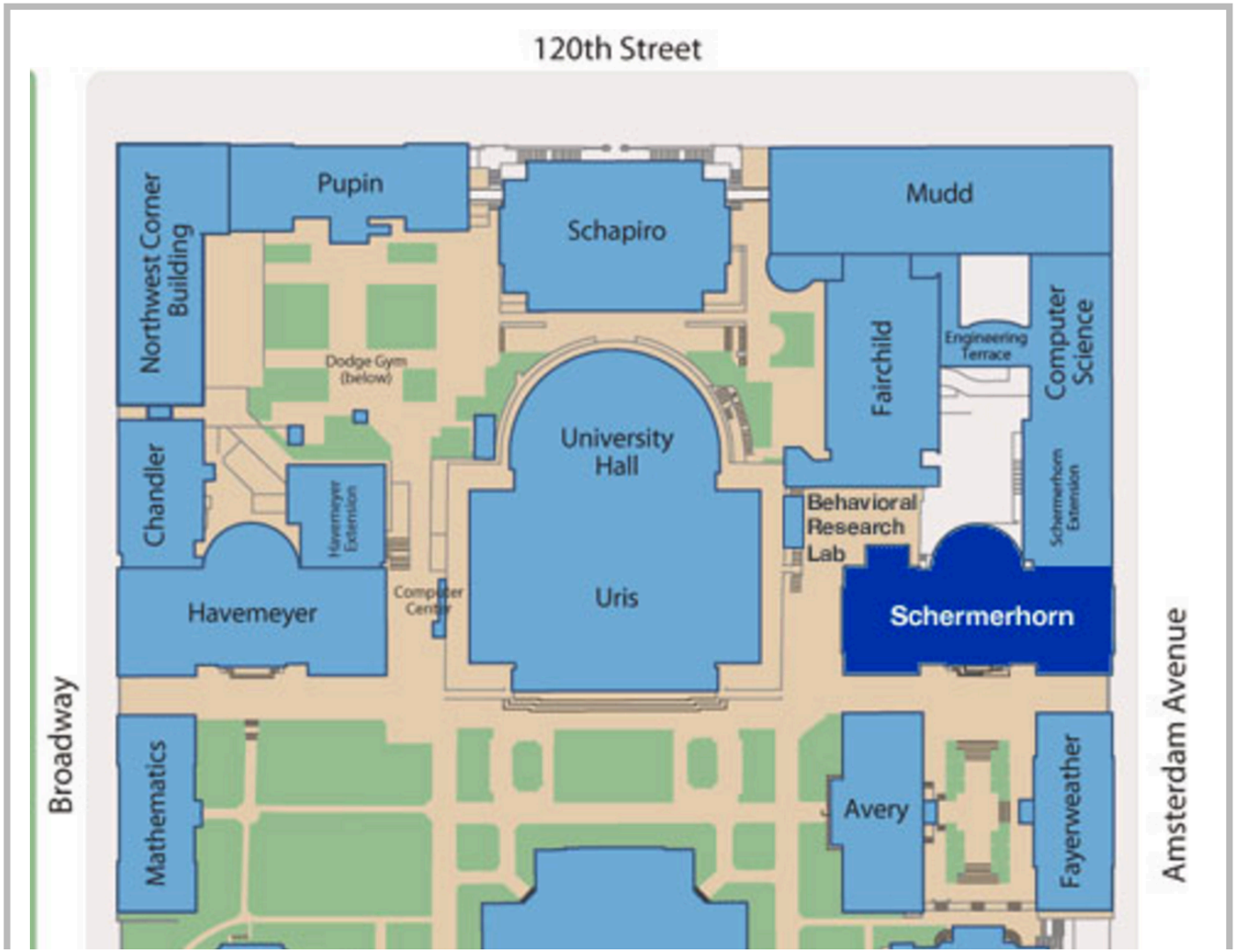
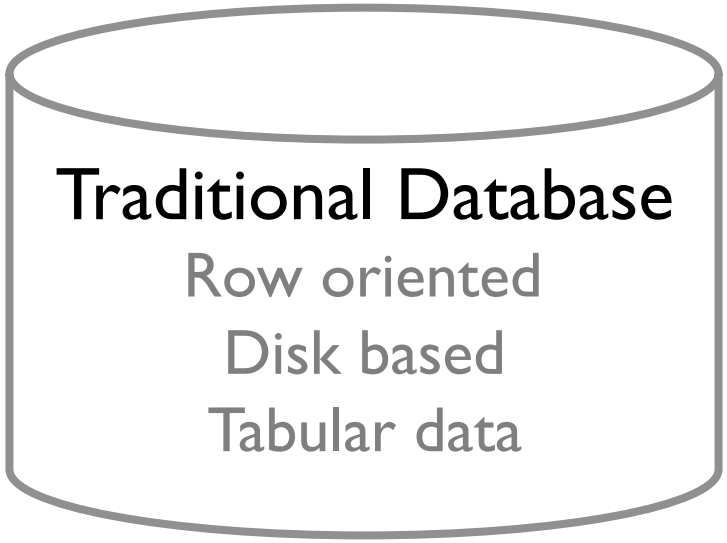
L25: Modern Data Storage

Even UNI: 614 Schermerhorn

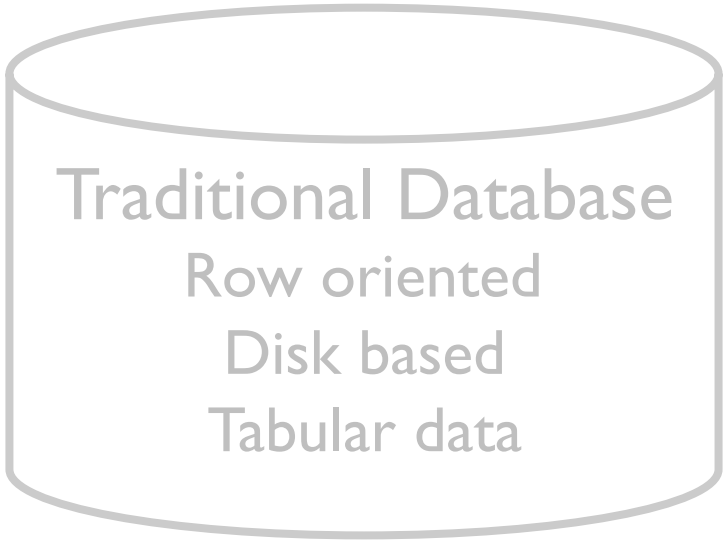


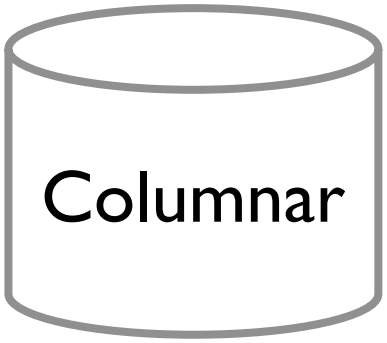
One Size Fits All

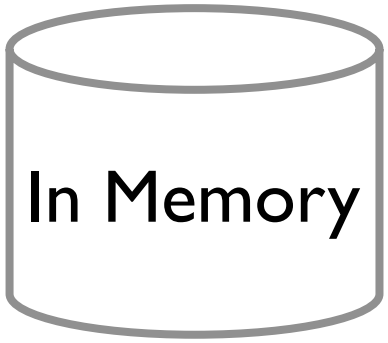


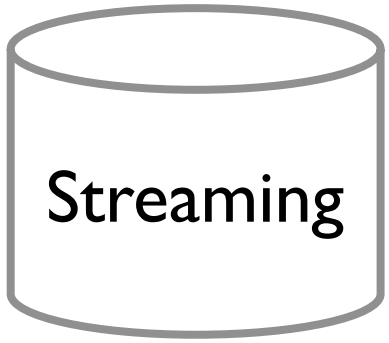
Consistent tools to access data
Widely understood
Rich, sophisticated queries, tools and features
Reliable

