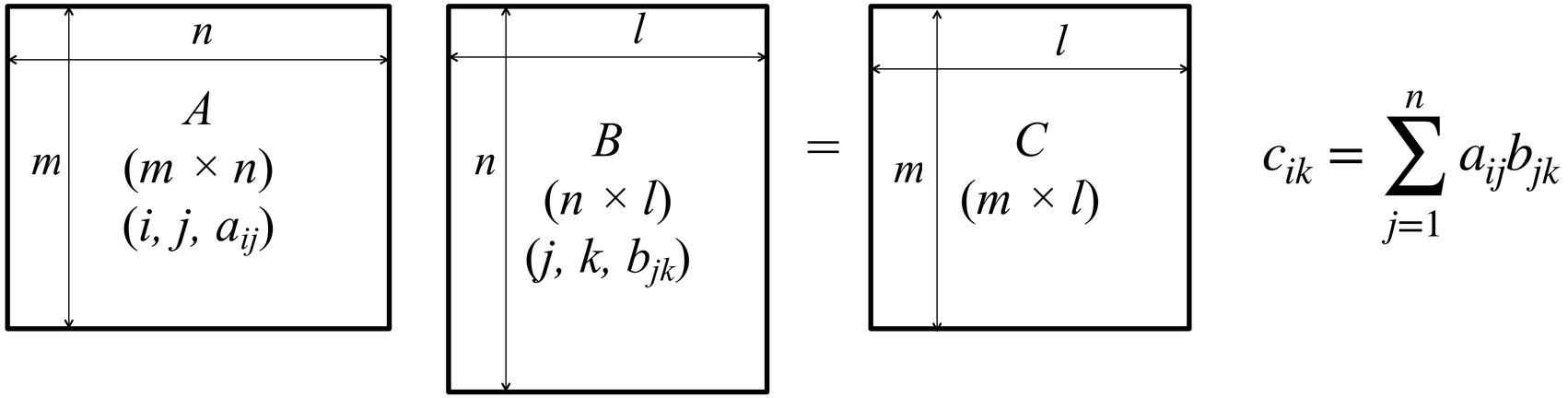
- Natural join of (I, J, V) and (J, K, W) produces tuples $(i, j, k, a_{ij}, b_{jk})$
- Map:
  - For every (i, j, a_{ij}), emit key value pair $(j, (A, i, a_{ij}))$
  - For every (j, k, b_{jk}), emit key value pair$(j, (B, k, b_{jk}))$
- Reduce:

  for each key j

    for each value (A, i, a_{ij}) and (B, k, b_{jk})

      produce a key value pair $((i, k), (a_{ij} b_{jk}))$ [could also output $i, k, a_{ij} b_{jk}$, but doing some work in advance]

Matrix multiplication using MapReduce



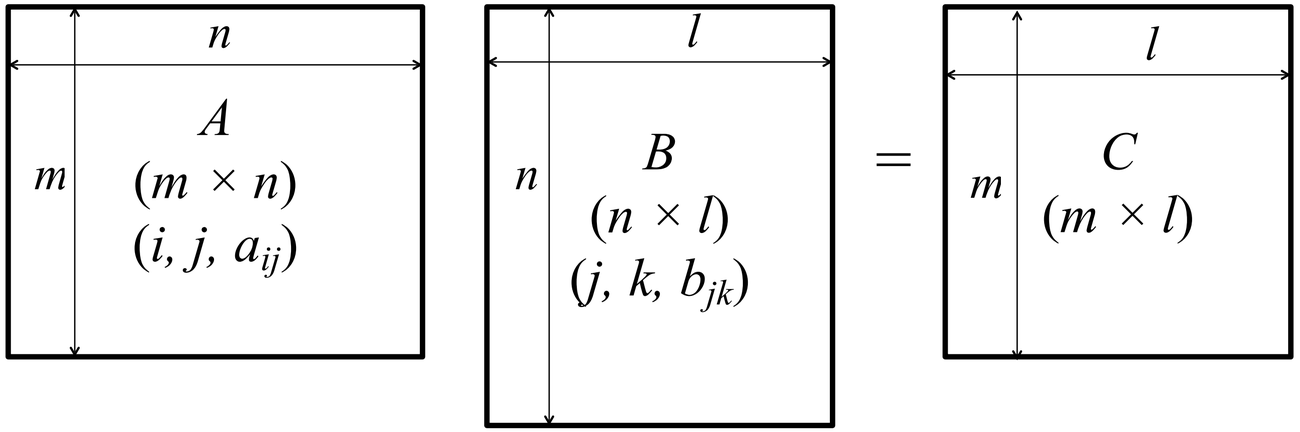
$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

