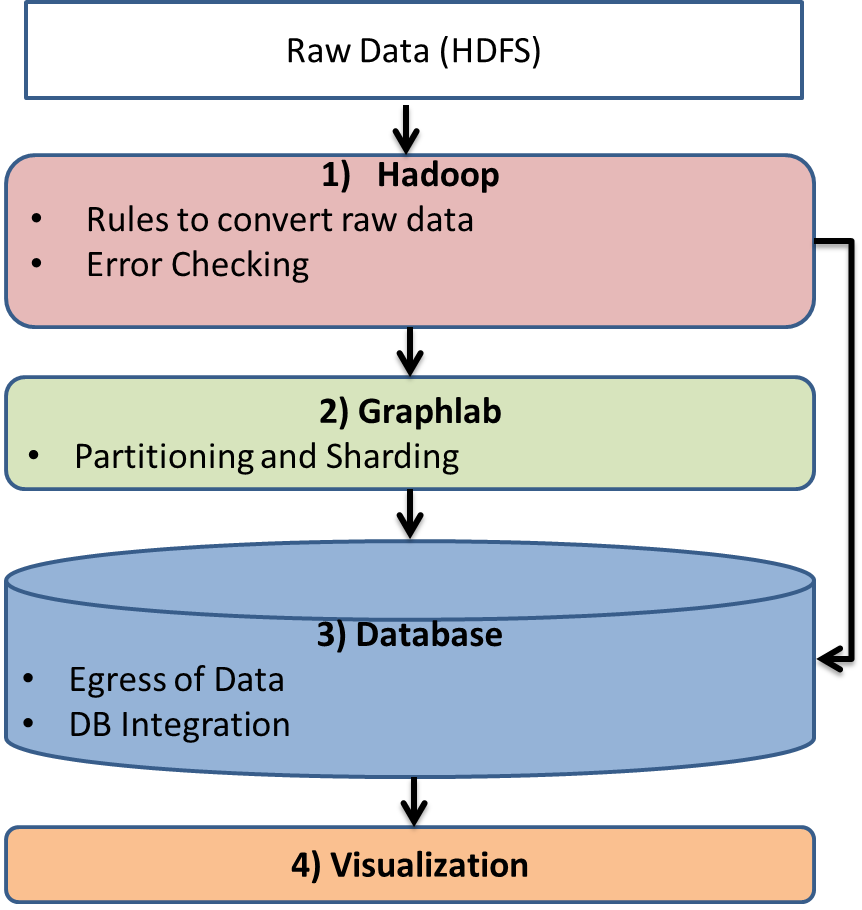
Graph builder Project   
for Graphlab Workshop 2012

This is a preliminary sketch on the data feed and pre-processing step for Graphlab. It is likely that most of the work will be concentrating on a good abstraction to generalize the rules of converting raw data to the graph format.

# Features to implement



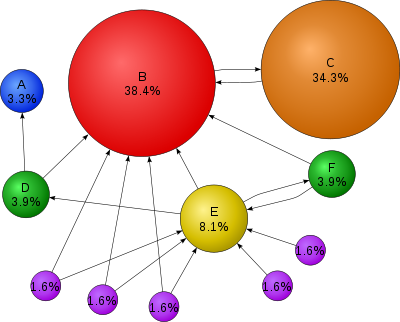
# Rules to convert raw data

Inspired from graph theory, three different types of graphs have been identified. They translate to three different policies for converting raw data. Each has general characteristics that allow defining API functions to cover most parsing raw data problems into Graphlab.

**1) Homogeneous graphs:** Derived from problems that categorize all nodes to be of the same type with vertices that are clearly defined by explicit relationships. Some of the algorithms in this category are under graph mining for community detection, ranking,

*Hadoop:* Preparation of data could be carried out in a single iteration of map reduce. There could be two functions: 1) Define **nodes types** in map step 2) Define the **relationships** in reduce step. These two functions could be overloaded to be defined by the programmer. The interface creates the graph output.

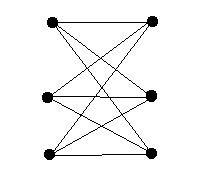
*Example***:** social networks fall under this category. Each node is only one type (i.e. a person) the vertex is whether a given person knows someone or not. The types of computations that may occur could be popularity (page rank), groups of friends (clustering). In Hadoop for PageRank in Wikipedia, the *node type function* would be extracting all link id, and the *relationship function* would be extracting all links from a given link id.



