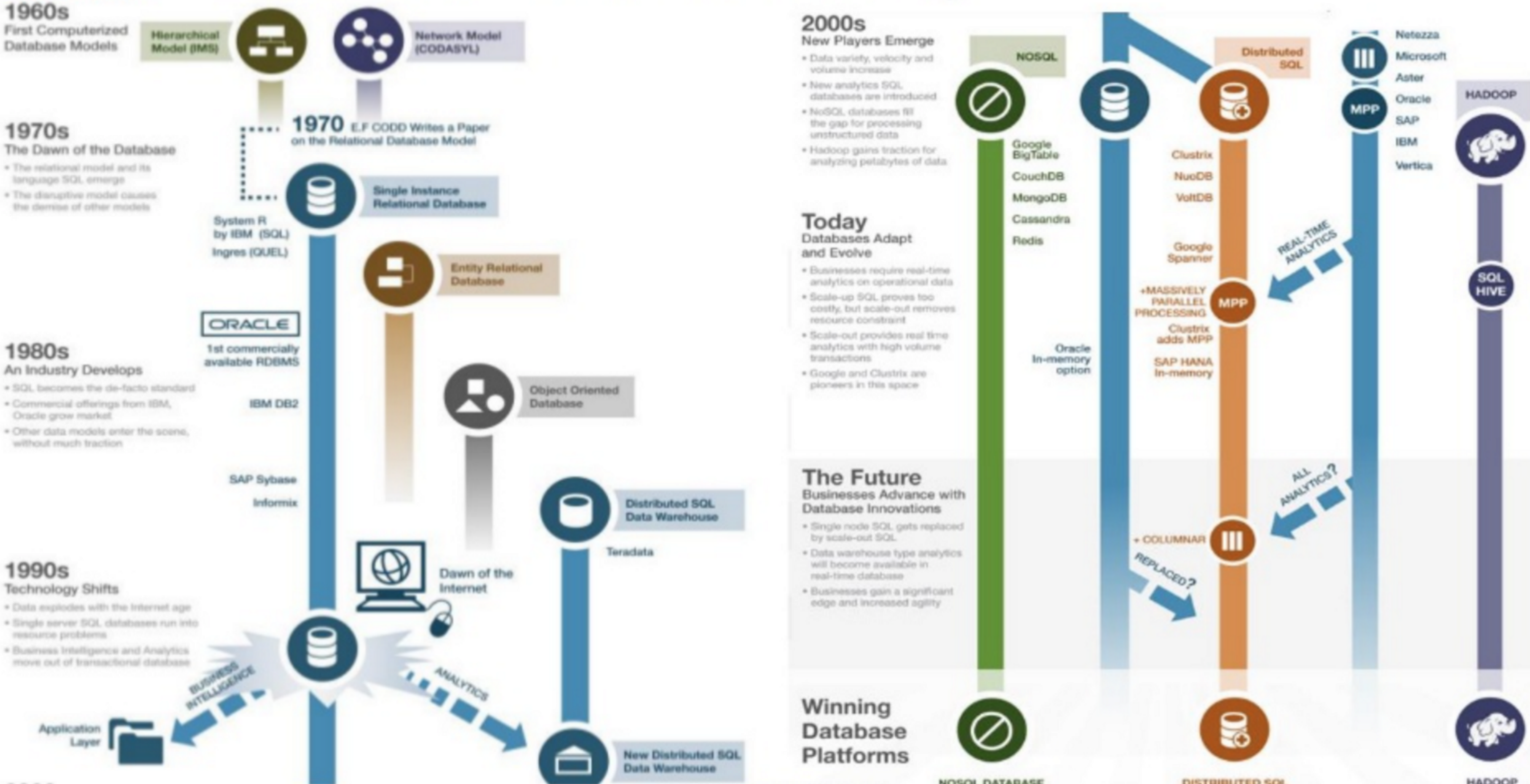
Database Evolution History



Source: Robin Purohit

Source: Robin Purohit

Technologies du Big-Data

- 16 Nov. Map/Reduce 1CM + 2TDTP
- 23 Nov. Map/Reduce 3TD/TP
  - rendu TP 3 décembre
- 30 Nov. Big-Data & Cognitive (IBM) 2CM

  présence obligatoire
- 7 Déc. Big-Data & Cognitive (IBM) 2CM

Map/Reduce

Slides partially collected from
J. Ullman, J. Leskovec and A.Rajarman

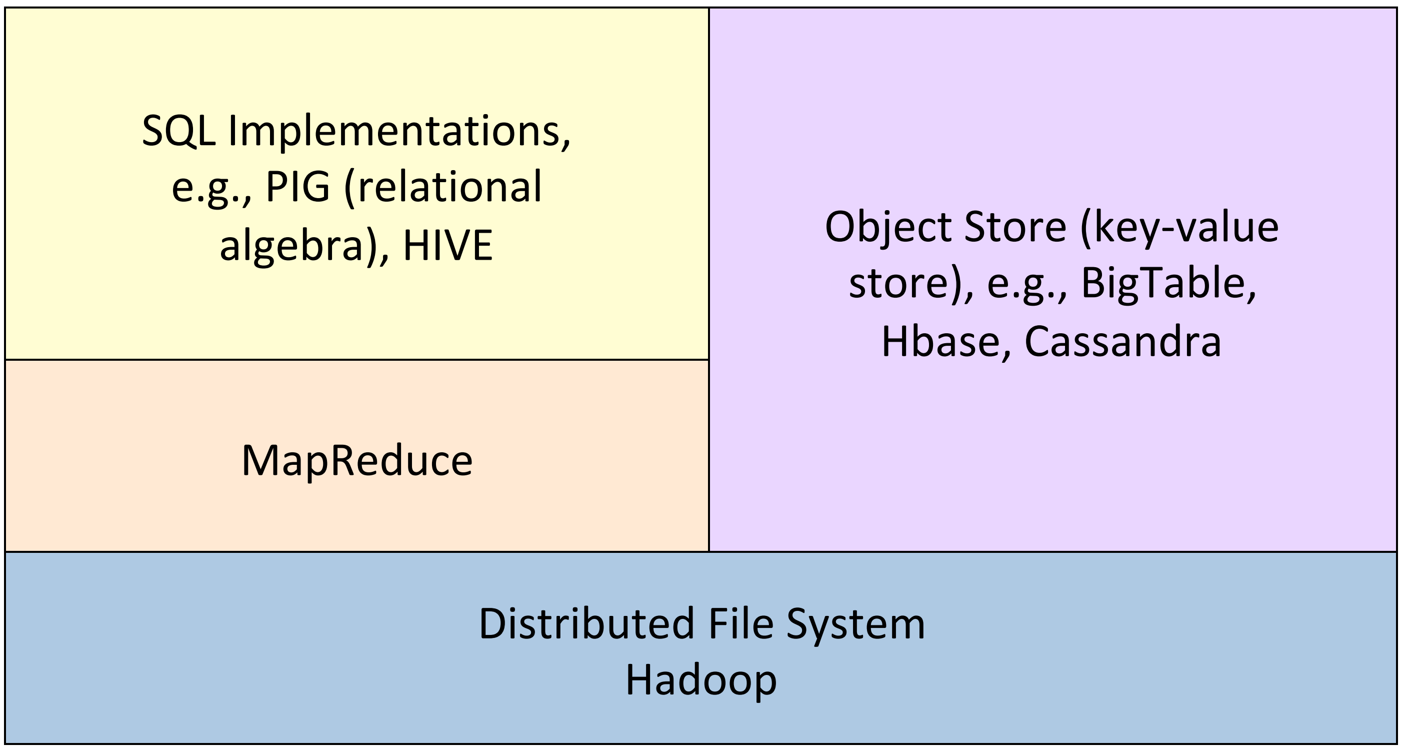
What is Hadoop-Map/Reduce

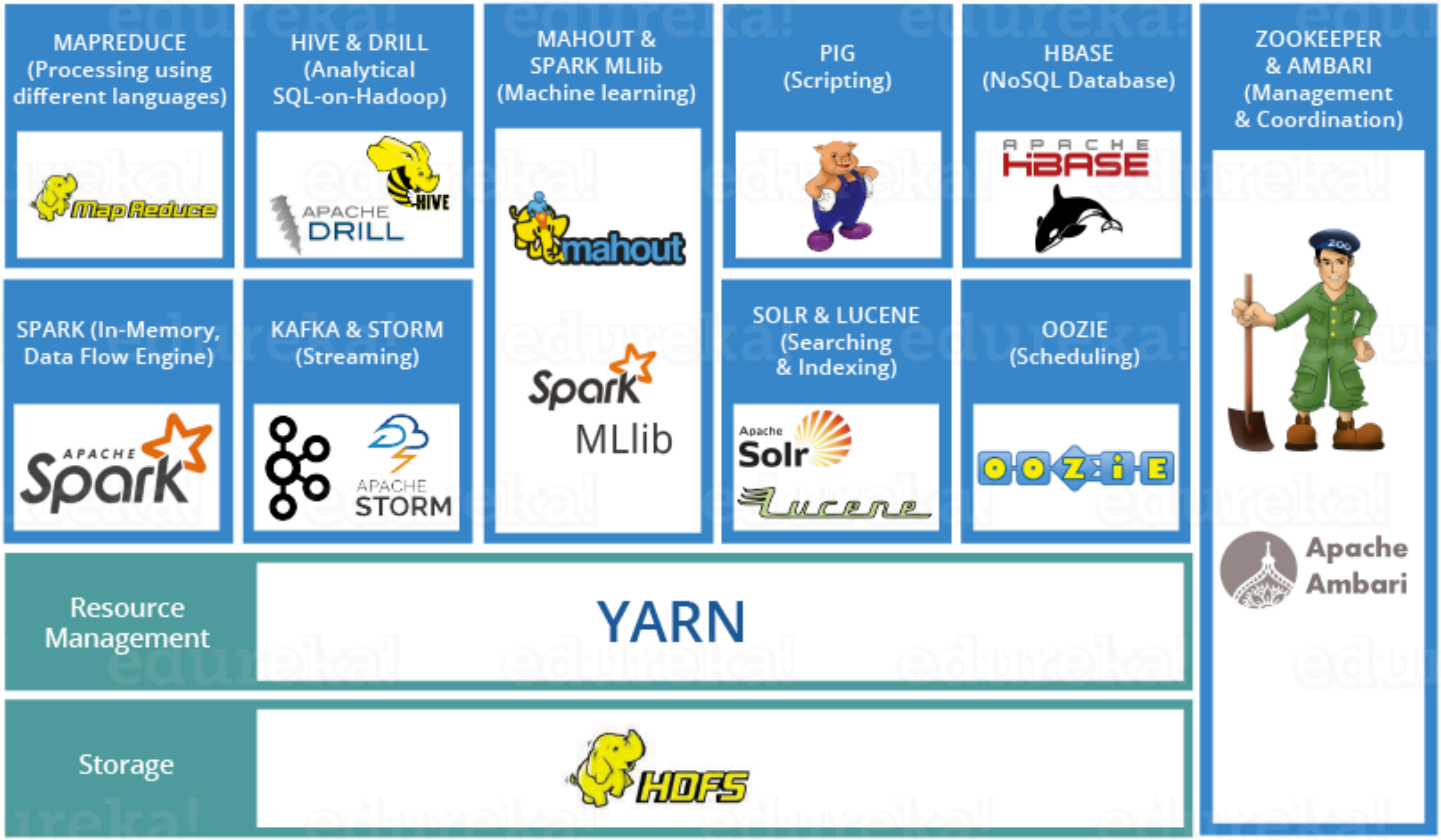
- Google'solution to solve Big Data (Volume) problems by means of massive parallelization

- Parallel databases exist since '90, but M/R processing is much more flexible and easier to deploy

- **Hadoop** : distributed file system (holds the data)
- **Map-Reduce** : programming paradigm (compute)

The New Stack




MAPREDUCE
(Processing using different languages)
Map Reduce
HIVE & DRILL
(Analytical SQL-on-Hadoop)
APACHE DRILL
HIVE
MAHOUT &
SPARK MLlib
(Machine learning)
mahout
Spark
MLlib
PIG
(Scripting)
HBASE
(NoSQL Database)
APACHE HBASE
ZOOKEEPER
& AMBARI
(Management
& Coordination)
ZOO
Apache Ambari
SPARK (In-Memory,
Data Flow Engine)
APACHE Spark
KAFKA & STORM
(Streaming)
APACHE STORM
SOLR & LUCENE
(Searching
& Indexing)
Apache Solr
Lucene
OOZIE
(Scheduling)
OOZIE
Resource Management
YARN
Storage
HDFS

Source : http://www.teradata.com.au/Resources/White-Papers/Hadoop-and-the-Data-Warehouse-When-to-Use-Whi

Requirement	Data Warehouse	Hadoop
Low latency, interactive reports, and OLAP	●	
ANSI 2003 SQL compliance is required	●	
Preprocessing or exploration of raw unstructured data		●
Online archives alternative to tape		●
High-quality cleansed and consistent data	●	
100s to 1000s of concurrent users	●	●*
Discover unknown relationships in the data	●	●
Parallel complex process logic		●
CPU intense analysis	●	●
System, users, and data governance	●	
Many flexible programming languages running in parallel		●
Unrestricted, ungoverned sand box explorations		●
Analysis of provisional data		●
Extensive security and regulatory compliance	●	
Real time data loading and 1 second tactical queries	●	●*

Hadoop-M/R does three things

1. Makes it easier to run distributed computations

2. Makes it easier to write distribute programs

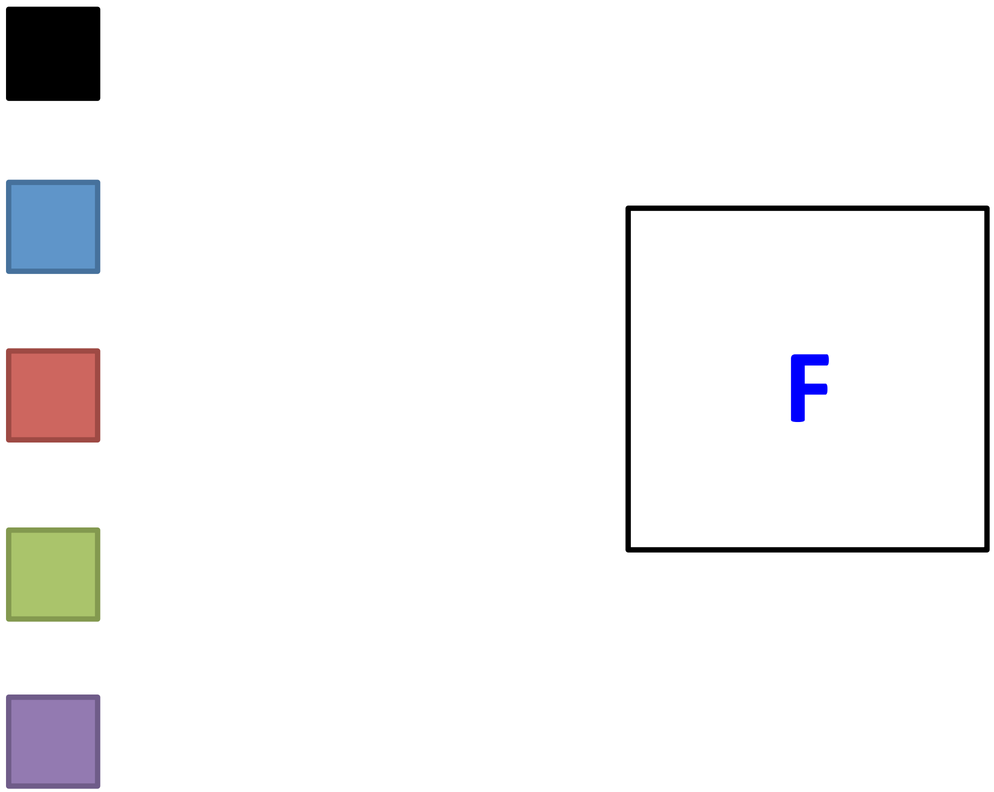
3. Makes it easier to deal with node failure

