

## Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Speaker bio: Michael Malak is a Principal Software Engineer at Oracle. He has been working on distributed graph processing and machine learning for over 15 years.

ORACLE

Java

Scala

Python

C/C++

Fortran

Julia

Go

Rust

JavaScript

Fortran

# Finding Graph Isomorphisms in GraphX and GraphFrames

*Michael Malak*



Source code:

<https://www.manning.com/books/spark-graphx-in-action>

<https://gist.github.com/michaelmalak>



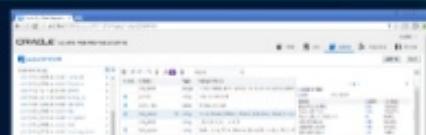
# ORACLE®

*Powered by*  
  
**Spark**

PaaS



DaaS





ORACLE Cloud

## Big Data Preparation

Oracle Big Data Preparation X

slc03hw7003/appu/APP\_JET/#laps/malak20160430

ORACLE Big Data Preparation Cloud Service

Home Job Catalog Knowledge Policies

malak20160430

Add File Done

Transform Script

Identify Col\_0983 as domain name\_size  
Identify Col\_0984 as domain first\_name  
Identify Col\_0981 as domain country  
Identify Col\_0982 as domain country\_02  
Identify Col\_0985 as domain last\_name  
Identify Col\_0988 as domain City  
Identify Col\_0923 as domain occupation  
Identify Col\_0989 as domain state  
Identify Col\_0924 as domain company\_name

Certificate column\_email  
Certificate column\_street\_address

Recommendations

✓ enrich last\_name with name\_first,gender  
✓ zip last\_name to first\_name with delimiter ','

Status Column Type Sample Values

Status	Column	Type	Sample Values
Col_0001	integer	37561; 39304; 42261; 49347; 39134; 43586; 49152; 44985	
gender	string	male;female;F;M	
name_size	string	Mr.; Ms.; Mrs.; Dr.	
first_name	string	James; Robert; William; Charles; John; Mary; David; Joseph	
Col_0006	string	J; R; S; D; C; H; L; A; K; T	
last_name	string	Smith; Jones; White; Johnson; Brown; Richardson; Martin; Parker	
street_address	string	123 Main Street, Anytown, USA	
City	string	Chicago; Los Angeles; Houston; New York; Seattle; Dallas	
state	string	CA; TX; MI; IL; PA; NY; FL; MA; OH; WI	
zipcode	string	48993; 60906; 62110; 28036; 96017; 43985; 13262; 48230	
country	string	US	
country_02	string	United States	
email	string	john.doe@example.com; jdoe@nowhere.org	
Col_0014	string	Hembeida; Goolfitter; Yonne1929; Drivell; Fering; Theplain; In	

Columns Profile

Name	first_name
MinValue	
MaxValue	
Total Rows in File	514
Rows with Data	514
Rows with No Data/Null	0
Duplicated Values	330
Duplicate Values	184
Duplicated Patterns	10
Percent	100.00%
Count	100.00%

Detected Type: TEXT  
Rationale: same\_text\_label  
Winning Domain: same\_first  
Match Accuracy: 100%  
All Matching Domains: [ ]



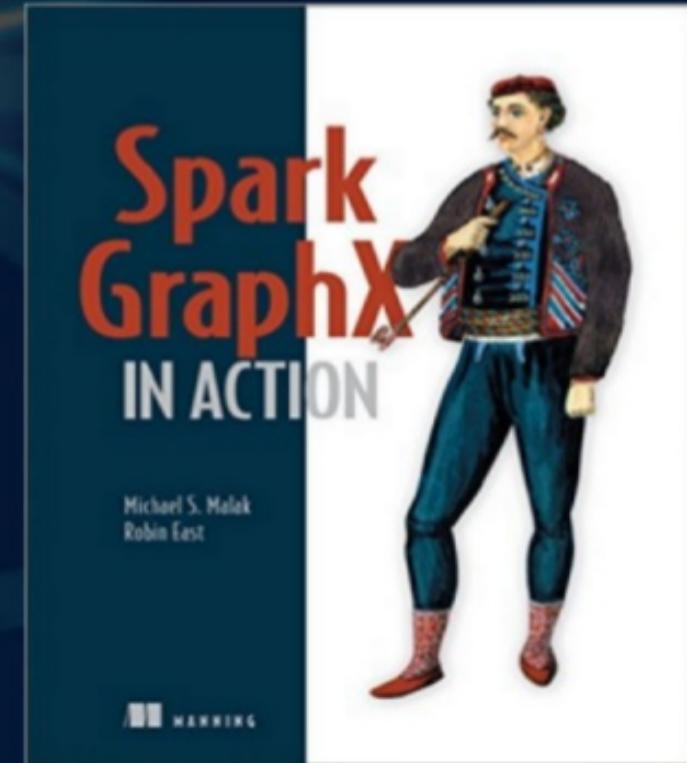
# Spark GraphX

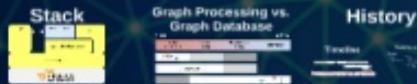
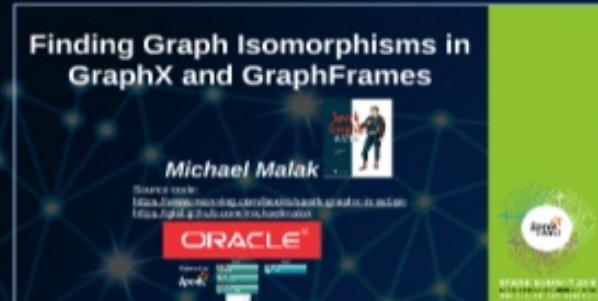
*Michael Malak*

the code:

[www.manning.com/books/spark-graphx-in-action](http://www.manning.com/books/spark-graphx-in-action)

[gist.github.com/michaelmalak](https://gist.github.com/michaelmalak)



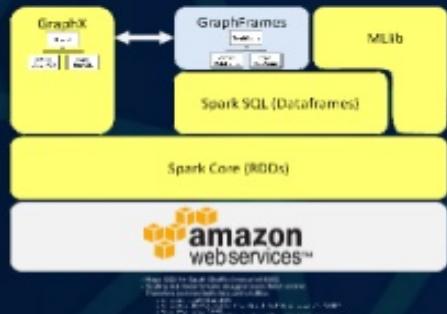


## Isomorphisms



# Spark

# Stack



# Graph Processing vs. Graph Database

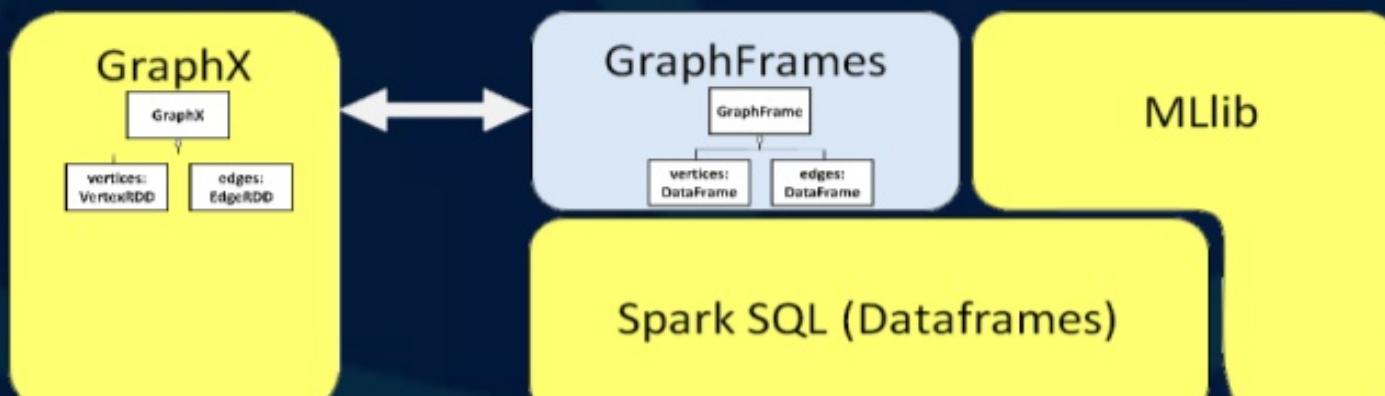
OLAP-like		OLTP-like	
Parallel algorithms	Parallel query	Single-anchor query	Transactions
GraphChi			
GraphFrames			

# History



Features & Performance	
Graph	 <b>GraphFrames</b> <ul style="list-style-type: none"> <li>• Row-based API</li> <li>• 100+ built-in graph operators</li> <li>• DataFrames API</li> <li>• GraphX API</li> <li>• Machine learning</li> </ul>
GraphX	 <b>GraphX</b> <ul style="list-style-type: none"> <li>• Row-based API</li> <li>• 100+ built-in graph operators</li> <li>• DataFrames API</li> <li>• GraphFrames API</li> <li>• Machine learning</li> </ul>

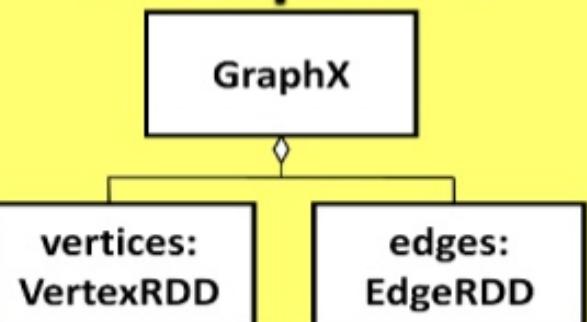
# Stack



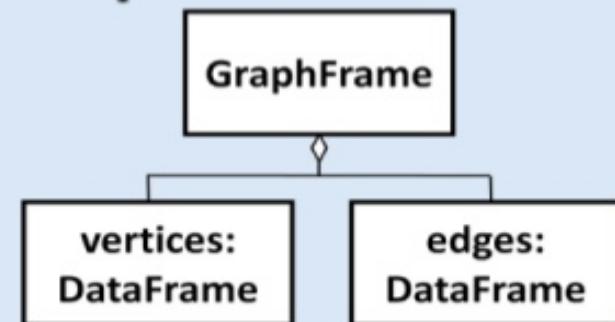
- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.  
Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
```

## GraphX

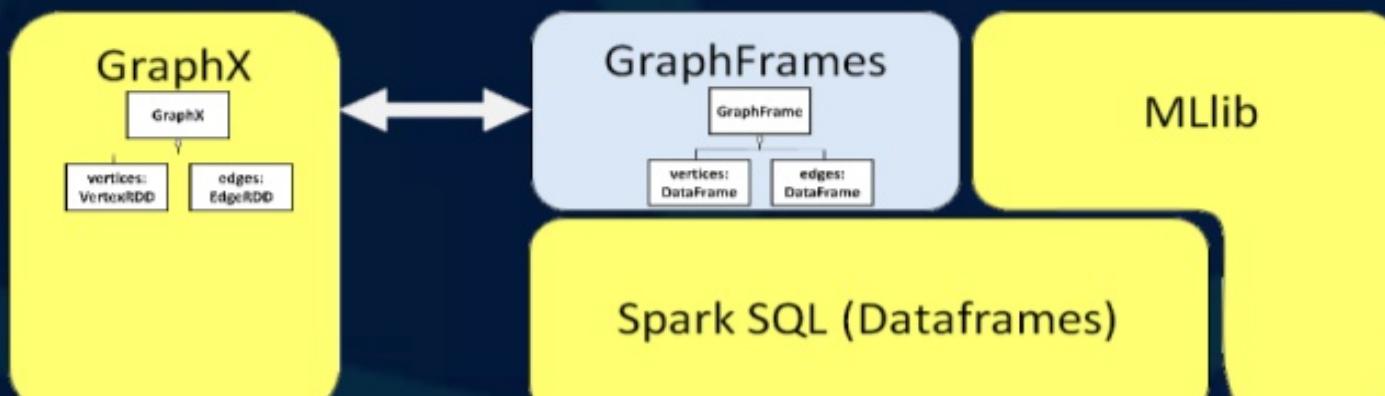


## GraphFrames



Spark SQL (D)

# Stack



- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.  
Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
```

# Graph Processing vs. Graph Database

OLAP-like

OLTP-like

Parallel algorithms	Parallel query	Single-anchor query	Transactions
 GraphX			
	 GraphFrames		
		 neo4j	 TITAN  OrientDB

# History

## Timeline



## Features & Performance

### Bagel

- Features
  - Pregel
  - Property graphs



- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Act on edge properties
- Performance
  - Routing table

### GraphFrames

- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Java, Python bindings
  - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
  - Dataframes
    - Catalyst
    - Tungsten
  - GraphFrames
    - Join Elimination
    - Materialized Views

# Timeline



# Features & Performance

## Bagel

- Features
  - Pregel
  - Property graphs



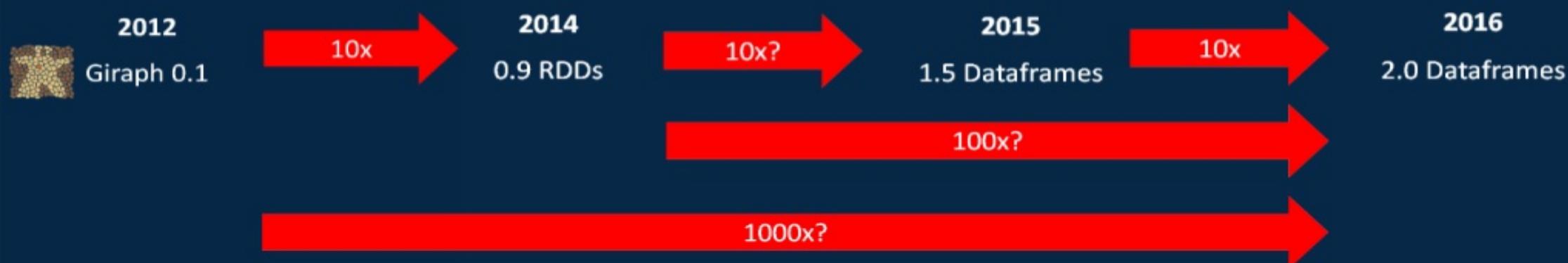
- Features
  - Pregel
  - aggregateMessages
  - True property graphs
  - Act on edge properties
- Performance
  - Routing table

## GraphFrames

- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Java, Python bindings
  - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
  - Dataframes
    - Catalyst
    - Tungsten
  - GraphFrames
    - Join Elimination
    - Materialized Views

# Performance

*Expected from Dataframes (Catalyst & Tungsten)*



*Actual (anecdotal)*



- Before:  
• Spark 2.0 improvements  
• Upcoming Graphframes improvements

# Features & Performance

## Bagel

- Features
  - Pregel
  - Property graphs



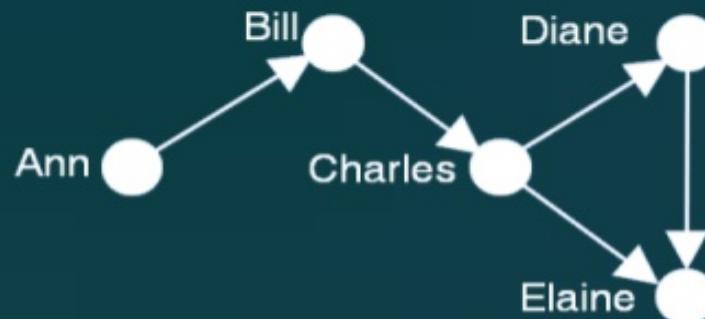
- Features
  - Pregel
  - aggregateMessages
  - True property graphs
  - Act on edge properties
- Performance
  - Routing table

