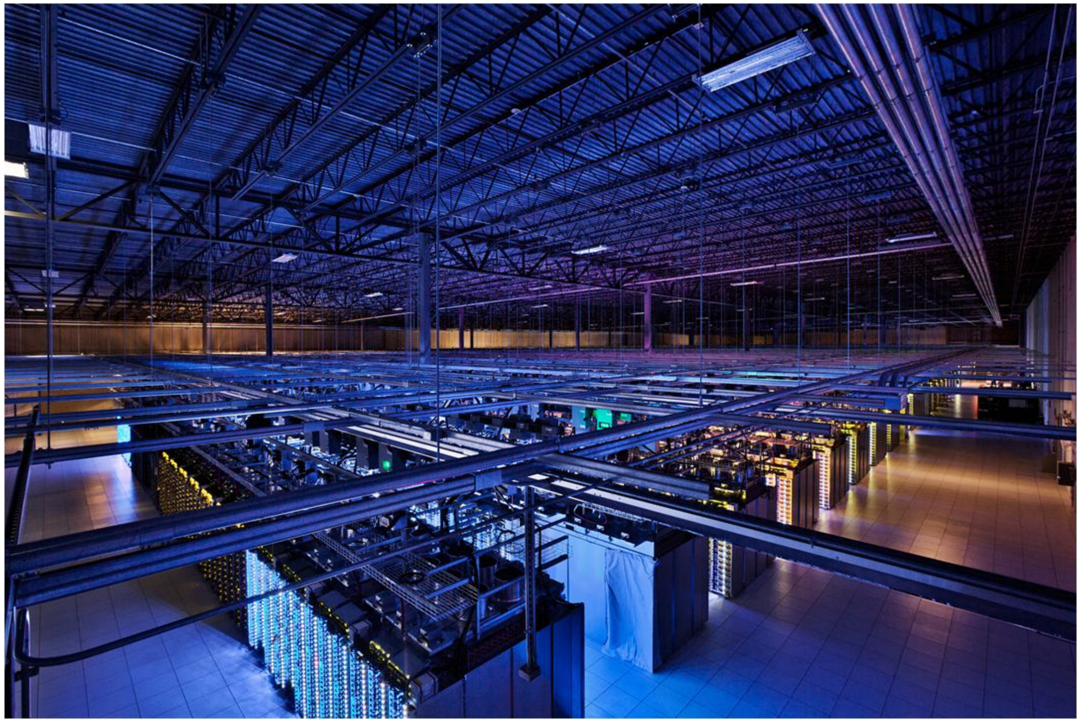
Centrum Wiskunde & Informatica

Large-Scale Data Engineering

Modern SQL-on-Hadoop Systems



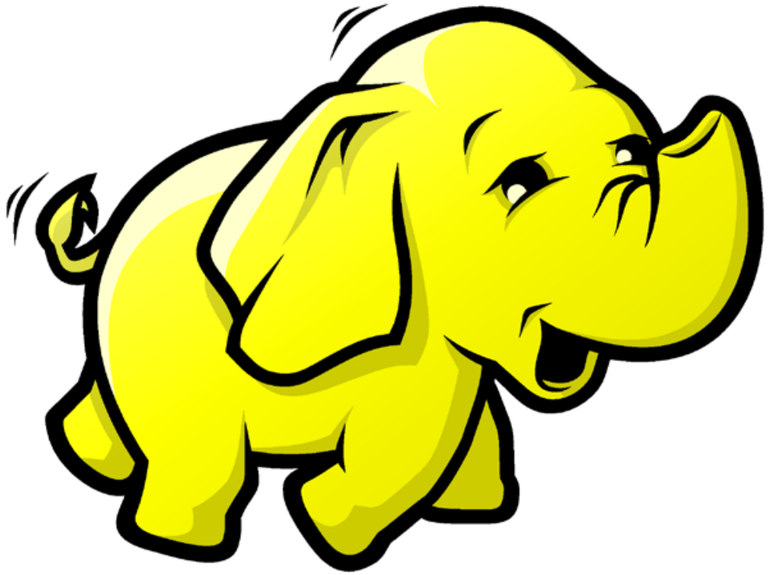
event.cwi.nl/lsde2015

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

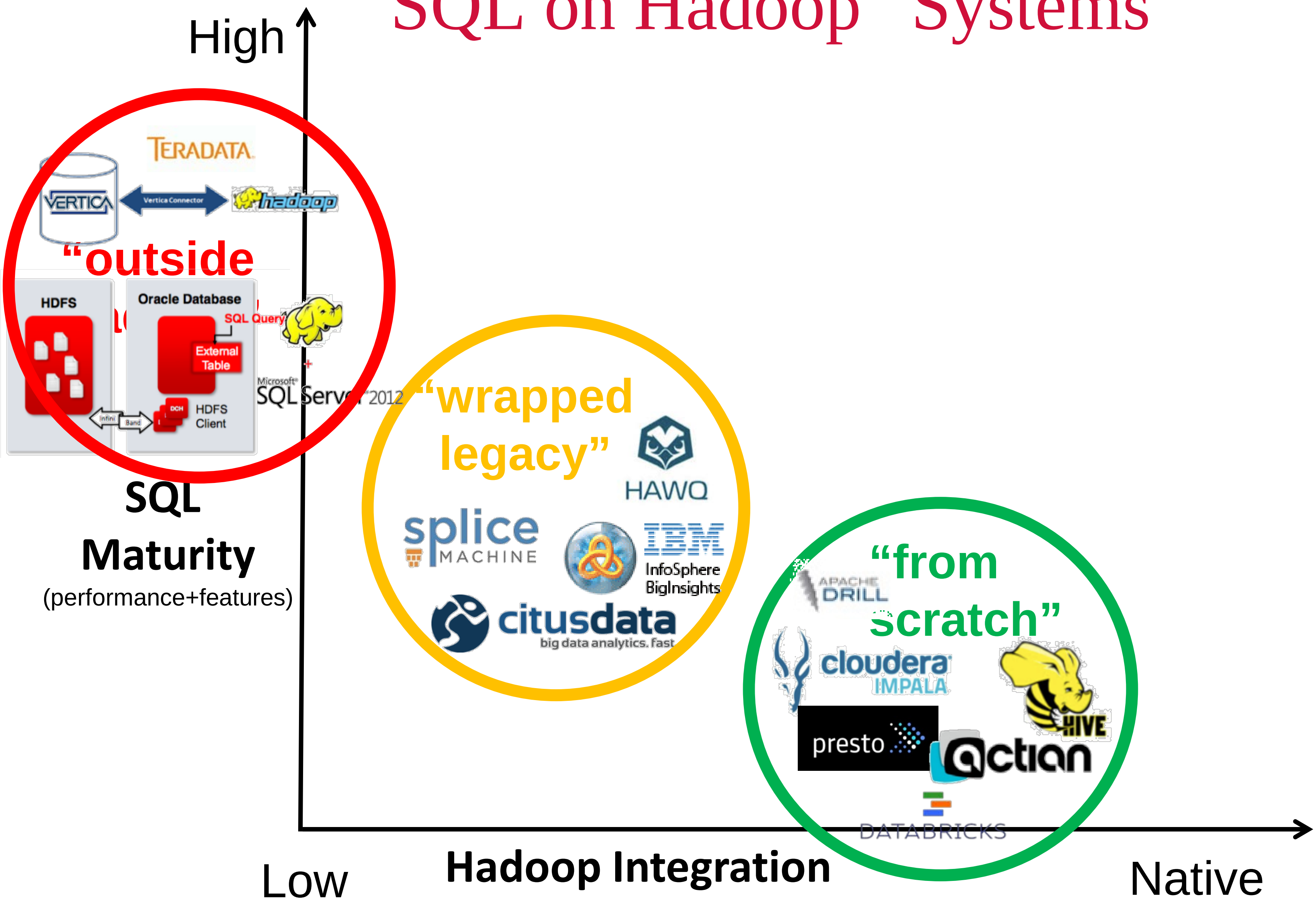
Open Source:

- 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

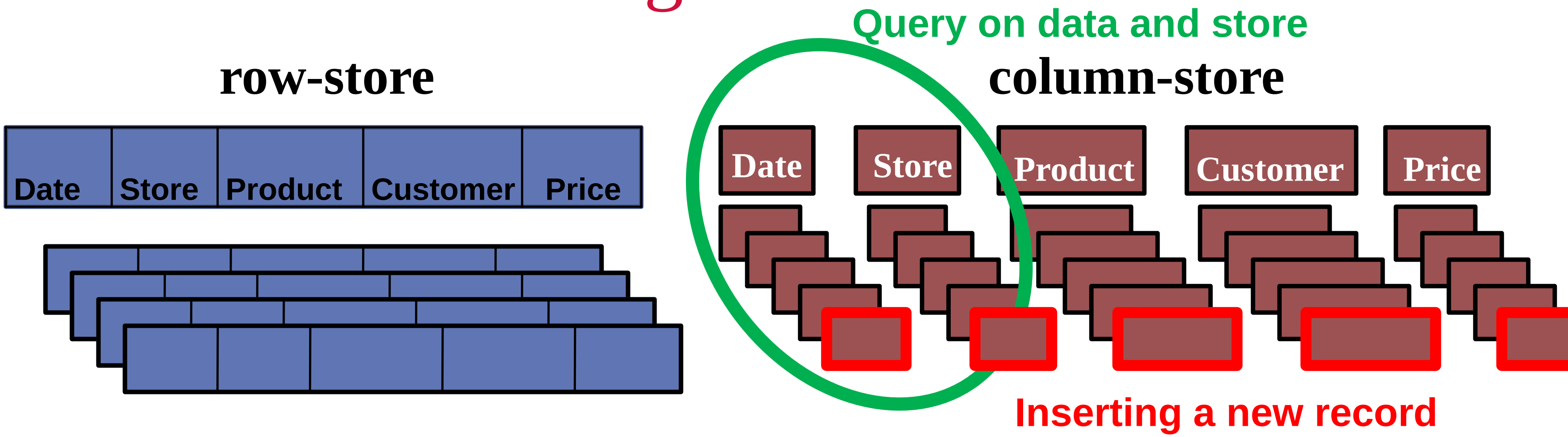
- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

