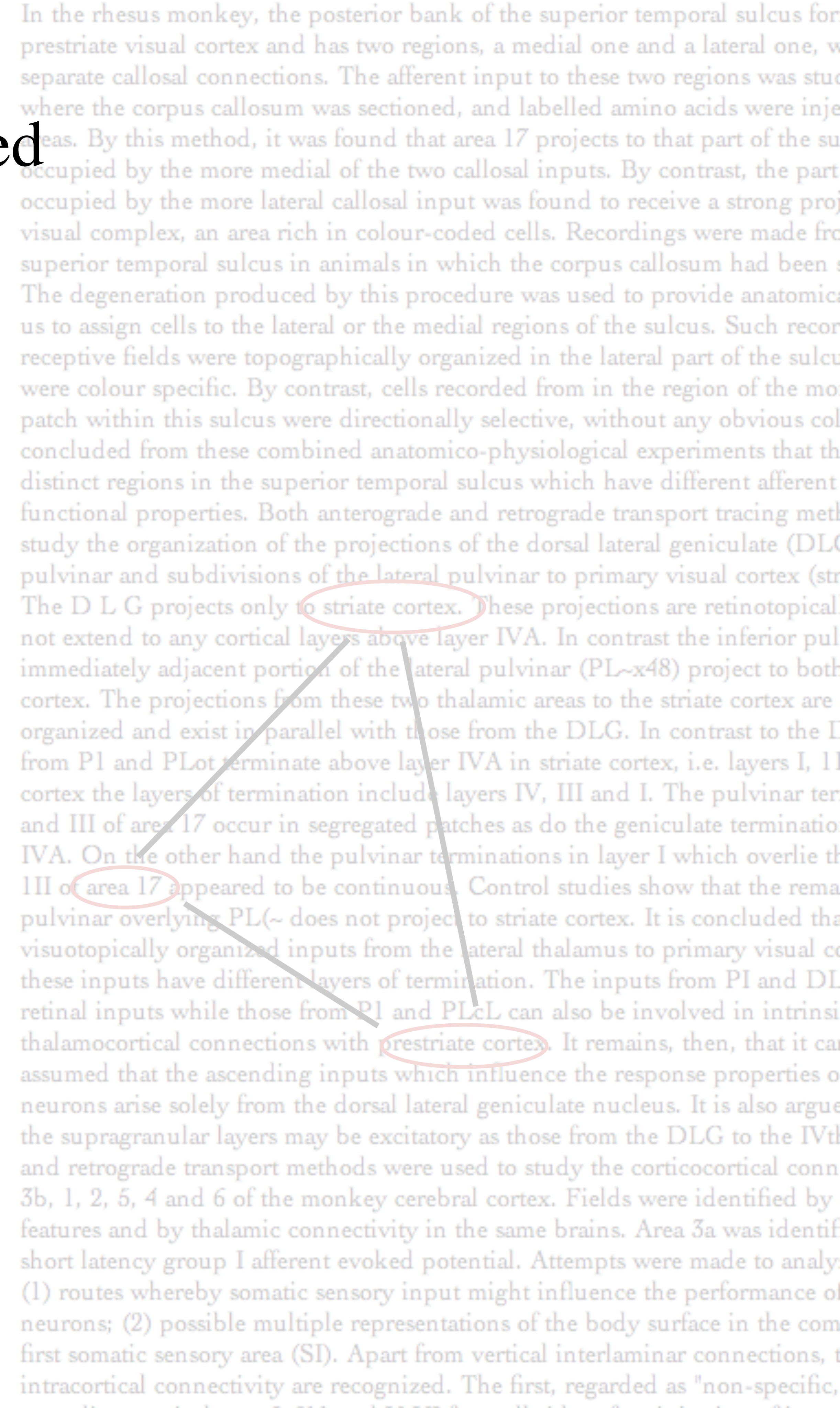
Design Considerations for a Distributed Graph Analytics Pipeline

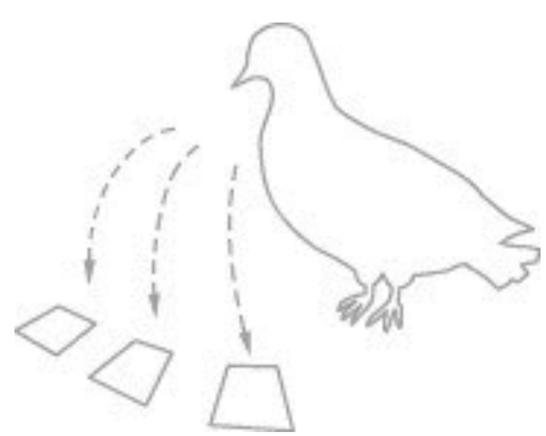


Kyle H. Ambert, PhD

Intel Graph Analytics Operation

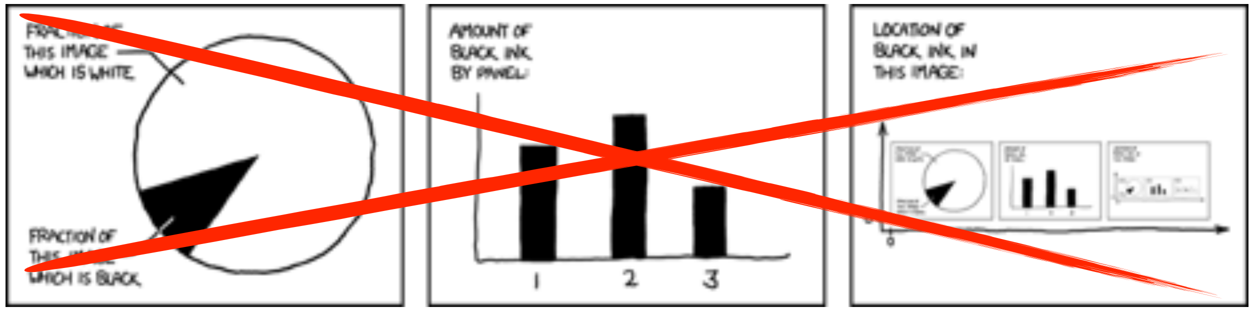
May 23rd, 2014

About me.



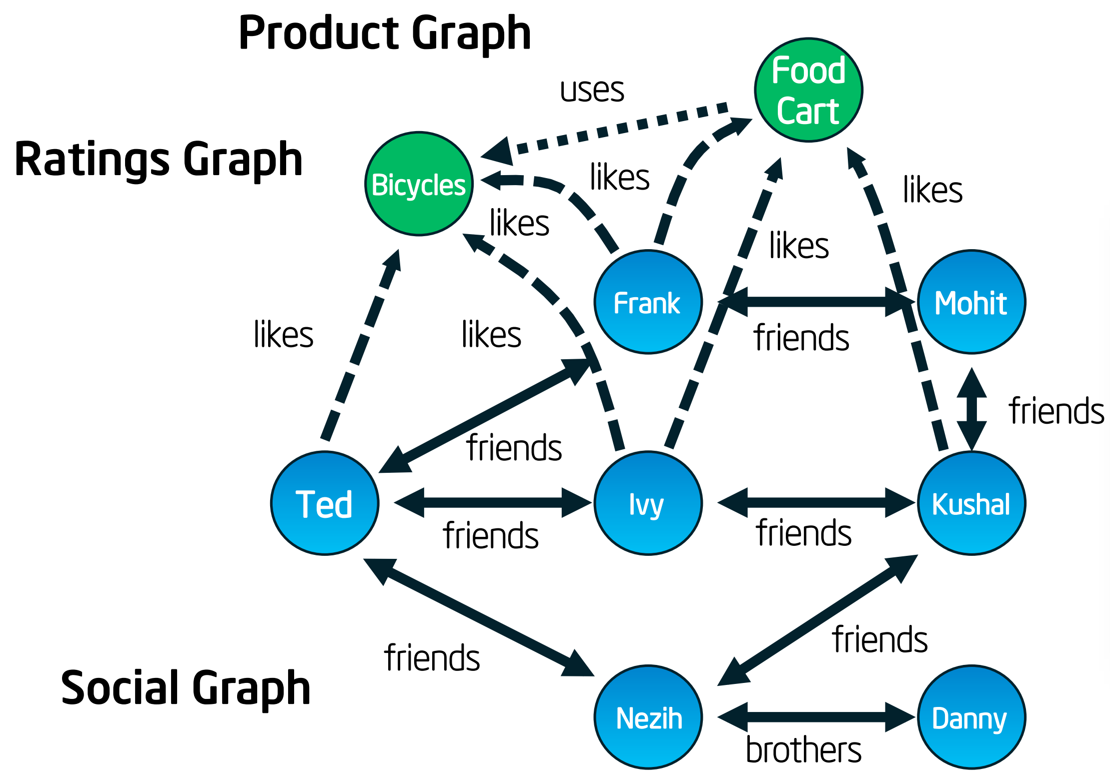
Graphs in the Biological Sciences

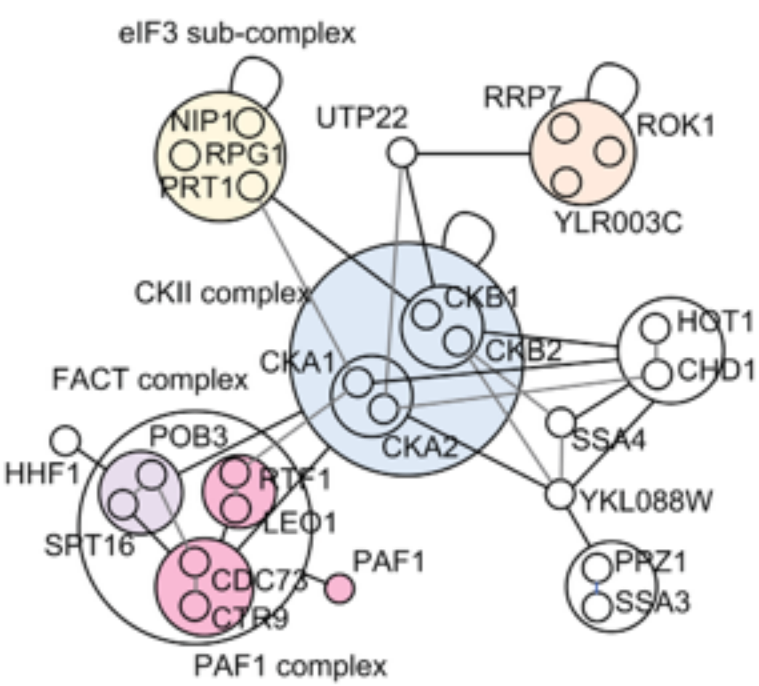
(not these)



- A convenient way to represent certain types of data!

- Recommender systems are frequently powered by a graph-centric analysis pipeline.



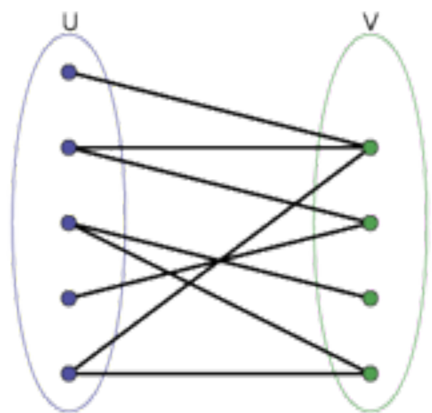


- Analogous networks of information can be found in biology!

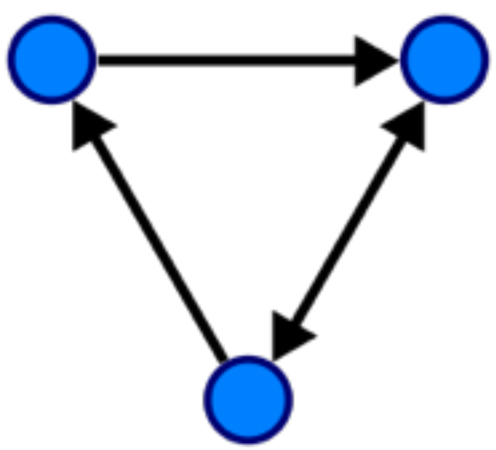
http://imgs.xkcd.com/comics/self_description.png

Graphs

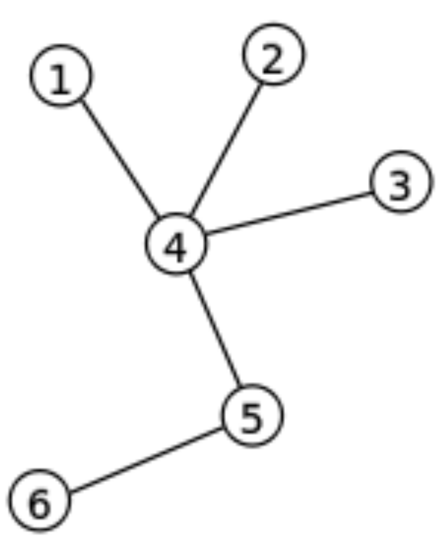
Bipartite



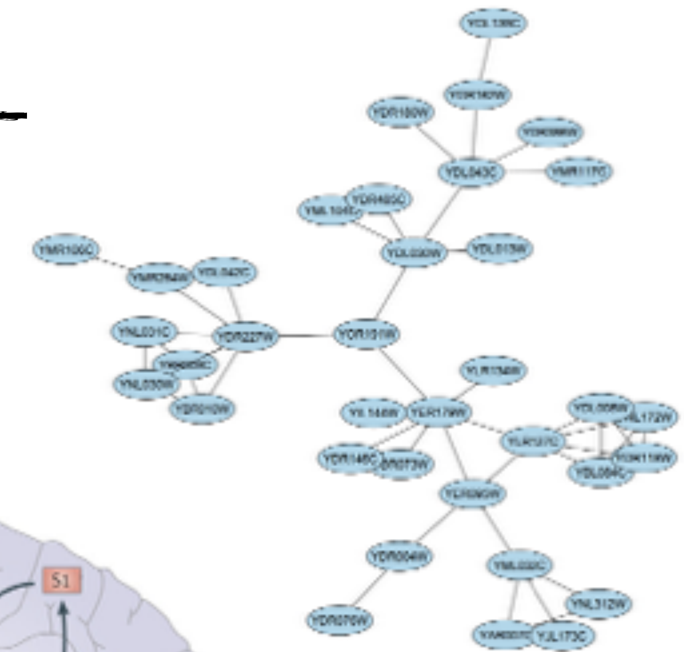
Directed

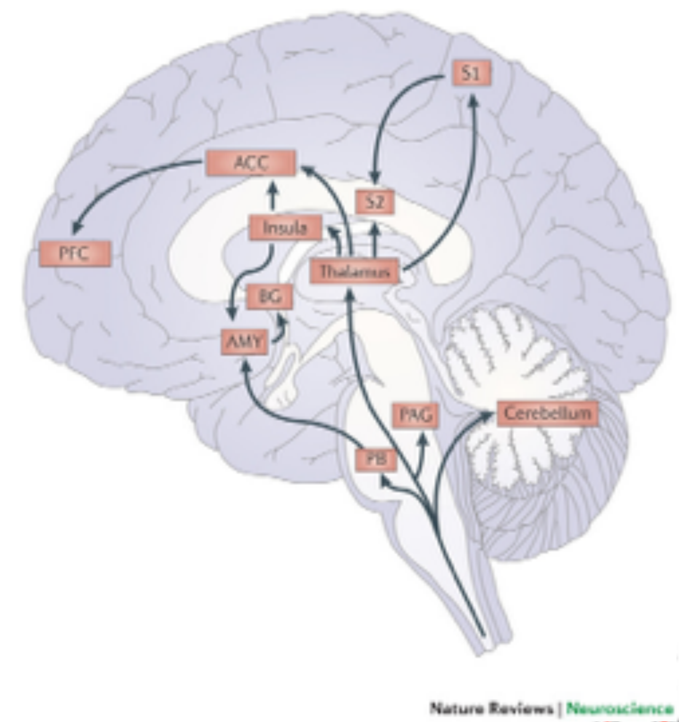


Tree

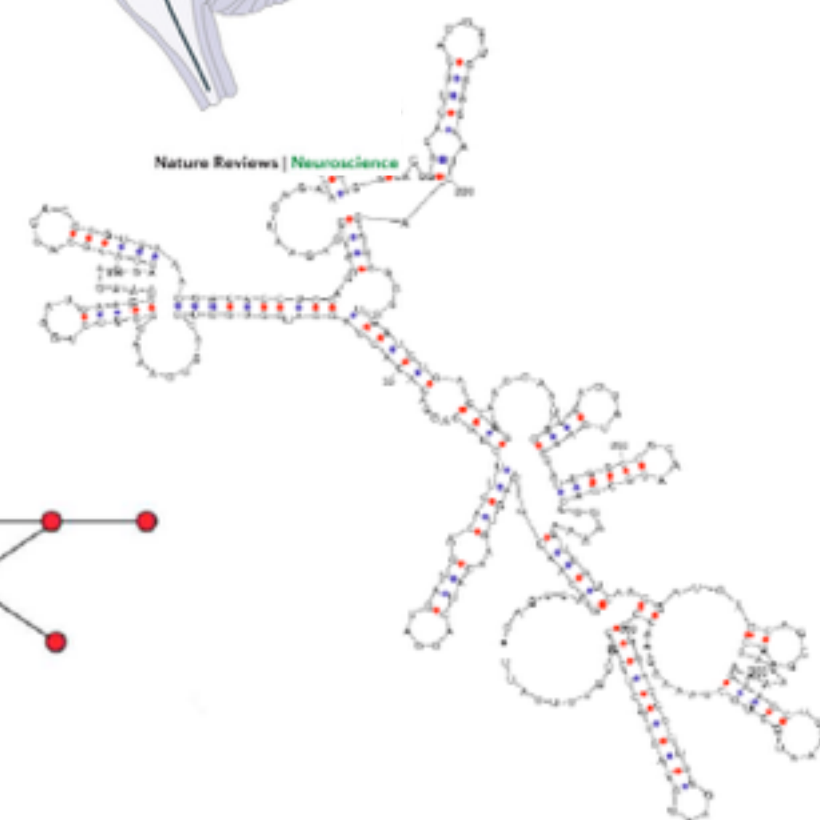


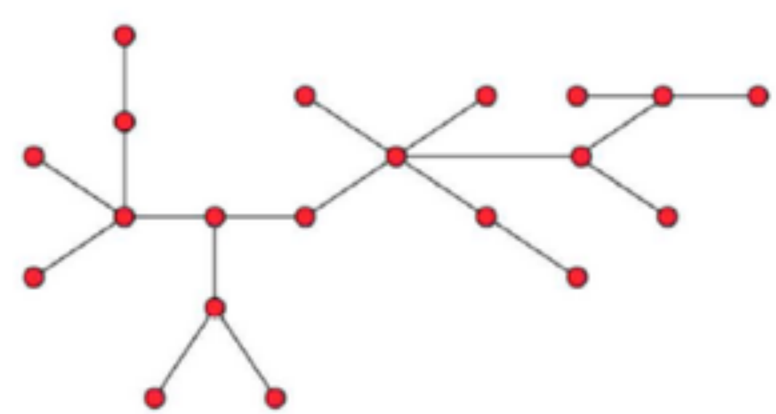
Protein co-expression network in yeast





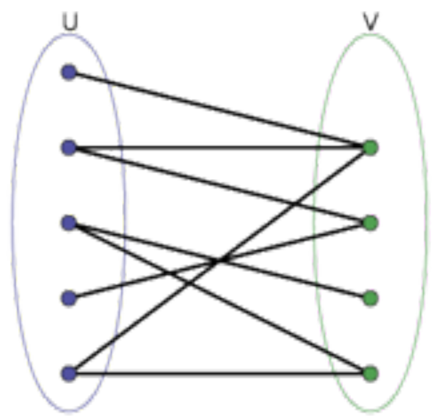
The tree representation of an RNA molecule's secondary structure



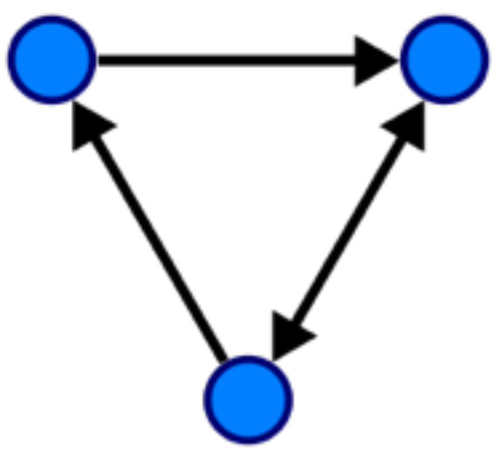


Graphs as Data Models

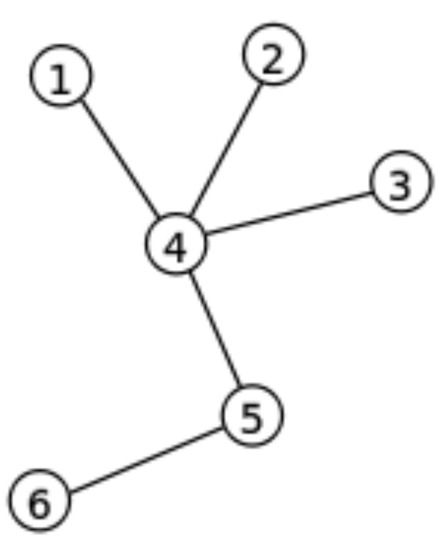
Bipartite



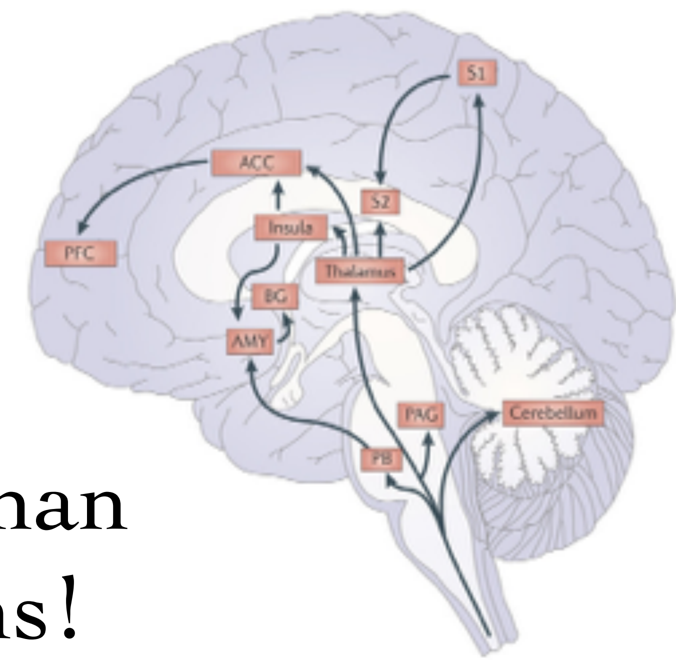
Directed



Tree



There are advantages and disadvantages to each!



