Introduction to Hadoop & Map-Reduce Programming

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

C. Menichetti (IBM)
Big-Data and Data-Science :
Problems, Challenges, Use-Cases

Query Optimization

Datawarehouses : Modelling, Updating, Optimizations.

Hadoop &
Map/Reduce

27 Nov. 1CM
4 Déc. 1CM
11 Déc. 1CM
présence obligatoire aux séminaires !

C. Menichetti (IBM)
Big-Data and Data-Science :
Problems, Challenges, Use-Cases

Query Optimization

Datawarehouses : Modelling,
Updating, Optimizations.

Hadoop &
Map/Reduce

13 Nov. 1CM + 1TP
TP : 20 + 27 Nov
4 + 11 Dec

What is Hadoop ?

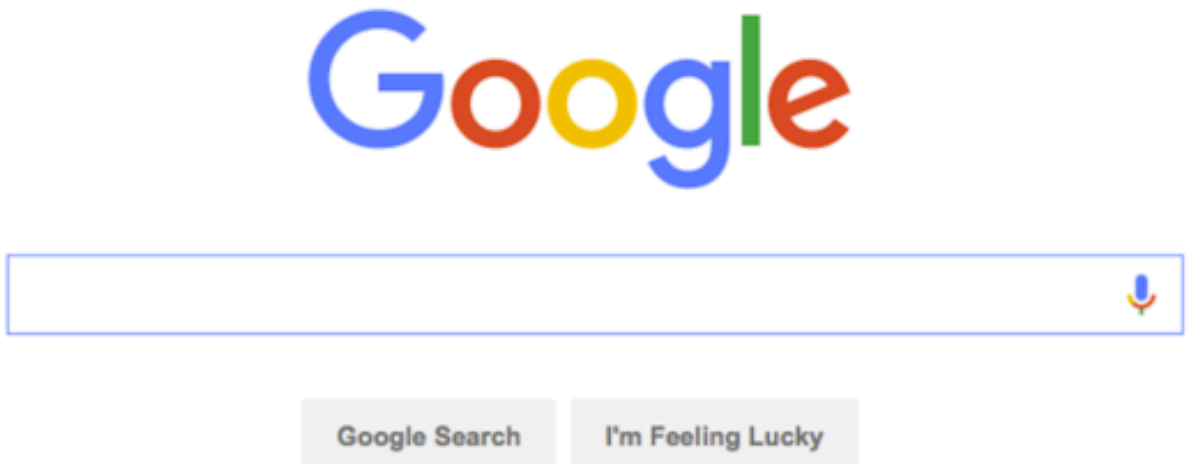
- Google'solution to solve Big Data problems by means of massive parallelization

APACHE hadoop

Apache Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.


Google
Google Search
I'm Feeling Lucky

Google

how many google searches per day ?

Google Search

I'm Feeling Lucky

The « Problem » at Google : Big-Data

- It is estimated that Google processes **5.6 billion** searches per day
  source : https://seotribunal.com/blog/google-stats-and-facts/
  - 80% queries use 1-4 words
    source
    https://www.statista.com/statistics/269740/number-of-search-terms-in-internet-research-in-the-us/
  - let’s estimate 20 Bytes to store a search (without accounting for metadata, compression, etc..)
    ~ 100 GB/day ~ 3 TB/month ~ 36 TB/year

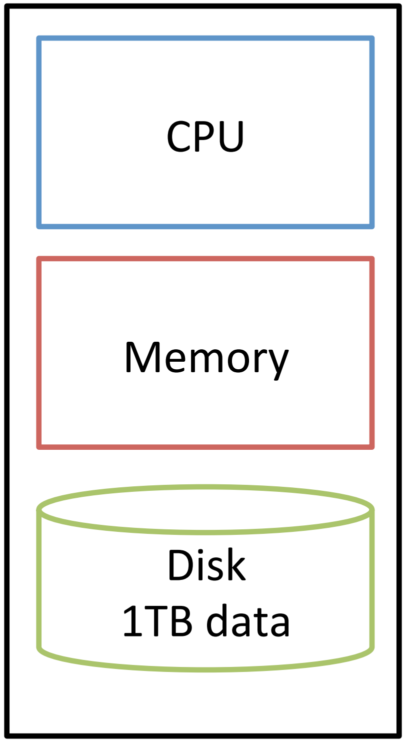
Google collects a lot of data, <u>from us</u>!

- Searches (web, images, news, blogs, etc.)
- Clicks on search results
- Web crawling
- Web analytics (package used by websites)
- Ad Service (which Ads are requested/clicked?)
- Email
- G Suite (Docs, Sheets, Slides, Calendar, Drive, etc.)
- Google Chrome
- Android OS
- Google Pixel Usage
- Google Assistant and Google Home
- Google Finance
- Youtube
- Google Translate
- Google Books
- Google Flights
- Google Maps/Earth
- Our contact network

« Google as big brother » by Randy Siegel from 10.10.2005 NY Times

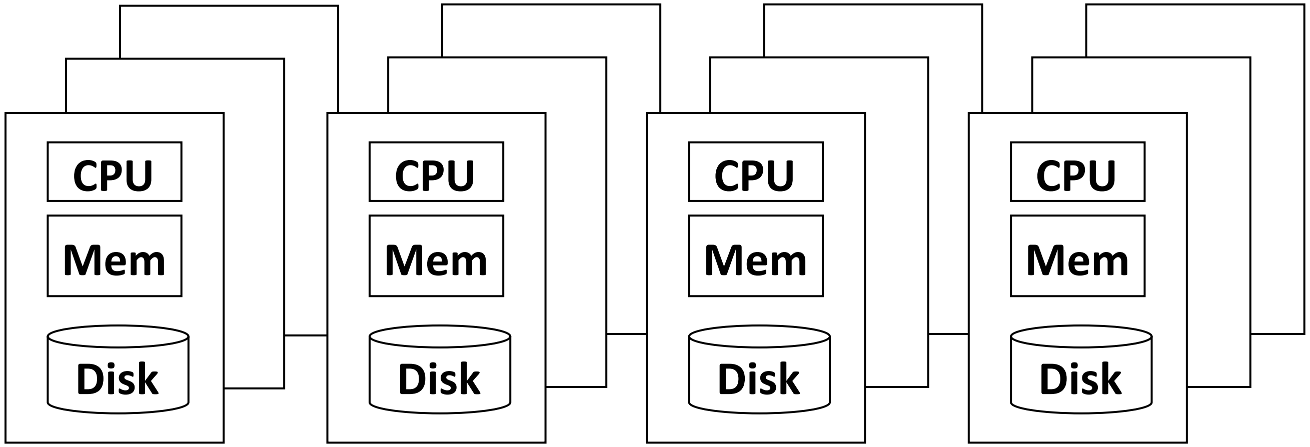
Do we really need parallelization ?

- The problem : **disk is slow** wrt memory&cpu
- Consider Single Node Architecture
- Imagine putting 1TB (10^6MB) data in 1 machine with 1 disk
- With 10^2MB/s transfer speed we can read the data in ... 10^4s = 2.5h
- Can a query wait for so long ? **Impractical**



Cluster Architecture

- Better idea : spread data over many disks !
- Run computations in parallel
  - 10 disks **read** 1TB in 16min, 100 in 1.6min, 1000 in 160ms
- So yes : we need parallelization.



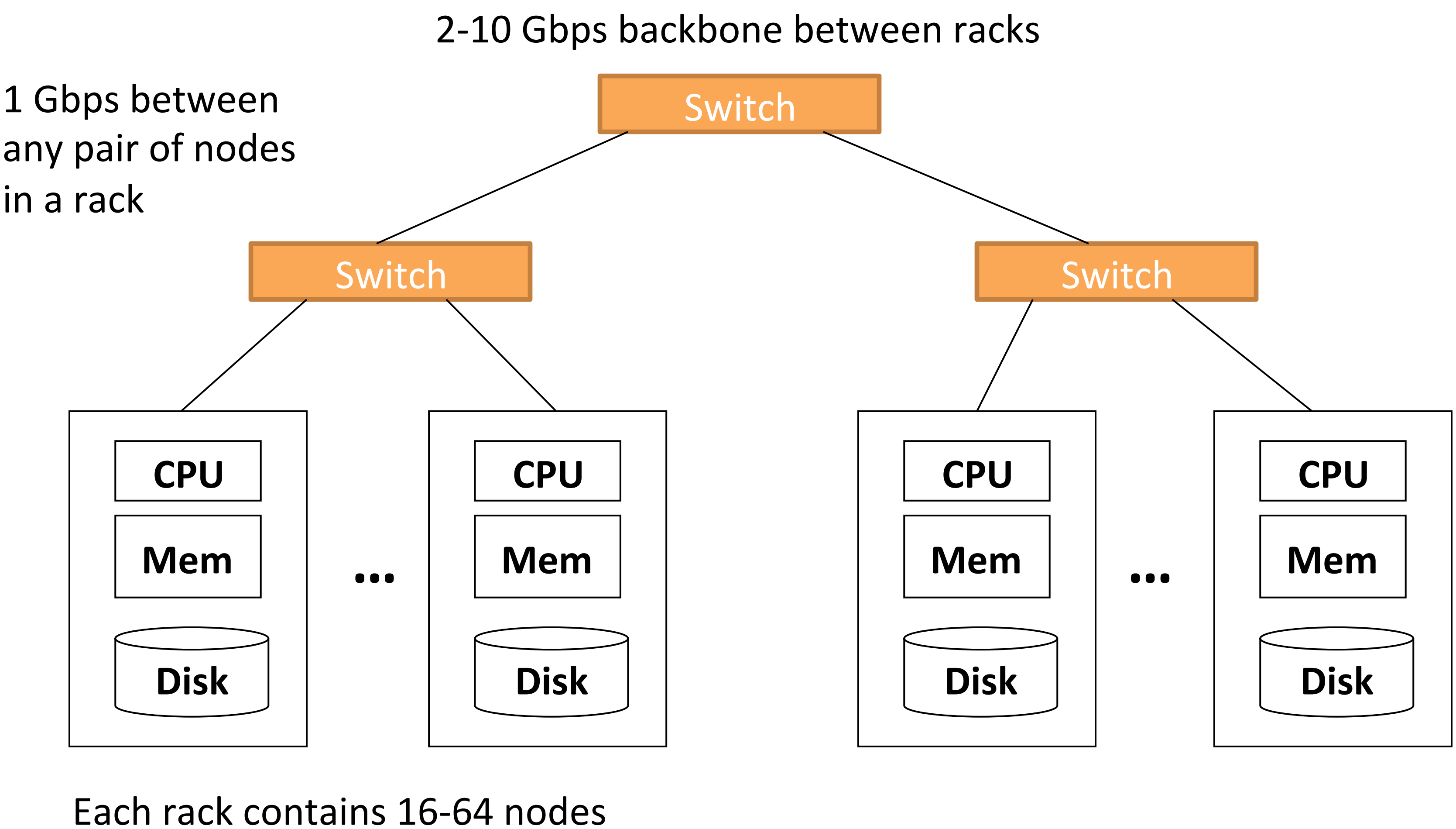
In 2011 it was estimated that Google had 1M machines, http://bit.ly/Shh0RO

Big-Data Cluster

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

- Cluster of commodity Linux nodes (no hig-end machines)
- 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

Cluster Architecture



Did we really need Hadoop ?

- Why Google does not use Parallel Database or Datawarehouse ?
  - Well it does, but not a traditional one!
- Build a Database/Datawarehouse **only if you master the business !**
  - you know the data (and are able to define a model for it)
  - you know the queries you are likely to ask
- Google wanted also to run **exploratory** analysis
  - to discover what the data can tell us ?
  - analyzing data sets to summarize their main characteristics
  - to understand which queries should be asked on this data

Did we really need Hadoop ?

- Massive data : parallel DB / DW is also
  - Expensive to **configure** and **maintain** **(DBA)**
  - Expensive to **prepare** and **load data** for it (ETL)
  - Constrained to **SQL** (unpractical for analyzing unstructured data like search logs, text, etc)

So let there be Hadoop !

- **Hadoop** : a software stack for big-data analysis on clusters.

Key components :

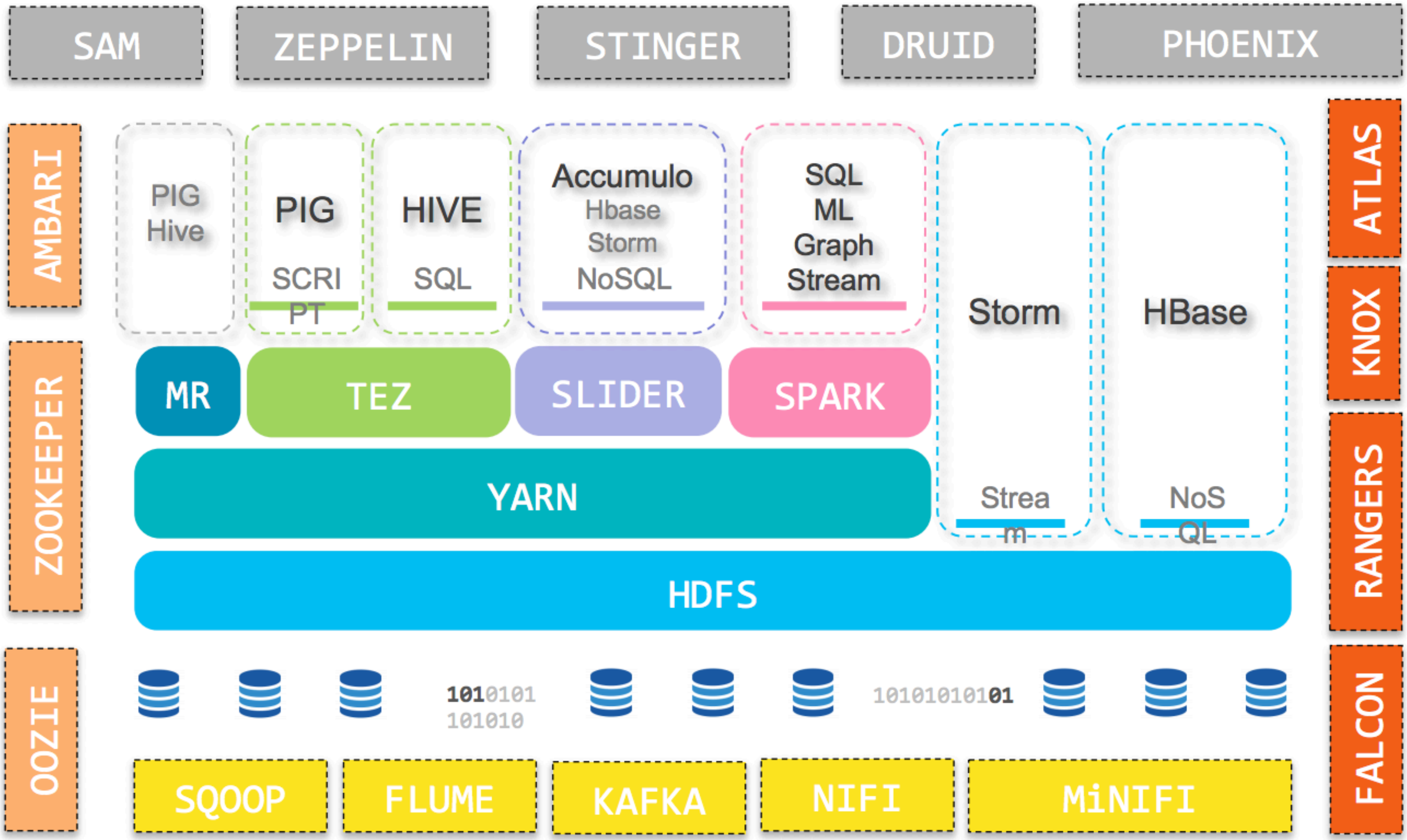
- **HDFS** : distributed file system (holds data)

- **Map-Reduce** : programming paradigm (compute)

Challenges for Hadoop

- Distributing computation over a network can be non-trivial
  - scheduling of tasks (which machine does what)
  - data distribution (moves processes to data)
  - synchronization (collects partial results, sorts, and shuffles intermediate data)

- Machines fail:
  - errors and faults (and restarts tasks if needed)
  - server may stay up 3 years (1,000 days) ;
    with 1000 servers expect to loose 1/day


SAM
ZEPPELIN
STINGER
DRUID
PHOENIX
AMBARI
ZOOKEEPER
OOZIE
PIG
Hive
PIG
SCRIPT
HIVE
SQL
Accumulo
Hbase
Storm
NoSQL
SQL
ML
Graph
Stream
Storm
Stream
HBase
NoSQL
MR
TEZ
SLIDER
SPARK
YARN
HDFS
1010101
101010
10101010101
SQOOP
FLUME
KAFKA
NIFI
MiNIFI
ATLAS
KNOX
RANGERS
FALCON

Hadoop and DW are complementary!

Requirement		Data Warehouse	Hadoop
Low latency, interactive reports, and OLAP		●	
ANSI 2003 SQL compliance is required		●	
Preprocessing or exploration of raw unstructured data	Googles Logs		●
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	Not only SQL !		●
CPU intense analysis		●	●
System, users, and data governance		●	
Many flexible programming languages running in parallel	Not only SQL !		●
Unrestricted, ungoverned sand box explorations	Not only SQL !		●
Analysis of provisional data	Outside a DBMS		●
Extensive security and regulatory compliance		●	
Real time data loading and 1 second tactical queries		●	●*

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

Hadoop and DW are complementary!

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	●	●*

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

Hadoop and DW are complementary!

