01: Introduction

Andrew Crotty // CS497 // Fall 2023

Today's Agenda

Course Logistics
History of Database Systems
Modern OLAP

Course Objectives

Comprehensive overview of the internals of modern OLAP DBMSs.

Students will learn to:
→ Read + evaluate systems research papers
→ Identify system tradeoffs and justify design decisions
→ Craft + deliver presentations to convey research ideas
→ Plan + execute a final project that answers an interesting systems research question

This is **not** a course about classical DBMSs.

Background

This course is designed for students interested in systems research (grads + advanced undergrads).

I assume you have already taken an intro DB systems course (e.g., CS339) or equivalent.

Things that we will <u>not</u> cover:
Intro DBMS concepts like SQL, Serializability Theory, Relational Algebra, Basic Algorithms + Data Structures, etc.

Topics

Data Storage

Compression

Vectorized Execution

Query Compilation

Parallel Join Algorithms

Index Structures

Course Logistics

Course Policies + Schedule:

→ Refer to course web page

Academic Integrity:

→ Refer to Northwestern policy page
→ If you're not sure, ask me.
→ Seriously, don't plagiarize or you **will** get wrecked.

Grading Rubric

Compression Project – 40%
Encyclopedia Article – 10%
Final Project – 50%

Compression Project

Groups of 1-2 students will be given 3 real-world datasets to compress as much as possible.

Deliverables include:

→ **Project presentation** (5-10 minutes, modeled after a short conference talk) – 5%
→ **Written report** (at least 4 pages excluding references, modeled after a workshop paper) – 10%
→ **Programming component** (code, benchmarks, demo, etc.) – 25%

Encyclopedia Article

The Database of Databases is an online encyclopedia of DBMSs maintained by the CMU DB Group.

Groups of 1-2 students will write an article.
→ Must provide citations and attributions
→ Avoid unscientific (i.e., marketing) language

You may **not** copy text / images directly from papers or other sources.

Final Project

Groups of 1-2 will complete a final project on an approved topic related to the course content.

Deliverables include:

→ **Project proposal** (5-10 minutes, modeled after a short conference talk) – 5%

→ **Project presentation** (20-30 minutes, modeled after a full conference talk) – 10%

→ **Written report** (at least 6 pages excluding references, modeled after a conference paper) – 10%

→ **Programming component** (code, benchmarks, demo, etc.) – 25%

HISTORY OF DATABASE SYSTEMS

WHAT GOES AROUND COMES AROUND
READINGS IN DB SYSTEMS, 4TH EDITION, 2006.

WHAT GOES AROUND COMES AROUND… AND AROUND
UNDER SUBMISSION 2023

History Repeats Itself

Old issues are still relevant today. Many of the ideas in today's systems are not new.

Every decade, someone invents a SQL replacement and some combination of the following happens.
→ It fails.
→ The project slowly adds SQL features.
→ The SQL standard absorbs the best parts.

1960s – IDS

Integrated **D**ata **S**tore

Developed internally at GE in early 1960s.

GE sold their computing division to Honeywell in 1969.

One of the first DBMSs.

1960s – CODASYL

COBOL people got together and proposed a standard for how programs will access a database. Lead by Charles Bachman.

→ Network data model

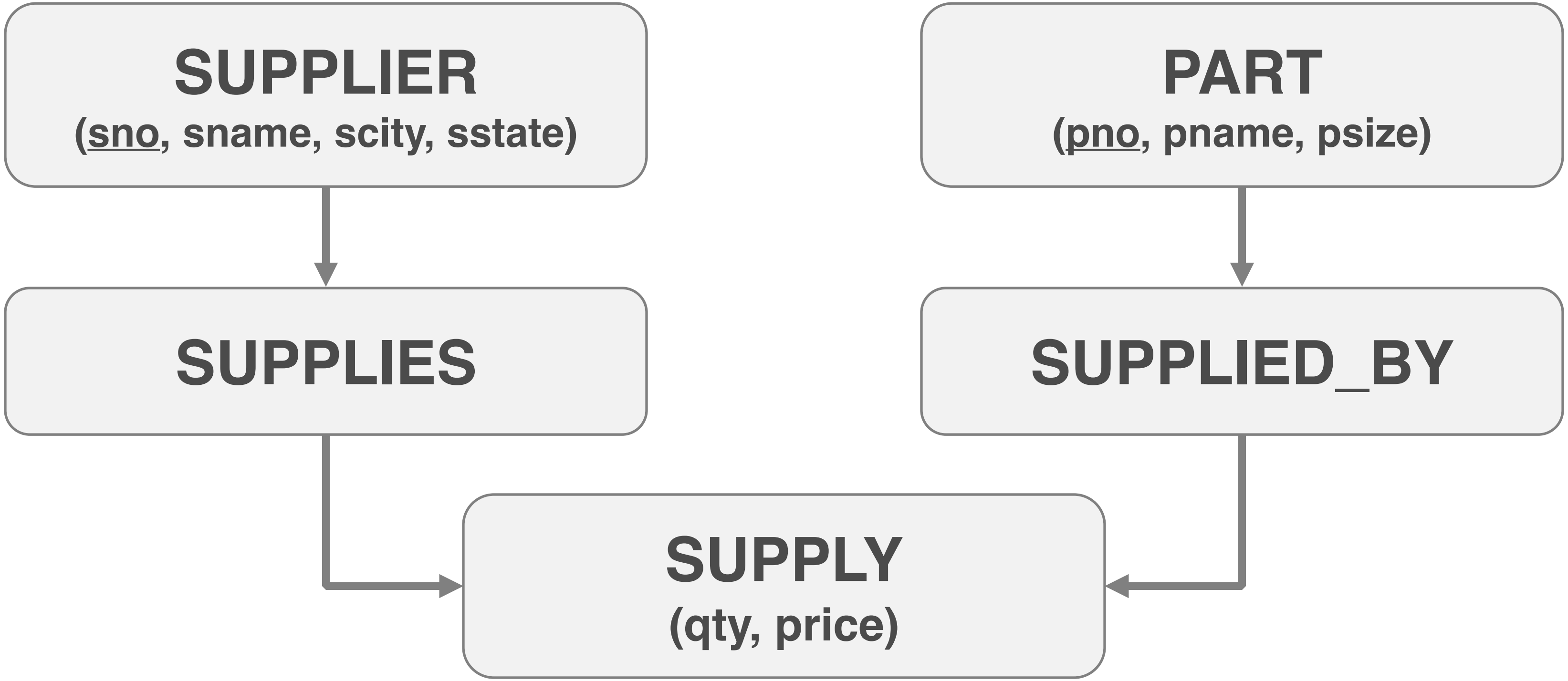
→ Tuple-at-a-time queries

Bachman

Bachman also worked at Culliane Database Systems in the 1970s to help build **IDMS**.

Network Data Model

Schema



Network Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

qty	price
10	$100
14	$99

Network Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLIES

parent	child

SUPPLIED_BY

parent	child

SUPPLY

qty	price
10	$100
14	$99

Network Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

SUPPLIES

parent	child

SUPPLY

qty	price
10	$100
14	$99

PART

pno	pname	psize
999	Batteries	Large

SUPPLIED_BY

parent	child

Network Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLIES

parent	child

SUPPLIED_BY

parent	child

SUPPLY

qty	price
10	$100
14	$99

1960S – IBM IMS

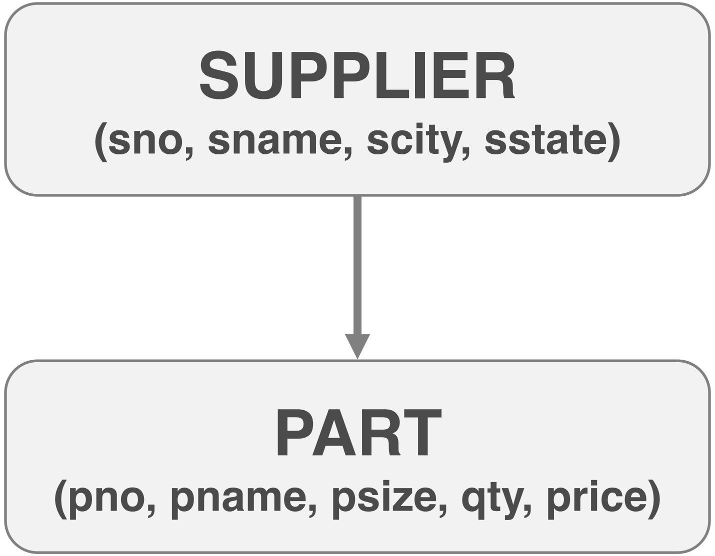
Information **M**anagement **S**ystem

Early DBMS developed to keep track of purchase orders for Apollo moon mission.

→ Hierarchical data model.
→ Programmer-defined physical storage format.
→ Tuple-at-a-time queries.

Hierarchical Data Model

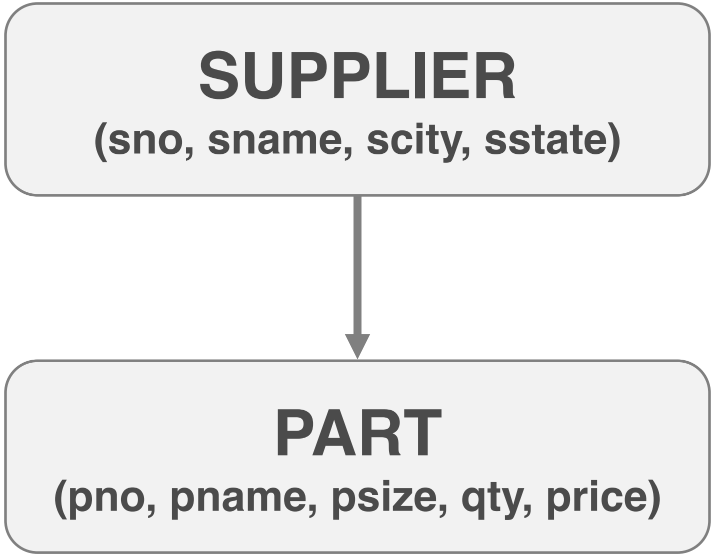
Schema



Instance

Hierarchical Data Model

Schema



Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

Hierarchical Data Model

Schema

SUPPLIER

(sno, sname, scity, sstate)

PART

(pno, pname, psize, qty, price)

Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

pno	pname	psize	qty	price
999	Batteries	Large	10	$100

Hierarchical Data Model

Schema

SUPPLIER
(sno, sname, scity, sstate)

↓

PART
(pno, pname, psize, qty, price)

Instance

sno	sname	scity	sstate	parts
1001	Dirty Rick	New York	NY	
1002	Squirrels	Boston	MA	

