Distributed graph and stream processing

Gombos Gergő

Mire nem jó a Hadoop?

- Stream
  - HDFS-ről dolgozik
- Iteratív algoritmusok
  - Gráf feldolgozás

Stream

- Mi az a stream?
- Miért kell?
- Miért elosztva?
- Storm

Stream

- Stream
  - Folyamatos adatfolyam
- Kihívások
  - Nagy méretű adatok generálódnak
  - Nincs idő letárolni
  - Azonnali elemzési eredmények szükségesek

Stream példák

- Közösségi hálózatok
  - Twitter real-time search
- Website statistics
  - Google analytics
- Támadás detektálás
  - A legtöbb adatközpontban

MapReduce?

- Batch processing
  - Sokat kell várni mire lefut nagy adatra
- Nincs megoldás hosszú stream-ekre

Storm

- Apache projekt
- Több programozási nyelv támogatása
  - Python
  - Ruby
  - Java
- Több nagy cég használja
  - Twitter: keresés, személyreszabás
  - Wheater channel

Storm használók

Storm component

- Tuples
- Streams
- Spouts
- Bolts
- Topologies

Tuple

- Elemek rendezett listája
- Elemek nevesítve vannak
- Példa:
  - <user, tweet>
    - <„user1”, „hello Storm”>
    - <„user2”, „hello Korszerű”>
  - <url, sourceIP, date, time>
    - <„inf.elte.hu”, 192.1.68.5, ”2015-04-28”, ”9:00”>
    - <„google.com”, 168.92.65.8, ”2015-02-18”, ”19:00”>

Stream

- Tuple-k sorozata
  - Nincs korlátozva
- Példa
  - <„user1”, „hello Storm”>, <„user2”, „hello Korszerű”>
  - <„inf.elte.hu”, 192.1.68.5, ”2015-04-28”, ”9:00”>, <„google.com”, 168.92.65.8, ”2015-02-18”, ”19:00”>

Spout

- Külső forrás a stream-be
- Több is lehet egyszerre
- Külső forrás lehet:
  - Crawler
  - Adatbázis
  - RabbitMQ
  - ...

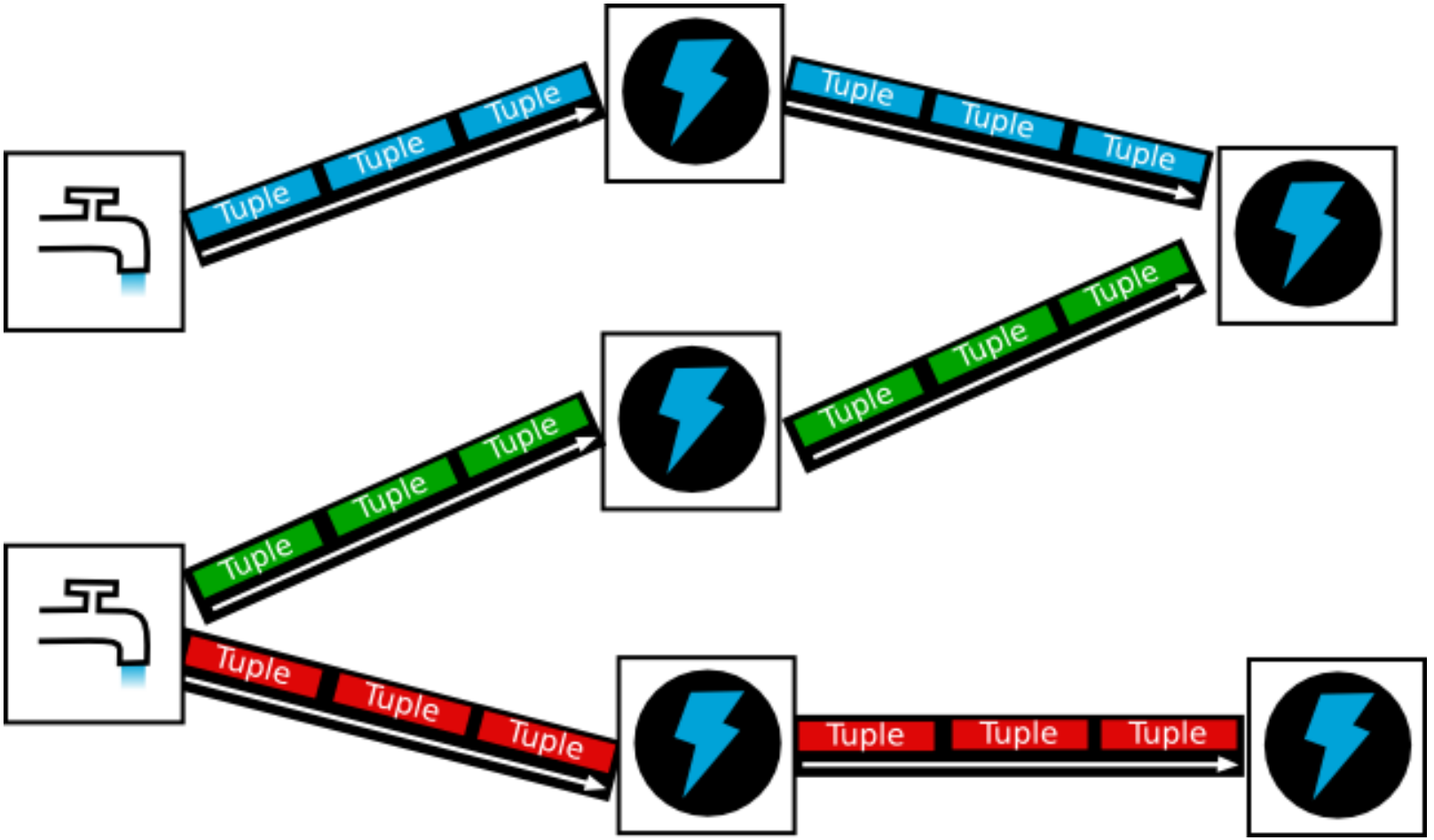
Bolt

- Feldolgozó egység
- Bemente lehet:
  - Stream(ek)
  - Másik Bolt(ok)
- Kimenete stream más Boltok számára

Topology

- Irányított gráf
  - Tartalmazhat köröket
  - Határozatlan ideig is futhatnak
- Részei:
  - Bolt(ok)
  - Spout(ok)
- Ezt nevezzük Storm alkalmazásnak

