

Columnar Data in 2024: The Future of Efficient Data Analytics

Jeremy Taylor
Head of Product
jdt@juxt.pro
[@refset](https://twitter.com/refset)





a glacier with ancient frozen columns of binary data visible within







COMPUTE





COMPUTE

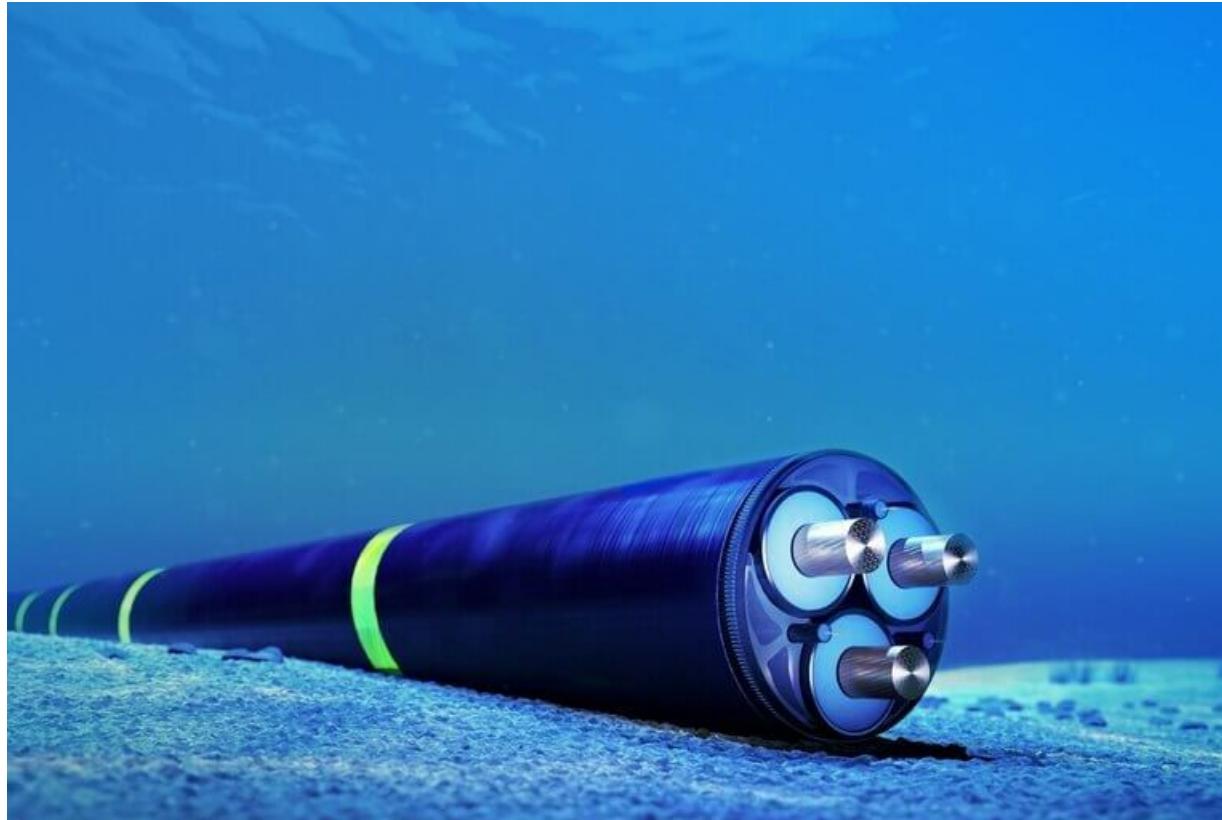


STORAGE





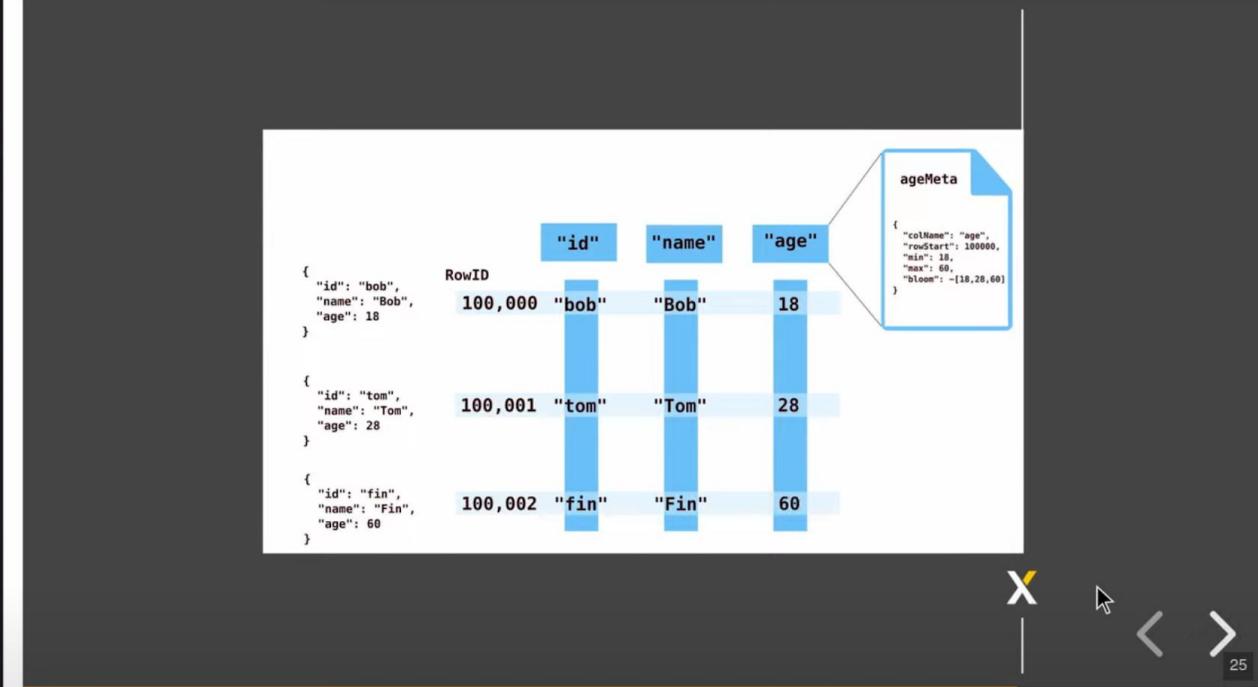
LATENCY



BANDWIDTH

About Me

Ex-IBM
Head of Product, xtdb.com
@refset



"State of XTDB" by Jon Pither



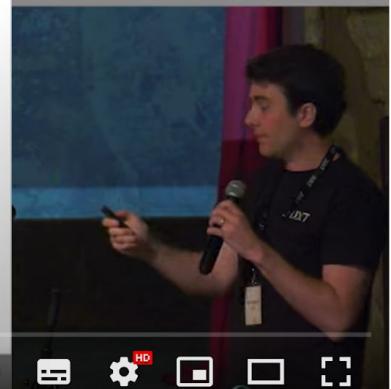
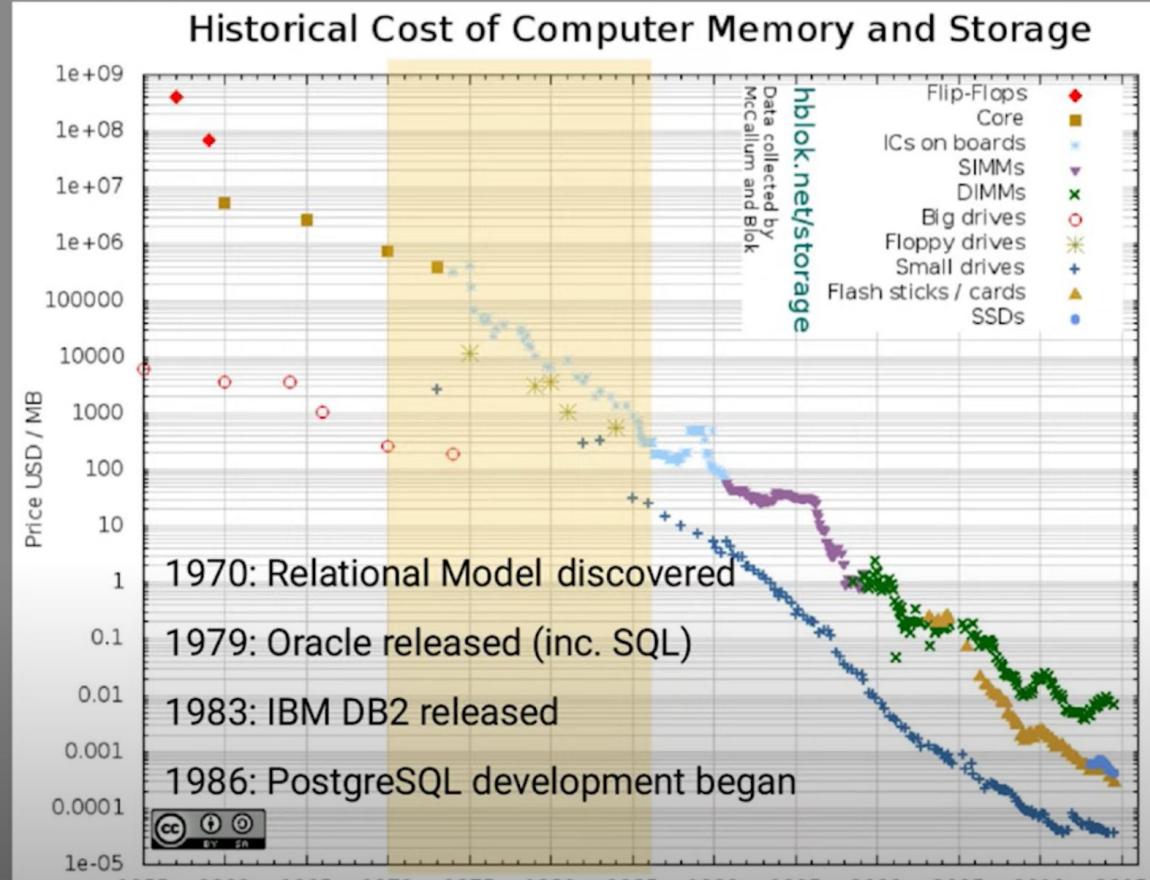
44

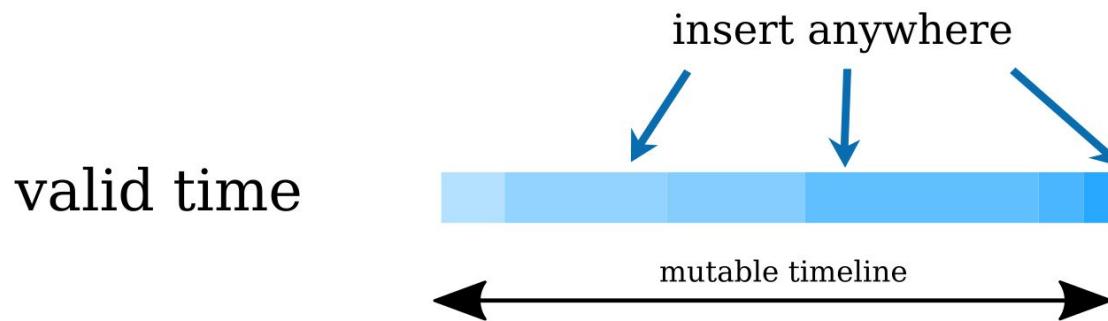
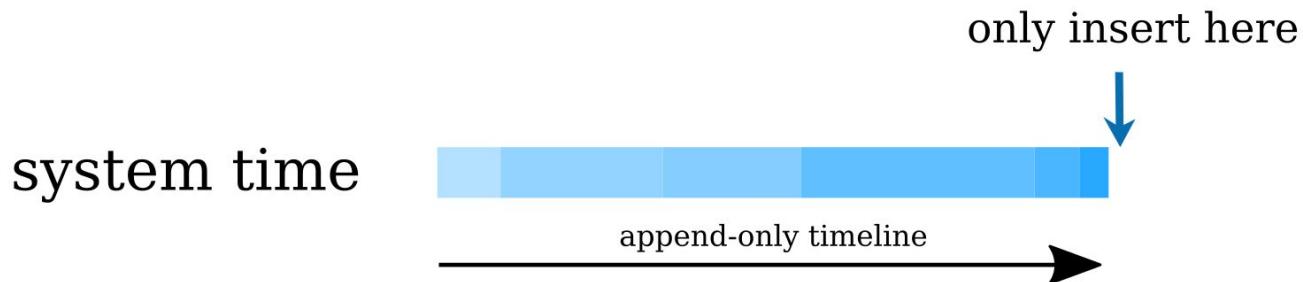


Share

Save







Today's Topics

1. Columnar data 101
2. Context & trends
3. Modern analytic systems
4. R&D

RELATIONAL MODEL

A relation is an unordered set that contain the relationship of attributes that represent entities.

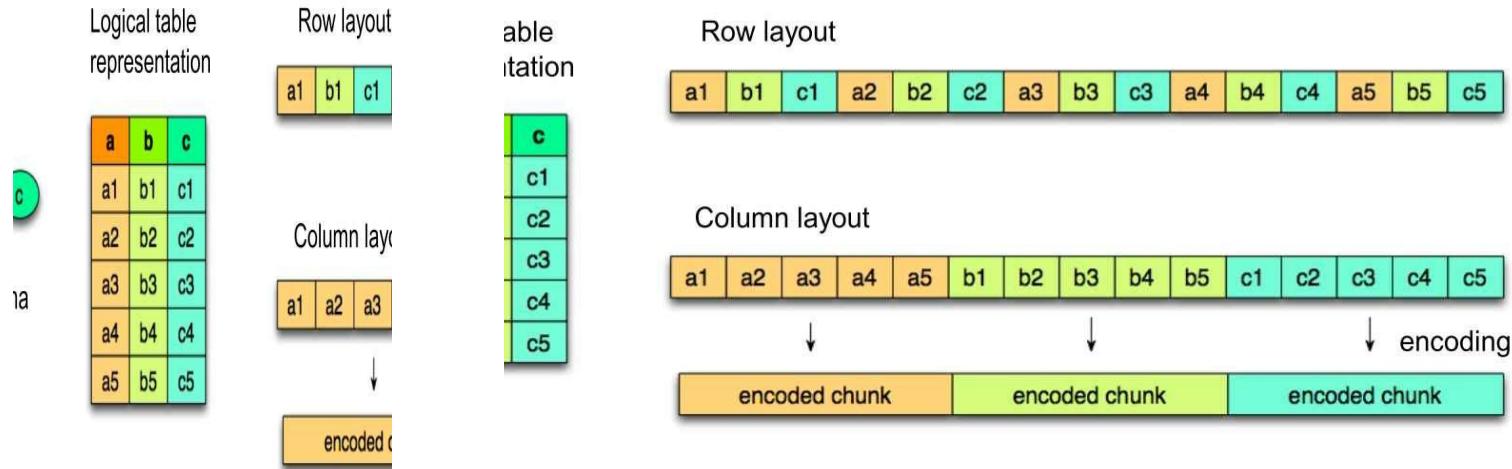
A tuple is a set of attribute values (also known as its domain) in the relation.
→ Values are (normally) atomic/scalar.
→ The special value **NULL** is a member of every domain (if allowed).

Artist(name, year, country)

| name | year | country |
|---------------|------|---------|
| Wu-Tang Clan | 1992 | USA |
| Notorious BIG | 1992 | USA |
| GZA | 1990 | USA |

n-ary Relation
=

Table with n columns



| | session_id | timestamp | source_ip |
|-------|------------|-----------------|----------------|
| Row 1 | 1331246660 | 3/8/2012 2:44PM | 99.155.155.225 |
| Row 2 | 1331246351 | 3/8/2012 2:38PM | 65.87.165.114 |
| Row 3 | 1331244570 | 3/8/2012 2:09PM | 71.10.106.181 |
| Row 4 | 1331261196 | 3/8/2012 6:46PM | 76.102.156.138 |

Row-oriented Memory Buffer

| | |
|--------|-----------------|
| Row 1 | 1331246660 |
| Row 2 | 3/8/2012 2:44PM |
| Row 3 | 99.155.155.225 |
| Row 4 | 1331246351 |
| Row 5 | 3/8/2012 2:38PM |
| Row 6 | 65.87.165.114 |
| Row 7 | 1331244570 |
| Row 8 | 3/8/2012 2:09PM |
| Row 9 | 71.10.106.181 |
| Row 10 | 1331261196 |
| Row 11 | 3/8/2012 6:46PM |
| Row 12 | 76.102.156.138 |

Column-oriented Memory Buffer

| | |
|------------|-----------------|
| session_id | 1331246660 |
| timestamp | 1331246351 |
| source_ip | 1331244570 |
| session_id | 1331261196 |
| timestamp | 3/8/2012 2:44PM |
| source_ip | 3/8/2012 2:38PM |
| session_id | 3/8/2012 2:09PM |
| timestamp | 3/8/2012 6:46PM |
| source_ip | 99.155.155.225 |
| session_id | 65.87.165.114 |
| timestamp | 71.10.106.181 |
| source_ip | 76.102.156.138 |

The idea behind *column-oriented storage* is simple: don't store all the values from one row together, but store all the values from each *column* together instead.

