INDIAN INSTITUTE OF TECHNOLOGY PATNA

NPTEL

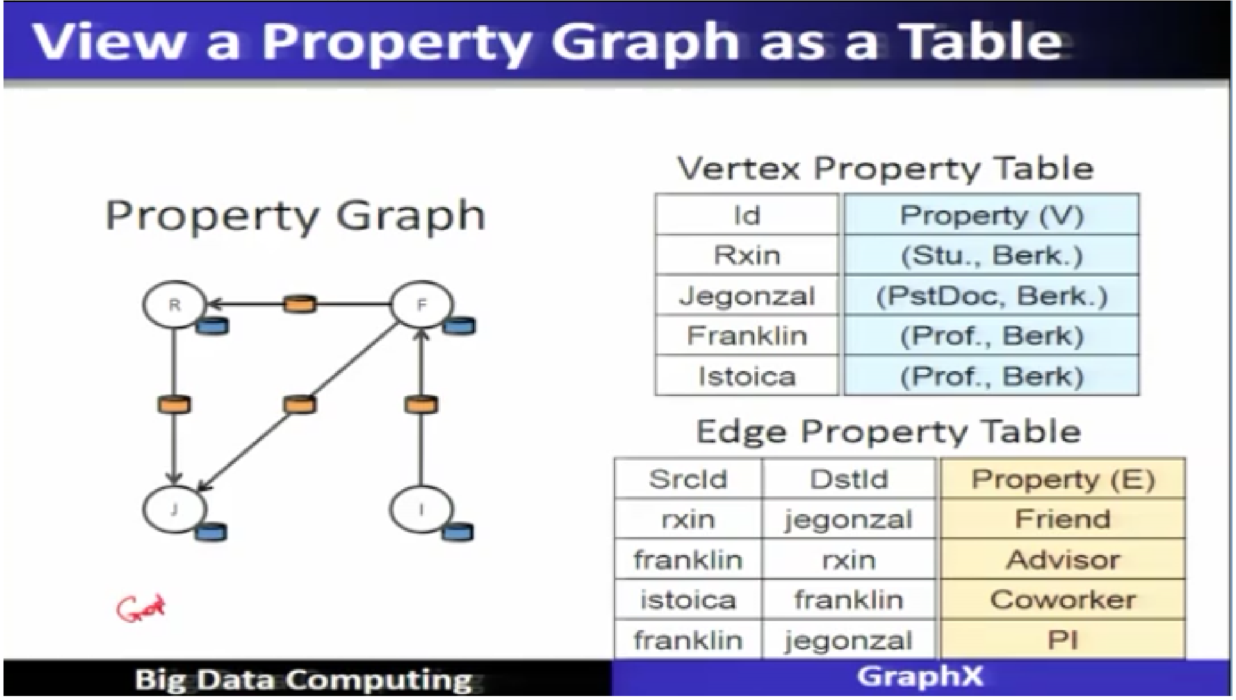
NATIONAL PROGRAMME ON TECHNOLOGY ENHANCED LEARNING

COURSE TITLE
BIG DATA COMPUTING

LECTURE-33
SPARK GRAPH-X AND GRAPH ANALYTICS
PART-II

BY
DR. RAJIV MISRA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PATNA

(Refer Slide Time: 00:16)



View a Property Graph as a Table

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

Big Data Computing GraphX

The graph in GraphX is represented as the property graph, so graph in a GraphX is called the property graph. Property graph has information such that every node has the vertex ID, and also it has the properties. This particular vertex has two informations, one is called vertex ID, the other is called the property and both this information are stored in a table which is called vertex property table.

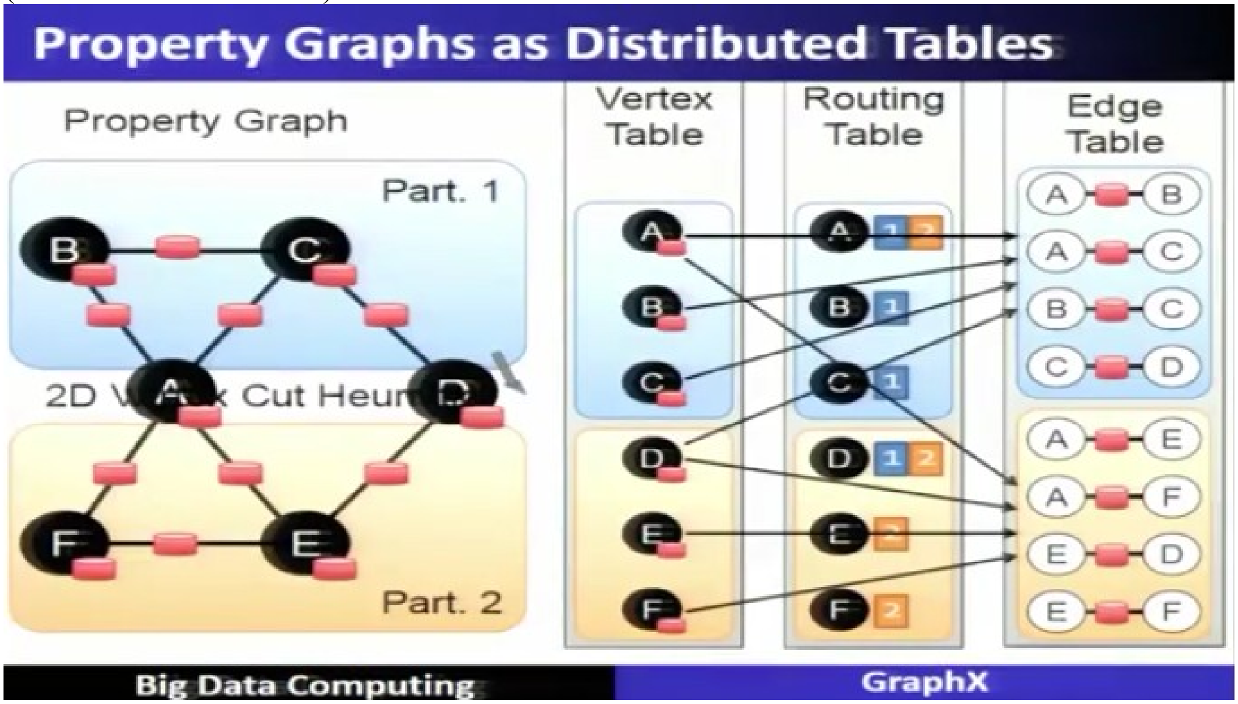
Now similarly the edges have, so the edges have the ID's of both the ends, so for example in the directed edge it will be original, is the source ID and the destination ID, so this will be the edge ID's and then every edge will also contain the property with associated with that every edge, so this information is stored in a table which is called edge property table, so we can see here that for example the vertex ID I and vertex ID F, so vertex ID I and vertex ID F, so vertex ID I will have this particular property that it is from, it is a professor and it is from Berkley. Similarly the vertex ID which is F, the name is Franklin or the ID is the F, so it is also a professor, but it is also working in the Berkley.

Now if you see this particular edge between I and F, so the edge between I and F is represented over here and it represents the property which is called coworker, so we have seen that the vertices have the properties, vertices have ID's and also a property and we can represent the vertices in a form of a table which is called vertex property table.

Similarly the edges are represented by source ID and definition ID and every edge also has a property and this edge property table contains the information about all the edges, so vertex property table and edge property table will form the representation of a property graph, so graph can be represented in a form of, a property graph can be represented in a form of the table property graph can be represented in a form of a table which are called as a vertex property table, and the edge property table, so the graph again I'm repeating this aspect which is very important that graph in GraphX is called a property graph, and the property graph is represented in a form of a table which is nothing but vertex property table and edge property table.

So graph can be viewed, so as graph is represented as the property graph and it can be changed or it can be viewed as the table also, that is having the property table, vertex property table and edge property table.

(Refer Slide Time: 04:34)



Property Graphs as Distributed Tables

Big Data Computing

GraphX

So property graph is represented as the distributed table, so we can see here that if a graph which is called a property graph, if it is too big it cannot fit in 2 1 machine, then we can cut across these vertex, and once it is cut across this vertex the first plot can be stored in one machine, the second part can be stored on the other machine.

So for example the corresponding vertex table and the edge table will be, can be stored in two different machines, so this is machine number 1, this is machine number 2, so A, B and C, so the upper part will be stored in, the first three node of the table will be stored in one machine, the another three nodes will be stored in other machine, similarly the edges, so all the edges which are in part 1 will be stored in the first edge table, similarly the remaining edges are stored in another table.

These vertices and edges, how they are communicating each other is communicating through each other via routing table, so routing table is also another table which is added in between, so there are three different tables which you will see here are to represent the graph as a form of a table, and this particular property graph will be represented in this way as the distributed tables, so vertex table, routing table and the edge table.

By routing table we mean that so the, for example the vertex A, vertex A is a being referred by the edges in both the parts that is, it will be referred by A, it will be referred by B, it will be referred by F, so that means the vertex it will be referred in both the partitions, therefore it is represented as the routing table will contain the partitions where this vertex A will be referred.

And for example the node or a vertex E will be only referred in the second partition, hence the routing table for E will contain only 2, the second partition, this way the vertex table and edge table can be distributed and with the help of routing table together will be used as the distributed tables.
(Refer Slide Time: 07:25)

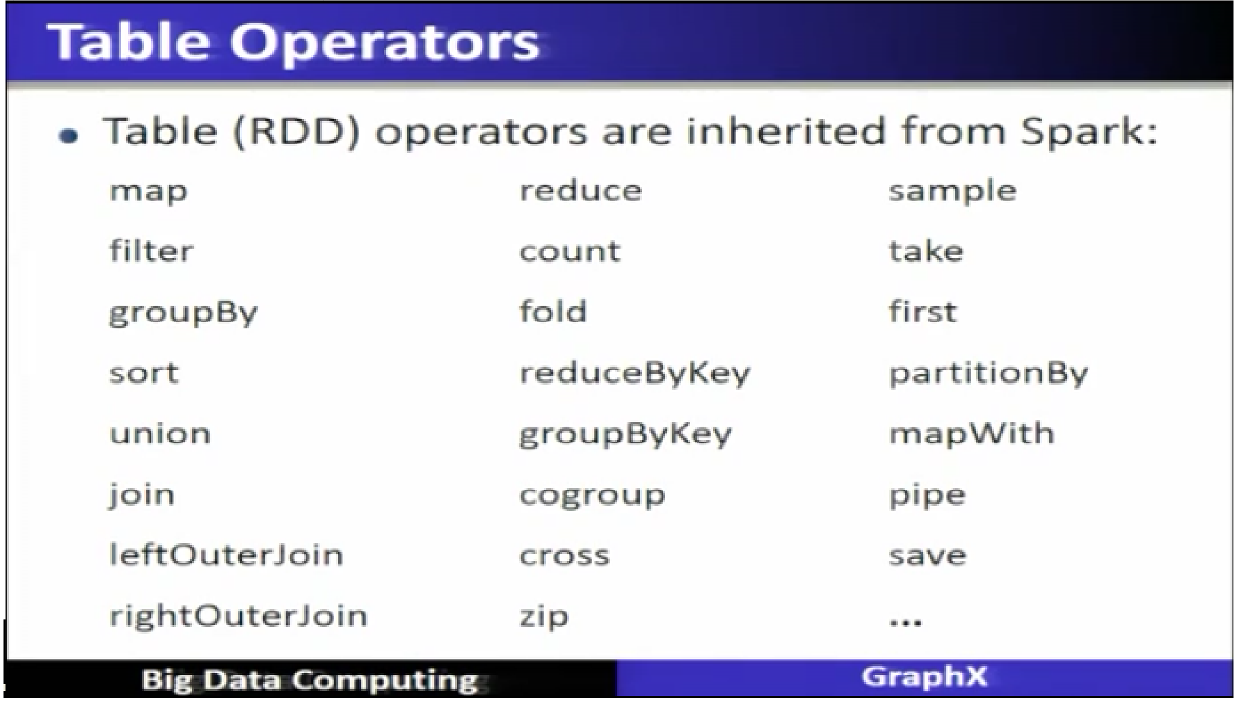


Table Operators

- Table (RDD) operators are inherited from Spark:

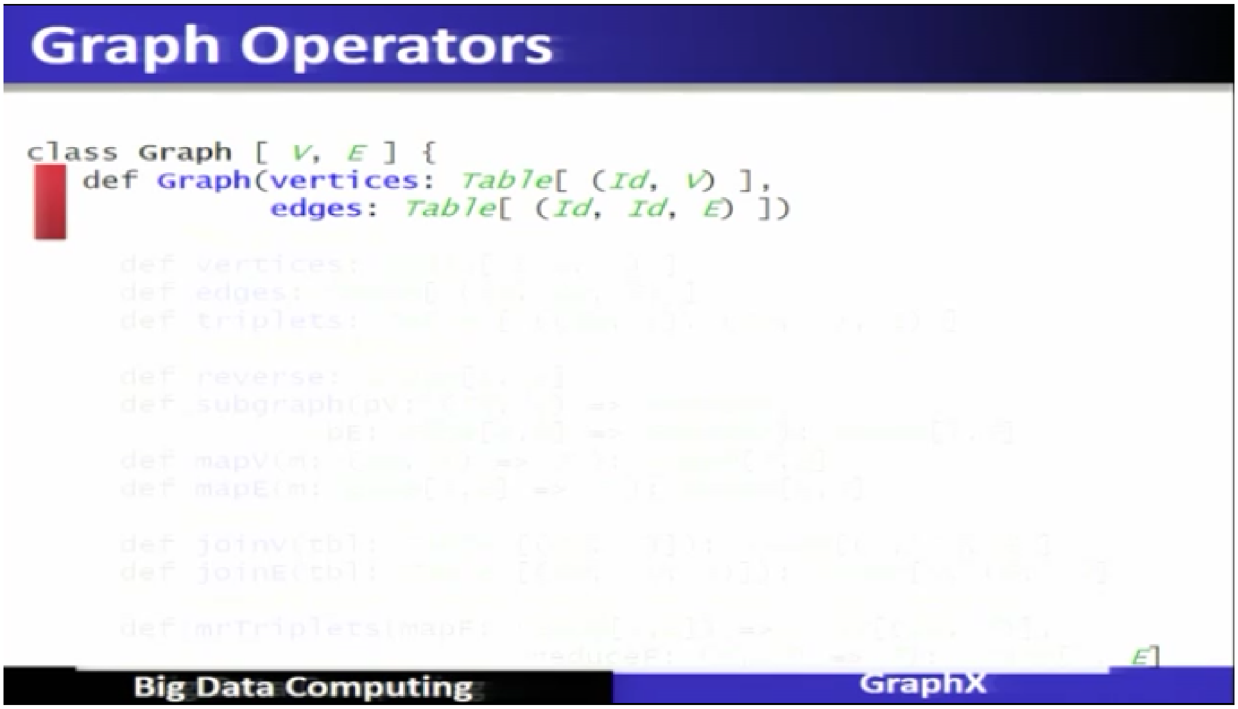
map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin

reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip

sample
take
first
partitionBy
mapWith
pipe
save
...

Big Data Computing GraphX

There are different table operators which are inherited from the spark, that we can see here,
(Refer Slide Time: 07:35)



Graph Operators

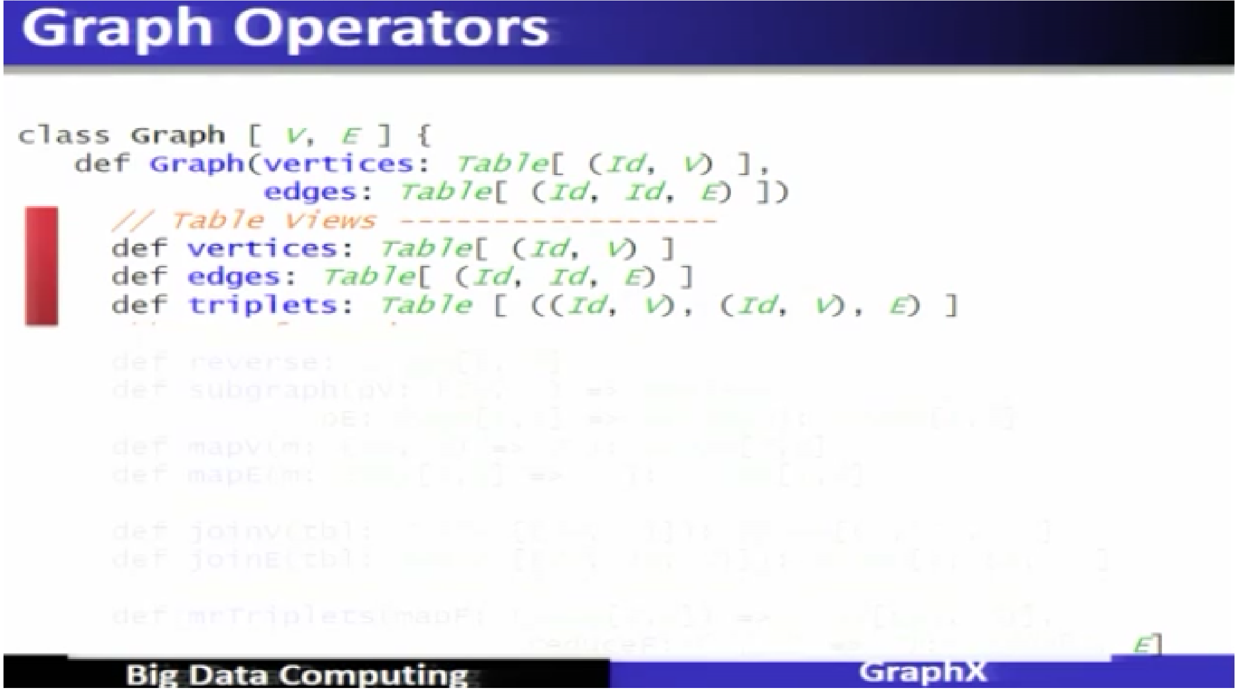
```
class Graph [ V, E ] {
  def Graph(vertices: Table[ (Id, V) ],
            edges: Table[ (Id, Id, E) ])
```

Big Data Computing

GraphX

let us see the graph operators which are supported here in GraphX, so graph is represented as a class graph vertex and edges where the graph vertices is nothing but a having a table as off ID's and vertices,

(Refer Slide Time: 07:54)



Graph Operators

```
class Graph [ V, E ] {
  def Graph(vertices: Table[ (Id, V) ],
            edges: Table[ (Id, Id, E) ])
    // Table Views ------------------
    def vertices: Table[ (Id, V) ]
    def edges: Table[ (Id, Id, E) ]
    def triplets: Table [ ((Id, V), (Id, V), E) ]
```

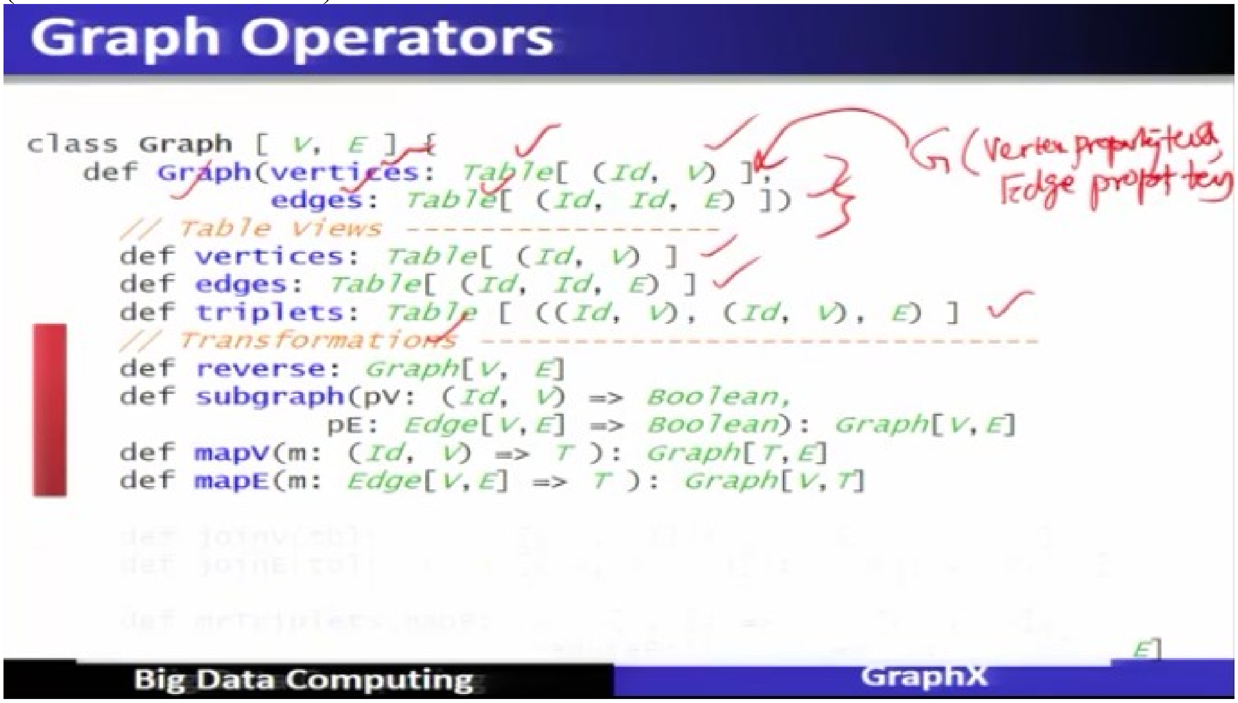
Big Data Computing

GraphX

and edges are also having the table of source vertices and definition vertices and the edges, properties, so this is the representation of a graph, so graph is represented as the vertices and the edges, whereas the vertices are represented as the vertex property table, and edge is represented as the edge property table, and these are represented together as the graph, so graph here is represented as the vertex property table, edge property table, and which is shown in this particular representation of the graph.

Now having defined this particular graph in this manner, so you can also see the graph as table view, so the vertices is represented by this table that is called a vertex property table, edge is represented by the edge property table, and also there will be a triplet view of a particular graph, that means so triplet view of a graph is also triplet view table can be represented.