- First MapReduce process has produced key value pairs $((i,k),(a_{ij}b_{jk}))$
- Another MapReduce process to group and aggregate
  - Map: identity, just emit the key value pair $((i,k),(a_{ij}b_{jk}))$
  - Reduce:

    for each key (i,k)

    produce the sum of the all the values for the key: $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$

Matrix multiplication using MapReduce: Method 2



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

- A method with one MapReduce step
- Map:
  - For every (i, j, a_{ij}), emit for all $k = 1, \ldots, l$, the key value $((i,k), (A, j, a_{ij}))$
  - For every (j, k, b_{jk}), emit for all $i = 1, \ldots, m$, the key value $((i,k), (B, j, b_{jk}))$
- Reduce:

  for each key (i,k)

  sort values (A, j, a_{ij}) and (B, j, b_{jk}) by j to group them by j

  for each j multiply a_{ij} and b_{jk}

  sum the products for the key (i,k) to produce $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$

Idea: the final keys have to be ik pairs, so create all possible ik combinations as keys upfront

May not fit in main memory and may have to perform expensive external memory sort. If so, method 1 is more practical.

Combiners

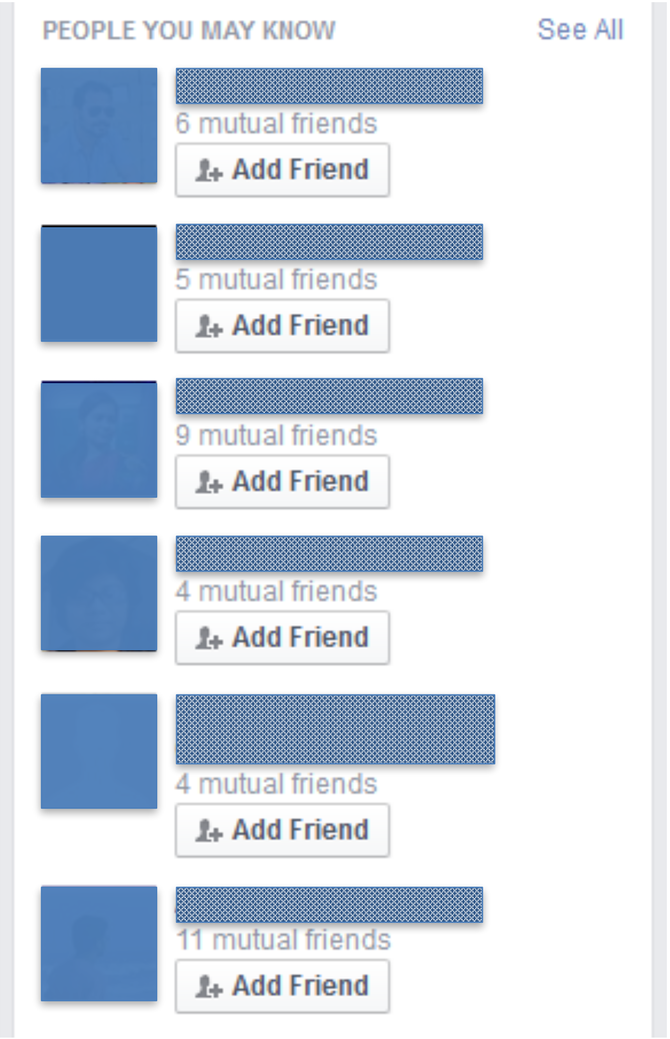
- If the aggregation operator in the reduce algorithm is associative and commutative
  - Associative: $(a * b) * c = a * (b * c)$
  - Commutative: $(a * b) = (b * a)$
  - It does not matter in which order we aggregate
- We can do some work for reduce in the map step
- Example: the word count problem
  - Instead of emitting $(w,1)$ for each word
  - We can emit (w, n) for each document or each map task where n is the number of times the word w appears in the whole input chunk

A problem: find people you may know

A Social Network

- Task: suggest friends
- There are several factors to consider
- A very important one is by number of mutual friends
- How to compute a list of suggestions?
- Consider a simple criterion:

If X and Y have at least 3 mutual friends then Y is a suggested friend of X



The Data

- For every member, there is a list of friends

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

N members.
Every member has n friends.
$N \gg n$.