If each column is stored in a separate file, a query only needs to read and parse those columns that are used in that query, which can save a lot of work.

Columnar data storage example

- Suppose we have a 1TB table with 100 columns.
- We have a query that requires 5 columns of the table.
 - Row-store:
Read entire 1TB of data from disk at 100MB/s \approx 3 hours
 - Columnar storage:
Read 5 columns (50GB) from disk \approx 8 minutes

Compression

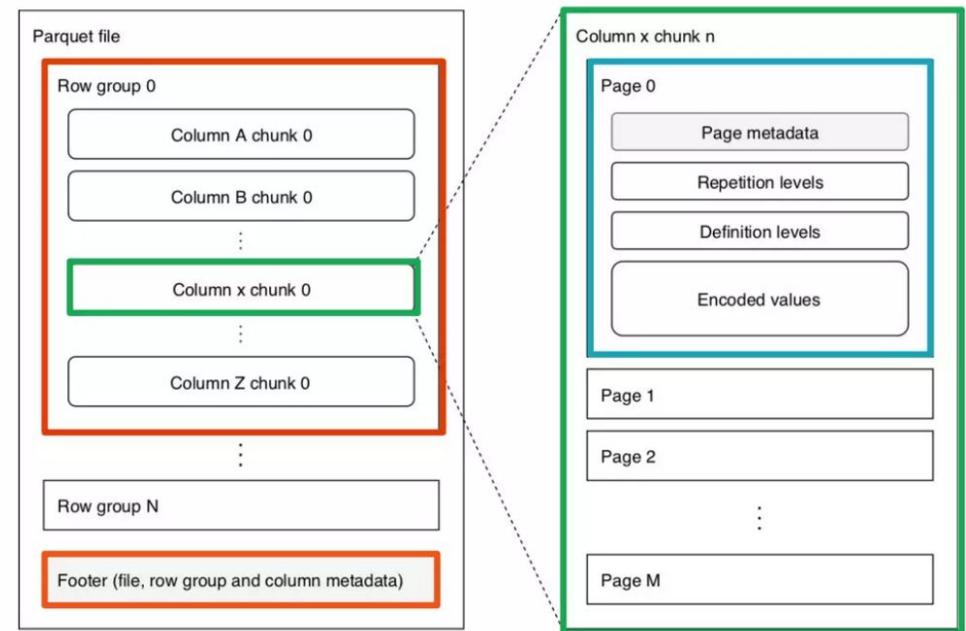
- We have a query that requires 5 columns of the table.
 - No compression:
Read 5 columns (50GB) from disk ≈ 8 minutes
 - Compression:
Read 5 compressed columns (5x -> 10GB) from disk ≈ 1:40 minutes



- 1) Columnar layout
- 2) On-disk compression (e.g. Run-Length Encoding)
- 3) ~Ideal cold storage

Parquet: data organization

- Data organization
 - Row-groups (*default 128MB*)
 - Column chunks
 - Pages (*default 1MB*)
 - Metadata
 - Min
 - Max
 - Count
 - Rep/def levels
 - Encoded values



Optimization: predicate pushdown

```
SELECT * FROM table WHERE x > 5
```

Row-group 0: x: [min: 0, max: 9]

Row-group 1: x: [min: 3, max: 7]

Row-group 2: x: [min: 1, max: 4]

...

- Leverage min/max statistics

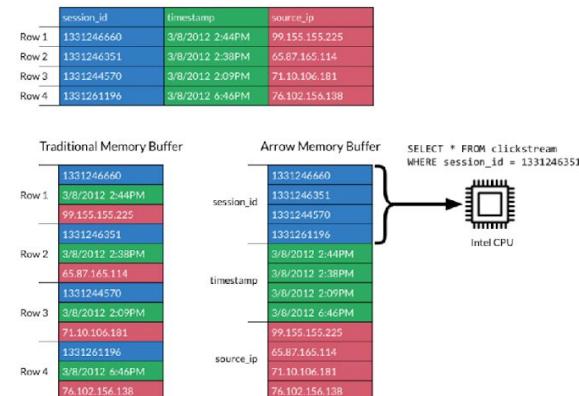
Apache Arrow Overview

Apache Arrow is a software development platform for building high performance applications that process and transport large data sets. It is designed to both improve the performance of analytical algorithms and the efficiency of moving data from one system or programming language to another.

A critical component of Apache Arrow is its **in-memory columnar format**, a standardized, language-agnostic specification for representing structured, table-like datasets in-memory. This data format has a rich data type system (included nested and user-defined data types) designed to support the needs of analytic database systems, data frame libraries, and more.

Columnar is Fast

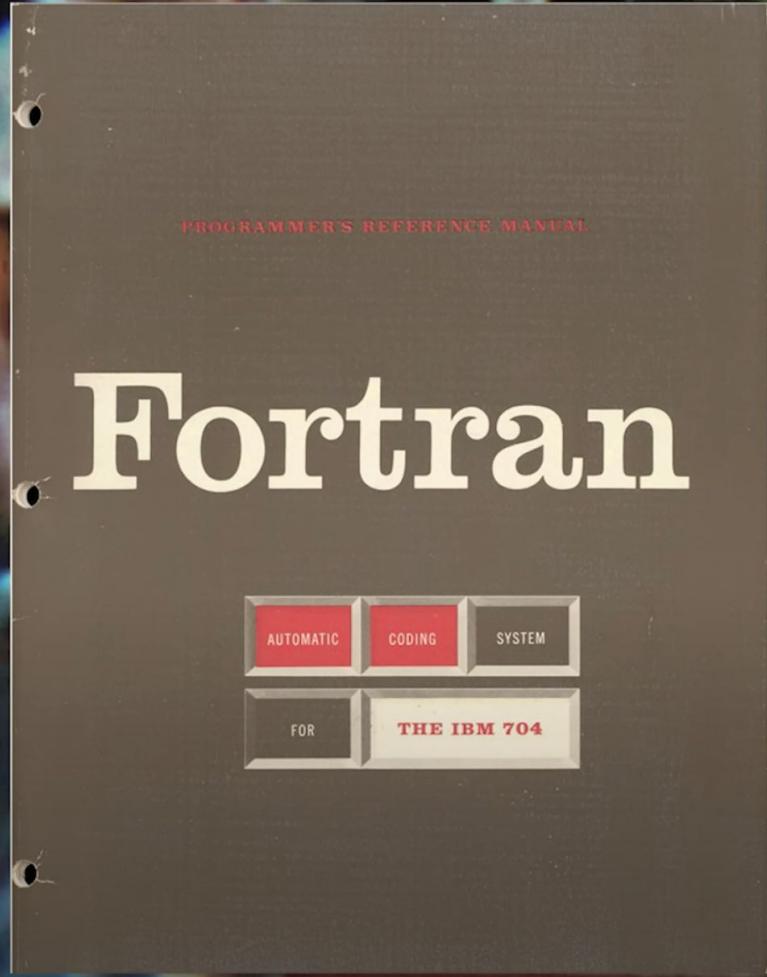
The Apache Arrow format allows computational routines and execution engines to maximize their efficiency when scanning and iterating large chunks of data. In particular, the contiguous columnar layout enables vectorization using the latest SIMD (Single Instruction, Multiple Data) operations included in modern processors.



Context & Trends

REAL A,B,C,D,X,Y

三



FORMULA TRANSLATOR

FORTRAN



The Beginnings of FORTRAN (Complete)

“our primary objective was to permit people to concentrate on the essence of their problems and eliminate preoccupation with the mechanics of the computer”

- Irv Ziller, The Beginnings of Fortran

Passing arrays between languages

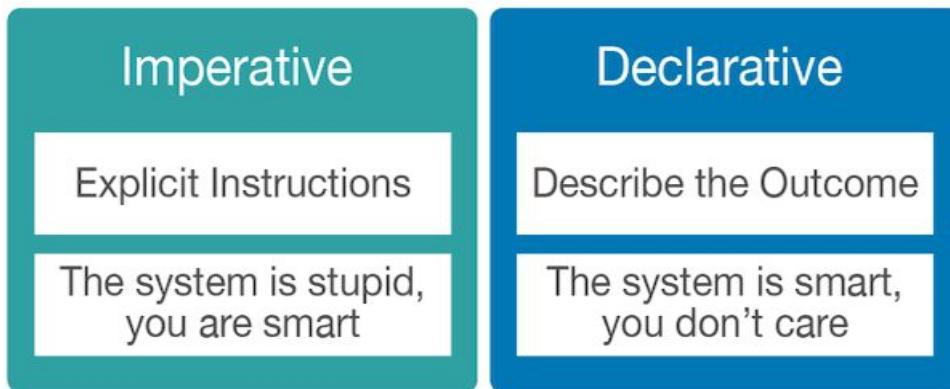
Fortran stores array elements in ascending storage units in column-major order. C stores array elements in row-major order. Fortran array indexes start at 1, while C array indexes start at 0.

HOT long vectorization

When you specify any of the following:

- **-O4** and higher
- **-qhot** with **-qnostict**

you enable **-qhot=vector** by default. Specifying **-qnostict** with optimizations other than **-O4** and **-O5** ensures that the compiler looks for long vectorization opportunities. This can optimize loops in source code for operations on array data by ensuring that operations run in parallel where applicable. The compiler uses standard machine registers for these transformations and does not restrict vector data size; supporting both single- and double-precision floating-point vectorization. Often, HOT vectorization involves transformations of loop calculations into calls to specialized mathematical routines supplied with the compiler such as the Mathematical Acceleration Subsystem (MASS) libraries. These mathematical routines use algorithms that calculate results more efficiently than executing the original loop code.





Why do I talk about SQL?

SQL is the only ever successful, mainstream, and general-purpose 4GL ([Fourth-Generation Programming Language](#))

And it is awesome!

Copyright (c) 2009-2017 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0



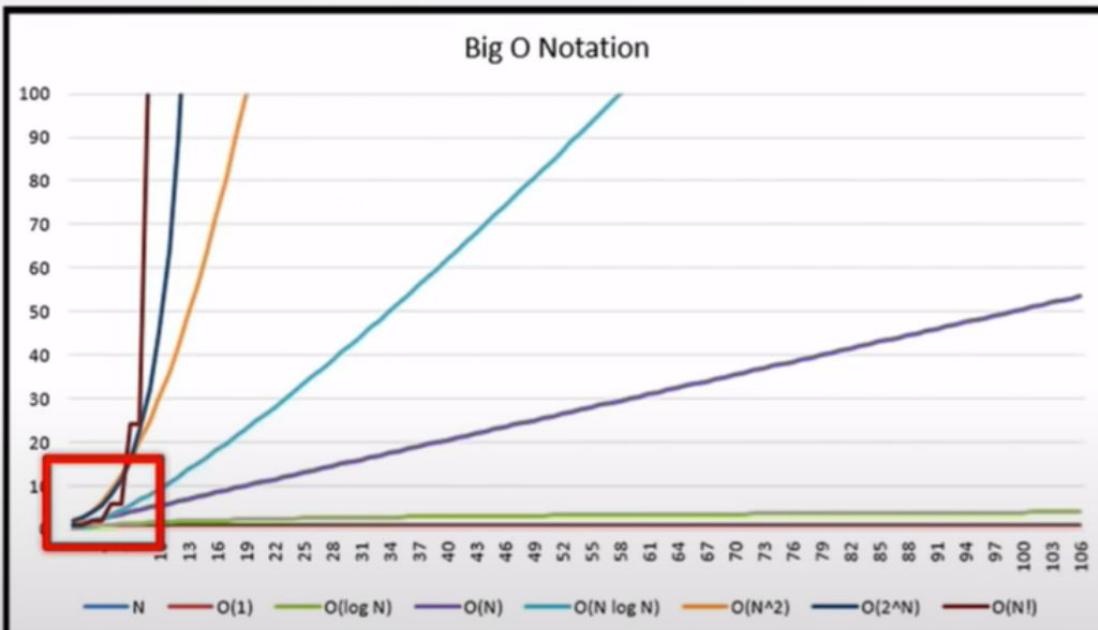
How Modern SQL Databases Come up with Algorithms that You Would Have Never Dreamed Of by Lukas Eder

What's the actual algorithm?

How many rows from operation

| OPERATION | OBJECT_NAME | CARDINALITY | COST |
|------------------------------------|-------------------------|-------------|------|
| SELECT STATEMENT | | 13 | 9 |
| SORT (ORDER BY) | | 13 | 9 |
| HASH (GROUP BY) | | 13 | 9 |
| HASH JOIN | | 154 | 7 |
| Access Predicates | | | |
| ACTOR.ACTOR_ID=FILM_ACTOR.ACTOR_ID | | | |
| TABLE ACCESS (BY INDEX ROWID) | ACTOR | 6 | 2 |
| INDEX (RANGE SCAN) | IDX_ACTOR_LAST_NAME | 6 | 1 |
| Access Predicates | | | |
| ACTOR.LAST_NAME LIKE 'A%' | | | |
| Filter Predicates | | | |
| ACTOR.LAST_NAME LIKE 'A%' | | | |
| INDEX (FAST FULL SCAN) | IDX_FK_FILM_ACTOR_ACTOR | 5462 | 4 |

Your estimate / dev environment is here



Latency Numbers Every Programmer Should Know

■ 1ns

■ L1 cache reference: 0.5ns

■ Branch mispredict: 5ns

■ L2 cache reference: 7ns

■ Mutex lock/unlock: 25 ns

= ■ 100 ns

■ Main memory reference: 100ns

= ■ 1 μs

■ Compress 1KB with Zippy: 3 μs

= ■ 10 μs

■ Send 1KB over 1Gbps network: 10 μs

■ SSD random read (1Gb/s SSD): 150 μs

■ Read 1MB sequentially from memory: 250 μs

■ Round trip in same datacenter: 500 μs

= ■ 1 ms

■ Read 1MB sequentially from SSD: 1 ms

■ Disk seek: 10 ms

■ Read 1MB sequentially from disk: 20 ms

■ Packet roundtrip CA to Netherlands: 150 ms

Source: <https://gist.github.com/2841832>

Lets multiply all these durations by a billion:

Magnitudes:

Minute:

| | | |
|--------------------|-------|------------------------|
| L1 cache reference | 0.5 s | One heart beat (0.5 s) |
| Branch mispredict | 5 s | Yawn |
| L2 cache reference | 7 s | Long yawn |
| Mutex lock/unlock | 25 s | Making a coffee |

Hour:

| | | |
|------------------------------|--------|--|
| Main memory reference | 100 s | Brushing your teeth |
| Compress 1K bytes with Zippy | 50 min | One episode of a TV show (including ad breaks) |

Day:

| | | |
|-----------------------------------|--------|-------------------------------|
| Send 2K bytes over 1 Gbps network | 5.5 hr | From lunch to end of work day |
|-----------------------------------|--------|-------------------------------|

Week

| | | |
|------------------------------------|-----------|---|
| SSD random read | 1.7 days | A normal weekend |
| Read 1 MB sequentially from memory | 2.9 days | A long weekend |
| Round trip within same datacenter | 5.8 days | A medium vacation |
| Read 1 MB sequentially from SSD | 11.6 days | Waiting for almost 2 weeks for a delivery |

