In-Database Handling of Missing Data

Milos Nikolic
University of Edinburgh

FDB Workshop, August 2022

Joint work with Massimo Perini

(Clean) Data Is Important

- Data underlies decision making
- Data is key for data analytics
- The need for clean data
  - Value derived from data is as good as the data itself
  - Garbage in, garbage out

Focus of This Talk: Missing Data

- Missing data is common in practice
  - Human errors, equipment malfunctions, data integration, etc.
- Problems
  - Can introduce significant bias
  - Reduces the power of reasoning
  - Requires special handling as tools assume complete data

Handling Missing Values

- Common approach: discard tuples with missing values
  - May introduce bias in the analysis
  - Reduces dataset size, especially in multivariate analysis
- Data imputation
  - Preserve all tuples by replacing missing values with estimates
  - Imputed dataset can then be analyzed using standard techniques

Data Imputation

Mean imputation

Replace missing values with the mean of observed values for that attribute

Preserves the mean but distorts estimated variances and correlations

Regression imputation

Regression model for predicting Y from X = (X_1, ..., X_n)

Overstates the strength of the relationship between Y and X

No uncertainty about the predicted value

Multiple Imputation

Compute multiple imputations for each missing values

- Produces multiple plausible versions of the complete dataset
- The analysis results are combined to get estimates and std errors

Multiple Imputation by Chained Equations (MICE)

- Series of regression models predicting each variable with missingness using all other variables

Multiple Imputation by Chained Equations (MICE)

Age	Income	Level
40	x	Senior
24	45,000	x
x	35,000	Junior

x – missing value

Age	Income	Level
40	**40,000**	Senior
24	45,000	**Senior**
32	35,000	Junior

Mean imputation

Age	Income	Level
40	**40,000**	Senior
24	45,000	**Senior**
x	35,000	Junior

Income, Level → Age

Age	Income	Level
40	**40,000**	Senior
24	45,000	**Senior**
27.8	35,000	Junior

Predict Age

Age	Income	Level
40	x	Senior
24	45,000	**Senior**
27.8	35,000	Junior

Age, Level → Income

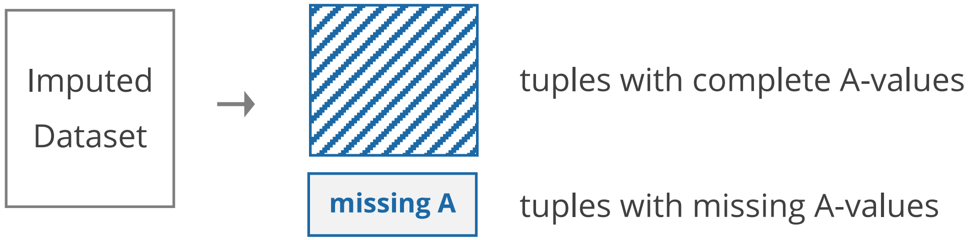
Age	Income	Level
40	**56,764**	Senior
24	45,000	**Senior**
27.8	35,000	Junior