Nature Reviews | Neuroscience

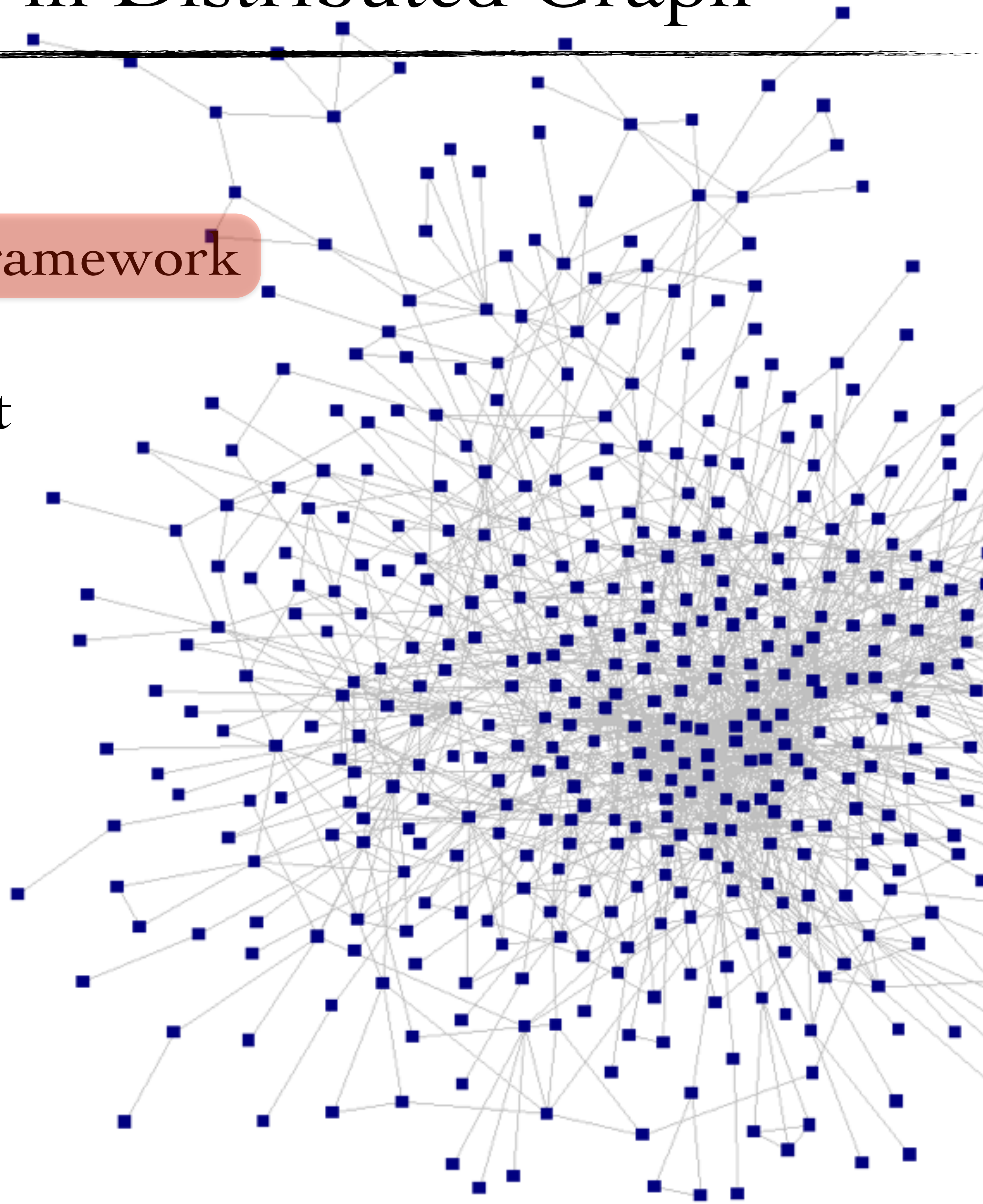
ML is more than just algorithms!

Would you represent a connectome as a big table?

maybe...

Some Open Questions in Distributed Graph

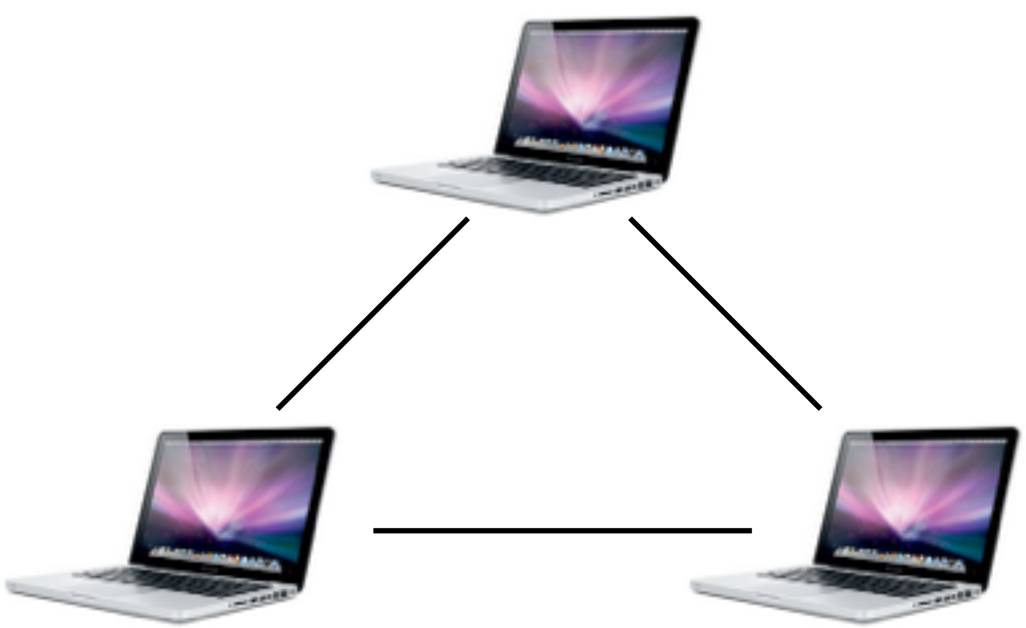
[1] Distributed computing framework
[2] Graph abstraction
[3] Building a useable toolkit



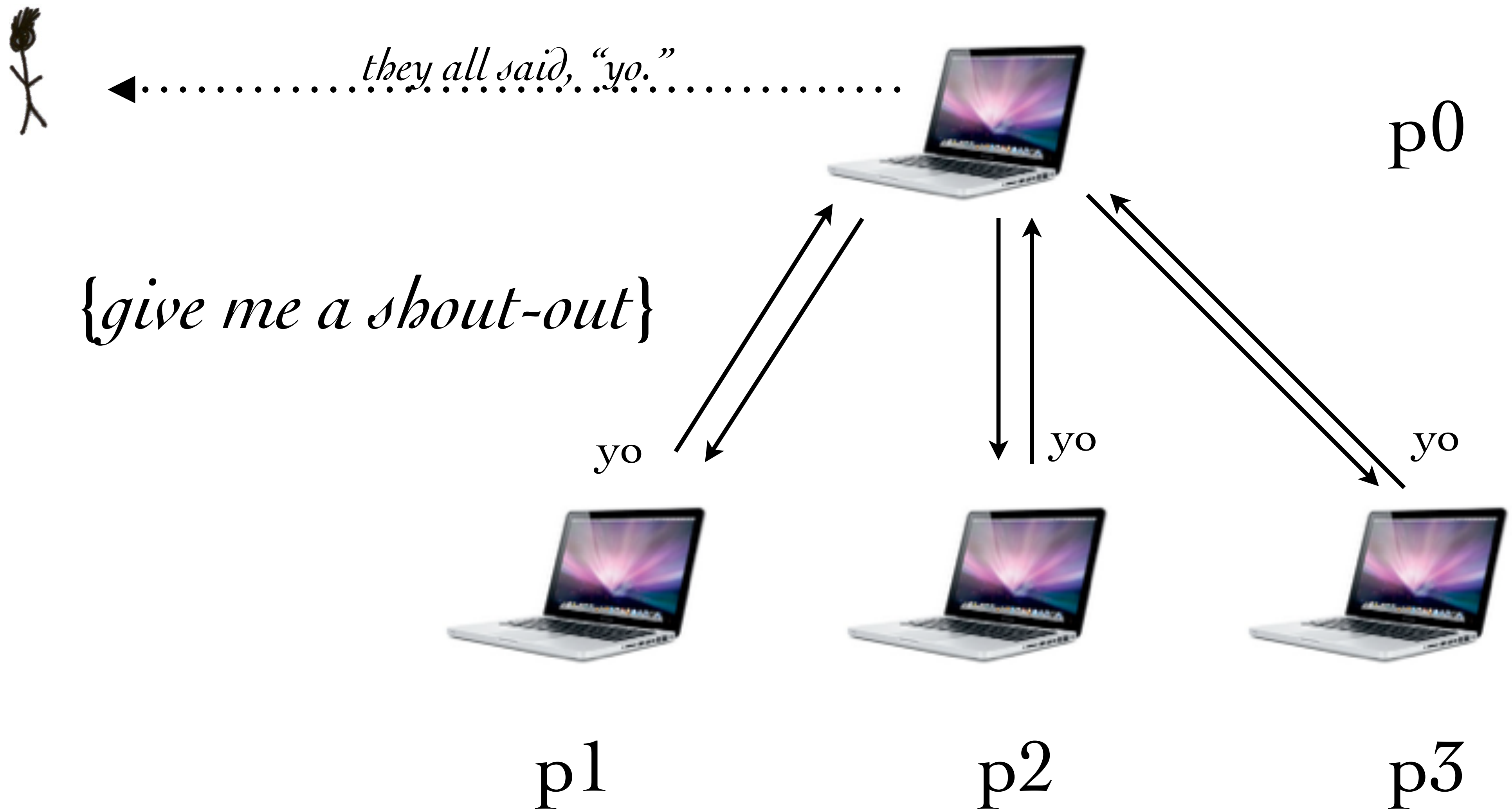
orgnet.com/alters2.gif

Distributed Frameworks & Tools

- MPI
- MapReduce/Hadoop
- Next generation



Distributed Frameworks: *MPI!*



Messages can be…
- —collective or peer-to-peer
- —synchronous or asynchronous
- —blocking or non-blocking

Distributed Frameworks: *MPI!*

woo!

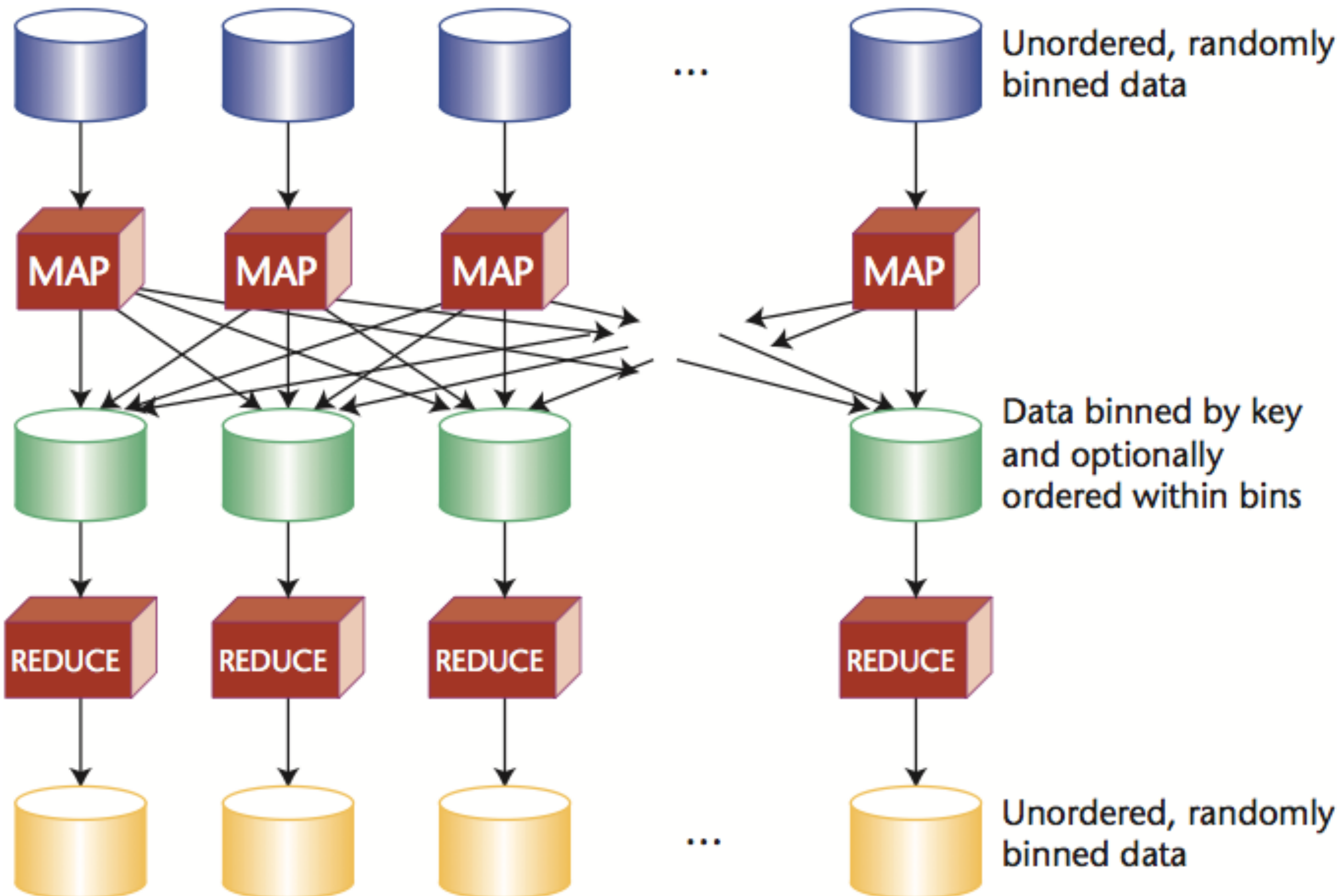
- good if you're writing a parallel library
- scalability, compatibility, and portability
- mature and well-understood
- can be quite fast

lame!

- no fault tolerance
- hard to learn
- most commonly implemented in FORTRAN, C, or C++

```c
/*
 "Hello World" MPI Test Program
*/
#include <mpi.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 128
#define TAG 0

int main(int argc, char *argv[])
{
  char idstr[32];
  char buff[BUFSIZE];
  int numprocs;
  int myid;
  int i;
  MPI_Status stat;
  /* MPI programs start with MPI_Init; all 'N' processes exist thereafter */
  MPI_Init(&argc,&argv);
  /* find out how big the SPMD world is */
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  /* and this processes' rank is */
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  /* At this point, all programs are running equivalently, the rank
     distinguishes the roles of the programs in the SPMD model, with
     rank 0 often used specially... */
  if(myid == 0)
  {
    printf("%d: We have %d processors\n", myid, numprocs);
    for(i=1;i<numprocs;i++)
    {
      sprintf(buff, "Hello %d! ", i);
      MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
    }
    for(i=1;i<numprocs;i++)
    {
      MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &stat);
      printf("%d: %s\n", myid, buff);
    }
  }
  else
  {
    /* receive from rank 0: */
    MPI_Recv(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD, &stat);
    sprintf(idstr, "Processor %d ", myid);
    strncat(buff, idstr, BUFSIZE-1);
    strncat(buff, "reporting for duty\n", BUFSIZE-1);
    /* send to rank 0: */
    MPI_Send(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD);
  }

  /* MPI programs end with MPI Finalize; this is a weak synchronization point */
  MPI_Finalize();
  return 0;
}
```

Distributed Frameworks: *MapReduce!*



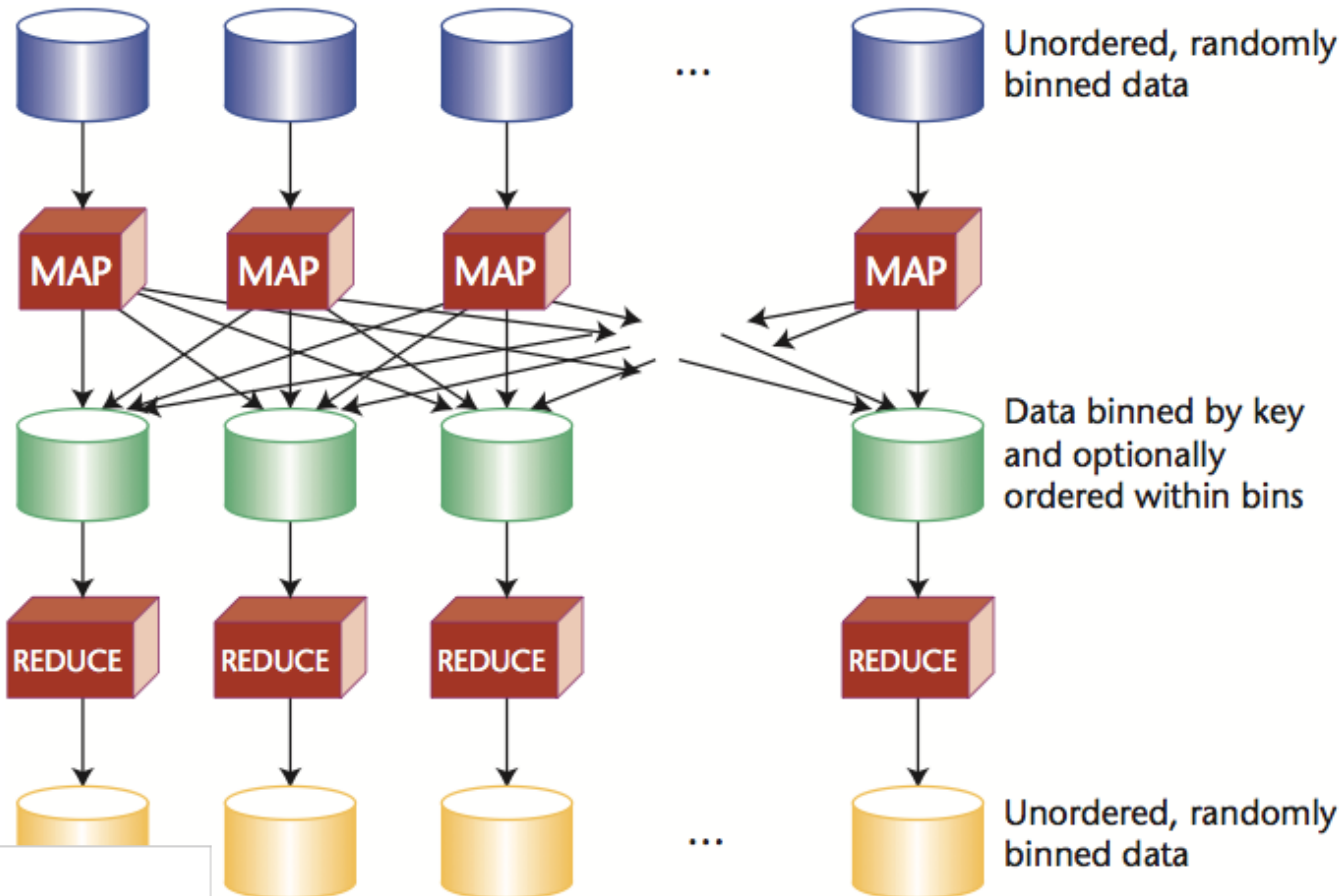
woo!

