Big data management and platforms

Distributed data analysis systems management for
Processing large amounts of data

Toni Espinosa *Dep. Arquitectura Computadors i Sistemes Operatius UAB* 12/12/17

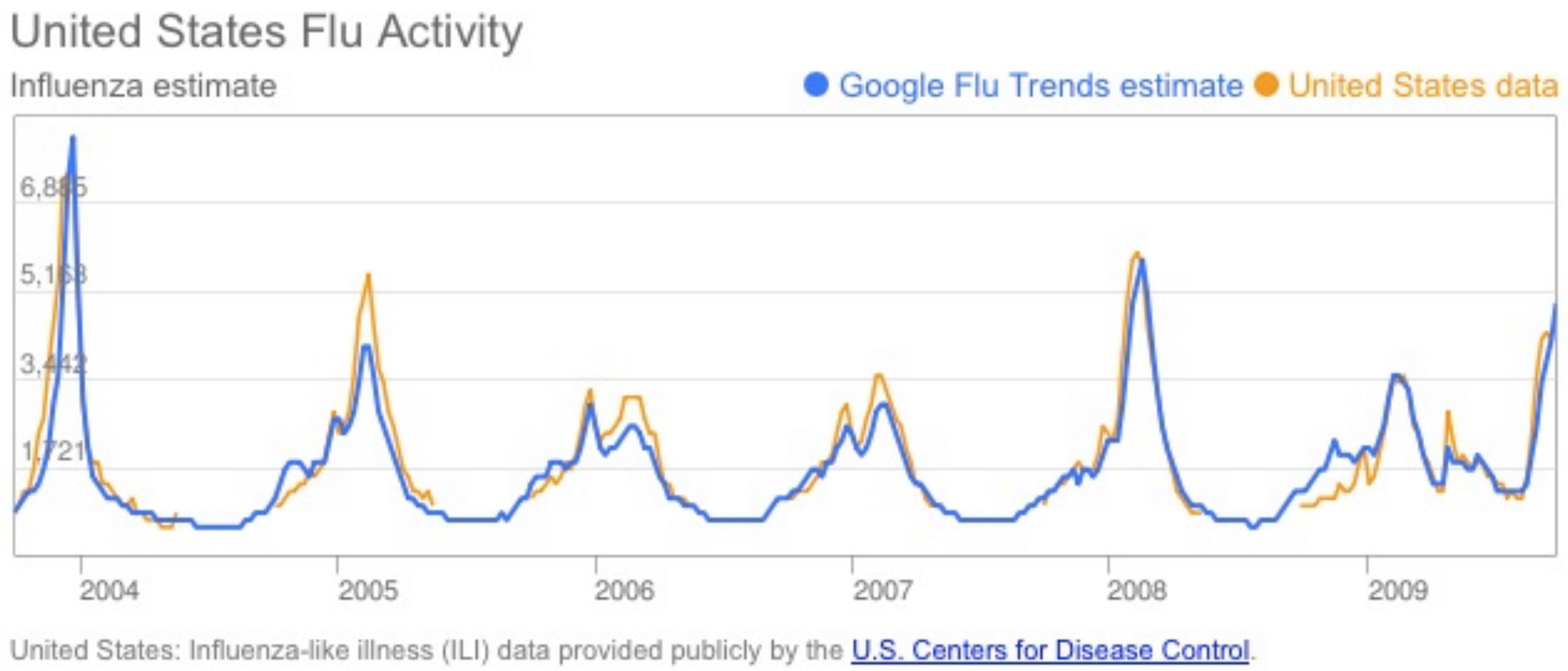
antoniomiguel.espinosa@uab.cat

Topics covered

- Big data definition
- Tools for Big Data
- Map-Reduce programming model and applications
- Hadoop
- Hbase
- Spark
- Computational infrastructure context

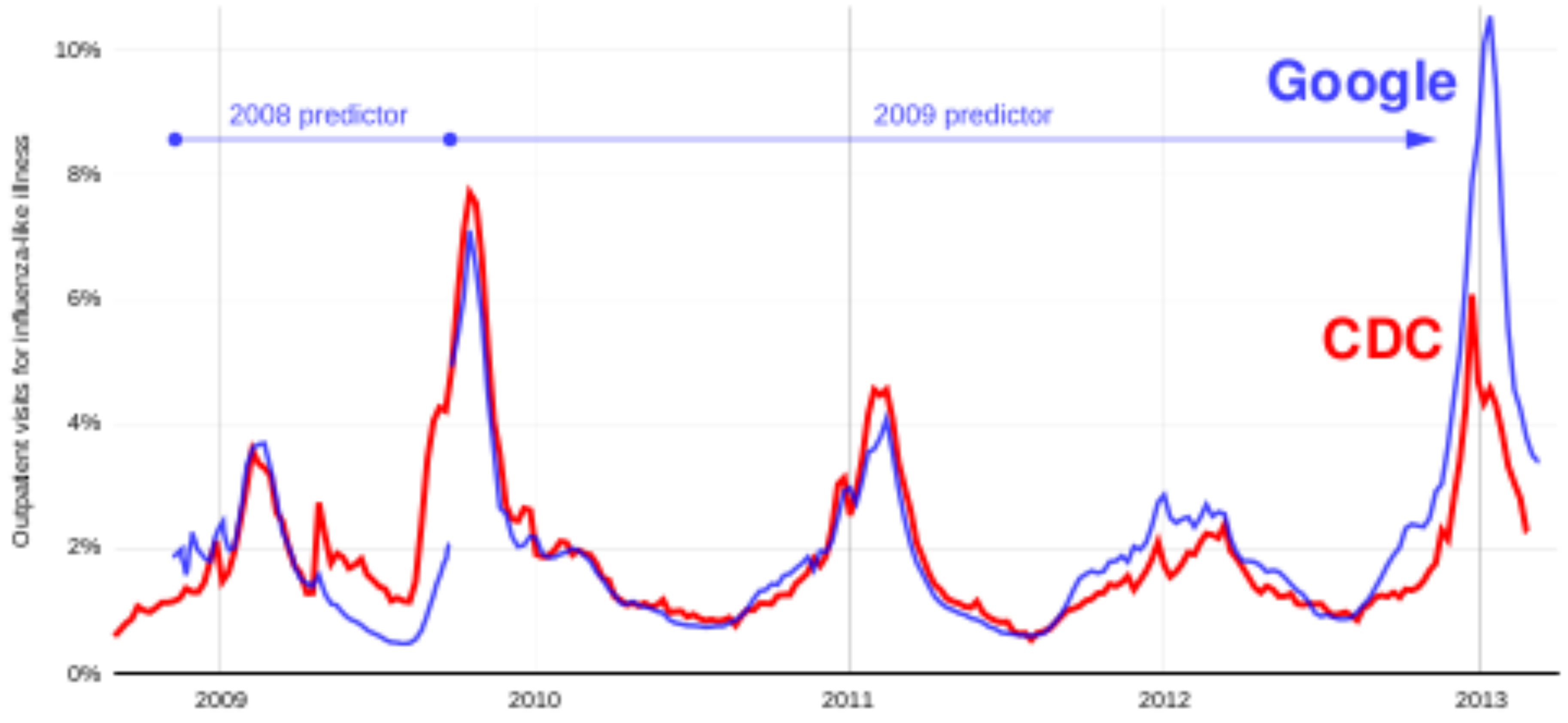
The origin: Google Flu prediction 2003-2008

Estimation of flu activity vs Centre for disease control real data



But, model was overfitted for 2013 predictor

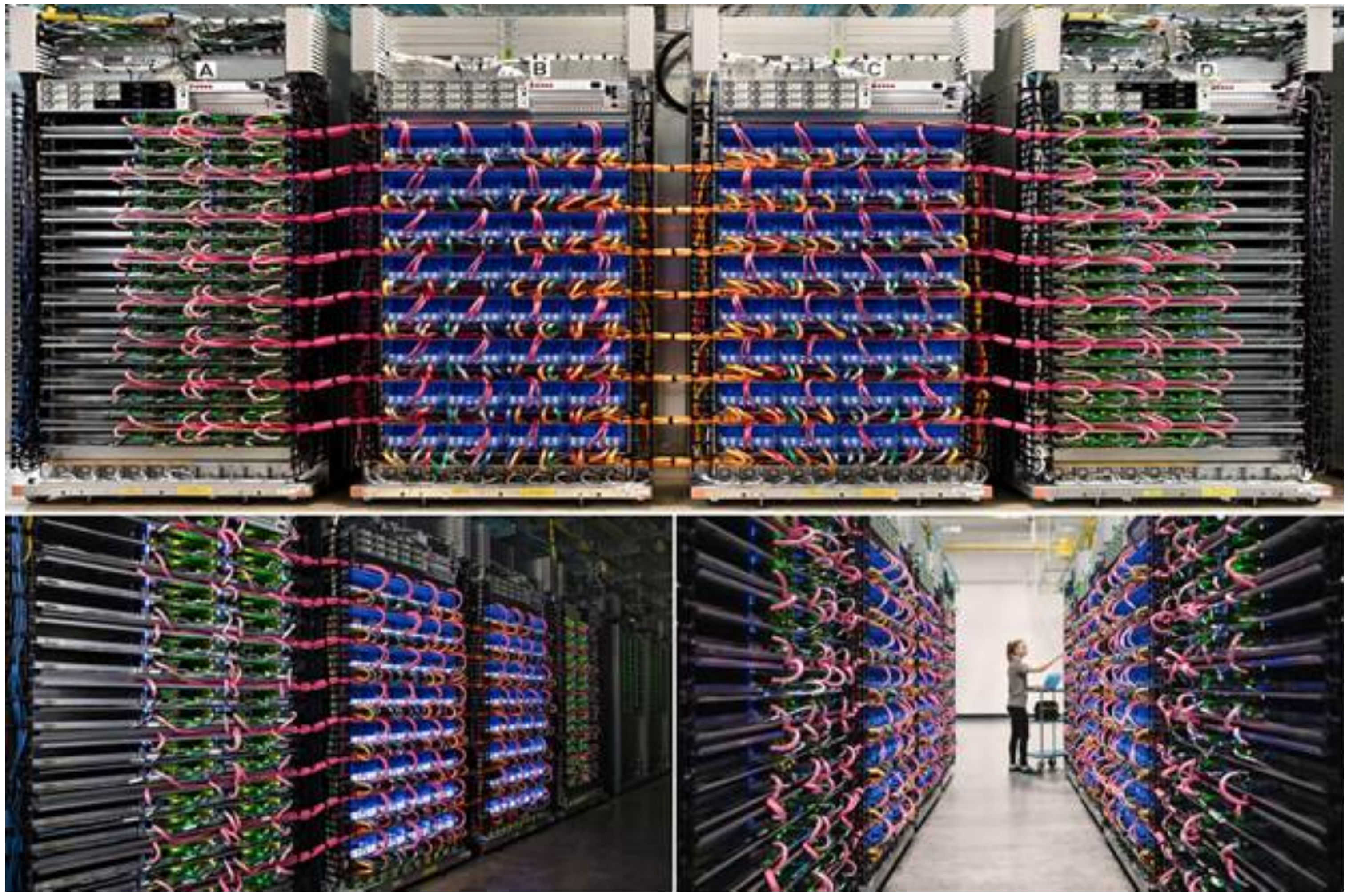
Second divergence in 2012–2013 for U.S.



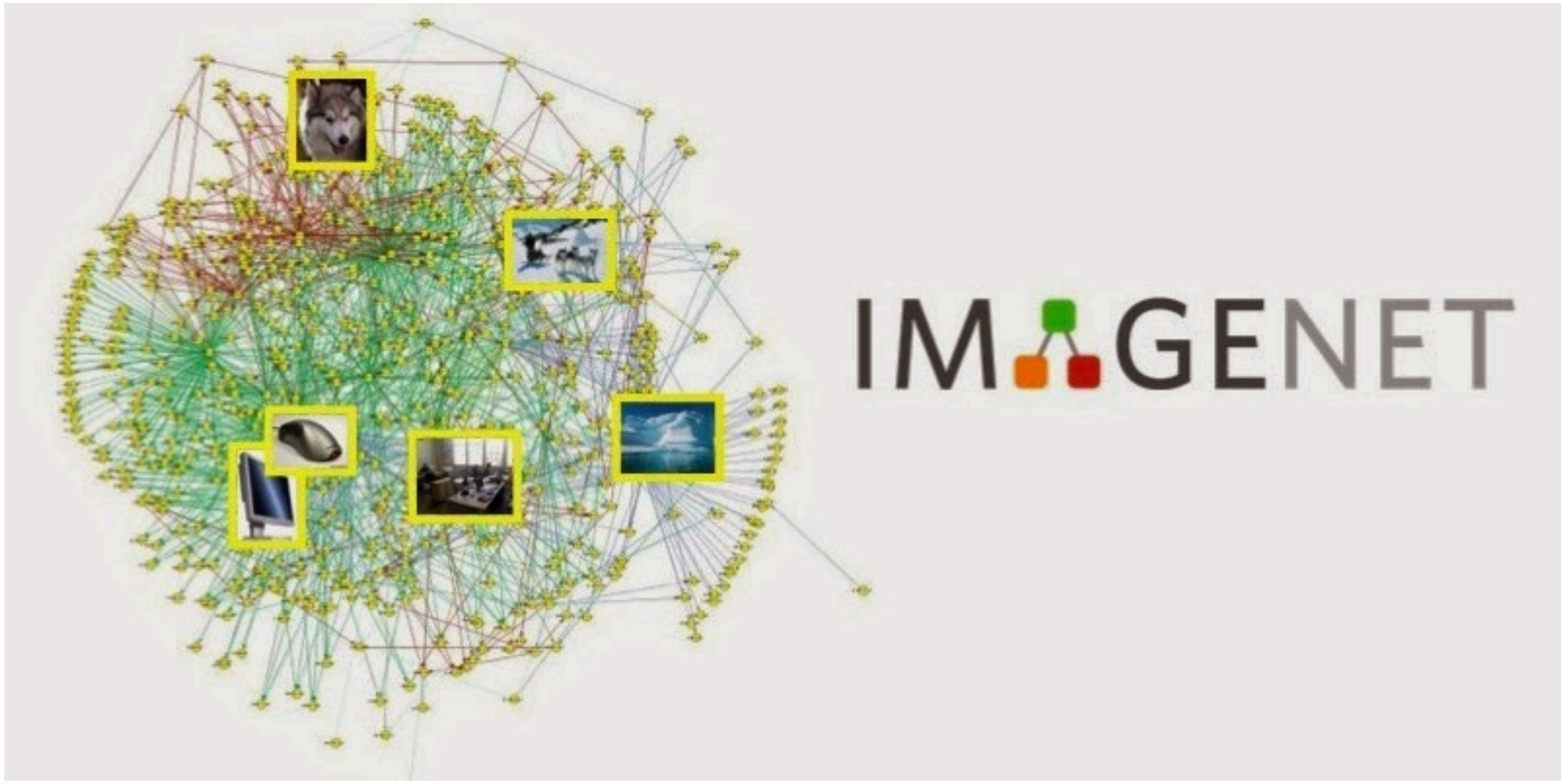
The parable of Google Flu: traps in big data analysis
http://science.sciencemag.org/content/343/6176/1203

