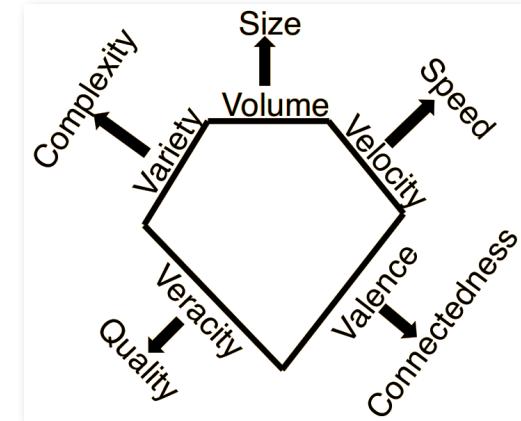




BDMS (Big Data Management System)

- Elastyczny, (semi-)ustrukturyzowany model danych
- Wsparcie dla współczesnych typów danych
tekstowe, temporalne, przestrzenne...
- Dojrzały język zapytań (*przynajmniej* SQL)
- Wydajne czasowo przetwarzanie równoległe
- Szeroki zakres możliwych *rozmiarów* zapytań
- Ciągłe zasilanie danymi; przetwarzanie strumieniowe
- Łatwo skalowalne przechowywanie i procesowanie dużych wolumenów danych; stosowanie dużych klastrów
- Narzędzie wspierające szeroki wachlarz możliwych scenariuszy wykorzystywania zgromadzonych danych – potężne, a proste



Consistency Model

ACID



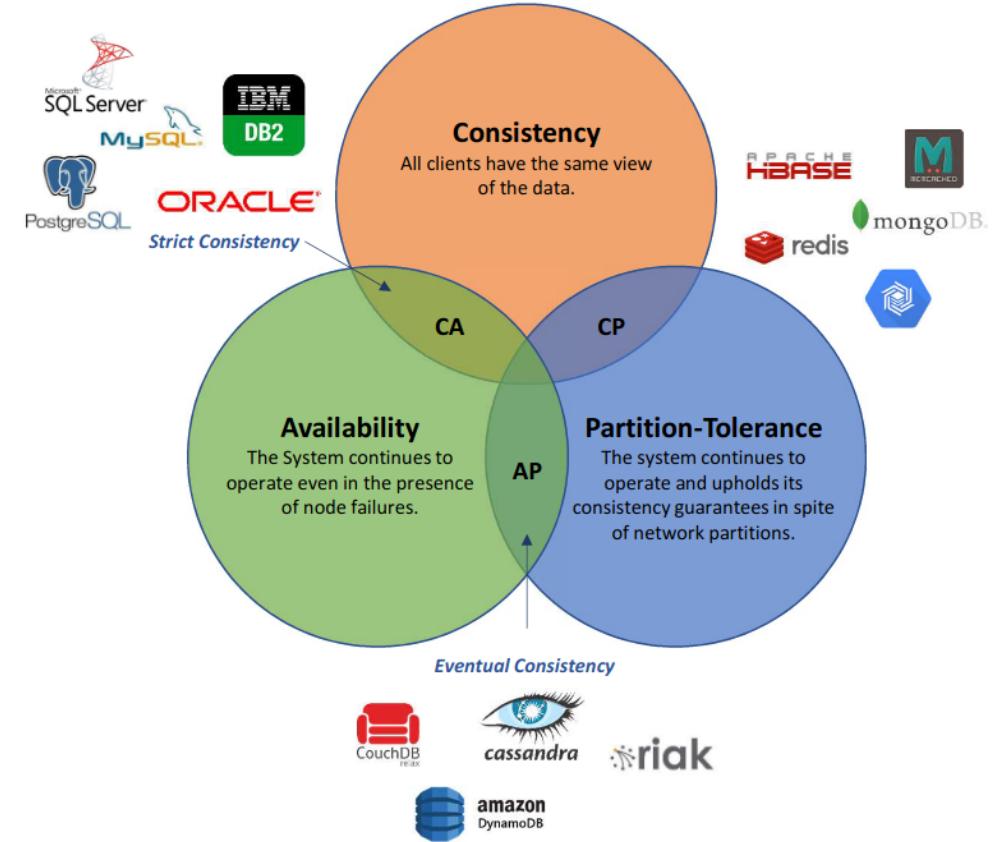
BASE

- Atomic
 - Consistent
 - Isolated
 - Durable
- Basic Availability
 - Soft state
 - Eventual consistency

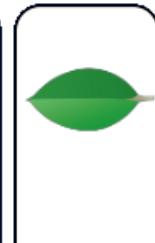
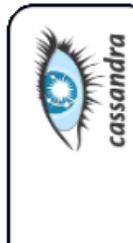
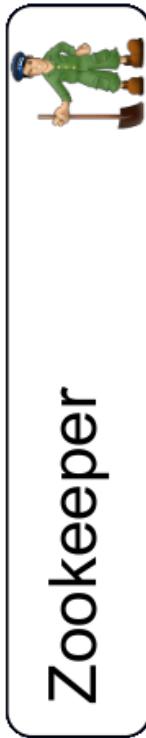
CAP Theorem

„[...] a distributed system can deliver only two of three desired characteristics“:

1. Consistency
2. Availability
3. Partition tolerant



The Hadoop Ecosystem



YARN



HDFS



Big Data Processing Systems

| | Execution Model | Latency | Programming Language |
|---|--|---|----------------------------|
|  | Batch processing using disk storage | High-latency +Fault Tolerance: Replication | Java |
|  | Batch and stream processing using disk or memory storage | Low-latency for small micro-batch size | Scala, Python, Java, R |
|  | Batch and stream processing using disk or memory storage | Low-latency | Java and Scala |
|  | Batch and stream processing | Low-latency | Java and Scala |
|  APACHE STORM™ <small>Distributed • Resilient • Real-time</small> | Stream processing | Very low-latency | Many programming languages |

Cloud Computing Providers



Google Cloud Platform



CloudSigma



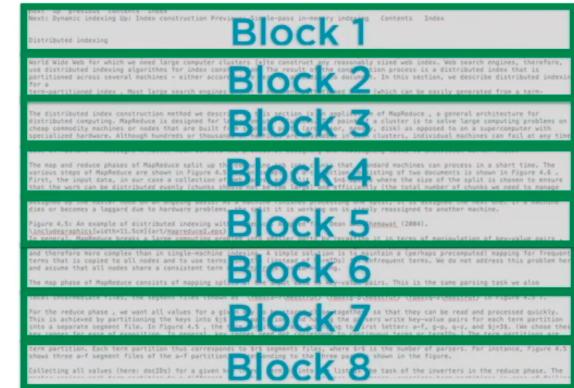
The Hadoop Distributed File System (HDFS)

- Zbudowany w oparciu o zwykły sprzęt.
- Odporny na awarie sprzętu (przyjęte jako norma)
- Odpowiedni dla przetwarzania wsadowego – wyższa przepustowość kosztem wyższych opóźnień
- Wspiera naprawdę duże zbiory danych; większe niż „norma przewiduje”

HDFS File Blocks

Dane są przechowywane w blokach

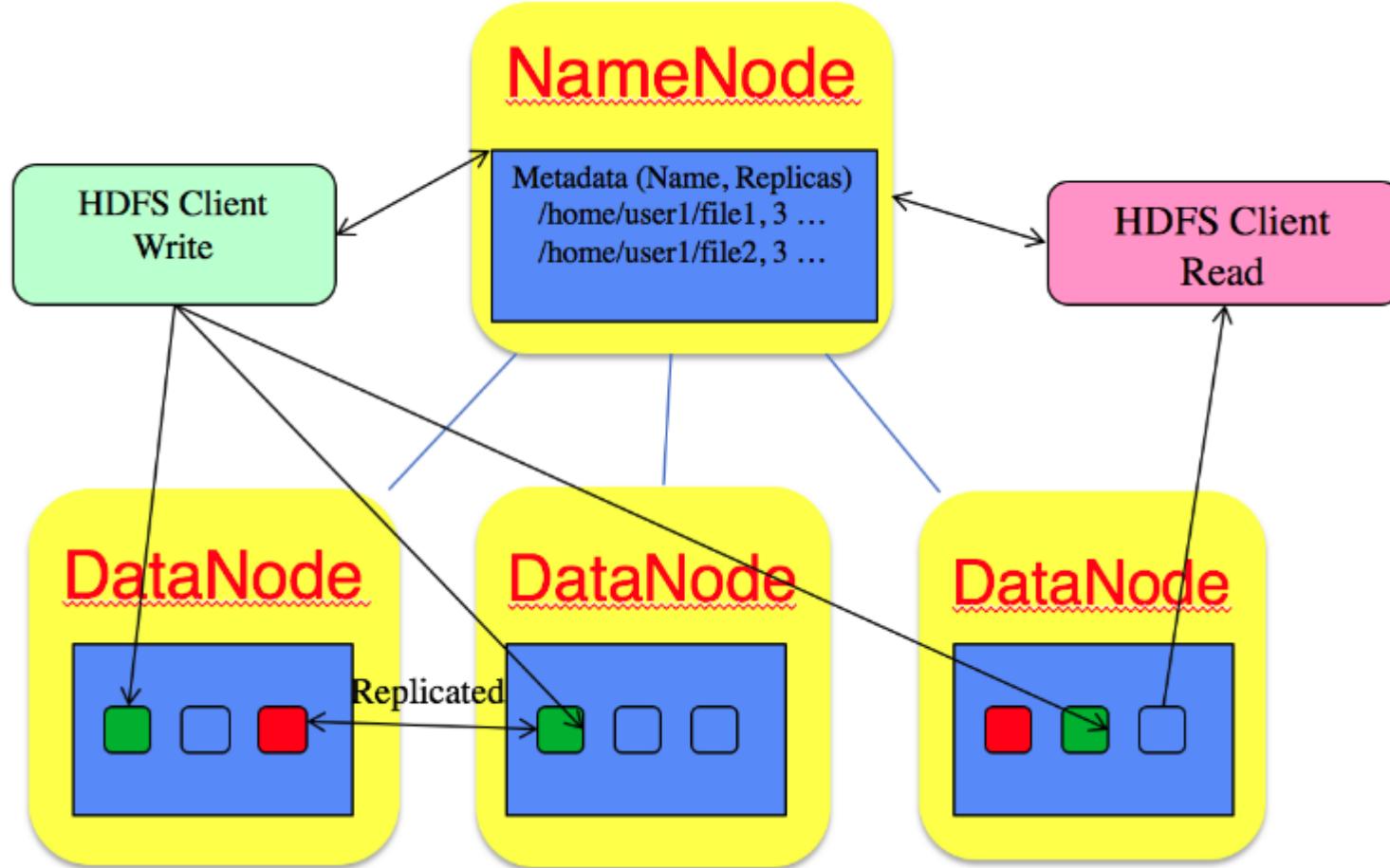
- Mechanizm jest niezależny od rozmiaru plików
- Uproszczone przechowywanie
- Blok jest podstawą replikacji i mechanizmem odporności na awarie
- *Domyślny rozmiar bloku to 128 MB*



Block size is a trade off



HDFS Nodes: Name & Data



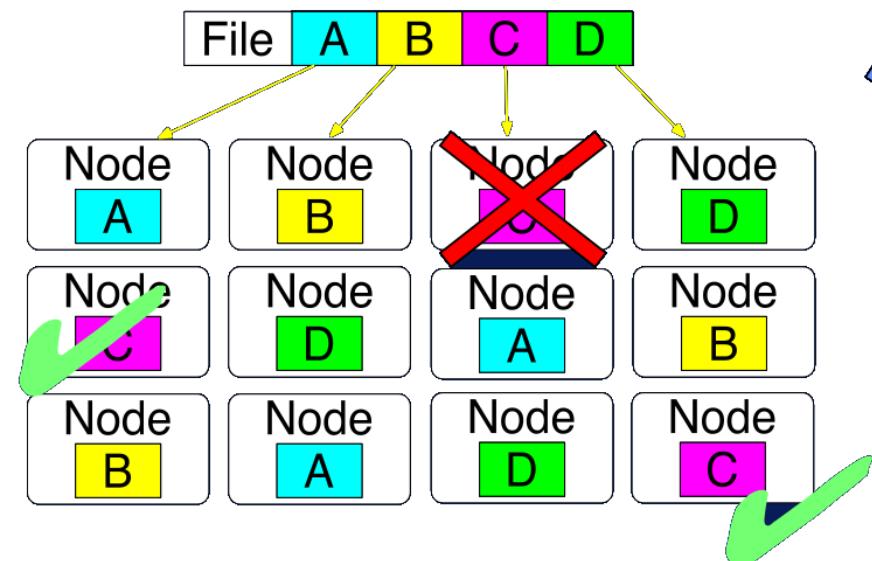
HDFS Concepts

Data partitioning

Scalability

Data replication

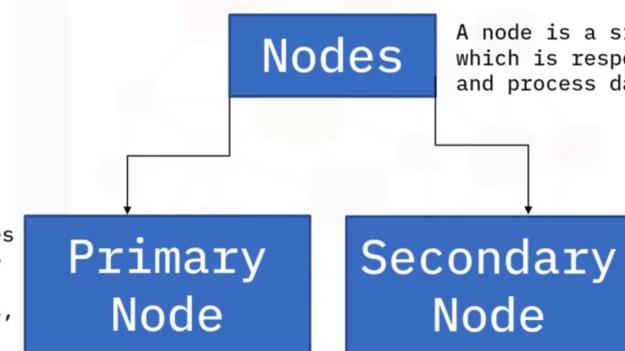
Fault tolerance



Data locality

Nodes

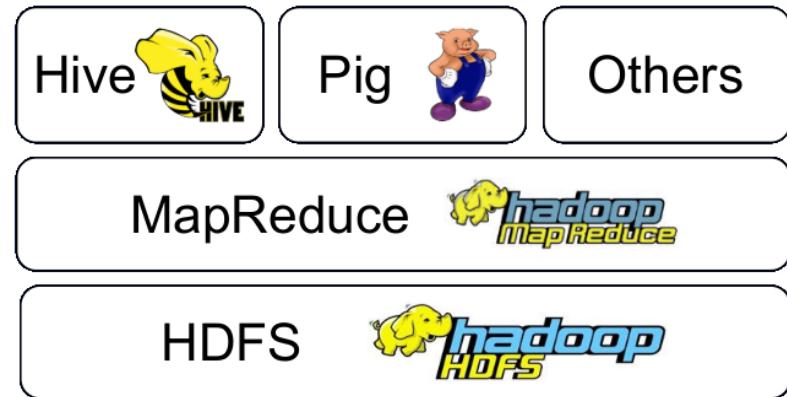
This node regulates
file access to the
clients and
maintains, manages,
and assigns tasks
to the secondary
node