- Fault tolerant!
- Great for MRable problems

lame!

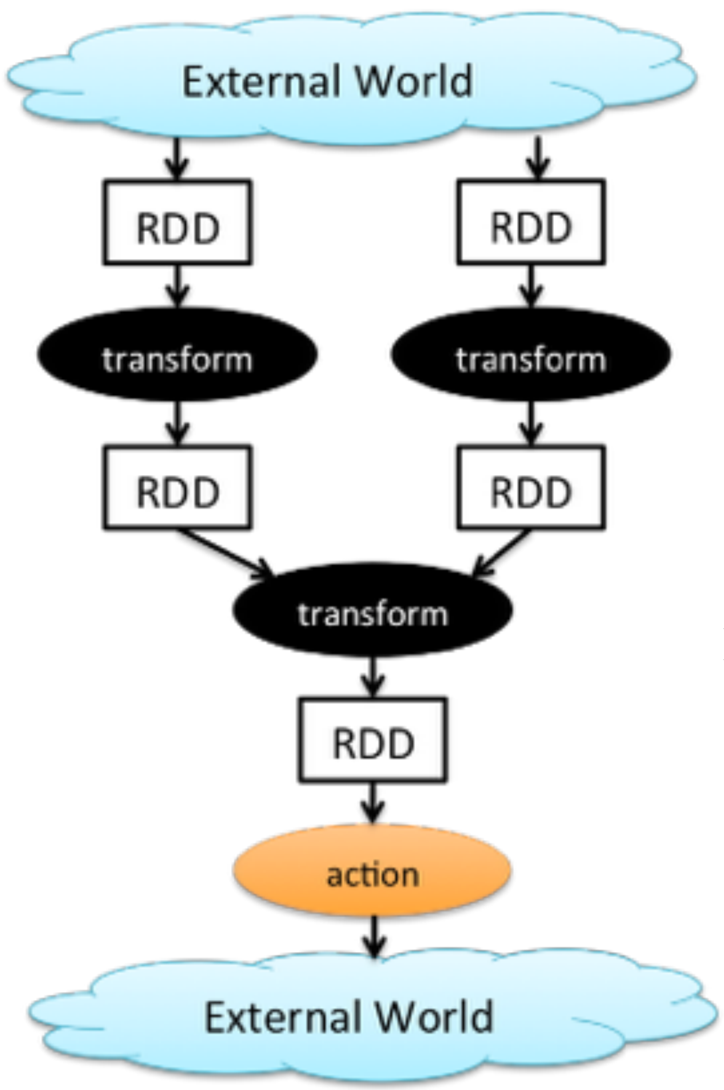
- Nodes can't talk to each other
- Batch processing
- No real-time processing
- Overhead for shuffling large amounts of data over network
- Not super great for graphs

Distributed Frameworks: *Everyone and their elephant has one*




Spark
nasa.gov

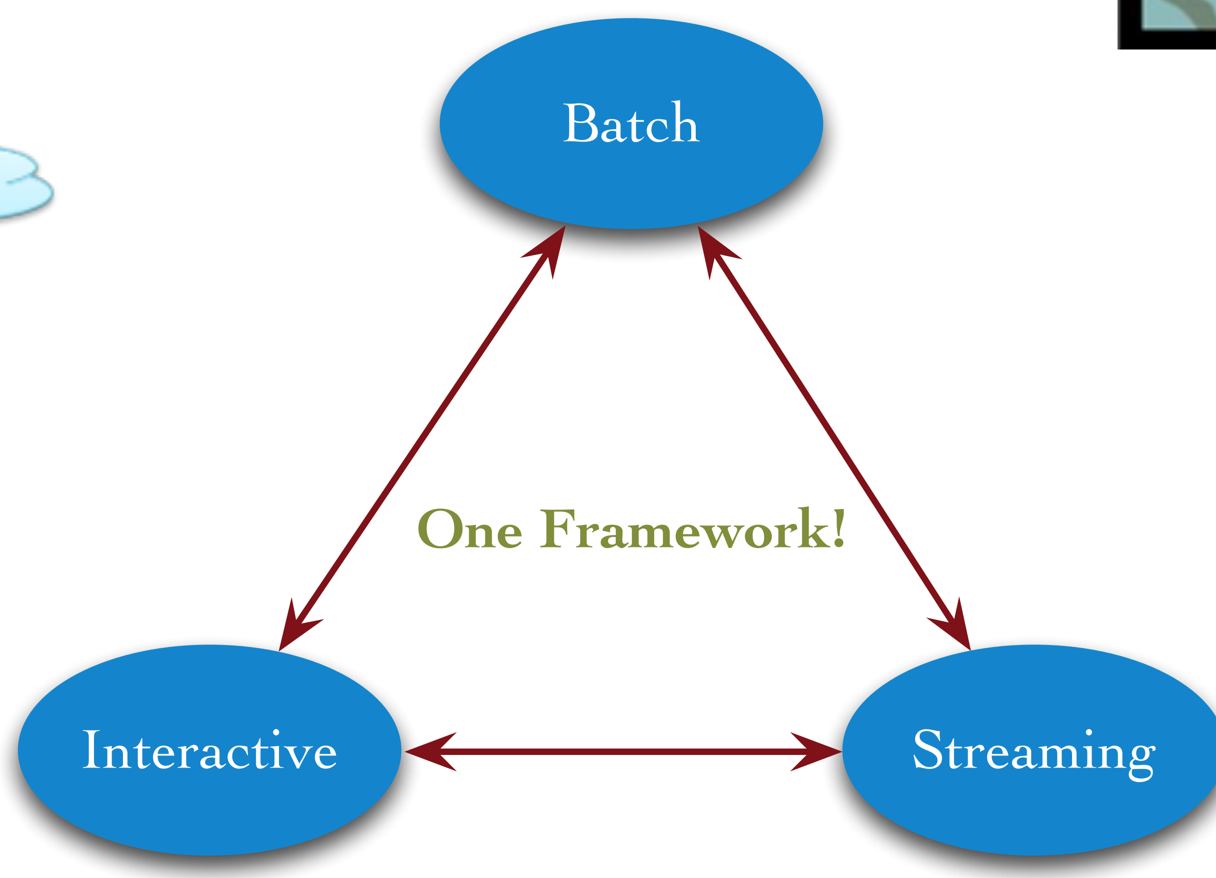
Distributed Frameworks: Spark



- Immutable, partitioned collections of objects stored in RAM across the cluster!
- Built through lazy parallel transformations
- Automatically rebuilt on failure!

Development was motivated by lack of stream-processing applications —

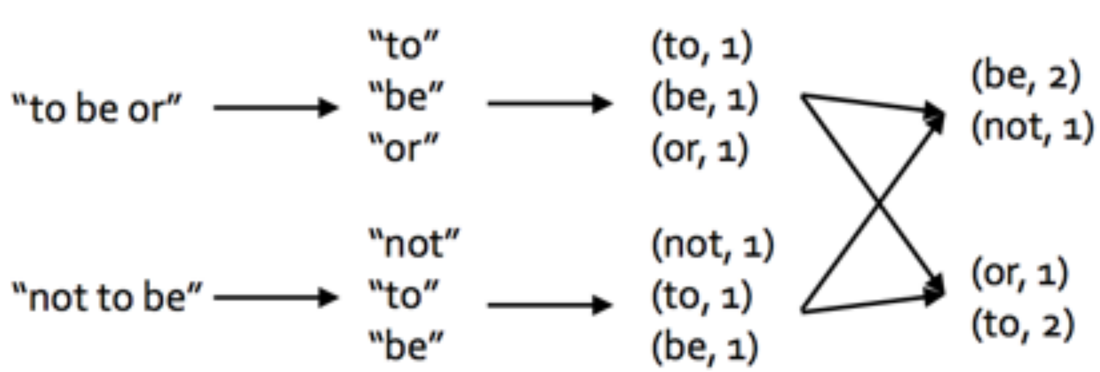
Runs streams as small, deterministic batch jobs —



Spark: Some operations

```
val pets = sc.parallelize(
              List((“cat”, 1), (“dog”, 1), (“cat”, 2)))

pets.reduceByKey(_ + _) // => {(cat, 3), (dog, 1)}

pets.groupByKey() // => {(cat, Seq(1, 2)), (dog, Seq(1)}

pets.sortByKey()  // => {(cat, 1), (cat, 2), (dog, 1)}
```

```
val lines = sc.textFile(“hamlet.txt”)

val counts = lines.flatMap(line => line.split(“ ”))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)
```



Log Mining Example!

```
val lines = spark.textFile(“hdfs://test.txt”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()

messages.filter(_.contains(“hadoop”)).count
messages.filter(_.contains(“graphbuilder”)).count
```

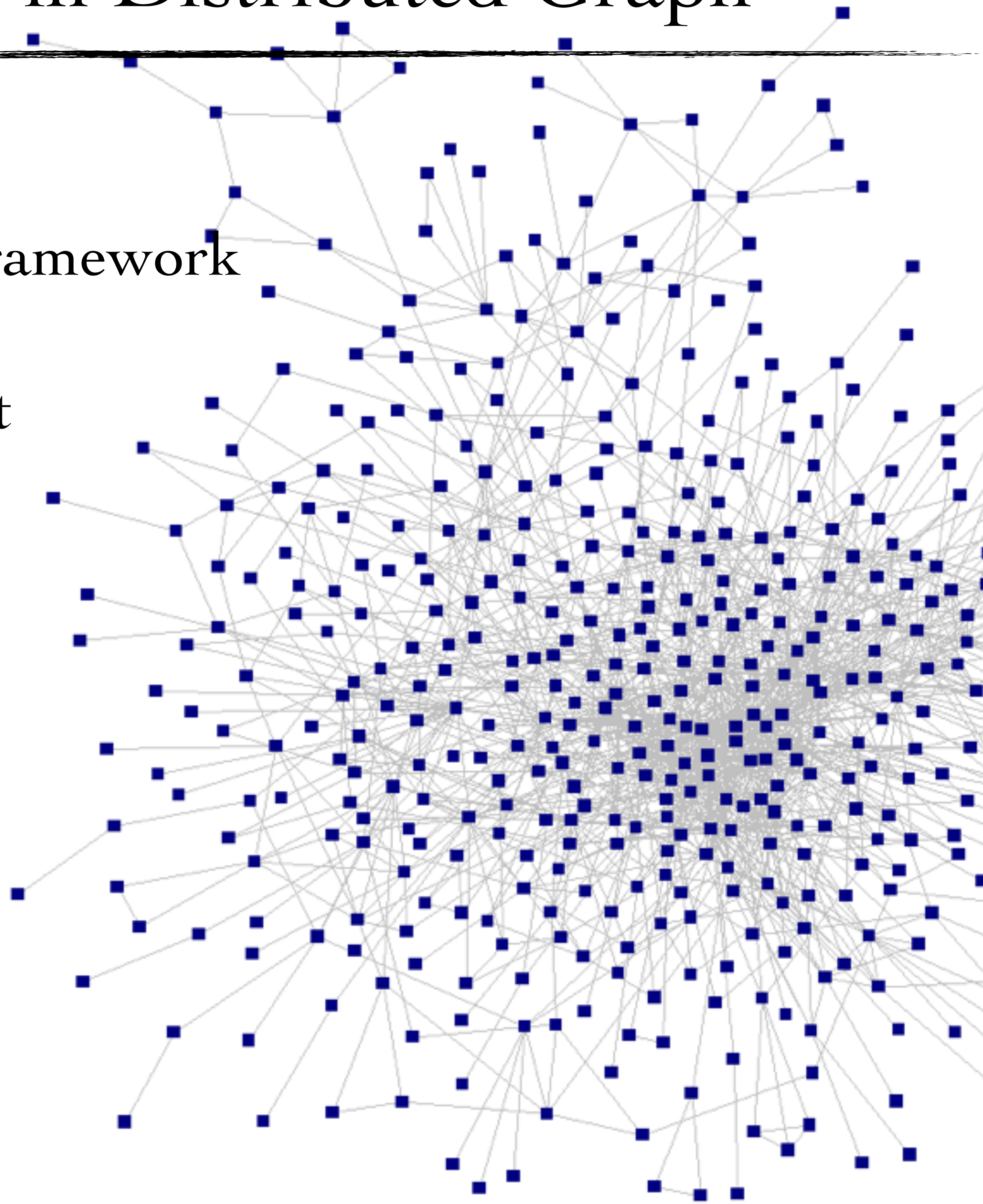
scales to 1TB data in 5-7s!
(170s for on-disk data)

Some Open Questions in Distributed Graph

[1] Distributed computing framework

[2] Graph abstraction

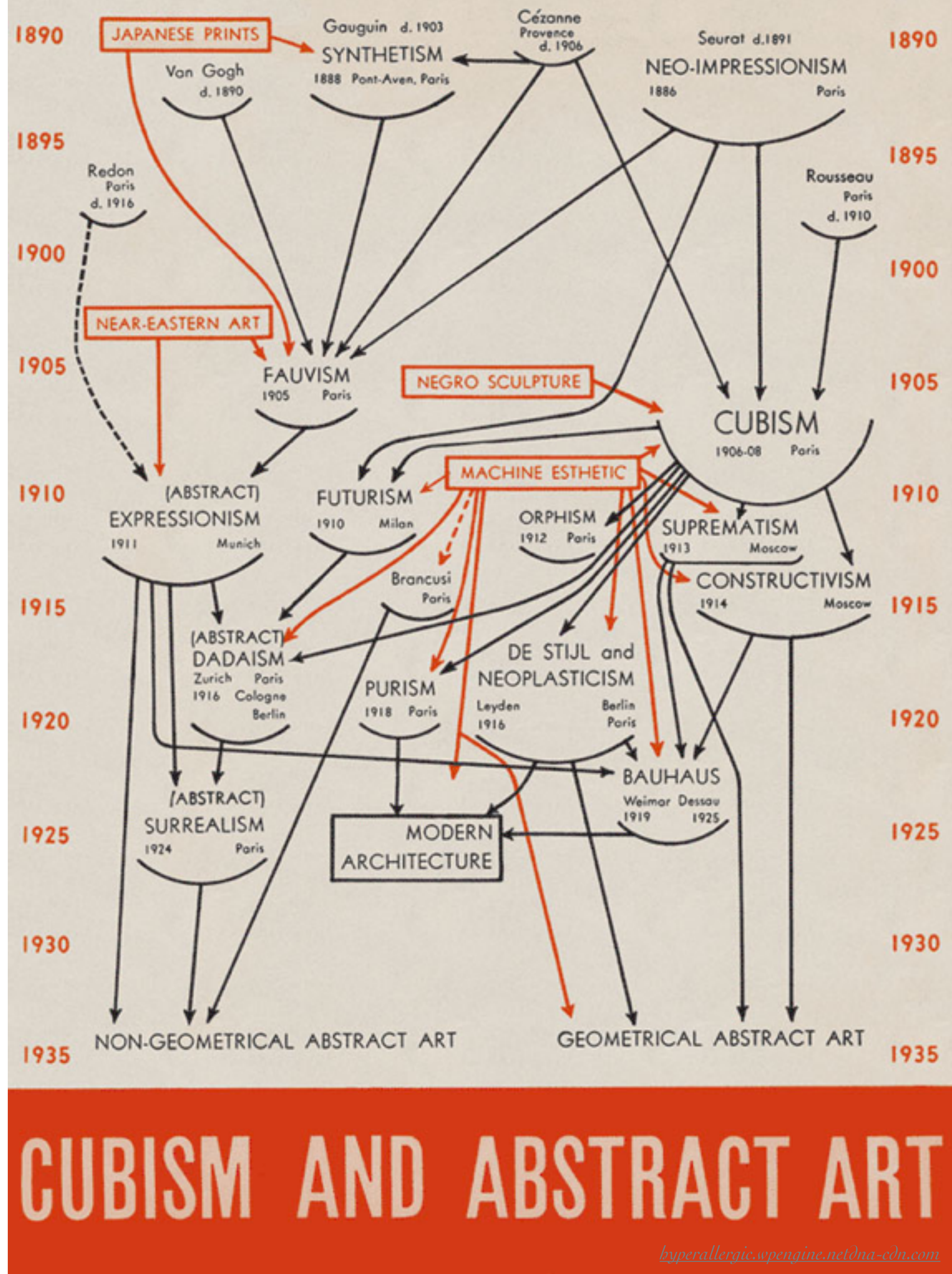
[3] Building a useable toolkit



orgnet.com/alters2.gif

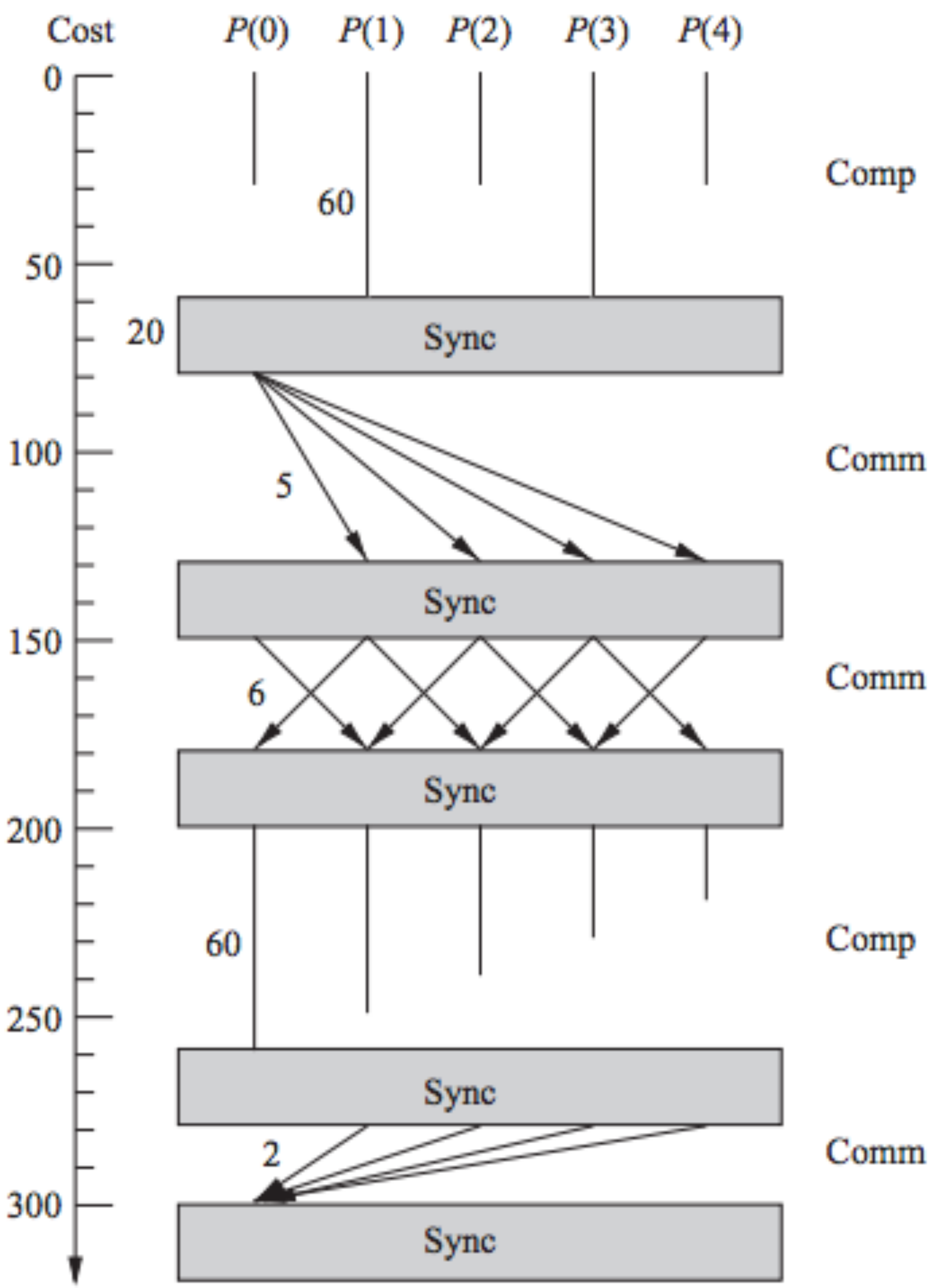
Graph Abstractions

- Not all problems are MapReducible!
- Expressive for graphs, to prevent needing to solve the same challenges over and over.
- Abstractions save us time and allows us to exploit the structure of our data!
  - GAS
  - BSP



