Dirk Habich

Scalable Data Management (SDM)

Data Fragmentation – Scale-Up/Scale-Out

Focus – Modern Database Systems

Dresden Database Research Group

Foundations

- Relational Data and SQL
- Traditional Database Architecture
- Transactions

Parallelism

- Data-oriented Architecture
- Coarse- and Fine-grained parallelism
- Data Fragmentation

Transactions

- Data Replication
- Distributed Transactions
- CAP-Theorem

Additional

- Map/Reduce and Hadoop
- Spark and its Ecosystem
- Database in the cloud

Trends to Face

Dresden Database Research Group

Application trends

Complex analytical queries

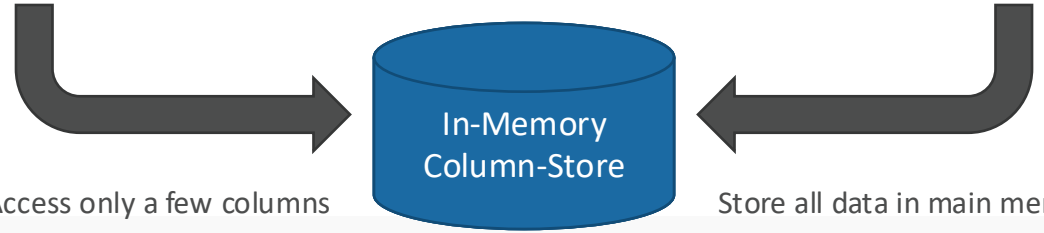
Example: Star Schema Benchmark (SSB) Query 3.1

```
SELECT c_nation, s_nation, d_year, SUM(lo_revenue)
FROM customer, lineorder, supplier, date
WHERE lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_orderdate = d_datekey
  AND c_region = 'ASIA' AND s_region = 'ASIA'
  AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year ASC, revenue DESC;
```

Hardware trends

Main Memory

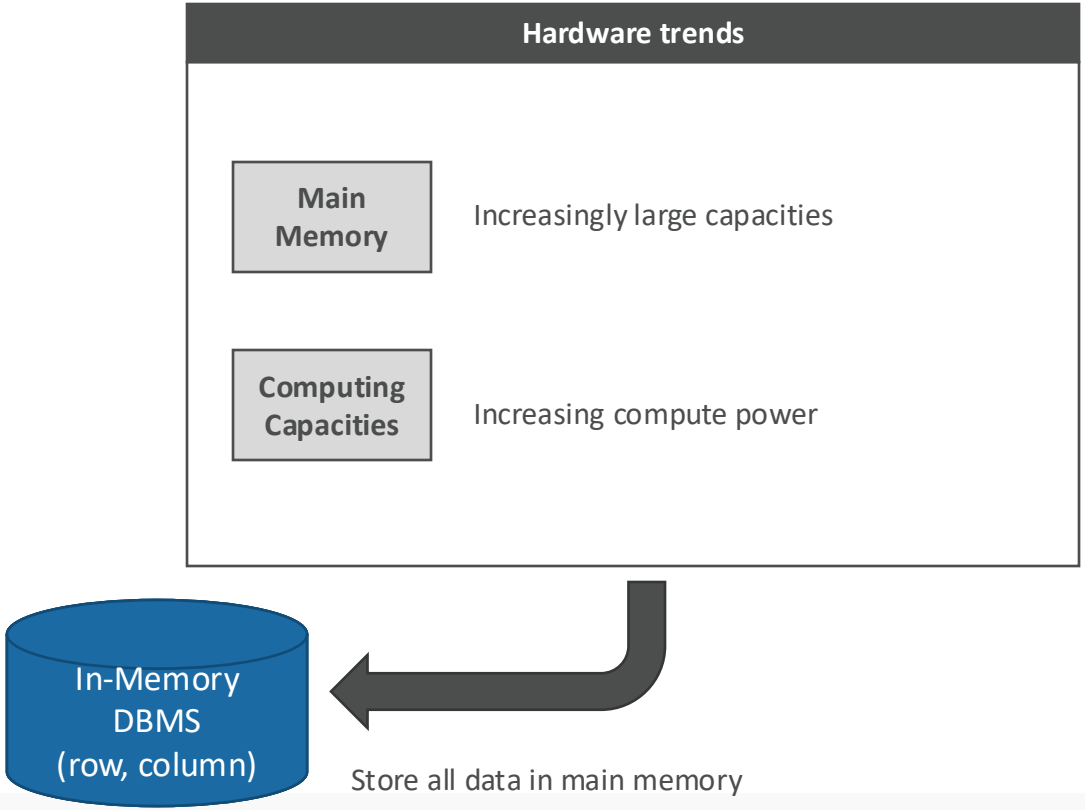
Increasingly large capacities



Access only a few columns

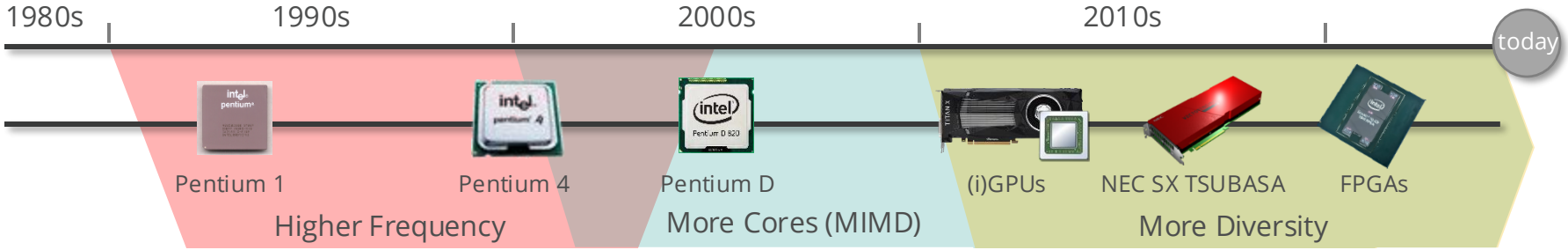
Store all data in main memory

Trends to Face - Parallelism



Evolution of Processing Units

Dresden Database Research Group



Multiple Instruction Multiple Data (MIMD)