## GraphFrames

- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Java, Python bindings
  - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
  - Dataframes
    - Catalyst
    - Tungsten
  - GraphFrames
    - Join Elimination
    - Materialized Views

# Querying

*Who are the friends of friends of Ann?*



## GraphX

```
val g2 = g.outerJoinVertices(g.aggregateMessages[Int]({  
    ctx => if (ctx.srcAttr == "Ann" &&  
        ctx.attr == "knows") ctx.sendToDst(1,  
        math.max(_,_))((vid, vname, d) =>  
            (vname, d.getOrElse(0)))  
g2.outerJoinVertices(g2.aggregateMessages[Int]({  
    ctx => if (ctx.srcAttr._2 == 1 &&  
        ctx.attr == "knows") ctx.sendToDst(2,  
        math.max(_,_))((vid, vname, d) =>  
            (vname, d.getOrElse(0))).  
    vertices.map(_.._2).filter(_.._2 == 2)  
        .map(_.._1._1).collect  
    })
```

## Neo4j

```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)  
RETURN p
```

## GraphFrames

```
val gf = GraphFrame.fromGraphX(myGraph)  
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")  
    .filter("e1.attr = 'is-friends-with' AND " +  
        "e2.attr = 'is-friends-with' AND " +  
        "u.attr='Ann'")  
    .select("w.attr")  
    .collect  
    .map(_.0.toString)
```



## *Neo4j*

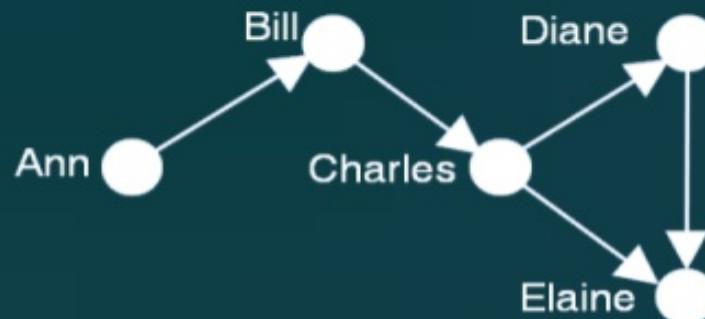
```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)  
RETURN p
```

## *GraphFrames*

```
val gf = GraphFrame.fromGraphX(myGraph)  
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")  
  .filter("e1.attr = 'is-friends-with' AND " +  
         "e2.attr = 'is-friends-with' AND " +  
         "u.attr='Ann'")  
  .select("w.attr")  
  .collect  
  .map(_.0.toString)
```

# Querying

*Who are the friends of friends of Ann?*



## GraphX

```
val g2 = g.outerJoinVertices(g.aggregateMessages[Int]({  
    ctx => if (ctx.srcAttr == "Ann" &&  
        ctx.attr == "knows") ctx.sendToDst(1,  
        math.max(_,_))((vid, vname, d) =>  
            (vname, d.getOrElse(0)))  
g2.outerJoinVertices(g2.aggregateMessages[Int]({  
    ctx => if (ctx.srcAttr._2 == 1 &&  
        ctx.attr == "knows") ctx.sendToDst(2,  
        math.max(_,_))((vid, vname, d) =>  
            (vname, d.getOrElse(0))).  
    vertices.map(_.._2).filter(_.._2 == 2)  
        .map(_.._1._1).collect
```

## Neo4j

```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)  
RETURN p
```

## GraphFrames

```
val gf = GraphFrame.fromGraphX(myGraph)  
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")  
    .filter("e1.attr = 'is-friends-with' AND " +  
        "e2.attr = 'is-friends-with' AND " +  
        "u.attr='Ann'")  
    .select("w.attr")  
    .collect  
    .map(_.0.toString)
```

# Features & Performance

## Bagel

- Features
  - Pregel
  - Property graphs



- Features
  - Pregel
  - aggregateMessages
  - True property graphs
  - Act on edge properties
- Performance
  - Routing table

## GraphFrames

- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Java, Python bindings
  - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
  - Dataframes
    - Catalyst
    - Tungsten
  - GraphFrames
    - Join Elimination
    - Materialized Views

# History

## Timeline



## Features & Performance

### Bagel

- Features
  - Pregel
  - Property graphs



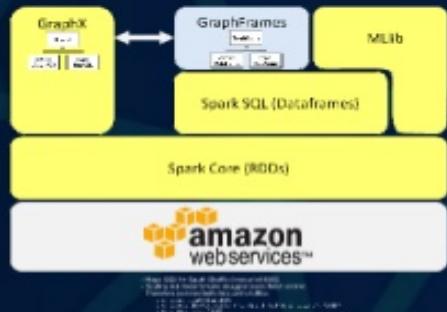
- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Act on edge properties
- Performance
  - Routing table

### GraphFrames

- Features
  - aggregateMessages
  - True property graphs
  - Act on edge properties
  - Java, Python bindings
  - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
  - Dataframes
    - Catalyst
    - Tungsten
  - GraphFrames
    - Join Elimination
    - Materialized Views

# Spark

# Stack



# Graph Processing vs. Graph Database

OLAP-like		OLTP-like	
Parallel algorithms	Parallel query	Single-anchor query	Transactions
GraphChi			
GraphFrames			

# History



• Page 1000 for basic troubleshooting info  
• Page 1001 for more detailed troubleshooting info  
• Page 1002 for more detailed troubleshooting info  
• Page 1003 for more detailed troubleshooting info  
• Page 1004 for more detailed troubleshooting info

# Isomorphisms

## Definition



## Detection



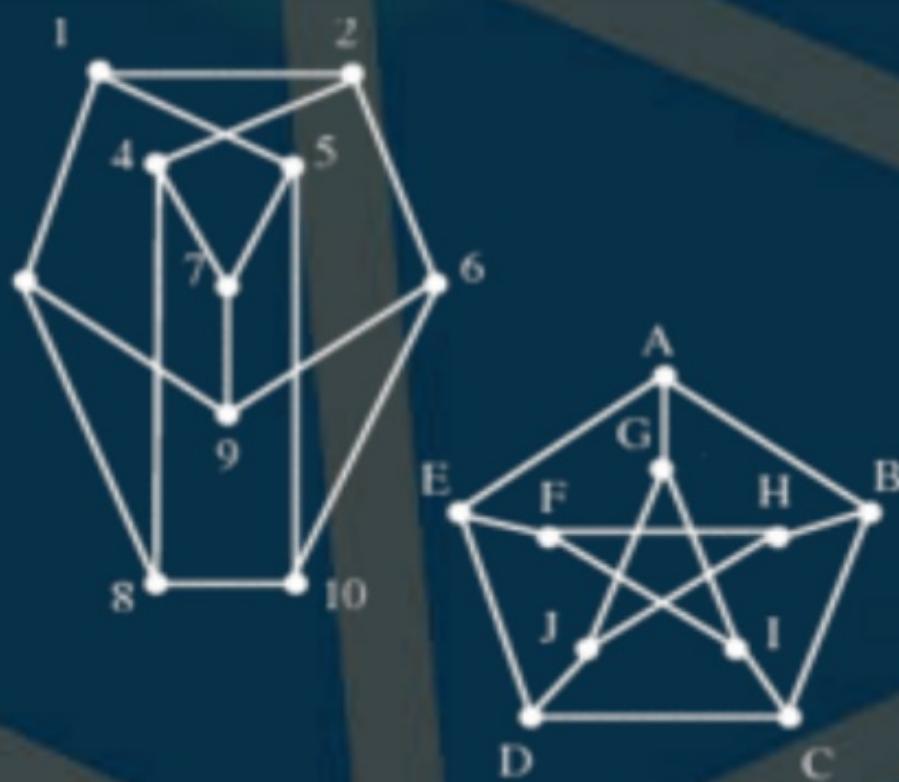
(Not the problem  
we're (I'm) trying  
to solve)

## Finding



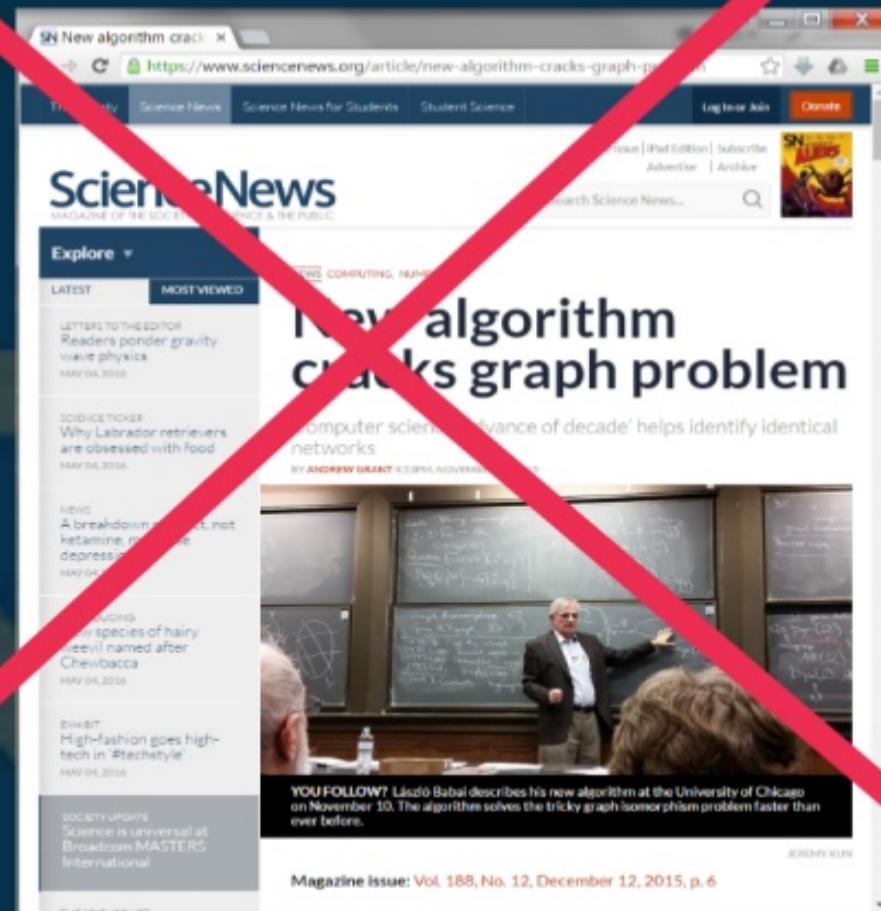
- Rule Mining
- Inductive Logic Programming
  - Network Analysis
  - Social Network Analysis
  - Social Link Analysis

# Definition



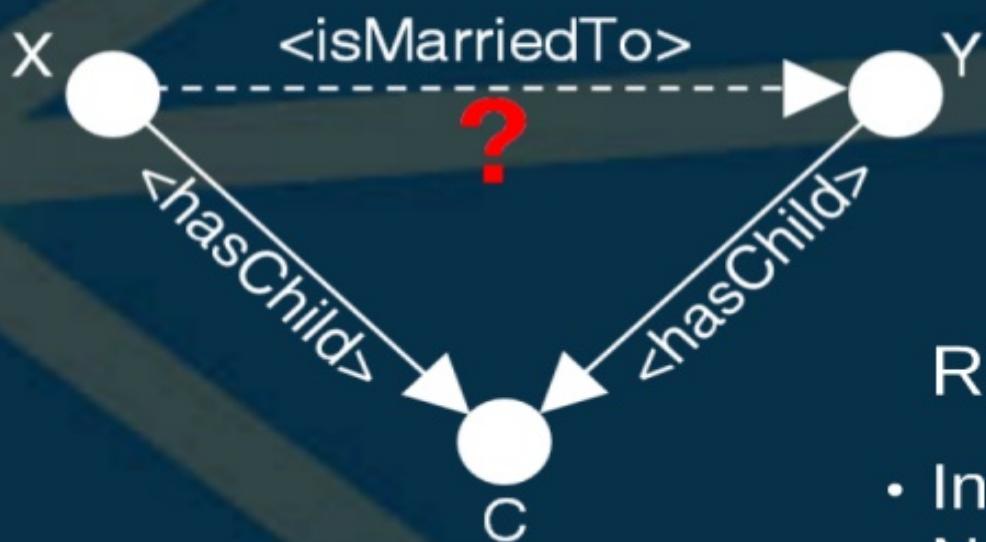
from cs.stonybrook.edu

# Detection



(Not the problem  
we're (I'm) trying  
to solve)

# Finding



## Rule Mining

- Inductive Logic Programming
- Network Analysis
- Social Network Analysis
- Social Link Analysis

# Isomorphisms

## Definition



## Detection



(Not the problem  
we're (I'm) trying  
to solve)

## Finding



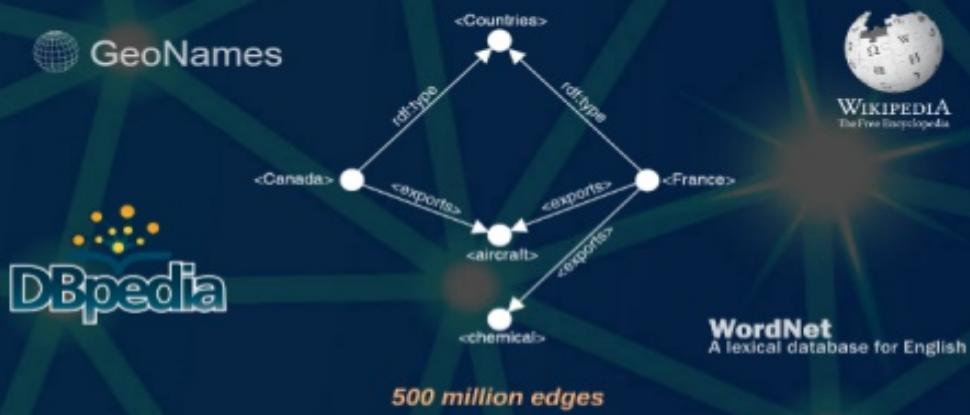
- Rule Mining
- Inductive Logic Programming
  - Network Analysis
  - Social Network Analysis
  - Social Link Analysis