Graph Abstractions: *Bulk synchronous parallel*

- Generally, a model of parallel computation
- Vertical and Horizontal structure
- Open source version of Google's Pregel
- Vertex-centric



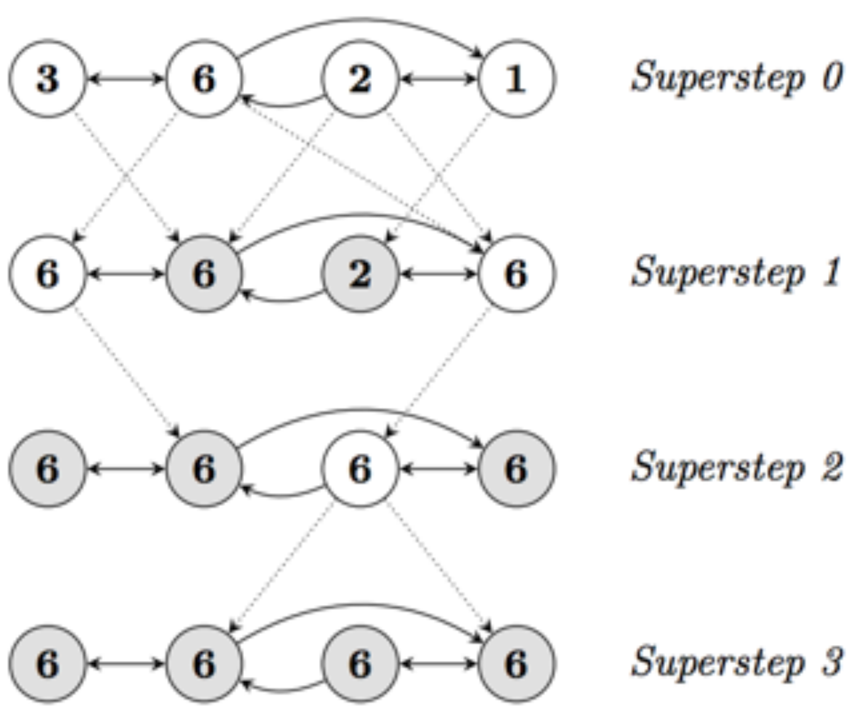
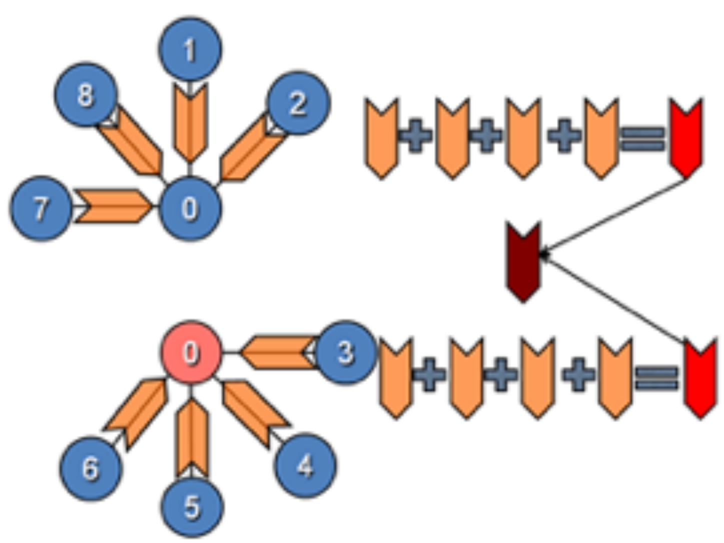


Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

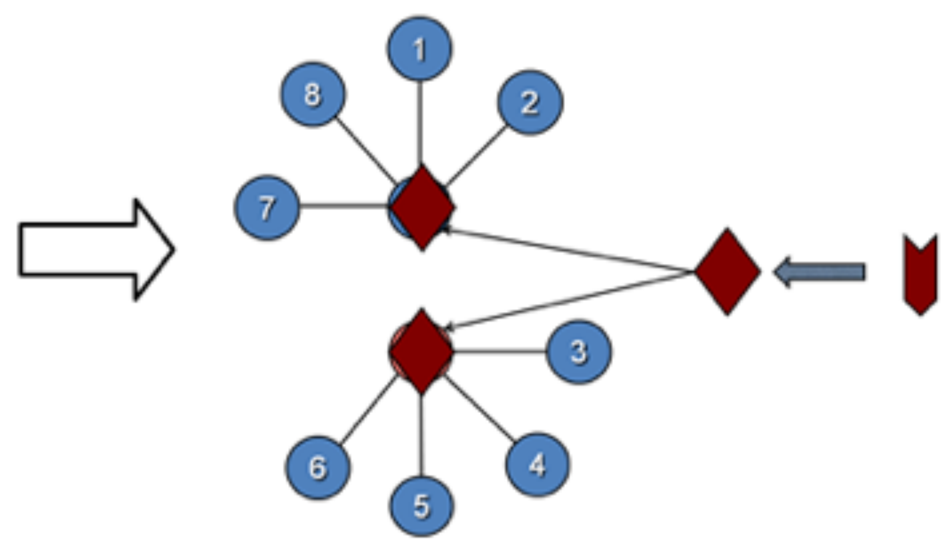
staff.science.uu.nl/~bisse101/Book/PSC/psc1_2.pdf; cs.duke.edu/courses/spring13/compsci590.2/slides/lec14.pdf

Graph Abstractions: *Gather, apply, scatter*

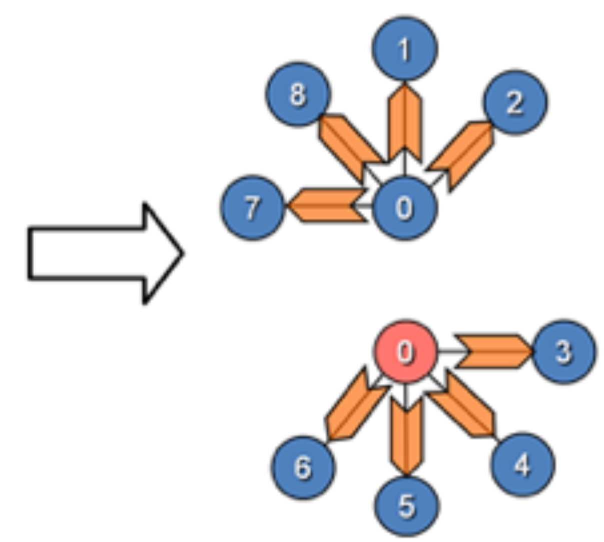
- Exploits graph locality to achieve parallel computation
- Based on BSP, but each vertex is decomposed into three phases



(1)gather



(2)apply

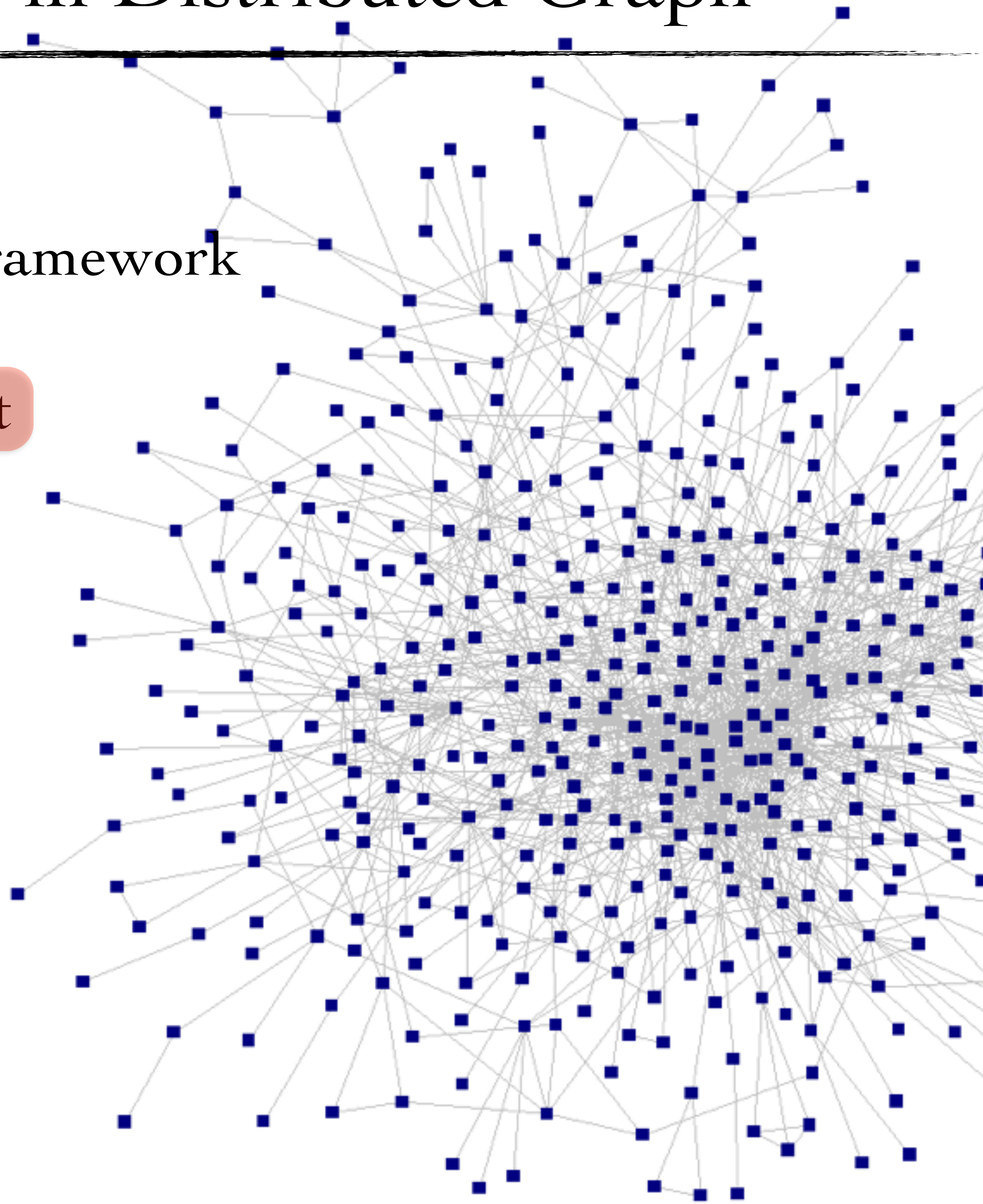


(1)scattter

staff.science.uu.nl/~bisse101/Book/PSC/psc1_2.pdf; cs.duke.edu/courses/spring13/compsci590.2/slides/lec14.pdf

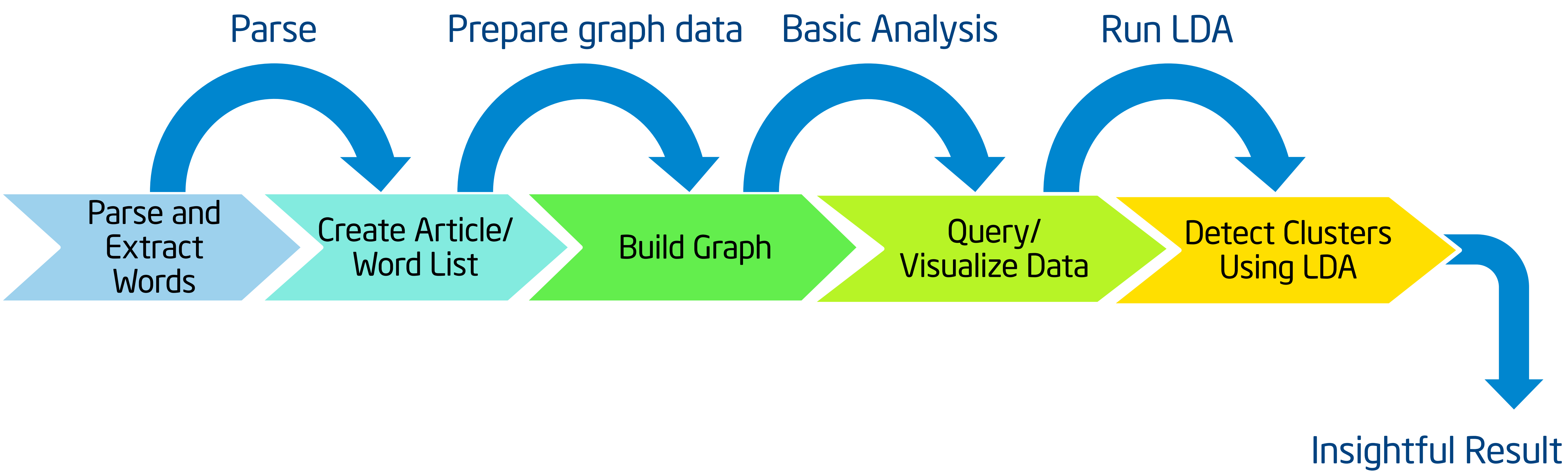
Some Open Questions in Distributed Graph

[1] Distributed computing framework
[2] Graph abstraction
[3] Building a useable toolkit

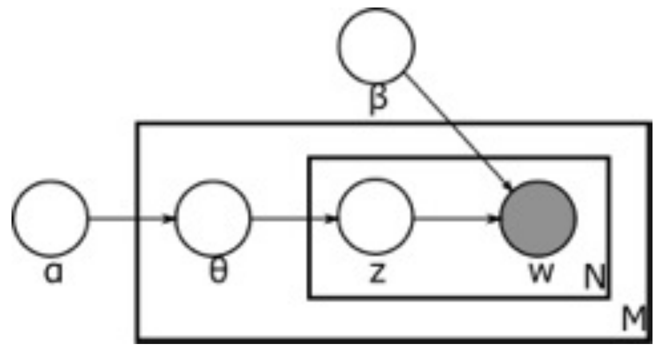


orgnet.com/alters2.gif

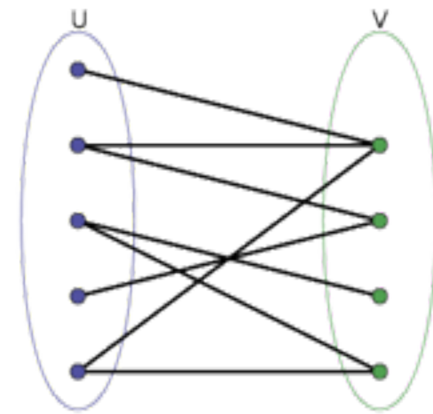
Overview of the Graph Analytics Pipeline

