Large-Scale Data Analysis: Bridging the Gap

Alekh Jindal, Yagiz Kargin, Sarath Kumar, Vinay Setty

SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

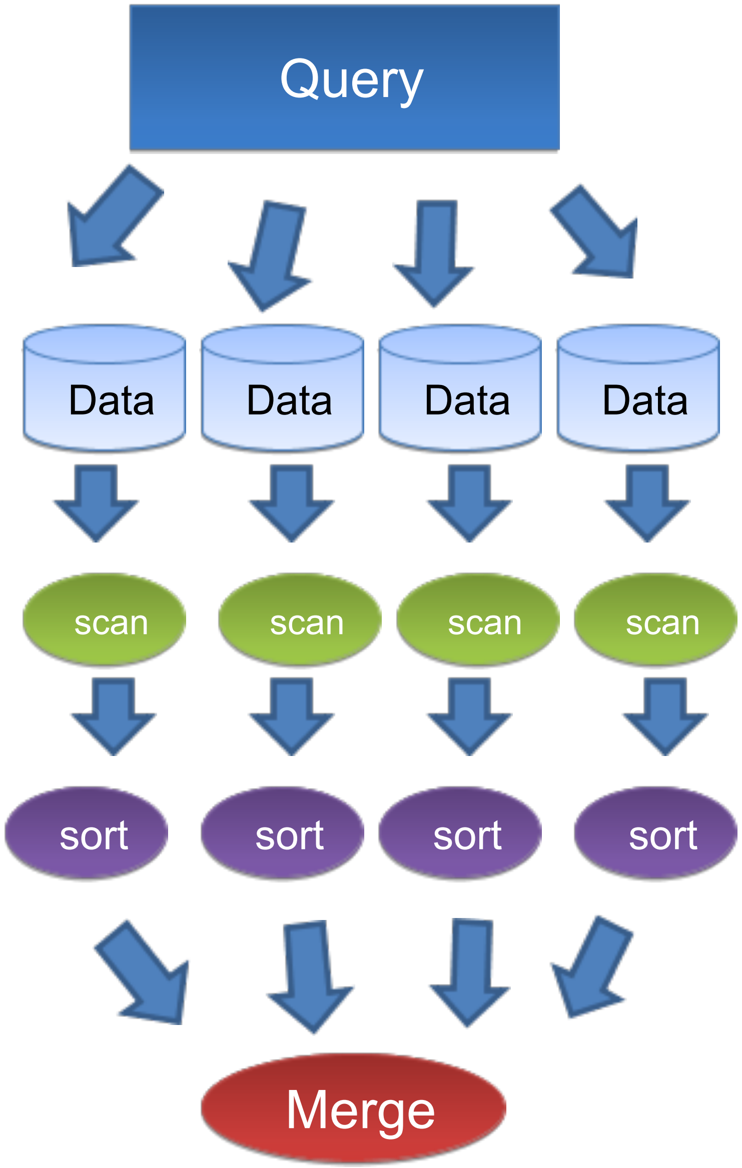
Outline

- Motivation: Parallel DBMS vs Map/Reduce
- Schema & Benchmarks Overview
- Original(Pavlo) Map/Reduce Plans
- Improved(SAVY) Design & Implementation
- Improving Hadoop
    - Indexing
    - Co-Partitioning
- Experiments
- Conclusion

Motivation

- Ever growing data
  - About 20TB per Google crawl!
- Computing Solutions
  - High-end server: 1625.60€/core, 97.66€/GB
  - Share-nothing nodes: 299.50€/core, 166.33 €/GB
- Two Paradigms
  - Parallel DBMS
  - Map/Reduce

Parallel DBMS



[DeWitt, D. and Gray, J. 1992.]

Parallel DBMS: Advantages

- Can be column based
  - Example: Vertica
- Local joins possible
  - Partition based on join key
- Can work on compressed data
  - reduced data transfer
- Flexible query plans
- Supports Declarative languages like SQL

Parallel DBMS - Shortcomings

- Not free of cost
- Not open source
- Cannot scale to thousands of nodes: why?
    - Less fault tolerant
    - Assumes homogeneous nodes
- Not so easy to achieve high performance
    - Needs highly skilled DBA
    - Needs high maintenance

Map/Reduce(Hadoop): Advantages

- Free of cost
- Open source
- Fault tolerant
- Scales well to thousands of nodes
- Less maintenance
- Flexible query framework

Map/Reduce(Hadoop): Shortcomings

- Lack of inbuilt Indexing Current Focus
- Cannot guarantee local joins Current Focus
- Performance degradation for SQL like queries
  - Multiple MR phases Current Focus
  - Each MR phase adds extra cost
- No Flexible query plans
- Data transfer not optimized

Benchmarks and Schema

Schema

CREATE TABLE **Documents** (
url VARCHAR
(100) PRIMARY KEY,
contents TEXT
);

CREATE TABLE **Rankings** (
pageURL VARCHAR
(100) PRIMARY KEY,
pageRank INT,
avgDuration INT
);

Schema

```
CREATE TABLE UserVisits (
    sourceIP VARCHAR(16),
    destURL VARCHAR(100),
    visitDate DATE,
    adRevenue FLOAT,
    userAgent VARCHAR(64),
    countryCode VARCHAR(3),
    languageCode VARCHAR(6),
    searchWord VARCHAR(32),
    duration INT
    );
```

Benchmarks 1&2

- **Selection task (Benchmark 1)**
  - SELECT pageURL, pageRank FROM Rankings WHERE pageRank > X;
- **Aggregation task (Benchmark 2)**
  - SELECT sourceIP, SUM(adRevenue) FROM UserVisits GROUP BY sourceIP;
  - SELECT SUBSTR(sourceIP, 1, 7), SUM(adRevenue) FROM UserVisits GROUP BY SUBSTR(sourceIP, 1, 7);

Benchmark 3: Join Task

Projection & Aggregatio

- SELECT INTO Temp **sourceIP, AVG (pageRank) as avgPageRank, SUM (adRevenue)** as totalRevenue

FROM **Rankings** AS R, **UserVisits** AS UV

Join

WHERE **R.pageURL = UV.destURL** AND UV. visitDate BETWEEN Date('2000-01-15') AND ('2000-01-22') GROUP BY UV.sourceIP;

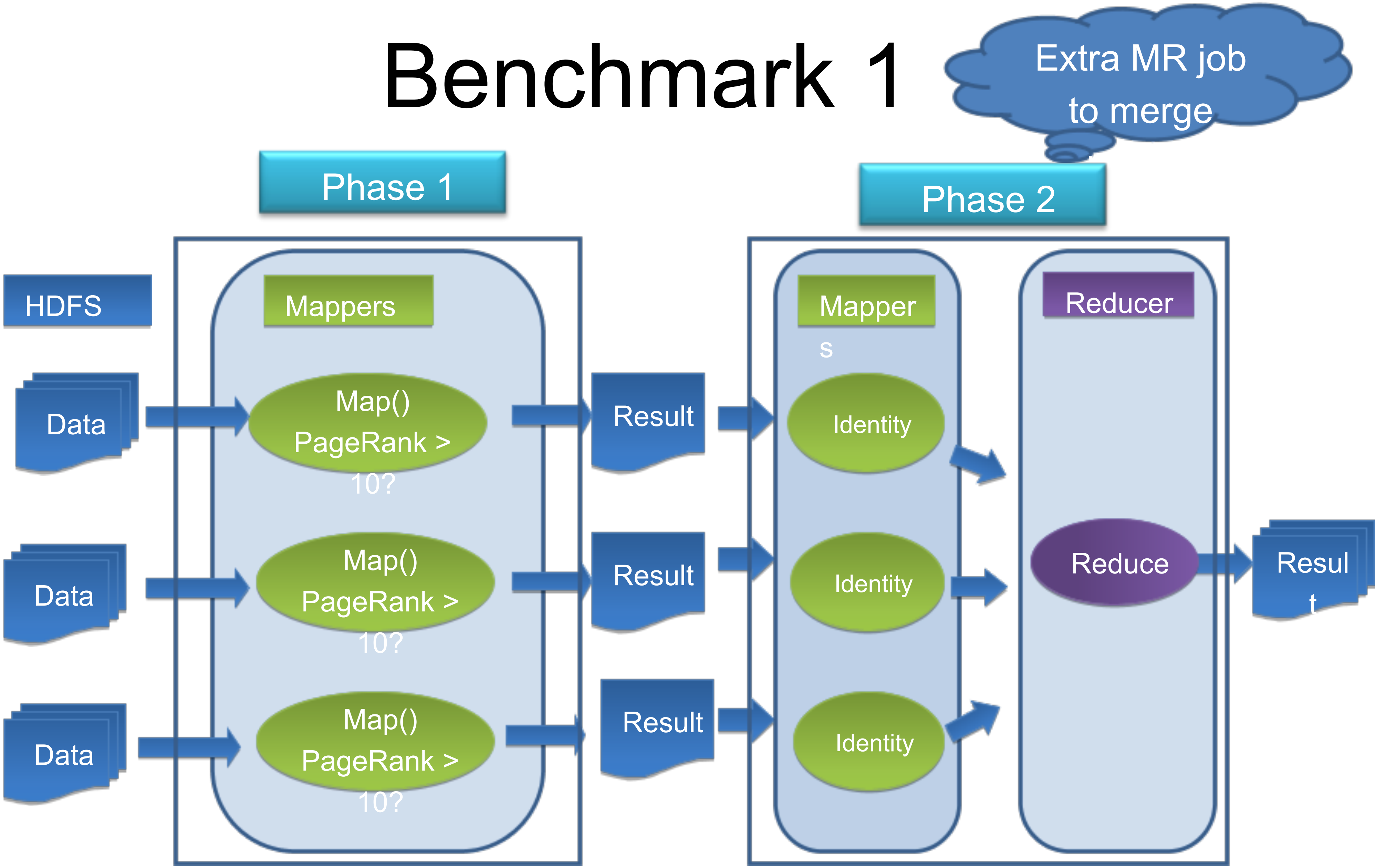
selectio
n

- SELECT sourceIP, totalRevenue, avgPageRank

FROM Temp **ORDER BY totalRevenue DESC LIMIT 1**;

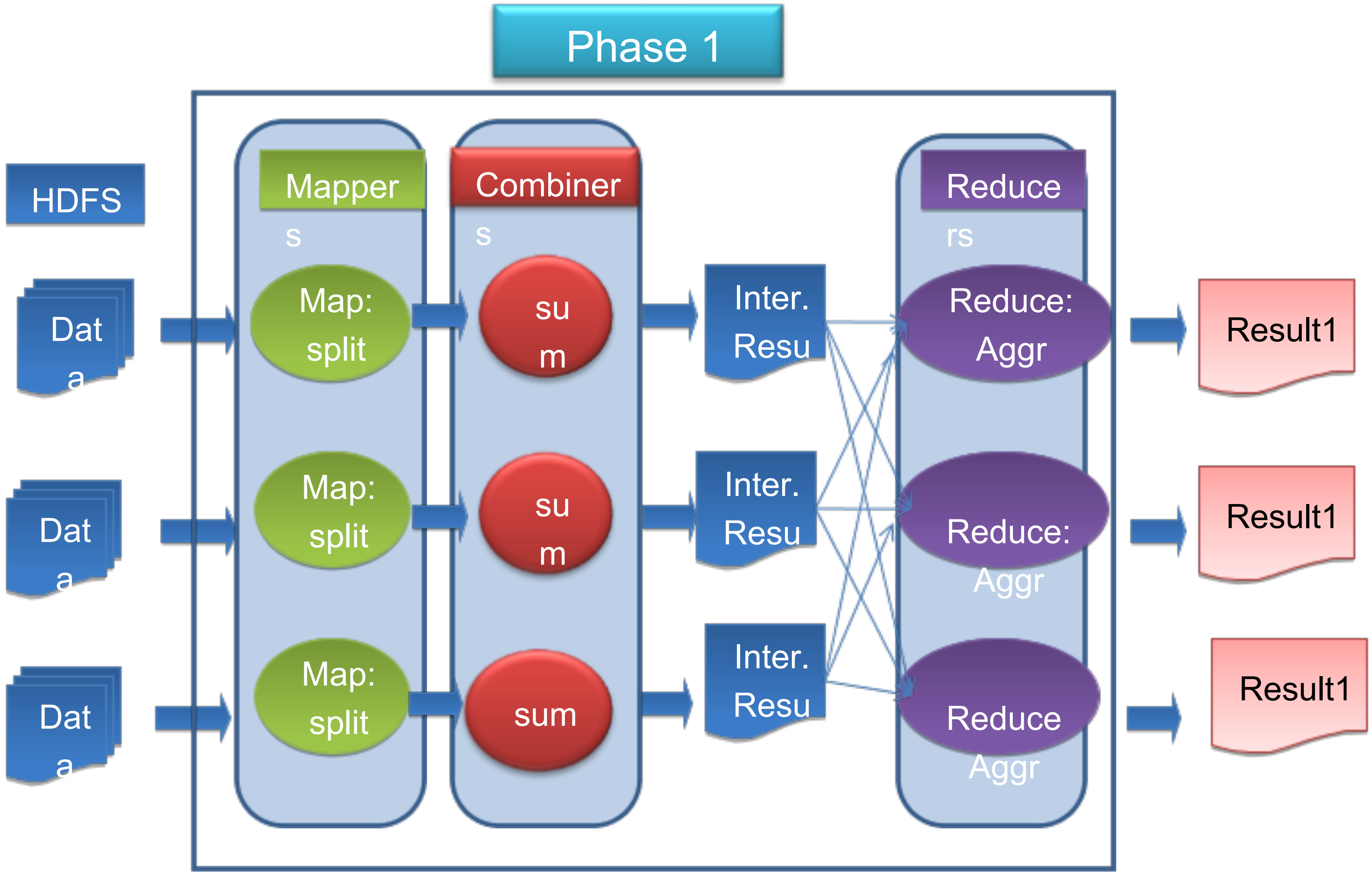
Original (Pavlo) MR Plans

Benchmark 1

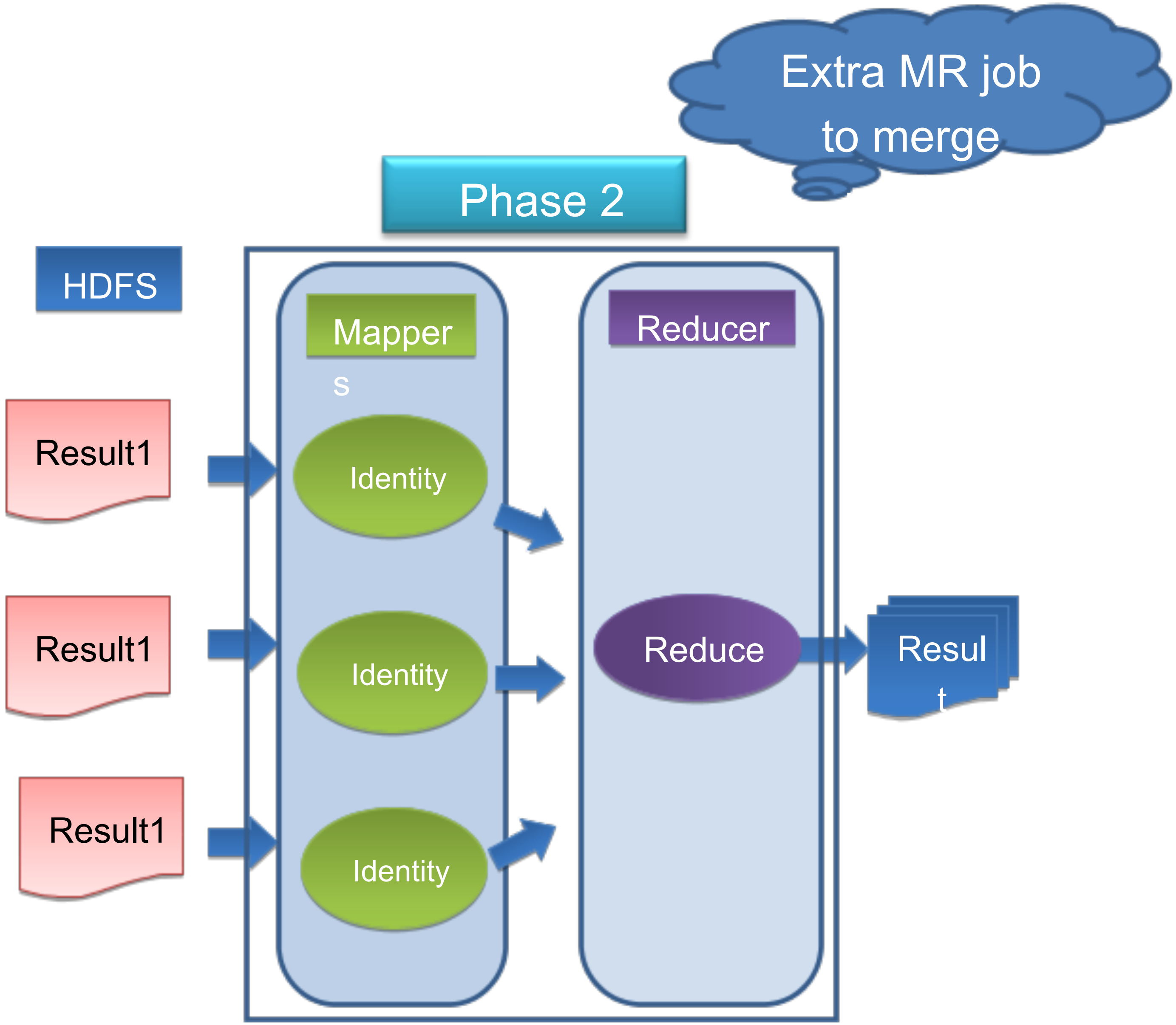


SELECT pageURL, pageRank FROM Rankings
WHERE pageRank > 10;

