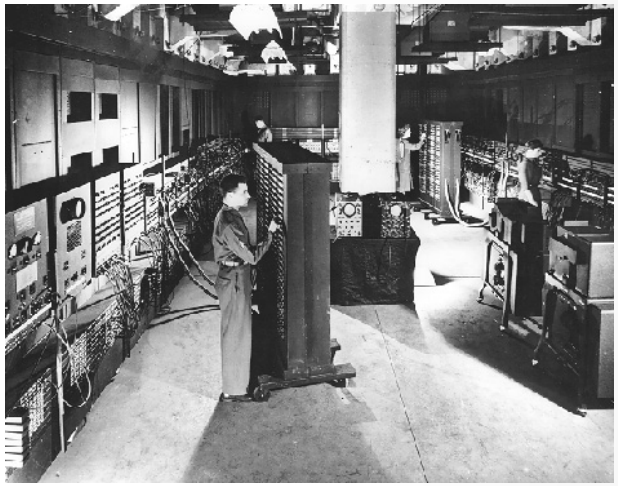
Large Scale Data Mining

Advanced Data Mining

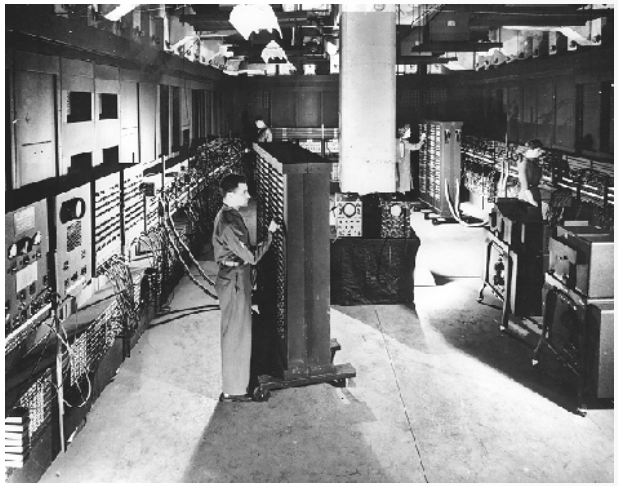
How large is large?

- **It depends…**
- In the **early days (before 80s)**: **1 Kilobyte is big**.
- The first computer ENIAC has **1 KB** of core memory (equivalent memory space).



How large is large?

- **It depends…**
- In the **early days (before 80s)**: **1 Kilobyte is big**.
- The first computer ENIAC has **1 KB** of core memory (equivalent memory space).



- Personal consumer computer Altair is equipped with **4 KB** memory. (1975)

In the 80s

- **1 Megabyte of data is considered as big.**
- Apple's Macintosh came with **128 KB** of memory.

- IBM PC has **16 KB ~ 256 KB** of memory.

In the 90s

- 1Gigabyte is big.
- My first computer has **512 KB** of memory ☺

- When it retired in 1998, its memory has been upgraded to **12 MB** ☺

Picture from the Internet

In the Web-Age

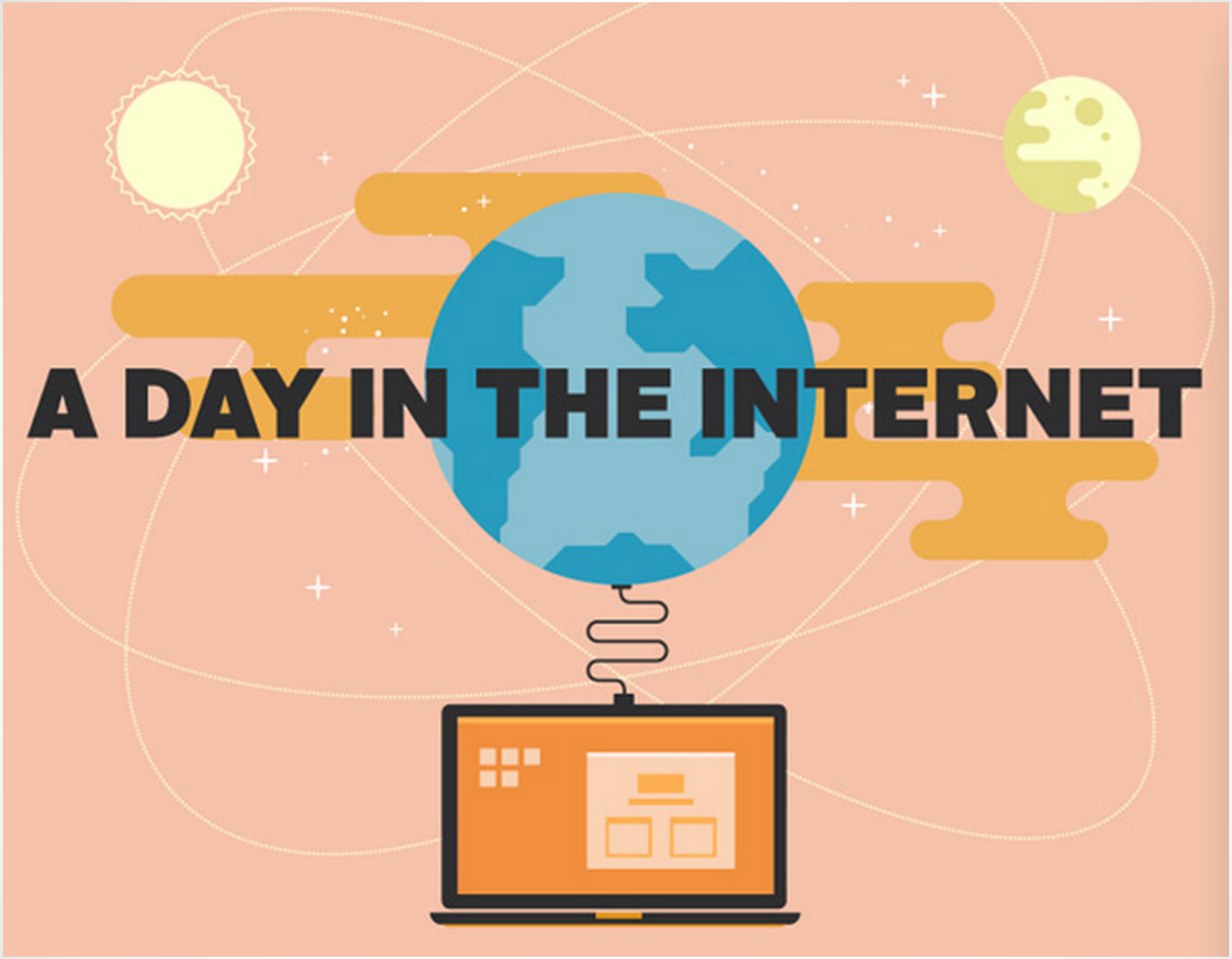
10 Gigabytes?

100 Gigabyte?

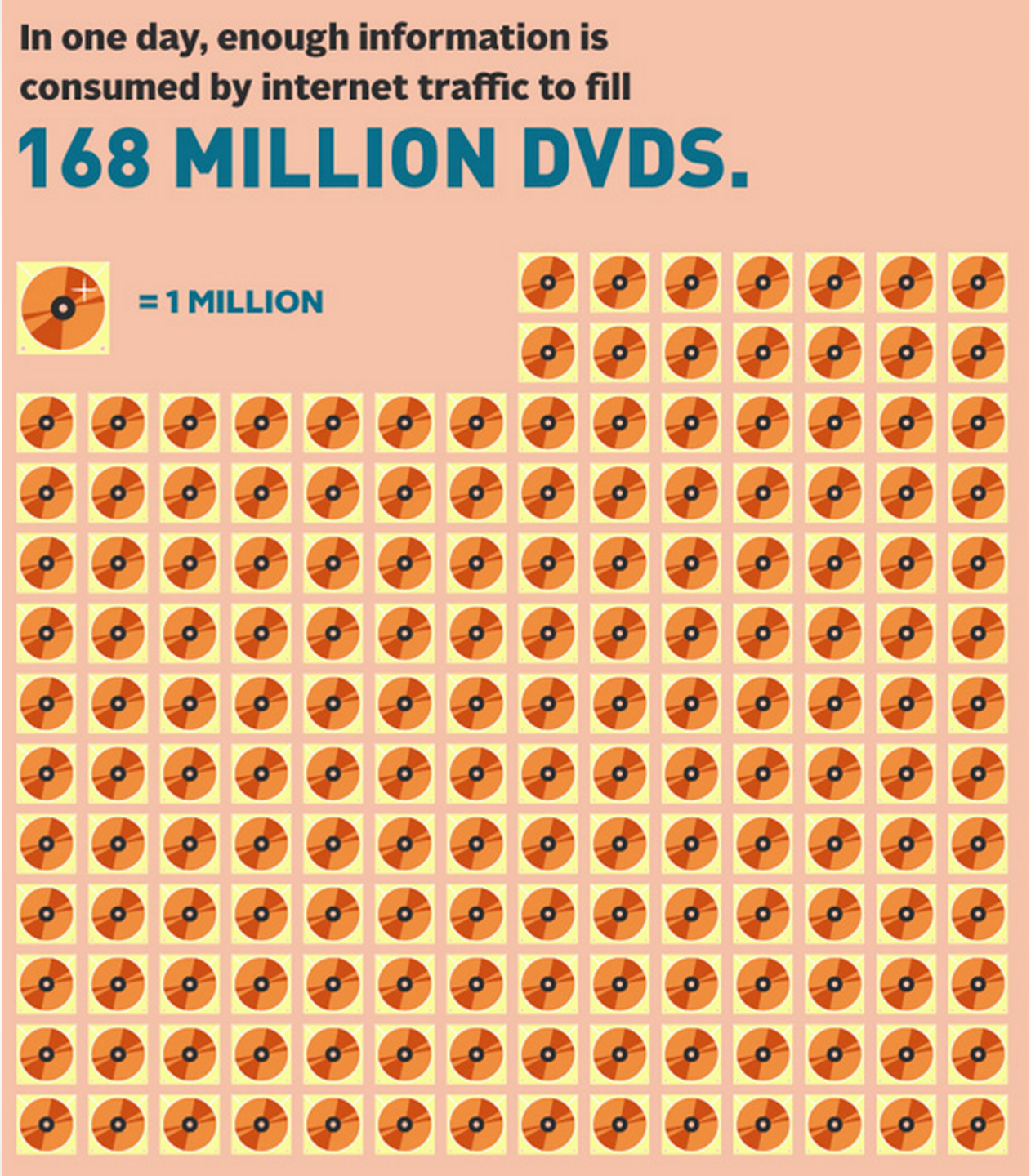
1 Terabyte?

1 Petabyte?

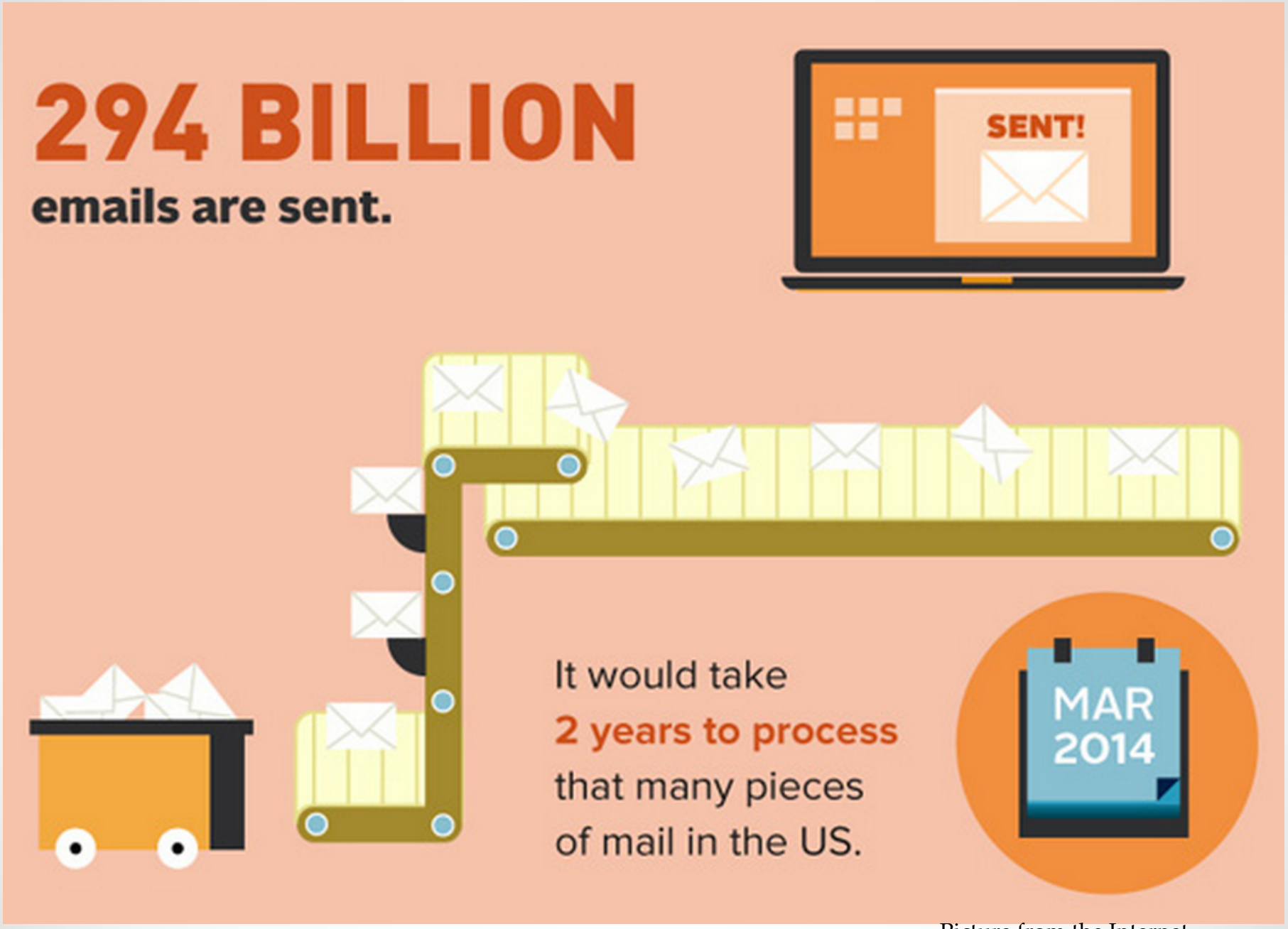
…??



