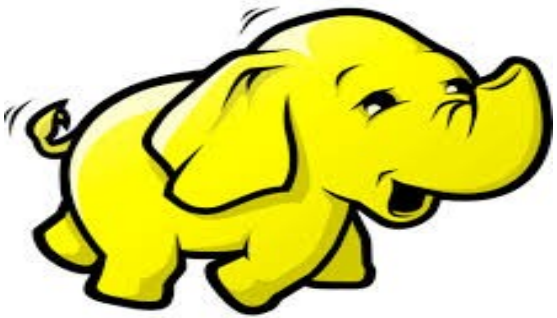
Themes

- Background:
  - Hadoop and MapReduce
  - HDFS (Hadoop Distributed File System)
- Introduction to Spark
- Exercise 1

Hadoop and MapReduce

Hadoop and MapReduce



- Hadoop is a running software framework
  - massively distributed computing
    - on terabytes of data and beyond
    - over thousands of computing nodes *(or on your laptop)*
  - mostly written in Java
  - components:
    - *HDFS* – Hadoop Distributed File System
    - *MapReduce* – the original distributed computing model
    - *YARN* – job tracking and process monitoring
    - *Common* – libraries and utilities (.jar-files)
  - most components can be run separately
  - part of Apaches bigger ecology of big-data technologies

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Hadoop big-data ecosystem

- Developed by Google
- Maintained by The Apache Software Foundation today
- (One of) the first big data frameworks
- Many newer frame-works exist today
- Hadoop is still a good reference and starting point

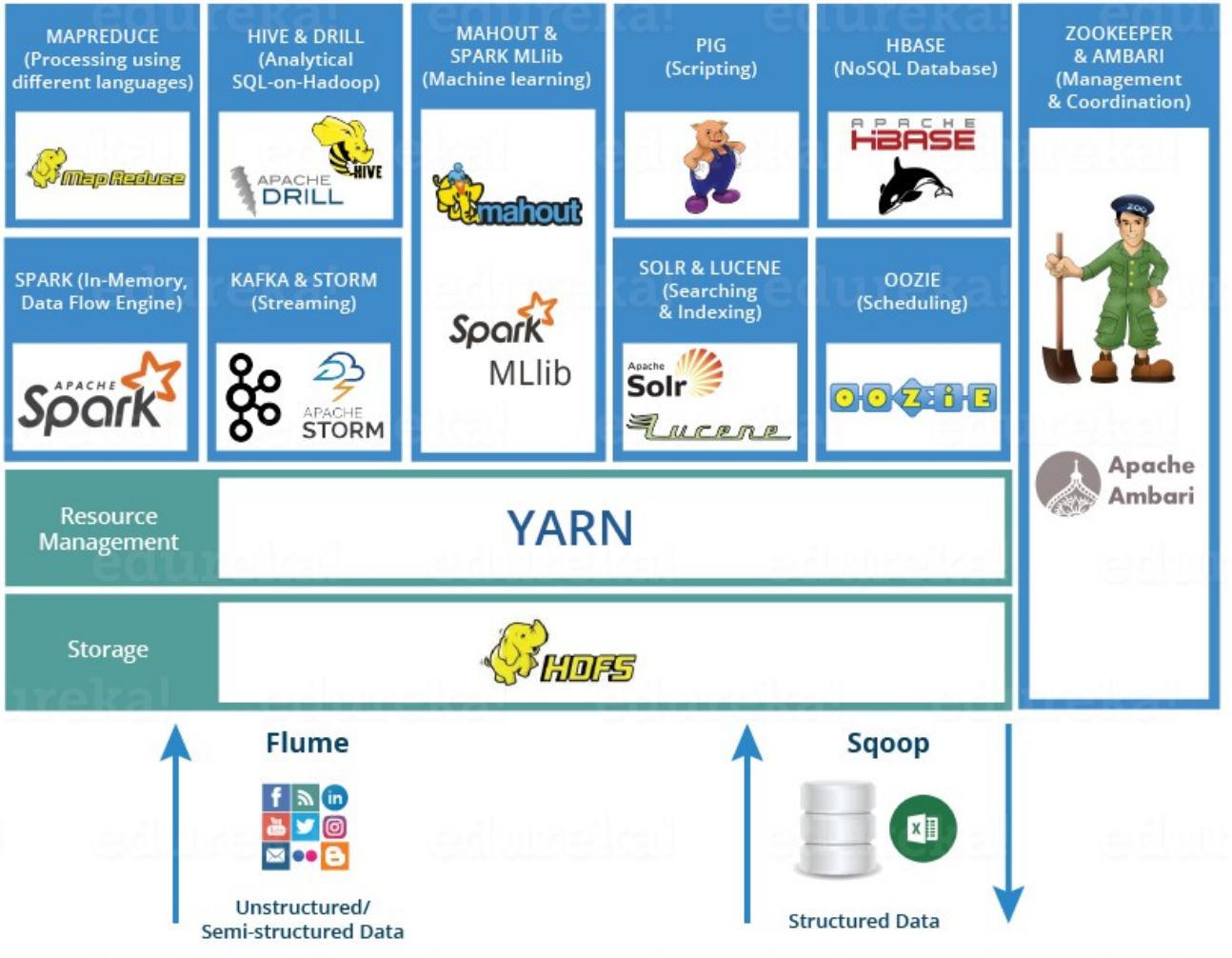


Figure: https://www.bogotobogo.com/Hadoop/BigData_hadoop_Ecosystem.php

HDFS (Hadoop Distributed File System)

- Distributed, scalable, portable file system / data store
  - optimised for mostly immutable files
- Cluster of *data nodes*
  - splits large files (Tb-Pb) into *blocks* (many Mb) that are
  - blocks are replicated across nodes (and racks / switches) – *sharding*
- Single *name node*
  - keeps track of the blocks
  - can be replicated
- TCP / IP
- Appears to clients as a *single logical file storage*

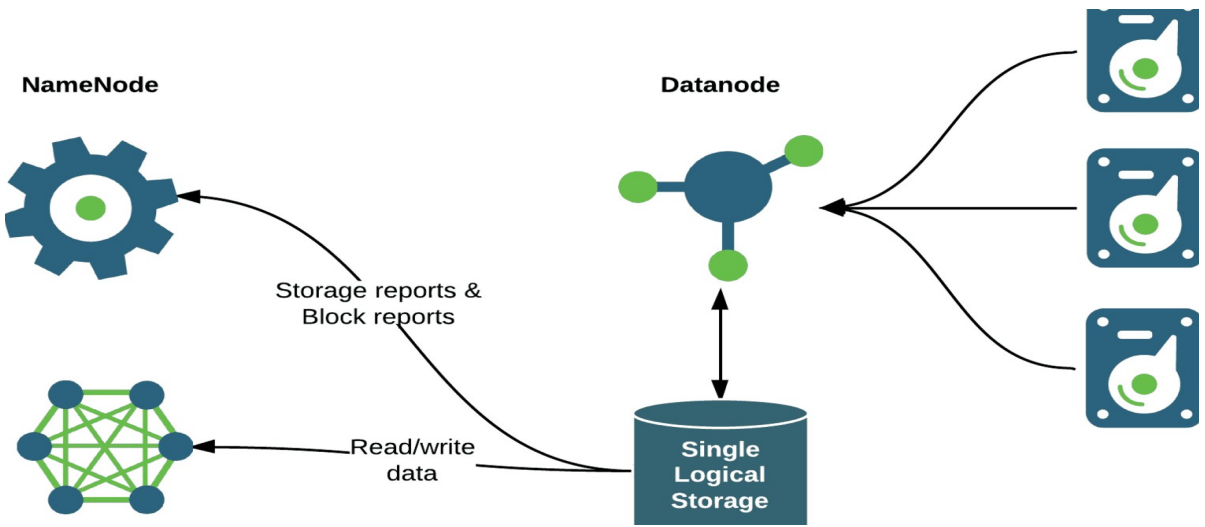
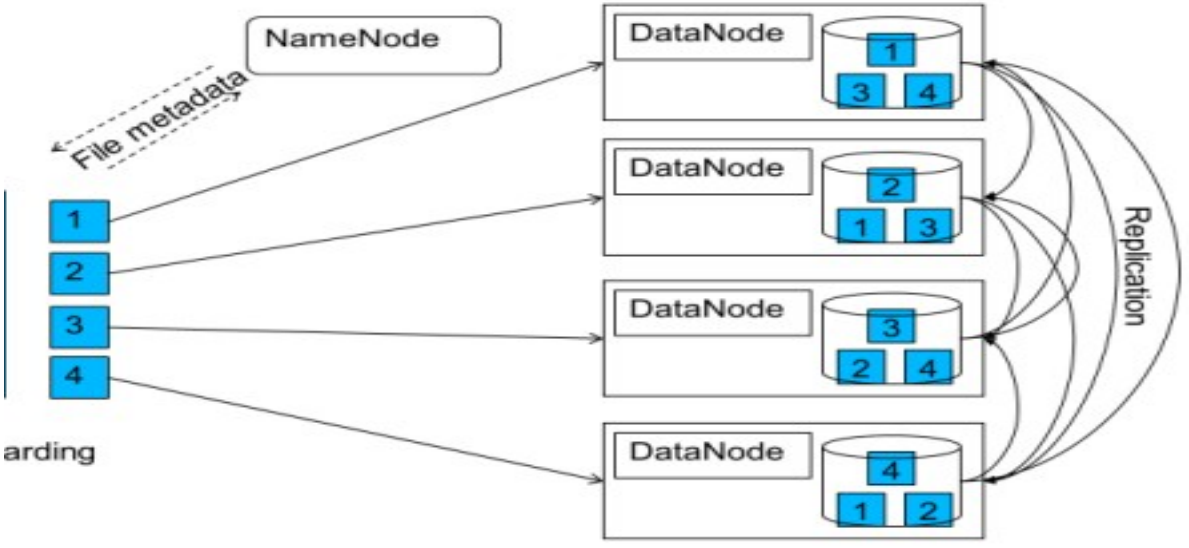