- Process data concurrently on multiple cores
- Modern In-Memory OLAP Engines:
  - Multi-Core, Shared nothing

Accelerators

- Ship data to device and get results back

Co-Processing

- Shared memory regions for heterogeneous computing

Processing Units → Memory → Network → Evolving Hardware

High Performance General-Purpose CPUs

Dresden Database Research Group

Classical Hardware

- Single Core
- Small RAM
- Sequential Execution

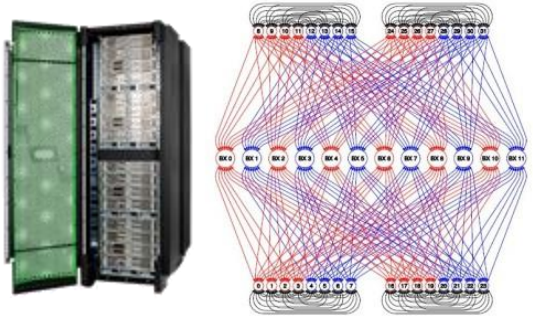
Current Trend

- Many Core Architectures
- Interconnected Sockets
- Each Socket with own RAM

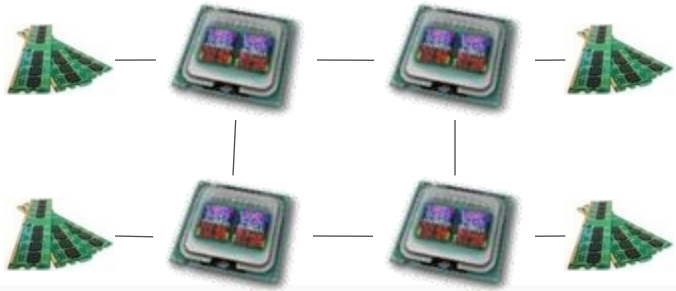
→ Shared Memory

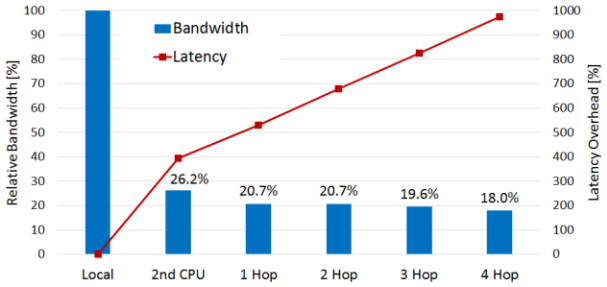
High performance

- SGI, SPARC, …



Increased latency and less bandwidth with each hop!

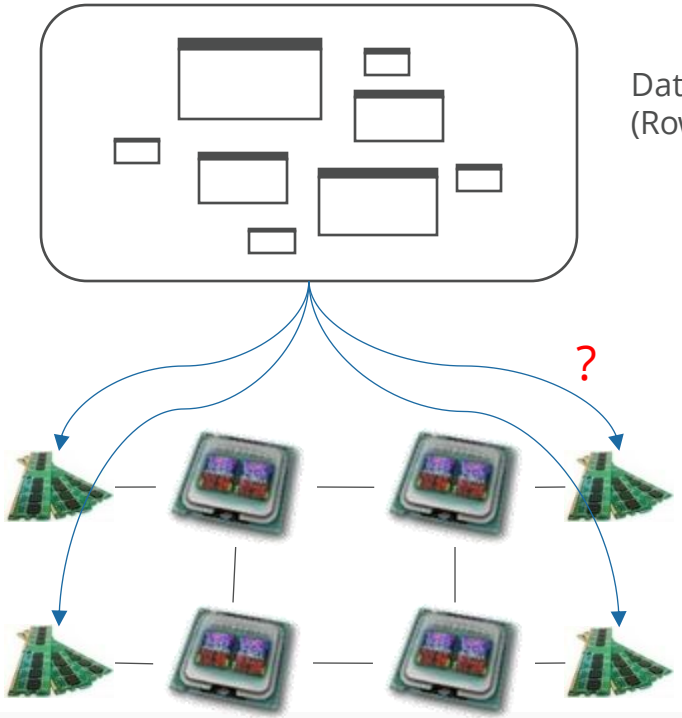




High Performance General-Purpose CPUs

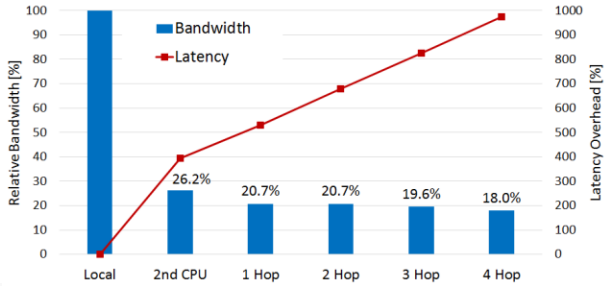
Dresden Database Research Group

Challenge: Data Placement



Database
(Row or Column Format)

Increased latency and less bandwidth with each hop!



Scale-up vs. scale-out Hardware

Dresden Database Research Group

- **Scale-up**: high-cost powerful CPUs, more sockets, more cores, more memory
- **Scale-out**: adding more low cost, commodity servers

- Supercomputer vs. data center
- **Scale**
  - Blue waters = 40K 8-core “servers”
  - Microsoft Chicago Data centers = 50 containers = 100K 8-core servers
- **Network architecture**
  - Supercomputers: InfiniBand, low-latency, high bandwidth protocols
  - Data Centers: (mostly) Ethernet based networks
- **Storage**
  - Supercomputers: separate data farm
  - Data Centers: use disk on node + memory cache

