Modern OLAP Databases

Iztok Savnik, FAMNIT

October, 2025.

Sources

- Course:
  - Carnegie Mellon University (CMU)
  - Advanced Database Systems
  - Prof. Andy Pavlo
  - Transparencies
- Papers:
  - Conferences VLDB, SIGMOD, CIDR, etc.
  - Journals ACM TODS, IS, VLDBJ, etc.

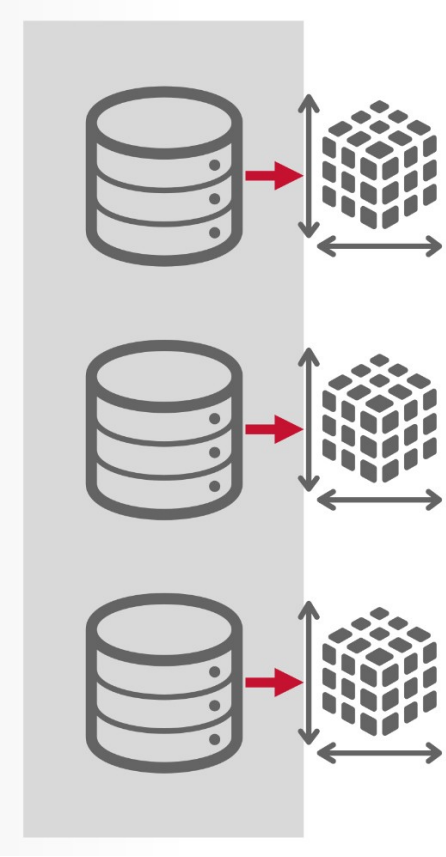
Outline

- Background
- Architecture Overview
- Query Execution
- Project Discussion

Background

- Types of database systems: OLTP, OLAP
- On-line analytical processing (OLAP)
  - Organizations use OLAP to extract new information from existing data sets.
  - Ingesting new information (various sources, from the outside, from organizations).
  - Run queries to obtain new knowledge.
- History of OLAP
  - Monolithic DBMSs that had all an organization's data in centralized managed storage.

1990s – DATA CUBES



```
SELECT product, region, cdate,
SUM(amount) AS total_sales
FROM sales
GROUP BY CUBE (product, region, cdate);
```

