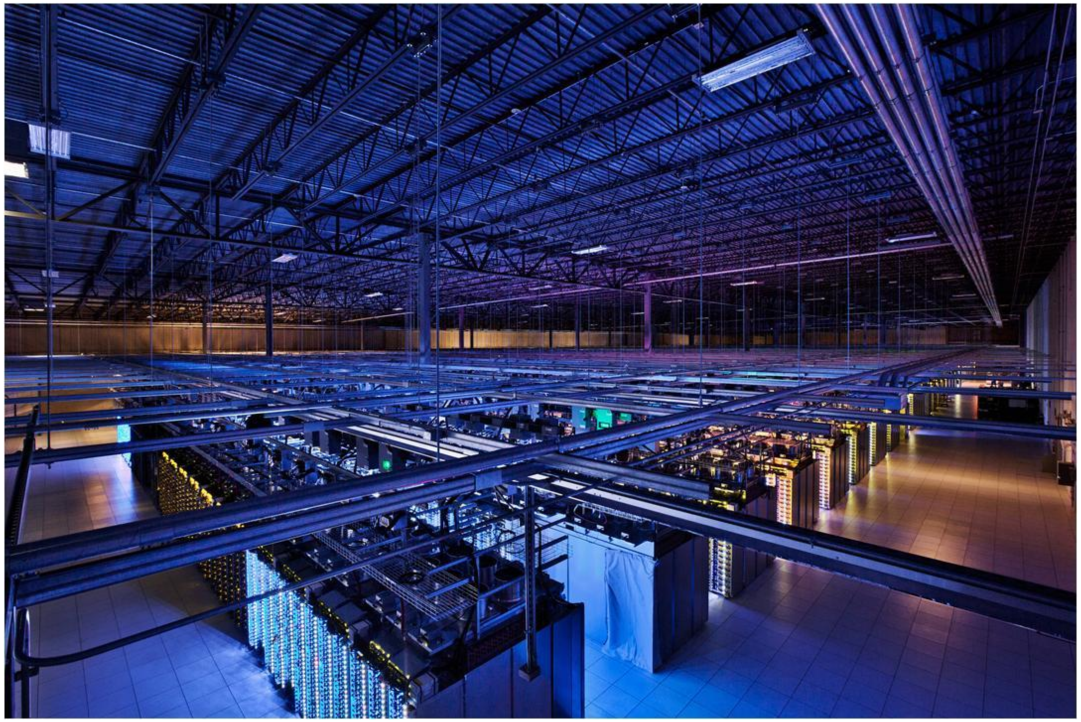
Large-Scale Data Engineering

SQL on Big Data



event.cwi.nl/lsde

THE DEBATE: DATABASE SYSTEMS VS MAPREDUCE

CWI
Centrum Wiskunde & Informatica

A major step backwards?

Michael Stonebraker
Turing Award 2015

- MapReduce is a step backward in database access
  - Schemas are good
  - Separation of the schema from the application is good
  - High-level access languages are good
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example)
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions…
- MapReduce is incompatible with DMBS tools

Known and unknown unknowns

- Databases only help if you know what questions to ask
  - “Known unknowns”
- What’s if you don’t know what you’re looking for?
  - “Unknown unknowns”

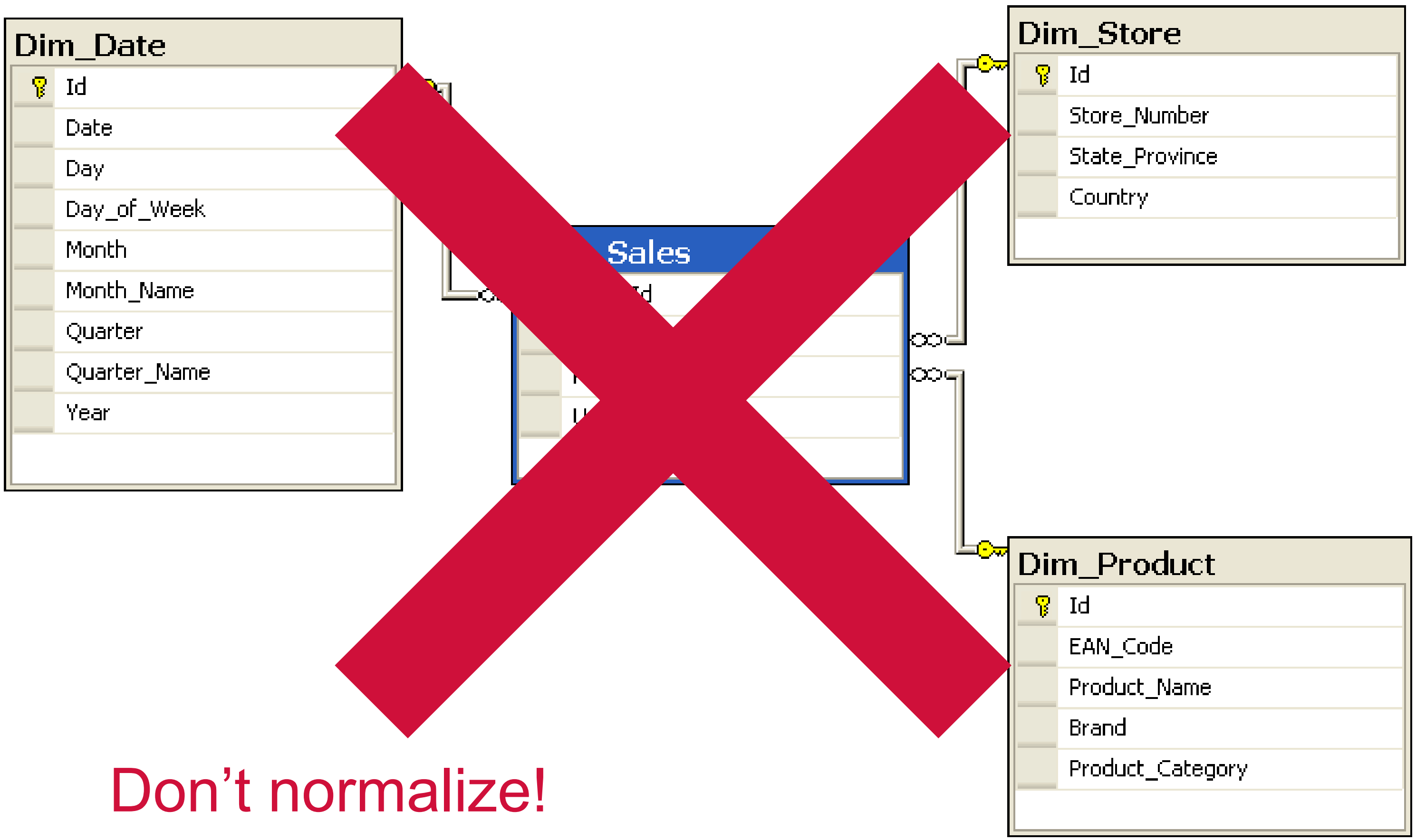
Centrum Wiskunde & Informatica

ETL: redux

- Often, with noisy datasets, ETL *is* the analysis!
- Note that ETL necessarily involves brute force data scans
- L, then E and T?

Centrum Wiskunde & Informatica

Structure of Hadoop warehouses



Don’t normalize!

Source: Wikipedia (Star Schema)

Relational databases vs. MapReduce

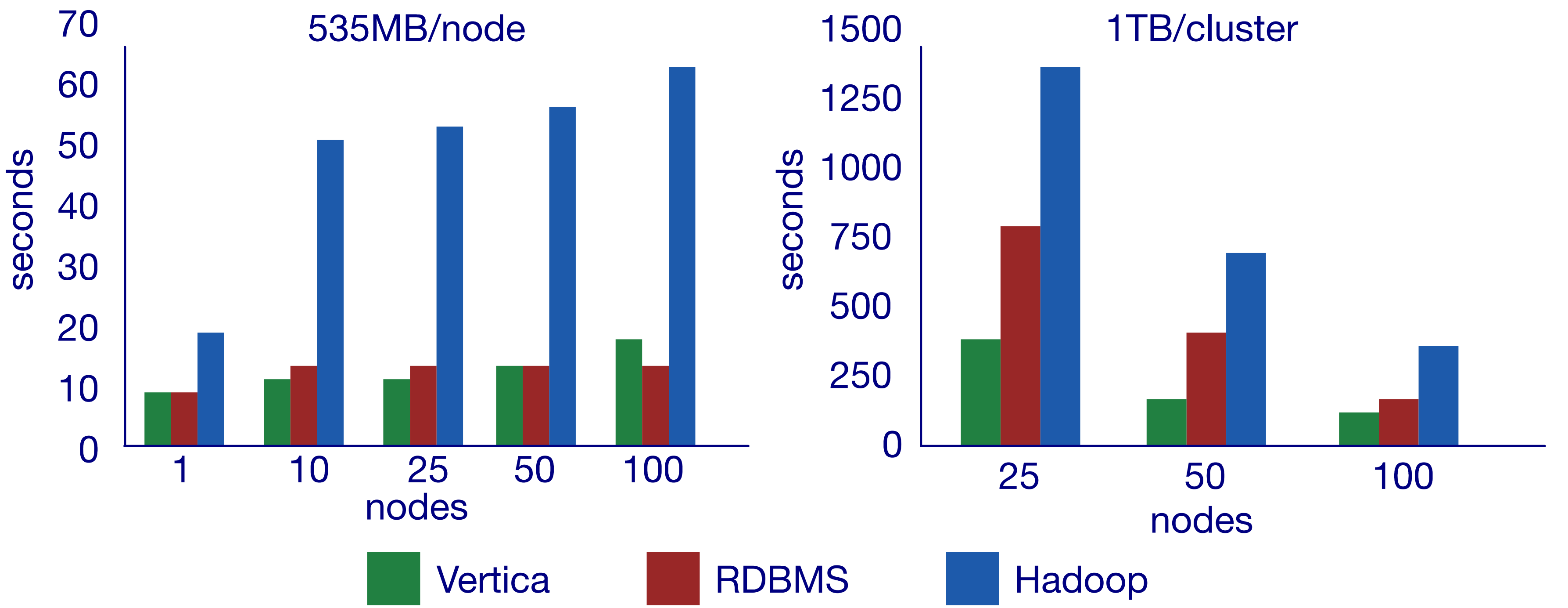
- Relational databases:
  - Multipurpose: analysis and transactions; batch and interactive
  - Data integrity via ACID transactions
  - Lots of tools in software ecosystem (for ingesting, reporting, etc.)
  - Supports SQL (and SQL integration, e.g., JDBC)
  - Automatic SQL query optimization
- MapReduce (Hadoop):
  - Designed for large clusters, fault tolerant
  - Data is accessed in “native format”
  - Supports many query languages
  - Programmers retain control over performance
  - Open source