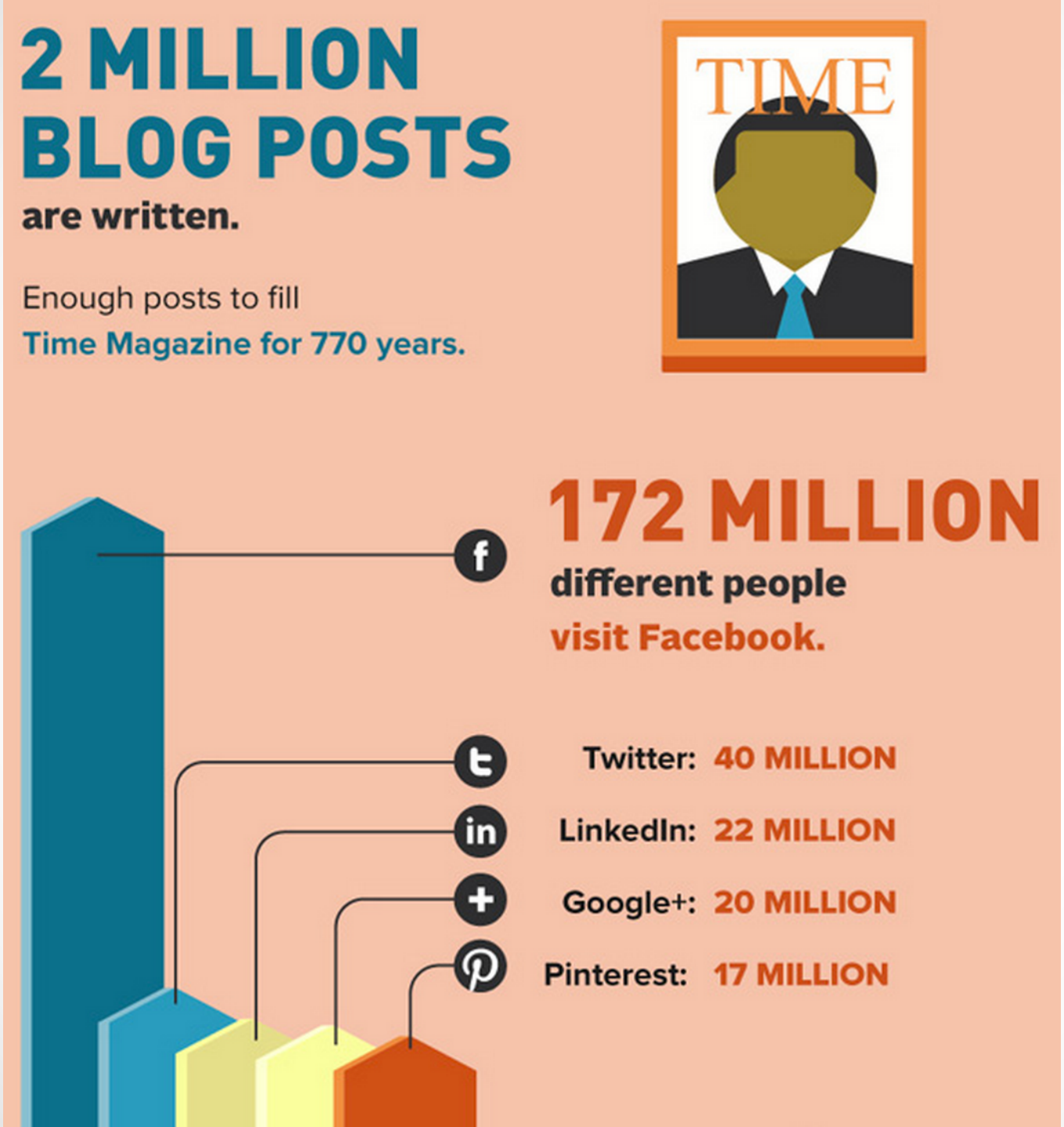
Pictures from MBAonline.com



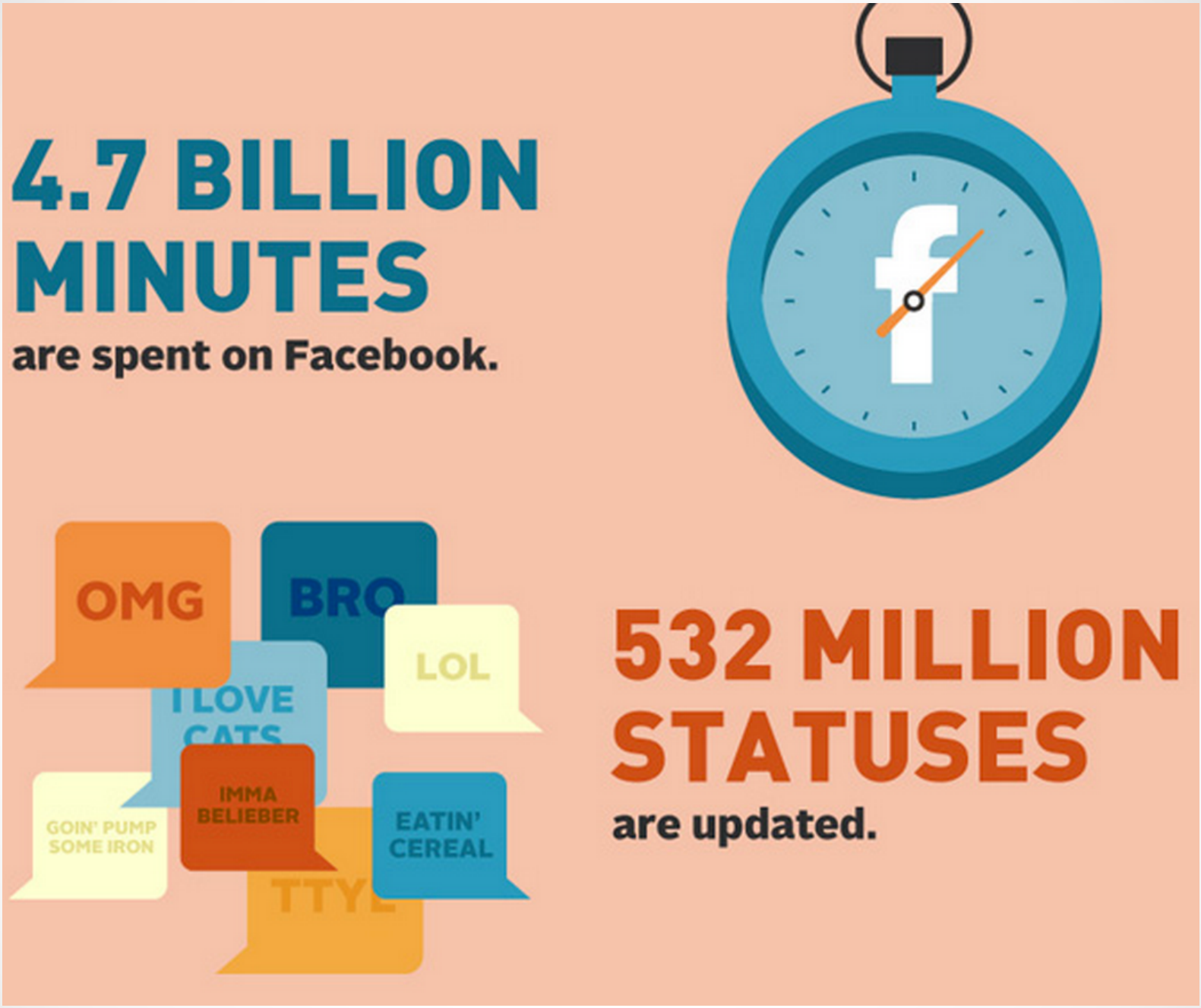
Picture from the Internet



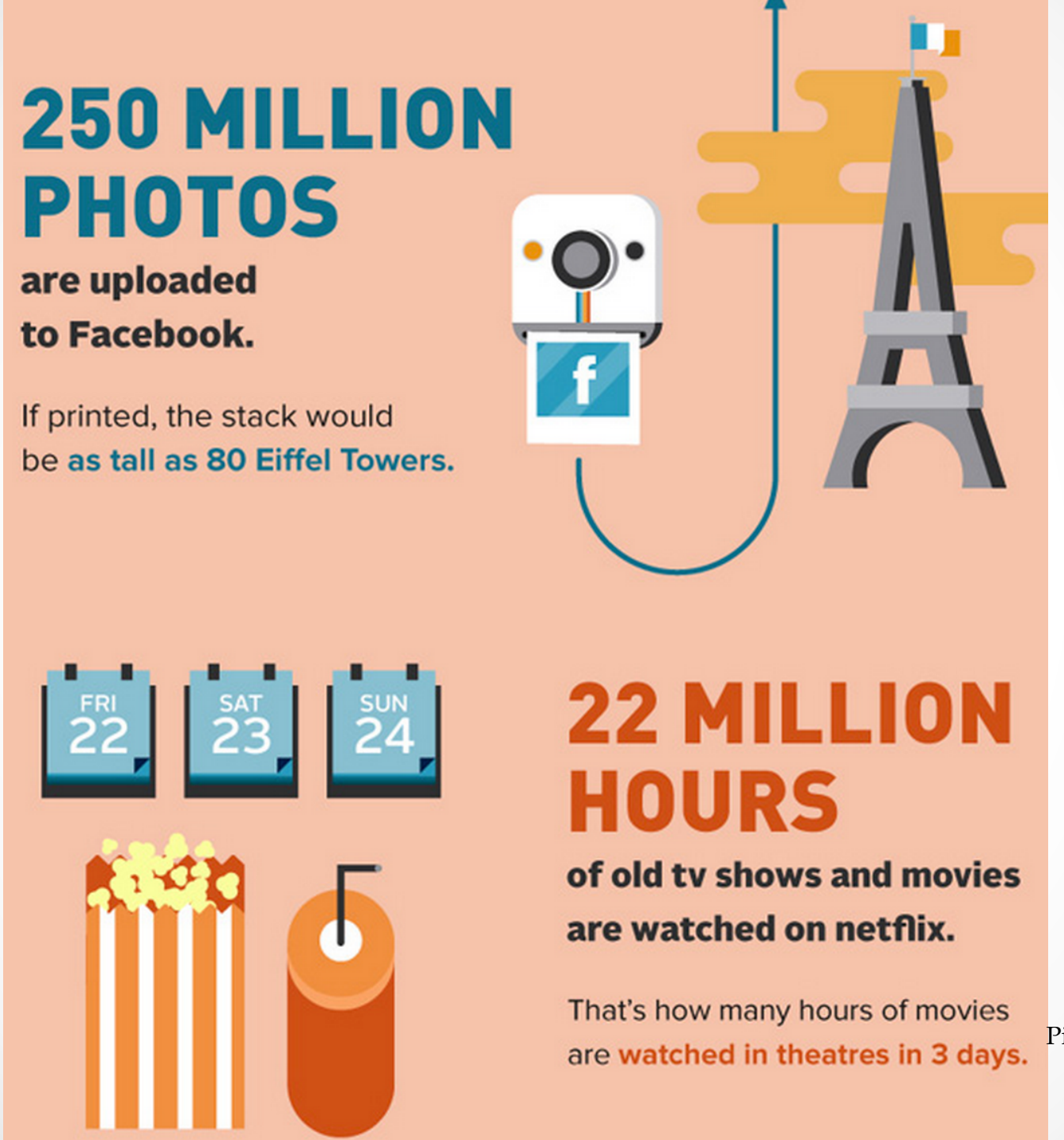
Picture from the Internet



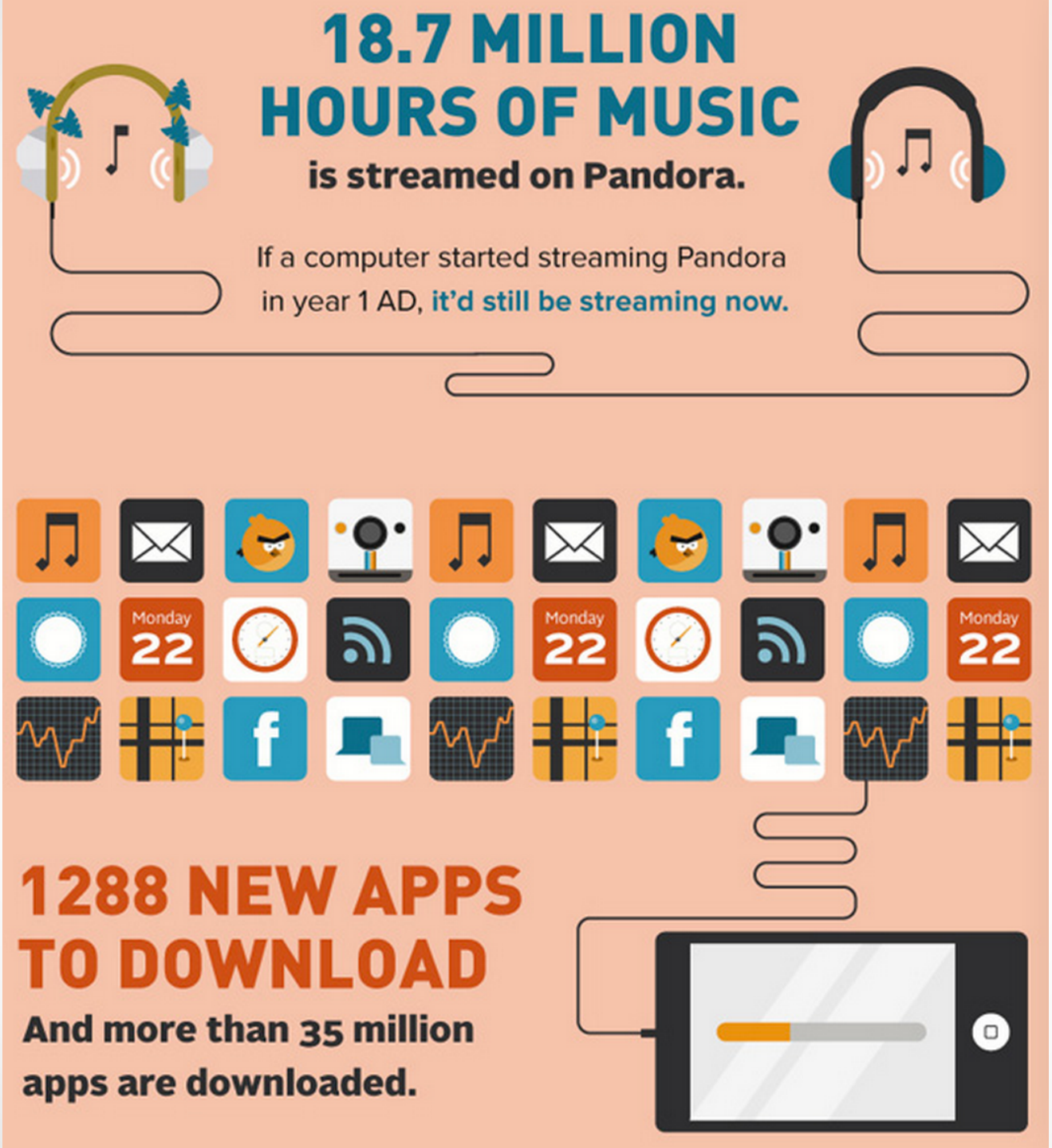
Picture from the Internet



Picture from the Internet



Picture from the Internet



Picture from the Internet

How do we make use of these data?

Large Scale Data Mining

Why not 'classical' data mining?

Data cannot be efficiently processed by one machine

- The algorithms as well as their implementations are mainly single machine version.
- Most of them run in single thread/process.

Time is money!

Out of Memory

- Data cannot be load into memory.

Out of Storage Space

- Data cannot be stored in one machine. No single computer can store petabytes of data.

The Foundation of Large Scale Data Mining

- Infrastructure
  - Machines
  - High speed interconnection
  - Auxiliary devices
- Framework
  - MapReduce
  - Bulk Synchronous Parallel
  - Distributed Real Time Computing Framework
- Algorithm
  - Parallelization
  - Online Learning

Computers. A lot of them!

You probably need.

A small cluster

Computers. A lot of them!

You probably need.

A small cluster

Or a rack

Or a data center

The Auxiliaries

- Network
  - Routers
  - Cables

The Auxiliaries

- Network
  - Routers
  - Cables

- Cooling
  - Air conditioners
  - Water cooling devices

The Auxiliaries

- Network
  - Routers
  - Cables

- Cooling
  - Air conditioners
  - Water cooling devices

- Power Management

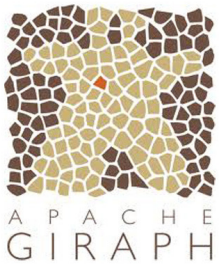
Those cost too much!

- Use (IaaS) cloud service

Computing Framework

- Map-Reduce
  - Hadoop

- Bulk Synchronous Parallel (BSP)
  - Giraph
  - Hama



- In-memory Cluster Computing
  - Spark

- Real Time Computing Framework
  - Storm

Map-Reduce: Algorithmic Perspective

- Map-Reduce is not a new concept!
- Map and Reduce are two commonly used operations.

- **Map**: A ***high order function*** that applies a function (a.k.a. **Transformer**) to each element of the list.

- **Reduce**: A ***high order function*** that applies a combiner function to a list of elements.

Map

- **map** (function, list)

Example:

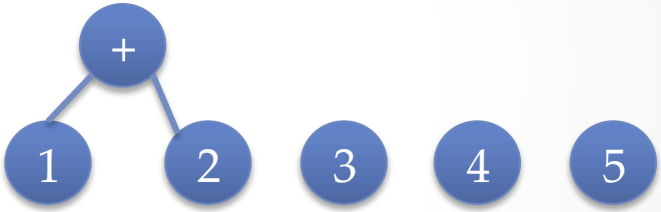
- Convert all the letters to lower case:
  - **map**(**toLower**, "HeLLO, World!") -> "hello, world!"
- Count the number of words of each element
  - **map**(**wordCount**, ["hello world", "good evening", "hi"]) -> [2, 2, 1]
- Inverse the key and value of each element
  - **map**(**inverse**, ["key1": "value1", "key2": "value2"]) -> ["value1": "key1", "value2": "key2"]

Reduce

- **reduce**(function, list)

Example:

- Get the sum of a list of elements
  - **reduce**(**plus**, [1, 2, 3, 4, 5]) -> 15

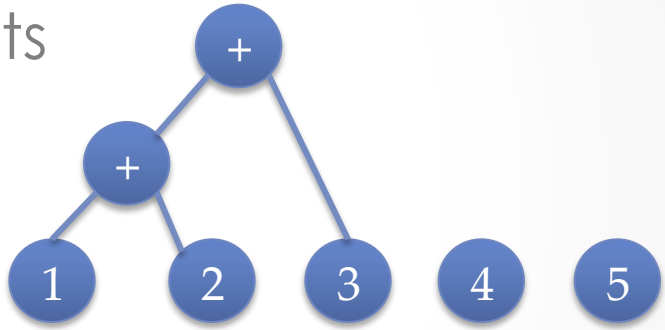


Reduce

- **reduce**(function, list)

Example:

- Get the sum of a list of elements
  - **reduce**(**plus**, [1, 2, 3, 4, 5]) -> 15

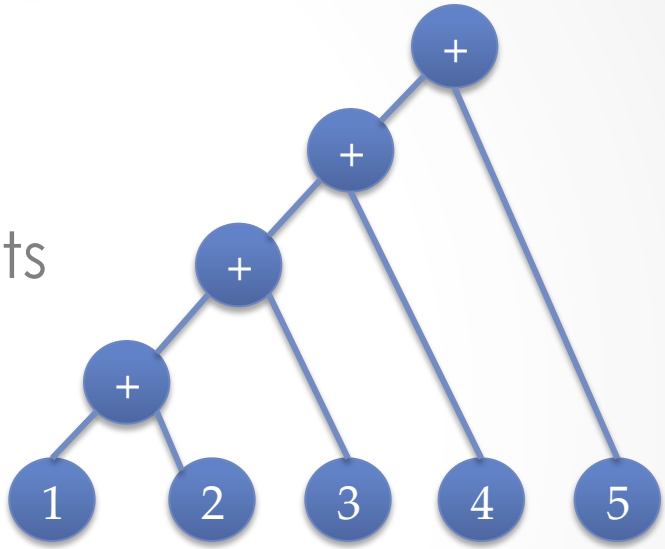


Reduce

- **reduce**(function, list)

Example:

- Get the sum of a list of elements
  - **reduce**(**plus**, [1, 2, 3, 4, 5]) -> 15

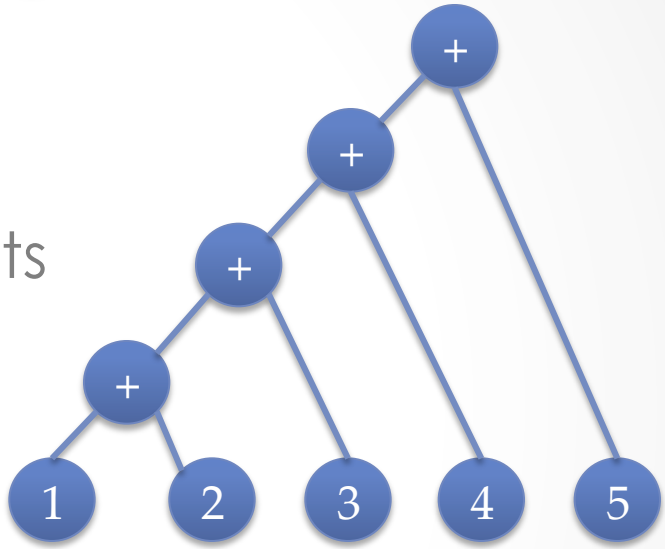


Reduce

- **reduce**(function, list)

Example:

- Get the sum of a list of elements
  - **reduce**(**plus**, [1, 2, 3, 4, 5]) -> 15

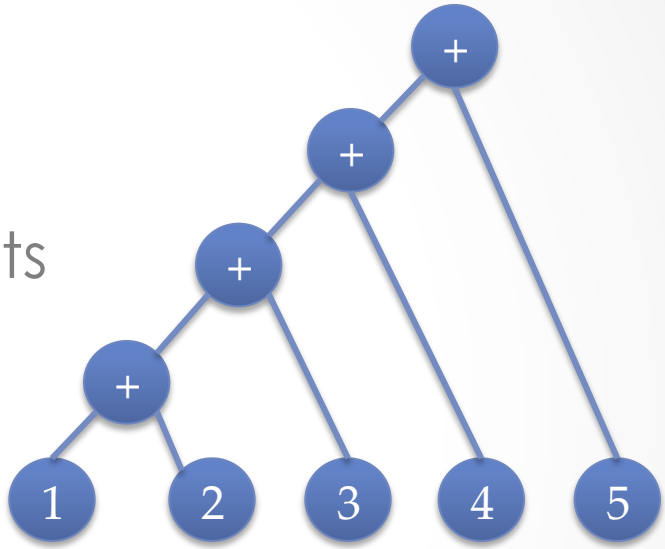


Reduce

- **reduce**(function, list)

Example:

- Get the sum of a list of elements
  - **reduce**(**plus**, [1, 2, 3, 4, 5]) -> 15



- Aggregate the keys to but inverse index
  - **reduce**(**aggregate**, [“key1”: “value1”, “key1”: “value2”, “key1”: “value3”, “key2”: “value3”] -> [“key1”: [“value1”, “value2”, “value3”], “key2”: “value3”]

Map-Reduce: Parallel Computing Perspective

Distributed computing framework running on top of commodity computers.