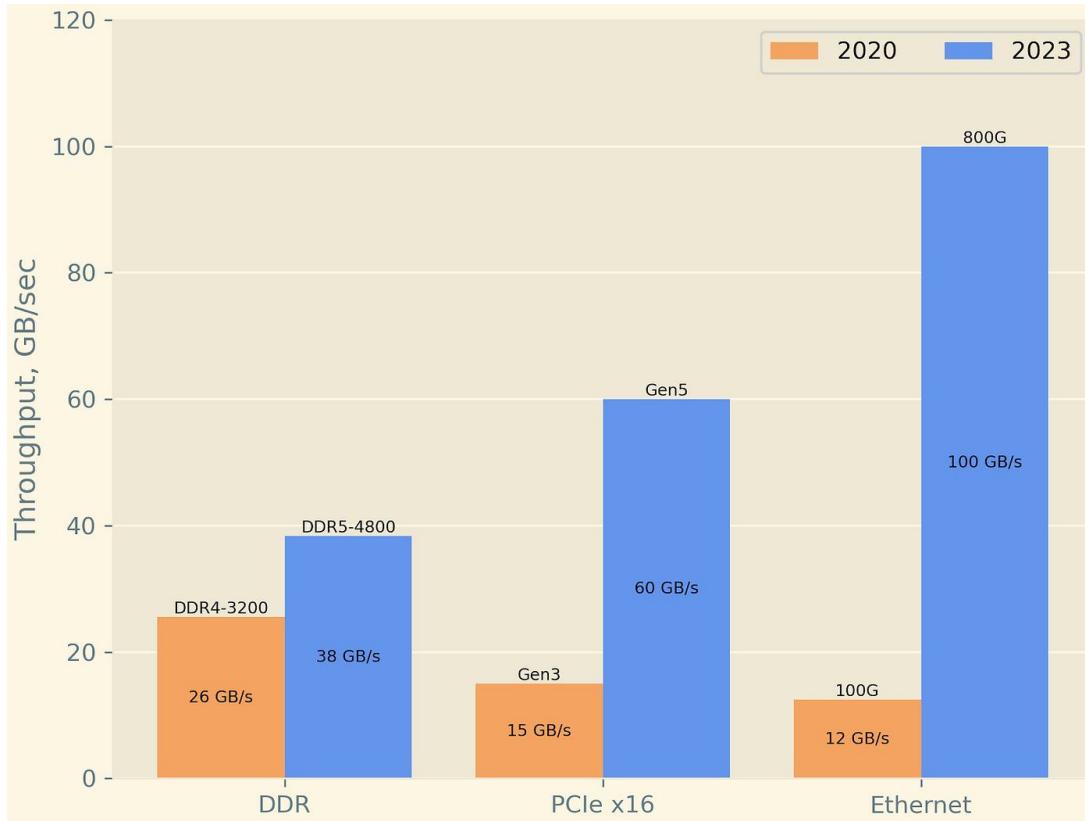
Year

| | | |
|----------------------------------|------------|------------------------------------|
| Disk seek | 16.5 weeks | A semester in university |
| Read 1 MB sequentially from disk | 7.8 months | Almost producing a new human being |
| The above 2 together | 1 year | |

Decade

| | | |
|---------------------------------|-----------|---|
| Send packet CA->Netherlands->CA | 4.8 years | Average time it takes to complete a bachelor's degree |
|---------------------------------|-----------|---|

The ebb and flow of relative bandwidth.



The Three Dimensions of Big Data Management

- **Data Gravity** – costs of moving
- **Data Residency** – legal implications
- **Data Latency** – the hardest technical challenge

Mechanical Sympathy

- **Mechanical sympathy is when you use a tool or system with an understanding of how it operates best**
 - You don't need to be a hardware engineer
 - You do need to understand how the hardware works and take that into consideration when you design software

Mechanical Sympathy



View Presentation

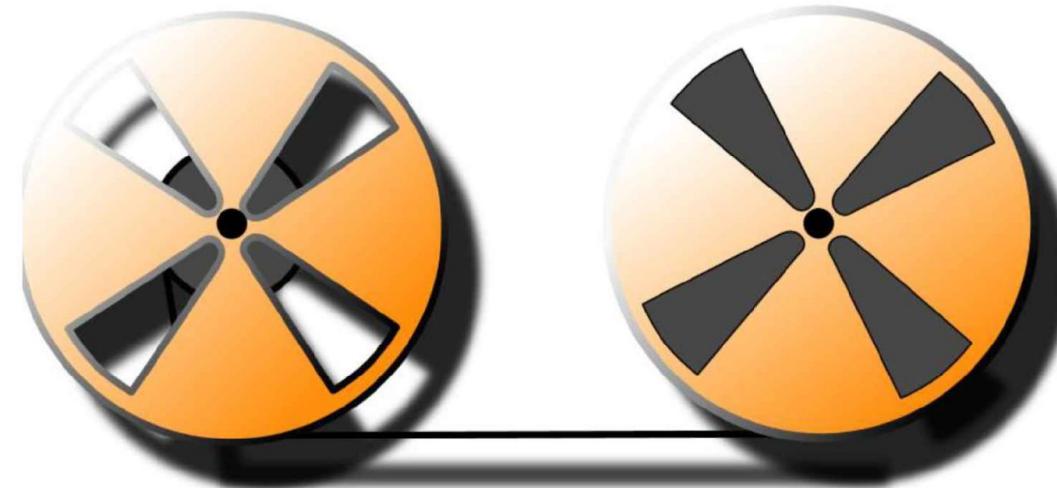
Speed: 1X 1.25X 1.5X 2X

A video player interface showing a presentation. On the left, a video frame shows a man in a white shirt speaking. To his right is a large red logo. Below the video frame are controls: a play button, a progress bar from 36:51 to 49:34, volume, and a three-dot menu. At the bottom are download options: Download, MP3, and SLIDES. The total duration is 49:34.

Summary

Martin Thompson ponders if there is a mechanical sympathy between developers and computers, and how to balance elegant design with the application of science in the development of modern software.

All Storage is Tape



Mechanical Sympathy



View Presentation

Speed: 1X 1.25X 1.5X 2X



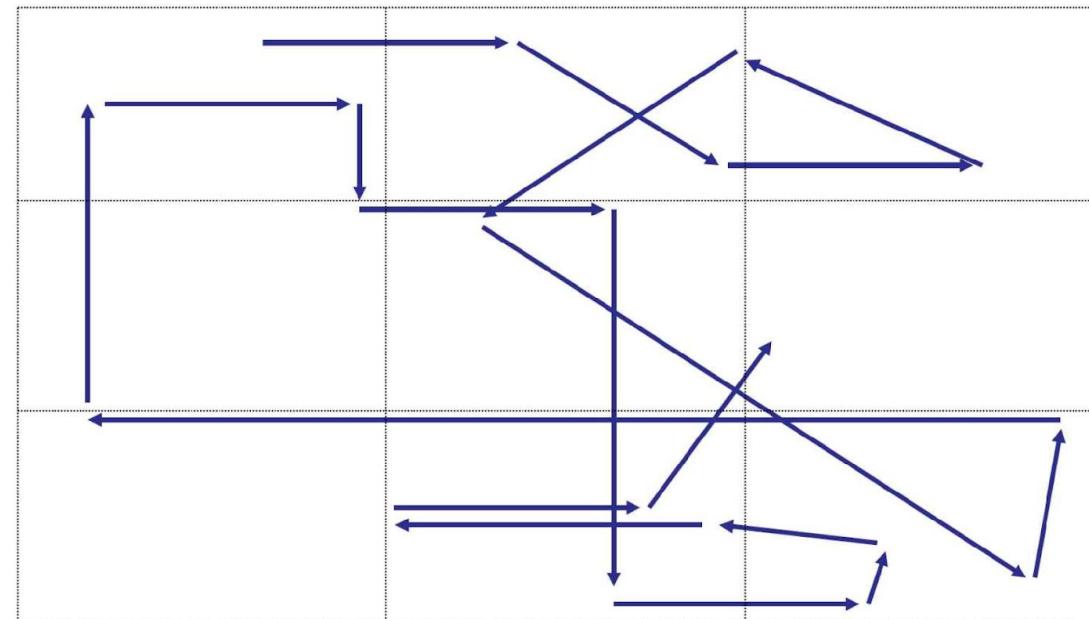
Download MP3 SLIDES

49:34

Summary

Martin Thompson ponders if there is a mechanical sympathy between developers and computers, and how to balance elegant design with the application of science in the development of modern software.

Memory Access Patterns Matter



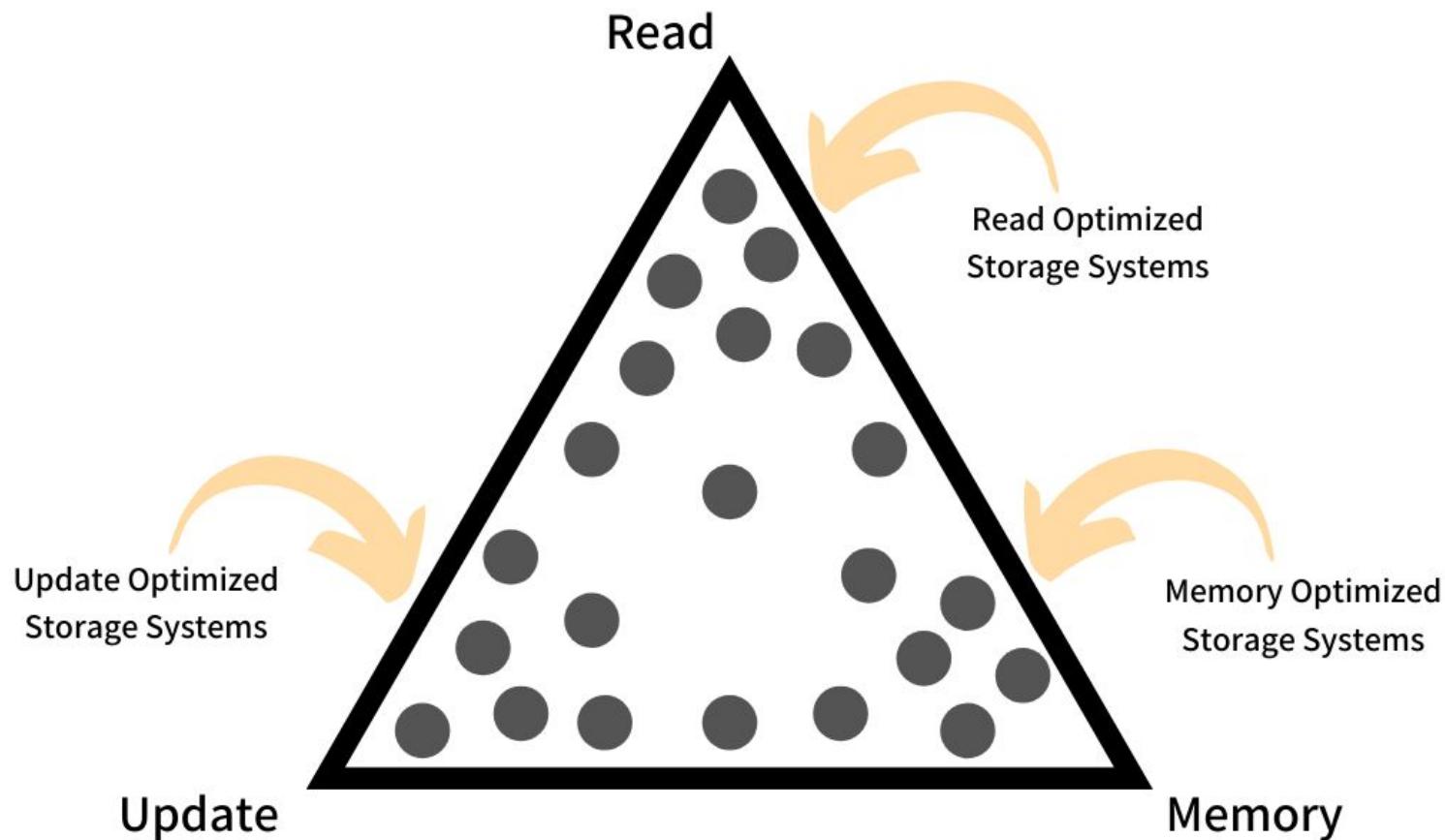
Mechanical Sympathy in Software Design

- **The key to high performance: minimize instructions, minimize data**
 - Do the most work in the fewest instructions
 - Reduce the data being shunted around
- You achieve this by modeling the problem domain and eliminating non-essential complexity.

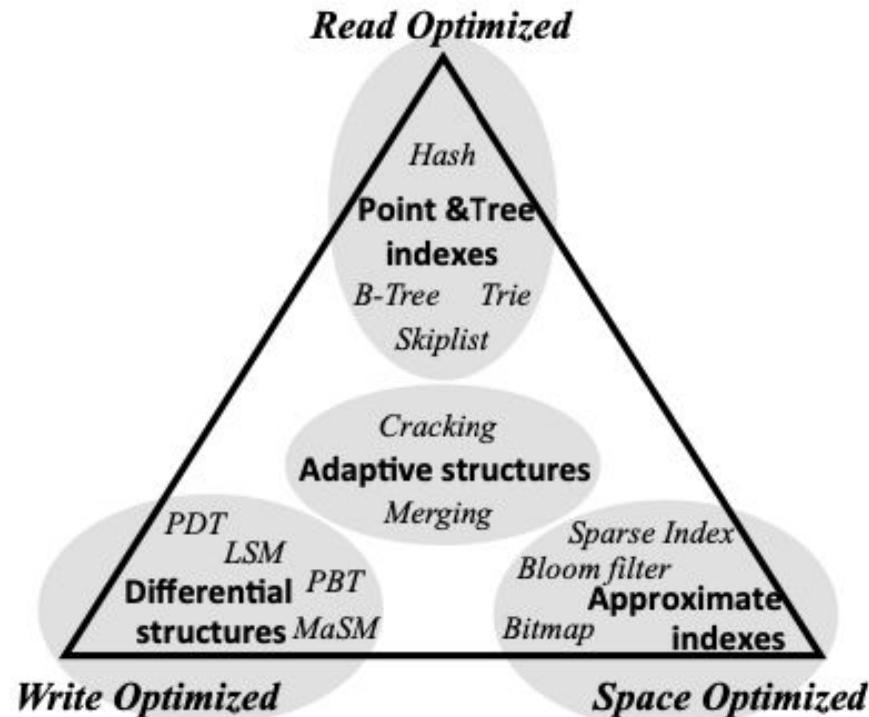
Mechanical Sympathy in Main Memory

- Bandwidth has exploded while latencies are the ~same
- To hide latency CPUs use evermore complex layers of caches
- CPUs hide memory latency using 3 heuristics:
 - **Temporal**: Recent data will likely be required again soon
 - **Spatial**: Adjacent data is likely to be required next
 - **Striding**: Memory access is likely to follow simple patterns

The RUM Conjecture

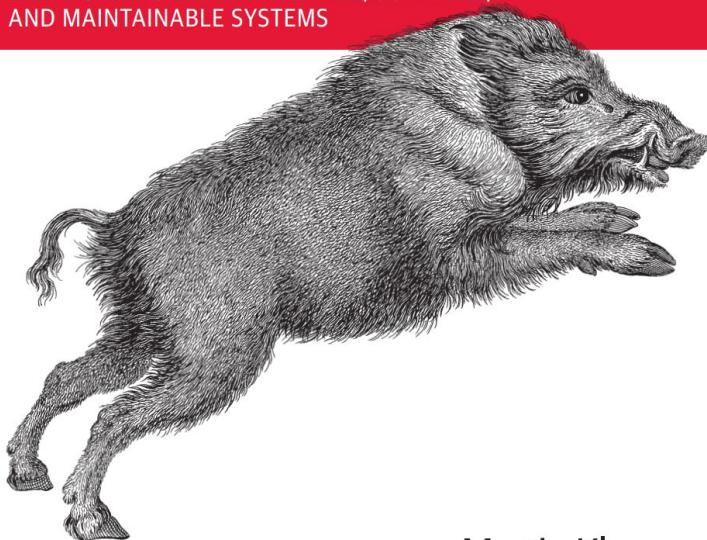


A single algorithm can only minimize two out of: Read / Update / Memory overheads



