AI-Systems

Learning in a DBMS

(Database Management System)

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu

Why are we starting with Machine Learning in Database Systems?

Why do ML in a Database System

- **Proximity to Data:** minimize data movement
  - Avoid data duplication → inconsistency
- **Optimized for Data:** database systems are optimized for efficient **access** and **manipulation** of data.
  - Data layout, buffer management, indexing, …
  - Normalization can improve performance
  - Schema information can help in modeling
- **Predictions with Data:** trained models often used with data in the database.
  - Incorporate predictions into SQL queries
- **Security:** control who and what models have access to what data
  - leverage existing access control lists (ACLs)

Challenges of Learning in Database

- **Abstractions:** How does database expose data to alg.?
  - Some algorithms are a natural fit for existing abstractions
- **Access Patterns:** How does algorithm access data?
  - Sequentially, randomly, repeated scans
- **Cost Models and Learning:** How does database system aid in optimizing learning algorithm execution?
  - Exposing a broader set of **trade-offs**
- **Data Types:** Does data fit in the relational models?
  - Images, video, models

Database Systems and ML

- Database Systems supporting "Learning"
  - Data mining techniques heavily studied in DB community
    - Apriori algorithm for frequent item set (VLDB'94), widely cited
    - BIRCH large-scale clustering alg. (SIGMOD'96)
  - Most database systems have support for analytics and ML
    - Often specialized for particular techniques (e.g., SVM, decision tree,…)
- "Learning" for Database Systems (Later in Semester)
  - Cardinality estimation using statistical models
  - Dynamic programming for query optimization
  - Recent excitement around RL + Deep Learning in databases

Objectives For Today

- Review (some) Concepts in Database Systems
  - Relational Model
  - Data Independence
  - User defined aggregates
  - Out of core computation and latencies
    - Grace Hash Join Example
- This Weeks Reading
  - Review big ideas in each paper
  - Key technical details
  - What to look for when reading

Big Ideas in Database Systems

Relational Database Systems

- ***Logically*** organize data in **relations** (tables)

Sales relation:

Name	Prod	Price
Sue	iPod	$200.00
Joey	Bike	$333.99
Alice	Car	$999.00

Tuple (row)

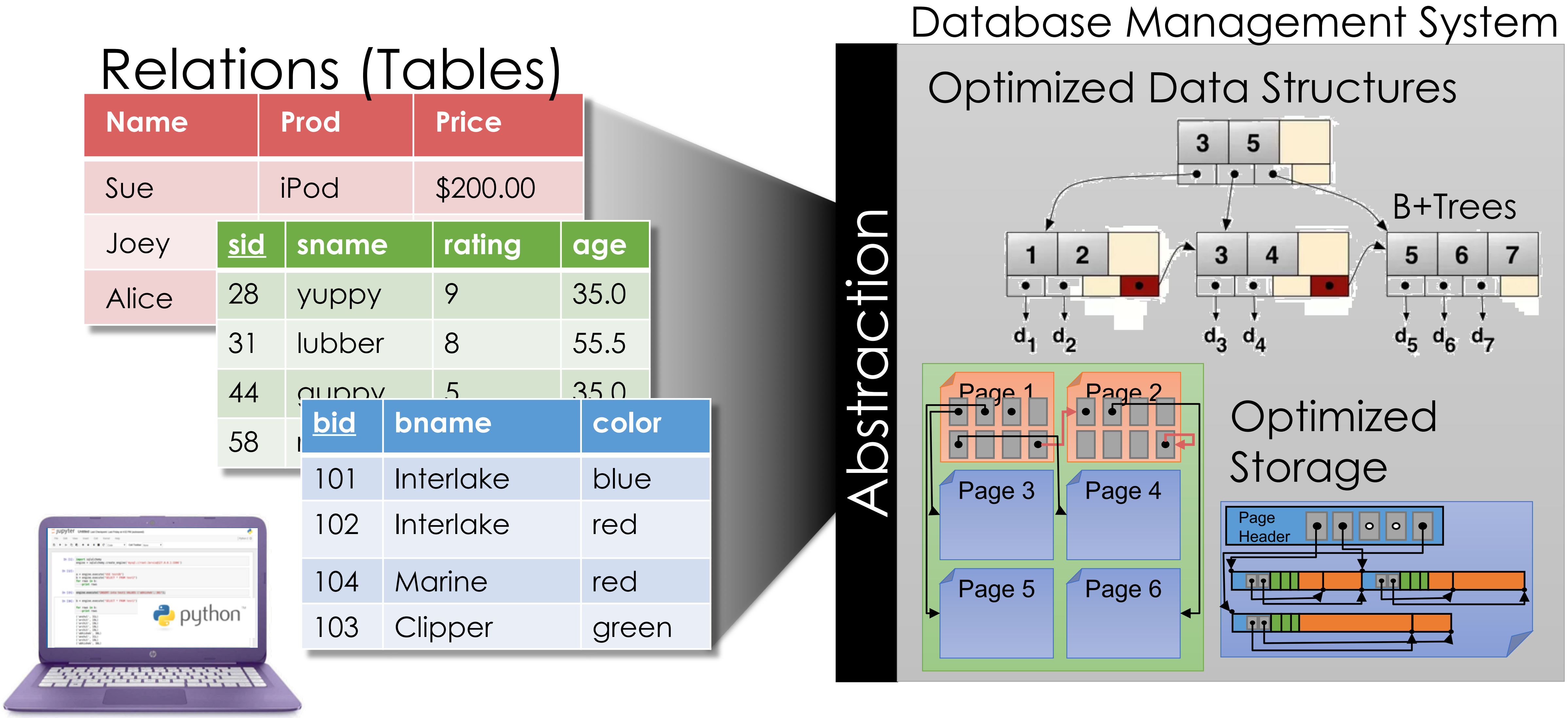
Attribute (column)

Describes *relationship:*

Name** purchased **Prod** at **Price.

How is data **physically** stored?

Relational Data **Abstraction**



<u>Physical Data Independence:</u>

Database management systems **hide how data is stored** from end user applications

→ System can **optimize storage** and **computation** without changing applications.

Big Idea in Data Structures
Data Systems &
Computer Science

Physical **Data Independence**

- Physical data layout/ordering is **determined by system**
  - goal of maximizing performance
- **Data Clustering**
  - Organize group of records to improve access efficiency
  - Example: grouped/ordered by key
- **Implications on Learning?**
  - Record ordering may depend on data values
  - Arbitrary ordering ≠ Random ordering

Relational Database Systems

- *Logically* organize data in **relations** (tables)
- Structured Query Language **(SQL)** to define, manipulate and compute on data.
  - A common language used by many data systems
  - Describes logical organization of data as well as computation

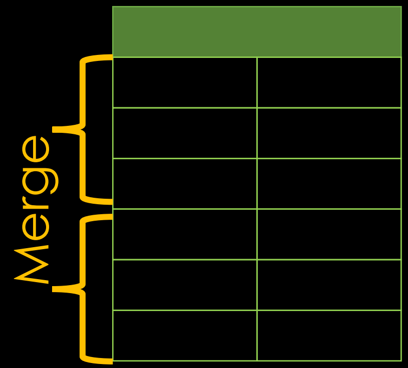
SQL is a **Declarative** Language

- **Declarative:** "Say ***what*** you want, not ***how*** to get it."
  - **Declarative Example:** *I want a table with columns "x" and "y" constructed from tables "A" and "B" where the values in "y" are greater than 100.00.*
  - **Imperative Example:** For each record in table "A" find the corresponding record in table "B" then drop the records where "y" is less than or equal to 100 then return the "x" and "y" values.
- **Advantages** of declarative programming
  - Enable the system to find the best way to achieve the result.
  - More compact and easier to learn for non-programmers (Maybe?)
- **Challenges** of declarative programming
  - System performance depends heavily on automatic optimization
  - Limited language (not Turing complete) → need extensions

User Defined Aggregates

- Provide a **low-level API** for defining functions that aggregate state across records in a table
  - Much like **fold** in functional Programming

```
CREATE AGGREGATE agg_name (…){
   # Initialize the state for aggregation.
   initialize(state) → state
   # Advance the state for one row.  Invoked repeatedly.
   transition(state, row) → state
   # Compute final result.
   terminate(state) → result
   # (Optional) Merge intermediate states from parallel executions.
   merge(state, state) → state
}
```



Closed Relational Model and Learning

- All operations on tables produce tables…
- Training a model on a table produces?
  - A row containing a model
  - A table containing model weights
  - An (infinite) table of predictions
    - MauveDB: Supporting Model-based User Views in Database Systems
- Predictions as views
  - Opportunity to **index** predictions
  - Relational operations to manipulate predictions

Out-of-core Computation

- Database systems are typically designed to operate on **databases larger than main memory** (big data?)
- Algorithms must manage **memory buffers** and **disk**
  - Page level memory buffers
  - Sequential reads/writes to disk
- Understand **relative costs** of memory vs disk

Reasoning about Memory Hierarchy

Latency Numbers Every Programmer Should Know --Jeff Dean

L1 Cache	**0.5**	**ns** (few clock cycles)
L2 Cache	**7**	**ns**
Main Memory	**100**	**ns**
Read 1MB from RAM (Seq.)	**250K**	**ns**
Read 1MB SSD (Seq.)	**1M**	**ns** (1ms)
Read 1MB Disk (Seq.)	**20M**	**ns** (20ms)

Reasoning about Memory Hierarchy

Latency Numbers Every Programmer Should Know	***<u>Human Readable</u>***	
L1 Cache	**1 second**	
L2 Cache	**14 seconds**	**Database Systems**
Main Memory	**3.3 minutes**	Page Buffers
Read 1MB from RAM (Seq.)	**5.8 days**	
Read 1MB SSD (Seq.)	**23 days**	Sequential Read/Write
Read 1MB Disk (Seq.)	**1.3 years**	

Example Out-of-Core Alg.:
Grace Hash Join

Grace Hash Join

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

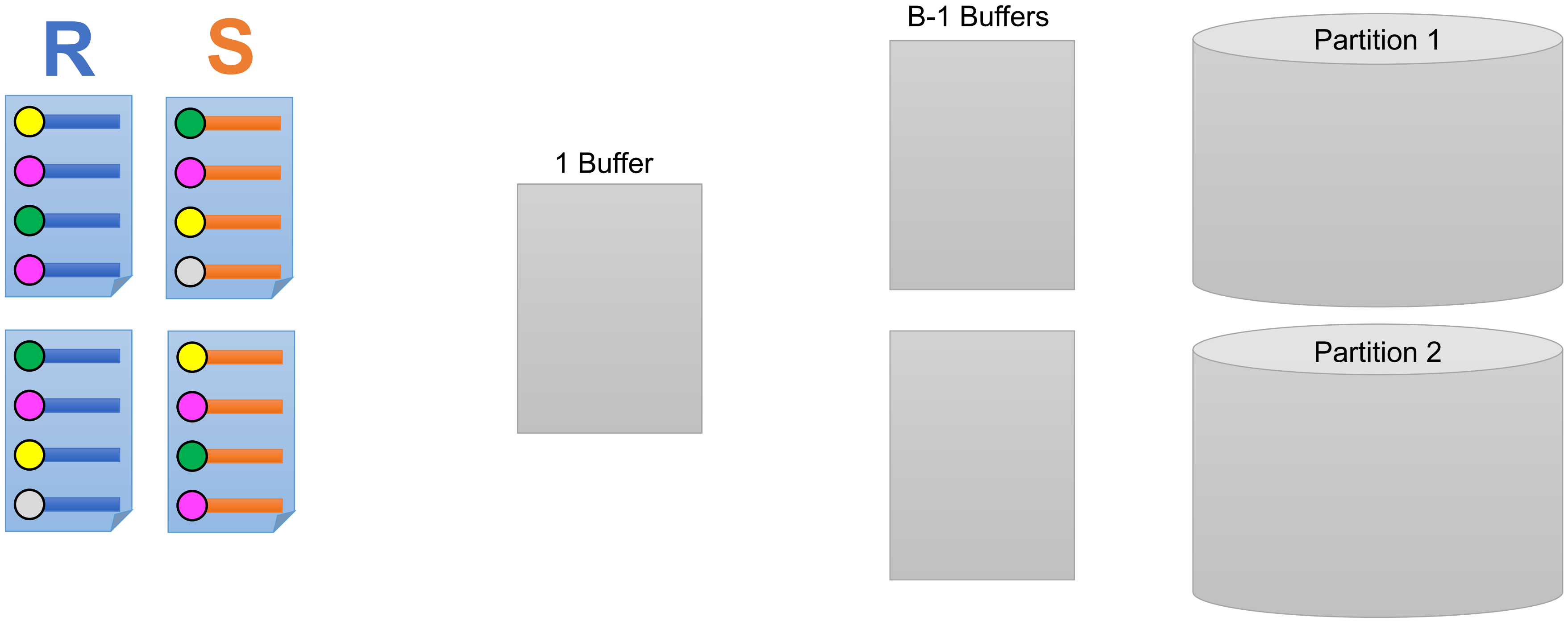
- Requires equality predicate θ:
  - Works for **Equi-Joins** & **Natural Joins**

- Two Stages:
  - **Partition** tuples from R and S by join key
    - all tuples for a given key in same partition
  - **Build & Probe** a separate hash table for each partition
    - Assume **partition** of smaller rel. fits in memory
      - Recurse if necessary…

