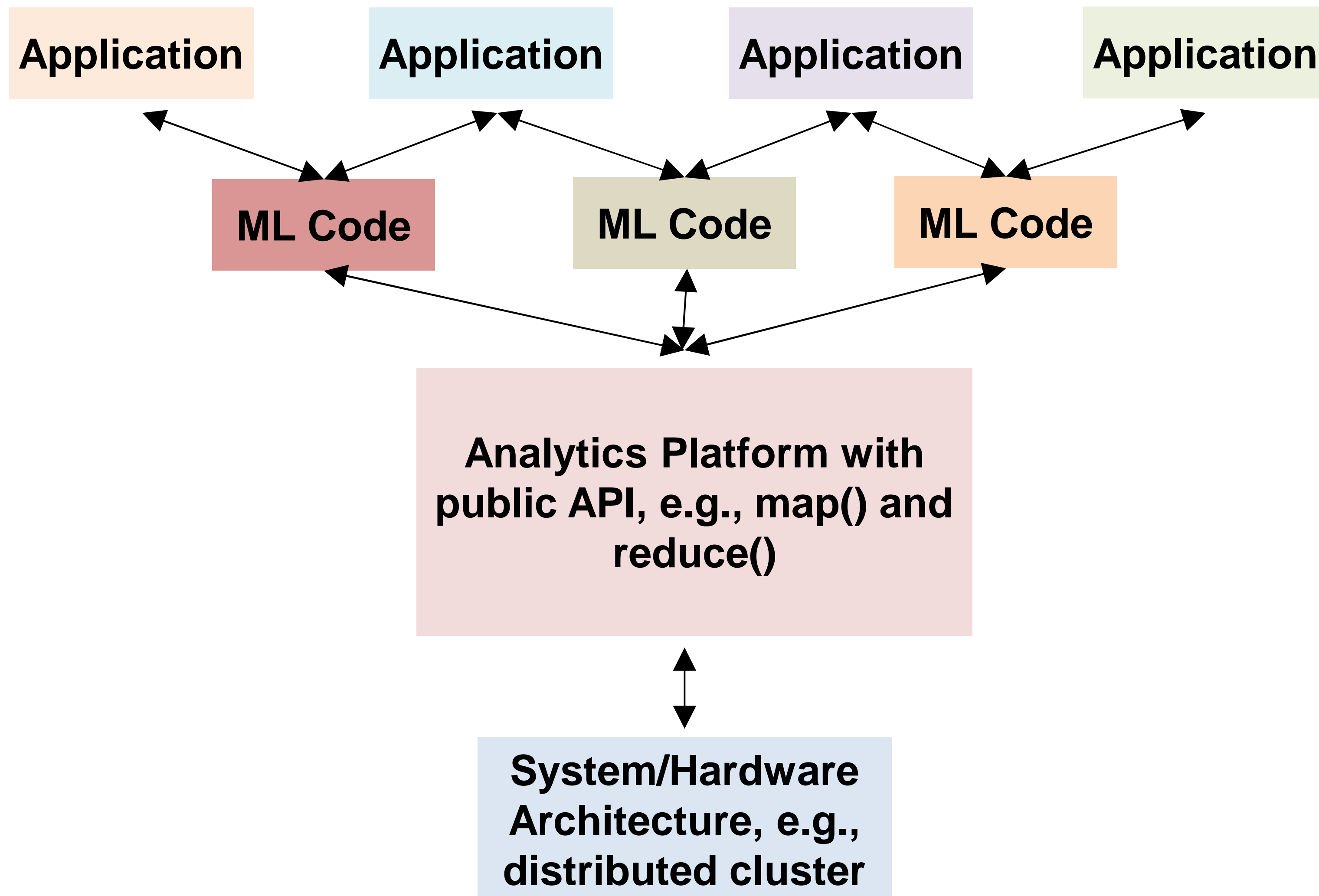
Very Large Scale Bayesian ML Systems

Zhuhua Cai
Google, Rice University
caizhua@gmail.com

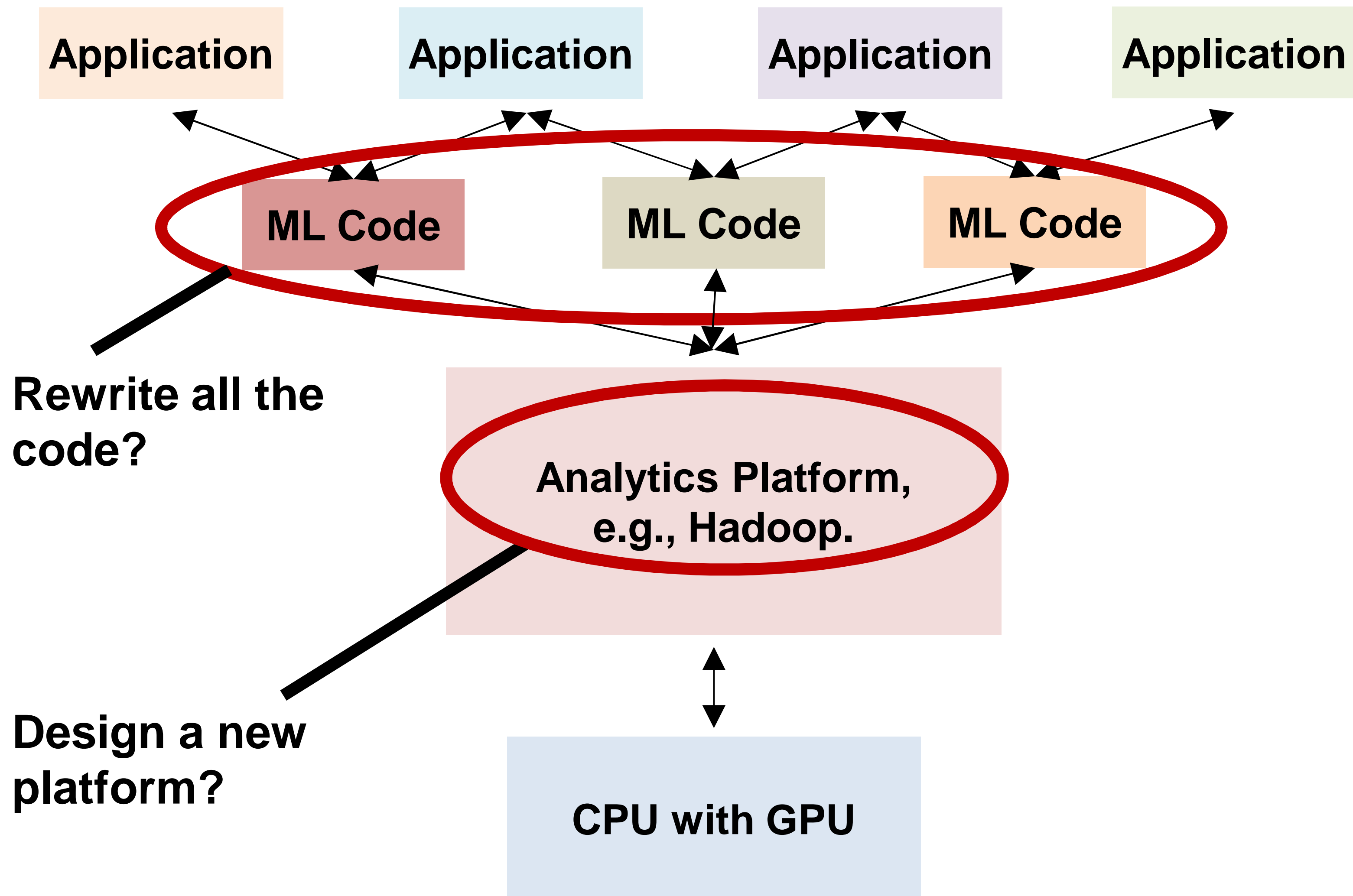
Outline

- MapReduce extensions
- Gaussian Mixture Model
- Parallel DB
- Graph ML
- Benchmark

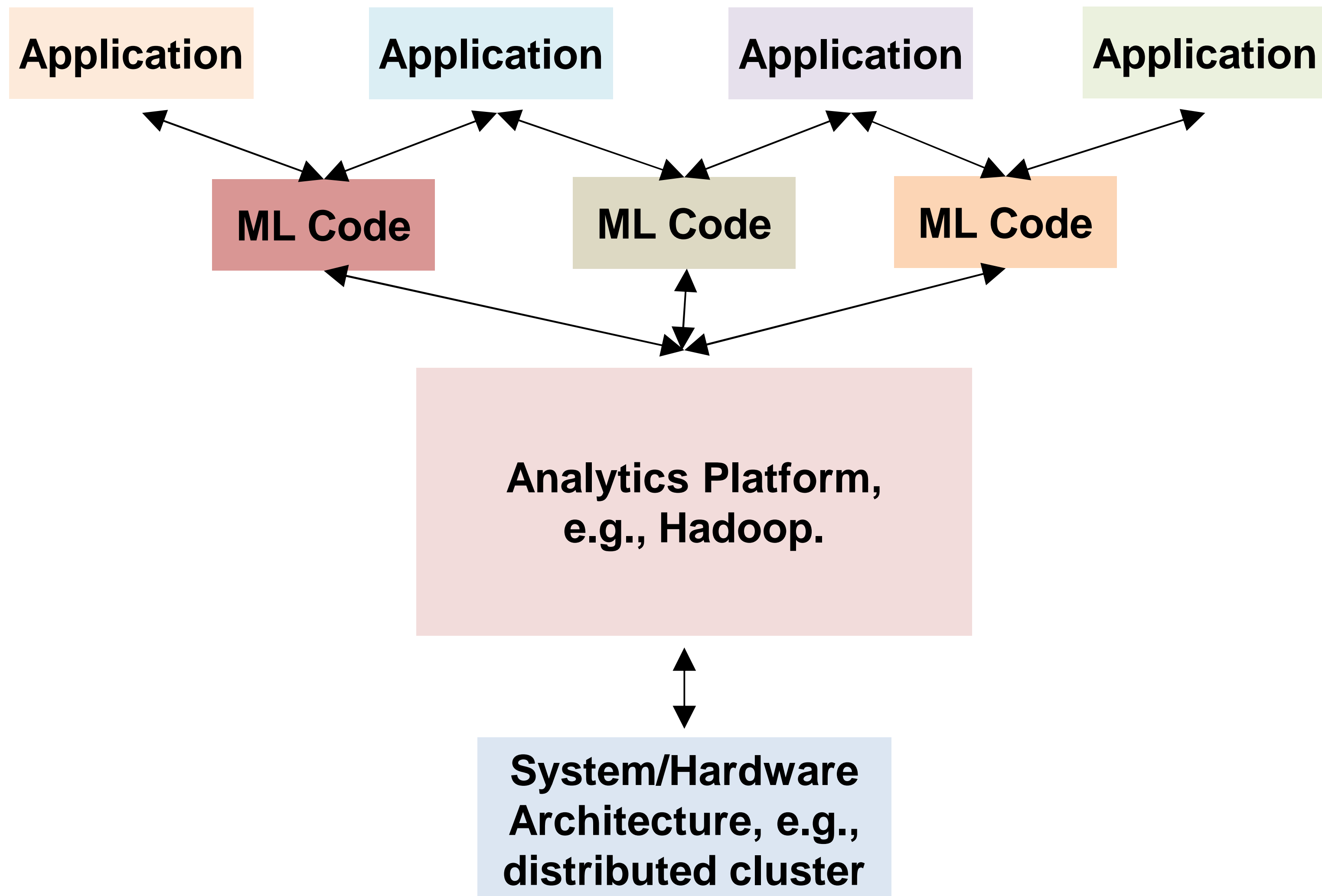
Parallel ML in A Higher Level



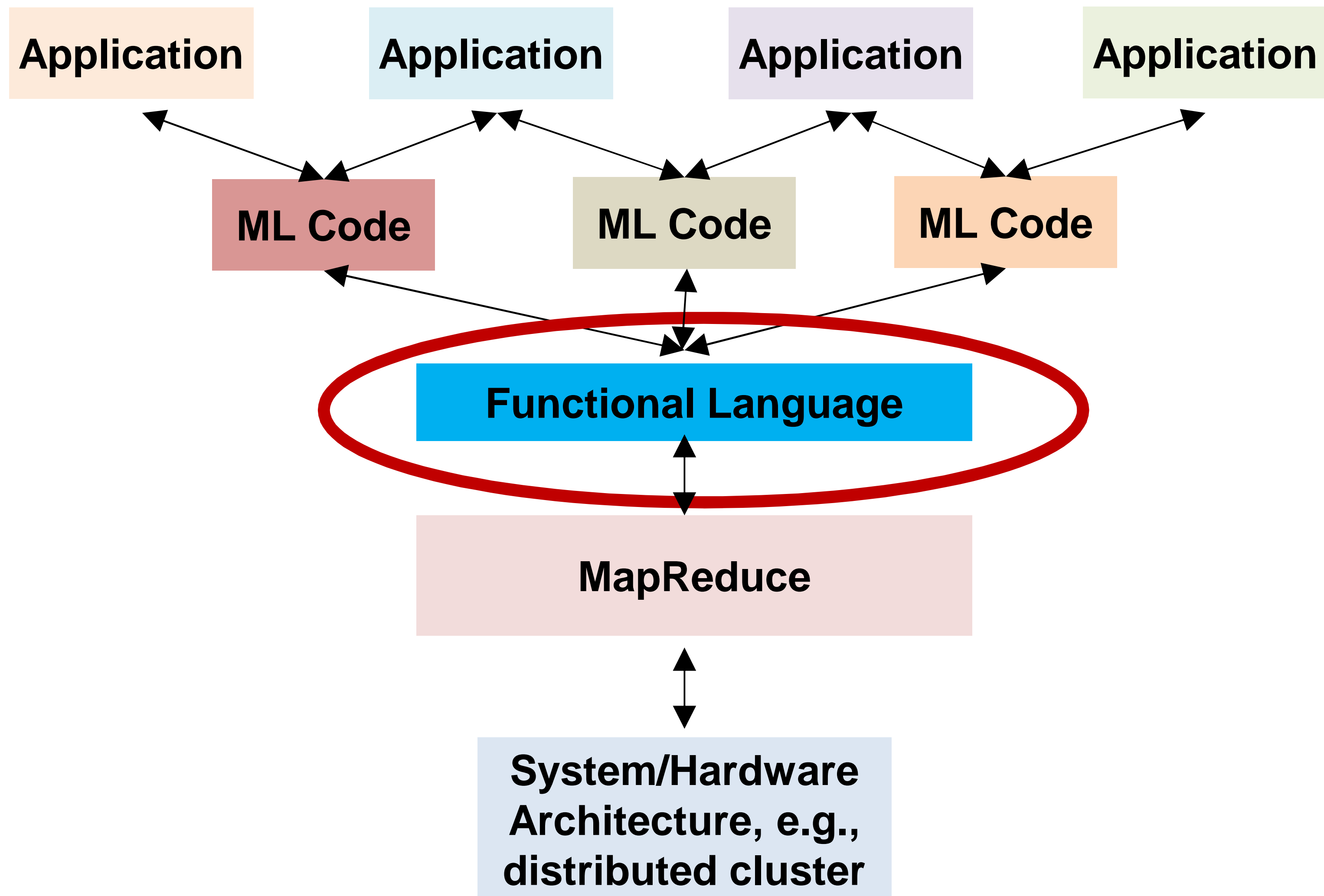
Data Independence



Repeated Work in ML code

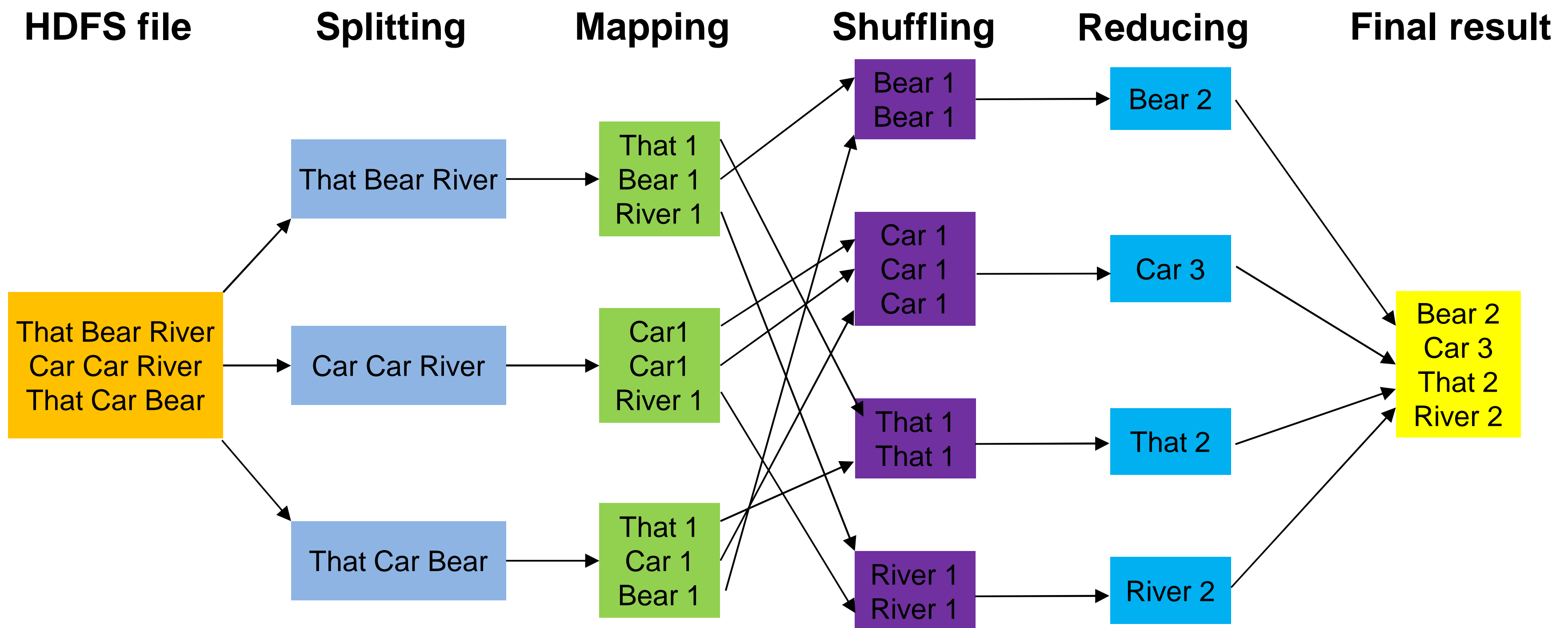


Flume/Spark



Problem 5

- Given 20news-group dataset, for each word, compute its count in the whole dataset.



