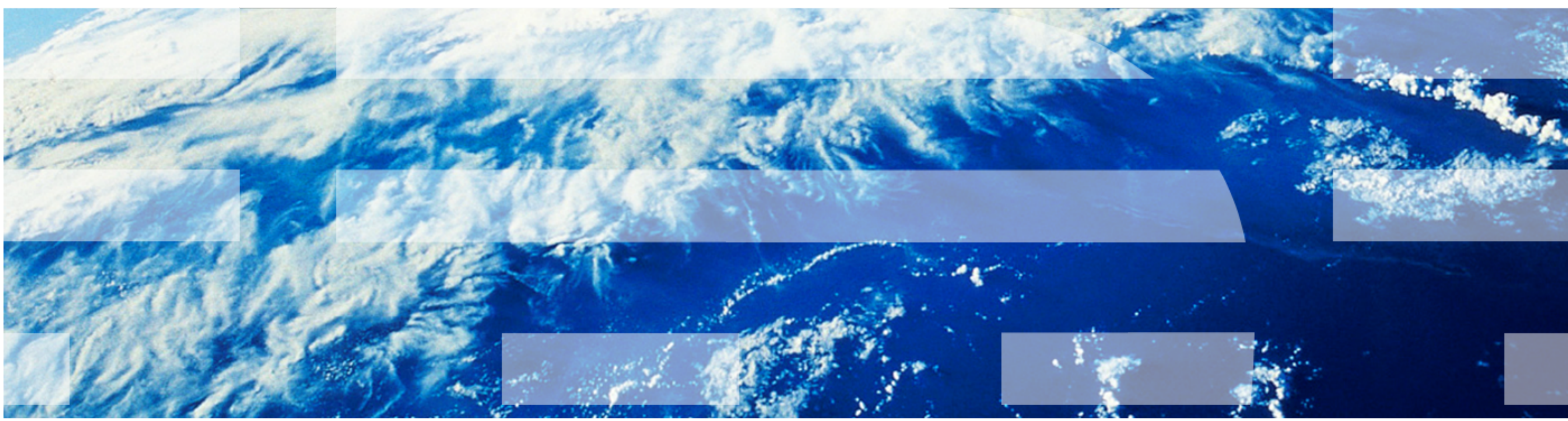
E6895 Advanced Big Data Analytics Lecture 4:

Data Store

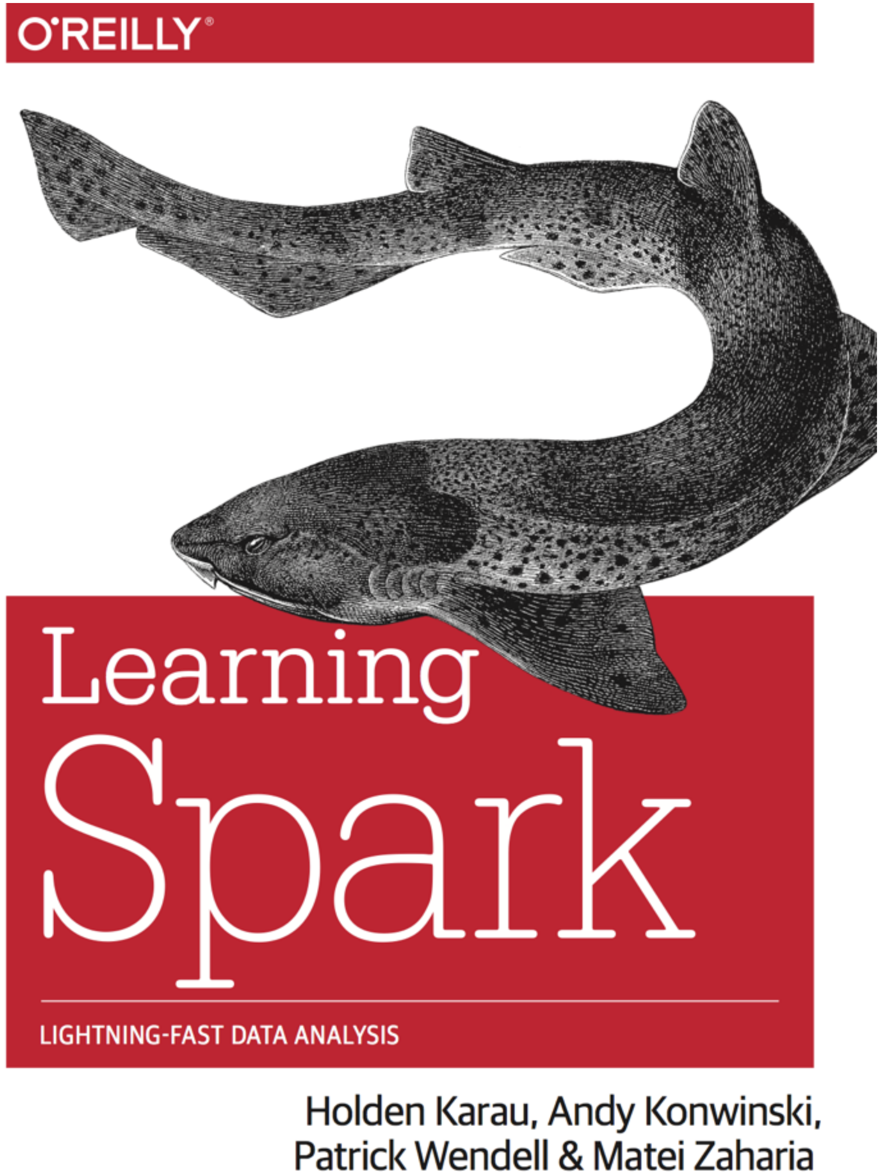
Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

Chief Scientist, Graph Computing, IBM Watson Research Center



Reference


O'REILLY
Learning
Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

Spark SQL

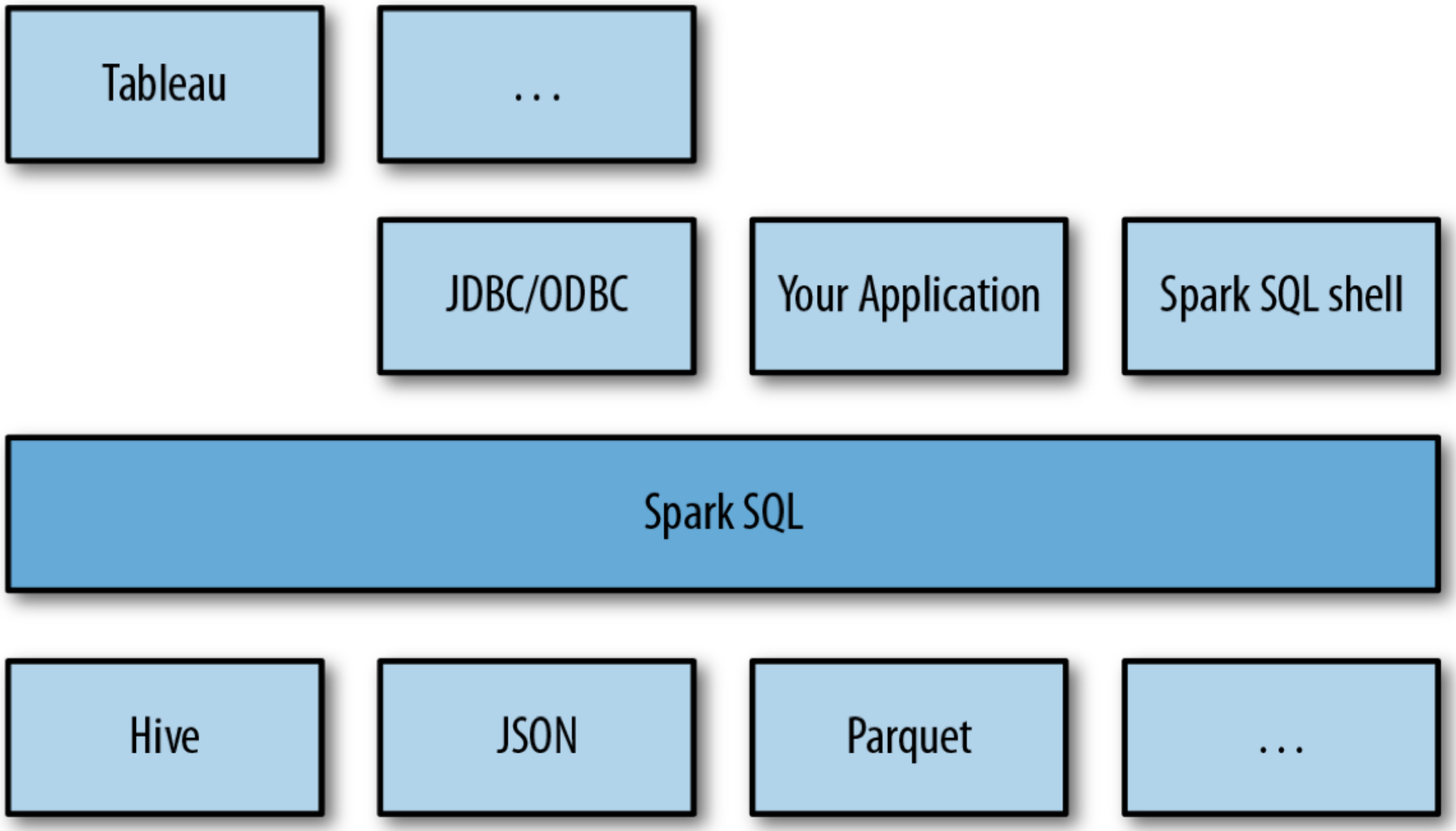
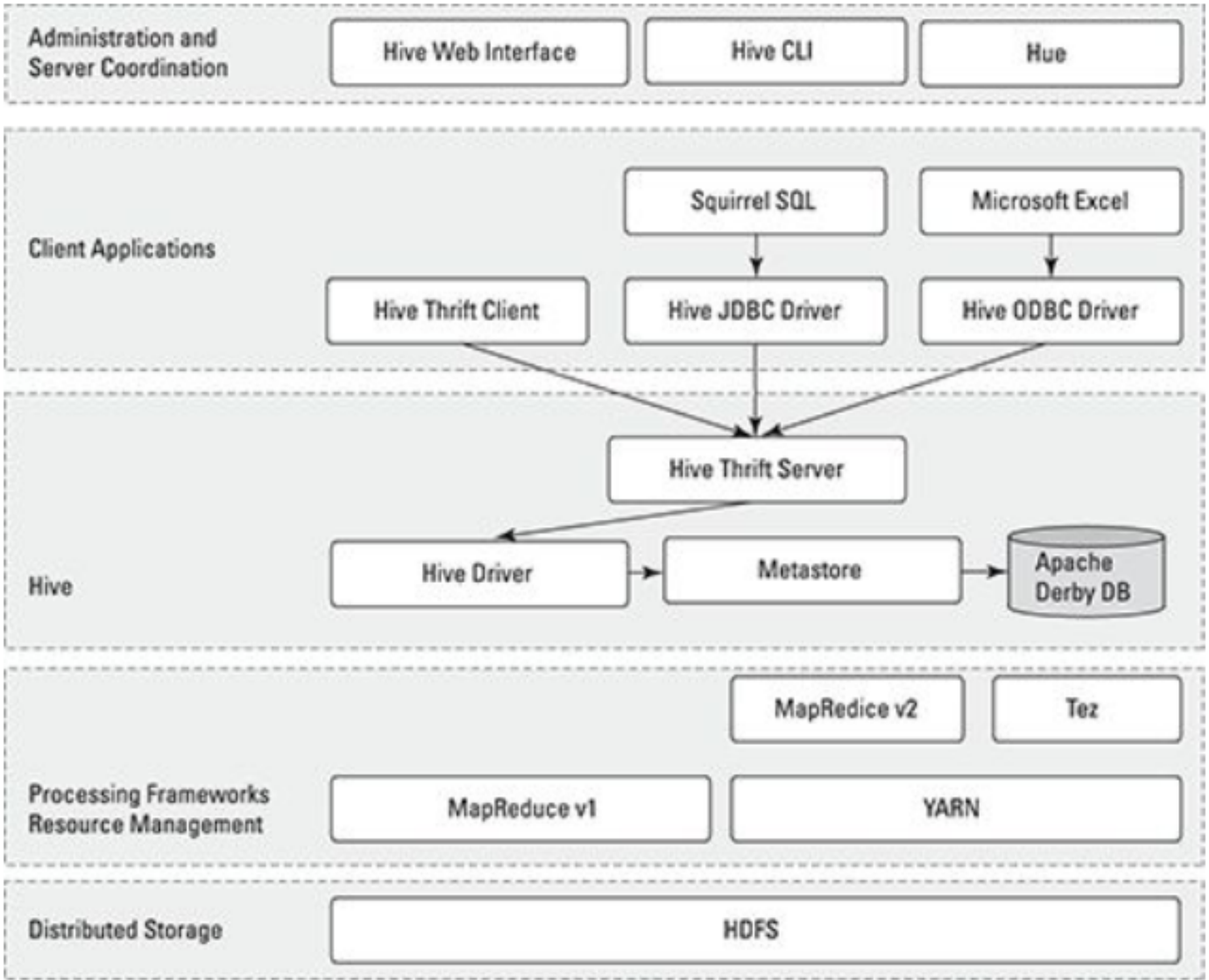


Tableau
. . .
JDBC/ODBC
Your Application
Spark SQL shell
Spark SQL
Hive
JSON
Parquet
. . .

Spark SQL

Spark SQL can be built with or without Apache Hive, the Hadoop SQL engine. Spark SQL with Hive support allows us to access Hive tables, UDFs (user-defined functions), SerDes (serialization and deserialization formats), and the Hive query language (HiveQL). Hive query language (HQL) It is important to note that including the Hive libraries does not require an existing Hive installation. In general, it is best to build Spark SQL with Hive support to access these features. If you download Spark in binary form, it should already be built with Hive support. If you are building Spark from source, you should run `sbt/sbt -Phive assembly`.

Apache Hive



Administration and Server Coordination
Hive Web Interface
Hive CLI
Hue
Client Applications
Squirrel SQL
Microsoft Excel
Hive Thrift Client
Hive JDBC Driver
Hive ODBC Driver
Hive Thrift Server
Hive
Hive Driver
Metastore
Apache Derby DB
MapRedice v2
Tez
Processing Frameworks Resource Management
MapReduce v1
YARN
Distributed Storage
HDFS

Using Hive to Create a Table

```
(A) $ $HIVE_HOME/bin hive --service cli
(B) hive> set hive.cli.print.current.db=true;
(C) hive (default)> CREATE DATABASE ourfirstdatabase;
OK
Time taken: 3.756 seconds
(D) hive (default)> USE ourfirstdatabase;
OK
Time taken: 0.039 seconds
(E) hive (ourfirstdatabase)> CREATE TABLE our_first_table (
                > FirstName     STRING,
                > LastName      STRING,
                > EmployeeId     INT);
OK
Time taken: 0.043 seconds
hive (ourfirstdatabase)> quit;
(F) $ ls /home/biadmin/Hive/warehouse/ourfirstdatabase.db
our_first_table
```

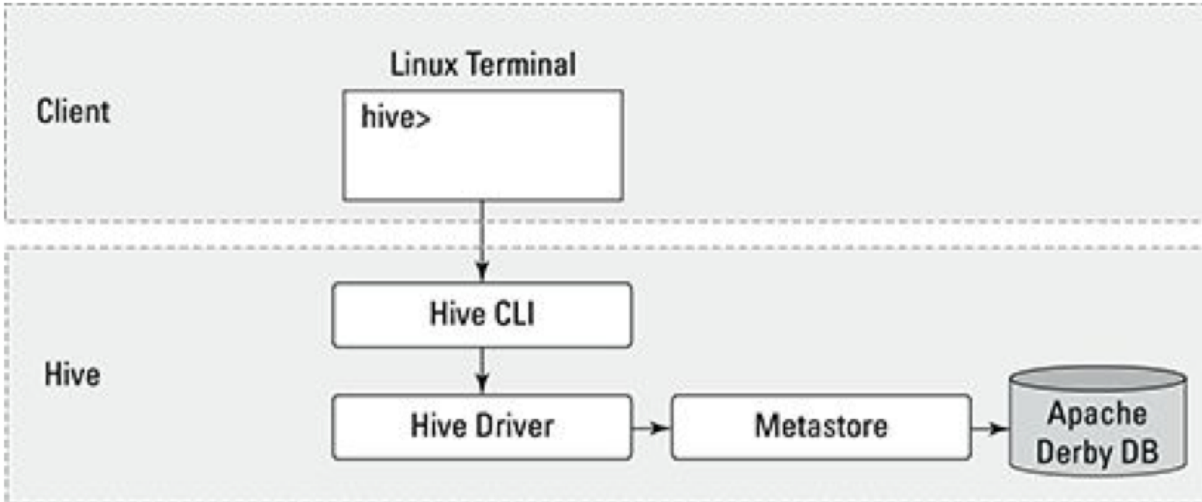
Creating, Dropping, and Altering DBs in Apache Hive

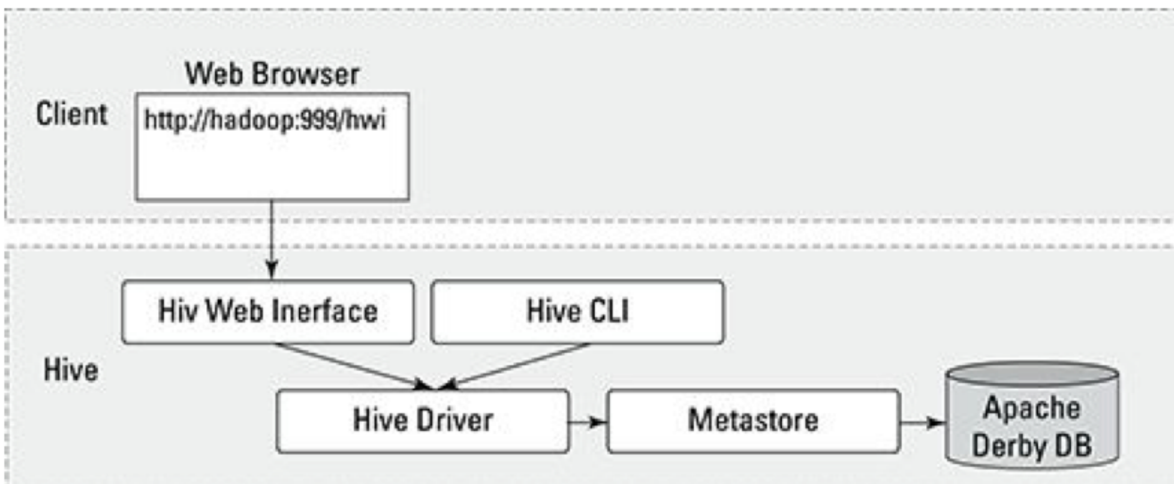
```
(1) $ $HIVE_HOME/bin hive --service cli
 (2) hive> set hive.cli.print.current.db=true;
 (3) hive (default)> USE ourfirstdatabase;
 (4) hive (ourfirstdatabase)> ALTER DATABASE
 ourfirstdatabase SET DBPROPERTIES
 ('creator'='Bruce Brown', 'created_for'='Learning Hive
 DDL');
 OK
 Time taken: 0.138 seconds
 (5) hive (ourfirstdatabase)> DESCRIBE DATABASE
 EXTENDED ourfirstdatabase;
 OK
 ourfirstdatabase                    file:/home/biad
 min/Hive/warehouse/ourfirstdatabase.db  {created_f
 or=Learning Hive DDL, creator=Bruce Brown}
 Time taken: 0.084 seconds, Fetched: 1 row(s)CREATE
 (DATABASE|SCHEMA) [IF NOT EXISTS]
 database_name
 (6) hive (ourfirstdatabase)> DROP DATABASE
 ourfirstdatabase CASCADE;
 OK
 Time taken: 0.132 seconds
```