Predict Income

Repeat for Level, Age, Income, …

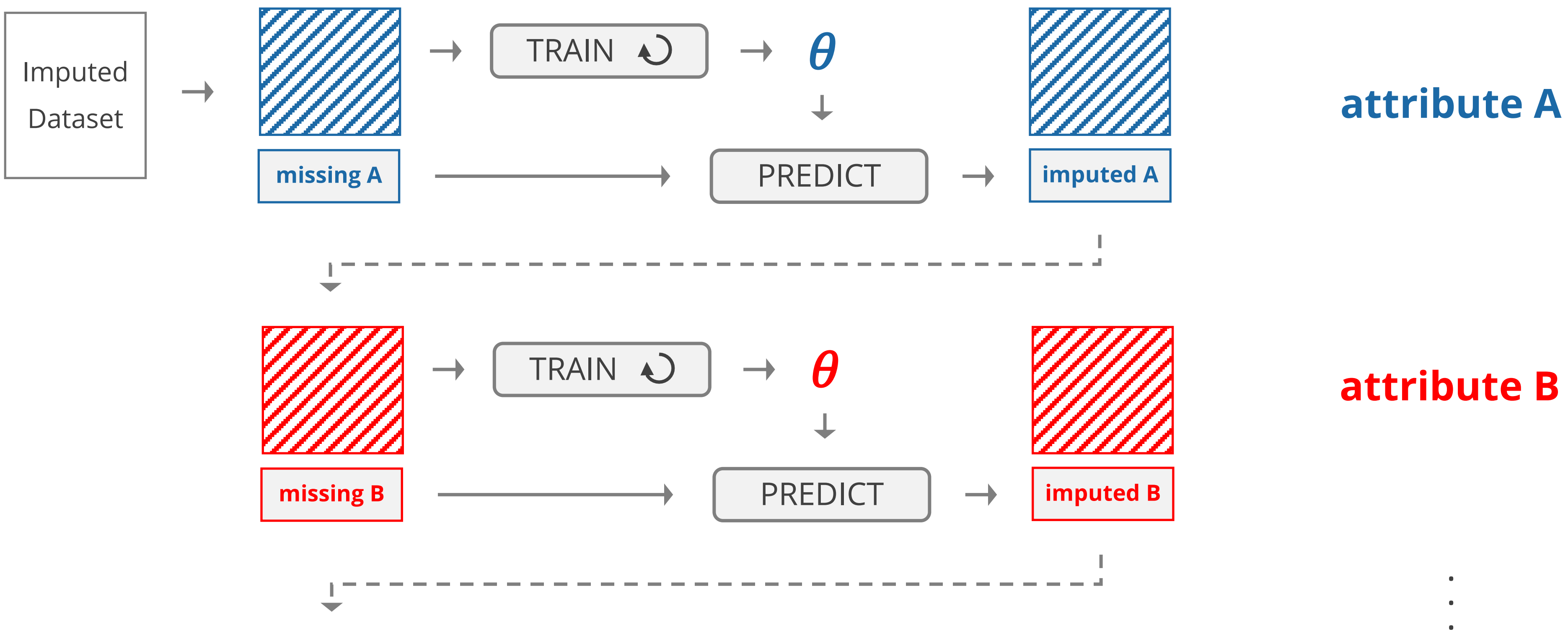
Round robin until convergence

MICE Overview



attribute A

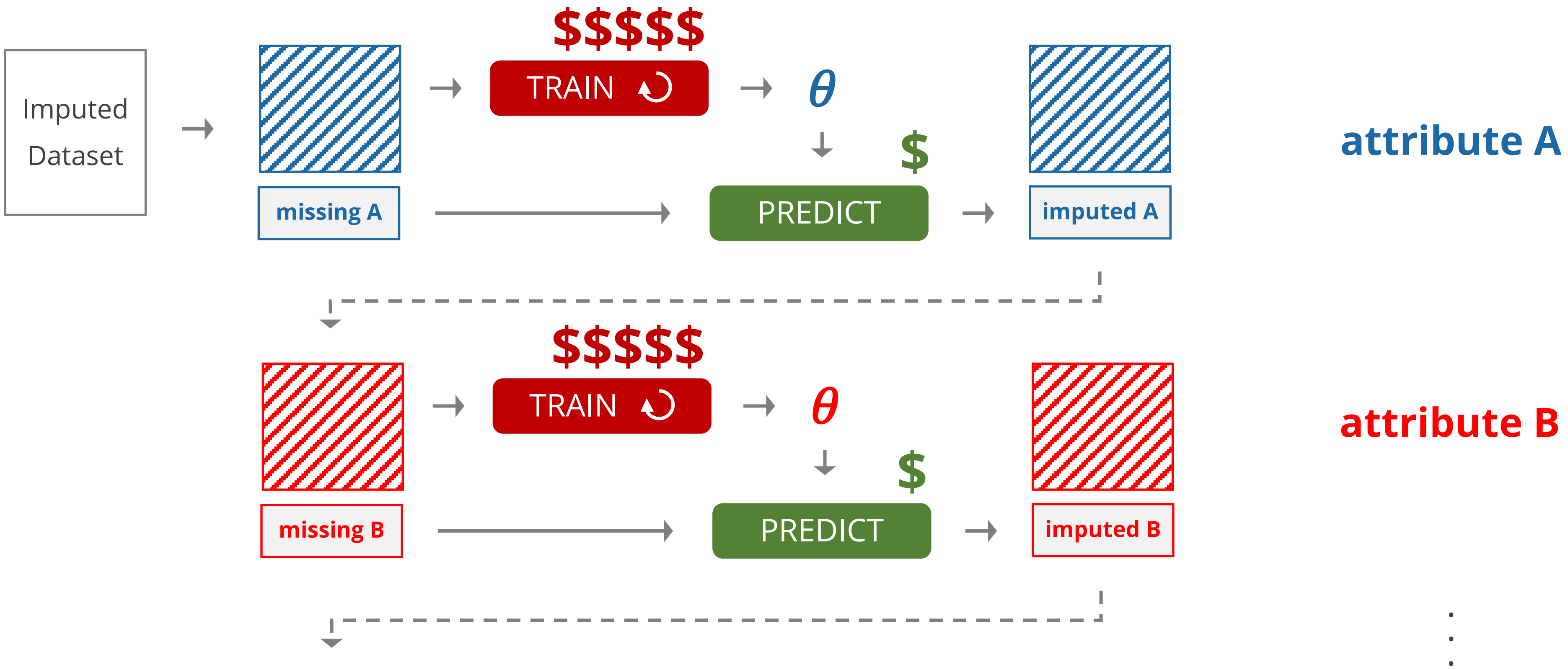
MICE Overview

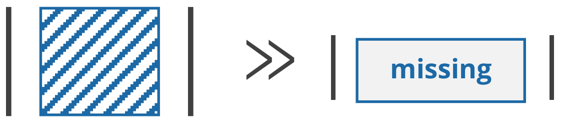


Models trained over different subsets of data

Models may differ: regression and classification

MICE Overview



Common assumption: 

Retraining is $$$!

Data Imputation: State of Affairs

Statistical libraries can handle complex imputation but do not scale

DBMSs can handle large data but not complex imputation

Possible solution: UDFs over denormalized data

- Joining data is expensive
- Data export/import is costly

In-Database Data Imputation

Goal: Efficient, scalable, in-database MICE implementation

Step 1: Reformulate model training as DB aggregation

Linear regression (continuous) [Schleich & Olteanu], [Nikolic & Olteanu]

Gaussian Discriminant Analysis (categorical)

Step 2: Exploit sharing opportunities across models

Linear Regression

[Schleich & Olteanu]

Linear model

$$LR = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n = \sum_{j \in [n]} \theta_j x_j$$

Loss function

$$J(\theta) = \frac{1}{2|X|} \sum_{(x,y) \in X} \Big(\sum_{j \in [n]} \theta_j x_j - y\Big)^2 = \frac{1}{2|X|} \sum_{x \in X} \Big(\underbrace{\sum_{j \in [n]}}_{\theta_y = -1} \theta_j x_j\Big)^2$$

Gradient descent

```
repeat until convergence:
```

$$\forall k \in [n] : \theta_k := \theta_k - \alpha \cdot \nabla_k J(\vec{\theta})$$

$$:= \theta_k - \alpha \cdot \frac{1}{|X|} \sum_{x \in X} \Big(\sum_{j \in [n]} \theta_j x_j\Big) x_k$$

