What is MLlib?

MLlib is a Spark subproject providing machine learning primitives:

- initial contribution from AMPLab, UC Berkeley
- shipped with Spark since version 0.8
- 33 contributors

What is MLlib?

Algorithms:

- **classification:** logistic regression, linear support vector machine (SVM), naive Bayes
- **regression:** generalized linear regression (GLM)
- **collaborative filtering:** alternating least squares (ALS)
- **clustering:** k-means
- **decomposition:** singular value decomposition (SVD), principal component analysis (PCA)

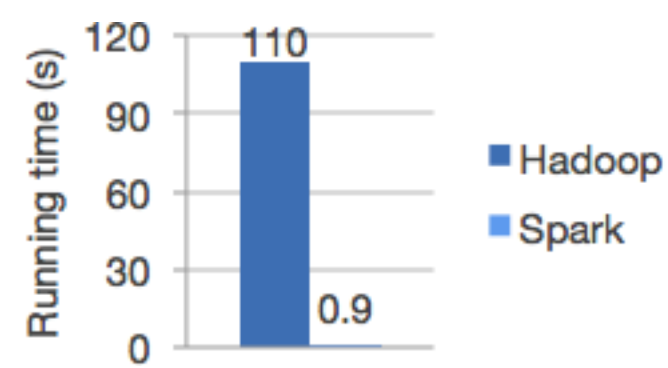
scikit-learn?

Algorithms:

- **classification:** SVM, nearest neighbors, random forest, …
- **regression:** support vector regression (SVR), ridge regression, Lasso, logistic regression, …
- **clustering:** k-means, spectral clustering, …
- **decomposition:** PCA, non-negative matrix factorization (NMF), independent component analysis (ICA), …

Why MLlib?

- It is built on Apache Spark, a fast and general engine for large-scale data processing.
  - Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.



  - Write applications quickly in Java, Scala, or Python.

Gradient descent

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

k-means (scala)

```
// Load and parse the data.
val data = sc.textFile("kmeans_data.txt")
val parsedData = data.map(_.split(' ').map(_.toDouble)).cache()

// Cluster the data into two classes using KMeans.
val clusters = KMeans.train(parsedData, 2, numIterations = 20)

// Compute the sum of squared errors.
val cost = clusters.computeCost(parsedData)
println("Sum of squared errors = " + cost)
```

Dimension reduction + k-means

```
// compute principal components
val points: RDD[Vector] = ...
val mat = RowRDDMatrix(points)
val pc = mat.computePrincipalComponents(20)

// project points to a low-dimensional space
val projected = mat.multiply(pc).rows

// train a k-means model on the projected data
val model = KMeans.train(projected, 10)
```

Collaborative filtering

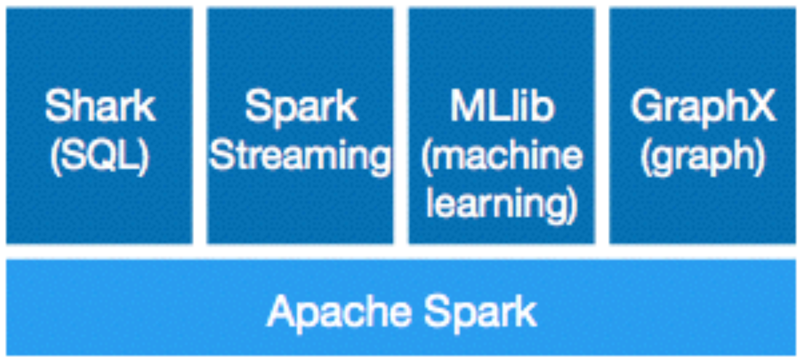
```scala
// Load and parse the data
val data = sc.textFile("mllib/data/als/test.data")
val ratings = data.map(_.split(',') match {
    case Array(user, item, rate) =>
      Rating(user.toInt, item.toInt, rate.toDouble)
})

// Build the recommendation model using ALS
val model = ALS.train(ratings, 1, 20, 0.01)

// Evaluate the model on rating data
val usersProducts = ratings.map { case Rating(user, product, rate) =>
  (user, product)
}
val predictions = model.predict(usersProducts)
```

Why MLlib?

- It ships with Spark as a standard component.



Why MLlib?

