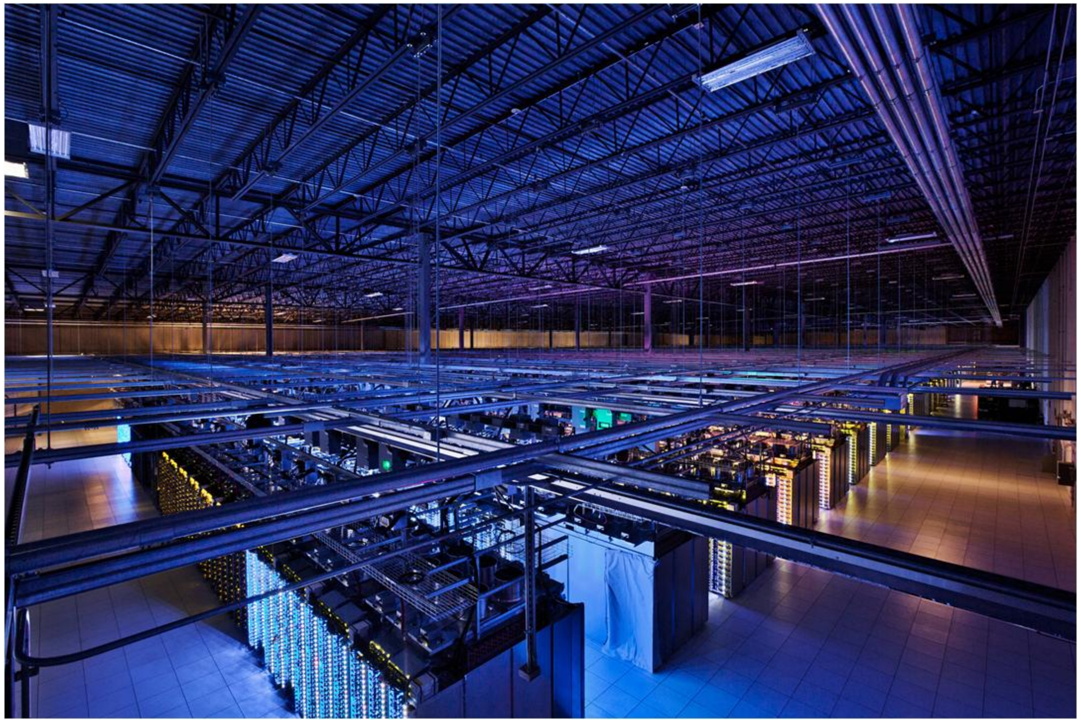
Big Data for Data Science

SQL on Big Data



www.cwi.nl/~boncz/bigdatacourse

THE DEBATE: DATABASE SYSTEMS VS MAPREDUCE

Centrum Wiskunde & Informatica

A major step backwards?

- 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

Michael Stonebraker
Turing Award Winner 2015

Known and unknown unknowns

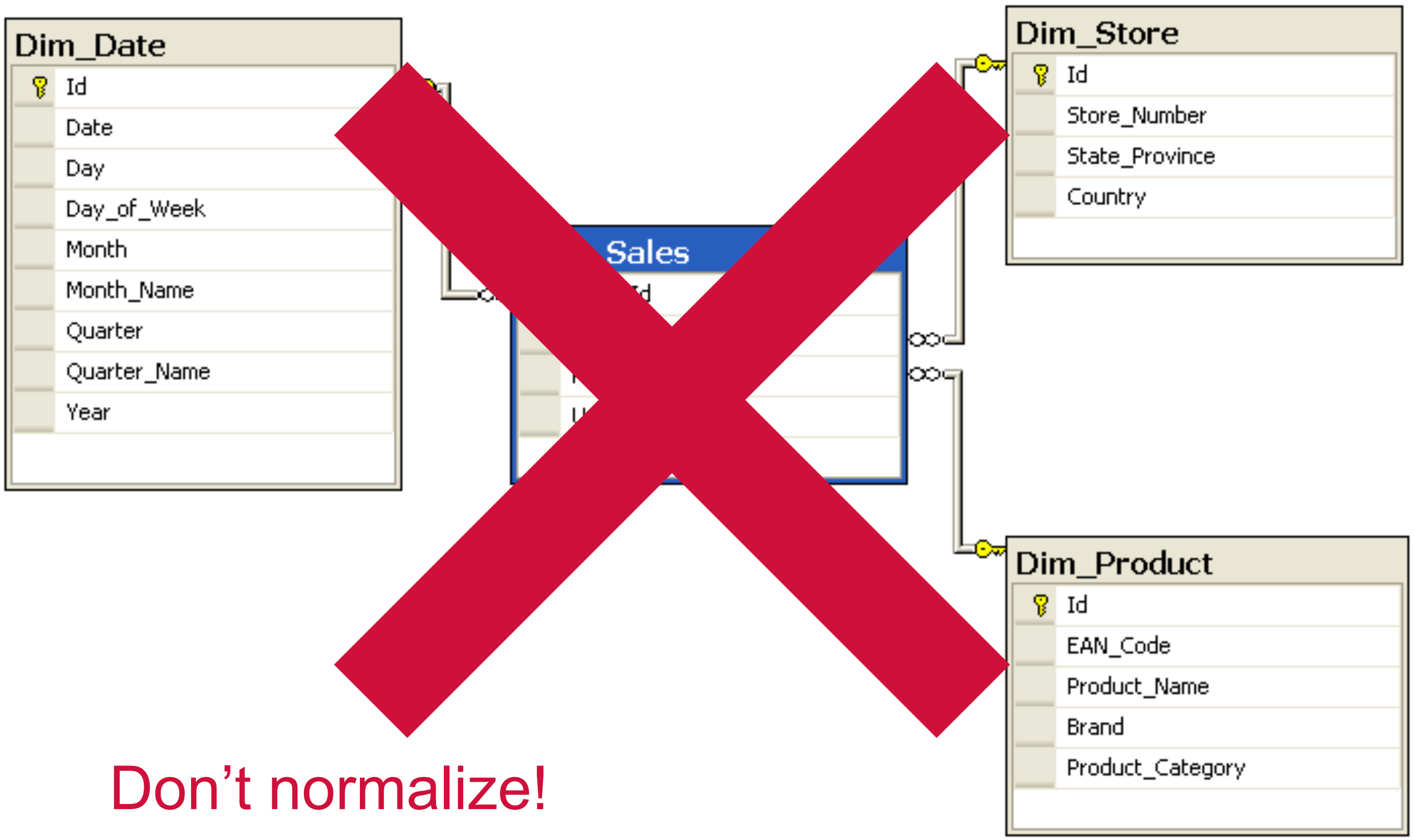
- Databases only help if you know what questions to ask
  - "Known unknowns"
- What's if you don't know what you're looking for?
  - "Unknown unknowns"

Centrum Wiskunde & Informatica

ETL: redux

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

Structure of Hadoop warehouses



Don't normalize!

