DBM1
Part 5: Distributed databases

Vincent Primault

vincent.primault@insa-lyon.fr

Course outline

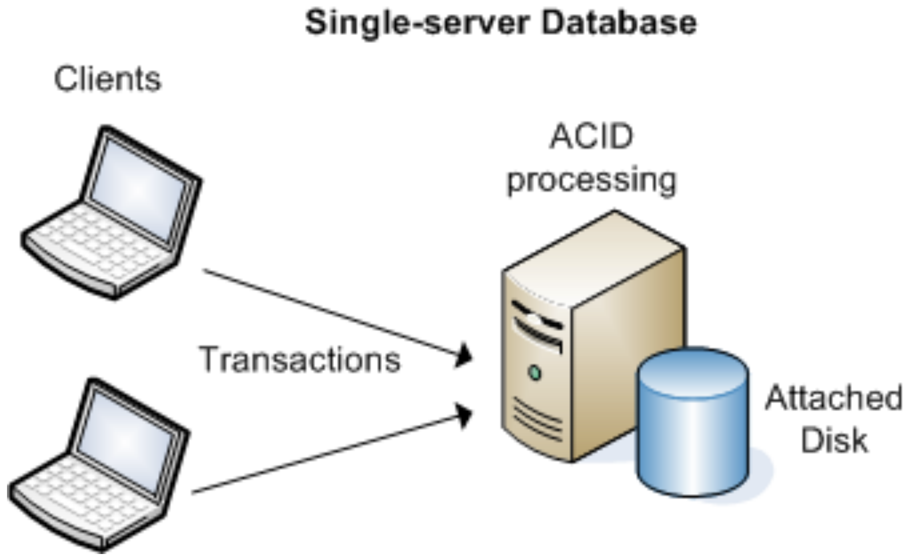
- ~~Databases fundamentals~~ **Done!**
- ~~Relational algebra~~ **Done!**
- ~~SQL language~~ **Done!**
- ~~Database internals~~ **Done!**
- Distributed databases **Today**

Sources of this lecture

- Stanford, CS347 – Parallel and distributed data management
- Dr Brian Cooper

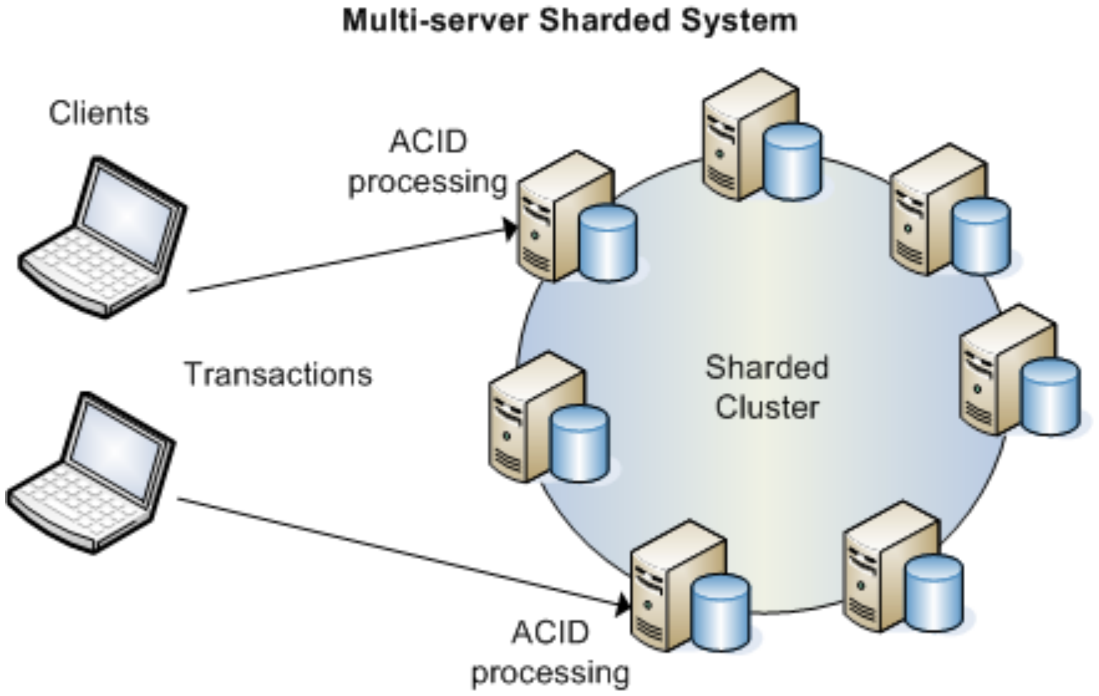
- Database System Concepts, 5th edition
- Prof. Avi Silberschatz, Dr Henry F. Korth, Prof. S. Sudarshan

Client/server architecture



Distributed systems

- Data is spread across multiple machines (or nodes).
- **Network** interconnects machines.



Trade-offs in distributed systems

- Sharing data: users at one node able to access the data residing at some other nodes.
- Autonomy: each node is able to retain a degree of control over data stored locally.
- Replication: data can be replicated at remote nodes, and system can function even if a node fails.
- Scalability: it becomes possible to handle really huge amounts of data.
- But added complexity to ensure proper coordination among nodes.
  - Software development cost.
  - Greater potential for bugs.
  - Increased processing overhead.

Distributed data storage

DBM1 – Part 5: Distributed databases

Distributed data storage

- Replication: system maintains multiple copies of data, stored in different nodes, for faster retrieval and fault tolerance.
- Fragmentation: relation is partitioned into several fragments stored in distinct nodes.
- Replication and fragmentation can be combined. A relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

Data replication

- A relation or fragment of a relation is replicated if it is stored redundantly in two or more nodes.
- Full replication of a relation is the case where the relation is stored at all nodes.
- Fully redundant databases are those in which every node contains a copy of the entire database.

Data replication (cont'd)

- Advantages
  - Availability: failure of node containing relation r does not result in unavailability of r is replicas exist.
  - Parallelism: queries on r may be processed by several nodes in parallel.
  - Reduced data transfer: relation r is available locally at each node containing a replica of r.
- Disadvantages
  - Increased cost of updates: each replica of relation r must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
  - One solution: choose one copy as primary copy and apply concurrency control operations on primary copy.

Master/slave model

- One particular case of data replication.
- One node is elected as the master and is the authoritative source.
- Other nodes (slaves) are synchronized with the master.
- Writes are done against the master, and then propagated to slaves.
- Reads can be done against master or slaves.

- Efficient for read intensive applications.

Data fragmentation (or sharding)

- Division of a relation *r* in fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation *r*.
- Horizontal fragmentation: each tuple of r is assigned to one or more fragments.
- Vertical fragmentation: the schema of relation r is split into several smaller schemas.
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.
- Vertical and horizontal fragmentation can be mixed.

Vertical fragmentation

id	title	year
ad34r09	Casino Royale	2006
f45gha4	Quantum of Solace	2008
902b3cc	Skyfall	2012

$\text{Shows}_1 = \Pi_{\text{id,title,year}}(\text{Shows})$

id	kind	suspended
ad34r09	movie	0
f45gha4	movie	0
902b3cc	movie	0

$\text{Shows}_2 = \Pi_{\text{id,kind,suspended}}(\text{Shows})$

Advantages of vertical fragmentation

- Also called partitioning or sharding.
- Allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed.
- Tuple-id attribute allows efficient joining of vertical fragments.
- Allows parallel processing on a relation.

Desired properties

- Completeness: each attribute is present in at least one fragment.
- Lossless join: from the natural join of fragments, it is possible to reconstruct the entire relation.

- Match access patterns: if two attributes are frequently accessed together, they should be placed in the same fragment.

Horizontal fragmentation

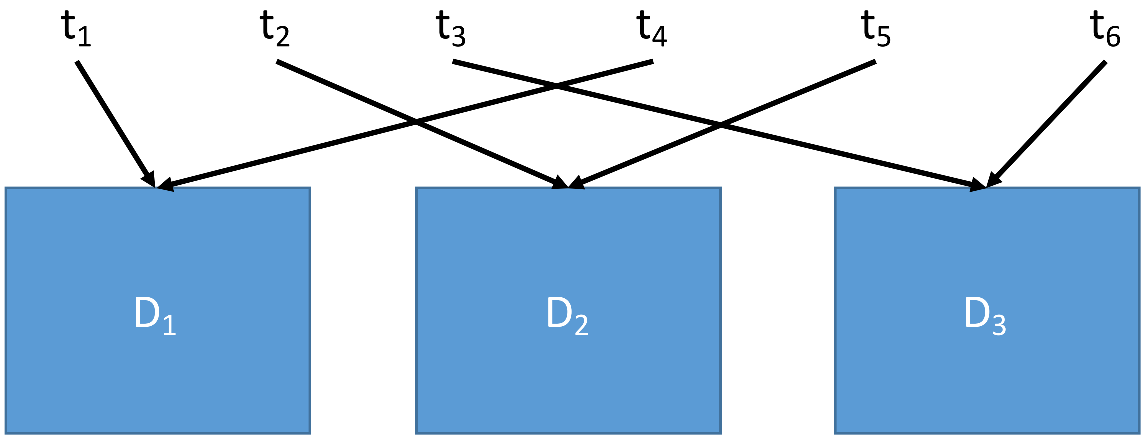
id	title	year	kind	suspended
ad34r09	Casino Royale	2006	movie	0
f45gha4	Quantum of Solace	2008	movie	0

$$\text{Shows}_1 = \sigma_{\text{year >= 2000 AND year < 2010}}(\text{Shows})$$

id	title	year	kind	suspended
902b3cc	Skyfall	2012	movie	0

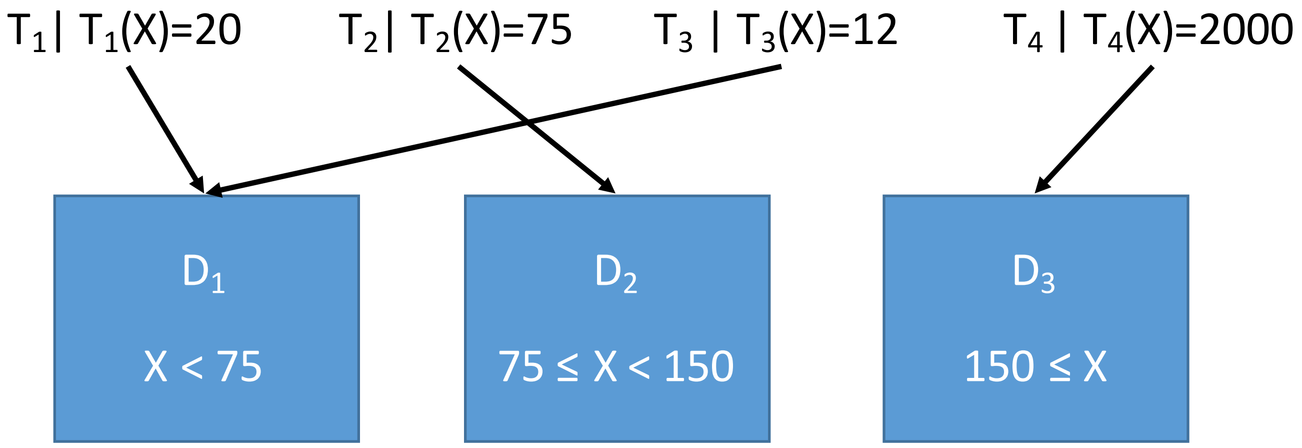
$$\text{Shows}_2 = \sigma_{\text{year >= 2010}}(\text{Shows})$$

Round robin partitioning



- Evenly distributes data.
- Good for scanning full relation.
- Not good for point or range queries.