Designing Data-Intensive Applications

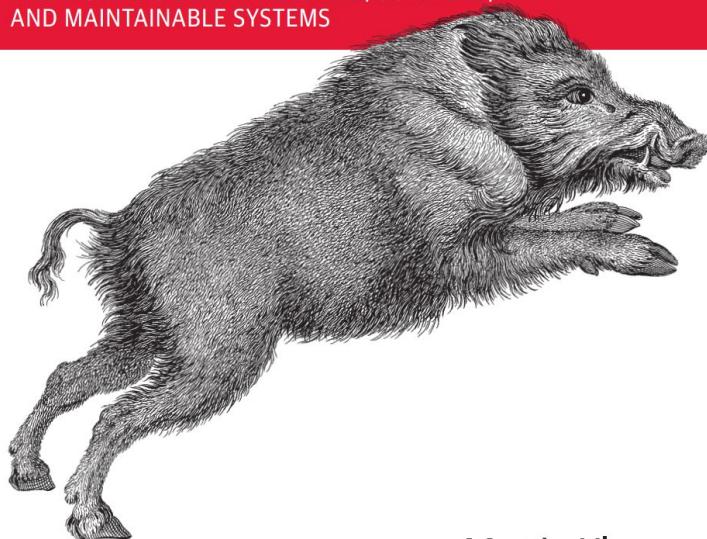
THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



Martin Kleppmann

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,
AND MAINTAINABLE SYSTEMS



Martin Kleppmann

...as opposed to “Compute-Intensive”

Key qualities:

- Reliability (fault-tolerance)
- Scalability (response to load)
- Maintainability (simplicity)

Table 3-1. Comparing characteristics of transaction processing versus analytic systems

| Property | Transaction processing systems (OLTP) | Analytic systems (OLAP) |
|----------------------|---|---|
| Main read pattern | Small number of records per query, fetched by key | Aggregate over large number of records |
| Main write pattern | Random-access, low-latency writes from user input | Bulk import (ETL) or event stream |
| Primarily used by | End user/customer, via web application | Internal analyst, for decision support |
| What data represents | Latest state of data (current point in time) | History of events that happened over time |
| Dataset size | Gigabytes to terabytes | Terabytes to petabytes |

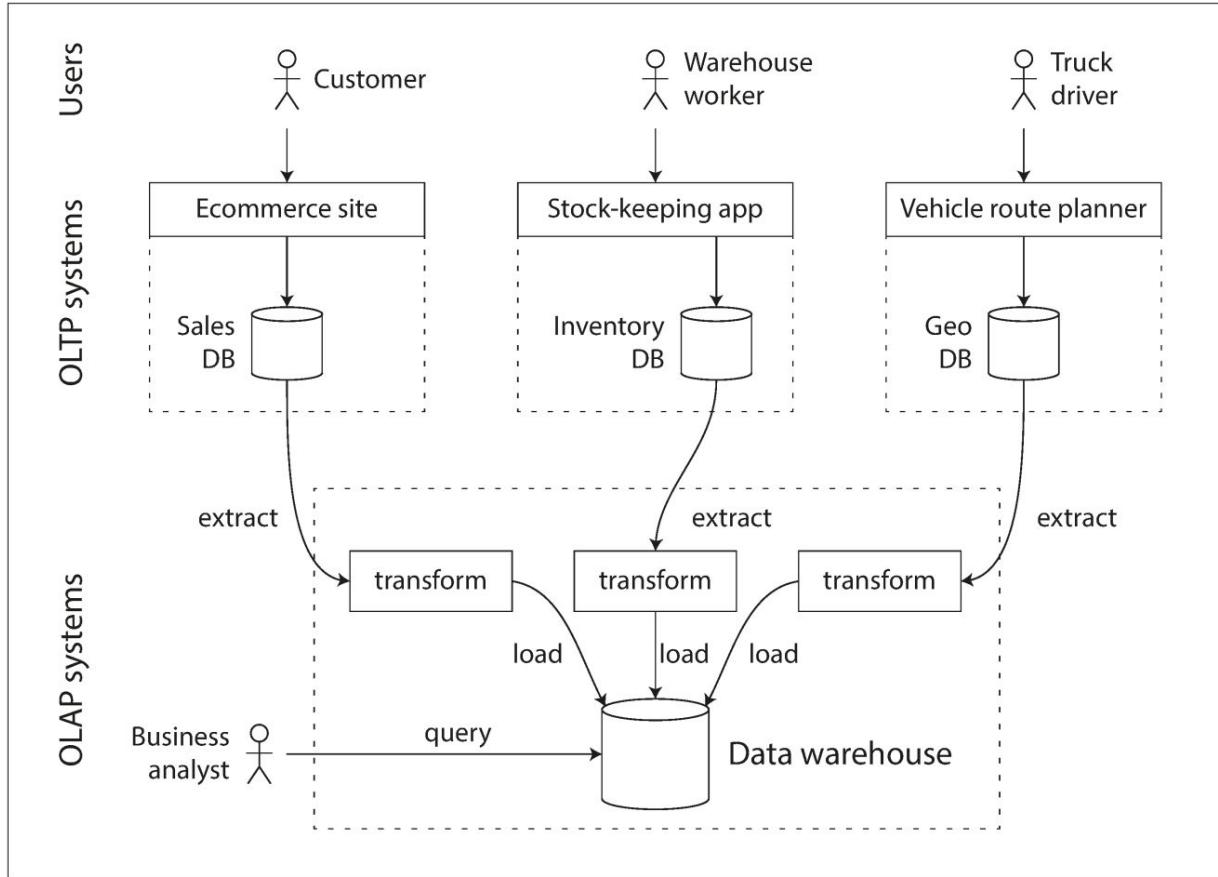


Figure 3-8. Simplified outline of ETL into a data warehouse.

"In a typical data warehouse, tables are often very wide: **fact tables** often have **over 100 columns**, sometimes several hundred"

fact_sales table

| date_key | product_sk | store_sk | promotion_sk | customer_sk | quantity | net_price | discount_price |
|----------|------------|----------|--------------|-------------|----------|-----------|----------------|
| 140102 | 69 | 4 | NULL | NULL | 1 | 13.99 | 13.99 |
| 140102 | 69 | 5 | 19 | NULL | 3 | 14.99 | 9.99 |
| 140102 | 69 | 5 | NULL | 191 | 1 | 14.99 | 14.99 |
| 140102 | 74 | 3 | 23 | 202 | 5 | 0.99 | 0.89 |
| 140103 | 31 | 2 | NULL | NULL | 1 | 2.49 | 2.49 |
| 140103 | 31 | 3 | NULL | NULL | 3 | 14.99 | 9.99 |
| 140103 | 31 | 3 | 21 | 123 | 1 | 49.99 | 39.99 |
| 140103 | 31 | 8 | NULL | 233 | 1 | 0.99 | 0.99 |

Columnar storage layout:

date_key file contents: 140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103

product_sk file contents: 69, 69, 69, 74, 31, 31, 31, 31

store_sk file contents: 4, 5, 5, 3, 2, 3, 3, 8

promotion_sk file contents: NULL, 19, NULL, 23, NULL, NULL, 21, NULL

customer_sk file contents: NULL, NULL, 191, 202, NULL, NULL, 123, 233

quantity file contents: 1, 3, 1, 5, 1, 3, 1, 1

net_price file contents: 13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99

discount_price file contents: 13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Figure 3-10. Storing relational data by column, rather than by row.

"We can only reconstruct a row because we know that the k th item in one column belongs to the same row as the k th item in another column."

Column values:

product_sk: 69 69 69 69 74 31 31 31 31 29 30 30 31 31 31 68 69 69

Bitmap for each possible value:

product_sk = 29: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

product_sk = 30: 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0

product_sk = 31: 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 0 0 0 0

product_sk = 68: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

product_sk = 69: 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

product_sk = 74: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Run-length encoding:

product_sk = 29: 9, 1 (9 zeros, 1 one, rest zeros)

product_sk = 30: 10, 2 (10 zeros, 2 ones, rest zeros)

product_sk = 31: 5, 4, 3, 3 (5 zeros, 4 ones, 3 zeros, 3 ones, rest zeros)

product_sk = 68: 15, 1 (15 zeros, 1 one, rest zeros)

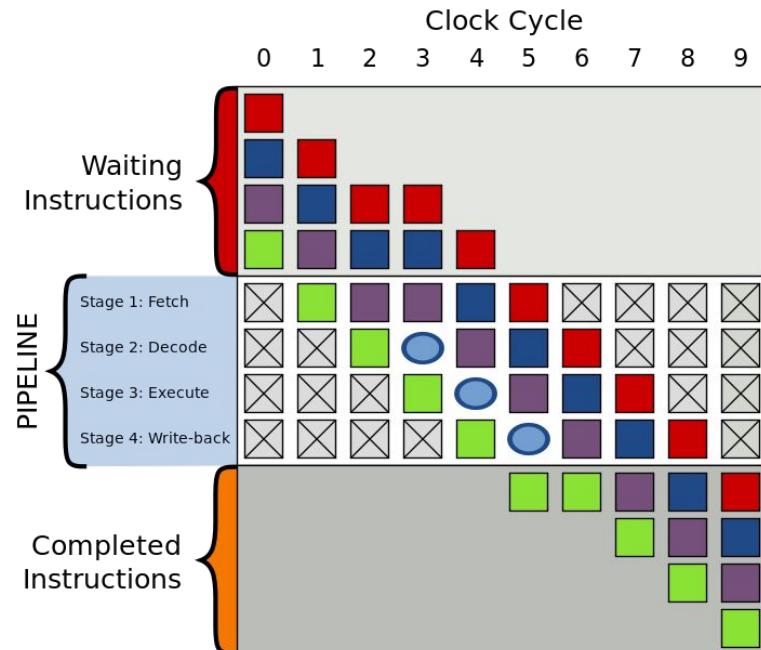
product_sk = 69: 0, 4, 12, 2 (0 zeros, 4 ones, 12 zeros, 2 ones)

product_sk = 74: 4, 1 (4 zeros, 1 one, rest zeros)

Figure 3-11. Compressed, bitmap-indexed storage of a single column.

Scanning Performance \propto Bandwidth

- CPU Caches < Memory < Disk < Network
- Make efficient use of CPU cycles, keep the CPU fed!
 - Exploit pipelining with “tight loops” (avoid bubbles)



“Next time you are developing an important algorithm, try pondering that a cache-miss is a lost opportunity to have executed ~500 CPU instructions! “ - Martin Thompson

Vectorized query engines

- Two key differences from tuple-at-a-time engines(e.g. Postgres):
 - **Column oriented processing** – Write query processing algorithms that operate on columns as long as possible in the execution plan. Refrain from working on tuples till late in the plan (e.g. during projecting result-set back to the user)
 - **Push batches of column vectors through**

Vectorized query processing

- **Better cache locality and efficient utilization of CPU cache** – we can quickly loop through tightly packed values of a column and do the necessary processing – predicate evaluation, arithmetic computations etc. Cache lines are filled with related values (from the same column) as opposed to heterogeneous values from multiple

Writing columns

- Update-in-place approaches are not possible with compressed columns
- If you want to insert a row in the middle of a table, you most likely have to rewrite all the columns. **As rows are identified by their position within a column, the insertion has to update all columns consistently**
- Query engines often augment column files with LSM-trees
 - All writes first go to an in-memory store
 - When enough writes have accumulated, they are merged with the column files on disk and written to new files in bulk
 - Queries need to examine both the column data on disk and the recent writes in memory, and combine the two
 - The query optimizer hides this distinction from the user, such that inserts, updates, and deletes are immediately reflected in subsequent

Sorting of columns

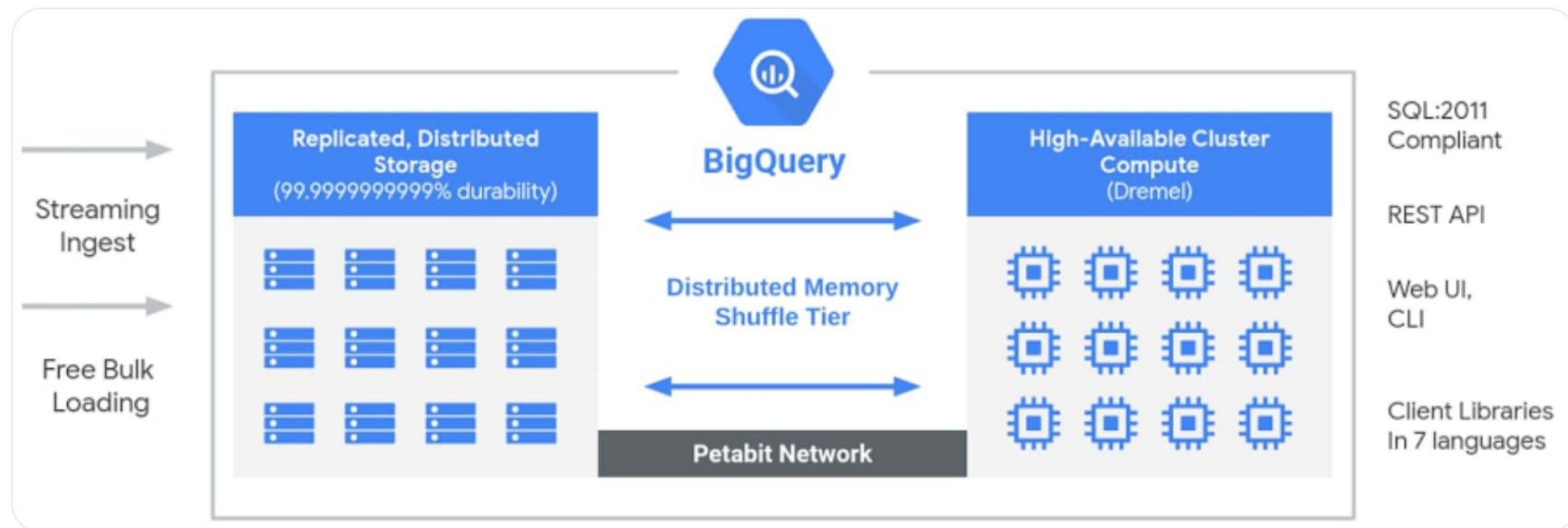
- You can choose the columns by which the table should be sorted, given advance knowledge of common queries
- e.g. **if queries often target date ranges, such as “last month”, sort by date_key first**, then the query optimizer can scan only the rows from the last month, which will be much faster than scanning all rows

The Rise of Cloud Data Warehouses

- A cloud data warehouse makes no trade-offs from a traditional data warehouse, but extends capabilities and runs on a fully managed service in the cloud
- Cloud data warehousing offers instant scalability to meet changing business requirements and powerful data processing to support complex analytical queries

BigQuery Architecture

BigQuery's serverless architecture decouples storage and compute and allows them to scale independently on demand. This structure offers both immense flexibility and cost controls for customers because they don't need to keep their expensive compute resources up and running all the time. This is very different from traditional node-based cloud data warehouse solutions or on-premise massively parallel processing (MPP) systems. This approach also allows customers of any size to bring their data into the data warehouse and start analyzing their data using Standard SQL without worrying about database operations and system engineering.



BigQuery Architecture



EVERYTHING IS EASIER IN THE DATA CLOUD

Snowflake delivers ease of use, instant elasticity, and lower TCO.

[START FOR FREE](#)

[WHY SNOWFLAKE?](#)



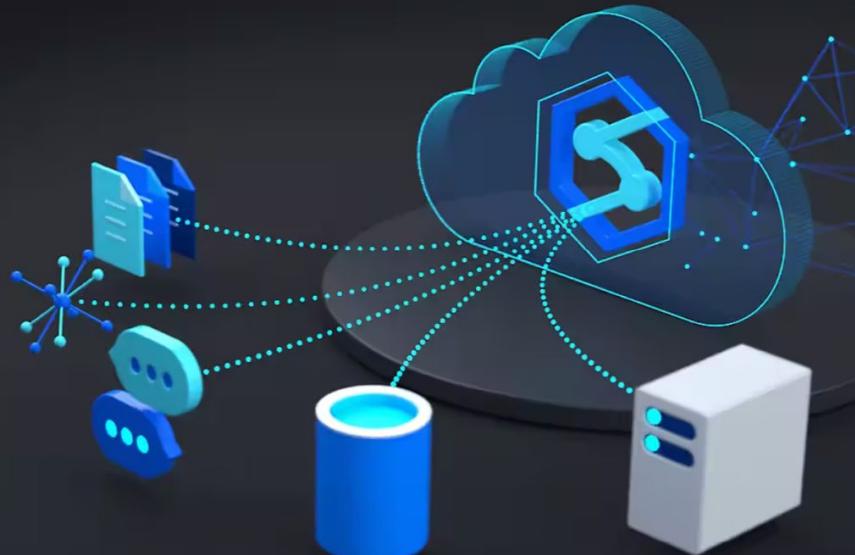


Azure Synapse Analytics

Accelerate time to insight across enterprise data warehouses and big data systems.

