Dharwad
ज्ञानेन विकासः

INDIAN INSTITUTE OF
INFORMATION
TECHNOLOGY

Data Science, Knowledge Discovery, and Big Data Analytics - Systems

Dr. Animesh Chaturvedi
Assistant Professor: IIIT Dharwad
Young Researcher: Heidelberg Laureate Forum
Postdoc: King's College London & The Alan Turing Institute
PhD: IIT Indore MTech: IIITDM Jabalpur

HEIDELBERG
LAUREATE
FORUM

KING'S
College
LONDON

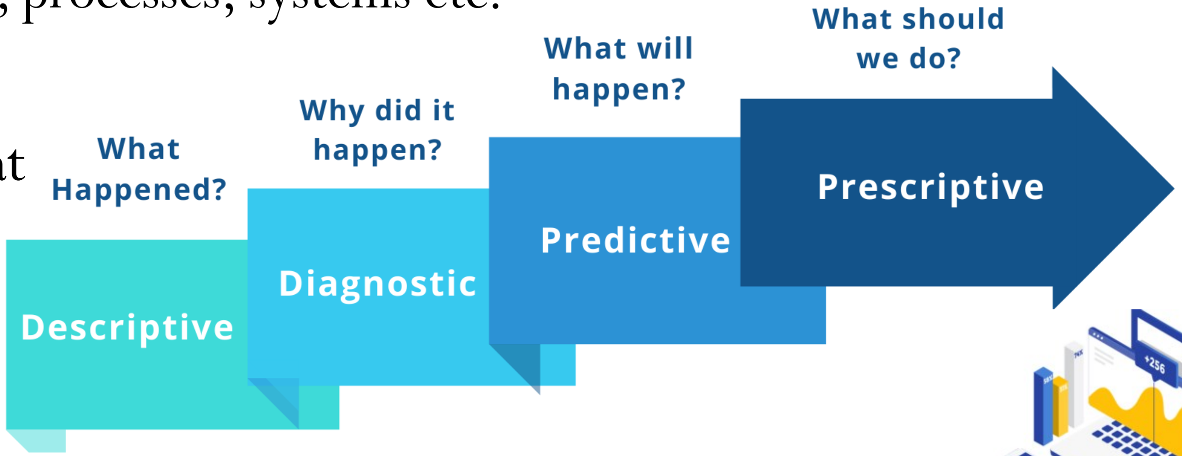
Indian Institute of Technology Indore
भारतीय प्रौद्योगिकी संस्थान इंदौर

PDPM
Indian Institute of Information Technology,
Design and Manufacturing, Jabalpur

The
Alan Turing
Institute

Data Science and Data Analytics

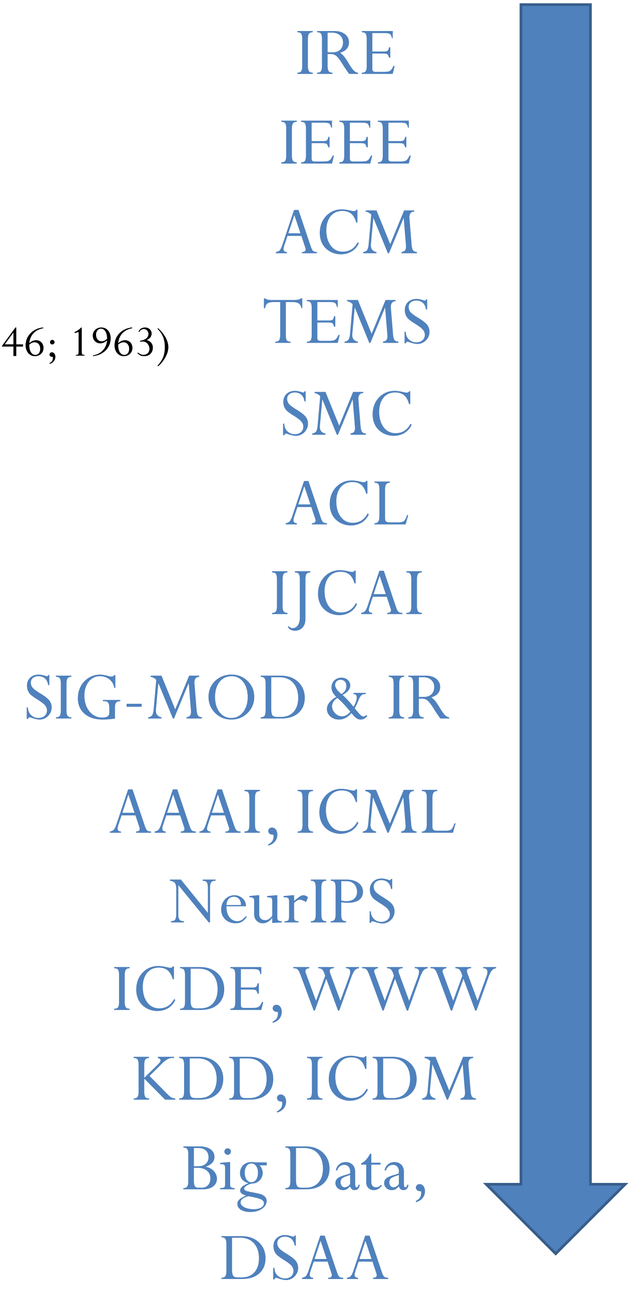
- **Data Science:** interdisciplinary science that
  - deals with data: methods, algorithms, processes, systems etc.
  - Theory oriented
- **Data Analytics:** analysis of data that
  - discovers trends, graph, tables etc.
  - Technology oriented



- Both
  - extracts knowledge and apply actions through structured and unstructured data insights
  - deals with data mining, machine learning, data management, and big data.
- **Data Scientist** and **Data Analyst** applies Data Science and Data Analytics

Evolution of Data based Societies

- Institute of Radio Engineers (**IRE**) in 1951
- Institute of Electrical and Electronics Engineers (**IEEE**), **IEEE Computer Society** (1946; 1963)
- Association for Computing Machinery (**ACM**) in 1947
- Technology Engineering and Management Society (**TEMS**) 1955
- IEEE Systems, Man, and Cybernetics Society (**SMC**) (1958; 1972)
- Association for Computational Linguistics (**ACL**) 1962
- International Joint Conf. on Artificial Intelligence (**IJCAI**) 1969
- Special Interest Group on Management of Data (**SIG-MOD**), 1975
- Special Interest Group on Information Retrieval (**SIG-IR**) 1978
- Association for the Advancement of Artificial Intelligence (**AAAI**) 1979
- International Conf. on Machine Learning (**ICML**) 1980 in Pittsburgh
- Conf. on Neural Information Processing Systems (**NeurIPS**) 1986 -1987
- 38th IEEE International Conf. on Data Engineering (**IEEE ICDE**)*
- International World Wide Web Conf. (**WWW**) - Web Conf. 1994
- 28th International Conf. on Knowledge Discovery and Data Mining (**SIG-KDD**)*
- 22nd IEEE International Conf. on Data Mining (**IEEE ICDM**)*
- 10th IEEE International Conf. on Big Data (**IEEE Big Data**)*
- 9th IEEE International Conf. on Data Science and Advanced Analytics (**IEEE DSAA**)*



IRE

IEEE

ACM

TEMS

SMC

ACL

IJCAI

SIG-MOD & IR

AAAI, ICML

NeurIPS

ICDE, WWW

KDD, ICDM

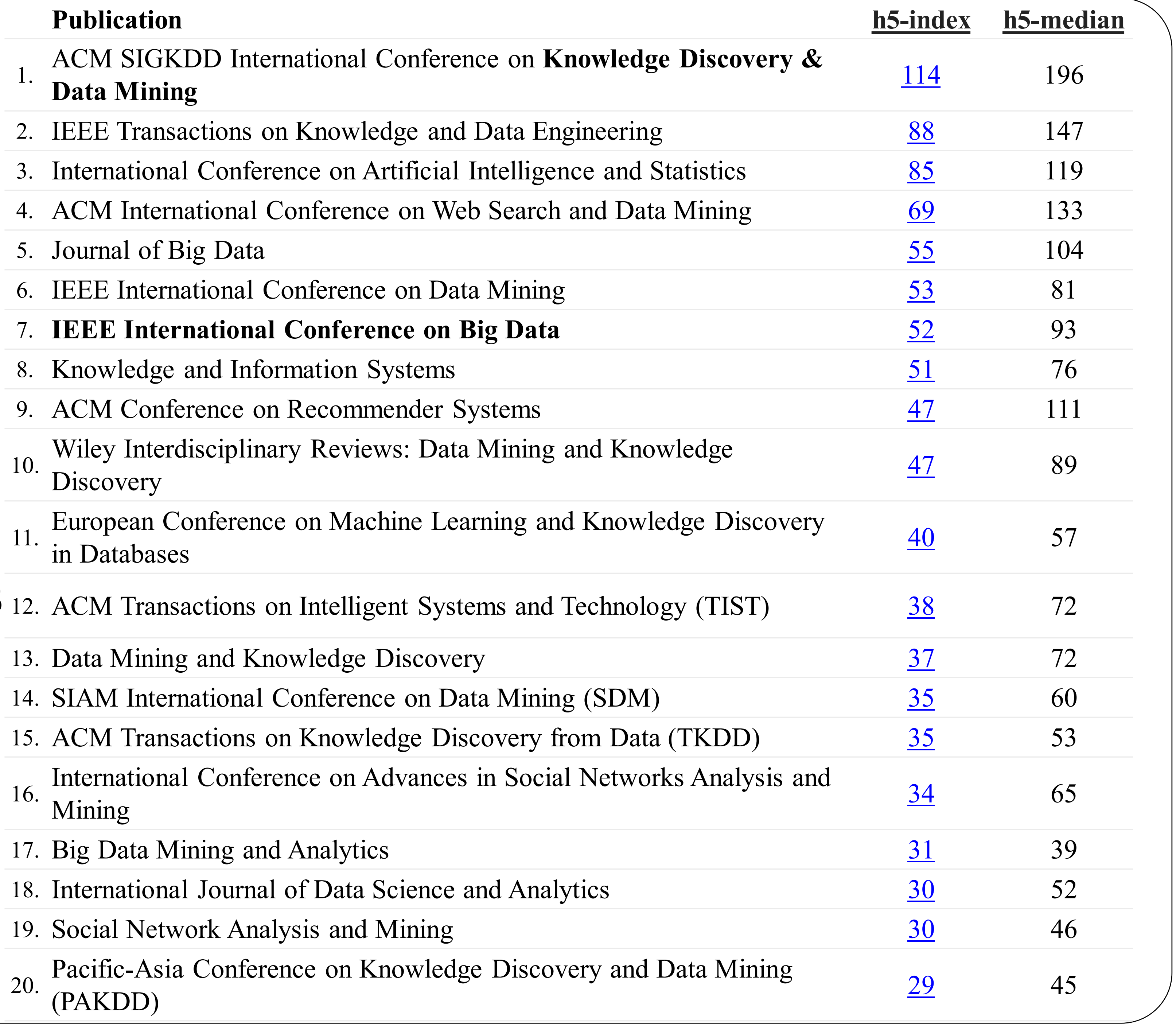
Big Data,

DSAA

*** in 2022**

DBLP & Google Scholar

- "Data" in DBLP search
  - There are 459+ Venues matched in the DBLP, world most referred computer science bibliography website.
  - DBLP launched in 1993
- "Data" in Google Scholar metrics
  - Top 20 publications venues matching "data"