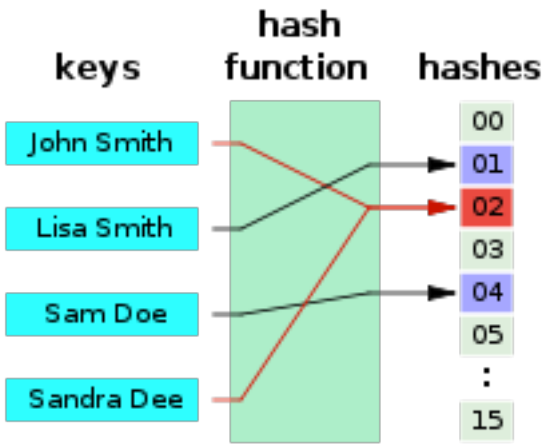
Range-based partitioning



- Good for <u>some</u> range queries on X.
- Need to select a good vector to have a balanced distribution. Else data will be skewed and execution will get no benefit.

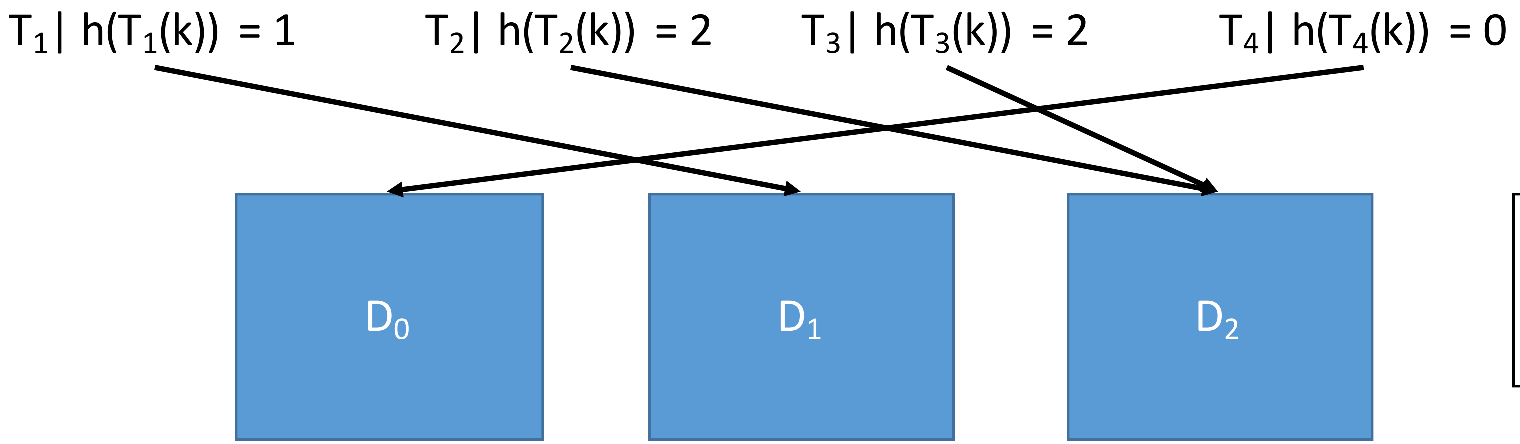
Reminder: hash functions

- A hash function is any function that can be used to map data of arbitrary size to data of fixed size.
- The values returned by a hash function are calledhash values, hash codes, hash sums, or simply hashes.



A hash function that maps names to integers from 0 to 15.
There is a collision between keys "John Smith" and "Sandra Dee".

Hash-based partitioning.



Example of a hash function if k is an integer:

$$h(k) = k \% 3$$

- Evenly distributed data (if hash function is good...).
- Goot for point queries on the key and for joins.
- Not good for range queries and point queries not on the key.

Advantages of horizontal fragmentation

- Allows parallel processing on fragments of a relation.
- Allows a relation to be split so that tuples are located where they are most frequently accessed.
- Better performance because of partition pruning (in range-based).

How to choose a good fragmentations?

- $Shows_1 = \sigma_{year < 1990}(Shows), Shows_2 = \sigma_{year \geq 2000}(Shows)$

$\Rightarrow$ Some tuples are lost.

- $Shows_1 = \sigma_{year < 2000}(Shows), Shows_2 = \sigma_{year \geq 1995}(Shows)$

$\Rightarrow$ Tuples with $1995 \leq year < 2000$ are duplicated.

- Prefer to deal with replication explicitely.
  - $Shows_1 = \sigma_{year < 1995}(Shows)$, $Shows_2 = \sigma_{year \geq 1995}(Shows)$
  - $Shows_2$ is replicated on two different nodes.

Desired properties

- Completeness: each tuple is present in at least one fragment.
- Disjointness: each tuple is present in at most one fragment.
- Reconstruction: from the union of fragments, it is possible to reconstruct the entire relation.

- Match access patterns: tuples which are frequently accessed together should be placed in the same fragment.

Distributed query processing

- Added complexity compared to single-host query processing.
- Need to transfer potentially large amounts of data between nodes.
- There is an optimization problem about the placement of fragments
  - minimize latency
  - maximize throughput
  - mininize data transfer...

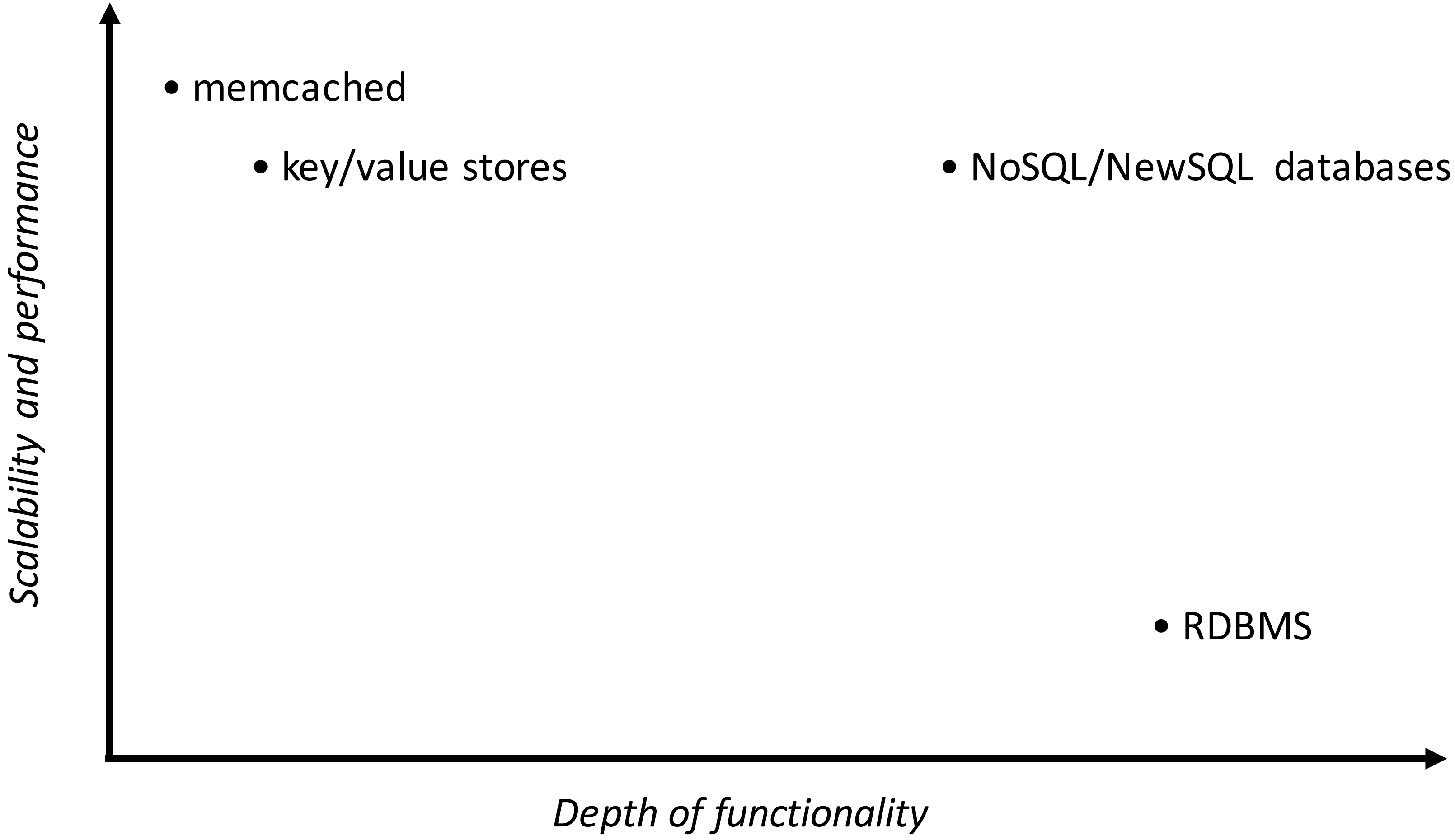
Synthesis

- In modern real-life applications, fragmentation and replication must be implemented to allow servers to sustain the load and deliver results in real-time.
- Several ways to fragment a database.
- Methods exist to optimize the fragmentation, but the optimization problem stays hard.

Document oriented databases

DBM1 – Part 5: Distributed databases

Databases scope



Document oriented database

Relational world

- Database
- Table
- Tuples

Documents world

- Database
- Collection
- Document/object

Document oriented database (cont'd)

- MondoDB is a schemaless database.
- A unique object identifier is automatically assigned to each document (acts as a primary key).
- Secondary indexes can still be created
- Collections do not have to be created (they are automatically when the first record is inserted).

From tuples to documents

Tuple

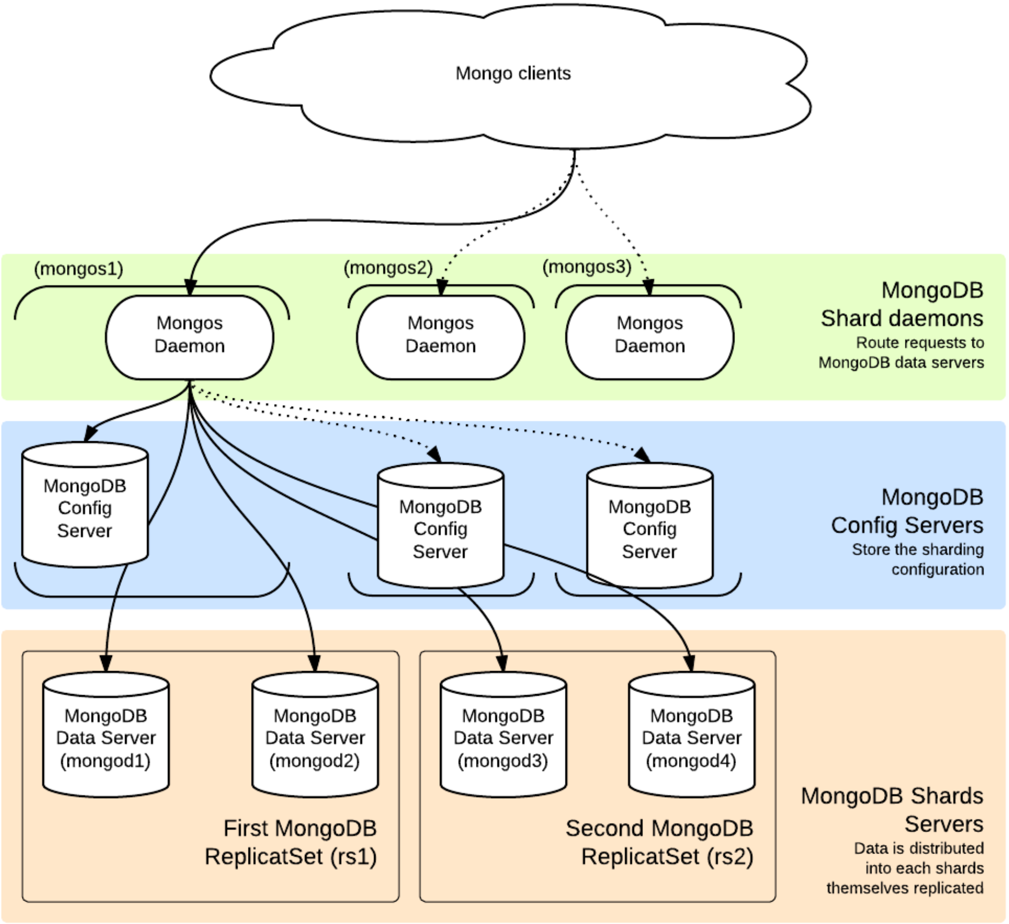
id	title	year	suspended	kind
ad34r09	Casino Royale	2006	0	movie

Document

```
{
    "id": "ad34r09",
    "title": "Casino Royale",
    "year": 2006,
    "suspended": false,
    "kind": "movie"
}
```

MongoDB architecture

- MongoDB uses an architecture with multiple nodes leveraging horizontal fragmentation.
- Sharding is automatic, i.e., built-in on the database side.
- Range-based, hash-based or user provided sharding strategies.