Linear Regression

Gradient rewriting

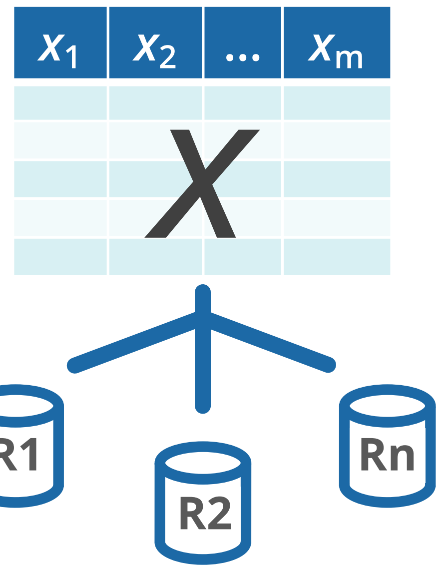
$$\nabla_k J(\vec{\theta}) := \frac{1}{|X|} \sum_{x \in X} \Big(\sum_{j \in [n]} \theta_j x_j \Big) x_k$$

$$:= \frac{1}{|X|} \sum_{j \in [n]} \theta_j \cdot \sum_{x \in X} (x_j \cdot x_k)$$

Compute data-dependent part

```
for each (Xj,Xk):
  Qjk = SELECT SUM(Xj * Xk)
          FROM R1 JOIN R2 JOIN ... JOIN Rn
```

Aggregates Q_{jk} are entries in matrix $\Sigma = X^T X$



Linear Regression over Joins

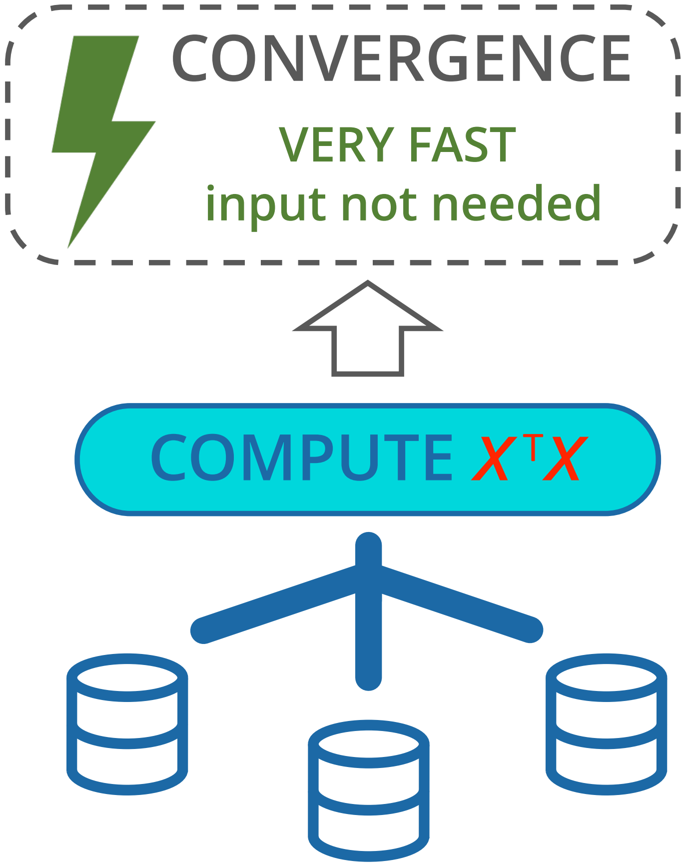
Problem: Compute a batch of SUM(x_i * x_j), for each (x_i, x_j)

```
Q = SELECT SUM(X_1 * X_1), ..., SUM(X_1 * X_n),
           ...
           SUM(X_n * X_1), ..., SUM(X_n * X_n)
      FROM R1 JOIN R2 JOIN ... JOIN Rn
```

Chances for sharing computation

SUMs of similar form & over same relations

No DBMS can do it efficiently



Ring Generalization

[Nikolic & Olteanu]

Problem: Compute $X^{T}X$ once for all iterations

```
Q = SELECT SUM(X₁ * X₁), ..., SUM(X₁ * Xₙ),
           ...
           SUM(Xₙ * X₁), ..., SUM(Xₙ * Xₙ)
      FROM R1 JOIN R2 JOIN ... JOIN Rn
```

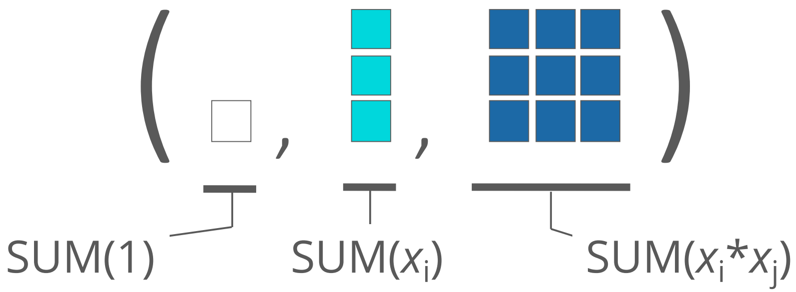
Solution:

Define a ring for aggregate values

Compute just ONE compound aggregate

Ring Generalization for $X^{\mathsf{T}}X$

Compute $X^{\mathsf{T}}X$ as a **triple of aggregates** (c, s, Q)



$(c_a, \mathbf{s}_a, \mathbf{Q}_a) + (c_b, \mathbf{s}_b, \mathbf{Q}_b) = (c_a + c_b, \mathbf{s}_a + \mathbf{s}_b, \mathbf{Q}_a + \mathbf{Q}_b)$

$(c_a, \mathbf{s}_a, \mathbf{Q}_a) * (c_b, \mathbf{s}_b, \mathbf{Q}_b) = (c_a c_b, c_b\mathbf{s}_a + c_a\mathbf{s}_b, c_b\mathbf{Q}_a + c_a\mathbf{Q}_b + \mathbf{s}_a\mathbf{s}_b^{\mathsf{T}} + \mathbf{s}_b\mathbf{s}_a^{\mathsf{T}})$

SUM(1) reused for all SUM(x_i) and SUM(x_i * x_j)

SUM(x_i) reused for all SUM(x_i * x_j)