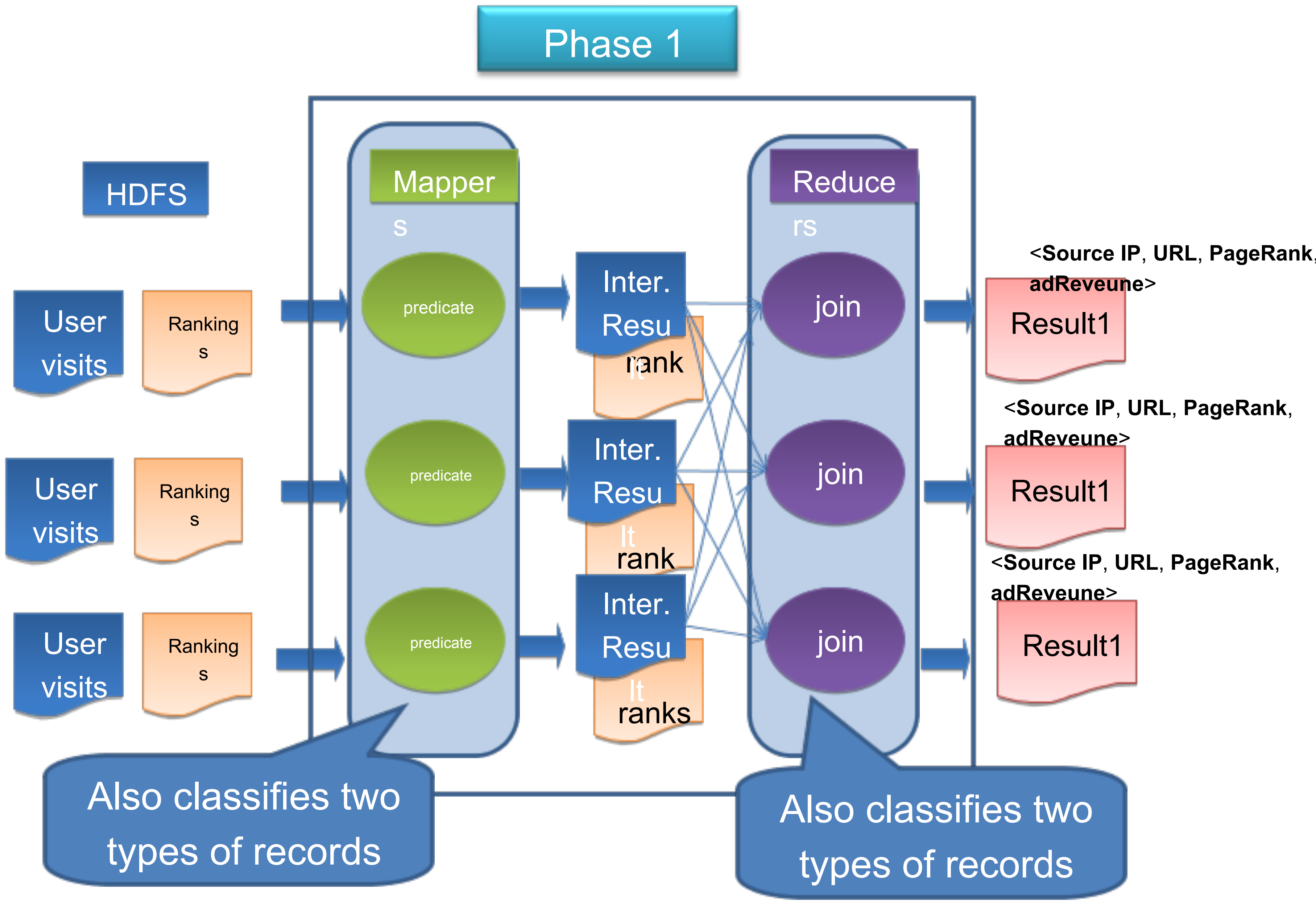
Benchmark 2: Phase 1



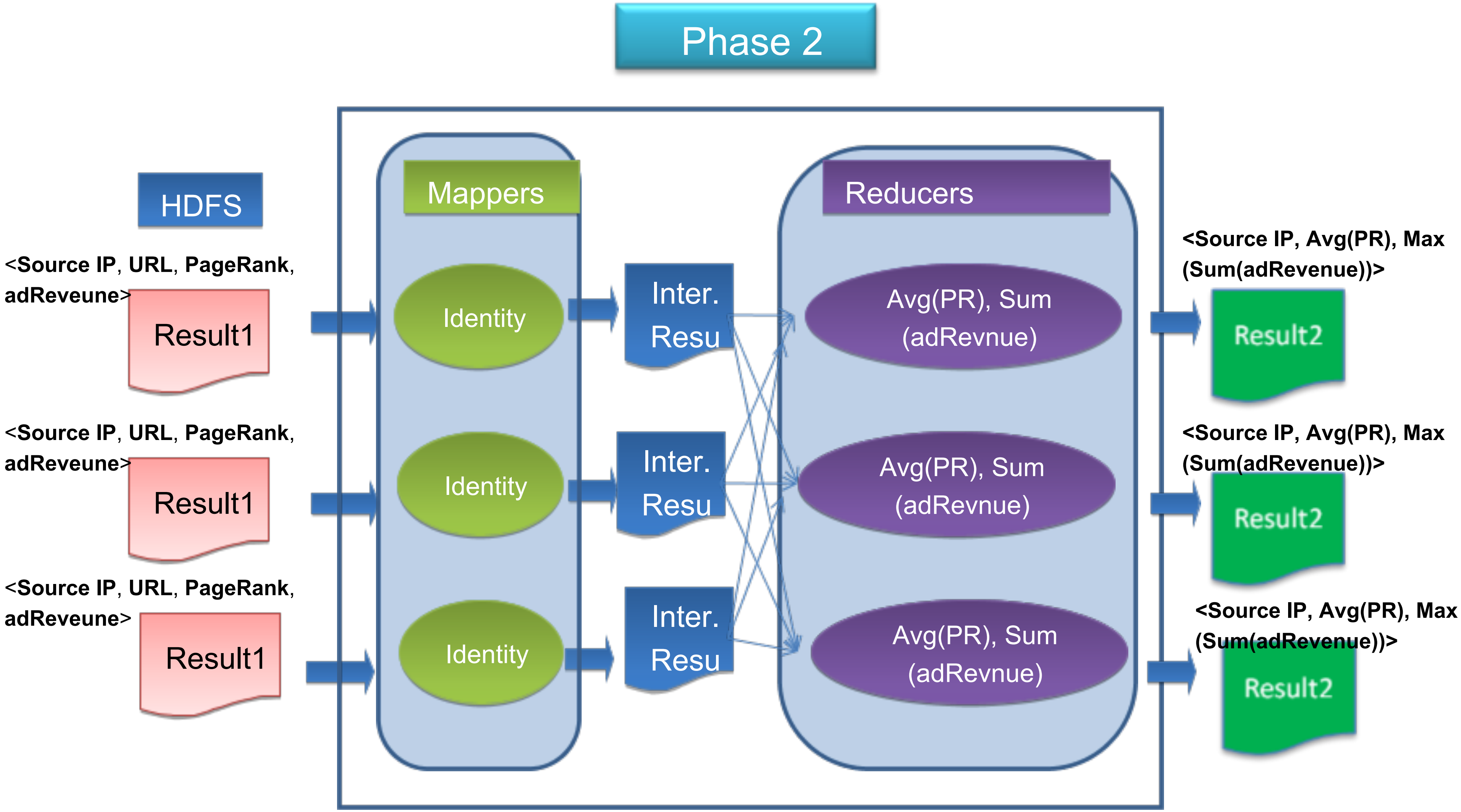
Benchmark 2: Phase 2



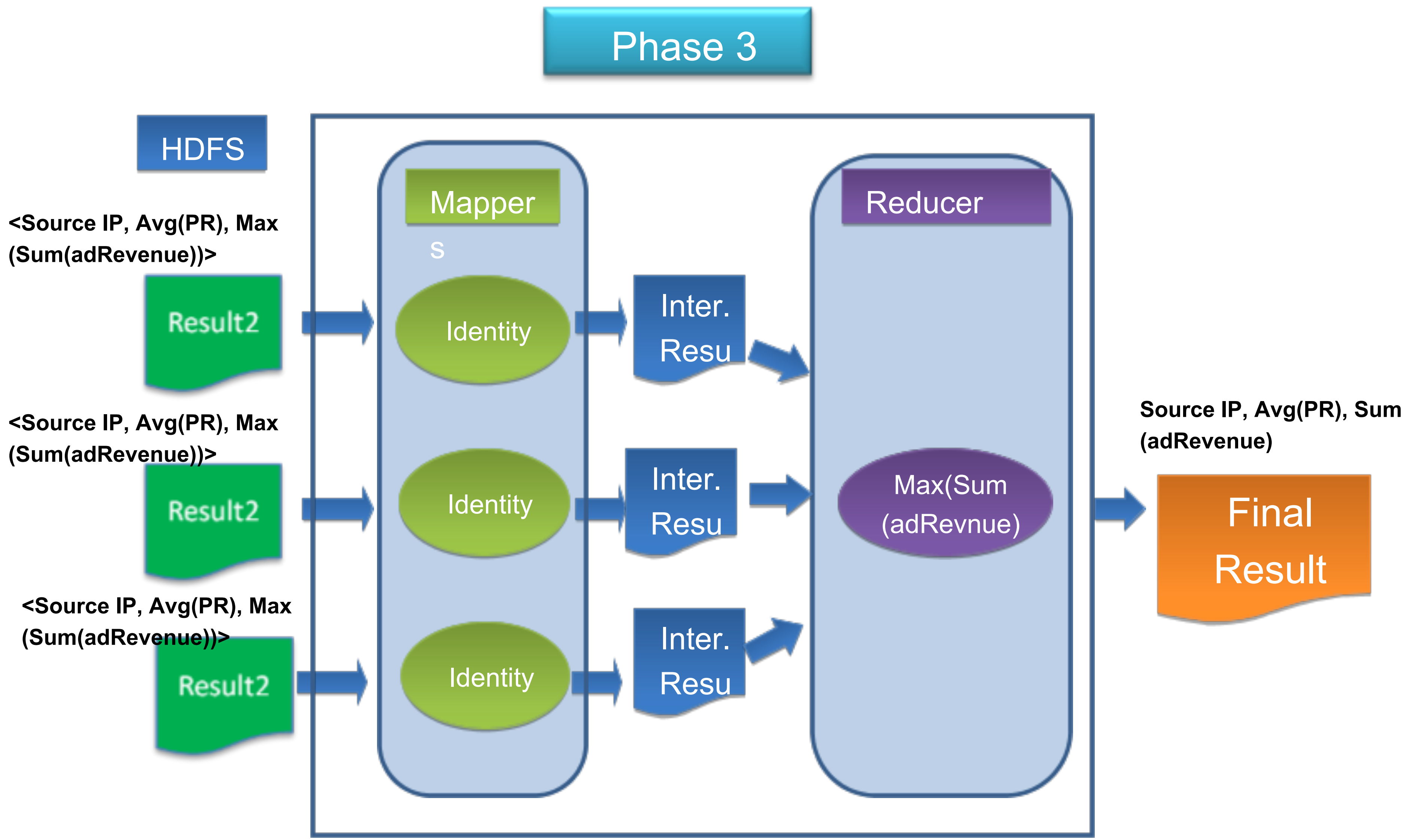
Benchmark 3 – Phase 1



Benchmark 3 – Phase 2



Benchmark 3 – Phase 3

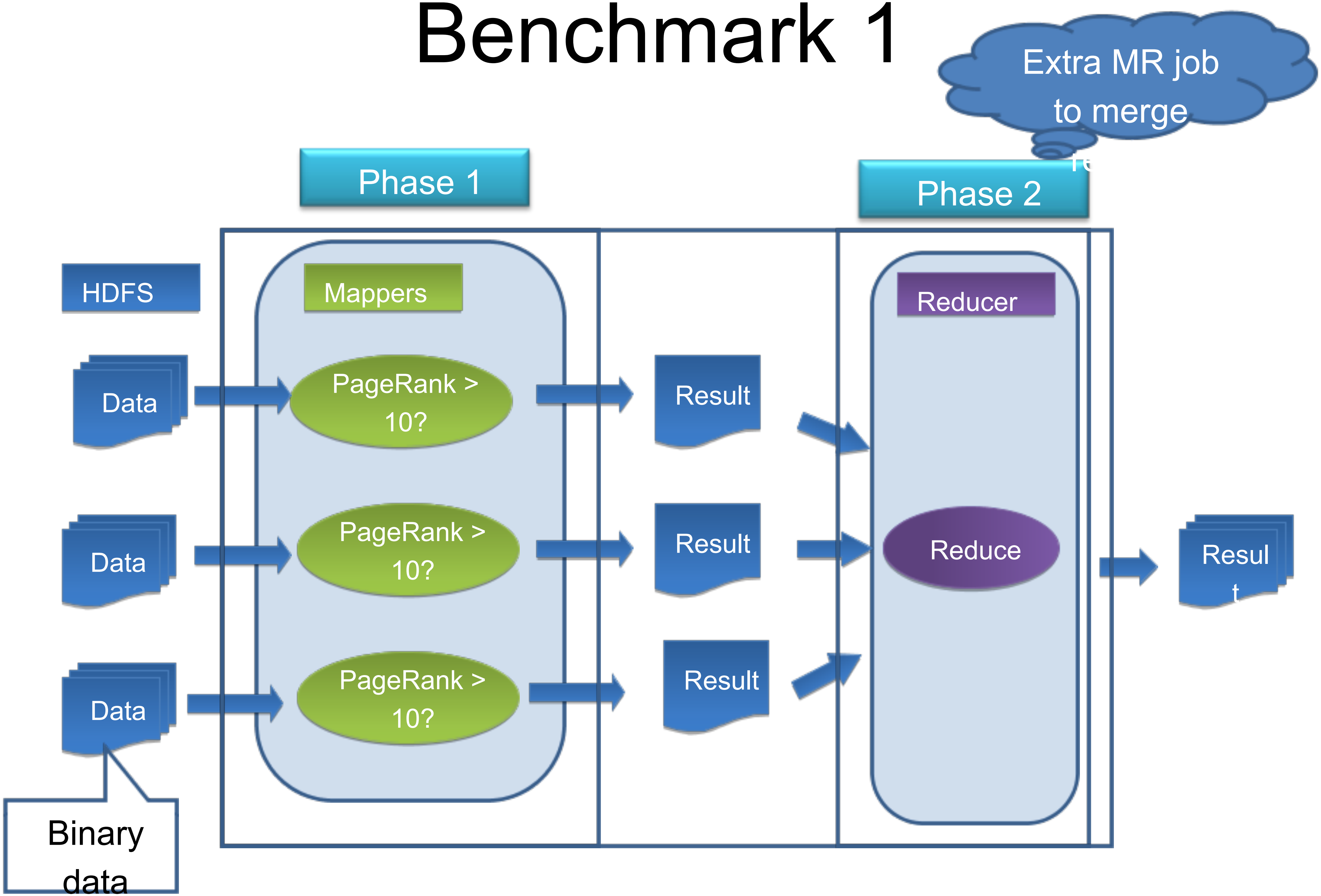


Improved (Savy) MR Plans

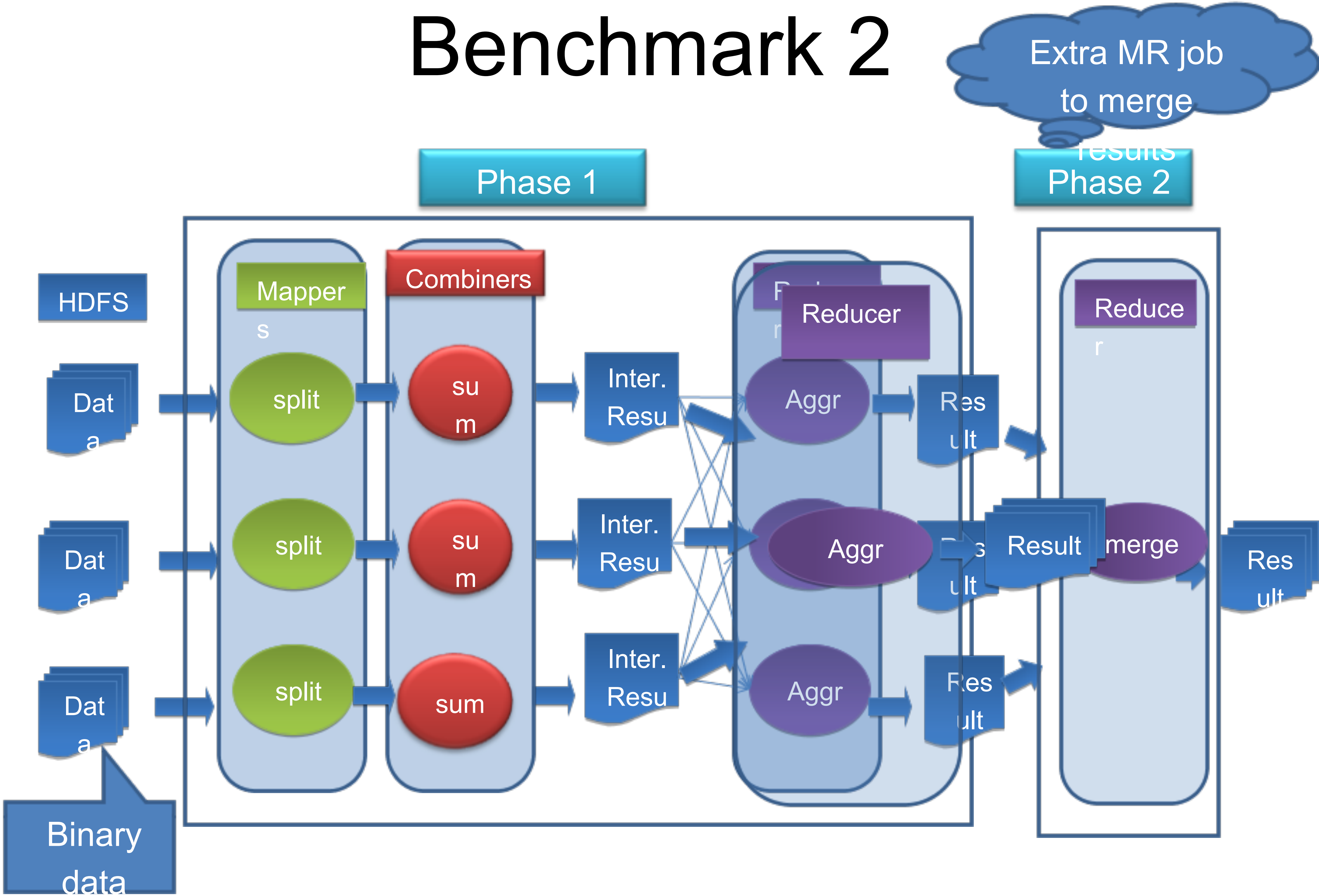
Binary Data

- Eliminates delimiters
- Avoids splitting
- Makes tuples of fixed length
- Helps in indexing

Benchmark 1

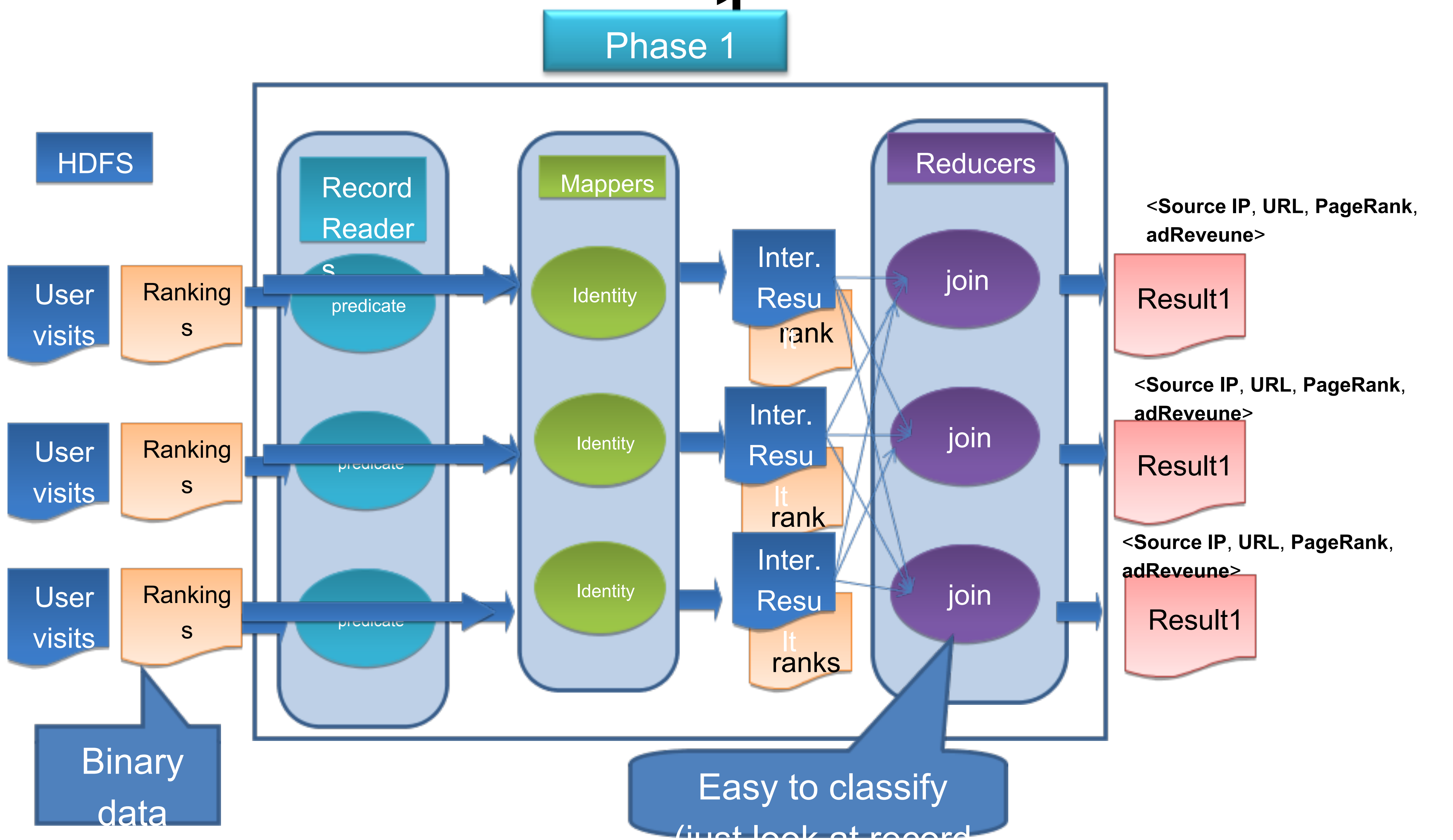

Extra MR job to merge
Phase 1
Phase 2
HDFS
Mappers
Reducer
Data
Data
Data
PageRank > 10?
PageRank > 10?
PageRank > 10?
Result
Result
Result
Reduce
Resul t
Binary data