First time draw attention.

- Google's distributed computing framework: MapReduce: Simplified Data Processing on Large Clusters, OSDI 2004.

Get popular.

- Apache Hadoop. Doug Cutting and Mike Cafarella et al. 2005.
  - 1.2.X Hadoop 1.
  - 2.2.X Hadoop 2 (YARN).

Why Map-Reduce?

- The size of data has far beyond the processing capability of the single machine.

Example: A small search engine

- 20+ billion billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35MB/sec from disk would use 4+ month to just read all the data.
- ~1000 hard drives to store the data.
- Much more time to do something to the data…

Why Map-Reduce? (Cont)

- Super-computer
  - Too expensive
  - Requires non-trivial extra time to train the programmer

- Traditional parallel computing framework
  - Requires knowledge of low-level details: communication, fault-tolerance
  - Difficult to maintain
  - Programmer would be distracted

Fundamentals of Map-Reduce

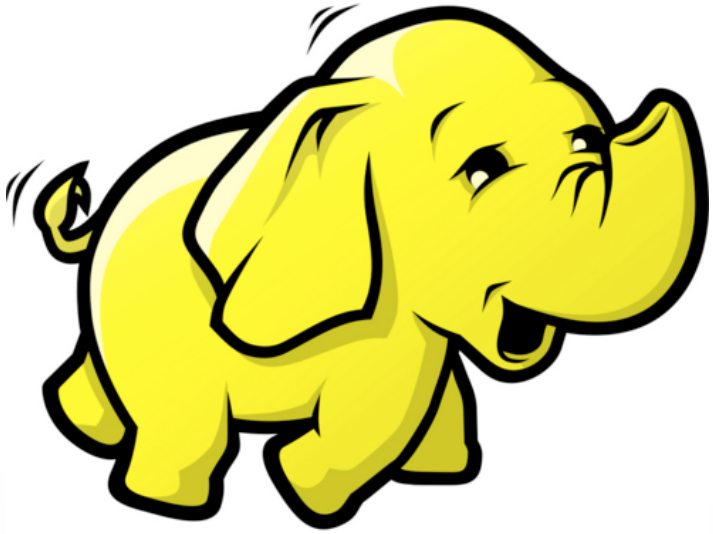
Two Main Steps:

- **Map**: Extract something you care about.
  - Map<Key1, Value1> -> Key2, Value2

- **Reduce**: Aggregate, Summarize, Filter, Or Transform
  - Reduce<Key1, List(Value1)> -> Key2, Value2

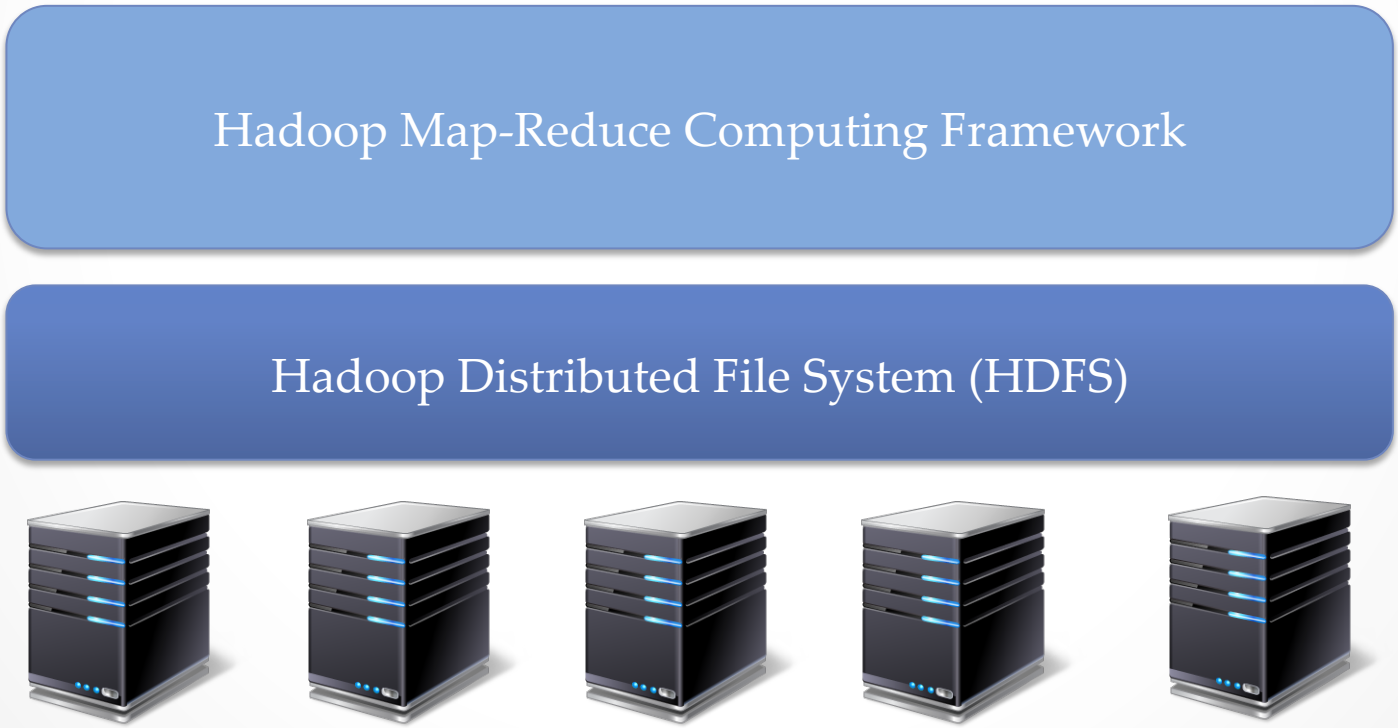
Apache Hadoop

- A Java implementation of Map-Reduce.
- Initialized by Yahoo!
- An Apache open source project. Initially a sub-project of Apache Nutch, a web crawler system.

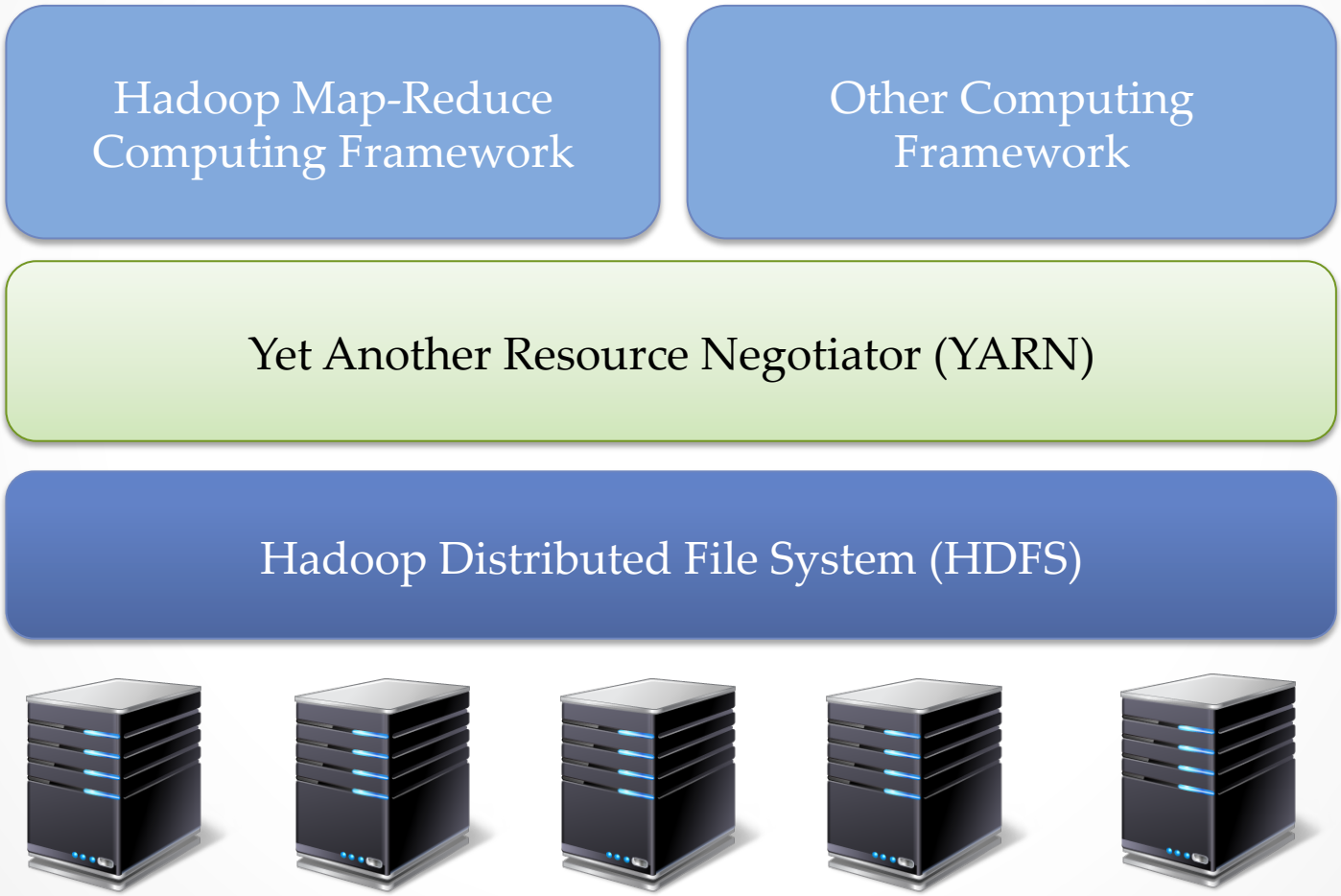


Architecture of Hadoop 1

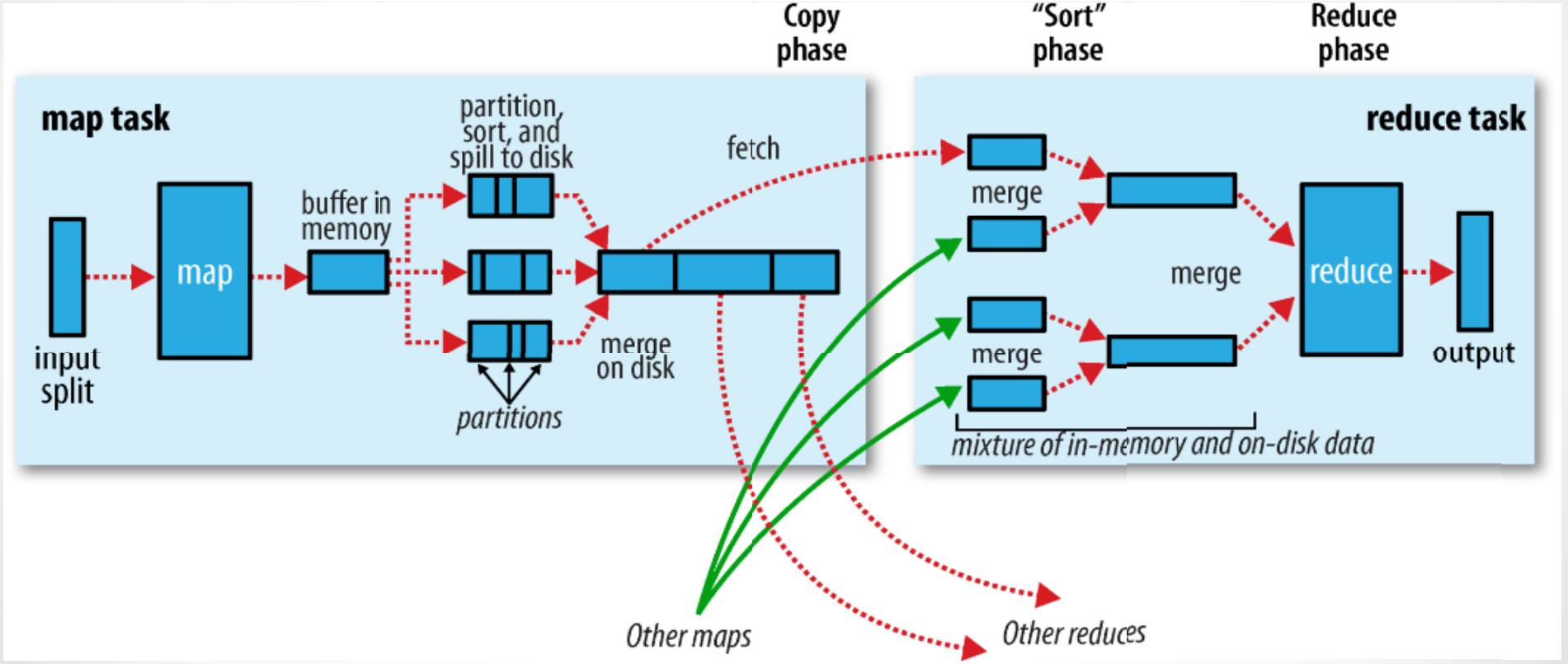
- Hadoop is not only Map-Reduce.



Architecture of Hadoop 2



Procedures in Hadoop



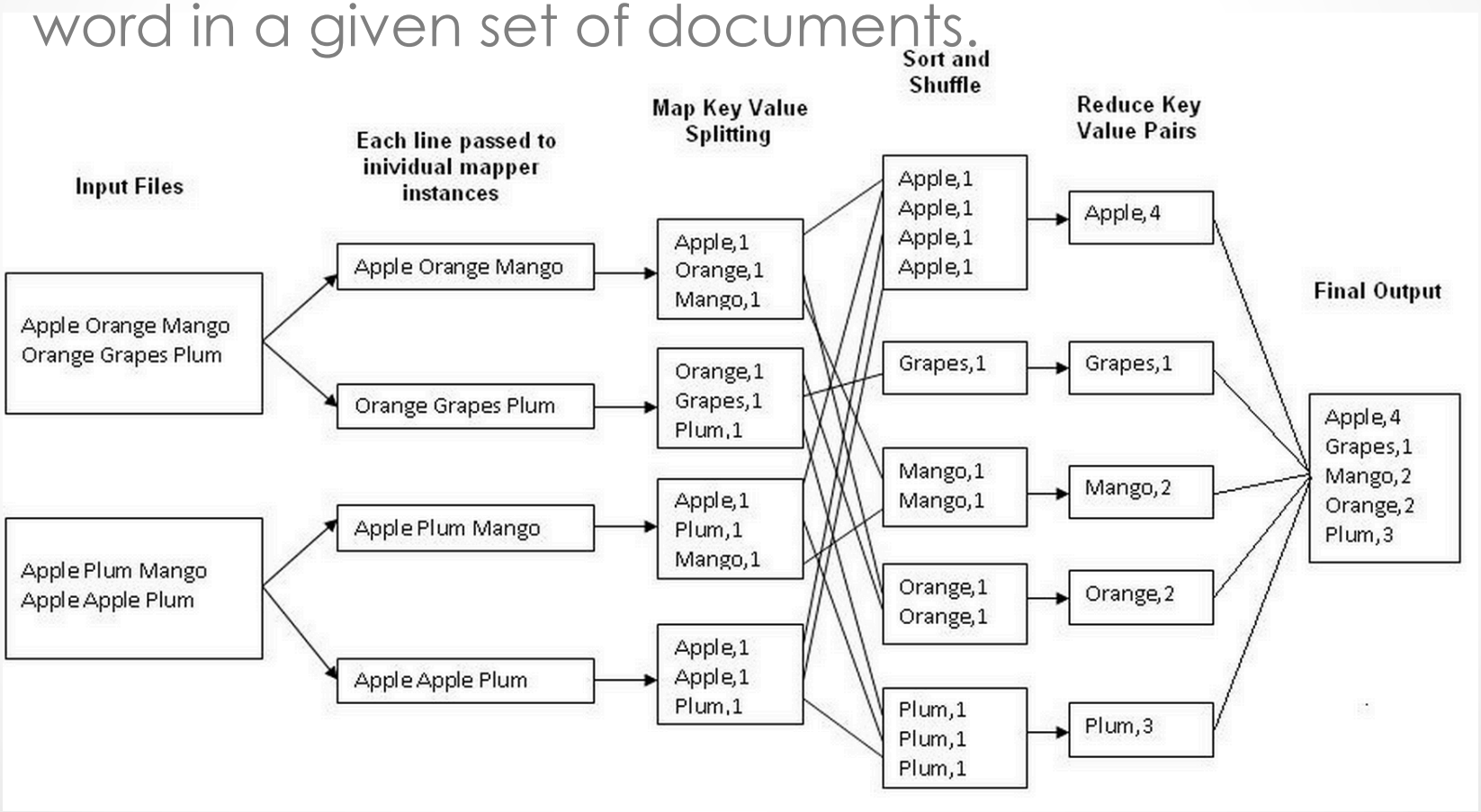
from Hadoop: The Definitive Guide

Main API

- Mapper:
  - setup(Context context)
  - map(Key1 key, Value1 value, Context context)
  - cleanup(Context context)
- Reducer:
  - setup(Context context)
  - reduce(Key3 key, Iterable<Value3> values, Context context)
- WritableComparable
  - readFields(DataInput in)
  - Write(DataOutput out)
  - compareTo
- Partitioner, InputFormat, InputSplit

"Hello World" in Hadoop

- Wordcount: count the number of each distinct word in a given set of documents.