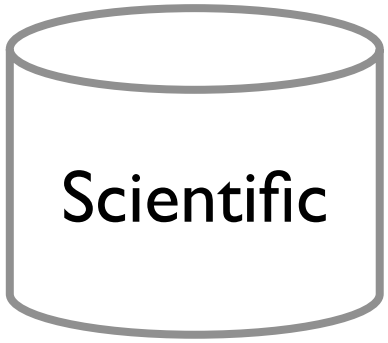
One Size Does Not Fits All



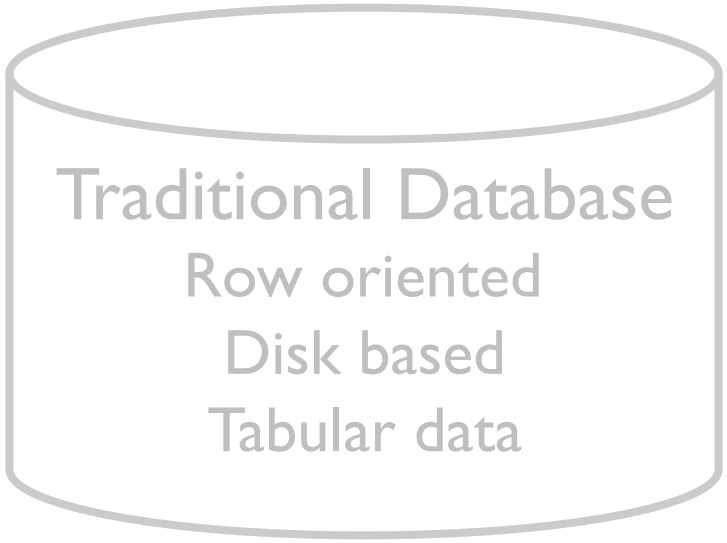


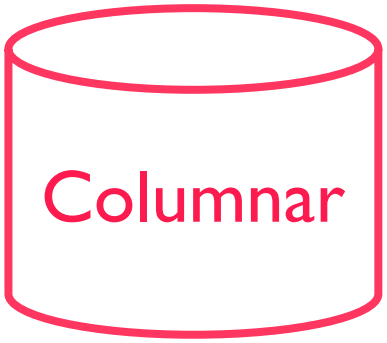


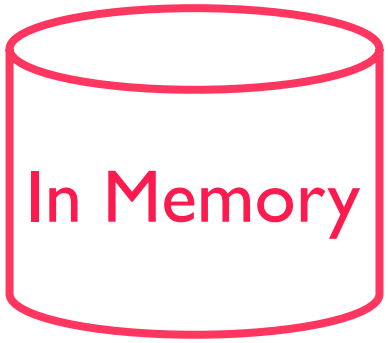


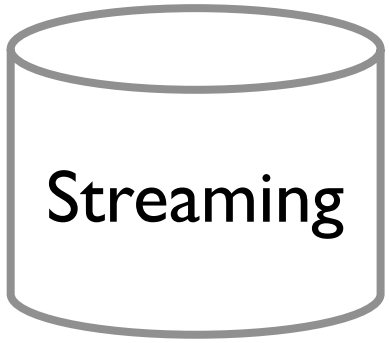


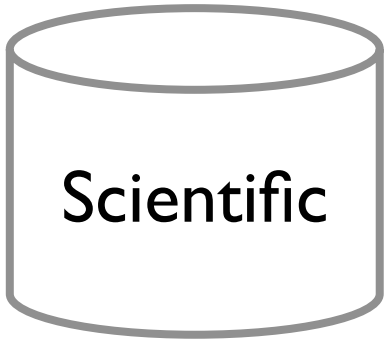
One Size Does Not Fits All











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, …
- Key-value: Mongo, Cassandra, Memcache, Redis, …

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, Cassandra, Memcache, Redis, …

DBMS-as-a-Service

- Microsoft Azure, Amazon Redshift/RDS, etc…

Modern Database Systems

90s: The Internet:

Every application is 24x7x365

Some applications have huge numbers of users

Traditional solutions fall over

Slashdot effect, Twitter fail whale, etc etc

Solution? Sharding (partitioning)

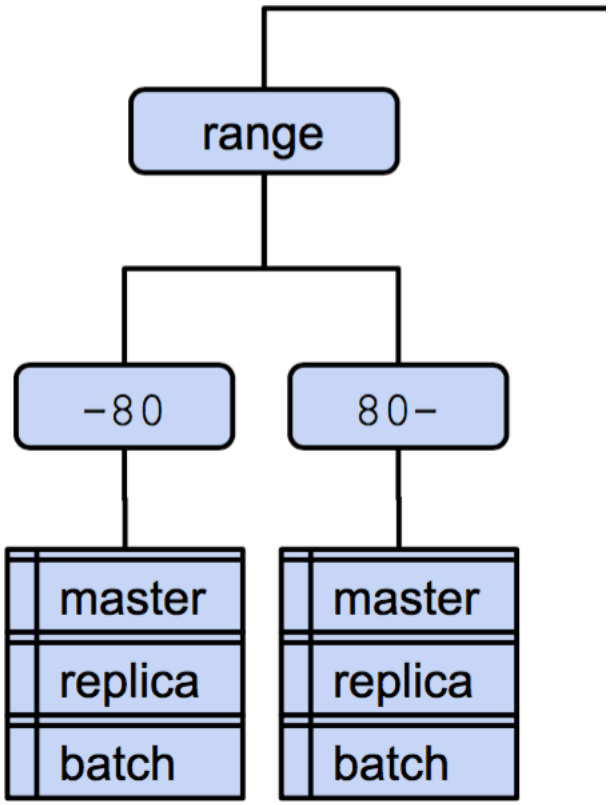
Split one database into many

Don't access different "shards" at once

Ebay: Shard per auction

Gmail: Shard per email

Many successes: EBay, Facebook, YouTube, Salesforce



YouTube

Sharding challenges

Limits supported operations

e.g. No joins/transactions between shards

Hot/large/imbalanced shards

e.g. Huge customers, popular pages

Manage many database instances

Transfers many challenges back to application

Google Bigtable; 2006

Sharding means you can't use relational model

Use simpler model: Key/value

Store unique keys, associated with values

key: "evan"

Value: "adjunct:w4111:ej@evanjones.ca"

Google Bigtable; 2006

Simple key/value model can be easily scaled

Split tables into tablets

Distribute tablets to multiple servers

Split tablets that have grown too big

Created the “NoSQL” movement

Other NoSQL systems

MongoDB, Cassandra: Distributed systems

Memcached (2003): Simple key/value in memory

Redis (2009): Key/value with lots of features!

Google Spanner, 2012

Globally distributed transactions

SQL interface

… basically a global relational database

Distributed databases today

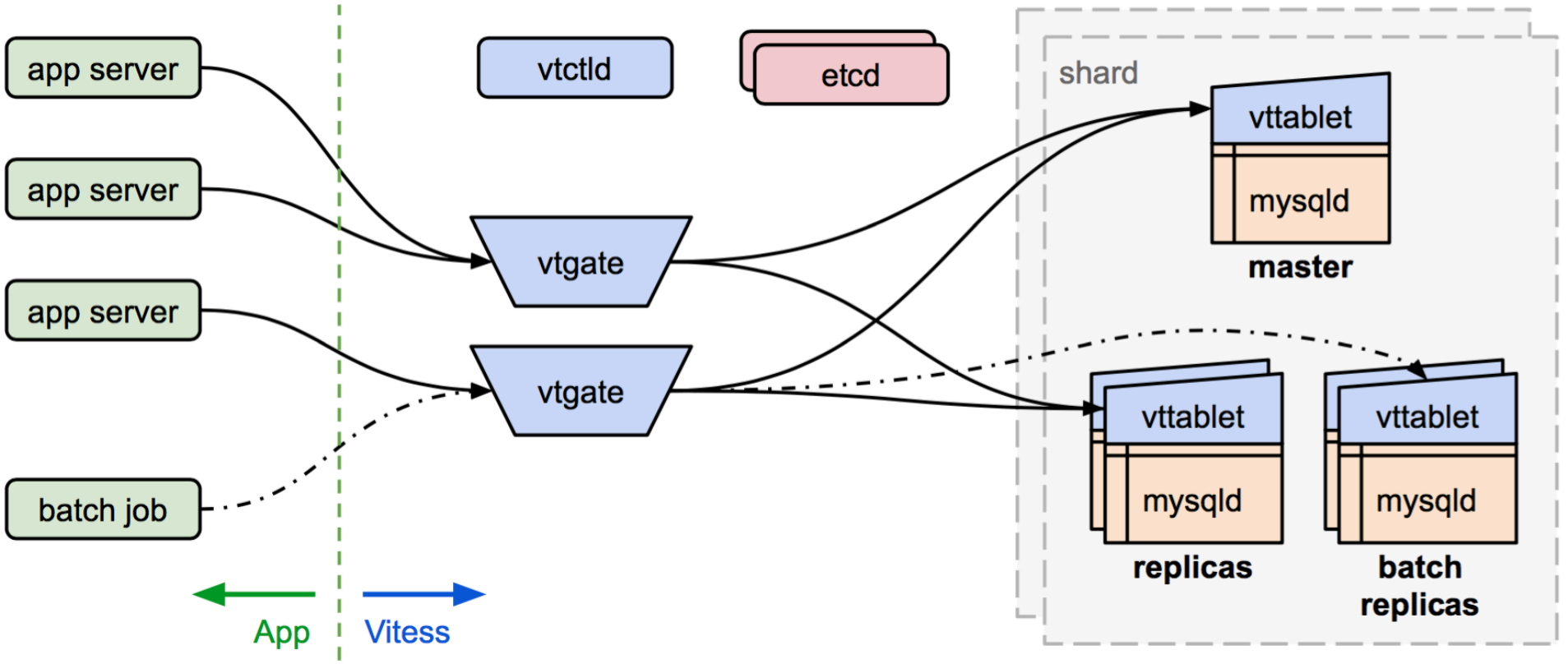
Surprisingly few that are widely used

Many analytic database product (next slides)

NoSQL: Cassandra, MongoDB, HBase

Many startups, no clear winners

Lots of sharded MySQL/Postgres/etc with custom tools: e.g. YouTube Vitess


app server
app server
app server
batch job
vtctld
etcd
vtgate
vtgate
shard
vttablet
mysqld
master
vttablet
mysqld
replicas
vttablet
mysqld
batch
replicas
App
Vitess

Why no commercial products? Hypothesis:

A "commodity" server today is big:

32 CPUs, 208 GB RAM, $1000/month cloud

Big enough for many apps

Extremely large apps: have own dev team

Sharding is painful, but does work

OLTP vs OLAP

OnLine Transaction Processing

- Interactive queries, low latency
- Small amount of data per transaction
- Modifies data

OnLine Analytical Processing

- Batch queries; "high" latency
- Aggregates, summaries
- Mostly read only

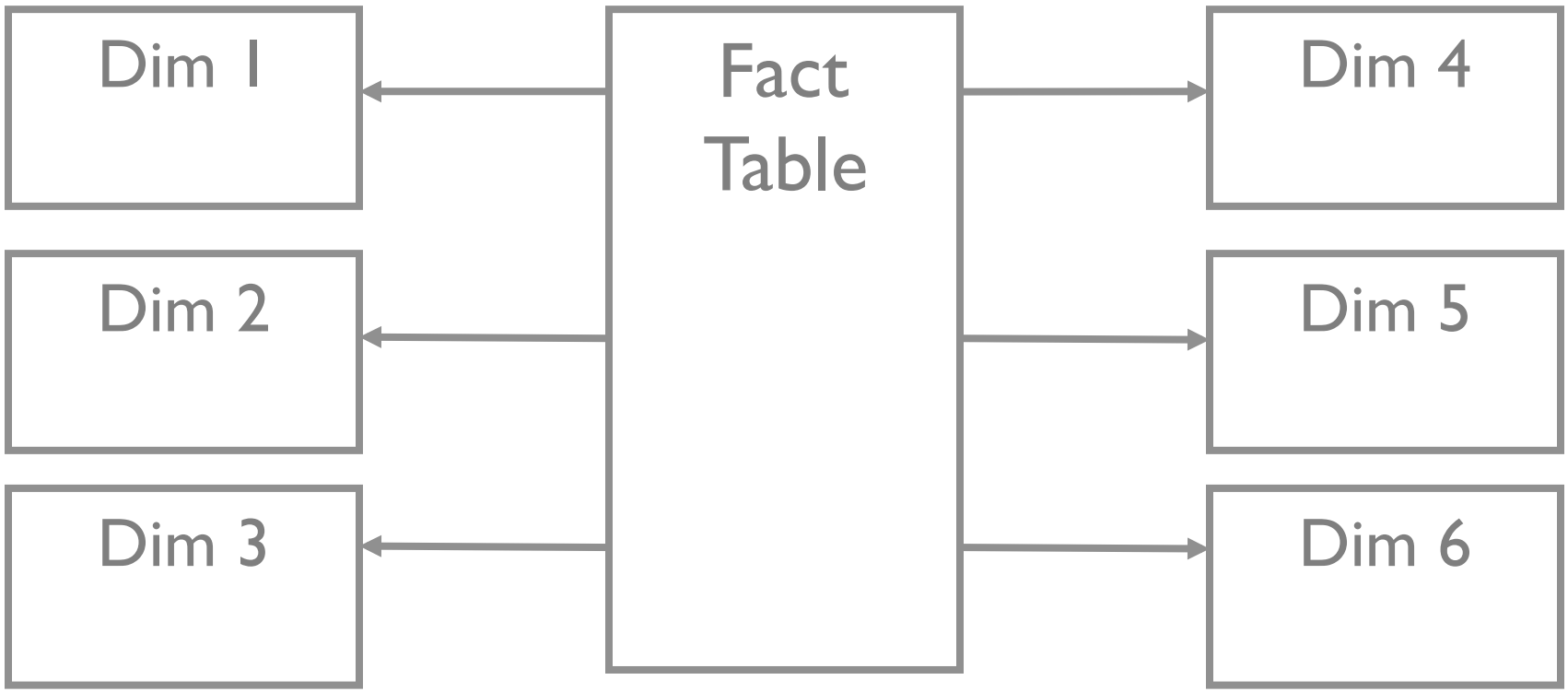
Data Warehouses

Store all historical data for future analysis

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

Most companies have something that serves this purpose

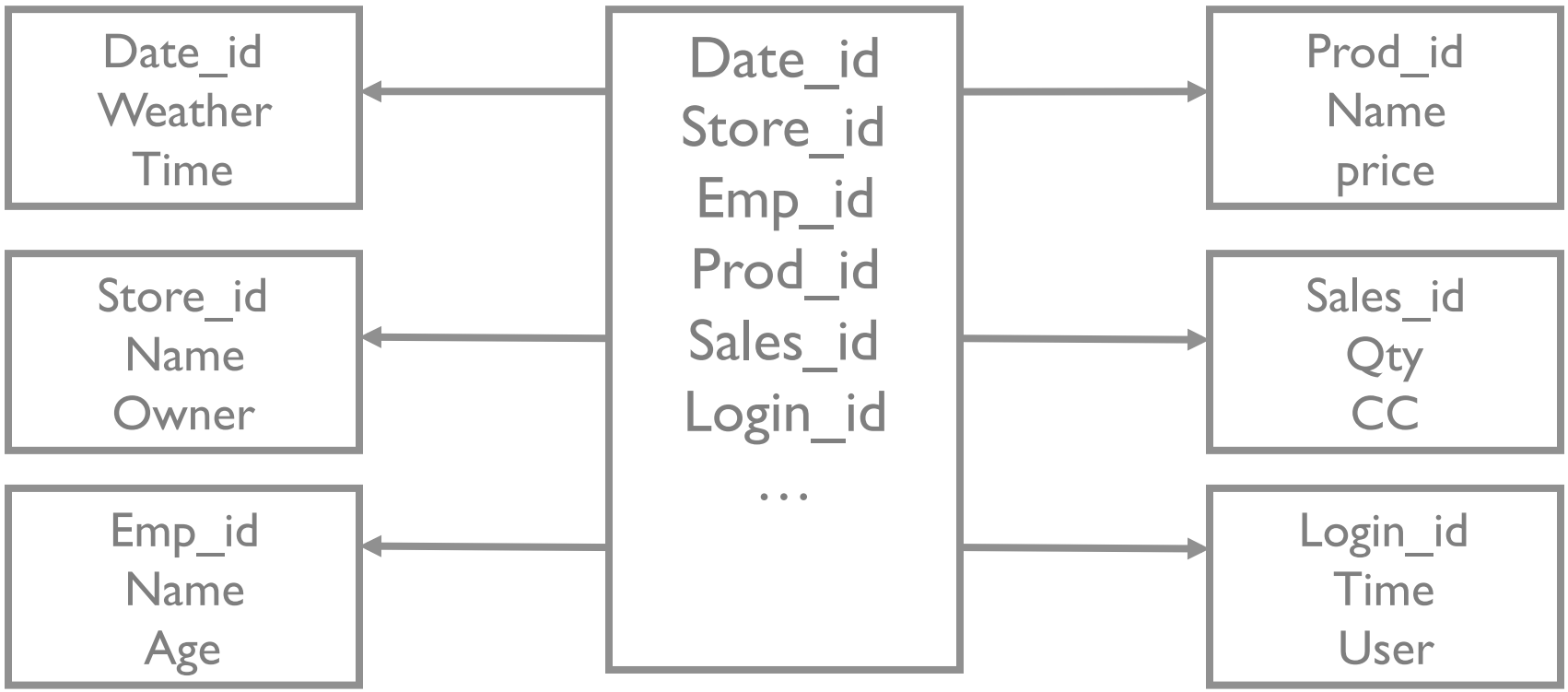
Star Schema

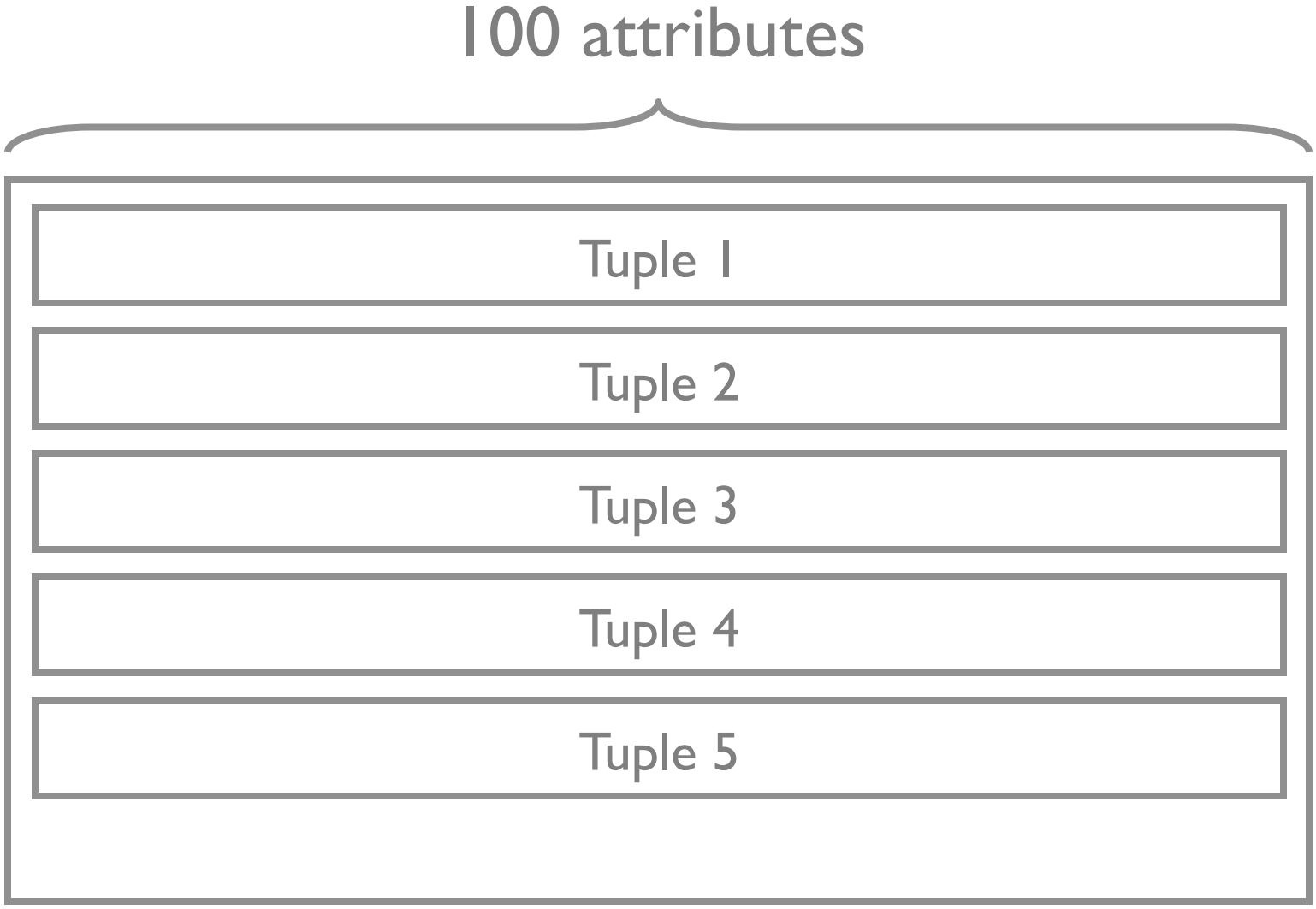


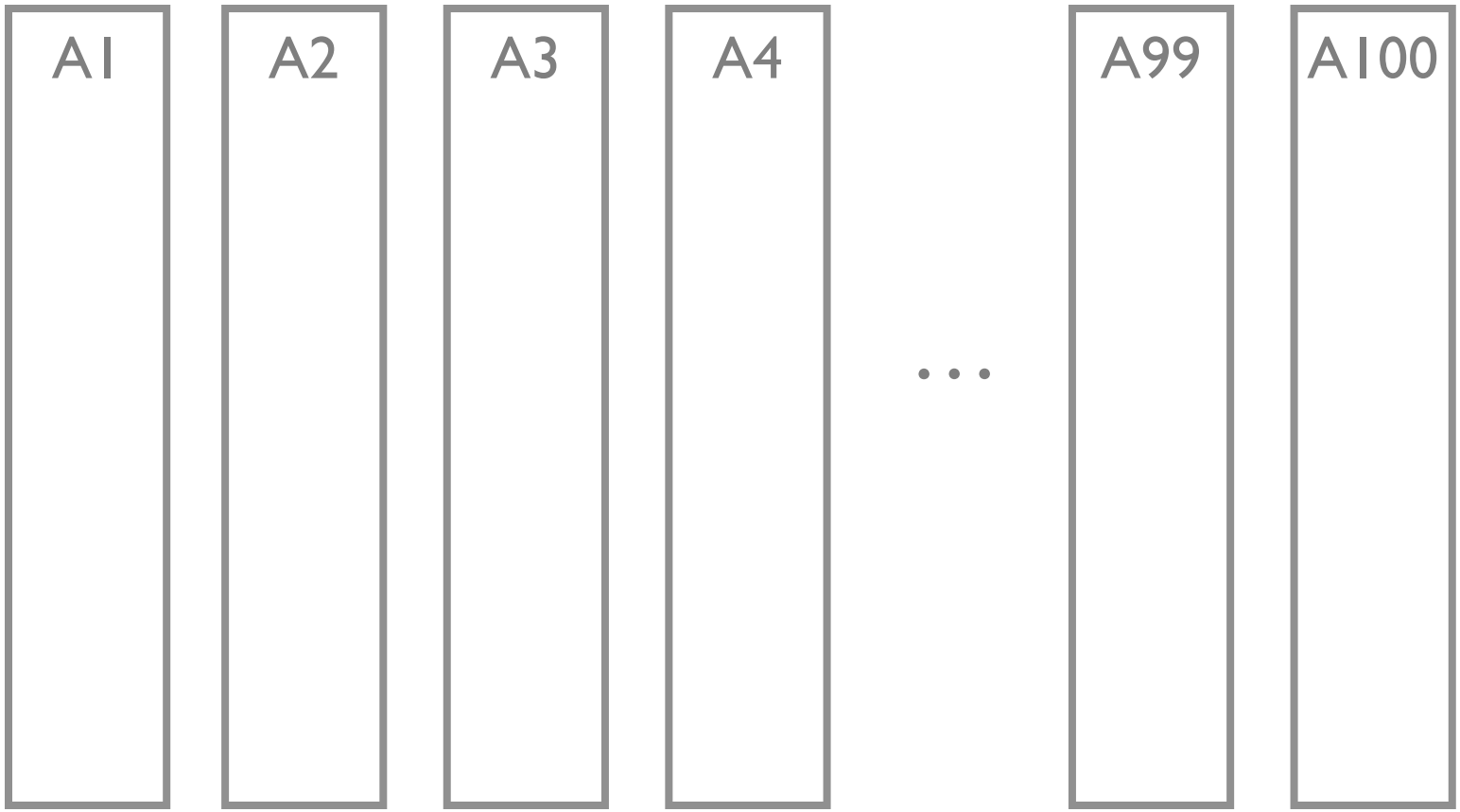
Star Schema

Fact table is “fat”; Dimensions are denormalized

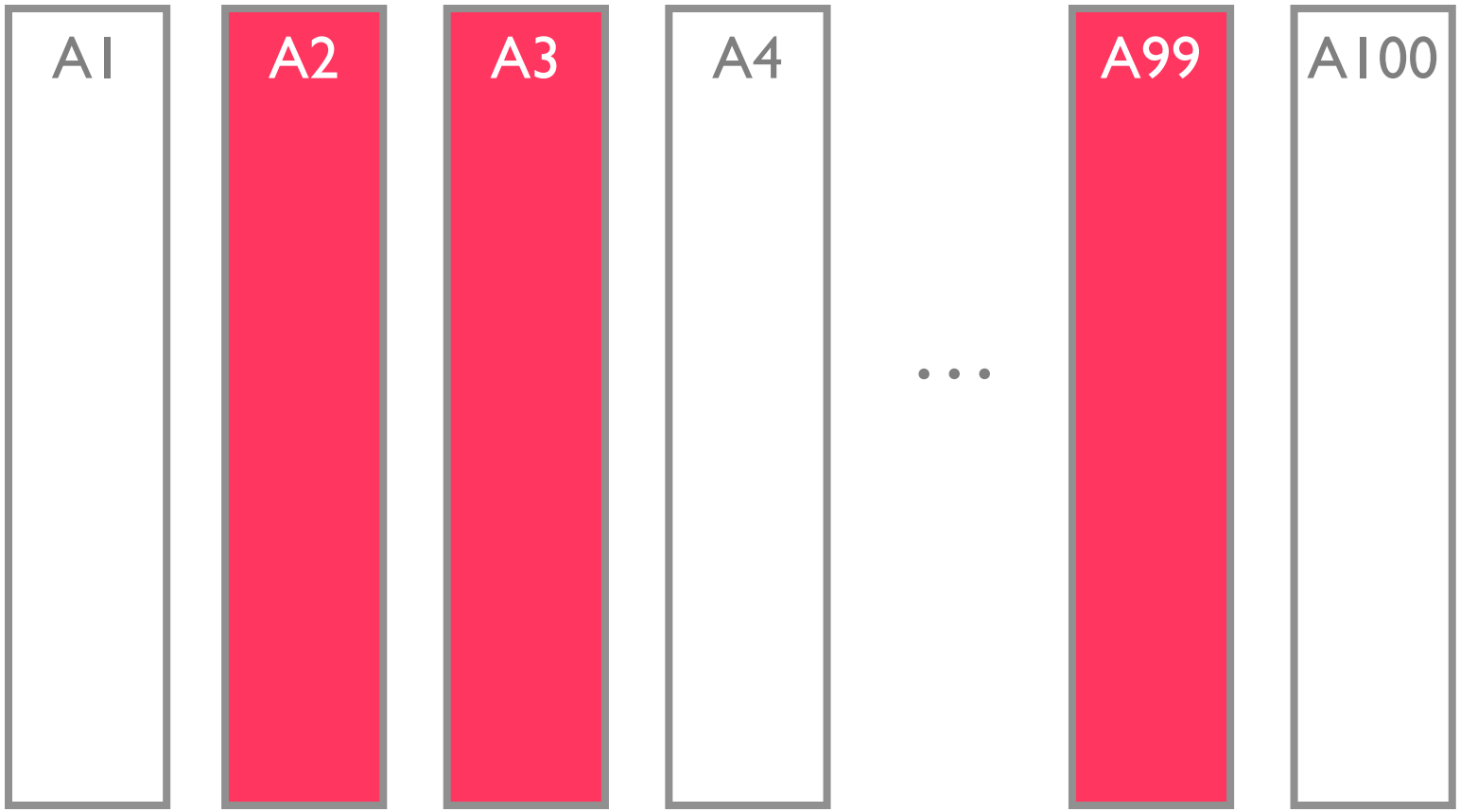
Queries access ~6 attrs




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


A1
A2
A3
A4
A99
A100

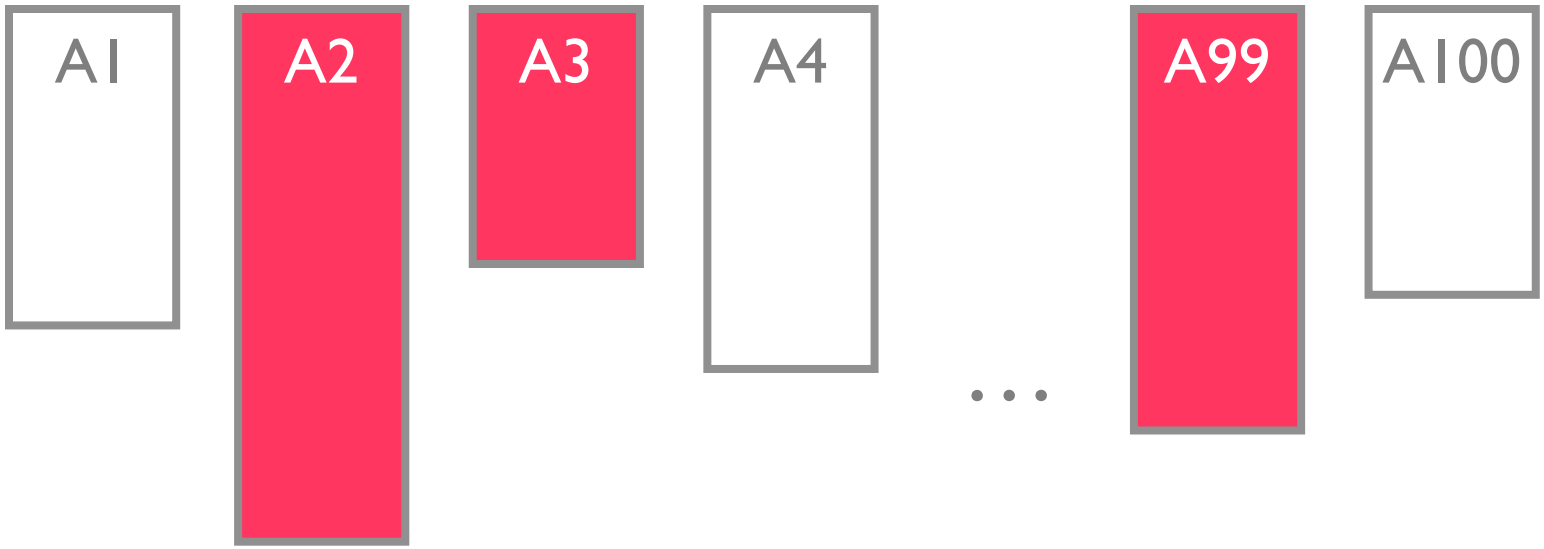
16x less data read. Unfair advantage.



16x less data read. Unfair advantage.

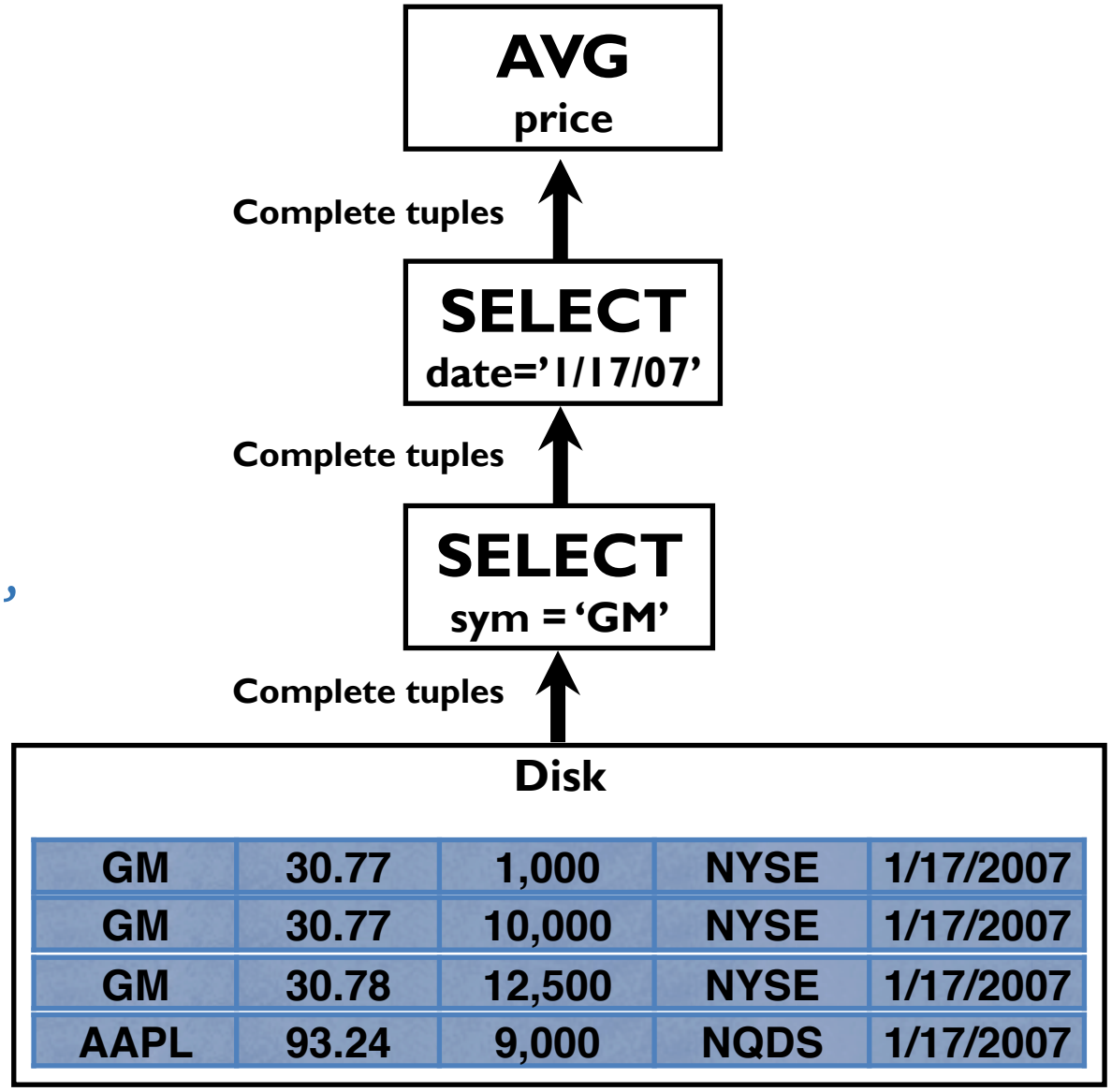
Compression better on single column

Execute on compressed data



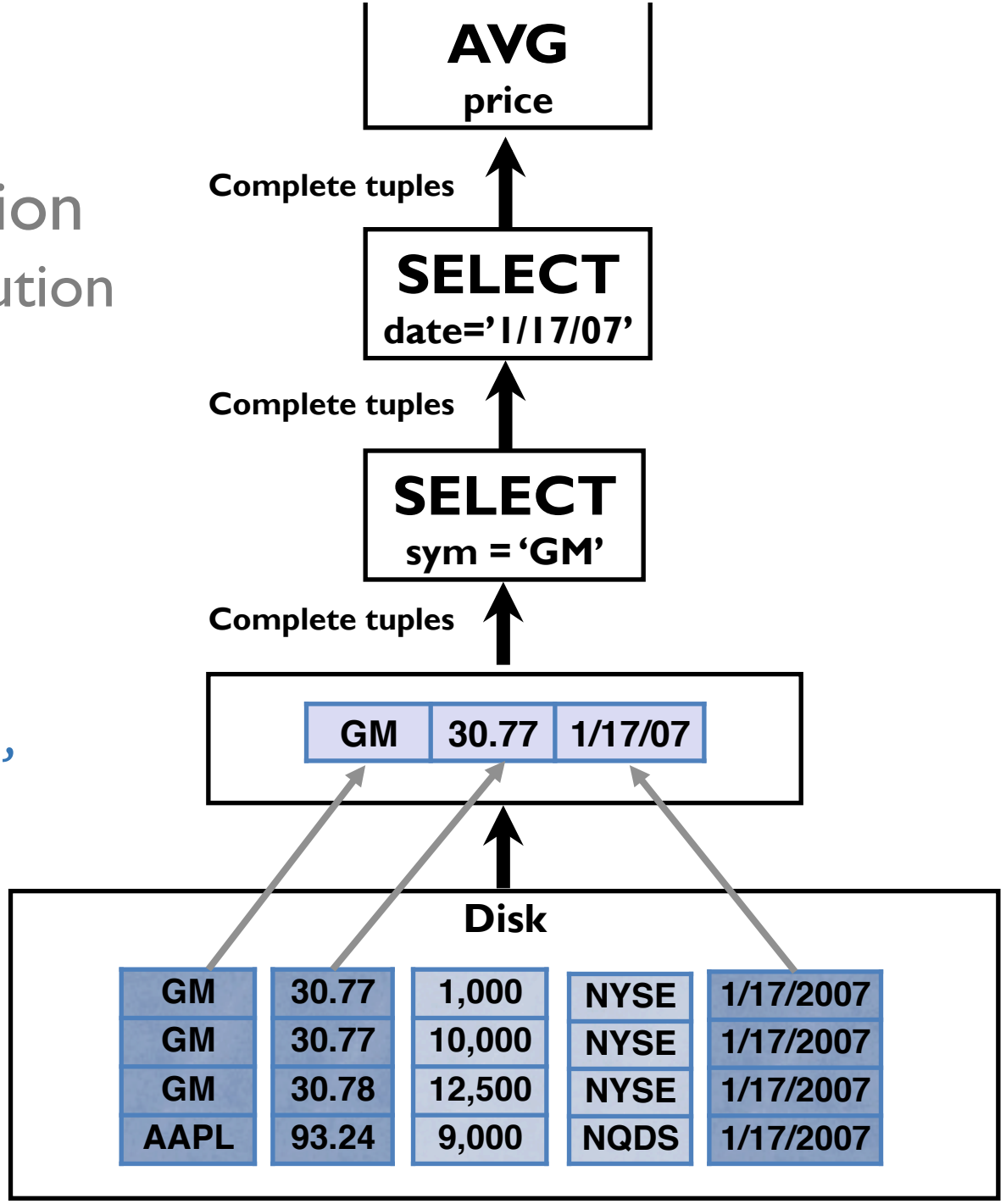
Traditional DBMS

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



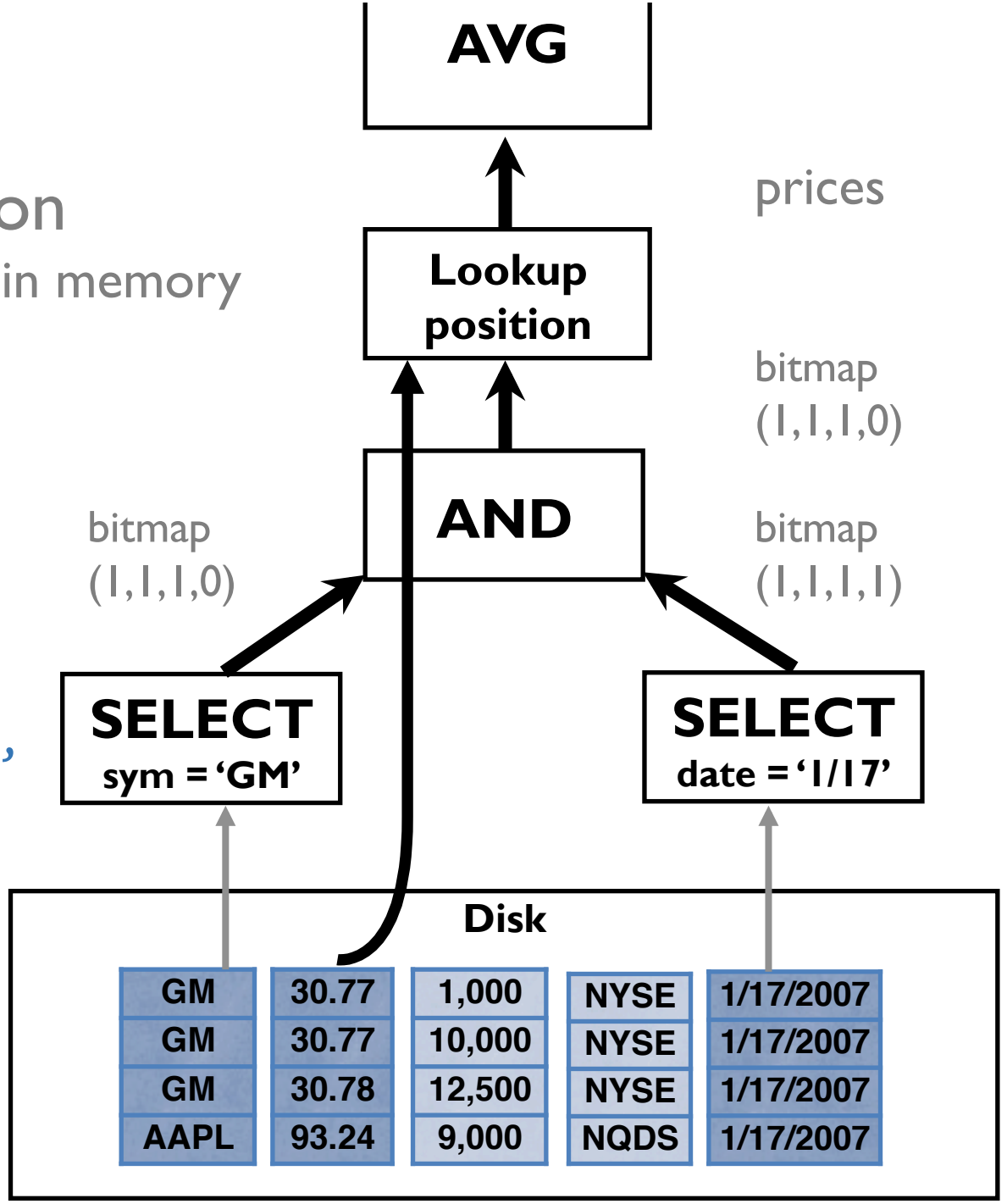
Naïve:
Early Materialization
Row oriented execution

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



C-Store
Late Materialization
Much less data moving in memory

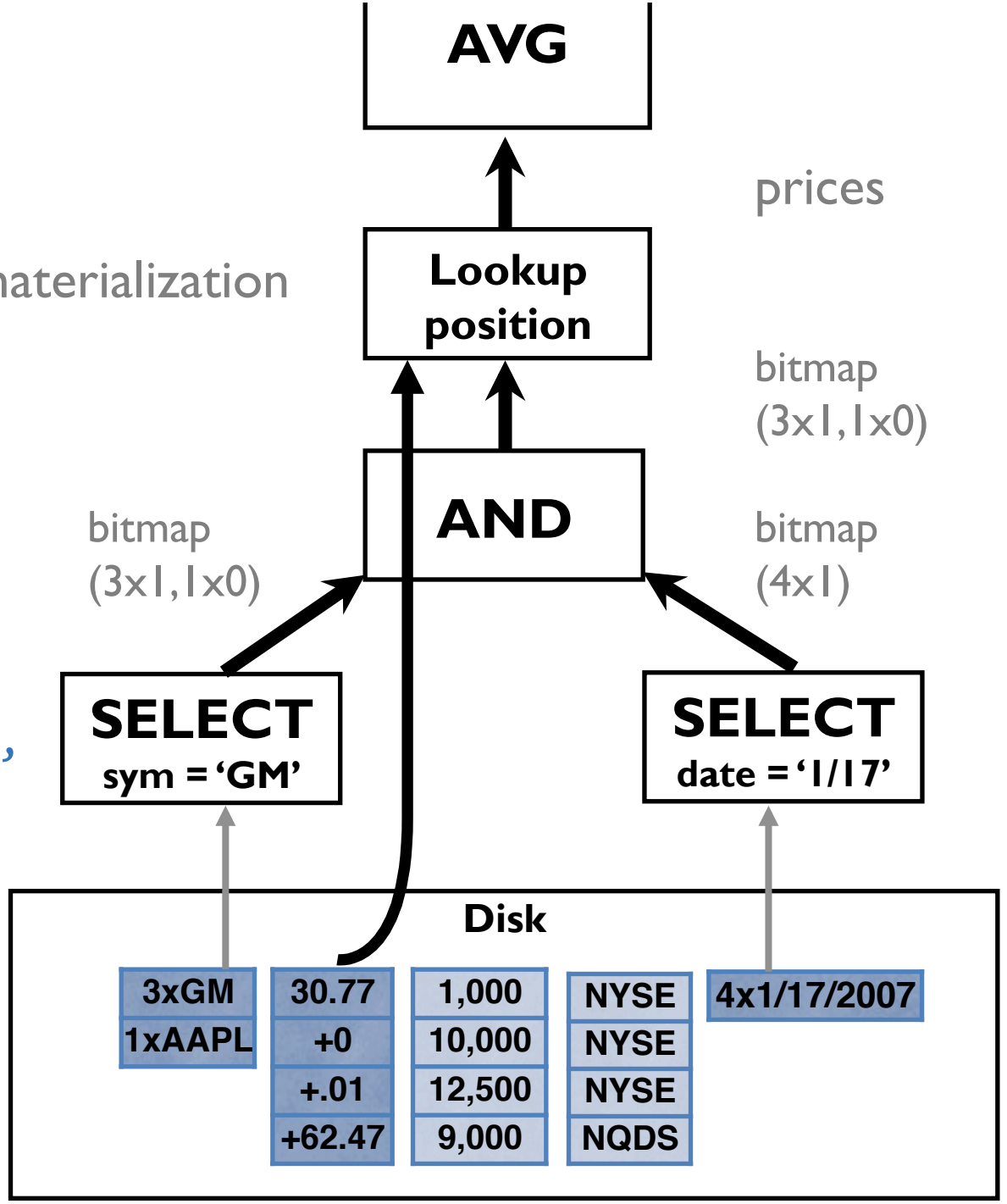
```
SELECT avg(price)
FROM tickstore
WHERE symbol = 'GM'
AND date = '1/17/2007'
```



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

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

(for OLAP queries)

Many successful products

HP Vertica

Teradata

Netezza

… many others

Meanwhile at the Internet companies

Record everything!

On hundreds or thousands of computers!

… how do we do anything with it?

Google MapReduce 2004

map(records) → (key, value)

sort all keys

reduce(key, [value]) → (key2, value2)

distributed to thousands of machines

Example: requests per day

map(request log record) → (day, 1)

sort all keys

reduce(day, [1, 1, …]) → (day, sum(counts))

Apache Hadoop, January 2006

Open source clone started by Doug Cutting

Cutting was at Yahoo!, working on search

Gained significant adoption within ~2 years

Cloudera started to commercialize

Hortonworks spun out of Yahoo much later

Hadoop versus Databases "debate"

Database industry: MapReduce is a bad implementation of distributed databases

MapReduce crowd: databases can't scale

More rational perspective

MapReduce:
Relatively easy to scale
Amazing for unstructured data
Manual work for every “query”

Databases:
Setup and “loading” takes work/time
Rich queries without much effort

Today:

Google Powerdrill/BigQuery: SQL interface

Cloudera Impala: SQL interface

Facebook Presto: SQL interface

Reason? Not perfect but well understood

Queries better than implementing joins by hand!

Optimizers frequently faster than humans

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 recovery

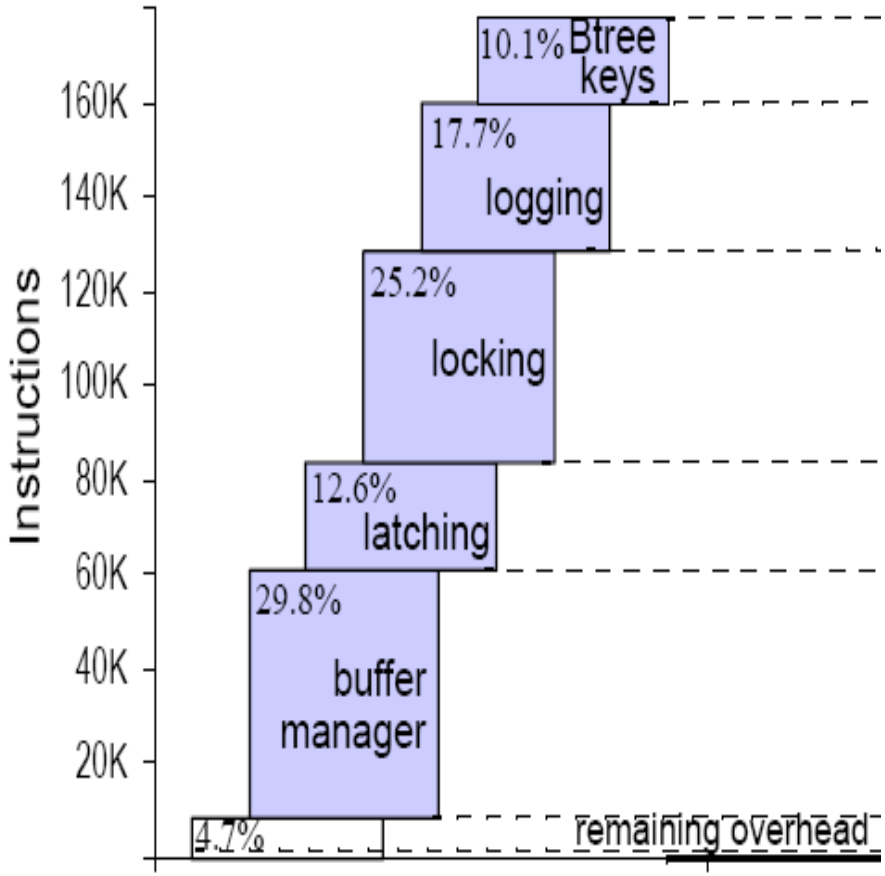
Active-active replication for fault-tolerance

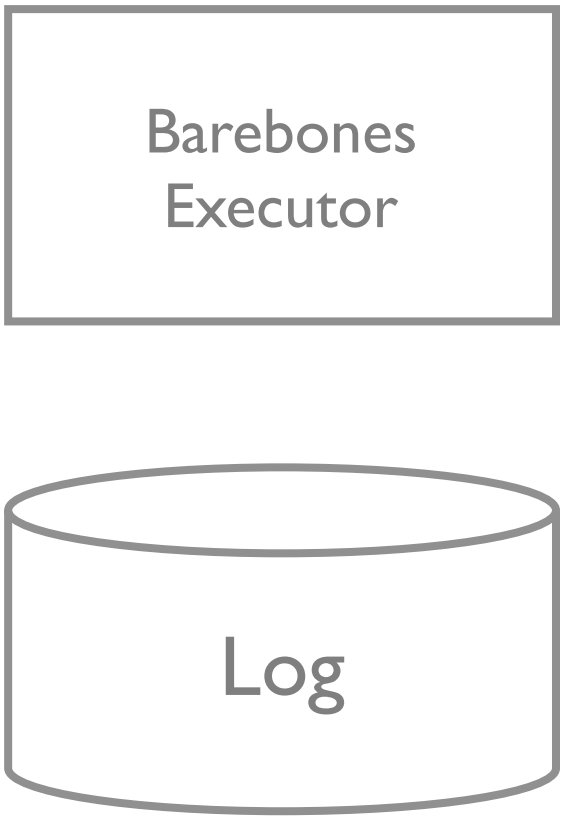
Traditional Database

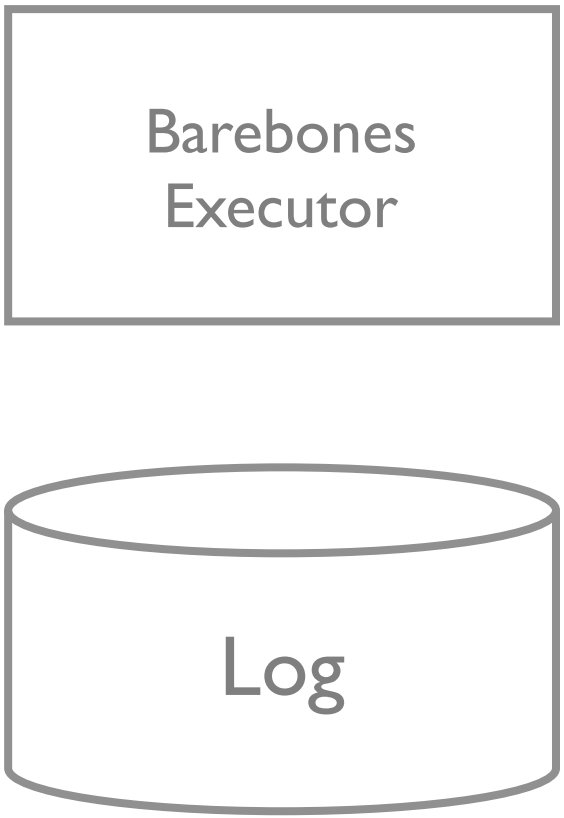
Indexes	queries go faster
Concurrency	queries go faster
Locking	serializability
Logging	recovery
Buffer Manager	manage pages in memory

Results after removing the components (in # instruction)

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




Barebones
Executor
Log



Redis?
Memcache?
SQL?

Databases as a Service

Amazon: Redshift (columnar), RDS (traditional), DynamoDB (key/value)

Google: Datastore/Bigtable (key/value), Cloud SQL (traditional), BigQuery (columnar)

Microsoft: SQL Azure, DocumentDB (key/value)

Internal: Click to provision

Database Service Challenges

Legal rules: Data residency (EU, others)

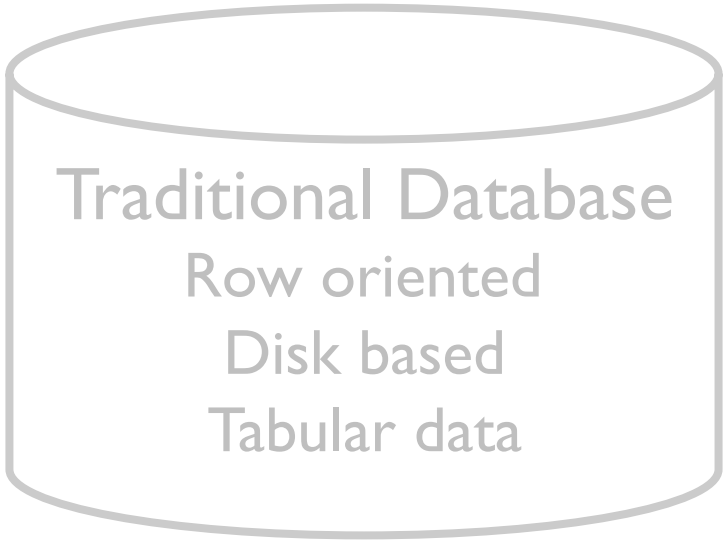
Privacy/Compliance (e.g. HIPAA for health)

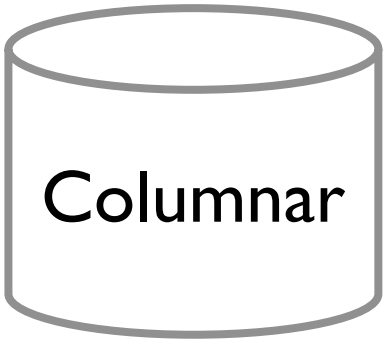
Security/Compliance (e.g. PCI for credit cards)

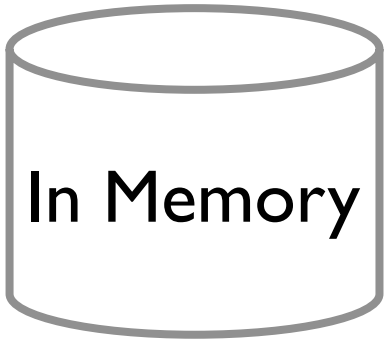
Data gravity:

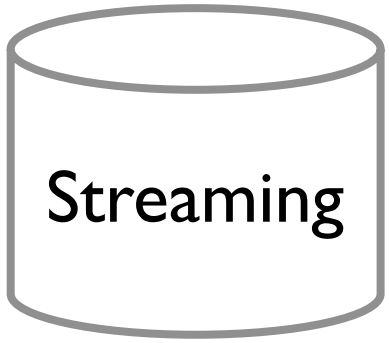
- Large data is hard to move
- Latency across the Internet can be bad

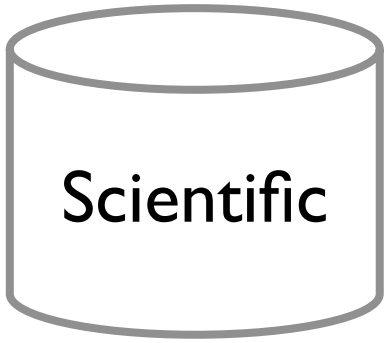
One Size Does Not Fits All



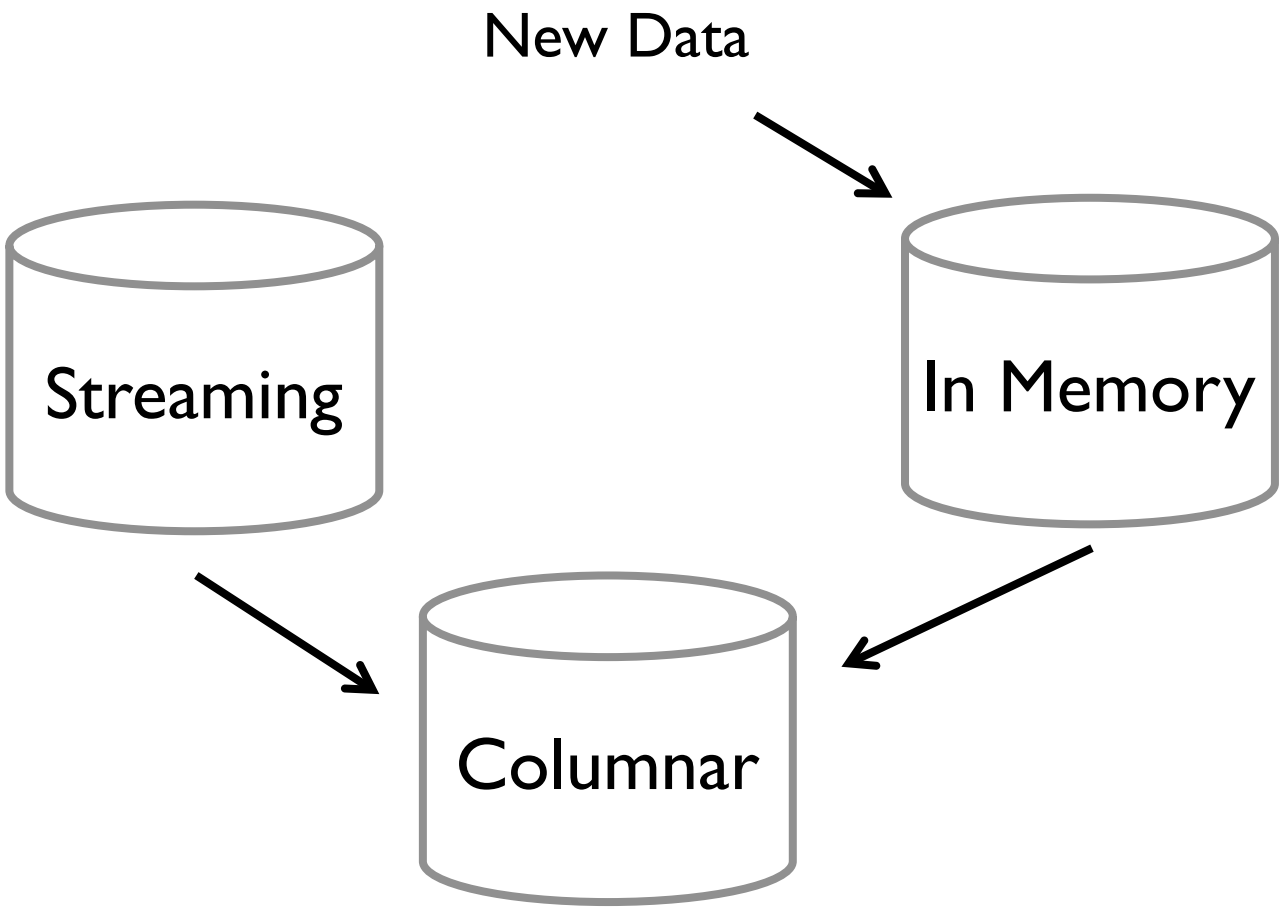








“Modern” Systems: Connected!



Thanks and good luck!

Feel free to contact me:

ej@evanjones.ca

http://www.evanjones.ca/