Data driven applications

- Personalization services: Netflix, Amazon.com
- Google PageRank algorithm
- The Deep Learning revolution



The deep learning revolution

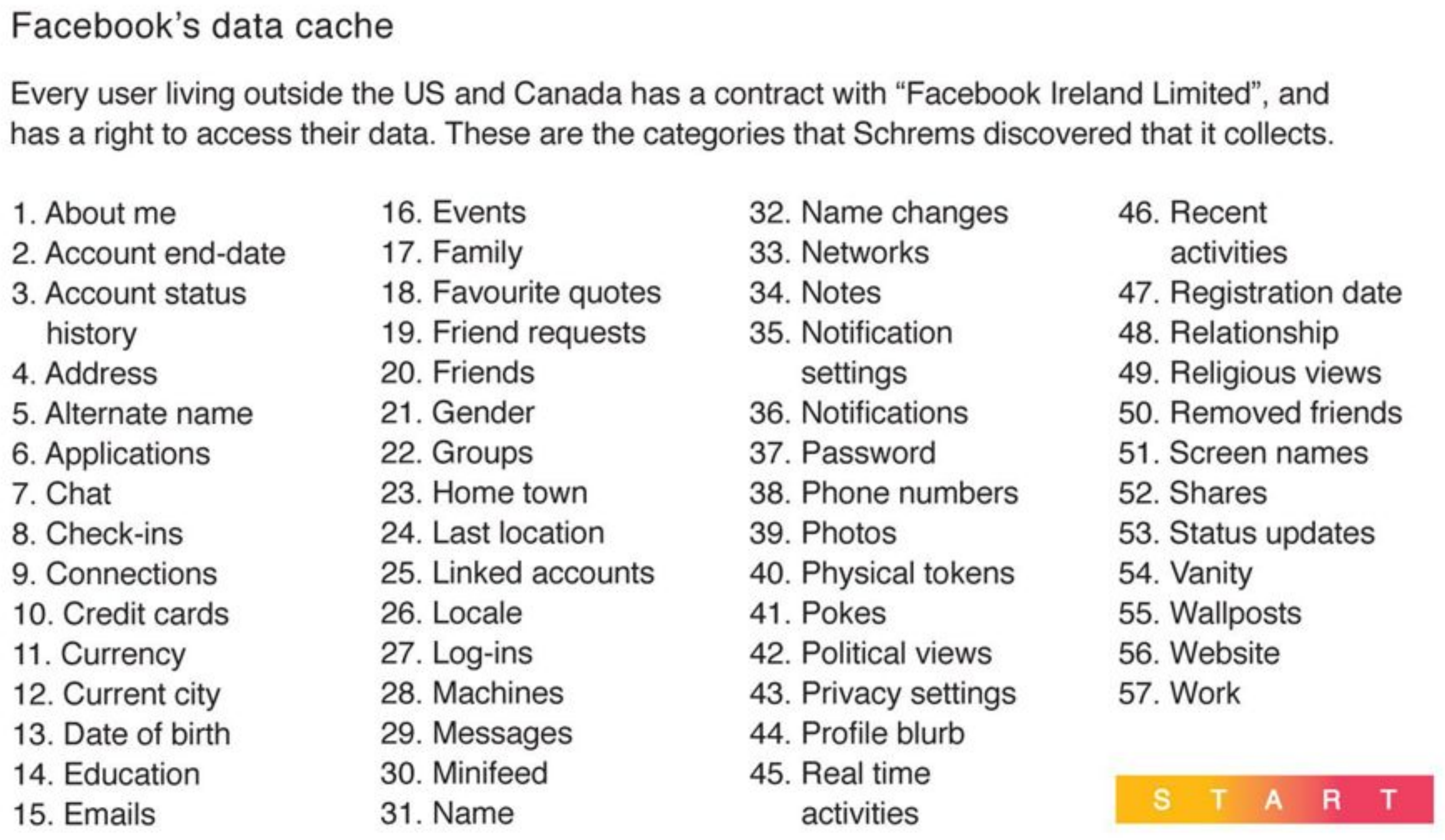


correlation structure of variables can be revealed,
but only by processing really huge amounts of data

Where does data come from?

- Public data sites: infochimps, factual, kaggle
- Public APIs: twitter, facebook
- Mobile apps events
- Activity devices: sports gear, traffic apps, city sensors, smart watches
- Shopping: point of sale devices
- Other: Next Generation Sequencing, climate data, public traffic vehicle info
- Future: self charged chips in skin for 1c

Facebook data cache: digital life events



Facebook's data cache

Every user living outside the US and Canada has a contract with "Facebook Ireland Limited", and has a right to access their data. These are the categories that Schrems discovered that it collects.

1. About me
2. Account end-date
3. Account status history
4. Address
5. Alternate name
6. Applications
7. Chat
8. Check-ins
9. Connections
10. Credit cards
11. Currency
12. Current city
13. Date of birth
14. Education
15. Emails
16. Events
17. Family
18. Favourite quotes
19. Friend requests
20. Friends
21. Gender
22. Groups
23. Home town
24. Last location
25. Linked accounts
26. Locale
27. Log-ins
28. Machines
29. Messages
30. Minifeed
31. Name
32. Name changes
33. Networks
34. Notes
35. Notification settings
36. Notifications
37. Password
38. Phone numbers
39. Photos
40. Physical tokens
41. Pokes
42. Political views
43. Privacy settings
44. Profile blurb
45. Real time activities
46. Recent activities
47. Registration date
48. Relationship
49. Religious views
50. Removed friends
51. Screen names
52. Shares
53. Status updates
54. Vanity
55. Wallposts
56. Website
57. Work

S T A R T

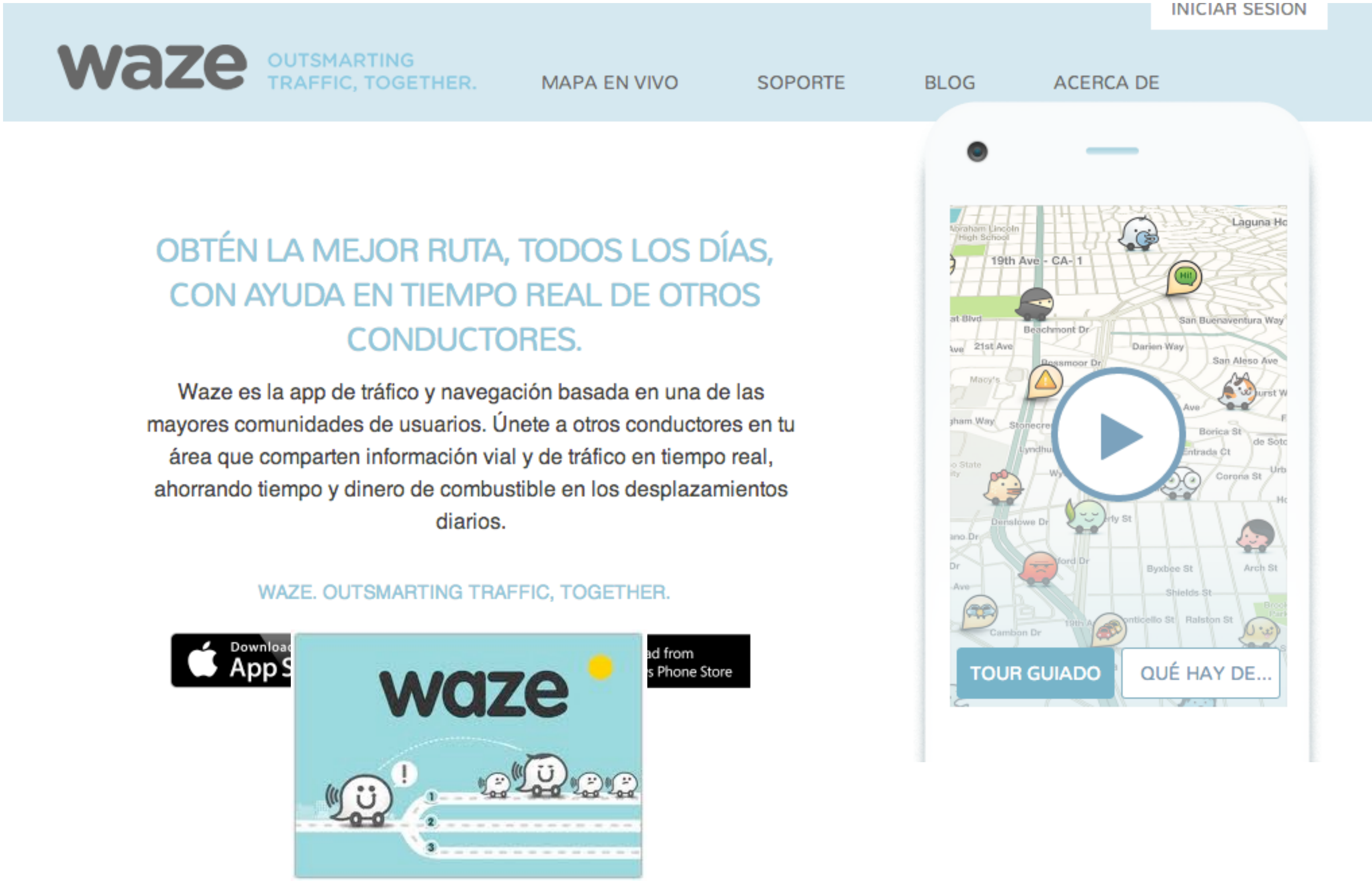
http://europe-v-facebook.org/EN/en.html

Facebook big data challenges

- **Sampling at logging time in a distributed environment** what is the accuracy of query results over sampled data? The entirety of the data is not available for comparison.
- **Trading storage space and CPU**
- **Reliability of pipelines** How do we improve reliability? Retry? Speculation? Checkpointing? Is there something we can take from the map-reduce framework without introducing too much overhead?
- **Globally distributed warehouse** local decisions about replication and lots of copies of important data
- **Time series correlation and anomaly detection** which algorithms should we use? Can we apply them to compressed data?

Applications with new kind of data needs

- **Waze** : real time data processing
- **Airbnb** : fast growing global application
- Recommendation engine : personalization of content



INICIAR SESIÓN

waze OUTSMARTING TRAFFIC, TOGETHER.

