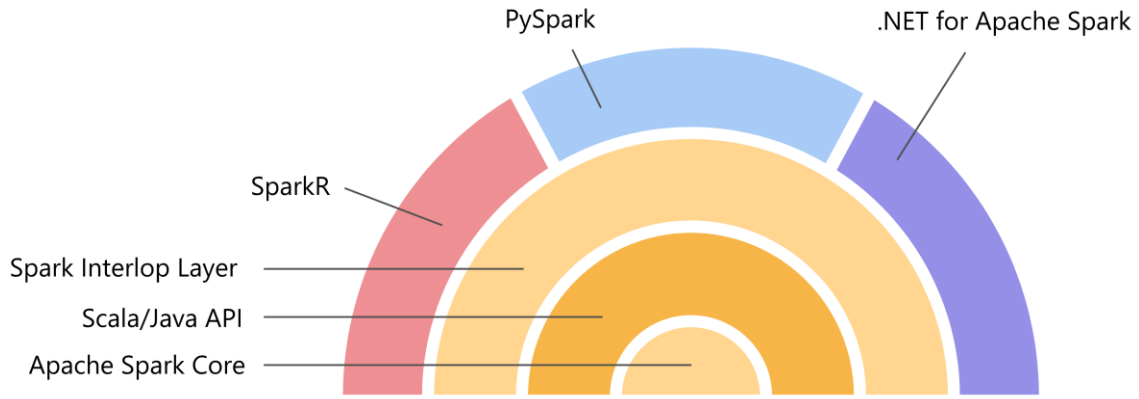
Apache Spark és kapcsolódó keretrendszerek áttekintése

Emődi Márk

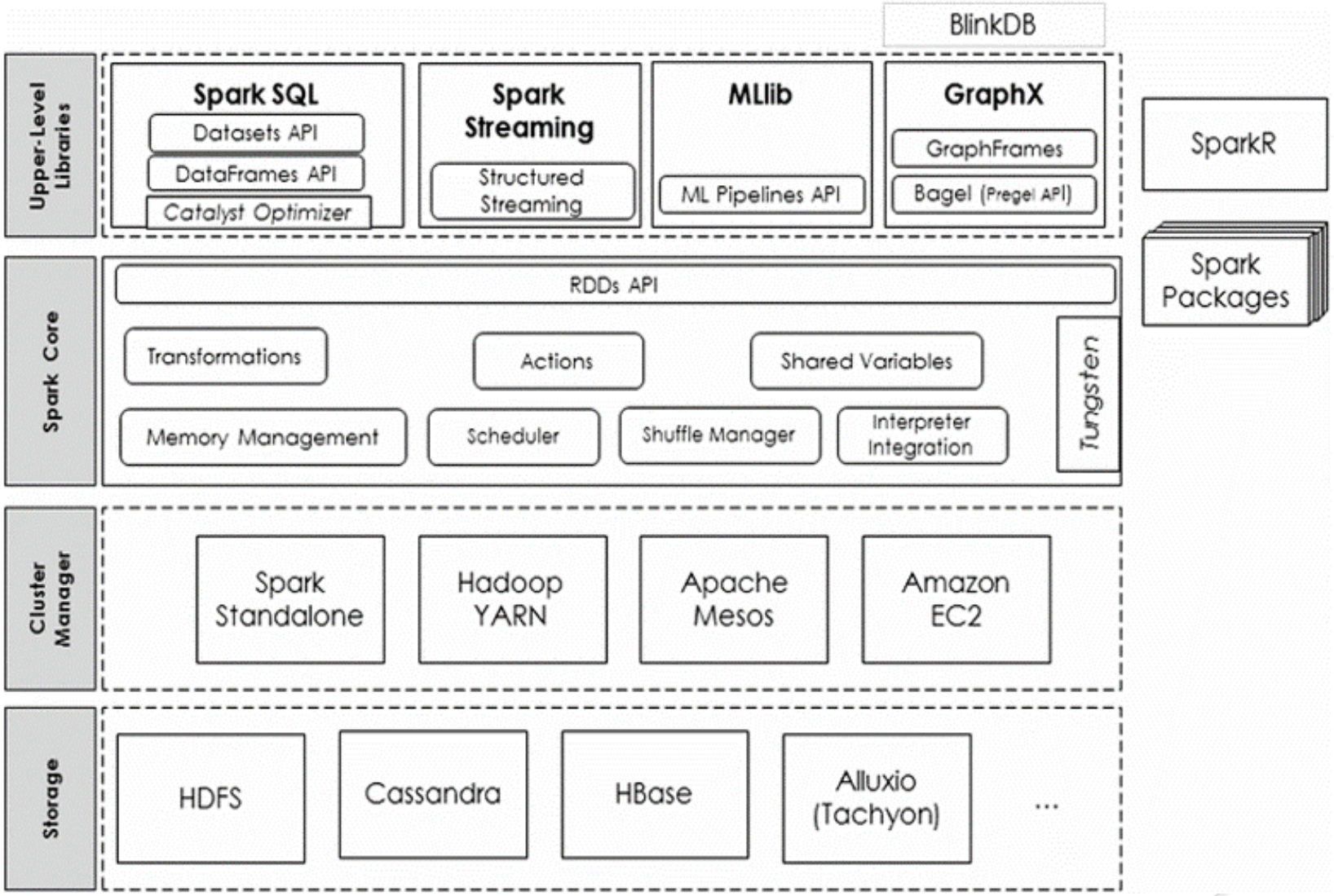
emodi.mark@sztaki.hu

Apache Spark



- Berkeley egyetem fejlesztése (2009)
- Scale programozási nyelven íródott
- Alábbi nyelveken nyújt natív interfészt
  - Scala, Java, R és Python
  - Nem hivatalos interfészek (.net, Julia, ...)
- OS független megvalósítás (Linux/Unix/Windows)
- Adatok memóriába történő beolvasása
  - Gyorsabb feldolgozás diszken végzett műveletekhez képest
- Elosztott működés (Master-Worker állomcsomópontok)