Hadoop-M/R means three things

1. A model of computing
2. A paradigm of programming
3. A distributed-system architecture

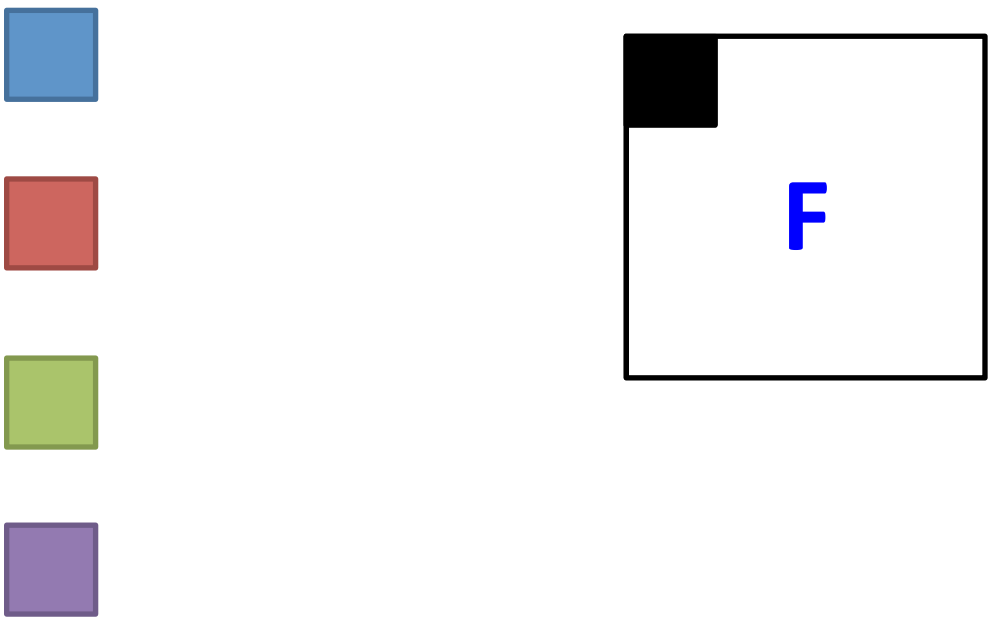
Sequential Computing

- iterate over inputs to compute a given function



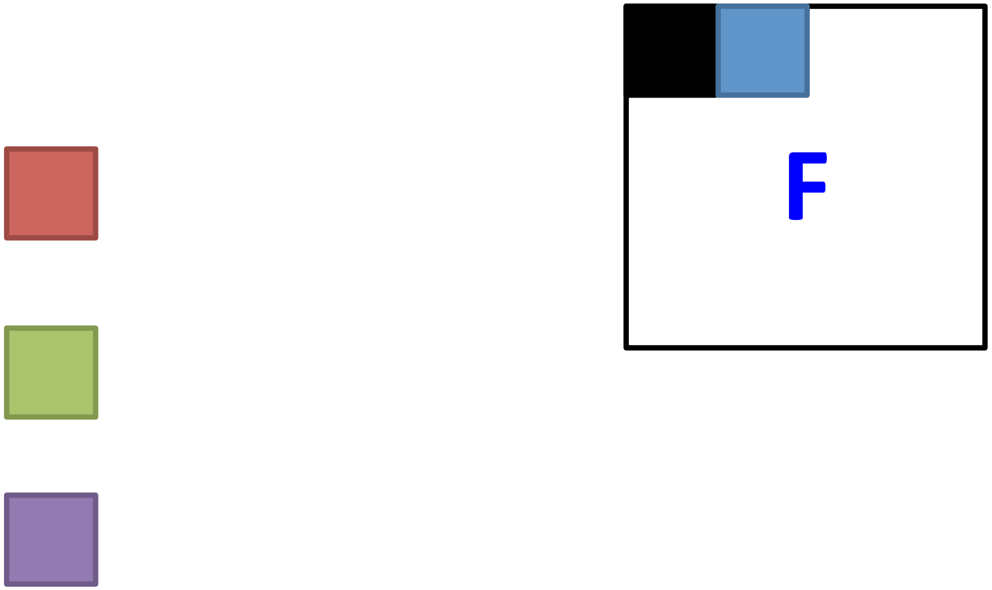
Sequential Computing

- iterate over inputs to compute a given function



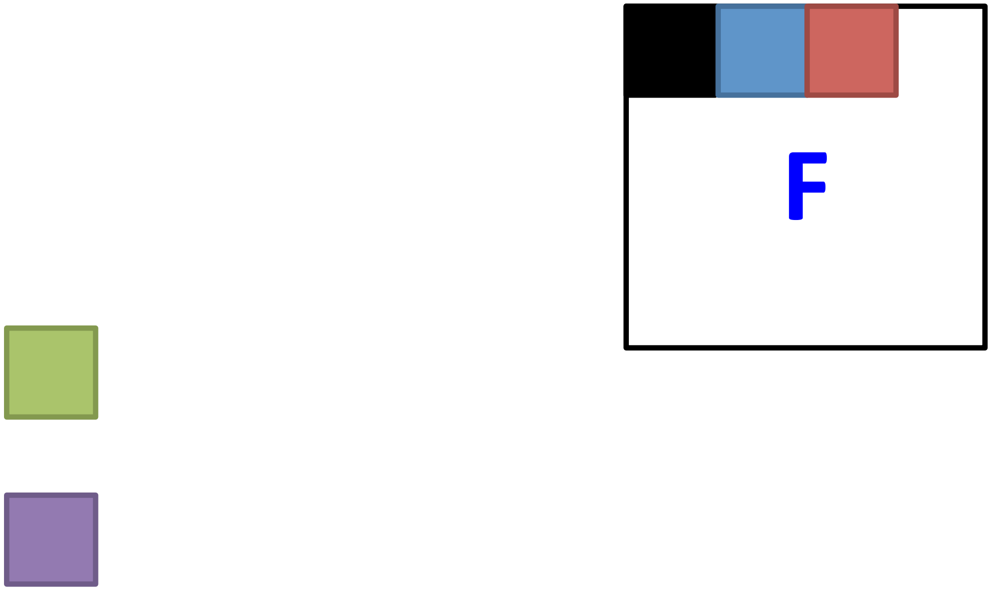
Sequential Computing

- iterate over inputs to compute a given function



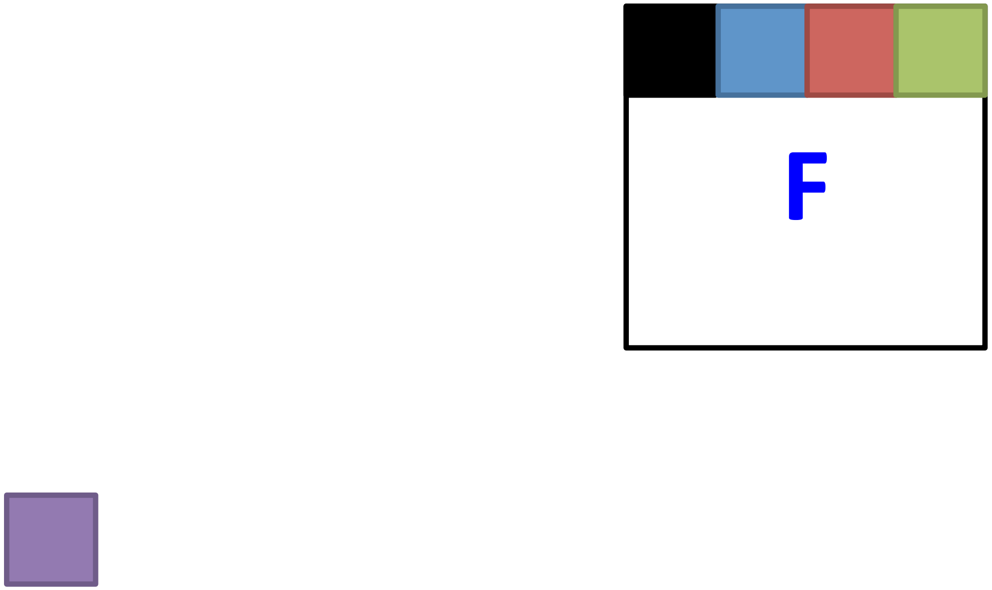
Sequential Computing

- iterate over inputs to compute a given function



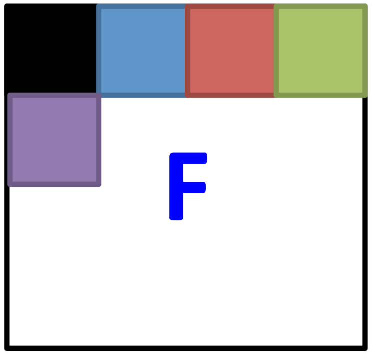
Sequential Computing

- iterate over inputs to compute a given function



Sequential Computing

- iterate over inputs to compute a given function



note : there is a lot of sequential computing inside parallel programs

Just before we introduce M/R

- How do you expect it to work ?

- Maybe...
  - read all inputs sequentially
  - then distribute the evaluation of a given function

- Or ...
  - read all inputs in parallel
  - and distribute the evaluation of a given function

M/R Computing Model

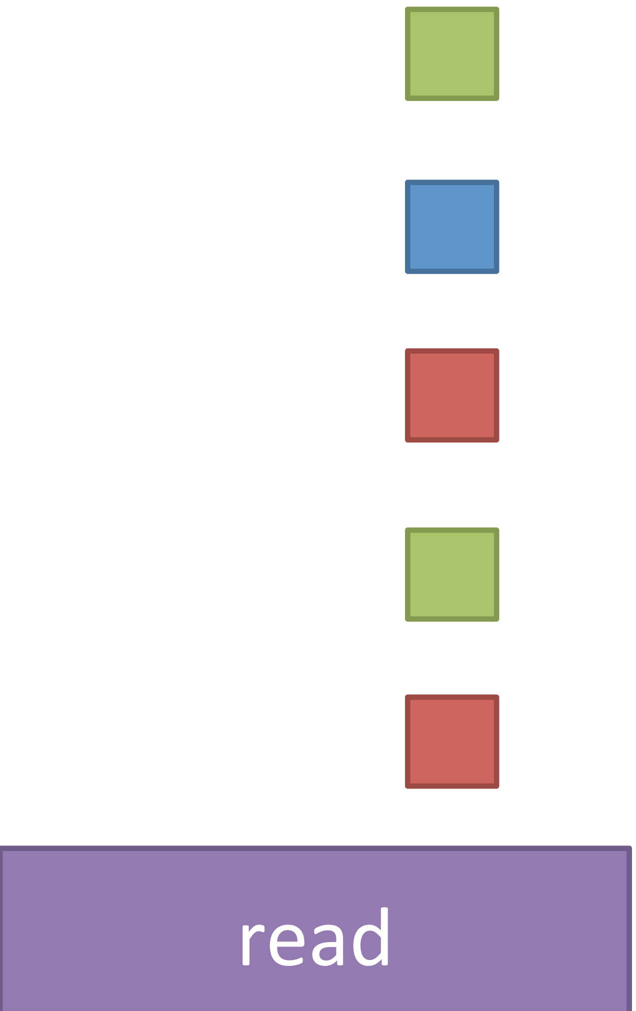
1. **Read** the inputs

2. **Regroup**-the inputs

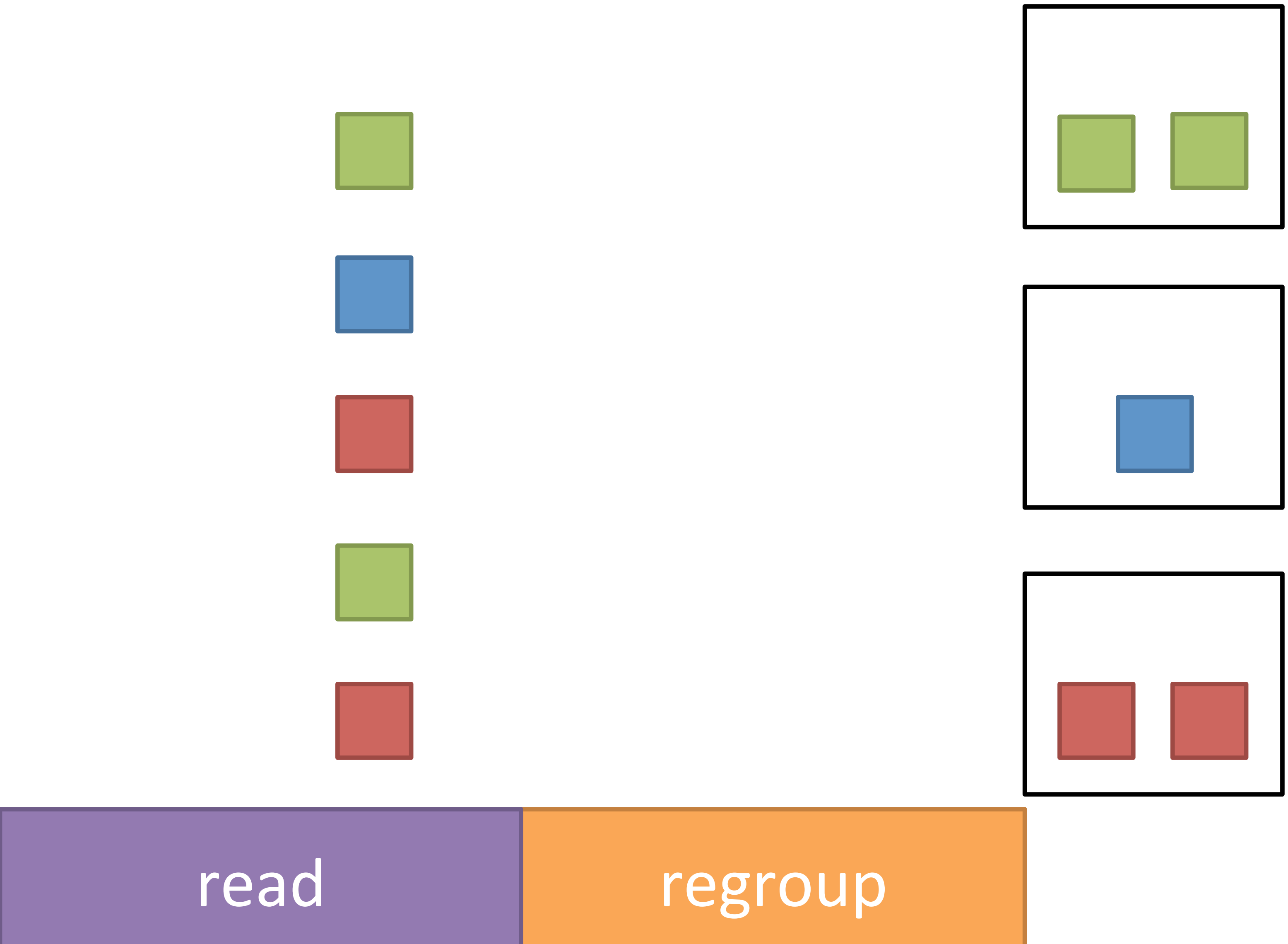
3. **Evaluate** a function on the regrouped inputs

…and all of these can be done in parallel !

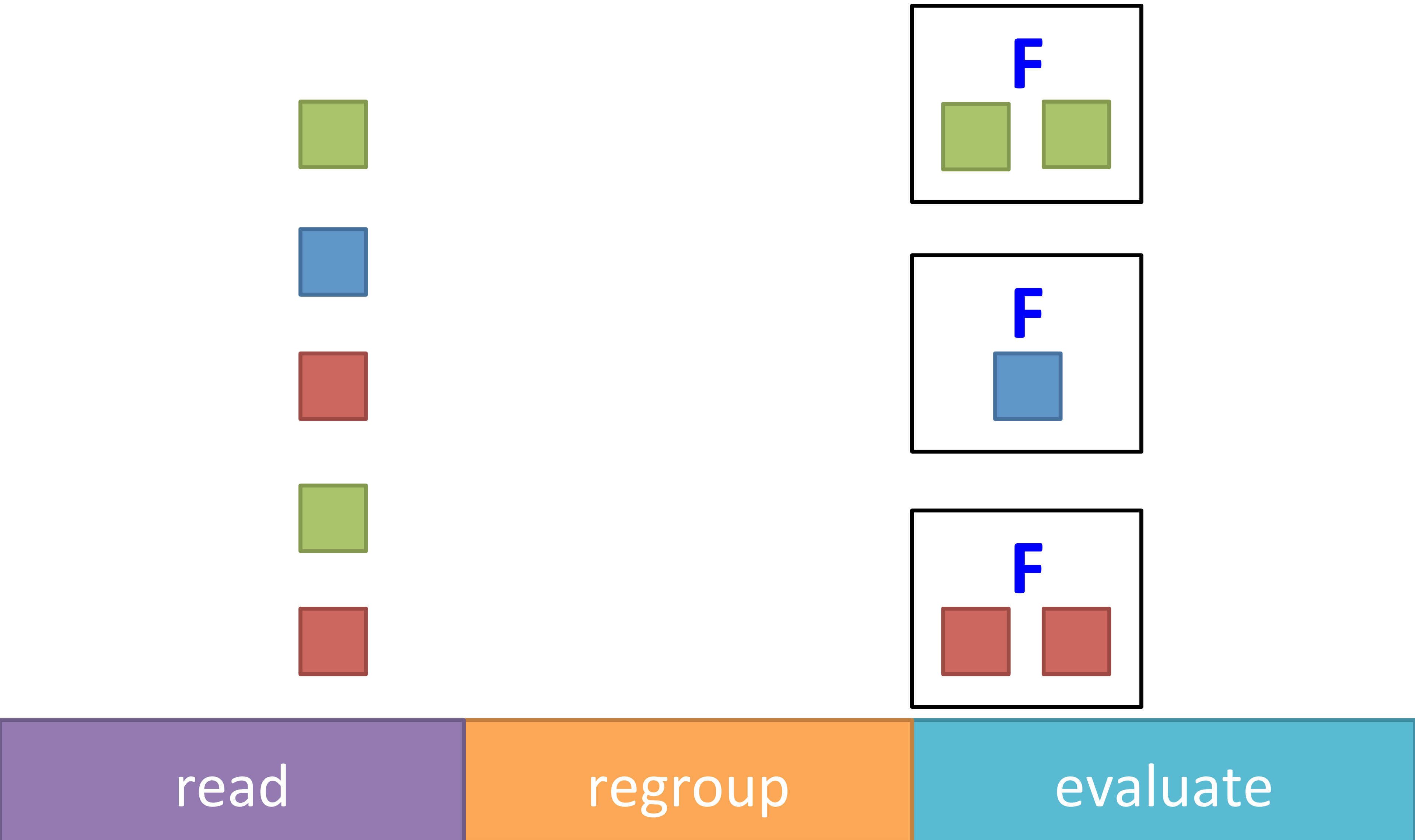
M/R Computing Model



M/R Computing Model



M/R Computing Model



M/R Computing Model

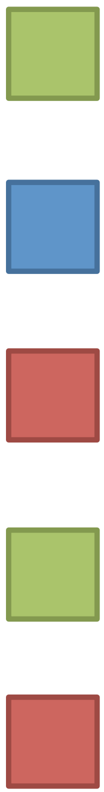
Take a set of inputs (files, tables, text…) and a function **F**

1. **Read** (batches of) inputs and assign each input to a group
   - this is called the MAP phase
   - and can be done in parallel *(according to file distribution)*