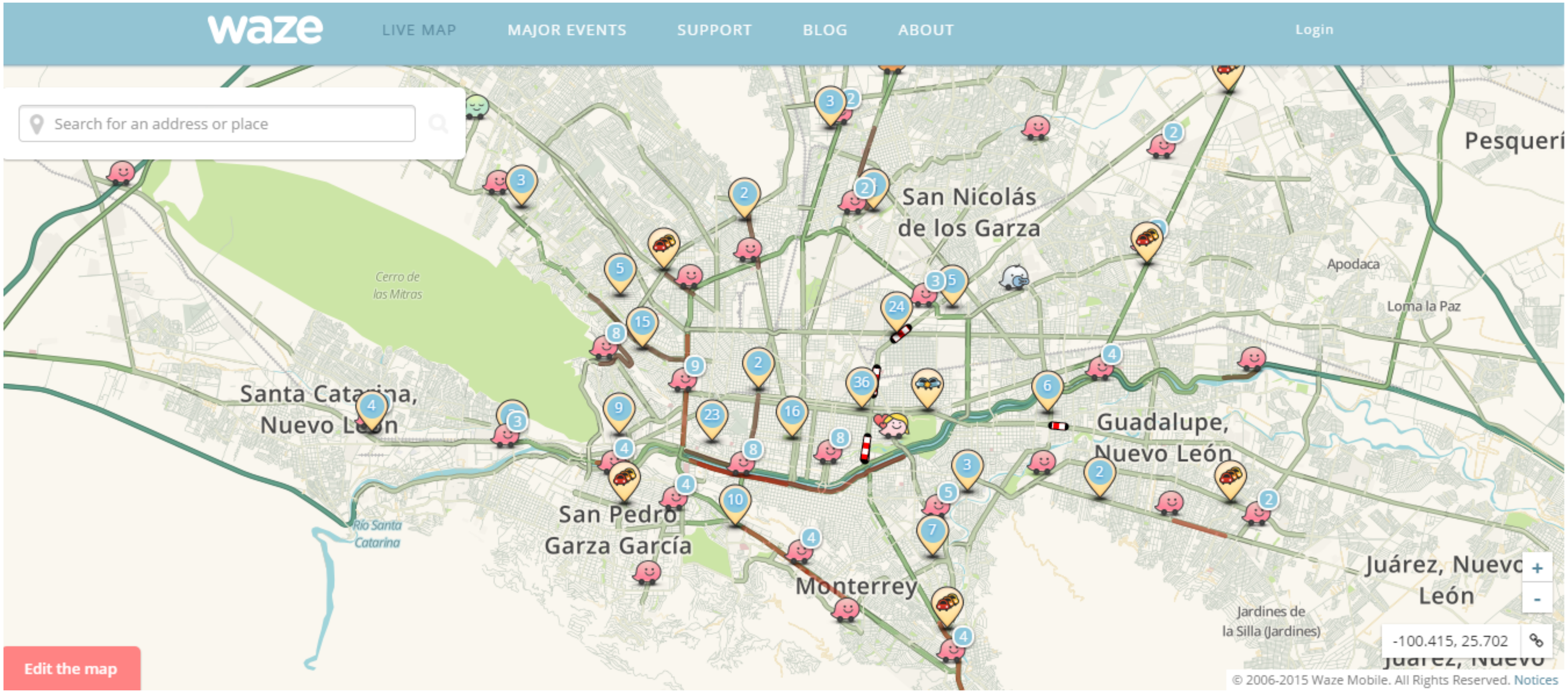
MAPA EN VIVO SOPORTE BLOG ACERCA DE

OBTÉN LA MEJOR RUTA, TODOS LOS DÍAS, CON AYUDA EN TIEMPO REAL DE OTROS CONDUCTORES.

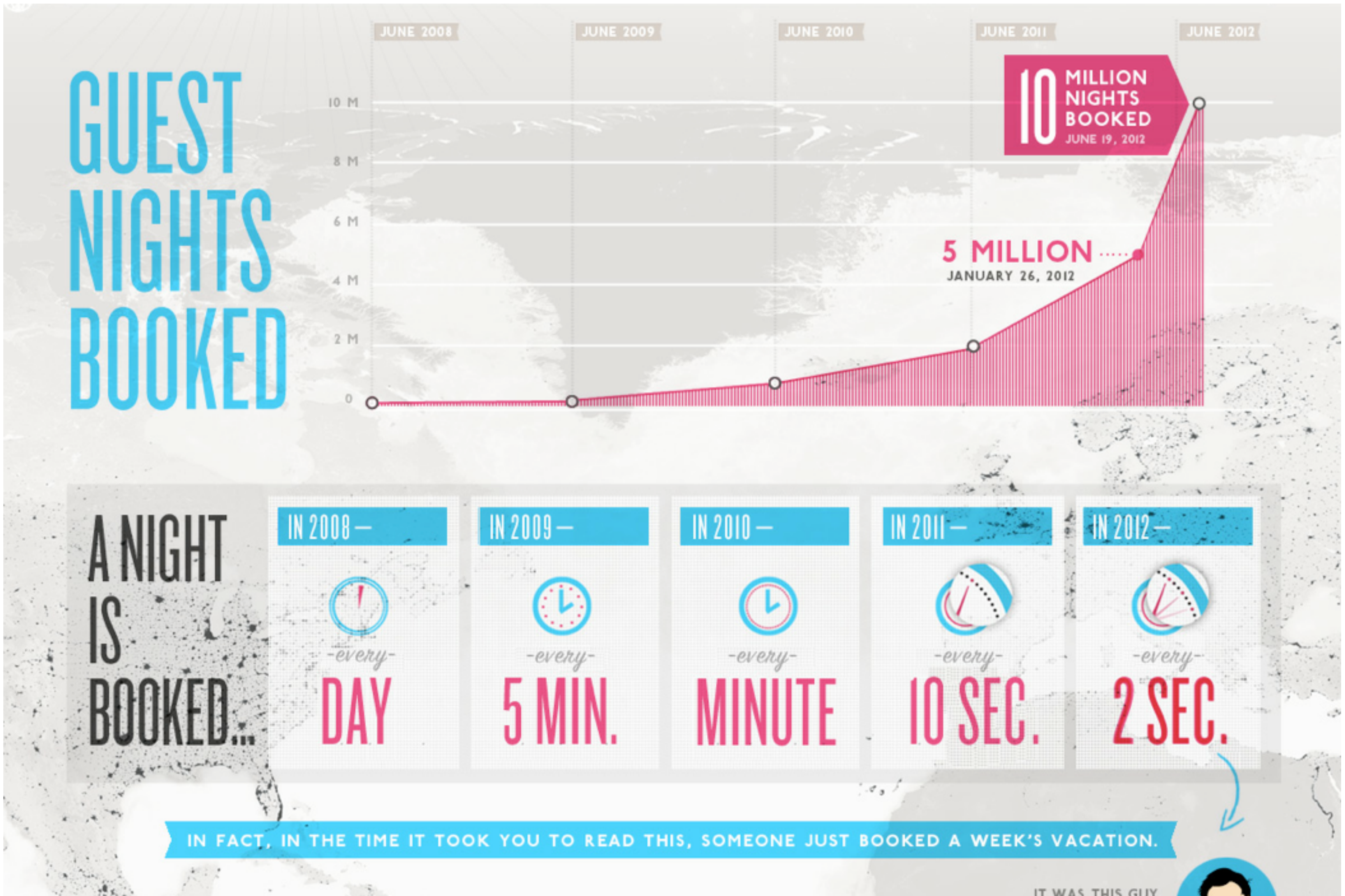
Waze es la app de tráfico y navegación basada en una de las mayores comunidades de usuarios. Únete a otros conductores en tu área que comparten información vial y de tráfico en tiempo real, ahorrando tiempo y dinero de combustible en los desplazamientos diarios.

WAZE. OUTSMARTING TRAFFIC, TOGETHER.