Source: O’Reilly Blog post by Joseph Hellerstein (11/19/2008)

Philosophical differences

- Parallel relational databases
  - Schema on write
  - Failures are relatively infrequent
  - “Possessive” of data
  - Mostly proprietary
- MapReduce
  - Schema on read
  - Failures are relatively common
  - In situ data processing
  - Open source

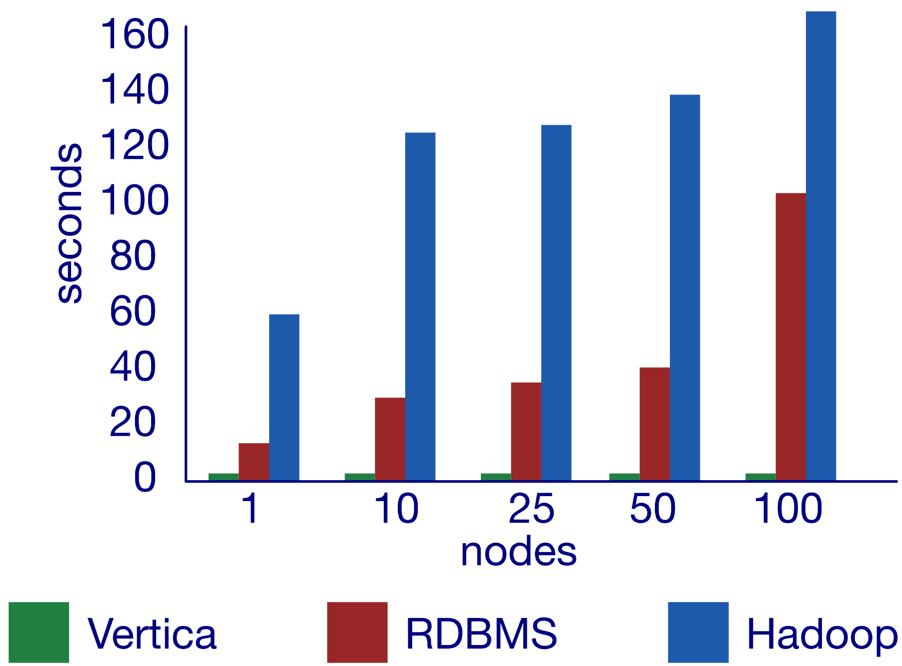
MapReduce vs. RDBMS: grep




```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

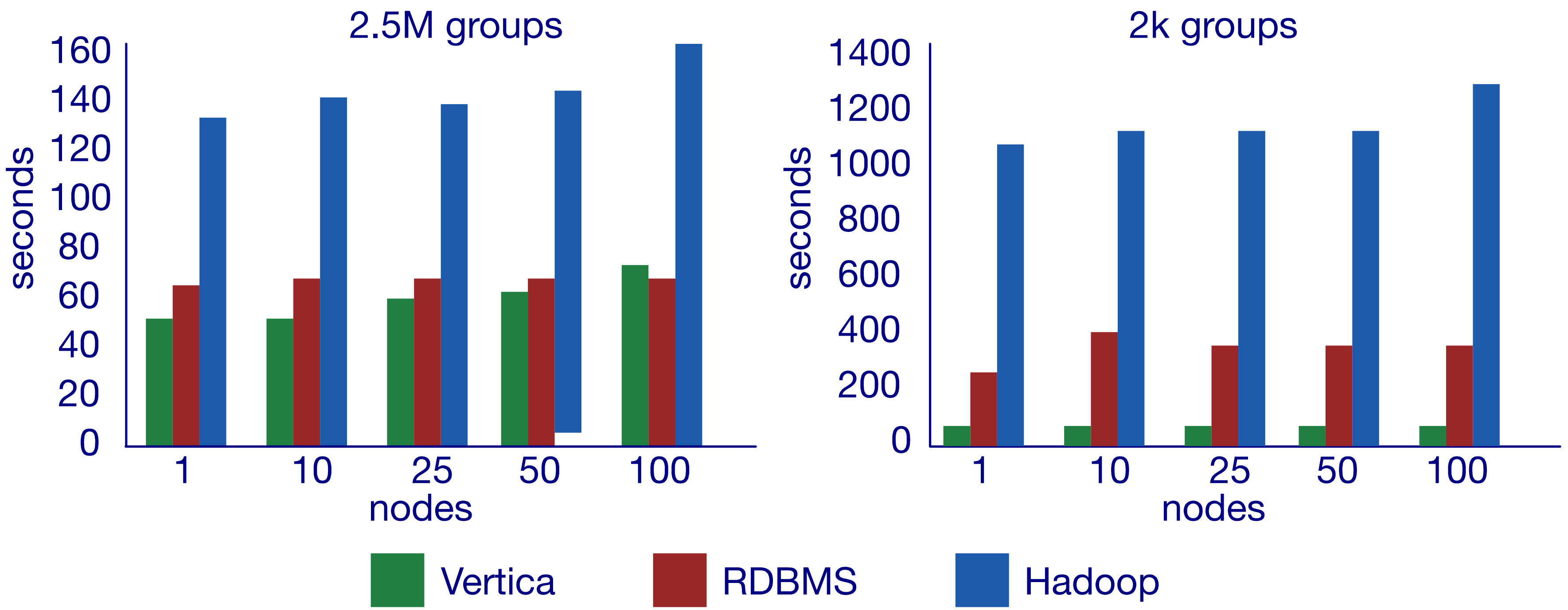
MapReduce vs. RDBMS: select




```
SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X;
```


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

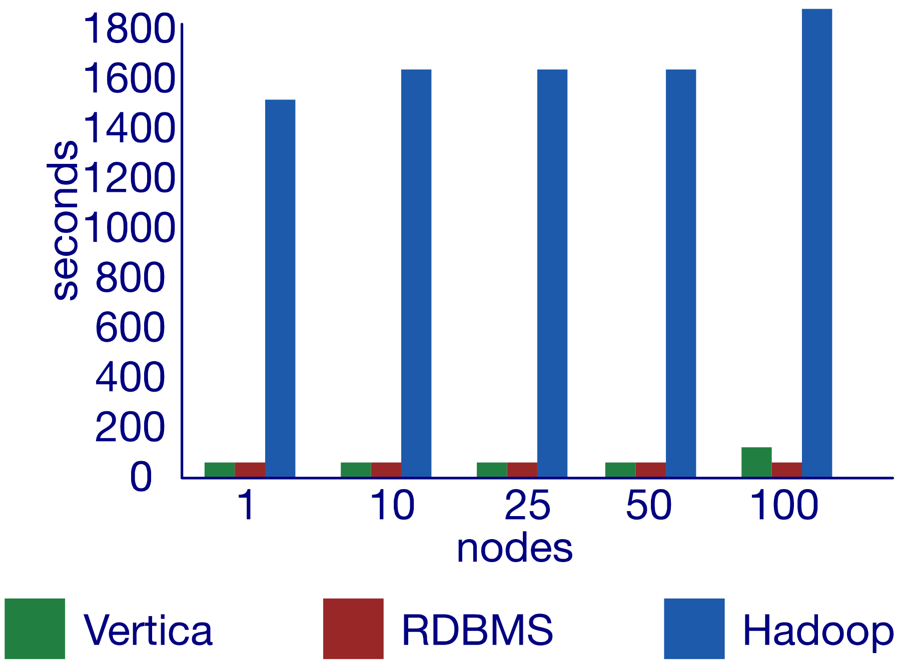
MapReduce vs. RDBMS: aggregation