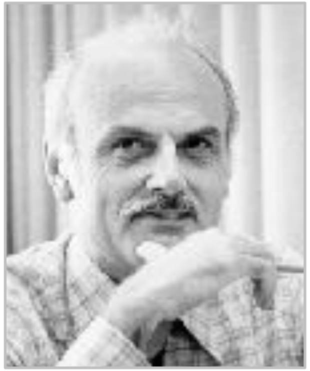
pno	pname	psize	qty	price
999	Batteries	Large	10	$100

pno	pname	psize	qty	price
999	Batteries	Large	14	$99

1970s – Relational Model

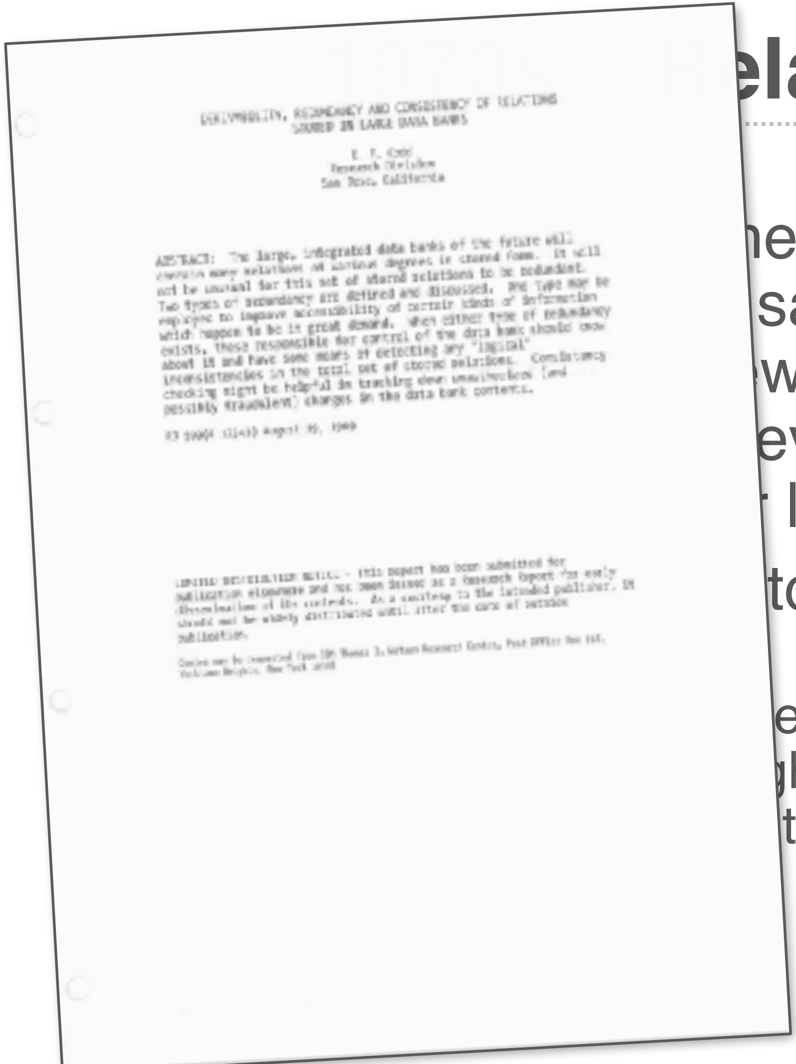
Ted Codd was a mathematician working at IBM Research. He saw developers spending their time rewriting IMS and CODASYL programs every time the database's schema or layout changed.

Database abstraction to avoid this maintenance:
→ Store database in simple data structures.
→ Access data through high-level language.
→ Physical storage left up to implementation.



Codd

elational Model



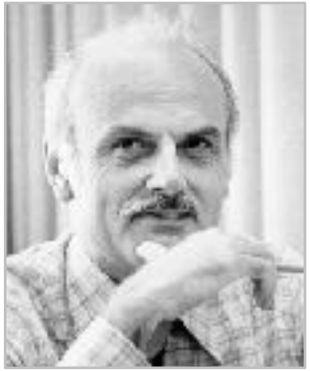
DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS
STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

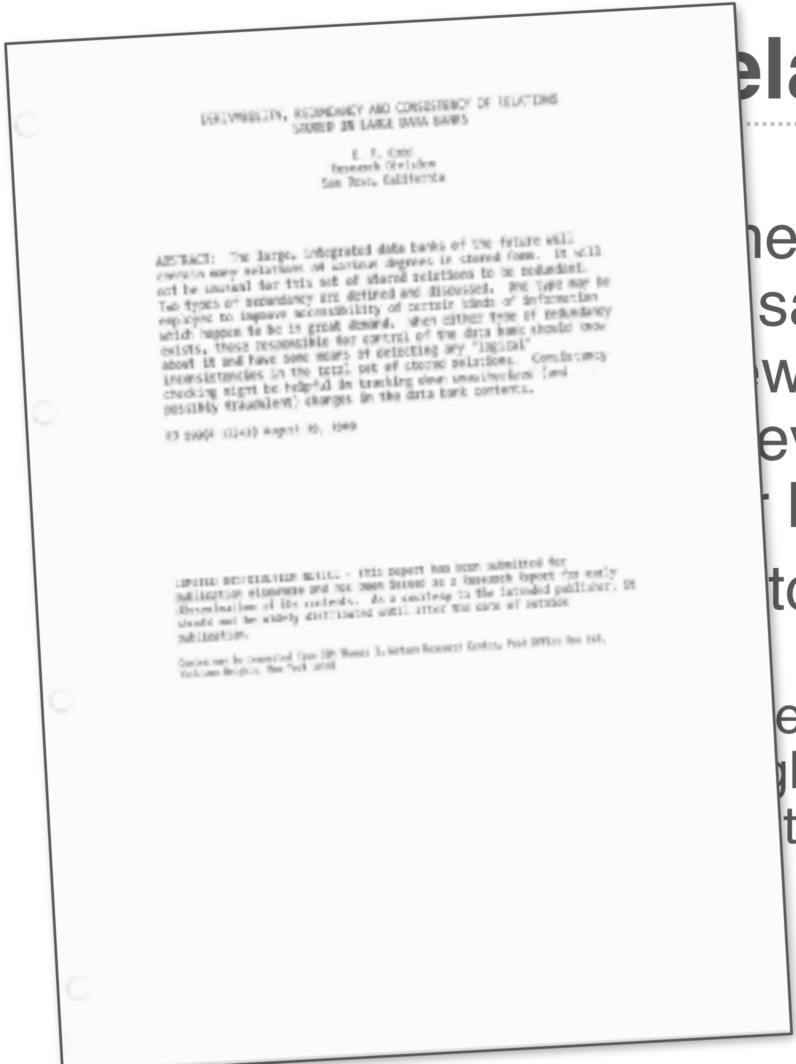
hematician working
saw developers
writing IMS and
every time the
layout changed.

to avoid this

e data structures.
gh-level language.
to implementation.



Codd



DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

1. Relational Model and Normal Form

Relational Data Model

Schema

Relational Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

sno	pno	qty	price
1001	999	10	$100
1002	999	14	$99

Relational Data Model

Instance

SUPPLIER

sno	sname	scity	sstate
1001	Dirty Rick	New York	NY
1002	Squirrels	Boston	MA

PART

pno	pname	psize
999	Batteries	Large

SUPPLY

sno	pno	qty	price
1001	999	10	$100
1002	999	14	$99

1970s – Relational Model

Early implementations of relational DBMS:

→ **Peterlee Relational Test Vehicle** – IBM Research (UK)
→ **System R** – IBM Research (San Jose)
→ **INGRES** – U.C. Berkeley
→ **Oracle** – Larry Ellison
→ **Mimer** – Uppsala University


Gray


Stonebraker


Ellison

1980s – Relational Model

The relational model wins.
→ IBM first releases SQL/DS in 1981.
→ IBM then releases DB2 in 1983.
→ "SEQUEL" becomes the standard (SQL) after supposedly Stonebraker refused to talk to the ANSI standards committee.

Many new "enterprise" DBMSs but Oracle wins marketplace.

Stonebraker creates Postgres as an "object-relational" DBMS.

1980s – Relational Model

The relational model wins.
→ IBM first releases SQL/DS in 1
→ IBM then releases DB2 in 198
→ "SEQUEL" becomes the stand
after supposedly Stonebraker
to the ANSI standards commit

Many new "enterprise" DBM
but Oracle wins marketplac

Stonebraker creates Postgres as an "object-relational" DBMS.

But Ingres did not show up at the committee meetings because founder Mike Stonebraker detested the idea of having technology standards. Stonebraker was vocal about it. He thought they inhibited innovation and artificially restricted what got to the marketplace. Maybe so, but his hard-line position probably did not help his company. Don Deutsch, who served as chairman of the database committee, summed things up this way: "I tell you, QUEL was a much nicer language than SQL. No rational person would have chosen SQL instead of QUEL. . . . Ingres was stupid."

1980s – Object-Oriented Databases

Avoid "relational-object impedance mismatch" by tightly coupling objects and database.

Few of these original DBMSs from the 1980s still exist today but many of the technologies exist in other forms (e.g., XML, JSON).

VERSANT ObjectStore. MarkLogic

Object-Oriented Model

Application Code

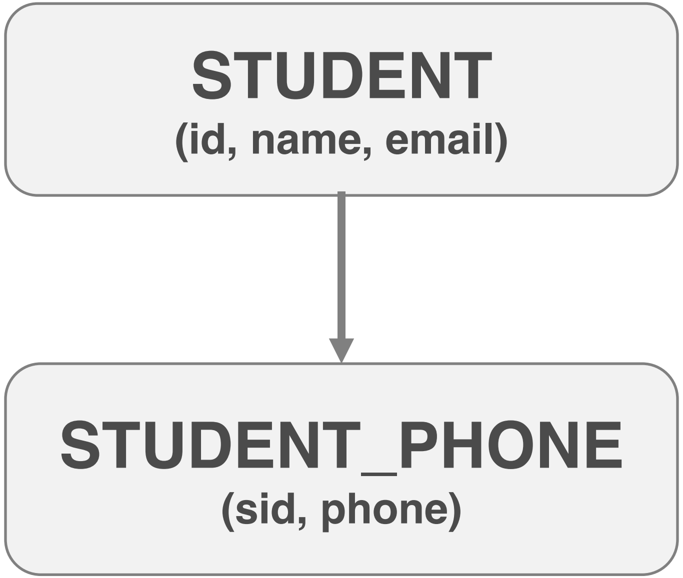
```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

Object-Oriented Model

Application Code

