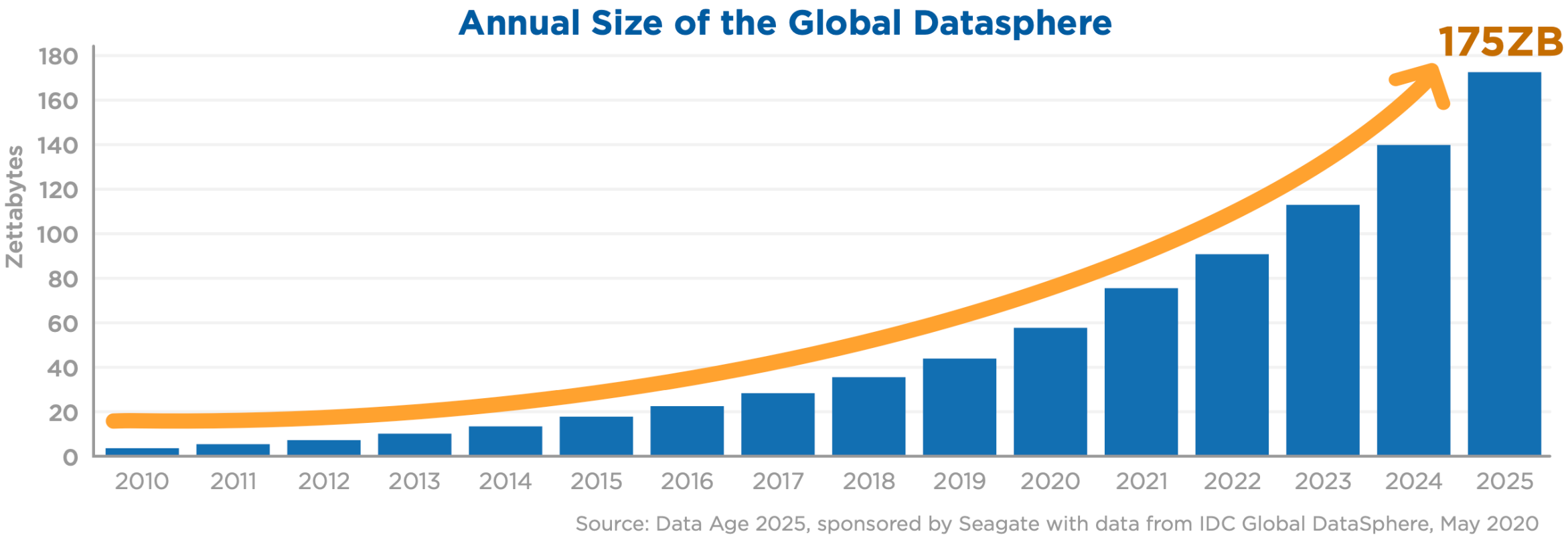
A Deep Dive into Common Open Formats for Analytical DBMSs

Chunwei Liu[1], Anna Pavlenko[2], Matteo Interlandi[2], Brandon Haynes[2]

[1]*MIT CSAIL*, [2]*Microsoft Gray Systems Lab*

The Data Deluge

Figure 1 – Annual Size of the Global Datasphere

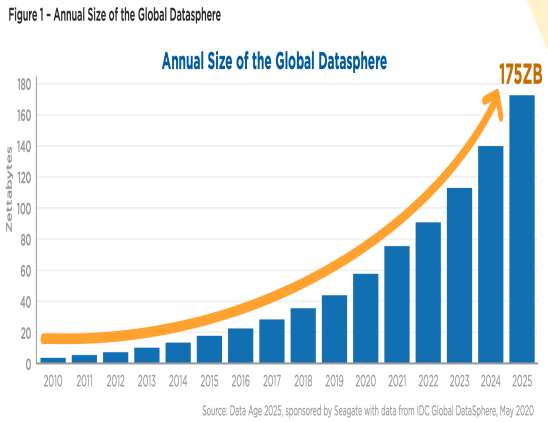


Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, May 2020

Popularity of Columnar Database

BIG DATA

OLAP System



Columnar File Formats

monetdb

presto

VERTICA

vectorwise

Azure Synapse Analytics

Spark

amazon REDSHIFT

Parquet

APACHE ARROW

Apache orc™

APACHE CarbonData™

Which one is better?

Apache Arrow vs. Parquet and ORC

data representation?

Apache Parquet and Ap

proposition revolves ar

data in a set of two dim

storage is one-dimensi

primary options for stor

the first column sequen

Big Data File Formats

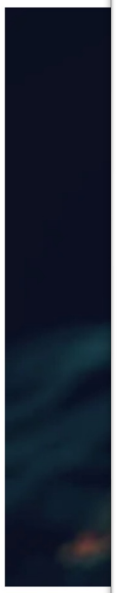
What are file formats? What are the common Hadoop file format features? Which format should you be using?

Wes McKinney

Some co

Apache

Parquet, Avro or ORC?



In this

comm

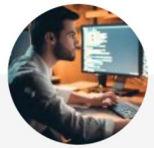
forma

studie

manag

ORC , PARQUET and Avro Highlighted

Created: Jun 22, 2022 10:23:23 2014 0 0 0 0



View the author 1#

Apache Arrow an

Why We Needed

Columnar Data, C

Memory

Apache Parquet and Apache Arrow both focus

analytics. These two projects optimize perform

By Julien LeDem, architect, Dremio.

Columnar data structures provide a number of p

row-oriented data structures for analytics. These

File forn

In this po

some con

Hadoop,

structured

It also su

You have

In fact, H

The data

the best

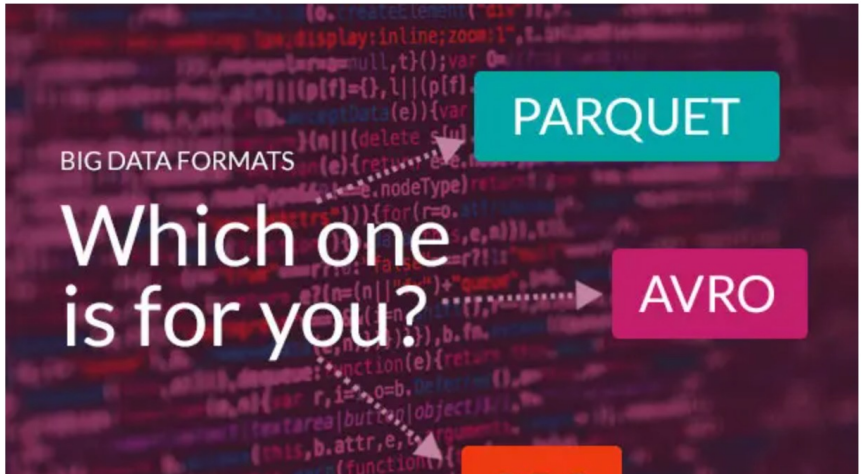
When y

are vari

case and

AWS ETL

How to choose between Parquet, ORC and AVRO for S3, Redshift and Snowflake?



Search ...