PySpark for WordCount

```
sc = SparkContext(appName="WordCount")
lines = sc.textFile(sys.argv[1], 1)
wordCount= lines.flatMap(lambda x: split(x, “"\":;, (){}\t\r\n,|”)).
        map(lambda x: (x,1)).reduceByKey(add)
output = wordCount.collect()
```

JavaSpark for WordCount

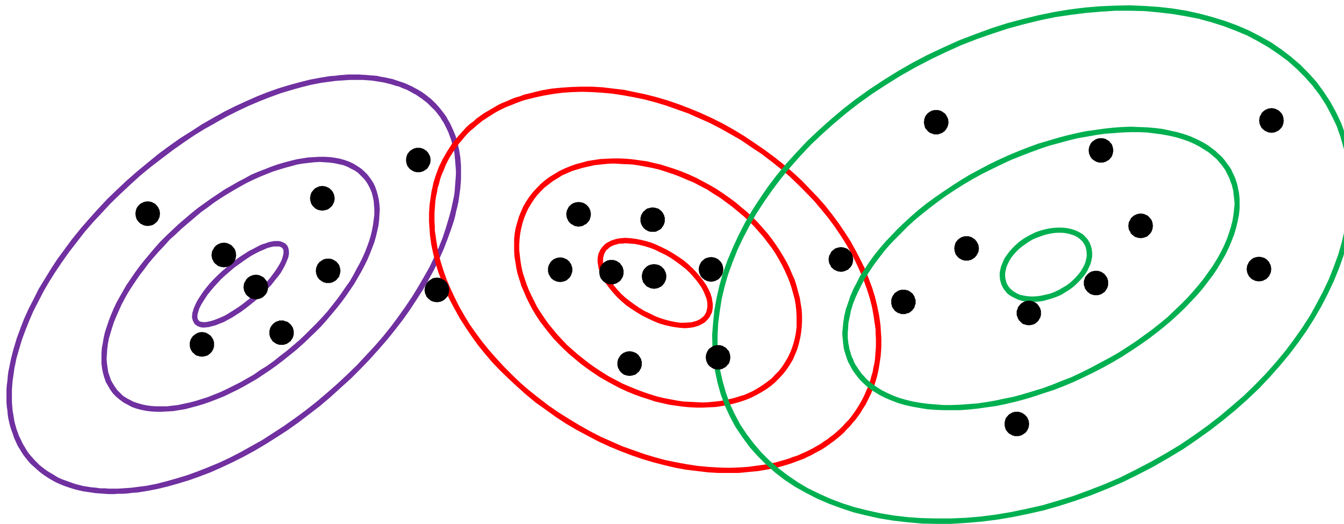
```
JavaSparkContext sc = new JavaSparkContext(conf);
JavaRDD<String> lines = sc.textFile(args[1]);
JavaRDD<String, Integer> wordCount = lines.flatMap(
   new FlatMapFunction<String, String>() {
        public Iterable<String> call(String s) {
                return Arrays.asList(split(x, “"\":;, (){}\t\r\n,|”));
        }
   }).map(new PairFunction<String, String, Integer>() {
        public Tuple2<String, Integer> call(String s) {
                return new Tuple2<String, Integer>(s, 1);
        }
   }).reduceByKey(new Function2<Integer, Integer, Integer>() {
        public Integer call(Integer a, Integer b) {return a + b;   }
   });
Map<Integer, Vector> tempMap = cluster_model.collectAsMap();
```

Outline

- MapReduce extensions
- Gaussian Mixture Model
- Parallel DB
- Graph ML
- Benchmark

Gaussian Mixture Model

- Infer this:



Generative Model for GMM

1. Generate the centroids of clusters (μ_j, Σ_j) from a multi-Gaussian and inverse-wishart distribution.
2. Generate the fraction vector π, i.e., the fraction of data points in each cluster.
3. Generate the membership of each data point and data values.

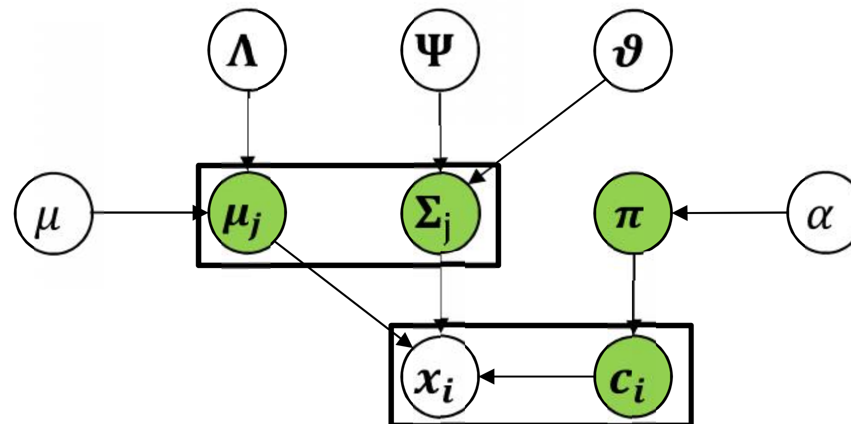
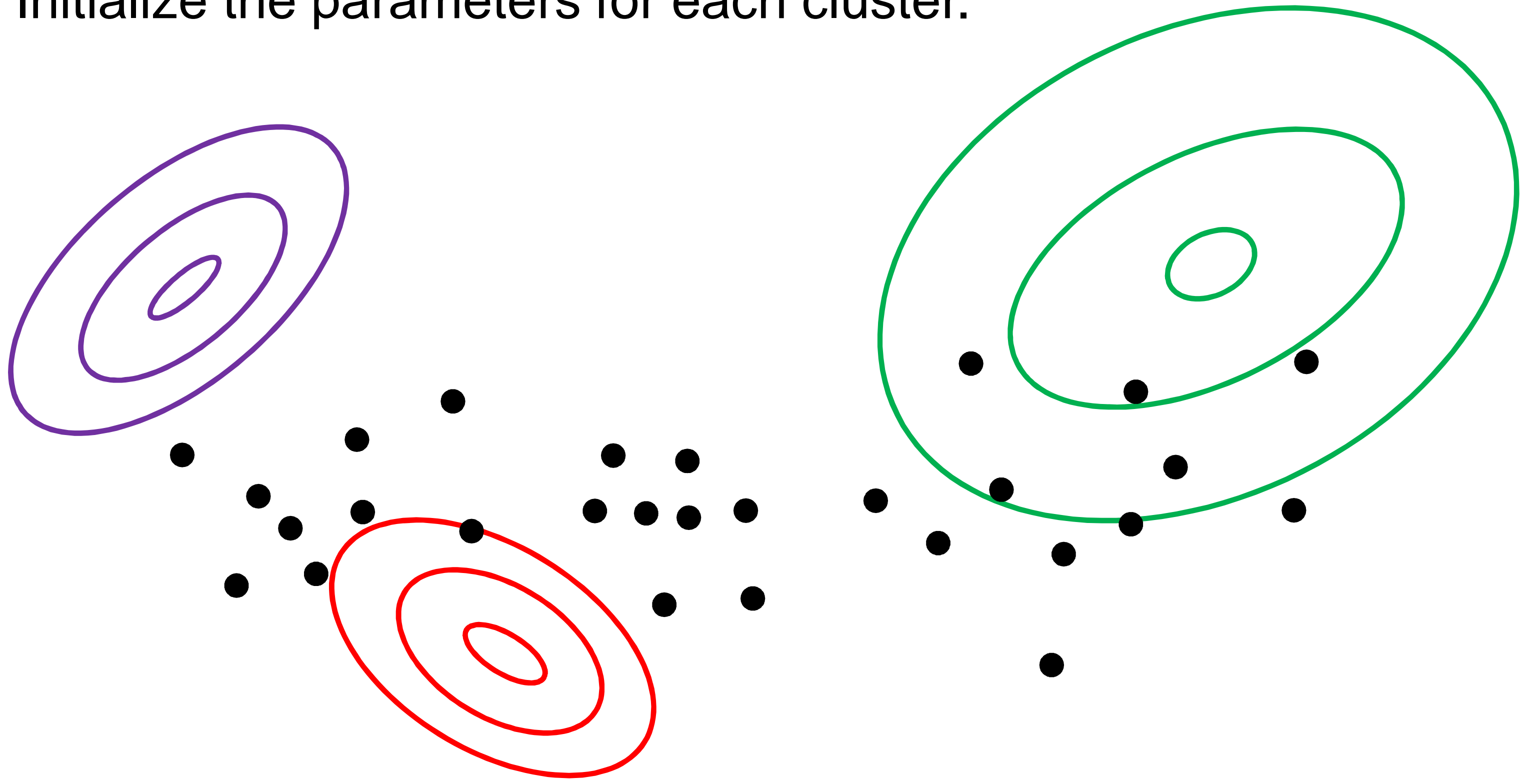


Figure. Graphical model for Gaussian Mixture Model

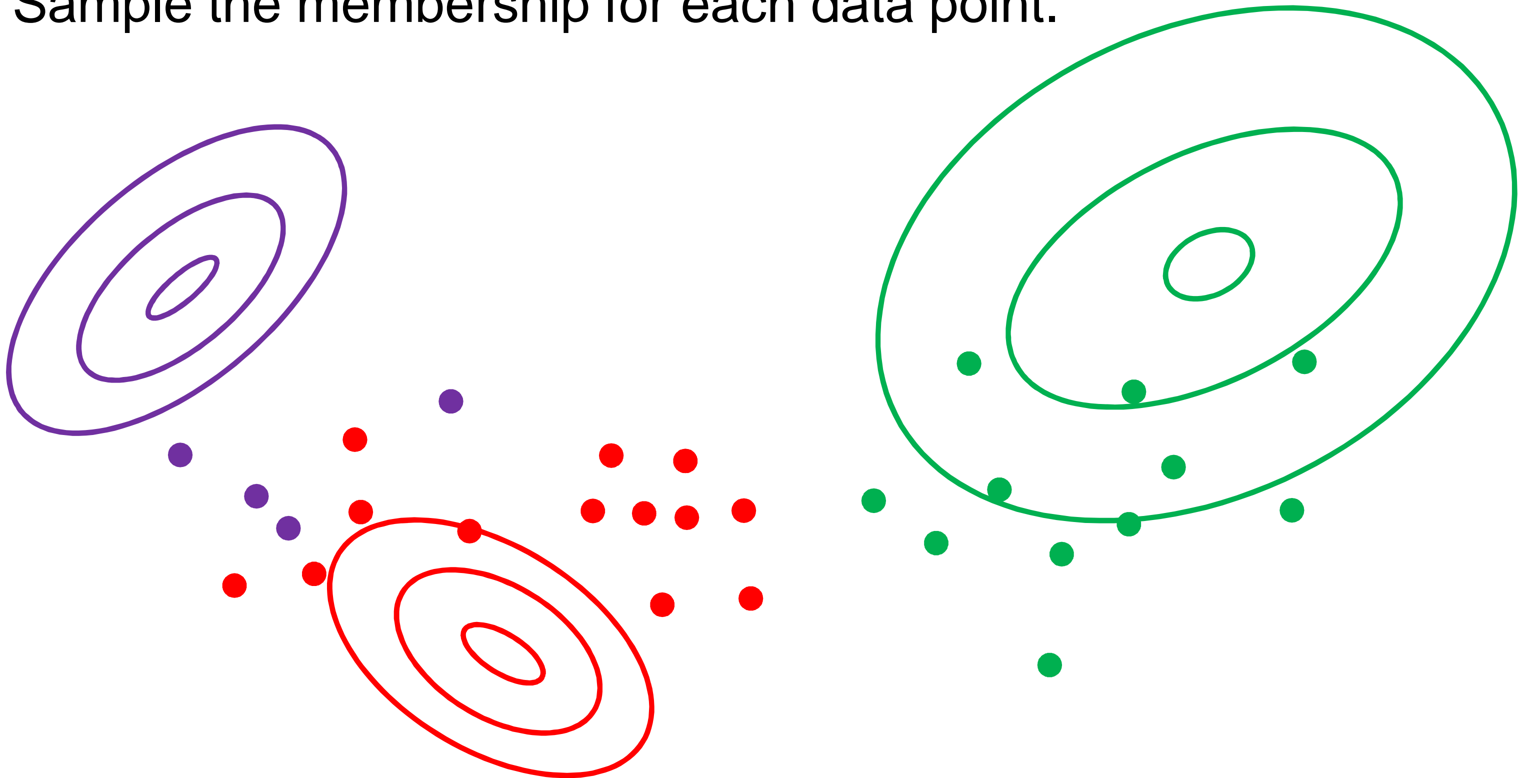
Learning steps

1. Initialize the parameters for each cluster.



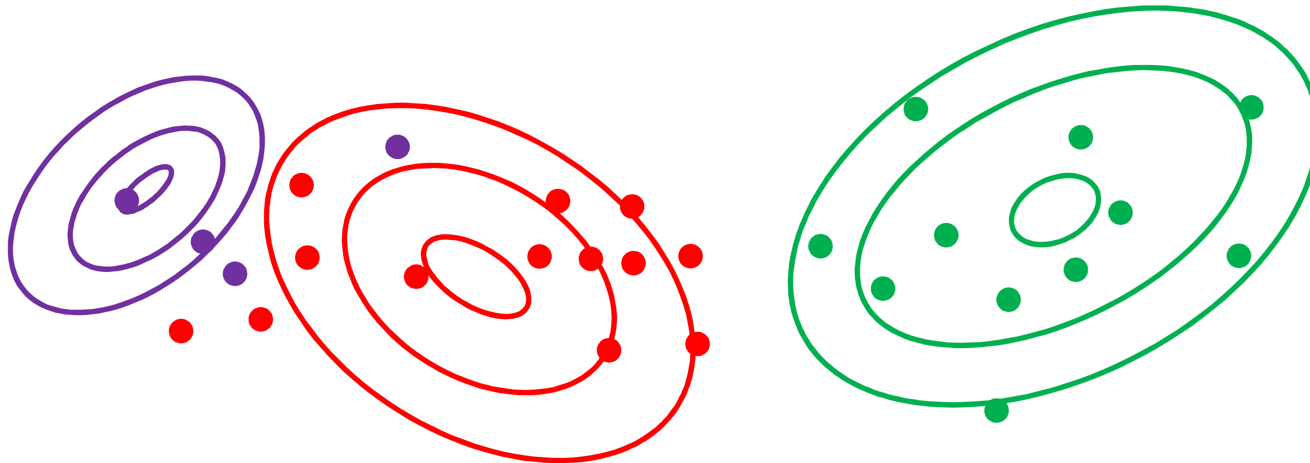
Learning steps

2 Sample the membership for each data point.



Learning steps

③ Update the parameters for each clusters.