```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

Relational Schema



Object-Oriented Model

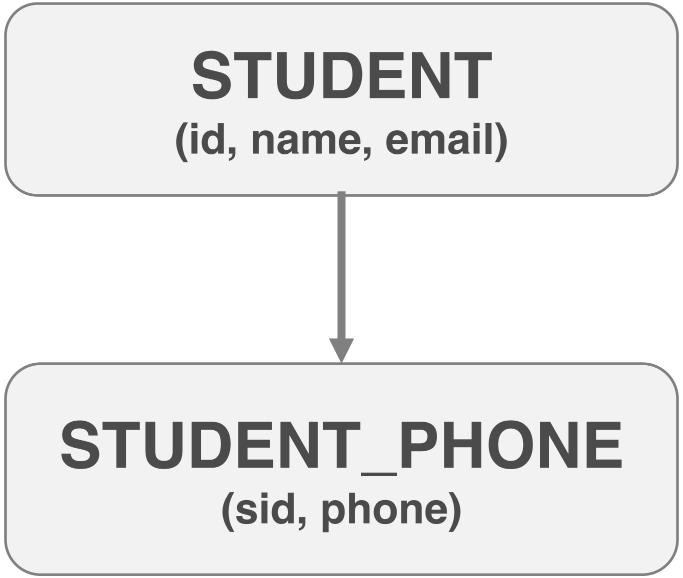
Application Code

```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

id	name	email
1001	M.O.P.	ante@up.com

sid	phone
1001	444-444-4444
1001	555-555-5555

Relational Schema



Object-Oriented Model

Application Code

```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

id	name	email
1001	M.O.P.	ante@up.com

sid	phone
1001	444-444-4444
1001	555-555-5555

Relational Schema

STUDENT
(id, name, email)

STUDENT_PHONE
(sid, phone)

Object-Oriented Model

Application Code

```
class Student {
  int id;
  String name;
  String email;
  String phone[];
}
```

Object-Oriented Model

Application Code

```
class Student {
   int id;
   String name;
   String email;
   String phone[];
}
```

Student

```
{
  "id": 1001,
  "name": "M.O.P.",
  "email": "ante@up.com",
  "phone": [
     “444-444-4444”,
     “555-555-5555”
  ]
}
```

1990s – Boring Years

No major advancements in database systems or application workloads.
→ Microsoft forks Sybase and creates SQL Server.
→ MySQL is written as a replacement for mSQL.
→ Postgres gets SQL support.
→ SQLite started in early 2000.

Some DBMSs introduced pre-computed data cubes for faster analytics.

2000s – Internet Boom

All the big players were heavyweight and expensive. Open-source DBMSs were missing important features.

Many companies wrote their own custom middleware to scale across many independent single-node DBMS instances.

2000s – Data Warehouses

Rise of the special purpose OLAP DBMSs.
→ Distributed / Shared-Nothing
→ Relational / SQL
→ Usually closed-source

Significant performance benefits from using **columnar data storage** model.

2000s – MapReduce

Distributed programming and execution model for analyzing large data sets.
→ First proposed by Google (**MapReduce**).
→ Yahoo! created an open-source version (**Hadoop**).
→ Data model decided by user-written functions.

People (eventually) realized this was a bad idea and grafted SQL on top of MapReduce. That was a bad idea too.

2000s – NoSQL

Focus on high-availability & high-scalability:

→ Schema-less (i.e., "Schema Last")
→ Non-relational data models (document, key/value, column-family)
→ No ACID transactions
→ Custom APIs instead of SQL
→ Usually open-source

2010s – NewSQL

Provide same performance for OLTP as NoSQL without giving up ACID:

→ Relational / SQL

→ Distributed

Almost all of the first group of systems failed.

Second wave of "distributed SQL" systems are (potentially) doing better.

GenieDB dbShards H-Store Clustrix ScaleArc TRANSLATTICE VOLTDB FOUNDATIONDB Google Spanner MEMSQL codeFutures NUODB ScaleBase

2010s – NewSQL

Provide same performance for OLTP as NoSQL without giving up ACID:
→ Relational / SQL
→ Distributed

Almost all of the first group of systems failed.

Second wave of "distributed SQL" systems are (potentially) doing better.

2010s – HTAP

Hybrid **T**ransactional-**A**nalytical **P**rocessing

Execute fast OLTP like a NewSQL system while also executing complex OLAP queries like a data warehouse system.

→ Distributed / Shared-Nothing
→ Relational / SQL
→ Mixed open/closed-source.

MEMSQL HyPer SNAPPY SAP HANA
JustOne DB splice MACHINE Peloton

2010s – Stream Processing

Execute continuous queries on streams of tuples, extending semantics to include notion of windows.

Often used in combination with batch-oriented systems in a **lambda architecture**.

kafka samza PULSAR Flink APACHE STORM HERON

2010s – The Cloud

First database-as-a-service (DBaaS) offerings were "containerized" versions of existing DBMSs.

There are newer DBMSs that are designed from scratch explicitly for running in a cloud environment.

2010s – Shared-Disk

Instead of writing a custom storage manager, the DBMS leverages distributed storage.
→ Scale execution layer independently of storage.
→ Favors log-structured approaches.

This is what most people think of when they talk about a **data lake**.

2010s – Graphs

Systems for storing and querying graph data.
→ Similar to the network data model (CODASYL)

Their (supposed) advantage over other data models is to provide a graph-centric query API
→ SQL:2023 is adding graph query syntax (SQL/PCG)

Latest research (2023) shows that relational DBMSs outperform graph DBMSs.

2010s – Time Series

Specialized systems that are designed to store time series / event data.

The design of these systems make deep assumptions about the distribution of data and workload query patterns.

2020s – Blockchains

Decentralized distributed log with incremental checksums (Merkle Trees).
→ Uses Byzantine Fault Tolerant (BFT) protocol to determine next entry to append to log.

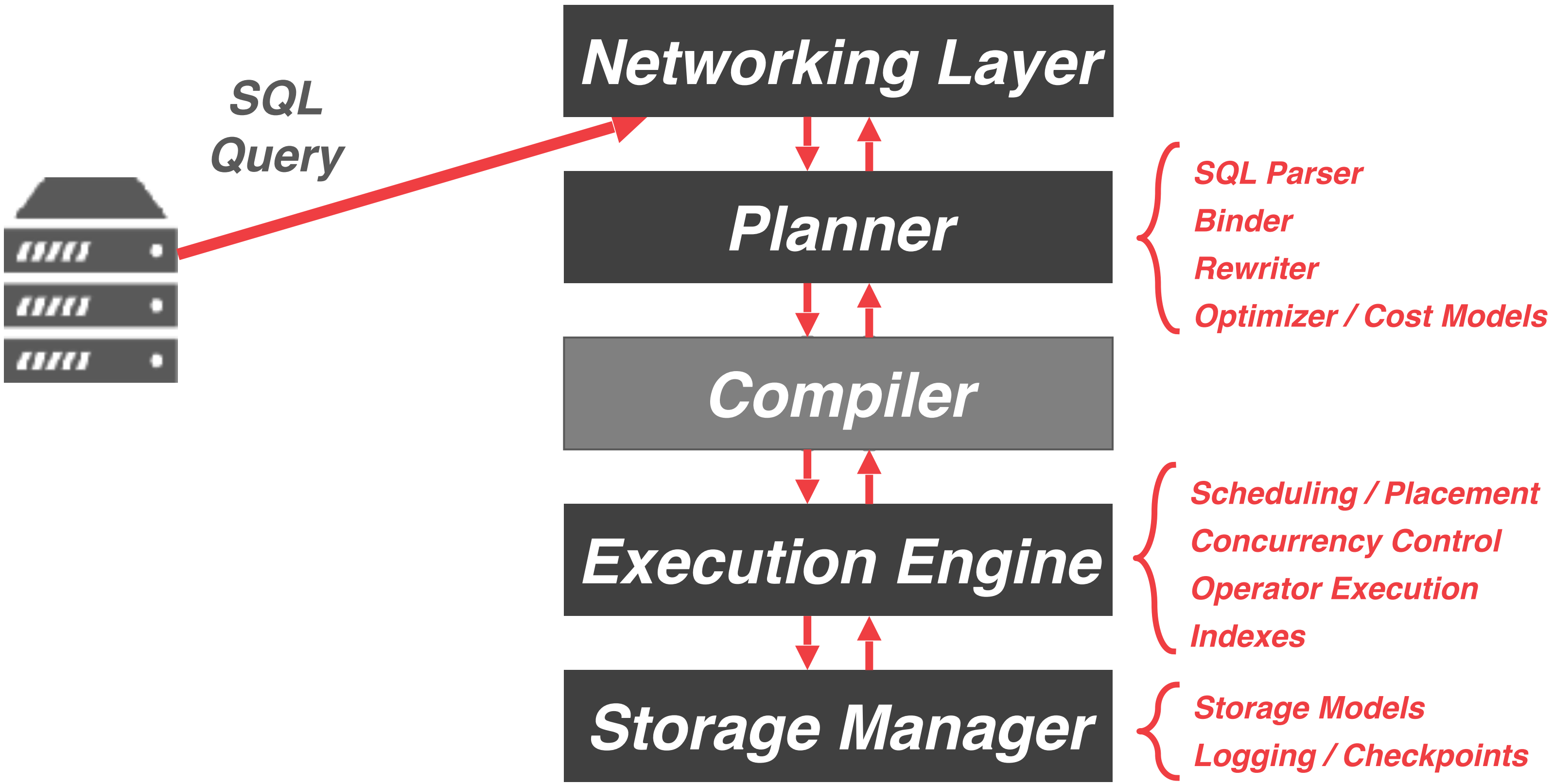
Many blockchain use cases seem like they can be solved with a "traditional" OLTP DBMS and/or external policies (e.g., authentication).

Current State of Affairs

The demarcation lines of DBMS categories will continue to blur over time as specialized systems expand the scope of their domains.
→ Every NoSQL DBMS (except for Redis) now supports SQL.

The relational model and declarative query languages promote better data engineering.

DBMS Overview

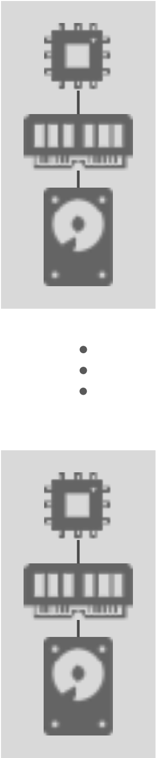


Distributed Query Execution

Executing OLAP queries in a distributed DBMS is roughly the same as on a single node.
→ Query plan is a DAG of physical operators.