Recap: Linear Regression over Joins

Solution: Compute $X^T X$ as ONE aggregate

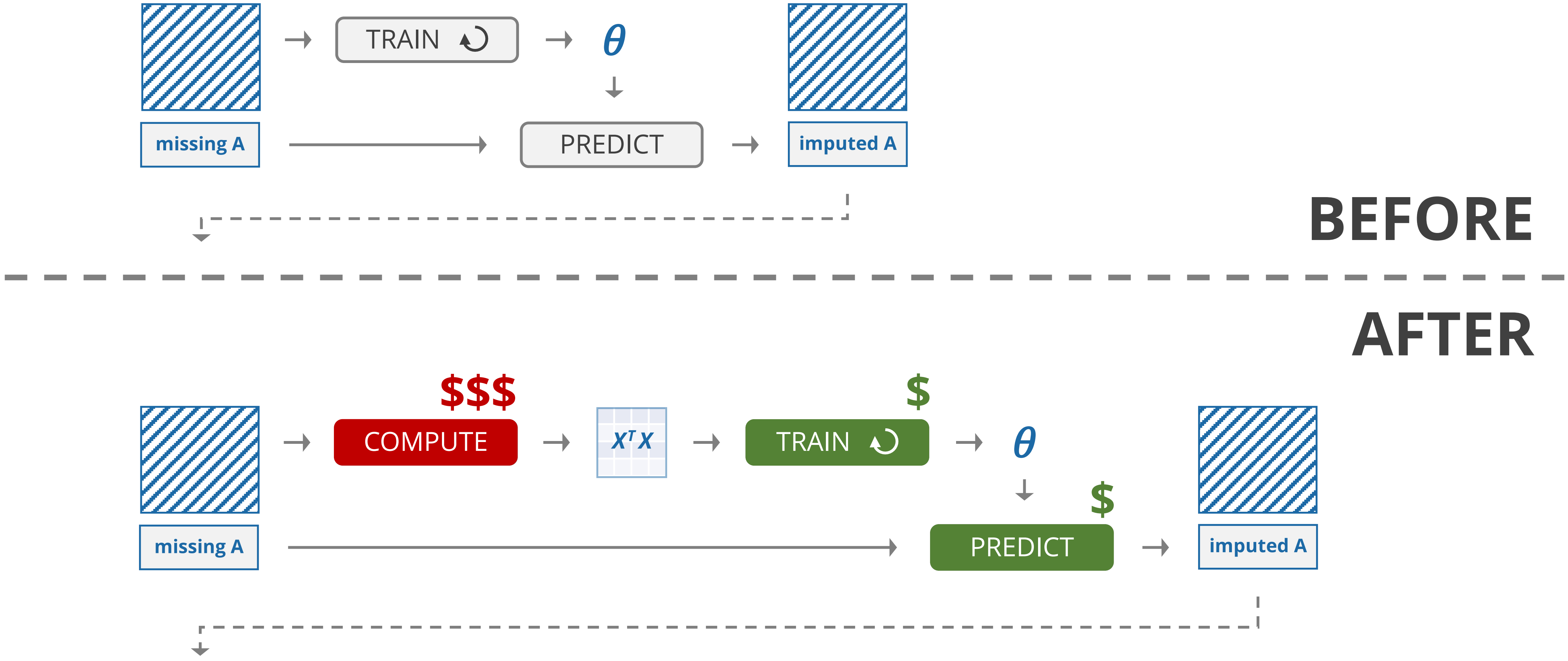
```
Q = SELECT SUM(g₁(X₁) * ... * gₙ(Xₙ))
      FROM R1 JOIN R2 JOIN ... JOIN Rn
```

$$g_i(x) = \left(\boxed{1},\ \begin{array}{cc} & \\ & \begin{bmatrix} 0 \\ x \\ 0 \end{bmatrix} \\ i & \end{array},\ \begin{array}{cc} & \begin{array}{ccc} & i & \end{array} \\ i & \begin{bmatrix} 0 & 0 & 0 \\ 0 & x^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \right)$$

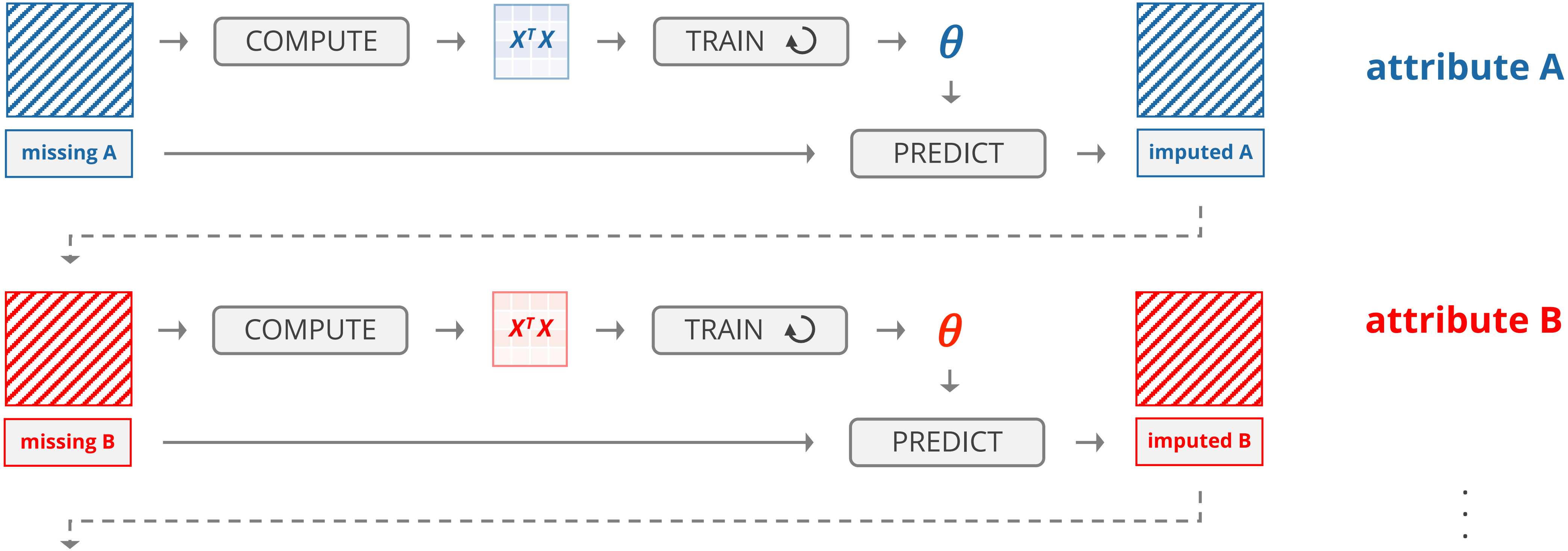
but with a specific payload ring and g_i functions

