Big Data Architektúrák és Elemző módszerek

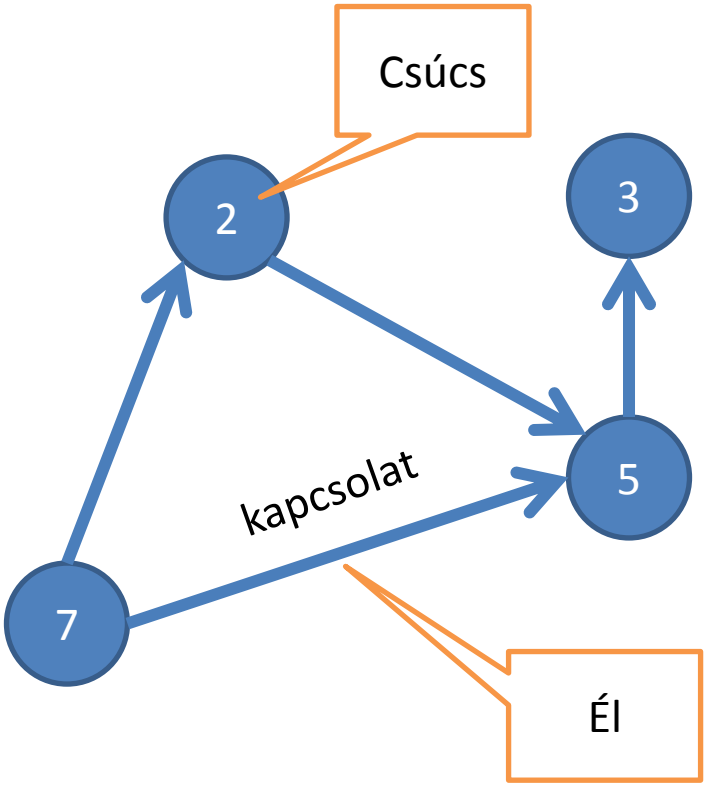
Gombos Gergő, Laki Sándor

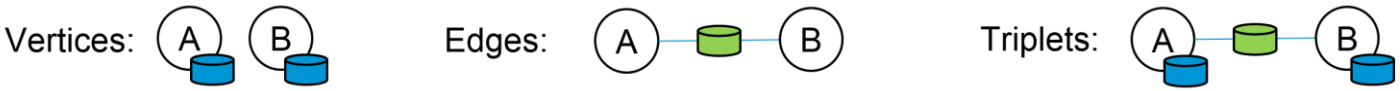
source: cognitiveclass.ai, mapR, Stanford GraphLab

Nagy gráfok feldolgozása

Mi a gráf?

- G = (V,E)
- V : csúcsok
  - attribútommal rendelkeznek
- E : élek
  - attribútomok
  - lehetnek irányítottak, iránytatlanok





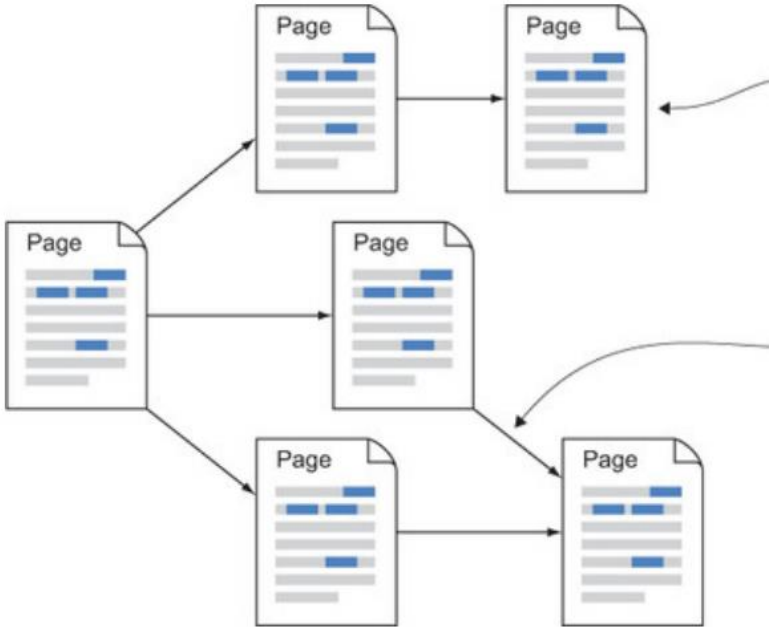
Adatbányászat, Machine Learning feladatok gráfokon

- Megtalálni a befolyásos (influencer) embereket
- Megtalálni összefüggő csoportokat
- Ajánlásokat adni termékekről

….

Valós gráfok

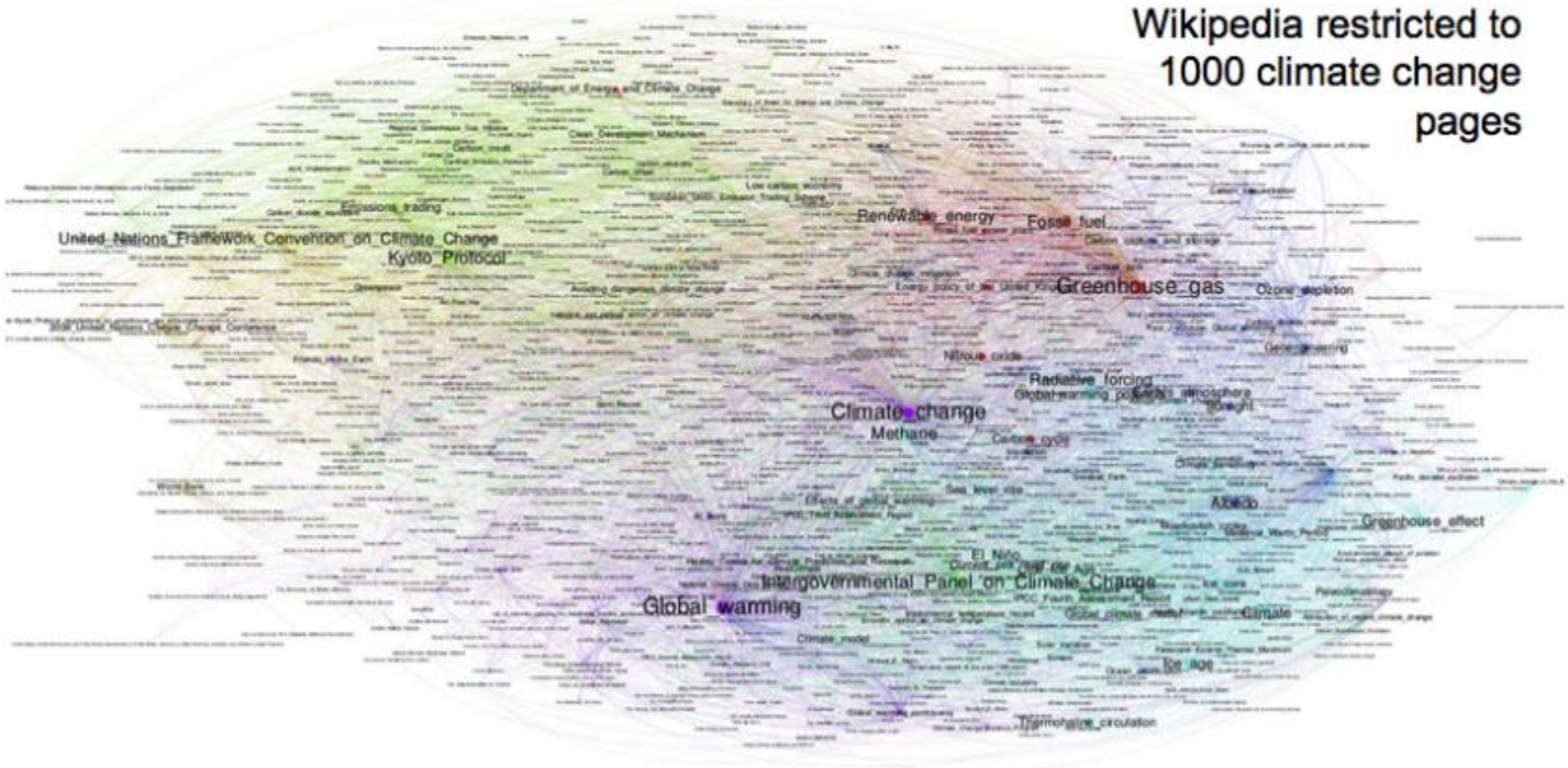
- Weboldalak



weboldalak

linkek a weboldalak között

Web Graphs



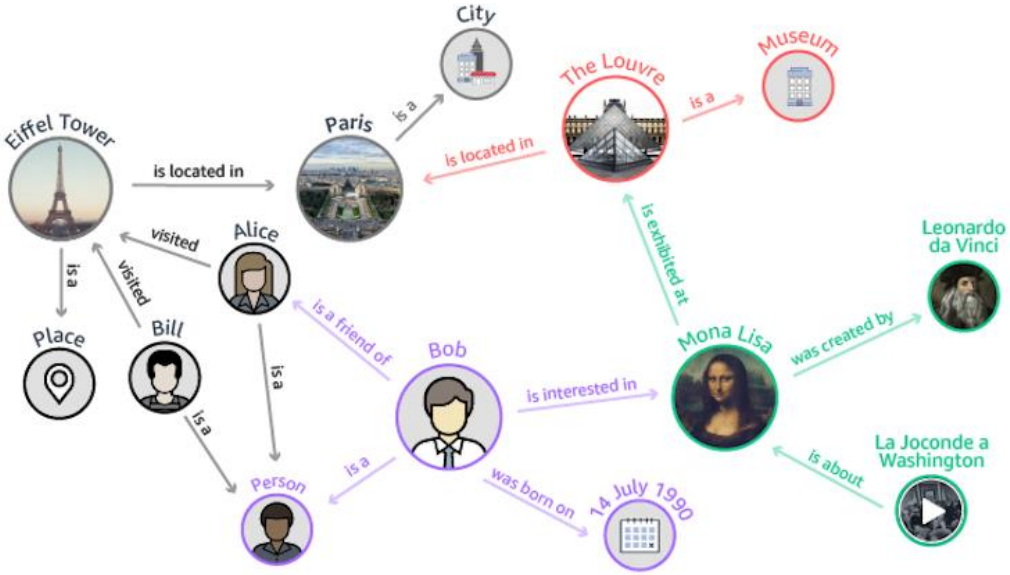
- Vertices: *Web-pages*
- Edges: *Links (Directed)*

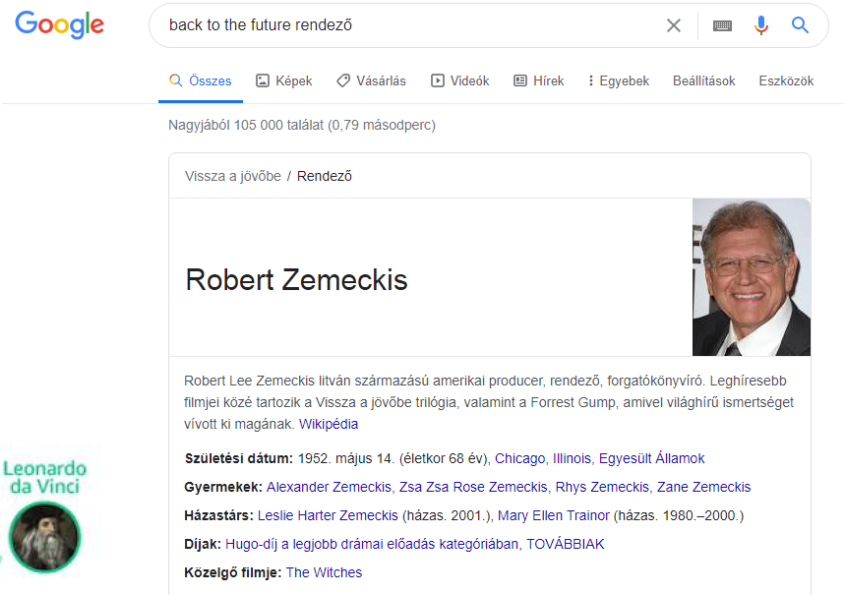
Generated Content:

- Click-streams

Valós gráfok

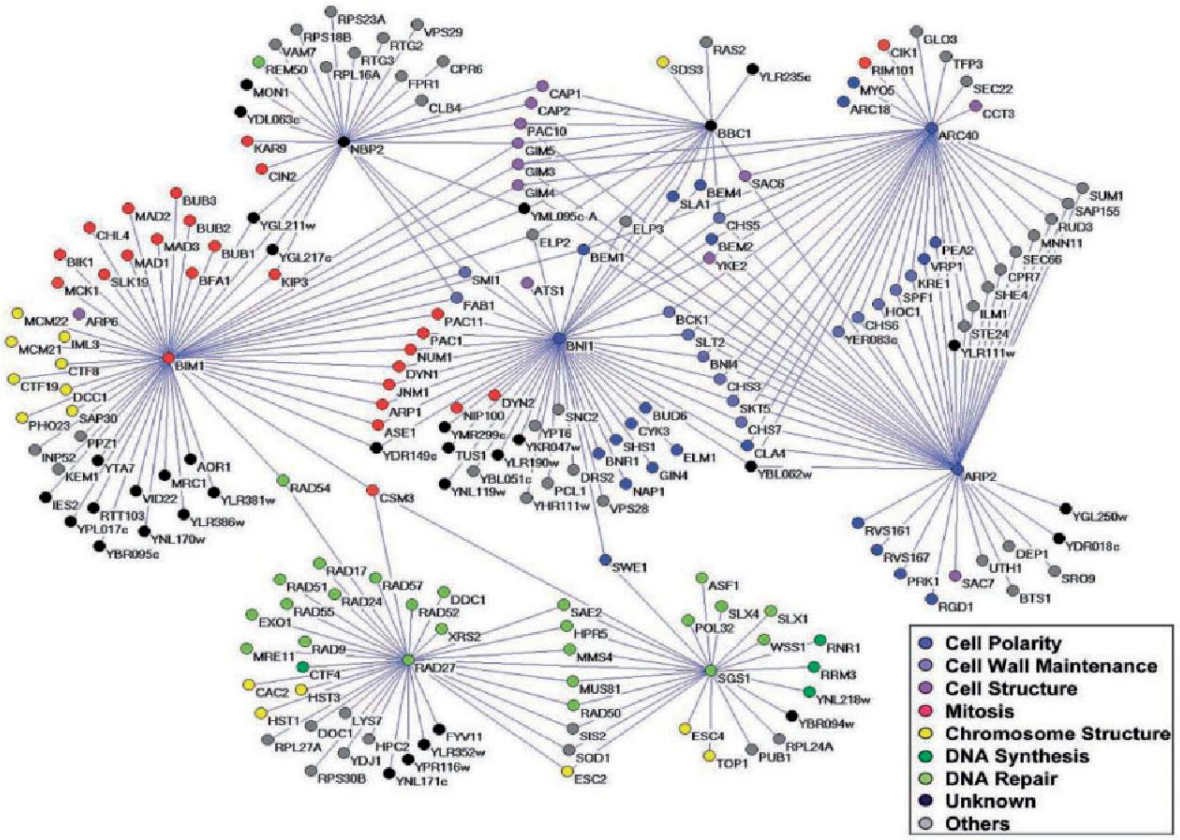
- Tudásbázisok, tudás hálók
  - Google Knowledge gráf





Valós gráfok

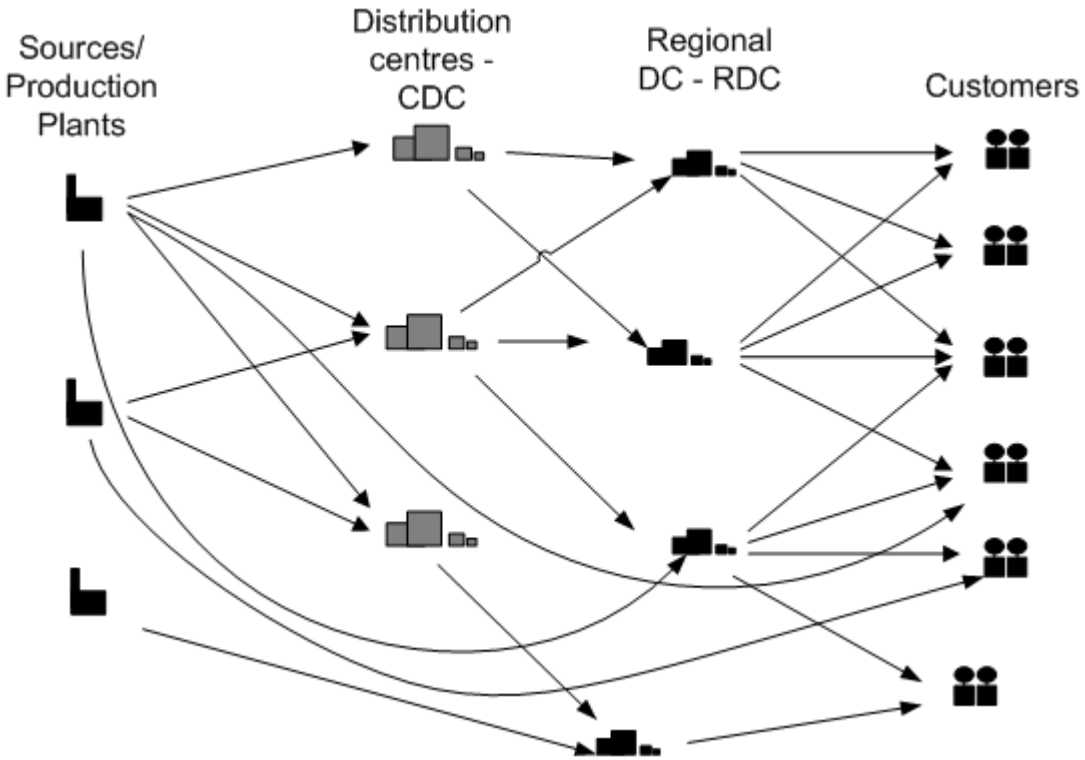
- Biológia gráfok
  - DNS kapcsolatok
  - Gyógyszer alkotóelemek hatáskapcsolata
  - protein kapcsolatok



X . Ma, Lin Gao: **Biological network analysis: insights into structure and functions.**

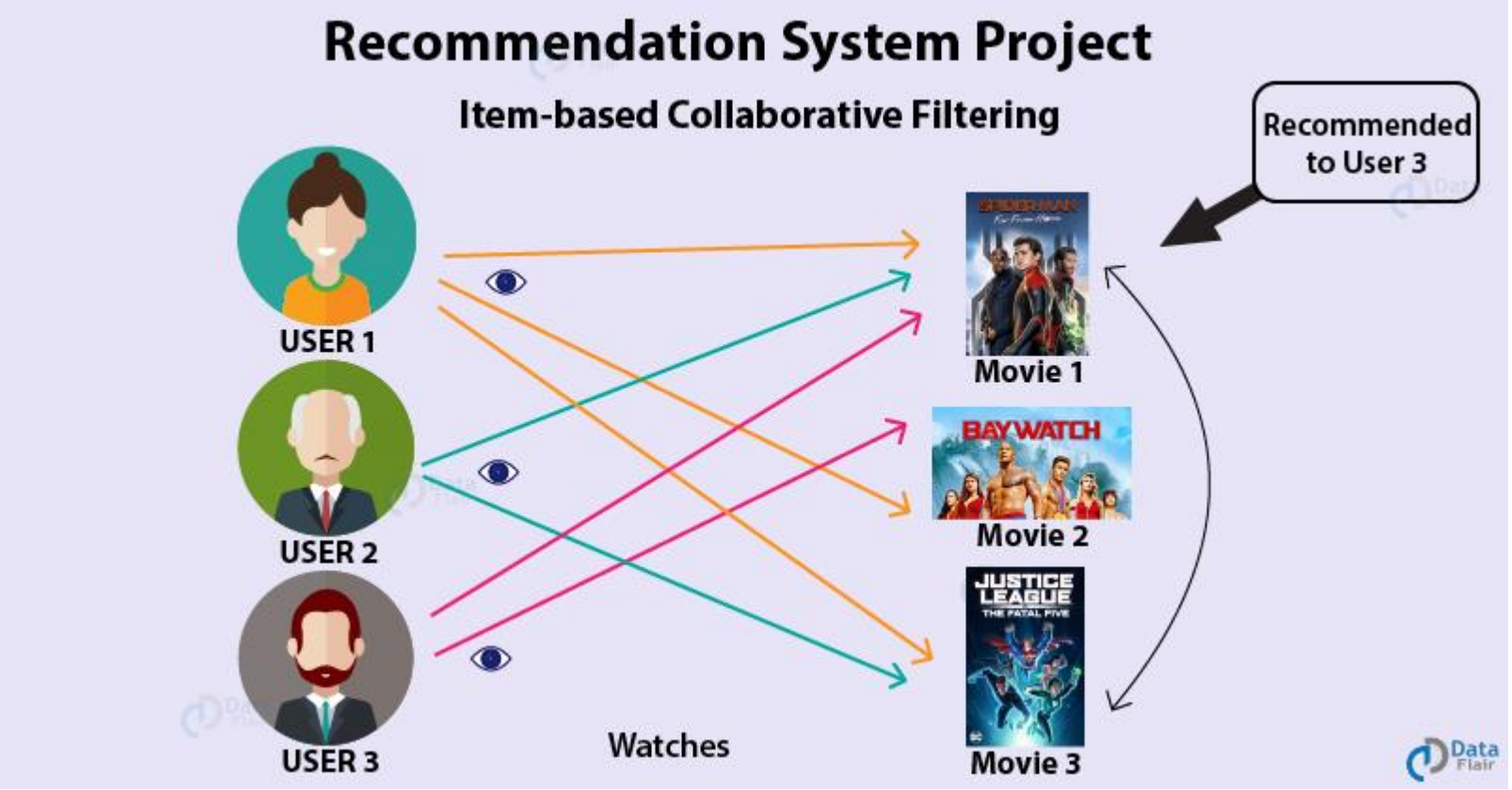
Valós gráfok

- Logisztika, útvonal keresés



Valós gráfok

- Ajánlások

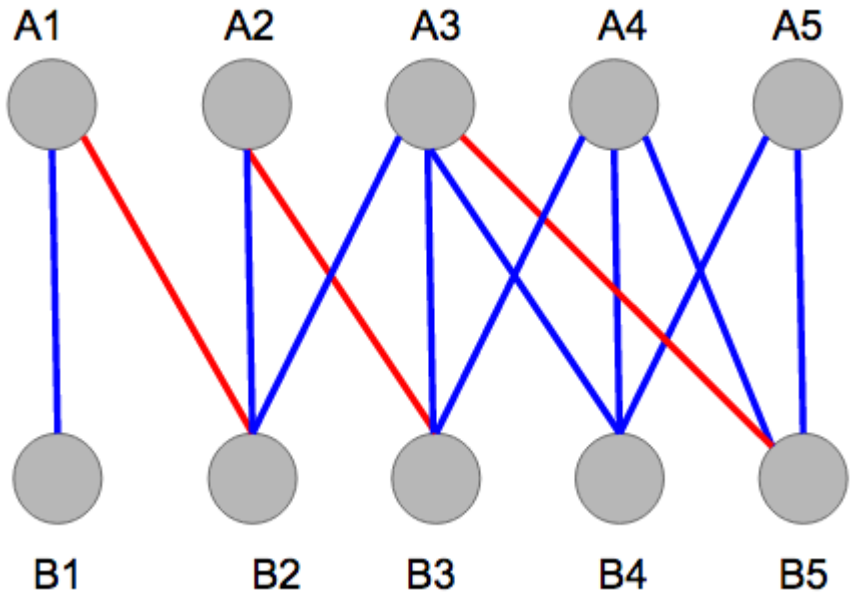


Gráf feldolgozási feladatok

- Alap statisztikák
  - Pl: Bemenő/kimenő élek
- Származtatott tulajdonságok a gráfból
  - Klaszterezési együttható
- Legrövidebb út keresés
  - Internet (routing)

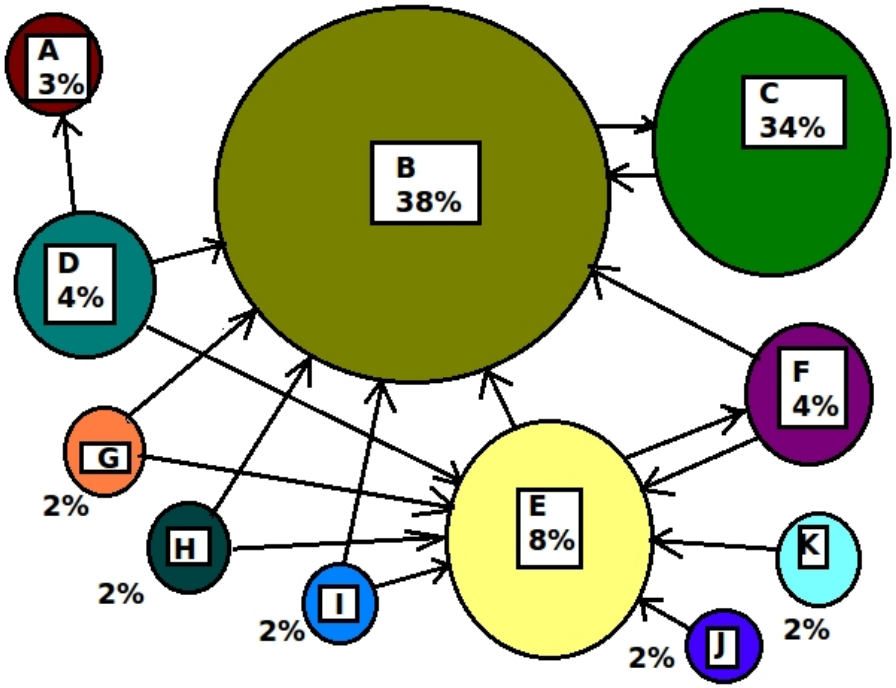
Gráf feldolgozási feladatok

- Párosítás
  - Randi oldalak (match.com)