Source: Wikipedia (Star Schema)

Relational databases vs. MapReduce

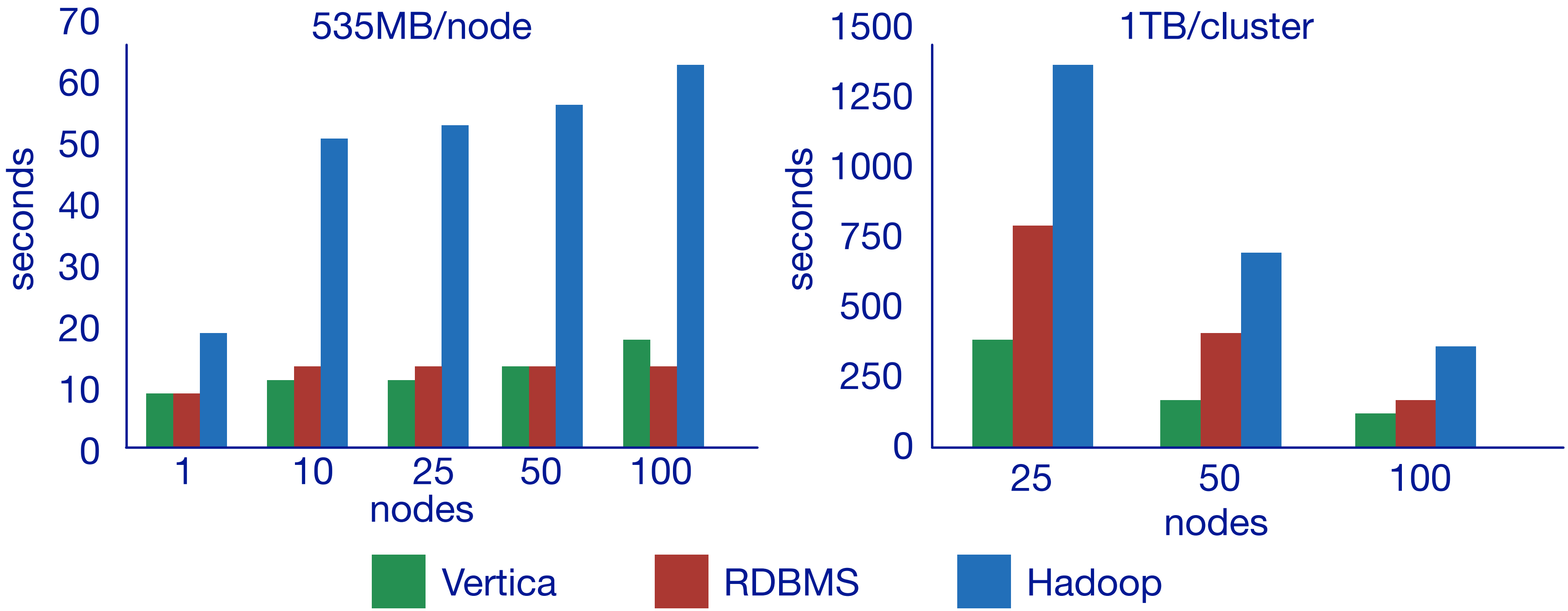
- Relational databases:
  - Multipurpose: analysis and transactions; batch and interactive
  - Data integrity via ACID transactions
    - ACID = Atomicity, Consistency, Isolation, Durability
  - Lots of tools in software ecosystem (for ingesting, reporting, etc.)
  - SQL: query language, automatic query optimization
- MapReduce (Hadoop):
  - Designed for large clusters, fault tolerant
  - Data is accessed in “native format”
  - 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

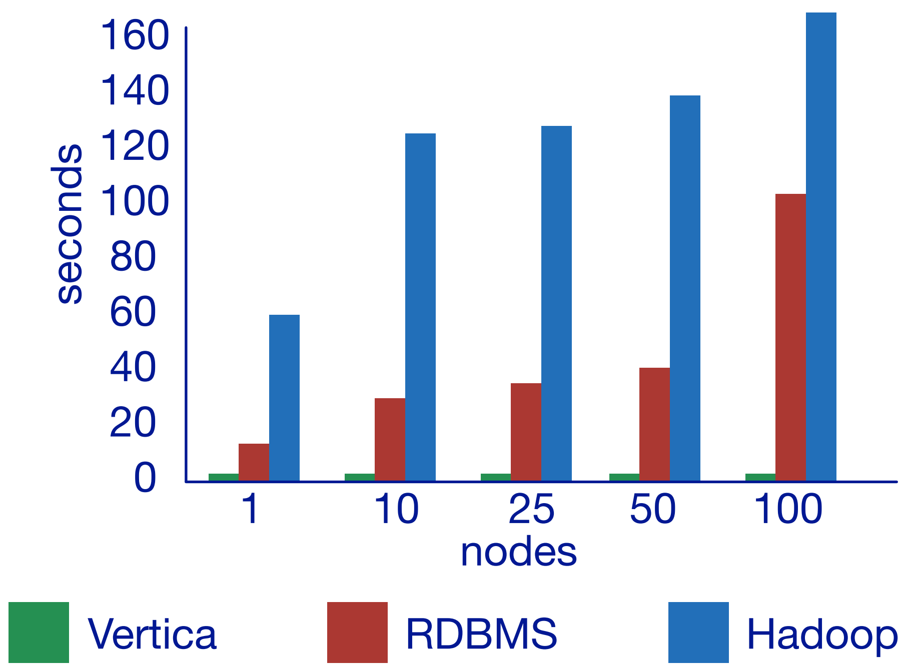
MapReduce vs. RDBMS: grep




```
SELECT * FROM Data WHERE field LIKE ‘%XYZ%’;
```


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

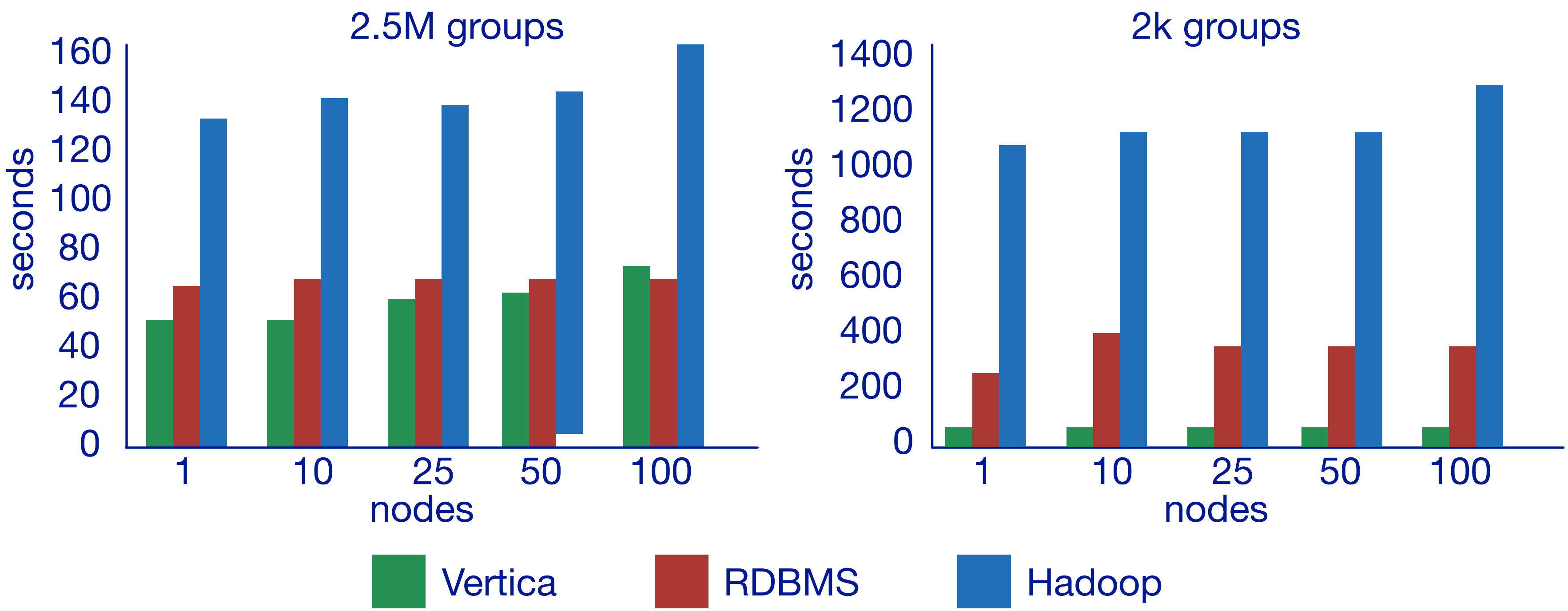
MapReduce vs. RDBMS: select




```
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

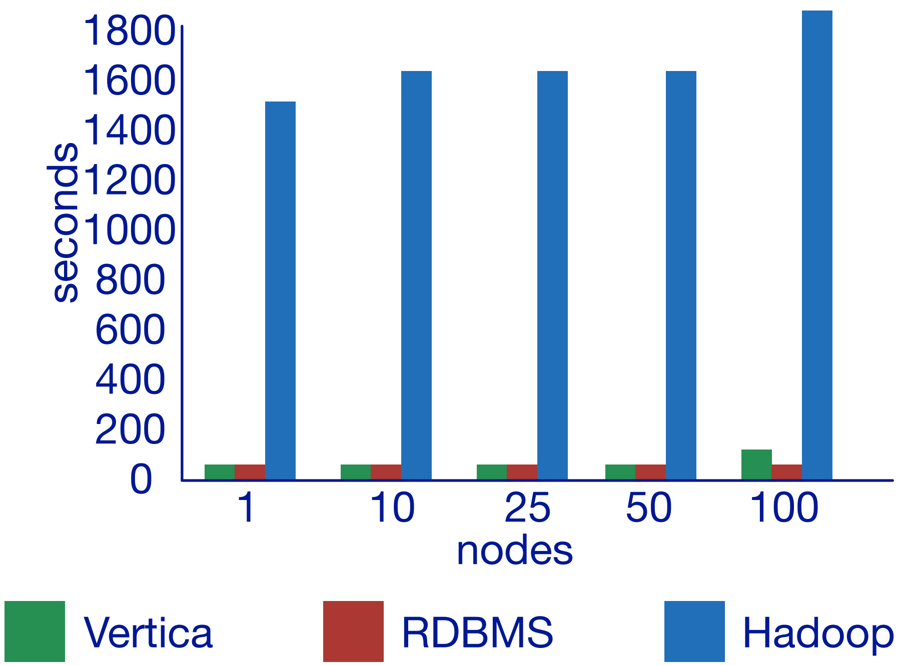