- ✓ Fully shared computation
- ✓ Factorized computation as for ordinary sums
- ✓ Incremental computation

Step 1: ML Problem ⇒ DB Problem

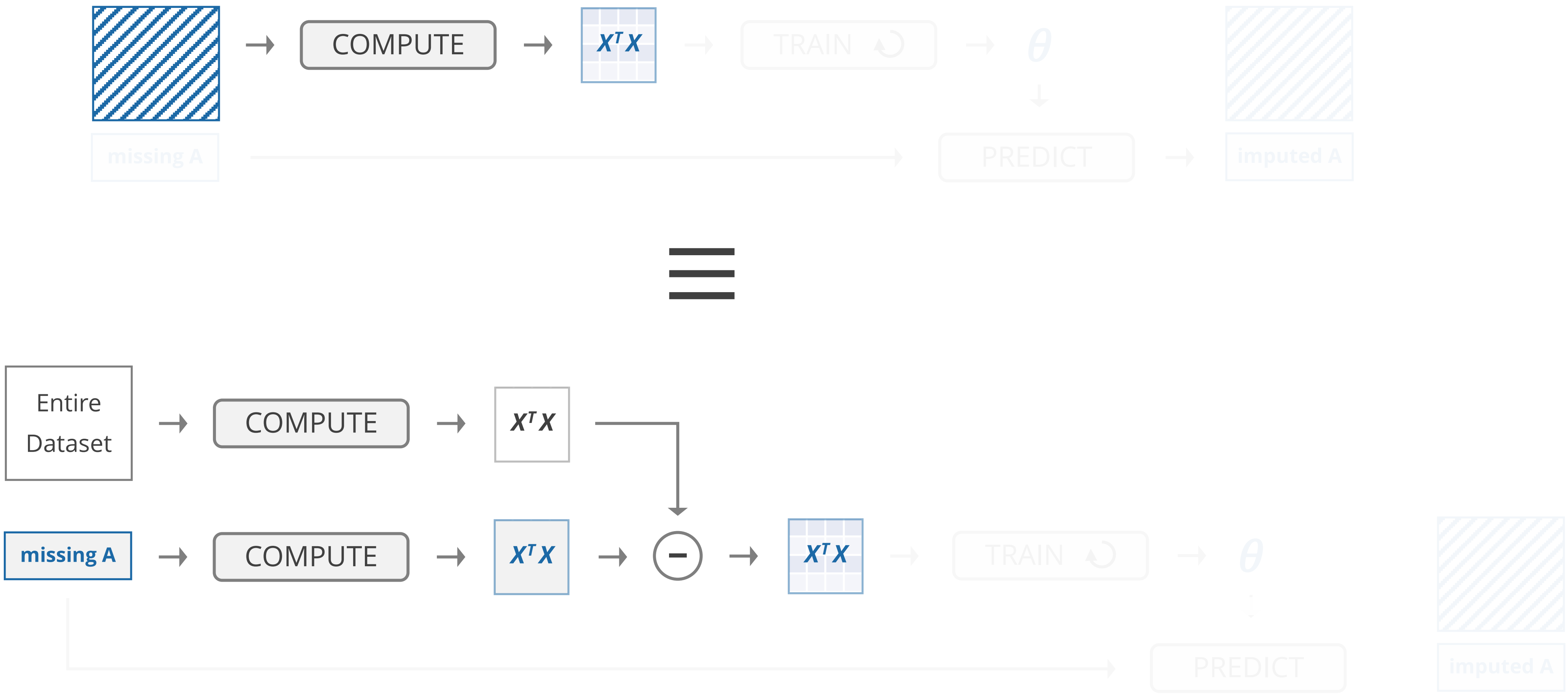


Checkpoint: In-Database MICE

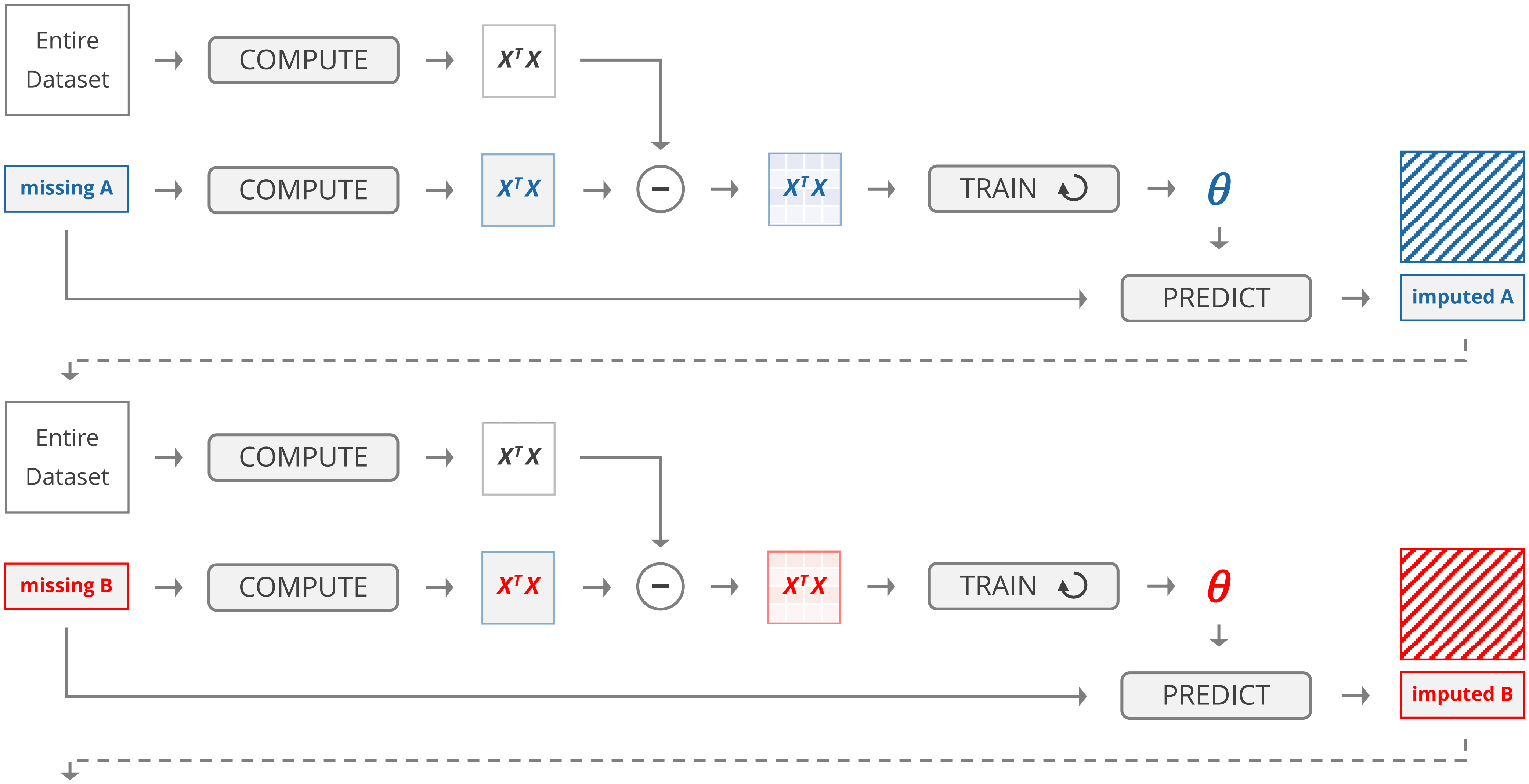


X^TX computed over overlapping subsets of complete data. Sharing?

