Storage Systems (StoSys)
XM_0092

Lecture 9: Distributed / Storage Systems - I

Animesh Trivedi
Autumn 2020, Period 2

VRIJE
UNIVERSITEIT
AMSTERDAM

Syllabus outline

1. ~~Welcome and introduction to NVM (today)~~
2. ~~Host interfacing and software implications~~
3. ~~Flash Translation Layer (FTL) and Garbage Collection (GC)~~
4. ~~NVM Block Storage File systems~~
5. ~~NVM Block Storage Key-Value Stores~~
6. ~~Emerging Byte-addressable Storage~~
7. ~~Networked NVM Storage~~
8. ~~Programmable Storage~~
9. Distributed Storage / Systems - I
10. Distributed Storage / Systems - II

Today's Agenda

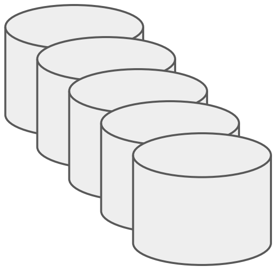
1. We are going to learn about managing temporary/ephemeral data - a new class of data type

2. Building a distributed store with high-performance networking and storage devices

3. Data formats? JSON, Parquet, ORC, are they good enough?

What is Temporary/Ephemeral Data?

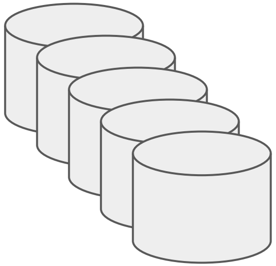
Any guesses?



Input datasets

Distributed data processing frameworks

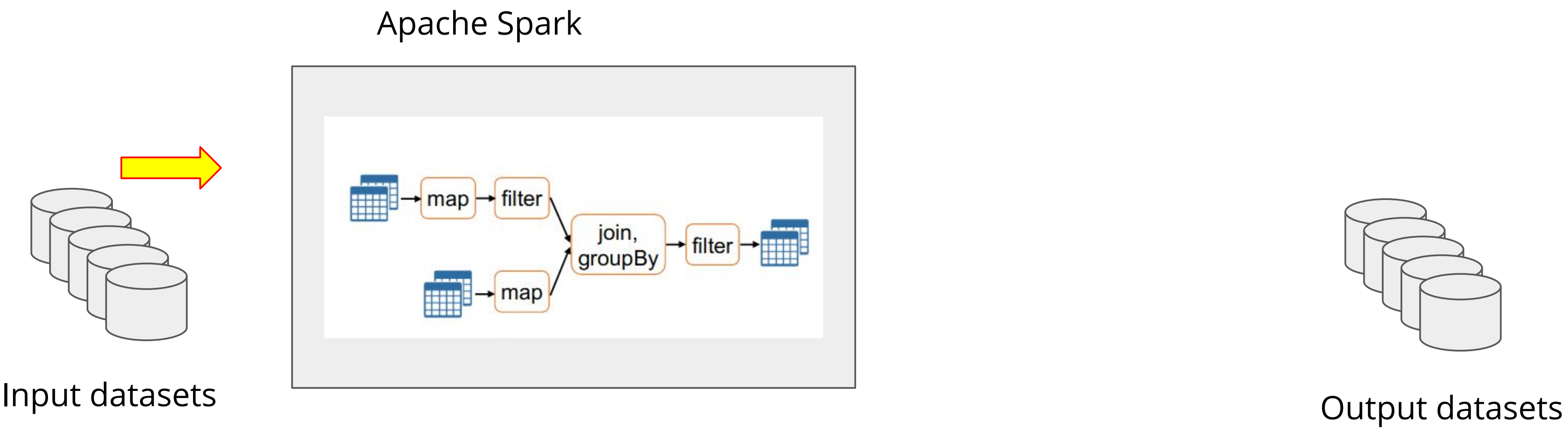
- Apache Spark
- Apache Hadoop (MapReduce)
- GraphLab
- Naiad (Dataflow)
- TensorFlow
- PyTorch
- ...



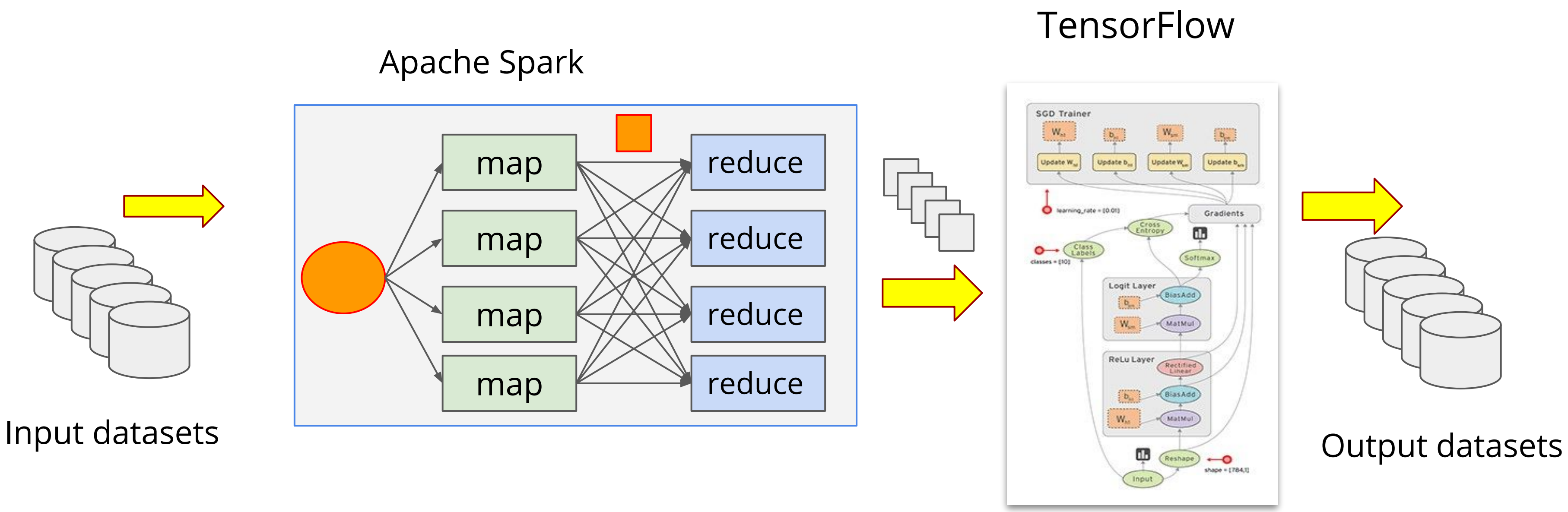
Output datasets

What is Temporary/Ephemeral Data?

Any guesses?

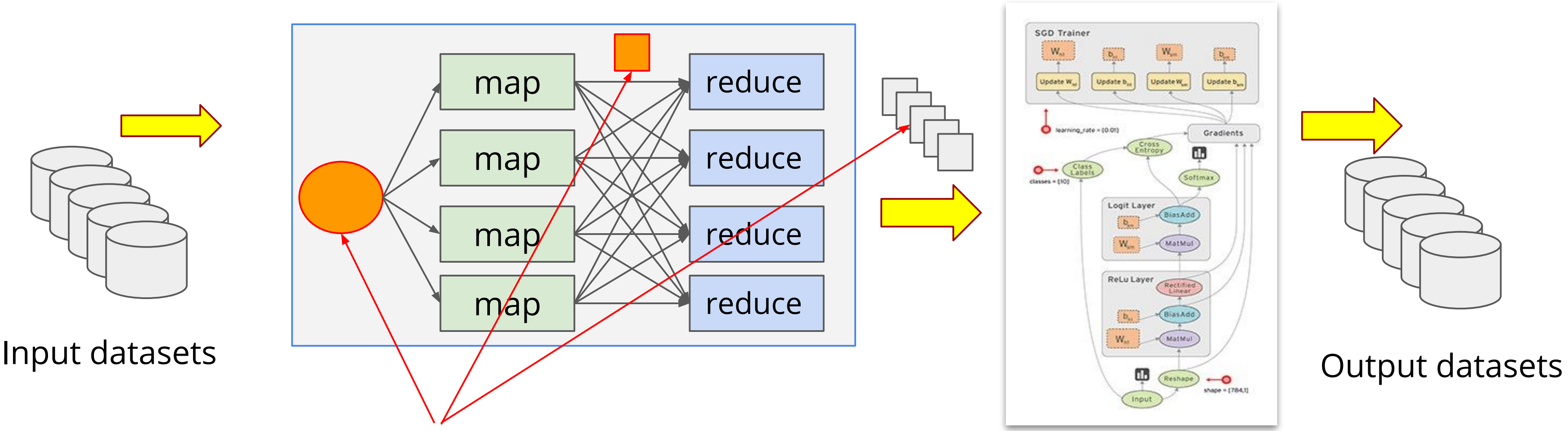


What is Temporary/Ephemeral Data?



What is Temporary/Ephemeral Data?

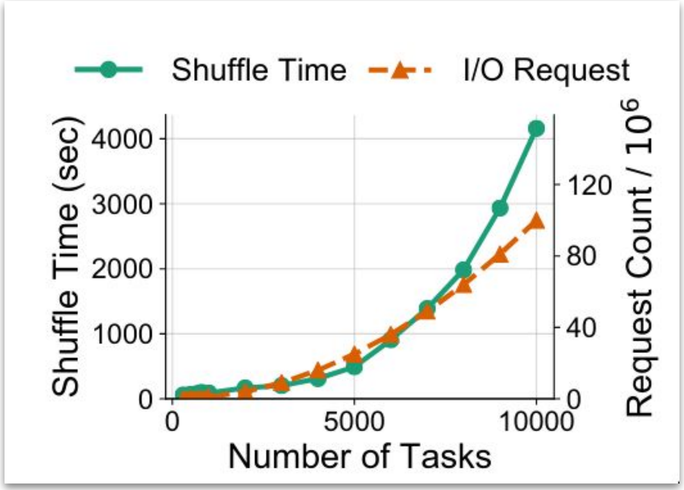
1. Read images, transform 2. Feature extraction 3. Training 4. Saving the model

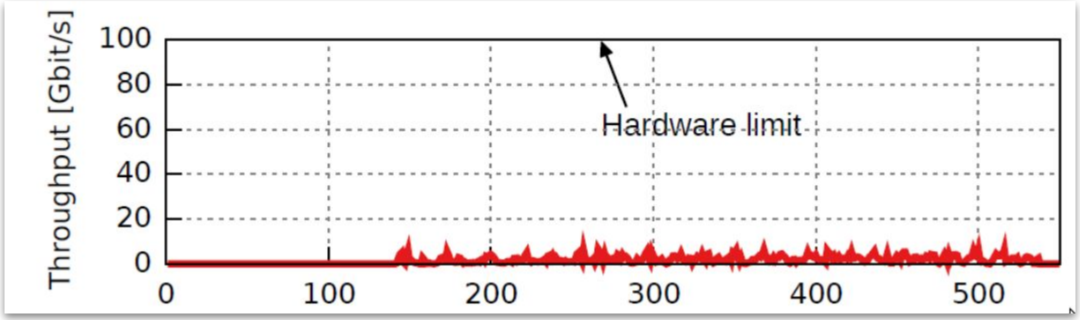


Between the initial dataset read, and the final dataset saved - there are many in-flight data objects which are temporary and ephemeral datasets

Challenges with Temporary Data Storage

1. Temporary data is performance critical - new network (100 Gbps) and storage (NVMe) can help





- Zhang et al., Riffle: optimized shuffle service for large-scale data analytics. In EuroSys 2018
- Ousterhout et al., Making sense of performance in data analytics frameworks. NSDI 2015.
- Trivedi et al., On the [ir]relevance of network performance for data processing. HotCloud 2016.