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	●	●*

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

MAP-REDUCE

Map-Reduce means two things

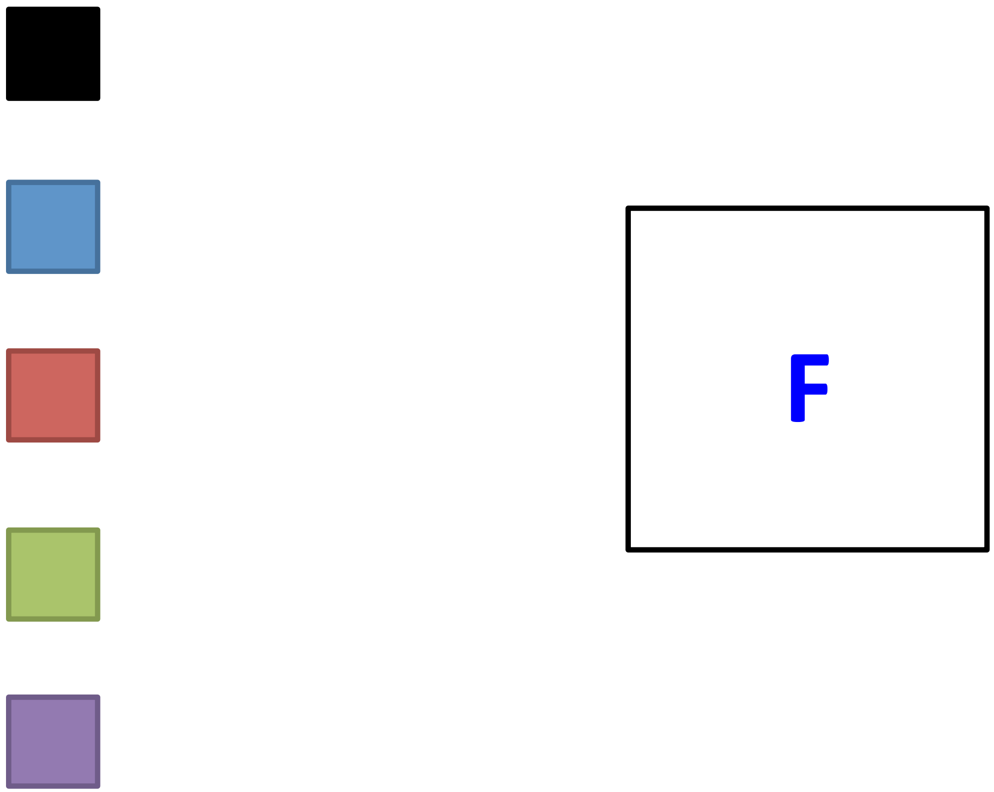
1. An abstract model of computing

2. A paradigm of programming

We will look at both

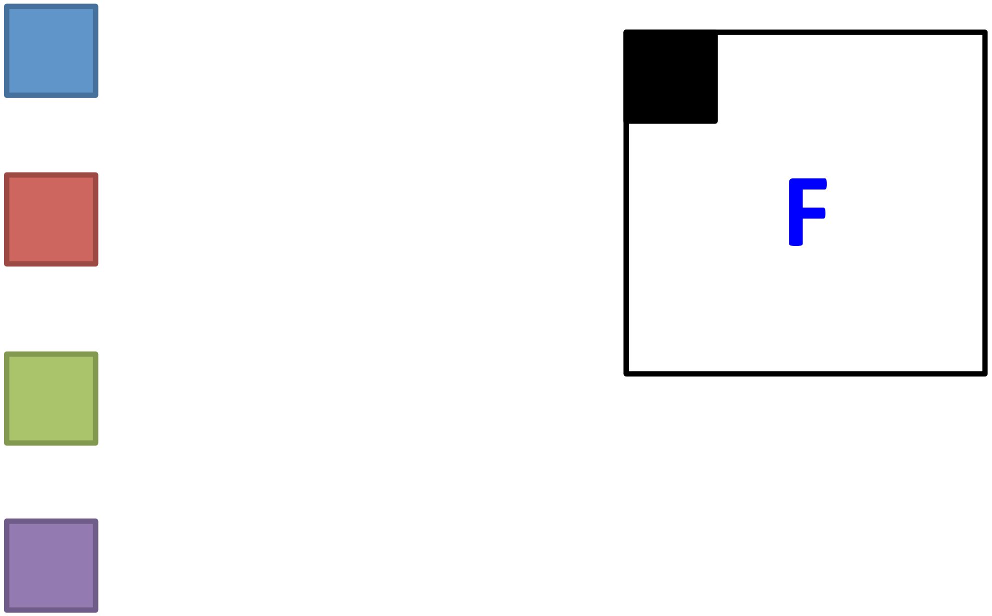
Sequential Computing

- iterate over inputs to compute a given function



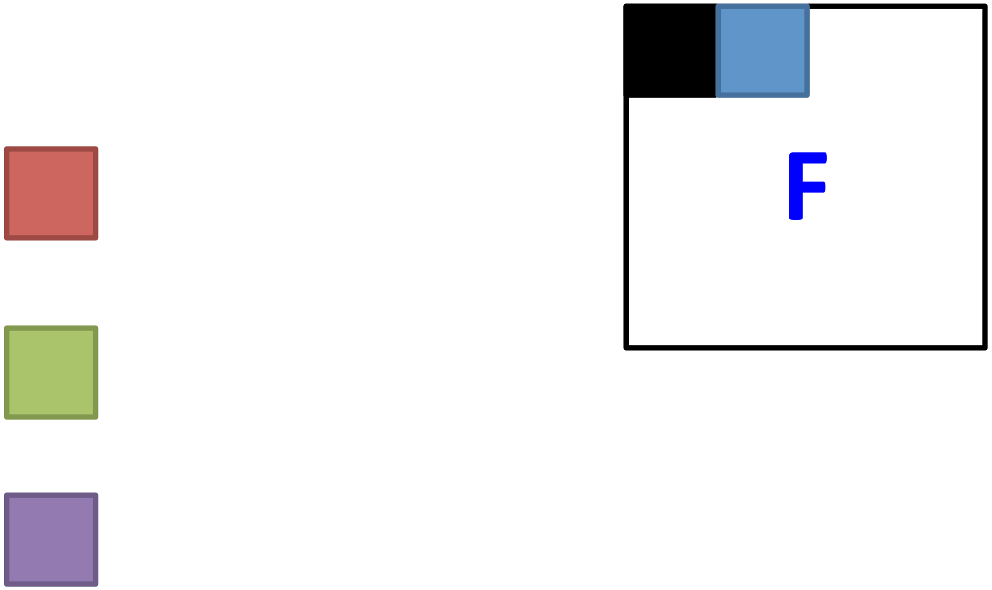
Sequential Computing

- iterate over inputs to compute a given function



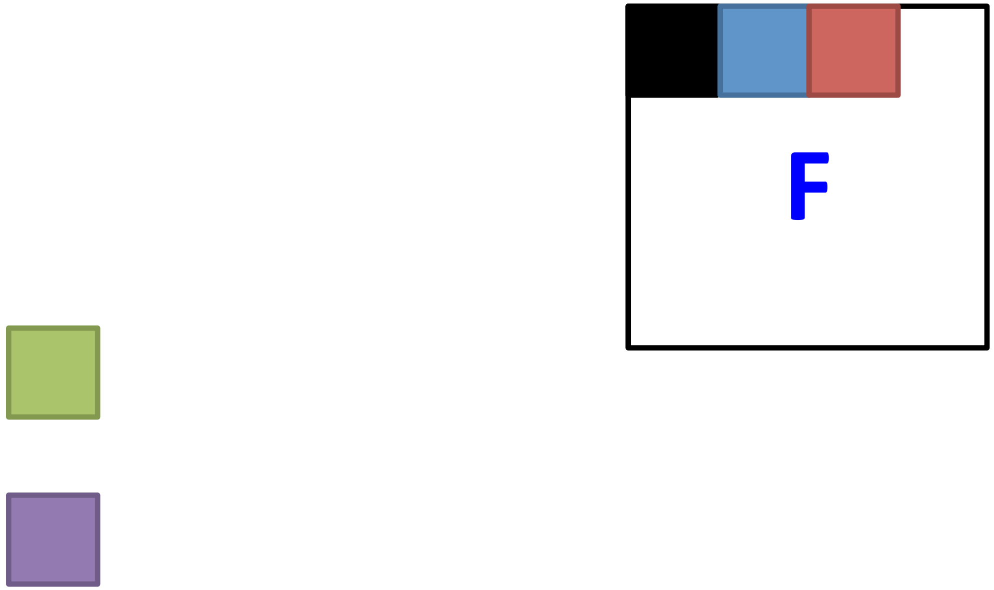
Sequential Computing

- iterate over inputs to compute a given function



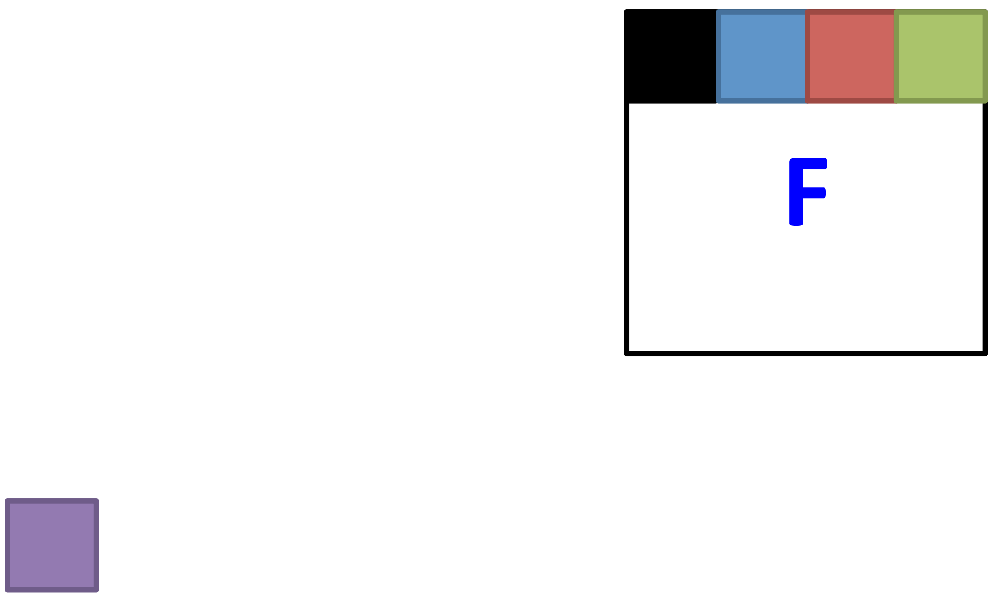
Sequential Computing

- iterate over inputs to compute a given function



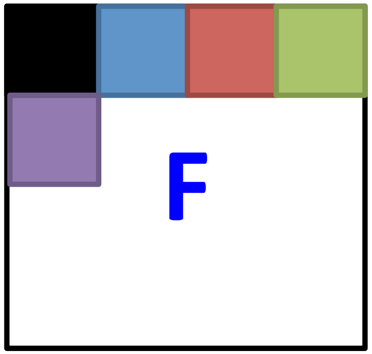
Sequential Computing

- iterate over inputs to compute a given function



Sequential Computing

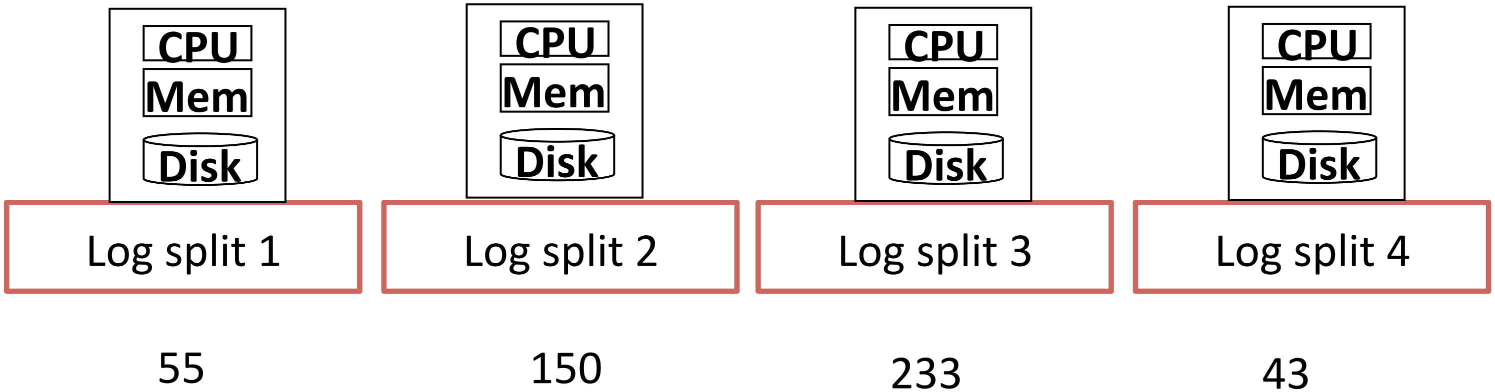
- iterate over inputs to compute a given function



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

Perfectly Parallelizabile Problems

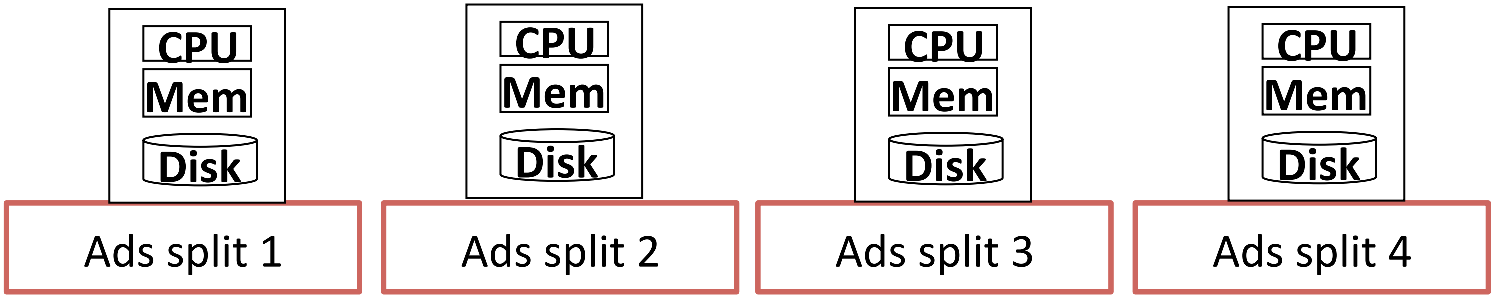
- Easy case : no effort is needed to separate the problem into a number of parallel tasks.
- Ex : count Google searches talking of **Elections**



Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : profit made with **Ads** <u>**for each sport**</u>

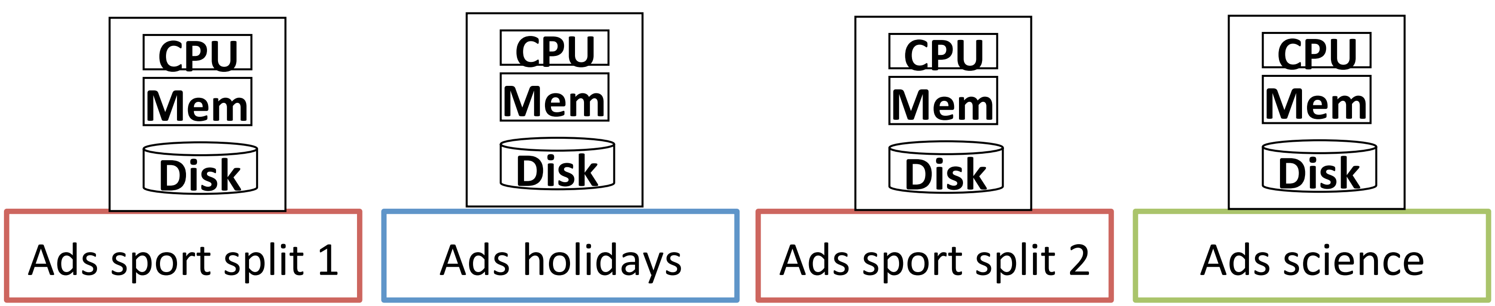


Here has to compute many values (1 x sport)
Load balancing issue : nodes with more sport data work more than other nodes

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : profit made with **Ads** **for each sport**

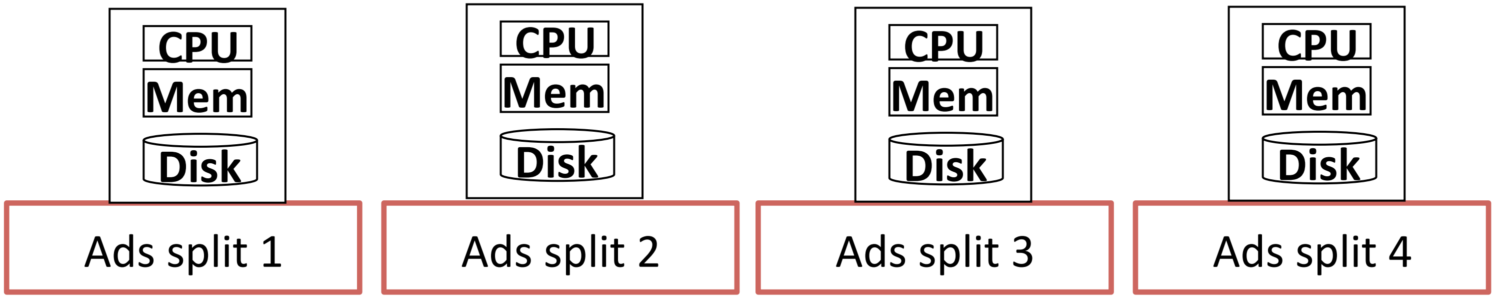


Perfectly parellelizable if Ads were already splitted by type
Also, only nodes with Ads sport will be used. The other could stay inactive.

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : profit made with **Ads** for each month

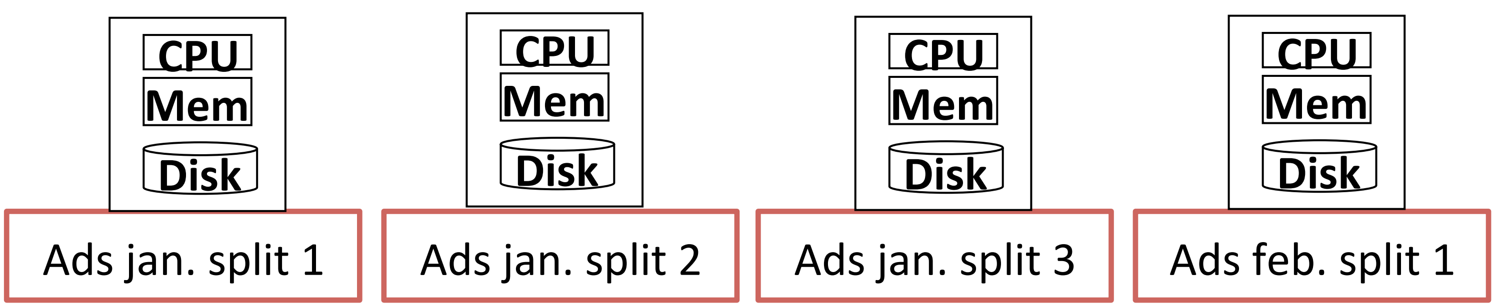


Here has to compute many values (1 x month)
Load balancing issue : nodes with more sport data work more than other nodes

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : profit made with **Ads** for each month

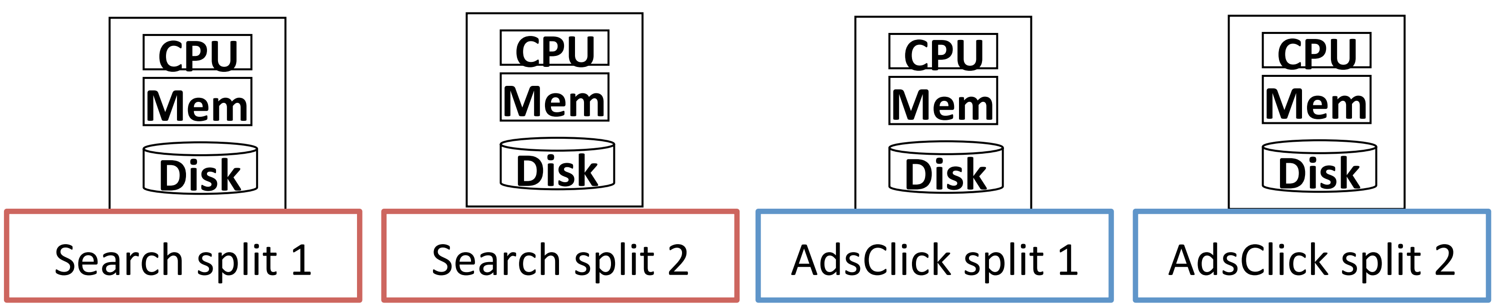


Perfectly parellelizable if Ads were already splitted by month
Also, only nodes with Ads sport will be used. The other could stay inactive.

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : **#Clicks** for Ads on most **searched** items

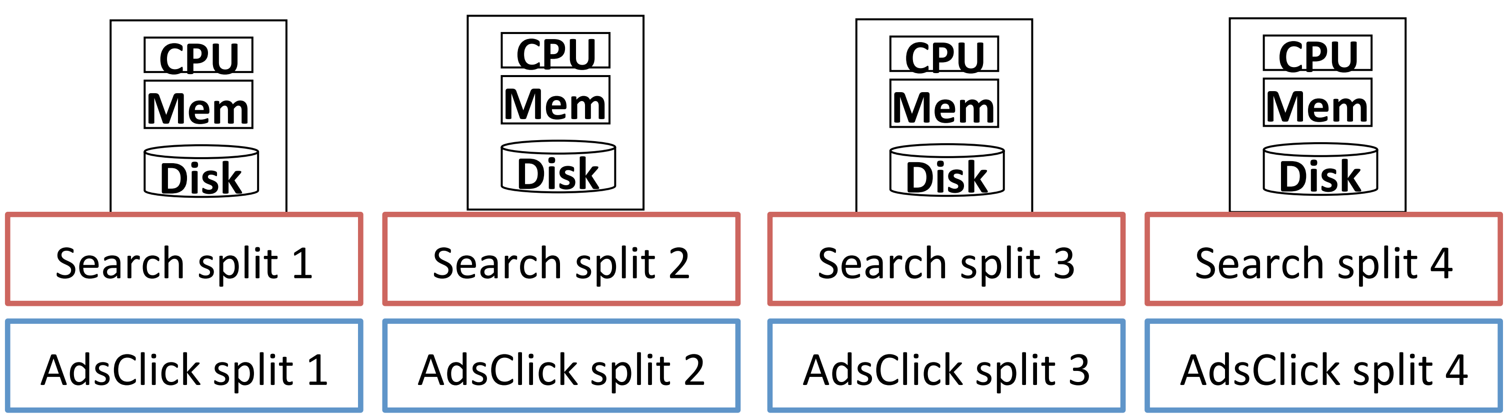


Impossible to parallelize : every node does not have engough information

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : **#Clicks** for Ads on most **searched** items

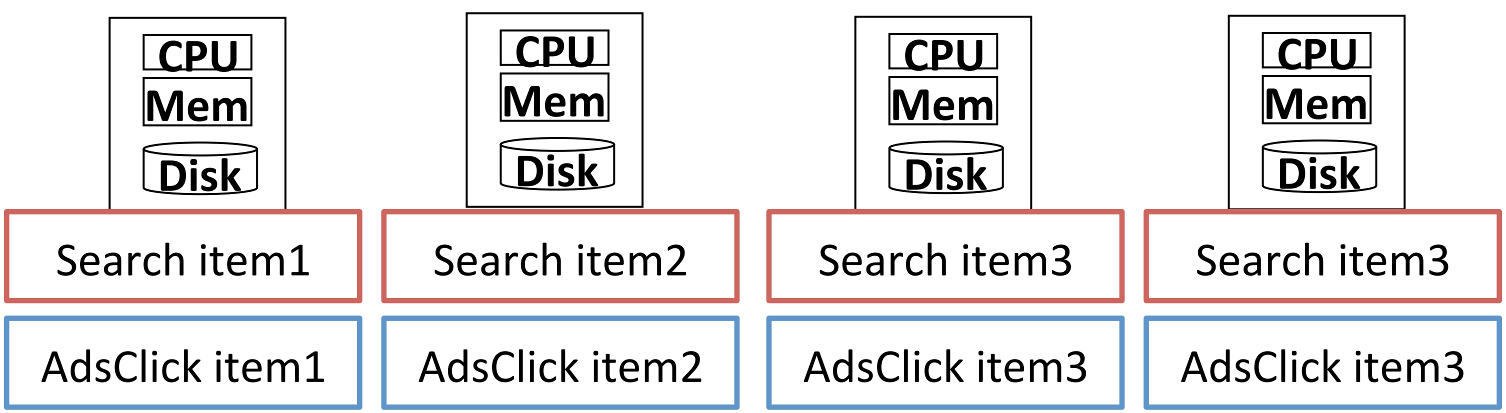


A bit better but still does not work, an item can be in two different splits…

Perfectly Parallelizabile Problems

- Easy case : no effort is needed to separate the problem into a number of parallel tasks.

- Ex : **#Clicks** for Ads on most **searched** items



If data were like that, the problem would be perfectly parallelizable.

Not problems are perfectly parallelizable

- Unfortunately, many interesting problems introduce **dependencies** and **comminications**
  - #Clicks for Ads on most searched items
  - Find all flight paths from any two european capitals
  - Find most similar users
  - Matrix multiplication (key for machine learning)

So what to do ?