```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY sourceIP;
```

www.cwi.nl/~boncz/bigdatacourse

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

MapReduce vs. RDBMS: join

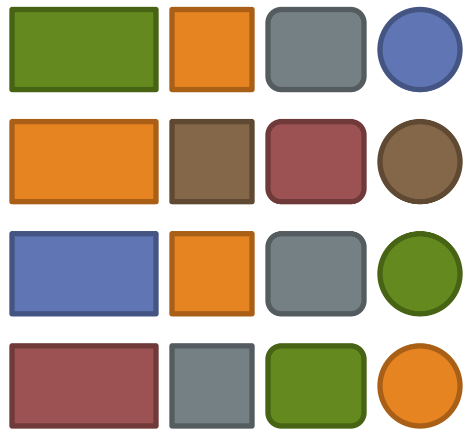


Source: Pavlo et al. (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD.

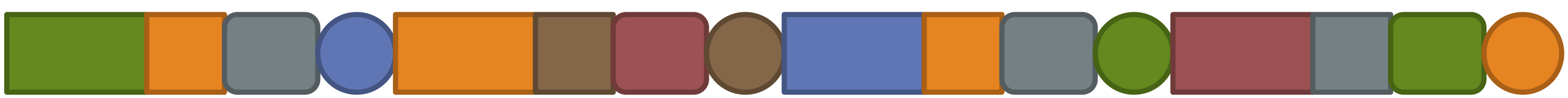
Why?

- 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

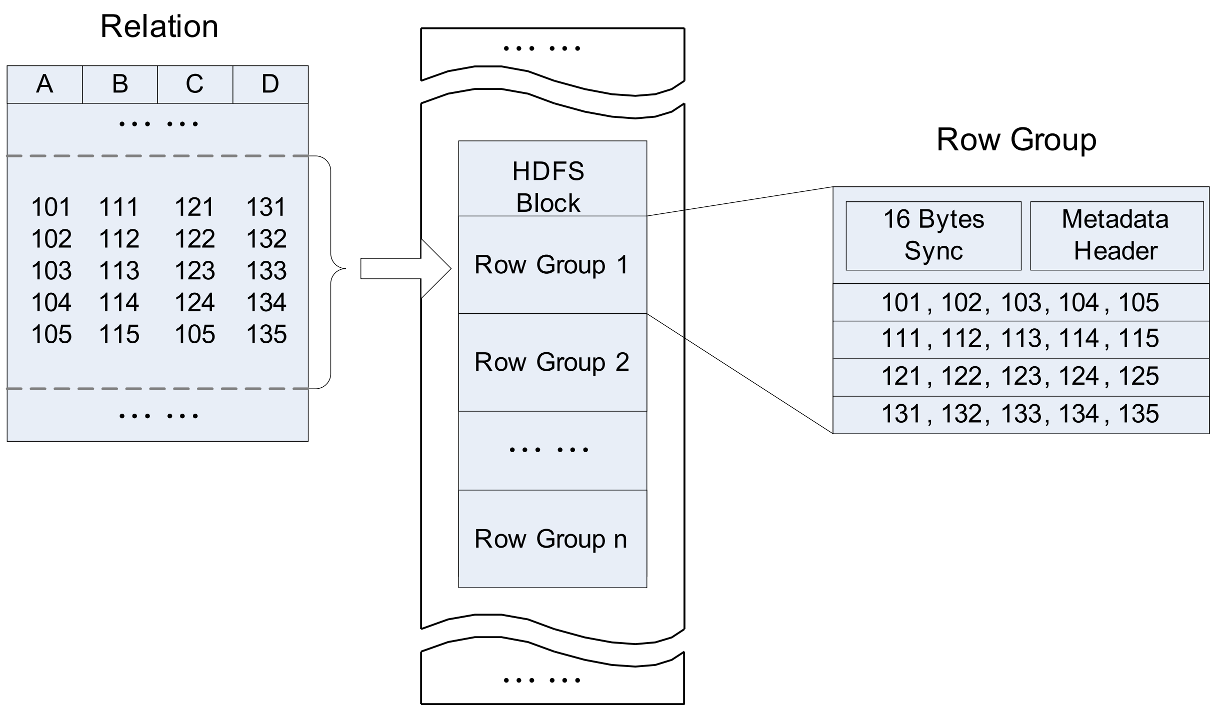
Advantages of column stores

- 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

CWI
Centrum Wiskunde & Informatica

Why not in Hadoop?

RCFile



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

RCFile ➔ ORC, Parquet (+compression)

BIG DATA SQL SYSTEMS

Big SQL System Architecture

storage

- –**columnar storage** + compression
- –table partitioning / distribution
- –clustering and indexing

query-processor

- vectorized or JIT codegen
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

cluster

- (meta-) data sharing
- elastic resource provisioning
- continous update infrastructure