[Try Azure Synapse Analytics free](#)

[Create a pay-as-you-go account](#)

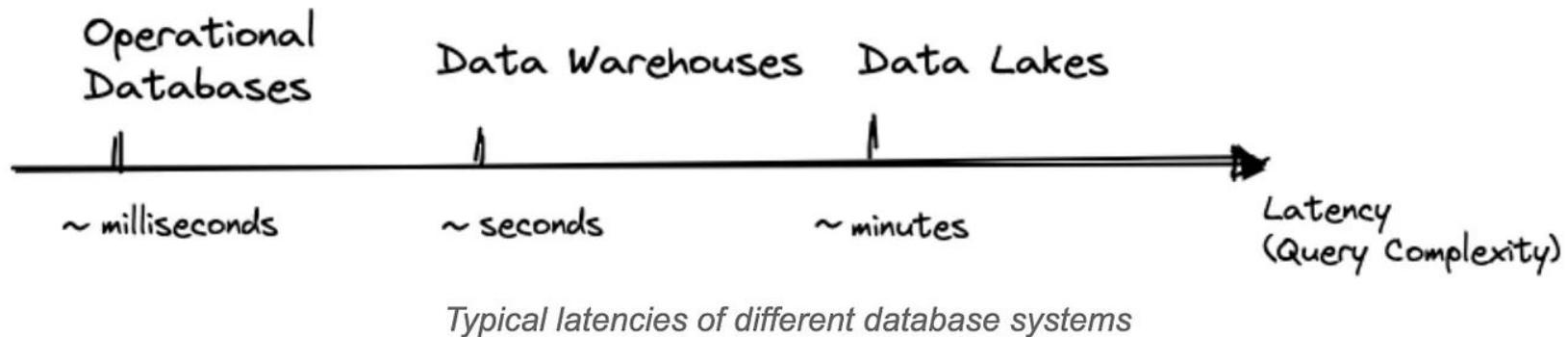


[Overview](#) [Features](#) [Security](#) [Pricing](#) [Get started](#) [Customer stories](#) [Resources](#) [FAQ](#)

Experience a new class of data analytics

Azure Synapse Analytics is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems. It brings together the best of SQL technologies used in enterprise data warehousing, Apache Spark technology for big data processing, and Data Explorer for log and time series analytics.

“HTAP” (Hybrid Transactional/Analytical Processing) – one system to rule them all?



Azure Synapse Link for Azure Cosmos DB: Near real-time analytics use cases

Article • 10/12/2022 • 7 contributors

Feedback

In this article

[Supply chain analytics, forecasting & reporting](#)

[Real-time personalization](#)

[IOT predictive maintenance](#)

[Sample scenario: HTAP for Azure Cosmos DB](#)

[Next steps](#)

APPLIES TO: NoSQL MongoDB Gremlin

Azure Synapse Link for Azure Cosmos DB is a cloud native hybrid transactional and analytical processing (HTAP) capability that enables you to run near real-time analytics over operational data. Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.

You might be curious to understand what industry use cases can leverage this cloud native HTAP capability for near real-time analytics over operational data. Here are three common use cases for Azure Synapse Link for Azure Cosmos DB:

AlloyDB for PostgreSQL

Announcing AlloyDB AI for building generative AI applications with PostgreSQL. [Read the blog.](#)

AlloyDB for PostgreSQL

[Benefits](#)[Key features](#)[Customers](#)[What's new](#)[Documentation](#)[Compare features](#)[All features](#)[Pricing](#)[Partners](#)[Take the next step](#)

AlloyDB for PostgreSQL

A fully managed PostgreSQL-compatible database service for your most demanding enterprise workloads. AlloyDB combines the best of Google with PostgreSQL, for superior performance, scale, and availability.

[Go to console](#)[Documentation](#)

- ✓ Fully compatible with PostgreSQL, providing flexibility and true portability for your workloads
- ✓ Superior performance, 4x faster than standard PostgreSQL for transactional workloads*
- ✓ Fast, real-time insights, up to 100x faster analytical queries than standard PostgreSQL*
- ✓ [AlloyDB AI](#) can help you build a wide range of generative AI applications

Working with zero-ETL integrations

Getting started with zero-ETL integrations

Creating zero-ETL integrations

Adding and querying data

Viewing and monitoring zero-ETL integrations

Deleting zero-ETL integrations

Troubleshooting zero-ETL integrations

▶ Using Aurora Serverless v2

▶ Using Aurora Serverless v1

▶ Using the Data API

▶ Using the query editor

▶ Code examples

Best practices with Aurora

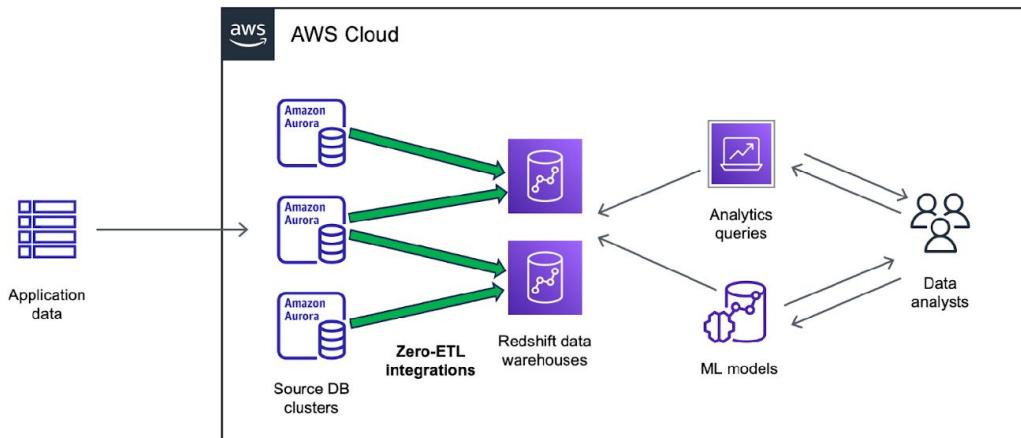
Performing an Aurora proof of concept

▶ Security

Quotas and constraints

To create a zero-ETL integration, you specify an Aurora DB cluster as the *source*, and an Amazon Redshift data warehouse as the *target*. The integration replicates data from the source database into the target data warehouse.

The following diagram illustrates this functionality:



The integration monitors the health of the data pipeline and recovers from issues when possible. You can create integrations from multiple Aurora DB clusters into a single Amazon Redshift namespace, enabling you to derive insights across multiple applications.

For information about pricing for zero-ETL integrations, see [Amazon Aurora pricing](#) and [Amazon Redshift pricing](#).

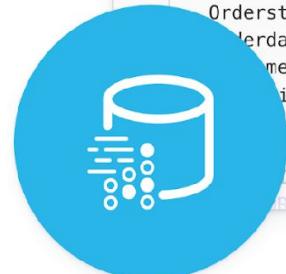




WORKLOADS

SNOWFLAKE UNISTORE

Simplify development by uniting transactional and analytical data.



```
1 USE DATABASE UNISTORE_139;
2 USE WAREHOUSE WH_UNISTORE_139;
3 USE SCHEMA UNISTORE_STREAMLIT_ORDERING_APP;
4
5 /* Orders Data Model */
6 CREATE OR REPLACE HYBRID TABLE Orders (
7     Orderkey number(38,0) PRIMARY KEY AUTOINCREMENT,
8     Itemkey number(38,0) FOREIGN KEY REFERENCES MenuItem(Itemkey),
     Orderstatus varchar(20),
     Orderdate timestamp_ntz DEFAULT current_timestamp::timestamp_ntz,
     Customername varchar(50),
     index_o_orderdate(Orderdate)
9
10 /* Items Data Model */
11 CREATE OR REPLACE HYBRID TABLE MenuItem (
```

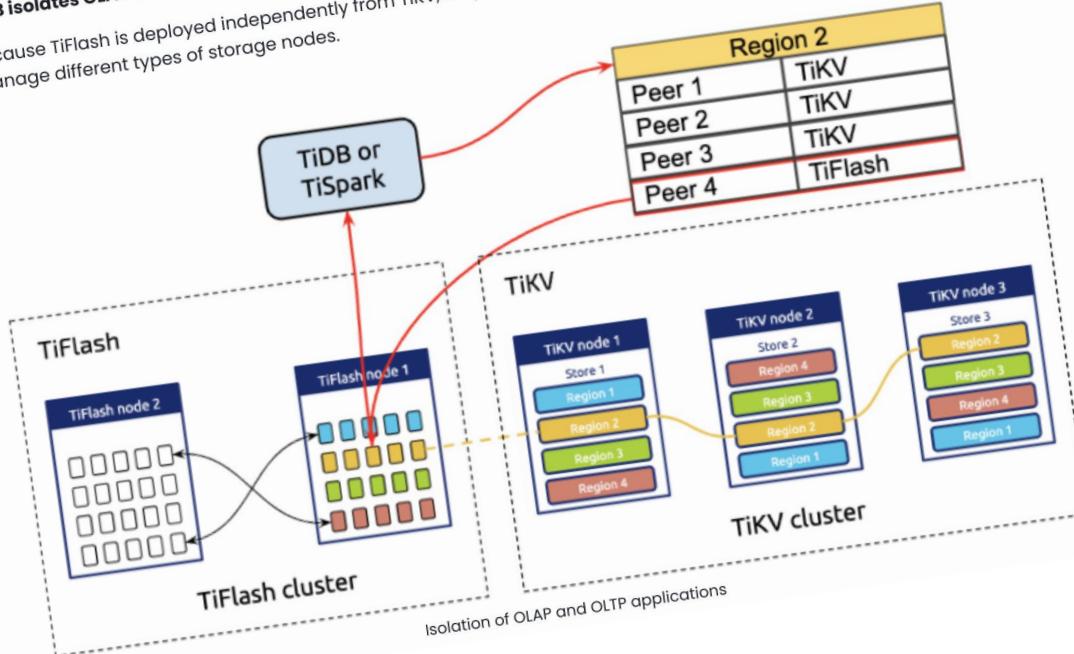
SHARE:



Minimizing the effect of OLAP applications on OLTP applications

TiDB isolates OLAP and OLTP applications, minimizing the effect of OLAP on OLTP.

Because TiFlash is deployed independently from TiKV, it's possible to isolate hardware resources. TiDB uses labels to manage different types of storage nodes.



[Product](#)[Solutions](#)[Docs](#)[Resources](#)[Company](#)[Docs](#)[Book a Demo](#)[Sign In](#)[Try Free >](#)

SHARE:



Transactions + Analytics SQL + NoSQL. Real-time AI. All in one.

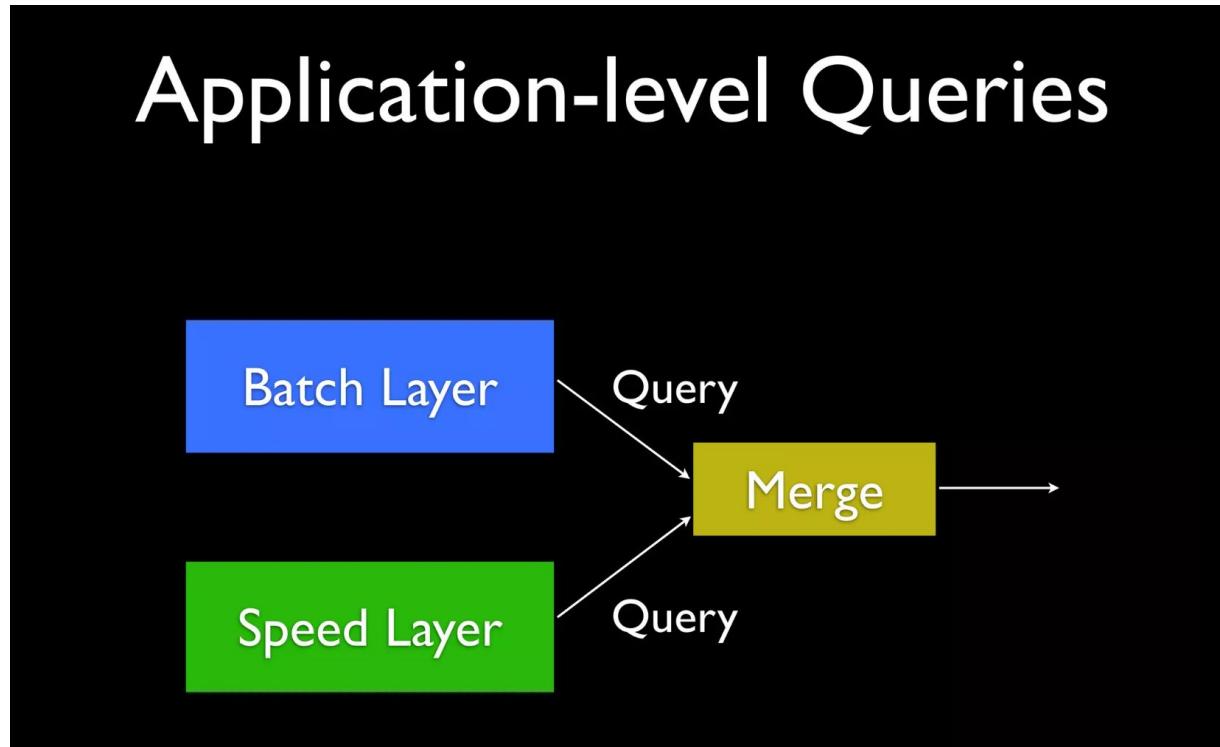
Transact, analyze and contextualize your data in real time.

SingleStoreDB empowers the world's makers to build, deploy and scale modern, intelligent applications — leading to real-time decisions, lasting customer experiences.

SingleStoreDB is a distributed SQL database that offers high-throughput transactions (inserts and upserts), low-latency analytics and context from real-time vector data.

SingleStoreDB meets you wherever you are in your cloud journey — giving you flexibility to deploy wherever you need: self-managed on-premises, or as a fully managed cloud service.

The Lambda Architecture for “real time” analytics



Columnar data
is good for
batch

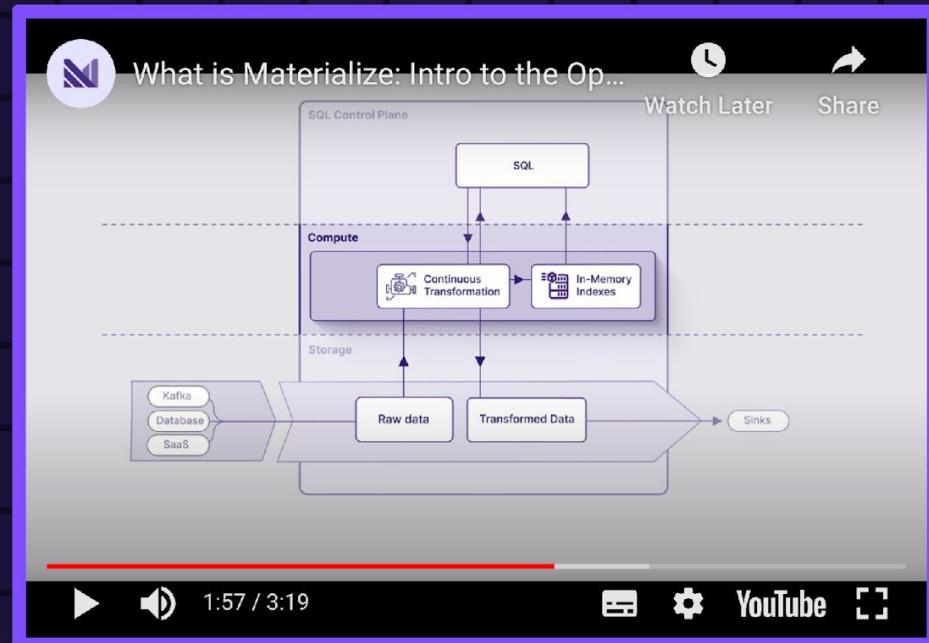
See also:



Google
DataFlow

The Operational Data Warehouse for Better Business Outcomes

Materialize is a cloud data warehouse with streaming internals, built for work that needs **action** on what's happening **right now**.