Apache Spark

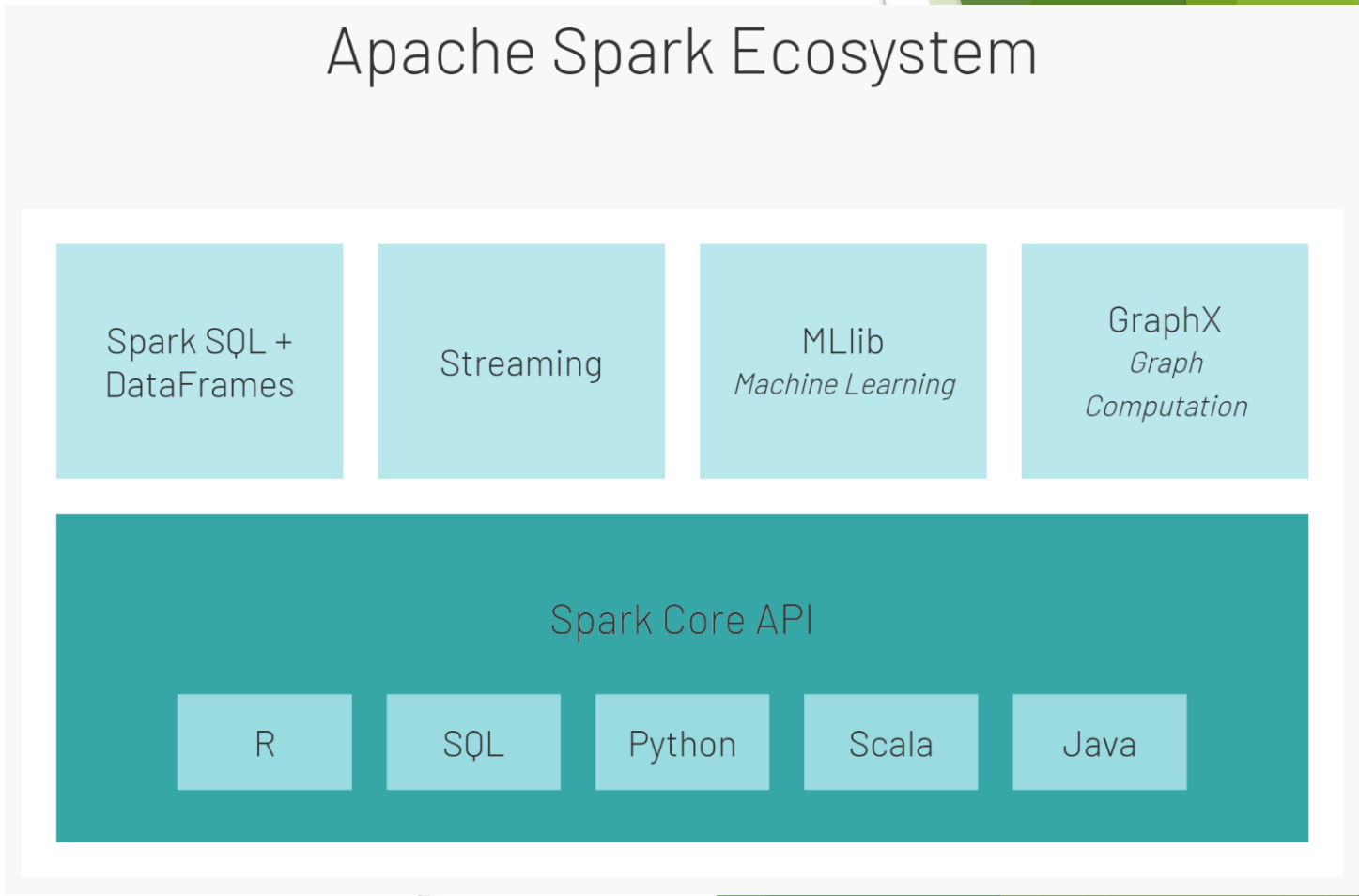


ELKH Cloud

APACHE Spark™

Apache Spark - Áttekintés

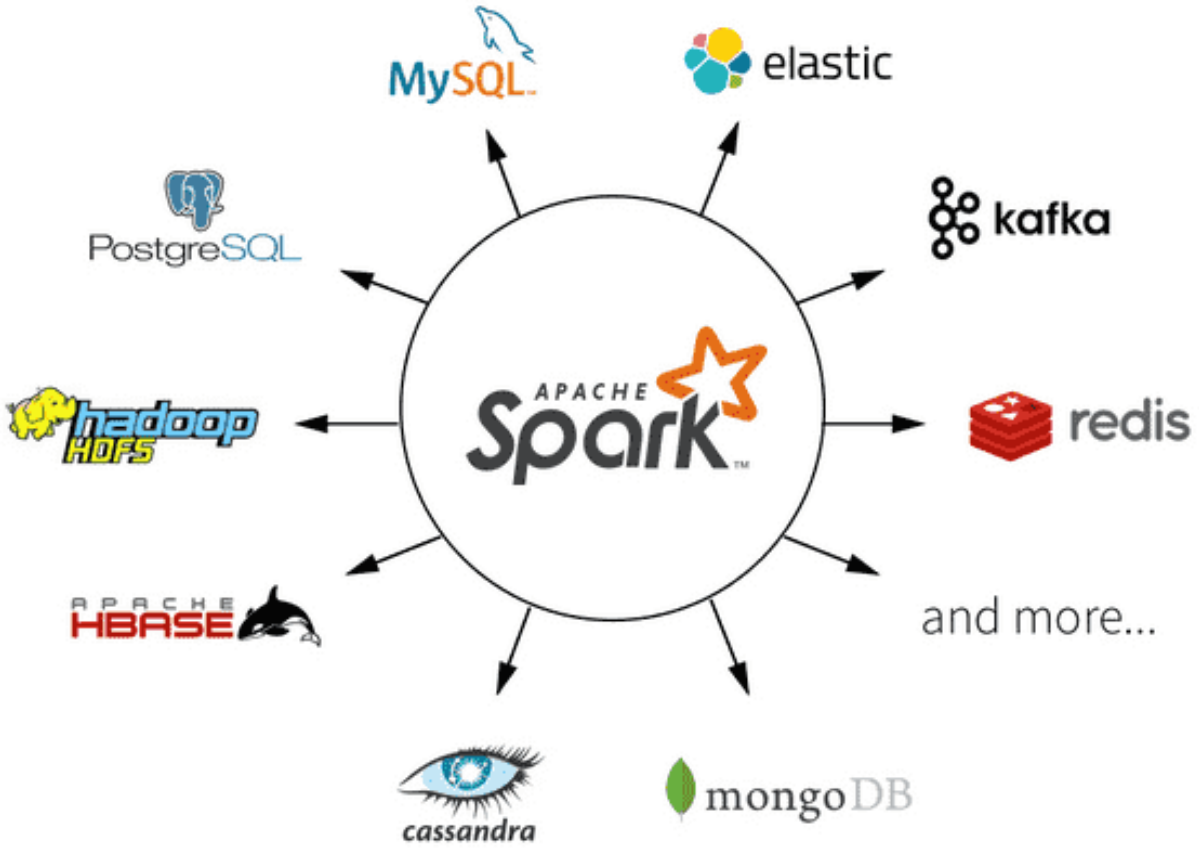
- Spark Core: A Spark Core a Spark platform általános végrehajtó motorja, amelyre az összes többi funkció épül
  - Nagyteljesítményű, általános végrehajtási modellt nyújt az alkalmazások számára. Java, Scala és Python interfészeket támogat
- Core API-ban leggyakrabban megjelenő fogalom az „RDD”: Resilient Distributed Dataset