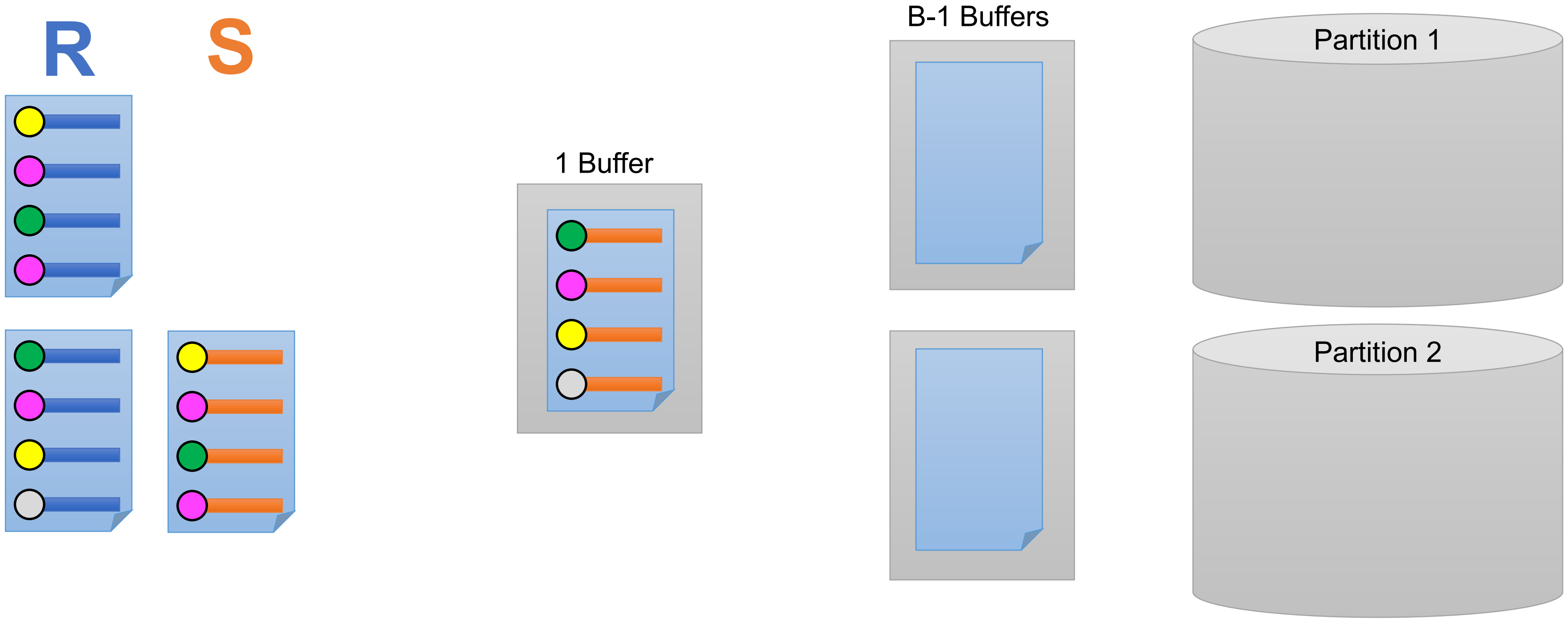
Divide

Conquer

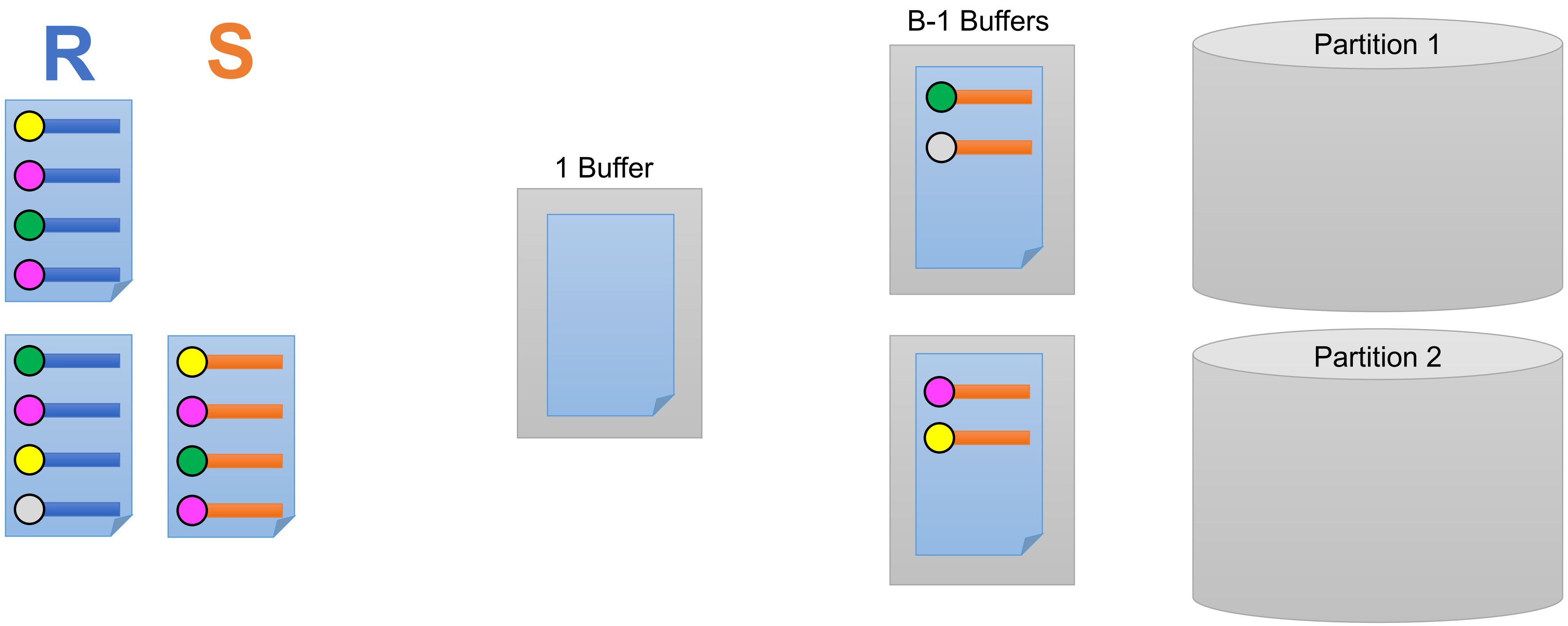
Grace Hash Join: ***Partition***



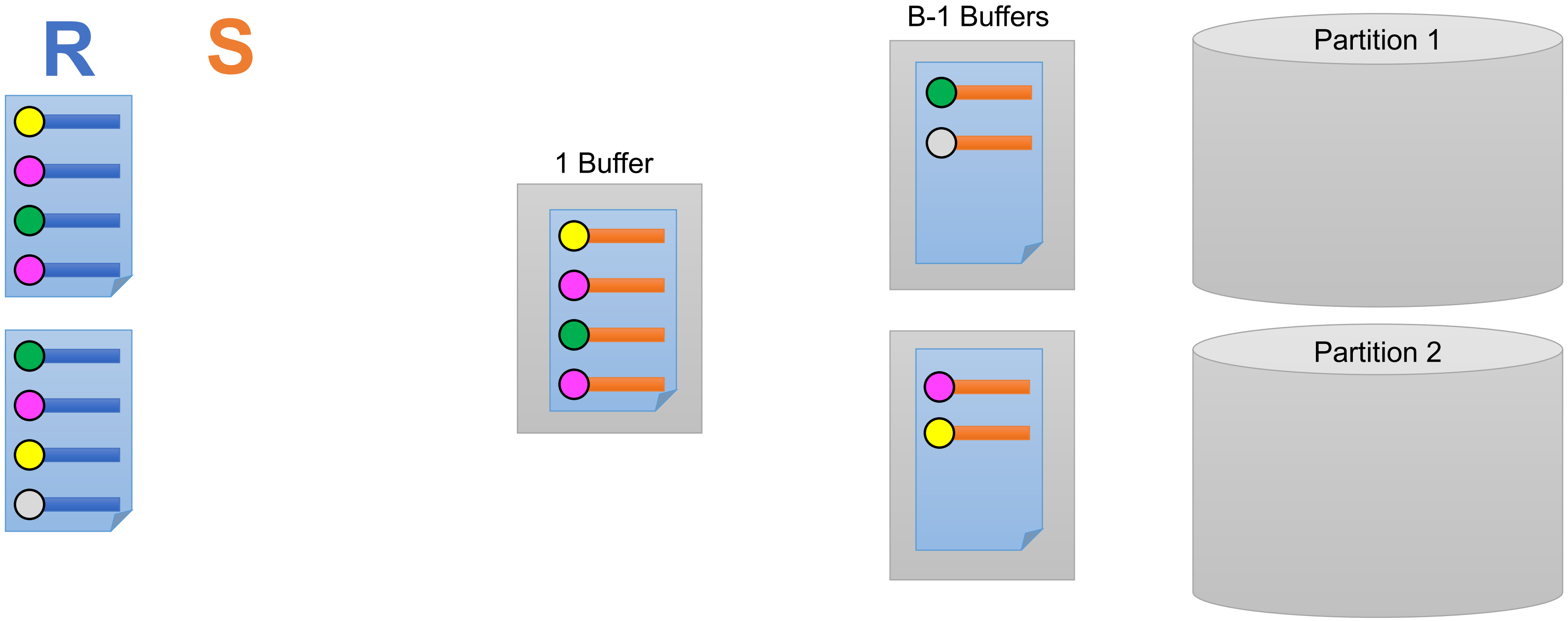
Grace Hash Join: ***Partition***



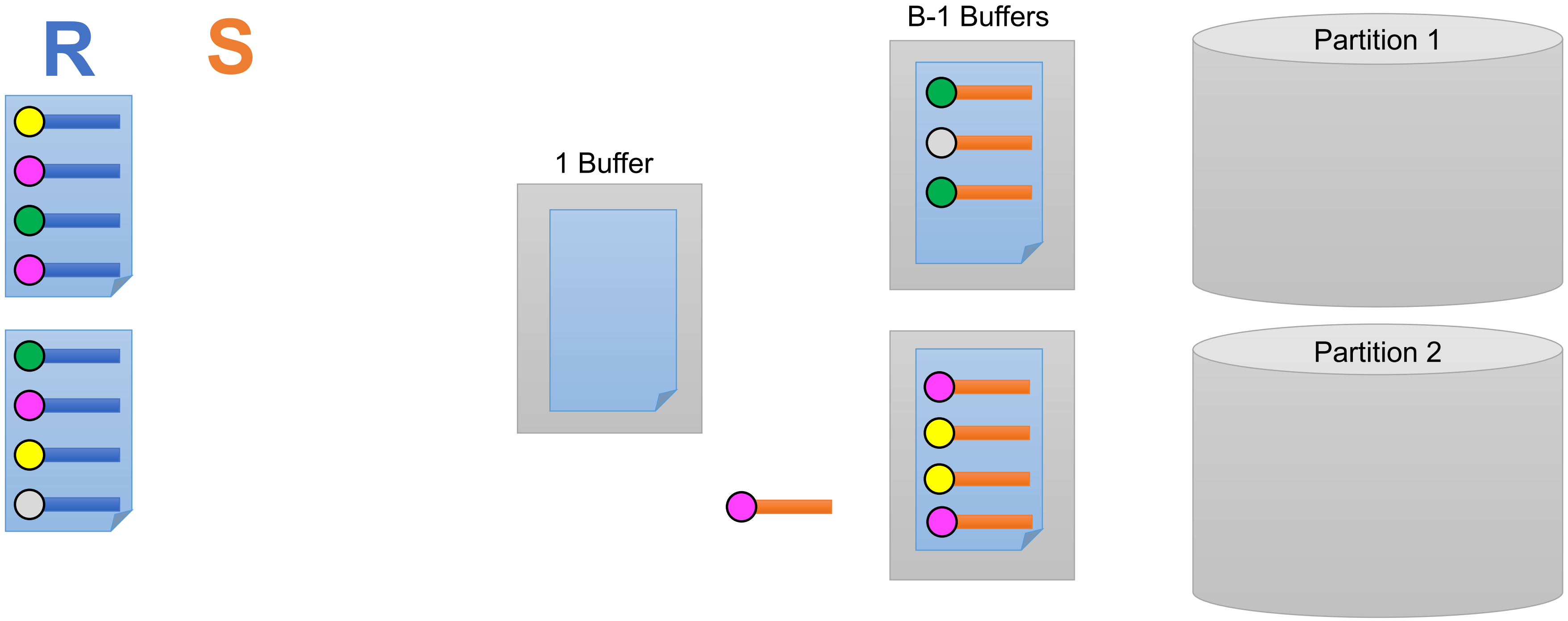
Grace Hash Join: ***Partition***



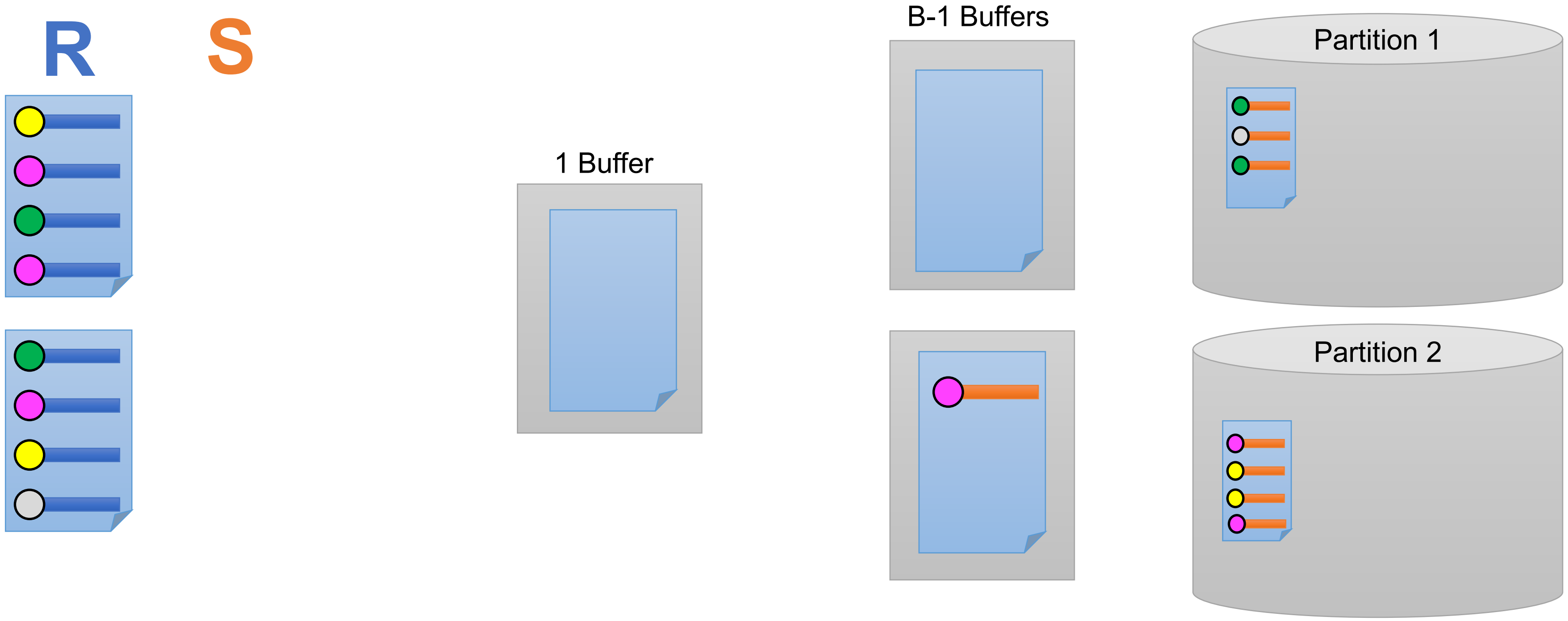
Grace Hash Join: ***Partition***



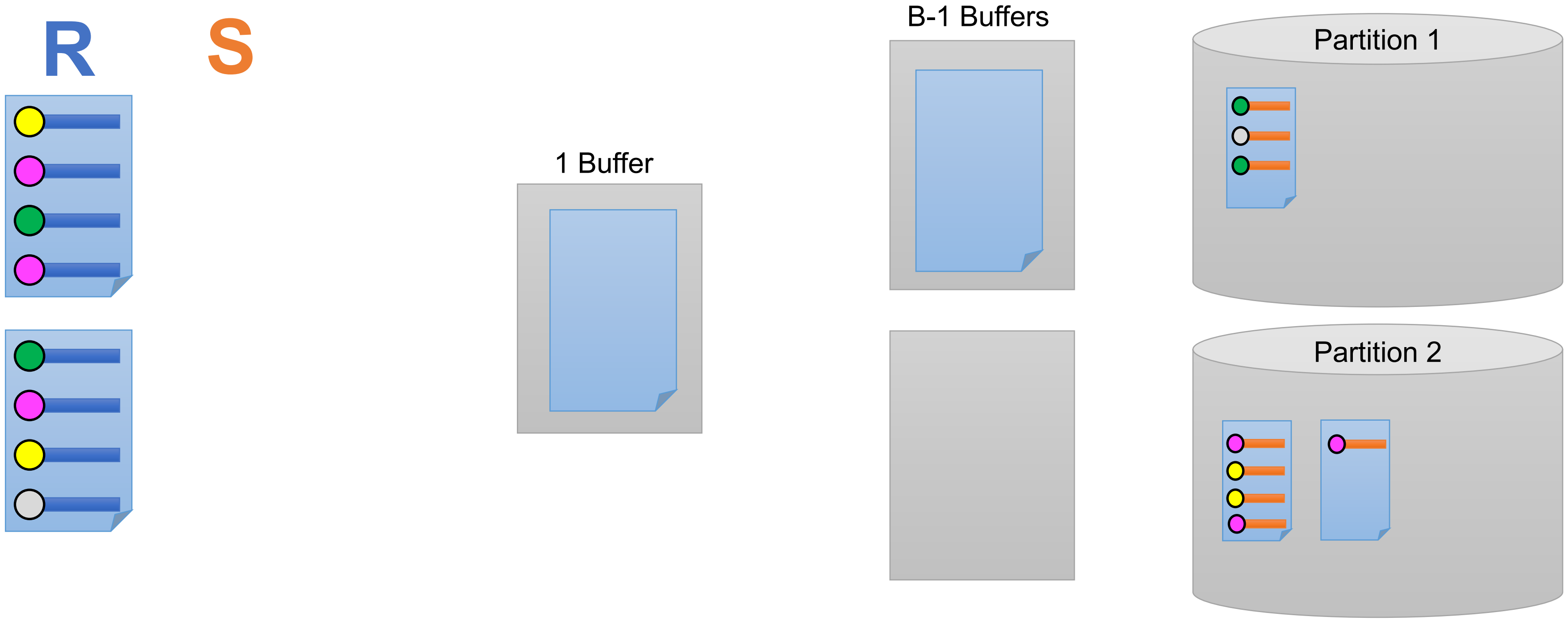
Grace Hash Join: ***Partition***



