Building Streaming Recommendation Engines on Spark

Rui Vieira

rcardoso@redhat.com

Overview

- Collaborative Filtering
    - Batch Alternating Least Squares (ALS)
    - Streaming ALS
- Apache Spark
    - Distributed Streaming ALS
- OpenShift deployment

Collaborative Filtering

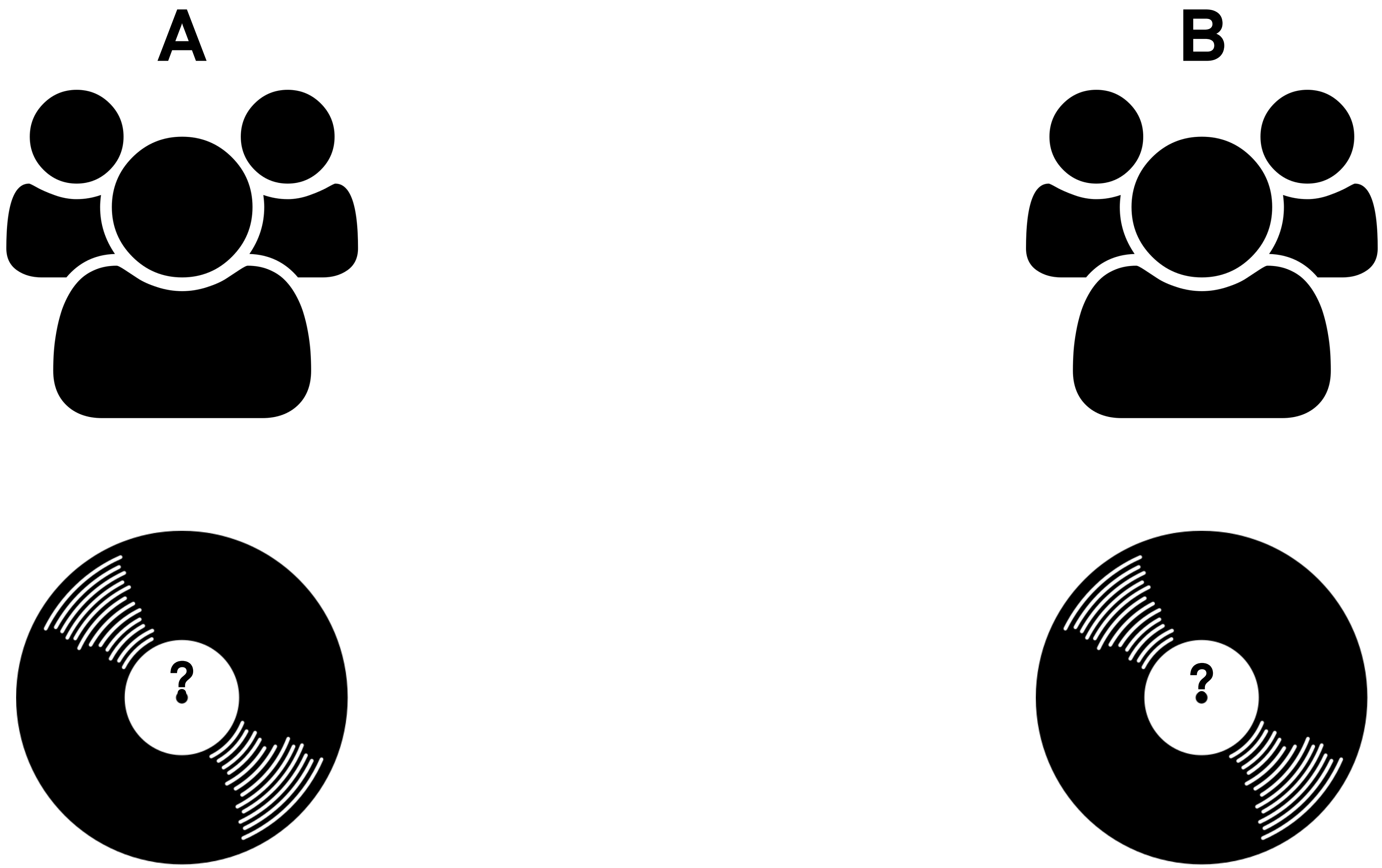
- Users, products and ratings

$$(\text{user}, \text{product}) \rightarrow \text{rating}$$

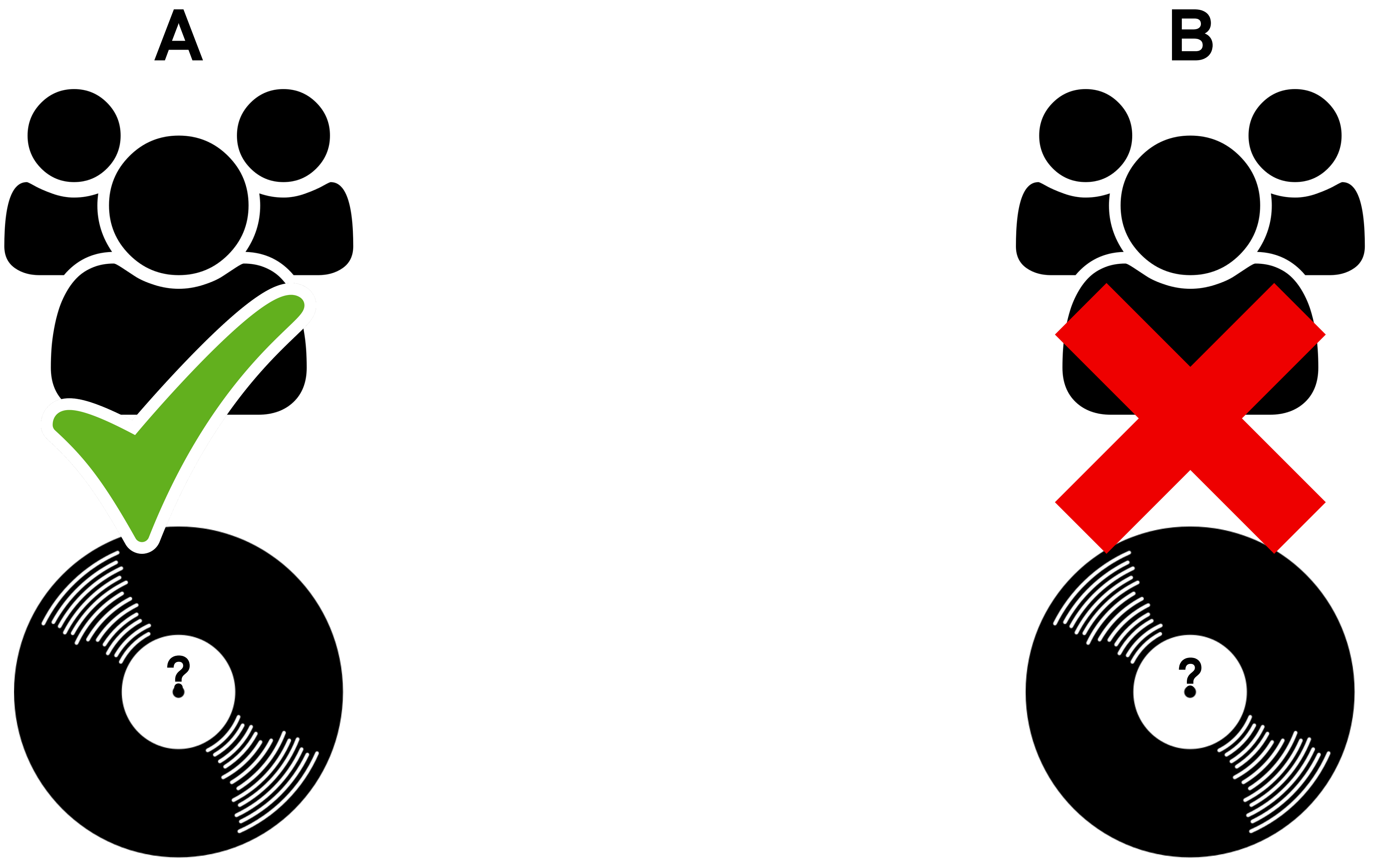
- Collaborative

- “Filtering”

Collaborative Filtering



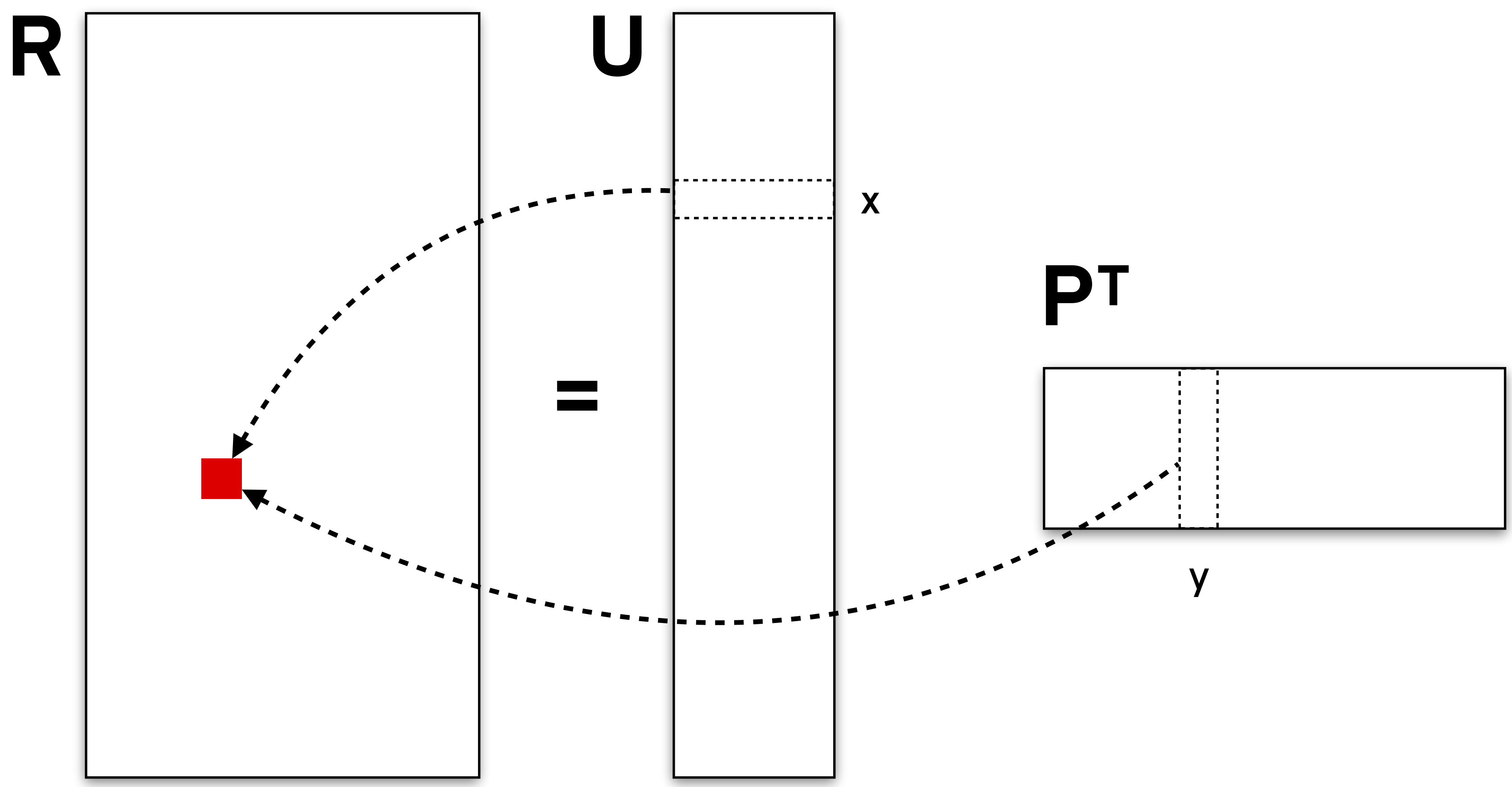
Collaborative Filtering



Alternating Least Squares

$$
R = \begin{array}{c c}
\begin{array}{ccccc} \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user N} \end{array} & \\
\begin{bmatrix}
1 & 4.5 & ? & \cdots & 3 \\
? & 3 & 3 & \cdots & 4 \\
5 & 3 & ? & \cdots & ? \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
2 & 4 & 1 & \cdots & ?
\end{bmatrix}
&
\begin{array}{c} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array}
\end{array}
$$

Alternating Least Squares



$$\hat{r}_{x,y} = \mathsf{U}_x \mathsf{P}_y^T$$

Batch ALS

$$\mathsf{loss} = \sum_{x,y} \Big(\underbrace{r_{x,y} - \hat{r}_{x,y}}_{\epsilon_{x,y}} \Big)^2 + \lambda_x \sum_x \|\mathsf{U}_x\|^2 + \lambda_y \sum_y \|\mathsf{P}_y\|^2$$

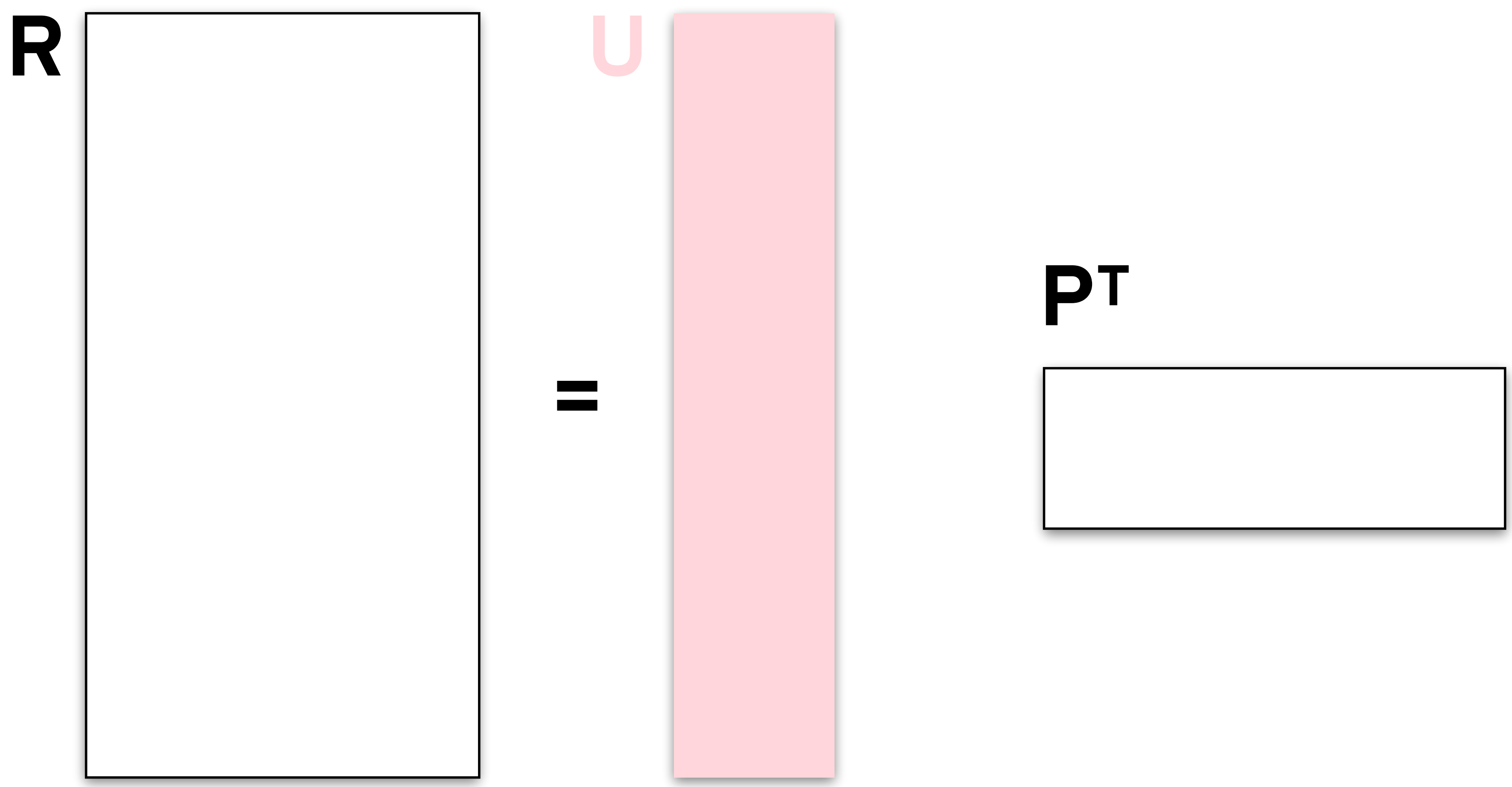
Batch ALS

$$\text{loss} = \sum_{x,y} \Big(\underbrace{r_{x,y} - \hat{r}_{x,y}}_{\epsilon_{x,y}} \Big)^2 + \lambda_x \sum_x \|\mathsf{U}_x\|^2 + \lambda_y \sum_y \|\mathsf{P}_y\|^2$$