2. **Regroup** the inputs according to the map criterion :
   - this is called the SHUFFLE phase
   - again, this can be done in parallel *(according to where data is sent)*

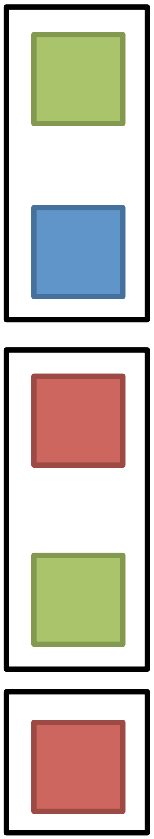
3. **Evaluate** the fuction on the new groups
   - this is called the REDUCE phase
   - again, this can be done in parallel *(according to where data is sent)*

M/R Computing Model



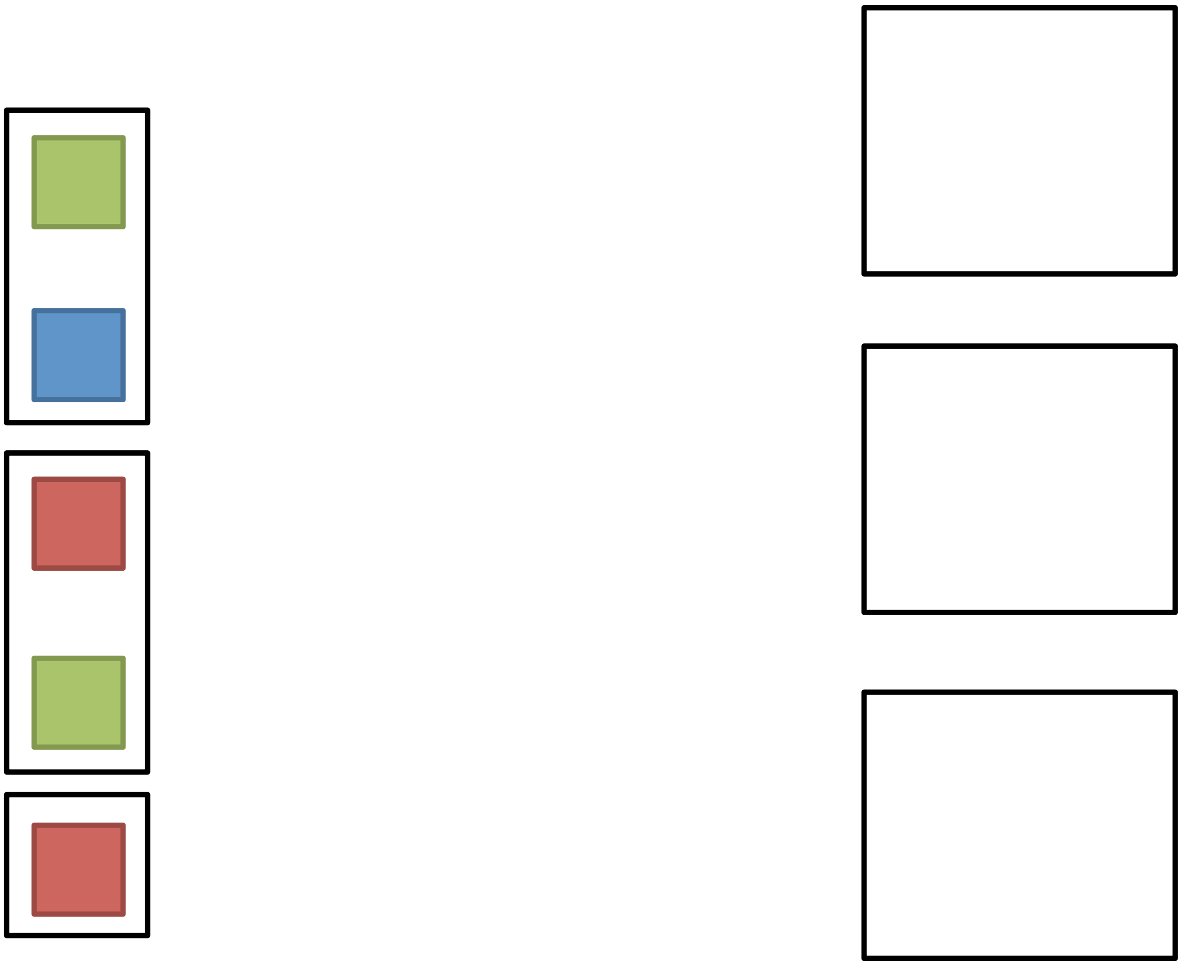
MAP : read batches of inputs

M/R Computing Model



MAP : read batches of inputs

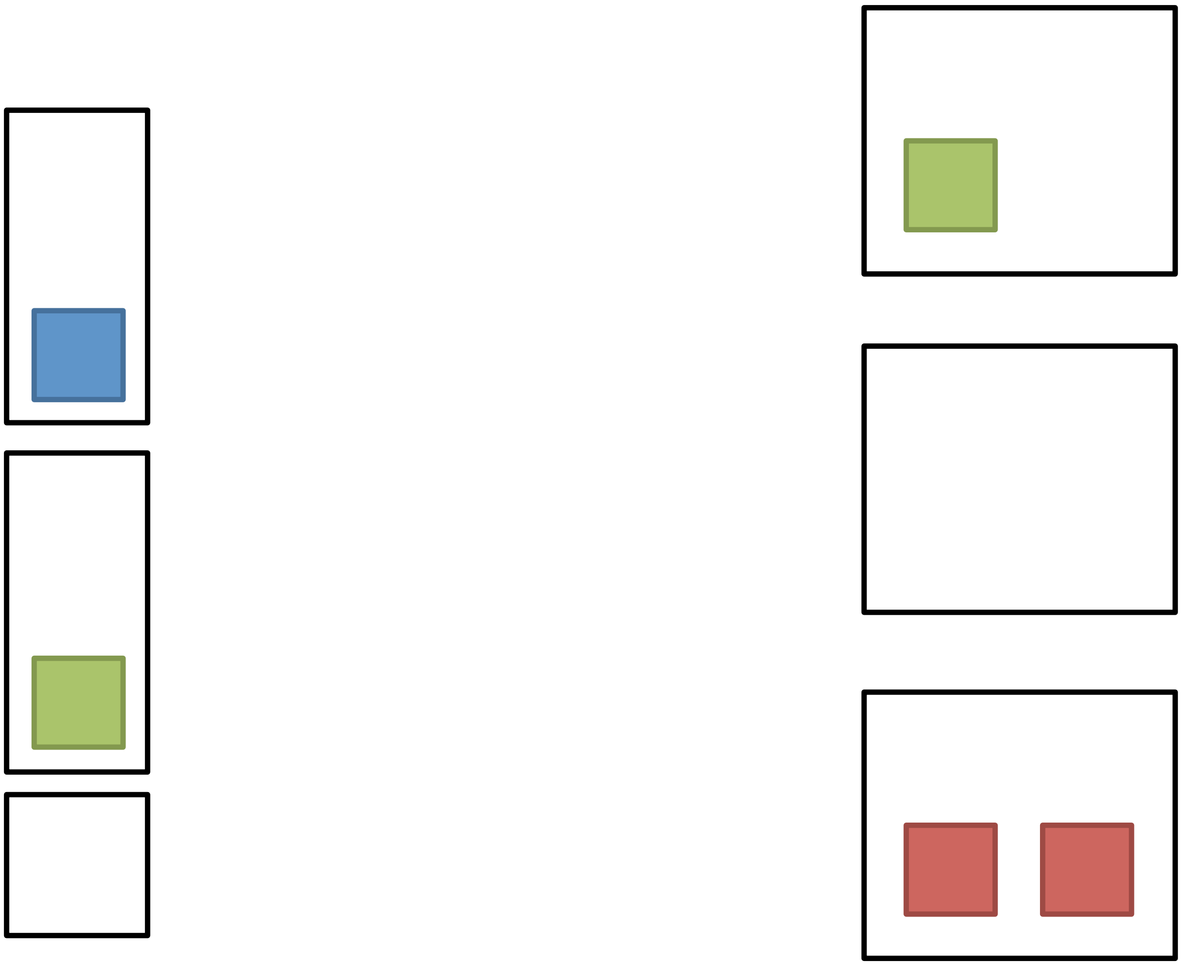
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

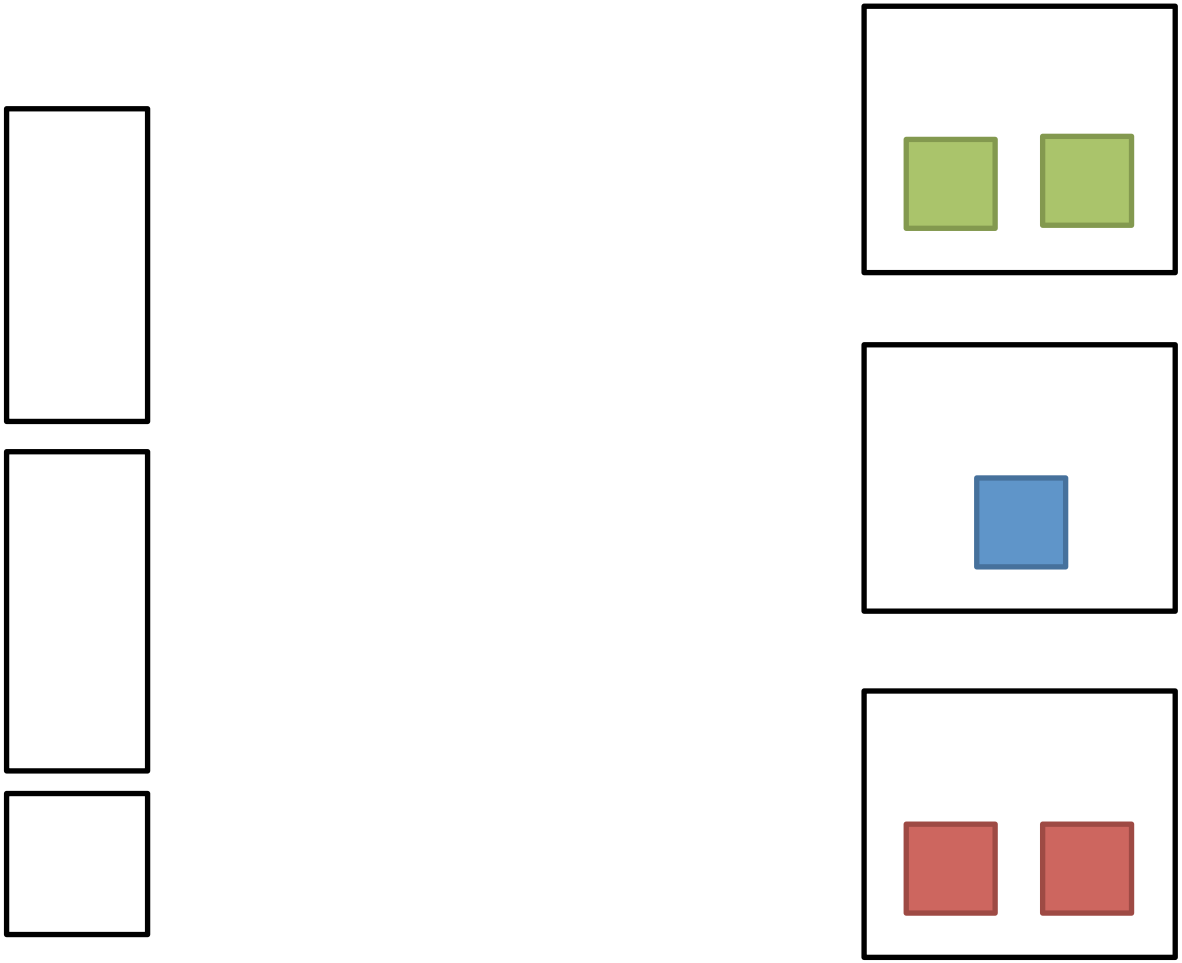
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

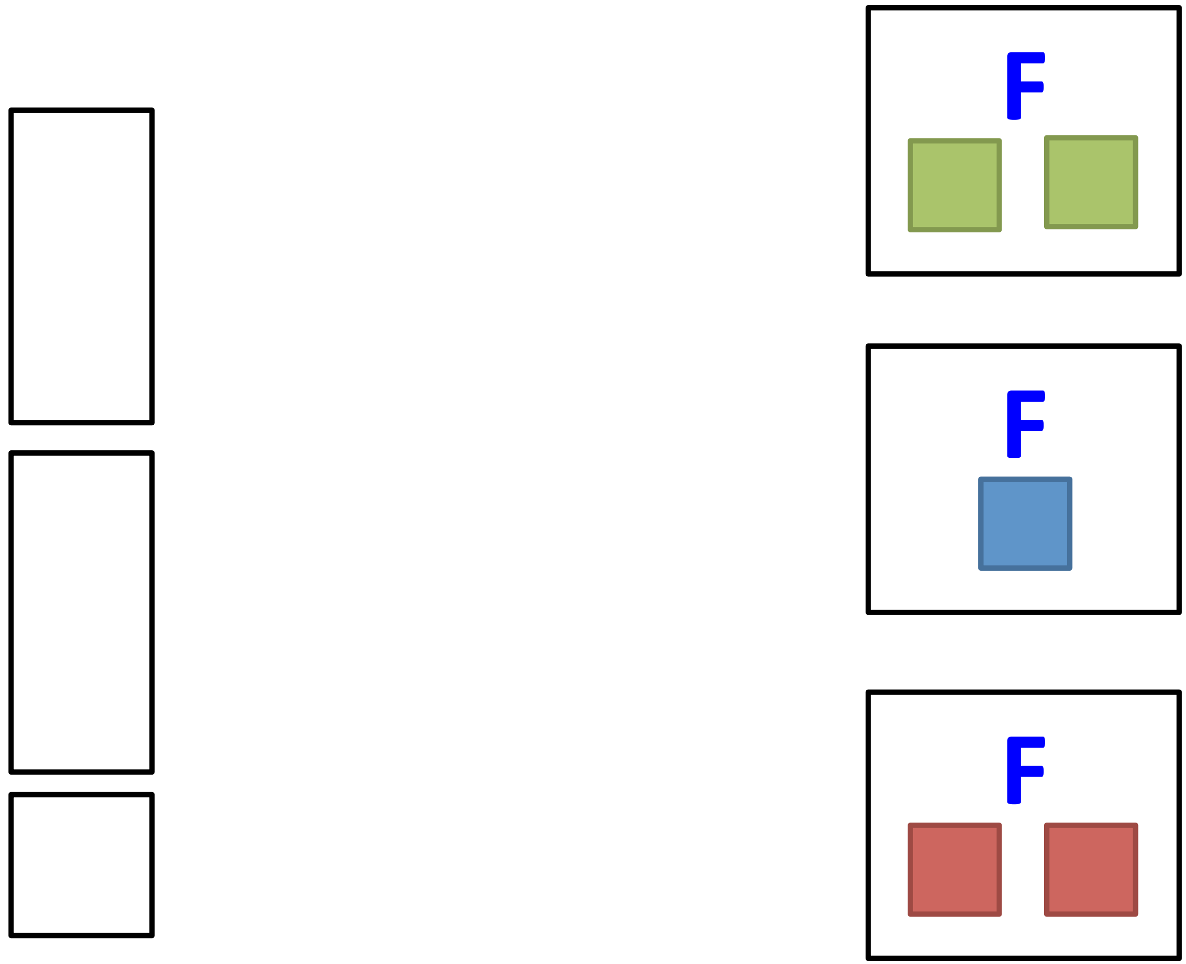
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

