Large-scale Ads CTR prediction with Spark and deep learning: lessons learned

Yanbo Liang
Apache Spark committer
Software engineer @ Hortonworks

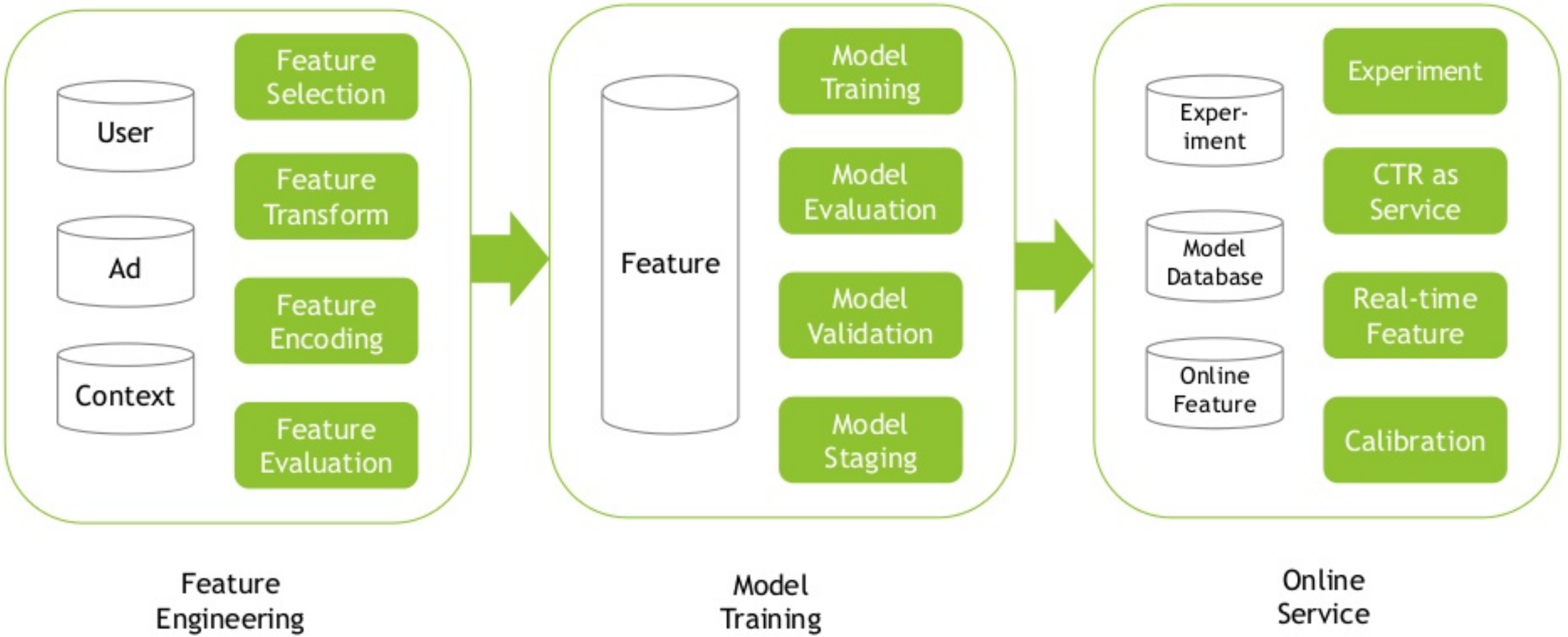
Outline

- Basic CTR estimation pipeline
- Scalable CTR estimation pipeline
  - Vector-free LBFGS on Spark
  - Large-scale logistic regression on vector-free LBFGS
- Advanced CTR estimation pipeline
  - Learn part of features from deep neural network

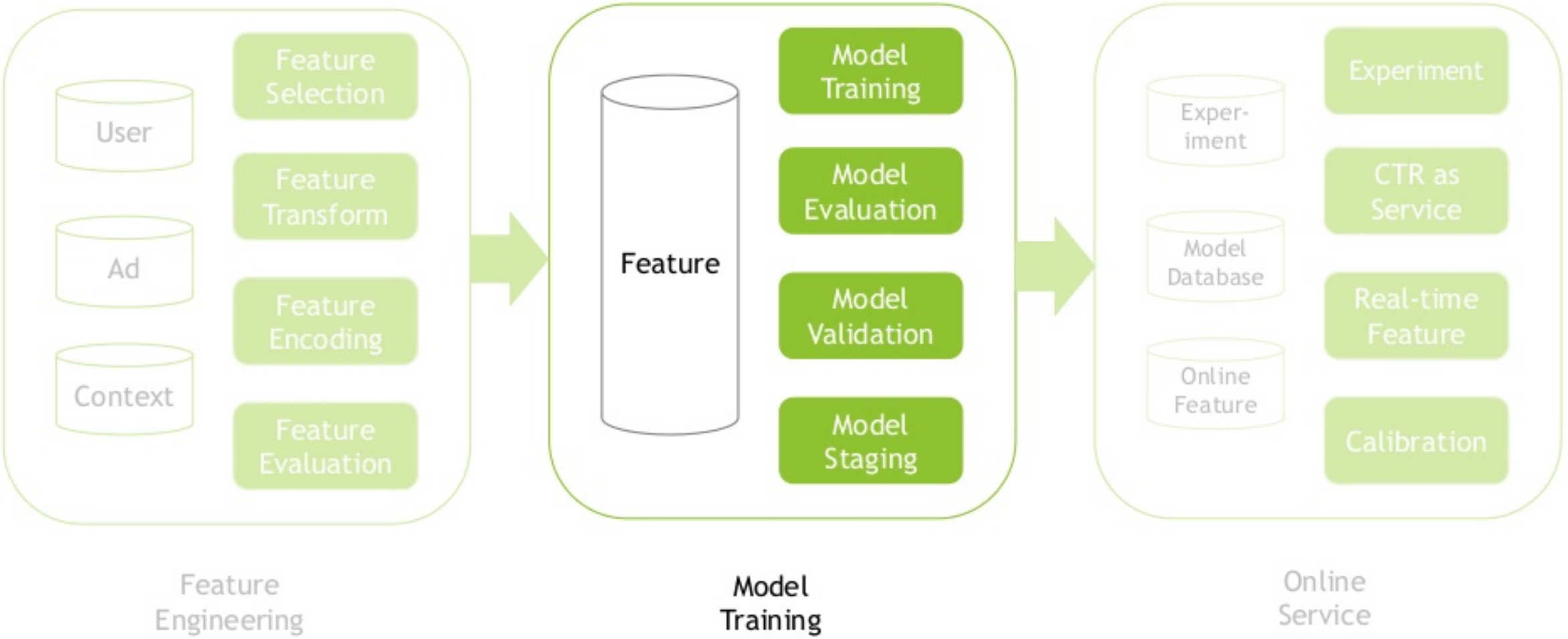
Outline

- Basic CTR estimation pipeline
- Scalable CTR estimation pipeline
  - Vector-free LBFGS on Spark
  - Large-scale logistic regression on vector-free LBFGS
- Advanced CTR estimation pipeline
  - Learn part of features from deep neural network