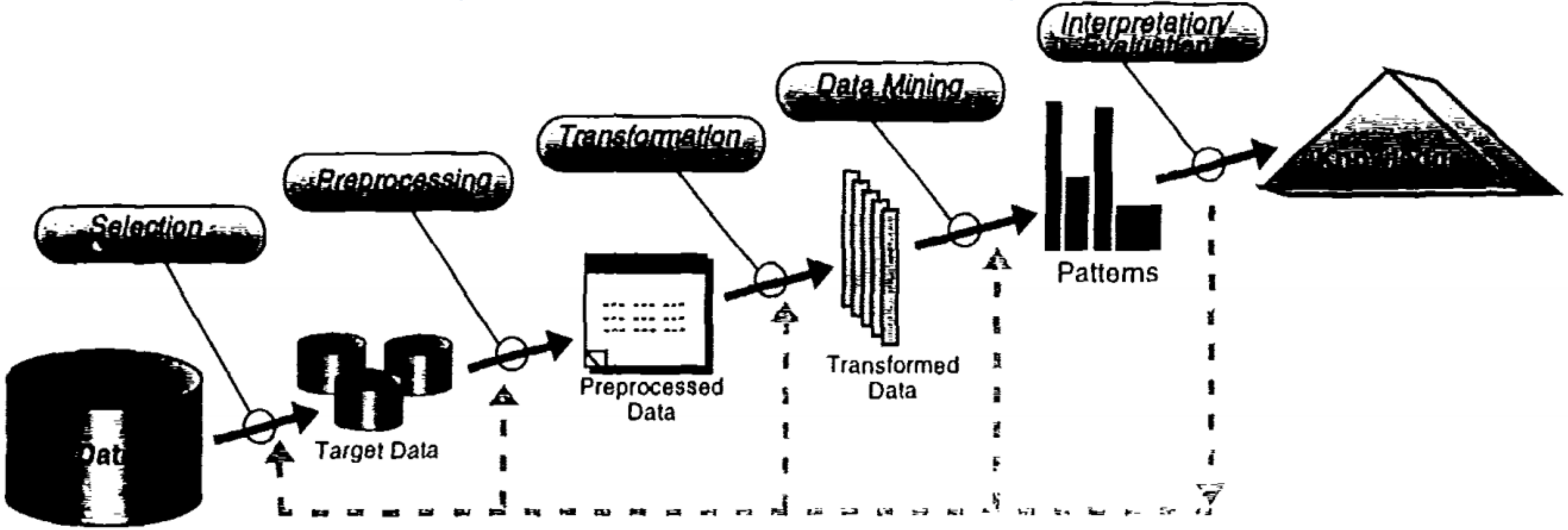


	Publication	h5-index	h5-median
1.	ACM SIGKDD International Conference on **Knowledge Discovery & Data Mining**	114	196
2.	IEEE Transactions on Knowledge and Data Engineering	88	147
3.	International Conference on Artificial Intelligence and Statistics	85	119
4.	ACM International Conference on Web Search and Data Mining	69	133
5.	Journal of Big Data	55	104
6.	IEEE International Conference on Data Mining	53	81
7.	**IEEE International Conference on Big Data**	52	93
8.	Knowledge and Information Systems	51	76
9.	ACM Conference on Recommender Systems	47	111
10.	Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery	47	89
11.	European Conference on Machine Learning and Knowledge Discovery in Databases	40	57
12.	ACM Transactions on Intelligent Systems and Technology (TIST)	38	72
13.	Data Mining and Knowledge Discovery	37	72
14.	SIAM International Conference on Data Mining (SDM)	35	60
15.	ACM Transactions on Knowledge Discovery from Data (TKDD)	35	53
16.	International Conference on Advances in Social Networks Analysis and Mining	34	65
17.	Big Data Mining and Analytics	31	39
18.	International Journal of Data Science and Analytics	30	52
19.	Social Network Analysis and Mining	30	46
20.	Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)	29	45

Knowledge Discovery and Data Mining

- “a unifying framework for Knowledge Discovery in Database (KDD)”
- links between data mining, knowledge discovery, and other related field

Selection, Preprocessing, Transformation, Data Mining, Interpretation/Evaluation



Data

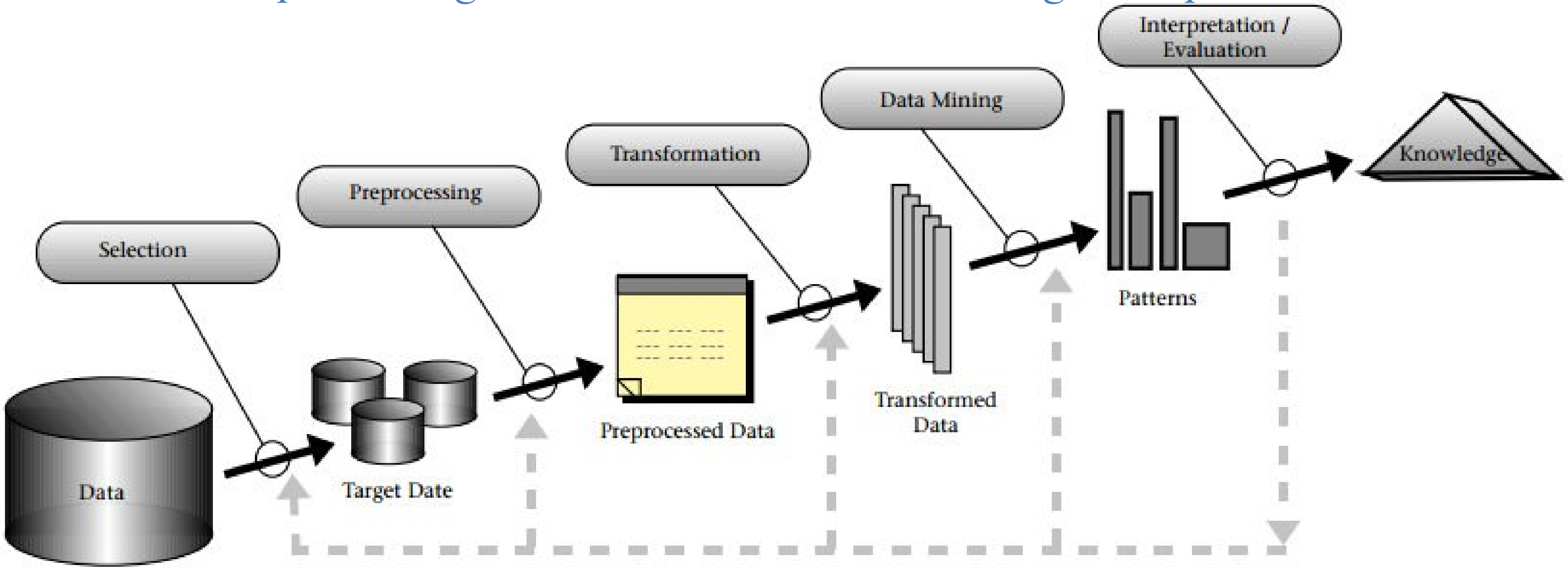
Figure 1: An overview of the steps comprising the KDD process.

Fayyad, Usama M., Gregory Piatetsky-Shapiro, and Padhraic Smyth.
"Knowledge Discovery and Data Mining: Towards a Unifying Framework." ***KDD*. 1996.**

Knowledge Discovery and Data Mining

- “a unifying framework for Knowledge Discovery in Database (KDD)”
- links between data mining, knowledge discovery, and other related field

Selection, Preprocessing, Transformation, Data Mining, Interpretation/Evaluation

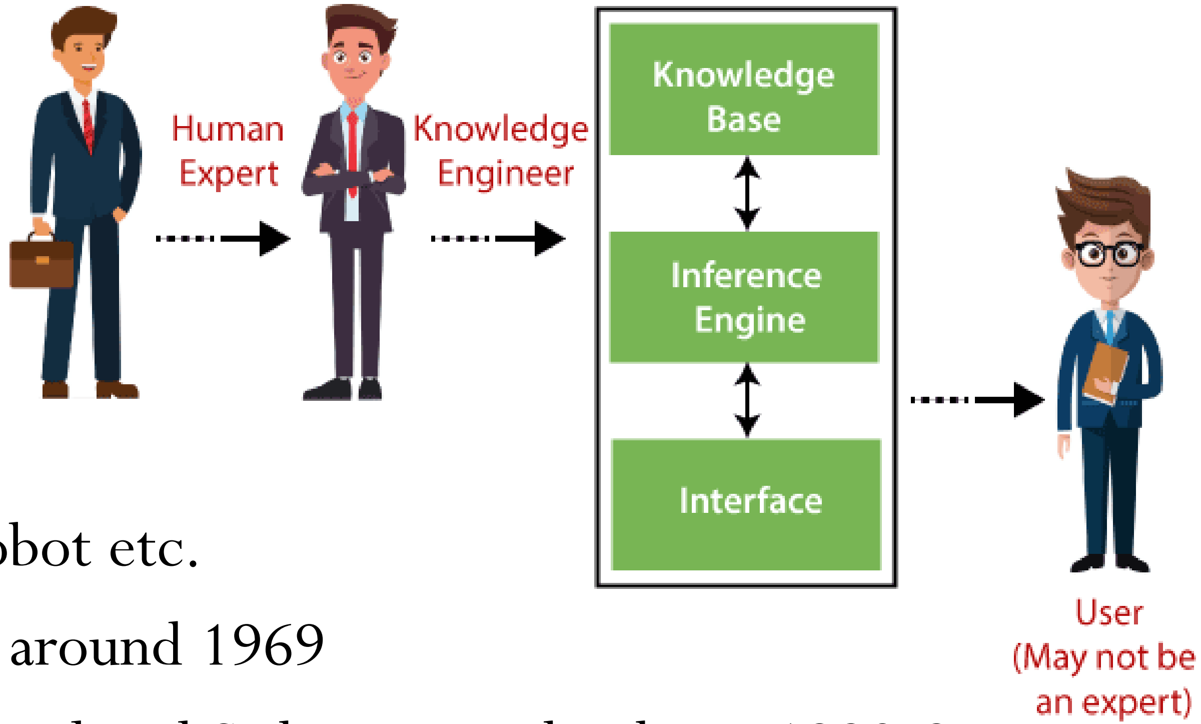


Data

https://www.kdnuggets.com/2016/03/data-science-process-rediscovered.html/2

Fayyad, Usama M., Gregory Piatetsky-Shapiro, and Padhraic Smyth.
"Knowledge Discovery and Data Mining: Towards a Unifying Framework." ***KDD*. 1996.**

Expert, System, and Knowledge Engineer



- Expert Systems (1960-74)
  - Explicit rules
- Playing Chess,
- Organic and Biology recommendations
- Solving word problem in Algebra
- Natural Language processing, Mobile Robot etc.
- Backpropagation based Neural Network around 1969
- Association Rule Mining by Rakesh Agrawal and Srikant Ramakrishnan 1993-95
- Big Files and Google File System in 1995-2005 by Larry Page, Sergey Brin and Sanjay Ghemawat, et al.

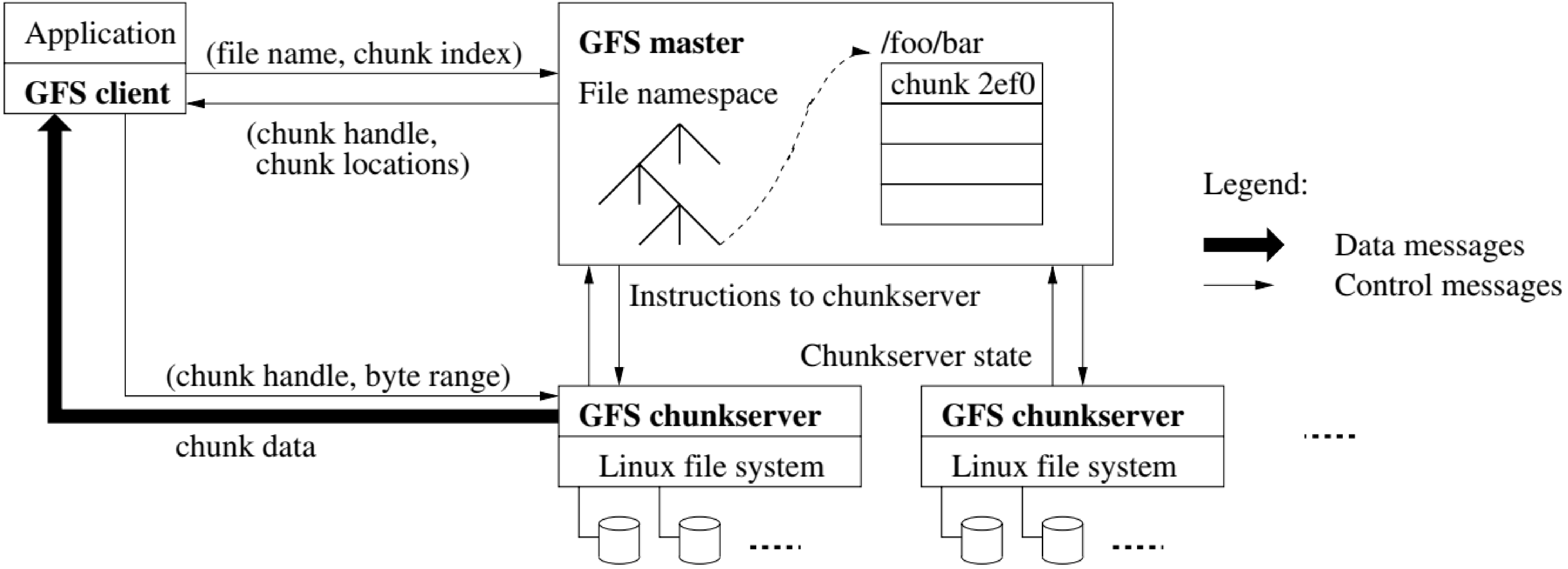
Big Files to Google File System (GFS)

- Earlier Google effort, "BigFiles", developed by Larry Page and Sergey Brin.
  - Supervisors: Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman, and Terry Winograd
- “Big File” was regenerated as “Google File System” by Sanjay Ghemawat, et al.
- Google File System (GFS)
  - “It is widely deployed within Google as the storage platform for the generation and processing of data used by Google service as well as research and development efforts that require large data sets.” 2003
  - “The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.” 2003

http://infolab.stanford.edu/~backrub/google.html

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

Google File System (GFS)



https://en.wikipedia.org/wiki/Google_File_System

https://sites.google.com/site/gfsassignmentwiki/home

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

GFS to HDFS

- Google File System (GFS) has similar open-source
  - “Hadoop Distributed File System (HDFS)”
- GFS and HDFS are distributed computing environment to process “Big Data”.
- GFS and HDFS are not implemented in the kernel of an operating system, but they are instead provided as a userspace library.
- GFS and HDFS properties
  - Files are divided into fixed-size chunks of 64 megabytes.
  - Scalable distributed file system for large distributed data intensive applications.
  - Provides fault tolerance.
  - High aggregate performance to a large number of clients.

https://en.wikipedia.org/wiki/Google_File_System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003.