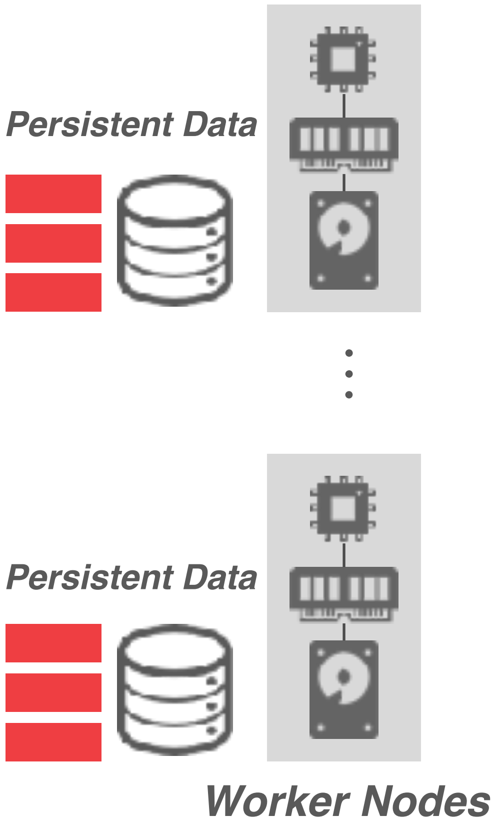
For each operator, the DBMS considers where input is coming from and where to send output.
→ Table Scans
→ Joins
→ Aggregations
→ Sorting