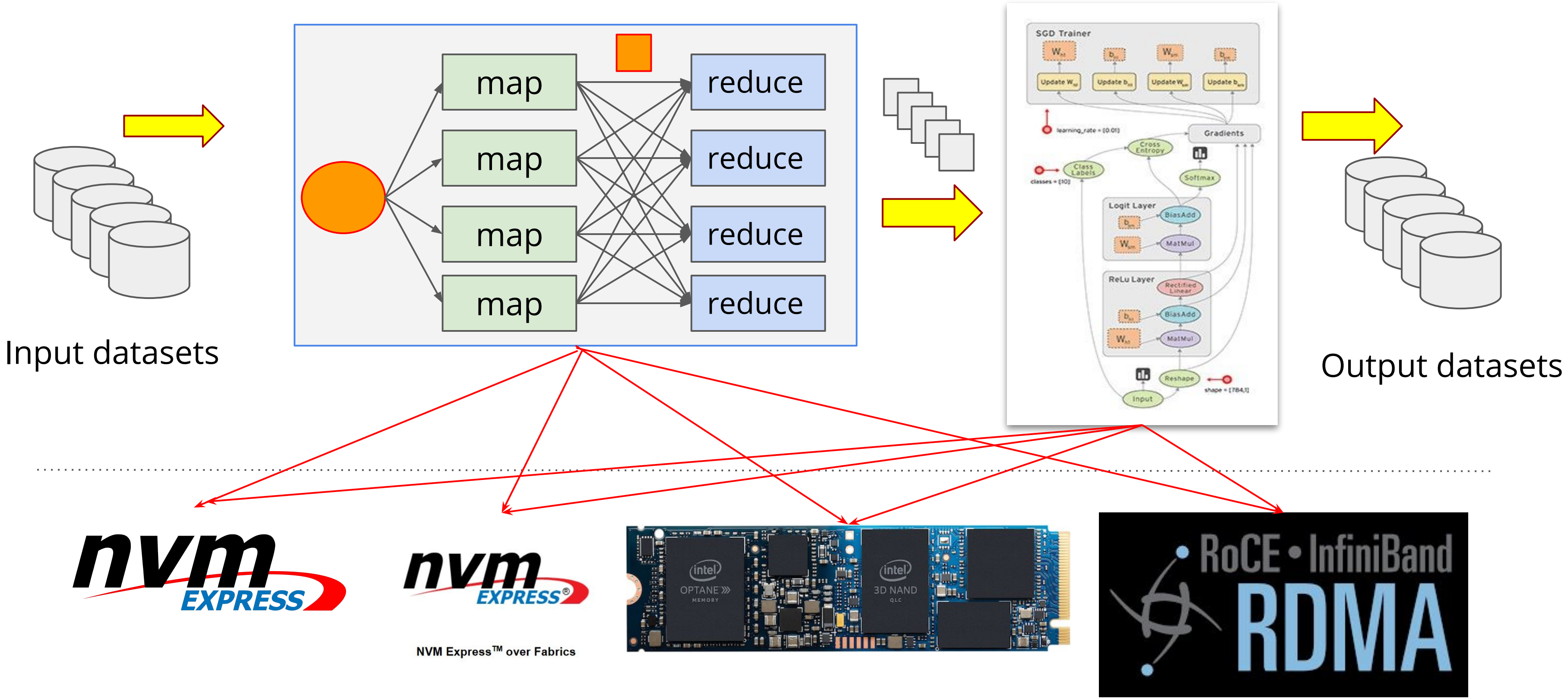
Challenges with Temporary Data Storage

1. Temporary data is performance critical - new network (100 Gbps) and storage (NVMe) can help

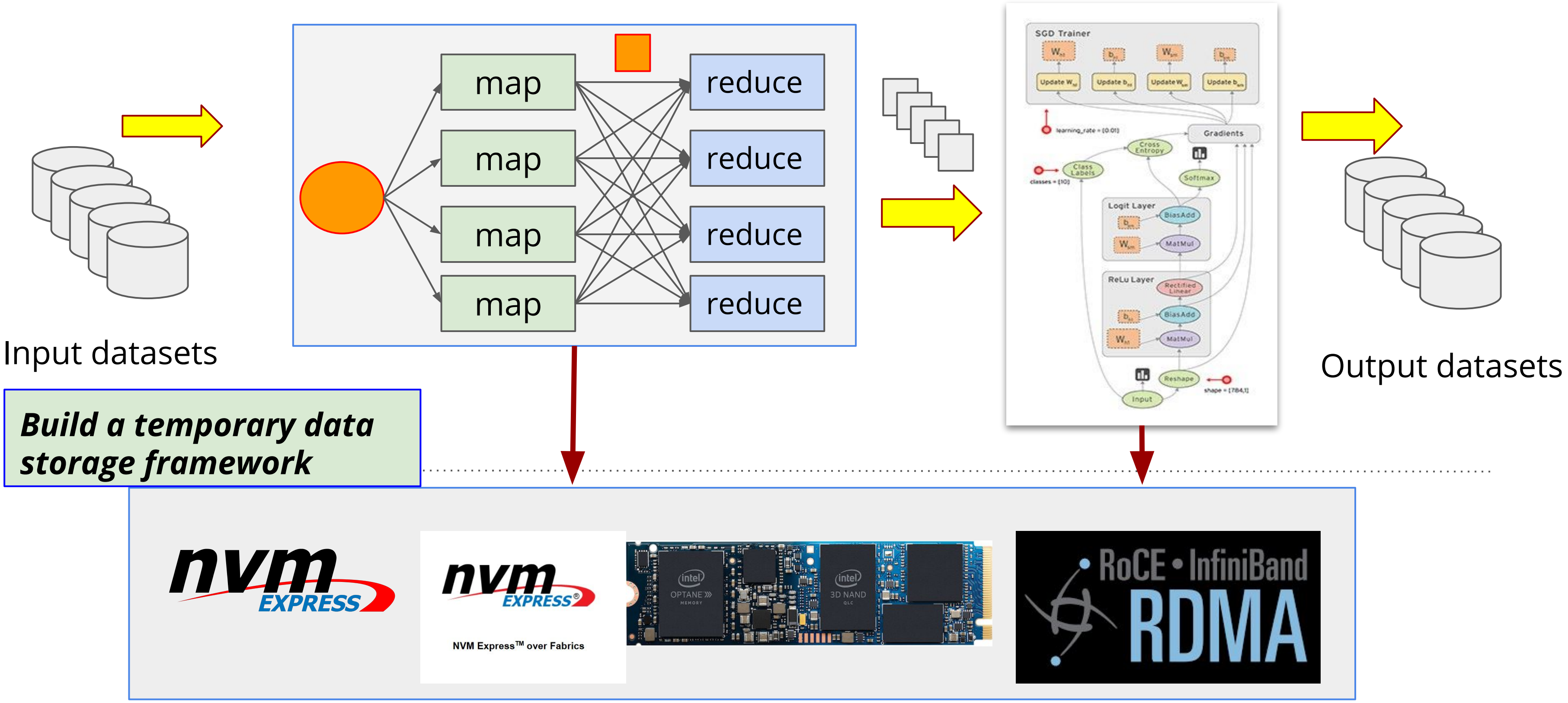
2. Temporary data have different needs
   a. No need to persist and provide fault tolerance
   b. Fault tolerance is often baked in compute framework used - Spark or TensorFlow

3. Complex integration into the compute framework
   a. Spark, TensorFlow, GraphLab, PyTorch -- all have their own way of processing data (RPCs)
   b. New technologies are coming - NAND Flash, Optane storage, PMEM, and mix of these
   c. New deployment models: DAS vs Disaggregated
   d. Programmable storage?

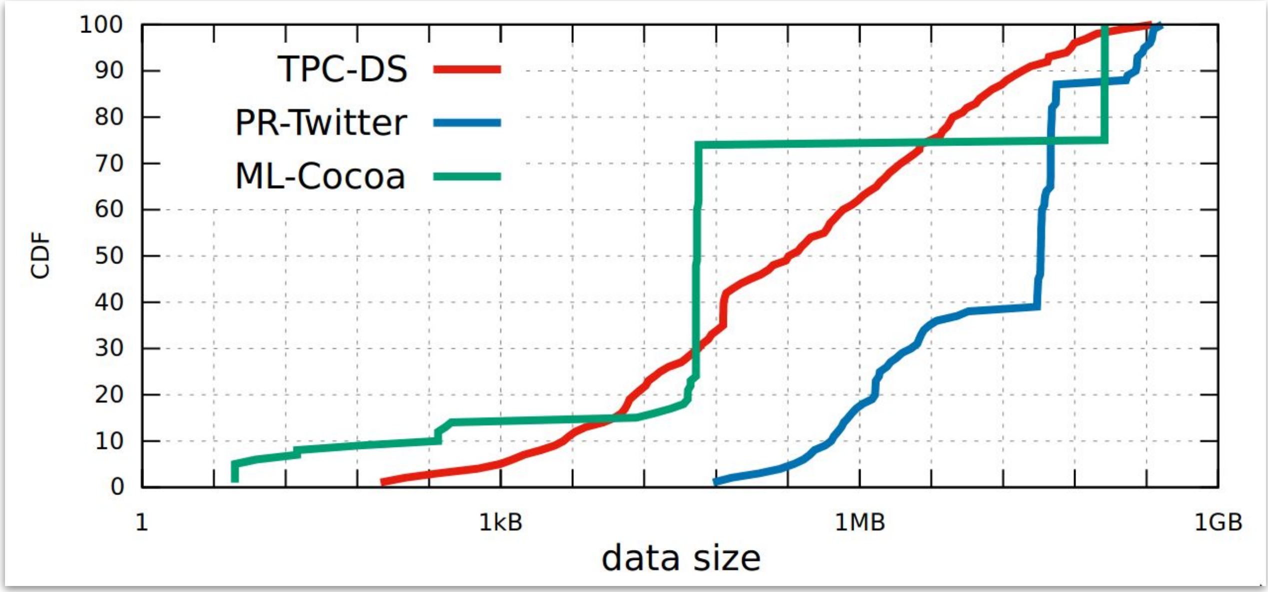
Temporary Data Management Spaghetti



Temporary Data Management Spaghetti



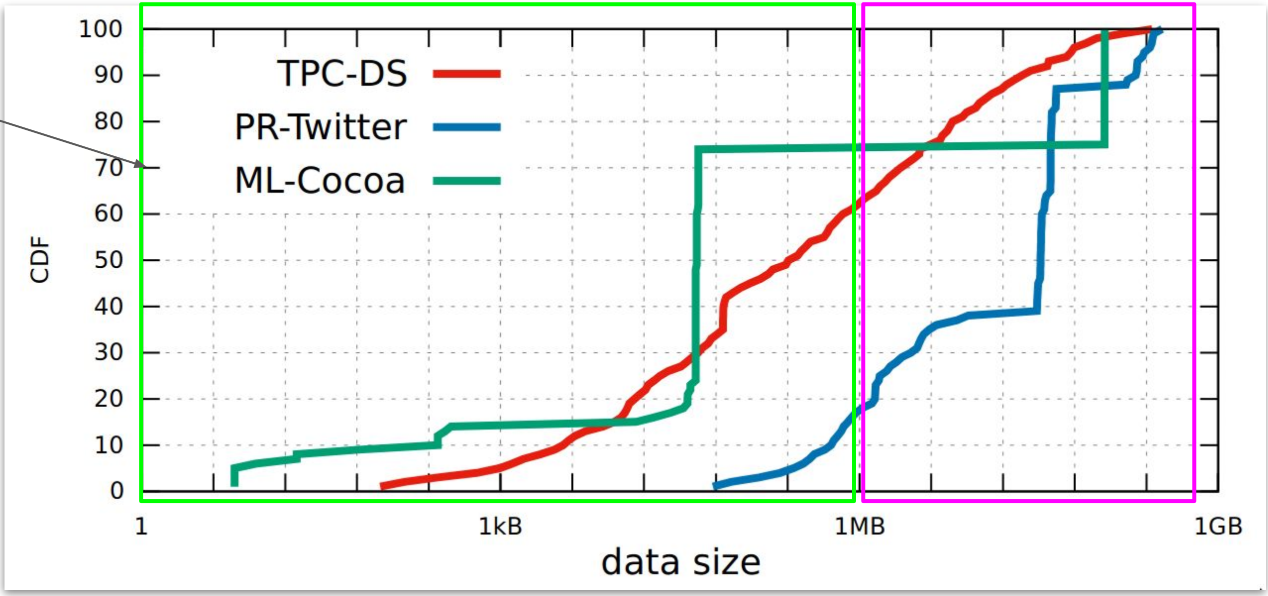
Can Existing Solutions Work?



Temporary data size distribution for three workloads (i) analytics; (ii) graph processing; (iii) ML

Can Existing Solutions Work?

Typically small values can be stored in KV Stores (latency driven)

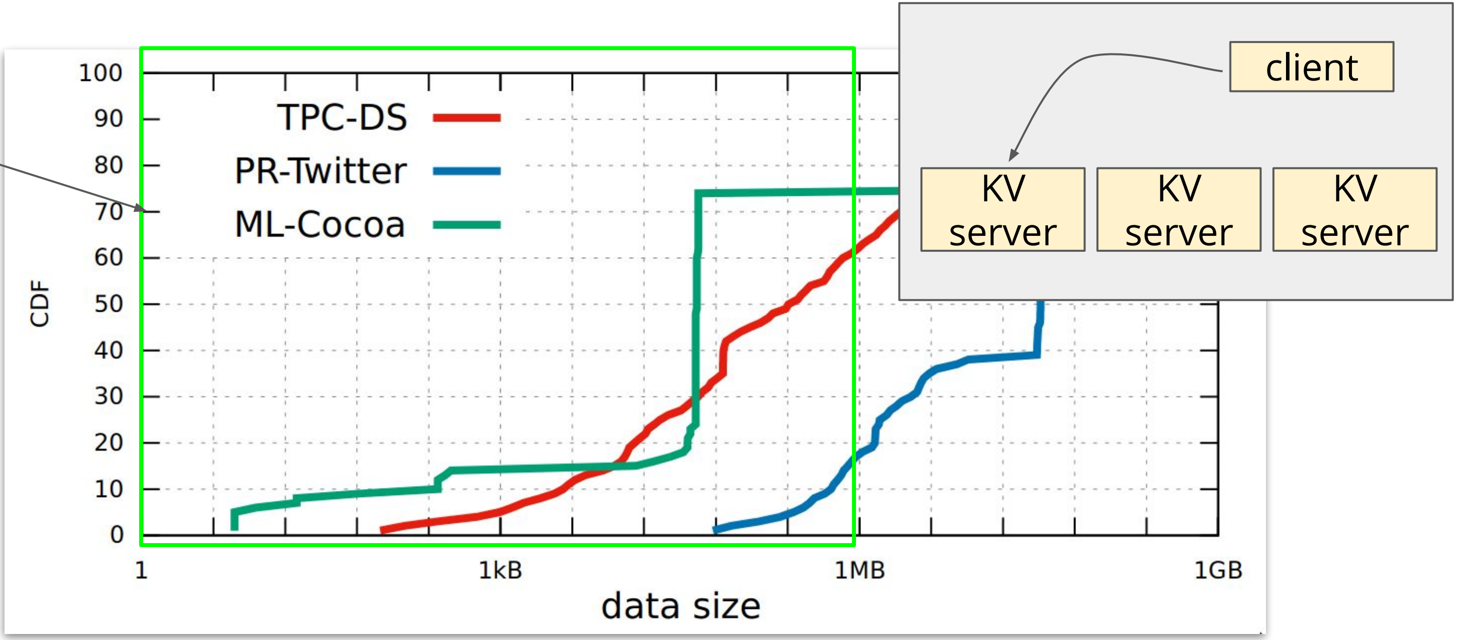


Typically large values are stored in file systems (bandwidth driven)

Temporary data size distribution for three workloads (i) analytics; (ii) graph processing; (iii) ML

Can Existing Solutions Work?

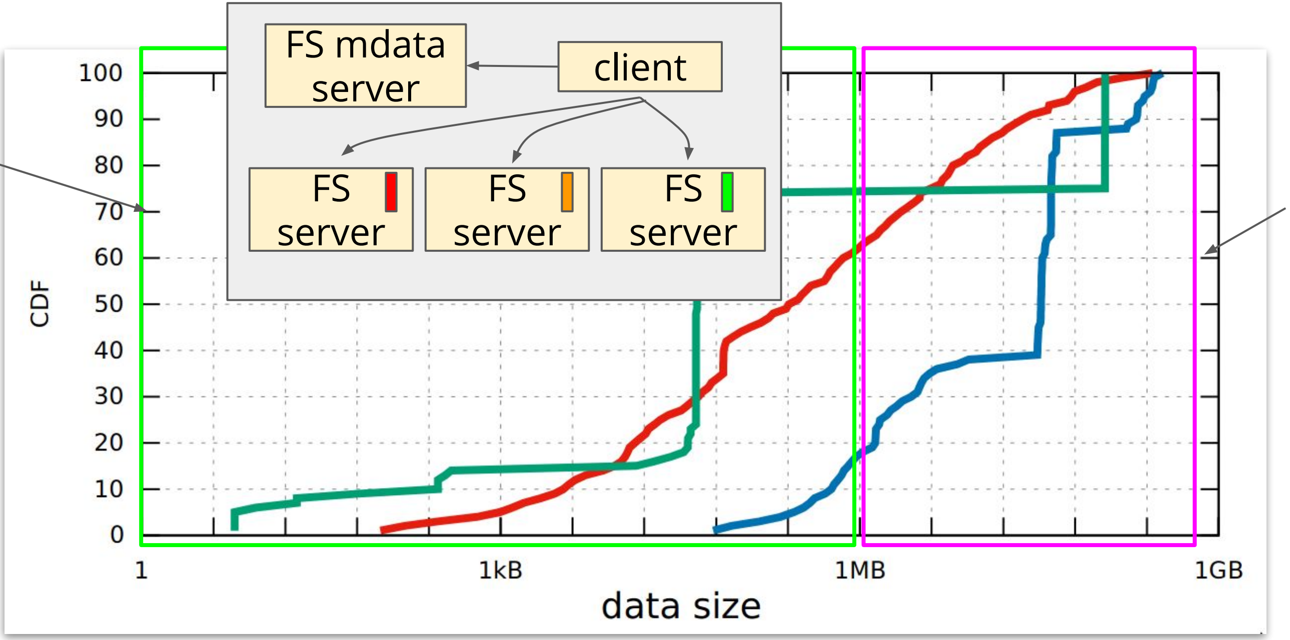
Typically small values can be stored in KV Stores (latency driven)



Temporary data size distribution for three workloads (i) analytics; (ii) graph processing; (iii) ML

Can Existing Solutions Work?

Typically small values can be stored in KV Stores (latency driven)



Typically large values are stored in file systems (bandwidth driven)

Temporary data size distribution for three workloads (i) analytics; (ii) graph processing; (iii) ML

The NodeKernel Architecture

A fused KV + File system distributed storage designed for temporary data storage, basic ideas

1. With fast network - FS and KV semantics can be provided in a single system
   a. Key Value Store = contact a single server + data transfer
   b. File Systems = contact metadata server + data servers + data transfer
   c. Nodes can be specialized : Tables, Directories, Files, workload specific files, Append-only, etc.

2. Split control and data planes
   a. Control plane = fast asynchronous RPCs
   b. Data plane = One-sided RDMA operations and NVMeF for I/O from DRAM and Flash storage

Trick: prepare and allocate all resources (carefully manage the NVM runtime) and do not intervene in offloaded I/O access operations

The NodeKernel Architecture

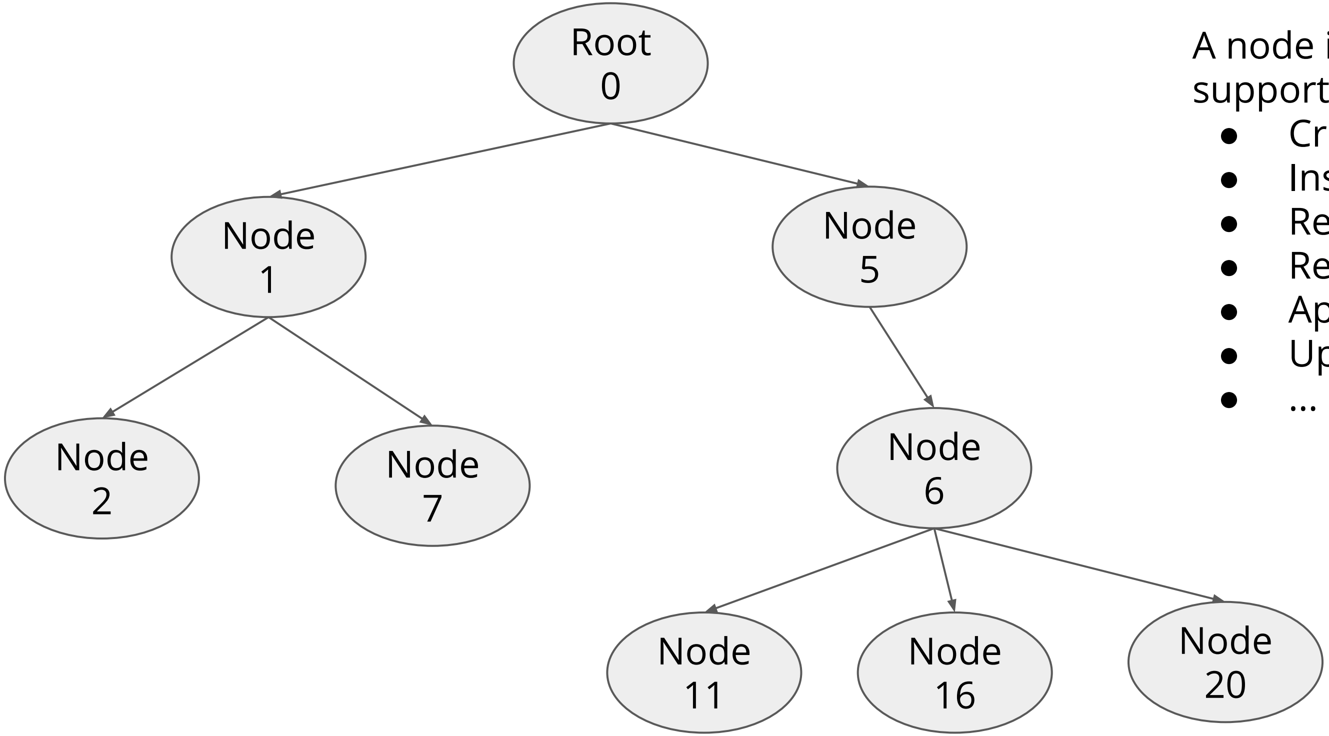
A fused KV + File system distributed storage designed for temporary data storage, basic ideas

- **Data Plane** - code path or calls where the actual work is done
  - Data r/w, make it straight forward, no blocking calls, everything is ready to go
- **Control Plane** - code path or call where resources are managed
  - Slow(er), resourced need to be allocated and managed, can block

ly,

- **Fast path** - common case execution (typically few branches, decision making, very simple code)
  - Read a file from start to finish, all blocks arrive in order
- **Slow path** - more sanity checks (more branches, hence poor(er) performance)
  - Read a file in fragments with random accesses in between, error handling

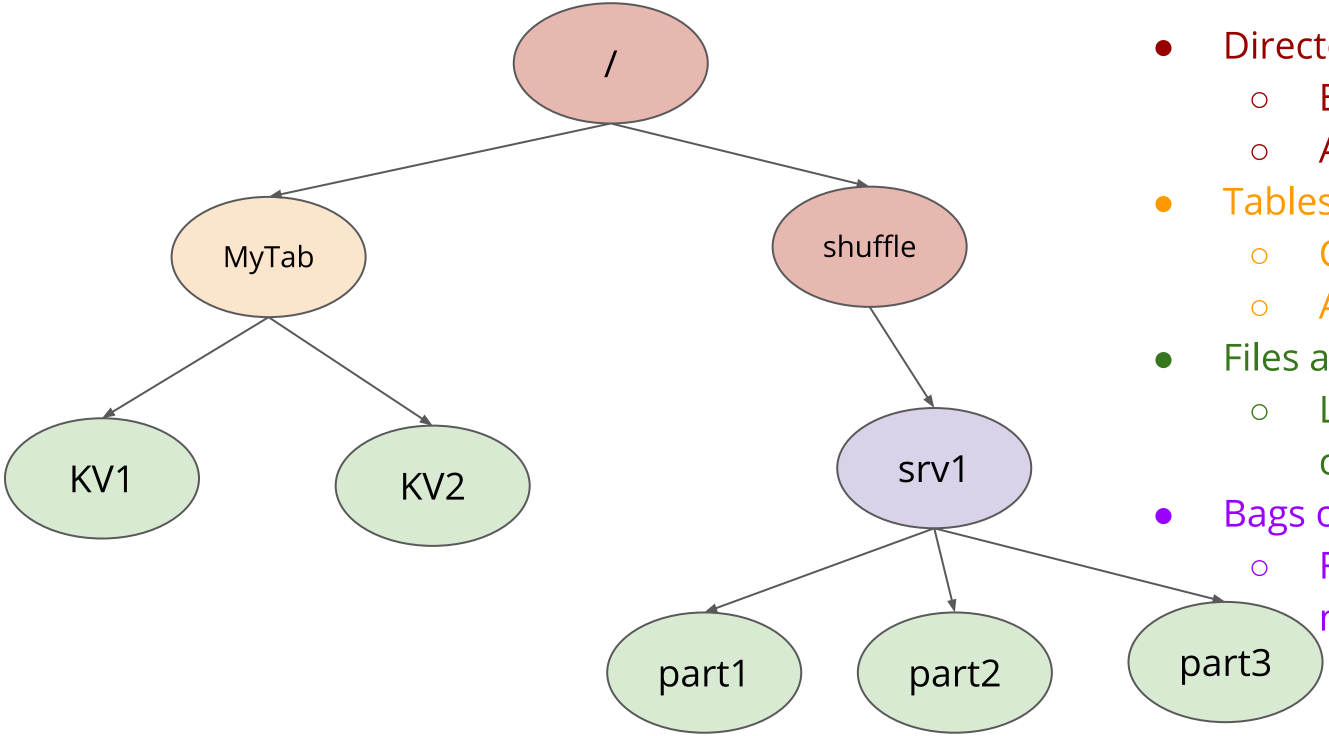
NodeKernel: A High-Level Idea



A node is an abstract type that supports

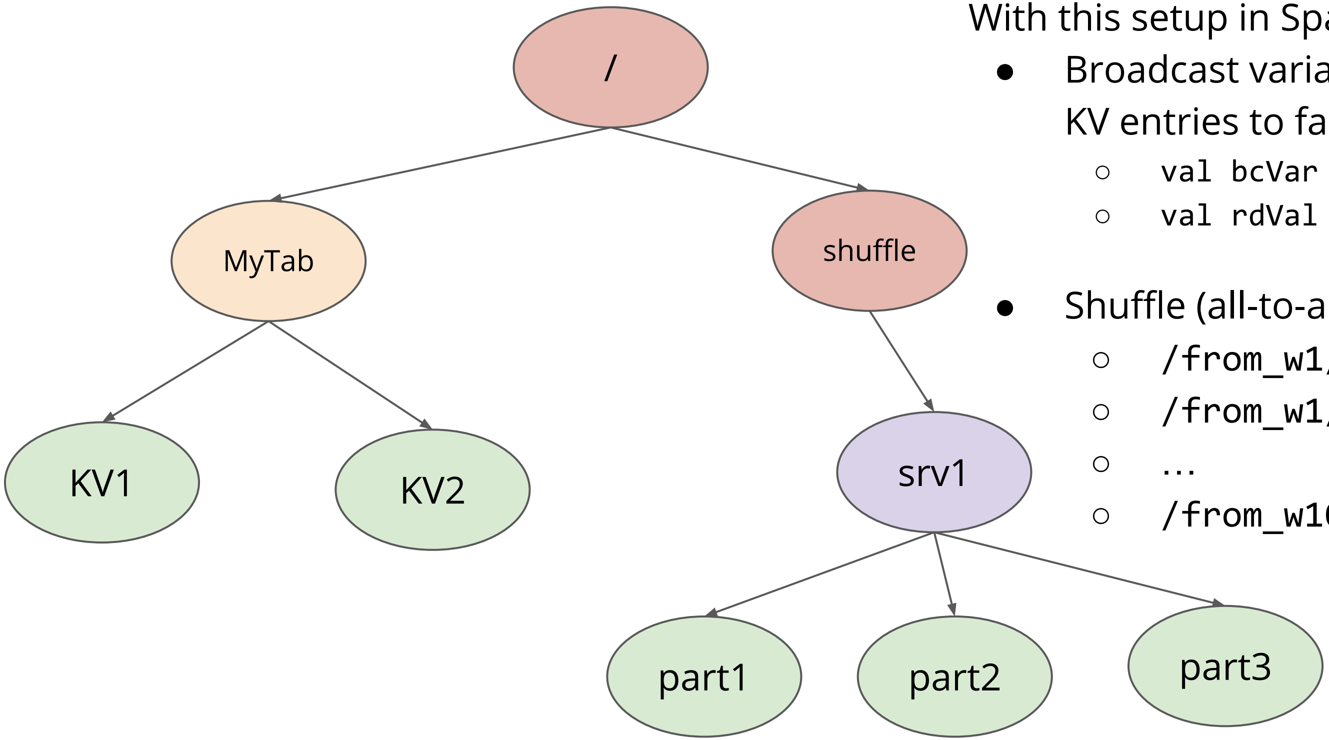
- Creating a node
- Inserting into the tree
- Removing from the tree
- Read
- Append
- Update
- …

NodeKernel: A High-Level Idea



- Directory
  - Enumerate
  - Add/remove files
- Tables
  - Collection of KVs
  - Add, remove KV files
- Files and KV files
  - Last winner vs error on concurrent creator
- Bags of directories
  - Fast data reading over multiple directories and files

NodeKernel: A High-Level Idea

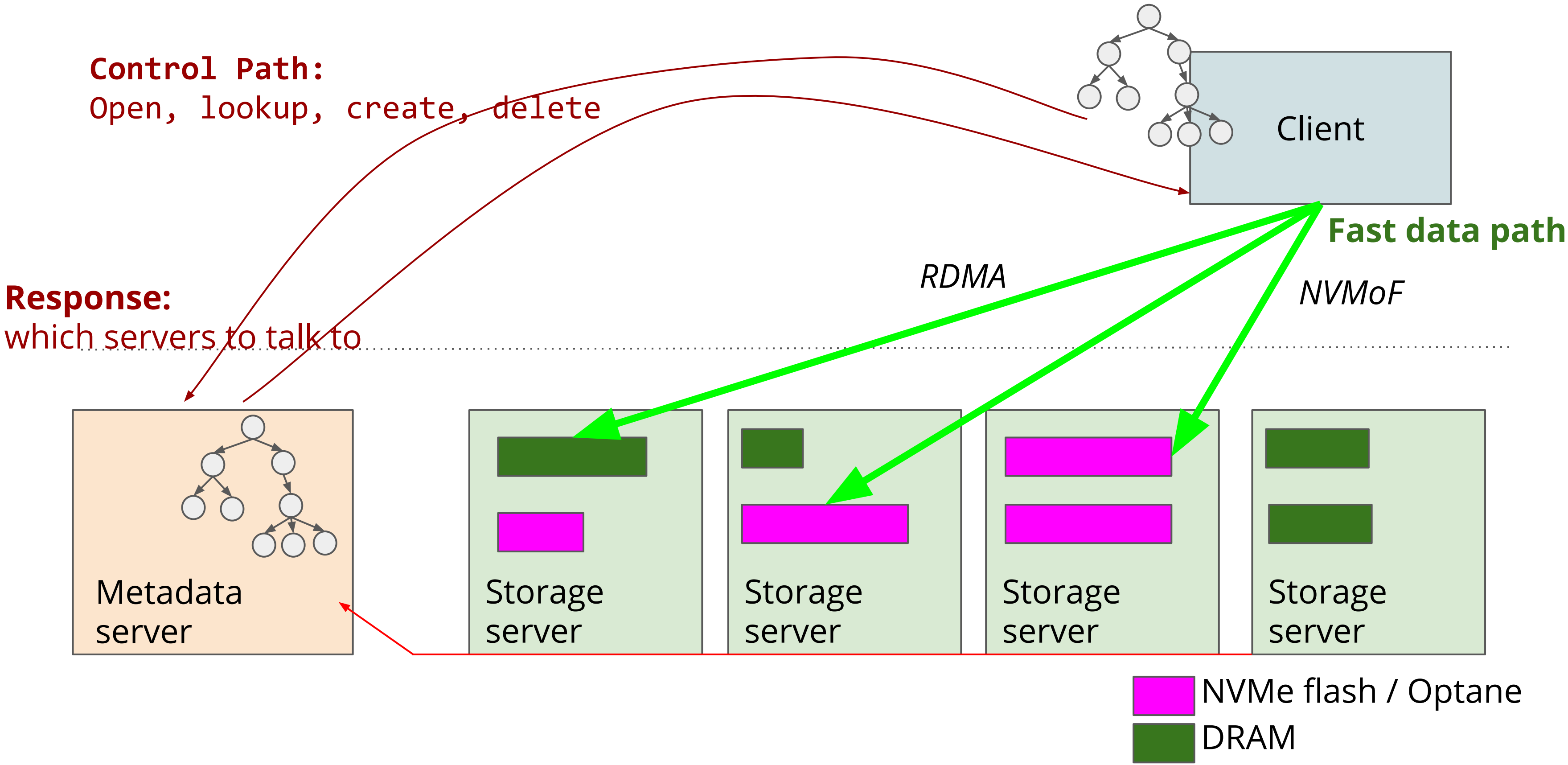


With this setup in Spark

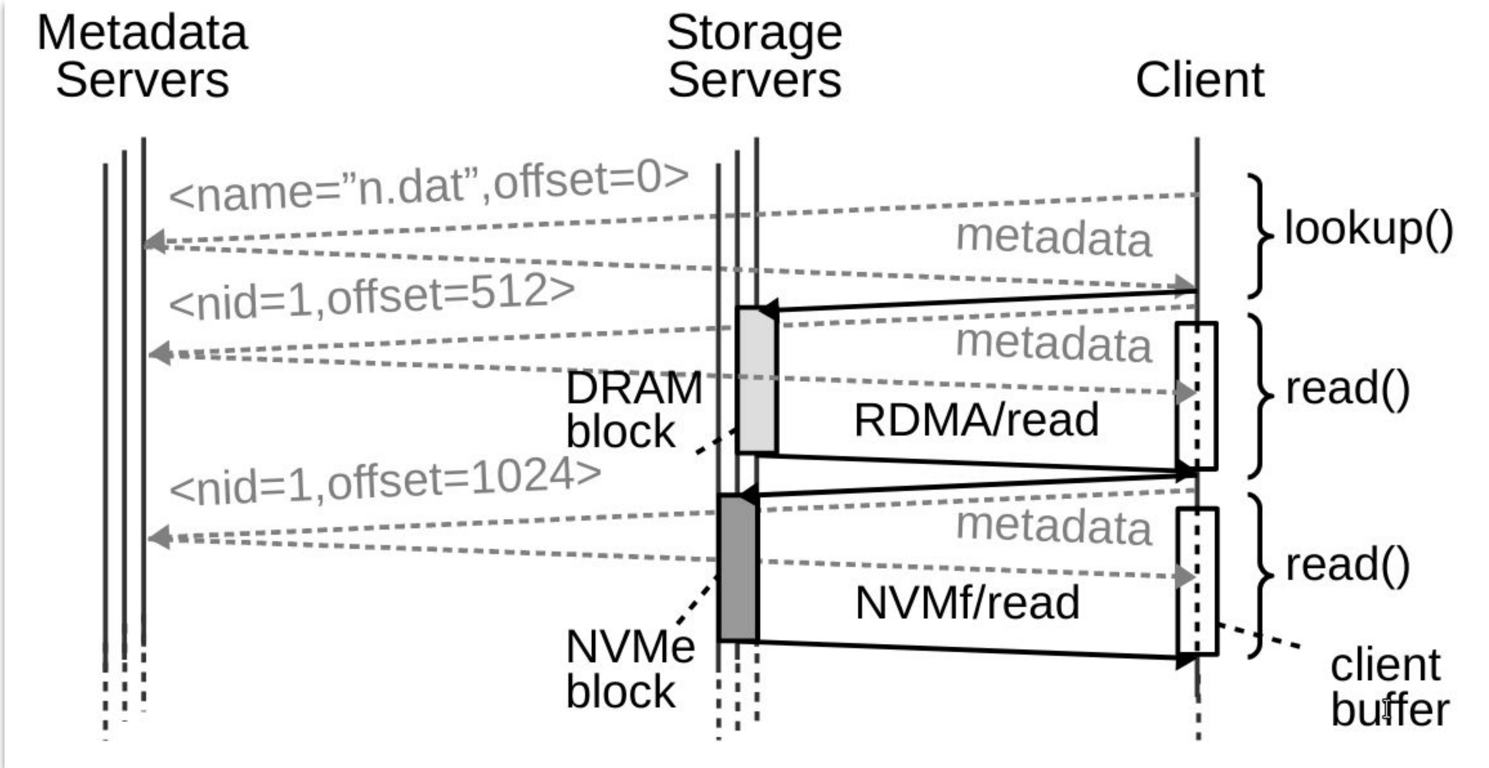
- Broadcast variables can be stored as a fast KV entries to fast lookups
  - `val bcVar = sc.broadcast(“10”) //put`
  - `val rdVal = bcVar.value.get() //get`

- Shuffle (all-to-all) can be (path enumeration)
  - `/from_w1/to_w2/file1`
  - `/from_w1/to_w3/file1`
  - …
  - `/from_w10/to_w1/file`

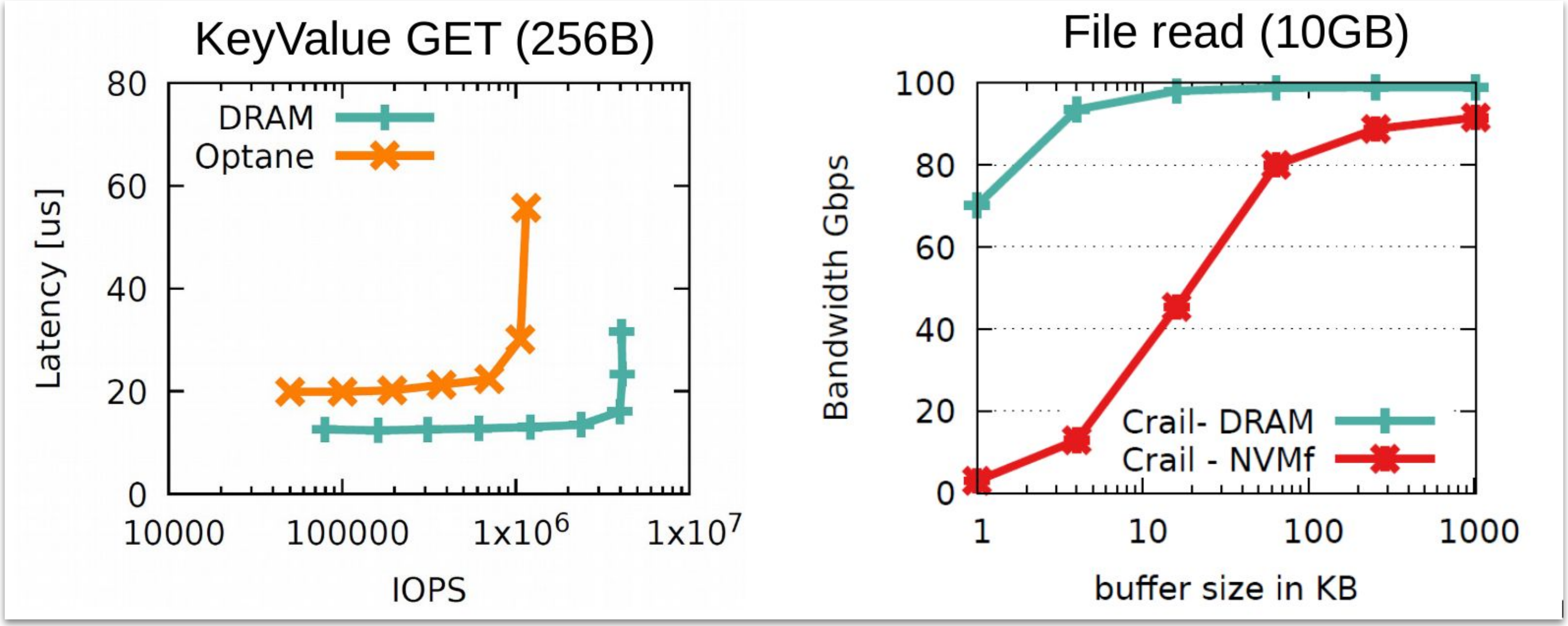
Implementation in Apache Crail



Heavily Pipelined Architecture



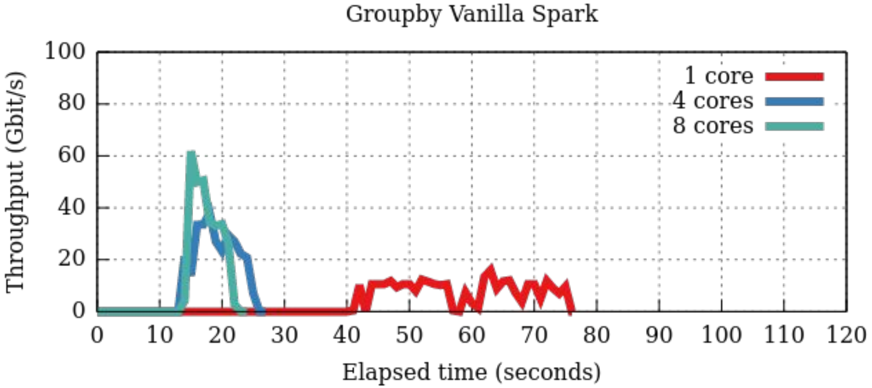
Performance

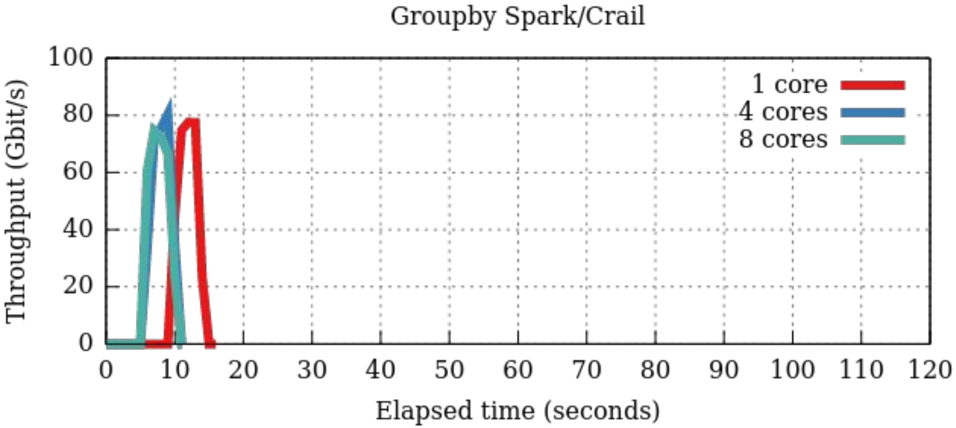


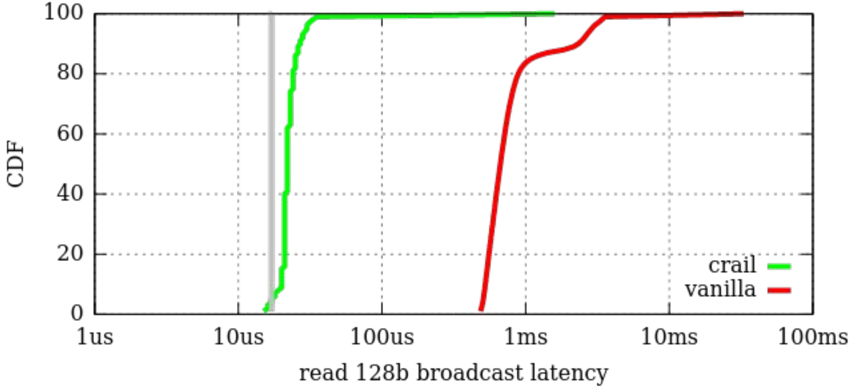
Small data sets (in KV mode): low latency with high IOPS

Large data sets (~10s GB, in FS mode): deliver high bandwidth

Integration with Spark: Shuffle and Broadcast





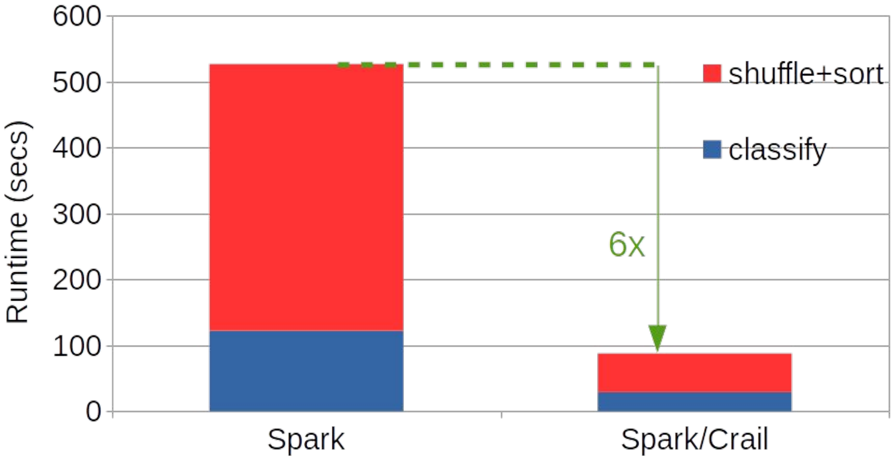


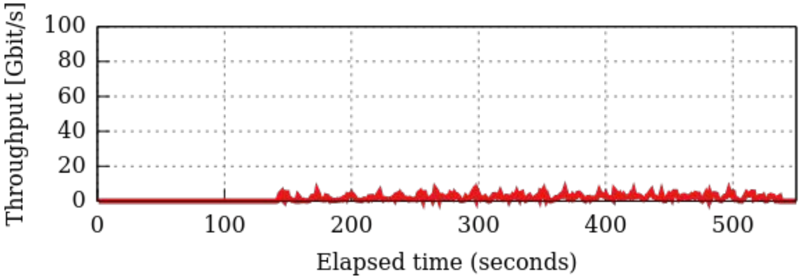
- 2-5x performance improvement in shuffle
- More than 10x gains for broadcast

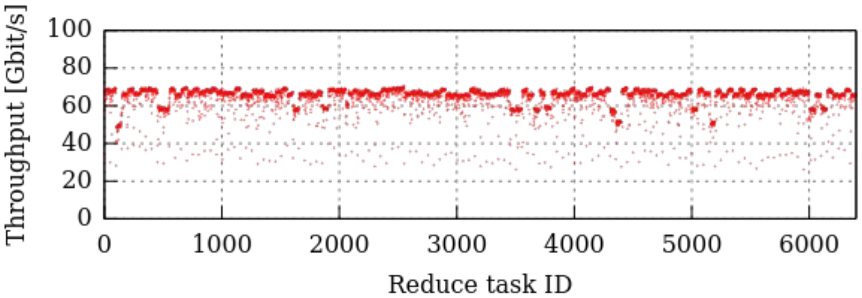
http://crail.incubator.apache.org/blog/2017/08/crail-memory.html

Putting Everything Together: TeraSort

- 128 nodes x 100 Gbps RoCE cluster, total dataset 12.8 TB

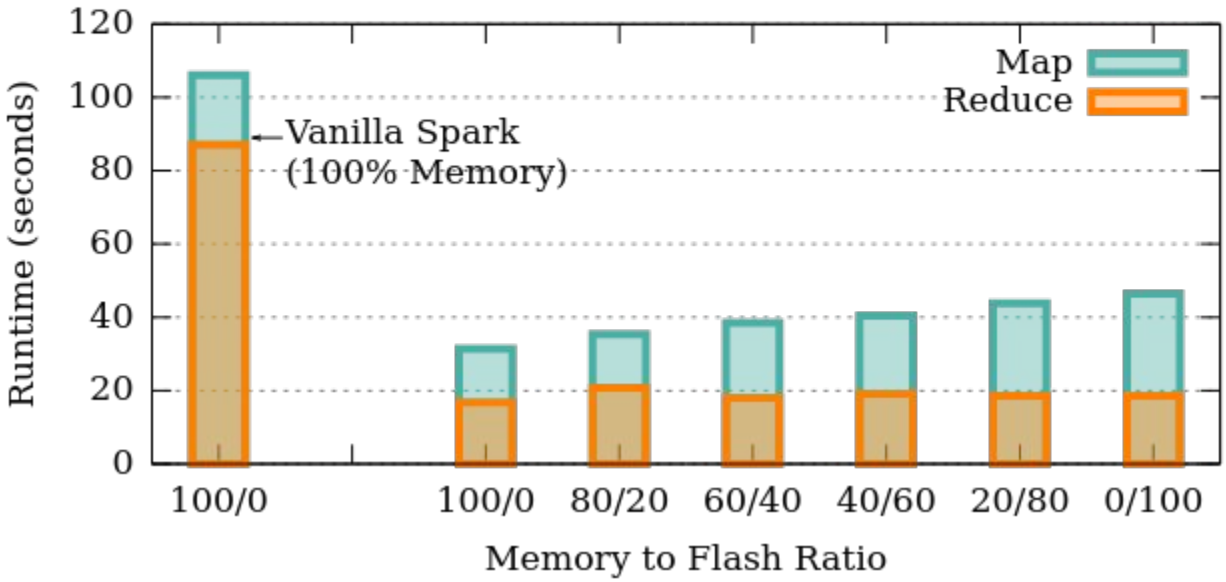






http://crail.incubator.apache.org/blog/2017/01/sorting.html

Storage Disaggregation

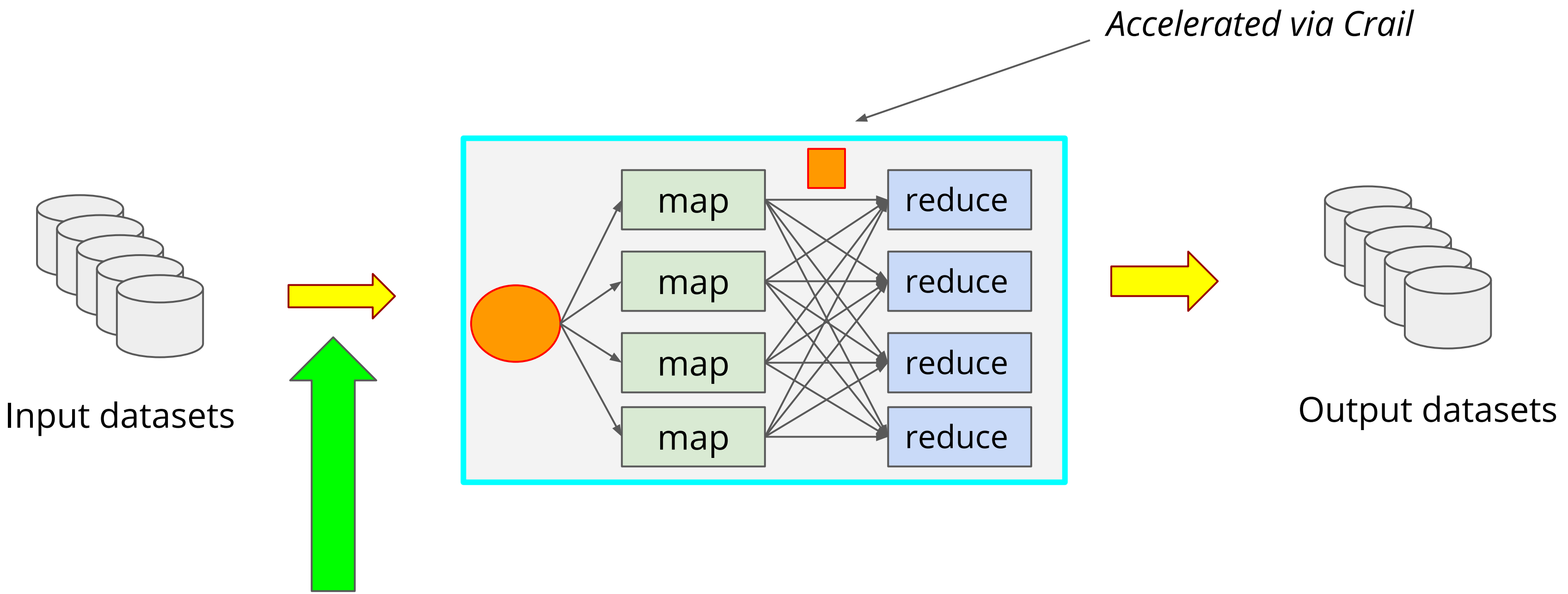


Storage disaggregation and moving all data from memory to flash for storage

- Flash is cheaper, more energy efficient, and denser than the DRAM

The whole in-flash shuffle storage is still faster than vanilla Spark in DRAM

So Far ...



We have seen that we can shuffle data very close to the hardware limits.
Can we actually feed data at that speed too?

Relational Data Processing Stacks 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 Stacks in the Cloud

Relational Engines

File Formats

Distributed Storage

Amazon S3

Back to the Future - It is 2010

Relational Engines

APACHE Spark

cloudera IMPALA

APACHE DRILL

HIVE

File Formats

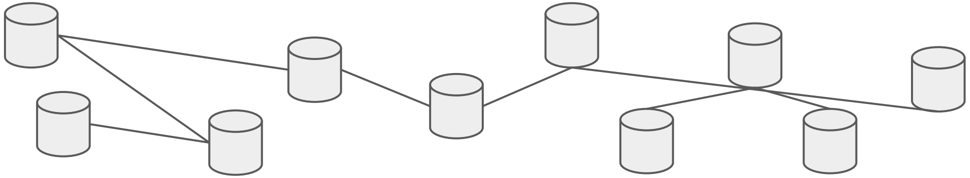
JSON

AVRO

Apache orc

Parquet

Hardware



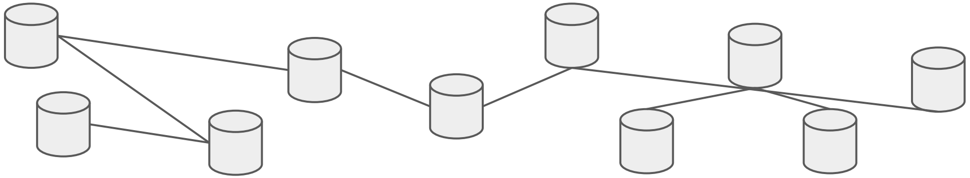
Disks connected over 1/10 Gbps network

Back to the Future - It is 2010

Relational Engines

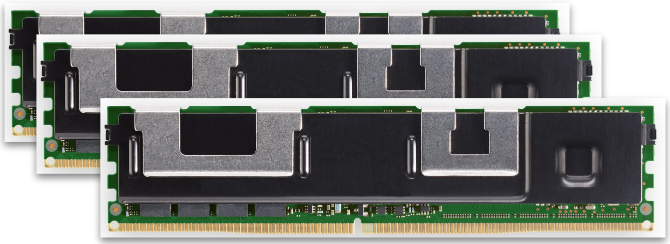
File Formats

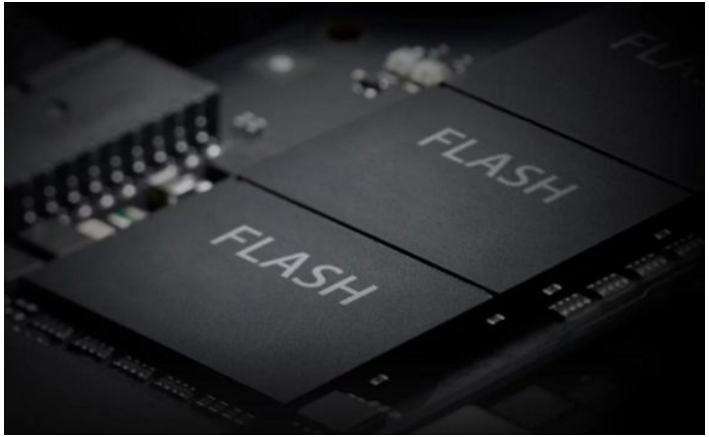
Hardware



Disks connected over 1/10 Gbps network

The I/O Revolution

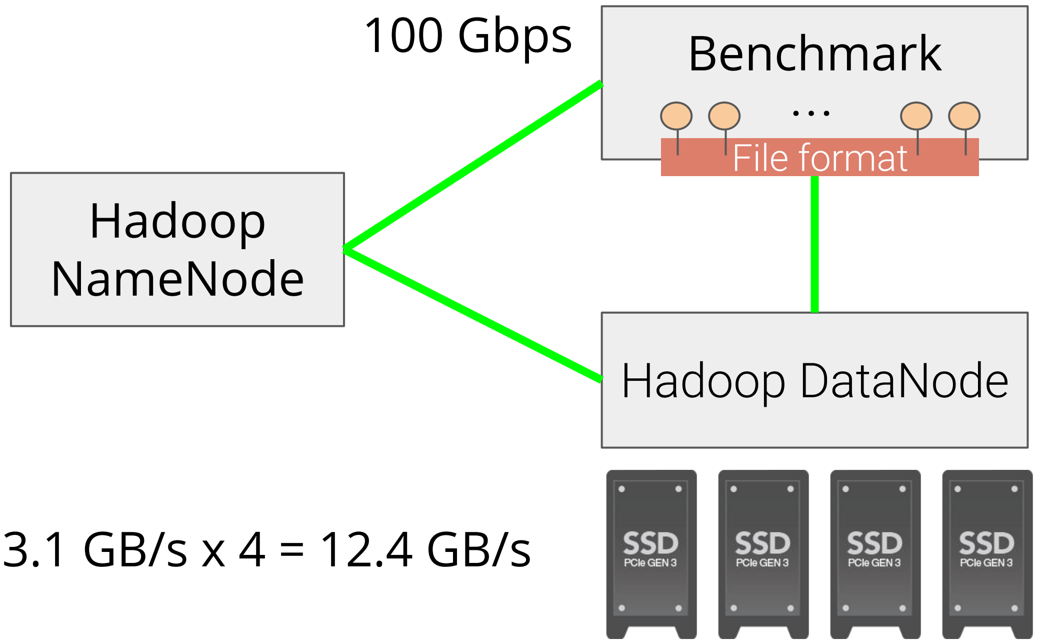




2-3 orders of magnitude performance improvements

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

The Impact of the Revolution



3.1 GB/s x 4 = 12.4 GB/s

Micro-benchmark*

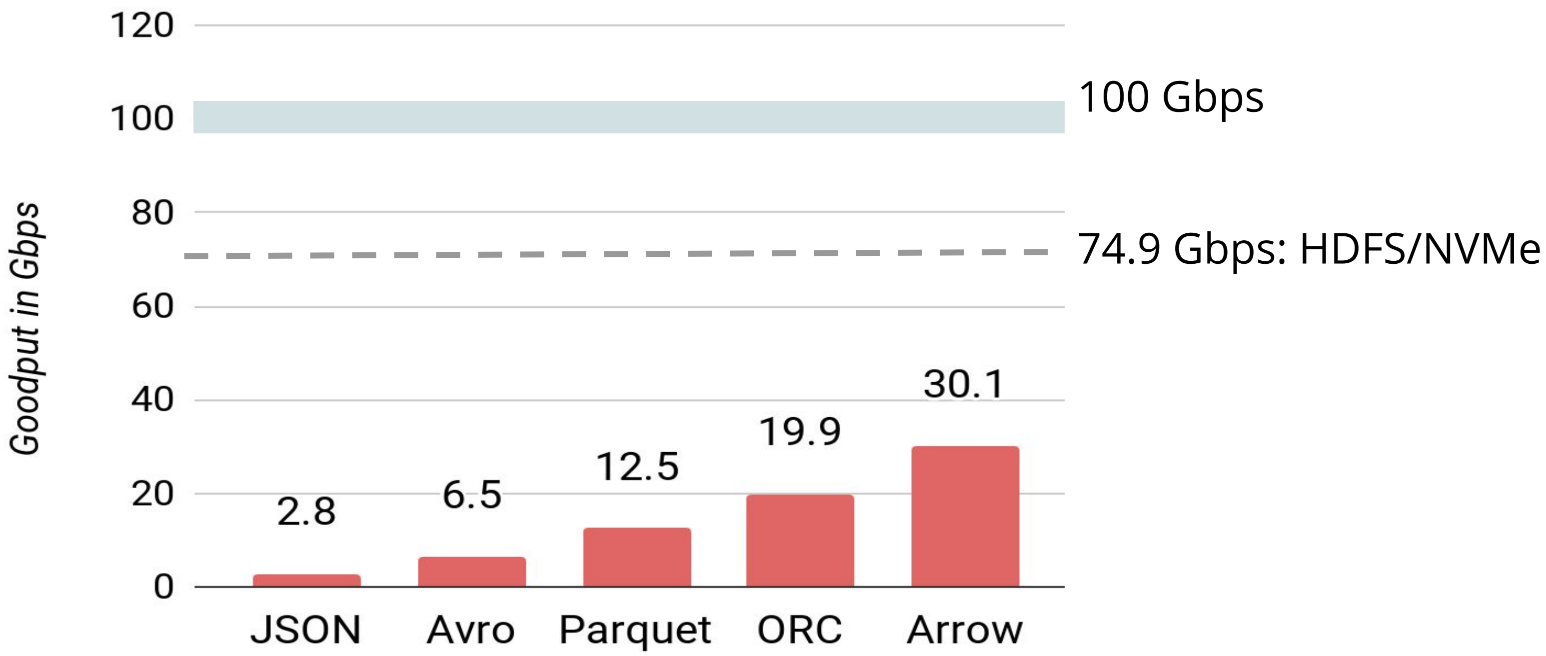
16 cores in parallel, reading TPC-DS data set.
What is the bandwidth?

Why micro-benchmark?
Decouple from the SQL engine

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

The Impact of the Revolution

The Impact of the Revolution



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

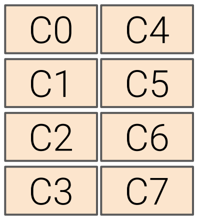
Distributed systems assumptions

Language/runtimes assumptions

2. Avoid slow, random small I/O

- preference for large block scans

But leads to bad CPU cache performance



128 MB

1 GB cache size?

Bounded by the number of instructions/row

Bounded by the poor cache/IPC performance

The Outdated Assumptions and Impact

End-host assumptions

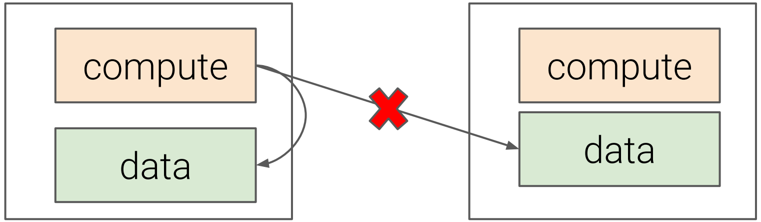
Distributed systems assumptions

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

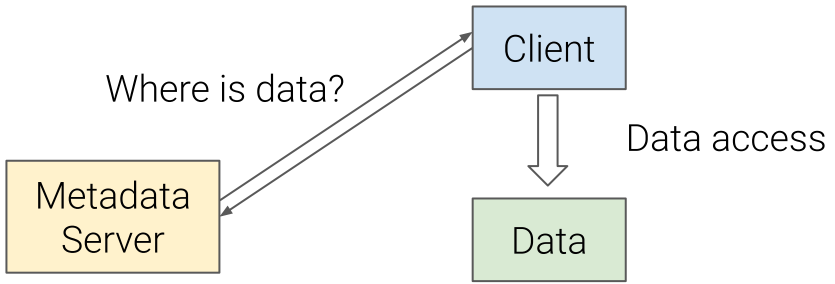
Distributed systems assumptions

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

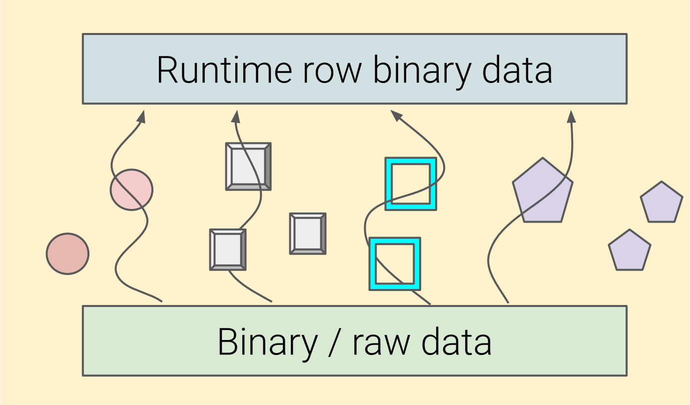
End-host assumptions

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

- 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

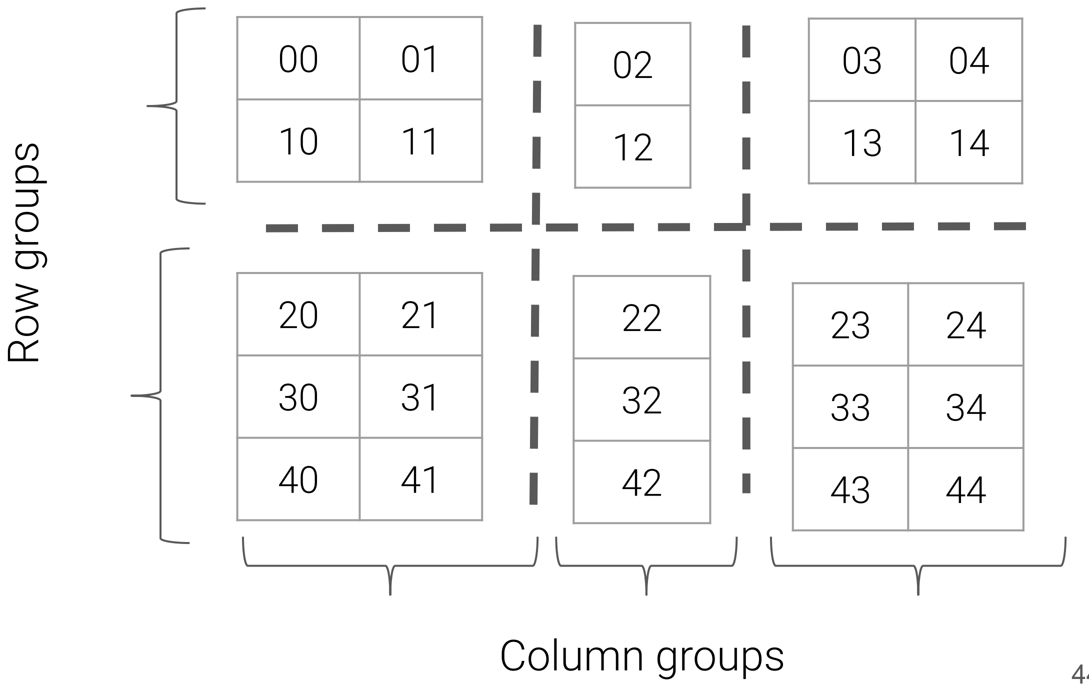


Table Storage Logic

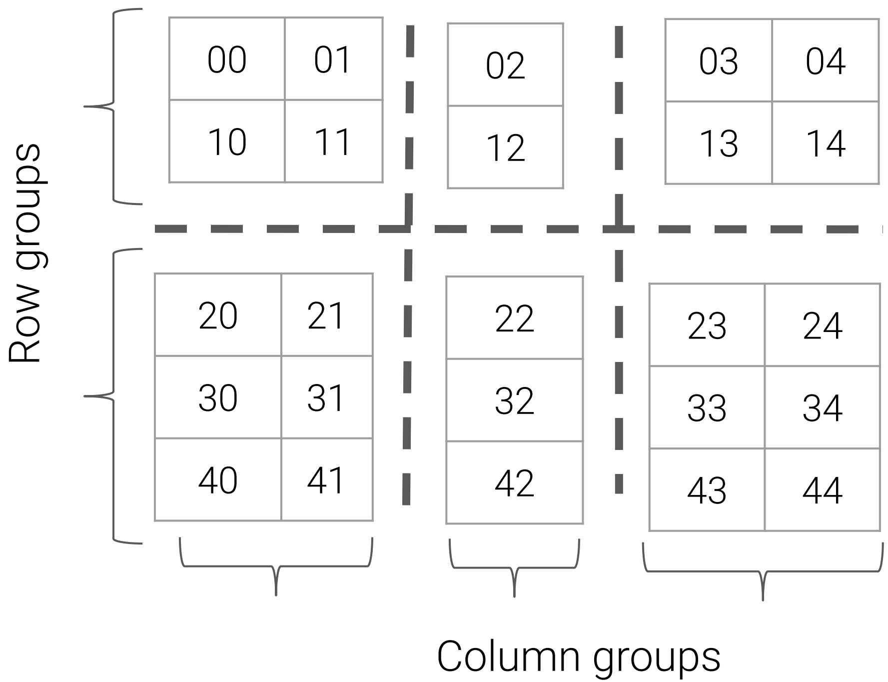
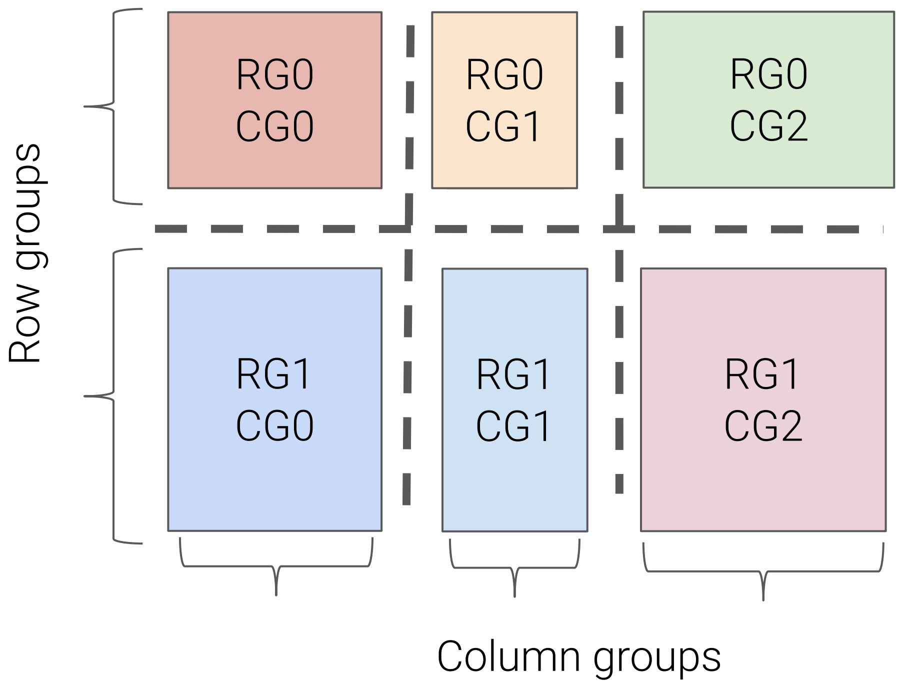
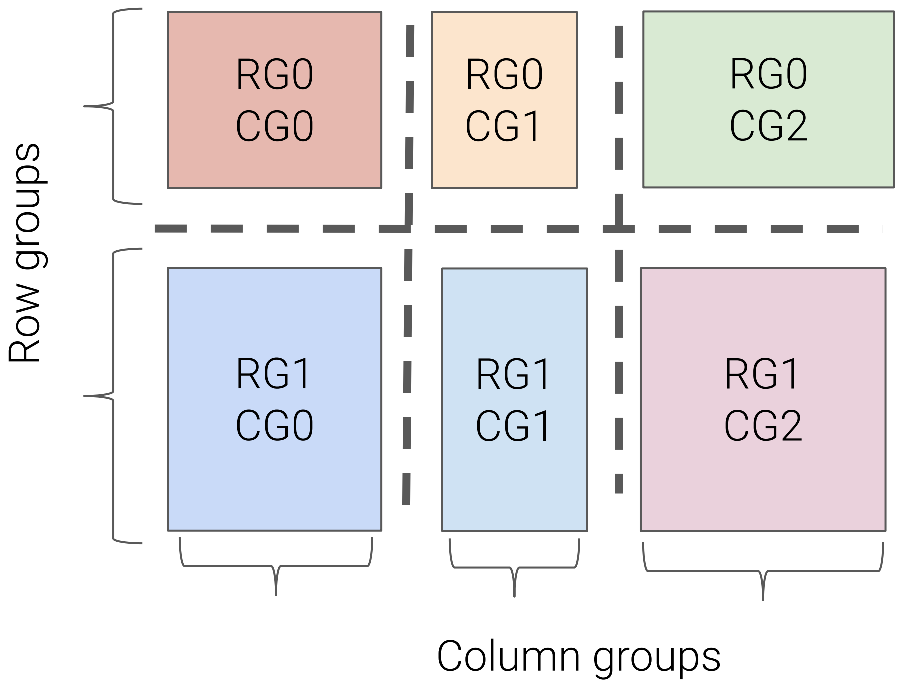


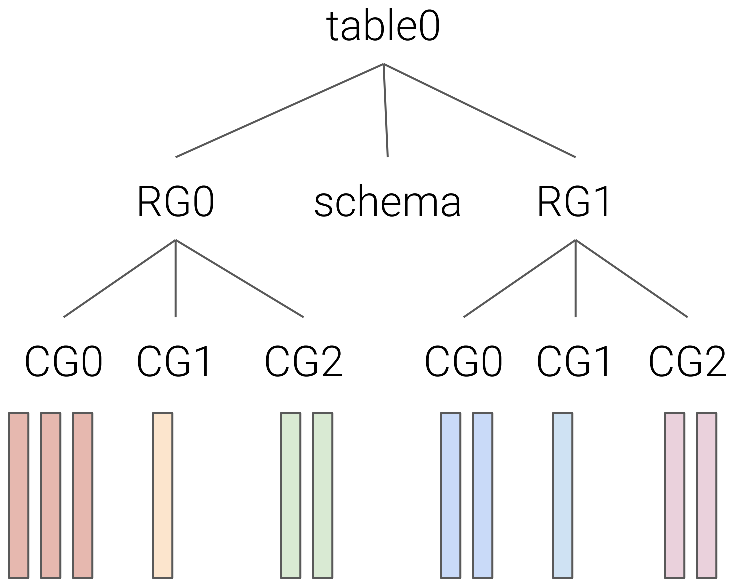
Table Storage Logic



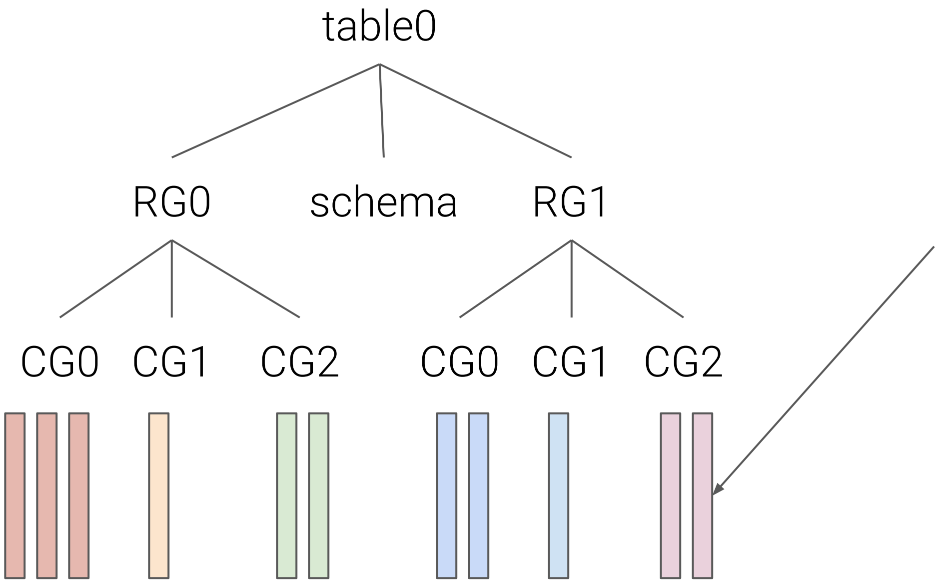
If there is only 1 column group : Row store
If there are ‘n’ column groups : Columns store

Table Storage Logic





Row Storage Format

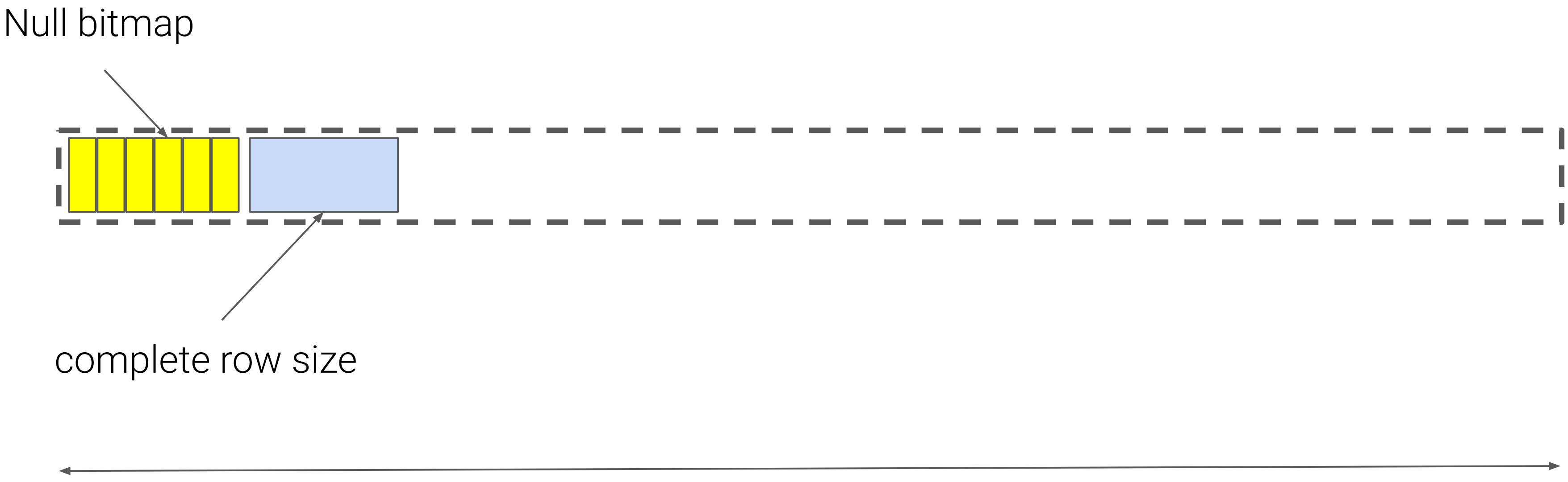


How is a single row of data stored in these files?

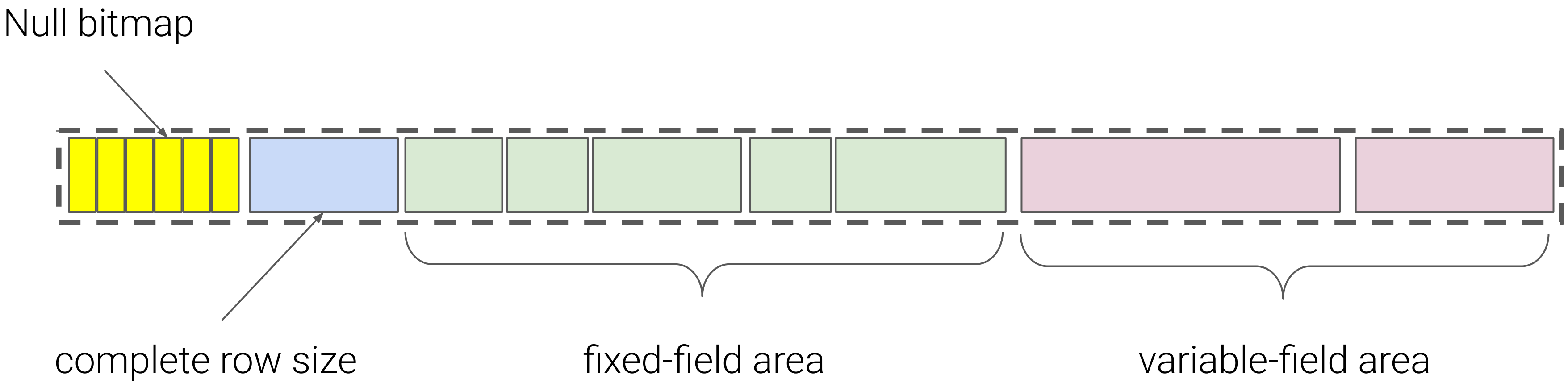
Row Storage Format

Marking null columns values

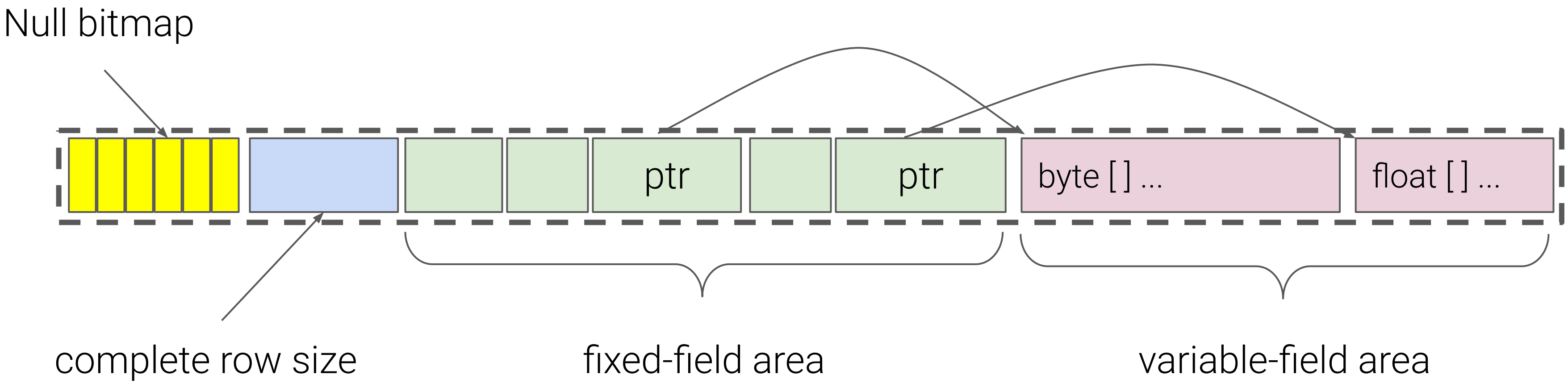
Row Storage Format



Row Storage Format

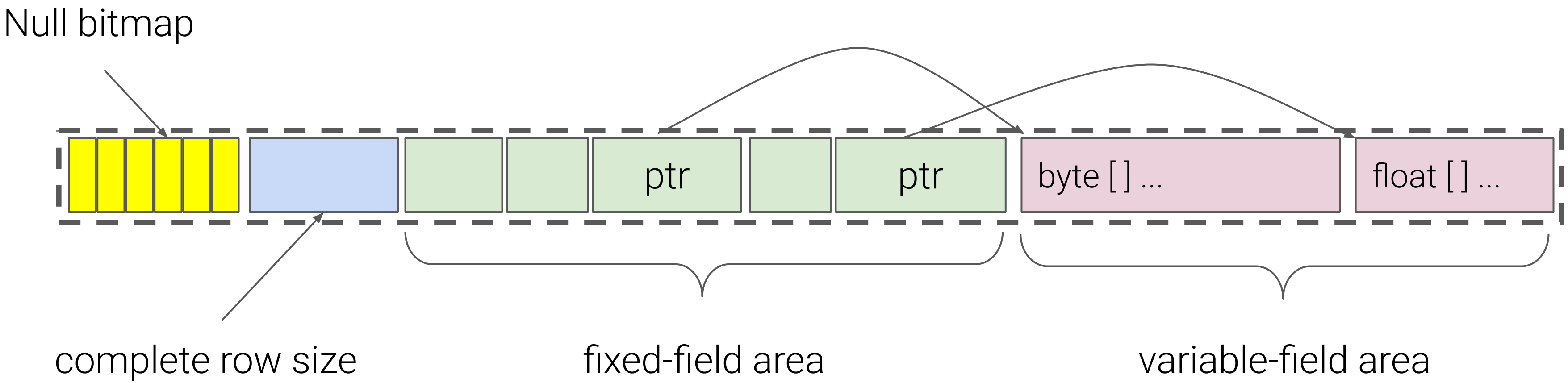


Row Storage Format



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

Row Storage Format



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

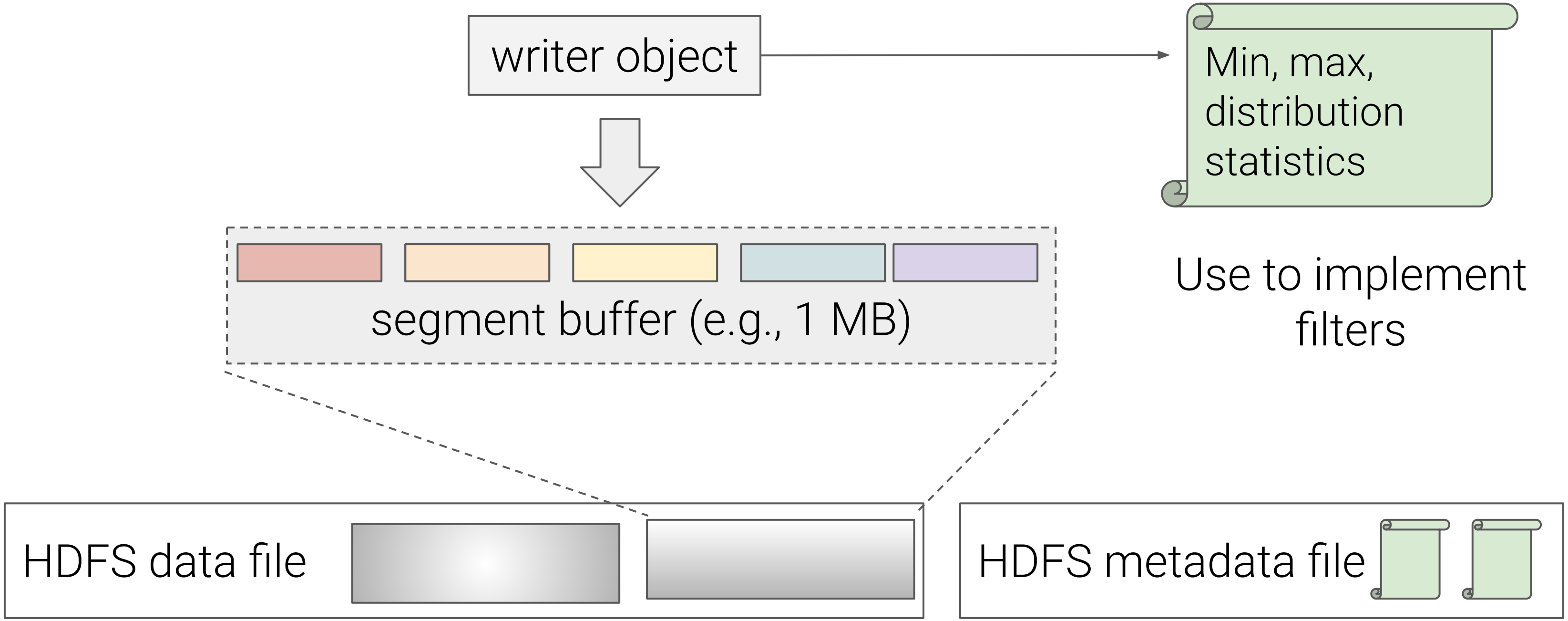
+ 1 byte bitmap (because there are 5 columns)

+ 4 byte size

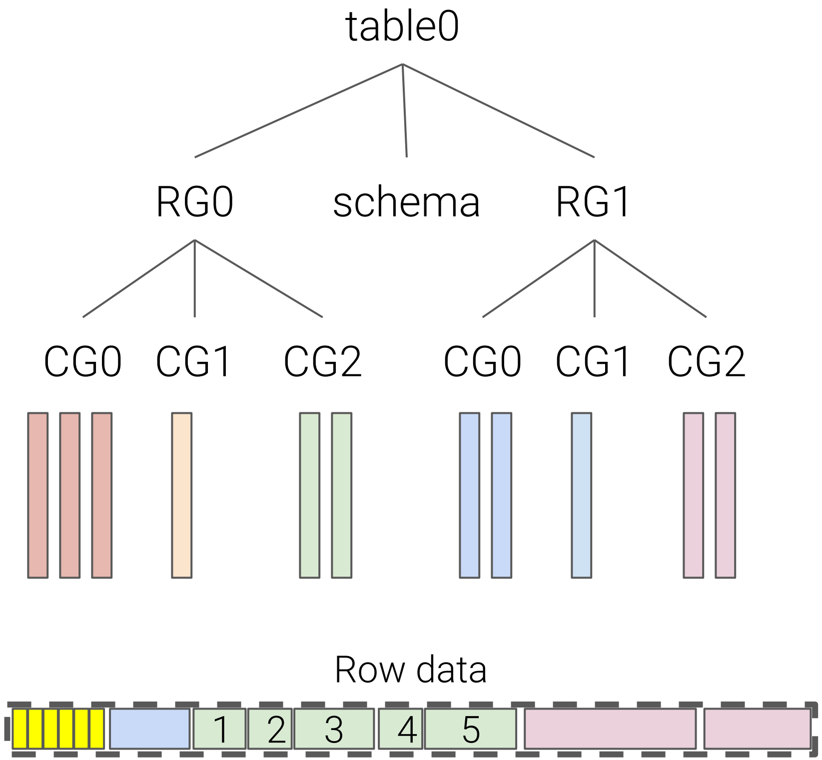
+ 4 byte (int) + 8 byte (double) + 8 byte (offset + size, ptr) + 1 byte (char) + 8 byte (offset + size, ptr)

= 34 bytes + variable area.

Writing Rows



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

Evaluation

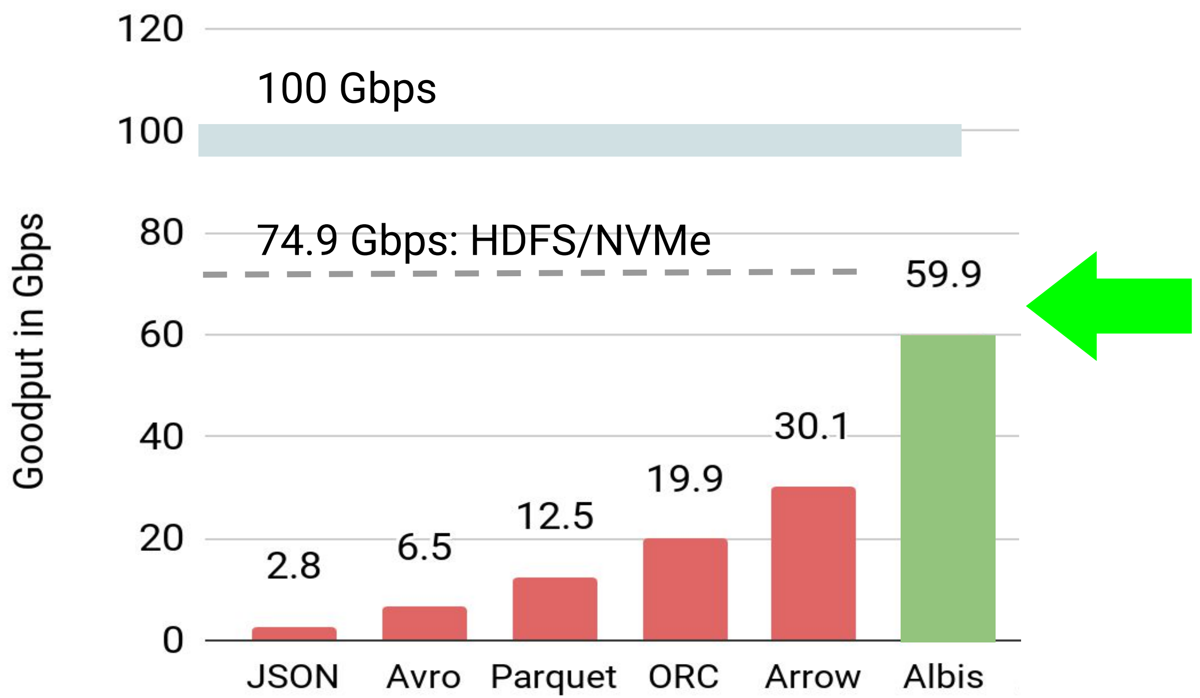
All experiments on a 4-node cluster with 100 Gbps network and flash devices

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

Three fundamental questions

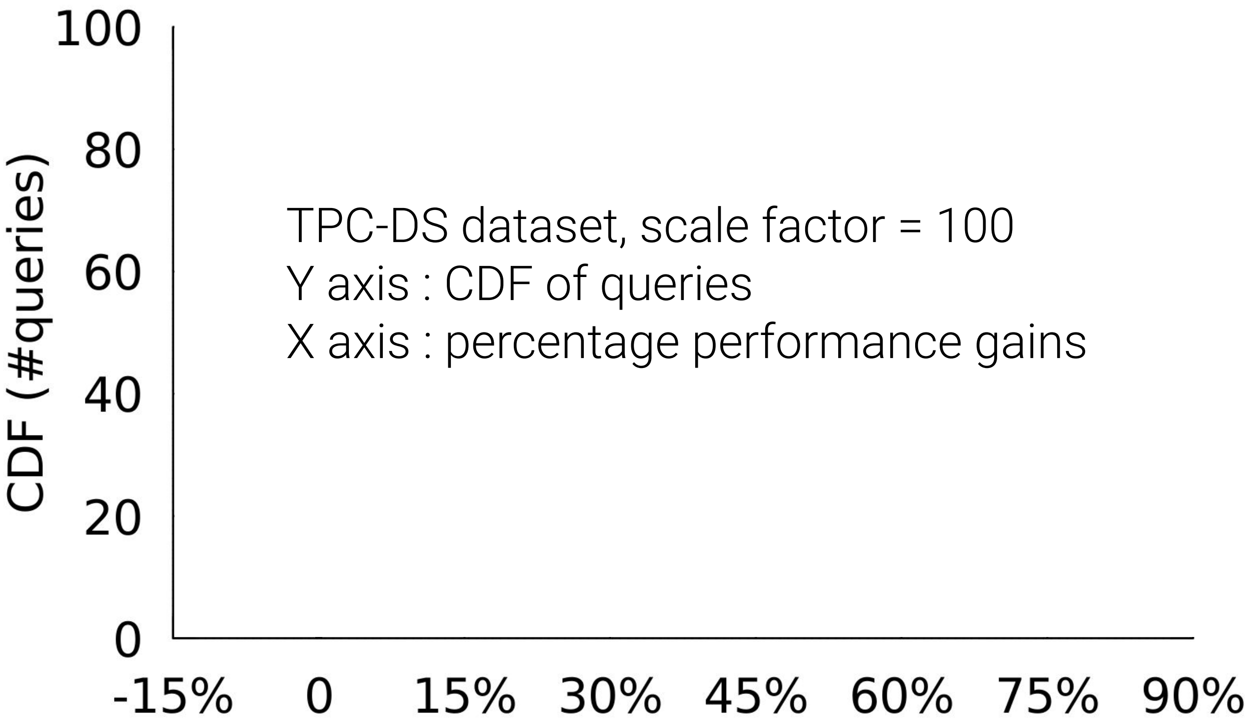
- Does Albis deliver better performance for micro-benchmarks?
- Does micro-benchmark performance translate to better workload performance?
- What is the performance and space trade-off in Albis?