Benchmark 2

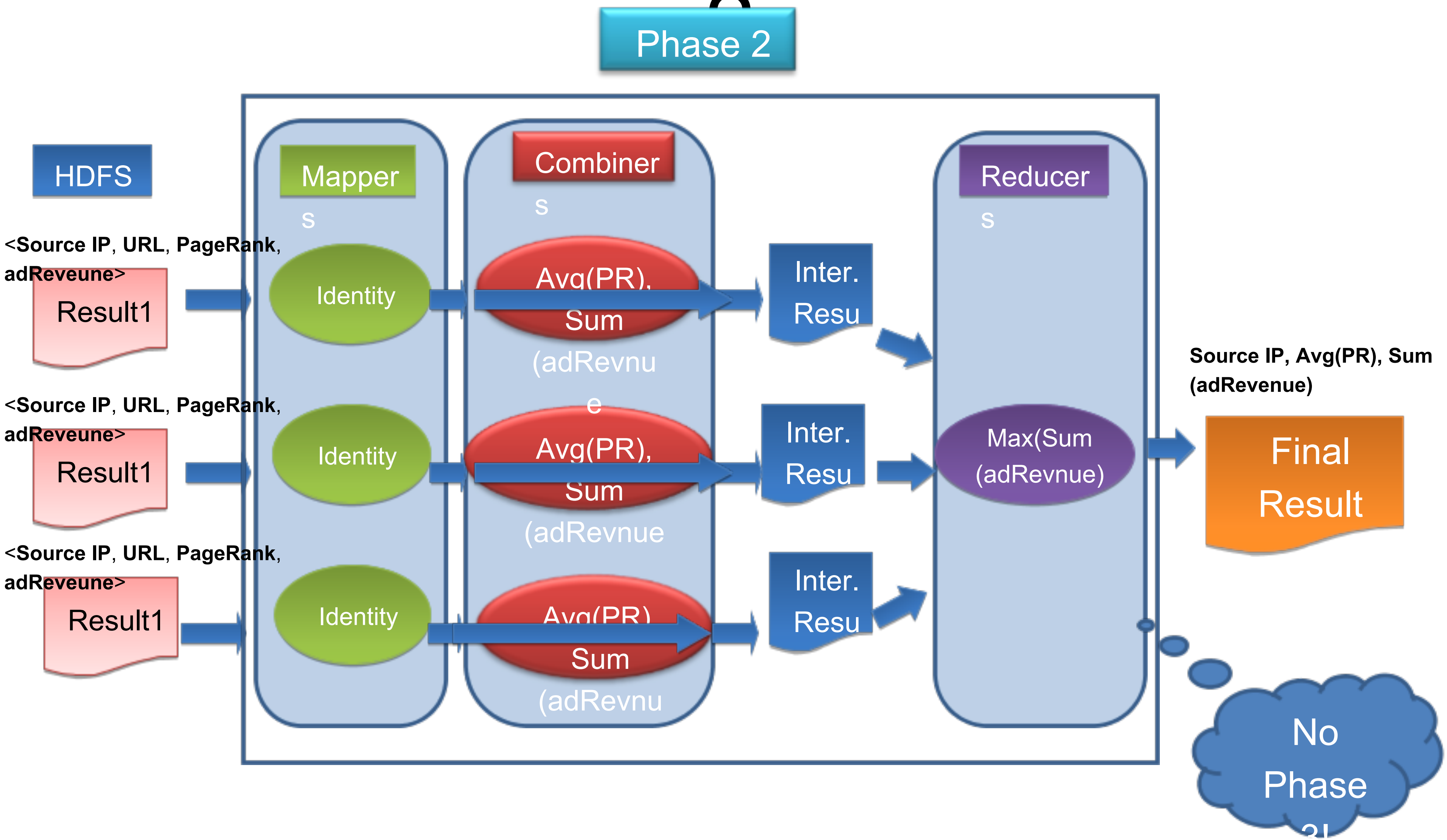


Extra MR job
to merge
results
Phase 1
Phase 2
HDFS
Mappers
Combiners
Reducer
Reducer
Data
Data
Data
Binary
data
split
split
split
sum
sum
sum
Inter.
Resu
Inter.
Resu
Inter.
Resu
Aggr
Aggr
Aggr
Result
Result
Result
Result
merge
Result

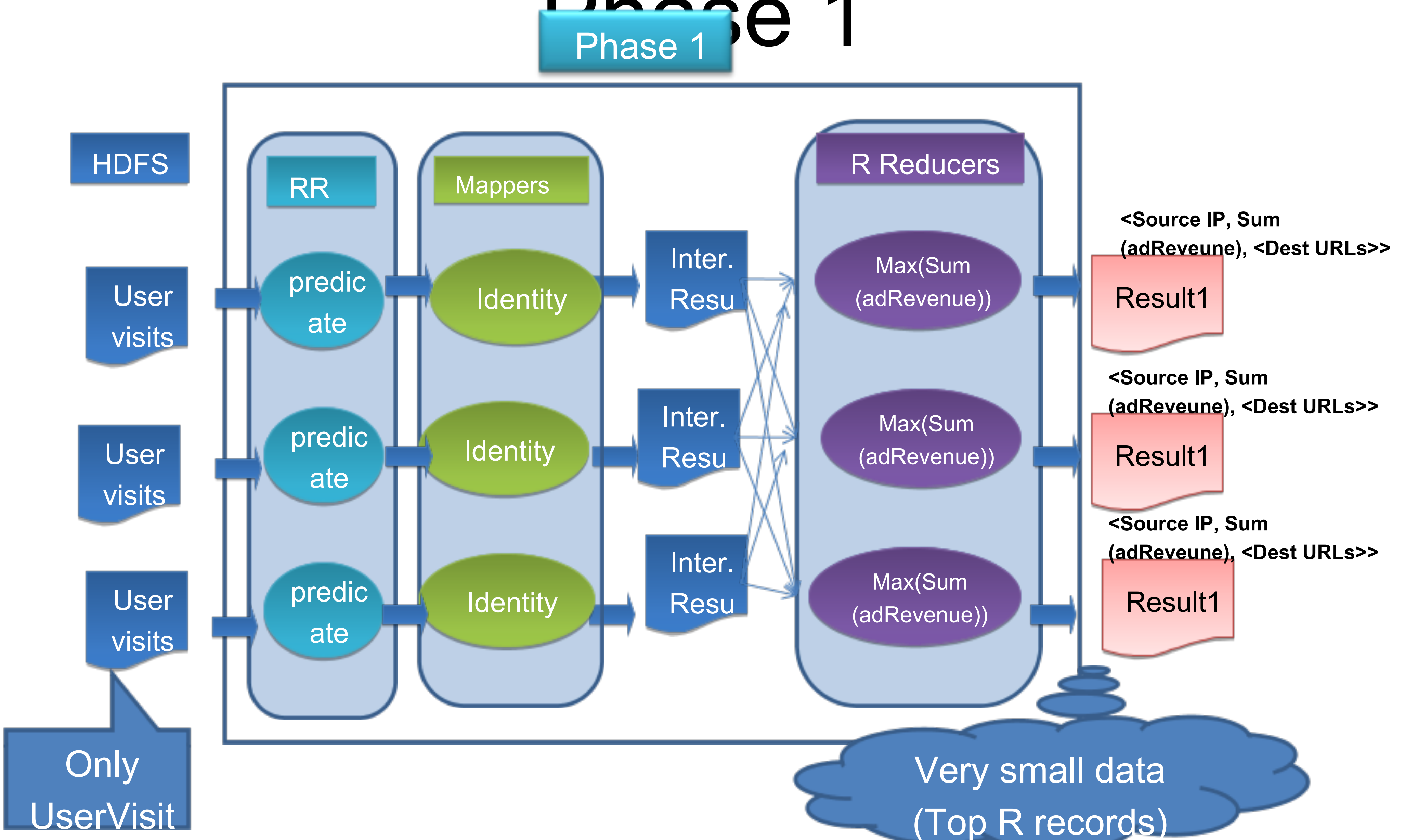
Benchmark 3(Design I) – Phase 1



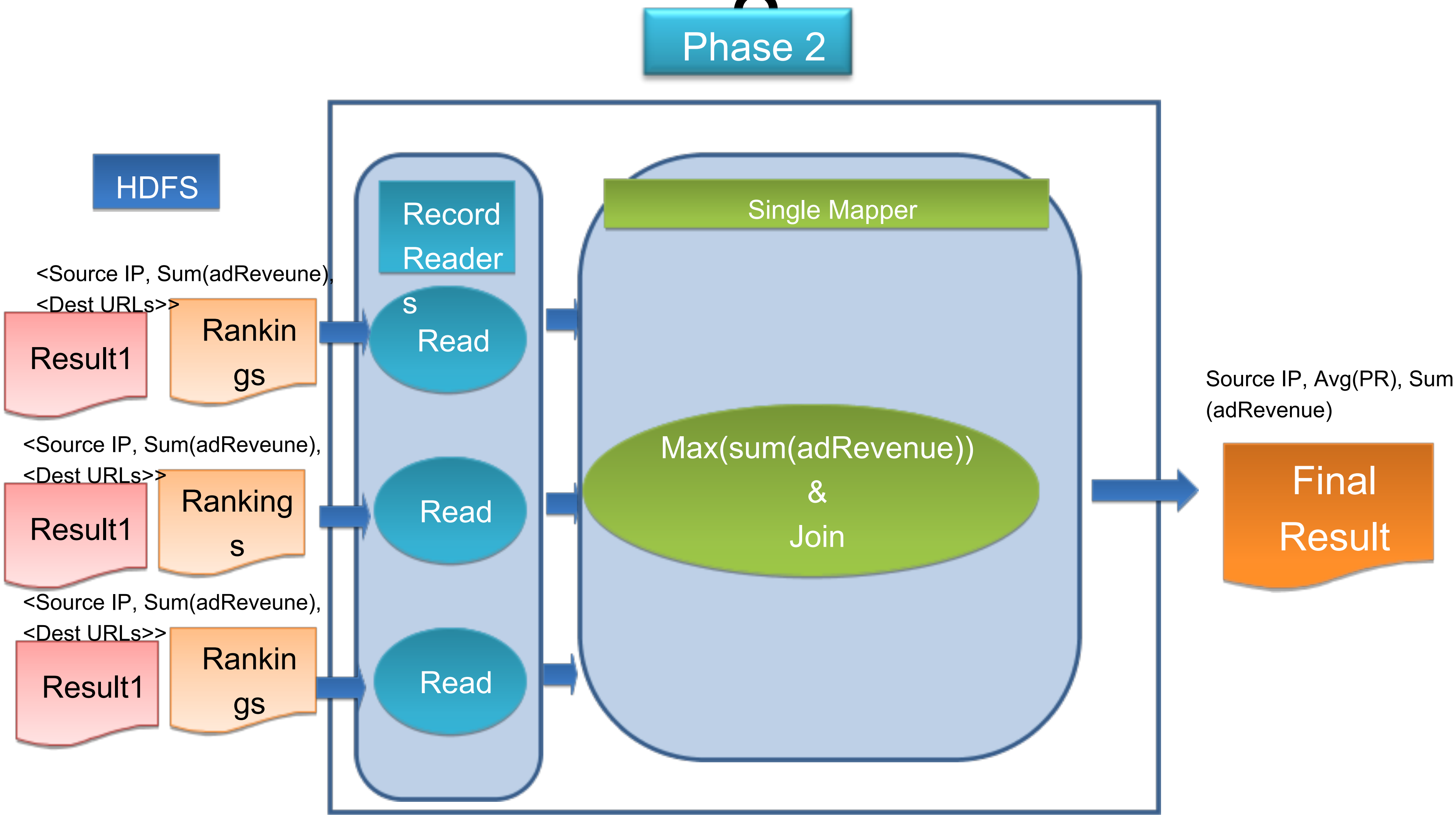
Benchmark 3(Design I) – Phase 2



Benchmark 3(Design II) – Phase 1



Benchmark 3(Design I) – Phase 2



Improving Hadoop

Improving Hadoop

- Improve Selection (Indexing)
- Improve Join (Co-partitioning)

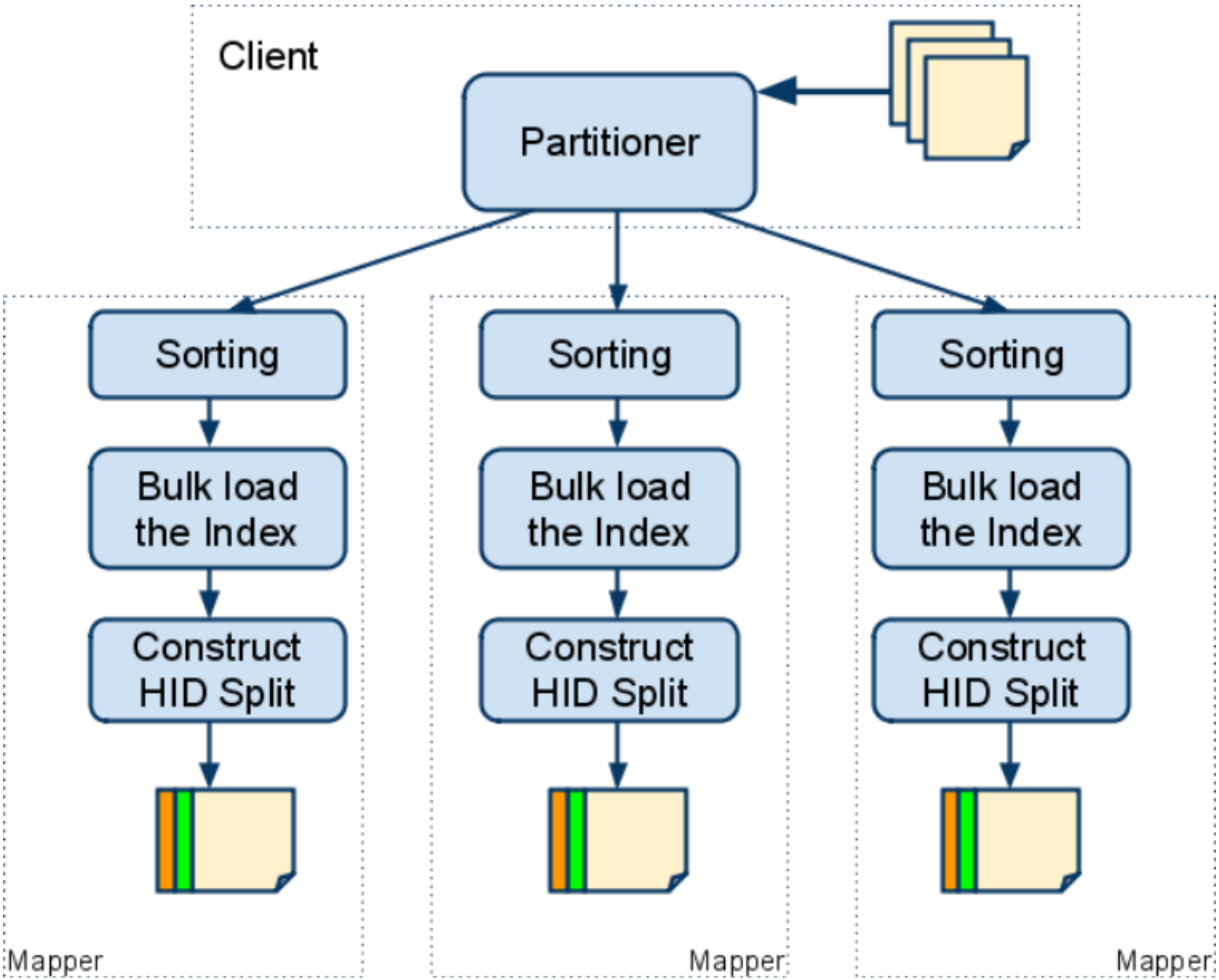
Indexing

- Data Loading
  - index and load data into DFS
- Query Execution
  - index look-up and selection
- Implementation on Hadoop