Jeffrey Dean & Sanjay Ghemawat [OSDI 2004]

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

M/R Computing Model

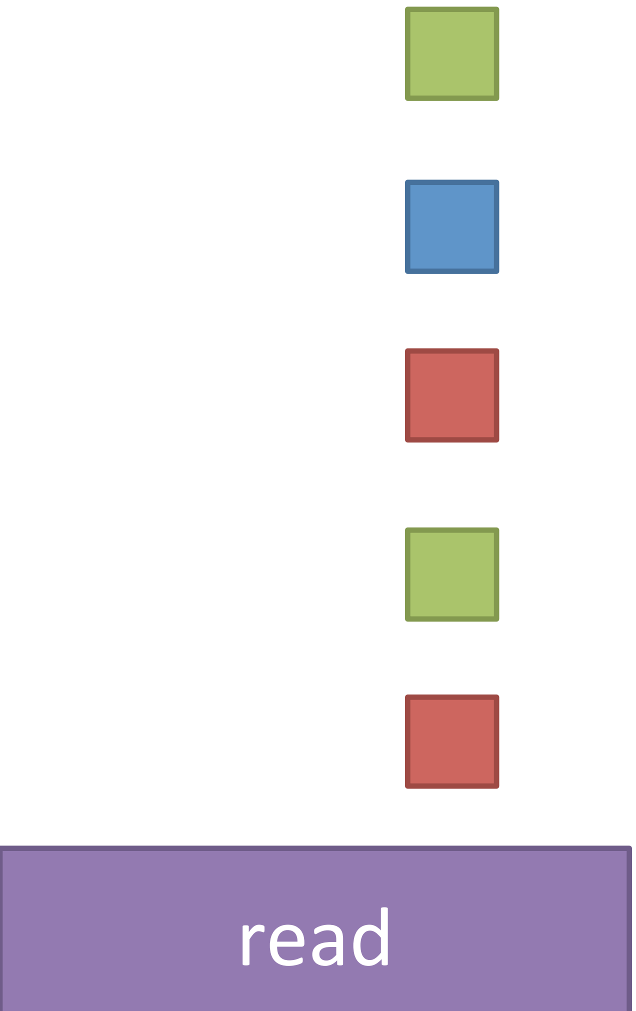
1. **Read** the inputs

2. **Regroup**-the inputs

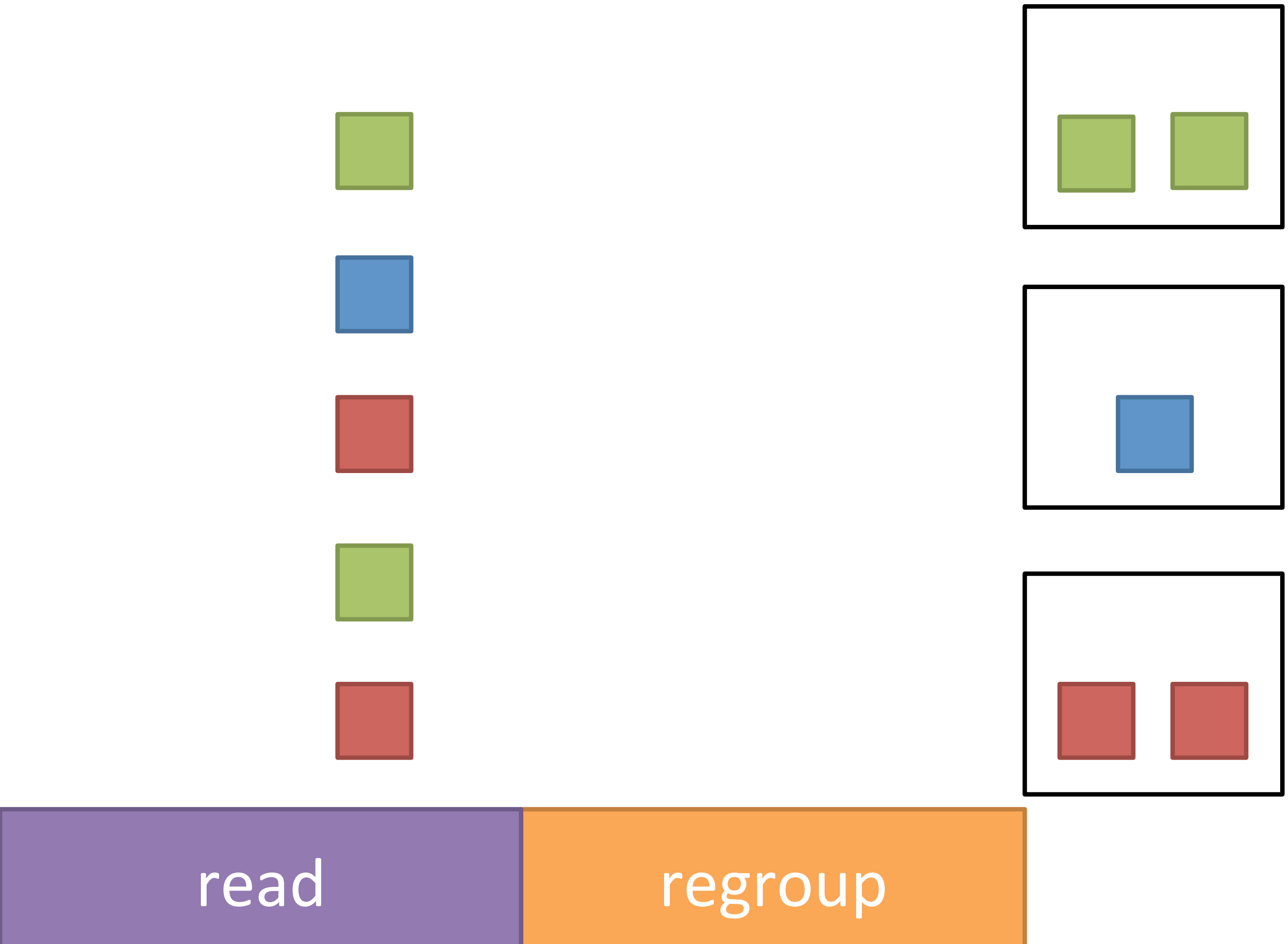
3. **Evaluate** a function on the regrouped inputs

…and all of these can be done in parallel !

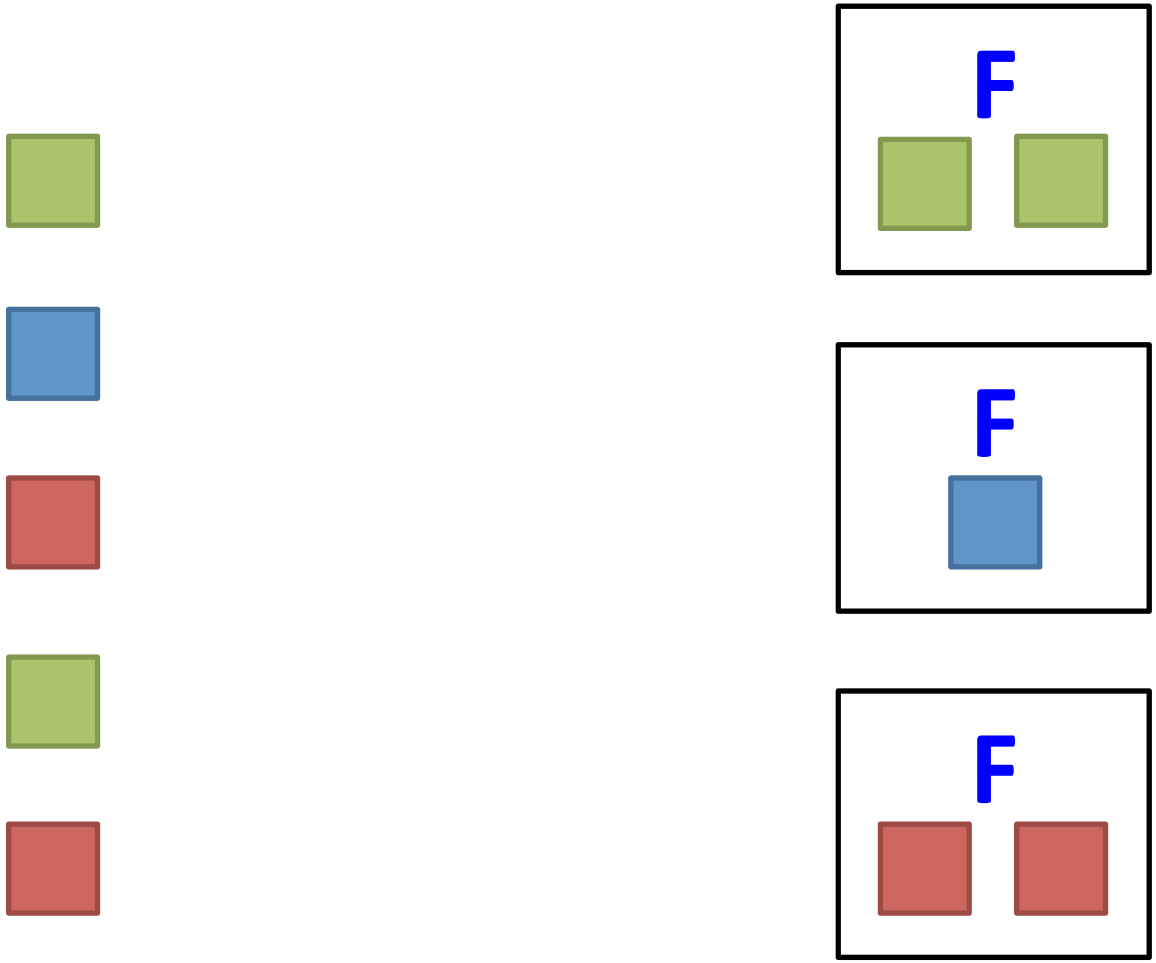
M/R Computing Model



M/R Computing Model



M/R Computing Model



read | regroup | evaluate

M/R Computing Model

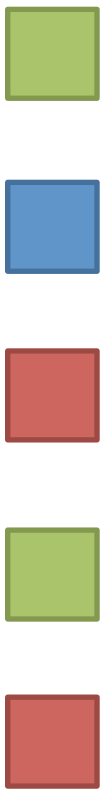
Take a set of inputs (files, tables, text…) and a function **F**

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)*

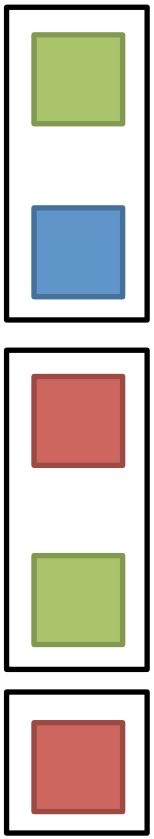
3. **Evaluate** the fuction on the new groups
   - this is called the REDUCE phase
   - again, this can be done in parallel *(according to where data is sent)*

M/R Computing Model



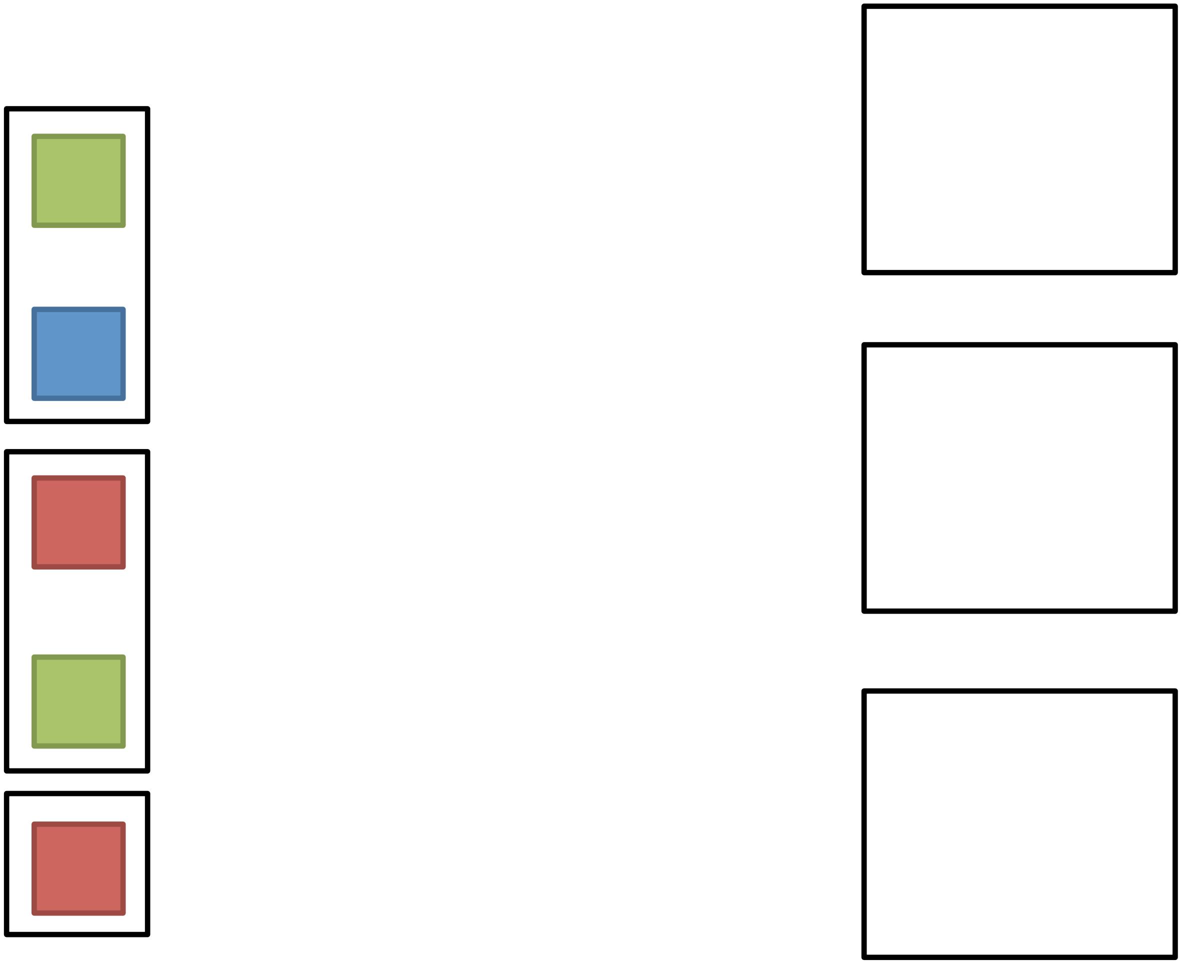
MAP : read batches of inputs

M/R Computing Model



MAP : read batches of inputs

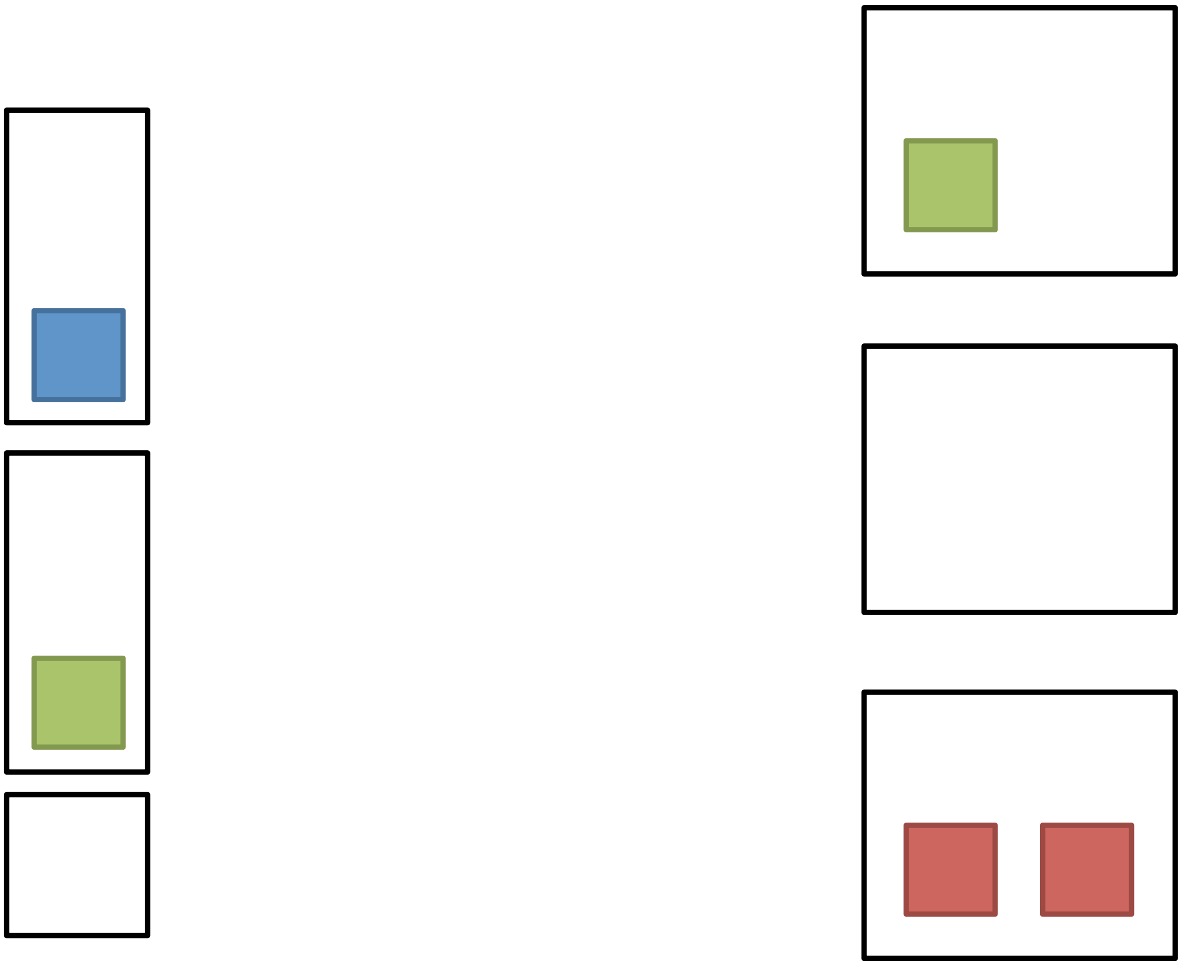
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

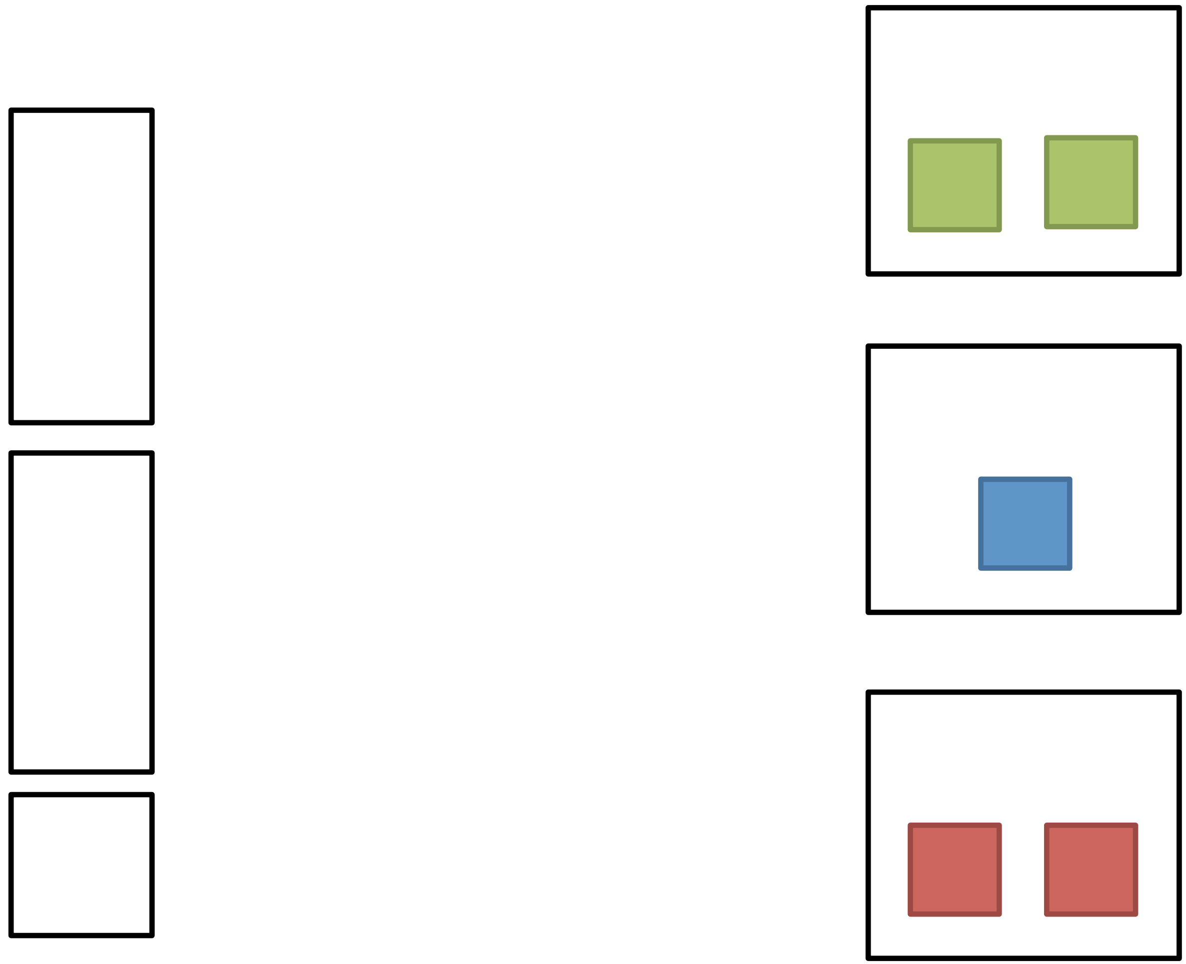
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