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

CWI Centrum Wiskunde & Informatica

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

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

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

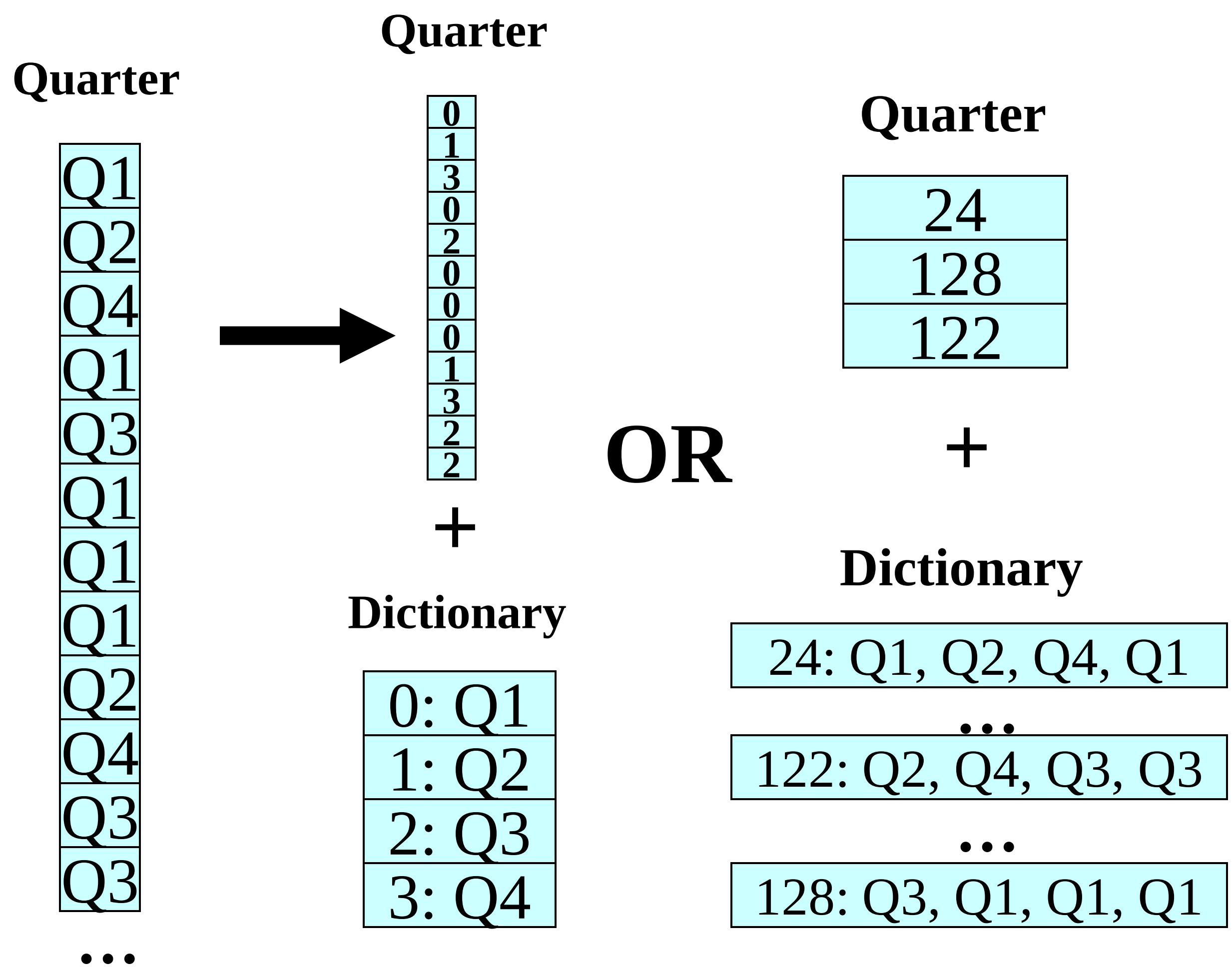
Bitmap Encoding

- 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

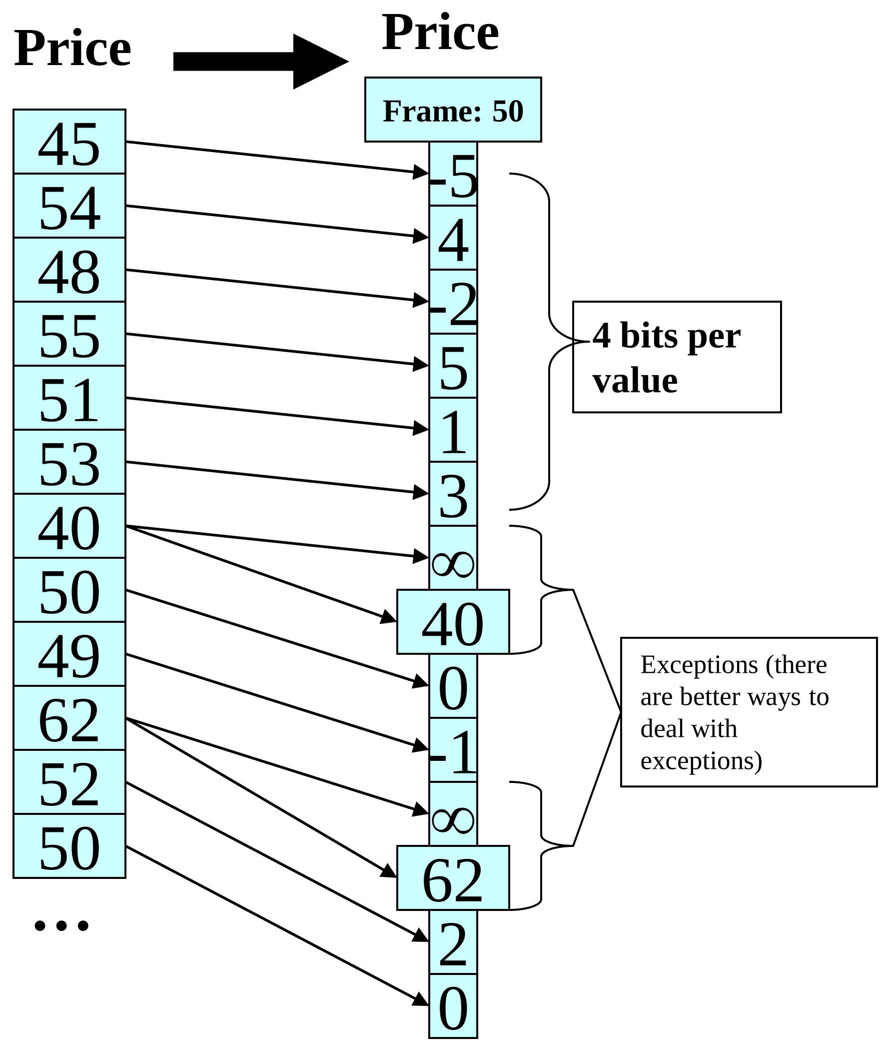
- For each unique value create dictionary entry
- Dictionary can be per-block or per-column
- Column-stores have the advantage that dictionary entries may encode multiple values at once



Frame Of Reference Encoding

- Encodes values as b bit offset from chosen frame of reference
- Special escape code (e.g. all bits set to 1) indicates a difference larger than can be stored in b bits
  - After escape code, original (uncompressed) value is written

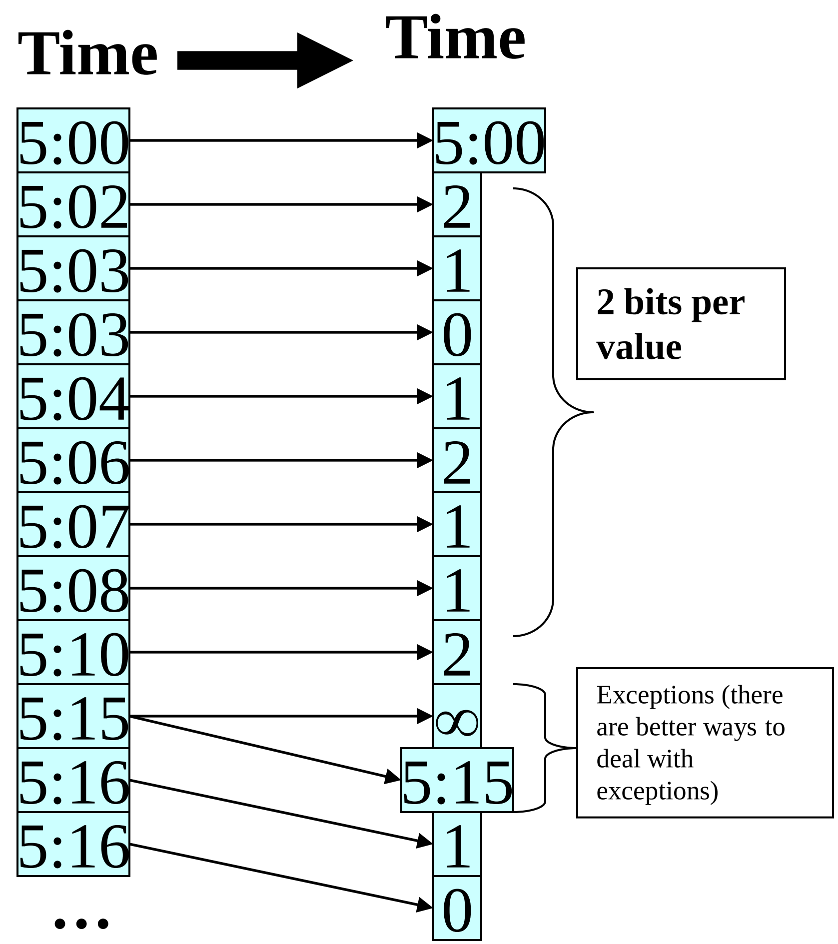
"Compressing Relations and Indexes" Goldstein, Ramakrishnan, Shaft, ICDE'98