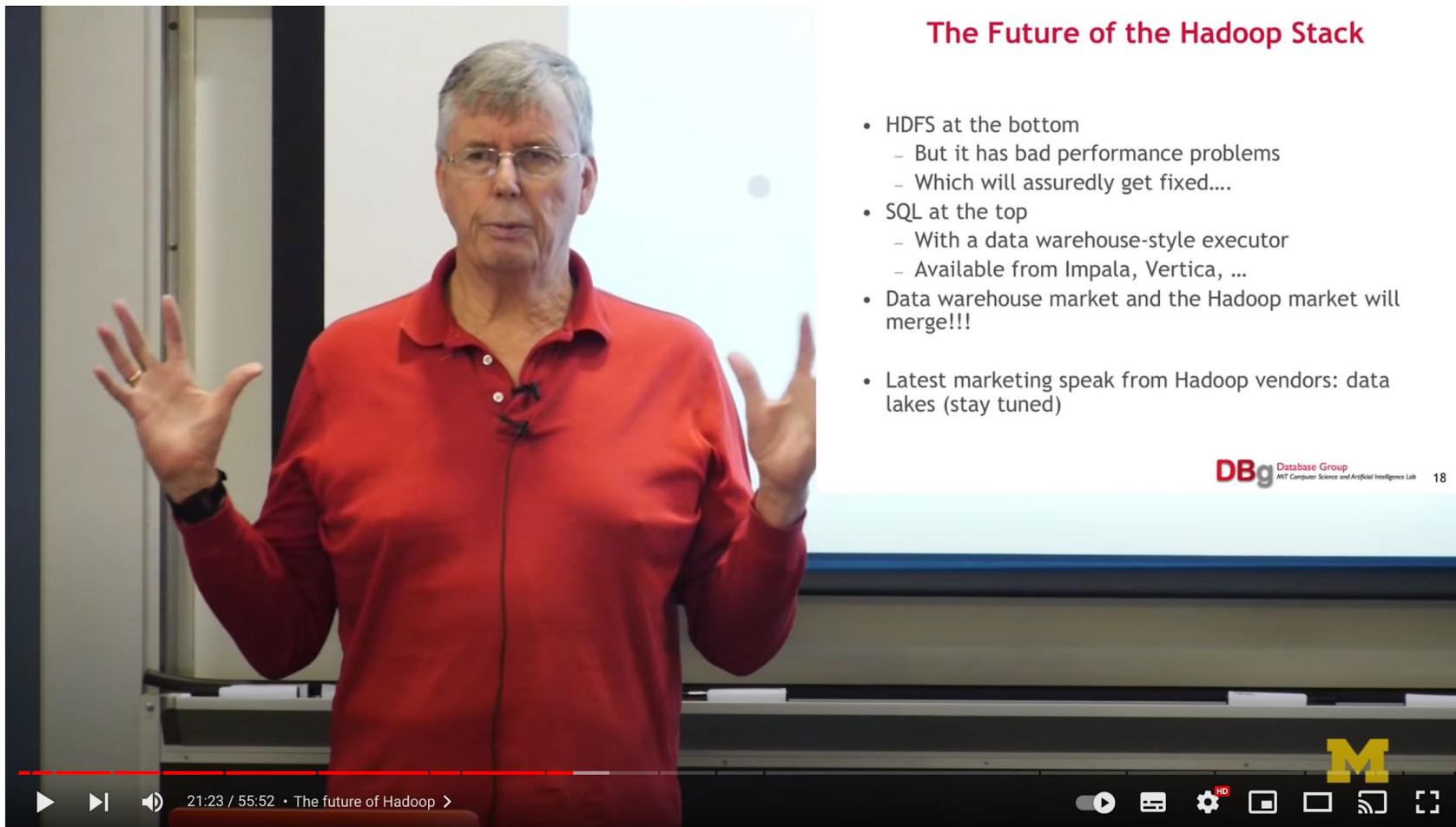
[GET A DEMO ▾](#)[GET STARTED](#)

R&D

The Future of the Hadoop Stack

- HDFS at the bottom
 - But it has bad performance problems
 - Which will assuredly get fixed....
- SQL at the top
 - With a data warehouse-style executor
 - Available from Impala, Vertica, ...
- Data warehouse market and the Hadoop market will merge!!!
- Latest marketing speak from Hadoop vendors: data lakes (stay tuned)

DBG Database Group
MIT Computer Science and Artificial Intelligence Lab 18



Michael Stonebraker | Big Data is (at least) Four Different Problems



Michigan Engineering
44.6K subscribers

Subscribe

380



Share

Download

Clip

Save

...

25K views 6 years ago

Big Data is (at least) Four Different Problems

Michael Stonebraker, Co-Director, Intel Science & Technology Center, MIT

Foundations and Trends® in Databases
Vol. 5, No. 3 (2012) 197–280
© 2013 D. Abadi, P. Boncz, S. Harizopoulos,
S. Idreos and S. Madden
DOI: 10.1561/1900000024



The Design and Implementation of Modern Column-Oriented Database Systems

Daniel Abadi
Yale University
dna@cs.yale.edu

Peter Boncz
CWI
P.Boncz@cwi.nl

Stavros Harizopoulos
Amiato, Inc.
stavros@amiato.com

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu



Databases

Authors and titles for cs.DB in Aug 2023

[total of 86 entries: 1-25 | 26-50 | 51-75 | 76-86]
[showing 25 entries per page: fewer | more | all]

[25] [arXiv:2308.08702](#) [pdf, other]

Finding a Second Wind: Speeding Up Graph Traversal Queries in RDBMSs Using Column-Oriented Processing

Mikhail Firsov, Michael Polyntsov, Kirill Smirnov, George Chernishev

Subjects: **Databases (cs.DB)**; Performance (cs.PF)

[9] [arXiv:2309.06051](#) [pdf, other]

OmniSketch: Efficient Multi-Dimensional High-Velocity Stream Analytics with Arbitrary Predicates

Wieger R. Punter, Odysseas Papapetrou, Minos Garofalakis

Subjects: **Databases (cs.DB)**

[19] [arXiv:2309.11322](#) [pdf, other]

Vector database management systems: Fundamental concepts, use-cases, and current challenges

Toni Taipalus

Comments: 12 pages, 5 figures

Subjects: **Databases (cs.DB)**

<https://www.scattered-thoughts.net/log/0040/>

0040: olap survey, lobster, feldera, innovation, wizard papers, umbra papers, olap papers

Published 2023-09-29

I published [a shallow survey of OLAP and HTAP query engines](#).

The last 2/3rds or so of this post contains all the supporting notes. Also a lot of papers on strategies for low-latency compilation.

<https://www.scattered-thoughts.net/log/0041/>

[Column Sketches: A Scan Accelerator for Rapid and Robust Predicate Evaluation](#) (2018)

Goal is to accelerate column scans regardless of data distribution and workload.

Build a histogram of data. Divide into evenly full buckets. Give each bucket a short binary code.

Evaluate predicates against buckets first, and then scan for matching buckets.

Their experiments demonstrate good performance across different query selectivities with 1 byte codes over uniformly distributed 32 bit integers.

Performance remains identical under data skew, while bitweaving degrades.

Builds much faster than bitweaving too. (Note that BtrBlocks builds histograms anyway - might be able to produce the sketch almost for free.)

Mainlining Databases: Supporting Fast Transactional Workloads on Universal Columnar Data File Formats

Tianyu Li
litanyu@mit.edu
MIT

Wan Shen Lim
wanshenl@cs.cmu.edu
Carnegie Mellon University

Matthew Butrovich
mbutrovi@cs.cmu.edu
Carnegie Mellon University

Wes McKinney
wes@ursalabs.org
Ursa Labs

Amadou Ngom
angom@cs.cmu.edu
Carnegie Mellon University

Andrew Pavlo
pavlo@cs.cmu.edu
Carnegie Mellon University

ABSTRACT

The proliferation of modern data processing tools has given rise to open-source columnar data formats. The advantage of these formats is that they help organizations avoid repeatedly converting data to a new format for each application. These formats, however, are read-only, and organizations must use a heavy-weight transformation process to load data from on-line transactional processing (OLTP) systems. We aim to reduce or even eliminate this process by developing a storage architecture for in-memory database management systems (DBMSs) that is aware of the eventual usage of its data and emits columnar storage blocks in a universal open-source format. We introduce relaxations to common analytical data formats to efficiently update records and rely on a lightweight transformation process to convert blocks to a read-optimized layout when they are cold. We also describe how to access data from third-party analytical tools with minimal serialization overhead. To evaluate our work, we implemented our storage engine based on the Apache Arrow format and integrated it into the DB-X DBMS. Our experiments show that our approach achieves comparable performance with dedicated OLTP DBMSs while enabling orders-of-magnitude faster data exports to external data science and machine learning tools than existing methods.

1 INTRODUCTION

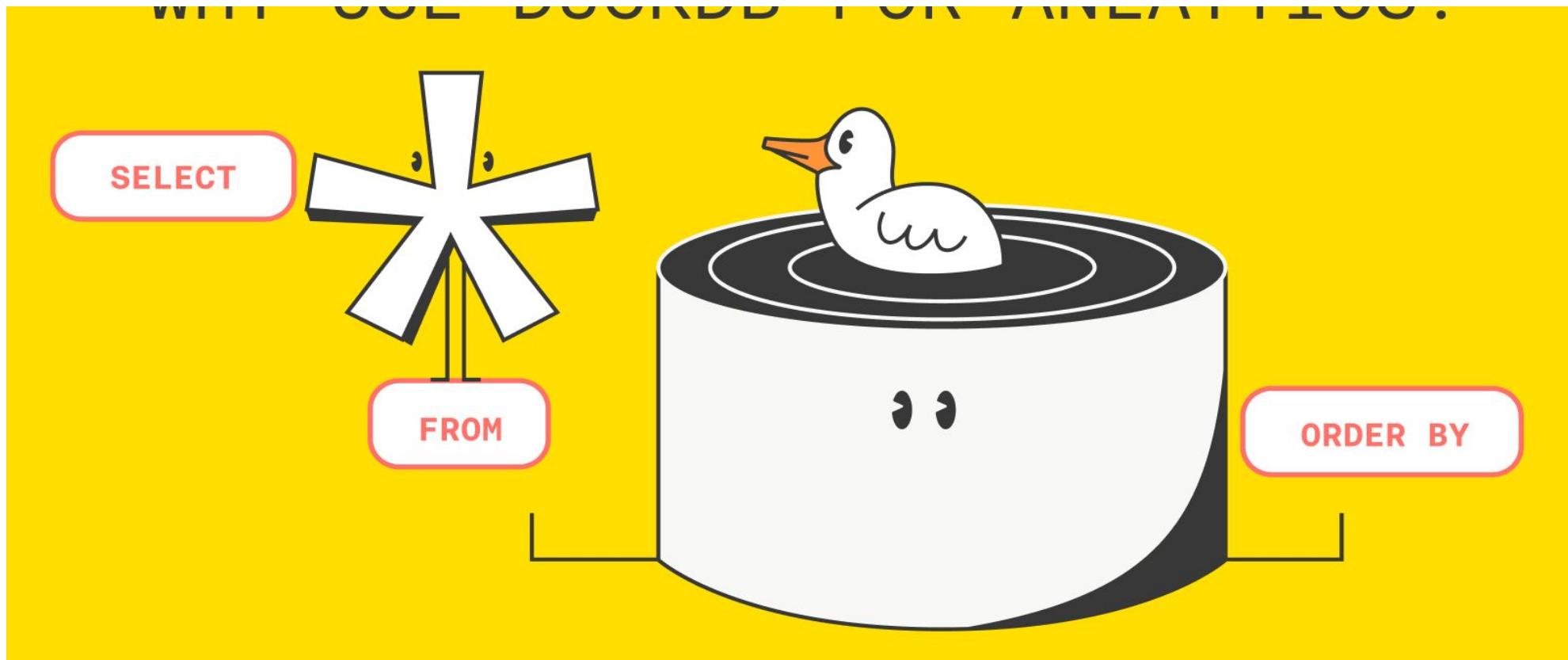
Data analysis pipelines allow organizations to extrapolate insights from data residing in their OLTP systems. The tools in

Although a DBMS can perform some analytical duties, modern data science workloads often involve specialized frameworks, such as TensorFlow, PyTorch, and Pandas. Organizations are also heavily invested in personnel, tooling, and infrastructure for the current data science eco-system of Python tools. We contend that the need for DBMS to efficiently export large amounts of data to external tools will persist. To enable analysis of data as soon as it arrives in a database is, and to deliver performance gains across the entire data analysis pipeline, we should look to improve a DBMS's interoperability with external tools.

If an OLTP DBMS directly stores data in a format used by downstream applications, the export cost is just the cost of network transmission. The challenge in this is that most open-source formats are optimized for read/append operations, not in-place updates. Meanwhile, divergence from the target format in the OLTP DBMS translates into more transformation overhead when exporting data, which can be equally detrimental to performance. A viable design must seek equilibrium in these two conflicting considerations.

To address this challenge, we present a multi-versioned DBMS that operates on a relaxation of an open-source columnar format to support efficient OLTP modifications. The relaxed format can then be transformed into the canonical format as data cools with a light-weight in-memory process. We implemented our storage and concurrency control architecture in **DB-X** [10] and evaluated its performance. We target Apache Arrow, although our approach is also appli-

DuckDB...



End-to-end-query optimization



- Expression rewriting
- Join ordering
- Subquery flattening
- Filter / projection pushdown
 - * Automatic in DuckDB
 - * Manual in Pandas



```
1 # filter out the rows
2 filtered_df = t[ t['a'] > 0]
3 # perform the aggregate
4 result = filtered_df.groupby(['b']).agg(
5     Min=('a', 'min')
6 )
```



DuckDB

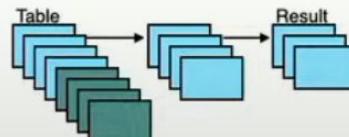




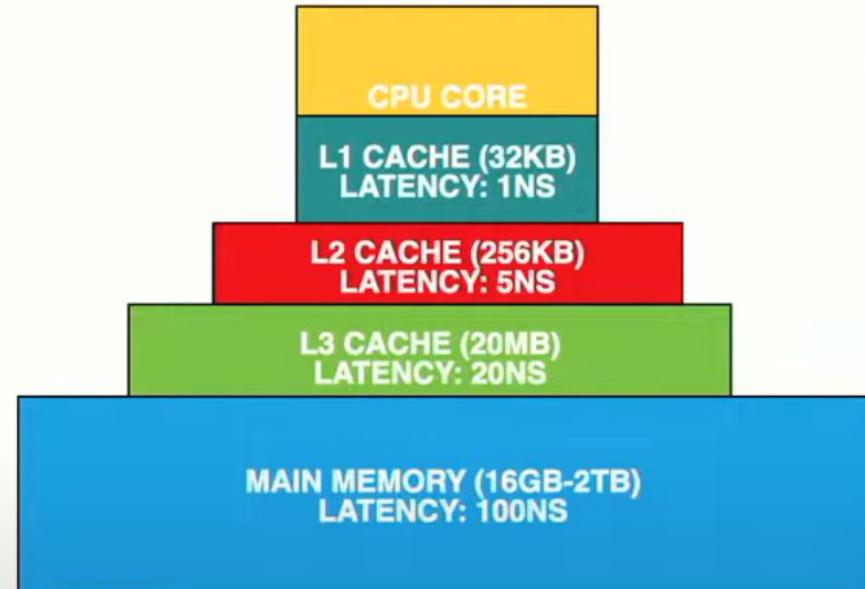
Execution

- Vectorized Processing (DuckDB)
 - Optimized for CPU Cache locality
 - SIMD instructions, Pipelining
 - Small intermediates
(ideally fit in L1 cache)

Vectorized Processing



DuckDB



 tech.ml.dataset

Public

Sponsor

Watch 21

Fork 32

Starred 599



master

5 branches

401 tags

Go to file

Add file

<> Code

About

A Clojure high performance data processing system

Readme

EPL-1.0 license

Activity

599 stars

21 watching

32 forks

Report repository

Releases

401 tags

| | | | |
|------------------------------|---|---------------------|---------------|
| ezand | Bump fastexcel version and expose 'stableId' (#385) | cdb71cf 3 weeks ago | 1,645 commits |
| .clj-kondo | Fixing metamorph and adding codegen step to project.clj. | 2 years ago | |
| .github | Run automated tests with different JDK versions (#378) | last month | |
| dev-resources | Parquet write simplification (#228) | 2 years ago | |
| dev/tech/v3/dataset | Fixing metamorph and adding codegen step to project.clj. | 2 years ago | |
| docs | Release 7.021 | last month | |
| graal-native/tech/v3/dataset | Graal native 1 (#151) | 3 years ago | |
| java | Bump fastexcel version and expose 'stableId' (#385) | 3 weeks ago | |
| java_public_api/tech/v3 | Hamf integration (#337) | last year | |
| java_test | 6.069 | last year | |
| neanderthal/tech/v3/dataset | Tests pass with nil key eliding. (#364) | 3 months ago | |
| scripts | Test that changing the type of an int doesn't break the join (#380) | last month | |
| src/tech/v3 | Bump fastexcel version and expose 'stableId' (#385) | 3 weeks ago | |

Accessing this from Clojure, through TMD, is also easy:

```
user> (require '[tmducken.duckdb :as duckdb])
nil
user> (require '[tech.v3.dataset :as ds])
nil
user> (duckdb/initialize!)
Sep 06, 2023 11:00:12 AM clojure.tools.logging$eval7454$fn__7457 invoke
INFO: Attempting to load duckdb from "./binaries/libduckdb.so"
true
user> (def db (duckdb/open-db "data.ddb"))
#'user/db
user> (def conn (duckdb/connect db))
#'user/conn
user> (time (duckdb/sql->dataset conn "SELECT COUNT(*) AS n FROM data"))
"Elapsed time: 10.305756 msecs"
:_unnamed [1 1]:
|      n |
|-----:|
| 4000000000 |
```

Object Store Table Formats – columns without a database

- 1) Open standards for huge, petabyte-scale analytic tables that are accessible for heterogeneous processing/querying - using Spark DataFrames, SparkSQL, Dremio, Trino/Presto, etc.
- 2) “ACID for Big Data”
- 3) “Think about data, not about files” - “a table format is a way to organize a dataset’s files to

Leading Open Table Formats

- 1) Databricks' Delta Lake
- 2) Apache Hudi - donated by Uber, built to avoid batch processing with Spark+HDFS - focussed on eventing
- 3) Apache Iceberg - donated by Netflix, built to replace Hive due to correctness issues, also used by Apple, Twitter, Expedia, etc.