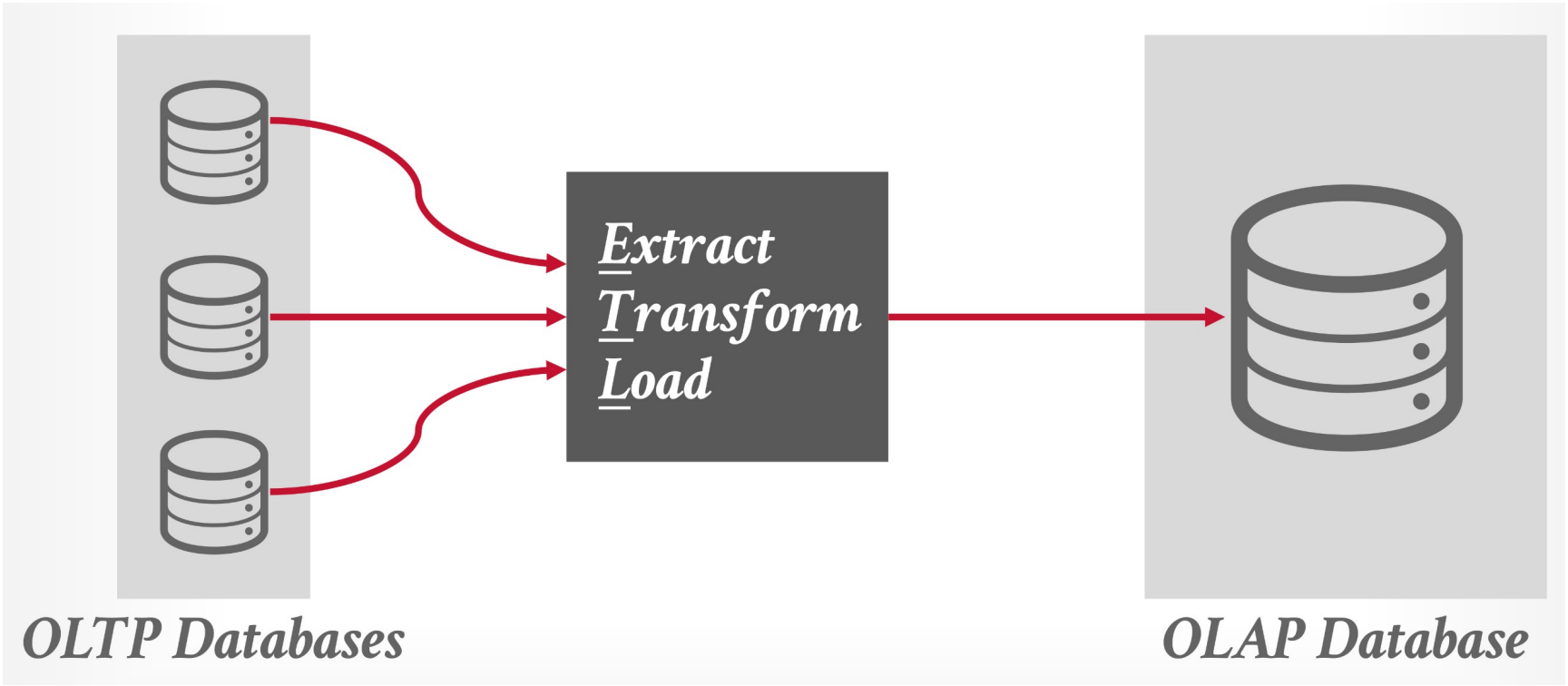
2000s – DATA WAREHOUSES

Monolithic DBMSs designed to efficiently execute OLAP workloads using shared-nothing architectures and column-oriented data.

- Many systems from this era started as forks of Postgres.

DBMS-managed storage using proprietary data encoding / formats.