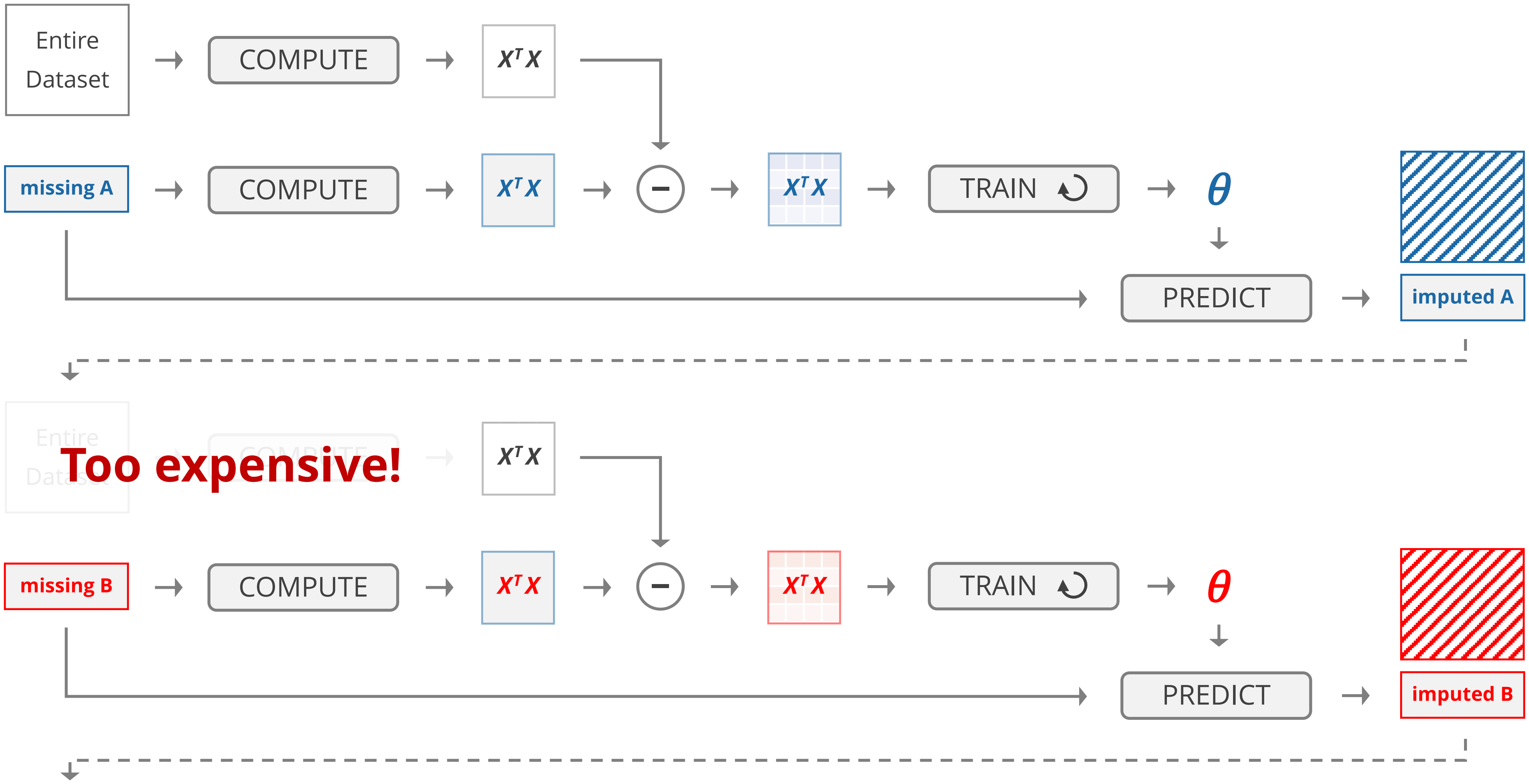
Step 2: Sharing Opportunities



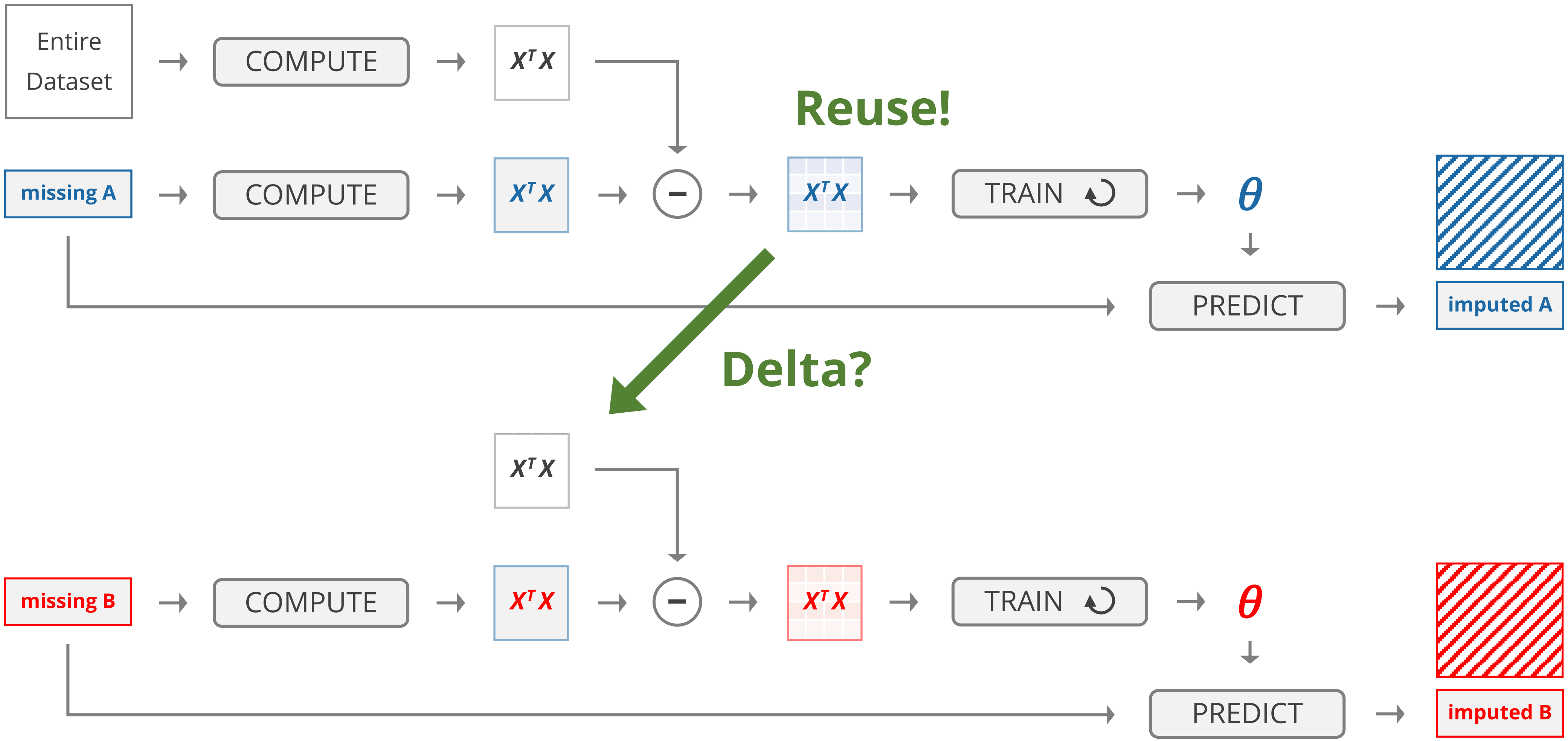
Step 2: Sharing Opportunities (Cont.)