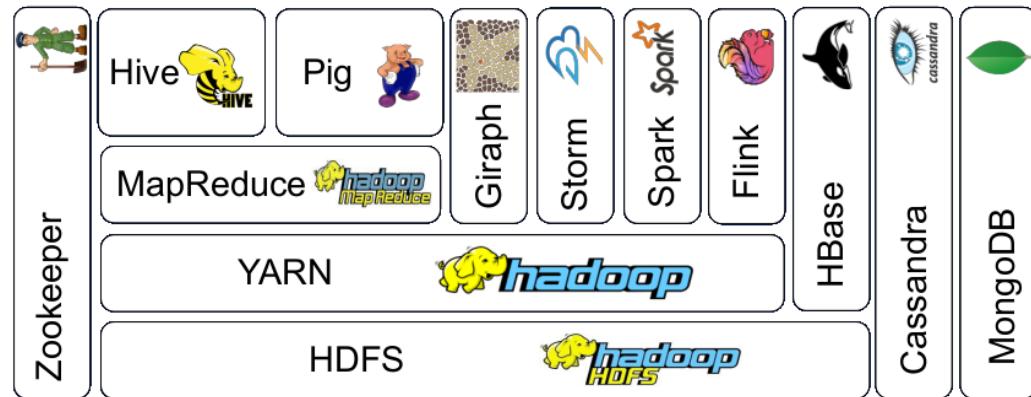
These nodes are the
actual workers in
the HDFS system and
take instructions
from the primary
node

Ewolucja Hadoopa

Hadoop 1.0



Hadoop 2.0



HADOOP 1.0

MAP REDUCE

HDFS

HADOOP 2.0



YARN
(Yet Another Resource Negotiator)