Data Mining Algorithms on Top of Mapreduce

What data mining algorithms are suitable to be implemented in MapReduce style?

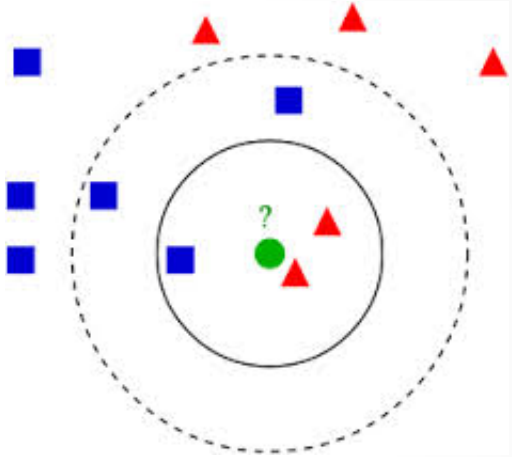
Data Mining Algorithms on Top of Mapreduce

What data mining algorithms are suitable to be implemented in MapReduce style?

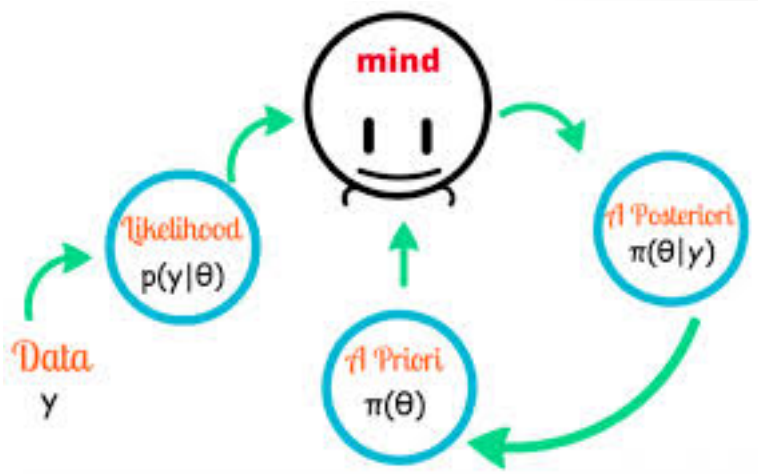
- Algorithms can be finished in a constant number of steps
- Algorithms that can run in pure parallel (no information exchange between partition)

Algorithms Perfect Fit to MapReduce

- K-Nearest-Neighbor
  - Lazy classification

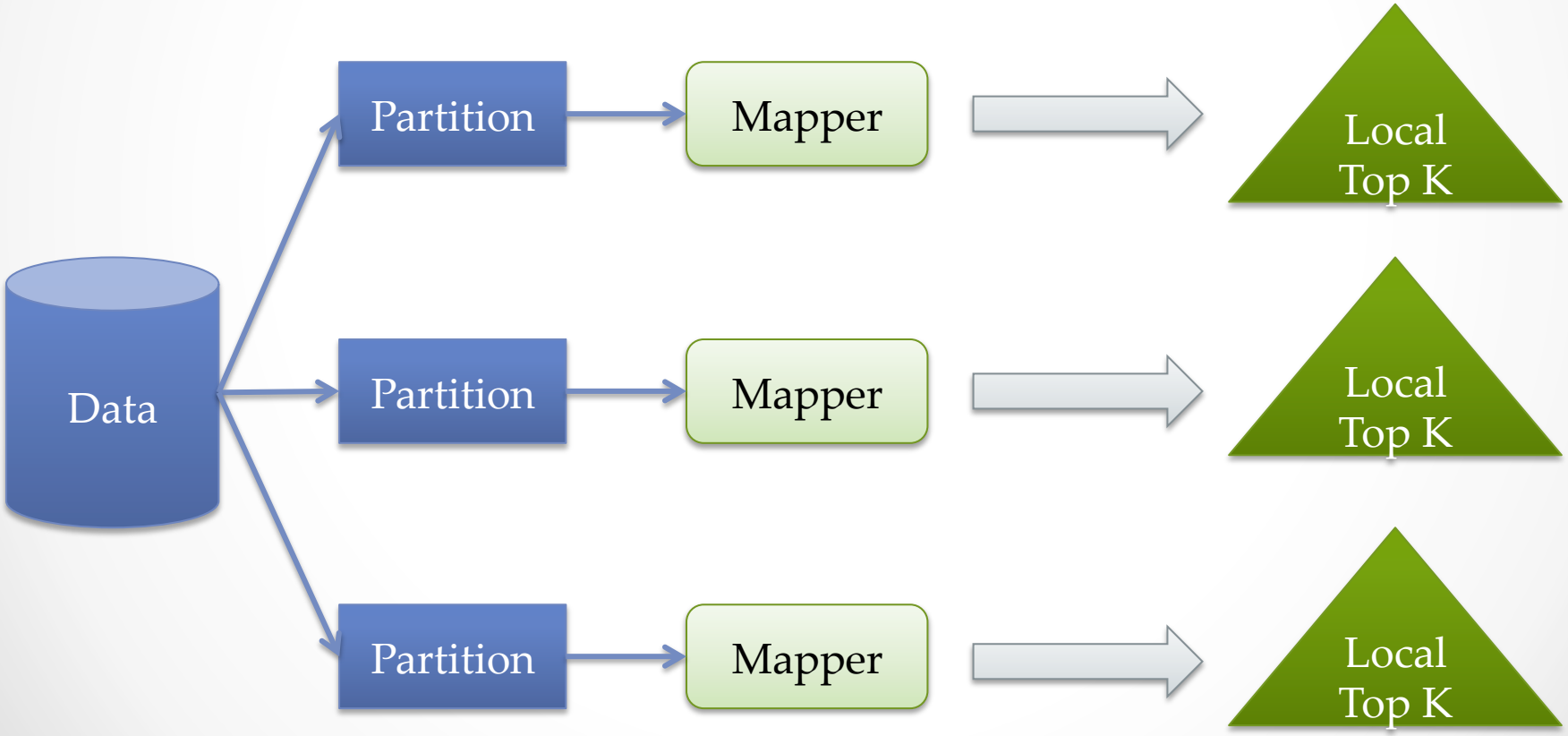


- Naïve Bayes
  - Generative model
  - Classification



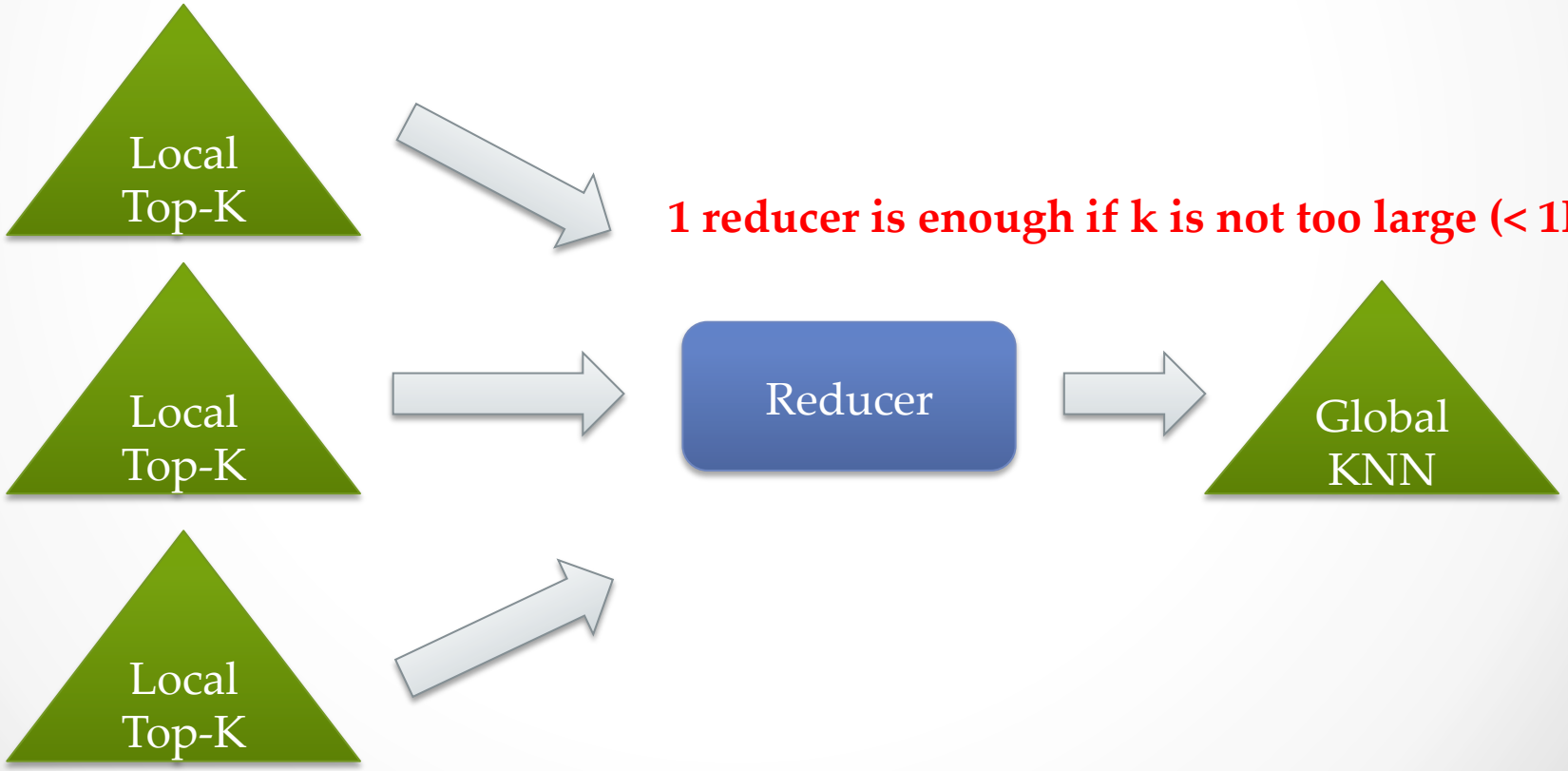
KNN Implementation

- **Mapper**: Find the k-Neareat neighbors locally.



KNN Implementation

- **Reducer**: Get the global k-Nearest Neighbors from all the locally k-Nearest Neighbors.



Sample Code

Mapper:

```
map(LongWritable key, VectorWritable vec, Context context) {
  distance = dist(vec, targetVec);
  localHeap.add(new Pair(vec, distance);
  if (localHeap.size() > k) {
    localHeap.pop();
  }
}
cleanup(Context context) {
  output all elements in localHeap (key is a constant, value is the pair)
}
```

Reducer:

```
reduce(LongWritable key, Iterable<Pair> vecIterable) {
  for each pair {
    globalHeap.add(pair);
    if (globalHeap.size() > k) {
      globalHeap.pop();
    }
    output the majority of label
  }
}
```

Naïve Bayes Training

- Example: Text Classification Scenario
- Given a set of training documents D = $\{d_i, c_j\}$, build a naïve bayes classifier.
- Prerequisite: Feature extraction.
  - Use bag-of-word to represent a document with count vector.
  - Each document can be represented as $d_i = [\#w_1, \#w_2, \ldots, \#w_n]$.

This is just an example, not the optimal solution!

Naïve Bayes Training

- The formula:

$$p(C_j|F_1, ..., F_n) = \frac{p(F_1, ..., F_n|C_j)p(C_j)}{p(F_1, ..., F_n)}$$

$$p(C_j|F_1, ..., F_n) \propto p(F_1, ..., F_n|C_j)p(C_j)$$

$$p(C_j|F_1, ..., F_n) \propto p(C_j)\Pi_i p(F_i|C_j)$$

Easy to get underflow

$$\log p(C_j|F_1, ..., F_n) \propto p(C_j) \sum_i \log p(F_i|C_j)$$

$$\log p(C_j|F_1, ..., F_n) \propto \frac{\#C_j}{|D|} \sum_i \log \frac{\#(F_i|C_j)}{\#C_j}$$

Naïve Bayes Training

- The formula:

$$p(C_j|F_1, ..., F_n) = \frac{p(F_1, ..., F_n|C_j)p(C_j)}{p(F_1, ..., F_n)}$$

$$p(C_j|F_1, ..., F_n) \propto p(F_1, ..., F_n|C_j)p(C_j)$$

$$p(C_j|F_1, ..., F_n) \propto p(C_j)\Pi_i p(F_i|C_j)$$

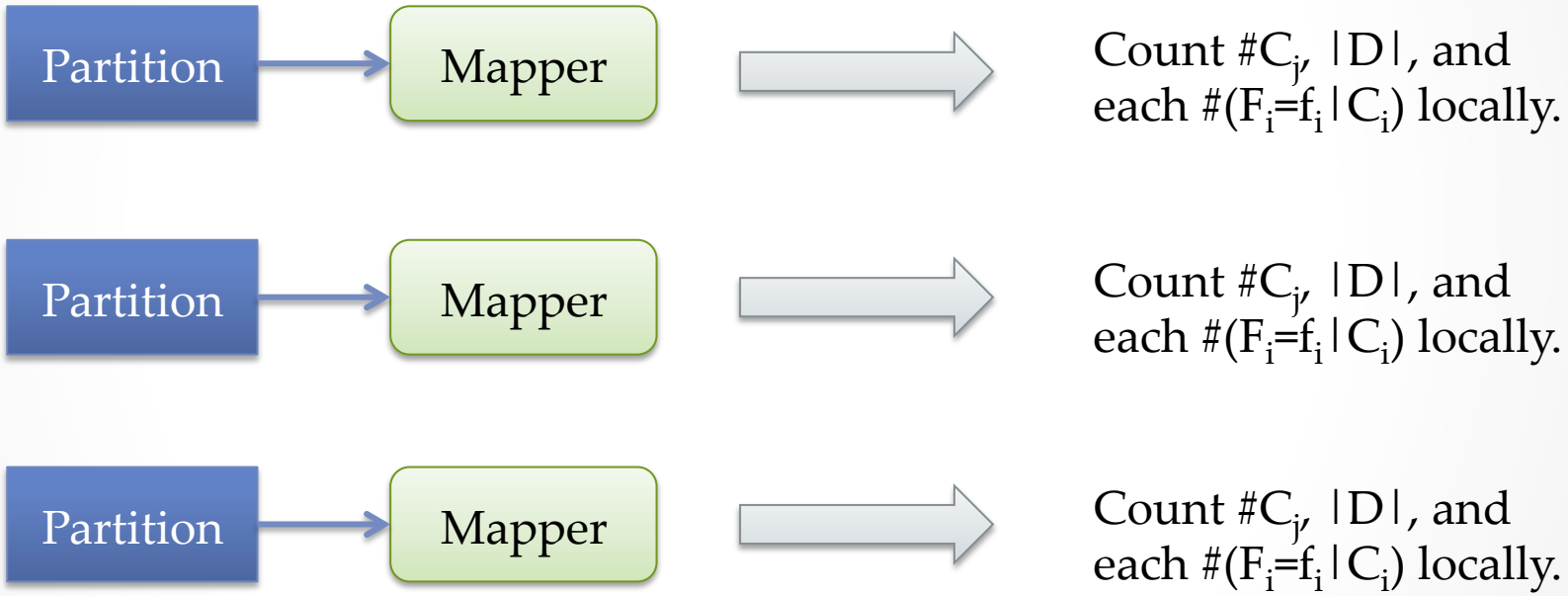
Easy to get underflow

$$\log p(C_j|F_1, ..., F_n) \propto p(C_j) \sum_i \log p(F_i|C_j)$$

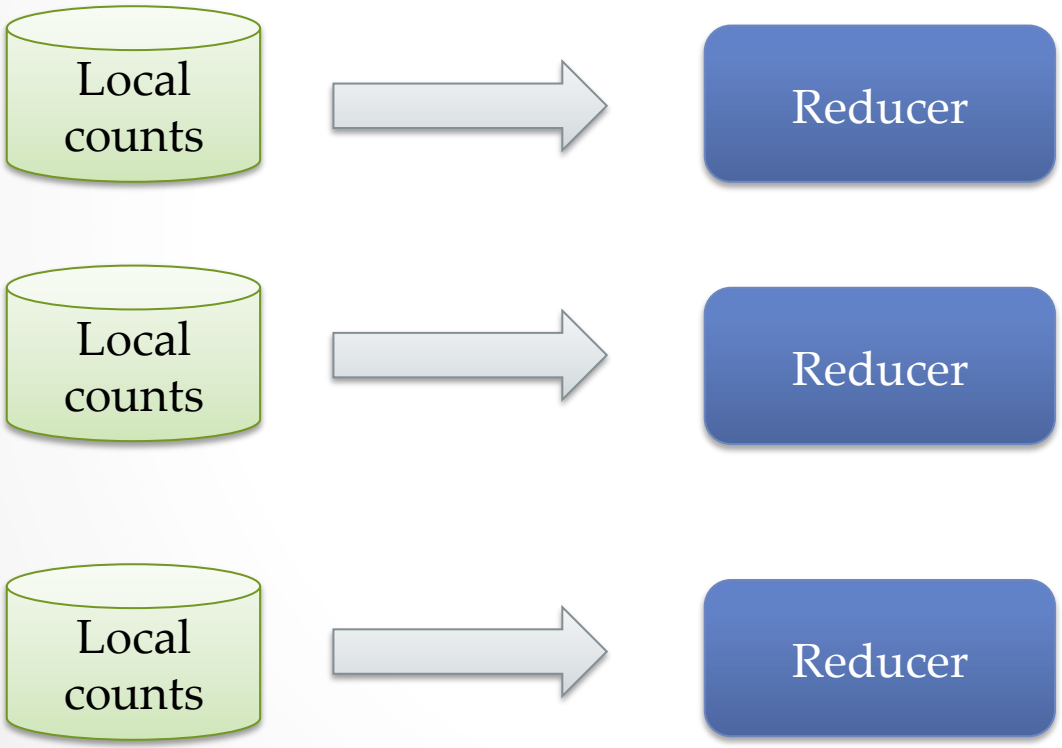
$$\log p(C_j|F_1, ..., F_n) \propto \frac{\#C_j}{|D|} \sum_i \log \frac{\#(F_i|C_j)}{\#C_j}$$

- Things need to be calculated:
  - $p(C_j)$: needs $\#C_j$ and $|D|$.
  - $p(F_i = f_i | C_j)$: needs $\#(F_i=f_i | C_j)$ and $\#C_j$.