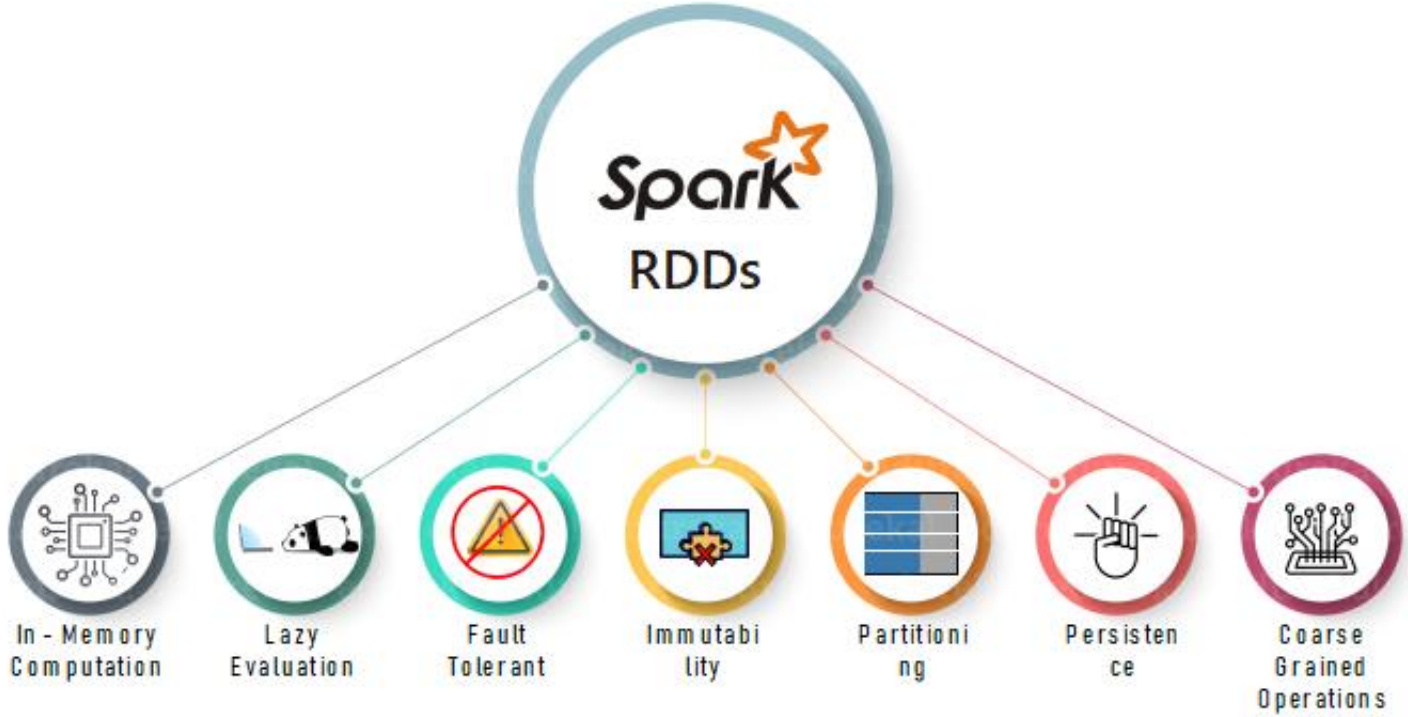
Apache Spark - Core

- Memória kezelés
- Hibatűrés
- Feladat ütemezés és szétosztás, valamint monitorozás
- Interakció a tároló egységekkel
- Spark alapból nem rendelkezik tároló réteggel
  - Adatfeldolgozás többféle adattarolóból történhet
    - HDFS
    - NoSQL

      Hbase, Cassandra, mongoDB, …
    - RDBMS
- RDD alapvető építőeleme



Apache Spark - Core



- Az RDD az alapfogalom a Sparkon belül az adatra, amely absztrakciót is magába foglal:
  - RDD csak olvasható (Immutability)
  - Partíciónált sokasága a rekordoknak (Partitioning)
  - Műveletek párhuzamosan futnak le
  - Lusta kiértékelés (Lazy Evaluation)
  - Hibatűrő (Fault Tolerant)
  - Adatok memóriában tartása (Persistance)
  - Műveletek iterálása az egész adathalmazon (Coarse Grained Operations)

Apache Spark - Core

- RDD kétféleképpen hozható létre:
  - Külső adatforrásból adatbetöltéssel (pl. HDFS-ről)

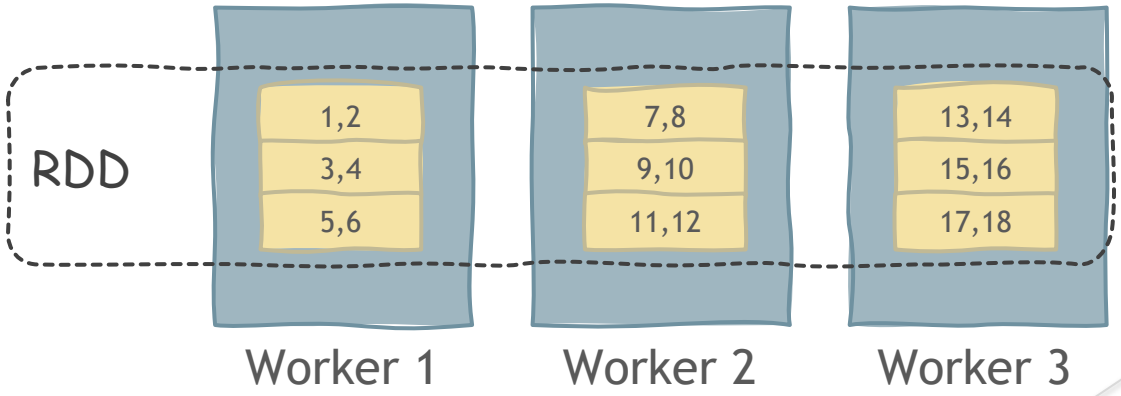
```
>>> distFile = sc.textFile("data.txt")
```

  - Vagy a driver programban (felhasználó által írt program) létrehozott objektumokból (pl. egy létrehozott lista)

```
>>> rdd = sc.parallelize((1, 2, 3, 4))
```

Apache Spark - Core

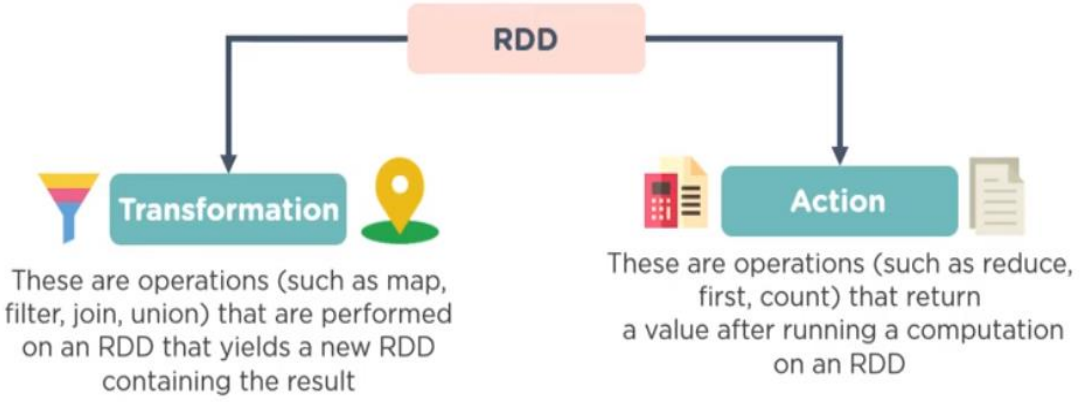
- RDD-n végezhető műveletek lehetnek:
  - Transzformációk (transformation), pl. map(), filter()
  - Lezáró műveletek (action), pl. count(), first(), take()
- RDD létrehozásakor megadhatjuk, hogy a memóriában/lemezen kerüljön tárolásra az adat, illetve milyen módszerrel vagy irányelvvel történjen a tárolás



Apache Spark - Core – RDD (Resilient Distributed Datasets)