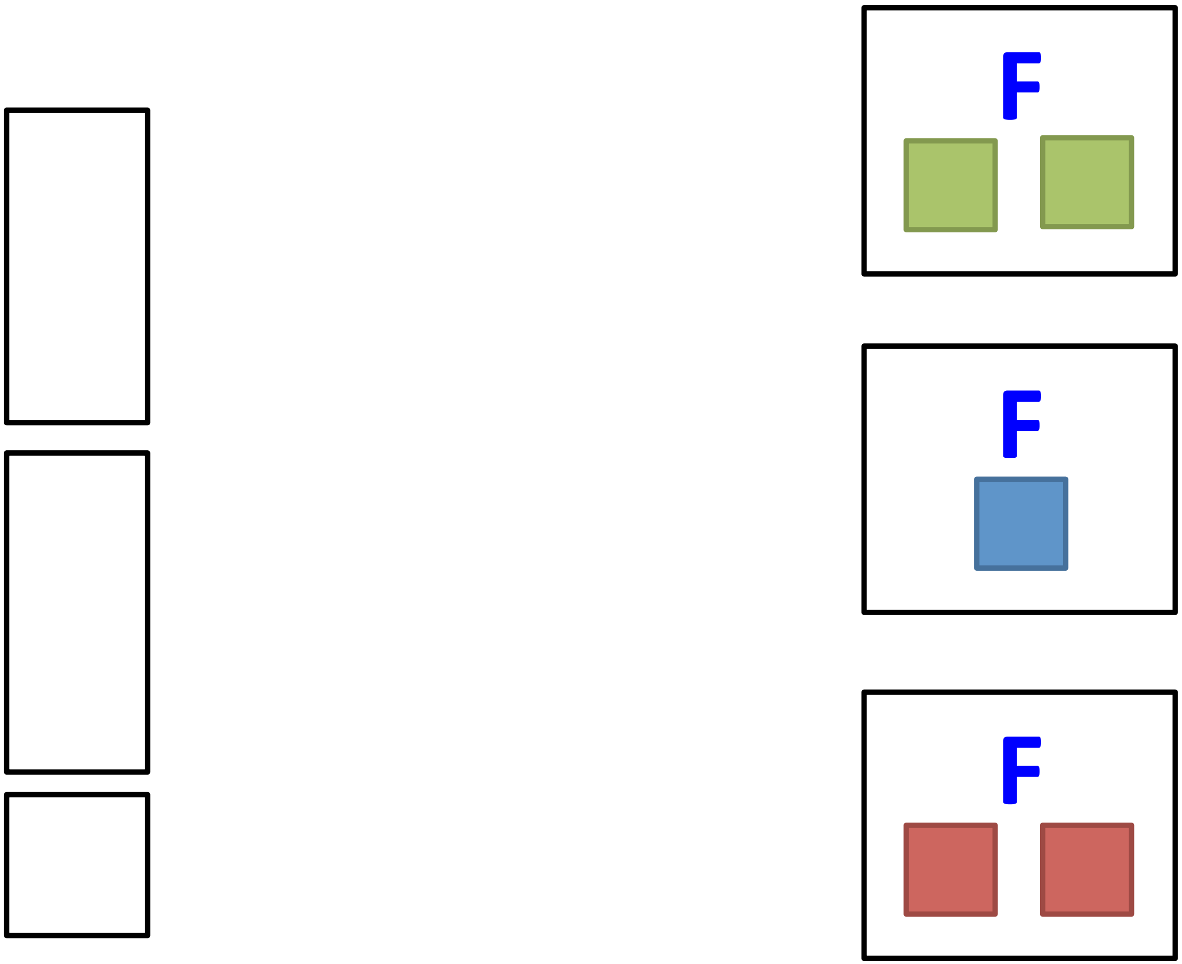
M/R Computing Model



MAP : read batches of inputs

SHUFFLE : regroup the inputs

M/R Computing Model

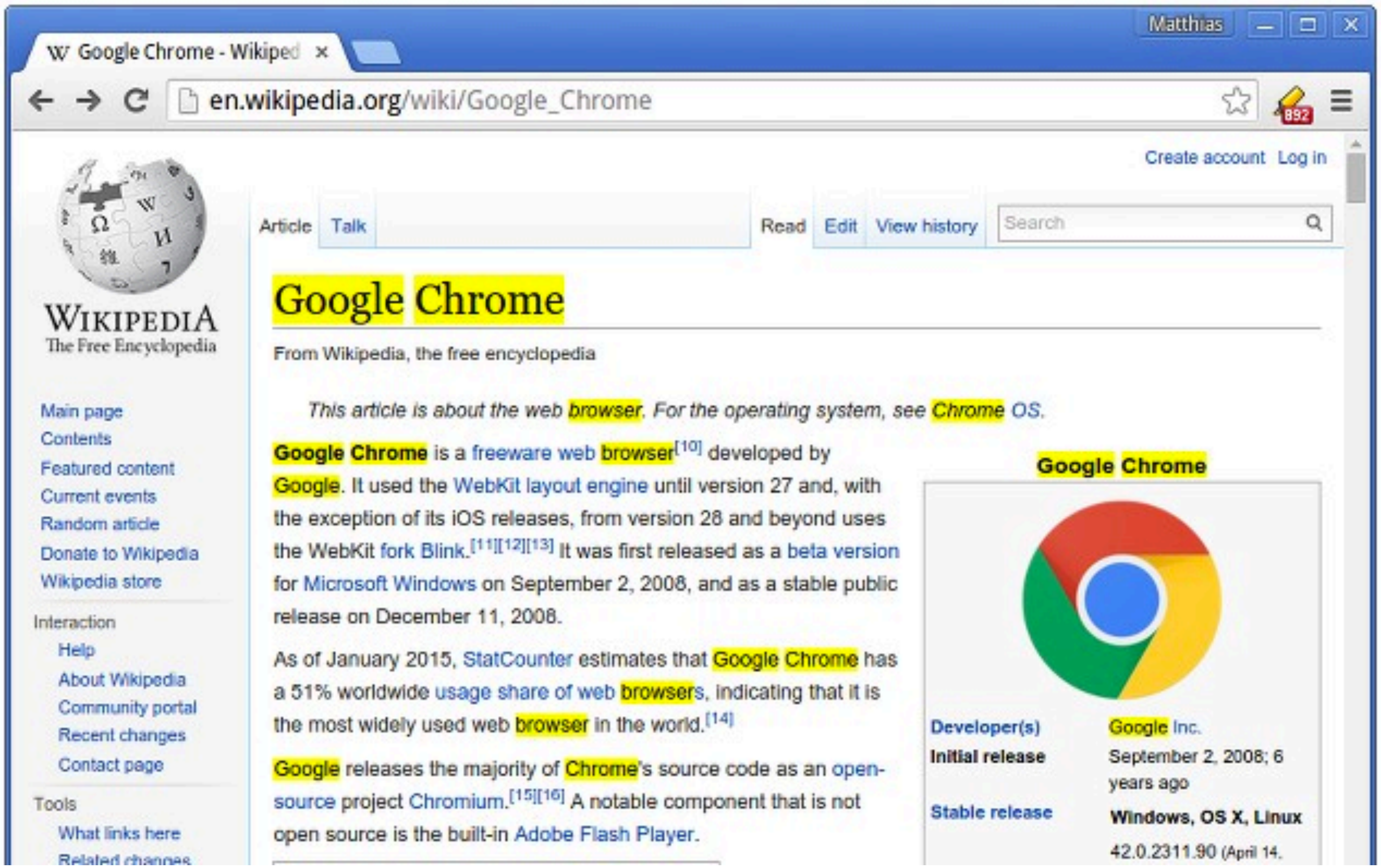


MAP : read batches of inputs

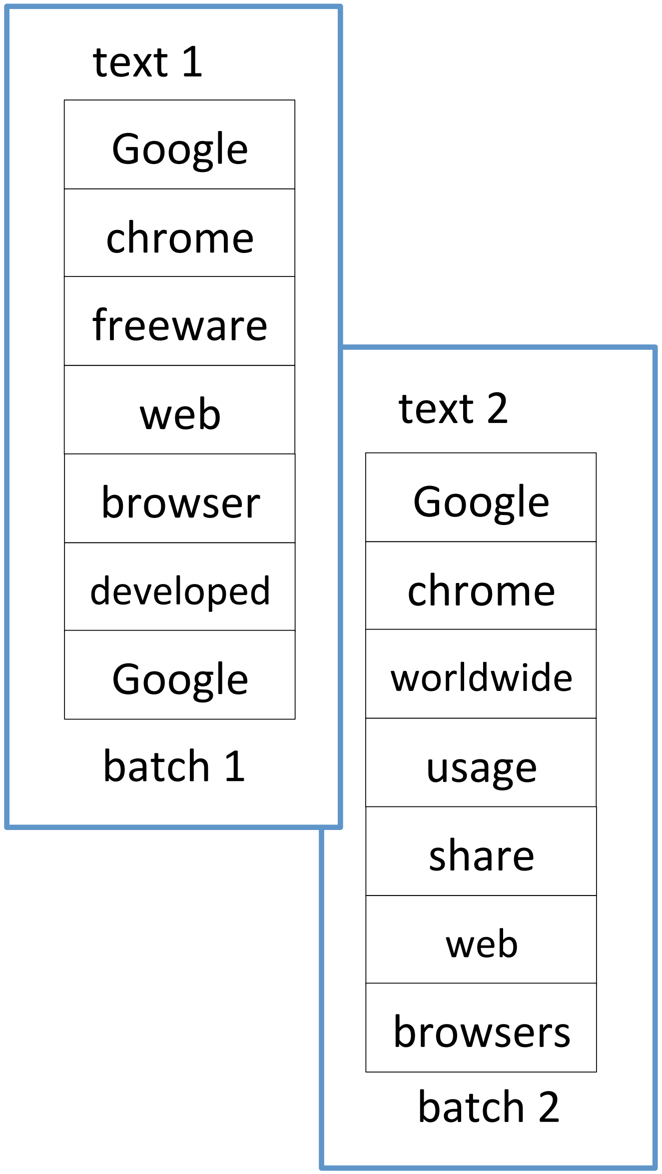
SHUFFLE : regroup the inputs

REDUCE : evaluate the function

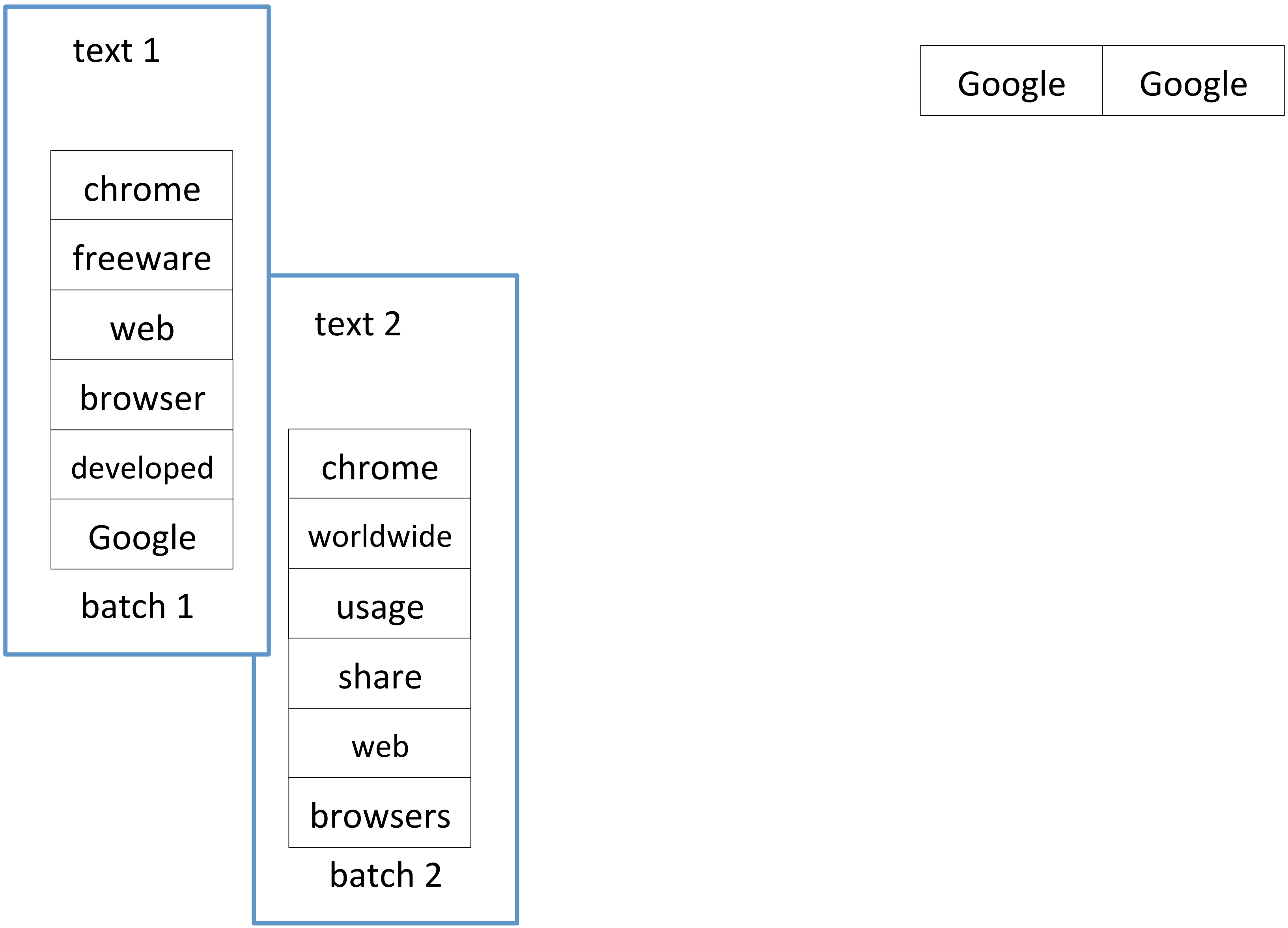
Text Mining : Keyword Count



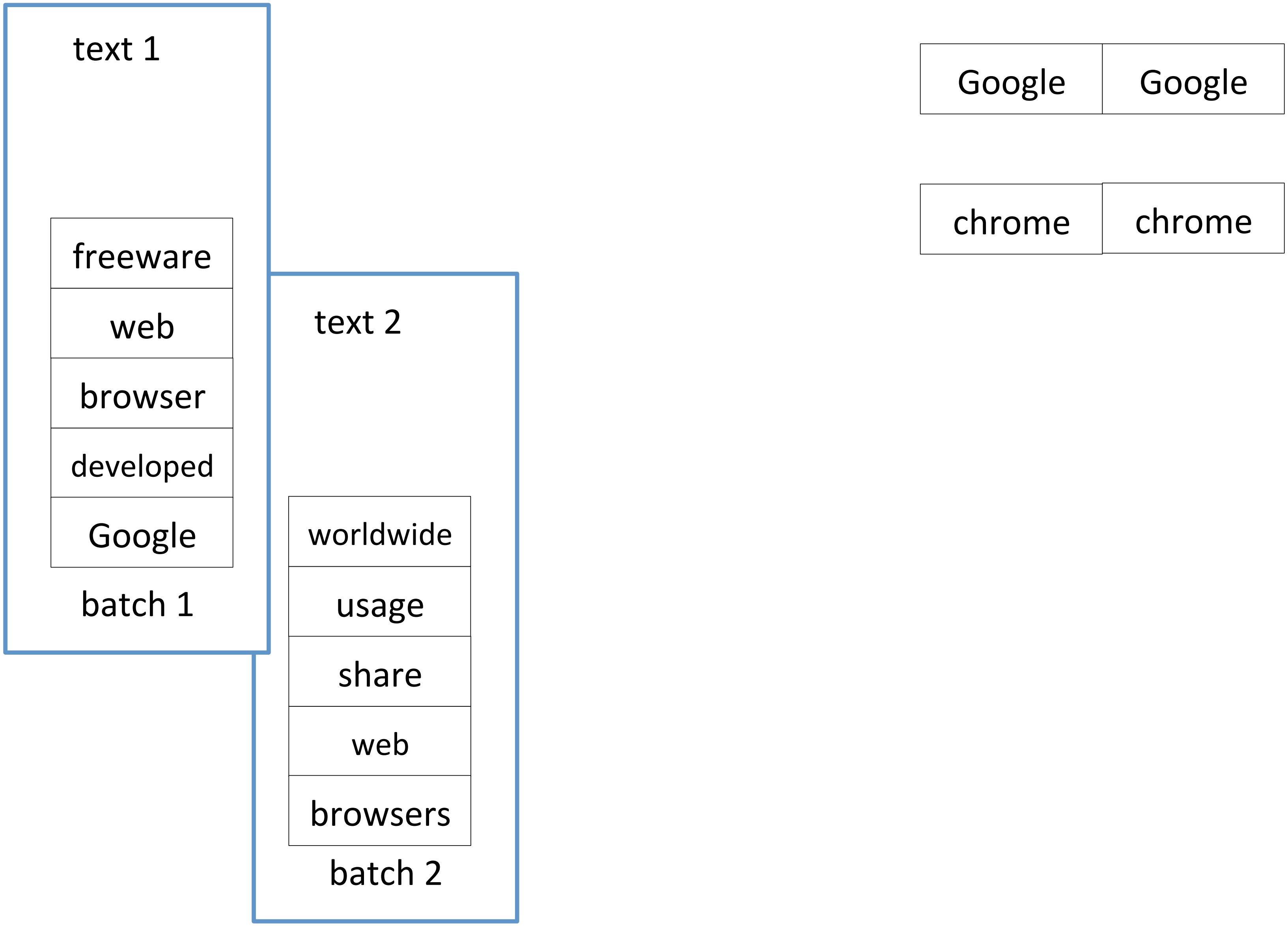
M/R Computing Model



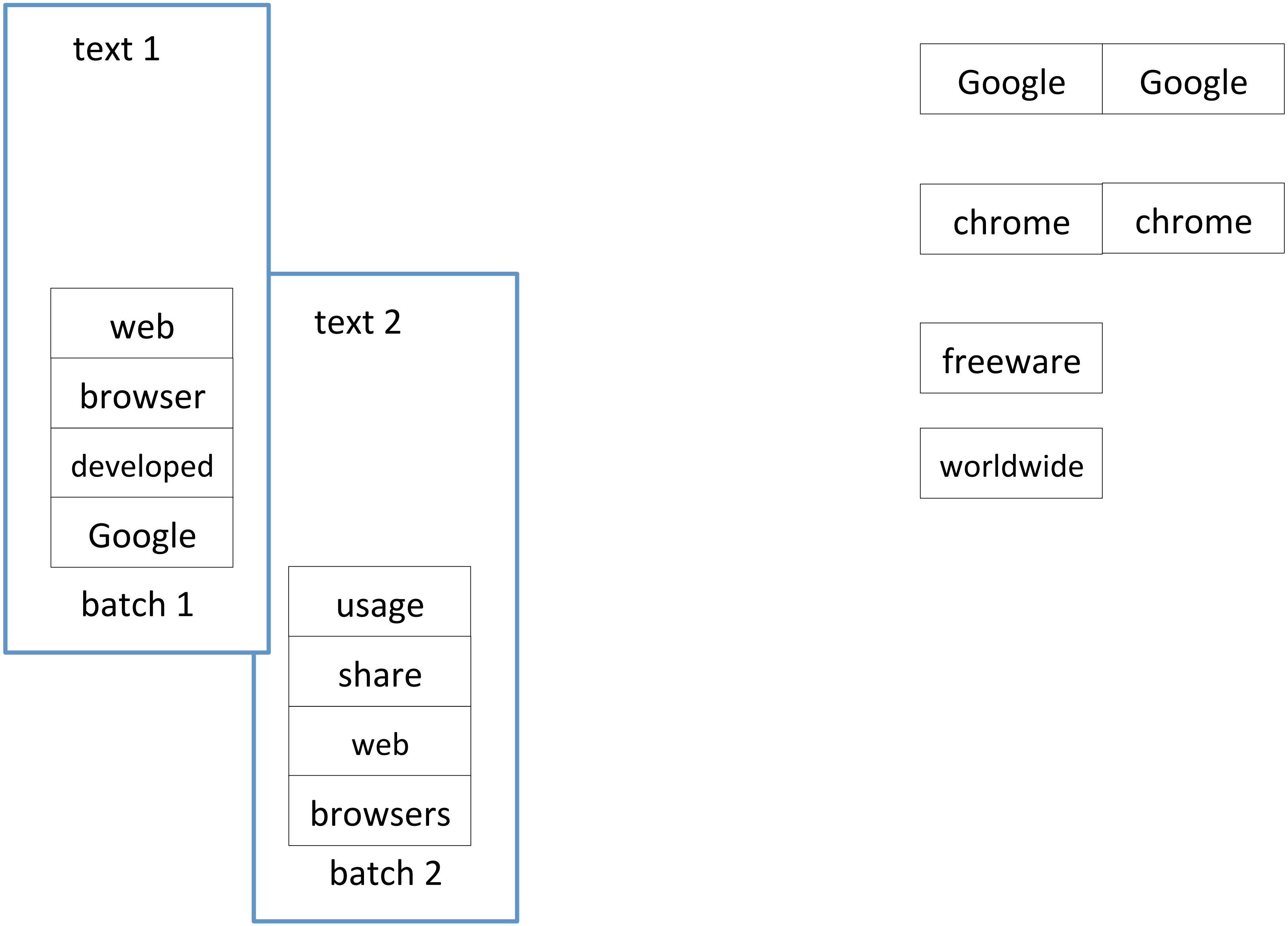
M/R Computing Model



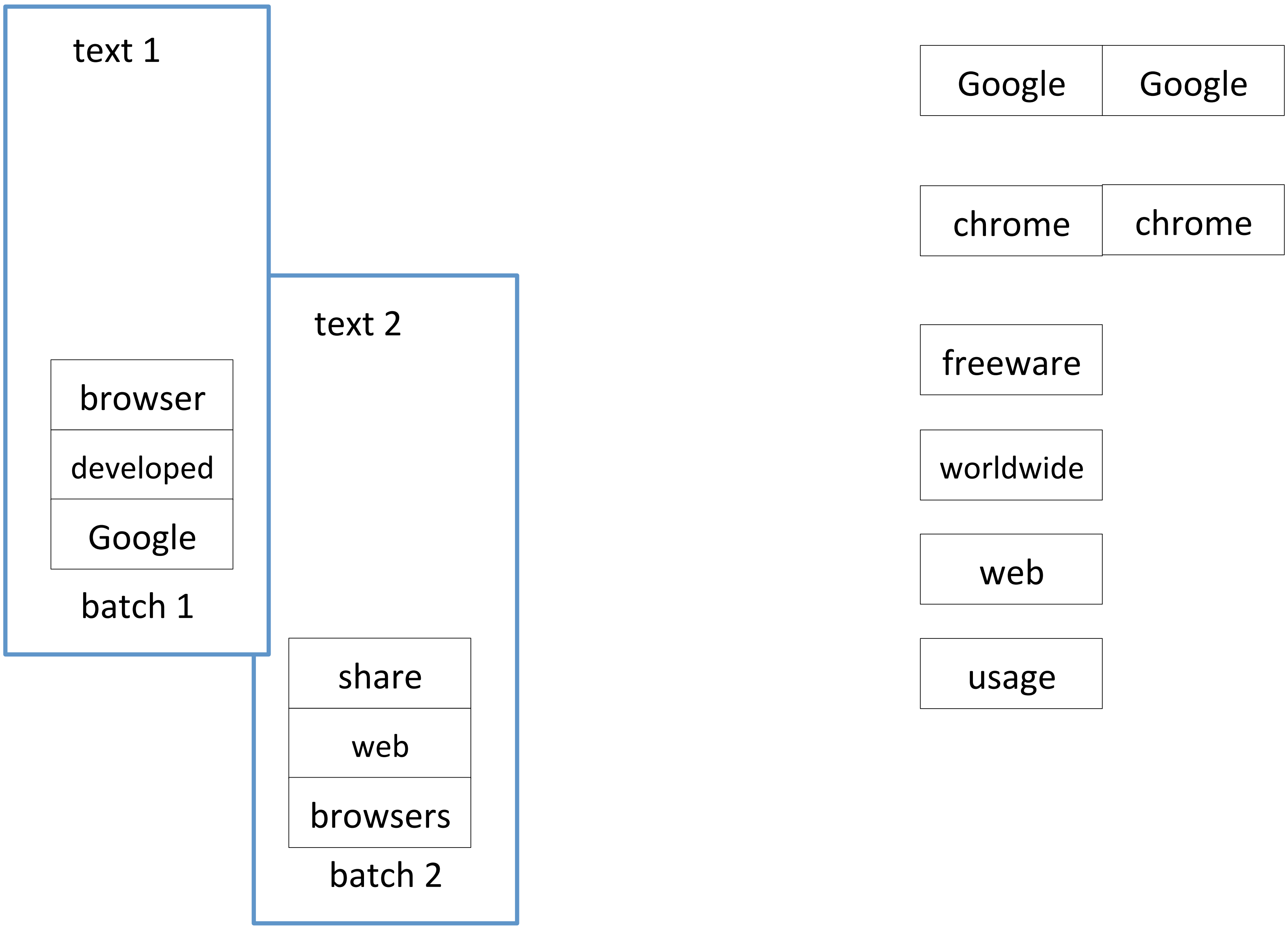
M/R Computing Model