Figure: https://hortonworks.com/blog/heterogeneous-storages-hdfs/

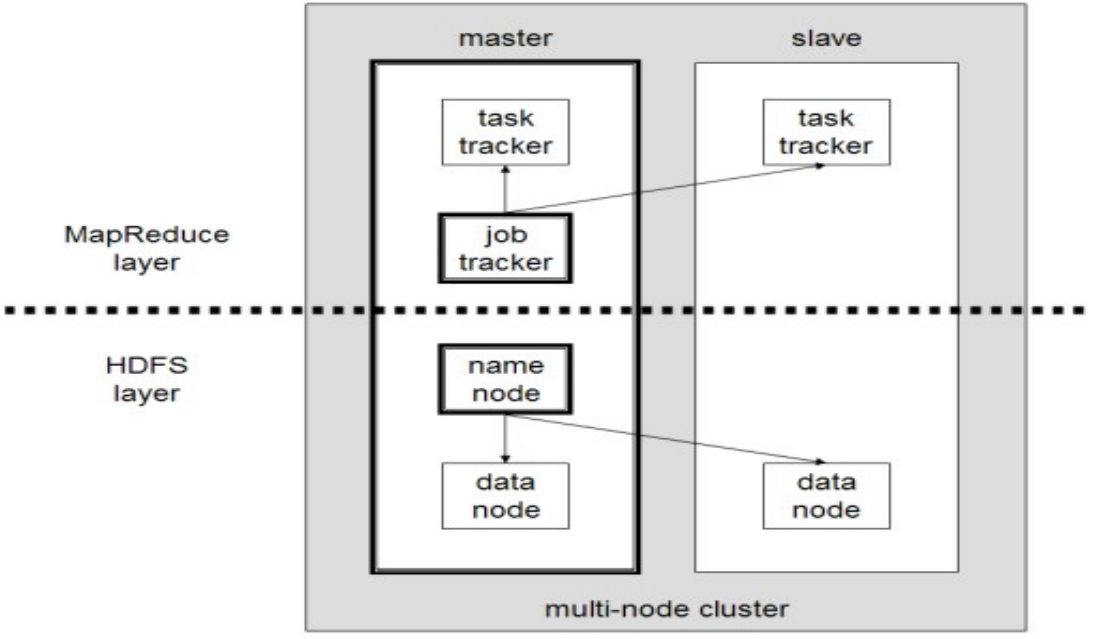
HDFS (Hadoop Distributed File System)

- Distributed, scalable, portable file system / data store
  - optimised for mostly immutable files
- Cluster of *data nodes*
  - splits large files (Tb-Pb) into *blocks* (many Mb)
  - blocks are replicated across nodes (and racks / switches) – *sharding*
- Single *name node*
  - keeps track of the blocks
  - can be replicated
- TCP / IP
- Appears to clients as a *single logical file storage*



Hadoop cluster

- MapReduce and HDFS *may run on the same nodes*
- *Master node*:
  - runs the *job tracker* and name node (in Hadoop 1)
  - and can be a slave too
- *Slave nodes*:
  - runs *task trackers* *and* data nodes
  - possibly not both
- Tasks can be run *close to their data*
  - *moving tasks to data*



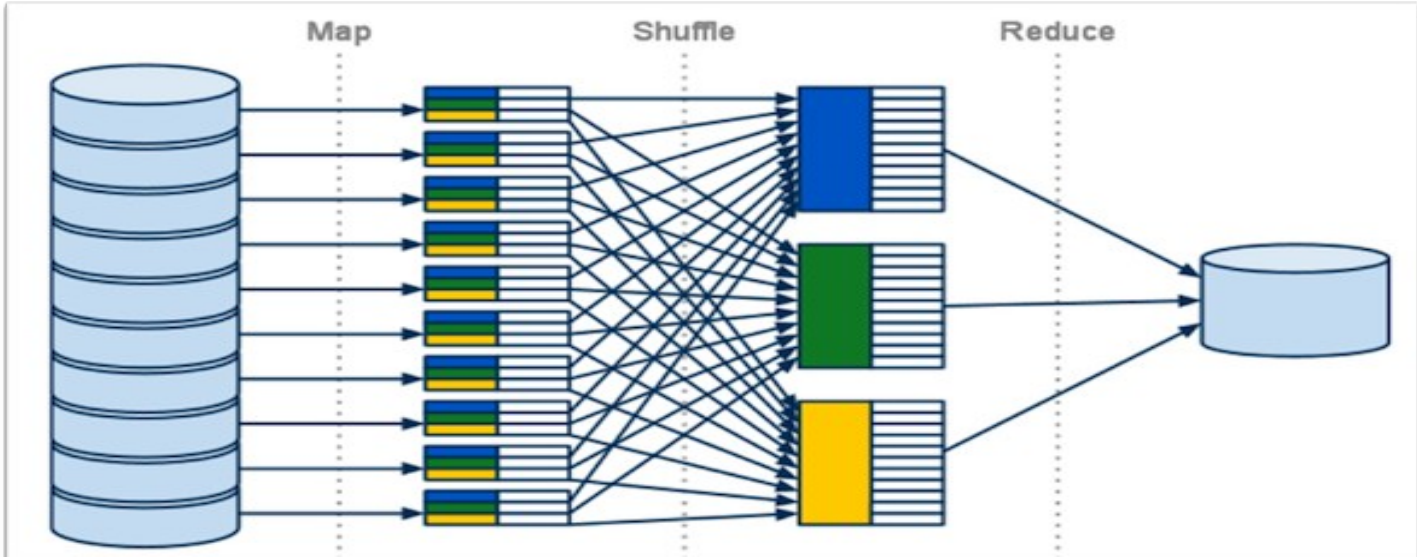
Hadoop cluster

- Single *JobTracker*:
  - *YARN ResourceManager* (in Hadoop 2)
  - receives MapReduce jobs from client(application)-s
  - starts a *YARN MRAppManager* per application
  - pushes the work to task trackers close to the data
  - (simple) scheduling and rescheduling
- Multiple *TaskTrackers*:
  - *YARN NodeManager*
  - has *task slots* available (called *containers*)
  - spawns (lots of) separate JVMs
- *...so what are these tasks?*

MapReduce programming model

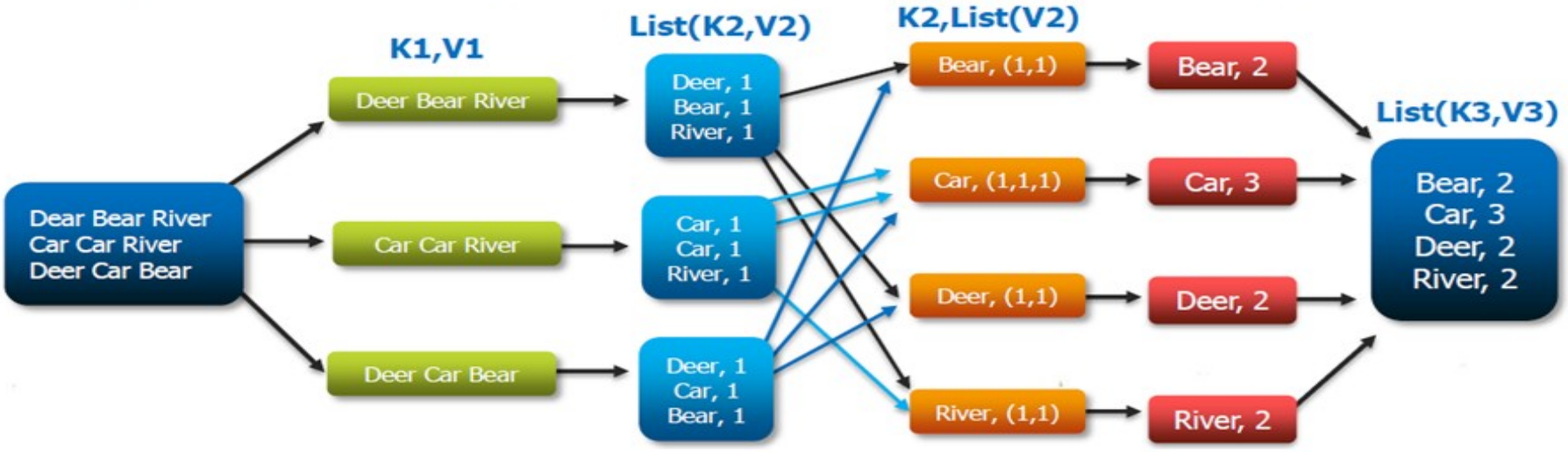
- Two main tasks with an intermediate step:
  - *Map*: main task that filters and sorts local input data
  - *Shuffle*: redistributes intermediate data across nodes
  - *Reduce*: main task that summarises the data in each node

  ...all three steps are parallel (and more can be added)
- *Example:*
  - *the inputs are texts*
  - *we want to count the occurrences of each word*



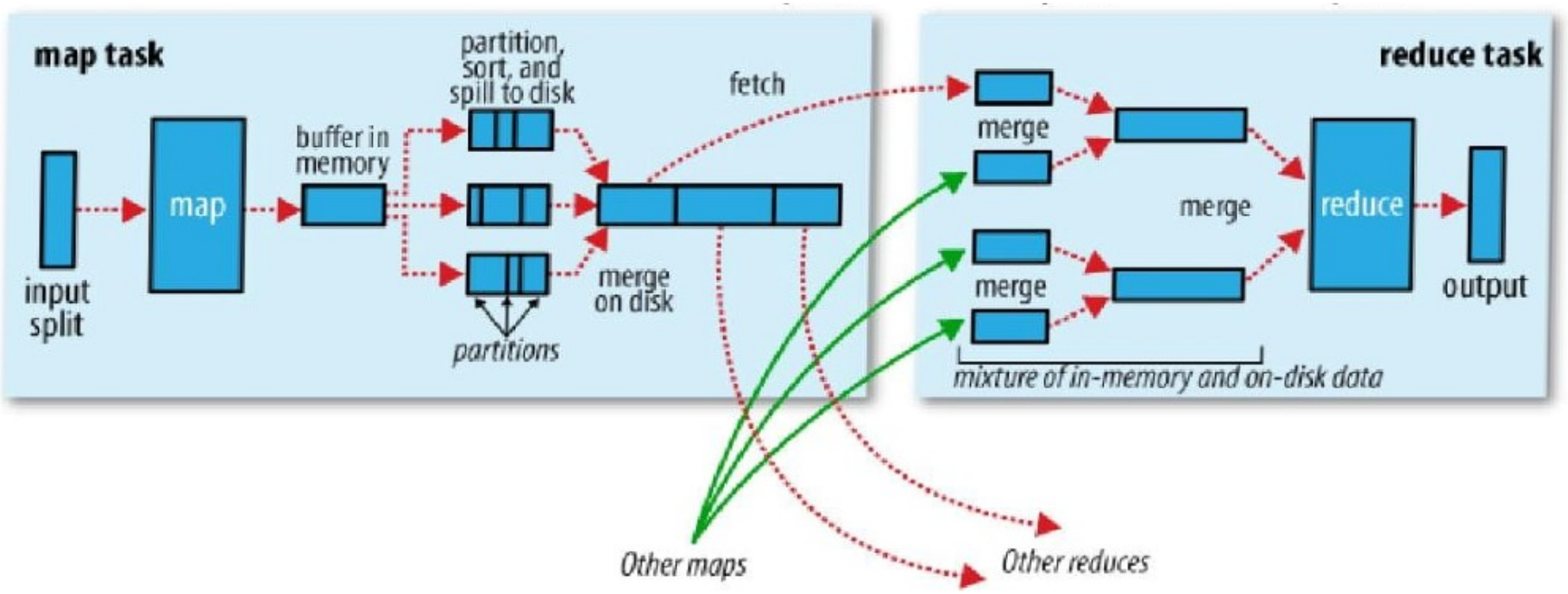
Data are *(key, value)*-pairs

- Map(k1,v1) → list(k2,v2)
  - *Map(<file_id>,<text>) → list(<word>,<count>)*
- Reduce(k2, list (v2)) → (k2, v3 *(or list, or nothing)*)
  - *Reduce(<word>, list (<count>)) → (<word>, <count>)*
  - *associativity* and *commutativity are* helpful