Topology példa



Bolt lehetőségek

- Szűrés
  - Csak a megfelelő tuple-őket továbbítja
- Join
  - Két streamből egyet csinál, feltételnek megfelelően
- Transform
  - Mégváltozatja a tuple értékét

Párhuzamos Bolt

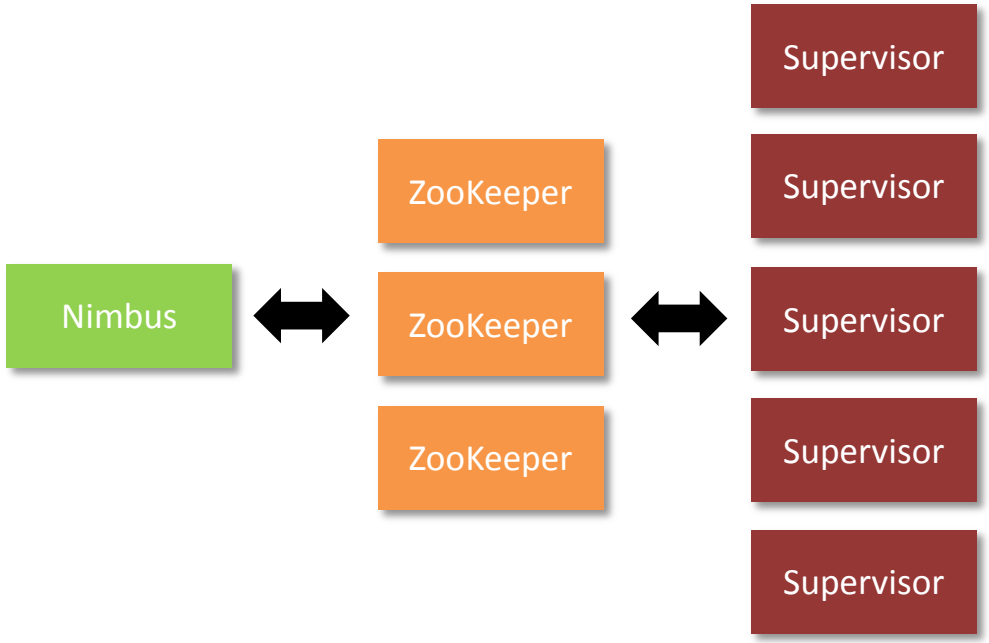
- A feldolgozásnak gyorsnak kell lennie
- Több „task”-ot tartalmaz egy Bolt
- A stream-et szétosztja a taskok között
  - Minden tuple egy task-hoz kerül
    - Ezt a „Grouping startegy” határozza meg
- 3 féle csoportosítás van.

Csoportosítások

- Shuffle grouping
  - Stream szétosztódik a taskok között
  - Round-Robin stílusú
- Fields grouping
  - A stream valamely elem szerinti szétosztás
  - Pl: tweet_user [A-M], [N-Z]
- All grouping
  - Minden task megkapja az összes tuple-t
  - Join műveleteknél

Storm Cluster

- Master node
  - Neve: Nimbus
  - Elosztja a kódot a clusterben
  - Taskokat gépekhez rendeli
  - Figyeli a hibákat
- Worker node
  - Neve: Supervisor
  - Futtatja a taskokat
- ZooKeeper
  - Koordinálja a kommunikációt a Nimbus és a Supervisor között
  - A Nimbus és Supervisor állapota is elérhető benne



Üzenet garancia

- Minden tuple feldolgozása végig követett a topológiába
- Ha hiba van újra emitálni kell a tuple-t a spout-ba
- Emit(tuple, output)
  - Az output-ra írja a tuple-t
- Ack(tuple)
  - Visszaigazoljuk hogy feldolgoztuk a tuple-t
- Fail(tuple)
  - Hibát jelzünk,

Hibakezelés

- Fontos!!!
- A tuple-ök memóriában tárolódnak.
- Ha hiba van akkor memory leak-et okozhatnak.

Cluster Hibák

- Nimbus hiba
  - Topology fut tovább
  - ZooKeeper újraindítja
- Supervisor hiba
  - ZooKeeper újraindítja
- Task hiba
  - Supervisor indítja újra

Storm UI

Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.4	1d 8h 40m 39s	1	1	0	1	6	6

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
demo	demo-1-1430054005	INACTIVE	1d 7h 28m 25s	1	6	6

Supervisor summary

Id	Host	Uptime	Slots	Used slots
0265fa4c-89b8-478d-b16b-61f361ba1871	dbpc62.inf.elte.hu	1d 8h 40m 18s	1	1

Nimbus Configuration

Key	Value
dev.zookeeper.path	/tmp/dev-storm-zookeeper
drpc.childopts	-Xmx768m
drpc.invocations.port	3773

Spark Streaming

- A bemenő stream-et X másodpercenként átadja egy batch folyamatnak
- Spark Streaming = kis méretű batch jobok sorozata
- Batch vágás < 0,5 másodperc
- Feldolgozás ~ 1 másodperc

Stream Összegzés

- Stream feldolgozás fontos napjainkban
- Storm (Spark Streaming)
- Párhuzamosítás
- Alkalmazás topológia
- Hibakezelés

Graph Processing

Mi a gráf?

- Nem konkrét adathalmaz
- Csomópontokat tartalmaz
  - Facebook: felhasználó
- Éleket tartalmaz
  - Kapcsolat csomópontok között
  - Facebook: barát kapcsolat

Gráfok

- Nagy méretű gráfok mindenhol
  - Internet gráf: routerek, switchek, vezetékek
  - Web: weboldalak, hivatkozások
  - Közösségi hálózatok
    - Facebook: user, friend
    - Twitter: user, follower
  - Biológiai gráfok
    - Ökoszisztem gráf

Gráf feldolgozási feladatok

- Alap statisztikák
  - Pl: Bemenő/kimenő élek
- Származtatott tulajdonságok a gráfból
  - Clusterezési együttható
- Legrövidebb út keresés
  - Internet (routing)
- Párosítás
  - Randi oldalak (match.com)
- …

Kihívások

- Gráfok nagyok
  - Milliós nagyságrendű csúcsok és élek
- Tárolás, feldolgozás egy serveren:
  - Memória limit
  - Lassú
- Megoldás: Elosztva

