Albis: High-Performance File Format for Big Data Systems

Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle,
Adrian Schuepbach, Bernard Metzler,
IBM Research, Zurich

2018 USENIX Annual Technical Conference

Relational Data Processing Stack in the Cloud

Relational Engines

One of the most popular data processing paradigms

- Data organized in tables
- Analyzed using DSL like SQL
- Integrity protected using variants

But unlike classical RDBMs systems, they don't manage their own storage

Relational Data Processing Stack in the Cloud

Relational Engines	Apache Spark, Cloudera Impala, Apache Drill, Hive
File Formats	JSON, Avro, Apache ORC, Parquet
Distributed Storage	Amazon S3, hadoop HDFS, riak, HBase, Neo4j, CouchDB, NoSQL, Redis, Cassandra, mongoDB

Back to the Future - It is 2010

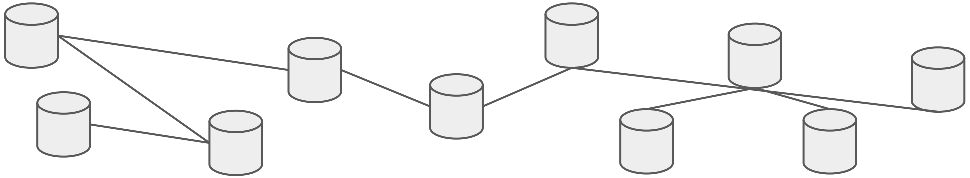
Relational Engines

Apache Spark · Cloudera Impala · Apache Drill · Hive

File Formats

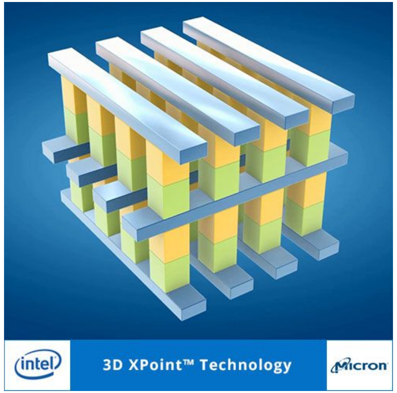
JSON · Avro · Apache ORC · Parquet · Arrow

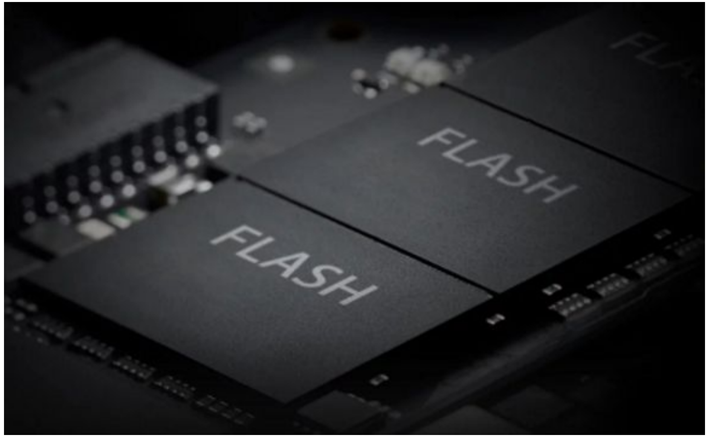
Hardware



Disks connected over 1/10 Gbps network

The I/O Revolution

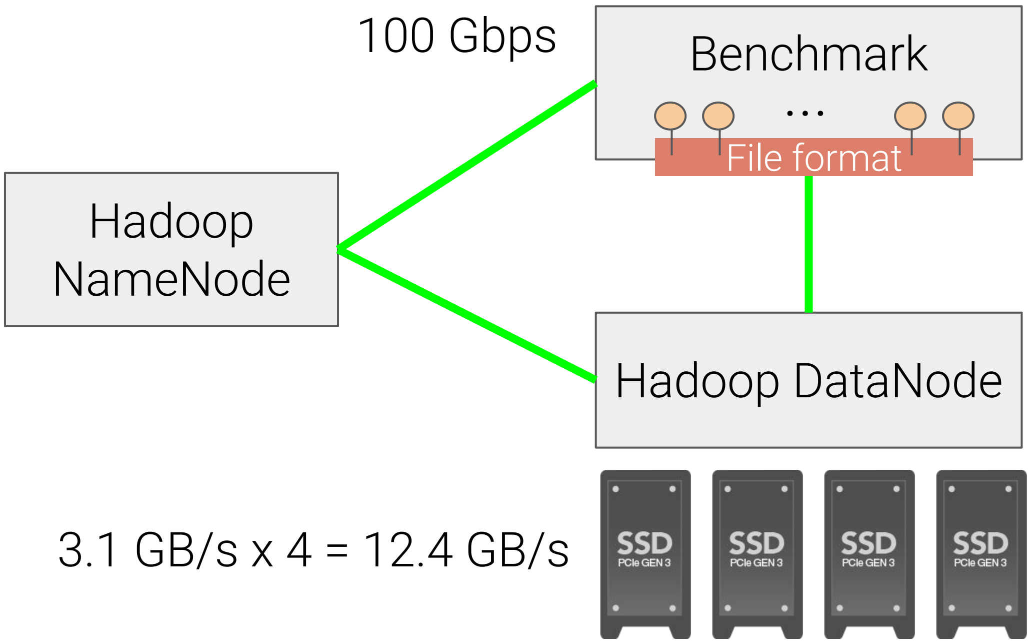




2-3 orders of magnitude performance improvements

- latency : from msecs to µsecs
- bandwidth : from MBps to GBps
- IOPS : from 100s to 100K

The Impact of the Revolution



<u>Micro-benchmark*</u>

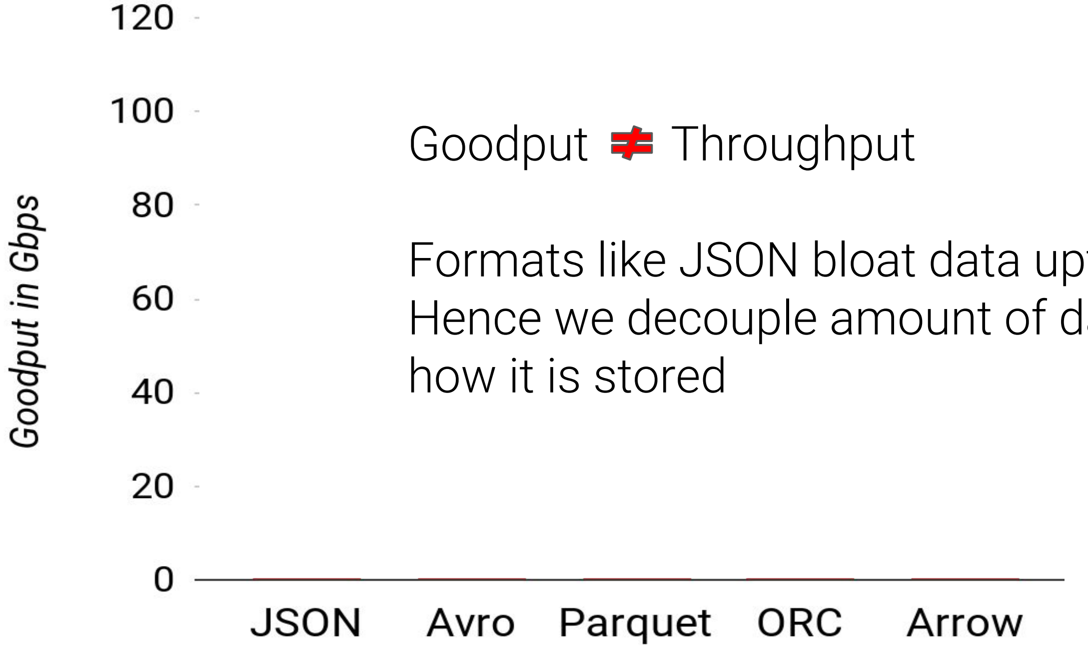
16 cores in parallel, reading TPC-DS data set.
What is the bandwidth?