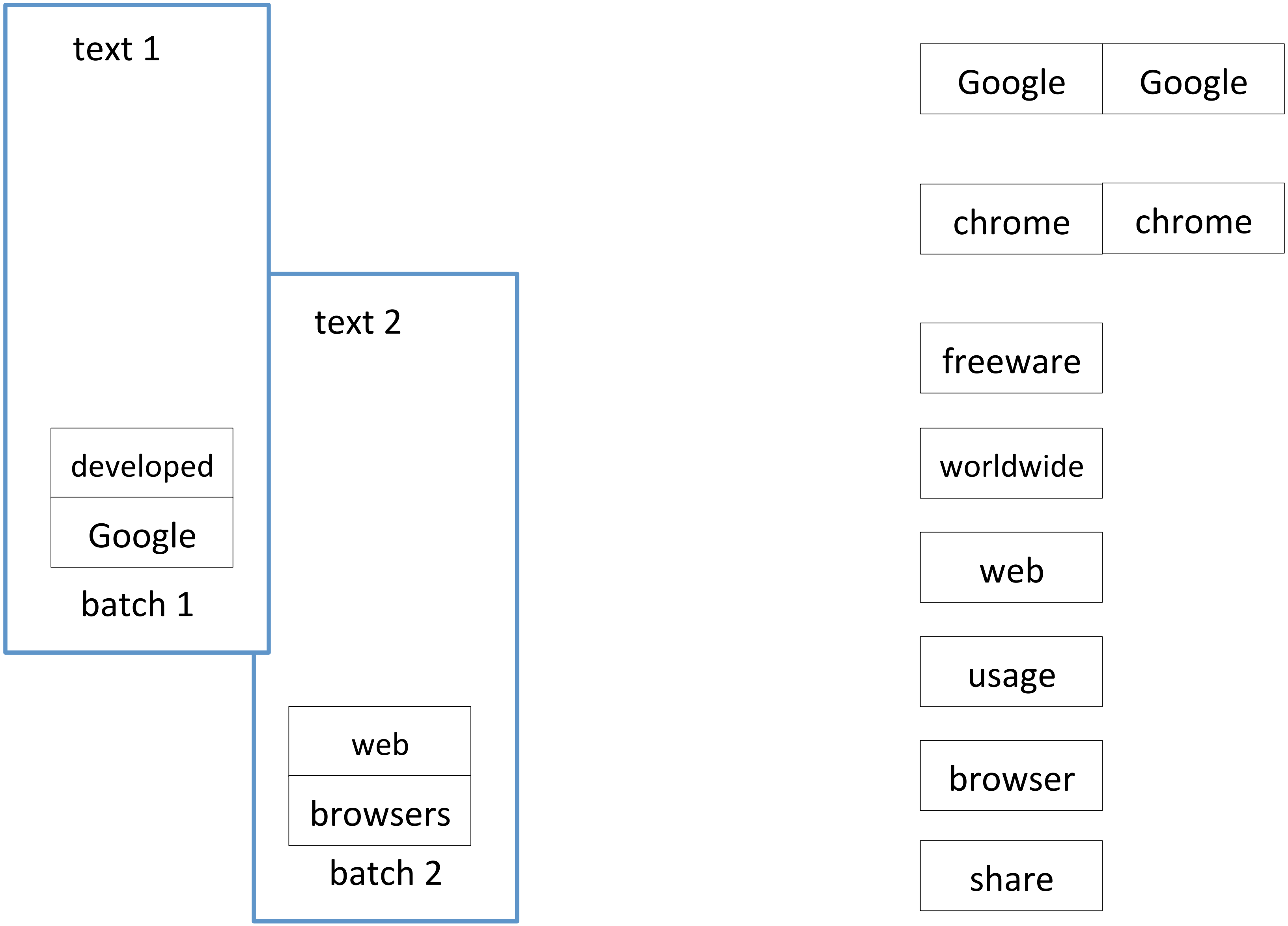
M/R Computing Model



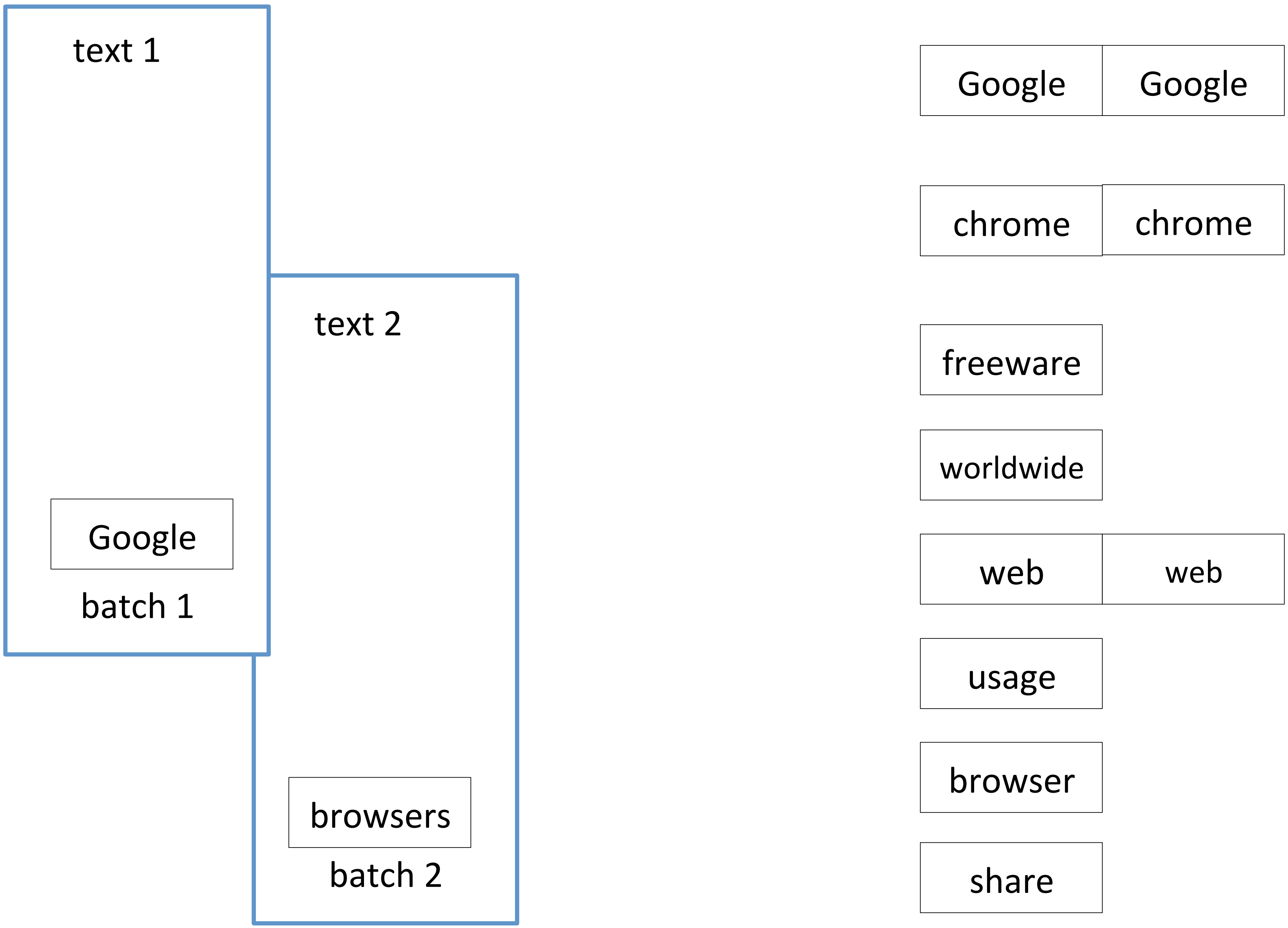
M/R Computing Model



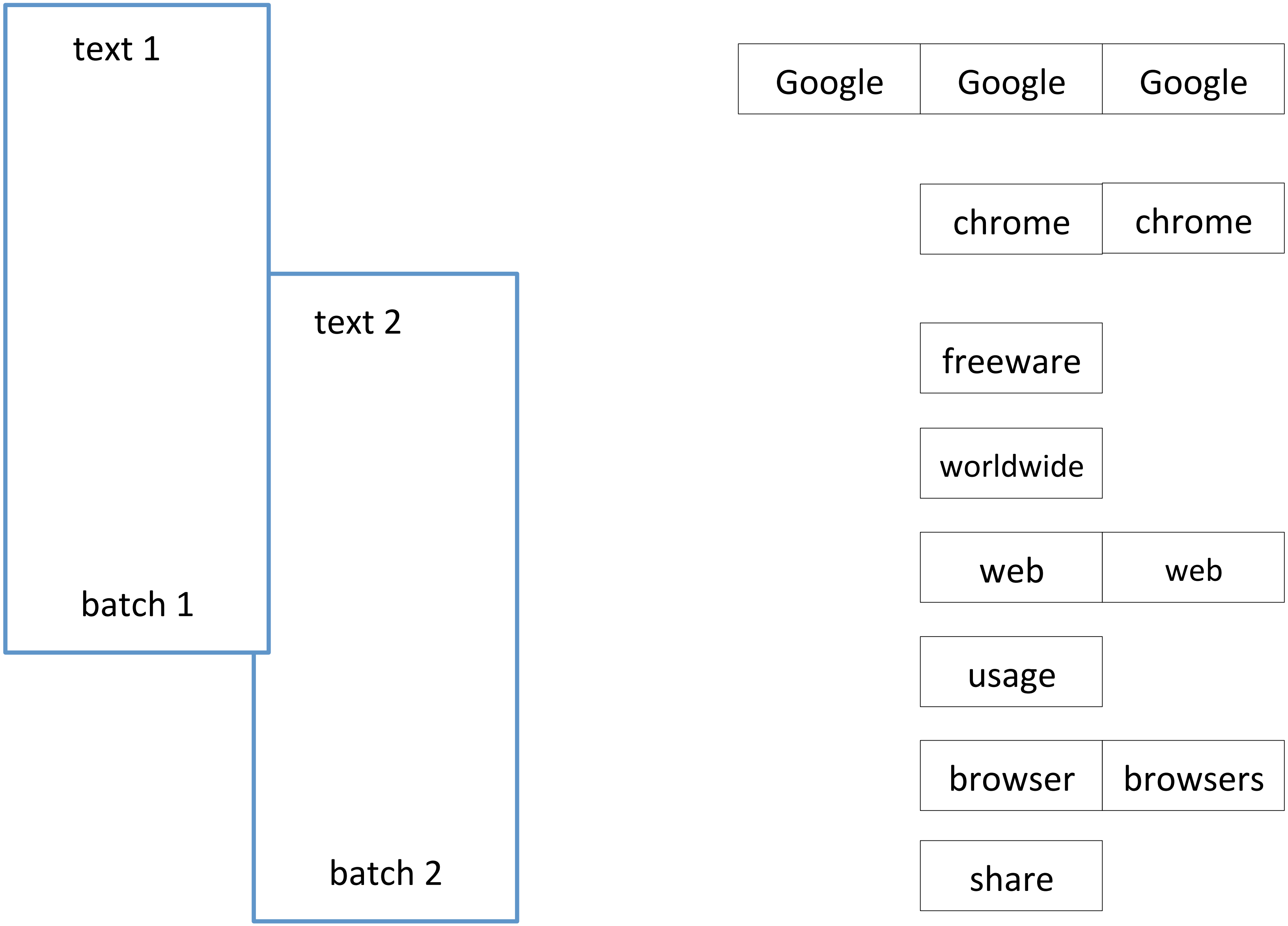
M/R Computing Model



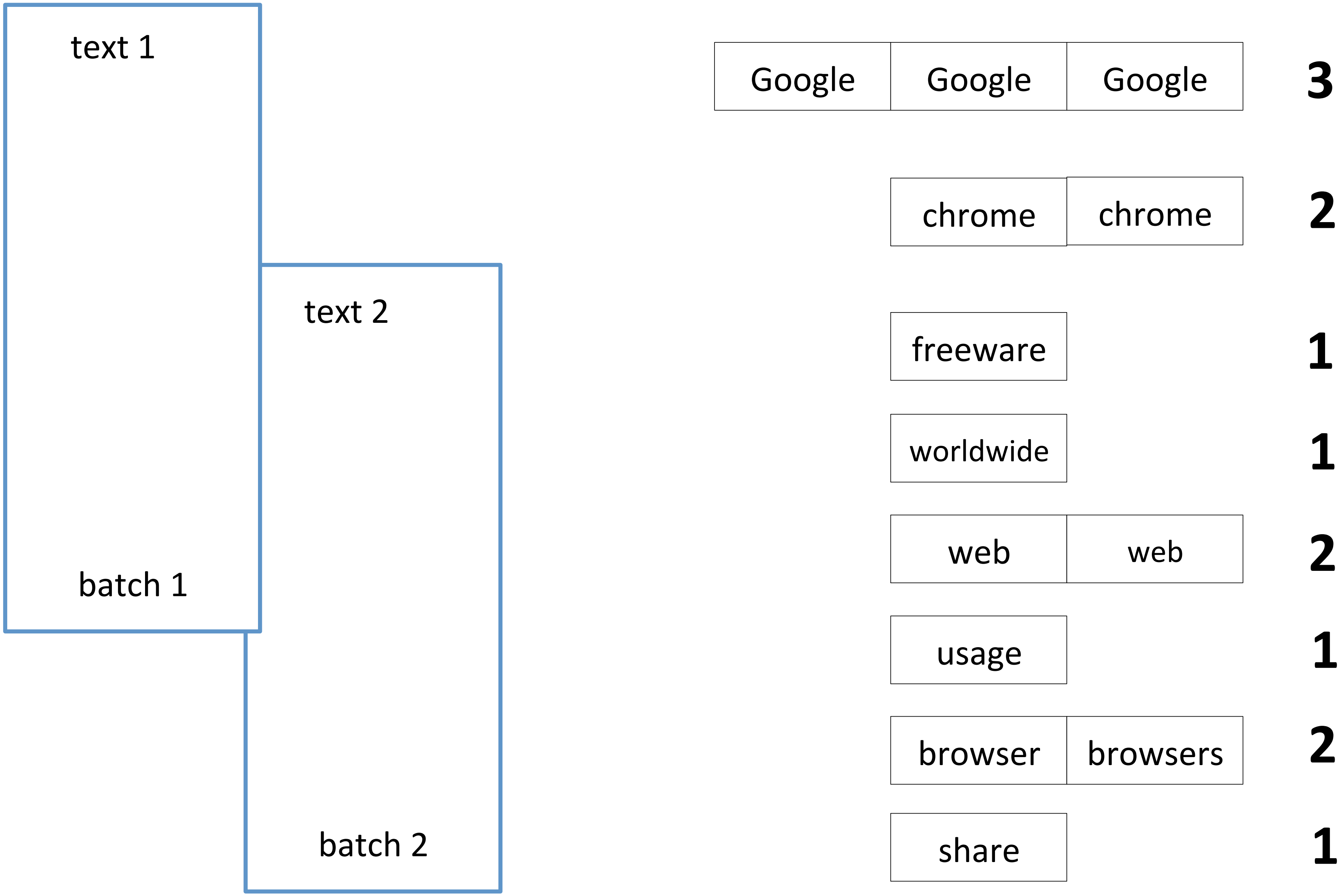
M/R Computing Model



M/R Computing Model



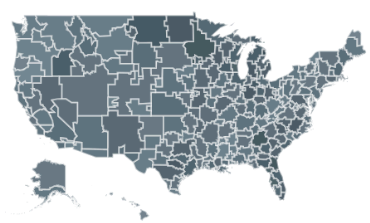
M/R Computing Model



Analyze search logs to find popular trends

Midterm Elections 2018

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

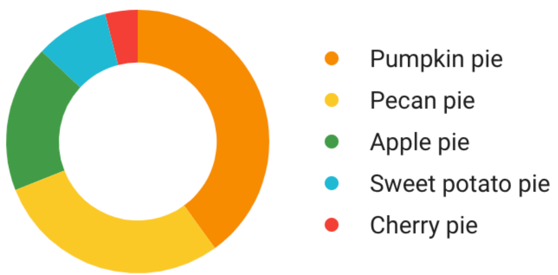


Search interest in voting, 10/30 to 11/06

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 ?