Naïve approach:

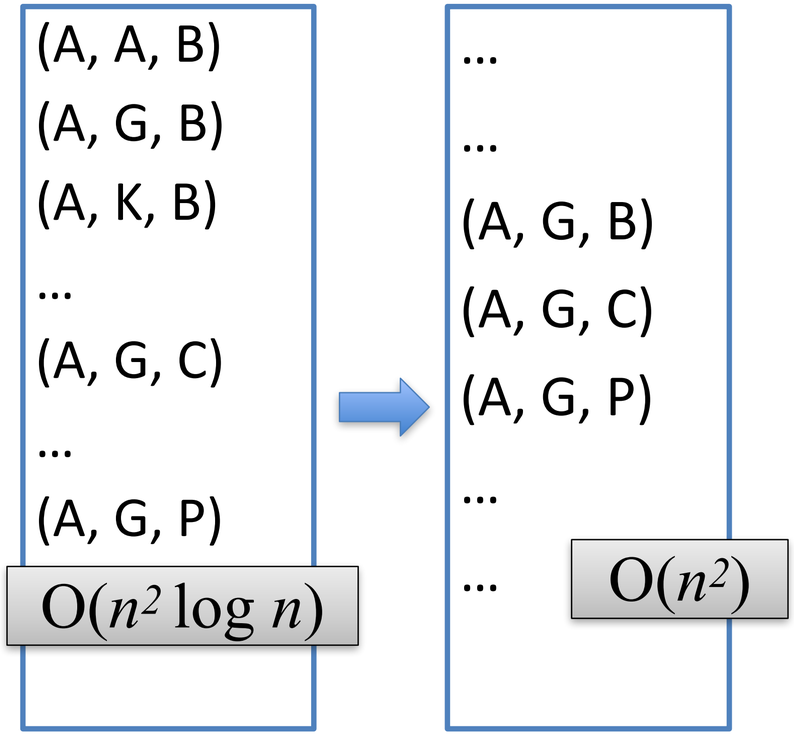
- For every friend X of A $O(n)$
  - For every friend Y of X $O(n)$
    - Compute number of mutual friends between Y and A
    - Intersect the friend-list of Y and friend-list of A $O(n)$

$O(n^3)$ per member. Total $O(N\,n^3)$!!

A better method

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

- For each friend X of A $O(n)$
  - For each friend Y of X $O(n)$
    - Write: (A, Y, X) $O(n^2)$
- What do we get?
- Now sort by first two entries
  - All the (A,G)'s are together
- For each (A, Y), count how many times it occurs
  - Number of mutual friends between A and Y



$O(n^2 \log n)$ per member

Challenges

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

- For each friend X of A
  - For each friend Y of X
    - Write: (A, Y, X)
- Now sort by first two entries
  - All the (A,G)'s are together
- For each (A, Y), count how many times it occurs

Data may be too huge, distributed

Friendlist of A in one part of storage
Friendlist of friend of A in another part

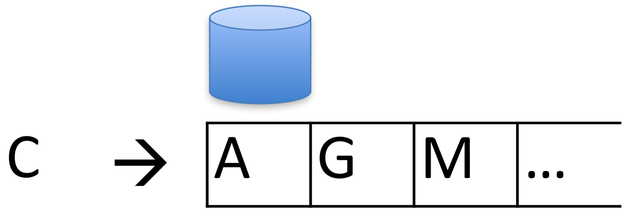
(A, A, B)
(A, G, B)
(A, K, B)
…
(A, G, C)
…
(A, G, P)
…

…
…
(A, G, B)
(A, G, C)
(A, G, P)
…
…

Using MapReduce

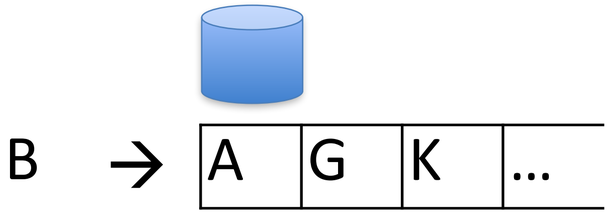
Map:

- For each member X, fetch his/her friendlist Y_1, Y_2, … , Y_n
  - For each pair (Y_1, Y_2), emit the key value pair {(Y_1, Y_2), X}