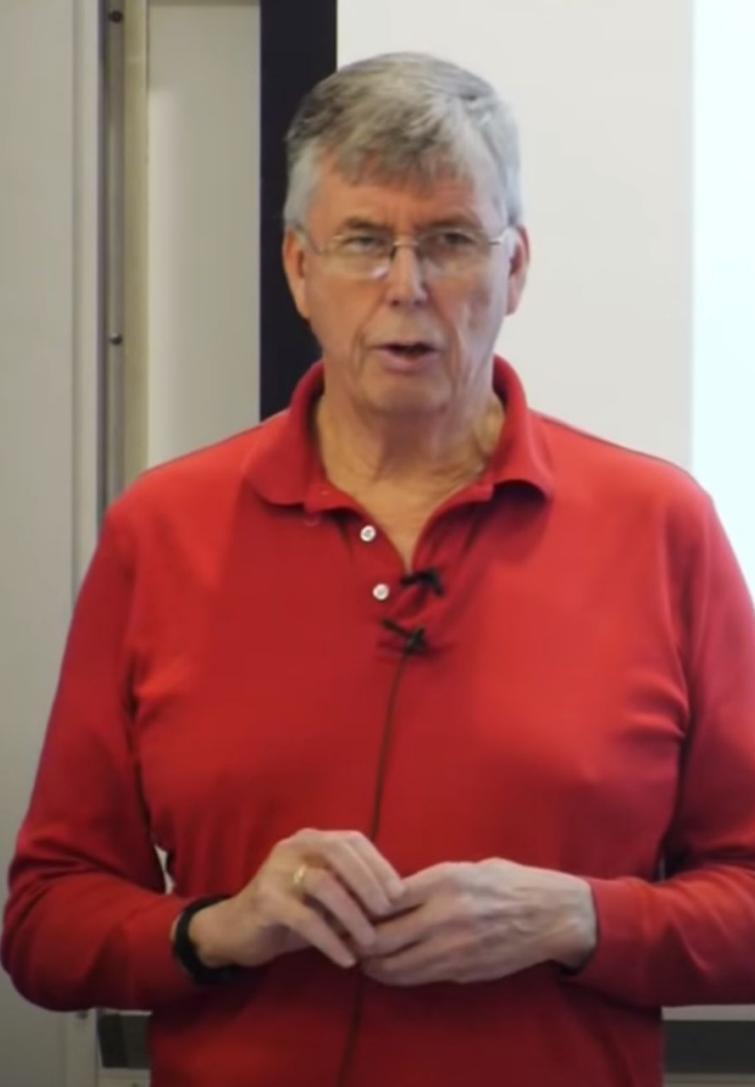
A Thorough Comparison of Delta Lake, Iceberg and Hudi

A Quick Comparison

2020-05

| | Delta Lake (open source) | Apache Iceberg | Apache Hudi |
|---------------------------|--|--|--|
| Transaction (ACID) | Y | Y | Y |
| MVCC | Y | Y | Y |
| Time travel | Y | Y | Y |
| Schema Evolution | Y | Y | Y |
| Data Mutation | Y (update/delete/merge into) | N | Y (upsert) |
| Streaming | Sink and source for spark struct streaming | Sink and source(wip) for Spark struct streaming, Flink (wip) | DeltaStreamer HiveIncrementalPuller |
| File Format | Parquet | Parquet, ORC, AVRO | Parquet |
| Compaction/Cleanup | Manual | API available (Spark Action) | Manual and Auto |
| Integration | DSv1, Delta connector | DSv2, InputFormat | DSv1, InputFormat |
| Multiple language support | Scala/java/python | Java/python | Java/python |
| Storage Abstraction | Y | Y | N |
| API dependency | Spark-bundled | Native/Engine bundled | Spark-bundled |
| Data ingestion | Spark, presto, hive | Spark, hive | DeltaStreamer |

2023 Update:
*Still no clear winner
(but probably Iceberg)*

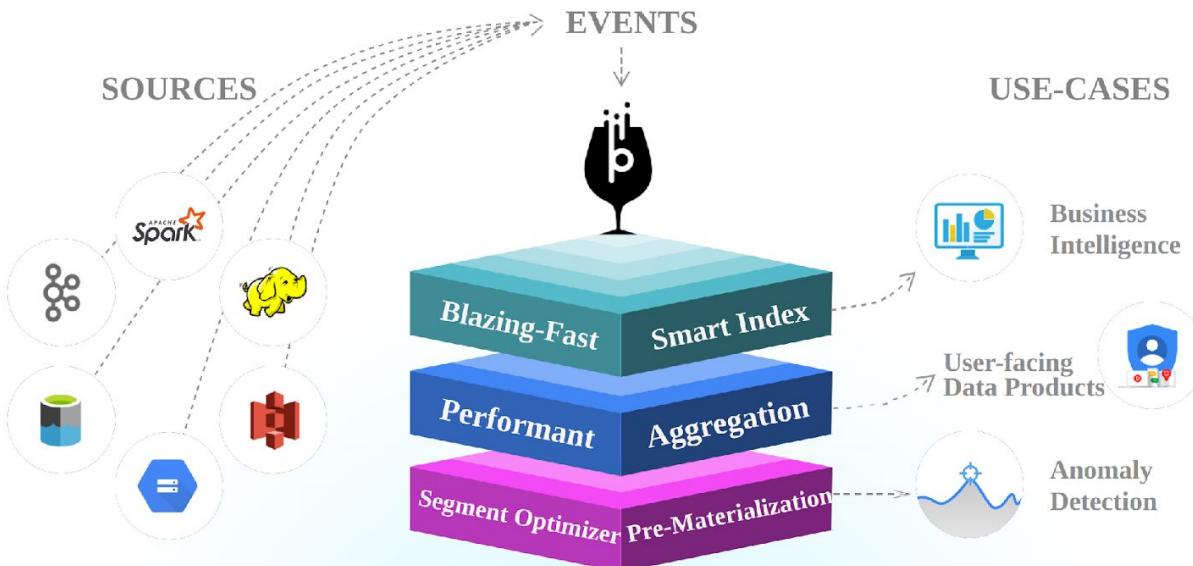


Take away One Size Does Not Fit All

- Column store (stupid analytics)
- Array store (smart analytics)
- Streaming (one velocity solution)
- New SQL (another velocity solution)
- No SQL (low end; semi-structured data)
- Legacy stuff (in place now - but obsolete)
- One or more curation systems (800 pound gorilla)
- Use the right tool for the job!!!!!

Apache Pinot™

Realtime distributed OLAP datastore, designed to answer OLAP queries with low latency



Getting Started

Join our Slack

Pinot is proven at **scale in LinkedIn** powers 50+ user-facing apps and serving **100k+ queries**

About ClickHouse

**Blazing fast**

Exceeds other column-oriented database management systems

**Linearly scalable**

Incredible scaling both horizontally and vertically

**Fault tolerant**

Supports async replication and can be deployed across multiple datacenters

**Hardware efficient**

Processes analytical queries faster than traditional row-oriented systems

**Highly reliable**

Purely distributed system, including enterprise-grade security

**Feature-rich**

User-friendly SQL query dialect, built-in analytics capabilities, and more

Why choose ClickHouse?

Blazing fast

ClickHouse uses all available hardware to its full potential to process each query as fast as possible. Peak processing performance for a single query stands at more than 2 terabytes per second (after decompression, only used columns). In distributed setup reads are automatically balanced among healthy replicas to avoid increasing latency.

Fault-tolerant

ClickHouse supports multi-master asynchronous replication and can be deployed across multiple datacenters. All nodes are equal, which allows avoiding having single points of failure. Downtime of a single node or the whole datacenter won't affect the system's availability for both reads and writes.

Easy to use

ClickHouse is simple and works out-of-the-box. It streamlines all your data processing: ingest all your structured data into the system and it becomes instantly available for building reports. SQL dialect allows expressing the desired result without involving any custom non-standard API that could be found in some DBMS.

Highly reliable

ClickHouse can be configured as a purely distributed system located on independent nodes, without any single points of failure. It also includes a lot of enterprise-grade security features and fail-safe mechanisms against human errors.

Welcome to TileDB Cloud!

TUTORIALS

Start Here!

Product Tour

Serverless Compute 101

Task Graphs 101

Use Cases



CONCEPTS

Universal Data Management

Serverless Compute

Console and API

TileDB Cloud Internals



Pricing and Billing

Marketplace

HOW TO

TileDB Cloud is based on the [TileDB Embedded](#) open-source universal storage engine, which models and efficiently stores all data as (dense or sparse) **multi-dimensional arrays**, providing a common API and a large number of APIs and tool integrations.



↑ ↑ Pluggable Compute: Efficient APIs & Tool Integrations ↓ ↓

TileDB Cloud

- ❑ Access control and logging
- ❑ Serverless SQL, UDFs, task graphs
- ❑ Jupyter notebooks and dashboards

Unified data management
and easy serverless compute
at global scale

TileDB Embedded

- ❑ Data versioning & time traveling
- ❑ Columnar, cloud-optimized
- ❑ Parallel IO, rapid reads & writes

Open-source interoperable
storage with a universal
open-spec array format



The TileDB Cloud stack at a glance

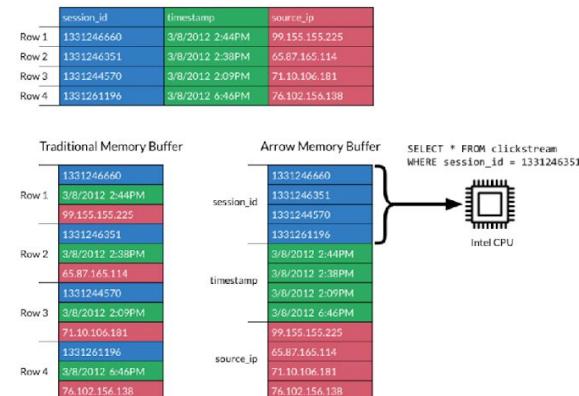
Apache Arrow Overview

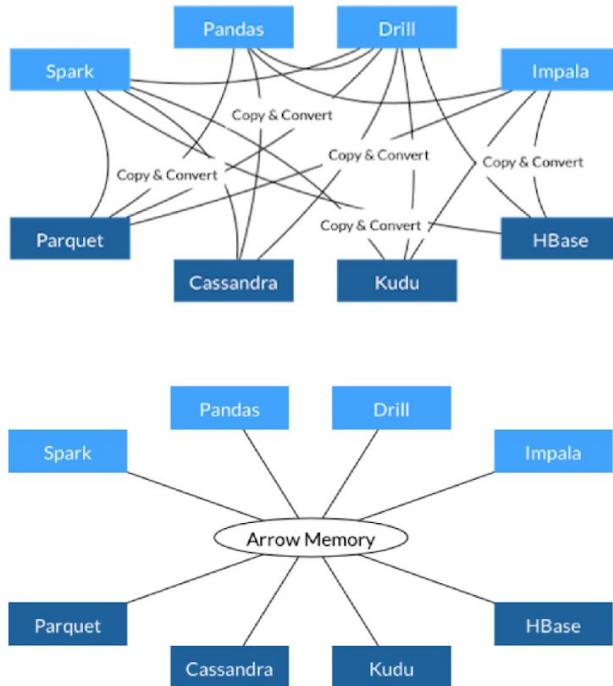
Apache Arrow is a software development platform for building high performance applications that process and transport large data sets. It is designed to both improve the performance of analytical algorithms and the efficiency of moving data from one system or programming language to another.

A critical component of Apache Arrow is its **in-memory columnar format**, a standardized, language-agnostic specification for representing structured, table-like datasets in-memory. This data format has a rich data type system (included nested and user-defined data types) designed to support the needs of analytic database systems, data frame libraries, and more.

Columnar is Fast

The Apache Arrow format allows computational routines and execution engines to maximize their efficiency when scanning and iterating large chunks of data. In particular, the contiguous columnar layout enables vectorization using the latest SIMD (Single Instruction, Multiple Data) operations included in modern processors.





Arrow Libraries

The Arrow project contains libraries that enable you to work with data in the Arrow columnar format in many languages. The [C++](#), [C#](#), [Go](#), [Java](#), [JavaScript](#), [Julia](#), and [Rust](#) libraries contain distinct implementations of the Arrow format. These libraries are [integration-tested](#) against each other to ensure their fidelity to the format. In addition, Arrow libraries for [C \(Glib\)](#), [MATLAB](#), [Python](#), [R](#), and [Ruby](#) are built on top of the C++ library.

Standardization Saves

Without a standard columnar data format, every database and language has to implement its own internal data format. This generates a lot of waste. Moving data from one system to another involves costly serialization and deserialization. In addition, common algorithms must often be rewritten for each data format.

Arrow's in-memory columnar data format is an out-of-the-box solution to these problems. Systems that use or support Arrow can transfer data between them at little-to-no cost. Moreover, they don't need to implement custom connectors for every other system. On top of these savings, a standardized memory format facilitates reuse of libraries of algorithms, even across languages.

Voltron Data

- Key steward of Apache Arrow

Augment Rather Than Replatform

Data systems need to be modular and composable because tech stacks are always evolving. Don't rip and replace, rather augment with open industry standards.

Hardware Accelerated Analytics is Here

Accelerated computing is now accessible for analytics. Customize and compose your tech stack with hardware standards to increase performance and scale data at a reduced cost.

Work in Your Language of Choice

Developers, analysts, and data scientists should be able to work in their programming language of choice and seamlessly operate on the same data inside whatever engine they choose. We make that possible.

Our Customers

Meta



snowflake

Quadrature

1.2 Standardizing on Arrow

Arrow is an open source project that enables developers to efficiently build fast, interoperable data systems based on open standards. The Arrow project ticks all the boxes for a solid standard:

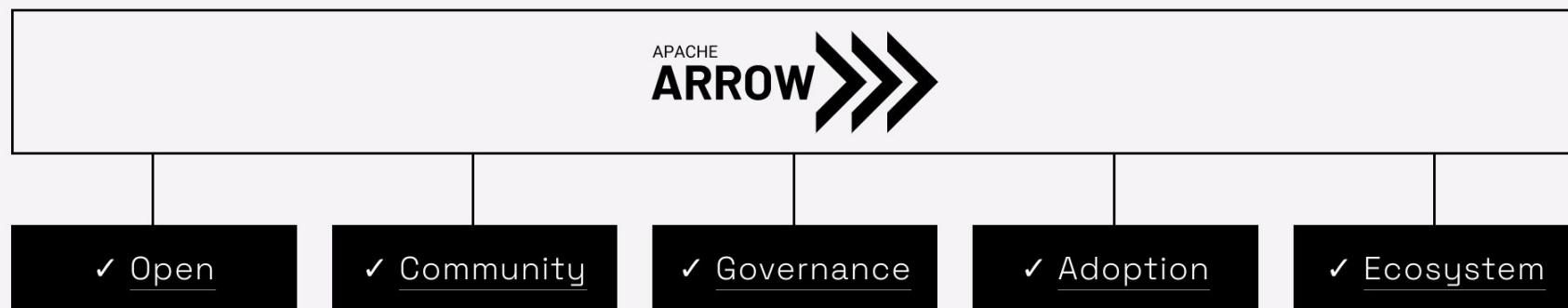


Figure 01.06. Arrow is a standard that we build with and trust, in part based on these five factors.