Data Loading

- Partitioning
- Sorting
- Bulk Loading
- HID Splits

Data Loading

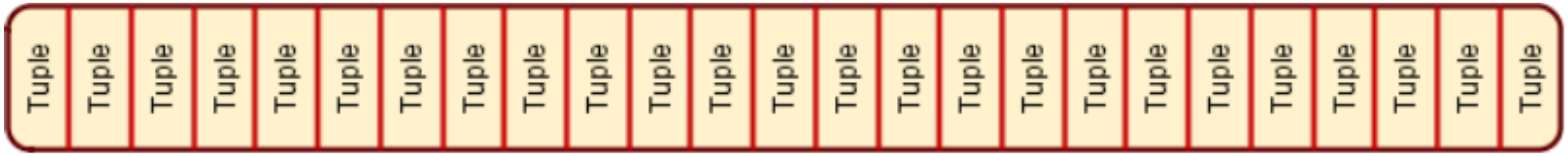


Partitioning

Split input data at tuple boundaries

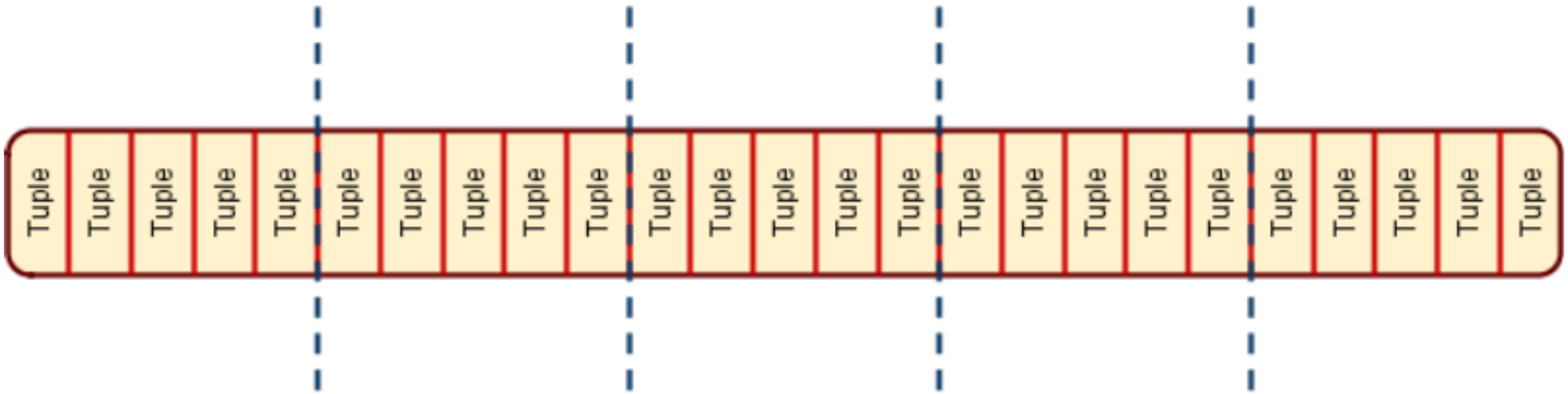
Partitioning

Split input data at tuple boundaries



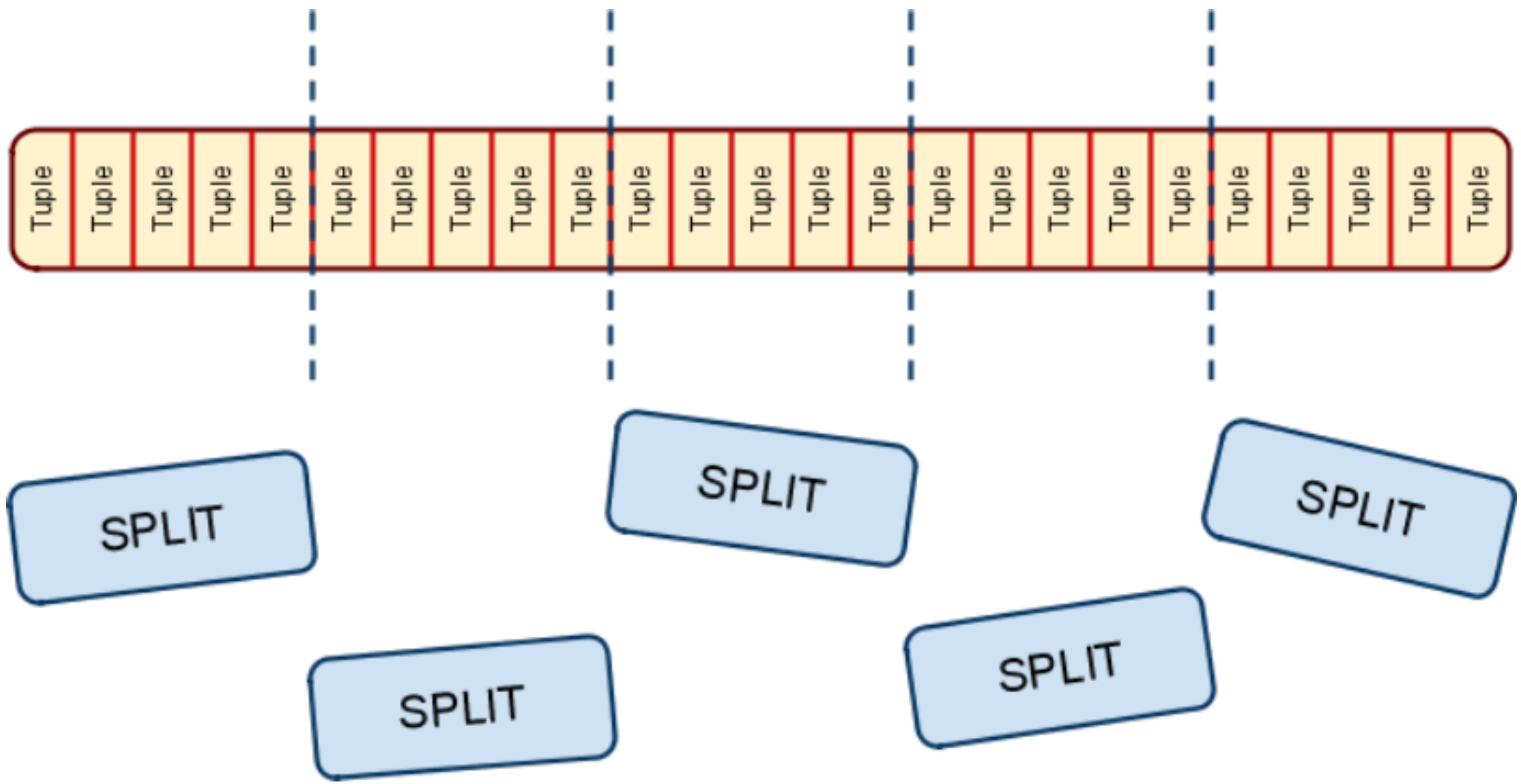
Partitioning

Split input data at tuple boundaries



Partitioning

Split input data at tuple boundaries



Sorting

Sort each split on the index key

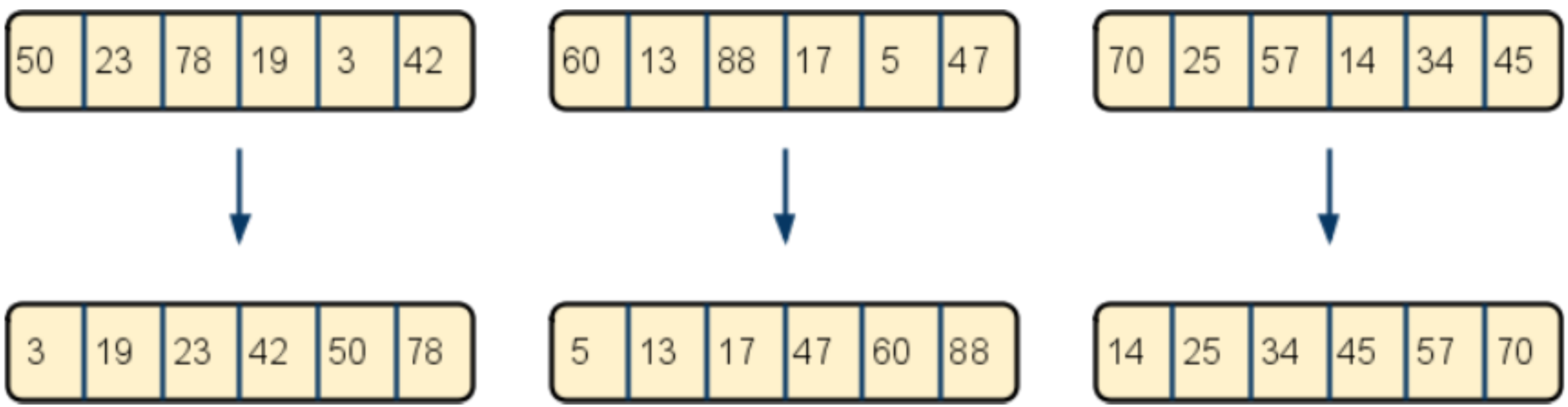
50	23	78	19	3	42

60	13	88	17	5	47

70	25	57	14	34	45

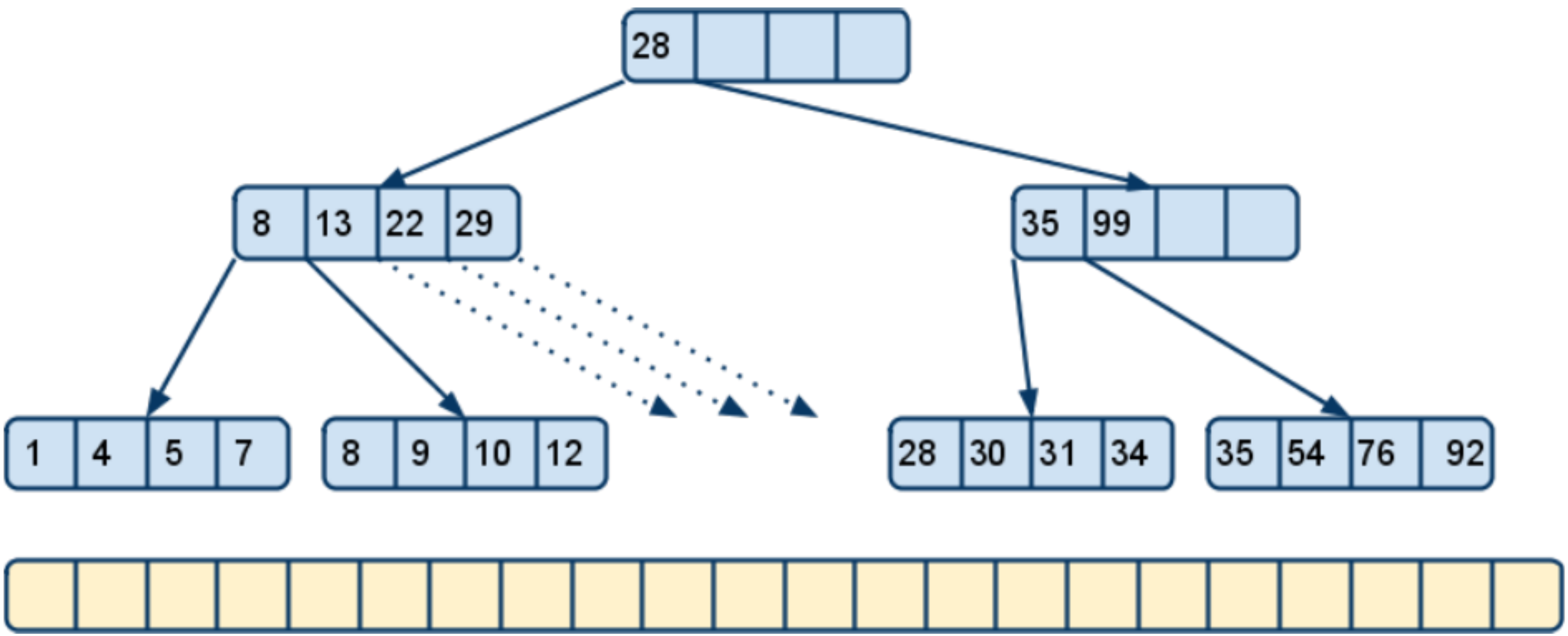
Sorting

Sort each split on the index key



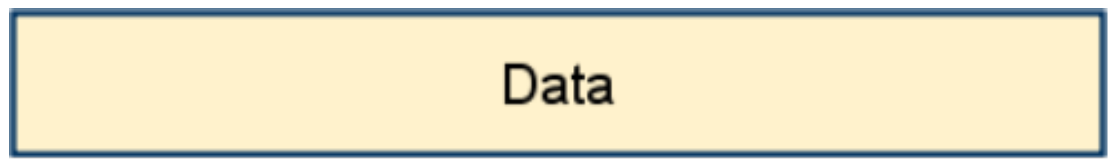
Bulk Loading

Bulk load CSS tree index



HID Split

Construct ***H**eader-**I**ndex-**D**ata* Split



HID Split

Construct *__H__eader-__I__ndex-__D__ata* Split

HID Split

Construct ***H****eader-****I****ndex-****D****ata* Split

Header: Index end offset
Data end offset
Start index key
End index key

HID Split

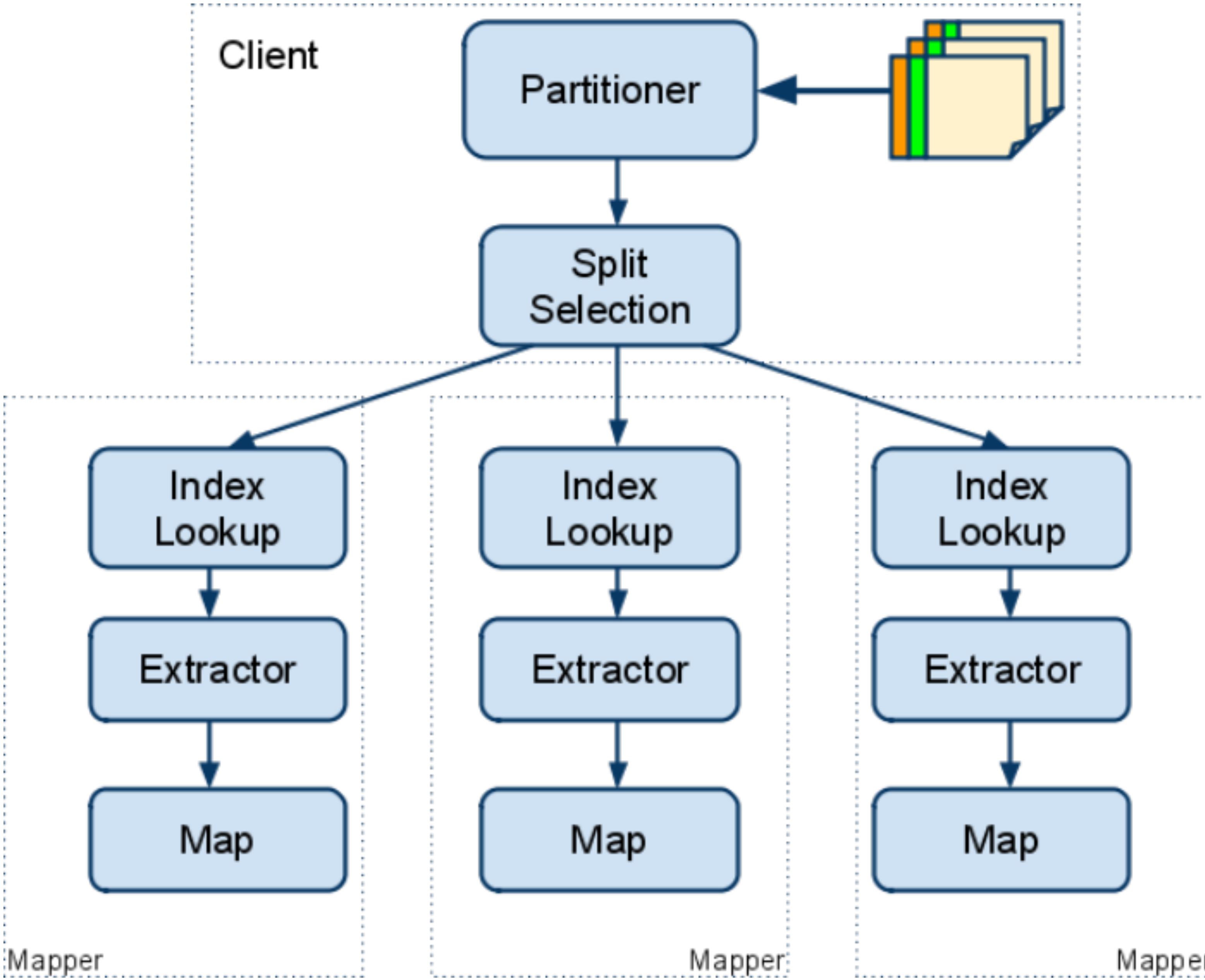
Construct ***H***_eader_***-I***_ndex_***-D***_ata_ Split