Map

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
…	…



Map

Key	Value
(A,G)	B
(A,K)	B
(G,K)	B
…	…

Using MapReduce

Reduce:

- For each Key = (X,Y), determine Count(X,Y) = # of mutual friends

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
…	…

Key	Value
(A,G)	B
(A,K)	B
(G,K)	B
…	…

Shuffle and sort

Key	Value
(A,G)	B
(A,G)	C
(A,G)	P
…	…

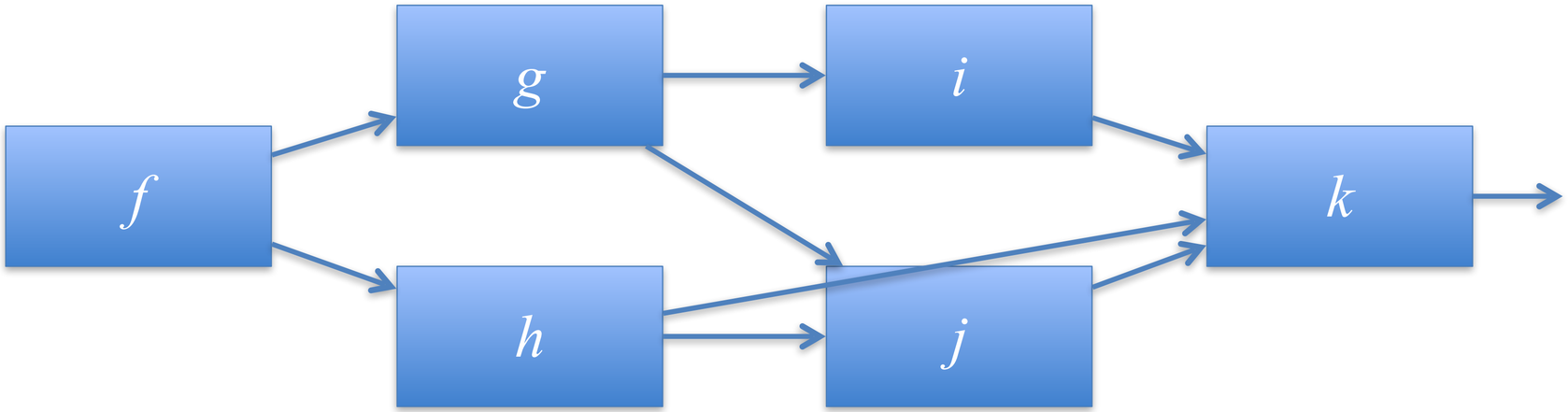
Key	Value
(S,T)	Q
(S,T)	V
(S,T)	P
…	…

Reduce → Count(A,G)

Reduce → Count(S,T)