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

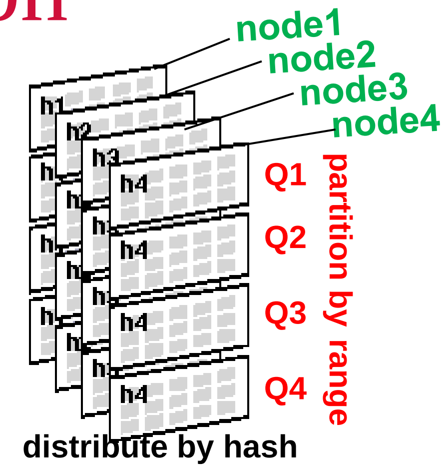
query-processor

- 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

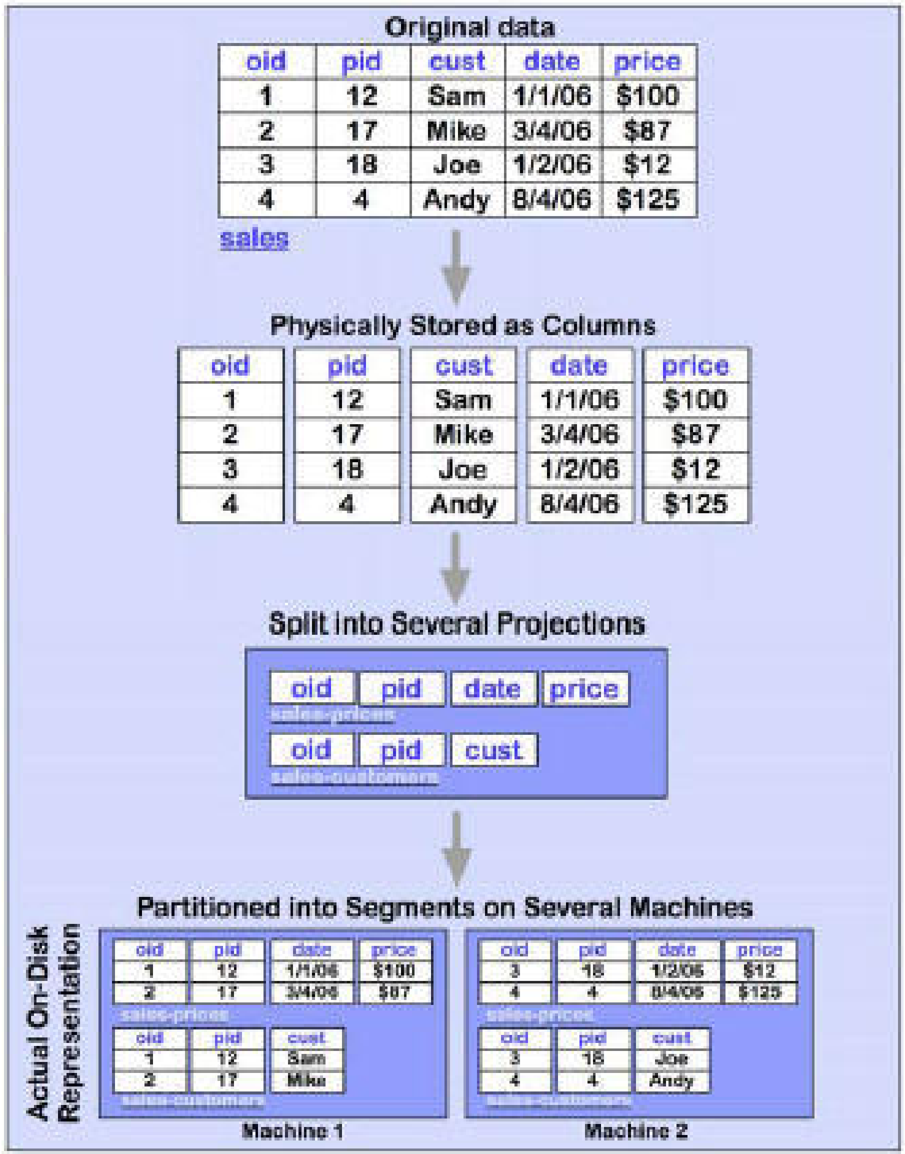
Table Partitioning and Distribution



- 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")

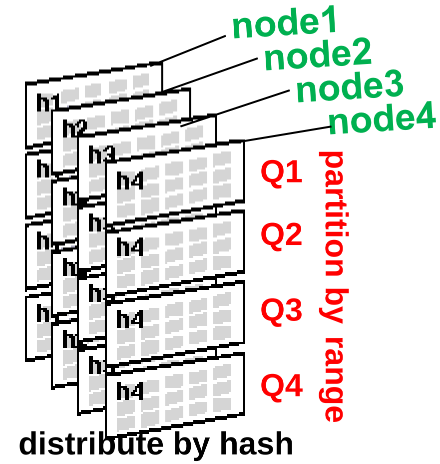
Which kind of function would you use for which method?

Vertica Multiple Orders (Projections)



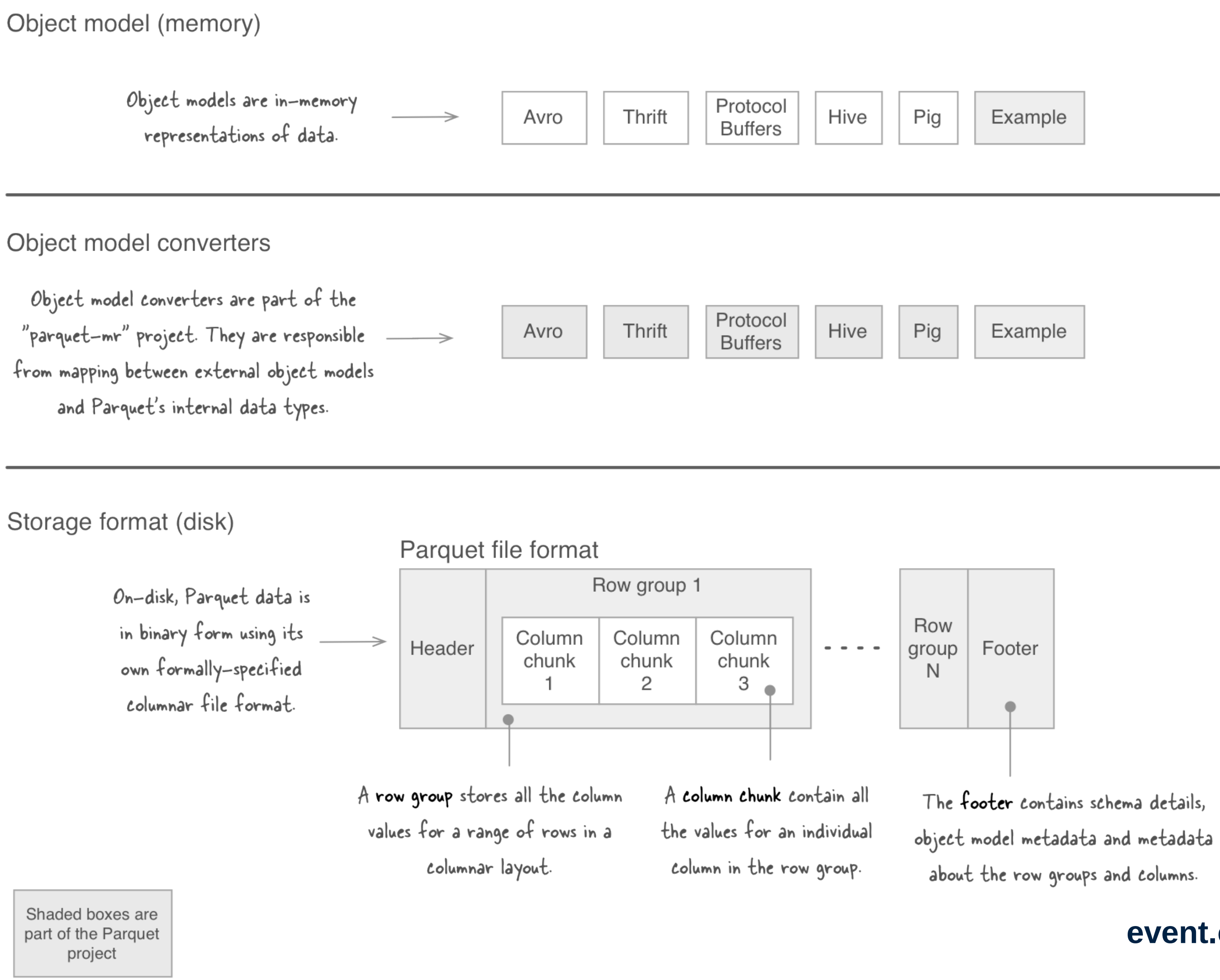
- Precomputed Projections reduce join effort
- Projections are ordered (e.g. on “date”, or on “cust”)
- Ordered data allows “selection pushdown”
  - Scan less data
- Ordered Data enhances compression
  - Run-length encoding
  - Frame of Reference

Data Placement in Hadoop



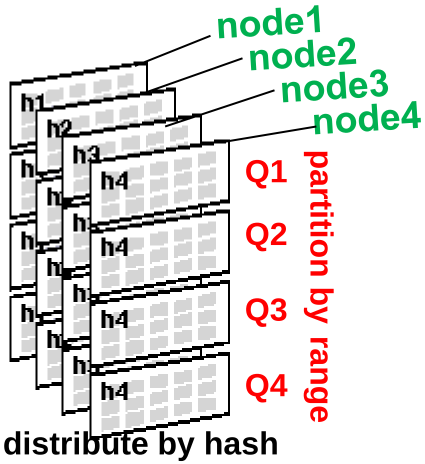
- Each node writes the partitions it owns
  - Where does the data end up, really?
- HDFS default block placement strategy:
  - Node that initiates writes gets first copy
  - 2nd copy on the same rack
  - 3rd copy on a different rack
- Rows from the same record should on the same node
  - Not entirely 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