Why micro-benchmark?
Decouple from the SQL engine

*https://github.com/animeshtrivedi/fileformat-benchmarks

The Impact of the Revolution

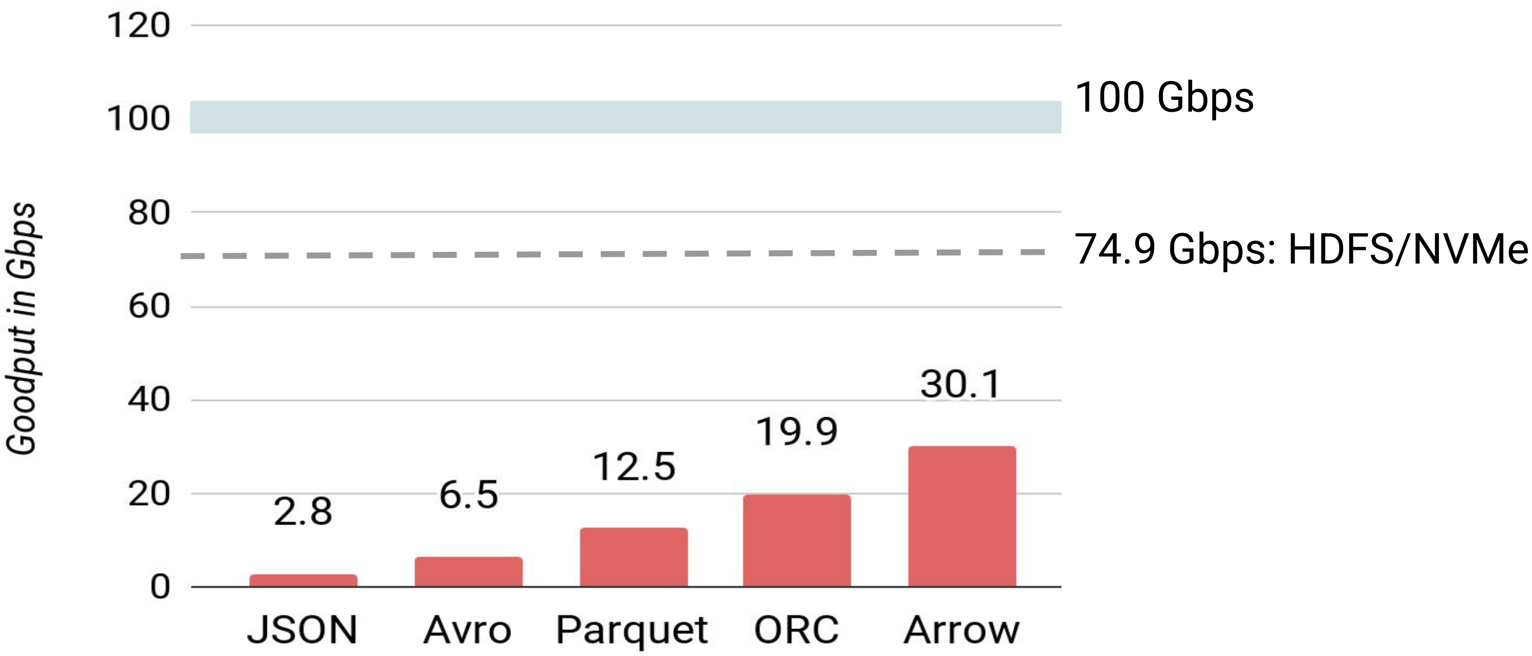
The Impact of the Revolution



Goodput ≠ Throughput

Formats like JSON bloat data upto 10x. Hence we decouple amount of data vs. how it is stored

The Impact of the Revolution



None of the modern file formats delivered performance close to the hardware

The Outdated Assumptions and Impact

End-host assumptions

Distributed systems assumptions

Language/runtimes assumptions

The Outdated Assumptions and Impact

End-host assumptions

Distributed systems assumptions

Language/runtimes assumptions

1. CPU is fast, I/O is slow

- trade CPU for I/O
- compression, encoding

But why now? CPU core speed is stalled, but ...

	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495

The Outdated Assumptions and Impact

End-host assumptions

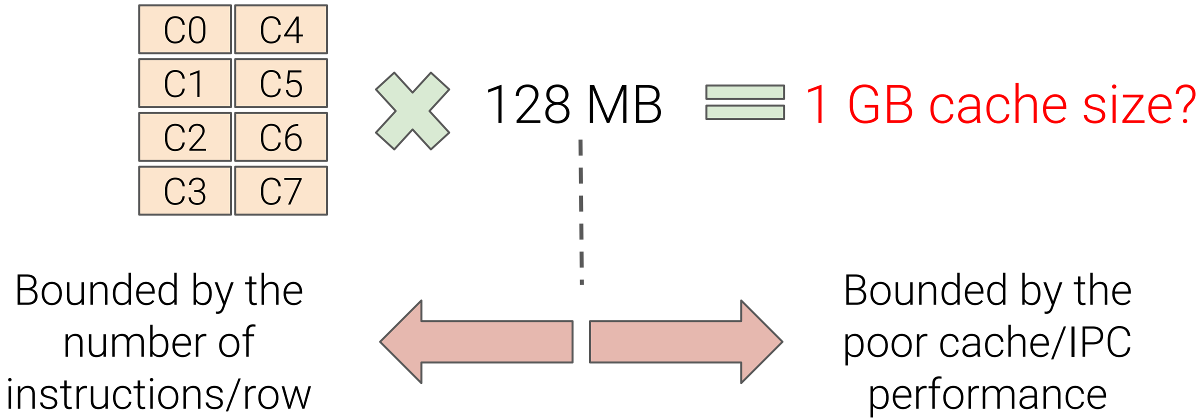
Distributed systems assumptions

Language/runtimes assumptions

2. Avoid slow, random small I/O

- preference for large block scans

But leads to bad CPU cache performance



The Outdated Assumptions and Impact

End-host assumptions

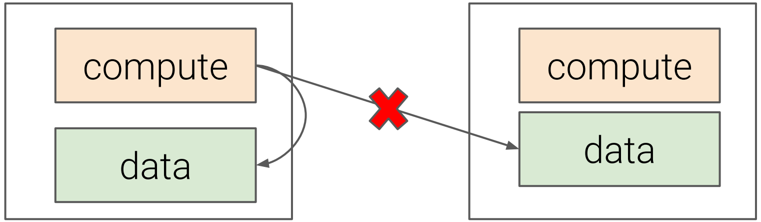
Distributed systems assumptions

Language/runtimes assumptions

3. Remote I/O is slow

- pack data/metadata together
- schedule tasks on local blocks

But now network/storage is super fast? then why still pack all data in a single block and try to co-schedule tasks?



The Outdated Assumptions and Impact

End-host assumptions

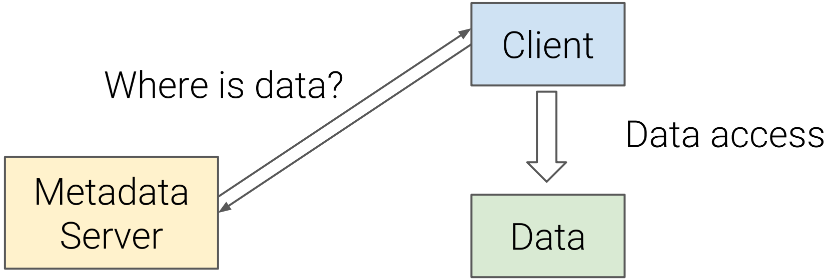
Distributed systems assumptions

Language/runtimes assumptions

4. Metadata lookups are slow

- decrease number of lookups by decreasing number of files/directories

RAMCloud, Crail can do 10 millions of lookups/sec. Does this design still make sense?



The Outdated Assumptions and Impact

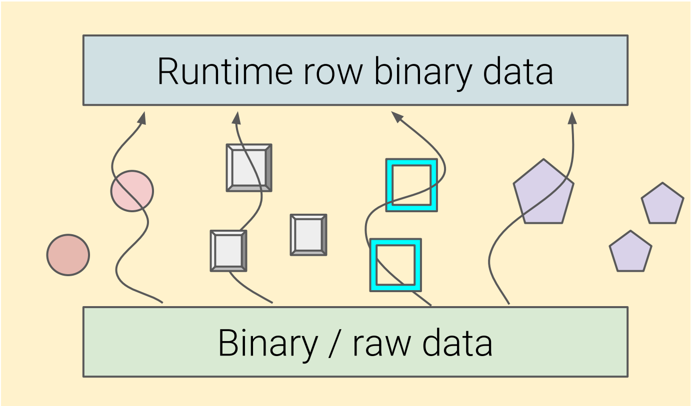
End-host assumptions

Distributed systems assumptions

Language/runtimes assumptions

5. Disregard for the runtime environment:

- group encoded/decoded
- heavy object pressure
- independent layers, no shared object
- materialize all objects



Albis

Can we reset all assumptions and start from scratch for modern high-performance I/O devices?