MCMC Algorithm

$$\mu_j \sim N\left(\left(\Lambda + n_j \Sigma_j^{-1}\right)^{-1} \left(\Lambda\mu + \Sigma_j^{-1} \sum_{c_i = j} x_i \right), \left(\Lambda + n_j \Sigma_j^{-1}\right)^{-1} \right)$$

$$\Sigma_j \sim InvWish\left(\Psi + \sum_{c_i = j} (x_i - \mu_j)(x_j - \mu_j)^T, n_j + v \right)$$

$$\pi \sim Dirichlet(\alpha + [n_0, n_1, \ldots, n_{m-1}])$$

$$c_i \sim DiscreteChoice([p_0, p_1, \ldots, p_{m-1}]), p_j \propto \pi_j \times N(x_j | \mu_j, \Sigma_j)$$

MapReduce Job Design

1. Initialize μ_j, Σ_j and π.
2. MapReduce job

   Mapper takes in μ_j, Σ_j and π, samples membership of each data point c_i, output $\left(c_j, (x_i - \mu_{c_i})(x_i - \mu_{c_i})^T\right)$ and (c_j, x_i). Both are key-value pairs.

   Reducer aggregates $_{c_i=j}(x_i - \mu_j)(x_j - \mu_j)^T$, and sample Σ_j and then μ_j.

3. Collect n_j, μ_j, Σ_j from reducers and sample π.
4. Go to step 2.

PySpark for GMM

```
// read data from hdfs, and create RDD data.
lines = sc.textFile("hdfs://master:54310/data.txt"
data = lines.map(parseLine).cache()
// initialization hyper-parameters
num = data.count()
hyper_mean = data.reduce(add)/num
hyper_cov_diagonal = data.map(lambda x: square(x - hyper_mean)).reduce(add)/num
numpy.fill_diagonal(hyper_cov, hyper_cov_diagonal)
```

PySpark for GMM

```
// Initial the model.
c_model = sc.parallelize(range(0, K)).map(lambda x:
  (x, (mvnrnd(hyper_mean, hyper_cov), invWishart(hyper_cov, len(hyper_mean)+2))))
  .collectAsMap()
pi = np.zeros(K, float)
pi.fill(1.0/K)
```

PySpark for GMM

```
// MCMC iterations.
// First sample membership and compute sum of X and gram matrix sum.
c_agg = data.map(lambda x: sample_mem(x, pi, c_model))
                .reduceByKey(lambda (x1, y1, z1), (x2, y2, z2):(x1+x2, y1+y2, z1+z2))
// Update model.
c_model = c_agg.mapValues(lambda (c_num, x_sum, sq_sum):
      updateModel(c_num, x_sum, sq_sum, len(hyper_mean)+2, hyper_mean, hyper_cov))
     .collectAsMap()
// update pi.
c_num = c_agg.mapValues(lambda (c_num, x_sum, sq_sum): c_num).collectAsMap()
pi = sample_dirichlet(c_num)
```

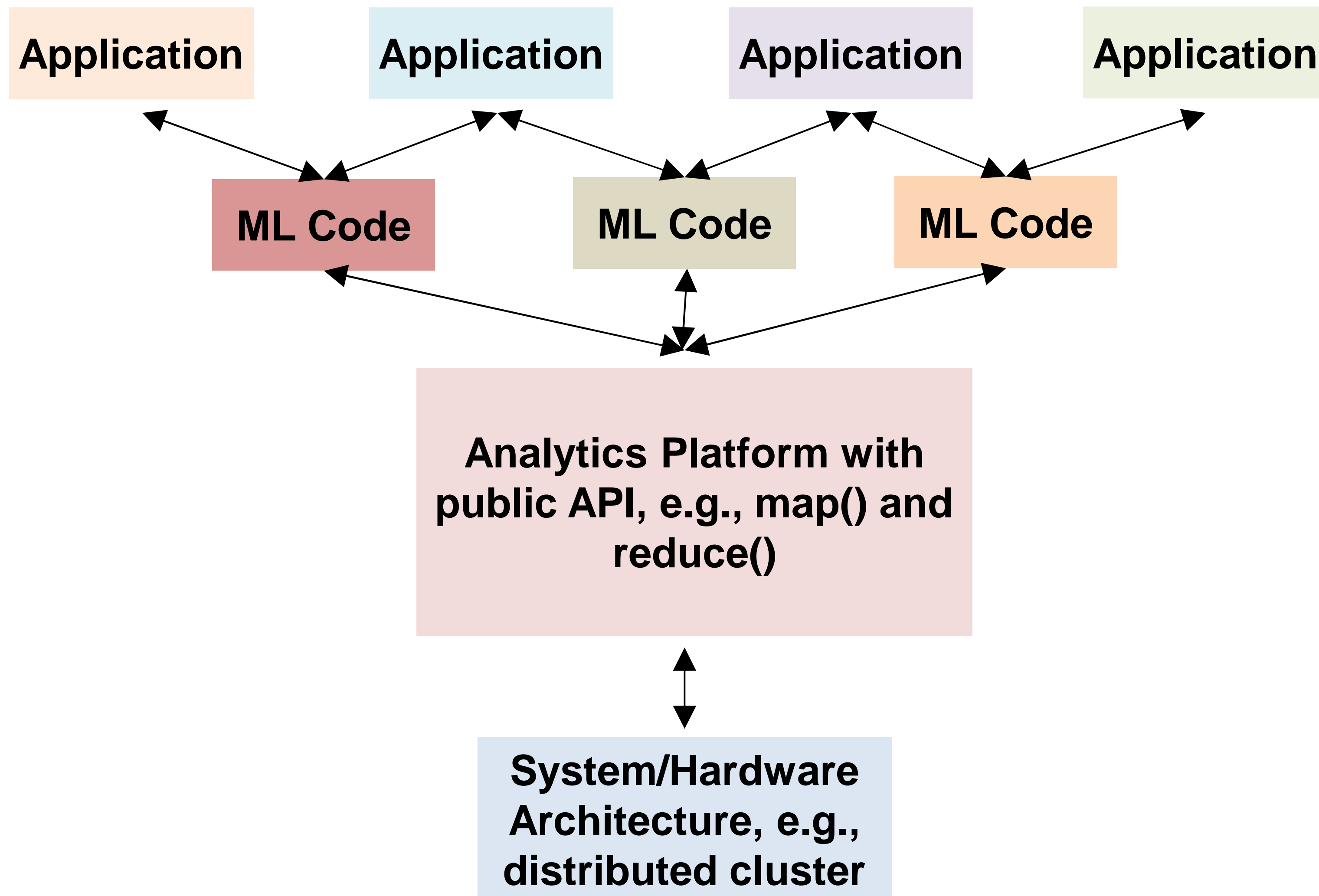
Code

- PySpark GMM.py
- Java version Gmm.java

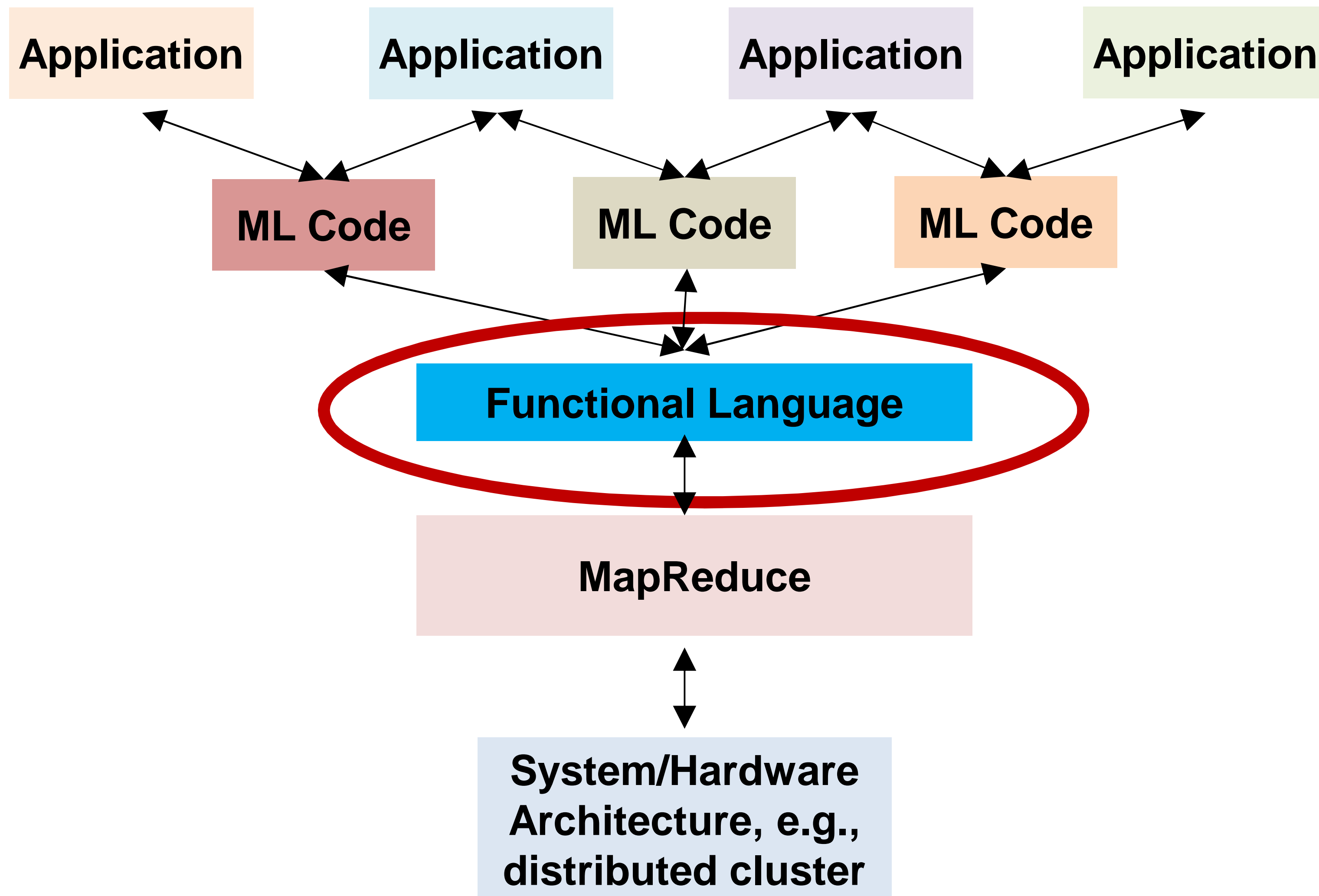
Outline

- Gaussian Mixture Model
- MapReduce extensions
- Parallel DB
- Graph ML
- Benchmark