Distributed Query Execution

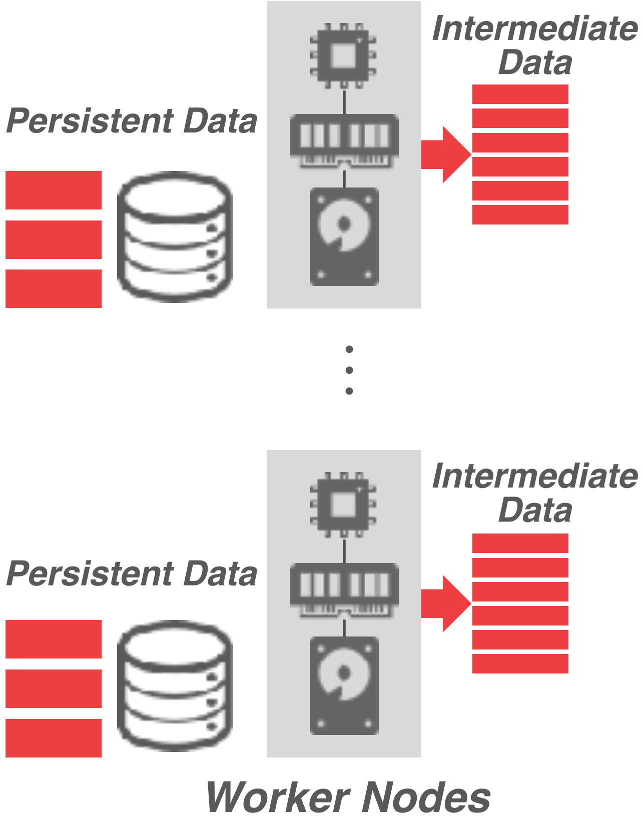


Worker Nodes

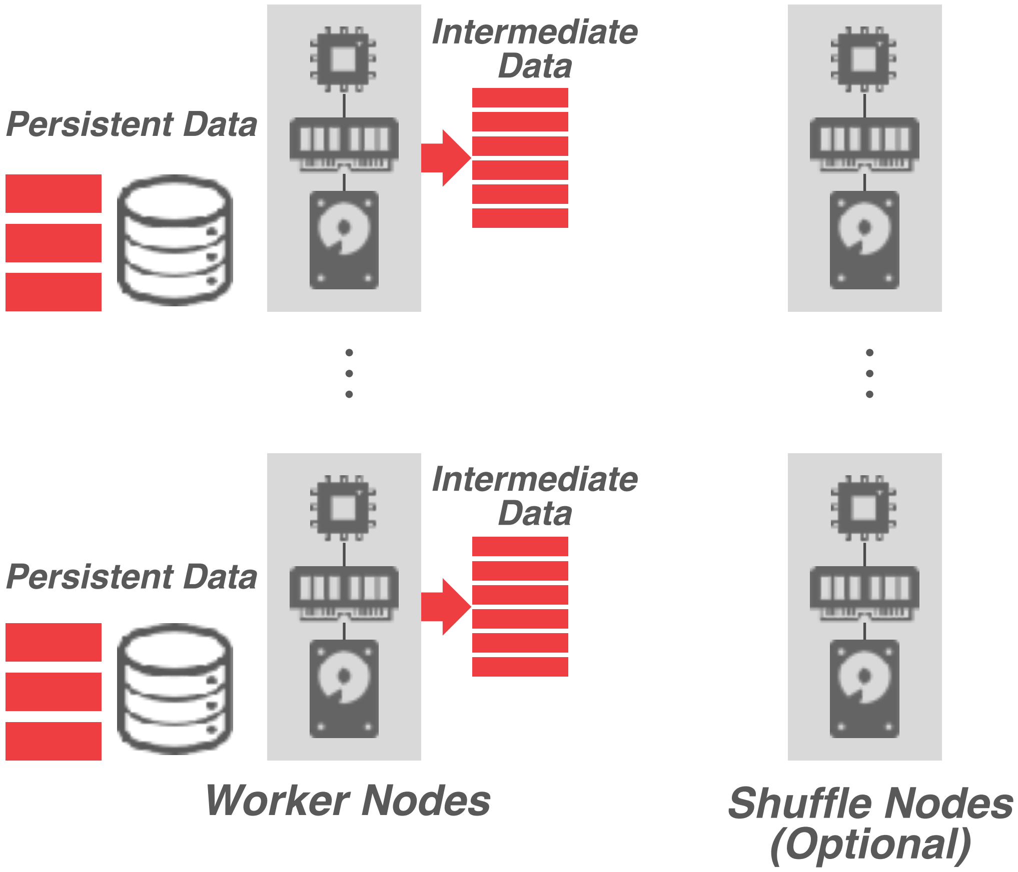
Distributed Query Execution



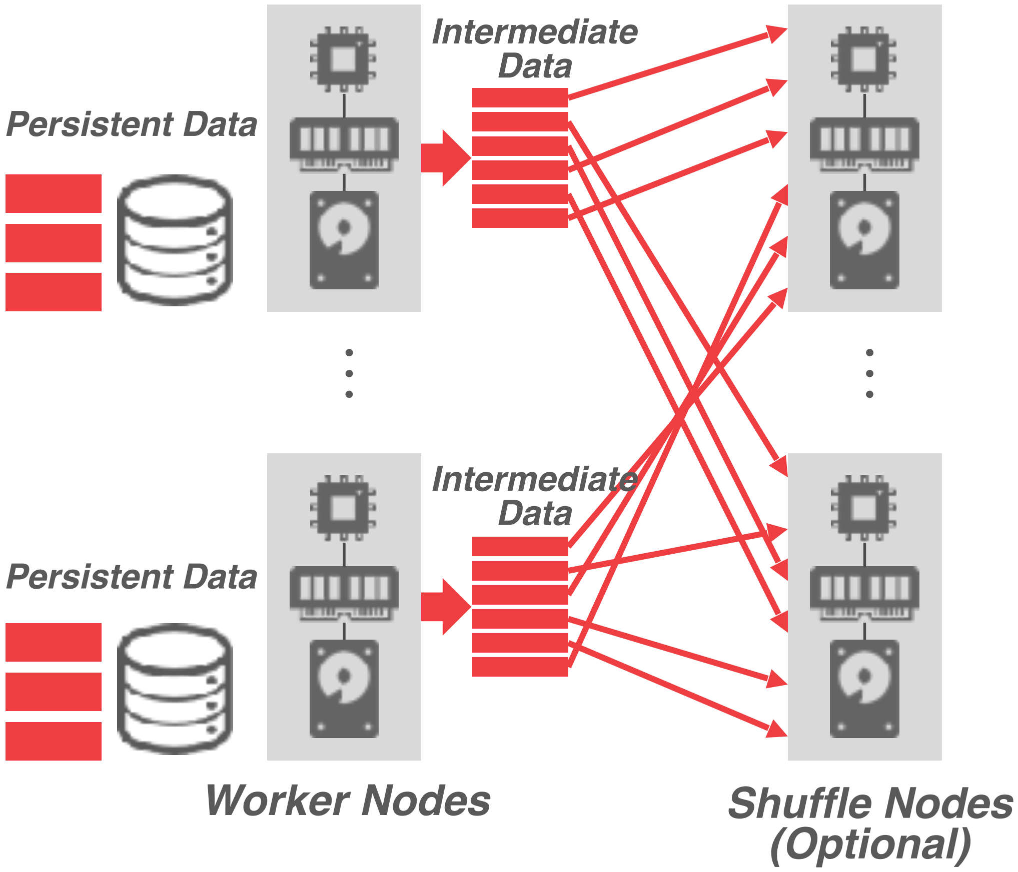
Distributed Query Execution



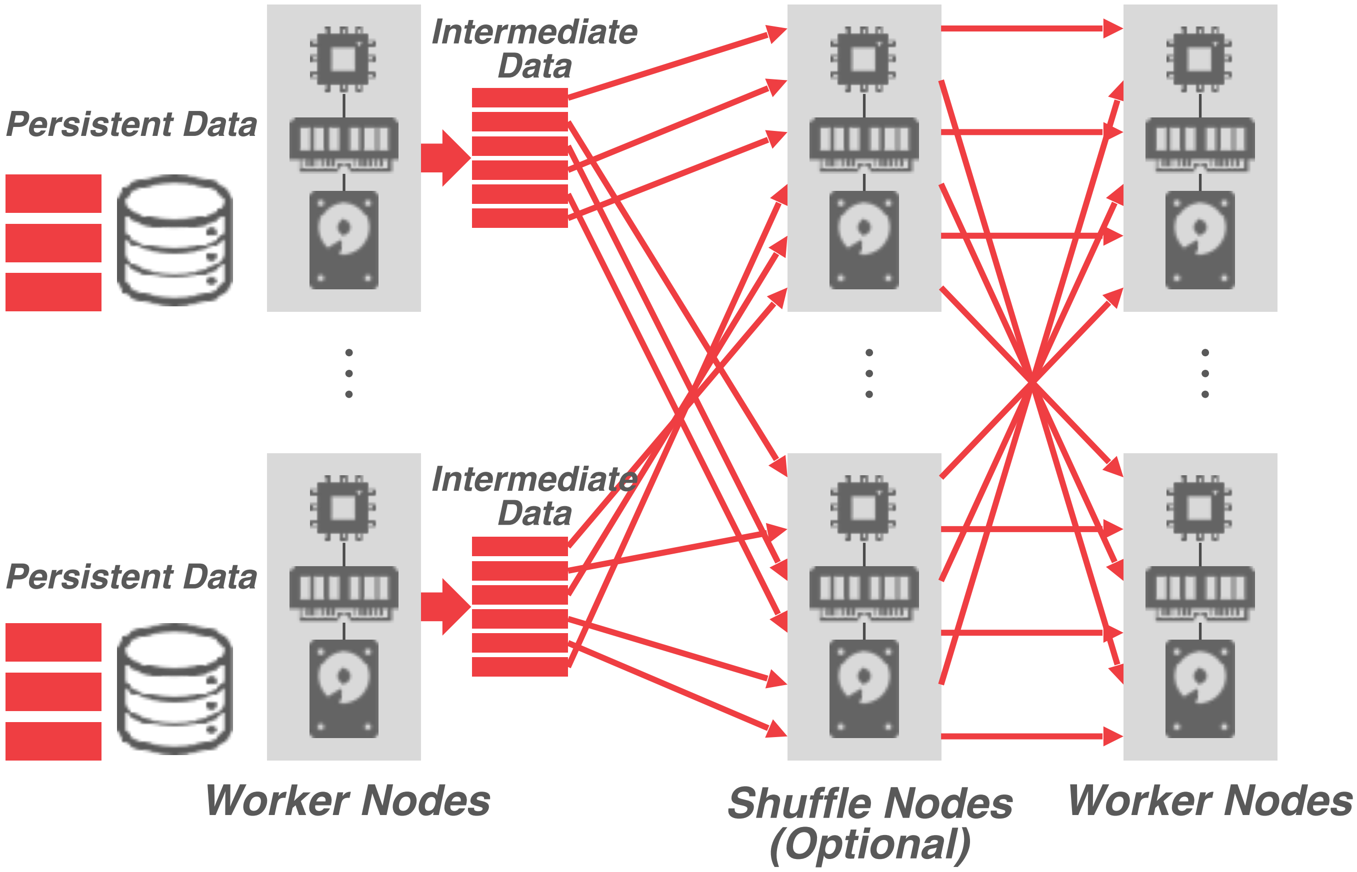
Distributed Query Execution



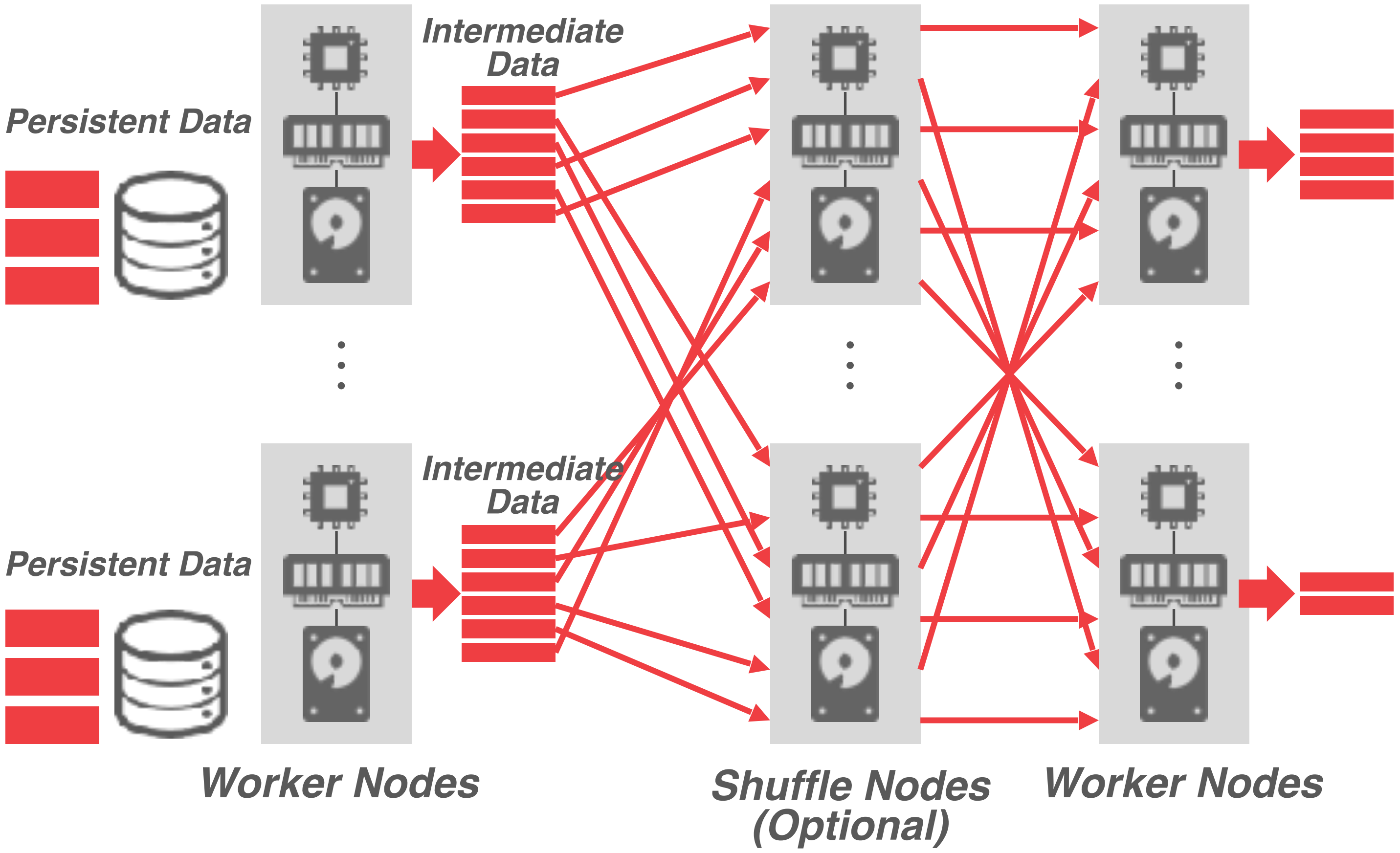
Distributed Query Execution



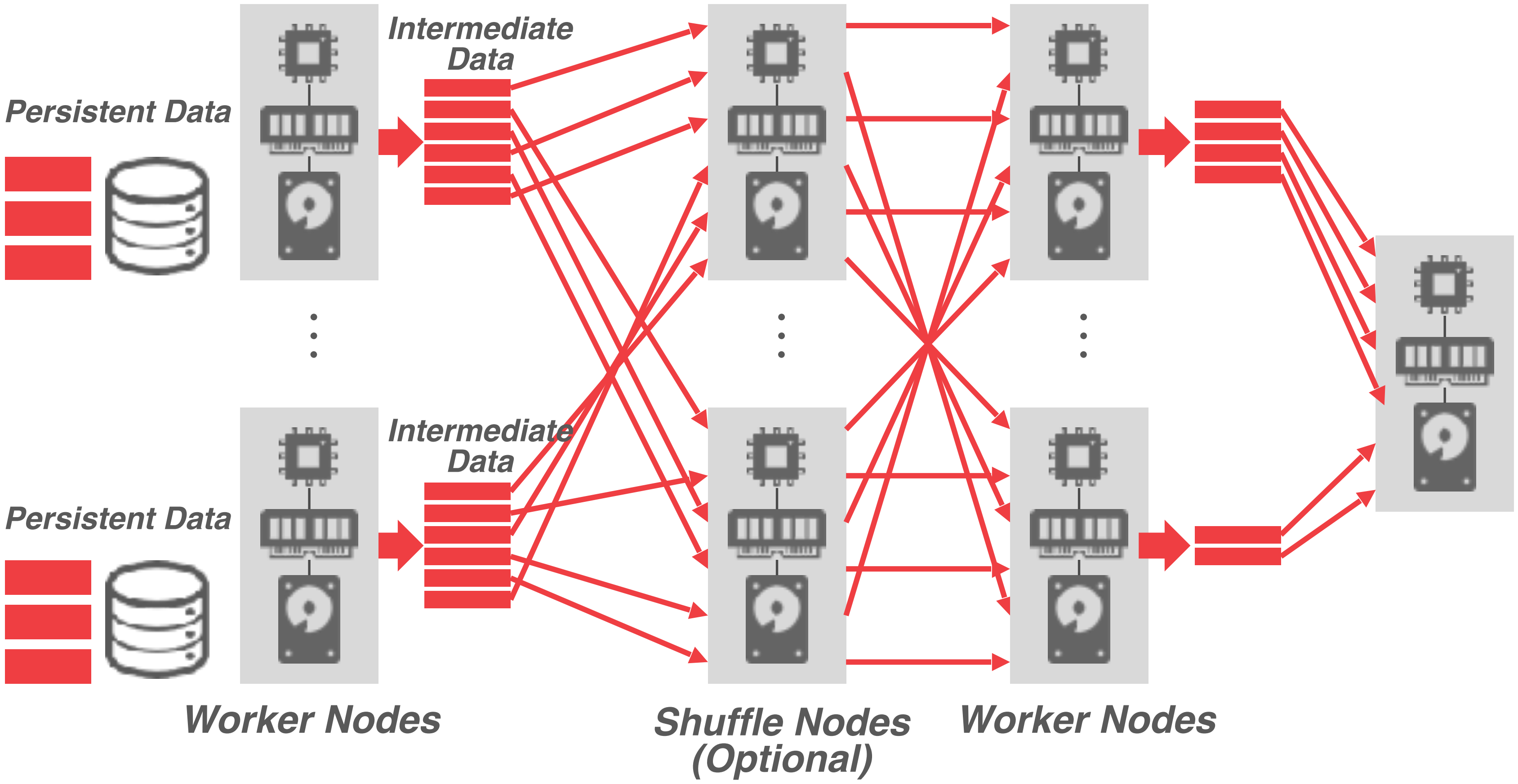
Distributed Query Execution



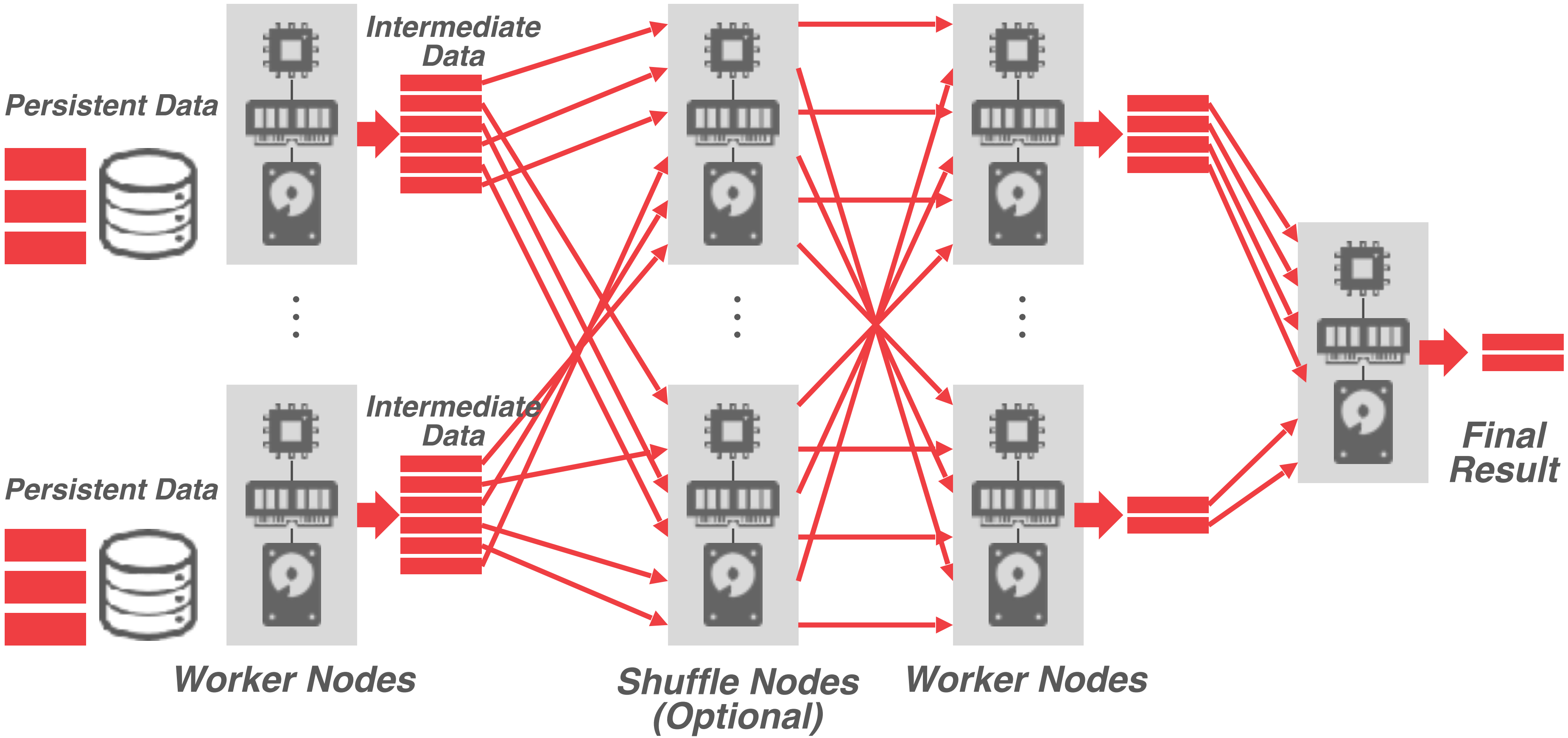
Distributed Query Execution



Distributed Query Execution



Distributed Query Execution



Types of Data

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 correlation to amount of persistent data that it reads or the overall execution time.

Distributed 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**

THE CASE FOR SHARED NOTHING
HPTS 1985

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.

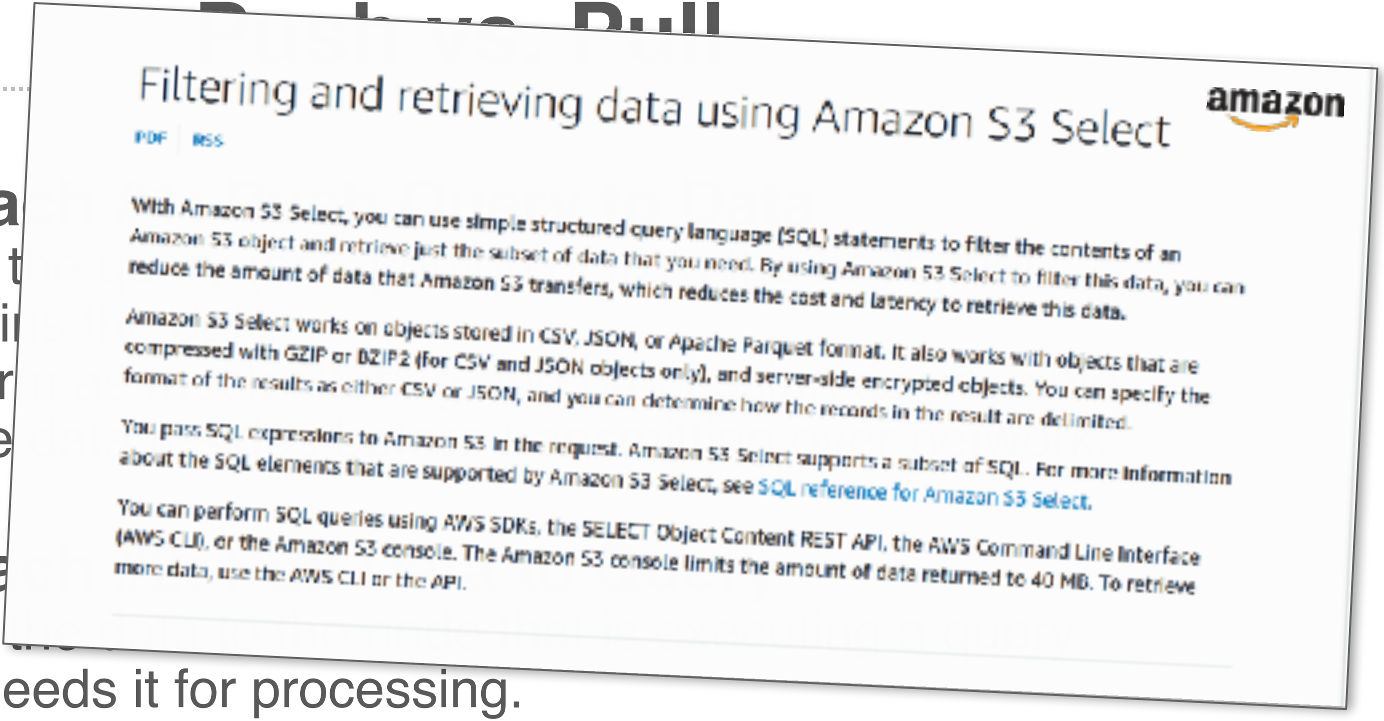
Approach #2: Pull Data to Query

→ 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 vs. Pull

Approa
→ Send t
contai
→ Perfor
where


Filtering and retrieving data using Amazon S3 Select

PDF | RSS