```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY sourceIP;
```


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

MapReduce vs. RDBMS: join

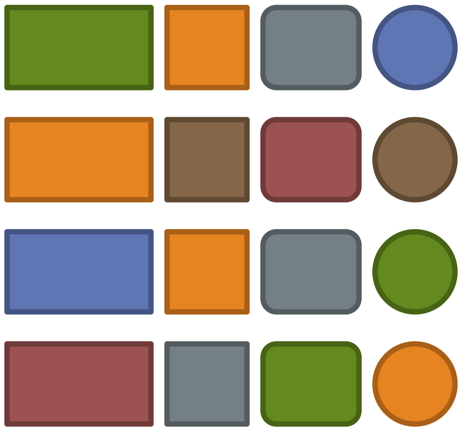


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

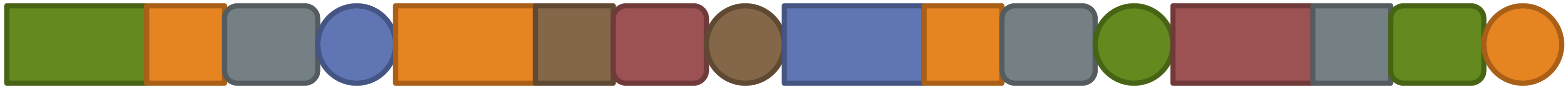
Why?

- Schemas are a good idea
  - Parsing fields out of flat text files is slow
  - Schemas define a contract, decoupling logical from physical
- Schemas allow for building efficient auxiliary structures
  - Value indexes, join indexes, etc.
- Relational algorithms have been optimised for the underlying system
  - The system itself has complete control of performance-critical decisions
  - Storage layout, choice of algorithm, order of execution, etc.

Storage layout: row vs. column stores



Row store



Column store

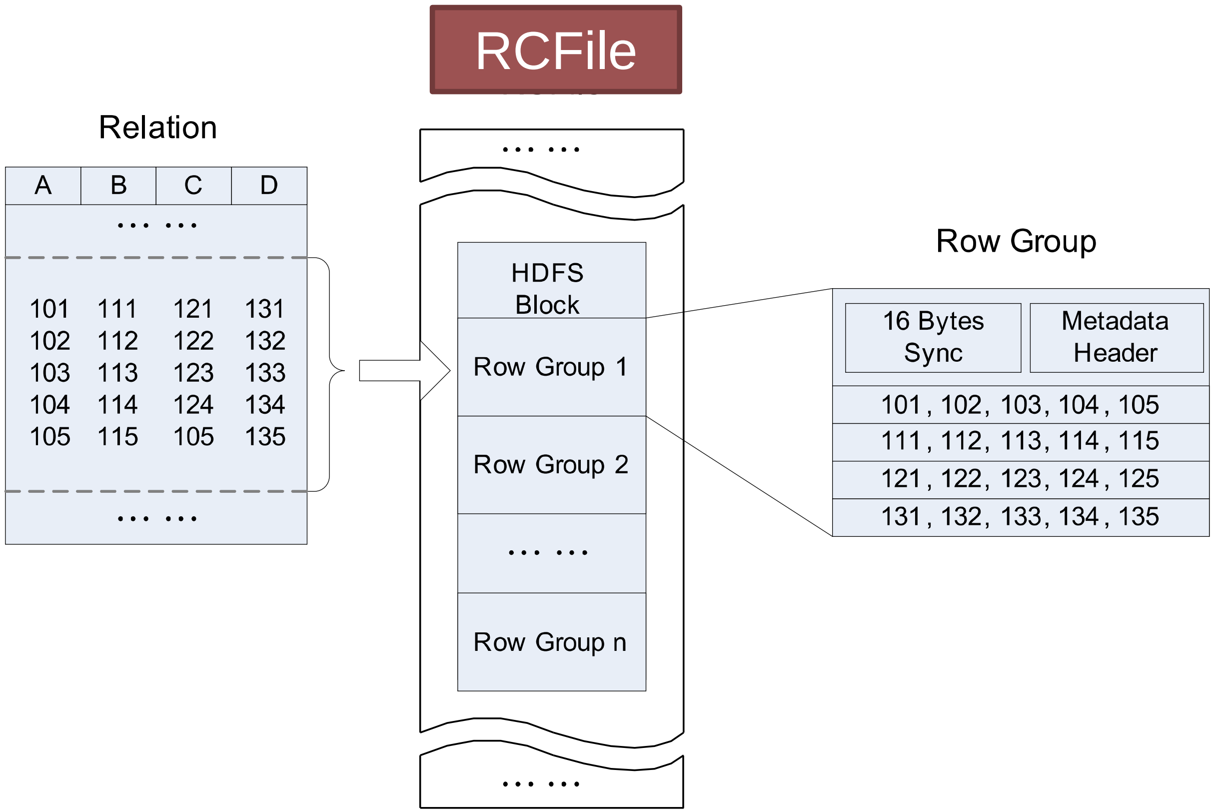
Storage layout: row vs. column stores

- Row stores
  - Easy to modify a record
  - Might read unnecessary data when processing
- Column stores
  - Only read necessary data when processing
  - Tuple writes require multiple accesses

Advantages of column stores

- Read efficiency
  - If only need to access a few columns, no need to drag around the rest of the values
- Better compression
  - Repeated values appear more frequently in a column than repeated rows appear
- Vectorised processing
  - Leveraging CPU architecture-level support
- Opportunities to operate directly on compressed data
  - For instance, when evaluating a selection; or when projecting a column

Why not in Hadoop?



No reason why not

event.cwi.nl/lsde

Source: He et al. (2011) RCFile: A Fast and Space-Efficient Data Placement Structure in MapReduce-based Warehouse Systems. ICDE.

Some small steps forward

- MapReduce is a step backward in database access:
  - Schemas are good ✔
  - Separation of the schema from the application is good ✔
  - High-level access languages are good ?
- MapReduce is poor implementation
  - Brute force and only brute force (no indexes, for example) ✔
- MapReduce is not novel
- MapReduce is missing features
  - Bulk loader, indexing, updates, transactions… ?
- MapReduce is incompatible with DMBS tools

Source: Blog post by DeWitt and Stonebraker

MODERN
SQL-ON-HADOOP SYSTEMS

Analytical Database Systems

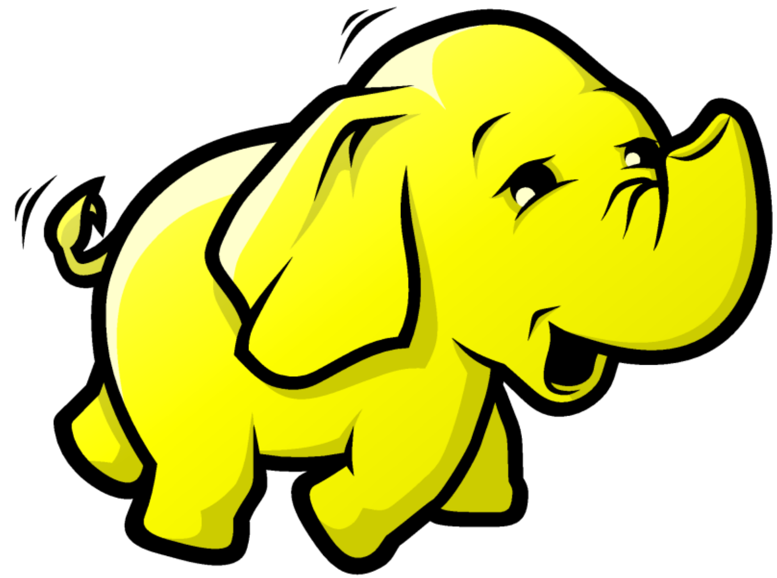
Parallel (MPP):

Teradata
Paraccel
Pivotal
Vertica
Redshift

Oracle (IMM)
Netteza
DB2-BLU
InfoBright
SQLserver (columnstore)
Vectorwise

open source:

MySQL
LucidDB
MonetDB



?

Centrum Wiskunde & Informatica

SQL-on-Hadoop Systems

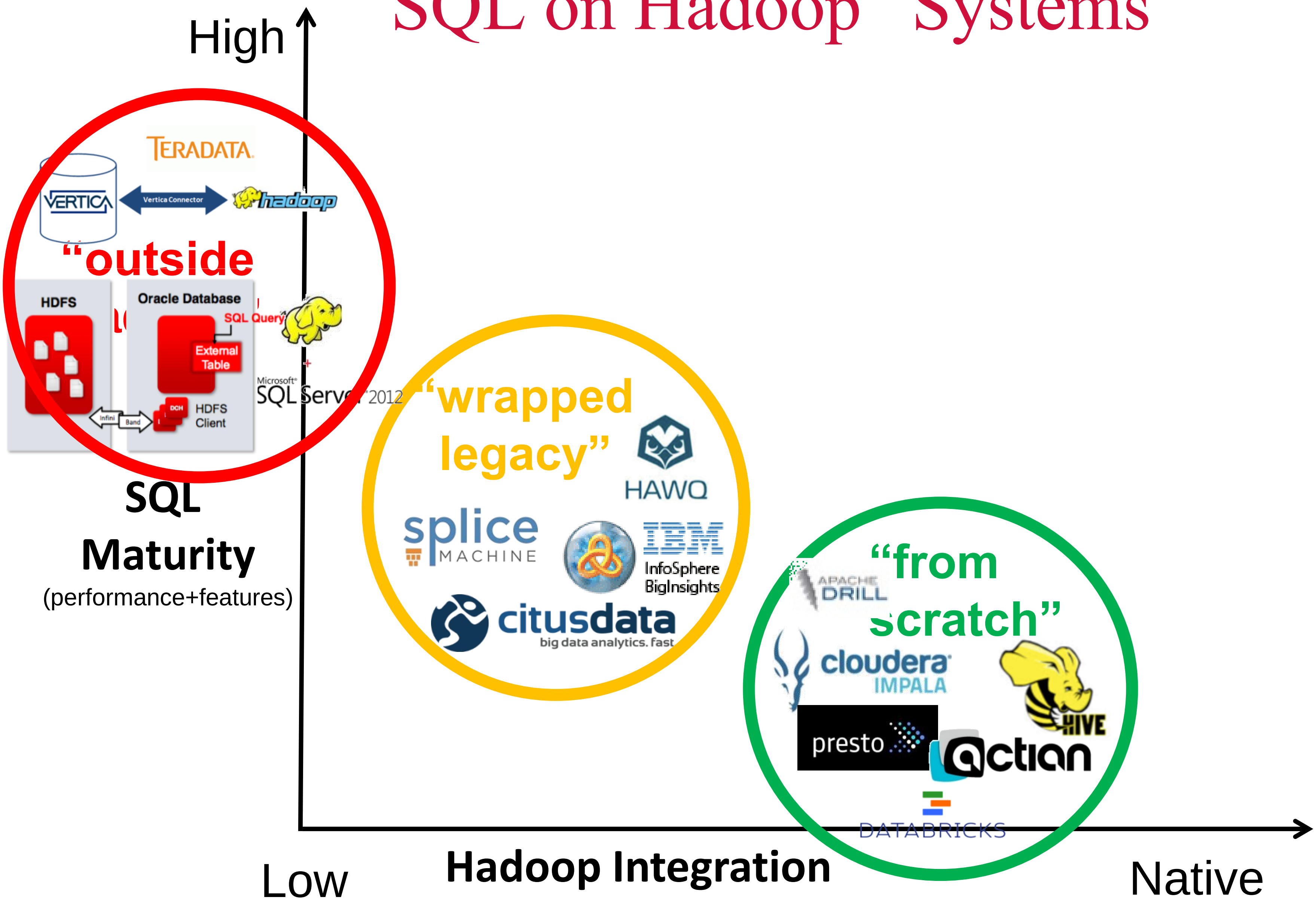
Open Source:

- Hive (HortonWorks)
- Impala (Cloudera)
- Drill (MapR)
- Presto (Facebook)

Commercial:

- HAWQ (Pivotal)
- Vortex (Actian)
- Vertica Hadoop (HP)
- BigQuery (IBM)
- DataBricks
- Splice Machine
- CitusData
- InfiniDB Hadoop

“SQL on Hadoop” Systems



Analytical DB engines for Hadoop

storage

– **columnar storage** + compression
– table partitioning / distribution
– exploiting correlated data

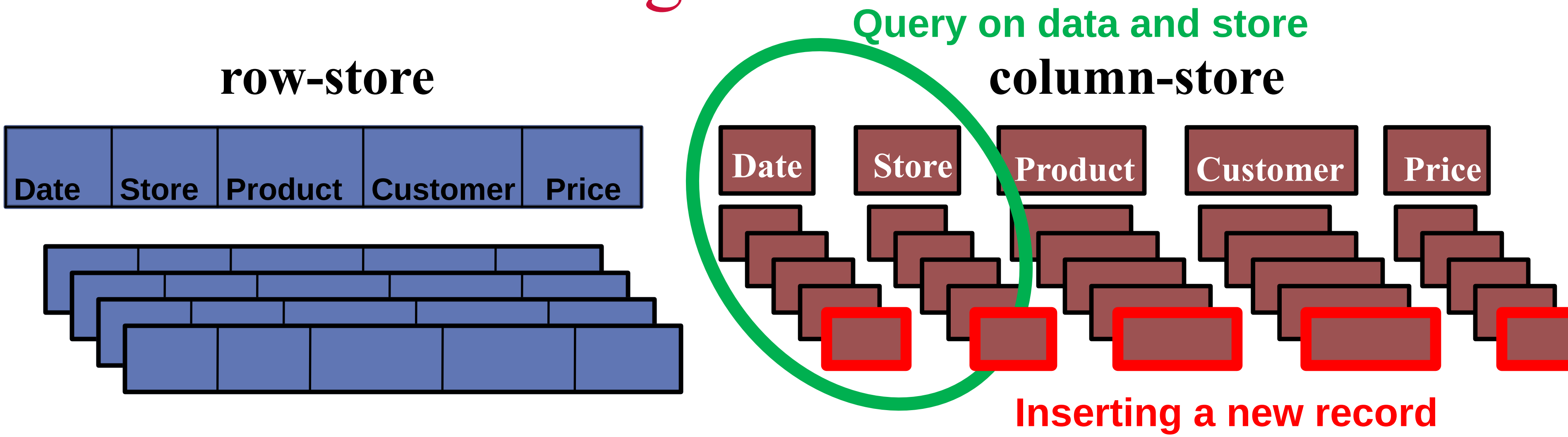
query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

Columnar Storage



+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

=> *suitable for read-mostly, read-intensive, large data repositories*

Centrum Wiskunde & Informatica

Analytical DB engines for Hadoop

storage

– columnar storage + **compression**