Big Data

- Big data can be described by the following characteristics:
  - Volume: size large than terabytes and petabytes
  - Variety: type and nature, structured, semi-structured or unstructured
  - Velocity: speed of generation and processing to meet the demands
  - Veracity: the data quality and the data value
  - Value: Useful or not useful
- The main components and ecosystem of Big Data
  - Data Analytics: data mining, machine learning and natural language processing etc.
  - Technologies: Business Intelligence, Cloud computing & Databases etc.
  - Visualization: Charts, Graphs etc.

Architecture for Elasticity

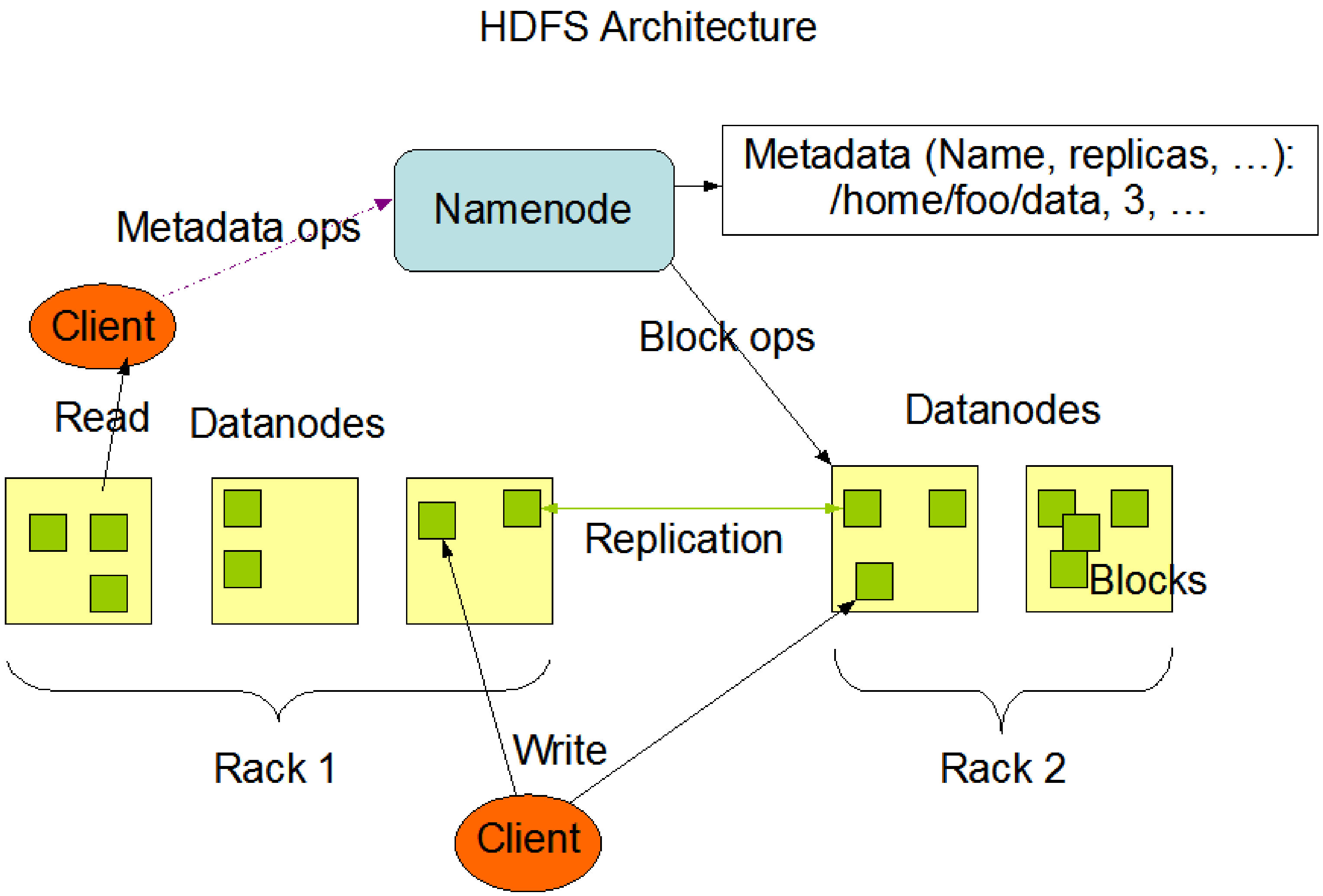
- **Vertical Scale-Up**
  - Keep on adding resources to a unit to increase computation power.
  - Process the job to single computation unit with high resources.
- **Horizontal Scale Out**
  - Keep on adding discrete resources for computation and make them behave as in converged unit.
  - Splitting job on multiple discrete machines, combine the output.
  - Distribute database.
- For HPC second option is better than first. Because Complexity and cost of first option is very high.

Hadoop

- “The Apache Hadoop project develops open source software for reliable, scalable, distributed computing.” Software library and a framework.
- Created by Doug Cutting Named on his son's stuffed elephant
- For distributed processing of large data sets across clusters of computers using simple programming models.
- Locality of reference
- Scalability: Scale up from single servers to thousands of machines,
  - Each offering local computation and storage
  - Program remains same for 10, 100, 1000,… nodes
  - Corresponding performance improvement
- Fault-tolerant file system:
  - Detect and handle failures and Delivering a highly-available.
- Hadoop Distributed File System (HDFS) Modeled on Google File system
- MapReduce for Parallel computation using
- Components – Pig, Hbase, HIVE, ZooKeeper

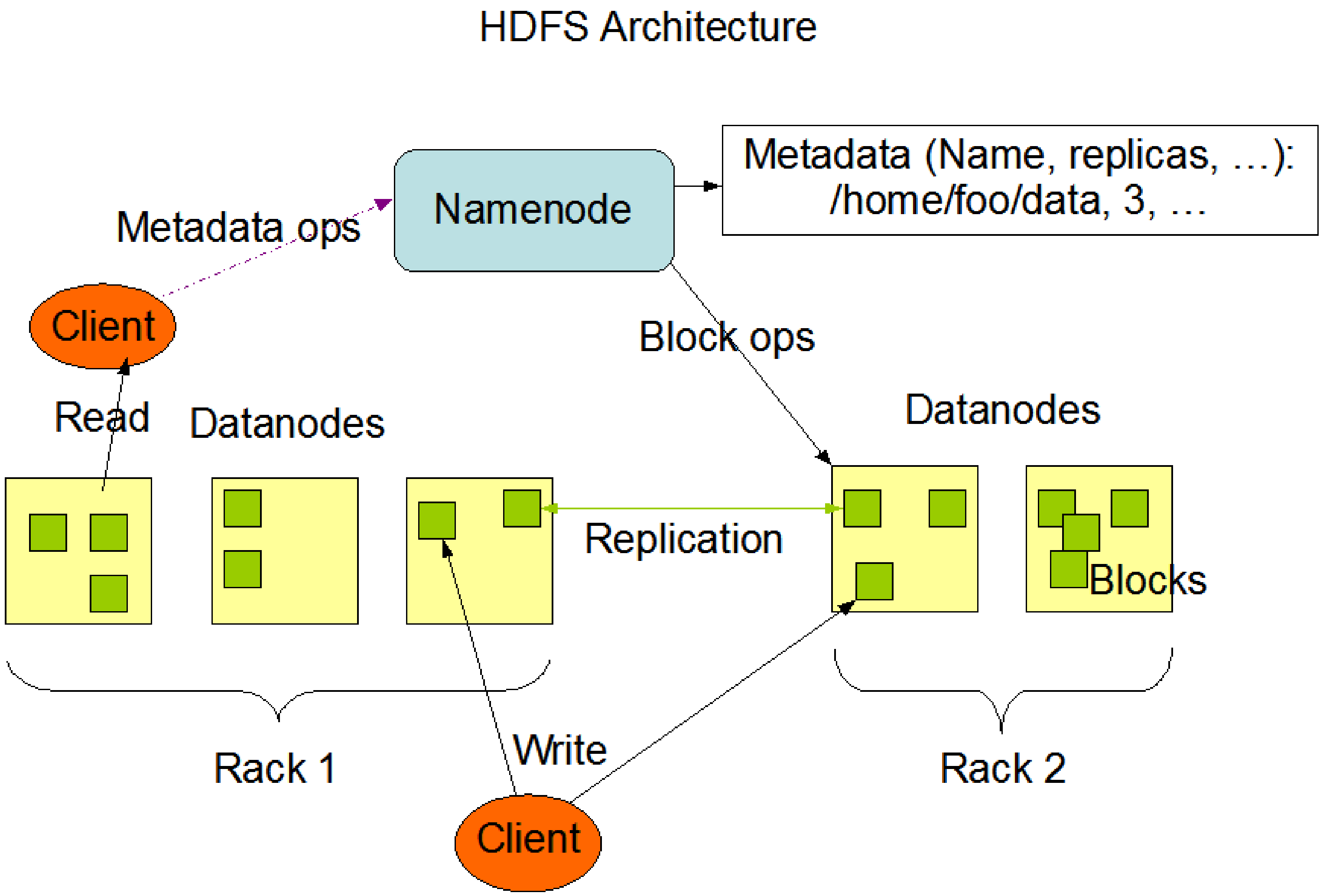
Hadoop Distributed File System (HDFS)

- HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients.
- In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.



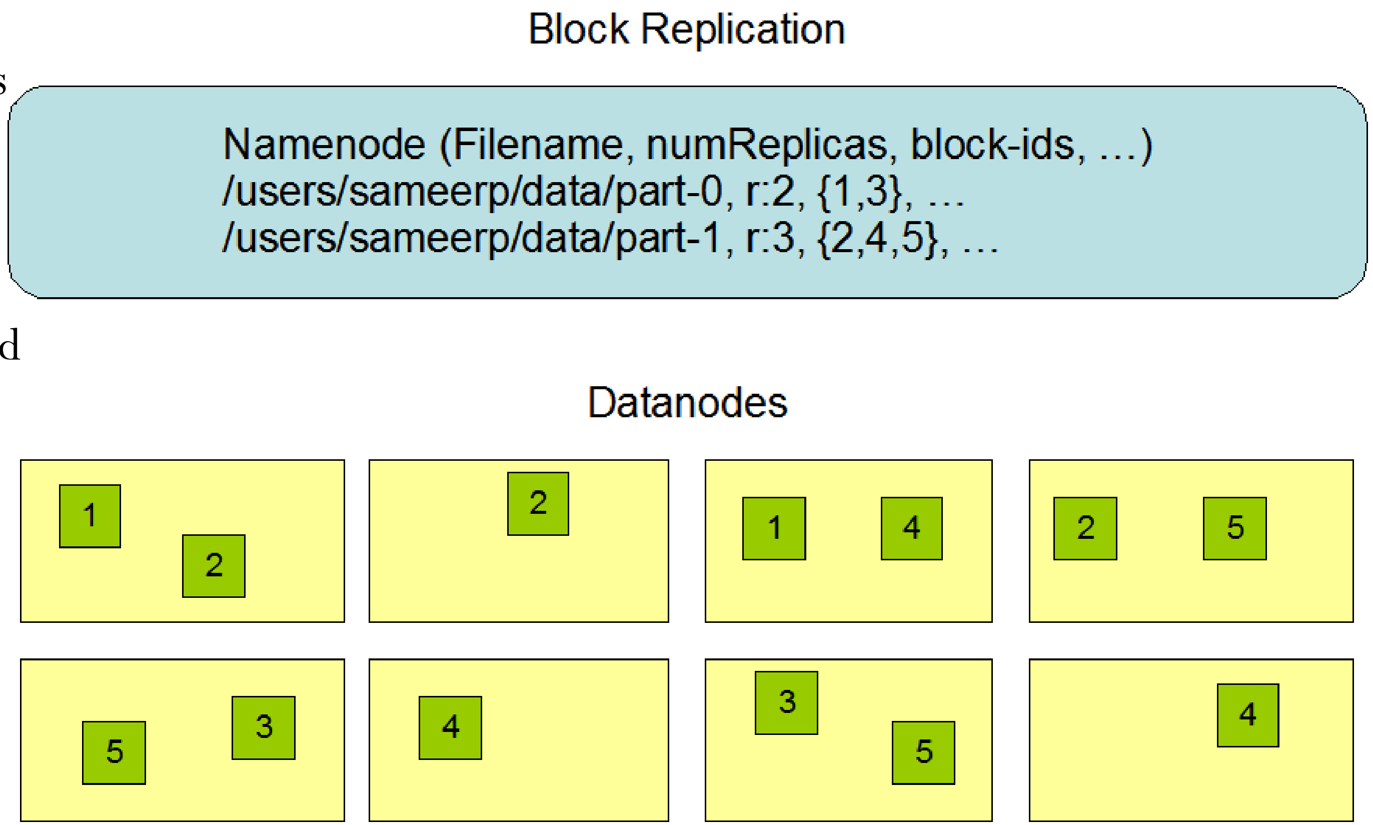
Hadoop Distributed File System (HDFS)

- Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
- The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.
- The DataNodes are responsible for serving read and write requests from the file system’s clients.
- The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode



Hadoop Distributed File System (HDFS)

- **Name Node:**
  - Manages File System - mapping files to blocks and blocks to data nodes. Maintains status of data nodes
  - Heartbeat: Datanode sends heartbeat at regular intervals, if heartbeat is not received, datanode is declared to be dead
  - Blockreport: DataNode sends list of blocks on it. Used to check health of HDFS
- **Data Node:**
  - Replicates (On Datanode failure, On Disk failure and On Block corruption)
  - Data integrity (Checksum for each block, Stored in hidden file)
  - Rebalancing- balancer tool (addition of new nodes, decommissioning and deletion of some files)

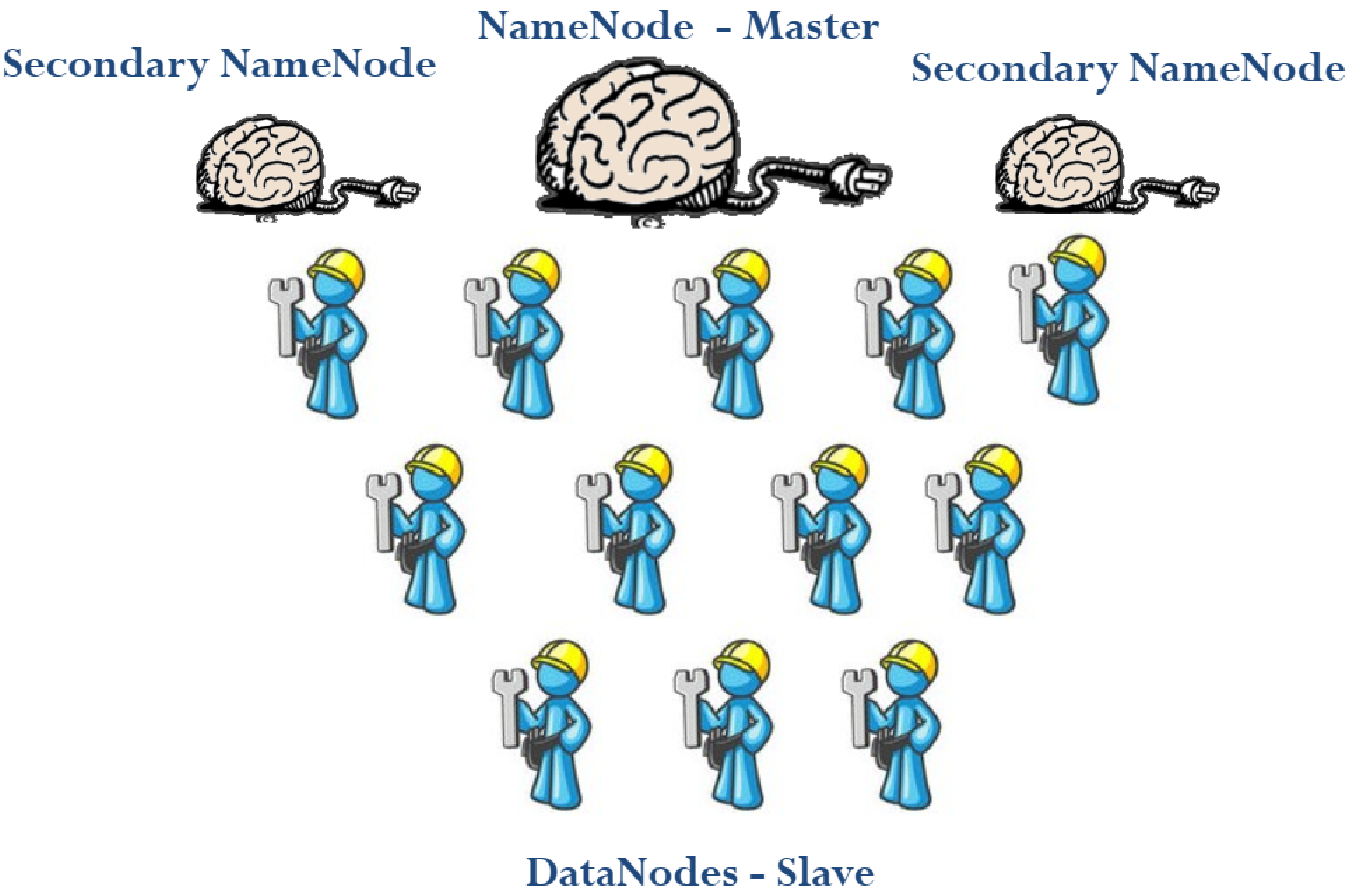


Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes

HDFS Force



Secondary NameNode

NameNode - Master

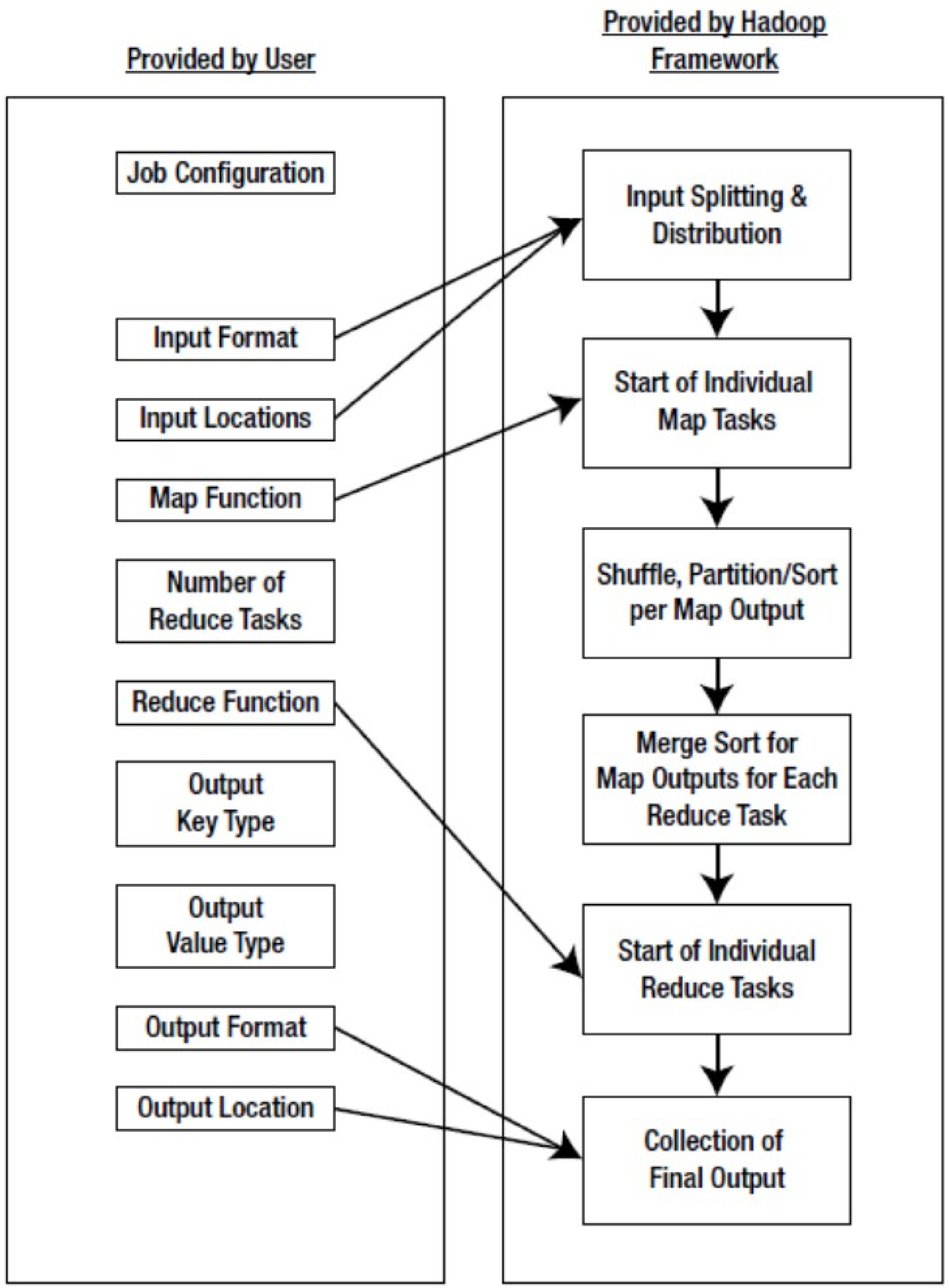
Secondary NameNode

DataNodes - Slave

Map Reduce

- Format of input- output (key, value)
  - Map: (k1, v1) → list (k2, v2)
  - Reduce: (k2, list v2) → list (k3, v3)

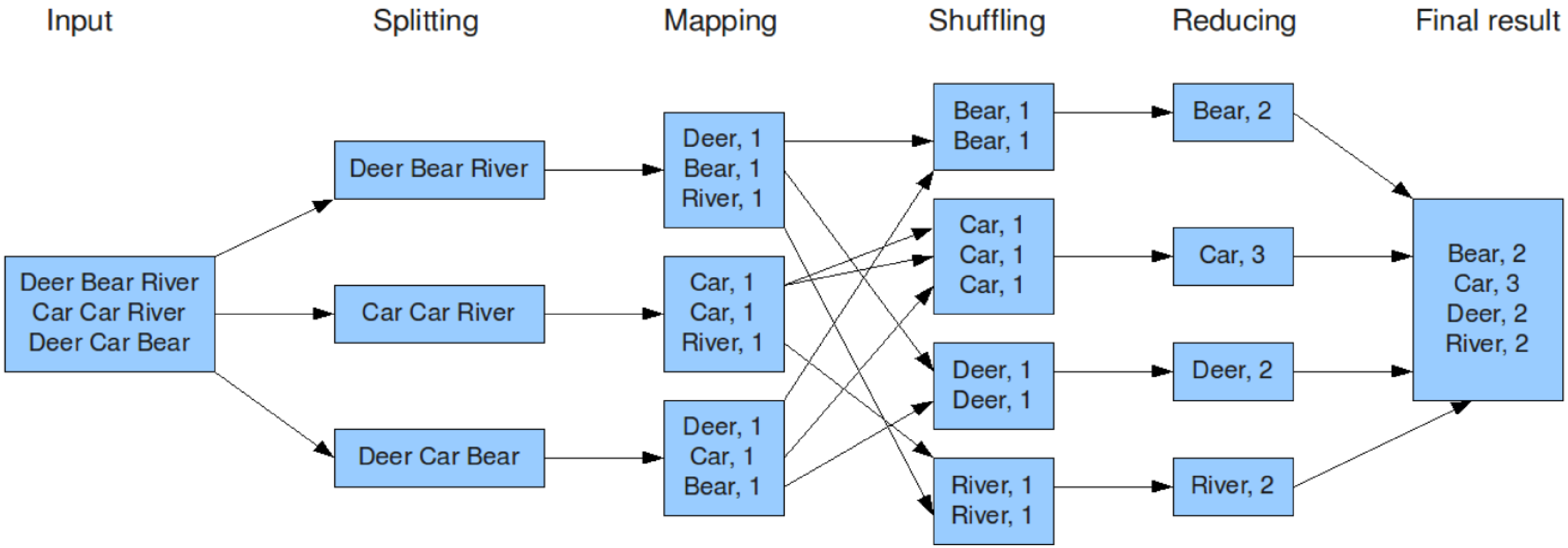
- MapReduce will not work for
  - Inter-process communication
  - Data sharing required
  - Example: Recursive functions



Provided by User
Provided by Hadoop Framework
Job Configuration
Input Format
Input Locations
Map Function
Number of Reduce Tasks
Reduce Function
Output Key Type
Output Value Type
Output Format
Output Location
Input Splitting & Distribution
Start of Individual Map Tasks
Shuffle, Partition/Sort per Map Output
Merge Sort for Map Outputs for Each Reduce Task
Start of Individual Reduce Tasks
Collection of Final Output