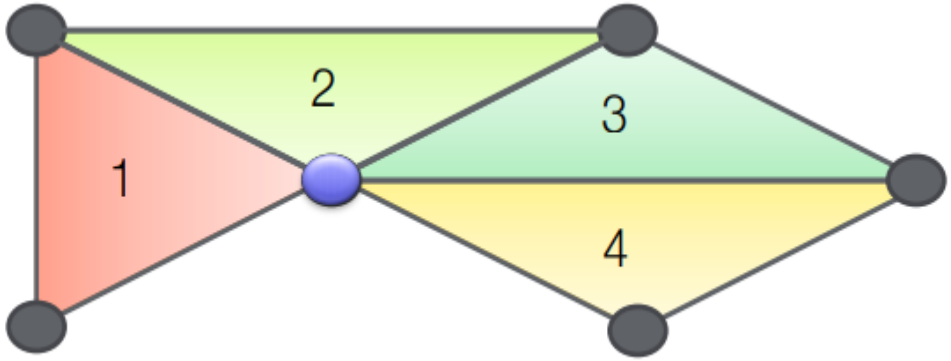
Gráf feldolgozási feladatok

- PageRank
  - Node fontosság meghatározás

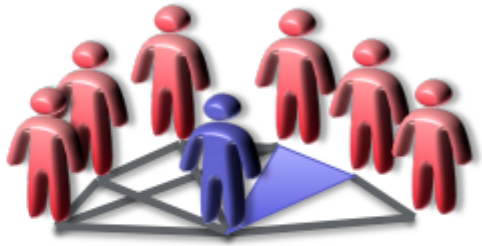


Gráf feldolgozási feladatok

- Háromszögek keresése
  - Társaságok megtalálása



Fewer Triangles
Weaker Community

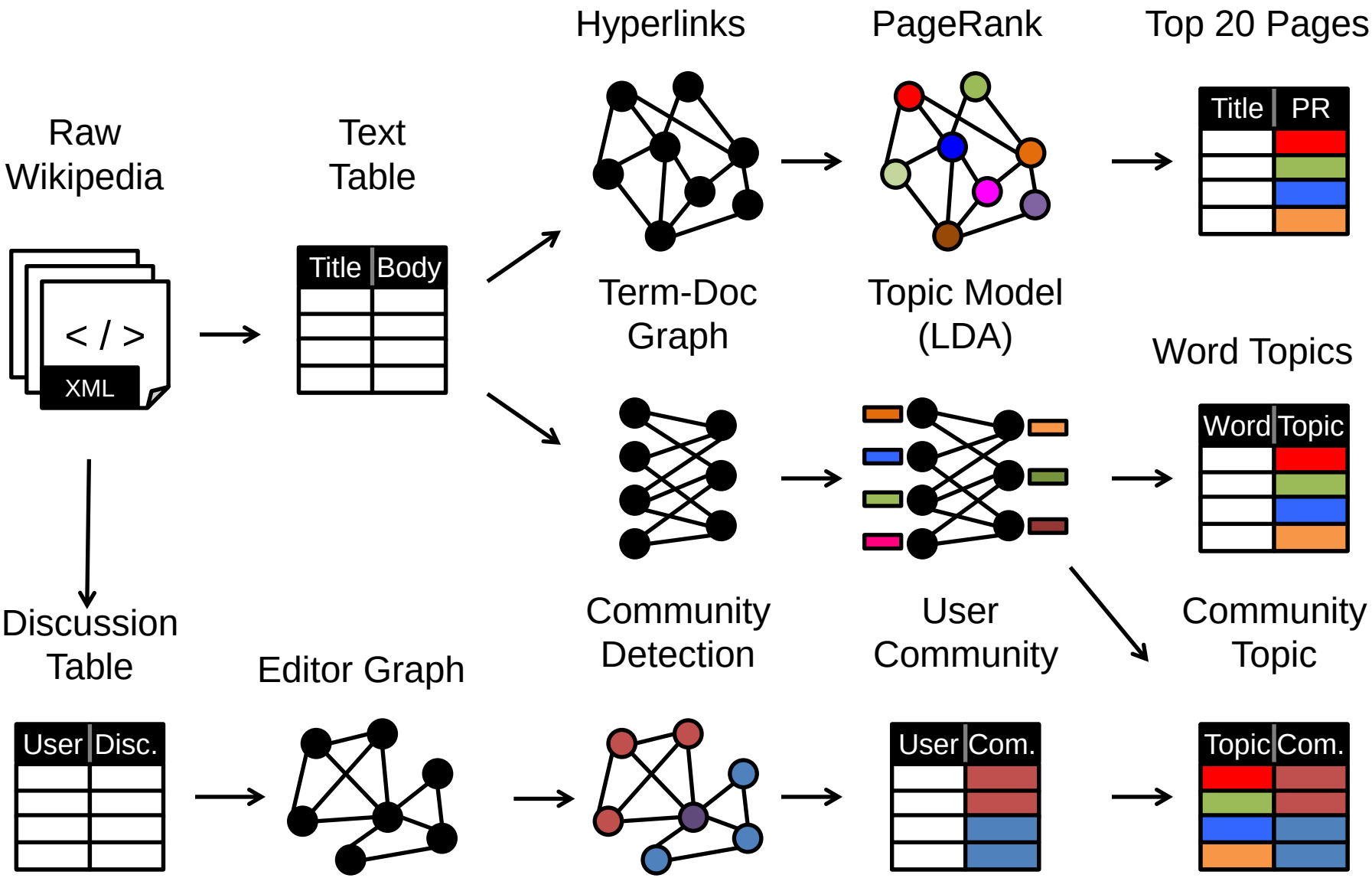


More Triangles
Stronger Community

Kihívások

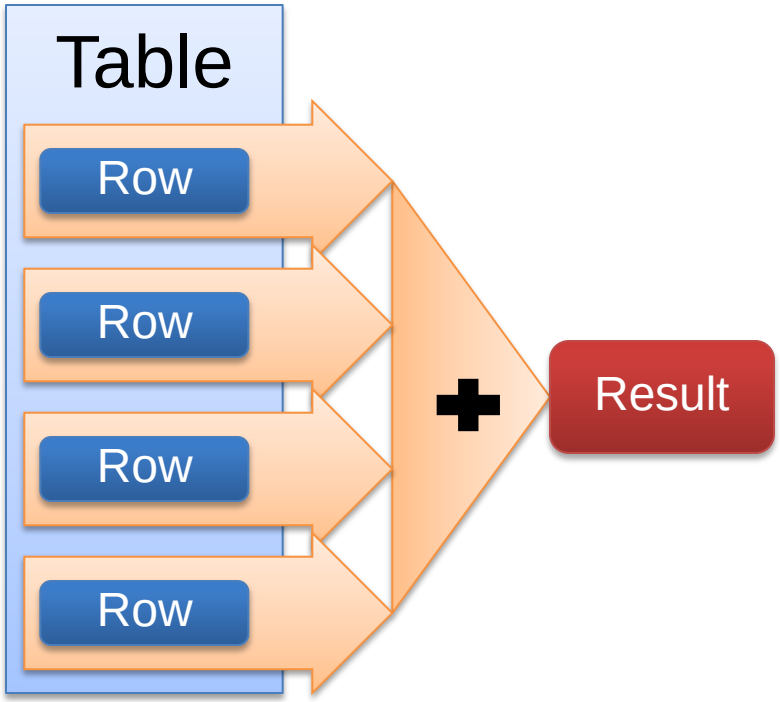
- Gráfok nagyok
  - Milliós nagyságrendű csúcsok és élek
- Tárolás, feldolgozás egy serveren:
  - Memória limit
  - Lassú
- Tárolás elosztva:
  - Sok kommunikáció a gépek között

A Gráfok fontosak az elemzésekben

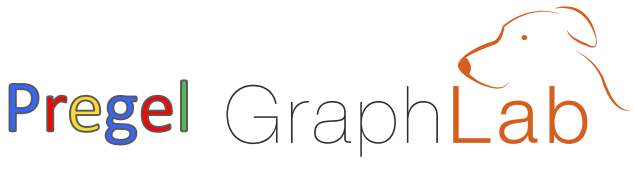


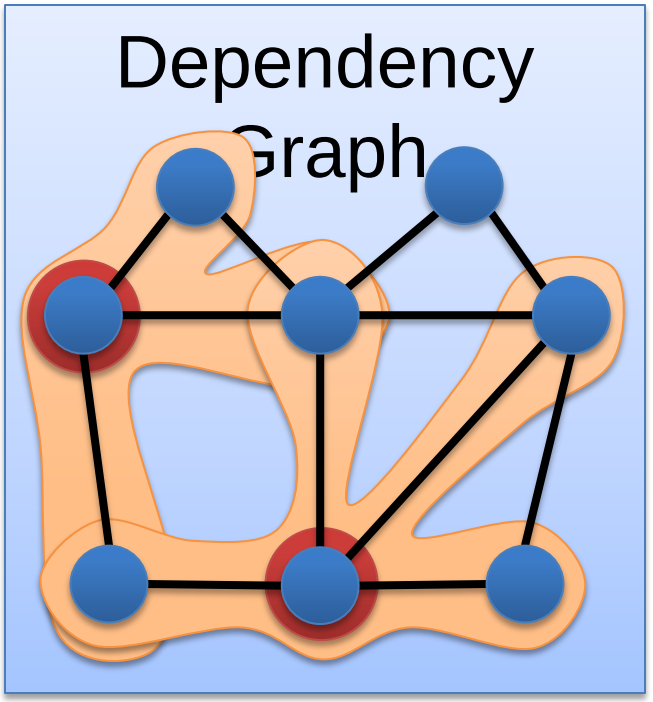
Különböző rendszerek más megközelítés

Táblás nézet



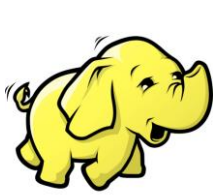
Gráfos nézet





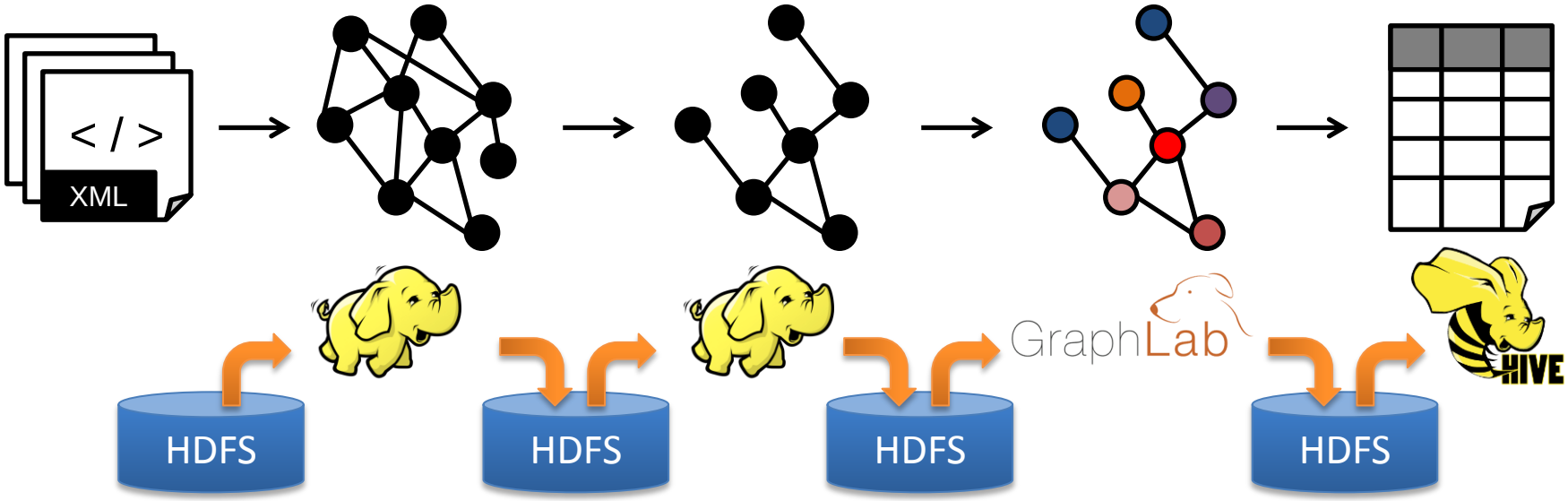
Nehézkes használat

- A felhasználóknak különböző rendszereket kell megtanulni, használni
- Sokszor nehézkes interface-ken keresztül



Nem hatékony

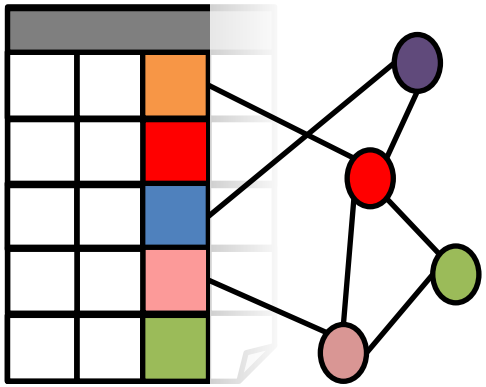
- Költséges az adatok másolás a hálózaton és a fájlrendszeren a különböző rendszerek között
- Nem lehet felhasználni a korábbi adatokat