# yago

select knowledge

# Yet Another Great Ontology (YAGO)



## How It's Distributed



# Yet Another Great Ontology (YAGO)

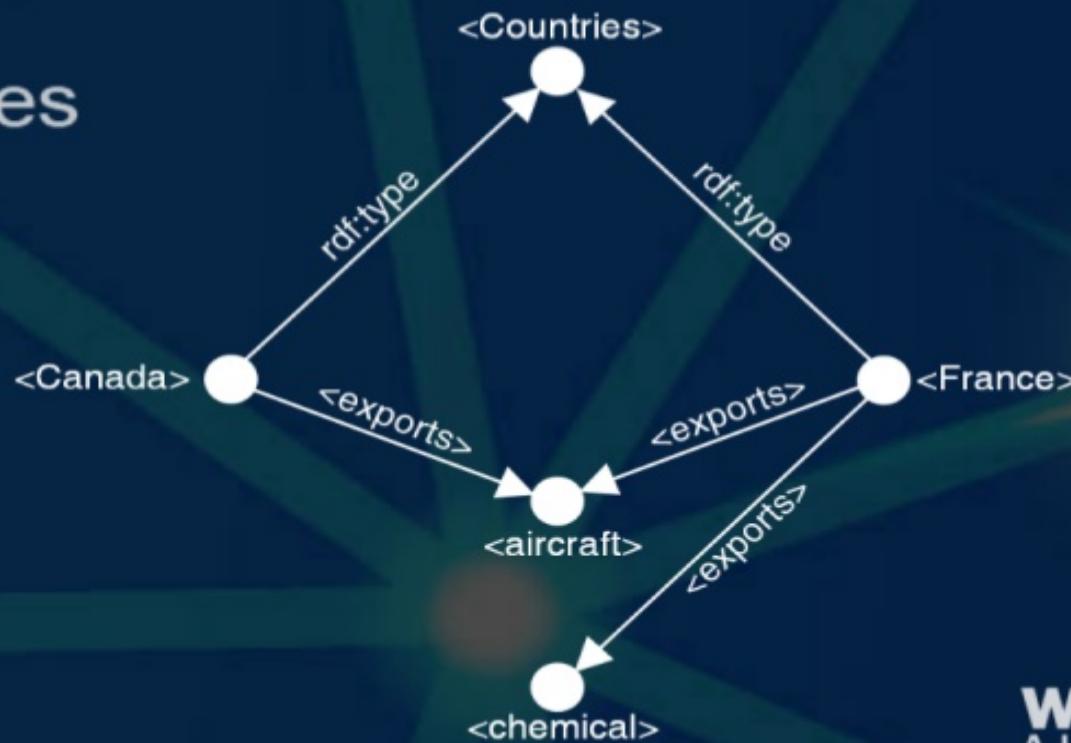
 GeoNames



WIKIPEDIA  
The Free Encyclopedia

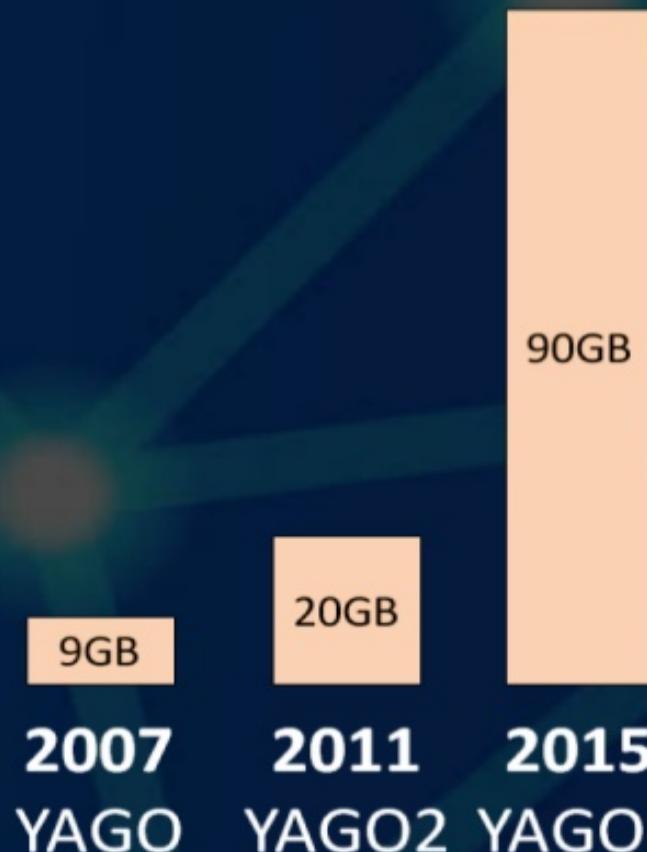
 DBpedia

**WordNet**  
A lexical database for English



*500 million edges*

# How It's Distributed



## 76 .tsv Files in YAGO3

yagoFacts.tsv: 425 MB  
yagoLabels.tsv: 3348 MB  
yagoTaxonomy.tsv: 62 MB

## Functions Needed To Read YAGO

```
readRdf()  
graph()  
mergeGraphs()
```

# 76 .tsv Files in YAGO3

## ***yagoFacts.tsv: 425 MB***

```
<Friedrich_Wilhelm_Joseph_Schelling> <influences> <Alexander_von_Humboldt>
<Jefferson_County,_Texas> <owns> <Jack_Brooks_Regional_Airport>
```

## ***yagoLabels.tsv: 3348 MB***

```
<Francesco_I_Crispo> rdfs:label "Francesco Ier Crispo"@fra
<Monroe_City_Schools> skos:prefLabel "Monroe City Schools"@eng
```

## ***yagoTaxonomy.tsv: 62 MB***

```
<wikicat_Birds_of_Tunisia> rdfs:subClassOf <wordnet_bird_101503061>
<wordnet_knot_113885836> rdfs:subClassOf <wordnet_distorted_shape_113867276>
```

# Functions Needed To Read YAGO

## readRdf()

1. ZipWithIndex: Number the vertices
  2. Join: convert source vertex name to ID
  3. Join: convert destination vertex name to ID

GraphX

```

def resulted(sc:spark.SparkContext, filename: String) = {
    val r = sc.textFile(filename).map(_.split("\\t"))
    val v = r.map(x => (x(0), (x(1), x(2)))).
        distinct.
        zipWithIndex.
        groupByKey.
        map(x => (x._1, (x._2, x._3)))
    .join(v).
    .map(x => (x._1._1, x._1._2, (x._2, x._3, x._4, x._5, x._6))).
    join(v).
    .map(x => new Edges(x._1, x._2, x._3, x._4, x._5, x._6))
}

```

GraphFrames

```

def evalNafConfig(spark:spark.SparkContext, filename:String) = {
    val p = sc.textFile(filename).map(_.split("\\n"))
    val v = naf(np.(_1).union(np.(_3))).distinct.zipWithIndex.map(
        x => Root(x._1).cache
    )
    val stv = StructType(StructField("id", LongType) :: Nil)
    val vnf = v.selectExpr("distinct(concat_ws(',', id), cache")
    val stn = StructType(StructField("id", StringType) :: Nil)
    StructField("subject", StringType) :: Nil)
    StructField("predicate", StringType) :: Nil)
    StructField("object", StringType) :: Nil)
    val ent = sparkContext.createDataFrame(vnf.rdd.map((x,)).map(
        joi => (joi._1, "entity", "entity")
    ).selectExpr("id AS `ent`", "subject", "predicate")
    .join(stn, "subject == $subject")
    .join(stn, "object == $object")
    .selectExpr("src", "id AS dst", "predicate AS attr")
    ).persist(false)
    np.nafConfig(vnf, ent)
}

```

## mergeGraphs()

*GraphFrames*

```

def mainmenuitemselected(StringListStringList) {
    var v = getcurrentmenuitem();
    if(v != null) {
        v.setselected(true);
        v.sethighlighted(true);
    }
}

def mainmenuselected(StringString) {
    if(menu == null) {
        menu = new menubar();
        menu.setname("File");
        menu.additem("New");
        menu.additem("Open");
        menu.additem("Save");
        menu.additem("Exit");
    }
    menu.selectitem(menuitems.indexOf(menuitem));
}

```

GraphFrames

```

def mergeAnnotations(g1, g2, frame) = {
  val vendor = g1.getAnnotations("com.google.cloud.vision.Vendor")
  val labels = g1.getAnnotations("com.google.cloud.vision.Label")
  val regions = g1.getAnnotations("com.google.cloud.vision.Region")
  val structures = g1.getAnnotations("com.google.cloud.vision.Structure")
  val entities = g1.getAnnotations("com.google.cloud.vision.Entity")
  val types = g1.getAnnotations("com.google.cloud.vision.Type")
  val typesMap = types.asMap()
  new AnnotatorType[Annotator](Vendor, labels, regions, structures,
    structures.asMap(), String(type))
}

def copyAnnotationsFromProto(g1: GraphFrame, g2: GraphFrame) = {
  g1.vertices
    .map(_.id)
    .foreach(id => {
      val vertex = g1.vertex(id)
      val annotations = vertex.annotations
      val vertex2 = g2.vertex(id)
      vertex2.annotations = annotations
    })
}

def copyAnnotationsFromGraphFrame(g1: GraphFrame, g2: GraphFrame) = {
  g1.vertices
    .map(_.id)
    .foreach(id => {
      val vertex = g1.vertex(id)
      val annotations = vertex.annotations
      val vertex2 = g2.vertex(id)
      vertex2.annotations = annotations
    })
}
}

```

# readRdf()

1. ZipWithIndex: Number the vertices
2. Join: convert source vertex name to ID
3. Join: convert destination vertex name to ID

## GraphX

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
    val r = sc.textFile(filename).map(_.split("\t"))  
    val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex  
    Graph(v.map(_.swap),  
        r.map(x => (x(1),(x(2),x(3))))  
        .join(v)  
        .map(x => (x._2._1._2,(x._2._2,x._2._1._1)))  
        .join(v)  
        .map(x => new Edge(x._2._1._1, x._2._2, x._2._1._2)))  
}
```

## GraphFrames

```
def readRdfDf(sc:org.apache.spark.SparkContext, filename:String) = {  
    val r = sc.textFile(filename).map(_.split("\t"))  
    val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex.map(  
        x => Row(x._2,x._1)).cache  
    val stv = StructType(StructField("id",LongType) ::  
                        StructField("attr",StringType) :: Nil)  
    val vdf = sqlContext.createDataFrame(v,stv).cache  
    val str = StructType(StructField("rdfId",StringType) ::  
                        StructField("subject",StringType) ::  
                        StructField("predicate",StringType) ::  
                        StructField("object",StringType) :: Nil)  
    val edf = sqlContext.createDataFrame(r.map(Row.fromSeq(_)),str)  
        .join(vdf, $"subject" === $"attr")  
        .selectExpr("id AS src", "predicate", "object")  
        .join(vdf, $"object" === $"attr")  
        .selectExpr("src", "id AS dst", "predicate AS attr")  
    v.unpersist(false)  
    GraphFrame(vdf,edf)  
}
```

# mergeGraphs()

## GraphX

```
def mergeGraphs(g1:Graph[String,String], g2:Graph[String,String]) = {  
    val v = g1.vertices.map(_._2).union(g2.vertices.map(_._2)).distinct  
        .zipWithIndex  
    def edgesWithNewVertexIds(g:Graph[String,String]) =  
        g.triplets  
            .map(et => (et.srcAttr, (et.attr,et.dstAttr)))  
            .join(v)  
            .map(x => (x._2._1._2, (x._2._2,x._2._1._1)))  
            .join(v)  
            .map(x => new Edge(x._2._1._1,x._2._2,x._2._1._2))  
    Graph(v.map(_.swap),  
          edgesWithNewVertexIds(g1).union(edgesWithNewVertexIds(g2)))  
}
```

## GraphFrames

```
def mergeGraphs(g1:GraphFrame, g2:GraphFrame) = {  
    val vunion = g1.vertices.sqlContext.createDataFrame(  
        g1.vertices.select($"attr").unionAll(g2.vertices.select($"attr"))  
            .distinct.rdd.map(_(0))  
            .zipWithIndex.map(t => Row.fromTuple(t.swap)),  
        new StructType(Array(StructField("id",LongType),  
                           StructField("attr",StringType)))  
    ).cache  
  
    def edgesWithNewVertexIds(gf:GraphFrame) = {  
        gf.triplets  
            .join(vunion, $"src.attr" === $"attr")  
            .selectExpr("id AS src","edge.attr AS edgeattr",  
                      "dst.attr AS dstattr")  
            .join(vunion, $"dstattr" === $"attr")  
            .selectExpr("src","id AS dst","edgeattr AS attr")  
    }  
  
    GraphFrame(vunion, edgesWithNewVertexIds(g1).unionAll(  
        edgesWithNewVertexIds(g2)))  
}
```

# Functions Needed To Read YAGO

## readRdf()

1. ZipWithIndex: Number the vertices
2. Join: convert source vertex name to ID
3. Join: convert destination vertex name to ID

### GraphX

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
    val r = sc.textfile(filename).map(_.split("\t"))  
    val v = r.map(_(0)).union(r.map(_(3))).distinct.zipWithIndex  
    GraphVX.map(_.name,  
        r.map(x => (x(0),(x(2),x(3))))  
        .join(v)  
        .map(x => (x._2._1,_2,(x._2._2,x._2._3,_3)))  
        .join(v)  
        .map(x => new Edge(x._2._1,_1, x._2._2, x._2._3,_3))  
    )  
}
```

### GraphFrames

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
    val r = sc.textfile(filename).map(_.split("\t"))  
    val v = r.map(_(0)).union(r.map(_(3))).distinct.zipWithIndex.map(  
        x => Row(x._2,_1)).cache  
    val sv = StructType(StructField("id",StringType)::Nil)  
    val svf = StructField("id",StringType)::Nil  
    val sm = StructType(StructField("id",StringType)::Nil)  
    val smf = StructField("subject",StringType)::Nil  
    val mp = StructField("predicate",StringType)::Nil  
    val ob = StructField("object",StringType)::Nil  
    val ed = sqlContext.createDataFrame(r.map(_.map(Seq(_))),sv)  
        .join(v, "id AS src")  
        .selectExpr("id AS src", "predicate", "object")  
        .join(v, "id AS object")  
        .selectExpr("src", "id AS dst", "predicate AS attr")  
    v.persist(false)  
    GraphFrame(ed,vf)  
}
```

## mergeGraphs()

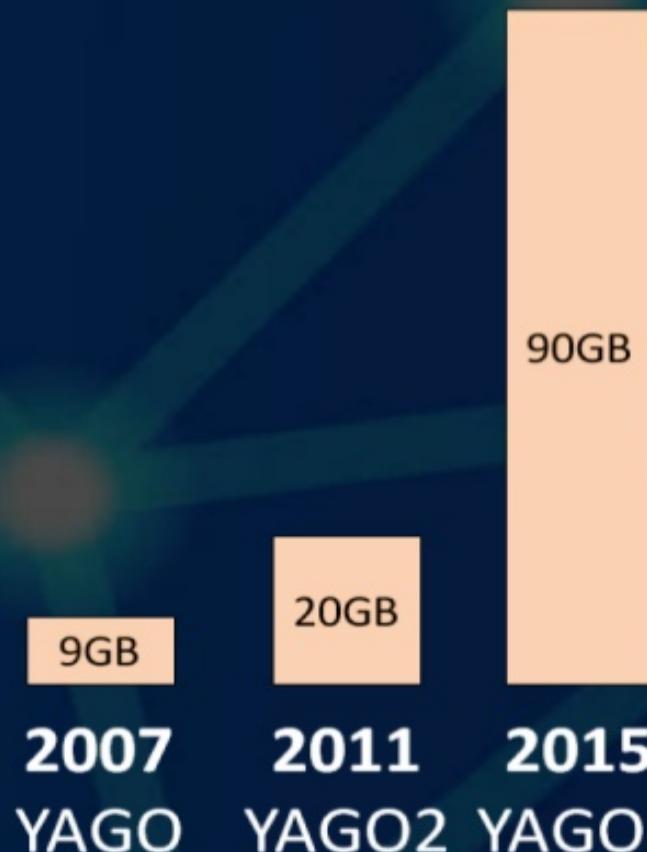
### GraphX