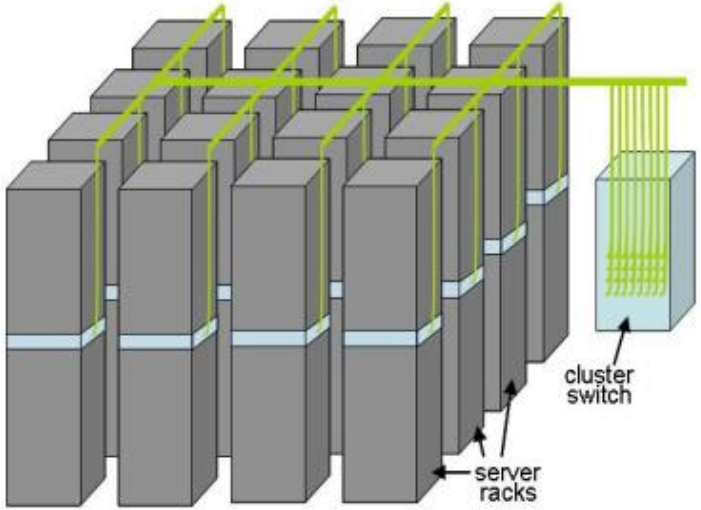
Traditional Data Center Architecture (Scale-Out)

Dresden Database Research Group

Servers mounted on 19'' rack cabinets

Servers
- CPUs
- DRAM
- Disks

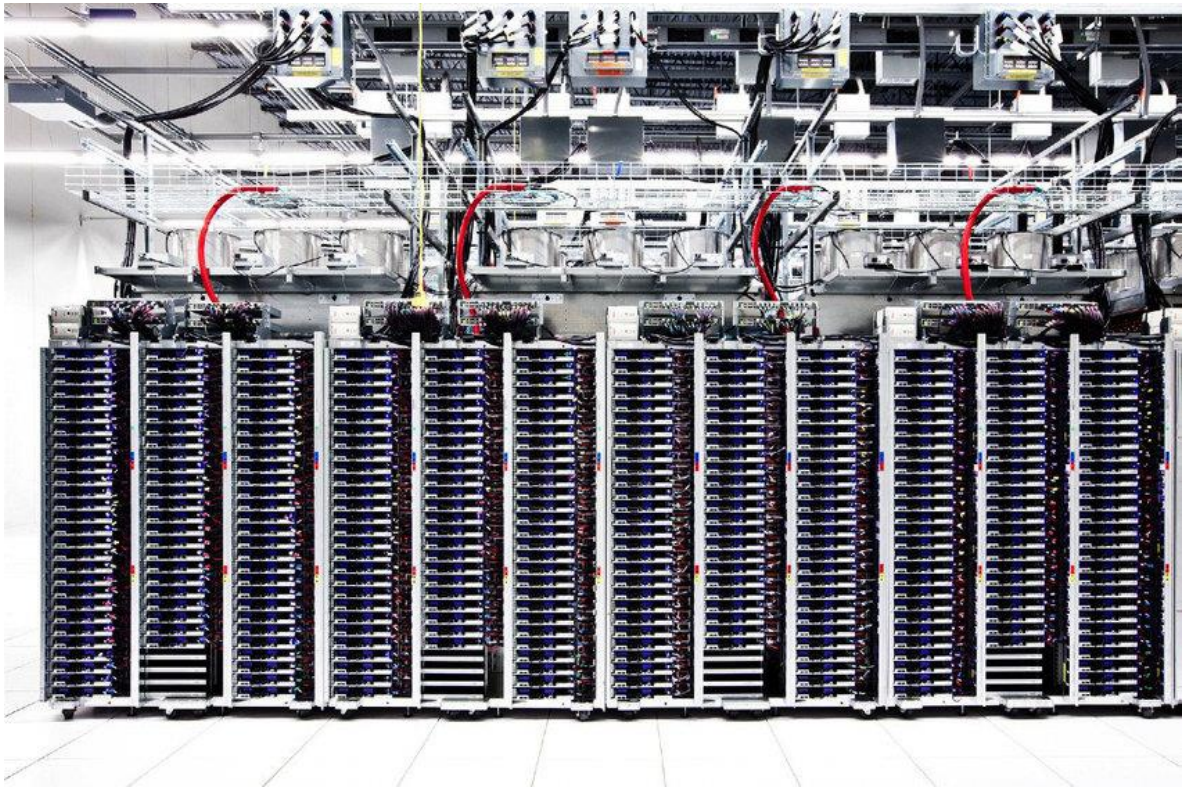
Racks
- 40-80 servers
- Ethernet switch



Clusters

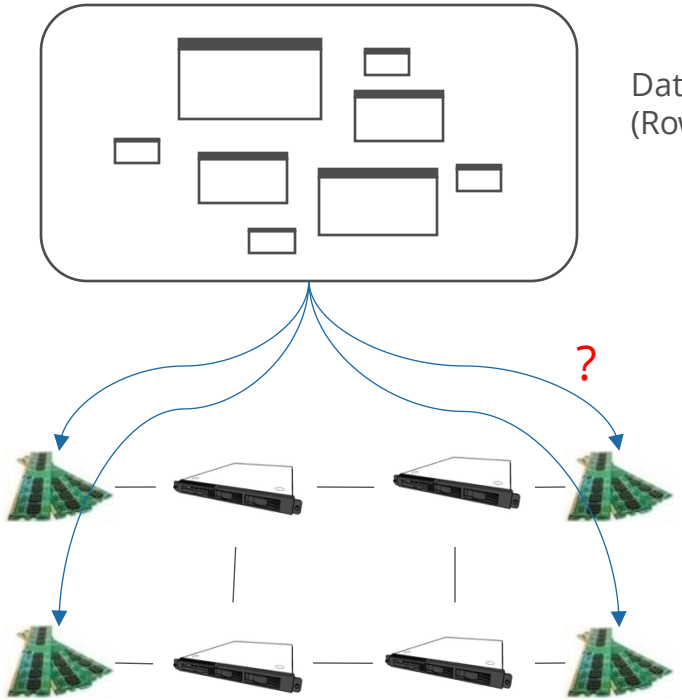
Racks are placed in single rows forming corridors between them.