Microbenchmark Performance - Revised



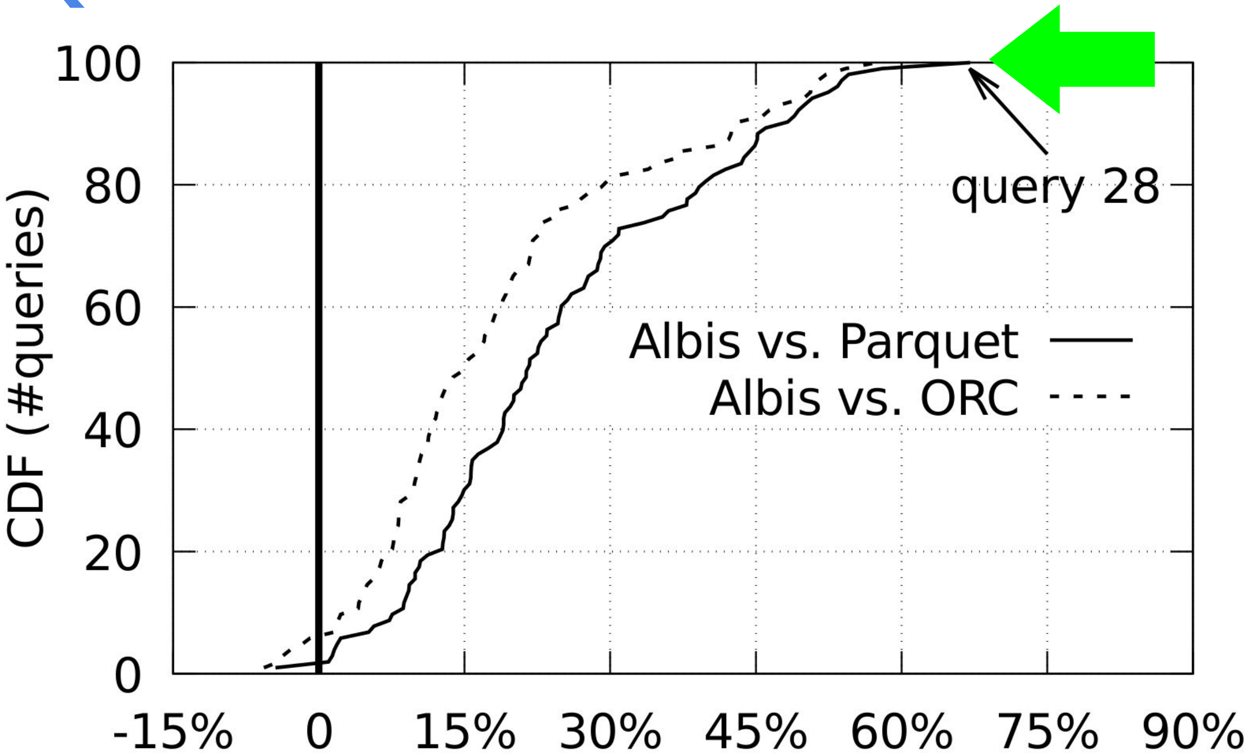
Albis delivers 1.9 - 21.3x performance improvements over other formats

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



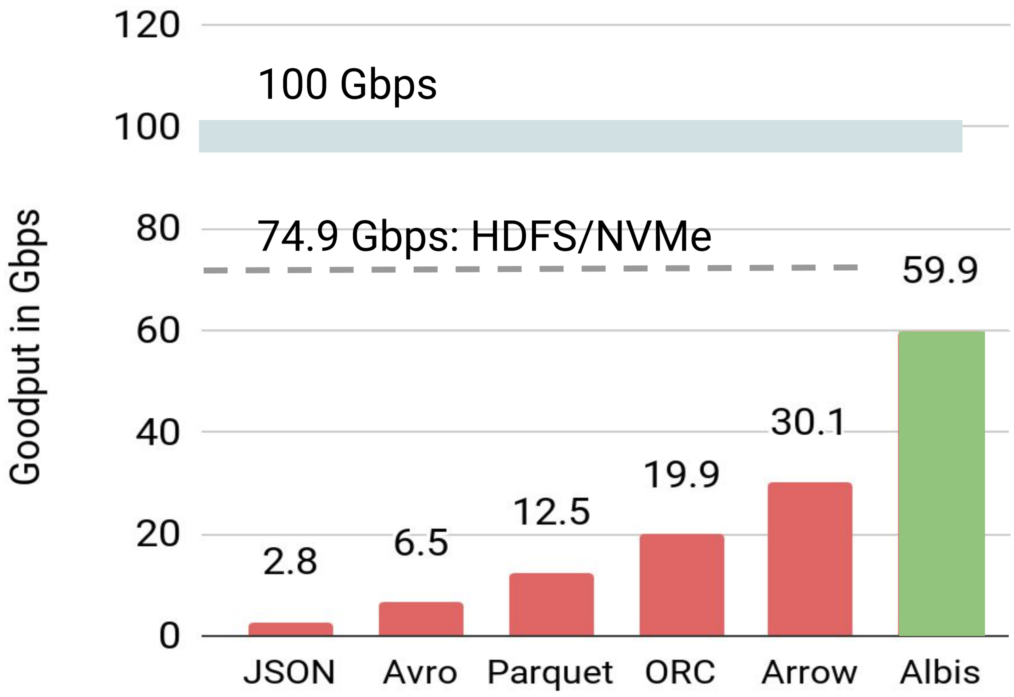
Albis delivers up to 3x performance gains for TPC-DS queries

Space vs. Performance Trade-off

	None	Snappy	Gzip	zlib
Parquet	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

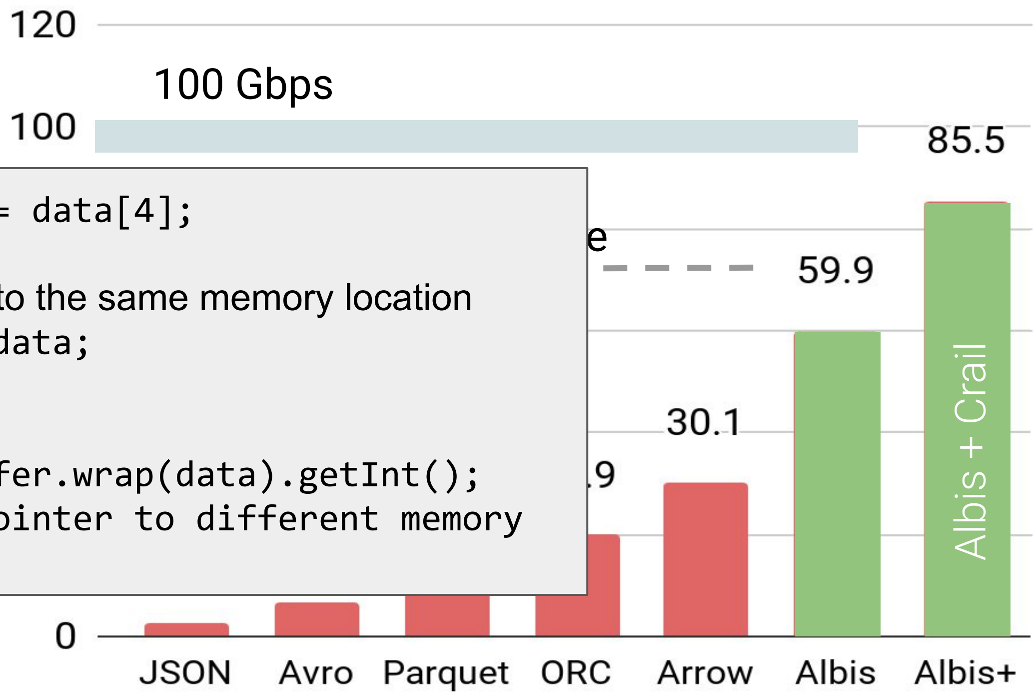
Albis inflates data by 1.3 - 2.7x, but gives 3.4 - 7.2x performance gains

Microbenchmark Performance - Revised



What would it take to deliver 100 Gbps?

Microbenchmark Performance - Revised

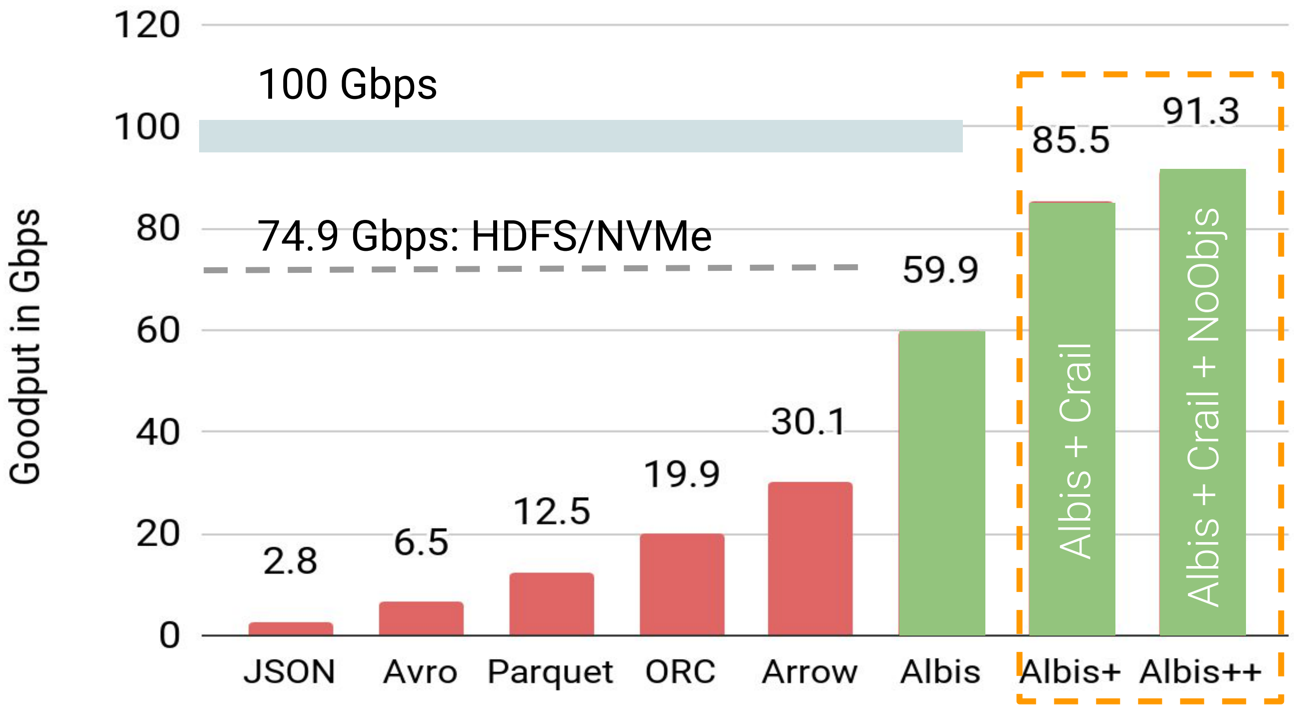



```
byte [] raw_data = data[4];

// In C/C++ , pointing to the same memory location
int val = (int*) data;

// in Java/Scala
int val = ByteBuffer.wrap(data).getInt();
// val and data pointer to different memory
// locations
```

JVM object overheads

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

Microbenchmark Performance - Revised



Albis can deliver performance within 10% of hardware

Think about

When does Albis-type data storage format does not make sense?

1. CPU is fast enough to compute (compress, encode, materialize objects) faster than I/O device bandwidth
   a. Is CPU getting faster? Are I/O devices getting faster?

2. Is space vs. performance trade-off acceptable?
   a. Not all data is equally performance sensitive
   b. Not all data is hot - cold data needs to be compressed and stored efficiently

3. Anything else? Albis is only evaluated in the Cloud/HDFS/Crail
   a. Building Albis on OCSSDs would be an interesting exercise

From this Lecture You Should Know

1. What is temporary data
2. Why does temporary data needs special treatment
    a. In the critical path
    b. Large size distribution
    c. No fault tolerance (can be supported by the framework itself)
3. How does modern networking (RDMA) and storage (NVMe/NVMeF) help to build fast Crail-type system
    a. What does control and data path split means
    b. What does unification of abstractions in the NodeKernel model mean
4. What is Albis and how does its design leverage modern networking and storage hardware
    a. Reduce CPU involvement - simple format and easy layout on file system

Further Reading

1. Disaggregating Persistent Memory and Controlling Them Remotely: An Exploration of Passive Disaggregated Key-Value Stores, USENIX ATC 2020.
2. Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled distributed persistent memory file system. In Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). USENIX Association, USA, 773–785.
3. Animesh Trivedi, Patrick Stuedi, Jonas Pfefferle, Adrian Schuepbach, and Bernard Metzler. 2018. Albis: high-performance file format for big data systems. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '18). USENIX Association, USA, 615–629.
4. Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Ana Klimovic, Adrian Schuepbach, and Bernard Metzler. 2019. Unification of temporary storage in the nodekernel architecture. In Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19). USENIX Association, USA, 767–781.
5. https://www.alluxio.io/
6. RAMCloud Project, https://ramcloud.atlassian.net/wiki/spaces/RAM/overview
7. V. Srinivasan, Brian Bulkowski, Wei-Ling Chu, Sunil Sayyaparaju, Andrew Gooding, Rajkumar Iyer, Ashish Shinde, and Thomas Lopatic. Aerospike: Architecture of a real-time operational dbms.Proc. VLDB Endow.,9(13):1389–1400, September 2016.
8. Shuotao Xu, Sungjin Lee, Sang-Woo Jun, Ming Liu,Jamey Hicks, and Arvind. Bluecache: A scalable distributed flash-based key-value store. Proc. VLDB Endow., 10(4):301–312, November 2016