Another Hive Example

```
(A) CREATE TABLE IF NOT EXISTS FlightInfo2007 (
 Year SMALLINT, Month TINYINT, DayofMonth TINYINT, DayOfWeek TINYINT,
 DepTime SMALLINT, CRSDepTime SMALLINT, ArrTime SMALLINT, CRSArrTime SMALLINT,
 UniqueCarrier STRING, FlightNum STRING, TailNum STRING,
 ActualElapsedTime SMALLINT, CRSElapsedTime SMALLINT,
 AirTime SMALLINT, ArrDelay SMALLINT, DepDelay SMALLINT,
 Origin STRING, Dest STRING,Distance INT,
 TaxiIn SMALLINT, TaxiOut SMALLINT, Cancelled SMALLINT,
 CancellationCode STRING, Diverted SMALLINT,
 CarrierDelay SMALLINT, WeatherDelay SMALLINT,
 NASDelay SMALLINT, SecurityDelay SMALLINT, LateAircraftDelay SMALLINT)
COMMENT 'Flight InfoTable'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
TBLPROPERTIES ('creator'='Bruce Brown', 'created_at'='Thu Sep 19 10:58:00 EDT 2013');
```


Hive's operation modes



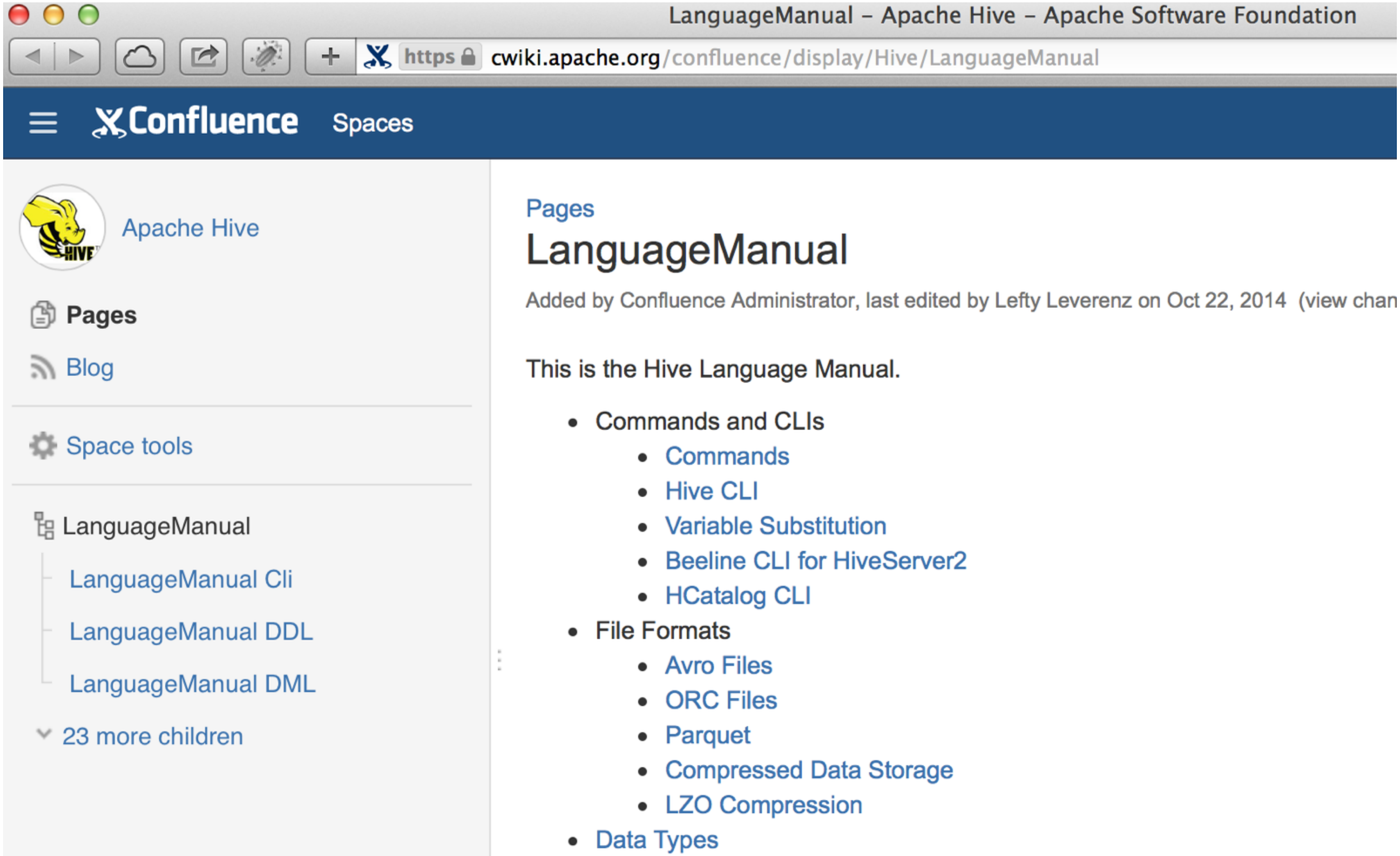


Using HiveQL for Spark SQL

When programming against Spark SQL we have two entry points depending on whether we need Hive support. The recommended entry point is the `HiveContext` to provide access to HiveQL and other Hive-dependent functionality. The more basic `SQLContext` provides a subset of the Spark SQL support that does not depend on Hive. The separation exists for users who might have conflicts with including all of the Hive dependencies. Using a `HiveContext` does not require an existing Hive setup.

HiveQL is the recommended query language for working with Spark SQL. Many resources have been written on HiveQL, including *Programming Hive* and the online Hive Language Manual. In Spark 1.0 and 1.1, Spark SQL is based on Hive 0.12,

Hive Language Manual



Using Spark SQL — Steps and Example

Example 9-5. Python SQL imports

```
# Import Spark SQL
from pyspark.sql import HiveContext, Row
```

Example 9-8. Constructing a SQL context in Python

```
hiveCtx = HiveContext(sc)
```

Example 9-11. Loading and quering tweets in Python

```
input = hiveCtx.jsonFile(inputFile)
# Register the input schema RDD
input.registerTempTable("tweets")
# Select tweets based on the retweetCount
topTweets = hiveCtx.sql("""SELECT text, retweetCount  FROM
  tweets ORDER BY retweetCount LIMIT 10""")
```

Query testtweet.json

Get it from Learning Spark Github ==> https://github.com/databricks/learning-spark/tree/master/files

{"createdAt":"Nov 4, 2014 4:56:59 PM","id":529799371026485248,"text":"Adventures With Coffee, Code, and Writing.","source":"\u003ca href\u003d\"http://twitter.com\" rel\u003d\"nofollow\"\u003eTwitter Web Client\u003c/a\u003e","isTruncated":false,"inReplyToStatusId":-1,"inReplyToUserId":-1,"isFavorited":false,"retweetCount":0,"isPossiblySensitive":false,"contributorsIDs":[],"userMentionEntities":[],"urlEntities":[],"hashtagEntities":[],"mediaEntities":[],"currentUserRetweetId":-1,"user":{"id":15594928,"name":"Holden Karau","screenName":"holdenkarau","location":"","description":"","descriptionURLEntities":[],"isContributorsEnabled":false,"profileImageUrl":"http://pbs.twimg.com/profile_images/3005696115/2036374bbadbed85249cdd50aac6e170_normal.jpeg","profileImageUrlHttps":"https://pbs.twimg.com/profile_images/3005696115/2036374bbadbed85249cdd50aac6e170_normal.jpeg","isProtected":false,"followersCount":1231,"profileBackgroundColor":"C0DEED","profileTextColor":"333333","profileLinkColor":"0084B4","profileSidebarFillColor":"DDEEF6","profileSidebarBorderColor":"FFFFFF","profileUseBackgroundImage":true,"showAllInlineMedia":false,"friendsCount":600,"createdAt":"Aug 5, 2011 9:42:44 AM","favouritesCount":1095,"utcOffset":-3,"profileBackgroundImageUrl":"","profileBackgroundImageUrlHttps":"","profileBannerImageUrl":"","profileBackgroundTiled":true,"lang":"en","statusesCount":6234,"isGeoEnabled":true,"isVerified":false,"translator":false,"listedCount":0,"isFollowRequestSent":false}}

```
>>> print topTweets.collect()
[Row(text=u'Adventures With Coffee, Code, and Writing.', retweetCount=0)]
...
```

SchemaRDD

Both loading data and executing queries return SchemaRDDs. SchemaRDDs are similar to tables in a traditional database. Under the hood, a SchemaRDD is an RDD composed of `Row` objects with additional schema information of the types in each column. `Row` objects are just wrappers around arrays of basic types (e.g., integers and strings).

Row Objects

Row objects represent records inside SchemaRDDs, and are simply fixed-length arrays of fields.

Example 9-14. Accessing the text column in the topTweets SchemaRDD in Python

```
topTweetText = topTweets.map(lambda row: row.text)
```

Spark SQL/HiveQL type	Scala type	Java type	Python
STRUCT<COL1: COL1_TYPE, ...>	Row	Row	Row

Types stored by Schema RDDs

Spark SQL/HiveQL type	Scala type	Java type	Python
TINYINT	Byte	Byte/byte	int/long (in range of – 128 to 127)
SMALLINT	Short	Short/short	int/long (in range of – 32768 to 32767)
INT	Int	Int/int	int or long
BIGINT	Long	Long/long	long
FLOAT	Float	Float/float	float
DOUBLE	Double	Double/double	float
DECIMAL	Scala.math.BigDecimal	Java.math.BigDecimal	decimal.Decimal
STRING	String	String	string
BINARY	Array[Byte]	byte[]	bytearray
BOOLEAN	Boolean	Boolean/boolean	bool
TIMESTAMP	java.sql.TimeStamp	java.sql.TimeStamp	datetime.datetime
ARRAY<DATA_TYPE>	Seq	List	list, tuple, or array
MAP<KEY_TYPE, VAL_TYPE>	Map	Map	dict

Look at the Schema

```
>>> input.printSchema()
root
 |-- contributorsIDs: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- createdAt: string (nullable = true)
 |-- currentUserRetweetId: integer (nullable = true)
 |-- hashtagEntities: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- id: long (nullable = true)
 |-- inReplyToStatusId: integer (nullable = true)
 |-- inReplyToUserId: integer (nullable = true)
 |-- isFavorited: boolean (nullable = true)
 |-- isPossiblySensitive: boolean (nullable = true)
 |-- isTruncated: boolean (nullable = true)
 |-- mediaEntities: array (nullable = true)
 |    |-- element: string (containsNull = false)
 |-- retweetCount: integer (nullable = true)
 |-- source: string (nullable = true)
 |-- text: string (nullable = true)
 |-- urlEntities: array (nullable = true)
 |    |-- element: string (containsNull = false)
```

(not a complete screen shot)

Another way to create SchemaRDD

Example 9-28. Creating a SchemaRDD using Row and named tuple in Python

```
happyPeopleRDD = sc.parallelize([Row(name="holden", favouriteBeverage="coffee")])
happyPeopleSchemaRDD = hiveCtx.inferSchema(happyPeopleRDD)
happyPeopleSchemaRDD.registerTempTable("happy_people")
```

JDBC Server

Spark SQL provides JDBC connectivity, which is useful for connecting business intelligence tools to a Spark cluster and for sharing a cluster across multiple users.

The server can be launched with `sbin/start-thriftserver.sh` in your Spark directory (Example 9-31). This script takes many of the same options as `spark-submit`. By default it listens on `localhost:10000`, but we can change these with either environment variables (`HIVE_SERVER2_THRIFT_PORT` and `HIVE_SERVER2_THRIFT_BIND_HOST`), or with Hive configuration properties (`hive.server2.thrift.port` and `hive.server2.thrift.bind.host`). You can also specify Hive properties on the command line with `--hiveconf property=value`.

Example 9-31. Launching the JDBC server

```
./sbin/start-thriftserver.sh --master sparkMaster
```

Example 9-32. Connecting to the JDBC server with Beeline

```
holden@hmbp2:~/repos/spark$ ./bin/beeline -u jdbc:hive2://localhost:10000
Spark assembly has been built with Hive, including Datanucleus jars on classpath
scan complete in 1ms
Connecting to jdbc:hive2://localhost:10000
Connected to: Spark SQL (version 1.2.0-SNAPSHOT)
```

User-Defined Functions (UDF)

UDFs allow you to register custom functions in Python, Java, and Scala to call within SQL.

This is a very popular way to expose advanced functionality to SQL users in an organization, so that these users can call into it without writing code.

Example 9-36. Python string length UDF

```
# Make a UDF to tell us how long some text is
hiveCtx.registerFunction("strLenPython", lambda x: len(x), IntegerType())
lengthSchemaRDD = hiveCtx.sql("SELECT strLenPython('text') FROM tweets LIMIT 10")
```

Streaming

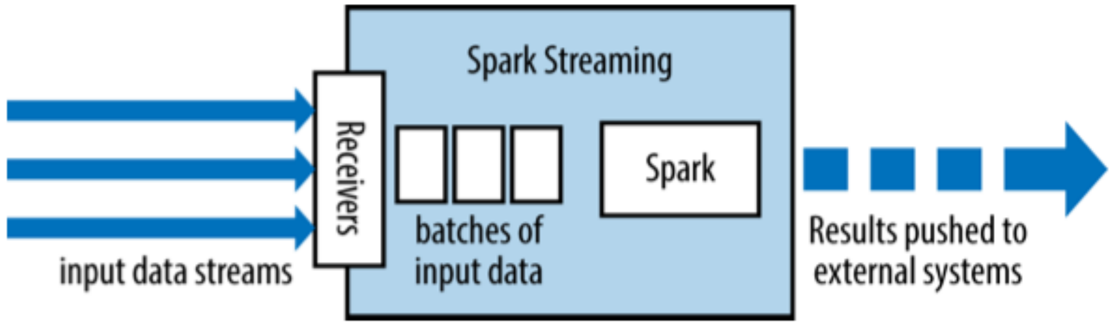
Spark Streaming

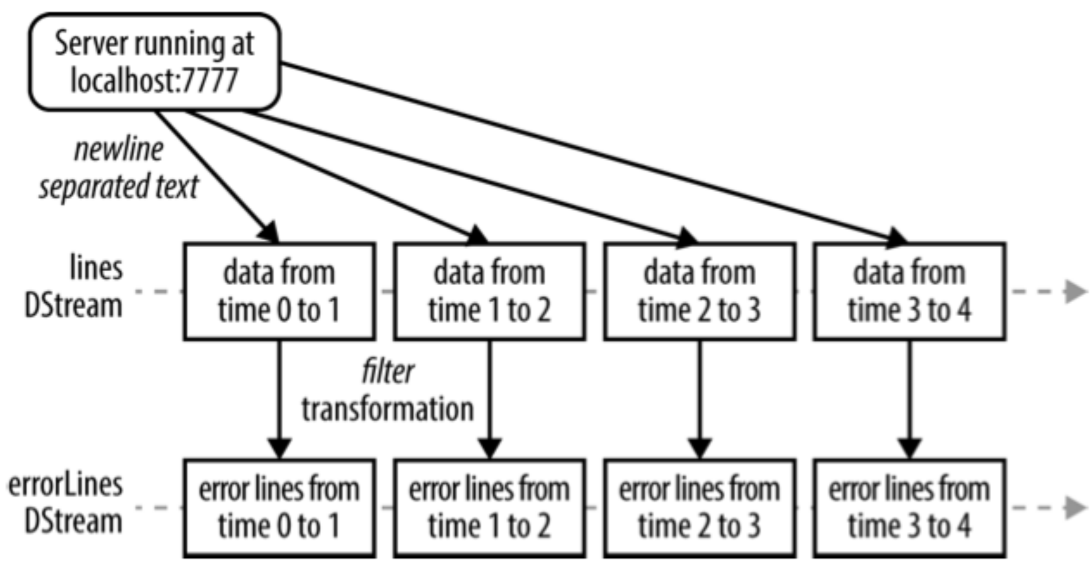
Many applications benefit from acting on data as soon as it arrives. For example, an application might track statistics about page views in real time, train a machine learning model, or automatically detect anomalies. Spark Streaming is Spark's module for such applications. It lets users write streaming applications using a very similar API to batch jobs, and thus reuse a lot of the skills and even code they built for those.

Much like Spark is built on the concept of RDDs, Spark Streaming provides an abstraction called *DStreams*, or *discretized streams*. A DStream is a sequence of data arriving over time. Internally, each DStream is represented as a sequence of RDDs arriving at each time step (hence the name "discretized").