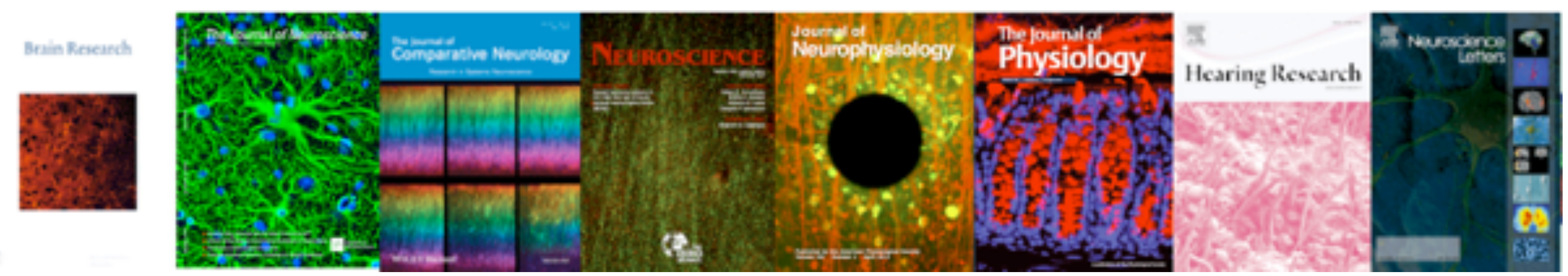


A Motivating Example: Topic Modeling the Hard Way

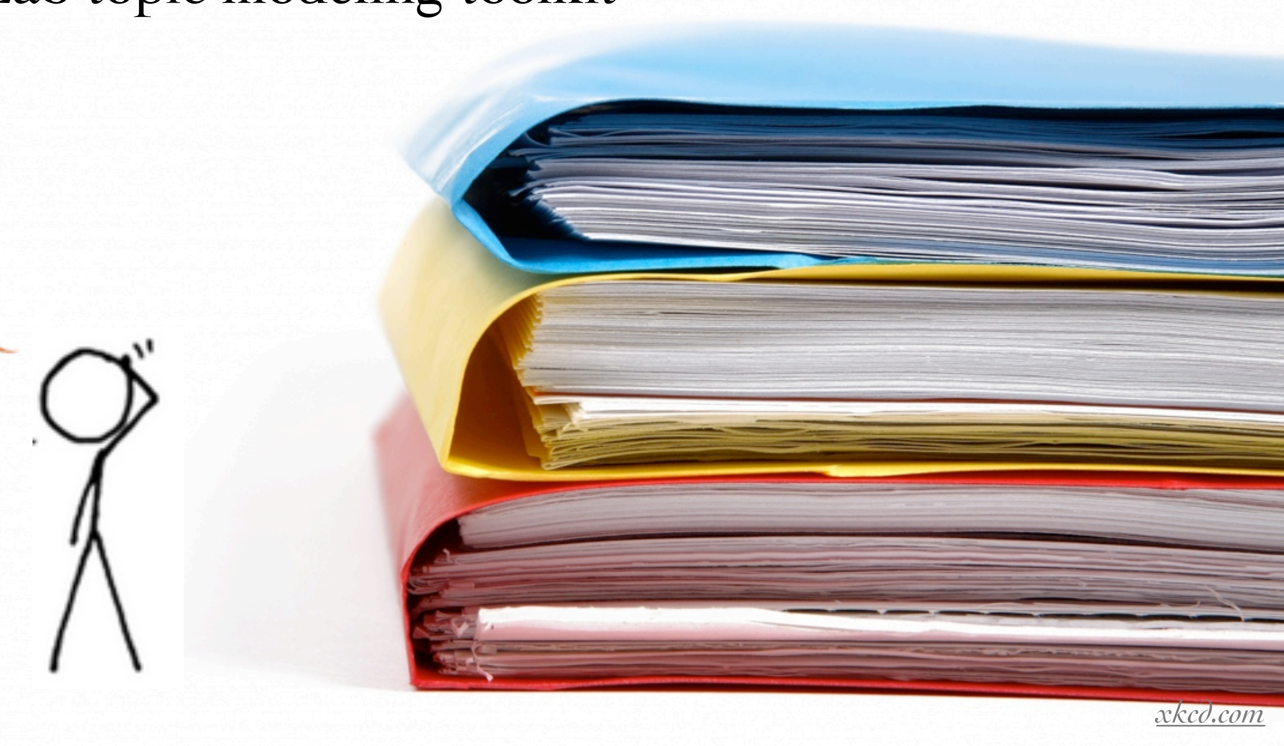


Bipartite

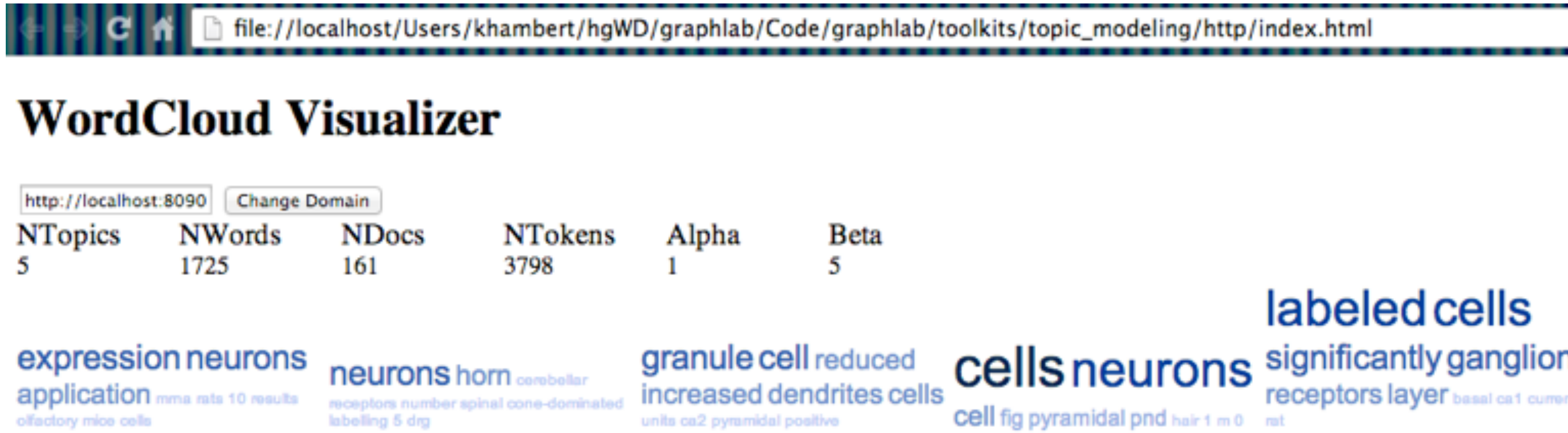


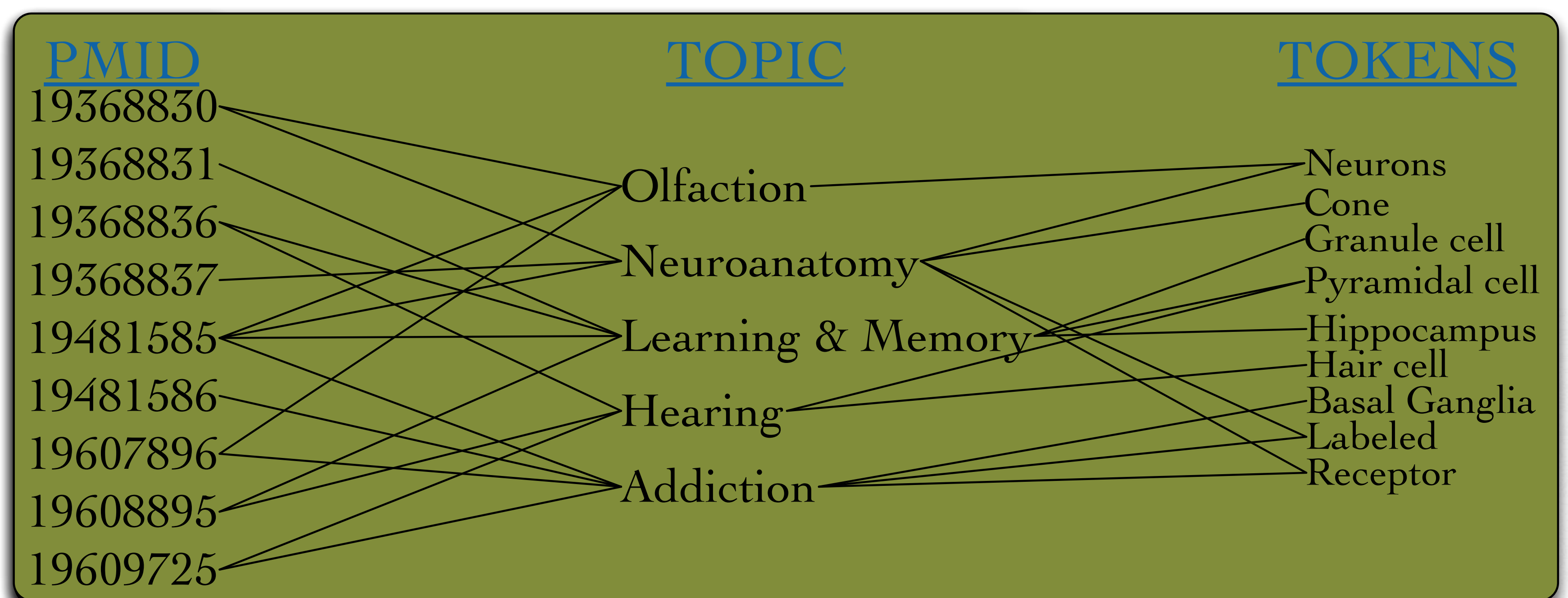


- 161 full text neuroscience publications were acquired (*pdf*).
- The *pdf*s were converted to plain text, and cleaned up.
- The data set was merged and reshaped to a tab-delimited file.
- A bipartite graph was created in static files
- LDA (α=1, β=5, n_{topics}=5) was run in using the GraphLab topic modeling toolkit



A Motivating Example: Topic Modeling the Hard Way

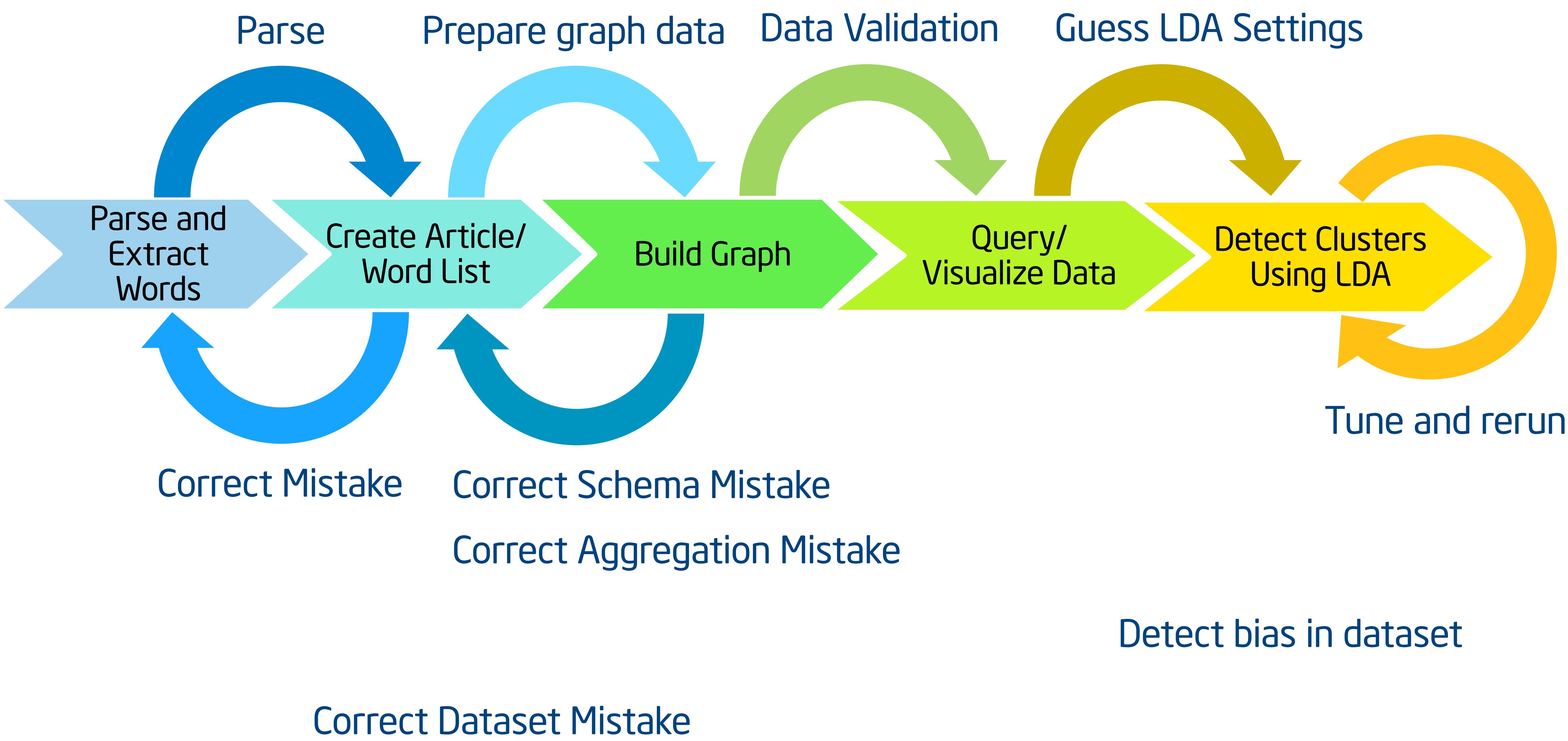




- Reasonably interesting, but follow-up experiments would be helpful
- So I need to re-write my software for the new use-case?

probably.

Overview of the Graph Analytics Pipeline



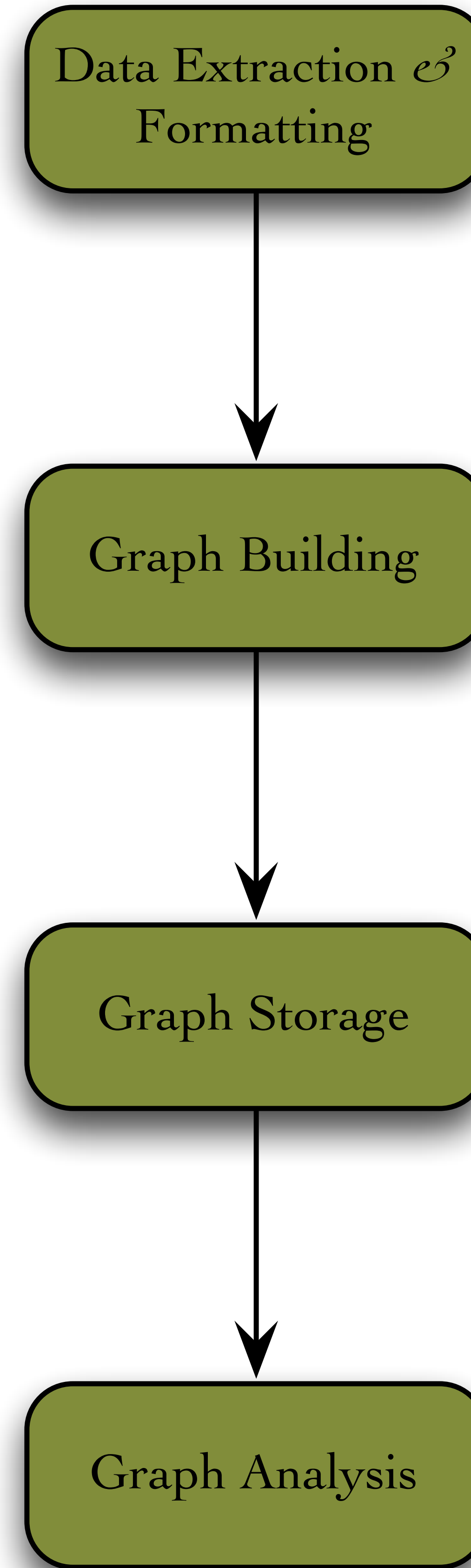
http://imgs.xkcd.com/comics/self_description.png

Graph Analytics Pipeline

- What if there was a re-useable framework for graph analytics?

WEKA
The University of Waikato

- What components would we include?

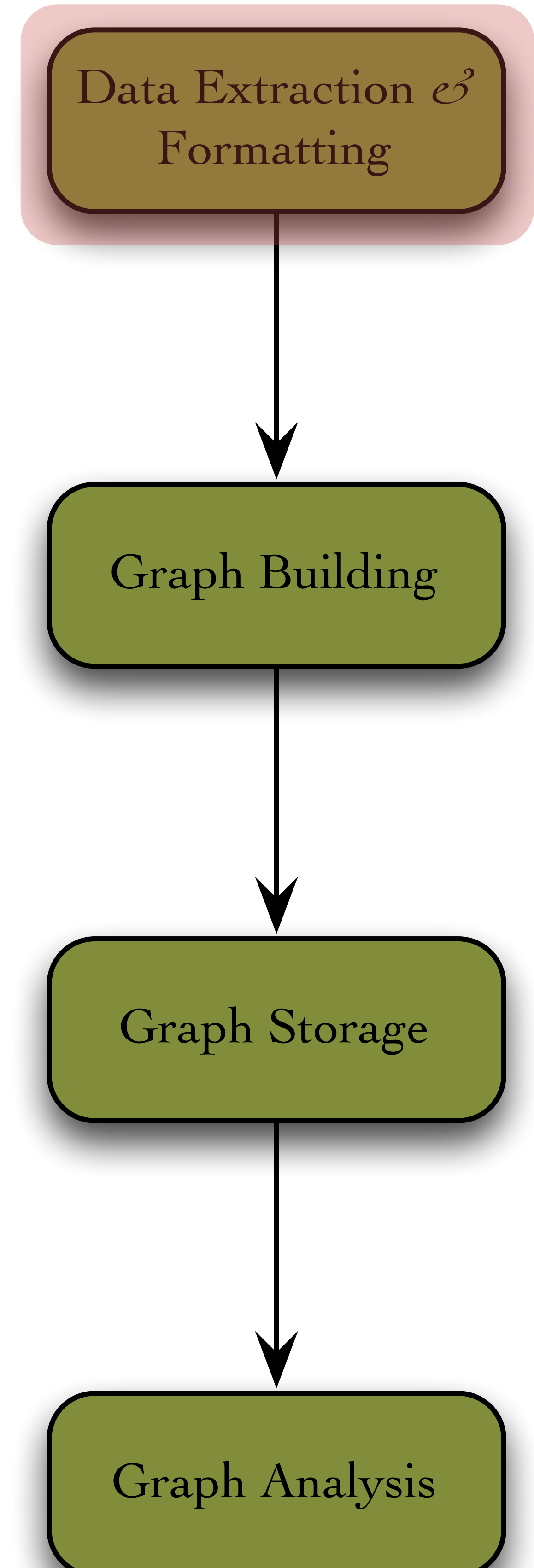


Hall et al., 2009; Ambert & Cohen, [2013, 2012, 2011]

Data Extraction

Goal: Obtain/extract data, convert it into a machine-readable, semi-structured format, & do any necessary pre-processing.

An important step for getting your data into a format for scalable operations

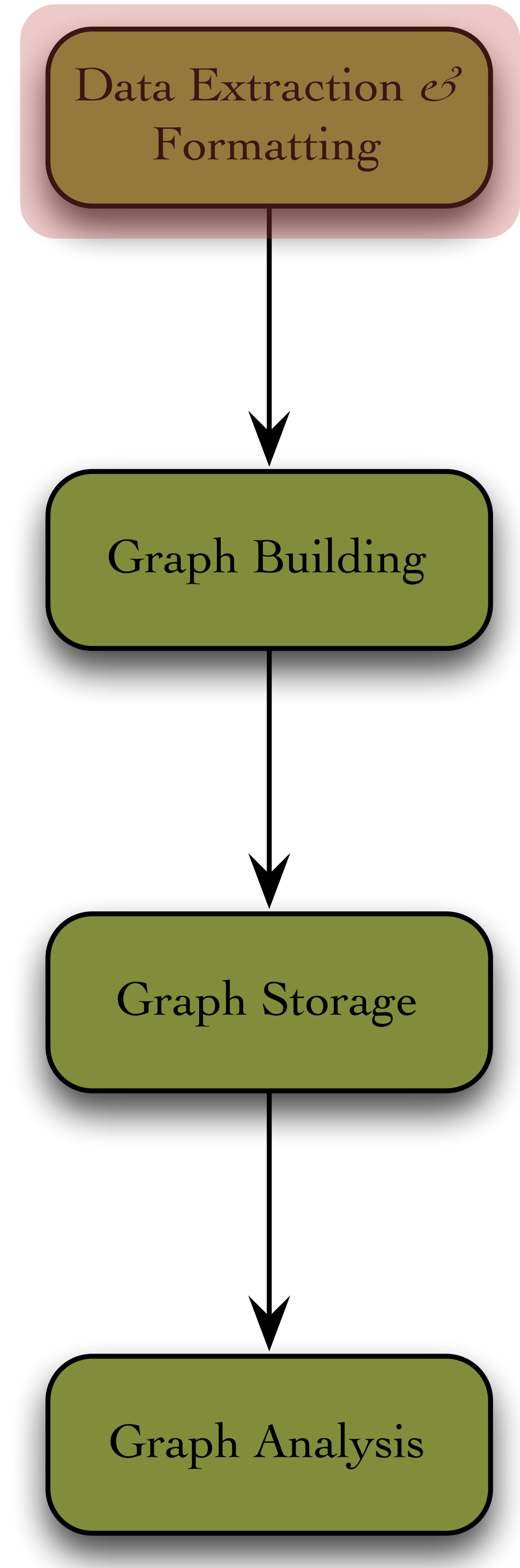


Data Extraction

Goal: Obtain/extract data, convert it into a machine-readable, semi-structured format, & do any necessary pre-processing.

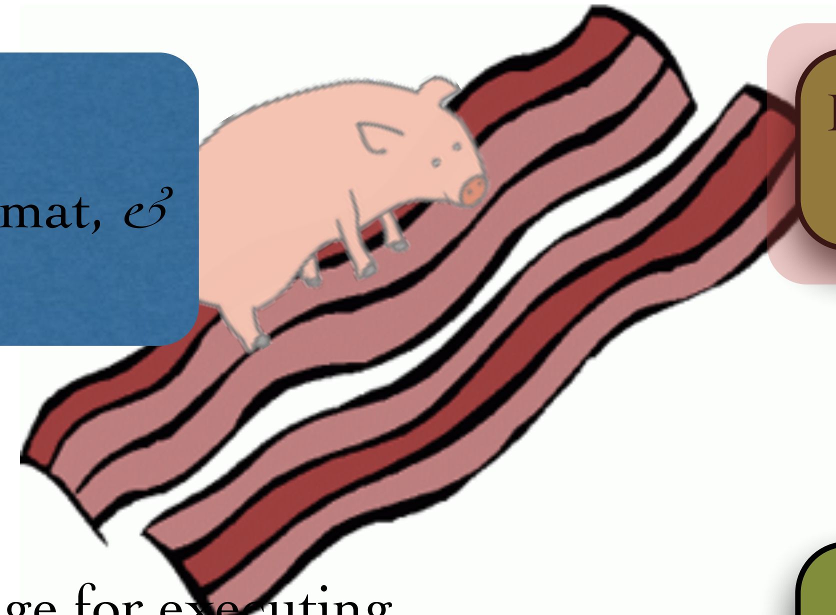
— Python, et al.

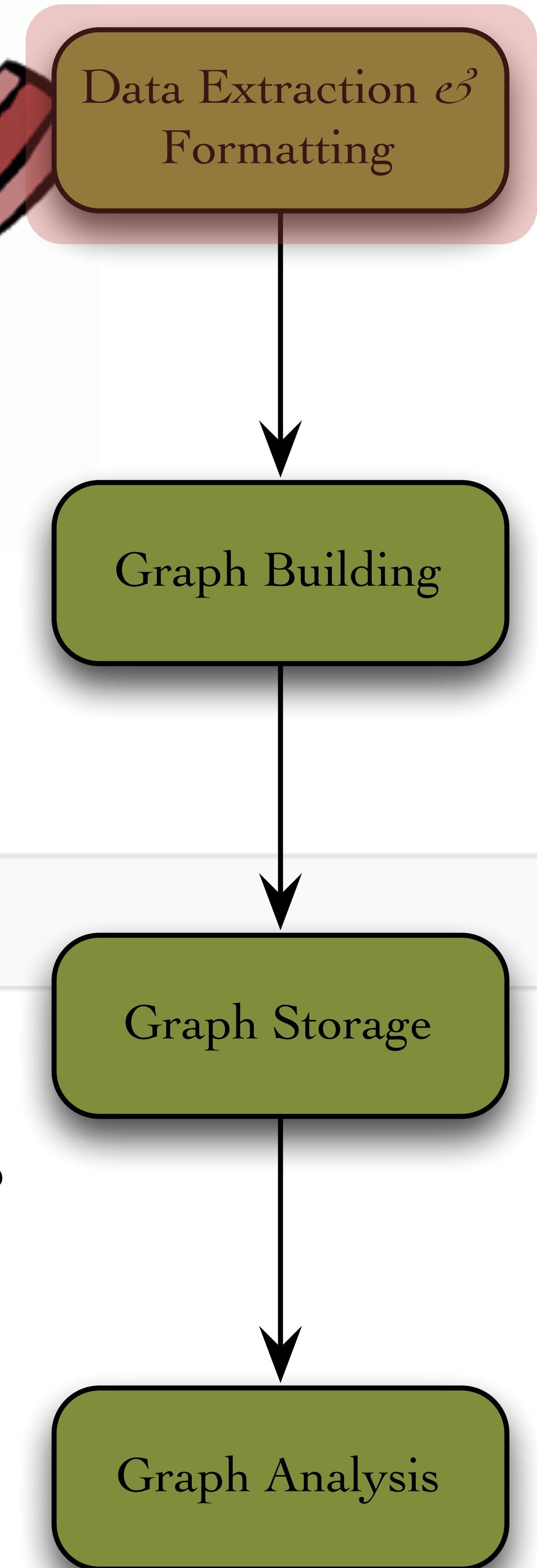
- Data scientists tend to be scripting experts
- Domain expertise leads to effective script development
- This is not likely to change, and a self-contained software pipeline should leverage this.
  - e.g., Python API for text-processing.



Data Extraction

Goal: Obtain/extract data, convert it into a machine-readable, semi-structured format, & do any necessary pre-processing.





- Python, et al.
- Apache Pig

A high-level language for executing “embarrassingly parallel” tasks

Example: look for a regular expression in a set of messages

```
messages = LOAD 'messages';
warns = FILTER messages BY $0 MATCHES '.*WARN+.*';
STORE warns INTO 'warnings';
```

Table 1. Incomplete list of Pig Latin relational operators

Operator	Description
FILTER	Select a set of tuples from a relation based on a condition.
FOREACH	Iterate the tuples of a relation, generating a data transformation.
GROUP	Group the data in one or more relations.
JOIN	Join two or more relations (inner or outer join).
LOAD	Load data from the file system.
ORDER	Sort a relation based on one or more fields.
SPLIT	Partition a relation into two or more relations.
STORE	Store data in the file system.