**2) Bipartite Graphs:** One to one mapping that can be obtained from a Matrix with two well defined groups represented by distinctive column and row. These are graphs with two sets of vertices, where the ones from group one strictly has edges to the ones in group two and vice versa. In other words, it is a disjoint set of data.

*Hadoop:* two scenarios A) data could already be in matrix format, so a simple conversion will be sufficient. B) Otherwise, similar to the case in homogenous case, there are two functions: 1) Define **node set in group one** and **node set in group two** 2) Define **relationship** between the members to each opposing group

*Example:* Applications in matrix factorization such as user to product recommendation in ALS. Another example is LDA for topic categorization with group one of document id and group two of word id. In Hadoop for LDA in Wikipedia, the map step would define the *node set function* for doc id and word id and carry out a word count, and the *relationship function* in the reduce step would emit a frequency count linking documents to words.



**3) Heterogeneous Graphs**: Based on hyper-graphs structure could depend on graphical model. (Need more definition…)

*Example:* Tensors, or grid model in video segmentation

# Partition and Sharding

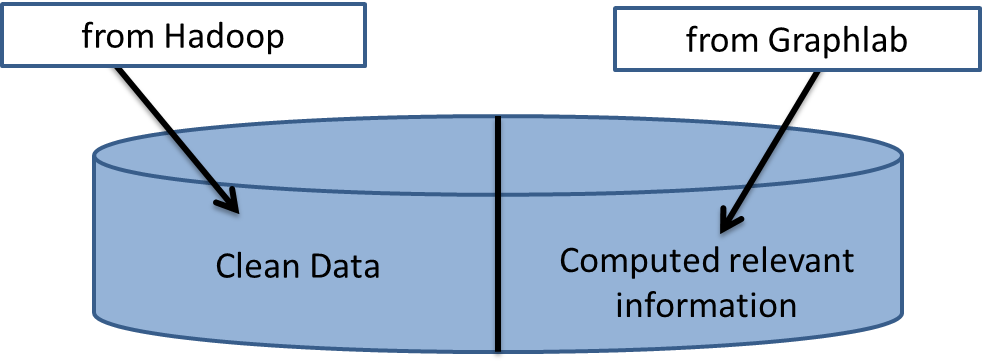
Work already carried out internally in Graphlab that handles the load balancing of partitioning a graph among N machines as optimal as possible.

# Error Checking

* Check presence of duplicate edges and vertices: Possible implementation uses a hash table for each edges and vertices as one does not imply the other
* Check data structure or types: differentiation between types of edges according to structure of graph while maintaining consistency in data types as C++ is strongly typed

# Egress of Data

To enable query and search, store “clean up data” after initial passes from Hadoop as well as store back computed results from Graphlab that provide new relevant information such as predictions, labels, ranks, inferences or probability distributions.



*Example:* On Wikipedia computation for pagerank and LDA

* ***clean data*** would contain an entry per article, the corpus of text or file location
* ***computed relevant information*:** would contain ranking from page rank and topic categorization from LDA

The definition in our API needs functions to extract computed results from the binary graph file out of Graphlab to obtain the necessary information from the vertices or edges. For example in PageRank, the ranks need to be extracted from the vertex data that has the weight. In LDA, the topics are extracted from the document id maximum probability from the distribution of topics.

The interface would read the binary output from Graphlab with template functions to extract the relevant fields that would follow the data structure defined by the programmer in Graphlab. Finally, these fields would be stored back this into a chosen database.

# Database Integration

**SQL:** can be integrated through ODBC in C (Open Database Connectivity) or JDBC (Java Database Connectivity). This would allow query of data independent of OS through the driver translation layer protocol that communicates clients to the database. In addition, JDBC has already been integrated to Hadoop. (For more information refer [here](http://www.cloudera.com/blog/2009/03/database-access-with-hadoop/))

**NoSQL:** Most have support and the API that already make use of Hadoop. Example [here](http://api.mongodb.org/hadoop/MongoDB%2BHadoop+Connector.html) from mongoDB

**GraphDB:** From Nilesh’s slide

# Visualization

For demo purposes, create a visual to showcase work.

# Reference

* <http://en.wikipedia.org/wiki/PageRank>
* <http://www.cs.ucsb.edu/~xyan/classes/CS595D.htm>
* <http://www.orafaq.com/wiki/JDBC>
* <http://www.cloudera.com/blog/2009/03/database-access-with-hadoop/>
* <http://api.mongodb.org/hadoop/MongoDB%2BHadoop+Connector.html>