1.2.1 The Arrow format

Arrow started as a standardized in-memory format for structured tabular data. Why start there? Because when you are building data-intensive analyses and applications, systems get stuck on two main tasks:

1. **Moving data**

When a workload is **transport-bound** (or input/output[I/O]-bound), the speed of execution depends on the rate of transfer of data into or out of a system.

2. **Processing data**

When a workload is **compute-bound**, the speed of execution depends on the speed of the processor, whether it is a CPU, GPU, or another type of hardware.

A screenshot of a presentation slide titled "Apache Arrow". The slide features a sidebar on the left with green boxes containing links to various Apache Arrow components: Columnar Format, Interprocess and in-process (C) protocols, Query Engine Components, IO / File Format Backends, Flight & FlightSQL, and JIT Expression Compilation. The main content area contains the Apache Arrow logo (the word "APACHE" above "ARROW" with three right-pointing arrows) and a text block: "Apache Arrow is **doing for data analytics what LLVM did for compiler infrastructure**. Modular, reusable software components for building high-performance analytics systems." Below this text is a grid of logos for various programming languages and tools: C, C++, C#, Go, JavaScript, MATLAB, Python, R, and Scala. The bottom of the slide has a decorative blue wave graphic.

Apache Arrow

APACHE ARROW >>>

Apache Arrow is **doing for data analytics what LLVM did for compiler infrastructure**. Modular, reusable software components for building high-performance analytics systems.

C C++ C# =GO JS

MATLAB Python R Scala

Columnar Format
Interprocess and in-process (C) protocols
Query Engine Components
IO / File Format Backends
Flight & FlightSQL
JIT Expression Compilation

4

8:01 PM · Dec 15, 2021 · Twitter Web App

1 Retweet 1 Quote Tweet 10 Likes

Jeremy Taylor @refiset ...

8:01 PM · Dec 15, 2021 · Twitter Web App

1 2 10

Andy Pavlo
@andy_pavlo

Arrow has become a toolkit of components for building an OLAP DBMS: file format, execution engine, file storage, expression eval, networking protocol/transport.

If you've ever worked on a DBMS, you know that these things are hard to build. Arrow provides them for you.

8:01 PM · Dec 15, 2021 · Twitter Web App

1 Retweet 1 Quote Tweet 10 Likes

Tweet your reply

Andy ... @a... · Dec 15, 2021 ...

Replying to @andy_pavlo

The idea of composable DBMS parts is not new. Back the late 1990s, @surajitc proposed building RISC-style database components.

DuckDB quacks Arrow: A zero-... CMU_DB_12062021.pptx apache/arrow: Apache Arrow is ... CMU_DB_12062021.pptx.pdf

32 / 65 | - 114% + ↻ 🔍

Network of query engines that “speak” Arrow

Arrow-native
use Arrow internally

Dremio
DataFusion/Ballista/Polars
C++ execution engine (unnamed)
RAPIDS/BlazingSQL

Support Arrow C-data interface
columnar data exchange

DuckDB
Velox
(ClickHouse maybe someday)

Arrow-supported
can process and output Arrow data

BigQuery
Snowflake

Jeremy Taylor @refiset ...

6 28 ↻ ↑

To build a complete DBMS using [@ApacheArrow](#) you still need a SQL parser, optimizer, buffer pool manager, and storage layer.
[@duckdb](#) is a good example of how to do this, although they don't use Arrow's Flight SQL and other parts.
[@InfluxDB](#) is working on a new Arrow-engine too.

8:01 PM · Dec 15, 2021 · Twitter Web App

5 Retweets 1 Quote Tweet 28 Likes

Tweet your reply Reply



Search projects



Help

Sponsors

Log in

Register

pyarrow 14.0.1



[Latest version](#)

`pip install pyarrow`



Released: Nov 8, 2023

Python library for Apache Arrow

Navigation

 Project description

 Release history

 Download files

Project description

Python library for Apache Arrow

 [pypi](#) v14.0.1  [conda-forge](#)

This library provides a Python API for functionality provided by the Arrow C++ libraries, along with tools for Arrow integration and interoperability with pandas, NumPy, and other software in the Python ecosystem.

DuckDB is an in-process SQL OLAP database management system

[Installation ↓](#)[Documentation](#)[Live Demo](#)

Why DuckDB?



Simple

- In-process, serverless
- C++11, no dependencies, single file build
- APIs for Python/R/Java/...

[more →](#)

Feature-rich

- Transactions, persistence
- Extensive SQL support
- Direct Parquet & CSV querying

[more →](#)

Fast

- Vectorized engine
- Optimized for analytics
- Parallel query processing

[more →](#)

Free

- Free & Open Source
- Permissive MIT License

[more →](#)

Compare data engines running 17GB of Parquet data on GPUs and CPUs

Hardware

Blog | Single GPU Outperforms 88 CPUs by 2.5X



GPUs are essential for machine learning and AI at enterprise scale – and are becoming increasingly critical for data preprocessing workloads. Recent benchmarking showed how a single NVIDIA V100 GPU outperforms a total of 88 x86 CPU cores by ~2.5X. This also showed a single GPU running on RAPIDS outperforms Spark on an 88 core cluster by 20X. Get the full results in this featured resource.

| System Specifications | |
|-----------------------|------------------------------|
| GPUs | 2x V100 32 GB |
| GPU Memory | 64 GB Total |
| CPUs | 2x E5-2699V4, 88 Cores Total |
| System Memory | 1TB |
| Storage | 4TB, 4x 1TB NVMe |



↳ Benchmarking Data Engines

Swap Java-based engines with Velox for a 3X perf improvement



Report | Velox: 2023 Project to Watch



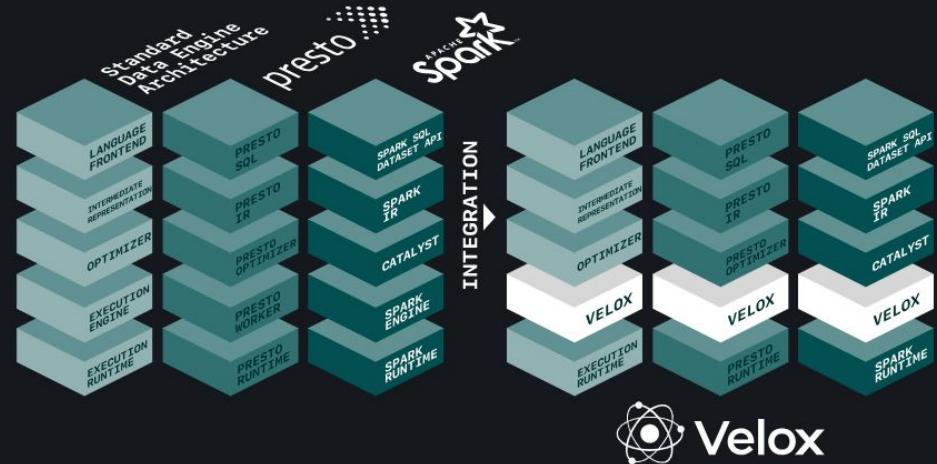
Velox is an embeddable columnar database engine designed for Arrow-based systems. This modular, unification standard benefits industries using and developing data management systems. Learn how Velox breaks down data silos and accelerates data processing.

[Velox GitHub ↗](#)

[Velox Data Thread Talk ↗](#)

[Velox Blog ↗](#)

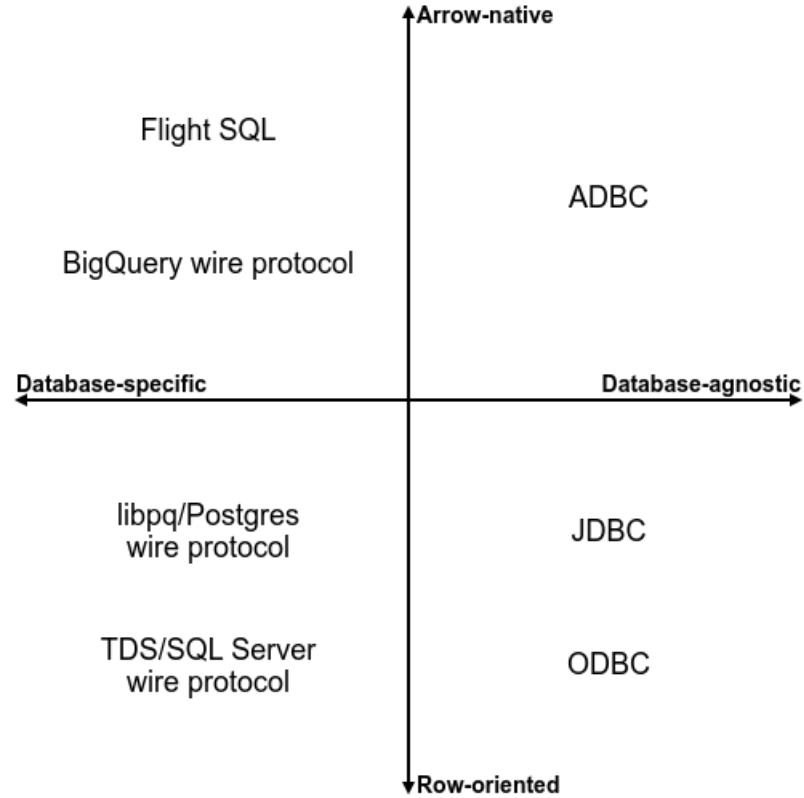
[Velox Research Paper from VLDB ↗](#)



↳ Velox Data Infrastructure

"Data management systems like Presto and Spark typically have their own execution engines and other components. Velox can function as a common execution engine across different data management systems."

Source: [Facebook Engineering Blog](#). Diagram by Phillip Bell.



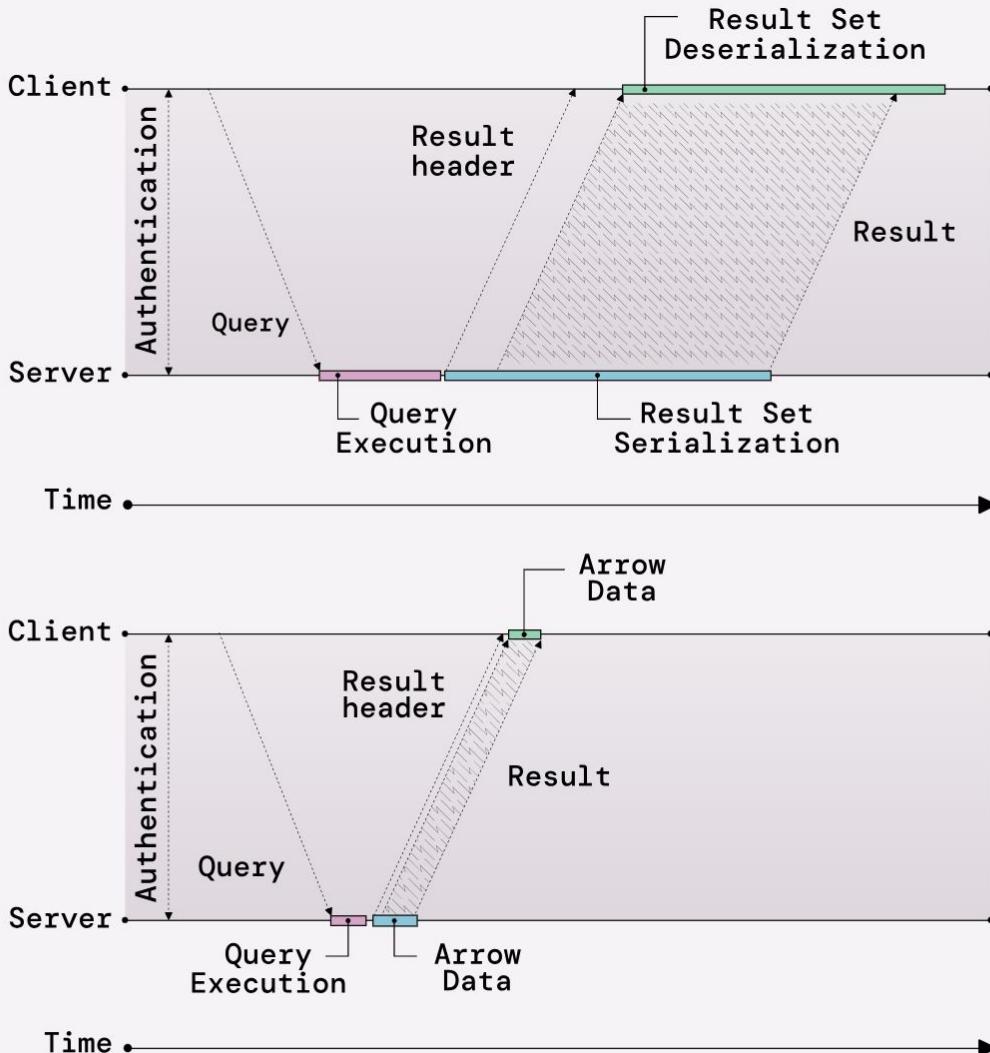
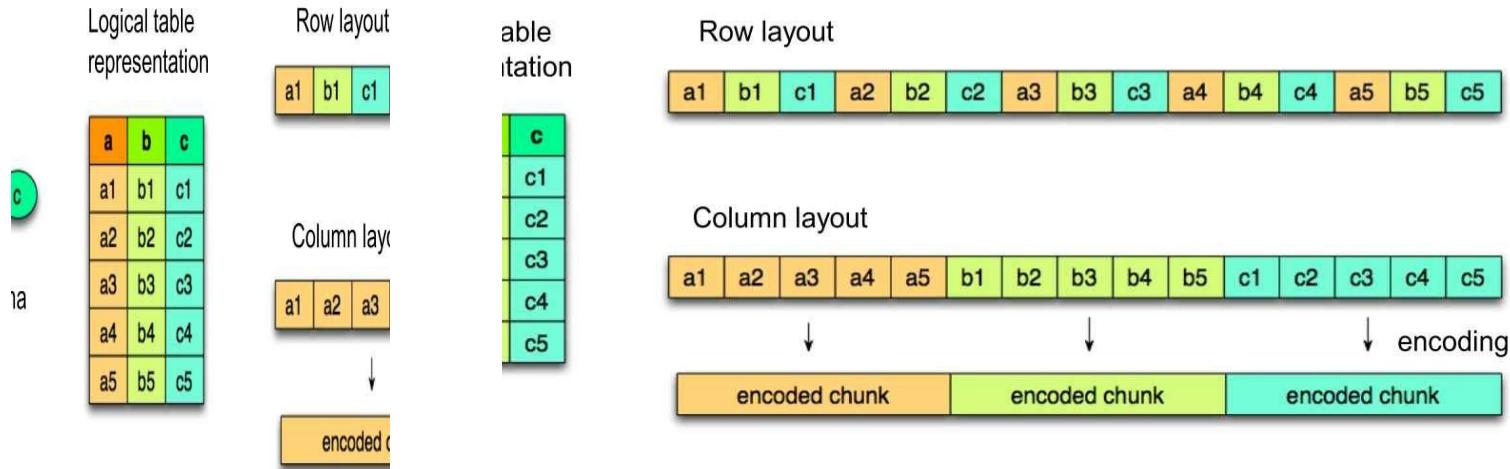


Figure 03.08.
 Comparison of typical client-server communication versus with Flight. Source:
[“Benchmarking Apache Arrow Flight - A wire-speed protocol for data transfer, querying and microservices”](#) by Ahmad et al.

Today's Topics

1. Columnar data 101
2. Context & trends
3. Modern analytic systems
4. R&D



THANK YOU 🙏
QUESTIONS?

jdt@juxt.pr
o
@refset