CTR estimation workflow



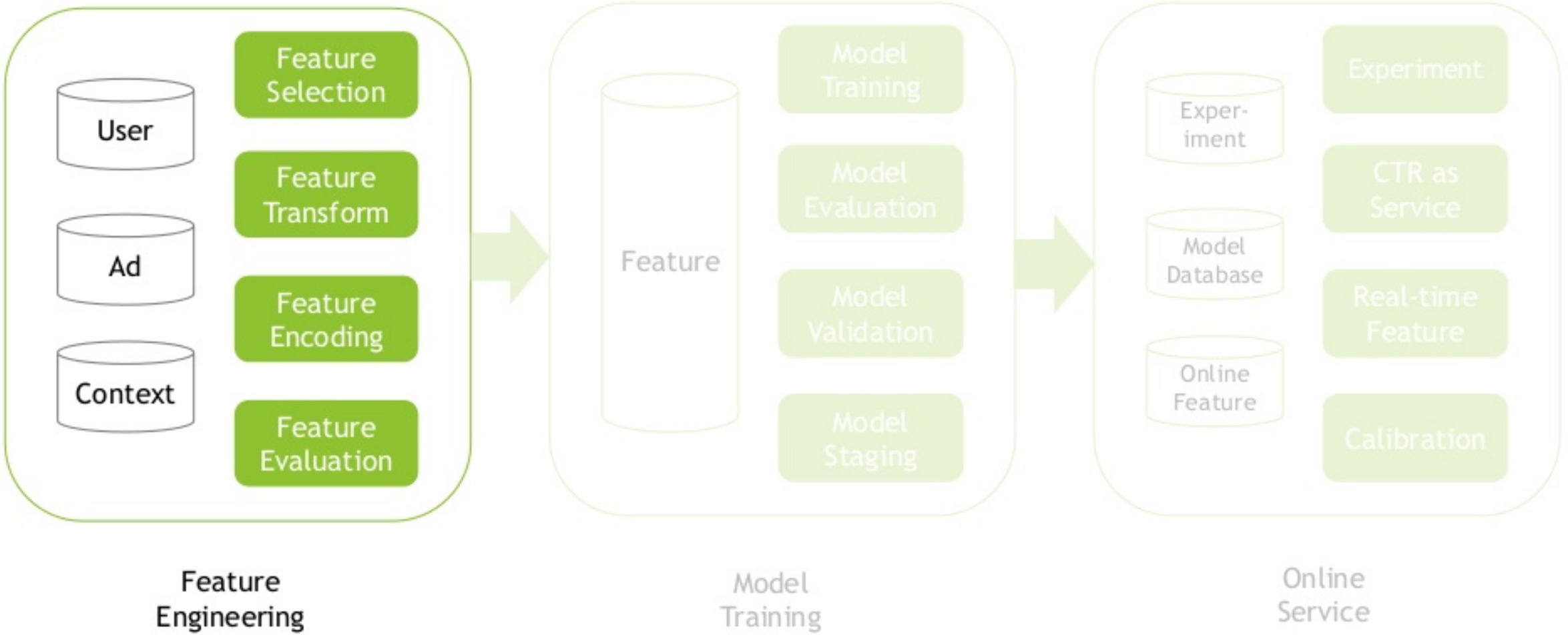
CTR estimation workflow



Occam's razor: Logistic Regression

- Train quickly, compared with non-linear model. Can work well on very large feature sets.
- Can be interpreted and debugged more easily. You can examine the weights assigned to each feature to figure out what's having the biggest impact on a prediction.
- High-dimensional feature space, but very sparse.
- Usually train with L1 regularization to produce sparse solution, which can meet the requirement of model serving.
- Be widely used.