2000s – DATA WAREHOUSES



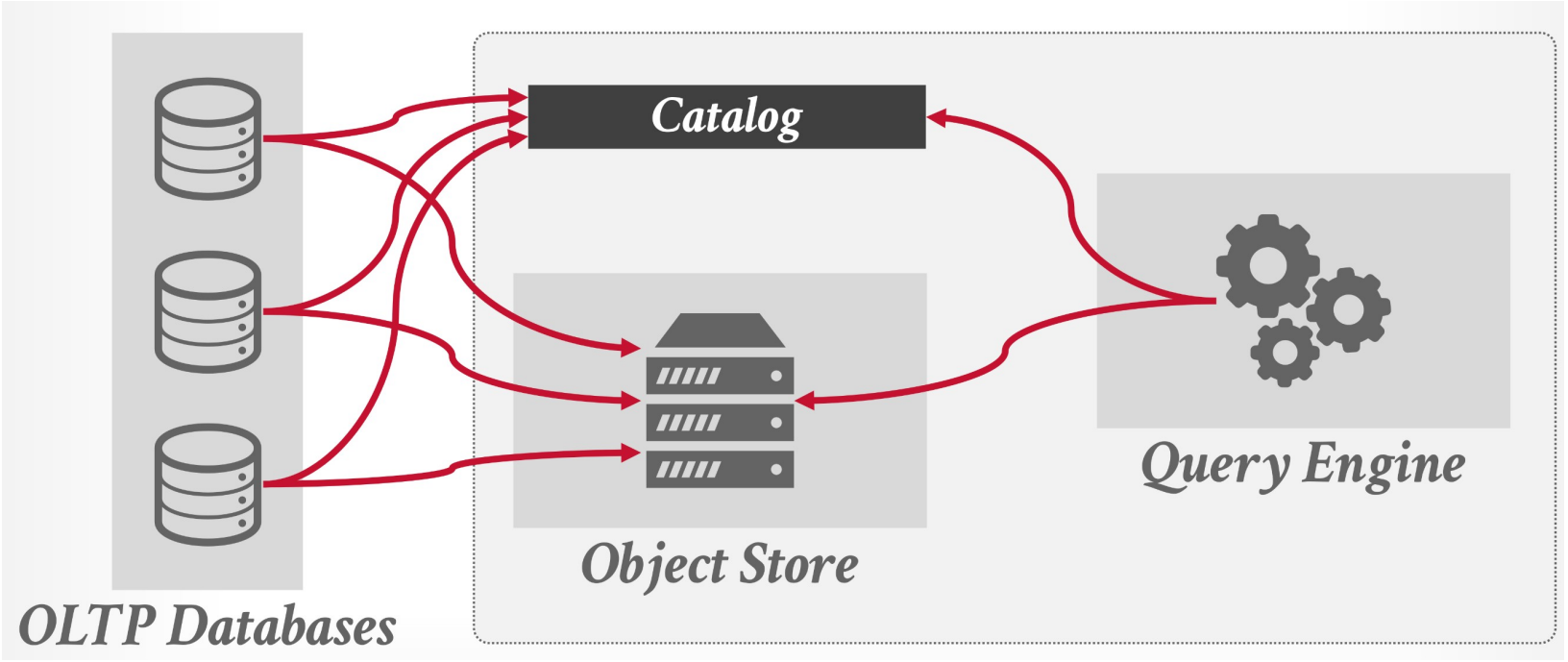
2010s – SHARED-DISK ENGINES

Shared-disk DBMS architectures that relied on third-party distributed storage (object stores) instead of using a custom storage manager.

- First generation of these systems managed data files themselves.
- Detaching storage from DBMS. Optimize compute level!

Newer systems allow external entities to add new data files to storage without enforcing schema (lakehouse)

2010s – SHARED-DISK ENGINES



2020s – LAKEHOUSE SYSTEMS

Middleware for data lakes that adds support for better schema control / versioning with transactional CRUD operations.

- Store changes in row-oriented log-structured files with indexes.
- Periodically compact recently added data into read-only columnar files.

2020s – LAKEHOUSE SYSTEMS

Observation #1:

- People want to execute more than just SQL on data.

Observation #2:

- Decoupling data storage from DBMS reduces ingest/egress barriers.

Observation #3:

- Most data is unstructured / semi-structured.

