Last Lecture

Eugene Wu

Administrivia

AA4: git pull for bug fix
We will present project 1 highlights Thursday

Midterm review Friday 10AM location TBA
Scribenotes open until day of midterm

Midterm next Tuesday 8:30

- Same location as midterm 1
- 1 page 8x11 cheat sheet, both sides
- Phone-a-friend

DBMSes in the Wild

Classic Relational

- $$: Oracle, IBM, Microsoft, Teradata, EMC, etc
- Free: MySQL, PostgreSQL

New Relational

- In-Memory, Column-store, Streaming

Non-traditional

- Search (Google, Bing, Lucene), Scientific, Geographic

NoSQL

- Big Data: Hadoop, Spark, etc
- Key-value: Mongo, BerkeleyDB, Cassandra, etc

DBMS-as-a-Service

- Microsoft Azure, Amazon Redshift/RDS, etc…

DBMSes in the Wild

Classic Relational

- $$: Oracle, IBM, Microsoft, Teradata, EMC, etc
- Free: MySQL, PostgreSQL

New Relational

- In-Memory, Column-store, Streaming

Non-traditional

- Search (Google, Bing, Lucene), Scientific, Geographic

NoSQL

- Big Data: Hadoop, Spark, etc
- Key-value: Mongo, BerkeleyDB, Cassandra, etc

DBMS-as-a-Service

- Microsoft Azure, Amazon Redshift/RDS, etc…

Modern Database Systems

Hardware changes affect

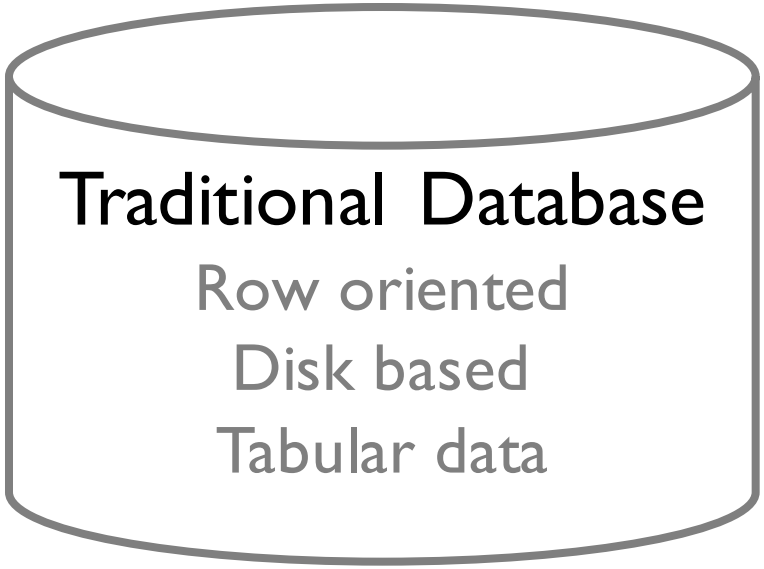
Compression is good → Column stores

- Large scale aggregation queries
- "Data Warehouses"

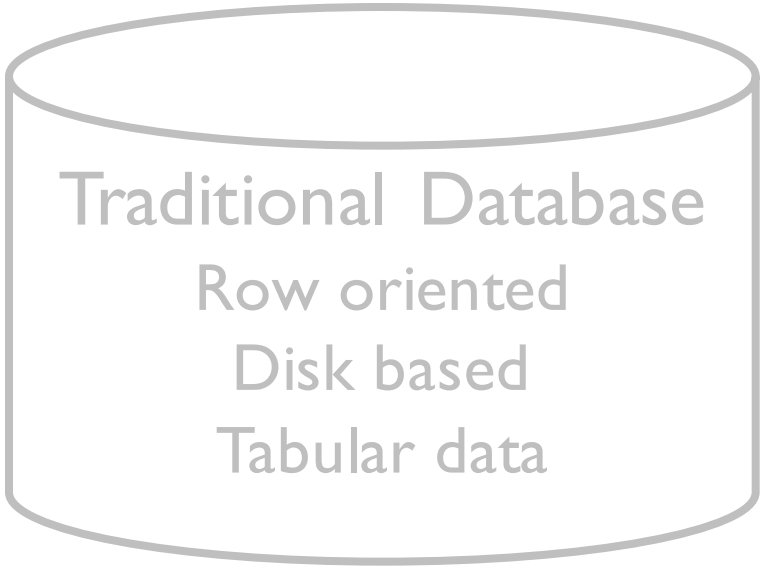
Memory is cheap → In-memory stores

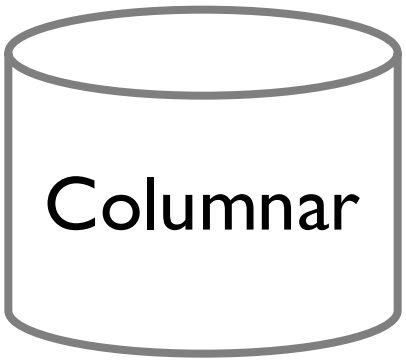
- Transactional systems

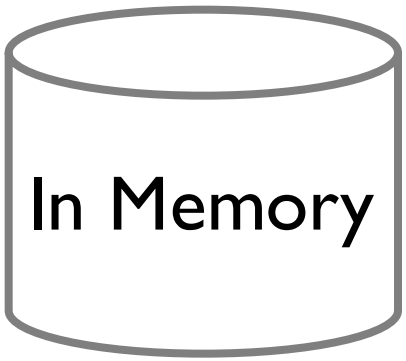
One Size Fits All

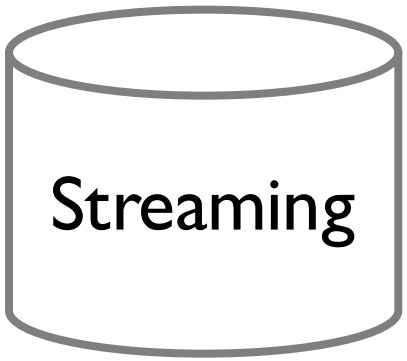


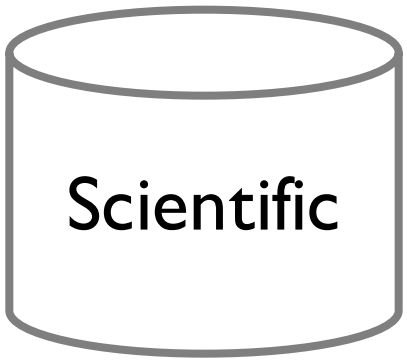
One Size Does Not Fits All



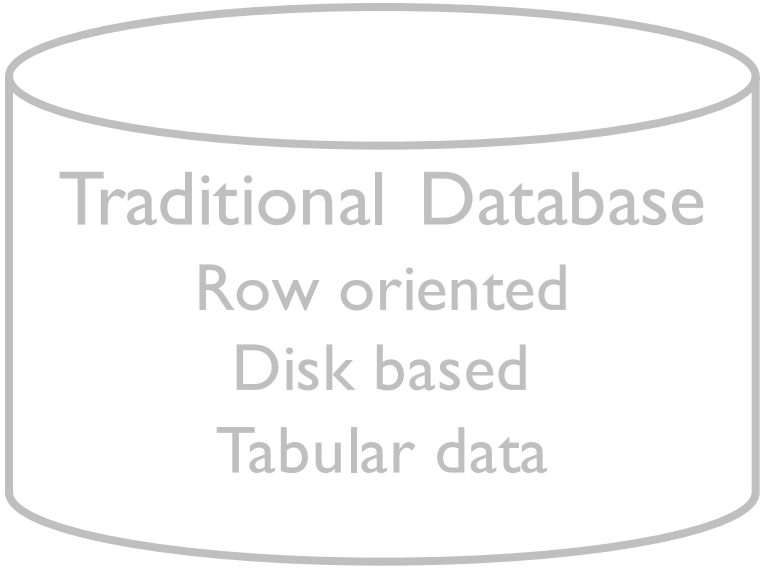


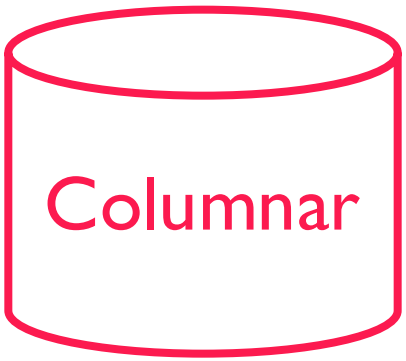


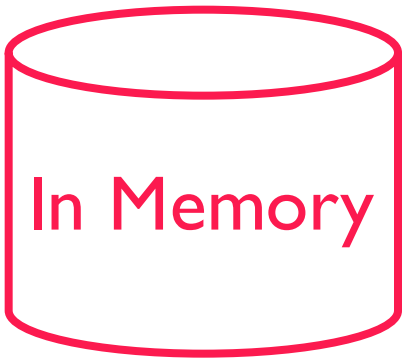


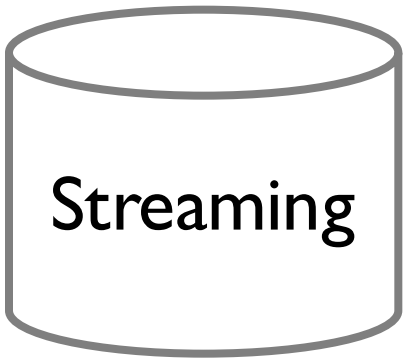


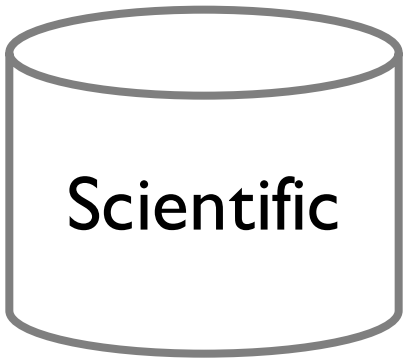
One Size Does Not Fits All











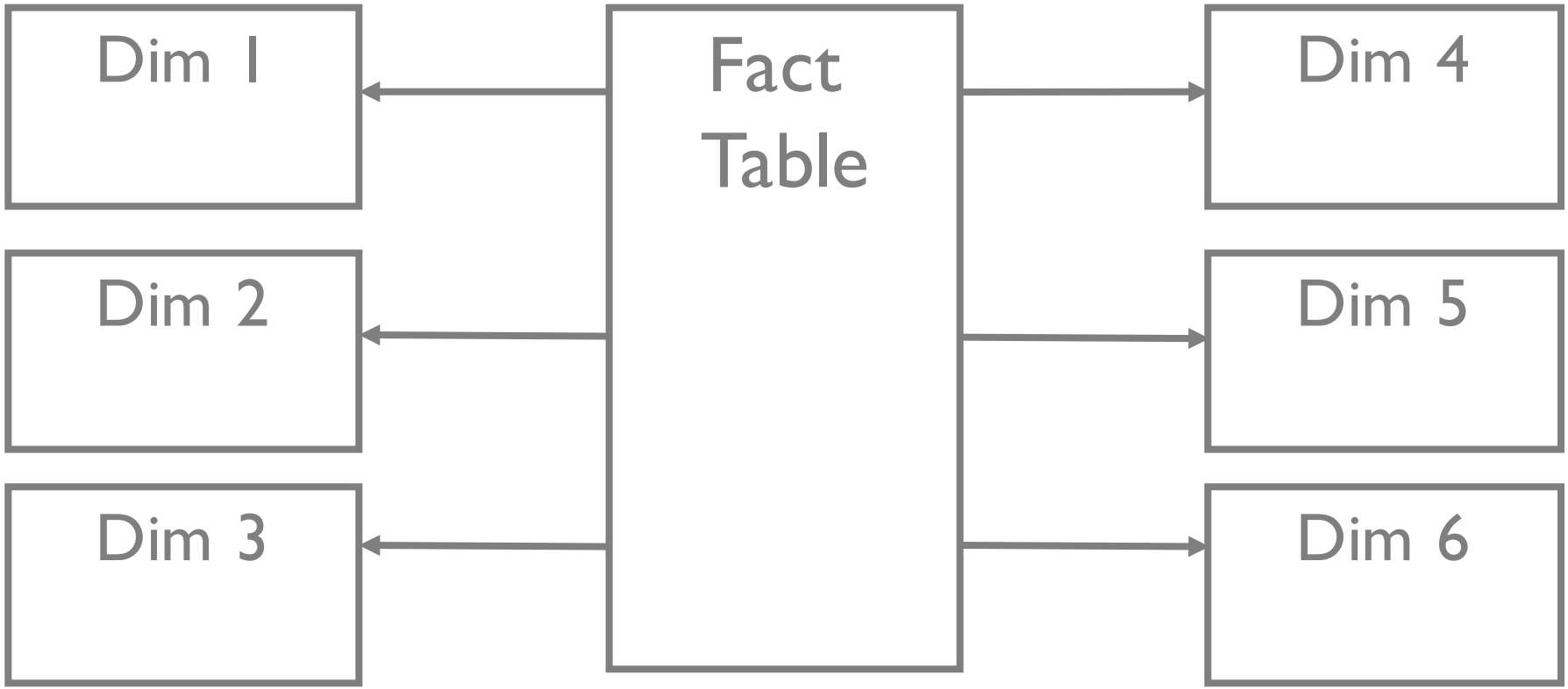
Data Warehouses

Store all historical data for future analysis

- Sales by month over past 20 years
- Clicks by youth in texas
- Cost by product component