Centrum Wiskunde & Informatica

Big SQL System Architecture

storage

- columnar storage + **compression**
- table partitioning / distribution
- clustering and indexing

query-processor

- vectorized or JIT codegen
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

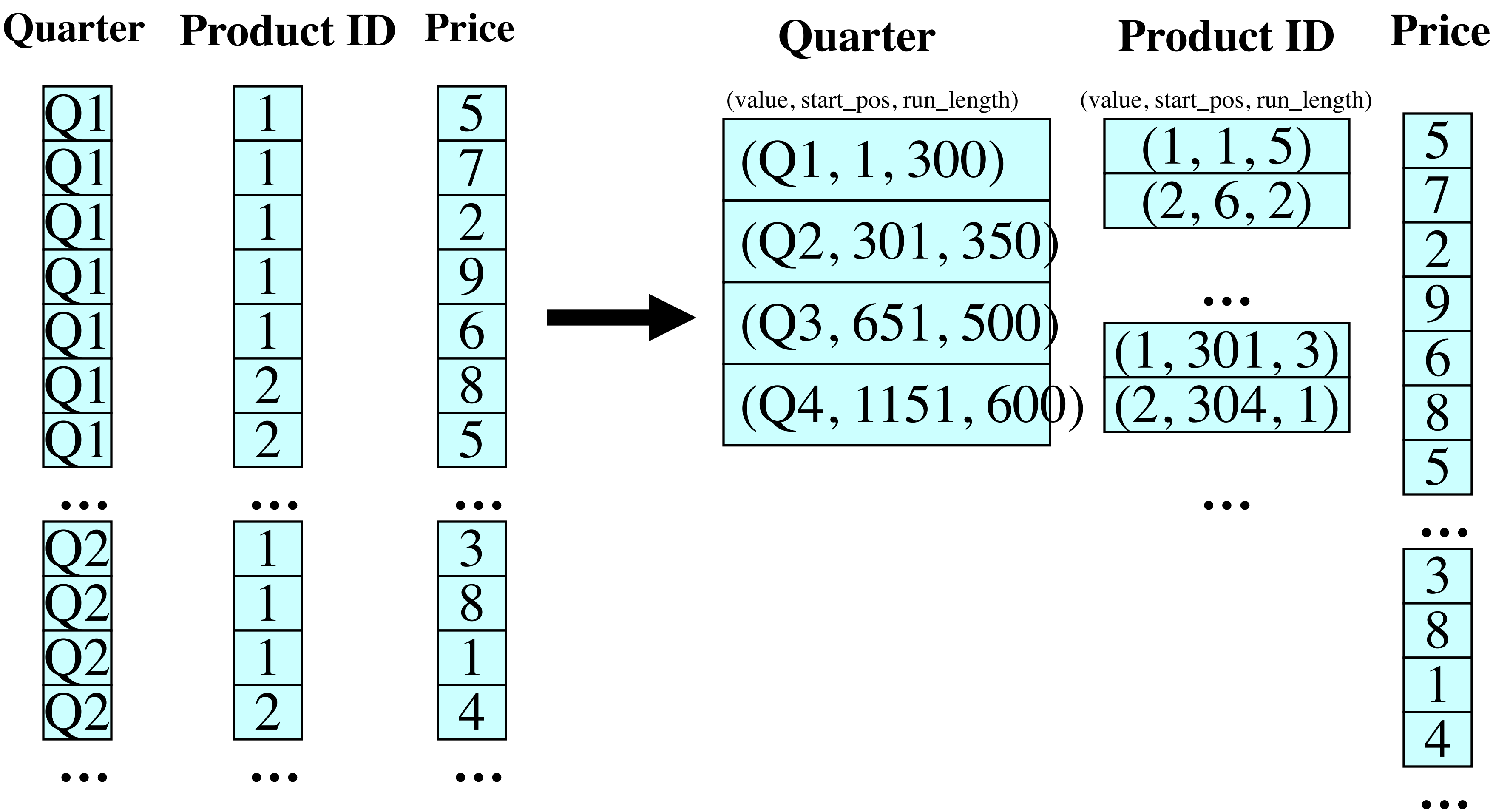
cluster

- (meta-) data sharing
- elastic resource provisioning
- continous update infrastructure

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 (Vertica approach)

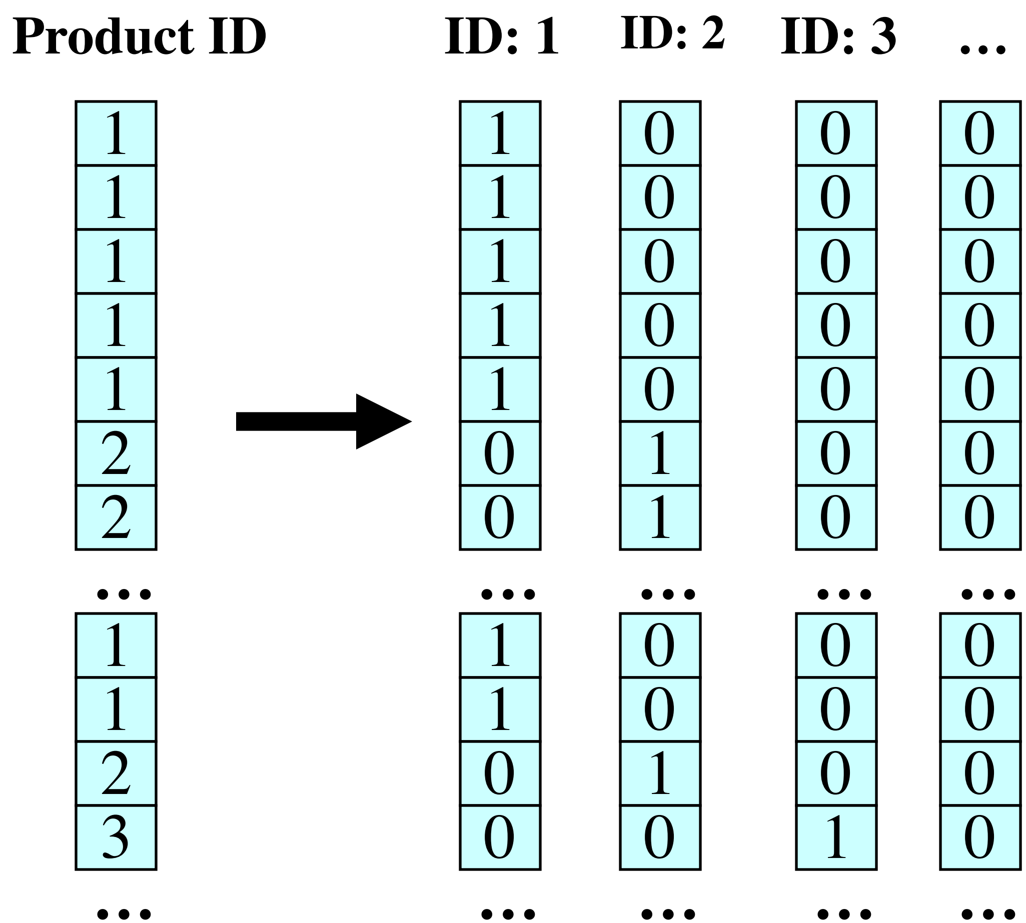
Run-length Encoding



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

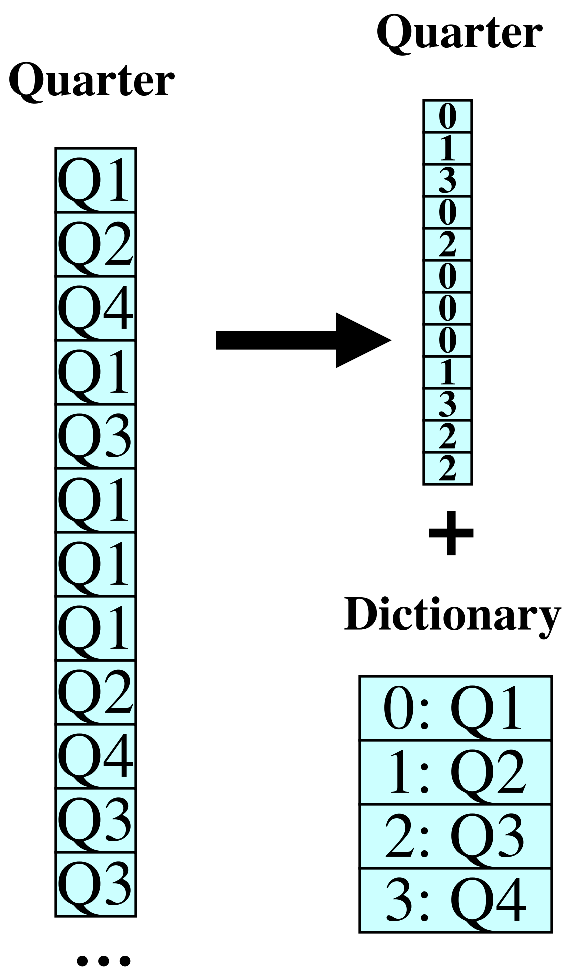
Bitmap Encoding

- 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



Dictionary Encoding

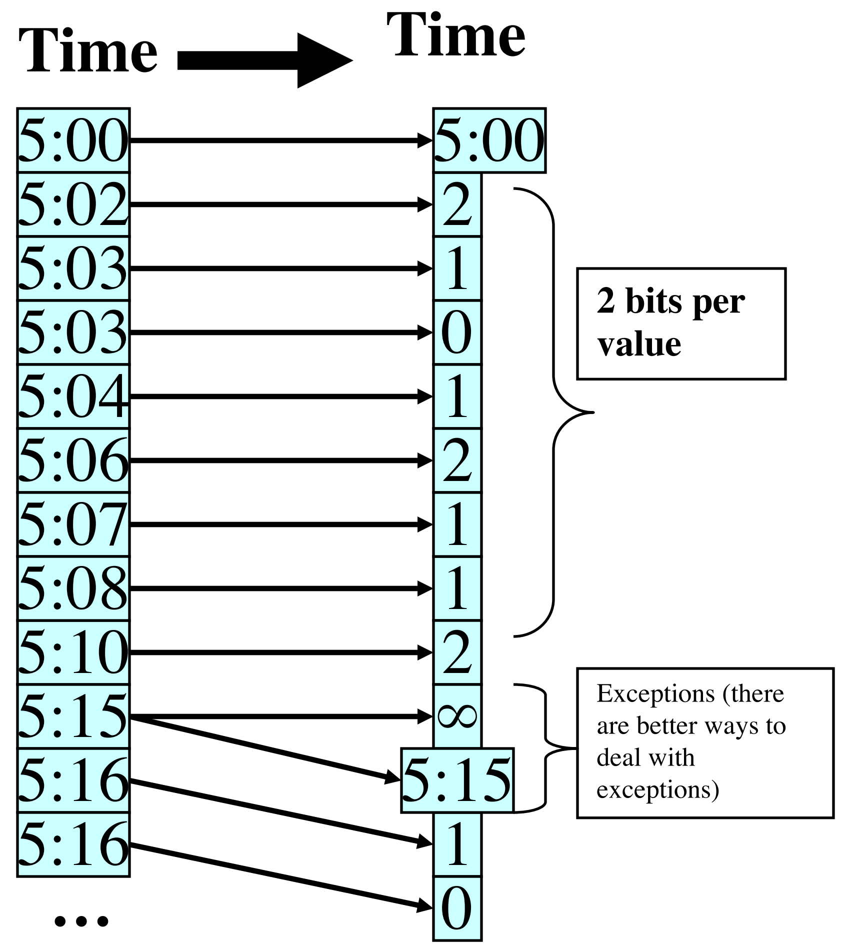
- For each unique value create dictionary entry
- Dictionary can be per-block or per-column
- Column-stores have the advantage that dictionary entries may encode multiple values at once



CWI
Centrum Wiskunde & Informatica

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