"Deliver the full hardware performance"

http://www.fotocommunity.de/photo/albiskette-chfleischli/39086845

Albis

- Albis - A file format to store relational tables for read-heavy analytics workloads
- Supports all basic primitive types with data and schema
  - nested schemas are flattened and data is stored in the leaves
- Three fundamental design decisions:
  1. avoid CPU pressure, i.e., no encoding, compression, etc.
  2. simple data/metadata management on the distributed storage
  3. carefully managed runtime - simple row/column storage with a binary API

Table Storage Logic

Int	double	byte[ ]	char	float[ ]
00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

Table Storage Logic

Int	double	byte[ ]	char	float[ ]
00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

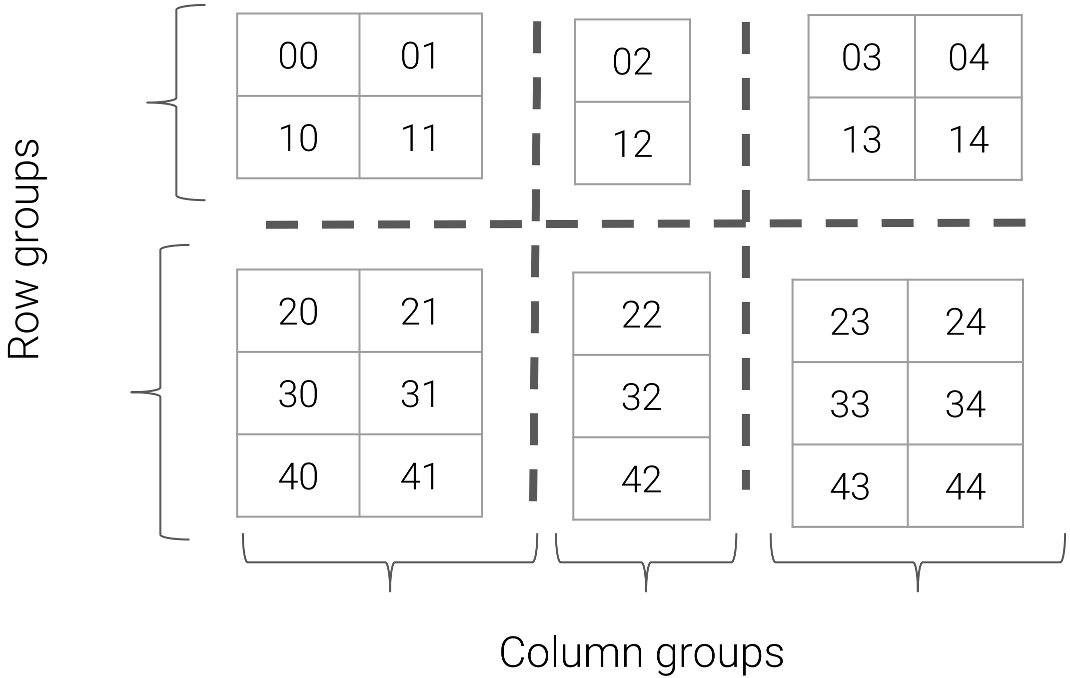


Table Storage Logic

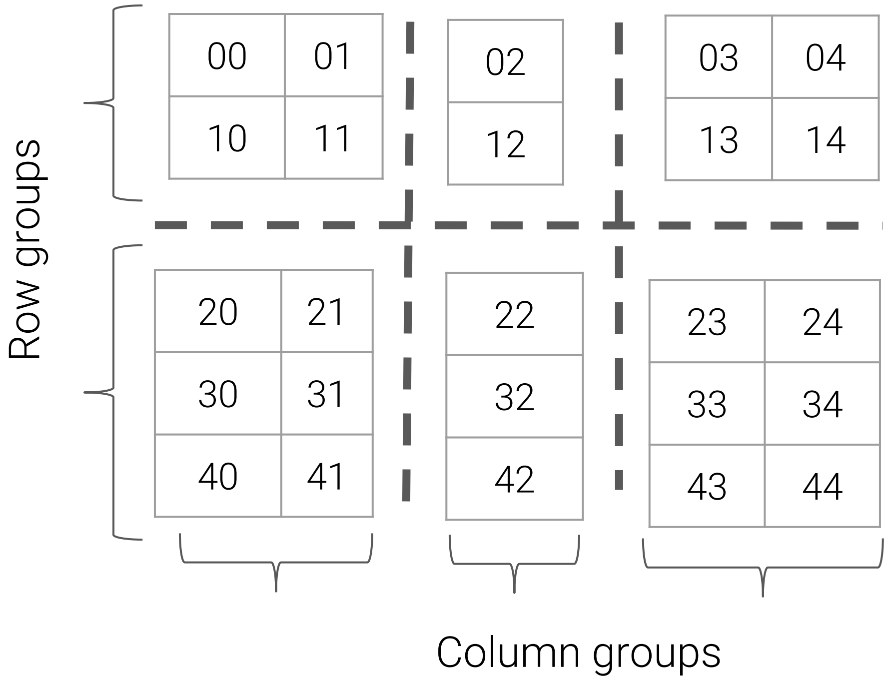
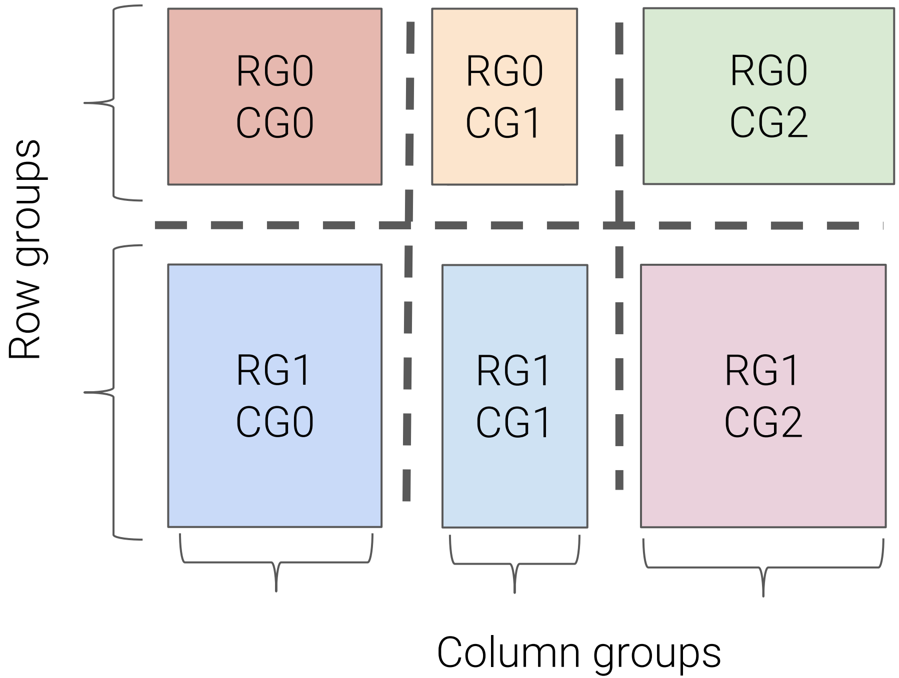
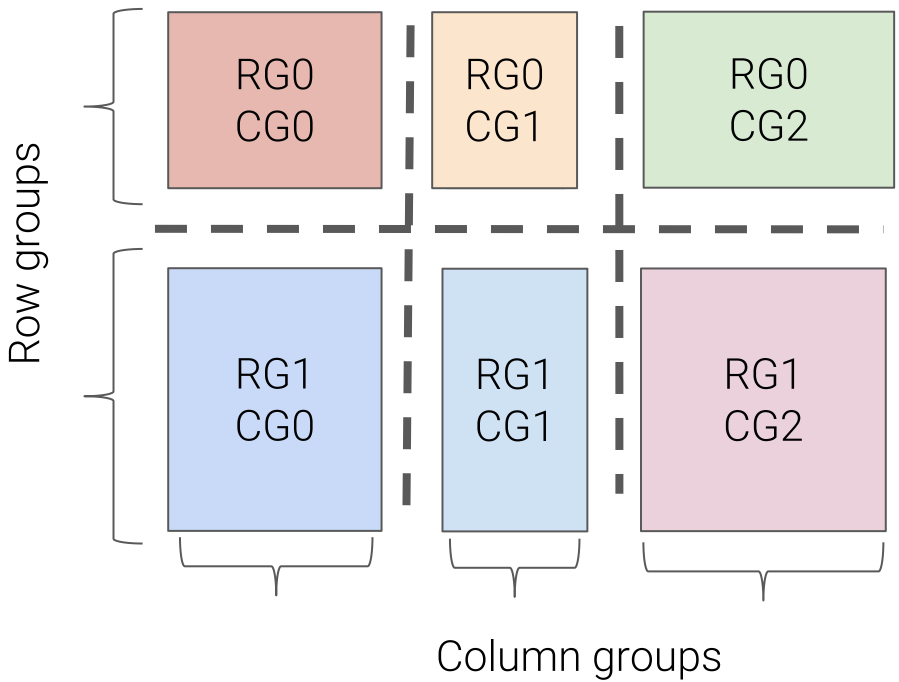