Defacto standard at any company

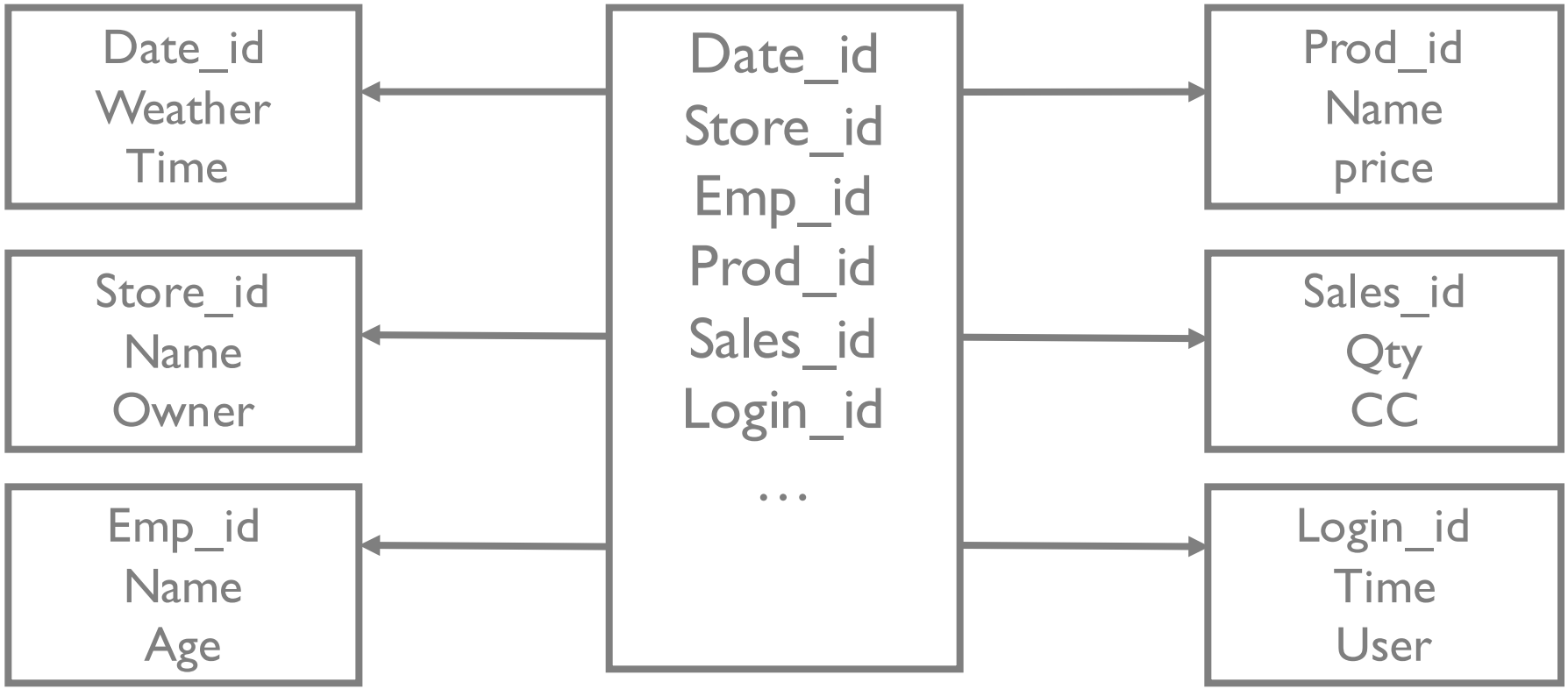
Star Schema

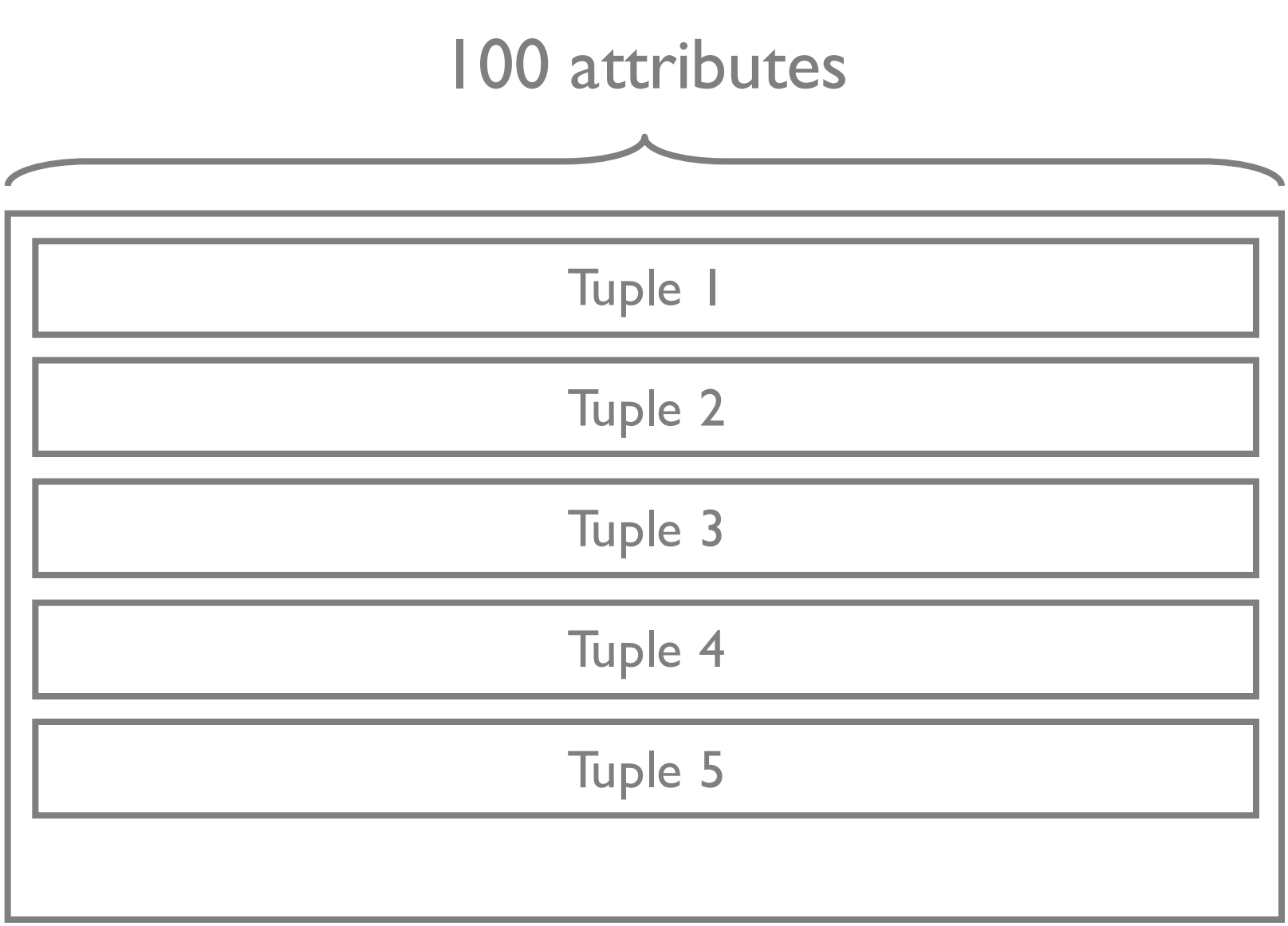


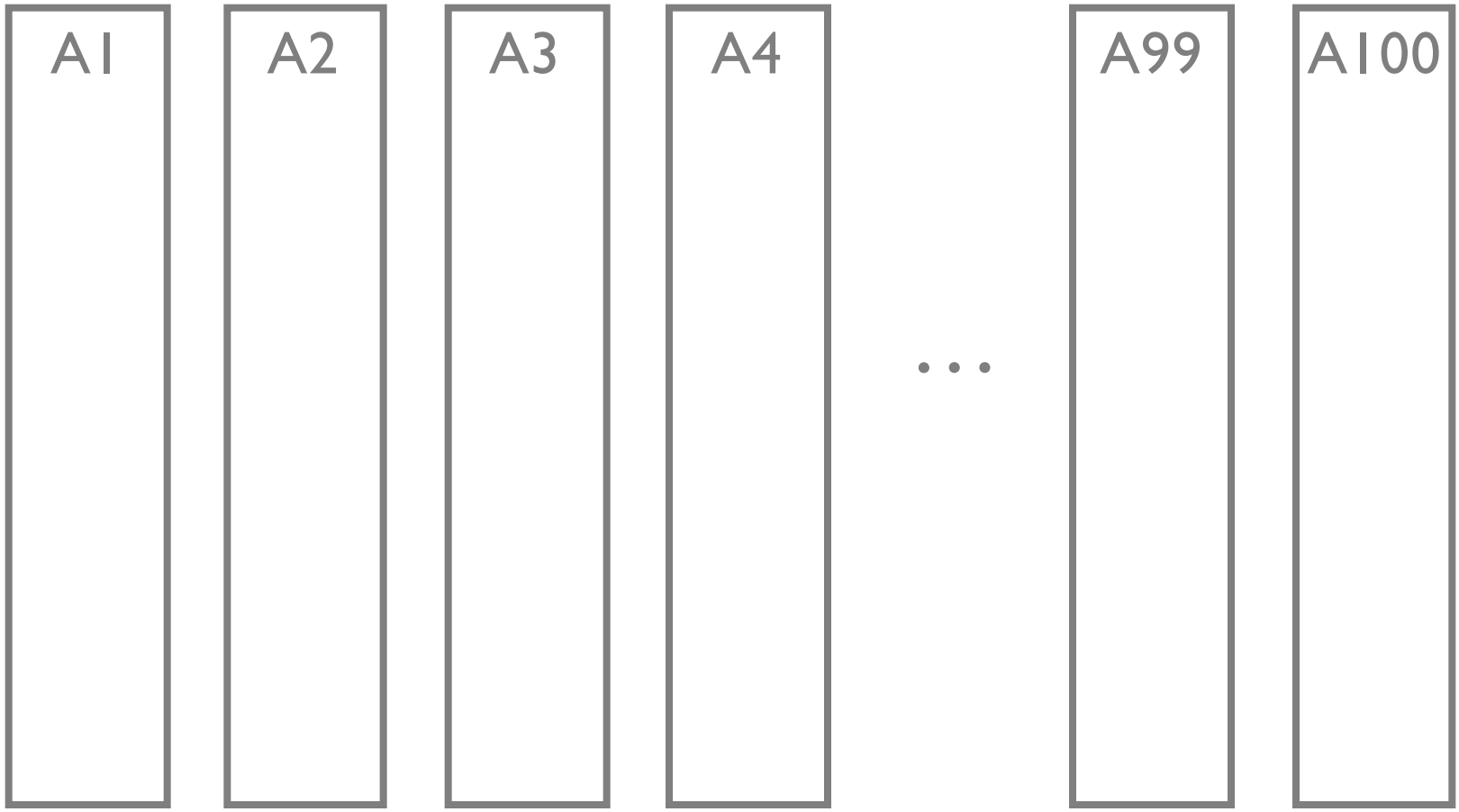
Star Schema

Fact table is “fat”