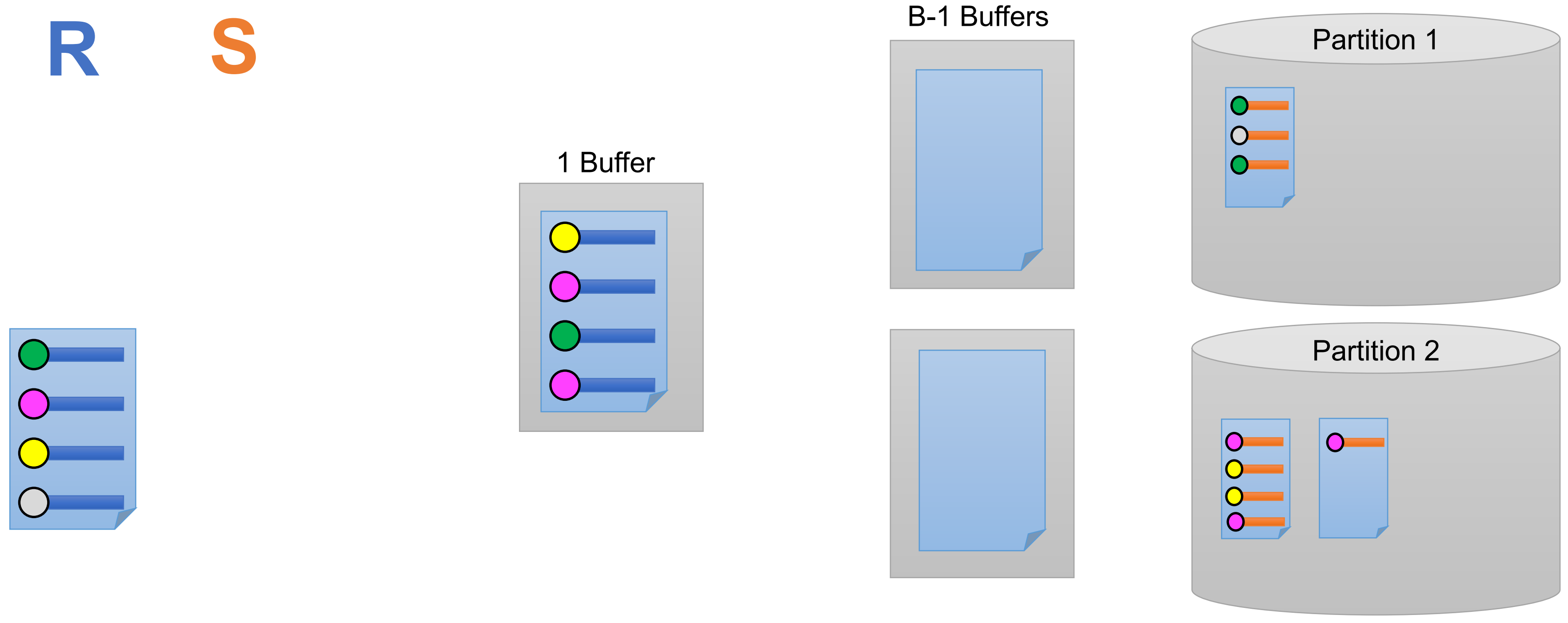
Grace Hash Join: ***Partition***



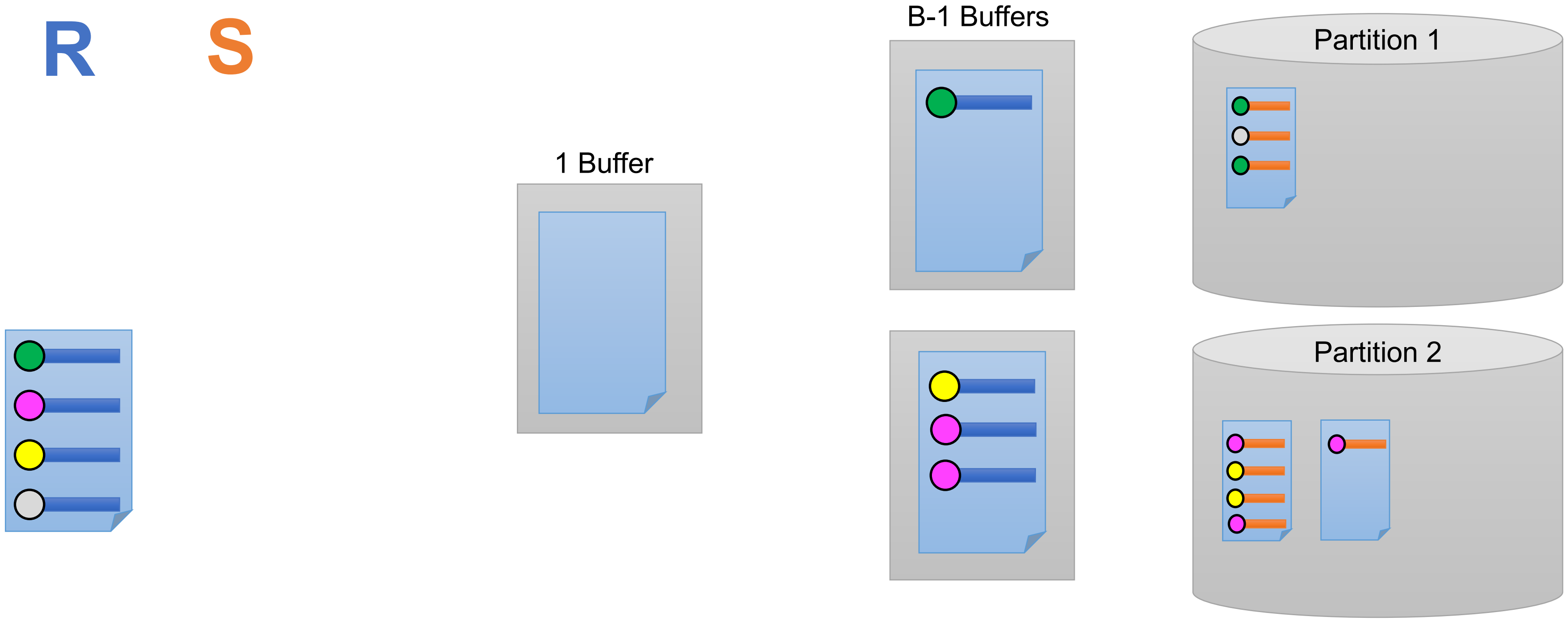
Grace Hash Join: ***Partition***



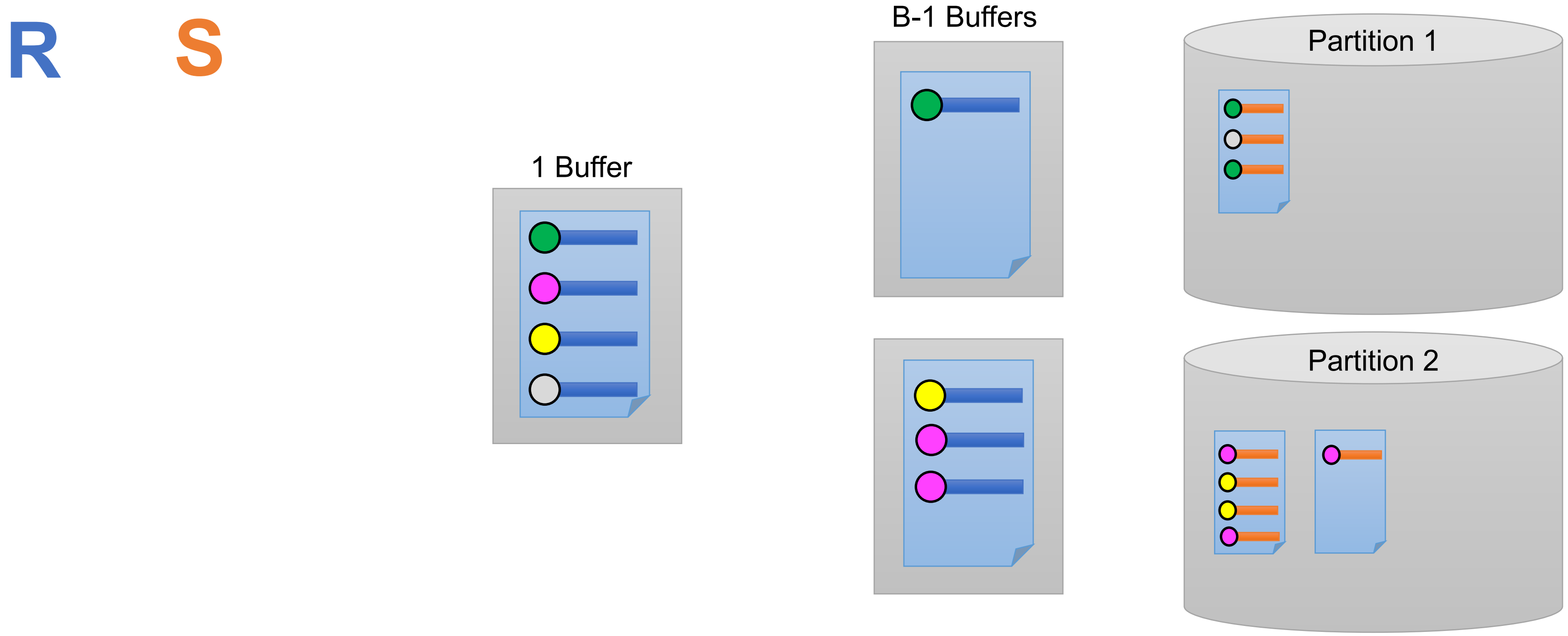
Grace Hash Join: ***Partition***



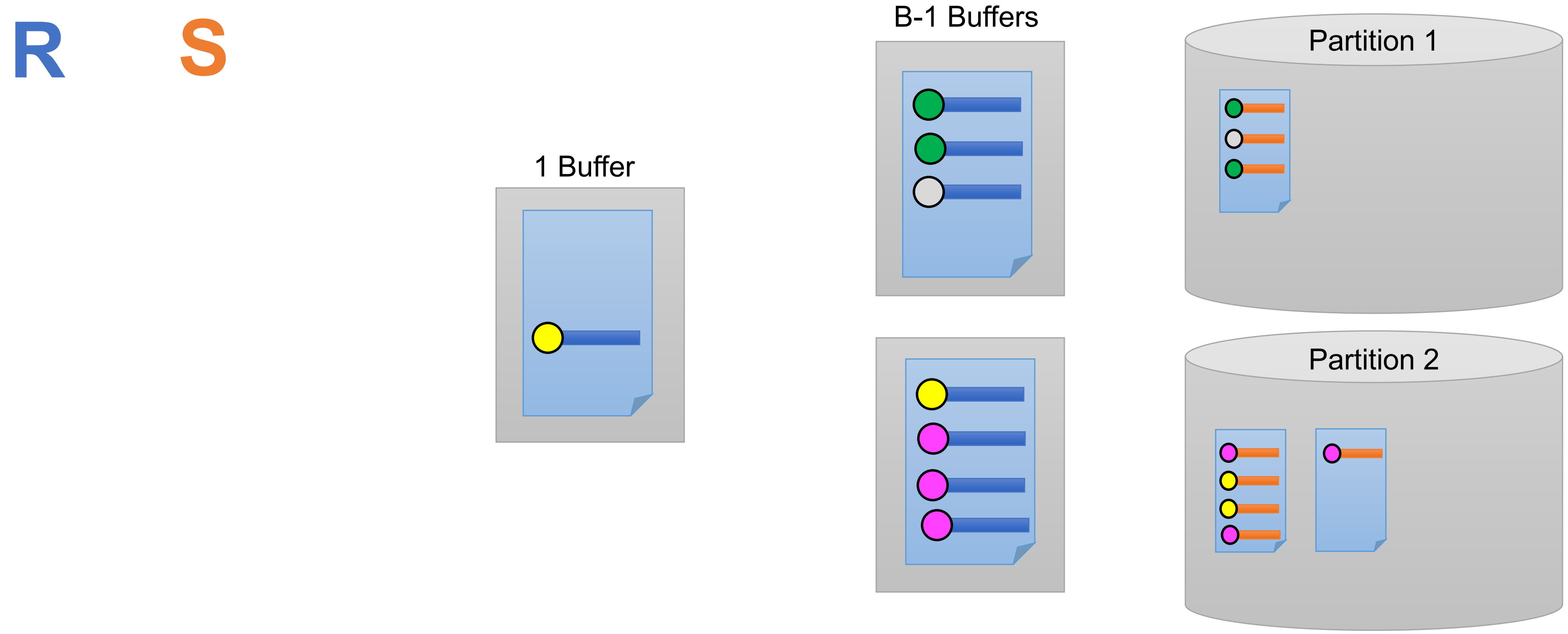
Grace Hash Join: ***Partition***



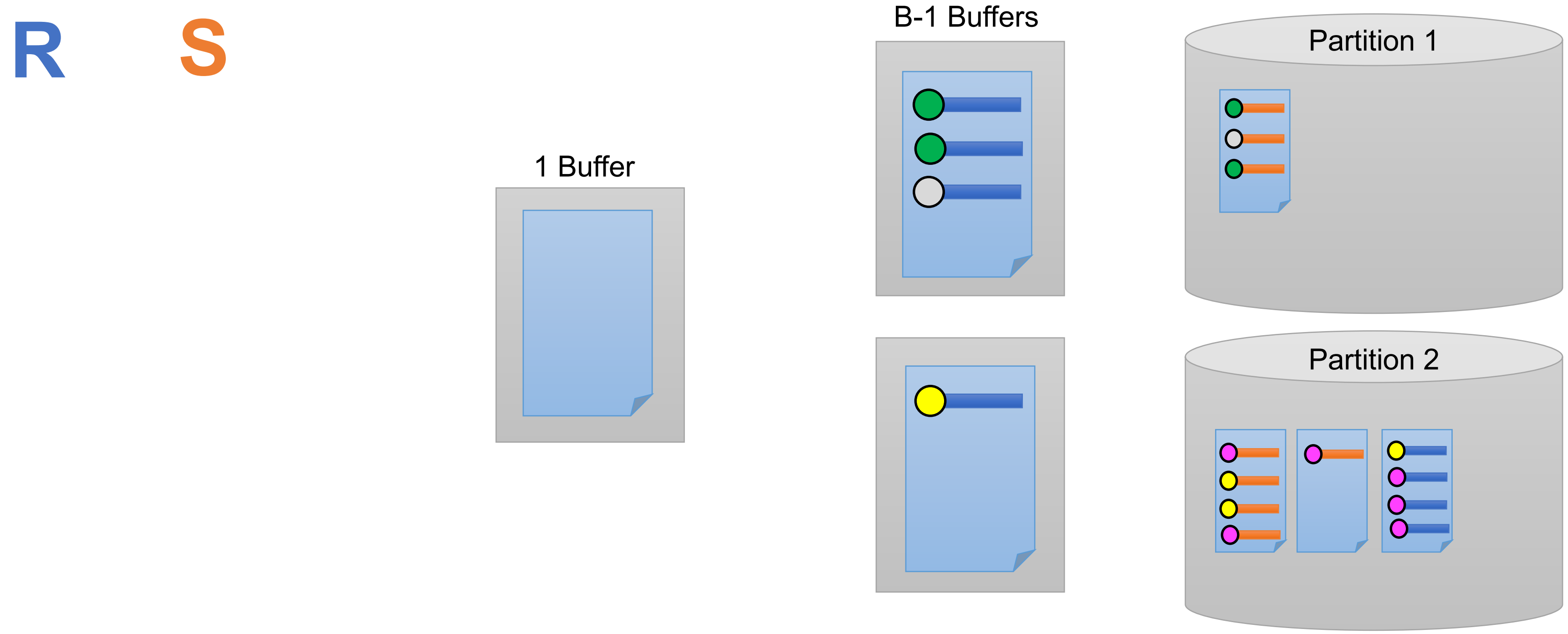
Grace Hash Join: ***Partition***



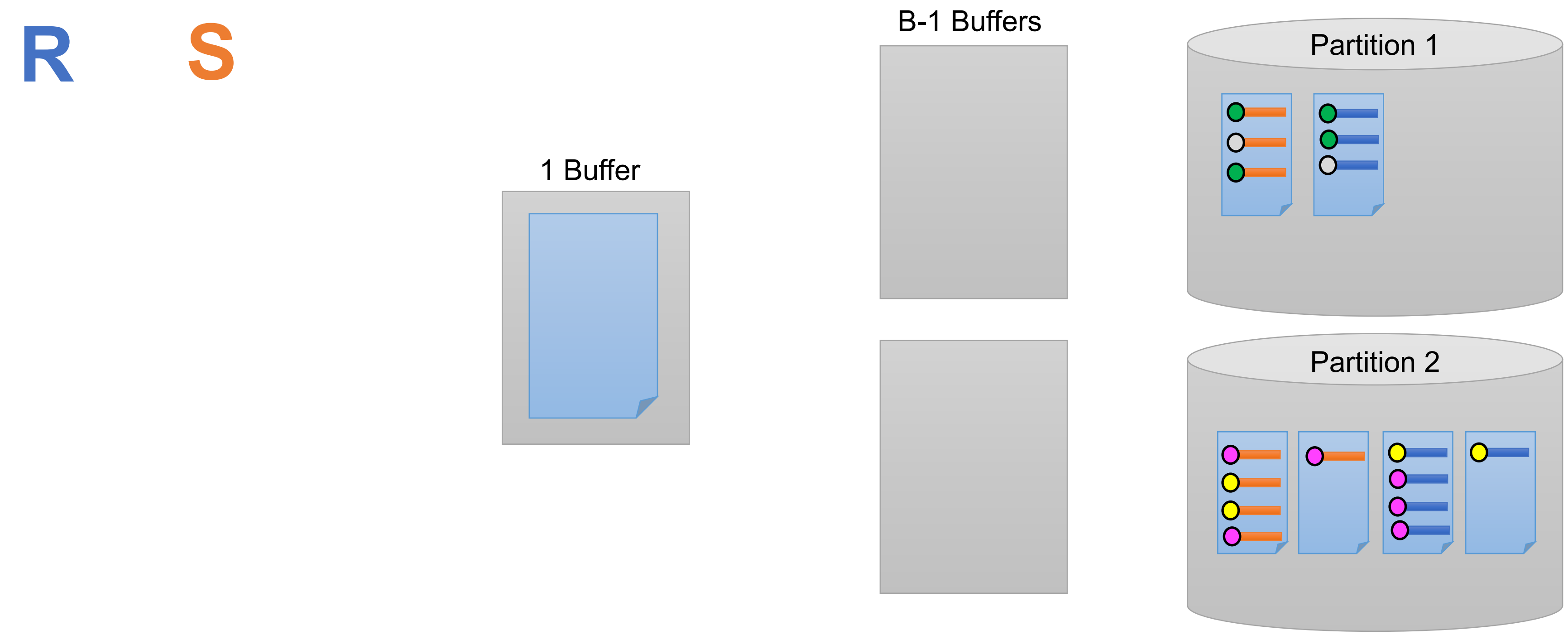