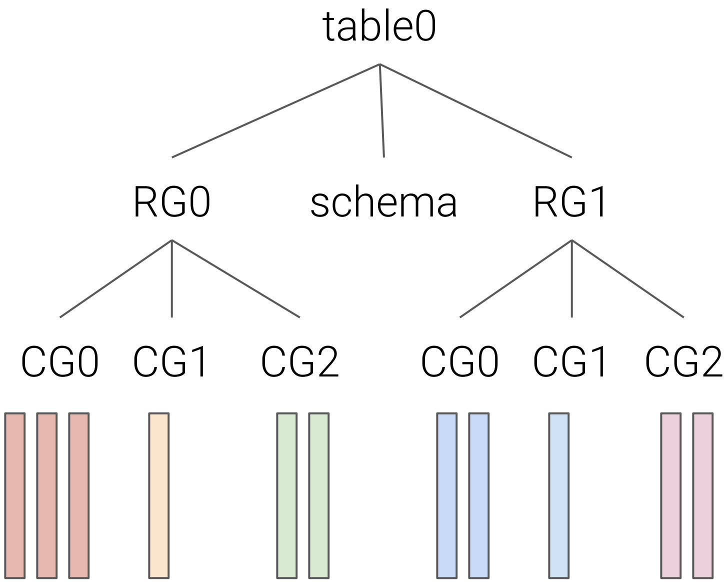
Table Storage Logic



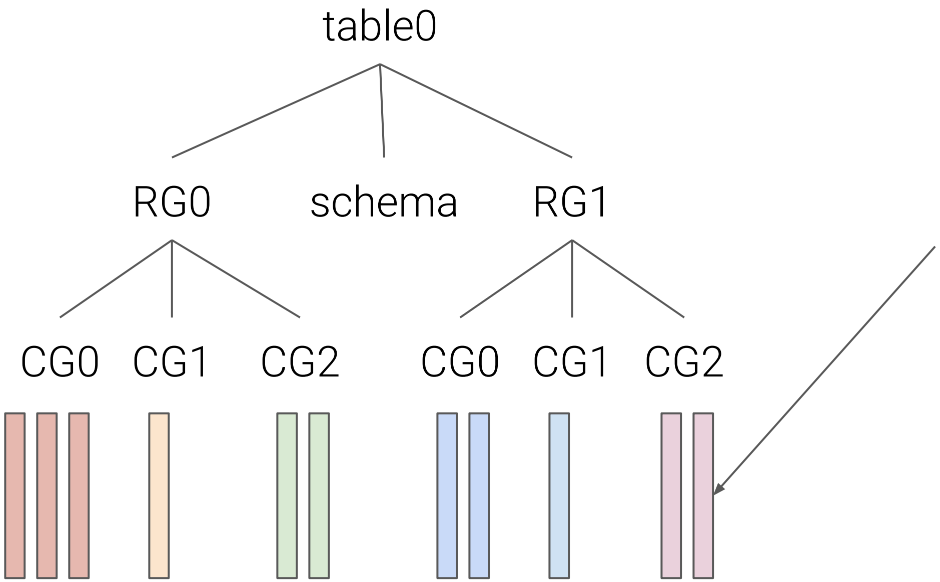
If there is only 1 column group : Row store
If there are ‘n’ column groups : Columns store

Table Storage Logic





Row Storage Format

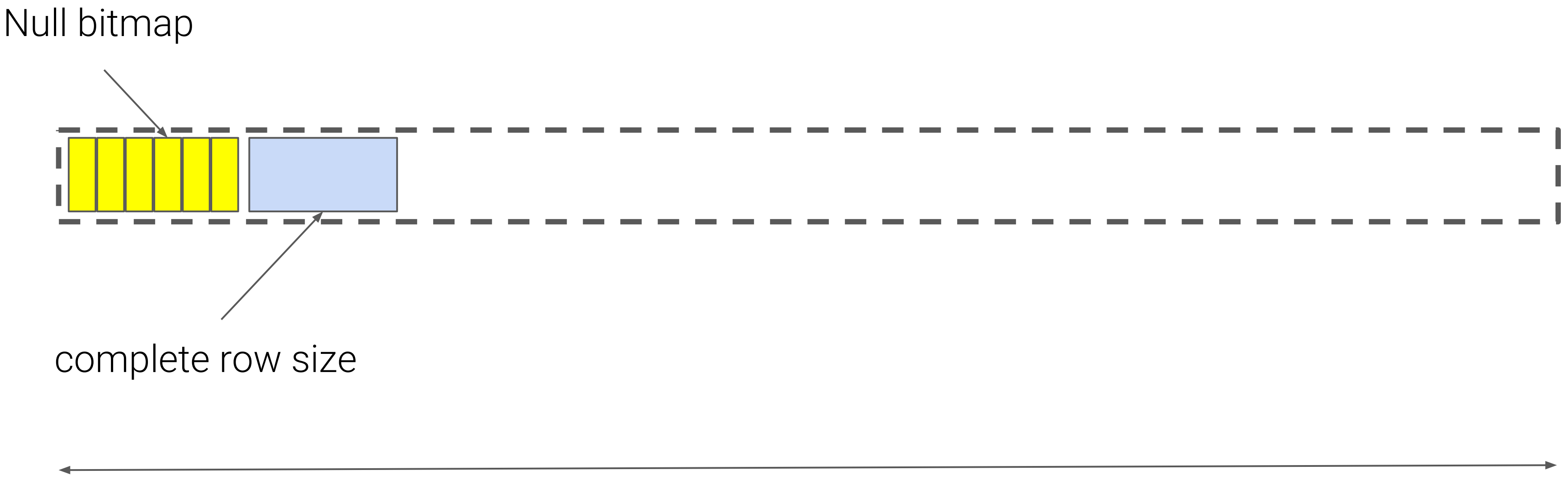


How is a single row of data stored in these files?