CWI
Centrum Wiskunde & Informatica

Big SQL System Architecture

storage

– columnar storage + compression
– table partitioning / distribution
– **clustering and indexing**

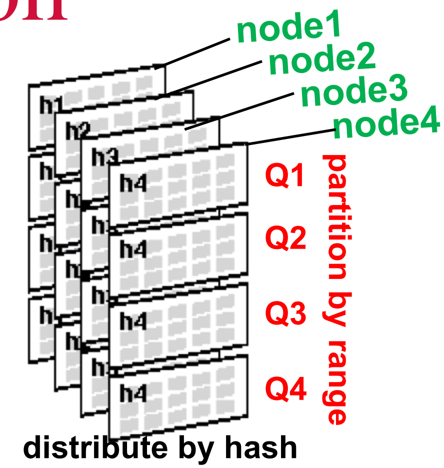
query-processor

- vectorized or JIT codegen
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

cluster

- (meta-) data sharing
- elastic resource provisioning
- continous update infrastructure

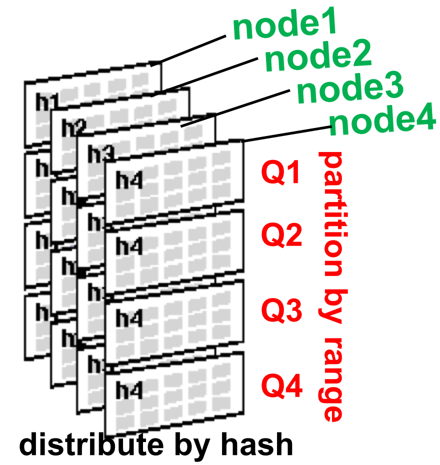
Table Partitioning and Distribution



- data is spread based on a Key
  - Functions: Hash, Range, Value
- "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")

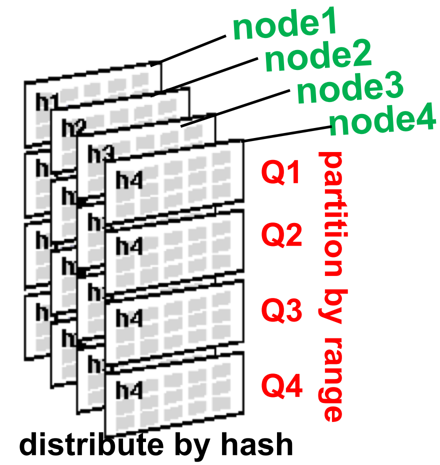
Which kind of function would you use for which method?

Data Placement in HDFS



- 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 be 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

Data Placement in the Cloud?



- Cloud storage (S3, Azure Blob Storage) provides **no locality**
  - High latency (100-200msec)
  - Slow-medium bandwidth (20-125MB/s)
- Partitioning and Distribution still make sense