Example: Parquet Format



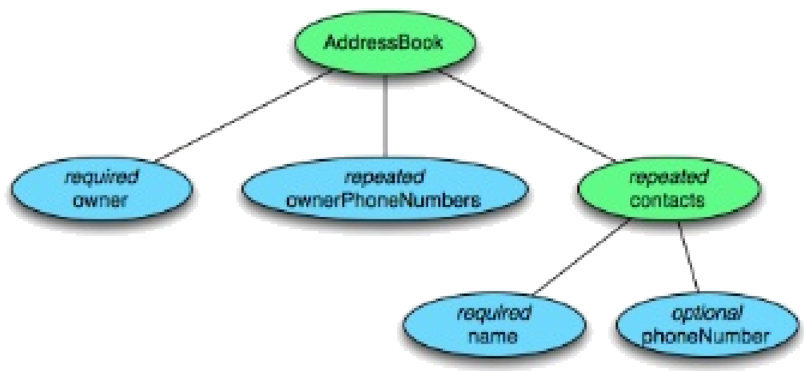
Popular File Formats in Hadoop

- 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



Example: Parquet Format

Table Format



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

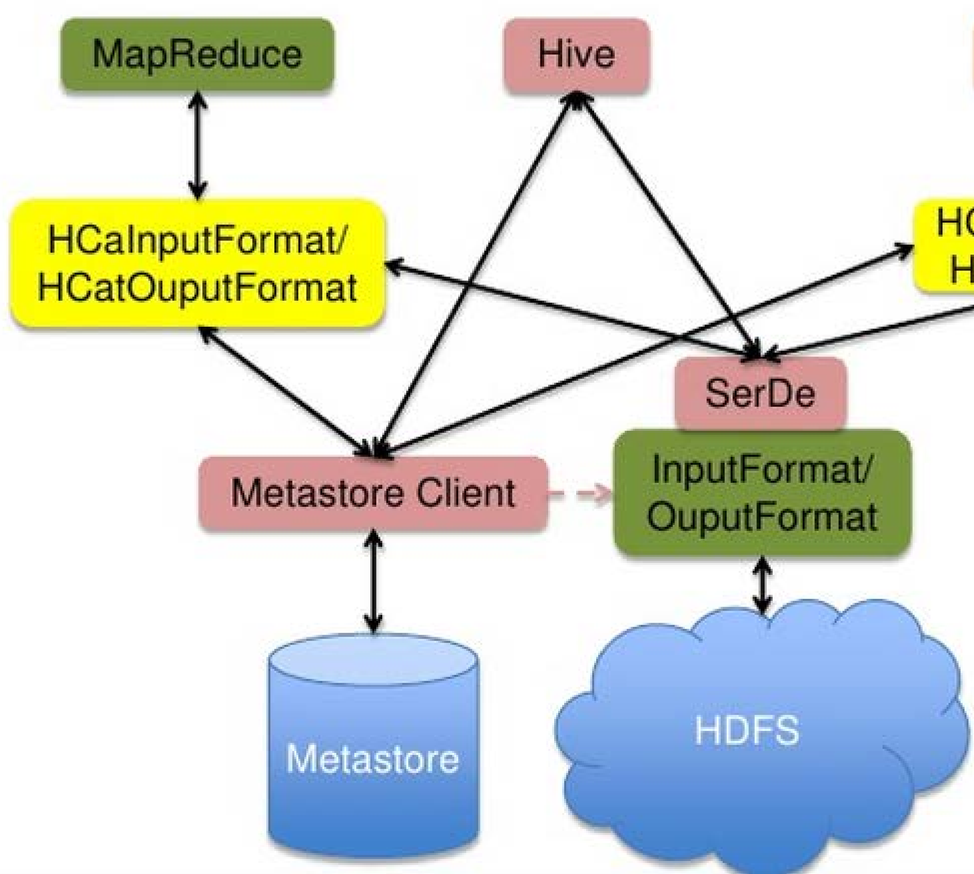
AddressBook			
owner	ownerPhoneNumbers	contacts	
		name	phoneNumber
...	...	...	...
...	...	...	...
...	...	...	...

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!

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

MinMax and Zone Maps

Q: acctno BETWEEN 150 AND 200?

- 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

Can we exploit correlation?

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

Accounts

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

bucket 0

bucket 1

bucket 2

bucket 3

Accounts.MinMax

bucket	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

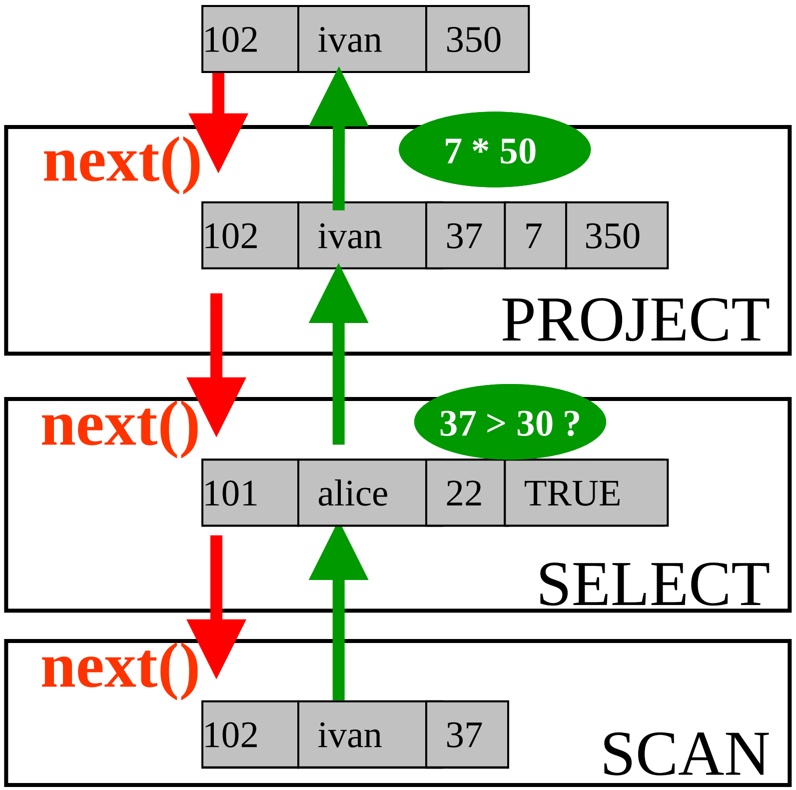
DBMS Computational Efficiency?

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

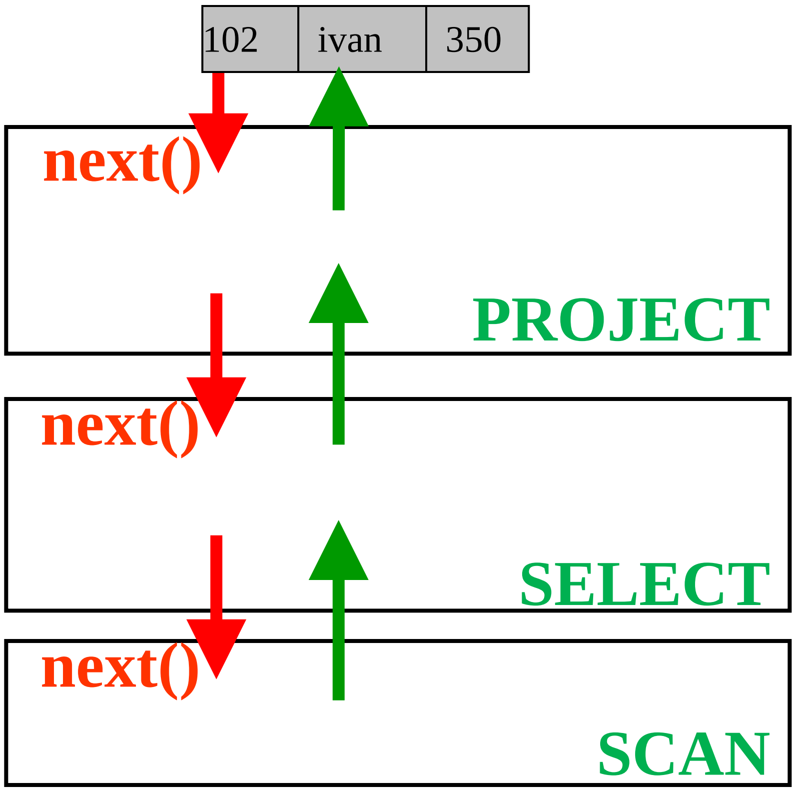
How Do Query Engines Work?



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

Centrum Wiskunde & Informatica

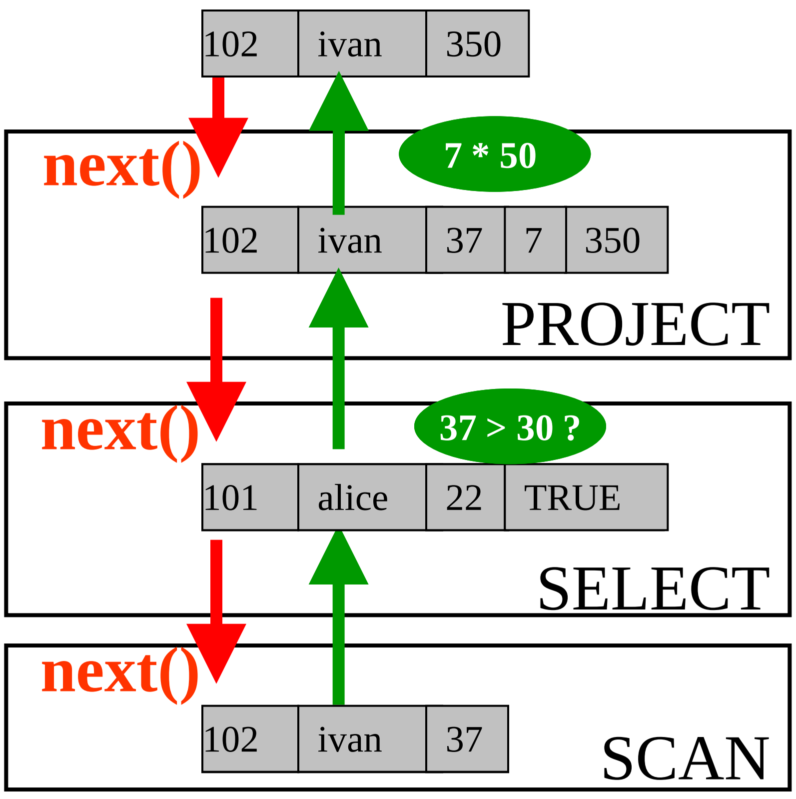
How Do Query Engines Work?



Operators

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

How Do Query Engines Work?