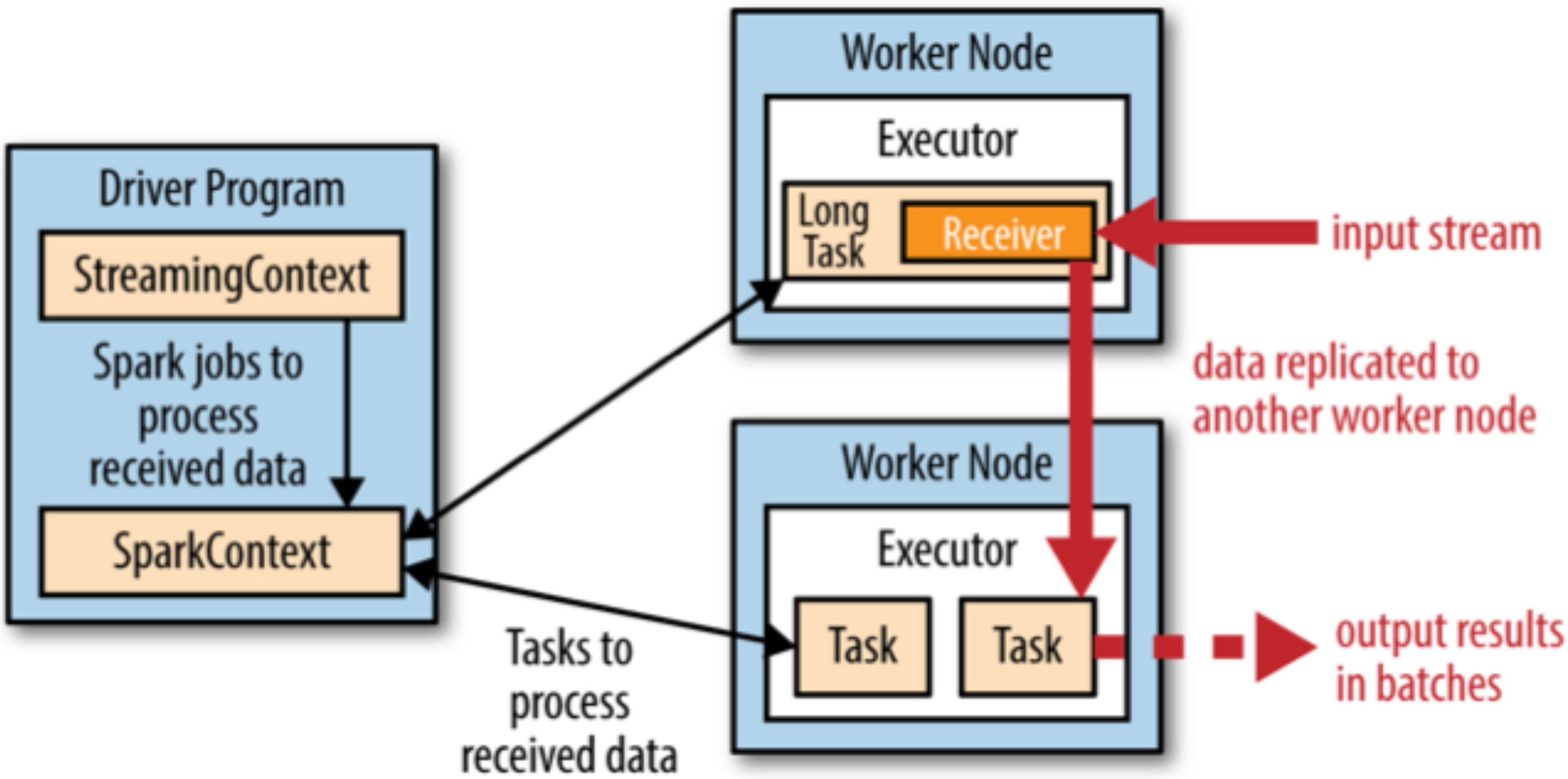
In Spark 1.1, Spark Streaming is available only in Java and Scala.
Spark 1.2 has limited Python support.

Spark Streaming architecture

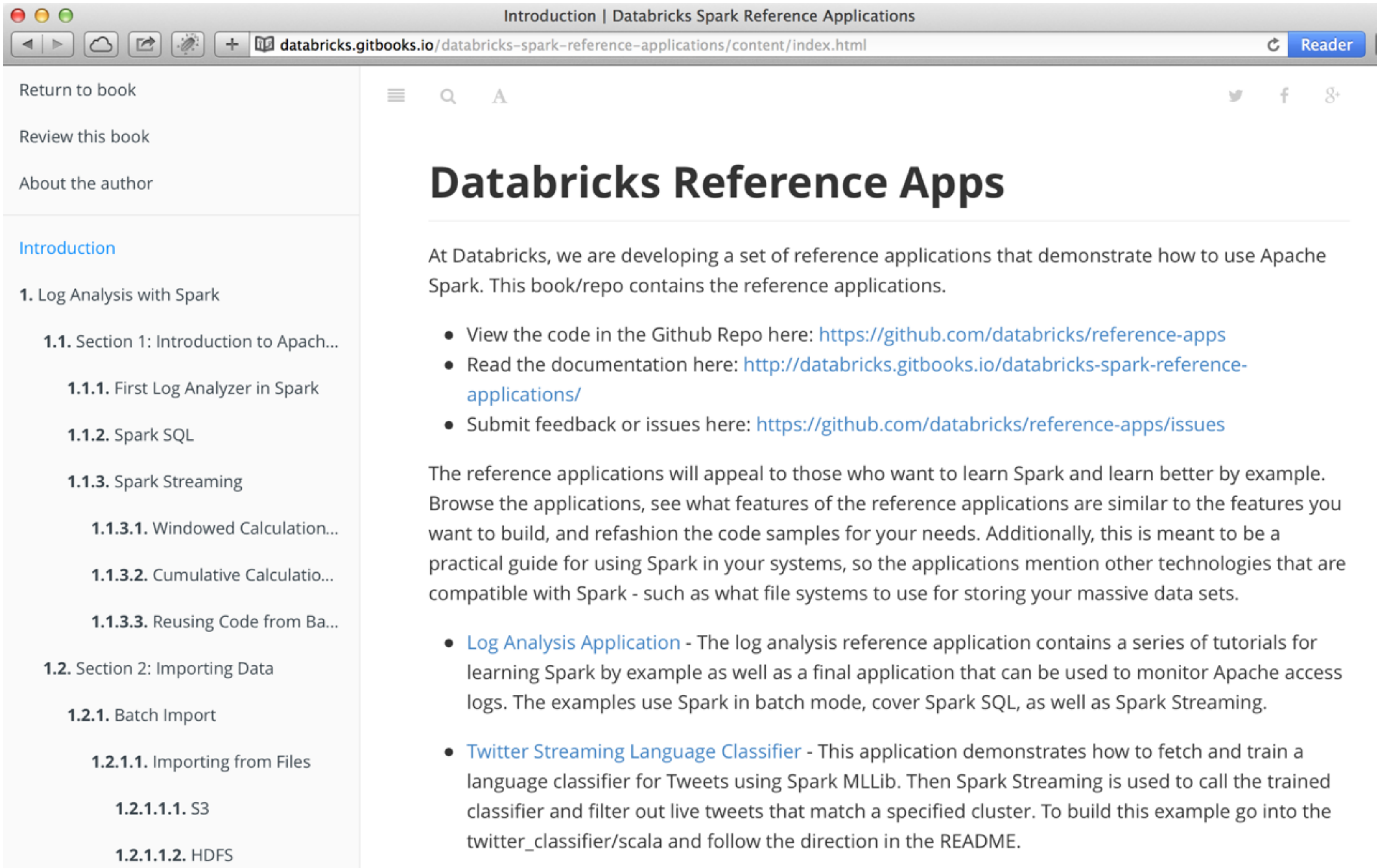




Spark Streaming with Spark's components



Try these examples



Introduction | Databricks Spark Reference Applications

databricks.gitbooks.io/databricks-spark-reference-applications/content/index.html

Reader

Databricks Reference Apps

At Databricks, we are developing a set of reference applications that demonstrate how to use Apache Spark. This book/repo contains the reference applications.

- View the code in the Github Repo here: https://github.com/databricks/reference-apps
- Read the documentation here: http://databricks.gitbooks.io/databricks-spark-reference-applications/
- Submit feedback or issues here: https://github.com/databricks/reference-apps/issues

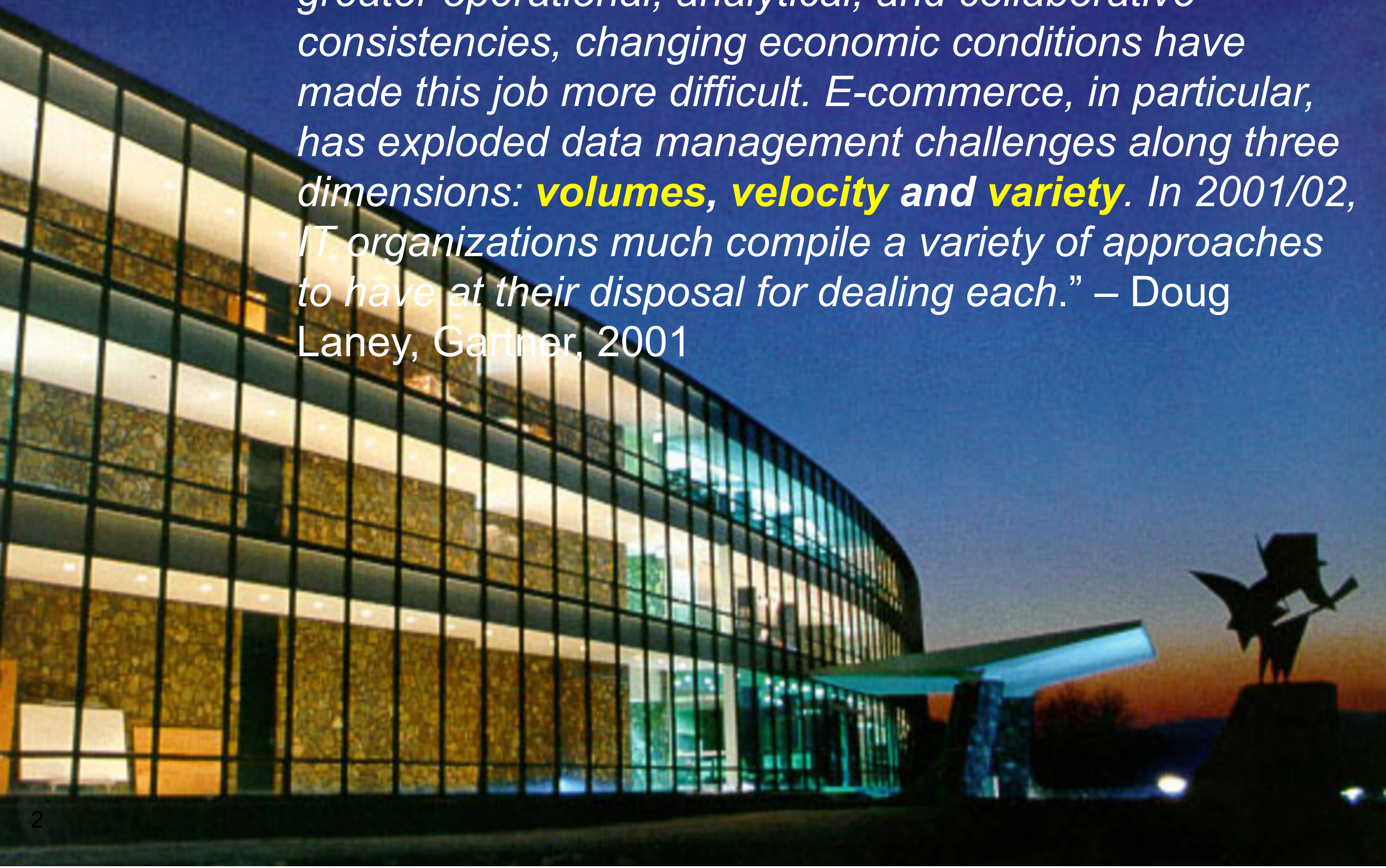
The reference applications will appeal to those who want to learn Spark and learn better by example. Browse the applications, see what features of the reference applications are similar to the features you want to build, and refashion the code samples for your needs. Additionally, this is meant to be a practical guide for using Spark in your systems, so the applications mention other technologies that are compatible with Spark - such as what file systems to use for storing your massive data sets.

- Log Analysis Application - The log analysis reference application contains a series of tutorials for learning Spark by example as well as a final application that can be used to monitor Apache access logs. The examples use Spark in batch mode, cover Spark SQL, as well as Spark Streaming.

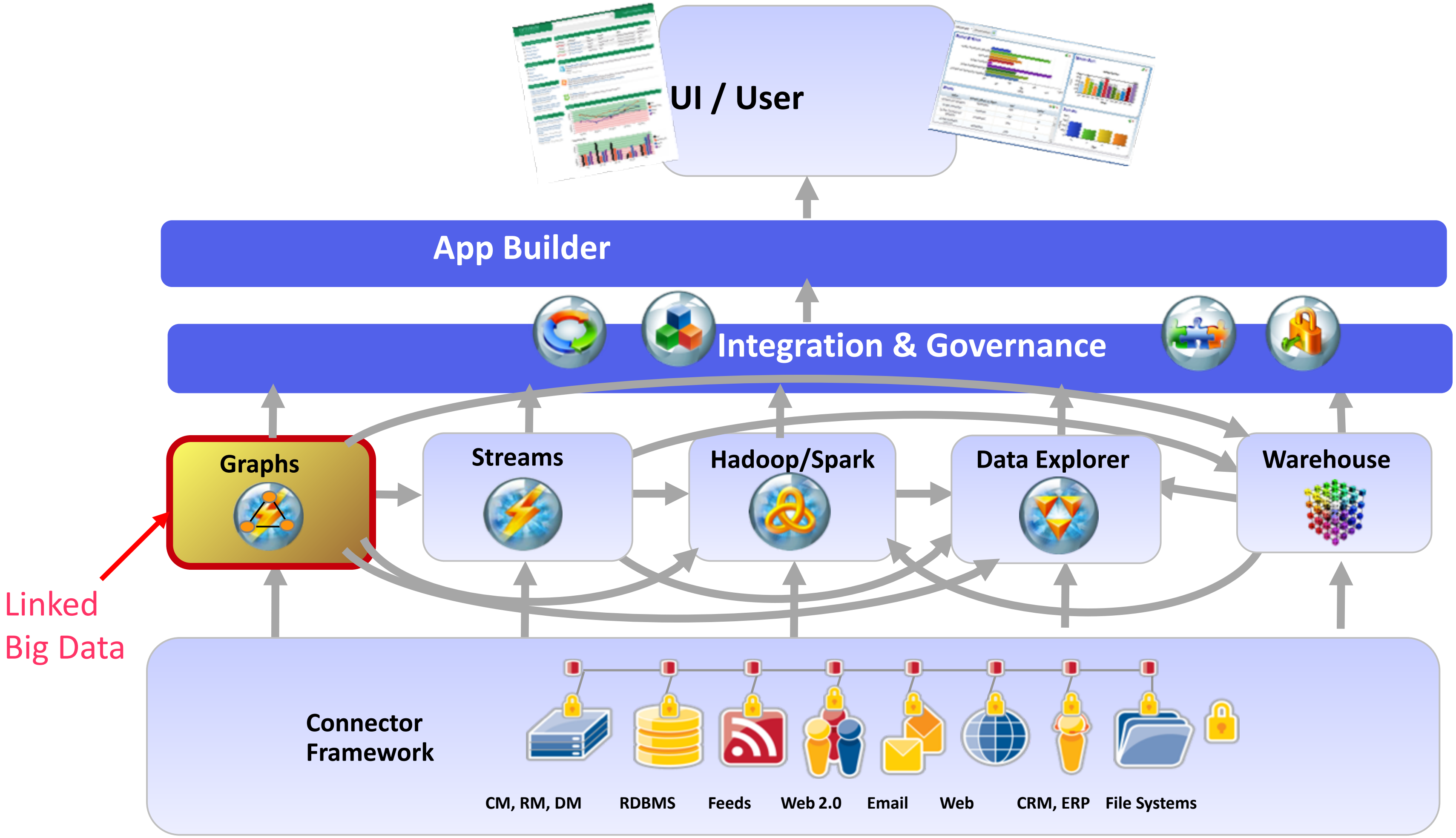
- Twitter Streaming Language Classifier - This application demonstrates how to fetch and train a language classifier for Tweets using Spark MLLib. Then Spark Streaming is used to call the trained classifier and filter out live tweets that match a specified cluster. To build this example go into the twitter_classifier/scala and follow the direction in the README.

Graph Database

Big Data: "*While enterprises struggle to consolidate systems and collapse redundant databases to enable greater operational, analytical, and collaborative consistencies, changing economic conditions have made this job more difficult. E-commerce, in particular, has exploded data management challenges along three dimensions:* ***volumes, velocity and variety****. In 2001/02, IT organizations much compile a variety of approaches to have at their disposal for dealing each.*" – Doug Laney, Gartner, 2001



Graph is a missing pillar in the existing Big Data foundation



Graph Computing is difficult because data cannot be easily partitioned

Graph Database Example

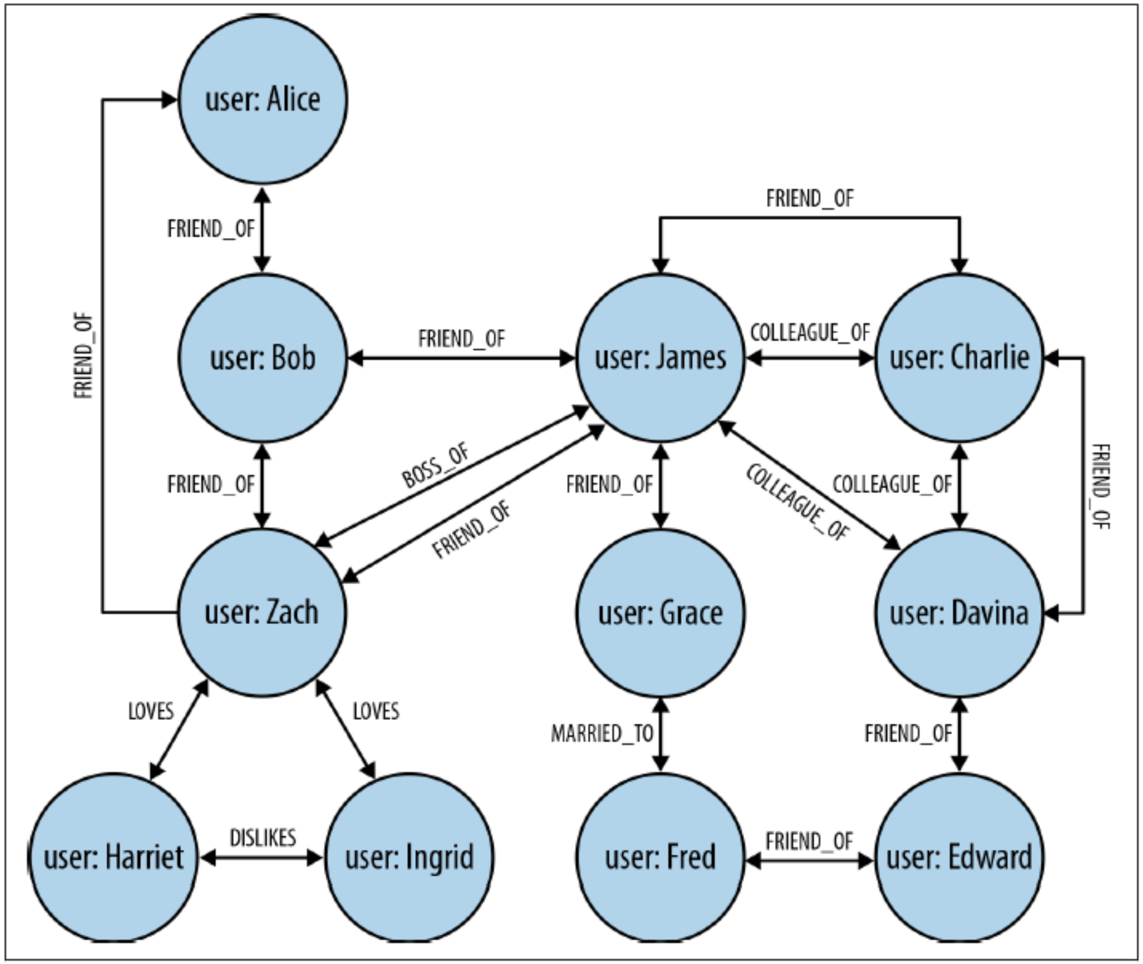
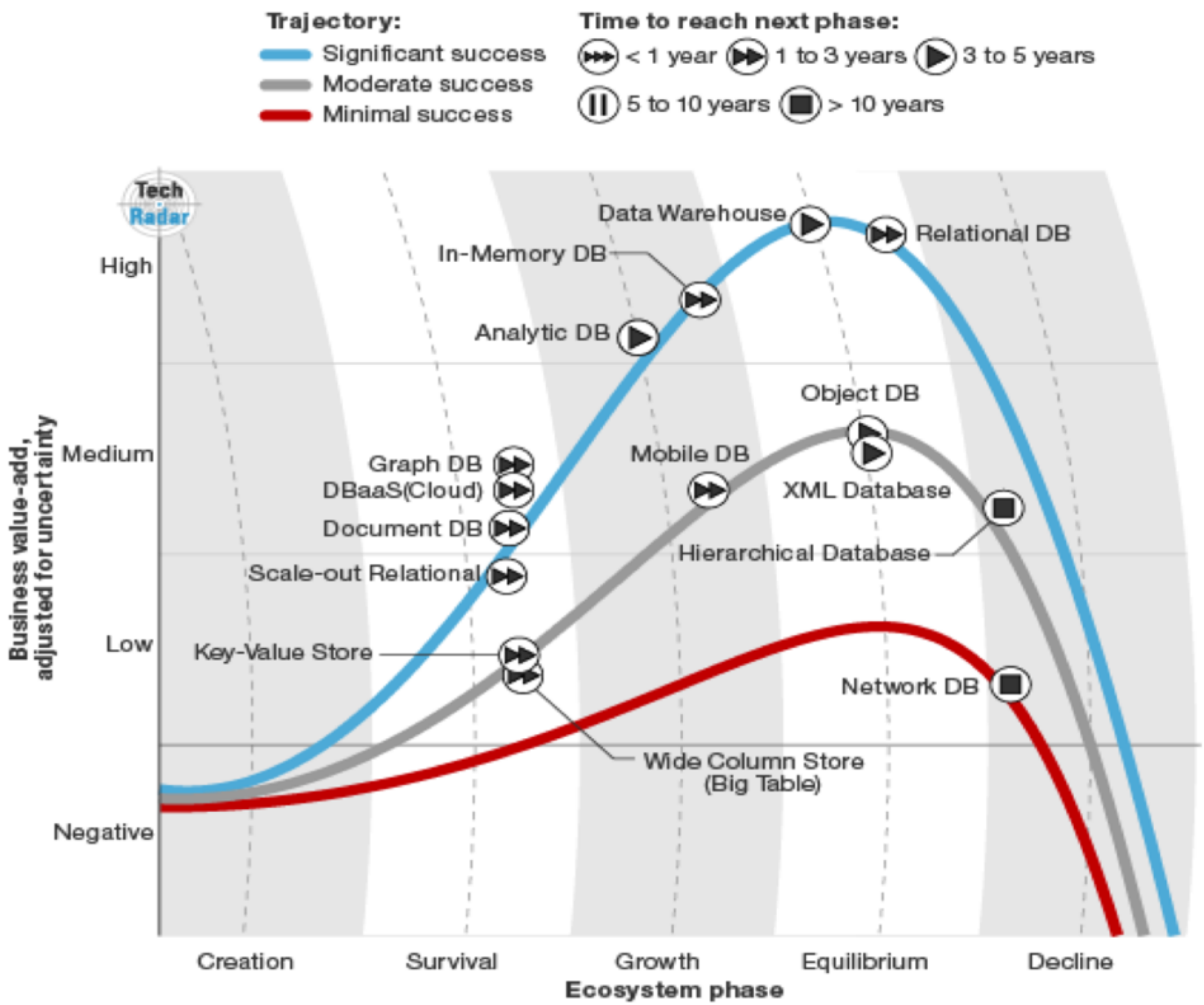