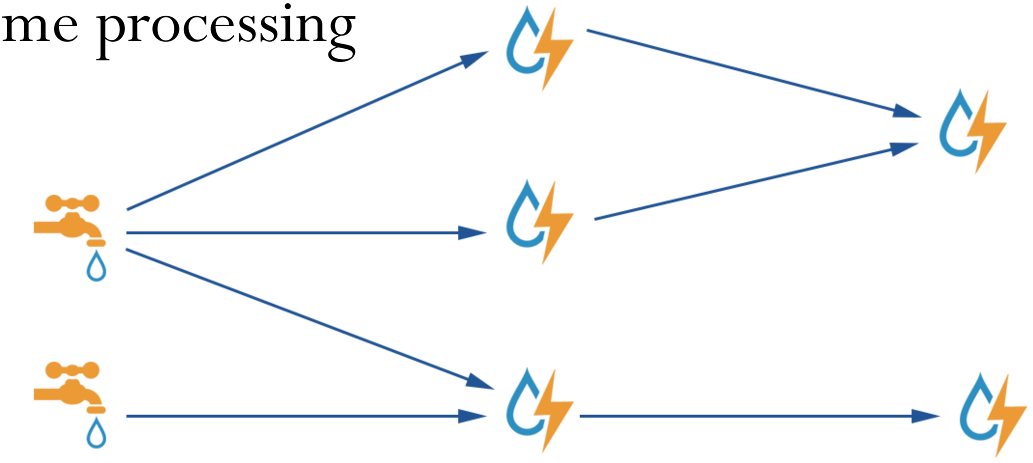
Word Count

- **Map:** Input lines of text to breaks them into words gives outputs for each word <key = word, value =1 >
- **Reduce:** Input <word, 1> output <word, + value>



Storm

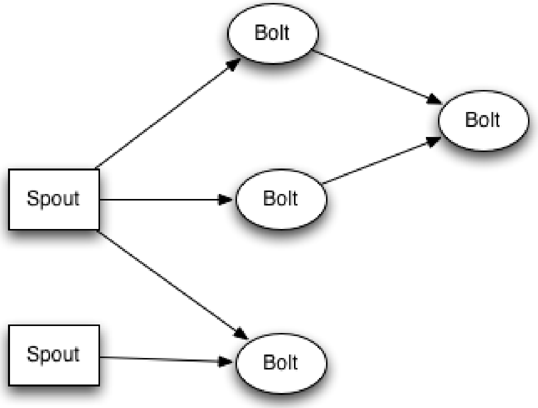
- Reliably for processing unbounded streams of data
- Hadoop for batch processing. Storm real-time processing
- Realtime analytics
- Online machine learning
- Continuous computation
- Distributed RPC.

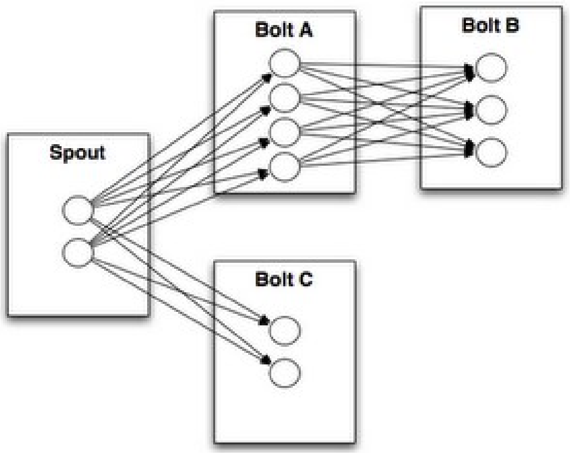


- A million tupples can be processed per second per node.
- It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Storm

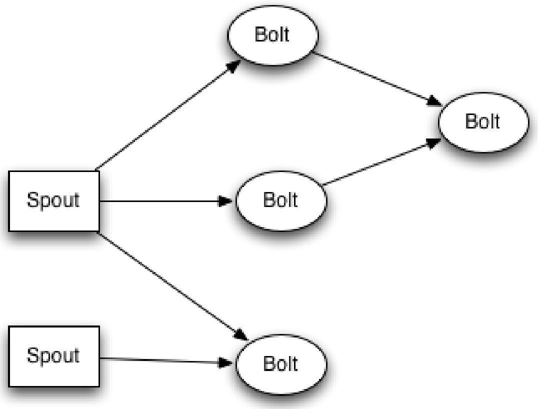
- Apache Storm is a free and open source distributed Realtime computation system.
- Apache Storm makes it easy to reliably process unbounded streams of data, doing for Realtime processing what Hadoop did for batch processing.
- Apache Storm integrates with the queueing and database technologies you already use.
- An Apache Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.





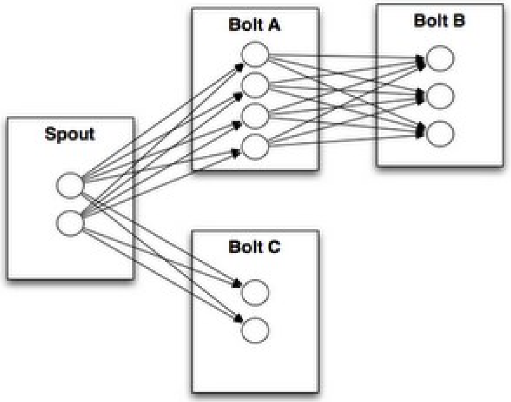
Storm

- Topologies: analogous to a MapReduce job. MapReduce job finishes, whereas a topology runs forever or until you kill it.
    - A topology is a graph of spouts and bolts that are connected with stream groupings.
- Streams: is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion.
    - Streams are defined with a schema that names the fields in the stream's tuples.
    - Schema are integers, longs, shorts, bytes, strings, doubles, floats, booleans, and byte arrays.
    - Every stream is given an id when declared.



Storm

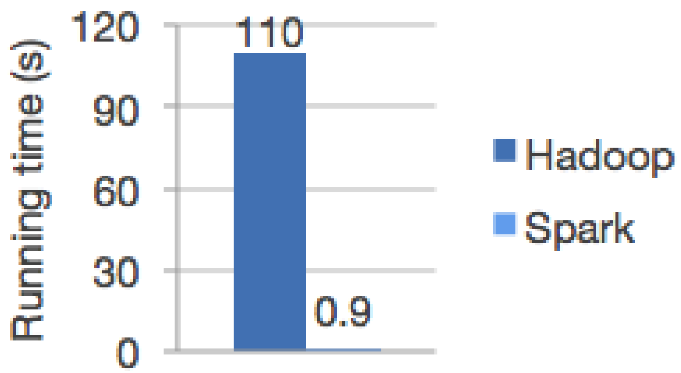
- **Spouts:** A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology.
  - **Reliable Spouts:** Replaying a tuple if it failed to be processed.
  - **Unreliable Spouts:** Forgets about the tuple as soon as it is emitted.



- **Bolts:** All processing in topologies is done in bolts.
  - Bolts can do filtering, functions, aggregations, joins, talking to databases, and more.
  - Bolts can do stream transformations into a new stream in a distributed and reliable way.
  - Complex stream transformations often requires multiple steps and thus multiple bolts.
  - For example, transform a stream of tweets into a stream of trending topics.

Apache Spark

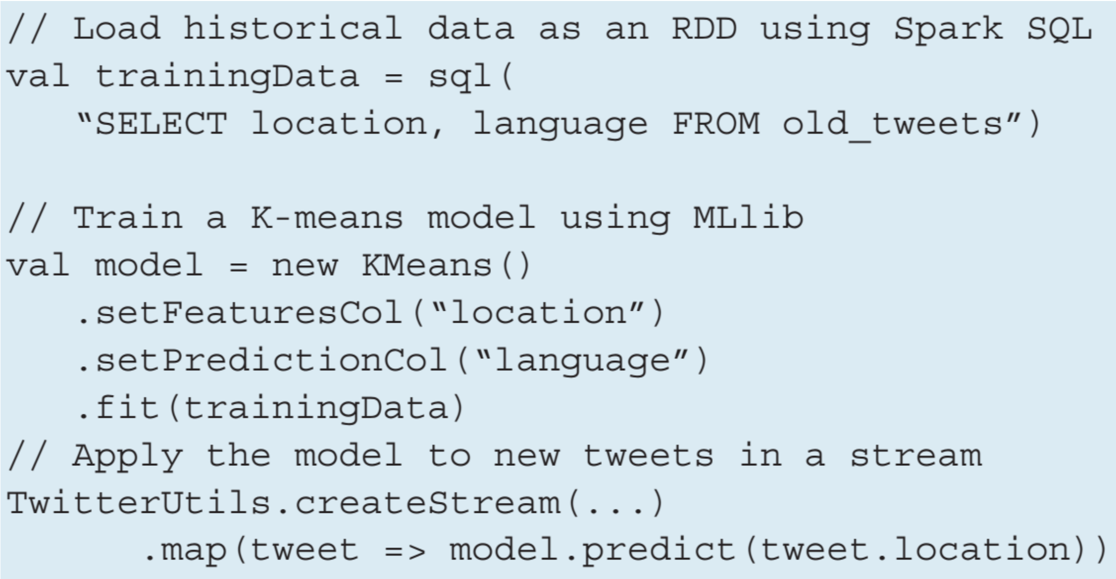
- Unified analytics engine for large-scale data processing.
- Speed: Run workloads 100x faster.
- Both batch and streaming data, using Directed Acyclic Graph (DAG) scheduler, a query optimizer, and a physical execution engine.
- Ease of Use: Write applications quickly in Java, Scala, Python, R, and SQL.
- Spark offers 80+ high-level operators to build parallel apps.



https://spark.apache.org/

Spark: Unified Big Data Analytics

- New applications of Big data workloads on Unified Engine of
  - Streaming, Batch, and Interactive.
- Composability in programming libraries for big data and encourages development of interoperable libraries
- Combining the SQL, machine learning, and streaming libraries in Spark



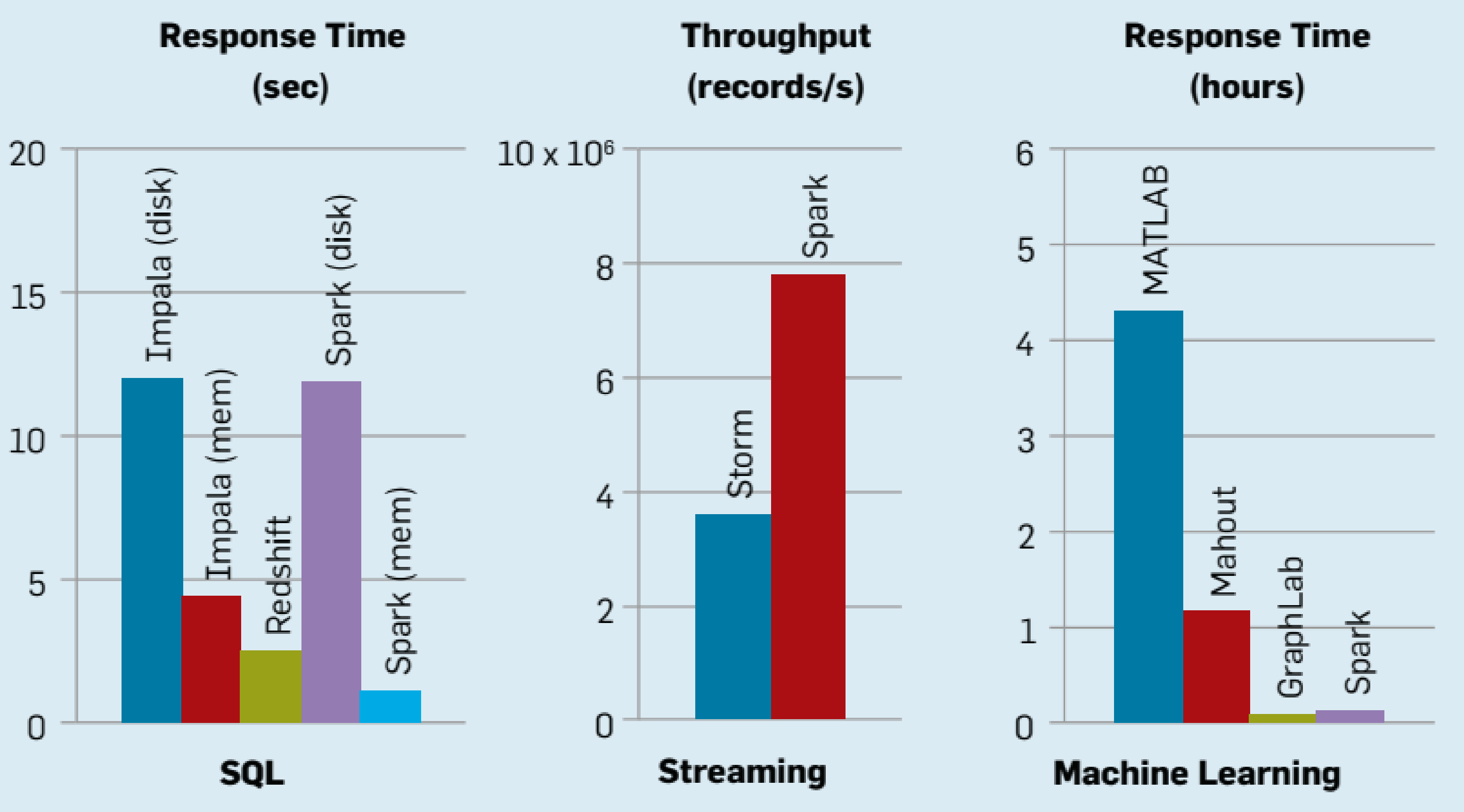
```
// Load historical data as an RDD using Spark SQL
val trainingData = sql(
   “SELECT location, language FROM old_tweets”)

// Train a K-means model using MLlib
val model = new KMeans()
   .setFeaturesCol(“location”)
   .setPredictionCol(“language”)
   .fit(trainingData)
// Apply the model to new tweets in a stream
TwitterUtils.createStream(...)
      .map(tweet => model.predict(tweet.location))
```

Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." *Communications of the ACM* 59.11 (2016): 56-65.