A Row of Servers in a Google Data Center



Scale-Out

Challenge: Data Placement



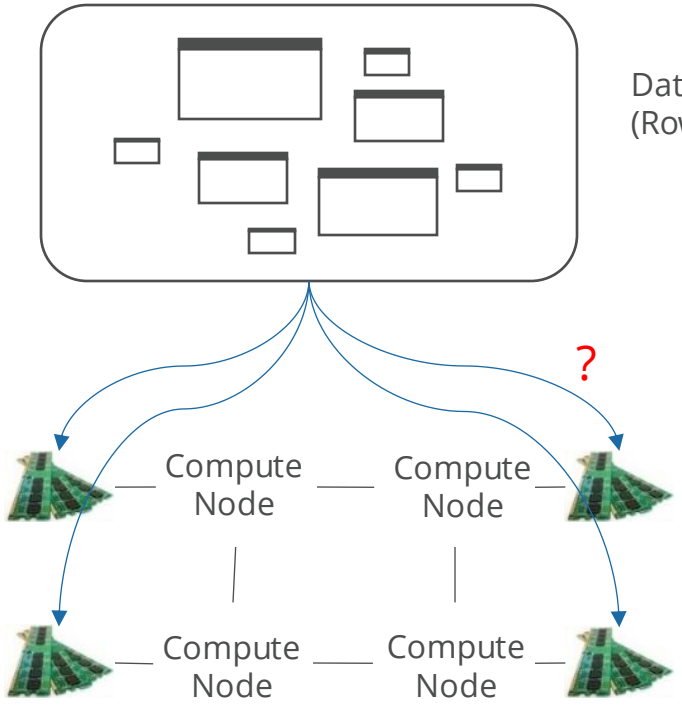
Database
(Row or Column Format)

Increased latency and less bandwidth with each hop!

Scale-Up and Scale-Out Similarity

Dresden Database Research Group

Challenge: Data Placement



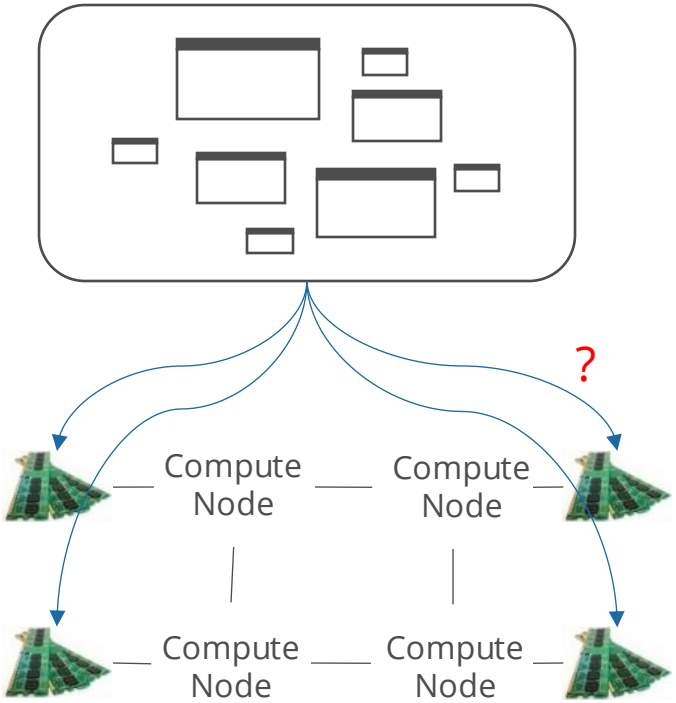
Increased latency and less bandwidth with each hop!

Fragmentation / Partitioning

Dresden Database Research Group

Partitioning

- Split the big dataset into smaller subsets called partitions.
- Each partition placed on a separate node.
- Reduces latency for analytical jobs
- Can improve availability



Partitioning

Partitioning

Dresden Database Research Group

For very large datasets, or very high throughput, we need to break the data up into partitions.

Clarifying terminology:

- What we call a partition here is called a shard in MongoDB, Elasticsearch, and SolrCloud;
- region in Hbase,
- a tablet in BigTable,
- avnode in Cassandra and Riak,
- and a vBucket in Couchbase.

Partitions are defined in such a way that a piece of data belongs to exactly one partition.

Partitioning is important for achieving better scalability, but it can also

- Reduce contention
- Improve performance
- Optimize storage costs
- Improve security

Why partition data?

Dresden Database Research Group

Improve scalability

- Different partitions can be placed on different nodes in a shared nothing cluster

Improve performance

- Data operations on each partition work on smaller data volume
- Operations that affect more than one partition can run in parallel

Improve security

- Can separate sensitive and non-sensitive data into different partitions and apply different security controls to the sensitive data

Improve availability

- Avoid a single point of failure. If one partition becomes unavailable, the others are still intact.

Allows better customization

Designing partitions

Dresden Database Research Group

Horizontal partition (sharding):

- Each partition is a separate data store, but all partitions have the same schema
- Each partition is known as a shard and holds a specific subset of the data
  - e.g., all the orders for a specific set of customers

Vertical partitioning:

- Each partition holds a subset of the fields for items in the data store
- e.g., frequently accessed fields, may be placed in one vertical partition and less frequently accessed fields in another.

Functional partitioning:

- Data is aggregated according to how it is used by each bounded context in the system
- e.g., An e-commerce might store invoice data in one partition and product inventory data in another

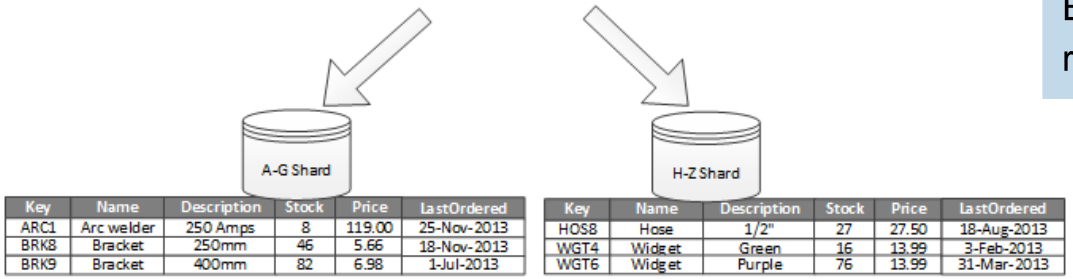
Horizontal Partitioning

Horizontal Partitioning (sharding)

Dresden Database Research Group

Example

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Product inventory data is divided into shards based on the product key.

Each shard holds the data for a cont. range of shard keys (A-G and H-Z)

The most important factor is the choice of sharding key.