Tipikus gráf feldolgozás

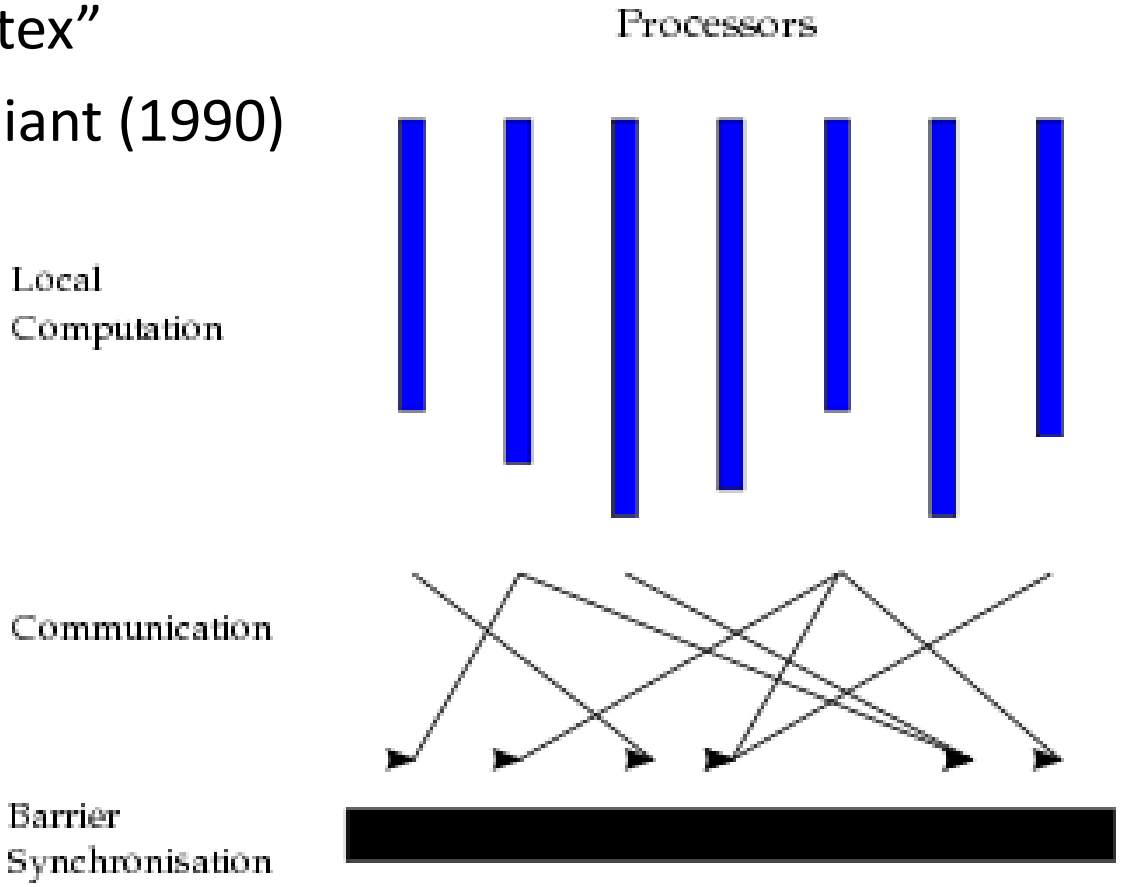
- Iteratív
- Minden csúcs rendelkezik értékkel
- Minden iterációban:
  - Megkapja a szomszéd csúcsok értékeit
  - Újraszámolja a saját értékét
  - Elküldi az új értékeket a szomszédoknak
- Iteráció vége:
  - Fix számú iteráció
  - Nincs küldött érték a szomszédoknak

MapReduce?

- 1 job = 1 iteráció
- A reduce kulcsa nodeid, értéke az üzenetek
- Előny:
  - Ismert
- Hátrány:
  - Minden csúcs átmegy a hálózaton minden iterációban
  - Minden csúcs ki lesz írva a HDFS-re
  - Lassú, nem hatékony

Bulk Synchronous Parallel(BSP) Model

- „Think like a vertex”
- Originally by Valiant (1990)



Elosztott gráf feldolgozás

- „Think like a vertex”
- Minden csúcs legyen egy szerveren
- Így a szerverek csúcshalmazt tárolnak
- Egy iteráció lépése:
  - Gather: megkapjuk a szomszéd csúcsok értékét
  - Apply: Kiszámoljuk az új értéket
  - Scatter: elküldjük az új értéket a szomszédoknak

Csúcs tárolás

- Honnan tudjuk melyik csúcs melyik szerveren tárolódik?
- Lehetőségek:
  - Hash-alapú:
    - Hash(csúcs id) mod num(server)
    - Hasonló: P2P
  - Lokalitás-alapú
    - A szomszédos csúcsokat próbáljuk egy szerveren tárolni
    - Csökken a szerver-szerver közötti kommunikáció

Pregel

- Google fejleszti
- Open-source verzió: Giraph
- Master
  - Workerek listája
  - Workerek monitorozása
  - Web UI a job futásokhoz
- Worker
  - Csúcs feldolgozás
  - Kommunikáció a többi workerrel
- Adatok elosztva vannak tárolva
- Átmeneti adatok a lokális tárolón

Pregel futása

1. A programkódok lemásolódnak a clusterban
2. Master felosztja a csúcsokat a Workereknek
   - Minden Worker betölti majd Active-ra jelenti
3. Master elindítatja a Workerekkel az iterációkat.
   1. Minden Worker lefuttatja az Active csúcsokon
   2. Üzenetek el lesznek küldve ha szükségesek
   3. Ha befejeződött az üzenetváltás, akkor Master indít új iterációt
4. Számolás megáll, ha nincs üzenetváltás vagy nincs Active csúcs
5. Master utasítja a Workereket hogy mentse a részgráfot

Hibakezelés

- Checkpoint
  - Periódikusan a Master kezdeményezésére a Workerek mentik a Rész gráfokat.
- Hiba detektálás
  - Periódikus „ping” üzenetek Master -> Worker
- Visszaállítás
  - Master újraosztja a részgráfokat
  - Worker visszaállítja az utolsó checkpoint alapján

Milyen gyors?