- Rugalmas elosztott adatszerkezetek
- Párhuzamosan feldolgozhatóak
- A transzformációk RDD-kből hoznak létre RDD-ket logikai szinten.
  - Akkor valósul meg, ha az első műveletet (actiont) meghívjuk

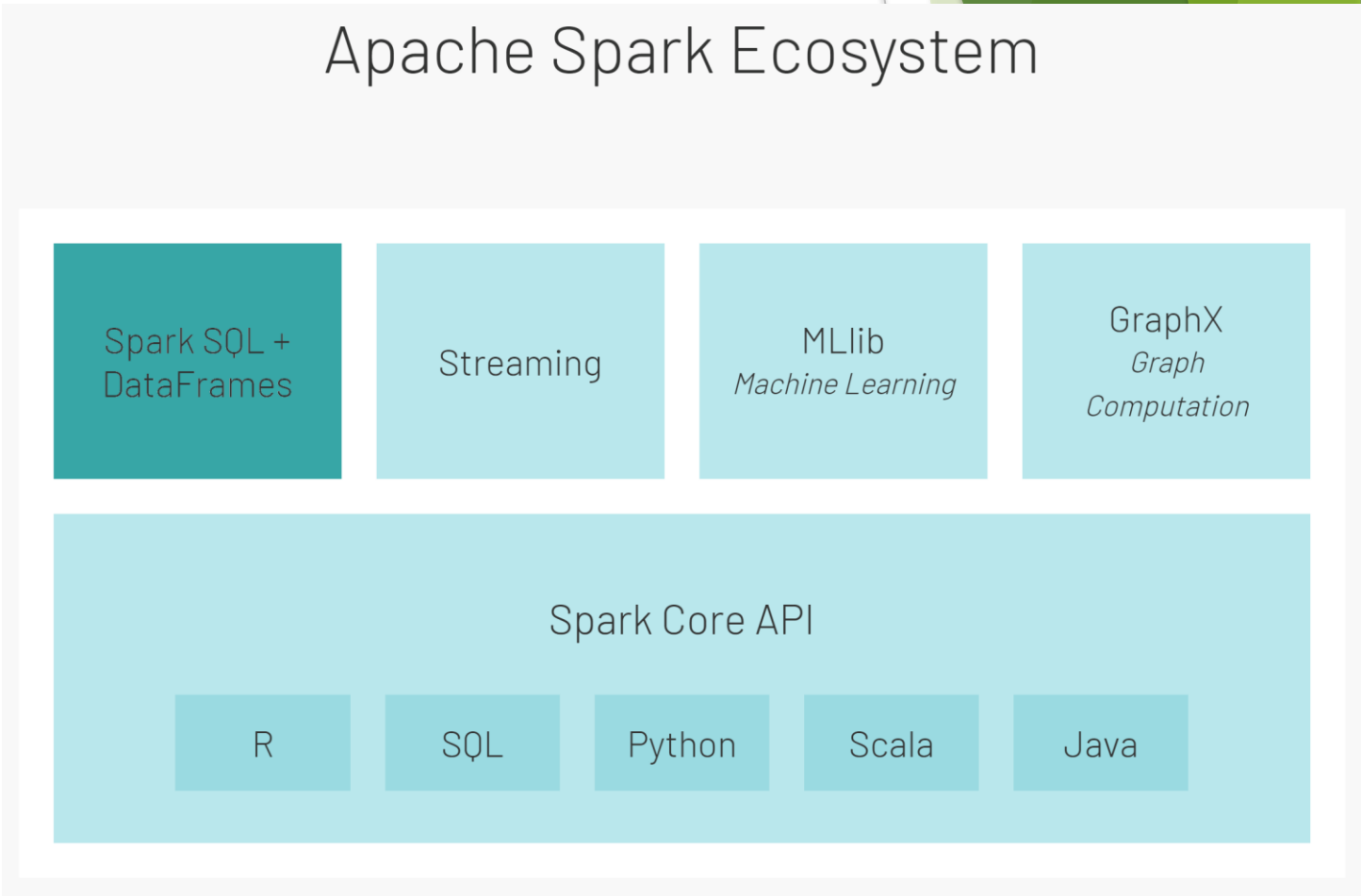
```
val x: sc.textfile => RDD (spark context, main entry point)
val y = x.map(….) => RDD
val z = y.filter(….) => RDD
z.count() => végrehajtódnak a transzformációk sorban
(a kiértékelés csak akkor történik meg, amikor szükség van rá)
```



Apache Spark – Spark SQL

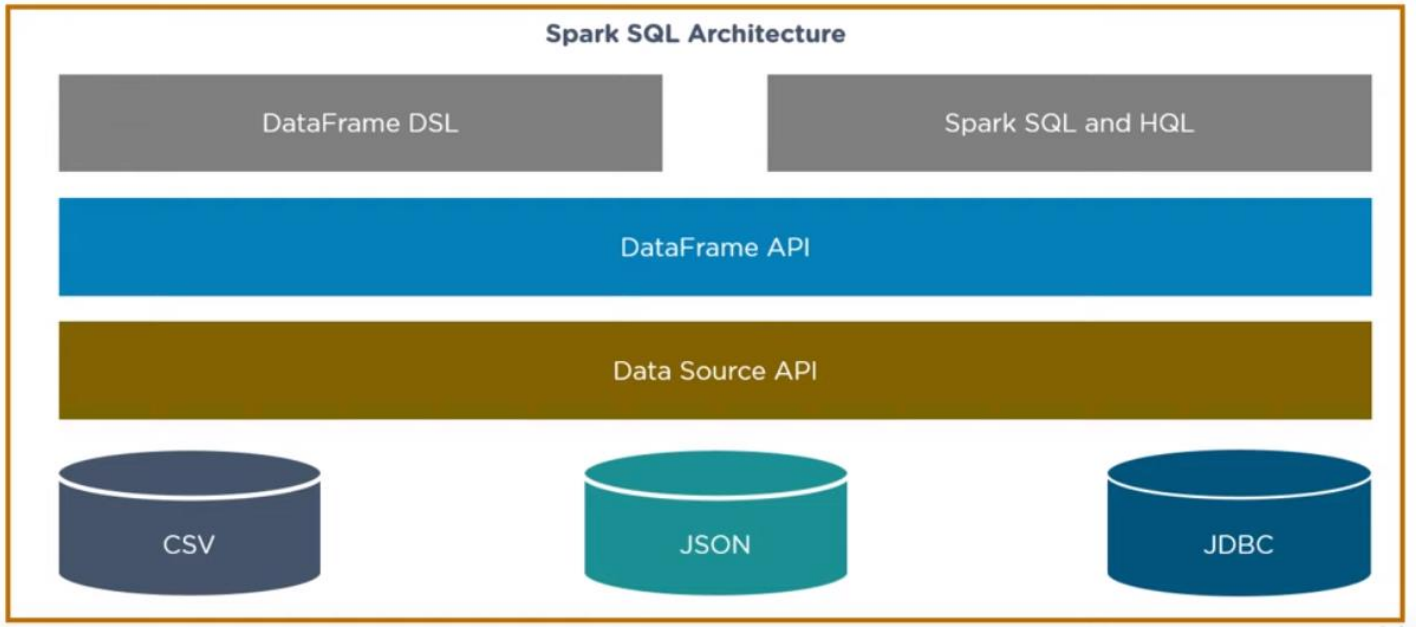
ELKH Cloud

- Spark SQL: SQL szerűen kérdezhetünk le strukturált adatokat, mint a dataframe-k és dataset-ek
- Az alapvető Spark RDD API-val ellentétben a Spark SQL által biztosított interfészek több információt nyújtanak a Sparknak mind az adatok, mind az elvégzett számítások felépítéséről. A Spark SQL használja ezeket az extra információkat további optimalizálások végrehajtására.
- Részben strukturált adat
  - nem feltétlen típusos, de valamilyen szinten strukturált, pl. csv vagy hasonló szerkezet