– table partitioning / distribution

– exploiting correlated data

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

Columnar Compression

- Trades I/O for CPU
  - A winning proposition currently
  - Even trading RAM bandwidth for CPU wins
    - 64 core machines starved for RAM bandwidth
- Additional column-store synergy:
  - Column store: data of the same distribution close together
    - Better compression rates
    - Generic compression (gzip) vs Domain-aware compression
  - Synergy with vectorized processing (see later) compress/decompress/execution, SIMD
  - Can use extra space to store multiple copies of data in different sort orders (see later)

Run-length Encoding

Quarter	Product ID	Price
Q1	1	5
Q1	1	7
Q1	1	2
Q1	1	9
Q1	1	6
Q1	2	8
Q1	2	5
…	…	…
Q2	1	3
Q2	1	8
Q2	1	1
Q2	2	4
…	…	…

→

Quarter (value, start_pos, run_length)
(Q1, 1, 300)
(Q2, 301, 350)
(Q3, 651, 500)
(Q4, 1151, 600)

Product ID (value, start_pos, run_length)
(1, 1, 5)
(2, 6, 2)
…
(1, 301, 3)
(2, 304, 1)
…

Price
5
7
2
9
6
8
5
…
3
8
1
4
…

“Integrating Compression and Execution in Column-Oriented Database Systems” Abadi et. al, SIGMOD ’06

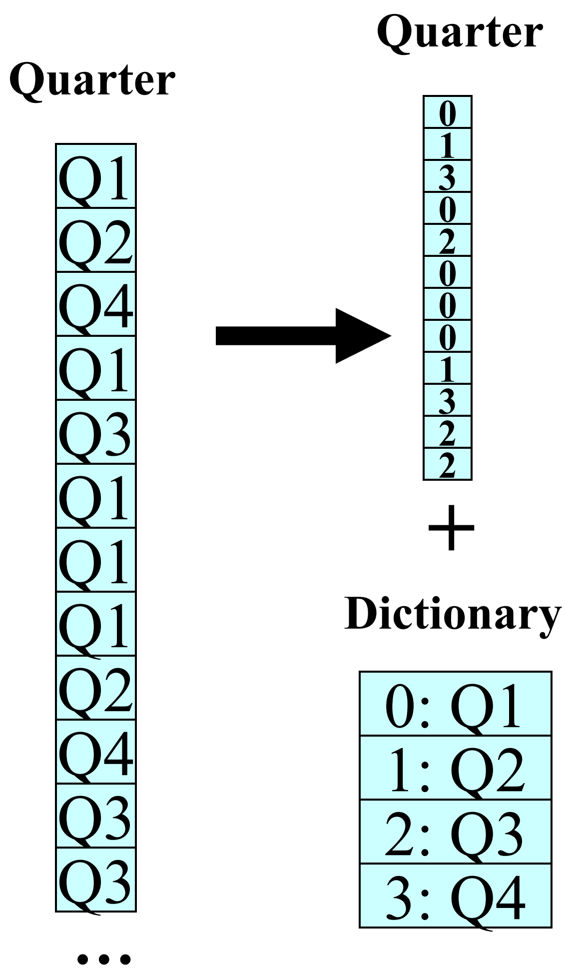
Bitmap Encoding

- For each unique value, v, in column c, create bit-vector b
  - b[i] = 1 if c[i] = v
- Good for columns with few unique values
- Each bit-vector can be further compressed if sparse

Product ID	ID: 1	ID: 2	ID: 3	…
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
1	1	0	0	0
2	0	1	0	0
2	0	1	0	0
…	…	…	…	…
1	1	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
…	…	…	…	…

Dictionary Encoding

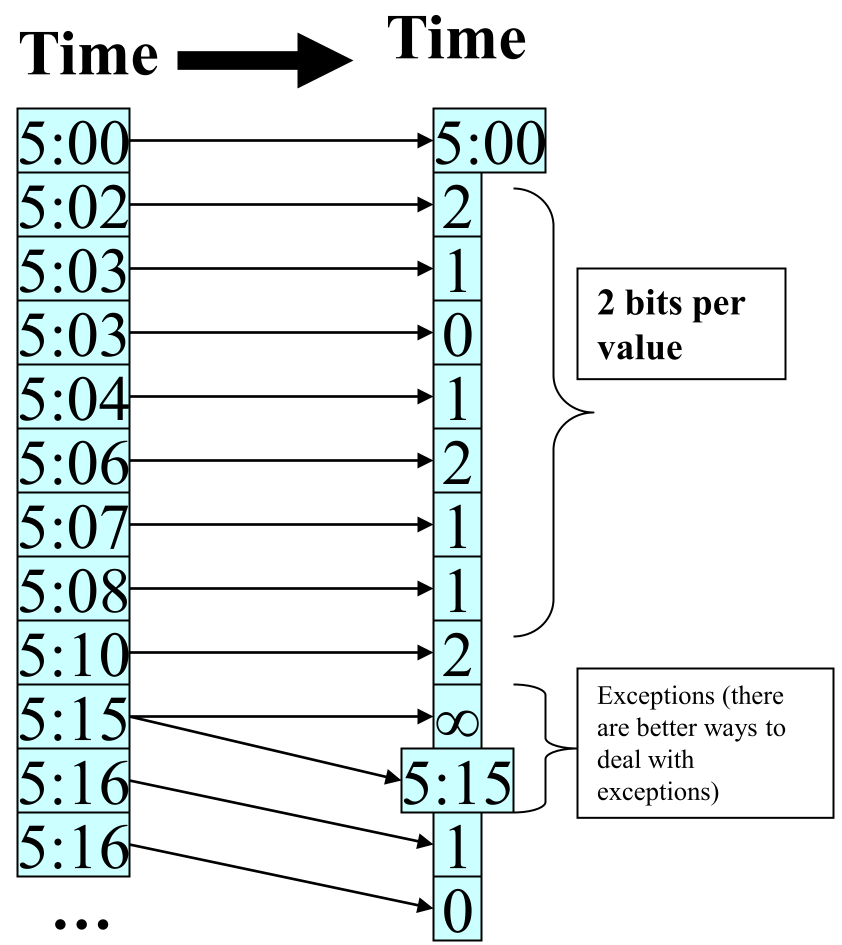
- **For each unique value create dictionary entry**
- **Dictionary can be per-block or per-column**
- **Column-stores have the advantage that dictionary entries may encode multiple values at once**



Differential Encoding

- **Encodes values as b bit offset from previous value**
- **Special escape code (just like frame of reference encoding) indicates a difference larger than can be stored in b bits**
  - **After escape code, original (uncompressed) value is written**
- **Performs well on columns containing increasing/decreasing sequences**
  - **inverted lists**
  - **timestamps**
  - **object IDs**
  - **sorted / clustered columns**

"Improved Word-Aligned Binary Compression for Text Indexing" Ahn, Moffat, TKDE'06



“Super-Scalar RAM-CPU Cache Compression”
Zukowski, Heman, Nes, Boncz, ICDE’06

Heavy-Weight Compression Schemes

Algorithm	Decompression Bandwidth
BZIP	10 MB/s
ZLIB	80 MB/s
LZO	300 MB/s

- Modern disks (SSDs) can achieve > 1GB/s
- 1/3 CPU for decompression ➔ 3GB/s needed

➔ **Lightweight compression schemes are better**

➔ **Even better: operate directly on compressed data**

Operating Directly on Compressed Data

Examples

- SUM_i(rle-compressed column[i]) ➔ SUM_g(count[g] * value[g])
- (country == "Asia") ➔ countryCode == 6

strcmp SIMD

Benefits:

- I/O - CPU tradeoff is no longer a tradeoff (CPU also gets improved)
- Reduces memory–CPU bandwidth requirements
- Opens up possibility of operating on multiple records at once

Analytical DB engines for Hadoop

storage

– columnar storage + compression
– **table partitioning / distribution**
– exploiting correlated data

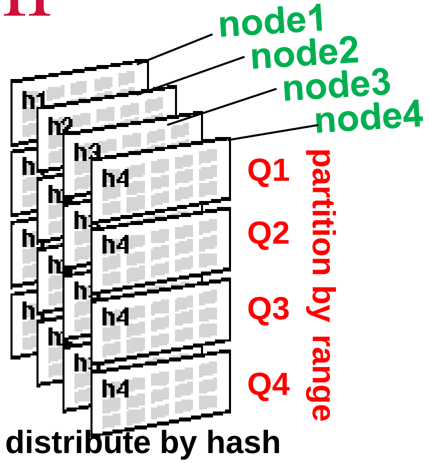
query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

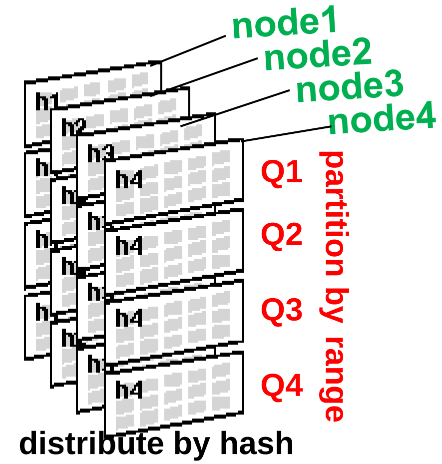