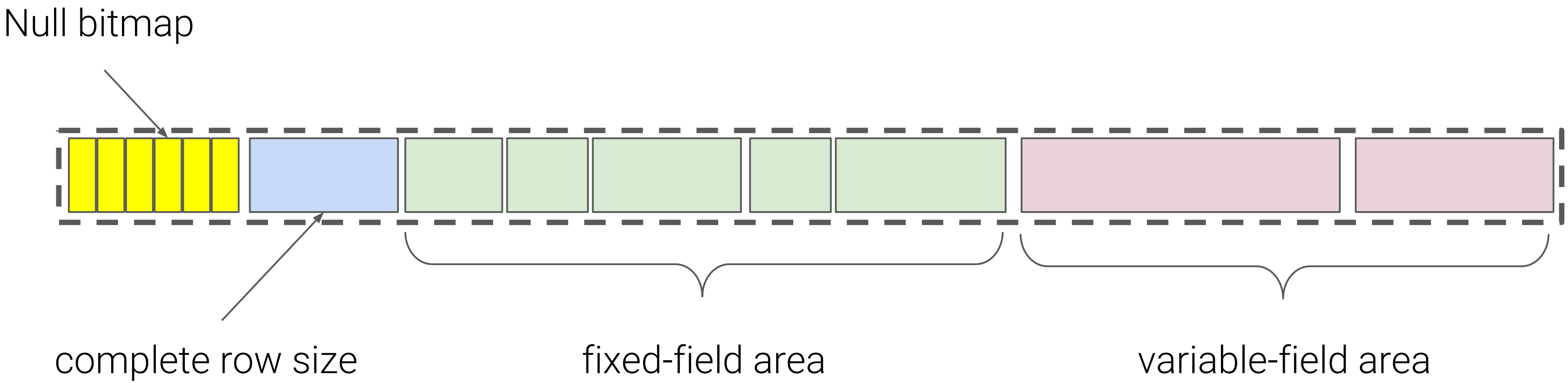
Row Storage Format

Marking null columns values

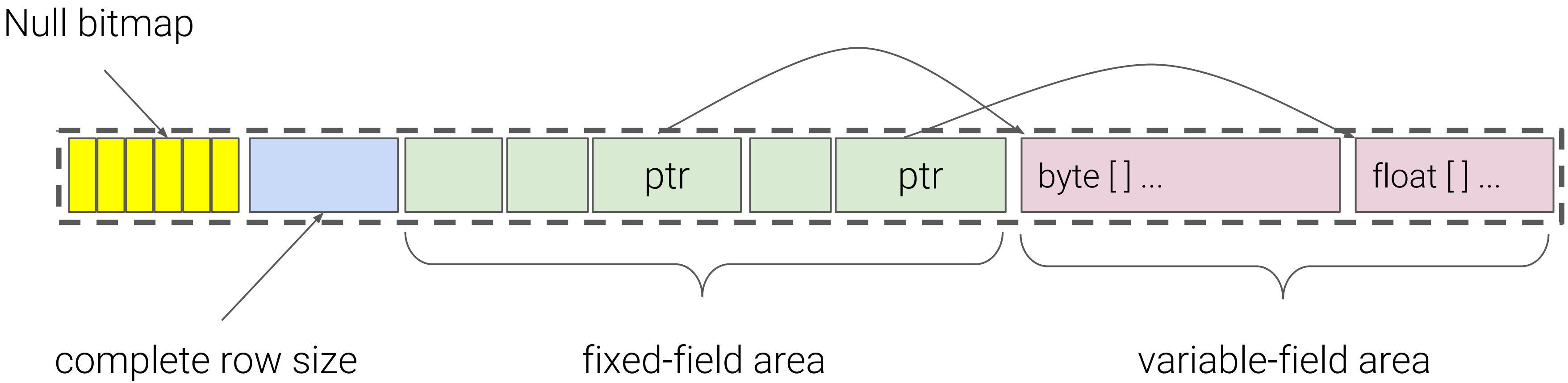
Row Storage Format



Row Storage Format

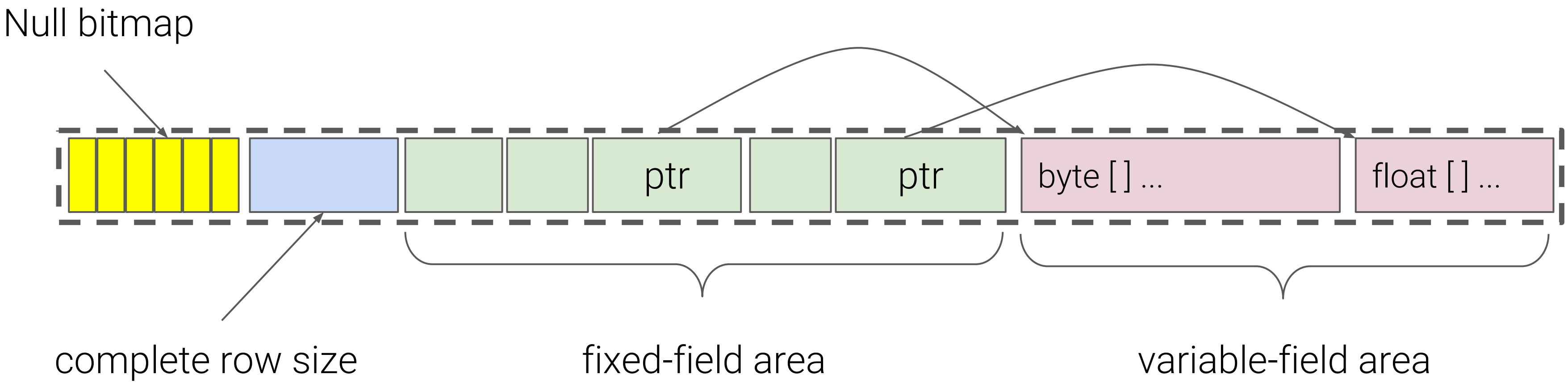


Row Storage Format



Schema of { int, double, byte[ ], char, float[ ] } :

Row Storage Format



Schema of { int, double, byte[ ], char, float[ ] } :

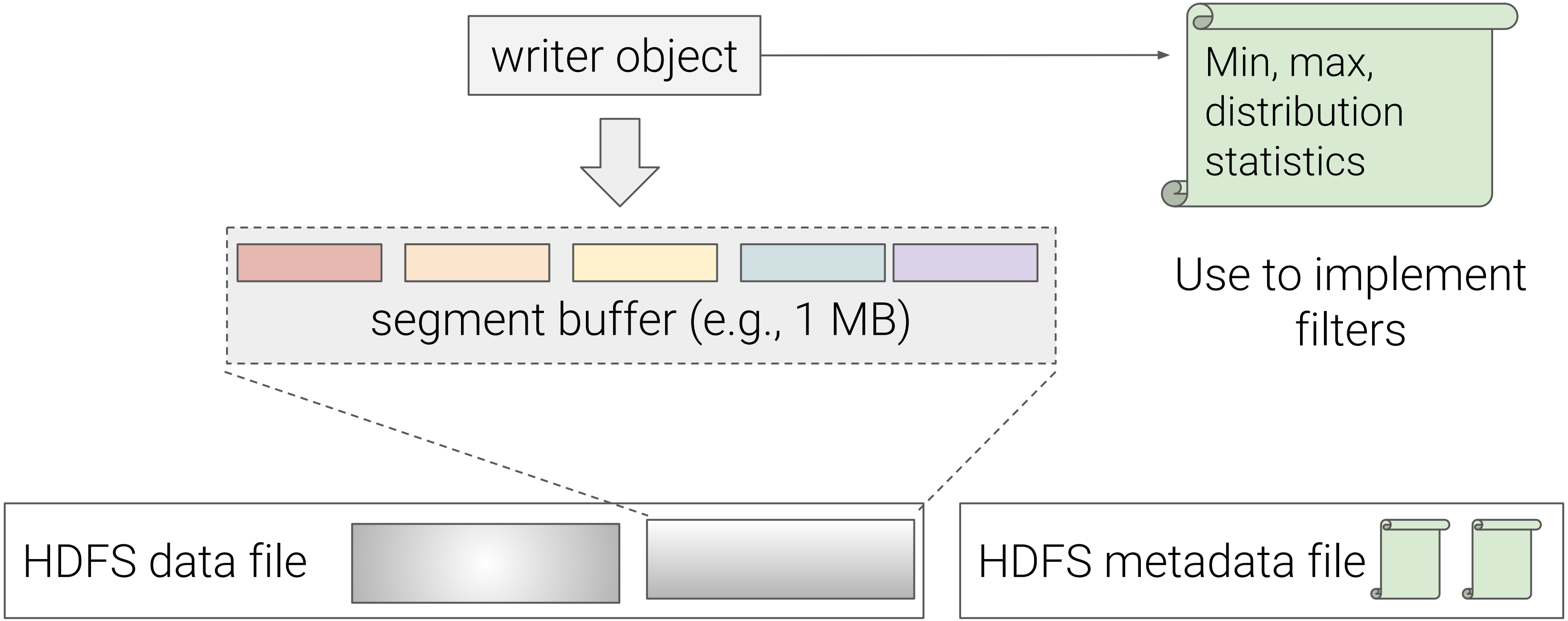
+ 1 byte bitmap (because there are 5 columns)

+ 4 byte size

+ 4 byte (int) + 8 byte (double) + 8 byte (offset + size, ptr) + 1 byte (char) + 8 byte (offset + size, ptr)

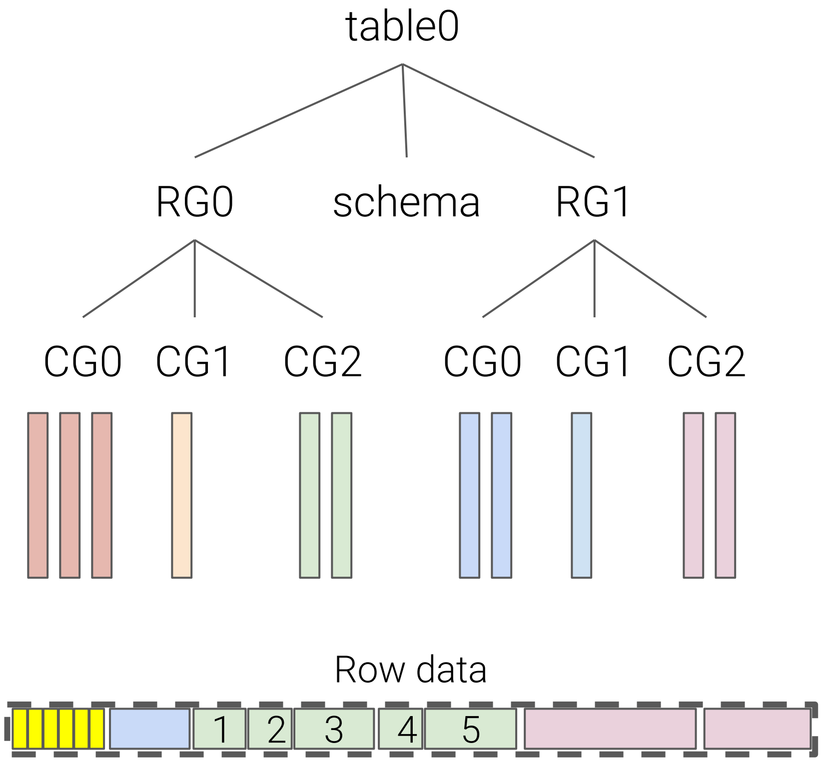
= 34 bytes + variable area.