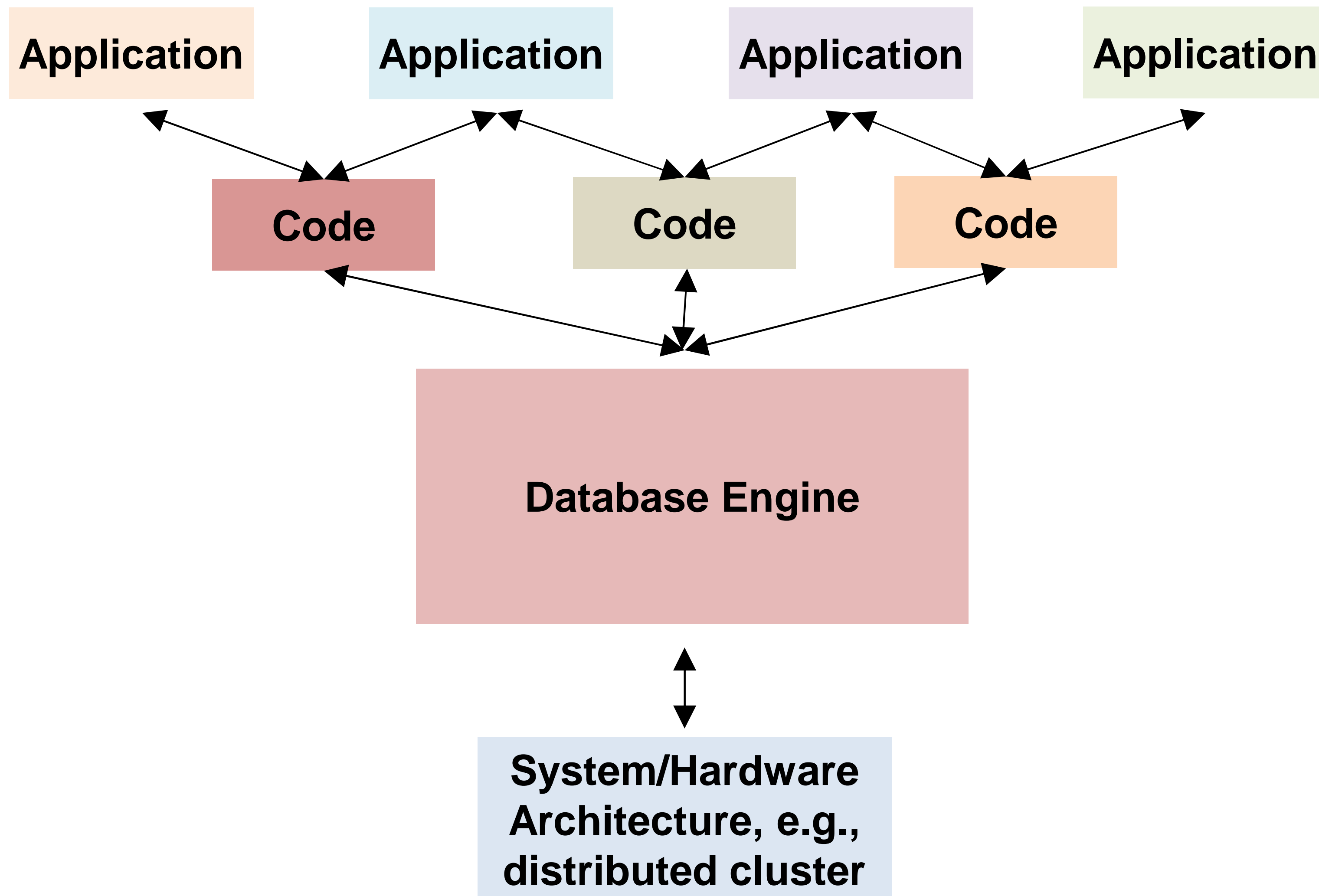
Parallel ML in A Higher Level



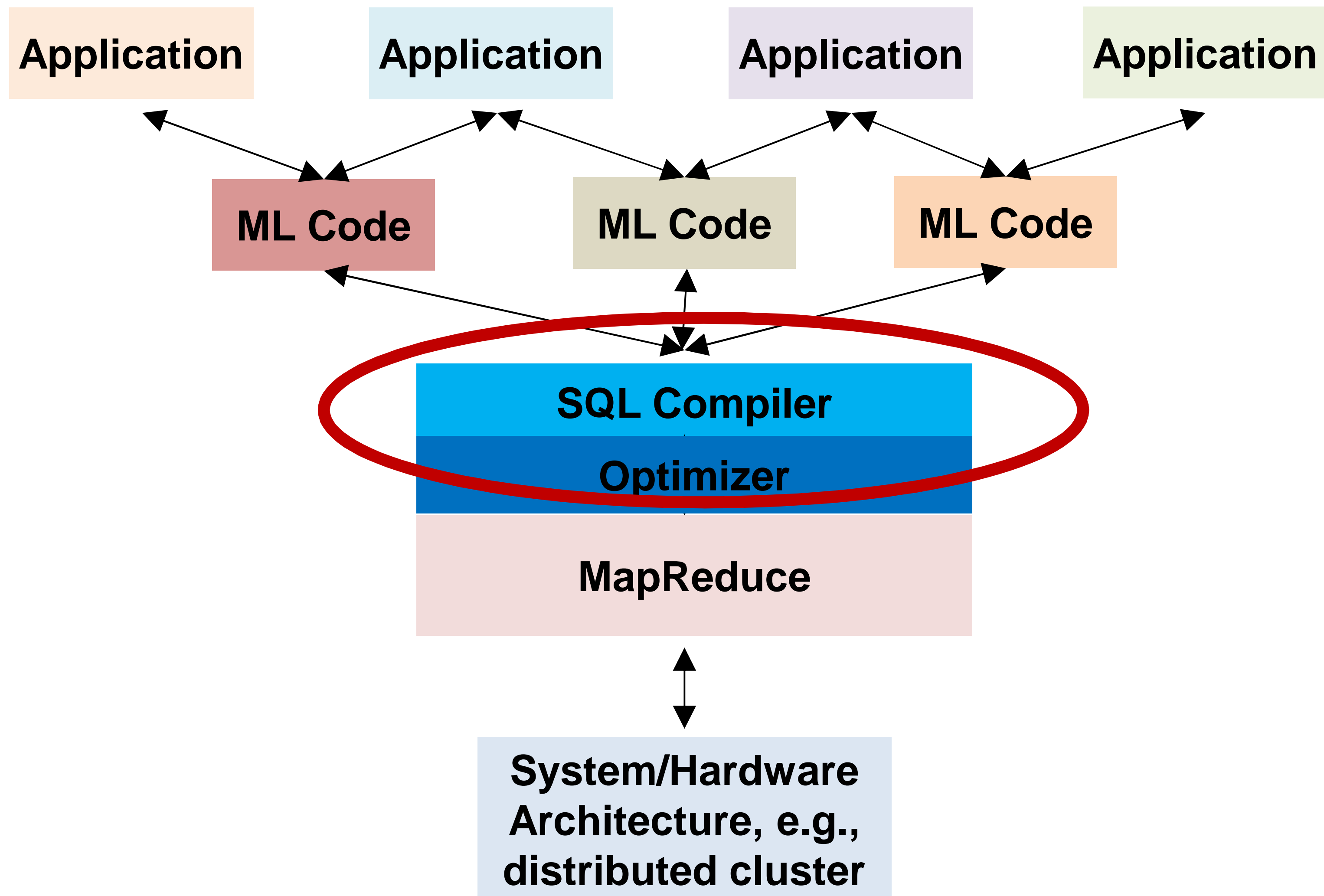
Flume/Spark



Database



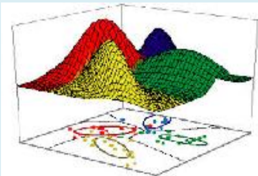
SimSQL



SimSQL Applications

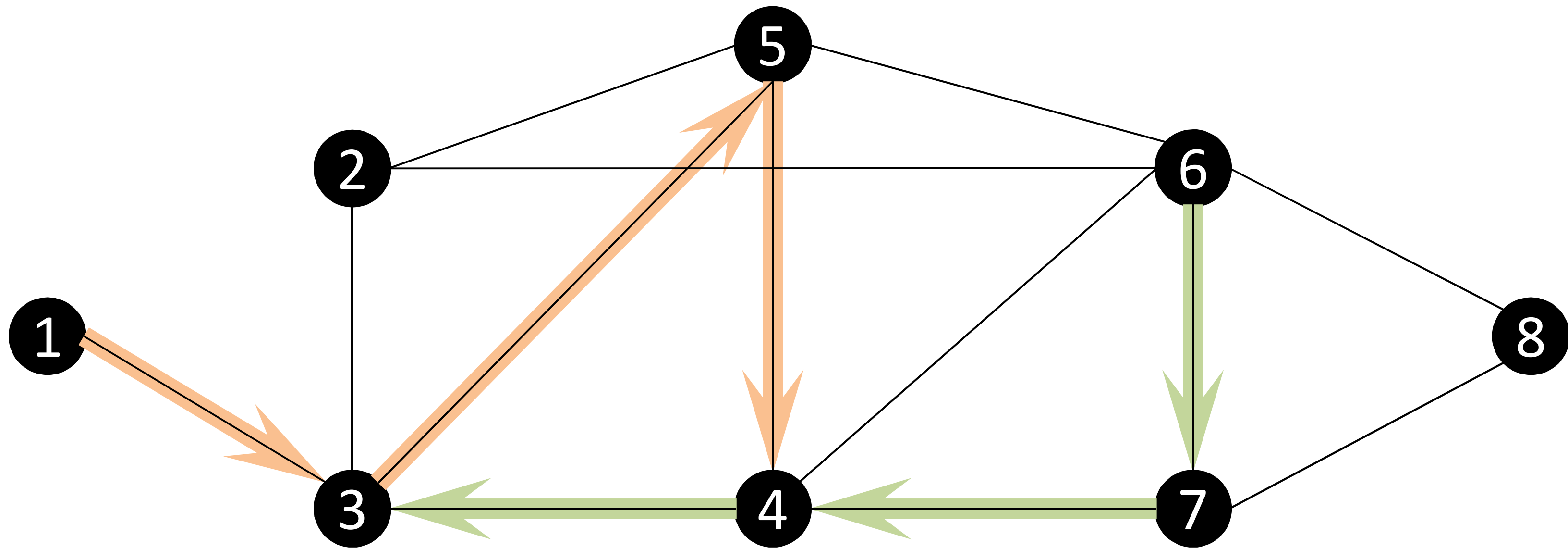
Traditional Database:
- DDL: table/view/UDF
- DML: select-query

Stochastic query:
- Stochastic table, vg function
- E.g., what-if query



Complex analytics:
- Iterative algorithm, i.e., EM, PageRank
- Markov chain simulations
- Bayesian ML, e.g., LDA, GMM, HMM, LR

Random Walk on Graphs



A simulation example: each user starts from itself, and then walks randomly in the graph.

Initializing Position[0]

Data Schema:
Link (id1, id2)
User (id)

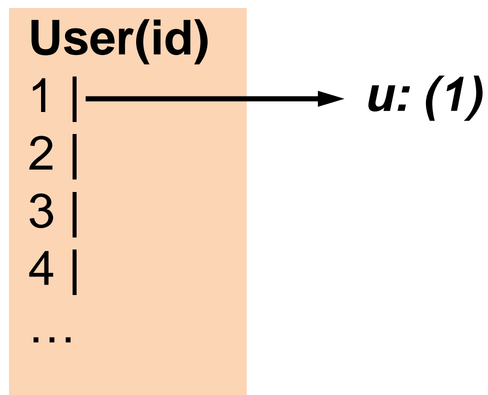
CREATE TABLE Position[0] (source, target) AS

```
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT id2 FROM Link
        WHERE id1 = u.id
    )
    SELECT u.id, next.id FROM next;
```

Initializing Position[0]

```
CREATE TABLE Position[0] (source,  target) AS
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT id2 FROM Link
        WHERE id1 = u.id
    )
    SELECT u.id, next.id FROM next;
```

loop through User (annotation pointing to "FOR EACH u IN User")



Initializing Position[0]

```
CREATE TABLE Position[0] (source,  target) AS
FOR EACH u IN User
     WITH next AS DiscreteChoice (
        SELECT id2 FROM Link
        WHERE id1 = u.id

     )
    SELECT u.id, next.id FROM next;
```

Link(id1, id2)
1 |2 |
1 |7 |
1 |9 |
2 |3 |
3 |5 |
4 |6 |
...

id2: [(2), (7), (9)]

User(id)
1 |
2 |
3 |
4 |
…

u: (1)

Initializing Position[0]

CREATE TABLE Position[0] (source, target) AS
FOR EACH u IN User
WITH next AS DiscreteChoice (
SELECT id2 FROM Link
WHERE id1 = u.id
)
SELECT u.id, next.id FROM next;

User(id)
1 |
2 |
3 |
4 |
…

Link(id1, id2)
1 |2 |
1 |7 |
1 |9 |
2 |3 |
3 |5 |
4 |6 |
...

u: (1)

id2: [(2), (7), (9)]

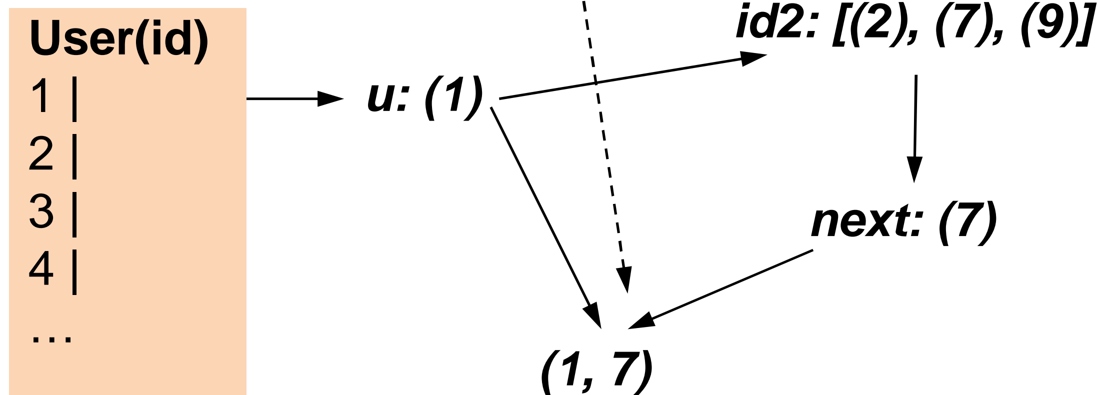
next: (7)

Initializing Position[0]

```
CREATE TABLE Position[0] (source,  target) AS
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT id2 FROM Link
        WHERE id1 = u.id
    )
```

SELECT u.id, next.id FROM next;

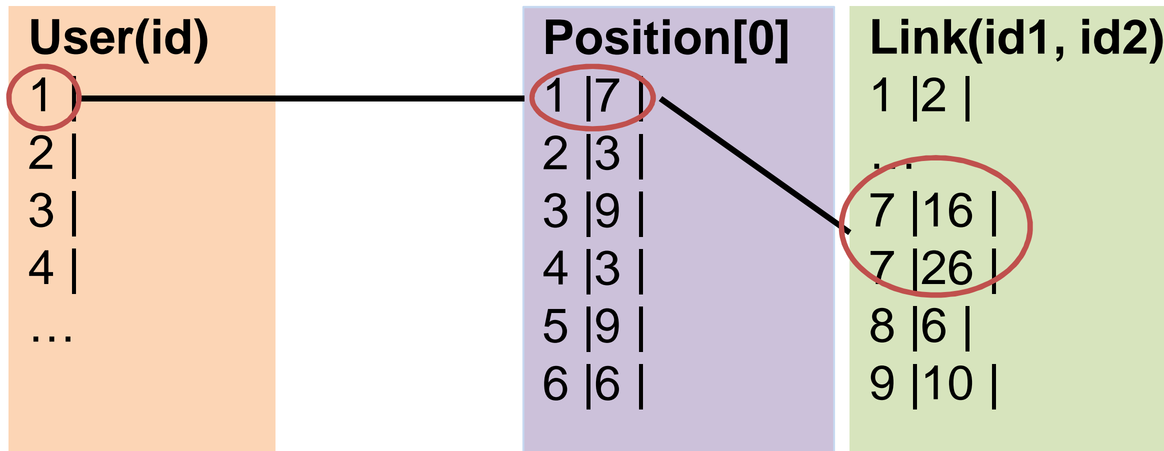
Link(id1, id2)
1 |2 |
1 |7 |
1 |9 |
2 |3 |
3 |5 |
4 |6 |
...



Updating Position[i]

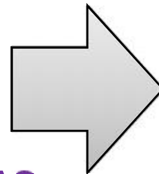
```
CREATE TABLE Position[i] (source,  target) AS
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT l.id2
        FROM Position[i-1] AS p, Link AS l
        WHERE p.source = u.id AND
                p.target = l.id1
    )
    SELECT u.id, next.id FROM next;
```

Data Schema:
Link (id1, id2)
User (id)

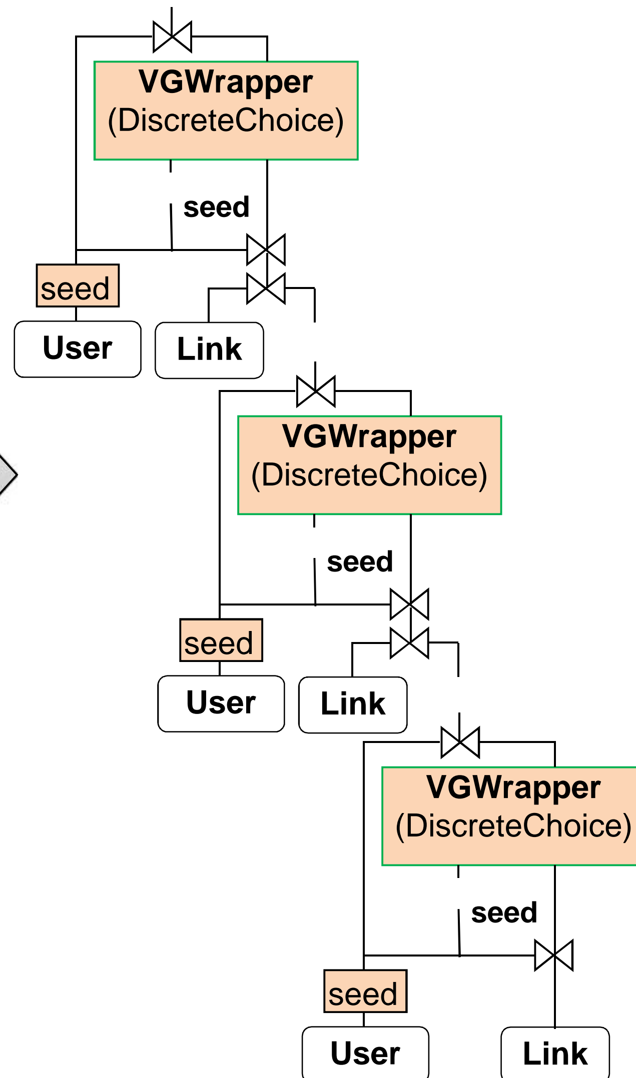


SimSQL Tasks

```
CREATE TABLE Position[0] (source,  target) AS
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT id2 FROM Link
        WHERE id1 = u.id
    )
    SELECT u.id, next.id FROM next;
```