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

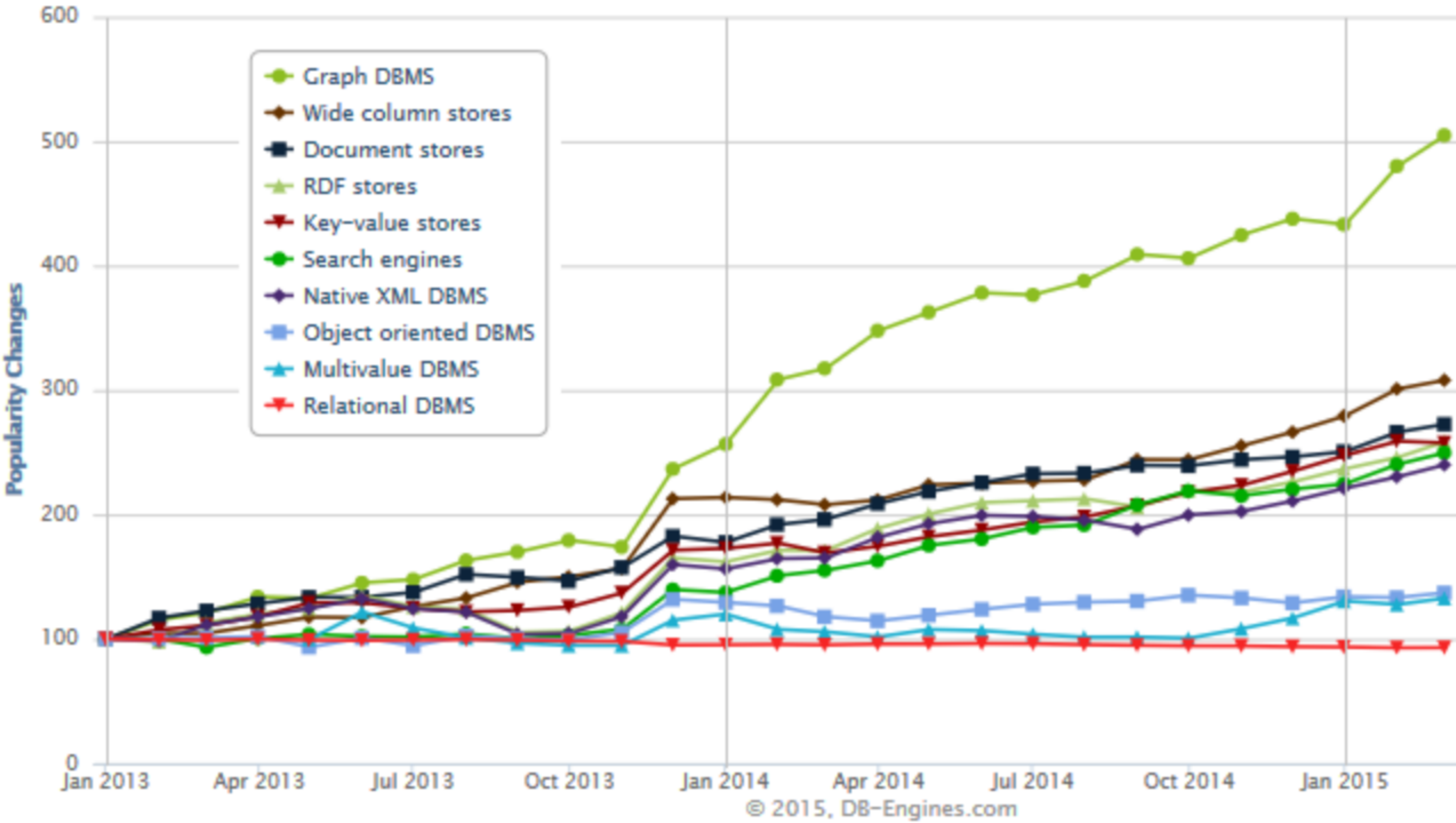
Forrester: Over 25% of enterprise will use Graph DB by 2017

TechRadar: Enterprise DBMS, Q12014

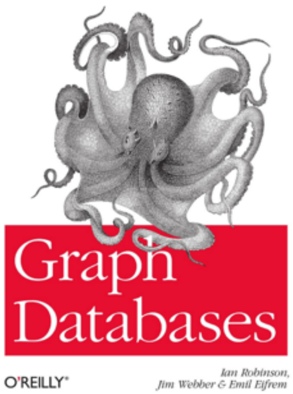


Graph DB is in the significant success trajectory, and with the highest business value in the upcoming DBs.

GraphDB has the largest Popularity Change among DBMS lately



Graph Database key differentiator — native store



In Relational DB, relationships are *distributed and stored as tables*

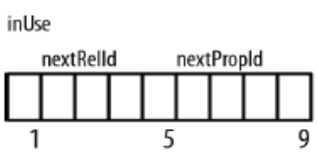
User					
UserID	**User**	**Address**	**Phone**	**Email**	**Alternate**
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

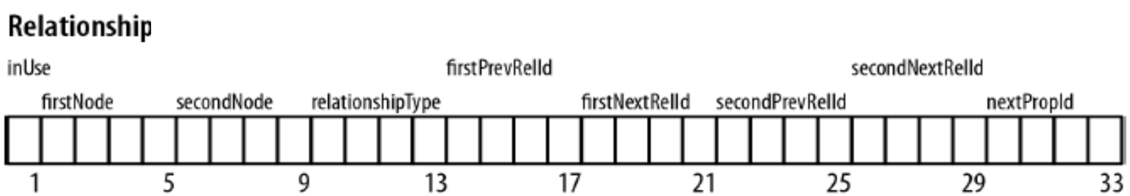
Order	
OrderID	**UserID**
1234	1
5678	1
...	...
5588	99

LineItem		
OrderID	**ProductID**	**Quantity**
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product		
ProductID	**Description**	**Handling**
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Native Graph DB stores nodes and relationships directly, It makes retrieval efficient.





*Retrieving multi-step relationships is a '**graph traversal**' problem*

Technology ==> Top Layer: Graph, Bottom Layer: Graph

Cited "Graph Database" O'liey 2013

A usual example

User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Figure 2-1. Semantic relationships are hidden in a relational database

Query Example – I

Person	
ID	**Person**
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	**FriendID**
1	2
2	1
2	99
...	...
99	1

Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking "who are Bob's friends?" is easy, as shown in Example 2-1.

Example 2-1. Bob's friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Query Examples – II & III

Example 2-2. Who is friends with Bob?

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.PersonID = p1.ID
JOIN Person p2
  ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice's friends-of-friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Computational intensive

Execution Time in the example of finding extended friends (by Neo4j)

Partner and Vukotic's experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Modeling Order History as a Graph

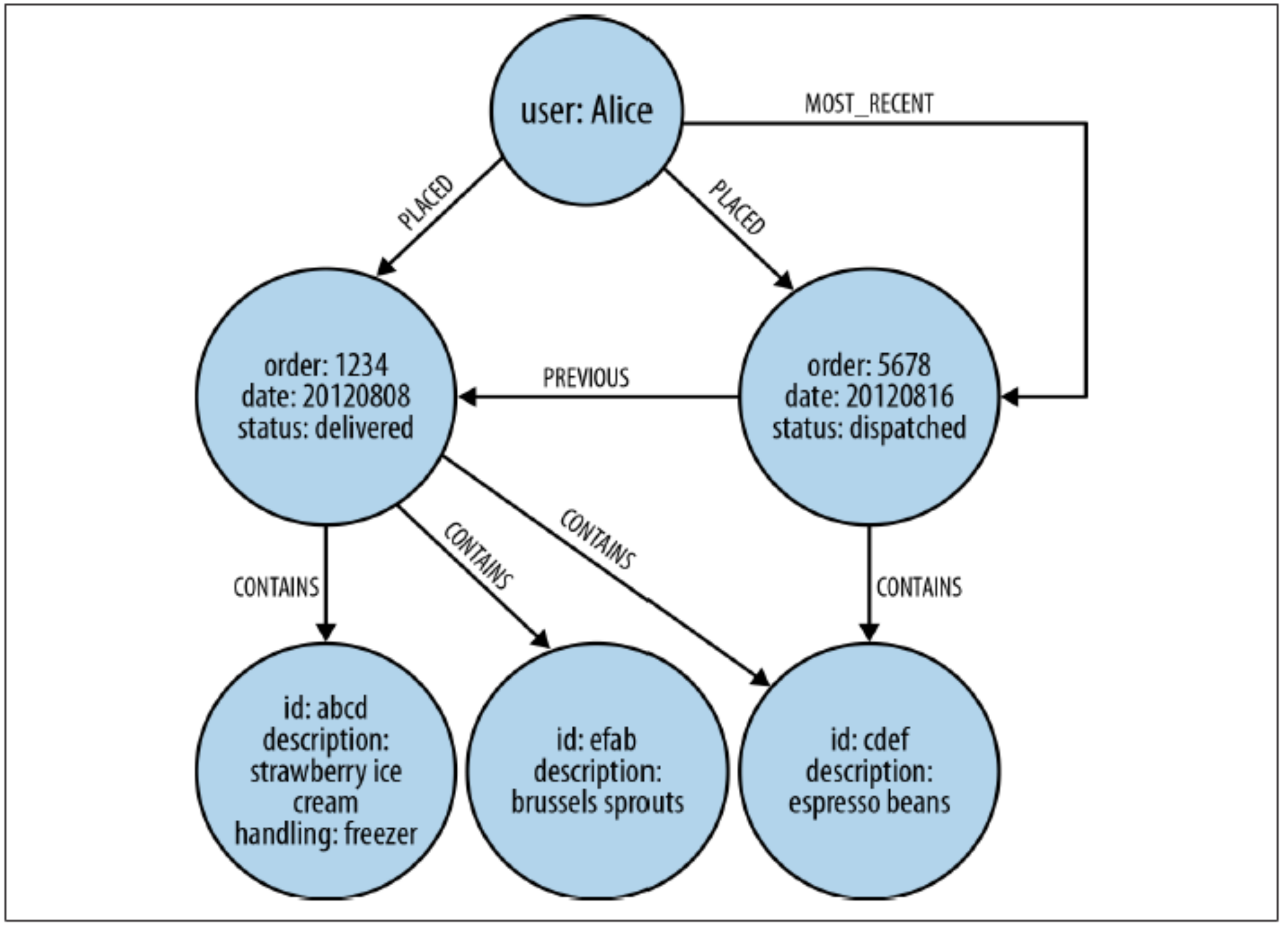


Figure 2-6. Modeling a user's order history in a graph

A query language on Property Graph – Cypher

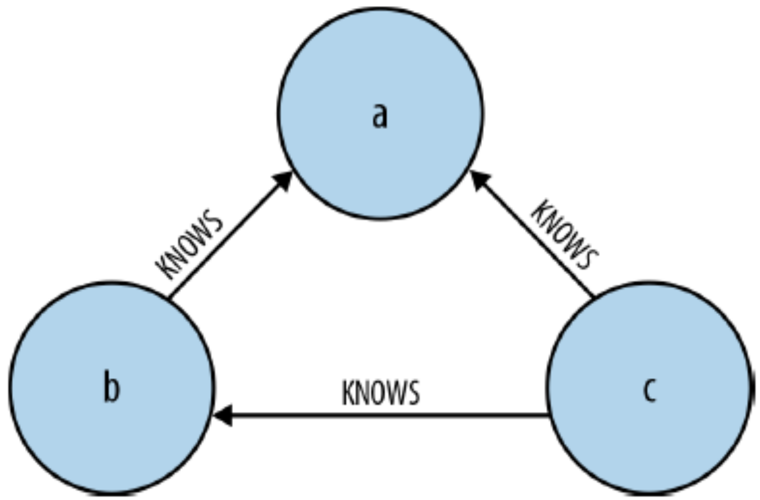


Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here's the equivalent ASCII art representation in Cypher:

```
(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
```

Property Graph Example – Shakespeare

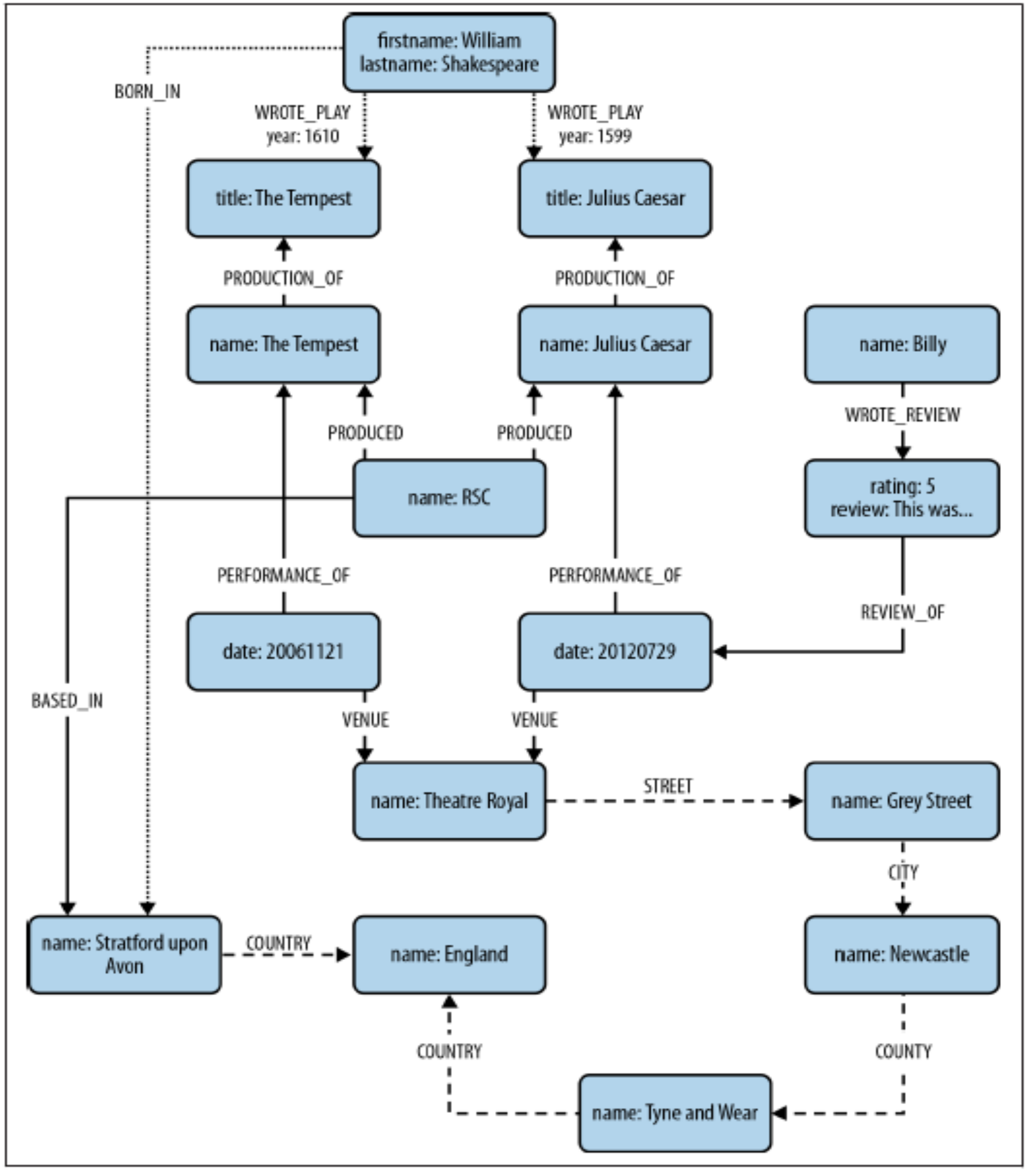


Figure 3-6. Three domains in one graph

Creating the Shakespeare Graph

```
CREATE (shakespeare { firstname: 'William', lastname: 'Shakespeare' }),
       (juliusCaesar { title: 'Julius Caesar' }),
       (shakespeare)-[:WROTE_PLAY { year: 1599 }]->(juliusCaesar),
       (theTempest { title: 'The Tempest' }),
       (shakespeare)-[:WROTE_PLAY { year: 1610}]->(theTempest),
       (rsc { name: 'RSC' }),
       (production1 { name: 'Julius Caesar' }),
       (rsc)-[:PRODUCED]->(production1),
       (production1)-[:PRODUCTION_OF]->(juliusCaesar),
       (performance1 { date: 20120729 }),
       (performance1)-[:PERFORMANCE_OF]->(production1),
       (production2 { name: 'The Tempest' }),
       (rsc)-[:PRODUCED]->(production2),
       (production2)-[:PRODUCTION_OF]->(theTempest),
       (performance2 { date: 20061121 }),
       (performance2)-[:PERFORMANCE_OF]->(production2),
       (performance3 { date: 20120730 }),
       (performance3)-[:PERFORMANCE_OF]->(production1),
       (billy { name: 'Billy' }),
       (review { rating: 5, review: 'This was awesome!' }),
       (billy)-[:WROTE_REVIEW]->(review),
       (review)-[:RATED]->(performance1),
       (theatreRoyal { name: 'Theatre Royal' }),
       (performance1)-[:VENUE]->(theatreRoyal),
       (performance2)-[:VENUE]->(theatreRoyal),
       (performance3)-[:VENUE]->(theatreRoyal),
       (greyStreet { name: 'Grey Street' }),
       (theatreRoyal)-[:STREET]->(greyStreet),
       (newcastle { name: 'Newcastle' }),
       (greyStreet)-[:CITY]->(newcastle),
       (tyneAndWear { name: 'Tyne and Wear' }),
       (newcastle)-[:COUNTY]->(tyneAndWear),
       (england { name: 'England' }),
       (tyneAndWear)-[:COUNTRY]->(england),
       (stratford { name: 'Stratford upon Avon' }),
       (stratford)-[:COUNTRY]->(england),
       (rsc)-[:BASED_IN]->(stratford),
       (shakespeare)-[:BORN_IN]->stratford
```