- Legrövidebb út keresés 1 csúcsból minden csúcshoz
- 1 milliárd csúcsú gráfon
  - 50 Worker: 180 másodperc
  - 800 Worker: 20 másodperc
- 50 milliárd csúcsra 800 Workerrel 700 mp

Gráf feldolgozás összegzés

- Rengeteg gráf alapú adatunk van
- Szükséges ezeknek az elemzése
- MapReduce nem jó választás
- Elosztott gráf feldolgozás: Pregel
- Sok Pregel alapú rendszer:
  - Giraph, Piccolo, Spark Graph

Spark

Spark

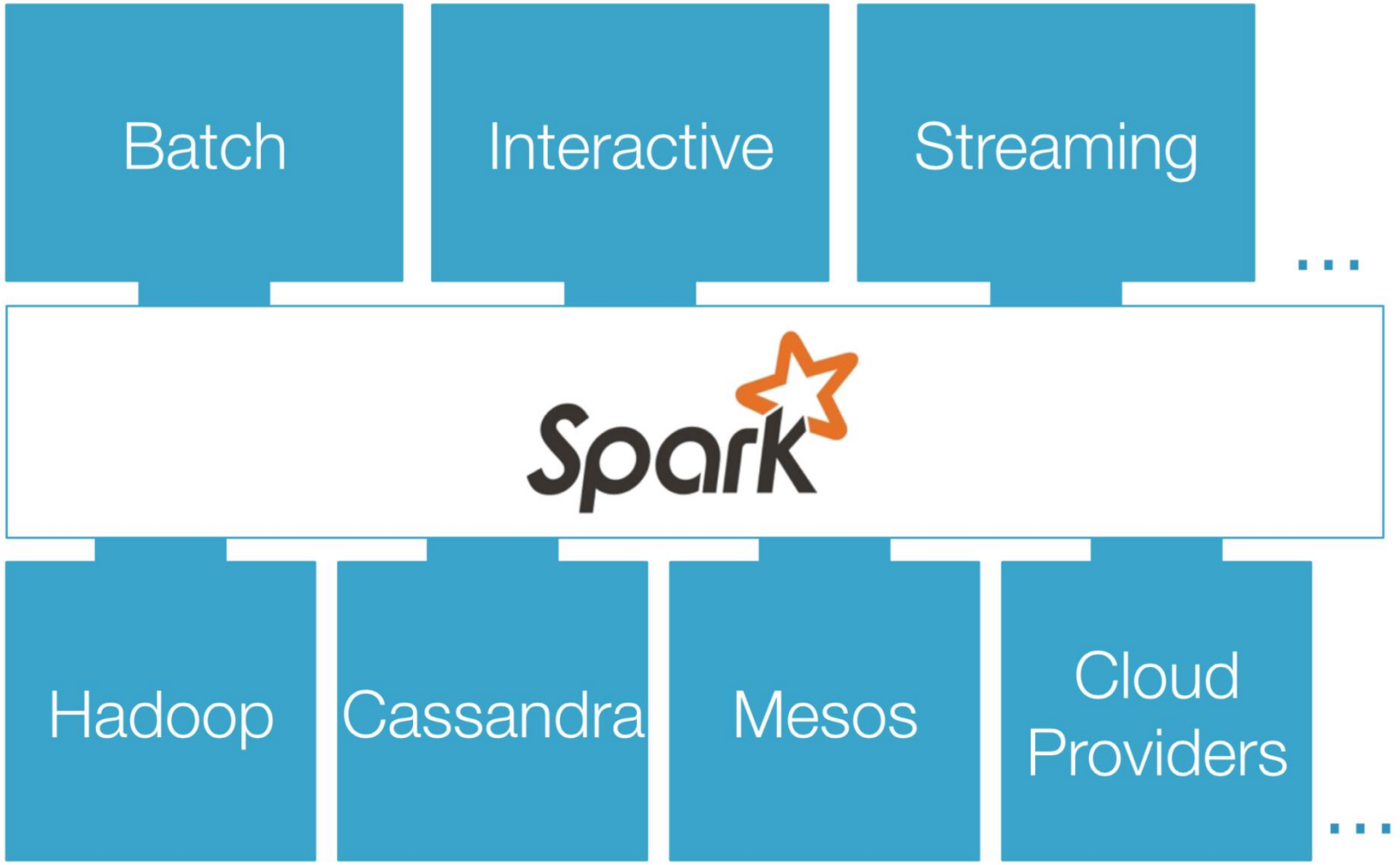
- 2010 cikk Berkely's AMPLab
  - Resilient Distributed Datasets (RDDs)

- Általánosított elosztott számítási platform
- Memória alapú számítási rendszer.
- Hibakezelés memória caching-re
- Bővíthető interface

