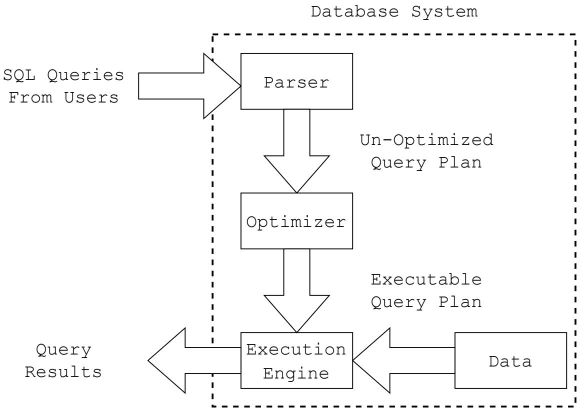
Introduction to Cloud Databases

What is a Database System?

- Relational Database Management Systems (RDBMS) store data in tables.
- They perform operations written in SQL



Relational DBs for Different Workloads

- **OLTP**: Online Transaction Processing systems handle frequent updates.
  e.g. banking services
- **OLAP**: Online Analytical Processing systems handle analysis of large datasets.
  e.g. business analytics and recommendations for large online shops
- **HTAP**: Hybrid transactions/analytical processing systems support both workloads.
  e.g. business analytics and recommendations for large online shops on live data

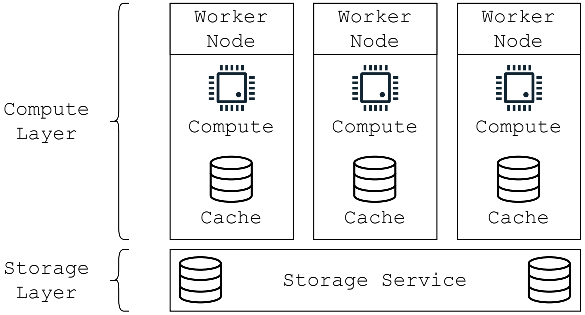
What what makes a good DB?

There are different requirements for DBs:

- Fast answers to queries
- Cheap answers to queries (as in cloud cost)
- Predictable pricing
- Simple setup (serverless)
- Huge datasets

What Changes in a Distributed System?

- Multiple nodes in a cluster
- Network communication



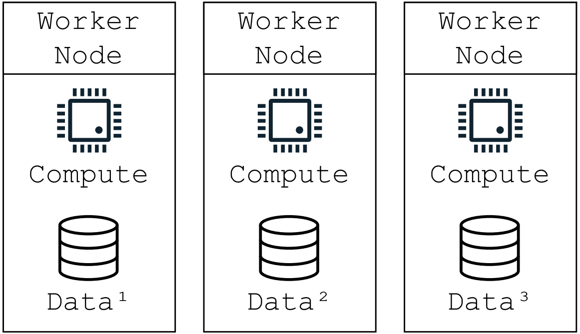
Distributed Architectures

- Shared Nothing
- Decoupled Storage
- Decoupled State and Storage

Distributed Architectures: Shared Nothing

Shared Nothing

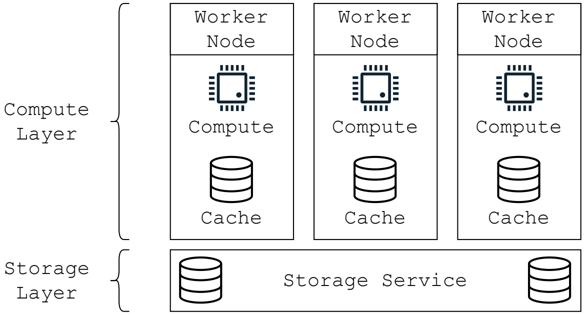
- Data is partitioned to nodes
- Changing cluster size difficult
- If a single node fails the whole system fails