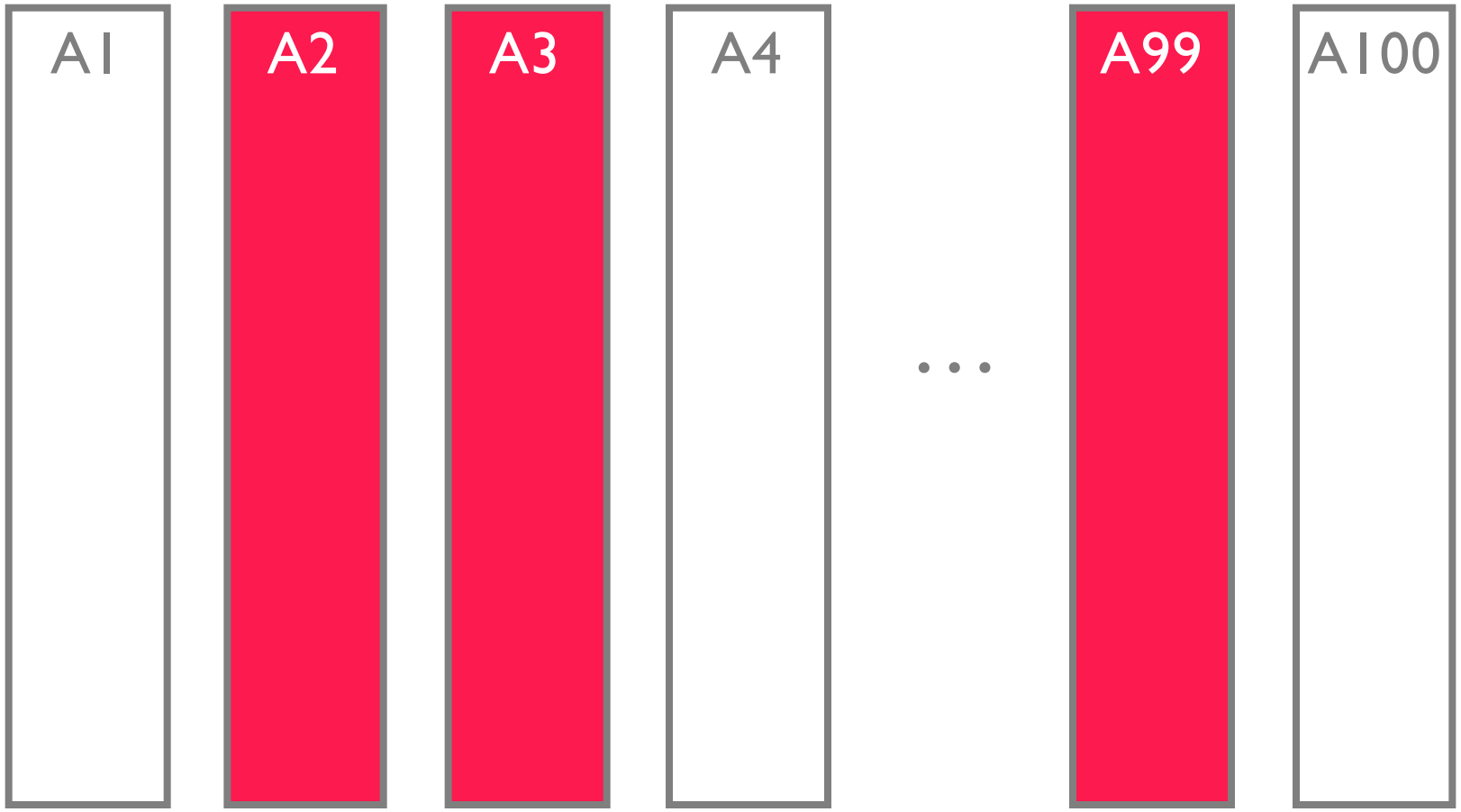
Queries access ~6 attrs




100 attributes
Tuple 1
Tuple 2
Tuple 3
Tuple 4
Tuple 5


A1
A2
A3
A4
A99
A100

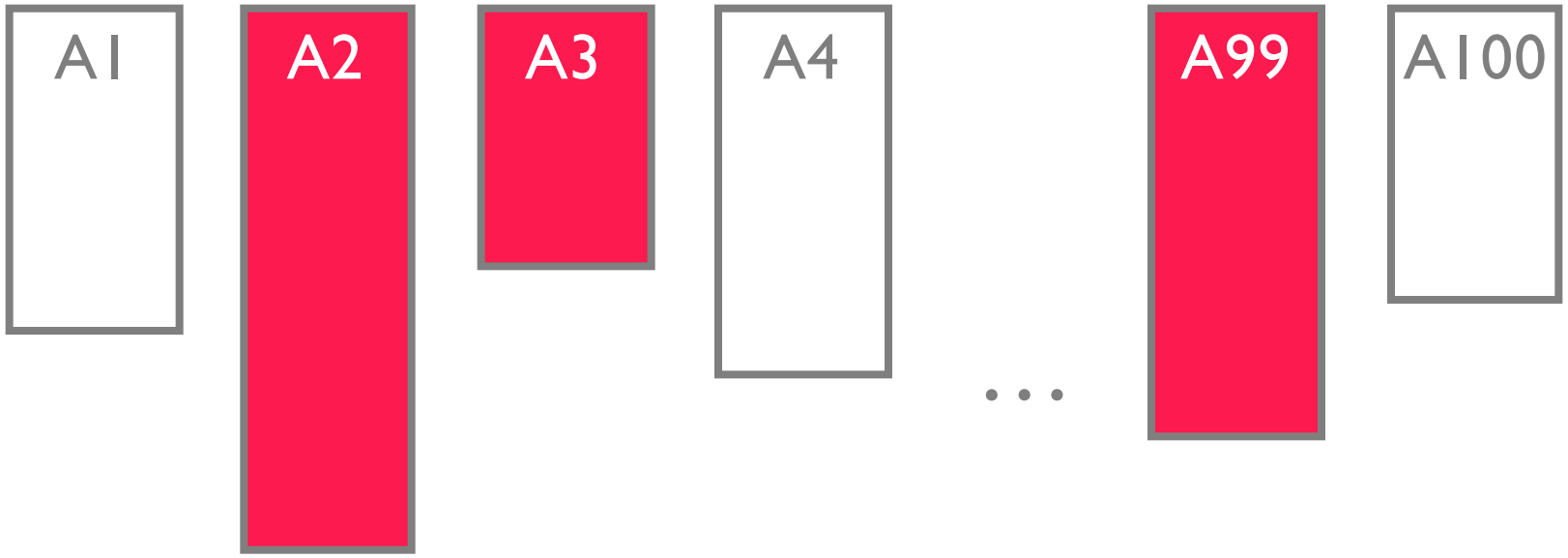
16x less data read. Unfair advantage.



16x less data read. Unfair advantage.

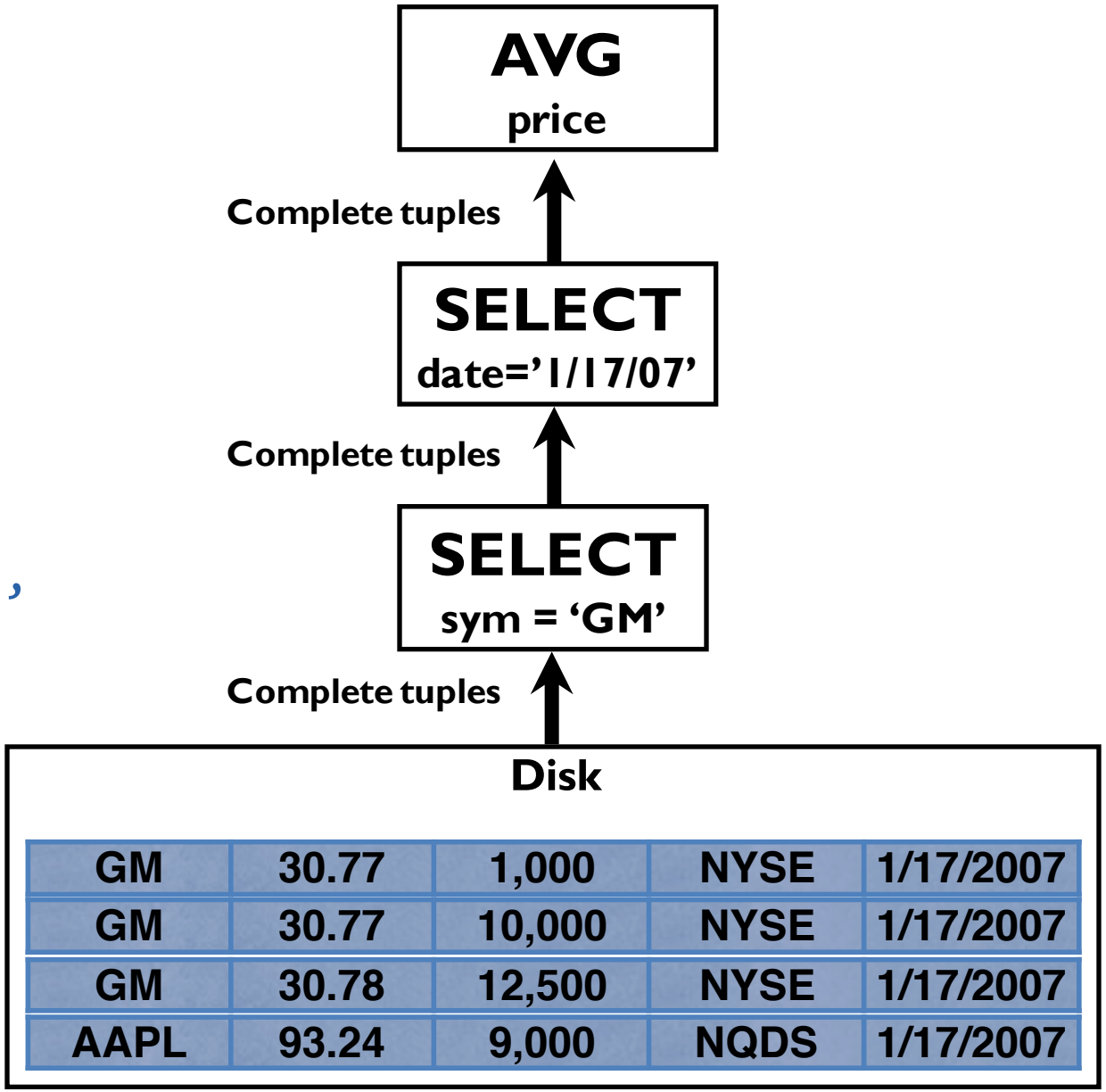
Compression better on single column

Execute on compressed data



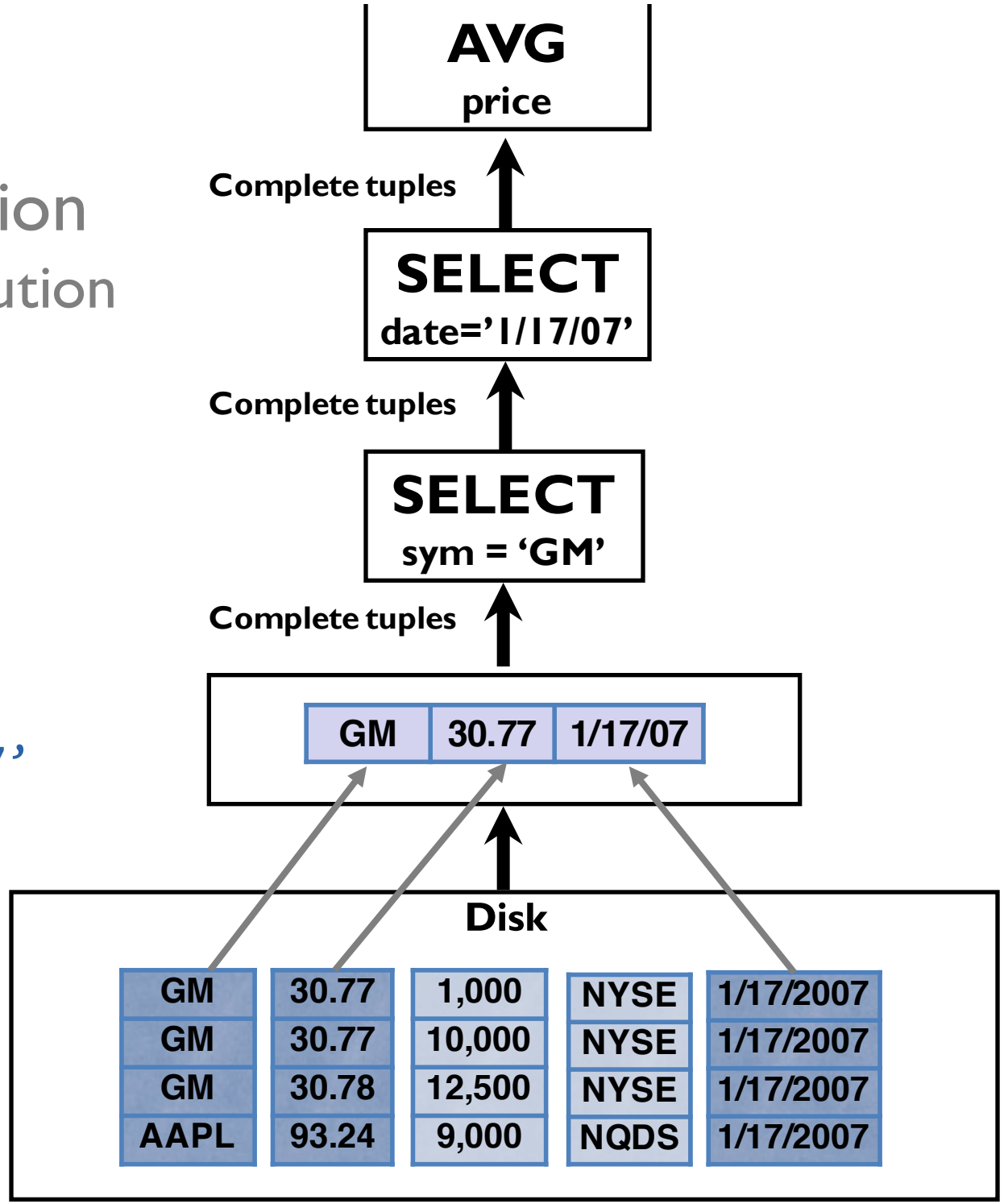
Traditional DBMS

```
SELECT avg(price)
FROM tickstore
WHERE symbol = 'GM'
AND date = '1/17/2007'
```



Naïve:
Early Materialization
Row oriented execution

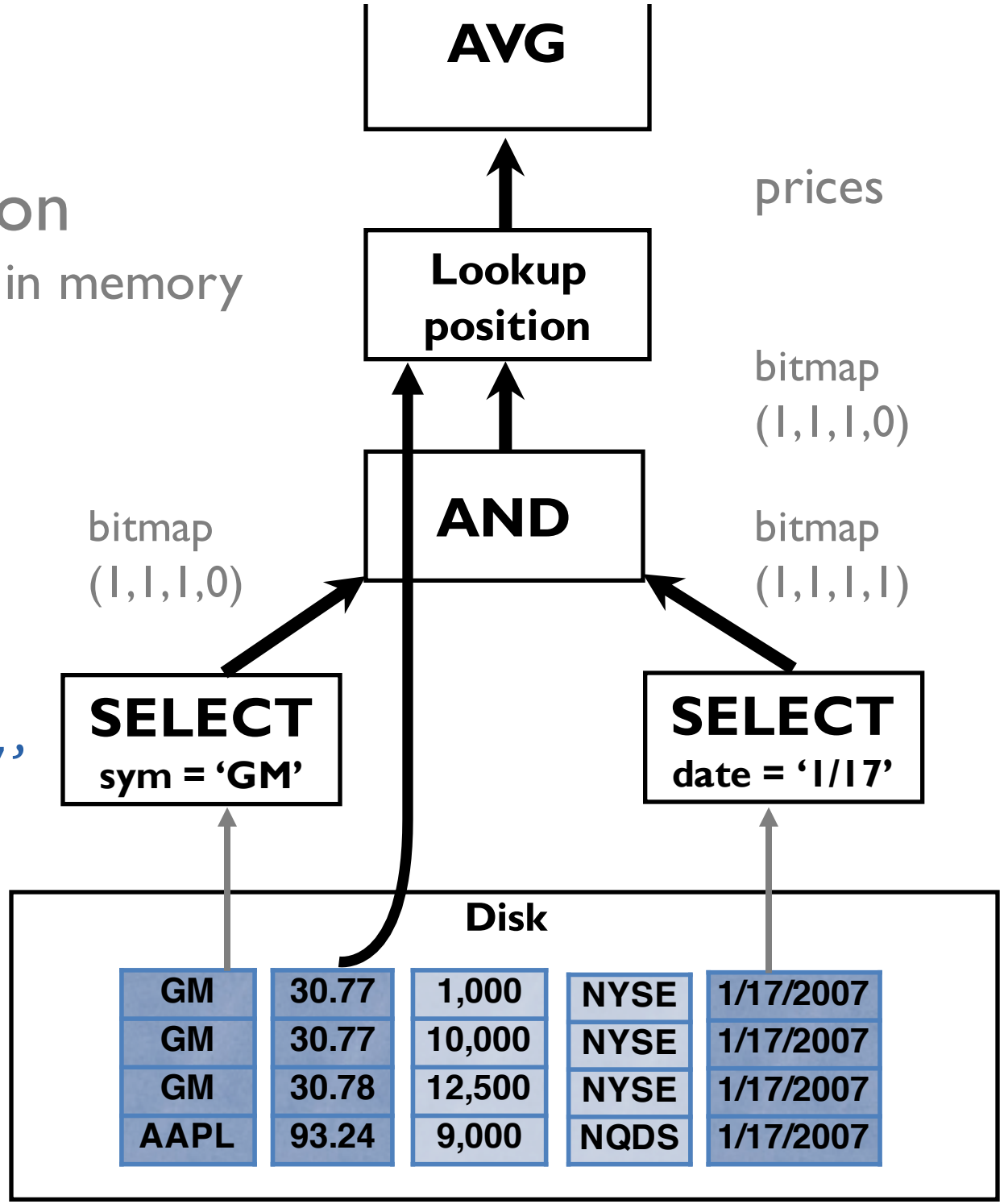
```
SELECT avg(price)
FROM tickstore
WHERE symbol = 'GM'
AND date = '1/17/2007'
```