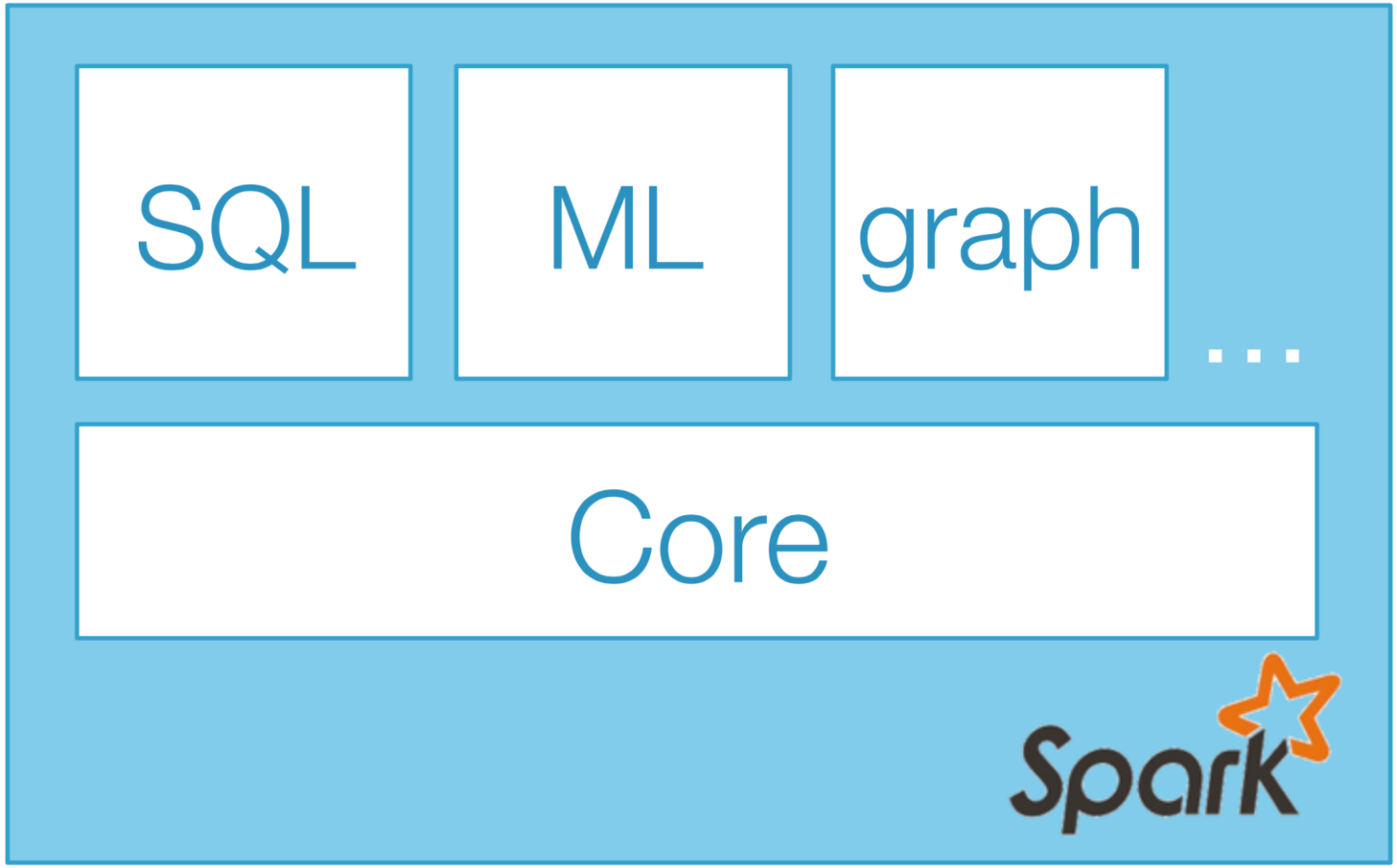
Daytona Gray Sort 100TB Benchmark

	Data Size	Time	Nodes	Cores
Hadoop MR (2013)	102.5 TB	72 min	2,100	50,400 physical
Apache Spark (2014)	100 TB	23 min	206	6,592 virtualized

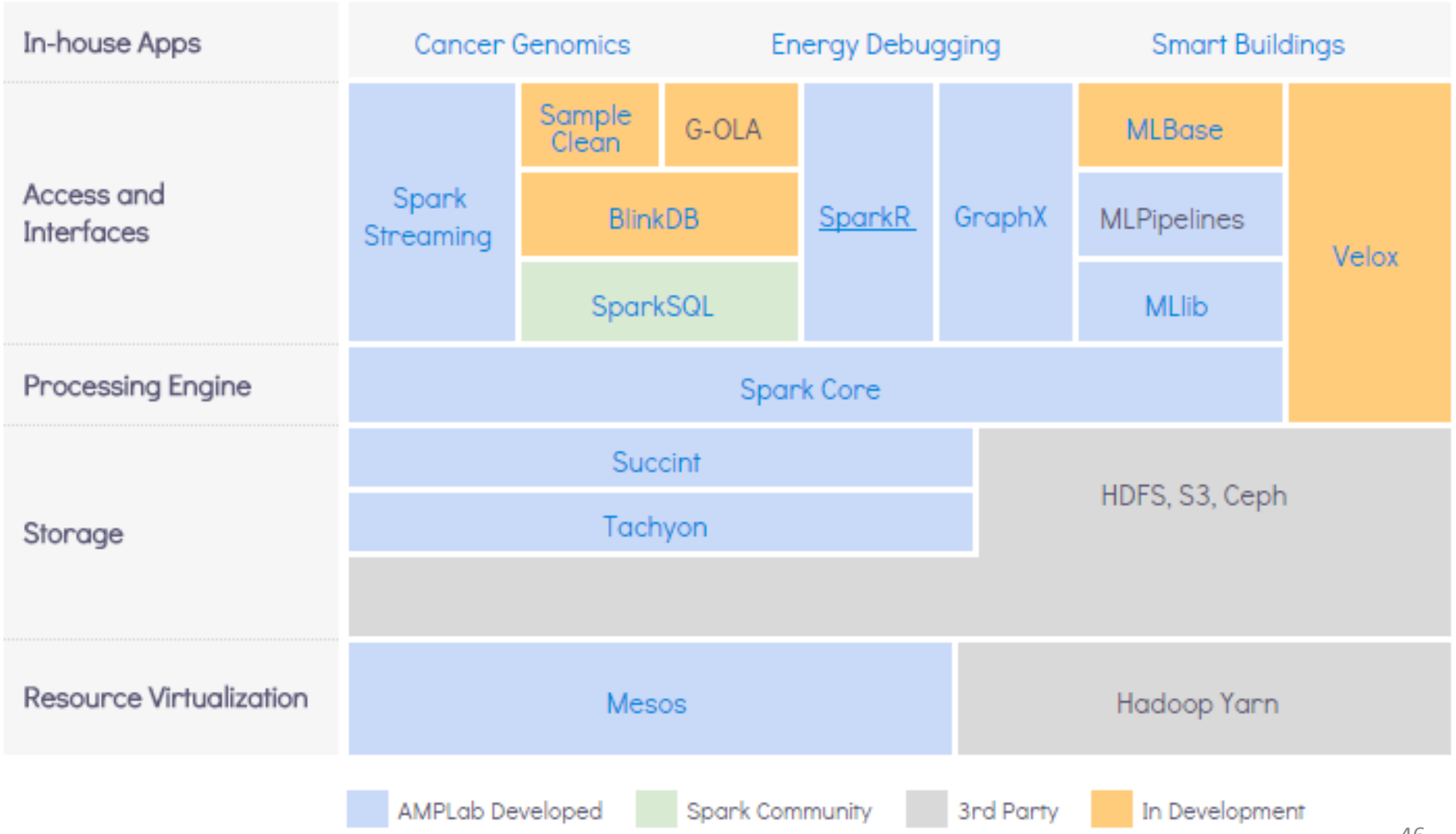
source: http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html


Batch
Interactive
Streaming
...
Spark
Hadoop
Cassandra
Mesos
Cloud Providers
...