Header: Index end offset
Data end offset
Start index key
End index key

Query Execution

- Partitioning
- Split selection
- Index lookup
- Extractor

Query Execution



Partitioning

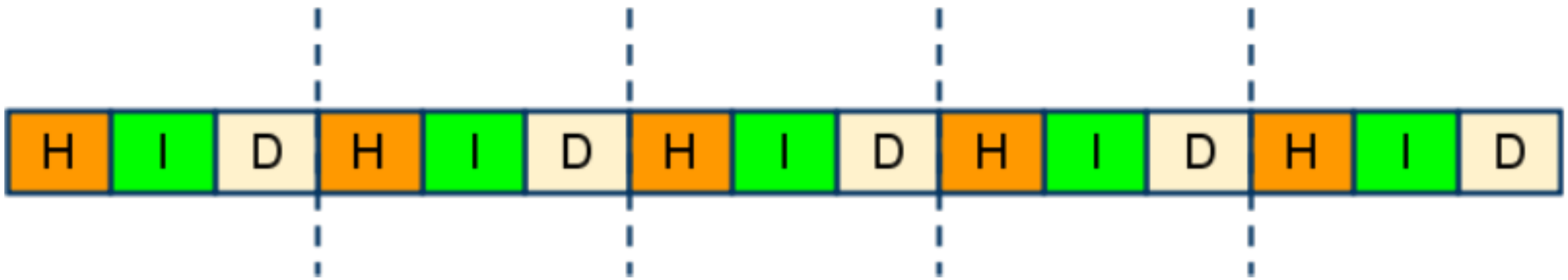
Read header to get HID boundaries

Partitioning

Read header to get HID boundaries

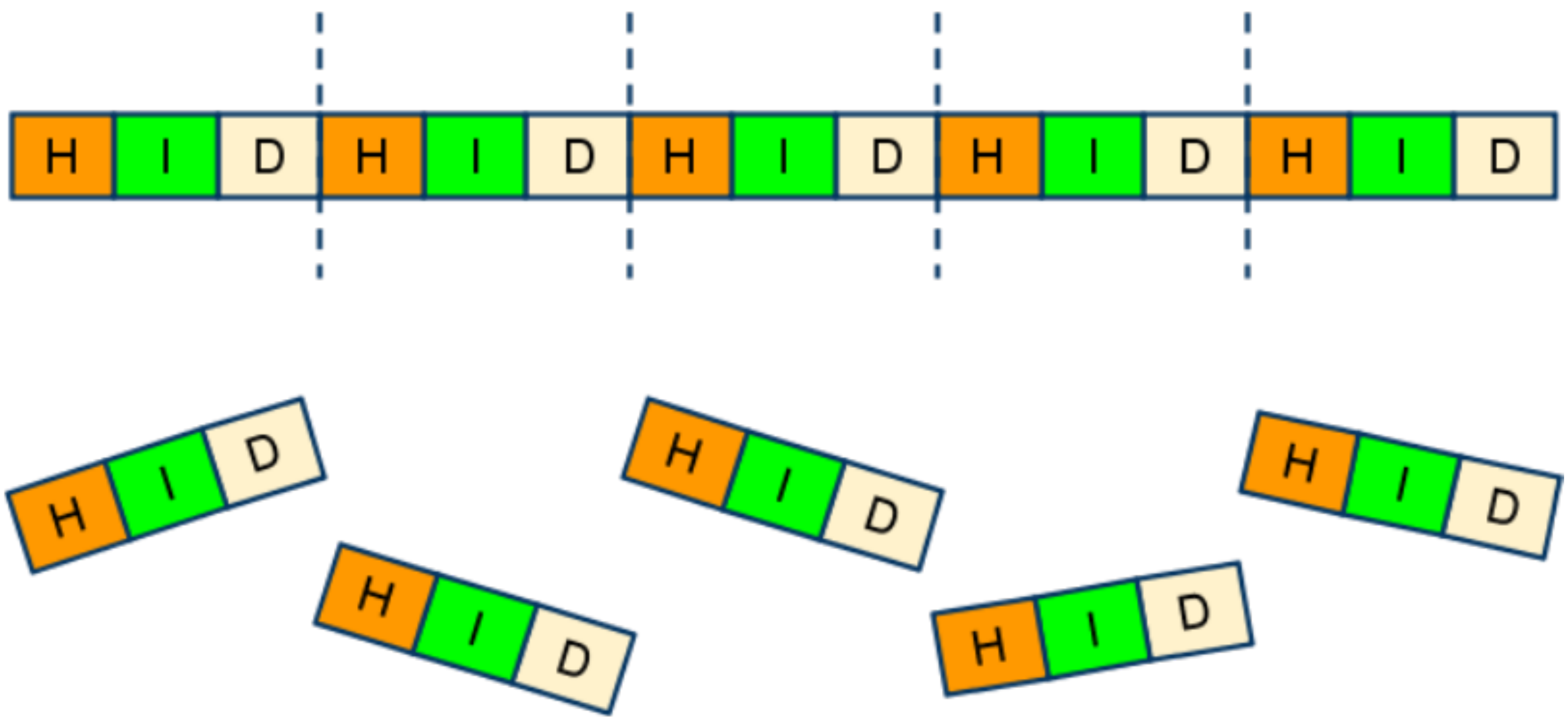
Partitioning

Read header to get HID boundaries



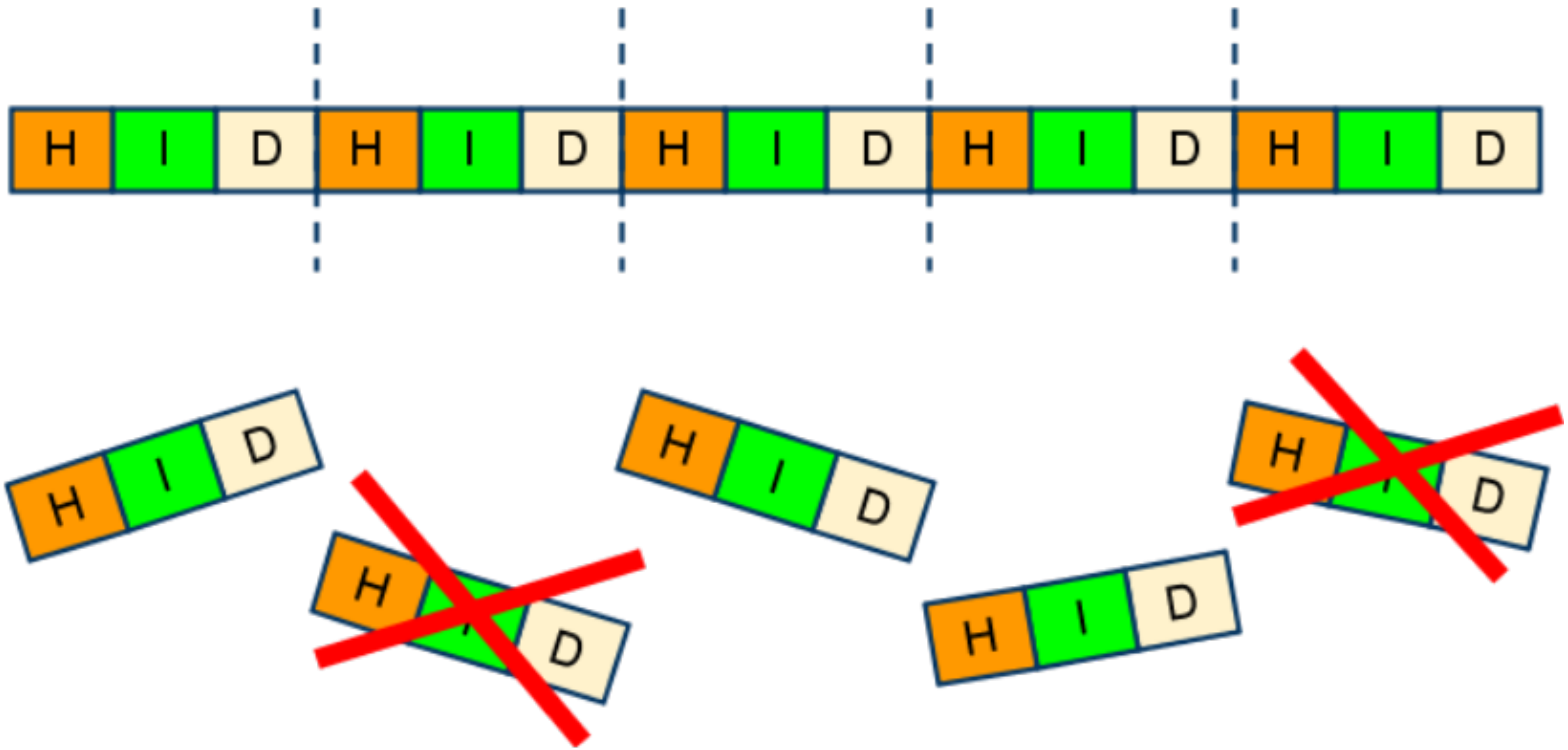
Partitioning

Read header to get HID boundaries



Split Selection

Discard splits containing out of range index keys

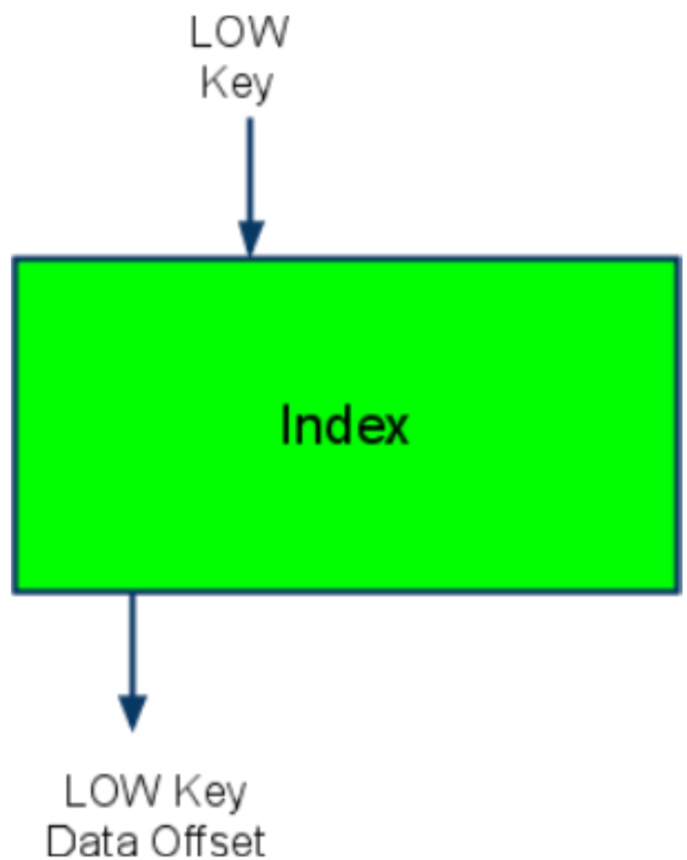


Index Lookup

Find data offsets corresponding to LOW and HIGH keys

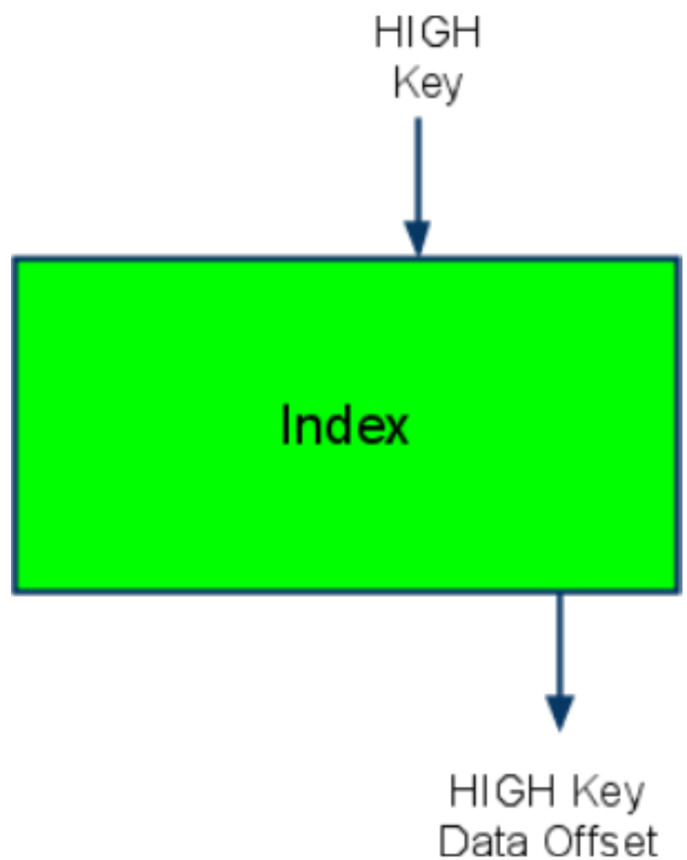
Index Lookup

Find data offsets corresponding to LOW and HIGH keys



Index Lookup

Find data offsets corresponding to LOW and HIGH keys



Index Lookup

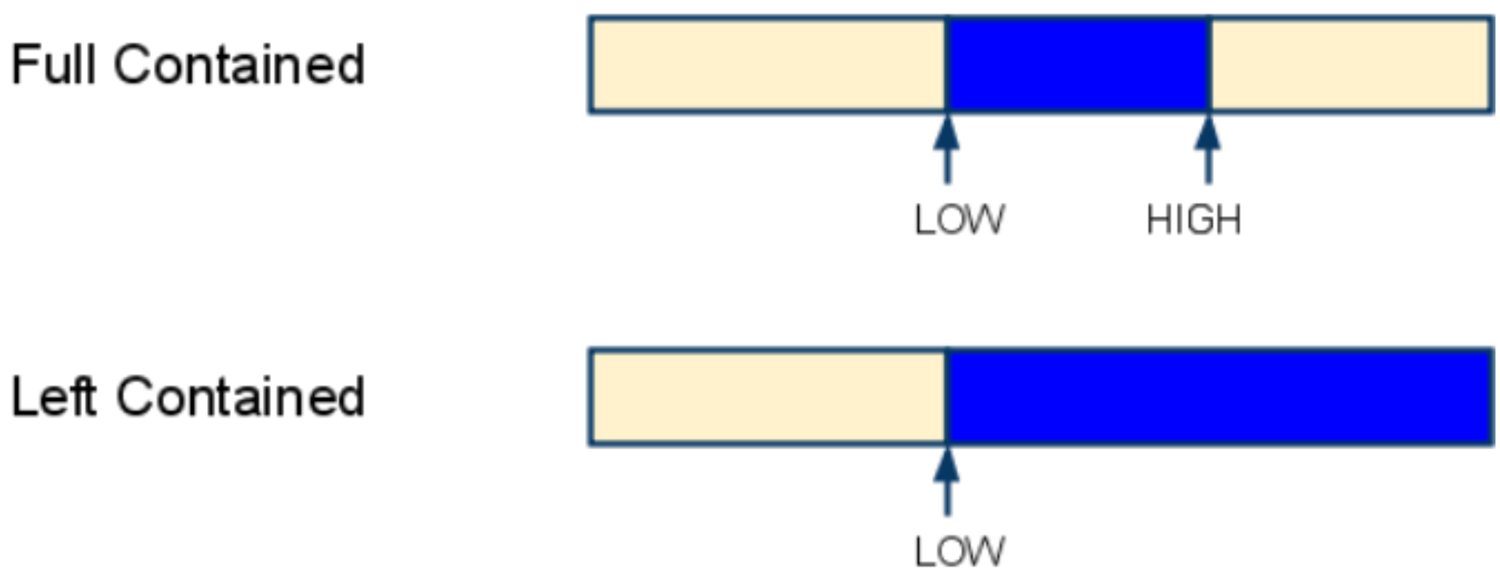
Find data offsets corresponding to LOW and HIGH keys

Index Lookup

Find data offsets corresponding to LOW and HIGH keys

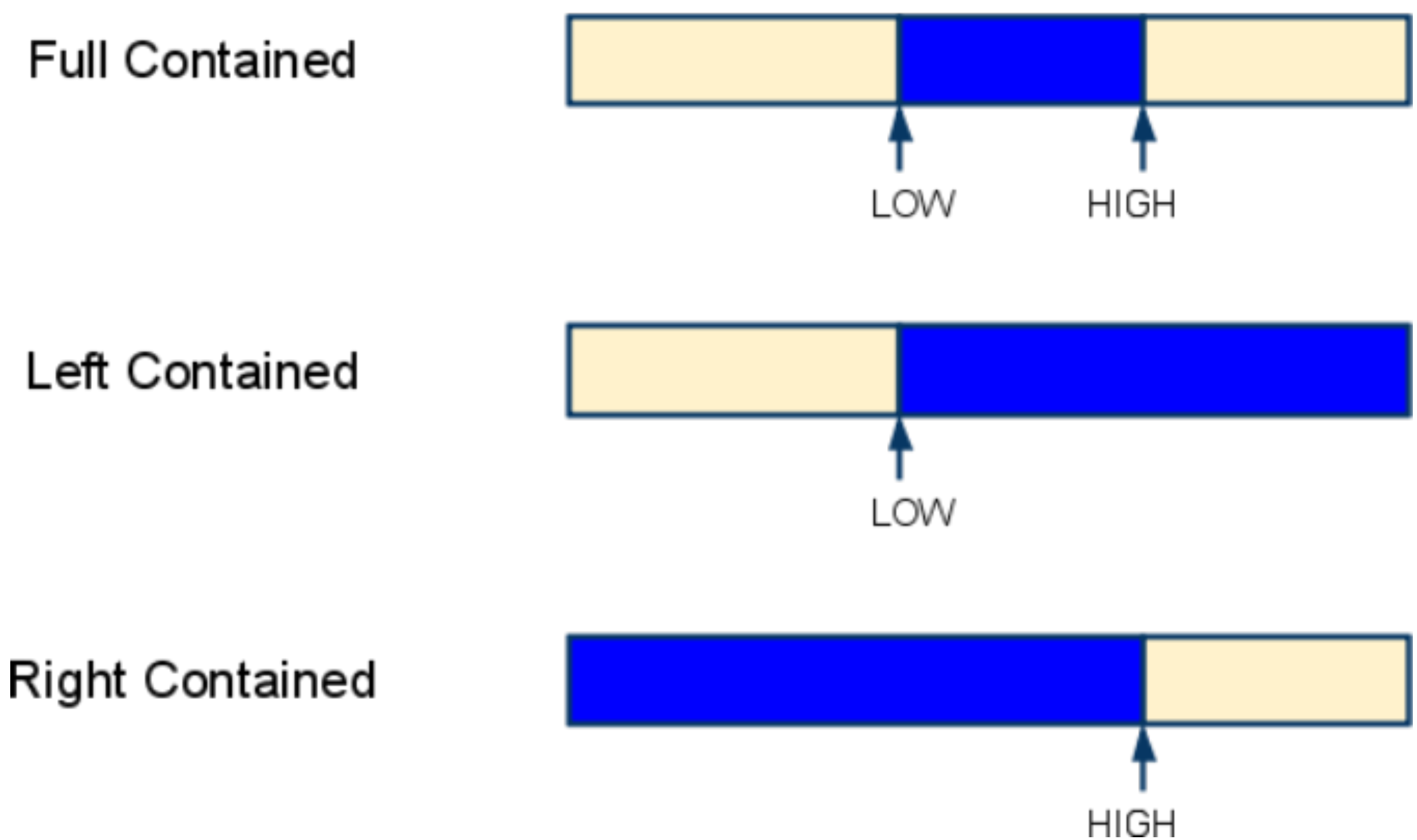
Index Lookup

Find data offsets corresponding to LOW and HIGH keys



Index Lookup

Find data offsets corresponding to LOW and HIGH keys

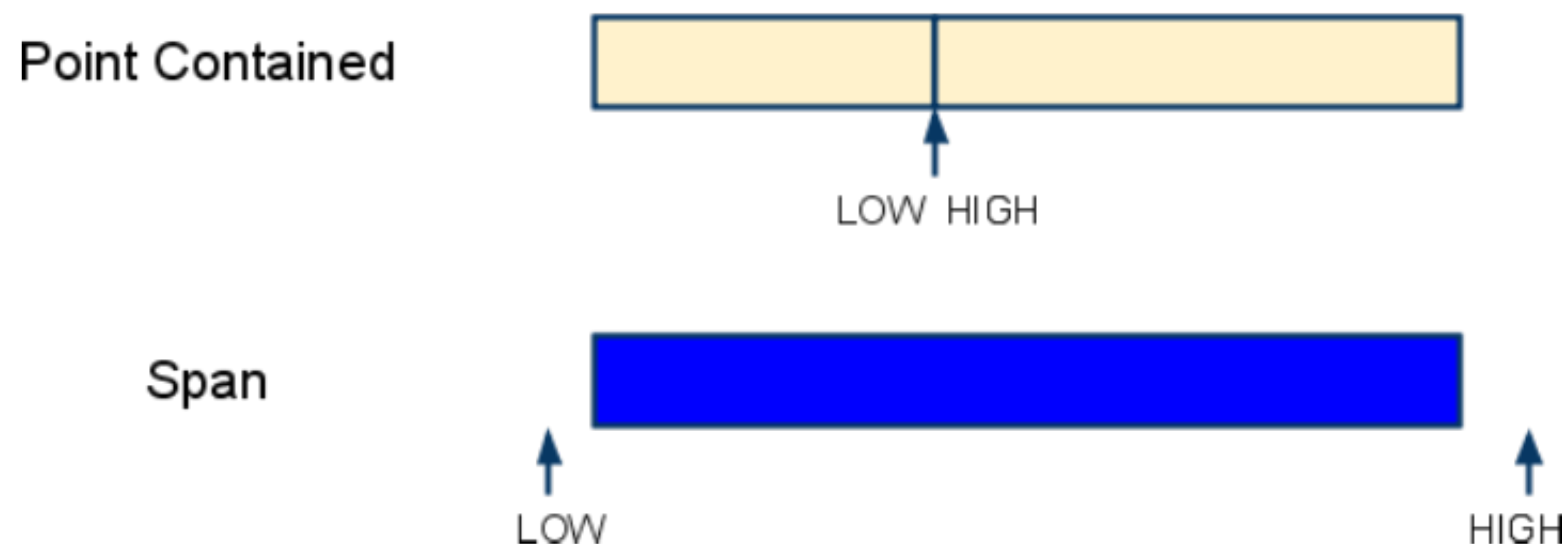


Index Lookup

Find data offsets corresponding to LOW and HIGH keys

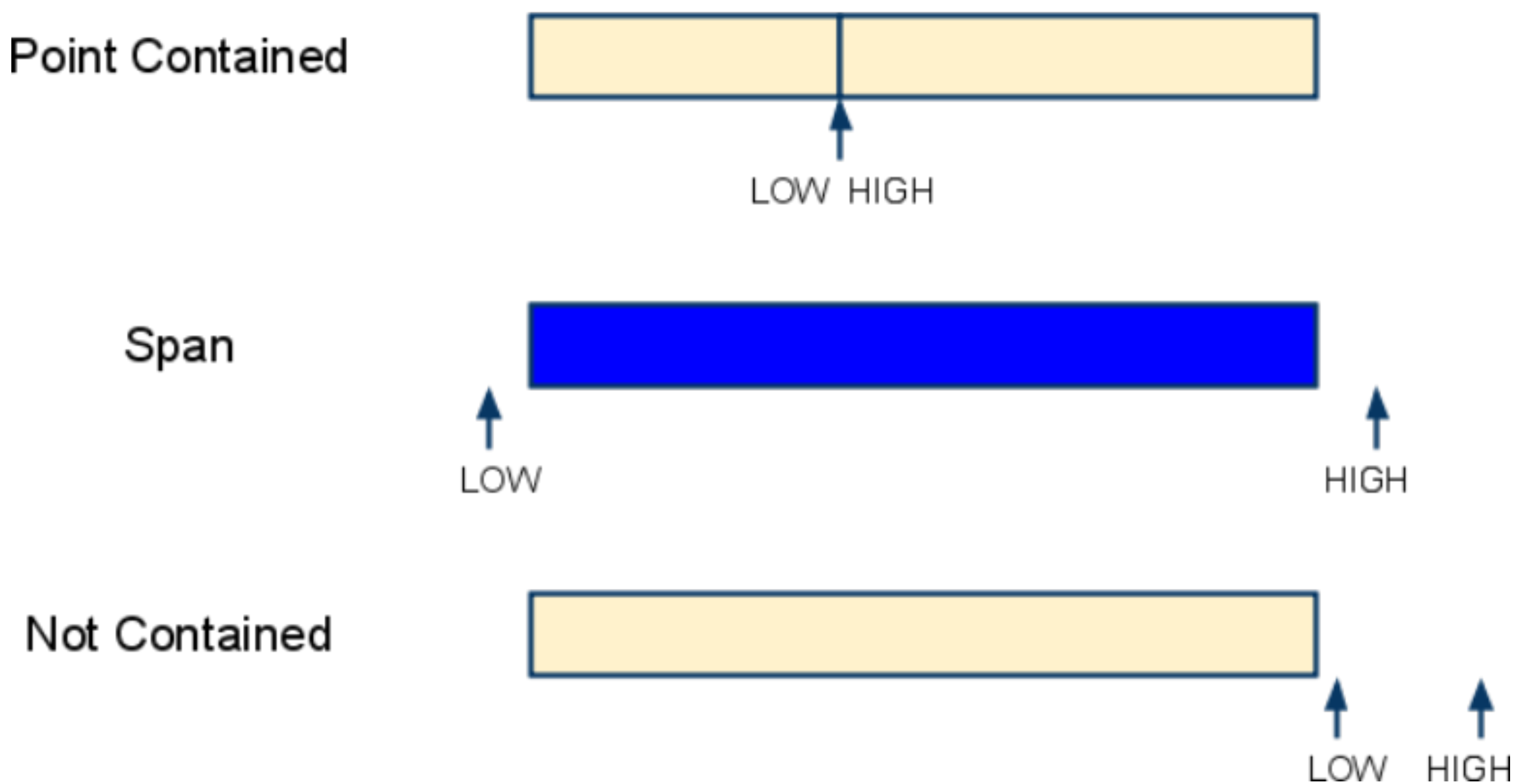
Index Lookup

Find data offsets corresponding to LOW and HIGH keys



Index Lookup

Find data offsets corresponding to LOW and HIGH keys

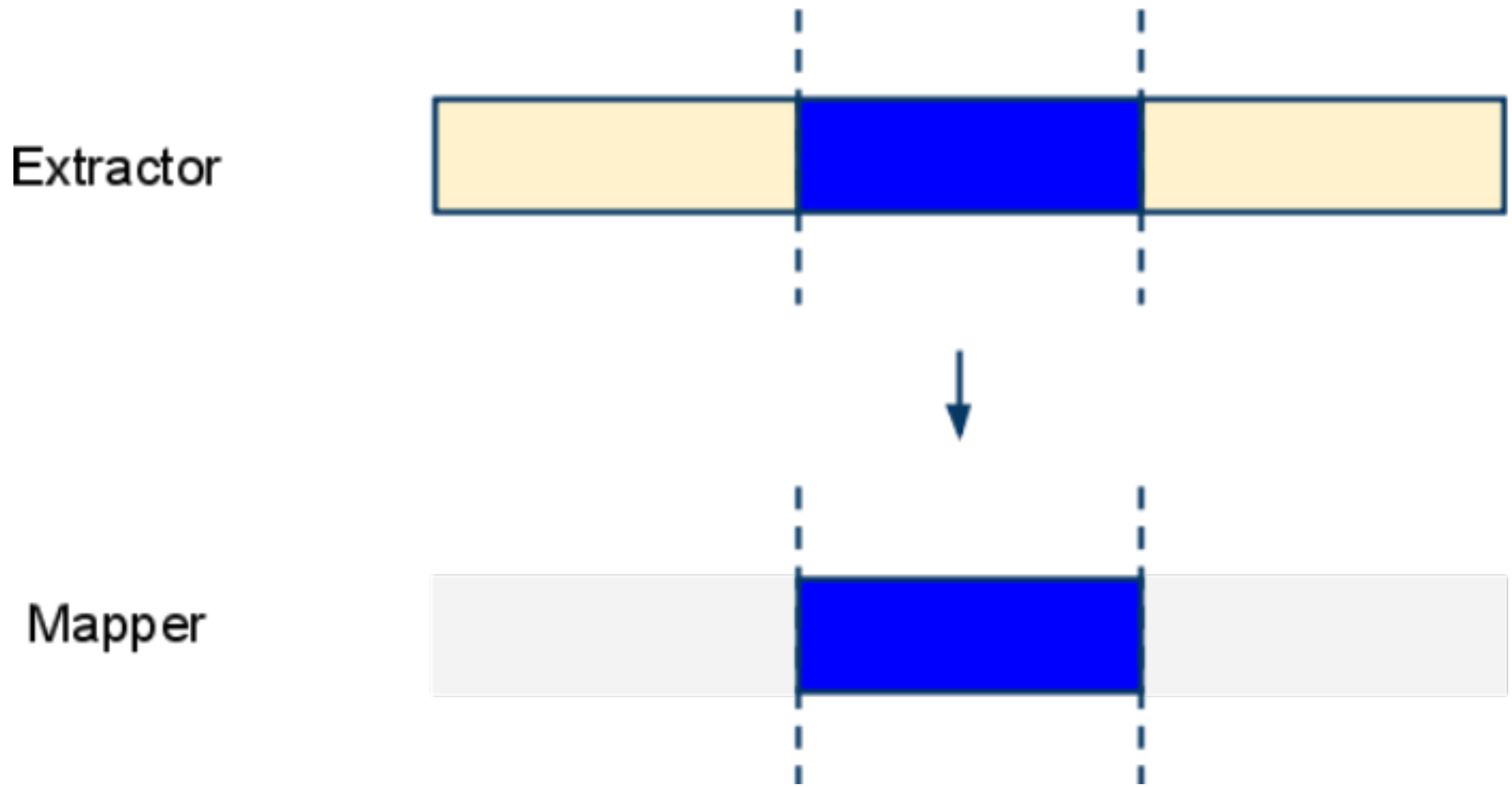


Extractor

Perform selection on data

Extractor

Pass sub-split to Record Reader for processing



Implementation on Hadoop

Loading

- CSS Tree Index
- Indirect index
- Four key types supported - Int, Float, Date, String
- Index stored as byte array
- Reducer to reduce number of files
- Integral number of HID splits per reducer output