Naïve Bayes Implementation



Naïve Bayes Implementation



Each Reducer counts $\#C_j$, $|D|$, and each $\#(F_i=f_i|C_i)$ globally.

Sample Code

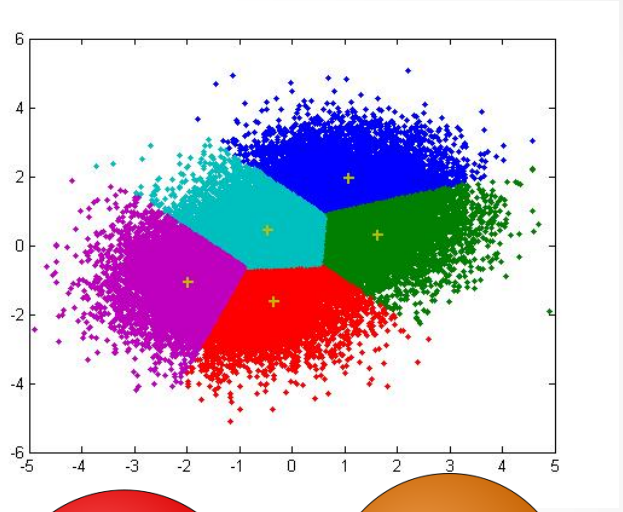
- 1 Job for training
  - **Mapper**: count the local #C_j, |D|, and #(F_i|C_j)
  - **Reducer**: get the global #C_j, |D|, and #(F_i|C_j).

- Classification:
  - Load the probabilities from file into memory. Space costs $O(\sum_i |F_i||C| + |C|)$
  - Given a test document with feature vector w, estimate the probability of belongings to each class.

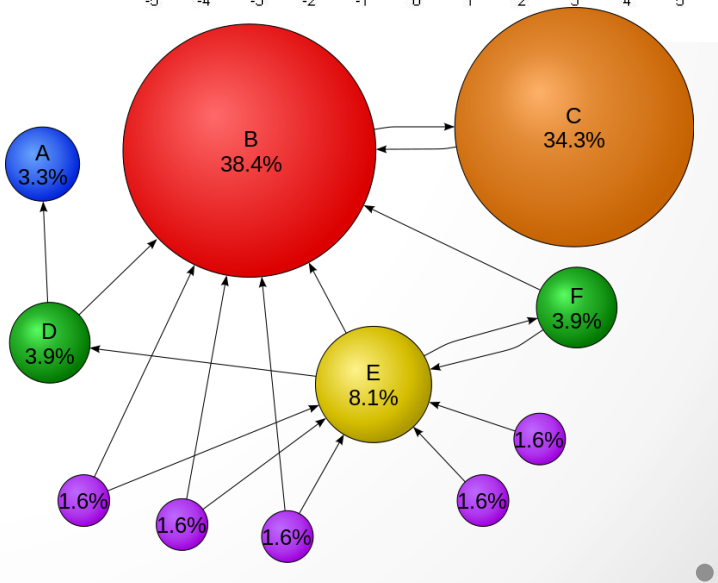
$$C_j = argmax_{C_j} \log p(C_j|F_1, ..., F_n) \propto \frac{\#C_j}{|D|} \log \sum_{i \in w} \frac{\#(F_i|C_j)}{C_j}$$

OK but not efficient

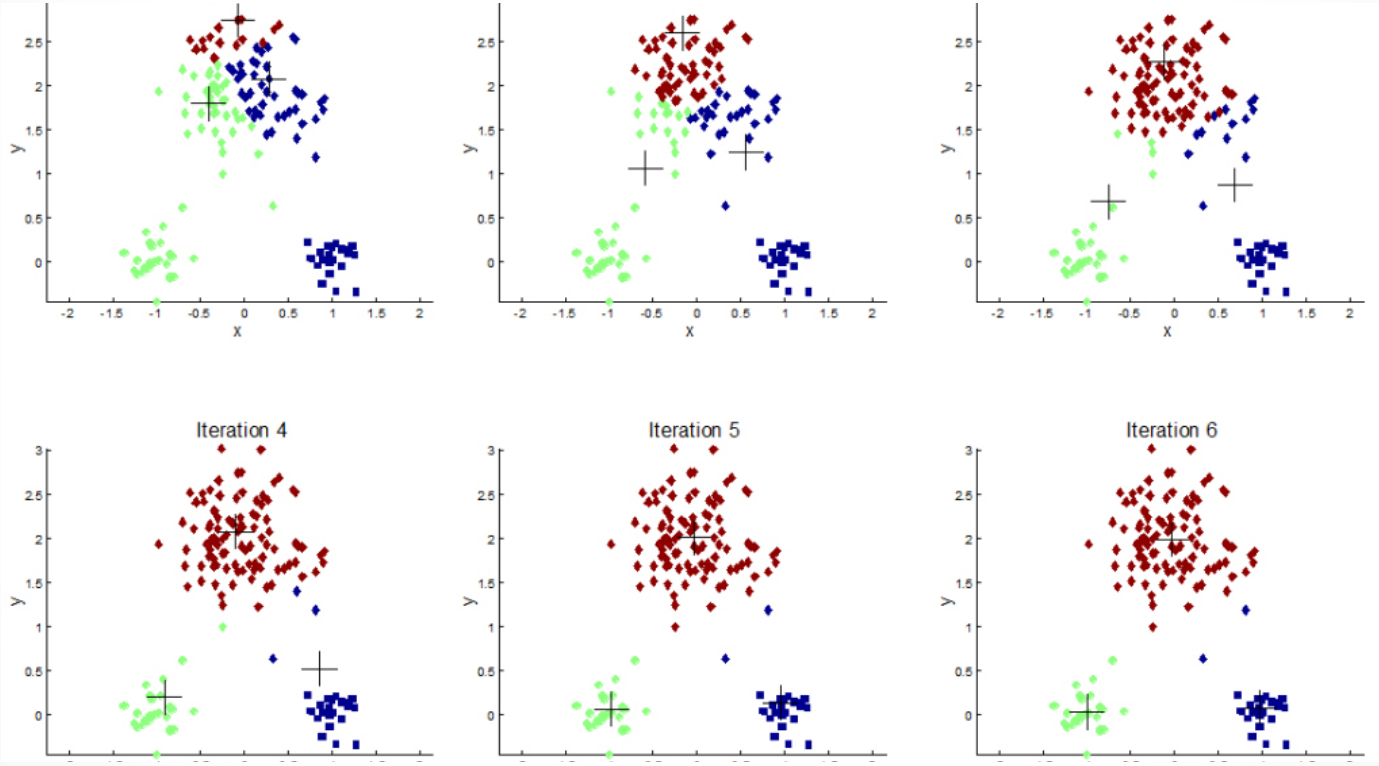
- Kmeans
  - Iterative algorithm
  - Clustering a set of instance into k clusters



- Pagerank
  - Graph algorithms
  - Score the entities by their in-links



Kmeans Revisited



Copy from Andrei Pandre's blog

Kmeans Implementation

Pseudocode: (Naïve solution)

Initialize Clusters
Load cluster information into distributed cache
do
    Assign points to clusters
    Write intermediate data back
    Read intermediate data
    Recalculate cluster center
while not converged

Cluster Assignment Job

Center Recalculation Job

Cluster Assignment Job

- **Driver**: Randomly initialize cluster centers
- **Mapper**:

```
setup(Context context) {
  read cluster centers into memory.
}
map(LongWritable key, VectorWritable vec, Context context) {
  for each center {
    calculate the distance between vec and the center.
  }
  context.write(nearestCenterId, vec);
}
```

- **Reducer**: (Identity Reducer)

```
map(LongWritable centerId, Iterable<VectorWritable> list, Context context) {
  for each element in list {
    context.write(centerId, vec);
  }
}
```

Center Recalculation Job

- **Driver**; Read assignment data
- **Mapper**: (Identity Mapper)

```
map(LongWritable centerId, VectorWritable vec, Context context) {
  context.write(centerId, context);
}
```

- **Reducer**:

```
reduce(LongWritable centerId, Iterable<VectorWritable> list, Context context) {
  Initialize an all-zero vector newCenter
  int count = 0;
  for each vector in list {
    ++count;
    newCenter += vector;
  }
  newCenter /= count;
}
```

Center Recalculation Job

- **Driver**; Read assignment data
- **Mapper**: (Identity Mapper)

```
map(LongWritable centerId, VectorWritable vec, Context context) {
  context.write(centerId, context);
}
```

- **Reducer**:

This Reducer is not scalable

```
reduce(LongWritable centerId, Iterable<VectorWritable> list, Context context) {
  Initialize an all-zero vector newCenter
  int count = 0;
  for each vector in list {
    ++count;
    newCenter += vector;
  }
  newCenter /= count;
}
```

How to deal with overflow?

PageRank Algorithm

- The PageRank score of page A:

$$PR(A) = \frac{1-d}{N} + d \sum_{i \in inlink} \frac{PR(i)}{L(i)}$$

N: number of pages
d: damping factor
L(i): # outlink of page i

- Data format:
  - PageId, <List of PageIds of outlink>
- Steps:
  - Initialize PageRank score of all pages to the same value.
  - **do**
    - Update PageRank score of each page
  - **while** converge

PageRank Implementation

Data Types

```
PageWritable {
  Integer pid;
  Double PR;
  List<Integer> outlinks;
}
```

Unique ID of a page

The PageRank score of a page

The ID of the pages that current page link to

Is it scalable?

PageRank Implementation

- **Mapper**:

```
map(PageWritable key, PageWritable page, Context context) {
  for each out page pid in page.outLinks {
    context.write(pid, page);
  }
}
```

- **Reducer**:

```
reduce(LongWritable pageId, Iterable<PageWritable> inLinkList, Context context){
  PR(pageId) = (1 – d) / N;
  for each inLink in inLinkList {
    PR(pageId) += d * PR(inLink) / L(inLink)
  }
}
```

How to save space?

Apache Mahout

- A scalable machine learning and data mining library.
  Current stable version 0.8. http://mahout.apache.org/.
- Classification:
  - Multinomial Naïve Bayes
  - Random Forest
  - …
- Clustering:
  - Kmeans Clustering
  - Spectral Clustering
  - Fuzzy Kmeans
  - …
- Recommendation:
  - User and item based recommendation
  - …
- Dimensional Reduction, Topic Modeling, Regression, Matrix Factorization
  - Singular Value Decomposition
  - Latent Dirichlet Allocation
  - Hidden Markov Model
  - …

Apache Mahout

Implemented but not distributed algorithms

- Logistic Regression
- SVM
- Linear Regression

What is missed?

- Classification:
  - Decision tree based classifier: CART, J48
  - KNN
  - Distributed Logisitic Regression
  - Distributed SVM
- Clustering:
  - DBSCAN
  - Hierarchical Clustering
- Matrix Factorization:
  - Non-Negative Matrix Factorization

Limitation of Map-Reduce

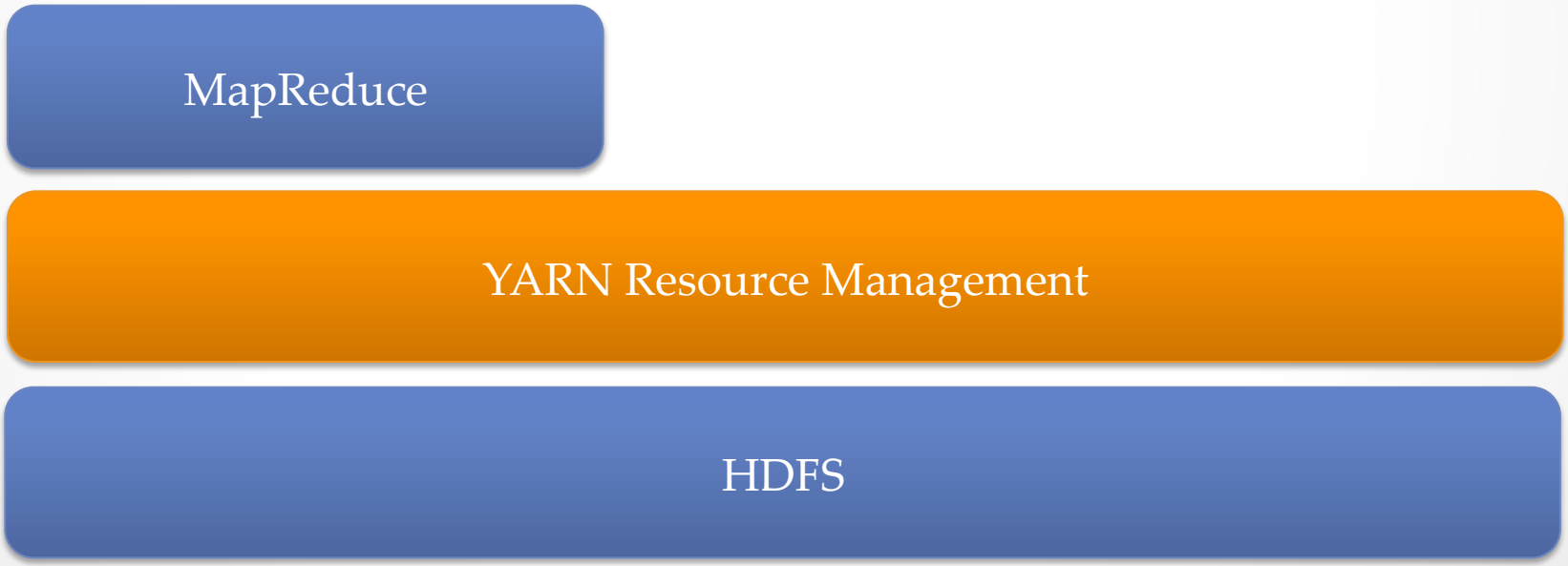
- Large overhead (10-20 seconds start up time)
- Not inherently support iteration
- No message passing
- Cannot get results in real time.

Limitation of Map-Reduce

- Large overhead (10-20 seconds start up time)
- Not inherently support iteration
- No message passing
- Cannot get results in real time.

- Extensions:
  - HaLoop: Efficient Iterative Data Processing on Large Clusters, VLDB 2010.
  - Twister: Iterative Hadoop. http://www.iterativemapreduce.org/.

Hadoop 2 Ecosystem



Hadoop 2 Ecosystem

MapReduce

YARN Resource Management

HDFS

Hadoop 2 Ecosystem

MapReduce

YARN Resource Management

HDFS

MapReduce vs. BSP

Data Intensive

Low communication needs

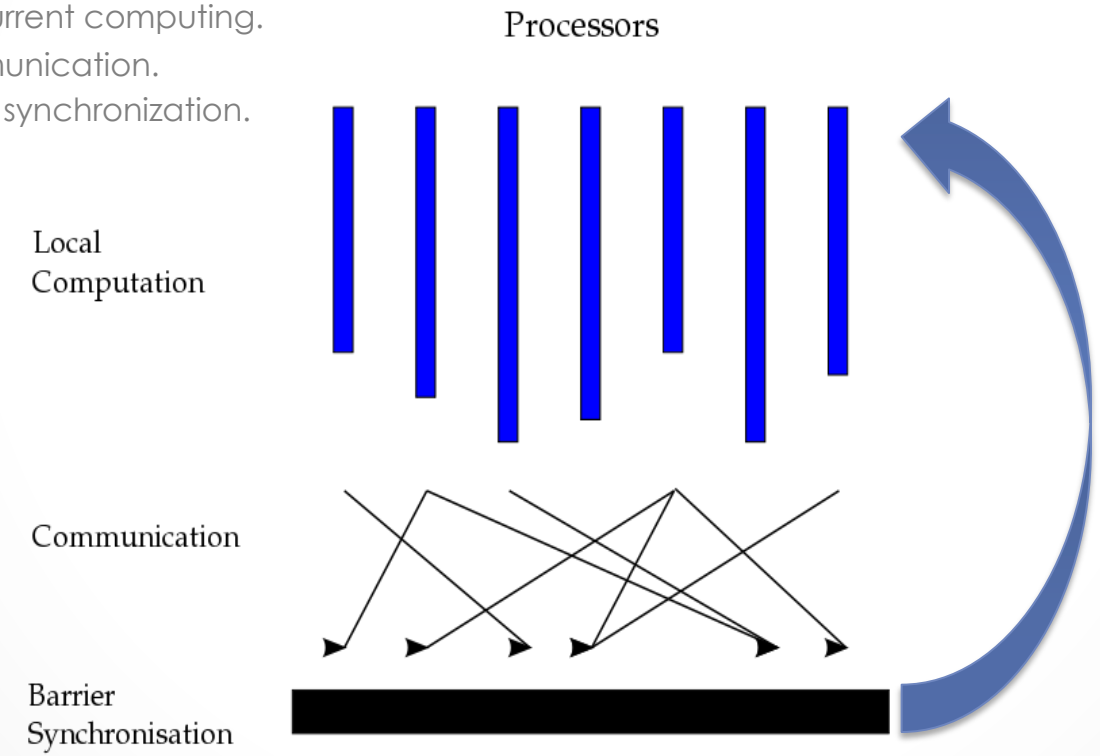
Complex Computation Intensive

High communication needs

Copy from Edward Yoon's slides of Hadoop in Seoul 2013

Bulk Synchronous Parallel

- A bridge model for designing parallel algorithms.
- Computation consists of a series of supersteps.
  - Concurrent computing.
  - Communication.
  - Barrier synchronization.