Table Partitioning and Distribution



- data is spread based on a Key
  - Functions: Hash, Range, List
- "distribution"
  - Goal: parallelism
    - give each compute node a piece of the data
    - each query has work on every piece (keep everyone busy)
- "partitioning"
  - Goal: data lifecycle management
    - Data warehouse e.g. keeps last six months
    - Every night: load one new day, drop the oldest partition
  - Goal: improve access patterm
    - when querying for May, drop Q1,Q3,Q4 ("partition pruning")

Which kind of function would you use for which method?

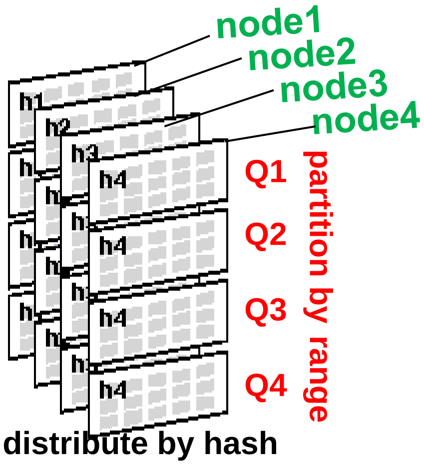
Data Placement in Hadoop



- Each node writes the partitions it owns
  - Where does the data end up, really?
- HDFS default block placement strategy:
  - Node that initiates writes gets first copy
  - 2nd copy on the same rack
  - 3rd copy on a different rack
- Rows from the same record should on the same node
  - Not entirely trivial in column stores
    - Column partitions should be co-located
  - Simple solution:
    - Put all columns together in one file (RCFILE, ORCFILE, Parquet)
  - Complex solution:
    - Replace the default HDFS block placement strategy by a custom one

Popular File Formats in Hadoop

- Good old CSV
  - Textual, easy to parse (but slow), better compress it!
- Sequence Files
  - Binary data, faster to process
- RCfile
  - Hive first attempt at column-store
- ORCfile
  - Columnar compression, MinMax
- Parquet
  - Proposed by Twitter and Cloudera Impala
  - Like ORCfile, no MinMax



CWI
Centrum Wiskunde & Informatica

Example: Parquet Format

Object model (memory)

Object models are in-memory representations of data.

Avro | Thrift | Protocol Buffers | Hive | Pig | Example

Object model converters

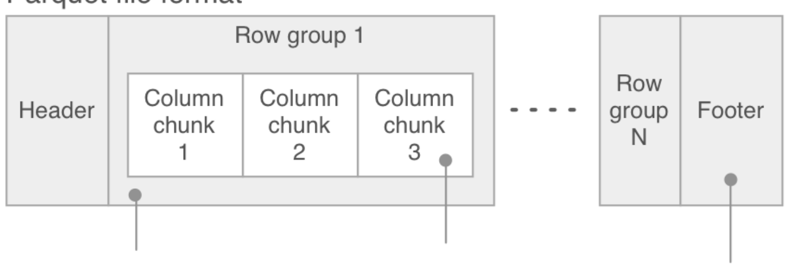
Object model converters are part of the "parquet-mr" project. They are responsible from mapping between external object models and Parquet's internal data types.

Avro | Thrift | Protocol Buffers | Hive | Pig | Example

Storage format (disk)

On-disk, Parquet data is in binary form using its own formally-specified columnar file format.

Parquet file format



A **row group** stores all the column values for a range of rows in a columnar layout.

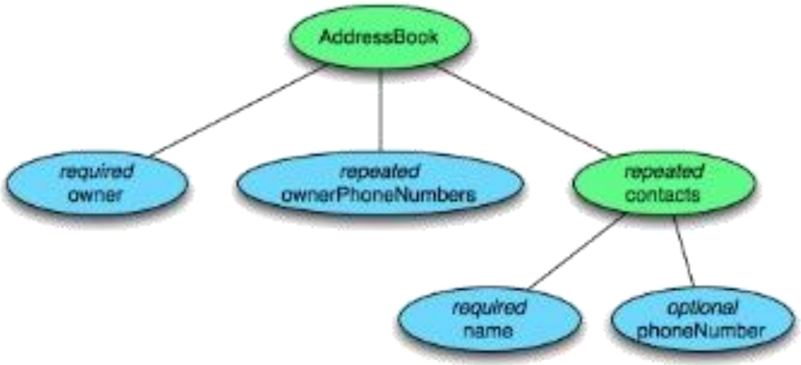
A **column chunk** contain all the values for an individual column in the row group.

The **footer** contains schema details, object model metadata and metadata about the row groups and columns.

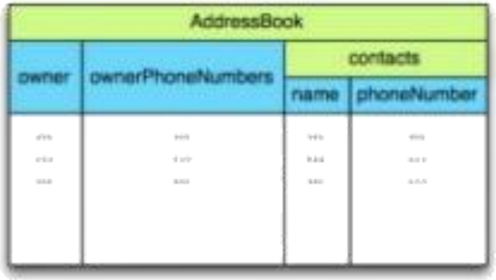
Shaded boxes are part of the Parquet project

Example: Parquet Format

Table Format



Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

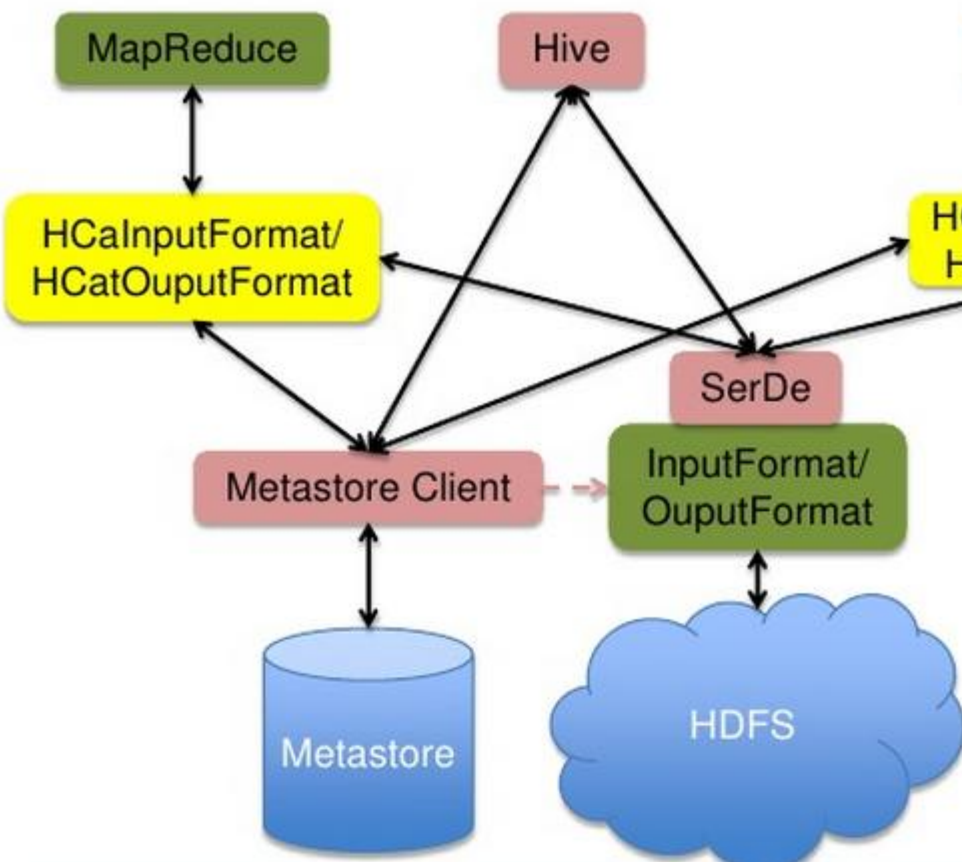


http://dataera.wordpress.com http://linkedin.com/in/yuechen2

HCatalog ("Hive MetaStore")

De-facto Metadata Standard on Hadoop

- Where are the tables? Wat do they contain? How are they Partitioned?
- Can I read from them? Can I write to them?



SQL-on-Hadoop challenges:

- Reading-writing many file formats
- Opening up the own datastore to foreign tools that read from it

HCatalog makes UDFs less important!

Centrum Wiskunde & Informatica

Analytical DB engines for Hadoop

storage

- columnar storage + compression
- table partitioning / distribution
- **exploiting correlated data**

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

Exploiting Natural Order