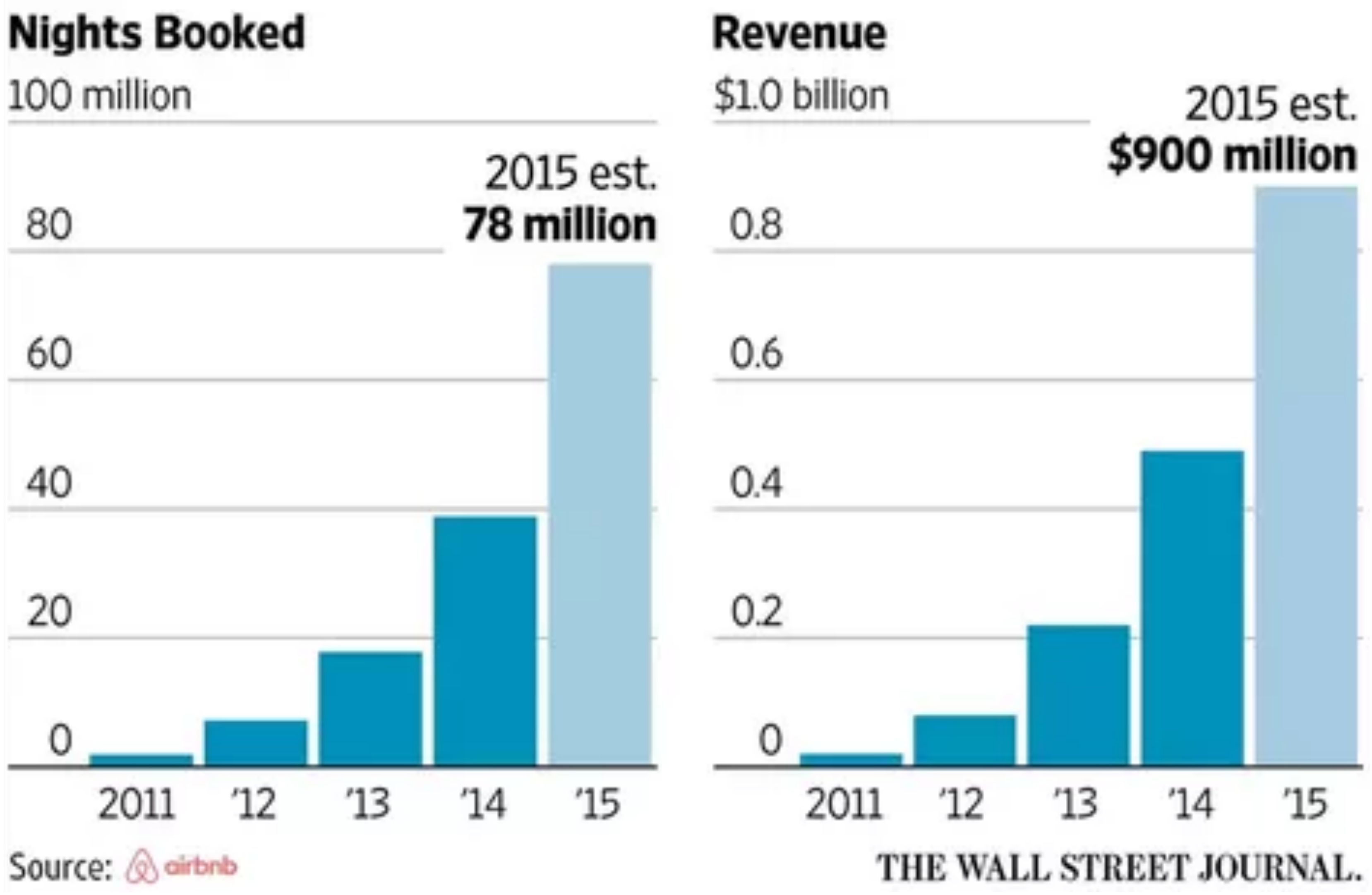
Waze live map



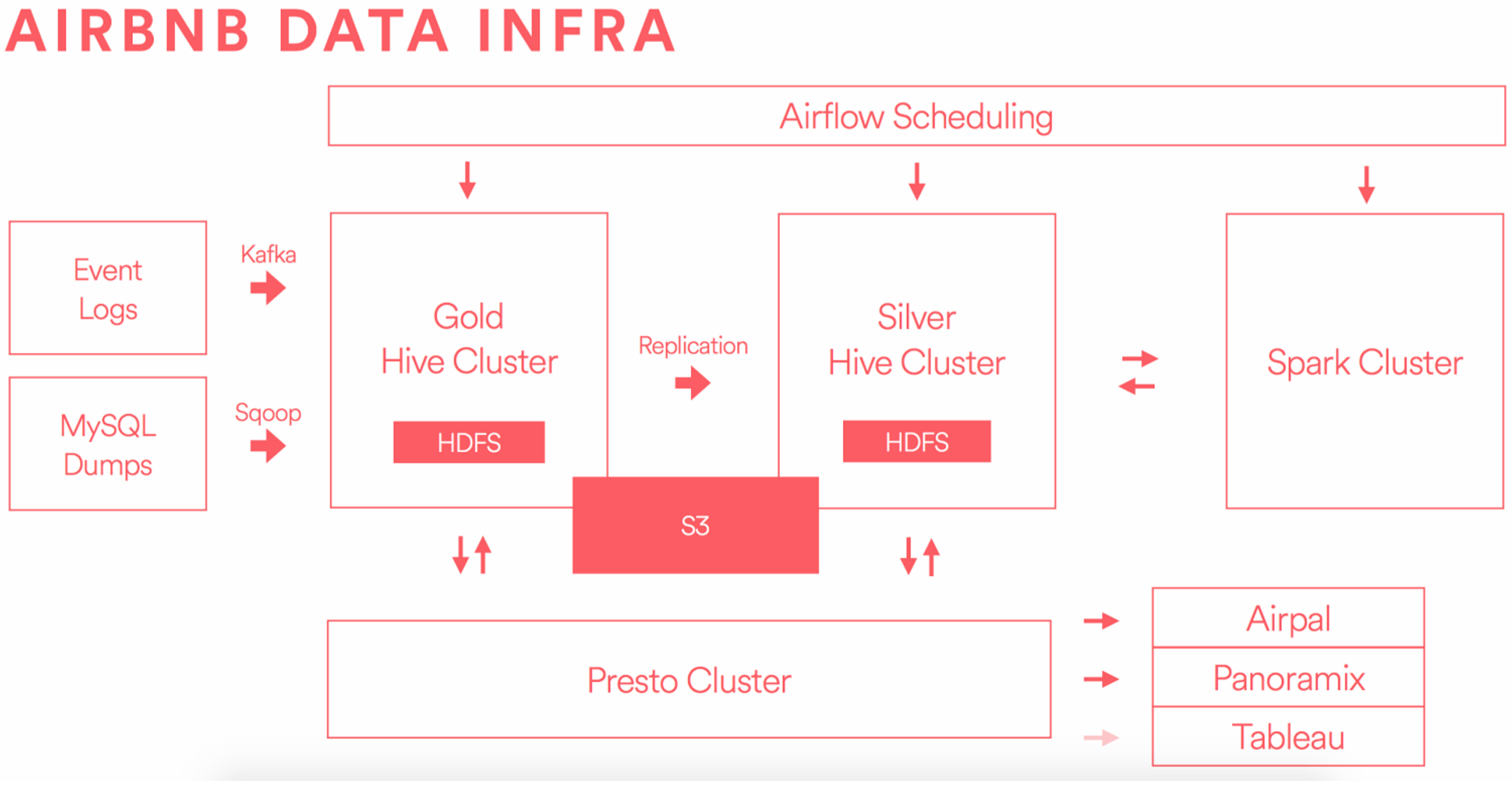
Airbnb: quick global growth



System architecture scalability

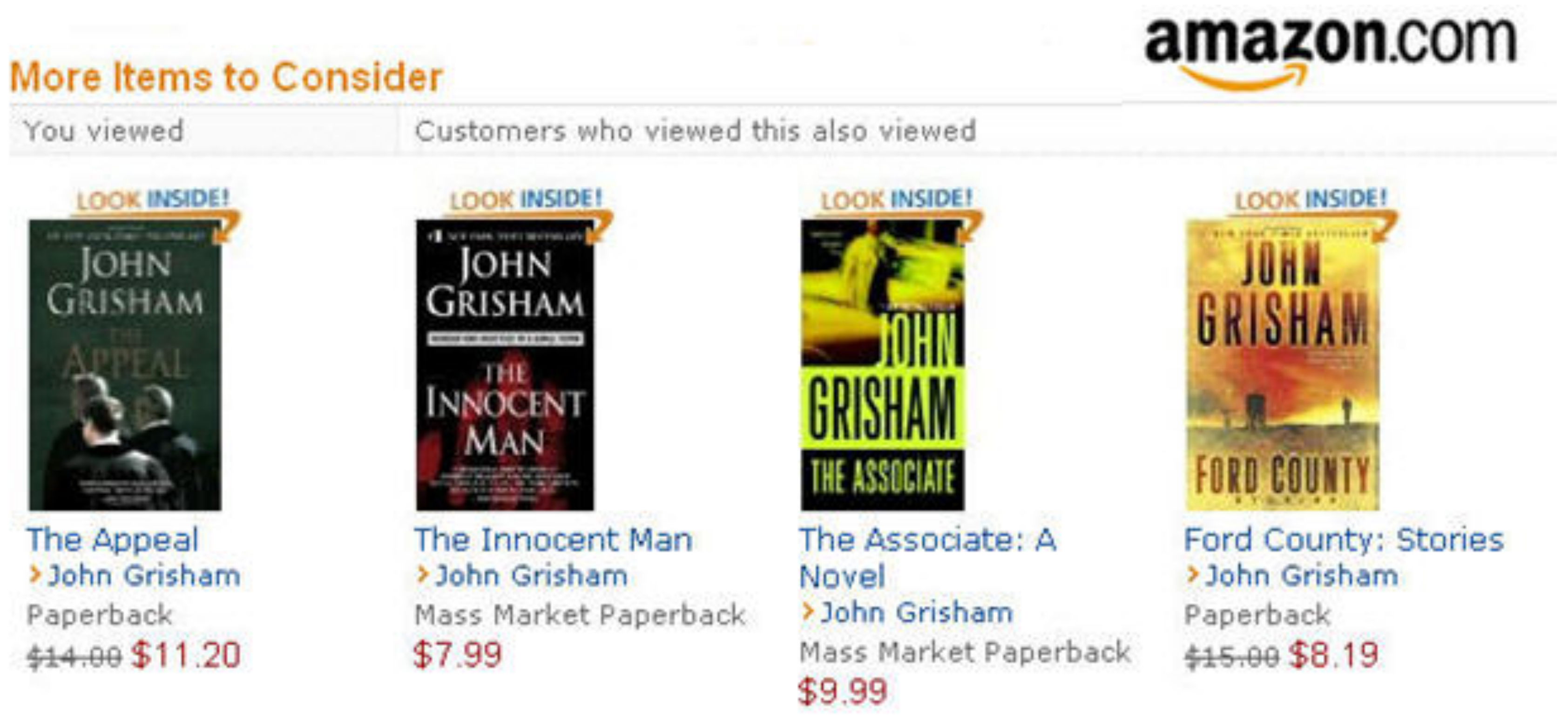


Airbnb data infrastructure



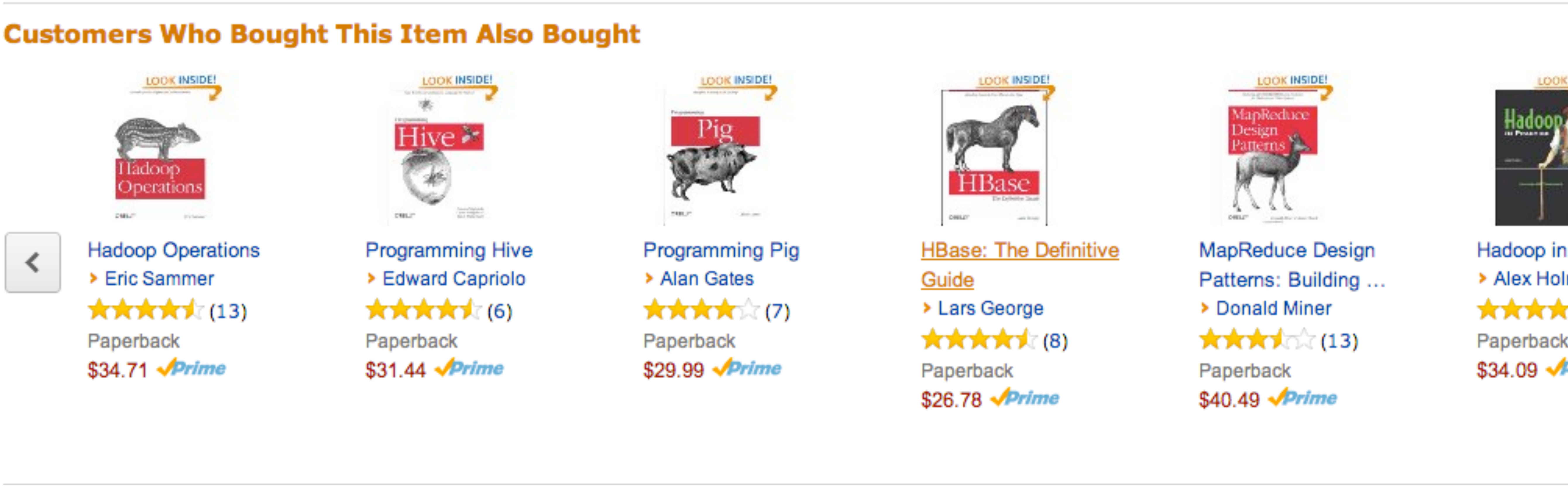
Data collected form users provides added value

- Amazon books is a trail of user visits to be correlated with users behavior
- Users are a in a feedback loop in which they contribute to the products in use
- Companies: How to use tracking info effectively?



You viewed	Customers who viewed this also viewed		
The Appeal	The Innocent Man	The Associate: A Novel	Ford County: Stories
John Grisham	John Grisham	John Grisham	John Grisham
Paperback	Mass Market Paperback	Mass Market Paperback	Paperback
~~$14.00~~ $11.20	$7.99	$9.99	~~$15.00~~ $8.19

Product recommendation



Customers Who Bought This Item Also Bought

Hadoop Operations
› Eric Sammer
★★★★½ (13)
Paperback
$34.71 ✓Prime

Programming Hive
› Edward Capriolo
★★★★½ (6)
Paperback
$31.44 ✓Prime

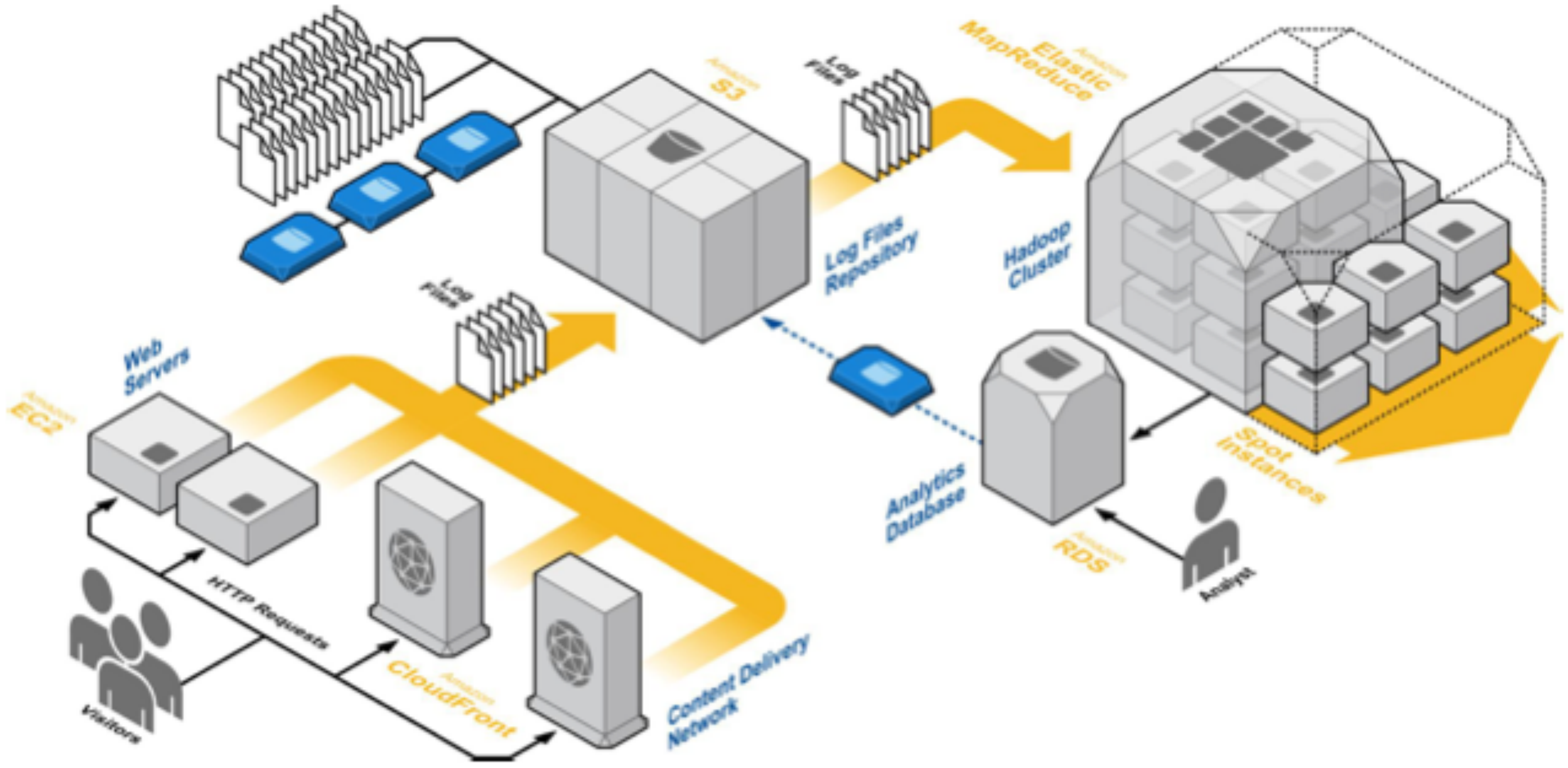
Programming Pig
› Alan Gates
★★★★☆ (7)
Paperback
$29.99 ✓Prime

HBase: The Definitive Guide
› Lars George
★★★★½ (8)
Paperback
$26.78 ✓Prime

MapReduce Design Patterns: Building …
› Donald Miner
★★★½☆ (13)
Paperback
$40.49 ✓Prime

Hadoop in
› Alex Hol
Paperback
$34.09

Recommending system software architecture



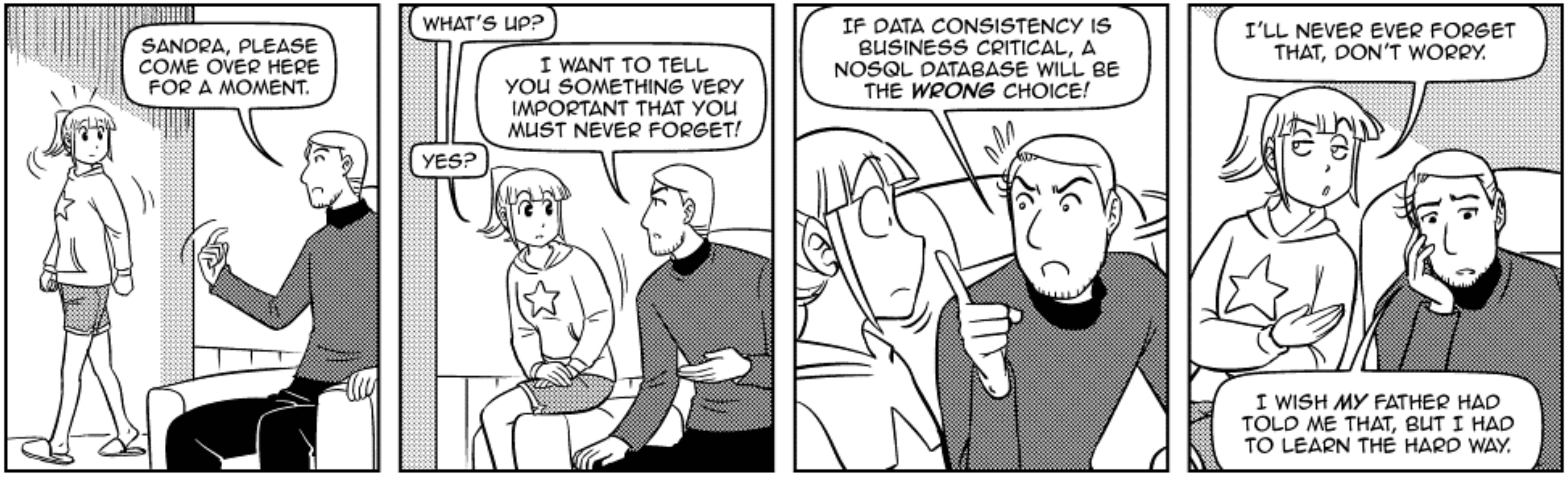
Working with data at scale

Working with data at scale

- The size of the data becomes part of the problem
- Information platforms, data spaces
- Designed for exploring and interpreting data
- Accept many data formats
- Data schemas evolve, not closed
- Beyond the relational database model

No SQL databases

- Consistency is not relevant anymore: Google search results, exact number of followers are approximate


SANDRA, PLEASE COME OVER HERE FOR A MOMENT.
WHAT'S UP?
I WANT TO TELL YOU SOMETHING VERY IMPORTANT THAT YOU MUST NEVER FORGET!
YES?
IF DATA CONSISTENCY IS BUSINESS CRITICAL, A NOSQL DATABASE WILL BE THE WRONG CHOICE!
I'LL NEVER EVER FORGET THAT, DON'T WORRY.
I WISH MY FATHER HAD TOLD ME THAT, BUT I HAD TO LEARN THE HARD WAY.

Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

Why not relational databases?

- Not effective at very large scale
- Multi server replication and data partitions are difficult to merge
- Hard restriction on static data schemas
- Analysis is usually comparative
- Correlation versus cause-effect

The new data servers: No SQL

- Descendants of Google BigTable and Amazon Dynamo
- Designed to be distributed among many nodes
- Provide partial, not absolute consistency
- Very flexible database schema
- More than 50 open source projects: Cassandra, Hbase, CouchDB, mongoDB, Apache spark, …

Developing Big Data applications

- How to build a Big data application?
- Which are the main parts of the system?
- Where is data stored?
- How can queries be distributed?
- How can all data be analysed and provided near-real time answers?

Scaling with a traditional database

- Web analytics application: track the number of page views to any URL a customer wishes to track
- The customer's web page pings the application's web server with its URL every time a page view is received
- The application should be able to tell you at any point what the top100 URLs are by number of page views

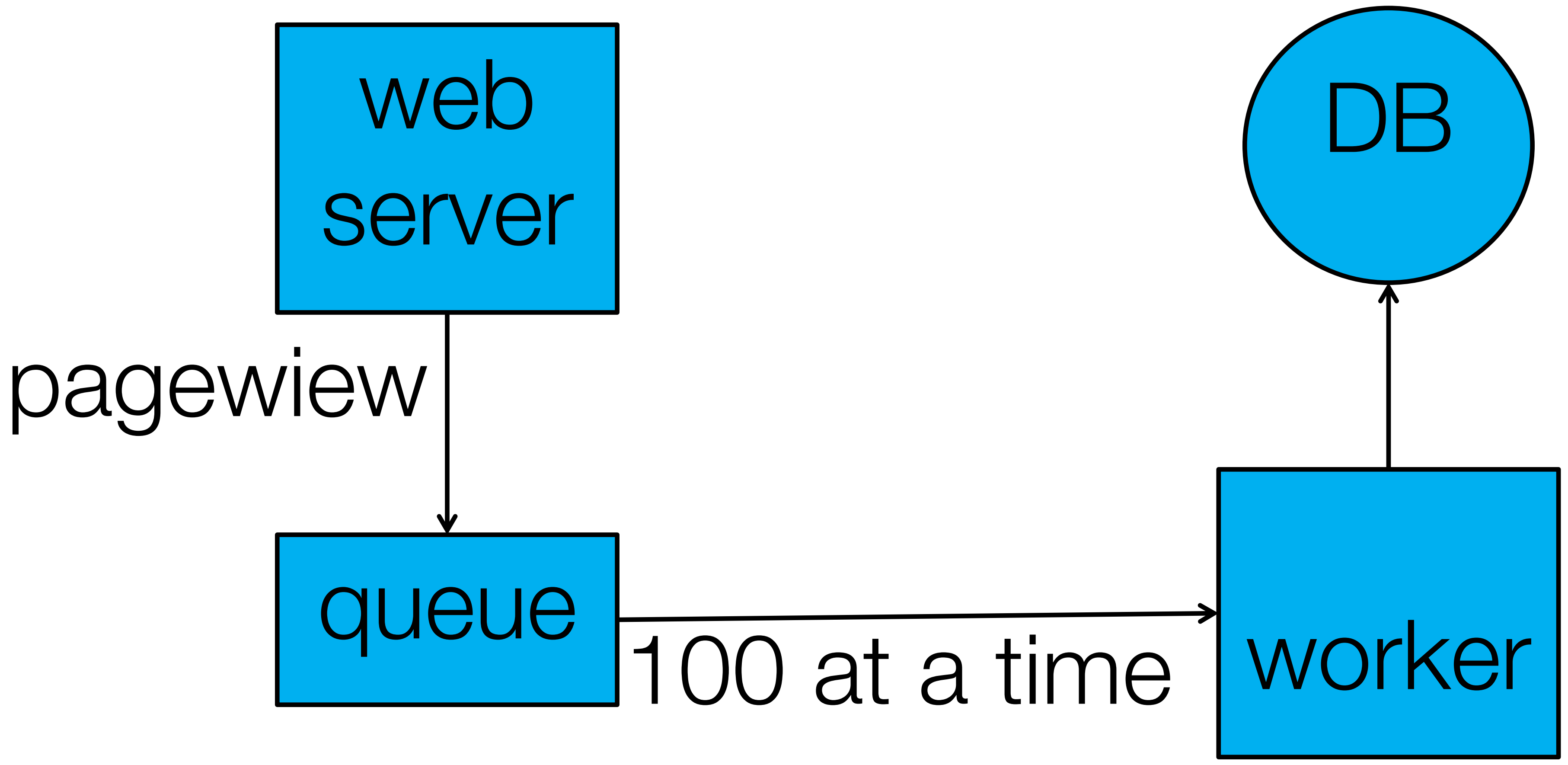
Traditional database schema for page views

- Web server increments corresponding row of the DBMS

COLUMN NAME	TYPE
ID	integer
USER ID	integer
URL	varchar(255)
PAGE VIEWS	bigint

Scaling with a queue

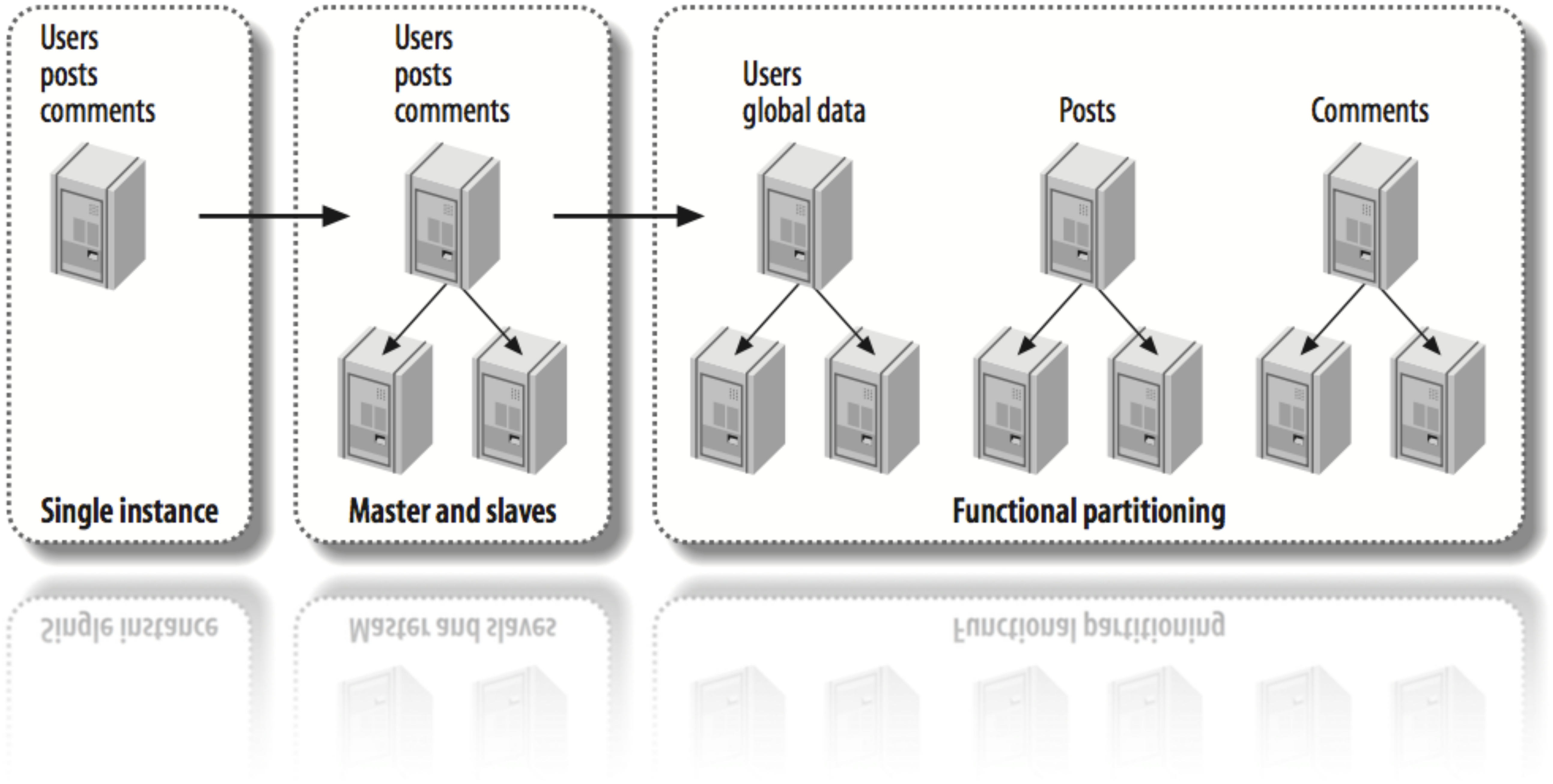
- “Timeout error on inserting to the database”: the database can’t keep up with the load so write requests to increment pageviews are timing out
- Batching updates with queue between web server and database



Service gets more requests

- Problem: how to scale heavy-write relational database
- Best approach is to use multiple database servers and spread the table across all the servers
- Each server will have a subset of the data for the table. This is known as "horizontal partitioning" also known as sharding
- Sharding technique spreads the write load across multiple servers

Sharding the database



Sharding implications

- **Mapping keys to shards** using a hash function causes the keys to be evenly distributed across the shards
- A program must map all the rows in a single database instance and split the data into shards
- Application code needs to know how to find the shard for each key
- Wrap a library around your database handling code that reads the number of shards for a configuration file and redeploy all application code
- Now modify top 100 URLs query to get the top 100 URLs from each shard and merge those lists together for the global top 100 URLs

More sharding implications

- Keep having to reshard the database into more shards to keep up with the increasing write load
- Can’t just run one script to do the resharding process. That would be too slow. Do all resharding in parallel and manage many active worker scripts
- If you forget to update the application code with the new number of shards, many increments will be written to the wrong shards: write a one-off script to manually go through the data and move whatever has been misplaced

Limitations of standard DB technology

- **Fault tolerance** is hard: if a shard machine fails, it is not possible to write results
- **Complexity** is pushed to the application layer: aplication needs to know which shard to look at for each key
- Lack of human **fault tolerance**: nothing prevents you from reading or writing data from the wrong shard, and logical bugs can irreversibly corrupt the database
- **Maintenance** requires a large amount of work: database should be self-aware of its distributed nature and manage the sharding process automatically