  - Distribution: allow jobs to be parallelized
  - Partitioning: partition-pruning, data lifecycle mgmt
- Data locality
  - Can only be achieved by caching: fill local disk on-the-fly, reuse data from it
  - Local NVMe disk (AWS i3 instance type):
    - 0.03msec latency, ~500MB/sec bandwidth, 500GB size (per core)

Natural Order Indexing

- 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
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

zone 0
zone 1
zone 2
zone 3

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

www.cwi.nl/~boncz/bigdatacourse

Q: key BETWEEN 13 AND 15?

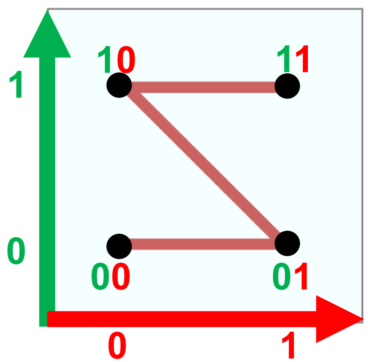
Multi-Dimensional Partitioning

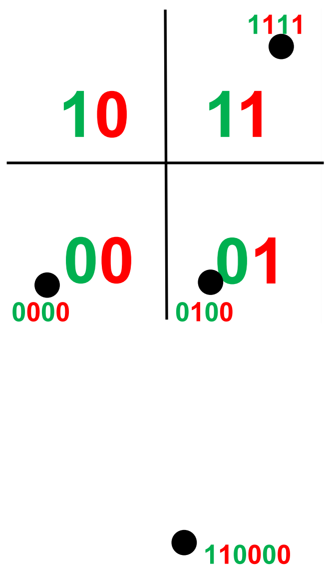
Partitioning is often done on the time dimension.

- what if you want to partition on multiple dimensions?

Use an ordering function that reduces **multiple** dimensions to a **single**

For instance **Z-order** mixes the dimension bits round robin (bitwise zero=0,one=1,two=10,three=11,four=100,five=101,six=110,seven=111, etc)