- Goal is to spread the data and query load evenly across the nodes.
- If the partitioning is unfair, some partitions will have more data or queries, we call it skewed.
- A partition with disproportionally high load is called a hot spot.

Horizontal Partitioning Strategies

Dresden Database Research Group

By Key Range

- Assign a continuous range of keys to each partition.
- The range of keys are not necessarily evenly spaced, because your data may not be evenly distributed.
  BigTable, Hbase, RethinkDB, and MongoDB before v2.4

Advantages

- Within each partition we can keep the keys in sorted order
- Range scans are fast and easy
- Can fetch several related records in one query

Disadvantage

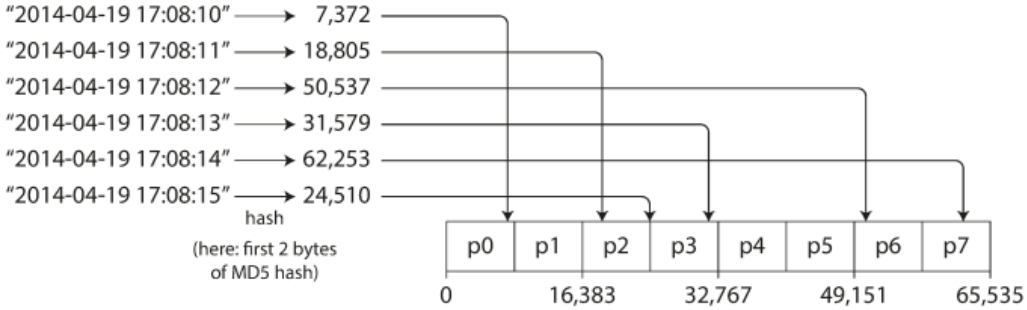
- Certain access primitives can lead to hot spots

Horizontal Partitioning Strategies/2

Dresden Database Research Group

By Hash of Key

- hash a key to determine the partition
- a partition for a range of hashes
- if a key's hash value belongs to a partition's range then the key is placed in that partition.



Advantage:

- No problem with skew and hot spots (overstatement, we may still have issues, but they are rare)
-

Disadvantage:

- No longer easy to do efficient range queries.
- e.g., range queries on the primary key are not supported by Riak, Couchbase or Voldemort.

Vertical & Functional Partitioning

Vertical Partitioning

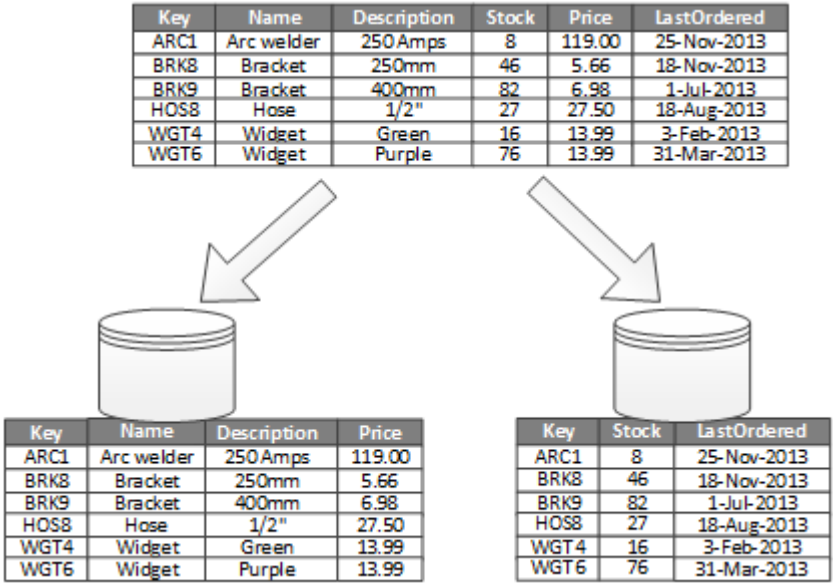
Dresden Database Research Group

Goal to reduce the I/O and performance costs when fetching items that are frequently accessed.

- Different properties of an item are stored in different partitions.
- One partition holds data that is accessed more frequently: product name, description and price
- Another holds inventory data: the stock count and the last ordered date.

Reasons

- Application regularly gets the product name, desc. and price when displaying the product details.
- Stock count and last ordered data are commonly used together and are more frequently modified.



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

Vertical Partitioning cont.

Dresden Database Research Group

Other advantages:

- Relatively slow-moving data can be separated from the more dynamic data
  - Slow moving data is a good candidate for an application to cache in memory
- Sensitive data can be stored in a separate partition with additional security control.

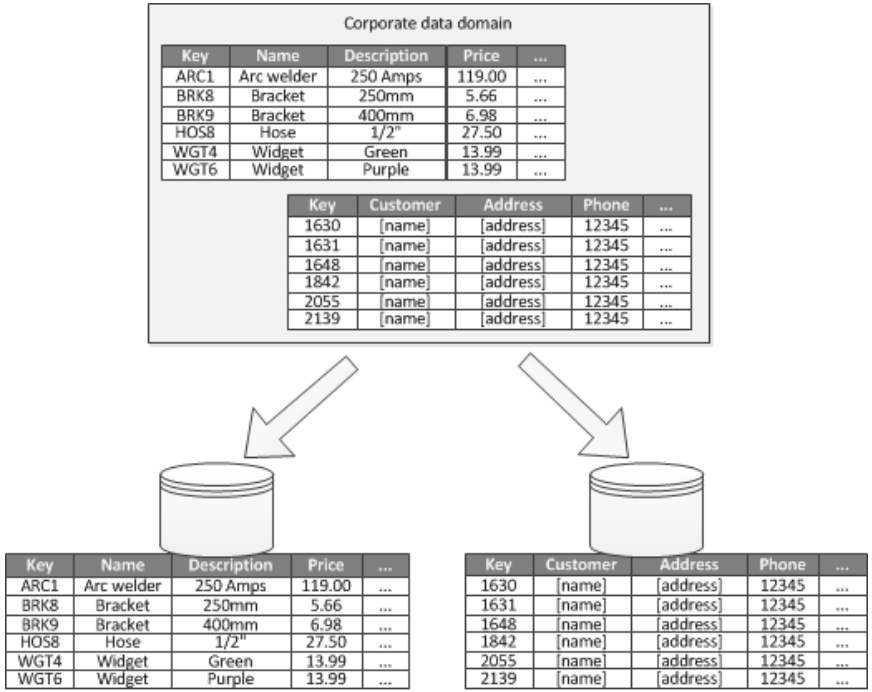
Ideally suited for column-oriented data stores.