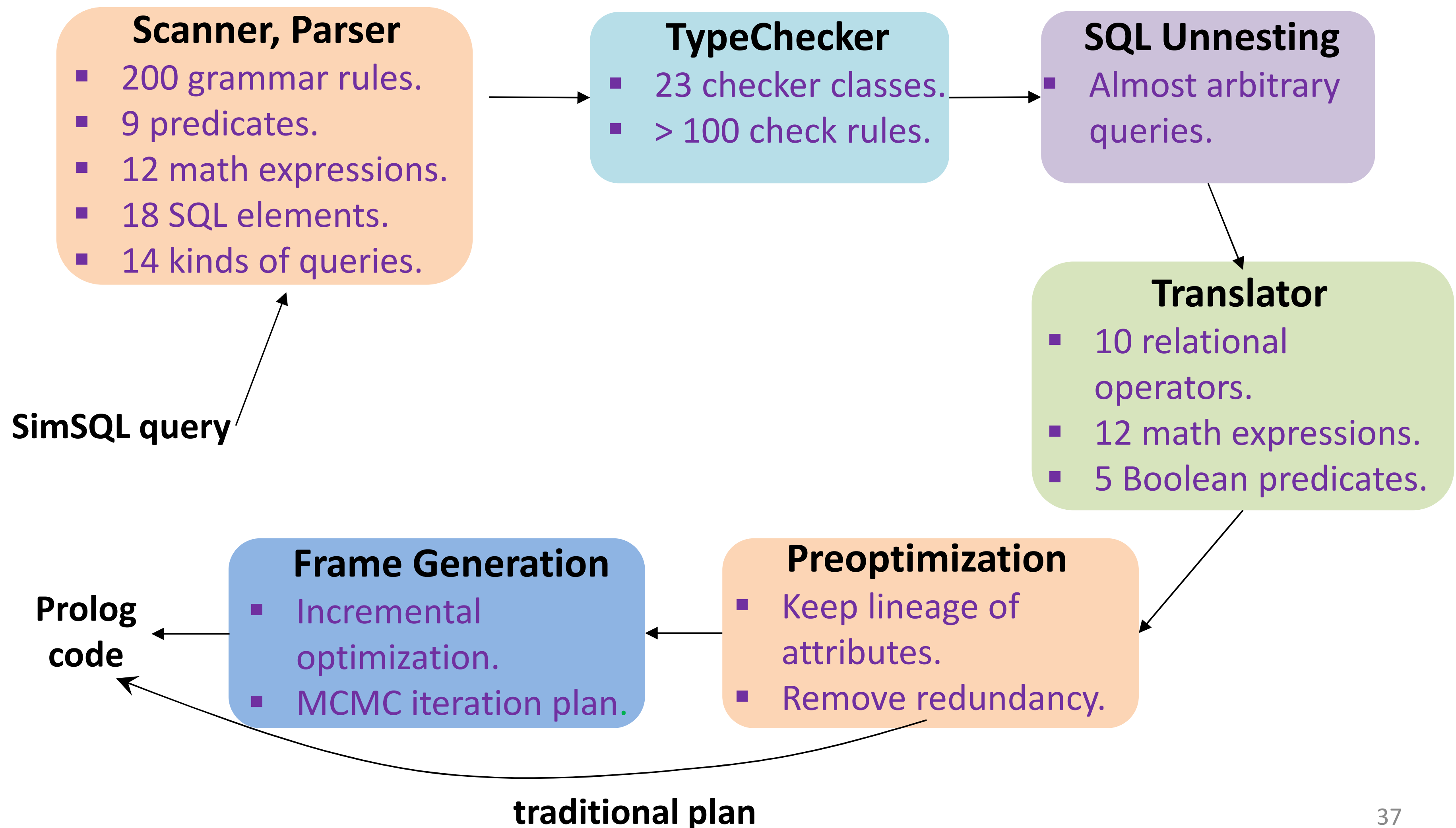
```
CREATE TABLE Position[i] (source,  target) AS
FOR EACH u IN User
    WITH next AS DiscreteChoice (
        SELECT l.id2
        FROM Position[i-1] AS p, Link AS l
        WHERE p.source = u.id AND
                    p.target = l.id1
    )
    SELECT u.id, next.id FROM next;
```



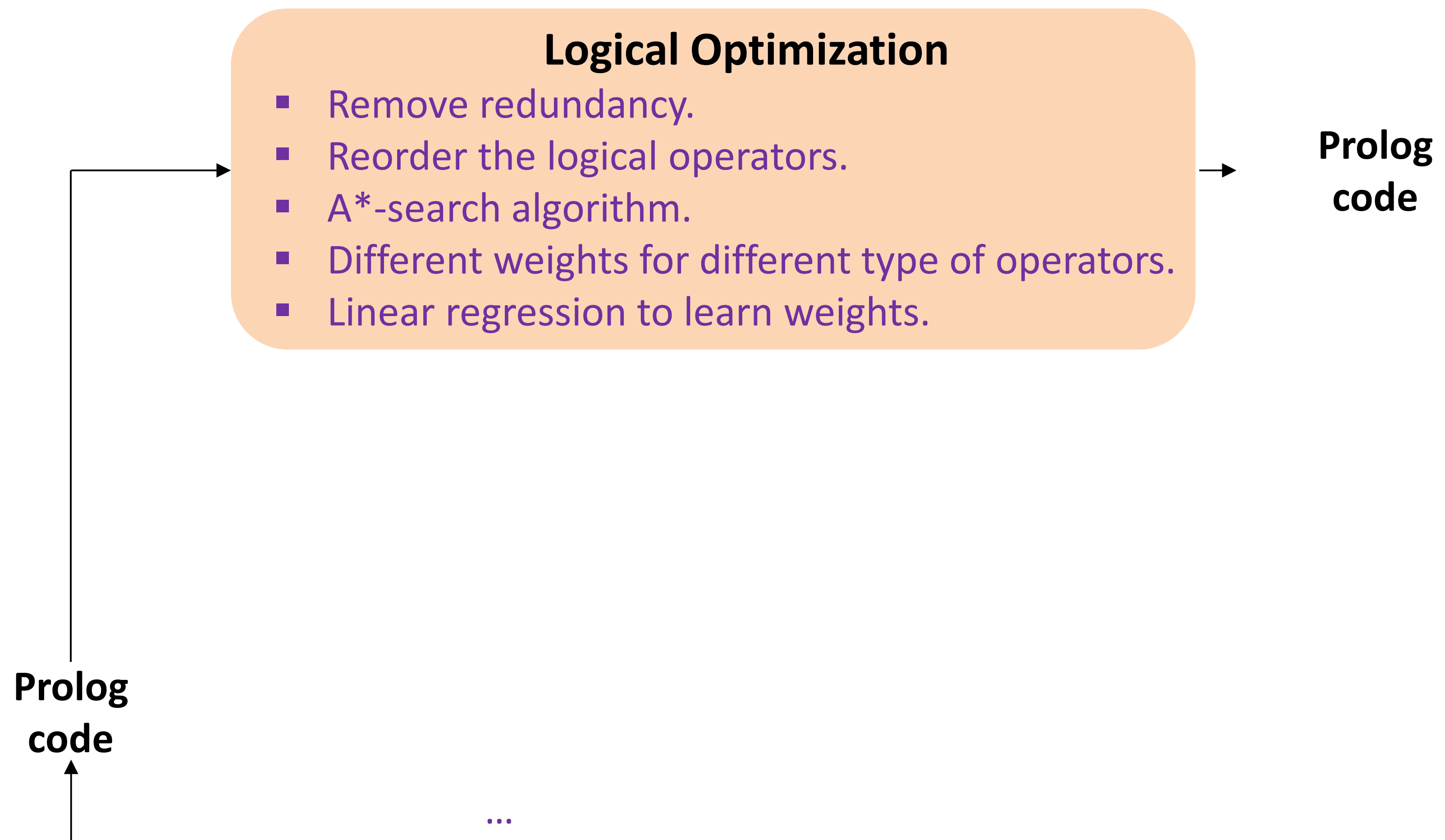
Gaussian Mixture Model

- GMM.sql

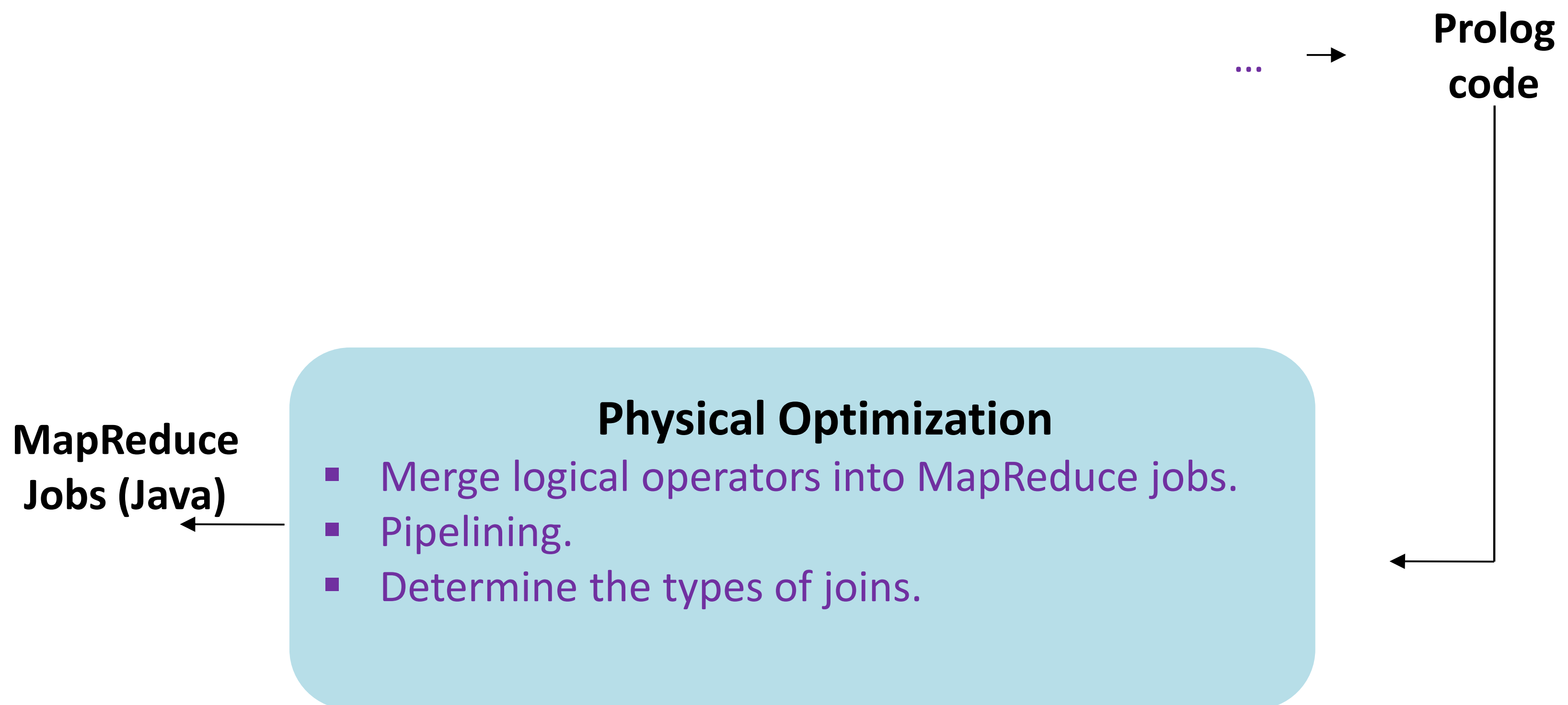
Compiler



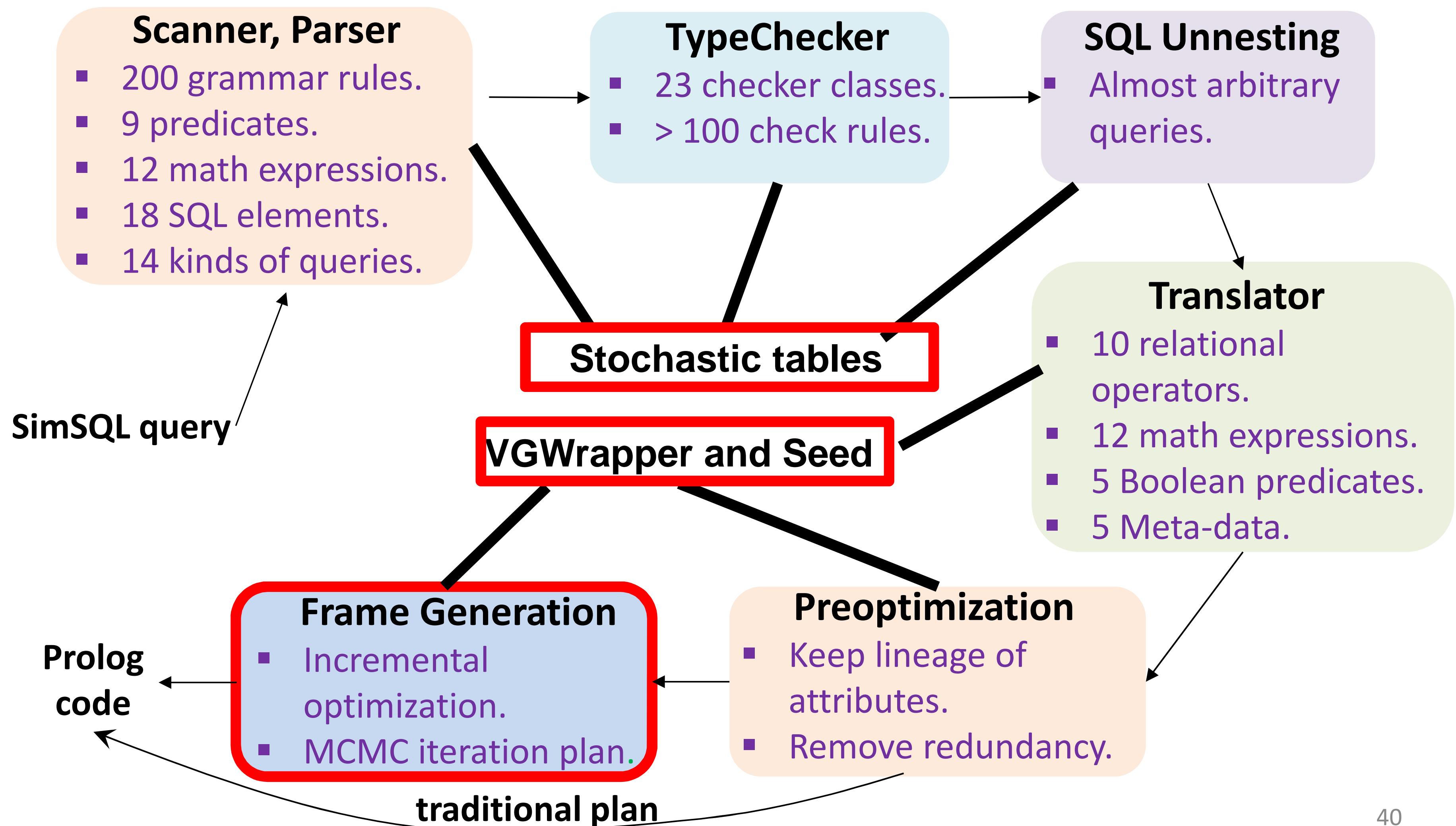
Logical Optimizer



Physical Optimizer



Changes to RDBMS

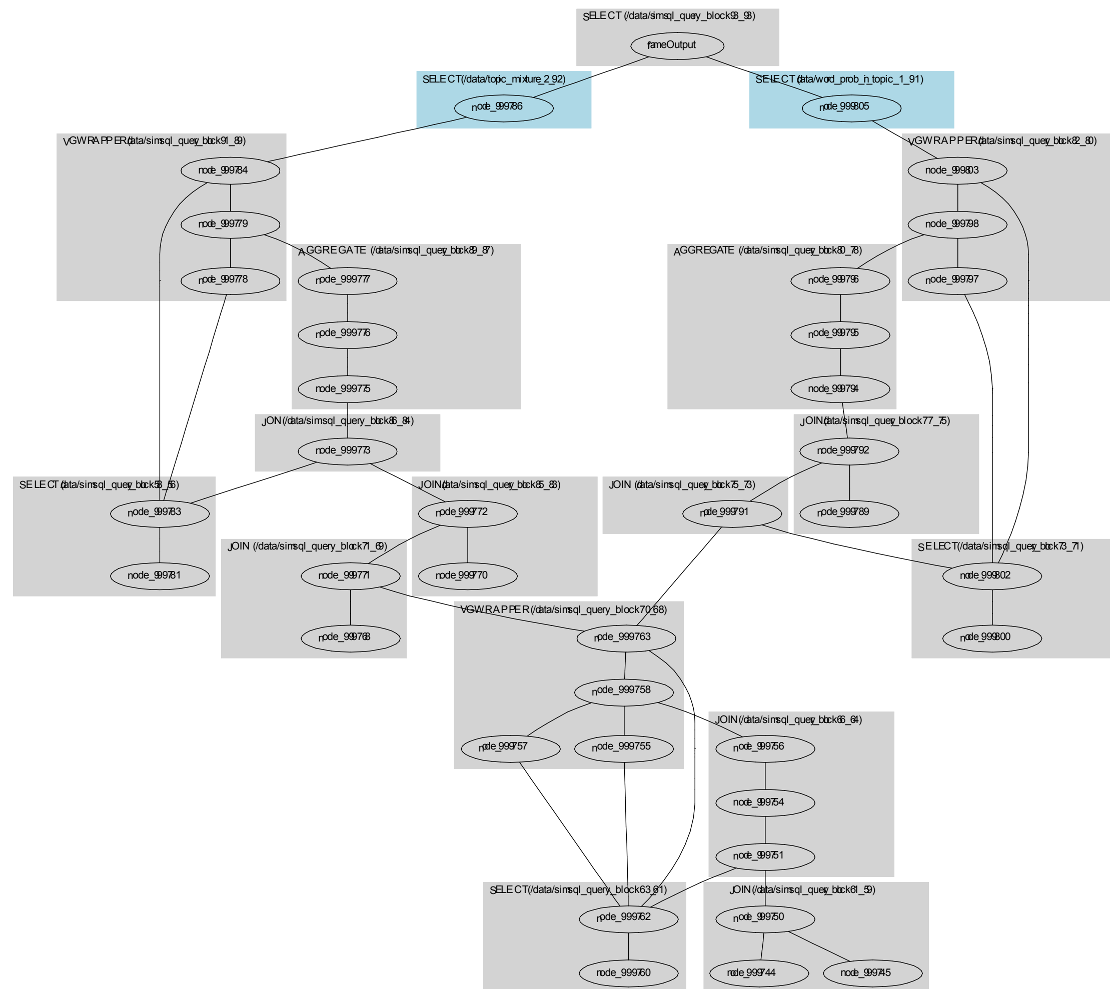


SimSQL for LDA (50 lines)