Good for processing semi-structured data over Hadoop

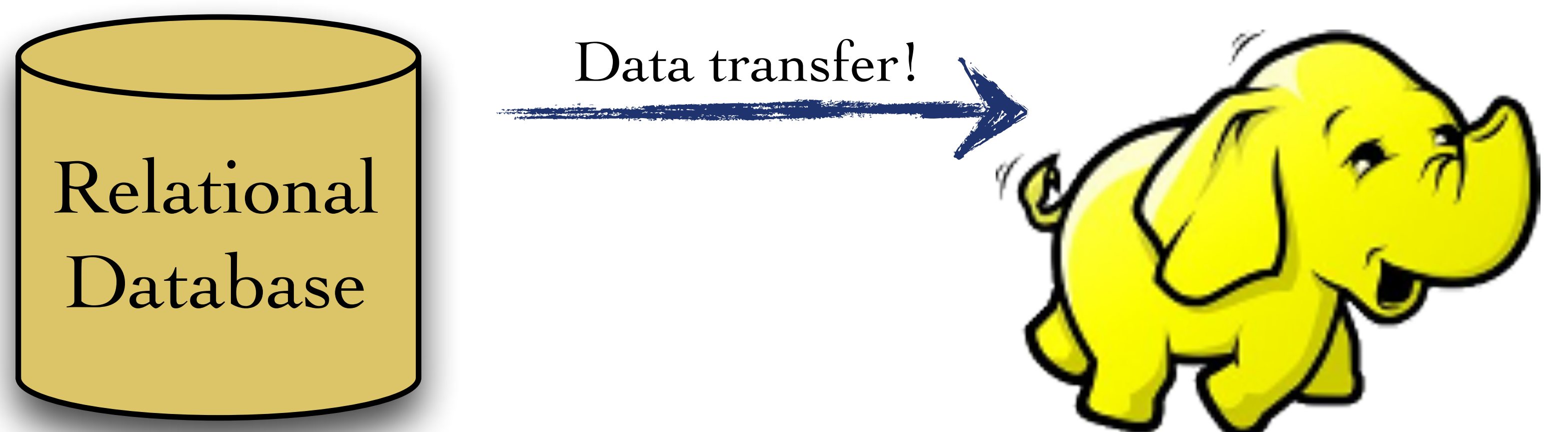
Good for filtering and merging data efficiently

http://4vector.com/i/free-vector-pig-clip-art_119373_Pig_clip_art_hight.png

Data Extraction

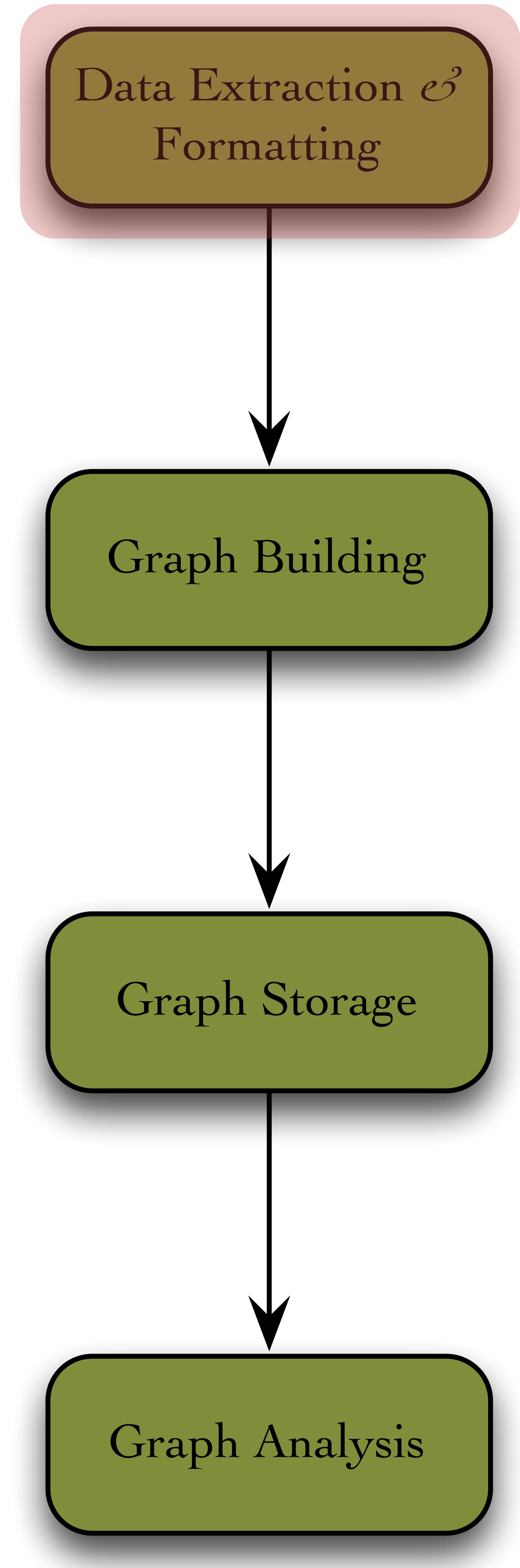
Goal: Obtain/extract data, convert it into a machine-readable, semi-structured format, & do any necessary pre-processing.

- Python, et al.
- Apache Pig
- Sqoop




```
sqoop import --connect <JDBC connection string> --
table <tablename> --username <username> --password
<password>
```

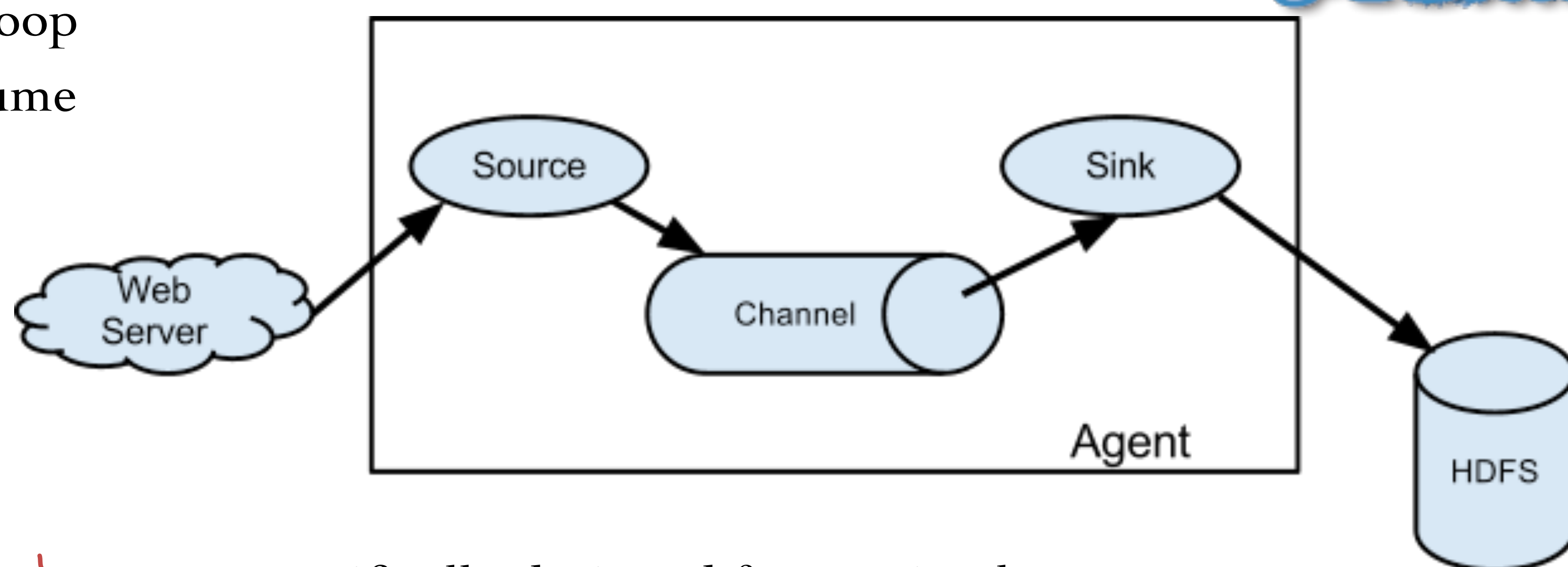
- Incremental loading
- Can be used to populate tables in HBase



Data Extraction

Goal: Obtain/extract data, convert it into a machine-readable, semi-structured format, & do any necessary pre-processing.

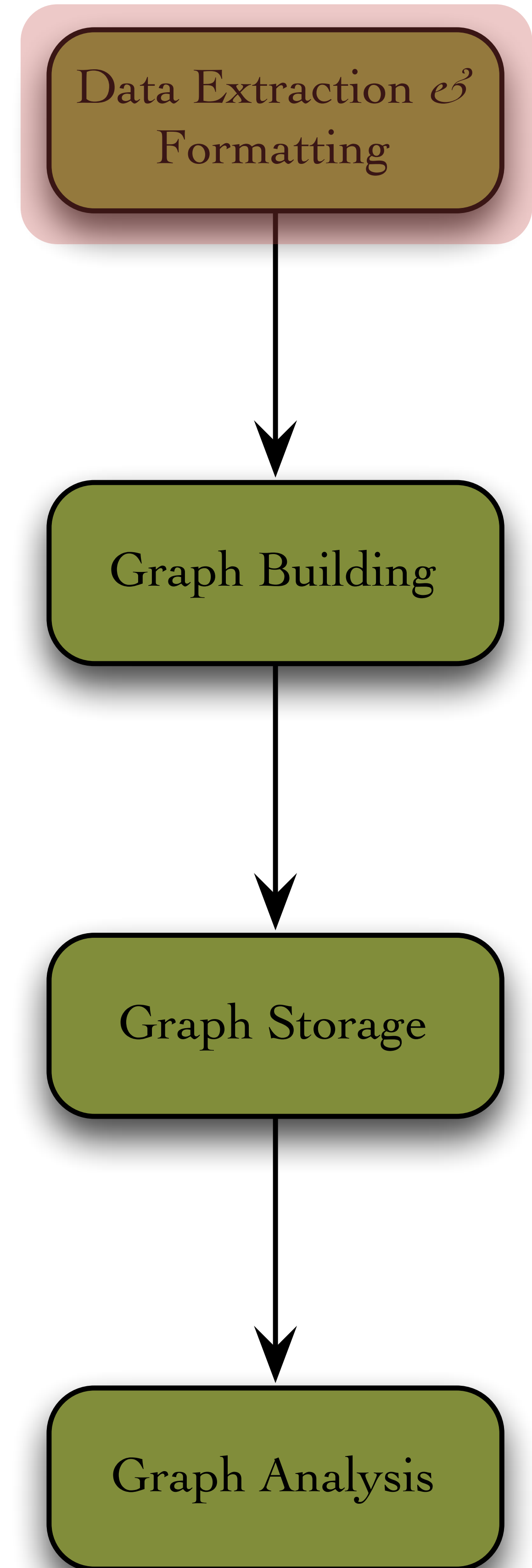
- Python, et al.
- Apache Pig
- Sqoop
- Flume



→ Specifically designed for moving large amounts of log files

→ Source consumes events from external source

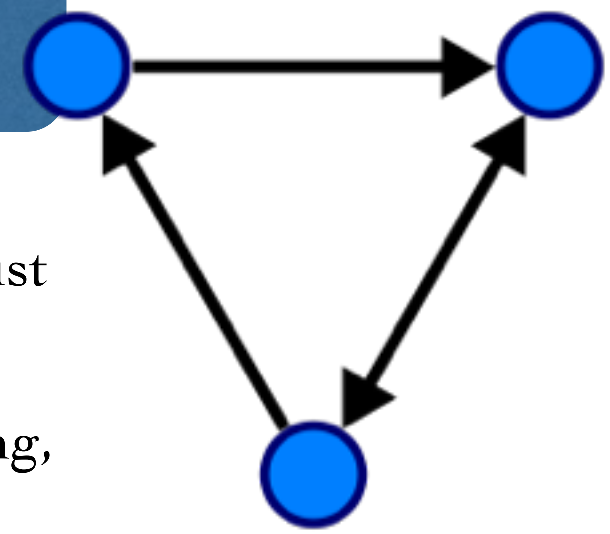
→ Data is stored in one or more channels until it is consumed by the Flume sink, which puts it in an external repository (e.g., HDFS)

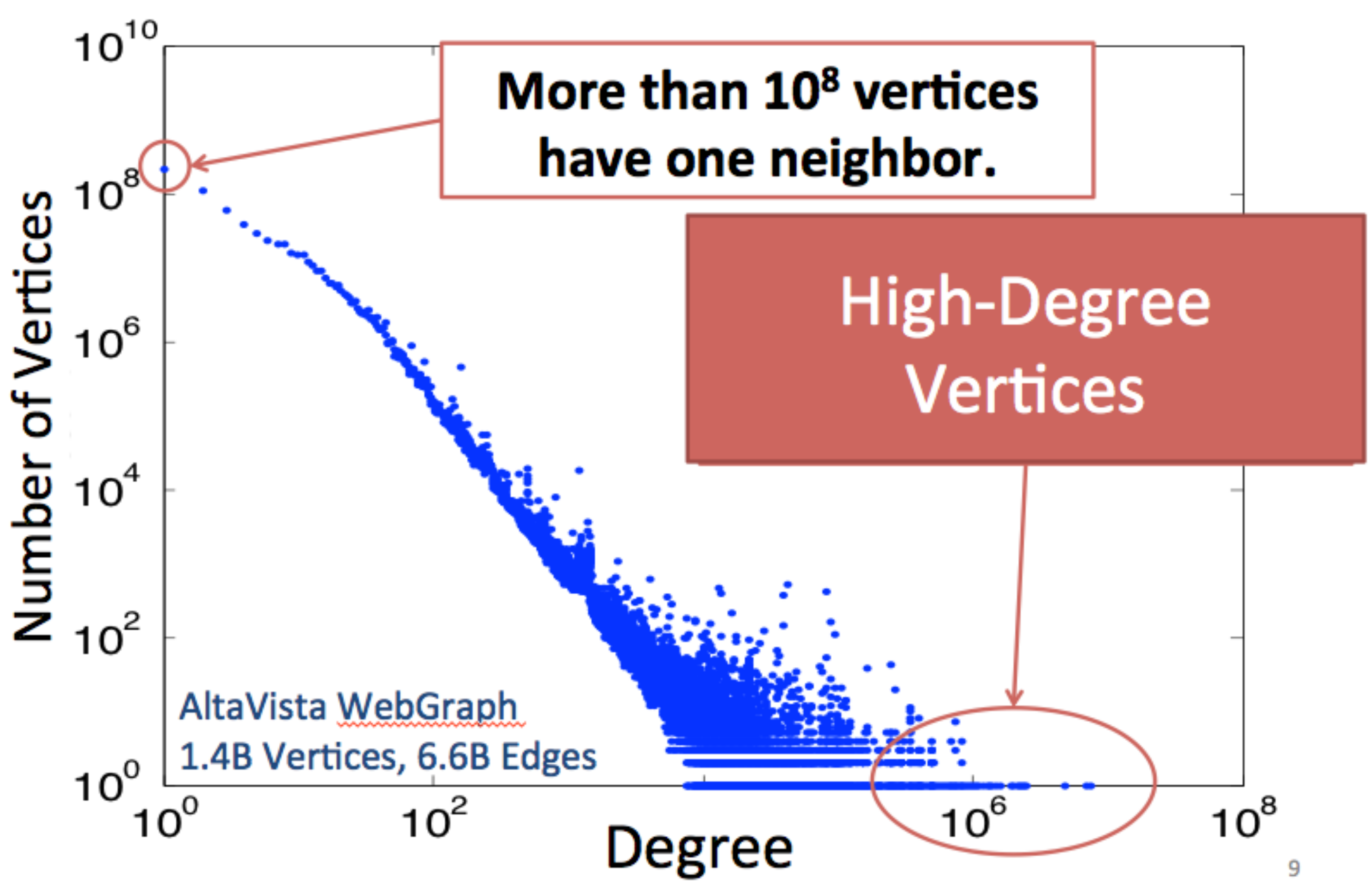


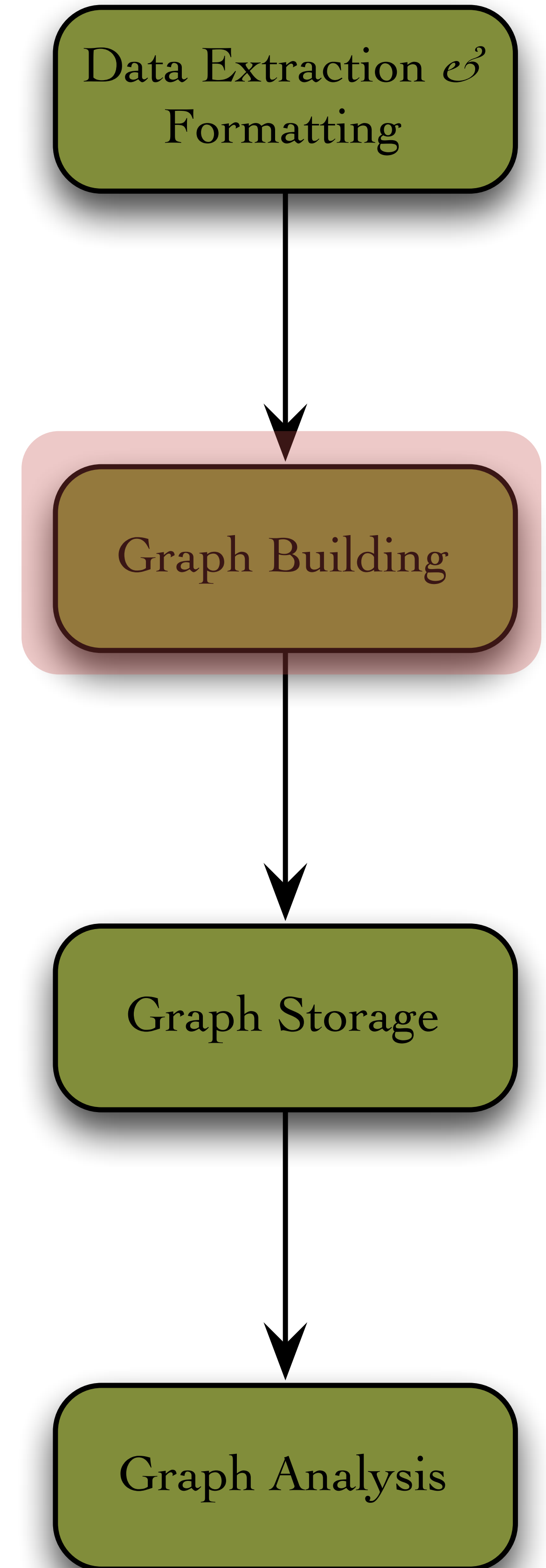
Building Graphs

Goal: Take semi-structured data, and format it as a graph.

- To analyze with standard graph libraries, data must be formatted in a way they understand
- Encompasses pre-processing, additional formatting, and partitioning







Building Graphs

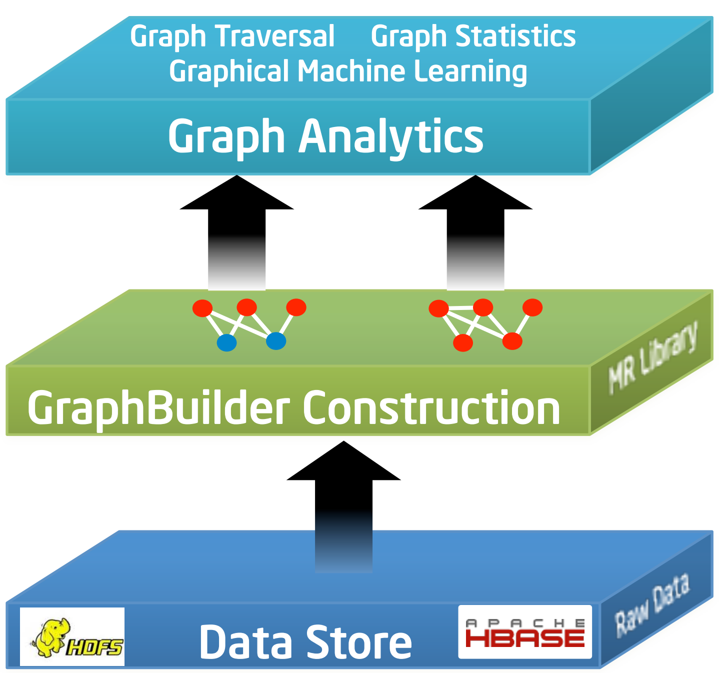
Goal: Take semi-structured data, and format it as a graph.