Query on the Shakespeare Graph

```
START theater=node:venue(name='Theatre Royal'),
      newcastle=node:city(name='Newcastle'),
      bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[:STREET|CITY*1..2]-(theater)
      <-[:VENUE]-()-[:PERFORMANCE_OF]->()-[:PRODUCTION_OF]->
      (play)<-[w:WROTE_PLAY]-(bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play
```

Adding this `WHERE` clause means that for each successful match, the Cypher execution engine checks that the `WROTE_PLAY` relationship between the Shakespeare node and the matched play has a `year` property with a value greater than 1608. Matches with a `WROTE_PLAY` relationship whose `year` value is greater than 1608 will pass the test; these plays will then be included in the results. Matches that fail the test will not be included in the results. By adding this clause, we ensure that only plays from Shakespeare's late period are returned:

```
+---------------+
| play          |
+---------------+
| "The Tempest" |
+---------------+
1 row
```

Building Application Example – Collaborative Filtering

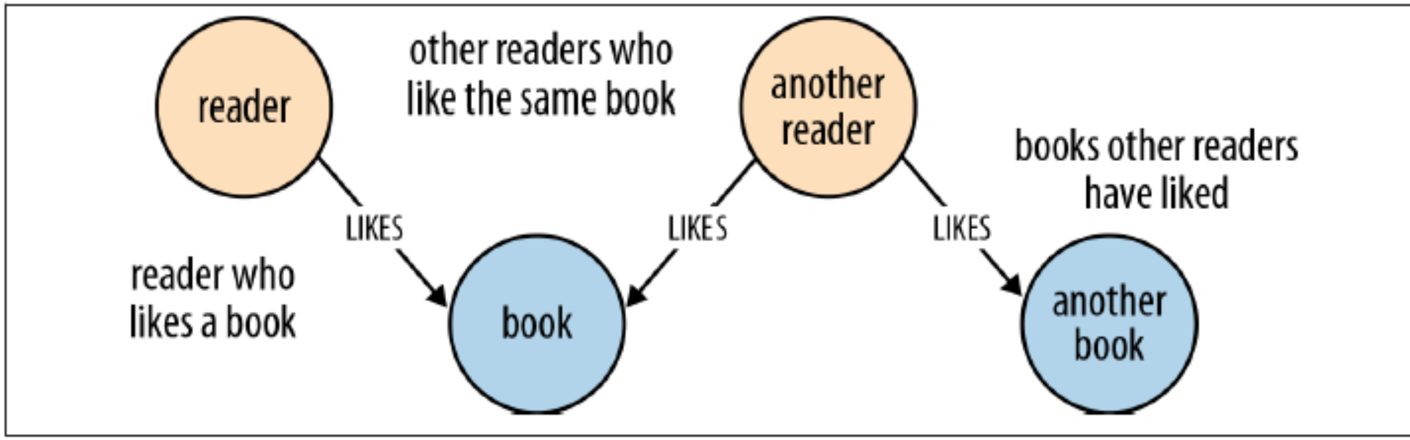


Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```
START reader=node:users(name={readerName})
      book=node:books(isbn={bookISBN})
MATCH reader-[:LIKES]->book<-[:LIKES]-other_readers-[:LIKES]->books
RETURN books.title
```

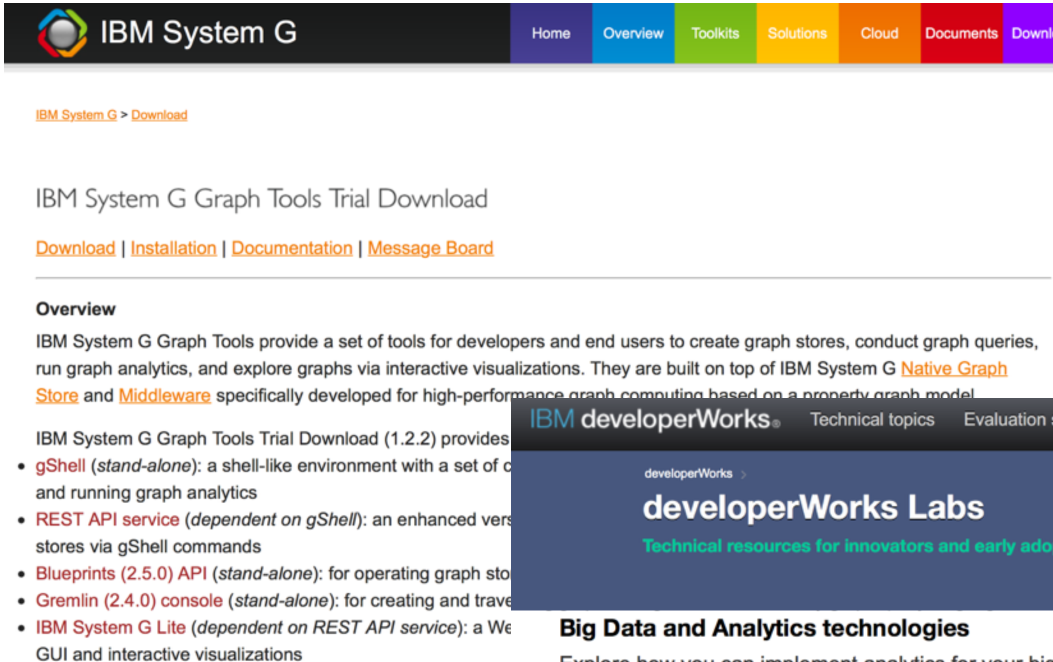
What is IBM System G?

http://systemG.research.ibm.com (Internet) or http://systemG.ibm.com (IBM internal site)

A Complete Graph Computing Suite — Toolkits, Solutions, & Cloud

Download IBM System G Standard Edition (on-premise)

http://systemg.research.ibm.com/download.html


IBM System G
Home | Overview | Toolkits | Solutions | Cloud | Documents | Downlo

IBM System G > Download

IBM System G Graph Tools Trial Download

Download | Installation | Documentation | Message Board

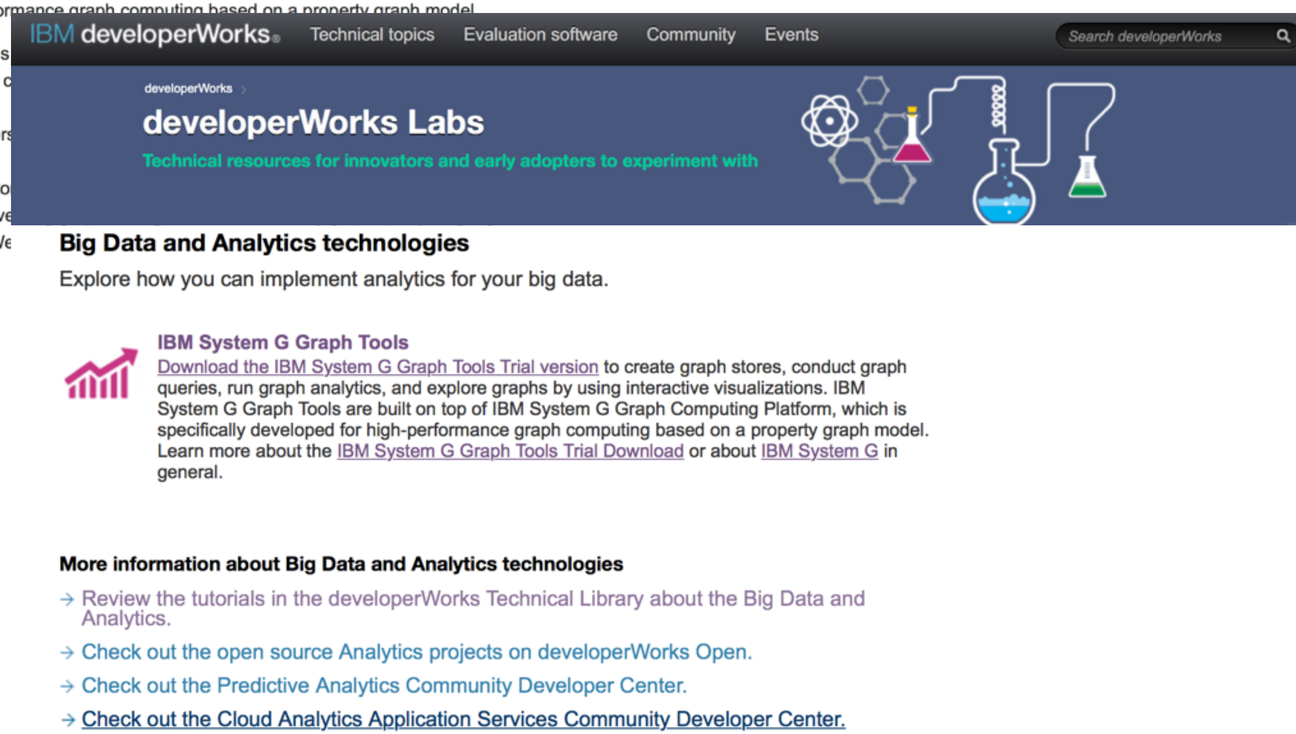
Overview

IBM System G Graph Tools provide a set of tools for developers and end users to create graph stores, conduct graph queries, run graph analytics, and explore graphs via interactive visualizations. They are built on top of IBM System G Native Graph Store and Middleware specifically developed for high-perfor

IBM System G Graph Tools Trial Download (1.2.2) provides

- gShell (*stand-alone*): a shell-like environment with a set of c and running graph analytics
- REST API service (*dependent on gShell*): an enhanced vers stores via gShell commands
- Blueprints (2.5.0) API (*stand-alone*): for operating graph sto
- Gremlin (2.4.0) console (*stand-alone*): for creating and trave
- IBM System G Lite (*dependent on REST API service*): a We GUI and interactive visualizations

or
http://www.ibm.com/developerworks/labs/


IBM developerWorks | Technical topics | Evaluation software | Community | Events | Search developerWorks

developerWorks >

developerWorks Labs

Technical resources for innovators and early adopters to experiment with

Big Data and Analytics technologies

Explore how you can implement analytics for your big data.

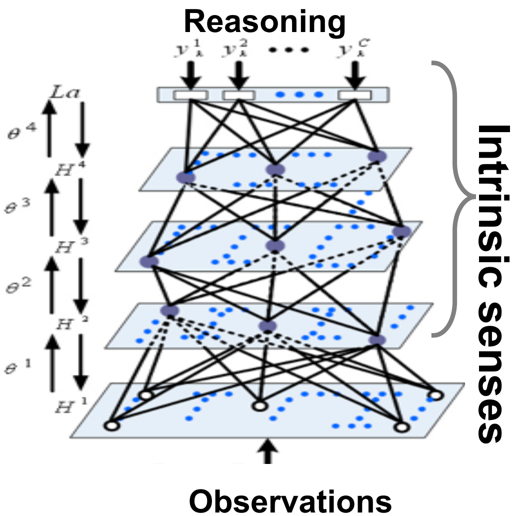
IBM System G Graph Tools

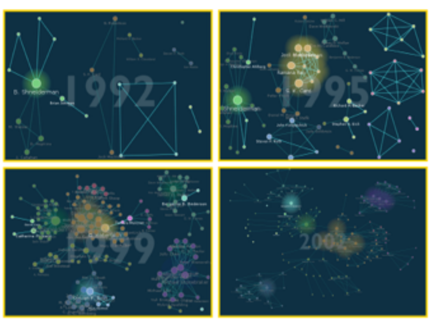
Download the IBM System G Graph Tools Trial version to create graph stores, conduct graph queries, run graph analytics, and explore graphs by using interactive visualizations. IBM System G Graph Tools are built on top of IBM System G Graph Computing Platform, which is specifically developed for high-performance graph computing based on a property graph model. Learn more about the IBM System G Graph Tools Trial Download or about IBM System G in general.

More information about Big Data and Analytics technologies

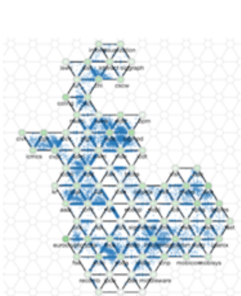
- → Review the tutorials in the developerWorks Technical Library about the Big Data and Analytics.
- → Check out the open source Analytics projects on developerWorks Open.
- → Check out the Predictive Analytics Community Developer Center.
- → Check out the Cloud Analytics Application Services Community Developer Center.

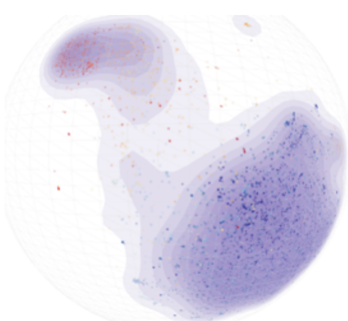
IBM System G Tools — 8 categories