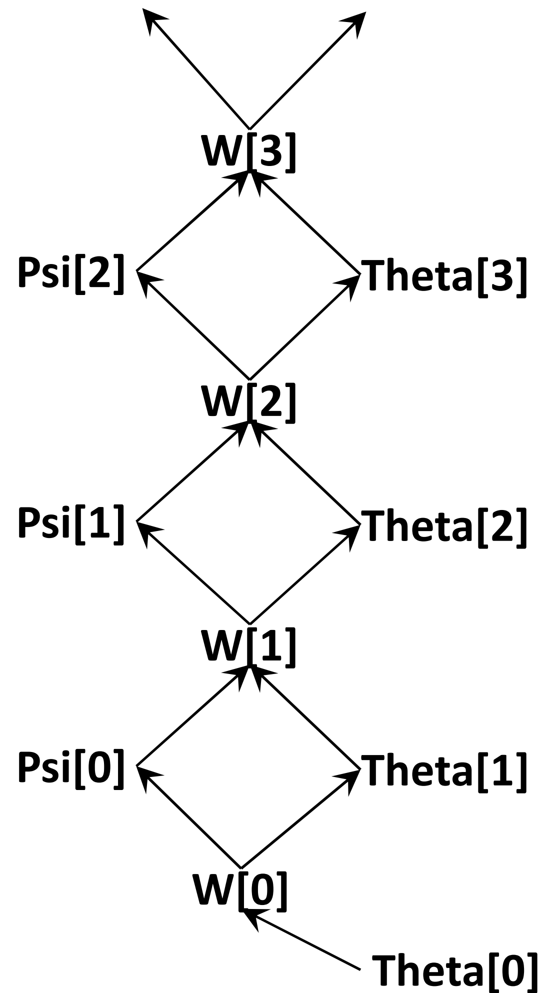
```
create table Theta[0](doc_id, topic_id, probability) as
for each d in docs
     with newprobs as Dirichlet (
          select topic_id, 1.0 from topics
     )
     select d.doc_id, n.out_id, n.probability
     from newprobs as n;

create table W[0] (doc_id, word_id, topic_id, count_num) as
for each dw in word_in_doc
     with topic_count as Multinomial (
          (select tm.topic_id, tm.probability
           from Theta[0] tm
           where tm.doc_id = dw.doc_id),
          (select dw.count_num)
     )
     select dw.doc_id, dw.word_id, wt.out_id, wt.out_count
     from topic_count wt;

create table Psi[i] (topic_id, word_id, probability) as
for each t in topics
     with newprobs as Dirichlet (
          select pw.word_id, sum(count_num) + 1.0
          from W[i] pw
          where pw.topic_id = t.topic_id
          group by pw.word_id
     )
     select t.topic_id, n.out_id, n.probability
     from newprobs n;
```

```
create table Theta[i] (doc_id, topic_id, probability) as
for each d in docs
     with newprobs as Dirichlet(
          select pw.topic_id, sum(count_num) +  1.0
          from W[i-1] pw, topics t
          where pw.doc_id = d.doc_id and pw.topic_id = t.topic_id
          group by pw.topic_id
     )
     select d.doc_id, n.out_id, n.probability
     from newprobs n;

create table W[i] (doc_id, word_id, topic_id, count_num) as
     for each dw in word_in_doc
          with topic_count as Multinomial
          (
               (
                    select tm.topic_id, wpt.probability *  tm.probability
                    from Psi[i-1] wpt, Theta[i] tm
                    where wpt.word_id = dw.word_id and
                    wpt.topic_id = tm.topic_id and
                    tm.doc_id = dw.doc_id
               ),
               (
                    select dw.count_num
               )
          )
          select dw.doc_id, dw.word_id, wt.out_id, wt.out_count
          from topic_count wt;
```

One Iteration Plan for LDA



A chain of these plans



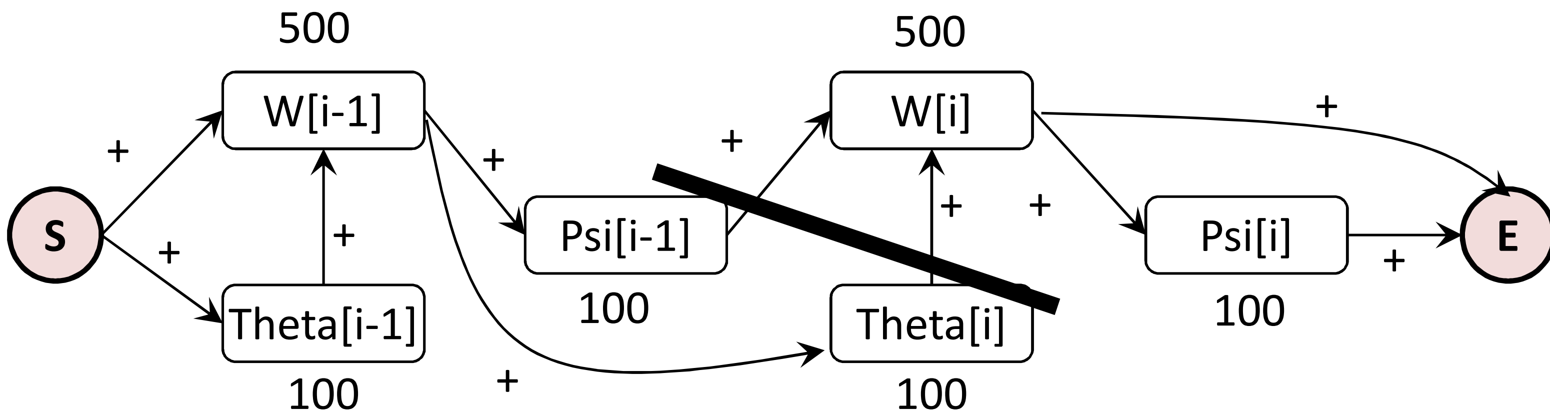
- Thousands of operators
- Optimize the whole plan together:
  - No way for optimization
  - No reliability
- Optimize random tables one by one:
  - Optimizer overhead
  - Hurts the optimization

Solution

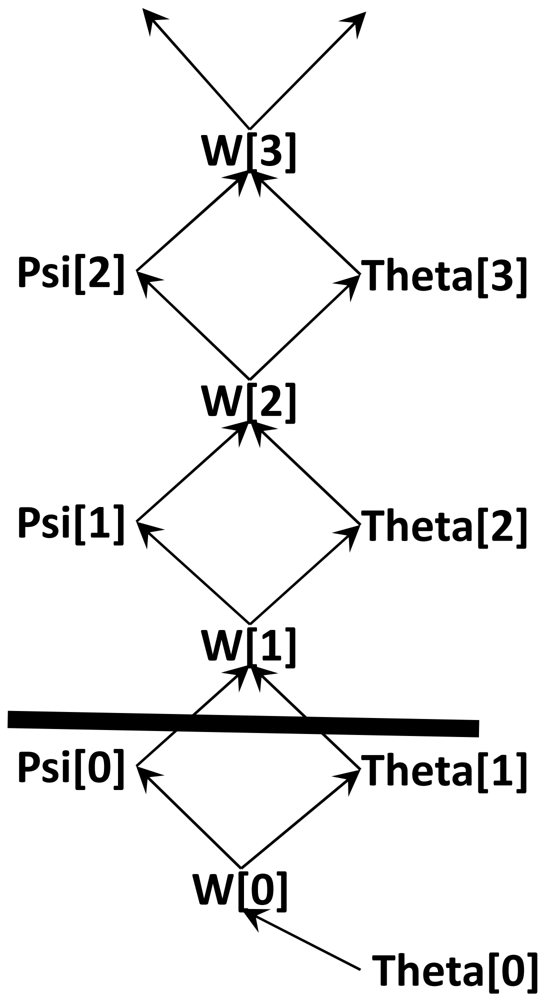
- Frame-based Optimization/Execution
  - Find the cut for the check-pointing.
  - Slice the plan, optimize, and execute frames alternatively.

Steps

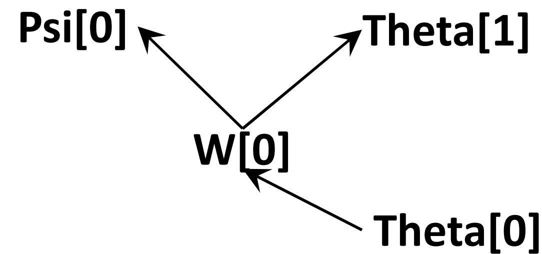
1. Compiler links together two non-baseline iteration of plans.
2. Compiler sends the plan for optimization.
3. System analyzes the plan, and figures out the minimum cut.



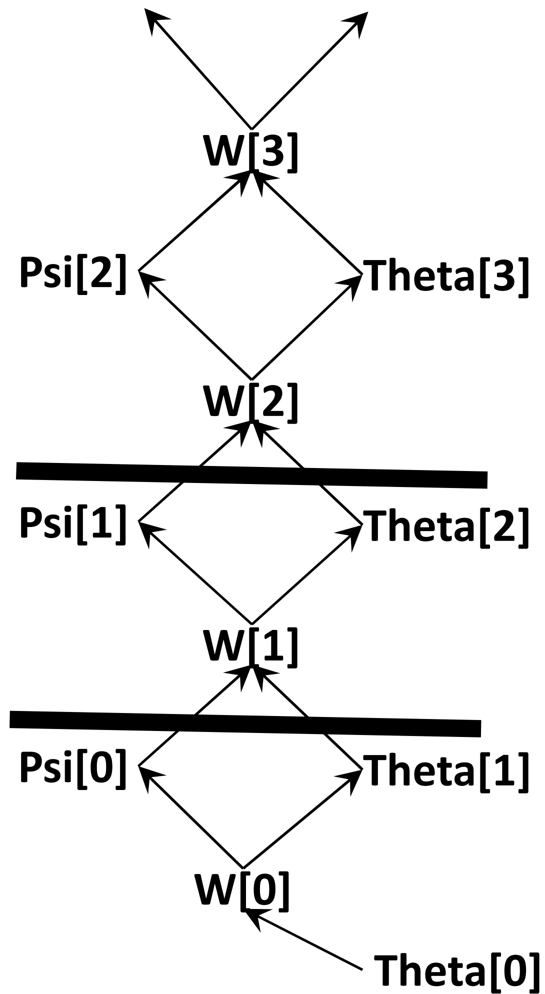
Steps



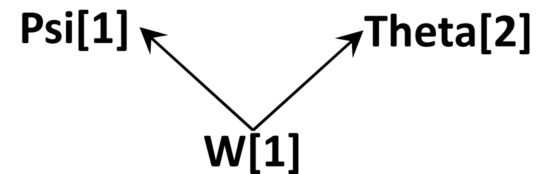
4. Generate the first piece of plan, optimize and execute it, and update the statistics.



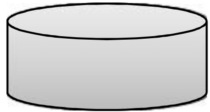
Steps



5. Generate the second plan, optimize and execute it, and update the statistics.



SimSQL for GMM



data(data_id, dim_id, dim_value)
cluster(clus_id, pi_prior)

1 We use four random tables to represent four parameters.

clus_means[i] (clus_id, dim_id, dim_value)
clus_covas[i] (clus_id, dim_id1, dim_id2, dim_value)
clus_prob[i] (clus_id, prob)
membership[i] (data_id, clus_id)

SimSQL for GMM

② Initialize hyper-parameters, e.g, μ_0

```
create view mean_prior(dim_id, dim_value) as
    select dim_id, avg(dim_value) from data group by dim_id;
```

③ Initialize parameters, e.g., π_0

```
create table clus_prob[0] (clus_id, prob) as
    with diri_res as Dirichlet(
        select clus_id, pi_piror from cluster)
    select diri_res.out_id, diri_res.prob
    from diri_res;
```

SimSQL for GMM

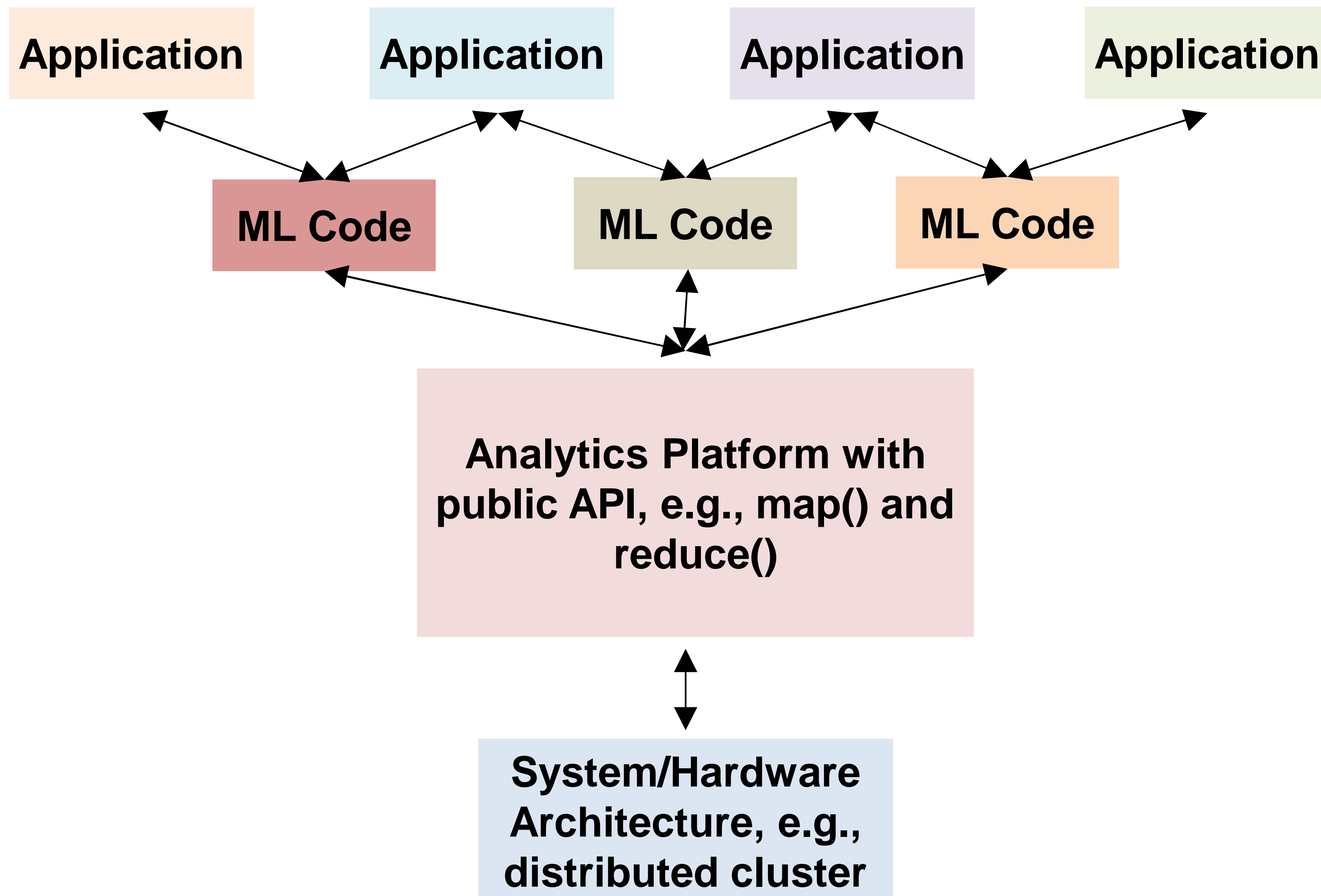
④ Update parameters, e.g., π_i

```
create table clus_prob[i] (clus_id, prob) as
    with diri_res as Dirichlet
    (
        select cmem.clus_id, cmem.c_num+clus.pi_piror
        from clus,
            (select cm.clus_id as clus_id, count(cm.data_id) as c_num
             from membership[i-1] as cm) as cmem
        where cmem.clus_id = clus.clus_id
     )
    select diri_res.out_id, diri_res.prob
    from diri_res;
```

Outline

- Gaussian Mixture Model
- MapReduce extensions
- Parallel DB
- Graph ML
- Benchmark

Parallel ML in A Higher Level



Graph Processing Systems

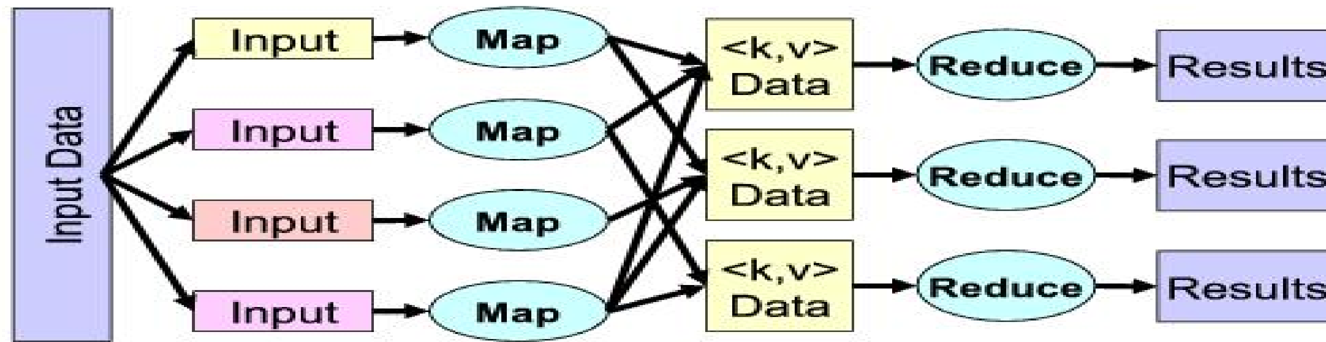


Figure. MapReduce Framework.