```
def mergeGraphs(g1:Graph[VertexId,Edge], g2:Graph[VertexId,Edge]) = {  
    val v1 = g1.vertices.collect.map{v1=>(v1.id,v1)}    val v2 = g2.vertices.collect.map{v2=>(v2.id,v2)}    val edges1 = g1.edges.collect.map{e1=>(e1.src,e1.dst,e1.attr)}    val edges2 = g2.edges.collect.map{e2=>(e2.src,e2.dst,e2.attr)}    edges1 ++ edges2  
}
```

### GraphFrames

```
def mergeGraphs(g1:GraphFrame, g2:GraphFrame) = {  
    val v1 = g1.vertices.collect.map{v1=>(v1.id,v1)}    val v2 = g2.vertices.collect.map{v2=>(v2.id,v2)}    val edges1 = g1.edges.collect.map{e1=>(e1.src,e1.dst,e1.attr)}    val edges2 = g2.edges.collect.map{e2=>(e2.src,e2.dst,e2.attr)}    edges1 ++ edges2  
}  
  
def edgesWithNewVertices(g1:GraphFrame) = {  
    g1.edges  
    .join(v1, "src AS src" ++ "id AS id")  
    .join(v1, "dst AS dst" ++ "id AS id")  
    .join(v1, "attr AS attr" ++ "id AS id")  
    .selectExpr("src","id AS dst","label AS id","attr AS attr")  
}  
  
GraphFrame(v1, edgesWithNewVertices(g1).unionAll(  
    edgesWithNewVertices(g2)))  
}
```

# How It's Distributed



## 76 .tsv Files in YAGO3

yagoFacts.tsv: 425 MB  
yagoLabels.tsv: 3348 MB  
yagoTaxonomy.tsv: 62 MB

## Functions Needed To Read YAGO

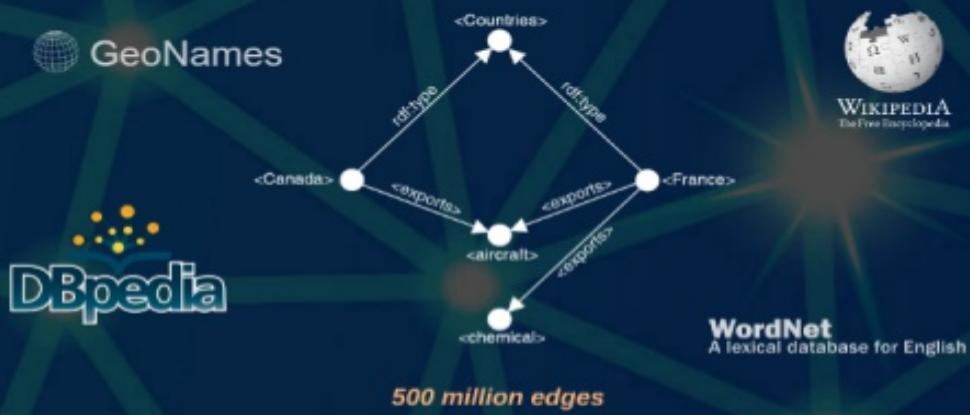
```
readRdf()  
graph()  
mergeGraphs()
```



# yago

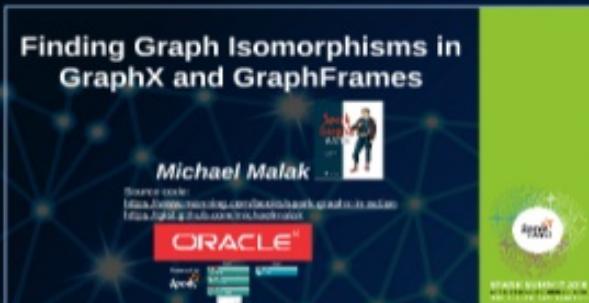
select knowledge

## Yet Another Great Ontology (YAGO)



## How It's Distributed





Spark



## Isomorphisms

### Definition



### Finding

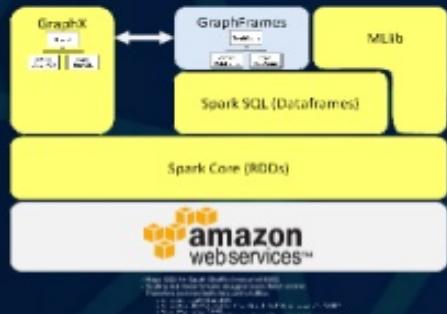
### Isomorphism

yago  
select knowledge



# Spark

# Stack



# Graph Processing vs. Graph Database

OLAP-like		OLTP-like	
Parallel algorithms	Parallel query	Single-anchor query	Transactions
GraphChi			
GraphFrames			

# History

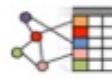


Features & Performance	
<b>Graph</b>	 GraphML
- Accurate	- High performance
- Large	- High memory efficiency
- Efficient graph- processing	- Direct access to any vertex or edge
- Persistence	- Subgraphs
- Hierarchical	- GraphFrames
- Partitioned	- Fault-tolerant
- Streaming	- High performance
- Machine learning	- High memory efficiency
- Graph neural networks	- Direct access to any vertex or edge
- Deep learning	- Subgraphs
- Graph embeddings	- GraphFrames
- Graph mining	- Fault-tolerant
- Graph clustering	- High performance
- Graph visualization	- High memory efficiency
- Graph search	- Direct access to any vertex or edge
- Graph mining	- Subgraphs
- Graph clustering	- Fault-tolerant
- Graph visualization	- High performance
- Graph search	- High memory efficiency

# Graph Processing vs. Graph Database

OLAP-like

OLTP-like

Parallel algorithms	Parallel query	Single-anchor query	Transactions
 GraphX			
	 GraphFrames		
		 neo4j	 TITAN  OrientDB

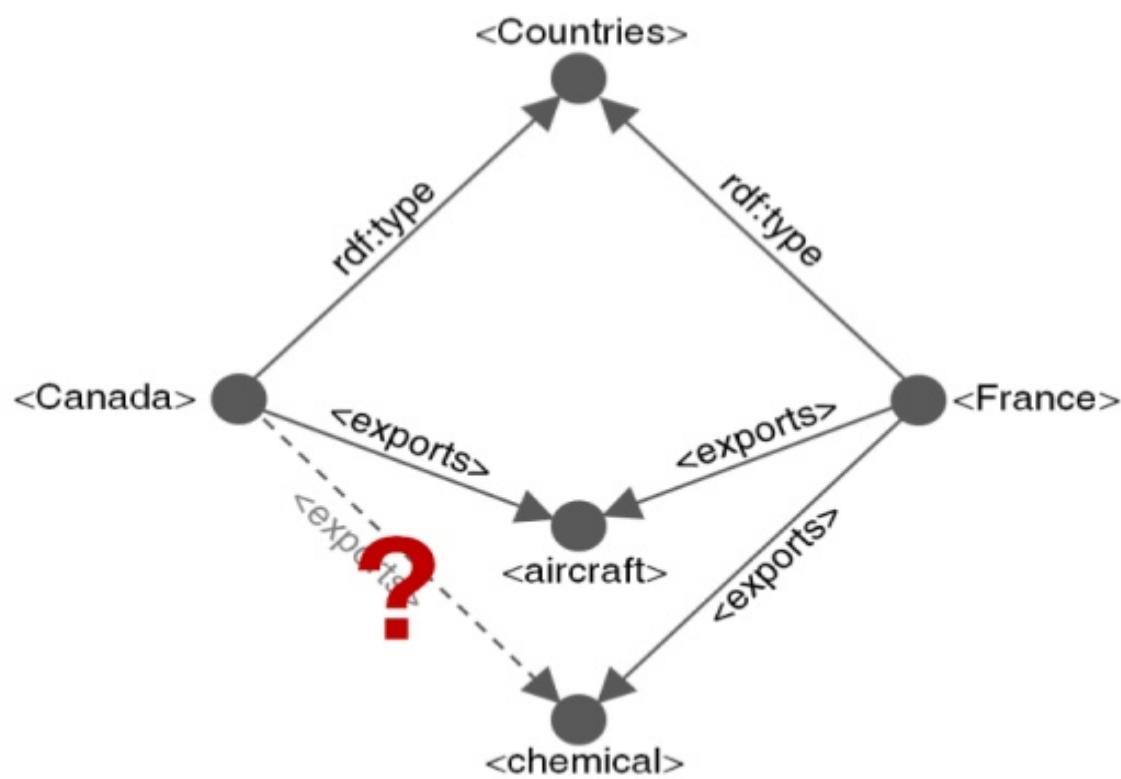


# GraphX

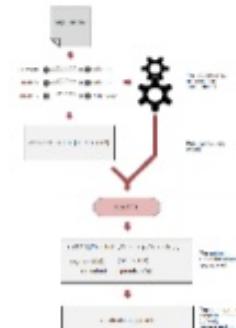
Country Exports Example



# Country Exports Example



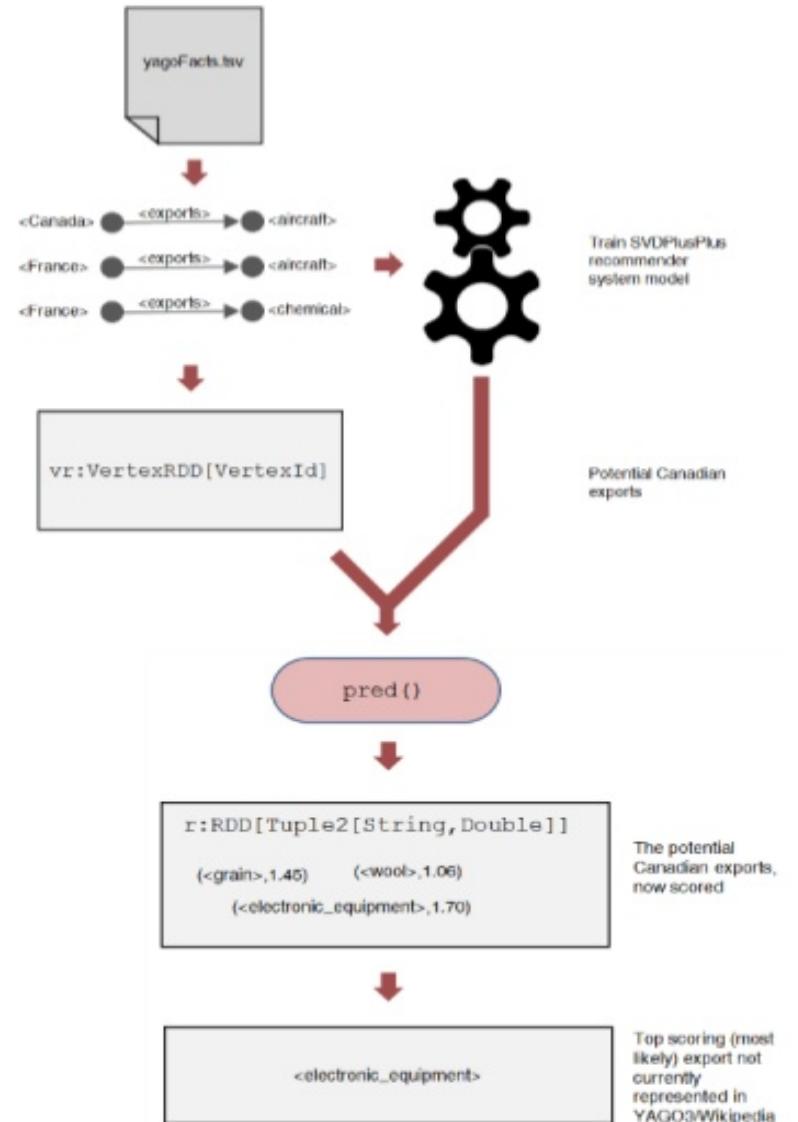
## Dataflow

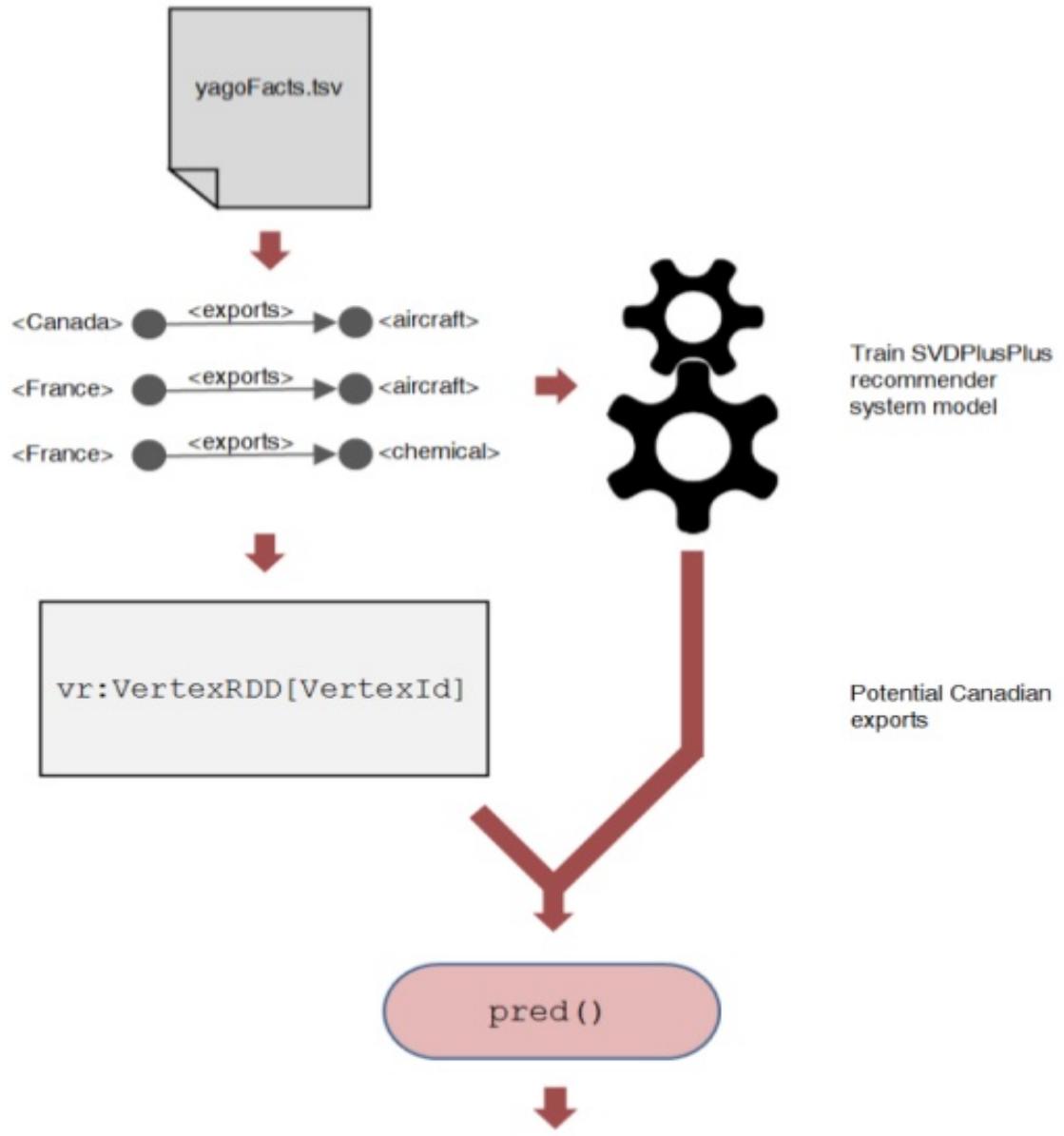


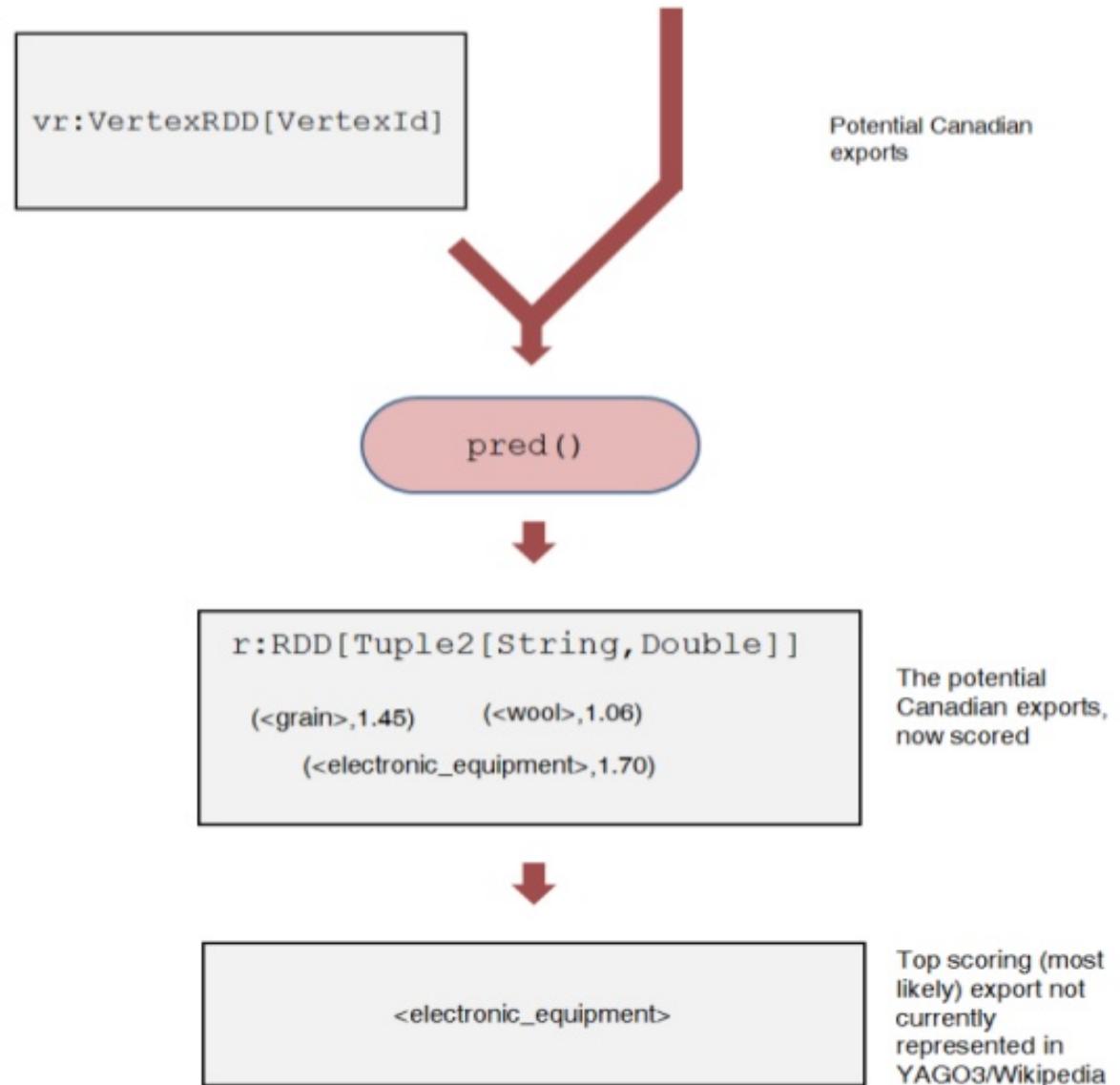
## Results