A special-purpose device may be better at one aspect than a general-purpose device. But the cost of context switching is high:

- different languages or APIs
- different data formats
- different tuning tricks

Spark SQL + MLlib

```scala
// Data can easily be extracted from existing sources,
// such as Apache Hive.
val trainingTable = sql("""
  SELECT e.action,
         u.age,
         u.latitude,
         u.longitude
  FROM Users u
  JOIN Events e
  ON u.userId = e.userId""")

// Since `sql` returns an RDD, the results of the above
// query can be easily used in MLlib.
val training = trainingTable.map { row =>
  val features = Vectors.dense(row(1), row(2), row(3))
  LabeledPoint(row(0), features)
}

val model = SVMWithSGD.train(training)
```

Streaming + MLlib

```
// collect tweets using streaming

// train a k-means model
val model: KMmeansModel = ...

// apply model to filter tweets
val tweets = TwitterUtils.createStream(ssc, Some(authorizations(0)))
val statuses = tweets.map(_.getText)
val filteredTweets =
  statuses.filter(t => model.predict(featurize(t)) == clusterNumber)

// print tweets within this particular cluster
filteredTweets.print()
```

GraphX + MLlib

```
// assemble link graph
val graph = Graph(pages, links)
val pageRank: RDD[(Long, Double)] = graph.staticPageRank(10).vertices

// load page labels (spam or not) and content features
val labelAndFeatures: RDD[(Long, (Double, Seq((Int, Double)))] = ...
val training: RDD[LabeledPoint] =
  labelAndFeatures.join(pageRank).map {
    case (id, ((label, features), pageRank)) =>
      LabeledPoint(label, Vectors.sparse(features ++ (1000, pageRank))
}

// train a spam detector using logistic regression
val model = LogisticRegressionWithSGD.train(training)
```

Why MLlib?

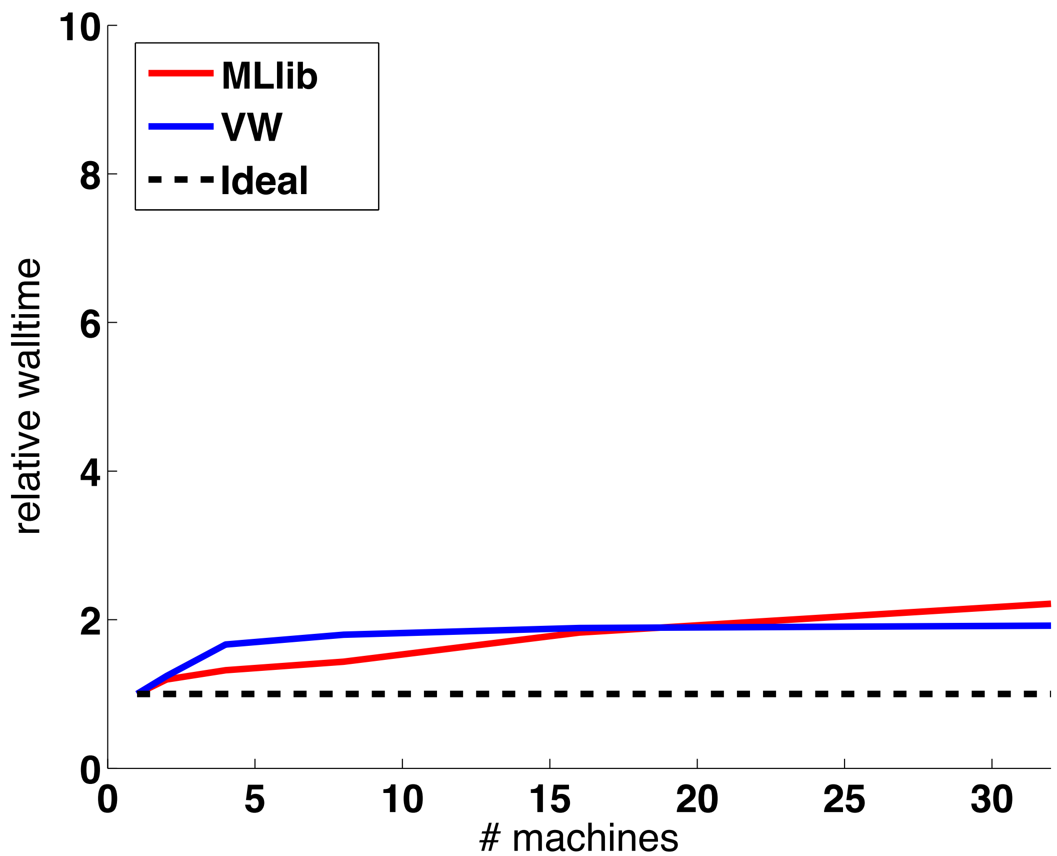
- Spark is a general-purpose big data platform.
  - Runs in standalone mode, on YARN, EC2, and Mesos, also on Hadoop v1 with SIMR.
  - Reads from HDFS, S3, HBase, and any Hadoop data source.
- MLlib is a standard component of Spark providing machine learning primitives on top of Spark.
- <u>MLlib is also comparable to or even better than other libraries specialized in large-scale machine learning.</u>