C-Store
Late Materialization

Much less data moving in memory

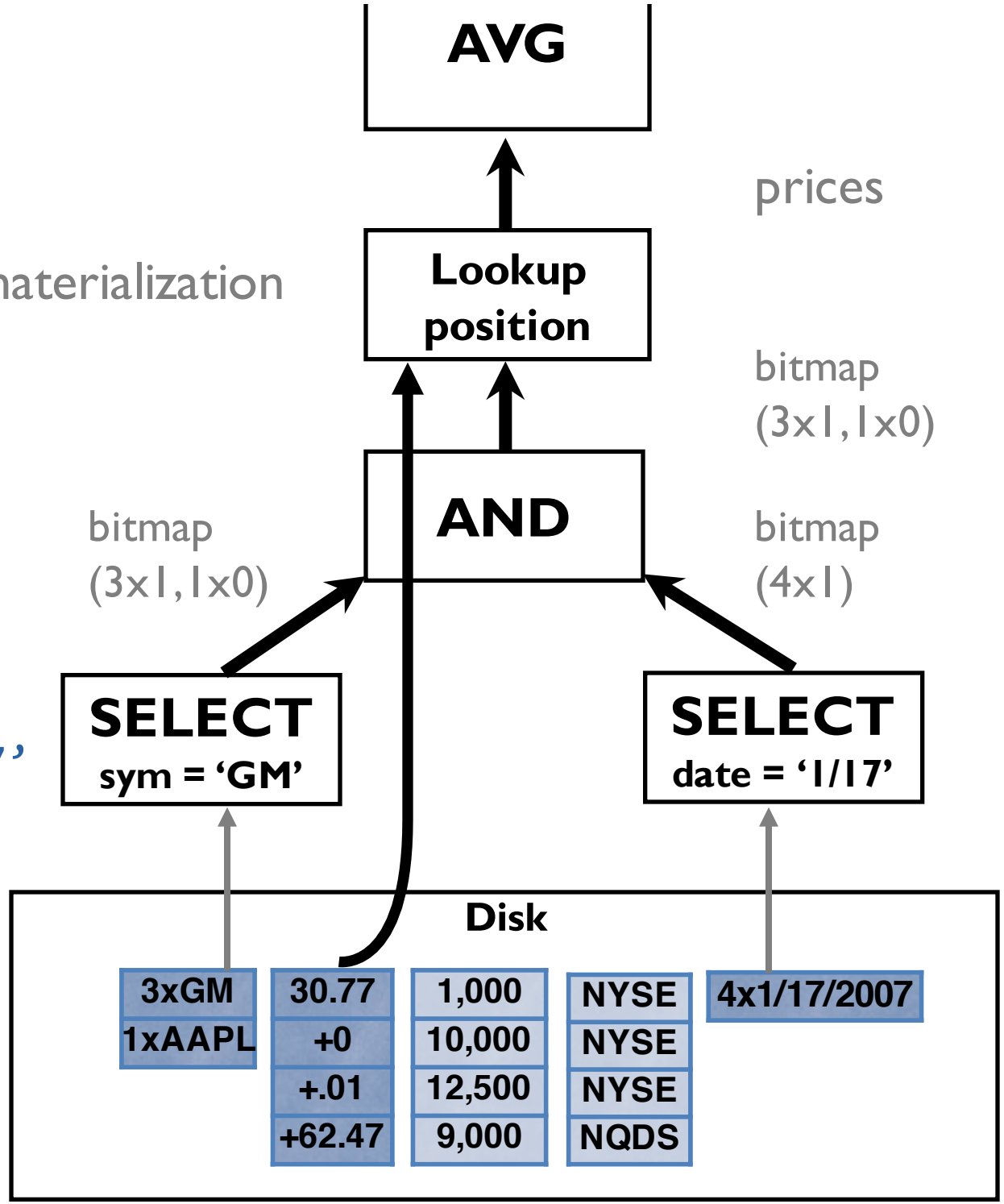
```
SELECT avg(price)
FROM tickstore
WHERE symbol = 'GM'
AND date = '1/17/2007'
```



C-Store Compression

Only possible w/ late materialization

```
SELECT avg(price)
FROM tickstore
WHERE symbol = 'GM'
AND date = '1/17/2007'
```



Column Stores

Optimized for data warehouses

Store data by attribute/column rather than row

Compression

Compressed *query plan execution*

50-100x faster than row store

In-Memory DBMSes

Transaction-oriented apps

- remove 1 unit from product
- move 5 units from org 1 to org 2
- (shopping carts, inventory)

Data stored in memory

- Disk only used for check pointing

No concurrency

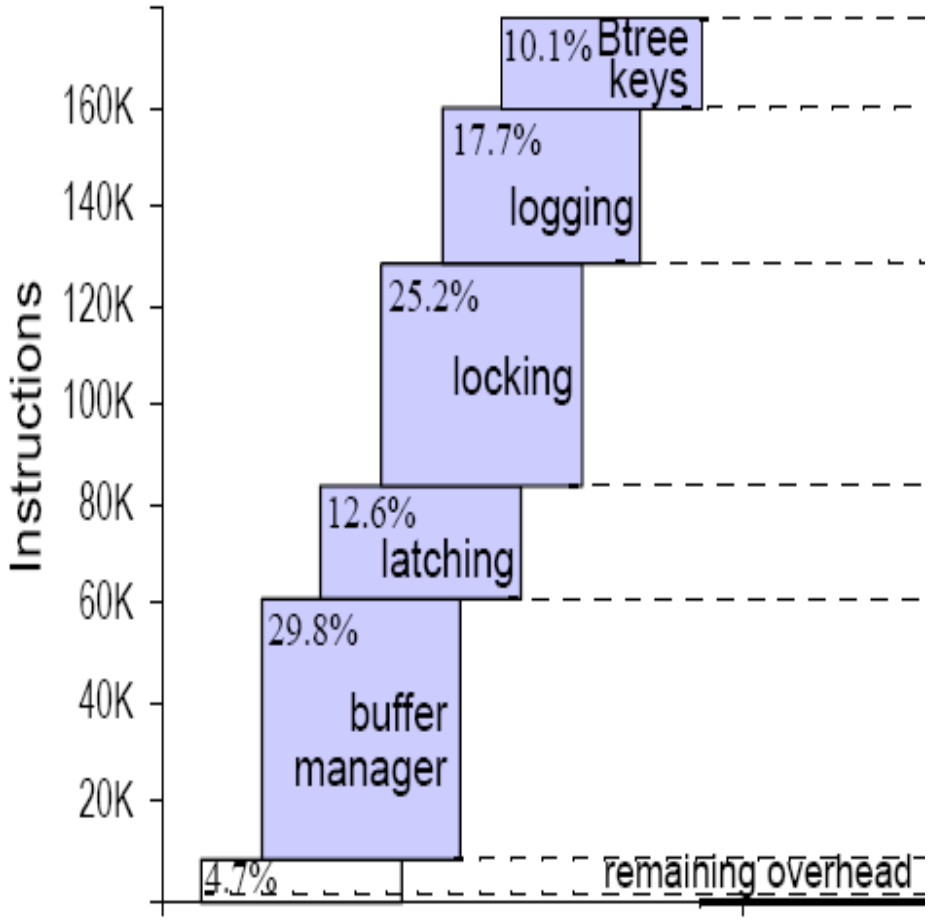
Active-active replication for fault-tolerance

Traditional Database

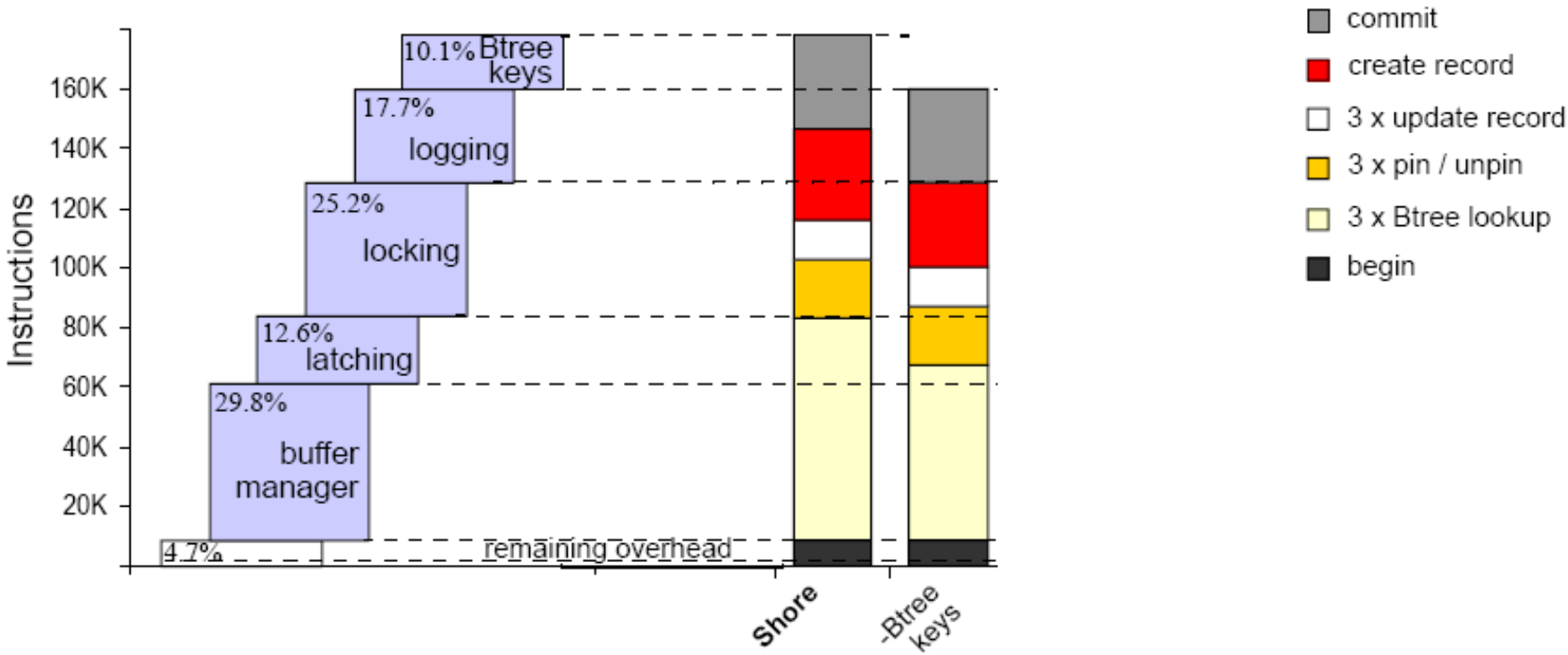
Indexes	queries go faster
Concurrency	queries go faster
Locking	serializability
Logging	recovery
Buffer Manager	manage pages in memory

Results after removing the components (in # instruction)

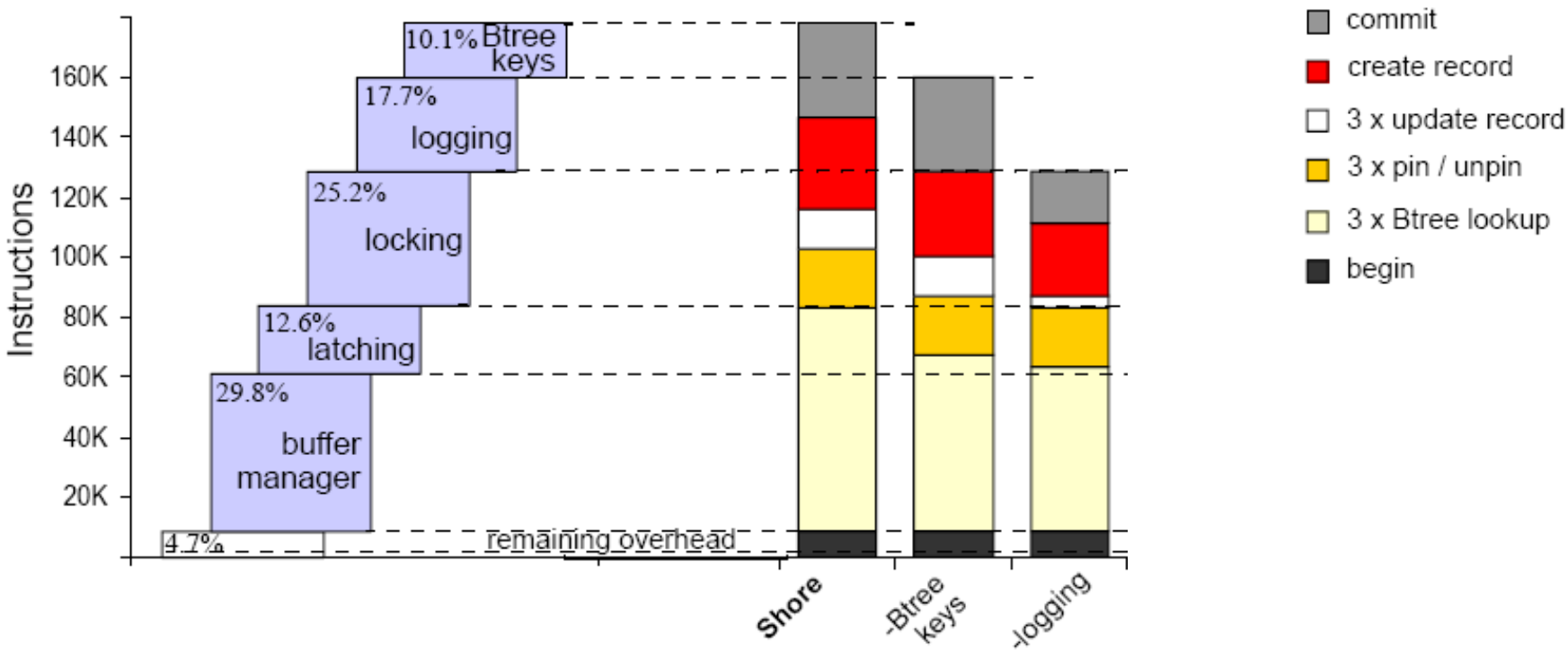
Instruction of useful work is only <2% of a memory resident DB



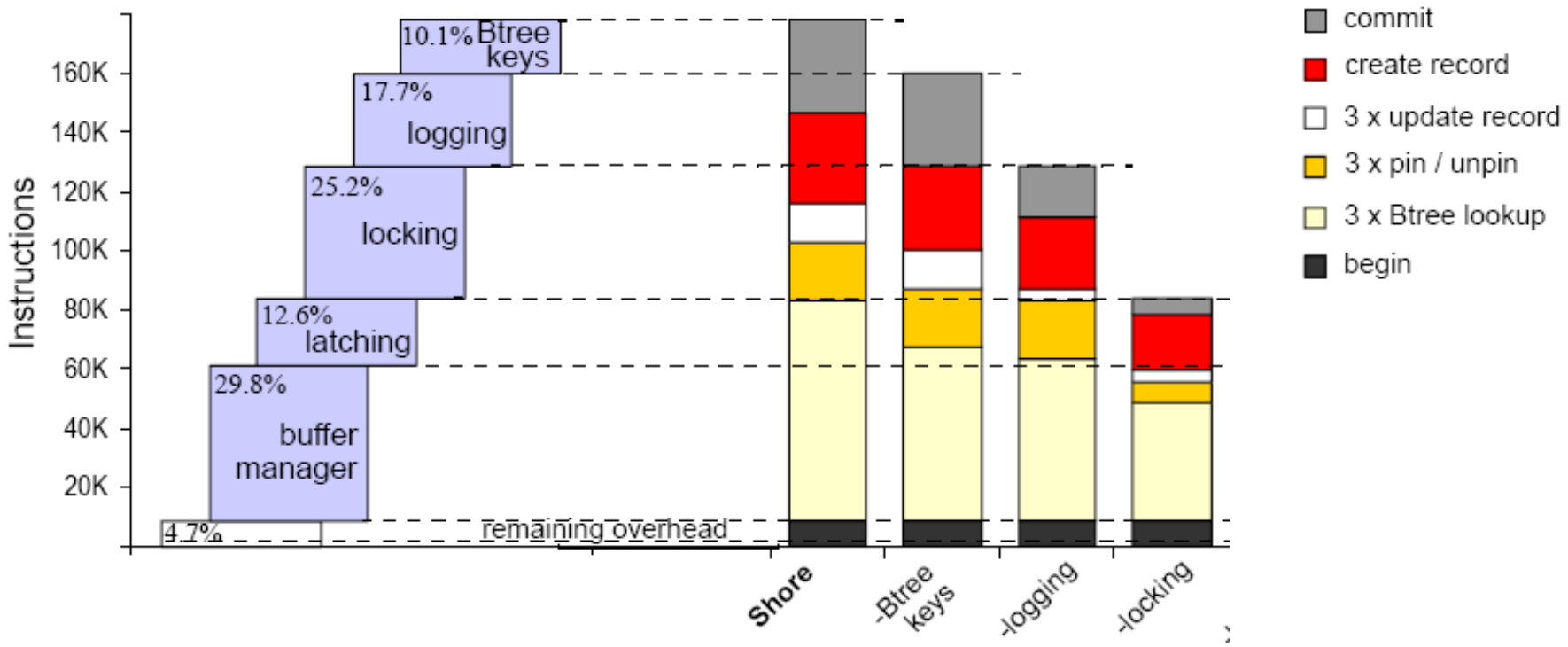
Effect of removing different components for payment (1/6)



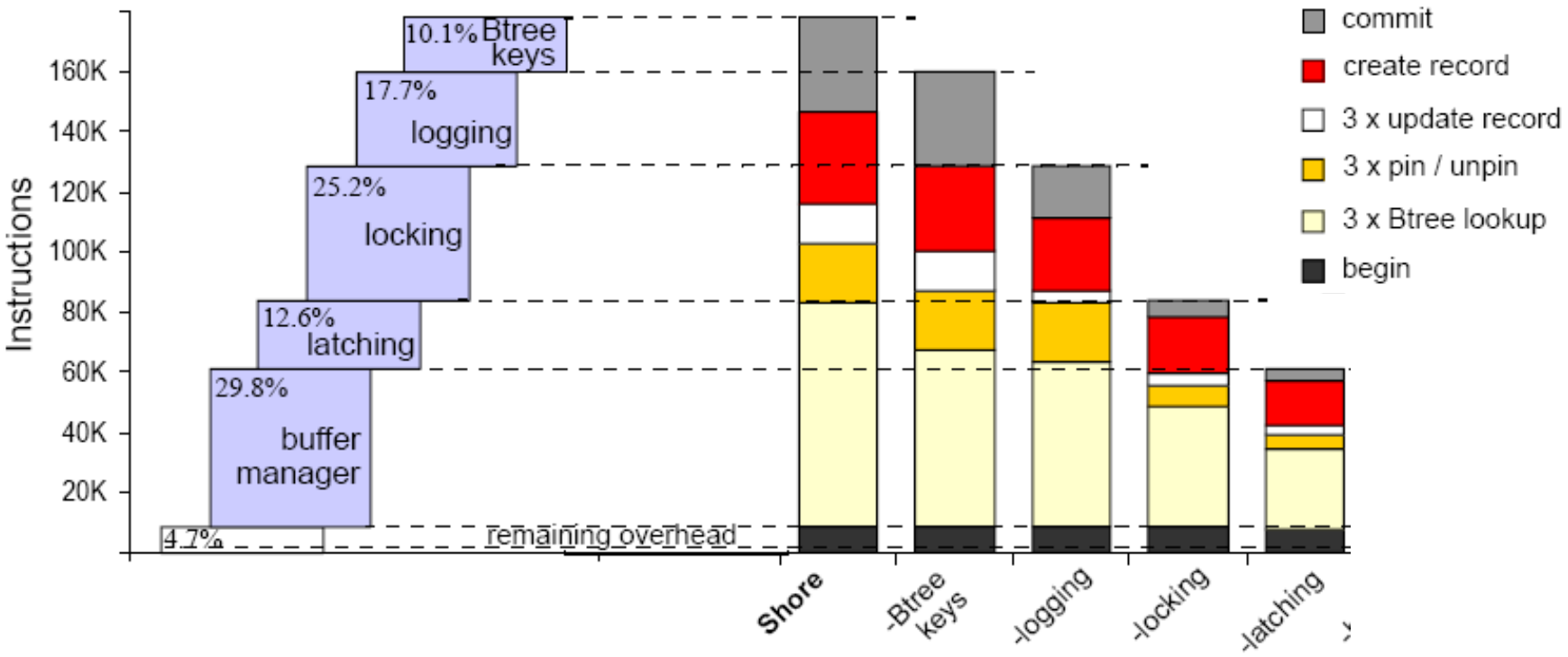
Effect of removing different components for payment (2/6)



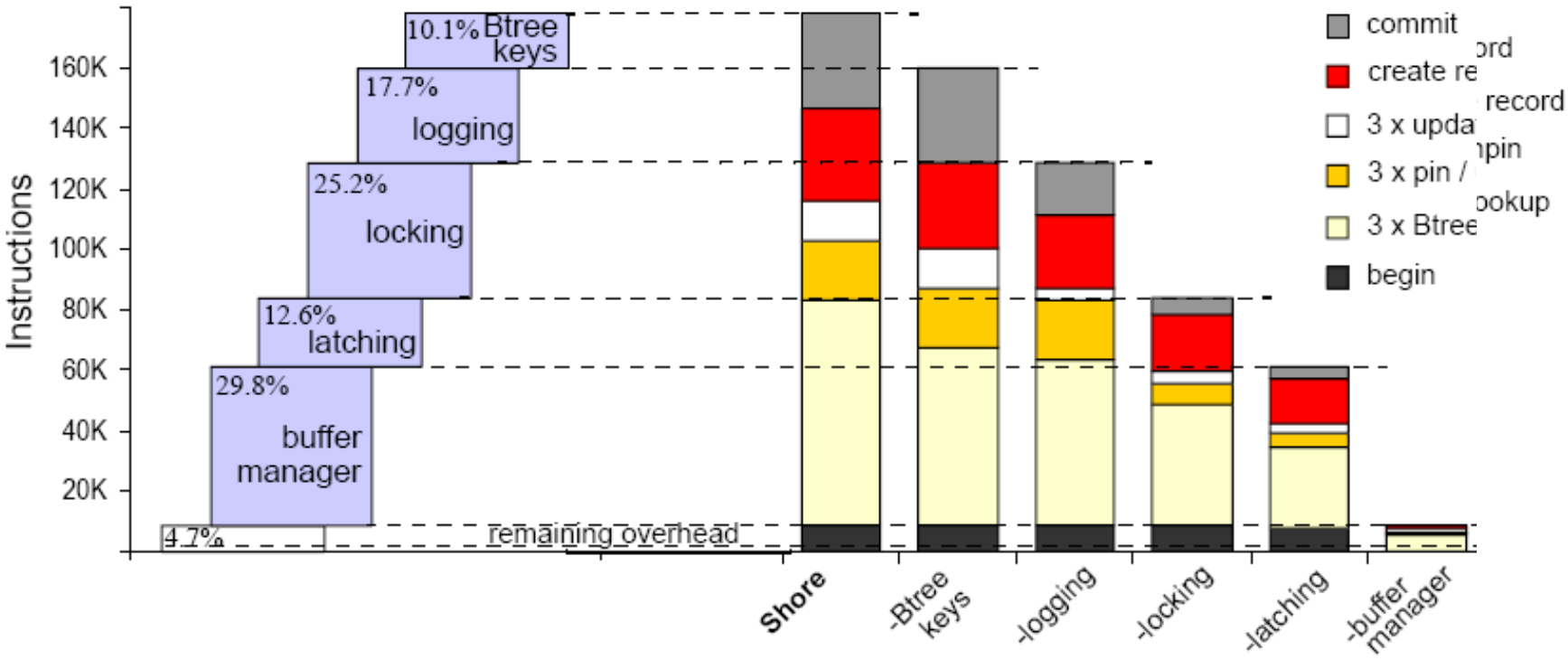
Effect of removing different components for payment (3/6)



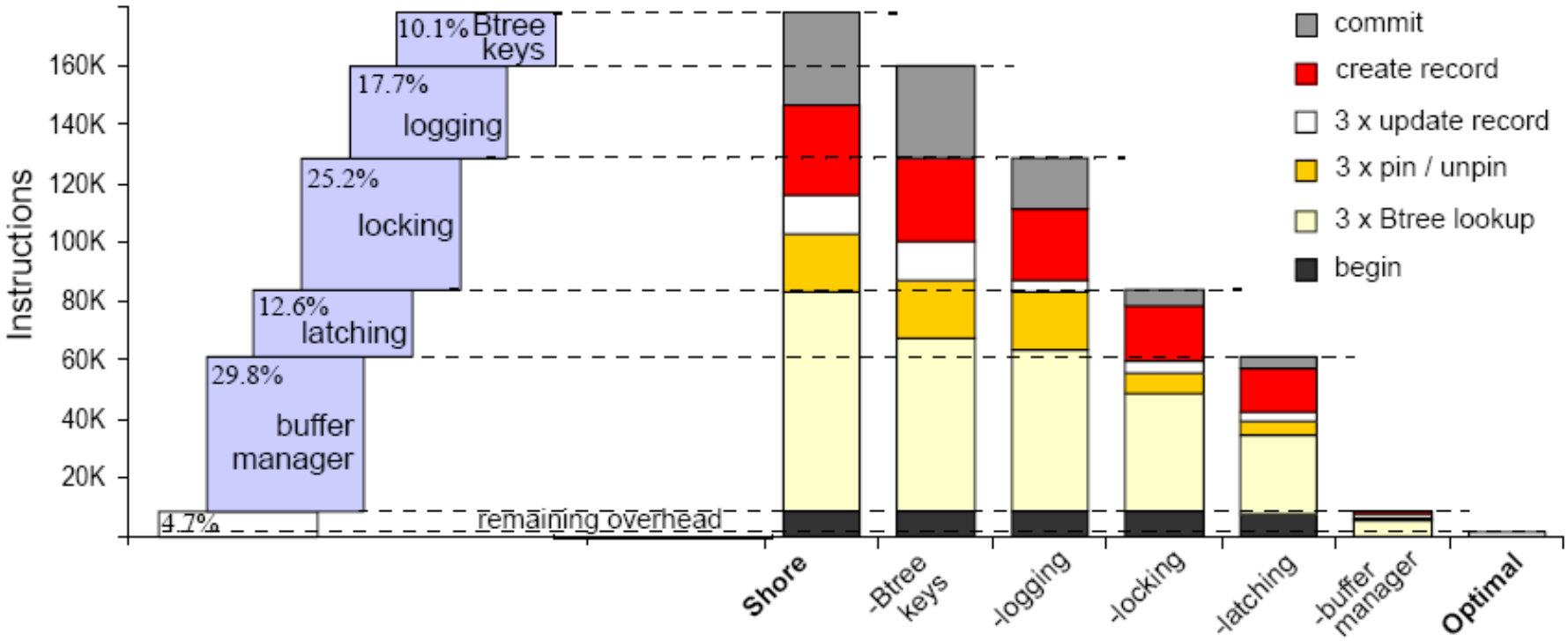
Effect of removing different components for payment (4/6)