Spark: Unified Big Data Analytics

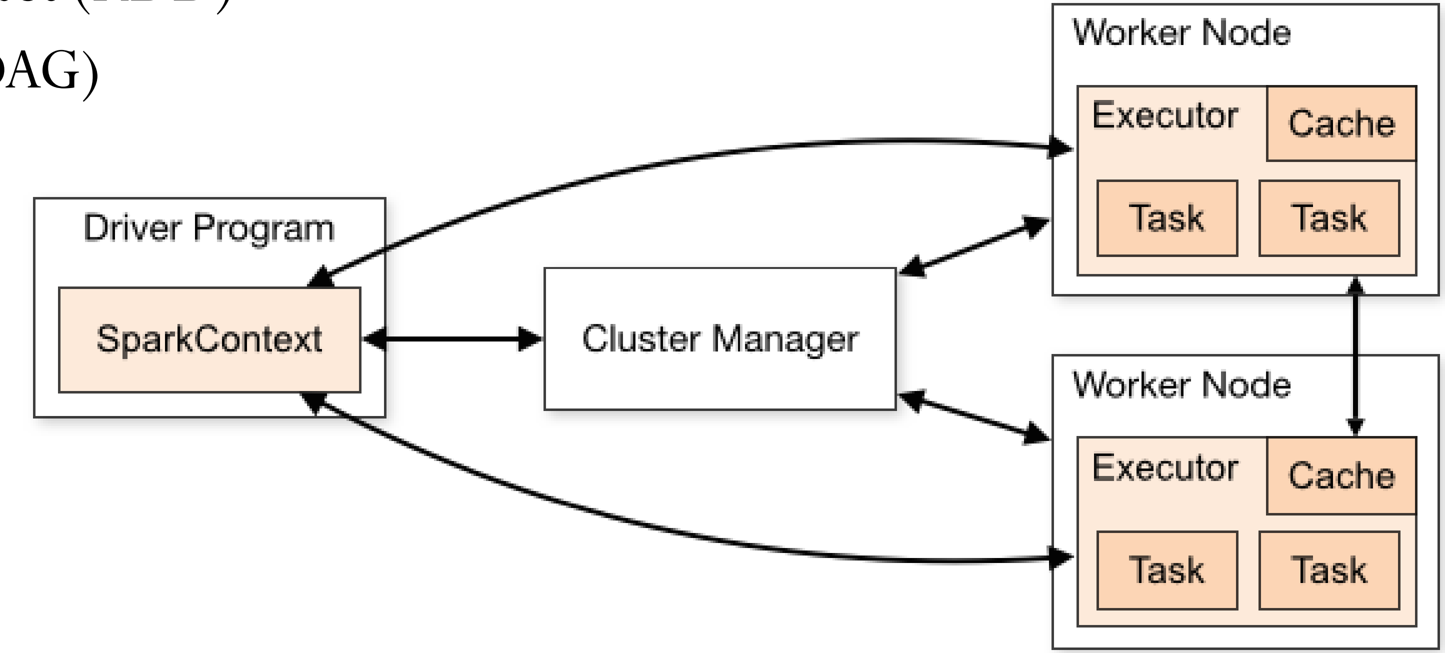
- Spark has MapReduce programming model with extended data-sharing abstraction called "Resilient Distributed Datasets," or RDDs.



Zaharia, Matei, et al. "Apache spark: a unified engine for big data processing." *Communications of the ACM* 59.11 (2016): 56-65.

Spark Architecture

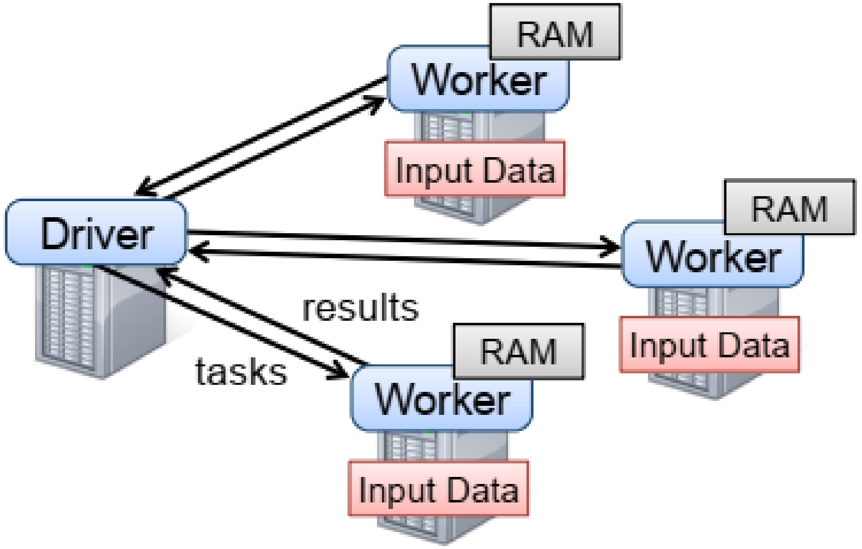
- Spark works on the popular Master-Slave architecture.
- Cluster works with a single master and multiple slaves.
- The Spark architecture depends upon two abstractions:
  - Resilient Distributed Dataset (RDD)
  - Directed Acyclic Graph (DAG)



https://spark.apache.org/docs/latest/cluster-overview.html

Resilient Distributed Datasets (RDD)

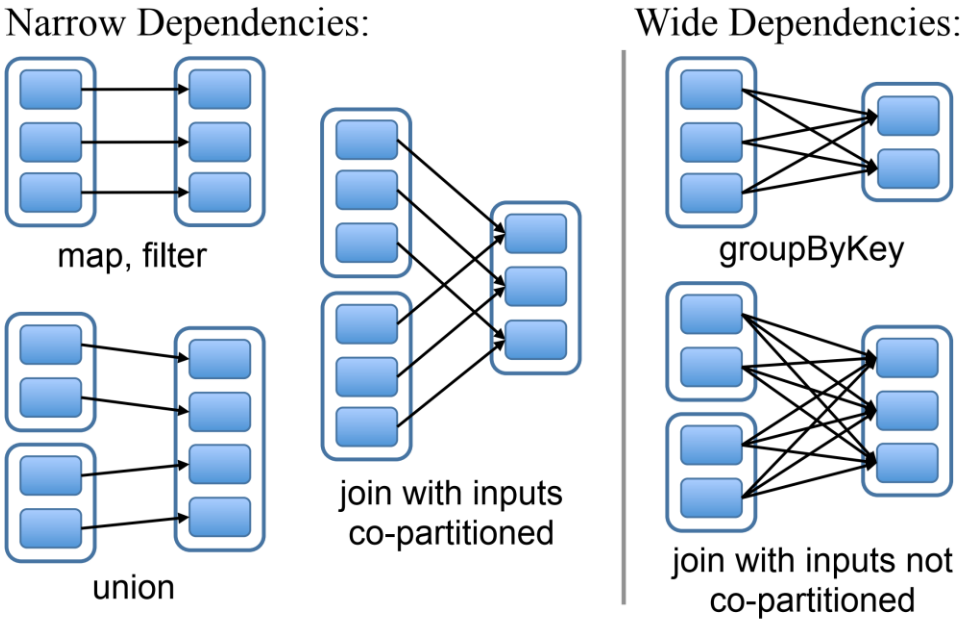
- A distributed memory abstraction: perform in-memory computations on large clusters
- Keeping data in memory can improve performance
- Spark runtime: Driver program launches multiple workers that read data blocks from a distributed file system and can persist computed RDD partitions in memory.



Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Resilient Distributed Datasets (RDD)

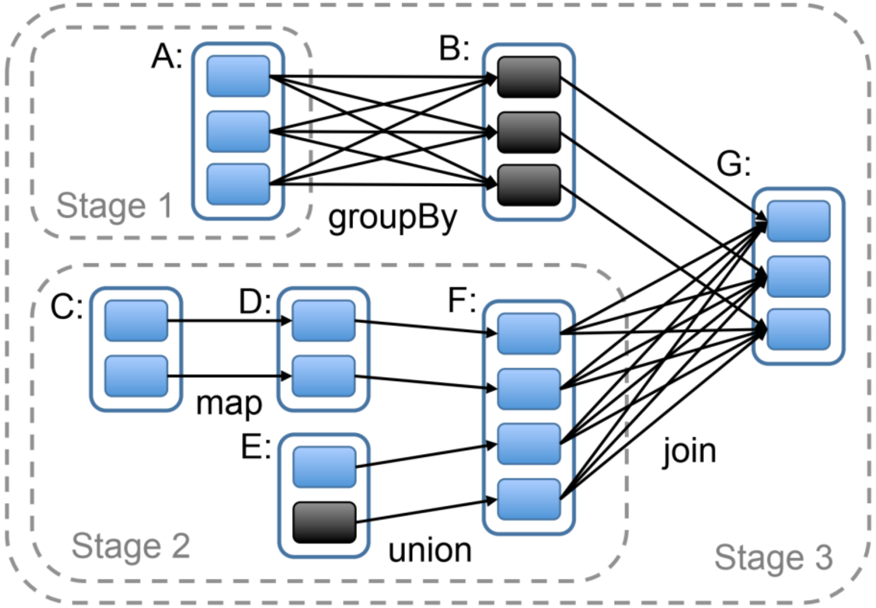
- Each box is an RDD, with partitions as shaded rectangles.
- *narrow* dependencies, where each partition of the parent RDD is used by at most one partition of the child RDD,
- *wide* dependencies, where multiple child partitions may depend on it.


Narrow Dependencies:
map, filter
union
join with inputs co-partitioned
Wide Dependencies:
groupByKey
join with inputs not co-partitioned

Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Resilient Distributed Datasets (RDD)

- Direct Acyclic Graph (DAG) to perform a sequence of computations
- Each node is an RDD partition,
- Run an action (*e.g.*, *count* or *save*) on an RDD, the scheduler examines that RDD's lineage graph to build a DAG of *stages* to execute.
- Each stage contains with narrow dependencies
- The boundaries of the stages are the shuffle operations required for wide dependencies.
- Scheduler computes missing partitions until it computed RDD.



Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012.

Apache Spark

- Combine SQL, streaming, and complex analytics. Spark libraries
  - SQL and DataFrames,
  - MLlib for machine learning,
  - GraphX, and
  - Spark Streaming.
- Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud.
- Run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes.
- Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

https://spark.apache.org/

Spark Code Example

- Word Count

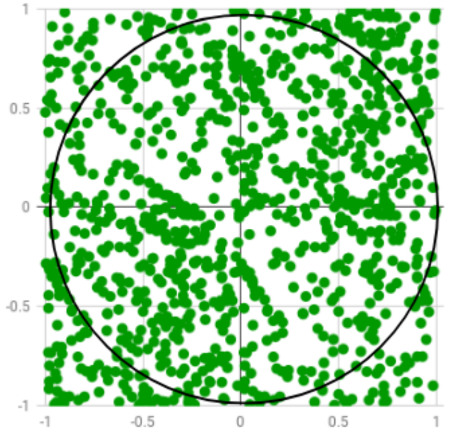
```
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

- Pi Estimation

```
List<Integer> l = new ArrayList<>(NUM_SAMPLES);
for (int i = 0; i < NUM_SAMPLES; i++) {
  l.add(i);
}

long count = sc.parallelize(l).filter(i -> {
  double x = Math.random();
  double y = Math.random();
  return x*x + y*y < 1;
}).count();
System.out.println("Pi is roughly " + 4.0 * count / NUM_SAMPLES);
```

$$\text{Pi} = 4\times\frac{\textit{number of random point inside the circle}}{\textit{number of random point inside the square}}$$



Spark SQL

- Working with structured data.
- Integrated: SQL queries with Spark programs.

```
results = spark.sql("SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Apply functions to results of SQL queries.

- Uniform Data Access: Connect to data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC.