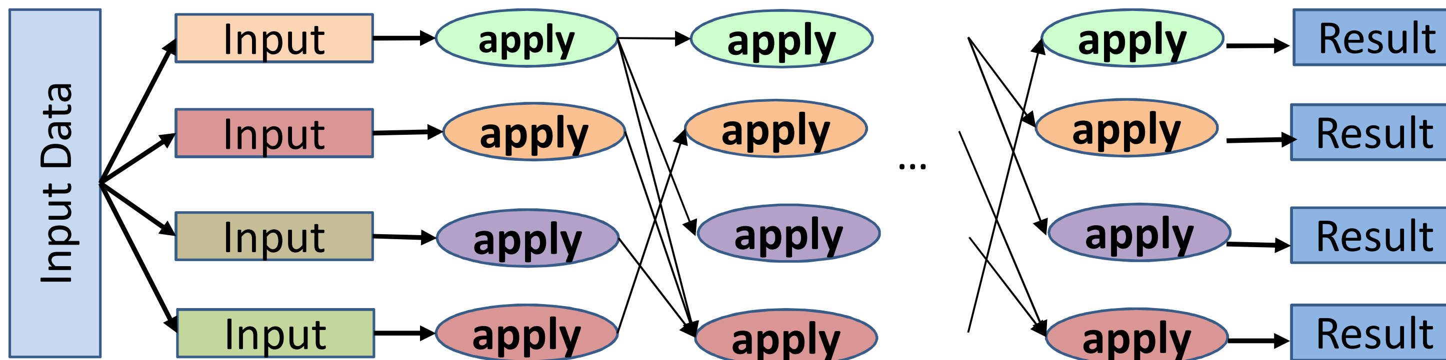


Figure. Graph Processing

Single Shortest Path

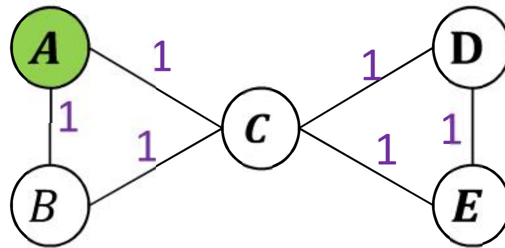


Figure. A graph with five nodes. The task is to find the length from A to all the nodes in the graph.

Pregel

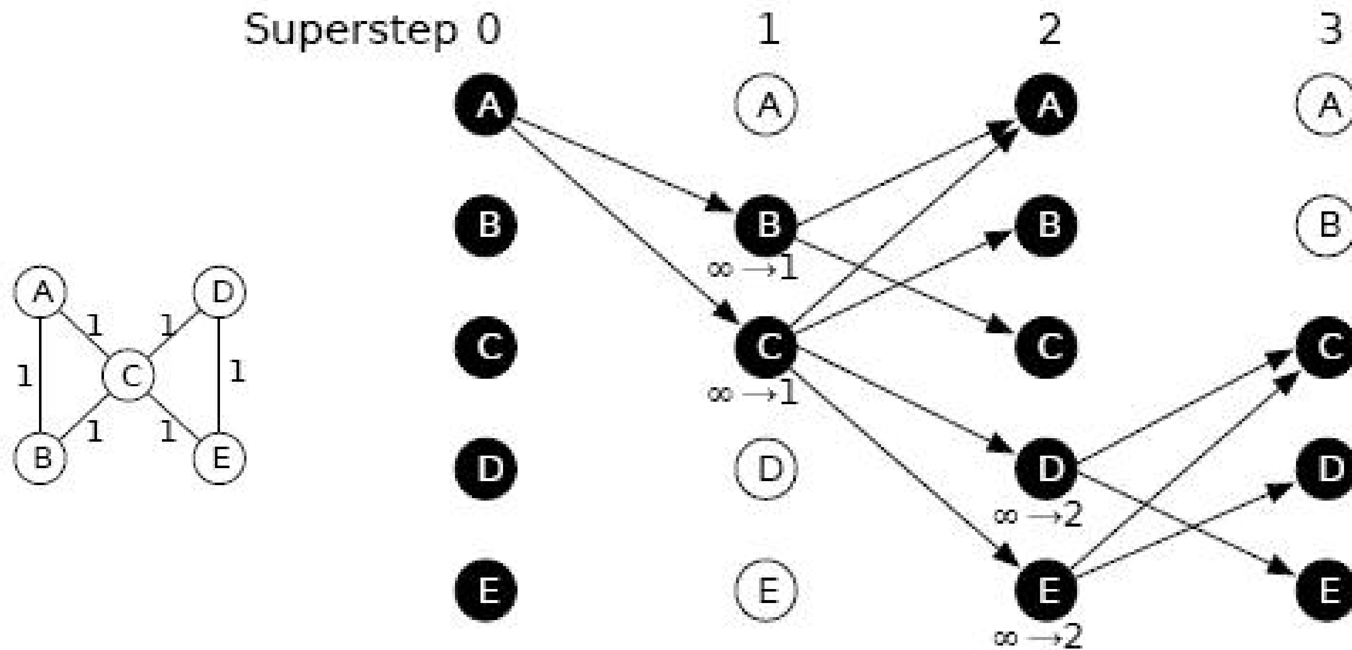


Figure. The execution procedures of Pregel for the single shortest path problem.

Algorithm

Algorithm 1 Computation function C for the single-source shortest paths algorithm.

1: **function** $C(v, S, I)$
2: $\quad mindist$ = is_source(v) ? 0 : $+inf$;
3: $\quad$ **for all** msg in I **do**
4: $\quad\quad mindist = \min(mindist, msg.value)$
5: $\quad$ **end for**
6: $\quad$ **if** $mindist < S.value$ **then**
7: $\quad\quad S.value = mindist$
8: $\quad\quad$ **for all** $edge$ in $v.get_edges()$ **do**
9: $\quad\quad\quad$ send_message($edge.dst$, $mindist + edge.value$)
10: $\quad\quad$ **end for**
11: $\quad\quad$ vote_to_halt();
12: $\quad$ **end if**
13: **end function**

Figure. Single shortest path algorithm with Pregel model.

Gaussian Mixture Model

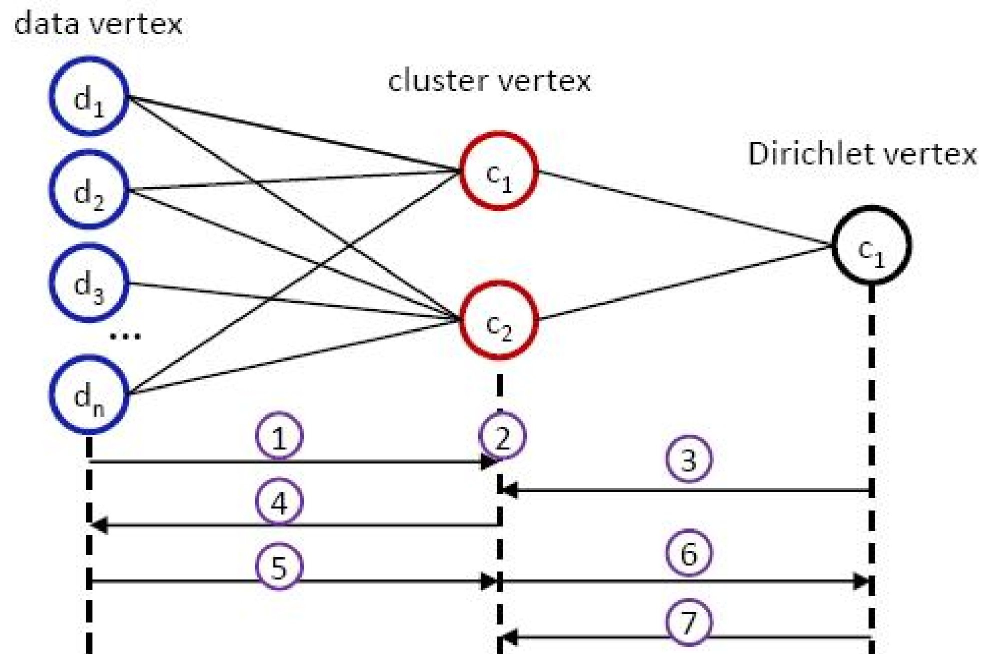


Figure. The execution procedures of GMM in Giraph.

Optimization

1. In Step 1, 5, 9, ..., using combiner to reduce communication overhead.
2. In Step 4, 8 ,12, …, broadcasting the model to data vertices.

Outline

- Gaussian Mixture Model
- MapReduce extensions
- Parallel DB
- Graph ML
- Benchmark

public ML libraries

- Mahout
- MLlib
- Pregasus
- …

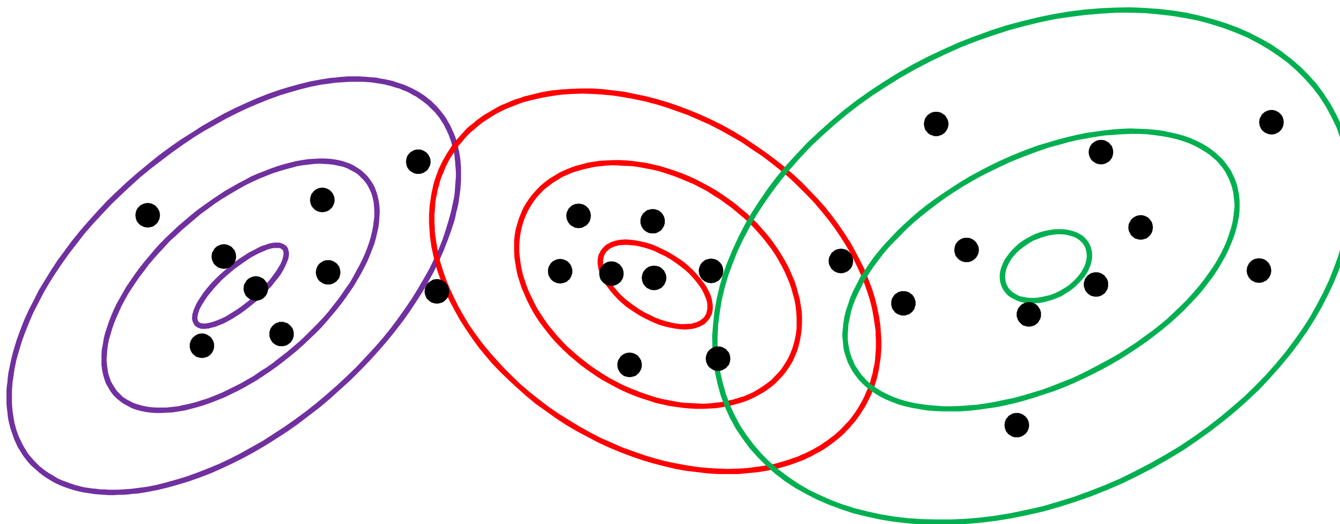
Benchmark configuration

Systems	Spark, SimSQL, GraphLab, Giraph.
Hardware	Amazon EC2, m.2.4x large instance, 8 cores, 64 GB RAM per machine, with 5, 20, 100 machines.
Problems	Gaussian mixture model (GMM) Latent Dirichlet Allocation (LDA) Bayesian Lasso Hidden Markov Model(HMM) Gaussian Imputation
Datasets	200 GB ~ 1TB
Comparison metrics	Programmability and performance

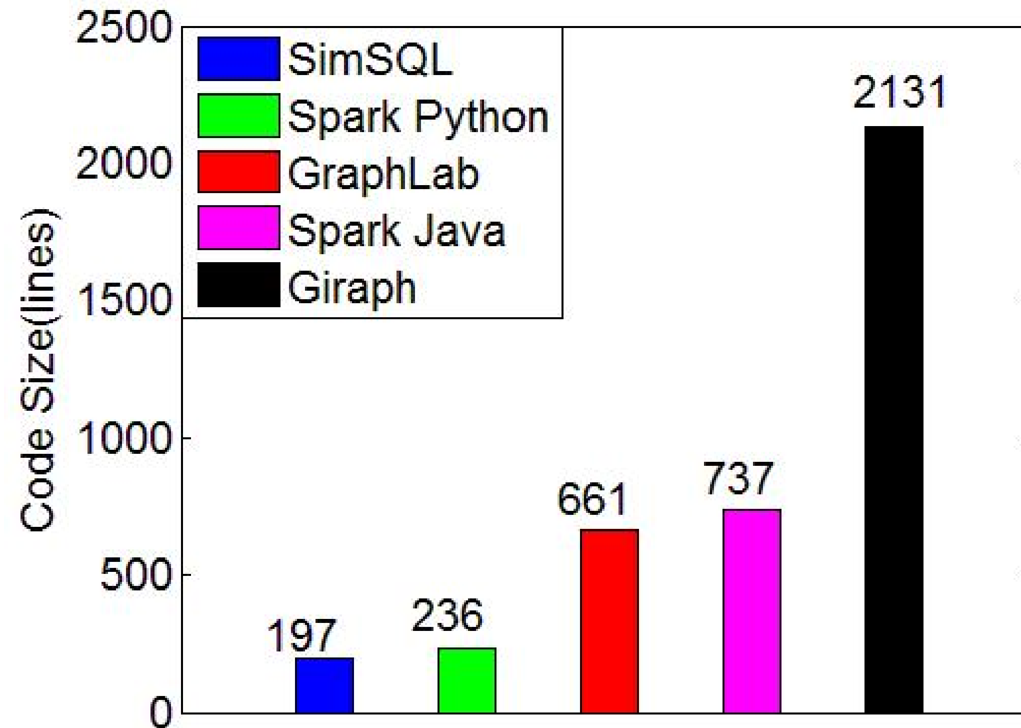
70, 000 + hours of Amazon EC2 time, $100, 000 @ on-demand price, 5 researchers, 5 months' work.