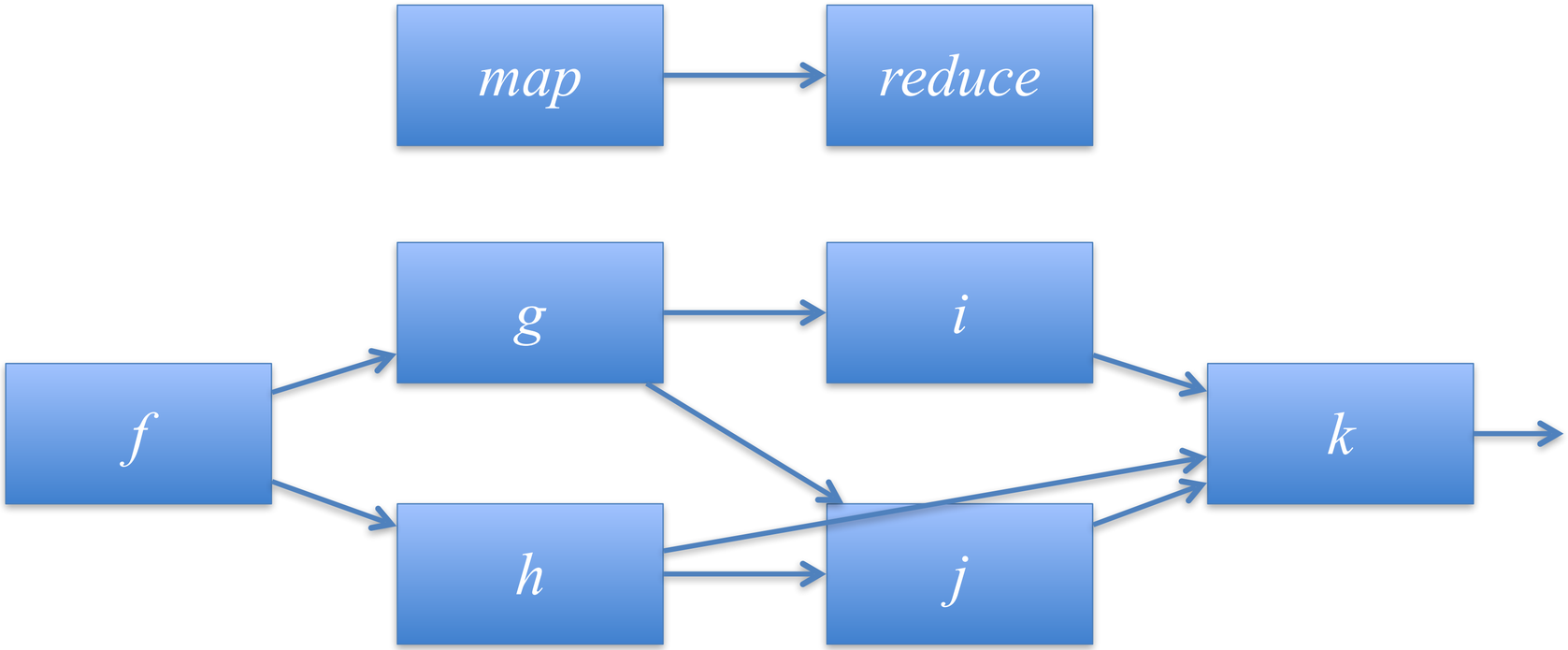
Extensions of MapReduce

- MapReduce: two-step workflow
- Extension: any collection of functions: acyclic graph representing the workflow



- Each function of the workflow can be executed by many tasks
- A master controller divides the work among the tasks
- Each task's output goes to the successor as input
- Two experimental systems: Clustera (University of Wisconsin), Hyracks (University of California at Irvine)
- Similarly, recursive use of MapReduce

Cost of MapReduce based algorithms



- Computation in the nodes → Computation cost
- Transferring of data through network → Communication cost
- Communication cost is most often the bottleneck. Why?

Why communication cost?

- The algorithm executed by each task is usually very simple, often linear in the size of its input
- Network speed within cluster $\sim$ 1Gigabit/s
  - But processor executes simple map or reduce tasks even faster
  - In a cluster several network transfers may be required at the same time $\rightarrow$ overload
- Data typically stored in disk
  - Reading the data into main memory may take more time than to process it in map or reduce tasks

Communication cost

- Communication cost of a task = The size of the input to the task
- Why size of the input?
  - Most often size of the input ≥ size of the output
  - Why?
  - Because unless summarized, the output can't be used much
  - If output size is large, then it is the input for another task (counting input size suffices)

History

MapReduce was first popularized as a programming model in 2004 by Jeffery Dean and Sanjay Ghemawat of Google (Dean & Ghemawat, 2004). In their paper, “MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS,” they discussed Google’s approach to collecting and analyzing website data for search optimizations. Google’s proprietary MapReduce system ran on the Google File System (GFS).

MapReduce helped in two key tasks Google needed to perform day in and day out at that time. One is creation of an inverted index for search, and another is computing PageRank.

Apache, the open source organization, began using MapReduce in the “Nutch” project, which is an open source web search engine that still is active today. Hadoop began as a subproject in the Apache Lucern project, which provides text search capabilities across large databases.

In 2006, Doug Cutting, an employee of Yahoo!, designed Hadoop, naming it after his son’s toy elephant. While it was originally a subproject, Cutting released Hadoop as an open source Apache project in 2007. (Hadoop, 2011). In 2008, Hadoop became a top level project at Apache. On July 2008, an experimental 4000 node cluster was created using Hadoop, and in 2009 during a performance test, Hadoop was able to sort a terabyte of data in 17 hours.

https://mindmajix.com/mapreduce/history-and-advantages-of-hadoop-mapreduce-programming

References and acknowledgements

- Primary reference: Mining of Massive Datasets, by Leskovec, Rajaraman and Ullman, Chapter 2
- Source for some slides on Hadoop: Dwaipayan Roy