Functional Partitioning

Dresden Database Research Group

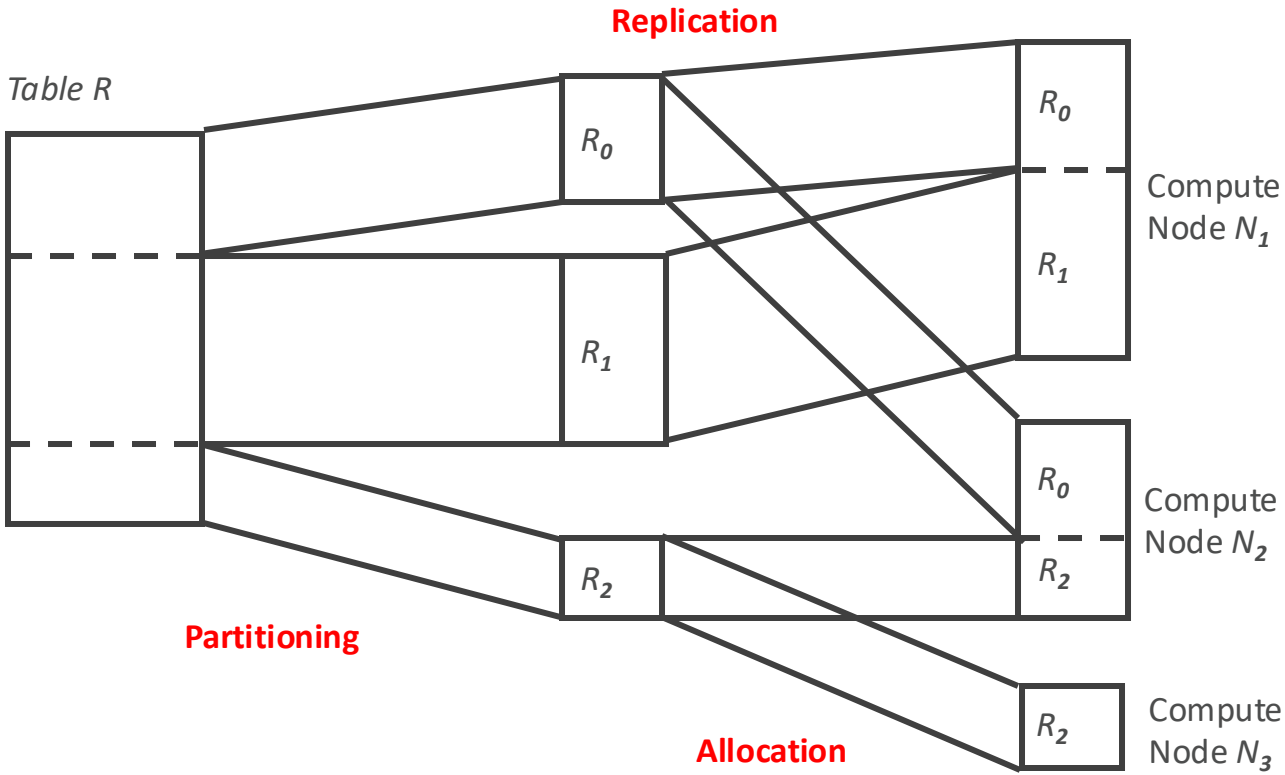
Properties

- When possible, identify a bounded context, functional partitioning is a way to improve isolation and data access performance.
- Another common use is to separate read-write data from read-only data
- This strategy can help reduce data access contention across different parts of the system



Relational Database-managed Distribution Schemes

Overview: Data Distribution Schemes



Overview Partitioning Strategies - Recap

Dresden Database Research Group

Horizontal Partitioning

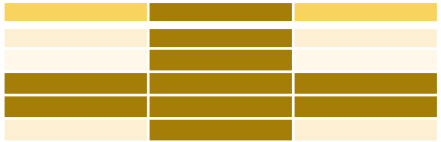
- Partitioning into tuple subsets

Vertical Partitioning

- Partitioning into column subsets

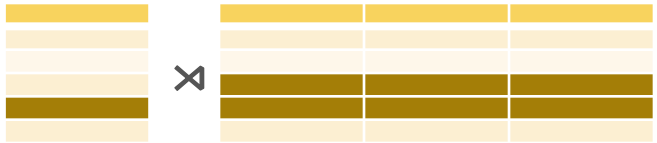
Hybrid Partitioning

- Hierarchical use of horizontal and vertical fragmentation



Derived horizontal Partitioning

- Partitioning in tuple subsets based on criteria derived from join partner



Correctness Properties

Dresden Database Research Group

Assumption

- $R \rightarrow R_1, R_2, \ldots, R_n$ (Relation R is partitioned into *n* fragments)

Completeness

- Each item from R must be included in at least one fragment

Reconstruction

- Exact reconstruction of fragments must be possible

Disjointness

- $R_i \cap R_j = \emptyset$ with $(1 \leq i, j \leq n, i \neq j)$
- Overlapping partitions introduce redundancy

Running Example

Dresden Database Research Group

ID	Firstname	Lastname	Affiliation	CID
102	Anastasia	Ailamaki	EPFL	1
103	Philip A.	Bernstein	Microsoft Research	2
107	Michael J.	Carey	University of California	3
117	Surajit	Chaudhuri	Microsoft Research	2
121	AnHai	Doan	University of Wisconsin	4
201	Daniela	Florescu	Oracle	5
208	Laura M.	Haas	IBM Almaden Research	6
235	Alon Y.	Halevy	Google	7
267	Donald	Kossmann	ETH Zürich	8
301	Samuel	Madden	MIT	9
341	Raghu	Ramakrishnan	YAHOO! Research	6
392	Michael	Stonebraker	MIT	9
394	Gerhard	Weikum	MPI	10

[Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Henry F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, Gerhard Weikum: The Claremont report on database research. SIGMOD Record (SIGMOD) 37(3):9-19 (2008)]

Running Example (2)

Dresden Database Research Group

Relation R

ID	...	Affiliation	CID
102	...	EPFL	1
103	...	Microsoft Research	2
107	...	University of California	3
117	...	Microsoft Research	2
121	...	University of Wisconsin	4
201	...	Oracle	5
208	...	IBM Almaden	6
235	...	Google	7
267	...	ETH Zürich	8
301	...	MIT	9
341	...	YAHOO! Research	6
392	...	MIT	9
394	...	MPI	10

Relation S

CID	City	Country
1	Lausanne	CH
2	Redmond	USA
3	Irvine	USA
4	Madison	USA
5	Redwood City	USA
6	San Jose	USA
7	Mountain View	USA
8	Zürich	CH
9	Cambridge	USA
10	Saarbrücken	GER

Horizontal Partitioning

Dresden Database Research Group

Horizontal partitioning of relation R into n fragments R_i

- complete, disjoint, re-constructable
- schema of fragments is equivalent to schema of base relation

Partitioning

- split by n selection predicates p_i (fragmentation predicate) on attributes of R
- $R_i = \sigma_{p_i} R$ with $(1 \leq i \leq n)$

Reconstruction

- union of all fragments
- $\cup_{1 \leq i \leq n} R_i$

Horizontal Fragmentation (2) – By Key

Dresden Database Research Group

Partitioning

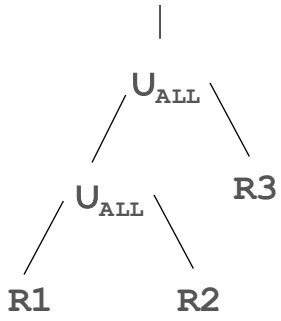
- `SELECT * INTO R3 FROM R WHERE ID≥300 AND ID<400`

R3:

ID	Firstname	Lastname	Affiliation	CID
301	Samuel	Madden	MIT	9
341	Raghu	Ramakrishnan	YAHOO! Research	6
392	Michael	Stonebraker	MIT	9
394	Gerhard	Weikum	MPI	10

Reconstruction

-
```
SELECT * FROM R1
  UNION ALL
SELECT * FROM R2
  UNION ALL
SELECT * FROM R3
```



Vertical Partitioning

Dresden Database Research Group

Vertical (column-wise) partitioning of relation R into n fragments R_i

- complete, re-constructable, but not disjoint (primary key)
- completeness: each attribute must be included in at least one fragment
- reconstruction: join (primary key required)

Partitioning

- partitioning by projection
- redundancy of primary key
- $R_i = \pi_{PK,A_i} R$ with $(1 \leq i \leq n)$

Reconstruction

- natural join over primary key
- $R = R_1 \bowtie_{PK} \cdots \bowtie_{PK} R_n$ with $(1 \leq i \leq n)$

Vertical Partitioning (2)

Dresden Database Research Group

Partitioning

- `SELECT ID, Firstname, Lastname INTO R1 FROM R`
- `SELECT ID, Affiliation, CID INTO R2 FROM R`

R1:

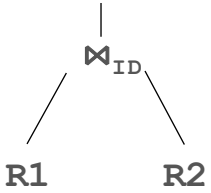
ID	Firstname	Lastname
102	Anastasia	Ailamaki
103	Philip A.	Bernstein
107	Michael J.	Carey
…	…	…

R2:

ID	Affiliation	CID
102	EPFL	1
103	Microsoft Research	2
107	University of California	3
…	…	…

Reconstruction

```
SELECT R1.ID, Firstname, Lastname,
       Affiliation, CID
  FROM R1, R2
  WHERE R1.ID = R2.ID
```



Derived Horizontal Partitioning

Dresden Database Research Group

Horizontal partitioning of relation R in n fragments R_i, where the fragmentation predicate is derived from relation S

- Potentially complete, re-constructable, disjoint
- foreign key relationships

Partitioning

- selection on independent relation S
- semi-join with dependable relation R
- $R_i = R \ltimes \sigma_{p_i} S = \pi_R(R \bowtie \sigma_{p_i} S)$

Reconstruction

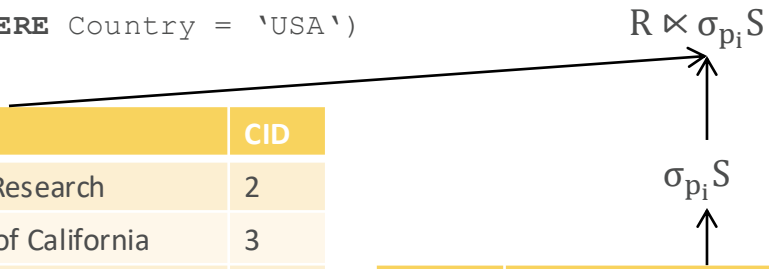
- analog to horizontal fragmentation
- union of all fragments
- $\mathrm{U}_{1 \le i \le n} R_i$

Derived Horizontal Partitioning (2)

Dresden Database Research Group

Partitioning

- ```
  SELECT * INTO R1
  FROM R R1
  WHERE CID IN ( SELECT CID FROM S WHERE Country = 'USA')
  ```



ID	...	Affiliation	CID
103	...	Microsoft Research	2
107	...	University of California	3
117	...	Microsoft Research	2
121	...	University of Wisconsin	4
201	...	Oracle	5
208	...	IBM Almaden	6
235	...	Google	7
301	...	MIT	9
341	...	YAHOO! Research	6
392	...	MIT	9