MongoDB query language

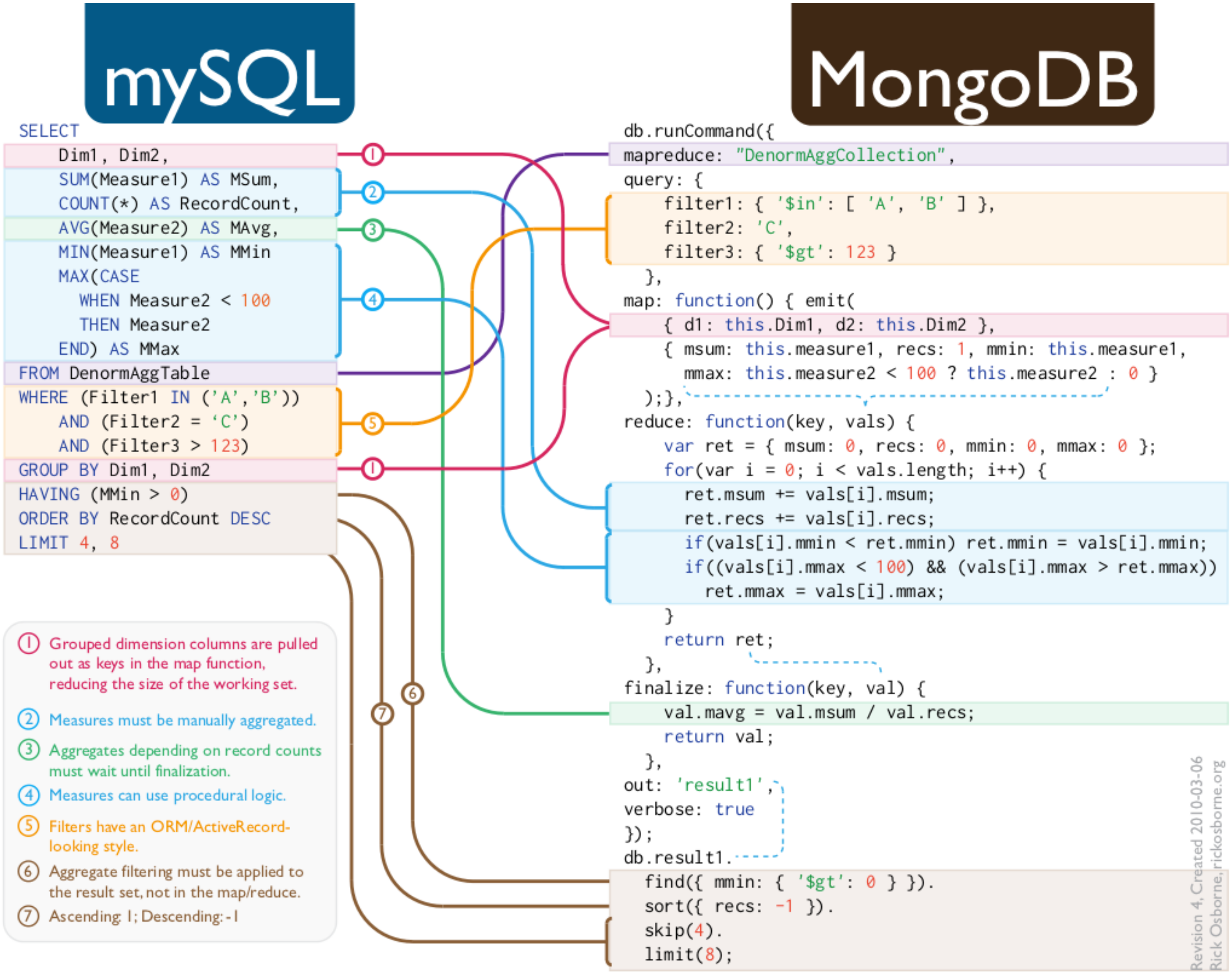
SQL

```
SELECT user_id, status
FROM users
WHERE status = "A"

SELECT *
FROM users
WHERE status = "A" OR age > 50
ORDER BY user_id DESC
```

MongoDB

```
db.users.find(
   { status: "A" },
   { user_id: 1, status: 1, _id: 0 }
)

db.users.find( {
   $or: [
      { status: "A" } ,
      { age: { $gt: 50 } }
   ]
} ).sort( {user_id: -1} )
```

MongoDB query language (cont'd)



MapReduce

DBM1 – Part 5: Distributed databases

MapReduce

- MapReduce is a programming model for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- Name comes from the paper of Google in OSDI 2014.
- Builds on two functions: *map()* and *reduce()*.

- Several open-source implementations.

Map Reduce

- Input: $R = \{r_1, r_2, \ldots r_n\}$, functions Map, Reduce
  - $\mathrm{Map}(r_i) \rightarrow \{ [k_1, v_1], [k_2, v_2], .. \}$
  - $\mathrm{Reduce}(k_i, vals) \rightarrow [k_i, vals']$
- Let $S = \{ [k, v] \mid r_i \in R, [k, v] = \mathrm{Map}(r_i) \}$
- Let $K = \{ k \mid [k,v] \in S \}$
- Let $G(k) = \{ v \mid [k, v] \in S \}$
- Output = $\{ [k, T] \mid k \in K, T = \mathrm{Reduce}(k, G(k)) \}$

S is bag

G is bag

Example: counting word occurrences

```
map(String doc, String value) {
    // doc is document name
    // value is document content
    for each word w in value:
        EmitIntermediate(w, “1”);
}
```

Example:

map(doc, “cat dog cat bat dog”) emits
[cat 1], [dog 1], [cat 1], [bat 1], [dog 1]

Example: counting word occurrences (cont'd)

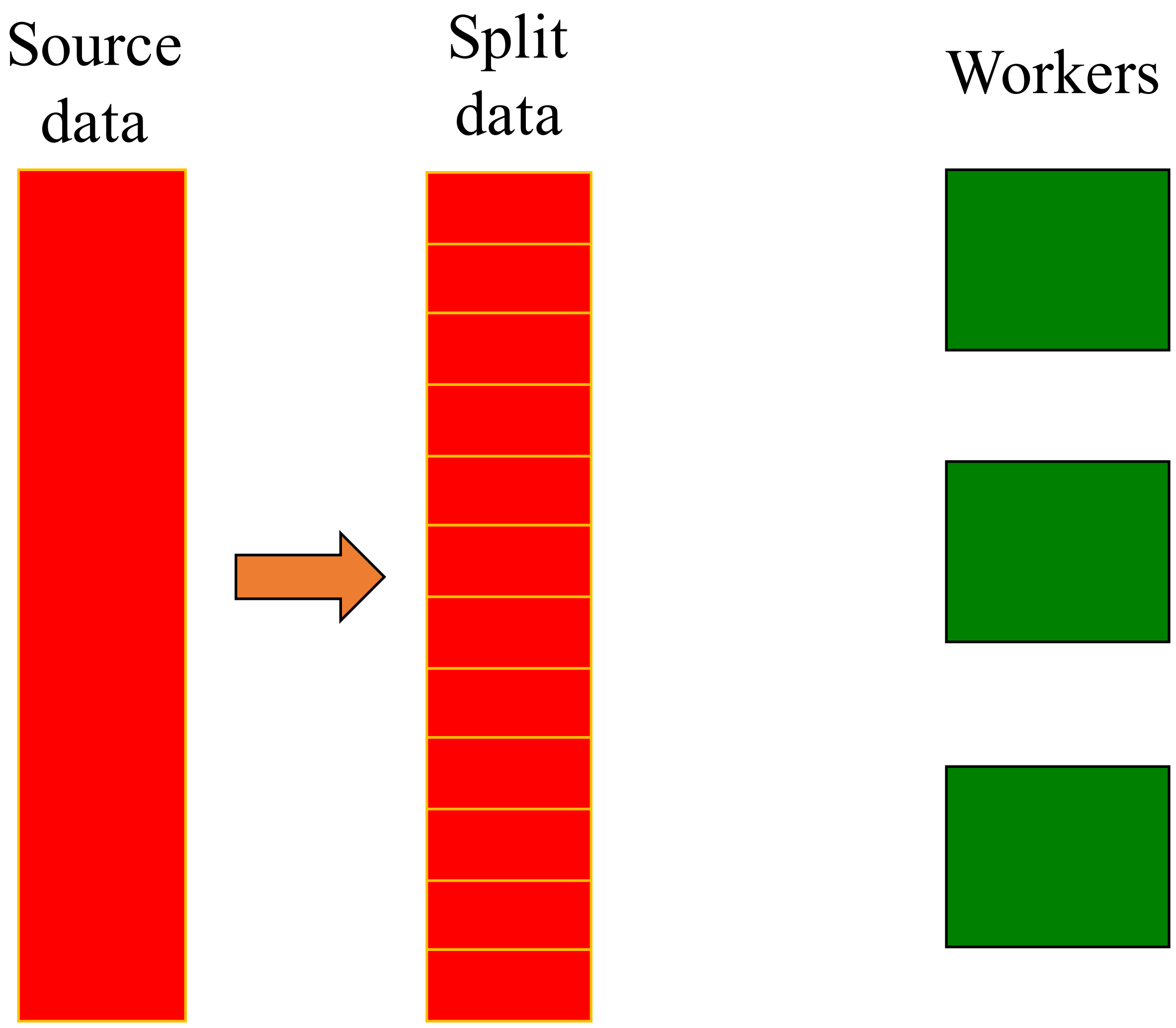
```
reduce(String key, Iterator values) {
    // key is a word
    // values is a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v)
    Emit(AsString(result));
}
```

Example:

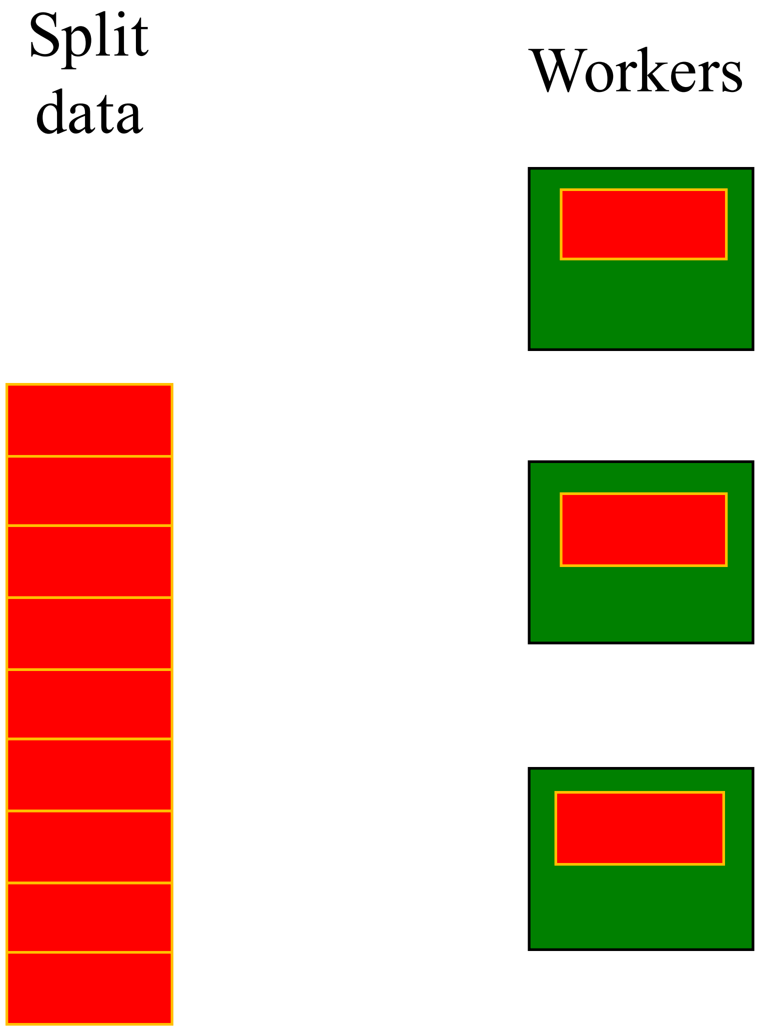
reduce("dog", "1 1 1 1") emits "4"