Armbrust-Etal-Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics-CIDR21

OLAP DBMS COMPONENTS

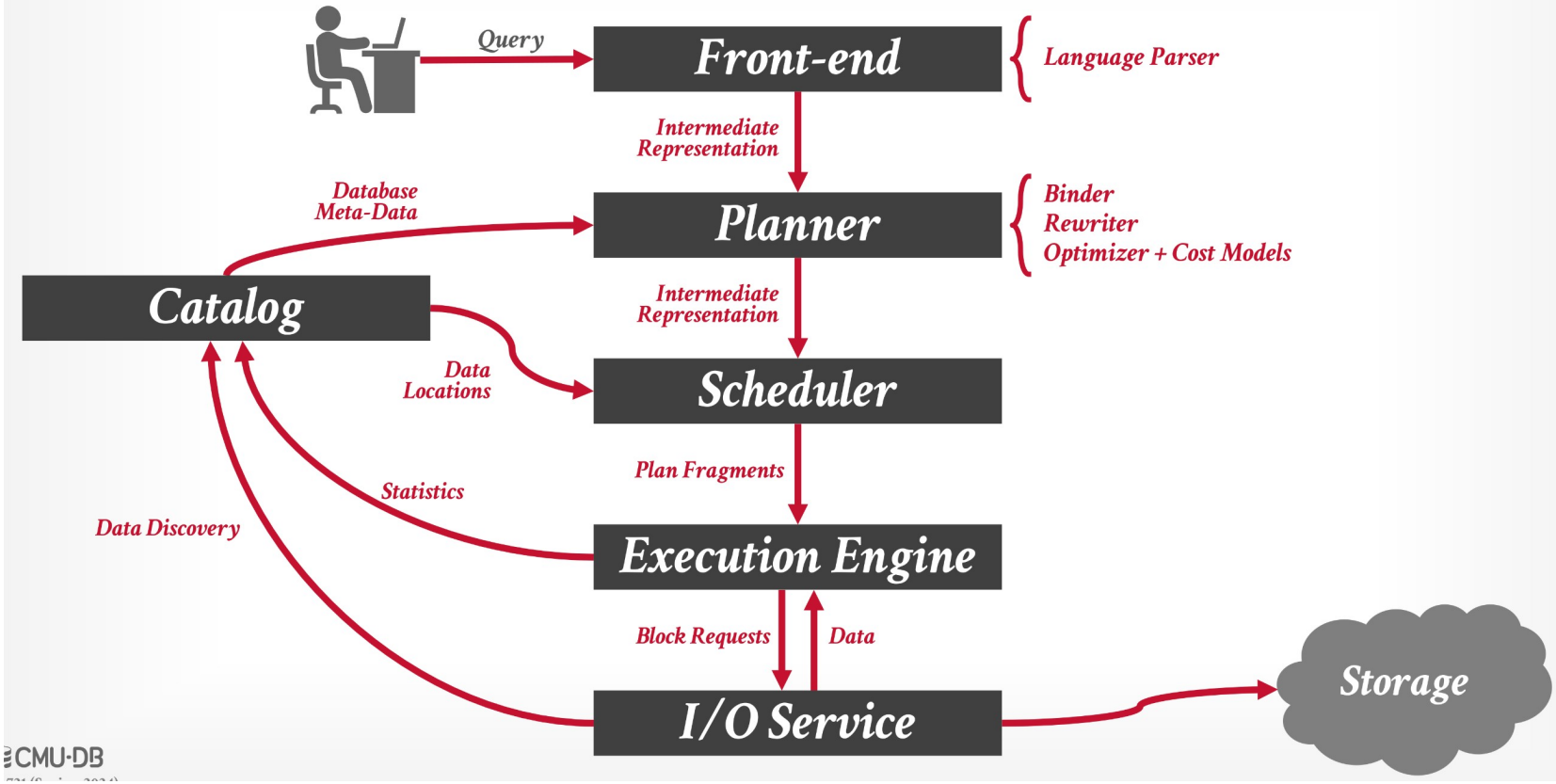
One recent trend of the last decade is the breakout of OLAP DBMS components into standalone services and libraries:

- System Catalogs
- Intermediate Representation
- Query Optimizers
- File Format / Access Libraries
- Execution Engines / Fabrics