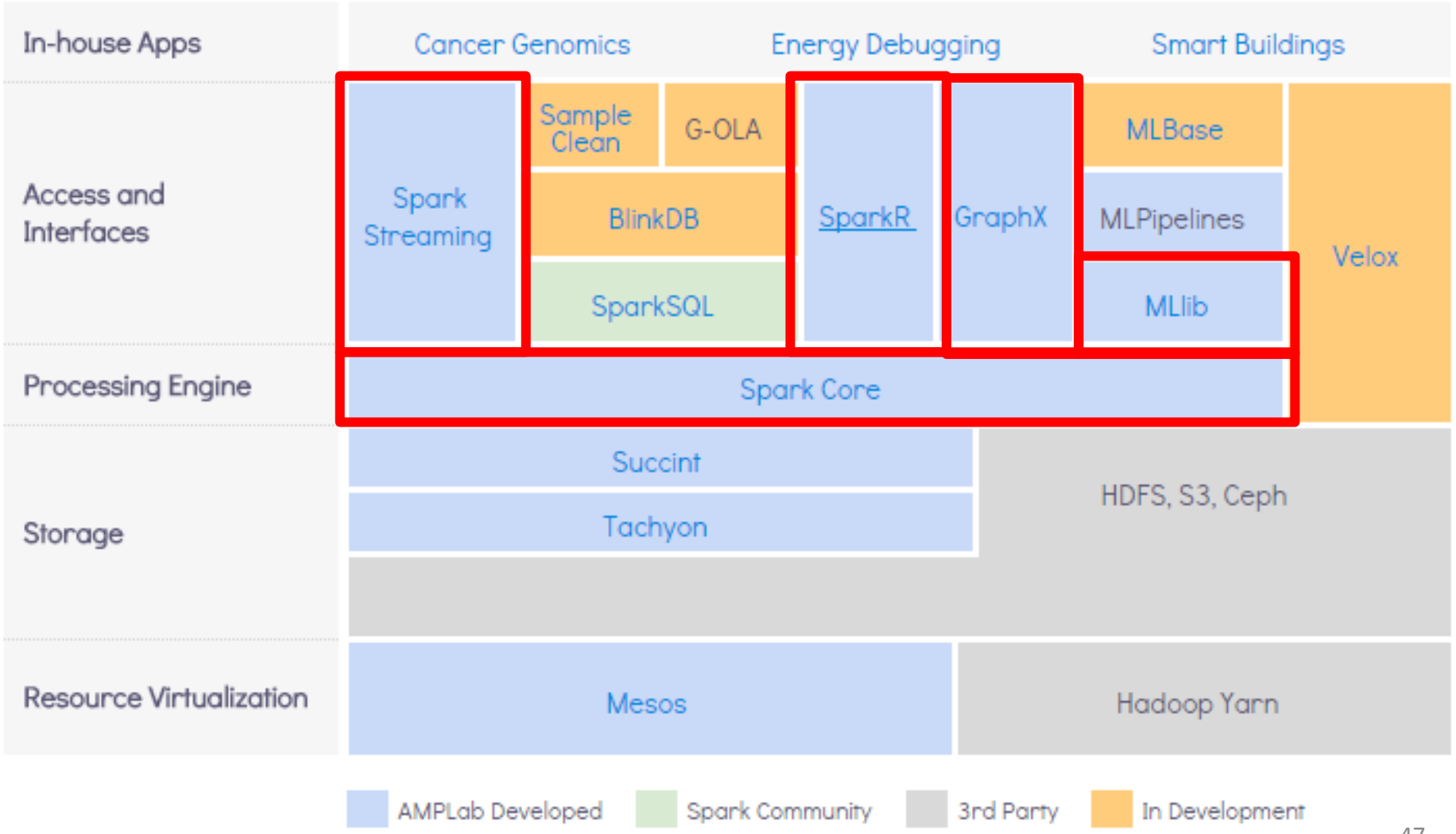
Python Scala Java R



Berkley AMPLab



Berkley AMPLab



Resilient Distributed Datasets (RDD)

RDD of Strings

Immutable **Collection** of Objects

Partitioned and **Distributed**

Stored in **Memory**

Partitions **Recomputed on Failure**

Spark Shell

- Interaktív lekérdezés
- Futtatási lehetőség:
  - Local
  - Cluster (Yarn, Mesos)
- Statikus típus ellenőrzés, autocomplete

RDD Transforms

- RDD-vel térnek vissza

- map(func)
  - új RDD a func-kal alakítva
- flatMap(func)
  - Hasonló mint a map, csak itt több visszatérési értéke is lehet
- filter(func)
  - Csak a kiválasztott elemek szerepelnek a visszatérési halmazban
- Union(otherDatasets)
  - Uniózza az RDD-ket
- Sample(*withReplacement*, *fraction*, *seed*)
  - Adott seed-del töredékeket ad az RDD-ből.
- …

RDD Actions

- Értékkel térnek vissza

- reduce(func)
  - Aggregálja az elemeket. #func(a,b) return c.
- collect()
  - Egy tömböt ad vissza az elemekkel
- count()
  - Visszaadja az RDD-ben szereplő elemek számát.
- first()
  - Az első elemez adja vissza
- saveAsTextFile(path)
  - Eredményt kiírja HDFS-re vagy a lokális diskre.