Becomes ("dog", 4)

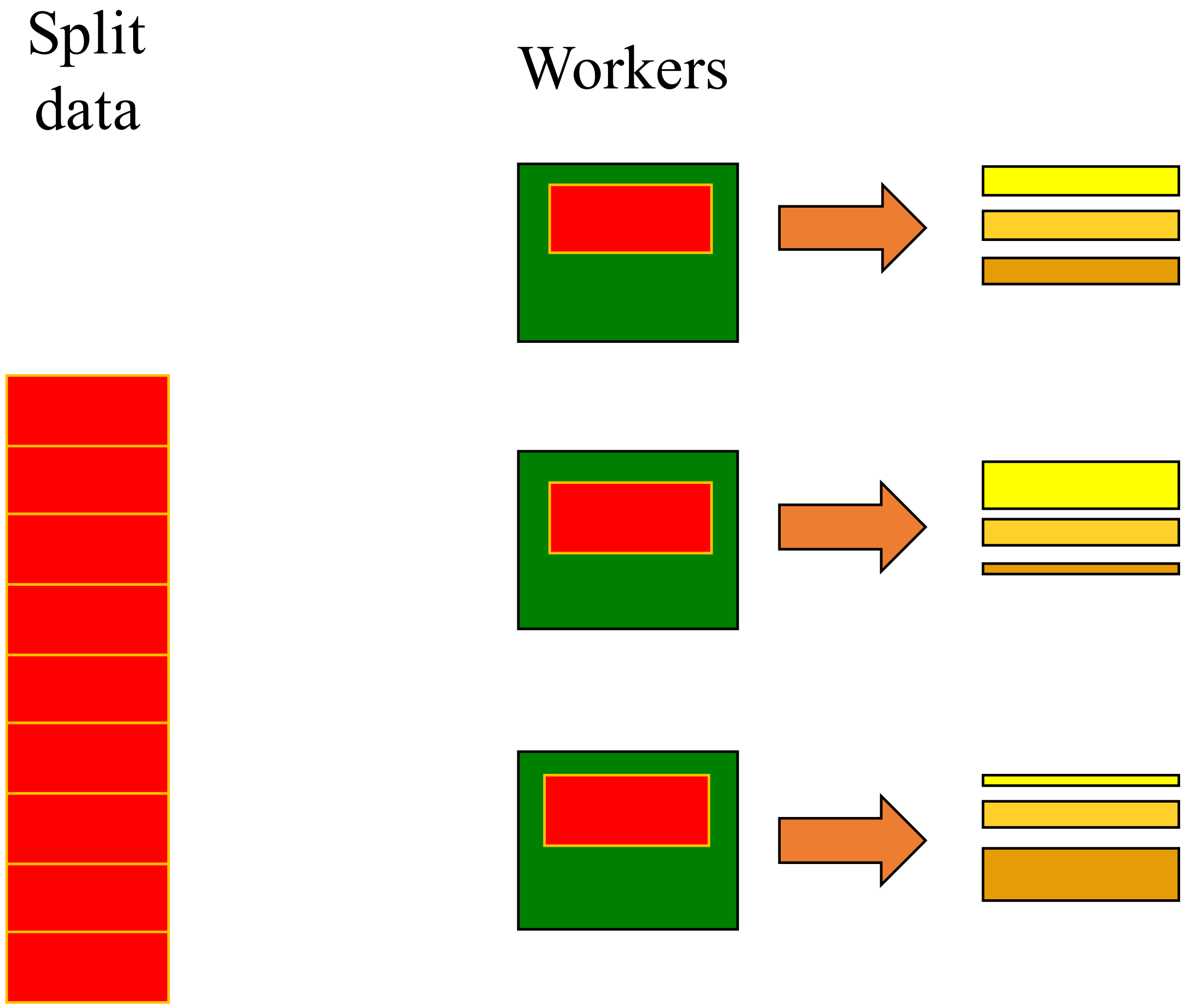
Mappers



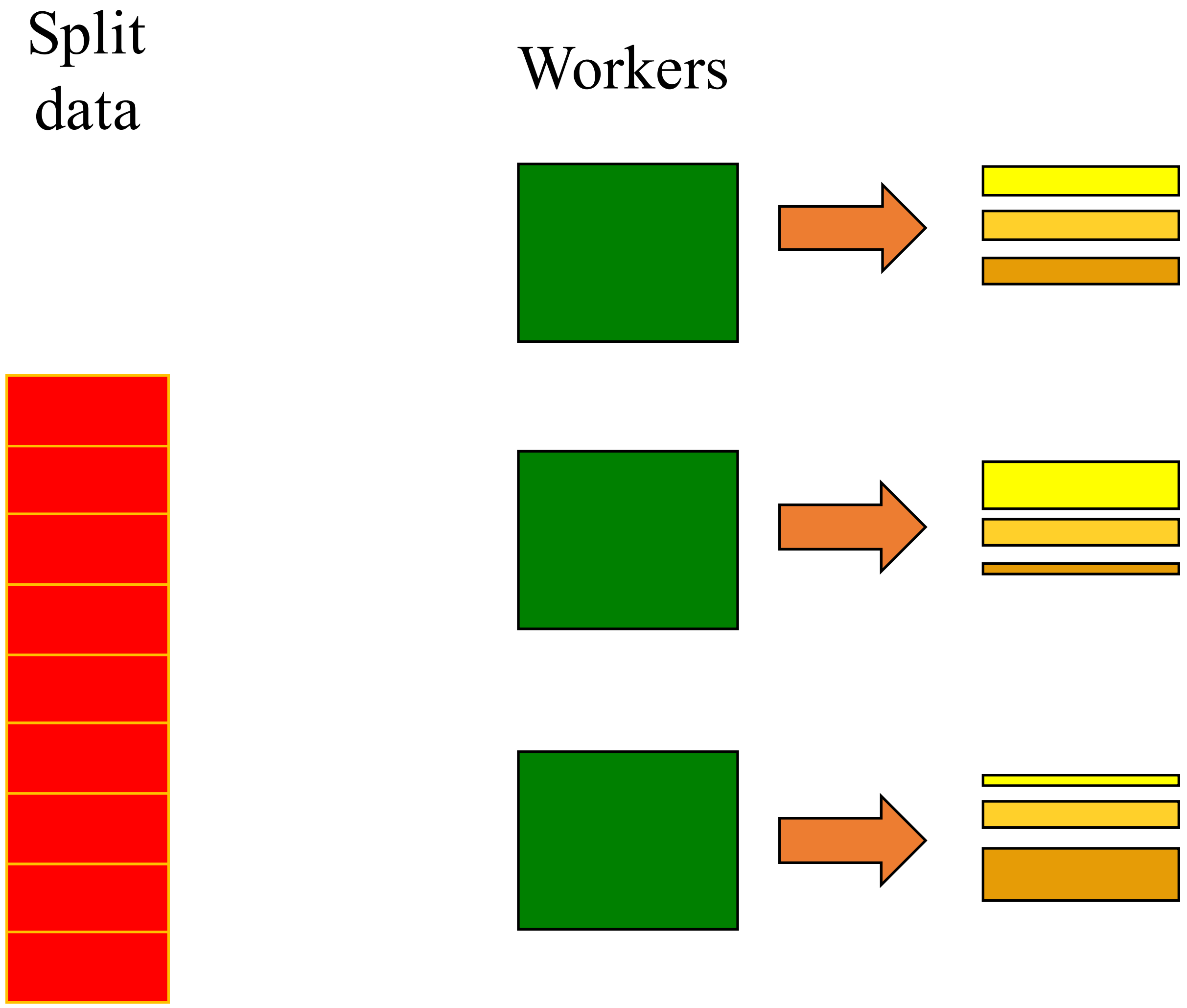
Mappers



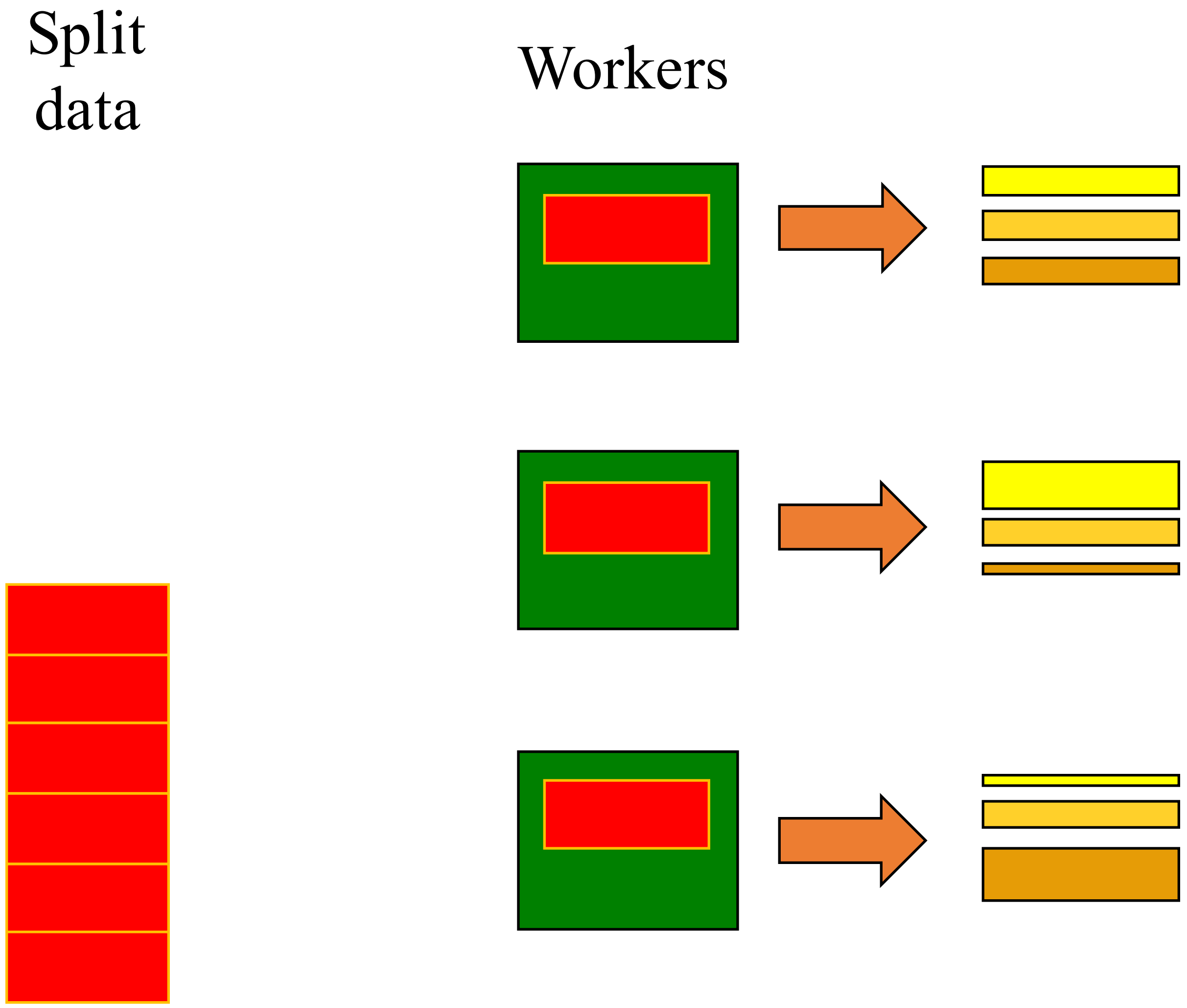
Mappers



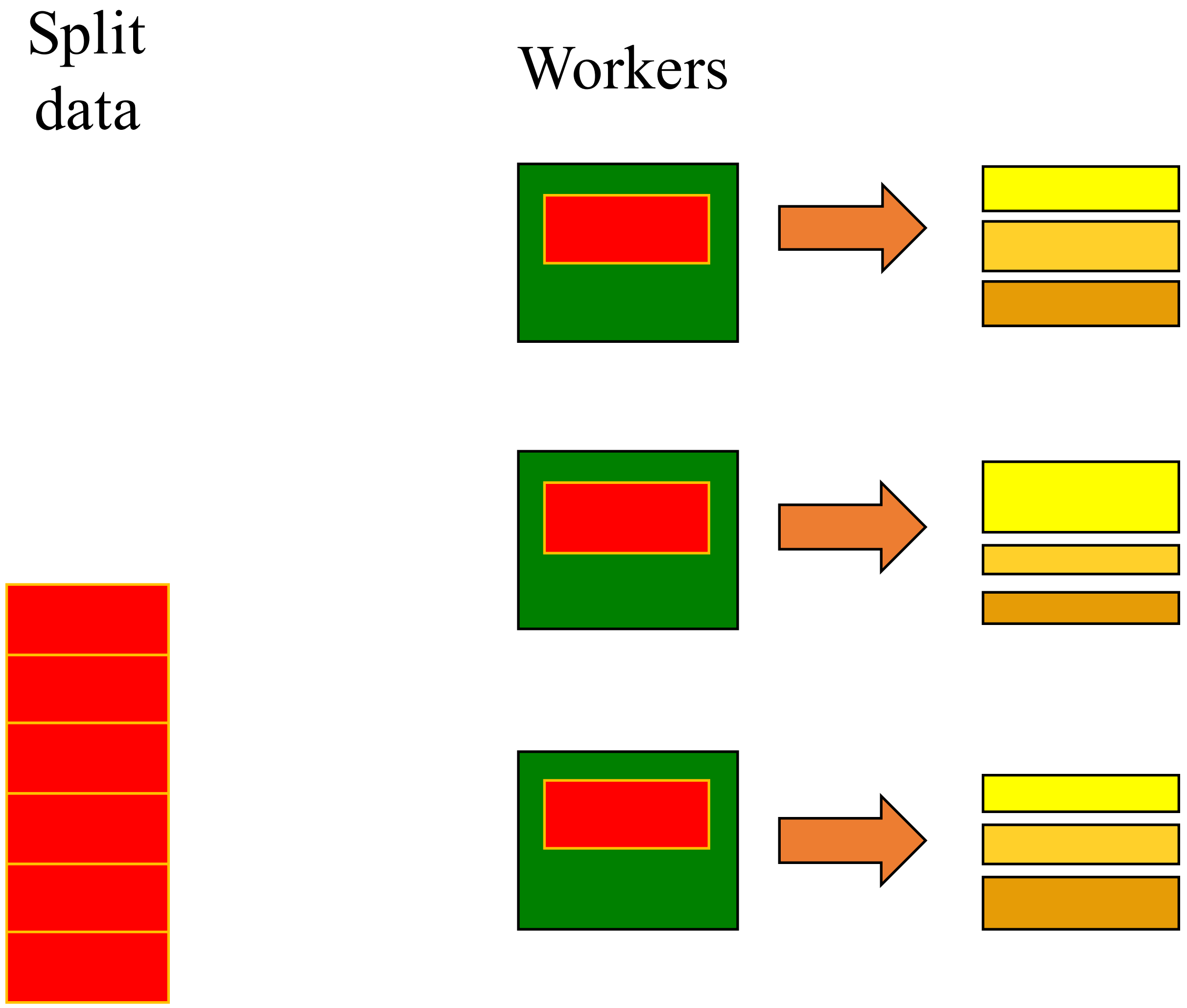
Mappers



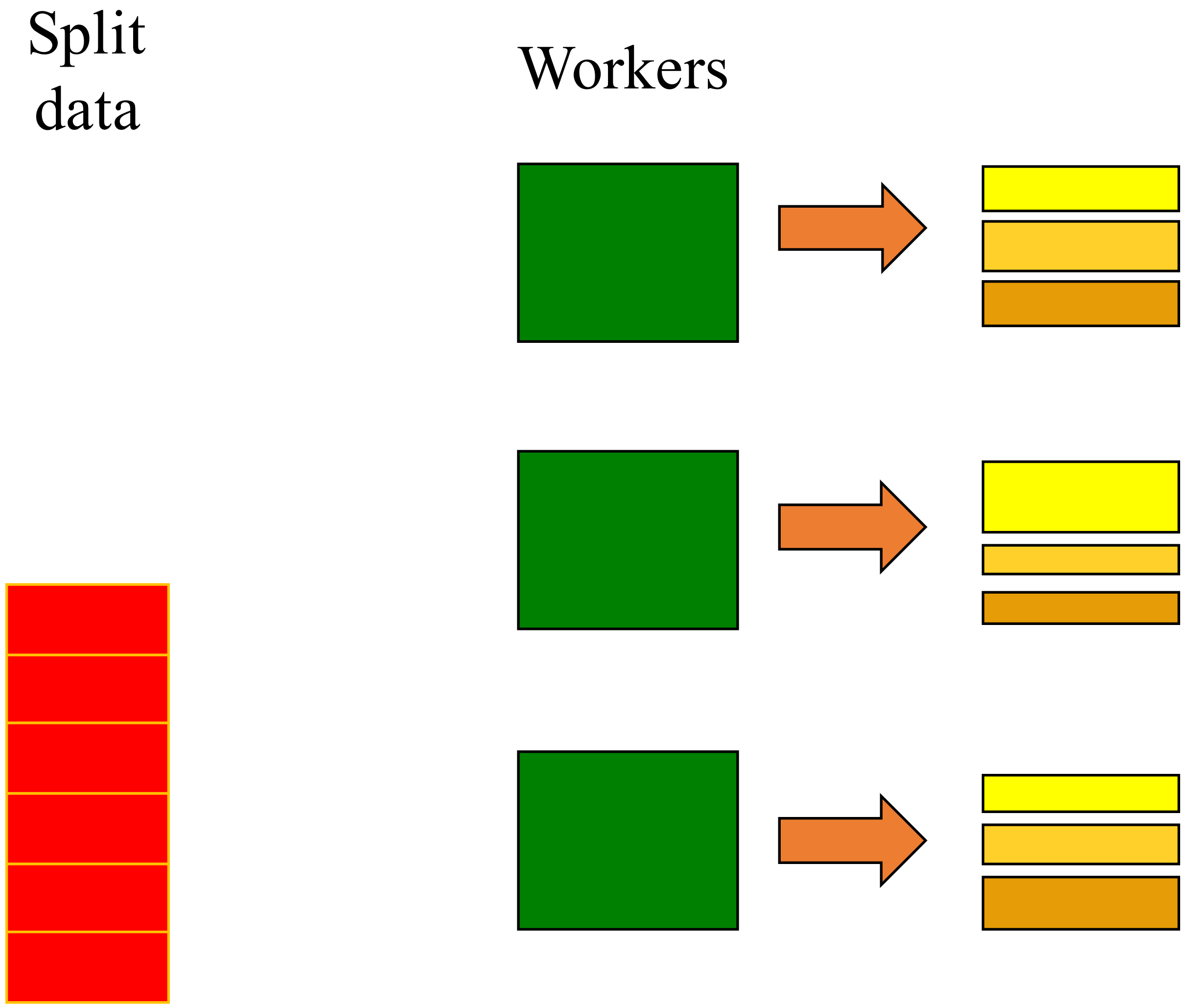
Mappers



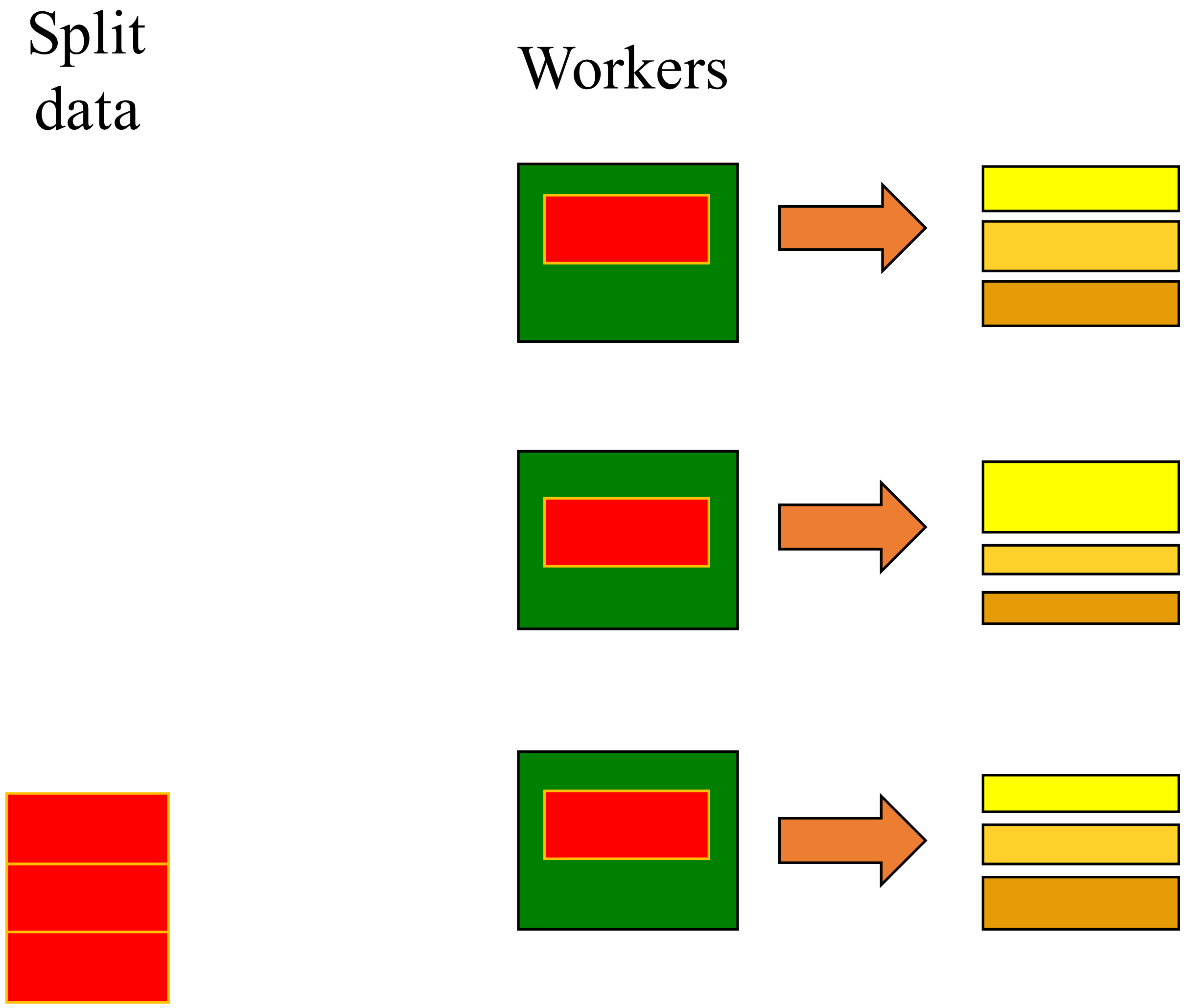
Mappers



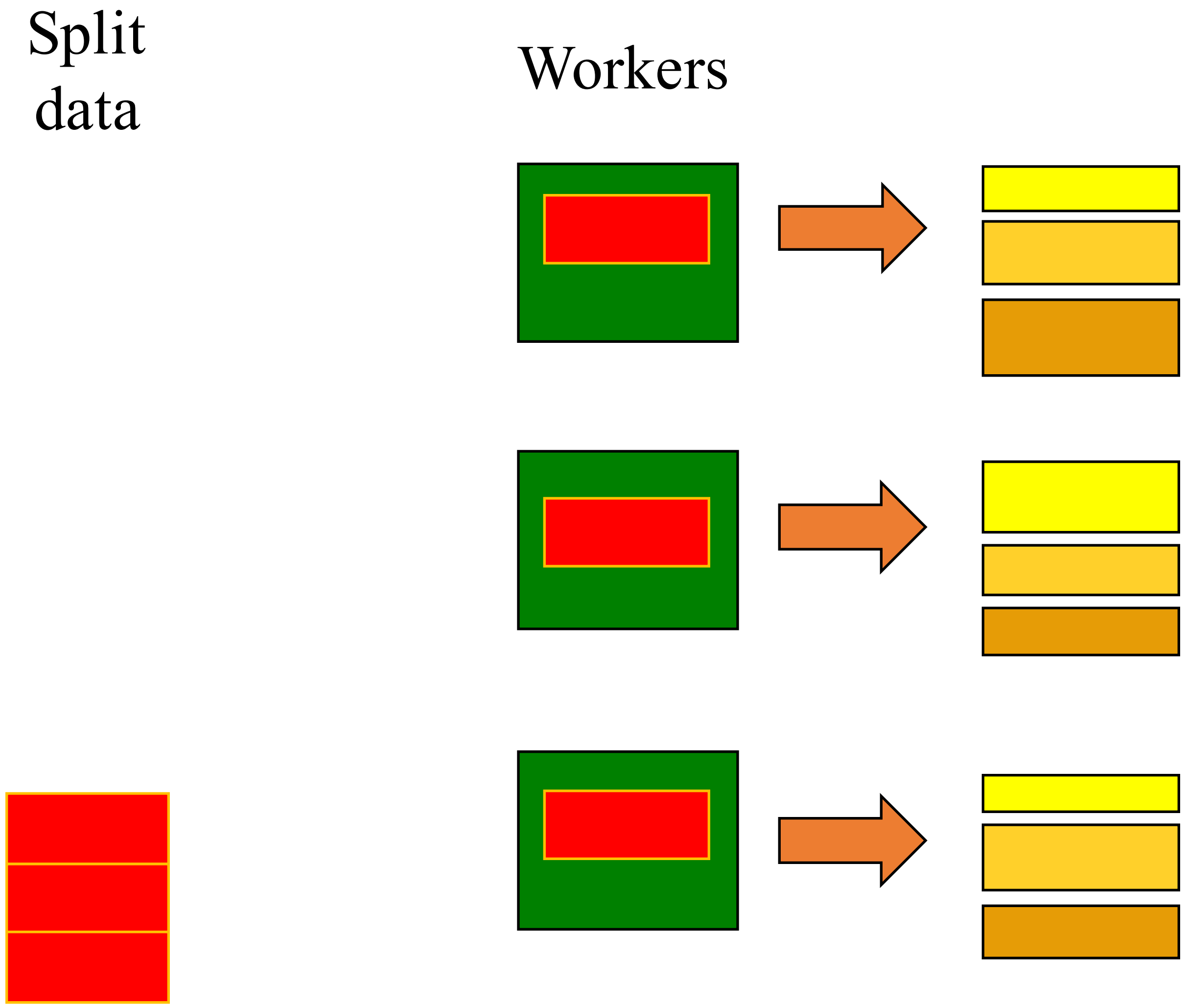
Mappers



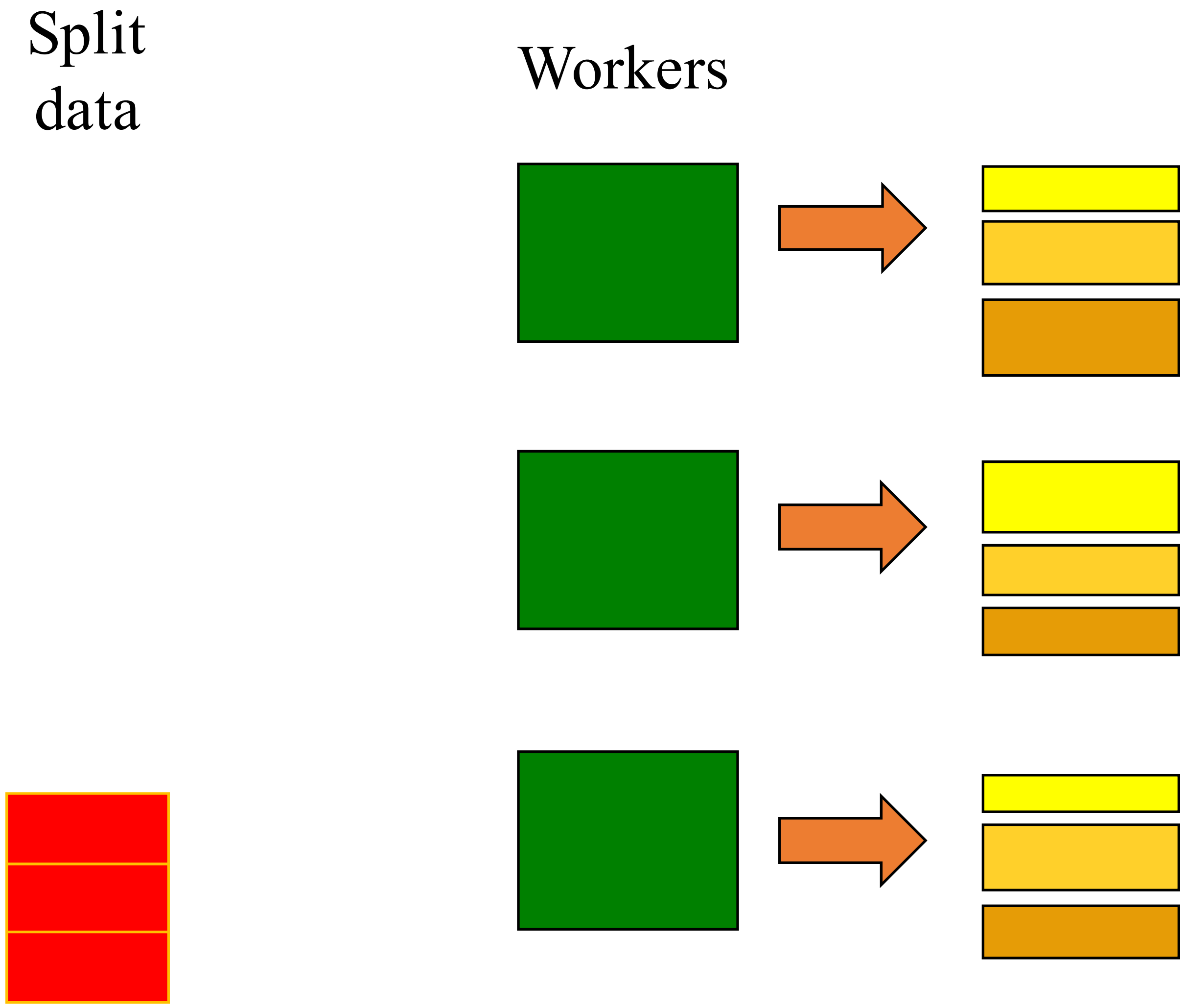
Mappers



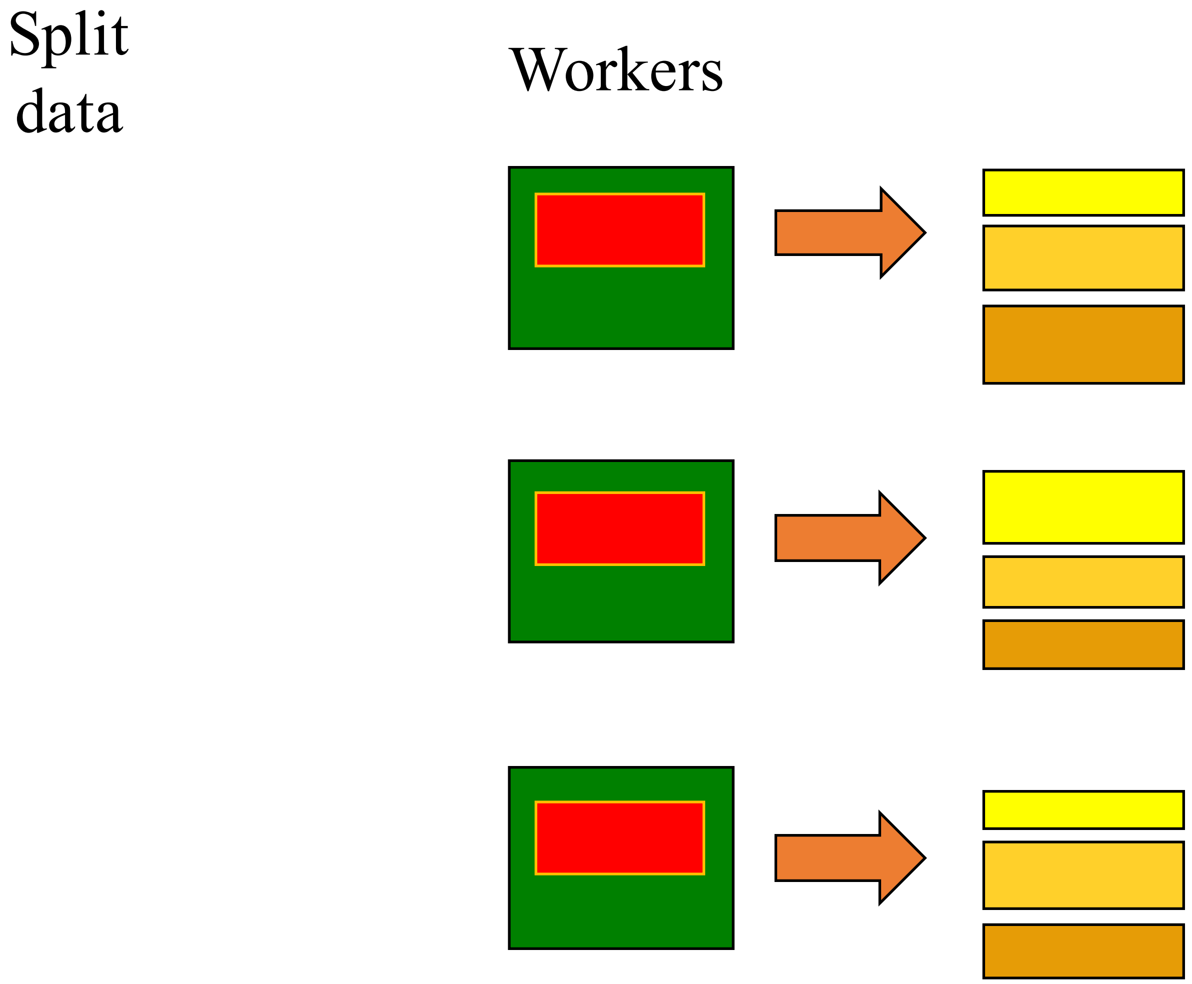
Mappers



Mappers

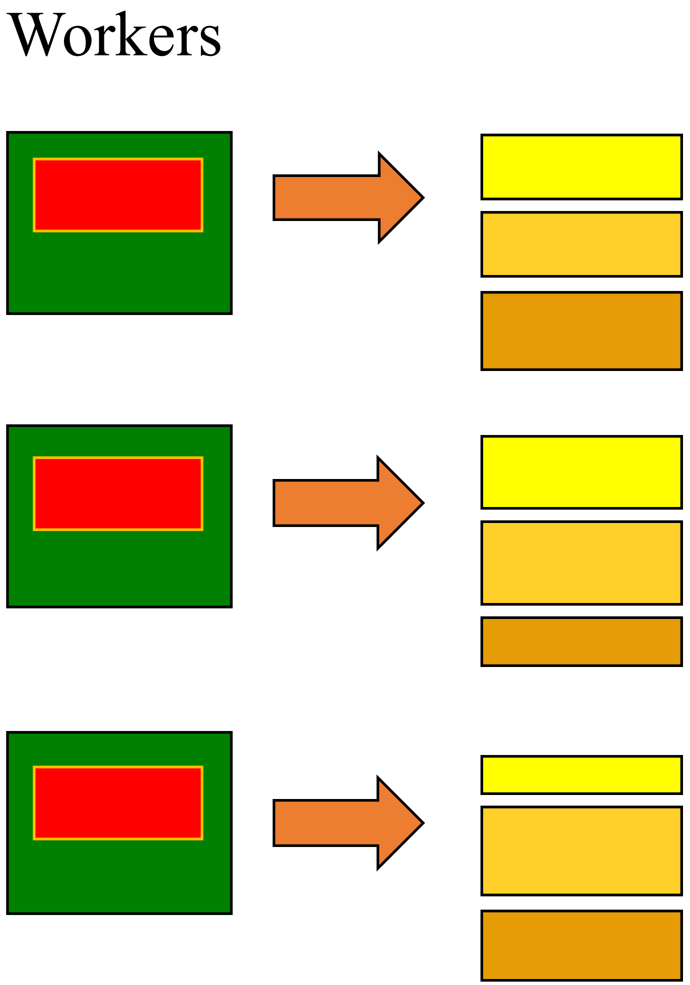


Mappers

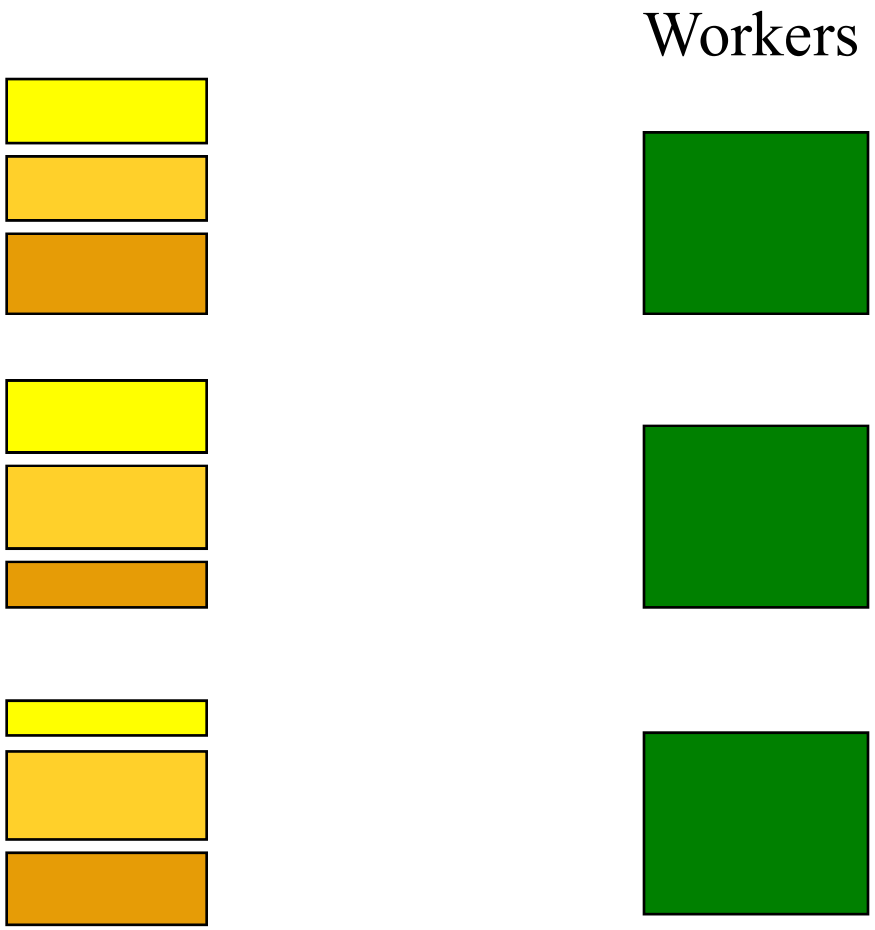


Mappers

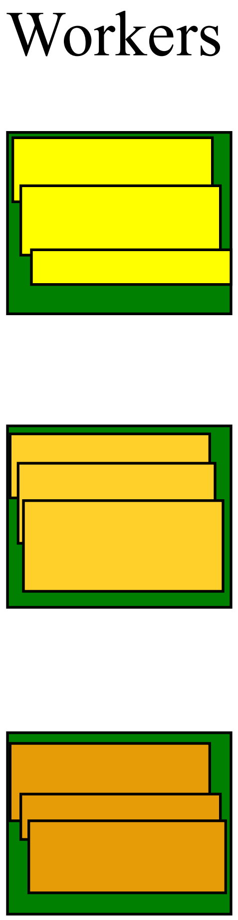
Split
data



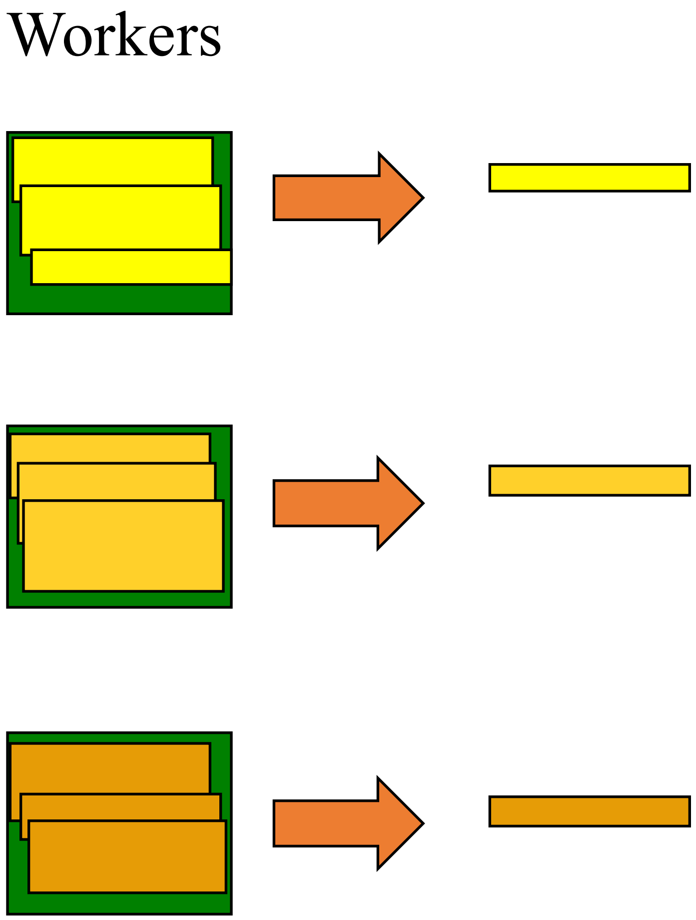
Shuffle



Shuffle



Reduce



Another way to think about it