CATEGORIES

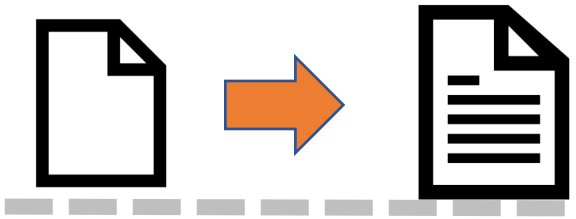
- Databricks Integration
- Oracle Data Replication
- SAP Data Replication
- AWS ETL
- Data Integration
- Snowflake Data Warehouse
- SQL Server Replication
- Press & Announcements

Query Workflow

OLAP Systems

amazon REDSHIFT, VERTICA, vectorwise, Azure Synapse, APACHE Spark, monetdb, presto

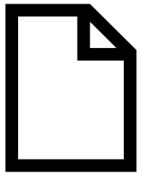
Conversion overhead



APACHE ARROW

Storage

Transmission & I/O overhead



Parquet, Apache orc, APACHE ARROW Feather

QUERY TIME STACK

- Query Evaluation
- Data Structure
- Decoding
- Decompression
- Parsing
- I/O
- Network Latency

Compiler optimization

In-situ query

SIMD

Data skipping

Storage format | In memory format

Open Columnar Formats

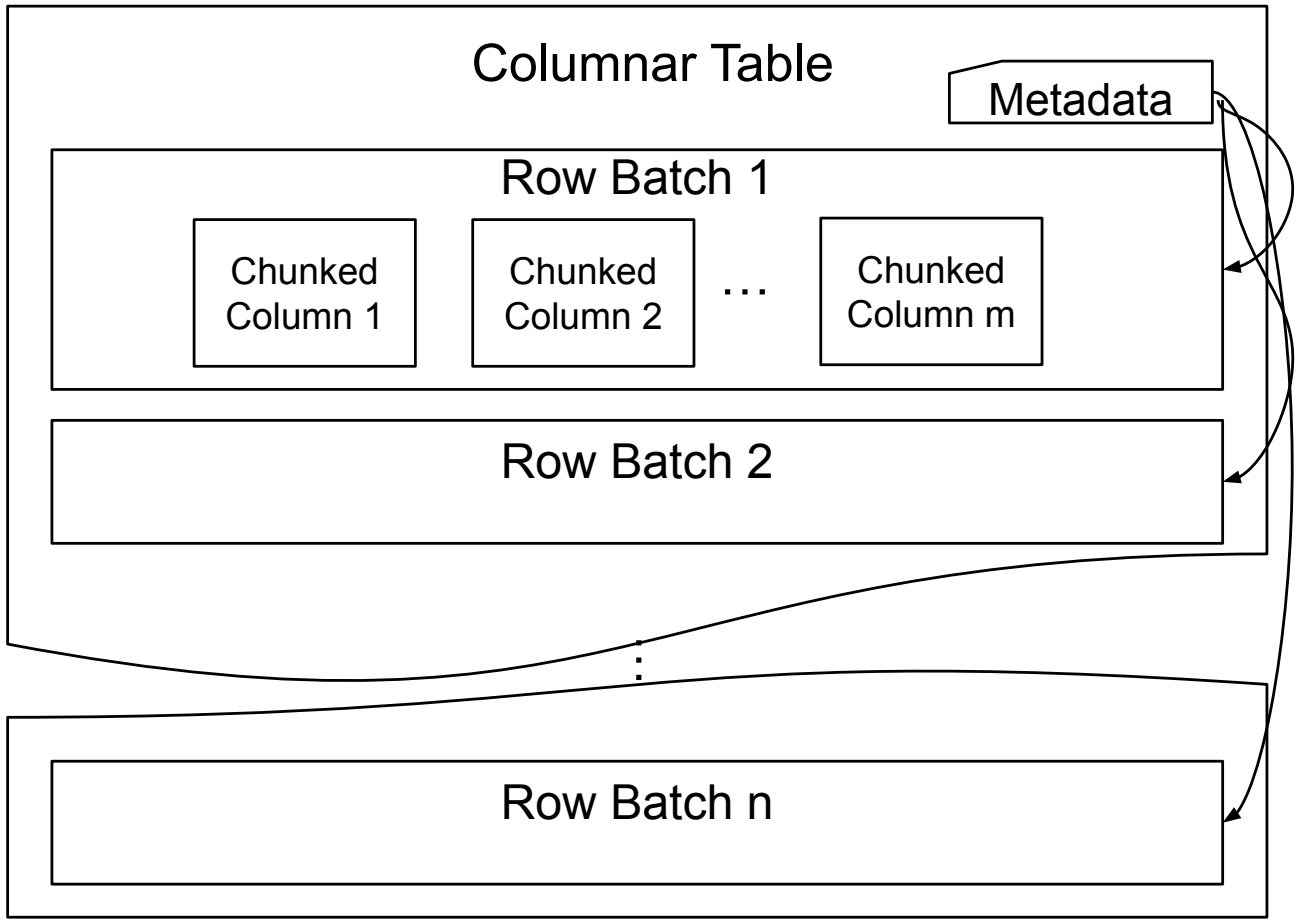


Figure A. Columnar format layout

Table 1. Column format name convention mapping

	Row Batch	Chunked Column
Arrow	Record Batch	Chunked Array
Parquet	Row Group	Column Chunk
ORC	Stripe	Row Column

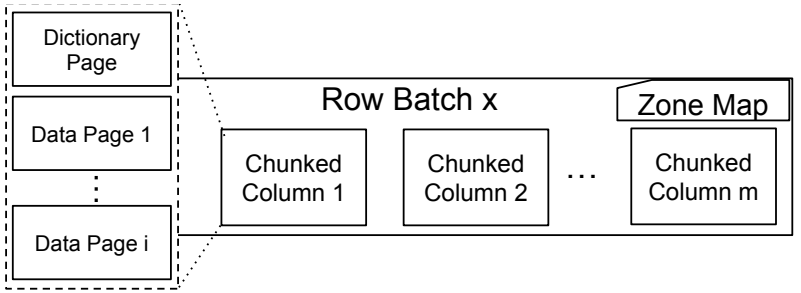


Figure B. A Parquet row batch

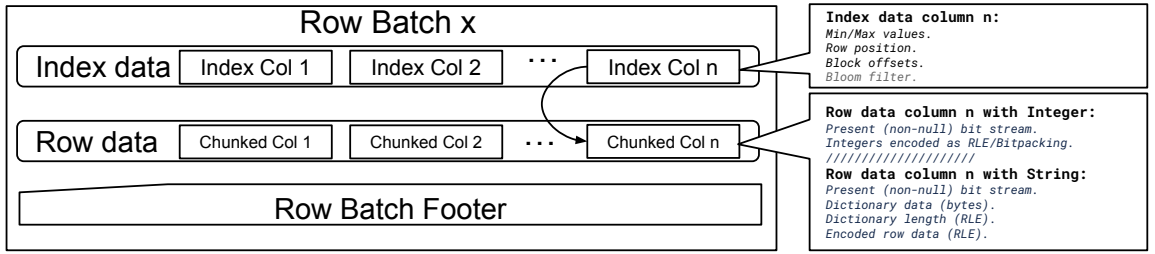


Figure C. An ORC row batch

Common features

	Encoding Methods	Compression Codecs	Data Skipping Granularity	Direct Query	Primary Purpose	Representative Systems
APACHE ARROW	DICT	None	Chunk	None	In-Memory Compute	Dremio, Spark, Pandas
APACHE ARROW Feather	DICT	Zstd, LZ4	None	None	On-Disk Storage	Pandas
Parquet	DICT, RLE, BP, Delta, ...	Gzip, Zstd, LZO, LZ4, Snappy, ...	Record	None	On-Disk Storage	Spark, Hive, Presto
Apache orc	DICT, RLE, BP, Delta	Snappy, Zlib, LZ4	Chunk	None	On-Disk Storage	Hive, Presto

Compression performance

Query performance

Methodology

To most fairly compare the formats, we evaluate each format across the following dimensions:

- Compression ratio
  - ~31K data columns extracted from real-world datasets, TPC-DS dataset (SF=10)
- Transcoding throughput
  - Compression and decompression overhead
- Data access
  - Micro-benchmarking over projection, predicate and bitmap evaluation
- End-to-end evaluation over subexpressions
  - End-to-end query performance over query subexpressions drawn from TPC-DS
- Advanced features
  - Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization

Evaluation outline

- **Compression ratio**
  - **~31K data columns extracted from real-world datasets, TPC-DS dataset**
- Transcoding throughput
  - Compression and decompression overhead
- Data access
  - Micro-benchmarking over projection, predicate and bitmap evaluation
- End-to-end evaluation over subexpressions
  - End-to-end query performance over query subexpressions drawn from TPC-DS
- Advanced features
  - Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization

Compression Ratio over Real World Datasets

Table I. Total size (in GB) by format for columns in the CodecDB, BI, and JOB datasets. (~31k cols)

Data Type	# Cols.	Raw Size	Parquet Size	ORC Size	Arrow Size	Arrow (DICT)
Integer	12k	57.3	9.8	13.5	59.3	59.3*
Float	7k	58.8	24.0	58.2	59.8	59.8*
String	13k	373.5	31.0	62.2	403.4	118.3
Total	31k	489.7	64.7	133.9	522.5	237.4
Compression Ratio (CR)			0.13	0.27	1.07	0.48