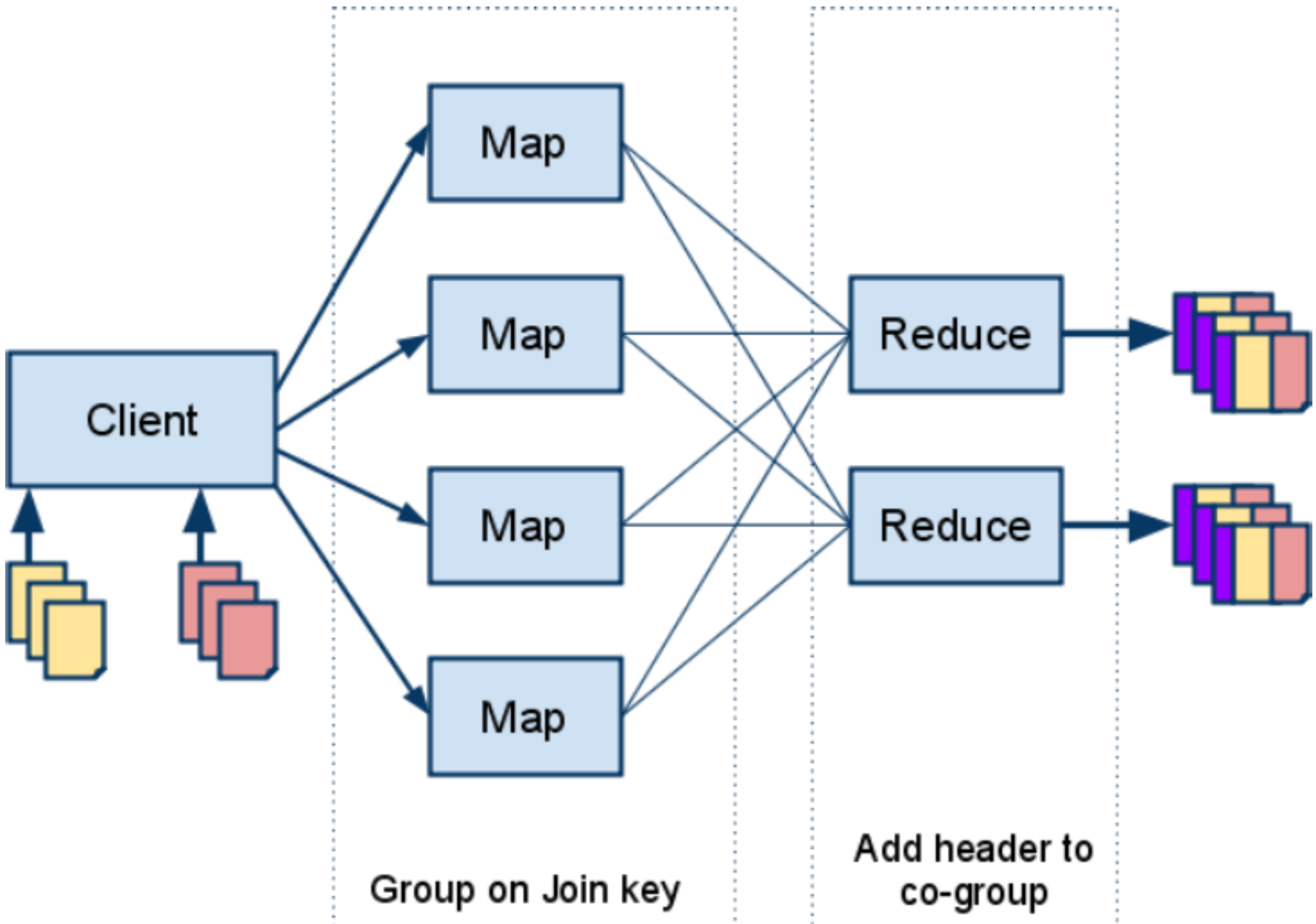
Querying

- Discover HID split boundaries from respective headers
- Read only the selected data from HDFS