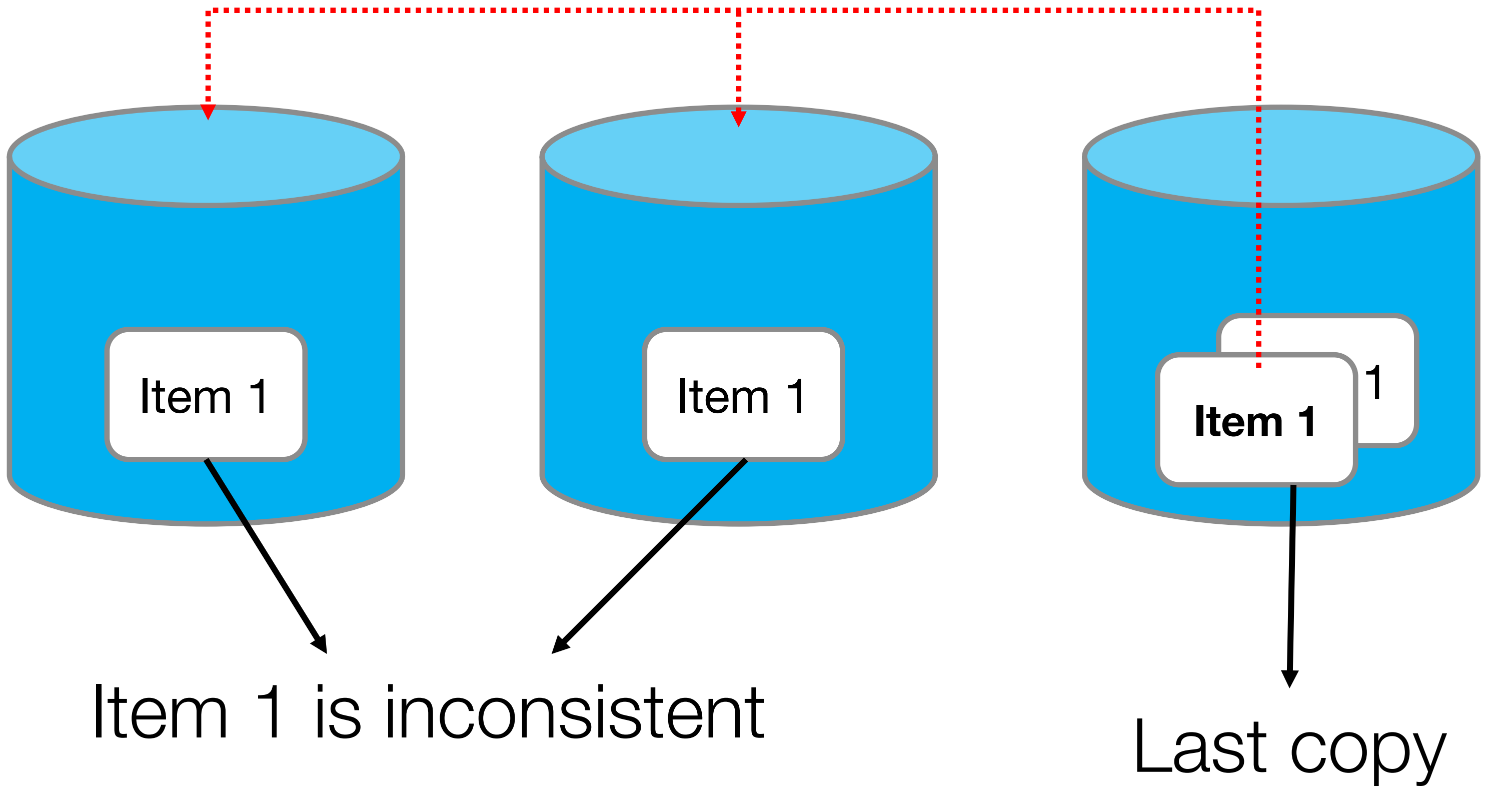
Consistency and NoSQL databases

- Stored data in database must be always trustful
- Any operation in the database does not invalid previous state
- When facing a new operation we must fulfill consistency:
  - By reaching a new consistent state
  - By rollback to the previous consistent state

Consistency models

- **Strict**: changes in data are atomic and affect database inmediately
- **Sequential**: each client applies changes in their occurring order
- **Causal**: changes are applied following causality order
- **Eventual**: changes are propagated when queues are empty
- **Weak**: no assurance of changes following causality

Consistency when data is modified?



CAP Theorem

- Distributed shared data systems properties
- When dealing with data modifications we can have only TWO of these properties:
- Data **Consistency**
- Data **Availability**
- Tolerance to data **Partitioning**

Consistency vs Availability

- In large systems data is highly physically distributed
- Two options:
    - **Consistency**: data will be not be available when not updated
    - **Availability**: data read may not have the last updated value

Eventual Consistency

- Amazon Dynamo and CassandraDB
- Optimistic view of updates
- Weak consistency model: allows clients to update items at any moment
- Assures that updates are going to be done in a consistency-save way

Entering Big Data alternative world, …

Back to basics: what does a data system do?

- **Answer questions** based on information that was acquired in the past
    - A social network profile answers questions like “what is the person’s name?” and “How many friends does this person have?”
- **Combine bits and pieces together to produce their answers**
    - A bank account balance, for example, is based on combining together the information about all the transactions on the account
- Not all bits of information are equal. **Some information is derived from other**
    - A friend count is derived from the friend list, and the friend list is derived from all the times the user added and removed friends from the profile

What is data?

- When you keep tracing back where information is derived from, you eventually end up at the most raw form of information – that was not derived from anywhere else
- This is the information you hold to be true simply because it exists. Let's call this information: "data"
- **"DATA" is that special information from which everything else is derived**
- **You answer questions on your data by running functions that take data as input**
    - Your function that answers the "friend count" question can derive this count by looking at all the add and *remove friend* events

query = function(all data)

- The most general purpose data system can answer questions by running functions that take in **the entire dataset as input**
- The most general purpose definition of a query is:
- **query= function(all data)**

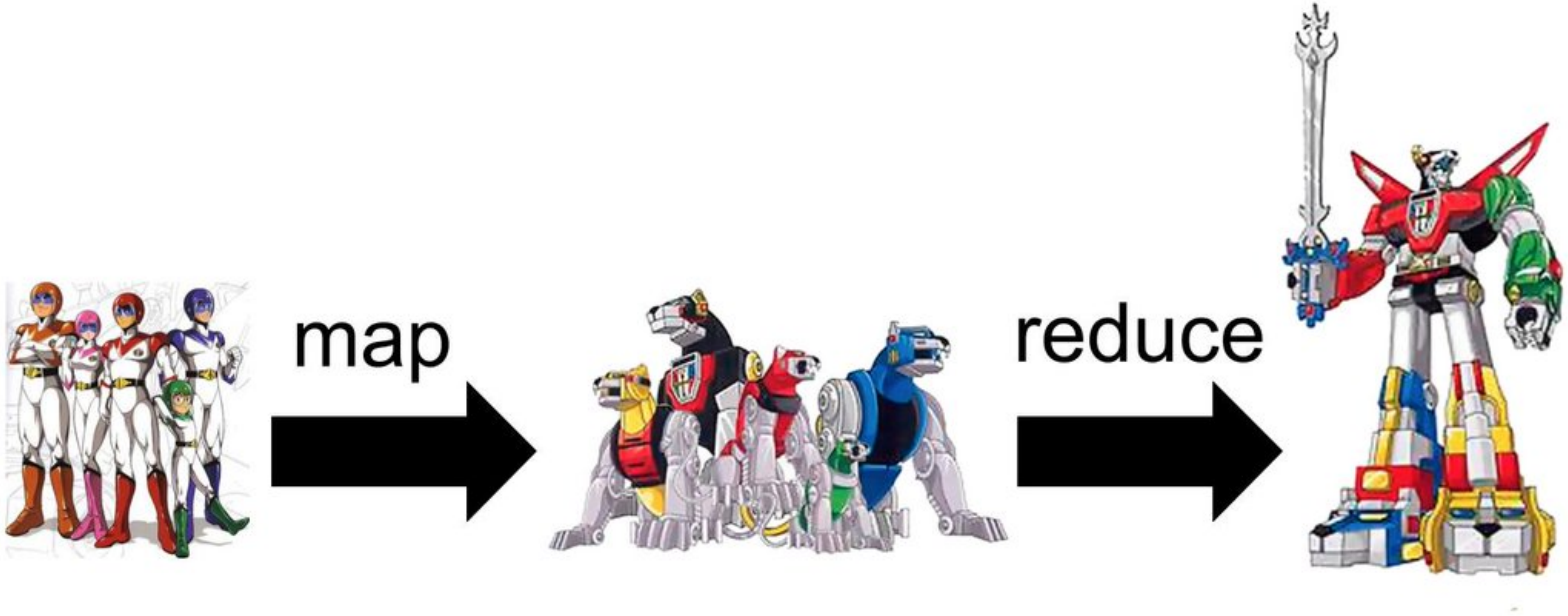
Big data systems desired properties

- **Robust and fault tolerant**: servers go down, corrupt values in database
- **Low latency** reads and updates: reads require to be satisfied with very low latency. Update latency requirements vary a great deal between applications
- **Scalable**: maintain performance in the face of increasing data and / or load by adding resources to the system
- **General**: useful for a variety of applications

Big Data systems desired properties

- **Extensible**: allow functionality to be added with a minimal development cost
- **Allow ad hoc queries**: nearly every large dataset has unanticipated values within it
- **Minimal maintenance required**: choosing components that have the minimum implementation complexity as possible. Simple algorithms and simple components

Welcome to the map reduce world



Map Reduce: batch processing

- A programming model for processing large data sets
- Framework for processing parallelizable problems across huge datasets using a large number of computers
- "Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes
- "Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output

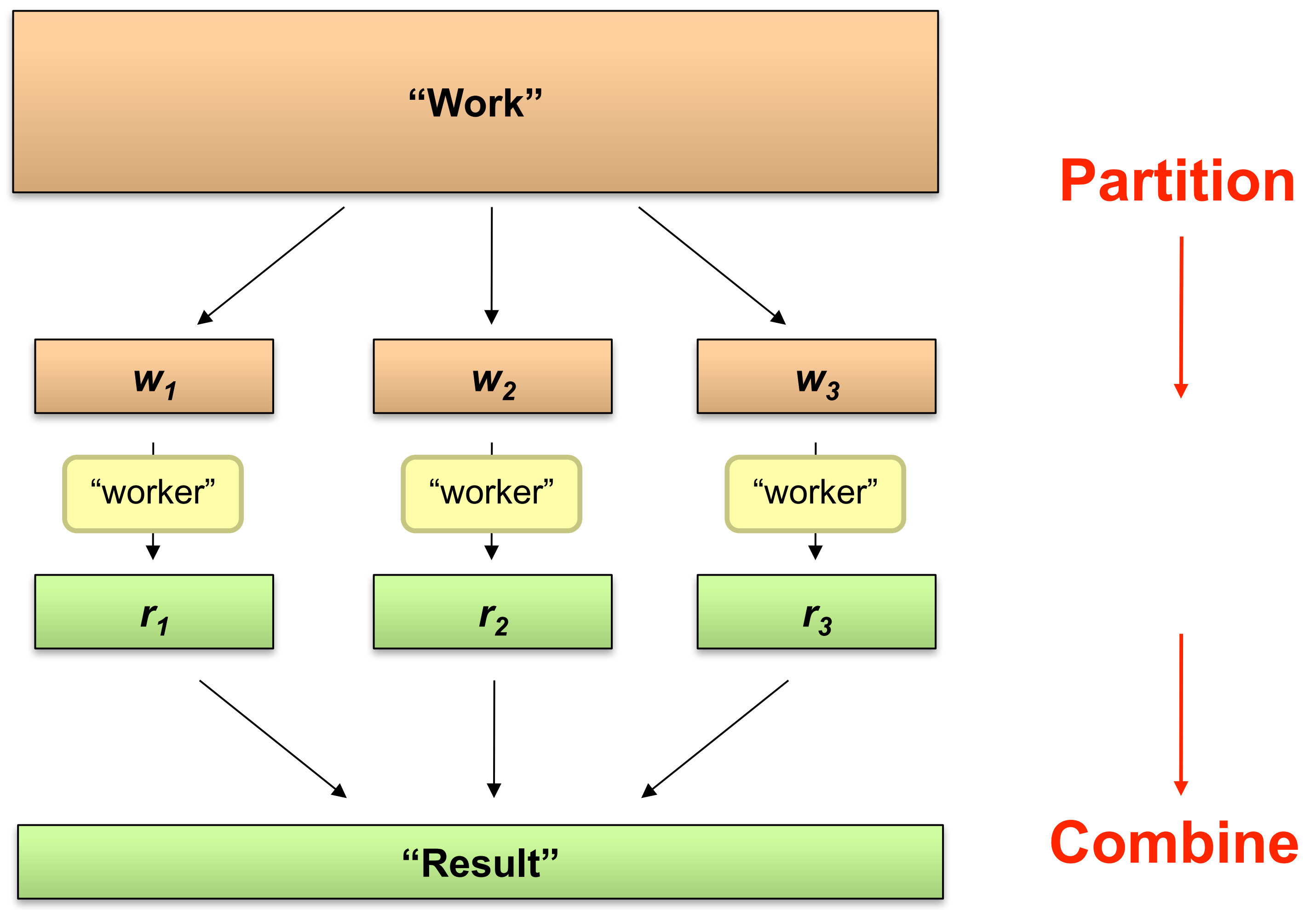
HADOOP

- An open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license
- Implements a computational paradigm named map/reduce
- Provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster
- It enables applications to work with thousands of computation-independent computers and petabytes of data
- Platform consists of: Hadoop kernel, MapReduce and Hadoop Distributed File System (HDFS), as well as a number of related projects – including Apache Hive, Apache HBase, and others

MapReduce

- **MapReduce** is a programming model
- Google has private implementation in C++
- Patent: US7650331
- **Hadoop** is an open source implementation in Java: *hadoop.apache.org*
  - Project led by Yahoo
  - First beta Release, dec 2011
  - First product release: 2013