HDFS

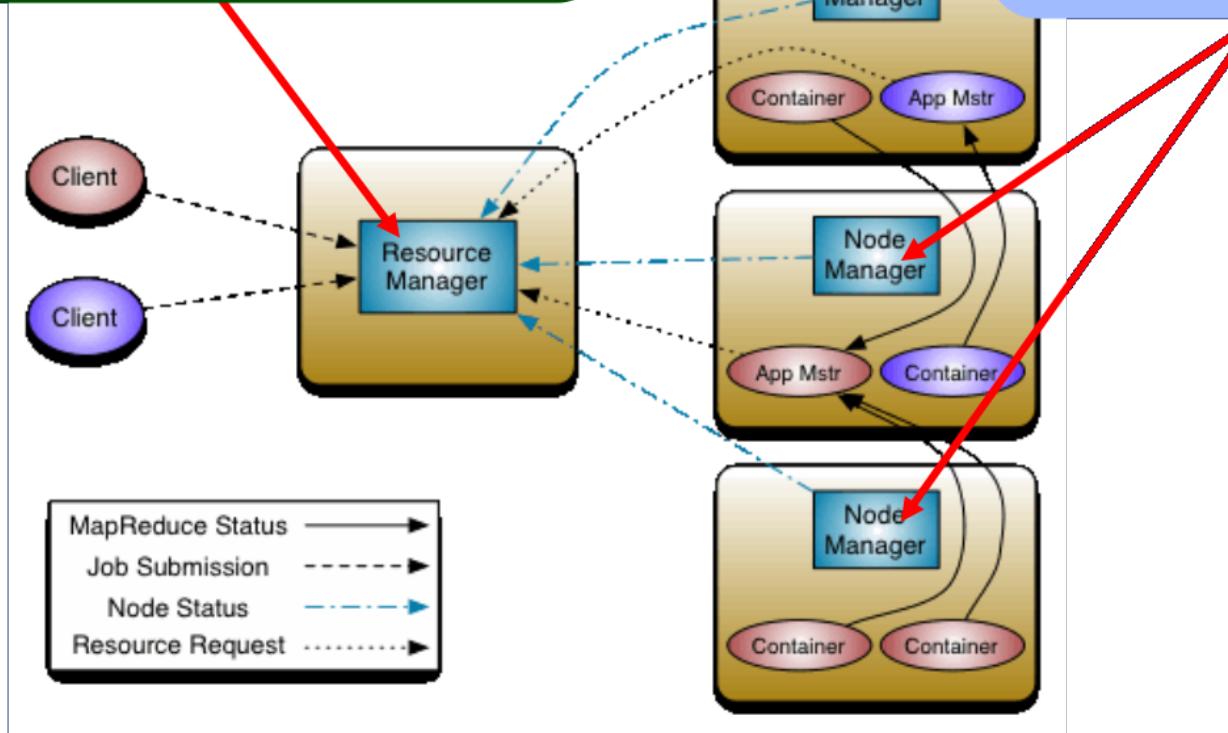
YARN = Resource + Node Managers

Central Resource Manager

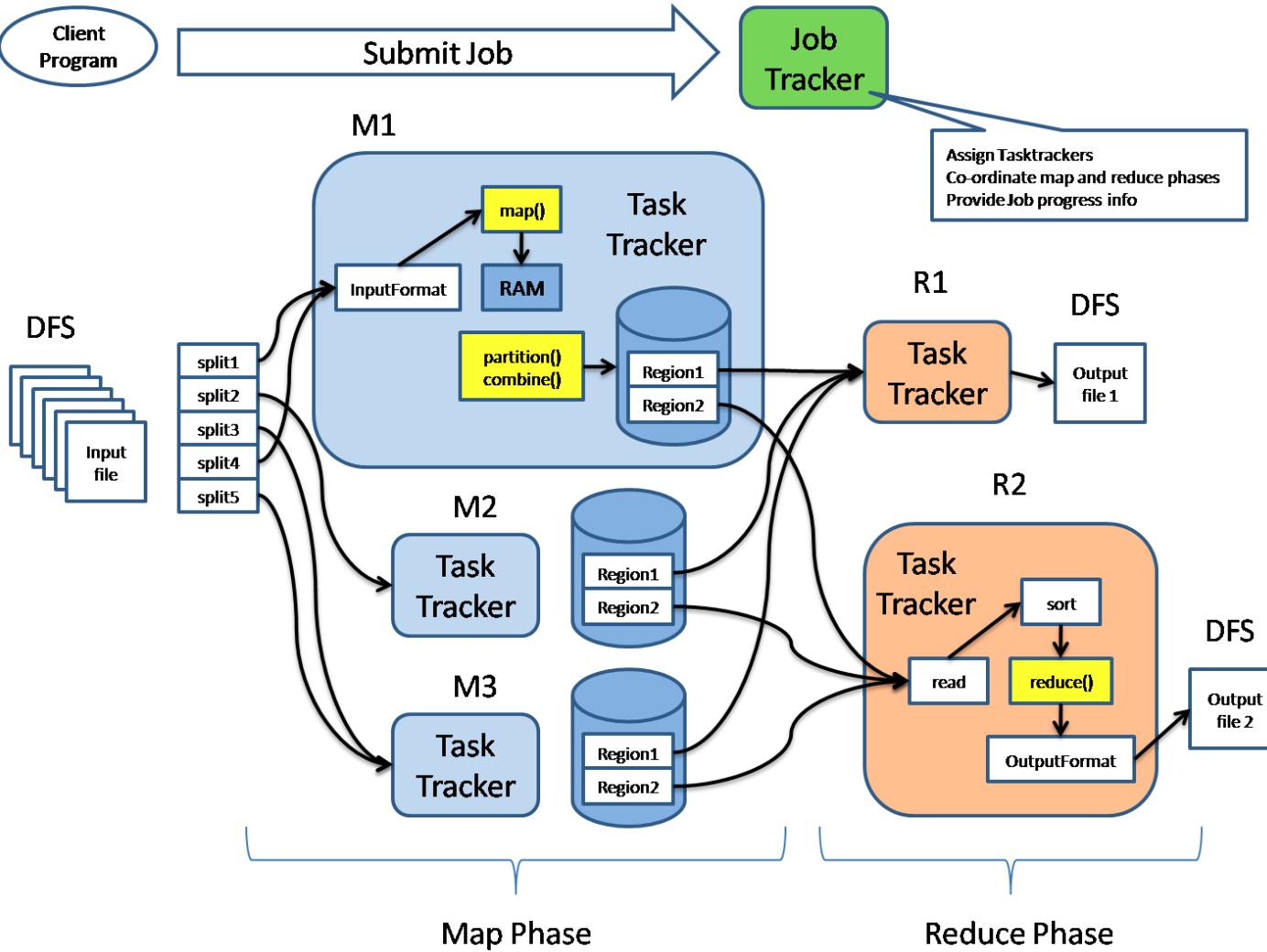
==

ultimate decision maker

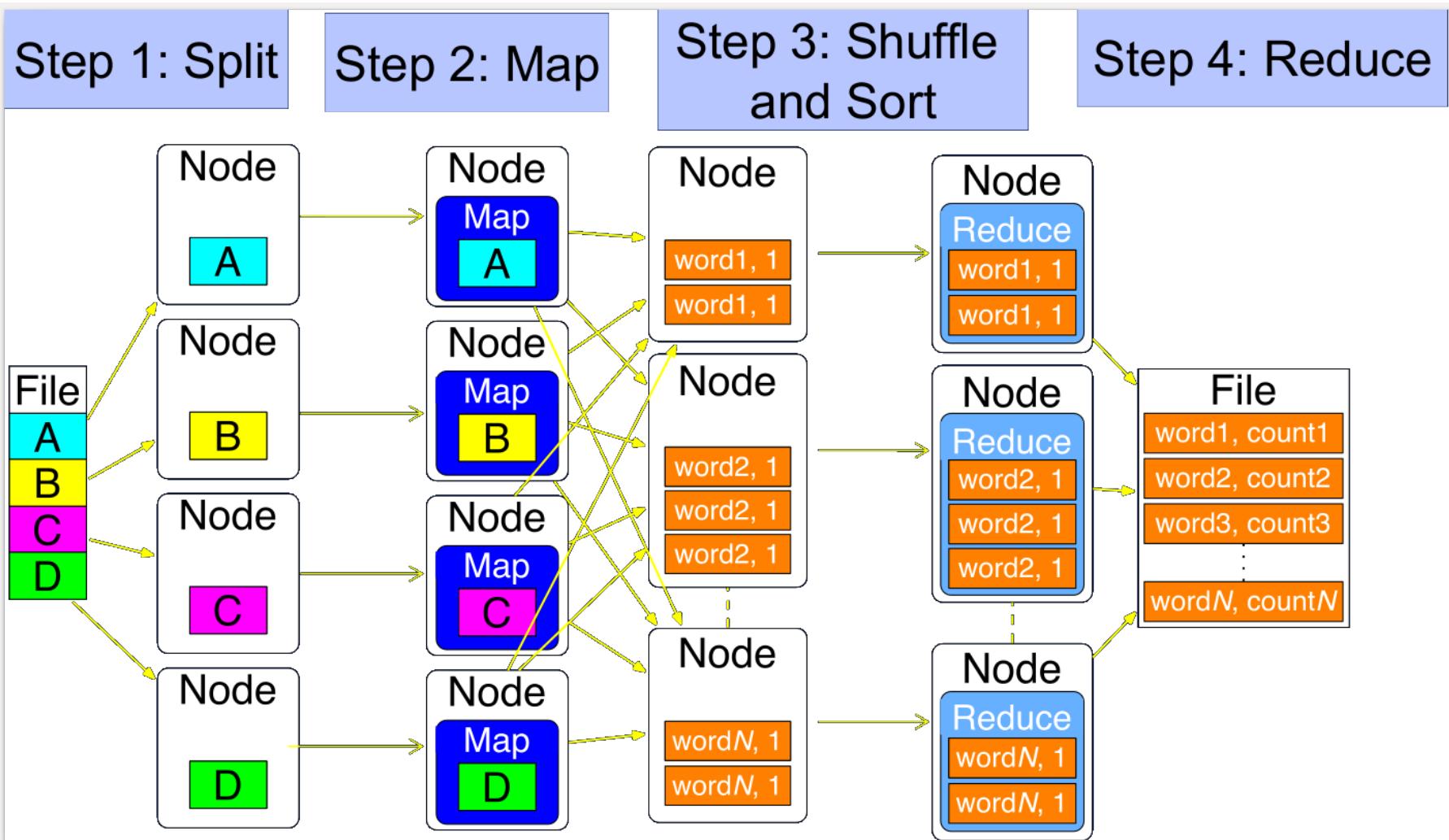
Each machine
gets a Node
Manager



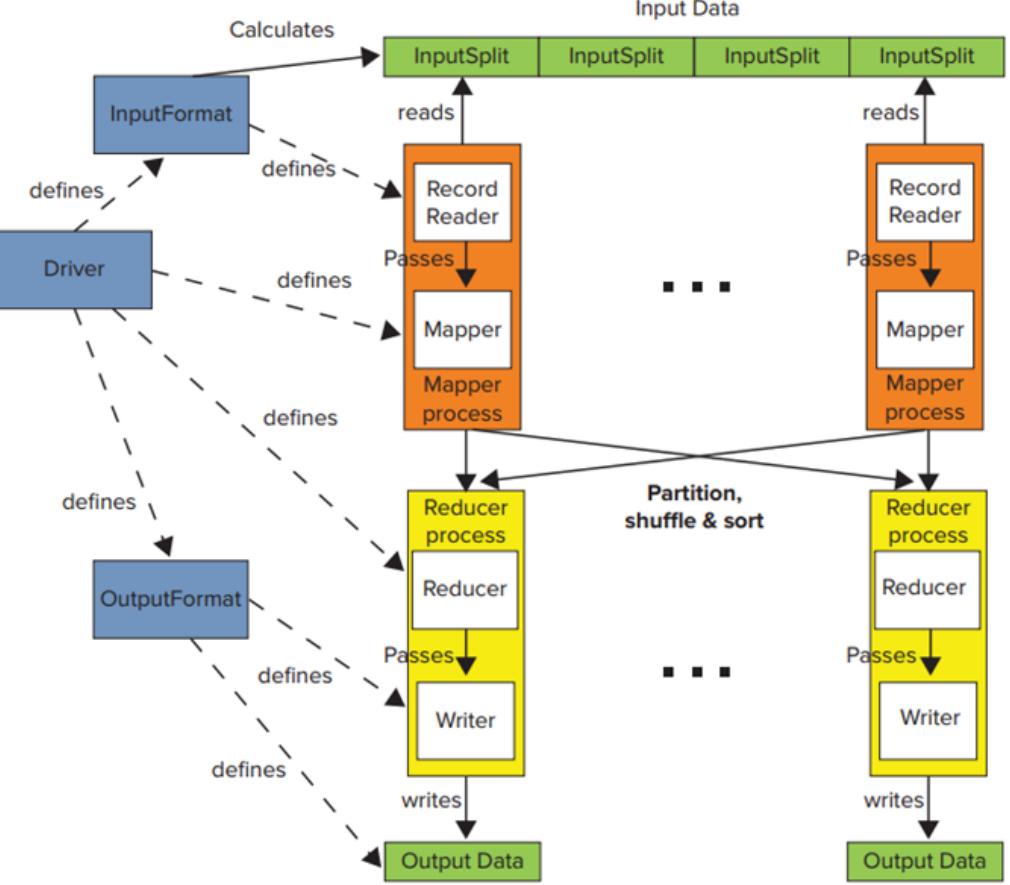
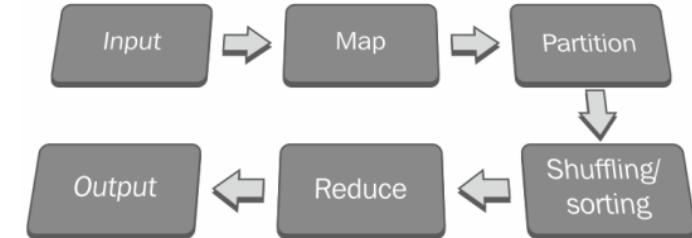
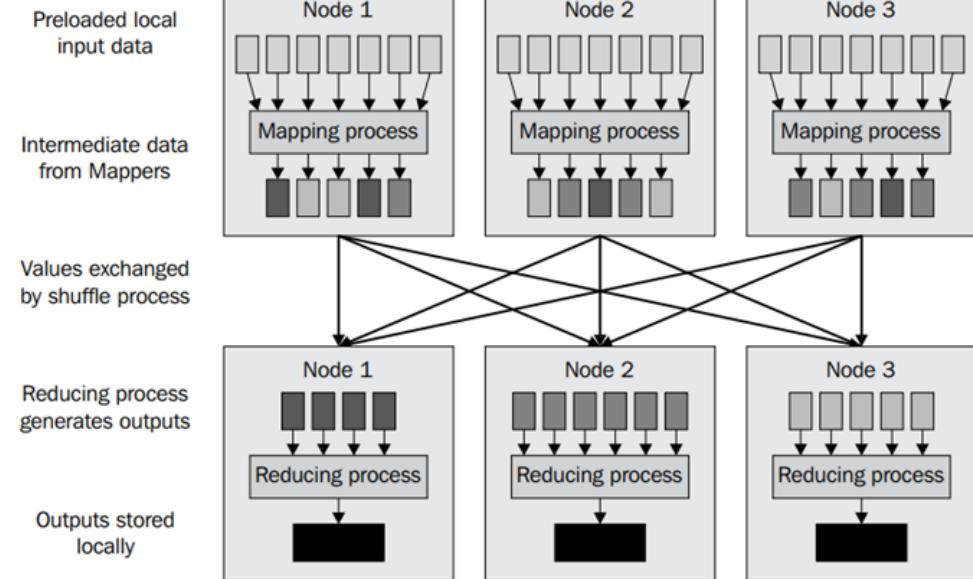
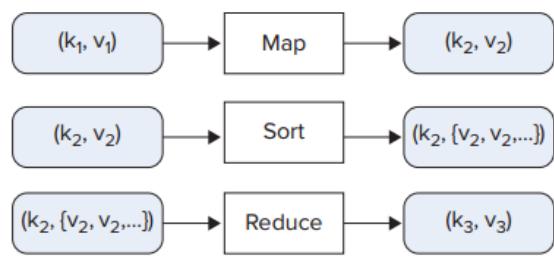
Map Reduce Architecture



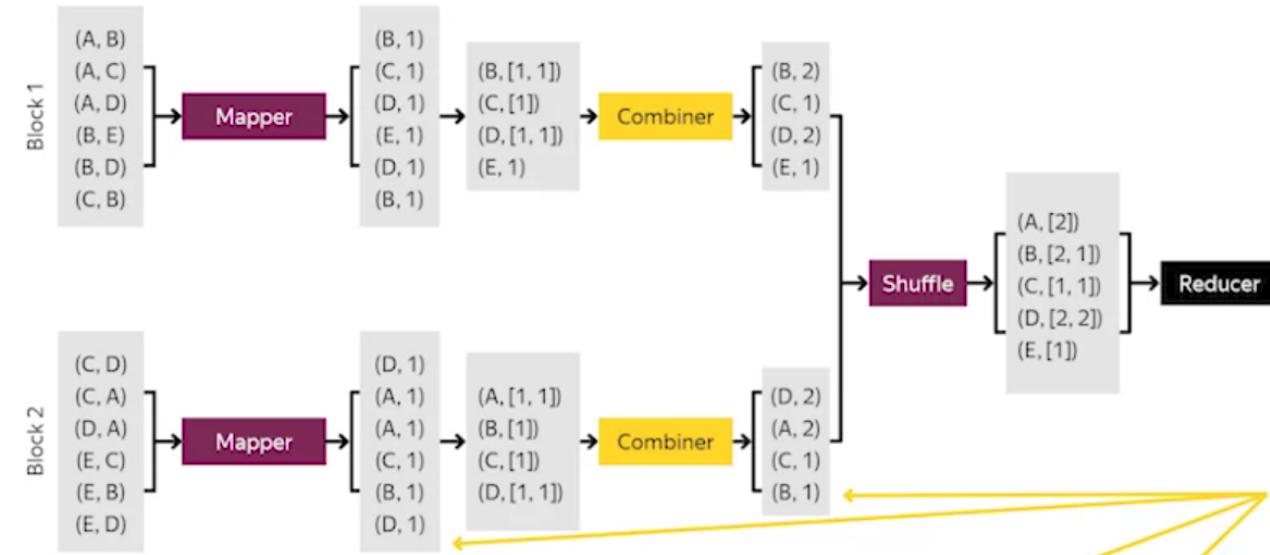
Map Reduce



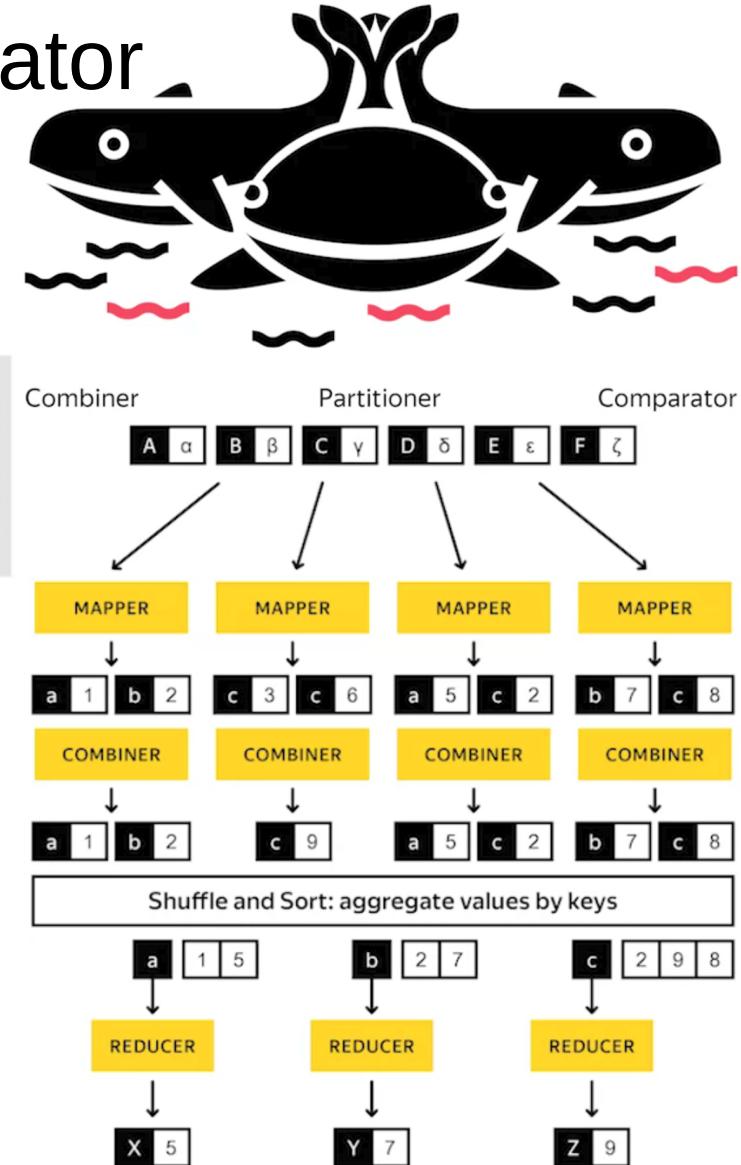
Map Reduce



MR – Combiner, Partitioner, Comparator



- read: $[(k_{in}, v_{in}), \dots]$
- map: $(k_{in}, v_{in}) \rightarrow [(k_{interm}, v_{interm}), \dots]$
- combiner: $[(k_{interm}, [v_{interm}, \dots])] \rightarrow [(k_{interm}, v_{interm}), \dots]$
- Shuffle & Sort: sort and group by k_{interm}
- reduce: $[(k_{interm}, [v_{interm}, \dots])] \rightarrow [(k_{out}, v_{out}), \dots]$





HIVE vs RDBMS



HIVE

RDBMS

Utrzymanie hurtowni danych z wykorzystaniem HQL (HIVE Query Language)

Utrzymanie baz danych z wykorzystaniem SQL (Structured Query Language)

Stosowany w analizie danych statycznych, jak pliki tekstowe zawierające dane historyczne

Stosowane w analize danych online'owych (czasu rzeczywistego), np. z sensorów

Zaprojektowany zgodnie z regułą „*write once, read many*”

Zaprojektowane z myślą o dowolnie potrzebnej liczbie operacji odczytu / zapisu

Maksymalny rozmiar danych to petabajty
 $1.13\text{e}15$ bajtów = 2^{53} bitów

Maksymalny rozmiar danych to terabajty
 $1.10\text{e}12$ bajtów = 2^{43} bitów

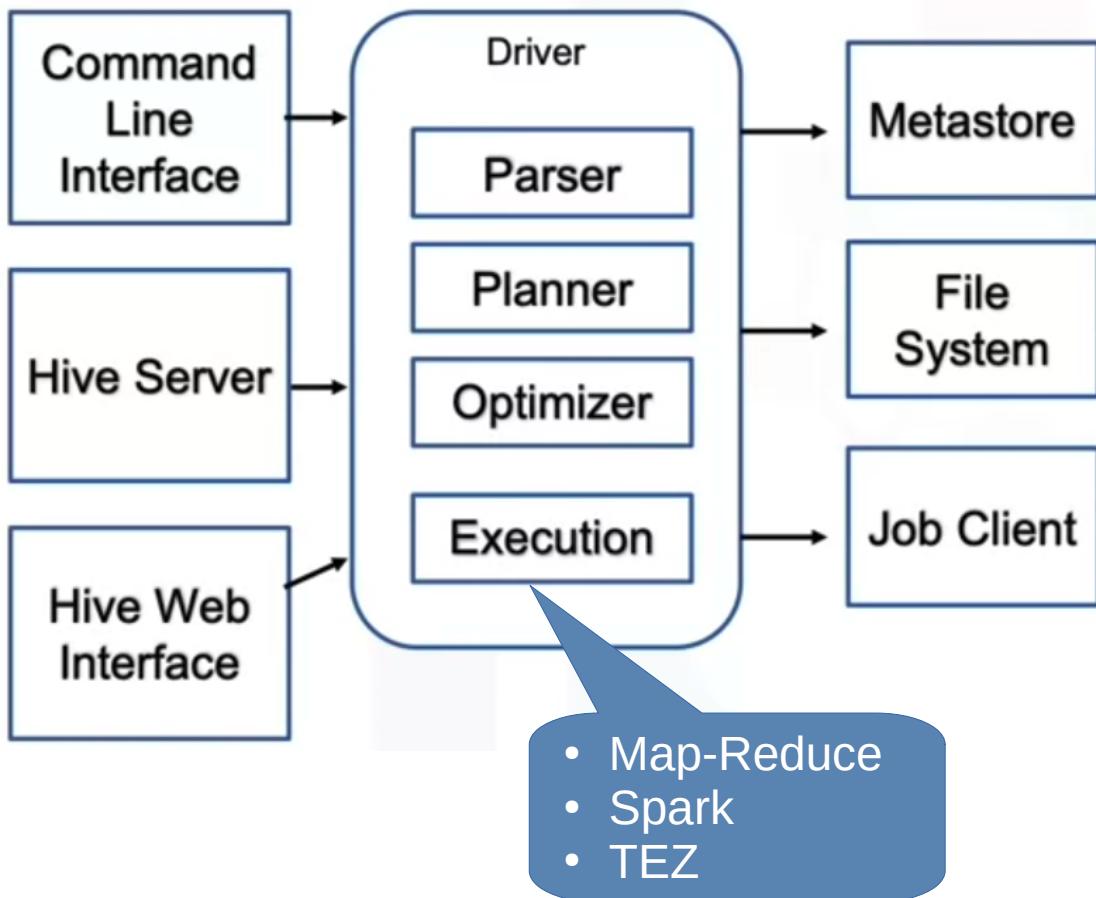
Nie wymusza zgodności ze schematem przy ładowaniu danych, uwzględnia przy odczytzie

Przed ładowaniem danych wymaga i weryfikuje ich zgodność ze schematem

Ma wbudowany mechanizm partycjonowania przechowywanych danych

Nie musi (ale może) mieć wbudowane wsparcia dla partycjonowania przechowywanych danych