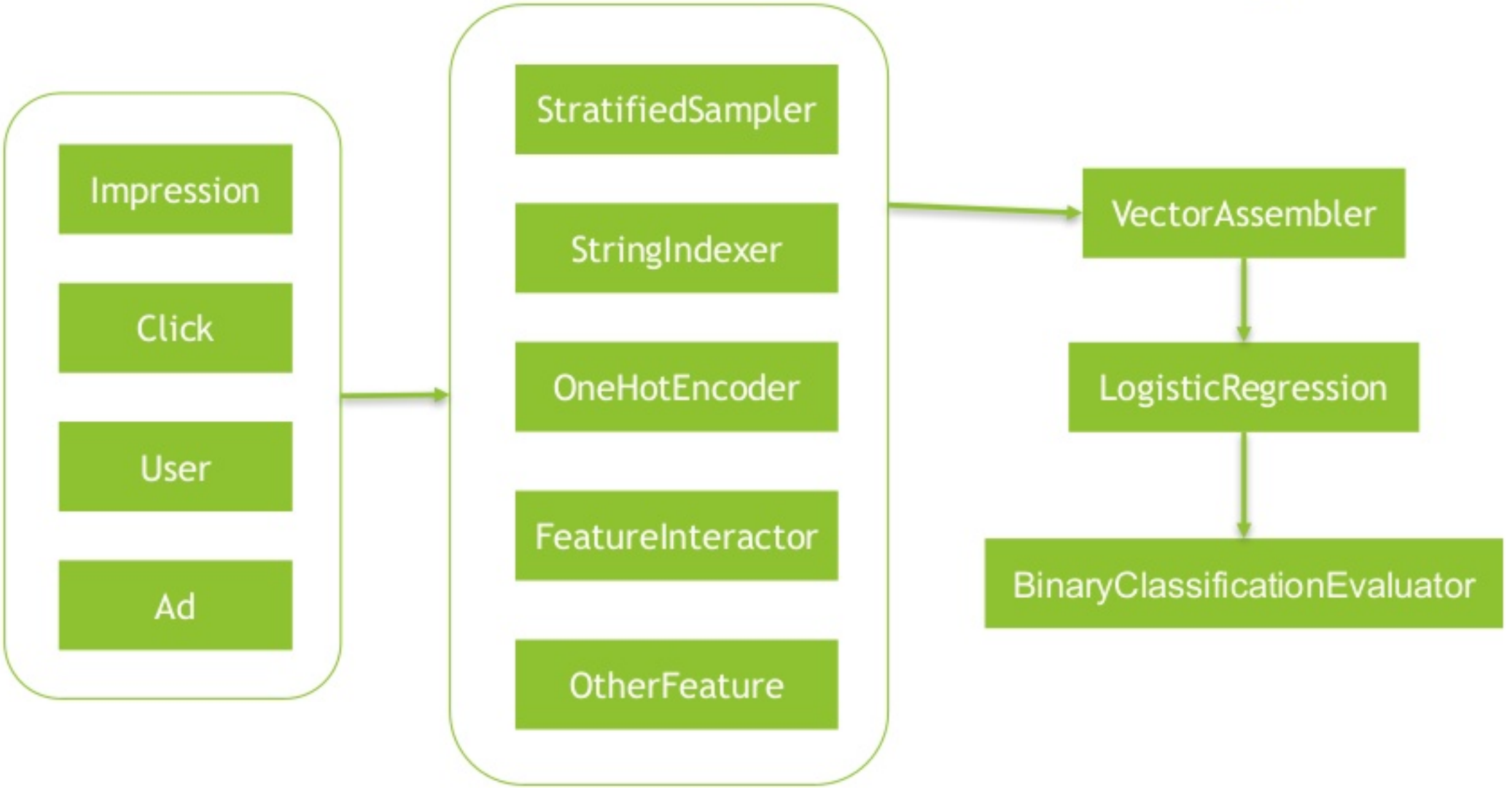
CTR estimation workflow



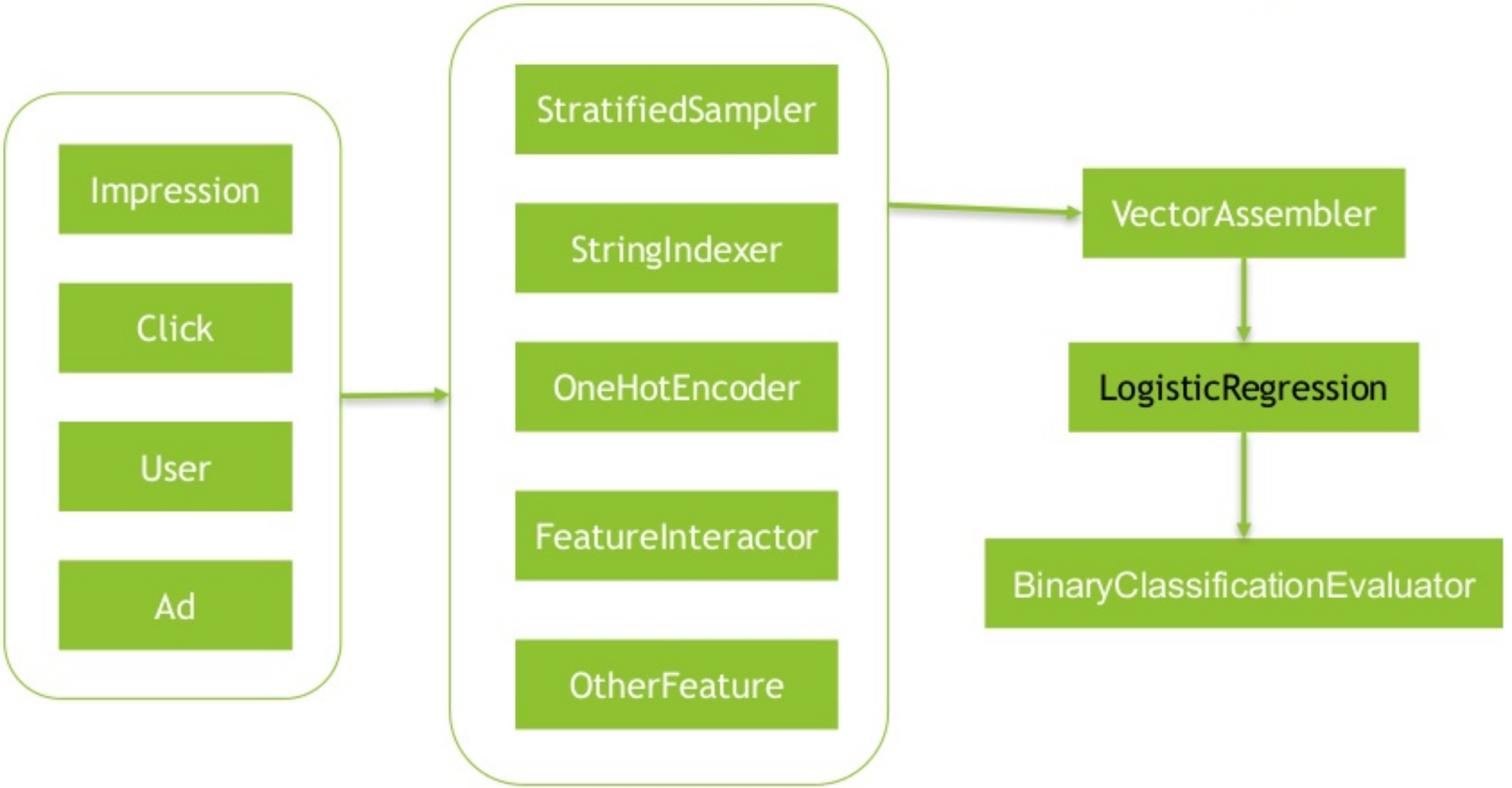
Feature Engineering

- Encoding for categorical feature:
  - OneHotEncoder
- Discretize continuous feature into buckets:
  - QuantileDiscretizer
  - StringIndexer
- Feature cross or combination: age_gender_state
  - User defined transformer (FeatureInteractor)
  - PolynomialExpansion