Copy from Wikipedia

Why BSP?

- Transition of the Technologies
- 2003~:
  - SQL Database connectivity interface
  - Web-scale data processing
    - MapReduce
    - Hive, Pig, Hbase
- 2007~:
  - Key/Value interface
  - Realtime, Iterative, and Graph processing
    - Storm, etc.
    - Apache Hama, etc.
    - GraphLab, Apache Giraph, Spark, etc.

Advantage of BSP?

- Low latency between iterations (supersteps).
  - Suitable for implementing iterative algorithm.
  - E.g. kmeans

- Low latency for information exchange (communication).
  - Suitable for implementing algorithms that need global or neighborhood information.
  - E.g. graph algorithms

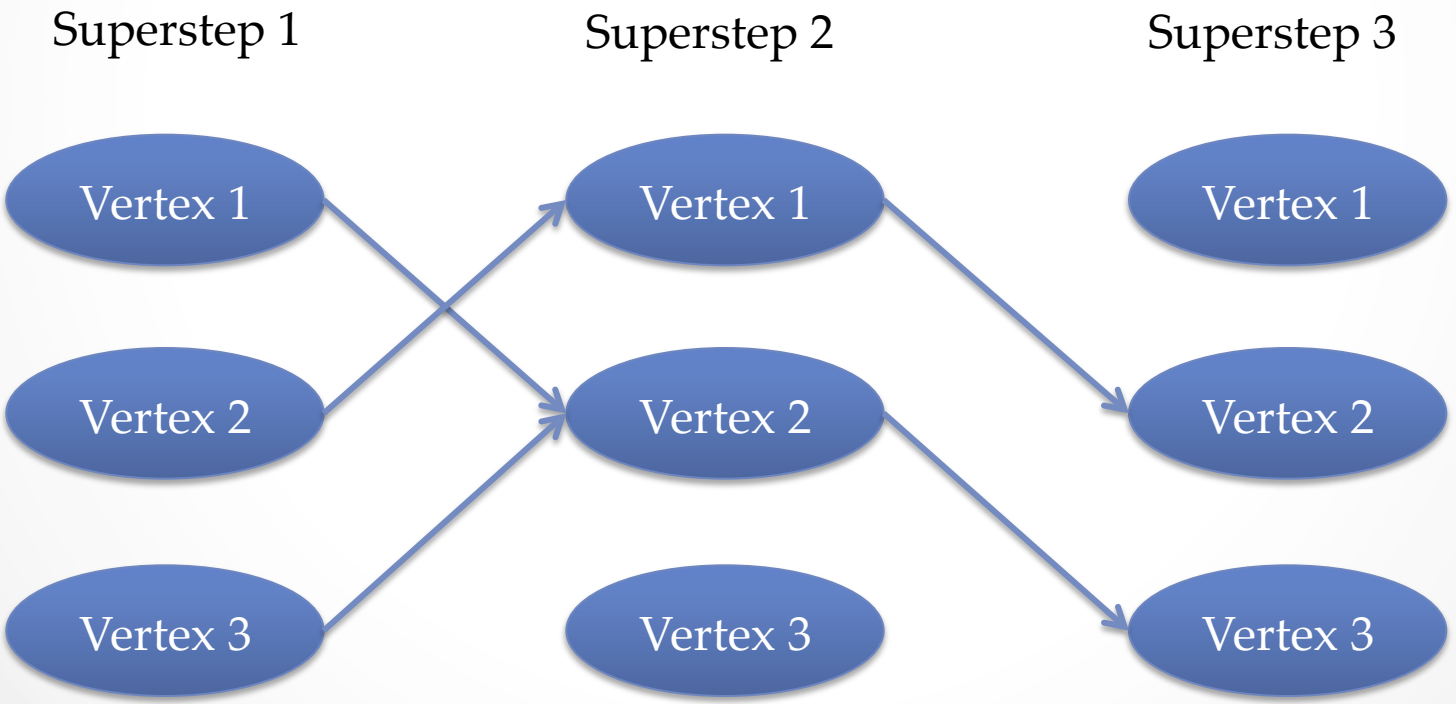
Apache Giraph

- An iterative graph processing system built for high scalability.
- An open source version of Google's Pregel.
- Current stable version: 1.0.0
- Currently mainly developed and maintained by Facebook

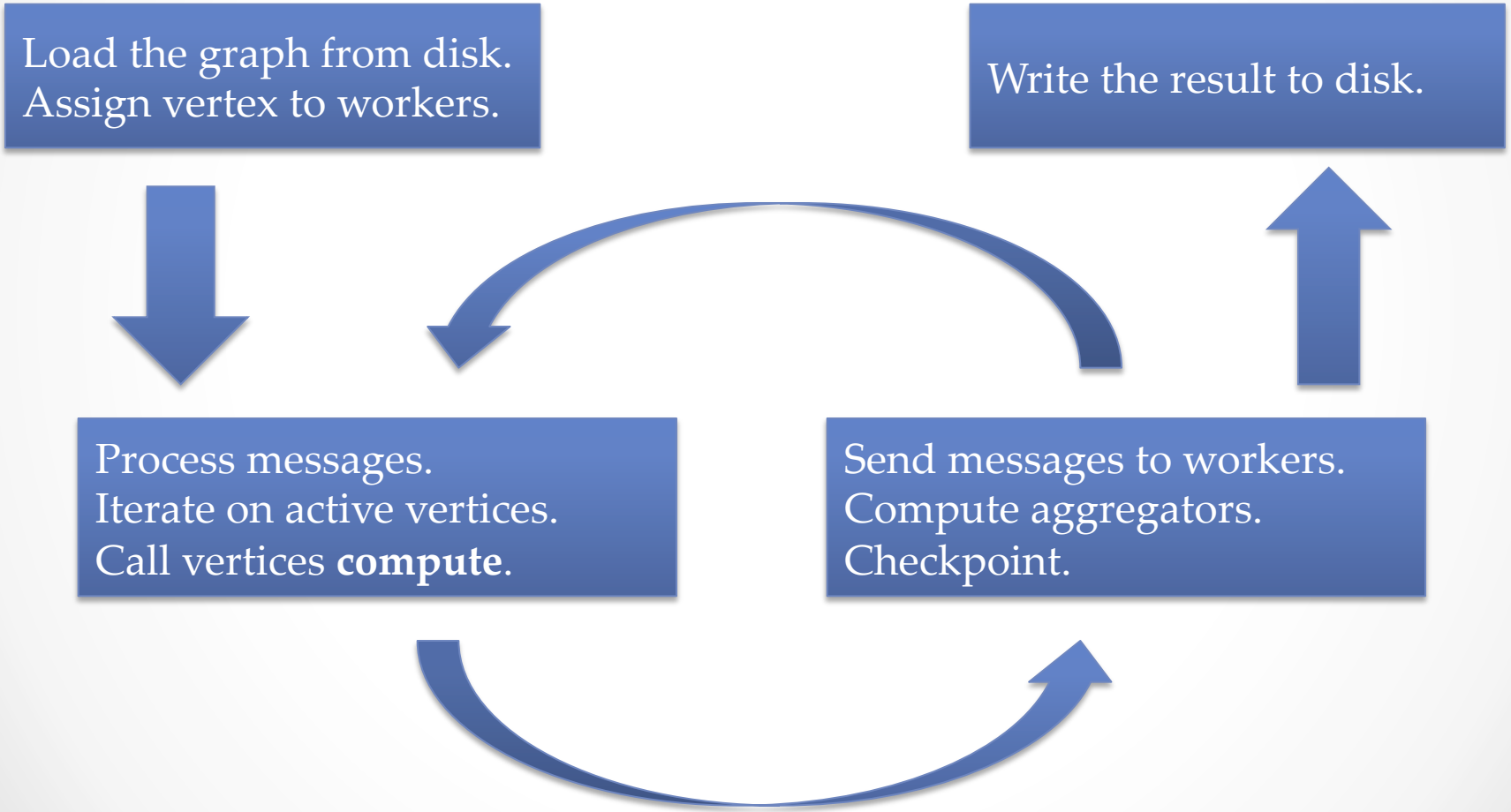
Fundamentals of Giraph

Graphs are composed of **vertices** and **edges**.

Vertex centric computing framework.



Giraph Workflow



Fundamentals of Giraph

Core entity types:

- **Vertex ID**: Identify a vertex
  - Implements WritableComparable. Typically a LongWritable.
- **Vertex Value**: Store a vertex associated data.
  - Implements Writable. Can contain anything that implements Writable.
- **Edge Value**: Store a value on an outgoing edge.
  - Implements Writable. Can contain anything that implements Writable.
- **Message Value**
  - Implements Writable. Can contain anything that implements Writable.

Core API

- **VertexReader**: Read the vertices from an input split.
  - getCurrentVertex(): An iterator method to retrieve the current vertex.

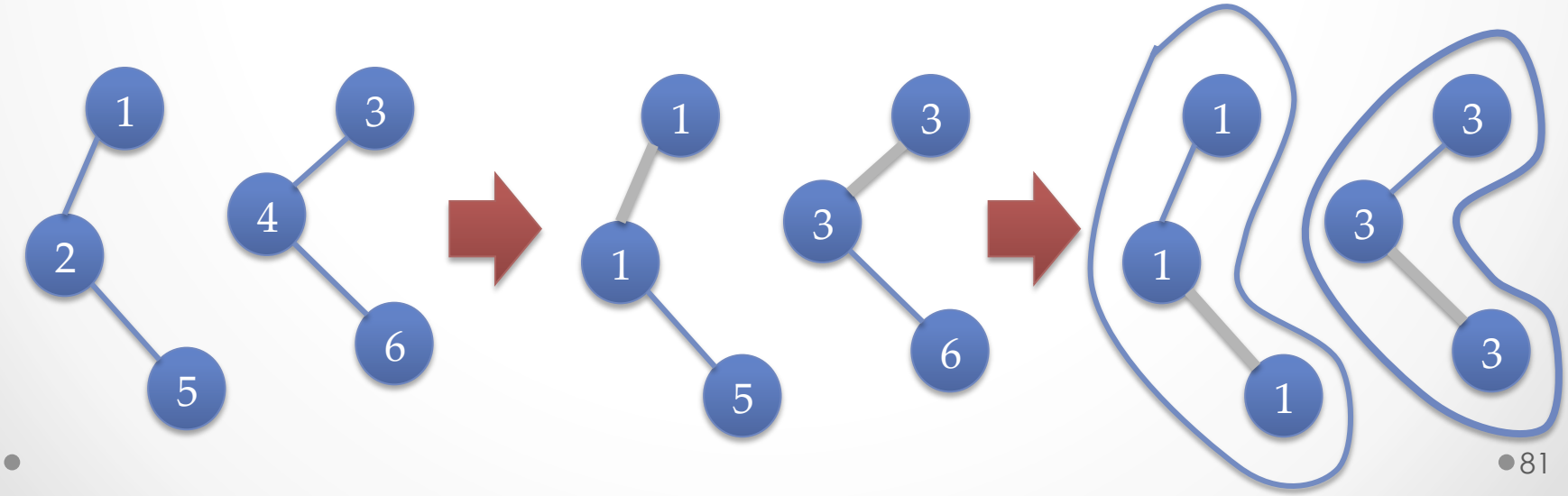
- **Vertex**: Store the vertex data, edge data of a vertex.
  - compute(Iterable message): The core computing logic of each vertex.

Giraph Basic Example

Connected Component

In each superstep:

1. Process received messages.
2. Get the id with smallest number.
3. Send id to all neighbors whose id is larger than current vertex.



Performance

According to Sebastian Schelter's reports.

Setup: 6 machines with 2x 8-core cpu, 4x1TB disks, 32G RAM each machine, 1 Giraph worker per core.

Experiment:

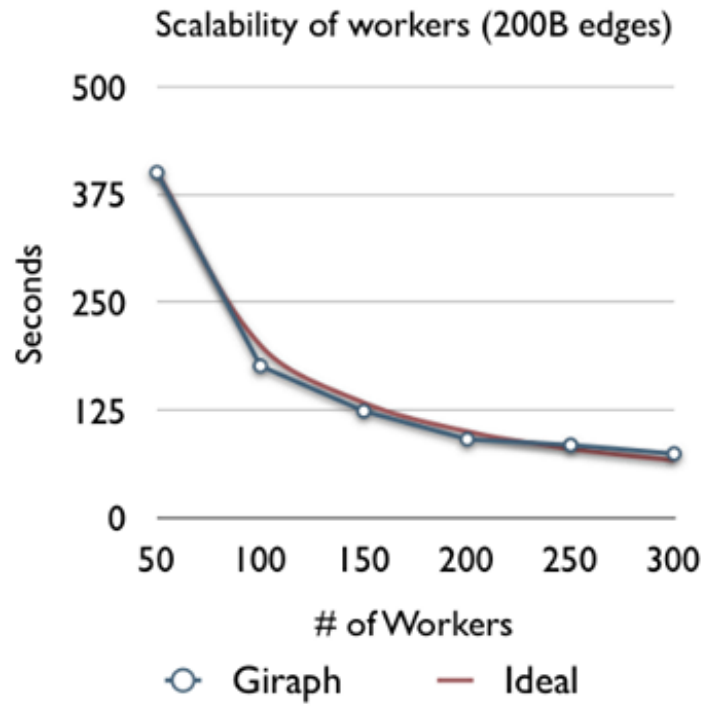
Data: Wikipedia page link graph (6M pages, 200M edges)

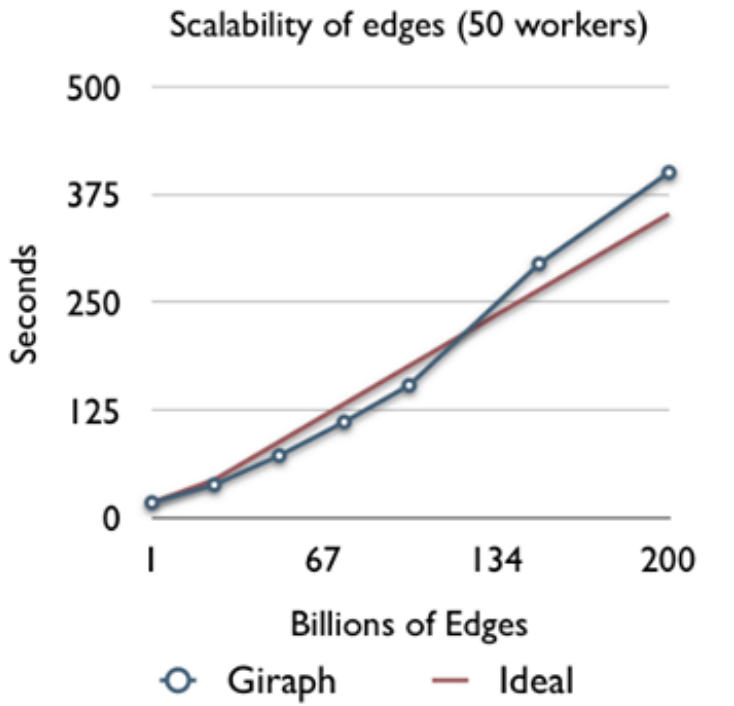
Experiments:

- PageRank on Hadoop/Mahout
  - 10 iterations approx. 29 min
  - Average time per iteration: approx. 3min
- PageRank on Giraph
  - 30 iterations approx. 15 min
  - Average time per iteration: approx. 30sec
- 6x Performance Improvement

Performance (Cont.)

- An experiment conducted by Facebook on 200 billion edges





Suitable Algorithms

- Most of the graph algorithms
- PageRank
- Alternative Least Square
- Non-Negative Matrix Factorization

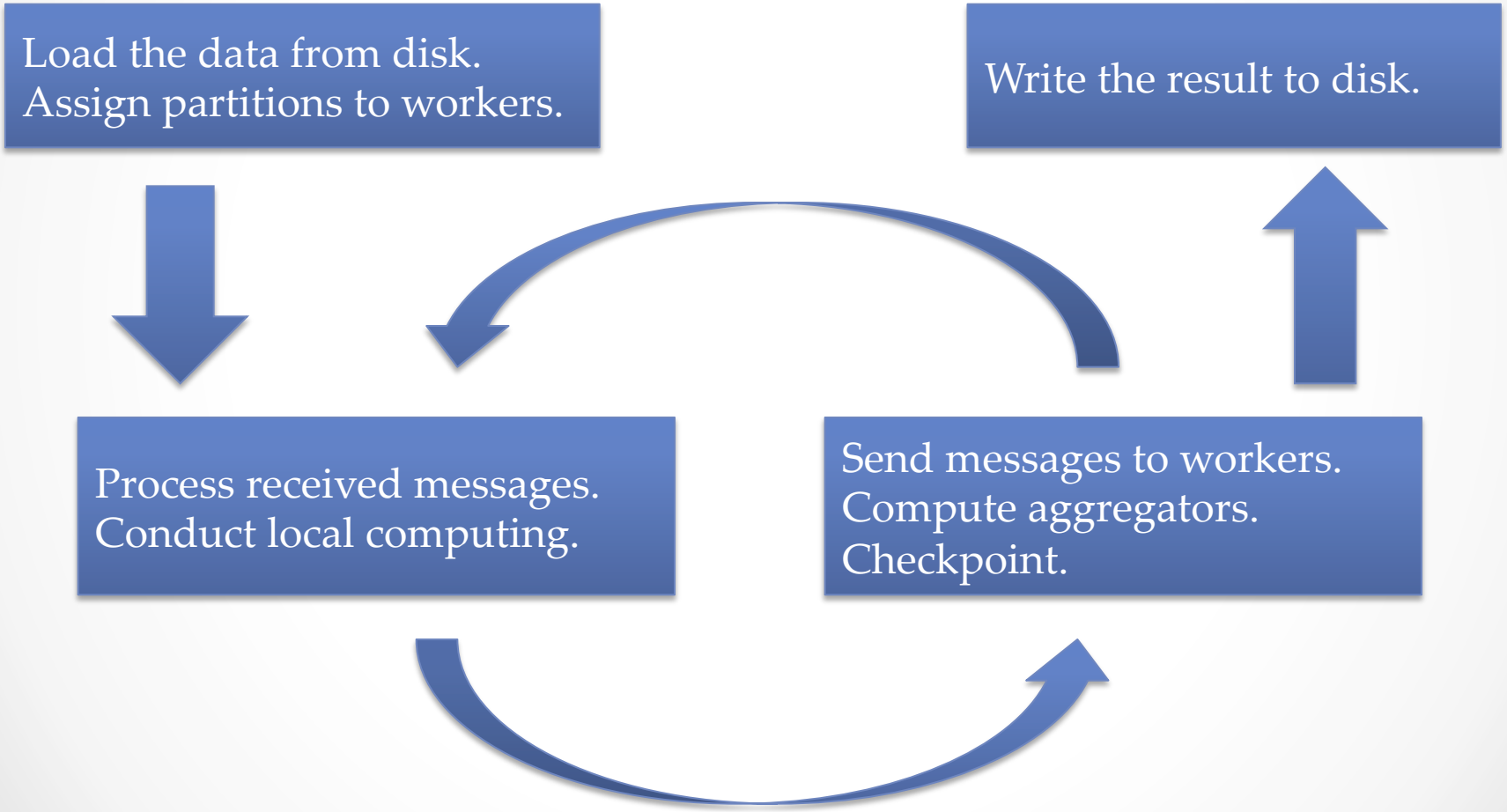
Related Materials

- **Giraph web site**: http://giraph.apache.org/
- **Github**: https://github.com/apache/giraph
- **User mailing list**: user@giraph.apache.org
- **Facebooks engineering blog**: https://www.facebook.com/Engineering