- Data is often naturally ordered
  - very often, on date
- Data is often correlated
  - orderdate/paydate/shipdate
  - marketing campaigns/date
  - ..correlation is everywhere

    ..hard to predict

Zone Maps

- Very sparse index
- Keeps MinMax for every column
- Cheap to maintain
  - Just widen bounds on each modification

Q: acctno BETWEEN 150 AND 200?

Accounts

KEY	acctno	name	balance	
00	019	Isabella	269.38	zone 0
01	038	Jackson	914.11	
02	072	Lucas	346.61	
03	156	Sophia	266.55	
04	153	Mason	850.90	zone 1
05	282	Ethan	521.60	
06	389	Emily	647.38	
07	314	Lily	119.40	
08	332	Chloe	526.08	zone 2
09	302	Emma	497.19	
10	533	Aiden	22.03	
11	592	Ava	140.67	
12	808	Mia	383.69	zone 3
13	896	Jacob	899.41	

Accounts.MinMax

zone	KEY		acctno		name		balance	
	min	max	min	max	min	max	min	max
0	00	03	019	156	Isabella	Sophia	266.55	914.11
1	04	07	153	389	Emily	Mason	119.40	850.90
2	08	11	332	592	Aiden	Emma	22.03	526.08
3	12	13	808	896	Mia	Jacob	383.69	899.41

Q: key BETWEEN 13 AND 15?

Analytical DB engines for Hadoop

storage

- columnar storage + compression
- table partitioning / distribution
- exploiting correlated data

query-processor

- **CPU-efficient query engine** (**vectorized** or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

DBMS Computational Efficiency?

TPC-H 1GB, query 1

- selects 98% of fact table, computes net prices and aggregates all
- Results:
  - C program: ?
  - MySQL: 26.2s
  - DBMS “X”: 28.1s

“MonetDB/X100: Hyper-Pipelining Query Execution ” Boncz, Zukowski, Nes, CIDR’05

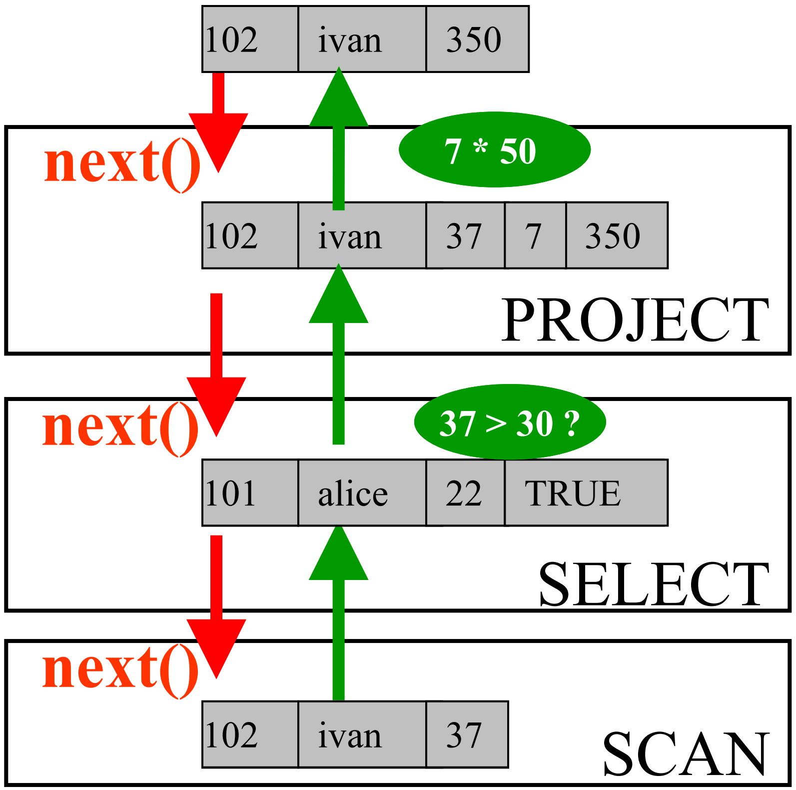
DBMS Computational Efficiency?

TPC-H 1GB, query 1

- selects 98% of fact table, computes net prices and aggregates all
- Results:
  - C program: **0.2s**
  - MySQL: 26.2s
  - DBMS “X”: 28.1s

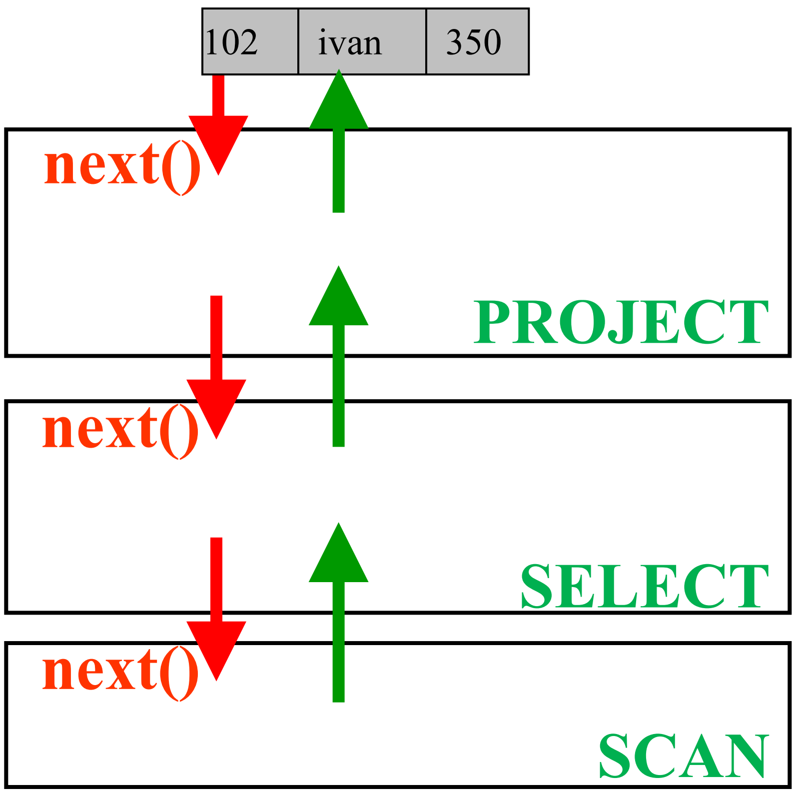
“MonetDB/X100: Hyper-Pipelining Query Execution ” Boncz, Zukowski, Nes, CIDR’05

How Do Query Engines Work?



SELECT id, name
(age-30)*50 AS bonus
FROM employee
WHERE age > 30

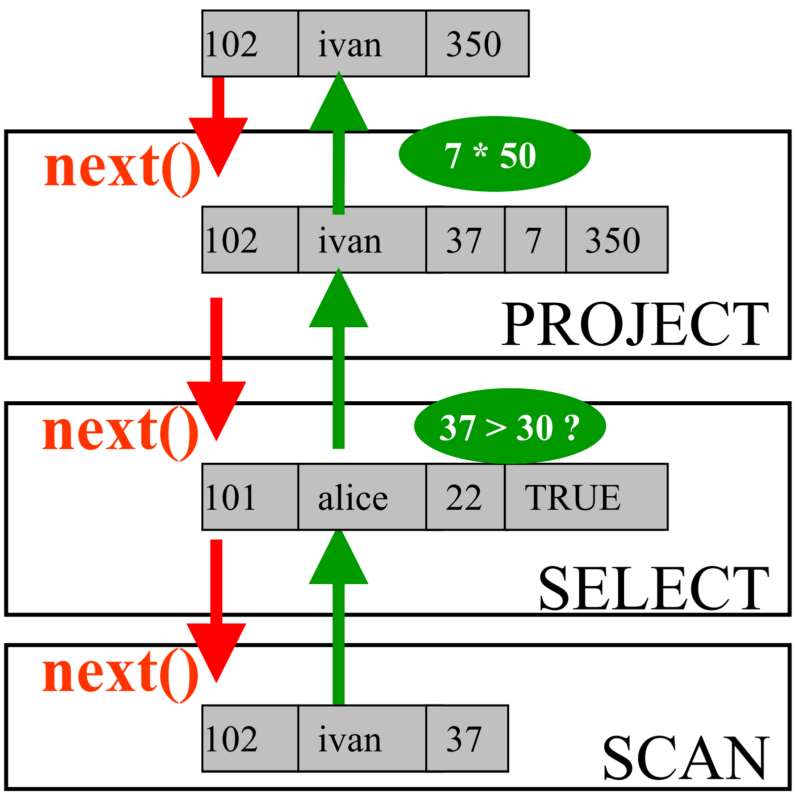
How Do Query Engines Work?



Operators

Iterator interface
-open()
-next(): tuple
-close()

How Do Query Engines Work?



Primitives

Provide computational functionality

All arithmetic allowed in expressions,
e.g. Multiplication

`mult(int,int)` ➔ `int`

"MonetDB/X100: Hyper-Pipelining Query Execution " Boncz, Zukowski, Nes, CIDR'05

vectorwise

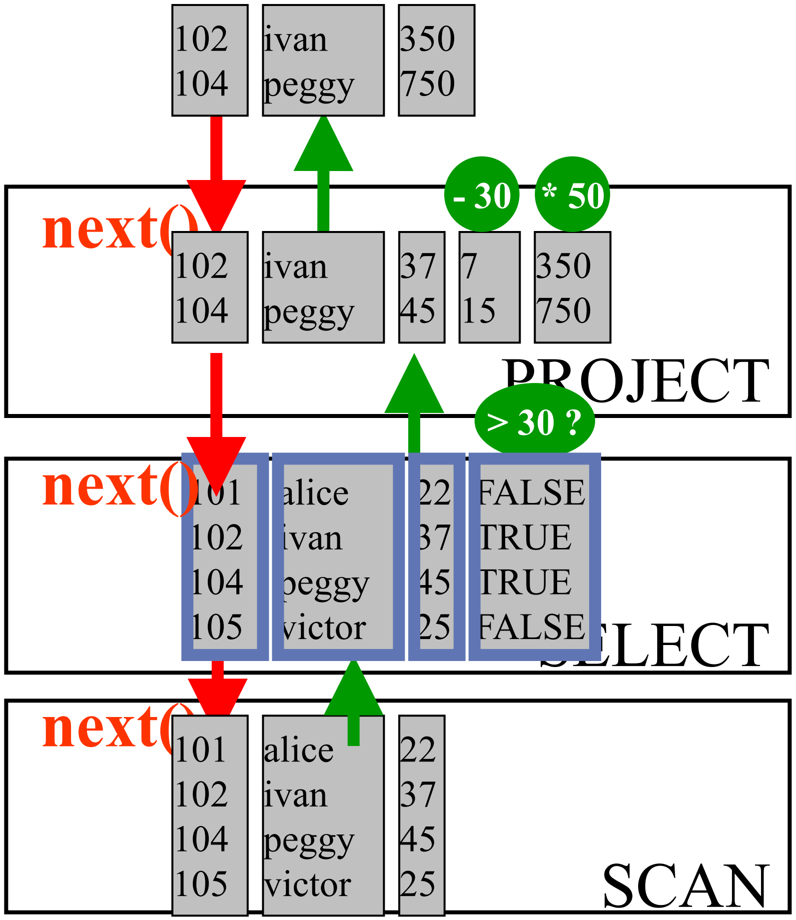
Observations:

"Vectorized In Cache Processing"

vector = array of ~100

processed in a tight loop

CPU cache Resident



Centrum Wiskunde & Informatica

"MonetDB/X100: Hyper-Pipelining Query Execution " Boncz, Zukowski, Nes, CIDR'05

vectorwise

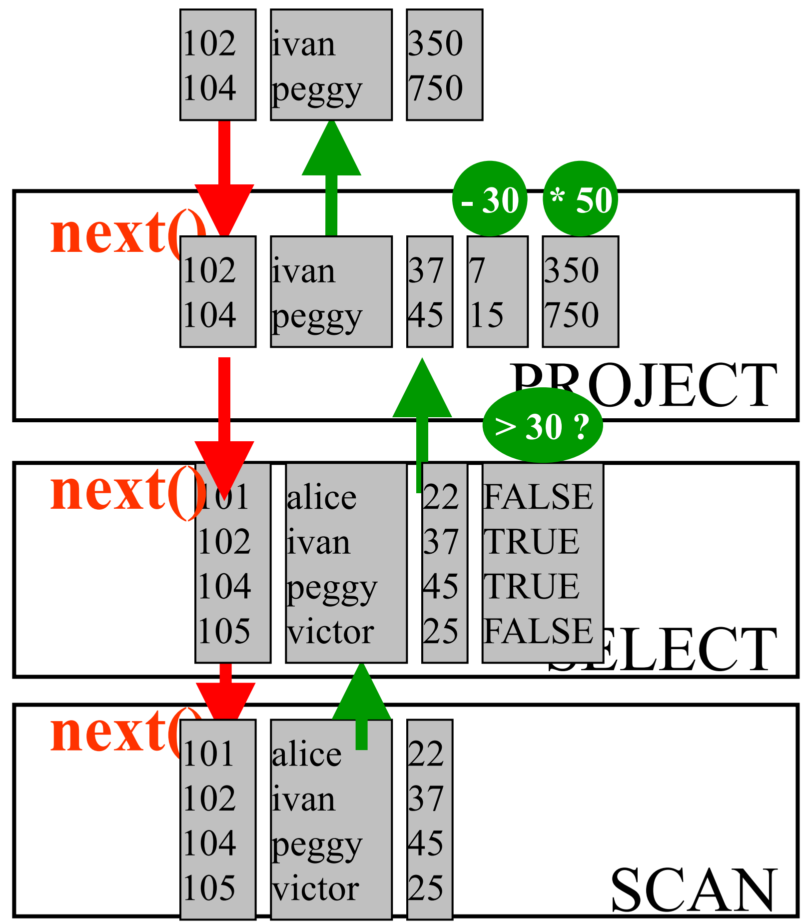
Observations:

next() called much less often ➔ more time spent in primitives less in overhead

primitive calls process an

CPU Efficiency depends on "nice" code
- out-of-order execution
- few dependencies (control,data)
- compiler support

Compilers like simple loops over arrays
- loop-pipelining
- automatic SIMD



vectorwise

Observations:

next() called much less often ➔ more time spent in primitives less in overhead

primitive calls process an

CPU Efficiency depends on "nice" code
- out-of-order execution
- few dependencies (control,data)
- compiler support

Compilers like simple loops over arrays
- loop-pipelining
- automatic SIMD

> 30 ?

FALSE
TRUE
TRUE
FALSE