```
at.westlogic.tltest_1...1    maxdelay=100000  
maxdelay=100000
```

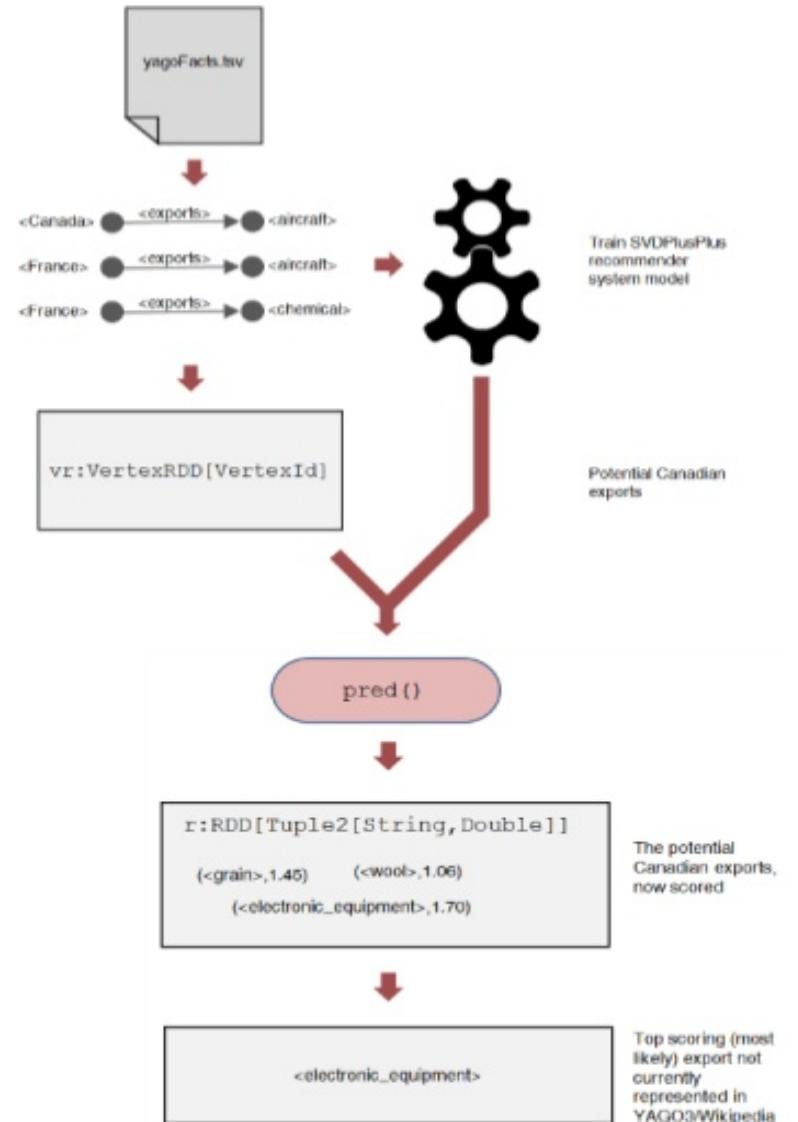
# Dataflow







# Dataflow



# Results

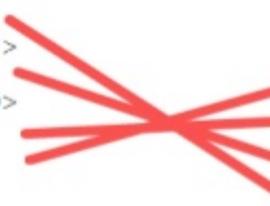
```
val cid = gf.vertices.filter(_.value == "<Canada>").first._1
val r = vr.map(v => (v, pred(vm, mean, cid, v)))

val maxKey = r.max()(new Ordering[Tuple2[VertexId, Double]]() {
    override def compare(x: (VertexId, Double), y: (VertexId, Double)): Int =
        Ordering[Double].compare(x.value, y.value)
}) ._1

gf.vertices.filter(_.value == maxKey).collect

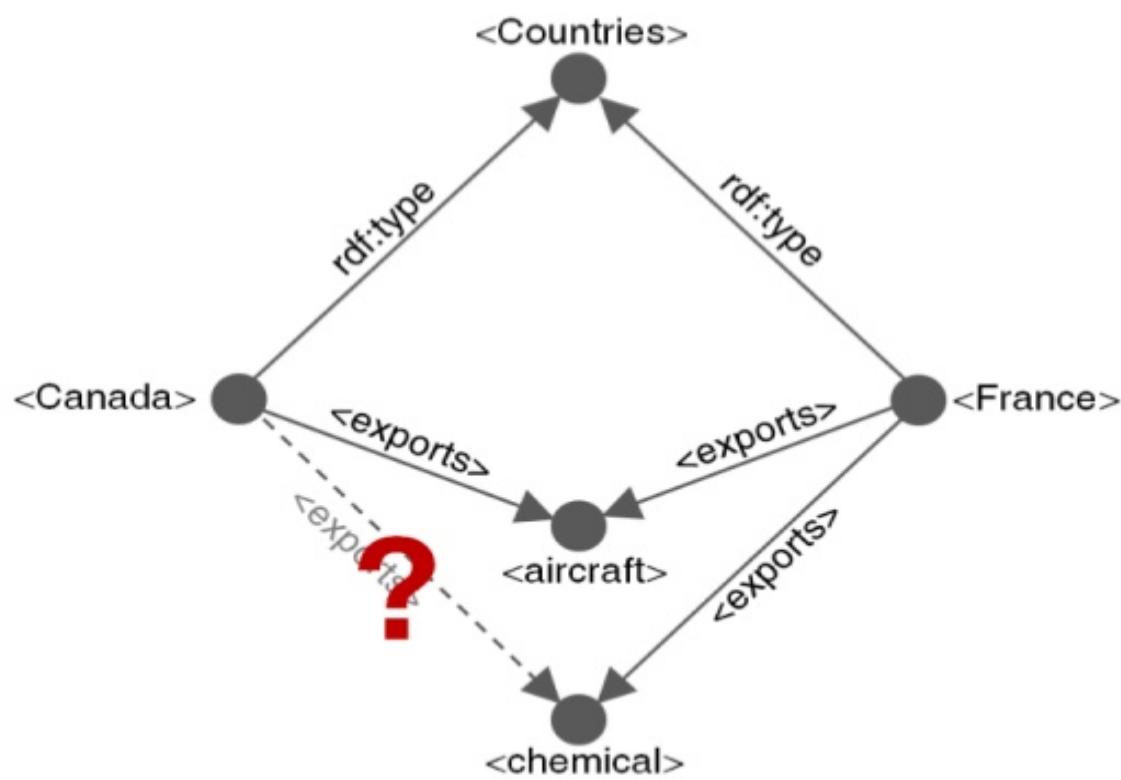
res0: Array[(org.apache.spark.graphx.VertexId, String)] =
  Array((1721488,<wordnet_electronic_equipment_103278248>))
```

[worldstopexports.com](http://worldstopexports.com)

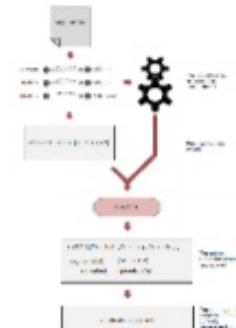
- 
1. Oil: US\$77.8 billion (19% of total exports)
  2. Vehicles: \$60 billion (14.7%)
  3. Machines, engines, pumps: \$31.1 billion (7.6%)
  4. Gems, precious metals: \$19 billion (4.7%)
  5. Electronic equipment: \$13.2 billion (3.2%)
  6. Plastics: \$12.5 billion (3.1%)
  7. Aircraft, spacecraft: \$12.3 billion (3%)
  8. Wood: \$11.8 billion (2.9%)
  9. Aluminum: \$8.2 billion (2%)
  10. Paper: \$7.7 billion (1.9%)