Writing Rows



Use to implement filters

Reading Rows



1. Read schema file
2. Check projection to figure out which files to read
   a. Complete CGs
   b. Partial CGs
3. Evaluate filters to skip segments
4. Materialize values
   a. Skip value materialization in partial CG reads

More Details in the Paper

- How to evolve schema? Adding and removing columns
- How to evolve data? Adding and removing rows
- How to process Albis files in a relational data processing engine?
- Concerns regarding data imbalance or re-grouping?
- ...

Evaluation

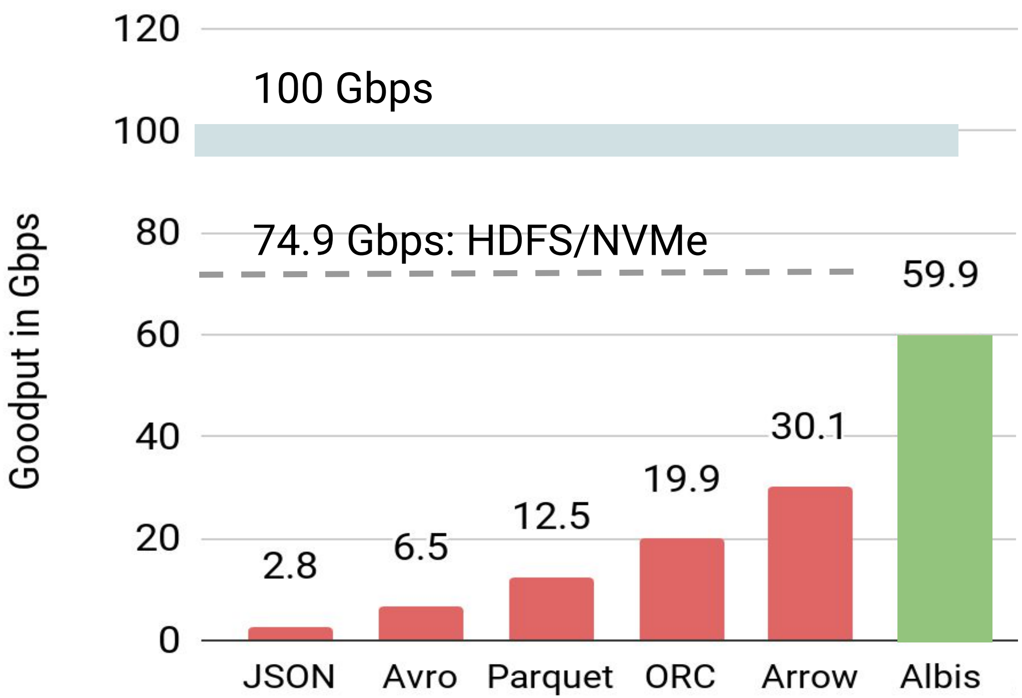
All experiments on a 4-node cluster with 100 Gbps network and flash devices

Dataset is TPC-DS tables with the scale factor of 100 (~100 GB of data)

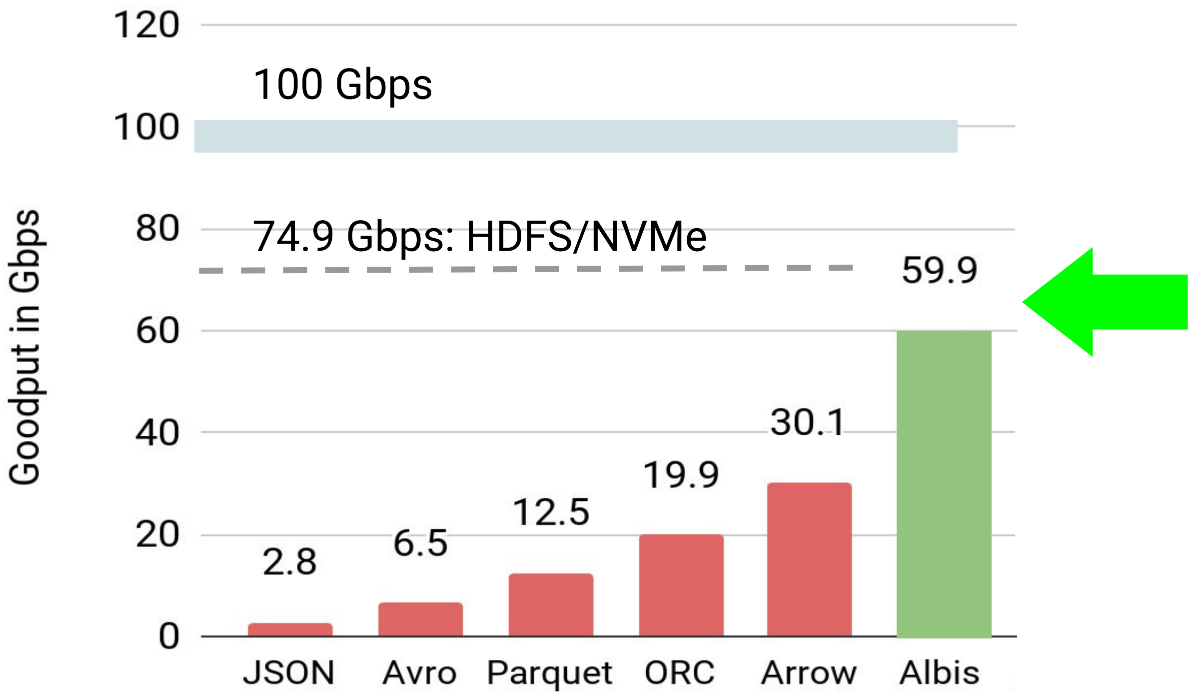
Three fundamental questions

- Does Albis deliver better performance for micro-benchmarks?
- Does micro-benchmark performance translate to better workload performance?
- What is the performance and space trade-off in Albis?