Megoldás: A graphX

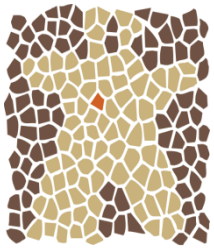
Új API

Elmossa a különbséget a táblák és a gráfok között



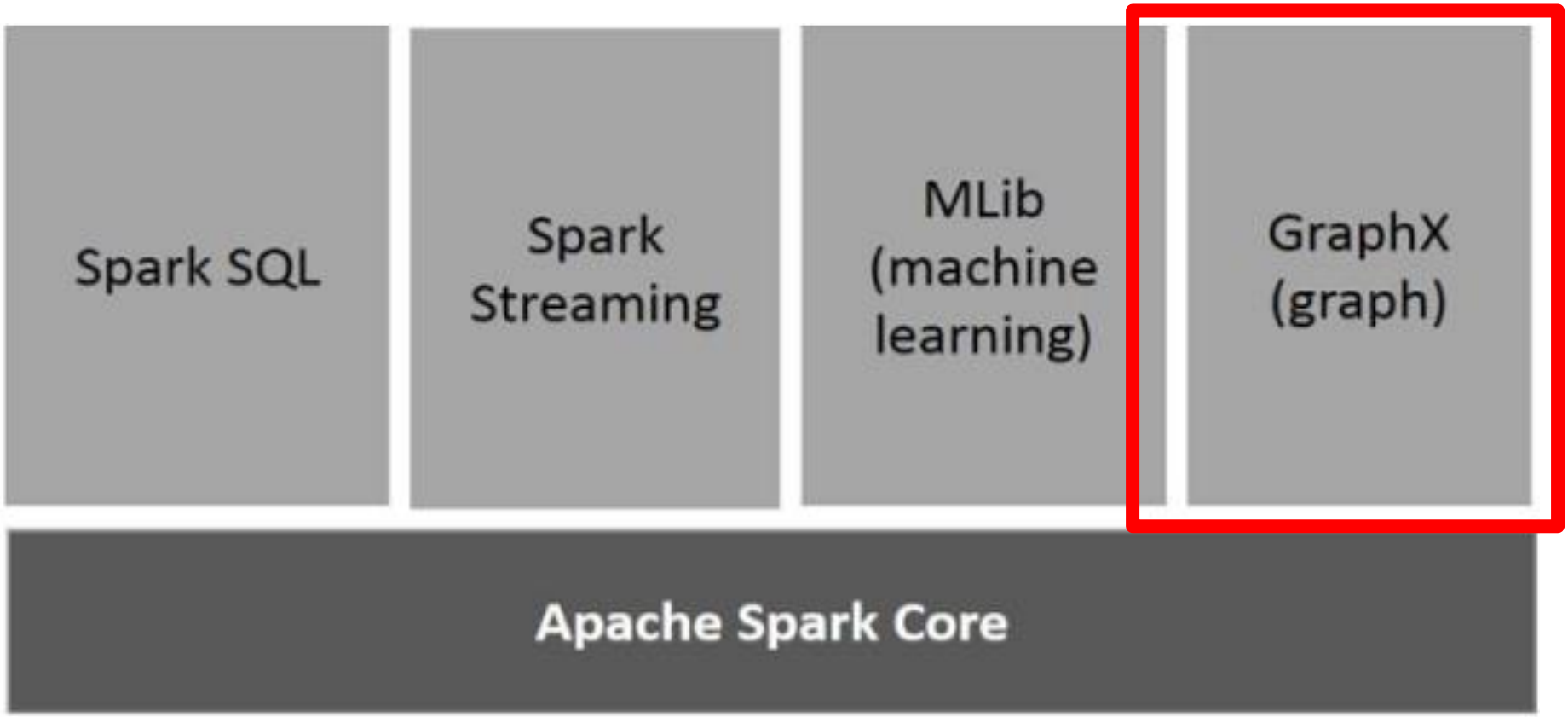
Új rendszer

Egységesíti a párhuazmos gráffeldolgozó rendszereket

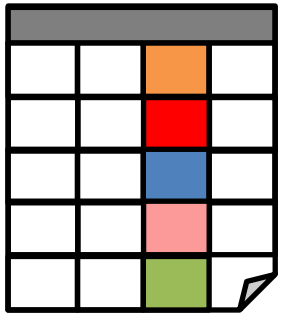


Egyszerű programozás

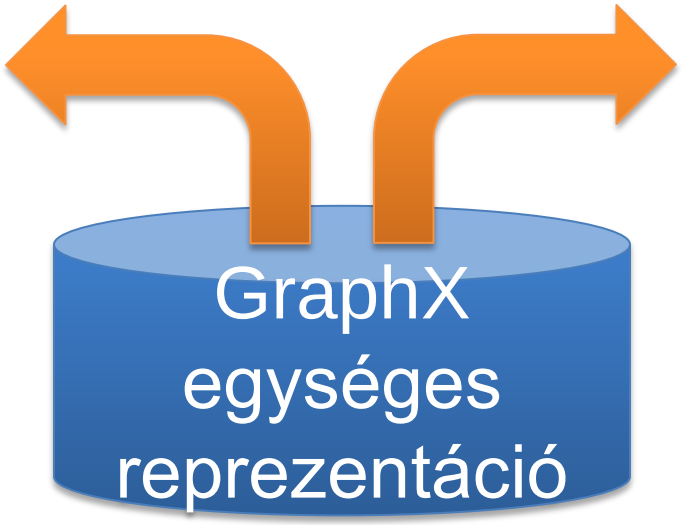
Spark eszközök

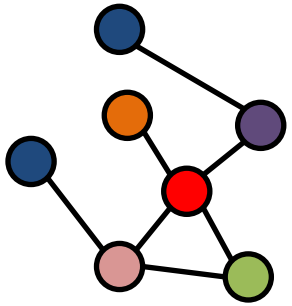


A táblázatos és gráfos megjelenítése is van ugyanannak a fizikai adatnak



Tábla nézet



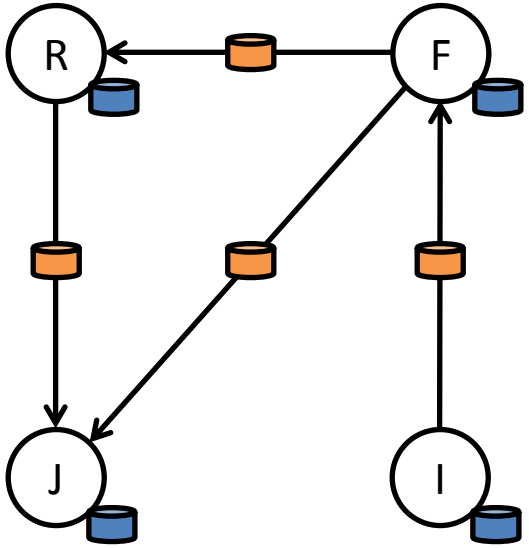


Gráfos nézet

Mindkét nézetnek saját operátorai vannak a hatékonyság miatt

Gráf táblás reprezentációja

Property Graph



Vertex Property Table

Id	Property (V)
Rxin	(Stu., Berk.)
Jegonzal	(PstDoc, Berk.)
Franklin	(Prof., Berk)
Istoica	(Prof., Berk)

Edge Property Table

SrcId	DstId	Property (E)
rxin	jegonzal	Friend
franklin	rxin	Advisor
istoica	franklin	Coworker
franklin	jegonzal	PI

Tábla műveletek

- Table (RDD) operátorok a Spark-ból származnak:

map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	cross	save
rightOuterJoin	zip	...

Gráf operátorok

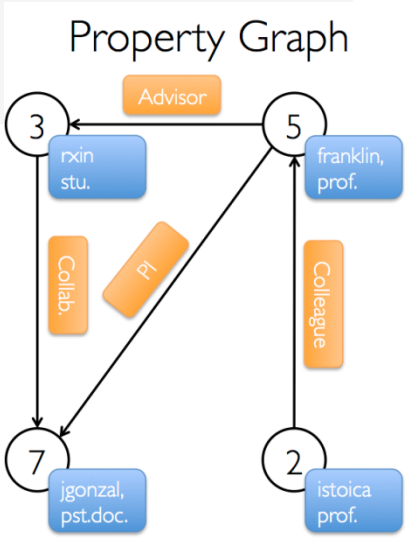
```scala
// Information about the Graph ===================================================================
val numEdges: Long
val numVertices: Long
val inDegrees: VertexRDD[Int]
val outDegrees: VertexRDD[Int]
val degrees: VertexRDD[Int]
// Views of the graph as collections =============================================================
val vertices: VertexRDD[VD]
val edges: EdgeRDD[ED]
val triplets: RDD[EdgeTriplet[VD, ED]]
// Functions for caching graphs ==================================================================
def persist(newLevel: StorageLevel = StorageLevel.MEMORY_ONLY): Graph[VD, ED]
def cache(): Graph[VD, ED]
def unpersistVertices(blocking: Boolean = true): Graph[VD, ED]
// Change the partitioning heuristic  ============================================================
def partitionBy(partitionStrategy: PartitionStrategy): Graph[VD, ED]
// Transform vertex and edge attributes ==========================================================
def mapVertices[VD2](map: (VertexID, VD) => VD2): Graph[VD2, ED]
def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]
def mapEdges[ED2](map: (PartitionID, Iterator[Edge[ED]]) => Iterator[ED2]): Graph[VD, ED2]
def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
def mapTriplets[ED2](map: (PartitionID, Iterator[EdgeTriplet[VD, ED]]) => Iterator[ED2])
  : Graph[VD, ED2]
```

Gráf operátorok

```scala
// Modify the graph structure ==========================================================
def reverse: Graph[VD, ED]
def subgraph(
    epred: EdgeTriplet[VD,ED] => Boolean = (x => true),
    vpred: (VertexID, VD) => Boolean = ((v, d) => true))
  : Graph[VD, ED]
def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
// Join RDDs with the graph ============================================================
def joinVertices[U](table: RDD[(VertexID, U)])(mapFunc: (VertexID, VD, U) => VD): Graph[VD, ED]
def outerJoinVertices[U, VD2](other: RDD[(VertexID, U)])
    (mapFunc: (VertexID, VD, Option[U]) => VD2)
  : Graph[VD2, ED]
// Aggregate information about adjacent triplets =======================================
def collectNeighborIds(edgeDirection: EdgeDirection): VertexRDD[Array[VertexID]]
def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[(VertexID, VD)]]
def aggregateMessages[Msg: ClassTag](
    sendMsg: EdgeContext[VD, ED, Msg] => Unit,
    mergeMsg: (Msg, Msg) => Msg,
    tripletFields: TripletFields = TripletFields.All)
  : VertexRDD[A]
```

GraphX

```scala
// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                       (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                       Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
```