```
grep "<exports>" yagoFacts.tsv | grep "<Canada>"
<id_1wrx1wu_dv6_1pqb7a4> <Canada> <exports> <wordnet_aluminum_114627820>
<id_1wrx1wu_dv6_j218e6> <Canada> <exports> <wordnet_electricity_111449907>
<id_1wrx1wu_dv6_t6wm01> <Canada> <exports> <wordnet_lumber_114943580>
<id_1wrx1wu_dv6_jhowo0> <Canada> <exports> <wordnet_natural_gas_114960090>
<id_1wrx1wu_dv6_s9bzqx> <Canada> <exports> <wordnet_aircraft_102686568>
<id_1wrx1wu_dv6_12fzkgg> <Canada> <exports> <wordnet_plastic_114592610>
```

# Country Exports Example



## Dataflow



## Results

[www.socfilter.org](http://www.socfilter.org) [socfilter@socfilter.org](mailto:socfilter@socfilter.org)



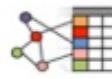
Country Exports Example



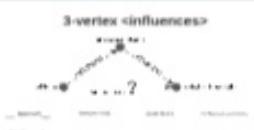
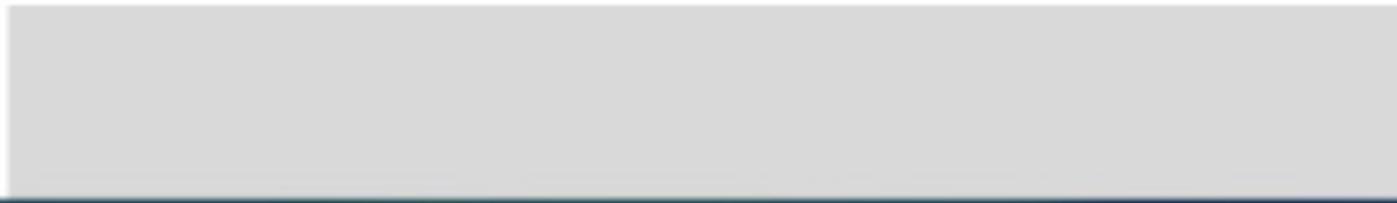
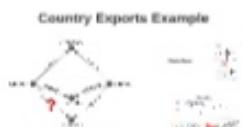
# Graph Processing vs. Graph Database

OLAP-like

OLTP-like

Parallel algorithms	Parallel query	Single-anchor query	Transactions
 GraphX			
	 GraphFrames		
		 neo4j	 TITAN  OrientDB

# GraphFrames



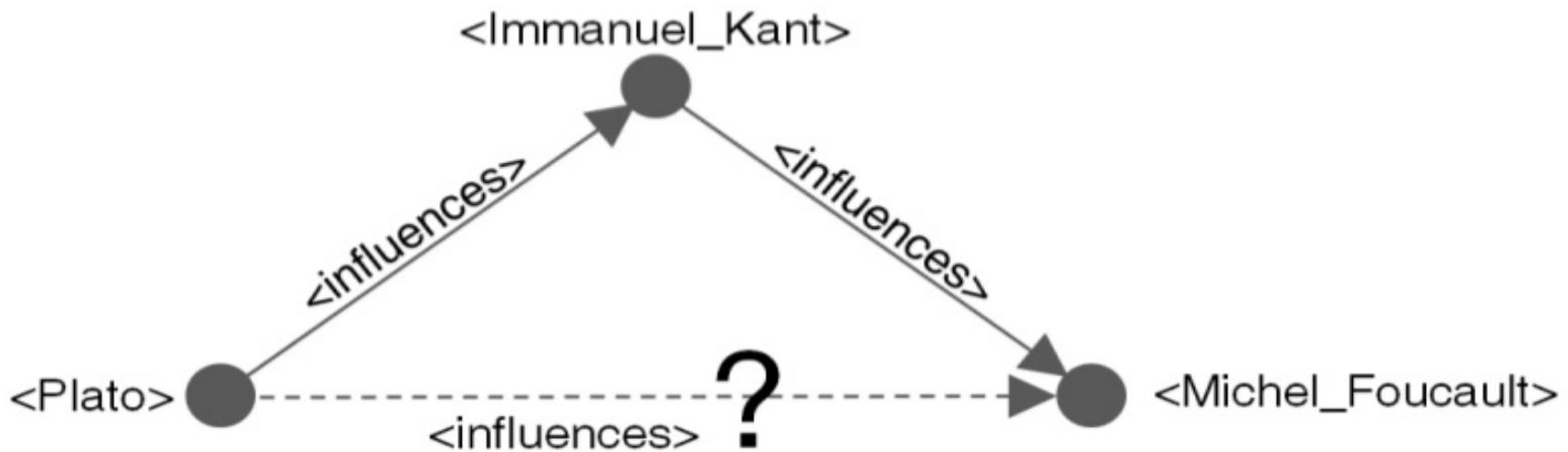
3-vertex general linear pattern

Using WPS Office 2013  
Part 1  
Getting Started

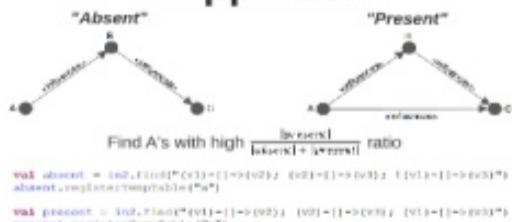
4-vertex general Y pattern

#### *Out of Memory*

## 3-vertex <influences>



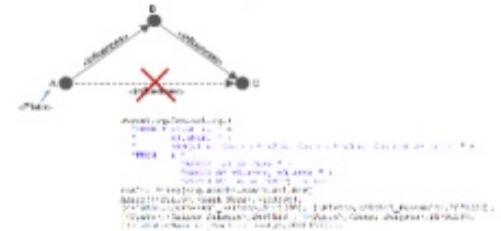
## Approach



Outdegree in SQL

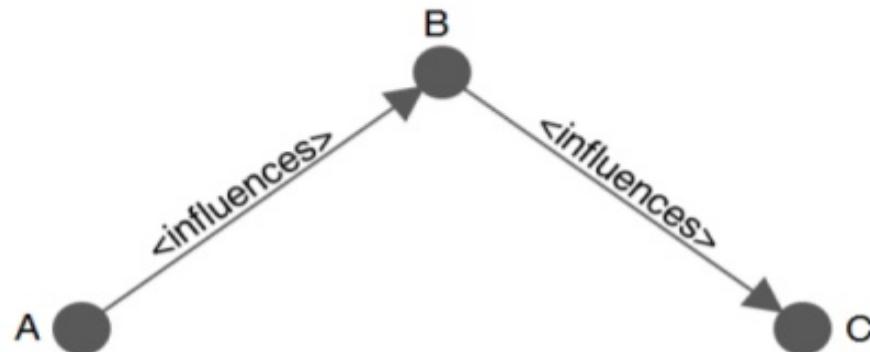
## **Score the A's**

**For Plato as A, score the C's**

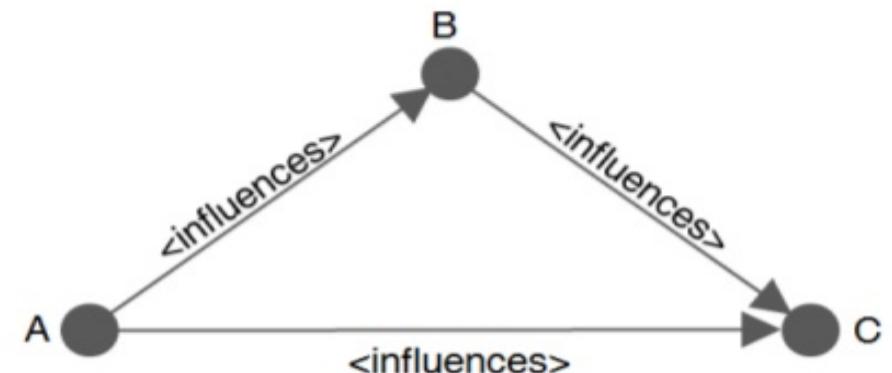


# Approach

**"Absent"**



**"Present"**



Find A's with high  $\frac{|present|}{|absent| + |present|}$  ratio

```
val absent = in2.find("(v1)-[]->(v2); (v2)-[]->(v3); !(v1)-[]->(v3)")  
absent.registerTempTable("a")
```

```
val present = in2.find("(v1)-[]->(v2); (v2)-[]->(v3); (v1)-[]->(v3)")  
present.registerTempTable("p")
```

# Outdegree in SQL

```
grep "<influences>" yagoFacts.tsv >yagoFactsInfluences.tsv

val in = readRdfDf(sc, "yagoFactsInfluences.tsv")

in.edges.registerTempTable("e")
in.vertices.registerTempTable("v")

val in2 = GraphFrame(in.vertices.sqlContext.sql(
    "SELECT v.id, " +
    "       FIRST(v.attr) AS attr, " +
    "       COUNT(*) AS outdegree " +
    "FROM   v " +
    "JOIN   e " +
    "ON     v.id=e.src " +
    "GROUP BY v.id").cache,
    in.edges)
```

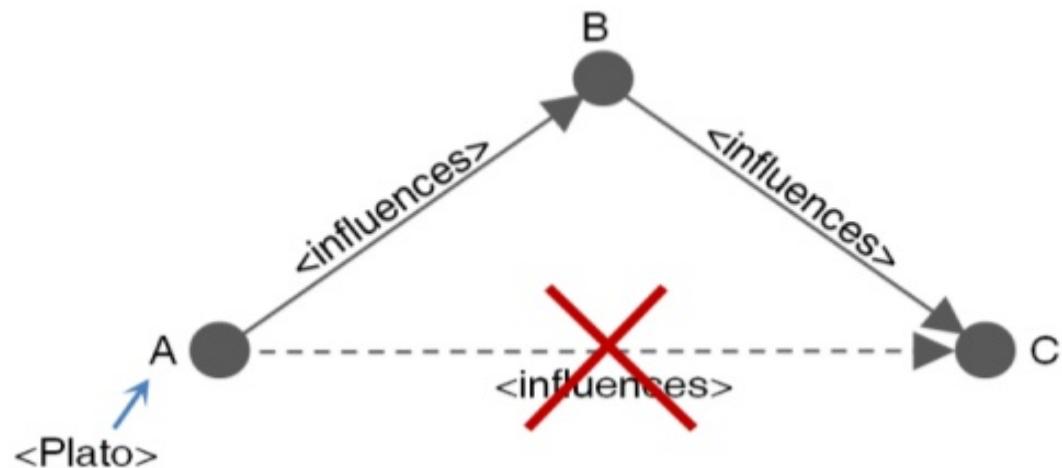
# Score the A's

```
absent.sqlContext.sql(  
  "SELECT v1.an," +  
   "  SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS ac " +  
 "FROM a " +  
 "GROUP BY v1").registerTempTable("aa")  
  
present.sqlContext.sql(  
  "SELECT v1.pn," +  
   "  SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS pc " +  
 "FROM p " +  
 "GROUP BY v1").registerTempTable("pa")  
  
absent.sqlContext.sql("SELECT an," +  
   "  ac * pc/(ac+pc) AS score " +  
 "FROM aa " +  
 "JOIN pa" +  
 "  ON an=pn " +  
 "ORDER BY score DESC").show
```

an	score
[7662,<Plato>,102]	3.822406412297308E7
[10648,<Aristotle...>]	3.2961326121938106E7
[4959,<Immanuel_K...>]	2.6445857520978764E7
[2961,<Georg_Wilh...>]	2.1092802441273782E7
[9304,<Baruch_Spi...>]	1.4513392385496272E7
[12217,<René_Desc...>]	1.2407118036818413E7
[12660,<Johann_Wo...>]	1.0109121178397963E7
[11895,<Jean-Jacq...>]	9081581.748842742
[11615,<Gottfried...>]	7146037.710399863
[2025,<Friedrich_...>]	6897244.1896990575
[1082,<William_Sh...>]	4168778.144288711
[11034,<Adam_Smit...>]	4100936.5022027283
[1121,<John_Locke...>]	3868447.819527024
[1566,<Heraclitus...>]	3616900.3025887734
[3746,<Karl_Marx>...]	3575419.671920321
[10954,<Søren_Kie...>]	3143375.914849735
[7322,<David_Hume...>]	3122089.3473657905
[8540,<Arthur_Sch...>]	2978239.727690162
[3186,<Ibn_Tufail...>]	2234249.031615453
[8267,<Epicurus>,24]	1812594.4073720106

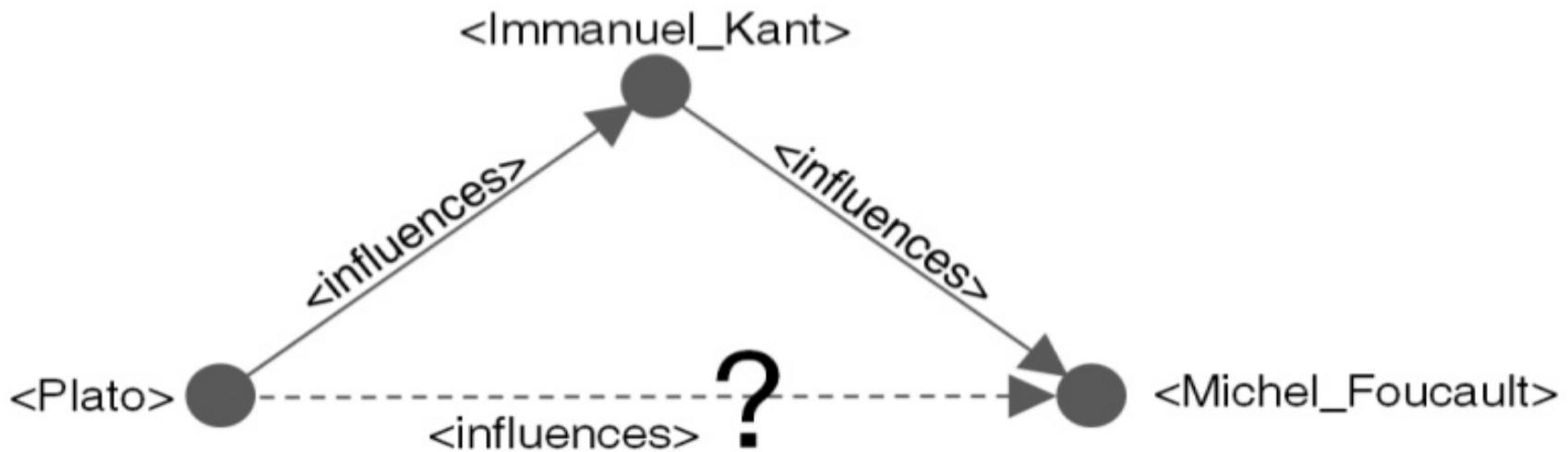
only showing top 20 rows

# For Plato as A, score the C's

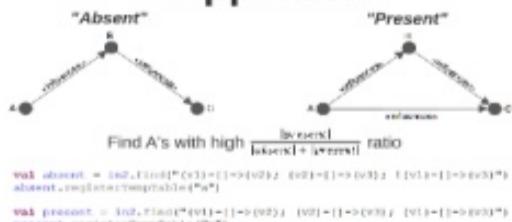


```
absent.sqlContext.sql(  
    "SELECT v1.attr, " +  
    "      v3.attr, " +  
    "      SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS score " +  
    "FROM    a " +  
            "WHERE  v1.id=7662 " +  
            "GROUP BY v1.attr, v3.attr " +  
            "ORDER BY score DESC").collect  
res24: Array[org.apache.spark.sql.Row] =  
Array([<Plato>,<Karl_Marx>,7139388],  
[<Plato>,<Jean-Paul_Sartre>,3143640], [<Plato>,<Michel_Foucault>,2871606],  
[<Plato>,<Gilles_Deleuze>,2689128], [<Plato>,<Henri_Bergson>,2179128],  
[<Plato>,<Maurice_Merleau-Ponty>,2088450]...)
```

## 3-vertex <influences>



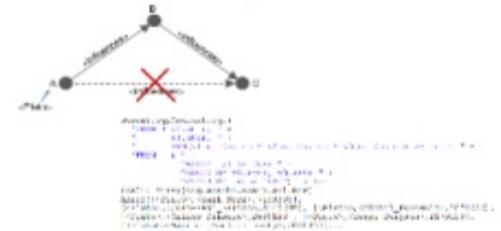
## Approach



Outdegree in SQL

## Score the A's

**For Plato as A, score the C's**



# 3-vertex general linear pattern

If we find  should we attach to it a 

to make 

## Reading YAGO

```
val yago = readHDFS(sc, "/home/ec2-user/yago/yagofacts.tsv")
def mergeIntoYago(filename:String):Unit = {
  val r = readHDFS(sc, filename)
  r.vertices.cache
  r.edges.cache
  val g = mergeGraphs(yago, r)
  val g2 = graphframe(g.vertices, g.edges.distinct)
  g2.vertices.cache
  g2.edges.count
  g2.edges.cache
  g2.edges.count
  yago.vertices.unpersist
  yago.edges.unpersist
  r.vertices.unpersist
  r.edges.unpersist
  yago = g2
}
mergeIntoYago("/home/ec2-user/yago/yagolabels.tsv")
mergeIntoYago("/home/ec2-user/yago/yagotaxonomy.tsv")
```

## Query