Table II: Average and stddev compression ratios by data type. (CR= compressed_size / original_size)

Type	Parquet		ORC		Arrow		ArrowDICT	
	AVG	STD	AVG	STD	AVG	STD	AVG	STD
Int	0.25	0.27	0.26	0.18	1.41	0.84	-	-
Float	0.34	0.26	1.43	1.00	1.49	1.09	-	-
String	0.21	0.34	0.22	0.31	1.54	0.68	0.92	0.87

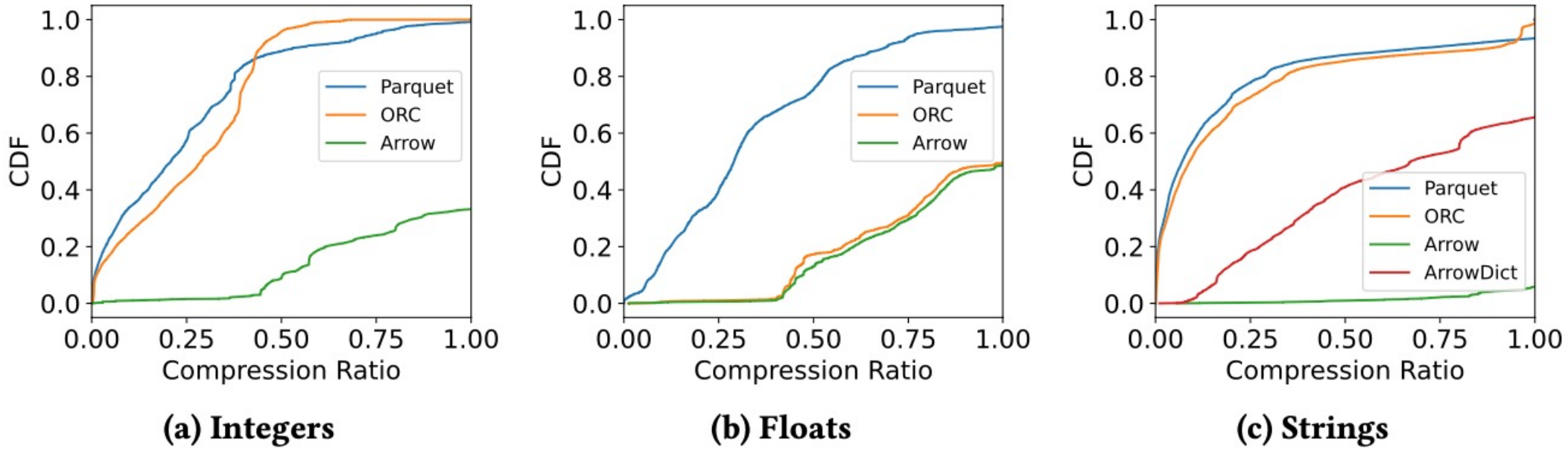
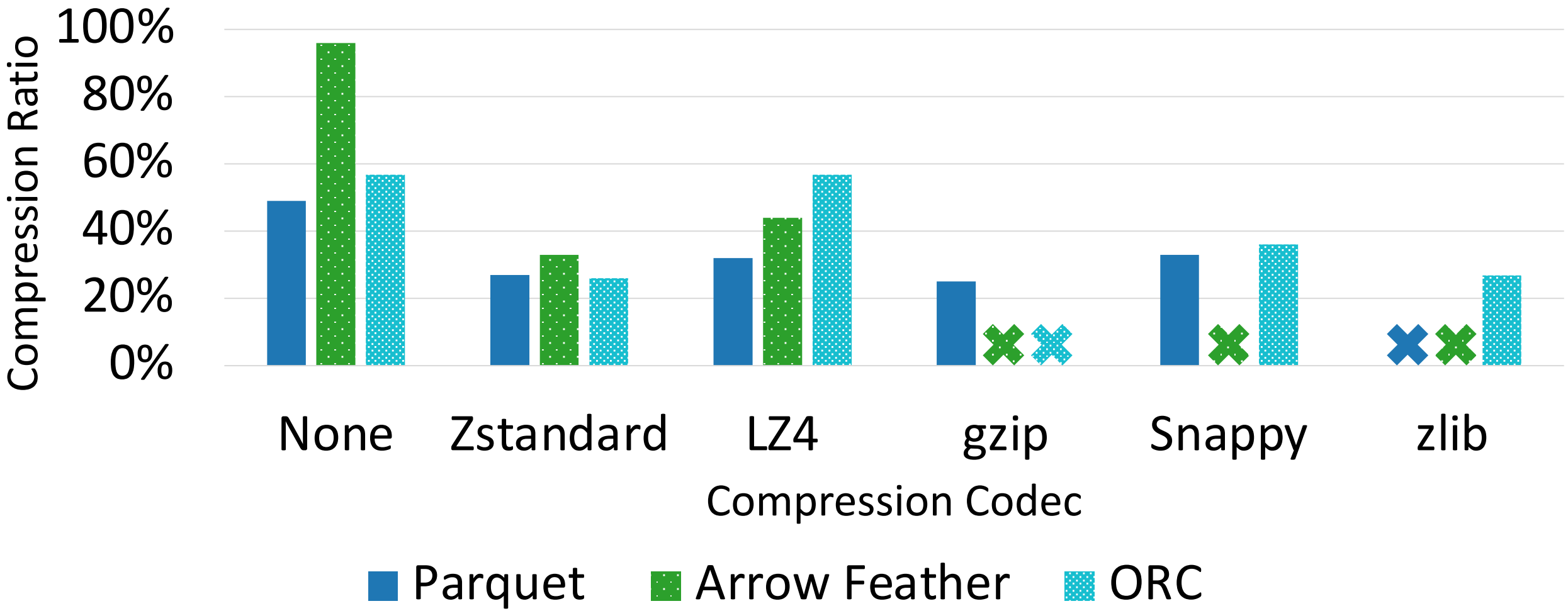


Fig A. Column compression ratio CDFs over the CodecDB, BI and JOB datasets.

Compression Ratio over TPC-DS Dataset



Evaluation outline

- Compression ratio
  - ~31K data columns extracted from real-world datasets, TPC-DS dataset
- **Transcoding throughput**
  - **Compression and decompression overhead (arrow in-memory table ⇔ format x)**
- Data access
  - Micro-benchmarking over projection, predicate and bitmap evaluation
- End-to-end evaluation over subexpressions
  - End-to-end query performance over query subexpressions drawn from TPC-DS
- Advanced features
  - Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization

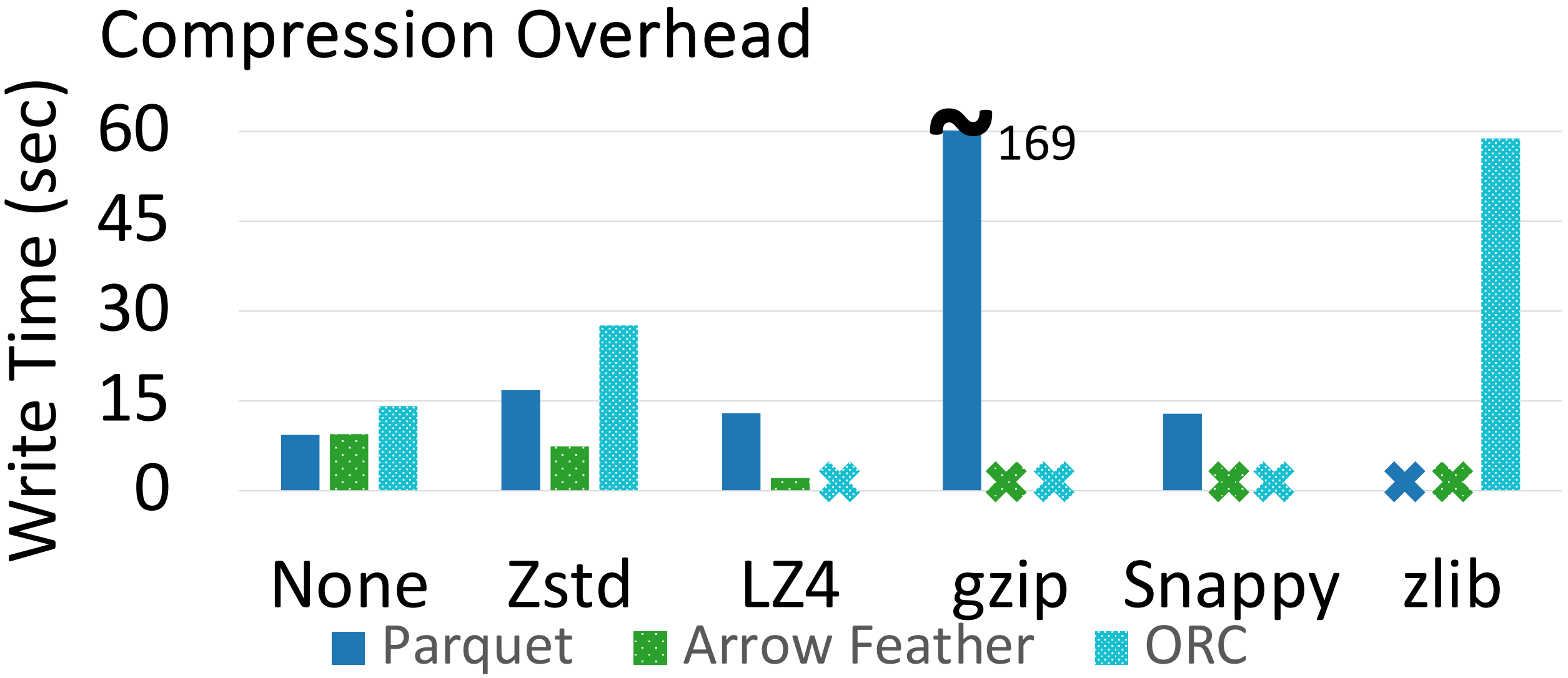


Figure A. Write time from an Arrow in-memory table to each format stored on disk

Decompression Overhead

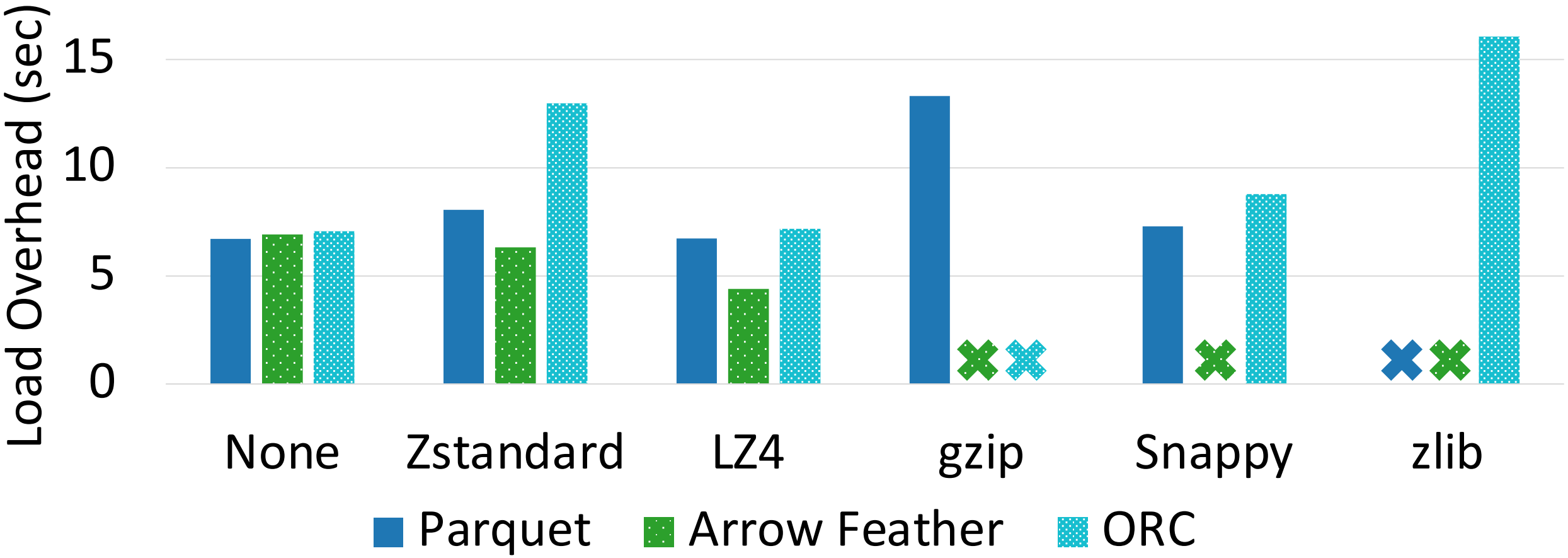


Figure A. Overheads (in seconds) for decompressing the TPC-DS catalog_sales table from the formats on disk into in-memory Arrow

Takeaways

Encoding and compression choices greatly impact performance, with formats like Parquet and ORC targeting size on disk, while Arrow targets raw read performance in memory.

To optimize both size and performance, formats should be carefully tuned to the workload and use case, and workload-aware compression selection is crucial.

It remains an open question of how much computation can be pushed into the encoded space to minimize the decoding step while maximizing the compression ratio.

Evaluation outline

- Compression ratio
  - ~31K data columns extracted from real-world datasets, TPC-DS dataset
- Transcoding throughput
  - Compression and decompression overhead
- **Data access**
  - **Micro-benchmarking over projection, predicate and bitmap evaluation**
- End-to-end evaluation over subexpressions
  - End-to-end query performance over query subexpressions drawn from TPC-DS
- Advanced features
  - Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization

Projection - Integer

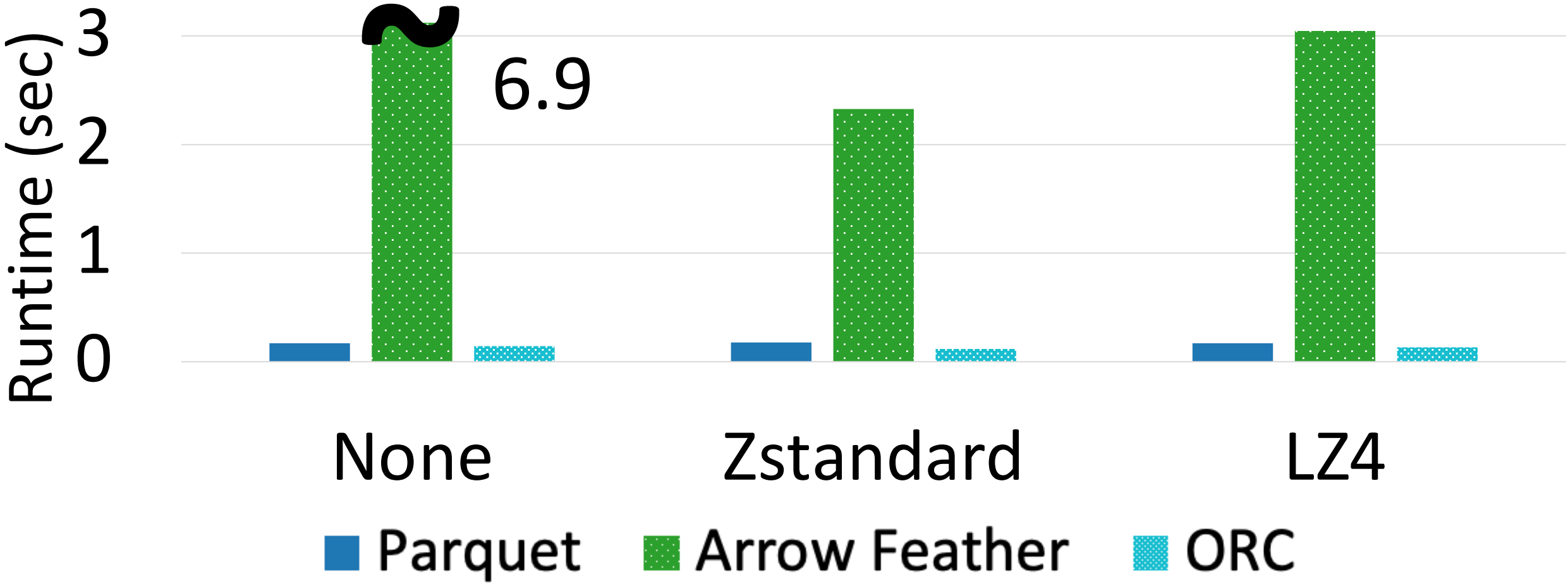


Figure A. Projecting Integer type on the catalog_sales table

Projection - Double

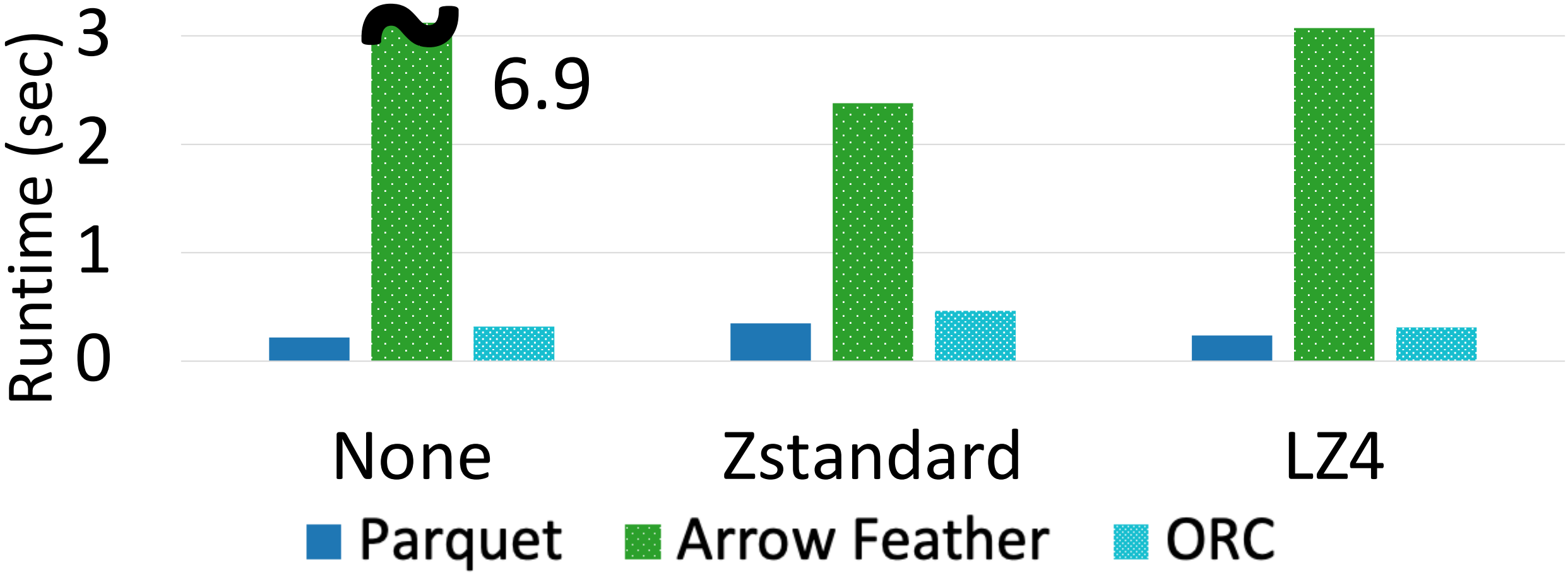


Figure A. Projecting Double type on the catalog_sales table

Profiling on The Loading Time

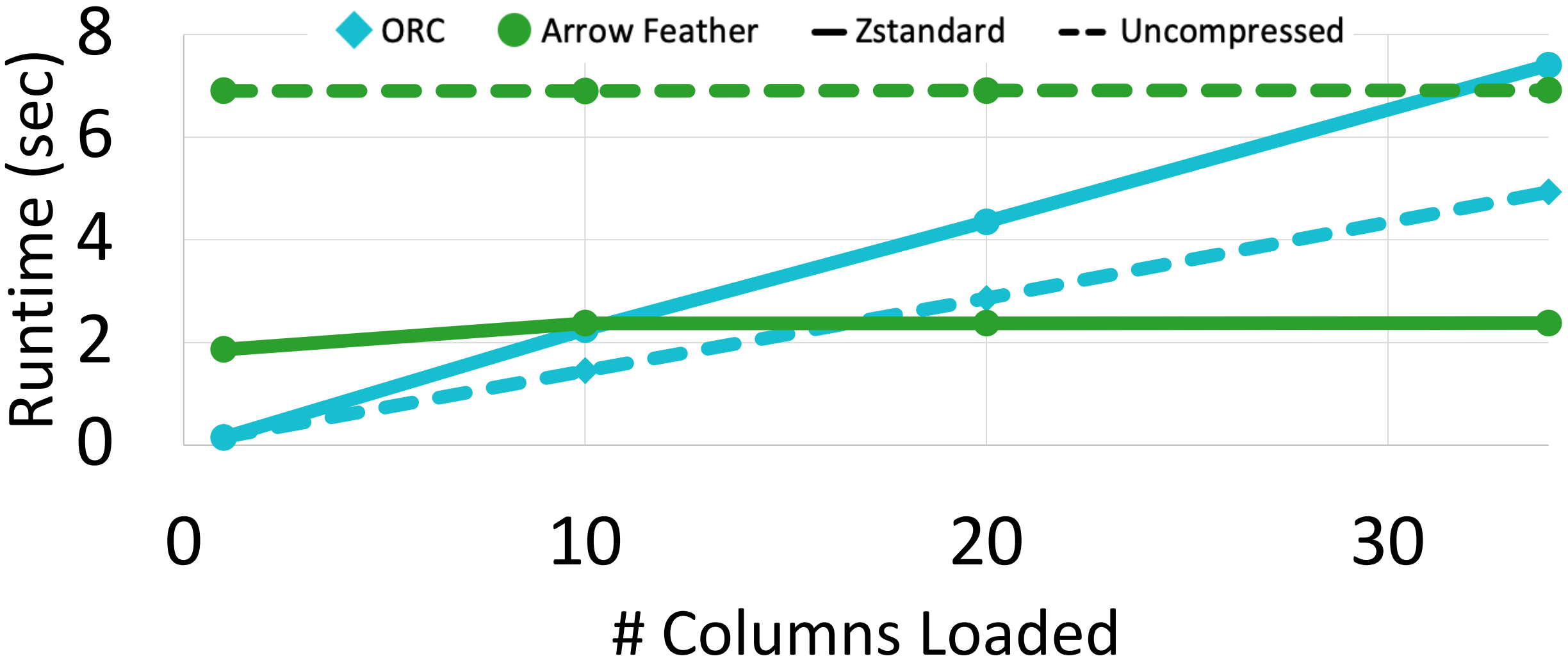


Figure A. Profiling single column to full table sequential loading for ORC and Apache Arrow from the catalog_sales table