With Amazon S3 Select, you can use simple structured query language (SQL) statements to filter the contents of an Amazon S3 object and retrieve just the subset of data that you need. By using Amazon S3 Select to filter this data, you can reduce the amount of data that Amazon S3 transfers, which reduces the cost and latency to retrieve this data.

Amazon S3 Select works on objects stored in CSV, JSON, or Apache Parquet format. It also works with objects that are compressed with GZIP or BZIP2 (for CSV and JSON objects only), and server-side encrypted objects. You can specify the format of the results as either CSV or JSON, and you can determine how the records in the result are delimited.

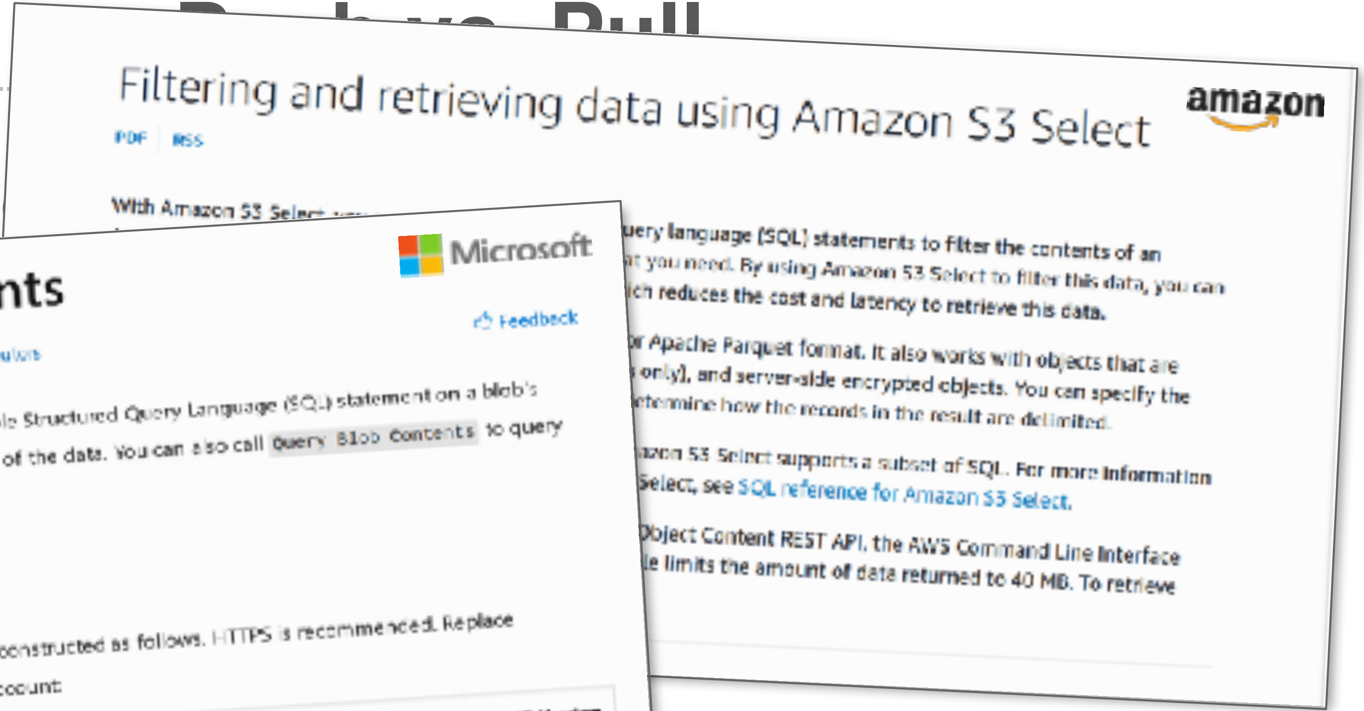
You pass SQL expressions to Amazon S3 in the request. Amazon S3 Select supports a subset of SQL. For more information about the SQL elements that are supported by Amazon S3 Select, see SQL reference for Amazon S3 Select.

You can perform SQL queries using AWS SDKs, the SELECT Object Content REST API, the AWS Command Line Interface (AWS CLI), or the Amazon S3 console. The Amazon S3 console limits the amount of data returned to 40 MB. To retrieve more data, use the AWS CLI or the API.

Approa
→ Bring
that needs it for processing.
→ This is necessary when there is no compute resources available where persistent data files are located.