Co-Partitioning

- Data loading
- Query execution

Data Loading

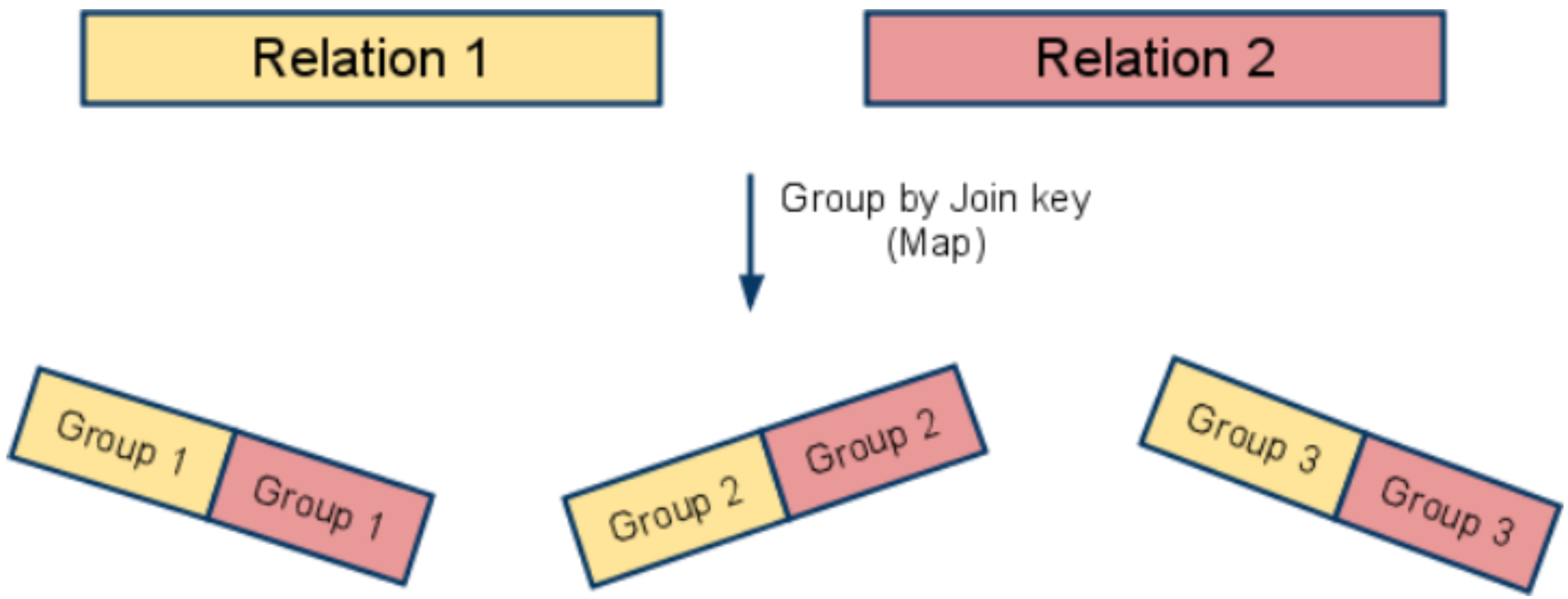


Data Loading

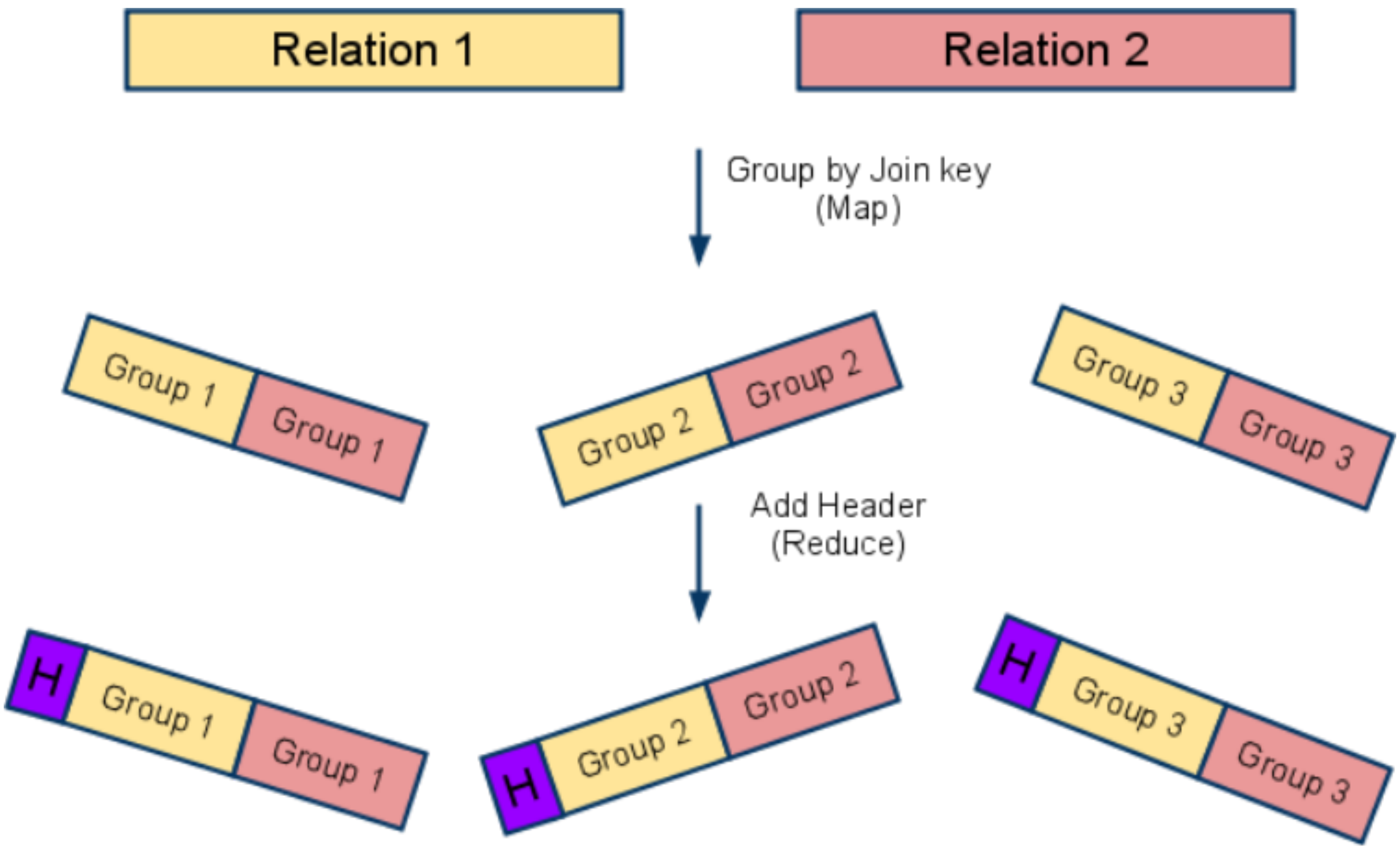
Relation 1

Relation 2

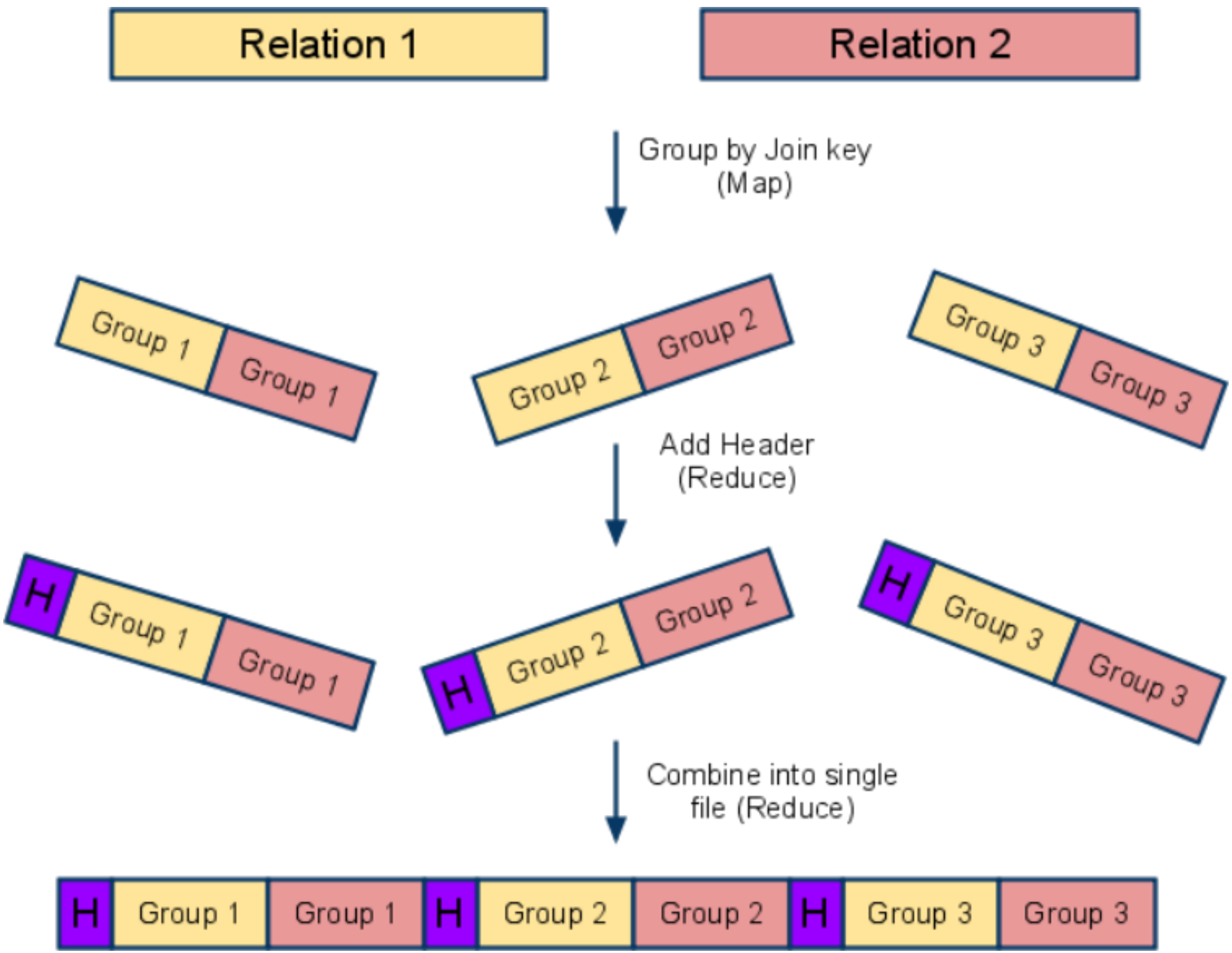
Data Loading



Data Loading

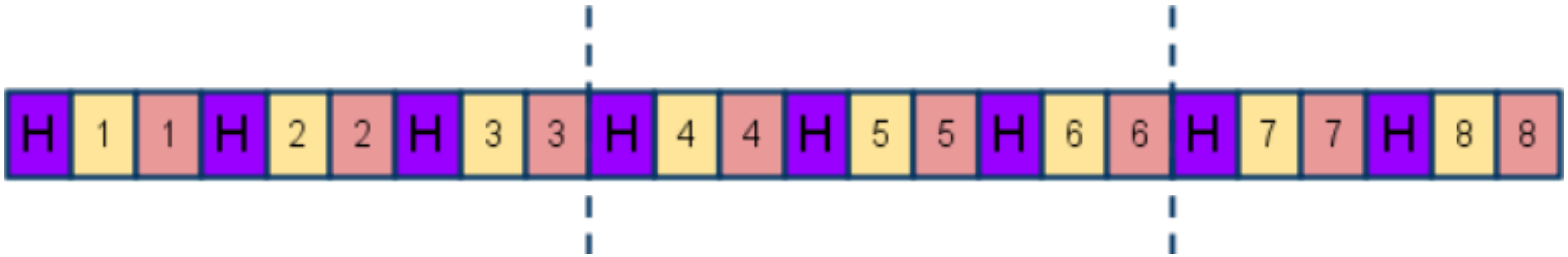


Data Loading

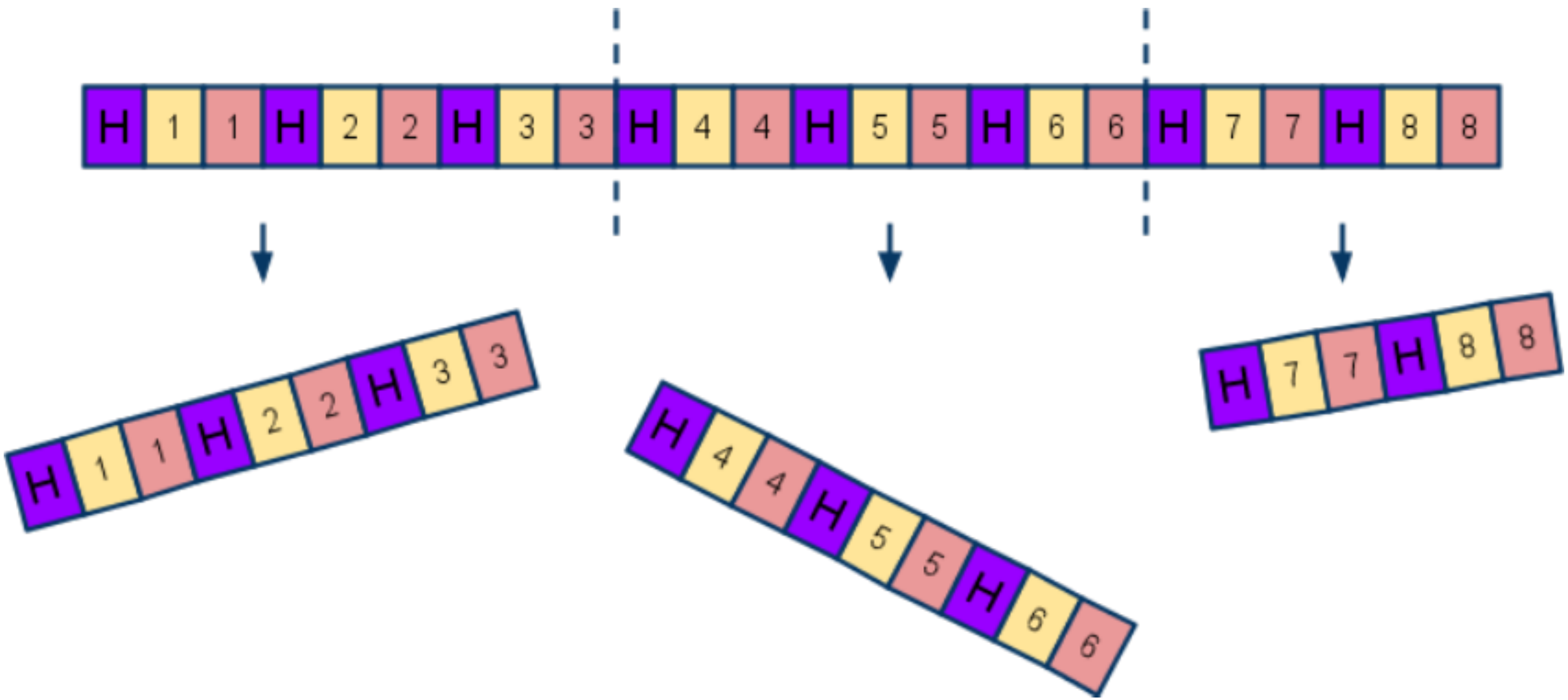


Query Execution

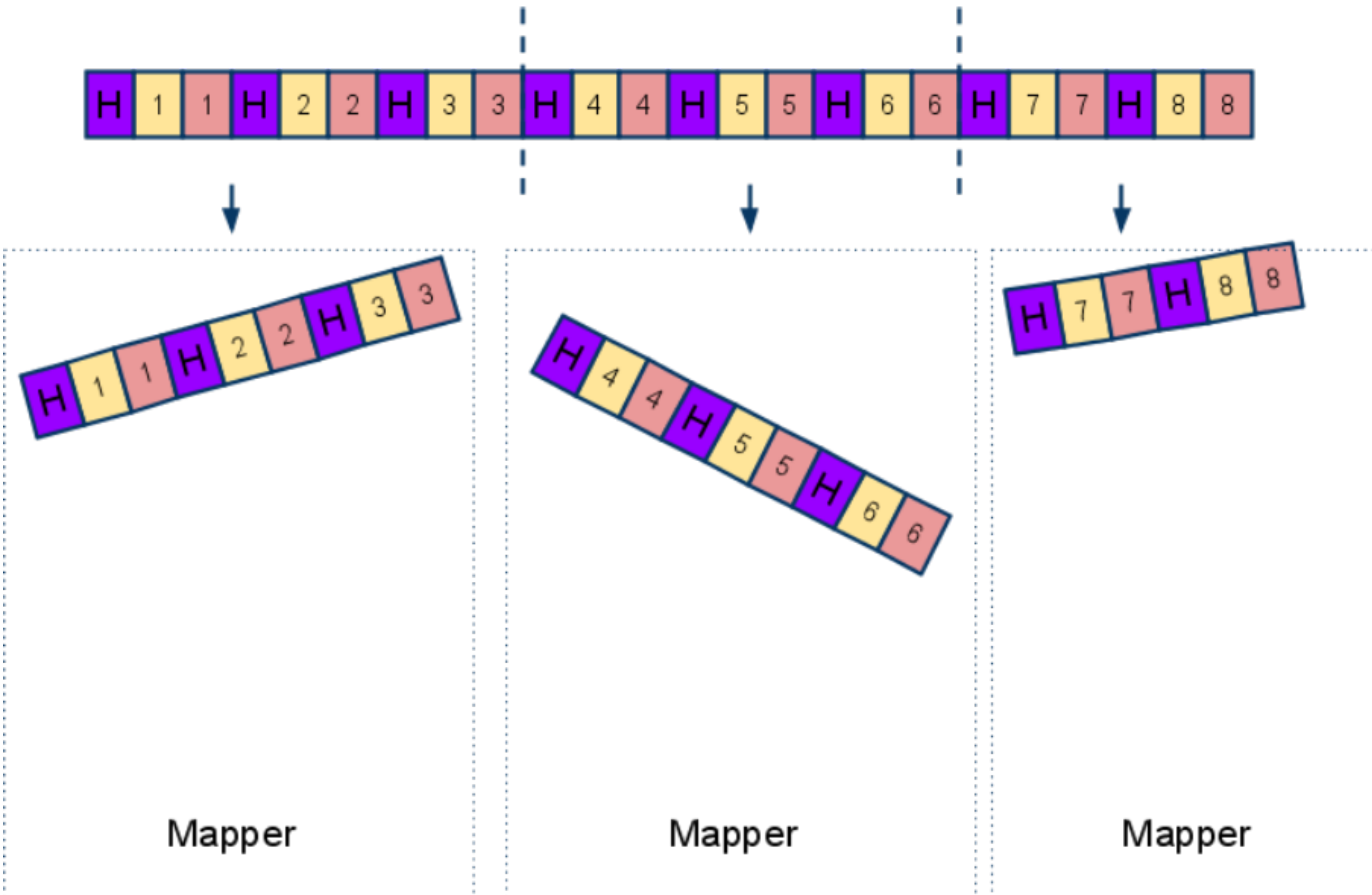
Query Execution



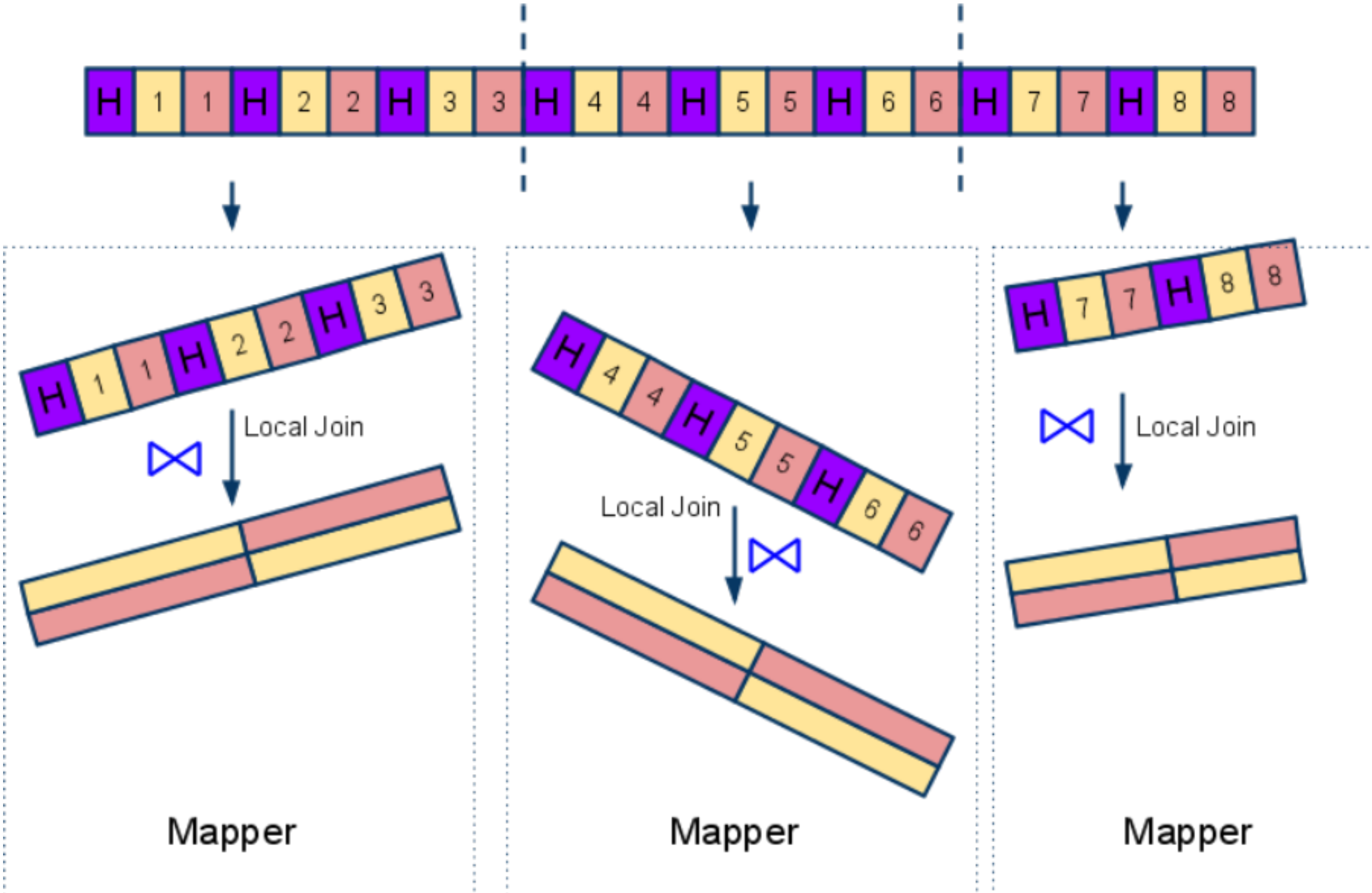
Query Execution



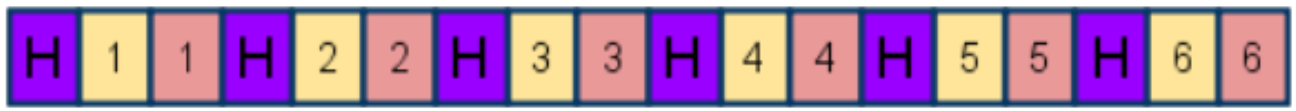
Query Execution



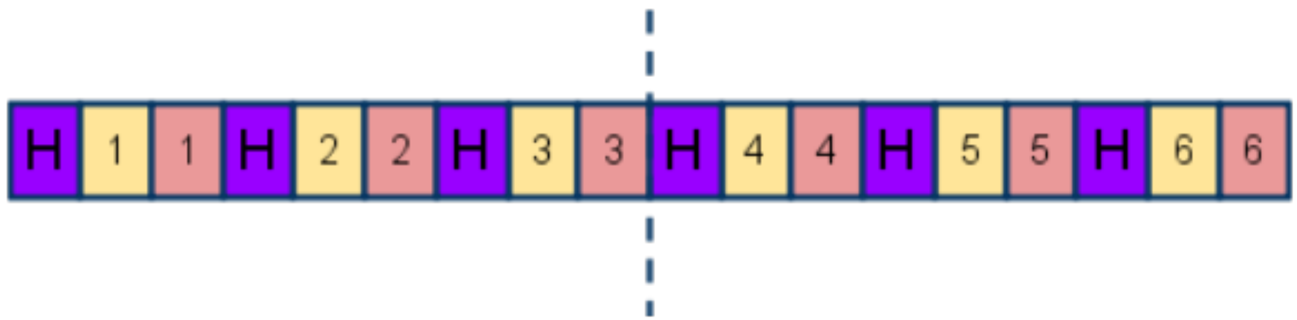
Query Execution



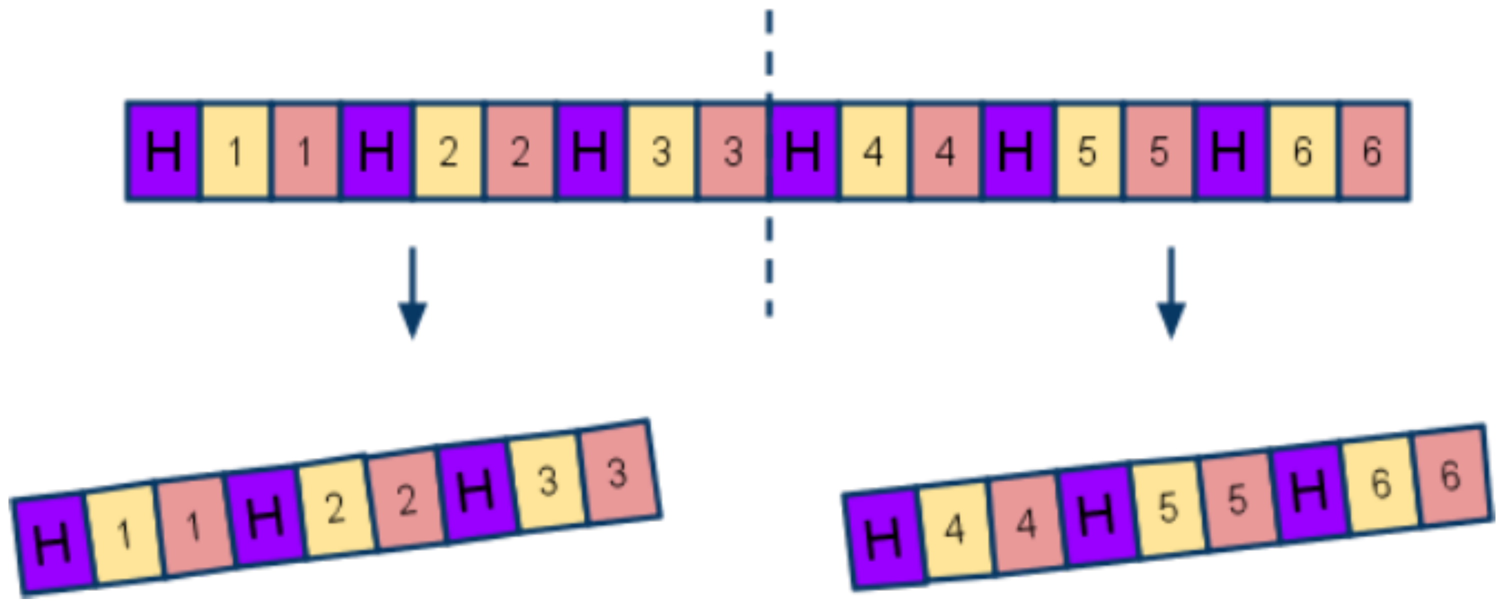
Indexing on top of Co-partitioning



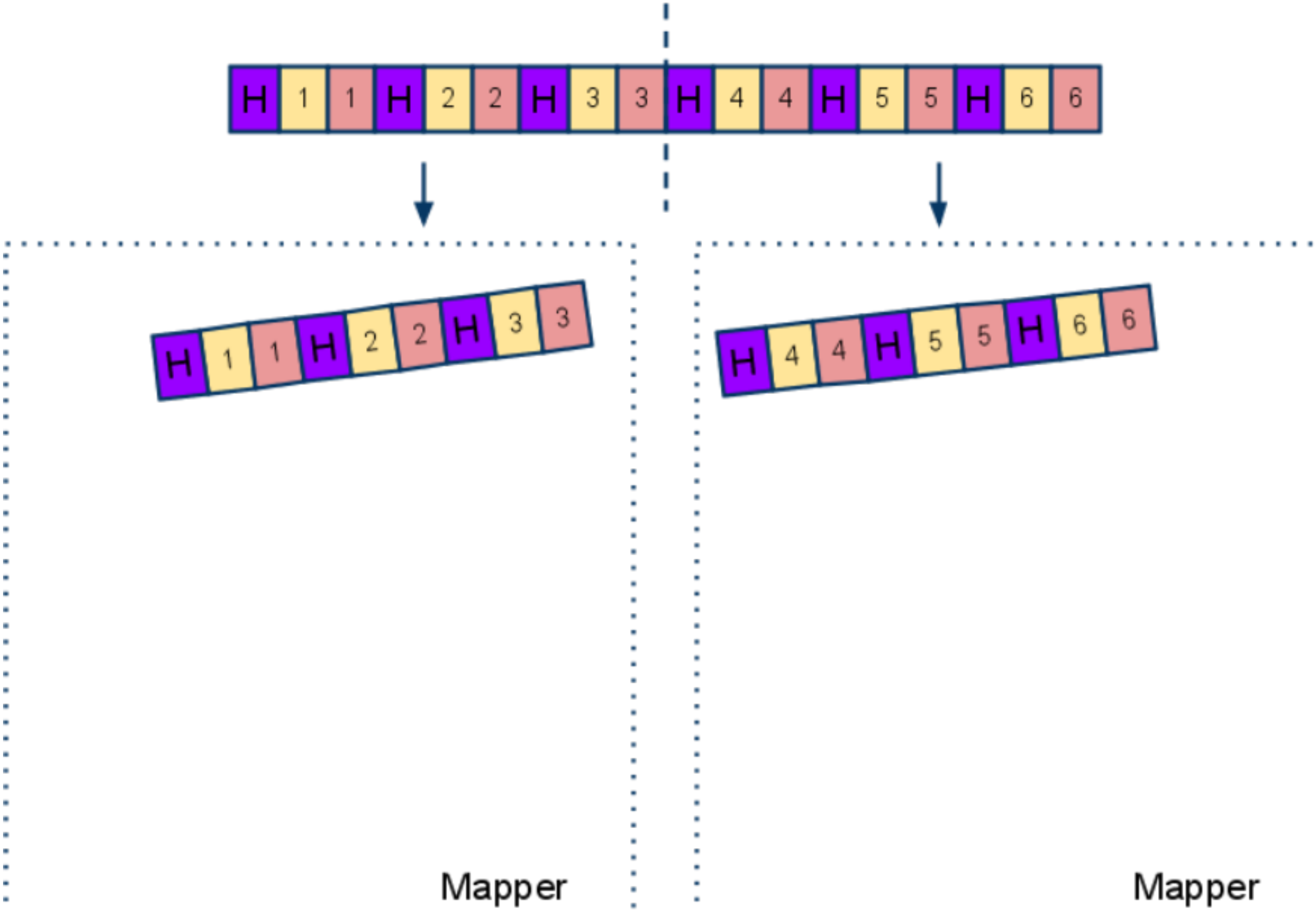
Indexing on top of Co-partitioning



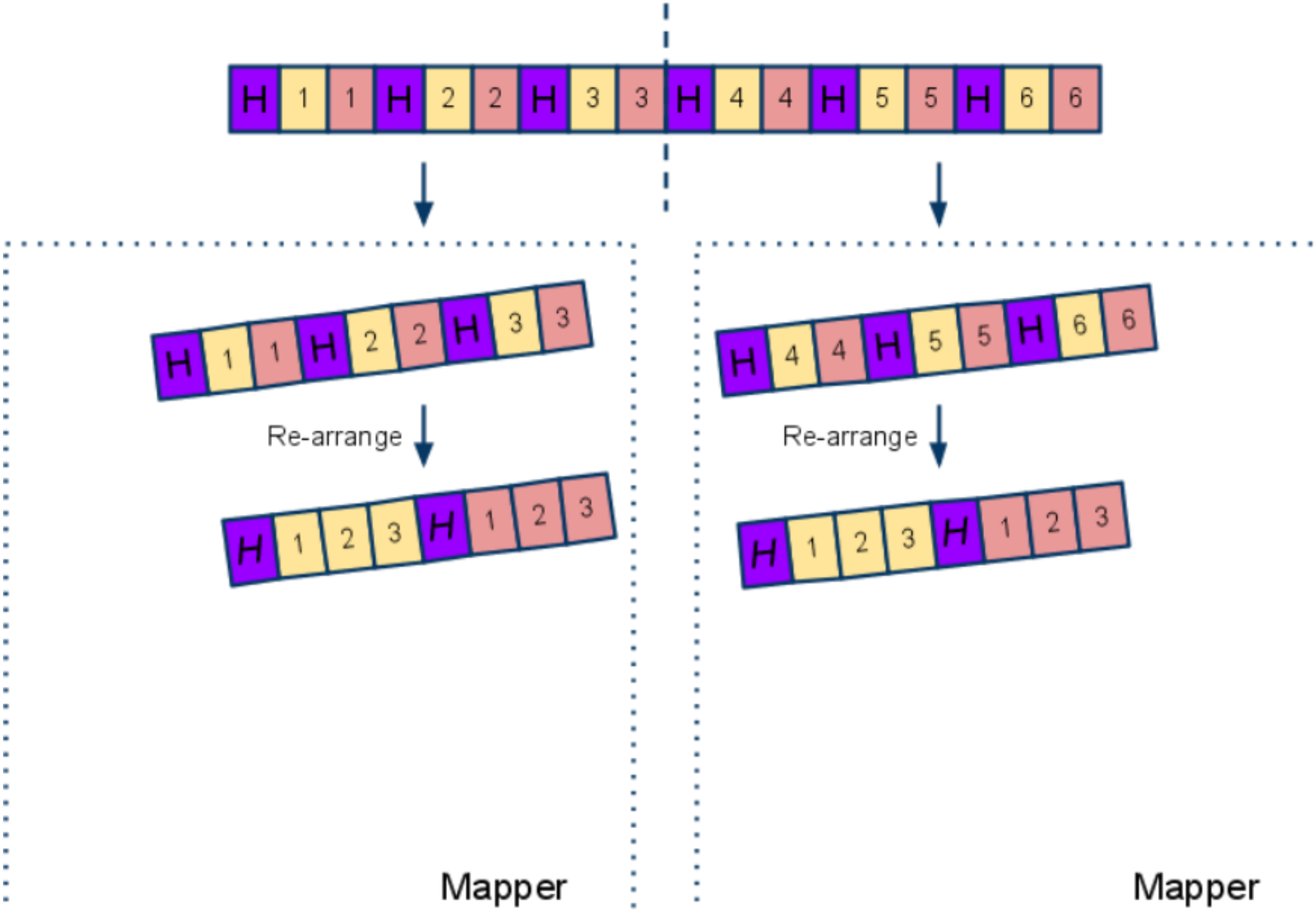
Indexing on top of Co-partitioning



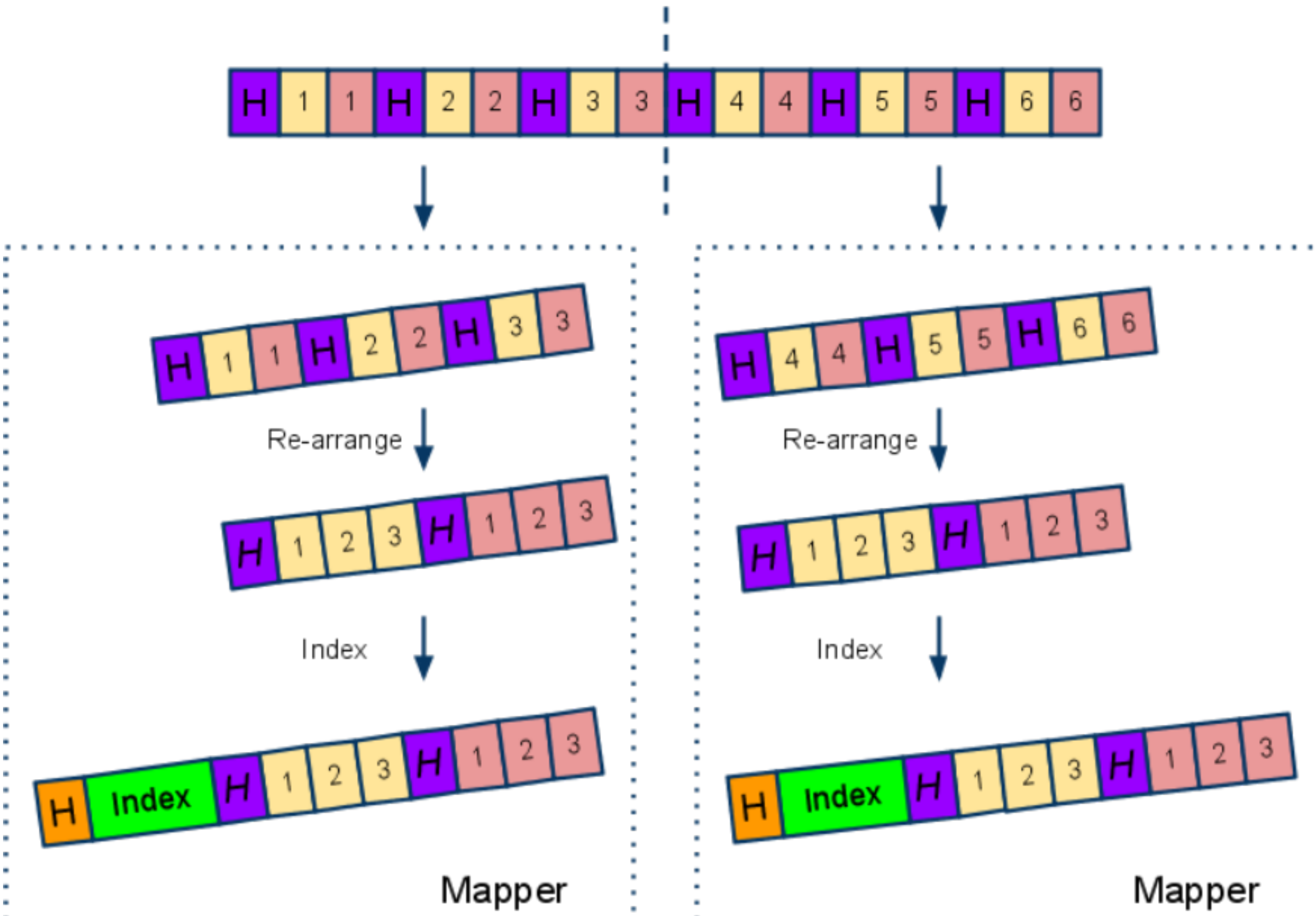
Indexing on top of Co-partitioning



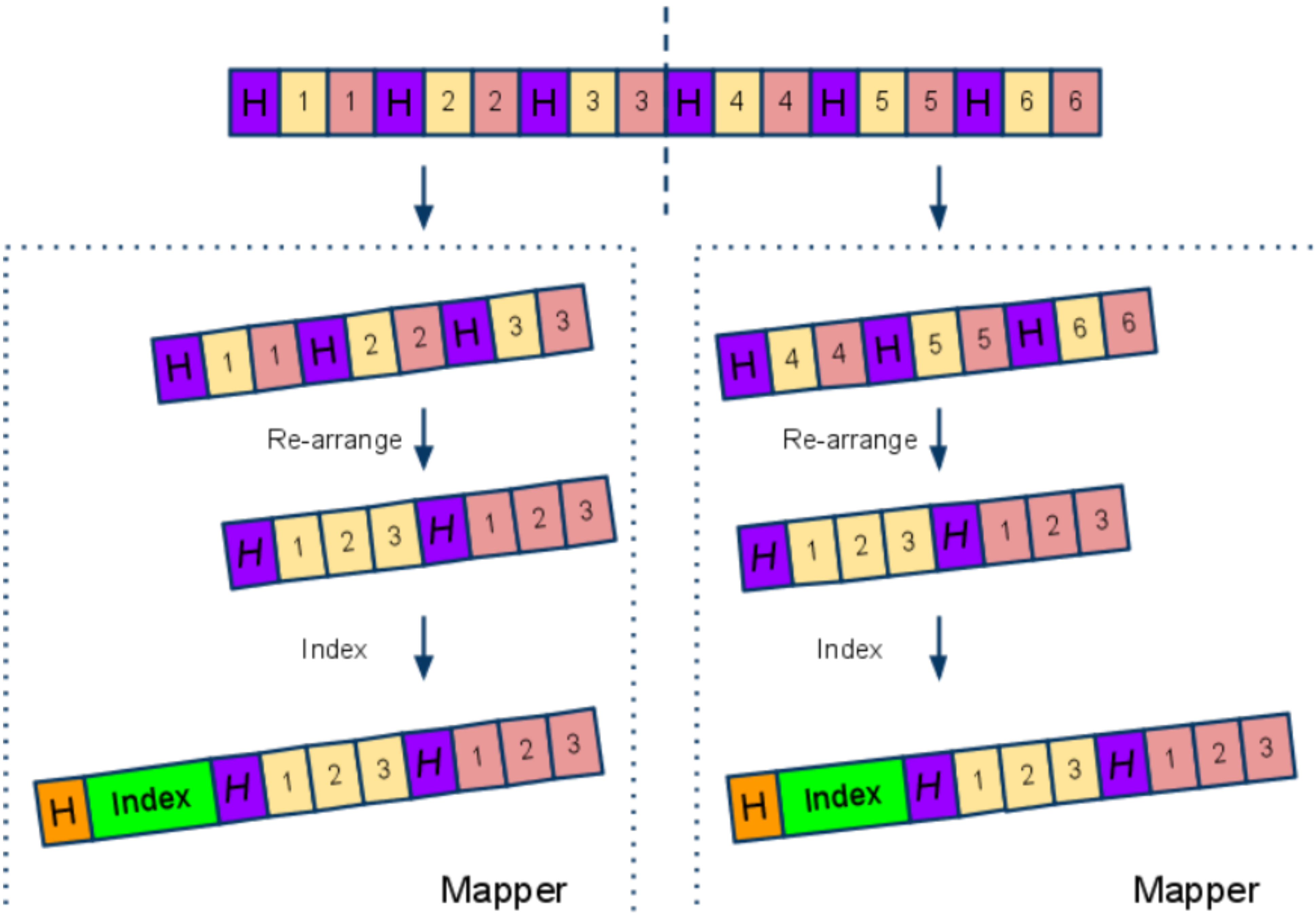
Indexing on top of Co-partitioning



Indexing on top of Co-partitioning



Indexing on top of Co-partitioning



Experiments

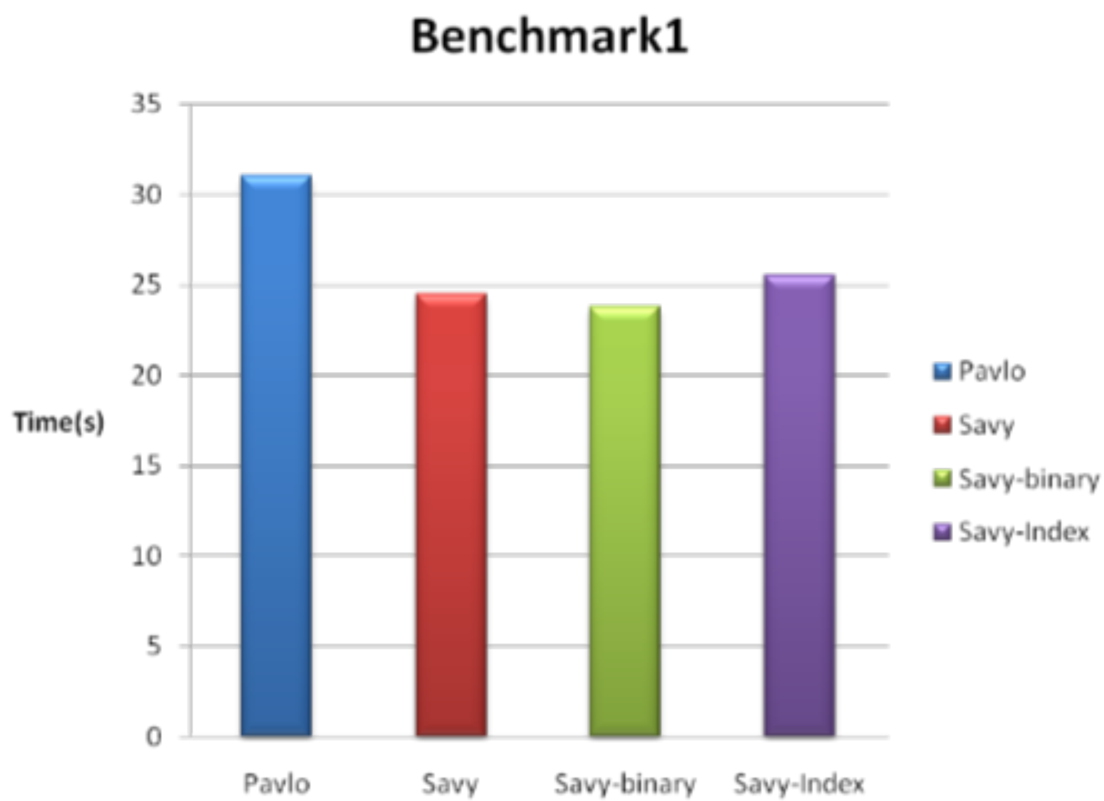
Experimental Setup

- Hadoop 0.19.1
- 5 nodes
- Speed?
- RAM?
- Gigabit Ethernet
- Data size
    - User Visits: 20GB
    - Rankings: 32MB

Results

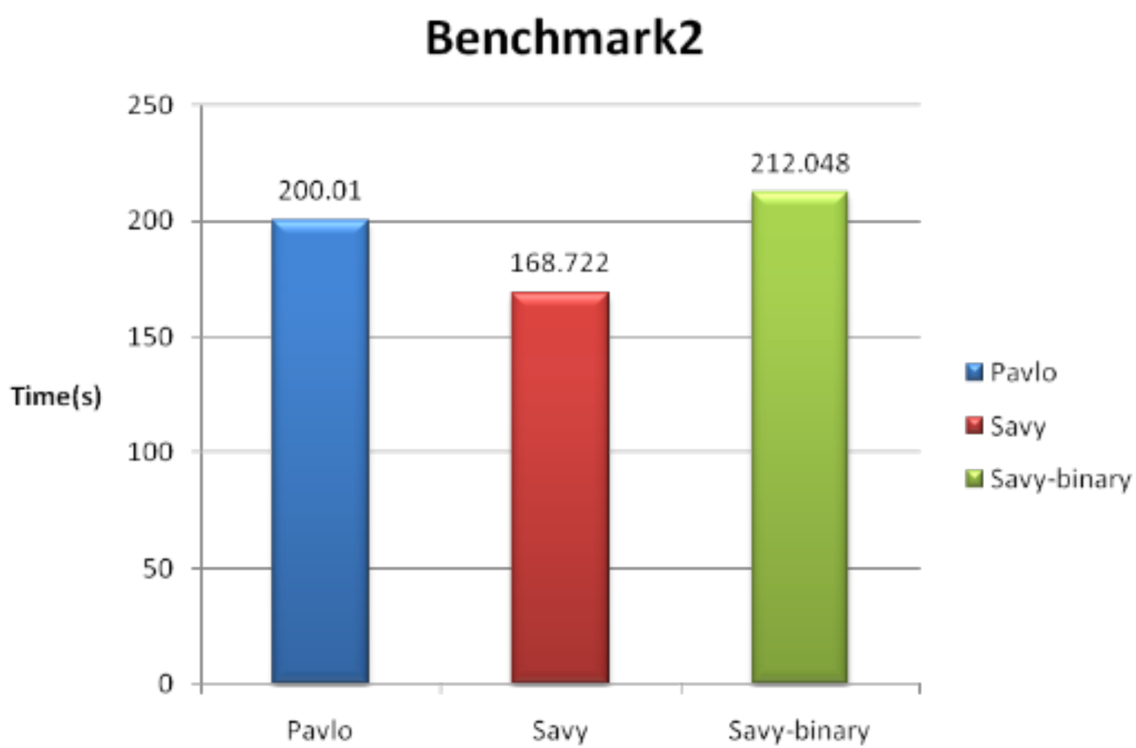
Data Size
40000
35000
30000
25000
20000
15000
10000
5000
0
32M
19G
64M
35G
Text
Binary
Rankings
UserVisits

Results

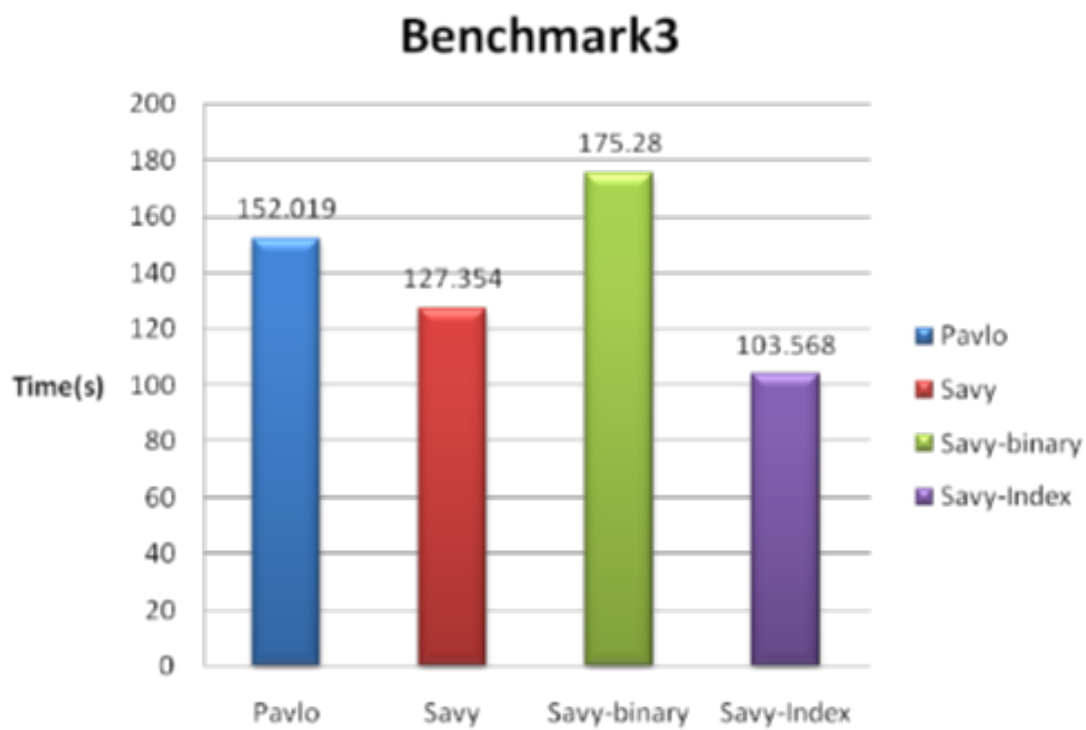


Benchmark1
35
30
25
20
15
10
5
0
Time(s)
Pavlo
Savy
Savy-binary
Savy-Index
Pavlo
Savy
Savy-binary
Savy-Index

Results

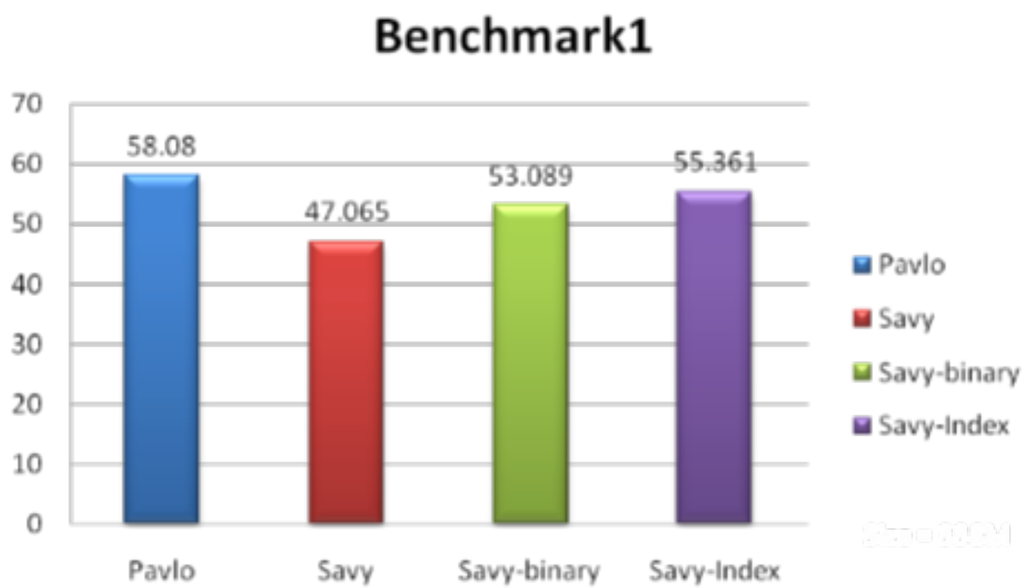

Benchmark2
Time(s)
250
200
150
100
50
0
200.01
168.722
212.048
Pavlo
Savy
Savy-binary
Pavlo
Savy
Savy-binary

Results



Benchmark3
Time(s)
200
180
160
140
120
100
80
60
40
20
0
152.019
127.354