Primitives

Provide computational functionality

All arithmetic allowed in expressions,
e.g. Multiplication

7 * 50

`mult(int,int)` ➔ `int`

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

vectorwise

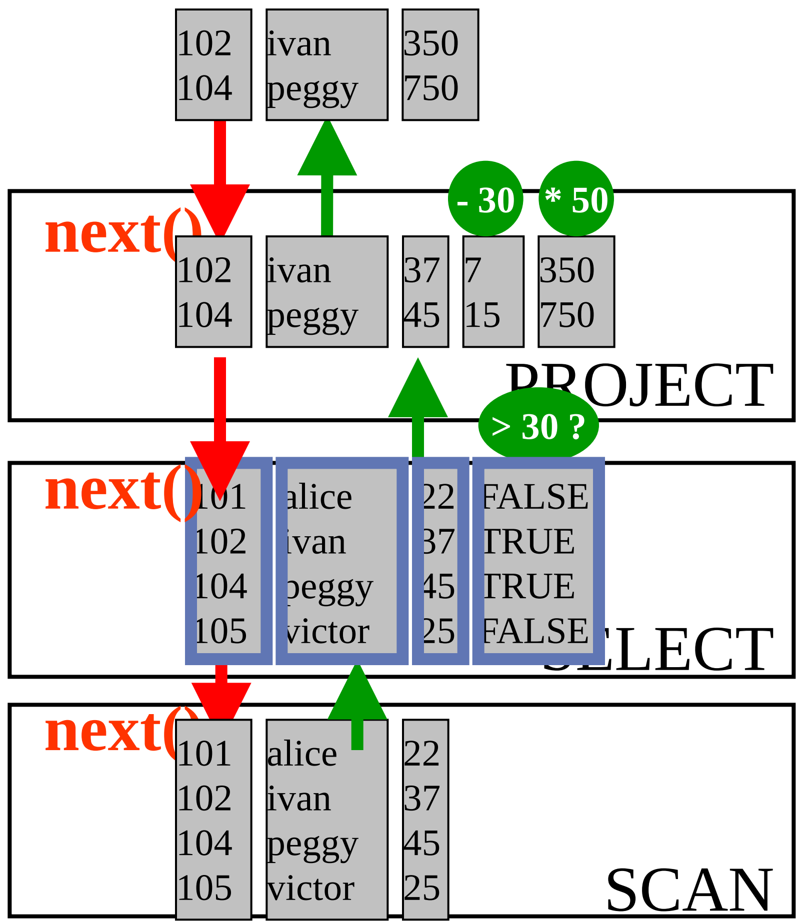
Observations:

“Vectorized In Cache Processing”

vector = array of ~100

processed in a tight loop

CPU cache Resident



vectorwise

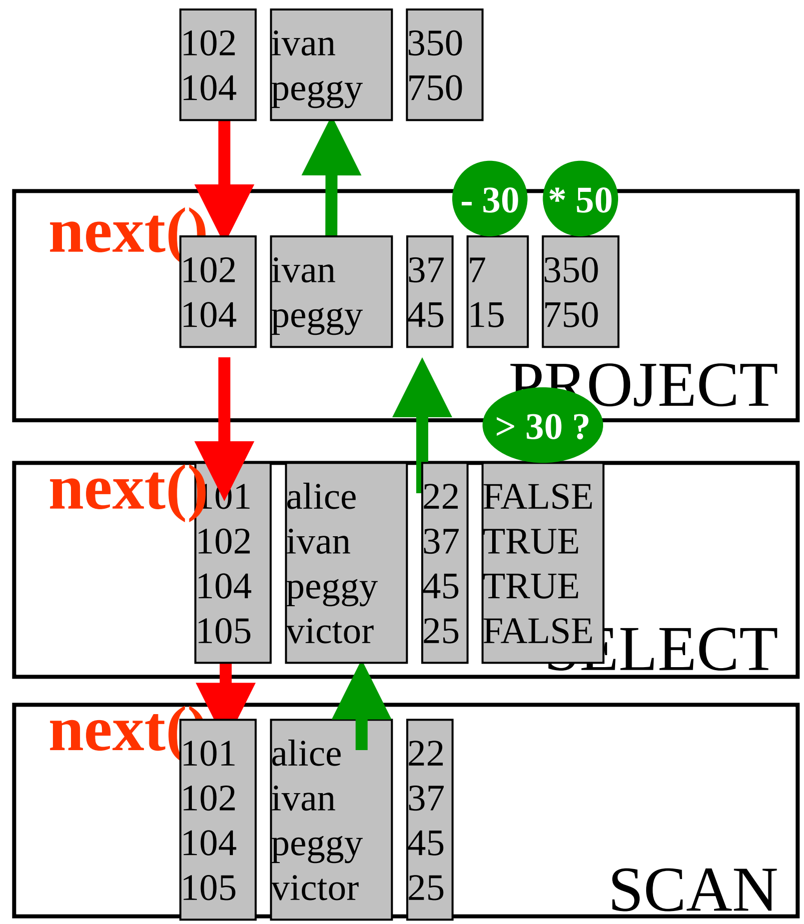
Observations:

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



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

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)
```

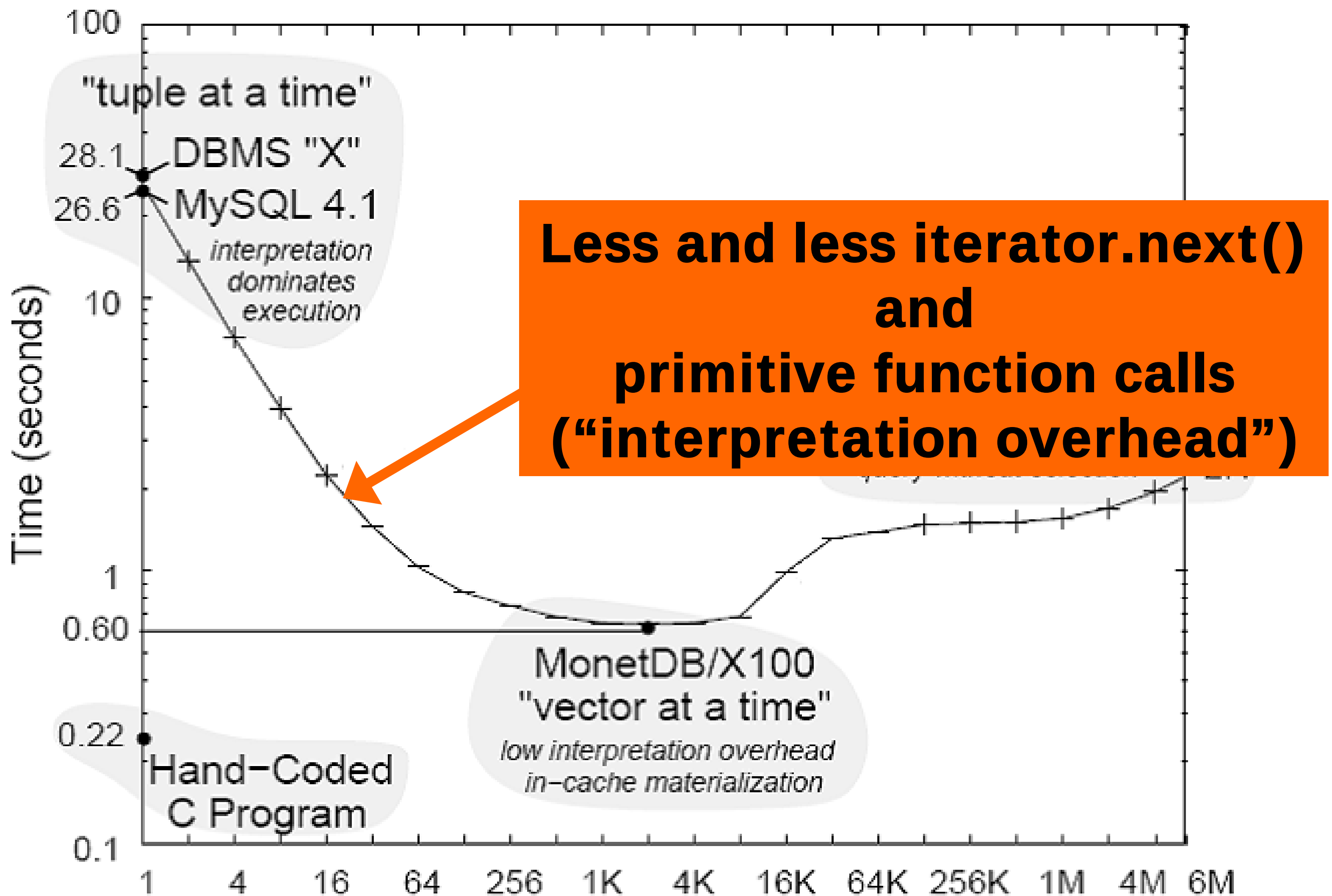
* 50

350
750

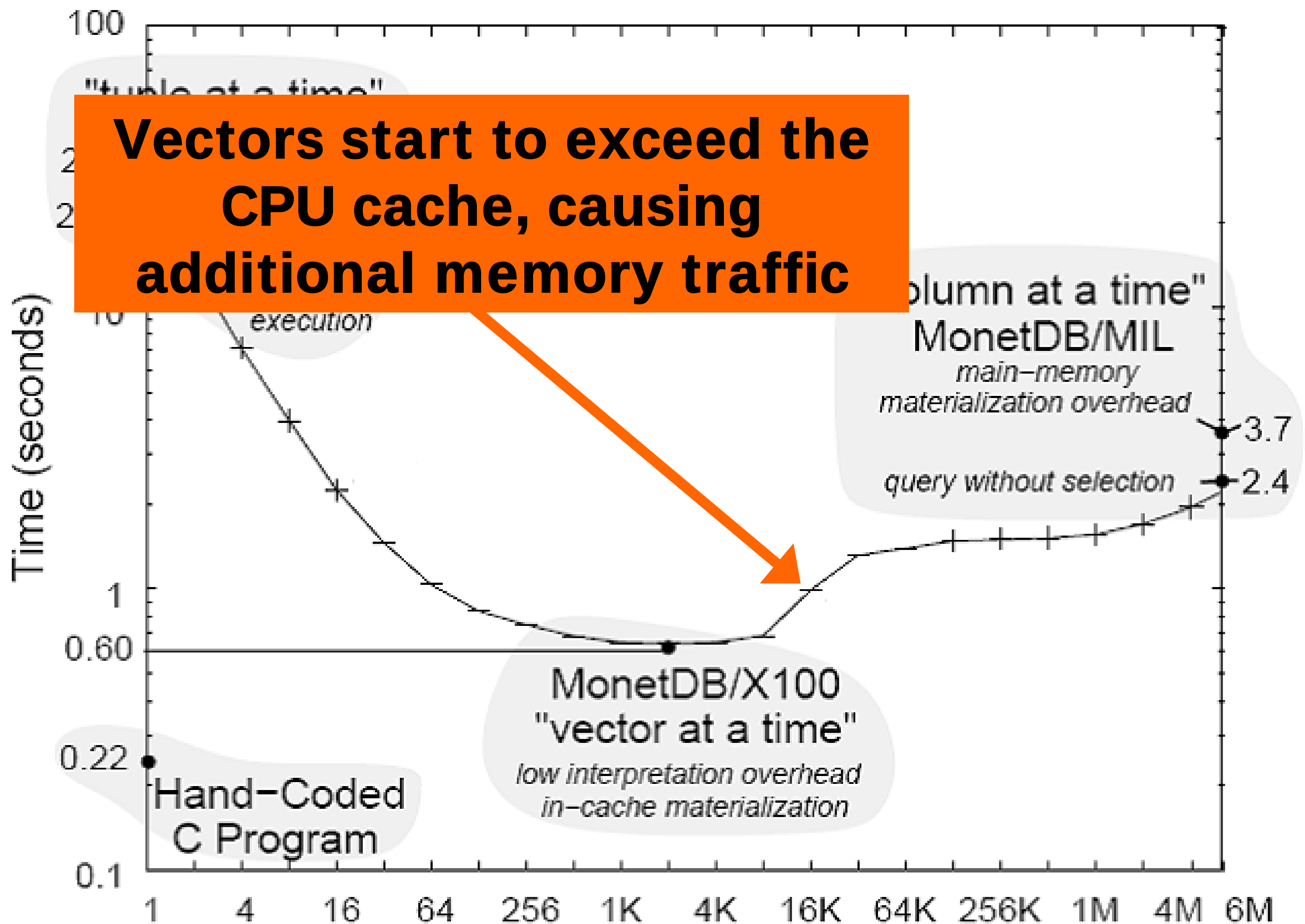
```
for(i=0; i<n; i++)
  res[i] = (col[i] * x)
```

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

Varying the Vector size



Varying the Vector size



Vectors start to exceed the CPU cache, causing additional memory traffic

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

Benefits of Vectorized Processing

"Block oriented processing of relational database operations in modern computer architectures" Padmanabhan, Malkemus, Agarwal, ICDE'01

- **Less Interpretation Overhead**
  - iterator.next(), primitives
  - Array-only, no complex record navigation
- **Compiler-friendly primitive code**
  - Move activities out of the loop ("strength reduction")
  - Loop-pipelining, automatic SIMD generation by the compiler

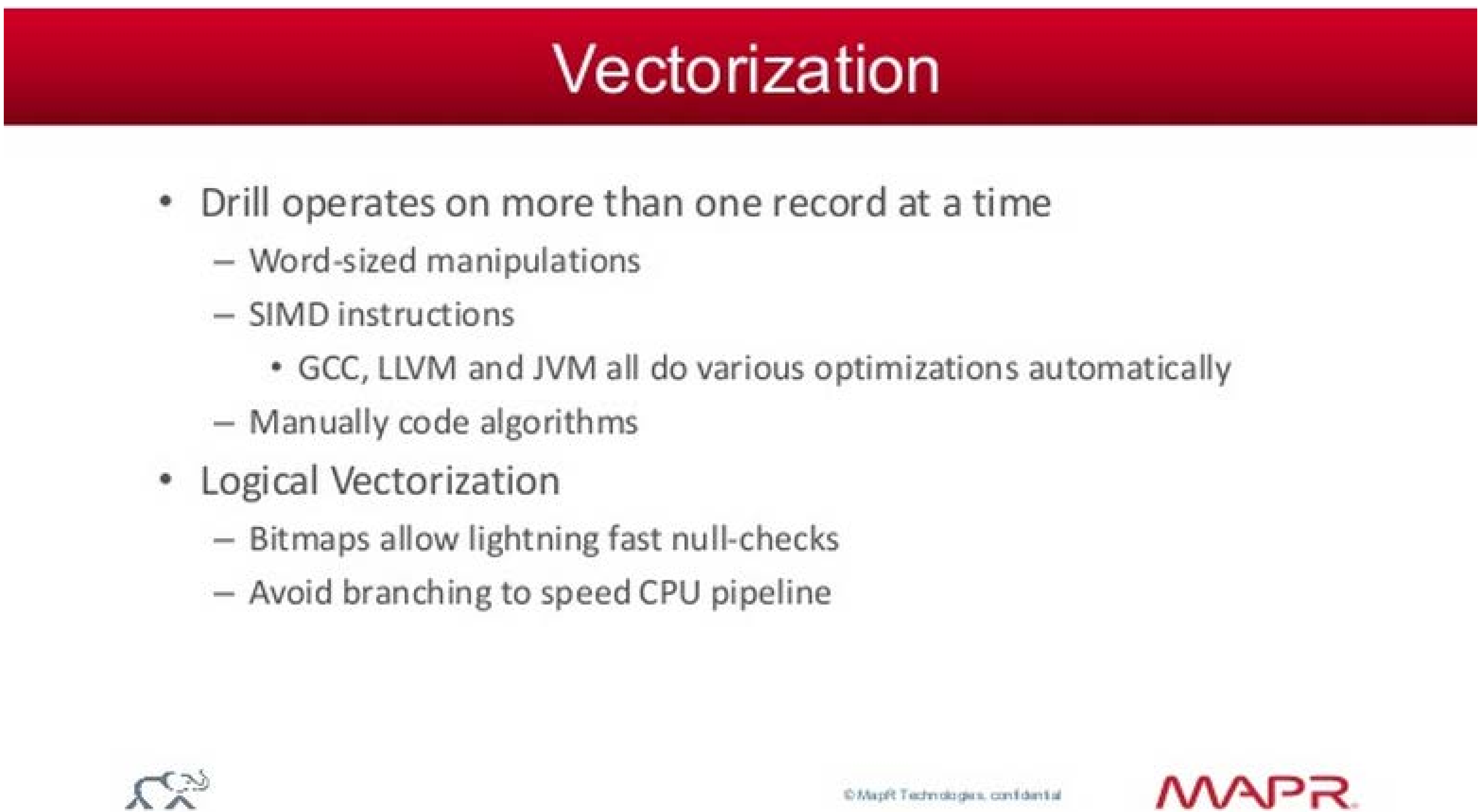
Buffering Database Operations for Enhanced Instruction Cache Performance" Zhou, Ross, SIGMOD'04

- **Less Cache Misses**
  - High instruction cache locality in the primitives
  - Data-Cache friendly sequential data placement
- **Profiling and Adaptivity**
  - Performance bookkeeping cost amortized over an entire vector
  - stats can be exploited during the query to select fastest primitive variants

Micro-adaptivity in Vectorwise, Raducanu, Zukowski, Boncz, SIGMOD'13

Systems That Use Vectorization

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



Analytical DB engines for Hadoop

storage

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

query-processor

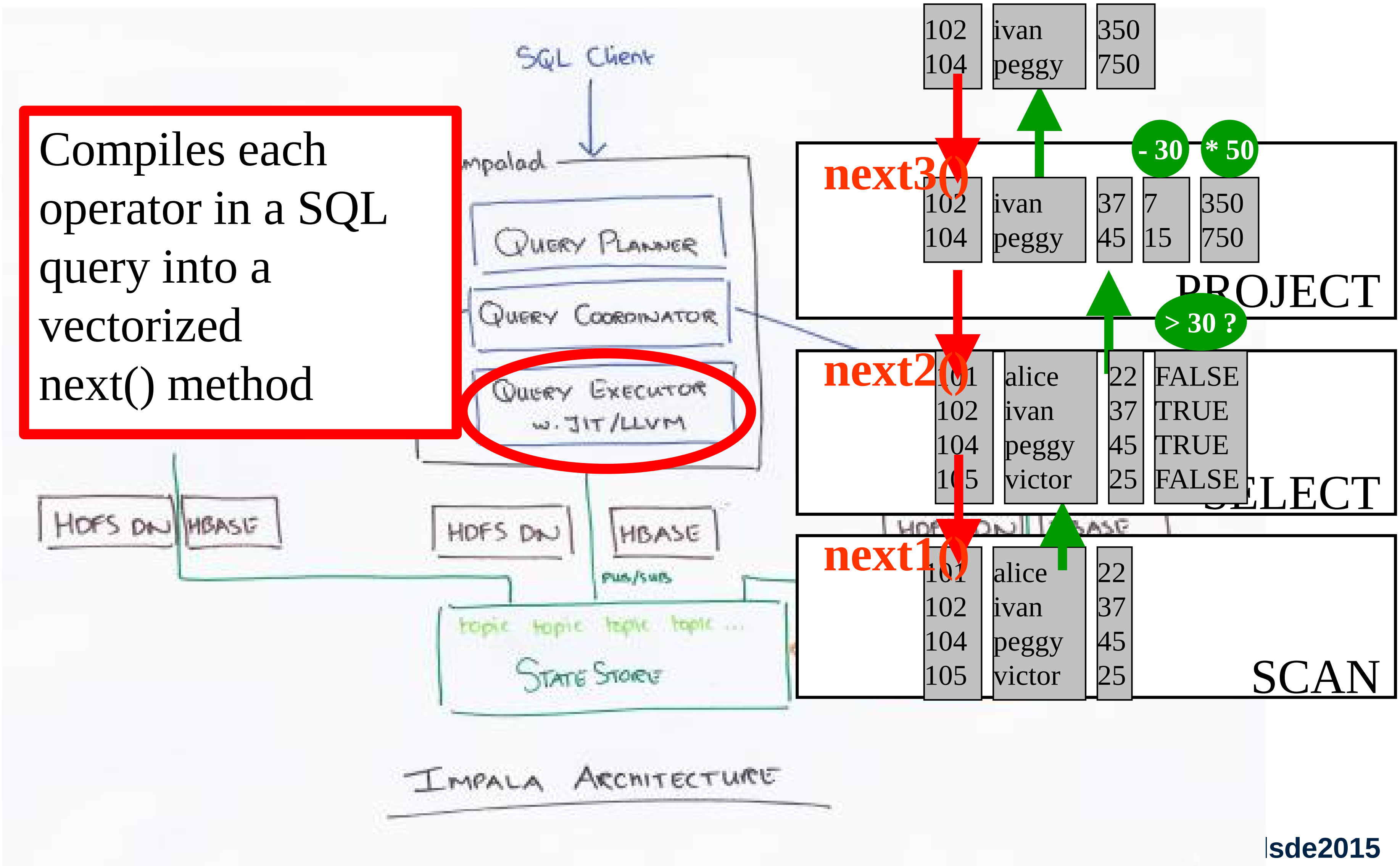
- **CPU-efficient query engine** (vectorized or **JIT codegen**)
- many-core ready
- rich SQL (+authorization+..)

system

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

Impala: Just In Time SQL➔LLVM (~asm)

Compiles each operator in a SQL query into a vectorized next() method



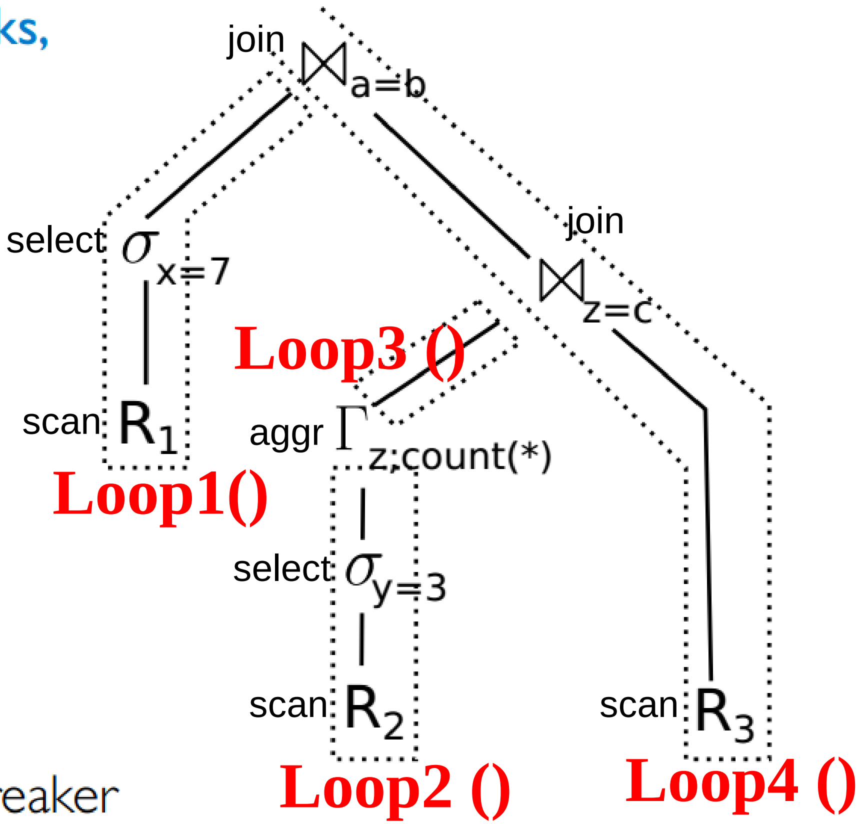
Hyper-db.de: compilation across operators

Main memory is so fast that CPU becomes the bottleneck

- classical iterator model fine for disks, but not so for main memory
- iterator model: many branches, bad code and data locality

HyPer's data-centric code generation

- touches data as rarely as possible
- prefers tight work loops
  1. load data into CPU registers
  2. perform all pipeline operations
  3. materialize into next pipeline breaker



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
- analytical SQL (windowing)

system

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

Batch Update Infrastructure (Vertica)

Challenge: hard to update columnar compressed data

Trickle Load

> Write Optimized Store (WOS)

A B C

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)

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

Batch Update Infrastructure (Hive)

Challenge: HDFS read-only + large block size

Merge During Query Processing

Base File

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

Logical File

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

Hortonworks

Batch Update Infrastructure

Hive (Spinner release) HDFS Layout:

- **Partition locations remain unchanged**
  - –Still warehouse/$db/$tbl/$part
- **Bucket Files Structured By Transactions**
  - –Base files $part/base_$tid/bucket_*
  - –Delta files $part/delta_$tid_$tid/bucket_*
- **Minor Compactions merge deltas**
  - –Read delta_$tid1_$tid1 .. delta_$tid2_$tid2
  - –Written as delta_$tid1_$tid2
- **Compaction doesn't disturb readers**

Hortonworks

Batch Update Infrastructure

Hive (Spinner release) HDFS Layout:

- **Partition locations remain unchanged**
  - –Still warehouse/$db/$tbl/$part
- **Bucket Files Structured By Transactions**
  - –Base files $part/base_$tid/bucket_*
  - –Delta files $part/delta_$tid_$tid/bucket_*
- **Minor Compactions merge deltas**
  - –Read delta_$tid1_$tid1 .. delta_$tid2_$tid2
  - –Written as delta_$tid1_$tid2
- **Compaction doesn't disturb readers**

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
```