Large problems



Map Reduce

Typical Large data problem

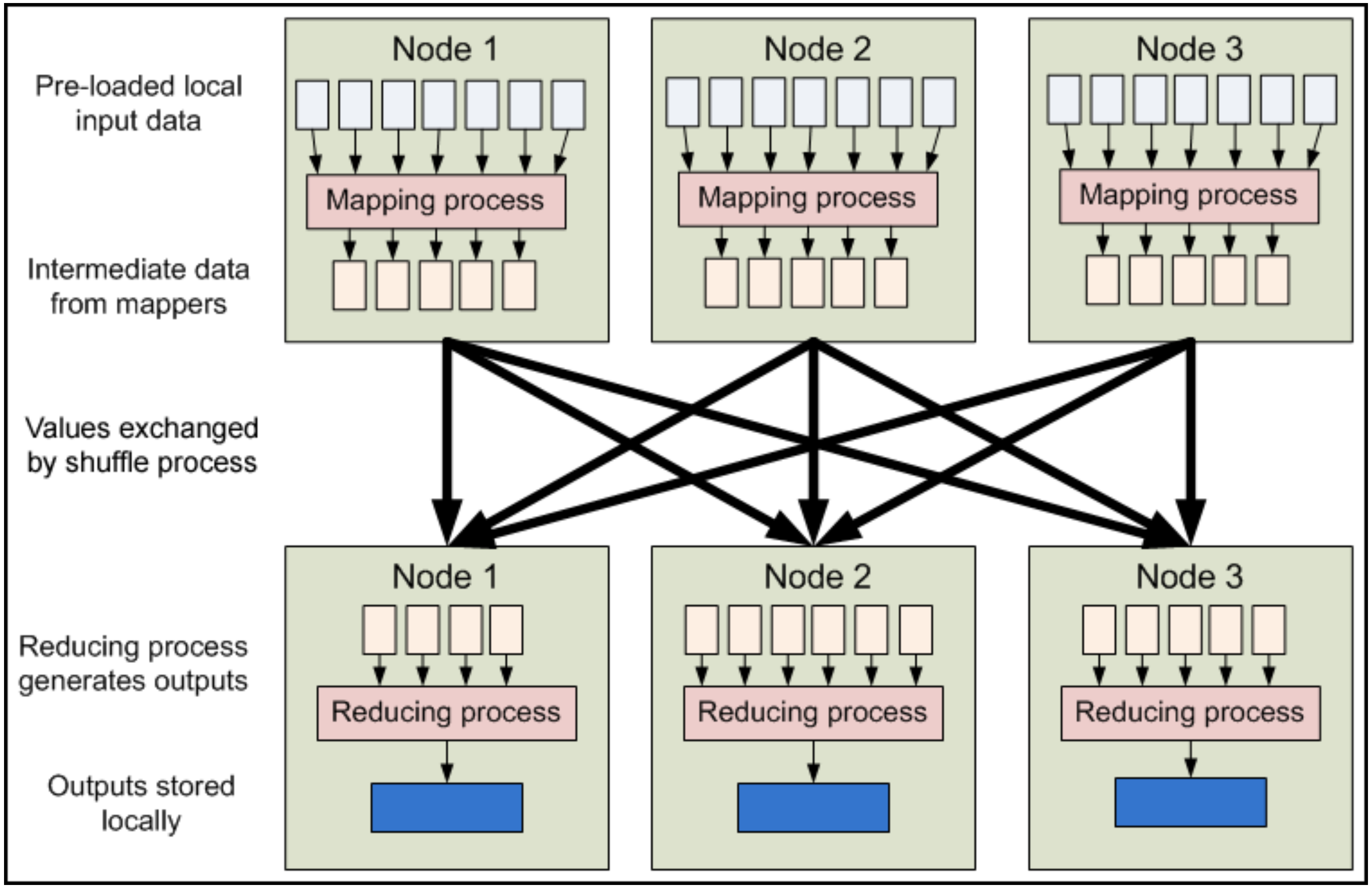
- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Hadoop word count example

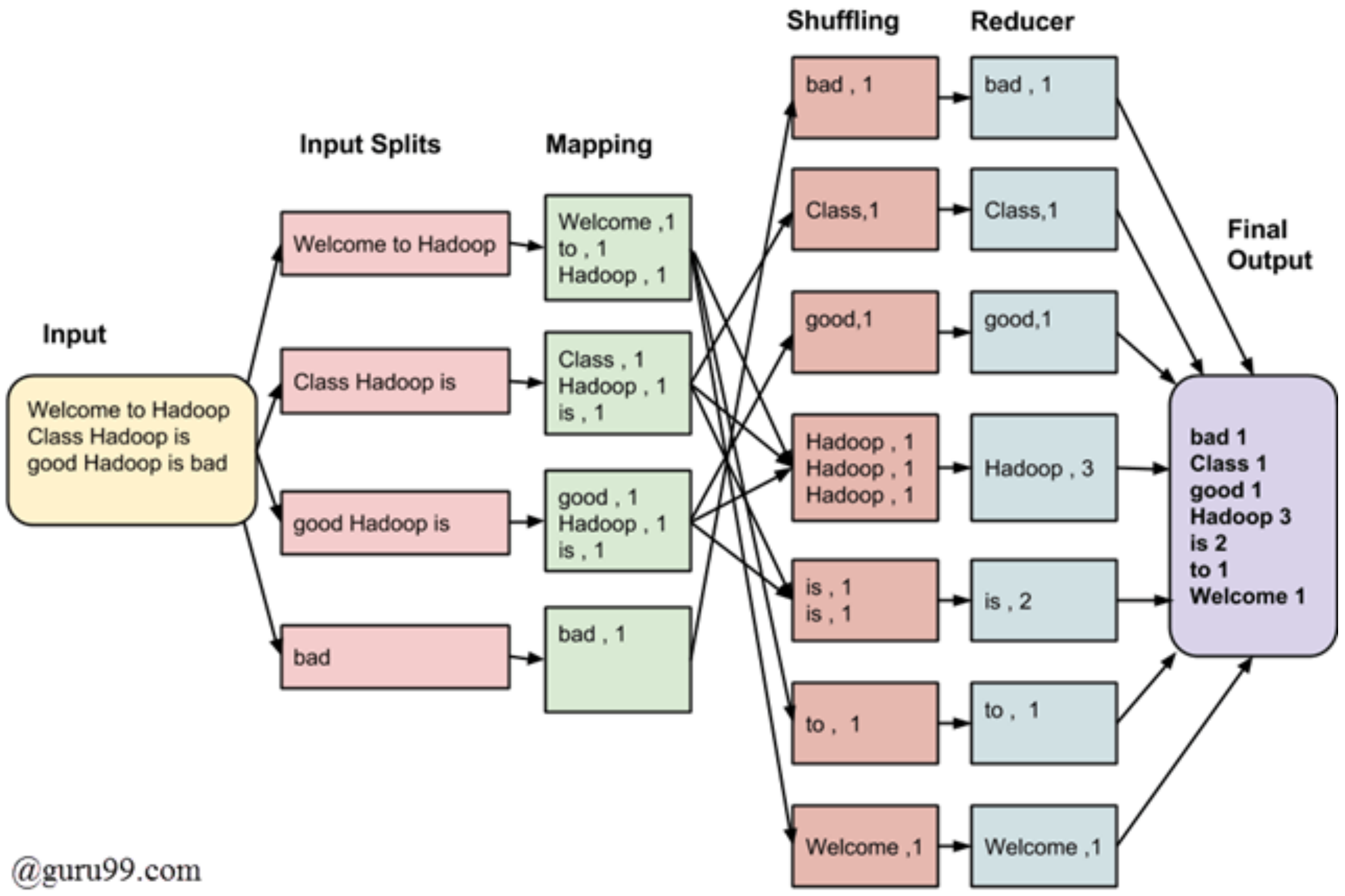
```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
    emit(term, sum);
```

Map Reduce system overview



Hadoop logical view



Hadoop runtime

- Handles scheduling: map and reduce tasks assigned to workers
- Handles data distribution: moves processes to data
- Handles synchronization: Gathers, sorts, shuffles intermediate data
- Handles faults: detects worker failures and restarts
- Everything happens on top of a distributed FS: hdfs

Hadoop components

- One of the things that make **MapReduce** so powerful is the fact that it is made of not just map and reduce tasks but rather multiple components working together.

- Each one of these components can be extended by the developer

Hadoop basic blocks

- Map: inputFormat: (getSplits, getReader), recordReader, Mapper.map, partitioner, combiner

- Reduce: shuffle, Reducer.reduce

Map phase components

- **InputFormat**: MapReduce jobs access their data through the InputFormat class. InputFormat implements two important methods:
- **getSplits**: This method implements the logic of how input will be distributed between the map processes. The most commonly used InputFormat is the *TextInputFormat* which will generate an input split per block and give the location of the block to the map task. The framework will then execute a mapper for each of the splits. This is why developers usually assume the number of mappers in a MapReduce job is equal to the number of blocks in the data set it will process.
- **getReader**: This method provides a reader to the map task that allows it to access the data it will process. Since the developer can override this method, it means that MapReduce can support any data type. As long as you can provide a method that reads the data into a writable object, you can process it with the MapReduce framework.

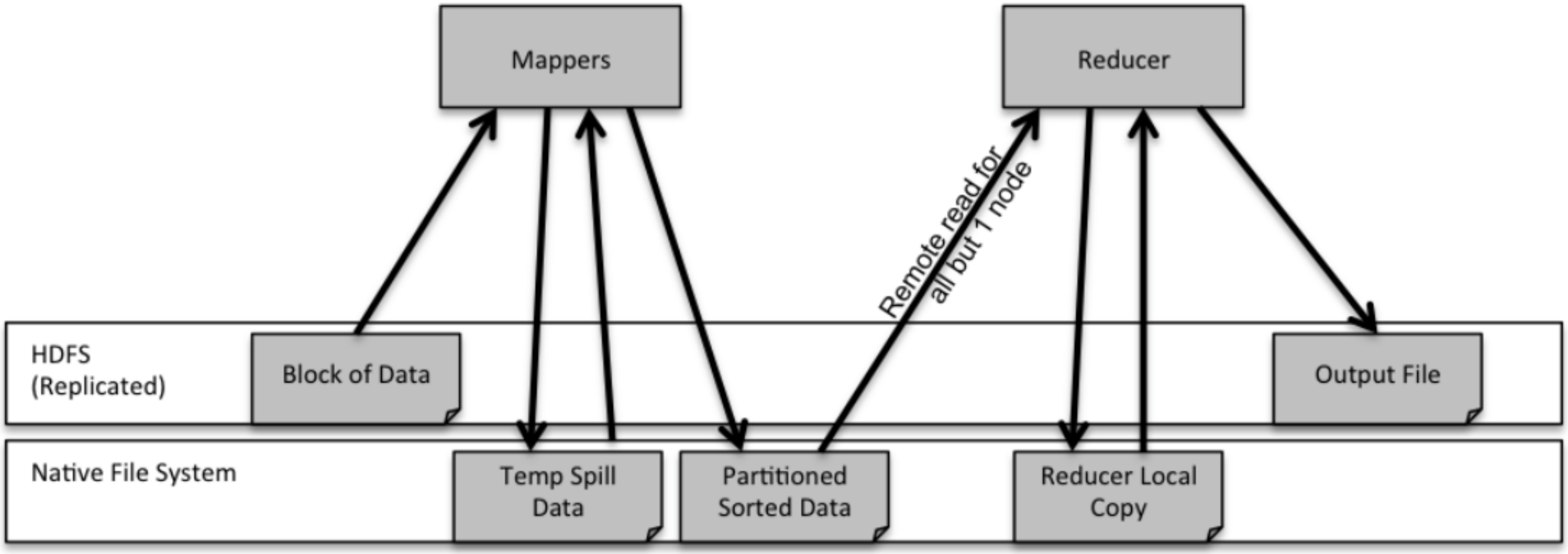
Map phase components

- **RecordReader**: This class reads the data blocks and returns key-value records to the map task.

- **Mapper.map** The map function is the heart of the mapper. Even if you decide to use the defaults and not implement any other component of the map task, you will still need to implement a map function. The map function has three inputs: key, value, and a context.

- When the map task writes to the reducer, the data it is writing is buffered and sorted. MapReduce will attempt to sort it in memory, with the available space defined by io.sort.mb configuration. If the memory buffer is too small, not all the output data can be sorted in memory. In this case the data is spilled to the local disk of the node where the map task is running and are sorted on disk

Map disk usage



Map phase components

- **Partitioner:** implements the logic of how data is partitioned between the reducers. The default partitioner will simply take the key, hash it using a standard hash function and divide by the number of reducers.
    - The remainder will determine the target reducer for the record. This guarantees equal distribution of data between the reducers.
    - If there is any requirement for keeping certain values together for processing by the reducers, you will need to override the default and implement your custom partitioner.
- **Combiner:** easy method to reduce the amount of network traffic between the mappers and reducers.
    - It can aggregate the values produced by the mapper.
    - It executes locally, on the same node where the mapper executes, so this aggregation reduces the output that is later sent through the network to the reducer