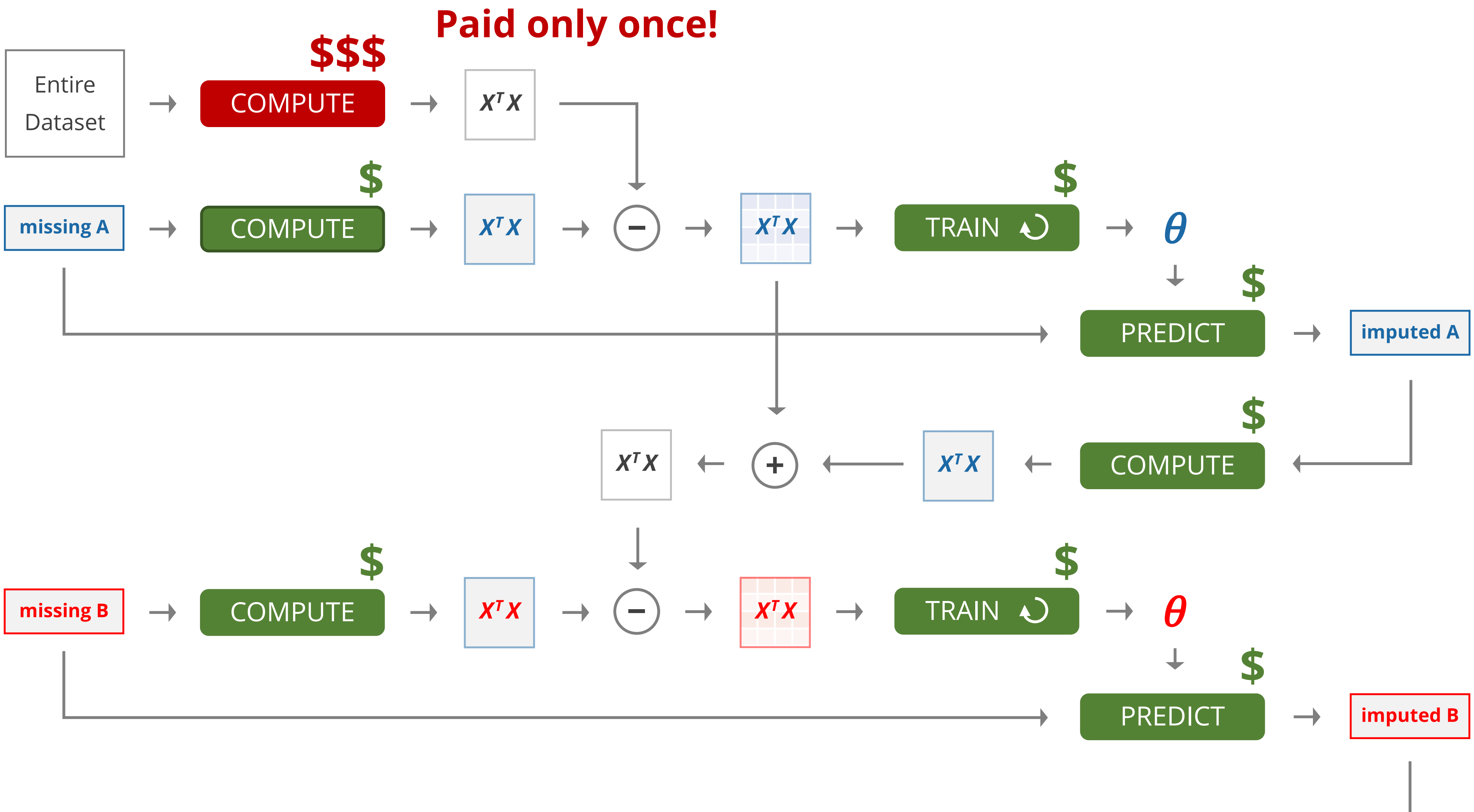
Step 2: Sharing Opportunities (Cont.)



Step 2: Sharing Opportunities (Cont.)



Step 2: Sharing Opportunities (Cont.)



MICE Implementation in PostgreSQL

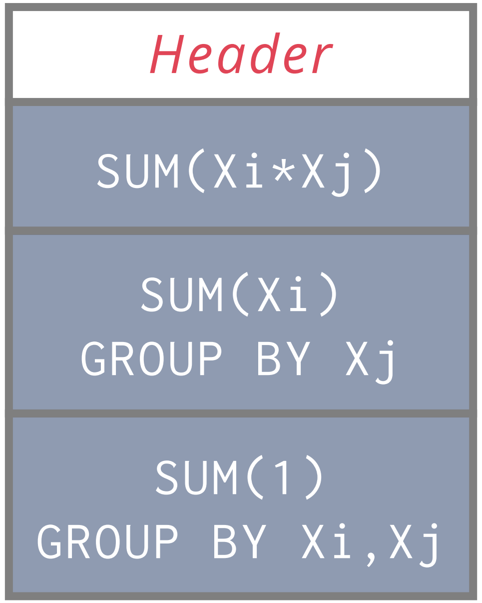
```
CREATE TYPE cofactor
```

- Parallel-safe operators +, -, *
- Support for continuous & categorical attributes
  - Avoids one-hot encoding

Model training in UDFs

- Linear regression and GDA

MICE driver in PL/pgSQL



Memory representation of a ring value

Implementation Challenges

Flat representation in memory

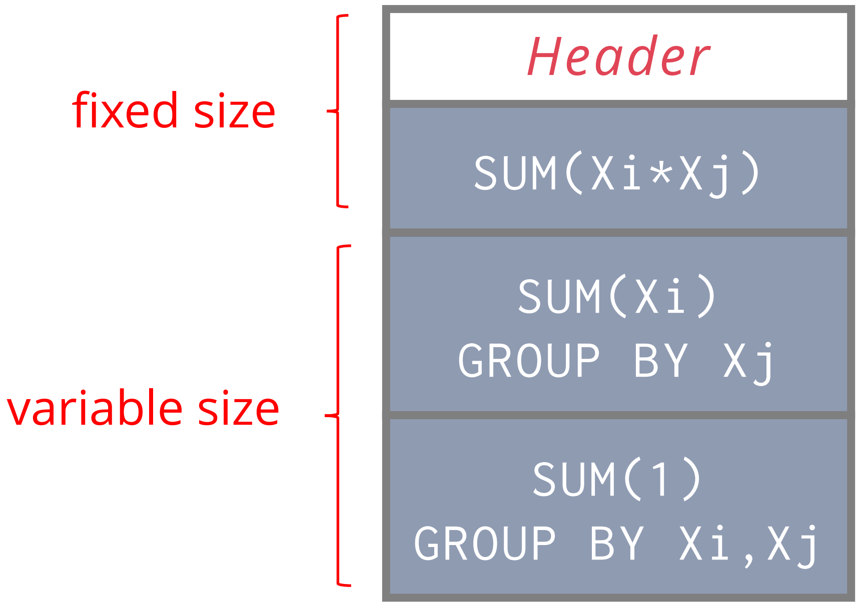
One contiguous memory chunk

No pointers to the outside of allocated memory

Max space for a * b is unknown beforehand

Solution:

Dry-run to compute max size of a * b



Memory representation of a ring value

Implementation Challenges (Cont.)

No automatic support for pushing SUM past joins

- **Solution:** rewrite/generate SQL queries to exploit factorization

Large `UPDATE` queries are **slow**

- Ex: imputing 1M values of one attribute can take few hours
- **Solution:** avoid in-place updates
  - Store imputed values in temporary tables
  - Recompute on-the-fly as `(R LEFT OUTER JOIN temp)`

Summary

- Data imputation within a DBMS
  - Ring computation + factorization + sharing
  - Further room for improvements in dealing with categorical values
- Using existing DBMSs for in-database ML ?
  - Performance looks promising
  - Few quirks complicate implementation