Példa

```
val titles = sc.textFile("titles.txt")

val countsRdd = titles
        .flatMap(tokenize)
        .map(word => (cleanse(word), 1))
        .reduceByKey(_ + _)

val counts = countsRdd
                .filter{case(_, total) => total > 10000}
                .sortBy{case(_, total) => total}
                .filter{case(word, _) => word.length >= 5}
                .collect
```

Shark (SQL Api)

- Alapja a DataFrame-k
- DataFrame:
  - Elosztott adatset
  - Nevesített oszlopokkal rendelkezik
- Forrás lehet:
  - Hive
  - Külső adatbázis
  - Meglévő RDDs
  - File

Shark (SQL Api)

```
val sc: SparkContext  // An existing SparkContext.
val sqlContext = new org.apache.spark.sql.SQLContext(sc)  // Create the DataFrame
val df = sqlContext.jsonFile("examples/src/main/resources/people.json")

// Show the content of the DataFrame
df.show()
// age name
// null Michael
// 30 Andy // 19 Justin

// Print the schema in a tree format
df.printSchema()
// root
// |-- age: long (nullable = true)
// |-- name: string (nullable = true)
```

Shark (SQL Api)

```
// Select only the "name" column
df.select("name").show()
// name
// Michael
// Andy
// Justin

// Select everybody, but increment the age by 1
df.select(df("name"), df("age") + 1).show()
// name (age + 1)
// Michael null
// Andy 31
// Justin 20
```

Shark (SQL Api)

```
// Select people older than 21
df.filter(df("age") > 21).show()
// age name
// 30 Andy

// Count people by age df.groupBy("age").count().show()
// age count
// null 1
// 19 1
 // 30 1
```

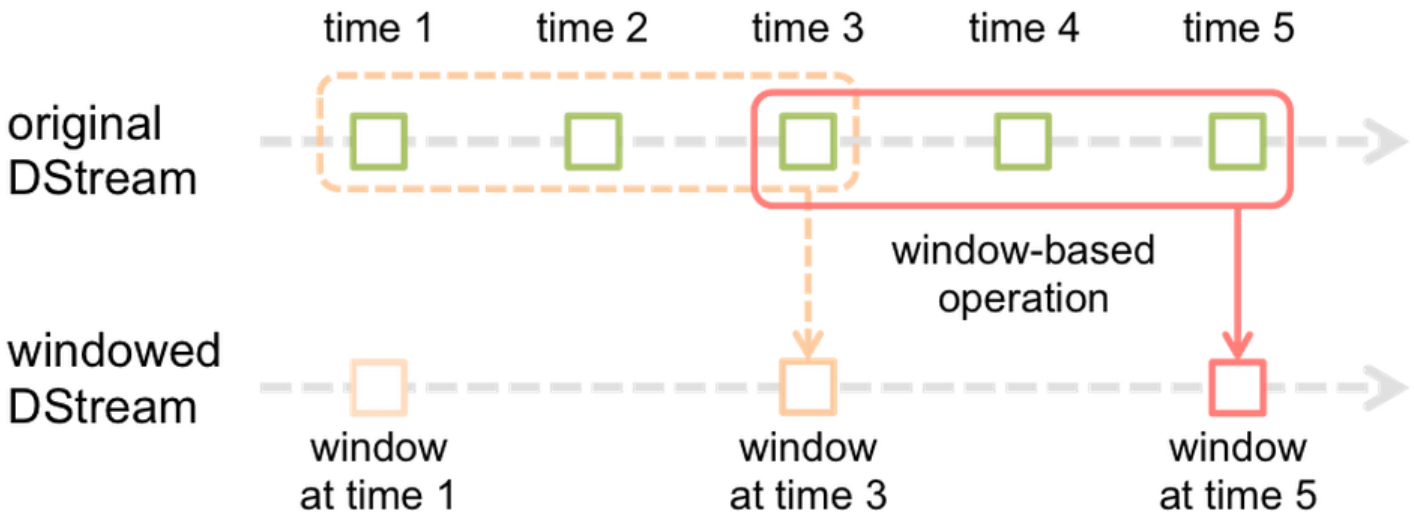
Shark (SQL Api)

SQL Futtatás

```
val sqlContext = ... // An existing SQLContext
val df = sqlContext.sql("SELECT * FROM table")
```

Spark Streaming

- Valósidejű adatfeldolgozás
- Input elosztva a memóriában tárolódik
- Ablakméretnyi RDDs feldolgozása



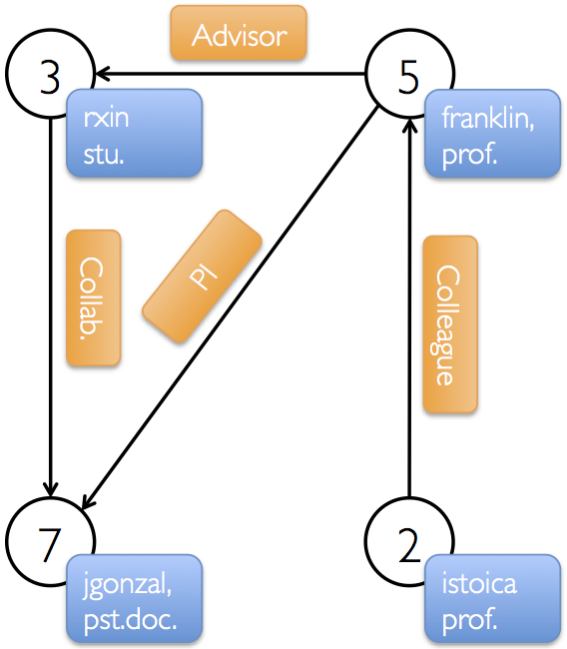
GraphX

- Optimális particionálás és indexelés a csúcsok és élek tárolására
- Alapstruktúra:
  - Property graph
- API
  - Bejárás
  - Elérés
- Alap függvények
  - PageRank
  - Connected components
  - Tringle counting

Property graph

- Felhasználói struktúrák a csúcsokhoz és élekhez

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

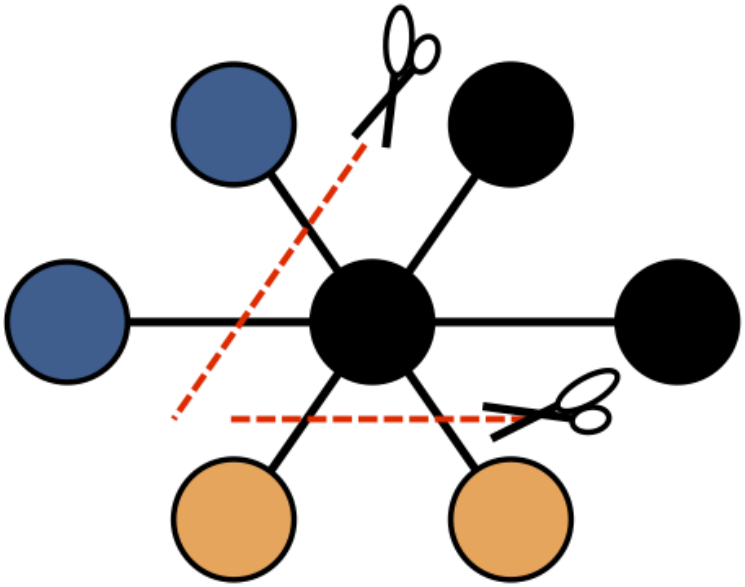
SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Graph operátorok