(Refer Slide Time: 09:24)



Graph Operators

```
class Graph [ V, E ] {
   def Graph(vertices: Table[ (Id, V) ],
             edges: Table[ (Id, Id, E) ])
     // Table Views -----------------
     def vertices: Table[ (Id, V) ]
     def edges: Table[ (Id, Id, E) ]
     def triplets: Table [ ((Id, V), (Id, V), E) ]
     // Transformations ------------------------------
     def reverse: Graph[V, E]
     def subgraph(pV: (Id, V) => Boolean,
                  pE: Edge[V,E] => Boolean): Graph[V,E]
     def mapV(m: (Id, V) => T ): Graph[T,E]
     def mapE(m: Edge[V,E] => T ): Graph[V,T]
```

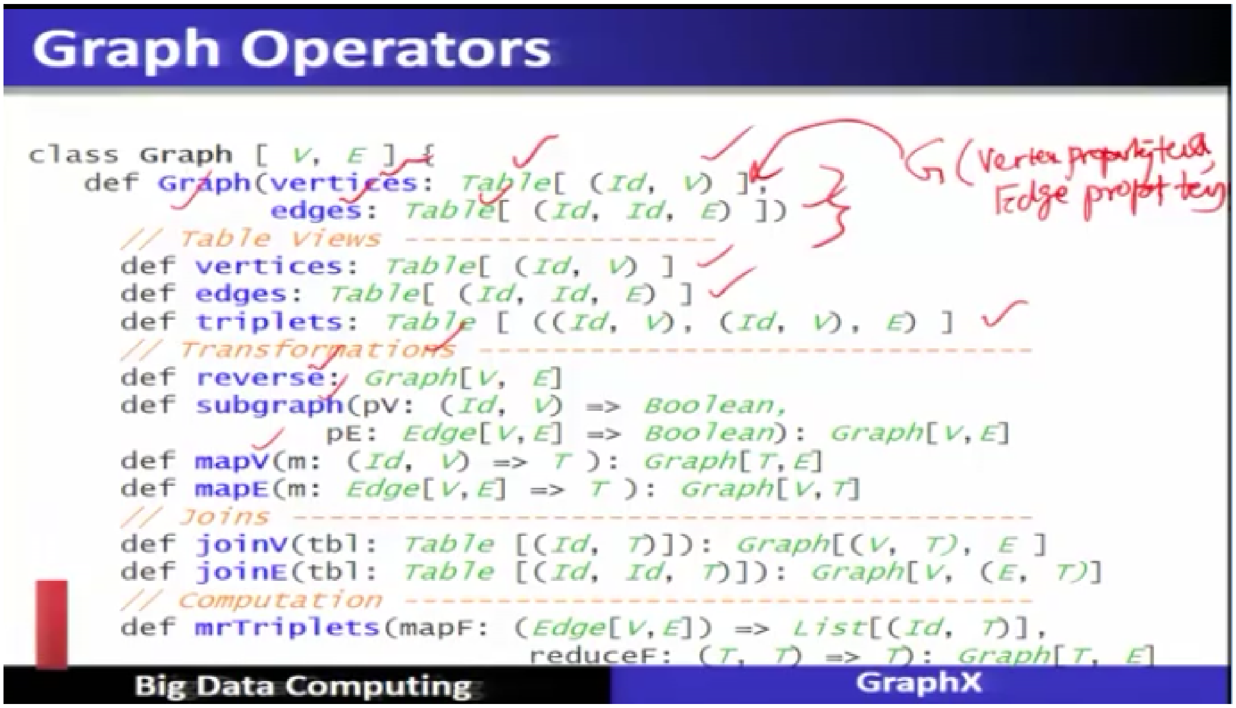
G (Vertex propertytable,
Edge propt tey

Big Data Computing

GraphX

Where various transformations in the graph which are supported here in the GraphX is called the reverse of, then subgraph, finding the subgraph and then taking the map of this particular graph, the joint operations are also performed on the graph,

(Refer Slide Time: 09:42)



Graph Operators

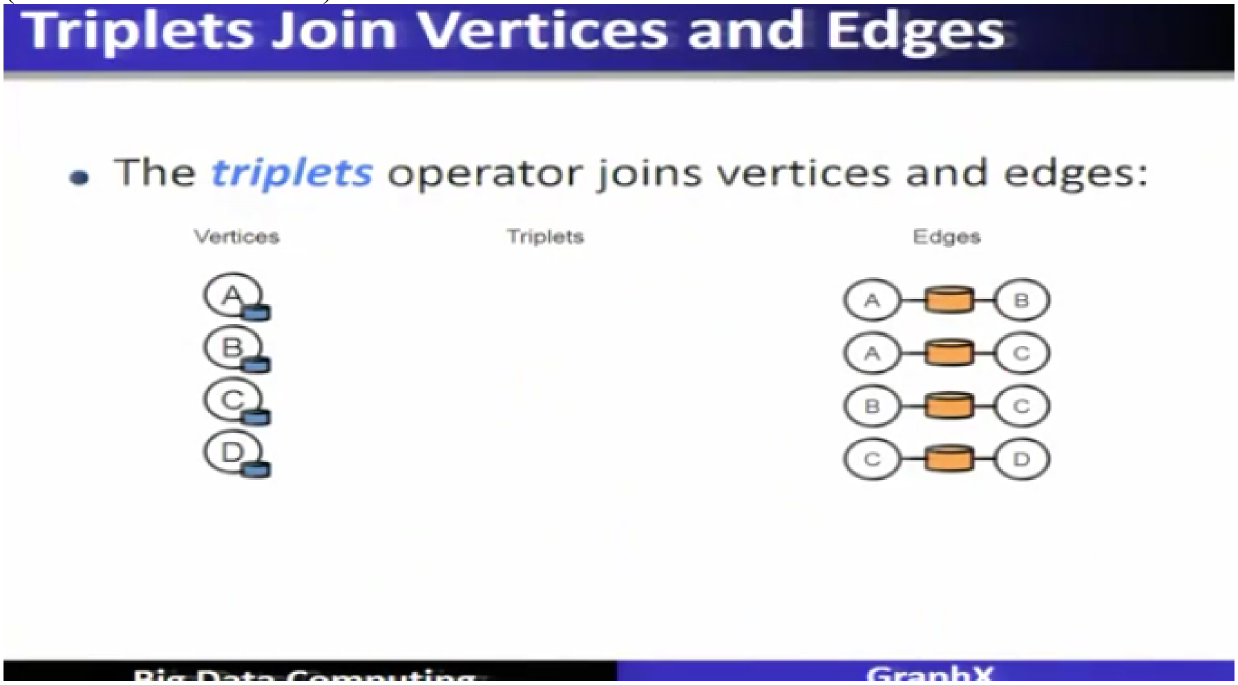
```
class Graph [ V, E ] {
   def Graph(vertices: Table[ (Id, V) ],
             edges: Table[ (Id, Id, E) ])
     // Table Views -----------------
     def vertices: Table[ (Id, V) ]
     def edges: Table[ (Id, Id, E) ]
     def triplets: Table [ ((Id, V), (Id, V), E) ]
     // Transformations ------------------------------
     def reverse: Graph[V, E]
     def subgraph(pV: (Id, V) => Boolean,
                  pE: Edge[V,E] => Boolean): Graph[V,E]
     def mapV(m: (Id, V) => T ): Graph[T,E]
     def mapE(m: Edge[V,E] => T ): Graph[V,T]
     // Joins ----------------------------------------
     def joinV(tbl: Table [(Id, T)]): Graph[(V, T), E ]
     def joinE(tbl: Table [(Id, Id, T)]): Graph[V, (E, T)]
     // Computation ----------------------------------
     def mrTriplets(mapF: (Edge[V,E]) => List[(Id, T)],
                              reduceF: (T, T) => T): Graph[T, E]
```

G (Vertex property table,
Edge propert tey

Big Data Computing | GraphX

and the computations are performed in the graph.

(Refer Slide Time: 09:47)



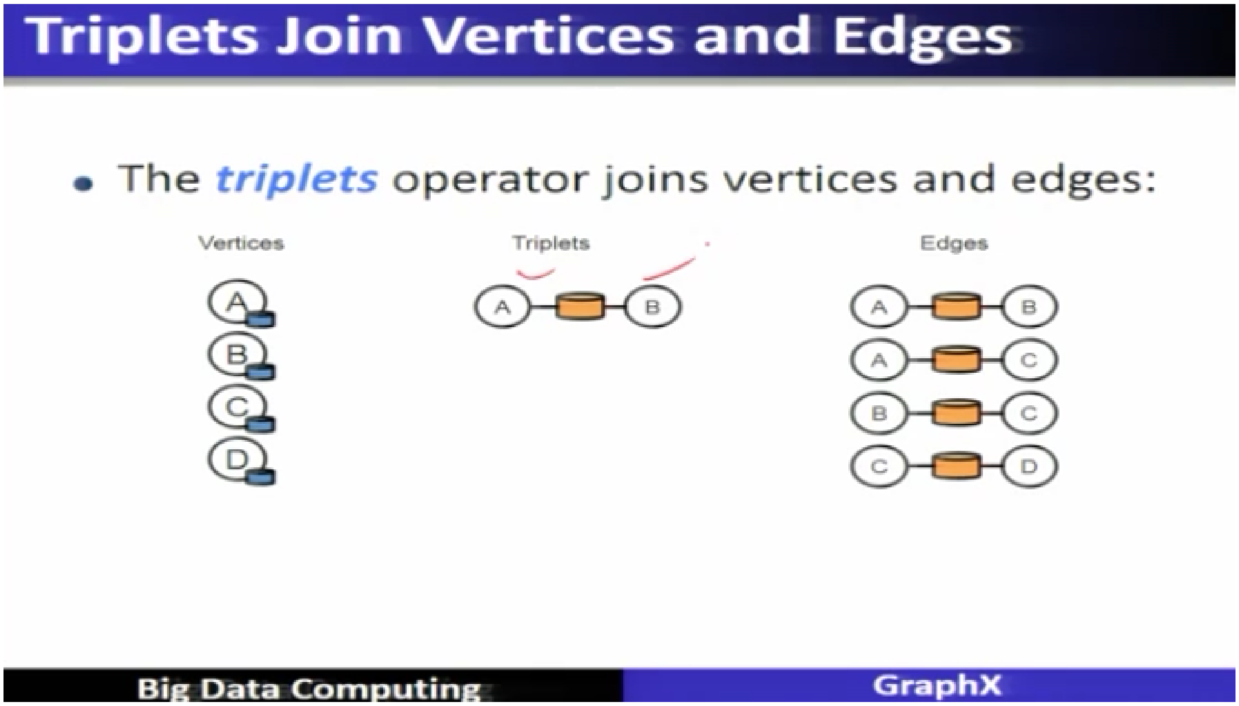
Triplets Join Vertices and Edges

- The *triplets* operator joins vertices and edges:

Big Data Computing | GraphX

Now the triplet join vertices and edges, so triplet operator joins the vertices and the edges, so if this is the edge between A and B,

(Refer Slide Time: 10:03)



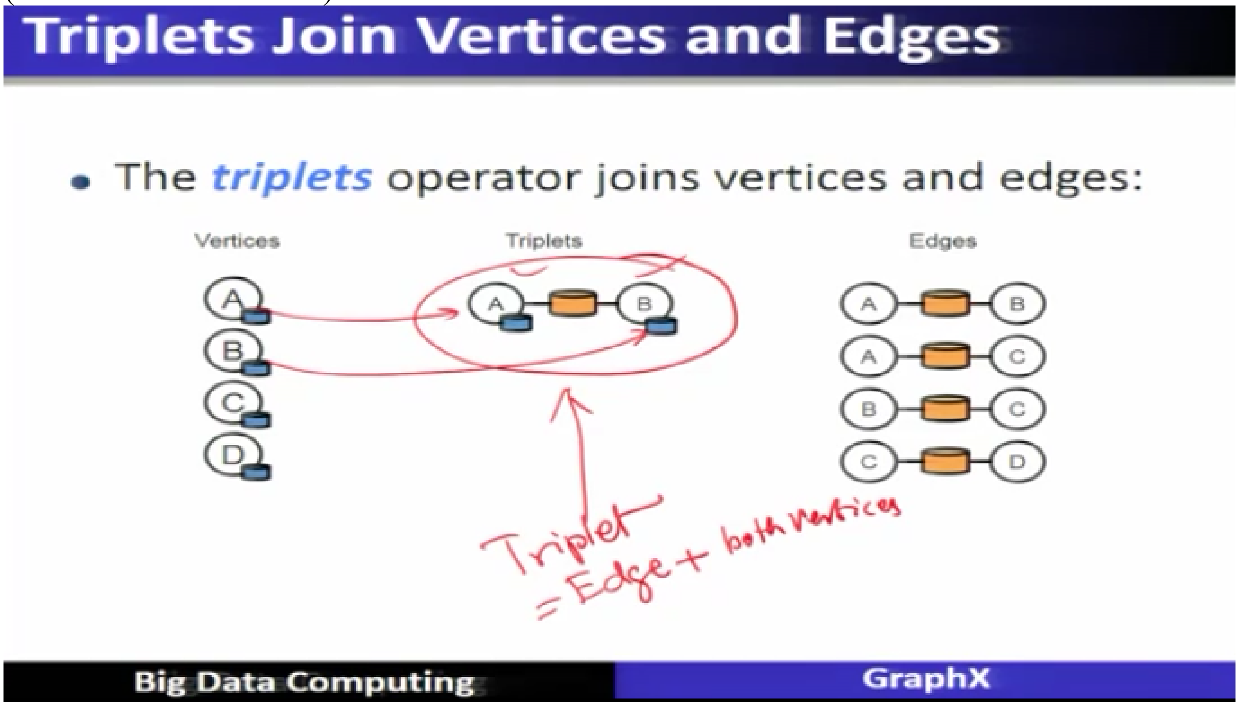
Triplets Join Vertices and Edges

- The *triplets* operator joins vertices and edges:

Big Data Computing GraphX

so it will also combine with both the vertices A and B along with this edge is called a triplet, so triplet means that, is that edge + both operates vertices is called the triplet.

So we can see here that this edge will have the property A also, property of vertex A,
(Refer Slide Time: 10:37)



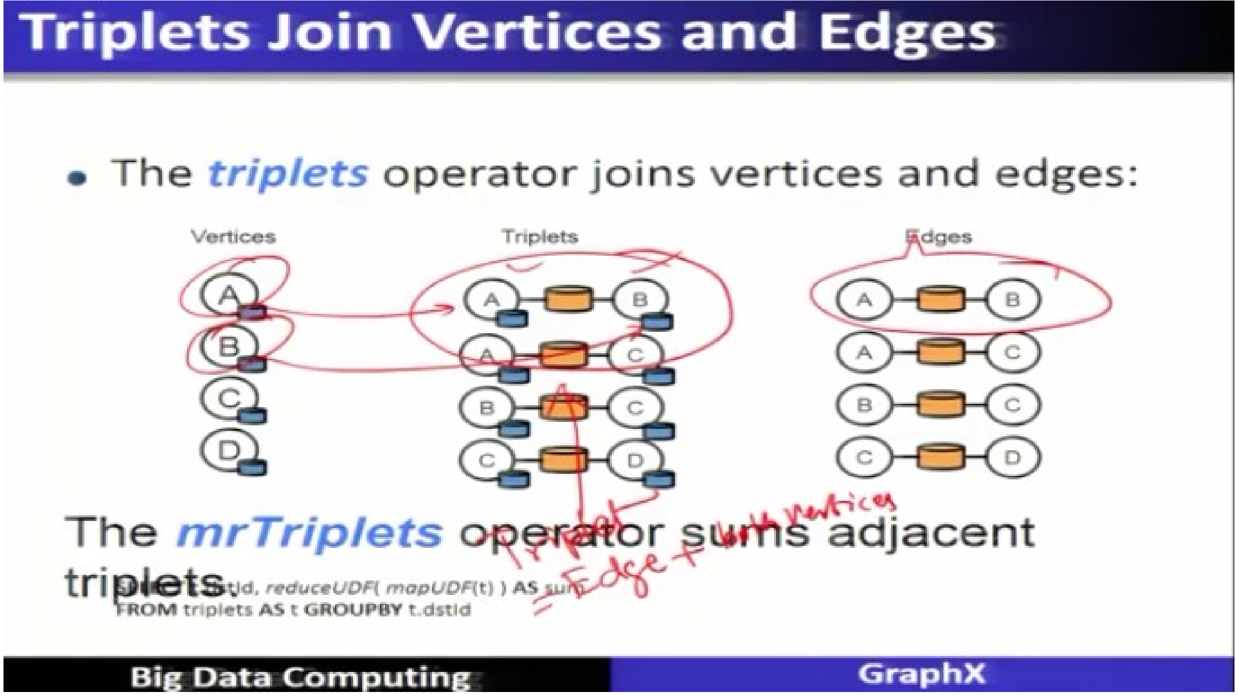
Triplets Join Vertices and Edges

- The *triplets* operator joins vertices and edges:

Big Data Computing GraphX

and this edge also has the property of vertex B as well as it has the property of the edge A, B. Together all three things is called the triplet. Triplet is supported in the GraphX and is very much useful in building the graph algorithms.

(Refer Slide Time: 10:55)



Triplets Join Vertices and Edges

- The *triplets* operator joins vertices and edges:

The *mrTriplets* operator sums adjacent triplets.

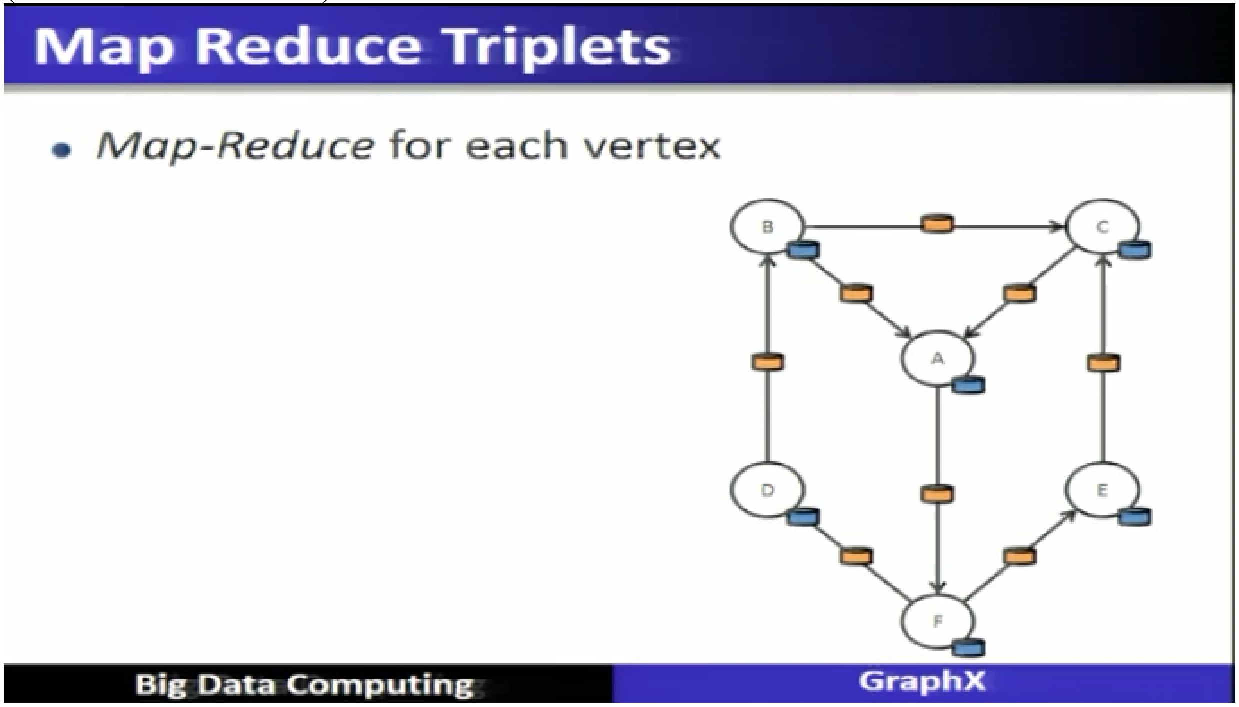
```
tId, reduceUDF( mapUDF(t) ) AS sum
FROM triplets AS t GROUPBY t.dstId
```

Triplet = Edge + both vertices

Big Data Computing GraphX

The map reduce triplet operator sums the adjacent triplets and which will be used in various graph algorithms,

(Refer Slide Time: 11:02)



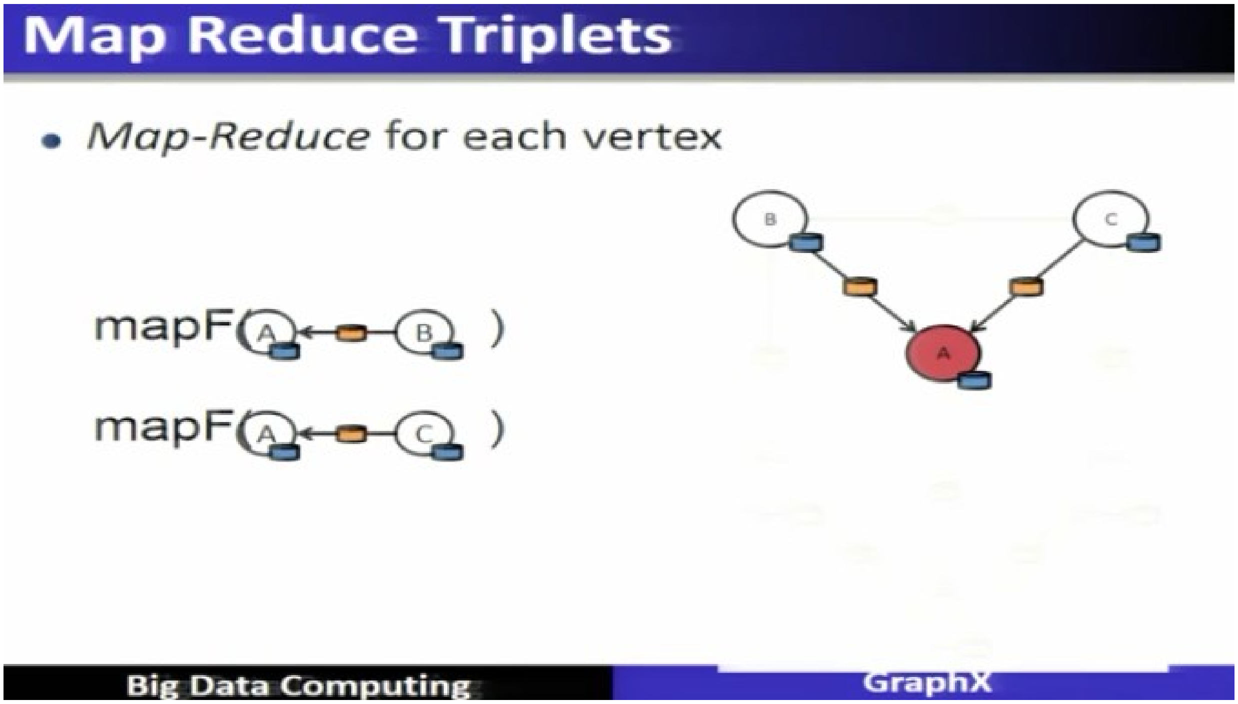
Map Reduce Triplets

- *Map-Reduce* for each vertex

Big Data Computing GraphX

let us see that how map-reduce using triplets can solve some of the important, some of the problems of a graph analytics, so for example this vertex A