Basic CTR estimation pipeline on Spark



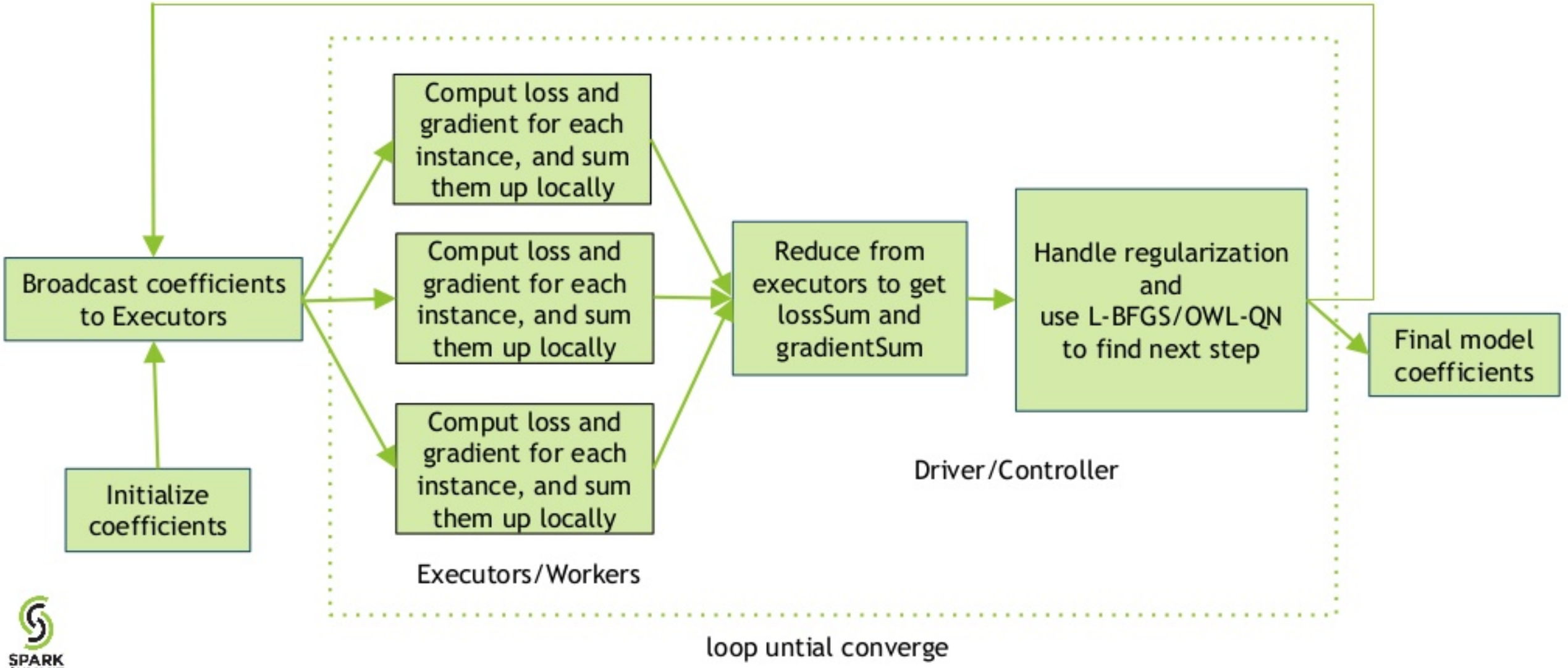
Basic CTR estimation pipeline on Spark



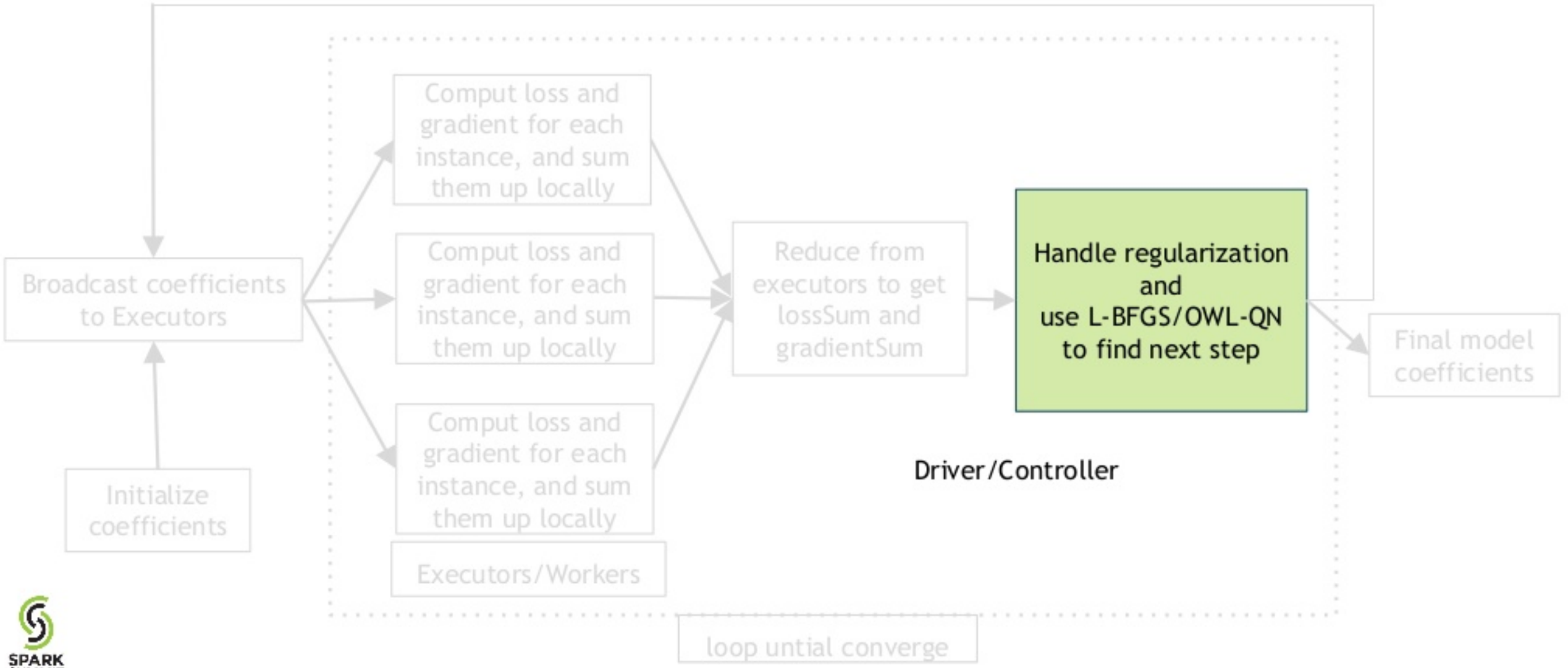
Outline

- Basic CTR estimation pipeline
- Scalable CTR estimation pipeline
  - Vector-free LBFGS on Spark
  - Large-scale logistic regression on vector-free LBFGS
- Advanced CTR estimation pipeline
  - Learn part of features from deep neural network

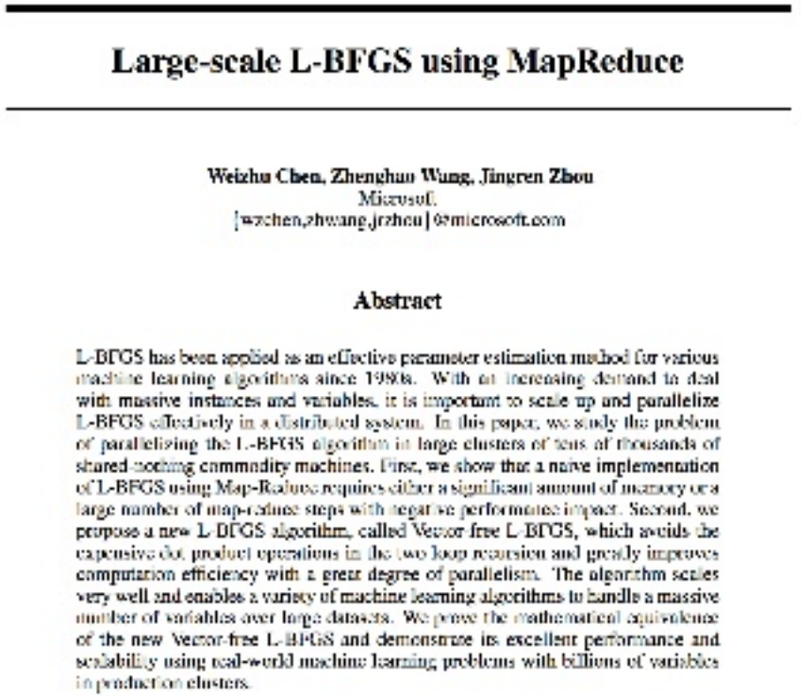
MLlib Logistic Regression



Not model parallel



Vector-free L-BFGS (VL-BFGS)

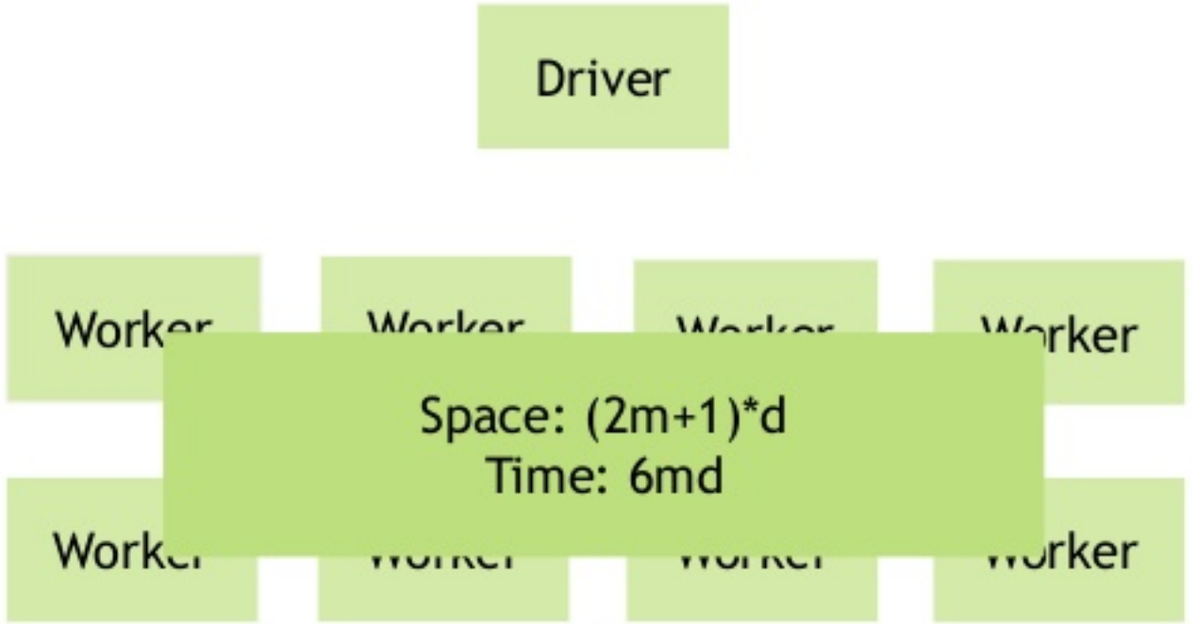


Large-scale L-BFGS using MapReduce

Weizhu Chen, Zhenghao Wang, Jingren Zhou
Microsoft
[wzchen,zhwang,jrzhou]@microsoft.com