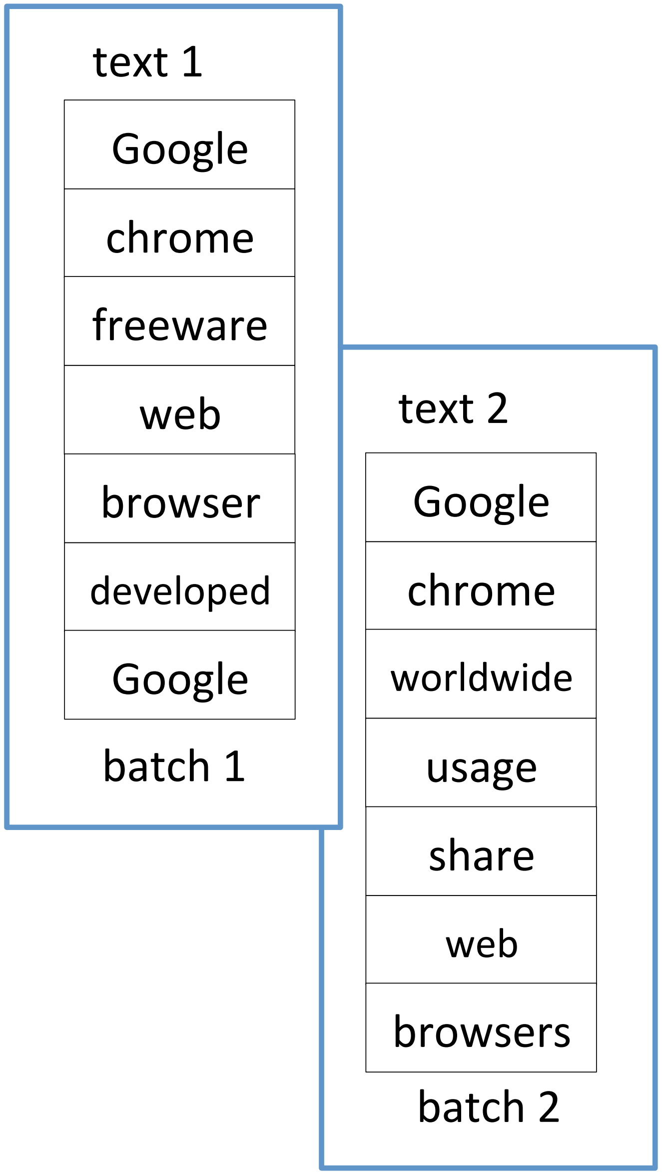
M/R Computing Model



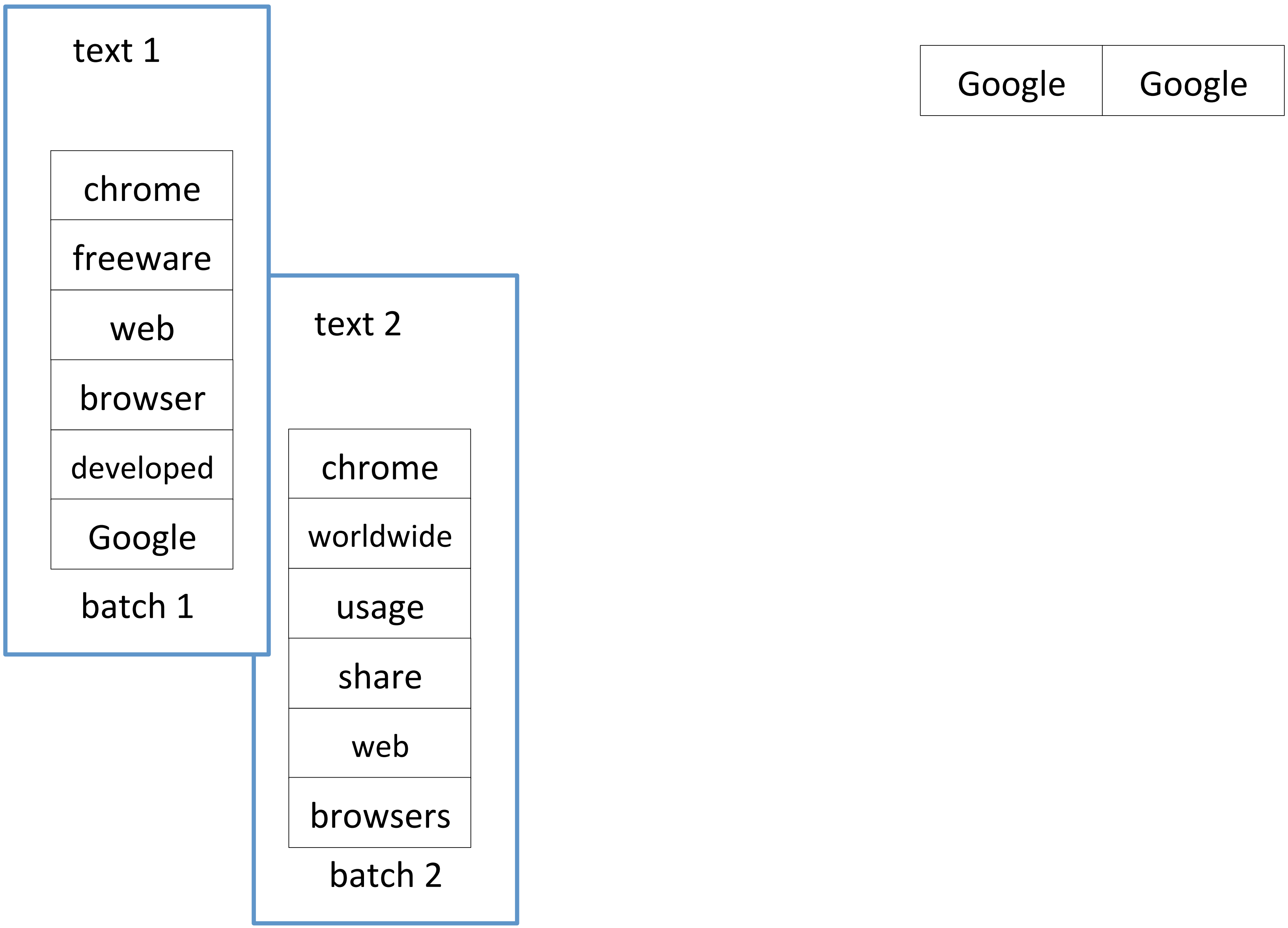
MAP : read batches of inputs	SHUFFLE : regroup the inputs	REDUCE : evaluate the function