```
for(i=0; i<n; i++)
  res[i] = (col[i] > x)
```

- 30

7
15

```
for(i=0; i<n; i++)
  res[i] = (col[i] - x)
```

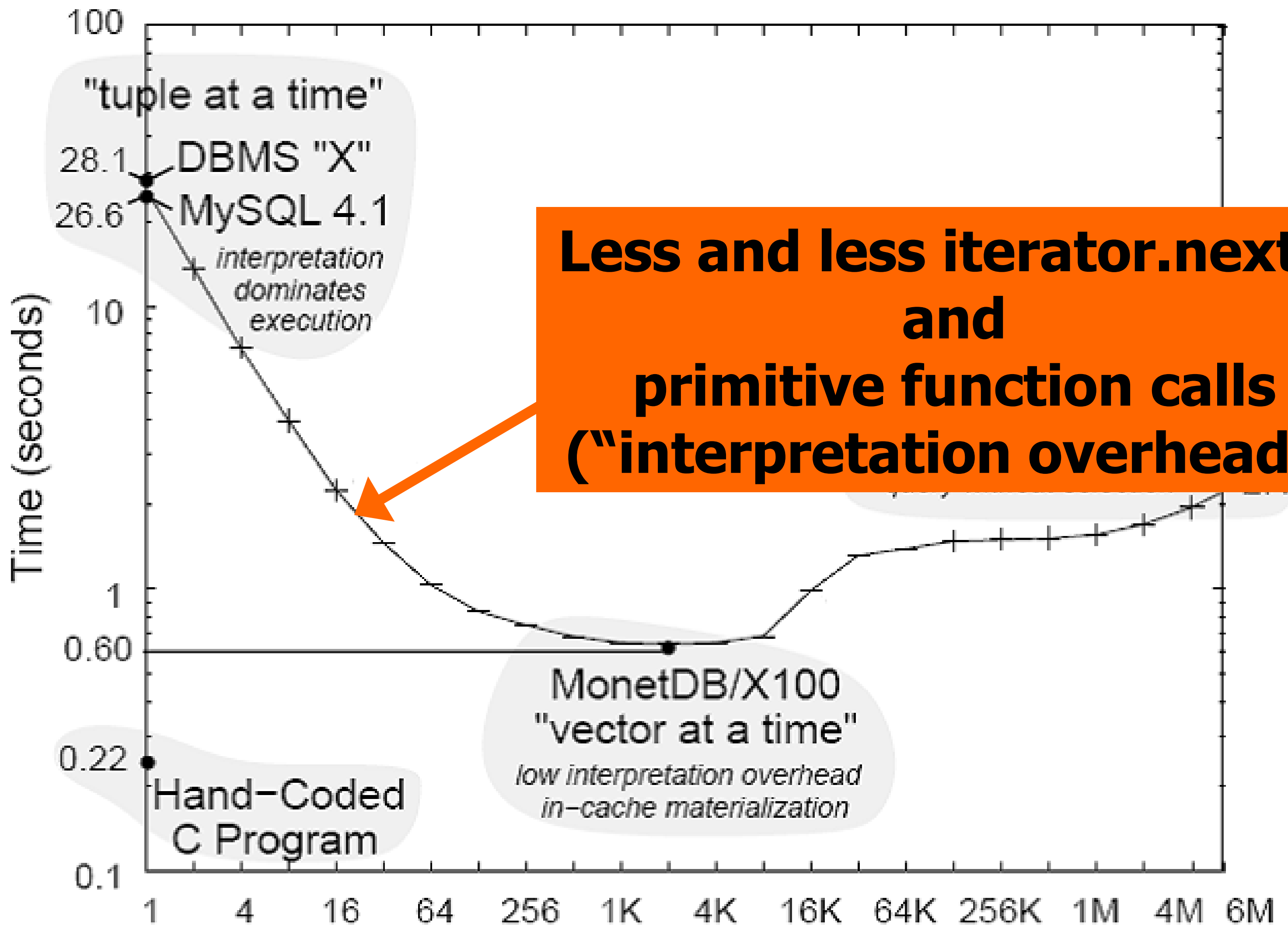
* 50

350
750