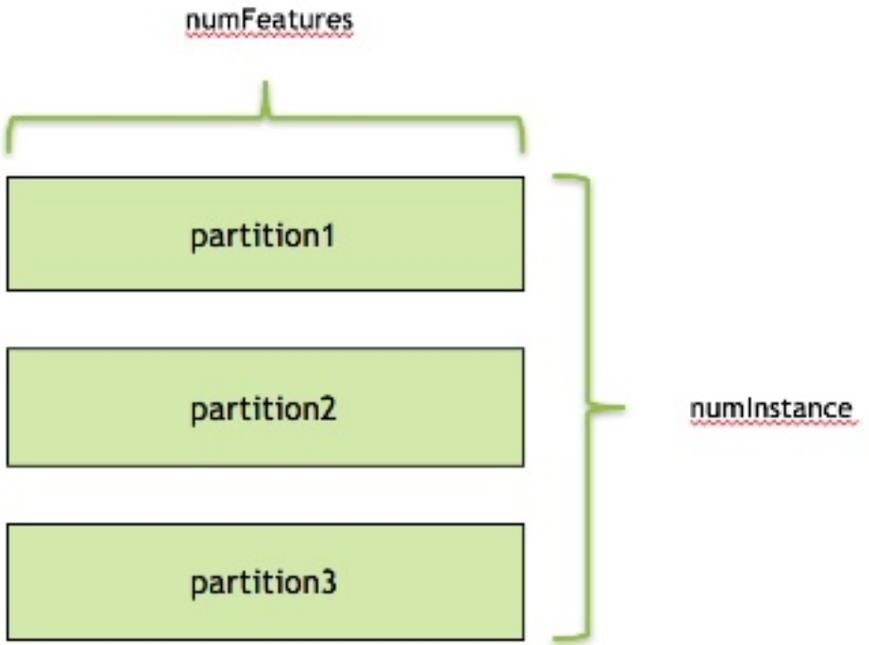
Abstract

L-BFGS has been applied as an effective parameter estimation method for various machine learning algorithms since 1980s. With an increasing demand to deal with massive instances and variables, it is important to scale up and parallelize L-BFGS effectively in a distributed system. In this paper, we study the problem of parallelizing the L-BFGS algorithm in large clusters of tens of thousands of shared nothing commodity machines. First, we show that a naive implementation of L-BFGS using Map-Reduce requires either a significant amount of memory or a large number of map-reduce steps with negative performance impact. Second, we propose a new L-BFGS algorithm, called Vector-free L-BFGS, which avoids the expensive dot product operations in the two loop recursion and greatly improves computation efficiency with a great degree of parallelism. The algorithm scales very well and enables a variety of machine learning algorithms to handle a massive number of variables over large datasets. We prove the mathematical equivalence of the new Vector-free L-BFGS and demonstrate its excellent performance and scalability using real-world machine learning problems with billions of variables in production clusters.

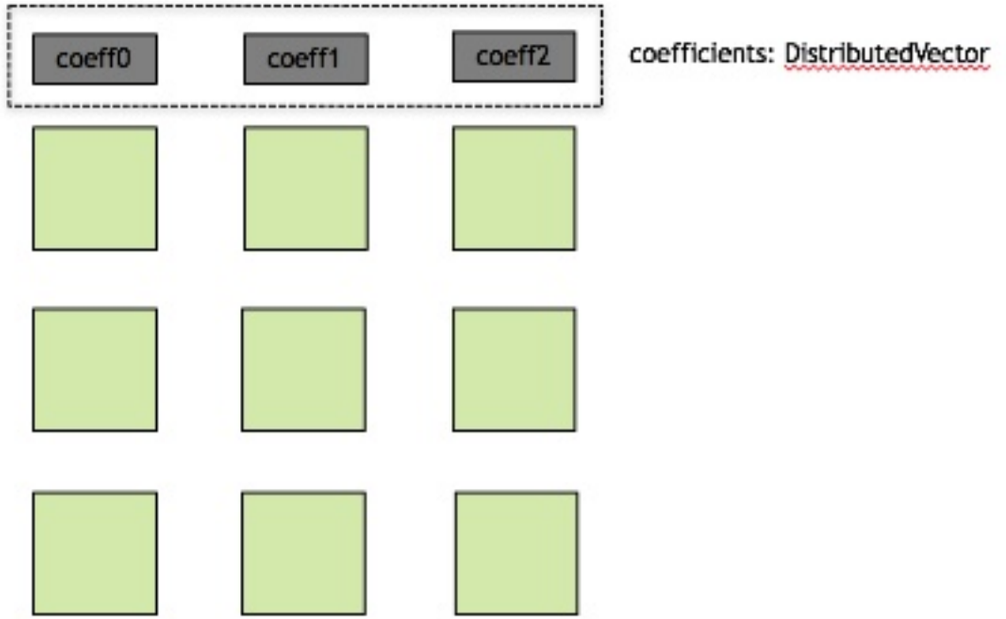


https://spark-summit.org/east-2017/events/scaling-apache-spark-mllib-to-billions-of-parameters/

Logistic Regression on VL-BFGS



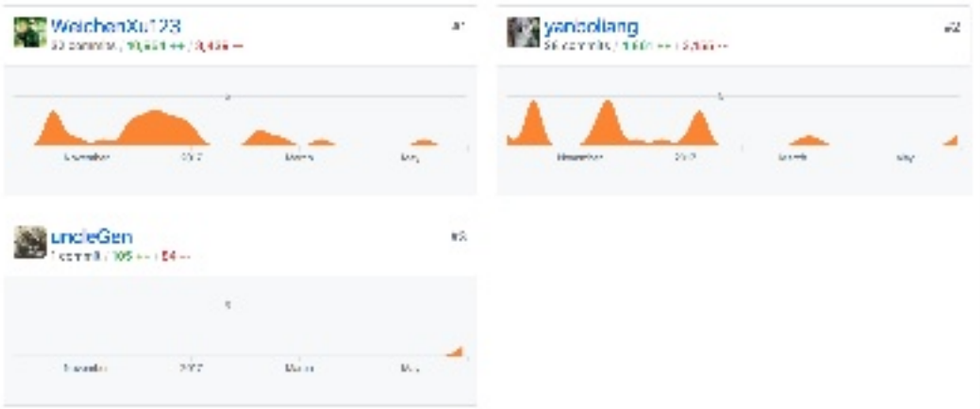
MLlib



spark-vlbfgs

spark-vlbfgs

- Open source (Apache license)
  - https://github.com/yanboliang/spark-vlbfgs
- Status
  - 1+ production user
  - 3+ experiment user
  - Contributors from 3 different companies



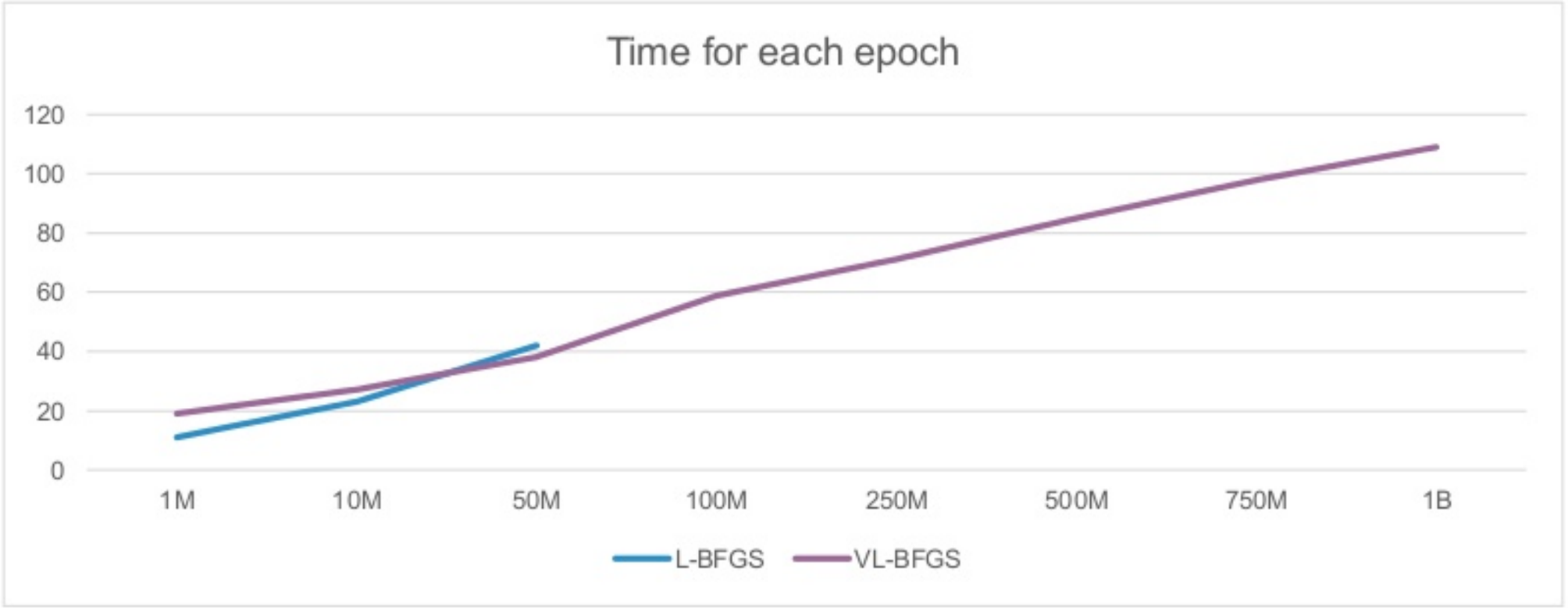
APIs

```
val dataset: Dataset[_] =
    spark.read.format("libsvm").load("data/a9a")
val trainer = new VLogisticRegression()
    .setColsPerBlock(100)
    .setRowsPerBlock(10)
    .setColPartitions(3)
    .setRowPartitions(3)
    .setRegParam(0.5)
val model = trainer.fit(dataset)
println(s"Vector-free logistic regression coefficients:
    ${model.coefficients}")
```