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

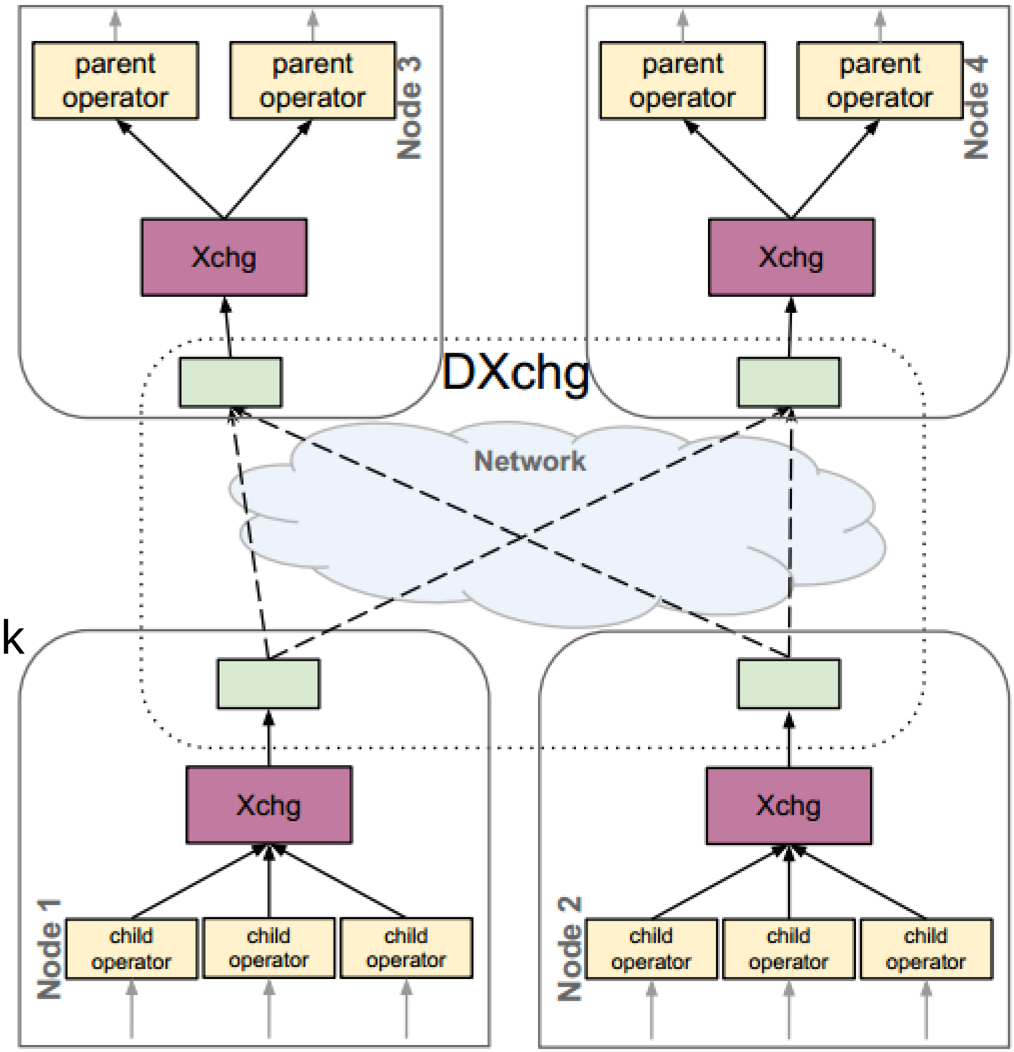
Scaling Through Parallel Query Processing

Scalability is hard!

- Core Contention
- Network Latency&Bandwidth

..Amdahls Law

- All nodes work on the query
  - Partitioning
  - ExCHanGe data over network
- All cores in a node work
  - Divide each partition (how?)