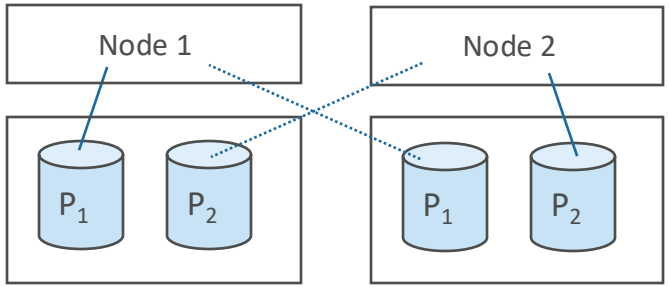
CID	City	Country
2	Redmond	USA
3	Irvine	USA
4	Madison	USA
5	Redwood City	USA
6	San Jose	USA
7	Mountain View	USA
9	Cambridge	USA
  ```

Allocation of Partitions to Nodes

Dresden Database Research Group

Assignment of partitions to nodes

- allocation
- Number of different data nodes (degree of parallelism)
- Balanced distribution of m fragments for D data nodes



Rebalancing partitions

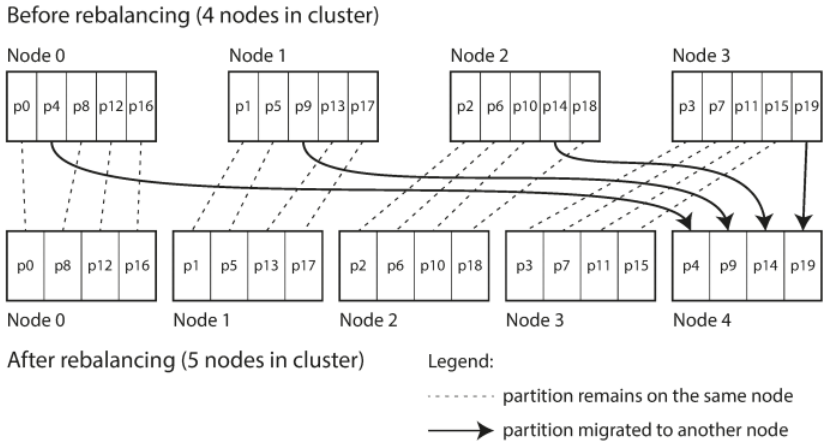
Dresden Database Research Group

Observation

- Rebalancing is often necessary

Strategies of rebalancing:

- How not to do it? Hash mod N.
  - If the number of nodes N changes, most of the keys will need to be moved from one node to another.



- Fix the number of partitions P so that P >> N
  - If a node is removed/added to the cluster, only a few (entire) partitions need to be moved.
  - The number of partitions remains the same, and the assignment of keys to partitions is not changed.

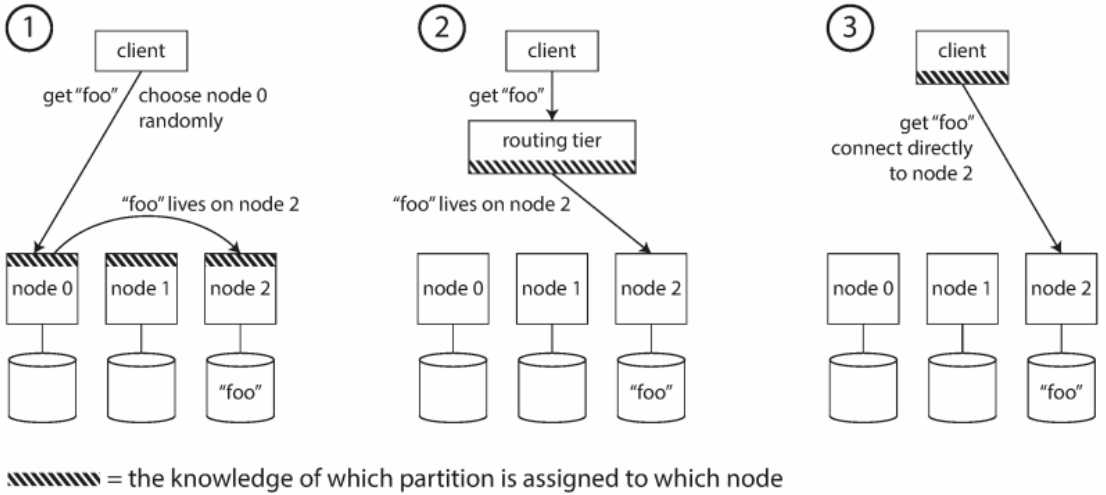
- Dynamic partitioning
  - Applicable with range and hash partitioning
  - When a partition grows to exceed a size, split it into two (like in a B-tree).
- Partitioning proportional to nodes
  - Have a fixed number of partitions per node.

Request Routing

Dresden Database Research Group

Open Question

- when a client wants to make a request, how does it know which node to ask?
  - As partitions are rebalanced, the assignment of partitions to nodes changes
- Someone needs to have the top-level overview.



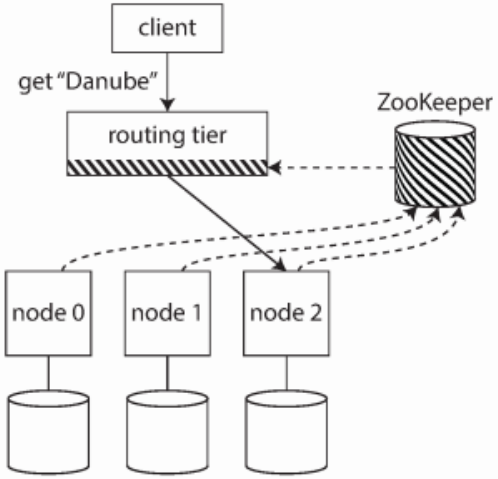
- **Three main options:**
  - The node layer
  - The routing tier (or third party)
  - The clients
  - It is a challenging problem as all participants need to agree → requires reaching a consensus.

Request Routing cont.

Dresden Database Research Group

Many systems rely on a coordination service such as Zookeeper to keep track of meta data.

- The routing tier can subscribe to this information from the ZooKeeper service



Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

= the knowledge of which partition is assigned to which node

Summary

Partitioning …

- is necessary when data and load volume exceeds a single machine's capacity.
- The goal is to spread the data and query load evenly across multiple machines, avoiding hotspots.
- Need to be careful when choosing the partitioning scheme so that it is appropriate to the data and workload properties

Three main types of partitioning:

- horizontal,
- vertical and
- functional.

Two main approaches for horizontal partitioning: key range and hash-based

Allocation of partition to nodes is a separate step

- Rebalancing is often required
- Various techniques are available