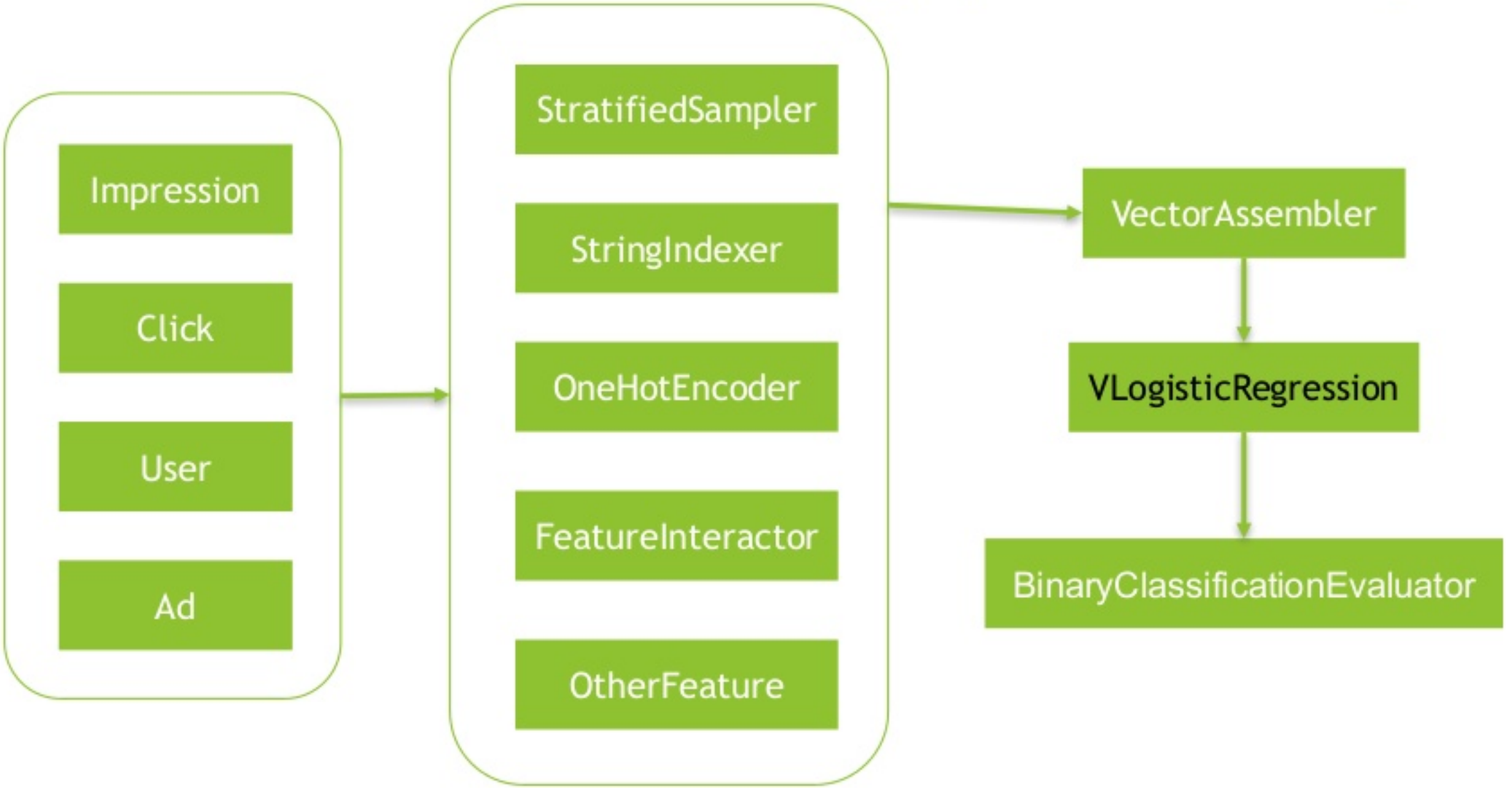
Key takeaways

- Full distributed calculation, full distributed model, can run successfully without OOM.
- VL-BFGS API is consistent with breeze L-BFGS, and it outputs exactly the same solution as breeze L-BFGS.
- Can implement other loss function on VL-BFGS.
- Pure library and can be deployed and used easily, doesn't require special components such as parameter servers.
- Can benefit from the Spark cluster operation and development experience.
- Use the same platform as the data size increasing.