```
WITH FILTER(?) AS R1, FILTER(?) AS R2, FILTER(?) AS R3
  .SELECTFROM("el-artist AS es1", "sq-artist AS es2")
  .SELECTFROM("v1-10-as-v2", "v2-10-as-v3")
  .JOIN("el-artist", "es1", "v1", "v2")
  .COUNT
  .JOIN(yago.edges, groupby("v2"))
  .COUNT
  .SELECTBEG("utter", "COUNT AS ccount"),
  "es1" --> "v2"
  .SELECTEND("el-artist", "es2", "v3", "v4", "COUNT(ccount) AS rcount")
  .ORDERBY("rname", "desc")
  .SHOW(10)
```



## Results

Artist	Artist	Artist	Count
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1

Discovered bug in YAGO 3.02 (not in YAGO 3.00 or forthcoming YAGO 3.03)

Artist	Artist	Artist	Count
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1
John Lennon	John Lennon	John Lennon	1

← What?

→ What?

# Reading YAGO

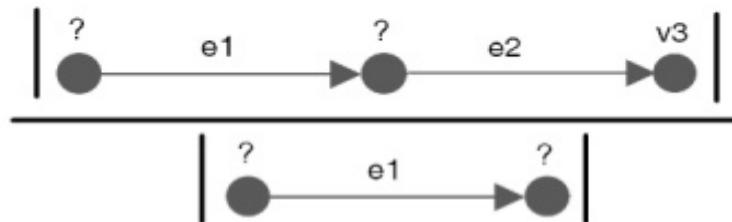
```
var yago = readRdfDf(sc, "/home/ec2-user/yago/yagoFacts.tsv")

def mergeIntoYago(filename:String):Unit = {
    val r = readRdfDf(sc, filename)
    r.vertices.cache
    r.edges.cache
    val g = mergeGraphs(yago, r)
    val g2 = GraphFrame(g.vertices, g.edges.distinct)
    g2.vertices.cache
    g2.vertices.count
    g2.edges.cache
    g2.edges.count
    yago.vertices.unpersist
    yago.edges.unpersist
    r.vertices.unpersist
    r.edges.unpersist
    yago = g2
}

mergeIntoYago("/home/ec2-user/yago/yagoLabels.tsv")
mergeIntoYago("/home/ec2-user/yago/yagoTaxonomy.tsv")
```

# Query

```
yago.find("()-[e1]-(v2); (v2)-[e2]-(v3)")  
    .distinct  
    .selectExpr("e1.attr AS e1attr", "e2.attr AS e2attr",  
                "v3.id AS v3id", "v3.attr AS v3attr")  
    .groupBy("e1attr", "e2attr", "v3id", "v3attr")  
    .count  
    .join(yago.edges.groupBy("attr")  
          .count  
          .selectExpr("attr", "count AS e1count"),  
          "$e1attr" === "$attr")  
    .selectExpr("e1attr", "e2attr", "v3attr", "count/e1count AS ratio")  
    .orderBy($"ratio".desc)  
    .show(40)
```



# Results



e1attr	e2attr	v3attr	ratio
<hasAcademicAdvisor>	<hasGender>	<male>	0.9746811345897582
<influences>	<hasGender>	<male>	0.8912262249101587
<hasGender>	skos:prefLabel	"male"@eng	0.8428451872279868
<hasGender>	rdfs:label	"male"@eng	0.8428451872279868
<isCitizenOf>	<participatedIn>	<War_on_Terror>	0.835011170256778
<dealsWith>	<participatedIn>	<War_on_Terror>	0.772226304188097
<isCitizenOf>	<participatedIn>	<Operation_Enduri...>	0.7614529608075682
<isCitizenOf>	<participatedIn>	<War_in_Afghanist...>	0.7572055051438343
<isCitizenOf>	<participatedIn>	<Operation_Enduri...>	0.7488484982210332
<isCitizenOf>	<dealsWith>	<China>	0.742863447058499
<isCitizenOf>	<participatedIn>	<Korean_War>	0.7089389635104946
<isCitizenOf>	<participatedIn>	<Unified_Task_Force>	0.6842540750751579
<dealsWith>	<participatedIn>	<Operation_Enduri...>	0.6796473181484203
<hasNeighbor>	<dealsWith>	<China>	0.6774193548387096

Discovered bug in YAGO 3.02 (not in YAGO 3.00 or forthcoming YAGO 3.03)

e1attr	e2attr	v3attr	ratio
<isCitizenOf>	<participatedIn>	<United_States>	0.6223365172667157
<dealsWith>	<dealsWith>	<China>	0.6193975018368847
<dealsWith>	<dealsWith>	<United_States>	0.6186627479794269
<dealsWith>	<participatedIn>	<Operation_Enduri...>	0.6018424028463469
<isCitizenOf>	<participatedIn>	<Operation_Desert...>	0.587031469785145
<isCitizenOf>	<participatedIn>	<Operation_Active...>	0.586865984499545
<isCitizenOf>	<participatedIn>	<Operation_Eagle'...>	0.5862316242380782
<isCitizenOf>	<participatedIn>	<Operation_Enduri...>	0.5859282345478115
<isCitizenOf>	<participatedIn>	<Iraqi_no-fly_zones>	0.5689935736547426
<isCitizenOf>	<participatedIn>	<Colombian_confli...>	0.5657390297046088
<isCitizenOf>	<participatedIn>	<Operation_Provid...>	0.5540447361888738
<isCitizenOf>	<participatedIn>	<European_theatre...>	0.5523071406900737
<hasOfficialLangu...	rdfs:label	"Joseph"@eng	0.5515832482124617
<isCitizenOf>	<participatedIn>	<Helmand_province...>	0.5459635380754061
<isCitizenOf>	<participatedIn>	<People's_General...>	0.5350139283448714
<isCitizenOf>	<participatedIn>	<Western_Front_(W...>	0.5346277960118047
<hasOfficialLangu...	rdfs:label	"Francis"@eng	0.5342185903983657

← What?

← What?

# 3-vertex general linear pattern

If we find  should we attach to it a 

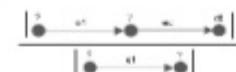
to make 

## Reading YAGO

```
val yago = readHDFS(sc, "/home/ec2-user/yago/yagofacts.tsv")
def mergeIntoYago(filename:String):Unit = {
  val r = readHDFS(sc, filename)
  r.vertices.cache
  r.edges.cache
  val g = mergeGraphs(yago, r)
  val g2 = graphframe(g.vertices, g.edges.distinct)
  g2.vertices.cache
  g2.edges.count
  g2.edges.cache
  g2.edges.count
  yago.vertices.unpersist
  yago.edges.unpersist
  r.vertices.unpersist
  r.edges.unpersist
  yago = g2
}
mergeIntoYago("/home/ec2-user/yago/yagolabels.tsv")
mergeIntoYago("/home/ec2-user/yago/yagotaxonomy.tsv")
```

## Query

```
WITH FILTER(?) AS R1, FILTER(?) AS R2, FILTER(?) AS R3
  .SELECTFROM("el-artist AS es1", "sq-artist AS es2")
  .SELECTFROM("el-artist AS es3", "sq-artist AS es4")
  .JOIN(es1, es2, "elArtist", "sqArtist")
  .COUNT
  .JOIN(yago.edges, groupby("elArtist"))
  .COUNT
  .SELECTBEG("elArtist", "COUNT AS elCount"),
  "elArtist" --> "elArtist"
  .SELECTEND("elArtist", "elArtist", "elArtist", "elArtist/COUNT AS elRatio")
  .ORDERBY("elRatio", DESC)
  .SHOW(10)
```



## Results

| elArtist    |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| John Lennon |
| John Lennon |
| John Lennon |
| John Lennon |

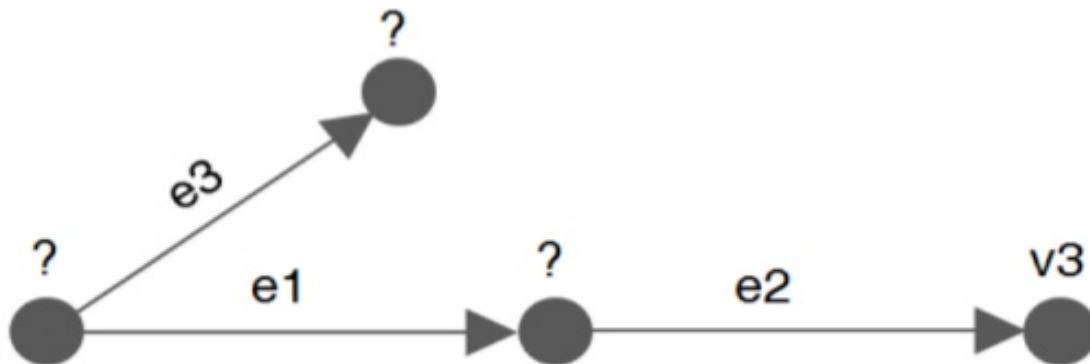
Discovered bug in YAGO 3.02 (not in YAGO 3.00 or forthcoming YAGO 3.03)

| elArtist    |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| John Lennon |
| John Lennon |
| John Lennon |
| John Lennon |

← What?

→ What?

## 4-vertex general Y pattern



```
y1 = yago.find("(v1)-[e3]-(v4); (v1)-[e1]-(v2); (v2)-[e2]-(v3)")  
    .distinct  
    .selectExpr("e3.attr AS e3attr", "e1.attr AS e1attr", "e2.attr AS e2attr",  
               "v3.id AS v3id", "v3.attr AS v3attr")  
    .groupBy("e3attr", "e1attr", "e2attr", "v3id", "v3attr")  
    .count  
    .cache
```

# Out of Memory

AWS 5 nodes of r3.8xlarge, each 244GB RAM, 32 vCPU, 320GB SSD



AWS 5 nodes of r3.8xlarge, each 244GB RAM, 32 vCPU, 320GB SSD

**3-vertex <hasChild> <isMarriedTo>**



## Initial Results



### Refinement



```

val y2 = filterdEdges(yapp, ("$attr" === "hasChild") || "$attr" === "isMarriedTo") || "$attr" === "hasGender"))
val df = y2.filter("parent1[\"id\"]<child[\"id\"]")((parent1:[id]; [parent2:[id]]<(child:[id]); (parent1:[id])<!(parent2:[id])>(parent2:[id])>!(child:[id])) &&
    .filter("($parent2.id.str" === "male") && ($parent2.id.str" === "female")) &&
    ("$x1.id.str" === "hasChild1") && ("$x2.id.str" === "hasChild2"))
    .select("parent1.id AS parent1", "parent2.id AS parent2", "child.id AS child")

```



# Initial Results



```

var yago = readRdfDf(sc, "/yago3/yagoFacts.tsv")

def removeSingletons(g:GraphFrame) =
  GraphFrame(g.vertices.sqlContext.createDataFrame(
    g.triplets.select("src").map(_.getStruct(0))
      .union(g.triplets.select("dst").map(_.getStruct(0)))
      .distinct,
    g.vertices.schema),
  g.edges)

def filterEdges(g:GraphFrame, condition:Column) =
  removeSingletons(GraphFrame(g.vertices, g.edges.where(condition)))

val y2 = filterEdges(yago, $"attr" === "<hasChild>" || $"attr" === "<isMarriedTo>")

val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); !(parent1)-[]->(parent2)")
  .filter($"parent1" !== $"parent2" && ("$e1.attr" === "<hasChild>") &&
    ("$e2.attr" === "<hasChild>"))
  .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2",
    "child.attr AS child")
  
```

parent1	parent2	child
<Arcadius>	<Aelia_Flaccilla>	<Honorius_(emperor)>
<Victor_Emanuel_...>	<Victor_Emanuel_...>	<Maria_Pia_of_Savoy>
<George_VI_of_the...>		<George_VI>
<Otto_I,_Duke_of_...>	<Otto_I_Wittelsba...>	<Louis_I,_Duke_of_...>
<Jack_Nicholas_Pr...>	<Rhoda_Pritzker>	<Nicholas_J._Prit...>
	<Augusta_Leigh>	<Lord_Byron>
<Mehmed_the_Conqu...>		<Elizabeth_Medora...>
<Jane_Stanhope,_C...>	<Charles_Stanhope...>	<Charles_Stanhope...>
<Eleanor_Beaucham...>	<Edmund_Beaufort,...>	<Edmund_Beaufort,...>
<Empress_Song_Fujin>		<Li_Bian>
	<Guy_XIV_de_Laval>	<Li_Jing_(Souther...>
	<Ronald_Reagan>	<Isabella_of_Brit...>
<Benedict_Swingat...>	<Dorothy_Walker_B...>	<George_H._W._Bush>
	<Elizabeth_Calvert>	<Eleanor_Calvert>
	<Hoapili>	<Kekuanaoa>
<Leopold_II_of_Be...>	<Prince_Philippe,...>	<Albert_I_of_Belg...>
	<Henry_the_Fowler>	<Henry_I_the_Fowler>
	<Lý_Hu?_Tông>	<Tr?n_Th?_Dung>
<Argia_(mythology)>		<Aristodemus>
<William_I_of_Eng...>	<William_the_Conq...>	<Robert_Curthose>
	<Emperor_Mommu>	<Emperor_Monmu>
		<Emperor_Shomu>

# Refinement



```

val y2 = filterEdges(yago, ("$attr" === "<hasChild>" || "$attr" === "<isMarriedTo>" || "$attr" === "<hasGender>"))