Grace Hash Join: ***Partition***



Grace Hash Join: ***Partition***

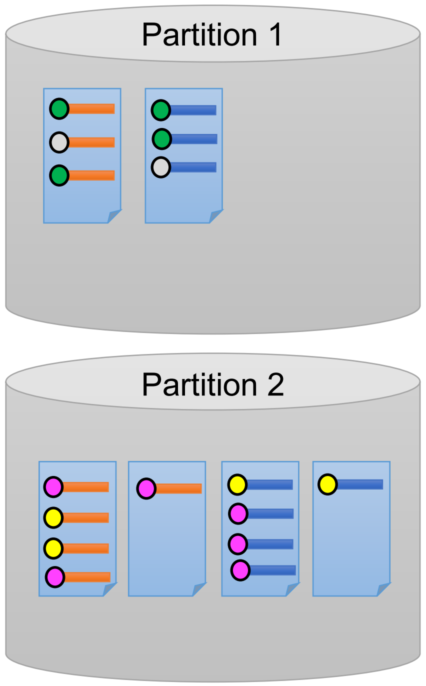


Grace Hash Join: ***Partition***

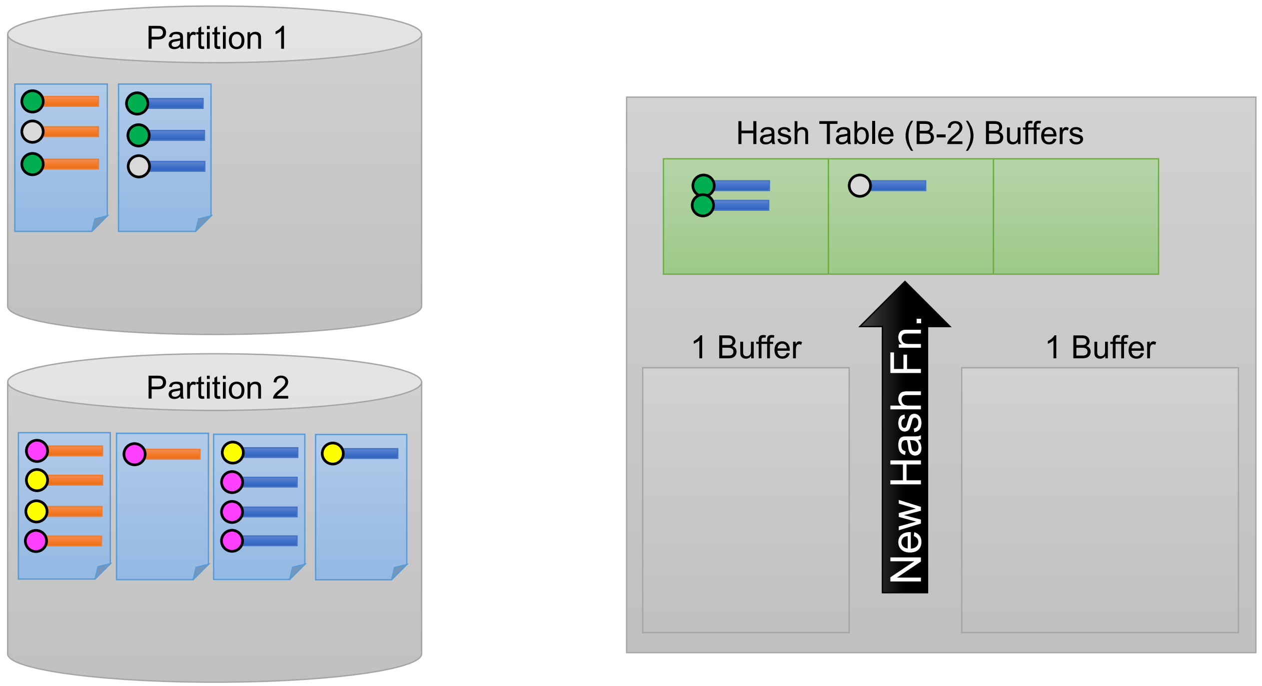


Post Hash Partitioning

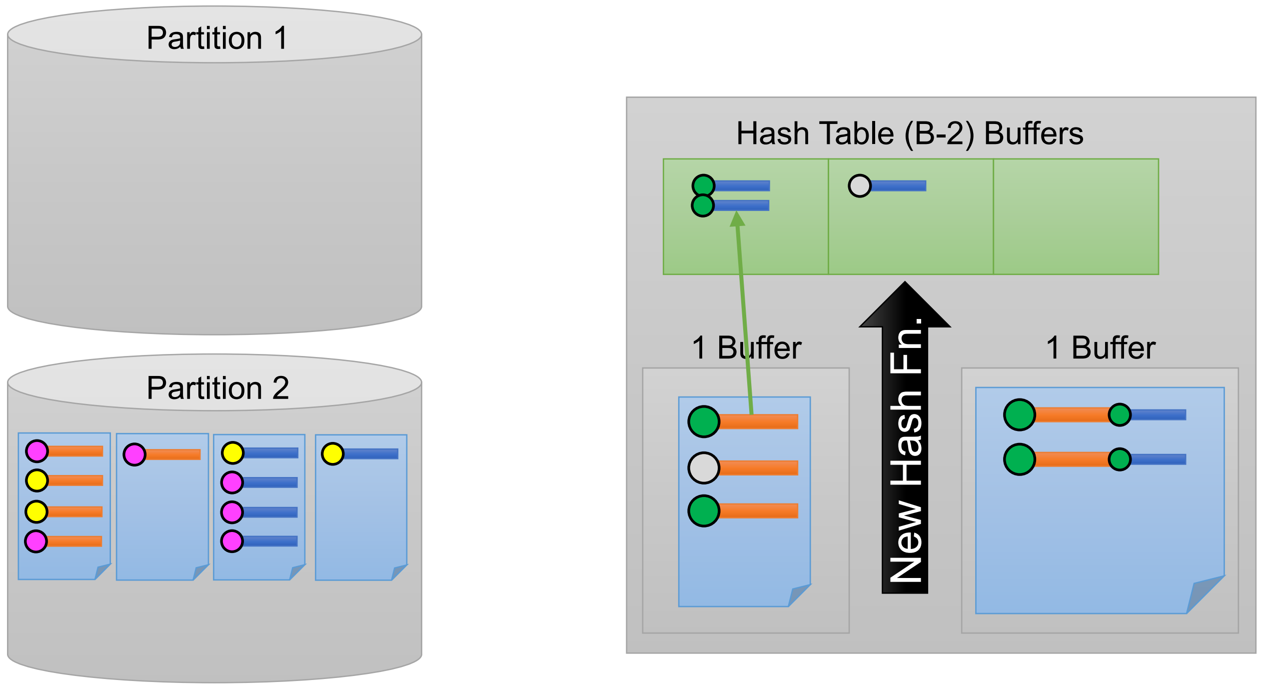
- Observe how memory buffers are directly managed
  - Paged to disk when full …
- Each key is assigned to one partition
  - e.g., green keys in partition1
- Sensitive to key Skew
  - Fuchsia Key