175.28
103.568
Pavlo
Savy
Savy-binary
Savy-Index

Results


Benchmark1
70
60
50
40
30
20
10
0
58.08
47.065
53.089
55.361
Pavlo
Savy
Savy-binary
Savy-Index
Pavlo
Savy
Savy-binary
Savy-Index

Roadblocks Faced

- Data generation:
  - 20GB UserVisits, 338MB Rankings in HDFS
  - Took 16 hours for generation
  - Too many OS/library dependencies
  - Poor documentation

- Number of nodes:
  - Allocated 6 nodes
  - Effective (up-and-running) 4 nodes
  - Map/Reduce parallelism not exploited
  - Per-split indexing ideally suited for highly parallel execution

Roadblocks Faced

- Data normalization
  - Schema uses VARCHAR data types
  - Input data normalized to fixed tuple-sized binaries
  - Byte oriented processing speedup negated by increased input size
  - However, facilitates indexing and co-partitioning

- Low selectivity
  - Selection task has selectivity close to 1
  - Indexing benefits are sabotaged

- Incorrect base result
  - Reported join task result was not correct

Roadblocks Faced

- Implementation deviation from the paper
  - Composite key is not really used in join task

Discussion: Loopholes

- Benchmarks are well suited (biased) for databases
- Huge difference in data loading time
- Queries make heavy use of indexing, sorting data
- Query optimization not done for Map/Reduce
- Fault tolerance not compared

Discussion: We can do better!

- Map/Reduce plans can be optimized
- Normalized binary input data can help
- Indexing feasible and performs good
- Co-partitioning feasible and looks promising

Conclusions

References

- Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., and Stonebraker, M. 2009. A comparison of approaches to large-scale data analysis. SIGMOD '09.
- DeWitt, D. and Gray, J. 1992. Parallel database systems: the future of high performance database systems. *Commun. ACM*35, 6 (Jun. 1992)