Vertex Table

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

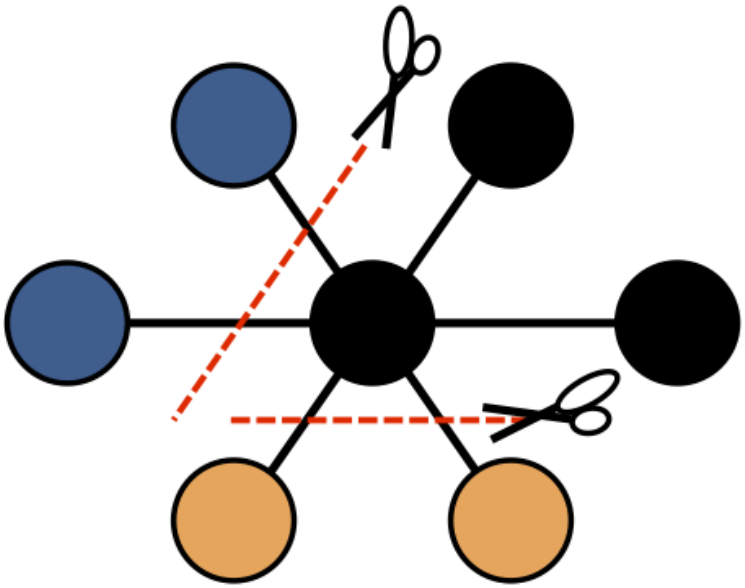
Edge Table

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

GraphX Achitektúra

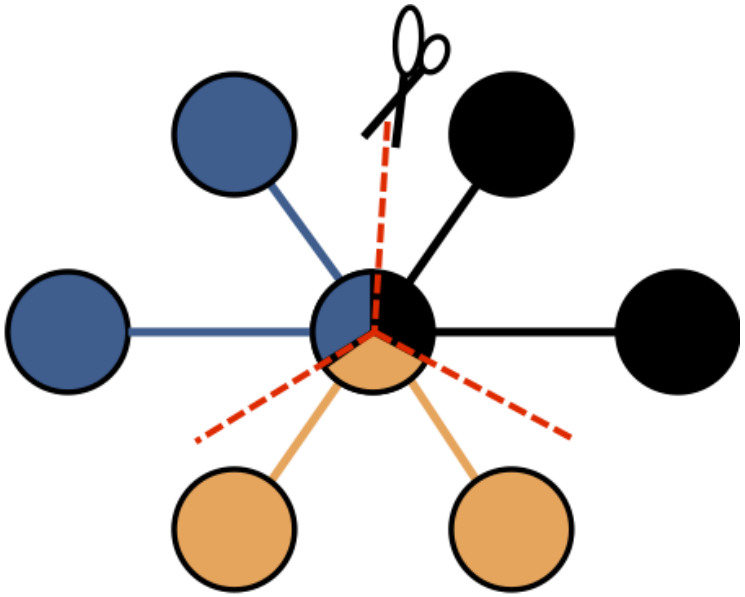
Optimális gráftárolás

Általában



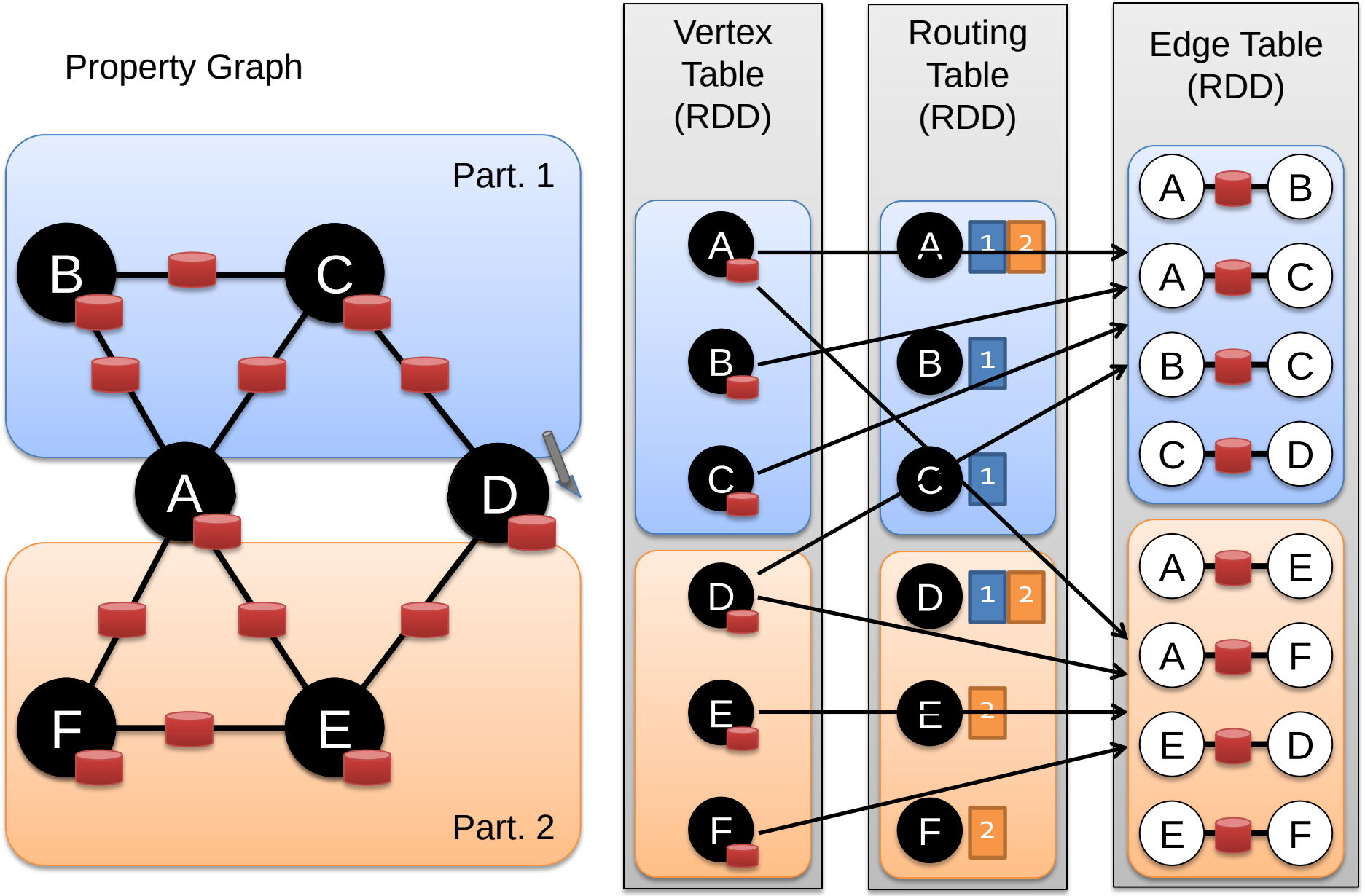
Edge Cut

GraphX



Vertex Cut

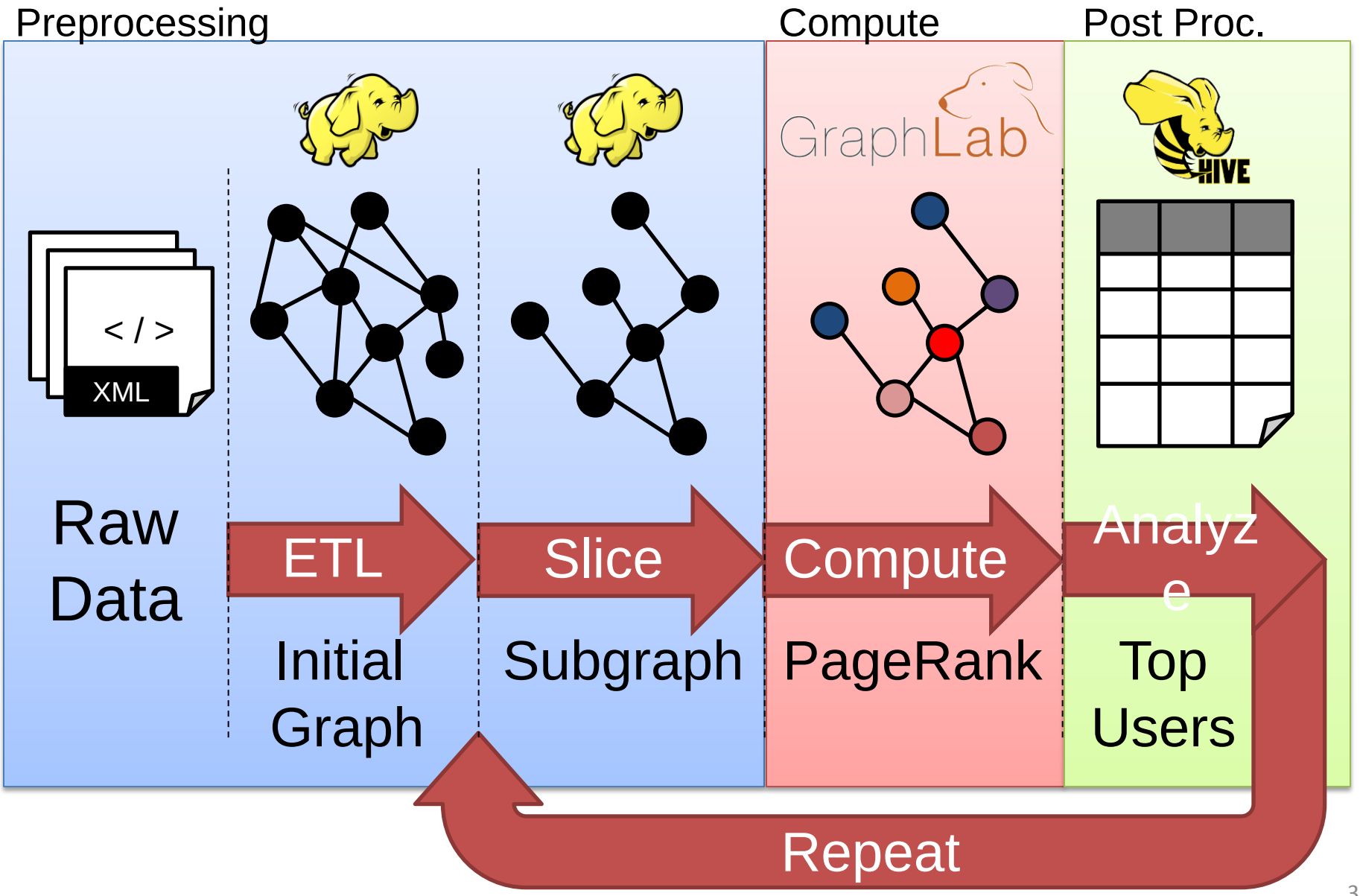
Elosztott gráf mint táblák (RDDk)