Arrow Parallel Loading

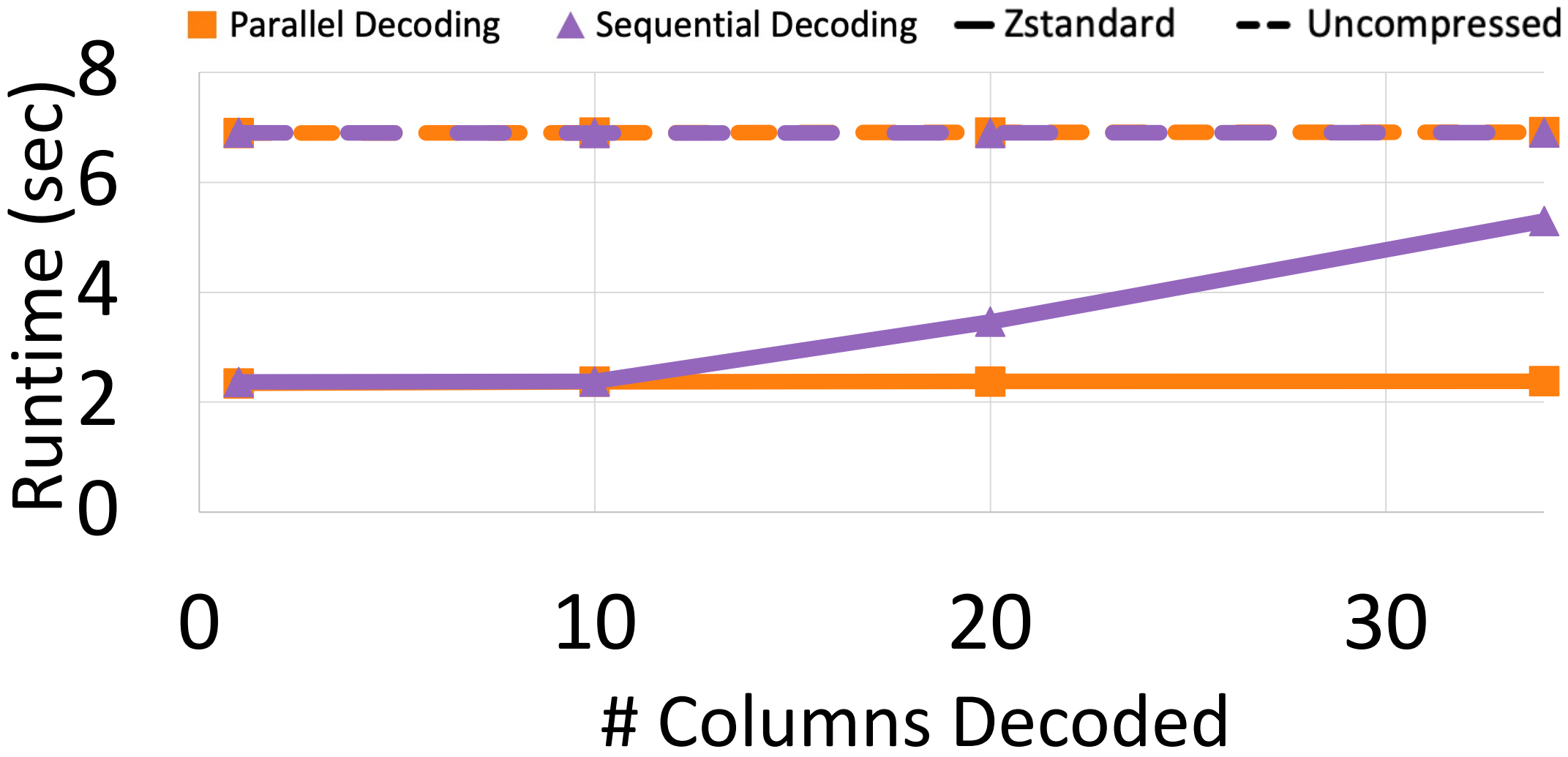


Figure A. Arrow serial vs parallel

Bit-vector performance by selectivity

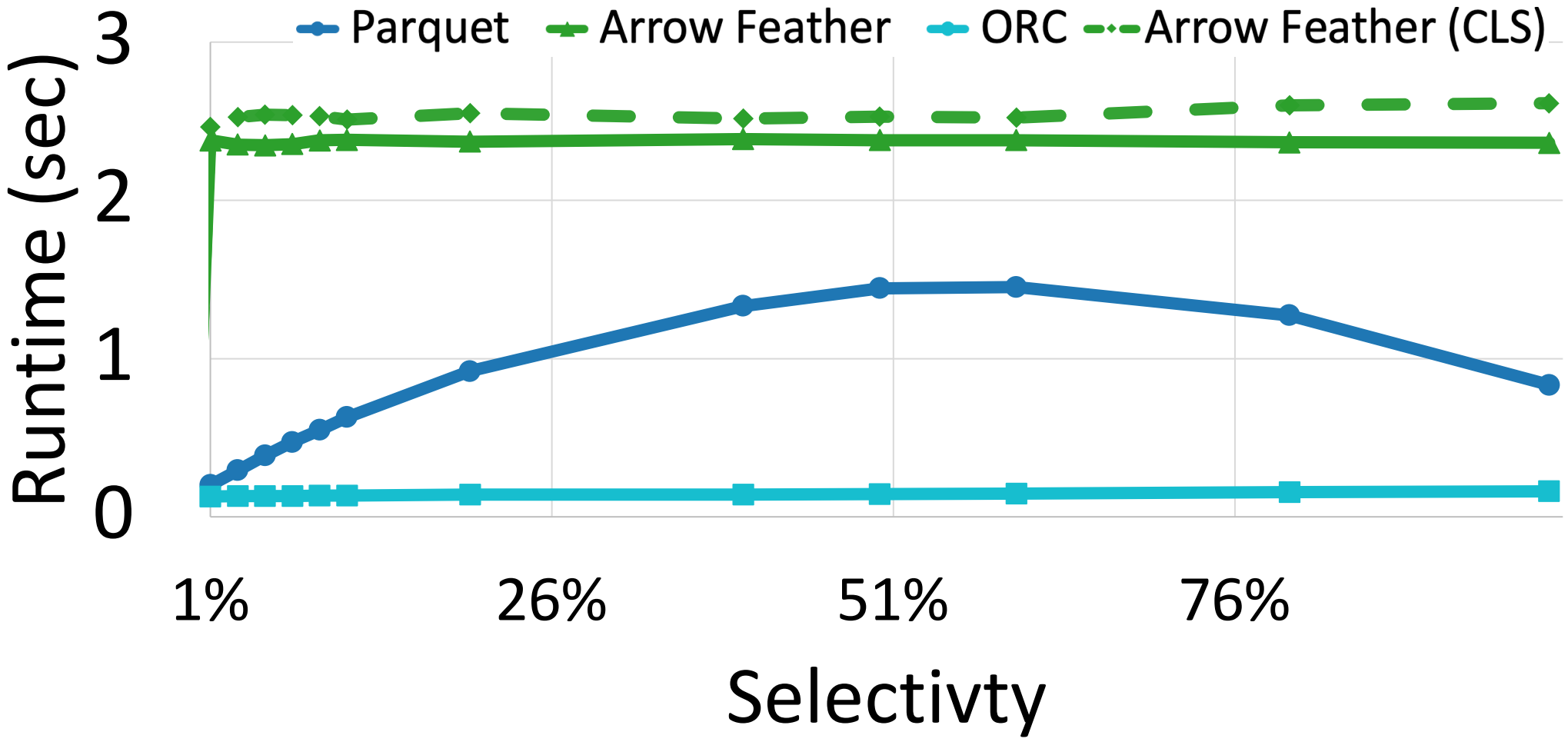
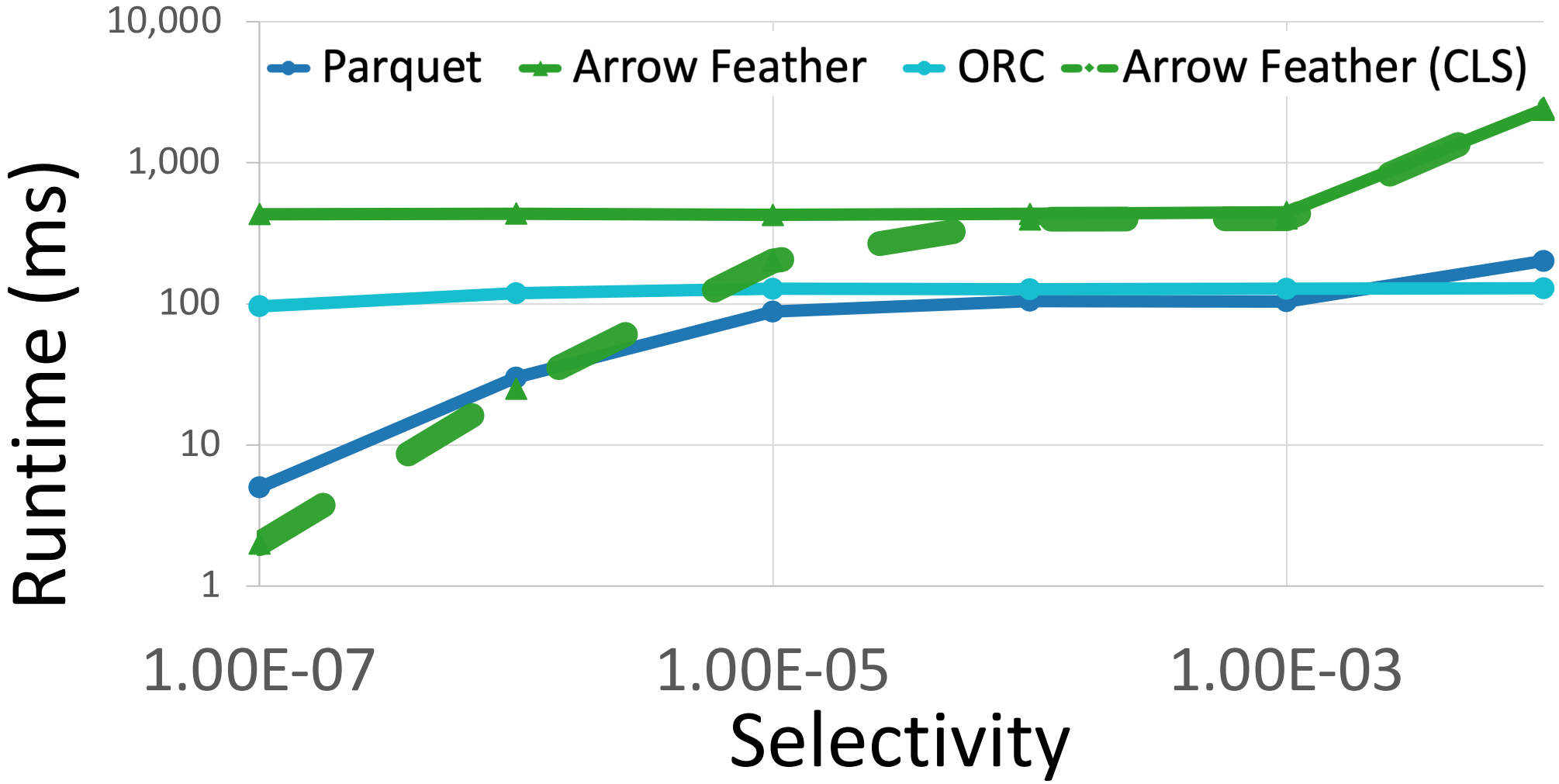