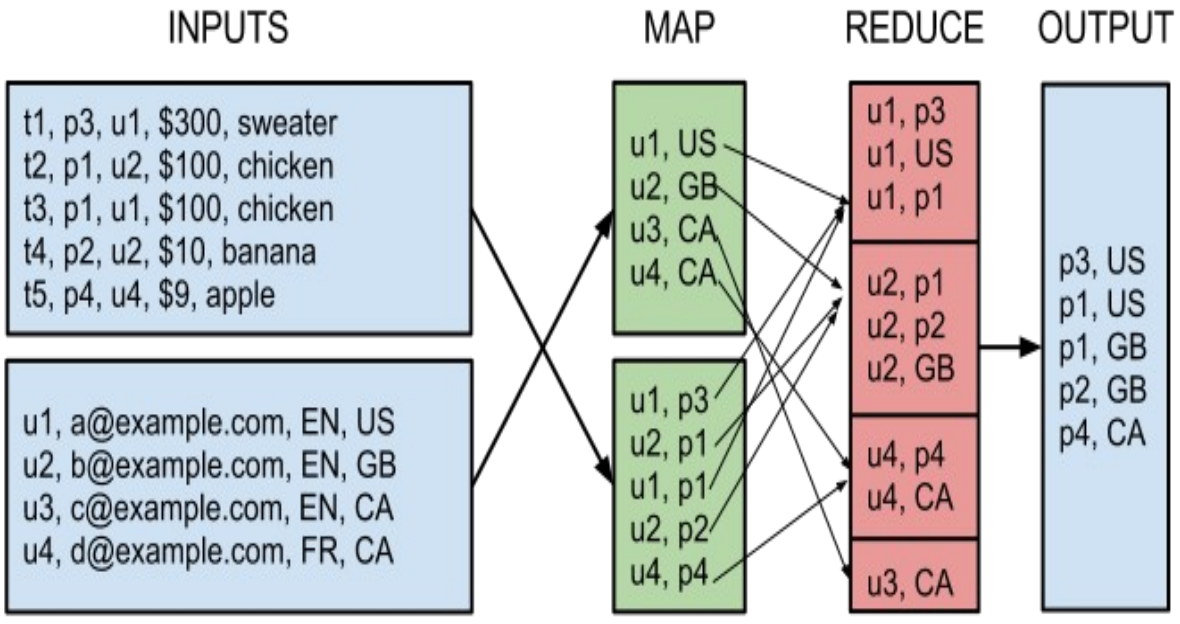
MapReduce internals

- Data shuffling over HTTP
- Highly configurable TaskManager nodes
  - in-memory versus on-disk tradeoffs



More powerful MapReduce processing

- Chained MapReduce jobs
- More complex (key, value)-structures
  - e.g., (IRI, (IRI, IRI)) and (IRI, (IRI, <*literal*>))
- Different maps as input to the same reduce:



More powerful MapReduce processing

- Parameter sweep:
  - input data are used as control parameters
  - actual data to be analysed is shared as configuration
- Additional task and control types:
  - Reduce: can be skipped
  - CompressionCodec: such as gzip / gunzip
  - InputSplit: before mapping, respecting record boundaries
  - Combiner: local aggregation of the Map results; cuts down data transfer Mapper → Reducer
  - Partitioner: controls which keys (and hence values) go to which Reducers (and thus how many are needed)
  - Comparator: to control sorting of values in the Reducer
  - Counter: to report Mapper and Reducer statistics

Configuration

- Map:
  - one map task can map one block of data
  - 10-100 map tasks suggested per computing node
  - should take minutes each, due to setup overhead
  - *example: 10TB input data, 128MB blocksize → 82 000 maps → 800-8000 computing nodes*
  - output buffer size
- Reduce:
  - 0.95 or 1.75 * #computing-nodes * #max-containers
  - not 1 or 2 * ...: leave containers for rescheduling etc.
  - 0.95 * …: all reduces start simultaneously
  - 1.75 * …: first round can start when data available

Beyond MapReduce

- Newer technologies extend / replace MapReduce:
  - less disk dependency
  - data schemas and optimisation
  - move from batch to streaming (live inputs / outputs)
  - more inter-task communication
  - higher-level interfaces and programming abstractions
  - SQL-on-Hadoop, OLAP-on-Hadoop
  - embedded support, e.g., for machine learning
  - *Spark*