Analytical DB engines for Hadoop

storage

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

query-processor

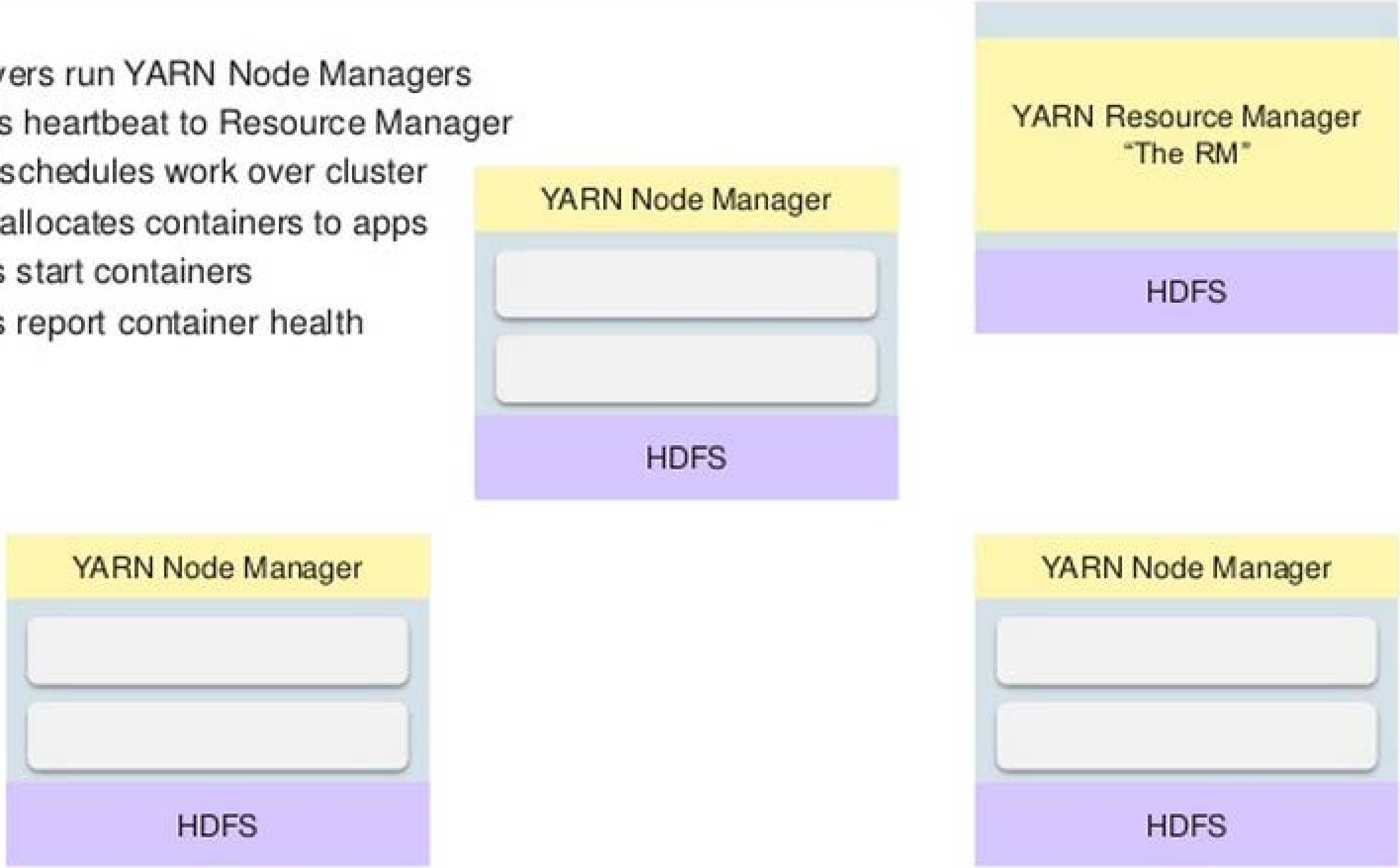
- CPU-efficient query engine (vectorized or JIT codegen)
- many-core ready
- rich SQL (+authorization+..)

system

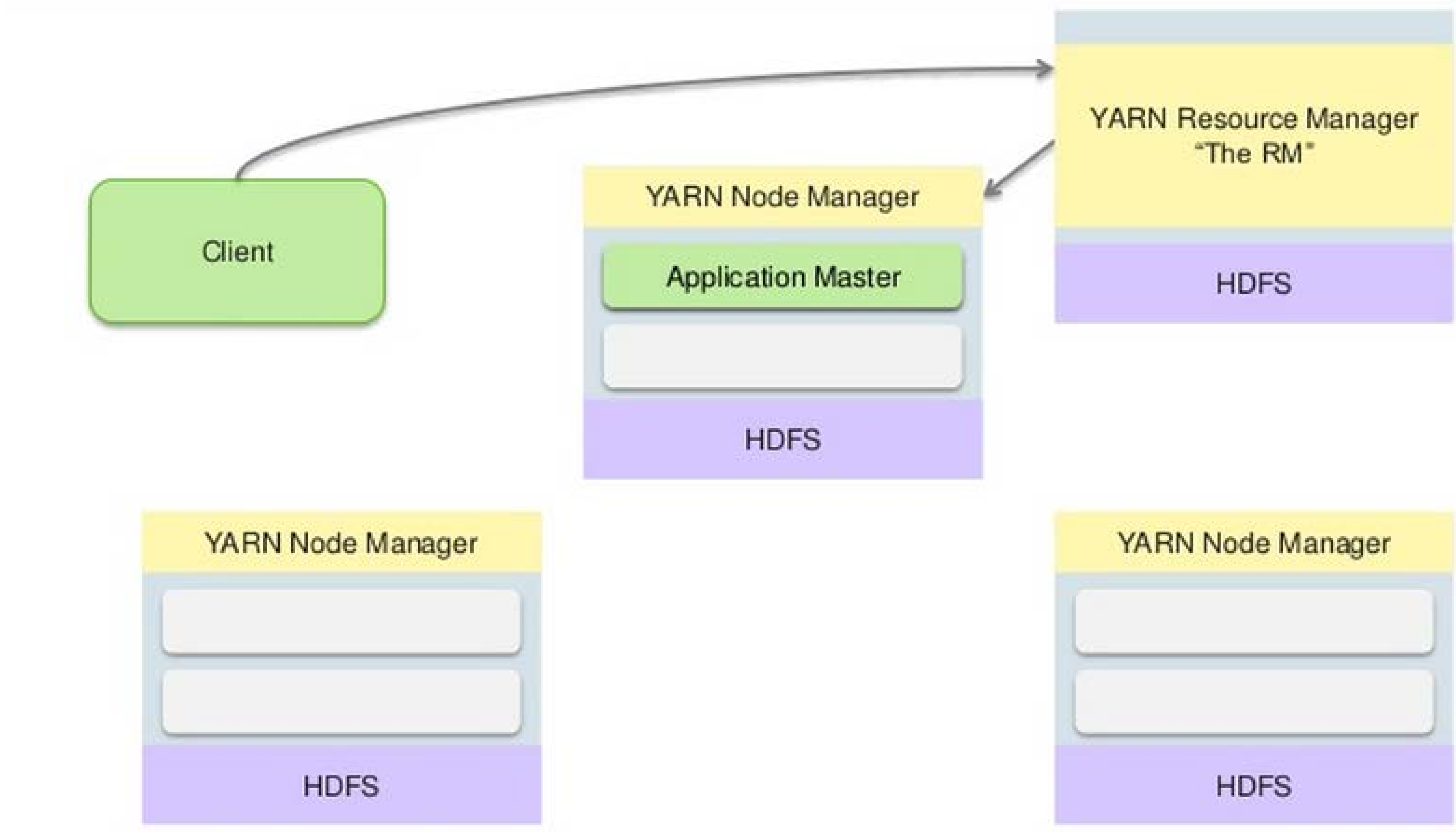
- batch update infrastructure
- scaling with multiple nodes
- MetaStore & file formats
- **YARN & elasticity**

YARN runs on all nodes in the cluster

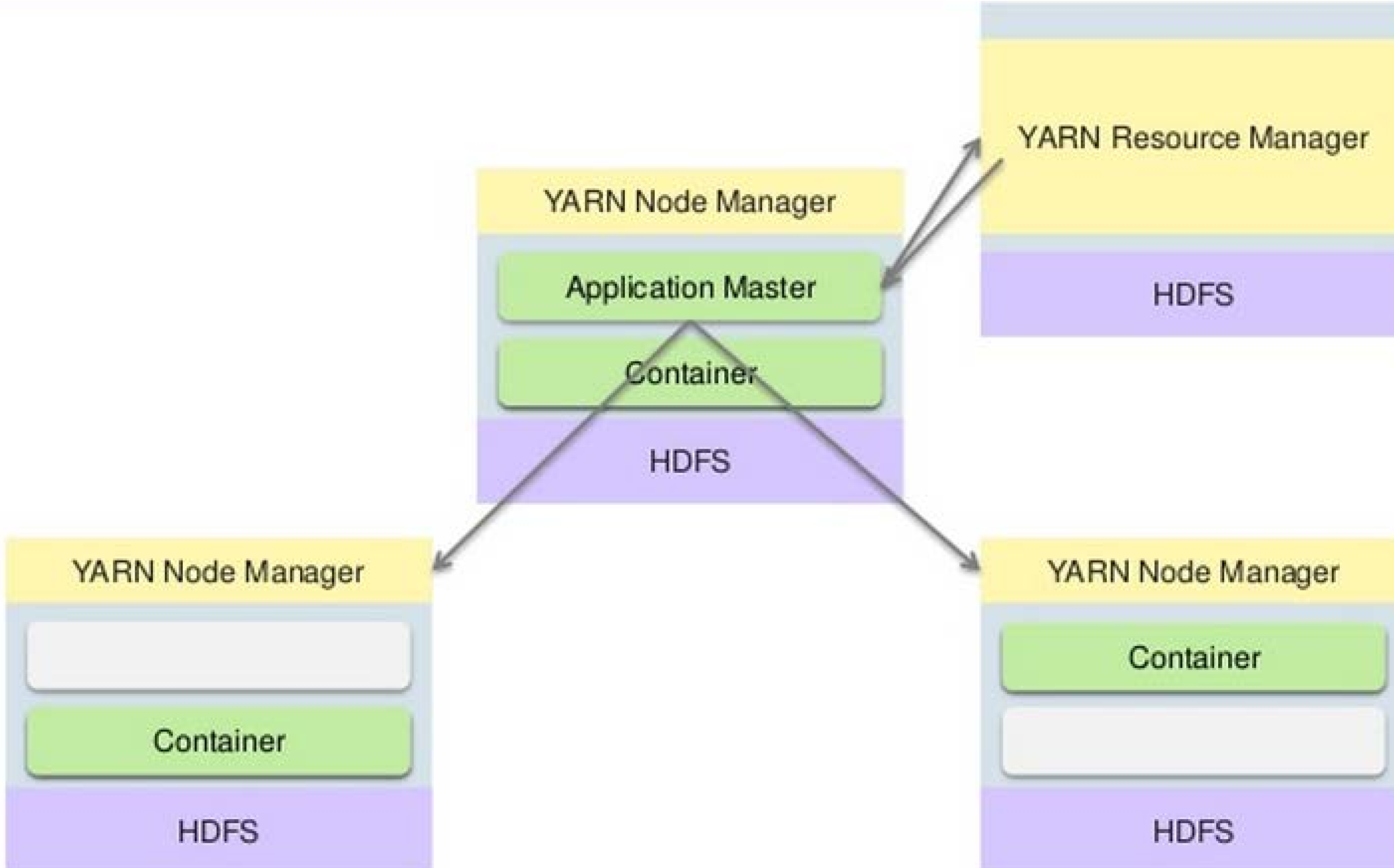
- Servers run YARN Node Managers
- NM's heartbeat to Resource Manager
- RM schedules work over cluster
- RM allocates containers to apps
- NMs start containers
- NMs report container health



Client creates Application Master



Application Master asks for Containers



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