Figure A. Performance by selectivity

High selectivity (Log-log scale)



Takeaways

Trade-offs exist for simple data access operations, with no format being the best in every case.

Data skipping is important, but it does not always help. Workload-aware data partitioning can be used to balance these factors.

Record-level data access APIs provide flexibility for data skipping, but has reduced performance on queries with high selectivity compared to bulk loading APIs.

To improve performance, on-disk formats should be more adaptive and co-designed with an in-memory representation.

Evaluation outline

- Compression ratio
  - ~31K data columns extracted from real-world datasets, TPC-DS dataset
- Transcoding throughput
  - Compression and decompression overhead
- Data access
  - Micro-benchmarking over projection, predicate and bitmap evaluation
- **End-to-end evaluation over subexpressions**
  - **End-to-end query performance over query subexpressions drawn from TPC-DS**
- Advanced features
  - Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization

End-to-End Evaluation over Subexpressions

Table A. Evaluated TPC-DS SP query subexpressions

Q1	SELECT cs_ship_date_sk, cs_bill_customer_sk FROM catalog_sales WHERE cs_sold_time_sk=12032 AND cs_sold_date_sk=2452653
Q2	SELECT cd_demo_sk, cd_dep_college_count FROM customer_demographics WHERE cd_gender='F' AND cd_education_status = 'Secondary'
Q3	SELECT cd_demo_sk FROM customer_demographics WHERE cd_gender = 'M' AND cd_marital_status = 'D' AND cd_education_status = 'College'
Q4	SELECT cs_ext_sales_price, cs_sold_date_sk, cs_item_sk FROM catalog_sales WHERE cs_wholesale_cost>80.0 AND cs_ext_tax < 500.0
Q5	SELECT cs_ext_sales_price, cs_sold_date_sk, cs_item_sk, cs_net_paid_inc_tax, cs_net_paid_inc_ship_tax, cs_net_profit FROM catalog_sales WHERE cs_wholesale_cost > 80

Leaf Subexpression Evaluation

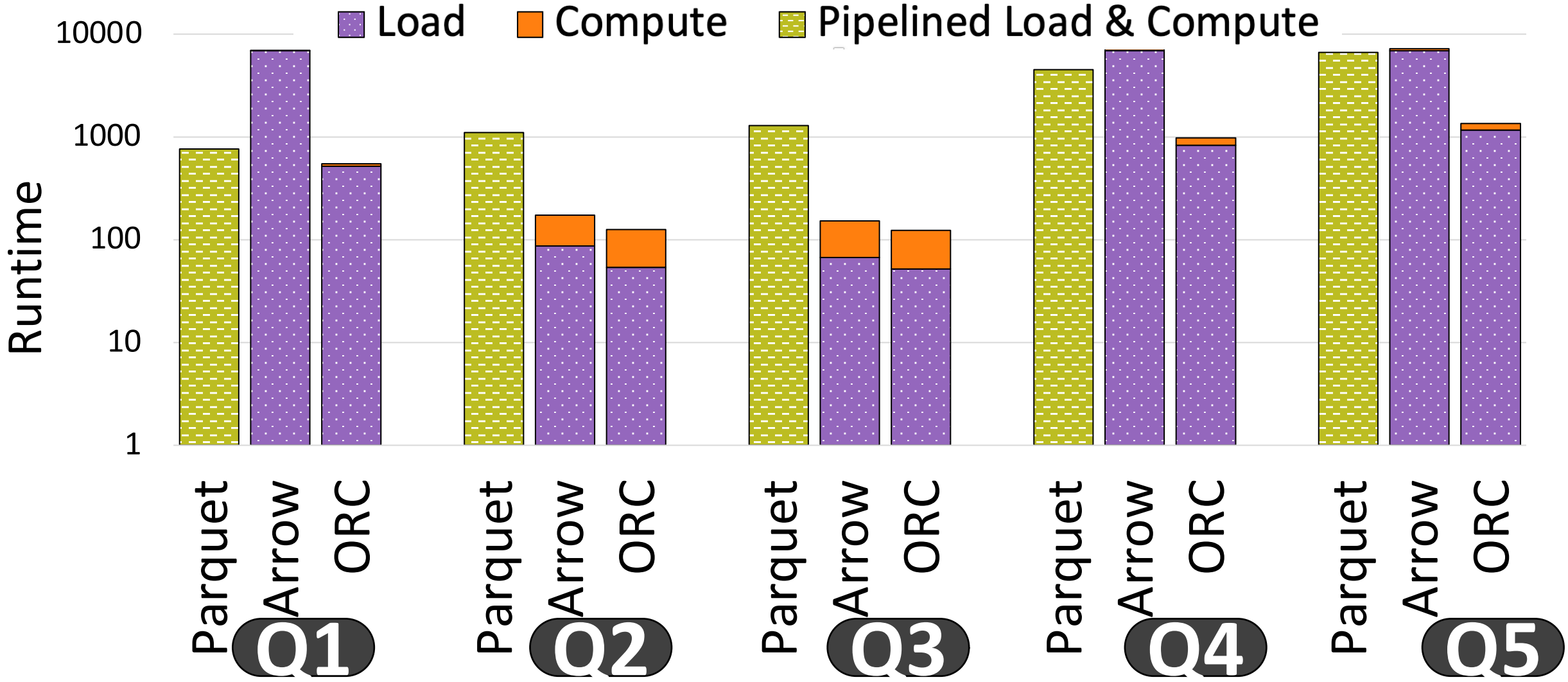


Figure A. Log-scale runtimes by format for queries with a cold cache.