Apache Spark - Spark SQL

- Sokféle bemenettel képes dolgozni
  - CSV, JSON, JDBC, avro, parquet, stb.
- A Data Source API egységes interfészt nyújt
- A DataFrame API úgy használható, mint egy relációs adatbázistábla, sorokat és oszlopokat tartalmaz.
  - A bejövő adatokból létre tudunk hozni DataFrameket és ezen lekérdezéseket tudunk végrehajtani
  - RDD-ből DataFramet tudunk konvertálni és a konverziót visszafele is végre tudjuk hajtani
- Lekérdezések támogatott nyelvei:
  - DataFrame DSL – Domain Specific Language
  - Spark SQL
  - HQL (Hive Querry Language)



Apache Spark - Spark SQL

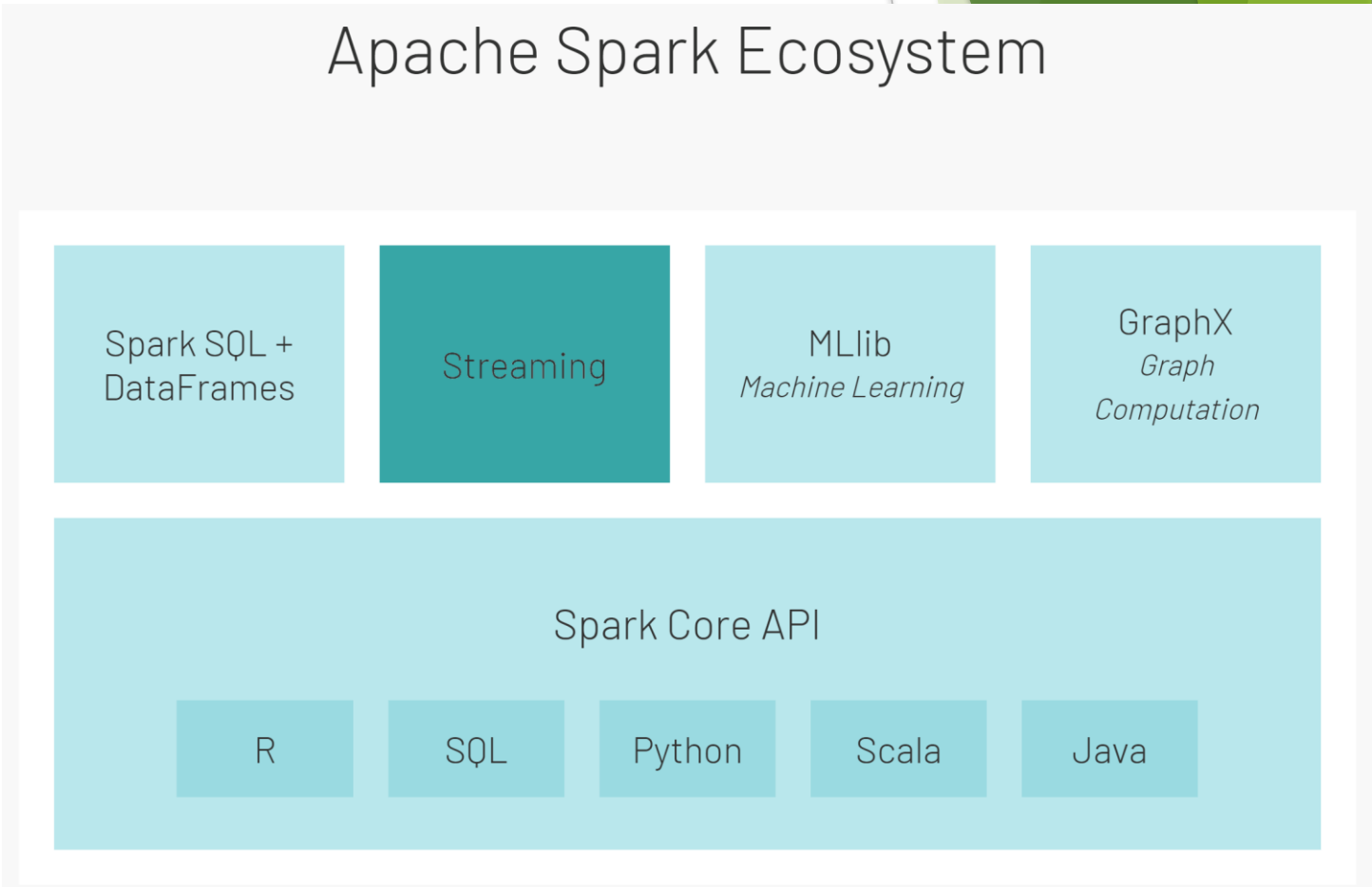
```
l = [('Alice', 1), ('Bill', 3), ('Bob', 5), ('Jay', 21), ('Jill', 11),('Mark', 51)]
rdd = sc.parallelize(l)
print(spark.createDataFrame(rdd).collect())
df = spark.createDataFrame(rdd, ['name', 'age'])
df.show()
```

```
[Row(_1='Alice', _2=1),
 Row(_1='Bill', _2=3),
 Row(_1='Bob', _2=5),
 Row(_1='Jay', _2=21),
 Row(_1='Jill', _2=11),
 Row(_1='Mark', _2=51)]
```

```
+-----+---+
| name|age|
+-----+---+
|Alice|  1|
| Bill|  3|
|  Bob|  5|
|  Jay| 21|
| Jill| 11|
| Mark| 51|
+-----+---+
```

Apache Spark - Spark Streaming

- Spark Streaming: Az alkalmazások nagyrészénél elvárás lehet, hogy nem csak a kötegelt adatfeldolgozásra legyen képes, hanem a valós idejű, új adatfolyamok feldolgozására és elemzésére is.
- A Spark tetején futó Spark Streaming erősen interaktív és elemző alkalmazásokat tesz lehetővé mind a streaming, mind a történelmi adatok feldolgozása során, miközben a Spark egyszerű használata és hibatűrési tulajdonságai jellemzik. Könnyen integrálható különböző adatforrással (pl. Kafka, Flume, HDFS, Twitter)
- Az adat beérkezése során végezhetünk rajta műveleteket/transzformációkat. Az adatok elemzése kisebb szeletekben történik
- Alapeleme a DStream, ami RDD sorozatoknak feleltethető meg