Example:
dataframe of 10.000 parquet files
queries filter on zipcode or time

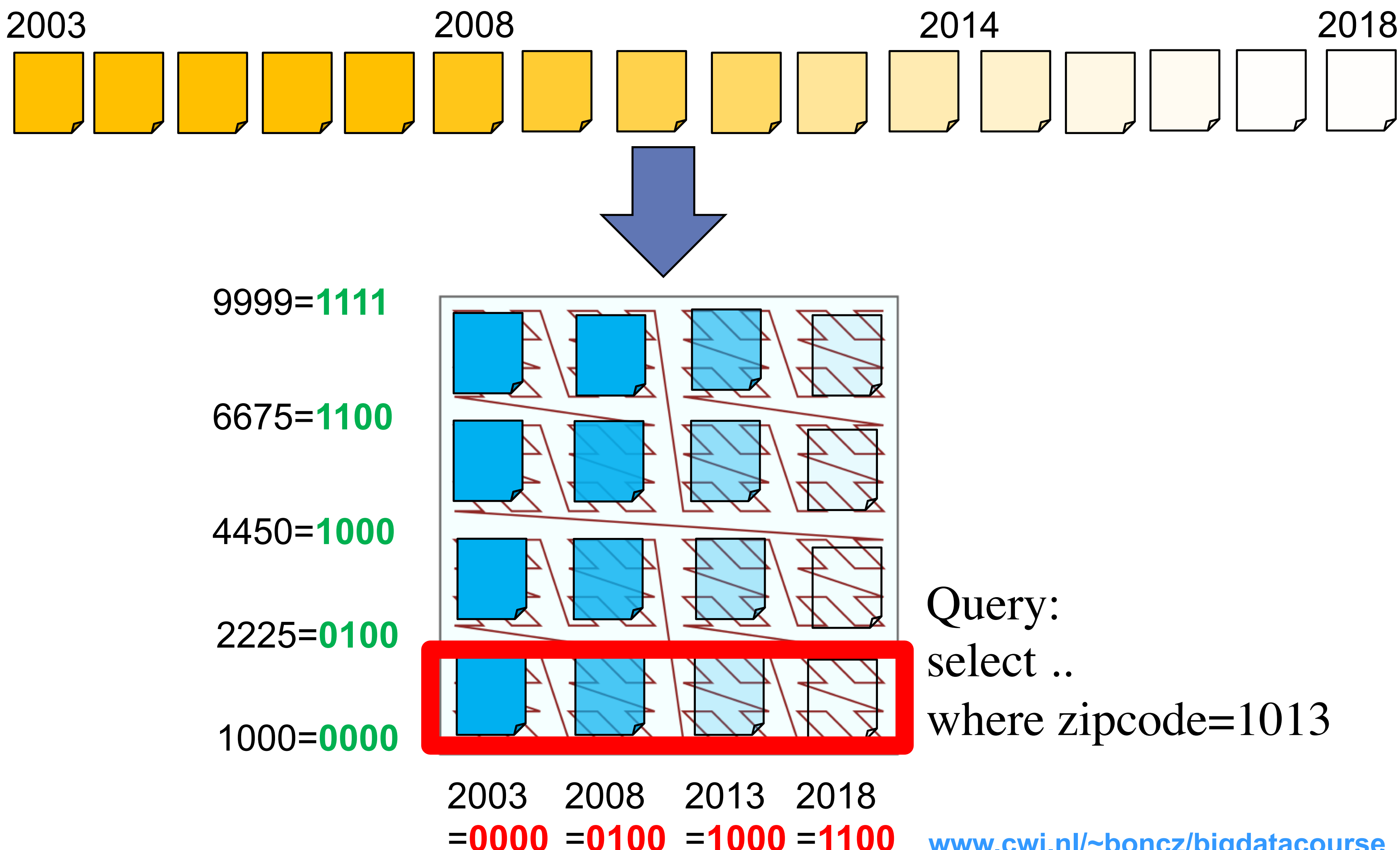
- create 128 time ranges (0-63 = 6bits)
- create 128 zipcode ranges (0-63 = 6bits)
- bitmix the range numbers (0-4095 = 12bits)
- re-partition the dataframe on this number

Desired result: 4096 new parquet files

Question: **will "partition pruning" based on MinMax become more effective?**

www.cwi.nl/~boncz/bigdatacourse

Example



Query:
select ..
where zipcode=1013

www.cwi.nl/~boncz/bigdatacourse

Big SQL System Architecture

storage

- columnar storage + compression
- table partitioning / distribution
- clustering and indexing

query-processor

- **vectorized** or JIT codegen
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

cluster

- (meta-) data sharing
- elastic resource provisioning
- continous update infrastructure

Centrum Wiskunde & Informatica

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

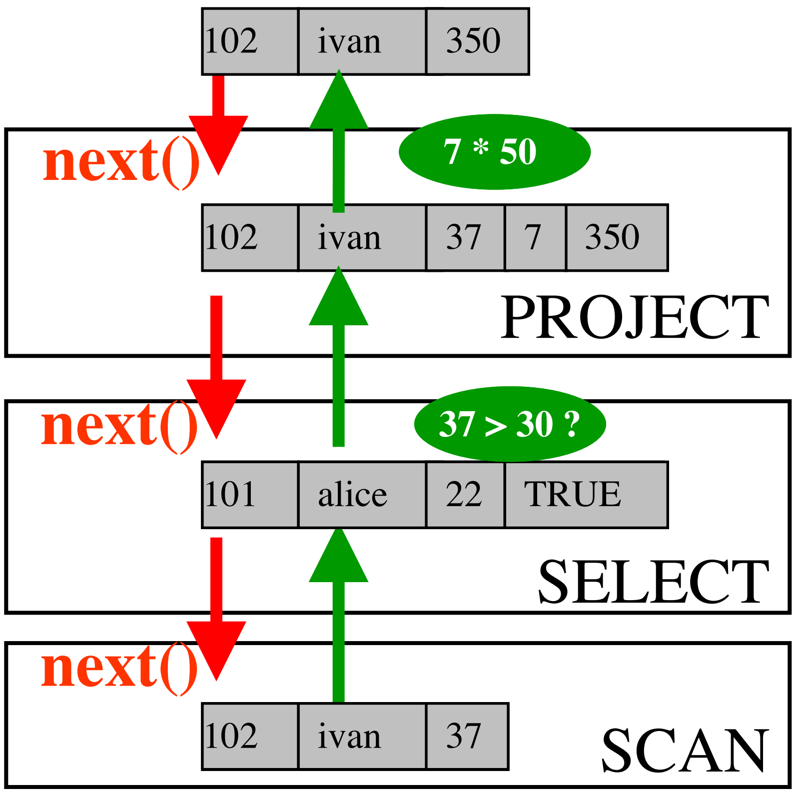
DBMS Computational Efficiency?

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