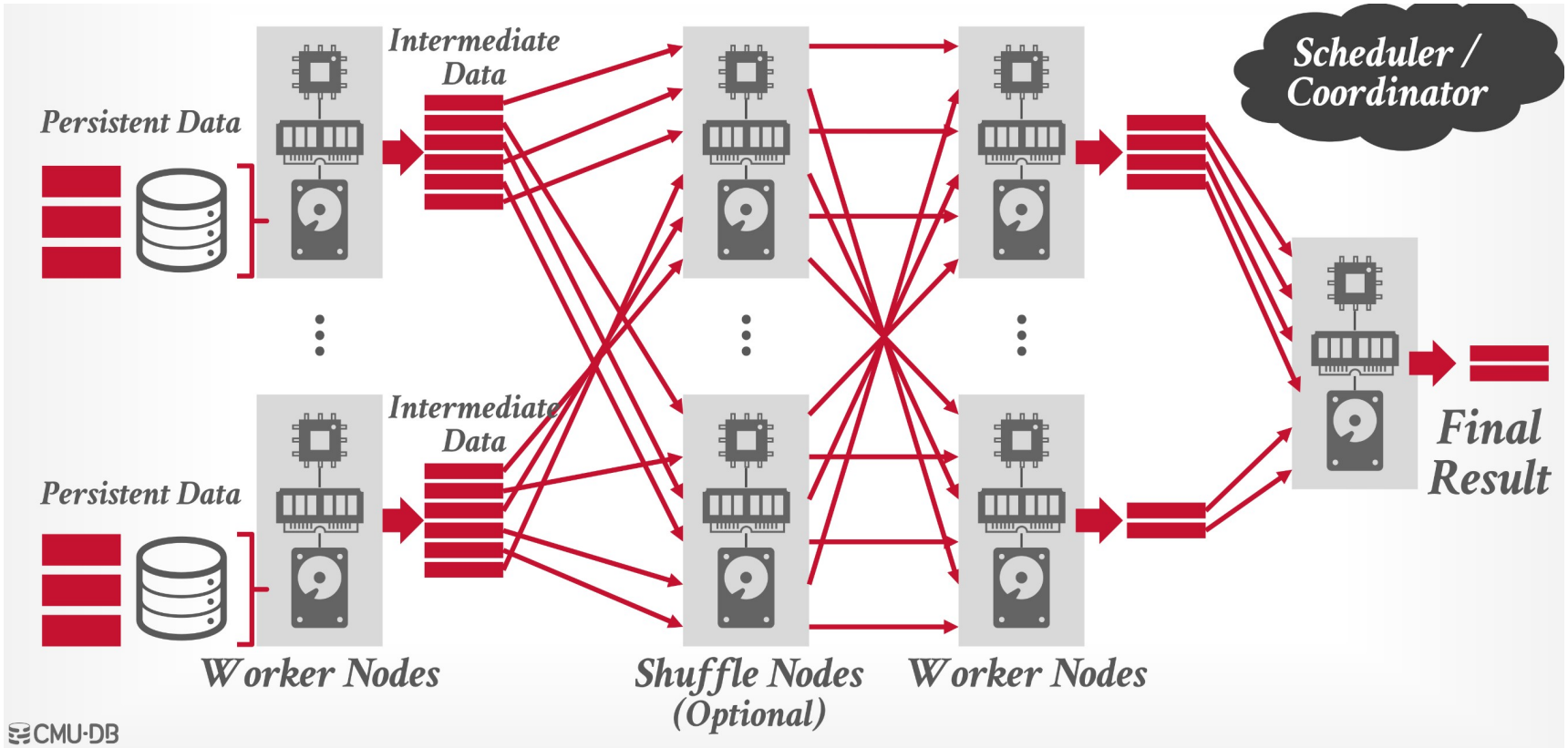
Lots of engineering challenges to make these components interoperable + performant.

Liu-Etal-A Deep Dive into Common Open Formats for Analytical DBMSs-VLDB23

ARCHITECTURE OVERVIEW



DISTRIBUTED QUERY EXECUTION



DATA CATEGORIES

Persistent Data:

- The "source of record" for the database (e.g., tables).
- Modern systems assume that these data files are immutable but can support updates by rewriting them.

Intermediate Data:

- Short-lived artifacts produced by query operators during execution and then consumed by other operators.
- The amount of intermediate data that a query generates has little to no correlation to amount of persistent data that it reads or the execution time.

Vuppalapati-Etal-Building An Elastic Query Engine on Disaggregated Storage-nsdi22

DISTRIBUTED SYSTEM ARCHITECTURE

A distributed DBMS's system architecture specifies the location of the database's persistent data files.

This affects how nodes coordinate with each other and where they retrieve/store objects in the database.

Two approaches (not mutually exclusive):

- **Push Query to Data**
- **Pull Data to Query**

PUSH VS. PULL

Approach #1: Push Query to Data

- Send the query (or a portion of it) to the node that contains the data.
- Perform as much filtering and processing as possible where data resides before transmitting over network.