Text Mining : Keyword Count

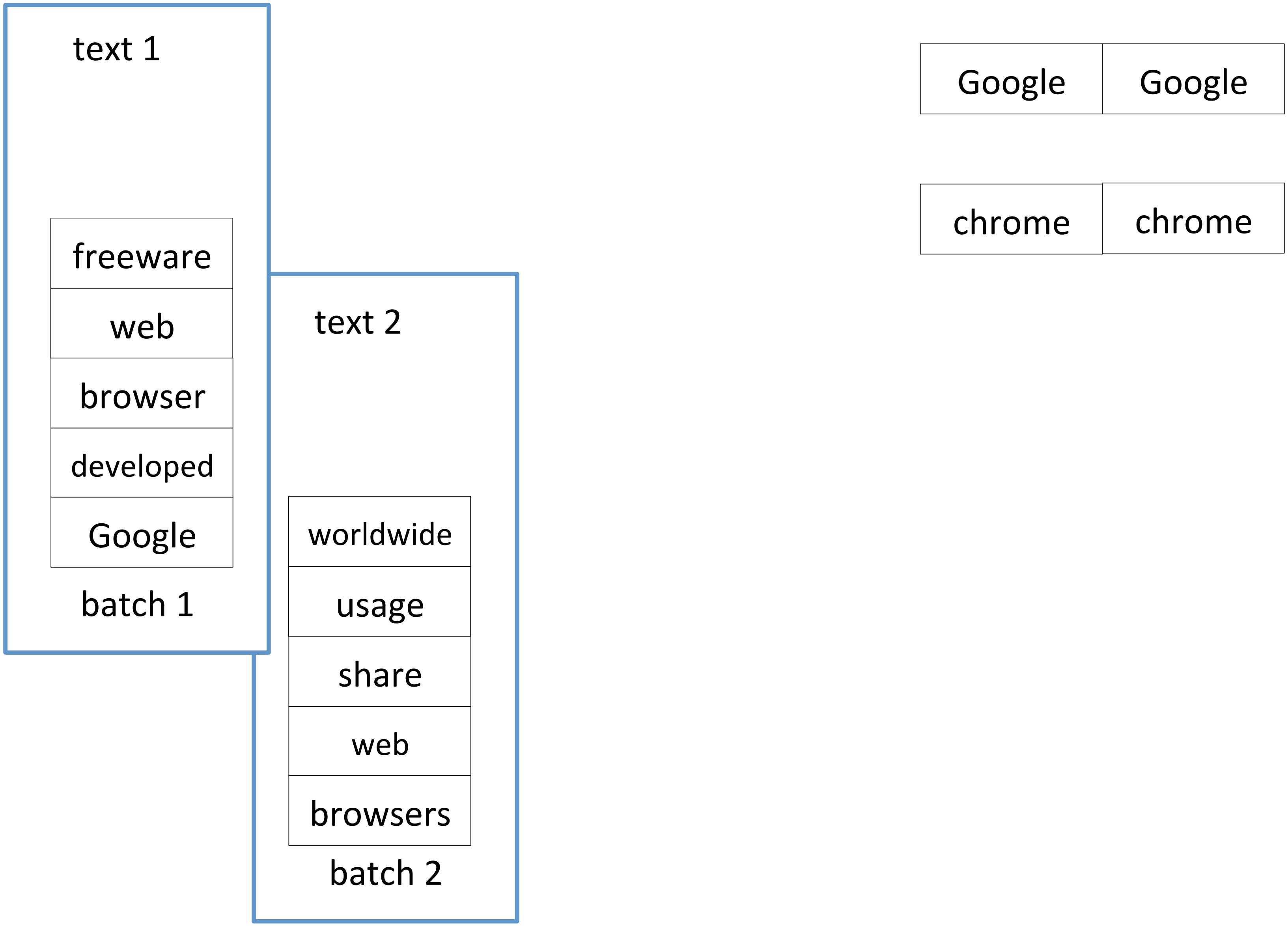
M/R Computing Model



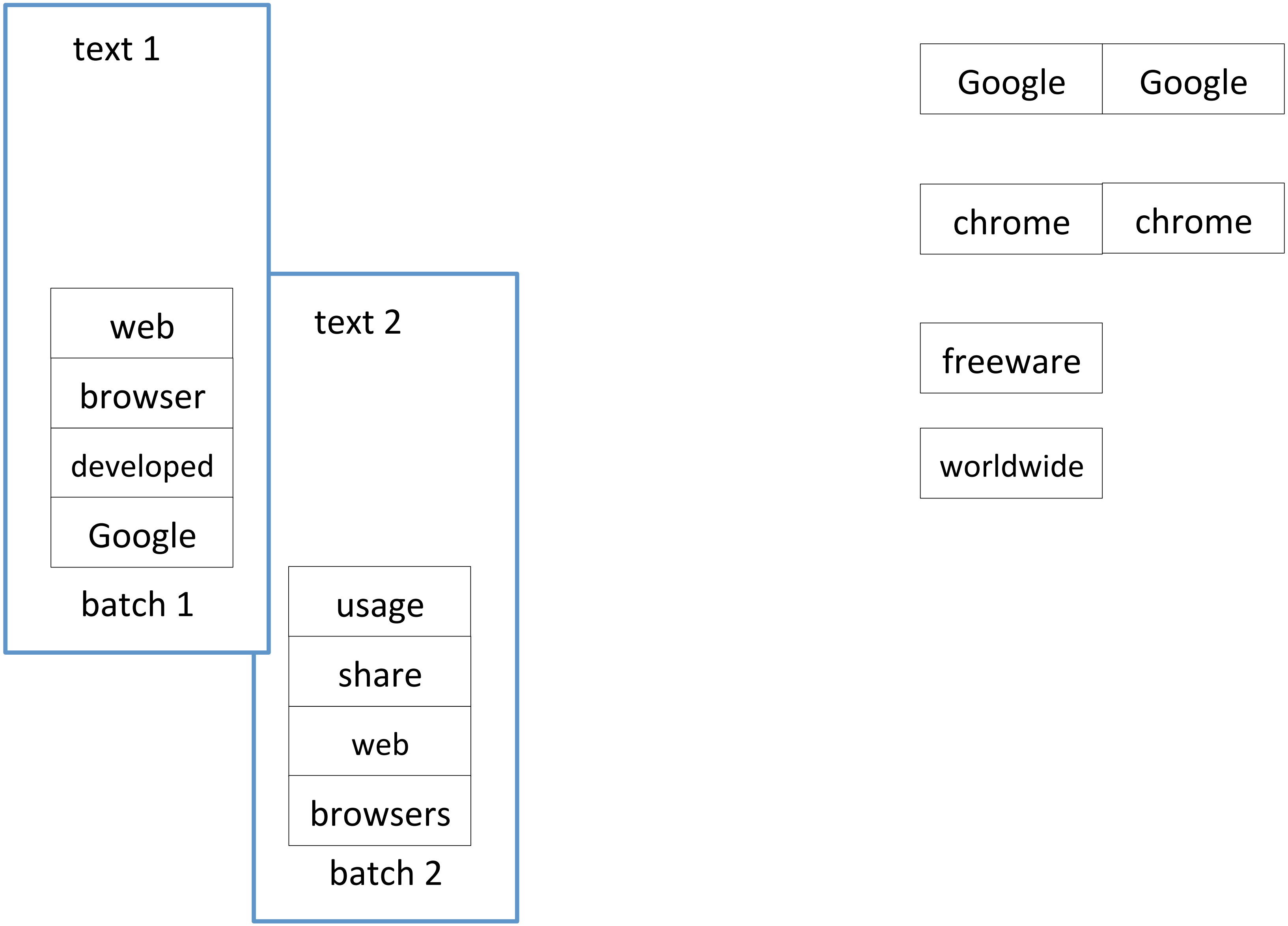
M/R Computing Model



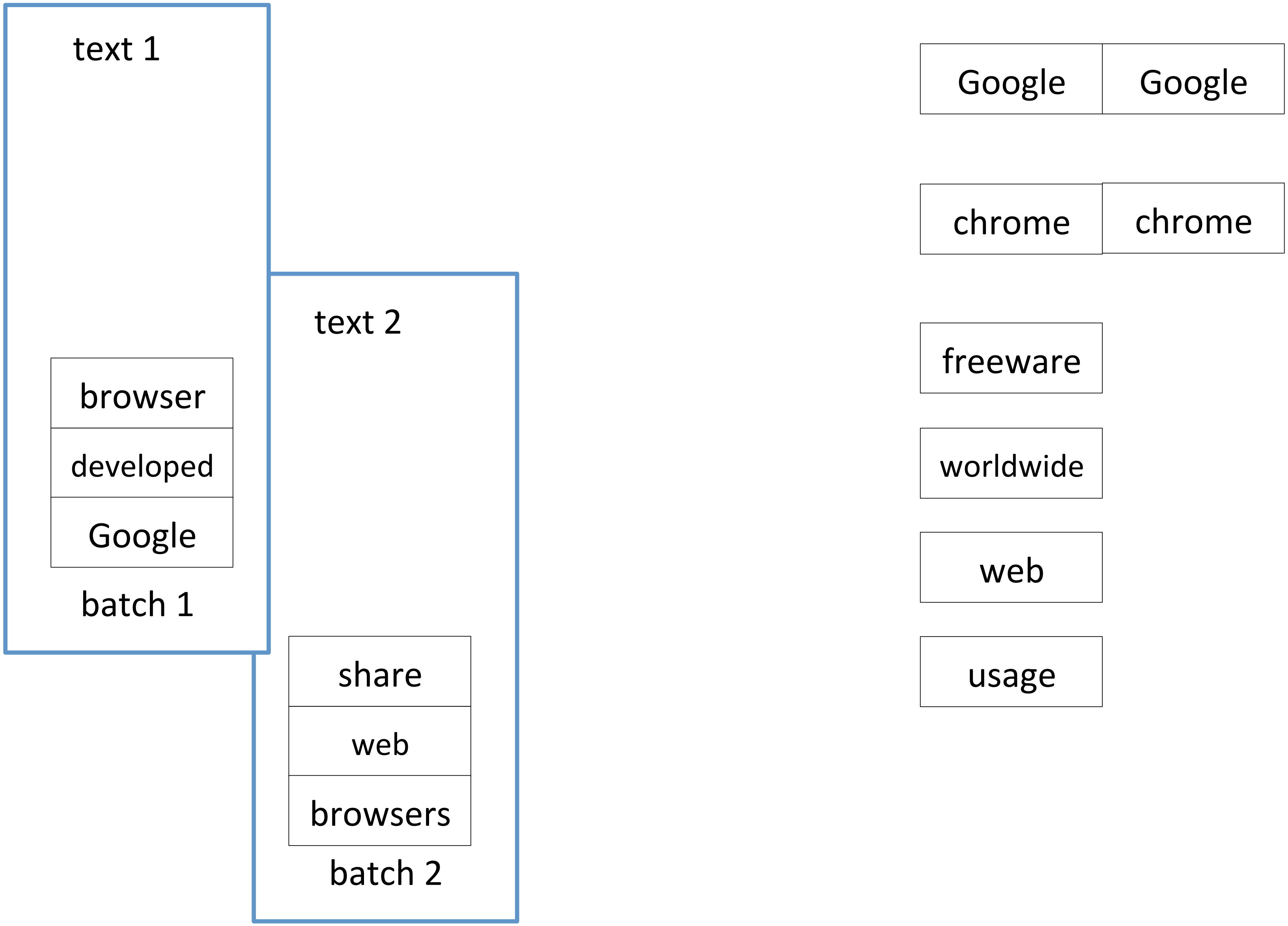
M/R Computing Model



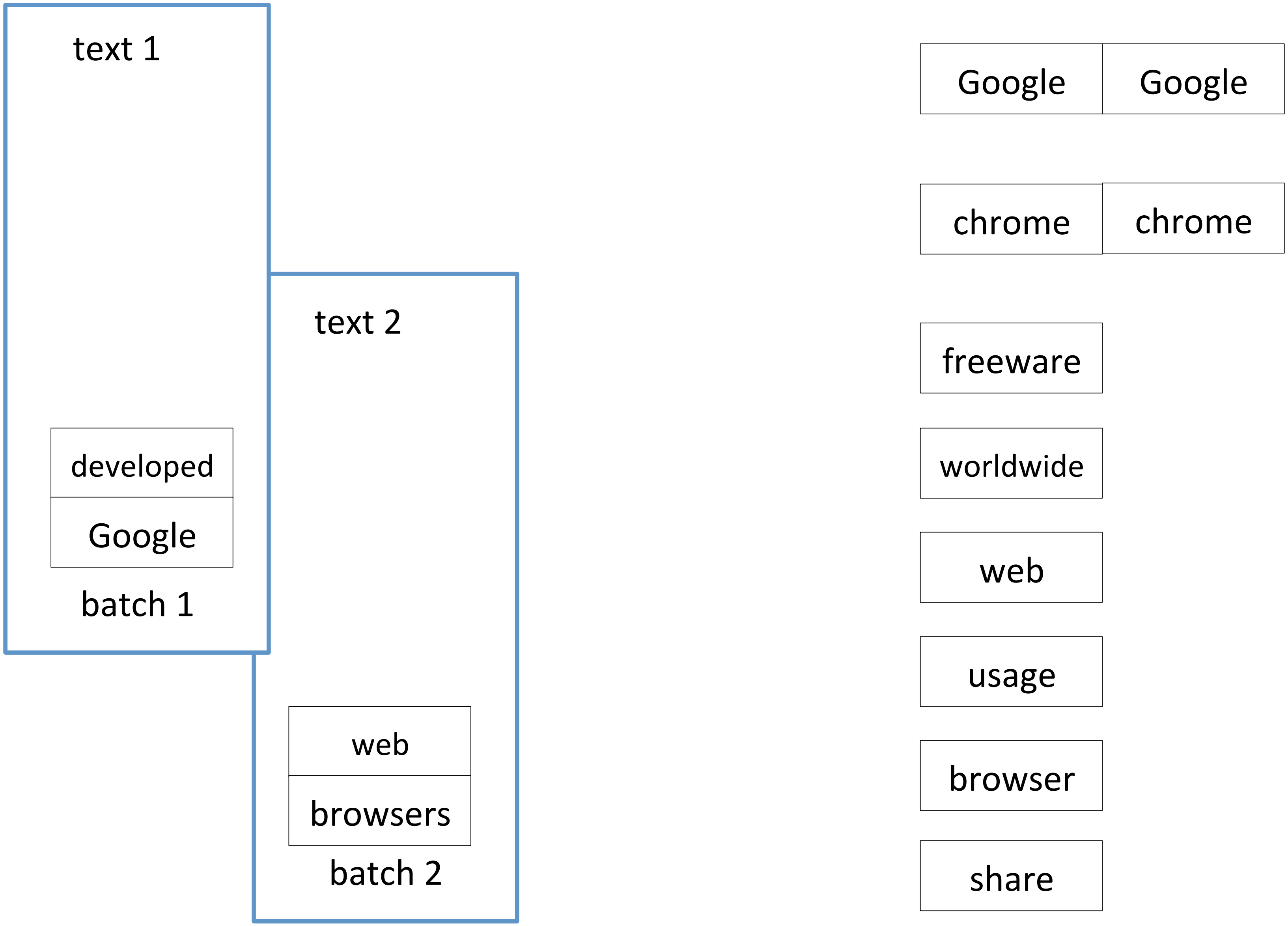
M/R Computing Model



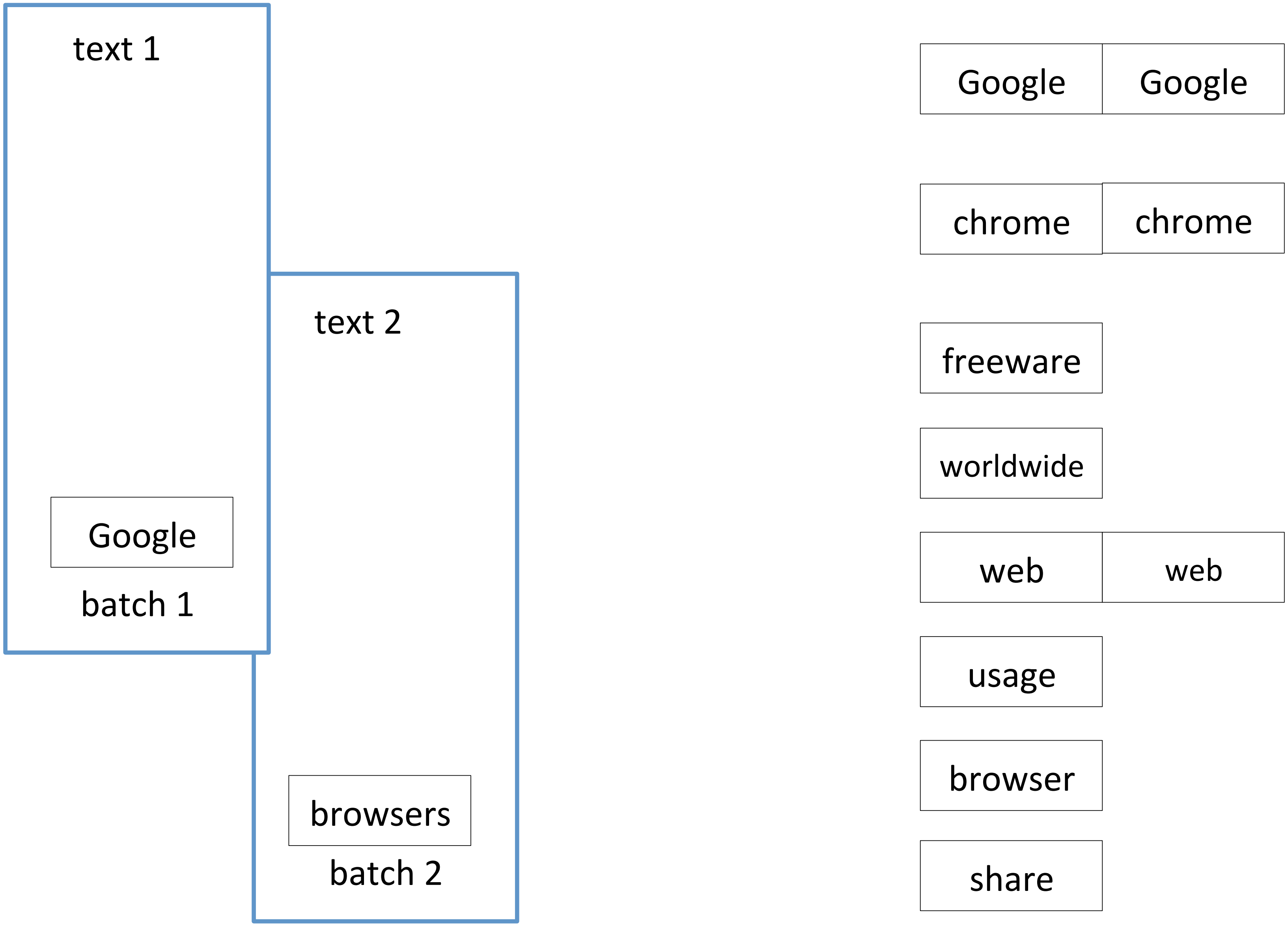
M/R Computing Model



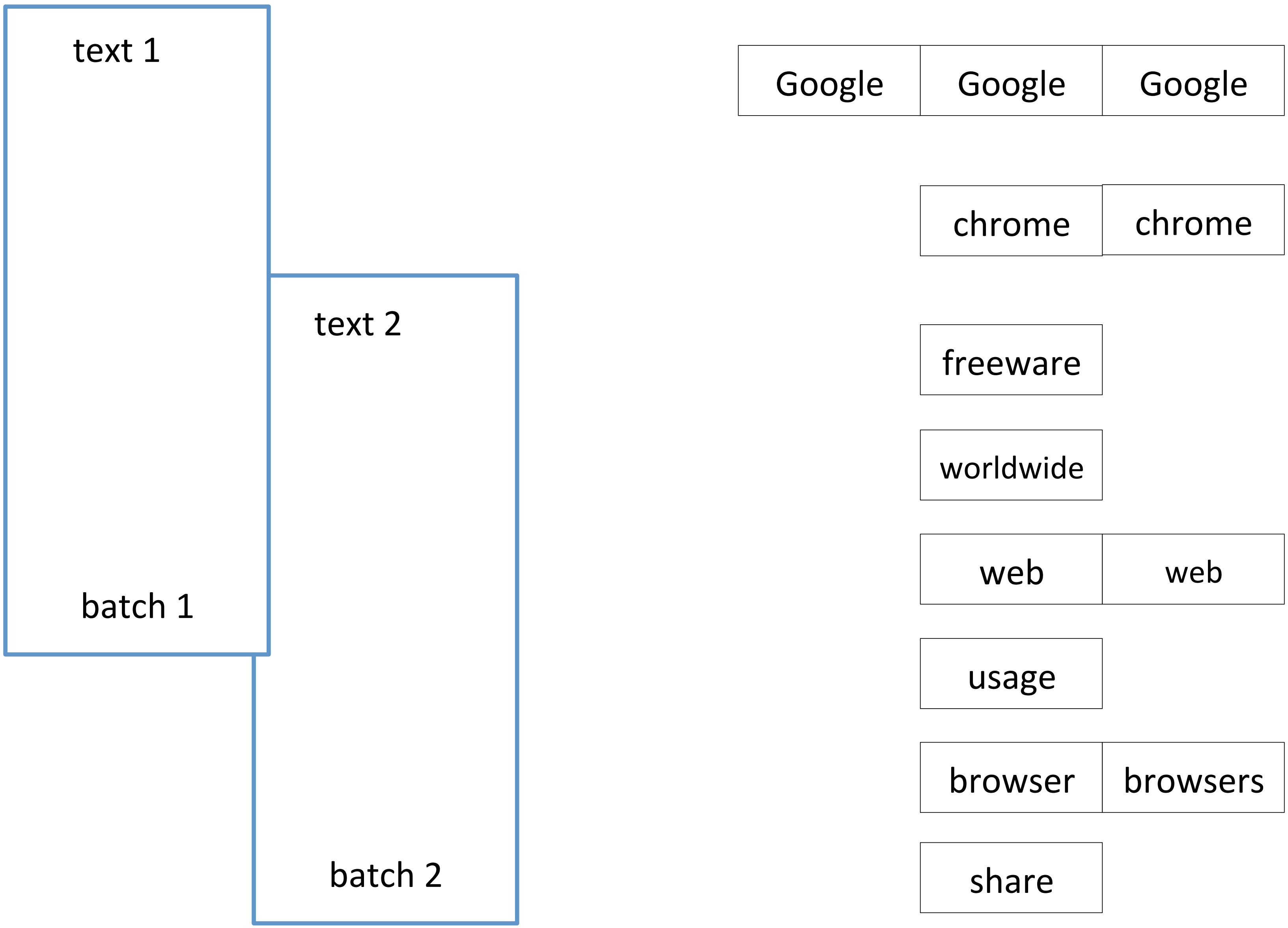
M/R Computing Model



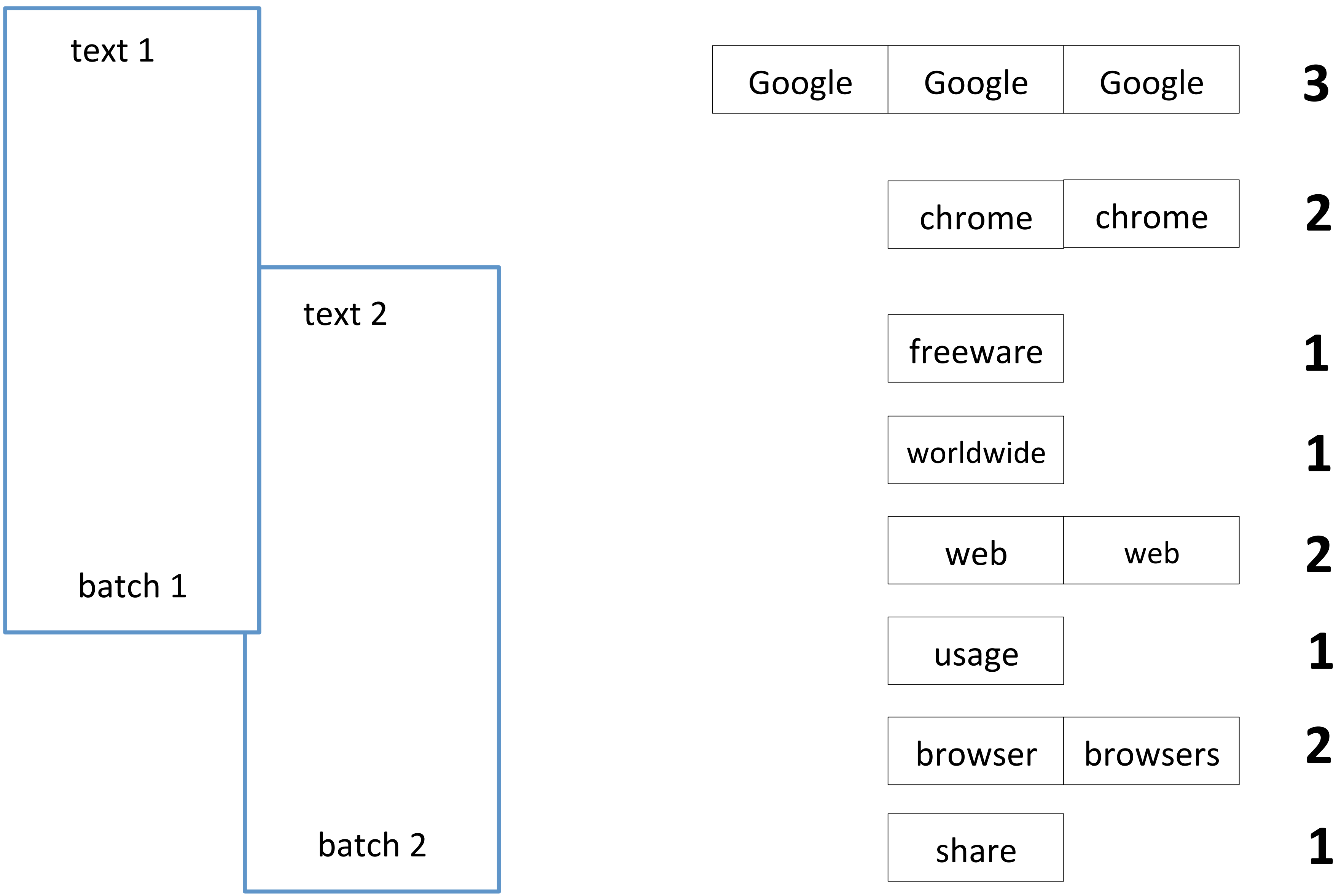
M/R Computing Model



M/R Computing Model



M/R Computing Model

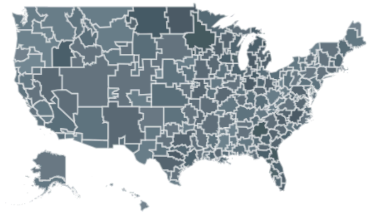


Analyze search logs to find popular trends

Google processes 3.5 billion searches per day

Midterm Elections 2018

Hundreds of candidates vied for your vote across the US. See the top issues in search.

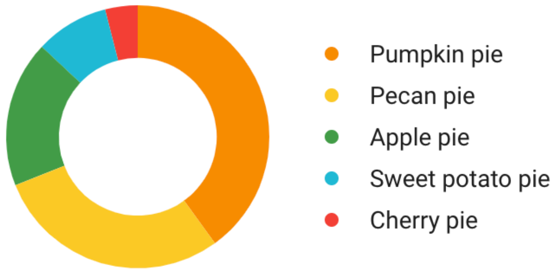


Search interest in voting, 10/30 to 11/06

READ MORE →

Thanksgiving 2018

Thanksgiving falls on the 4th Thursday of November every year.



Most searched pies, past week US

READ MORE →

Query Log Example

How many days until Thanksgiving?
What restaurants are open on Thanksgiving?
Is Trump party going to win the election ?
Where is Trump right now?
When is Thanksgiving?
Is Trump going to California ?
Can Trump win the next Presidency?

Why do we celebrate Thanksgiving?
When was the first Thanksgiving?
Why should I vote for Donald Trump ?
Why do people like Trump ?
What did Trump say ?
What tweeted Trump ?

Split the log

```
How many days until Thanksgiving?
What restaurants are open on Thanksgiving?
Is Trump party going to win the election ?
Where is Trump right now?
When is Thanksgiving?
Is Trump going to California ?
Can Trump win the next Presidency?
----------------------------------------------------
Why do we celebrate Thanksgiving?
When was the first Thanksgiving?
Why should I vote for Donald Trump ?
Why do people like Trump ?
What did Trump say ?
What tweeted Trump ?
```

Then analyze

How many days until Thanksgiving?
What restaurants are open on Thanksgiving?
Is Trump party going to win the election ?
Where is Trump right now?
When is Thanksgiving?
Is Trump going to California ?
Can Trump win the next Presidency?

Thanksgiving	5
Trump	8

Why do we celebrate Thanksgiving?
When was the first Thanksgiving?
Why should I vote for Donald Trump ?
Why do people like Trump ?
What did Trump say ?
What tweeted Trump ?