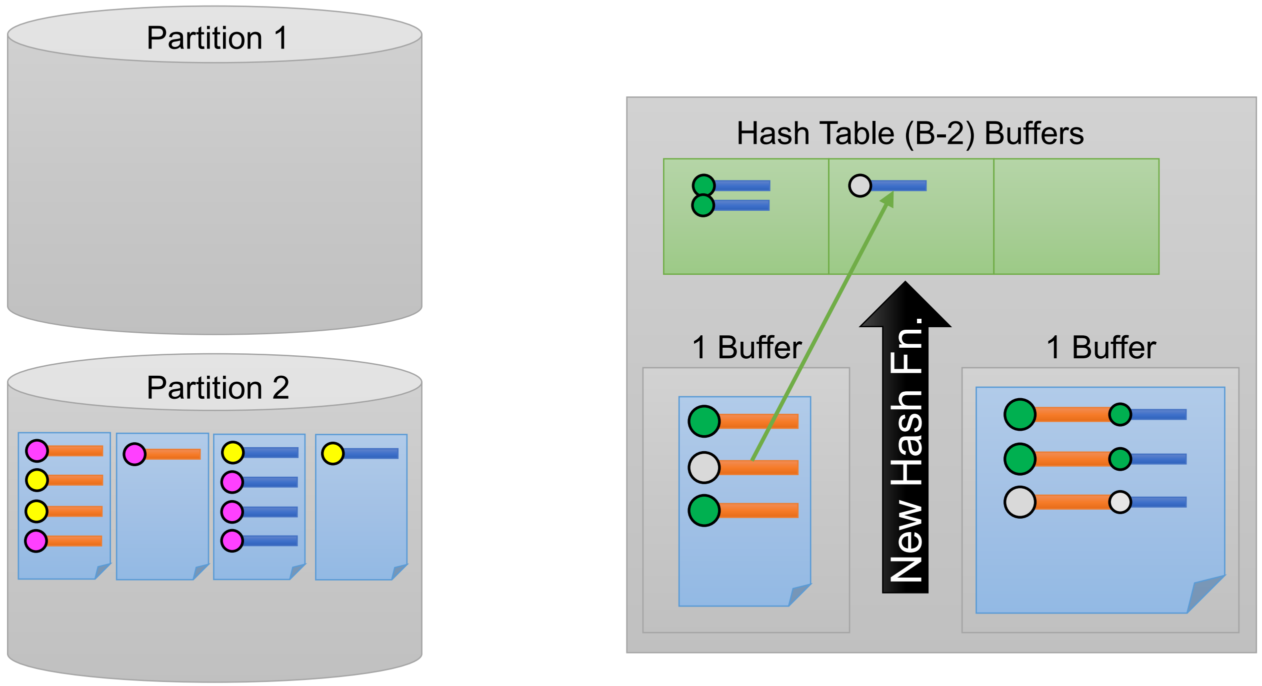
Grace Hash Join: ***Build*** & ***Probe***



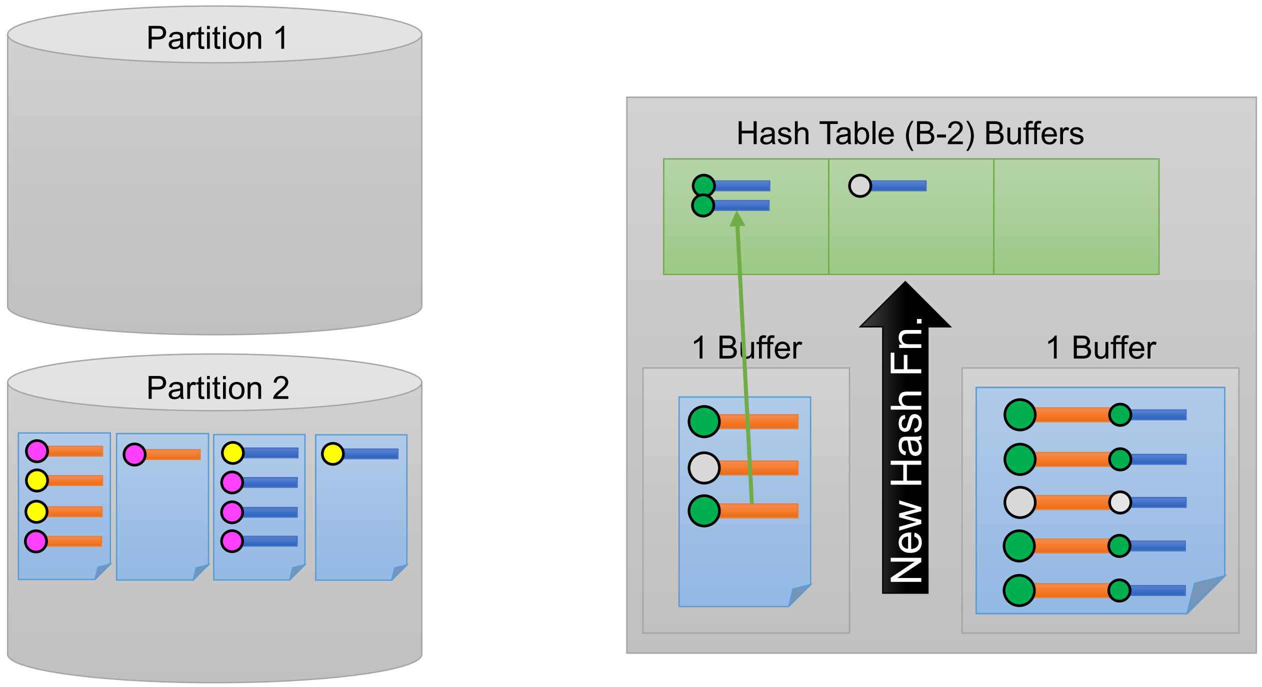
Grace Hash Join: ***Build*** **&** ***<u>Probe</u>***



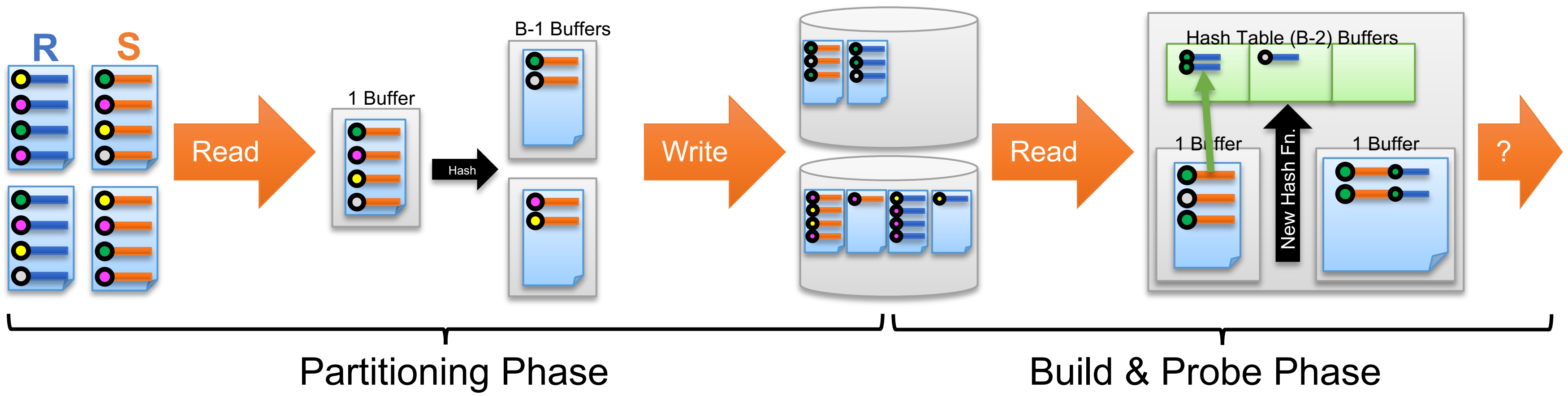
Grace Hash Join: ***Build*** & ***<u>Probe</u>***



Grace Hash Join: ***Build*** & ***<u>Probe</u>***

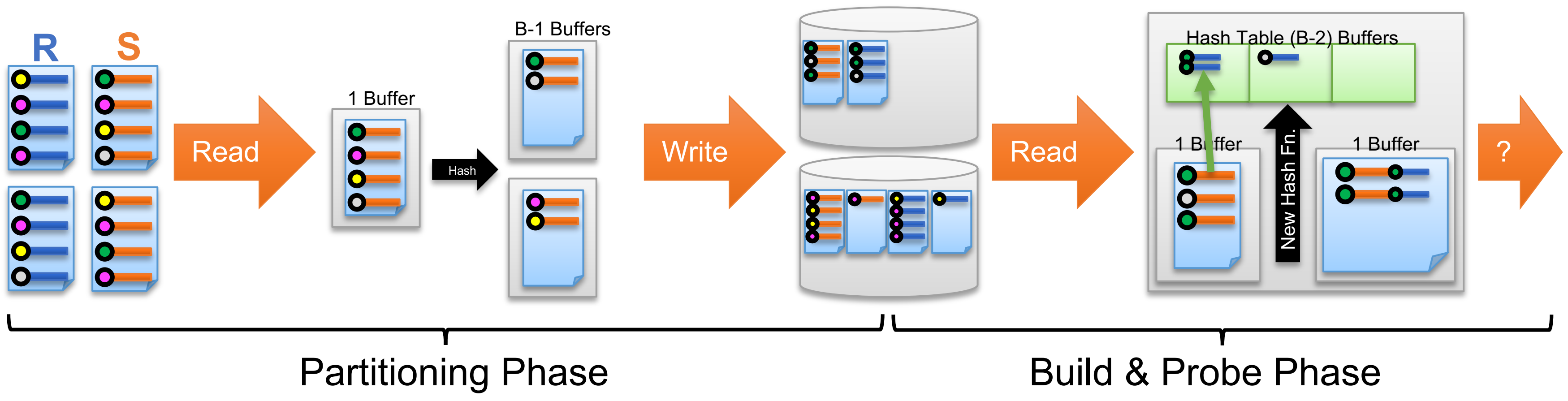


Cost of Hash Join



- Partitioning phase: read+write both relations
  ⇒ 2([R]+[S]) I/Os
- Matching phase: read both relations, forward output
  ⇒ [R]+[S]
- Total cost of 2-pass hash join = 3([R]+[S])

Cost of Hash Join



Memory Requirements?

➢ Build hash table on R with uniform partitioning

$\Rightarrow$ **Partitioning Phase** divides R into (**B**-1) runs of size [**R**] / (**B**-1)

$\Rightarrow$ **Build Phase** requires each [**R**] / (**B**-1) < (**B**-2)

$\Rightarrow$ **R** < (**B**-1) (**B**-2) ≈ B^2

This weeks reading

Reading for the Week

Two Chris Ré Papers. One of the leaders in DB+ML research

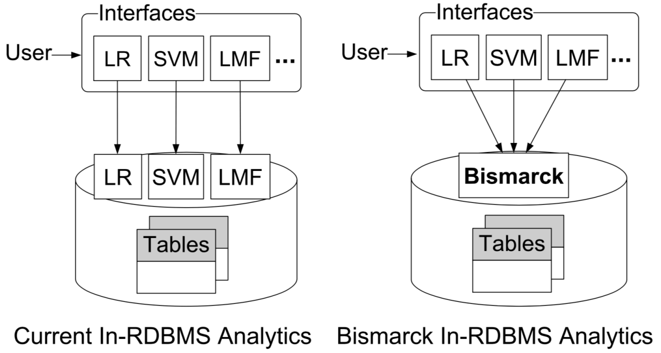
- Towards a Unified Architecture for in-RDBMS Analytics
  - SIGMOD'12,
  - Support **generic learning** within existing DBMS abstraction
- Materialization Optimizations for Feature Selection Workloads
  - SIGMOD'14 (Best Paper)
  - Optimize **feature engineering** workloads by exploiting **redundancy**
- Learning Generalized Linear Models Over Normalized Data
  - SIGMOD'15
  - Pushing learning through **joins** on **normalized data**

Note these are "older" papers but they cover big ideas

Towards a Unified Architecture for in-RDBMS Analytics

Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré

Towards a Unified Architecture for in-RDBMS Analytics



- **Context:** database system vendors building **specialized** in DB implementations of ML techniques.
  - Slow and costly to add support for new models/algorithms
  - Many ML techniques leverage (convex) empirical risk minimization
- **Key Idea:** Many ML techniques can be reduced to **mathematical programming** and there is a single solver (**IGD**) that fits existing database system abstractions (**UDAs**)
- **Contribution:** this paper demonstrates the advantages of leveraging existing optimized abstractions for learning

Challenges Addressed

- Mapping IGD to User Defined Aggregates (UDA)
- Affects of **data ordering** on convergence
  - Data often stored in a pathological ordering (e.g., by label)
- Parallelization of Incremental Algorithm
  - Adopt two standard solutions (model averaging, Hogwild!)

What is the difference between **Incremental** vs **Stochastic** Gradient Descent?

Short Answer: Stochastic gradient descent is a form of incremental gradient descent

- Incremental Gradient Descent
  - **Formally:** taking single gradient steps for each element of a decomposable loss
  - Ordering of gradient terms is arbitrary
- Stochastic Gradient Descent
  - **Formally:** *sampling* from the gradient of the empirical loss
  - *Sample* data and compute gradient of loss on sample
  - Today people often refer to incremental gradient methods as **stochastic gradient descent**

Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey

Mapping IGD to User Defined Aggregates (UDA)

```
CREATE AGGREGATE bismarck (…){
  initialize(args) → state:
      randomly initialize model weights
  transition(state, row) → state:
      single gradient update
```

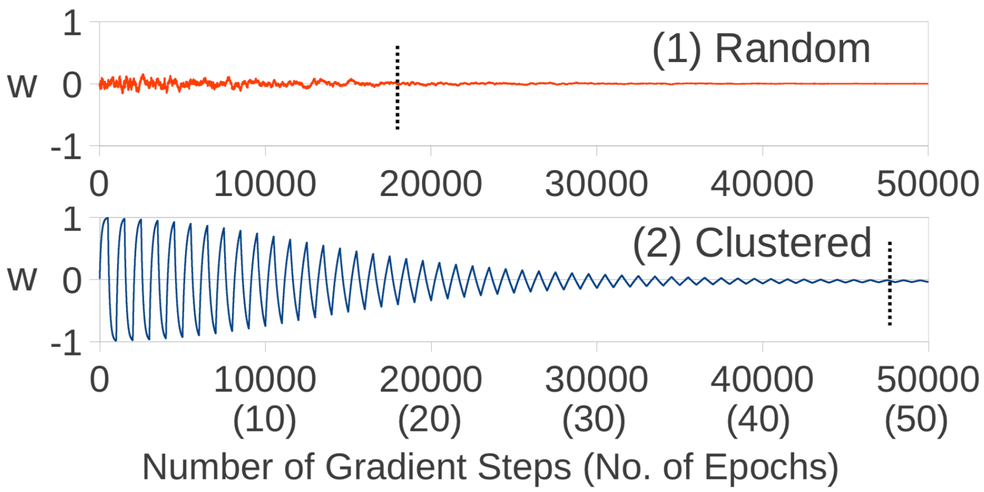
$$w^{(k+1)} \leftarrow w^{(k)} - \alpha_k \nabla L\left(\mathrm{row},\, w^{(k)}\right)$$

```
  terminate(state) → result
      return current model for epoch
  merge(state, state) → state
      used for parallel model averaging
}
```

- State contains:
  - Model weights, k, …
- Invoked repeatedly
  - Once per epoch
  - Bismarck stored procedure
- Termination cond.
  - Similar to IGD

Data Ordering Issues

- Data indexed/clustered on key feature or even the label
  - **Example:** predicting customer churn → data is partitioned by active customers and cancelled customers
    - Why?
- May slow down convergence:



Random Order

Sorted by **label**

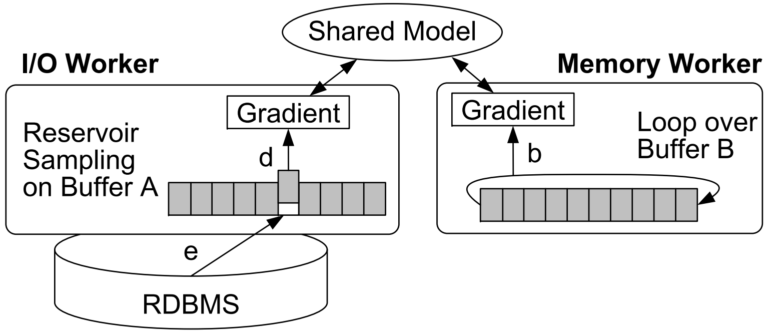
Data Order Solutions

Shuffle data

- **on each epoch** (pass through data): Closest to stochastic gradient alg.
  - Expensive data movement and duplication
- **Once:** good compromise but requires data movement and dup.