Apache Spark
Spark Streaming

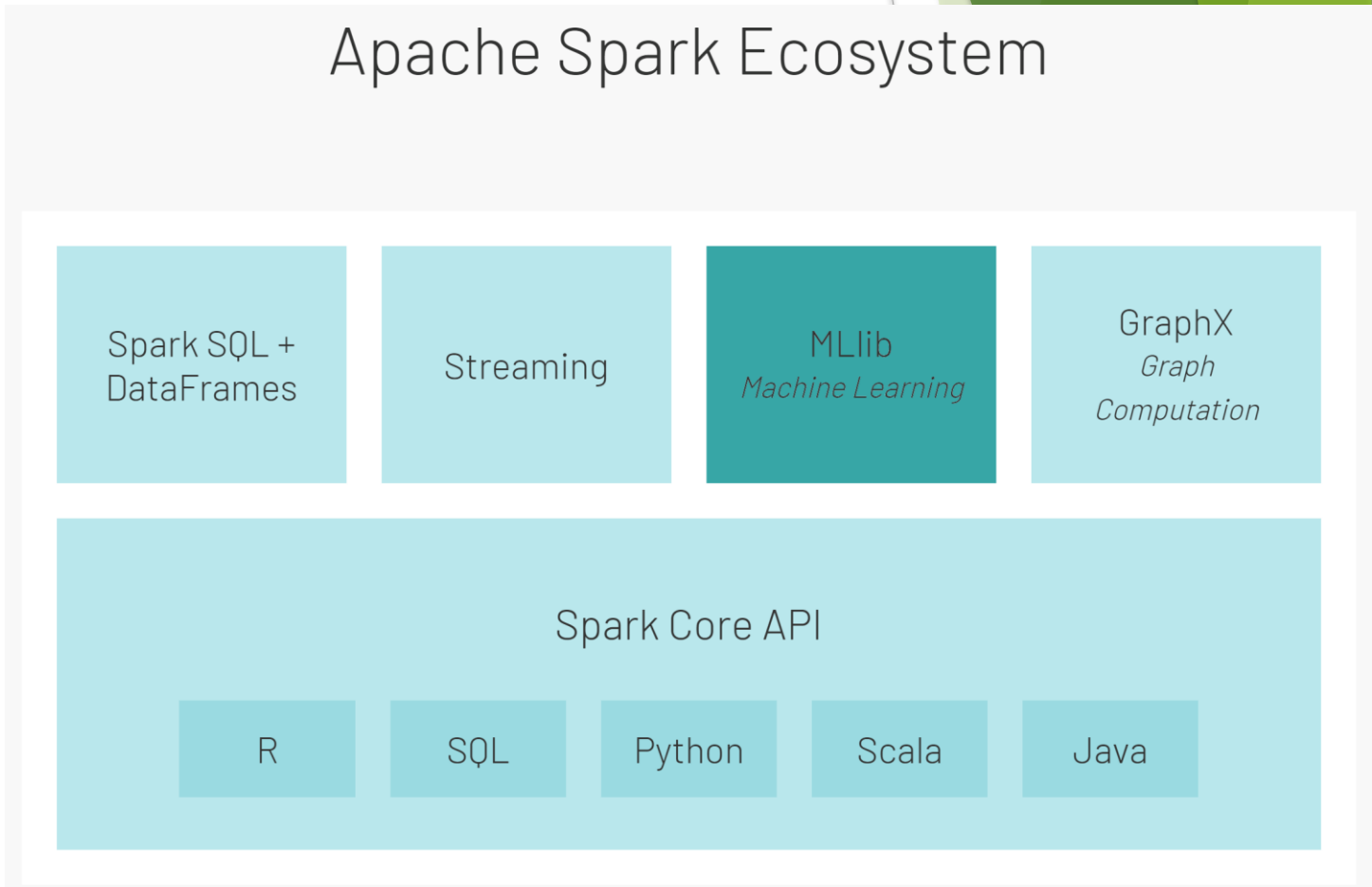
- Lehetővé teszi stream adatok real-time feldolgozását (pl. webszerver logok real-time kiértékelését)
- Külön API-t biztosít, amely hasonló a Spark Core által használt API-hoz
- **Input data stream**: állandóan növekvő fájl, event-ek (pl. kattintások alapján) vagy termékek alapján, amiket választanak a boltban
- **Spark Streaming application**: kisebb streamekre (szeletekre) bontja
  - pl. minden 5. másodpercben megnézi, hogy van-e új adat és azokat időablakokba rendezi
- **Spark Engine**: feldolgozza ezeket a szeleteket majd menti és/vagy megjeleníti

Apache Spark
Spark Streaming

- Alkalmazási példa:
  - Webshopban a rendszer a klikkeket figyeli, valamint a vevő eddigi vásárlásait… => ajánlások készítése a vevő számára
  - Képzeljünk el egy boltot, ahol kamerák figyelik ki melyik sorban/részlegben áll és nézelődik. Egy algoritmus figyeli valós időben, hogy mely termékeket nézik. Figyelembe veszi mennyi vevő van éppen, mennyi termék van és ezek alapján határoz meg egy olyan árat, amiért már megveszik az adott terméket.

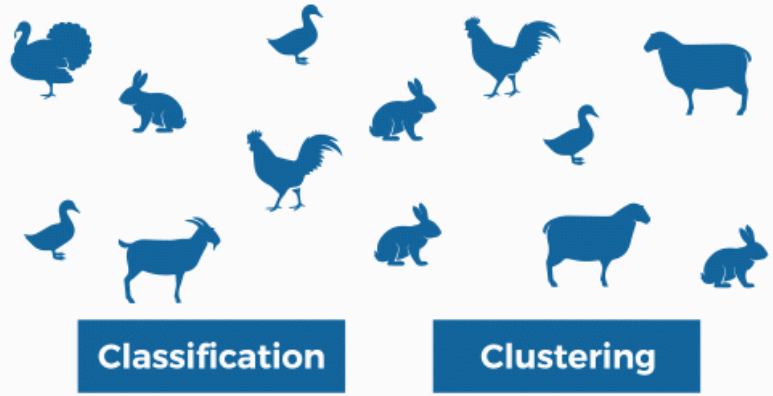
Apache Spark - MLlib

- MLlib: egy függvénygyűjtemény gépi tanuló algoritmusok fejlesztéséhez
  - prediktív analízishez
  - termék ajánló rendszerek
- A Spark saját machine learning könyvtára, amiben számos algoritmus beépítve megtalálható
- Egyszerűsített alkalmazhatóság elosztott környezetben



Apache Spark - MLlib

- Osztályozás: Ez egy felügyelt gépi tanulás alá sorolható módszer, amely a bemenetet több előre meghatározott osztály egyikébe sorolja.
- Klaszterezés: Klaszterezés során egy algoritmus az objektumokat kategóriákba csoportosítja az bemeneti példák közötti hasonlóságok elemzésével