<docid>	<wordid>	<count>
0	0	2
0	4	1
1	2	3
1	1	5
2	2	3
2	3	7

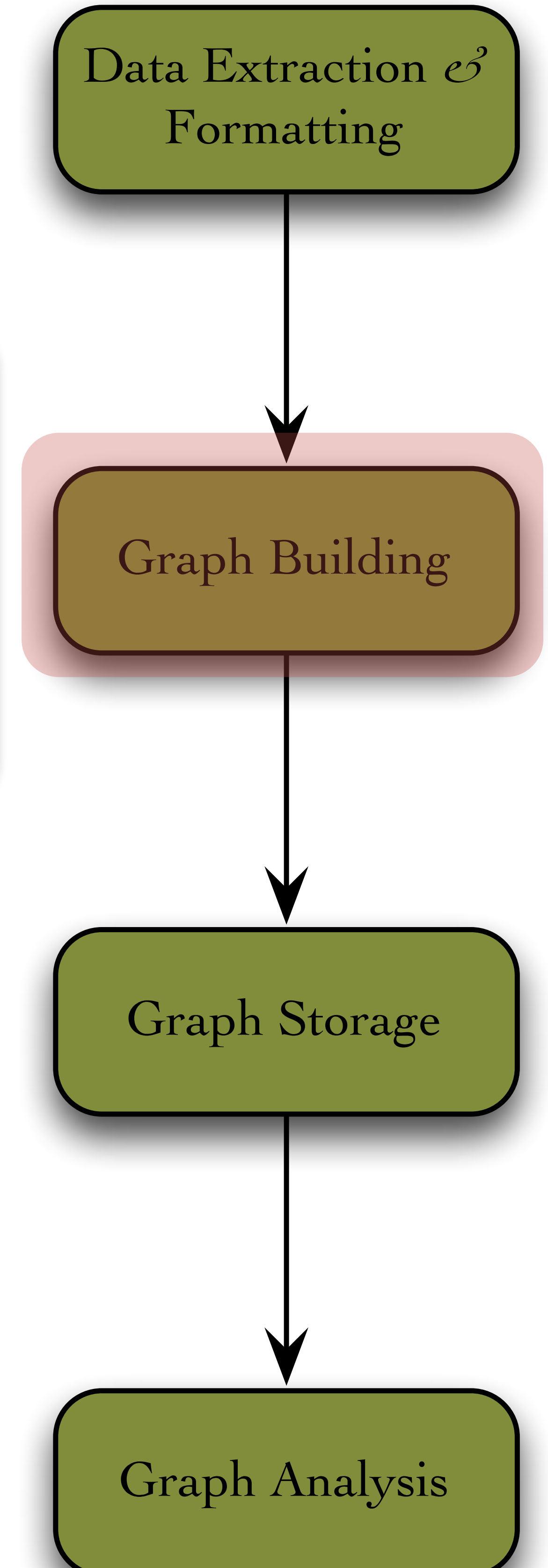
<wordid>	<word>
0	present
1	study
2	investigated
3	anatomical
4	distribution
5	cannabinoid-1
6	receptor
7	cb1r
8	lc
9	association
10	mu-opioid

WIKIPEDIA

GraphBuilder

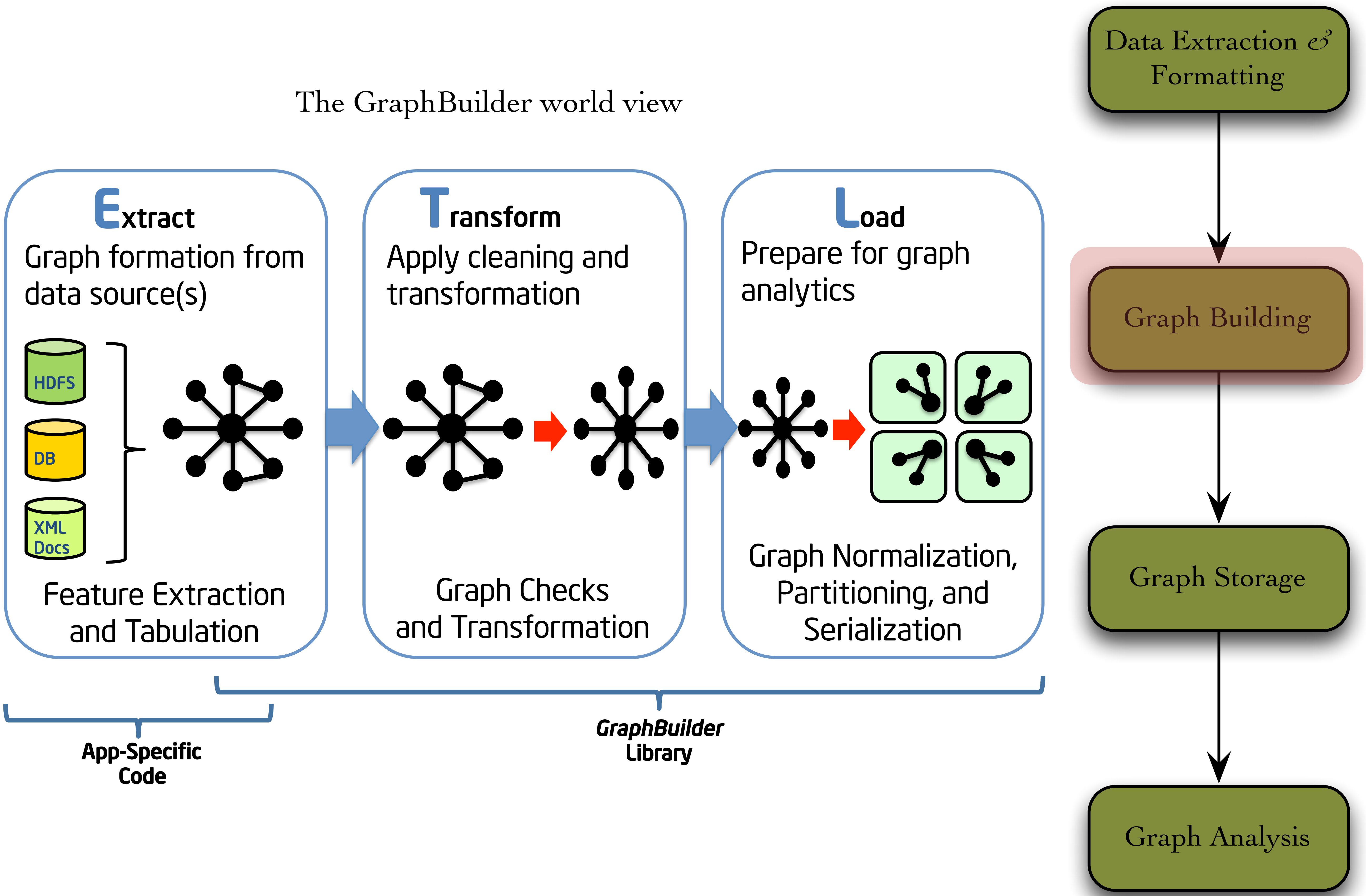


- Offloads domain expertise
- Written in Java for convenient integration with Hadoop MapReduce and applications
- Apache 2.0 OS code available at: www.01.org/graphbuilder



Building Graphs

The GraphBuilder world view

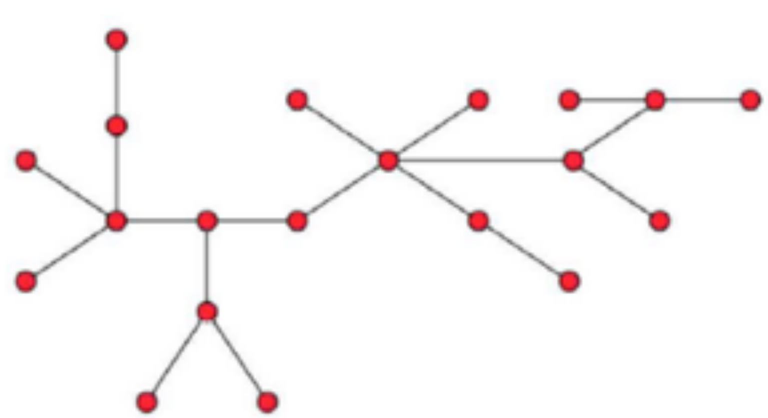


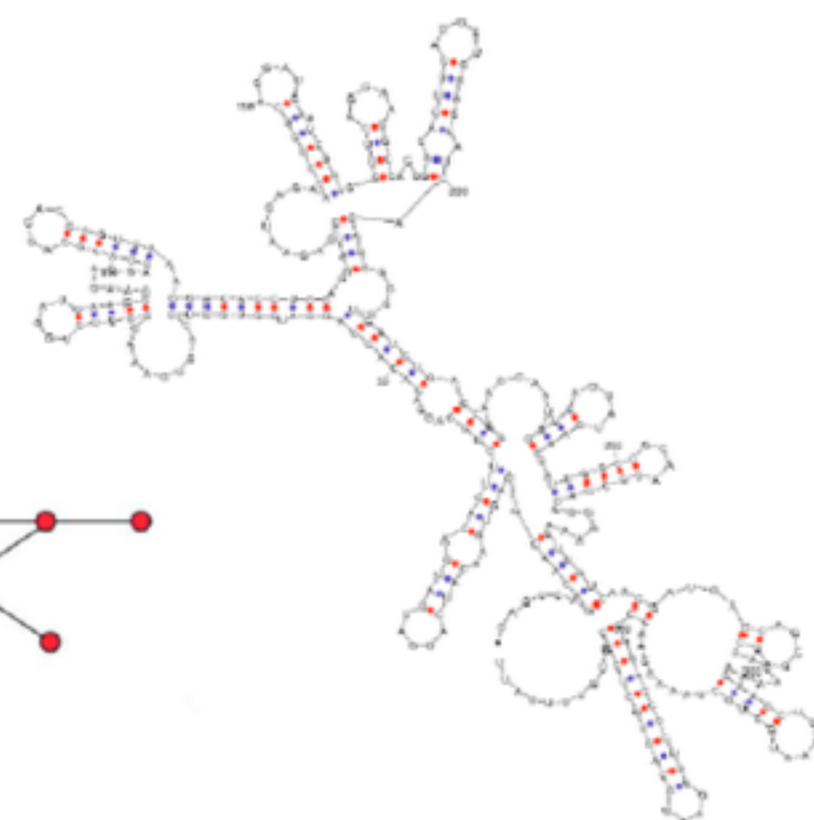
Storing Graphs

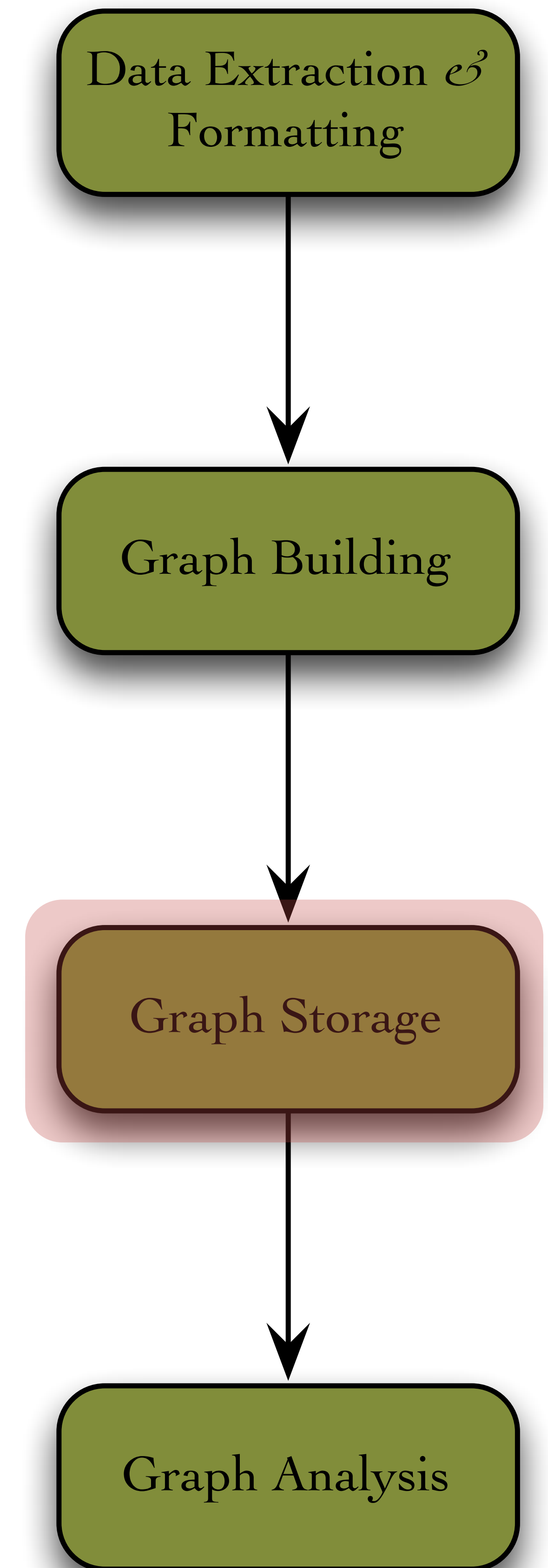
Goal: Store graph data for efficient querying.

- Why not use a relational database?
- Graph databases provide index-free adjacency
  - every node knows what it's connected with!

- Useful for making sequence comparisons







Storing Graphs

- FlockDB
- Titan
  - → ACID & eventual consistency
  - → Can support thousands of concurrent users executing complex graph traversals
  - → Scalability!

Storage Backend Options

cassandra

HBASE

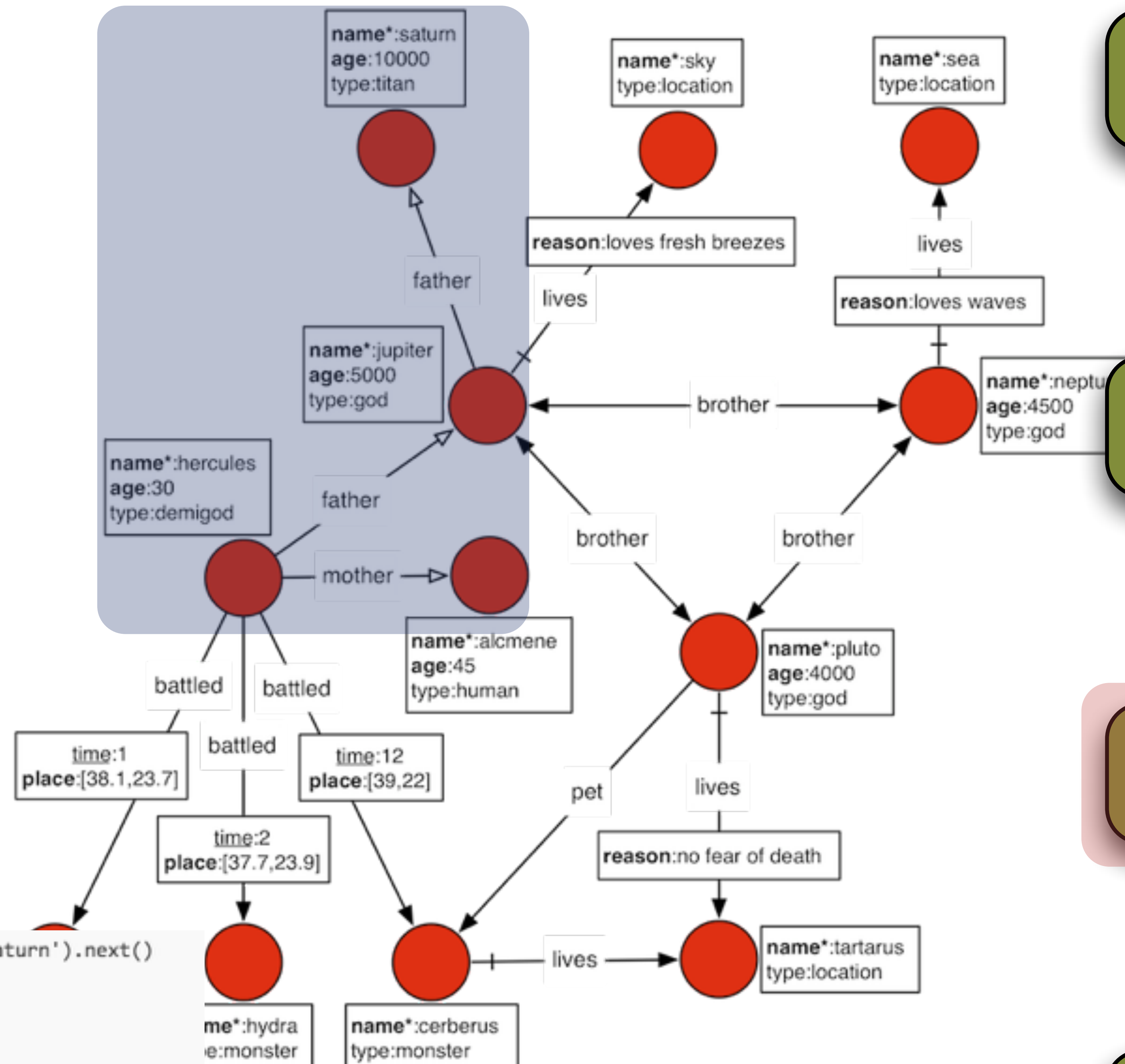
ORACLE BERKELEY DB

→ Supports property graphs

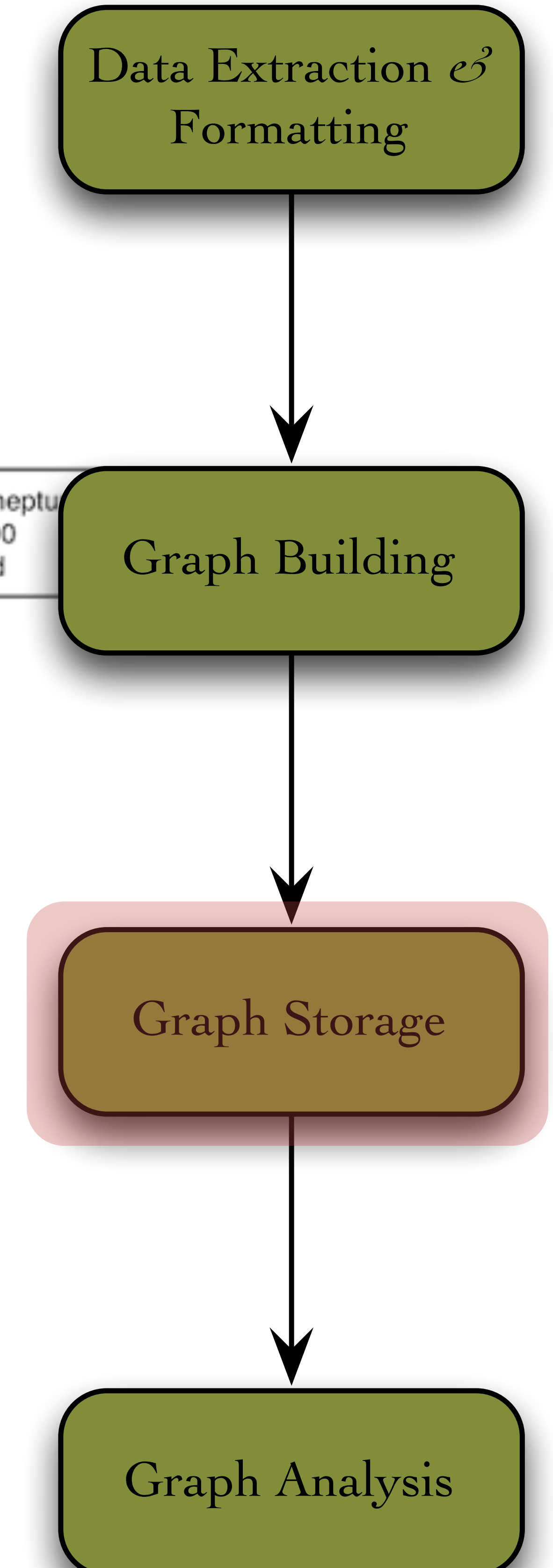
$G = (V, E, \lambda)$

Data Extraction & Formatting → Graph Building → Graph Storage → Graph Analysis

Storing Graphs

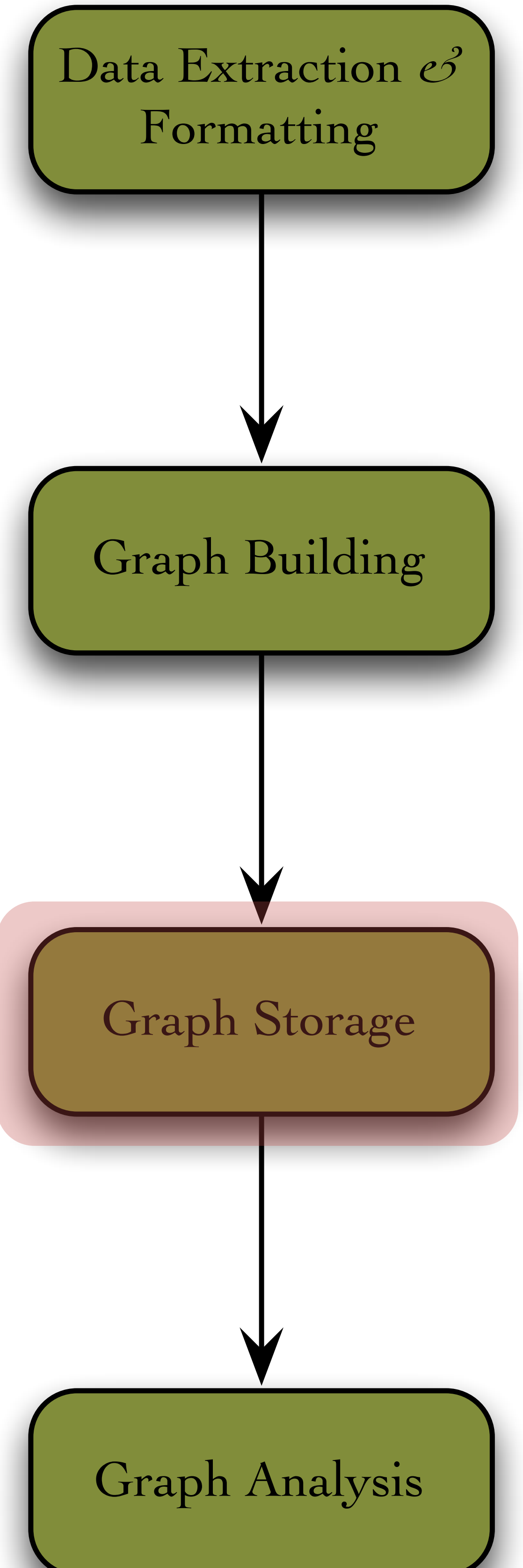



```
gremlin> saturn = g.V('name','saturn').next()
==>v[4]
gremlin> saturn.map()
==>name=saturn
==>age=10000
==>type=titan
gremlin> saturn.in('father').in('father').name
==>hercules
```