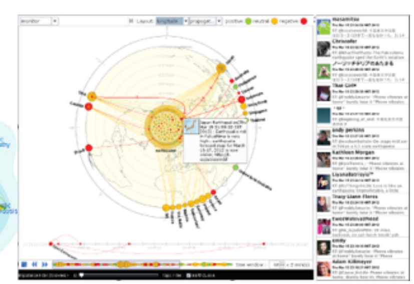




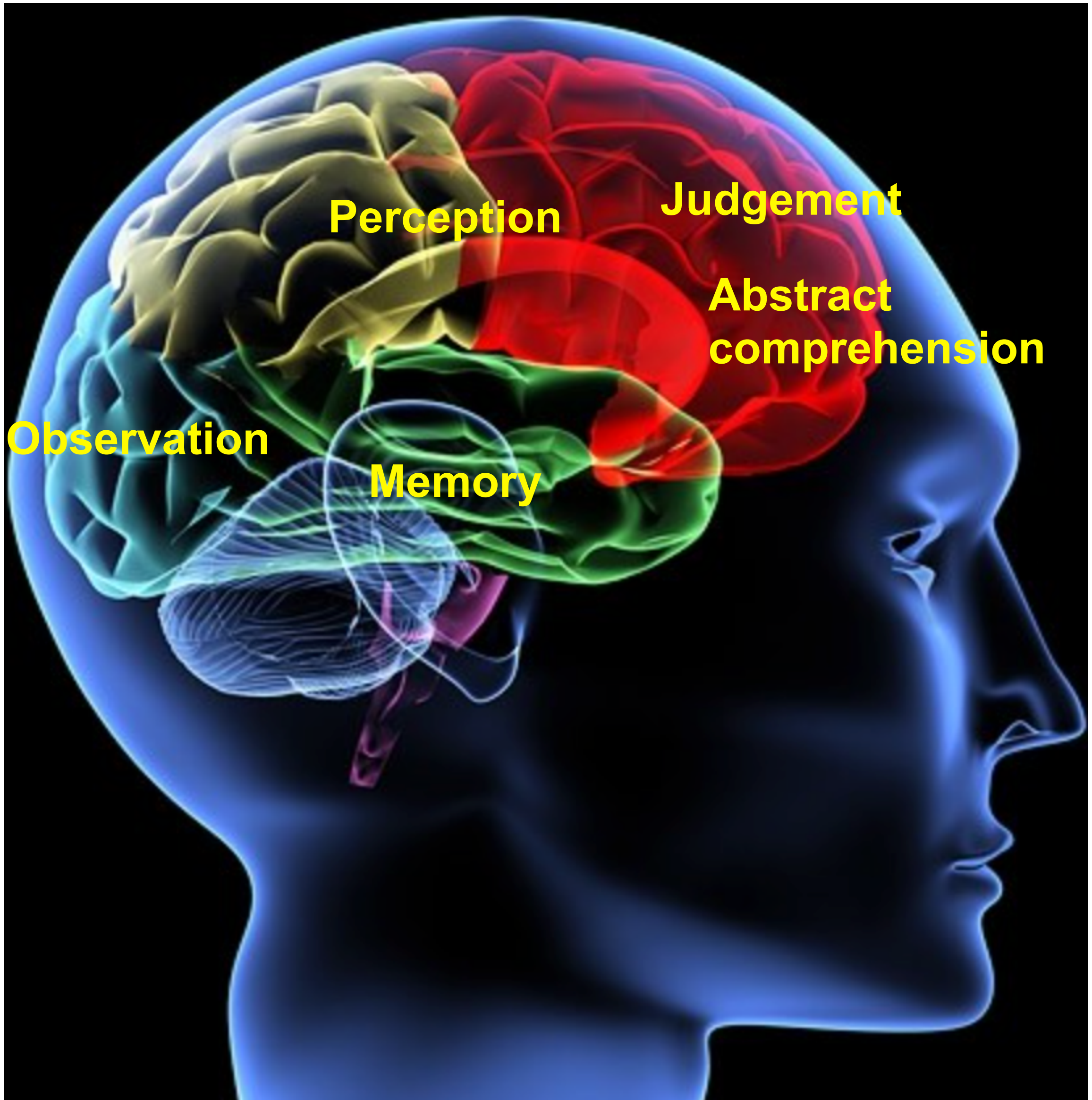
- Graph Database:
  - Native Store
  - GBase
- Scalable Middleware:
  - Parallel Prog. Lib.
  - Power Optimization
  - Software Defined Env.
- Contextual Analytics:
  - Topological Analysis
  - Matching and Search
  - Path and Flow
- Visual Analytics:
  - Multivariate Graph
  - Heterogeneous Graph
  - Dynamic Graph
  - Big Graph
- Reasoning Engine:
  - Markovian & Bayesian Networks
  - Anomaly Detection Tools
  - Brain Analysis Tool
- Cognitive Networks:
  - Deep Learning
  - Emotion Analysis
- Spatiotemporal Analytics:
  - Road Network Algorithms
  - Spatiotemporal Data Mining
  - Spatiotemporal Indexing
- Mobile & Sensor Analytics:
  - Mobile Security Tools
  - Sensor Analytics Tools



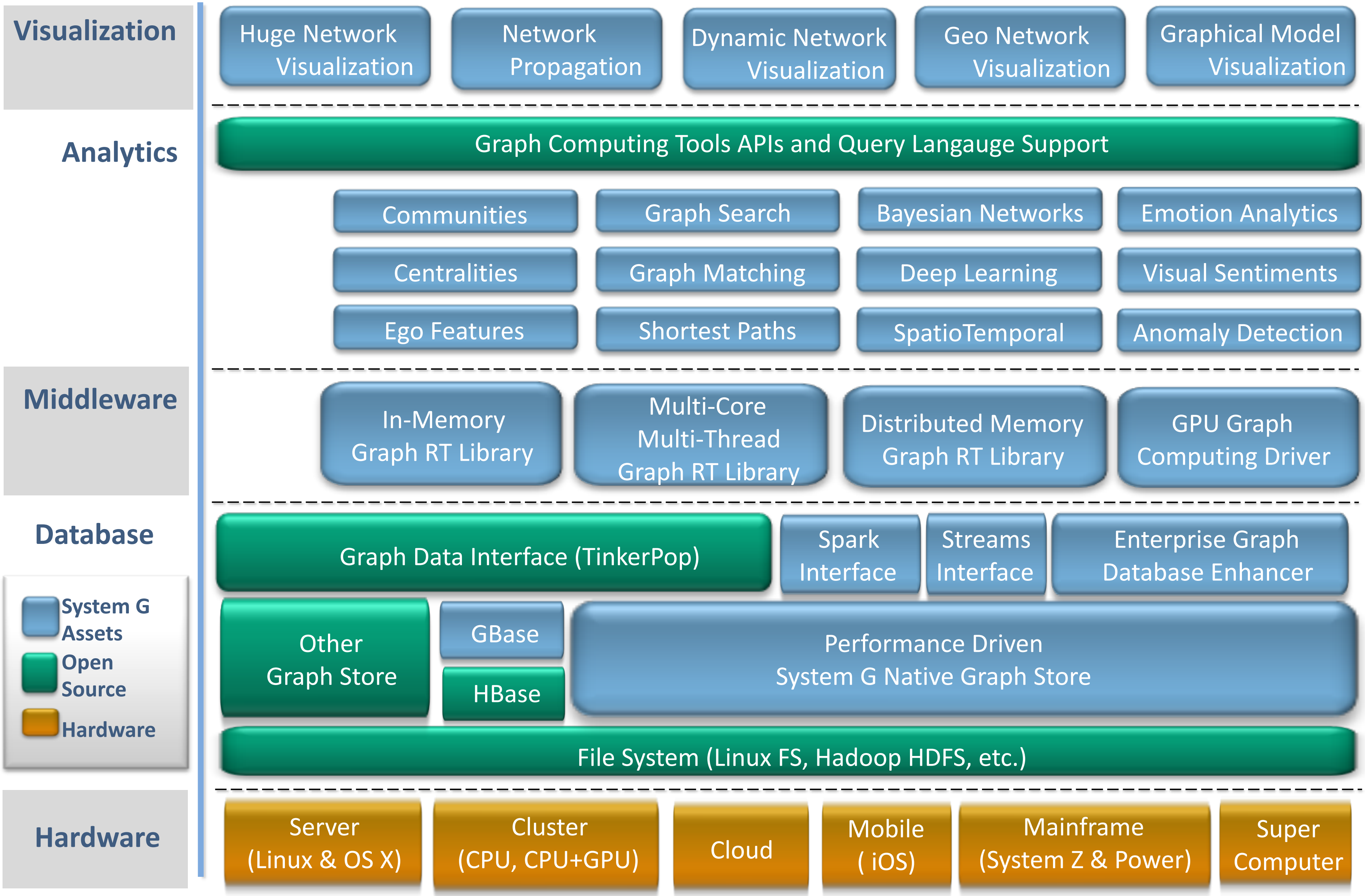




Similarity to the Brain Functions and Evolution

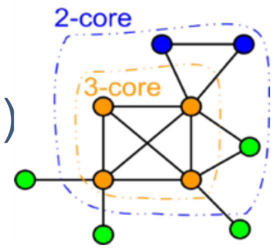


IBM System G Graph Computing Tools

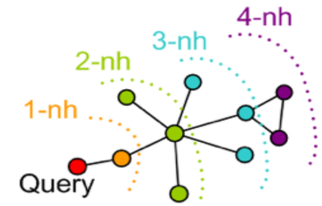


Graph Analytical Tools

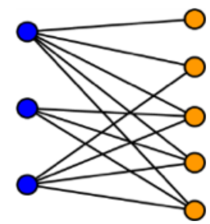
- ***Network topological analysis*** tools
  - •Centralities (degree, closeness, betweenness)
  - •PageRank
  - •Communities (connected component, K-core, triangle count, clustering coefficient)
  - •Neighborhood (egonet, K-neighborhood)
- ***Graph matching and search*** tools
  - •Graph search/filter by label, vertex/edge properties (including geo locations)
  - •Graph matching
  - •Collaborative filtering
- ***Graph path and flow*** tools
  - •Shortest paths
  - •Top K-shortest paths
- ***Probabilistic graphical model*** tools
  - •Bayesian network inference
  - •Deep learning

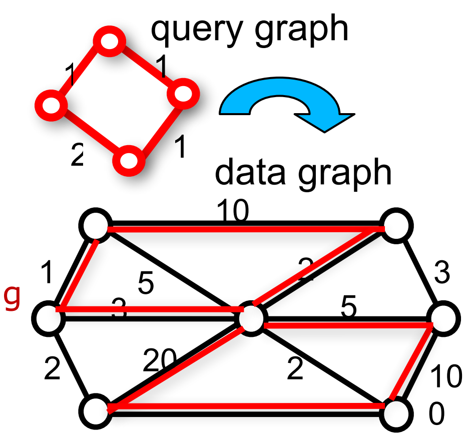


K-core

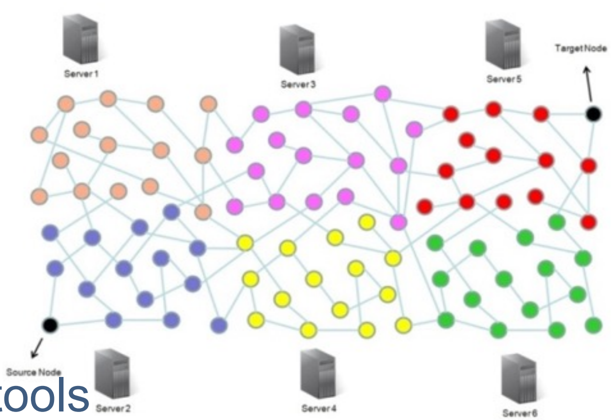


K-neighborhood


Collaborative filtering
Bipartite weighted graph matching

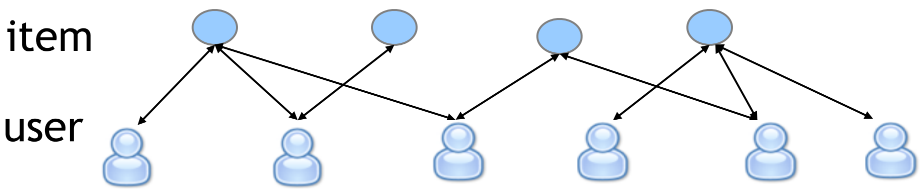


Graph matching

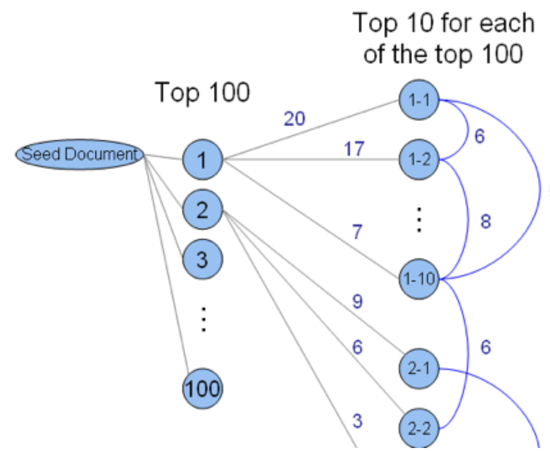

Top k-shortest paths


Bayesian network inference

Performance comparisons



People who bought this also bought that..



Recommendation ==> 2-hop traversal & ranking

For Visualization ==> 4-hop traversal & rankings

IBM KnowledgeView 1-year Access Log: 72.3K users, 82.1K docs, and 1.74 million downloads

Query Time (sec) / App. Type	DB2 via SQL	Oracle via SQL	DB2RDF via SPARQL	Neo4j	Titan (Berk.DB)	Titan (HBase)	System G GBase	System G Native Store
Recommendation	0.24	0.35	TBD	0.068	0.281	0.414	0.201	0.015
Visualization	52.0 (cold) 50.6 (cache)	201.0 (cold) 42.0 (cache)	TBD	4.8 (cold) 1.2 (cache)	17.3 (cold) 6.8 (cache)	24.2 (cold) 5.7 (cache)	27.0 (cold) 2.4 (cache)	4.2 (cold) 0.07 (cache)

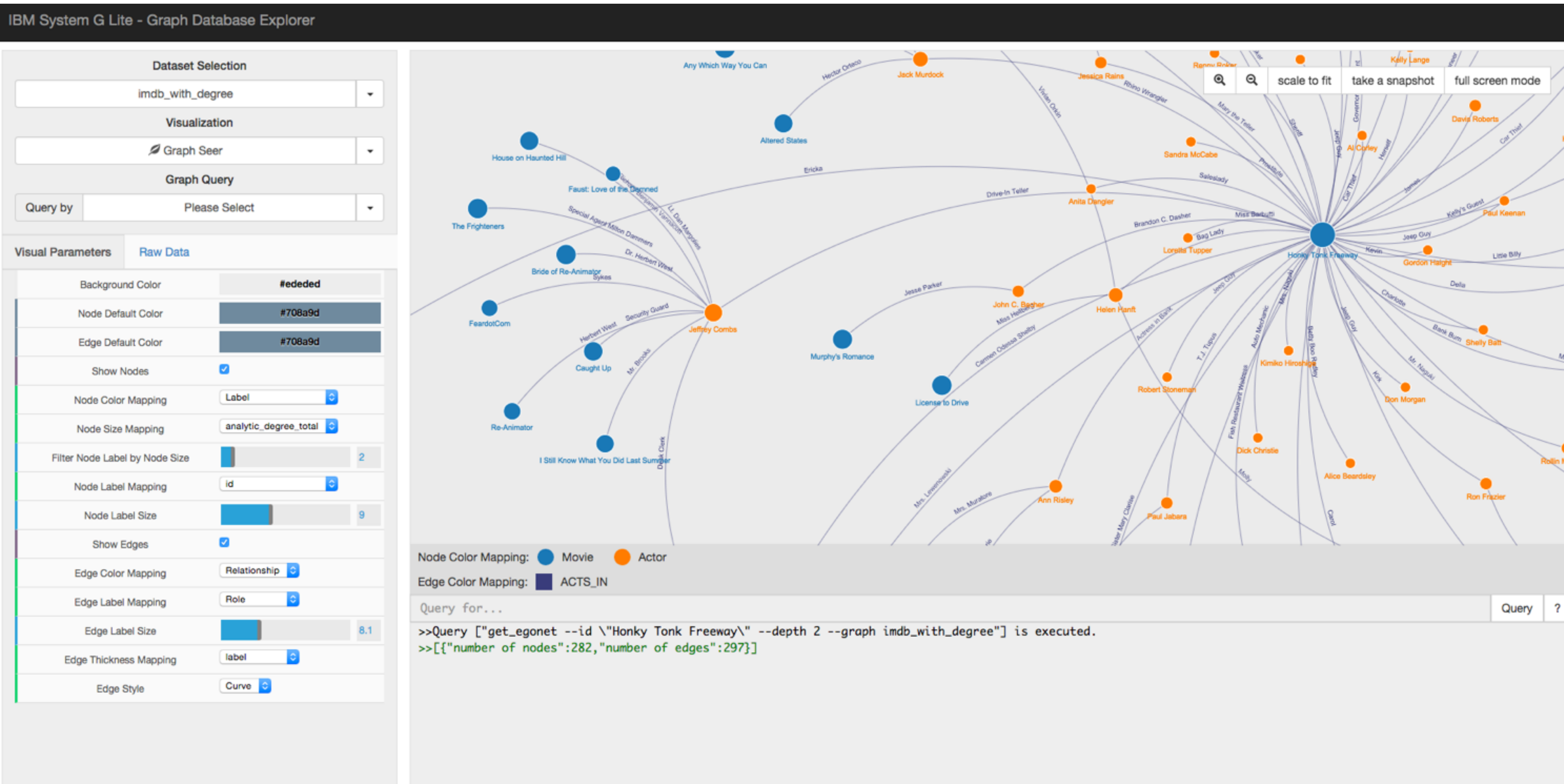
Products (DB2 via SQL, Oracle via SQL, DB2RDF via SPARQL) · *Startup* (Neo4j) · *Open Sources* (Titan) · *System G* (System G GBase, System G Native Store)

*All performance numbers are preliminary

IBM System G Visualizer – Graph Data Explorer

Visual Query Panel

Visualization Panel



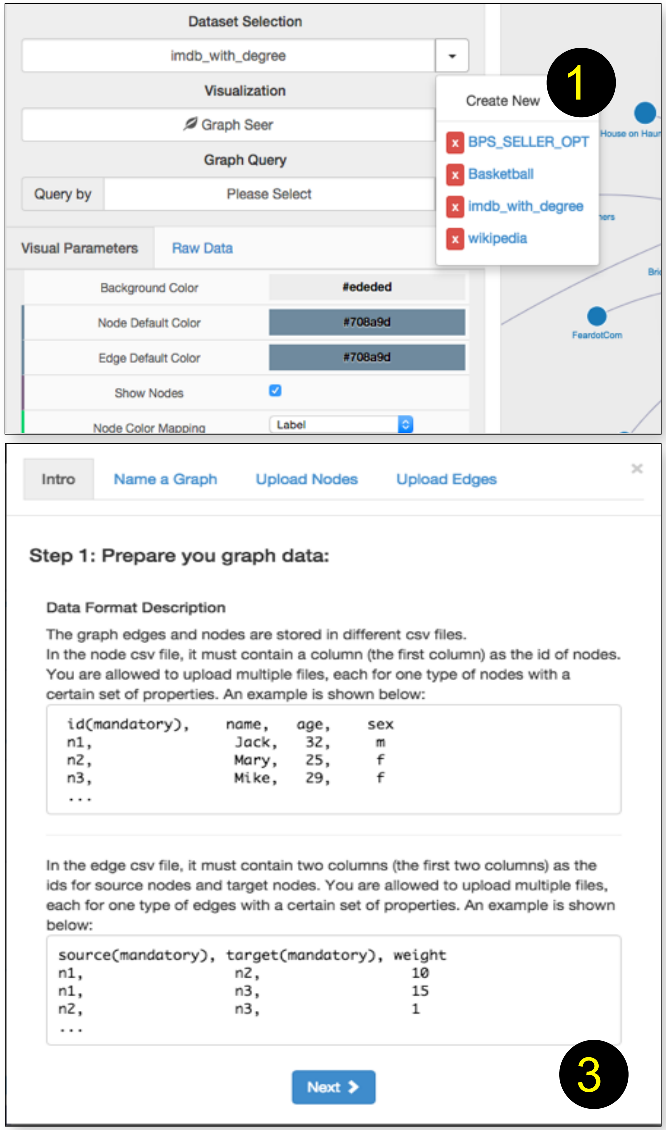
Visual Mapping Panel

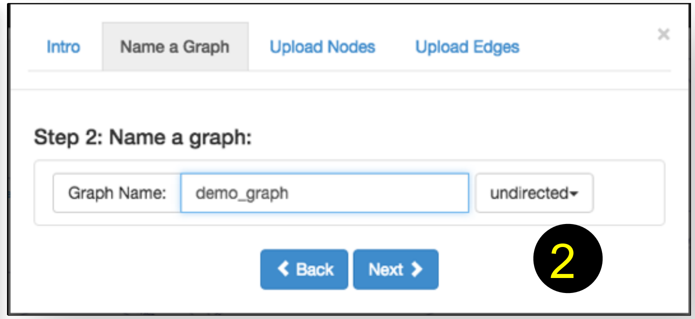
Console Panel

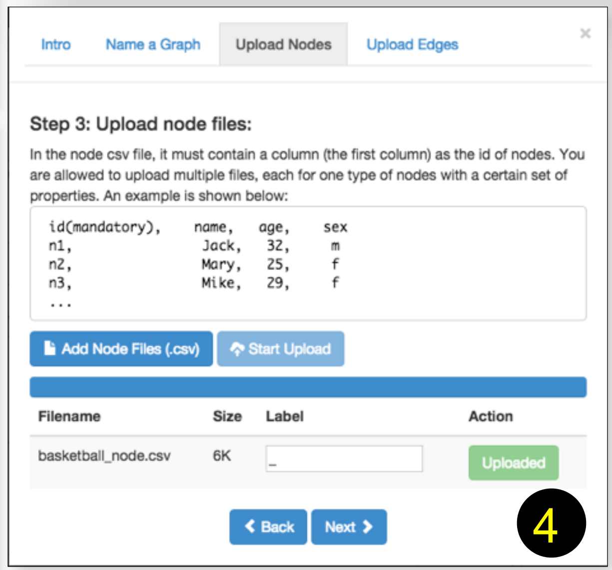
Panel Introduction

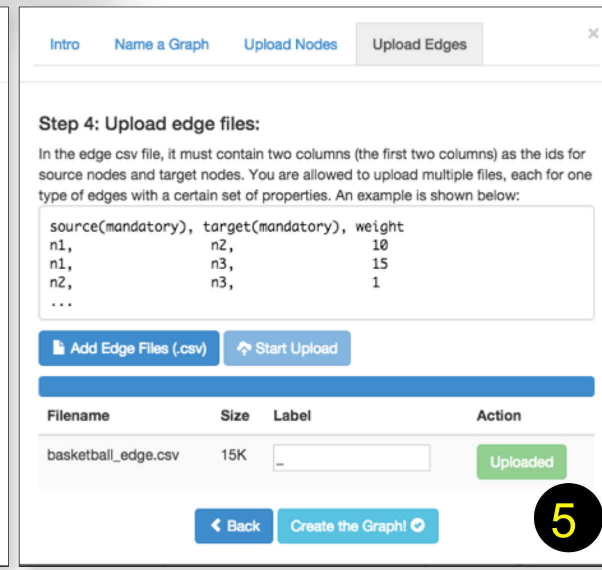
- Visual Query Panel
  - Providing users a friendly UI to create, delete, and query graphs from the System G native store.
- Console Panel
  - Display all the interaction information with System G native store.
  - Execute user defined query.
- Visualization Panel
  - Rendering graph structure on screen for users to visually explore graphs.
- Visual Mapping Panel
  - Customizing rendering effects to show desired graph information.

Visual Query Panel – Creating a graph



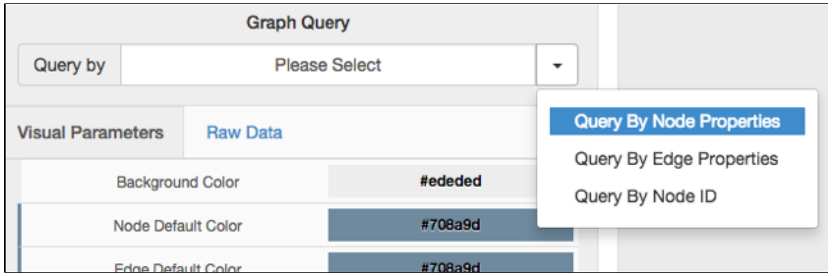


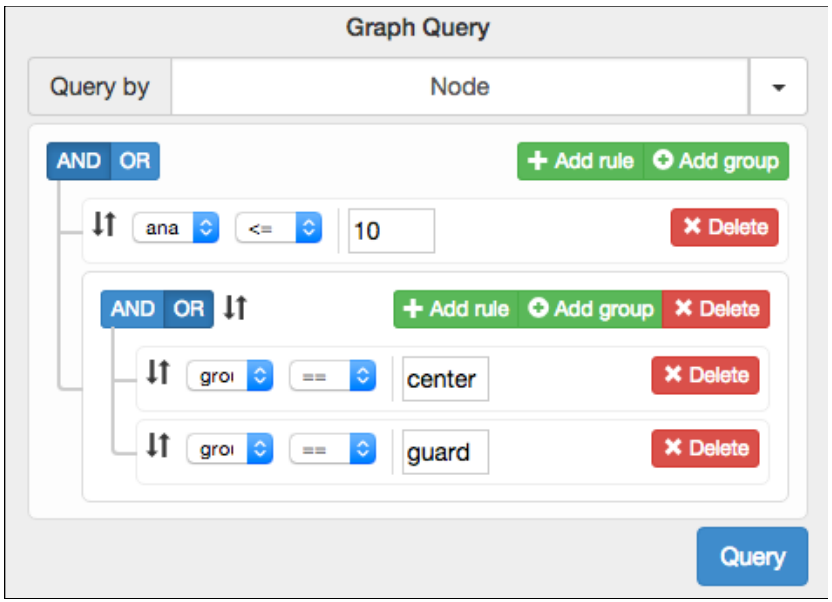


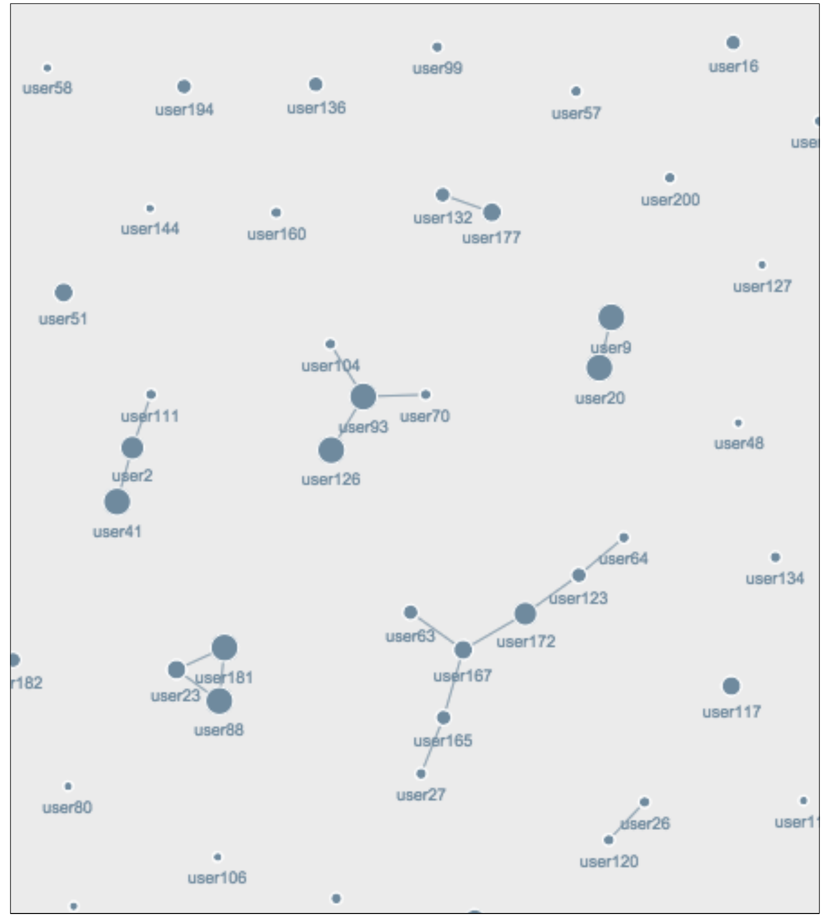


1: Click "Create Graph"; 2: Prepare the graph data
3: Set the graph name; 4: Upload node files;
5: Upload edge files and finalize creating the graph.

Visual Query Panel – Visual Query Builder





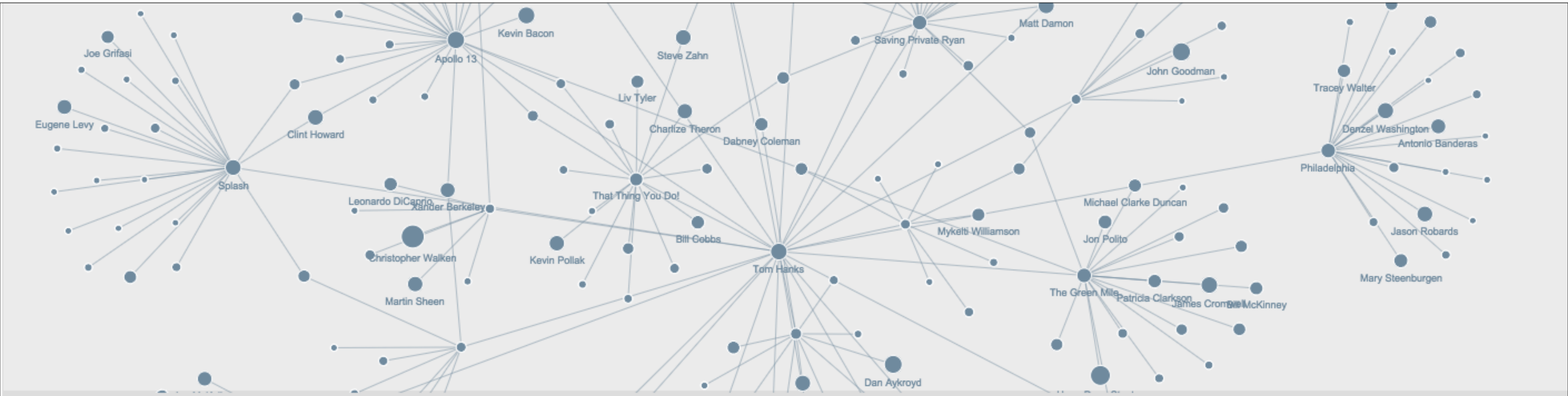


“analytics_degree <= 10 and (group == “center” or group == “guard”)

Console Panel – User typed query