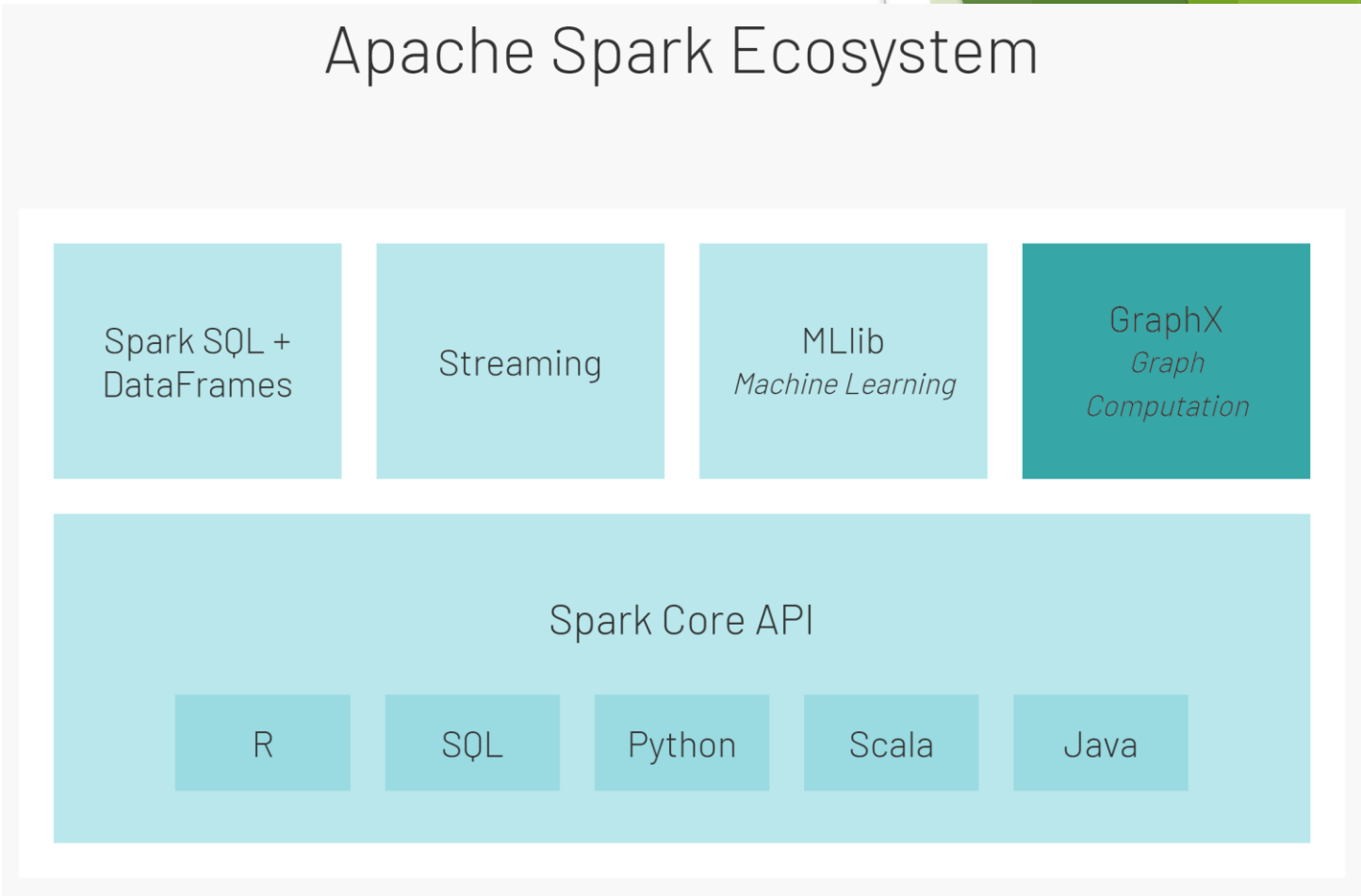


Apache Spark - MLlib

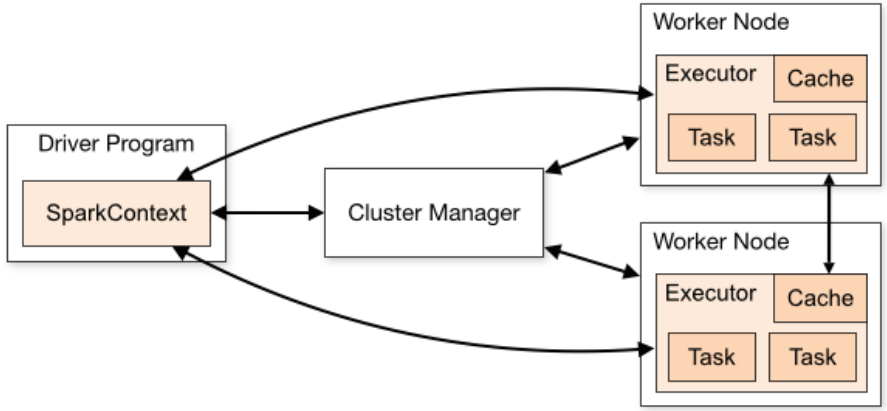
- Együttműködésen alapuló szűrés: Az együttműködésen alapuló szűrési algoritmus módszer arra, hogy automatikus előrejelzéseket (predikciókat) [szűrést] készítsen a felhasználó érdekeiről azáltal, hogy sok felhasználótól összegyűjti a preferenciákkal kapcsolatos információkat [együttműködő].
- Az együttműködésen alapuló szűrési megközelítés alapfeltevése az, hogy ha A személy ugyanazon a véleményen van egy kérdésben, mint B személy , akkor A-nak nagyobb valószínűséggel van B véleménye egy másik kérdésről, mint egy véletlenszerűen kiválasztott személyről.

Apache Spark - GraphX

- GraphX:
  - Gráf számítási motor
  - Gráfokba rendezett adatok kezeléséhez találták ki
  - Tartalmazza a gyakran használt gráf algoritmusokat



Klaszter menedzser



- A klaszter menedzserek lehetővé teszik a Spark skálázását
- Klaszter menedzserek
  - Standalone

    A Sparkhoz mellékelt egyszerű klaszterkezelő, amely megkönnyíti a klaszter beállítását
  - Hadoop YARN

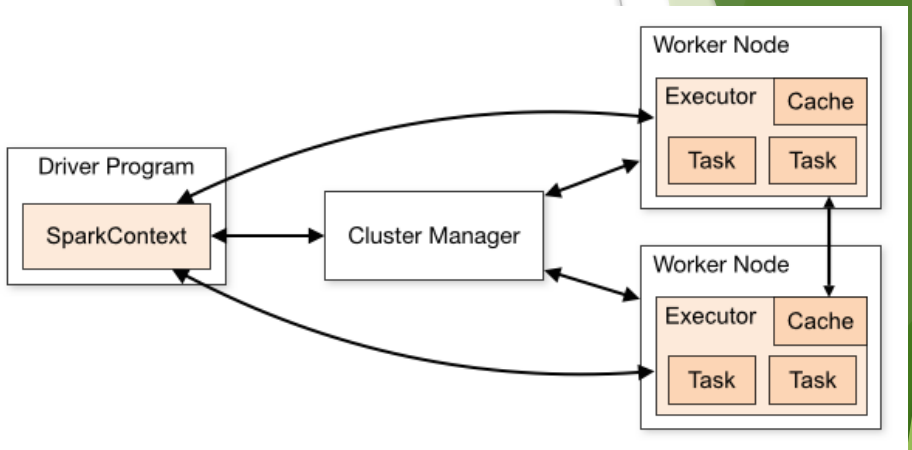
    A Hadoop 2 erőforrás-kezelő motorja
  - Mesos

    Általános klaszterkezelő, amely a Hadoop MapReduce, valamint a különböző szolgáltatás rendszereket képes futtatni
  - Kubernetes

    Konténeres alkalmazások telepítésének, méretezésének és kezelésének automatizálására használható