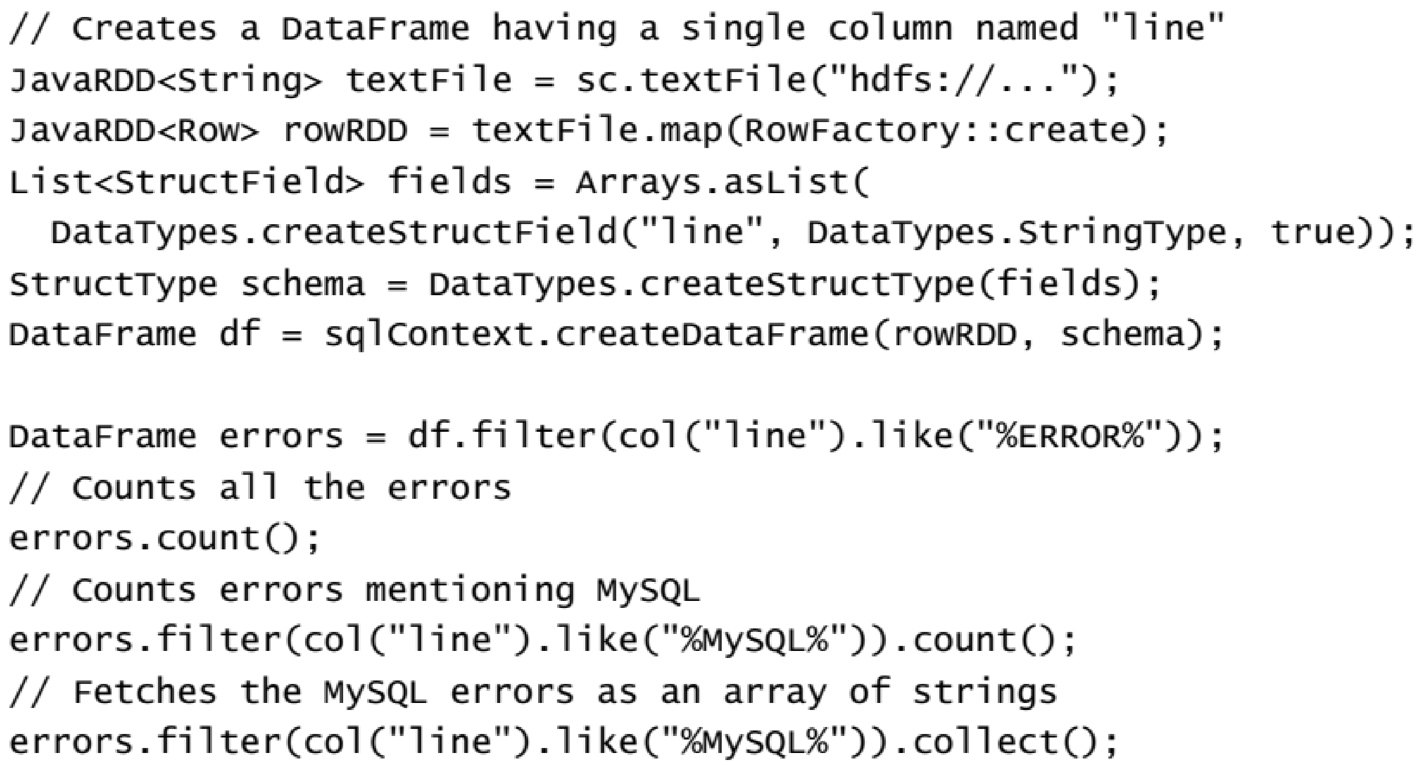
```
spark.read.json("s3n://...").registerTempTable("json")
results = spark.sql("""SELECT * FROM people JOIN json ...""")
```

- Query and join different data sources
- Hive Integration Spark HiveQL
- Standard Connectivity: Connect through JDBC or ODBC.
- Business intelligence tools to query big data.

https://spark.apache.org/sql/

DataFrame API Examples

- Collection of data organized into named columns
- Use DataFrame API to perform various relational operations
- Automatically optimized by Spark’s built-in optimizer



```
// Creates a DataFrame having a single column named "line"
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaRDD<Row> rowRDD = textFile.map(RowFactory::create);
List<StructField> fields = Arrays.asList(
  DataTypes.createStructField("line", DataTypes.StringType, true));
StructType schema = DataTypes.createStructType(fields);
DataFrame df = sqlContext.createDataFrame(rowRDD, schema);

DataFrame errors = df.filter(col("line").like("%ERROR%"));
// Counts all the errors
errors.count();
// Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count();
// Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect();
```

Spark Streaming

- Build scalable fault-tolerant streaming applications.
- Write streaming jobs -- Same Way -- Write batch jobs
  - Counting tweets on a sliding window

```
TwitterUtils.createStream(...)
.filter(_.getText.contains("Spark"))
.countByWindow(Seconds(5))
```

- Reuse the same code for batch processing
  - Find words with higher frequency than historic data:

```
stream.join(historicCounts).filter {
  case (word, (curCount, oldCount)) =>
    curCount > oldCount
}
```

Batch processing takes N unit time to process M unit of data	*Batch processing* **takes N+x unit time** to process **M+y unit of data**
Stream processing takes N unit time to process M unit of data	*Stream processing* **takes x unit time** to process **M+y unit of data**

https://spark.apache.org/streaming/

Spark GraphX

- Spark's API for graphs and graph-parallel computation

```
graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://...")
graph2 = graph.joinVertices(messages) {
  (id, vertex, msg) => ...
}
```

- Fast Speed for graph algorithms
- GraphX graph algorithms
  - PageRank
  - Connected components
  - Label propagation
  - SVD++
  - Strongly connected components
  - Triangle count

- The **joinVertices** operator joins the vertices with the input RDD and returns a new graph with the vertex properties obtained by applying the user defined map function to the result of the joined vertices.
- Vertices without a matching value in the RDD retain their original value.

https://spark.apache.org/graphx/

Spark MLlib

- Spark's scalable machine learning library
- Spark MLlib algorithms
  - Classification: logistic regression, naive Bayes,…
  - Regression: generalized linear regression, survival regression,…
  - Decision trees, Random forests, and Gradient-boosted trees
  - Recommendation: Alternating Least Squares (ALS)
  - Clustering: K-means, Gaussian mixtures (GMMs),…
  - Topic modeling: Latent Dirichlet Allocation (LDA)
  - **Frequent itemsets, Association rules, and Sequential pattern mining**

https://spark.apache.org/graphx/

Apriori algorithm: Association Rule Mining

- Let I = $\{i_1, i_2, \ldots.\ i_m\}$ be a set of literals, called items.
- *Support* of a rule X →Y is the percentage of transactions that contain both X and Y.
- *Confidence* of a rule is percentage the if-then statements (X →Y) are found true
- Find all rules that satisfy a user-specified *minimum support* and *minimum confidence*

TID	Transaction Items
1	Bread, Jelly, PeanutButter
2	Bread, PeanutButter
3	Bread, Milk, PeanutButter
4	Beer, Bread
5	Beer, Milk

{Bread} → {PeanutButter} (Sup = 60%, Conf = 75%)

{PeanutButter} → {Bread} (Sup = 60%, Conf = 100%)

{Beer} → {Bread} (Sup = 20%, Conf = 50%)

{PeanutButter} → {Jelly} (Sup = 20%, Conf = 33.33%)

{Jelly} → {PeanutButter} (Sup = 20%, Conf = 100%)

{Jelly} → {Milk} (Sup = 0%, Conf = 0%)

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." SIG-MOD. 1993.
Ramakrishnan Srikant, and Rakesh Agrawal. "Mining Generalized Association Rules.“VLDB 1995.

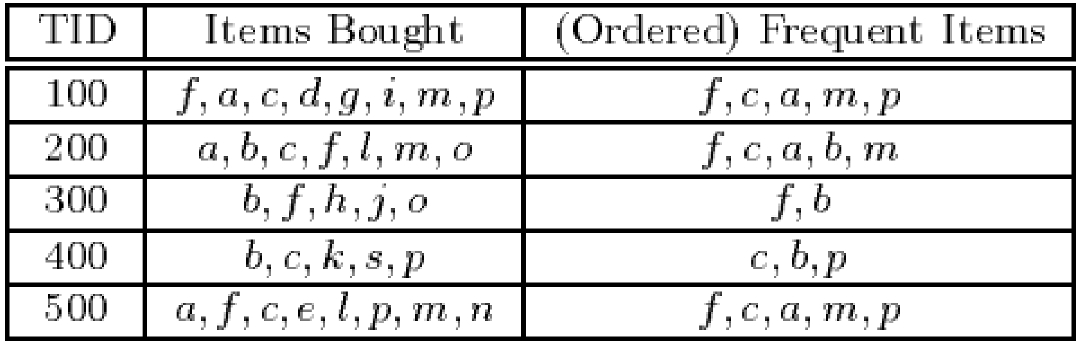
Apriori algorithm: Association Rule Mining

- Let I = $\{i_1, i_2, \ldots.\ i_m\}$ be a set of literals, called items.
- *Support* of a rule X →Y is the percentage of transactions that contain both X and Y.
- *Confidence* of a rule is percentage the if-then statements (X →Y) are found true
- Find all rules that satisfy a user-specified *minimum support* and *minimum confidence*
  - 100% of transactions that purchase *PeanutButter* (antecedent) also purchase *Bread* (consequent). The number 100% is the confidence factor of the rule
    - [PeanutButter] → [Bread] (Sup = 60%, Conf = 100%)
  - 98% of customers who purchase *Tires* and *Auto accessories* also buy some *Automotive services*; here 98% is called the confidence of the rule.
    - [*Auto Accessories*], [*Tires*] → [*Automotive Services*] 98%

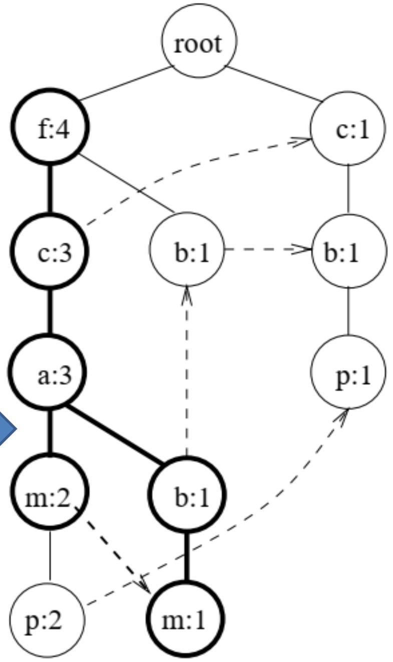
Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." SIG-MOD. 1993.
Ramakrishnan Srikant, and Rakesh Agrawal. "Mining Generalized Association Rules.“VLDB 1995.

FP-Growth for recommendation

- “FP” stands for Frequent Pattern in a Dataset of transactions
  1. calculate item frequencies and identify frequent items,
  2. a suffix tree (FP-tree) structure to encode transactions, and
  3. frequent itemsets can be extracted from the FP-tree.
- Input: Transaction database
- Intermediate Output: FP-Tree
- Output: {f, c, a → a, m p}, {f, c, a → b, m}



TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p



Han Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." *ACM SIGMOD Record* 29.2 (2000): 1-12.

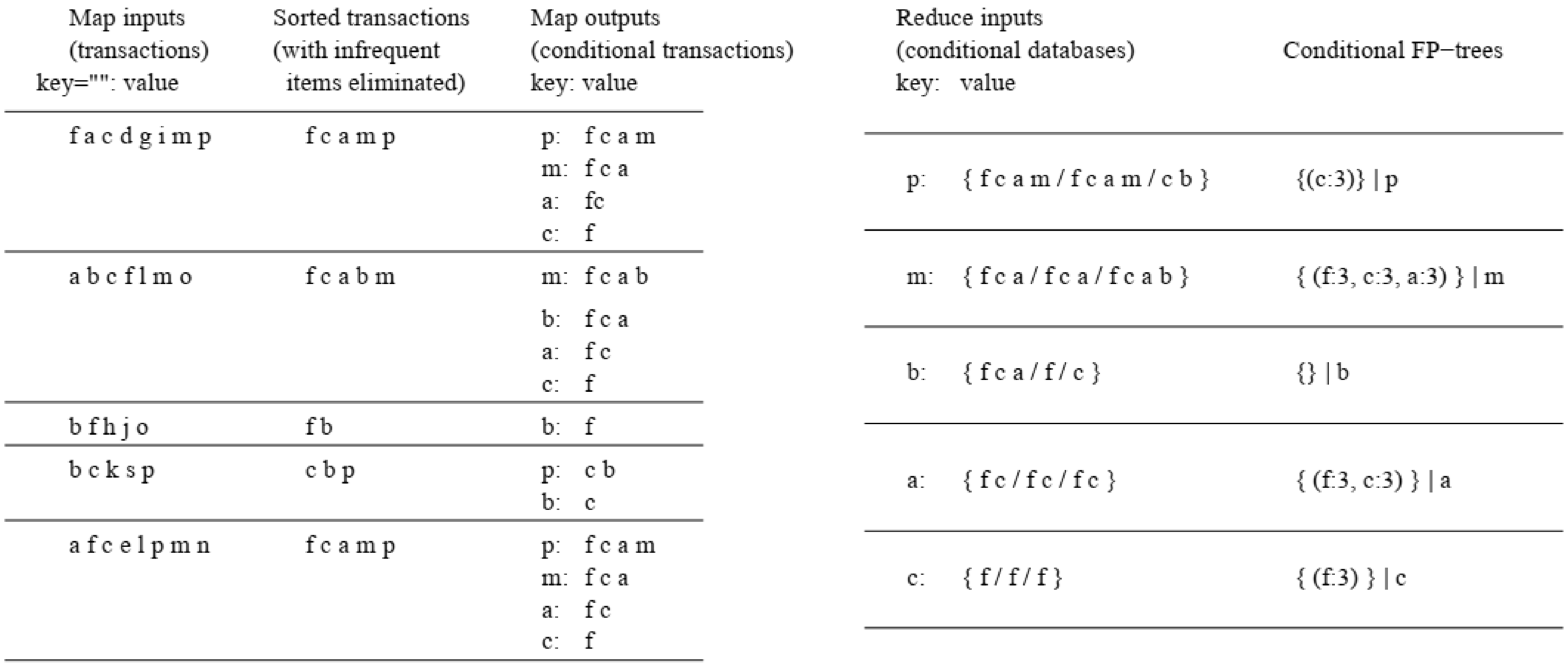
PFP: Parallel FP-Growth

- In Spark ML-Library (MLLib), a parallel version of FP-growth called PFP: Parallel FP-Growth
- PFP distributes the work of growing FP-trees based on the suffixes of transactions.
- More scalable than a single-machine implementation.
- PFP partitions computation, where each machine executes an independent group of mining tasks

Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation."
Proceedings of the 2008 ACM Conference on Recommender systems. 2008.