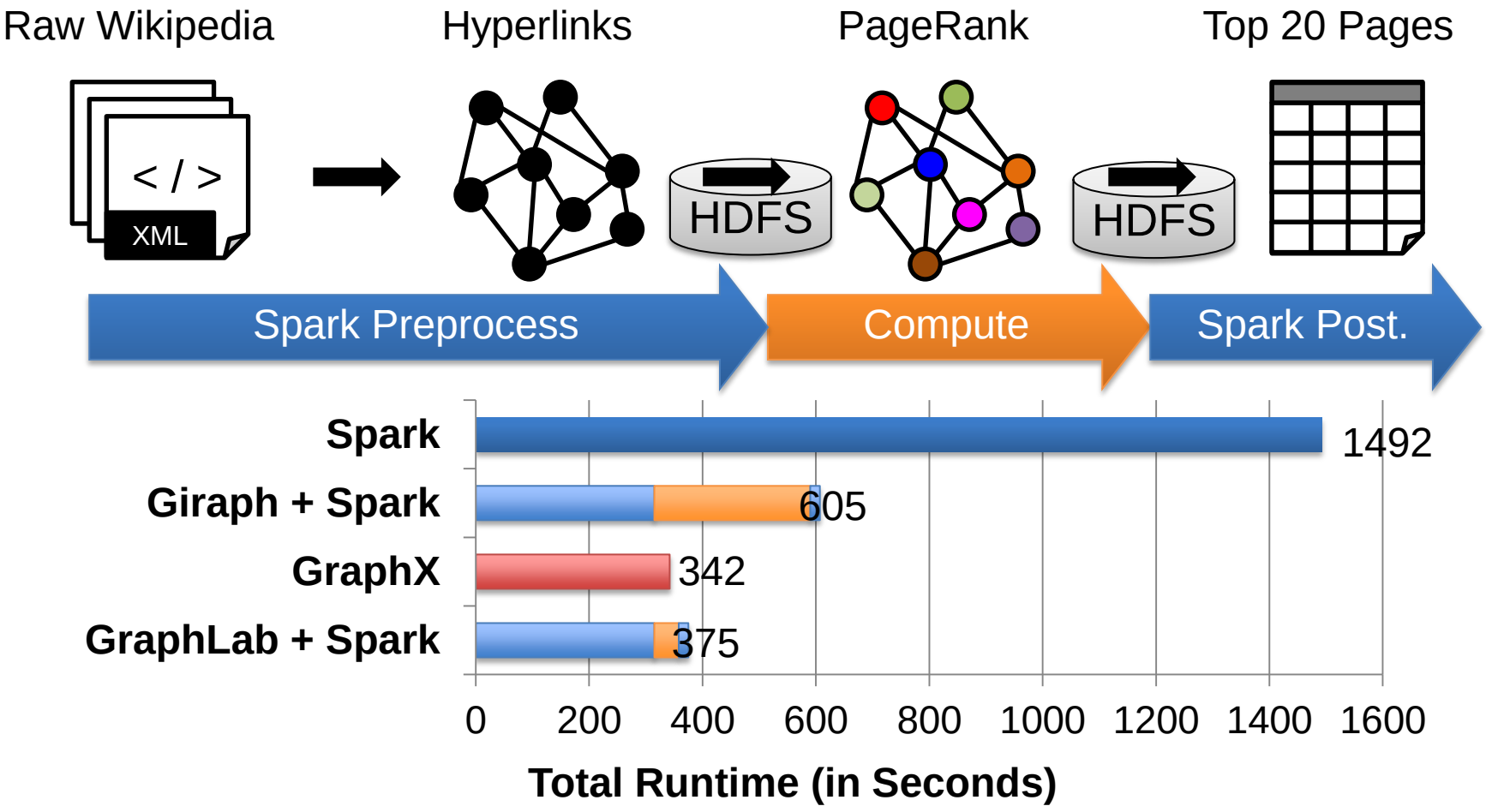
Csúcs tárolás

- 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

Gráf elemzés Pipeline

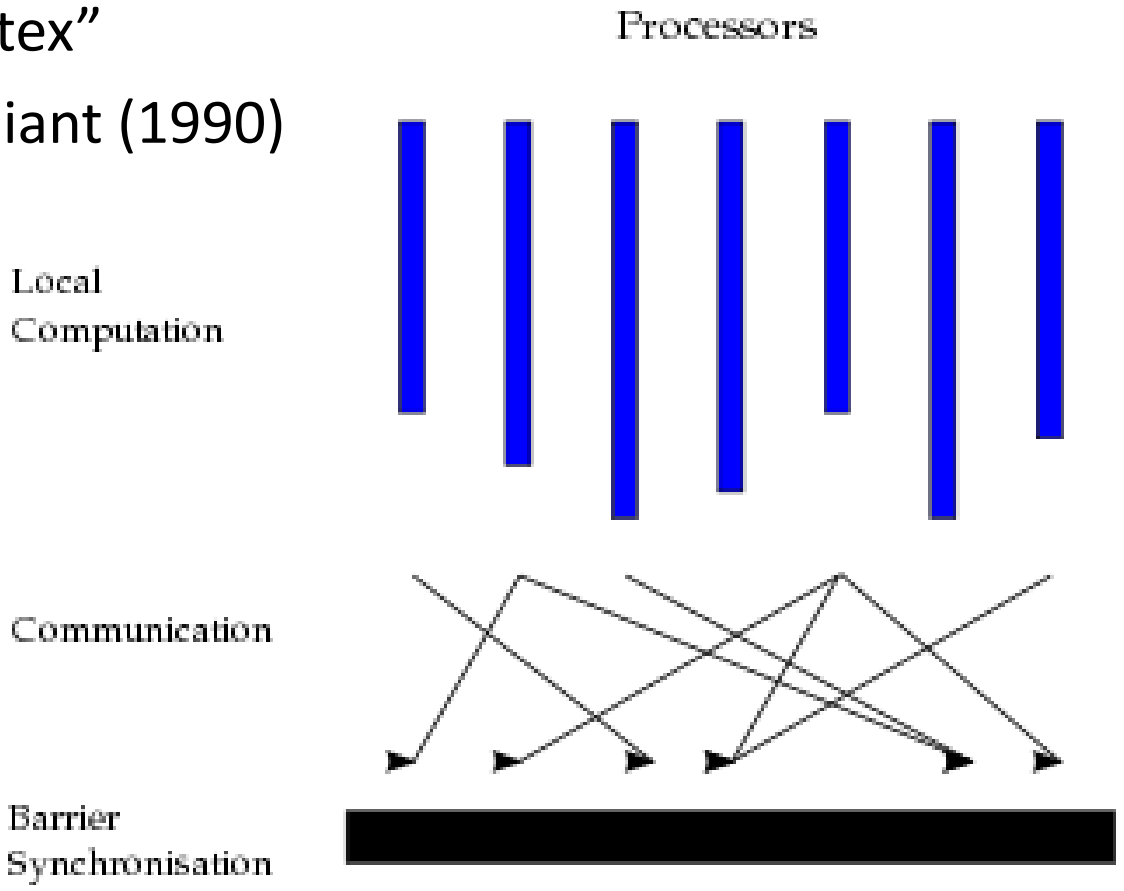


Gráf elmezés pipeline



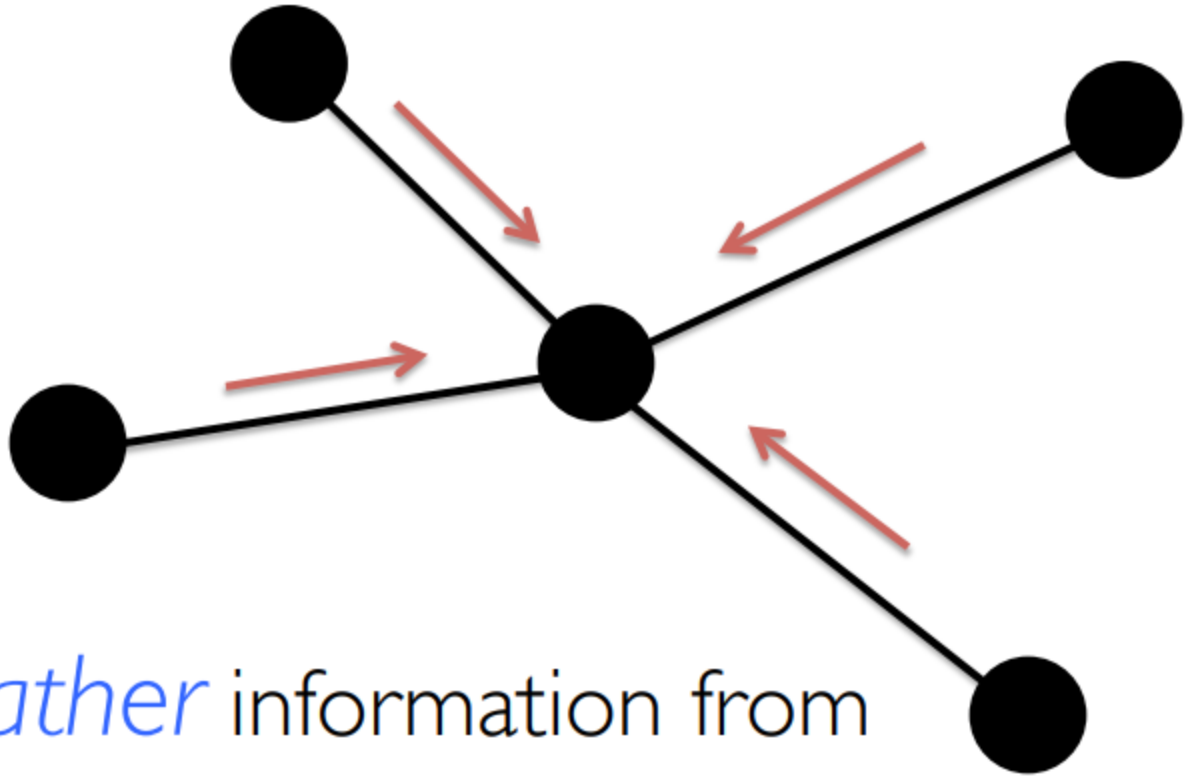
Bulk Synchronous Parallel(BSP) Model

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



Graph-Parallel Pattern

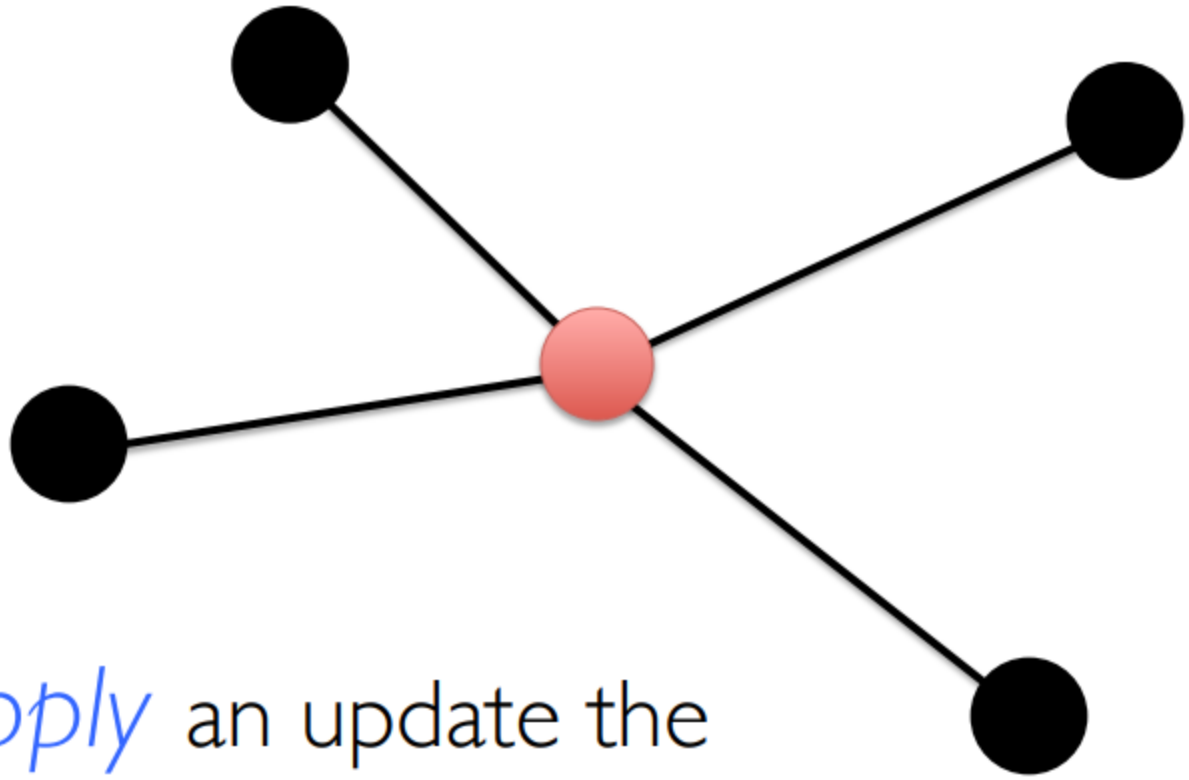
Gonzalez et al. [OSDI'12]



Gather information from neighboring vertices

Graph-Parallel Pattern

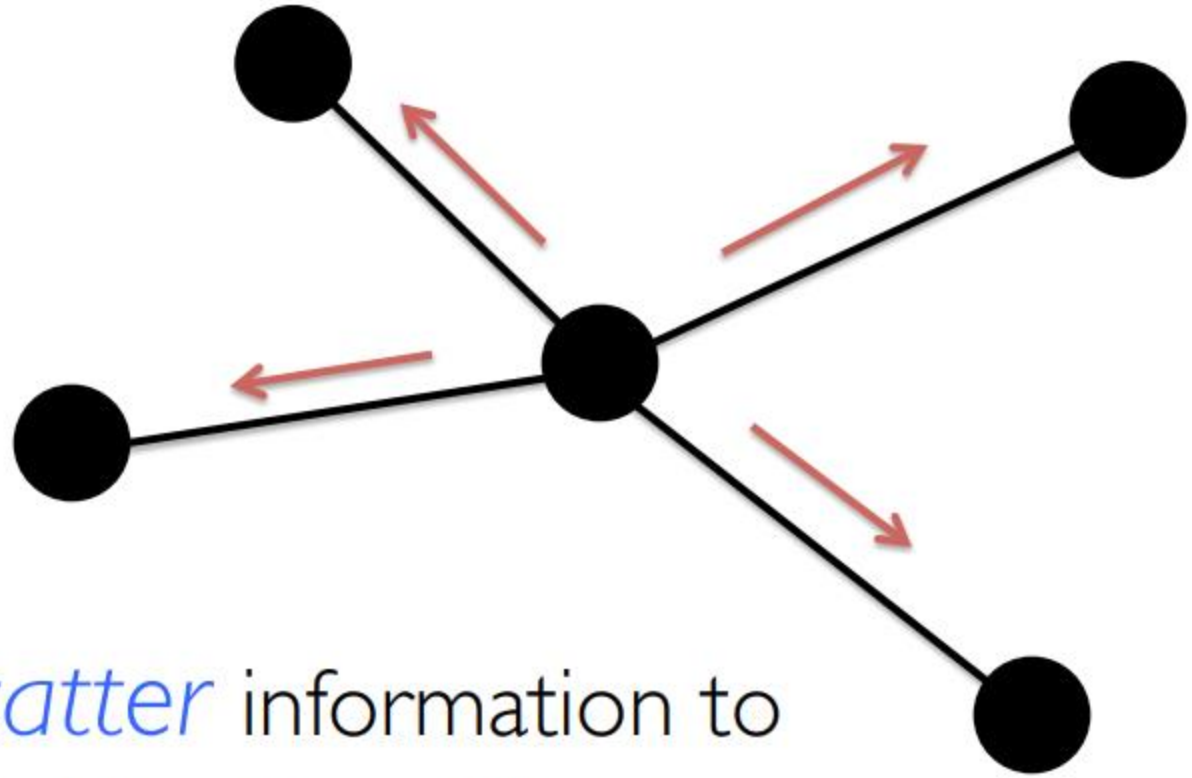
Gonzalez et al. [OSDI'12]



Apply an update the vertex property

Graph-Parallel Pattern

Gonzalez et al. [OSDI'12]



Scatter information to neighboring vertices

Pregel API

```
def pregel[A]
    (initialMsg: A,
     maxIter: Int = Int.MaxValue,
     activeDir: EdgeDirection = EdgeDirection.Out)
    (vprog: (VertexId, VD, A) => VD,
     sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
     mergeMsg: (A, A) => A)
```

- Első lista – konfiguráció
  - inicializáló üzenet
  - max iterációszám
  - aktív él irány – melyik irányba menjenek az üzenetek
    - következő iterációban csak azokkal az élekkel fog foglalkozni, amelyek kaptak üzenetet
    - alap beállítás mindkét irány

Pregel API

```
def pregel[A]
    (initialMsg: A,
     maxIter: Int = Int.MaxValue,
     activeDir: EdgeDirection = EdgeDirection.Out)
    (vprog: (VertexId, VD, A) => VD,
     sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
     mergeMsg: (A, A) => A)
```