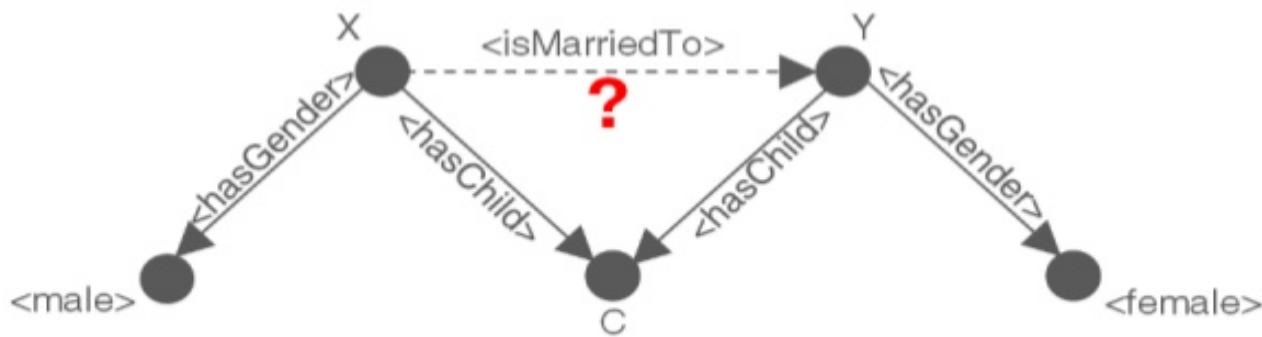
val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); (parent1)-[]->(p1gender); (parent2)-[]->(p2gender)")
    .filter(("p1gender.attr" === "<male>") && ("p2gender.attr" === "<female>") &&
            ("e1.attr" === "<hasChild>") && ("e2.attr" === "<hasChild>"))
    .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2", "child.attr AS child")

```

parent1	parent2	child
<Pineapple_1>	<GuitarPlayer_Hanout_1>	<Passionate_1>
<Pineapple_1>	<GuitarPlayer_Hanout_1>	<RockRehearsal_1>
<GuitarPlayer_Hanout_1>	<GuitarPlayer_Hanout_1>	<Guitar_Virtuoso_1>
<John_de_Foucault_1>	<Chargeman_1>	<Sleek_Co_1_of_Hizz_1>
<John_de_Foucault_1>	<Chargeman_1>	<Master_of_Poly_1>
<Prima_Trombones_1>	<Cleric_1>	<Princess_Yennefer_1>
<Prima_Trombones_1>	<Cleric_1>	<Jack_Dorobson_1>
<Guy_Beauchamp_1>	<Cleric_1>	<Emily_Bethune_1>
<Guy_Beauchamp_1>	<Cleric_1>	<Lilith_Bethune_1>
<Robert_Sidney_1>	<Cleric_1>	<Robert_Sidney_2>
<Bruce_Butler_1>	<Cleric_1>	<Freelie_Butler_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Beaute_de_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Isabella_of_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Marguerite_of_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Charles_VI_of_France_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Regina_the_Beautiful_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Frederick_II_Austria_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Audrey_Medieval_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Albrecht_Austria_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Edgarito_1>

parent1	parent2	child
<Pinedjem_I>	<Duathathor-Henut...>	<Psusennes_I>
<Pinedjem_I>	<Duathathor-Henut...>	<Menkheperre>
<AbdÜlmejid_I>	<GÜlüstü_Kadin_Ef...>	<Mehmed_VI>
<Charles_III_of_N...>	<Eleanor_of_Casti...>	<Blanche_I_of_Nav...>
<John_de_Foix,_1s...>	<Margaret_de_la_P...>	<Gaston_de_Foix,...>
<Prince_Tomohito_...>	<Princess_Tomohit...>	<Princess_Akiko_o...>
<Prince_Tomohito_...>	<Princess_Tomohit...>	<Princess_Yoko_of...>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Jack_Osbourne>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Kelly_Osbourne>
<Robert_Sidney,_1...>	<Barbara_Sidney,_...>	<Robert_Sidney,_2...>
<Bruce_Sudano>	<Donna_Summer>	<Brooklyn_Sudano>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Blanche_of_Franc...>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Isabella_of_Fran...>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Margaret_I,_Coun...>
<Edmund_I>	<Ælfgifu_of_Shaf...>	<Eadwig>
<Edmund_I>	<Ælfgifu_of_Shaf...>	<Edgar_the_Peaceful>
<Ludovico_III_Gon...>	<Barbara_of_Brand...>	<Federico_I_Gonza...>
<Thomas_Hardeman,...>	<Mary_Perkins>	<Bailey_Hardeman>
<Casimir,_Margrav...>	<Susanna_of_Bavaria>	<Albert_Alcibiade...>
<Edward_the_Elder>	<Ælfflæd,_wife_of...>	<Eadgyth>

# Refinement



```
val y2 = filterEdges(yago, ($"attr" === "<hasChild>" || $"attr" === "<isMarriedTo>" || $"attr" === "<hasGender>"))

val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); (parent1)-[]->(p1gender); (parent2)-[]->(p2gender)")
    .filter(($"p1gender.attr" === "<male>") && ($"p2gender.attr" === "<female>") &&
            ("$e1.attr" === "<hasChild>") && ("$e2.attr" === "<hasChild>"))
    .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2", "child.attr AS child")
```

parent1	parent2	child
<Pineapple_1>	<GuitarPlayer_Hanout_1>	<Passionate_1>
<Pineapple_1>	<GuitarPlayer_Hanout_1>	<RockRehearsal_1>
<GuitarPlayer_Hanout_1>	<GuitarPlayer_Hanout_1>	<Guitar_Virtuoso_1>
<John_de_Foucault_1>	<Chargeman_1>	<Sleek_Co_1_of_Hizz_1>
<John_de_Foucault_1>	<Chargeman_1>	<Master_of_Poly_1>
<Prima_Trombones_1>	<Cleric_1>	<Princess_Yennefer_1>
<Prima_Trombones_1>	<Cleric_1>	<Jack_Dorobson_1>
<Guy_Beauchamp_1>	<Cleric_1>	<Emily_Bethune_1>
<Guy_Beauchamp_1>	<Cleric_1>	<Lilith_Bethune_1>
<Robert_Sidney_1>	<Cleric_1>	<Robert_Sidney_2>
<Bruce_Butler_1>	<Cleric_1>	<Freelie_Butler_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Beaute_de_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Isabella_of_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Marguerite_of_France_1>
<Philippe_V_of_France_1>	<Count_II_Courtesy_1>	<Charles_VI_of_France_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Regina_the_Beautiful_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Frederick_II_Austria_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Audrey_Medieval_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Albrecht_Austria_1>
<Audrey_22_gam_1>	<Baroness_of_Sound_1>	<Edgarito_1>



AWS 5 nodes of r3.8xlarge, each 244GB RAM, 32 vCPU, 320GB SSD

## 3-vertex `<hasChild>` `<isMarriedTo>`



### Initial Results

```
val ypg = rodelIO(ys, "yago0/yagoFacts.ttl")
pgf.readString(ypg, "yago0/yagoFacts.ttl")
pgf.refiningVerticesInContext(rodelIOFreeze(
    g_triples, dataset("yag"), pgf._position(0))
    .select(triples.predicate("isMarriedTo"))
    .context(
        g_triples,
        g_triples),
    g_triples)

#filteringTriples, refiningLabels) <
refiningVerticesInContext(g_triples, g_triples, condition))
```

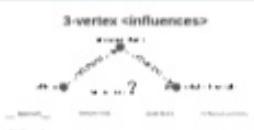
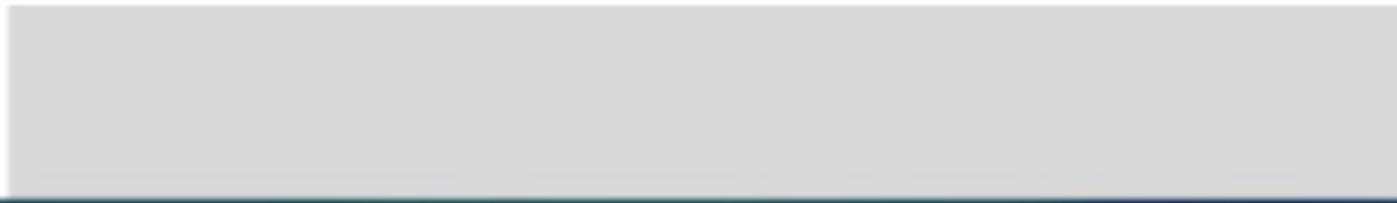
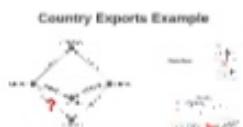
parent1	parent2	child
Yago:00000000000000000000000000000000	Yago:00000000000000000000000000000000	Yago:00000000000000000000000000000000

### Refinement

```
val y2 = #filteringTriples(ypg, ($"attr" == "hasChild" || $"attr" == "isMarriedTo" || $"attr" == "hasGender"))
val df = y2.find("parent1-[e1]->(child); (parent2)-[e2]->(child); (parent1)-[]-(p1gender); (parent2)-[]-(p2gender)")
    .filter((#"p1gender.attr" == "male") && (#"p2gender.attr" == "female")) &&
    (#"e1.attr" == "hasChild") && (#"e2.attr" == "hasChild"))
    .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2", "child.attr AS child")
```



# GraphFrames



3-vertex general linear pattern

Using WPS Office 2013  
Part 1  
Getting Started

4-vertex general Y pattern

#### *Out of Memory*

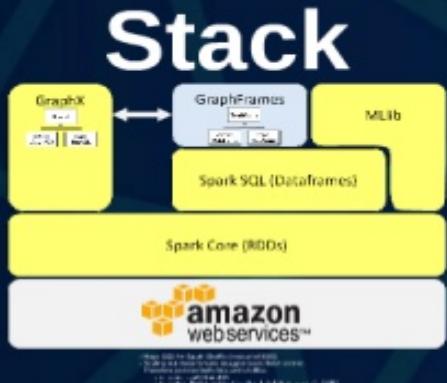
# Graph Processing vs. Graph Database

OLAP-like

OLTP-like

Parallel algorithms	Parallel query	Single-anchor query	Transactions
 GraphX			
	 GraphFrames		
		 neo4j	 TITAN  OrientDB

# Spark



## Graph Processing vs. Graph Database

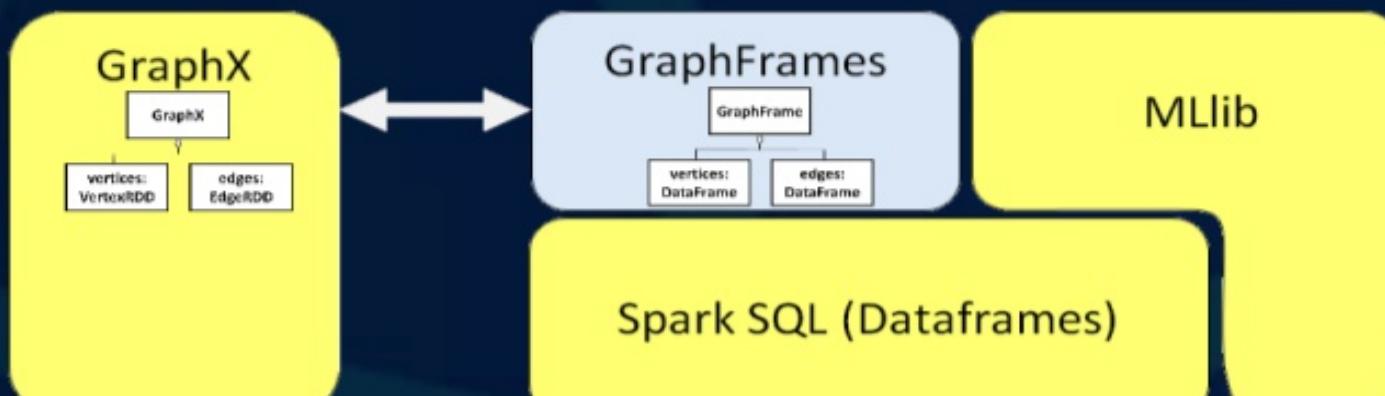
OLAP-like	Parallel algorithms	Parallel query	Single-anchor query	Transactions
	GraphX			
		GraphFrames		
			GraphFrames	

## History

### Timeline



# Stack



- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.  
Therefore partition both data and shuffles:

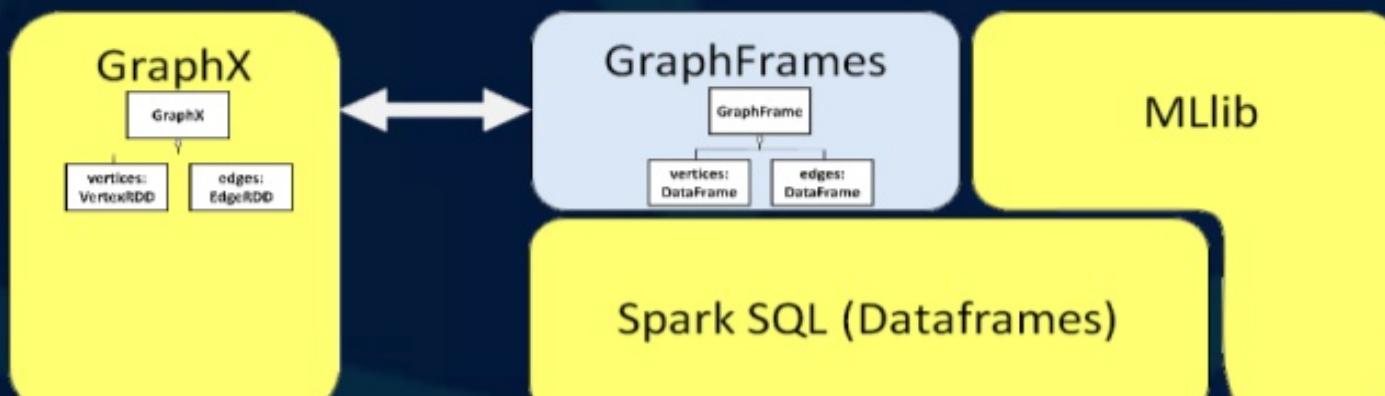
```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
```



- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.  
Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```

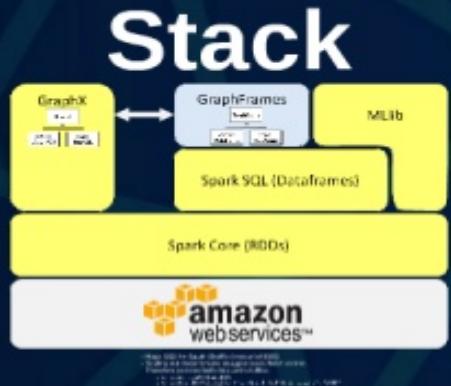
# Stack



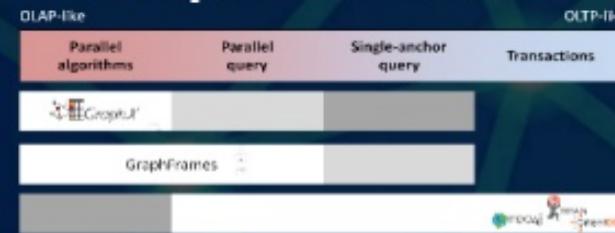
- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.  
Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
```

# Spark



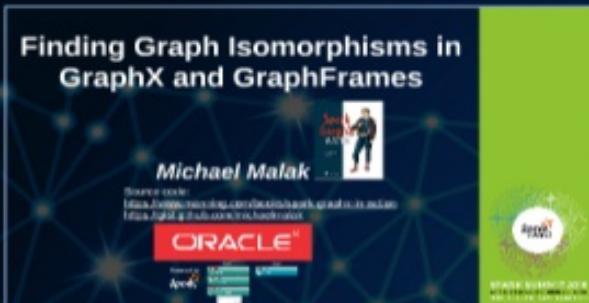
## Graph Processing vs. Graph Database



## History

### Timeline





Spark



## Isomorphisms

### Definition



### Finding

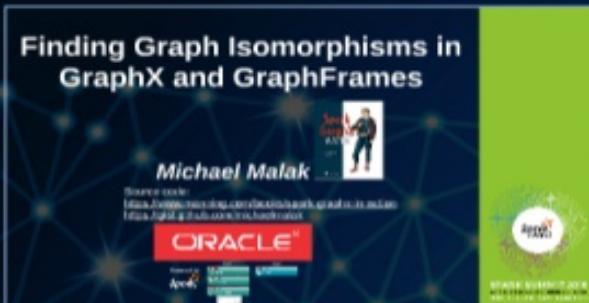
### Isomorphism

yago  
select knowledge



# Summary

- Mini-isomorphisms (aka rule mining) good for:
  - Data quality/data cleansing
  - Recommendations
    - Opportunities
    - Fraud alerts
    - Product recommendations
- SVD++ in GraphX useful for tiniest of rules
- Cypher subset in GraphFrames for larger patterns
  - For larger patterns, better to pre-filter



Spark



## Isomorphisms

### Definition



### Finding

### Isomorphism

yago  
select knowledge



# Finding Graph Isomorphisms in GraphX and GraphFrames

*Michael Malak*



Source code:

<https://www.manning.com/books/spark-graphx-in-action>

<https://gist.github.com/michaelmalak>



SPARK SUMMIT 2016  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO