Load - II

big data technology

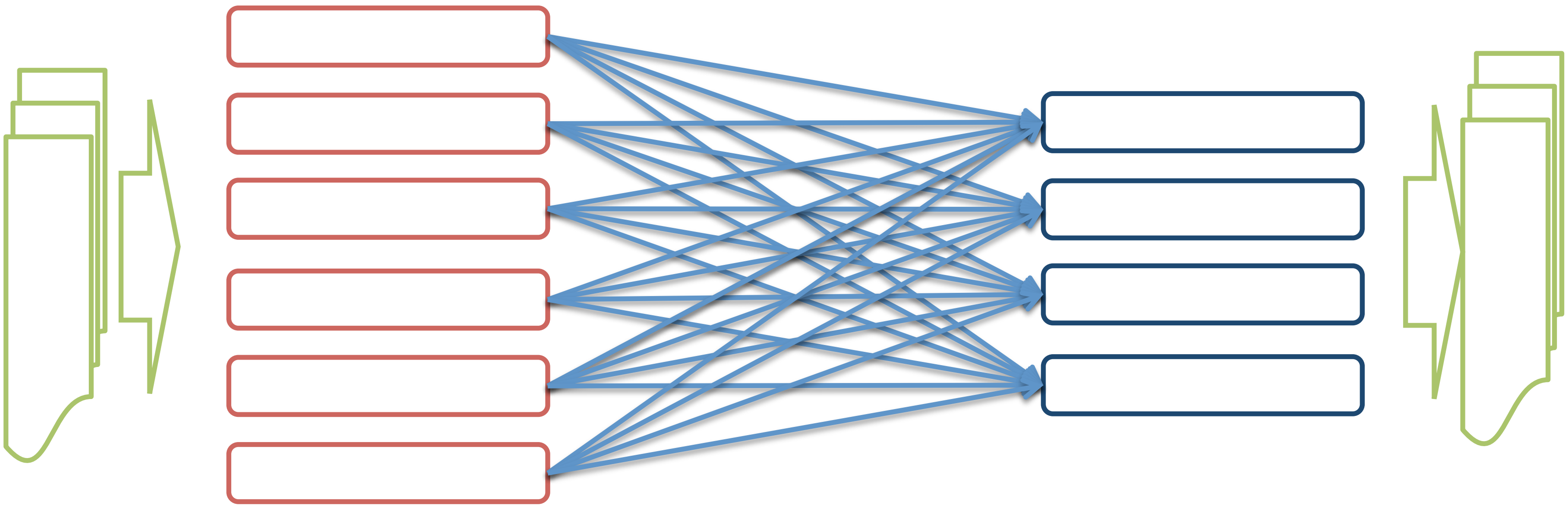
week 3:

map-reduce and programming assignment

week 4:

distributed file-systems, databases, and trends

map-reduce input and output



map-reduce reads data and *writes* fresh data –

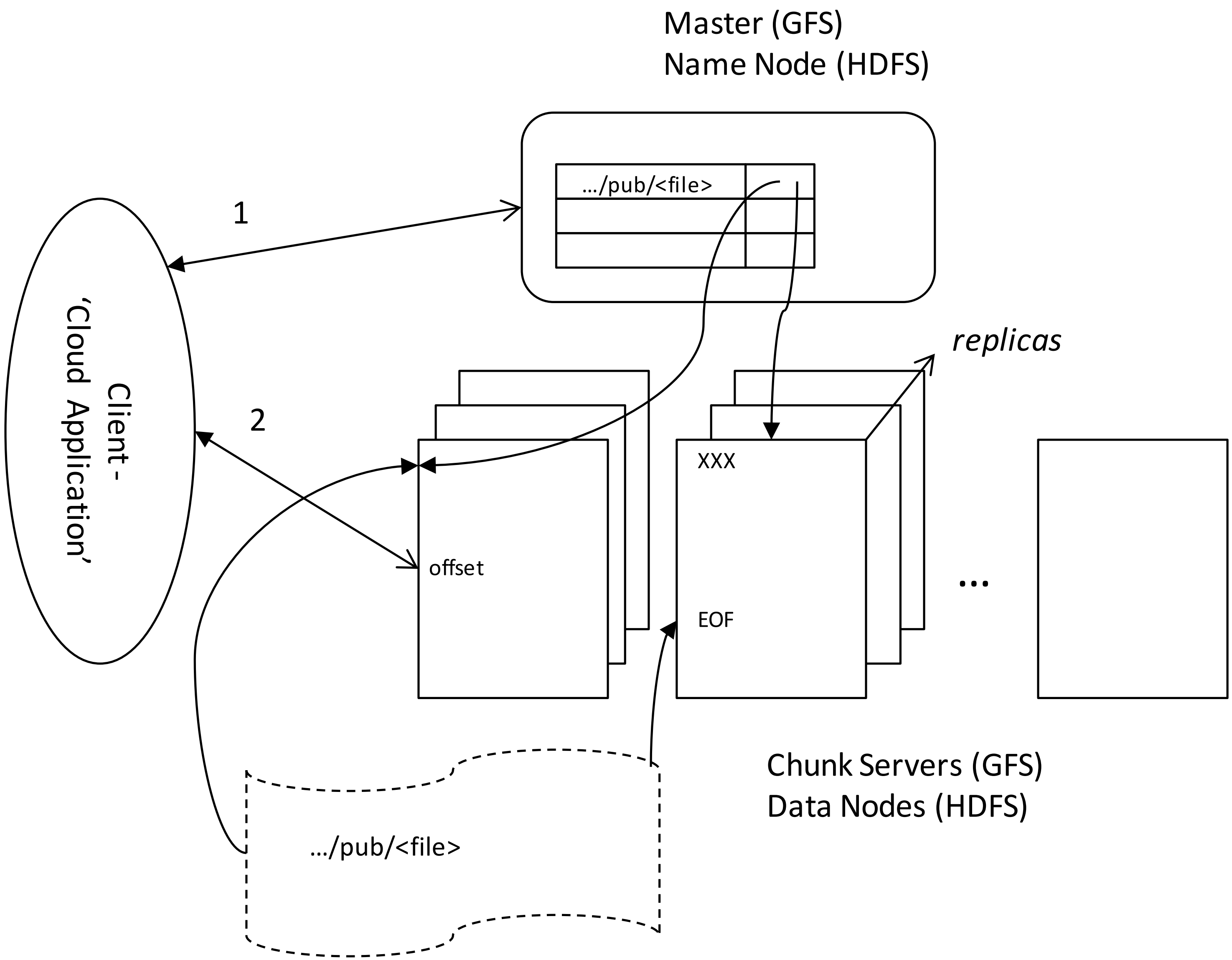
from where - potential bottleneck ?

distributed data, in a distributed file system or database

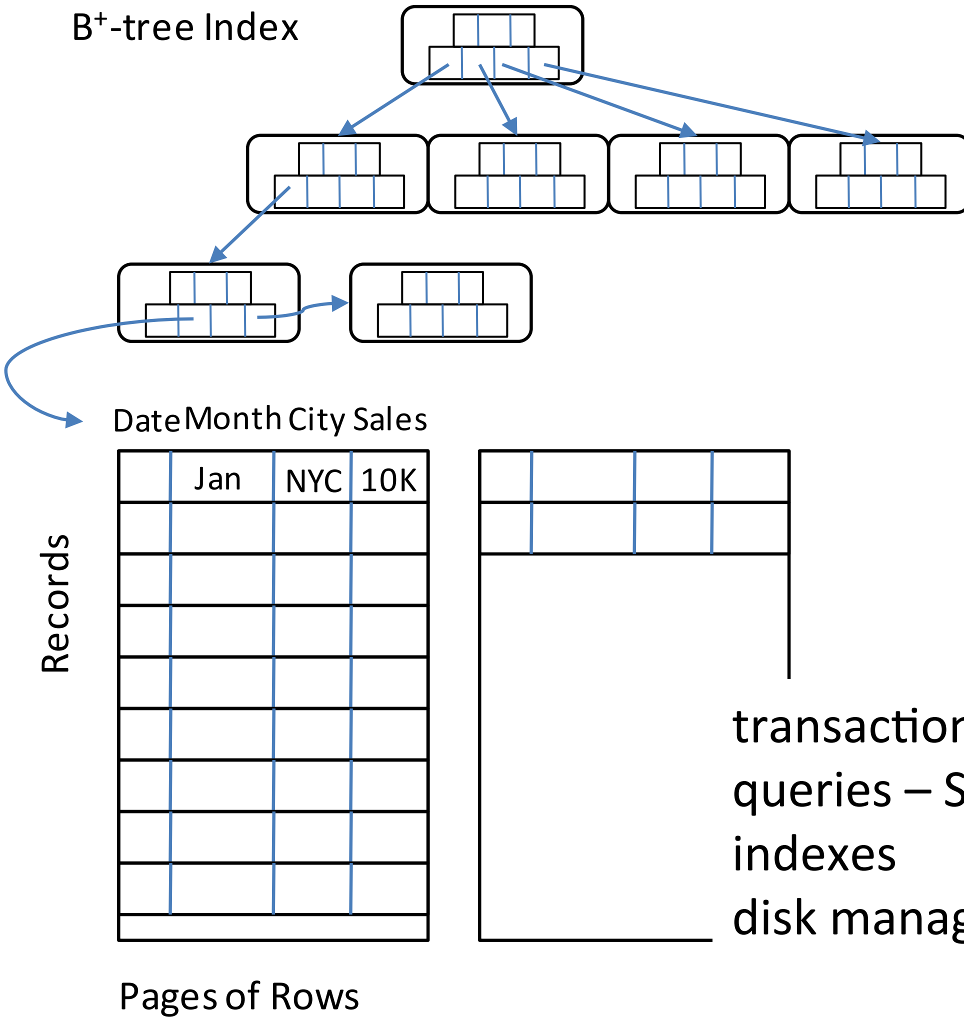
parallel reading and writing

we have discussed processing-node failures; what about data?

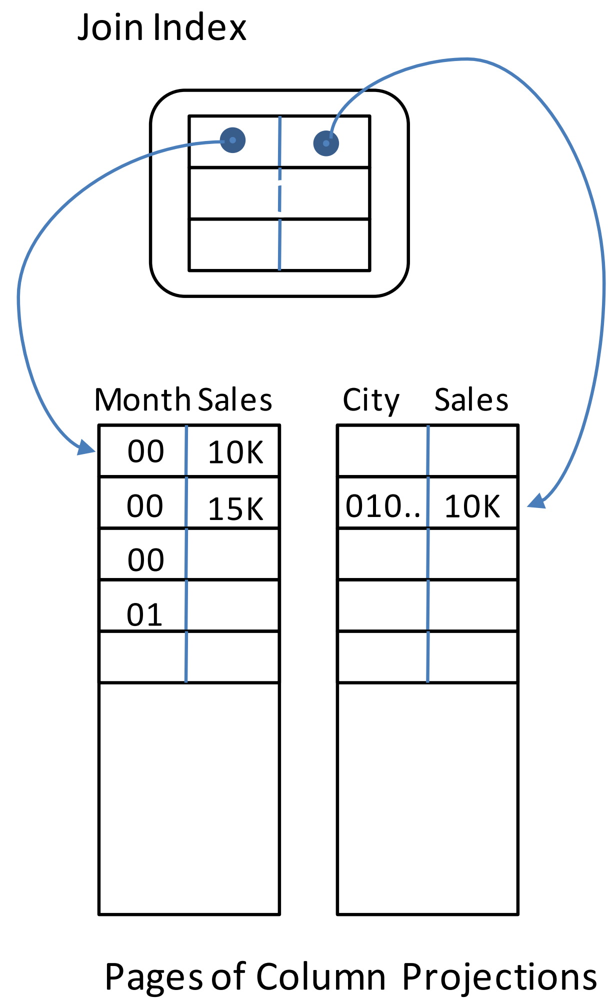
distributed file systems (GFS, HDFS)



overview of relational databases



- transactions
- queries – SQL
- indexes
- disk management

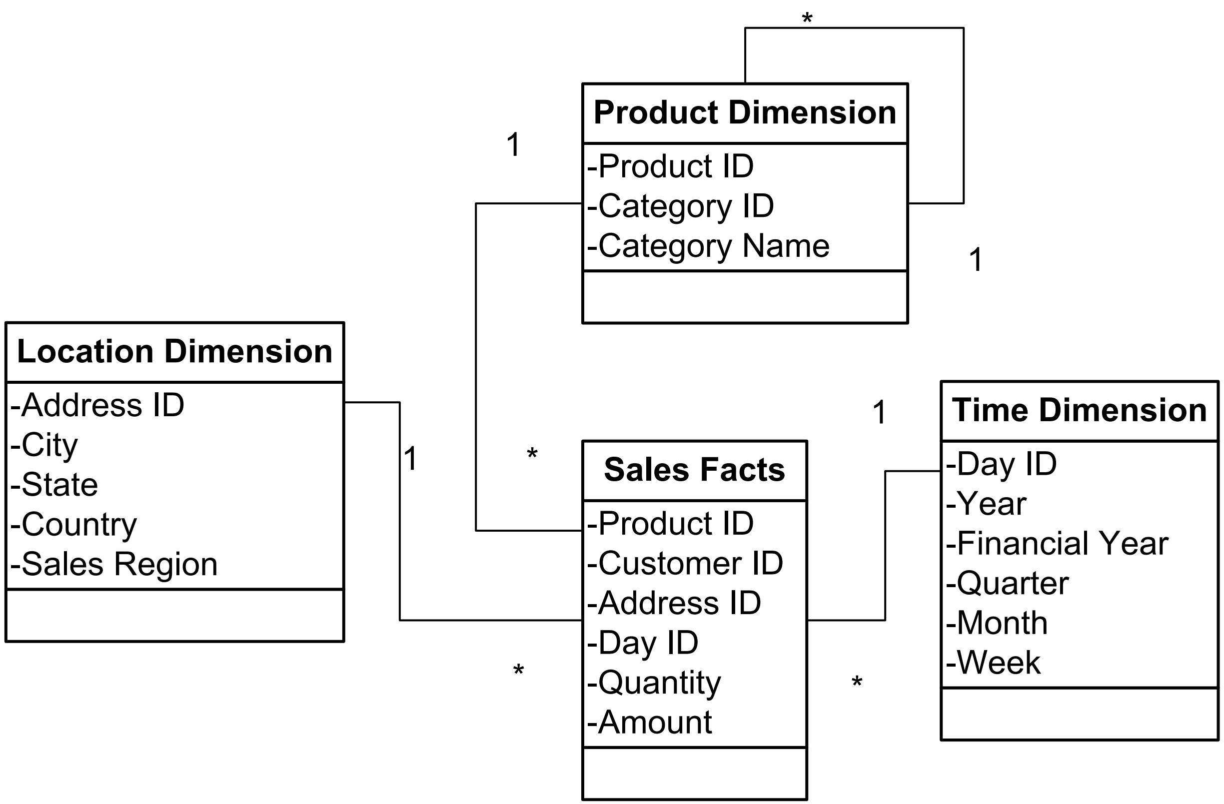


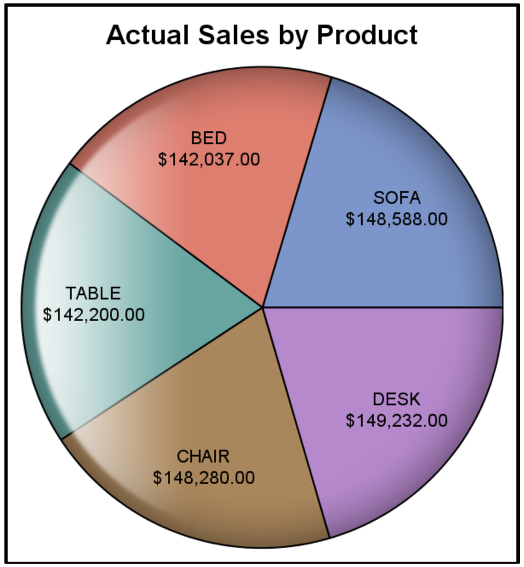
Row Oriented Database

Column Oriented Database

OLAP (“online analytical processing”)

e.g.: select SUM(S.amount), S.pid, P.catname from S where S.did=T.did S.pid = P.pid and T.qrtr = 3 group by catname



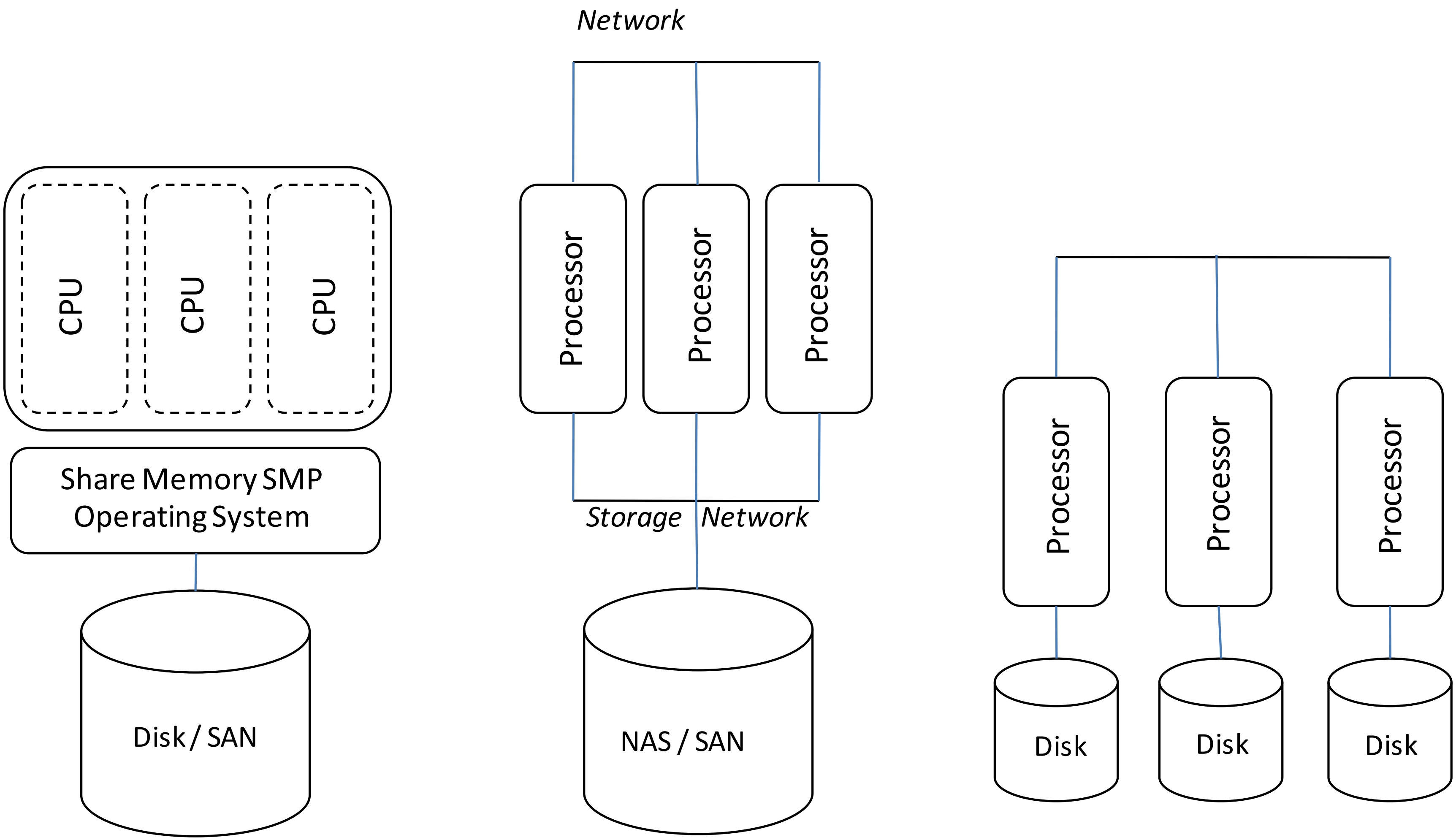


databases: why?

- transaction processing (ACID properties)
- SQL – queries and indexing

- transaction processing *not* need for analytics
  - though there *may* be advantages in not having to move data out of a transaction store if avoidable
- queries – yes, but if large volumes of data are being touched (e.g. joins, large-scale counting, building classifiers, etc.); indexes become *less* relevant
  - resilience to hardware failures, which MR provides, *is* vital.
- *but* OLAP – can be viewed as computing a part of the joint distribution $P(f_1 \ldots f_n)$ – using intuition to select

parallel databases



Shared Memory

Shared Disk

Shared Nothing

database evolution

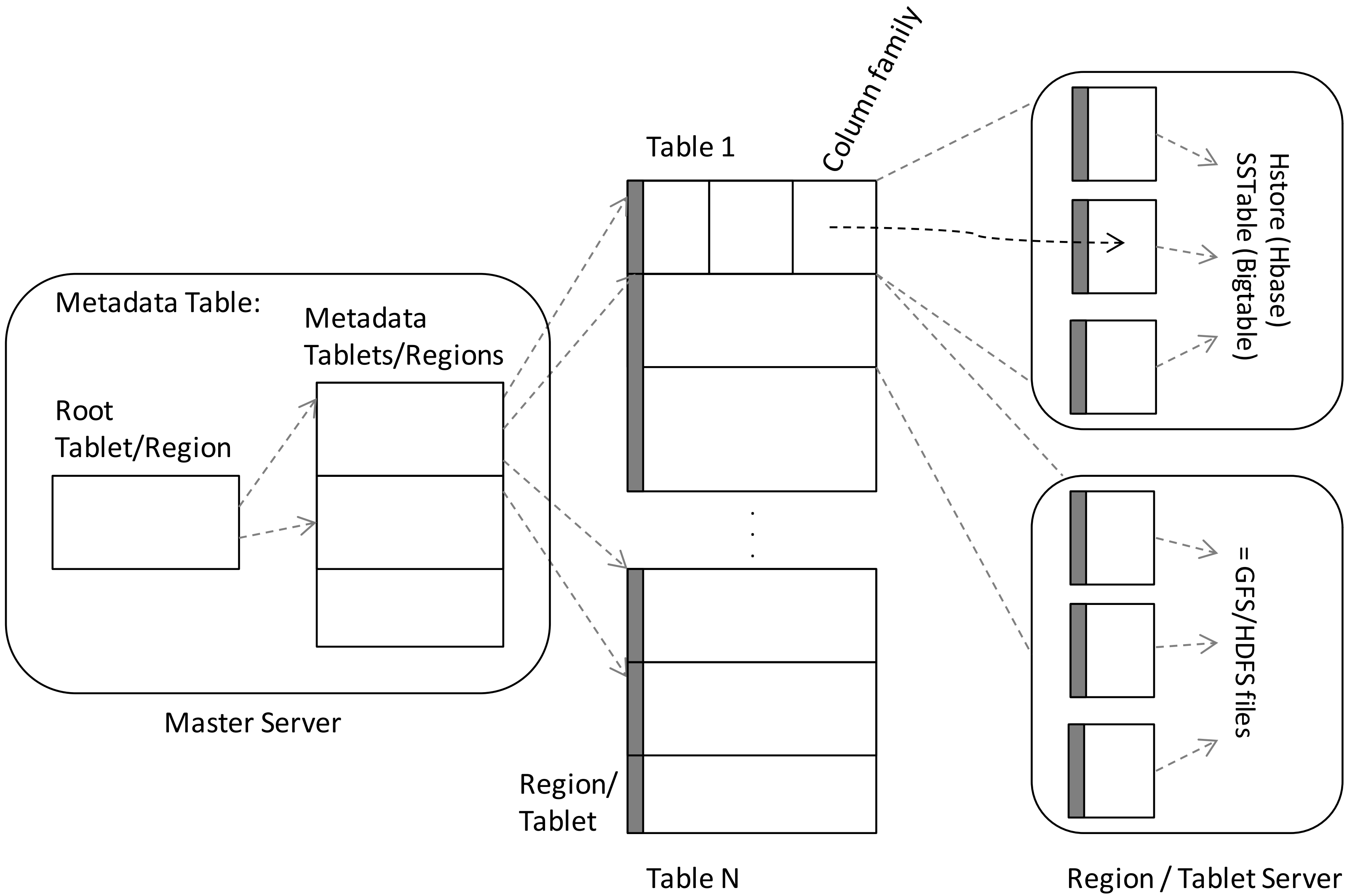
noSQL databases

- no ACID transactions
- *sharded* indexing
- restricted joins
- support columnar storage (if needed)

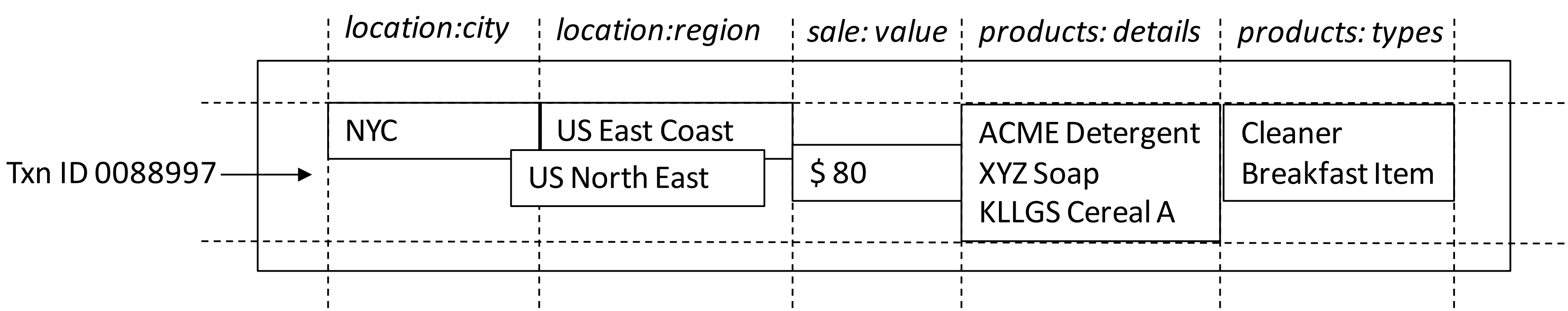
in-memory databases

- real-time transactions
- variety of indexes
- complex joins

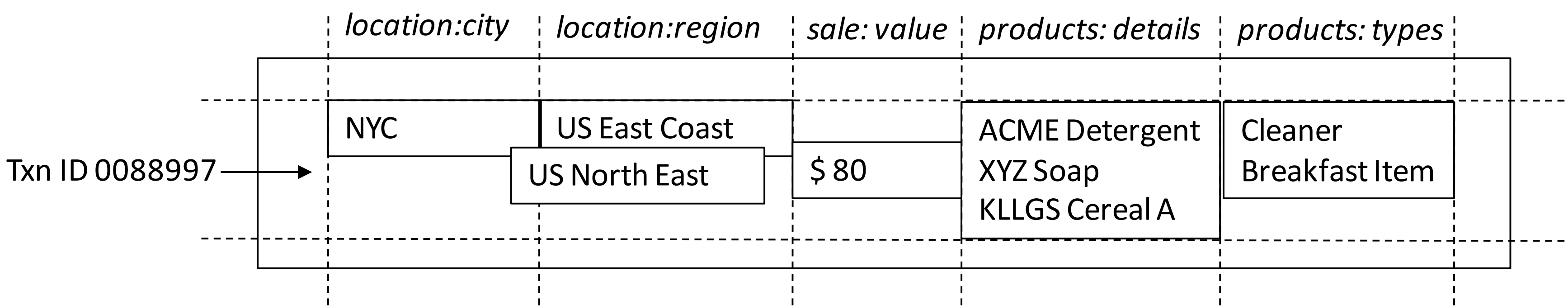
big-table (HBase)

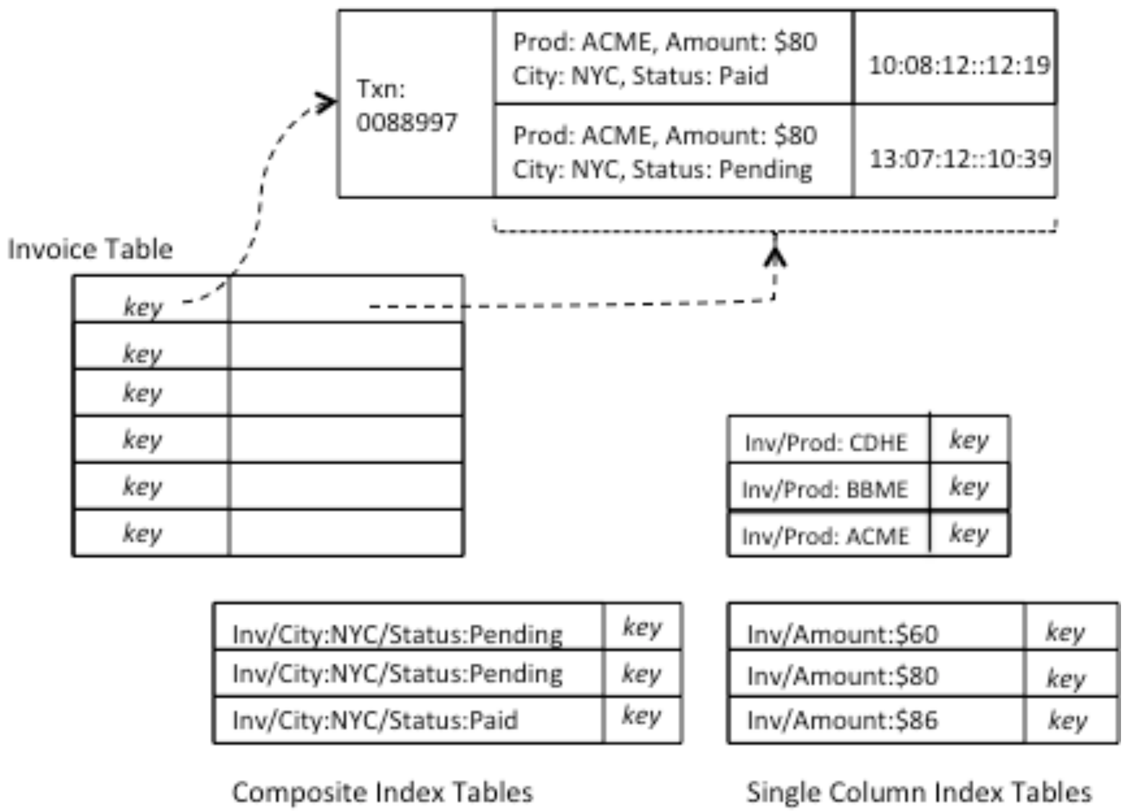


e.g. data in big-table



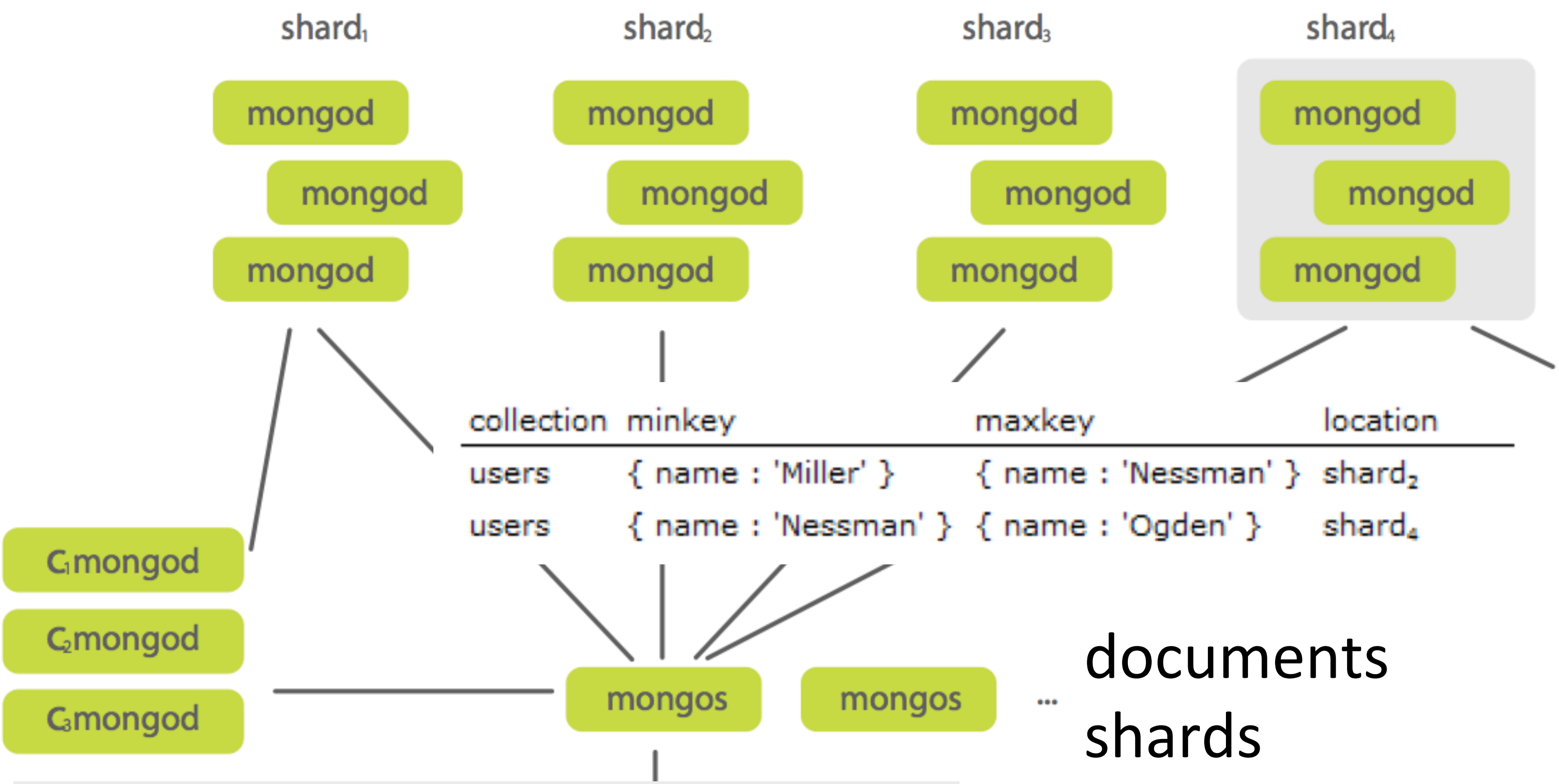
e.g. indexing *sharded* data (big-table)





Google App Engine
Datastore

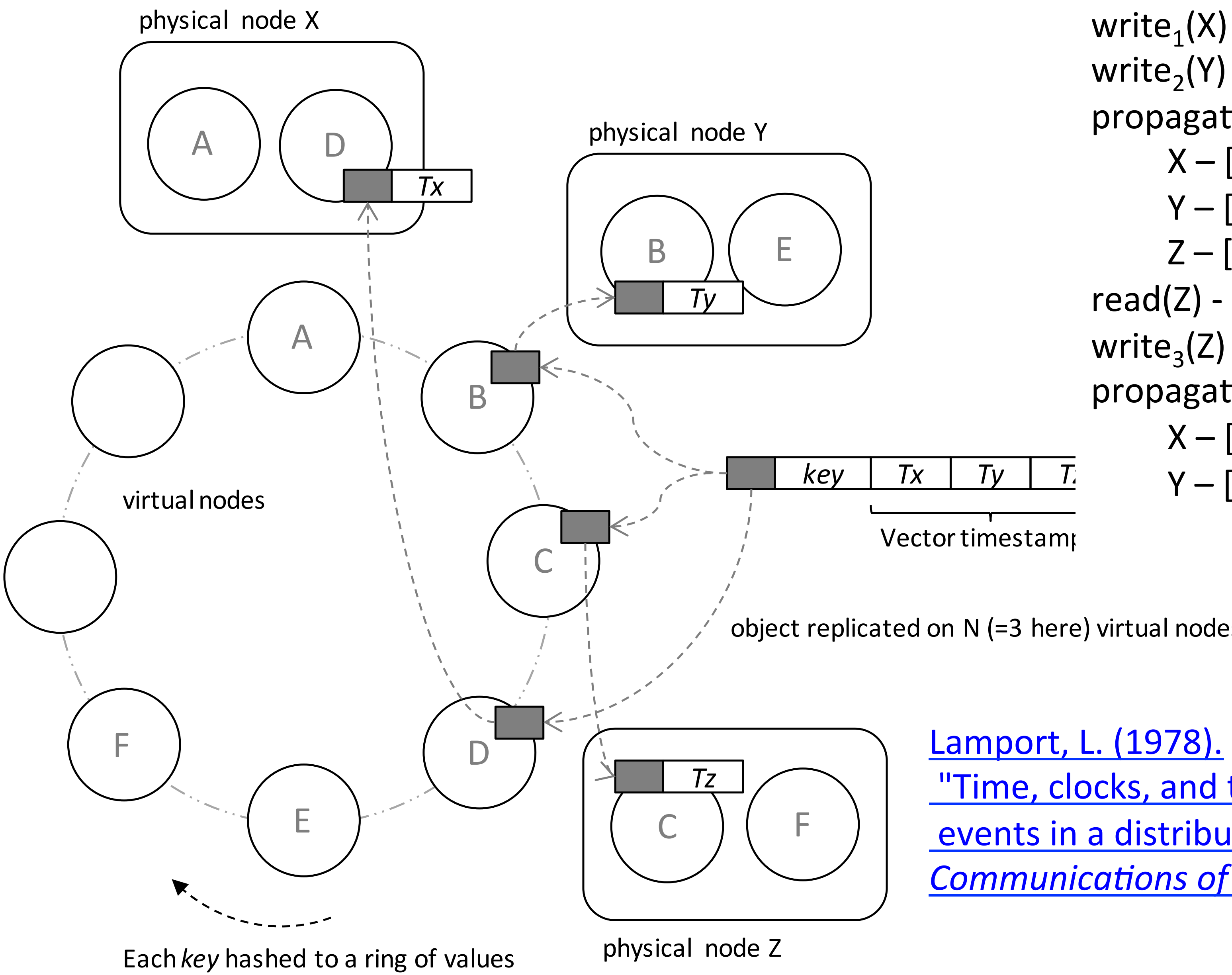
mongo DB




```
{
  title: "MongoDB",
  last_editor: "172.5.123.91",
  last_modified: new Date("9/23/2010"),
  body: "MongoDB is a...",
  categories: ["Database", "NoSQL", "Document Database"] ,
  reviewed: false
}
```

- documents
- shards
- indexes – incl. *text*
- map-reduce
    - (JavaScript)

eventual consistency



$write_1(X)$ – [1 0 0]
$write_2(Y)$ – [0 1 0]
propagation
X – [1 0 0], [0 1 0]
Y – [0 1 0], [1 0 0]
Z – [1 0 0], [0 1 0]
read(Z) - [1 0 0], [0 1 0]
$write_3(Z)$ – [1 1 1]
propagation
X – [1 1 1]
Y – [1 1 1]

Lamport, L. (1978). "Time, clocks, and the ordering of events in a distributed system". *Communications of the ACM*

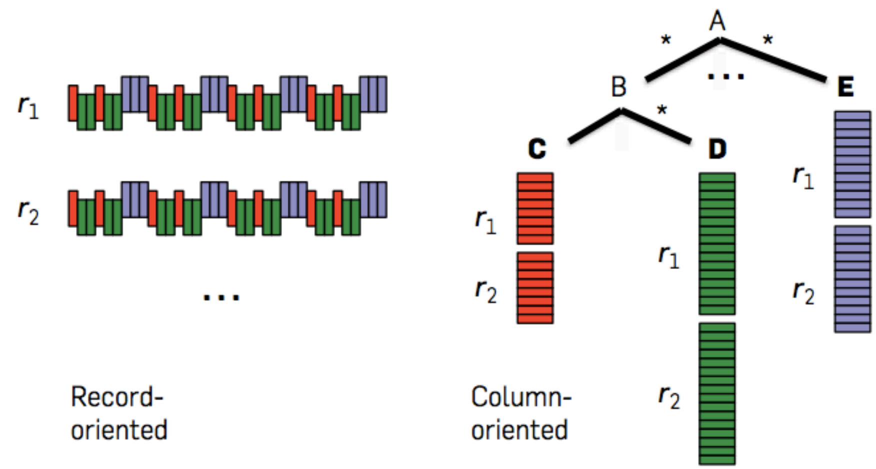
Dremel – new ‘kid’ on the block?

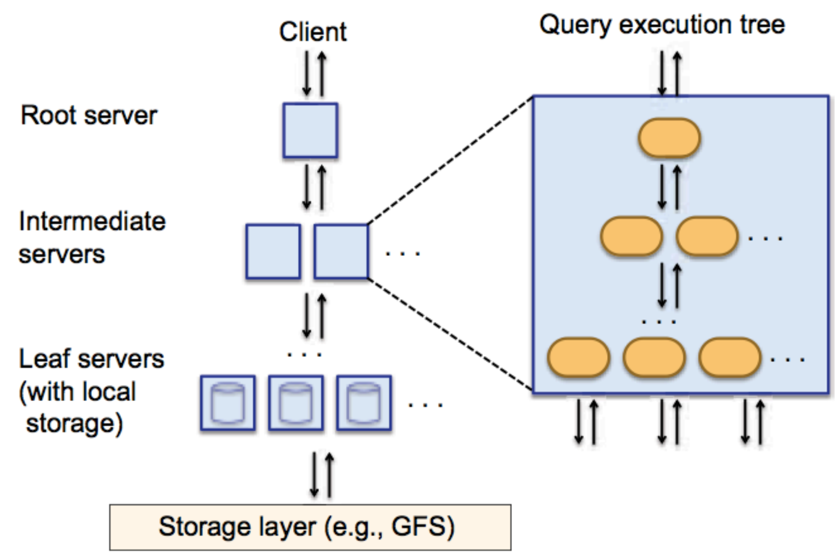
powers Google’s “BigQuery”

two important innovations:

- columnar storage for *nested*, possibly non-unique fields – *leaf servers*
- tree of *query servers* pass intermediate results from root to leaves and back

➢ *orders* of magnitude better than MR on petabytes of data – speed and storage





database evolution - summary

relational row-store: “one size fits all”; gigabytes

column-store data warehouse; terabytes

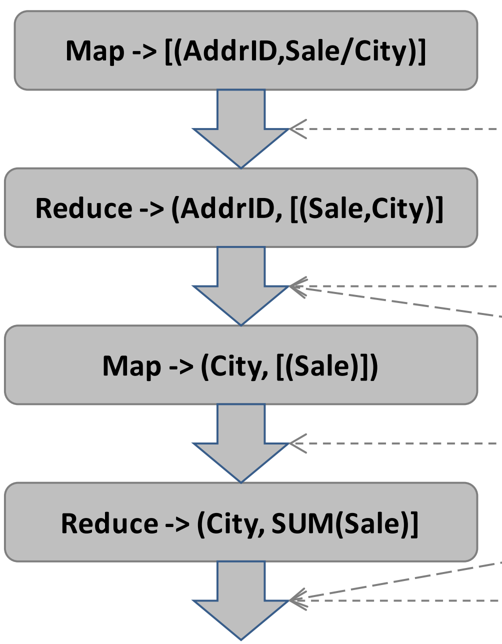
distributed noSQL row/column stores

+ map-reduce – for bulk analysis; 10s of terabytes

in-memory “one size fits all” databases; gigabytes

dremel – “one size fits all” for petabytes

SQL evolution: SQL-like MR coding



Pig Latin:

```
tmp  = COGROUP Sales  BY AddrID, Cities by AddrID
ioin = FOREACH tmp GENERATE FLATTEN(Sales), FLATTEN(Cities)
grp = GROUP join BY City
res = FOREACH grp GENERATE SUM(Sale)
```

HiveQL:

```
INSERT OVERWRITE TABLE join
SELECT s.Sale, c.City FROM Sales s
        JOIN  Cities c ON s.AddrID=c.AddrID;

INSERT OVERWRITE TABLE res
SELECT  SUM(join.Sale) FROM join GROUP BY join.City
```

SQL: SELECT SUM(Sale), City from Sales, Cities WHERE Sales.AddrID=Cities.AddrID GROUP BY City

SQL evolution: in-DB statistics, in parallel

```
sql> SELECT * FROM training;
 id | class | attributes
----+-------+------------
  1 |     1 | {1,2,3}
  2 |     1 | {1,2,1}
  3 |     1 | {1,4,3}
  4 |     2 | {1,2,2}
  5 |     2 | {0,2,2}
  6 |     2 | {0,1,3}
(6 rows)
```

```
sql> select * from toclassify;
 id | attributes
----+------------
  1 | {0,2,1}
  2 | {1,2,3}
(2 rows)
```

```
sql> SELECT madlib.create_nb_prepared_data_tables(
'training', 'class', 'attributes', 3, 'nb_feature_probs', 'nb_class_priors');
```

```
sql> SELECT madlib.create_nb_probs_view (
'nb_feature_probs', 'nb_class_priors', 'toclassify', 'id', 'attributes', 3, 'nb_probs_view_fast');

sql> SELECT * FROM nb_probs_view_fast;
 key | class | nb_prob
-----+-------+---------
   1 |     1 |     0.4
   1 |     2 |     0.6
   2 |     1 |    0.75
   2 |     2 |    0.25
(4 rows)
```

map-reduce evolution: iteration

many applications require repeated MR:

e.g. page-rank, continuous machine-learning …

1. iterate MR

   but make it more efficient: avoid data copy (HaLoop, Twister)

2. generalized data-flow graph of map->reduce tasks

   tasks are ‘blocking’ for fault-tolerance (Dryad/LINQ, Hyracks …)

3. direct implementation of recursion in MR

   how to recover from non-blocking tasks failing?

   graph model: (Pregel, Giraph)

   stream model: (S4)

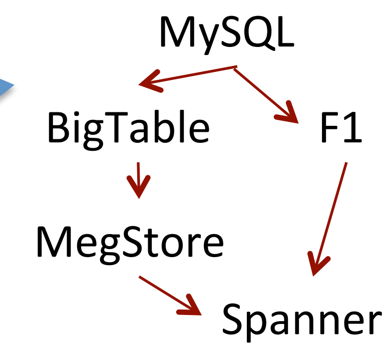
Enterprise Data Management and Big Data

Databases

designed for transaction processing

reporting and analytics came later: ‘afterthought’

(big data – exactly the reverse!)



Big Data Technologies

designed for analytics (counting, not queries)

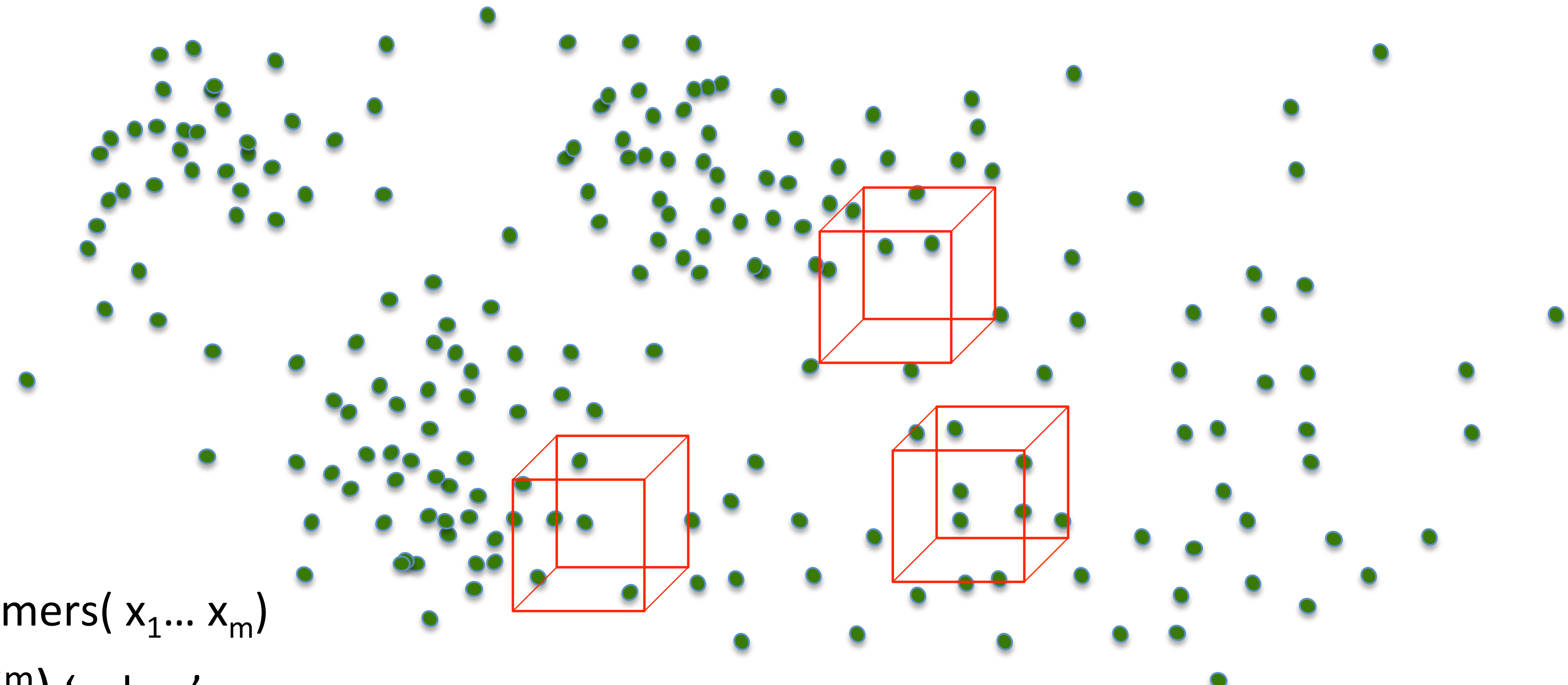
for retrieval – inverted-index (unstructured) search

data captured in the ‘raw’ (logs), no transactions

- ➢ price-performance advantage

even for ‘standard’ ETL and OLAP (in the bulk)

What about 'Business Intelligence' using SQL?



Customers($x_1 \ldots x_m$)

$O(\mathrm{d}^{2m})$ 'cubes'

for m=40, d=10 this becomes 10^{80} > # atoms in the universe

sampling $P(X)$ manually => *infinite time / infinite # people!*

so – BI folks need to learn analytics

<u>Big Data is about 'wide' data</u>

Analytics is about counting, not queries

Learning classifiers

computing likelihoods *P* (feature | class)

Clustering

computing hash-functions and distances

Data Mining

finding ‘bumps’: statistically ‘different’ regions

Learning latent features

sampling from distributions

(BN, CRF, MLN, LDA, Prob DB, ...)

matrix multiplications (NNMF, SVD)

Bottom line : you need to touch *all* your data

=> complex indexing for fast query processing is pretty useless

[Analytical] Database evolution & trends

noSQL databases
- no ACID transactions**
- *sharded* indexing
- restricted joins
- support columnar storage

in-memory data / databases
- real-time transactions
- variety of indexes, complex joins

challenge: *access patterns still matter*

relational row-store: "one size fits all"; gigabytes

column-store data warehouse; terabytes

distributed noSQL row/column stores

+ map-reduce – for bulk analysis; 10s of terabytes

in-memory "one size fits all" databases; gigabytes

Dremel (Impala*) – "one size fits all" for petabytes

for those who still want fast aggregation queries!

*Cloudera 2012

summary

- distributed files – 2^{nd} basic element of big-data
- what databases are good for
  - and why traditional DBs were a happy compromise
- evolution of databases
- evolution of SQL
- evolution of map-reduce
- ‘big picture’ for database evolution

Next week:

➢ Lecture by Srikanta Bedathur on Graph Databases

➢ following week (unit 5): Learn: ‘facts’ from data