(minimize)

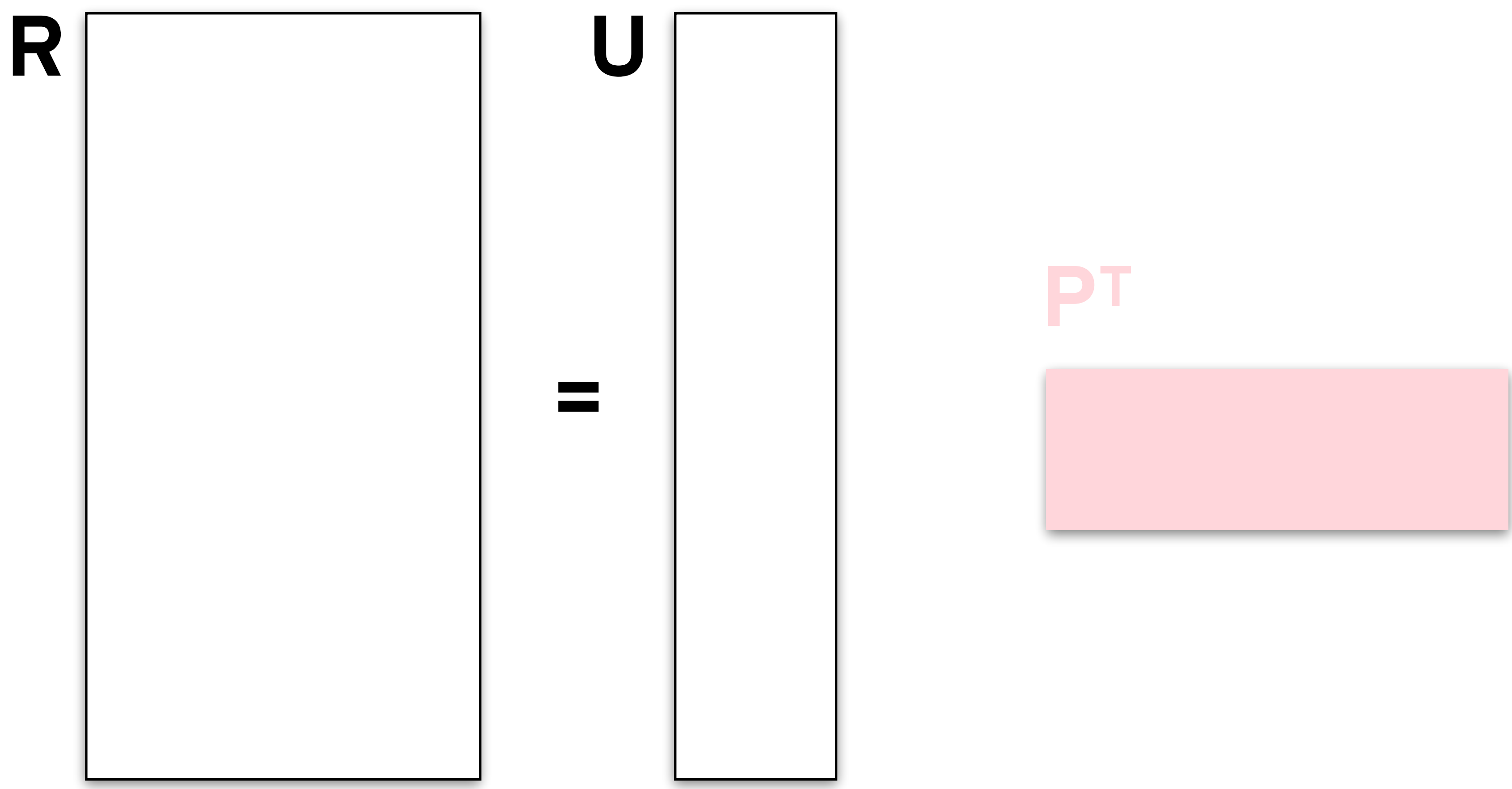
$$\frac{\partial \text{loss}}{\partial \mathsf{U}_x} = 0, \qquad \frac{\partial \text{loss}}{\partial \mathsf{P}_y} = 0$$

Alternating Least Squares



$$\mathsf{P}_y = r_y \mathsf{X} \left(X^T X + \lambda_y \mathsf{I} \right)^{-1}$$

Alternating Least Squares



$$\mathsf{U}_x = r_x \mathsf{Y}\left(Y^T Y + \lambda_x \mathsf{I}\right)^{-1}$$

Alternating Least Squares

$$
\begin{array}{c@{}c@{}l}
 & \begin{array}{ccccc} \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user N} \end{array} & \\
R = & \left[\begin{array}{ccccc} 1 & 4.5 & \color{red}{3.8} & \cdots & 3 \\ \color{red}{3.2} & 3 & 3 & \cdots & 4 \\ 5 & 3 & \color{red}{3.4} & \cdots & \color{red}{3.1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \cdots & \color{red}{2.7} \end{array}\right] & \begin{array}{c} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array}
\end{array}
$$

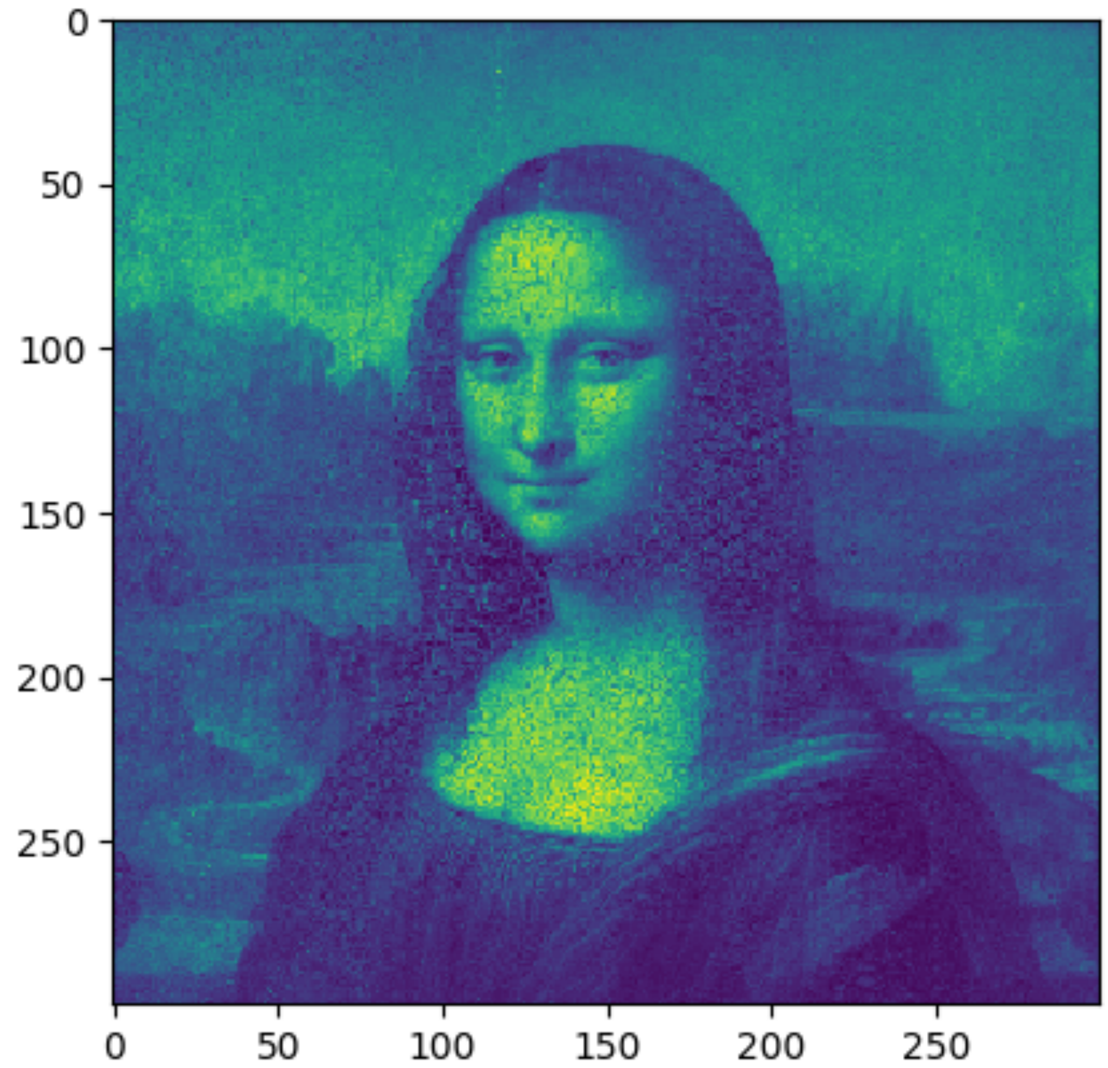
Batch ALS

	1	2	3	4	… 300
1	70	82	60	54	65
2	70	86	68	67	72
3	96	103	82	82	77
4	90	87	68	93	82
… 300	38	48	44	51	35

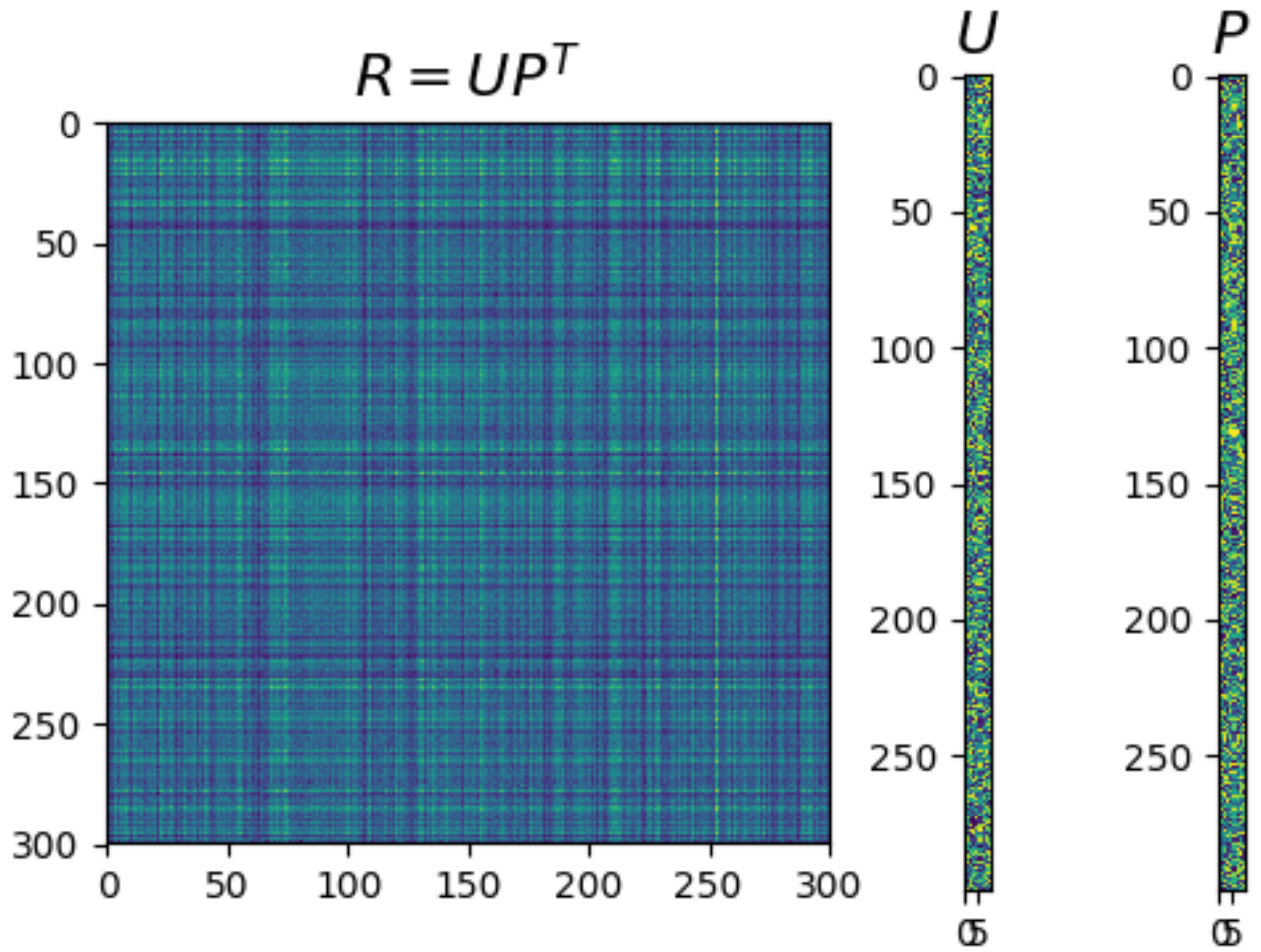
Batch ALS

	1	2	3	4	... 300
1	70	82	60	54	65
2	70	86	68	67	72
3	96	103	82	82	77
4	90	87	68	93	82
... 300	38	48	44	51	35