GMM

- Infer this:

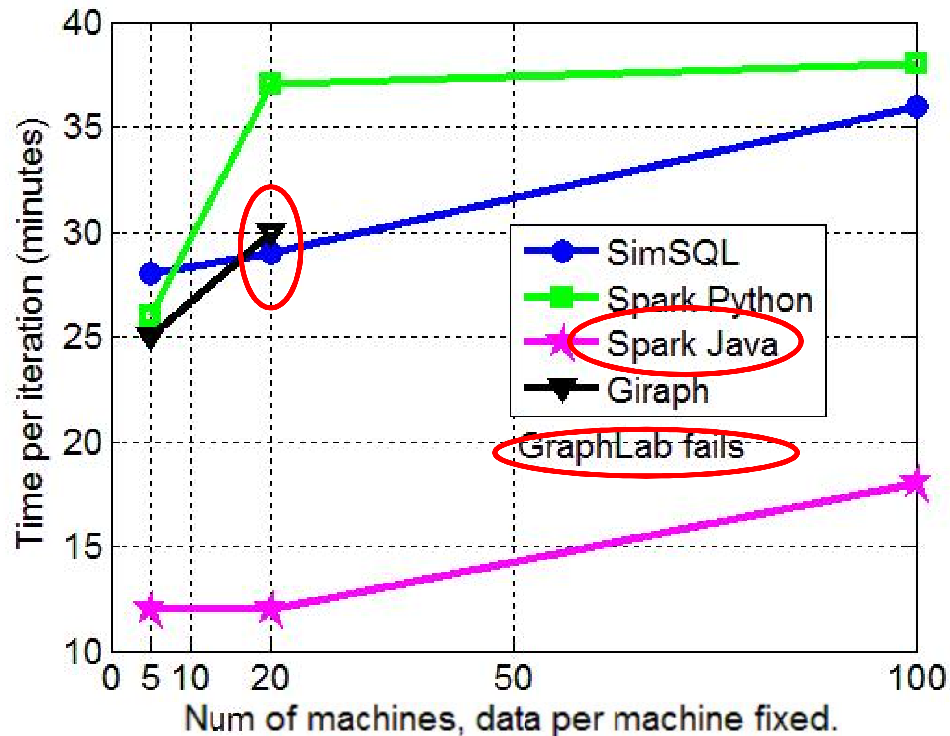


Programmability



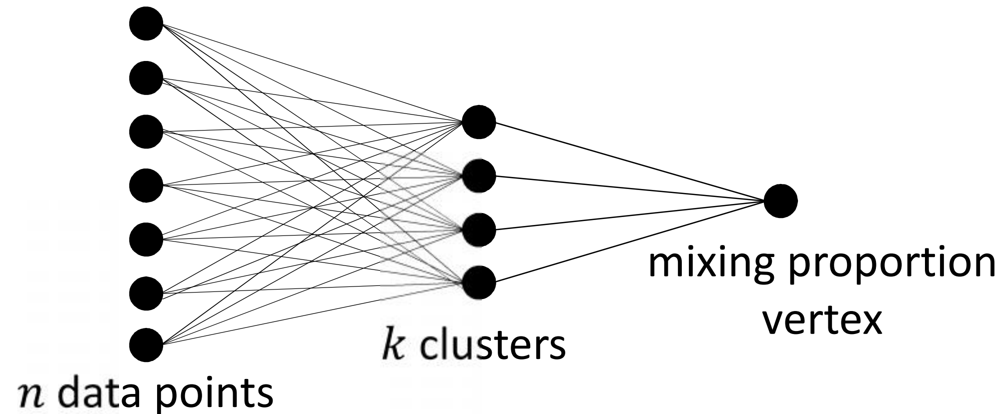
The programmability of different platforms for GMM in the "natural" implementation.

Performance



The "natural" implementation for GMM, and the data scale is: 10^7 *data points per machine* $\times$ 10 *dims* $\times$ 10*clusters* $\times n$ *machines* ($m = 5, 20, 100$).

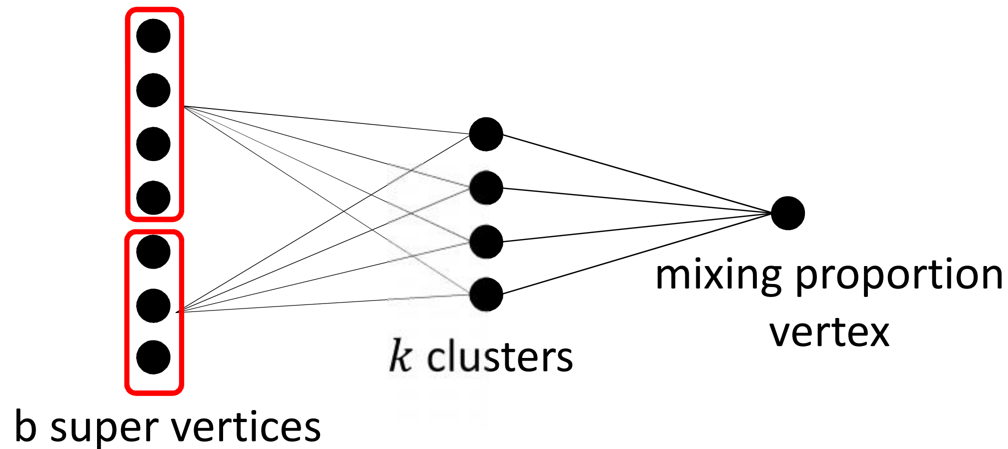
The problem of GraphLab



GraphLab / Giraph graphical model

- GraphLab
  - pull-based model: each *cluster* needs access to its neighbors.
  - Memory usage for the machine holding clusters is too large: 1cluster * 1 billion data points * 100 bytes = 100 G.
- Giraph
  - push-based model, combiner, aggregator.

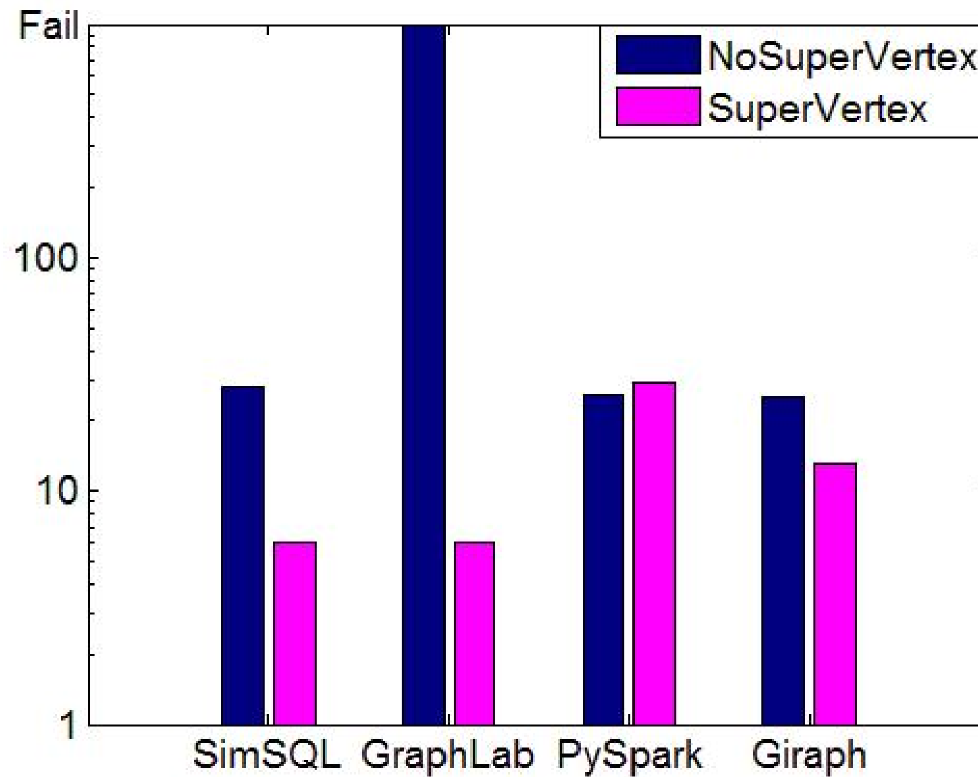
Super Vertex Implementation



- Super-vertex implementation for GraphLab

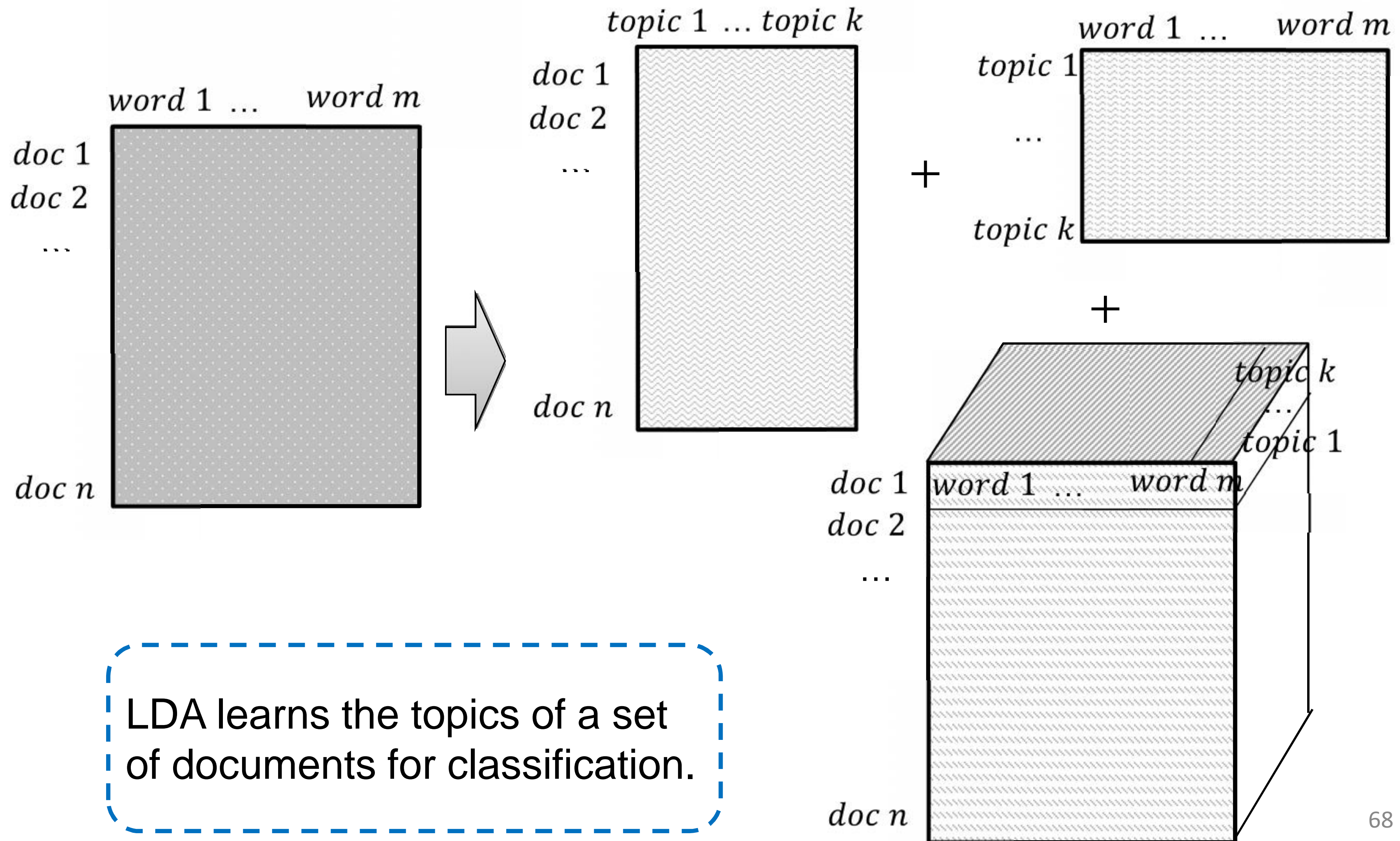
	Super Vertex	10 dimensions		
	Lines of code	5 machines	20 machines	100 machines
GraphLab	681	6:13	4:36	6:09

Performance Improvement of Super Vertex



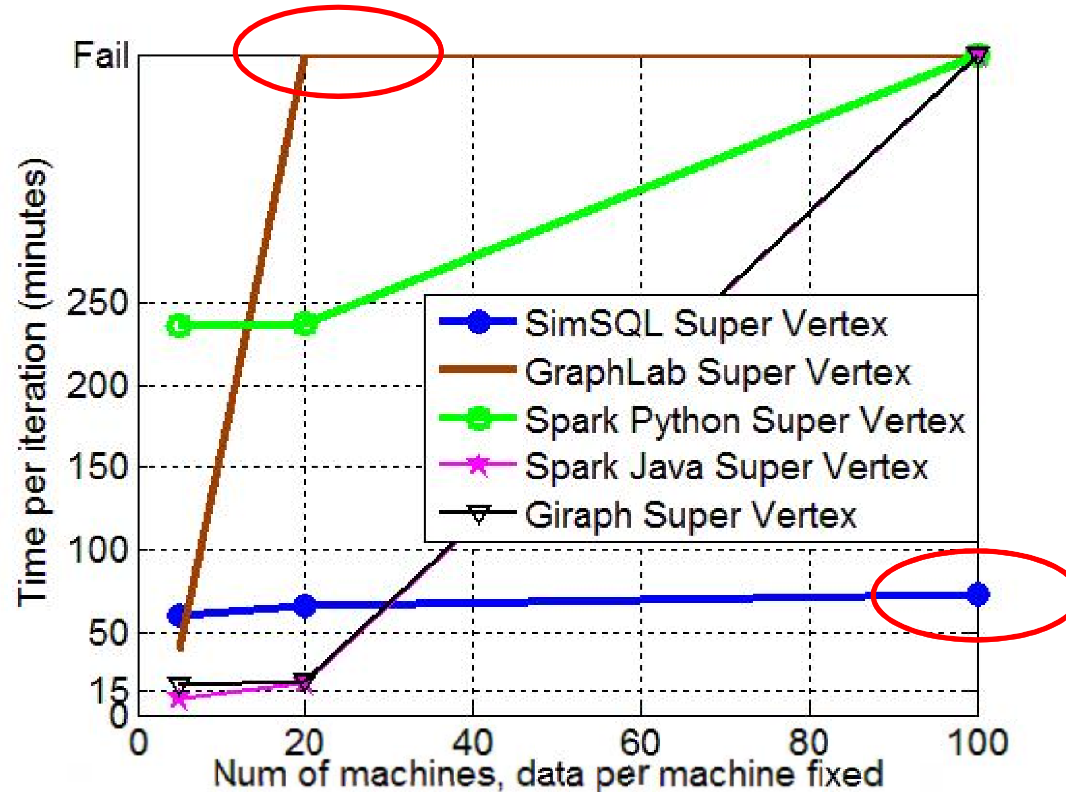
The impact of using super-vertex implementation.

Problem 3. LDA



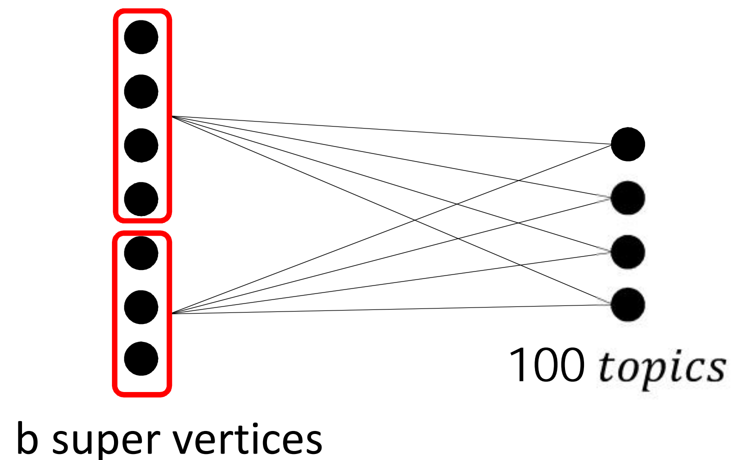
LDA learns the topics of a set of documents for classification.

Performance



The "multi-docs"-based LDA, and the data scale is: 2×10^6 $\boldsymbol{docs\ per\ machine} \times 10^4$ $\boldsymbol{words} \times 10^2\boldsymbol{topics} \times$ $\boldsymbol{n}$ $\boldsymbol{machines}$ $(\boldsymbol{m} = \mathbf{5, 20, 100})$.

Why GraphLab fails again?



A topic vertex needs memory:

$$\mathbf{10^4}\ \boldsymbol{super\ vertices} * \mathbf{10^2}\ \boldsymbol{topic} * \mathbf{10^4}\ \boldsymbol{dictionary} * \mathbf{4}(\boldsymbol{int\ size}) = \mathbf{40}\ \boldsymbol{G}$$

Homework

- Try to implement the LDA model in all three systems.