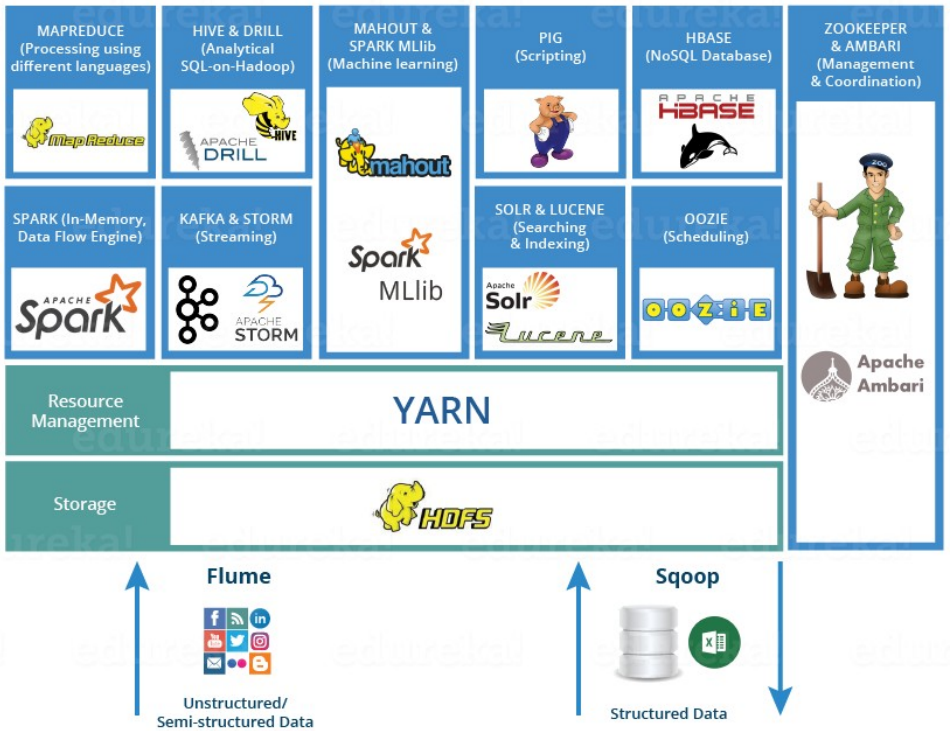
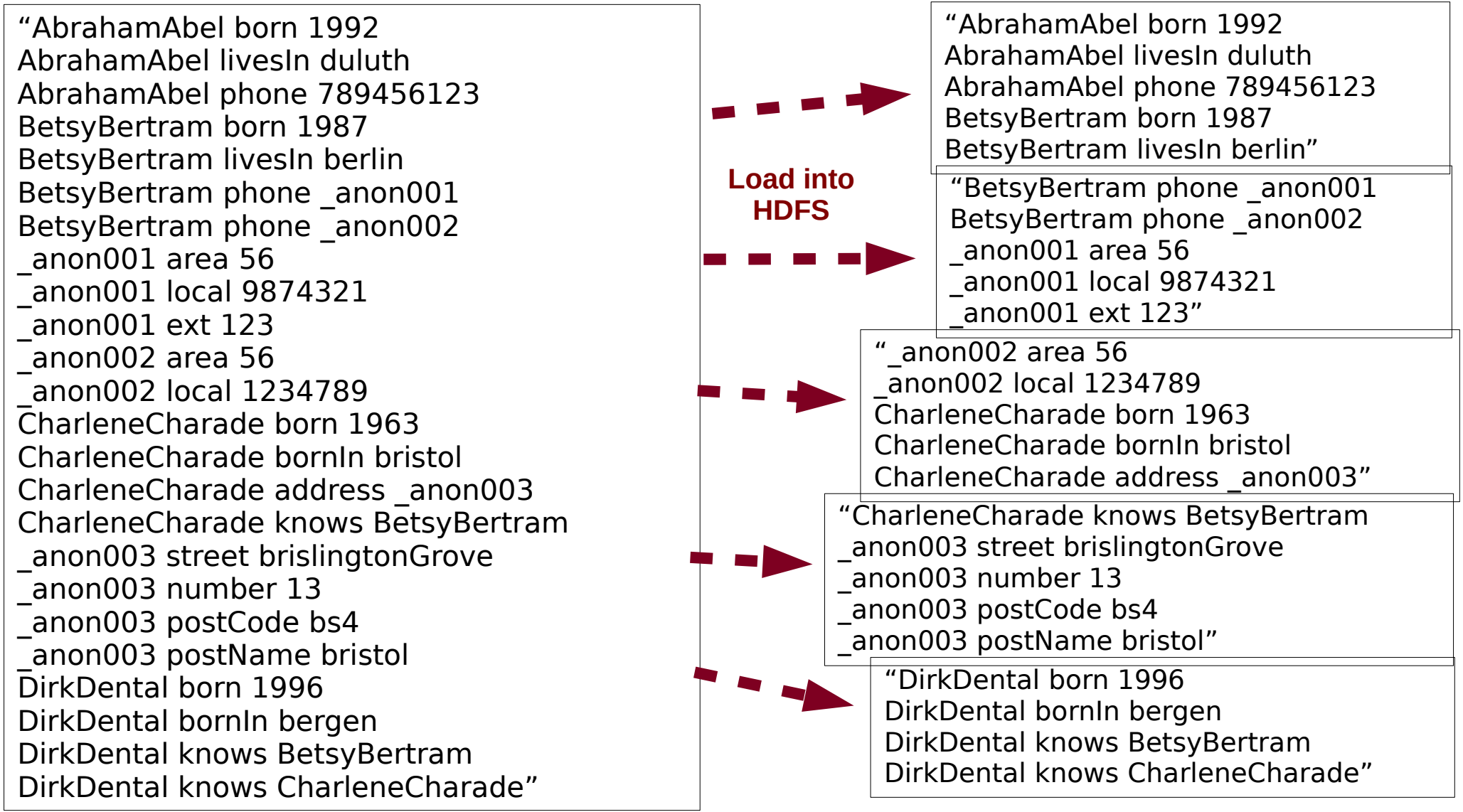
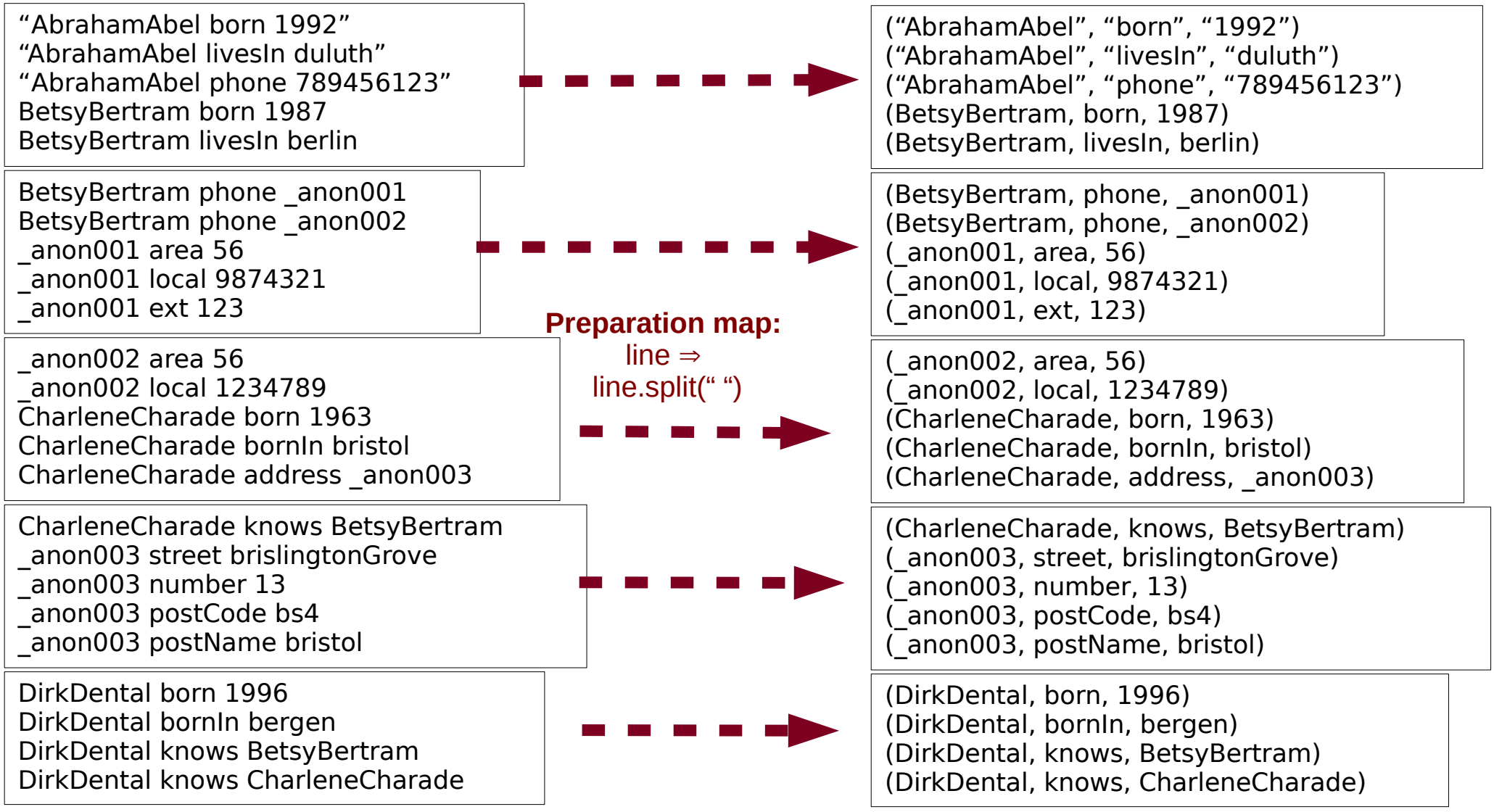
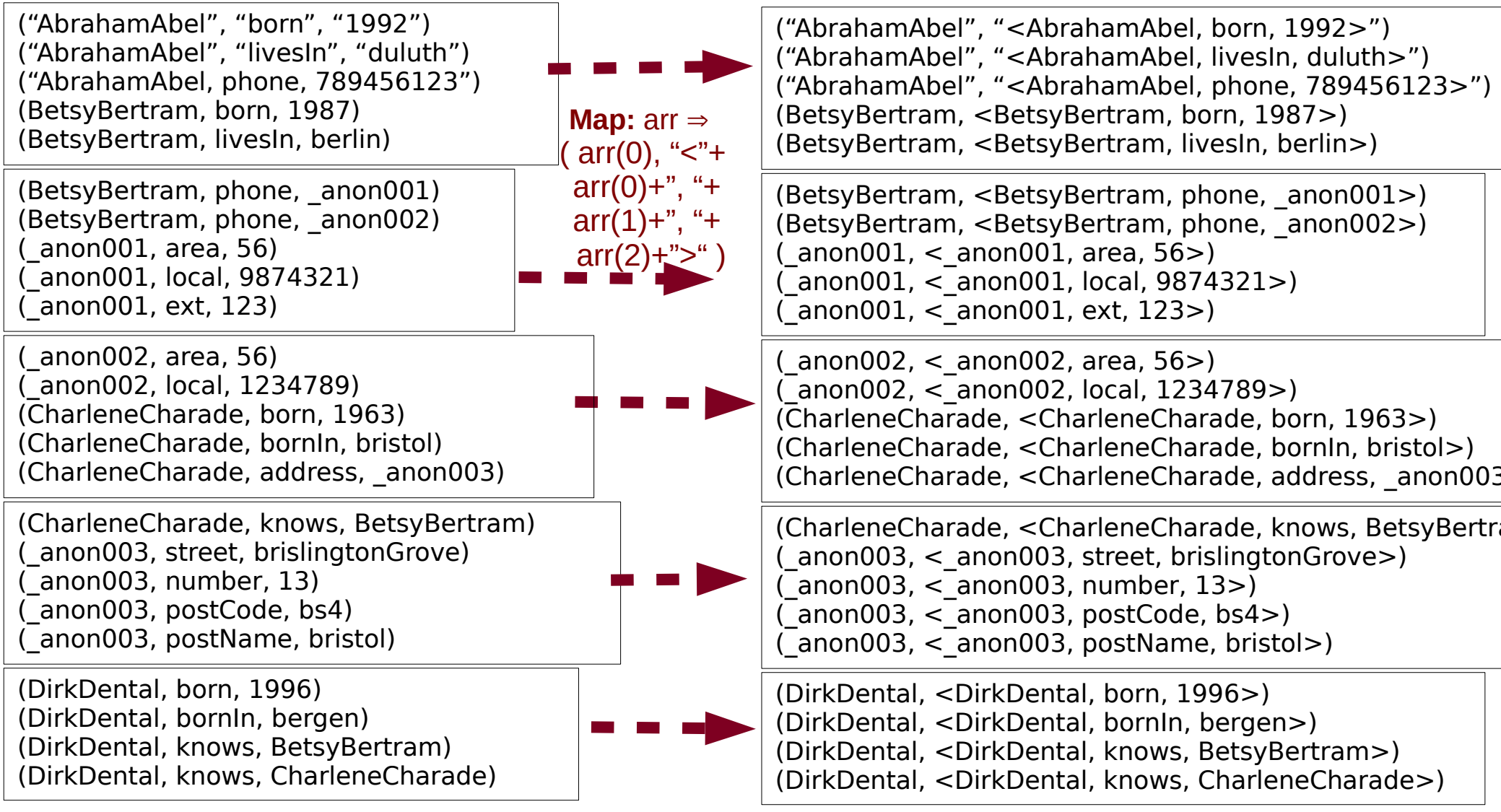


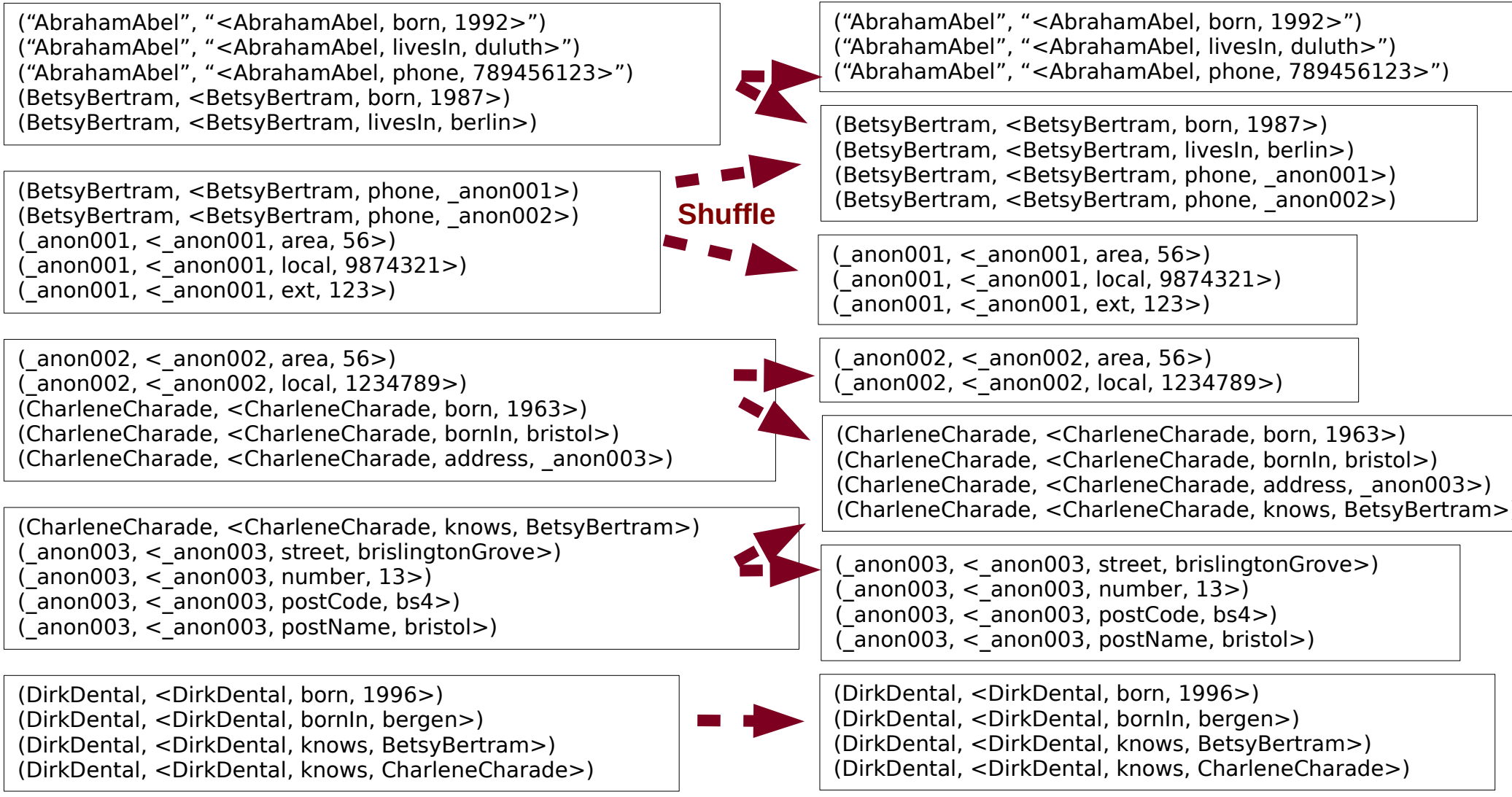
Figure: https://www.bogotobogo.com/Hadoop/BigData_hadoop_Ecosystem.php

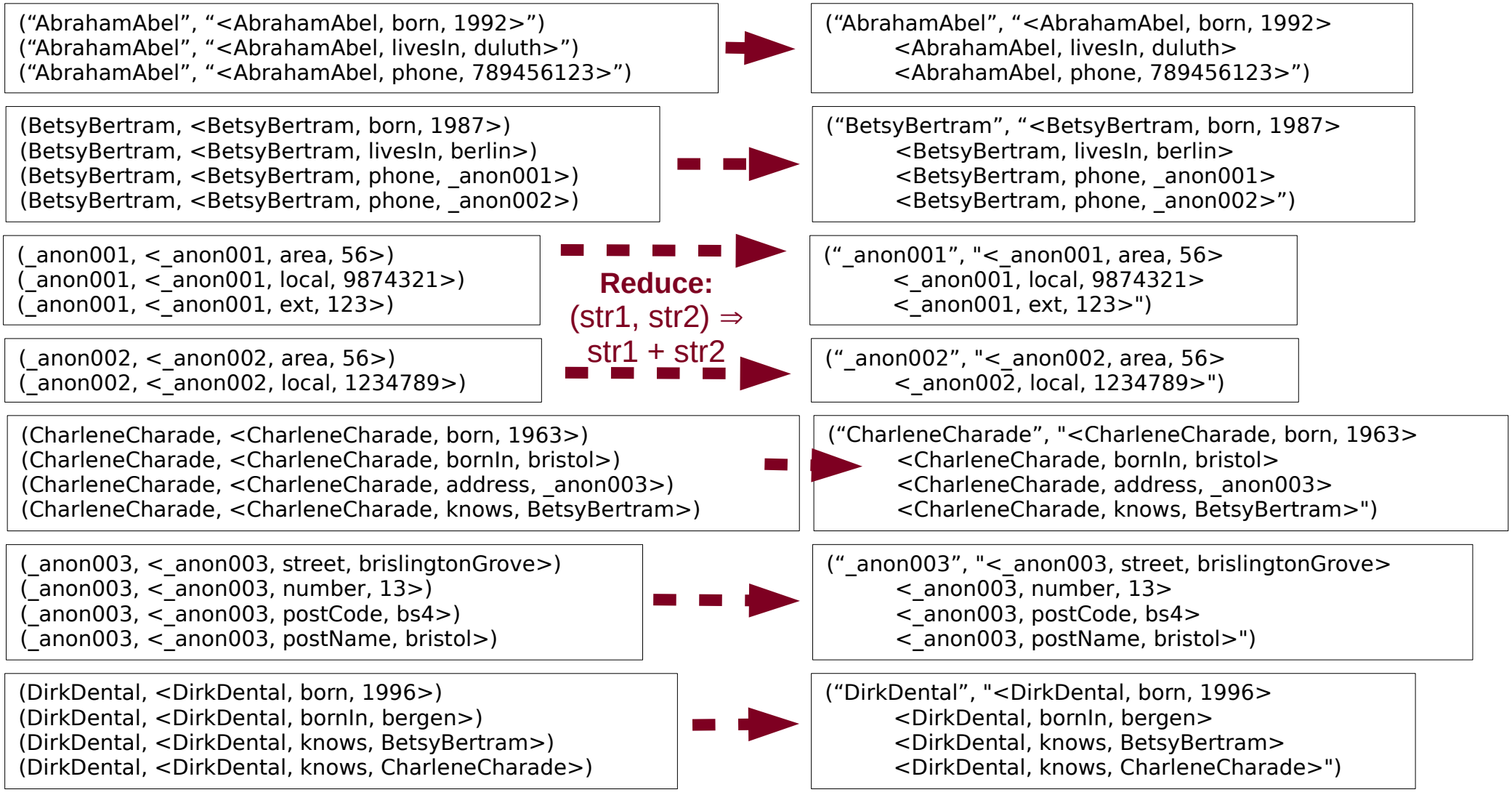
MapReduce example


“AbrahamAbel born 1992
AbrahamAbel livesIn duluth
AbrahamAbel phone 789456123
BetsyBertram born 1987
BetsyBertram livesIn berlin
BetsyBertram phone _anon001
BetsyBertram phone _anon002
_anon001 area 56
_anon001 local 9874321
_anon001 ext 123
_anon002 area 56
_anon002 local 1234789
CharleneCharade born 1963
CharleneCharade bornIn bristol
CharleneCharade address _anon003
CharleneCharade knows BetsyBertram
_anon003 street brislingtonGrove
_anon003 number 13
_anon003 postCode bs4
_anon003 postName bristol
DirkDental born 1996
DirkDental bornIn bergen
DirkDental knows BetsyBertram
DirkDental knows CharleneCharade”
Load into HDFS
“AbrahamAbel born 1992
AbrahamAbel livesIn duluth
AbrahamAbel phone 789456123
BetsyBertram born 1987
BetsyBertram livesIn berlin”
“BetsyBertram phone _anon001
BetsyBertram phone _anon002
_anon001 area 56
_anon001 local 9874321
_anon001 ext 123”
“_anon002 area 56
_anon002 local 1234789
CharleneCharade born 1963
CharleneCharade bornIn bristol
CharleneCharade address _anon003”
“CharleneCharade knows BetsyBertram
_anon003 street brislingtonGrove
_anon003 number 13
_anon003 postCode bs4
_anon003 postName bristol”
“DirkDental born 1996
DirkDental bornIn bergen
DirkDental knows BetsyBertram
DirkDental knows CharleneCharade”


“AbrahamAbel born 1992”
“AbrahamAbel livesIn duluth”
“AbrahamAbel phone 789456123”
BetsyBertram born 1987
BetsyBertram livesIn berlin
(“AbrahamAbel”, “born”, “1992”)
(“AbrahamAbel”, “livesIn”, “duluth”)
(“AbrahamAbel”, “phone”, “789456123”)
(BetsyBertram, born, 1987)
(BetsyBertram, livesIn, berlin)
BetsyBertram phone _anon001
BetsyBertram phone _anon002
_anon001 area 56
_anon001 local 9874321
_anon001 ext 123
(BetsyBertram, phone, _anon001)
(BetsyBertram, phone, _anon002)
(_anon001, area, 56)
(_anon001, local, 9874321)
(_anon001, ext, 123)
Preparation map:
line ⇒
line.split(“ “)
_anon002 area 56
_anon002 local 1234789
CharleneCharade born 1963
CharleneCharade bornIn bristol
CharleneCharade address _anon003
(_anon002, area, 56)
(_anon002, local, 1234789)
(CharleneCharade, born, 1963)
(CharleneCharade, bornIn, bristol)
(CharleneCharade, address, _anon003)
CharleneCharade knows BetsyBertram
_anon003 street brislingtonGrove
_anon003 number 13
_anon003 postCode bs4
_anon003 postName bristol
(CharleneCharade, knows, BetsyBertram)
(_anon003, street, brislingtonGrove)
(_anon003, number, 13)
(_anon003, postCode, bs4)
(_anon003, postName, bristol)
DirkDental born 1996
DirkDental bornIn bergen
DirkDental knows BetsyBertram
DirkDental knows CharleneCharade
(DirkDental, born, 1996)
(DirkDental, bornIn, bergen)
(DirkDental, knows, BetsyBertram)
(DirkDental, knows, CharleneCharade)


(“AbrahamAbel”, “born”, “1992”)
(“AbrahamAbel”, “livesIn”, “duluth”)
(“AbrahamAbel, phone, 789456123”)
(BetsyBertram, born, 1987)
(BetsyBertram, livesIn, berlin)
Map: arr ⇒
(arr(0), “<”+
arr(0)+”, “+
arr(1)+”, “+
arr(2)+”>“)
(“AbrahamAbel”, “<AbrahamAbel, born, 1992>”)
(“AbrahamAbel”, “<AbrahamAbel, livesIn, duluth>”)
(“AbrahamAbel”, “<AbrahamAbel, phone, 789456123>”)
(BetsyBertram, <BetsyBertram, born, 1987>)
(BetsyBertram, <BetsyBertram, livesIn, berlin>)
(BetsyBertram, phone, _anon001)
(BetsyBertram, phone, _anon002)
(_anon001, area, 56)
(_anon001, local, 9874321)
(_anon001, ext, 123)
(BetsyBertram, <BetsyBertram, phone, _anon001>)
(BetsyBertram, <BetsyBertram, phone, _anon002>)
(_anon001, <_anon001, area, 56>)
(_anon001, <_anon001, local, 9874321>)
(_anon001, <_anon001, ext, 123>)
(_anon002, area, 56)
(_anon002, local, 1234789)
(CharleneCharade, born, 1963)
(CharleneCharade, bornIn, bristol)
(CharleneCharade, address, _anon003)
(_anon002, <_anon002, area, 56>)
(_anon002, <_anon002, local, 1234789>)
(CharleneCharade, <CharleneCharade, born, 1963>)
(CharleneCharade, <CharleneCharade, bornIn, bristol>)
(CharleneCharade, <CharleneCharade, address, _anon003
(CharleneCharade, knows, BetsyBertram)
(_anon003, street, brislingtonGrove)
(_anon003, number, 13)
(_anon003, postCode, bs4)
(_anon003, postName, bristol)
(CharleneCharade, <CharleneCharade, knows, BetsyBertra
(_anon003, <_anon003, street, brislingtonGrove>)
(_anon003, <_anon003, number, 13>)
(_anon003, <_anon003, postCode, bs4>)
(_anon003, <_anon003, postName, bristol>)
(DirkDental, born, 1996)
(DirkDental, bornIn, bergen)
(DirkDental, knows, BetsyBertram)
(DirkDental, knows, CharleneCharade)
(DirkDental, <DirkDental, born, 1996>)
(DirkDental, <DirkDental, bornIn, bergen>)
(DirkDental, <DirkDental, knows, BetsyBertram>)
(DirkDental, <DirkDental, knows, CharleneCharade>)


(“AbrahamAbel”, “<AbrahamAbel, born, 1992>”)
(“AbrahamAbel”, “<AbrahamAbel, livesIn, duluth>”)
(“AbrahamAbel”, “<AbrahamAbel, phone, 789456123>”)
(BetsyBertram, <BetsyBertram, born, 1987>)
(BetsyBertram, <BetsyBertram, livesIn, berlin>)
(BetsyBertram, <BetsyBertram, phone, _anon001>)
(BetsyBertram, <BetsyBertram, phone, _anon002>)
(_anon001, <_anon001, area, 56>)
(_anon001, <_anon001, local, 9874321>)
(_anon001, <_anon001, ext, 123>)
(_anon002, <_anon002, area, 56>)
(_anon002, <_anon002, local, 1234789>)
(CharleneCharade, <CharleneCharade, born, 1963>)
(CharleneCharade, <CharleneCharade, bornIn, bristol>)
(CharleneCharade, <CharleneCharade, address, _anon003>)
(CharleneCharade, <CharleneCharade, knows, BetsyBertram>)
(_anon003, <_anon003, street, brislingtonGrove>)
(_anon003, <_anon003, number, 13>)
(_anon003, <_anon003, postCode, bs4>)
(_anon003, <_anon003, postName, bristol>)
(DirkDental, <DirkDental, born, 1996>)
(DirkDental, <DirkDental, bornIn, bergen>)
(DirkDental, <DirkDental, knows, BetsyBertram>)
(DirkDental, <DirkDental, knows, CharleneCharade>)
Shuffle
(“AbrahamAbel”, “<AbrahamAbel, born, 1992>”)
(“AbrahamAbel”, “<AbrahamAbel, livesIn, duluth>”)
(“AbrahamAbel”, “<AbrahamAbel, phone, 789456123>”)
(BetsyBertram, <BetsyBertram, born, 1987>)
(BetsyBertram, <BetsyBertram, livesIn, berlin>)
(BetsyBertram, <BetsyBertram, phone, _anon001>)
(BetsyBertram, <BetsyBertram, phone, _anon002>)
(_anon001, <_anon001, area, 56>)
(_anon001, <_anon001, local, 9874321>)
(_anon001, <_anon001, ext, 123>)
(_anon002, <_anon002, area, 56>)
(_anon002, <_anon002, local, 1234789>)
(CharleneCharade, <CharleneCharade, born, 1963>)
(CharleneCharade, <CharleneCharade, bornIn, bristol>)
(CharleneCharade, <CharleneCharade, address, _anon003>)
(CharleneCharade, <CharleneCharade, knows, BetsyBertram>)
(_anon003, <_anon003, street, brislingtonGrove>)
(_anon003, <_anon003, number, 13>)
(_anon003, <_anon003, postCode, bs4>)
(_anon003, <_anon003, postName, bristol>)
(DirkDental, <DirkDental, born, 1996>)
(DirkDental, <DirkDental, bornIn, bergen>)
(DirkDental, <DirkDental, knows, BetsyBertram>)
(DirkDental, <DirkDental, knows, CharleneCharade>)


(“AbrahamAbel”, “<AbrahamAbel, born, 1992>”)
(“AbrahamAbel”, “<AbrahamAbel, livesIn, duluth>”)
(“AbrahamAbel”, “<AbrahamAbel, phone, 789456123>”)
(“AbrahamAbel”, “<AbrahamAbel, born, 1992>
<AbrahamAbel, livesIn, duluth>
<AbrahamAbel, phone, 789456123>”)
(BetsyBertram, <BetsyBertram, born, 1987>)
(BetsyBertram, <BetsyBertram, livesIn, berlin>)
(BetsyBertram, <BetsyBertram, phone, _anon001>)
(BetsyBertram, <BetsyBertram, phone, _anon002>)
(“BetsyBertram”, “<BetsyBertram, born, 1987>
<BetsyBertram, livesIn, berlin>
<BetsyBertram, phone, _anon001>
<BetsyBertram, phone, _anon002>”)
(_anon001, <_anon001, area, 56>)
(_anon001, <_anon001, local, 9874321>)
(_anon001, <_anon001, ext, 123>)
Reduce:
(str1, str2) ⇒
str1 + str2
(“_anon001”, "<_anon001, area, 56>
<_anon001, local, 9874321>
<_anon001, ext, 123>")
(_anon002, <_anon002, area, 56>)
(_anon002, <_anon002, local, 1234789>)
(“_anon002”, "<_anon002, area, 56>
<_anon002, local, 1234789>")
(CharleneCharade, <CharleneCharade, born, 1963>)
(CharleneCharade, <CharleneCharade, bornIn, bristol>)
(CharleneCharade, <CharleneCharade, address, _anon003>)
(CharleneCharade, <CharleneCharade, knows, BetsyBertram>)
(“CharleneCharade”, "<CharleneCharade, born, 1963>
<CharleneCharade, bornIn, bristol>
<CharleneCharade, address, _anon003>
<CharleneCharade, knows, BetsyBertram>")
(_anon003, <_anon003, street, brislingtonGrove>)
(_anon003, <_anon003, number, 13>)
(_anon003, <_anon003, postCode, bs4>)
(_anon003, <_anon003, postName, bristol>)
(“_anon003”, "<_anon003, street, brislingtonGrove>
<_anon003, number, 13>
<_anon003, postCode, bs4>
<_anon003, postName, bristol>")
(DirkDental, <DirkDental, born, 1996>)
(DirkDental, <DirkDental, bornIn, bergen>)
(DirkDental, <DirkDental, knows, BetsyBertram>)
(DirkDental, <DirkDental, knows, CharleneCharade>)
(“DirkDental”, "<DirkDental, born, 1996>
<DirkDental, bornIn, bergen>
<DirkDental, knows, BetsyBertram>
<DirkDental, knows, CharleneCharade>")

Apache Spark

Limitations of Hadoop MapReduce

- *Batch*-oriented, with jobs that can take day(s)
- Heavily *disk* based
  - less suitable for *iterative* and *interactive* tasks
- *Rigid*, with few operations, e.g., for
  - structured data schemas and optimisation (e.g., SQL)
  - sharing data (broadcasts and accumulators)
  - machine learning
  - graph processing
  - streaming (or live) data

Apache Spark

- Next step after Hadoop / MapReduce:
  - disk → *memory*
  - batch → *streams*
  - map & reduce → lots of *lower- and higher-level operations*
- Comes with libraries for, e.g.,
  - SQL
  - machine learning (MLib)
  - graph processing (GraphX)
  - streaming (DStream)
  - big database connection (Cassandra)

Part of the Apache big-data ecosystem

- *HDFS and Hadoop*
- *Mahout* → machine learning
- *Giraph* → graph processing
- *Storm & Flink* → stream processing
- *Hive* → SQL
- *Pig* → OLAP

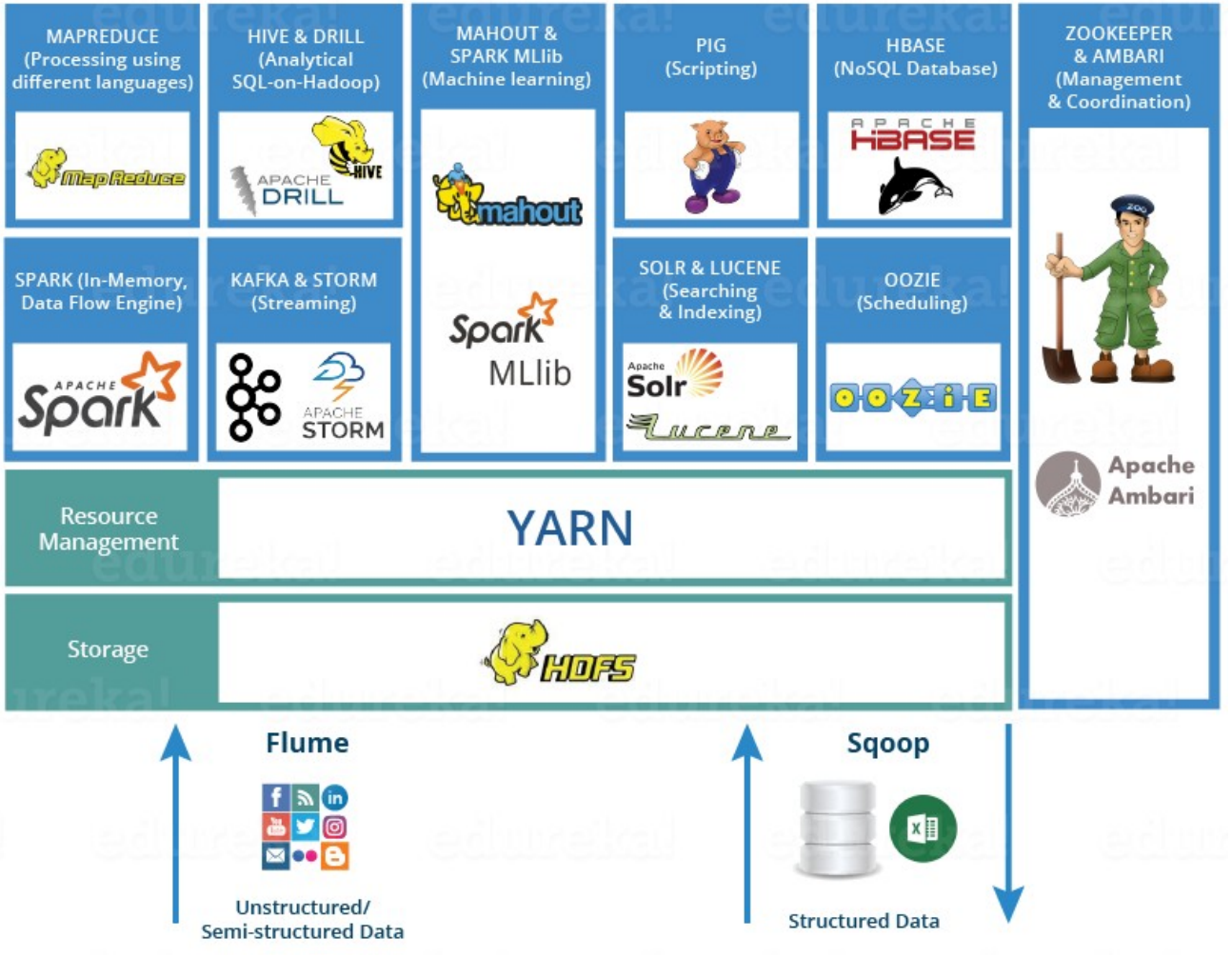
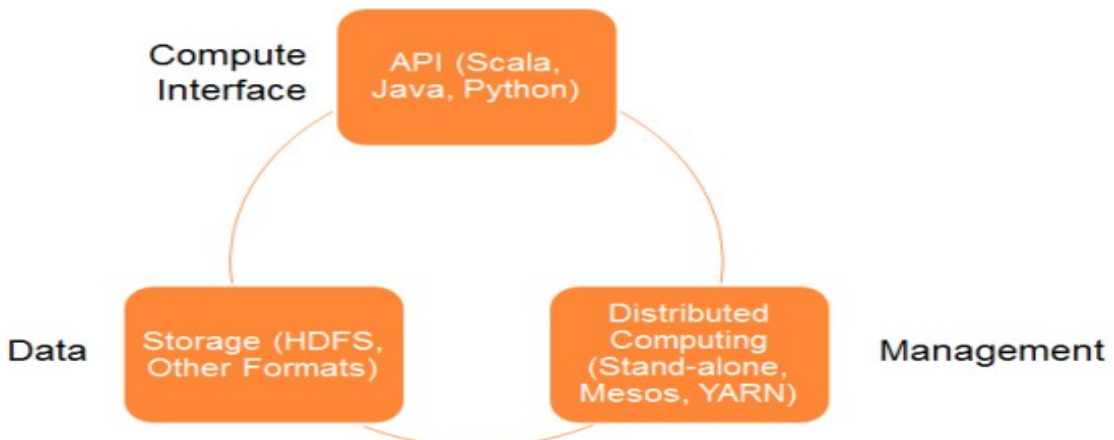


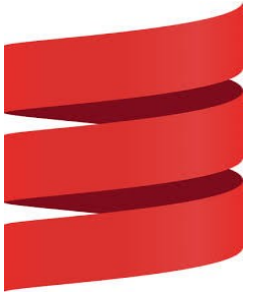
Figure: https://www.bogotobogo.com/Hadoop/BigData_hadoop_Ecosystem.php

Spark

- Written in Scala (→ next slide)
  - optionally integrated with Hadoop
  - lazy (on-demand) evaluation of *transformations*
  - based on *Resilient Distributed Datasets (RDDs)*
- Standalone or on top of HDFS, YARN...
- APIs for
  - Java, Scala, and Python
  - also: R, Clojure...
- Interactive shells:
  - Scala (spark-shell)
  - Python (pyspark)



Scala

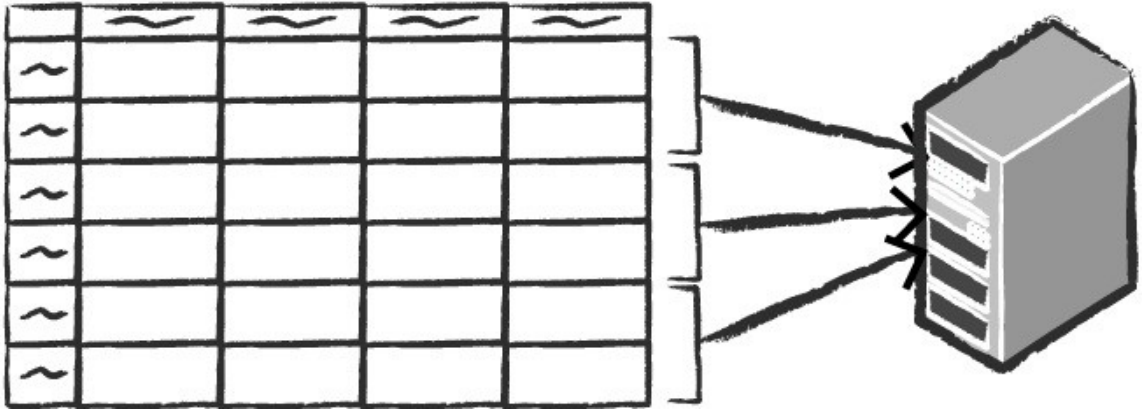


- Runs on the Java Virtual Machine (JVM)
- Similar to Java, but several improvements:
  - interpreted in Scala shell
  - no simple types (*int, long, float, double, char*…)
  - type inference gives simpler type definitions (*var, val*)
  - *functional programming*
    - one of the oldest *programming paradigms* (Lisp)
    - programs comprise mathematical functions
      - takes arguments, returns immutable values
      - functions have no side effects
    - *anonymous functions* / *function literals / closures*
      - defining an on-the-fly computation
      - that can be distributed across a computing cluster

Data partitioning

Spreadsheet on a single machine

Table or Data Frame partitioned across servers in a data center



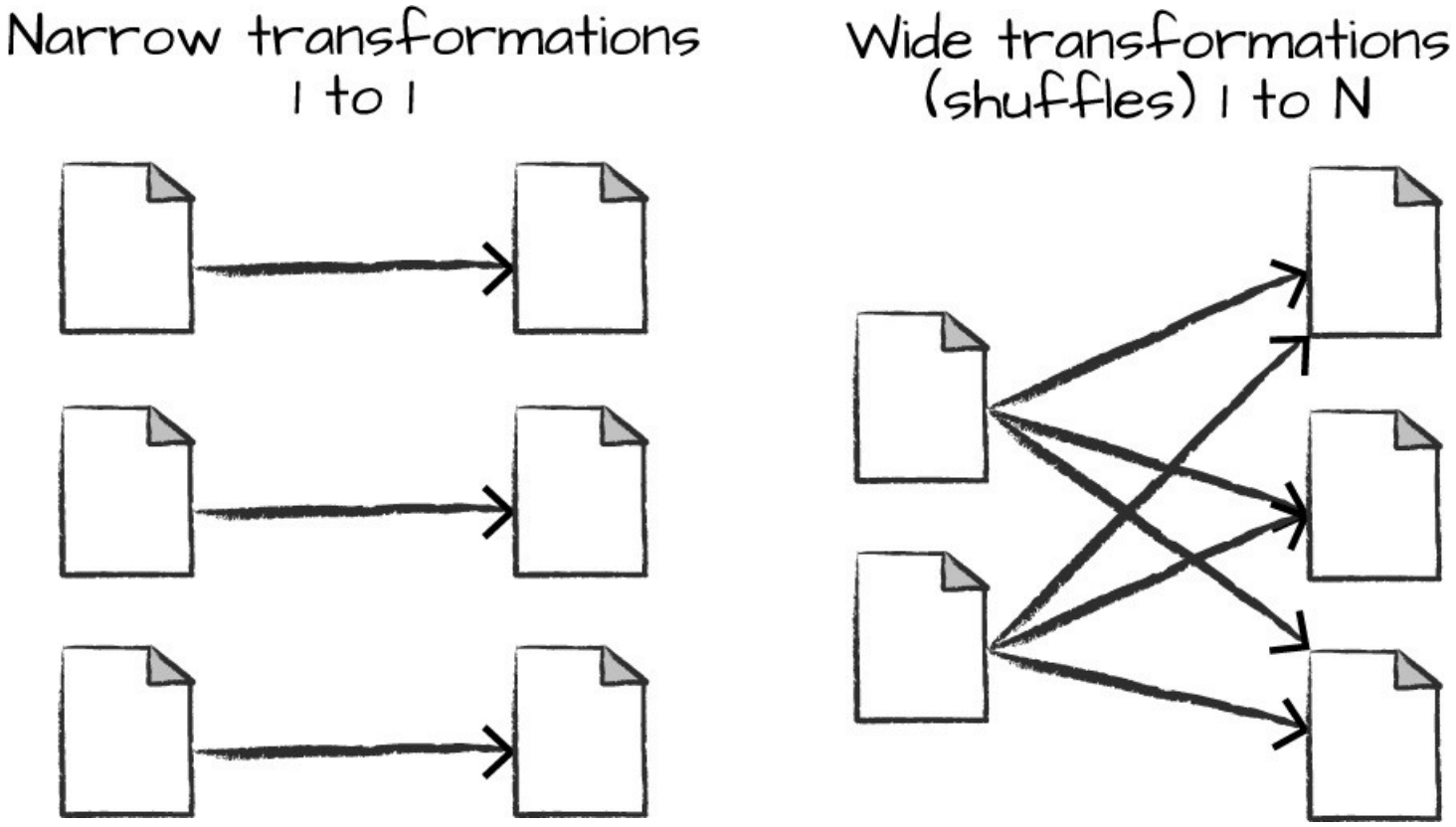
Low-level and structured APIs

- Spark 1.x:
  - low-level operations:
    - transformations, and many actions
  - simpler data types: RDDs, shared variables
  - less structured data (but RDDs can have types)
- Spark 2.x, 3.x:
  - structured operations:
    - SQL-like transformations, fewer actions
  - structured data types: DataFrames with Rows and Columns
  - implemented by the low-level operations
  - RDDs, shared variables, and low-level operations are still available

Low-level Spark concepts

- Computations are directed-acyclic graphs (DAGs) / pipelines of
  - *nodes:* *Resilient Distributed Dataset (RDD)*
    - may contain: key-value pairs, text, tables, graphs...
    - shareable, immutable (cannot change after creation)
    - stored in different partitions
  - *directed edges:* transformations and actions
    - *transformation:*
      - returns new RDD, lazy evaluation (on demand)
    - *action:*
      - evaluates and returns a value – or has a side effect
    - also: *broadcast variables* and *accumulators*

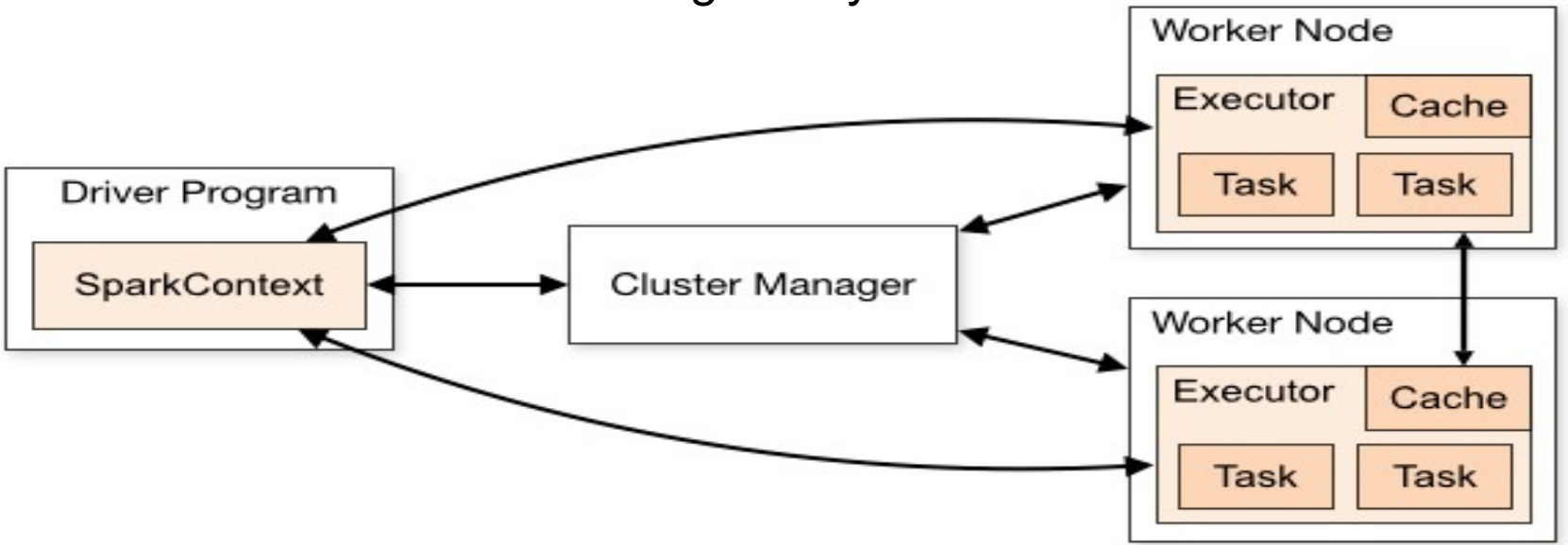
Narrow and wide transformations



Wide trans-formations *persist* the data

Driver and workers

- The driver, manager and each worker/executor can be
  - a processor core when running in a cluster
  - a Java thread when running locally