Sample data

- **single reservoir sample per pass**
  - Train on less data per scan → slower convergence
- **multiplexed reservoir sampling**
  - Concurrently training on sample and raw data streams



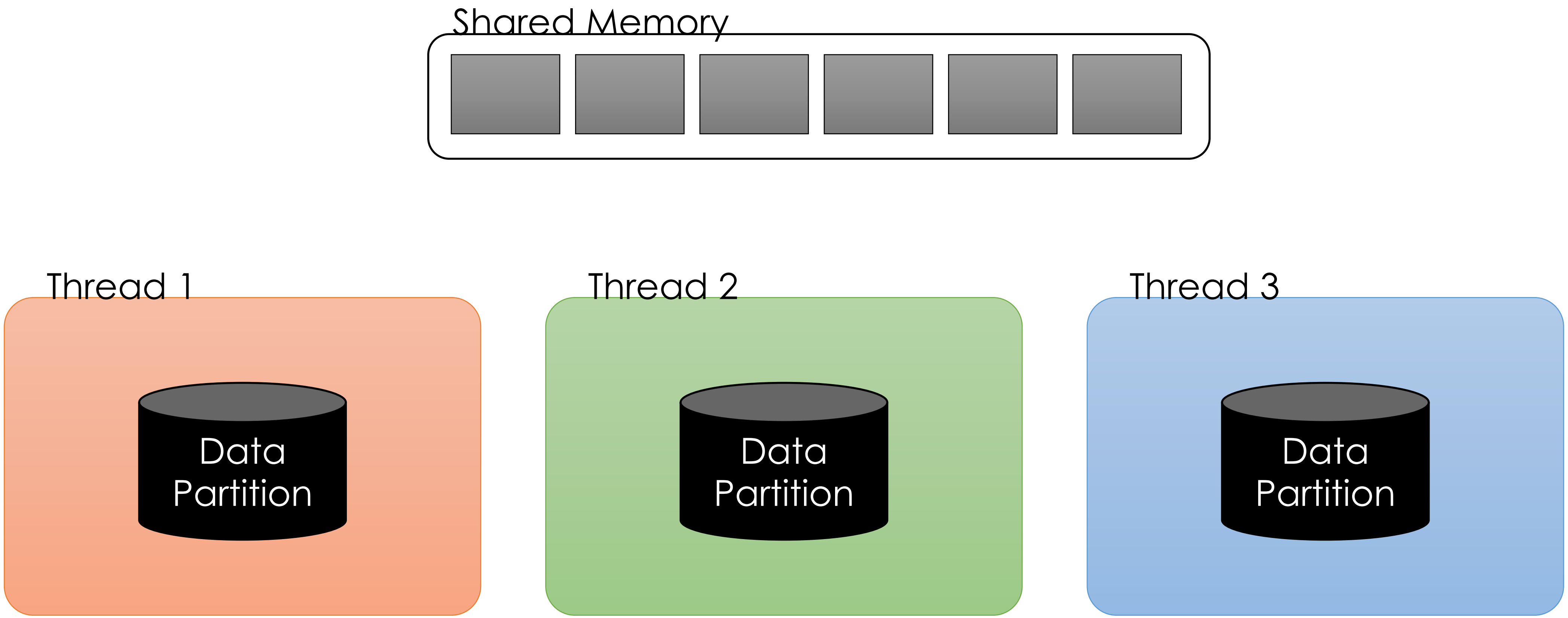
Parallelization

- **Pure UDA Version:** Primal (model) averaging
  - leverage merge operation
  - Appeals to result by Zinkevich* (requires iid data, convex loss, …)
  - Doesn't work as well in practice
- **Shared Memory UDA***
  - **Consistent (Atomic IG)**: atomic compare and swap for updates
    - Consistent but limited parallelism + bus traffic and branch misses
  - **No locks (Hogwild!™)**: write to memory and allow races
    - Word writes within cache lines are atomic (either old or new version wins)

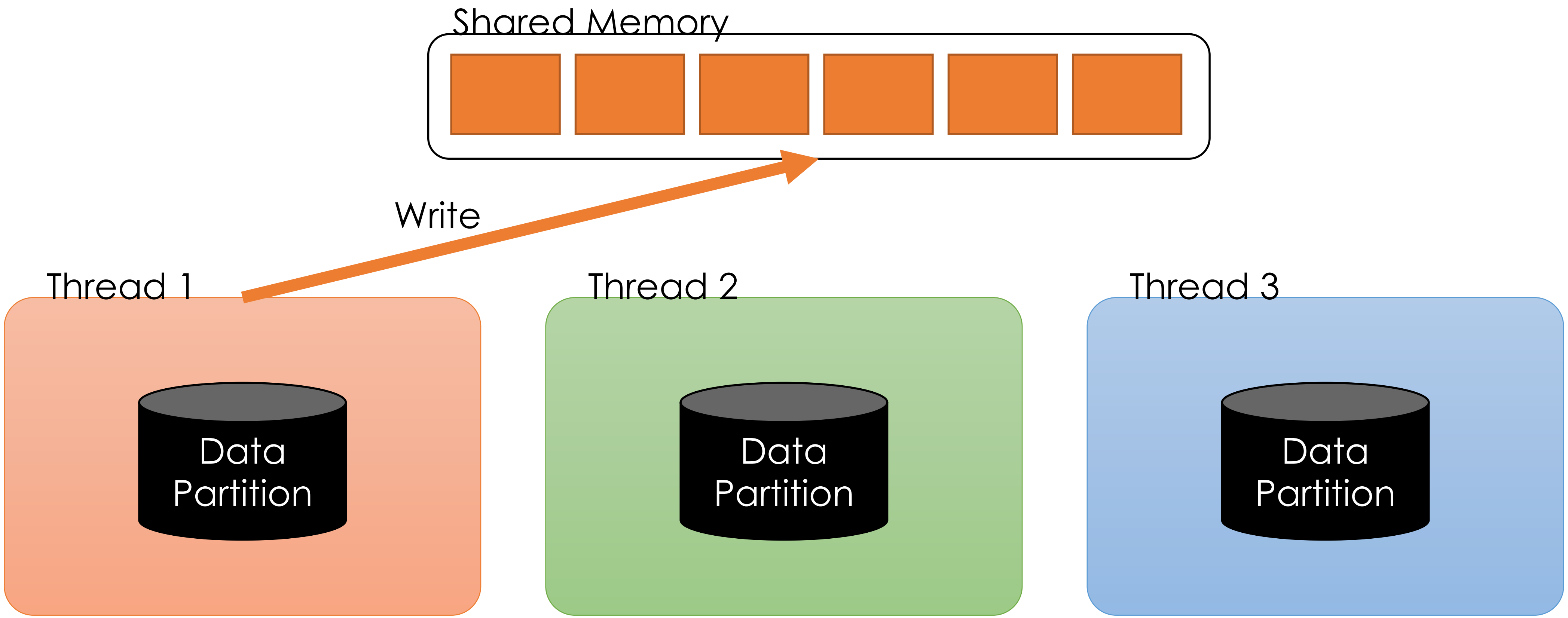
*Implications on distributed databases?

*[Parallelized Stochastic Gradient Descent](#)

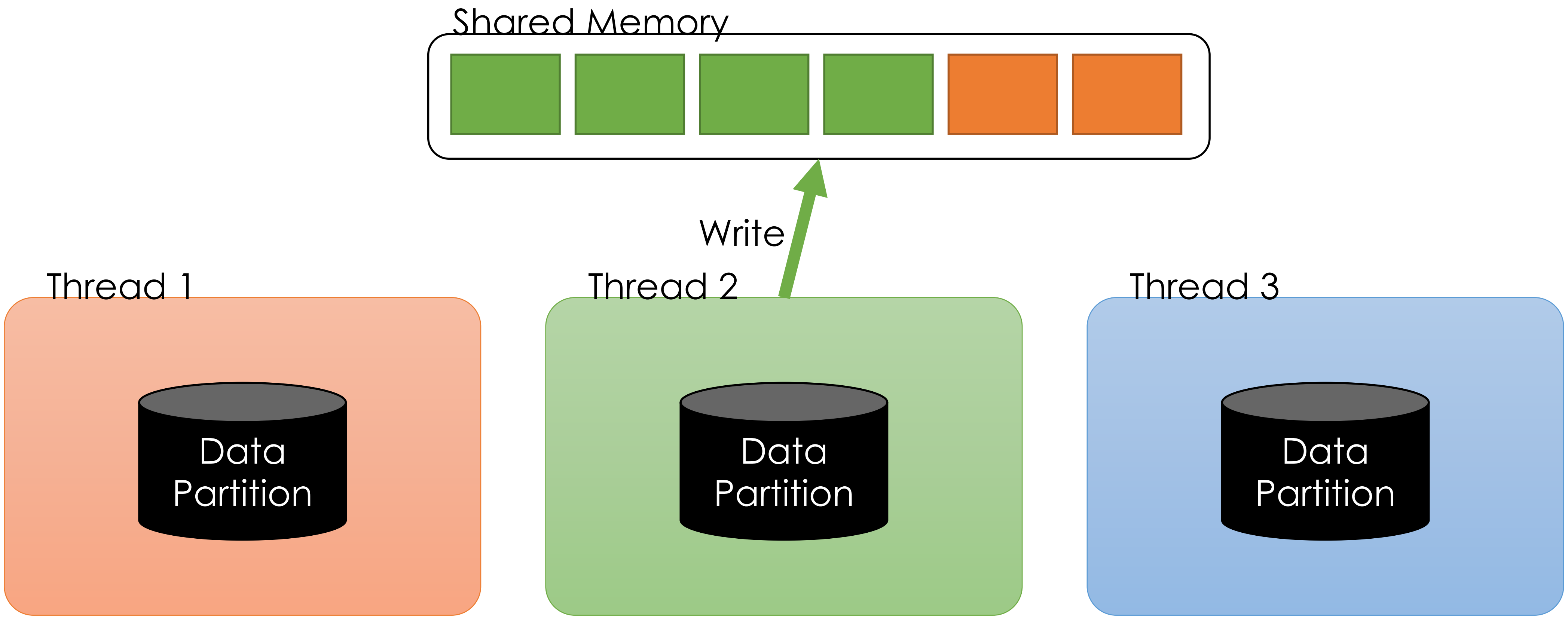
Hogwild! Algorithm



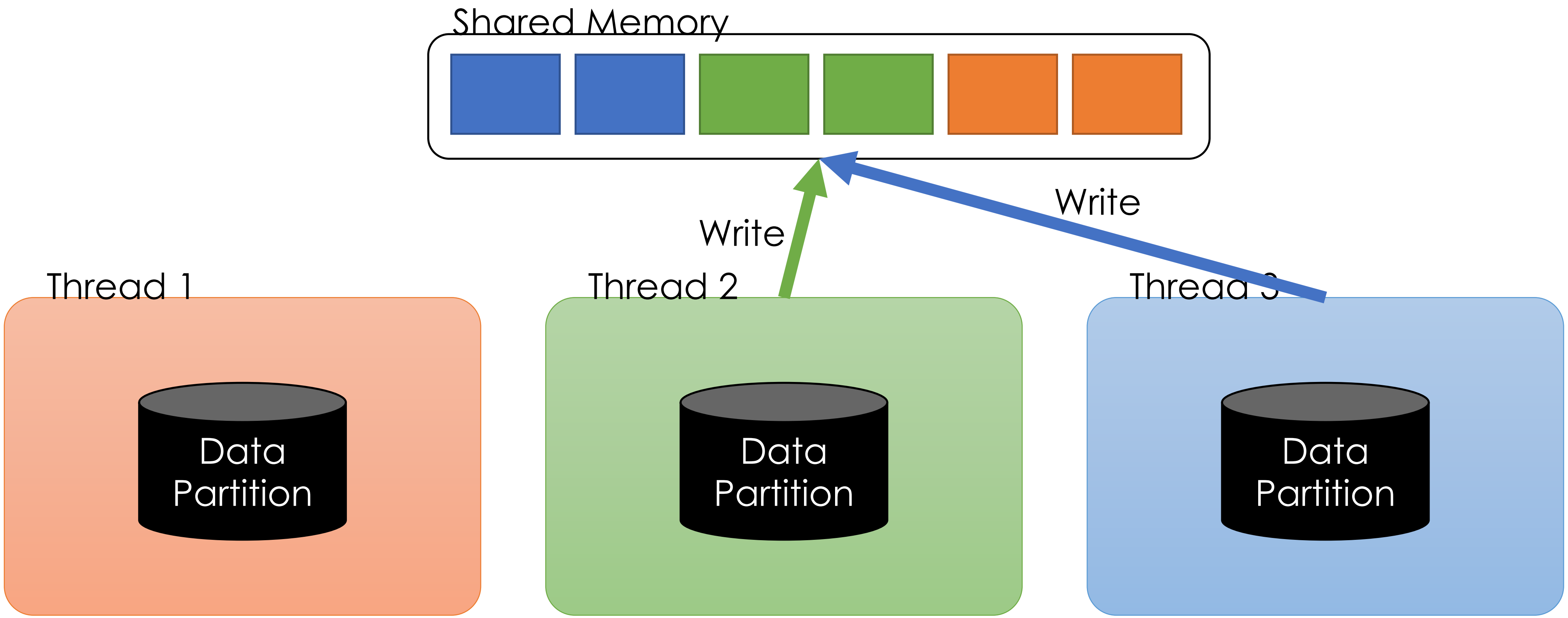
Hogwild! Algorithm



Hogwild! Algorithm



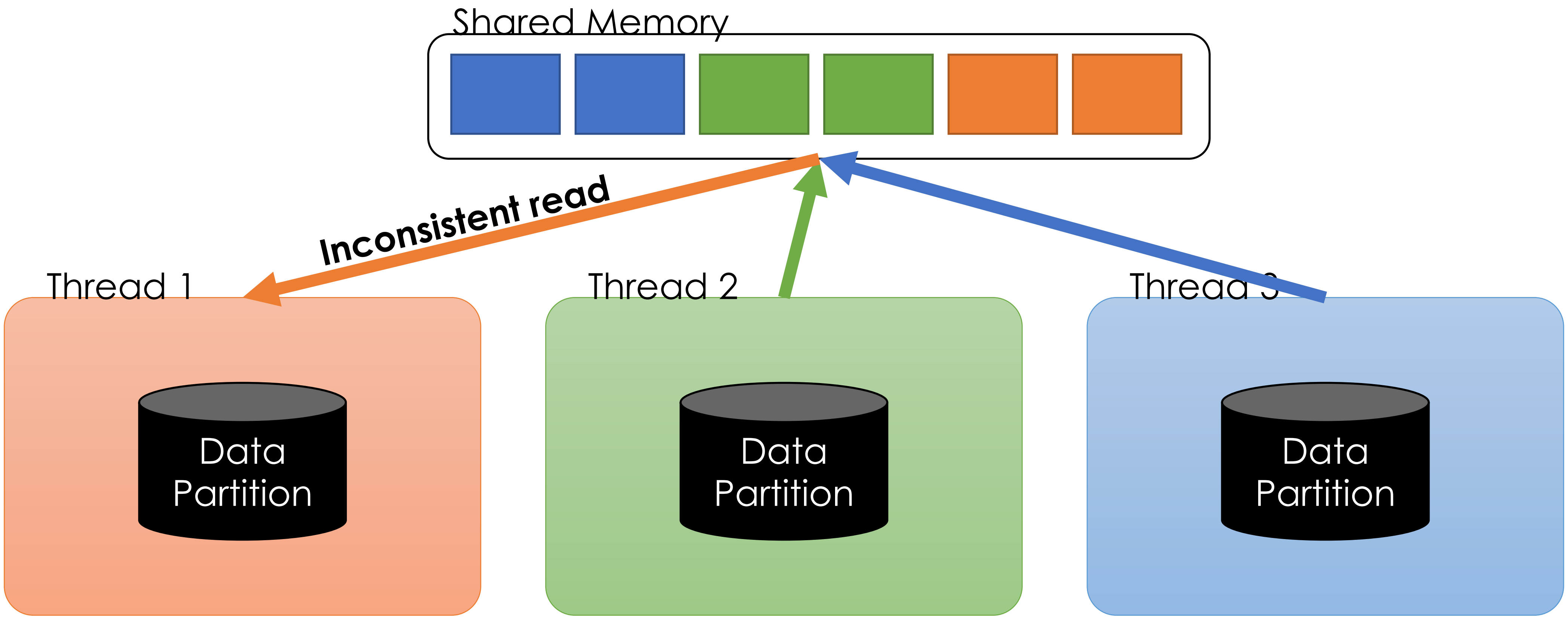
Hogwild! Algorithm



Hogwild! Algorithm

No corrupted floats:

Individual entries are consistent.



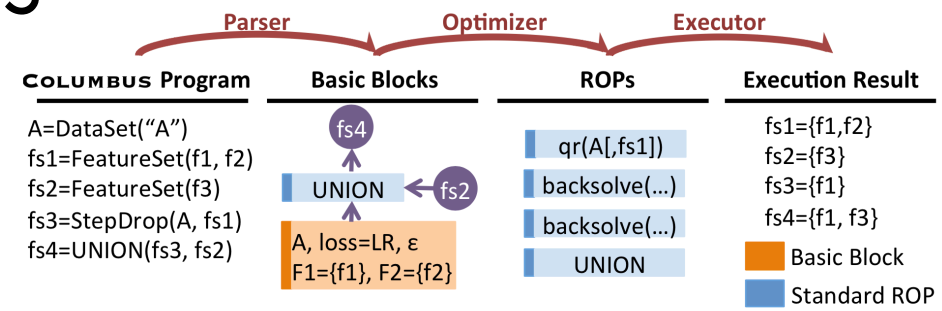
What to think about when reading?

- Implications in contemporary deep learning setting
  - TF/Pytorch training in PostgreSQL?
- Implications on distributed training?
- Multiplexed Reservoir Sampling
  - Relationship to **Replay Buffers** in RL
  - Could we leverage idea to mitigate data load to GPU?

Materialization Optimizations for Feature Selection Workloads

Ce Zhang, Arun Kumar, Christopher Ré

Materialization Optimizations for Feature Selection Workloads



- **Context:** feature selection using R scripts dominate machine learning workloads → substantial opportunity for reuse!
- **Key Idea:** Rich **tradeoff space** of what to **materialize**, how to leverage **sampling**, and **reuse computation**
- **Contribution:** this paper demonstrates the advantages of exploring the tradeoff space and describes ways in which various operations interact.

Problem Formulation

Data

A =

	f_1	f_2	f_3	f_4	f_5
r_1					
r_2					
r_3					
r_4					

b =

b

Generalized Linear Model

Solve (within ϵ of optimum)

$$x_t^* = \arg\min_{x \in \mathbb{R}^d} \sum_{i \in R_t} L\left((A \Pi_{F_t} x)_i, b_i\right)$$

Projection Matrix

For each t:

R_t : set of rows

F_t : set of cols

- Solve **multiple problems** for **subsets** of **rows** and **columns** of original data
- Block consists of:
  - Loss functions L
  - **Set of Sets** of Rows / Columns
  - Accuracies ϵ
- Explore **optimizations** targeted at solving the **related problems**
  - **Materialization, Sampling, Compute reuse**

Optimization: Lazy vs Eager Materialization

- **Lazy Materialization:** construct each feature table as it is needed from raw data
- **Eager Materialization:** precomputes the superset of columns (features) and then projects away what is not needed for each optimization task
- **Tradeoffs**
  - **Lazy →** Higher computational cost, less storage overhead
  - **Eager →** Less compute, greater storage overhead

Optimization: Sampling

- **No Sampling:** compute on full data
  - May waste computation when identifying features
- **Random Sampling:** work on random subset of data (rows)
  - Much faster but potentially less accurate conclusions
- **Coreset Sampling:** weighted sampling to improve approximation of loss estimate
  - Better captures outliers
  - Requires **multiple passes** through data and **rows >> columns**

Optimization: Compute Reuse

- **QR Factorization:** reuse computation across multiple solves of related linear systems
  - Clever (established) idea
  - Limited applicability squared loss + linear models + L_2 regularization
  - **Example Regularized Least Squares**:

Loss minimizer is the solution to:

Reuse: Solved O(d^3)  and $Q^T Q = I$

$$\left(A^T A + \lambda\right) \Pi_F x = \Pi_F A^T b \longrightarrow QR = \left(A^T A + \lambda\right)$$

$$(QR)\, \Pi_F x = \Pi_F A^T b \longrightarrow \left(R \Pi_F\right) x = Q^T \Pi_F A^T b$$

Solved O(d^2) using backward substitution: 

for any Π_F

Optimization: ADMM + Warmstart

- **ADMM Alg.:** rewrite more general convex optimization problems (e.g., LASSO, logistic regression, SVM) into sequence of least squares problems (leverage QR)
  - Clever (established) idea
  - Enables use of warm-start

$$x^{(k+1)} = \arg\min_x \frac{\rho}{2} \left|\left| A\Pi_F x - \left(z^{(k)} - u^{(k)}\right)\right|\right|_2^2$$

Repeatedly Solve Least Squares Problem (use QR technique)

Extra hyperparameter

$$z^{(k+1)} = \arg\min_z \sum_{i=1}^{N} l(z_i, b_i) + \frac{\rho}{2} \left|\left| A\Pi_F x^{(k+1)} - \left(z - u^{(k)}\right)\right|\right|_2^2$$

O(n) one-dimensional optimization problems

$$u^{(k+1)} = u^{(k)} + A\Pi_F x^{(k+1)} - z^{(k+1)}$$

What consider when reading?

- Problem formulation and discussion around user interviews
- Discussion and framing of tradeoffs
- Would these techniques be applicable beyond feature selection (e.g., hyperparameter search/model design)?

Learning Generalized Linear Models Over Normalized Data

Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel

Learning Generalized Linear Models Over Normalized Data

Sales **Fact Table**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
12	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	2	20
13	1	2	20
13	2	2	40
13	3	2	5

Locations

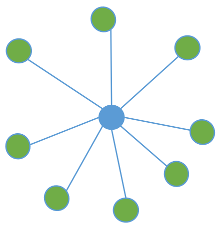
locid	city	state	country
1	Omaha	Nebraska	USA
2	Seoul		Korea
5	Richmond	Virginia	USA

Products

pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

timeid	Date	Day
1	3/30/16	Wed.
2	3/31/16	Thu.
3	4/1/16	Fri.



- **Context:** Training data is often **heavily denormalized** resulting in substantial redundancy.
  - increases **storage** and **data load time** and **computation**
- **Key Idea:** Push **learning through joins** to eliminate redundant loads and inner product calculations
- **Contribution:** this paper demonstrates the advantages of pushing learning through joins
  - Done using UDA abstractions

Context: Unnormalized Data

pname	category	price	qty	date	day	city	state	country
Corn	Food	25	25	3/30/16	Wed.	Omaha	NE	USA
Corn	Food	25	8	3/31/16	Thu.	Omaha	NE	USA
Corn	Food	25	15	4/1/16	Fri.	Omaha	NE	USA
Galaxy	Phones	18	30	1/30/16	Wed.	Omaha	NE	USA
Galaxy	Phones	18	20	3/31/16	Thu.	Omaha	NE	USA
Galaxy	Phones	18	50	4/1/16	Fri.	Omaha	NE	USA
Galaxy	Phones	18	8	[illegible]/30/16	Wed.	Omaha	NE	USA
Peanuts	Food	2	45	3/31/16	Thu.	Seoul		Korea

- **Big** table: many *columns* and *rows*
  - Substantial redundancy → expensive to store and access
  - Make mistakes while updating
- Could we organize the data more efficiently?

Multidimensional Data Model

Sales **Fact Table**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
12	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26

Locations

locid	city	state	country
1	Omaha	Nebraska	USA
2	Seoul		Korea
5	Richmond	Virginia	USA

Dimension Tables

Products

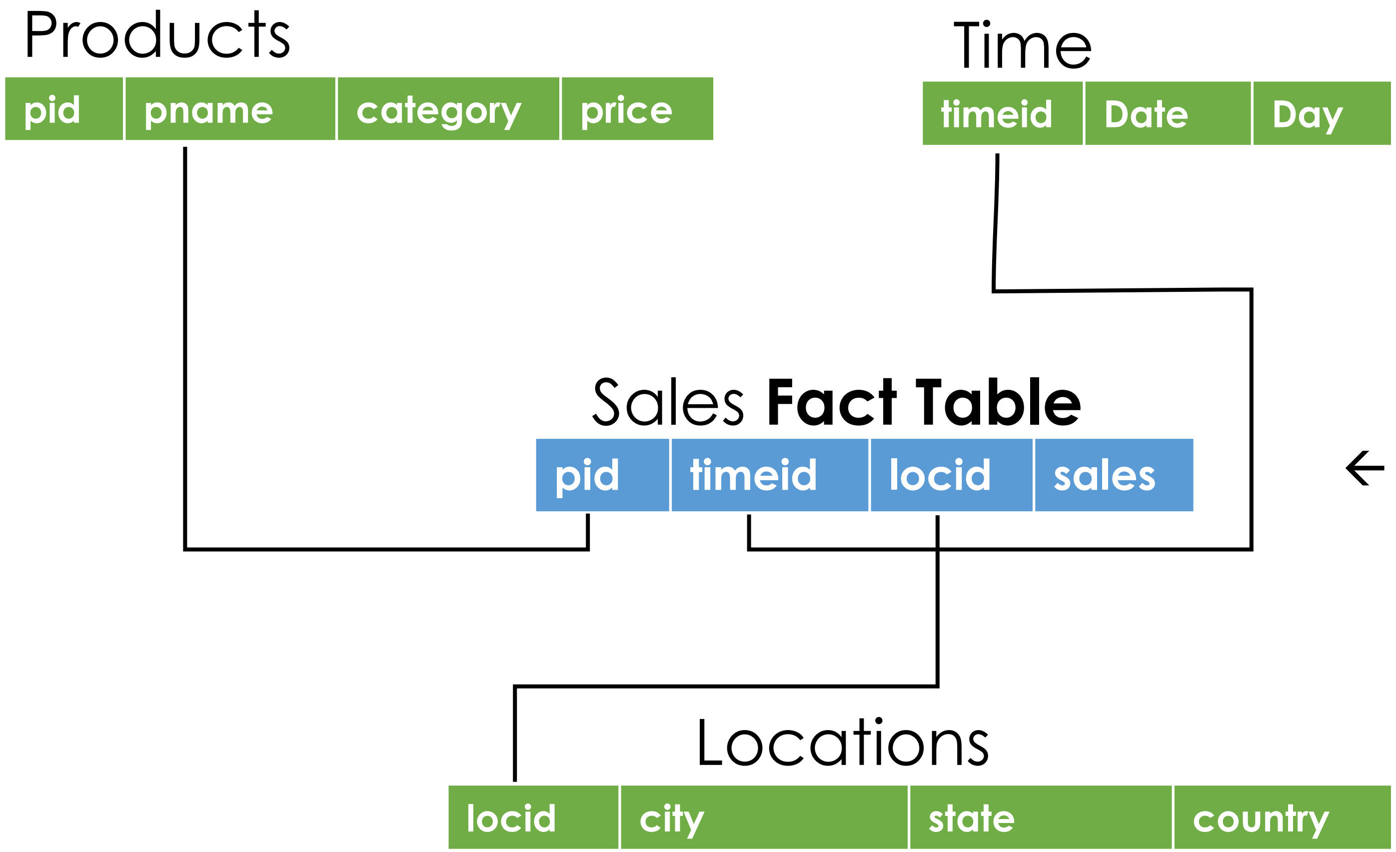
pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

timeid	Date	Day
1	3/30/16	Wed.
2	3/31/16	Thu.
3	4/1/16	Fri.

- Fact Table
  - Minimizes redundant info
  - Reduces data errors
- Dimensions
  - Easy to manage and summarize
  - Rename: Galaxy1 → Phablet
- Normalized Representation
- How do we do analysis?
  - **Joins!**

The Star Schema



← This looks like a star …

Multidimensional Data Model

Sales **Fact Table**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
12	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26

Locations

locid	city	state	country
1	Omaha	Nebraska	USA
2	Seoul		Korea
5	Richmond	Virginia	USA

Products

pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

timeid	Date	Day
1	3/30/16	Wed.
2	3/31/16	Thu.
3	4/1/16	Fri.

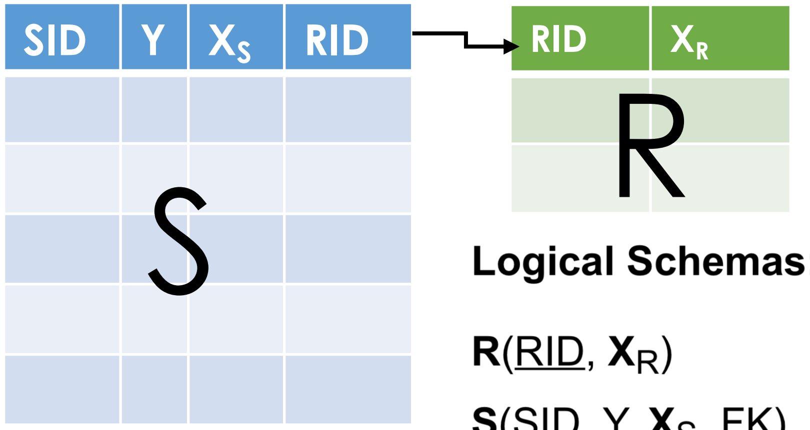
Dimension Tables

- Dimension tables contain **feature information**

Idea: Compute/store feature transformations for dimension tables?