(Refer Slide Time: 11:20)



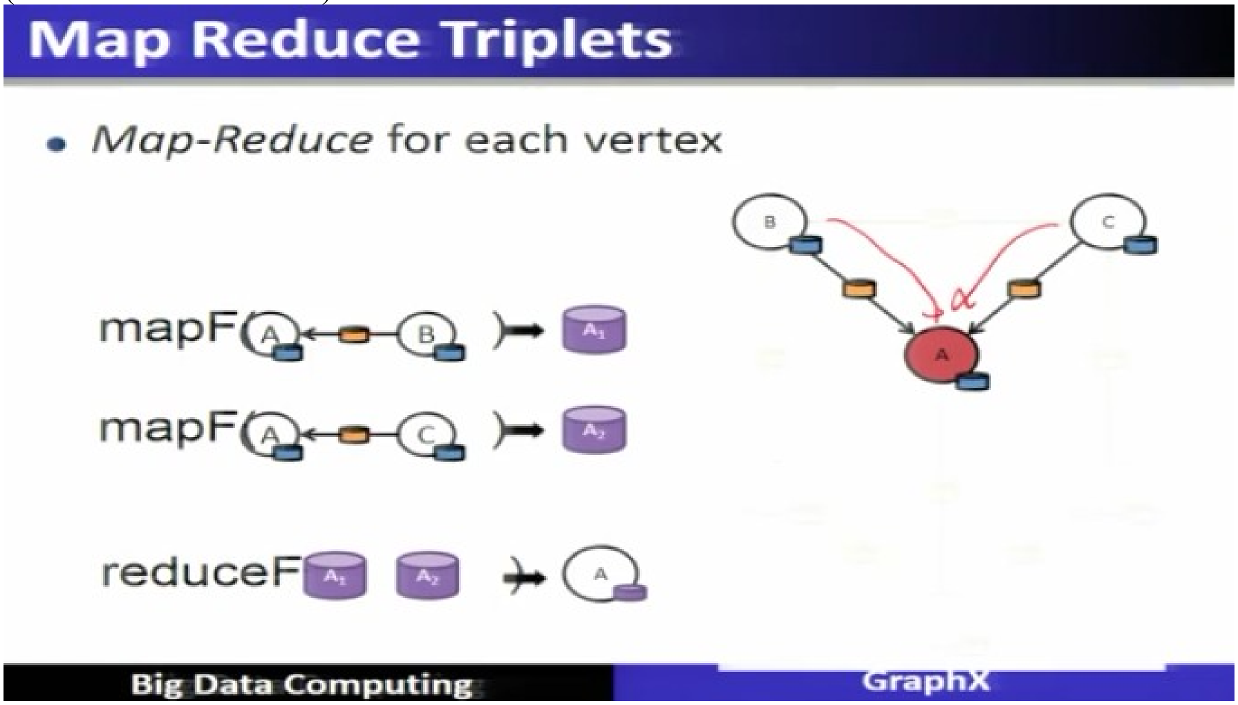
Map Reduce Triplets

- *Map-Reduce* for each vertex

Big Data Computing GraphX

want to do the computation, it requires the information from the neighboring vertices and using the map function on this triplet you can output, you can emit the A1, similarly for the map function between A and C, it will output A, and then reduce will combine them together

(Refer Slide Time: 11:48)



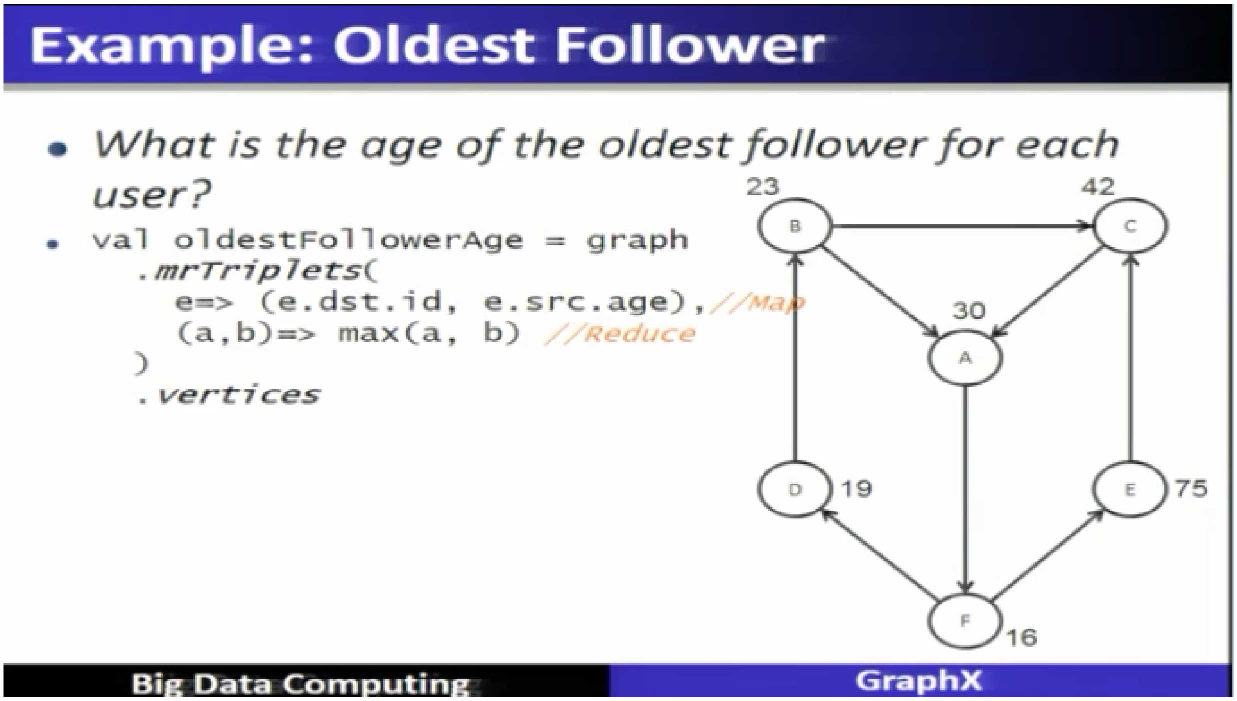
Map Reduce Triplets

- *Map-Reduce* for each vertex

Big Data Computing GraphX

this particular view, so once the triplet view is there then map function and reduce function can become very efficiently programmable in building the algorithms.

(Refer Slide Time: 12:00)



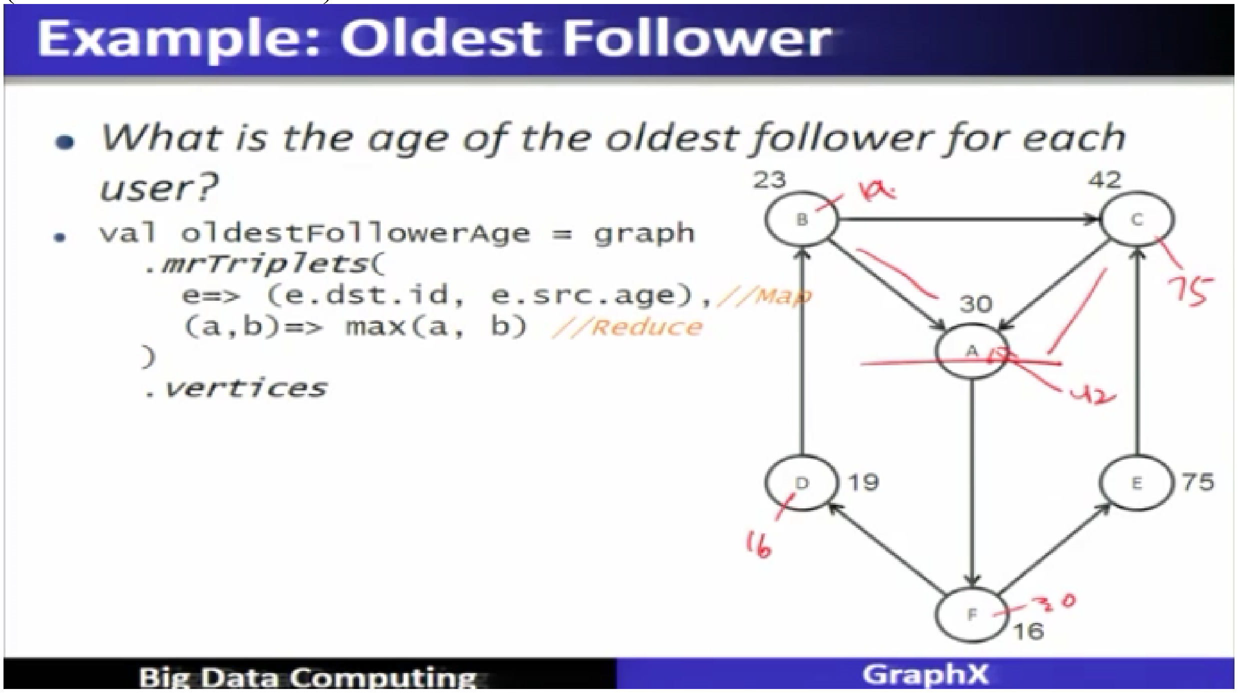
Example: Oldest Follower

- *What is the age of the oldest follower for each user?*
-
```
val oldestFollowerAge = graph
  .mrTriplets(
    e=> (e.dst.id, e.src.age),//Map
    (a,b)=> max(a, b) //Reduce
  )
  .vertices
```

Big Data Computing GraphX

Let us see that how using this triplet view we can write an algorithm or a program for finding the oldest follower, so the question is what is the age of the oldest follower for each of these users? So let us see a simple program which can solve this particular problem and, so you can see here that in this particular example, for example the node A is followed by C and, C and B, now let us see that it says that what is the age of the oldest follower of each user, so the follower of A, the oldest follower of A is 42, similarly you can find out the follower of C which is 75 and 23, this is 75 and then you can find out the follower of 16 is 30, and follower of 19 which is 16, and follower of D is basically 19,

(Refer Slide Time: 13:18)



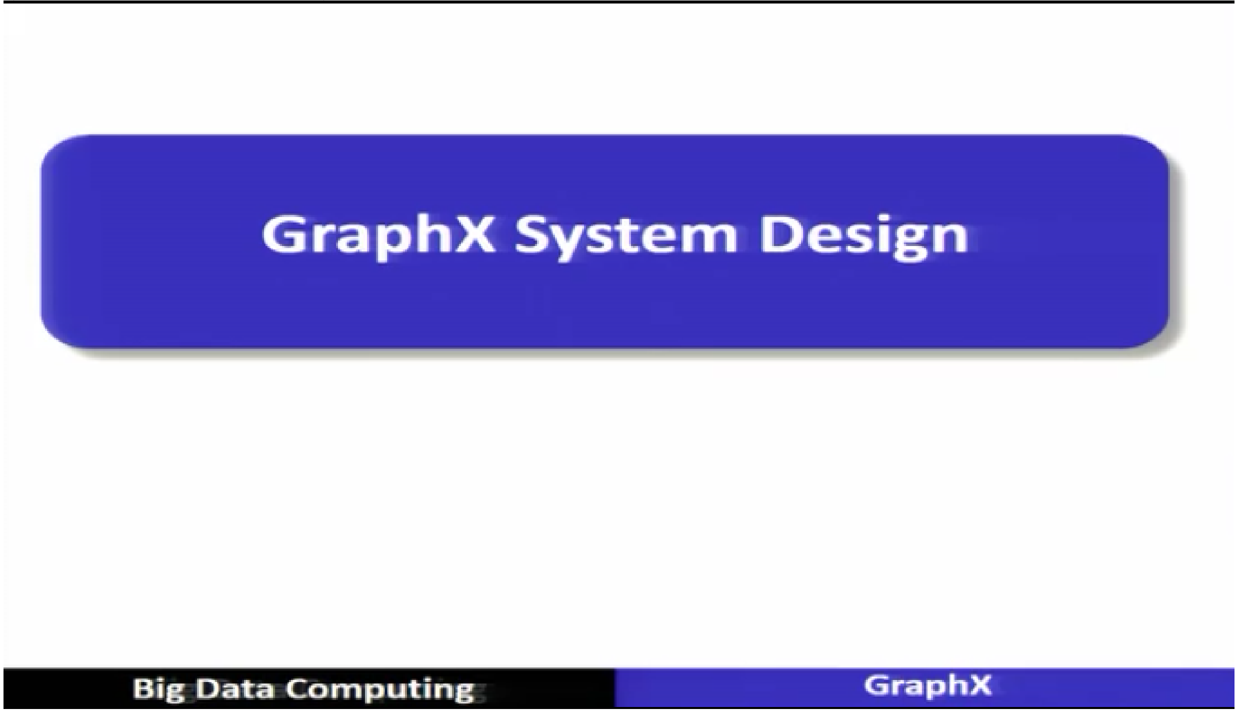
Example: Oldest Follower

- *What is the age of the oldest follower for each user?*
-
```
val oldestFollowerAge = graph
  .mrTriplets(
    e=> (e.dst.id, e.src.age),//Map
    (a,b)=> max(a, b) //Reduce
  )
  .vertices
```

Big Data Computing GraphX

so out of these different followers you can find out the maximum that is the answer will be, the C with the values 75, now this you can program using the triplet in a quite efficient manner, so you can find out using the graph triplet, using MR triplet which will say that the edges with the destination and source with the destination IDE and the source ID age will be now taken up as the maximum of all the triplets which are merging at any node, so you can see that this A will have this triplet, A will have another triplet, so it will find out the maximum of all the triplets a particular edge is having, and this way that will now then emit the age value and it will take the maximum in the reduce function, and this way you can see that using triplet it will be very simple program to find out these details.

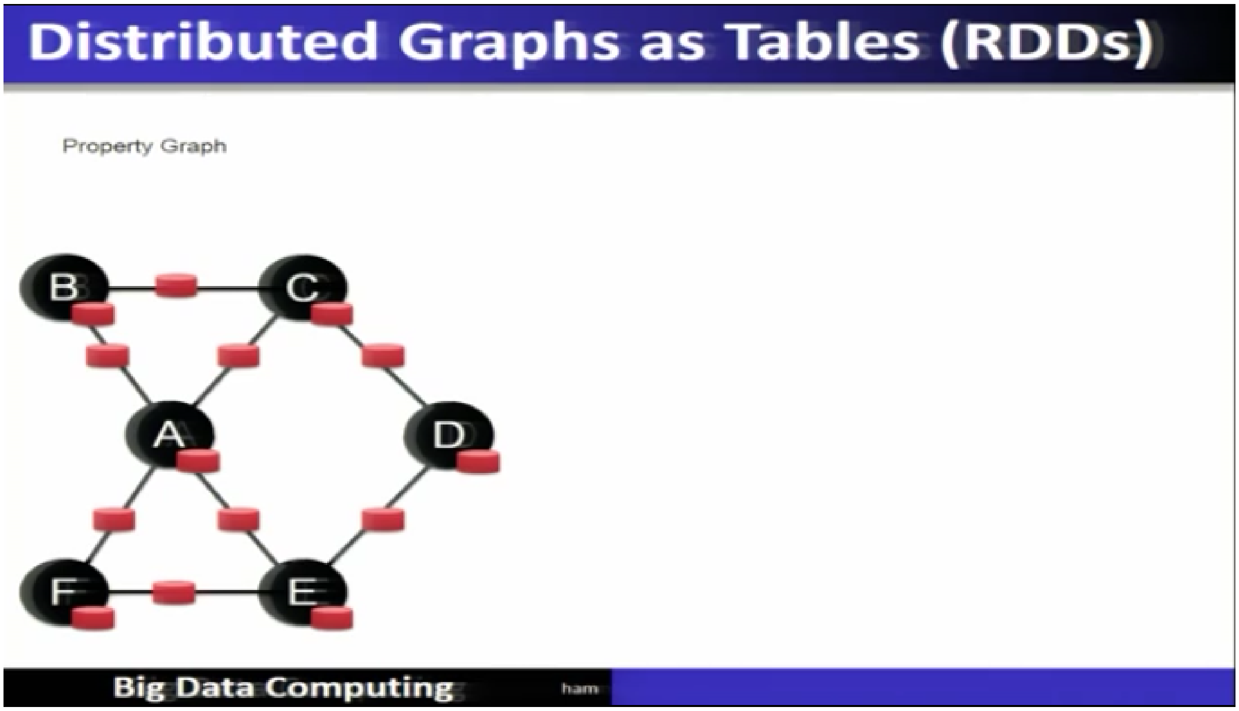
(Refer Slide Time: 14:30)



GraphX System Design

Big Data Computing GraphX

Let us see the GraphX system design details,
(Refer Slide Time: 14:38)



Distributed Graphs as Tables (RDDs)

Property Graph

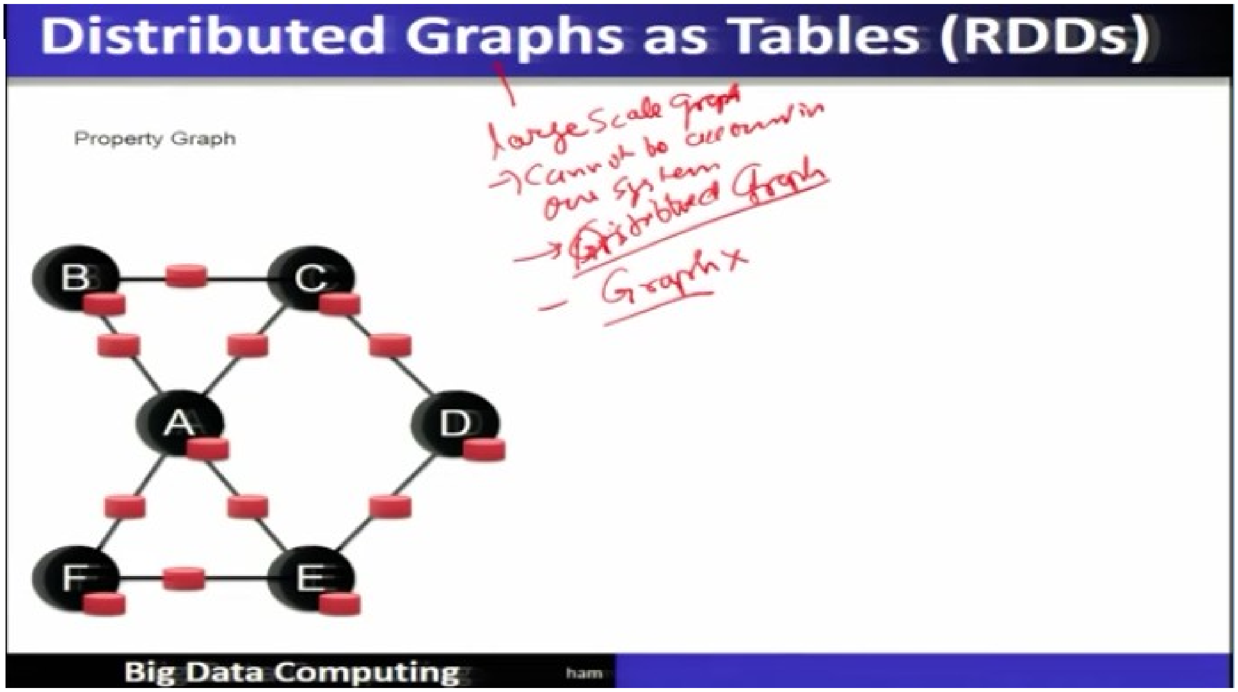
Big Data Computing

ham

now as we have seen that the graph is represented as the property graph, and which is nothing but the distributed tables, and here these different aspects that is the vertices, edges and property tables they are represented in the form of the RDD’s.

Let us see how this distributed graph is internally managed in GraphX, why distributed graph is required because the graph is very large, is called large scale graphs, now this large scale graph cannot be accommodated in one computer system, therefore it will be stored in a distributed manner, hence it is called a distributed graph.