Performance



Scalable CTR estimation pipeline on Spark



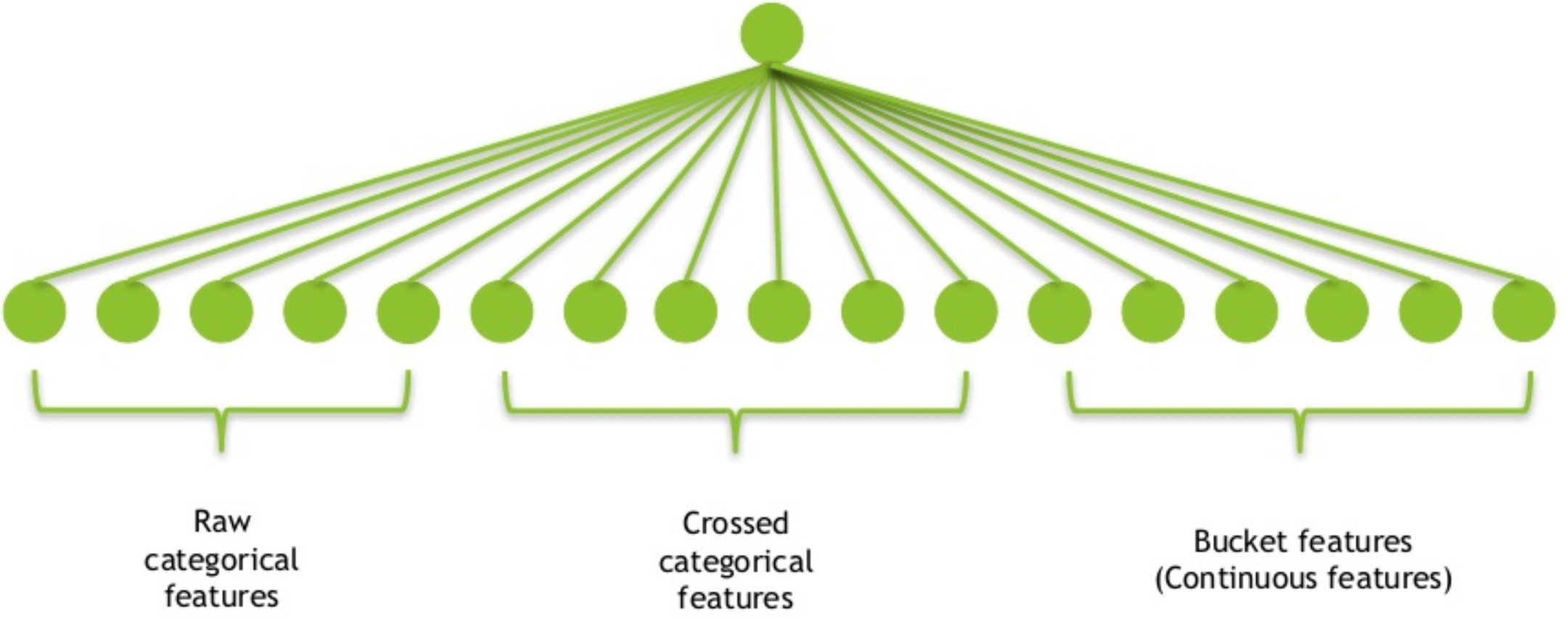
Offline experiment - online used

Model	System	AUC gain
LR	MLlib	0.000%
LR with more crosses	spark-vlbfgs	1.038%

Lessons learned

- More scalable logistic regression model (LR on VLBFGS).
  - Handle unbalanced data
  - RDD[Instance] -> Feature block matrix
  - Dot product between 2m+1 distributed vectors: use union + repartition rather than launch multiple jobs
  - COO sparse matrix (reduce 30% memory footprint)
- More effective non-linear feature combinations empirically.

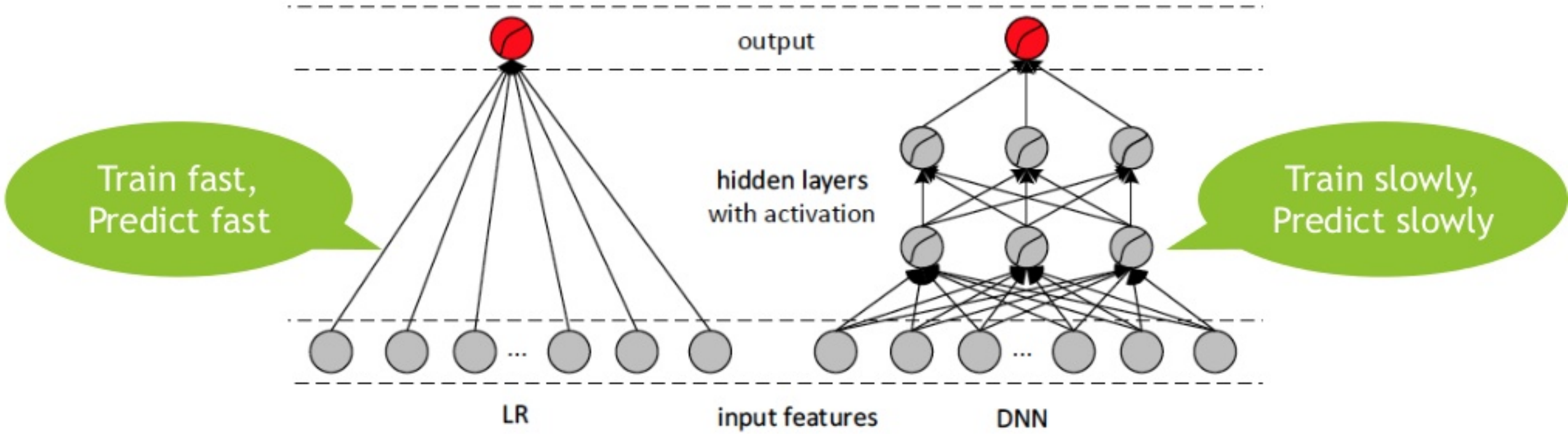
Model recap


Raw categorical features
Crossed categorical features
Bucket features (Continuous features)

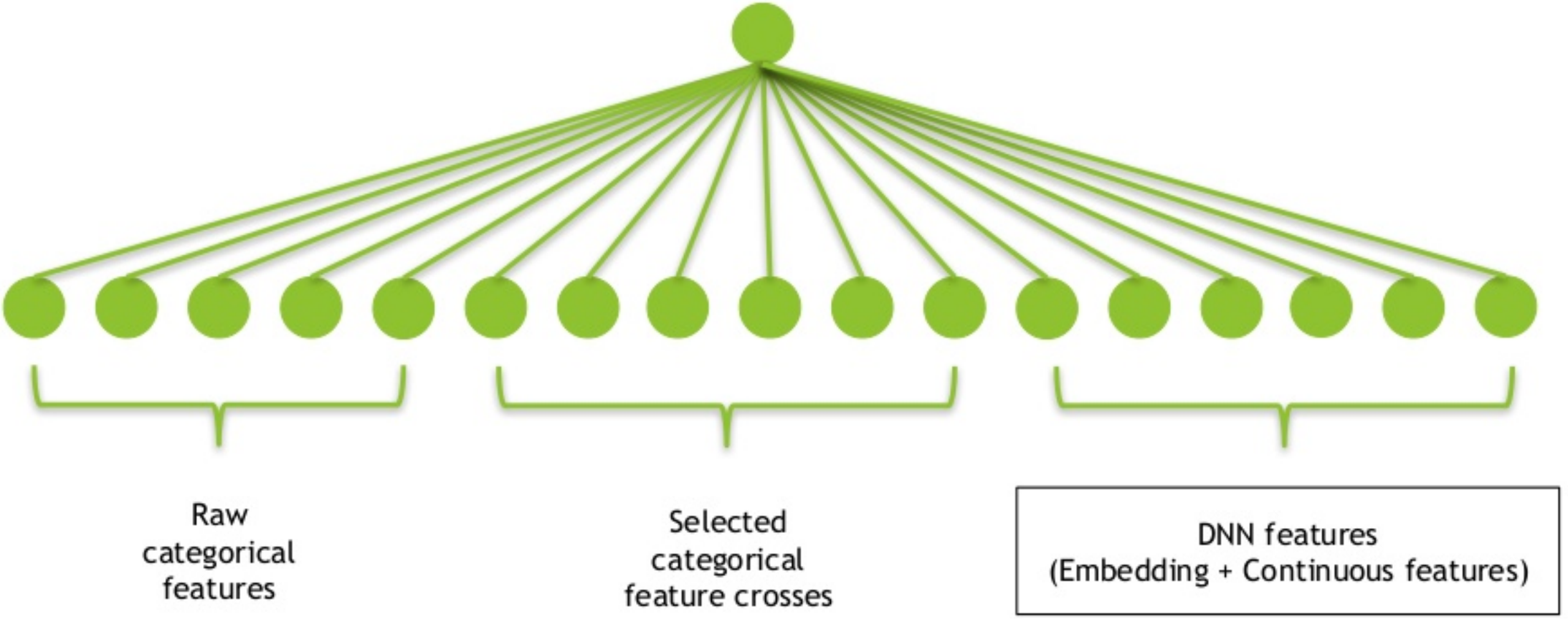
Outline

- Basic CTR estimation pipeline
- Scalable CTR estimation pipeline
  - Vector-free LBFGS on Spark
  - Large-scale logistic regression on vector-free LBFGS
- Advanced CTR estimation pipeline
  - Learn part of features from deep neural network