Distributed Architectures: Decoupled Storage

Decoupled Storage

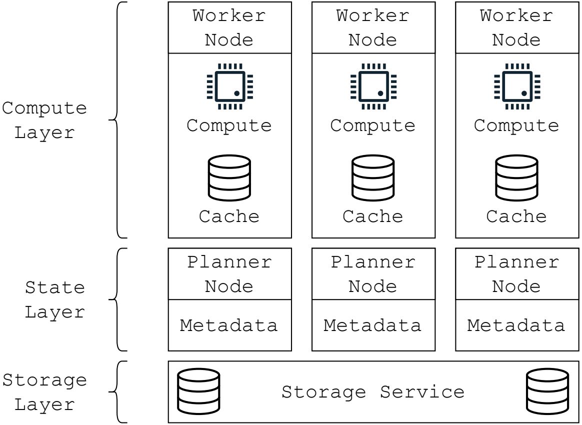
- Data is only cached at nodes
- Changing cluster size easy
- If a single node fails current queries will fail, but the cluster stays intact



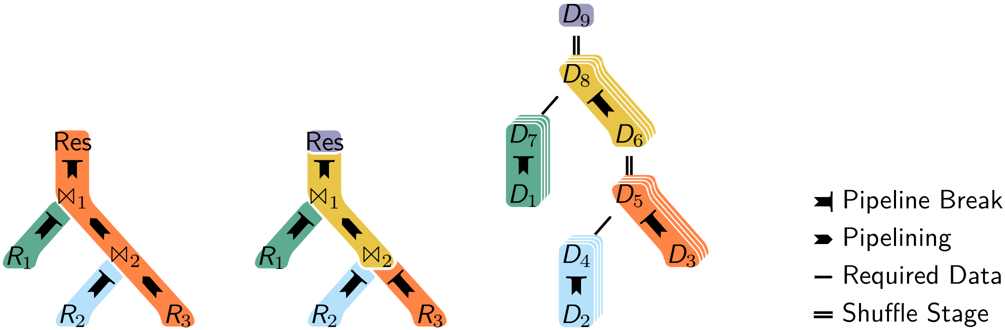
Distributed Architectures: Decoupled State and Storage

Decoupled State and Storage

- Intermediate results are always materialized to storage service
- Changing cluster size easy
- If a single compute node fails even current queries can resume from intermediate state
- May scale to thousands of nodes
- Might have significant overhead
- Microsoft Polaris does this



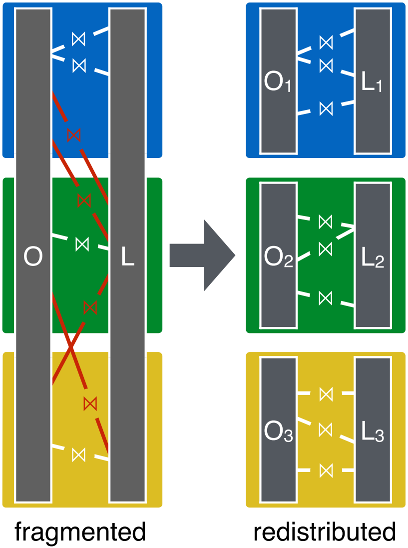
Distributed Query Execution



- Pipelines can be executed in parallel (Scan, Select, Map,...)
- Some operators require data shuffle between nodes (Joins, Aggregations,...)