If you say it’s a distributed graph that means the graph can be cut into different pieces and being stored across the cluster of machines, how that is all done, we are going to see how the GraphX will support this distributed graph.
(Refer Slide Time: 15:54)



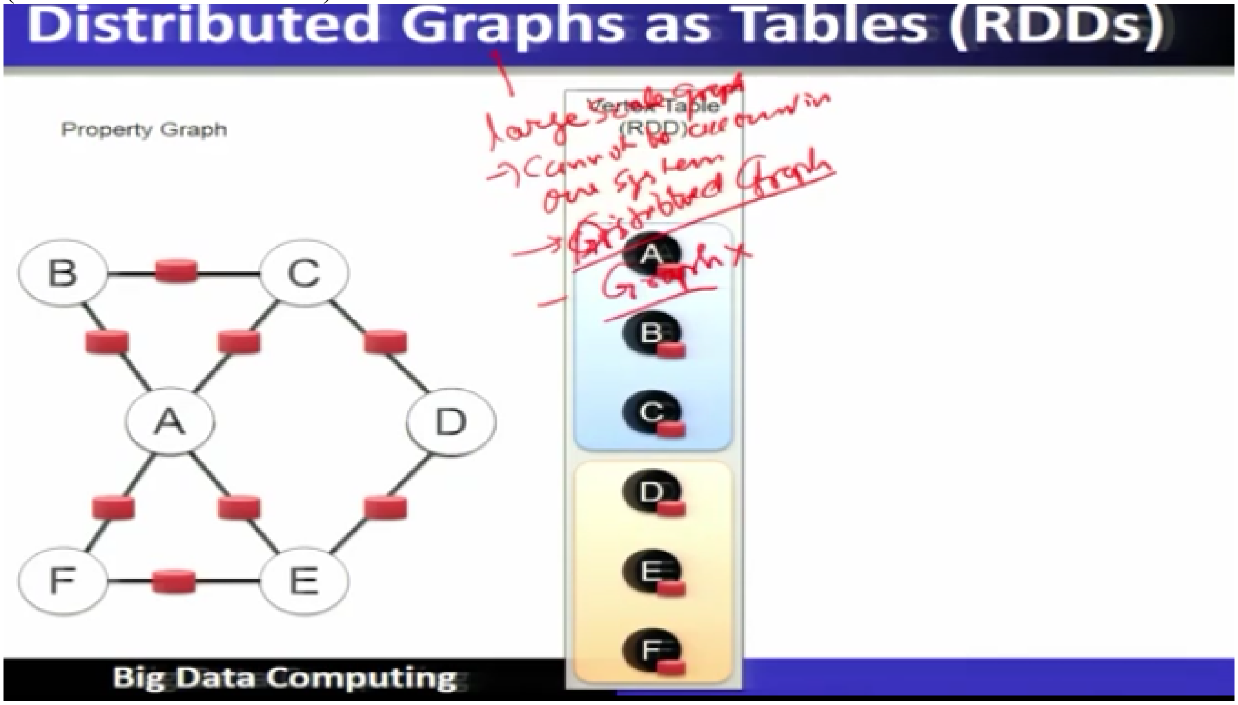
Distributed Graphs as Tables (RDDs)

Property Graph

large Scale Graph
→ Cannot be accommodated in one system
→ Distributed Graph
- GraphX

Big Data Computing

So this is the example of a small property graph, let us see how it is done in the GraphX,
(Refer Slide Time: 16:05)



Distributed Graphs as Tables (RDDs)

Property Graph

Vertex Table (RDD)

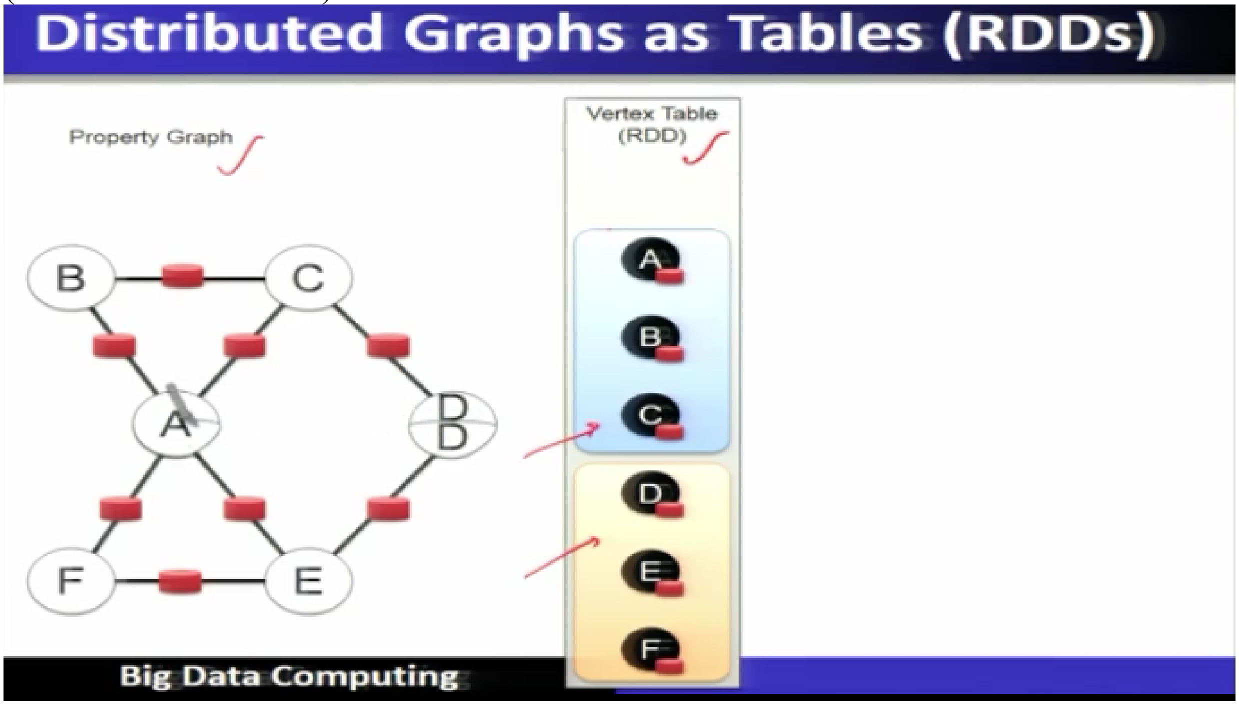
large Scale Graph
→ Cannot be accommodated in one system
→ Distributed Graph
- GraphX

Big Data Computing

so the graph comprises of two table which is called vertex, now this particular GraphX which is this particular graph which is called a property graph has the vertex property table and the edge property table, now vertex property table will be represented as the distributed table of this vertex table, so this is called vertex table RDD's which are supported in the GraphX, so the vertices, all the vertices of a graph let us say that it's a millions of vertices so it cannot be accommodated in one table, in one computer system so obviously several computer systems are required.

Let us understand that two computer systems is good enough for this example to store the vertex table in the form of RDDs,

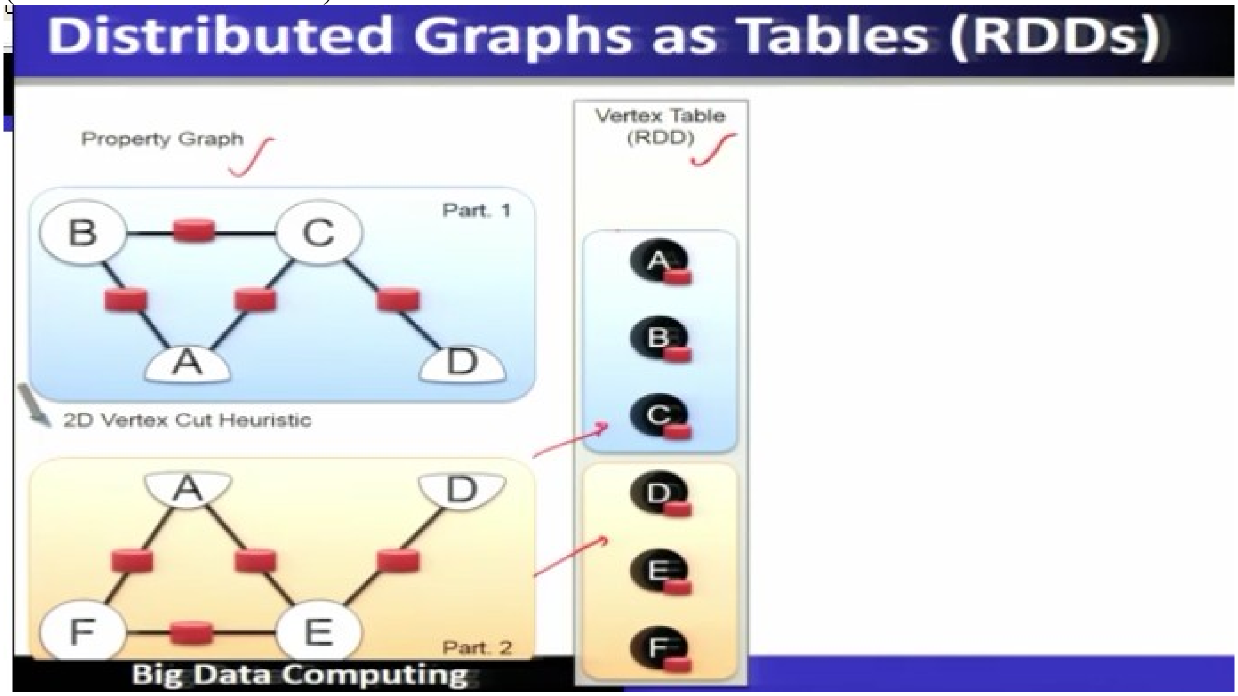
(Refer Slide Time: 17:11)



Distributed Graphs as Tables (RDDs)

now as far as the graph is concerned we are going to cut this particular graph across the vertices

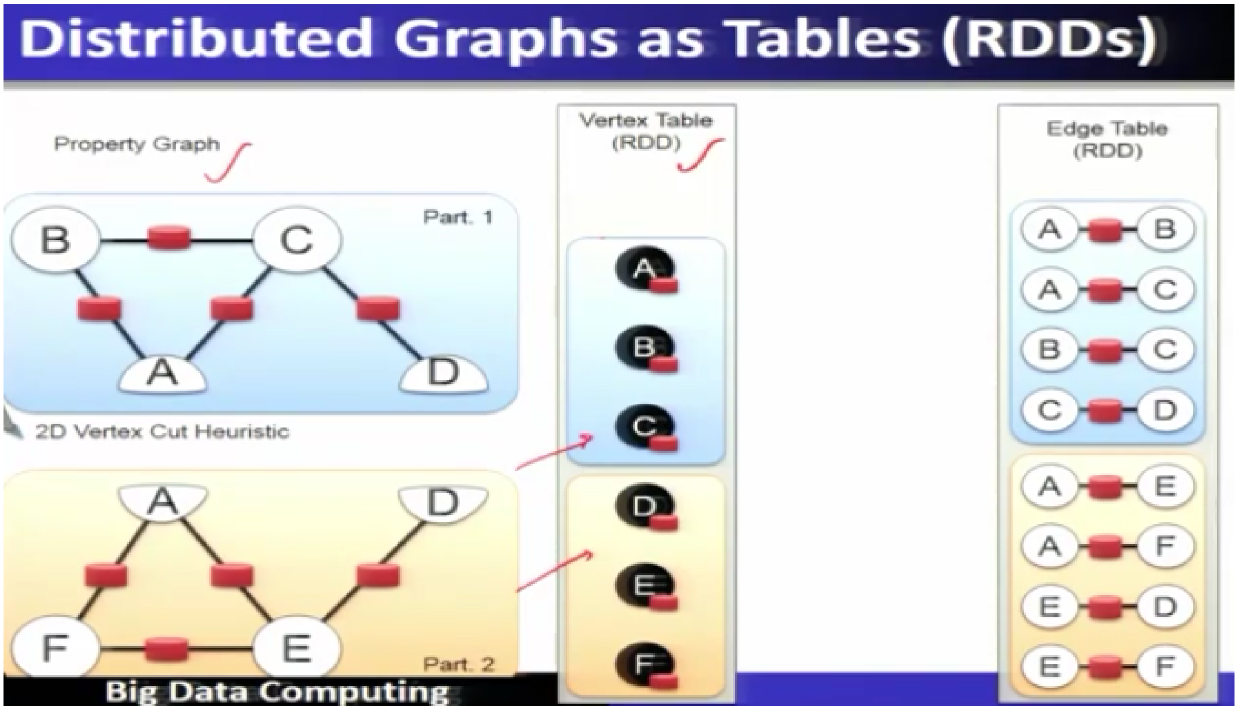
(Refer Slide Time: 17:20)



Distributed Graphs as Tables (RDDs)

and this particular way these part of the graphs are stored in as a distributed graphs.

(Refer Slide Time: 17:30)



Distributed Graphs as Tables (RDDs)

Property Graph

Vertex Table (RDD)

Edge Table (RDD)

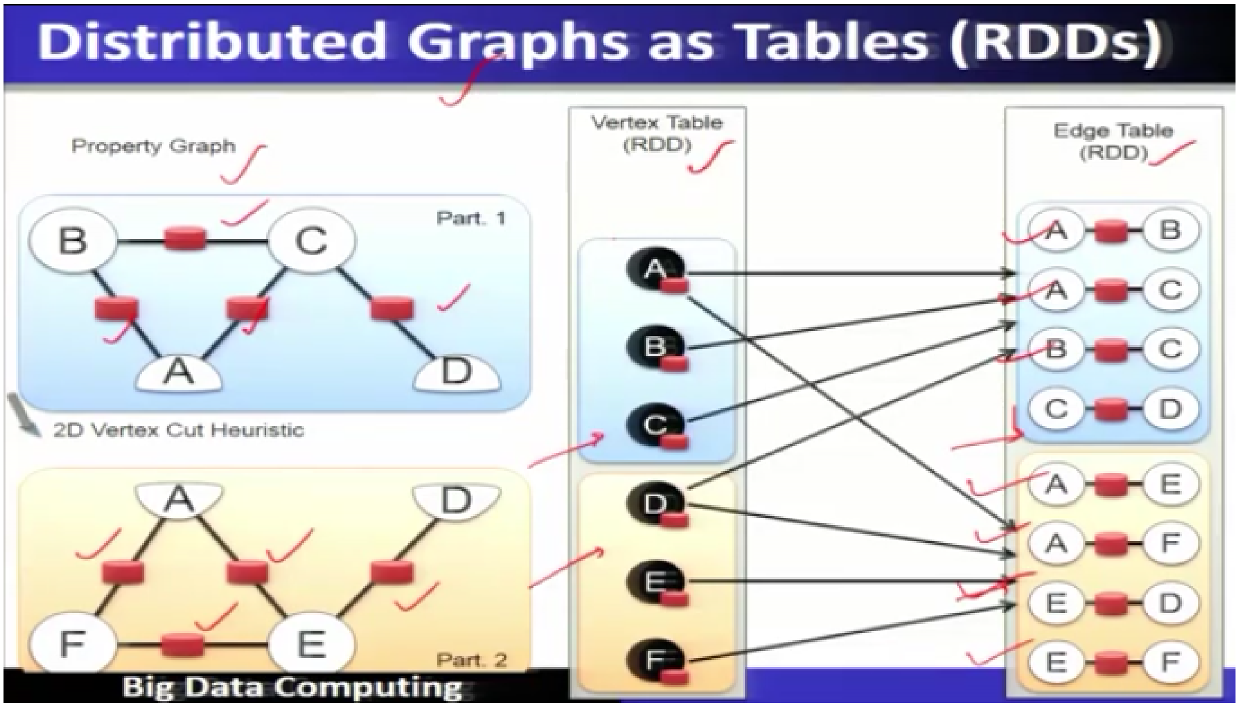
2D Vertex Cut Heuristic

Big Data Computing

So let us see that once we cut the vertices, then there are two parts, part 1 and part 2, so part 1 and part 2 they are stored in different machines, so obviously the edge part that is called edge property table which is nothing but edge table RDD which is stored in 2 different machines like this, so the first part will have AB and then AC, the BC and then CD, so the first part is represented in the form of a edge table, similarly the second part which is called AF, then AE, the FE, and then ED, so all the edges of the second part is stored on the second node, so this way the vertices are also is split and stored in different nodes, edges are also stored on different nodes and this way it is called the distributed graphs which are stored in this way.

Now the thing is one over, there will be a communication between AB or AC or between AF and so how this is all facilitated, that we are going to see because all these vertices and edges they are stored on different nodes, and it is called distributed graph,

(Refer Slide Time: 18:59)

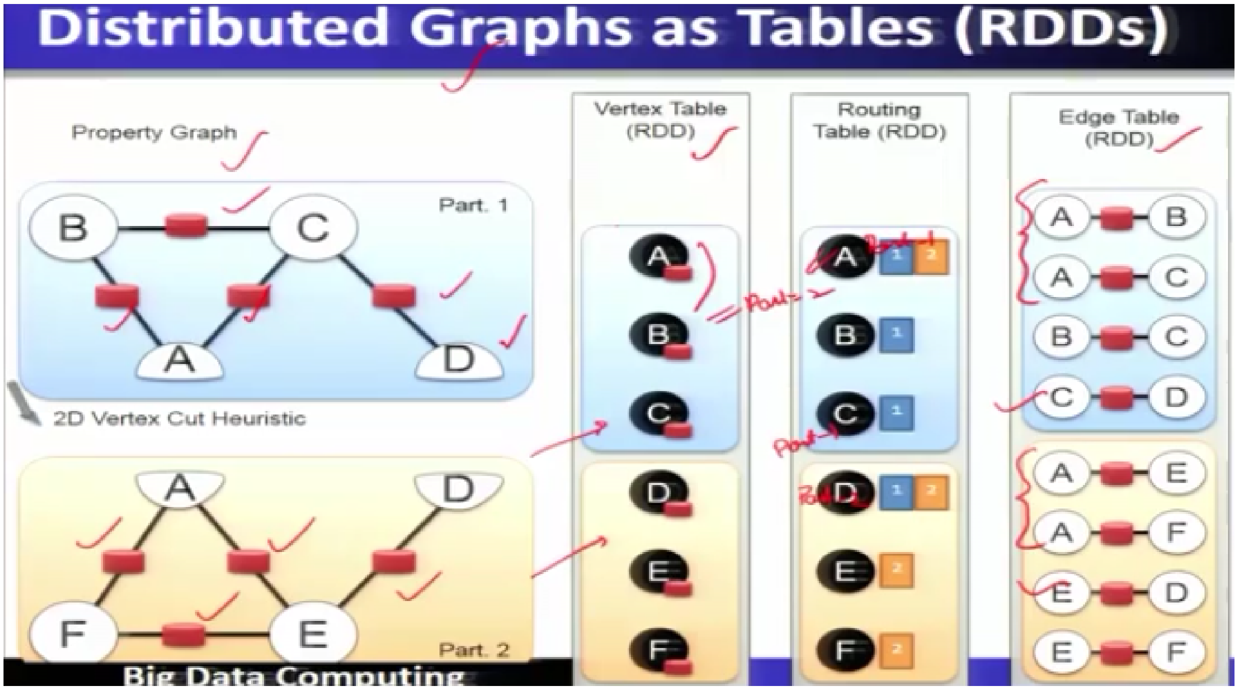


Distributed Graphs as Tables (RDDs)

so for example A would like to communicate with the both the nodes that is the part 1 and the part 2, both the parts will be stored on two different nodes, so this will be communicating with part 1 and the other communication is shown by the arrows as of the part 2, so A will be communicating to part 1 and A will be communicating to part 2, so to support these edges that is, to support these edges AE and AF, similarly A will communicate with part 1 to support the edges that is AB and AC.

Similarly as far as B is concerned, B can communicate with BC and BA, so BC and BA, so BC and BA both are in part 1, so it does not required to be communicating with the part 2 in any optic case.

Similarly D is concerned, D will have the communication with C on part 1 and D will have communication with E on part 1, so D will have part 1, and D will have communication with part 2 also, so therefore CD will be part 1, and then DE will be on the part 2, so therefore sum of these vertices requires to communicate in all the parts where the edge table is being stored, and some of the nodes are required to be communicating with or single part,
(Refer Slide Time: 20:53)

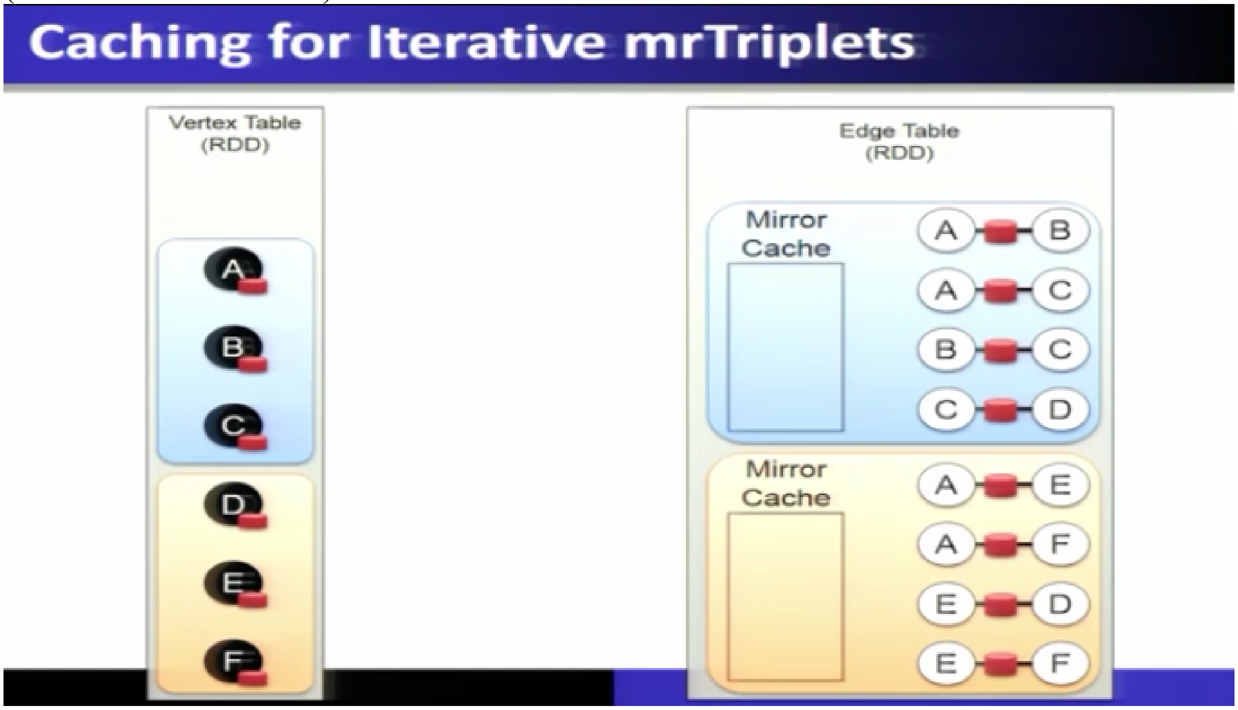


Distributed Graphs as Tables (RDDs)

so therefore a routing table will capture this information and this is shown over here that for the node A it has to communicate with the part 1 and part 2, for B will has to communicate with part 1, C has to communicate with part 1, D has to communicate with both the parts, so also for E and F, so once the routing table capture this information therefore the interactions between the nodes and their corresponding edges can be facilitated using the routing table.