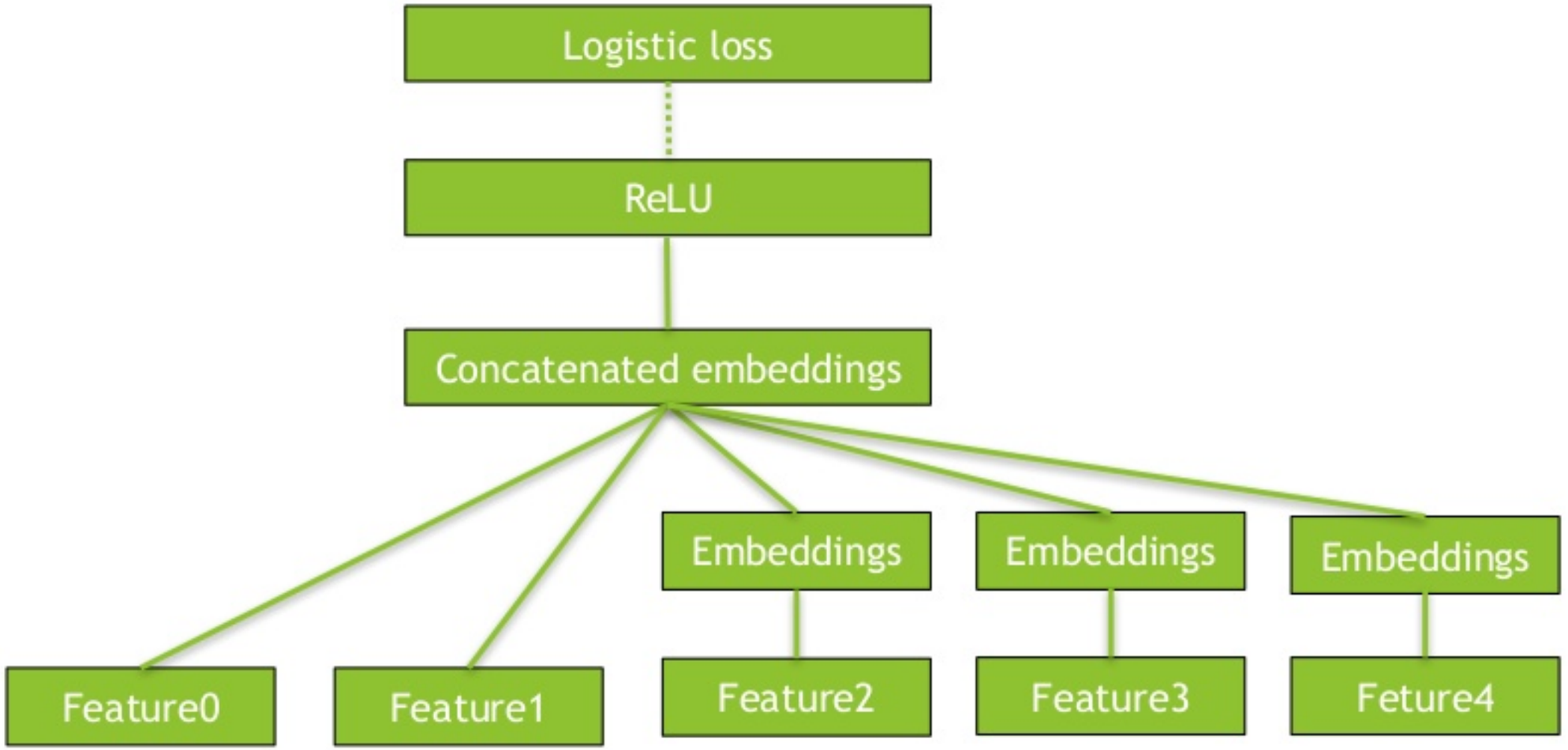
LR vs DNN


Train fast,
Predict fast
output
hidden layers
with activation
Train slowly,
Predict slowly
LR
input features
DNN

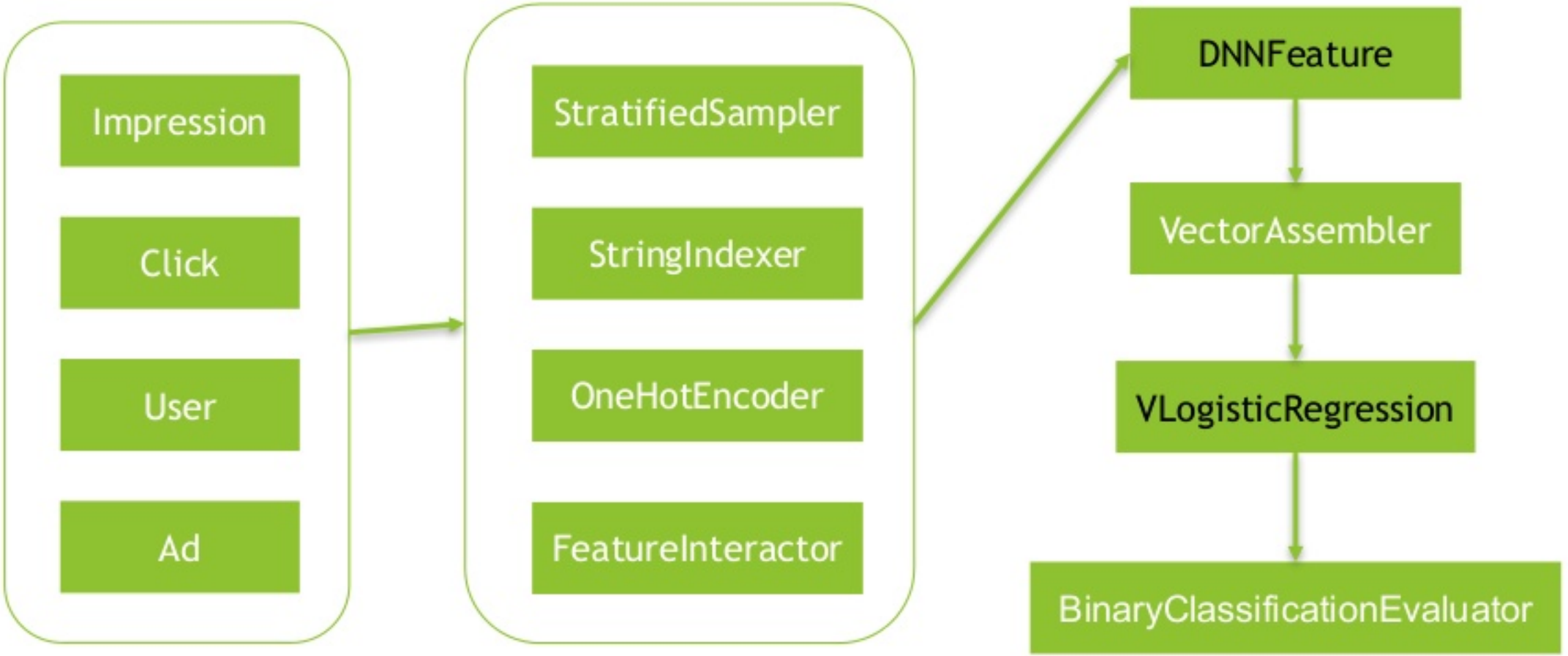
Updated solution



Learn feature from DNN



Advanced CTR estimation pipeline on Spark



Offline experiment - not online used

Model	System	AUC gain
LR	MLlib	0.000%
LR with more crosses	spark-vlbfgs	1.038%
LR with DNN features	spark-vlbfgs + TensorFlow	0.116%

Lessons learned

- DNN is promising to learn high-level representation directly from raw feature input.
- DNN feature learning reduces dependency to heavy physical work of hand-designed feature combination.
- Don’t feed all features into DNN, just those not changed frequently.
- Linear model is still useful.

Thank You.

yliang@apache.org