Microbenchmark Performance - Revised

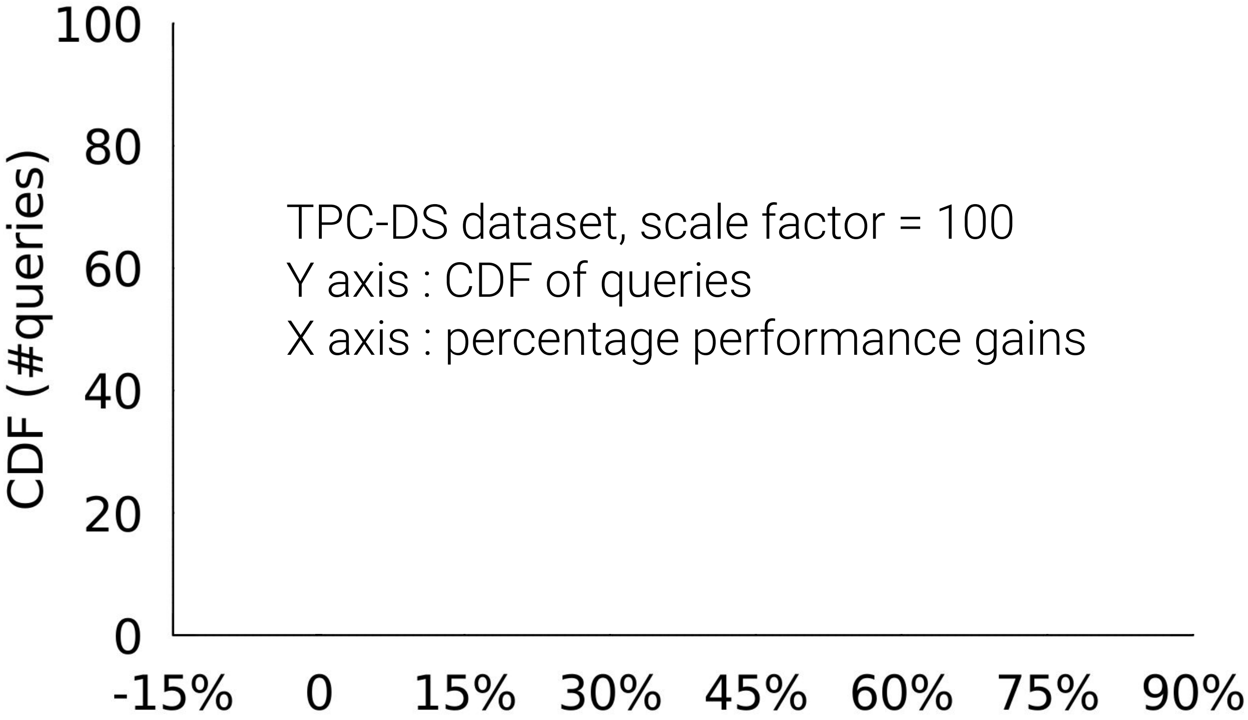


Microbenchmark Performance - Revised



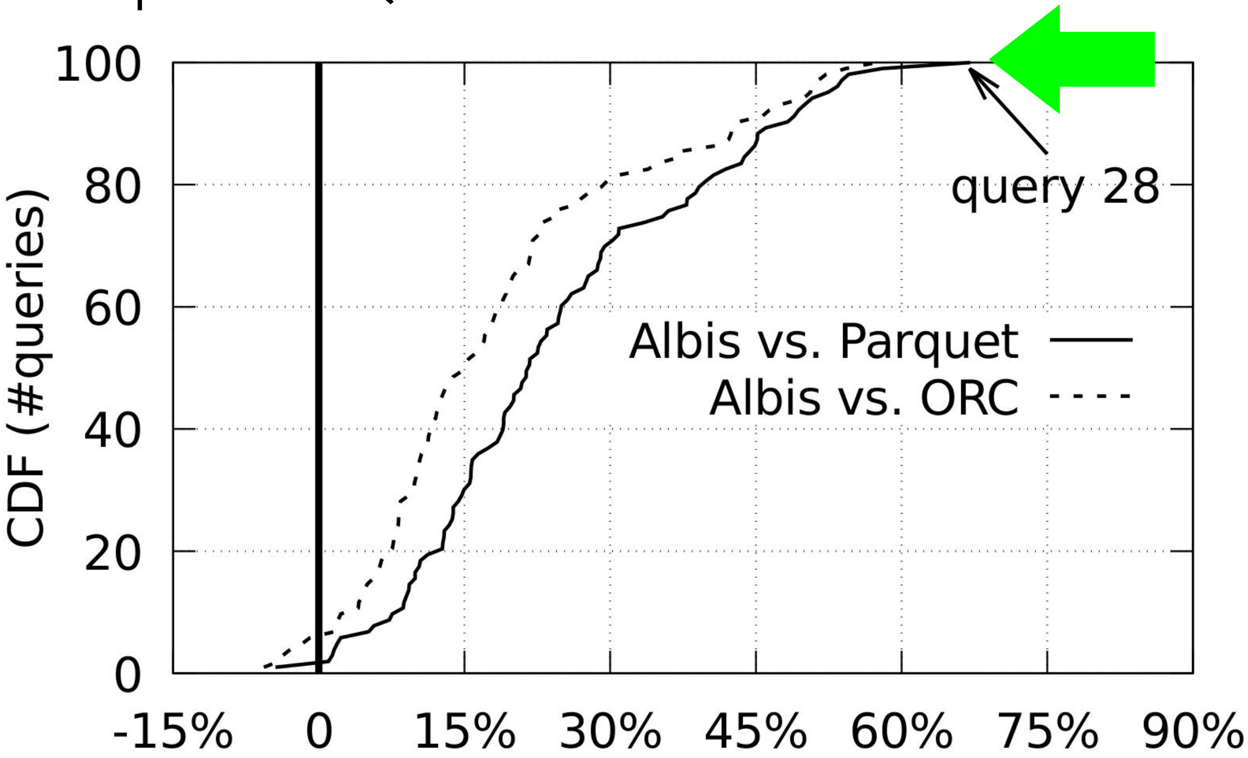
Albis delivers 1.9 - 21.3x performance improvements over other formats

Spark/SQL TPC-DS Performance



TPC-DS dataset, scale factor = 100
Y axis : CDF of queries
X axis : percentage performance gains

Spark/SQL TPC-DS Performance



Albis delivers up to 3x performance gains for TPC-DS queries

Space vs. Performance Trade-off

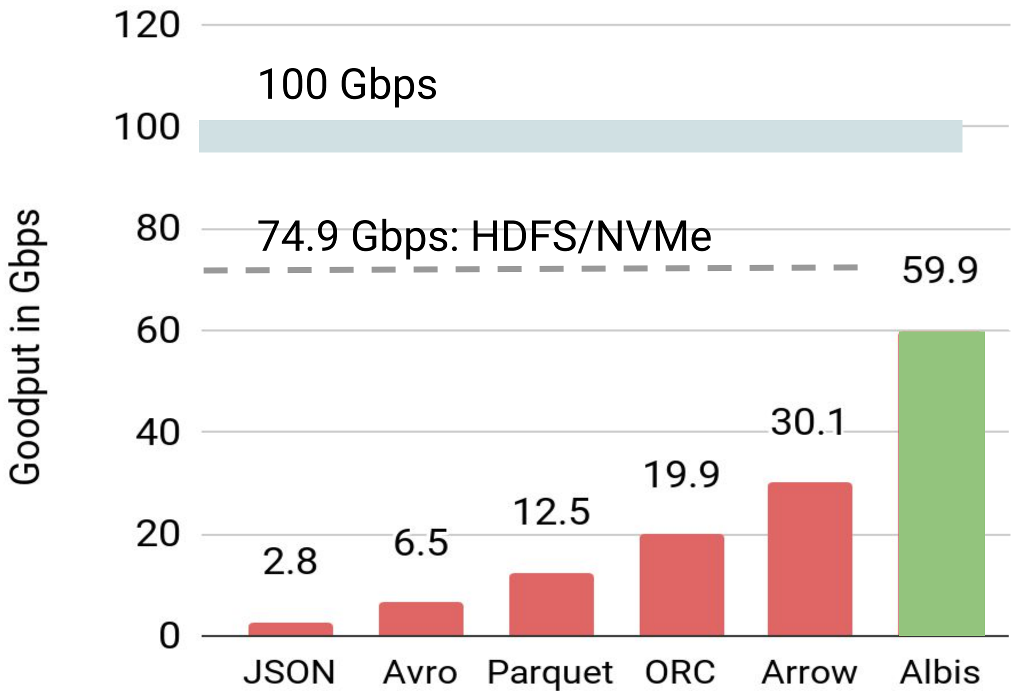
	None	Snappy	Gzip	zlib
Parquet				
ORC				
Albis				

Space vs. Performance Trade-off

	None	Snappy	Gzip	zlib
Parquet	58.6 GB 12.5 Gbps	44.3 GB 9.4 Gbps	33.8 GB 8.3 Gbps	N/A
ORC	72.0 GB 19.1 Gbps	47.6 GB 17.8 Gbps	N/A	36.8 GB 13.0 Gbps
Albis	94.5 GB 59.9 Gbps	N/A	N/A	N/A

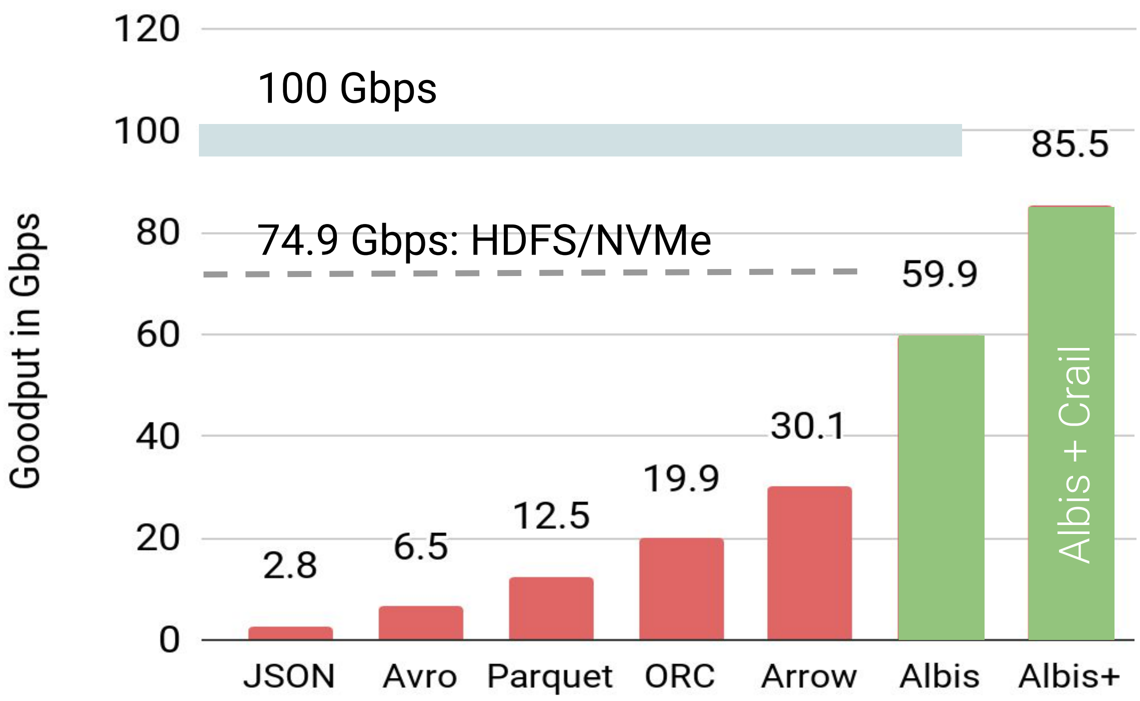
Albis inflates data by 1.3 - 2.7x, but gives 3.4 - 7.2x performance gains

Microbenchmark Performance - Revised



What would it take to deliver 100 Gbps?

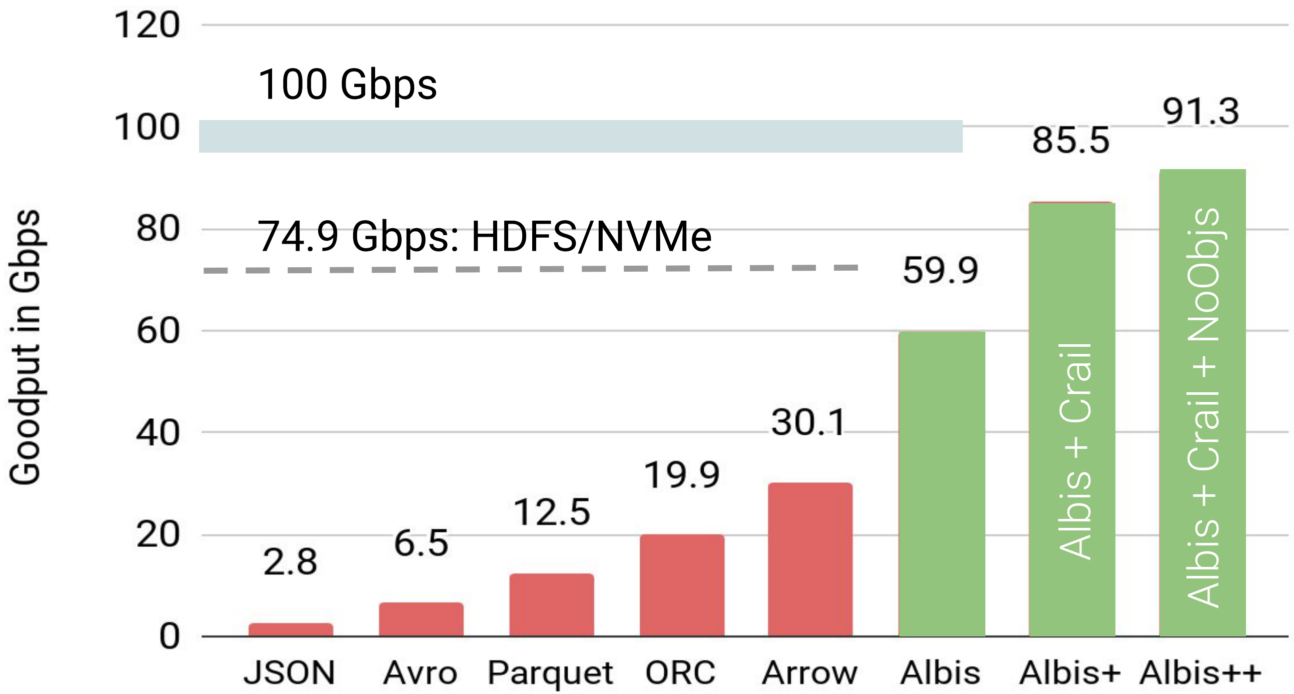
Microbenchmark Performance - Revised



JVM object overheads

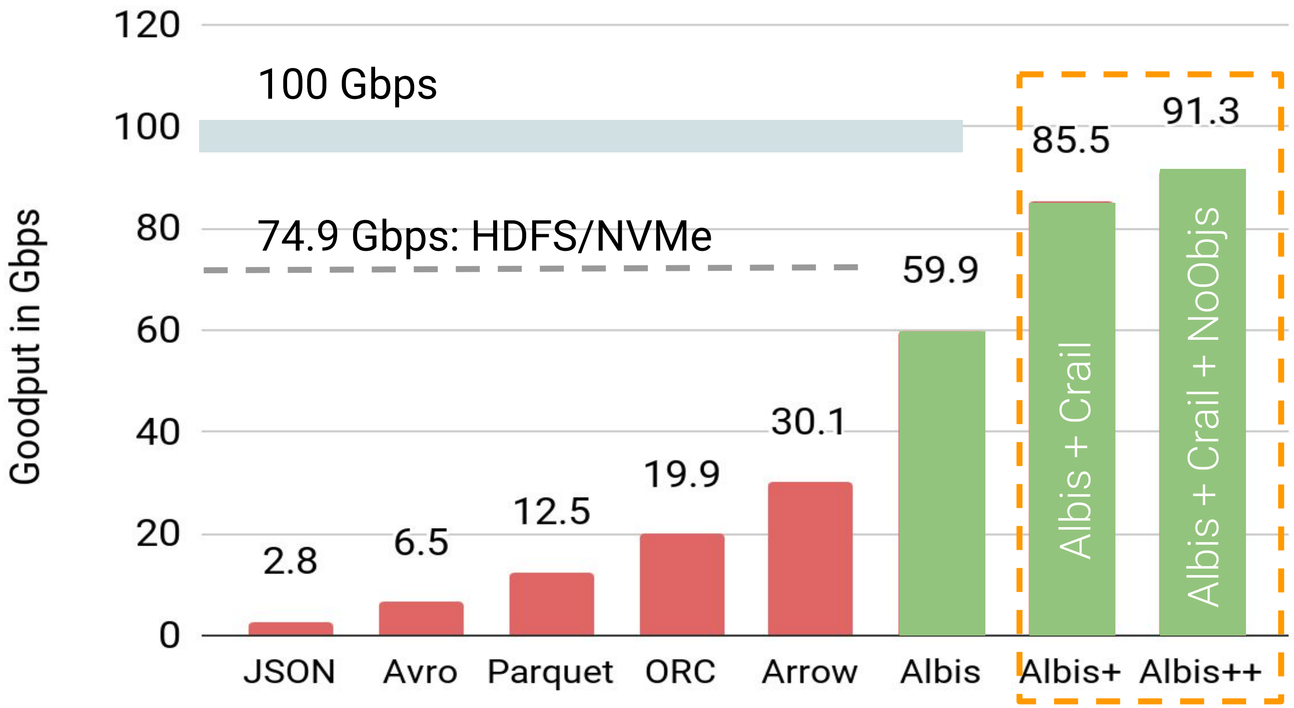
Apache Crail (Incubating) - A High-Performance Distributed Data Store, http://crail.incubator.apache.org/

Microbenchmark Performance - Revised



Data density

Microbenchmark Performance - Revised



Albis can deliver performance within 10% of hardware

Albis - Summary

- Albis - a high-performance file format for storing relational data
  - Open-source address: https://github.com/zrlio/albis
- Motivation: in presence of new network and storage devices, time to revise basic assumptions
  - no compression or encoding
  - simple data and metadata design
  - efficient object management with a binary API
- Revised software stack to lead to significant performance improvements
  - demonstrated it for the file format
  - very active research field - OSes designs (Arrakis, IX), networking and storage stacks

Notice

Backup

Microarchitectural Analysis

	Parquet	ORC	Arrow	Albis	**Gains**
Instructions per row	6.6K	4.9K	1.9K	1.6K	**1.2 - 4.1x**
Cache-misses per row	9.2	4.6	5.1	3.0	**1.7 - 3.0x**
Nanosecond per row	105.3	63.9	31.2	20.8	**1.5 - 5.0x**