Serwisy HIVE



Metastore przechowuje metadane tabel bazodanowych

Driver odbiera zapytania do wykonania

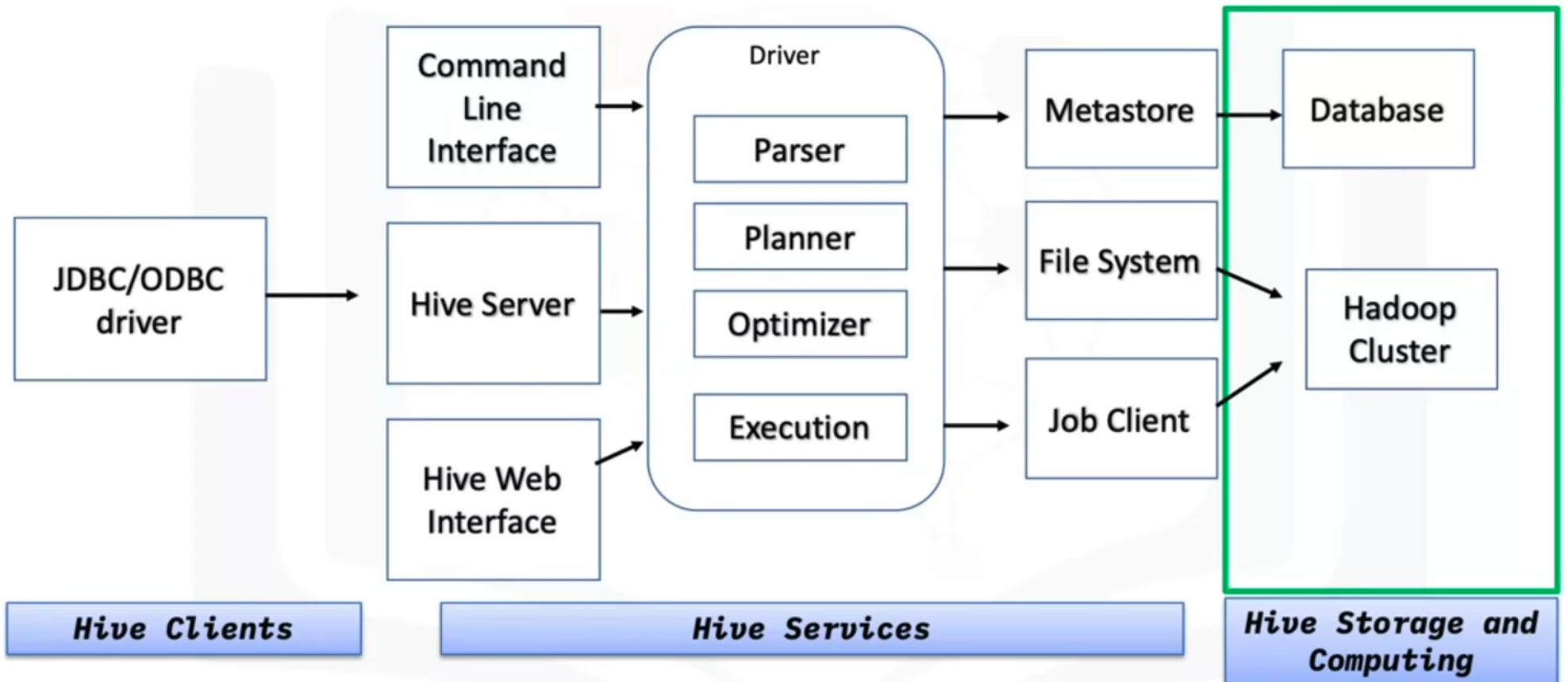
Optimizer odpowiada za efektywny podział zapytania na zadania

Executor wykonuje zapytania po ich podziale przez Optimizer

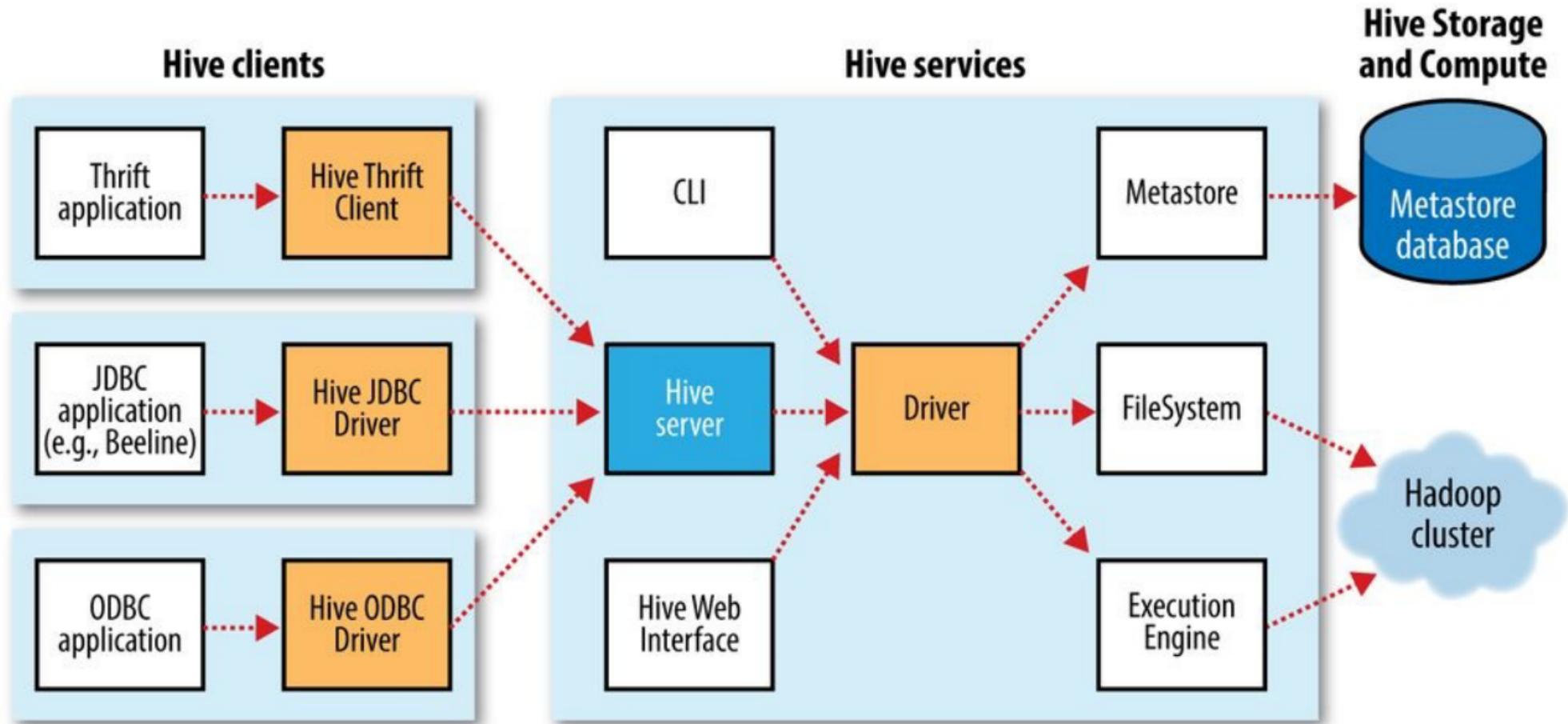
hiveserver2 uruchamia serwer (proces) i udostępnia usługę Thrift (wielodostęp i mechanizm uwierzytelniania).

beeline interfejs konsolowy używający JDBC do połączenia

Architektura platformy HIVE



Architektura platformy HIVE



CLI: hive (process)

```
SLF4J: Class path contains multiple SLF4J  
bindings.  
SLF4J: Found binding in [...]  
SLF4J: Found binding in [...]  
SLF4J: See [...] for an explanation.  
SLF4J: Actual binding is of type [...]  
  
Logging initialized using configuration in [...]  
Hive-on-MR is deprecated in Hive 2 [...]  
hive> use testdb;  
OK  
Time taken: 2.939 seconds  
hive> show tables;  
OK  
airline  
cdr_part  
drivers  
recharges  
temp_cdrs  
temp_drivers  
temp_recharges  
temp_timesheet  
timesheet  
Time taken: 0.137 seconds, Fetched: 9 row(s)  
hive> select count(*) from airline;  
OK  
6855029  
Time taken: 2.063 seconds, Fetched: 1 row(s)  
hive> exit;
```

JDBC: beeline

```
Beeline version 1.2.1.spark2 by Apache Hive  
beeline> !connect jdbc:hive2://master:10000/testdb  
Connecting to jdbc:hive2://master:10000/testdb  
Enter username for jdbc:hive2://master:10000/testdb: testuser  
Enter password for jdbc:hive2://master:10000/testdb: *****  
Connected to: Apache Hive (version 2.3.2)  
Driver: Hive JDBC (version 1.2.1.spark2)  
Transaction isolation: TRANSACTION_REPEATABLE_READ  
0: jdbc:hive2://master:10000/testdb> show tables;  
+-----+  
| tab_name |  
+-----+  
| airline |  
| cdr_part |  
| drivers |  
| recharges |  
| temp_cdrs |  
| temp_drivers |  
| temp_recharges |  
| temp_timesheet |  
| timesheet |  
+-----+  
9 rows selected (0.129 seconds)  
0: jdbc:hive2://master:10000/testdb> select count(*) from airline;  
+-----+  
| _c0 |  
+-----+  
| 6855029 |  
+-----+  
1 row selected (0.175 seconds)  
0: jdbc:hive2://master:10000/testdb> !closeall  
Closing: 0: jdbc:hive2://master:10000/testdb  
beeline> !quit
```

JDBC: JetBrains DataGrip

The screenshot shows the JetBrains DataGrip interface for managing databases. On the left, the database structure is displayed under the 'master' connection, showing a schema with tables like 'airline' and 'columns'. Below this is a 'Services' panel. The main area features a code editor with a syntax-highlighted query:

```
1 ✓ USE testdb;
2
3 ✓ SELECT t.Carrier, t.Airport, t.AvgDelay FROM (
4     SELECT d.Carrier, d.Airport, d.AvgDelay, DENSE_RANK() OVER (PARTITION BY d.Airport ORDER BY d.AvgDelay DESC) AS Rank FROM
5         (SELECT Airport, Carrier, AVG(Delay) AS AvgDelay FROM Airline GROUP BY Airport, Carrier) as d) as t
6 WHERE t.Rank < 3 ORDER BY AvgDelay DESC;
```

The 'Output' tab at the bottom shows the command-line session:

```
[2021-10-26 14:09:07] Connected
testdb> use testdb
[2021-10-26 14:09:08] completed in 228 ms
testdb> USE testdb
[2021-10-26 14:09:08] completed in 96 ms
testdb> SELECT t.Carrier, t.Airport, t.AvgDelay FROM (
    SELECT d.Carrier, d.Airport, d.AvgDelay, DENSE_RANK() OVE
        (SELECT Airport, Carrier, AVG(Delay) AS AvgDelay FROM A
    WHERE t.Rank < 3 ORDER BY AvgDelay DESC
[2021-10-26 14:10:15] 500 rows retrieved starting from 1 in 1 m 6 s
```

The 'Result 1-2' tab displays the query results as a table:

| carrier | airport | avgdelay |
|---------|---------|---------------------|
| YV | ABE | 29.197278911564627 |
| OO | ABE | 20.083333333333332 |
| MQ | ABI | 12.685975609756097 |
| MQ | ABQ | 32.921052631578945 |
| CO | ABQ | 18.677611940298508 |
| EV | ABY | 15.828703703703704 |
| XE | ACK | 35.179245283018865 |
| OH | ACK | 20.854368932038835 |
| MQ | ACT | 9.77312661498708 |
| OO | ACV | 16.374965151937552 |
| EV | ACY | 17.263636363636362 |
| AS | ADK | 12.538461538461538 |
| AS | ADQ | 8.53003003003003003 |

JDBC: Ambari HIVE View

HiveServer2 JDBC Url*

jdbc:hive2://master:10000;username=\${username}

Hive Metastore directory

/user/hive/warehouse

WebHDFS FileSystem URI*

webhdfs://master:9870

DATABASE

Select or search
database/schema

x testdb

Browse ▾

```
1 USE testdb;  
2 SELECT count(*) FROM airline;
```

✓ Execute

Save As

Insert UDF ▾

Visual Explain

RESULTS

LOG

VISUAL EXPLAIN

TEZ UI

Filter column

x



_c0

6855029

testdb ✓

Tables(9)

Search Tables



airline

cdr_part

drivers

recharges

temp_cdrs

temp_drivers

temp_recharges

temp_timesheet

timesheet

HDFS: wiersze (SerDe)

- SerDe to skrót od zwrotu „**S**erializer and **D**eserializer”
- HIVE korzysta z SerDe (oraz formatu plików [File / Storage]) do odczytywania i zapisywania wierszy tabeli

plik HDFS → format pliku wejściowego → <*klucz*, wartość> → **D**eserializer → wiersz
wiersz → **S**erializer → <*klucz*, wartość> → format pliku wyjściowego → plik HDFS
- HIVE **nie jest** właścicielem formatu plików HDFS
- Użytkownicy muszą mieć możliwość bezpośredniego odczytu tabel HIVE z użyciem narzędzi zewnętrznych (np. PIG)
- Użytkownicy muszą być w stanie zapisywać bezpośrednio do plików HDFS, które zostały załadowane jako tabele z wykorzystaniem poleceń "CREATE EXTERNAL TABLE" oraz "LOAD DATA INPATH" (przeniesienie to folderu tabel)

SerDes & Storage Formats

- CSV
- RegEx
- Avro
- ORC
- Parquet
- Thrift
- JsonSerDe
- ...
- TEXTFILE
- SEQUENCEFILE
- ORC
- PARQUET
- AVRO
- RCFILE
- JSONFILE
- ...

| COLUMNS | DDL | STORAGE INFORMATION | DETAILED INFORMATION |
|---------------|--|---------------------|----------------------|
| INFORMATION | VALUE | | |
| SerDe Library | org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe | | |
| Input Format | org.apache.hadoop.mapred.TextInputFormat | | |
| Output Format | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat | | |

| COLUMNS | PARTITIONS | DDL | STORAGE INFORMATION |
|---------------|--|-----|---------------------|
| INFORMATION | VALUE | | |
| SerDe Library | org.apache.hadoop.hive.ql.io.orc.OrcSerde | | |
| Input Format | org.apache.hadoop.hive.ql.io.orc.OrcInputFormat | | |
| Output Format | org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat | | |

Tabele - rodzaje

- zarządzane
managed
- zewnętrzne
external
- tymczasowe
temporary

```
CREATE TABLE table1  
(col1 STRING,  
 col2 INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
```

Internal table (folders deleted when table is dropped)

```
CREATE TABLE table2  
(col1 STRING,  
 col2 INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '  
STORED AS TEXTFILE LOCATION '/data/table2';
```

Default location (/hive/warehouse/table1)

Stored in a custom folder (but still internal, so the folder is deleted when table is dropped)

```
CREATE EXTERNAL TABLE table3  
(col1 STRING,  
 col2 INT)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '  
STORED AS TEXTFILE LOCATION '/data/table3';
```

External table (folders and files are left intact in Azure Blob Store when the table is dropped)

Ładowanie danych do tabeli zarządzanych

- Save data files in table folders (or create table on existing files!)

```
PUT myfile.txt /data/table1
```

- Use the LOAD statement

```
LOAD DATA [LOCAL] INPATH '/data/source' INTO TABLE MyTable;
```

- Use the INSERT statement

```
INSERT INTO TABLE Table2  
SELECT Col1, UPPER(Col2),  
FROM Table1;
```