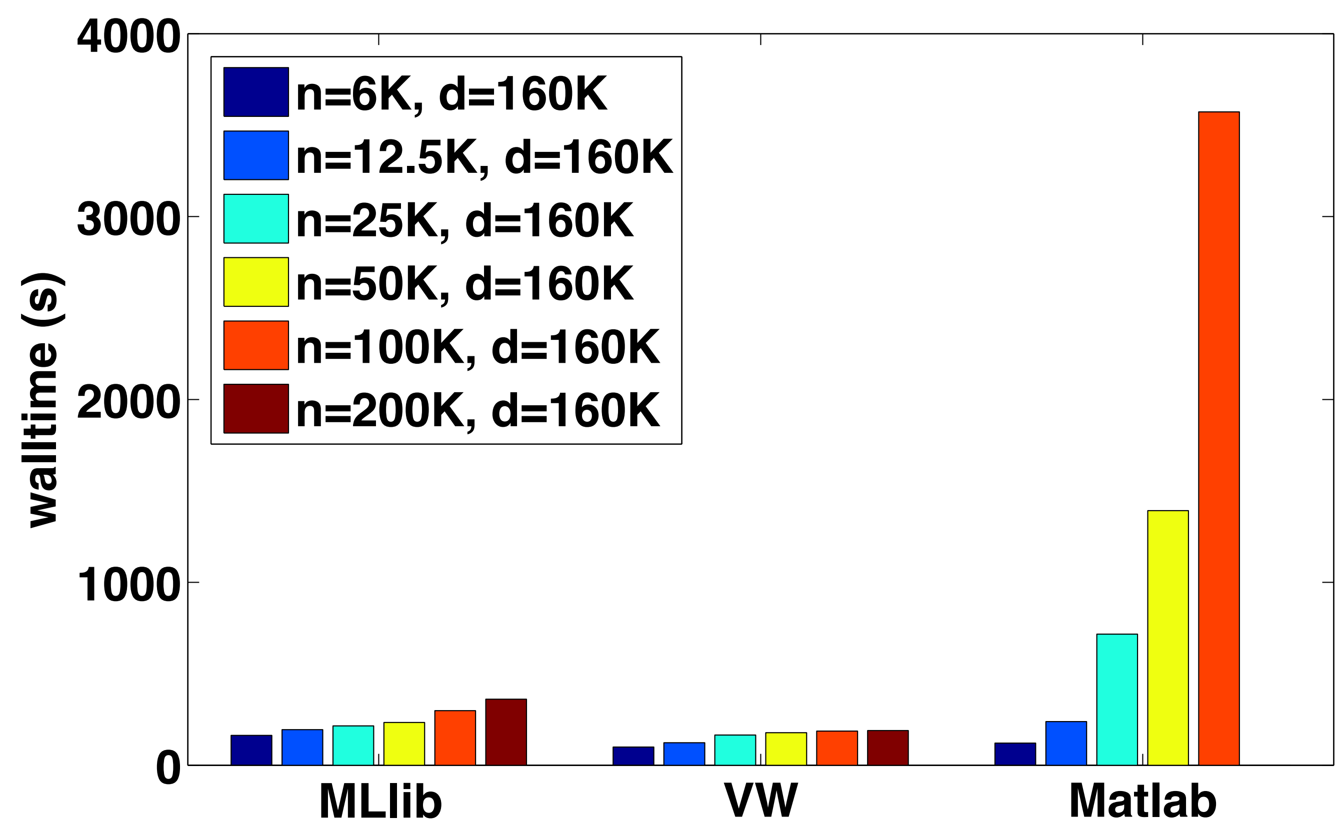
Why MLlib?

- Scalability
- Performance
- User-friendly APIs
- Integration with Spark and its other components