Apache Hama

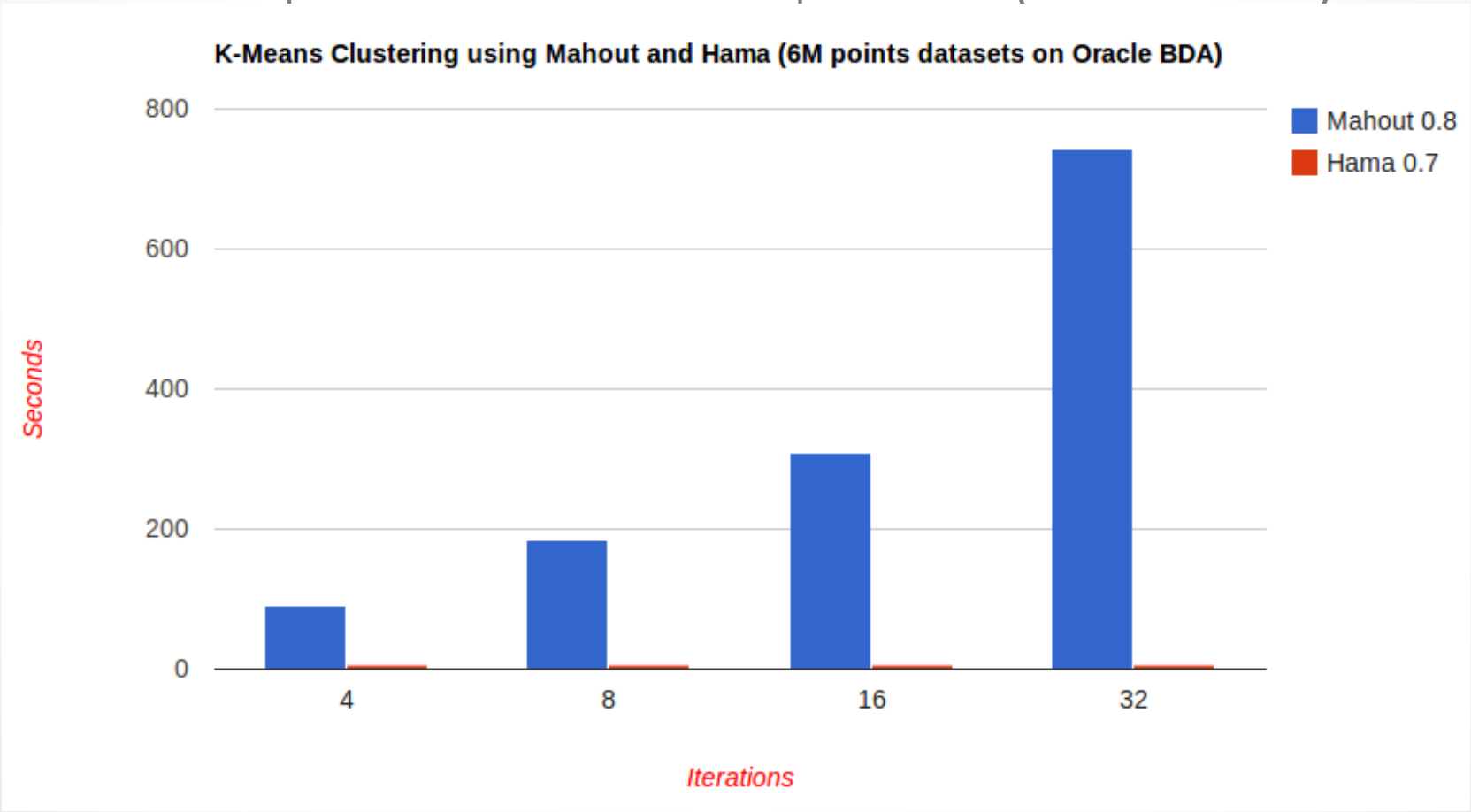
- An implementation of general BSP framework, lead by Edward Yoon in Oracle, now a founder of a stealth startup.
- Part of Hadoop 2 ecosystem, running on top of HDFS.
- Dedicatedly design for distributed iterative algorithms.

Hama Workflow



Performance Evaluation

- Kmeans performance comparison (100x faster)



More Performance Evaluation

- PageRank
  - Machine: 1 Rack Oracle BDA (18 nodes), 170 tasks
  - Dataset: 5,716,808 pages and 130,160,392 links.
  - Execution Time: 18.465 seconds.

- Single Source Shortest Path (SSSP)
  - Machine: 1 Rack Oracle BDA (18 nodes), 180 tasks.
  - Dataset: 1 Billion Edges.
  - Execution Time: 473.52 second.

Fundamental of Hama

- **BSP (interface):**
  - **setup**(BSPPeer): Conduct the initialization task, e.g. load parameters. Called before **bsp**.
  - **bsp**(BSPPeer): Conduct the core computation.
  - **cleanup**(BSPPear): Conduct the clean up tasks, e.g. write results to disk. Called after **bsp**.
- **BSPPeer**:
  - **send**(String peerName, M msg): Send a message to a peer.
  - **sync**: Enter the barrier synchronization stage.
  - **readNext**(K key, V value): Read a key/value pair from input stream.
  - **write**(K key, V value): Write a key/value pair to output collector.
  - **reopenInput**: Reopen the input and read from the beginning.
  - **getCurrentMessage**: Get the current message from a FIFO.
  - …

Sketch of a typical Hama Program

```
Sketch implements BSP {
  public void setup(BSPPeer peer) {
    Conduct initialization task.
  }

  public void cleanup(BSPPeer peer) {
    Conduct cleanup task.
  }

  public void bsp(BSPPeer peer) {
    while (condition not satisfied) {
      1. Open or reopen the input stream.
      2. Computing.
      3. Send message to peers.
      4. Sync
      5. Process the incoming messages.
    }
  }
}
```

“Hello World!” of Hama

```
public class ClassSerializePrinting extends BSP<NullWritable, NullWritable,
IntWritable, Text> {

  public static final int NUM_SUPERSTEPS = 15;

  public void bsp(BSPPeer<NullWritable, NullWritable, IntWritable, Text> bspPeer) {

    for (int i = 0; i < NUM_SUPERSTEPS; i++) {
      for (String otherPeer : bspPeer.getAllPeerNames()) {
        bspPeer.send(otherPeer, new IntegerMessage(bspPeer.getPeerName(), i));
      }
      bspPeer.sync();
      IntegerMessage msg = null;
      while ((msg = (IntegerMessage) bspPeer.getCurrentMessage()) != null) {
        bspPeer.write(new IntWritable(msg.getData()), new Text(msg.getTag()));
      }
    }
  }
}
```

Send Messages

Synchronization

Fundamental of Hama Graph

- VertexInputReader:
  - **parseVertext**(K key, V value, Vertex vertex): parse the input.
- Vertex:
  - **compute**(M messages): The computing logic of each vertex.
  - **voteToHalt**: Once all the vertices vote to halt, the program will terminate.

Sketch:

```
compute(Iterable<M> messages) {
  Initialization.
  Process incoming messages.
  Core computing.
  Check whether vote to halt.
  Send messages to neighbors.
}
```

Suitable Data Mining Algorithms

- PageRank
  - Using Hama's Graph API
- Kmeans
  - Programming in master/slave style
- Neural Network
  - Programming in master/slave style
- Linear Regression
- Logistic Regression
- Autoencoder

PageRank in Hama

- **Data format:**
  - VertexID <List of Neighbor IDs>
- **Graph Initialization:**

```
public boolean parseVertex(Text key, TextArrayWritable value,
        Vertex<Text, NullWritable, DoubleWritable> vertex) {
  vertex.setVertexID(key);

  for (Writable v : value.get()) {
    vertex.addEdge(new Edge<Text, NullWritable>((Text) v, null));
  }

  return true;
}
```

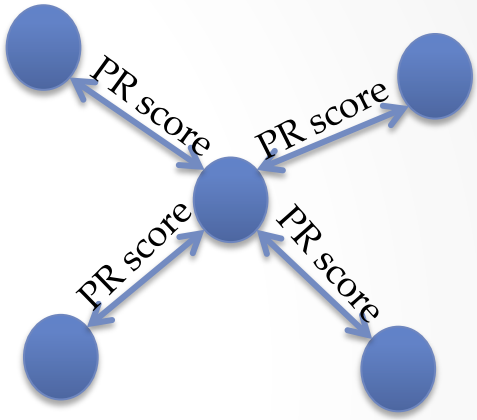
Add edges

PR Score Computation

```
public void compute(Iterable<DoubleWritable> messages) {
  if (getSuperStep() == 0) {
    setValue(new DoubleWritable(1.0 / getNumVertices());
  }
  else {
    double sum = 0;
    for (DoubleWritable msg : messages) {
      sum += msg.get();
    }
    setValue(
DAMPING_F
    if (COUNT
      voteToHal
      break;
    }
    sendMessage
  }
}
```

Initialize score

Process incoming message

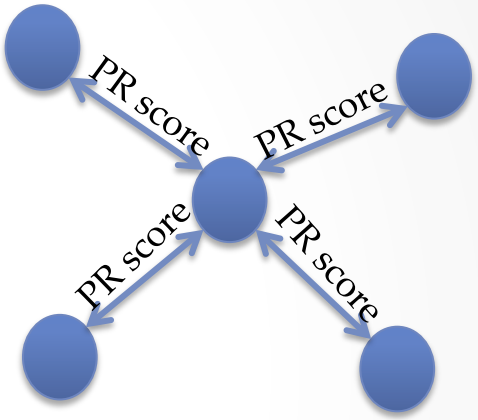


PR Score Computation

```
public void compute(Iterable<DoubleWritable> messages) {
  if (getSuperStep() == 0) {
    setValue(new DoubleWritable(1.0 / getNumVertices());
  }
  else {
    double sum = 0;
    for (DoubleWritable msg : messages) {
      sum += msg.get();
    }
    setValue(new DoubleWritable((1.0 – DAMPING_FACTOR) / getNumVertices() + (sum *
DAMPING_FACTOR));
    if (CONVERGENCE_THRESHOLD > getLastAggregatedValue()) {
      voteToHalt();
      break;
    }
    sendMessageToNeighbors(new DoubleWritable(getValue().get() / getEdges().size()));
  }
}
```

Initialize score

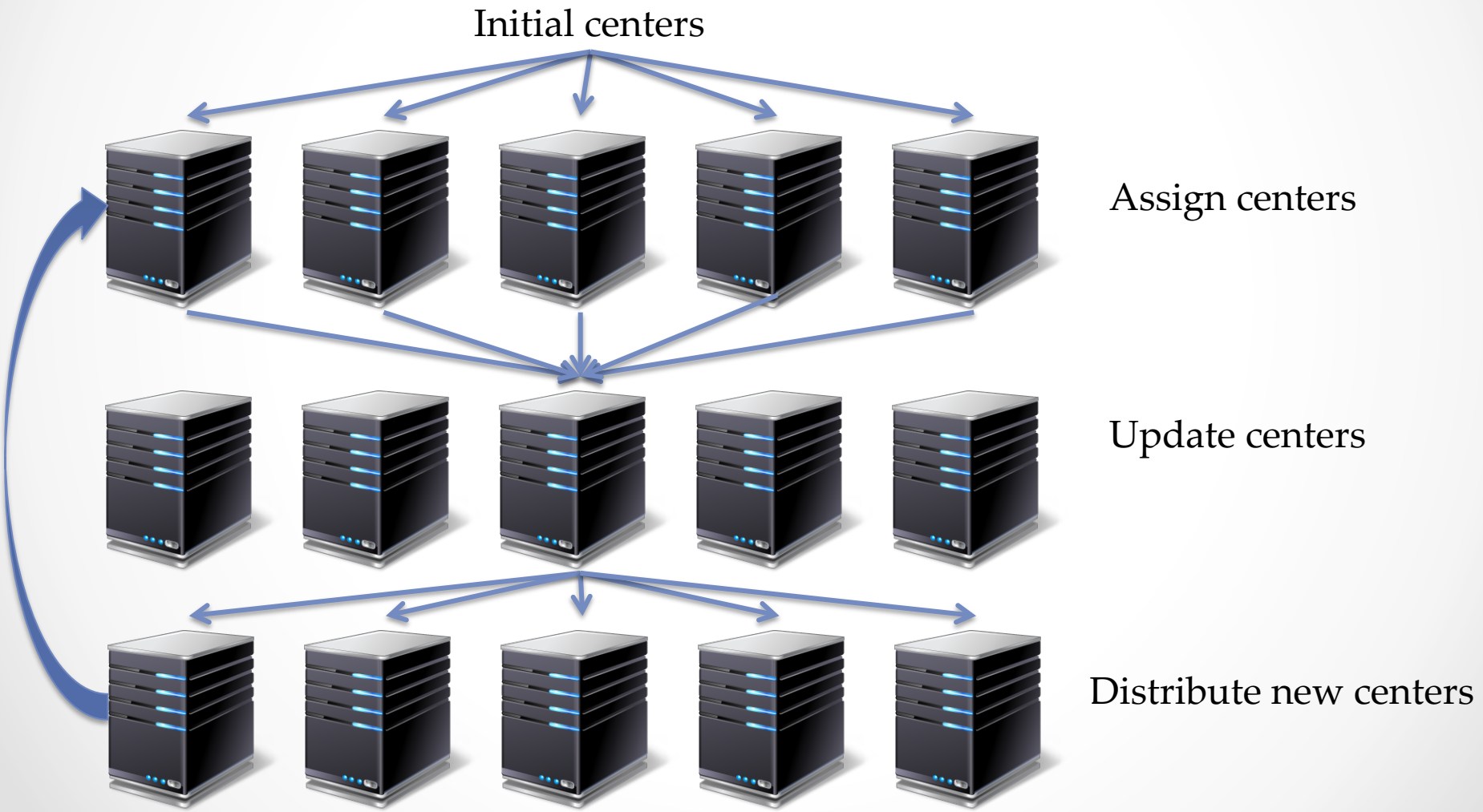
Process incoming message



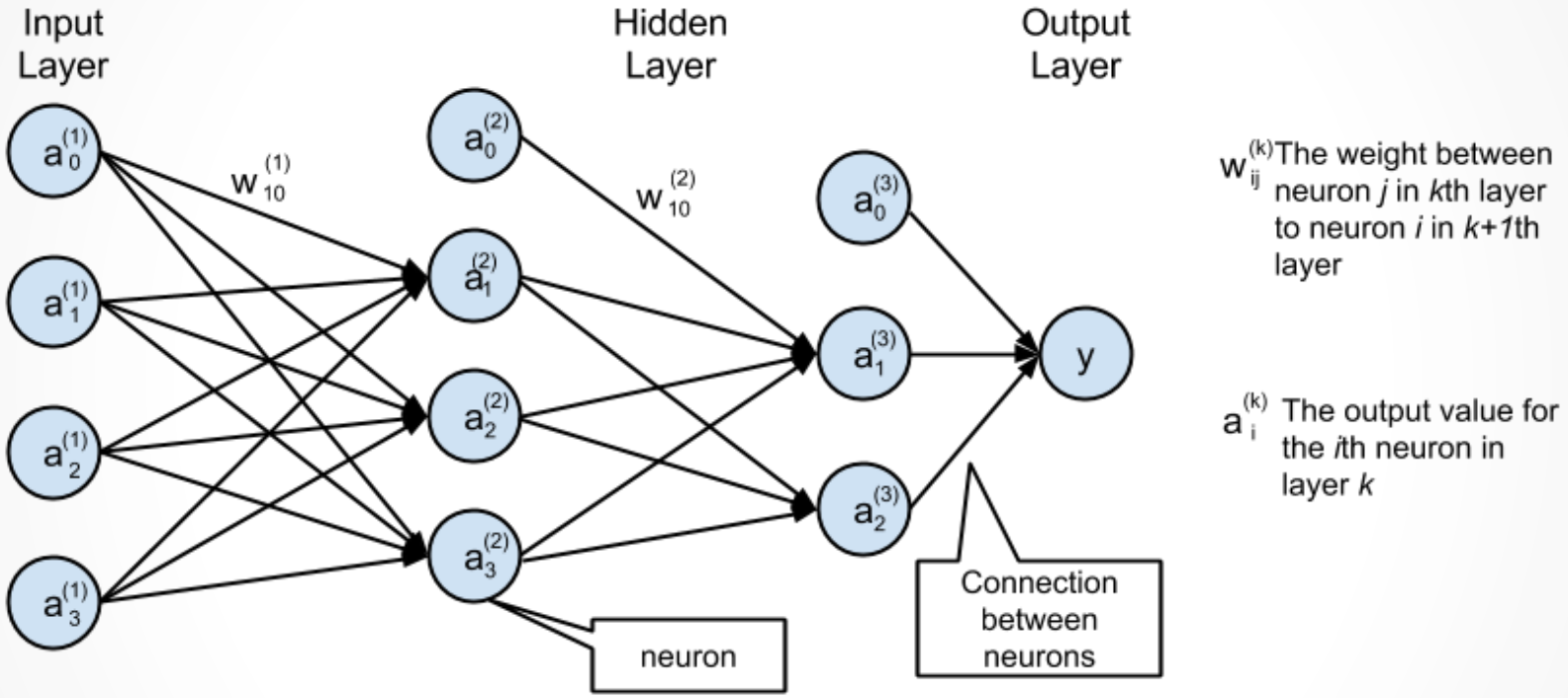
Check termination condition

Send messages to neighbors

Distributed Kmeans in Hama



Neural Network in Hama



- Train the model via mini-batch stochastic gradient descent.