Push vs. Pull

Approa


Filtering and retrieving data using Amazon S3 Select

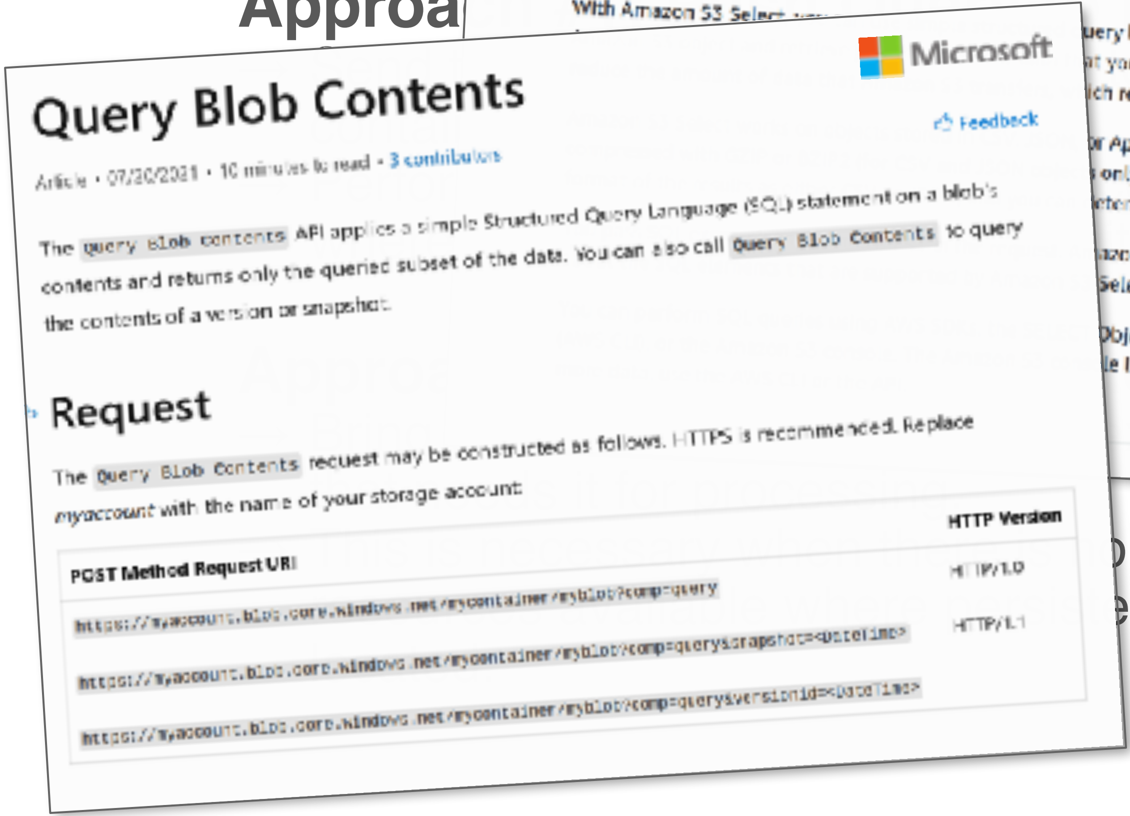
PDF | RSS

With Amazon S3 Select ... uery language (SQL) statements to filter the contents of an ... t you need. By using Amazon S3 Select to filter this data, you can ... ich reduces the cost and latency to retrieve this data.

... r Apache Parquet format. It also works with objects that are ... only), and server-side encrypted objects. You can specify the ... etermine how the records in the result are delimited.

... azon S3 Select supports a subset of SQL. For more information ... Select, see SQL reference for Amazon S3 Select.

... Object Content REST API, the AWS Command Line Interface ... le limits the amount of data returned to 40 MB. To retrieve


Query Blob Contents

Article • 07/30/2021 • 10 minutes to read • 3 contributors

Feedback

The `Query Blob Contents` API applies a simple Structured Query Language (SQL) statement on a blob's contents and returns only the queried subset of the data. You can also call `Query Blob Contents` to query the contents of a version or snapshot.

Request

The `Query Blob Contents` request may be constructed as follows. HTTPS is recommended. Replace *myaccount* with the name of your storage account:

POST Method Request URI	HTTP Version
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query`	HTTP/1.0
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query&snapshot=<DateTime>`	HTTP/1.1
`https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=query&versionid=<DateTime>`	

... compute

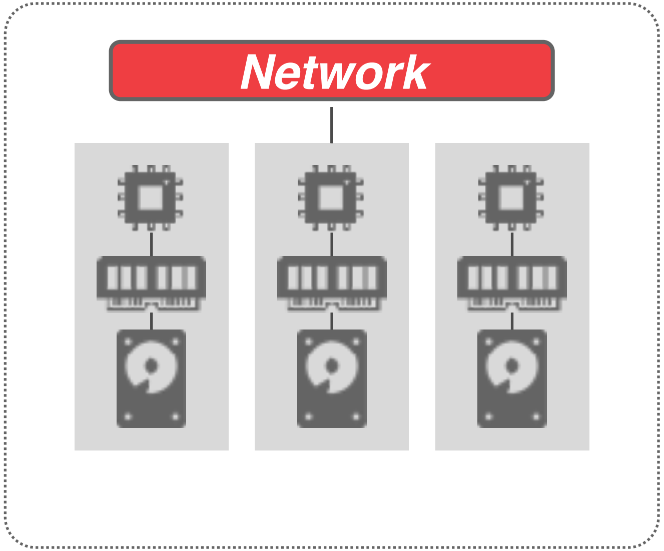
... ent data files are

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.

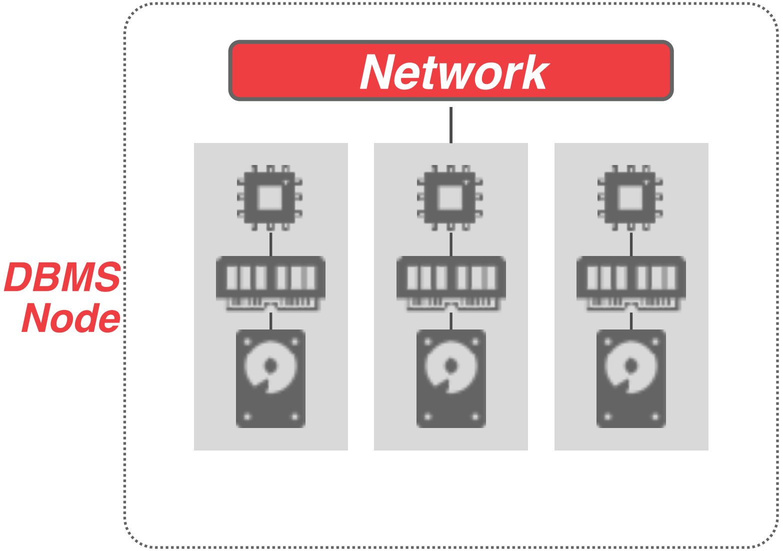


Shared-Nothing

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.

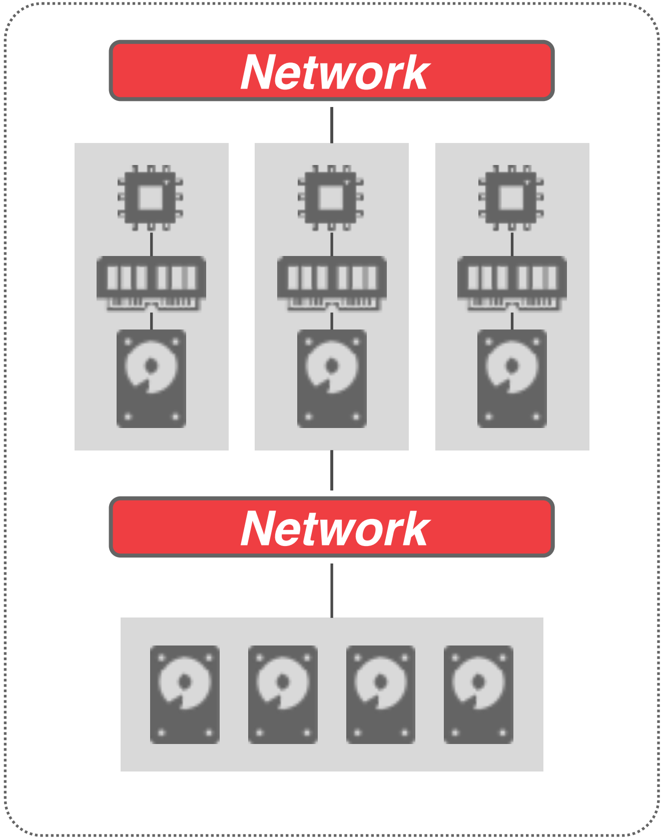
Since data is local, the DBMS can access it via POSIX API.



Shared-Disk

Each node accesses a single logical disk via an interconnect, but also has its own private memory and ephemeral storage.
→ Must send messages between nodes to learn about their current state.

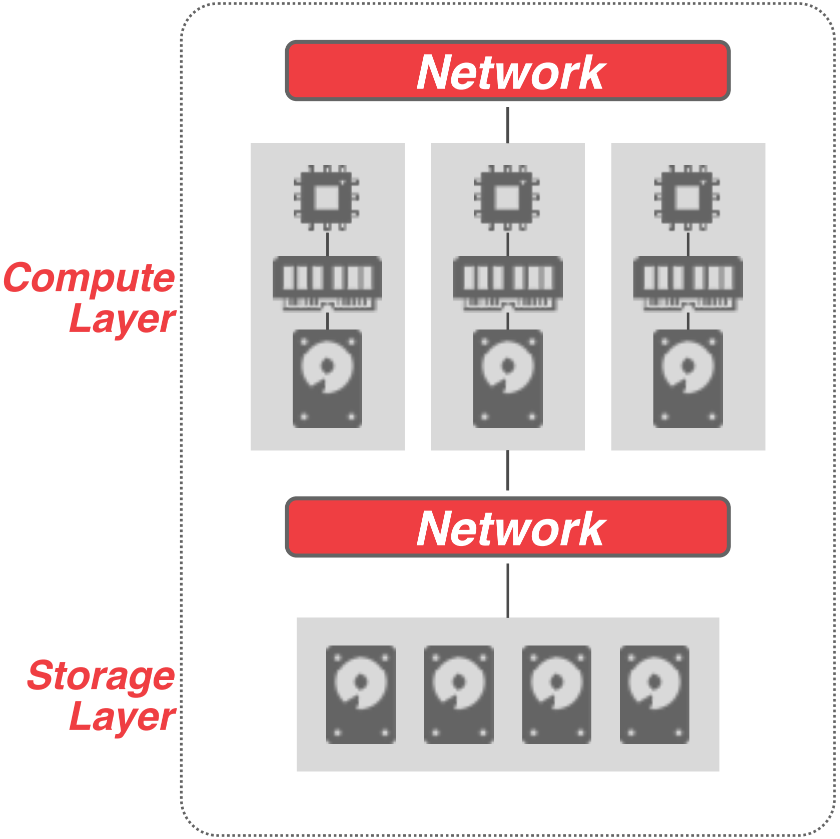
Instead of a POSIX API, the DBMS accesses disk via a userspace API.



Shared-Disk

Each node accesses a single logical disk via an interconnect, but also has its 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 via a userspace API.