```
for(i=0; i<n; i++)
  res[i] = (col[i] * x)
```

"MonetDB/X100: Hyper-Pipelining Query Execution " Boncz, Zukowski, Nes, CIDR'05

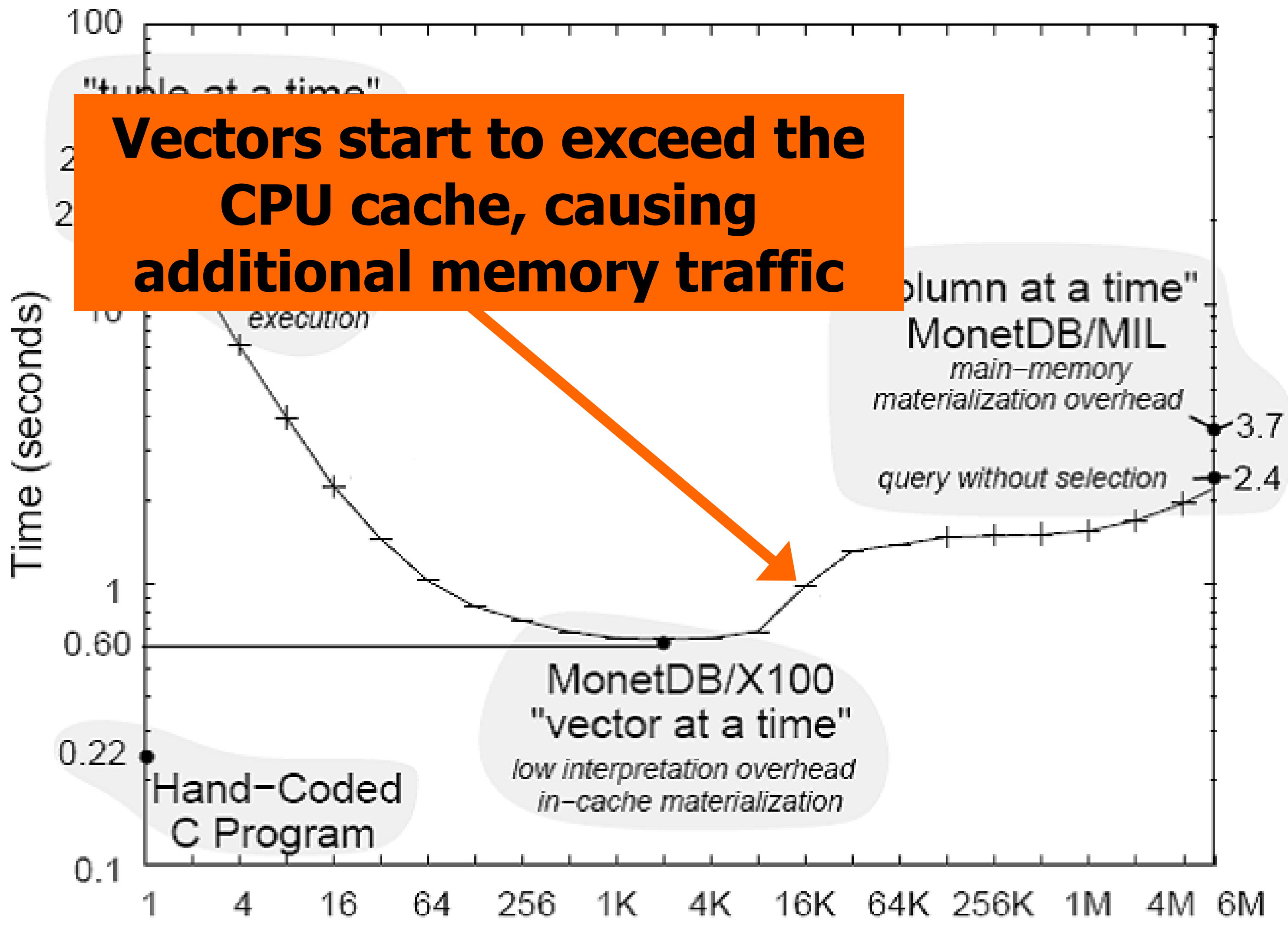
Varying the Vector size



Less and less iterator.next() and primitive function calls ("interpretation overhead")

"MonetDB/X100: Hyper-Pipelining Query Execution " Boncz, Zukowski, Nes, CIDR'05

Varying the Vector size



Systems That Use Vectorization

- Actian Vortex (Vectorwise-on-Hadoop)
- Hive, Drill

Vectorization

- Drill operates on more than one record at a time
  - Word-sized manipulations
  - SIMD instructions
    - GCC, LLVM and JVM all do various optimizations automatically
  - Manually code algorithms
- Logical Vectorization
  - Bitmaps allow lightning fast null-checks
  - Avoid branching to speed CPU pipeline

MAPR

Analytical DB engines for Hadoop

storage

- columnar storage + compression
- table partitioning / distribution
- exploiting correlated data

query-processor

- **CPU-efficient query engine** (vectorized or **JIT codegen**)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

Analytical DB engines for Hadoop

storage

– columnar storage + compression
– table partitioning / distribution
– exploiting correlated data

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- analytical SQL (windowing)

system

- **batch update infrastructure**
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

CWI
Centrum Wiskunde & Informatica

Batch Update Infrastructure (Vertica)

Challenge: hard to update columnar compressed data

Trickle Load

> Write Optimized Store (WOS)

A B C

- Memory based
- Unsorted / Uncompressed
- Segmented
- Low latency / Small quick inserts

TUPLE MOVER
Asynchronous Data Transfer

> Read Optimized Store (ROS)

- On disk
- Sorted / Compressed
- Segmented
- Large data loaded direct

A B C

(A B C | A)

Batch Update Infrastructure (Hive)

Challenge: HDFS read-only + large block size

Base File

Name	Purchase
Anne	Red Fish
Bill	Blue Fish
Christine	Blue Fish
David	Black Fish
Eric	Young Fish

Merge During Query Processing

Logical File

Name	Purchase
Joe	Old Fish
Ann	Star
Bill	Blue Fish
David	Black Fish

Update 1

Op	Txn Id	RowId	Name	Purchase
I	1	0	Joe	Red Fish
U	0	0	Anne	Star
D	0	4		

Update 2

Op	Txn Id	RowId	Name	Purchase
U	1	0	Joe	Old Fish
U	0	0	Ann	Star
D	0	2		

Hortonworks

Analytical DB engines for Hadoop

storage

– columnar storage + compression
– table partitioning / distribution
– exploiting correlated data

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- **rich SQL** (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- YARN & elasticity

SQL-99 OLAP Extensions

- ORDER BY .. PARTITION BY
  - window specifications inside a partition
    - first_value(), last_value(), …
  - Rownum(), dense_rank(), …

```
SELECT empno, deptno, sal,
       AVG(sal) OVER (PARTITION BY deptno) AS avg_dept_sal
FROM   emp;

     EMPNO     DEPTNO        SAL AVG_DEPT_SAL
---------- ---------- ---------- ------------
      7782         10       2450   2916.66667
      7839         10       5000   2916.66667
      7934         10       1300   2916.66667

      7566         20       2975         2175
      7902         20       3000         2175
      7876         20       1100         2175
      7369         20        800         2175
      7788         20       3000         2175

      7521         30       1250   1566.66667
      7844         30       1500   1566.66667
      7499         30       1600   1566.66667
      7900         30        950   1566.66667
      7698         30       2850   1566.66667
      7654         30       1250   1566.66667
```

Centrum Wiskunde & Informatica

Analytical DB engines for Hadoop

storage

- columnar storage + compression
- table partitioning / distribution
- exploiting correlated data

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- **many-core ready**
- rich SQL (+authorization+..)

system

- batch update infrastructure
- **scaling with multiple nodes**
- MetaStore & file formats
- YARN & elasticity

CWI

Centrum Wiskunde & Informatica

Analytical DB engines for Hadoop

storage

- columnar storage + compression
- table partitioning / distribution
- exploiting correlated data

query-processor

- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- **YARN & elasticity**

Centrum Wiskunde & Informatica

YARN possibilities and limitations

Containers are used to assign:

- cores
- RAM

Limitations:

- no support for disk I/O, network (thrashing still possible)
- Long-running systems (e.g. DBMS) may want to adjust cores and RAM over time depending on workload ➔ “elasticity”

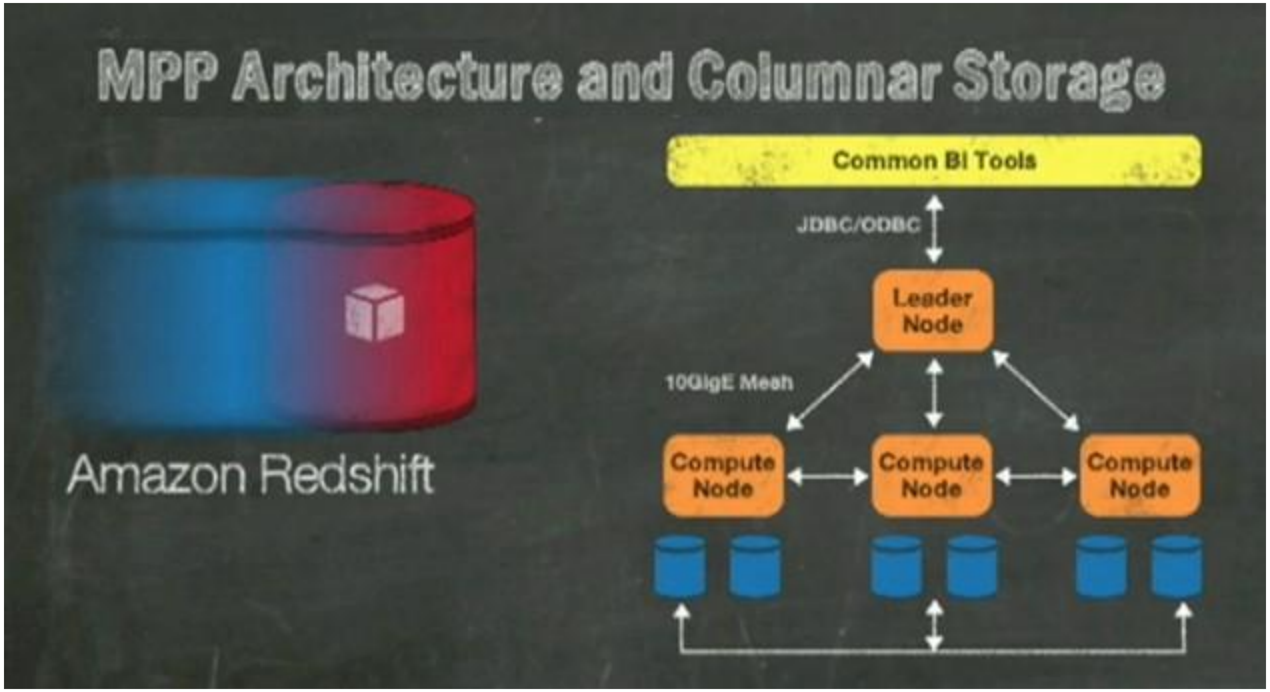
Conclusion

- SQL-on-Hadoop area is very active
  - many open-source and commercial initiatives
- There are many design dimensions
  - All design dimensions of analytical database systems
    - Column storage, compression, vectorization/JIT, MinMax pushdown, partitioning, parallel scaling, update handling, SQL99, ODBC/JDBC APIs, authorization
  - Hadoop design dimensions
    - HCatalog support, reading from and getting read from other Hadoop tools (/writing to..), file format support, HDFS locality, YARN integration

SQL IN THE CLOUD
- BUT NOT ON HADOOP

Amazon Redshift

- Cloud version of ParAccel, a parallel database
  - ParAccel is hard to manage, maintain
  - Redshift invested in simplying management, using web interface
    - No knobs, kind of elastics, User Defined Functions (python)
    - Highly performant, but storage more expensive than S3 (local disks)



Snowflake

- Brand-new, from-scratch system that works in AWS – RedShift competitor
- Stores data on S3 (cheap!) but caches it in local disks for performance
- Highly elastic, supports UDFs using JavaScript, table snapshots (“clone table”)
- Puts JSON documents in automatically recognized table format (queryable)