Centrum Wiskunde & Informatica

Vortex

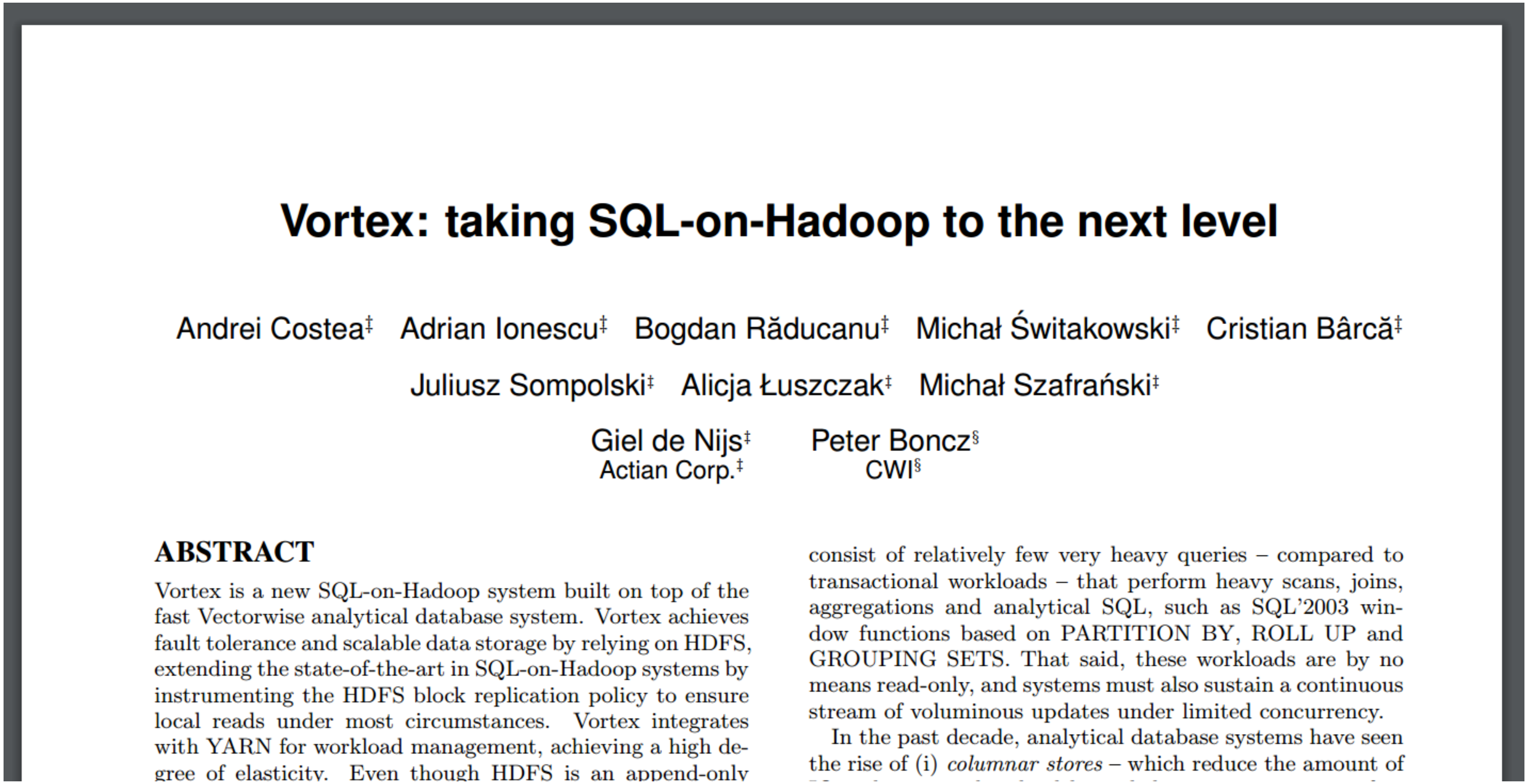
Vectorwise-on-Hadoop

Peter Boncz

TUM workshop, February 11, 2016

New SIGMOD Paper:

- www.cwi.nl/~boncz/vortex-sigmod2016.pdf



Vortex: taking SQL-on-Hadoop to the next level

Andrei Costea[‡] Adrian Ionescu[‡] Bogdan Răducanu[‡] Michał Świtakowski[‡] Cristian Bârcă[‡]

Juliusz Sompolski[‡] Alicja Łuszczak[‡] Michał Szafrański[‡]

Giel de Nijs[‡] Peter Boncz[§]

Actian Corp.[‡] CWI[§]

ABSTRACT

Vortex is a new SQL-on-Hadoop system built on top of the fast Vectorwise analytical database system. Vortex achieves fault tolerance and scalable data storage by relying on HDFS, extending the state-of-the-art in SQL-on-Hadoop systems by instrumenting the HDFS block replication policy to ensure local reads under most circumstances. Vortex integrates with YARN for workload management, achieving a high degree of elasticity. Even though HDFS is an append-only

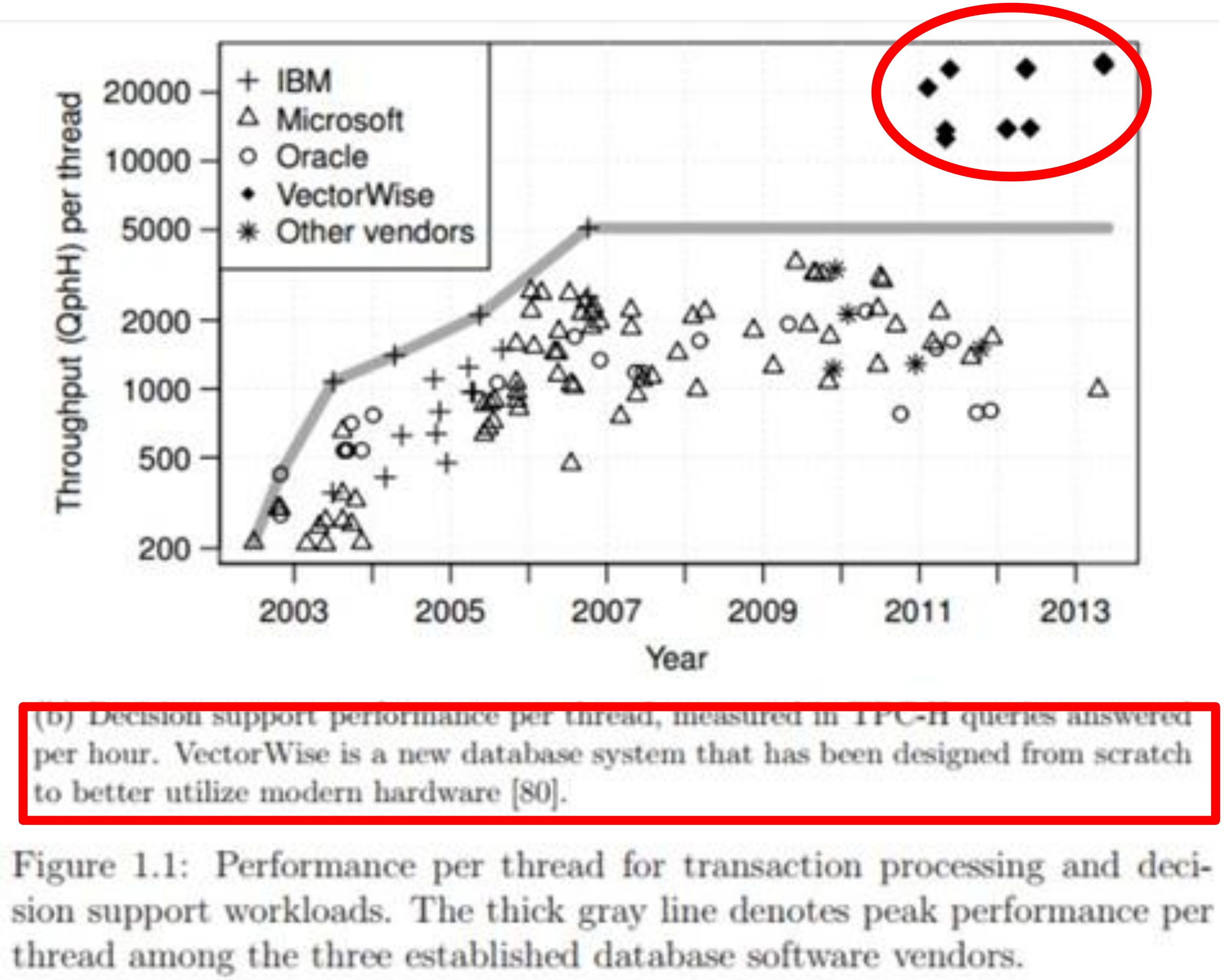
consist of relatively few very heavy queries – compared to transactional workloads – that perform heavy scans, joins, aggregations and analytical SQL, such as SQL'2003 window functions based on PARTITION BY, ROLL UP and GROUPING SETS. That said, these workloads are by no means read-only, and systems must also sustain a continuous stream of voluminous updates under limited concurrency.

In the past decade, analytical database systems have seen the rise of (i) *columnar stores* – which reduce the amount of

Vortex origin: Vectorwise

- 2005: invented as MonetDB/X100
  - Vectorized query processing
    - Reducing interpretation overhead, exploiting SIMD — cidr05
    - vectorized decompression (formatsPFOR,PDELTA,PDICT) — icde06
    - Cooperative Scans & Predictive Buffer Manager — vldb07&12
    - Mixing NSM and DSM in the query pipeline — damon08
    - Positional Delta Trees – for updates — sigmod10
    - Compilation &&-|| Vectorization — damon11
    - Run-time adaptation: “micro-adaptivity” — sigmod13
    - Advanced Table Clustering (➔ new BDCC paper) — vldbj16
    - Vectorized Scans in Hyper — sigmod16
- 2008: spin-off company
- 2010: product released, top TPC-H benchmarks

PhD thesis of Spyros Blanas (2013)

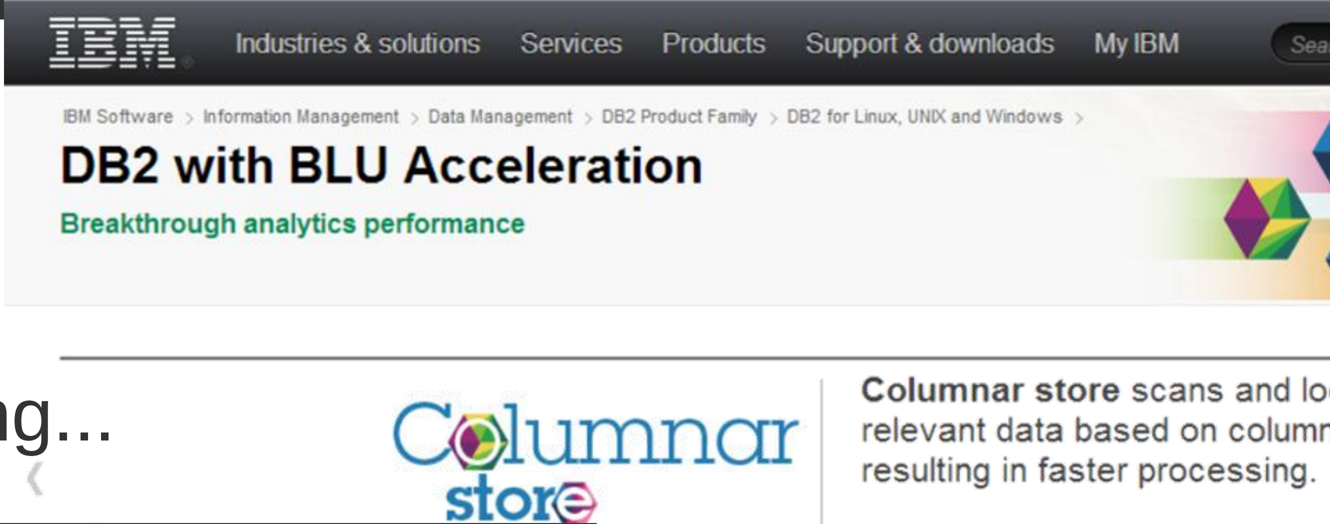


(b) Decision support performance per thread, measured in TPC-H queries answered per hour. VectorWise is a new database system that has been designed from scratch to better utilize modern hardware [80].

Figure 1.1: Performance per thread for transaction processing and decision support workloads. The thick gray line denotes peak performance per thread among the three established database software vendors.

Actian

- the relational industry is trying to adopt vector processing...



IBM Software > Information Management > Data Management > DB2 Product Family > DB2 for Linux, UNIX and Windows >

DB2 with BLU Acceleration

Breakthrough analytics performance

Columnar store scans and lo
relevant data based on column
resulting in faster processing.

BLAZING-FAST PERFORMANCE:

A Technical Best Practices Tour with ColumnStore Index

Susan Price
Senior Program Manager

Optimizing Transaction and Query Performance

Row Format Databases versus Column Format Databases

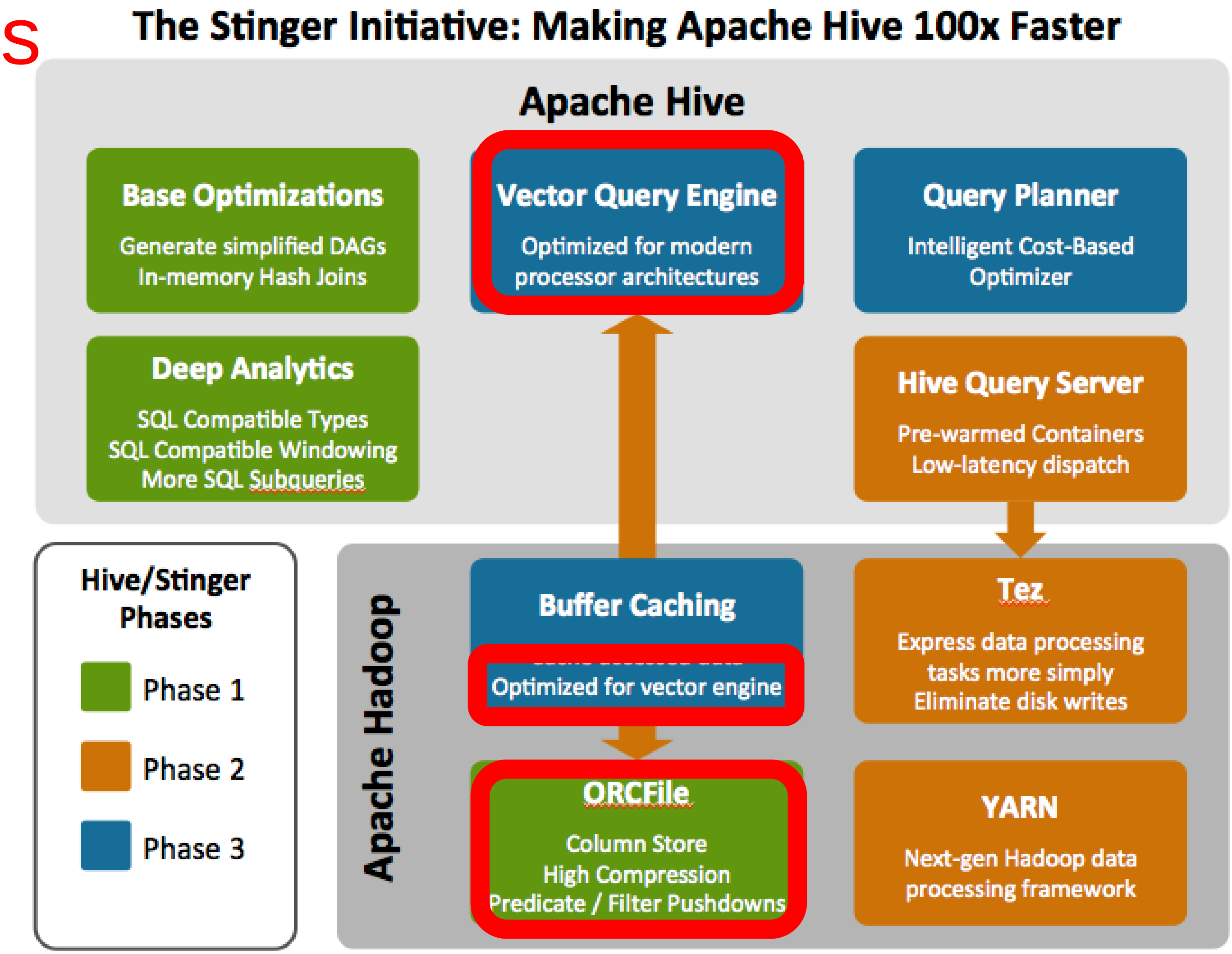
- Transactions run faster on row format
  - Insert or query a sales order
  - Fast processing few rows, many columns
- Analytics run faster on column format
  - Report on sales totals by state
  - Fast accessing few columns, many rows

Oracle 12c: Stores Data in Both Formats Simultaneously

SAP HANA® Solution

Redefines In-memory Computing

Hive gets it too!



The Stinger Initiative: Making Apache Hive 100x Faster

Apache Hive

Base Optimizations
Generate simplified DAGs
In-memory Hash Joins

Vector Query Engine
Optimized for modern processor architectures

Query Planner
Intelligent Cost-Based Optimizer

Deep Analytics
SQL Compatible Types
SQL Compatible Windowing
More SQL Subqueries

Hive Query Server
Pre-warmed Containers
Low-latency dispatch

Apache Hadoop

Buffer Caching
Optimized for vector engine

Tez
Express data processing tasks more simply
Eliminate disk writes

ORCFile
Column Store
High Compression
Predicate / Filter Pushdowns

YARN
Next-gen Hadoop data processing framework

Hive/Stinger Phases

🟩	Phase 1
🟧	Phase 2
🟦	Phase 3

“SQL on Hadoop”

- Big Data processing pipelines on Hadoop
  - Unstructured ➔ Structured
    - Unstructured: Data Mining, Pattern Matching (MapReduce)
    - Structured: Cleaner data, bulk loads into warehouse
  - Do we have to buy/manage **two** clusters??
    1. Hadoop/MapReduce
    2. MPP SQL warehouse

The case for SQL on Hadoop:

- **Reduced hardware cost** (1 cluster)
- **Agile**: no more data copying data between Hadoop and SQL
- Broaden access to Hadoop data through a **wealth of SQL apps**
- **Standardize cluster admin skills** on Hadoop (human resources)

Inroducing Vortex: Vector-on-Haddo

Key Features

- compressed vector data formats work **natively on HDFS**

HDFS (append-only) and compressed columnar storage are friends

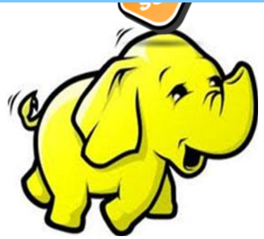
Vectorized, leading single-server TPC-H for years

Relies solely on Hadoop for system administration.

Partitioned table support and fully parallel loading

Incl. access control, analytic/window functions, complete SQL APIs

Enhanced with advanced distributed parallel execution for scale-up/out



Vortex

Hadoop Features :

- Automatic **HDFS block placement**

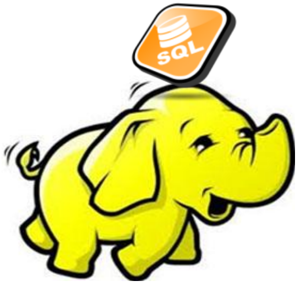
Leveraging replication, always HDFS shortcut reads also after nodes fail.

Text, Parquet, ORCfile

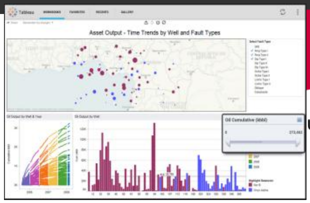
Thanks to special delta update structure (Positional Delta Trees)

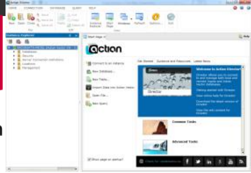
Co-existence of MapReduce and DBMS, avoiding stragglers

Workload-driven scaling up&down in 40 steps from 2.5% to 100%



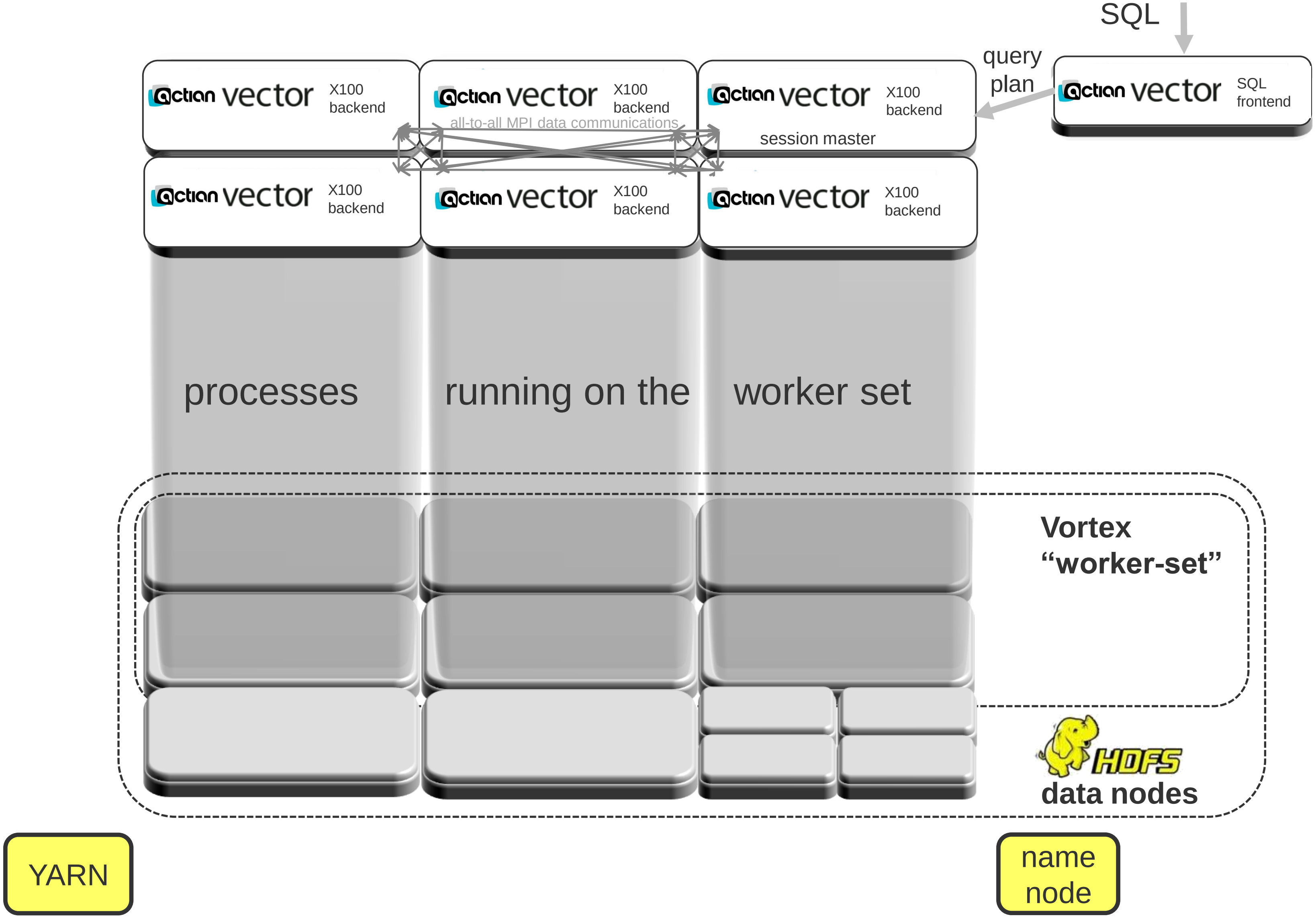
Vortex Architecture





um

Actian Director
for Management



SQL

query
plan

Actian vector SQL frontend

Actian vector X100 backend

Actian vector X100 backend

Actian vector X100 backend

session master

all-to-all MPI data communications

Actian vector X100 backend

Actian vector X100 backend

Actian vector X100 backend

processes running on the worker set

**Vortex
“worker-set”**

HDFS

data nodes

YARN

name
node

Storage

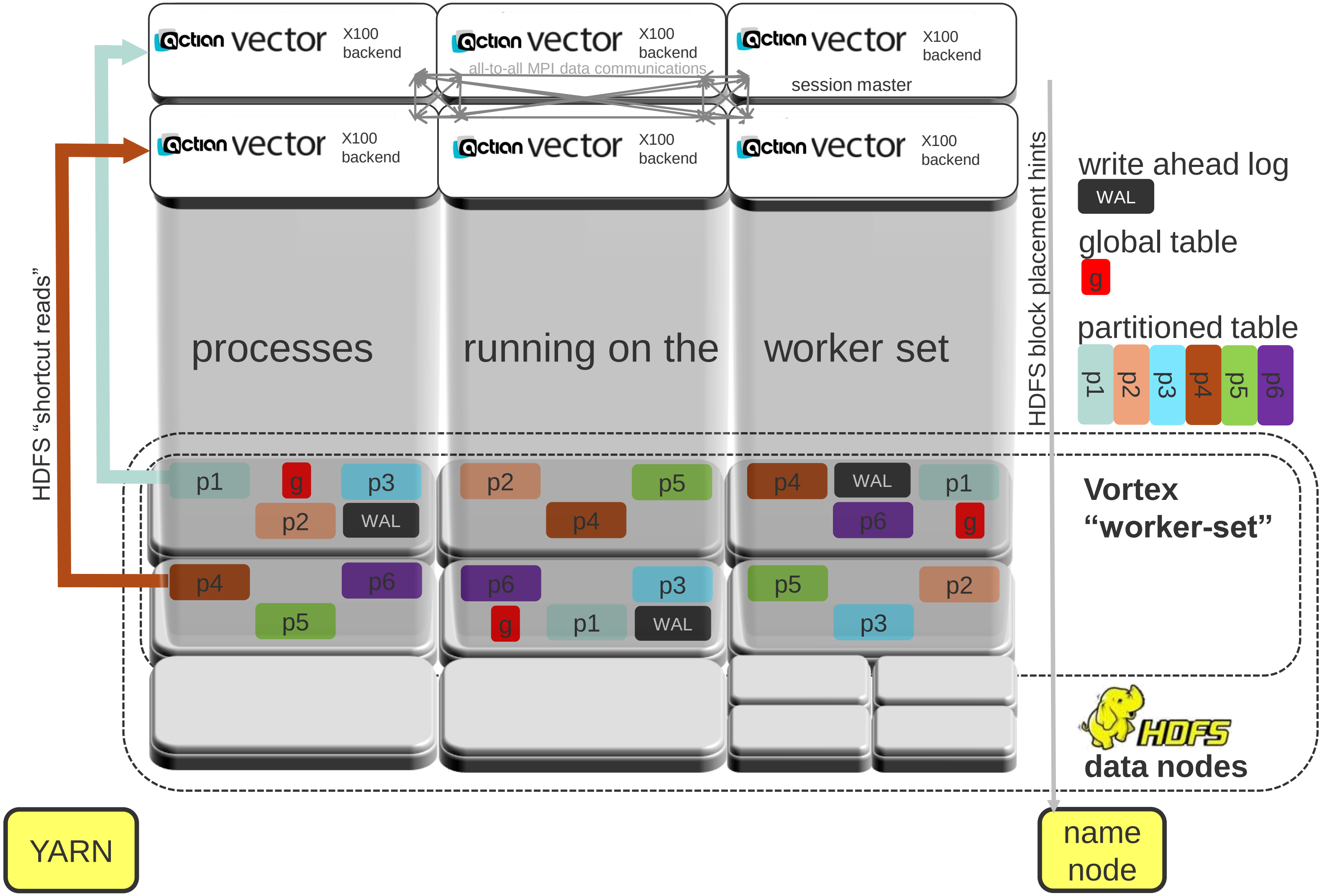
Data Format

- Vector native compressed data formats
- Fixed-size blocks, one table per block-file
- Horizontal splits for garbage collection; tail-file to stage small appends
- HDFS block placement: we decide were the replicas are
- Tables are either hash-partitioned or global (i.e. non-partitioned)

Global File System

- All I/O is through HDFS
  - Achieved in an append-only file system
  - Any worker can read any table partition
- Responsibilities for handling partitions is decided at session start
  - Optimization algorithm assigns partitions to nodes that have the file local
  - 100% HDFS “shortcut reads”, also when the node that wrote the partition is down

Vortex Architecture



Minimizing Data Transfer

Storage

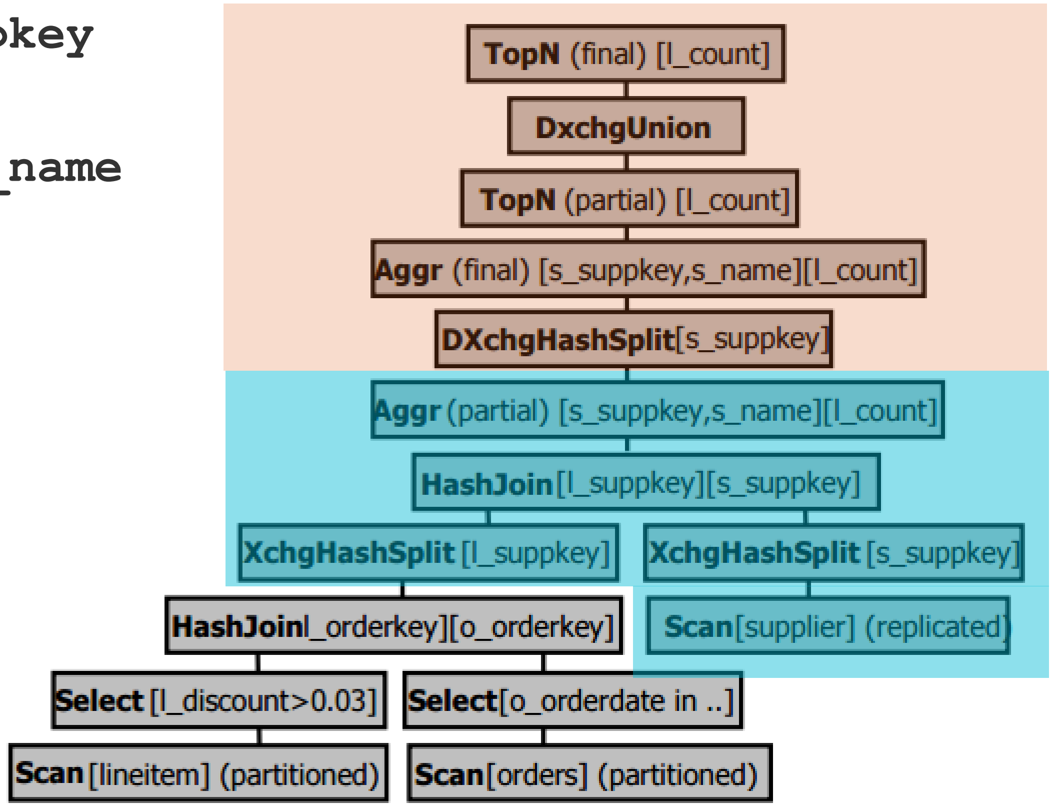
- Co-located partitions (local partitioned hash-joins)
- Replicated tables (local shared-HashTable hash-joins)
- Co-partitioned clustered indexes (local merge-joins)
- MinMax indexes for predicate pushdown (correlates over merge-joins)

Parallel Cost Model

- Distributed joins, distributed query optimizer considers:
  - Both key-partitioned and shared (broadcast) HashJoin
  - Local broadcast HashJoin for replicated tables
- Distributed GroupBy, distributed query optimizer considers:
  - Both key-partitioned and global re-aggregated GroupBy
  - Local early aggregation followed by partitioned aggregation

Example Query Plan

```
SELECT FIRST 10 s_suppkey, s_name, count(*) as l_count
FROM lineitem, orders, supplier
WHERE l_orderkey=o_orderkey
  AND o_orderdate BETWEEN ’1995-03-05’ AND ’1997-03-05’
  AND l_suppkey=s_suppkey
  AND l_discount>0.03
GROUP BY s_suppkey, s_name
ORDER BY l_count
```



TopN (final) [l_count]

DxchgUnion

TopN (partial) [l_count]

Aggr (final) [s_suppkey,s_name][l_count]

DXchgHashSplit[s_suppkey]

Aggr (partial) [s_suppkey,s_name][l_count]

HashJoin[l_suppkey][s_suppkey]

XchgHashSplit [l_suppkey]

XchgHashSplit [s_suppkey]

HashJoinl_orderkey][o_orderkey]

Scan[supplier] (replicated)

Select [l_discount>0.03]

Select[o_orderdate in ..]

Scan [lineitem] (partitioned)

Scan[orders] (partitioned)

Resource Mgmt

YARN integration

- Ask YARN which nodes are less busy, when enlarging the worker set
- Inform YARN of our usage (CPU, memory) to prevent overload
- Placeholder processes to decrease and increase YARN resources

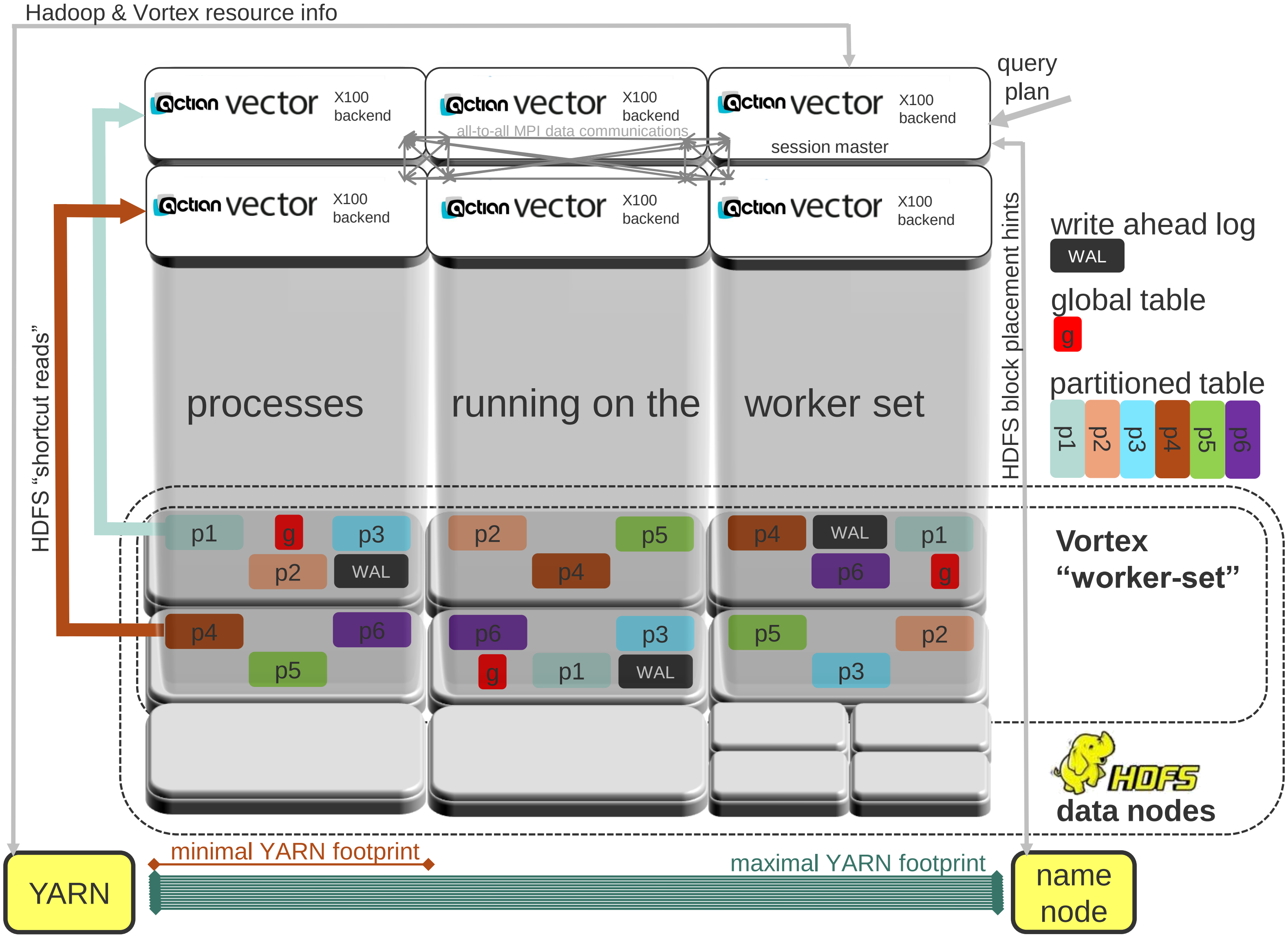
Workload management

- Workload monitoring to gradually determine Hadoop footprint
- Choose (# cores, RAM) for each query, given the current footprint
- Choose to involve all or just the minimal subset of workers

Elasticity

- Scale down to minimal subset of nodes, one core each
- Scale up to all nodes, all cores

Vortex Architecture



Data Ingestion

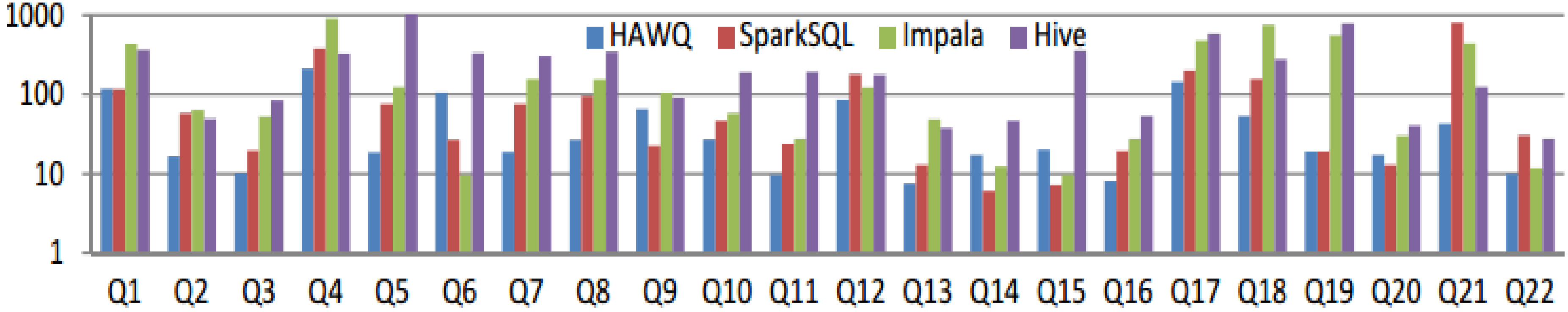
- Connectivity
  - Fast Parallel Loader, executes in parallel on all worker nodes
  - Spark Integration
    - to read and write Hadoop Formats; push computation into Spark

Updates (DML)

- Support for Insert, Modify, Delete, Upsert
  - Modify, Deleted, Upsert use Positional Delta Trees (PDTs)
  - Combination of distributed WAL and master WAL
  - 2PC coordinated by the session master
- Partitioned Tables partition DML to all nodes in worker set
  - Updates go to distributed WAL(s) – unless transaction is small
- Replicated Tables execute DML on the session master
  - Session master broadcasts all PDT changes to all worker nodes

Vortex: contributions

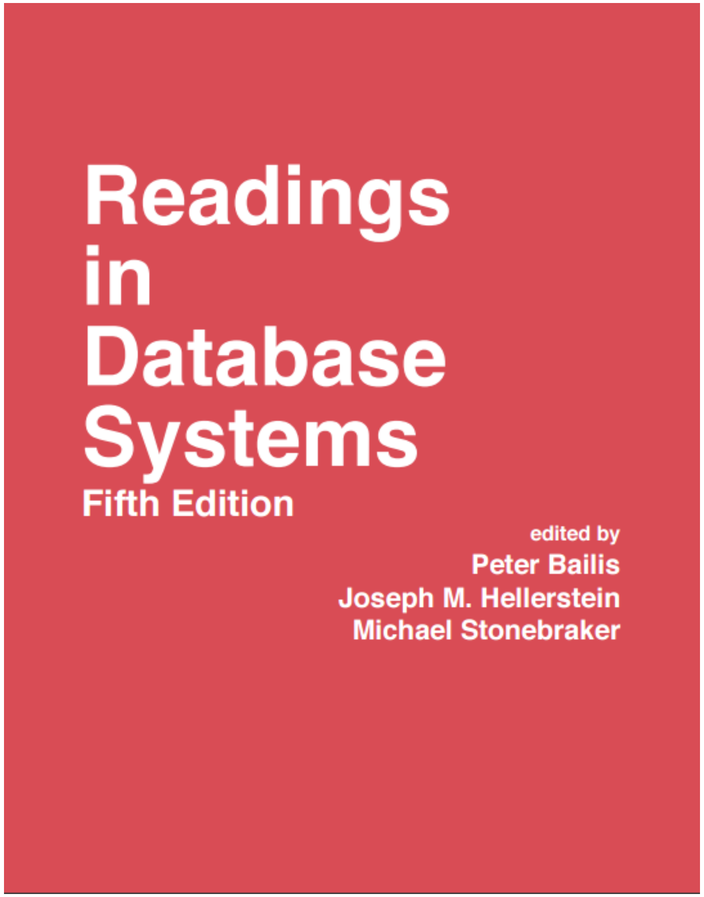
- Performance
  - TPC-H 1000GB – 10 node Hadoop cluster (16-core, 256GB RAM, 24 disks)
  - How many times faster is Vortex, compared to..? (well-tuned, same everything)



Vortex	1.34	1.29	3.15	0.18	1.94	0.19	2.37	1.8	11.77	1.21	1.28	0.37	3.69	1.13	1.56	1.73	1.21	1.63	1.29	2.47	1.99	2.96

(Vortex numbers = latency in seconds)

A New Red Book



Readings in Database Systems

Fifth Edition

edited by
Peter Bailis
Joseph M. Hellerstein
Michael Stonebraker

*"The advantages of a column executor are persuasively discussed in [2], although it is "down in the weeds" and **hard to read**."*

References:

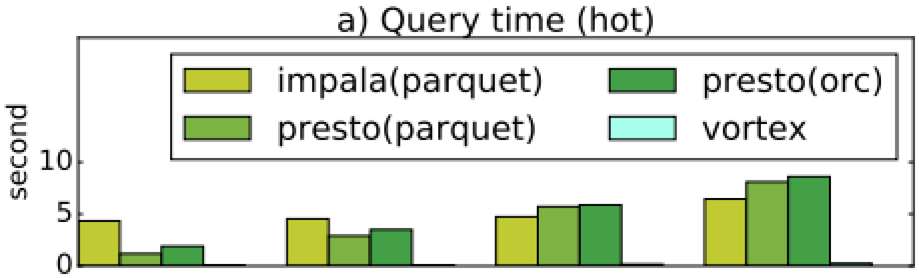
[1] Batory, D.S. On searching transposed files. *ACM Transactions on Database Systems (TODS)*. 4, 4 (Dec. 1979).

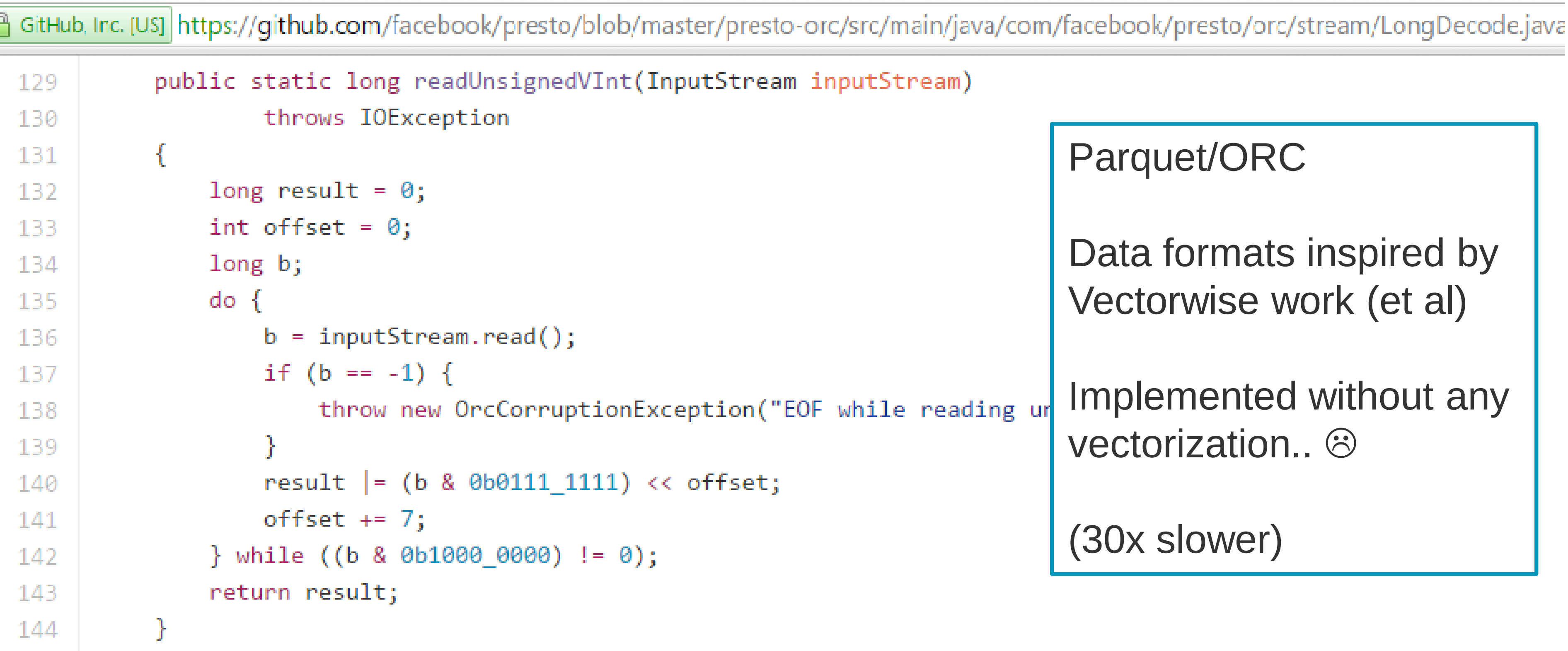
[2] Boncz, P.A., Zukowski, M. and Nes, N. MonetDB/X100: Hyper-pipelining query execution.*CIDR*, 2005.

Vectorwise: scan performance

- Fast vectorized decompression

a) Query time (hot)





https://github.com/facebook/presto/blob/master/presto-orc/src/main/java/com/facebook/presto/orc/stream/LongDecode.java

```java
    public static long readUnsignedVInt(InputStream inputStream)
            throws IOException
    {
        long result = 0;
        int offset = 0;
        long b;
        do {
            b = inputStream.read();
            if (b == -1) {
                throw new OrcCorruptionException("EOF while reading un
            }
            result |= (b & 0b0111_1111) << offset;
            offset += 7;
        } while ((b & 0b1000_0000) != 0);
        return result;
    }
```

Parquet/ORC

Data formats inspired by Vectorwise work (et al)

Implemented without any vectorization.. ☹

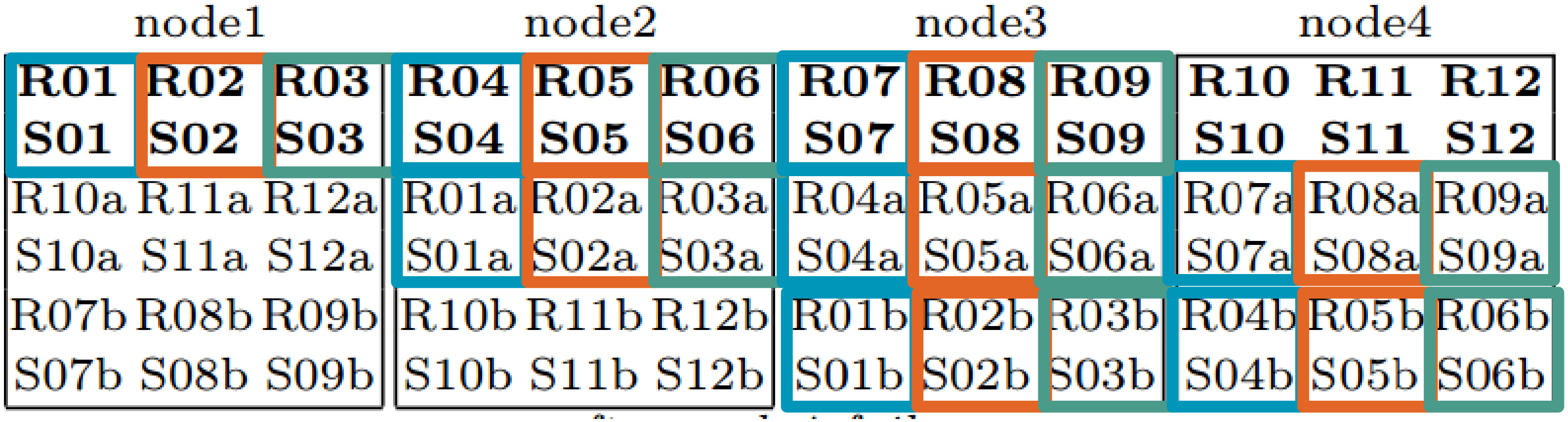
(30x slower)

Vortex: contributions

- Performance
- HDFS locality

HDFS locality

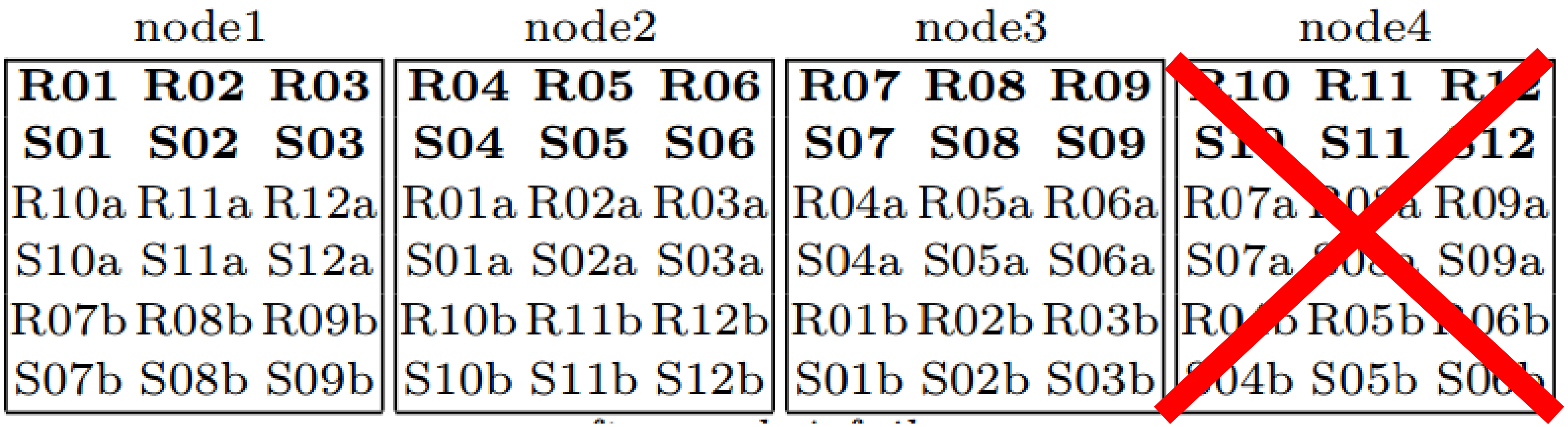
- Partitioned tables R,S (12 partitions)
- Co-located on S.FK➔R.PK
  - local joins
- 3-way HDFS replication
- One node (of 3) is “responsible” (bold)
  - handles updates to that partition – and most queries



node1			node2			node3			node4		
R01 **S01**	**R02** **S02**	**R03** **S03**	**R04** **S04**	**R05** **S05**	**R06** **S06**	**R07** **S07**	**R08** **S08**	**R09** **S09**	**R10** **S10**	**R11** **S11**	**R12** **S12**
R10a S10a	R11a S11a	R12a S12a	R01a S01a	R02a S02a	R03a S03a	R04a S04a	R05a S05a	R06a S06a	R07a S07a	R08a S08a	R09a S09a
R07b S07b	R08b S08b	R09b S09b	R10b S10b	R11b S11b	R12b S12b	R01b S01b	R02b S02b	R03b S03b	R04b S04b	R05b S05b	R06b S06b

HDFS locality

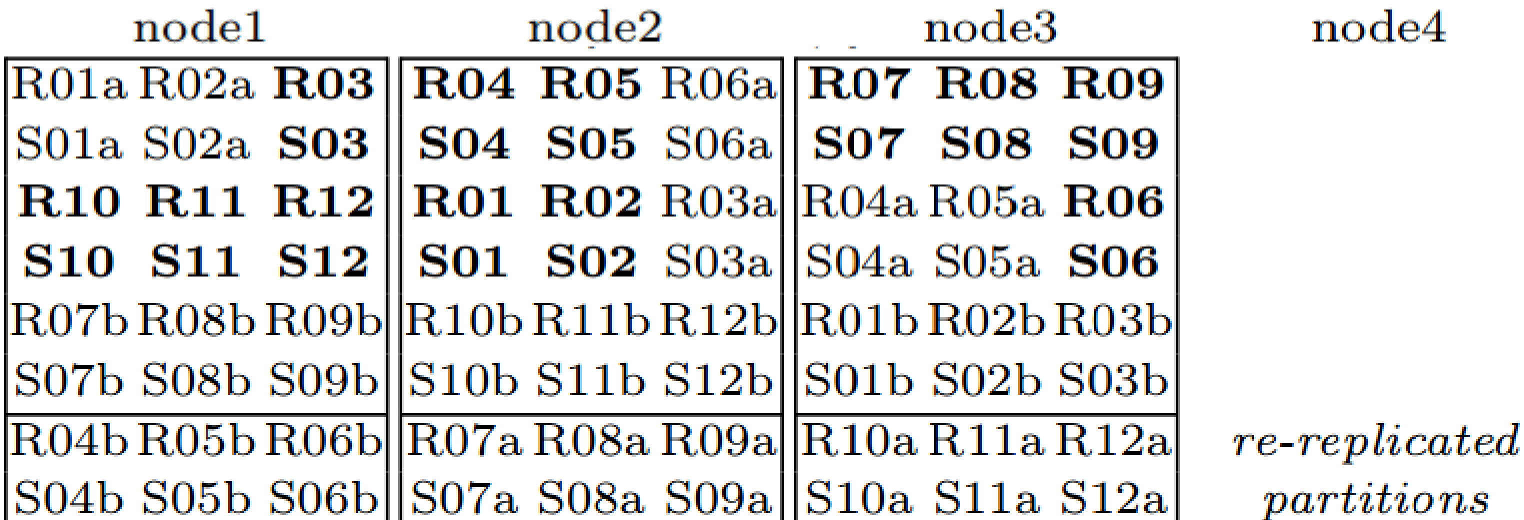
- Partitioned tables R,S (12 partitions)
- Co-located on S.FK➔R.PK
  - local joins
- 3-way HDFS replication
- One node (of 3) is “responsible” (bold)
  - handles updates to that partition – and most queries



node1	node2	node3	node4
R01 R02 R03	**R04 R05 R06**	**R07 R08 R09**	~~**R10 R11 R12**~~
S01 S02 S03	**S04 S05 S06**	**S07 S08 S09**	~~**S10 S11 S12**~~
R10a R11a R12a	R01a R02a R03a	R04a R05a R06a	~~R07a R08a R09a~~
S10a S11a S12a	S01a S02a S03a	S04a S05a S06a	~~S07a S08a S09a~~
R07b R08b R09b	R10b R11b R12b	R01b R02b R03b	~~R04b R05b R06b~~
S07b S08b S09b	S10b S11b S12b	S01b S02b S03b	~~S04b S05b S06b~~

HDFS locality

- Partitioned tables R,S (12 partitions)
- Co-located on S.FK➔R.PK
  - local joins
- 3-way HDFS replication
- One node (of 3) is “responsible” (bold)
  - handles updates to that partition – and most queries

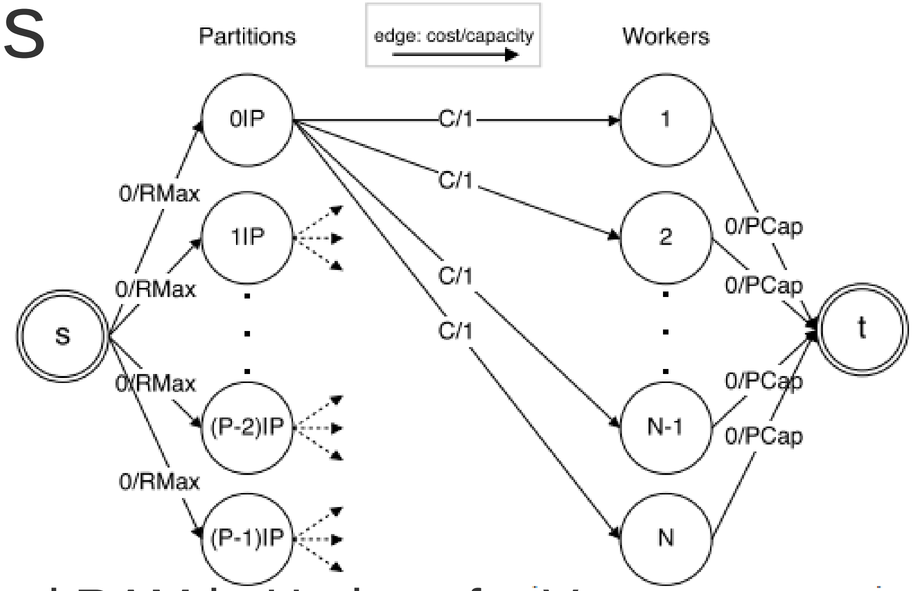


node1			node2			node3			node4
R01a	R02a	**R03**	**R04**	**R05**	R06a	**R07**	**R08**	**R09**	
S01a	S02a	**S03**	**S04**	**S05**	S06a	**S07**	**S08**	**S09**	
R10	**R11**	**R12**	**R01**	**R02**	R03a	R04a	R05a	**R06**	
S10	**S11**	**S12**	**S01**	**S02**	S03a	S04a	S05a	**S06**	
R07b	R08b	R09b	R10b	R11b	R12b	R01b	R02b	R03b	
S07b	S08b	S09b	S10b	S11b	S12b	S01b	S02b	S03b	
R04b	R05b	R06b	R07a	R08a	R09a	R10a	R11a	R12a	*re-replicated*
S04b	S05b	S06b	S07a	S08a	S09a	S10a	S11a	S12a	*partitions*

Vortex: contributions

- Performance
- HDFS locality
- YARN integration

Vortex: contributions



- Performance
- HDFS locality
- YARN integration

dbAgent

- Negotiates #nodes, #cores and RAM in Hadoop for Vortex
- Needs to work around YARN limitations (long-term tasks)
- Determines which nodes⇔data mapping
- Reacts to YARN priority scheduling
- Algorithms based on min-cost network flows

Vortex: contributions

- Performance
- HDFS locality
- YARN integration
- Updates

Positional Delta Trees (PDTs)

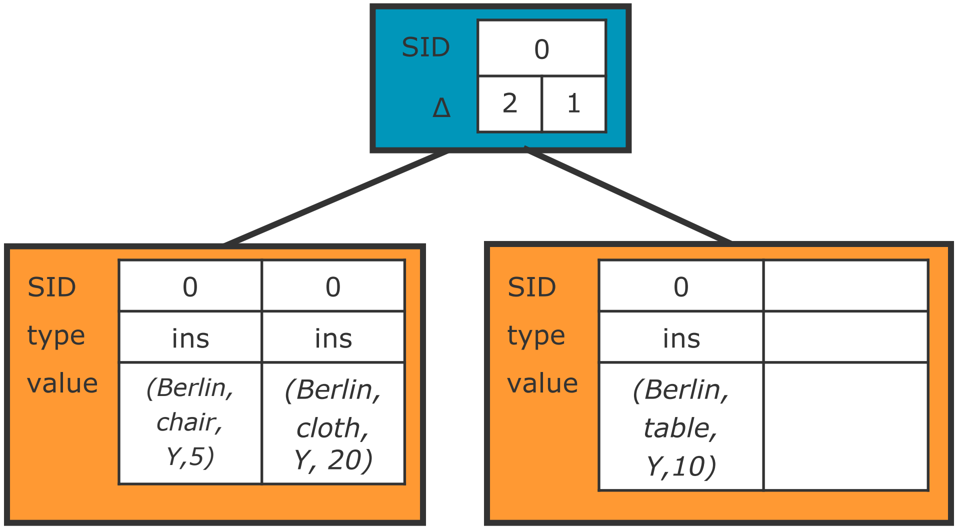
"Positional Update Handling in Column Stores" – SIGMOD 2010

SID	STORE	PROD	NEW	QTY	*RID*
0	London	chair	N	30	0
1	London	stool	N	10	1
2	London	table	N	20	2
3	Paris	rug	N	1	3
4	Paris	stool	N	5	4

TABLE_0

INSERT INTO inventory VALUES('Berlin', 'table', Y, 10)
INSERT INTO inventory VALUES('Berlin', 'cloth', Y, 20)
INSERT INTO inventory VALUES('Berlin', 'chair', Y, 5)

PDTs enable fine-grained updates on append-only data (HDFS)



Vortex: contributions

- Performance
- HDFS locality
- YARN integration
- Updates
  - HDFS is an append-only filesystem?
  - PDTs to the Rescue!
    - sigmod16: Hive slows down 40% after updates – Vortex: nothing
  - PhD thesis Heman (“updating compressed column stores” 2015)
    ➔ updating **nested** tables!
    - Nested data models (Dremel, Parquet, ORC) ⬄ relational join indexes
    - Help for co-locating tables in a distributed filesystem (HDFS)
    - Fast merge-joins

Vortex: contributions

- Performance
- HDFS locality
- YARN integration
- Updates

Vortex: in the cloud?

- Sure!
  - Amazon EMR setup available
  - USPs
    - Performance, Elasticity, SQL Maturity, Updates, Spark integration
- Work to do:
  - Current solution relies on ephimeral storage
    - Integrating S3 beyond incremental backup + DistCp
    - Ephimeral storage as automatic cache
    - Elasticity of “core instance group”
      - Can leverage Vortex control over HDFS placement

Conclusions

- Introduced Vortex: Vectorwise-on-Hadoop
  - High Performance – properly Vectorized
  - YARN integration, HDFS locality – min-cost flow optimizations
  - Updates on Nested Tables – PDTs on join indexes

- Vortex in the cloud
  - Works on EMR.
  - Interested in taking student projects (HDF-S3, elasticity)