System Architecture

Choice #1: Shared-Nothing:

→ Harder to scale capacity (data movement).
→ Potentially better performance & efficiency.
→ Apply filters where the data resides before transferring.

Choice #2: Shared-Disk:

→ Scale compute / storage layers independently.
→ Easy to shut down idle compute layer resources.
→ May need to pull uncached persistent data from storage layer to compute layer before filtering.

System Architecture

Choice #1: Shared-Nothing:

→ Harder to scale capacity (data movement).
→ Potentially better performance & efficiency.
→ Apply filters where the data resides before transferring.

Choice #2: Shared-Disk:

→ Scale compute / storage layers independently.
→ Easy to shut down idle compute layer resources.
→ May need to pull uncached persistent data from storage layer to compute layer before filtering.

Shared-Disk

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).

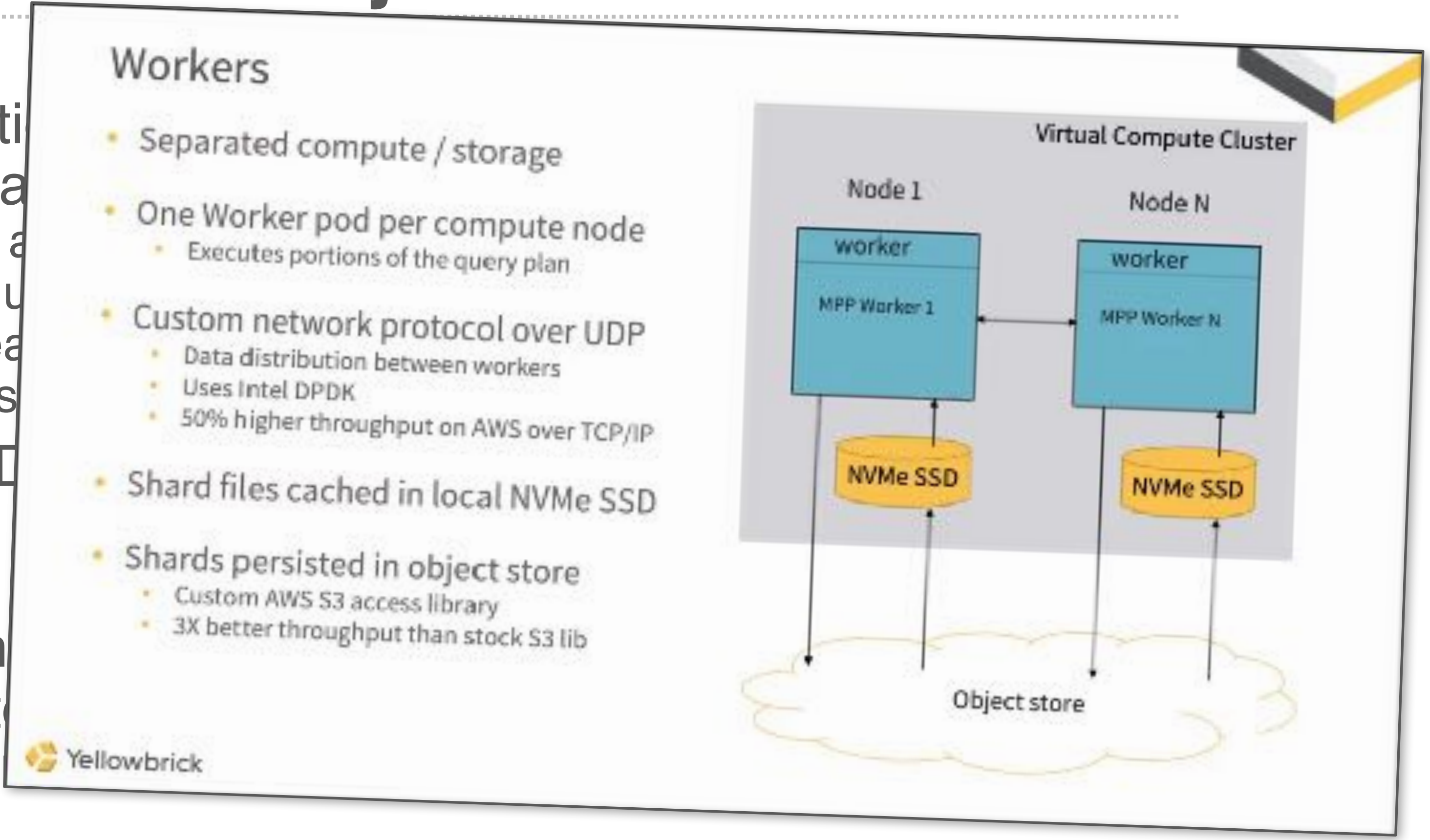
Each cloud vendor provides their own proprietary API to access data (**PUT**, **GET**, **DELETE**).
→ Some vendors support predicate pushdown (S3).

Object Stores

Partiti
into la
→ All a
colu
→ Hea
offs

The D
what

Each
API t
→ So



Observation

Snowflake is a monolithic system comprised of components built entirely in-house.

Most of the non-academic DBMSs we will cover this semester will have a similar overall architecture.

But this means that multiple organizations are writing the same DBMS software…

OLAP Commoditization

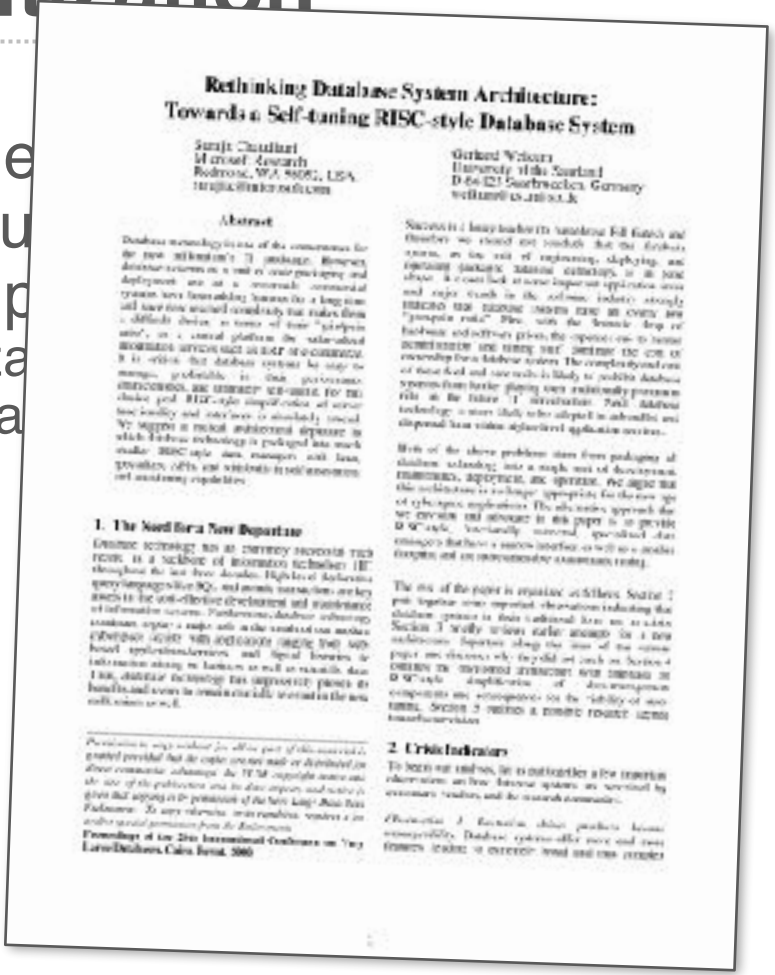
One recent trend in the last decade is the refactoring of OLAP engine sub-systems into standalone open-source components.
→ This is typically done by organizations not in the business of selling DBMS software.

Examples:
→ System Catalogs
→ Query Optimizers
→ File Format / Access Libraries
→ Execution Engines

OLAP Commoditization

One recent trend in the last de
refactoring of OLAP engine su
standalone open-source comp
→ This is typically done by organiza
business of selling DBMS softwa

Examples:
→ System Catalogs
→ Query Optimizers
→ File Format / Access Libraries
→ Execution Engines


Rethinking Database System Architecture:
Towards a Self-tuning RISC-style Database System

Surajit Chaudhuri
Microsoft Research
Redmond, WA 98052, USA

Gerhard Weikum
University of the Saarland
D-66123 Saarbruecken, Germany

Abstract

1. The Need for a New Departure

2. Crisis Indicators

Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

System Catalogs

A DBMS tracks a database's schema (e.g., table, columns) and data files in its catalog.

→ If the DBMS is on the data ingestion path, then it can maintain the catalog incrementally.

→ If an external process adds data files, then it also needs to update the catalog so that the DBMS is aware of them.

Notable implementations:

→ HCatalog

→ Google Data Catalog

→ Amazon Glue Data Catalog

Query Optimizers

Extendible search engine framework for heuristic- and cost-based query optimization.
→ DBMS provides transformation rules and cost estimates.
→ Framework returns either a logical or physical query plan.

This is the hardest part to build in any DBMS.

Notable implementations:
→ Greenplum Orca
→ Apache Calcite

ORCA: A MODULAR QUERY OPTIMIZER ARCHITECTURE FOR BIG DATA
SIGMOD 2014

APACHE CALCITE: A FOUNDATIONAL FRAMEWORK FOR OPTIMIZED QUERY PROCESSING OVER HETEROGENEOUS DATA SOURCES
SIGMOD 2018

File Formats

Most DBMSs use a proprietary on-disk binary file format for their databases. The only way to share data between systems is to convert data into a common text-based format
→ Examples: CSV, JSON, XML

There are open-source binary file formats that make it easier to access data across systems and libraries for extracting data from files.
→ Libraries provide an iterator interface to retrieve (batched) columns from files.

Storage Formats

Apache Parquet (2013)
→ Compressed columnar storage from Cloudera/Twitter

Apache ORC (2013)
→ Compressed columnar storage from Apache Hive.

Apache CarbonData (2013)
→ Compressed columnar storage with indexes from Huawei.

Apache Iceberg (2017)
→ Flexible data format that supports schema evolution from Netflix.

HDF5 (1998)
→ Multi-dimensional arrays for scientific workloads.

Apache Arrow (2016)
→ In-memory compressed columnar storage from Pandas / Dremio.

Execution Engines

Standalone libraries for executing vectorized query operators on columnar data.

→ Input is a DAG of physical operators.

→ Require external scheduling and orchestration.

Notable implementations:

→ Velox

→ DataFusion

→ Intel OAP

Conclusion

Today was about understanding the high-level history and concept of modern OLAP DBMSs.

→ Fundamentally, they are not very different from previous distributed / parallel DBMSs **except** for their use of cloud-based object stores.

Our focus for the rest of the quarter will be on state-of-the-art implementations of the various system components.

Next Class

Data Storage