Approach #2: Pull Data to Query

- Bring the data to the node that is executing a query that needs it for processing.
- This is necessary when there is no compute resources available where persistent data files are located.

PUSH QUERY TO DATA

Send a query tree, or a portion of it, to the node where the data is located.

- Query is smaller than the data.

Locality is important!

- Do filtering and projecting locally.
- Optimal amount of data is copied (data that is needed),

In case data in main memory

- Memory access will allways be faster then network, hence this invariant remains.

In case data in Amazon S3

- Simple Storage Service (S3)
- Because of shared-disk the access is not so fast!

PULL (MOVE) DATA TO QUERY

Disaggregation of storage, memory and compute.

- Ford's statement: *"Every complex problem can be solved by breaking it down into manageable steps."*
- Storing data in-between steps – PULL approach is neccessary!
- **Google's Dremel**: will be studied thoroughly.

Filtering while reading from object store

- S3 allow low-level filtering on data access
- Microsoft Azure allows simple SQL queries on blobs

PUSH VS. PULL: CONCLUSIONS

There are two competing approaches

- PUSH: Parallel and distributed approach
  - Locality: data access and data processing on one node
  - Often requires careful distribution of data!
- PULL: Dremel's approach (Google)
  - Steps in computation of a query
  - Automatic distribution of data

Many systems use hybrid approach.

SHARED-NOTHING

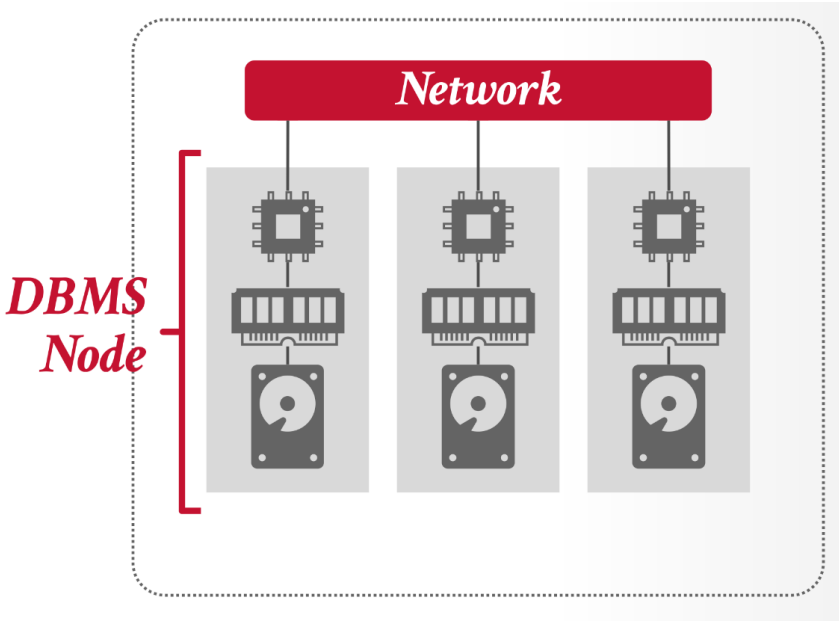
Each DBMS instance has its own CPU, memory, locally-attached disk.

- Nodes only communicate with each other via network.

Database is partitioned into disjoint subsets across nodes.

- Adding a new node requires physically moving data between nodes.

Since data is local, the DBMS can access it via POSIX API.