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 ?
```

Thanksgiving	**5**
Trump	**8**

Query Processing

- Hadoop-M/R is **not** a data management system, it is a general framework.
- It can therefore implement queries :
  - ***It scales easily, but does not <u>always</u> 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

store_id	sale_amount
	10
	30
	20

	40
	50
	10

	80
	60

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**

store_id **sale_amount**

SUM(sale_amount)
= 110

	30
	20
	10

	80
	50

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

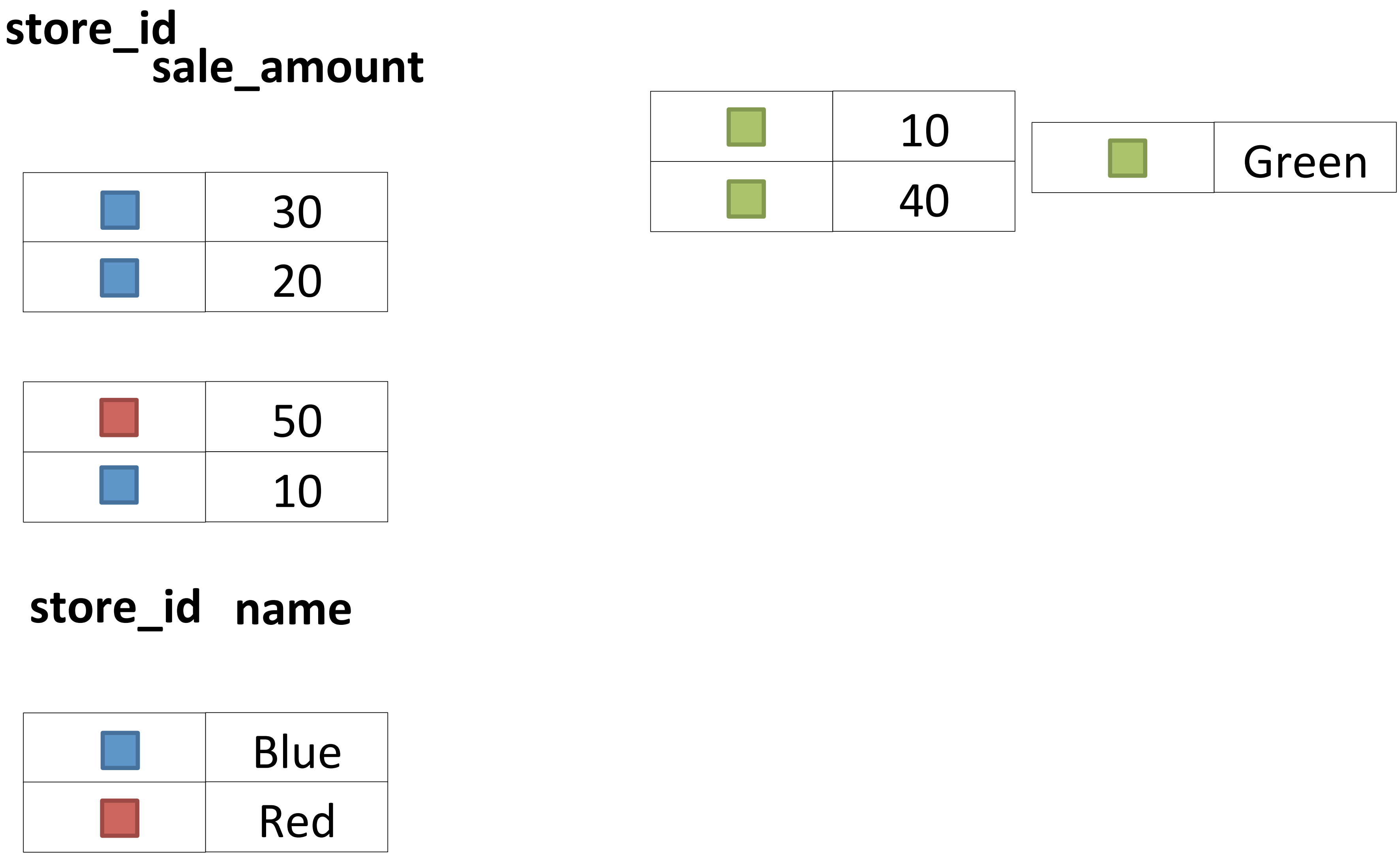
```
SELECT store_name, sale_amount
FROM   sales, store
WHERE
sales.store_id = store.store_id
```

Join in M/R

store_id	sale_amount
	10
	30
	20
	40
	50
	10

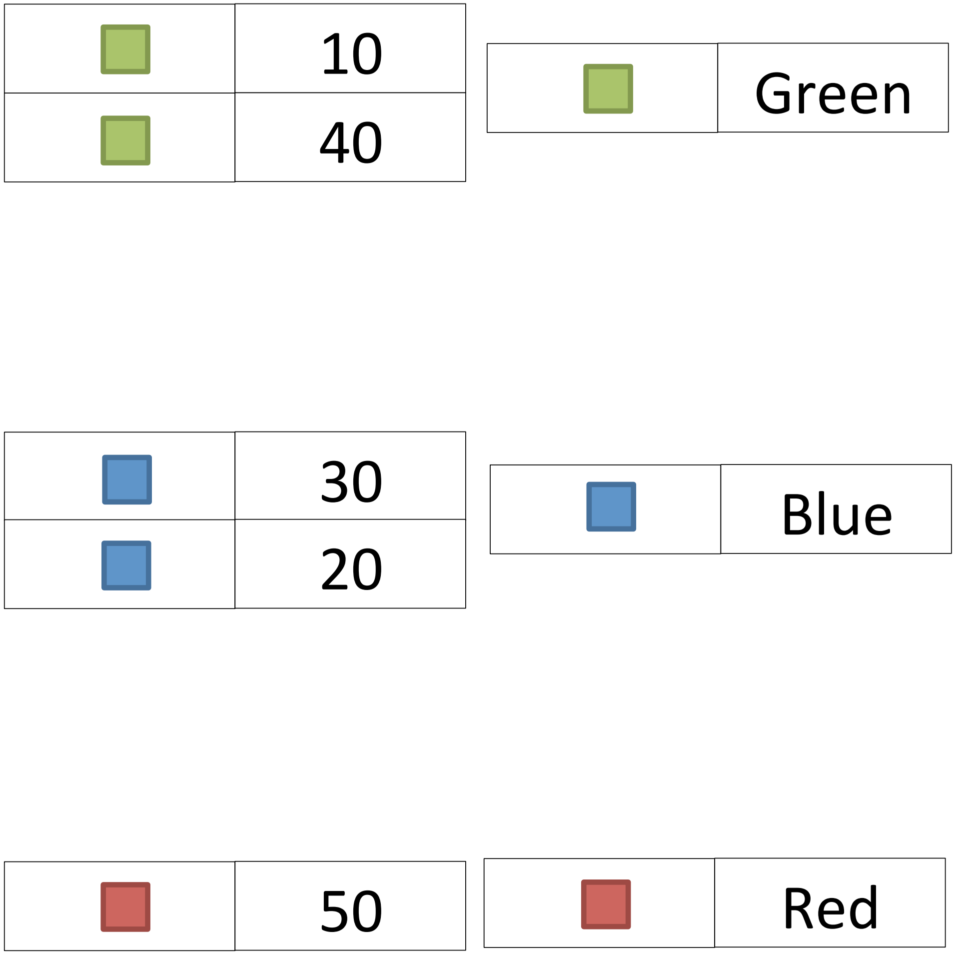
store_id	name
	Green
	Blue
	Red

Join in M/R

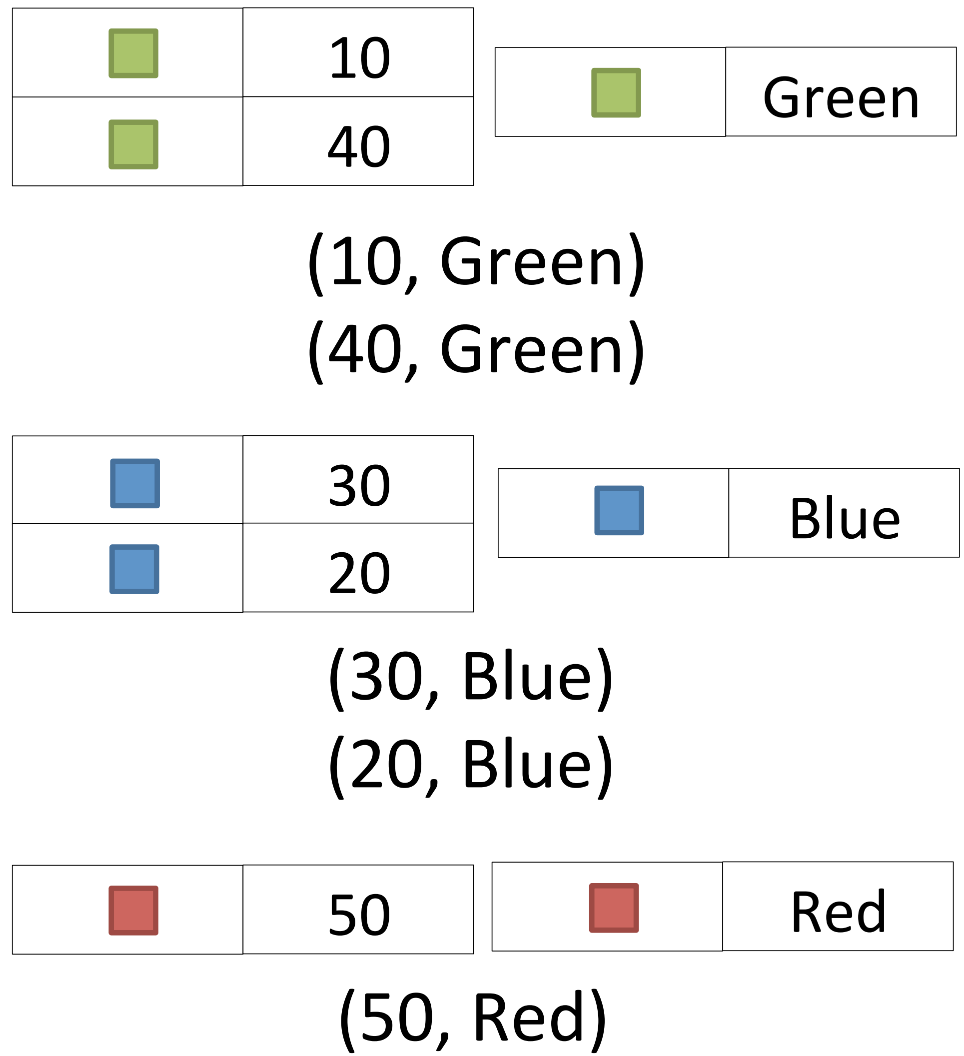


store_id
sale_amount
30
20
50
10
store_id
name
Blue
Red
10
40
Green

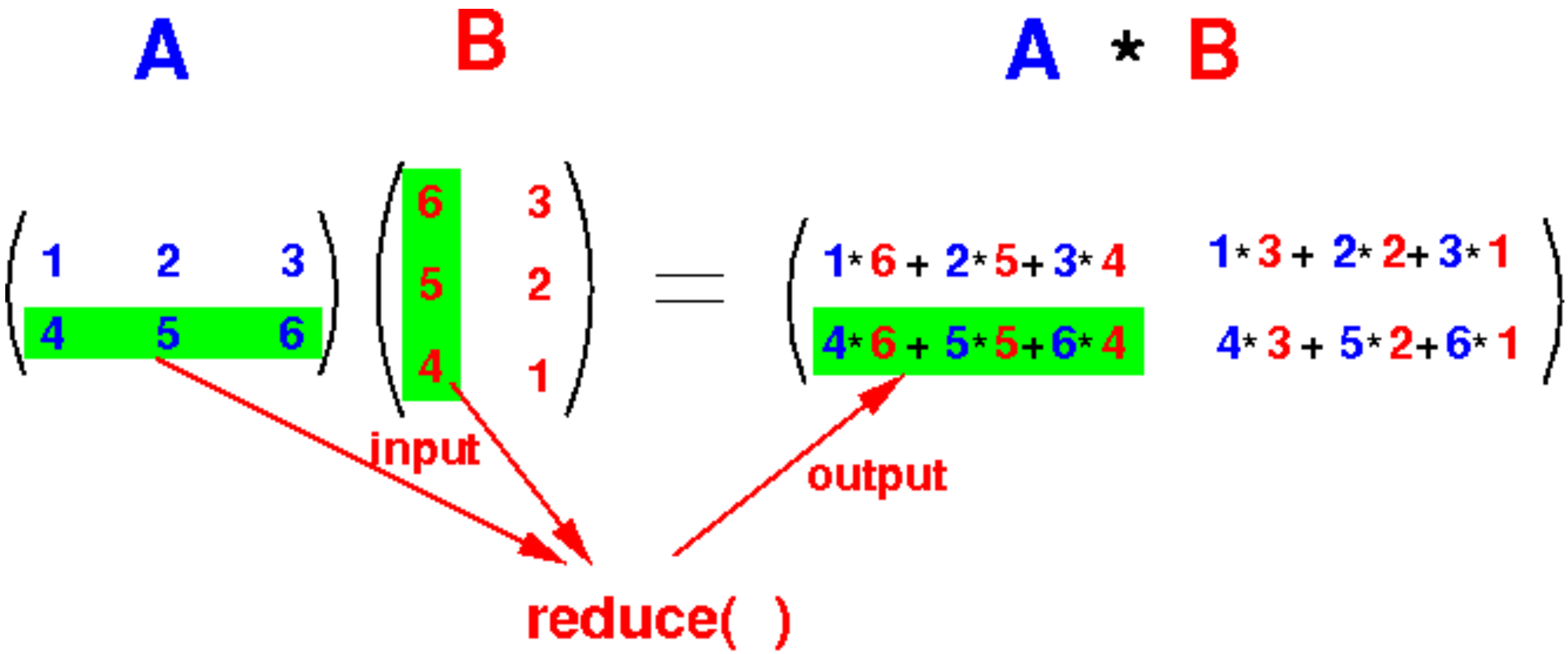
Join in M/R



Join in M/R


10
40
Green
(10, Green)
(40, Green)
30
20
Blue
(30, Blue)
(20, Blue)
50
Red
(50, Red)

Matrix Multiplication



Matrix Multiplication

Matrix A (2*3)

L1	1	1	1
L2	2	2	2

Matrix B (3*2)

1	2
1	2
1	2
C1	C2

Matrix Multiplication

Matrix A (2*3)

1 1 1

L1

L2 2 2 2

1

1

1

Matrix B (3*2)

2

2

2

C1 C2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1	1	1

1
1
1

Matrix B (3*2)

C1 C2

2	2	2

2
2
2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1	1	1

1
1
1

Matrix B (3*2)

C1 C2

2	2	2

2
2
2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1+

1	1

1
1

Matrix B (3*2)

2	2	2

2
2
2

C1 C2

Matrix Multiplication

Matrix A (2*3)

1	1

L1

L2

1+

1
1

Matrix B (3*2)

2	2	2

2
2
2

C1 C2

Matrix Multiplication

Matrix A (2*3)

1

L1

L2

1+1

1

Matrix B (3*2)

2	2	2

C1 C2

2
2
2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1+1

1

1

Matrix B (3*2)

2	2	2

2
2
2

C1 C2

Matrix Multiplication

Matrix A (2*3)

L1
L2

1+1+1=3

Matrix B (3*2)

2	2	2

C1 C2

4+

2
2
2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1+1+1=3

Matrix B (3*2)

2	2

C1 C2

4+

2
2

Matrix Multiplication

Matrix A (2*3)

L1

L2

1+1+1=3

Matrix B (3*2)

$\boxed{2}$

C1 C2

4+4+

$\boxed{2}$

Matrix Multiplication

Matrix A (2*3)

L1

L2

1+1+1=3

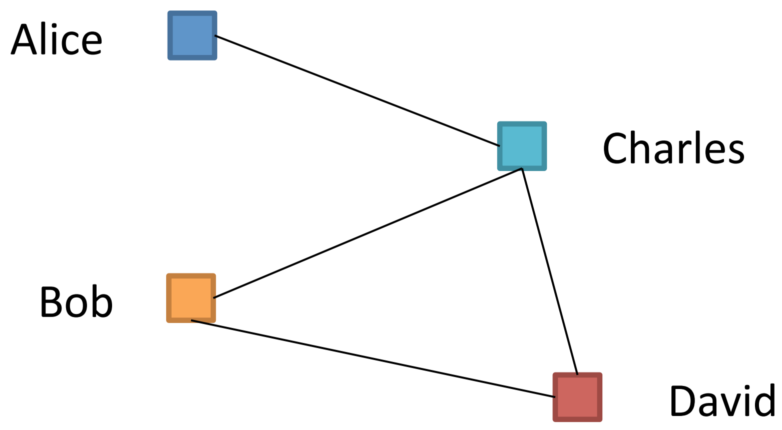
Matrix B (3*2)

C1 C2

4+4+4=12

Social Network Analysis

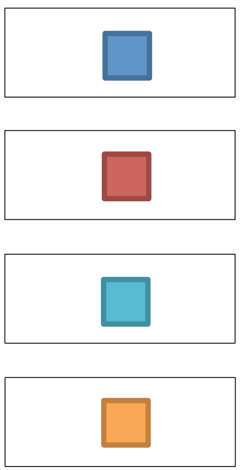
- Find all pairs of "similar" users
  - in terms of interests, age, country, behavior ...



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

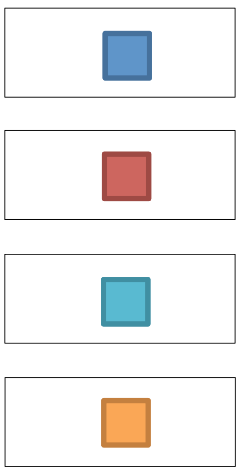
User-Similarity in M/R

user

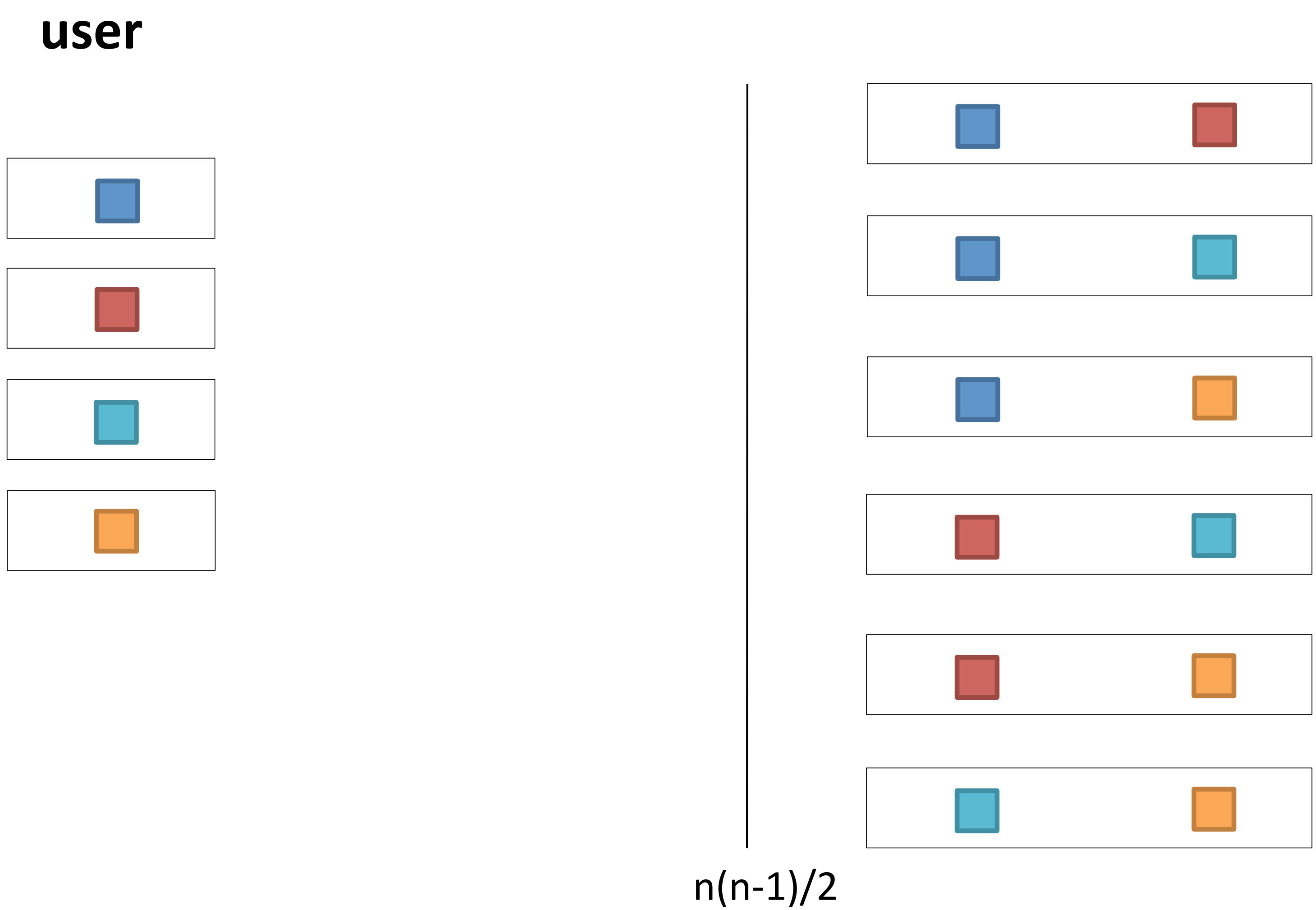


User-Similarity in M/R

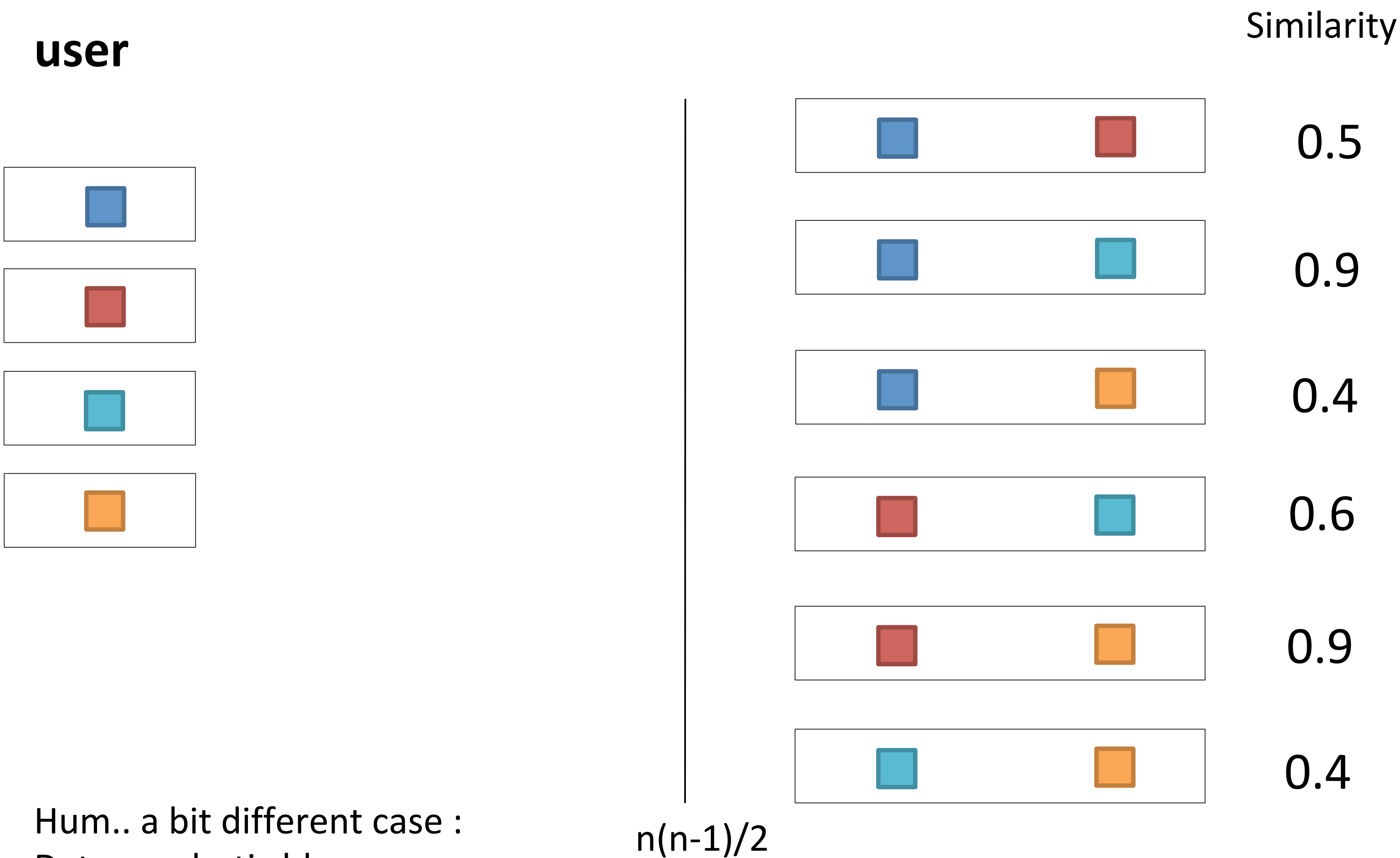
user



User-Similarity in M/R



User-Similarity in M/R



Hum.. a bit different case :
<u>Data quadratic blowup.</u>

It is called M/R but…

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

- why is that ?
  - because we just have to define 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

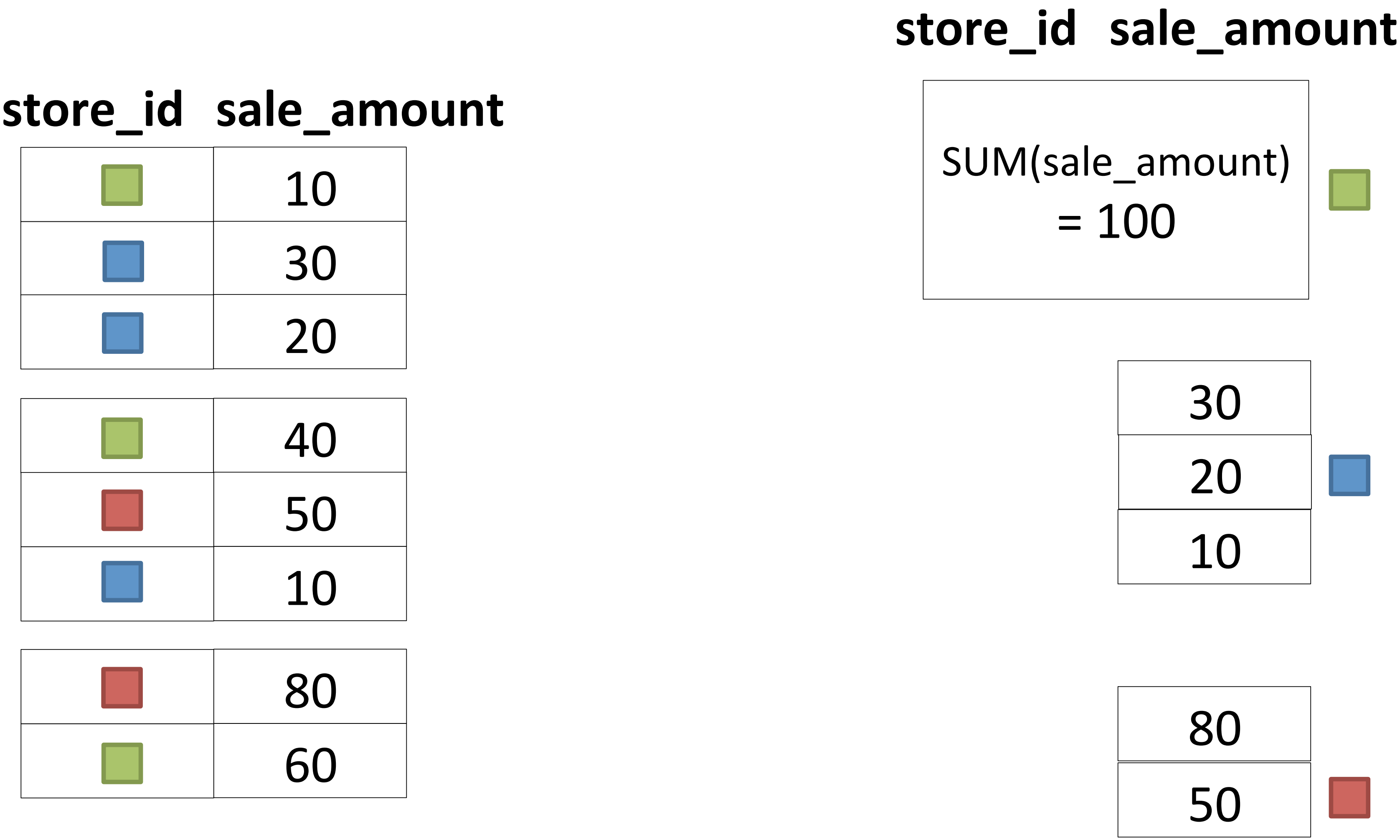
Query Trends Using MapReduce

```
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 tuple):

  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 tuple):

  if tuple belongs to SALES
    emit( t.store_id , <"profit",  t.sale_amount> )

  if tuple 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 )
```

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}
            emit( k_{i,j} , user_i_profile)
```