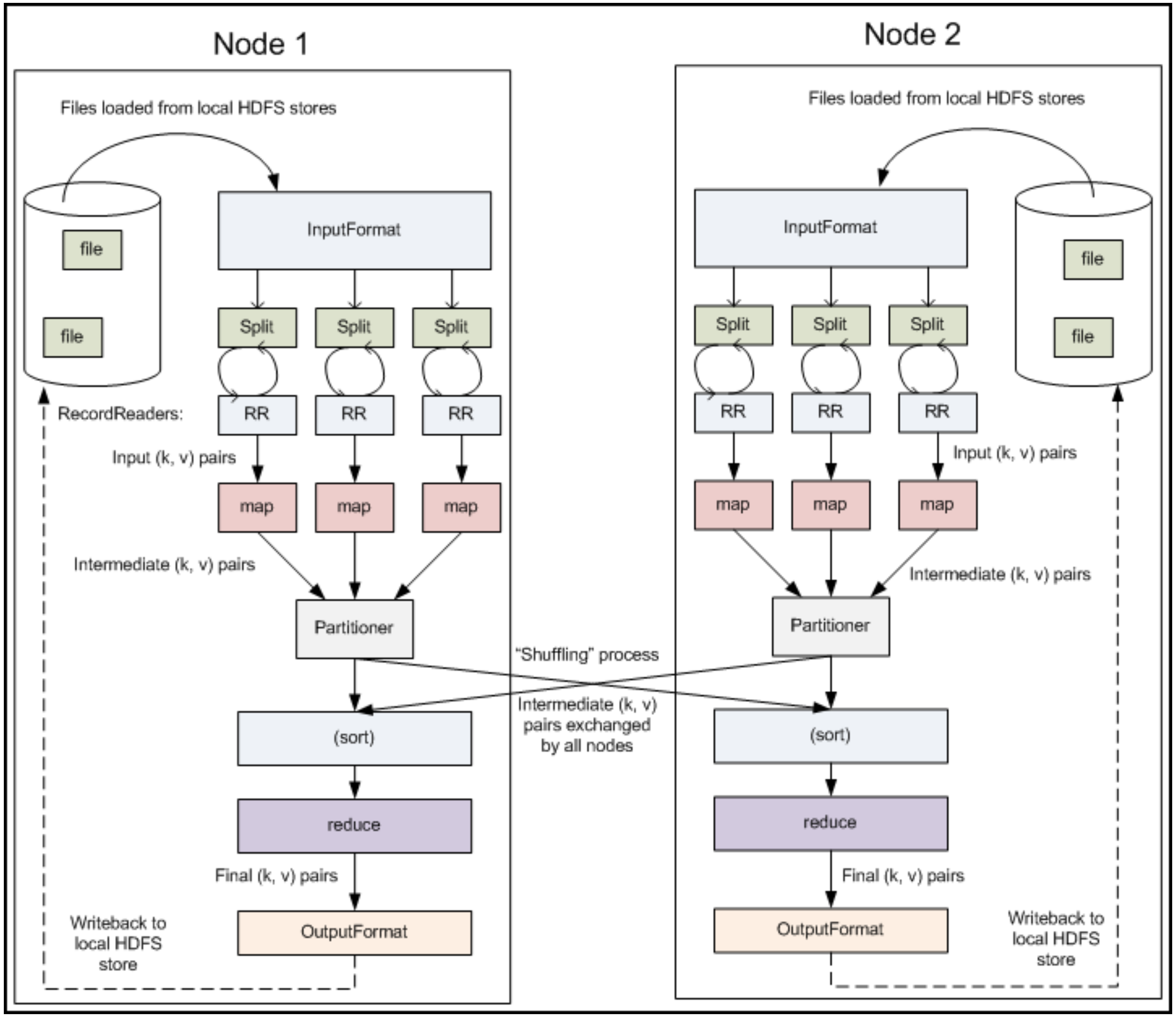
Reducer phase components

- **Shuffle:** Before the reduce stage begins, the reduce tasks copy the output of the mappers from the map nodes to the reduce nodes
- **Reducer.reduce:** Similar to the map method in the mapper, the reduce method is where the reducer does most of the data processing
  - The keys are sorted
  - For one key you will get an array containing all the values for that key. Using this array you can perform any type of aggregation and processing
  - It is important to remember that a key and all it's values will never be split across more then one reducer
  - If one reducer takes significantly longer to finish than the rest. This is typically the result of this reducer processing a key that has significantly more values than the rest. Must make sure data is partition between the reducers as evenly as possible while still aggregating the values correctly

Reducer phase components

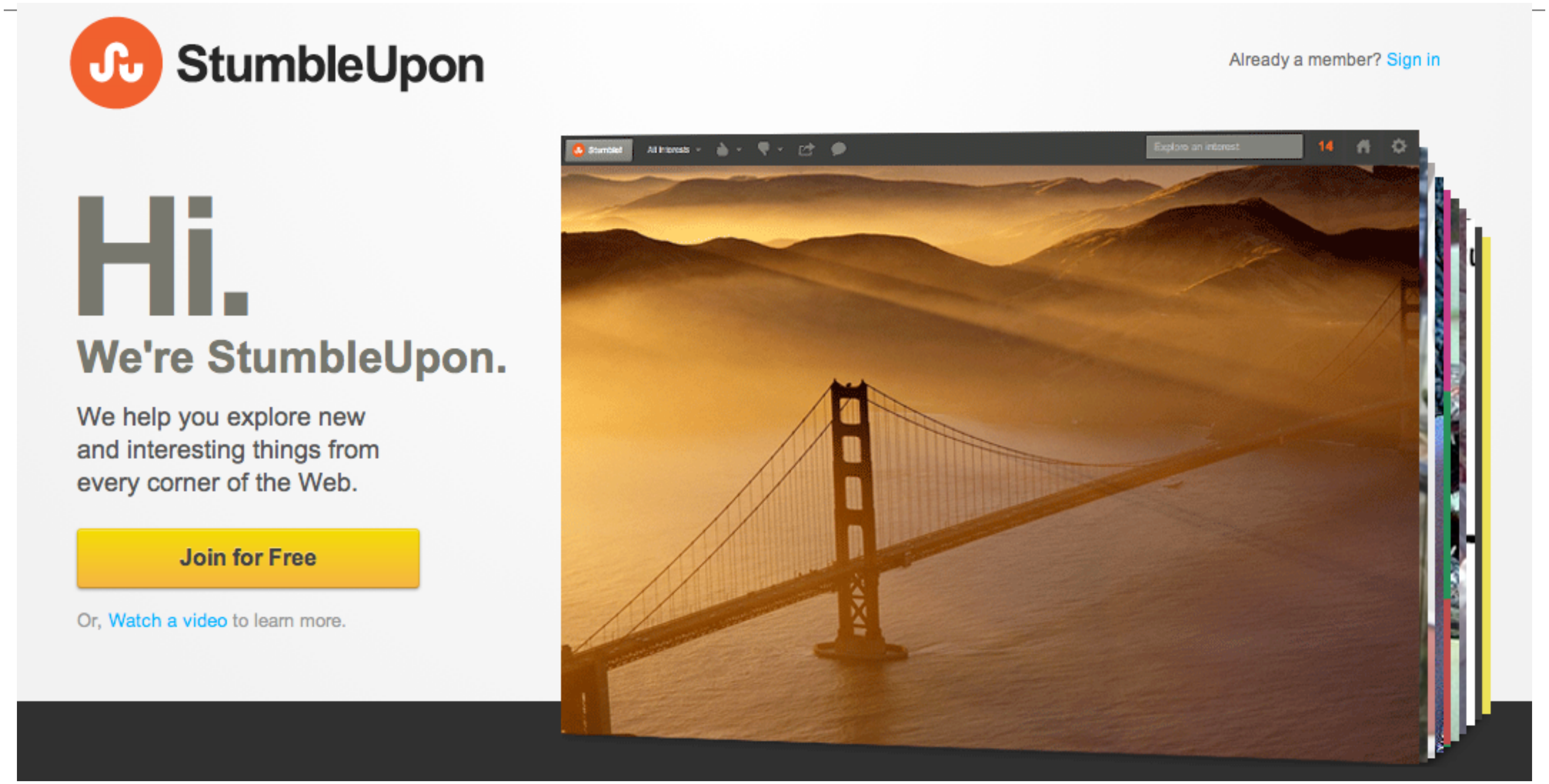
- **outputFormat:** Just like the inputFormat class handled the access to input data, the outputFormat call is responsible for formatting the output data and writing it out, typically to HDFS.

- With the outputFormat class, a single reducer will always write a single file, so on HDFS you will get a file per reducer. The files will be named something like part-r-00000 with the numbers ascending as the task number of the reducers ascend.


Node 1
Files loaded from local HDFS stores
file
file
InputFormat
Split
Split
Split
RecordReaders:
RR
RR
RR
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Partitioner
(sort)
reduce
Final (k, v) pairs
OutputFormat
Writeback to local HDFS store
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes
Node 2
Files loaded from local HDFS stores
InputFormat
Split
Split
Split
file
file
RR
RR
RR
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Partitioner
(sort)
reduce
Final (k, v) pairs
OutputFormat
Writeback to local HDFS store

Examples

Map-reduce and hadoop web application context



StumbleUpon

Already a member? Sign in

Hi.

We're StumbleUpon.

We help you explore new and interesting things from every corner of the Web.

Join for Free

Or, Watch a video to learn more.

Recommending web sites

- Present web sites suggested by other users
- Each user rates websites by “like” or “dislike”
- Stumbleupon uses ratings to classify users and predict new websites to users

Data model

- Contact data: email, img, name, url
- Statistics: last login, signup date
- Attributes: remote login credentials, to authenticate to a third party service
- Permissions: For access to site features and data

Stumbles per user, URL

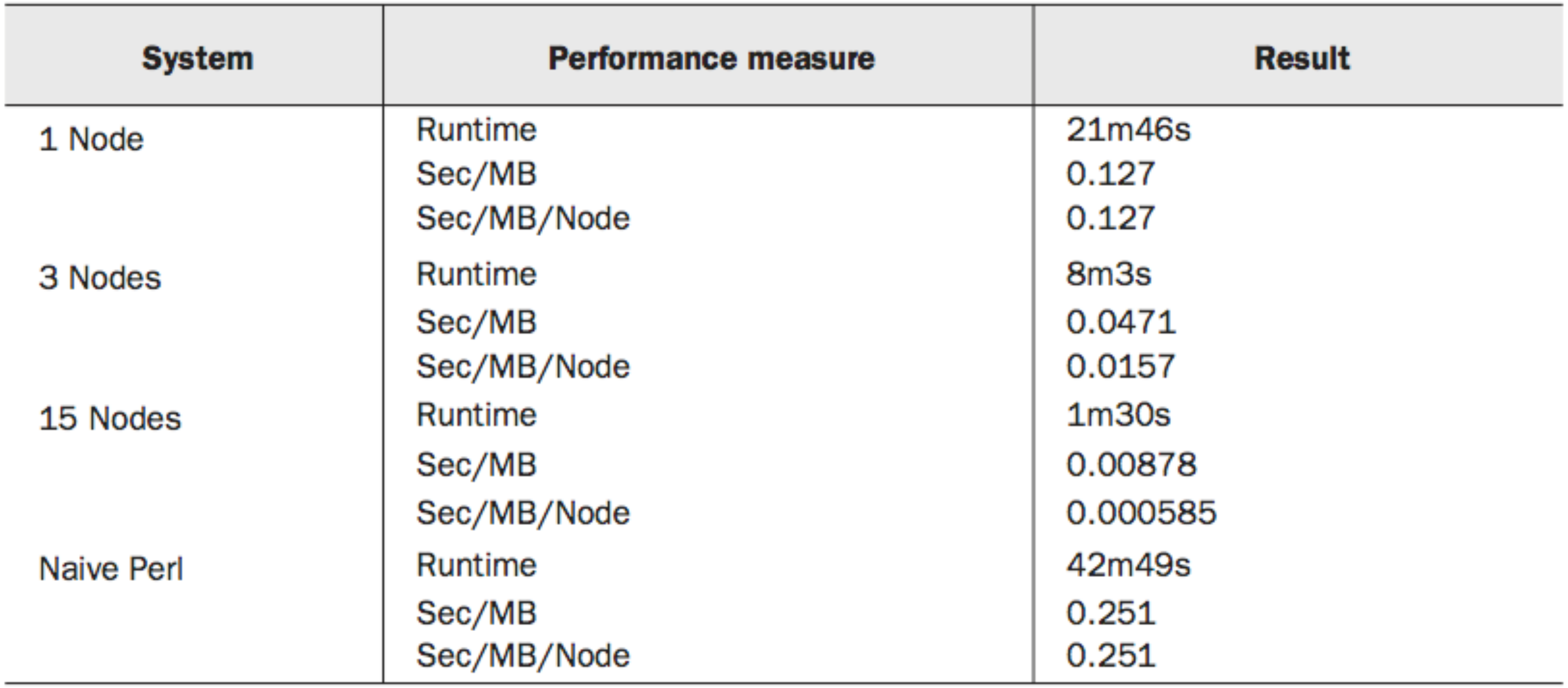
```
public void map(ImmutableBytesWritable key,
              RowResult value,
              OutputCollector<Text, Text> output,
              Reporter reporter) throws IOException {
     byte [] row = value.getRow();
     int userid = StumbleUtils.UserIndex.getUserId(row);
     int urlid = StumbleUtils.UserIndex.getUrlId(row);
     Text one = new Text("1");
     output.collect(new Text("U:" +
                   Integer.toString(userid)), one);
     output.collect(new Text("Url:" +
                   Integer.toString(urlid)), one);

  }
```

Reduce

```
public void reduce(Text key,
           Iterator<Text> values,
           OutputCollector<Text, Text> output,
           Reporter reporter) throws IOException {
   int count = 0;
   while (values.hasNext()) {
        values.next();
        count++;
   }
   output.collect(key, new
                  Text(Integer.toString(count)));
}
```

Times



System	Performance measure	Result
1 Node	Runtime	21m46s
	Sec/MB	0.127
	Sec/MB/Node	0.127
3 Nodes	Runtime	8m3s
	Sec/MB	0.0471
	Sec/MB/Node	0.0157
15 Nodes	Runtime	1m30s
	Sec/MB	0.00878
	Sec/MB/Node	0.000585
Naive Perl	Runtime	42m49s
	Sec/MB	0.251
	Sec/MB/Node	0.251

When to use MapReduce

- **MapReduce** is a very low level framework. The developer is responsible for very minute details of operation and setup. Because of that MapReduce code typically has more bugs and higher costs of maintenance.
- However, there is a subset of problems, such as file compaction, distributed file-copy or row-level data validation that translate to MapReduce quite naturally.
- At other times, code written in MapReduce can take advantage of properties of the input data to improve performance.
    - For example, if we know the input files are sorted, we can use Map Reduce to optimize merging of data sets