Batch ALS

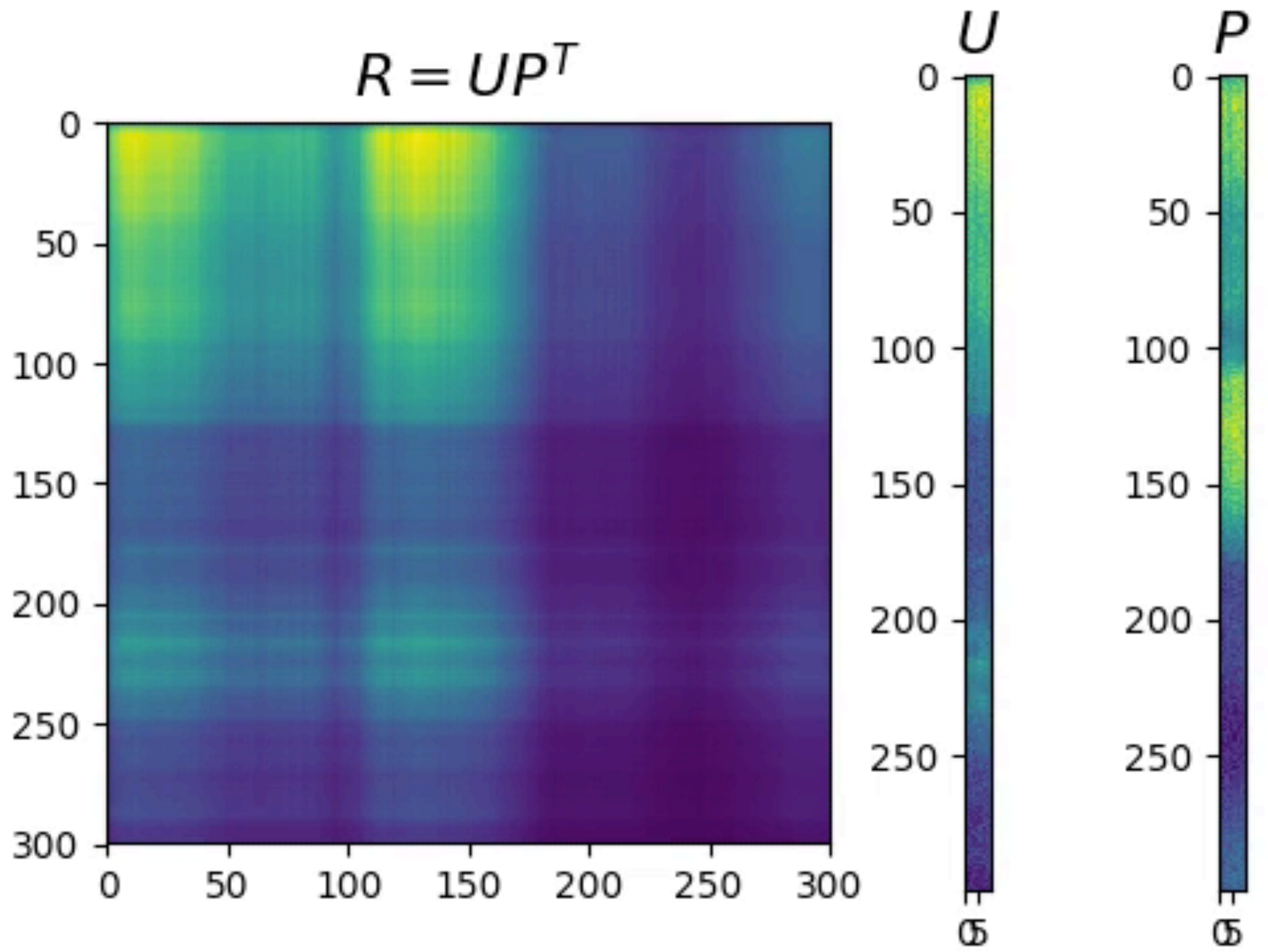


Batch ALS



$R = UP^T$
0
50
100
150
200
250
300
0
50
100
150
200
250
300
U
0
50
100
150
200
250
0
5
P
0
50
100
150
200
250
0
5

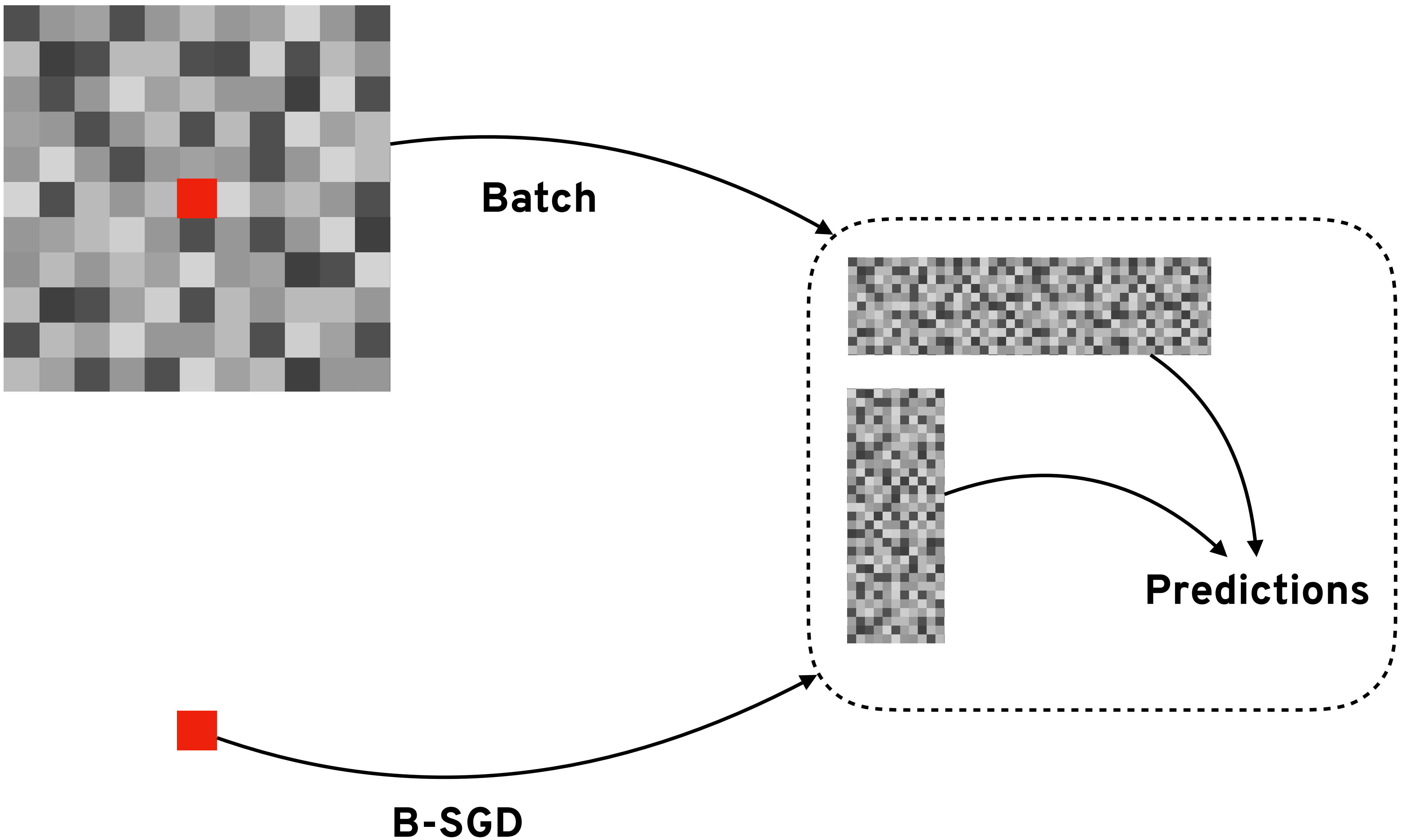
Batch ALS



Streaming ALS

- Can we update the model with a data stream?
- Stochastic Gradient Descent (SGD)
    - Bias SGD (B-SGD)

Streaming ALS



Streaming ALS

$$b_{x,y} = \mu + b_x + b_y$$

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

Streaming ALS

$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

$$\textsf{loss} = \sum_{x,y} \left(\underbrace{r_{x,y} - \hat{r}_{x,y}}_{\epsilon_{x,y}} \right)^2 + \cdots$$

Streaming ALS

bias

$$b_x \leftarrow b_x + \gamma \left(\epsilon_{x,y} - \lambda_x b_x\right)$$

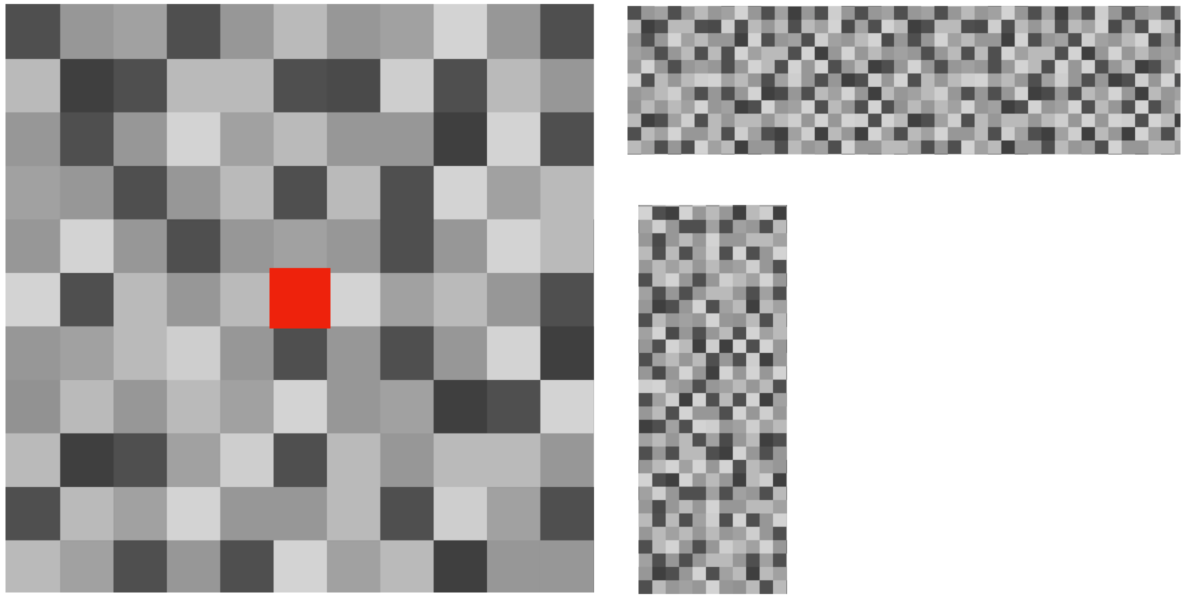
$$b_y \leftarrow b_y + \gamma \left(\epsilon_{x,y} - \lambda_y b_y\right)$$

factors

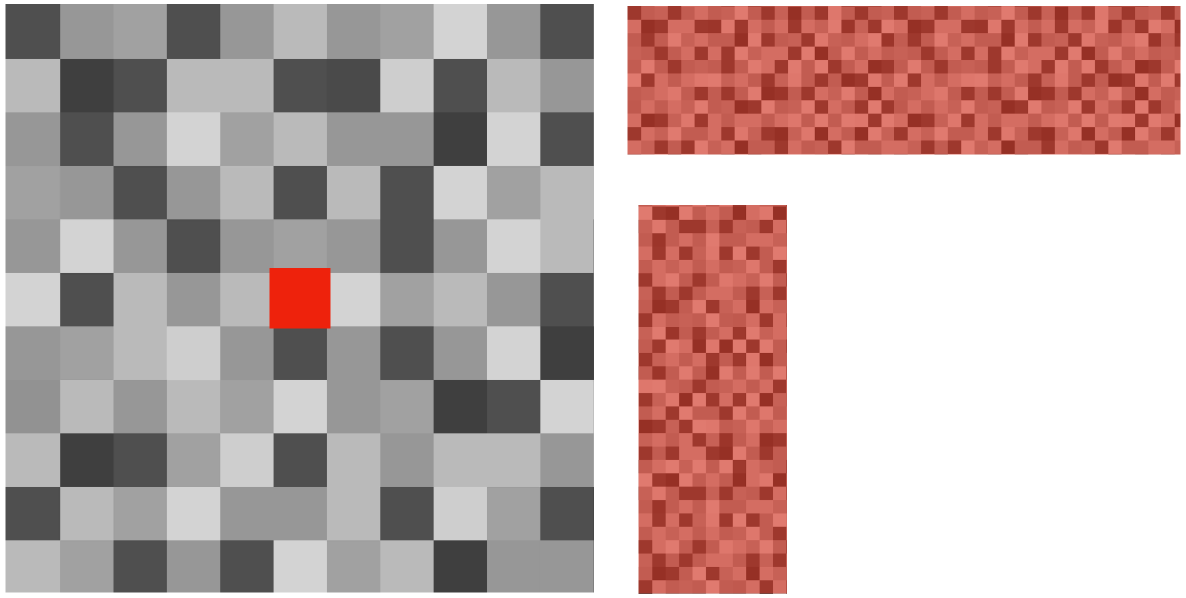
$$\mathsf{U}_x \leftarrow \mathsf{U}_x + \gamma \left(\epsilon_{x,y}\mathsf{P}_y - \lambda'_x \mathsf{U}_x\right)$$

$$\mathsf{P}_y \leftarrow \mathsf{P}_y + \gamma \left(\epsilon_{x,y}\mathsf{U}_x - \lambda'_y \mathsf{P}_y\right)$$