Example: Three Users

Mapper for user **1**

Reducer for k_{1,2}

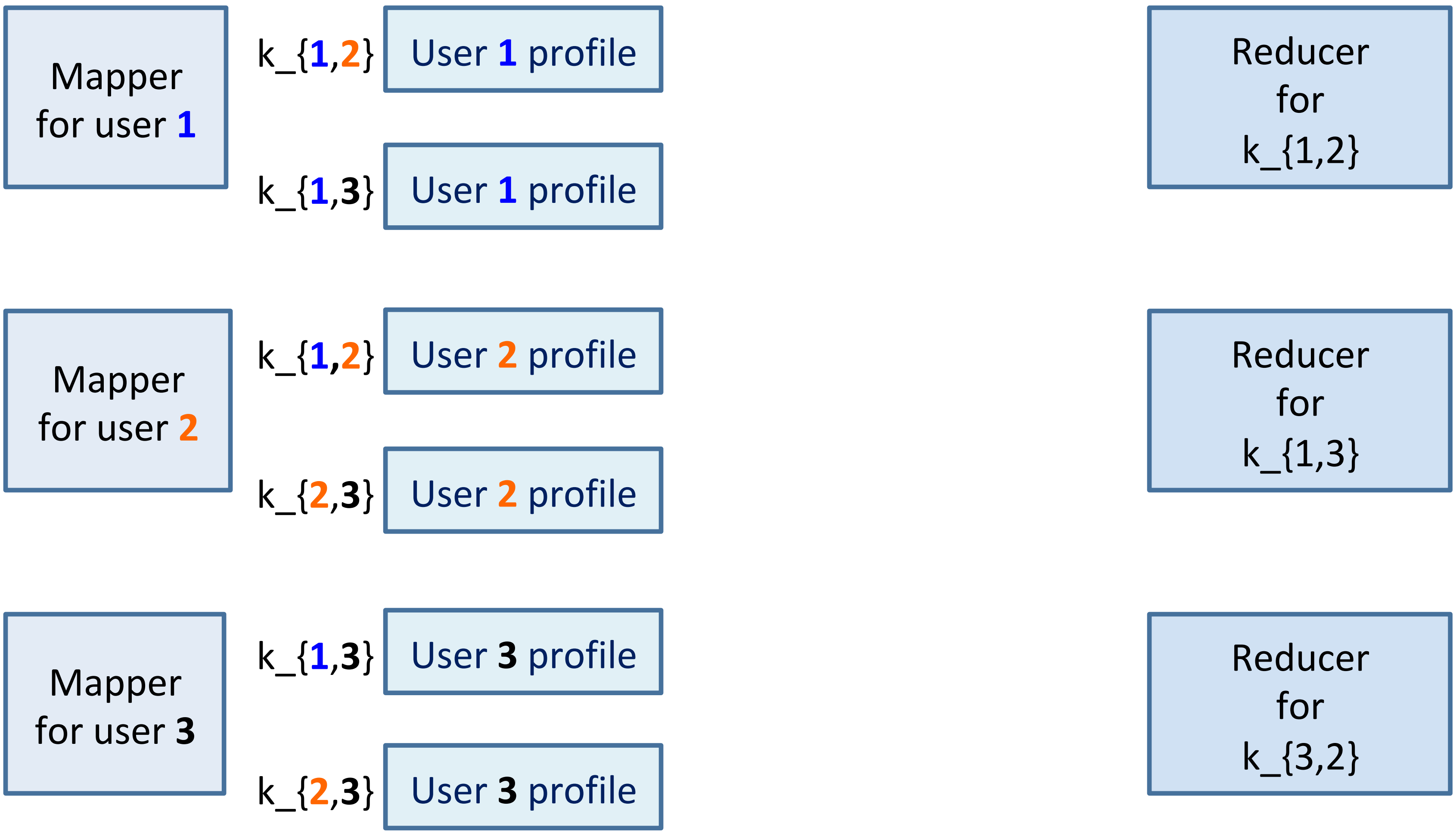
Mapper for user **2**

Reducer for k_{1,3}

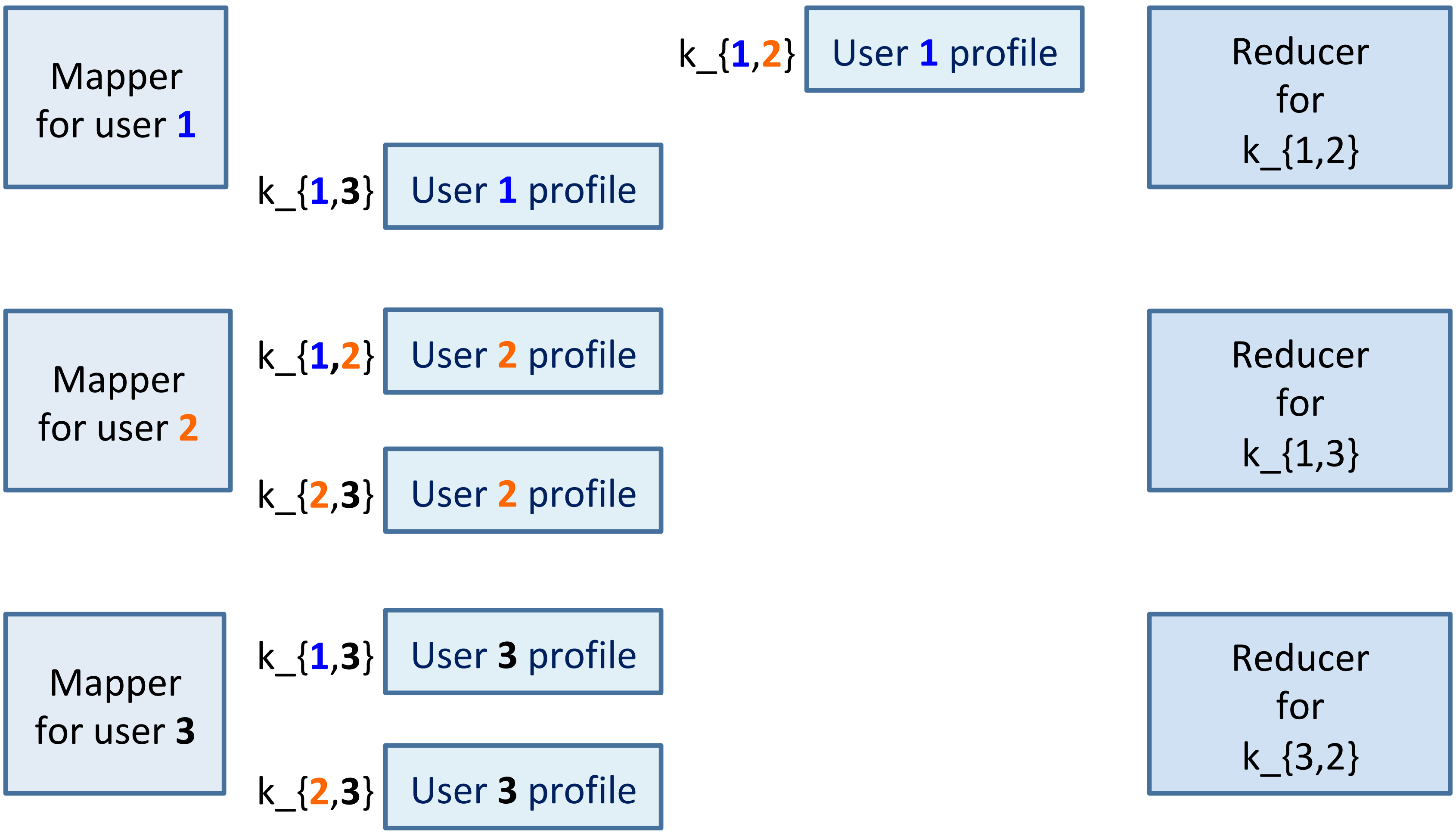
Mapper for user **3**

Reducer for k_{3,2}

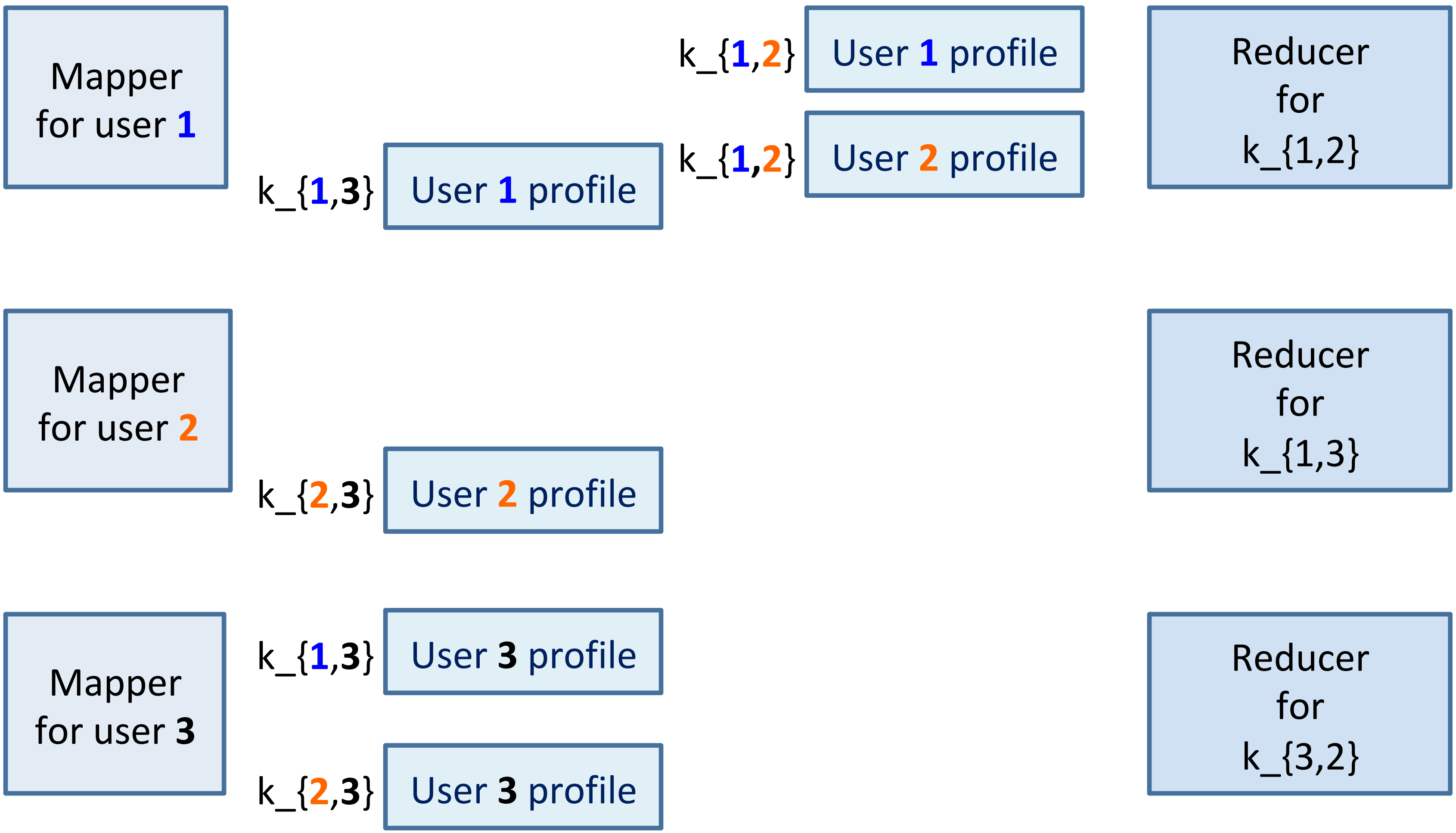
Example: Three Users



Example: Three Users



Example: Three Users



Example: Three Users

Mapper for user **1**

Mapper for user **2**

Mapper for user **3**

k_{**1**,**2**} User **1** profile

k_{**1**,**2**} User **2** profile

k_{**1**,**3**} User **1** profile

k_{**1**,**3**} User **3** profile

k_{**2**,**3**} User **2** profile

k_{**2**,**3**} User **3** profile

Reducer for k_{1,2}

Reducer for k_{1,3}

Reducer for k_{3,2}

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}
         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

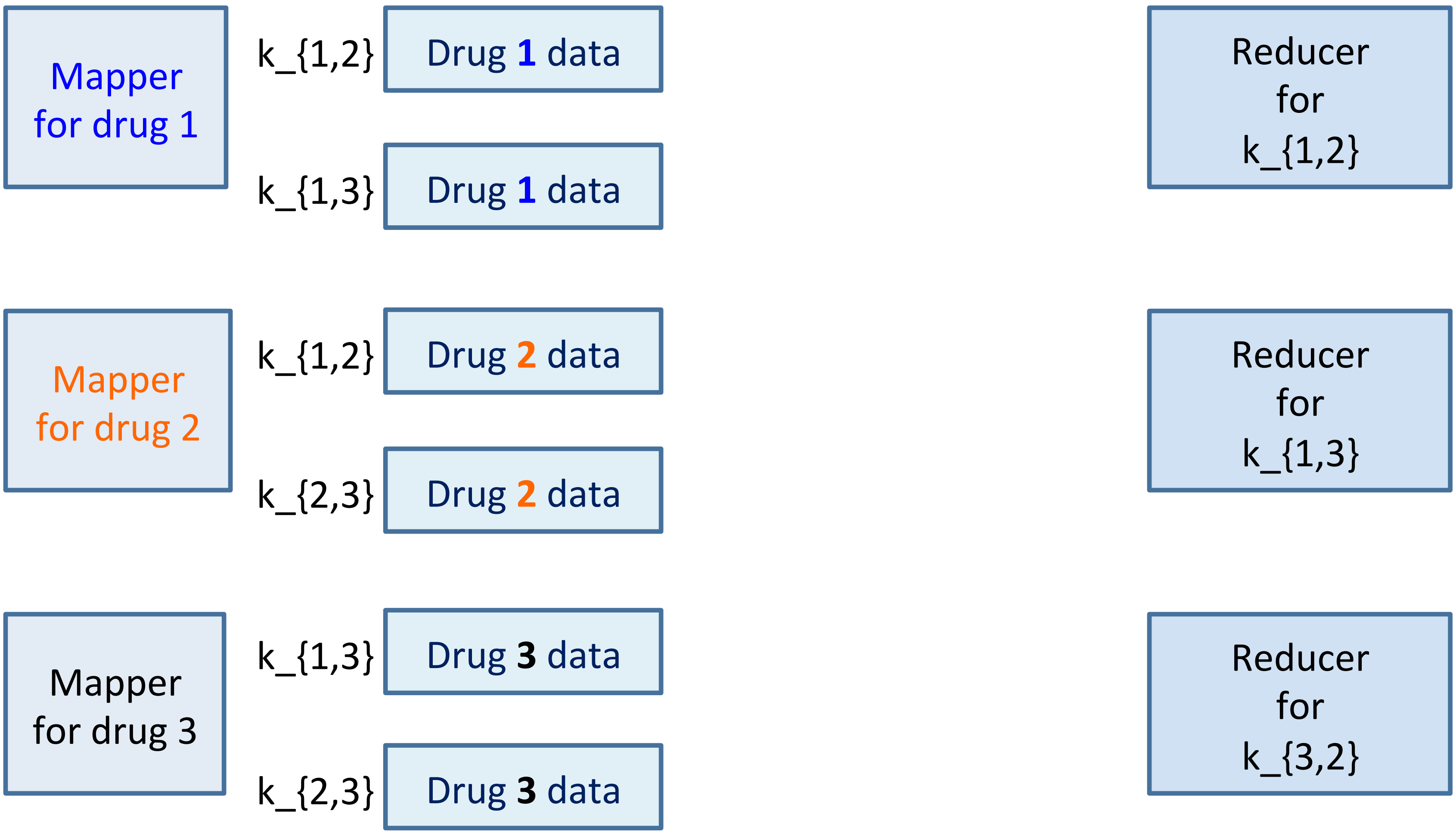
The “Drug Interaction” Problem

- 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.