PFP: Parallel FP-Growth

Example of MapReduce FP-Growth: Five transactions composed of lower-case alphabets representing items



Map inputs (transactions) key="": value	Sorted transactions (with infrequent items eliminated)	Map outputs (conditional transactions) key: value
facdgimp	fcamp	p: fcam m: fca a: fc c: f
abcflmo	fcabm	m: fcab b: fca a: fc c: f
bfhjo	fb	b: f
bcksp	cbp	p: cb b: c
afcelpmn	fcamp	p: fcam m: fca a: fc c: f

Reduce inputs (conditional databases) key: value	Conditional FP−trees
p: {fcam/fcam/cb}	{(c:3)} | p
m: {fca/fca/fcab}	{ (f:3, c:3, a:3) } | m
b: {fca/f/c}	{} | b
a: {fc/fc/fc}	{ (f:3, c:3) } | a
c: {f/f/f}	{ (f:3) } | c

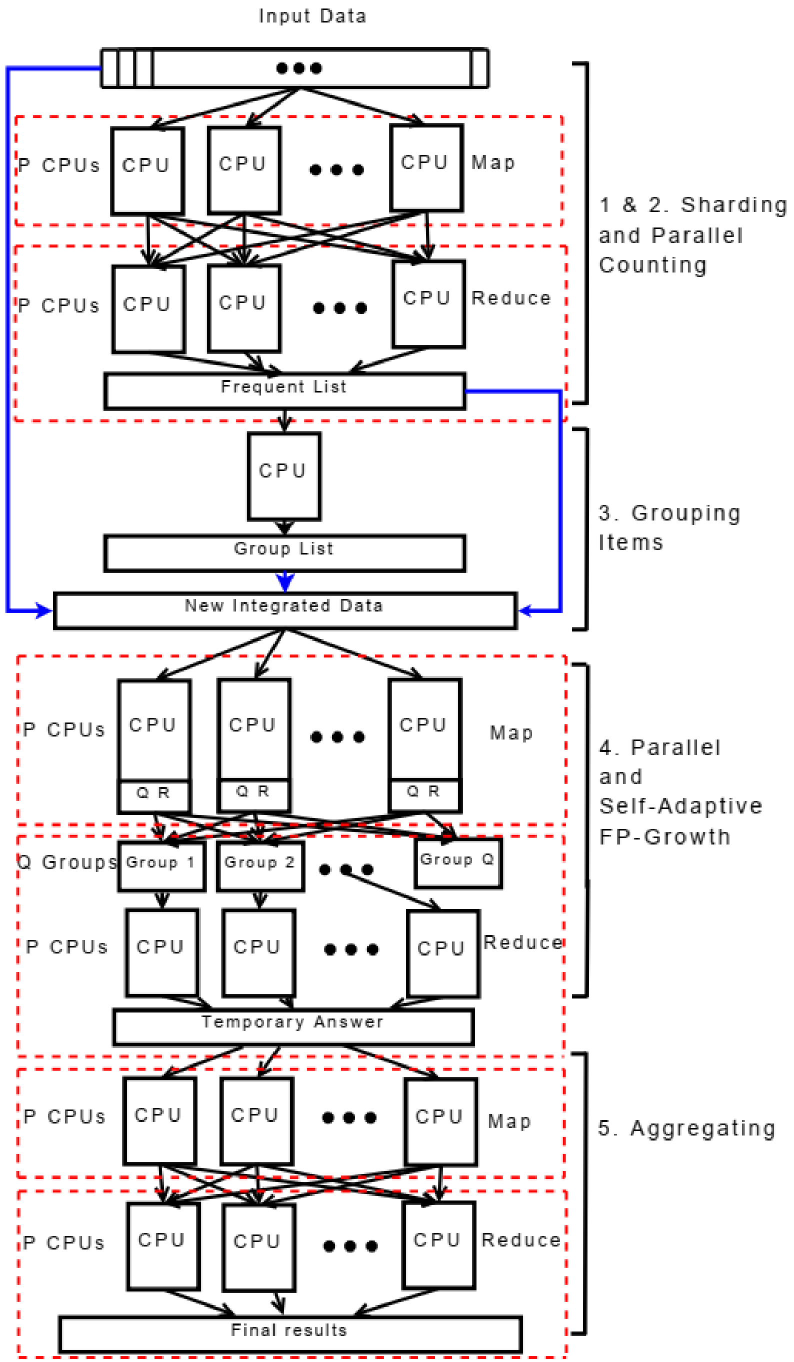
Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *Proceedings of the 2008 ACM Conference on Recommender systems*. 2008.

- **Sharding:** Divide DB into successive parts and storing the parts (as a Shard) on P different computers.
- **Parallel Counting:** MapReduce counts the support of all items that appear in DB. Each mapper inputs one shard of DB. The result is stored in F-list.
- **Grouping Items:** Dividing all the items on F-List into Q groups of a list (G-list).
- **Parallel FP-Growth:** A MapReduce

  - Mapper: Each mapper uses a Shard. It reads a transaction in the G-list and outputs one or more key-value pairs, where each key is a *group-id* and value is a **group-dependent transaction**.

  - For each *group-id*, the MapReduce groups all group-dependent transactions into a shard.

  - Reducer: Each reducer processes one or more group-dependent Shard. For each shard, a reducer builds a local FP-Tree and discover patterns.
- **Aggregating:** Aggregate the results generated as final result.



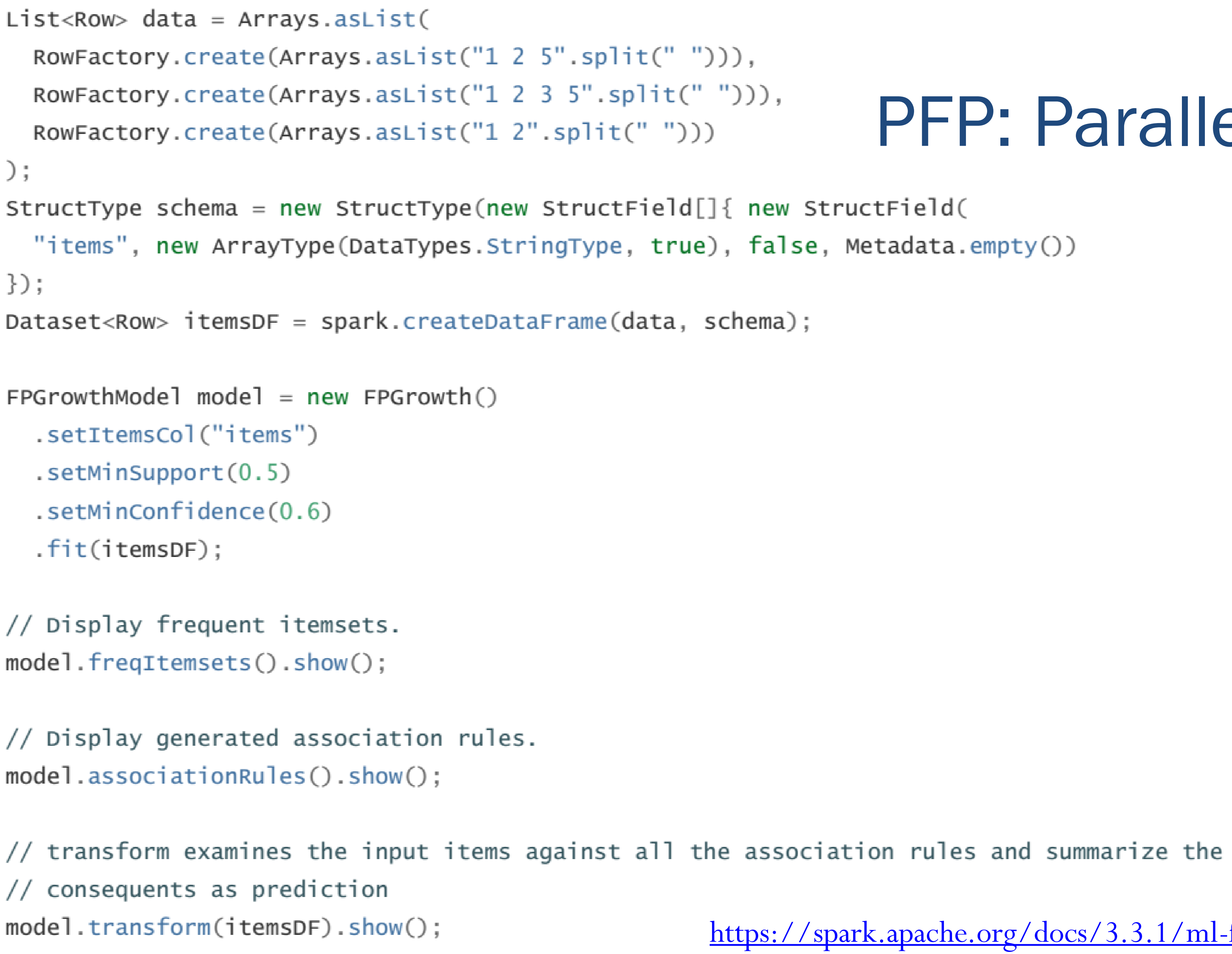
Li, Haoyuan, et al. "PFP: Parallel FP-Growth for query recommendation." *Proceedings of the 2008 ACM Conference on Recommender systems*. 2008.

PFP: Parallel FP-Growth

- FP-Growth implementation takes the following (hyper-)parameters
  - minSupport: the minimum support for an itemset to be identified as frequent e.g., if an item appears 3 out of 6 transactions, it has a support of 3/6=0.5.
  - minConfidence: minimum confidence for generating Association Rule e.g., if in the transactions itemset X appears 5 times, X and Y co-occur only 3 times, the confidence for the rule X => Y is then 3/5 = 0.6.
  - numPartitions: the number of partitions used to distribute the work.
- FP-Growth model provides:
  - freqItemsets: frequent itemsets in the format of DataFrame("items"[Array], "freq"[Long])
  - associationRules: association rules generated with confidence above minConfidence, in the format of DataFrame("antecedent"[Array], "consequent"[Array], "confidence"[Double]).

https://spark.apache.org/docs/3.3.1/ml-frequent-pattern-mining.html

PFP: Parallel FP-Growth



```
List<Row> data = Arrays.asList(
  RowFactory.create(Arrays.asList("1 2 5".split(" "))),
  RowFactory.create(Arrays.asList("1 2 3 5".split(" "))),
  RowFactory.create(Arrays.asList("1 2".split(" ")))
);
StructType schema = new StructType(new StructField[]{ new StructField(
  "items", new ArrayType(DataTypes.StringType, true), false, Metadata.empty())
});
Dataset<Row> itemsDF = spark.createDataFrame(data, schema);

FPGrowthModel model = new FPGrowth()
  .setItemsCol("items")
  .setMinSupport(0.5)
  .setMinConfidence(0.6)
  .fit(itemsDF);

// Display frequent itemsets.
model.freqItemsets().show();

// Display generated association rules.
model.associationRules().show();

// transform examines the input items against all the association rules and summarize the
// consequents as prediction
model.transform(itemsDF).show();
```

https://spark.apache.org/docs/3.3.1/ml-frequent-pattern-mining.html

ขอบคุณ

Thai

Grazie

Italian

תודה רבה

Hebrew

धन्यवादः

Sanskrit

ಧನ್ಯವಾದಗಳು

Kannada

Ευχαριστώ

Greek

Thank You

English

Gracias

Spanish

Спасибо

Russian

Obrigado

Portuguese

شكراً

Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci

French

多謝

Traditional Chinese

धन्यवाद

Hindi

Danke

German

நன்றி

Tamil

Tamil

多谢

Simplified Chinese

ありがとうございました

Japanese

감사합니다

Korean