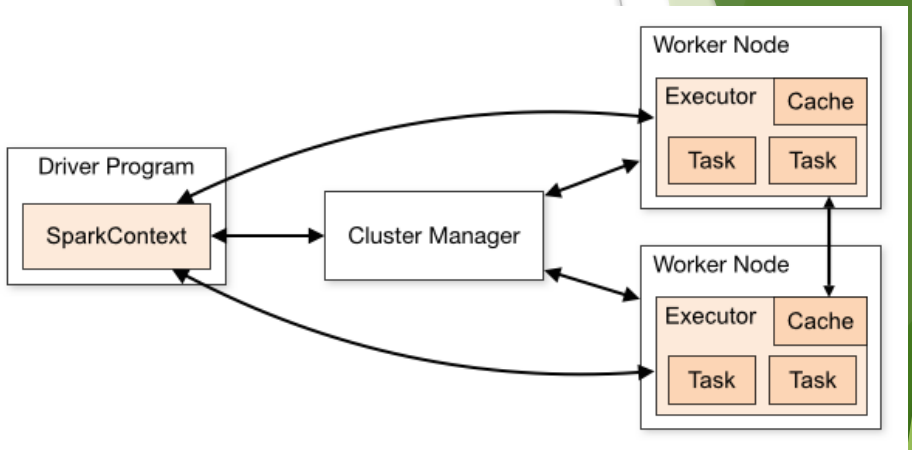
Spark elosztott működés

- Minden Spark programban először definiálni kell a futtatás kontextusát, a SparkContext-et, ez a program belépési pontja
- SparkContext-ben megadásra kerül az alkalmazás neve, illetve a futtatási környezet (lehet lokális, vagy futhat a klaszteren is), de konfigurálható részletesen is:
  - Használni kívánt CPU magok számát
  - Allokálni kívánt memóriák nagyságát
  - Tárolási irányelveket
- Amennyiben elosztott környezetben futtatjuk, úgy a műveletvégzés párhuzamosításra kerül a dolgozó állomások között



Spark elosztott működés

- SparkContext létrehozása után lehet RDD-t létrehozni, azokon műveleteket elvégezni egészen a SparkContext leállításáig
- A menedzser állomáson indítjuk a Spark programot
- A SparkContext a belépési pont
  - A SparkContext interakcióba lép a klaszter menedzserrel (pl. Standalone klaszter menedzser) és a menedzser szétosztja a különböző végrehajtó egységeknek a feladatokat
- A klaszter menedzser az erőforrás igényekkel a dolgozó állomás ágenséhez fordul, és ha rendelkezésre áll elegendő erőforrás, lefoglalja azt az alkalmazás számára

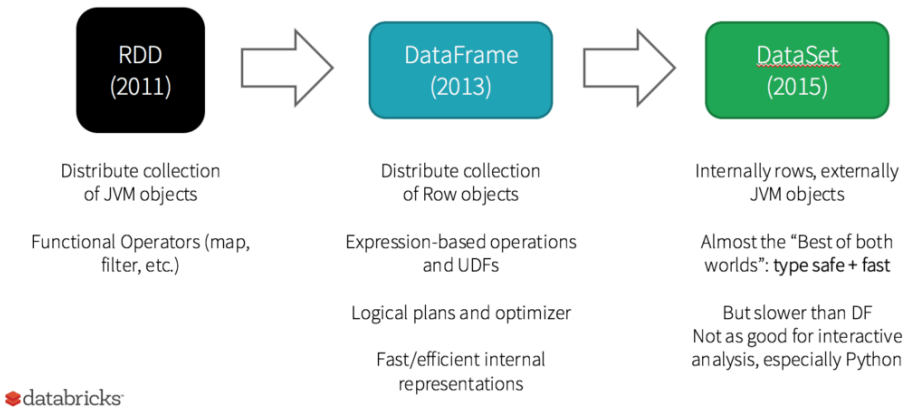


Apache Hadoop és Apache Spark

ELKH Cloud

hadoop	Apache Spark
Processing data using MapReduce in Hadoop is slow	Spark processes data 100 times faster than MapReduce as it is done in-memory
Performs batch processing of data	Performs both batch processing and real-time processing of data
Hadoop has more lines of code. Since it is written in Java, it takes more time to execute	Spark has fewer lines of code as it is implemented in Scala
Hadoop supports Kerberos authentication, which is difficult to manage	Spark supports authentication via a shared secret. It can also run on YARN leveraging the capability of Kerberos

DataFrame és DataSet



- DataFrame: Az RDD-k sorokba és elnevezett oszlopokba vannak rendezve
  - SQL lekérdezések támogatása
  - Egyéb programozási nyelvekben használt megoldások átjárhatósága
    - Pl. Pandas R/Python nyelven
- DataSet: DataFrame típusos oszlopokkal (erősen típusos nyelvekben hasznos
  - Pl. Java, Scala

Apache Spark adatstukturák

- **DataFrame, Dataset és RDD közötti átjárás egyszerű konverzióval megoldható. A DataFrame és Dataset RDD-re épül.**
- A strukturált és félig strukturált adatokhoz a DataFrames és a DataSet struktúra optimalizálási és teljesítménybeli előnyöket nyújt
- **Mikor érdemes RDD-t használni:**

1. Alacsony szintű transzformációk és műveletek esetén
2. Strukturálatlan adatok esetén (médiafolyamok, szövegfolyamok esetén)
3. Adatokat funkcionális programozási konstrukciókkal történő feldolgozása esetén (tartományspecifikus kifejezések)
4. Nem fontos számunkra egy séma (pl. oszlopos formátum, ha közben hozzáférünk az attribútumokhoz név vagy oszlop szerint is)

DataFrame konverziók

RDD → DF

```
l = [('Alice', 1), ('Bill', 3), ('Bob', 5), ('Jay', 21), ('Jill', 11),('Mark', 51)]
rdd = sc.parallelize(l)
spark.createDataFrame(rdd).collect()
df = spark.createDataFrame(rdd, ['name', 'age'])
df.show()
```

DF → RDD

```
# ROW RDD
converted_rdd = df.rdd
Vagy
# RDD
rdd = df.rdd.map(tuple)
vagy
rdd = df.rdd.map(list)
```

Összefoglaló

Fast processing

Spark contains Resilient Distributed Datasets (RDD) which saves time taken in reading, and writing operations and hence, it runs almost ten to hundred times faster than Hadoop

In-memory computing

In Spark, data is stored in the RAM, so it can access the data quickly and accelerate the speed of analytics

Flexible

Spark supports multiple languages and allows the developers to write applications in Java, Scala, R, or Python

Fault tolerance

Spark contains Resilient Distributed Datasets (RDD) that are designed to handle the failure of any worker node in the cluster. Thus, it ensures that the loss of data reduces to zero

Better analytics

Spark has a rich set of SQL queries, machine learning algorithms, complex analytics, etc. With all these functionalities, analytics can be performed better

Összefoglaló

- A Spark nagy számítási kapacitással bíró keretrendszer
- A gyors sebességét a saját végrehajtó motorjának köszönheti, amely törekszik az adatokat a memóriában tárolni, emellett párhuzamos végrehajtást valósít meg
- Gyakori alkalmazási területei a Machine Learning és real-time környezetekben végzett számítások
- Python, Java, Scala nyelveken programozható, illetve a SparkSQL támogatás révén hagyományos SQL lekérdezéseket is készíthetünk

Köszönöm a figyelmet!