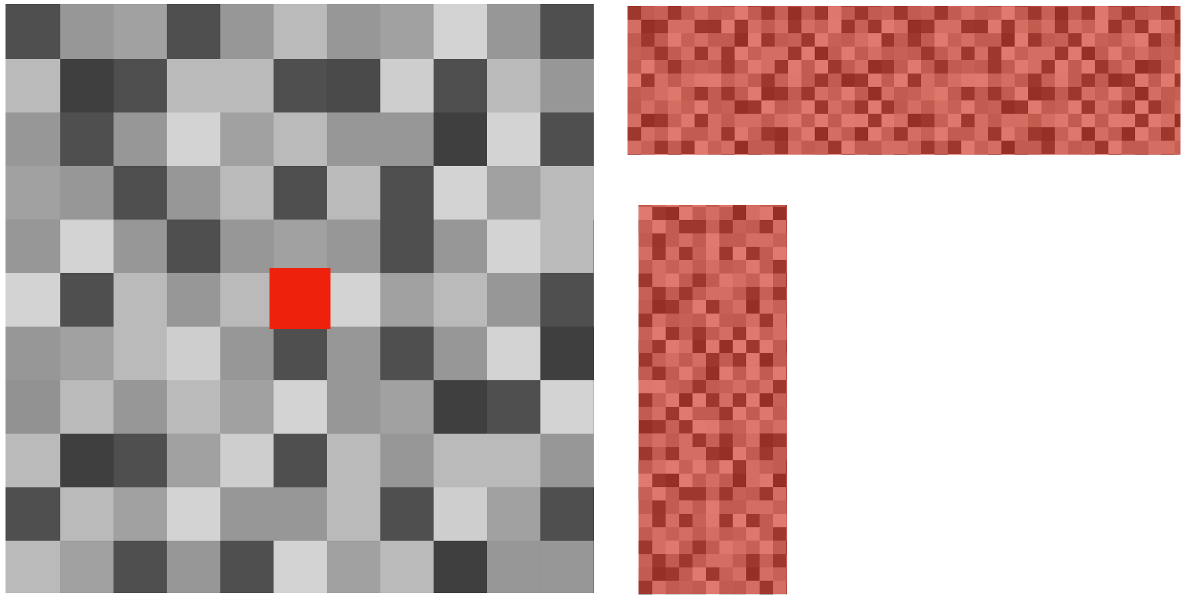
Streaming ALS

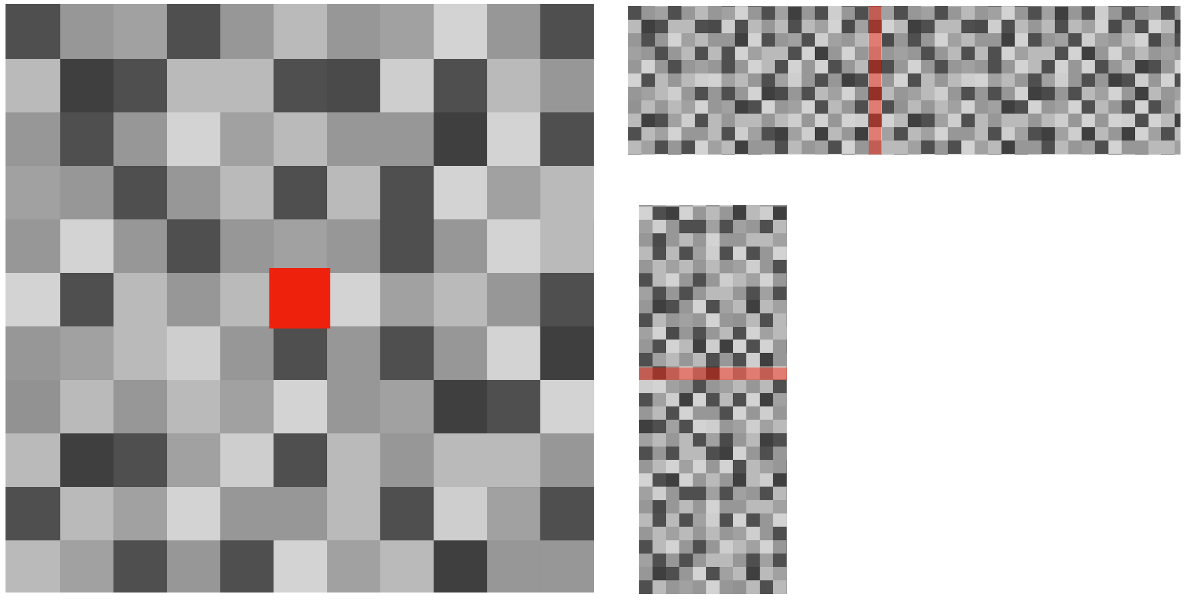


Streaming ALS

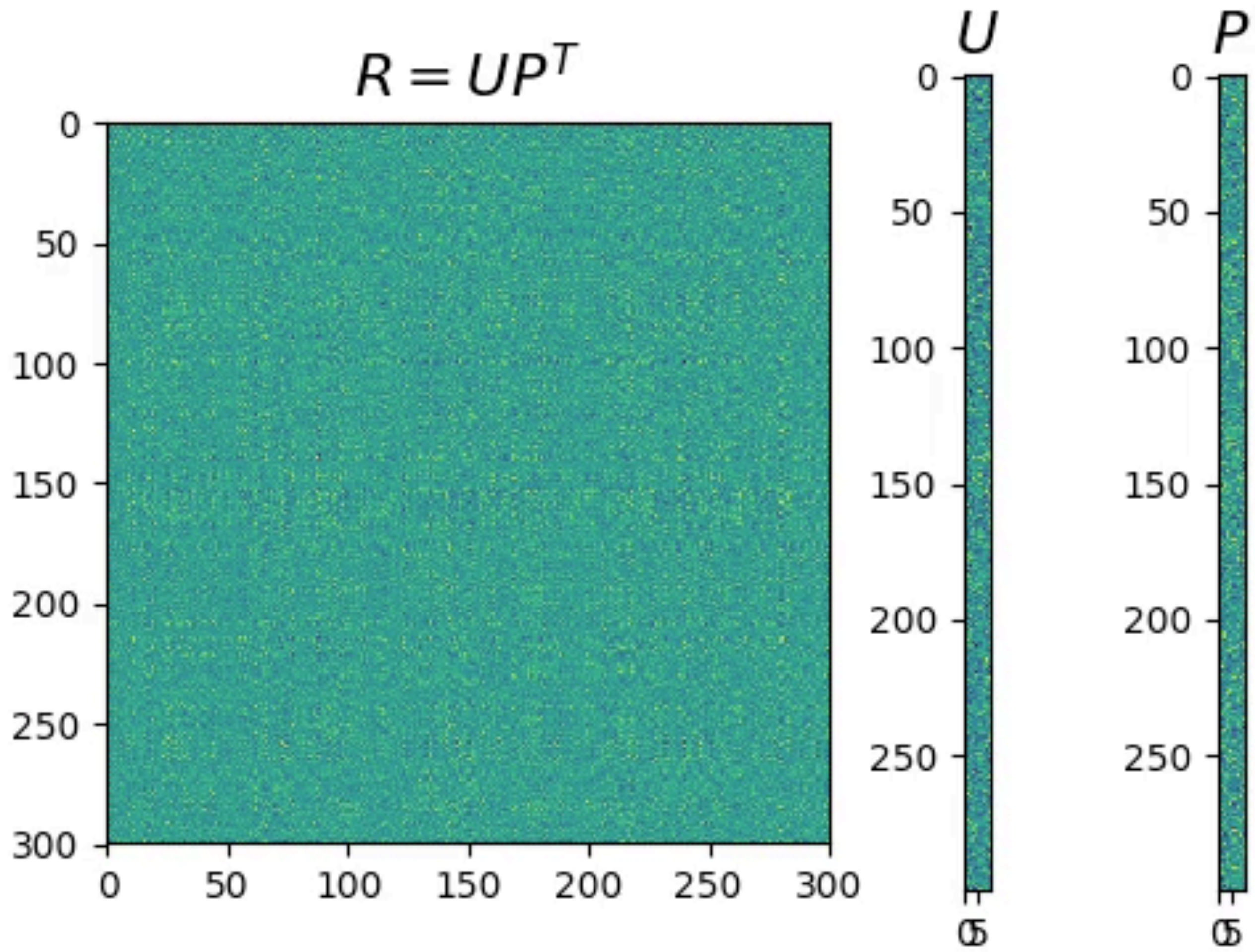


Streaming ALS



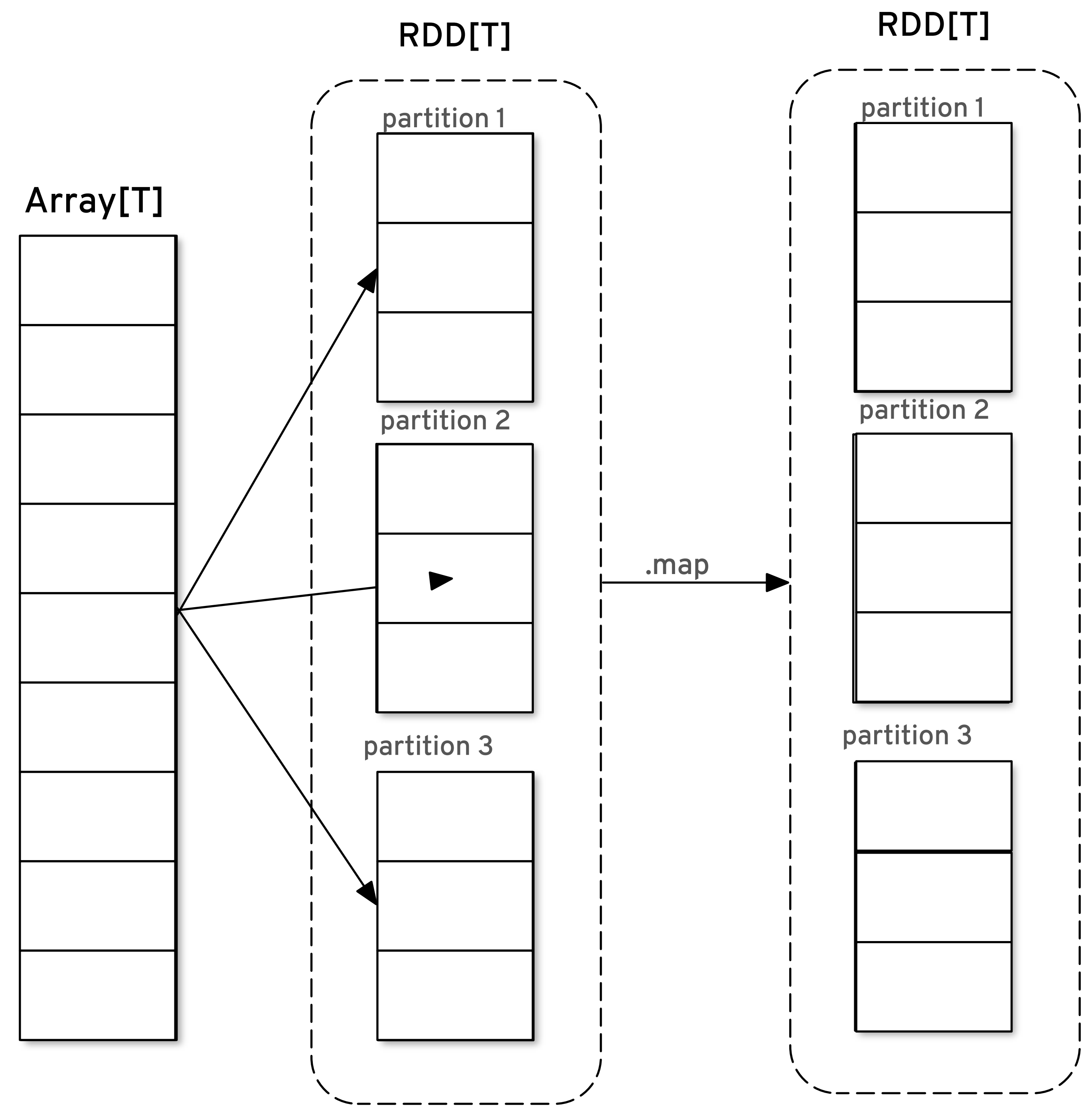


Streaming ALS



Apache Spark

Apache Spark



MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)

val ratings: RDD[Rating]
```

MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)

val ratings: RDD[Rating]

val rank: int
```

MLlib ALS

```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)

val ratings: RDD[Rating]

val rank: int

val iterations: int
```

MLlib ALS

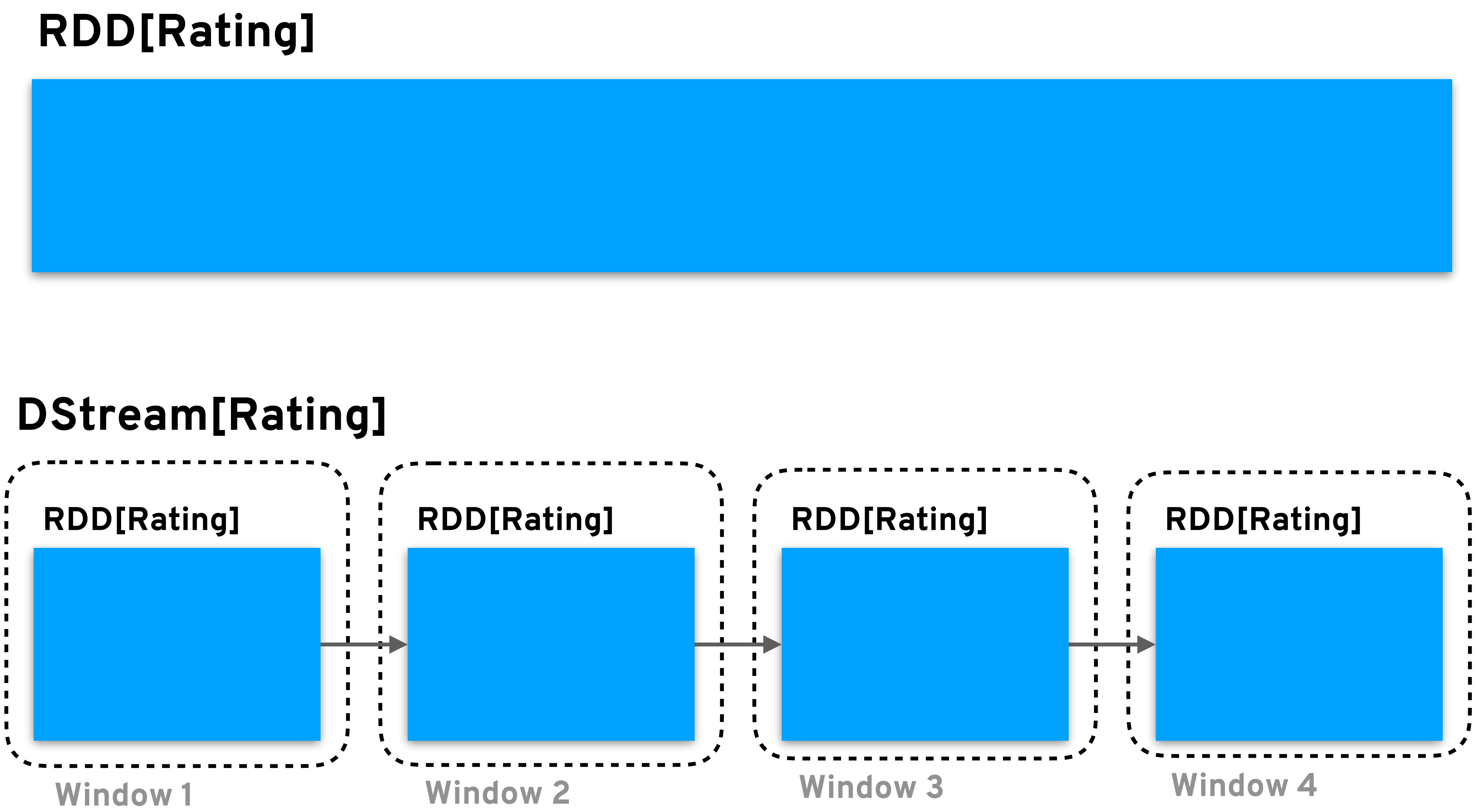
```
val model = ALS.train(ratings, rank, iterations, lambda)
```

```
case class Rating(int user, int product, double rating)

val ratings: RDD[Rating]

val rank: int

val iterations: int

val lambda: Double
```