```
find_vertex_max_degree --graph Basketball --edgetype all
>>Query ["print_all --graph Basketball"] is executed.
>>[{"number of nodes":199,"number of edges":826}]
>>Query ["find_vertex_max_degree --edgetype all --graph Basketball"] is executed.
>>[{"vertex id":"user72"},{"all-degree":46}]
```

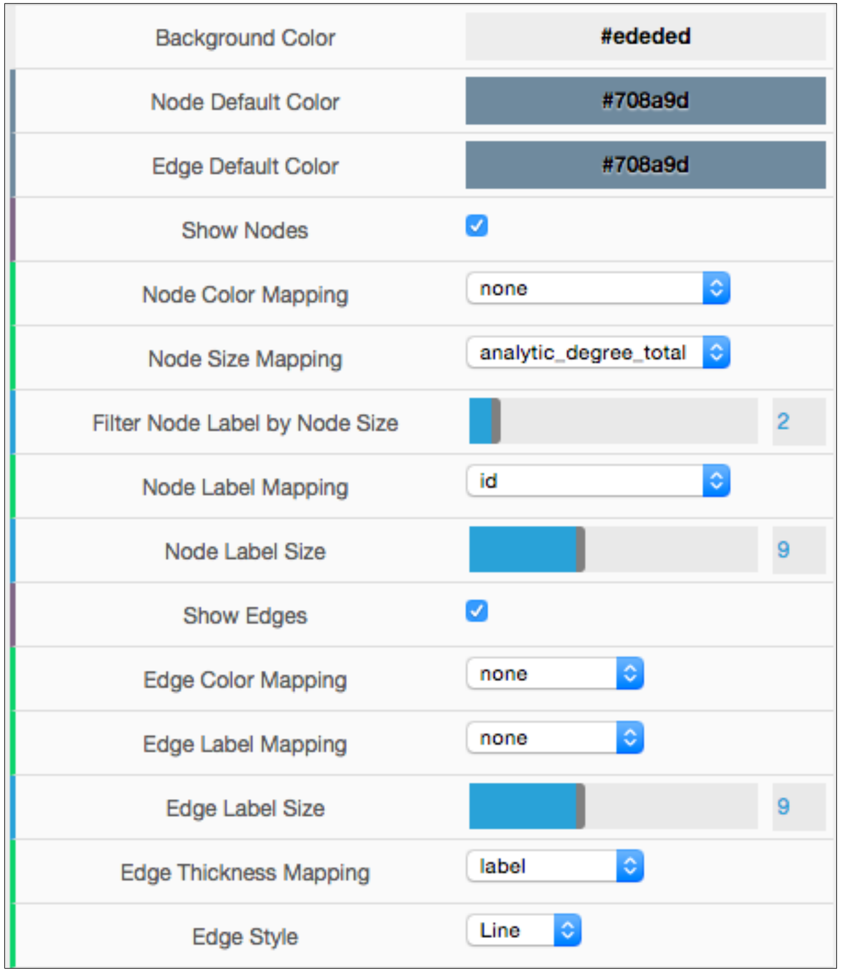
Query with no graph returned