- Use a CREATE TABLE AS SELECT (CTAS) statement

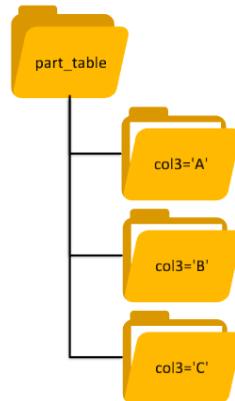
```
CREATE TABLE Table3  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
STORED AS TEXTFILE LOCATION '/data/summarytable'  
AS  
SELECT Col1, SUM(Col2) As Total  
FROM Table1  
GROUP BY Col1;
```

Tabele zarządzane

Partycje

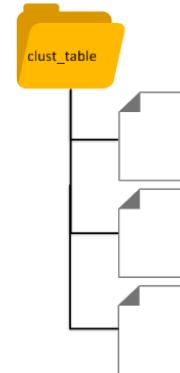
```
CREATE TABLE part_table  
(col1 INT,  
 col2 STRING)  
PARTITIONED BY (col3 STRING);  
  
INSERT INTO TABLE part_table PARTITION(col3='A')  
SELECT col1, col2, col3  
FROM stg_table  
WHERE col3 = 'A';  
  
SET hive.exec.dynamic.partition = true;  
SET hive.exec.dynamic.partition.mode=nonstrict;
```

```
INSERT INTO TABLE part_table PARTITION(col3)  
SELECT col1, col2, col3  
FROM stg_table;
```



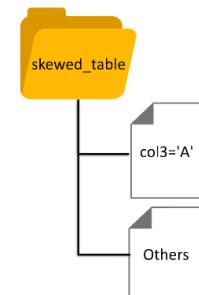
Kubełki buckets

```
CREATE TABLE clust_table  
(col1 INT,  
 col2 STRING,  
 col3 STRING)  
CLUSTERED BY (col3) INTO 3 BUCKETS;  
  
INSERT INTO TABLE clust_table  
SELECT col1, col2, col3  
FROM stg_table;
```



Kubełki skośne skew

```
CREATE TABLE skewed_table  
(col1 INT,  
 col2 STRING,  
 col3 STRING)  
SKEwed BY (col3) ON ('A') [STORED AS DIRECTORIES];  
  
INSERT INTO TABLE skewed_table  
SELECT col1, col2, col3  
FROM stg_table;
```



Język zapytań

Query

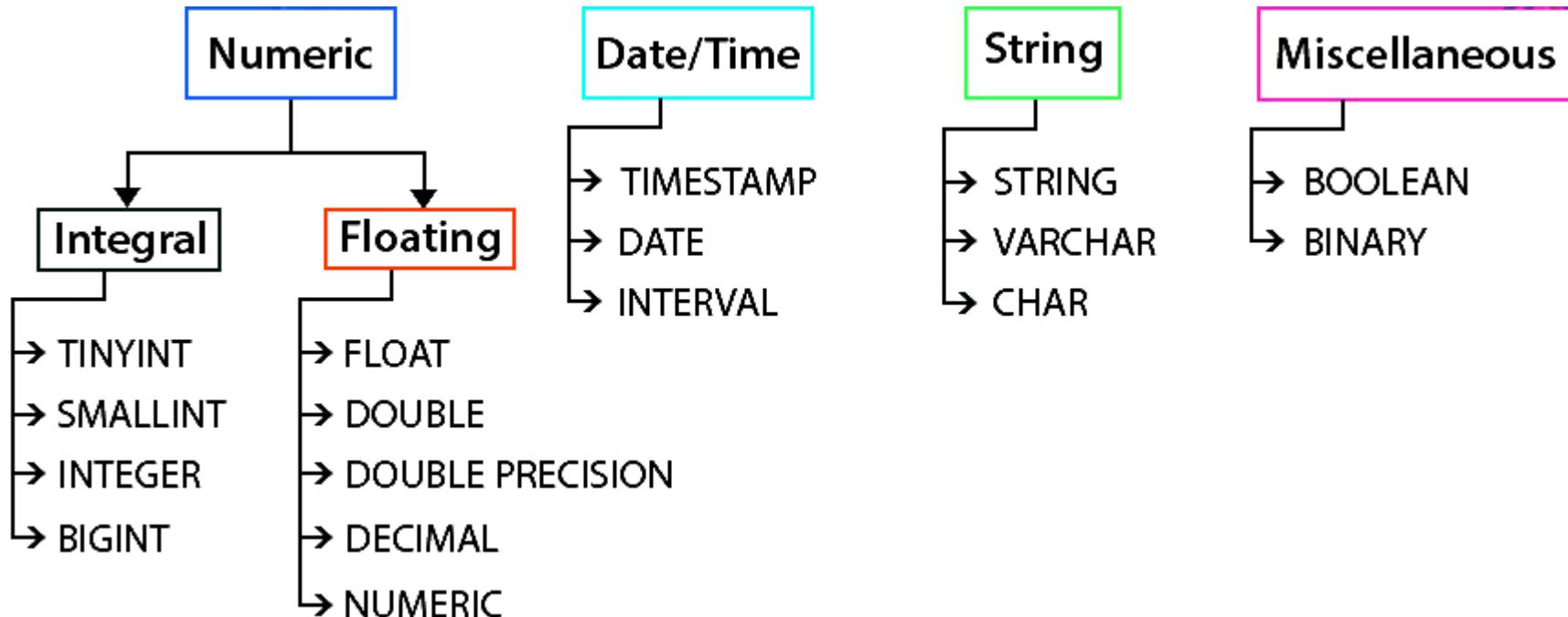
| Function | HiveQL |
|--|--|
| Retrieving information | SELECT from_columns FROM table WHERE conditions; |
| All values | SELECT * FROM table; |
| Some values | SELECT * FROM table WHERE rec_name = "value"; |
| Multiple criteria | SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2"; |
| Selecting specific columns | SELECT column_name FROM table; |
| Retrieving unique output records | SELECT DISTINCT column_name FROM table; |
| Sorting | SELECT col1, col2 FROM table ORDER BY col2; |
| Sorting backward | SELECT col1, col2 FROM table ORDER BY col2 DESC; |
| Counting rows | SELECT COUNT(*) FROM table; |
| Grouping with counting | SELECT owner, COUNT(*) FROM table GROUP BY owner; |
| Maximum value | SELECT MAX(col_name) AS label FROM table; |
| Selecting from multiple tables (Join same table using alias w/"AS") | SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name); |

Metadata

| Function | MySQL | HiveQL |
|----------------------------------|--------------------------|--------------------------------------|
| Selecting a database | USE database; | USE database; |
| Listing databases | SHOW DATABASES; | SHOW DATABASES; |
| Listing tables in a database | SHOW TABLES; | SHOW TABLES; |
| Describing the format of a table | DESCRIBE table; | DESCRIBE (FORMATTED EXTENDED) table; |
| Creating a database | CREATE DATABASE db_name; | CREATE DATABASE db_name; |
| Dropping a database | DROP DATABASE db_name; | DROP DATABASE db_name (CASCADE); |

Typy danych

Proste



Złożone

Array

Map

Struct

Union



HPL/SQL – proceduralny SQL

- Rozwiązanie implementujące język SQL z rozszerzeniami proceduralnymi dla narzędzi SQL-on-Hadoop (HIVE, Impala, SparkSQL) oraz innych RDMBS i NoSQL.
- Osobny, dedykowany silnik kompatybilny z PL/SQL, MySQL, T-SQL, PostgreSQL...
- Możliwość pisania funkcji, procedur, przetwarzania sekwencyjnego (pętle, kursory).
- Część Apache HIVE od wersji 2.0
- Integracja z HIVE:
 - helper function org.apache.hive.hql.udf.UDF
 - ← JDBC + hql-site.xml

```
18 // HPL/SQL Procedural SQL Extension Grammar
19 grammar Hplsql;
20
21 program : block EOF;
22
23 block : ((begin_end_block | stmt) T_GO?)+;           // Multiple consecutive blocks/statements
24
25 begin_end_block :
26     declare_block? T_BEGIN block exception_block? block_end
27     ;
28
29 single_block_stmt :                                // Single BEGIN END block (but nested blocks are possible) or single statement
30     T_BEGIN block exception_block? block_end
31     | stmt T_SEMICOLON?
32     ;
33
34 block_end :
35     {!_input.LT(2).getText().equalsIgnoreCase("TRANSACTION")}? T_END
36     ;
37
38 proc_block :
39     begin_end_block
40     | stmt+ T_GO?
41     ;
42
43 stmt :
44     assignment_stmt
45     | allocate_cursor_stmt
46     | alter_table_stmt
47     | associate_locator_stmt
48     | begin_transaction_stmt
49     | break_stmt
```

HPL/SQL – ciekawostka

Inne silniki SQL-on-Hadoop



Apache
Impala

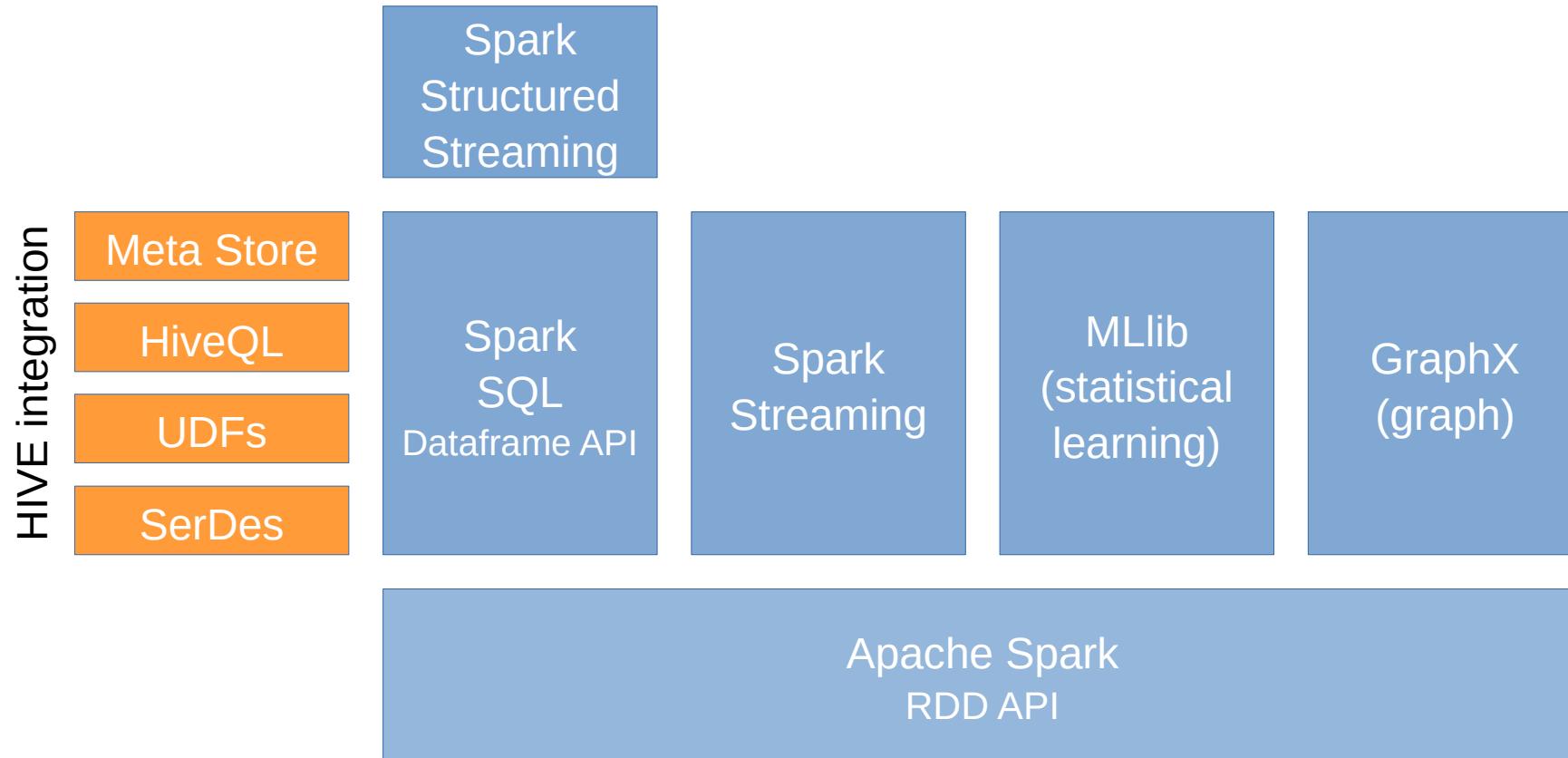


prestoDB

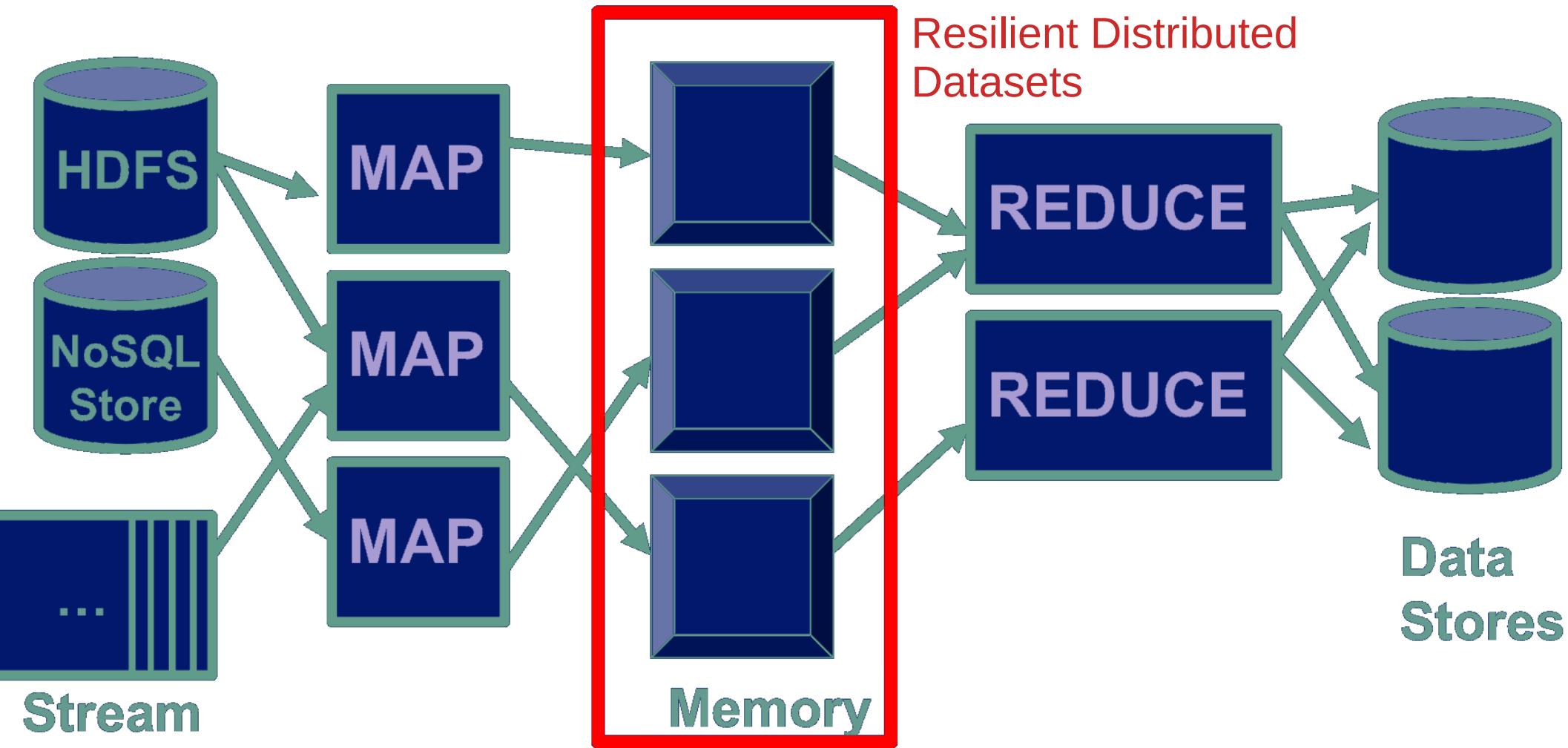




Komponenty Apache Spark



Spark – RDD



Resilient Distributed Datasets

Partycjonowany zbiór rekordów tylko do odczytu

- **Dataset** – zbiór wartości atomowych (prymitywnych), rekordów, krotek, obiektów. Stworzony z HDFS, S3, HBase, JSON, ciągów tekstuowego, hierarchii folderów, lub jako transformacja innego RDD.
- **Distributed** – rozproszony pomiędzy maszyny (węzły) klastra; podzielony na partycje, atomowe podzbiory danych.
- **Resilient** – odtwarzalny w przypadku błędów, np. awarii węzła, czy zbyt wolnego przetwarzania; śledzenie historii każdej partycji, ponowne przeliczanie transformacji.