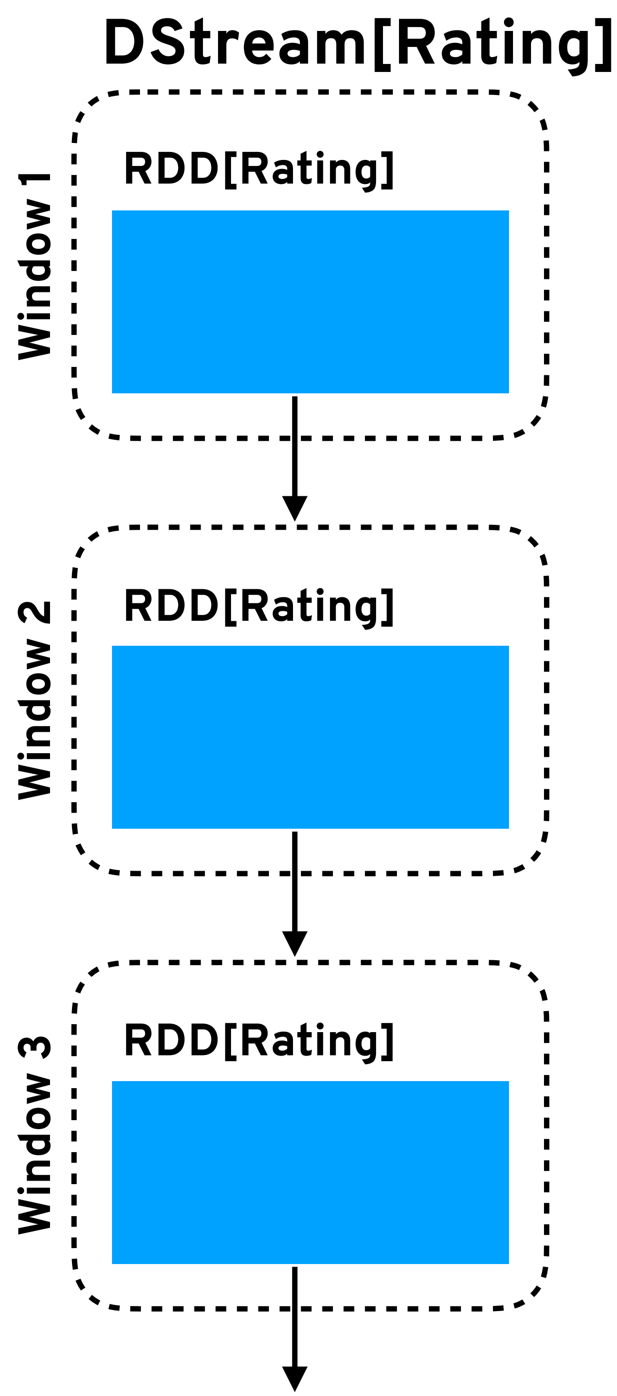
MLlib ALS

```
> val model = ALS.train(ratings, rank, iterations, lambda)

model: MatrixFactorizationModel

class MatrixFactorizationModel {

   val userFeatures: RDD[(Int, Array[Double])]
   val productFeatures: RDD[(Int, Array[Double])]

}
```

Spark Streaming ALS



Spark Streaming ALS




```
model = StreamingALS.train(rdd1, params)

model = model.train(rdd2)

model = model.train(rdd3)
```

Spark Streaming ALS

```
userBias += gamma * (error - lambda * userBias)

userFeature(i) += gamma * (error *  prodFeature(j) - lambda * userFeature(i))
```

Spark Streaming ALS

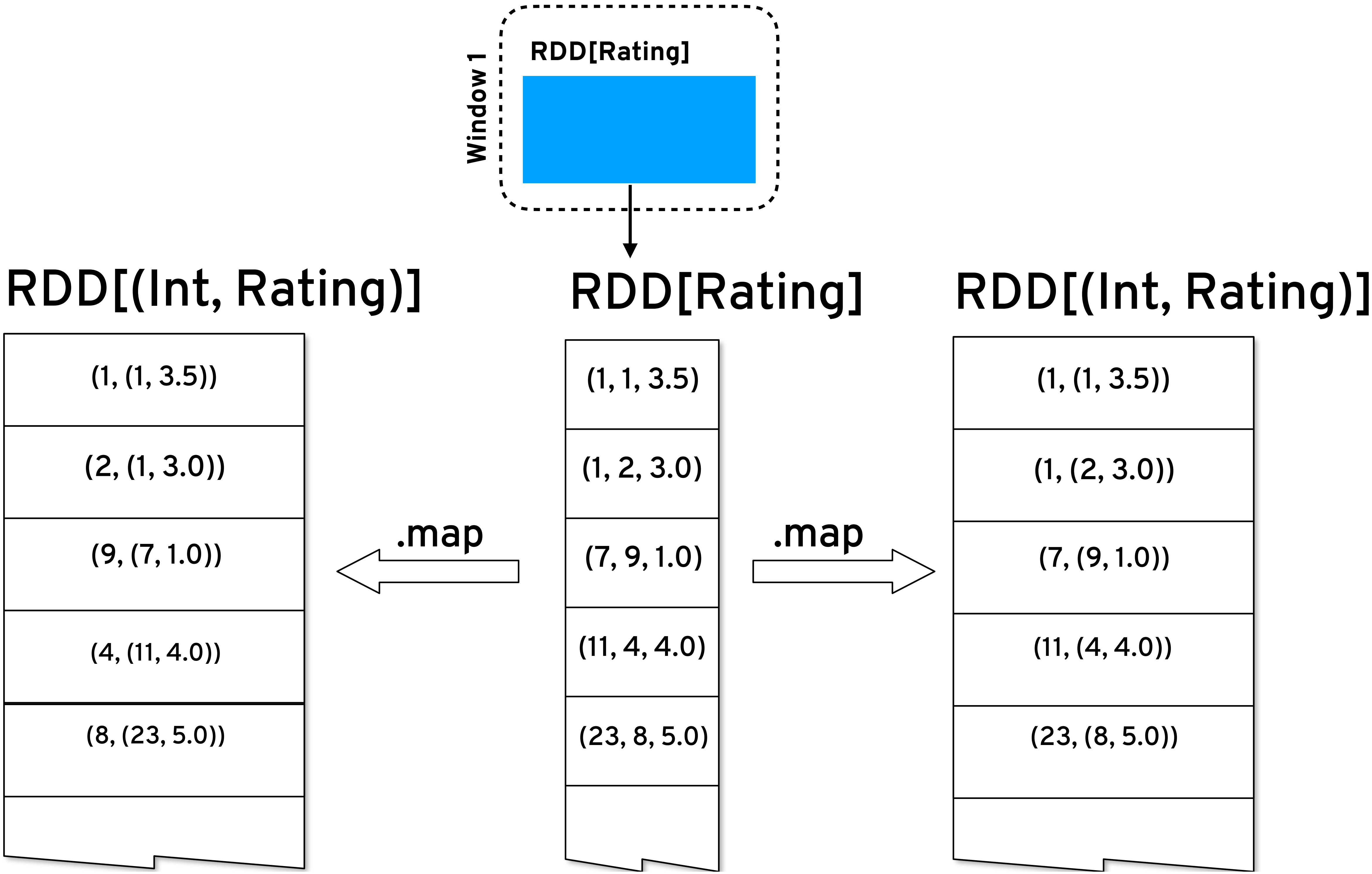
```
userBias += gamma * (error - lambda * userBias)

userFeature(i) += gamma * (error *  prodFeature(j) - lambda * userFeature(i))
```

```
case class Factor(var bias: Double, features: Array[Double])
  extends Serializable {

}
```

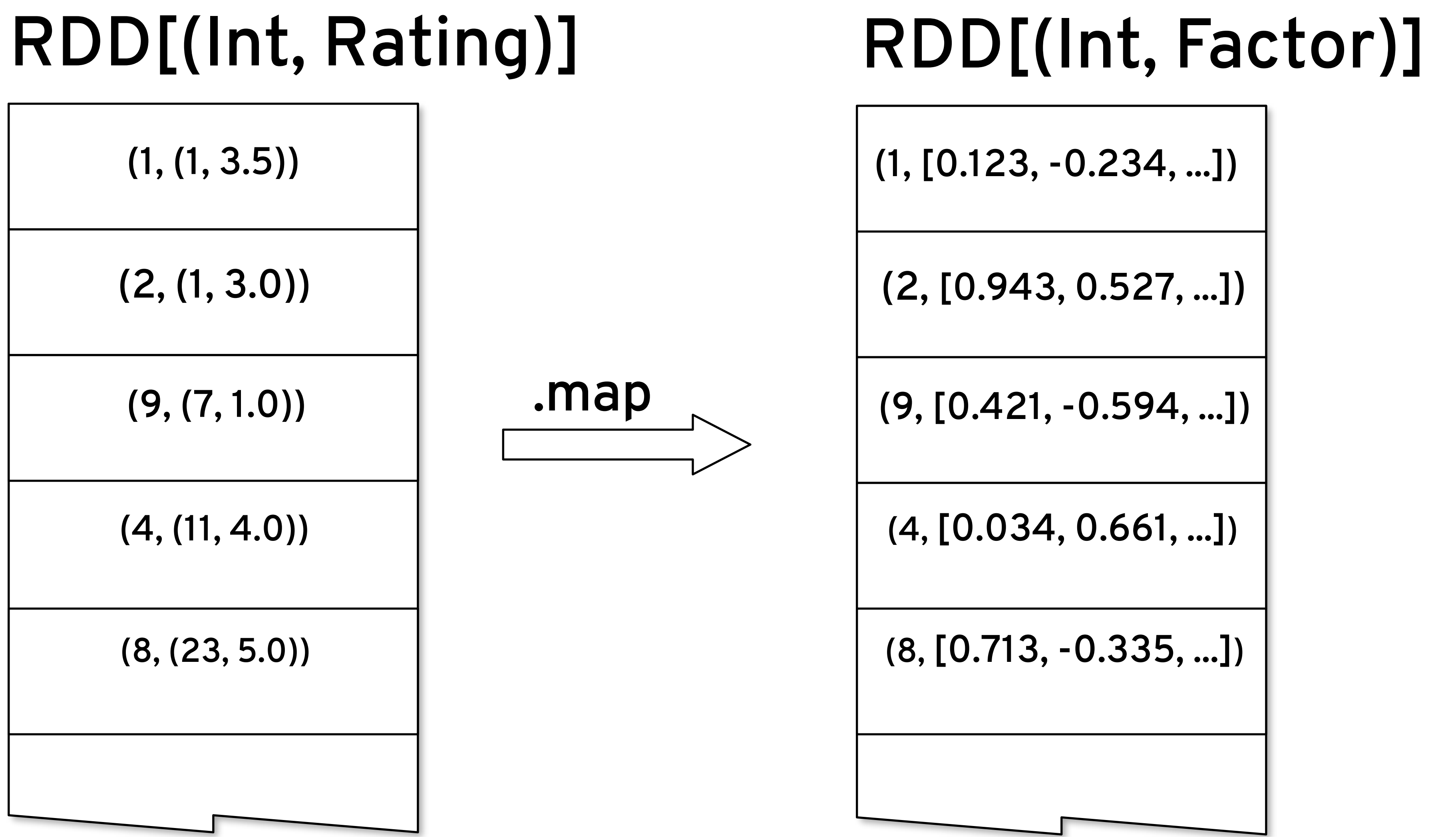
Batch → **Streaming**

RDD[(Int, Array[Double])] → **RDD[(Int, Factor)]**

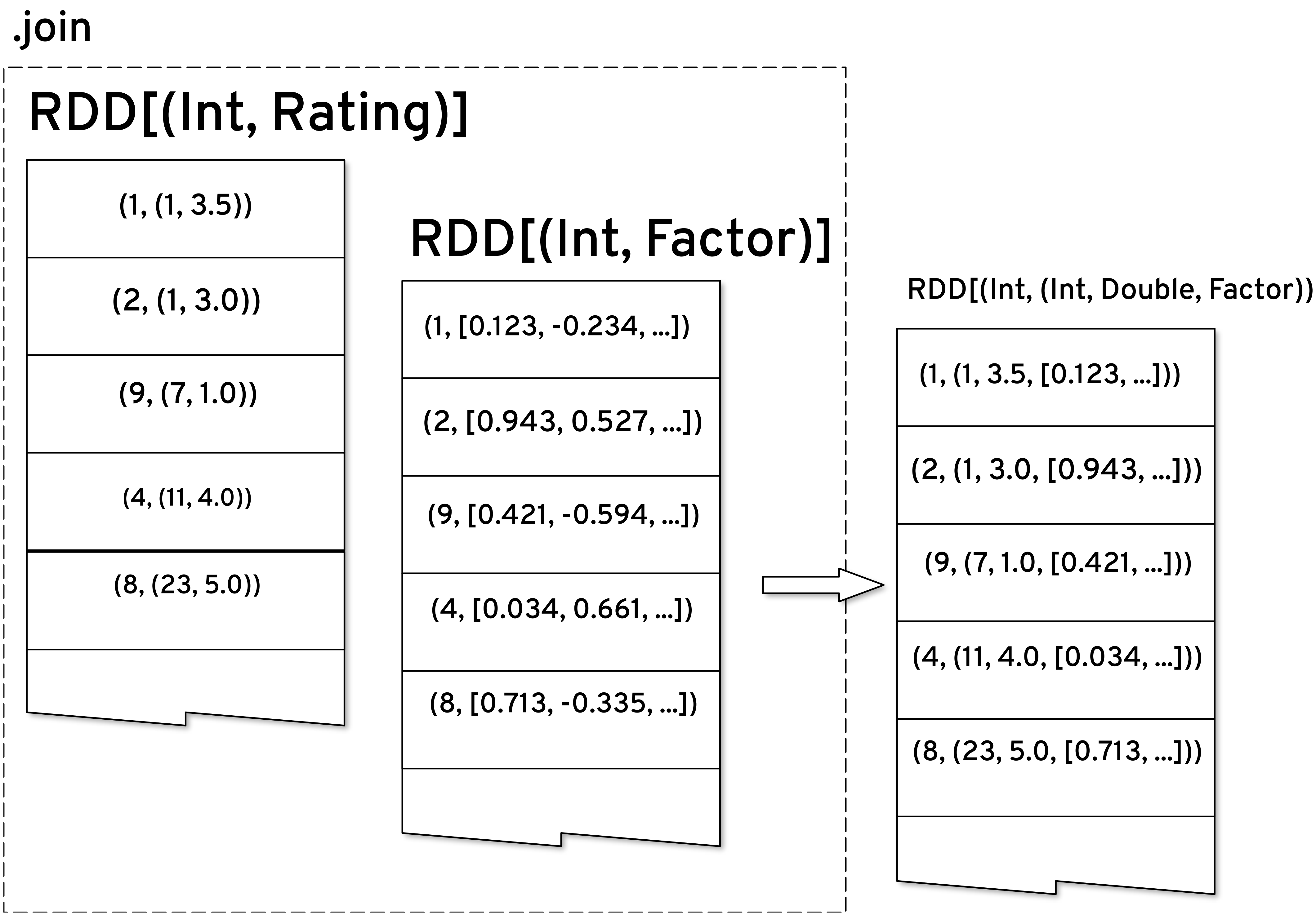
Spark Streaming ALS



Spark Streaming ALS

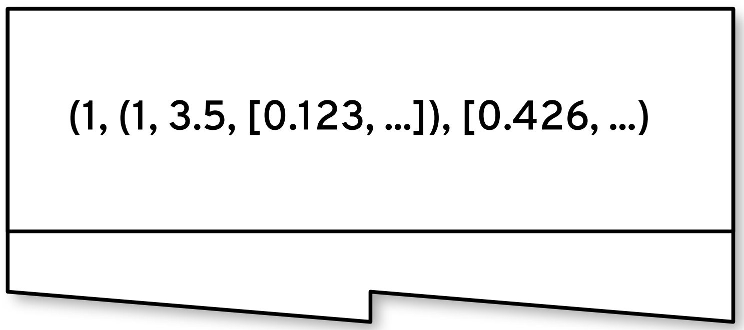


Spark Streaming ALS



Spark Streaming ALS

RDD[(Int, (Int, Double, Factor), Factor)]