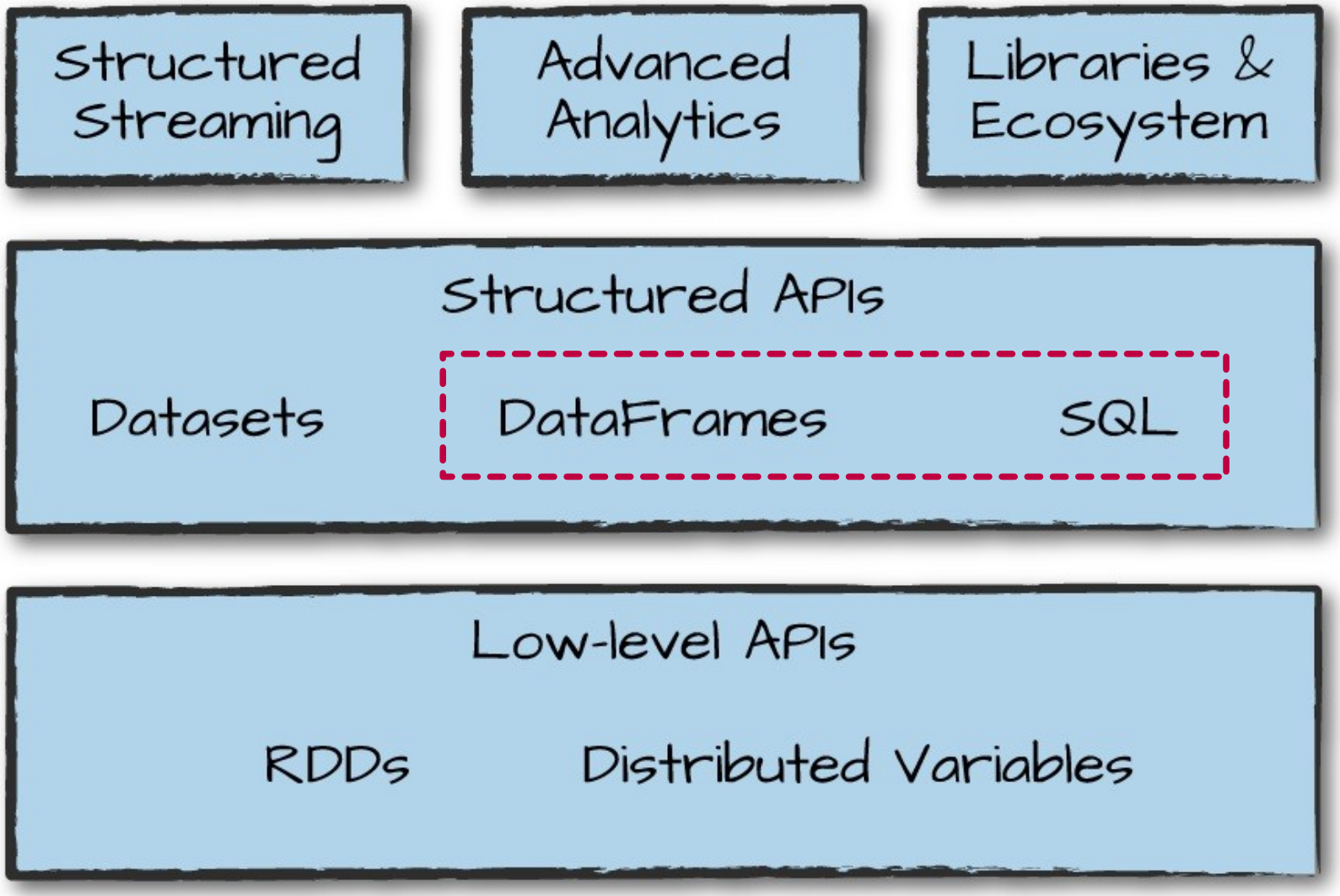
Resilient Distributed Dataset (RDD)

- Immutable, persistent, partitioned
- Created from:
  - native driver data (parallelize)
  - external (distributed) files (e.g., HDFS)
- Can be shared by parallel computations!
- Can contain strings, integers, complex types, etc.
- Normally computed on demand
  - can also be explicitly or implicitly persisted
- Fault-tolerant, through *lineage*

Low-level transformations and actions

- Examples of *transformation functions*:
  - filter, map, mapPartitions, flatMap, glom, sample, union, pipe…
    - *mostly narrow:* do not cause shuffling
  - distinct, intersection, groupByKey, reduceByKey, aggregateByKey, sortByKey, join, cogroup, cartesian, coalesce, repartition…
    - *usually wide:* trigger *data shuffling* between nodes
- Examples of *action operations*:
  - cache, persist, checkpoint
  - reduce, collect, count, first, take, takeSample, takeOrdered, saveAsTextFile, countByKey, foreach...

Structured Apache Spark


Structured Streaming
Advanced Analytics
Libraries & Ecosystem
Structured APIs
Datasets
DataFrames
SQL
Low-level APIs
RDDs
Distributed Variables

DataFrames

- Basic data type in Structured PySpark
  - Java, Scala also has more strongly typed Datasets
- Consists of Rows and Colums
- Create:
  - df = spark.createDataFrame(list_of_tuples, list_of_column_names)
  - df = spark.read.format(format).load(location)
- Get data actions:
  - df.show(n) # print on console
  - df.take(n) # as new DataFrame
  - df.first() # df.take(1) as Row
  - df.collect() # as List

DataFrame actions

- Get schema actions:
  - df.printSchema() # print on console
  - df.schema # as complex object
- Convert actions:
  - df.toPandas()
  - df.rdd # to low-level RDD
- Save actions:
  - df.write.format(format).save(folder_name) # writes a folder of files

DataFrame columns

- Columns:
  - df.select(‘column_name’)
  - df.select(df.column_name)
  - from pyspark.sql.functions import col, column
    df.column_var = column(‘column_name’)
    df.select(column_var)
  - ...can also select a list of columns

DataFrames ...and some SQL

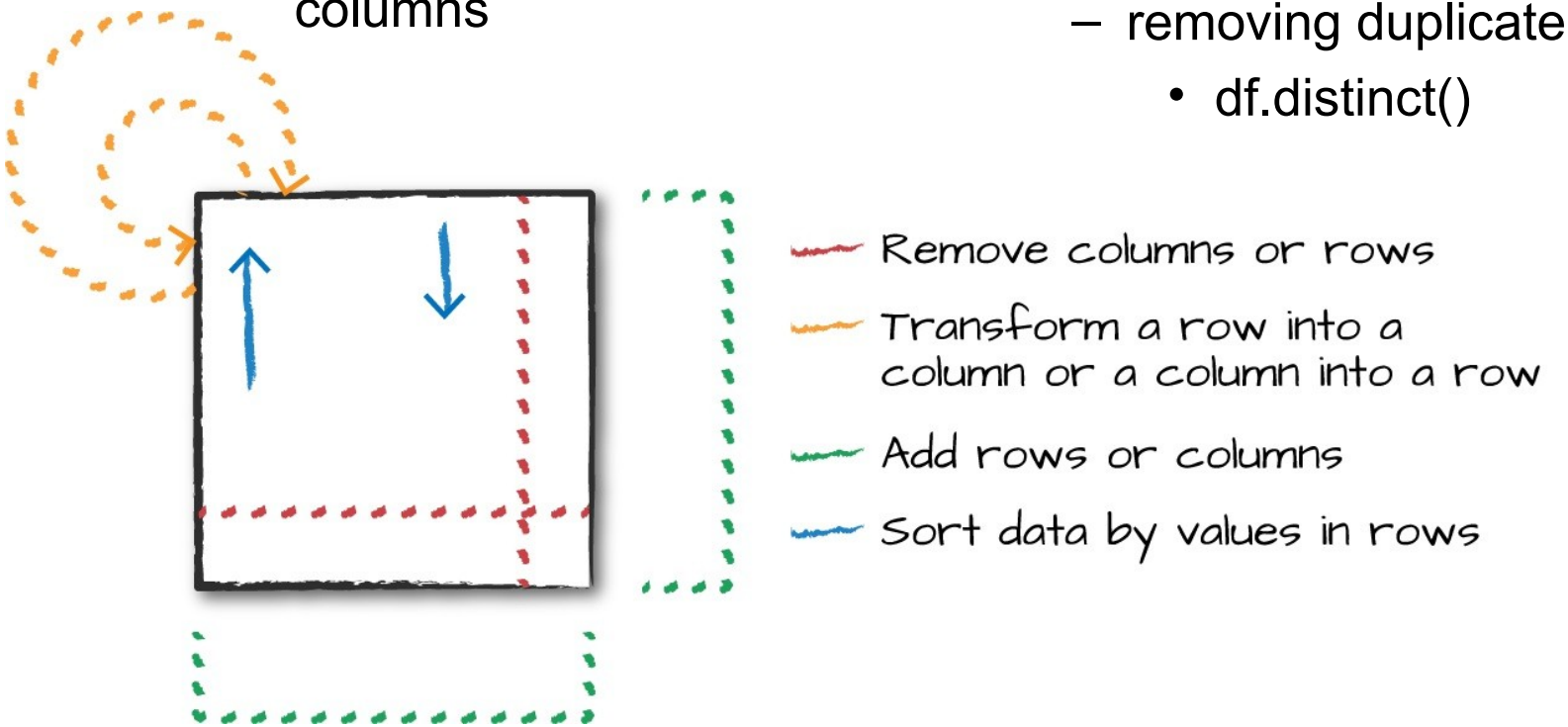
- Expressions:
  - from pyspark.sql.functions import col, expr
    col_var = col('col_name') # ...or expr('col_name')
    df.select(Python_like_expression_with_col_vars)
    # e.g., count_col*1.1 < 100
  - df.select(Python_like_expression_with_cols)
    # e.g., col('count')*1.1 < 100
    # or: expr('count')*1.1 < 100
    - but functions must be imported from pyspark.sql.functions
  - df.selectExpr('SQL-like_expression') # e.g., 'count*1.1 < 100'
    - you can use most SELECT-expressions allowed in SQL
  - from pyspark.sql.functions import expr
    df.select(expr('SQL-like_expression'))

DataFrames ...and some SQL

- Naming new columns:
  - from pyspark.sql.functions import expr
    df.select(expr(‘SQL-like_expression’).alias(new_col_name))
    # e.g., ‘count*1.1 < 100’
  - df.selectExpr(‘SQL-like_expression as new_col_name’)
    # e.g., ‘count*1.1 < 100 as is_small’
- Examples:
  - flight_ex.selectExpr('mean(count) as mean_count').show()
  - flight_ex.select(expr('mean(count)').alias('mean_count')).show()
  - from pyspark.sql.functions import mean
    flight_ex.select(mean(expr('count')).alias('mean_count')).show()
- Shows the pipeline:
  - df.explain()

DataFrames

- Operations/transformations for:
  - adding/removing rows
  - adding/renaming/removing columns
  - column-type casting
  - sorting/filtering rows
  - removing duplicate rows
    - df.distinct()



DataFrames

- Sorting tramsformations:
  - df.orderBy('column_name') # can also use list of columns
  - from pyspark.sql.functions import desc
    df.orderBy(desc('column_name')) # descending sort
- Filtering transformations:
  - df.filter(Python-like_expression)
  - df.filter('SQL-like_expression')
- Remove-duplicate transformations:
  - df.distinct()
- Aggregation transformations:
  - df.groupBy(columns).aggregate_function()

Example transformation pipeline