- Második lista
  - vertex program
  - üzenet küldése
  - üzenet aggregálása

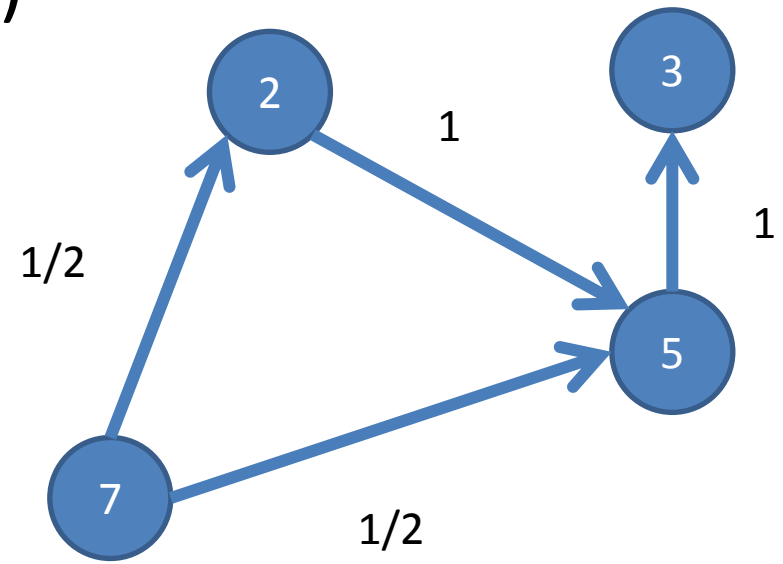
Legrövidebb út keresése Pregel

```scala
import org.apache.spark.graphx.{Graph, VertexId}
import org.apache.spark.graphx.util.GraphGenerators

// A graph with edge attributes containing distances
val graph: Graph[Long, Double] =
  GraphGenerators.logNormalGraph(sc, numVertices = 100).mapEdges(e => e.attr.toDouble)
val sourceId: VertexId = 42 // The ultimate source
// Initialize the graph such that all vertices except the root have distance infinity.
val initialGraph = graph.mapVertices((id, _) =>
    if (id == sourceId) 0.0 else Double.PositiveInfinity)
val sssp = initialGraph.pregel(Double.PositiveInfinity)(
  (id, dist, newDist) => math.min(dist, newDist), // Vertex Program
  triplet => {  // Send Message
    if (triplet.srcAttr + triplet.attr < triplet.dstAttr) {
      Iterator((triplet.dstId, triplet.srcAttr + triplet.attr))
    } else {
      Iterator.empty
    }
  },
  (a, b) => math.min(a, b) // Merge Message
)
println(sssp.vertices.collect.mkString("\n"))
```

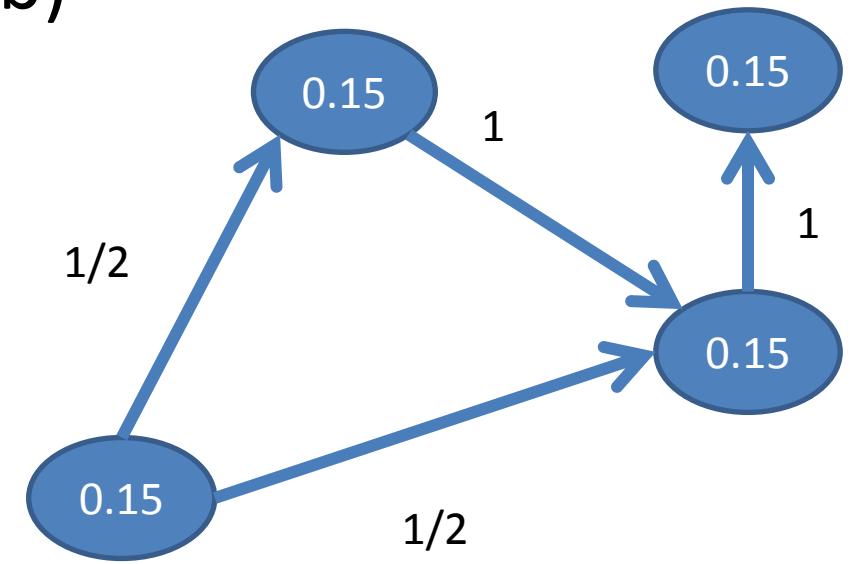
Pagerank

- vprog: 0.15 + 0.85 * msg
- sendMsg: n1 send n1.p*edge.attr
- aggrMsg: sum(a,b)



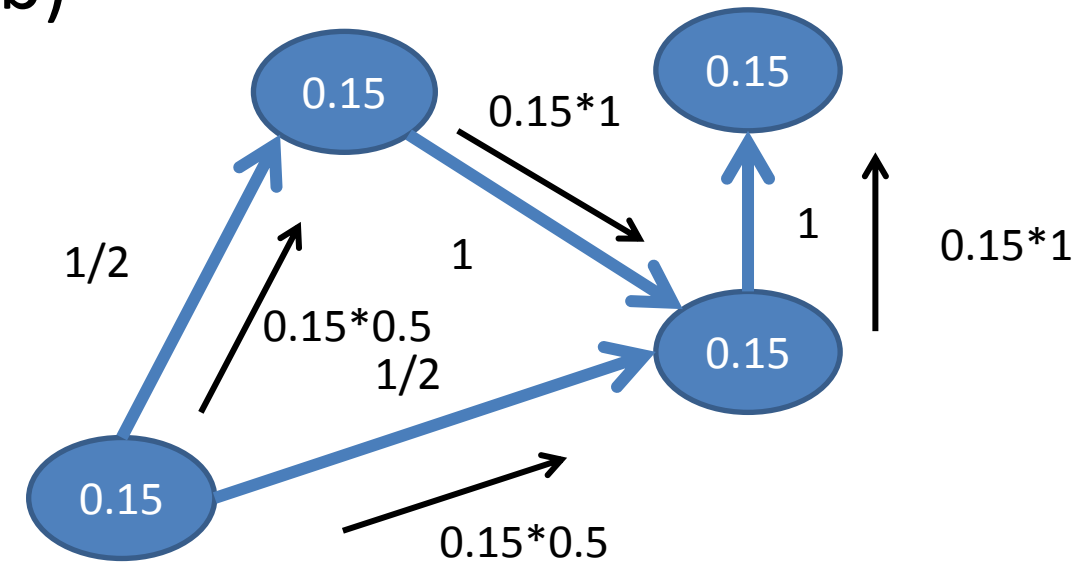
Pagerank

- vprog: 0.15 + 0.85 * msg
- sendMsg: n1 send n1.p*edge.attr
- aggrMsg: sum(a,b)



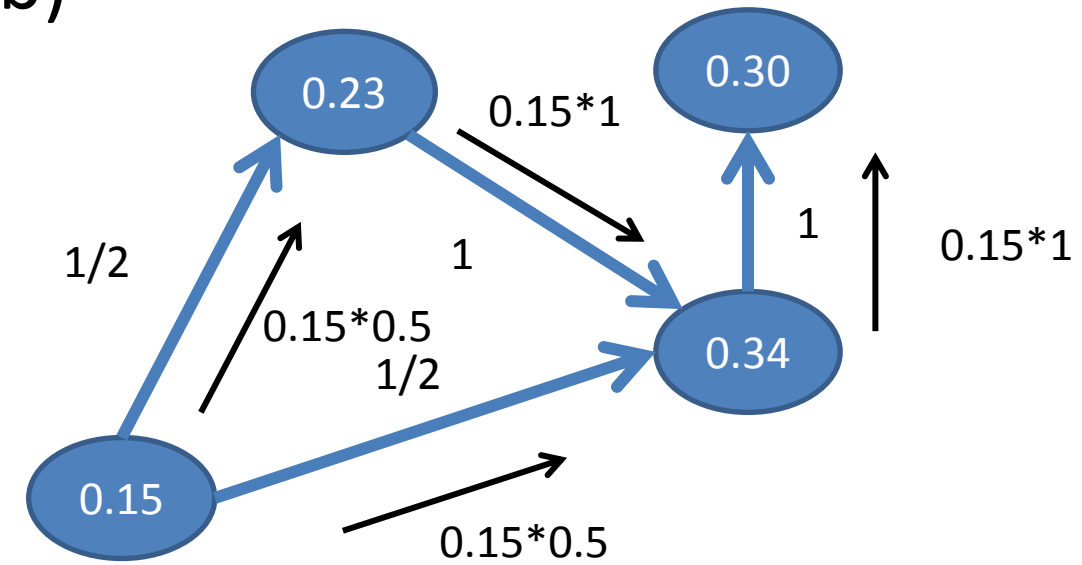
Pagerank

- vprog: 0.15 + 0.85 * msg
- sendMsg: n1 send n1.p*edge.attr
- aggrMsg: sum(a,b)



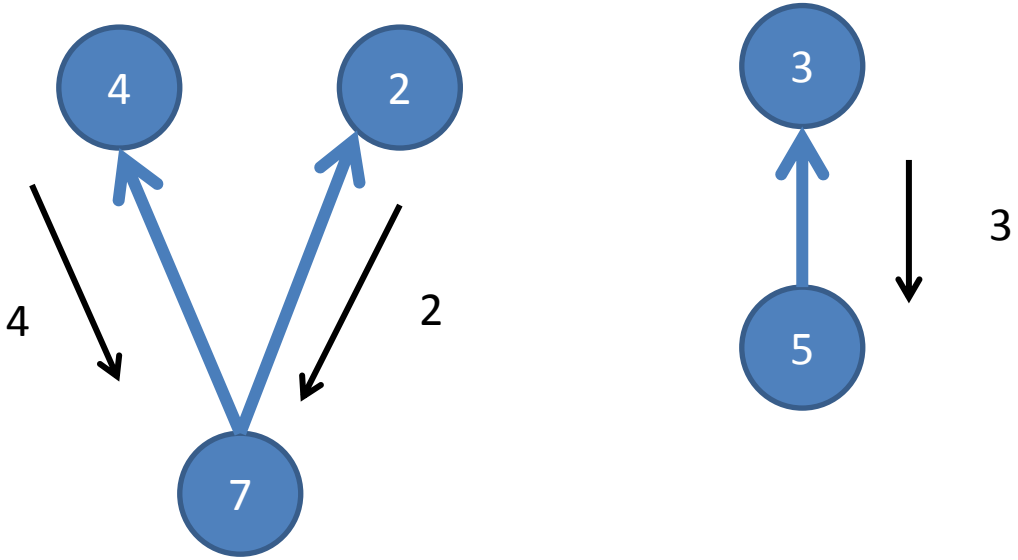
Pagerank

- vprog: 0.15 + 0.85 * msg
- sendMsg: n1 send n1.p*edge.attr
- aggrMsg: sum(a,b)



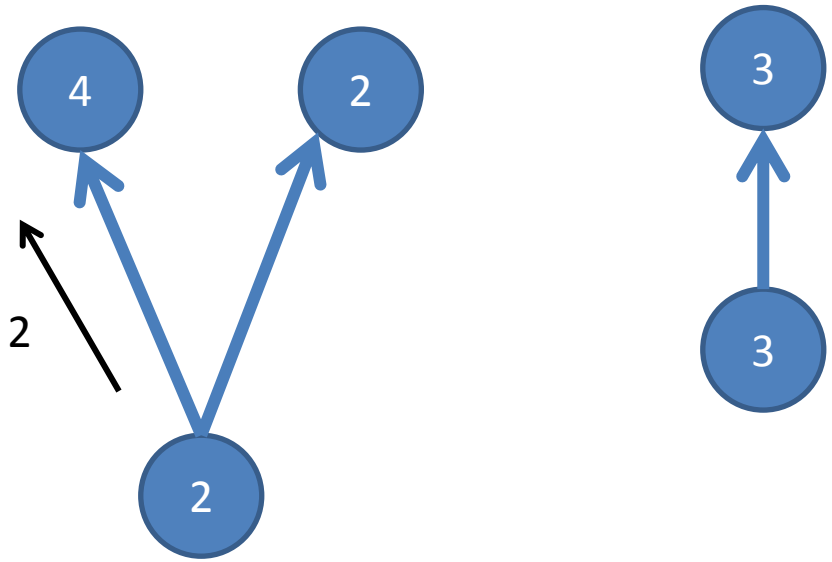
Connected Components

- vprog: math.min(msg,id)
- sendMsg: if n1.id<n2.id: n1 send n1.id
- aggrMsg: math.min(a,b)



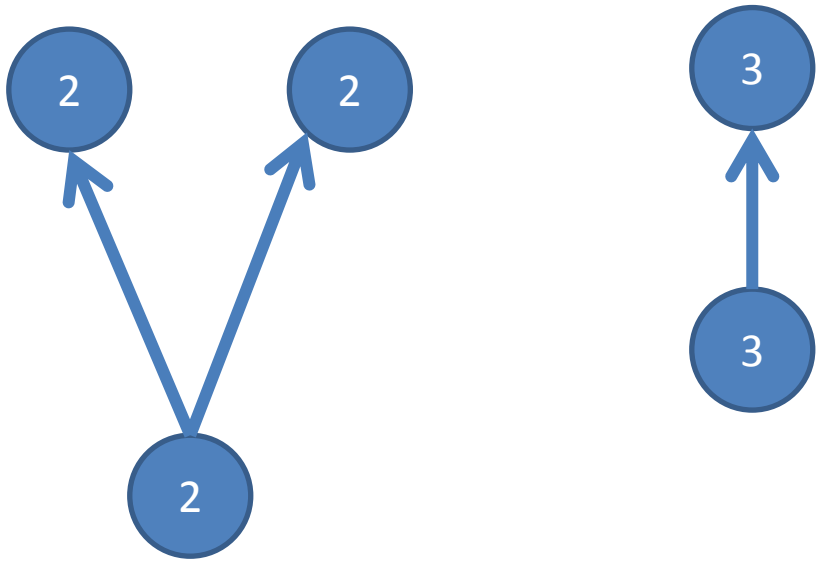
Connected Components

- vprog: math.min(msg,id)
- sendMsg: if n1.id<n2.id: n1 send n1.id
- aggrMsg: math.min(a,b)