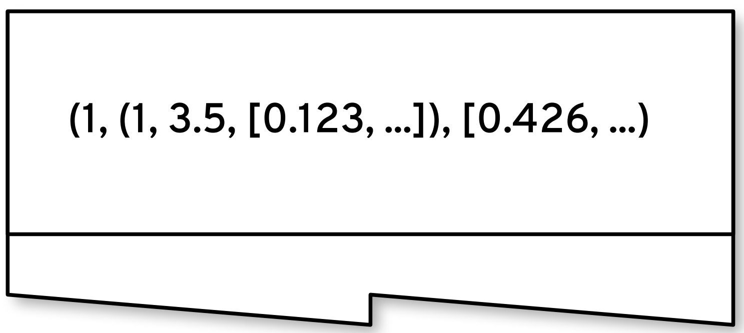


$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

```
prediction = dot(userFactors.features, itemFactors.features)
+ userFactors.bias + itemFactors.bias + bias
```

Spark Streaming ALS

RDD[(Int, (Int, Double, Factor), Factor)]



$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

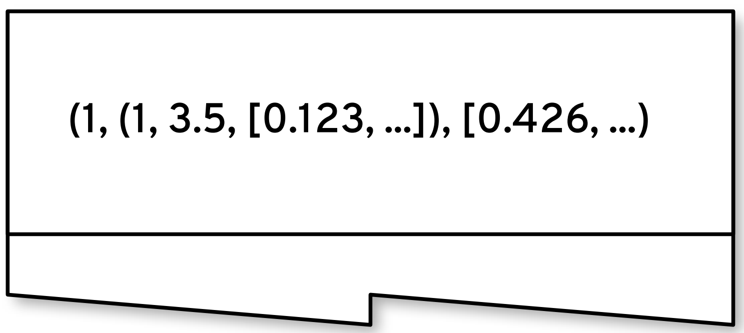
```
prediction = dot(userFactors.features, itemFactors.features)
+ userFactors.bias + itemFactors.bias + bias
```

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

```
eps = rating - prediction
```

Spark Streaming ALS

RDD[(Int, (Int, Double, Factor), Factor)]



$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

```
prediction = dot(userFactors.features, itemFactors.features)
+ userFactors.bias + itemFactors.bias + bias
```

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

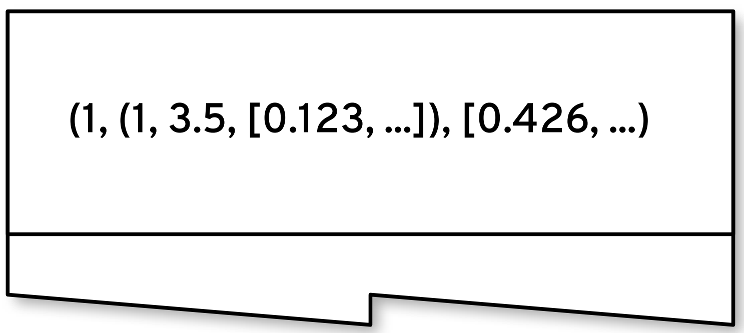
```
eps = rating - prediction
```

$$\gamma \left(\epsilon_{x,y} \mathsf{U}_x - \lambda'_y \mathsf{P}_y \right)$$

```
(0 until rank).map { i =>
    gamma * (eps * userFactors.features(i) - lambda * itemFactors.features(i))
  }
```

Spark Streaming ALS

RDD[(Int, (Int, Double, Factor), Factor)]



$$\hat{r}_{x,y} = \mu + b_x + b_y + \mathsf{U}_x \cdot \mathsf{P}_y^T$$

```
prediction = dot(userFactors.features, itemFactors.features)
+ userFactors.bias + itemFactors.bias + bias
```

$$\epsilon_{x,y} = r_{x,y} - \hat{r}_{x,y}$$

```
eps = rating - prediction
```

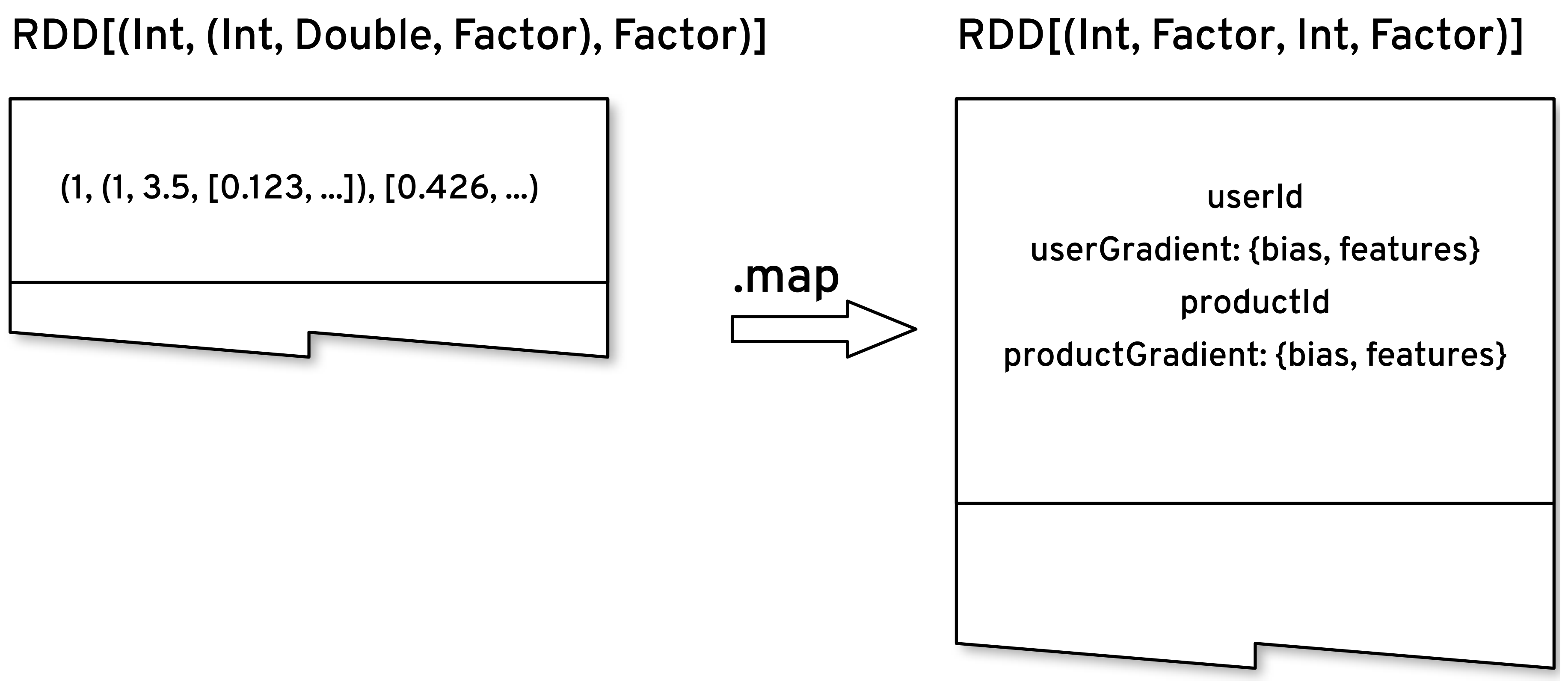
$$\gamma \left(\epsilon_{x,y} \mathsf{U}_x - \lambda'_y \mathsf{P}_y \right)$$

```
(0 until rank).map { i =>
    gamma * (eps * userFactors.features(i) - lambda * itemFactors.features(i))
  }
```

$$\gamma \left(\epsilon_{x,y} - \lambda_y b_y \right)$$

```
gamma * (eps - lambda * itemFactors.bias)
```

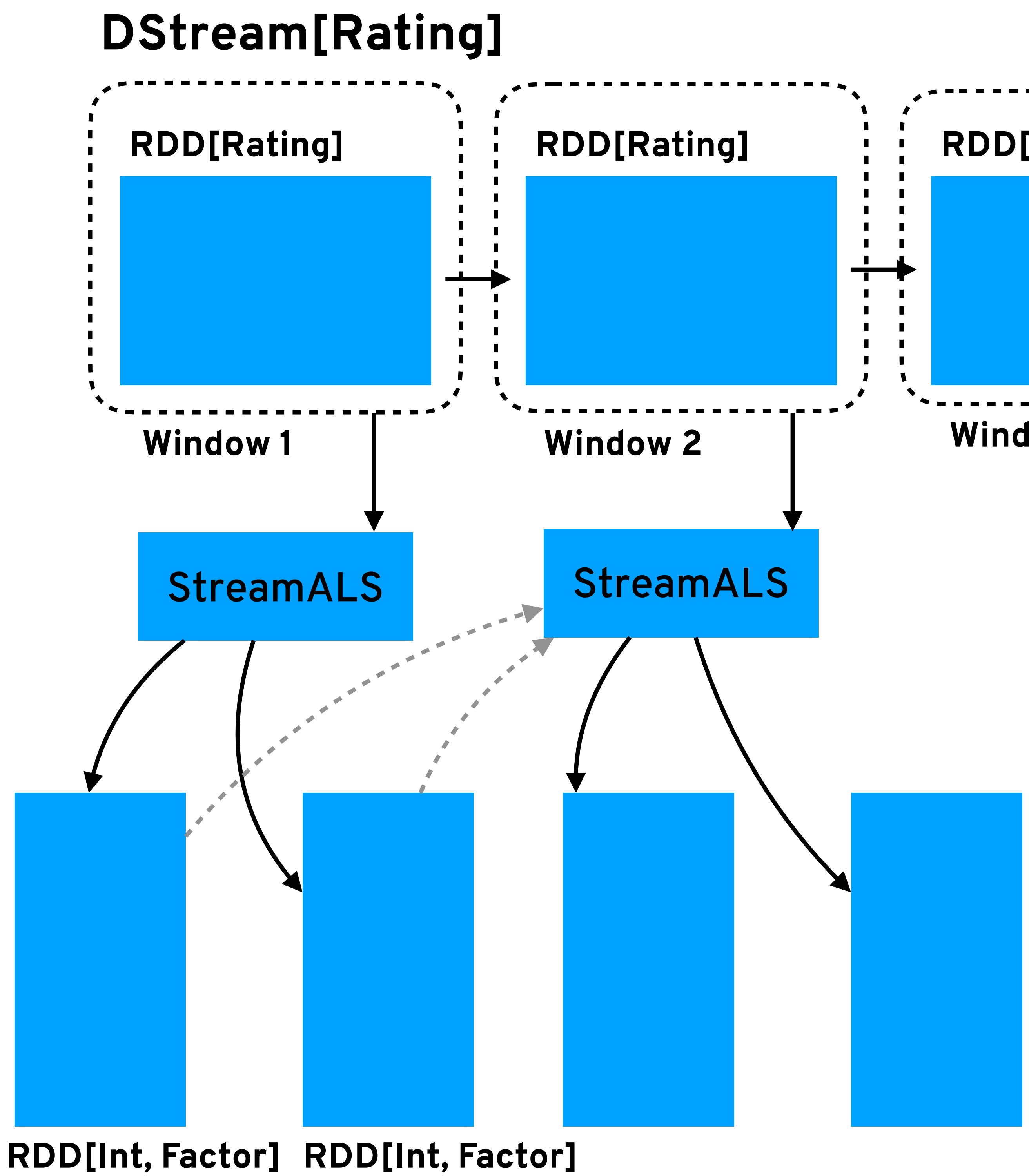
Spark Streaming ALS



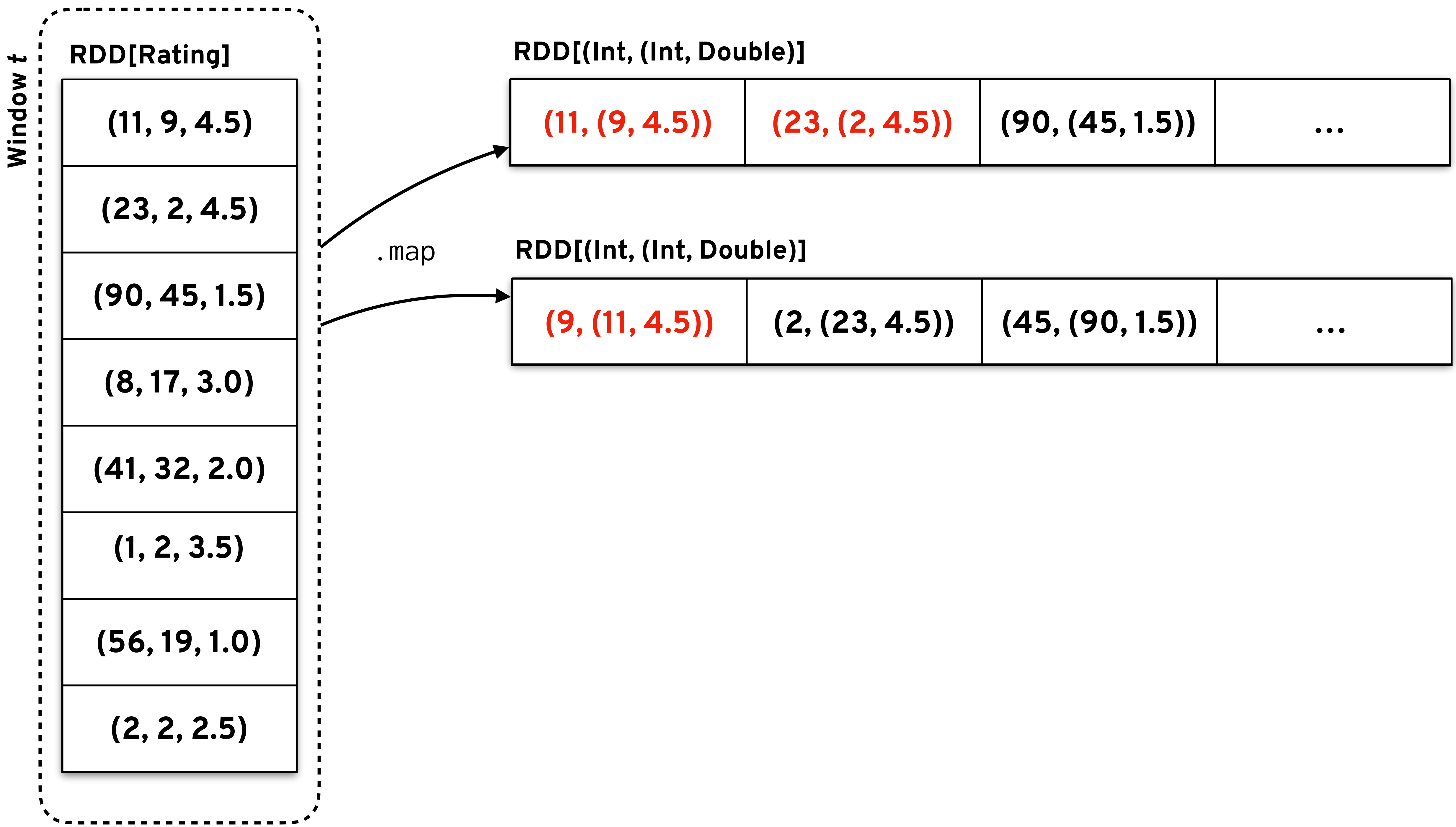
Spark Streaming ALS

- We need to .aggregateByKey
    - Bias, features for each user and product
- Finally recalculate final bias, features using .leftJoin
    - Add “original” Factor to gradient