Logistic regression

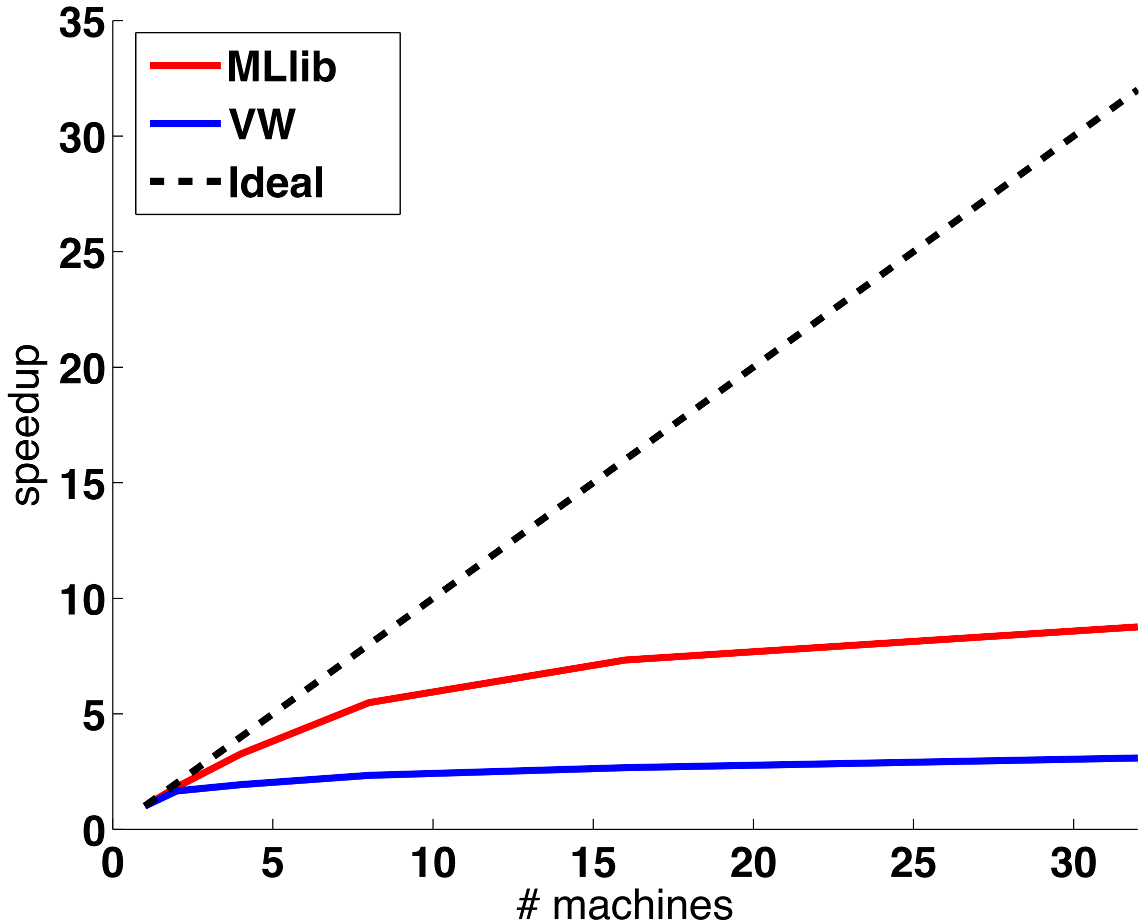
Logistic regression - weak scaling

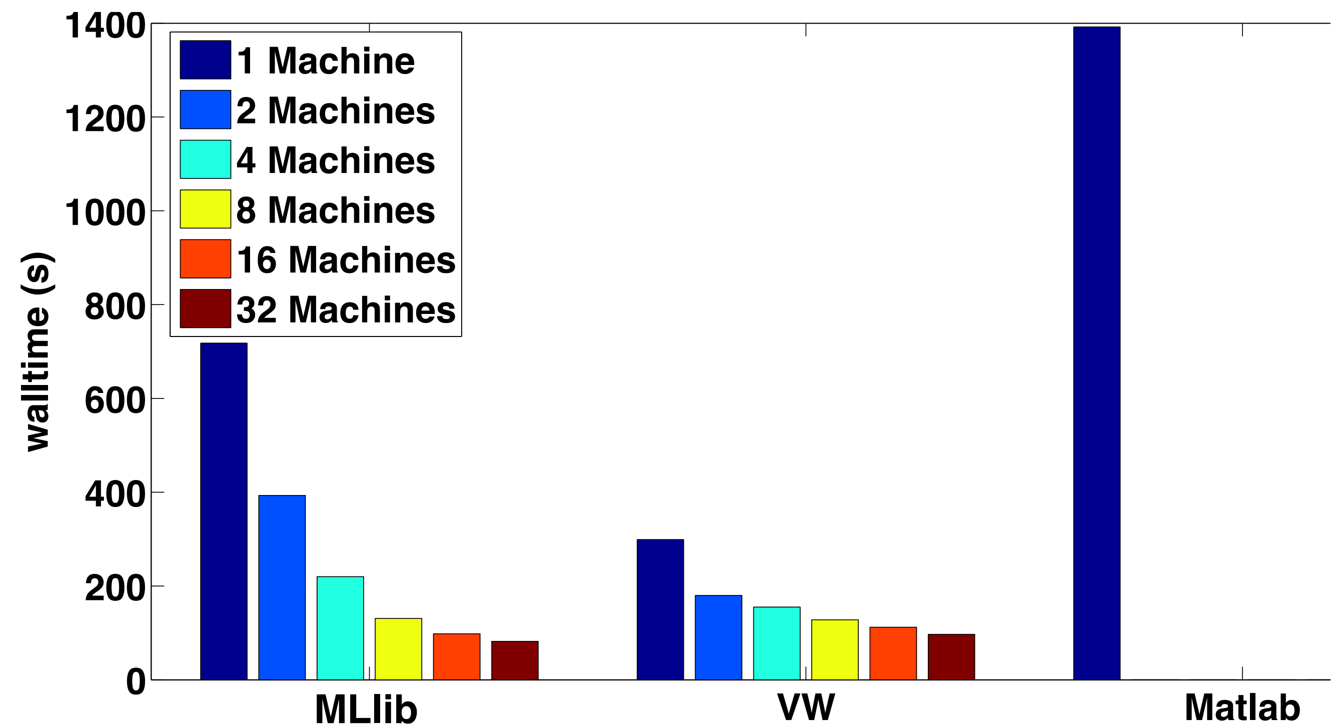




- Full dataset: 200K images, 160K dense features.
- Similar weak scaling.
- MLlib within a factor of 2 of VW's wall-clock time.

Logistic regression - strong scaling





- Fixed Dataset: 50K images, 160K dense features.
- MLlib exhibits better scaling properties.
- MLlib is faster than VW with 16 and 32 machines.

Collaborative filtering

Collaborative filtering

☆	☆☆☆☆	?
☆	☆☆☆	☆☆
☆☆☆☆	?	☆
☆	?	☆☆
?	☆☆☆	☆☆
☆☆☆☆	☆☆	?

- Recover a rating matrix from a subset of its entries.

ALS - wall-clock time

System	Wall-clock time (seconds)
MATLAB	15443
Mahout	4206
GraphLab	291
MLlib	481

- Dataset: scaled version of Netflix data (9X in size).
- Cluster: 9 machines.
- MLlib is an order of magnitude faster than Mahout.
- MLlib is within factor of 2 of GraphLab.