Distributed Joins (1)

- Tuples may need to join with tuples on remote nodes
- Repartition and redistribute both relations on join key
- All potential join-partners will be in same partition



Distributed Joins (2)

- Tuples may need to join with tuples on remote nodes
- Repartition and redistribute both relations on join key
- All potential join-partners will be in same partition

orders

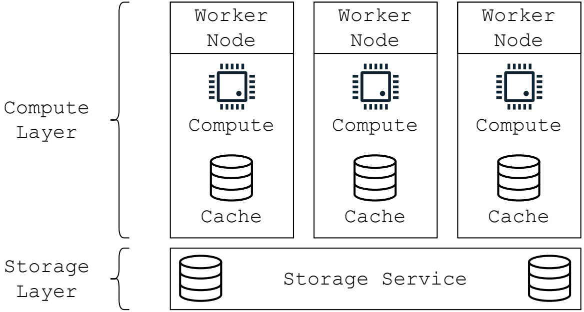
node	key	priority
node 1	1	1-URGENT
node 1	2	2-HIGH
node 1	3	1-URGENT
node 1	4	5-LOW
node 1	5	3-MEDIUM
node 1	6	1-URGENT
node 1	7	2-HIGH
node 1	8	1-URGENT
node 2	9	1-URGENT
node 2	10	2-HIGH
node 2	11	3-MEDIUM
node 2	12	5-LOW
node 2	13	1-URGENT
node 2	14	3-MEDIUM
node 2	15	1-URGENT
node 3	16	3-MEDIUM
node 3	17	2-HIGH
node 3	18	3-MEDIUM
node 3	19	5-LOW
node 3	20	1-URGENT
node 3	21	2-HIGH

lineitem

node	key	shipmode
node 1	1	MAIL
node 1	1	MAIL
node 1	1	MAIL
node 1	2	SHIP
node 1	2	MAIL
node 1	6	SHIP
node 1	6	SHIP
node 1	6	SHIP
node 2	9	MAIL
node 2	10	SHIP
node 2	11	MAIL
node 2	11	MAIL
node 2	13	MAIL
node 2	13	MAIL
node 3	16	MAIL
node 3	16	SHIP
node 3	17	MAIL
node 3	18	MAIL
node 3	18	MAIL
node 3	19	SHIP
node 3	20	SHIP

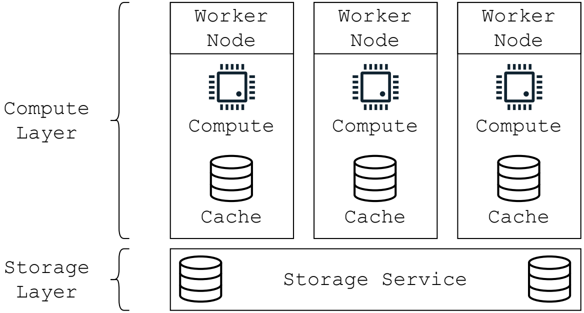
What Changes in the Cloud?

- Storage services allow cheap highly available storage of large datasets (Amazon S3)
- On demand use of compute nodes to save money (Amazon EC2)
- Available cloud instances and services dictate architecture
- Typical x86 cloud instance c5n.18xlarge:
  72 cores, 192 GiB RAM, 100 gbit/s network,
  ≈\$3.9/h



Storage Service: What are Object Stores?

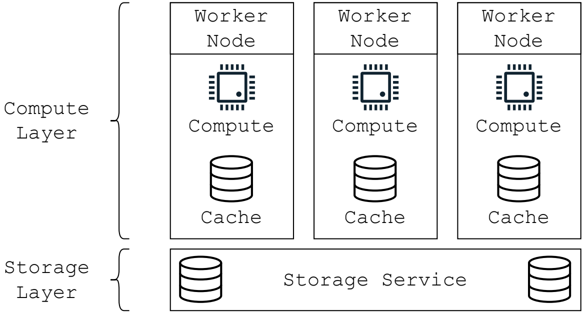
- **Objects** (blobs) are chunks of data (<=5TiB)
  - Read and write objects by their key
  - You can overwrite objects but not modify them
- Implemented as distributed systems
  - Very high availability
  - Very high bandwidth for concurrent requests
- Pricing based on used storage and per request
  - Much cheaper than running storage servers
- There exist many alternatives (Amazon S3, Azure Blobs, Google Cloud Storage, Backblaze B2)



System Properties

Modern systems may have some of the following properties:

- All data on storage service
- Ephemeral compute nodes to adjust to workload
- Heterogeneous nodes (storage, compute, network)
- Node-local cache of data and intermediate results
- Parallel execution of single queries (intra-level parallelism)
- Intermediate result materialization to storage service (recover large queries)



Partitioning Schemes

ID	Name	Age	Country	Occupation
1	John	30	USA	Engineer
2	Alice	25	UK	Teacher
3	Bob	35	Canada	Doctor
4	Maria	28	Brazil	Scientist
5	Emily	40	Australia	Lawyer

Vertical

ID	Name	Age
1	John	30
2	Alice	25
3	Bob	35
4	Maria	28
5	Emily	40

ID	Country	Occupation
1	USA	Engineer
2	UK	Teacher
3	Canada	Doctor
4	Brazil	Scientist
5	Australia	Lawyer

- Vertica uses a similar technique

Horizontal

ID	Name	Age	Country	Occupation
1	John	30	USA	Engineer
2	Alice	25	UK	Teacher

ID	Name	Age	Country	Occupation
3	Bob	35	Canada	Doctor
4	Maria	28	Brazil	Scientist
5	Emily	40	Australia	Lawyer

- May be Hash or Range Partitioning
- Allows distributing data processing
- Used by almost all DBs

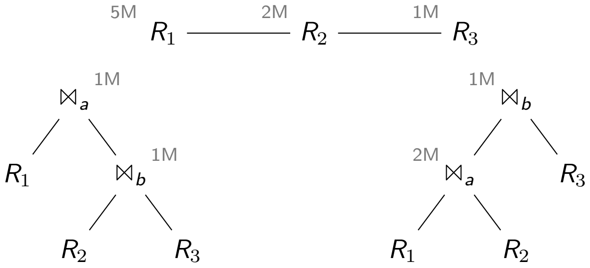
What is a Query Optimizer?

- SQL is declarative
- Many different execution plans compute the correct results
- How do we find the cheapest one?

Example Minimizing Cout

- One of the most important optimizations is join ordering:

```
SELECT *
FROM R1, R2, R3
WHERE R1.a = R2.a
  AND R2.b = R3.b
```



$C_{\text{out}} = 2M$ $\quad$ $C_{\text{out}} = 3M$

- Size of intermediate results approximates execution time (very roughly!)

Challenges for Optimizers

- Join ordering is an NP-hard problem
- Optimization time must be very quick
    - Queries may contain many joins, but execute very quickly
    - Latency might be dominated by optimization
- Cardinality estimation

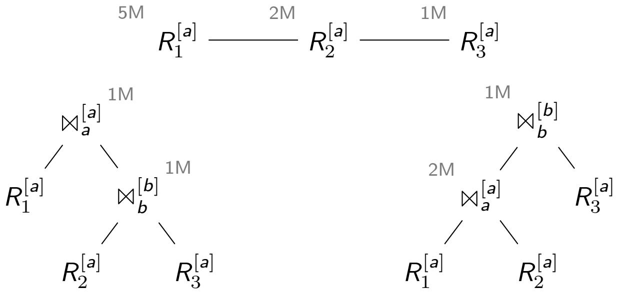
Cardinality Estimation

- We don't know exact sizes of intermediate results
- Systems use statistics of data
- Seems impossible for deep query trees behind join and group-by operators
- Also has to be very fast
- State of the art:
    - Works well about 2-3 joins deep
    - Uses a sample of the data and other statistics

What's New for Distributed Optimizers?

- Cost function
- Network vs. compute
- Search space much larger
- New problem of partition assignment

Example Distributed Cost Function



$C_{\text{out}} = 2M$ $\quad$ $C_{\text{out}} = 3M$

- Re-partitioning might be very expensive
- Re-partitionings can be avoided with good join order
- C_{out} does not show this cost
- We might want to approximate running time directly