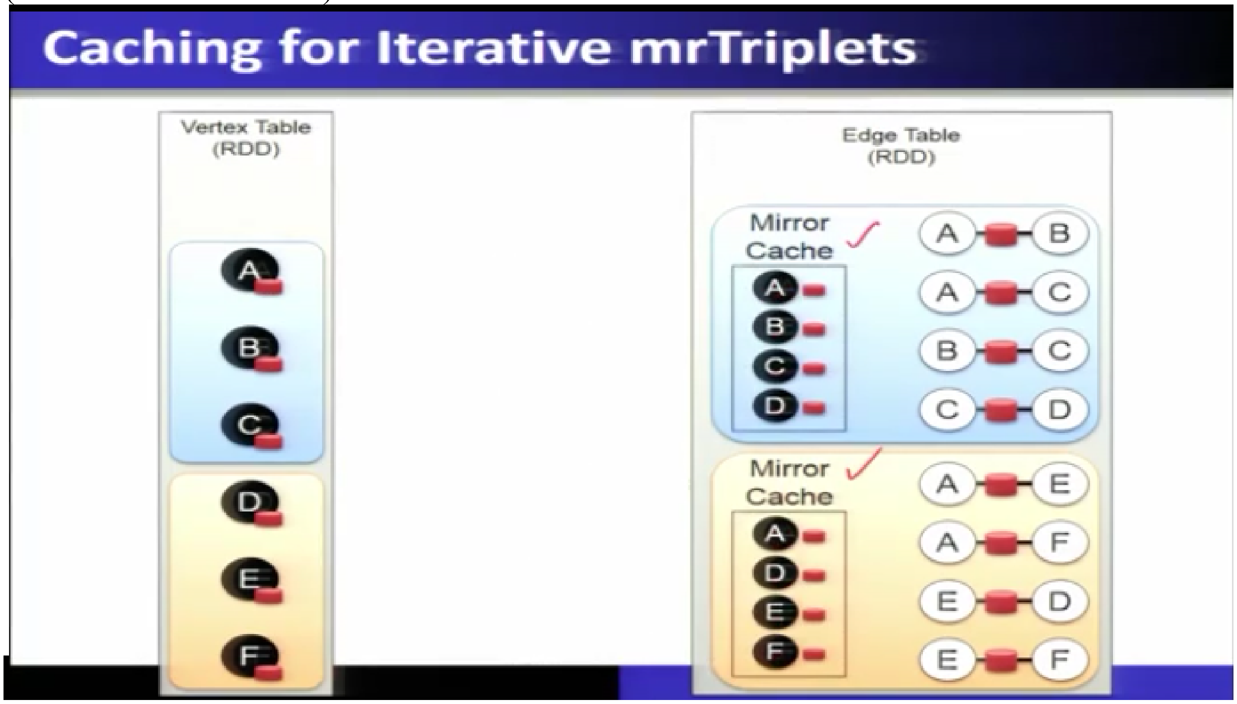
Now if we keep this information of the routing table part with the edge table also as a part of caching then this communication cost can also be reduced.

(Refer Slide Time: 21:45)



Caching for Iterative mrTriplets

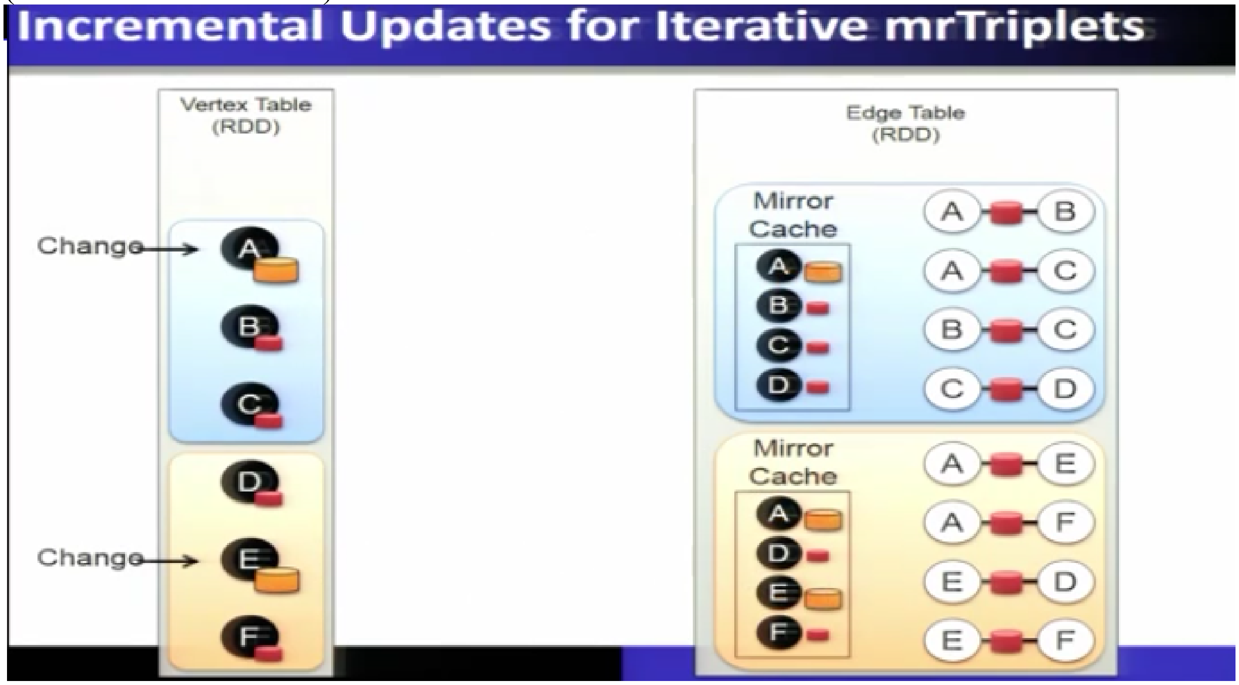
So here we can see here that the edge table will now contain the mirror cache of the routing table enter this,
(Refer Slide Time: 21:54)



Caching for Iterative mrTriplets

and therefore the vertices which are required to communicate in the first portion, the first part will be stored in its mirror cache, similarly the nodes which are required to communicate on the second part will require to be stored in the mirror cache of that corresponding part.

Now whenever there is a change happening at the vertex table on those nodes,
(Refer Slide Time: 22:28)

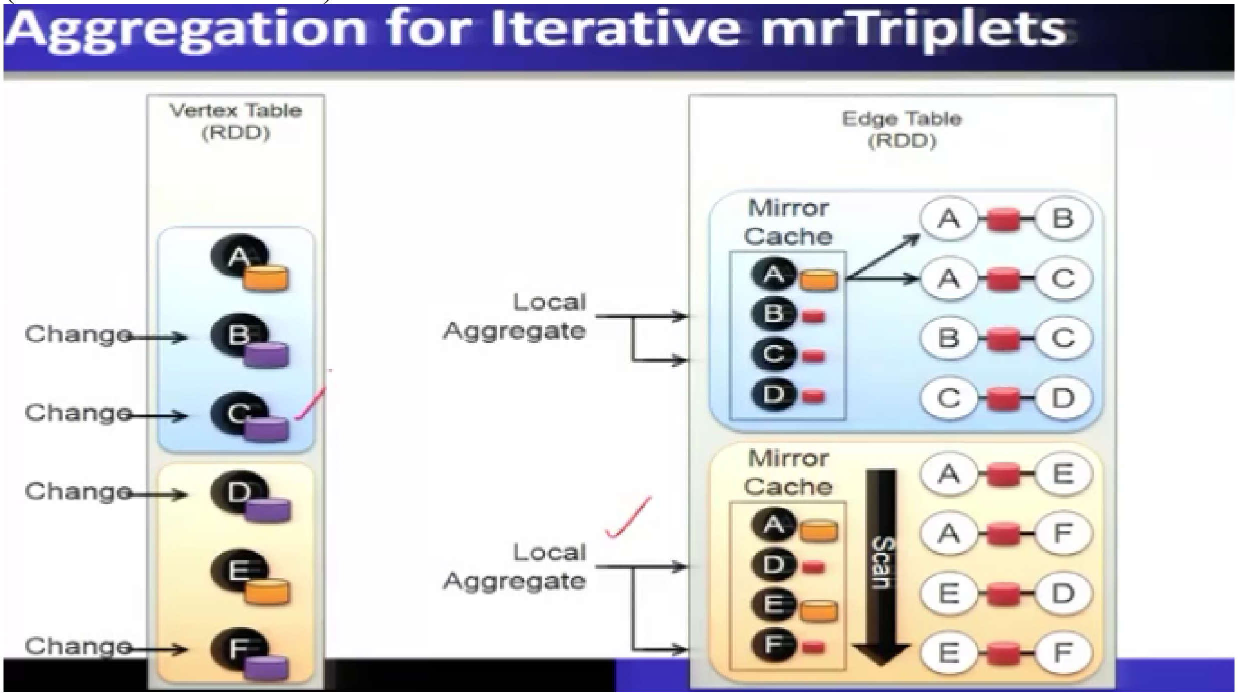


Incremental Updates for Iterative mrTriplets

and these particular changes are to be communicated in the mirror cache, and this mirror cache in turn will propagate those changes further on the edge table.

Now as far as whenever this particular mirror cache will be now whenever the nodes are is scanned and requires any changes in the mirror cache, so basically a local aggregation will be performed and this local aggregation will be communicated back to the vertex table, so therefore the number of communications are to be reduced here in this way of implementing the mirror cache and performing the local aggregates,

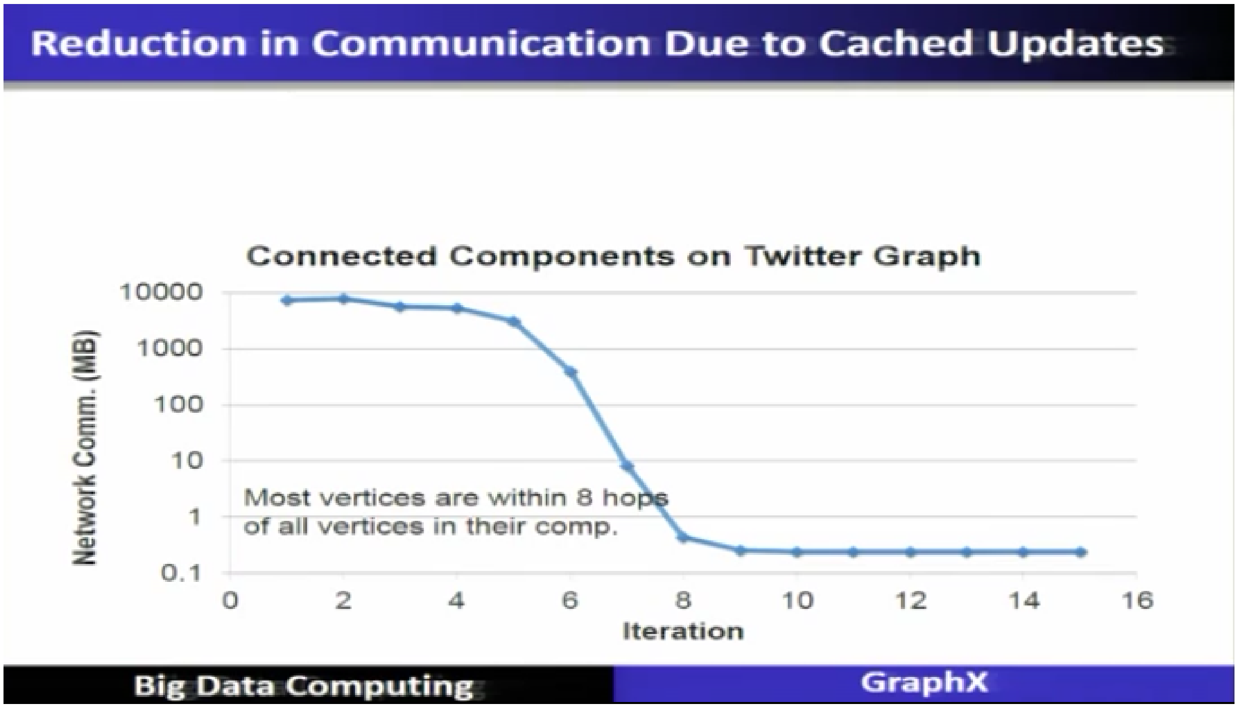
(Refer Slide Time: 23:17)



Aggregation for Iterative mrTriplets

so whether it is in the vertex whenever some modifications are done, or at while the scanning of the edges, any computation or any changes are happening at this edges all this will be now aggregated, and the number of communications can be reduced here in this way, hence the performance can be increased.

(Refer Slide Time: 23:43)



Reduction in Communication Due to Cached Updates

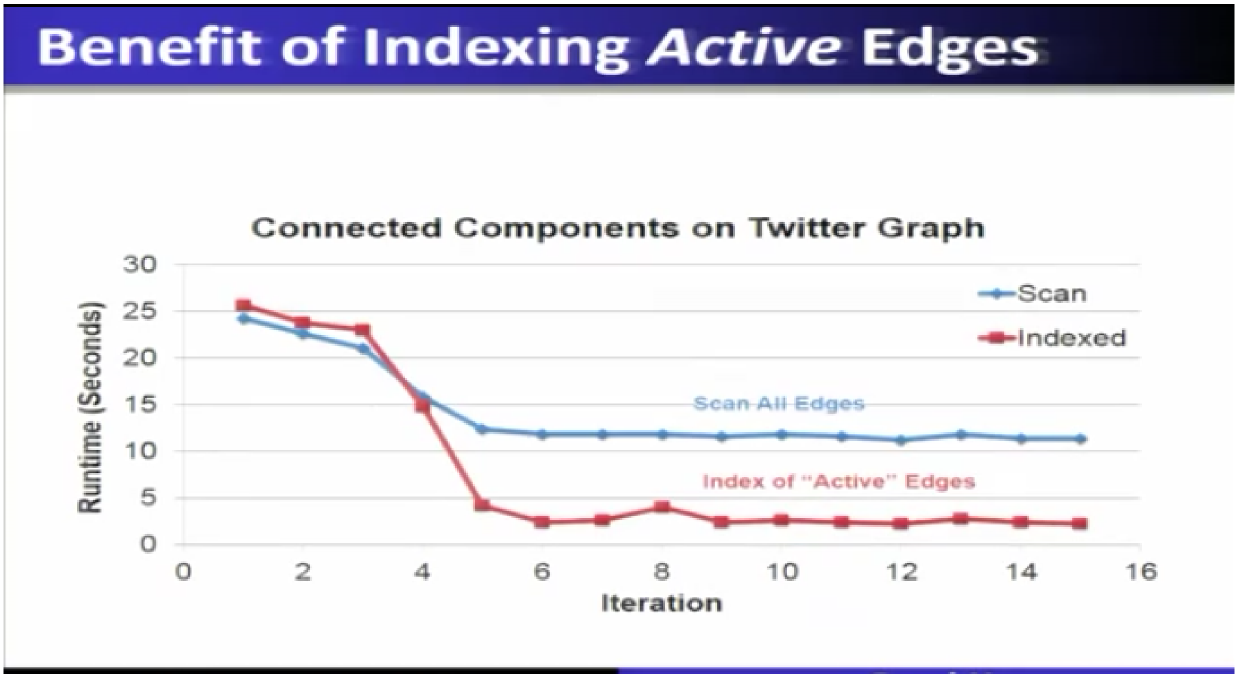
Most vertices are within 8 hops of all vertices in their comp.

Big Data Computing

GraphX

Now if you measure how much reduction in the communication is done due to the cache updates, so we can see here that even for the connected components, computation on a twitter graph we can see that so many number of that means the communication is heavily dropped here in this particular case, therefore the performance as we have seen earlier in the previous slide that if it is graph or then Pregel or GraphLab, when you compare to the GraphX, so GraphX becomes a better performance in compared to any other graph framework, graph computation framework which are available due to these internal details of implementation using cache updates.

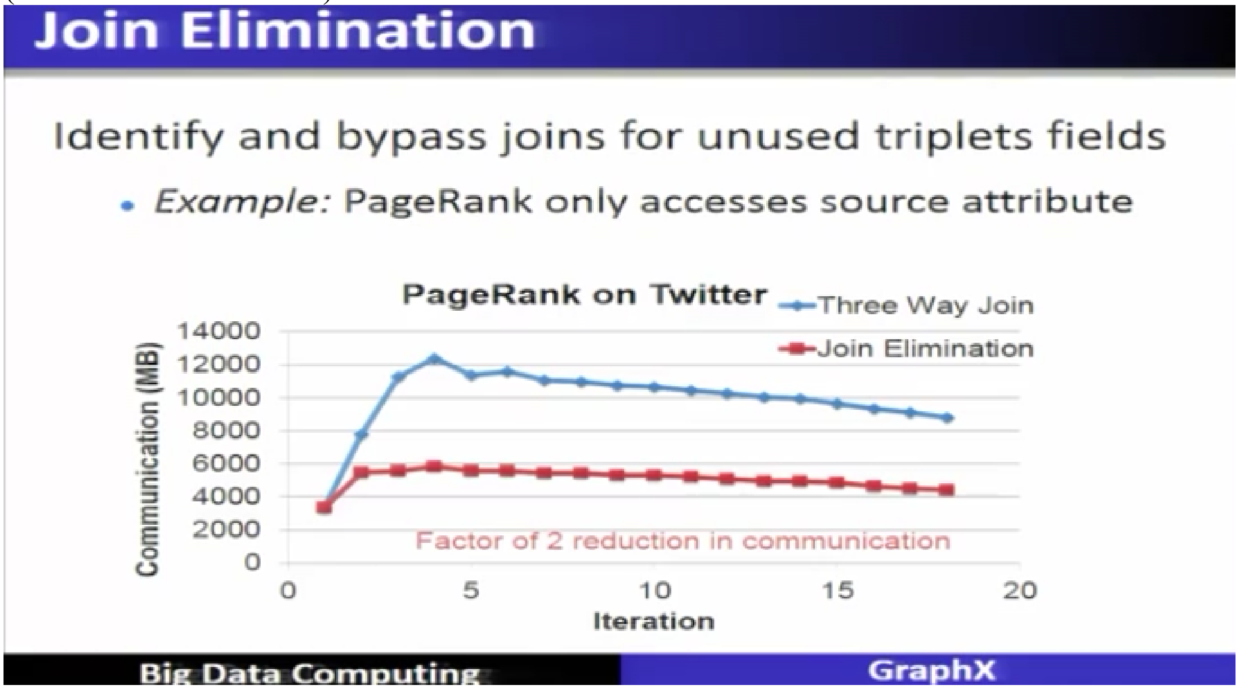
(Refer Slide Time: 24:38)



Benefit of Indexing *Active* Edges

So these are, we can see that this wave of scanning the edges and aggregating the local updates and communicating with them is nothing but we are indexing, they are indexing the active edges, so the indexing of active edges again in turn will reduce the run time and, why because, these run times will be reduced because all the data is available in memory and they can be executed efficiently, and the effect is shown here in this particular graph, the run time and the execution time is heavily reduced, why because the indexes of active pages is being handled automatically internally has the time will be reduced.