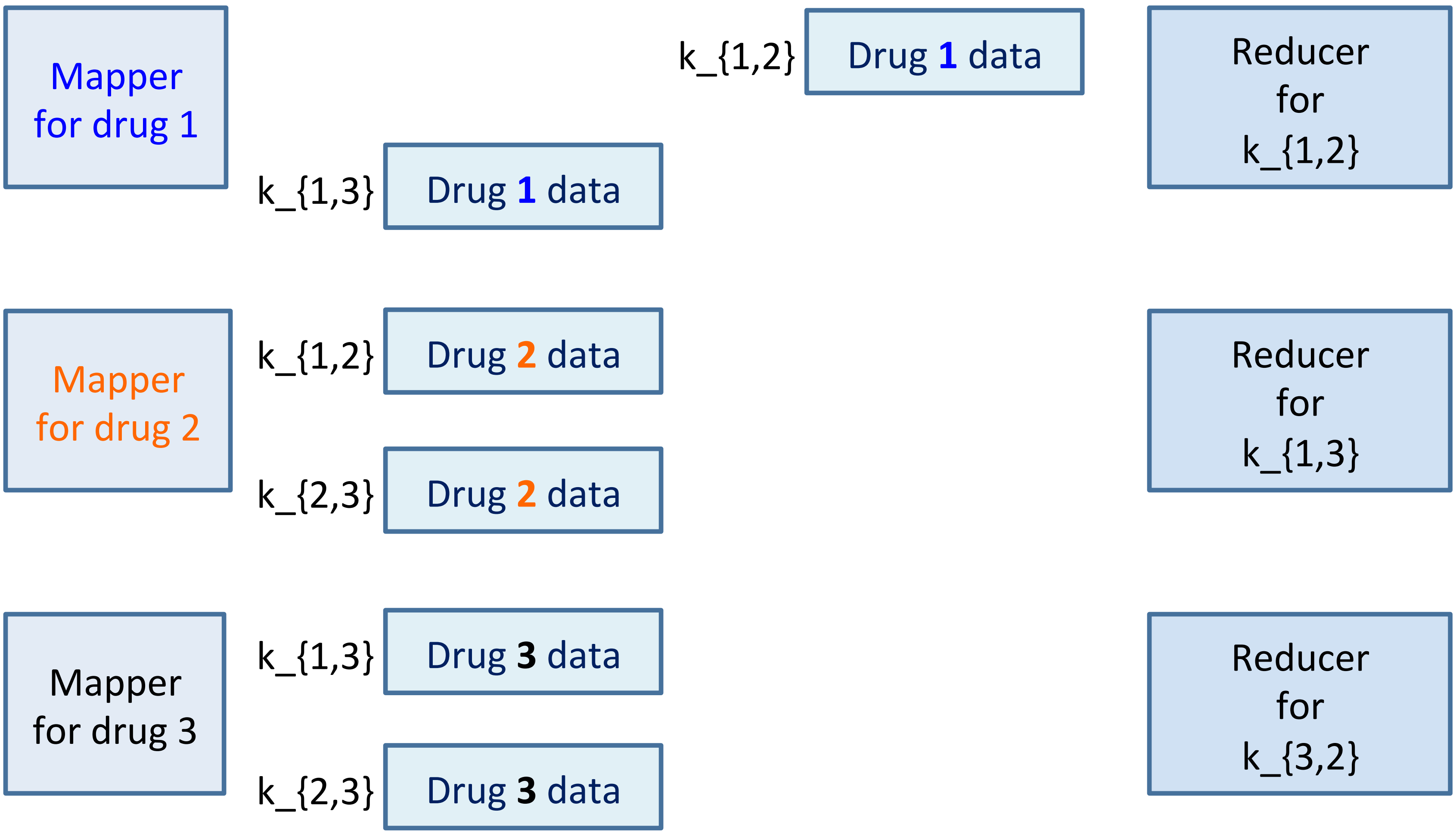
“Drug Interaction” Using MapReduce

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

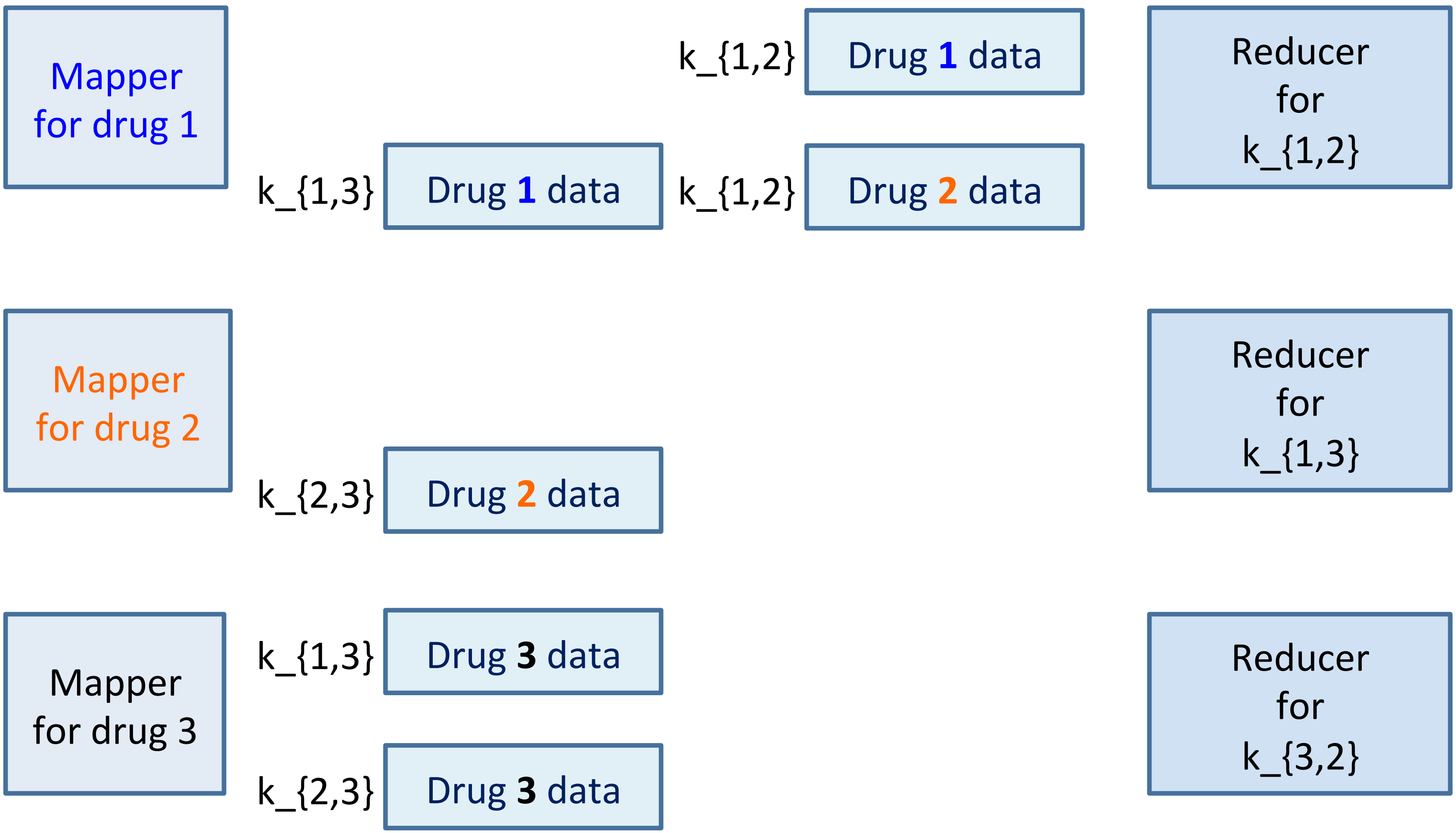
Example: Three Drugs



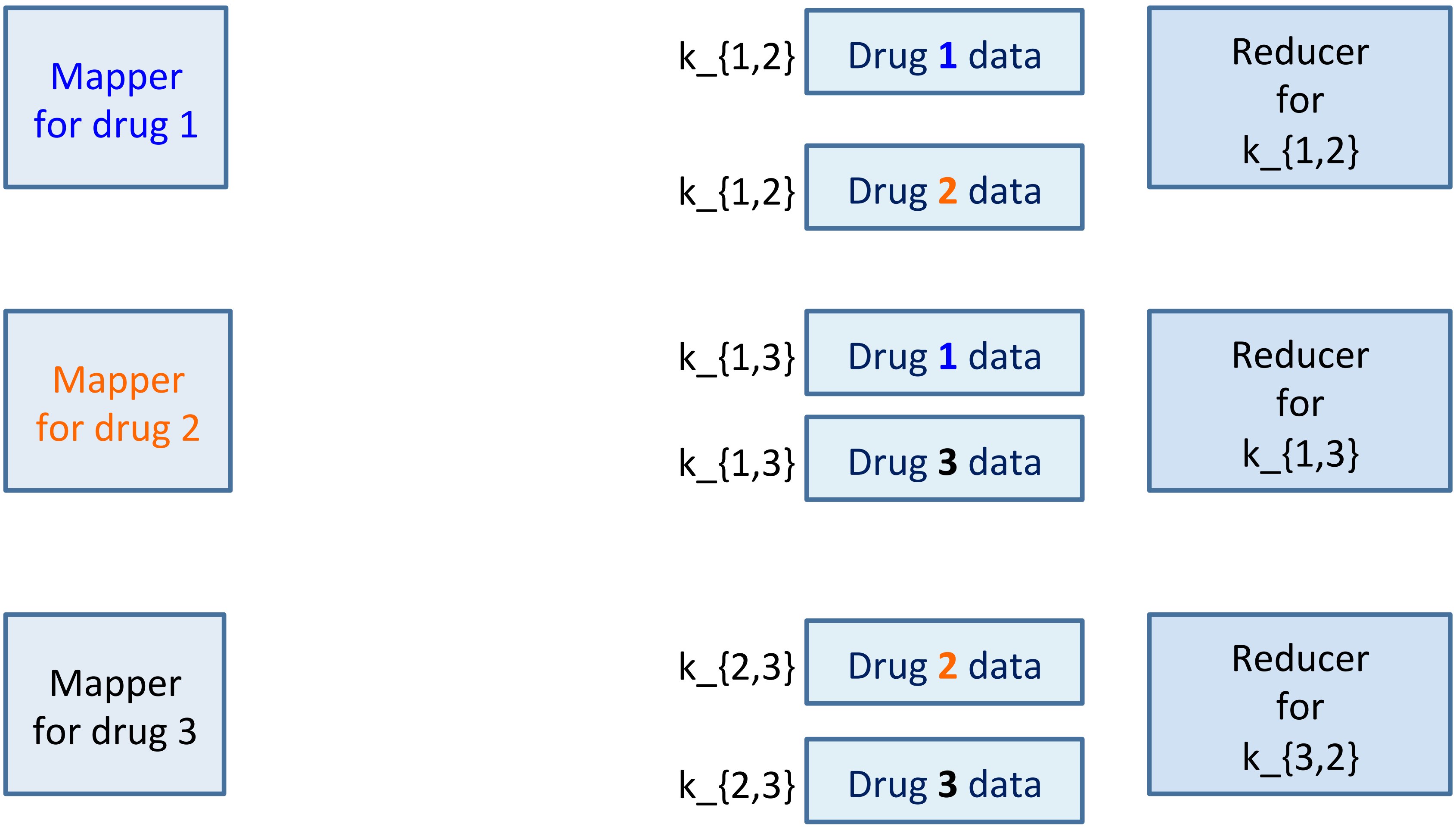
Example: Three Drugs



Example: Three Drugs



Example: Three Drugs



What Went Wrong?

- After several hours computation still did not end..

- 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

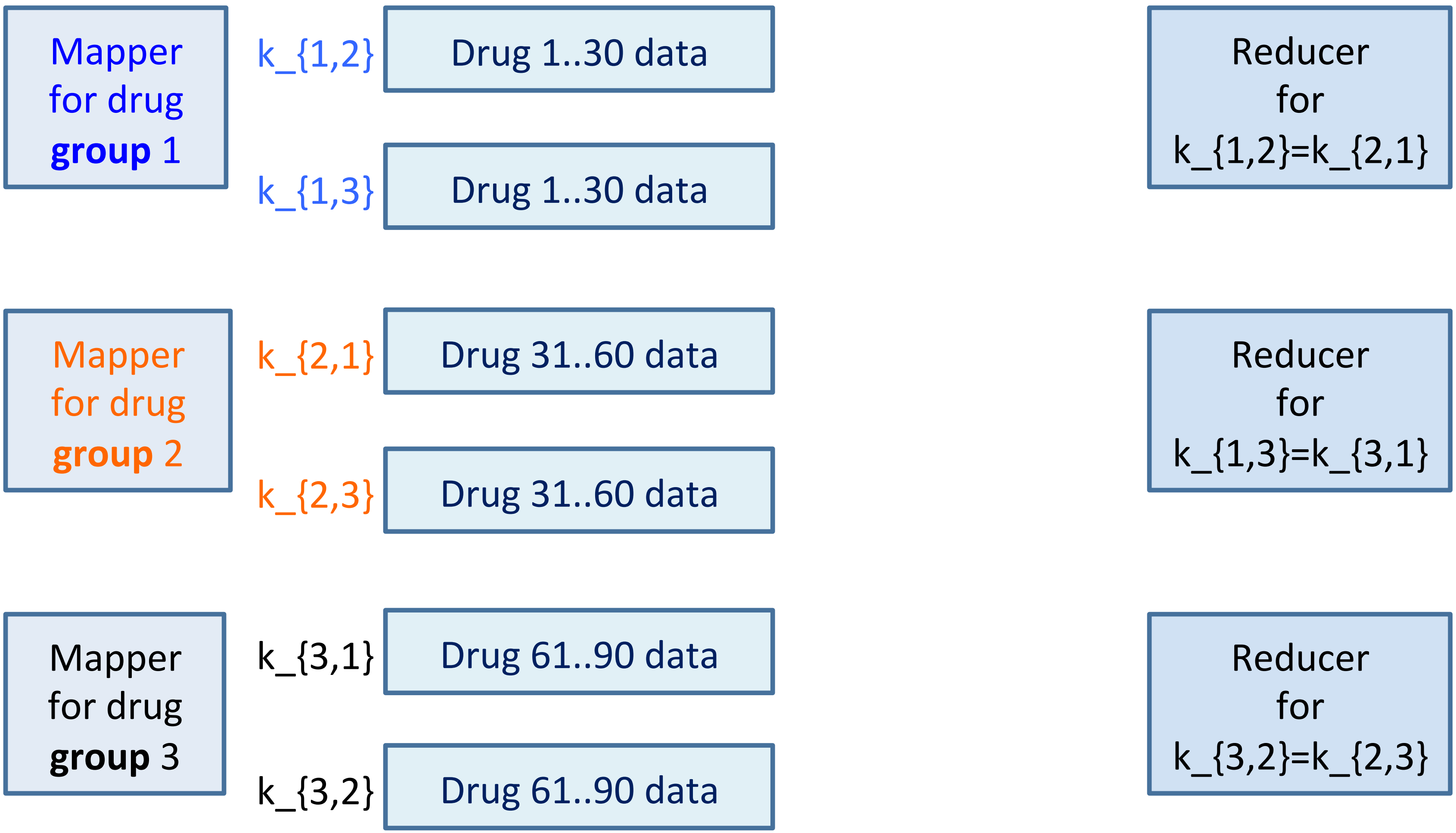
A Better Approach

- 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}
         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

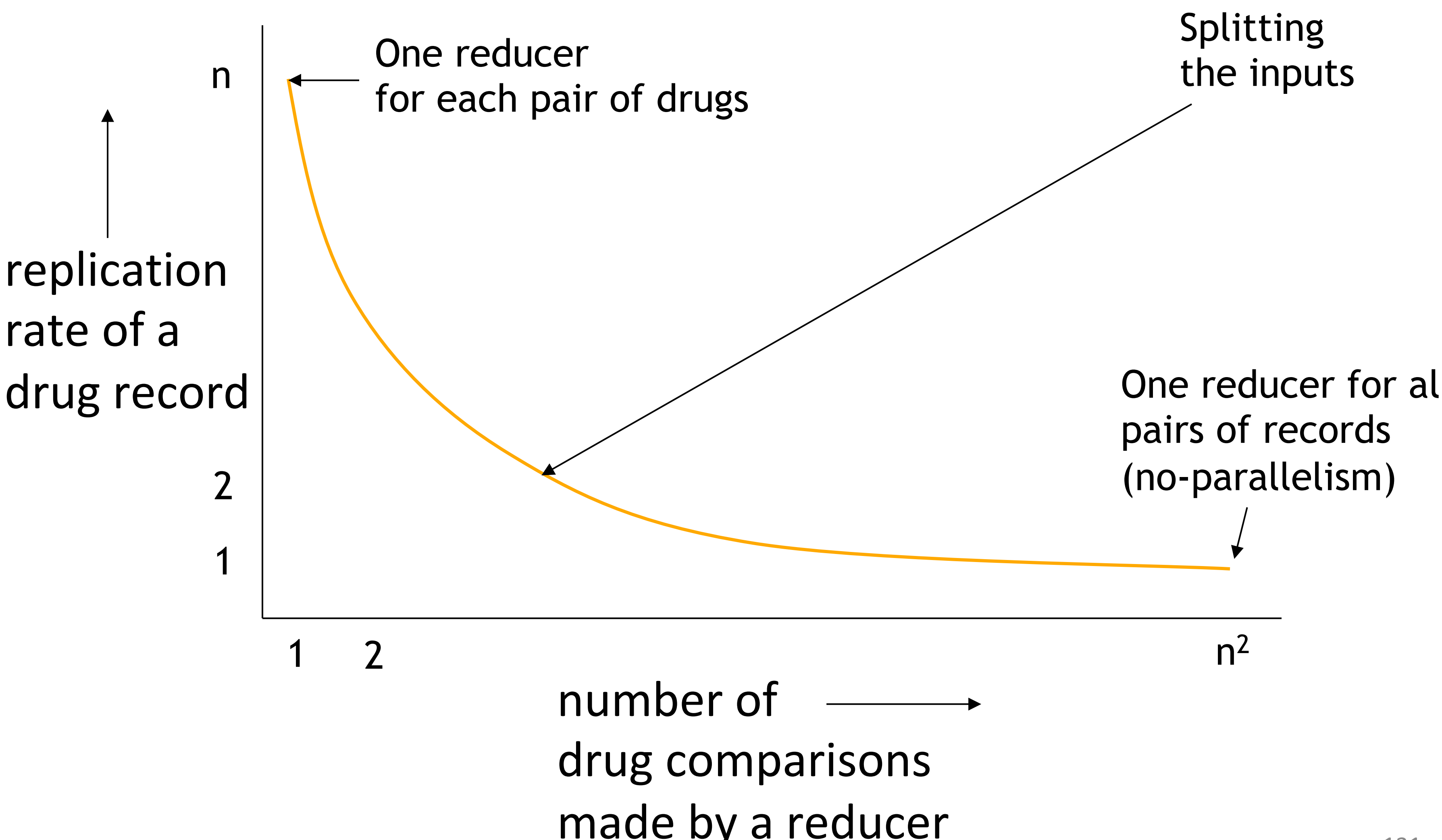


Why It Works

- 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
n2
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.

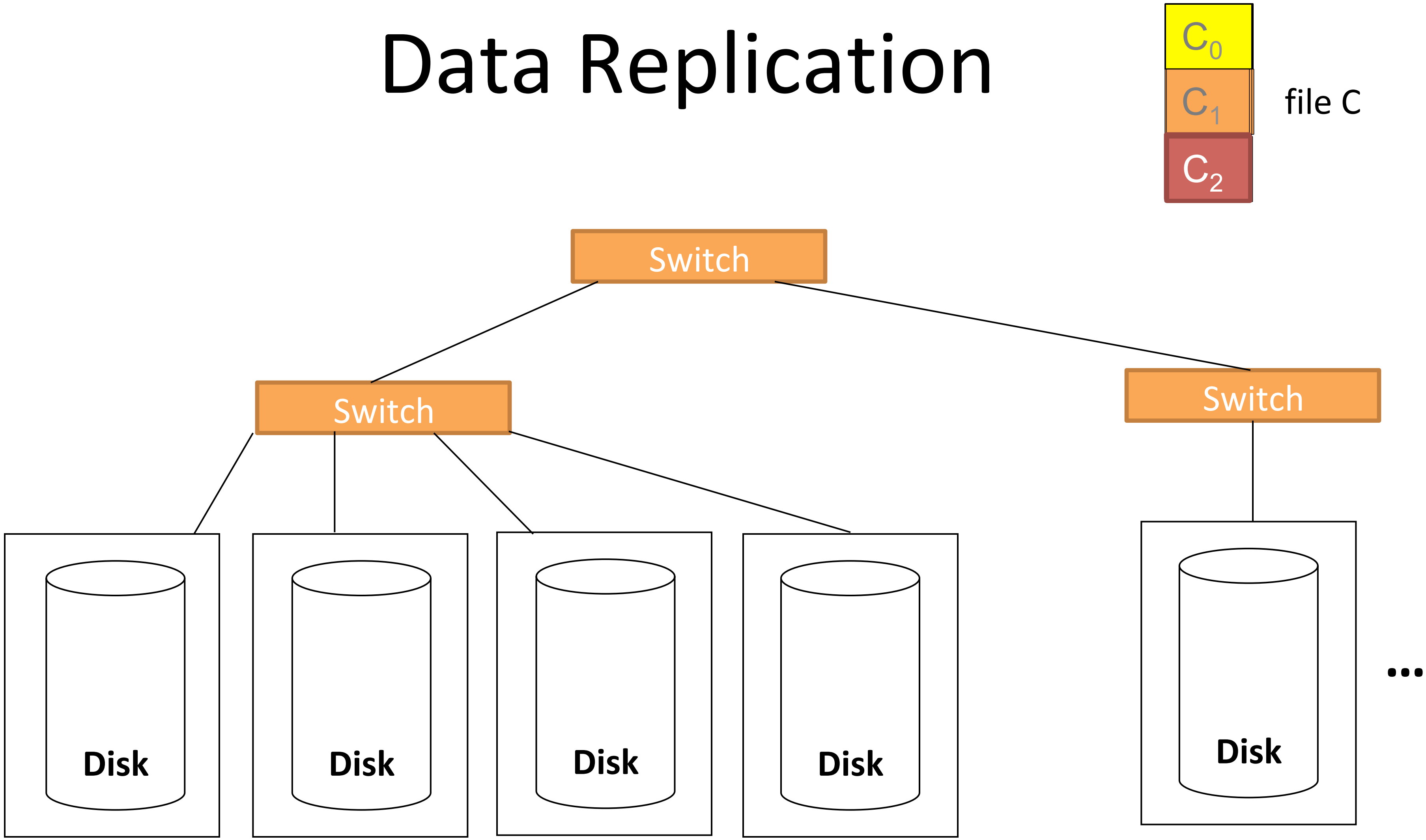
- The “Drug-Interaction Problem” is a good model for how one can trade off communication against parallelism.

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.

STORING DISTRIBUTED DATA

Data Replication



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.

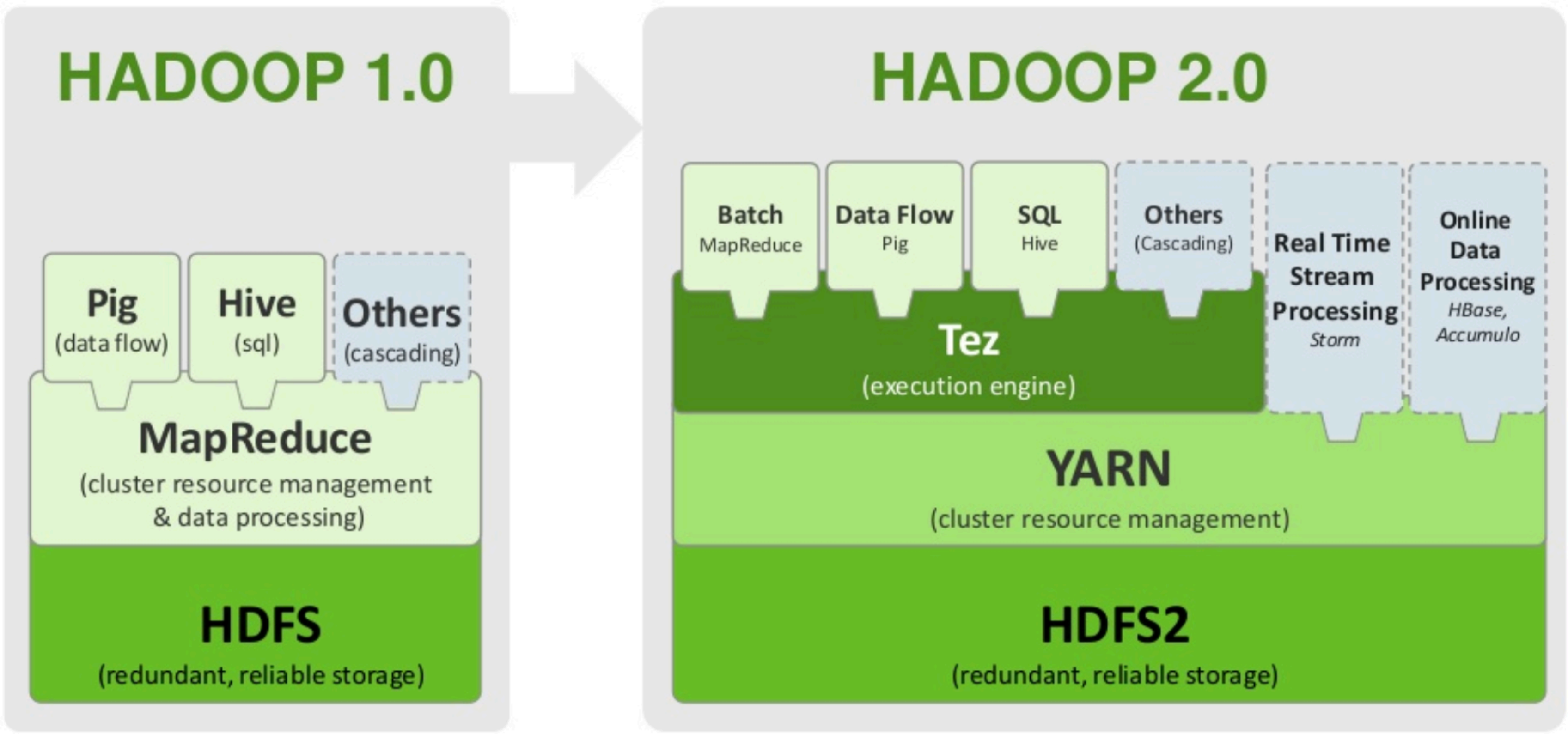
Distributed File System

Client library for file access

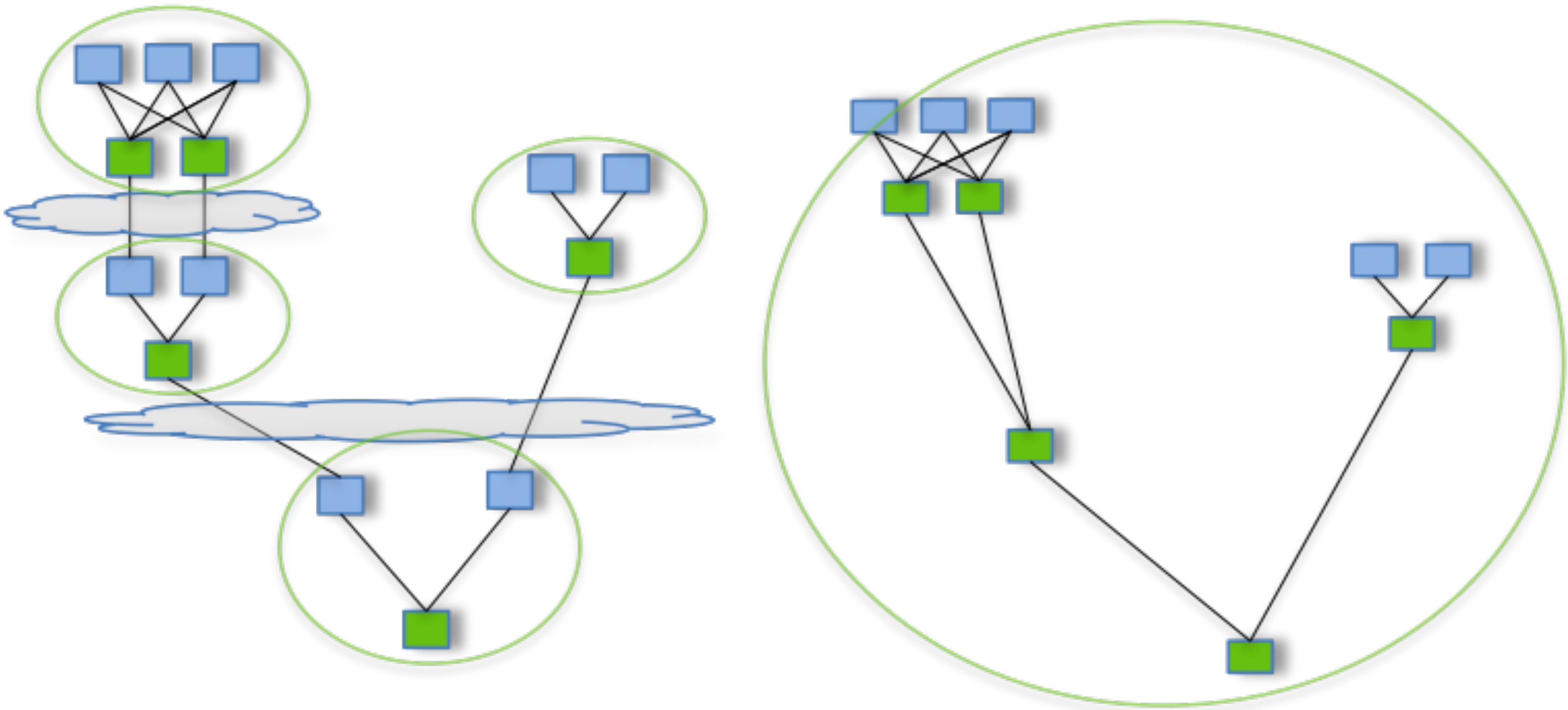
- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

- **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

Some Evolutions in Hadoop



Tez Allows for DAG-Dataflows



Pig/Hive - MR

Pig/Hive - Tez