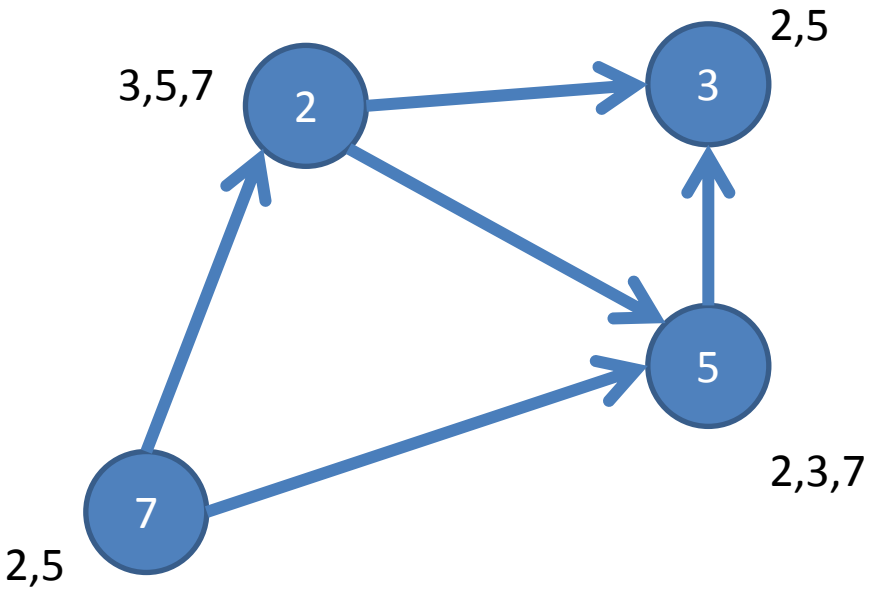
Connected Components

- vprog: math.min(msg,id)
- sendMsg: if n1.id<n2.id: n1 send n1.id
- aggrMsg: math.min(a,b)



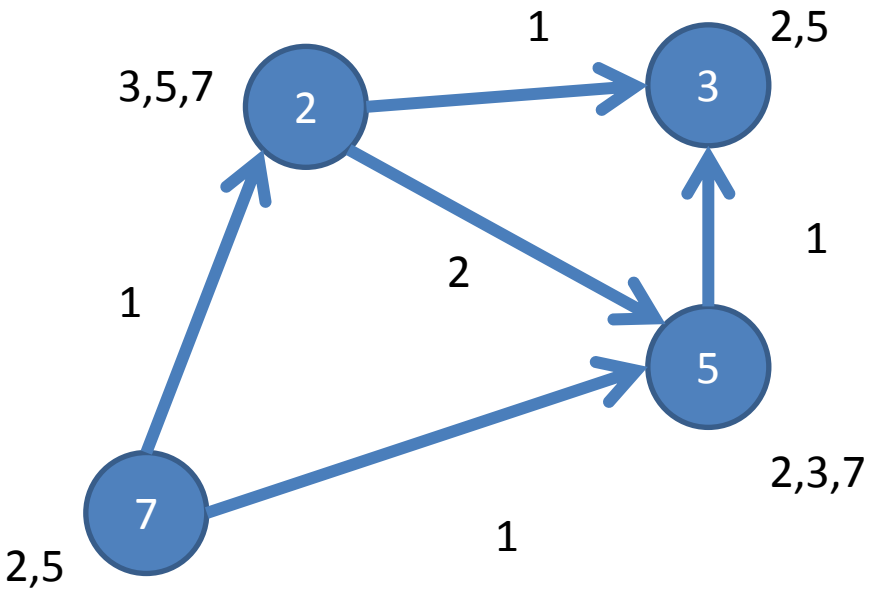
Triangle Count

- kiszámoljuk a szomszédok halmazát



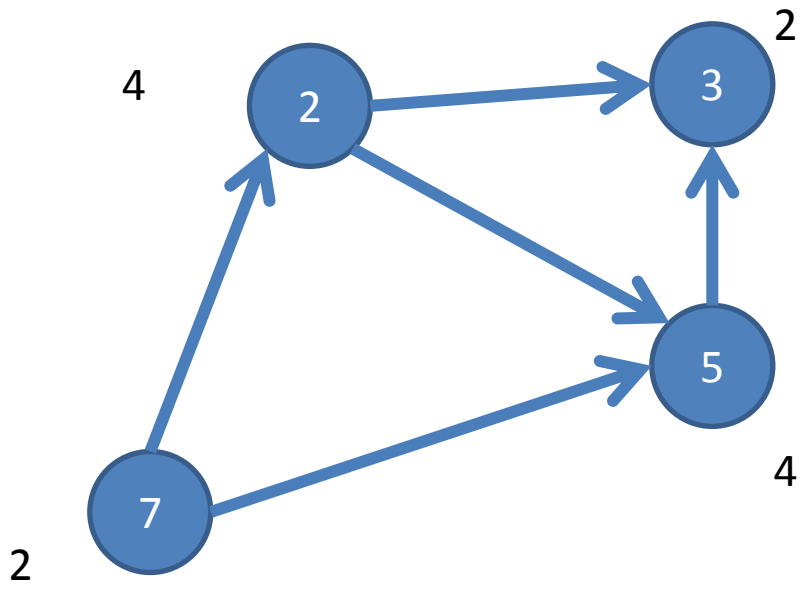
Triangle Count

- minden élen megnézzük hány egyező szomszéd van



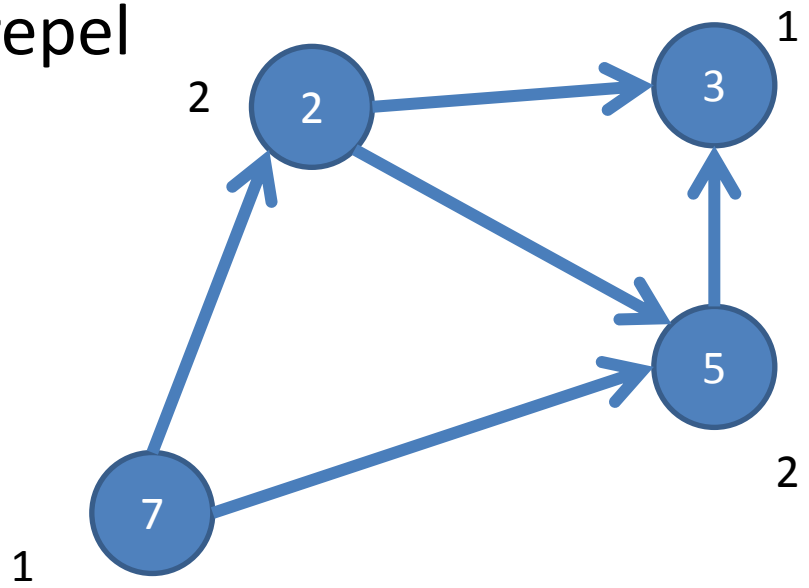
Triangle Count

- Minden élen mindkét irányba elküldjük az üzenetet, majd összegezzük



Triangle Count

- Mivel minden háromszög 2szer szerepel
  leosztjuk 2-vel
- Az eredmény megmondja, hogy egy node hány háromszögben szerepel



Create a Property Graph

1. Import required classes
2. Create vertex RDD
3. Create edge RDD
4. Create graph

Create a Property Graph

Import required classes

```
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
```

Create a Property Graph: Data Set

Vertices:

Vertex ID	Property (V)
Id (Long)	Name (String)

Edges:

Source ID	Dest ID	Property (E)
Id	Id	Distance (Integer)

Create a Property Graph

Id	Property
1	SFO
2	ORD
3	DFW

Create vertex RDD

```
// create  vertices RDD with ID and Name
val vertices=Array((1L, ("SFO")),
(2L,("ORD")),(3L,("DFW")))
val vRDD= sc.parallelize(vertices)
vRDD.take(1)
// Array((1,SFO))
```

Create a Property Graph

SrcId	DestId	Property
1	2	1800
2	3	800
3	1	1400

Create edge RDD

```
// create  routes  RDD with srcid, destid , distance
val edges = Array(Edge(1L,2L,1800), Edge(2L,3L,800),Edge(3L,1L,1400))
val eRDD= sc.parallelize(edges)
eRDD.take(2)
// Array(Edge(1,2,1800), Edge(2,3,800))
```

Create a Property Graph

Create graph

```
// define default vertex nowhere
val nowhere = "nowhere"

//build initial graph
val graph = Graph(vertices, edges, nowhere)

graph.vertices.take(3).foreach(print)
//  (2,ORD)(1,SFO)(3,DFW)

graph.edges.take(3).foreach(print)
// Edge(1,2,1800) Edge(2,3,800) Edge(3,1,1400)
```

Graph Operators

To answer questions such as:

- How many airports are there?
- How many flight routes are there?
- What are the longest distance routes?
- Which airport has the most incoming flights?
- What are the top 10 flights?

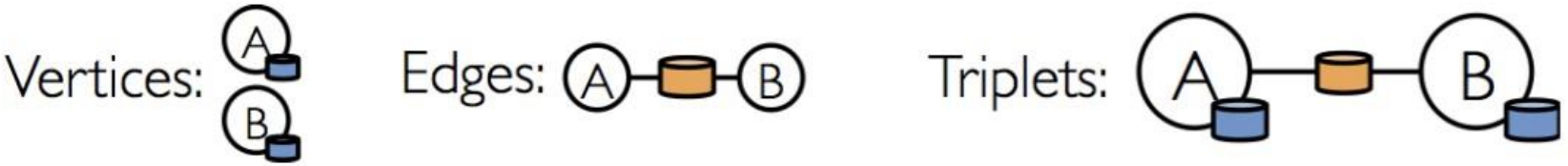
Graph Operators

Graph Operators

```
// How many airports?
val numairports = graph.numVertices
// Long = 3
// How many routes?
val numroutes = graph.numEdges
// Long = 3

// routes > 1000 miles distance?
graph.edges.filter {
    case ( Edge(org_id, dest_id,distance))=>
    distance > 1000
    }.take(3)
// Array(Edge(1,2,1800), Edge(3,1,1400)
```

Triplets




```
// Triplets add source and destination
properties to Edges
graph.triplets.take(3).foreach(println)
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)
```

Triplets What are the longest routes ?

```
((1,SFO),(2,ORD),1800)
((2,ORD),(3,DFW),800)
((3,DFW),(1,SFO),1400)
// print out longest routes
graph.triplets.sortBy(_.attr,
ascending=false)
.map(triplet =>"Distance" +
triplet.attr.toString + “from" +
triplet.srcAttr + “to" +
triplet.dstAttr)
.collect.foreach(println)
Distance 1800 from SFO to ORD
Distance 1400 from DFW to SFO
Distance 800 from ORD to DFW
```

Graph Operators

Which airport has the most incoming flights? (real dataset)

```
// Define a function to compute the highest
degree vertex
def max(a:(VertexId,Int),b:(VertexId, Int))
:(VertexId, Int) =
{ if (a._2 > b._2) a else b }

// Which Airport has the most incoming
flights?
val maxInDegree:(VertexId, Int)=
graph.inDegrees.reduce(max)
// (10397,152)  ATL
```

Graph Operators

Which 3 airports have the most incoming flights? (real dataset)

```
// get top 3
val maxIncoming = graph.inDegrees.collect
  .sortWith(_._2 > _._2)
  .map(x => (airportMap(x._1), x._2)).take(3)

maxIncoming.foreach(println)
(ATL,152)
(ORD,145)
(DFW,143)
```

Pregel Operator: Example

Use Pregel to find the cheapest airfare:

```
// starting vertex
val sourceId: VertexId = 13024

// a graph with edges containing airfare cost calculation
val gg = graph.mapEdges(e => 50.toDouble + e.attr.toDouble/20)

// initialize graph, all vertices except source have distance
infinity
val initialGraph = gg.mapVertices((id, _) =>
   if (id == sourceId) 0.0 else    Double.PositiveInfinity
```

Pregel Operator: Example

Use Pregel to find the cheapest airfare:

```
// call pregel on graph
val sssp = initialGraph.pregel(Double.PositiveInfinity)(
 // Vertex Program
 (id, distCost, newDistCost) =>
 math.min(distCost,newDistCost),
 triplet => {
  // Send Message
  if (triplet.srcAttr + triplet.attr
   <triplet.dstAttr) {
     Iterator((triplet.dstId, triplet.srcAttr +
   triplet.attr))
  } else {
   Iterator.empty}},
 // Merge Message
 (a,b) => math.min(a,b)
)
```

Pregel Operator: Example

Use Pregel to find the cheapest airfare:

```
// routes , lowest flight cost
println(sssp.edges.take(4).mkString("\n"))
Edge(10135,10397,84.6)  Edge(10135,13930,82.7)
Edge(10140,10397,113.45)  Edge(10140,10821,133.5)
```

Köszönöm a Figyelmet!