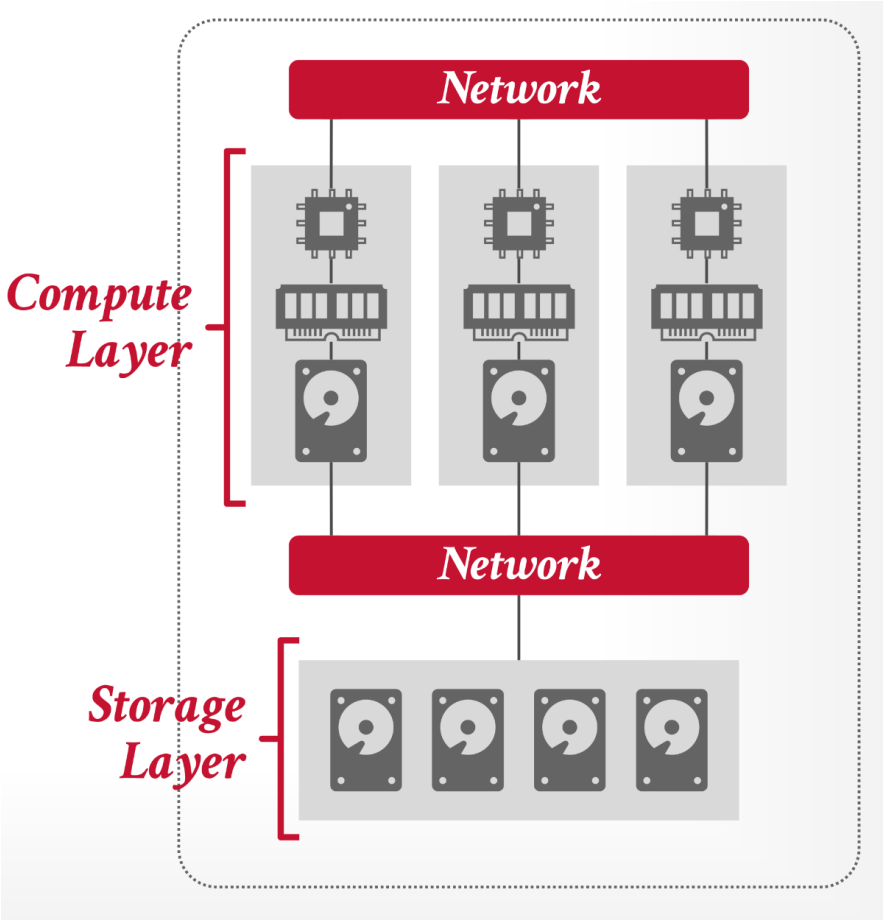


SHARED-DISK

Each node accesses a single logical disk via an interconnect, but also have their own private memory and ephemeral storage.

- Must send messages between nodes to learn about their current state.

Instead of a POSIX API, the DBMS accesses disk using a userspace API.



SYSTEM ARCHITECTURE

Choice #1: Shared-Nothing:

- Harder to scale capacity due to data movement.
- Potentially better performance & efficiency.
- Apply filters where the data resides before transferring.

Choice #2: Shared-Disk:

- Scale compute layer independently from storage layer.
- Easy to shutdown idle compute layer resources.
- May need to pull uncached persistent data from storage layer to compute layer before applying filters.

SHARED-DISK IMPLEMENTATIONS

Traditionally the storage layer in shared-disk DBMSs were dedicated on-prem NAS.

- Example: Oracle Exadata

Cloud **object stores** are now the prevailing storage target for modern OLAP DBMSs because they are "infinitely" scalable.

- Examples: Amazon S3, Azure Blob, Google Cloud Storage

OBJECT STORES

Partition the database's tables (persistent data) into large, immutable files stored in an object store.

- All attributes for a tuple are stored in the same file in a columnar layout (PAX).
- Header (or footer) contains meta-data about columnar offsets, compression schemes, indexes, and zone maps.

The DBMS retrieves a block's header to determine what byte ranges it needs to retrieve (if any).

Each cloud vendor provides their own proprietary API to access data (PUT, GET, DELETE).

CONCLUSION

Today was about understanding the high-level context of what modern OLAP DBMSs look like.

- Fundamentally these new DBMSs are not different than previous distributed/parallel DBMSs except for the prevalence of a cloud-based object store for shared disk.

Our focus for the rest of the semester will be about state-of-the-art implementations of these systems' components.

NEXT CLASS

- Storage Models
- Data Representation
- Encoding
- Compression