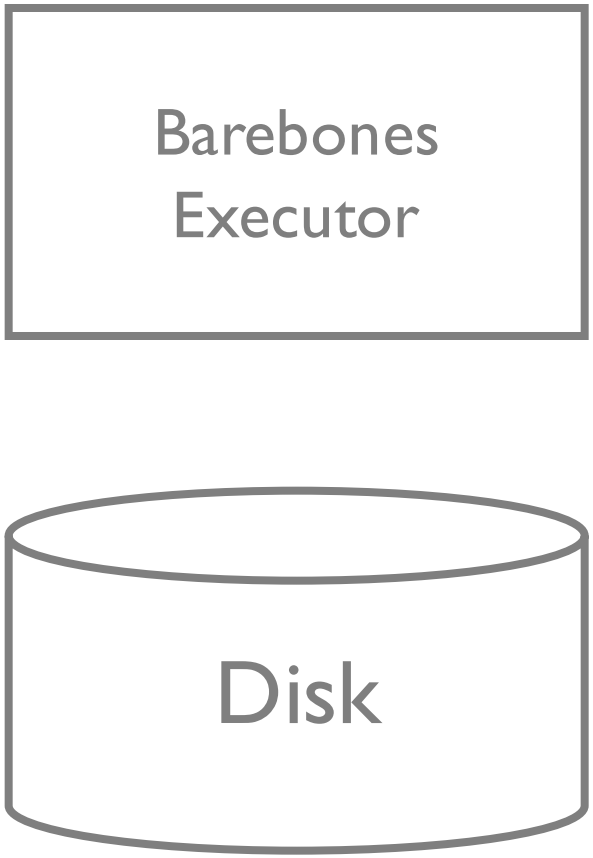


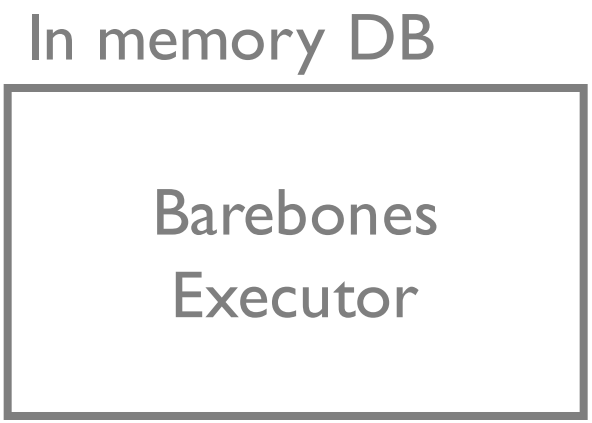
Effect of removing different components for payment (5/6)



Effect of removing different components for payment (6/6)

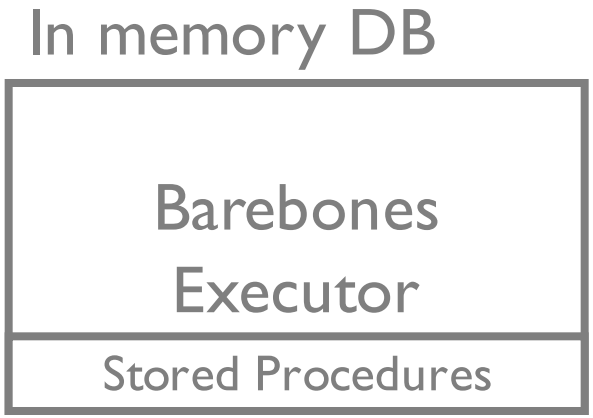



Barebones
Executor
Disk



What about

- Parsing
- Concurrency?
- Recovery?



What about

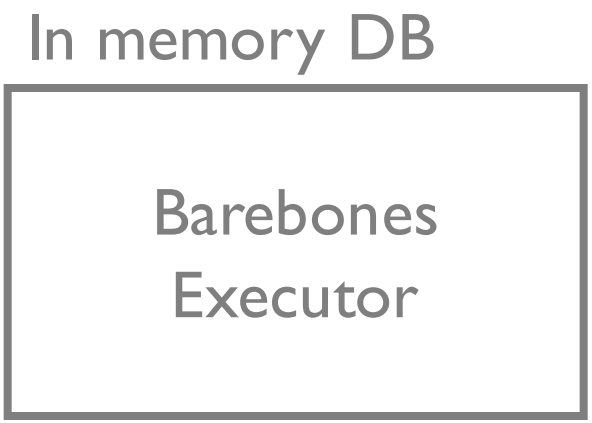
- Parsing
- Concurrency?
- Recovery?

Procedure:

```
p1 =  SELECT cost
      FROM fact_table
      WHERE id = ?
```

Query:

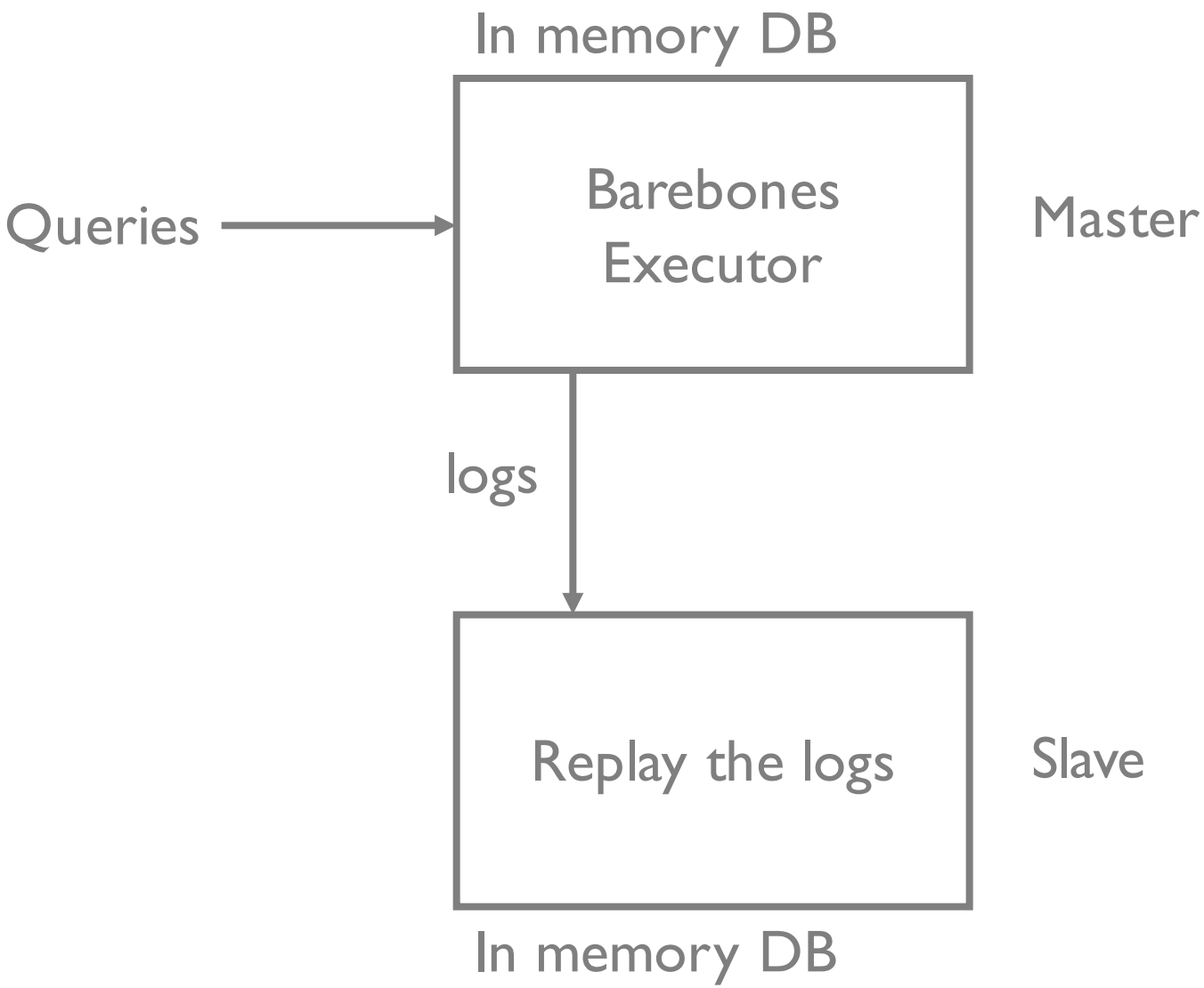
```
p1(10)
```



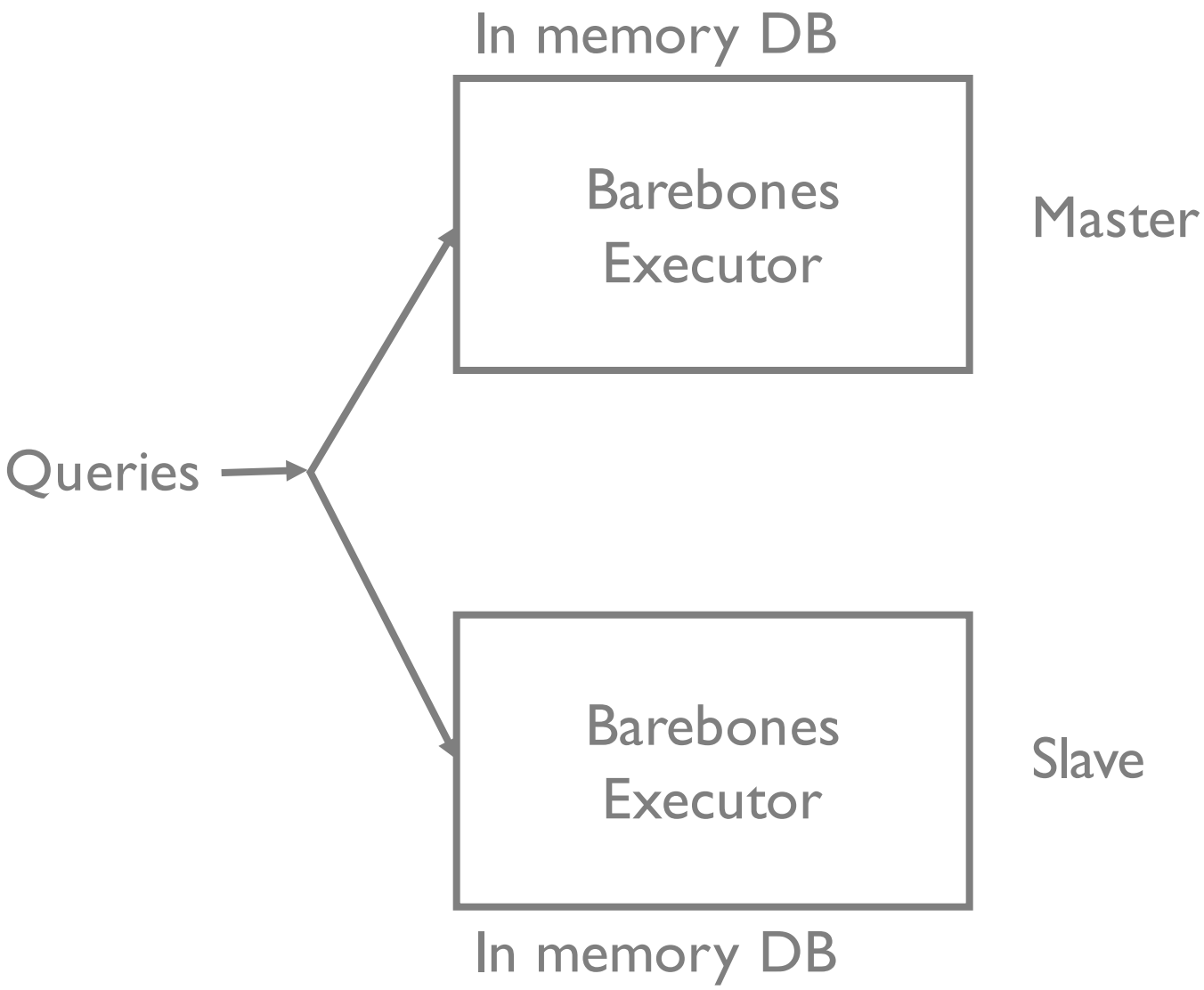
What about

- Parsing
- ~~Concurrency?~~
  - no buffer manager, no concurrency, no locks
- Recovery?

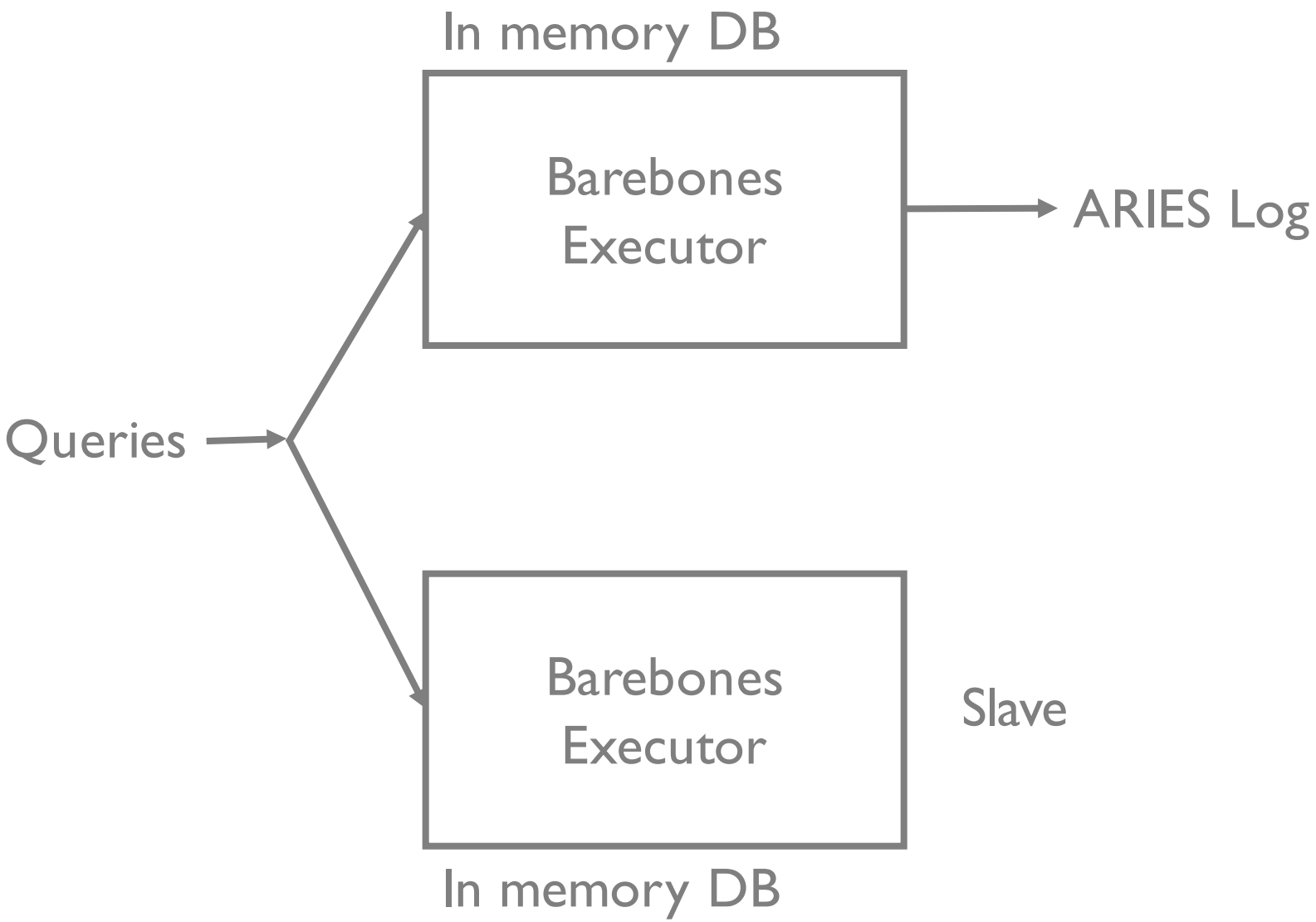
Log Shipping



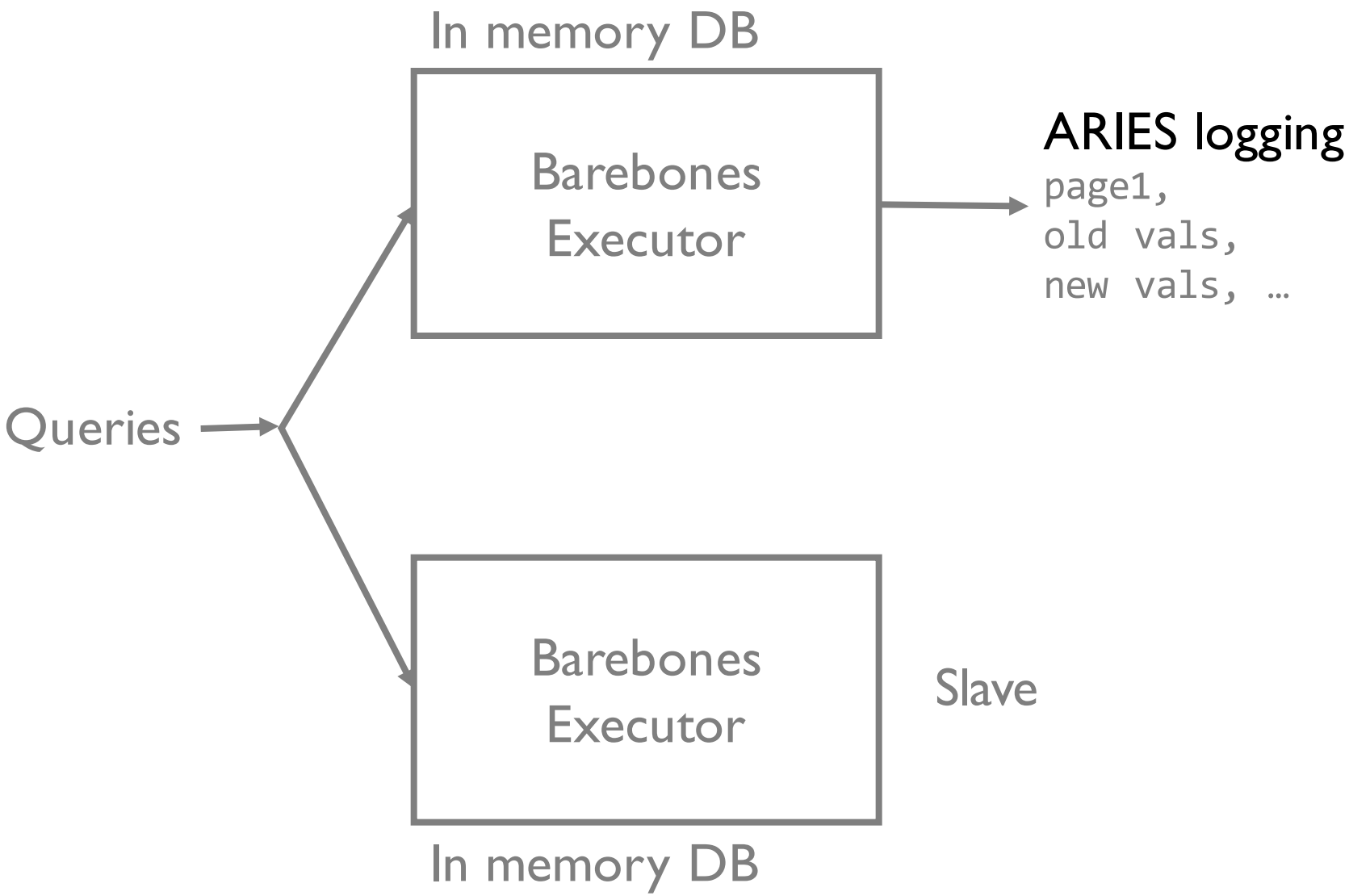
Active-Active



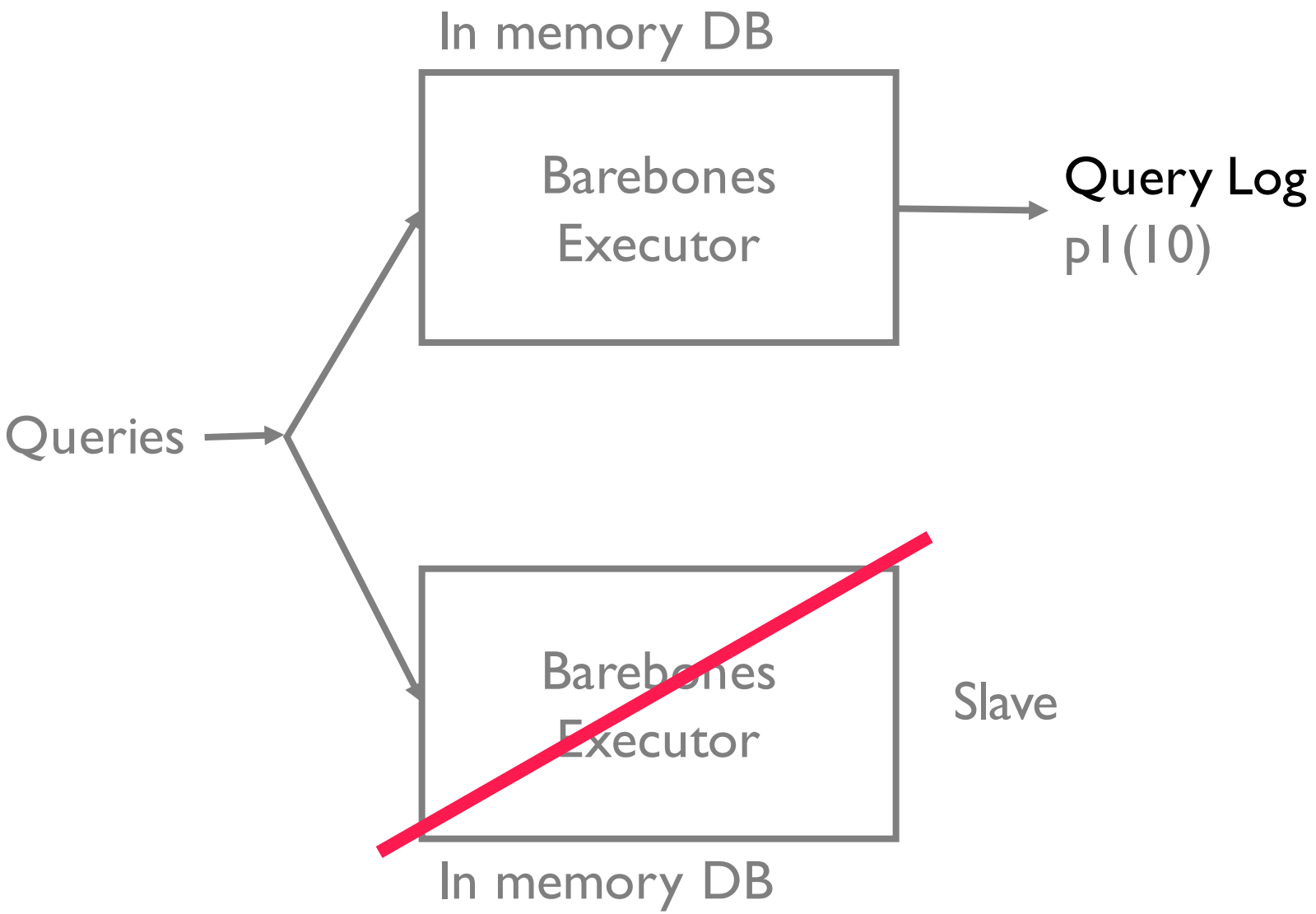
Recovery in Active-Active



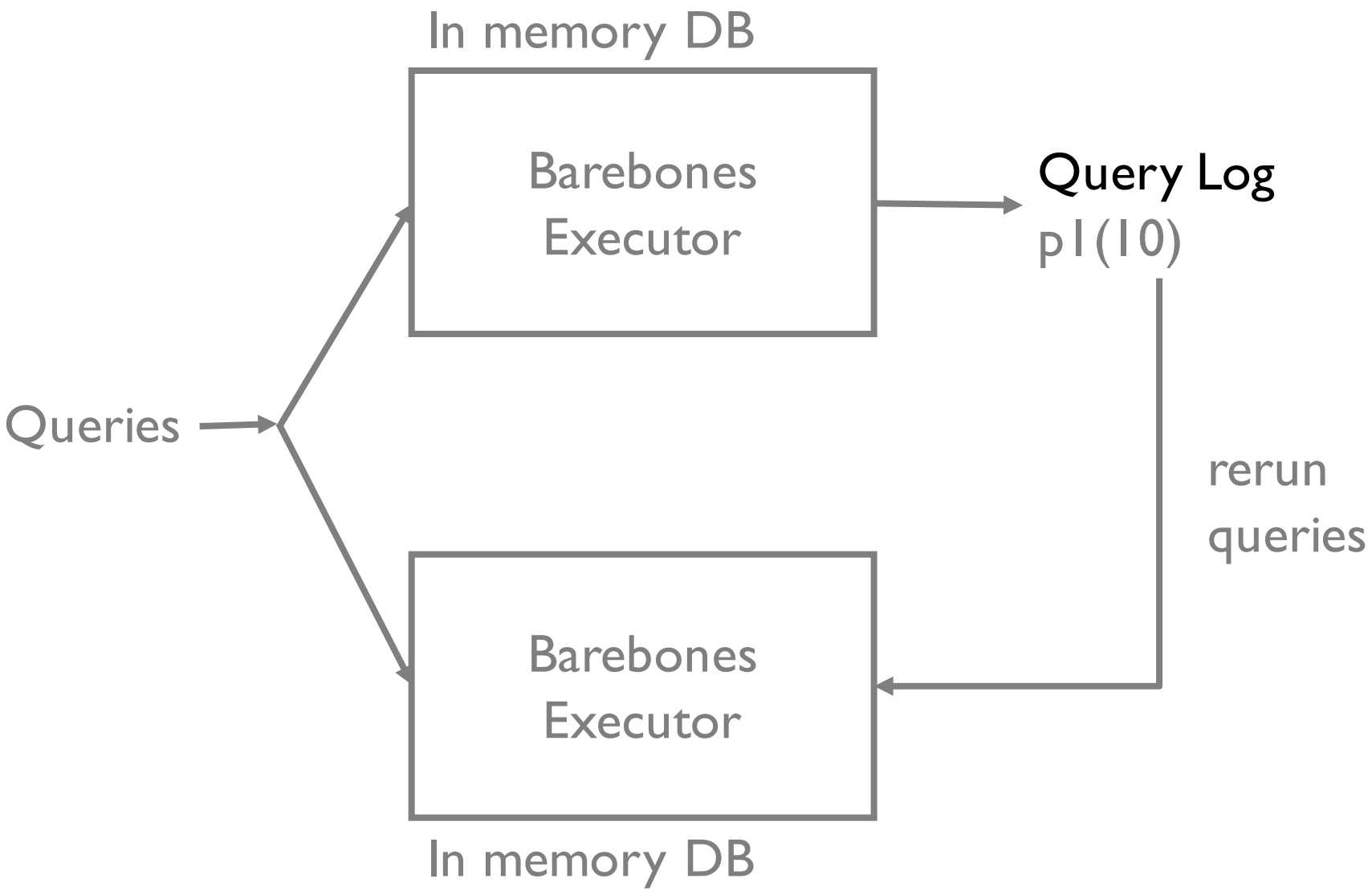
Recovery in Active-Active



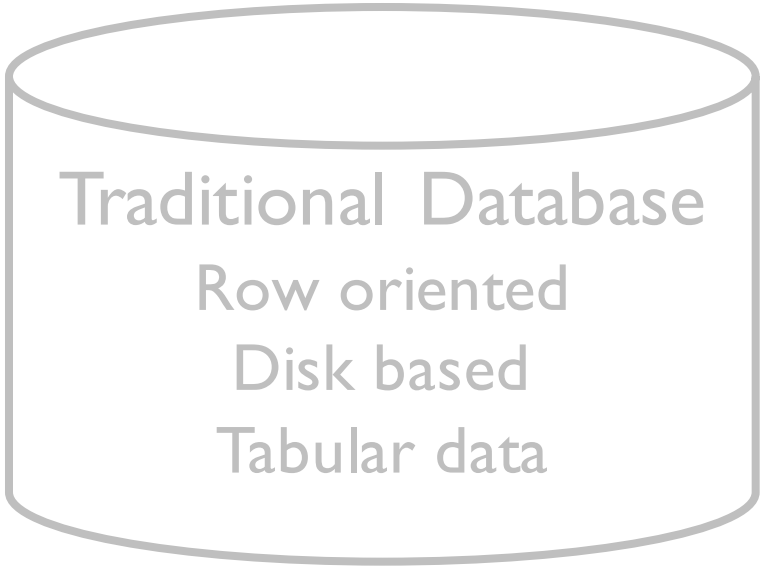
Recovery in Active-Active

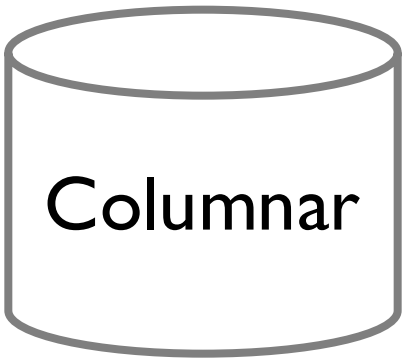


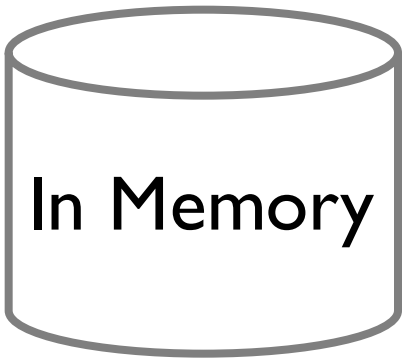
Recovery in Active-Active

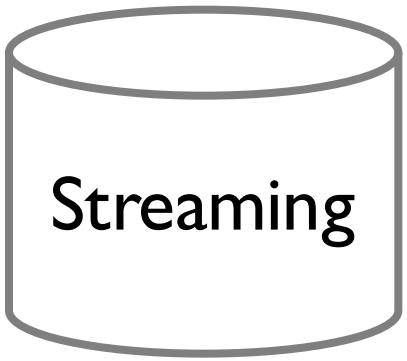


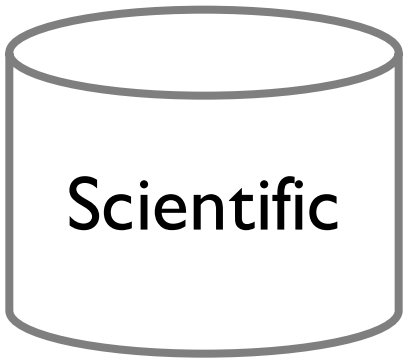
One Size Does Not Fits All