(Refer Slide Time: 25:29)



Join Elimination

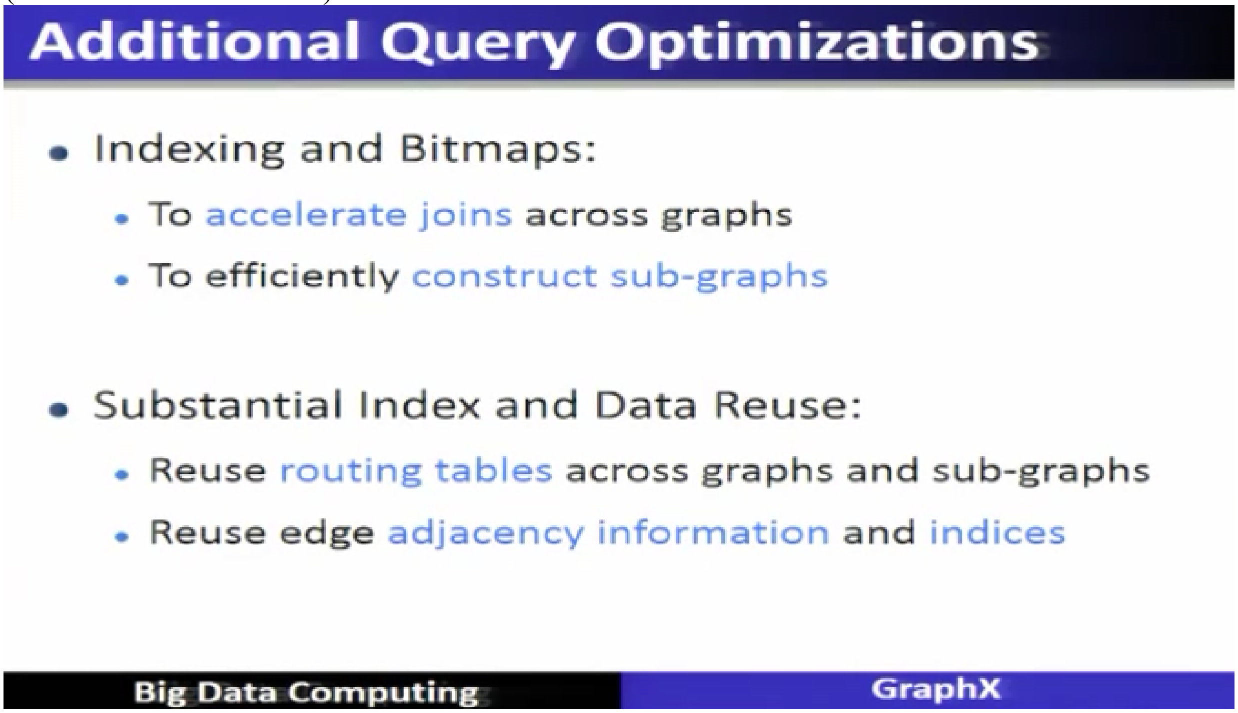
Identify and bypass joins for unused triplets fields

- *Example:* PageRank only accesses source attribute

Big Data Computing GraphX

Similarly in this particular way we have eliminated the join, so identifying and bypass the join for unused triplet fields, so even for the PageRank only accesses the source attributes, so therefore the factor of two reduction in the communication due to the join elimination you can see the improvement in the communication is quite reduced,

(Refer Slide Time: 26:01)



Additional Query Optimizations

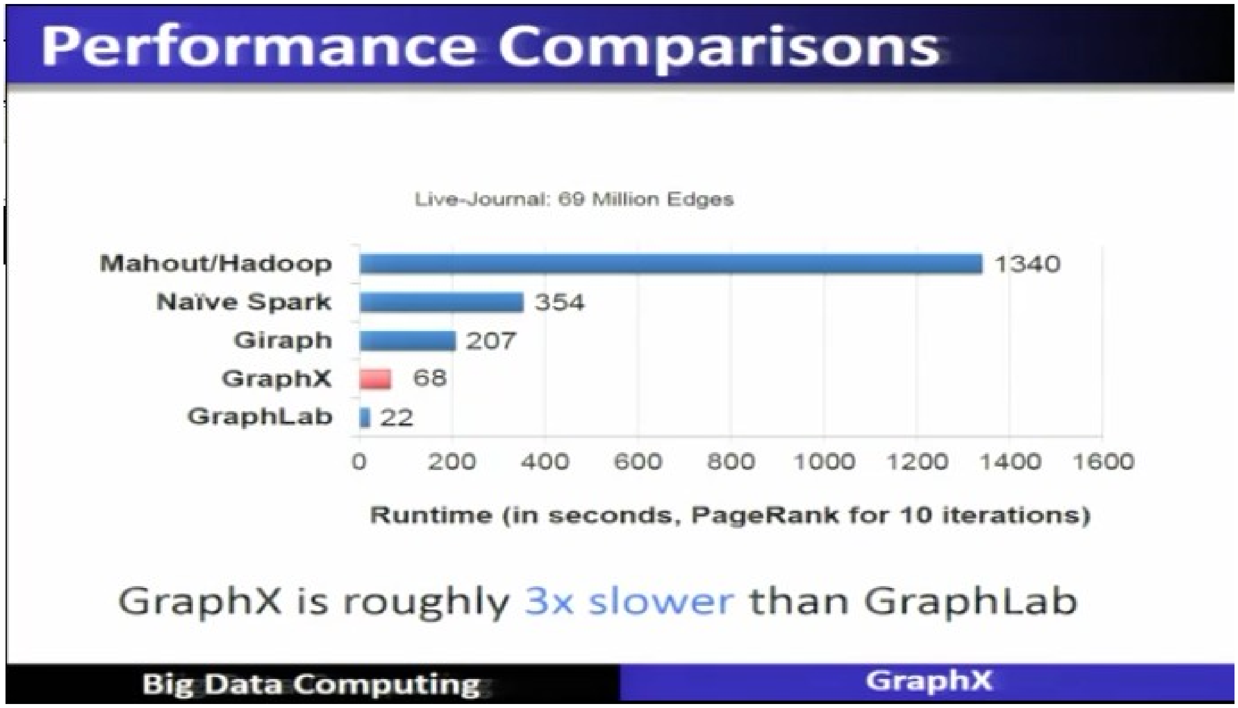
- Indexing and Bitmaps:
  - To accelerate joins across graphs
  - To efficiently construct sub-graphs
- Substantial Index and Data Reuse:
  - Reuse routing tables across graphs and sub-graphs
  - Reuse edge adjacency information and indices

Big Data Computing GraphX

therefore there are additional query optimizations also available for indexing and bitmaps to accelerate joins across the graphs, to efficiently construct sub-graphs, substantial index and data reuse, so reuse routing tables across graphs and sub-graphs, reuse edge adjacency information and indices, so these are all different query optimizations which are possible so that it will accelerate either the joins across the graph or it can efficiently construct the sub-graph or using the index and data reuse, it can reuse the routing table across the graphs and also reuse the adjacency information and indices.

To summarize we can say that in this particular framework that is the GraphX, various optimizations are already in place and we can exploit using these optimizations to make more efficient algorithms on the graphs.

(Refer Slide Time: 27:18)



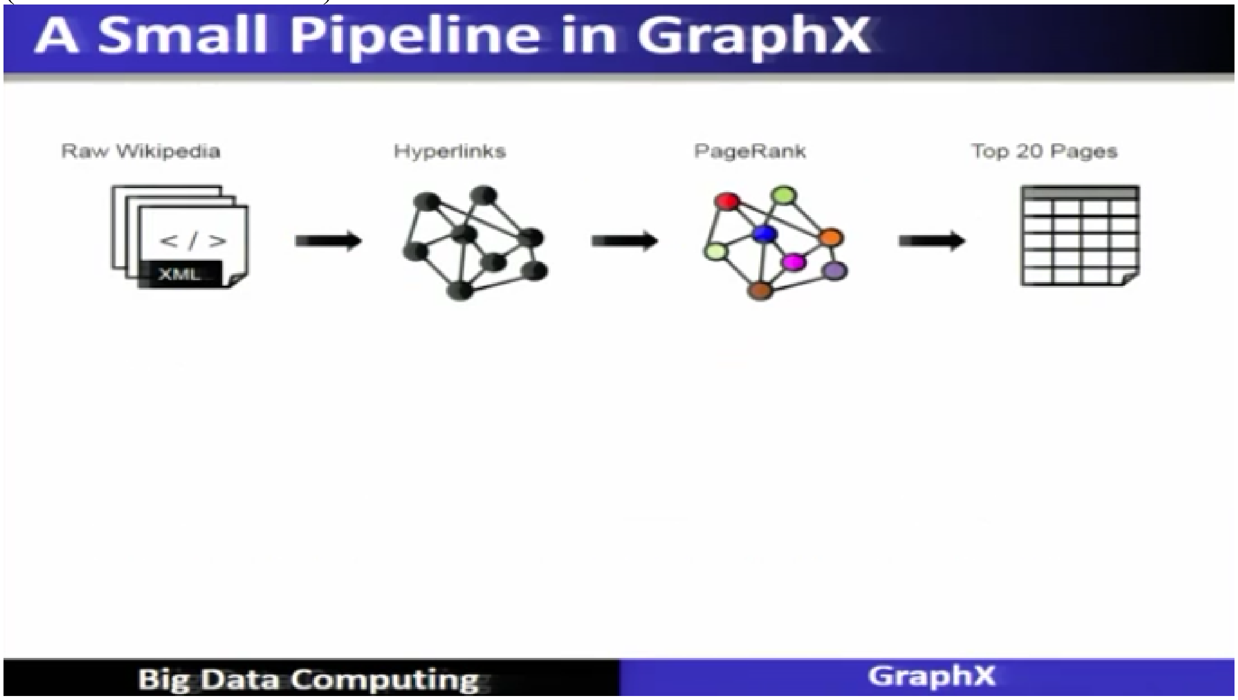
Performance Comparisons

GraphX is roughly 3x slower than GraphLab

Big Data Computing GraphX

So if you measure the performance we can see that the GraphX is roughly 3 times slower than the GraphLab, so therefore this large graph can be easily computable using the GraphX.

(Refer Slide Time: 27:41)

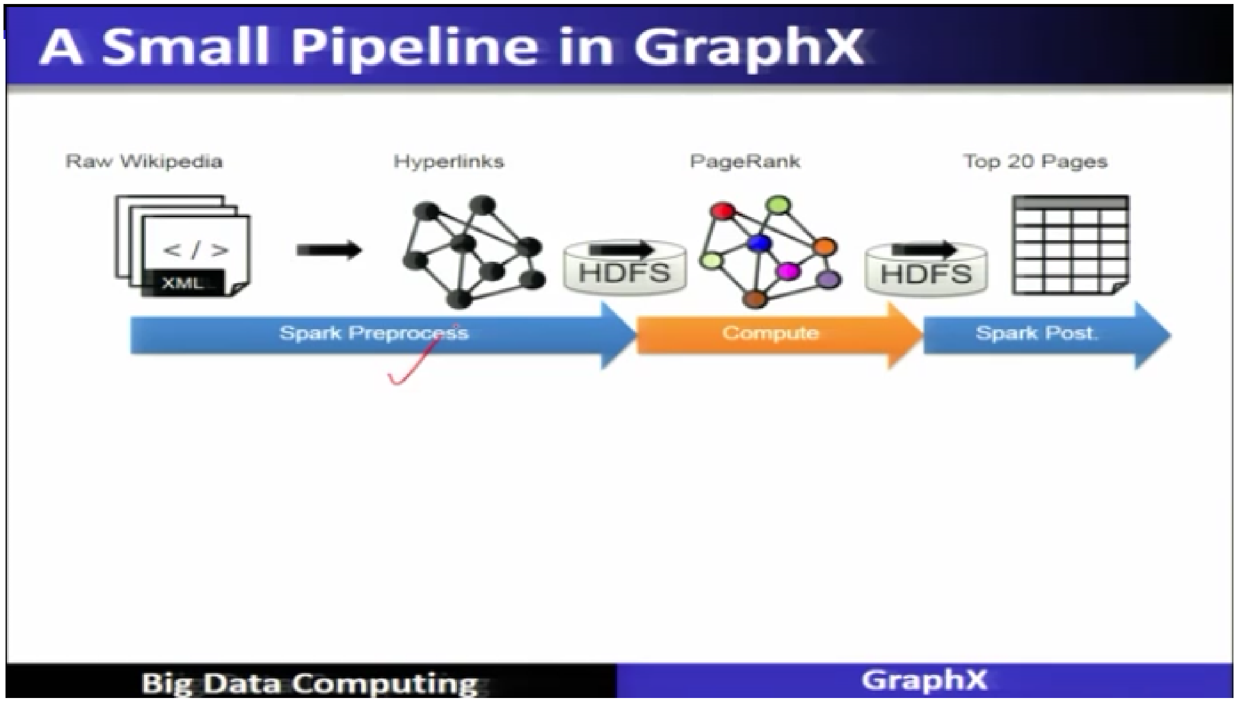


A Small Pipeline in GraphX

Big Data Computing GraphX

So let us see that again we have to come back and see that we can also build a small pipeline in GraphX and this pipeline can be optimized so that it will be getting better performance, so pipeline intern will start with a spark processing and framework,

(Refer Slide Time: 28:06)

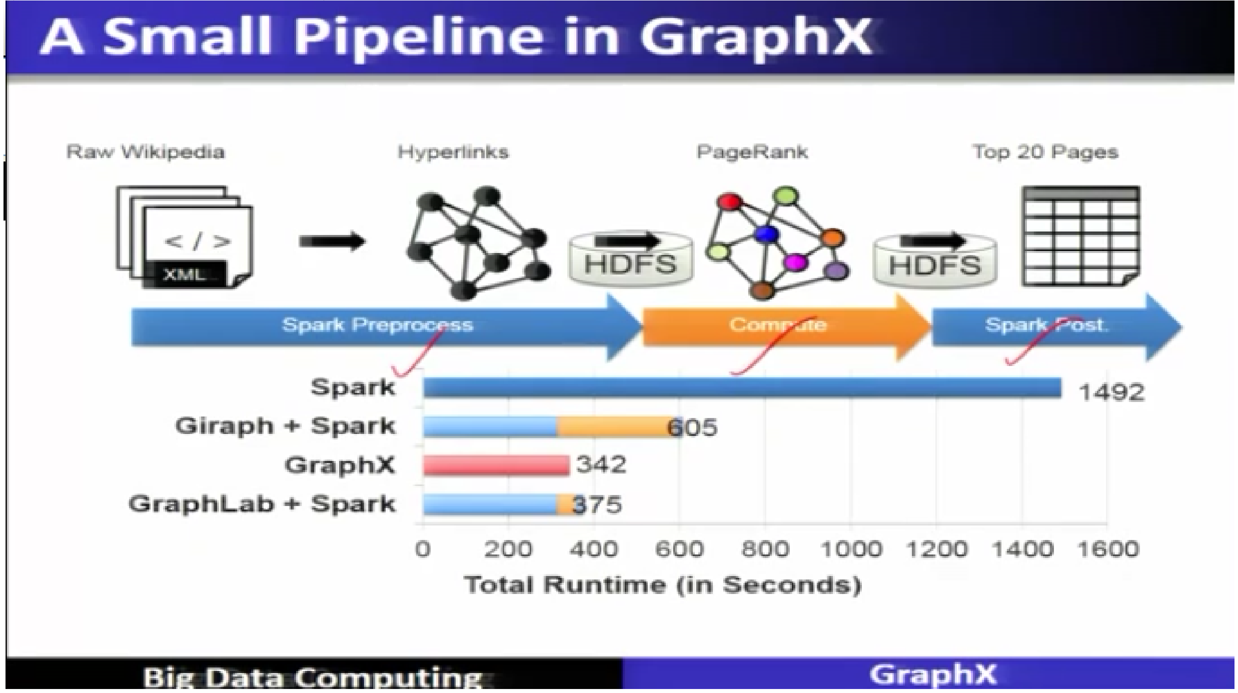


A Small Pipeline in GraphX

Big Data Computing GraphX

so spark preprocessed will take the raw Wikipedia data and convert in the form of a hyperlink graph, and use the HDFS to compute this PageRank and then perform the spark processing to capture the top 20 pages,

(Refer Slide Time: 28:24)

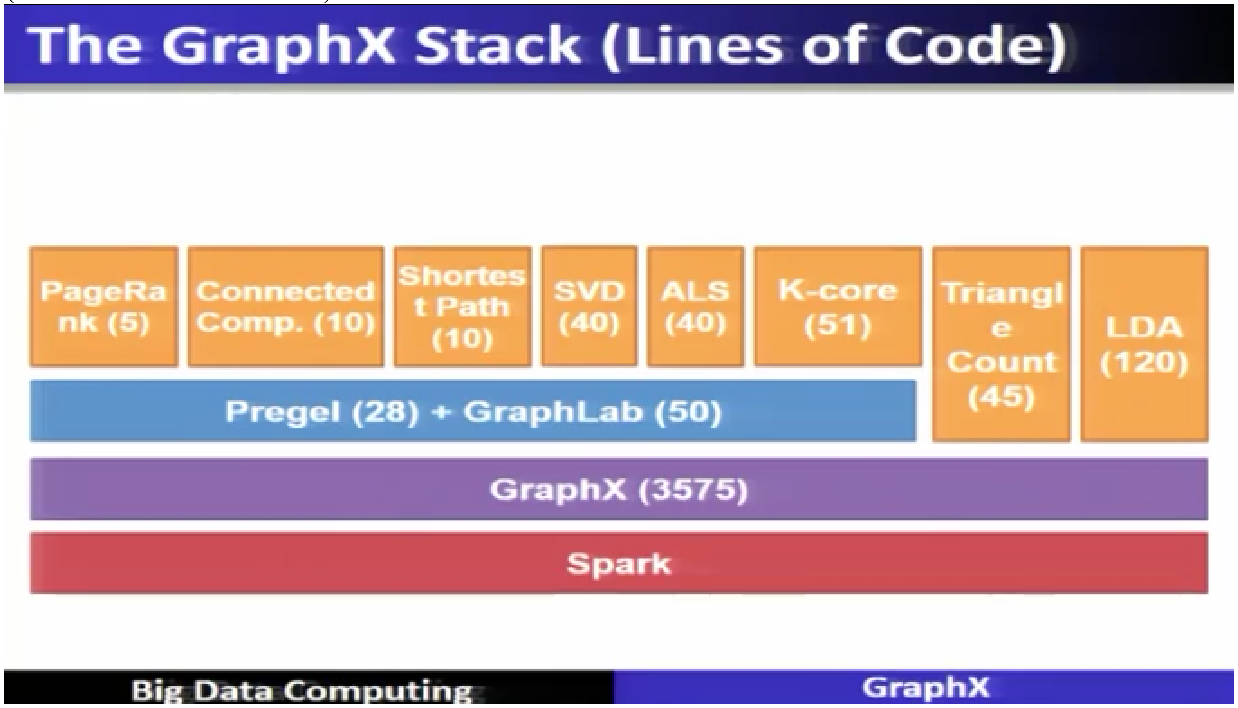


A Small Pipeline in GraphX

Big Data Computing GraphX

so is you use this particular pipeline, so you see that the smallest time which is basically is taking, is using GraphX, why because? This view of table and the graph is integrated or basically unified in GraphX, whereas in all other framework this particular pipeline needs lot of inefficiencies due to the internal HDFS storage for handling two different viewpoints, so therefore time end to end GraphX is faster than GraphLab also, so due to the fact that we have already seen that it has unified view of tables and graph which are supported in GraphX,

(Refer Slide Time: 29:32)

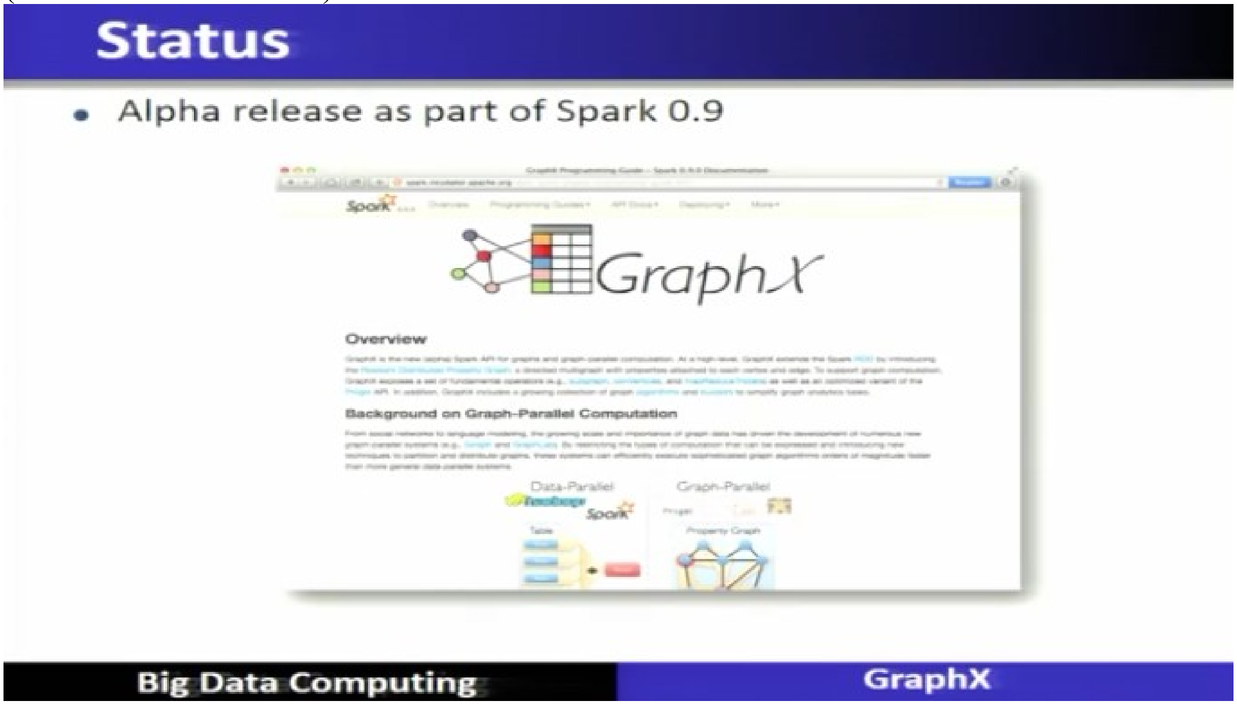


The GraphX Stack (Lines of Code)

Big Data Computing GraphX

So therefore GraphX stack has this kind of features that means it has these different lines of code which is great and above the spark, and different algorithms are available either through the Pregel or GraphLab API’s or directly which is implemented as the libraries of GraphX.