Interfejs RDD[T]

- Jest silnie typowany: RDD[T] – zbiór elementów typu T.

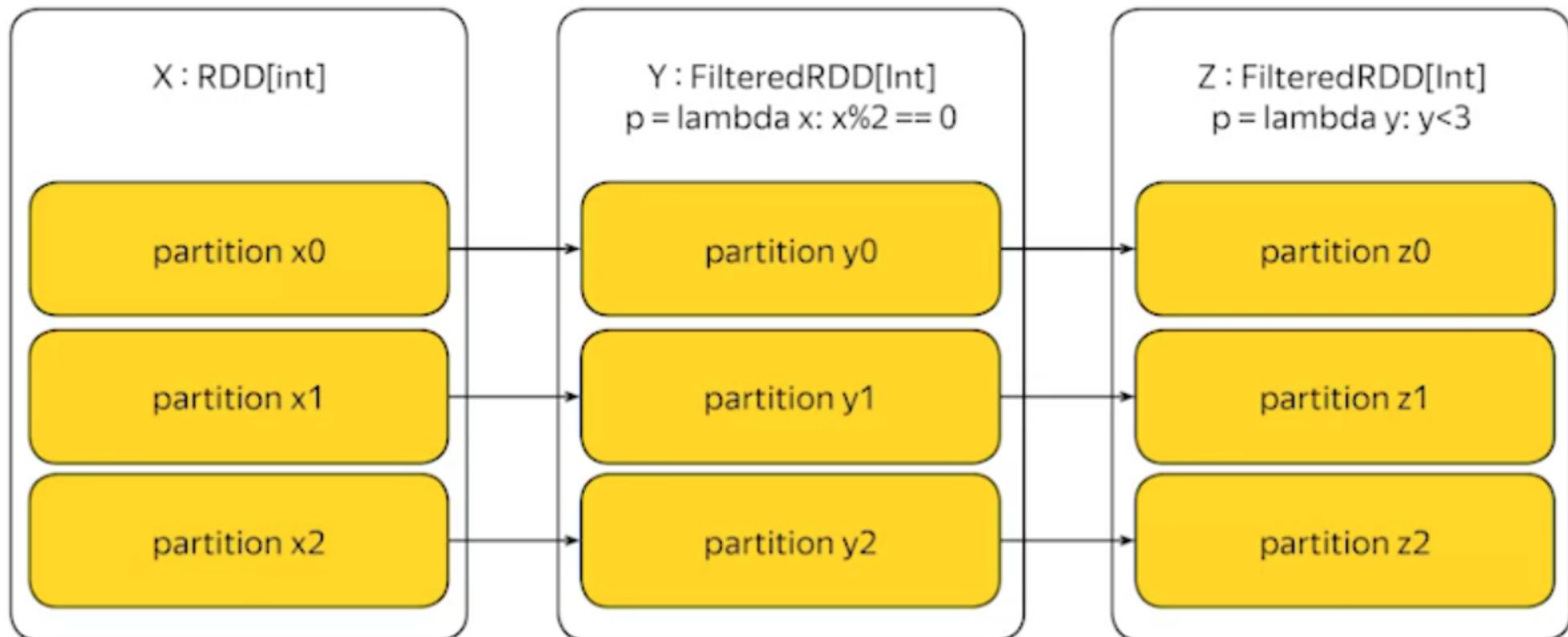
Zbiór **musi** implementować:

- **partitions()**→Array[Partition]
- **iterator(p: Partition, parents: Array[Iterator[_]])**→Iterator[T]
- **dependencies()**→Array[Dependency]

Mожет również implementować inne funkcje pomocnicze.

Graf zależności partycji

› $Z = X.\text{filter}(\lambda x: x \% 2 == 0).\text{filter}(\lambda y: y < 3)$



Resilient Distributed Datasets

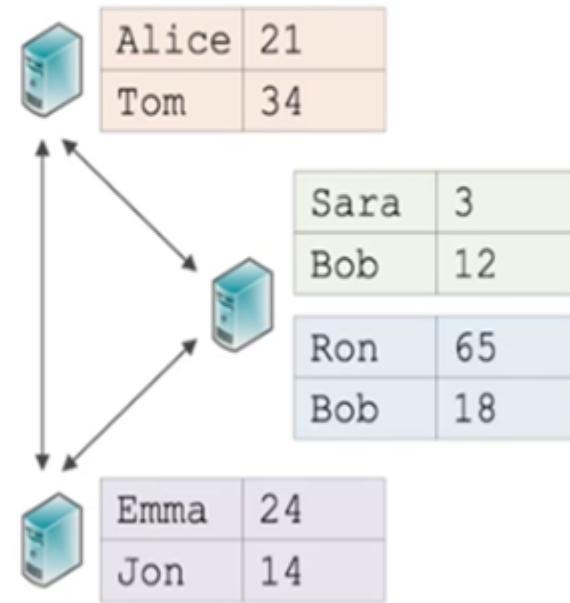
- **Immutable** – zbiory nie podlegają modyfikacjom, tylko transformacjom do nowych zbiów.
- **In-memory** – dane przechowywane w pamięci (w miarę możliwości).
- **Cacheable** – całość danych (domyślnie) trzymana w pamięci.
- **Persist** – może zapisywać i przechowywać wyniki operacji.
- **Partitioned** – dzieli na podzbiory, żeby procesować wolumeny za duże na pojedynczą maszynę.
- **Typed** – silnie typowany RDD[$\textcolor{red}{T}$].
- **Lazy evaluated** – dane zawarte w RDD nie są dostępne lub przekształcane do czasu wykonania akcji, która wyzwala proces przetwarzania.

Partycjonowanie

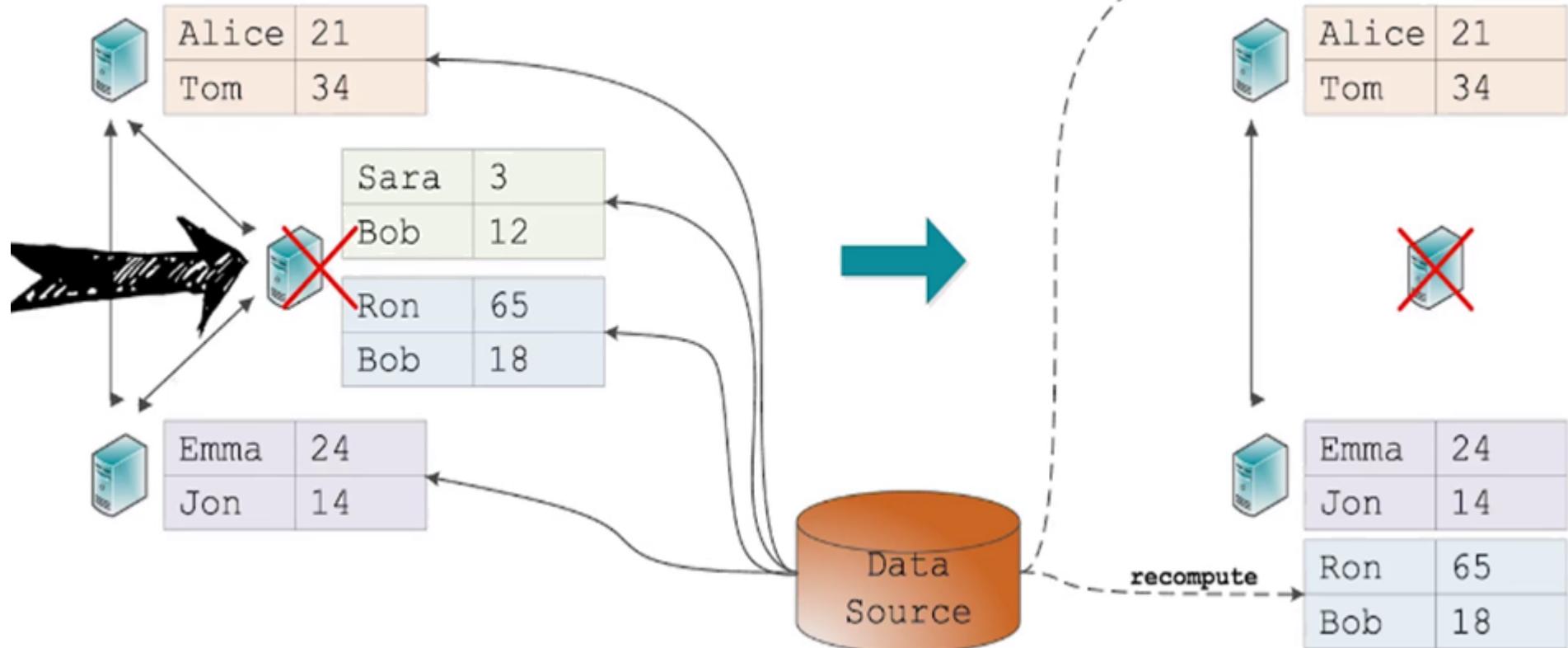
- Spark automatycznie partycjonuje dane.
- Spark automatycznie dystrybuuje (rozprasza) partycje pomiędzy węzłami klastra.

RDD[(String, Int)]

| | String | Int |
|-------|--------|-----|
| Alice | 21 | |
| Tom | 34 | |
| Sara | 3 | |
| Bob | 12 | |
| Ron | 65 | |
| Bob | 18 | |
| Emma | 24 | |
| Jon | 14 | |



Odporność na błędy



Tylko do odczytu

- RDD jest tylko do odczytu, ale...
- RDD może być transformowane (w nowe RDD).

RDD[(String, Int)]

| String | Int |
|--------|-----|
| Alice | 21 |
| Tom | 34 |
| Sara | 3 |
| Bob | 12 |
| Ron | 65 |
| Bob | 18 |
| Emma | 24 |
| Jon | 14 |

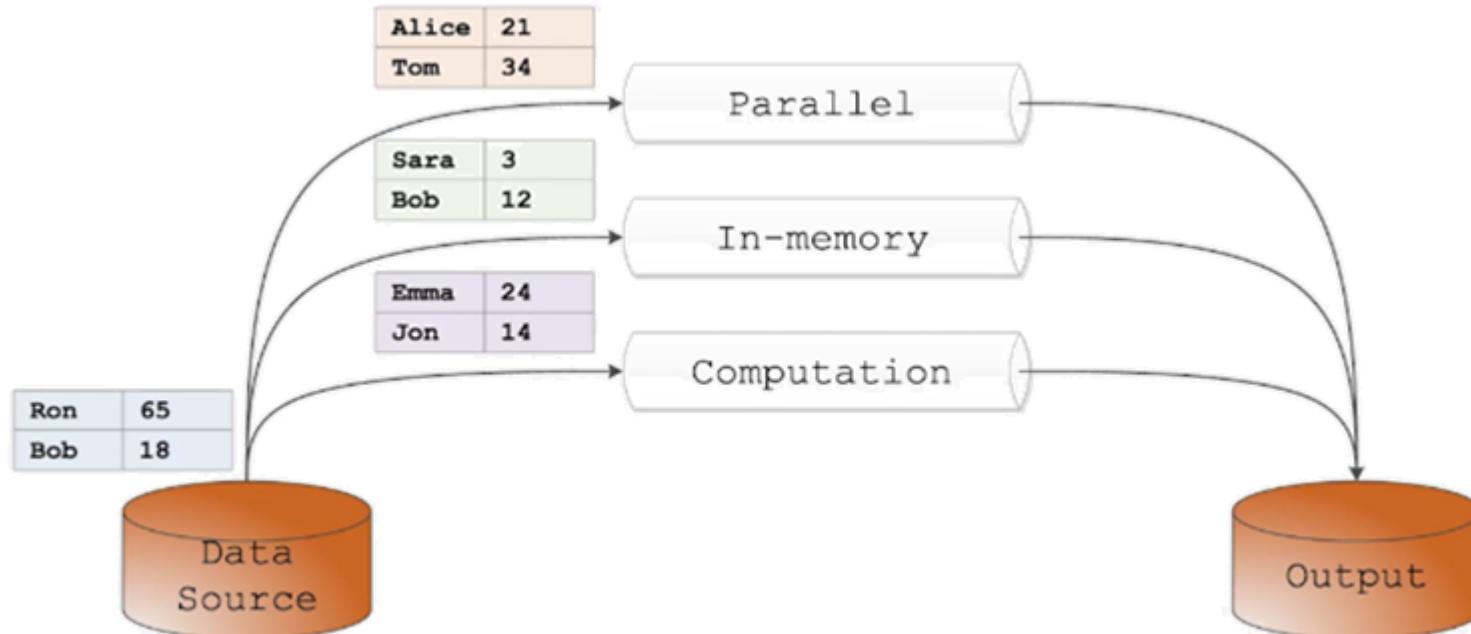
`filter{case (name,age) => age >= 21}`

RDD[(String, Int)]

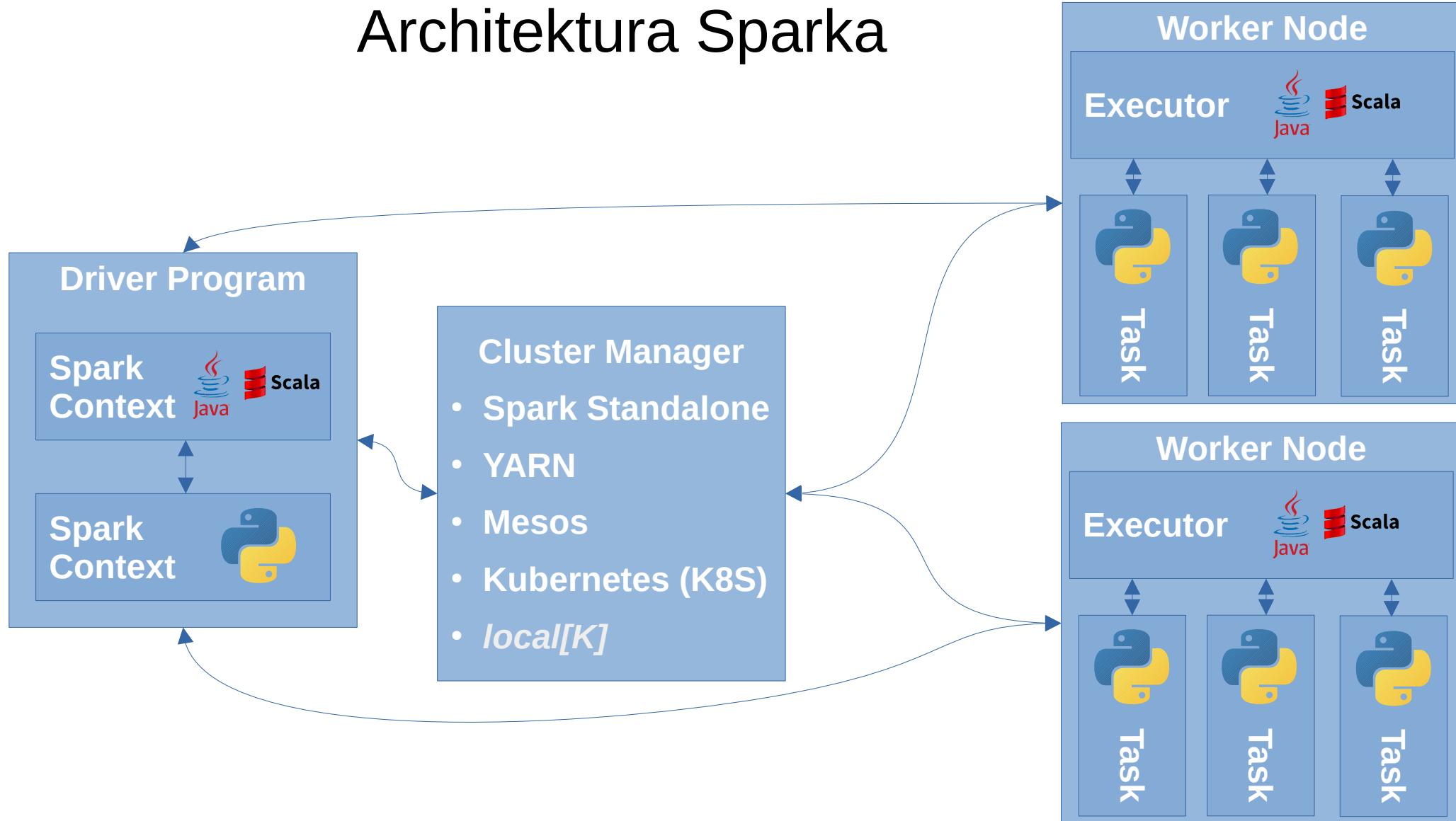
| String | Int |
|--------|-----|
| Alice | 21 |
| Tom | 34 |
| Ron | 65 |
| Emma | 24 |

Przetwarzanie w pamięci

- Partycje RDD są przetwarzane w pamięci.
- RDD (jako całość) nie musi się w pamięci mieścić.



Architektura Sparka



Transformacje

- Wszystkie transformacje są **leniwe**, czyli nie wyliczają wyniku od razu po zdefiniowaniu.
- Są zapisem przekształceń, jakim w kolejności poddawany jest zbiór wejściowy.
- Są wykonywane (przeliczane) jedynie wtedy, kiedy **akcja** wymaga aby rezultat został zwrócony do programu sterującego (driver program). Jest to element optymalizacji wydajności Sparka.

Transformacje – persystencja

- Domyślnie, każda transformacja RDD jest przeliczana za każdym wywołaniem następującej akcji. A czkolwiek...
- RDD (jako wynik transformacji) może być zapamiętany (zapisany) z wykorzystaniem metody **persist** lub **cache**. Spark przechowuje takie dane na klastrze celem szybszego dostępu przy kolejnych akcjach.

Transformacje – persystencja

`cache() == persist(StorageLevel.MEMORY_ONLY)`

| Storage Level | Meaning |
|---|---|
| MEMORY_ONLY | Store RDD as serialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level. |
| MEMORY_AND_DISK | Store RDD as serialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed. |
| MEMORY_ONLY_SER (Java and Scala) | Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than serialized objects, especially when using a fast serializer, but more CPU-intensive to read. |
| MEMORY_AND_DISK_SER (Java and Scala) | Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed. |
| DISK_ONLY | Store the RDD partitions only on disk. |
| MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc. | Same as the levels above, but replicate each partition on two cluster nodes. |
| OFF_HEAP (experimental) | Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled. |

Popularne transformacje

- **map(func)** – dla każdego elementu, nowy RDD zawiera jego *obraz* zwrócony przez **func**. Liczba elementów pozostaje bez zmian.
- **flatMap(func)** – dla każdego elementu, nowy RDD zawiera wszystkie obiekty zwrócone przez **func** jako kolekcja wartości. Pomiędzy RDD zachodzi relacja jeden-do-wielu.
- **filter(predicate)** – nowy RDD zawiera tylko te elementy, dla których **predicate** zwróciła wartość true.
- **sample** – losowy podzbiór elementów RDD z albo bez powtórzeń.
- **distinct** – eliminuje duplikaty.

Popularne transformacje

- mapPartitions
- mapPartitionsWithIndex
- mapValues
- union
- intersection
- groupByKey
- reduceByKey
- aggregateByKey
- sortByKey
- join
- cogroup
- cartesian
- coalesce
- repartition

Map-Reduce w Spark

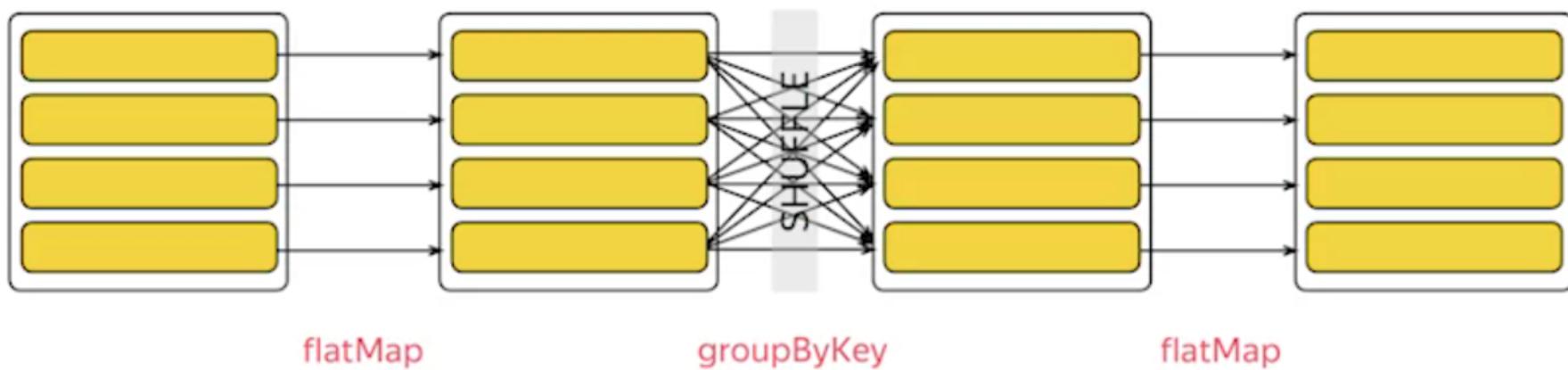
› Example: $Y = X.\text{flatMap}(m).\text{groupByKey}().\text{flatMap}(r)$

$X : RDD[T]$

$.flatMap(m) : RDD[(K, V)], \quad m: T \rightarrow Array[(K, V)]$

$.groupByKey() : RDD[(K, Array[V])]$

$.flatMap(r) : RDD[U], \quad r: (K, Array[V]) \rightarrow Array[U]$



Popularne akcje 1/3

- **collect()** – zbiera elementy i przekazuje do drivera.
Wyłącznie dla niewielkich zbiorów! Wszystkie dane ładowane są do pamięci programu sterującego.
- **take(n: int)** – zbiera wyłącznie **n** pierwszych elementów i przekazuje do programu sterującego.
- **top(n: int)** – zbiera wyłącznie **n największych** (relacja porządku) elementów i przekazuje do programu sterującego.
- **first()** – zwraca do drivera pierwszy element zbioru.

Popularne akcje 2/3

- **count()** – zwraca liczbę wszystkich elementów zbioru.
- **reduce(f: [T, T]→T)** – redukuje wszystkie elementy zbioru z wykorzystaniem agregującej dwuargumentowej funkcji f i zwraca wynik do drivera.
- **saveAsTextFile(path: str)** – każdy *executor* zapisuje swoje partycje w postaci tekstowej do pliku o zadanej ścieżce **path** i zwraca sterowanie do drivera.
- **saveAsHadoopFile(path: str, outputFormatClass: str)** – jw., ale z wykorzystaniem podanego formatu pliku **Hadoop**.

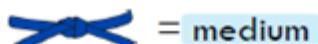
Popularne akcje 3/3

- **foreach(f: [T]→[J])** – każdy z executorów wywołuje f na każdym elemencie zbioru i zwraca sterowanie do drivera.
- **foreachPartition(f: Iterator[T]→[J])** – każdy z executorów wywołuje f na swoich partycjach i zwraca sterowanie do drivera.

Akcje i transformacje – taksonomia



= easy



= medium

Essential Core & Intermediate Spark Operations



General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

ACTIONS



- reduce
- collect
- aggregate
- fold
- first
- take
- foreach
- top
- treeAggregate
- treeReduce
- foreachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

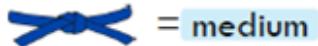
- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

Akcje i transformacje – taksonomia (K, V)



= easy



= medium

Essential Core & Intermediate PairRDD Operations

TRANSFORMATIONS

- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey

Math / Statistical

- sampleByKey

Set Theory / Relational

- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure

- partitionBy

ACTIONS

- keys
- values

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact



Odporność na błędy (fault-tolerance)

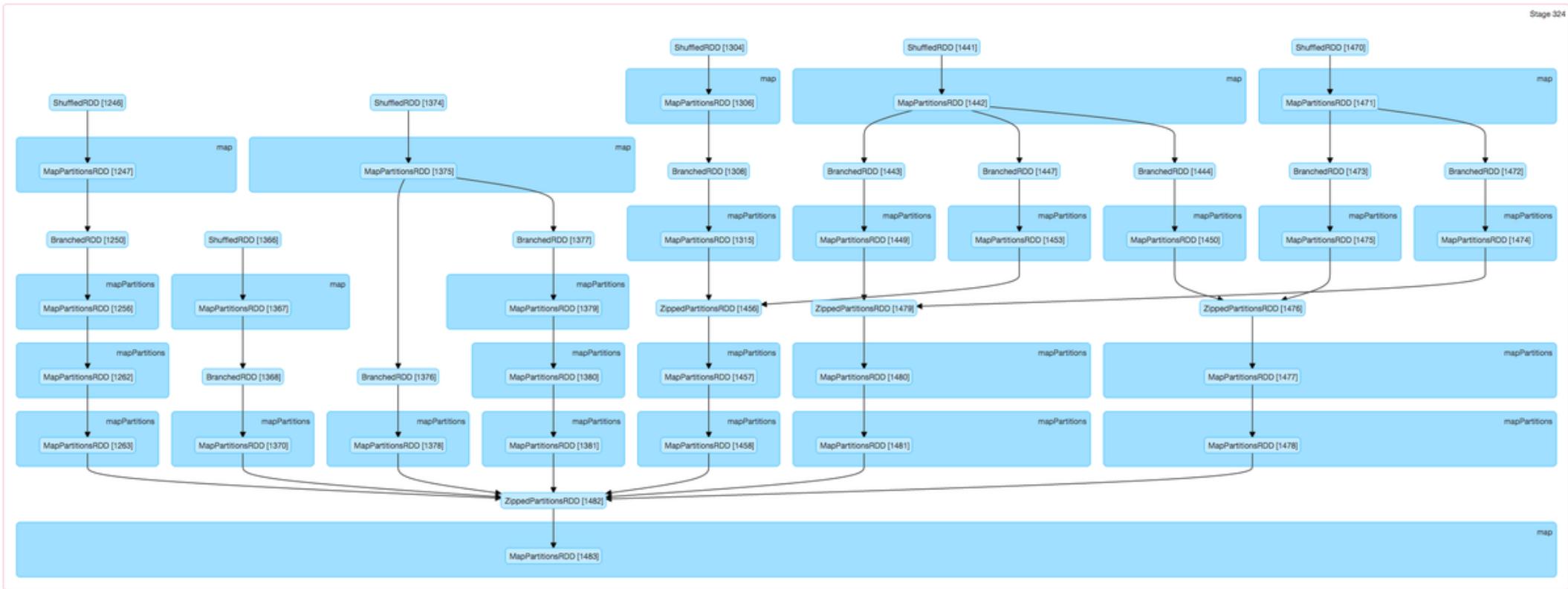
Dwa podstawowe aspekty:

- Niezawodne przechowywanie danych wejściowych i wyjściowych.
- Deterministyczne transformacje pozбавione efektów *ubocznych* (z uwzględnieniem domknięć transformacji).
- Deterministyczne – każde wywołanie funkcji zwraca takie same wyniki; przykład: brak wykorzystywania liczb losowych.
- Brak efektów *ubocznych* – wywołanie funkcji nie zmienia stanu zewnętrznego; przykład: brak zapisu do bazy danych, niestosowanie zmiennych globalnych.

Odporność na błędy a akcje

- Akcje *są efektem ubocznym* w Sparku.
- Akcje **muszą być idempotentne**, dzięki czemu można je bezpiecznie wywoływać wielokrotnie dla tych samych danych wejściowych.
- Przykład: **collect()**
RDD jest tylko do odczytu, więc wielokrotny odczyt nie zmienia jego stanu.
- Przykład: **saveAsTextFile(path: str)**
RDD jest tylko do odczytu, więc jego stan nie zmienia się z każdym zapisem.

Acykliczne grafy skierowane (DAG)



RDD / Dataframe / Dataset

| | RDD | Dataframe | Dataset |
|----------------------|--|--|---|
| Reprezentacja danych | Rozproszona kolekcja elementów bez żadnego schematu | Rozproszona kolekcja zorganizowana w nazwane kolumny | Rozszerzenie Dataframe o elementy obiektowe i spójności typów |
| Optymizacja | Brak, twórca kodu odpowiada za jego manualną optymalizację | Korzysta z catalyst optimizer | Również wykorzystuje catalyst optimizer |
| Planowanie schematu | Ręczna definicja schematu danych | Automatyczna detekcja schematu danych | Automatyczna detekcja schematu z wykorzystaniem silnika SQL |
| Operacje agregacji | Najwolniejsze w zestawieniu dla prostych operacji | Udostępnia proste API o największej wydajności w zestawieniu | Szybsze niż RDD, wolniejsze do Dataframe |

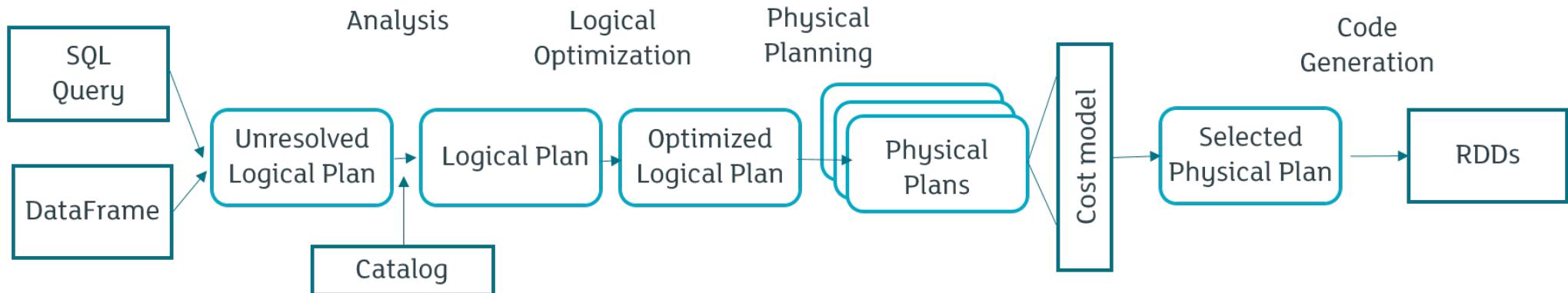
Dataframes

Źródła danych:

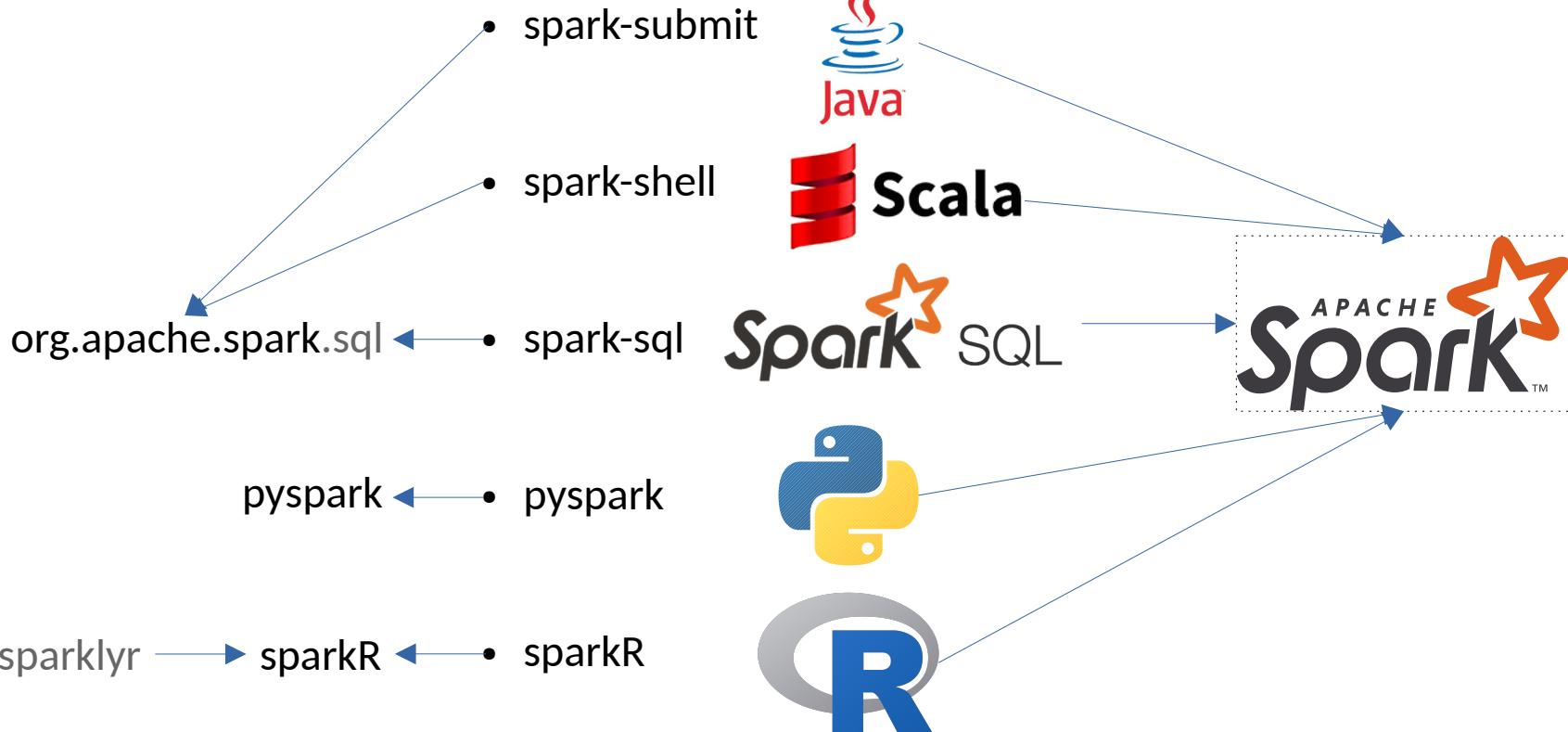
- RDD
- HIVE
- Parquet
- Cassandra
- RDBMS
- CSV
- XML
- JSON

Catalyst - optymalizacja generowanego bytecode'u Javy:

- drzewa (węzłów) Catalyst trees
- reguły: drzewo → drzewo
Scala pattern matching



Narzędzia – CLI

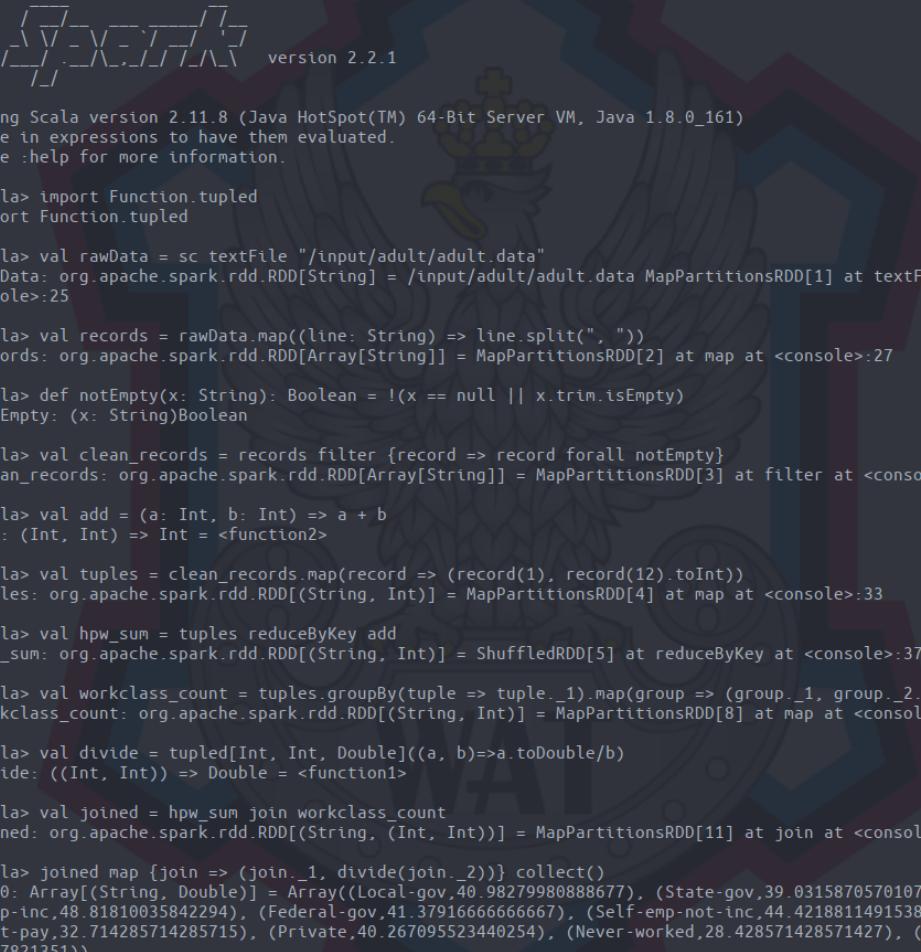


- $\text{RDD} \rightarrow \text{spark context } [\text{sc}]$
- $\text{Dataframe} \rightarrow \text{spark session } [\text{spark}]$

Narzędzia – CLI

| | |
|------------------------|---|
| --master | <ul style="list-style-type: none">• local – 1 worker thread• local[K] – K worker threads• local[*] – the number of logical cores worker threads• spark:// – Spark standalone• mesos://• yarn• k8s:// – Kubernetes |
| --deploy-mode | <ul style="list-style-type: none">• client (default) – locally• cluster – on the worker nodes |
| --num-executors | The total number of executors to use |
| --driver-cores | CPU cores to be used by: <ul style="list-style-type: none">• the driver process |
| --executor-cores | <ul style="list-style-type: none">• the executor process |
| --total-executor-cores | <ul style="list-style-type: none">• all executors |
| --driver-memory | Memory to be used by: <ul style="list-style-type: none">• the driver process |
| --executor-memory | <ul style="list-style-type: none">• The executor process |

testuser@master:~/lab-spark\$ spark-shell --master yarn --deploy-mode client --num-executors 3
 Spark context available as 'sc' (master = yarn, app id = application_1636934162434_2661).
 Spark session available as 'spark'.
 Welcome to



version 2.2.1

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_161)
 Type in expressions to have them evaluated.
 Type :help for more information.

```

scala> import Function.tupled
import Function.tupled

scala> val rawData = sc.textFile("/input/adult/adult.data")
rawData: org.apache.spark.rdd.RDD[String] = /input/adult/adult.data MapPartitionsRDD[1] at textFile at <console>:25

scala> val records = rawData.map((line: String) => line.split(", "))
records: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at <console>:27

scala> def notEmpty(x: String): Boolean = !(x == null || x.trim.isEmpty)
notEmpty: (x: String)Boolean

scala> val clean_records = records.filter {record => record forall notEmpty}
clean_records: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at filter at <console>:31

scala> val add = (a: Int, b: Int) => a + b
add: (Int, Int) => Int = <function2>

scala> val tuples = clean_records.map(record => (record(1), record(12).toInt))
tuples: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[4] at map at <console>:33

scala> val hpw_sum = tuples.reduceByKey add
hpw_sum: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey at <console>:37

scala> val workclass_count = tuples.groupBy(tuple => tuple._1).map(group => (group._1, group._2.size))
workclass_count: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[8] at map at <console>:35

scala> val divide = tupled[Int, Int, Double]((a, b)=>a.toDouble/b)
divide: ((Int, Int)) => Double = <function1>

scala> val joined = hpw_sum.join(workclass_count)
joined: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[11] at join at <console>:41

scala> joined.map {join => (join._1, divide(join._2))}.collect()
res0: Array[(String, Double)] = Array((Local-gov,40.9827998088677), (State-gov,39.03158705701079), (Self-emp-inc,48.81810035842294), (Federal-gov,41.37916666666667), (Self-emp-not-inc,44.421881149153876), (Without-pay,32.714285714285715), (Private,40.267095523440254), (Never-worked,28.428571428571427), (?,.31.91938997821351))

scala>
```

File Edit View Run Kernel Hub Tabs Settings Help [Toree] spark-shell

In [1]: 1 import Function.tupled

In [2]: 1 val rawData = sc.textFile("/input/adult/adult.data")
 2 rawData.count()

Out[2]: 32562

In [3]: 1 val records = rawData.map((line: String) => line.split(", "))
 2 records.take(2)

Out[3]: Array(Array(39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K), Array(50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K))

In [4]: 1 def notEmpty(x: String): Boolean = !(x == null || x.trim.isEmpty)
 2 val clean_records = records filter {record => record forall notEmpty}

In [5]: 1 val add = (a: Int, b: Int) => a + b
 2 val tuples = clean_records.map(record => (record(1), record(12).toInt))
 3 val hpw_sum = tuples reduceByKey add
 4 hpw_sum.first()

Out[5]: (Local-gov,85777)

In [6]: 1 val workclass_count = tuples.groupBy(tuple => tuple._1).map(group => (group._1, group._2.size))
 // Action: val workclass_count = tuples.countByKey()
 3 workclass_count sortBy(c => c._2, ascending=false) take 3

Out[6]: Array((Private,22696), (Self-emp-not-inc,2541), (Local-gov,2093))

Mean hours per week by work class

In [7]: 1 val divide = tupled[Int, Int, Double]((a, b)=>a.toDouble/b)
 2 val joined = hpw_sum join workclass_count
 3 joined.map {join => (join._1, divide(join._2))}.collect()

Out[7]: Array((Local-gov,40.9827998088677), (State-gov,39.03158705701079), (Self-emp-inc,48.81810035842294), (Federal-gov,41.37916666666667), (Self-emp-not-inc,44.421881149153876), (Without-pay,32.714285714285715), (Private,40.267095523440254), (Never-worked,28.428571428571427), (?,.31.91938997821351))

In [8]: 1 sc.stop