Implementation of k-means

Initialization:

- random
- k-means++
- k-means||

Implementation of k-means

Iterations:

- For each point, find its closest center.

$$l_i = \arg\min_j \|x_i - c_j\|_2^2$$

- Update cluster centers.

$$c_j = \frac{\sum_{i,l_i=j} x_j}{\sum_{i,l_i=j} 1}$$

Implementation of k-means

The points are usually sparse, but the centers are most likely to be dense. Computing the distance takes O(d) time. So the time complexity is O(n d k) per iteration. We don't take any advantage of sparsity on the running time. However, we have

$$\|x - c\|_2^2 = \|x\|_2^2 + \|c\|_2^2 - 2\langle x, c\rangle$$

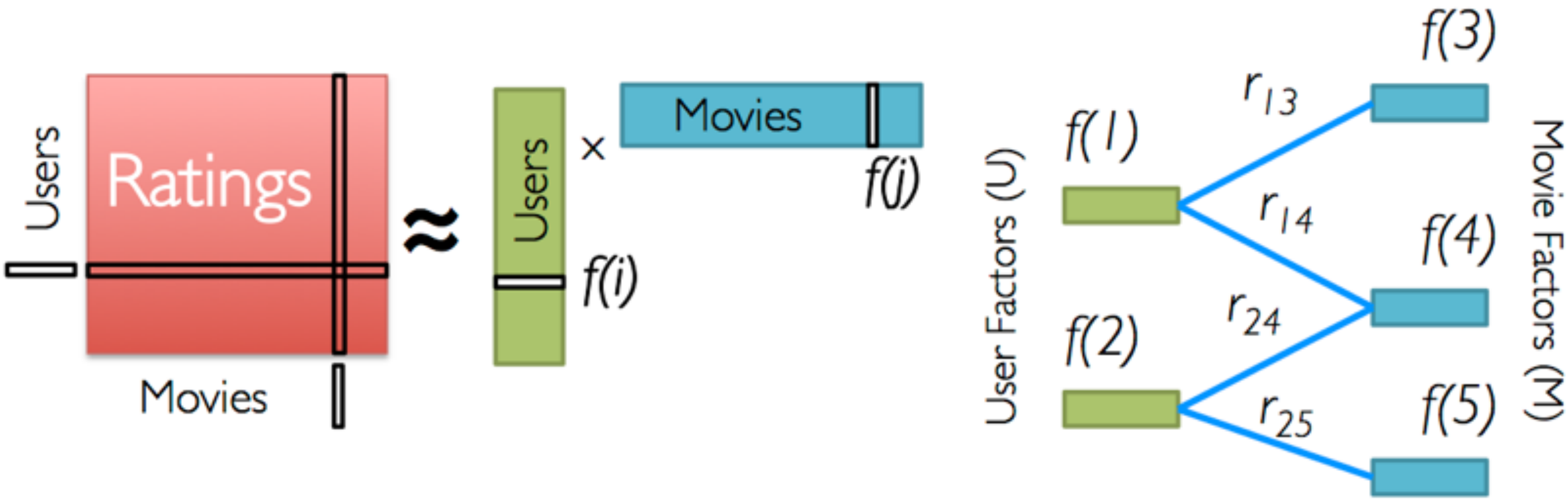
Computing the inner product only needs non-zero elements. So we can cache the norms of the points and of the centers, and then only need the inner products to obtain the distances. This reduce the running time to O(nnz k + d k) per iteration.

However, is it accurate?

Implementation of ALS

- broadcast everything
- data parallel
- fully parallel

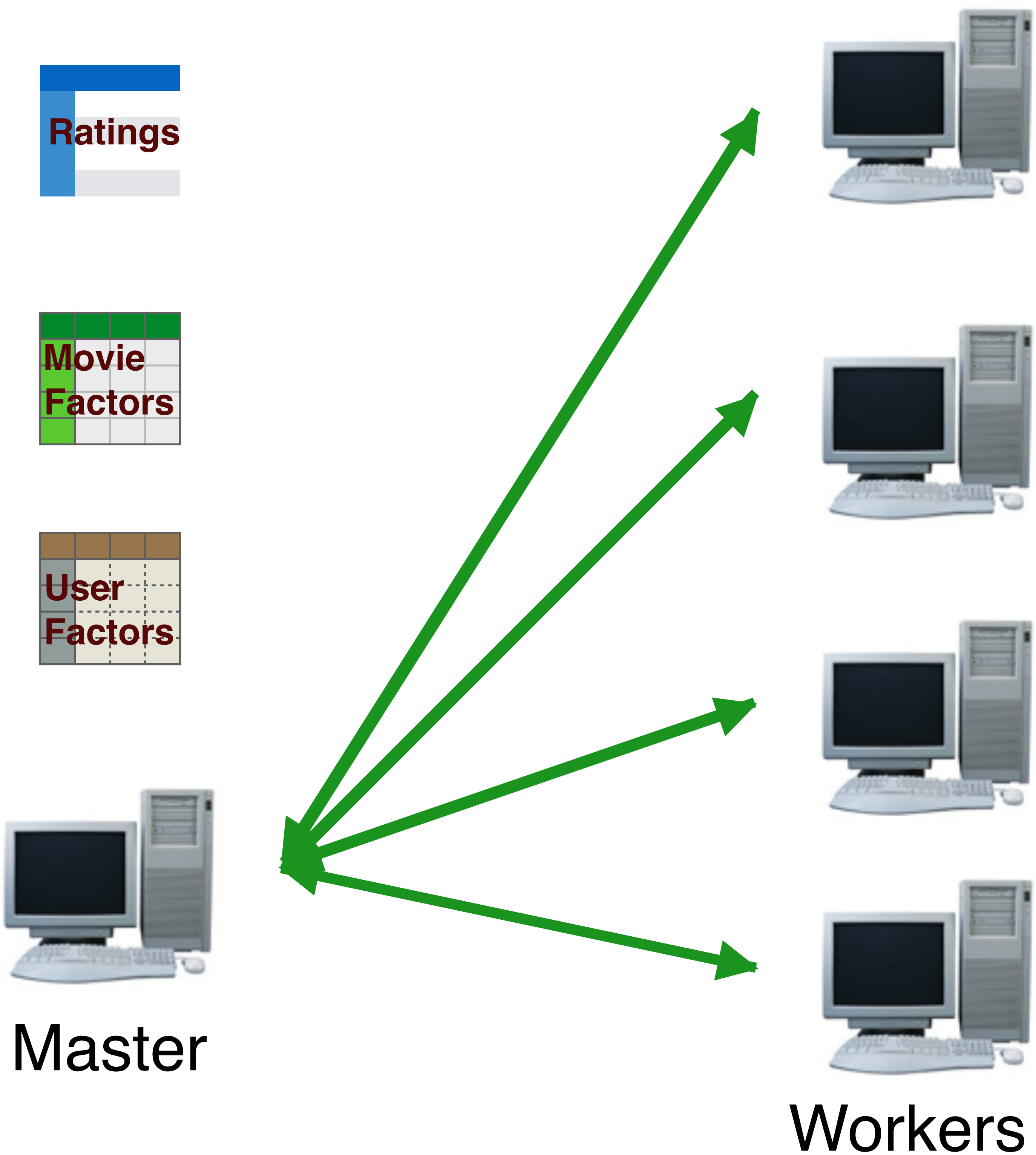
Alternating least squares (ALS)



Iterate:

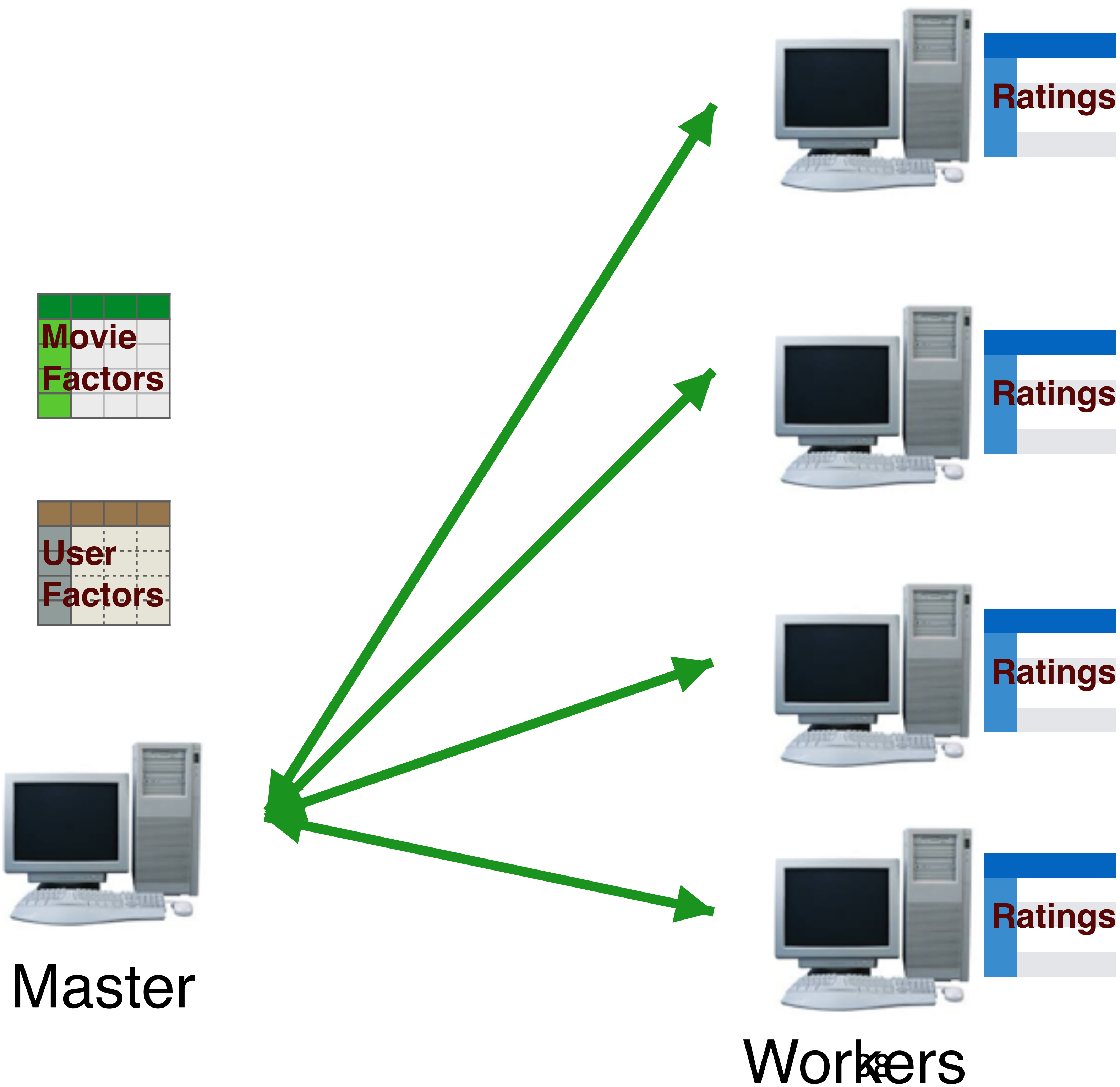
$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Broadcast everything



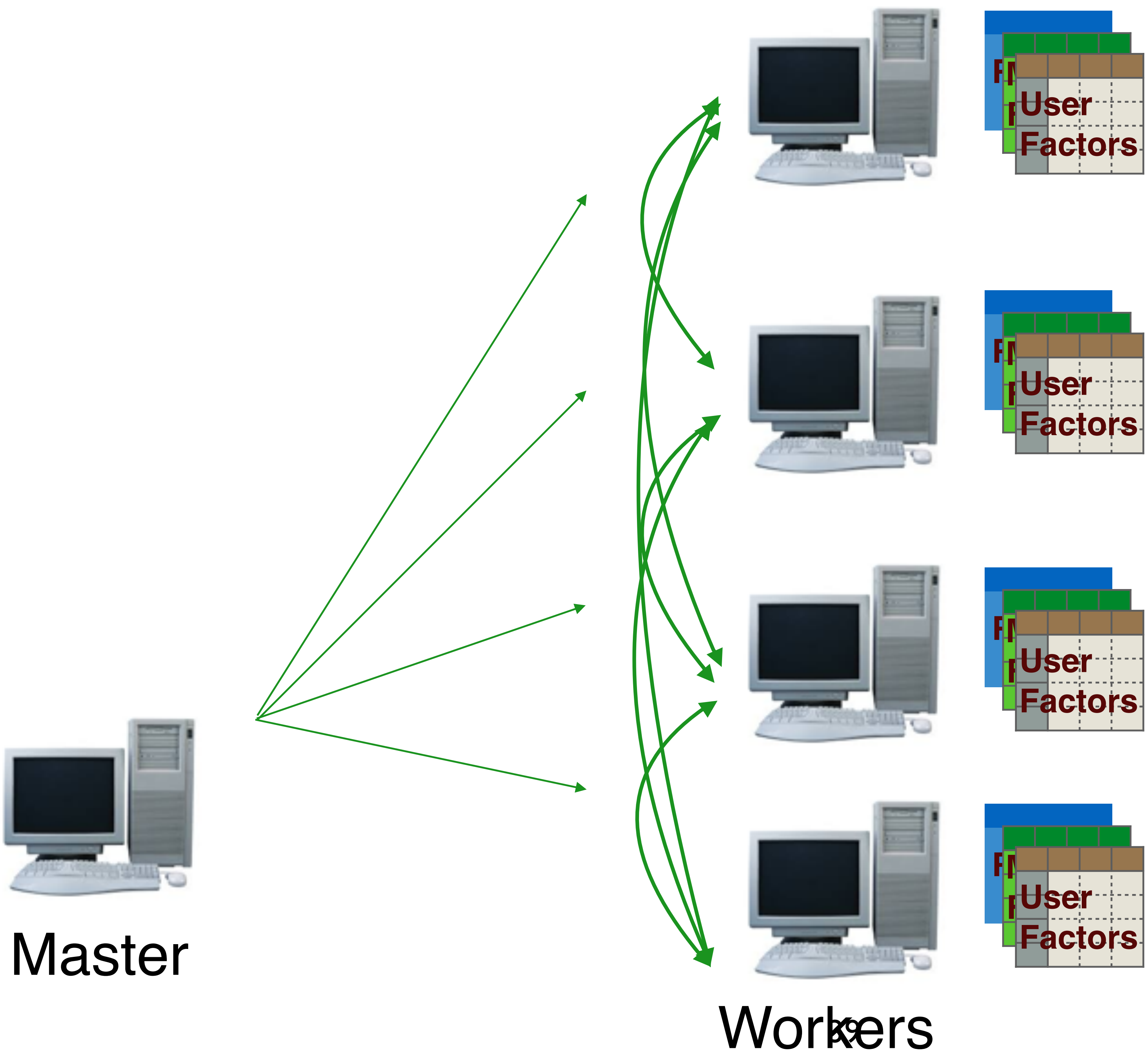
- Master loads (small) data file and initializes models.
- Master broadcasts data and initial models.
- At each iteration, updated models are broadcast again.
- Works OK for small data.
- Lots of communication overhead - doesn't scale well.

Data parallel



- Workers load data
- Master broadcasts initial models
- At each iteration, updated models are broadcast again
- Much better scaling
- Works on large datasets
- Works well for smaller models. (low K)

Fully parallel



- Workers load data
- Models are instantiated at workers.
- At each iteration, models are shared via join between workers.
- Much better scalability.
- Works on large datasets

Implementation of ALS

- ~~broadcast everything~~
- ~~data parallel~~
- ~~fully parallel~~
- block-wise parallel
    - Users/products are partitioned into blocks and join is based on blocks instead of individual user/product.

New features for v1.x

- Sparse data
- Classification and regression tree (CART)
- SVD and PCA
- L-BFGS
- Model evaluation
- Discretization