- Mapper: (some query)
- Shuffle: **GROUP BY**
- Reducer: **SELECT** Aggregate()

- But, data does not have to be relational
  - Mapper: parse data into K,V pairs.
  - Shuffle: repartition by K.
  - Reducer: transform the V's for a K into a V_{final}.

Analysis

- Claimed advantages:
  - Model easy to use, hides details of parallelization, fault recovery.
  - Many problems expressible in MapReduce framework.
  - Scales to thousands of machines.
- Possible disadvantages:
  - 1-input 2-stage data flow rigid, hard to adapt to other scenarios.
  - Custom code needs to be written even for the most common operations, e.g., projection and filtering.
  - Opaque nature of map, reduce functions impedes optimization.

Hadoop

- Open-source Map-Reduce system.
- Also, a toolkit
  - HDFS – filesystem for Hadoop
  - HBase – Database for Hadoop
- Also, a huge ecosystem
  - Tools
  - Recipes
  - Developer community

Make it database-ish?

- Simple idea: each operator is a MapReduce flow.
- How to do:
  - Select
  - Project
  - Group by, aggregate
  - Join

- There are platforms for SQL-like queries on Hadoop: Pig Latin, Hive…

Why not just use a DBMS?

Many DBMSs exist and are highly optimized.

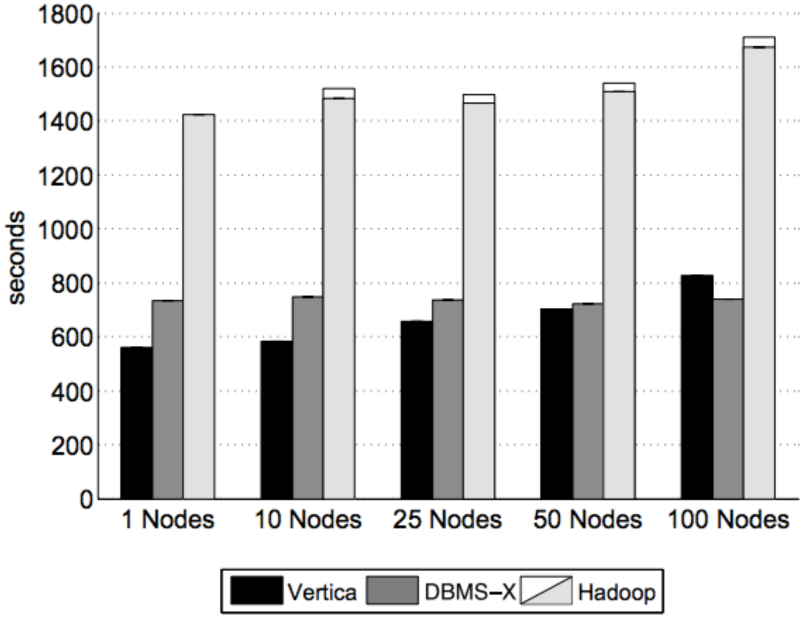


Figure 7: Aggregation Task Results (2.5 million Groups)

A comparison of approaches to large-scale data analysis. Pavlo et al, SIGMOD 2009.

Why not just use a DBMS? (cont'd)

One reason: loading data into a DBMS is hard.

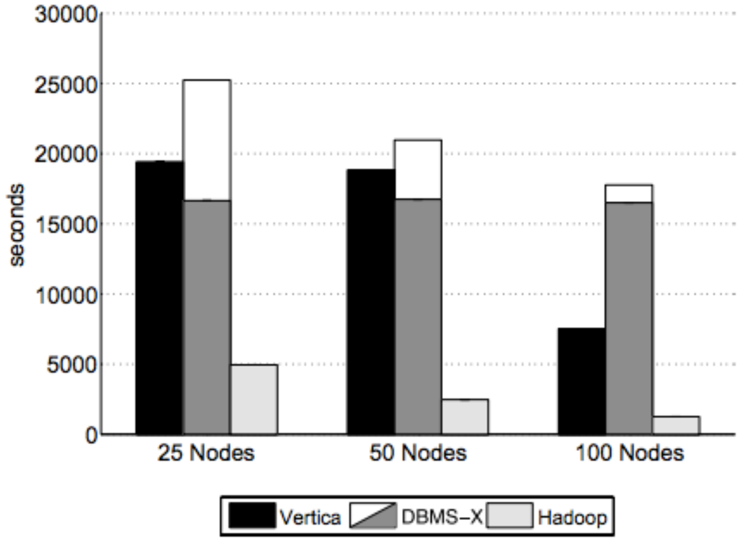


Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

A comparison of approaches to large-scale data analysis. Pavlo et al, SIGMOD 2009.

Why not just use a DBMS? (cont'd)

- Other possible reasons:
  - MapReduce is more scalable.
  - MapReduce is more easily deployed.
  - MapReduce is more easily extended.
  - MapReduce is more easily optimized.
  - MapReduce is free (that is, Hadoop).
  - I already know Java.
  - MapReduce is exciting and new.

BigTable, HBASE, Cassandra

DBM1 – Part 5: Distributed databases

Lots of buzz words!

"Apache Cassandra is an open-source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tunably consistent, column-oriented database that bases its distribution design on Amazon's dynamo and its data model on Google's Big Table."