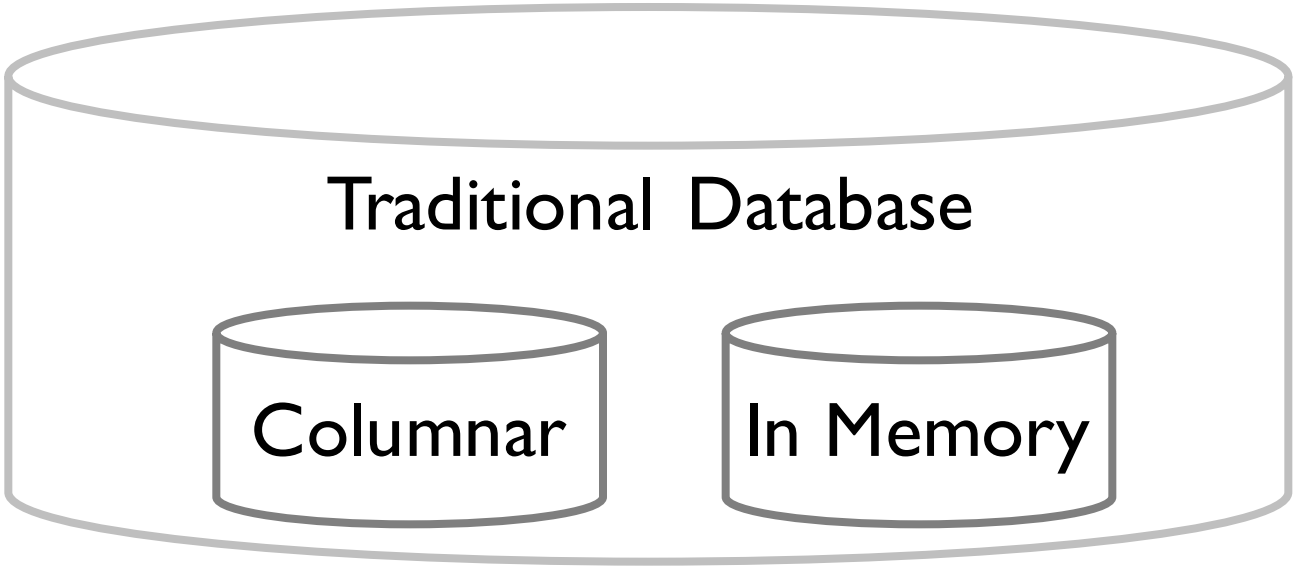




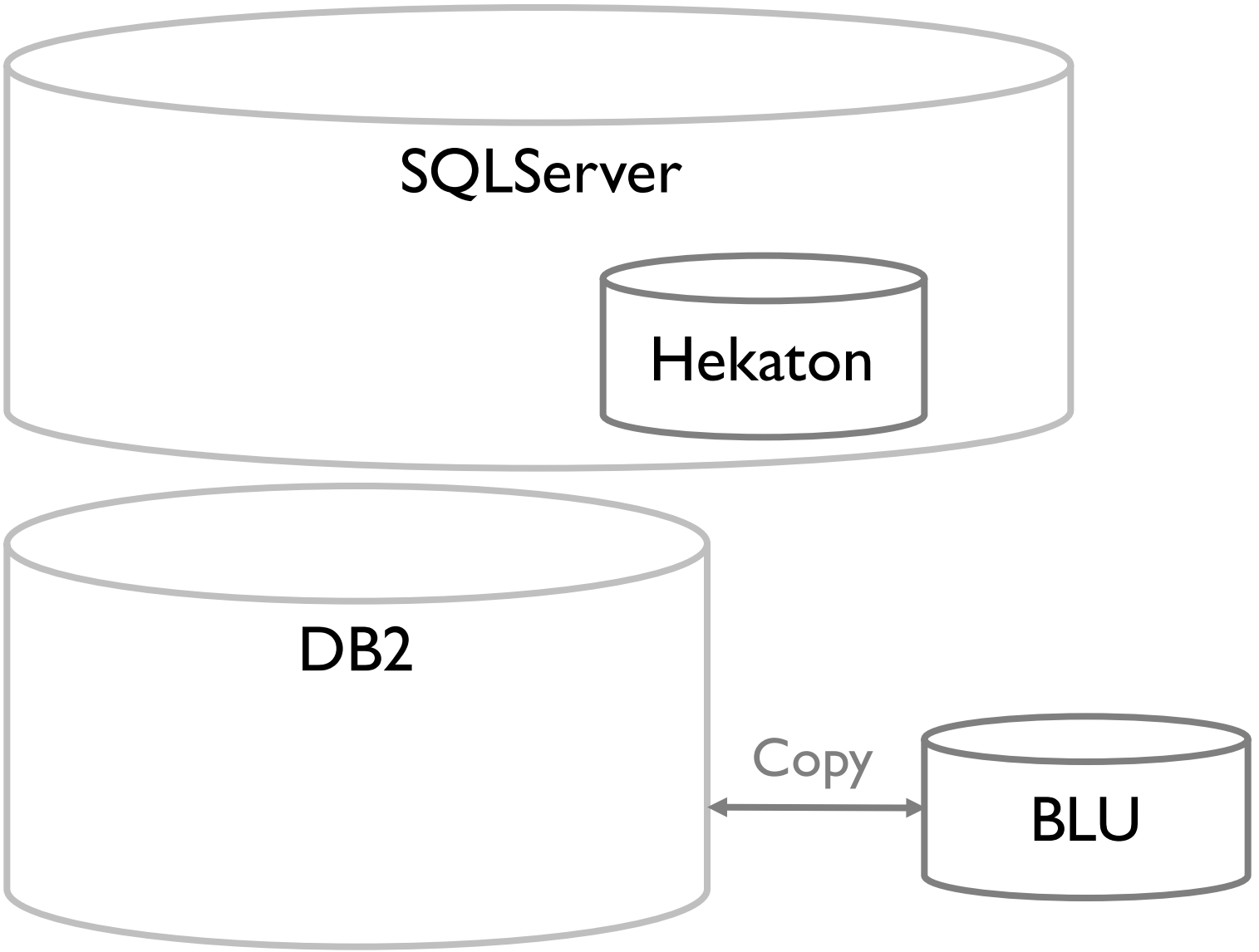




One Size Does Not Fits All



One Size Does Not Fits All



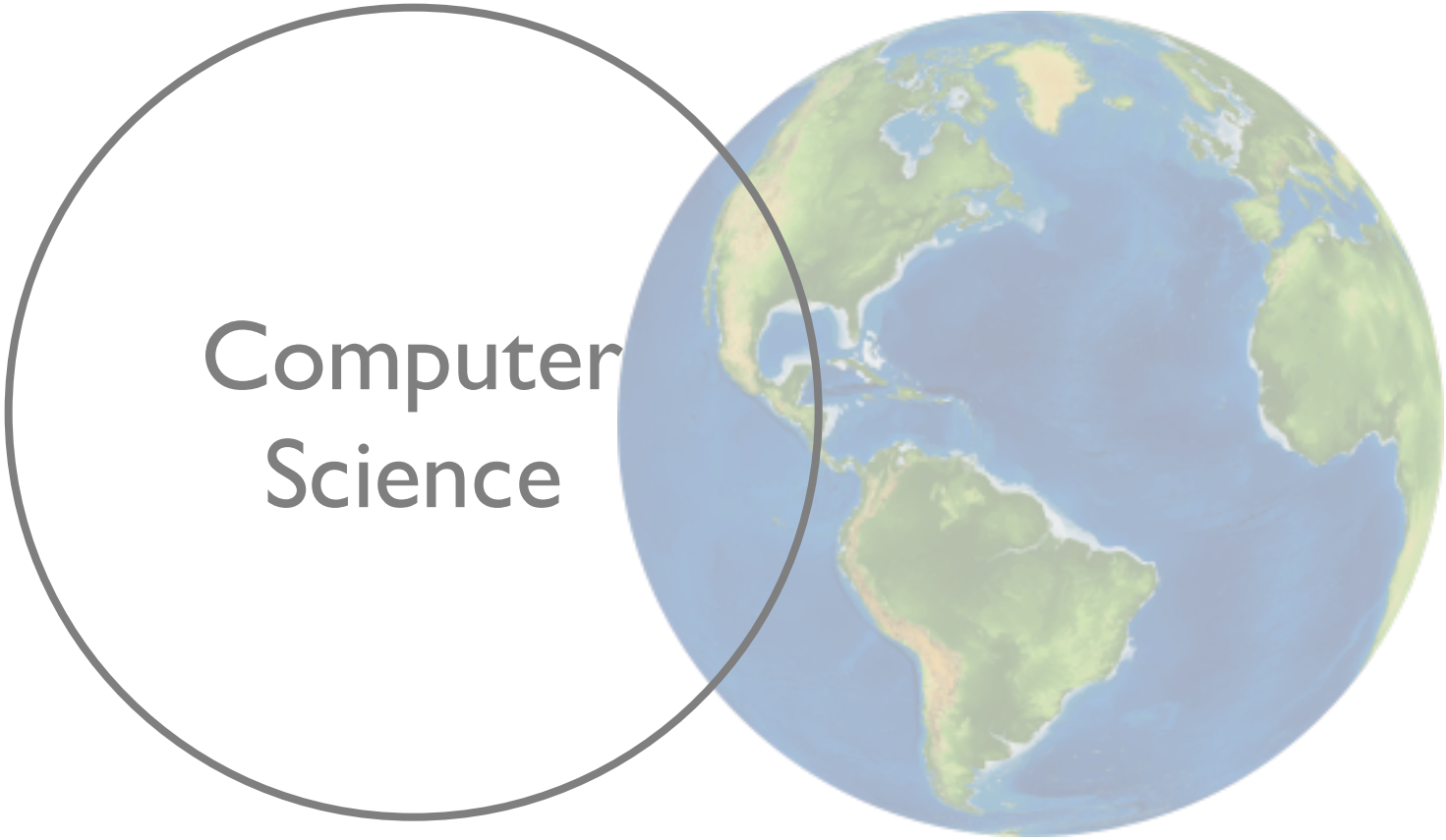
OK Let's Step Back

Discussed the *how*:

- how to model data needed by an application
- how to query databases
- how databases execute queries
- how databases run fast, correctly

To what end?

Computer Science


Computer
Science


Computer
Science

Data is crucial to

our lives as individuals and as a society

Thanks!

Please fill out courseworks survey

Role of 4111 in DSI curriculum