```
get_egonet --graph imdb_with_degree --id "Tom Hanks" --depth 2
>>Query ["print_all --graph Basketball"] is executed.
>>[{"number of nodes":199,"number of edges":826}]
>>Query ["find_vertex_max_degree --edgetype all --graph Basketball"] is executed.
>>[{"vertex id":"user72"},{"all-degree":46}]
>>Query ["get_egonet --id \"Tom Hanks\" --depth 1 --graph imdb_with_degree"] is executed.
>>[{"number of nodes":26,"number of edges":25}]
>>Query ["get_egonet --id \"Tom Hanks\" --depth 2 --graph imdb_with_degree"] is executed.
>>[{"number of nodes":383,"number of edges":401}]
```

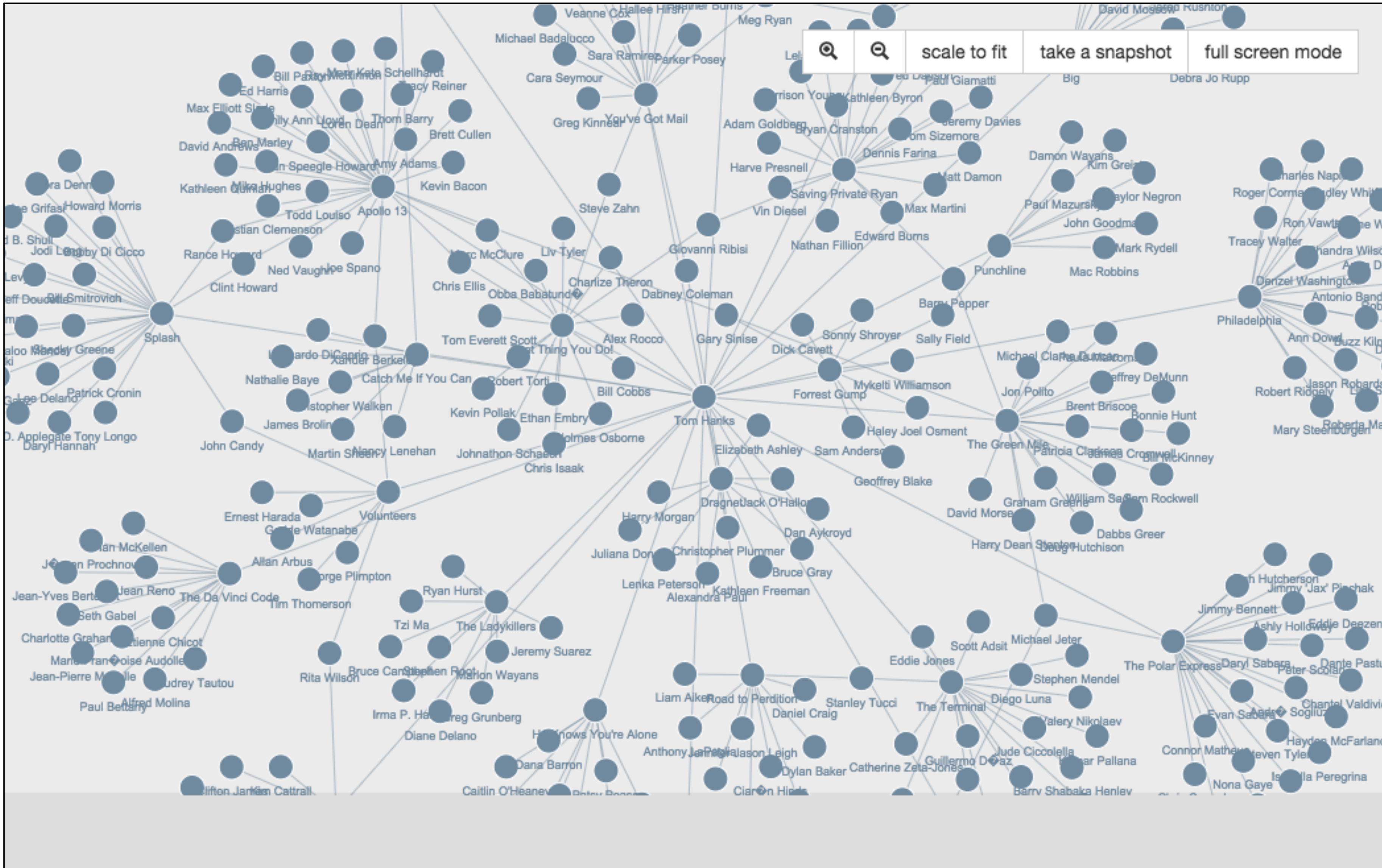
Query with graph returned

Visual Mapping Panel

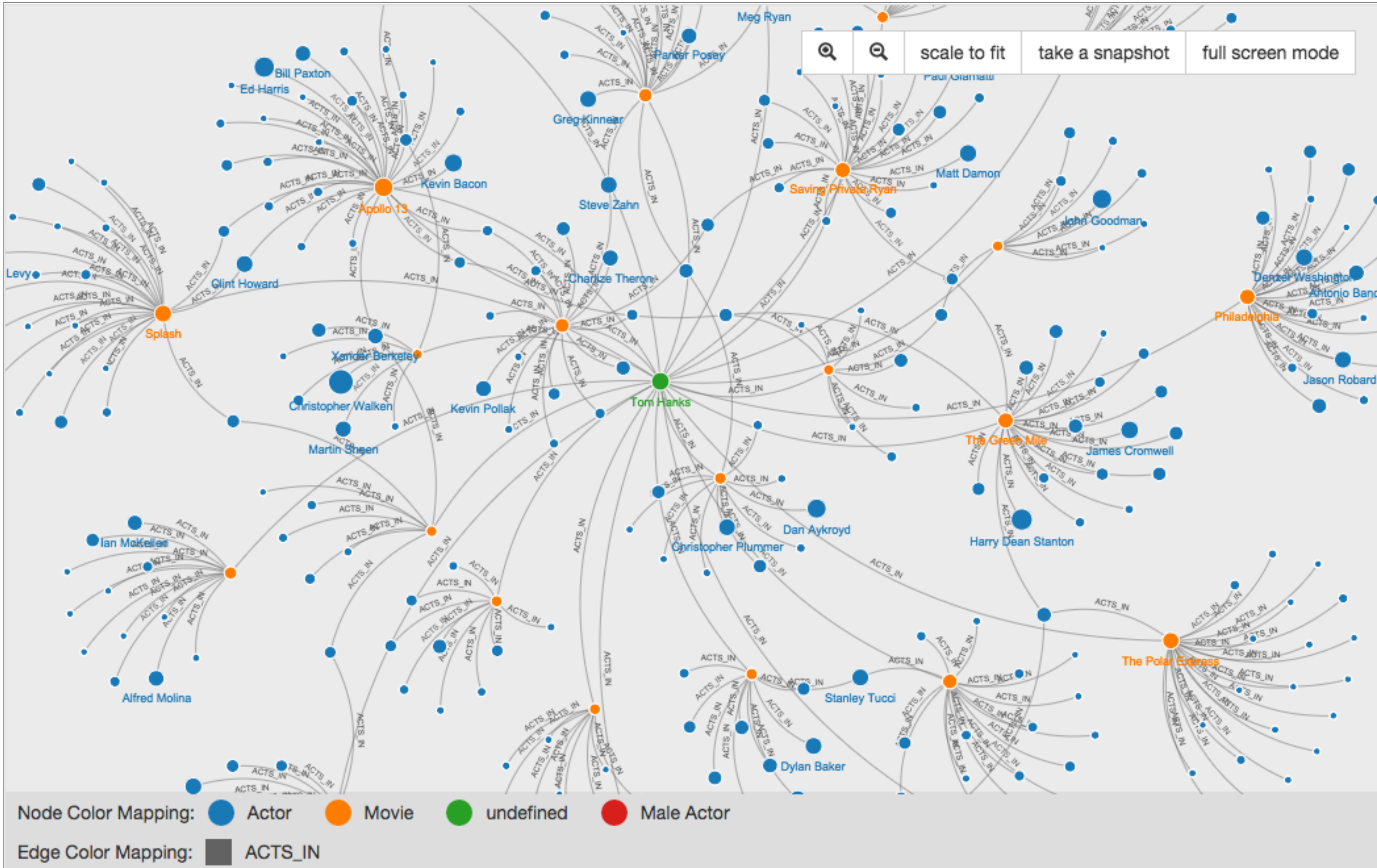


Name	Functionality
Background Color	Change the background color of the canvas.
Node Default Color	Set a unified color for all nodes.
Edge Default Color	Set a unified color for all edges.
Show Nodes	Set the visibility of all nodes.
Node Color Mapping	Assign color to nodes according to selected property of nodes.
Node Size Mapping	Assign the radius of nodes according to selected property of nodes.
Filter Node Label by Node Size	Selectively show the node label according to the threshold. Labels will be shown for the nodes of which the size is larger than the threshold.
Node Label Mapping	Set the label value according to selected property of nodes.
Node Label Size	Adjust the font size of node labels
Show Edges	Set the visibility of all edges
Edge Color Mapping	Assign color to edges according to selected property of edges.
Edge Label Mapping	Set the label value according to selected property of edges.
Edge Label Size	Adjust the font size of edge labels
Edge Thickness Mapping	Assign thickess to edges according to selected property of edges.
Edge Style	Select the rendering style of edges. For directed graphs, users also can choose if showing the arrows or not.

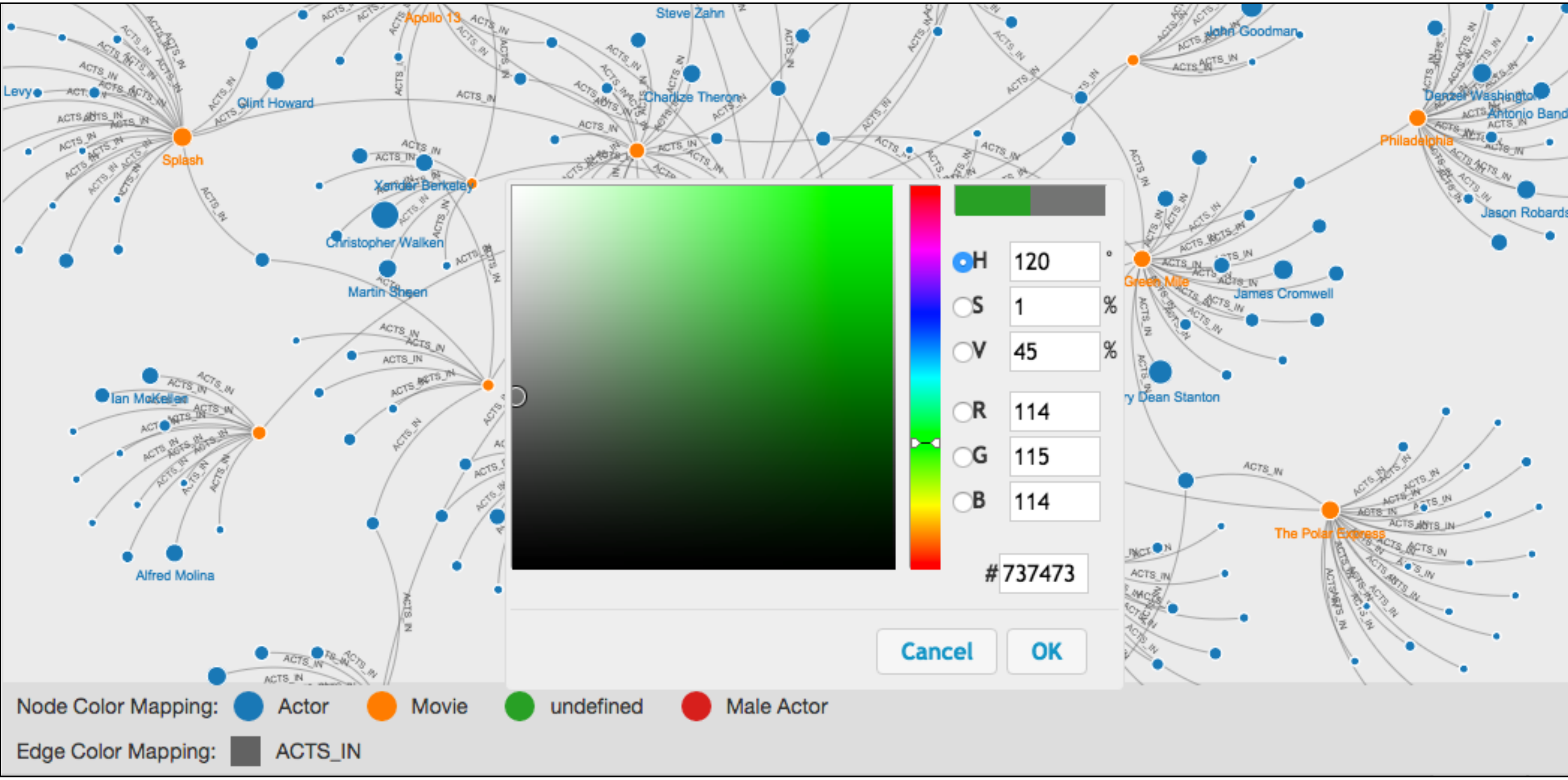
Visualization Panel – Before Customization



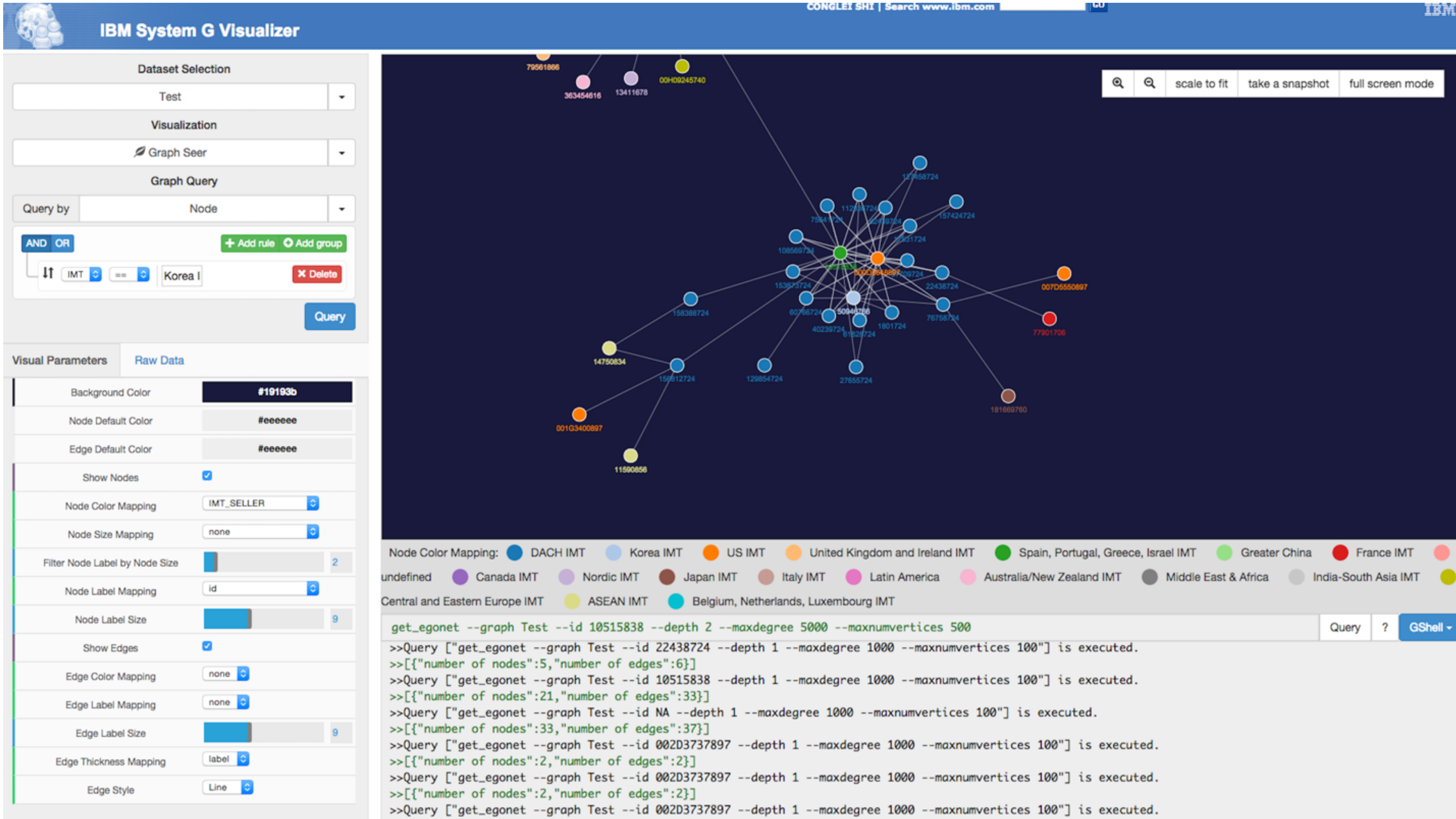
Visualization Panel – After Customization



Visualization Panel – Further Customization



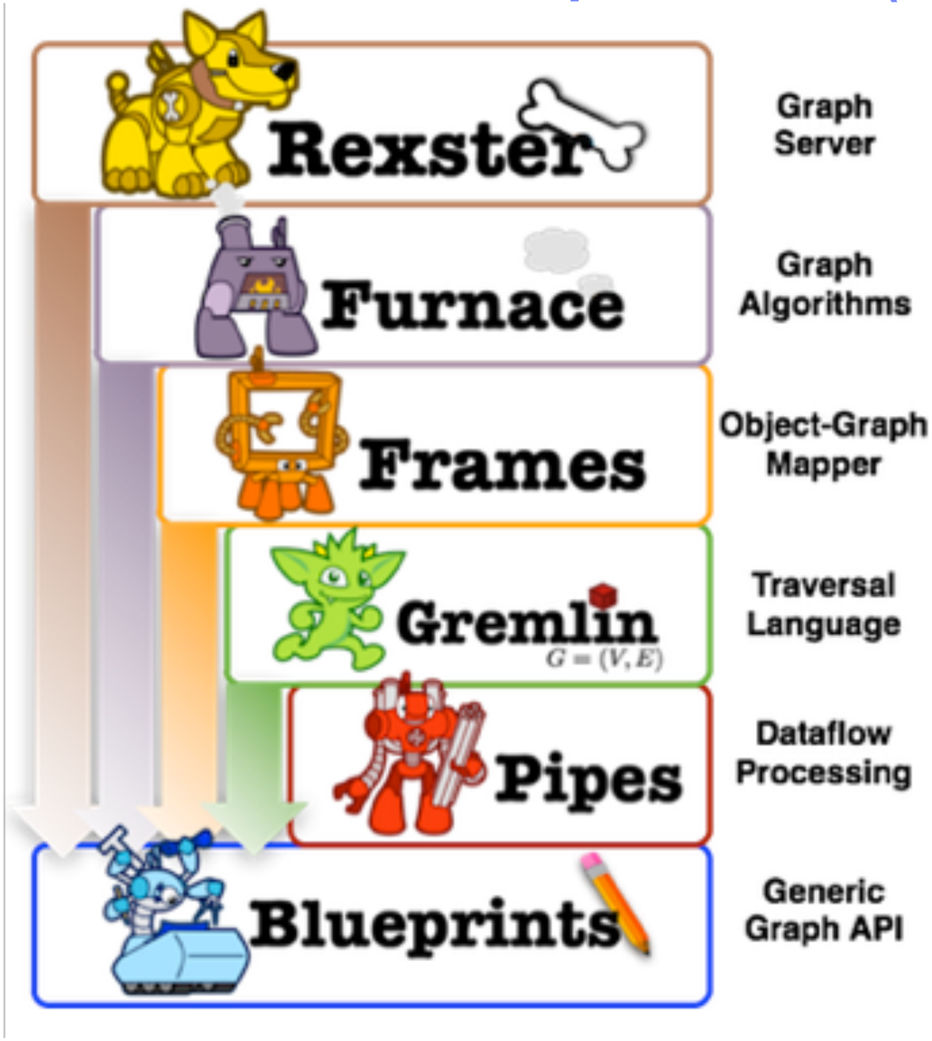
Users can further specify colors by clicking the color blocks shown in the legend area http://systemg.ibm.com/tool/visualizer/



Characteristics of IBM System G Graph Analytics

- Cover a wide range of graph analytics to support many application use cases in different domains, e.g.:
  - Enterprise social network analysis, expertise search, knowledge recommendation
  - Financial/security anomaly/fraud detection
  - Social media monitoring and analysis
  - Cellular network analytics in Telco operation
  - Patient and disease analytics for healthcare
  - Live neural brain network analysis
- Provide efficient in-memory computation as well as on-disk persistence
- Optimal performance enabled by IBM System G graph database technologies that focus on efficient use of available computing resources with architecture-aware design to leverage system/architecture advantages
- Single-threaded, concurrent (shared memory), and distributed versions
- Multiple deployment options to suit different customer preferences and needs
  - C++ executables in Linux environments (Redhat CentOS, Ubuntu, Mac OS X, Power)
  - TinkerPop (Blueprints) API
  - gShell (a shell-like environment with interactive, batch, and server/client modes to operate multiple graph stores simultaneously)
  - Gremlin console
  - REST API Web service
  - Python wrapper

Compatible with TinkerPop Interface (Apache Incubator)



http://sql2gremlin.com

http://tinkerpop.incubator.apache.org

Write Python Code based on System G

```
#!/usr/bin/python
from py_gShell import _py_gshell as gShell
import json

g = gShell()
g.delete_graph("testu")
g.create_graph("testu", "undirected")
g.load_csv_vertices(csvfile="data/test.vertices.dat", keypos=0, labelpos=1)
g.load_csv_edges(csvfile="data/test.edges.dat", srcpos=0, targpos=1, labelpos=3)
g.add_vertex(vertex_id="7", label="C", prop={"tag":"T2","value":0.1})
g.add_vertex(vertex_id="8", prop={"value":0.4})
g.add_vertex(vertex_id="9", label="C", prop={"value":0.5})
g.update_vertex(vertex_id="9",prop={"value":0.55,"other":"1"})
g.add_edge(src="7", targ="8", edgelabel="c")
g.add_edge(src="7", targ="1", edgelabel="c",prop={"weight":8.0})
g.update_edge(src="1",targ="7", prop={"weight":8.6, "other":"2"})
g.add_edge(src="8",targ="9")
g.update_edge(src="1",targ="2", prop={"weight":6.5})
g.analytic_start_engine(edgeweightpropname="weight")
print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2")), indent = 4)

print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2",label="b")), indent = 4)

g.analytic_stop_engine()
```

Output of the above Python script

g.analytic_find_path(src="1",sink="2")


```
{
    "paths": [
        {
            "src": "1",
            "path": "1-->2",
            "sink": "2",
            "distance": 1.0
        }
    ],
    "time": [
        {
            "TIME": "3.31402e-05"
        }
    ]
}
```

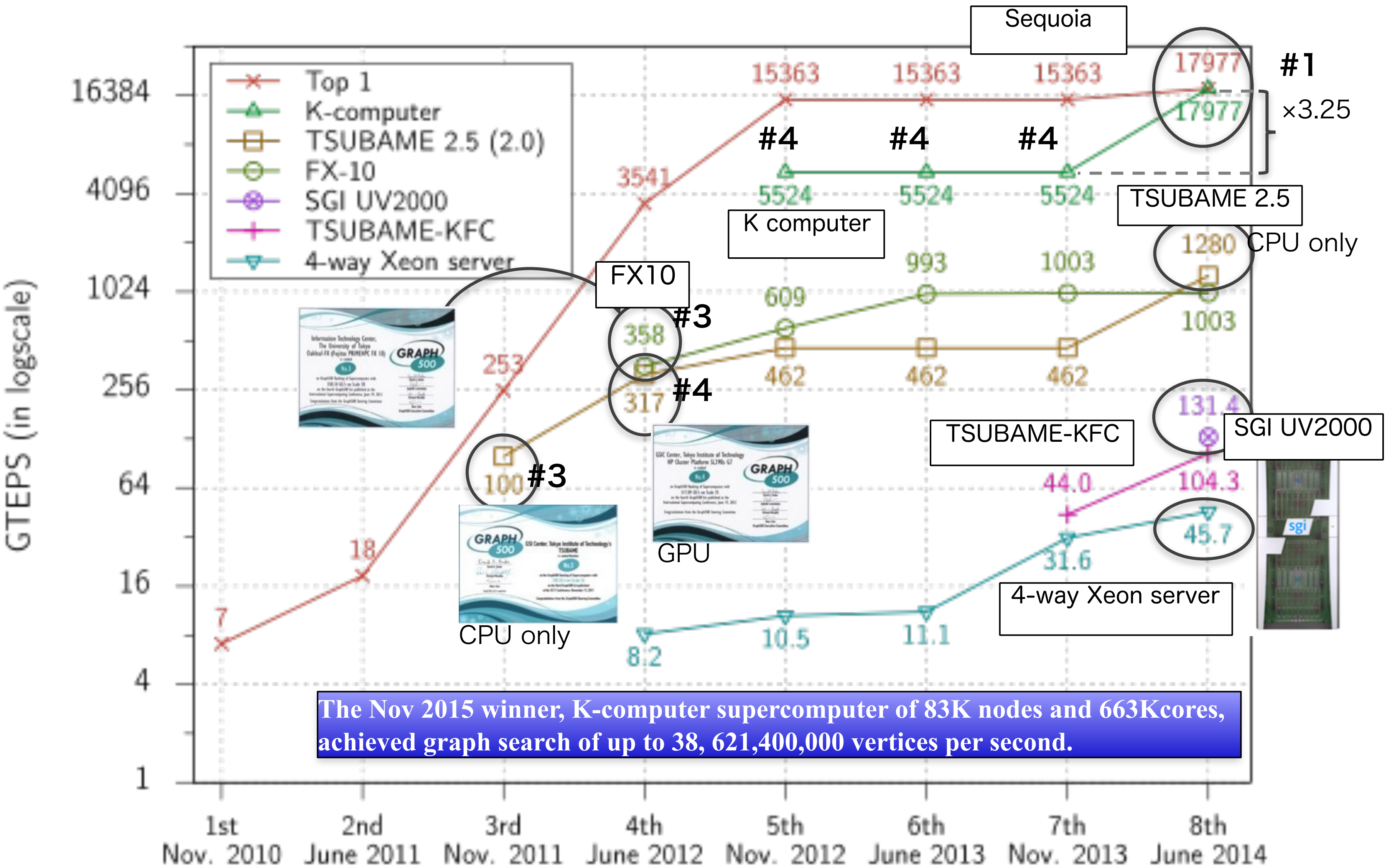

g.analytic_find_path(src="1",sink="2",label="b")


```
{
    "paths": [
        {
            "src": "1",
            "path": "1-->3-->5-->2",
            "sink": "2",
            "distance": 3.0
        }
    ],
    "time": [
        {
            "TIME": "2.09808e-05"
        }
    ]
}
```

Highly Scalable Graph Database

http://www.graph500.org

Nov 2015: IBM Research's Software powered all Top 3 winners of Graph 500 ben… 9 out of the Top 10 winners (supercomputers in US, Japan, France, UK, and Germ… in China).



Comparison of graph size