Query Processing

- Hadoop-M/R is **not** a data management system, it is a general framework.
- It can therefore implement queries :
  - ***does not have better performances than a DW***
  - ***but easier to setup & run, and more flexible***

Group By

```
SELECT store_id, sum(sale_amount)
FROM sales
GROUP BY store_id
```

Group-by in M/R

store_id	sale_amount
	10
	30
	20

	40
	50
	10

	80
	60

store_id **sale_amount**

Group-by in M/R

store_id	sale_amount
	30
	20

	50
	10

	60

store_id	sale_amount
	10
	40

	80

Group-by in M/R

store_id	sale_amount
	20
	10

store_id	sale_amount
	10
	40
	60
	30
	80
	50

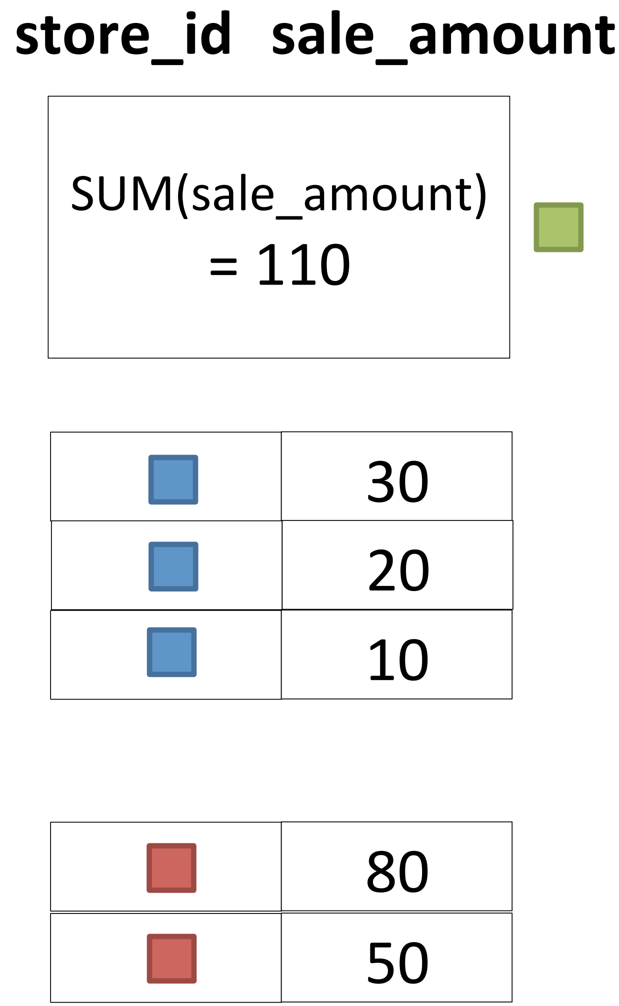
Group-by in M/R

store_id **sale_amount**

store_id	sale_amount
	10
	40
	60
	30
	20
	10
	80
	50

Group-by in M/R

store_id sale_amount



Group-by in M/R

store_id sale_amount

store_id sale_amount

SUM(sale_amount)
= 110

SUM(sale_amount)
= 60

SUM(sale_amount)
= 130

Join

```
SELECT store_name, sale_amount
FROM   sales, store
WHERE
sales.store_id = store.store_id
```

Join in M/R

store_id	sale_amount
	10
	30
	20
	40
	50
	10

store_id	name
	Green
	Blue
	Red

Join in M/R

store_id
sale_amount

	30
	20

	50
	10

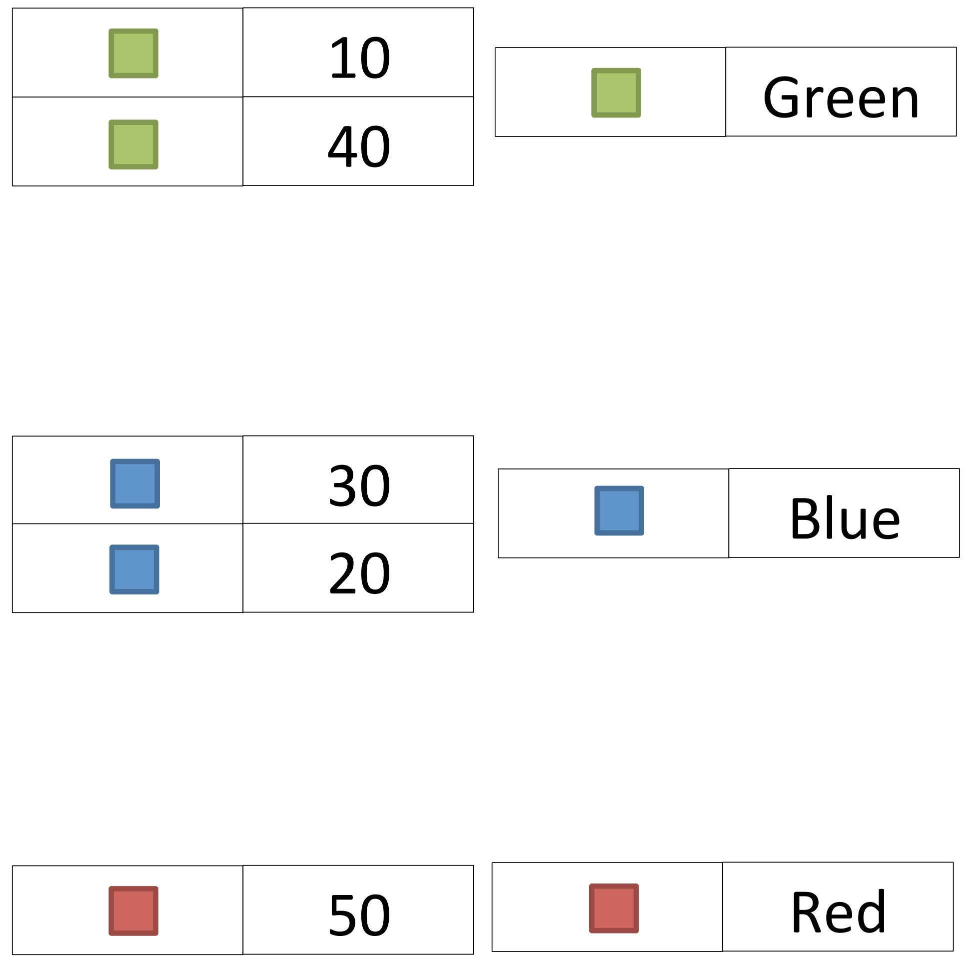
	10
	40

store_id **name**

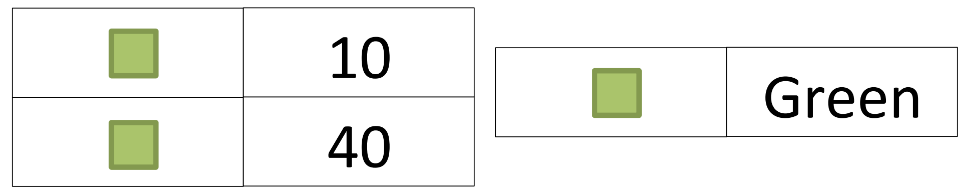
	Blue
	Red

	Green

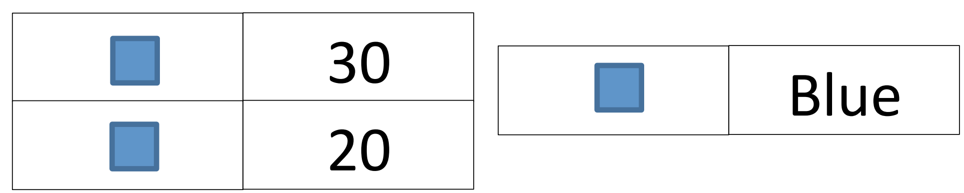
Join in M/R



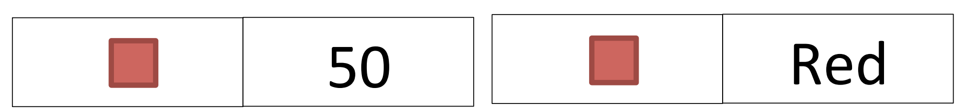
Join in M/R



(10, Green)
(40, Green)



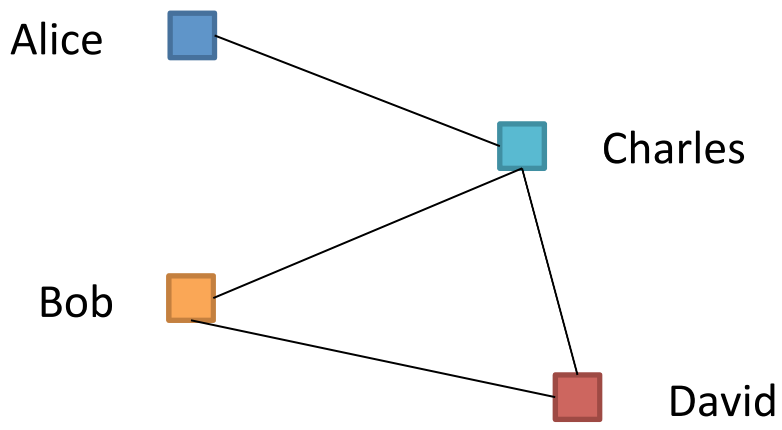
(30, Blue)
(20, Blue)



(50, Red)

Social Network Analysis

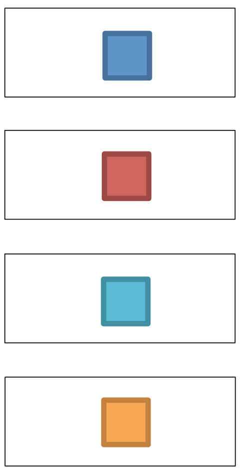
- Find all pairs of "similar" users
  - in terms of interests, age, country, behavior ...



- Worst-case n(n-1)/2 comparisons (n=#users)

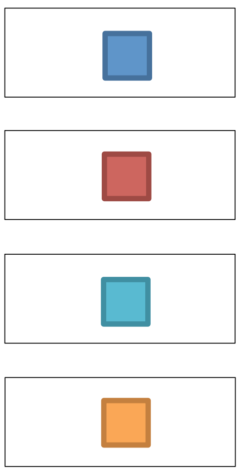
User-Similarity in M/R

user

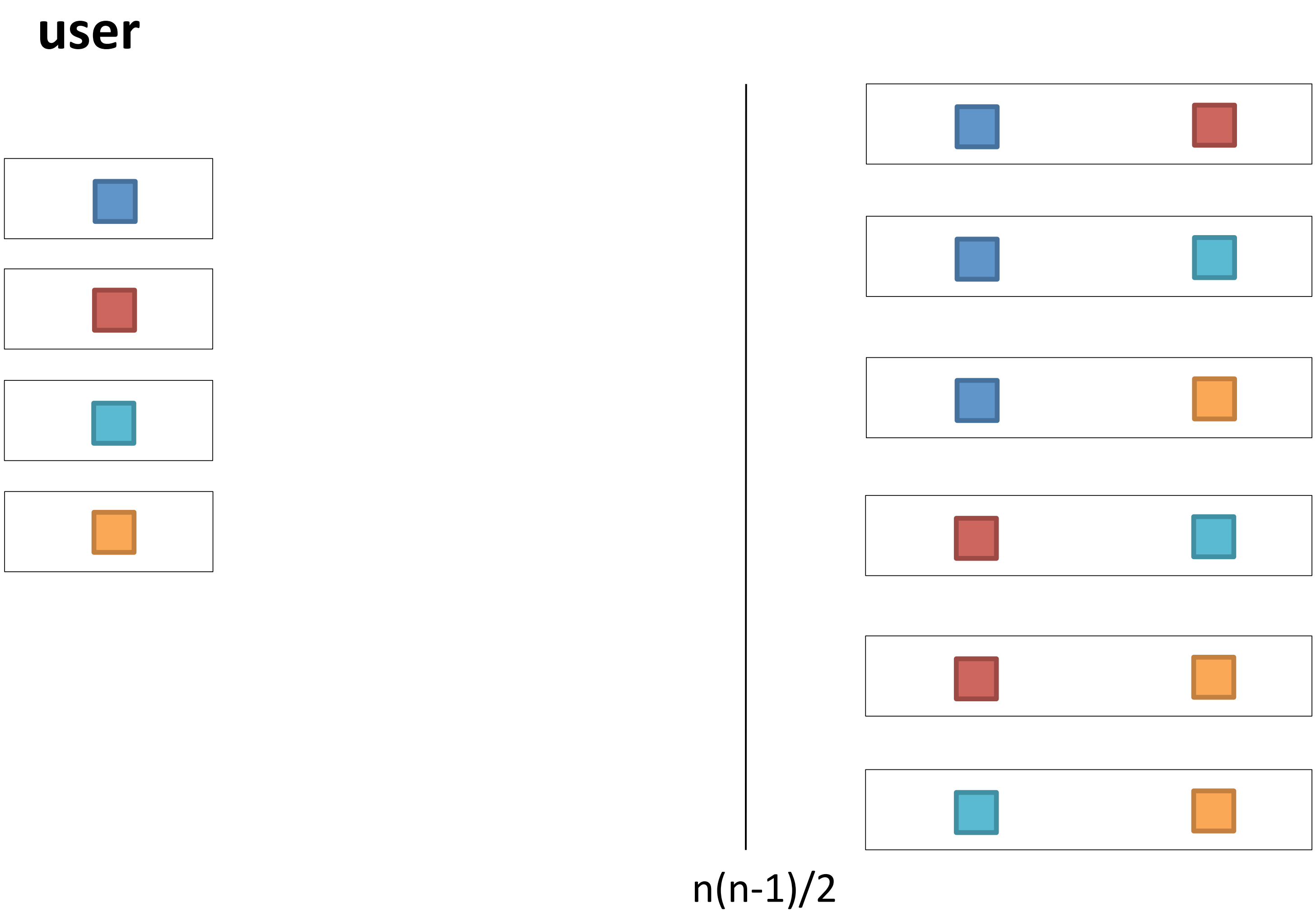


User-Similarity in M/R

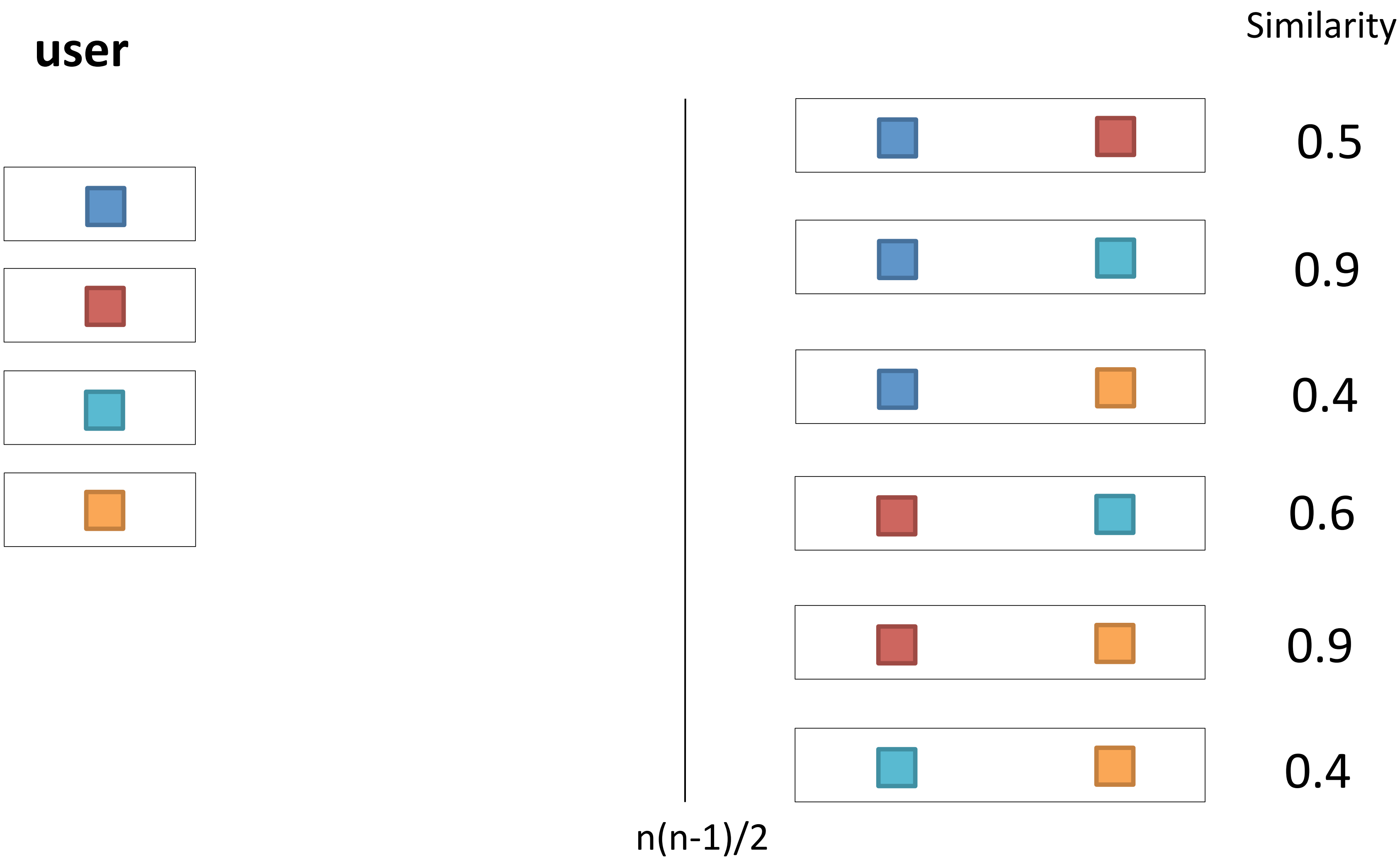
user



User-Similarity in M/R



User-Similarity in M/R



It is called M/R but…

- as we have seen it is rather :
  - Map
  - Shuffle (Regroup)
  - Reduce

- why is that ?
  - because the programmer just writes the Map and Reduce functions !

MAP-REDUCE PROGRAMMING

Central Notion

In MR very operation is expressed by using pairs

<key,value>

Keys are not only numbers !

Values are not only text !

Map(key k_{input}, value v) $\rightarrow$ Set<key,value>

Read : map function takes in input a key-value pair
and outputs a set of (key,value) pairs

A Map-call is executed for every (k_{input} , v) pair

- Word count : k_{input} is a line-id v is a line of text
- Trends : k_{input} is a line-id v is a log line
- Group by : k_{input} is a tuple id v is a tuple
- Join : k_{input} is a tuple id v is a tuple
- Similarity : k_{input} is a user id v is a user

Map(key k_{input}, value v) $\rightarrow$ Set<key,value>

Read : map function takes in input a key-value pair
and outputs a set of (k_{group} , v) pairs

A Map-call is executed for every (k_{input} , v) pair

- Word count : k_{input} is a line-id v is a line of text
- Trends : k_{input} is a line-id v is a log line
- Group by : k_{input} is a tuple id v is a tuple
- Join : k_{input} is a tuple id v is a tuple
- Similarity : k_{input} is a user id v is a user

Reduce(key k_{group}, Set<value> V) → Set<key,value>

Read : reduce function takes in input a key and a set of values and outputs a set of key-value pairs

> All (k_{group},v)-pairs for a given k_{group} are evaluated by the same reducer !

Wordcount :	k_{group} is a word	V number of occurrences
Trends :	k_{group} is a word	V number of occurrences
Group by :	k_{group} is a group-by attribute-value	V a set of tuples
Join :	k_{group} is a join attribute-value	V a set of tuples
Similarity :	k_{group} is a user-user pair id	V two users' profiles

Keyword Count Using MapReduce

```
map(key k, value phrase):

  for each word in phrase :

    emit( word , 1 )   //generates a <key,value> pair


reduce(key word, values occurrences):

   emit( key , occurrences.size() )
```

Word Count

phrase 1

Google
chrome
freeware
web
browser
developed
Google

phrase 2

Google
chrome
worldwide
usage
share
web
browser

Google,1	Google,1	Google,1	**3**
chrome,1	chrome,1		**2**
freeware,1			**1**
worldwide,1			**1**
web,1	web,1		**2**
usage,1			**1**
browser,1	browser,1		**2**
share, 1			**1**

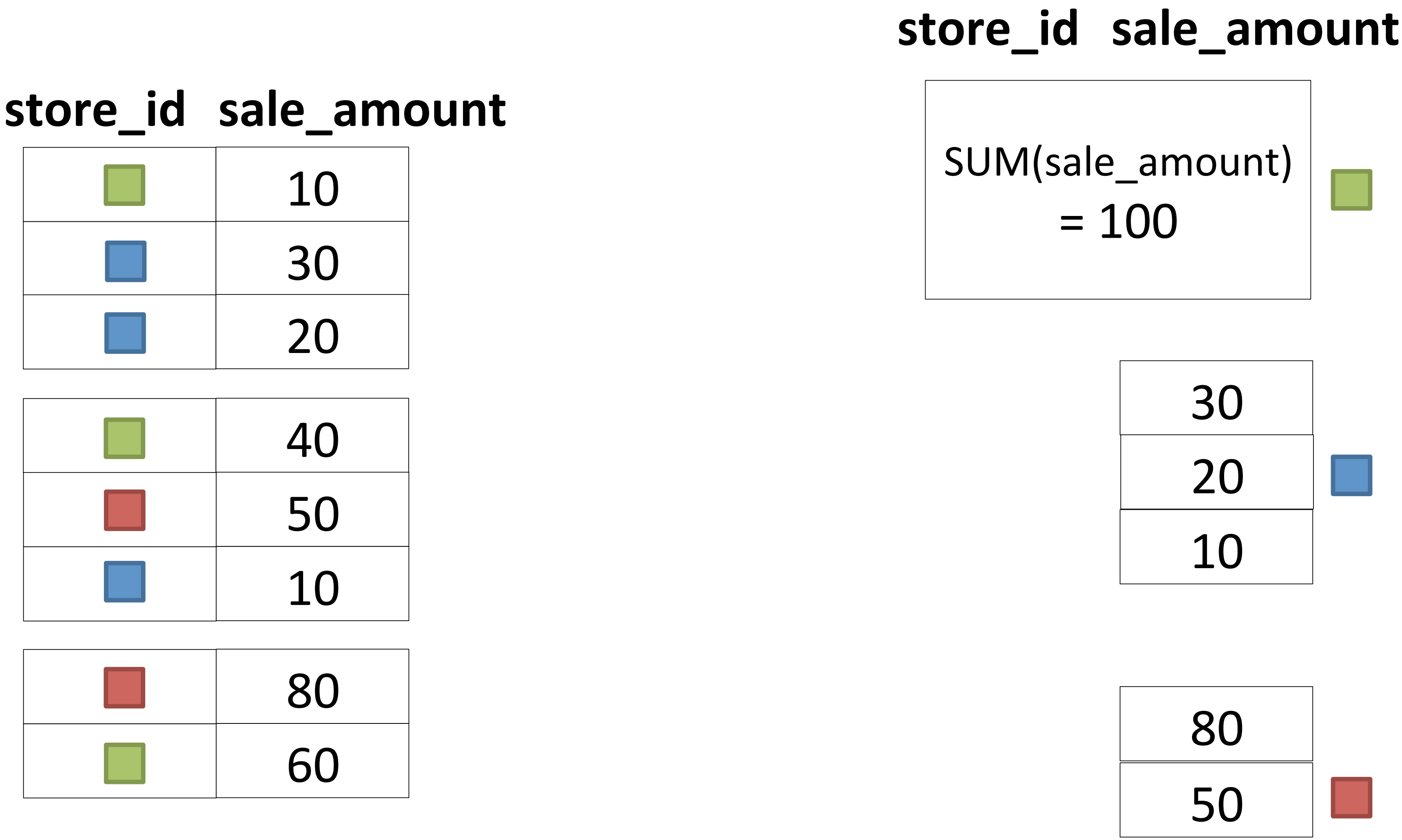
Query Trends Using MapReduce

```
map(key k, value batch_of_lines):

  for each word in batch_of_lines:
   emit( word , 1 )



reduce(key word , values occurrences):

   emit( key, occurrences.size() )
```

Group-by Using MapReduce

```
map(key k, value tuples):

  for each tuple in tuples:

    emit( tuple.store_id , tuple.sale_amount )

reduce(key store_id, values sales):

   total_sales=0;
   for each s in sales
      total_sales+=s;

   emit( store_id , total_sales )
```

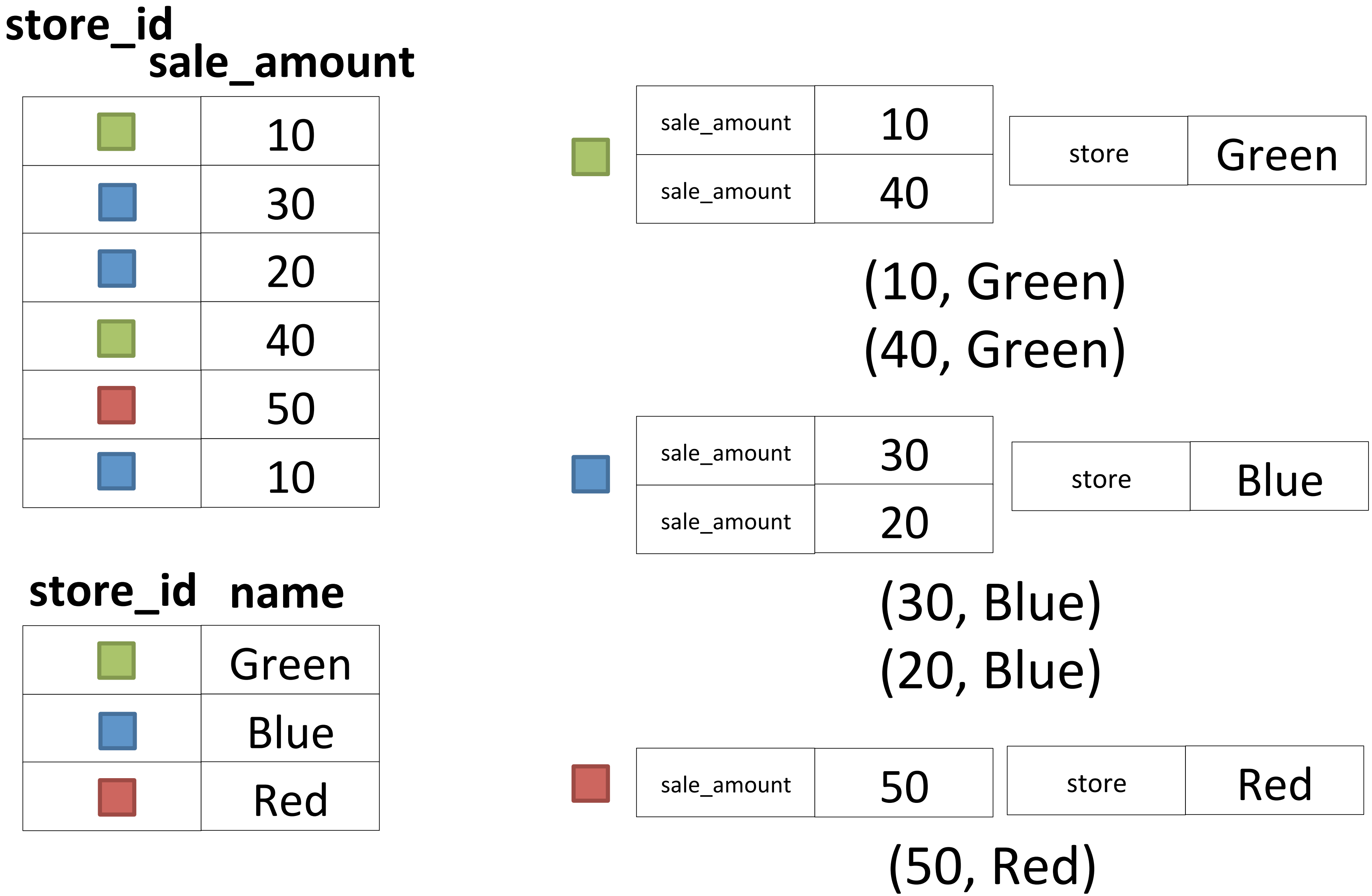
Group By in M/R



Join Using MapReduce

```
map(key k, value tuples):

  for each tuple t in tuples:
      if t belongs to SALES
        emit( t.store_id , <"profit",  t.sale_amount> )
      if t belongs to STORES
        emit( t.store_id , <"store", t.store_name>  )


reduce(key store_id, values mixed-attributes):

   for each a in mixed-attributes
      for each b in mixed-attributes
         if a[1]=="sale" and b[1]=="store"
            emit( store_id , <a[2],b[2]> )
```

Join in M/R


store_id
sale_amount
10
30
20
40
50
10
store_id
name
Green
Blue
Red
sale_amount
10
sale_amount
40
store
Green
(10, Green)
(40, Green)
sale_amount
30
sale_amount
20
store
Blue
(30, Blue)
(20, Blue)
sale_amount
50
store
Red
(50, Red)

User Similarity Using MapReduce

```
map(key user_i_id, value user_i_profile):

// send the profile of user i to all other users



reduce(key k_ij , values two_user_records):

// compare two users
```

User Similarity Using MapReduce

```
map(key user_i, value user_i_profile):
    for each user_j
        create a "new" destination-key k_{i,j}
            emit( k_{i,j} , user_i_profile )
```

Example: Three Users

Example: Three Users

Mapper for user 1

k_{1,2} User 1 profile

k_{1,3} User 1 profile

Mapper for user 2

k_{2,1} User 2 profile

k_{2,3} User 2 profile

Mapper for user 3

k_{3,1} User 3 profile

k_{3,2} User 3 profile

Reducer for k_{1,2}=k_{2,1}

Reducer for k_{1,3}=k_{3,1}

Reducer for k_{3,2}=k_{2,3}

User Similarity Using MapReduce

```
map(key user_i, value user_i_profile):
    for each user_j                          with i != j
        create a key k_{i,j}          with k_{i,j}=k_{j,i}
            emit( k_{i,j} , user_i_profile)
```

User Similarity Using MapReduce

```
map(key user_i, value user_i_profile):
    for each user_j                          with i != j
        create a key k_{i,j}          with k_{i,j}=k_{j,i}
            emit( k_{i,j} , user_i_profile)


reduce(key k_ij , values two_user_records):

   u1 = two_user_records[1]
   u2 = two_user_records[2]

   if similarity( u1 , u2 ) >= 0.9
      emit( “similar” , < u1.id , u2.id > )
```

Map-Reduce Programming

There is not a single line of code dedicated to parallelization !!

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key/shuffle** step
- Handling machine failures
- Managing required inter-machine communication

Jeffrey Ullman

WHAT CAN GO WRONG

The “Drug Interaction” Problem

- Data consists of records for 3000 drugs.
  - List of patients taking them, dates, diagnoses.
  - About 1M of data per drug.

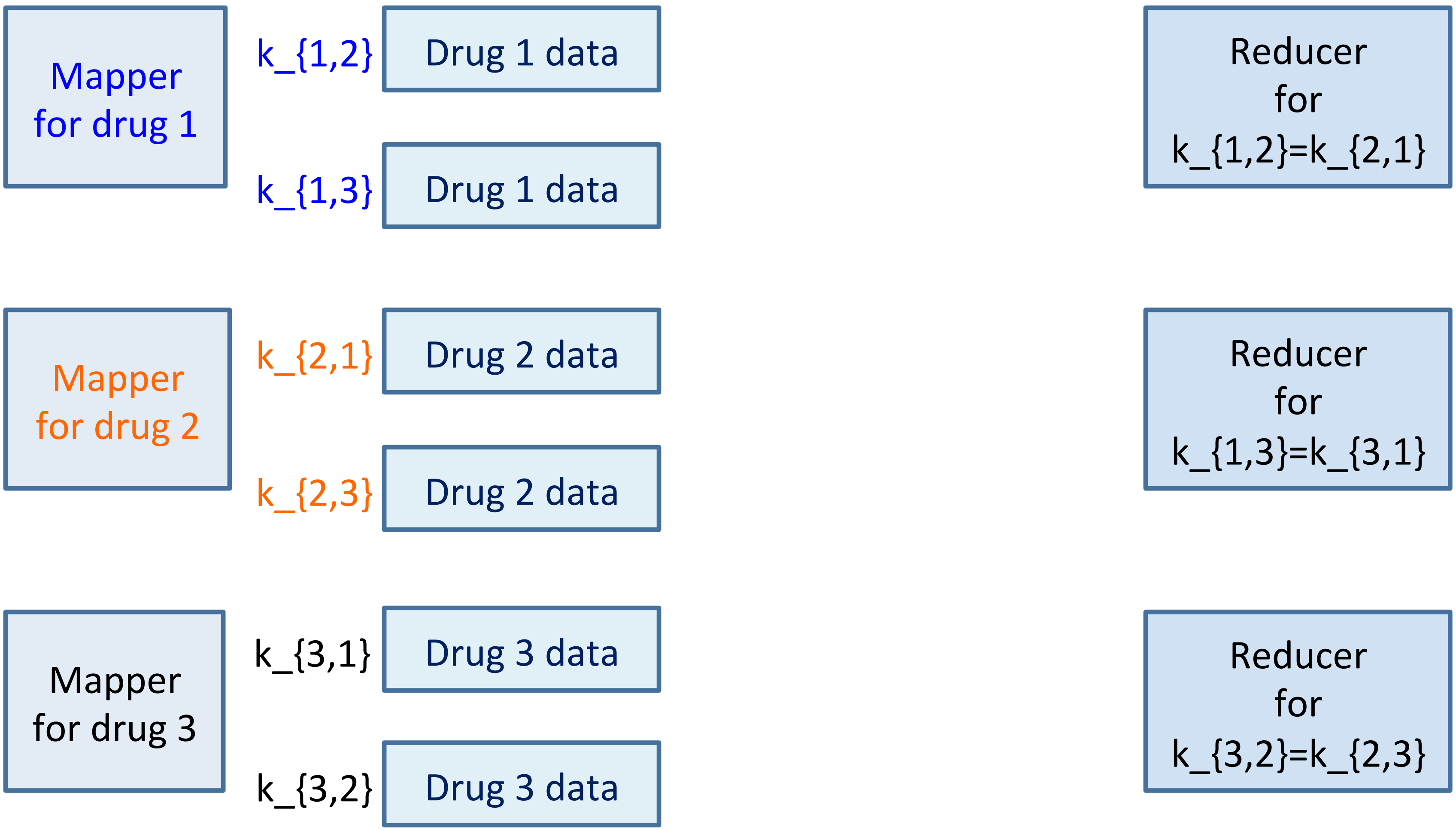
- Problem is to find drug interactions.
  - Example: two drugs that when taken together increase the risk of heart attack.

- Must examine each pair of drugs and compare their data using statistical tests.

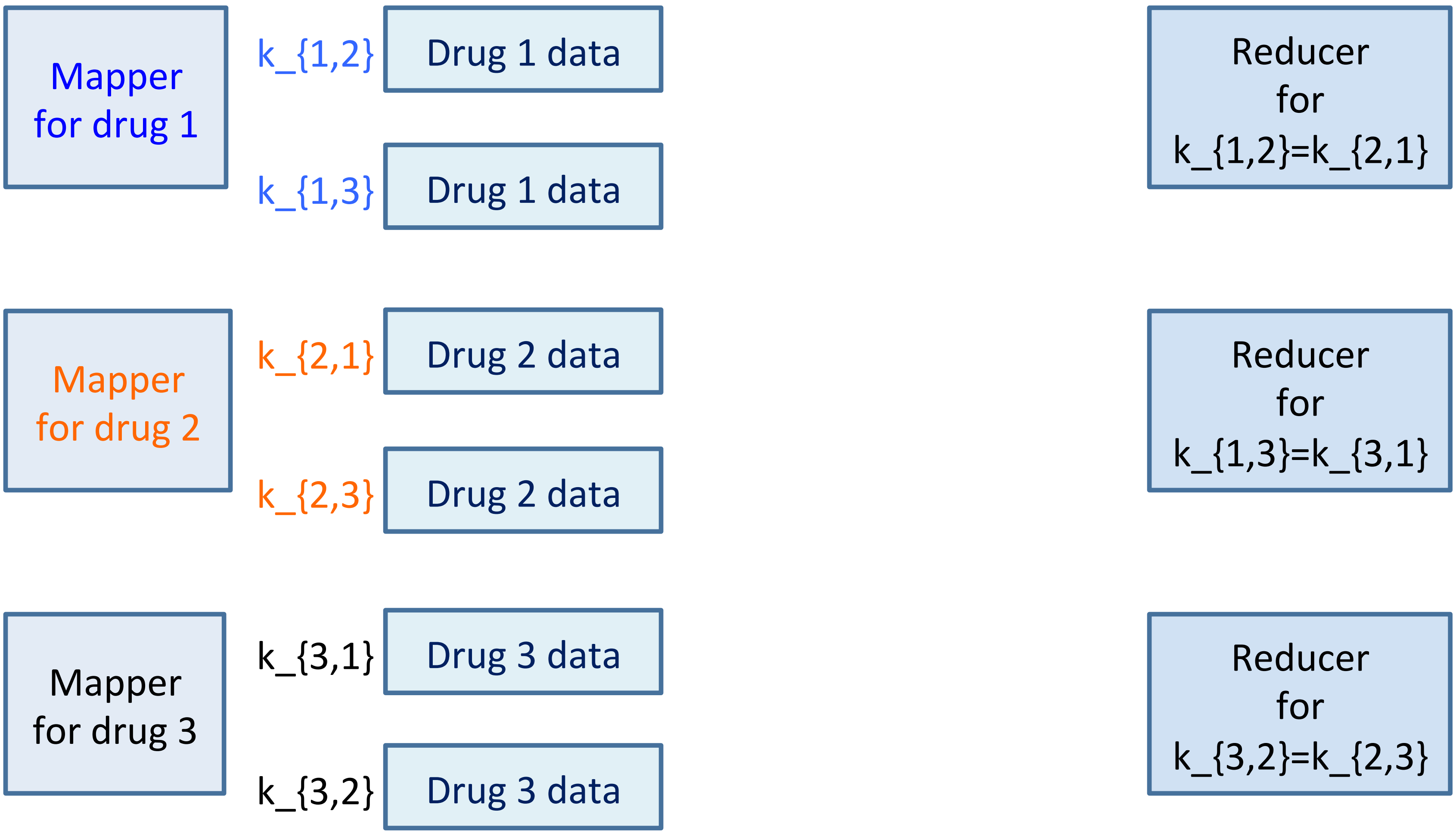
“Drug Interaction” Using MapReduce

```
map(key drug_i, value drug_i_record):
   for each j in 1..3000 with i != j
      create a key k_{i,j} such that k_{i,j}=k_{j,i}
         emit( k_{i,j} , drug_i_record )


reduce(key k_drug_pair , values two_drug_records):

   d1=two_drug_records [1]
   d2=two_drug_records [2]

   if statistical-test-significative( d1 , d2 )
      emit( “interacting”     , < d1.id , d2.id > )
   else
      emit( “non-interacting” , < d1.id , d2.id > )
```

Example: Three Drugs



Example: Three Drugs



What Went Wrong?

- 3000 drugs → 3000 map tasks
- each sends 2999 copies of a single drug record
- which amounts to 1MB
- = 9TB communicated over a 1Gb Ethernet
- ~ 90,000 seconds (25h) of network use.
  - assuming no other job is using the network

A Better Approach

- The way to handle this problem is to group the drugs
- For example : 30 groups of 100 drugs each
- This way, a single drug record is replicated 29 times instead of 2999

Drug Interaction Using MapReduce (2)

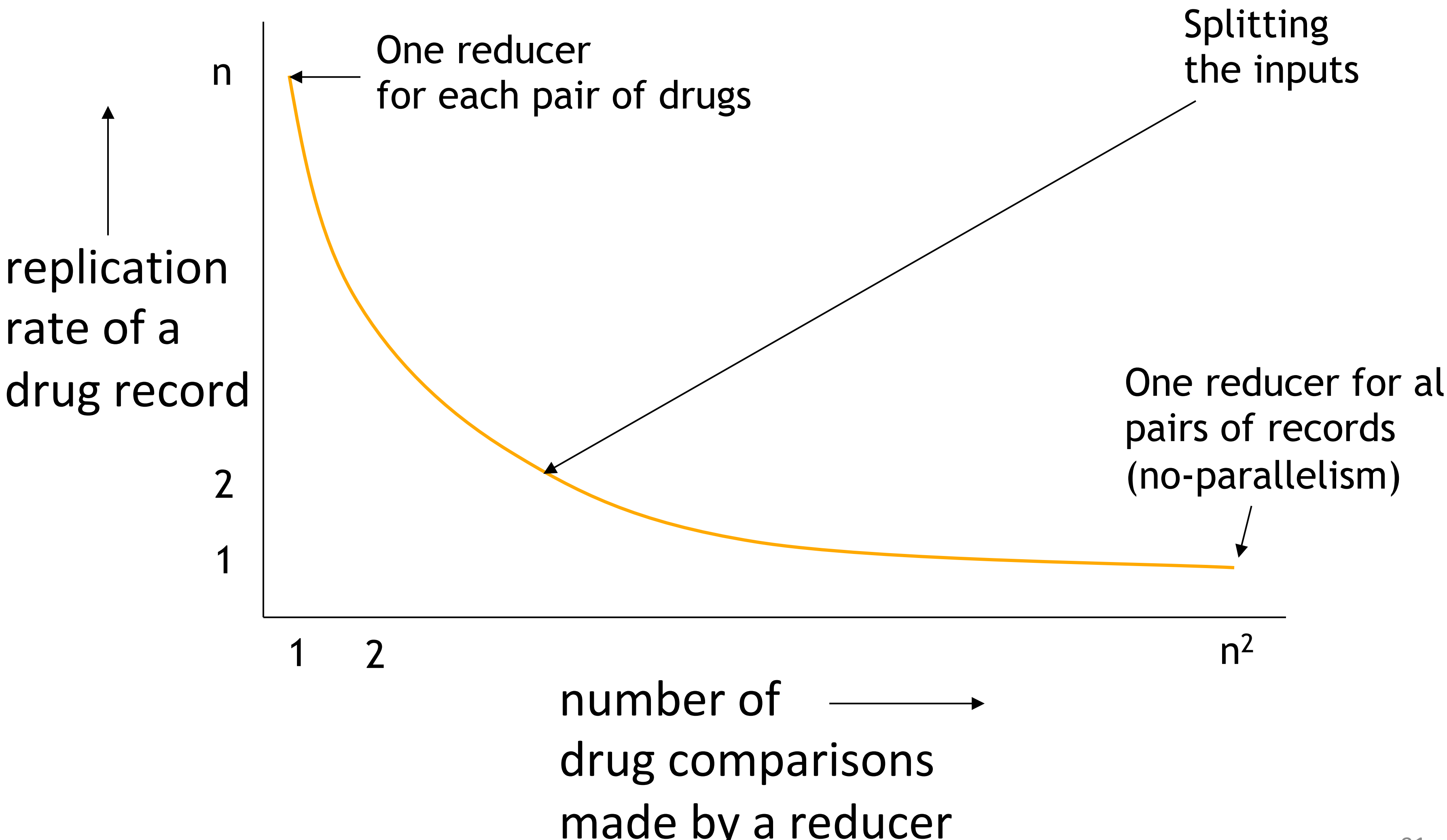
```
map(key drug_group_i_id, value drug_group_i_record):
    for each j in 1..30 with i != j
        create a key k_{i,j} such that k_{i,j}=k_{j,i}
            emit( k_{i,j} , drug_group_i_record )


reduce(key k_ij , values two_groups_records):

    g1=two_groups_records [1]
    g2=two_groups_records [2]
    for each d1 in g1
        for each d2 in g2
    if statistical-test-significative( d1 , d2 )
        emit( "interacting"     , < d1.id , d2.id > )
    else
        emit( "non-interacting" , < d1.id , d2.id > )
```

Example: Three Drugs

Mapper for drug **group** 1	k_{1,2}	Drug 1..30 data	Reducer for k_{1,2}=k_{2,1}
	k_{1,3}	Drug 1..30 data	
Mapper for drug **group** 2	k_{2,1}	Drug 31..60 data	Reducer for k_{1,3}=k_{3,1}
	k_{2,3}	Drug 31..60 data	
Mapper for drug **group** 3	k_{3,1}	Drug 61..90 data	Reducer for k_{3,2}=k_{2,3}
	k_{3,2}	Drug 61..90 data	

Why It Works

- The big difference is in the communication requirement.
- Now, each of 3000 drugs' 1MB records is replicated 29 times.
  - Communication cost = 87GB, vs. 9TB.



n
One reducer
for each pair of drugs
Splitting
the inputs
replication
rate of a
drug record
One reducer for all
pairs of records
(no-parallelism)
2
1
1
2
n^2
number of
drug comparisons
made by a reducer

Cost Measures for Algorithms

- **In MapReduce we quantify the cost of an algorithm using**

1. *Communication cost* = total I/O of all processes

2. *Computation cost* = total CPU time of all processes

Why is this important ?

- On a **public cloud**, you **pay for computation** and you also **pay for communication**.
  - Balancing the two is an important part of algorithm design.

- **If communication cost dominates total cost** it influences how much parallelism you can extract from an algorithm.
  - time reductions are not as good as expected

Reducer size is a key point

- In many cases, the big issue is whether a reducer has too much input to operate in main memory.
  - To get reducers with small input size, you may need a lot of communication.

- The "Drug-Interaction Problem" is a good model for how one can trade off communication against parallelism.

And Why User Similarity Works ?

- 3000 users → 3000 map tasks
- each sends 2999 copies of a each user record
- which amounts to ~1KB
- = 9 GB communicated over a 1Gb Ethernet
- ~ 90 seconds of network use.

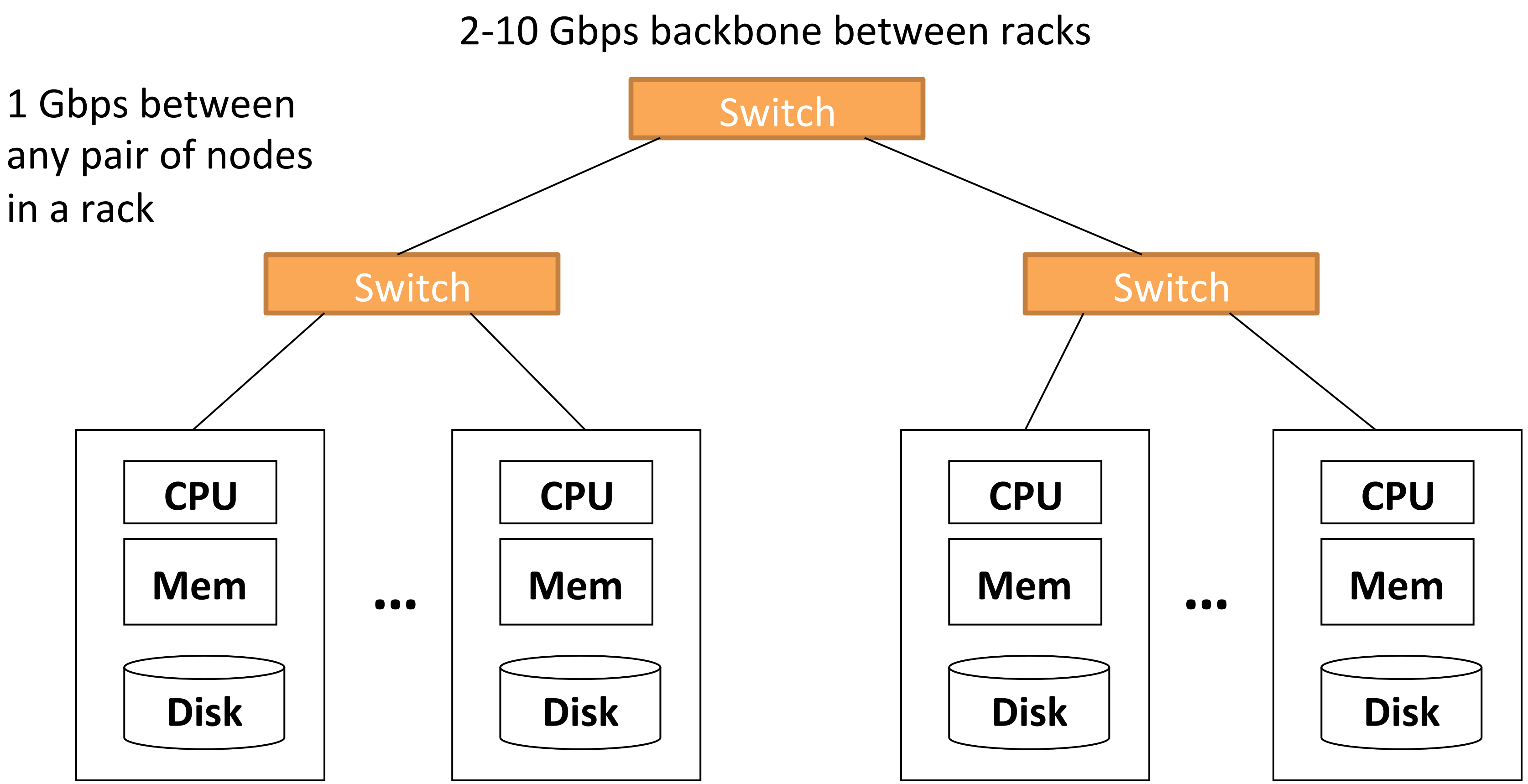
ARCHITECTURE

Big-Data Cluster

Today, a standard architecture for Big-Data is emerging:

- Cluster of commodity Linux nodes
- Commodity network (ethernet) to connect them
- Nodes Organized into racks
  - Intra-rack connection typically gigabit speed.
  - Inter-rack connection faster by a small factor.
- Shared-nothing : no shared memory
  - this is not High Performance Computing (HPC)

Cluster Architecture



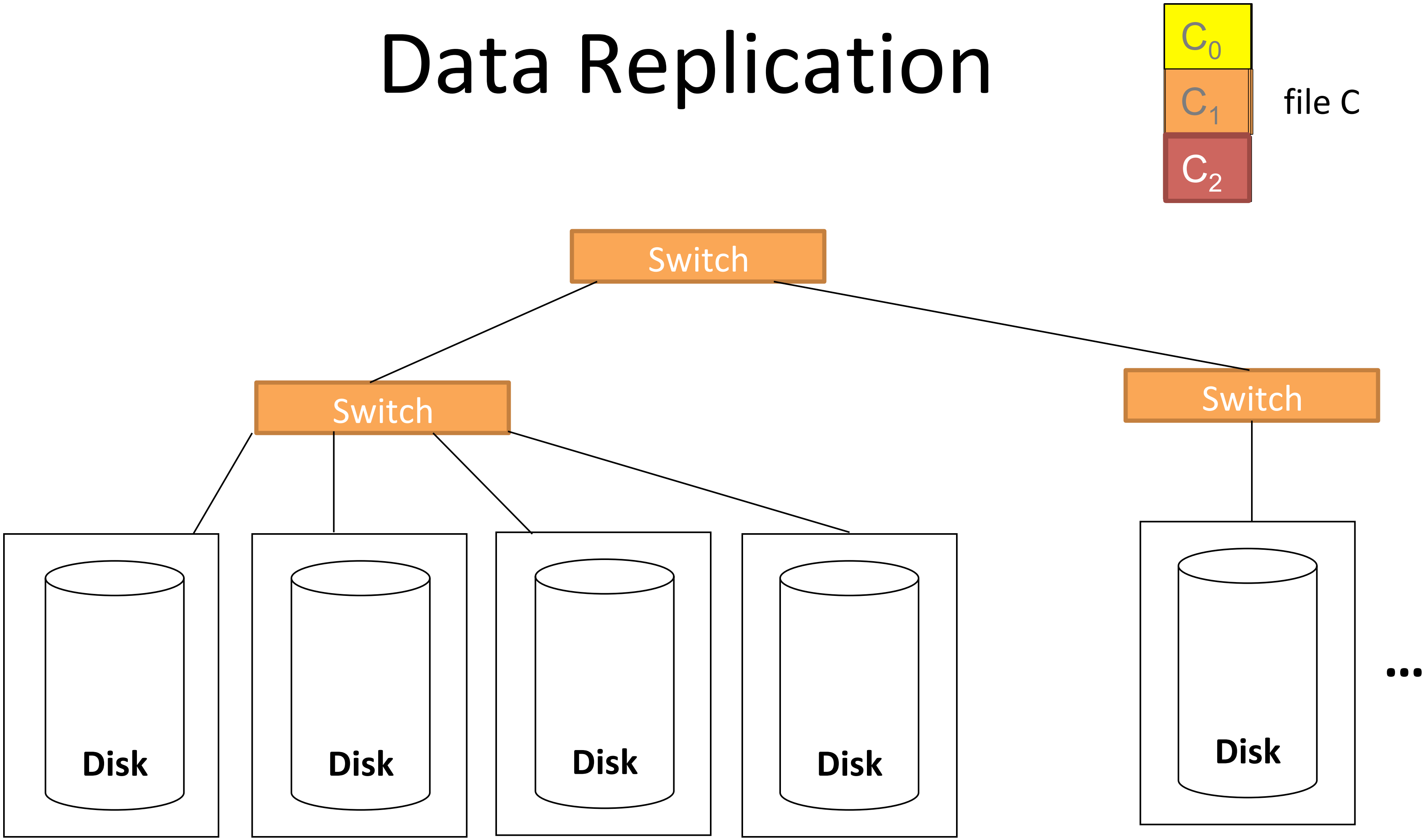
Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Challenges With Implementing M/R

- Distributing computation over a network can be non-trivial
- Data is huge and copying data over a network takes time
  - we have just seen an example
- Machines fail:
  - server may stay up 3 years (1,000 days) ; with 1000 servers expect to loose 1/day

Data Replication



Each rack contains 16-64 nodes

Distributed File System

- Chunk Servers.
  - File is split into contiguous chunks, typically 64MB.
  - Each chunk replicated (usually 2x or 3x).
  - Try to keep replicas in different racks.

- Master Node for a file.
  - Stores metadata, location of all chunks.
  - Possibly replicated.

Distributed File System

Client library for file access

- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

- **Try to send map computation where the data is**
  - **but cannot send too many jobs to the same machines, data could be moved before map is executed**
- During shuffle send all key-value pairs to the same reduce machine
  - better if closed to where the map has been done
    - but this is not always possible