Neural Network in Hama

- Cost function

$$cost(t, y) = \frac{1}{2}\sum_{i=1}^{N}(t - y)^2 = \frac{1}{2}\sum_{i=1}^{N}(t - f(x))$$

$$= \frac{1}{2}\sum_{i=1}^{N}(t - f(w^T o)) = \frac{1}{2}\sum_{i=1}^{N}(t - \frac{1}{1 + e^{-w^T o}})$$

o denotes the output of the output layer

Sigmoid function

Neural Network in Hama

- Cost function

$$cost(t,y) = \frac{1}{2}\sum_{i=1}^{N}(t-y)^2 = \frac{1}{2}\sum_{i=1}^{N}(t-f(x))$$

$$= \frac{1}{2}\sum_{i=1}^{N}(t-f(w^T o)) = \frac{1}{2}\sum_{i=1}^{N}(t-\frac{1}{1+e^{-w^T o}})$$

o denotes the output of the output layer

- Decompose the weight update

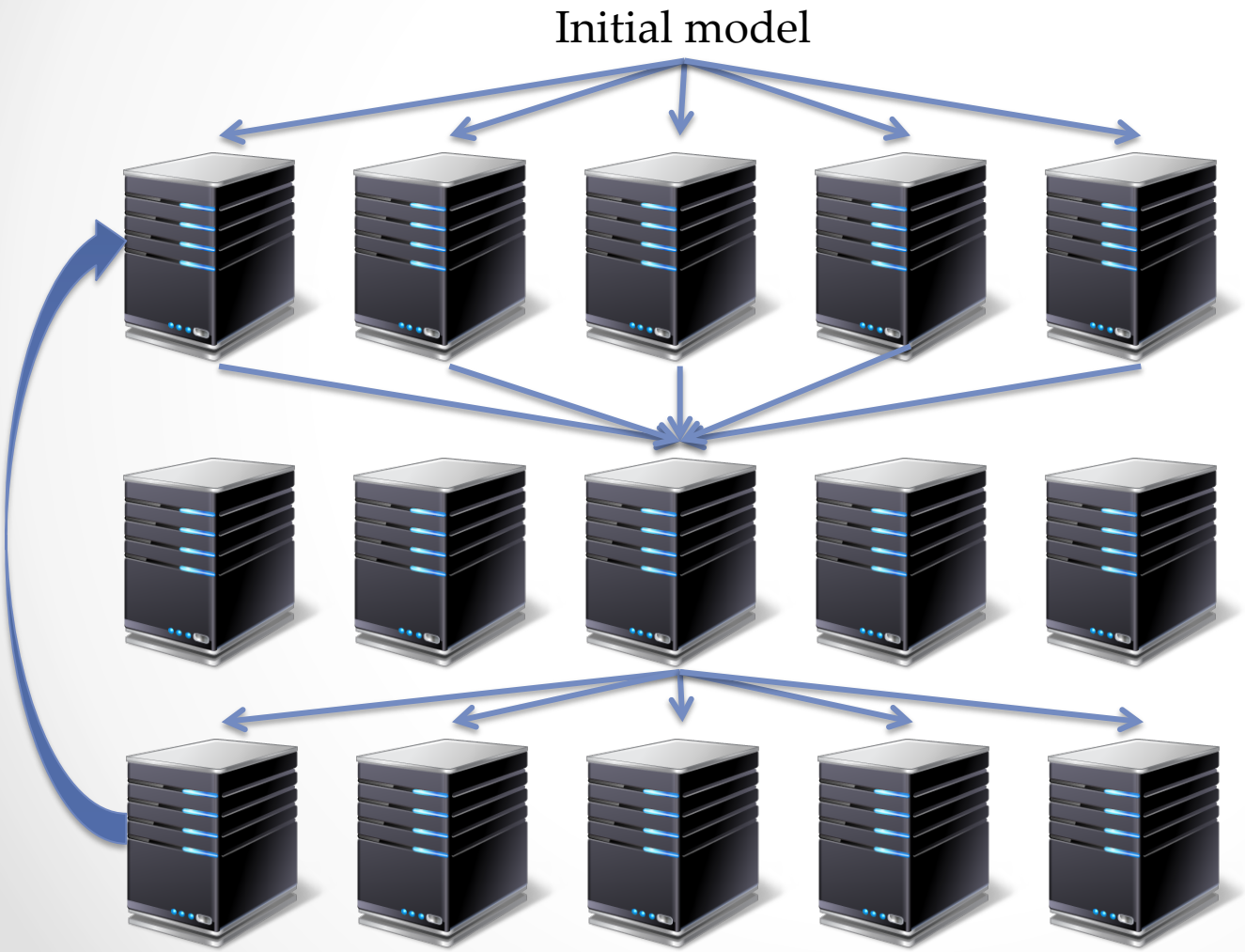
$$\frac{\partial cost(t,y)}{\partial w_i} = \frac{\partial cost(t,y)}{\partial f}\frac{\partial f}{\partial x}\frac{\partial x}{\partial w_i}$$

$$= -\sum_{i=1}^{N}(t-y)\cdot(f(x)\cdot(1-f(x)))\cdot o_i$$

$$= -(\sum_{i=1}^{N_i}(t-y)\cdot(f(x)\cdot(1-f(x)))\cdot o_i) + \cdots + (\sum_{i=1}^{N_m}(t-y)\cdot(f(x)\cdot(1-f(x)))\cdot o_i)$$

Each part can be computed independently

Distributed Neural Network in Hama



Related Materials

- **Hama web site**: http://hama.apache.org/
- **Hama wiki**: http://wiki.apache.org/hama/
- **User mailing list**: user@hama.apache.org
- **Book in preparation**: Hama in Action, Edward Yoon et al.

Disadvantage of Hadoop Mapreduce and BSP

- The computing is not efficient if data can be load into memory of all the machines.
  - Hadoop, Giraph, and Hama would implicitly write data to disk.

- Writing Java code is time-consuming and it is not concise.
  - It takes very long time to design and implement a distributed program.
  - One program can easily exceed 1k lines of code and is hard to maintain.

Apache Spark

- In-memory cluster computing framework.
- Implemented in Scala, a OOP and functional programing language.
- Invented by AMP Lab of University of California Berkeley, lead by Michael Franklin, Michael Jordan and Ion Stoica.
- Designed and developed by a 79 people team.
- Was contributed to Apache Software Foundation in 2013.

How concise it is?

- An example of Logistic Regression.

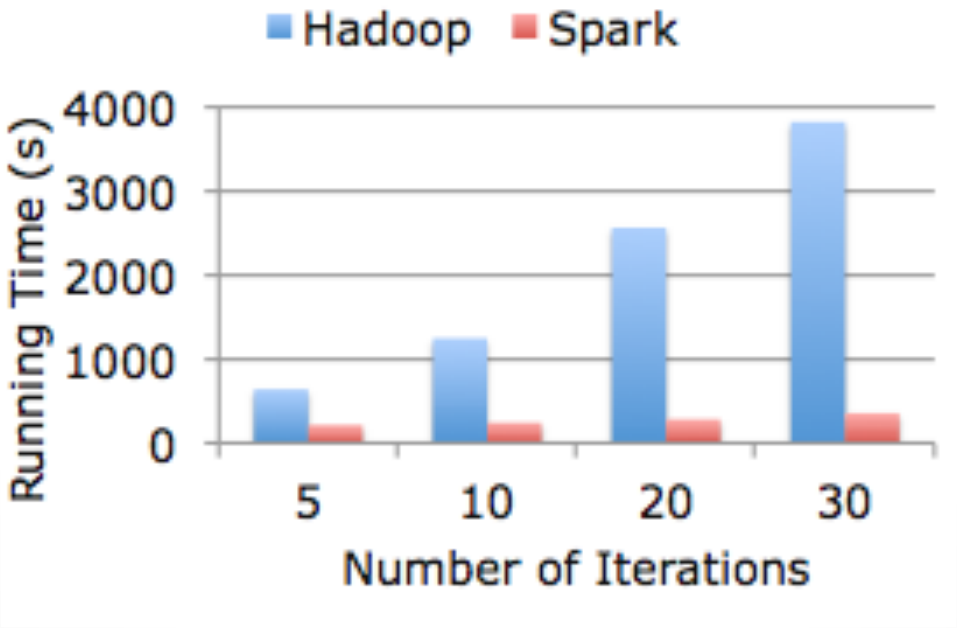
```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D) // current separating plane
for (i <- 1 to ITERATIONS) {
  val gradient = points.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}
println("Final separating plane: " + w)
```

Read and cache data

Iteratively update

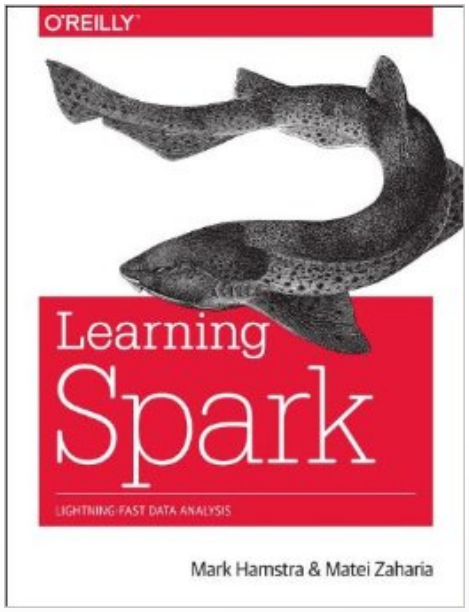
How fast it is?

- Data: 30 GB
- Machine: 80-core cluster
- 20x faster



Related Materials

- **Spark web site**: http://spark.incubator.apache.org/
- **Github**: https://github.com/apache/incubator-spark
- **User mailing list**: user@spark.incubator.apache.org
- **Book**: Learning spark, Mark Hamstra and Matei Zaharia.



Real Time Computing Framework

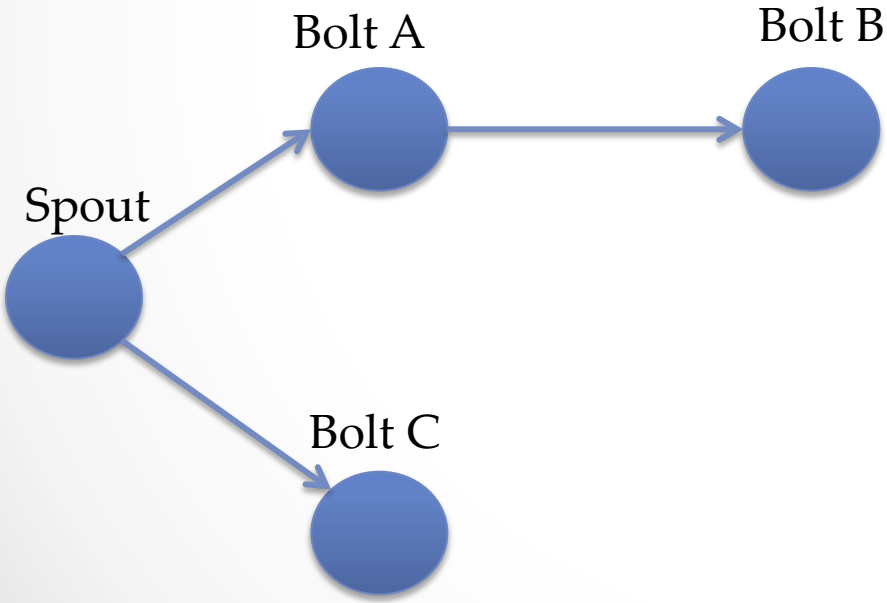
- Online algorithms are designed to be trained and evaluated in real time.
- Even with "light-fast" clustering computing, the results are not available in read time.
- Single machine version algorithms can be used in real time but not scalable.

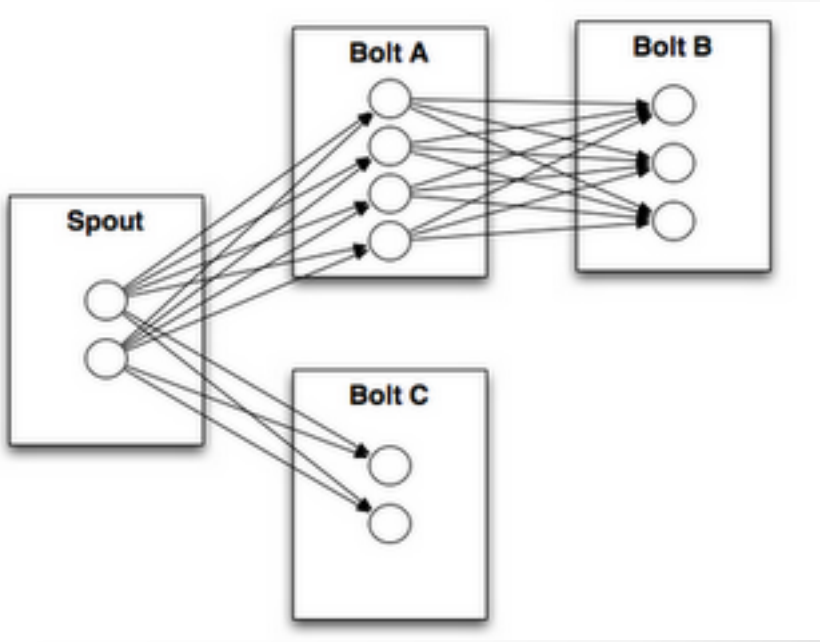
Apache Storm

- Invented by Nathan Marz in Twitter, who is now a founder of stealth startup.
- Implemented in Clojure, a pure functional programming.
- Contributed to Apache Software Foundation in 2013.
- Currently used by many companies.

Components

- **Spout**: The source of streams. Can get the data stream from data generator, e.g. message queue.
- **Bolt**: Consumes any number of input streams, does process, and may emit new data stream.



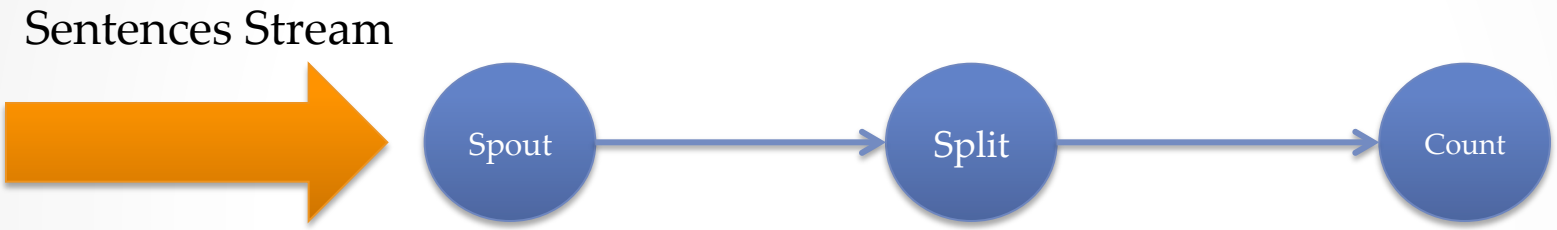


Core APIs

- **Spout (Take JMSSpout for example)**
  - **setJmsProvider**: Specify the user defined JMS provider. A provider is in charge of connecting to the message broker. It will send message to the spout.
  - **setJmsTupleProducer:** Specify the user defined JMS producer. A producer is in charge of parsing the received messages from JMS and sending parsed message to the bolt.

- **Bolt**
  - **prepare**: Conduct the initialization task.
  - **execute**: Conduct the core processing.
  - **cleanup**: Conduct the cleanup task.
  - **declareOutputField**: Define the format of output.

Word Count in Storm




```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("sentences", new RandomSentenceSpout(), 5);
builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("sentences");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

Word Count in Storm

- SplitBolt

```
public void execute(Tuple tuple, BasicOutputCollector collector) {
  String[] tokens = tuple.getString(0).split(" ");
  for (String token : tokens) {
    collector.emit(new Values(word, 1));
  }
}
```

- CountBolt

```
public void execute(Tuple tuple, BasicOutputCollector collector) {
  String word = tuple.getString(0);
  Integer count = counts.get(word);
  if (count == null)
    count = 0;
  count++;
  counts.put(word, count);
  collector.emit(new Values(word, count));
}
```

Machine Learning over Storm

- Trident-ml: An online algorithms library over storm.
  - Linear Classification
  - Linear Regression
  - Clustering (Kmeans)
  - Feature scaling (standardization, normalization)
  - Text feature extraction
  - Stream statistics (mean, variance)
  - Pre-trained Twitter Sentiment Classifier

The packaged algorithms are designed to fit into limited memory and processing time but they don't work in a distributed way!

Related Materials

- **Storm website**: http://storm-project.net/
- **Storm github**: https://github.com/nathanmarz/storm
- **Book**: Get started with storm, Jonathan Leibiusky, Gabriel Eisbruch, Dario Simonassi.

- **Mailing list**:
  - user@storm.apache.incubator.org

Summary

- This is the big data era.
- Most of the **practical** ML and DM projects in companies are considered to be large scale.
- Convert traditional ML and DM algorithms into a distributed algorithms is **not as easy as you think**.
- A typical time period of designing and implementing a new algorithm requires **several month in full time**.

How to learn more?

- **Slideshare**: Tons of interesting and useful slides.
- **InfoQ**: A practitioner-driven community news site focused on facilitating the spread of knowledge and innovation in enterprise software development.
- **Top companies' engineering blog**: First-hand resource from the people who invented the things.
- **Quora**: Ask question and get answers from the real domain experts!