Evaluation outline

- Compression ratio
  - ~31K data columns extracted from real-world datasets, TPC-DS dataset
- Transcoding throughput
  - Compression and decompression overhead
- Data access
  - Micro-benchmarking over projection, predicate and bitmap evaluation
- End-to-end evaluation over subexpressions
  - End-to-end query performance over query subexpressions drawn from TPC-DS
- **Advanced features**
  - **Apply popular optimization features e.g., lazy loading, in-situ query, vectorization, compiler optimization**

Optimizing Arrow

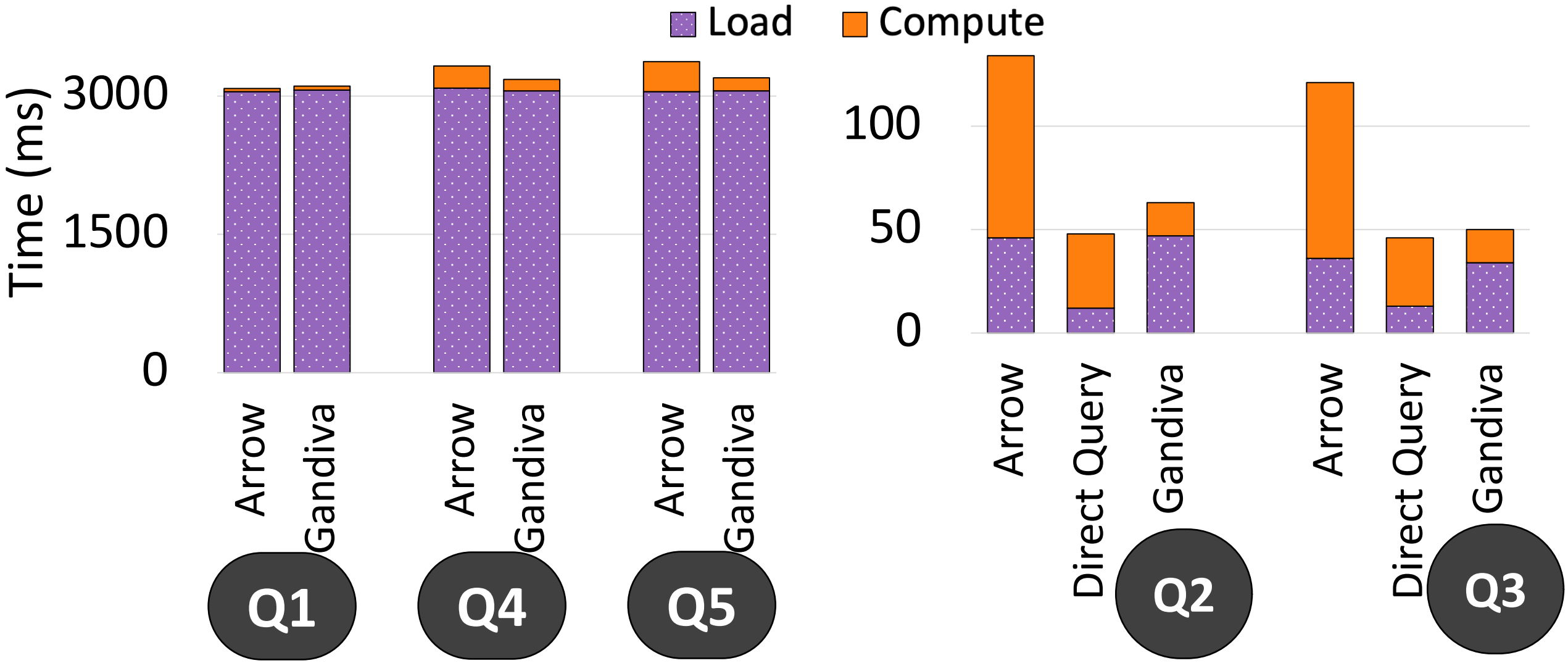


Figure A. Arrow Feather runtime with and without direct query (warm cache execution)

Optimizing Parquet

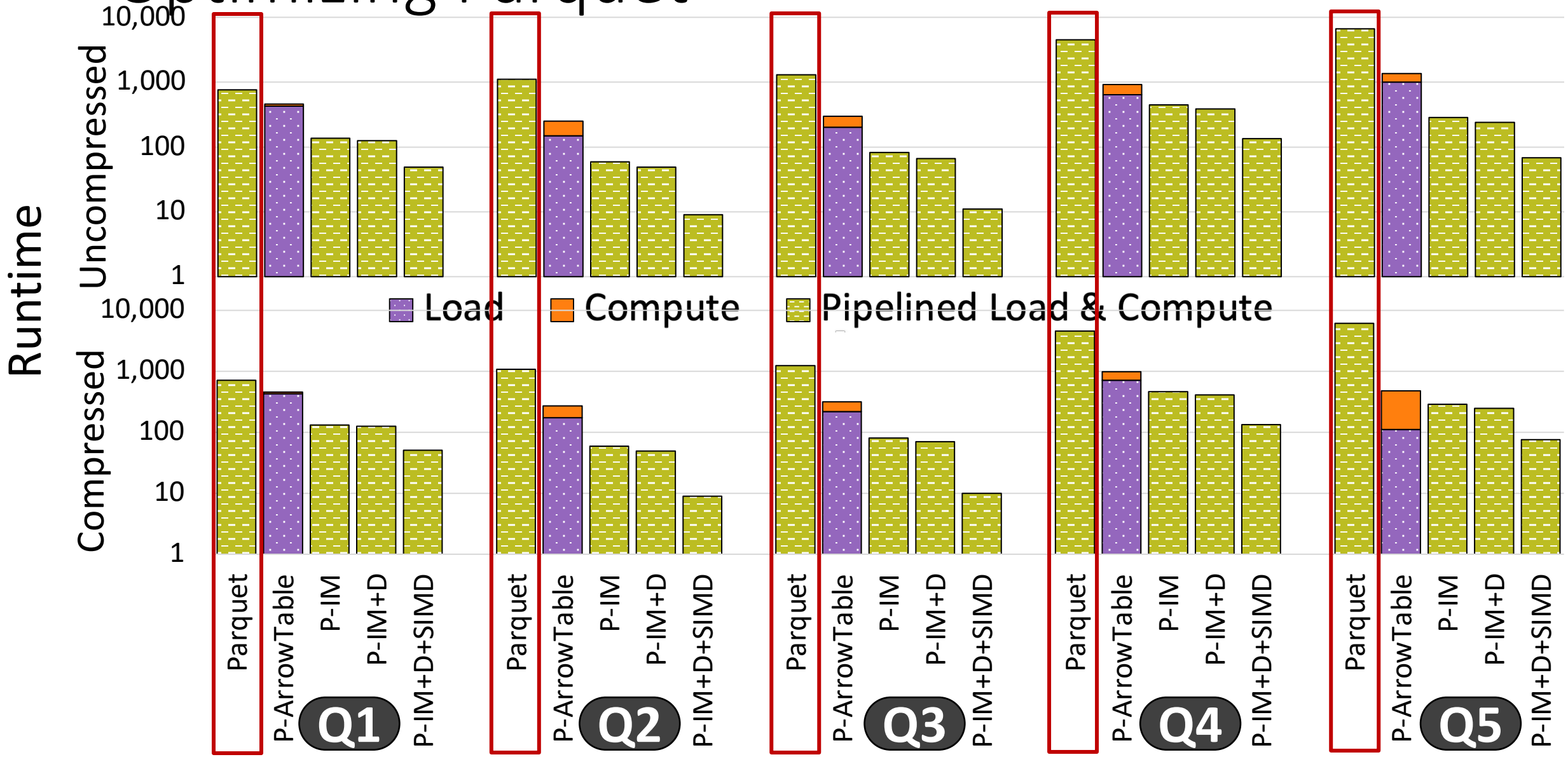


Figure A. Parquet performance with various optimizations

Optimization Takeaways

Compiler-based optimization can boost query performance in limited circumstances, but support is uneven.

Arrow computation is a good fit for computation-heavy operations, where data deserialization costs can be amortized.

Optimized Parquet is a good fit for storage side processing, when data is maintained in its original form and queries are pushed down to the compressed domain.

Thanks

Paper

Code

Contact:

Chunwei Liu

chunwei@mit.edu

Code & Paper:

aka.ms/formatdeepdive