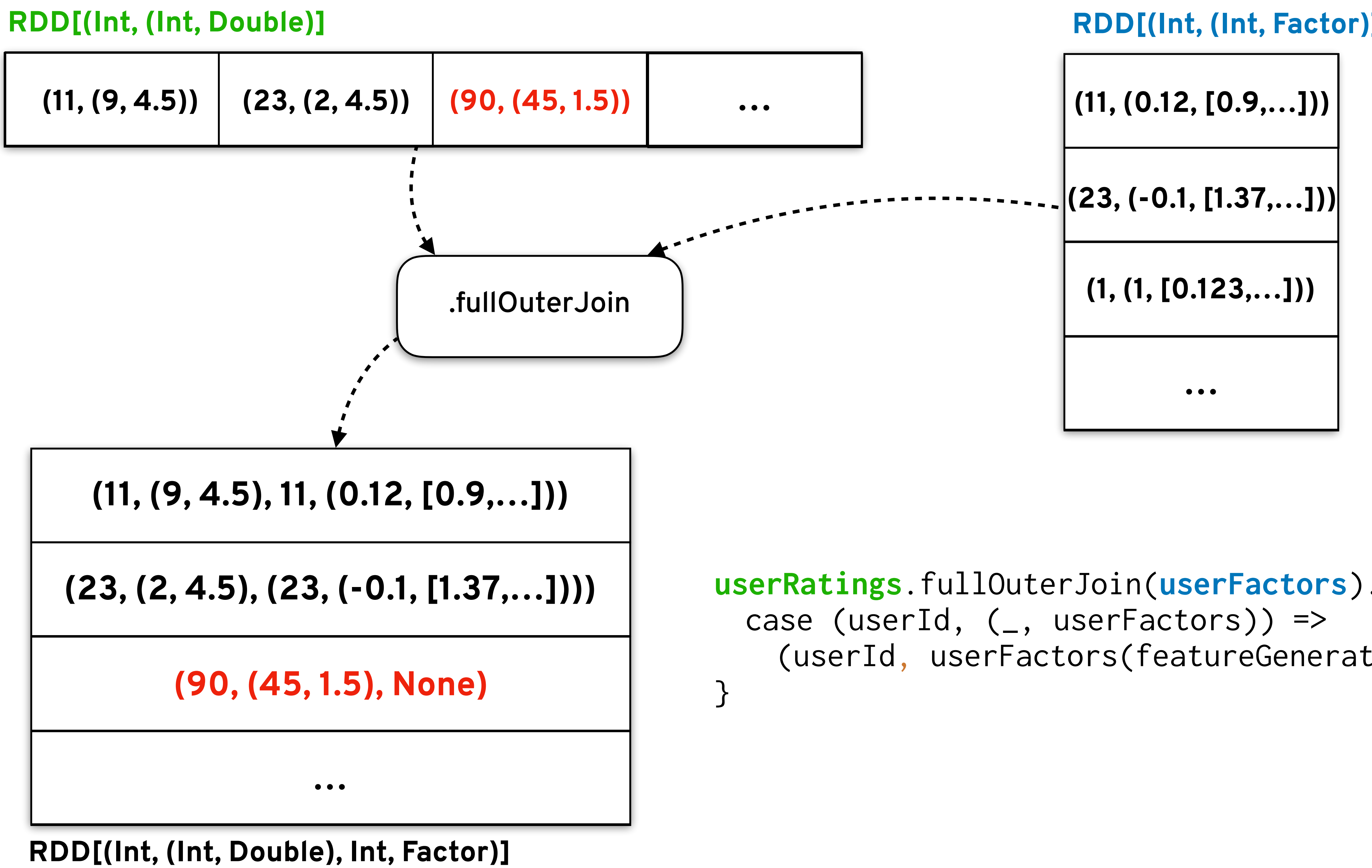
Spark Streaming ALS



Spark Streaming ALS



Spark Streaming ALS




```
userRatings.fullOuterJoin(userFactors).map {
  case (userId, (_, userFactors)) =>
    (userId, userFactors(featureGenerator.nextValue()))
}
```

Data

- MovieLens
- Widely used in recommendation engine research
- Variants
  - Small - 100,000 ratings / 9,000 movies / 700 users
  - Full - 26 million ratings / 45,000 movies / 270,000 users
- CSV data
  - Ratings
  - `(userId, movieId, rating, timestamp)`
  - `(100, 200, 3.5, 2010-12-10 12:00:00)`

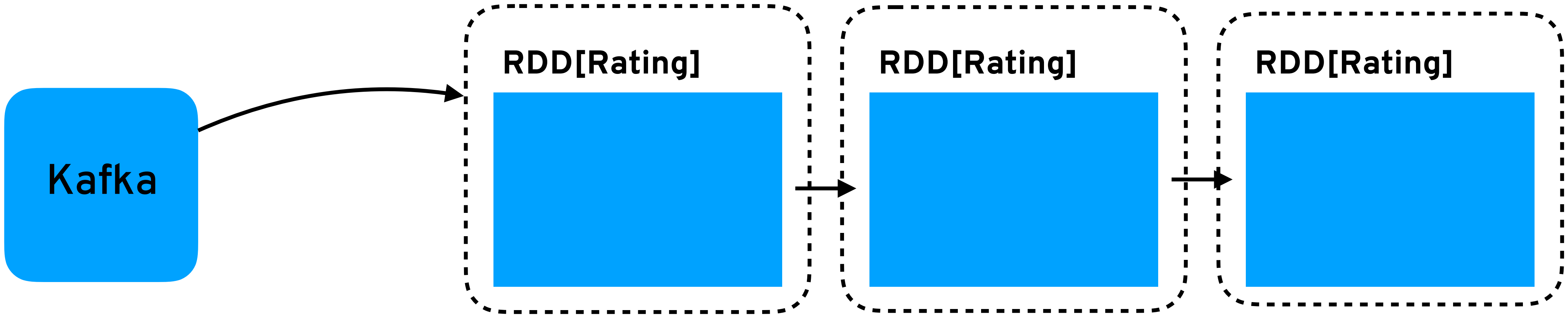
Training batch ALS

```
val split: Array[RDD[Rating]] = ratings.randomSplit(0.8, 0.2)

val model = ALS.train(split(0), rank, iter, lambda)
```

Training batch ALS

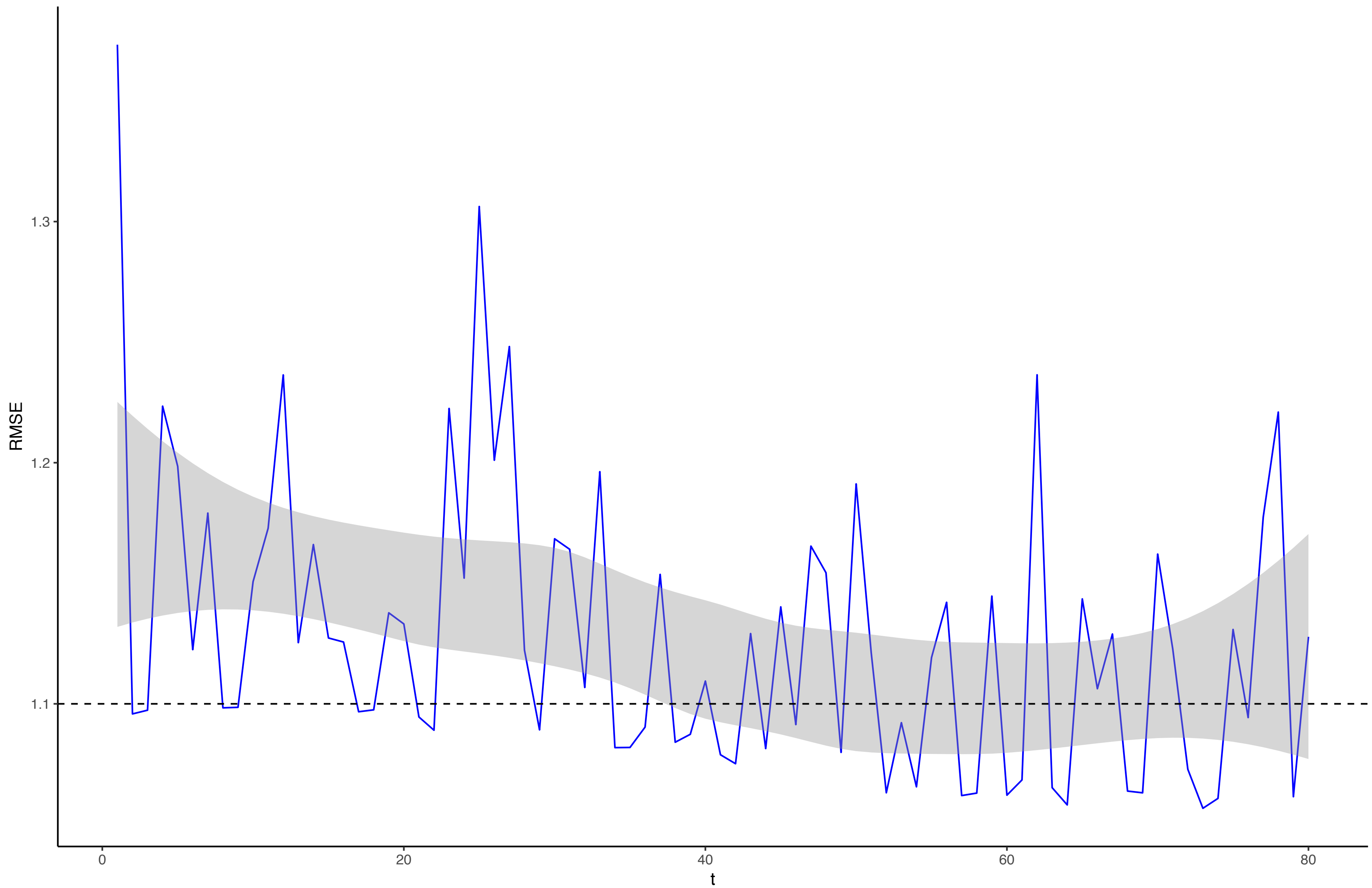
```
  val split: Array[RDD[Rating]] = ratings.randomSplit(0.8, 0.2)

  val model = ALS.train(split(0), rank, iter, lambda)


val predictions: RDD[Rating] = model.predict(split(1).map { x =>
  (x.user, x.product))
 }

val pairs = predictions.map(x => ((x.user, x.product), x.rating))
  .join(split(1).map(x => ((x.user, x.product), x.rating))
  .values

Val RMSE = math.sqrt(pairs.map(x => math.pow(x._1 - x._2, 2)).mean())
```

Training streaming ALS




```
val model = StreamingALS(rank, iterations, lambda, gamma)

trainingStreamSet.foreachRDD { rdd =>

  model.train(rdd)

  val RMSE = calculateRMSE(model, validation)

}
```

Comparison

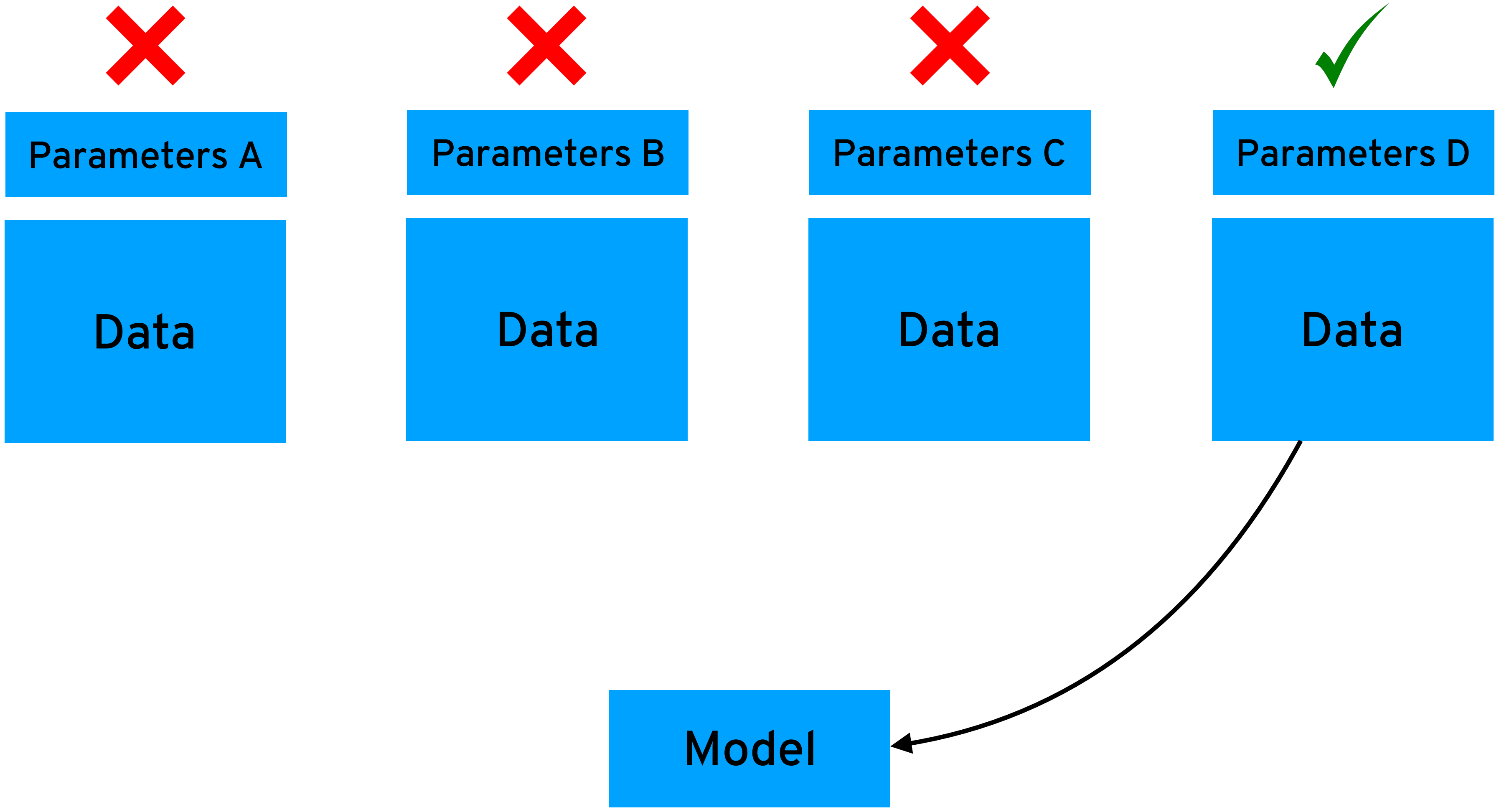


To consider

- “Cold start”
- Same as batch ALS
- Too few observations = meaningless
- Train offline

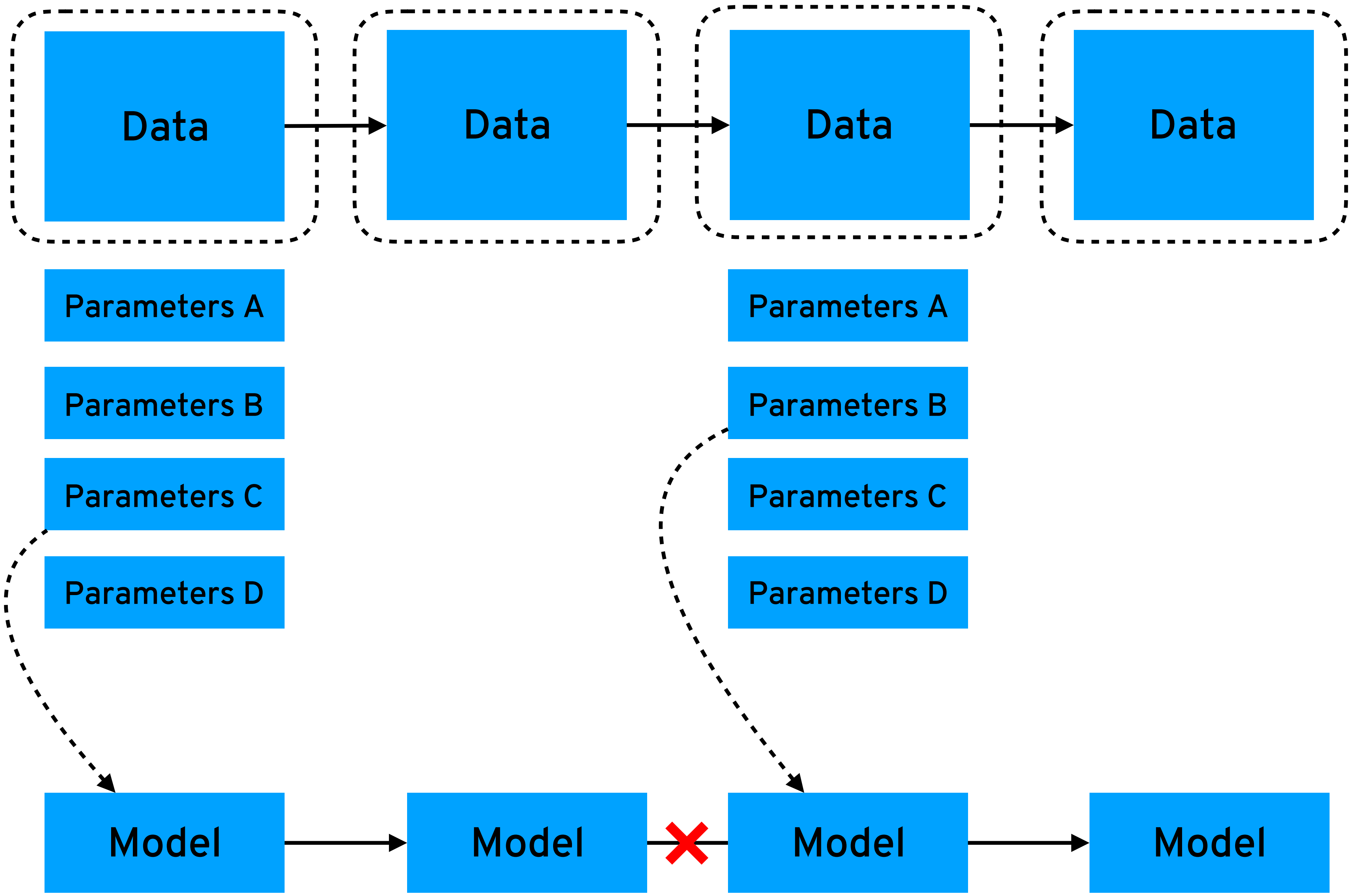
To consider

- **Hyper-parameter estimation**



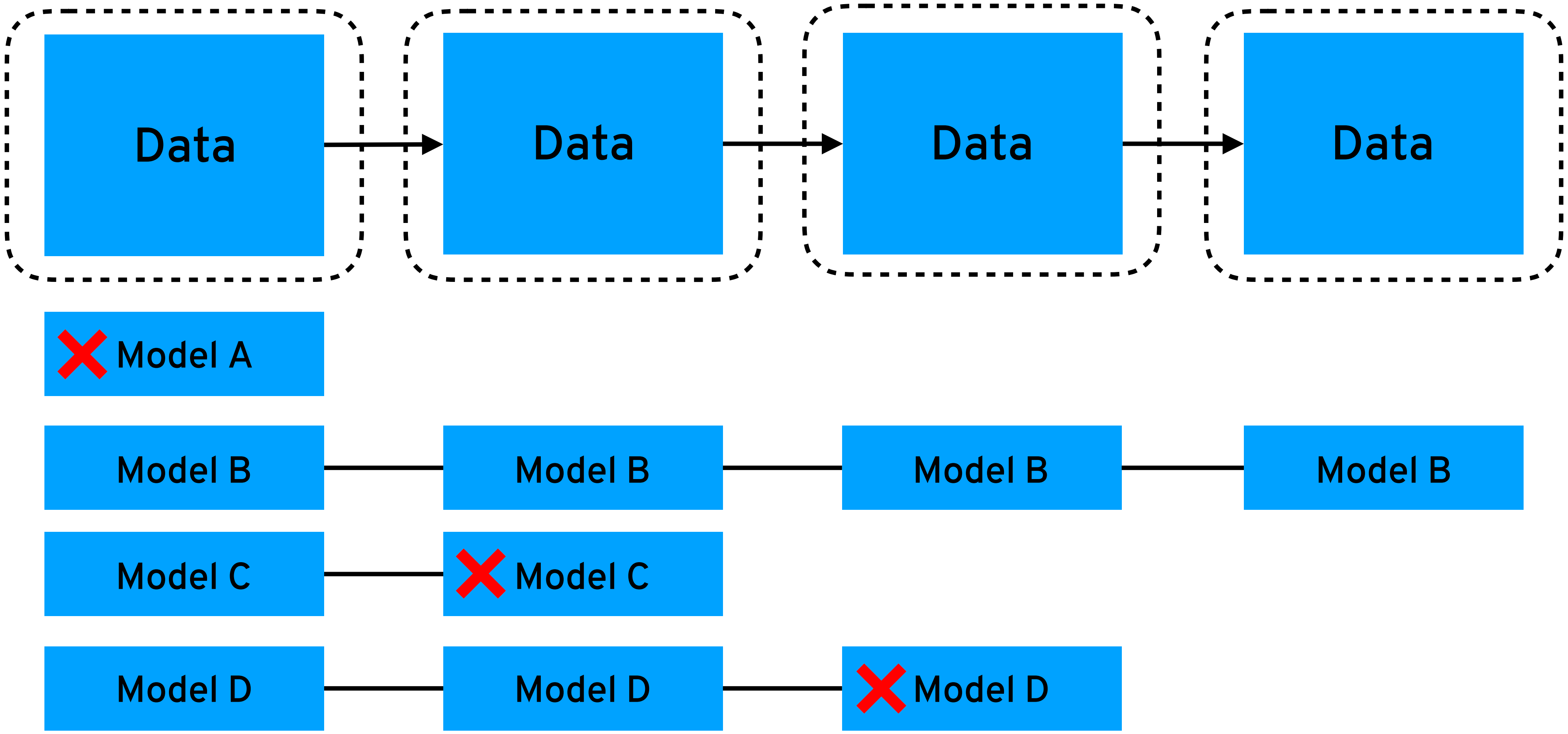
To consider

- **Hyper-parameter estimation**



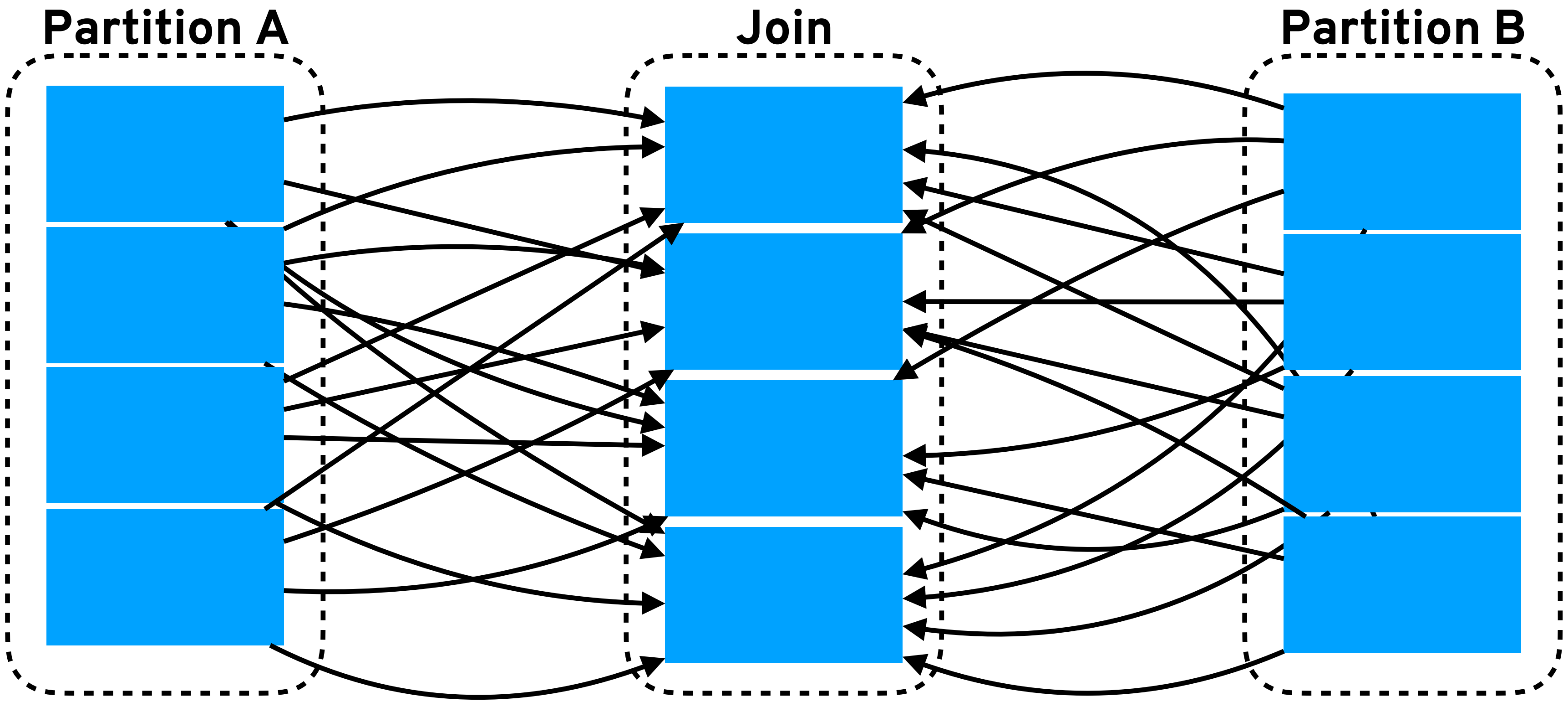
To consider

- **Hyper-parameter estimation**



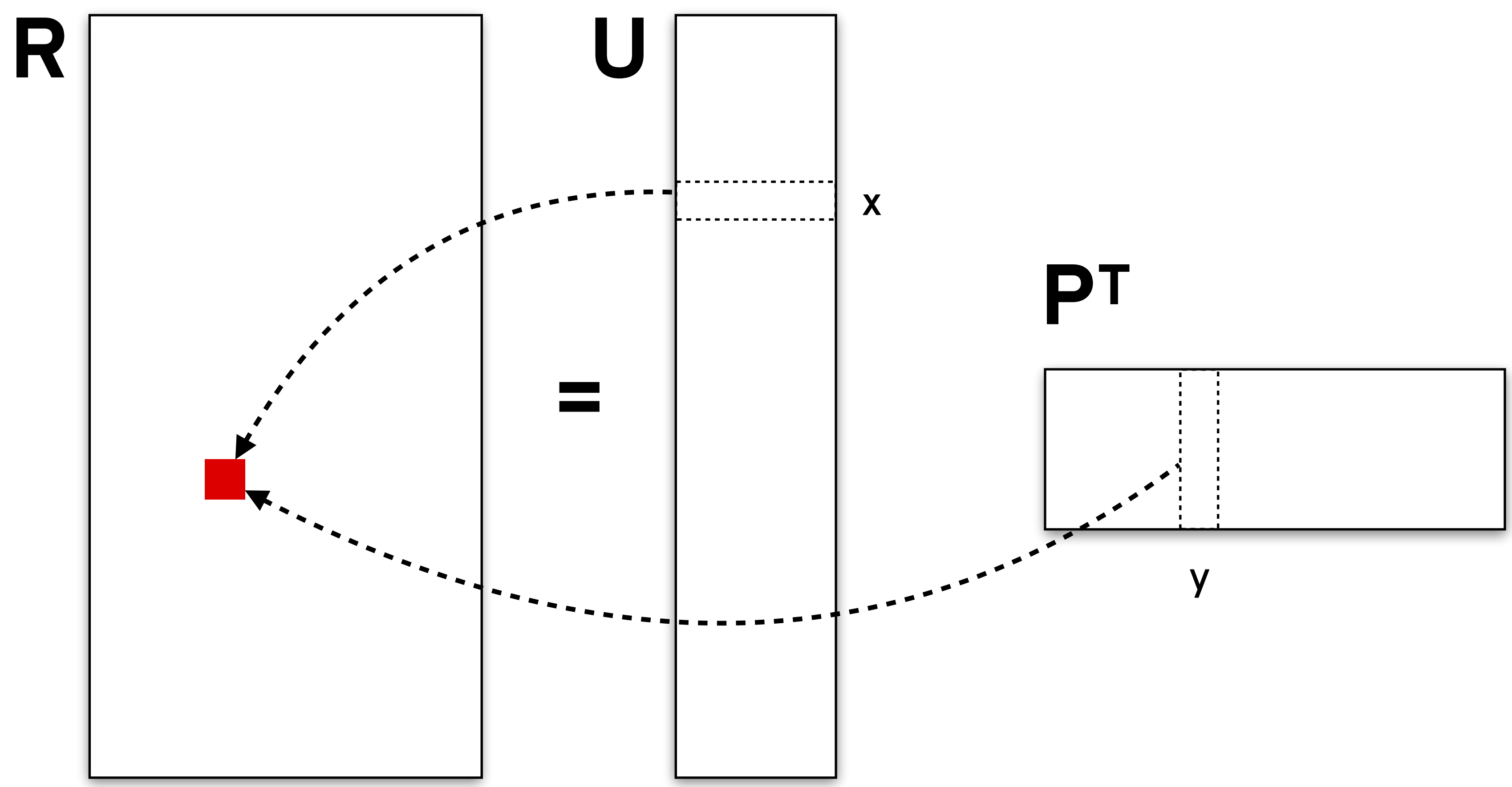
To consider

Partitioning



To consider

RDD random access?




```
val u = userFeatures.lookup(userId)
val p = productFeatures.lookup(productId)
val predicted = model.predict(userId, productId, u, p, globalBias)
```

Thank you!