Basic Idea: Key-Value Store

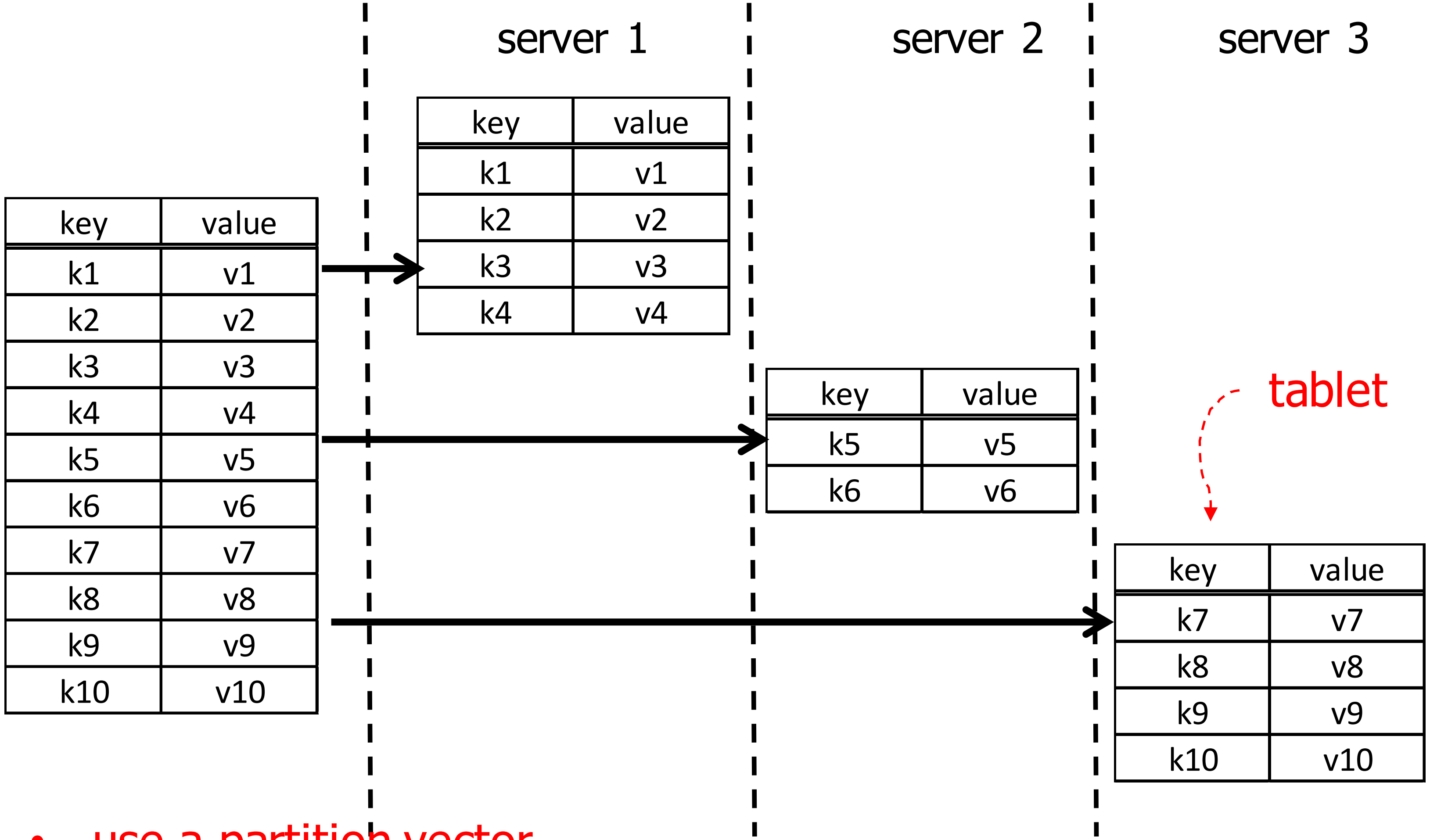
Table T:

key	value
k1	v1
k2	v2
k3	v3
k4	v4

keys are sorted

- API:
  - lookup(key) → value
  - lookup(key, range) → values
  - getNext → value
  - insert(key, value)
  - delete(key)
- Each row has timestemp
- Single row actions atomic
- No multi-key transactions
- Cassandra has an SQL-like query language built on top of that low level API.

Fragmentation (Sharding)



- use a partition vector
- “auto-sharding”: vector selected automatically

Tablet Replication

server 3

key	value
k7	v7
k8	v8
k9	v9
k10	v10

primary

server 4

key	value
k7	v7
k8	v8
k9	v9
k10	v10

backup

server 5

key	value
k7	v7
k8	v8
k9	v9
k10	v10

backup

- Cassandra:
  Replication Factor (# copies)
  R/W Rule: One, Quorum, All
  Policy (e.g., Rack Unaware, Rack Aware, ...)
  Read all copies (return fastest reply, do repairs if necessary)
- HBase: Does not manage replication, relies on HDFS

Needs a "directory"

- Table Name: Key → Server that stores key
  → Backup servers
- Can be implemented as a special table.

Column families

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

- For storage, treat each row as a single "super value"
- API provides access to sub-values (use family:qualifier to refer to sub-values, e.g., price:euros, price:dollars)
- Cassandra allows "super-column": two level nesting of columns (e.g., Column A can have sub-columns X & Y)

Vertical partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

can be <u>manually</u> implemented as

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

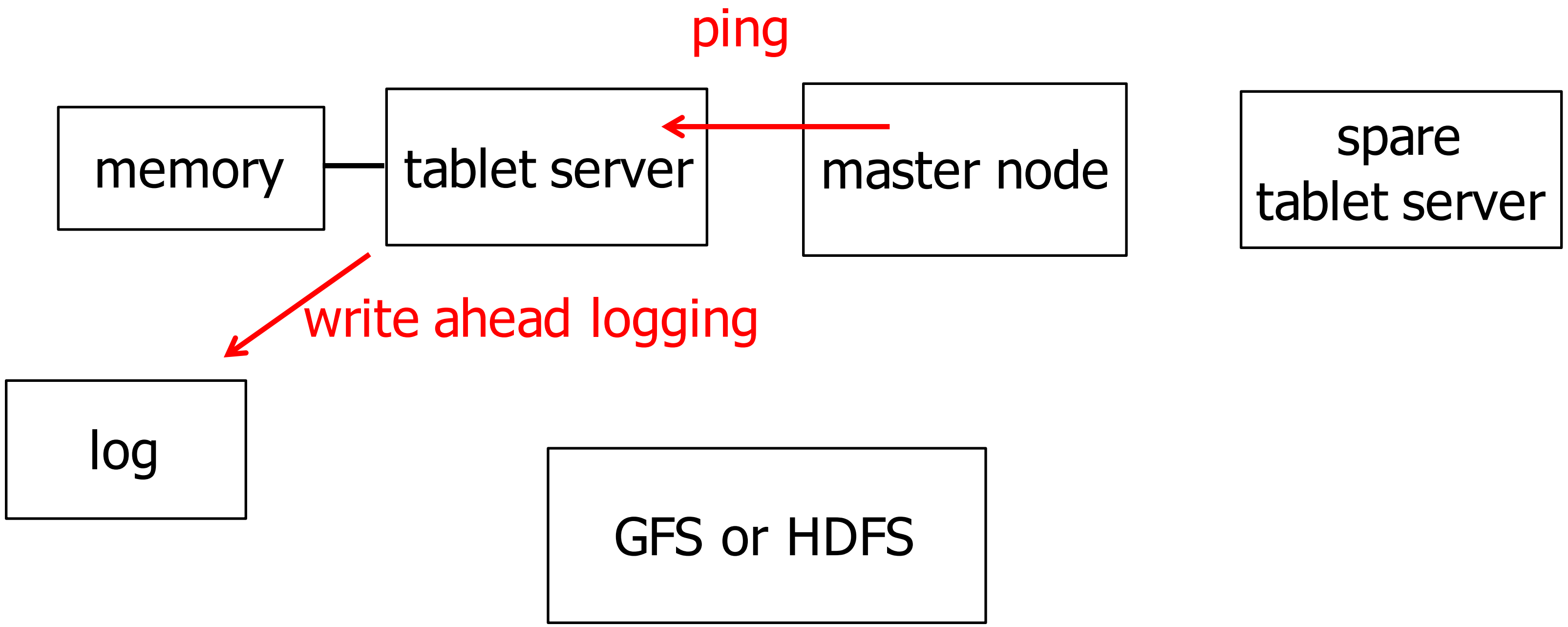
K	C
k1	c1
k2	c2
k4	c4

column family

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

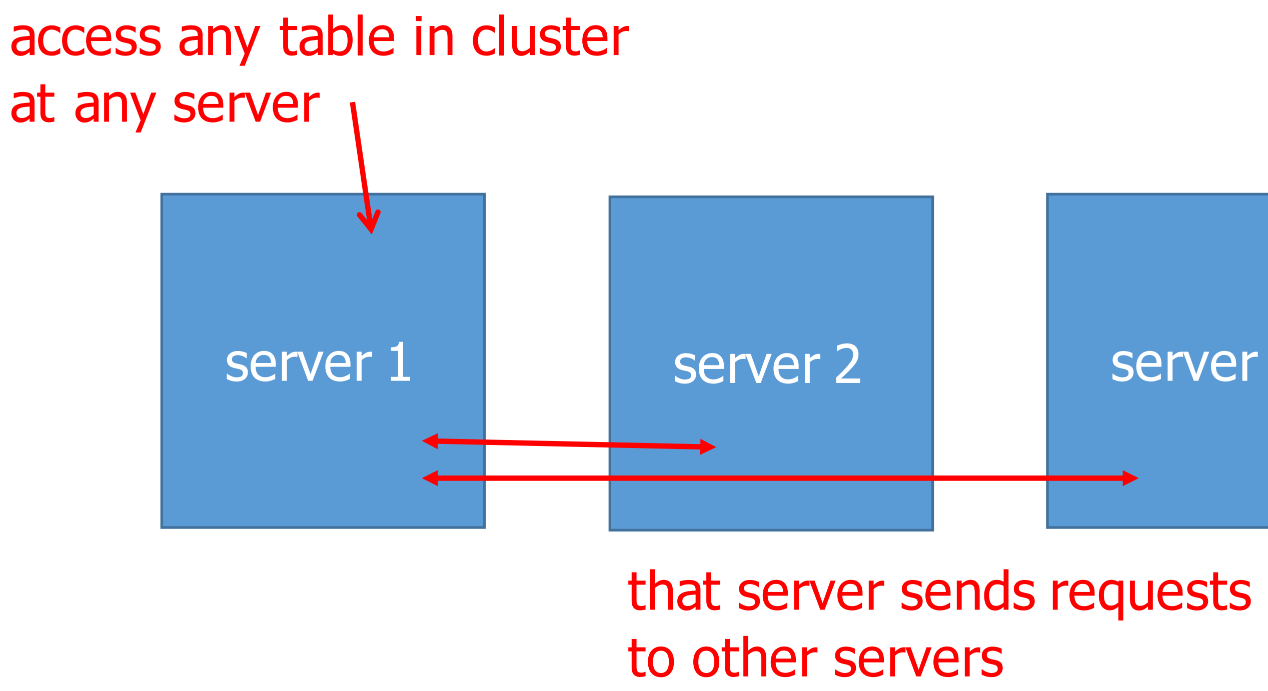
- Good for sparse data and column scans.
- Not so good for tuple reads.
- Are atomic updates to row still supported?
- API supports actions on full table; mapped to actions on column tables.
- API supports column "projection".
- To decide on vertical partition, need to know access patterns

Failure Recovery (BigTable, HBase)



Failure recovery (Cassandra)

No master node, all nodes in "cluster" are equal.



Synthesis

- Alternatives exist to plain old RDBMS.
- These alternatives scale well, but some traditional RDBM properties are sacrificied (e.g., ACID transactions, SQL query language).

- MapReduce introduces a way to process data without actually loading it into a database. It is only about data processing, data is managed by a (possibly distributed) file system.
- Newer frameworks exist, such as Spark.