- Példa:
  - Reverse(VD, ED)
    - Megfordítja az éleket a gráfban
  - Subgraph(…)
    - Részgráfot ad vissza
  - groupEdges(…)
    - Csúcsokat von össze

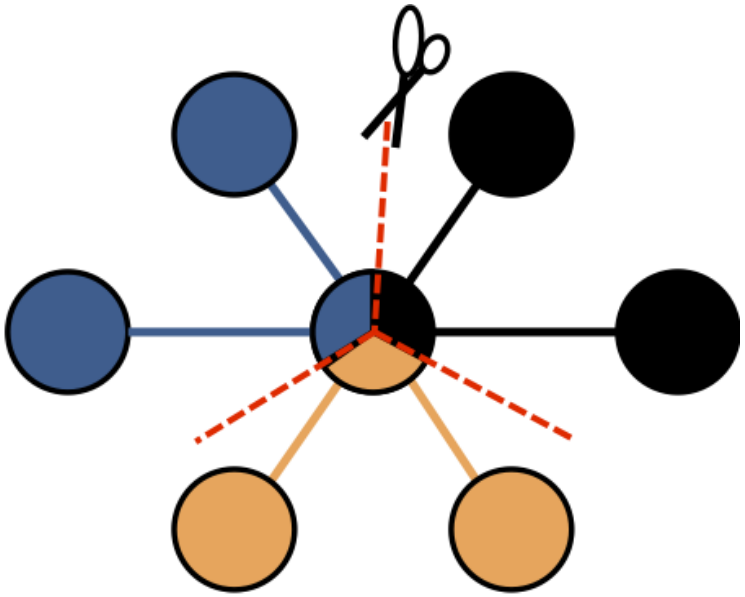
Optimális gráftárolás

Általában



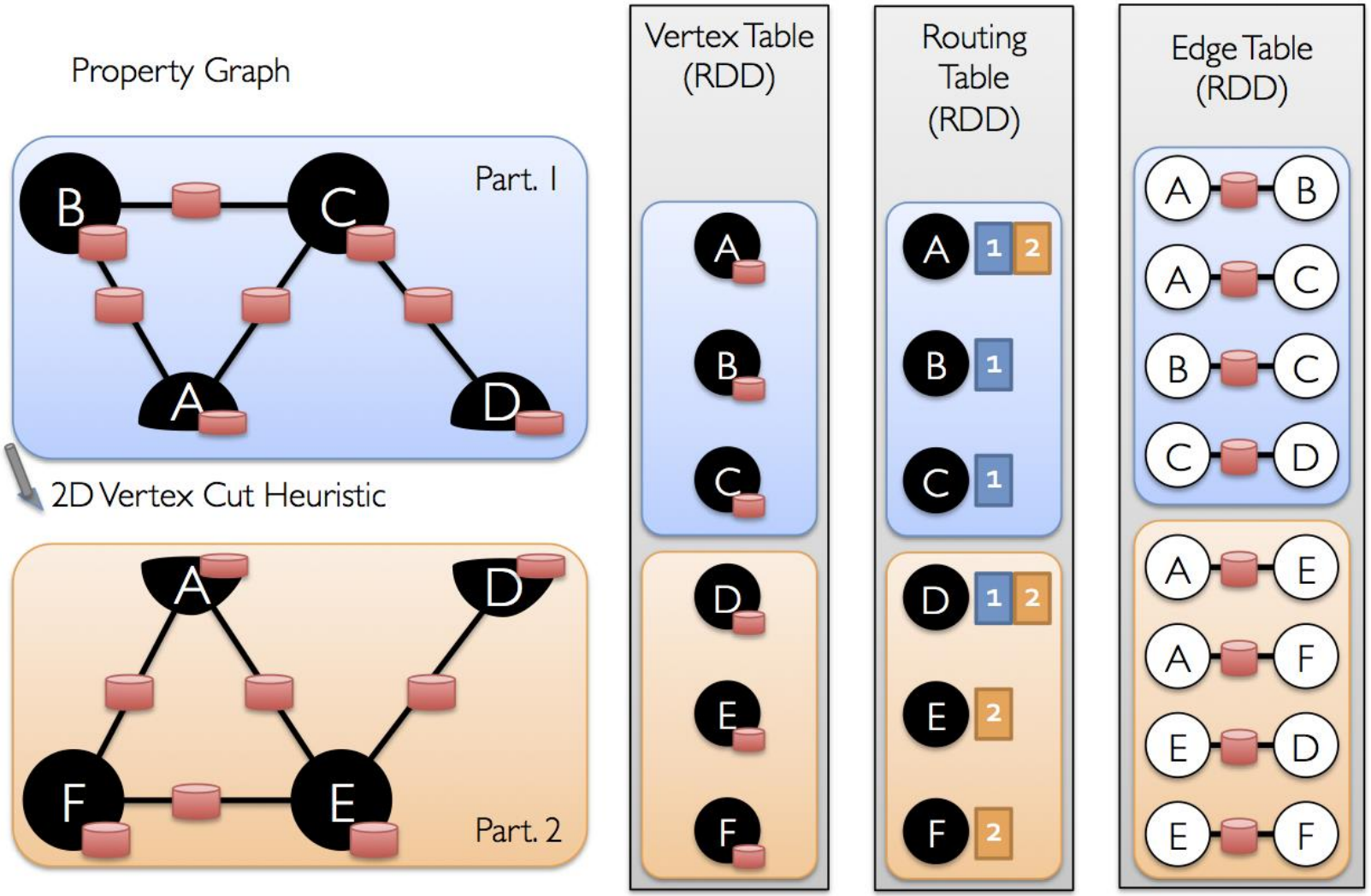
Edge Cut

GraphX



Vertex Cut

Optimális adattárolás



MLib

- Machine Learning Library
- Statistikák, regressziók, döntési fák, kluszterezés, Főkomponens analízis, …
- Iteratív algoritmusok gyorsabbak ha memóriában tárolt adatokon dolgoznak

Köszönöm a Figyelmet!