Factorize Algorithm

- Compute **partial inner products** with features in **R → HR**
- Join **HR** with **S**
  - Finish computing inner products
  - Aggregate sum of loss **F**
  - Aggregate gradient of loss for **S weights**
- Group join result on **RID** (foreign key)
  - Aggregate gradients on **S**
- Join aggregated gradients with **R**
  - Aggregate gradient of loss for **R weights**



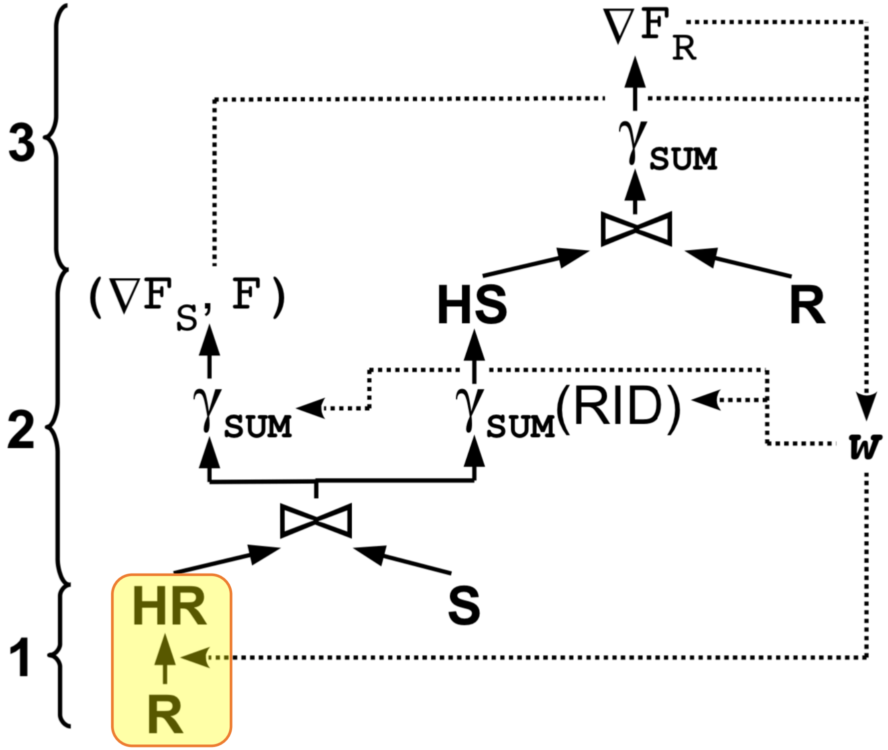
Logical Schemas:

R(RID, $\mathbf{X}_R$)

S(SID, Y, $\mathbf{X}_S$, FK)

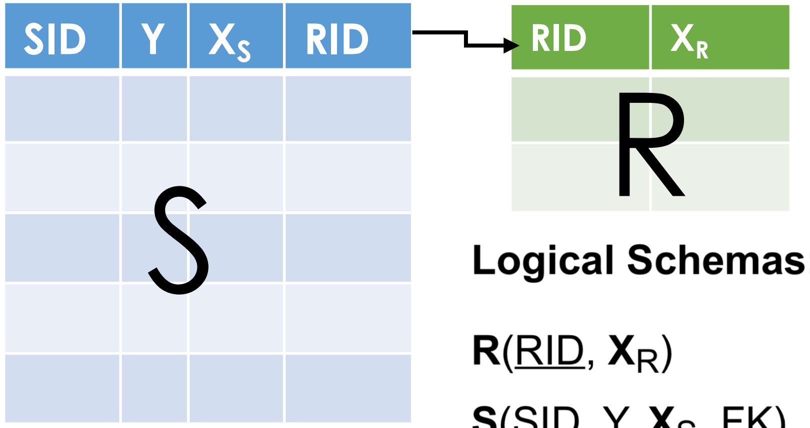
HR(RID, PartialIP)

HS(RID, SumScaledIP)



Factorize Algorithm

- Compute **partial inner products** with features in **R → HR**
- Join **HR** with **S**
  - Finish computing inner products
  - Aggregate sum of loss **F**
  - Aggregate gradient of loss for **S weights**
- Group join result on **RID** (foreign key)
  - Aggregate gradients on **S**
- Join aggregated gradients with **R**
  - Aggregate gradient of loss for **R weights**



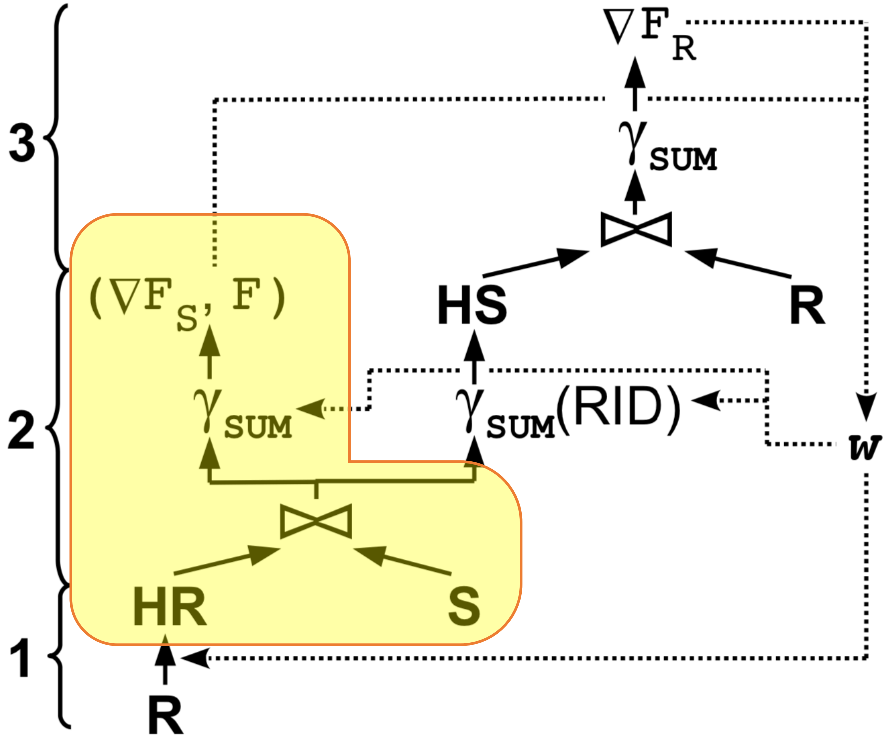
Logical Schemas:

R(<u>RID</u>, $\mathbf{X}_R$)

S(<u>SID</u>, Y, $\mathbf{X}_S$, FK)

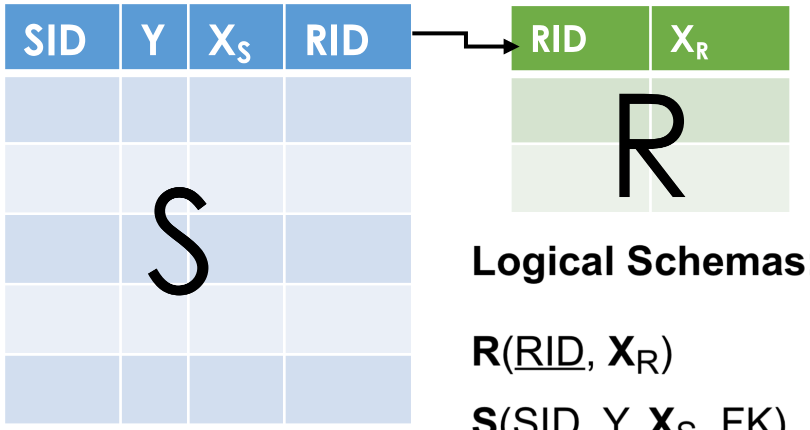
HR(<u>RID</u>, PartialIP)

HS(<u>RID</u>, SumScaledIP)



Factorize Algorithm

- Compute **partial inner products** with features in **R** → **HR**
- Join **HR** with **S**
  - Finish computing inner products
  - Aggregate sum of loss **F**
  - Aggregate gradient of loss for **S weights**
- Group join result on **RID** (foreign key)
  - Aggregate gradients on **S**
- Join aggregated gradients with **R**
  - Aggregate gradient of loss for **R weights**



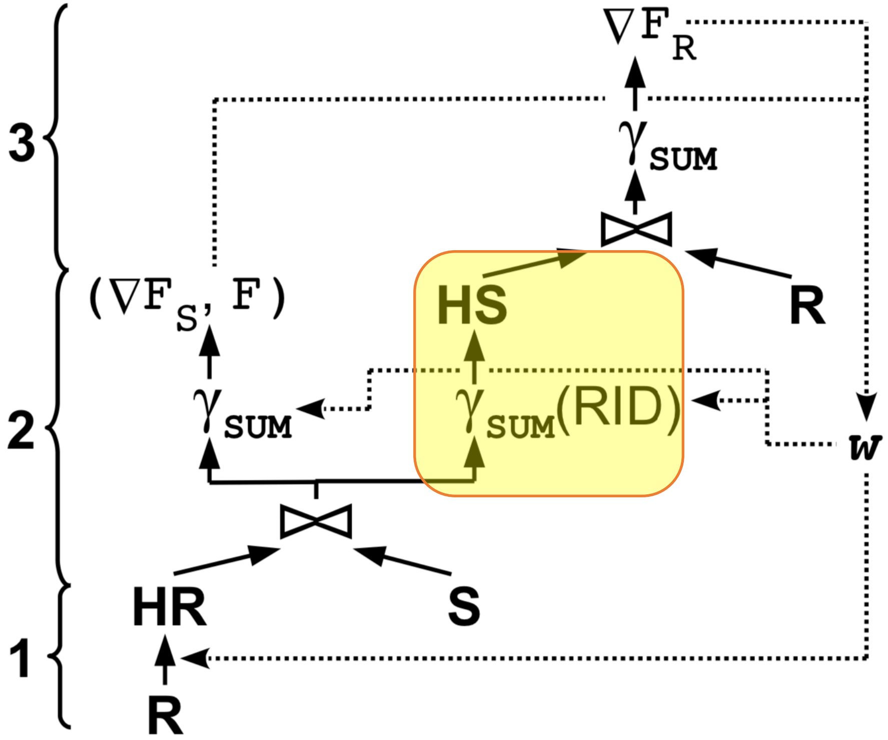
Logical Schemas:

R(<u>RID</u>, $\mathbf{X}_R$)

S(<u>SID</u>, Y, $\mathbf{X}_S$, FK)

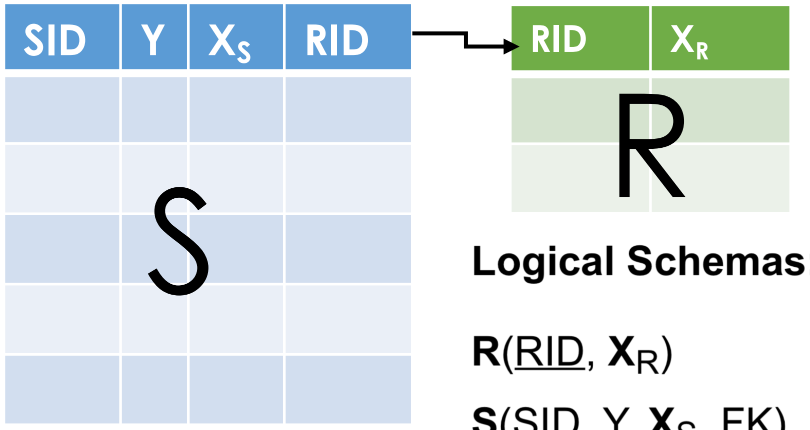
HR(<u>RID</u>, PartialIP)

HS(<u>RID</u>, SumScaledIP)



Factorize Algorithm

- Compute **partial inner products** with features in **R → HR**
- Join **HR** with **S**
  - Finish computing inner products
  - Aggregate sum of loss **F**
  - Aggregate gradient of loss for **S weights**
- Group join result on **RID** (foreign key)
  - Aggregate gradients on **S**
- Join aggregated gradients with **R**
  - Aggregate gradient of loss for **R weights**



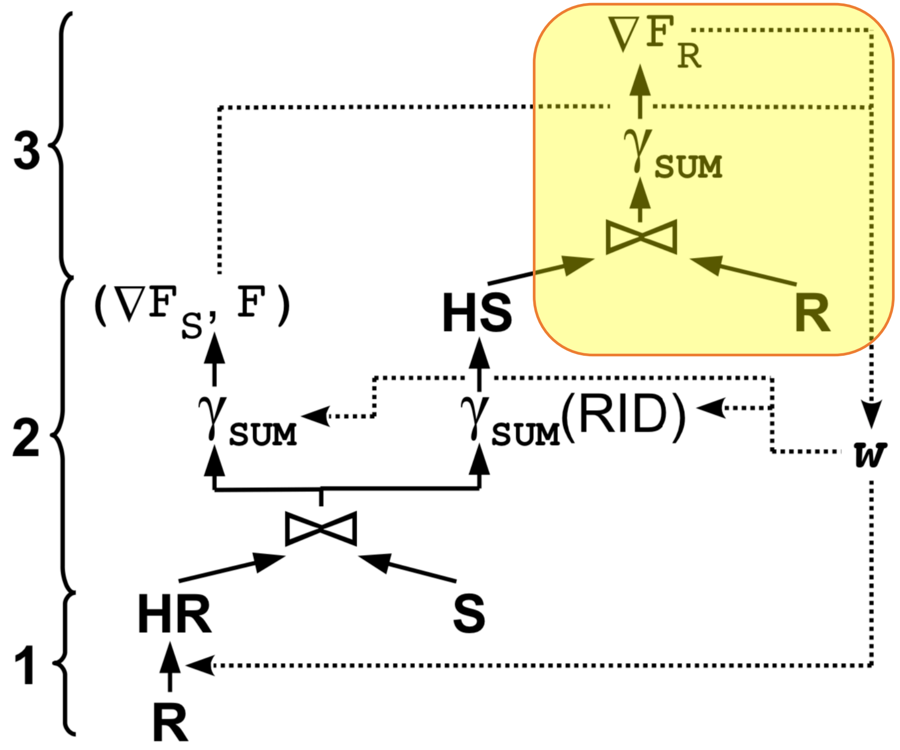
Logical Schemas:

R(<u>RID</u>, **X**$_R$)

S(<u>SID</u>, Y, **X**$_S$, FK)

HR(<u>RID</u>, PartialIP)

HS(<u>RID</u>, SumScaledIP)



Thoughts For Reading

- Emphasis on cost model
  - Can you work through the cost calculations?
- What would happen if features depended on cross terms between tables?
- Would these techniques be applicable beyond feature selection (e.g., hyperparameter search/model design)?
  - Are there scenarios where this optimization would work?

Done!