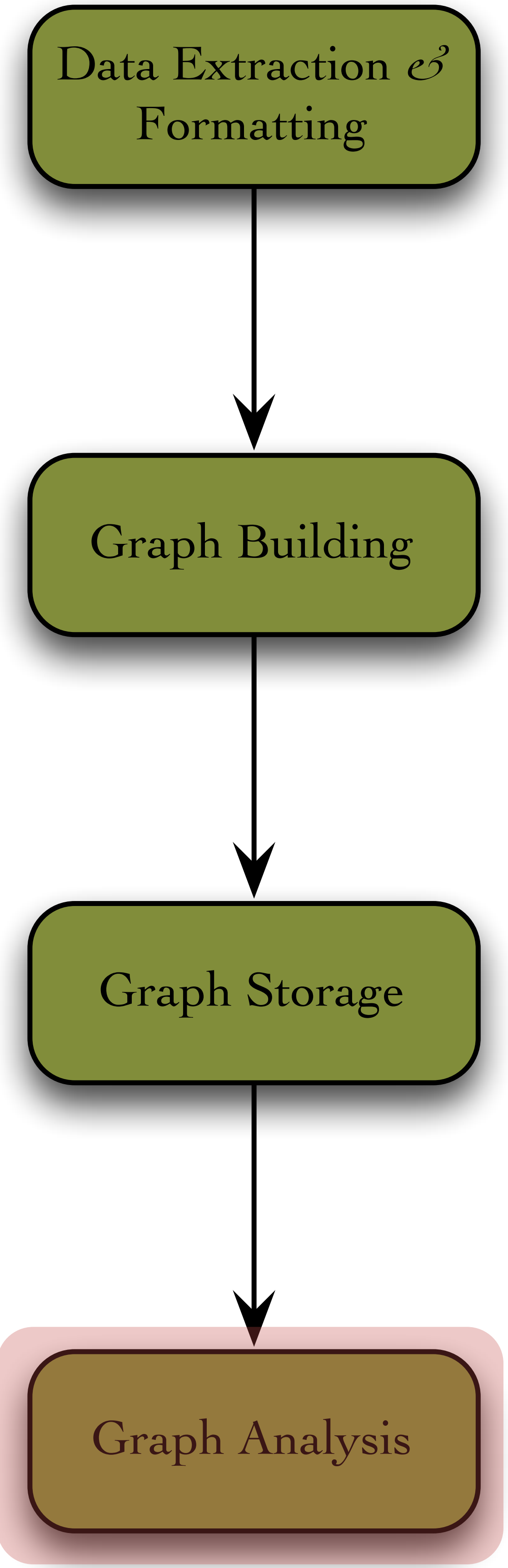
Storing Graphs

HBASE	cassandra
BigTable (Google)	DynamoDB (Amazon)
Hadoop	Storm or Hadoop
Optimized for reads	Optimized for writes
Optimized for reads, range-based scans	Excellent single-row reads, selecting rows based on column-value index
Easily scales horizontally	Rows larger than 10's MB are problematic
Pig integration	Pig integration



Graph Analyses

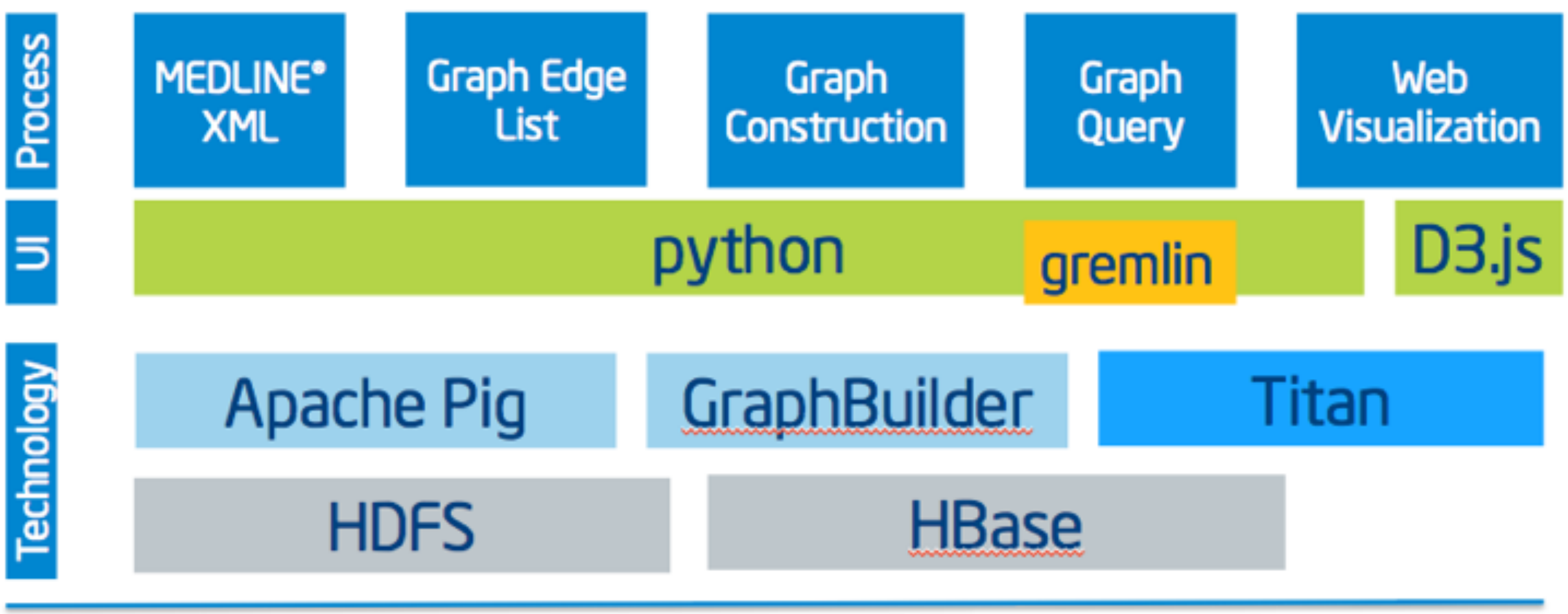
Goal: Synthesize the graph data into statistics & conclusions.

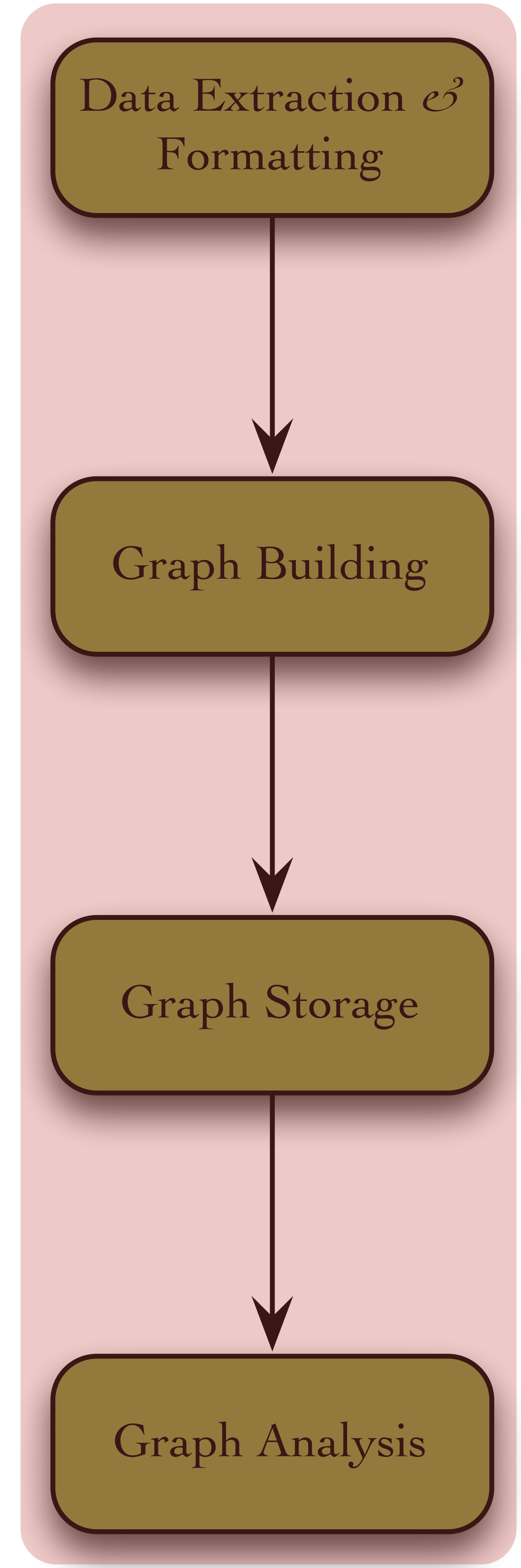


ibm.com/developerworks/library/os-giraph/

Graph Analysis Pipeline

Goal: Make graph analytics scalable and simple.



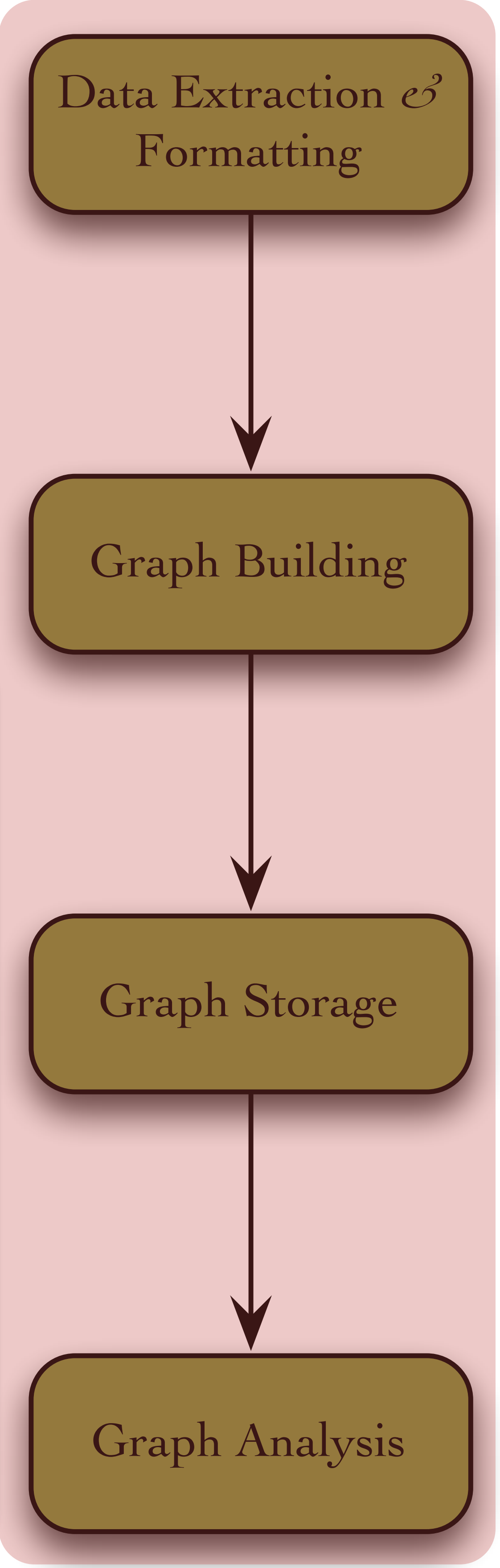


ibm.com/developerworks/library/os-giraph/

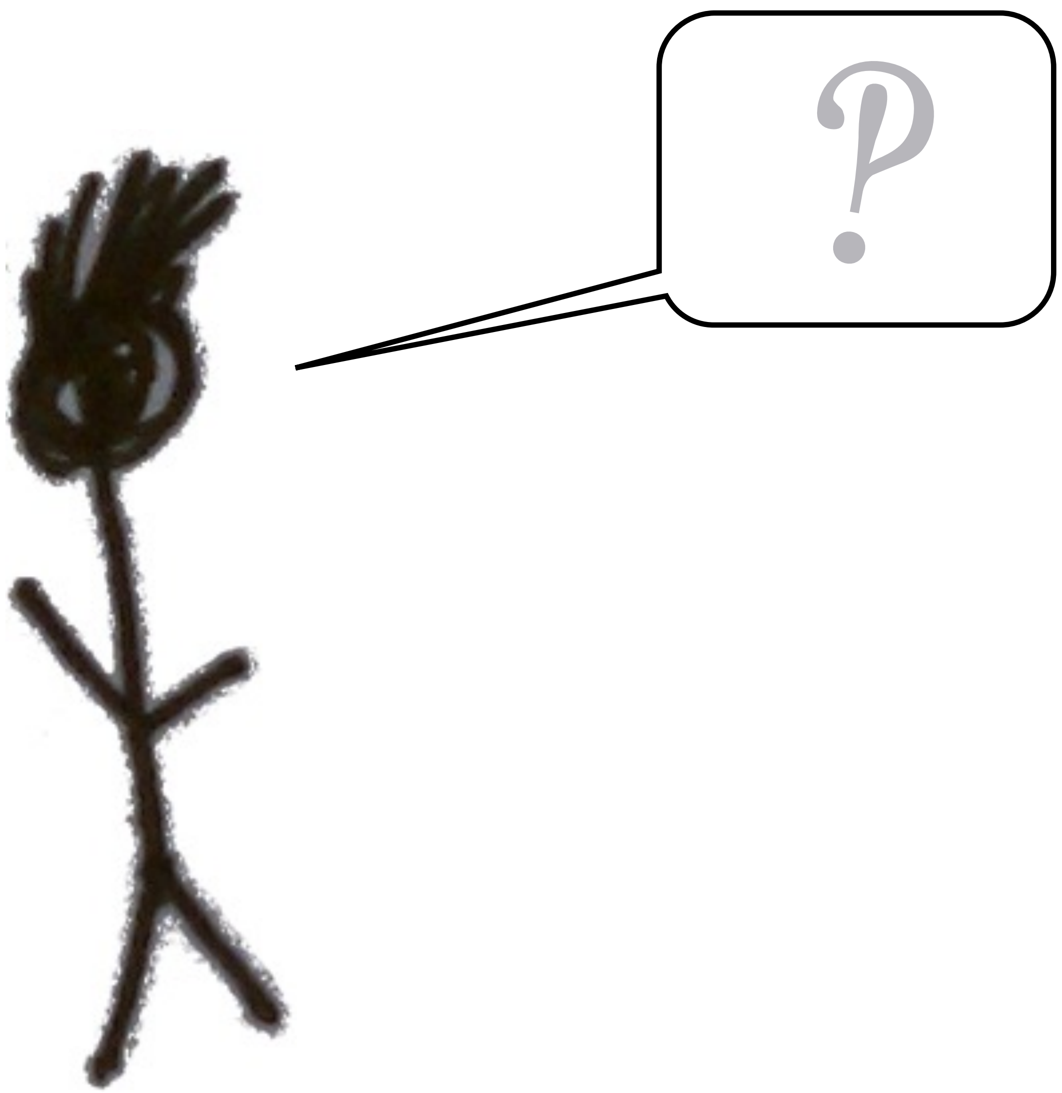
Graph Analysis Pipeline

Goal: Make graph analytics scalable and simple.

Approach	Algorithm	Category	Applications/Use Cases
Graph	Loopy Belief Propagation (LBP)	Structured Prediction	Personalized recs, image de-noising
	Label Propagation	Structured Prediction	Personalized recommendations
	Alternating Least Squares (ALS)	Collaborative Filtering	Recommenders
	Conjugate Gradient Descent (CGD)	Collaborative Filtering	Recommenders
	Connected Components	Graph Analytics	Network manipulation, image analysis
	Latent Dirichlet Allocation (LDA)	Topic Modeling	Document Clustering
	Structure Attribute	Clustering	Network analysis, consumer seg
	K-Truss	Clustering	Social network analysis
	KNN*	Clustering	Recommenders
	Logistic Regression*	Classification	Fraud detection
	Random Forest*	Classification	Fraud detection, consumer seg
	Generalized Linear Model (Binomial, Poisson)	Non-linear Curve Fitting	Forecasting, pricing, market mix models
	Association Rule Mining	Data Mining	Market basket analysis, recommenders
	Frequent Pattern Mining*	Data Mining	Pattern Recognition



ibm.com/developerworks/library/os-giraph/



In the rhesus monkey, the posterior bank of the superior temporal sulcus for
prestriate visual cortex and has two regions, a medial one and a lateral one, w
separate callosal connections. The afferent input to these two regions was stud
where the corpus callosum was sectioned, and labelled amino acids were inje
areas. By this method, it was found that area 17 projects to that part of the su
occupied by the more medial of the two callosal inputs. By contrast, the part
occupied by the more lateral callosal input was found to receive a strong proj
visual complex, an area rich in colour-coded cells. Recordings were made fro
superior temporal sulcus in animals in which the corpus callosum had been s
The degeneration produced by this procedure was used to provide anatomica
us to assign cells to the lateral or the medial regions of the sulcus. Such recor
receptive fields were topographically organized in the lateral part of the sulcu
were colour specific. By contrast, cells recorded from in the region of the mo
patch within this sulcus were directionally selective, without any obvious col
concluded from these combined anatomico-physiological experiments that th
distinct regions in the superior temporal sulcus which have different afferent
functional properties. Both anterograde and retrograde transport tracing meth
study the organization of the projections of the dorsal lateral geniculate (DLC
pulvinar and subdivisions of the lateral pulvinar to primary visual cortex (str
The D L G projects only to striate cortex. These projections are retinotopical
not extend to any cortical layers above layer IVA. In contrast the inferior pul
immediately adjacent portion of the lateral pulvinar (PL~x48) project to both
cortex. The projections from these two thalamic areas to the striate cortex are
organized and exist in parallel with those from the DLG. In contrast to the D
from P1 and PLot terminate above layer IVA in striate cortex, i.e. layers I, 1I
cortex the layers of termination include layers IV, III and I. The pulvinar ter
and III of area 17 occur in segregated patches as do the geniculate terminatio
IVA. On the other hand the pulvinar terminations in layer I which overlie th
1II of area 17 appeared to be continuous. Control studies show that the rema
pulvinar overlying PL(~ does not project to striate cortex. It is concluded tha
visuotopically organized inputs from the lateral thalamus to primary visual co
these inputs have different layers of termination. The inputs from PI and DL
retinal inputs while those from P1 and PLcL can also be involved in intrinsi
thalamocortical connections with prestriate cortex. It remains, then, that it ca
assumed that the ascending inputs which influence the response properties o
neurons arise solely from the dorsal lateral geniculate nucleus. It is also argue
the supragranular layers may be excitatory as those from the DLG to the IVth
and retrograde transport methods were used to study the corticocortical conn
3b, 1, 2, 5, 4 and 6 of the monkey cerebral cortex. Fields were identified by
features and by thalamic connectivity in the same brains. Area 3a was identif
short latency group I afferent evoked potential. Attempts were made to analy
(1) routes whereby somatic sensory input might influence the performance of
neurons; (2) possible multiple representations of the body surface in the com
first somatic sensory area (SI). Apart from vertical interlaminar connections, t
intracortical connectivity are recognized. The first, regarded as "non-specific