(Refer Slide Time: 29:58)



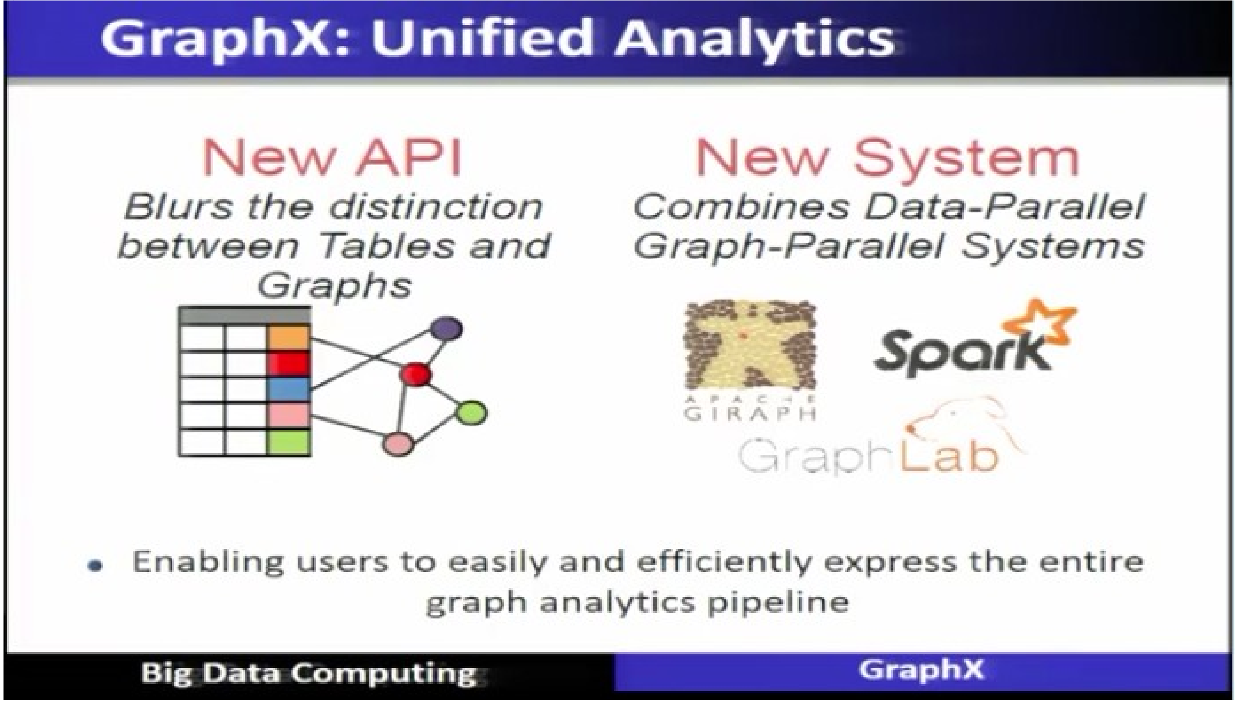
Status

- Alpha release as part of Spark 0.9

Big Data Computing GraphX

Now here we have discussed the GraphX which is an alpha release apart of the spark 0.9,

(Refer Slide Time: 30:07)



GraphX: Unified Analytics

New API

Blurs the distinction between Tables and Graphs

New System

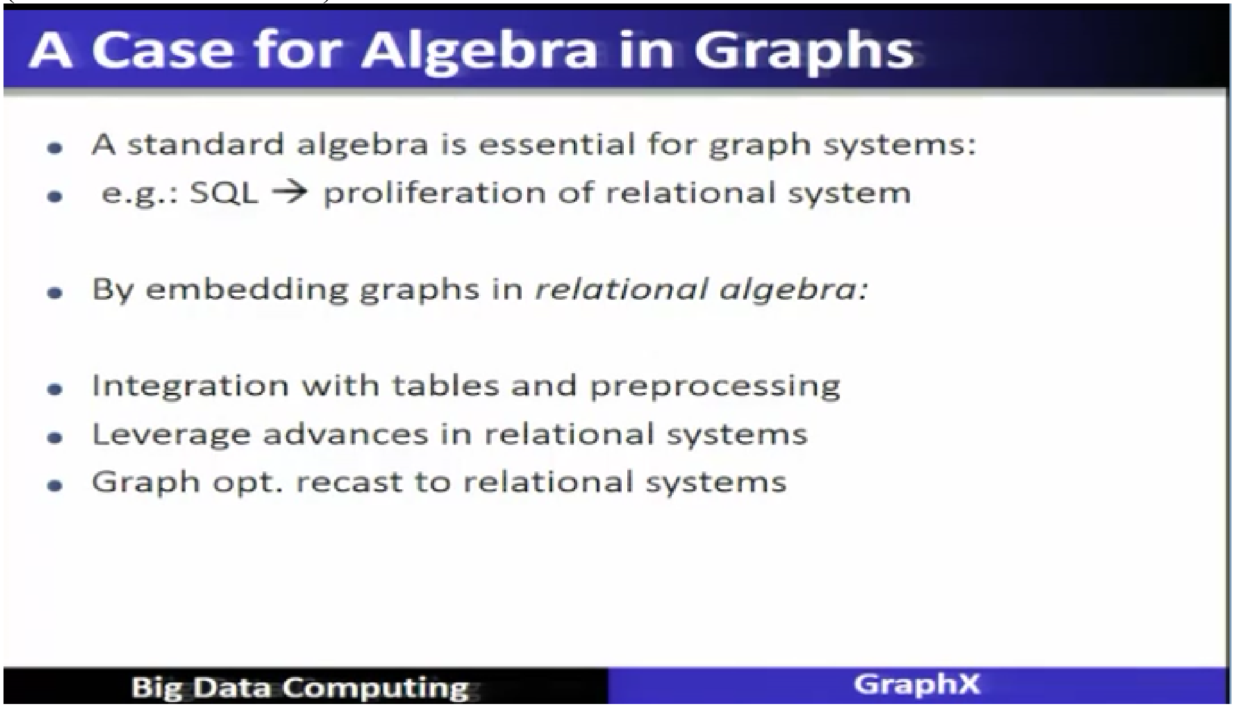
Combines Data-Parallel Graph-Parallel Systems

- Enabling users to easily and efficiently express the entire graph analytics pipeline

Big Data Computing GraphX

so again we have to see that this new API is available and it has given a new system which will combine the data parallel and a graph parallel system.

(Refer Slide Time: 30:17)



A Case for Algebra in Graphs

- A standard algebra is essential for graph systems:
- e.g.: SQL → proliferation of relational system

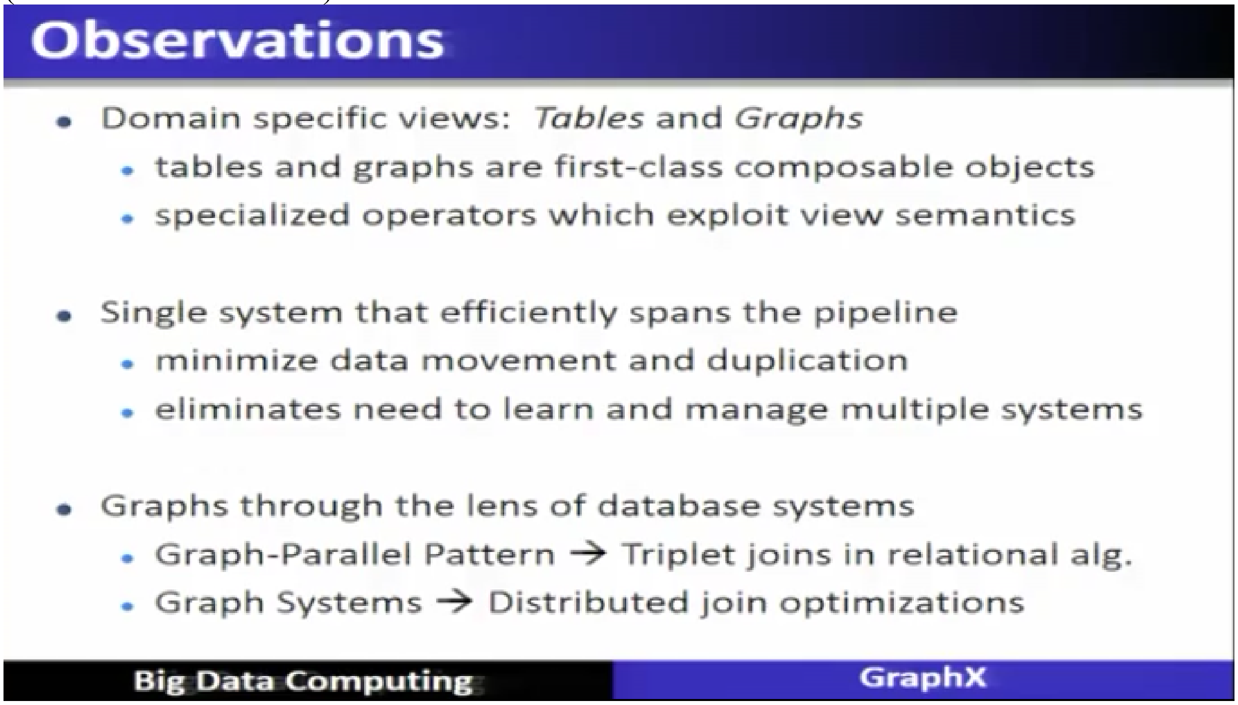
- By embedding graphs in *relational algebra:*

- Integration with tables and preprocessing
- Leverage advances in relational systems
- Graph opt. recast to relational systems

Big Data Computing GraphX

So this particular graphs also is used as the, I mean relational algebra so the queries, sequel queries can also be operated on the graph data structure, so we have already covered the specific views that tables and graphs, the tables and graphs are composable objects and specialized operators can exploit these semantics and also we can efficiently span or build the single pipeline which will minimize the data movement and therefore increase the efficiency,

(Refer Slide Time: 31:00)



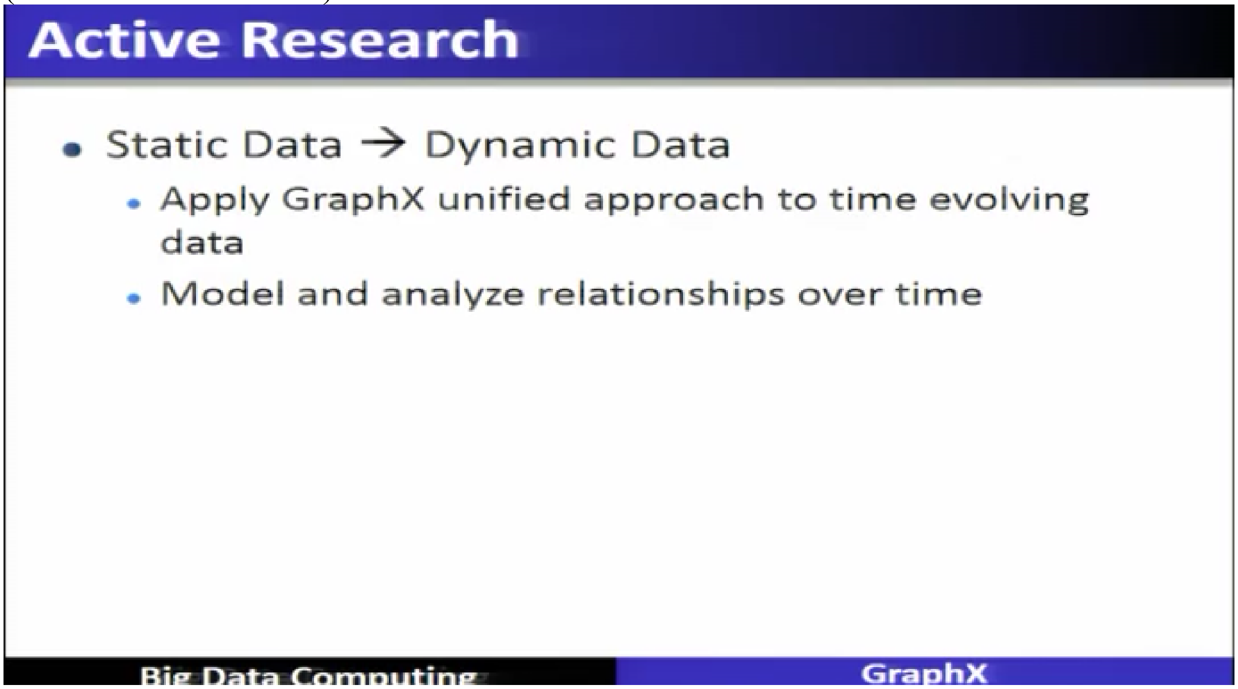
Observations

- Domain specific views: *Tables* and *Graphs*
  - tables and graphs are first-class composable objects
  - specialized operators which exploit view semantics
- Single system that efficiently spans the pipeline
  - minimize data movement and duplication
  - eliminates need to learn and manage multiple systems
- Graphs through the lens of database systems
  - Graph-Parallel Pattern → Triplet joins in relational alg.
  - Graph Systems → Distributed join optimizations

Big Data Computing GraphX

so graph through the lengths of database systems we can also use different relational sequel queries on top of this particular graph and various distributed join operations are also supported.

(Refer Slide Time: 31:13)



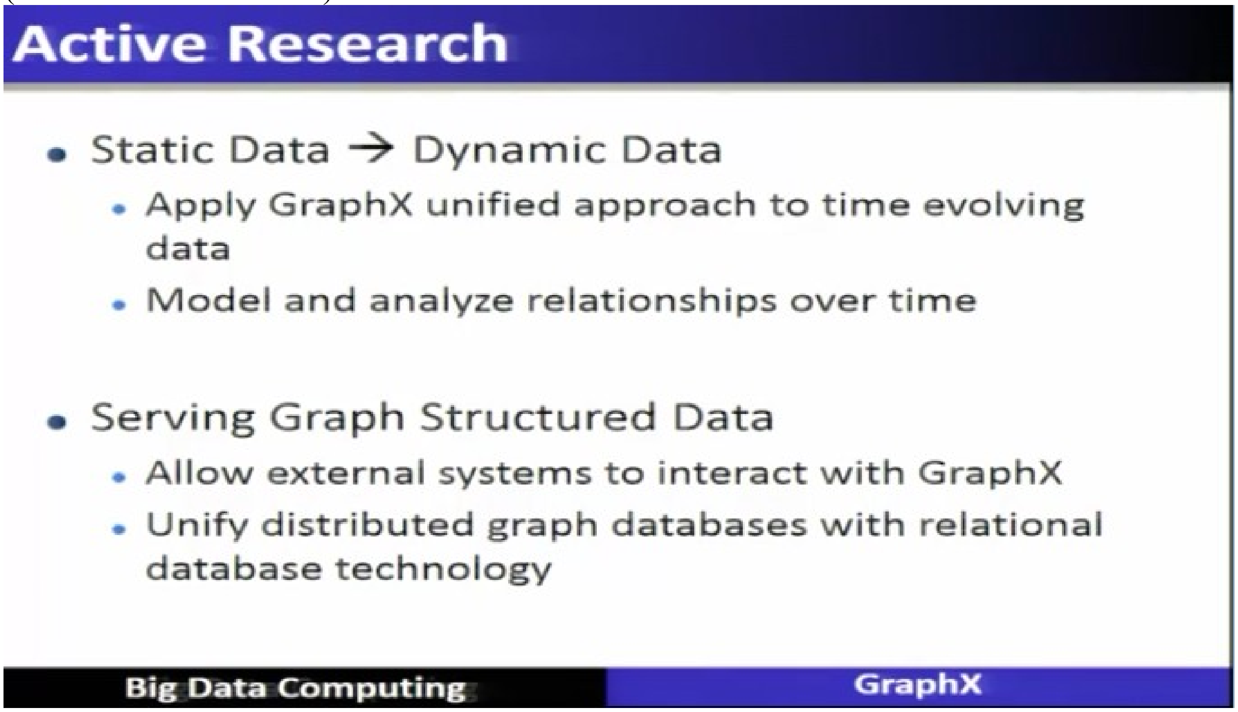
Active Research

- Static Data → Dynamic Data
  - Apply GraphX unified approach to time evolving data
  - Model and analyze relationships over time

Big Data Computing GraphX

So this has become an active area of research, how from static data we can build the dynamic data by applying the GraphX unified approach to a time evolving data model, and analyze the relationship over the time.

(Refer Slide Time: 31:28)



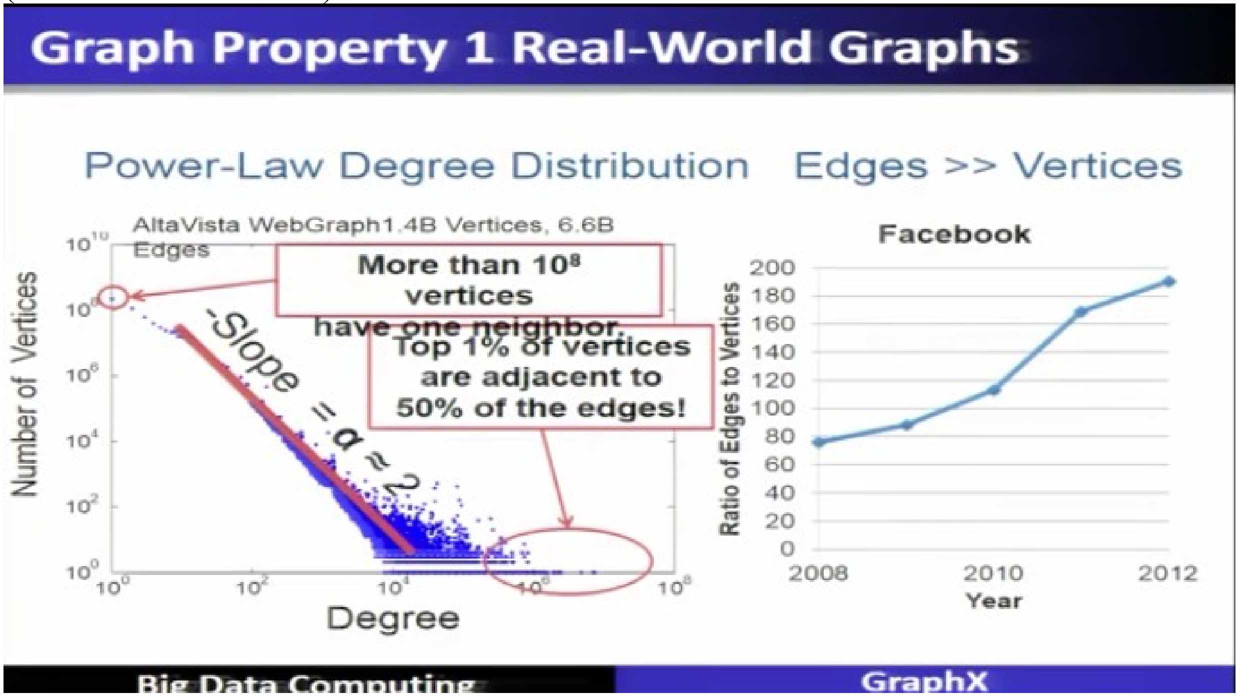
Active Research

- Static Data → Dynamic Data
  - Apply GraphX unified approach to time evolving data
  - Model and analyze relationships over time
- Serving Graph Structured Data
  - Allow external systems to interact with GraphX
  - Unify distributed graph databases with relational database technology

Big Data Computing GraphX

Now serving graph data structured data, serving graph structured data, now allow external system to interact with the graph and unify the distributed graph data based with the relational database technologies,

(Refer Slide Time: 31:41)



Graph Property 1 Real-World Graphs

Power-Law Degree Distribution Edges >> Vertices

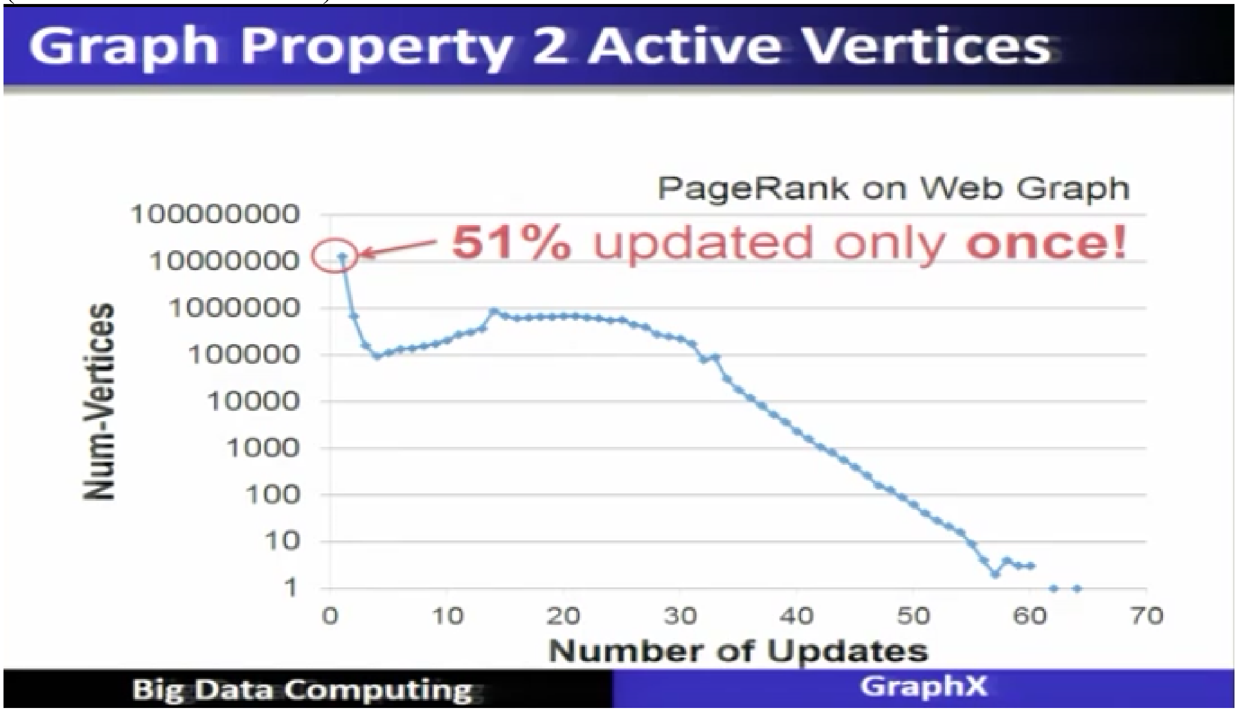
More than 10^8 vertices have one neighbor.

Top 1% of vertices are adjacent to 50% of the edges!

Big Data Computing GraphX

so we can see here that with this support of all these internal implementations or optimizations of a GraphX even the power large distribution of the large graphs can easily be supported even for the Facebook we can see.

(Refer Slide Time: 32:06)



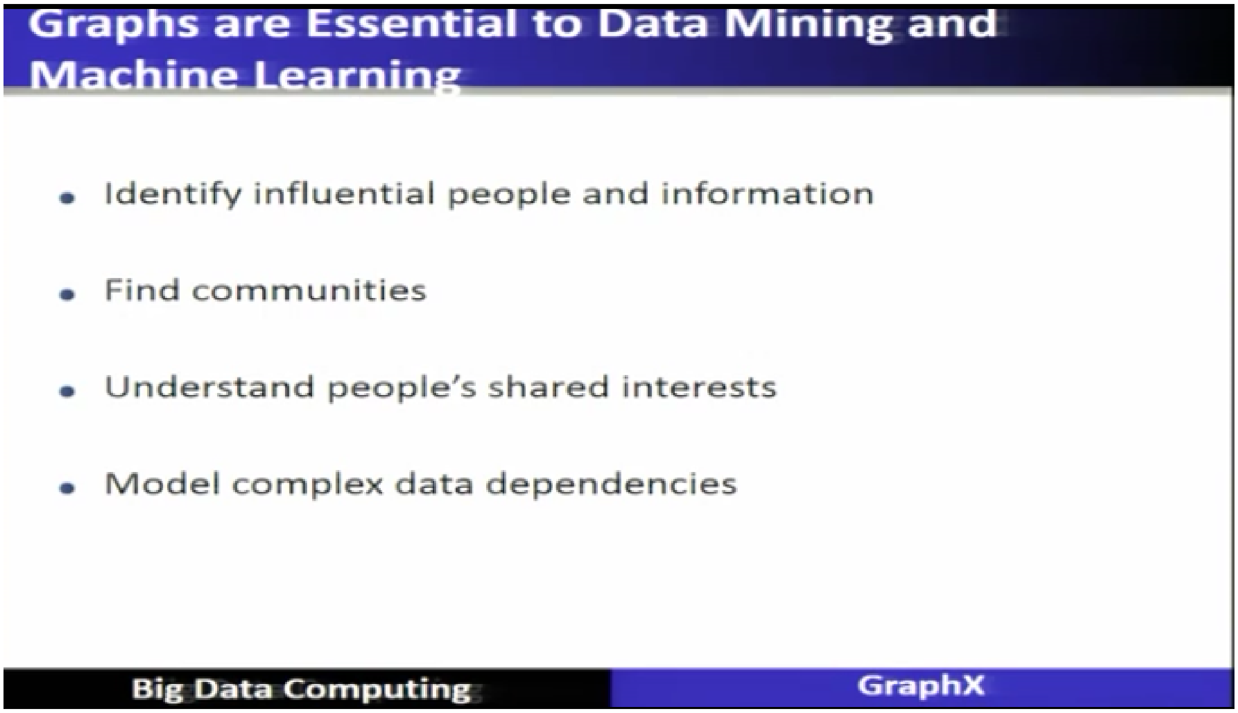
Graph Property 2 Active Vertices

Big Data Computing

GraphX

So here another property is about the active vertices, it's not all the time the entire graph is to be modified, only the active vertices are to be touched upon, hence the active vertices will improve the tracking of active vertices will improve the performance,

(Refer Slide Time: 32:23)



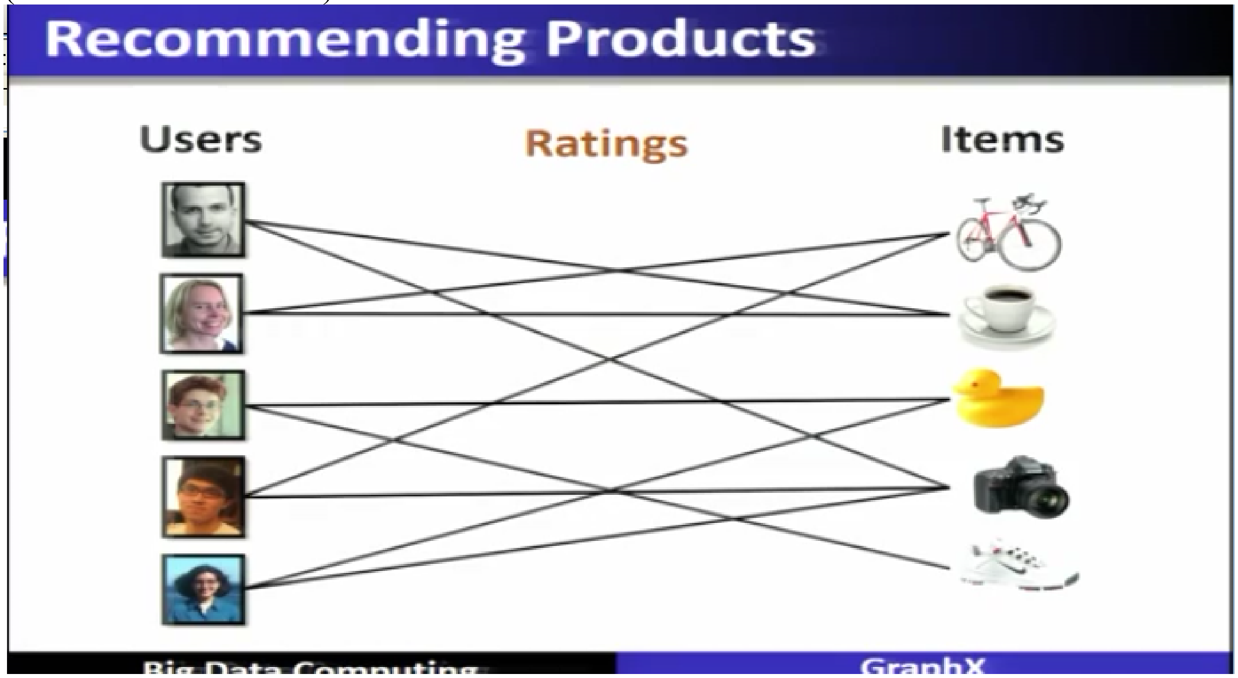
Graphs are Essential to Data Mining and Machine Learning

- Identify influential people and information
- Find communities
- Understand people's shared interests
- Model complex data dependencies

Big Data Computing GraphX

so graph are essentials to the data mining and machine learning, it will identify the influential people information, find the communities, understand people's shared interest and model the complex data dependencies.

(Refer Slide Time: 32:38)

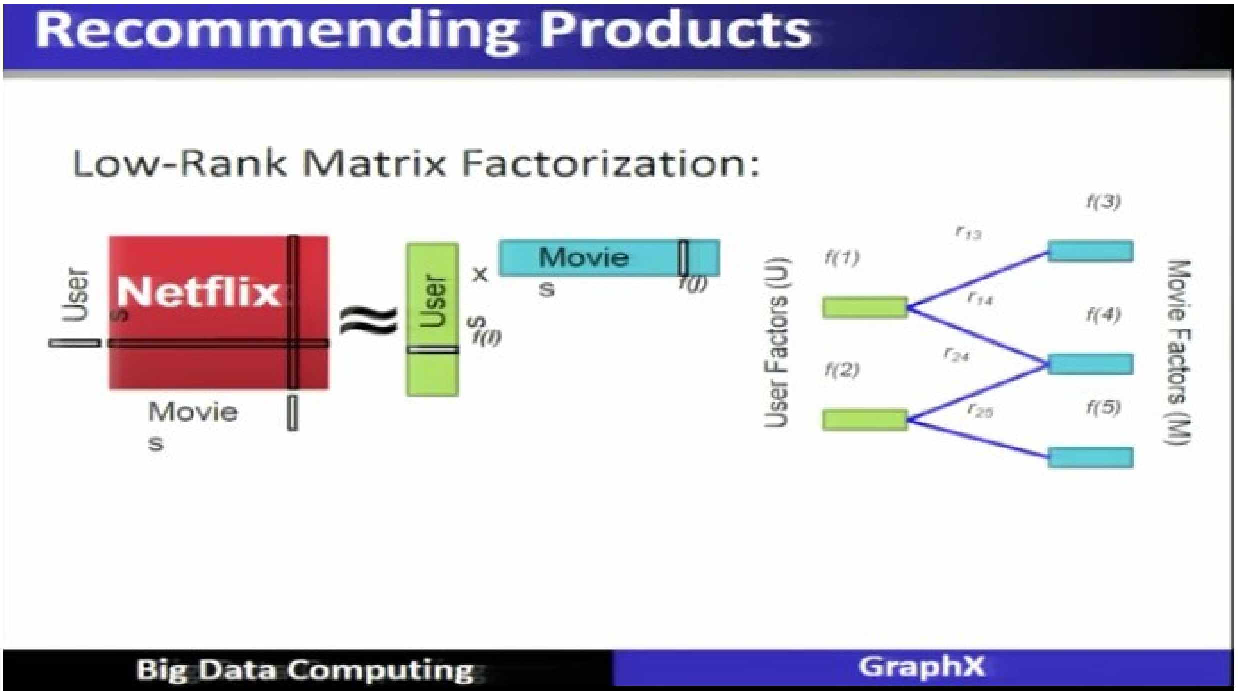


Recommending Products

Big Data Computing GraphX

Now the graphs can be modeled as the bipartite graph or recommending the products that we have seen,

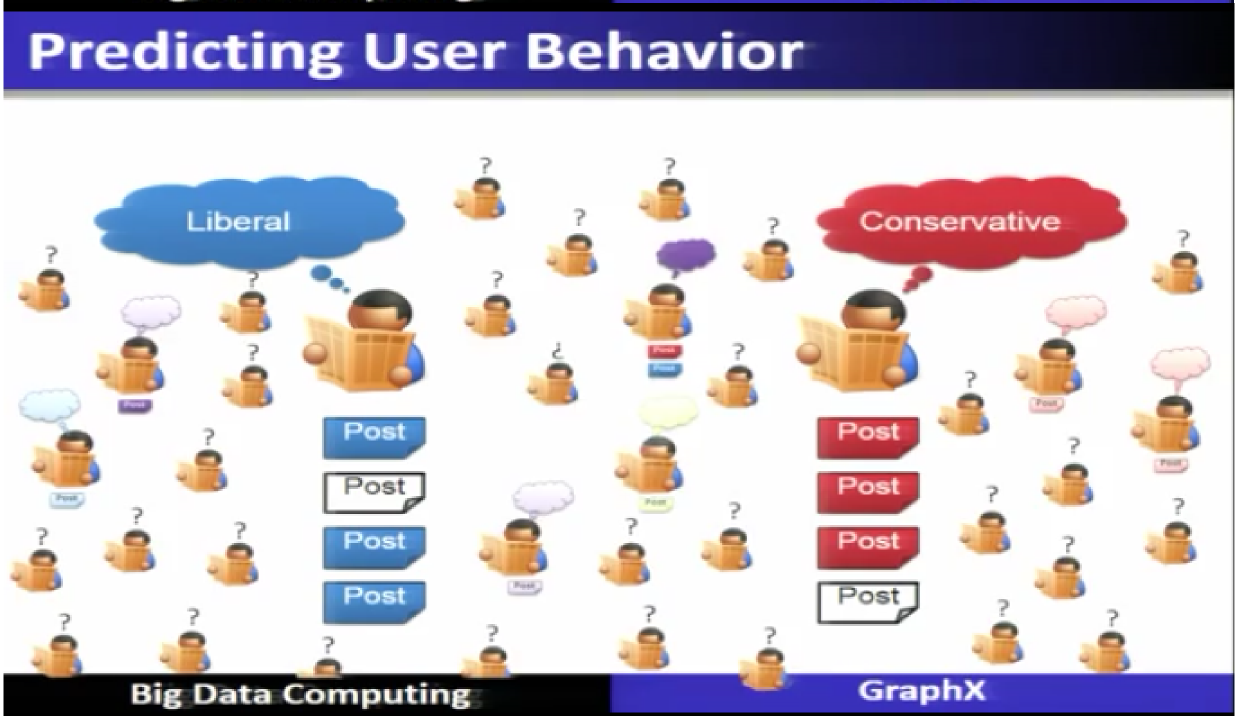
(Refer Slide Time: 32:49)



Recommending Products

Low-Rank Matrix Factorization:

Big Data Computing GraphX

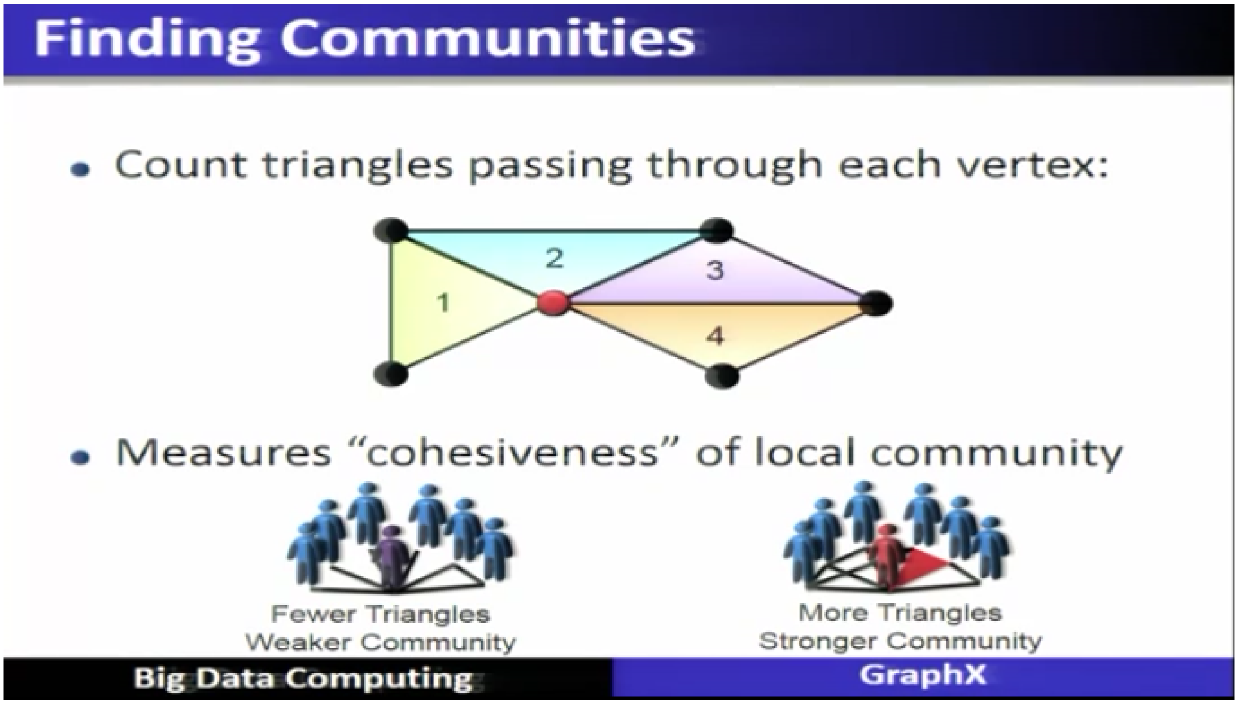


Predicting User Behavior

Big Data Computing GraphX

it is also for recommending the products or the movies using the recommendation engines, for example in the movie case the graph can also be used, similarly to protect the user behavior we can model the problem as the graph and we can do the analysis finding out the conditional random field or belief propagation,

(Refer Slide Time: 33:16)



Finding Communities

- Count triangles passing through each vertex:

- Measures "cohesiveness" of local community

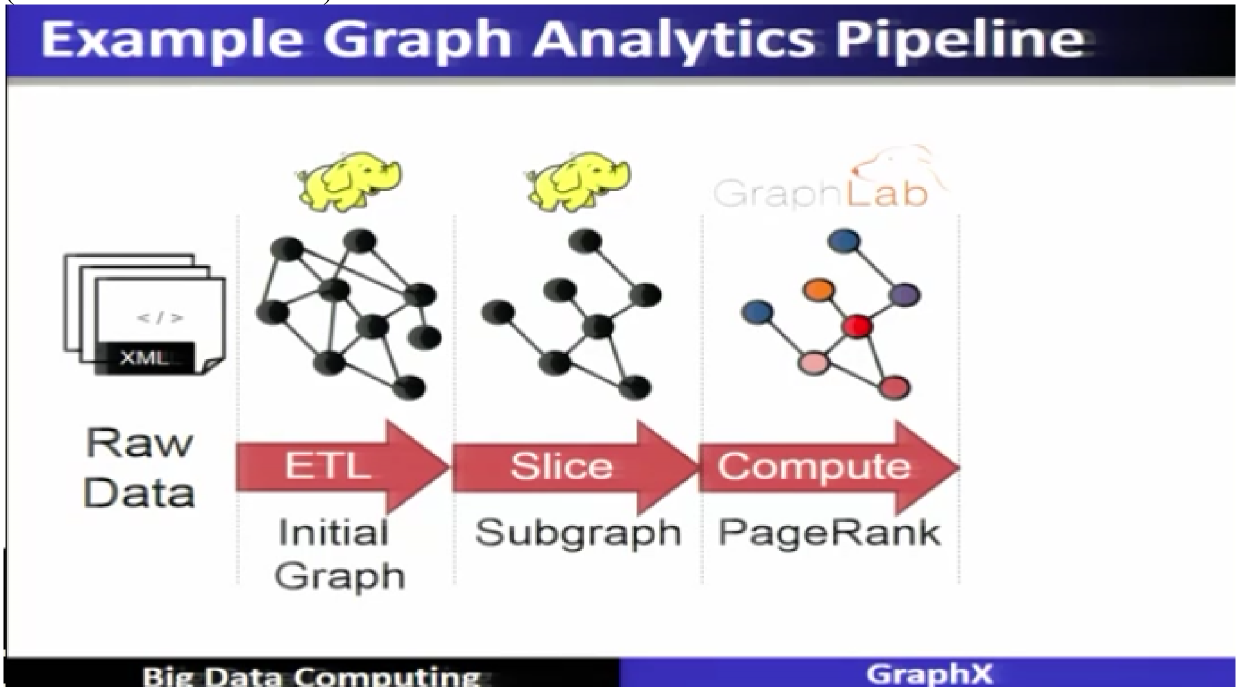
Fewer Triangles
Weaker Community

More Triangles
Stronger Community

Big Data Computing GraphX

finding the communities using triangle count and will also measure the cohesiveness of local community, for example fewer triangles means a weaker community, more triangle means a strong community,

(Refer Slide Time: 33:31)



Example Graph Analytics Pipeline

Raw Data

ETL
Initial Graph

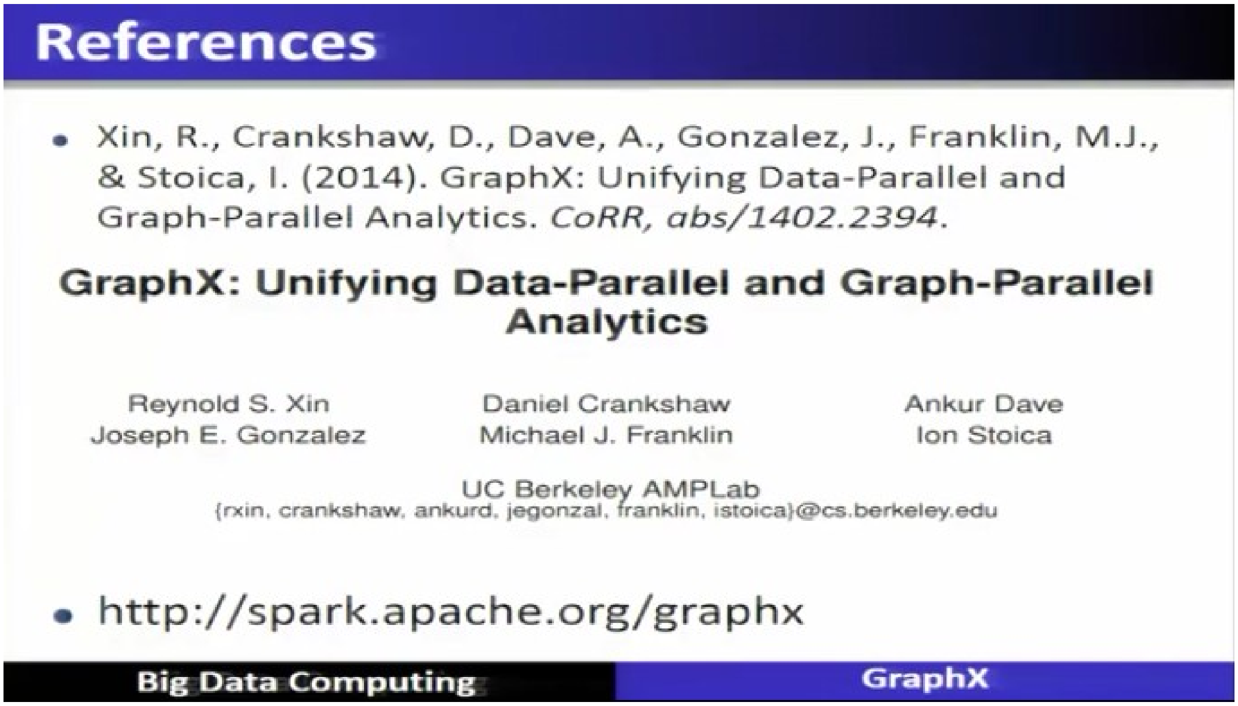
Slice
Subgraph

Compute
PageRank

Big Data Computing GraphX

and also we have seen the building how graph analytics can be achieved using building the pipeline and,

(Refer Slide Time: 33:38)



References

- Xin, R., Crankshaw, D., Dave, A., Gonzalez, J., Franklin, M.J., & Stoica, I. (2014). GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. *CoRR, abs/1402.2394*.

GraphX: Unifying Data-Parallel and Graph-Parallel Analytics

Reynold S. Xin, Daniel Crankshaw, Ankur Dave
Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica

UC Berkeley AMPLab
{rxin, crankshaw, ankurd, jegonzal, franklin, istoica}@cs.berkeley.edu

- http://spark.apache.org/graphx

Big Data Computing GraphX

so these are some of the references and this is the paper of GraphX unifying data parallel and graph parallel analytics.

Acknowledgement

Ministry of Human Resource & Development

Prof. Satyaki Roy

Co-coordinator, IIT Kanpur

NPTEL Team

Sanjay Pal
Bharat Lal
Ashish Singh
Badal Pradhan
Tapobrata Das
K. K. Mishra
Ashutosh Gairola
Puneet Kumar Bajpai
Bhadro Rao
Shikha Gupta
Aradhana Singh Rajawat
Sweta
Nipa Sikdar
Anupam Mishra
Ram Chandra

Manoj Shrivastava
Dilip Tripathi
Padam Shukla
Sharwan K Verma
Sanjay Mishra
Shubham Rawat
Santosh Nayak
Praduyman Singh Chauhan
Mahendra Singh Rawat
Tushar Srivastava
Uzair Siddiqui
Lalty Dutta
Murali Krishnan
Ganesh Rana
Ajay Kanaujia
Ashwani Srivastava
M.L. Benerjee

an IIT Kanpur Production