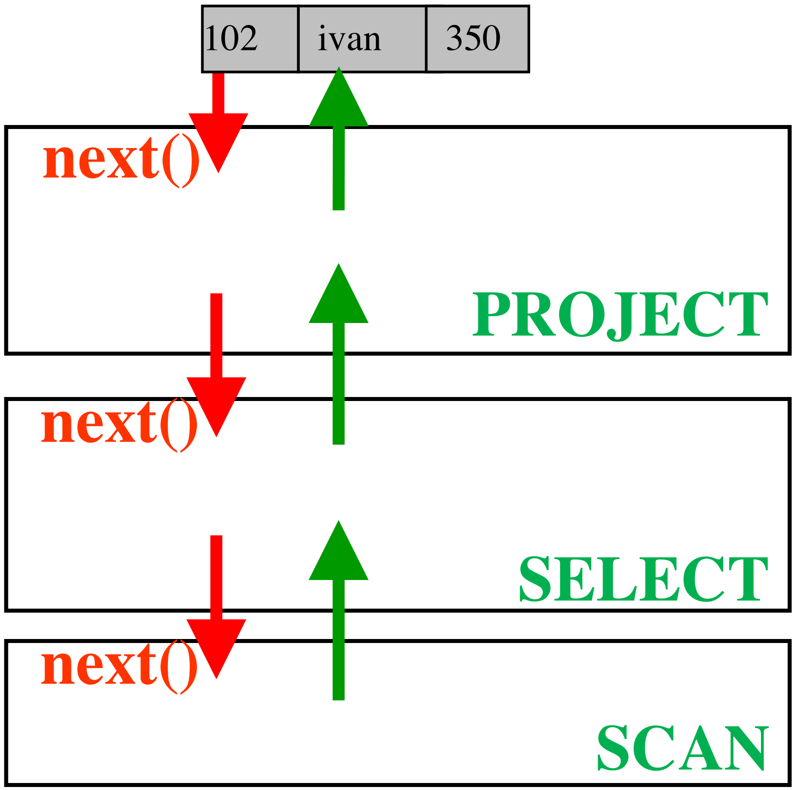
“MonetDB/X100: Hyper-Pipelining Query Execution ” Boncz, Zukowski, Nes, CIDR’05

How Do Query Engines Work?




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

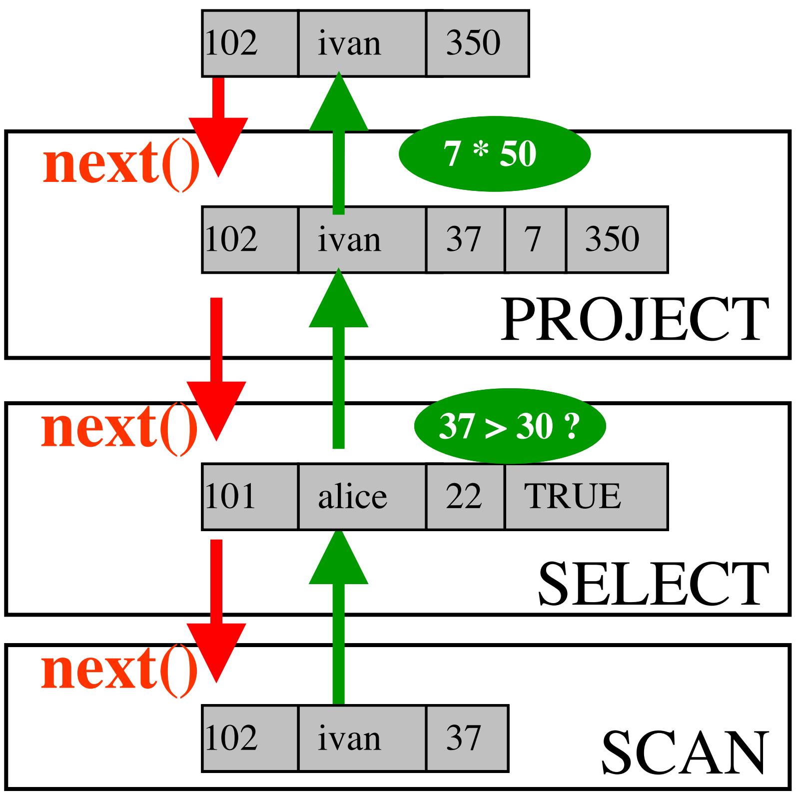
How Do Query Engines Work?



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

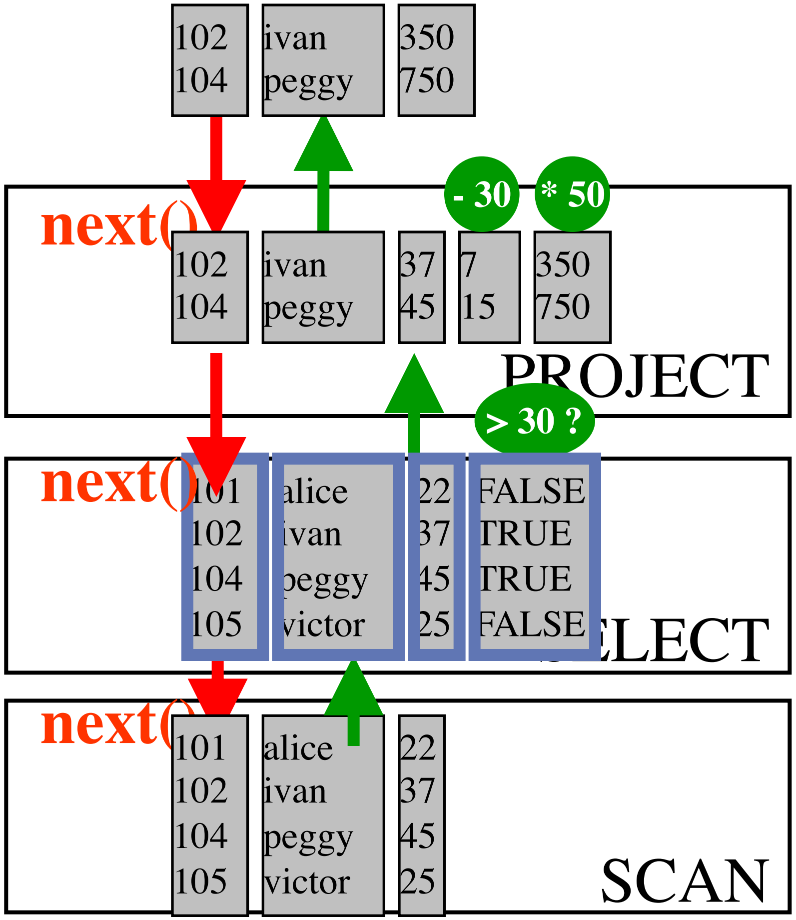
Observations:

"Vectorized In Cache Processing"

vector = array of ~100

processed in a tight loop

CPU cache Resident



CWI

Centrum Wiskunde & Informatica

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

vectorwise

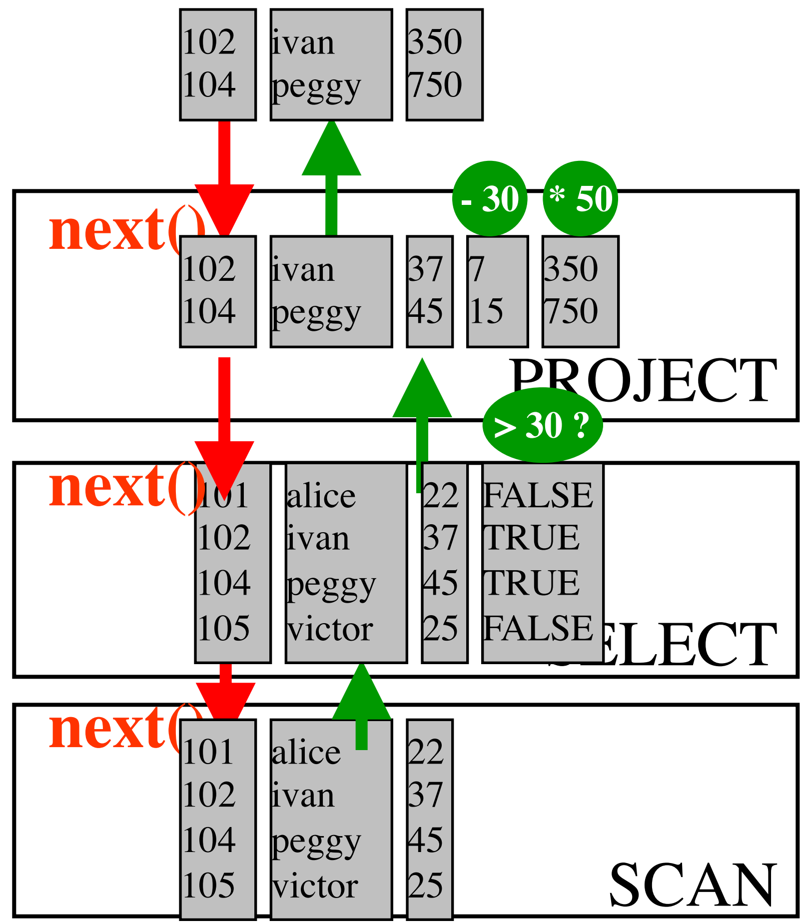
Observations:

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



www.cwi.nl/~boncz/bigdatacourse

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

vectorwise

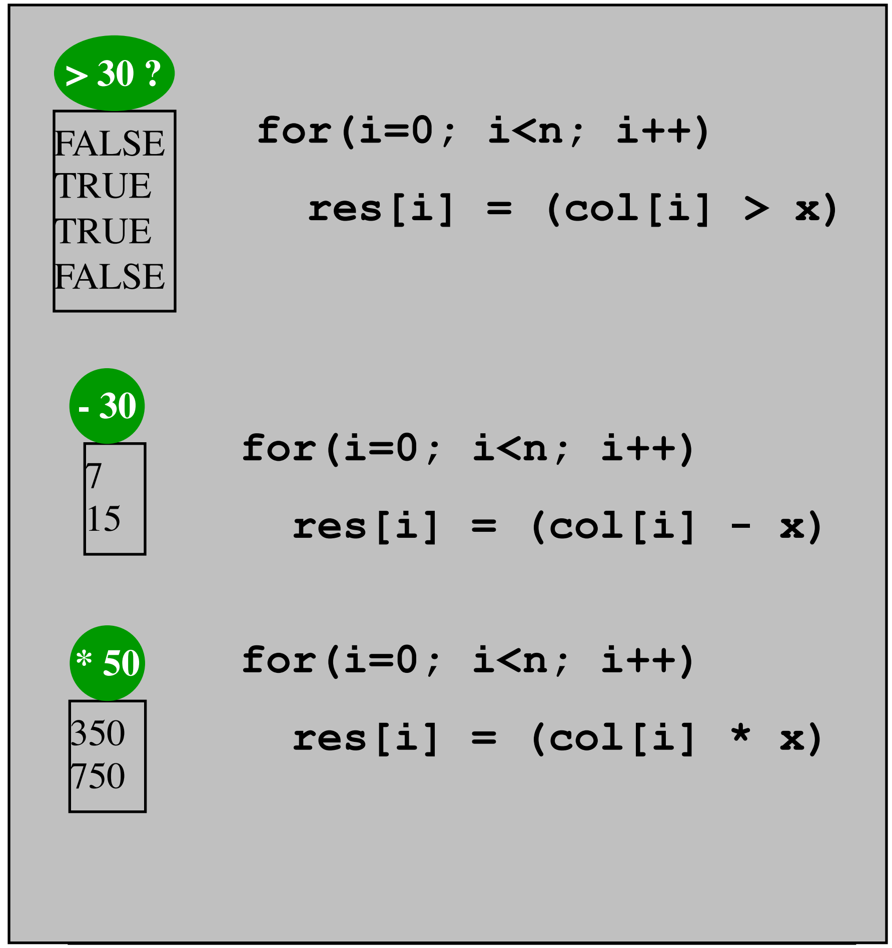
Observations:

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

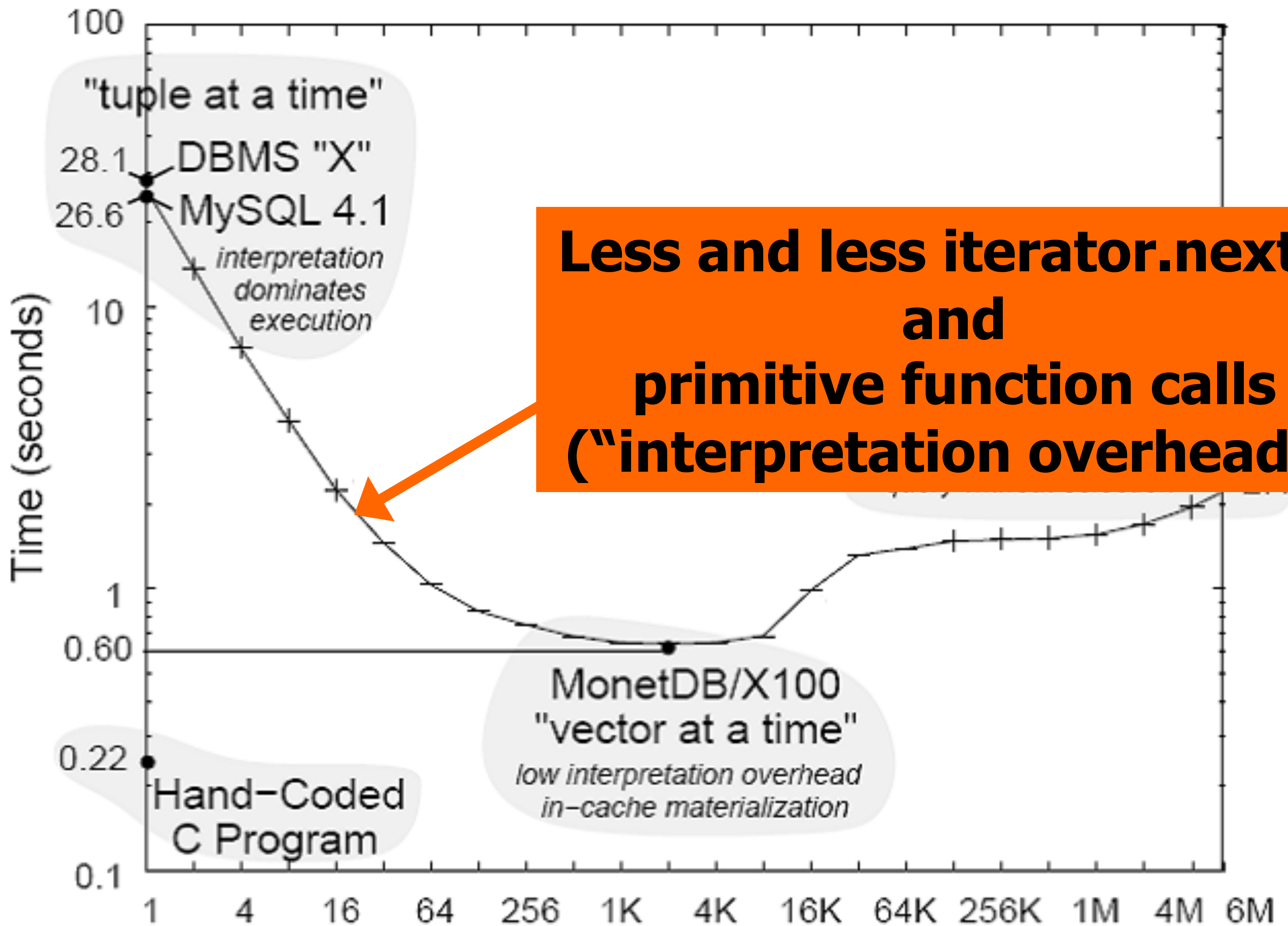
primitive calls process an

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



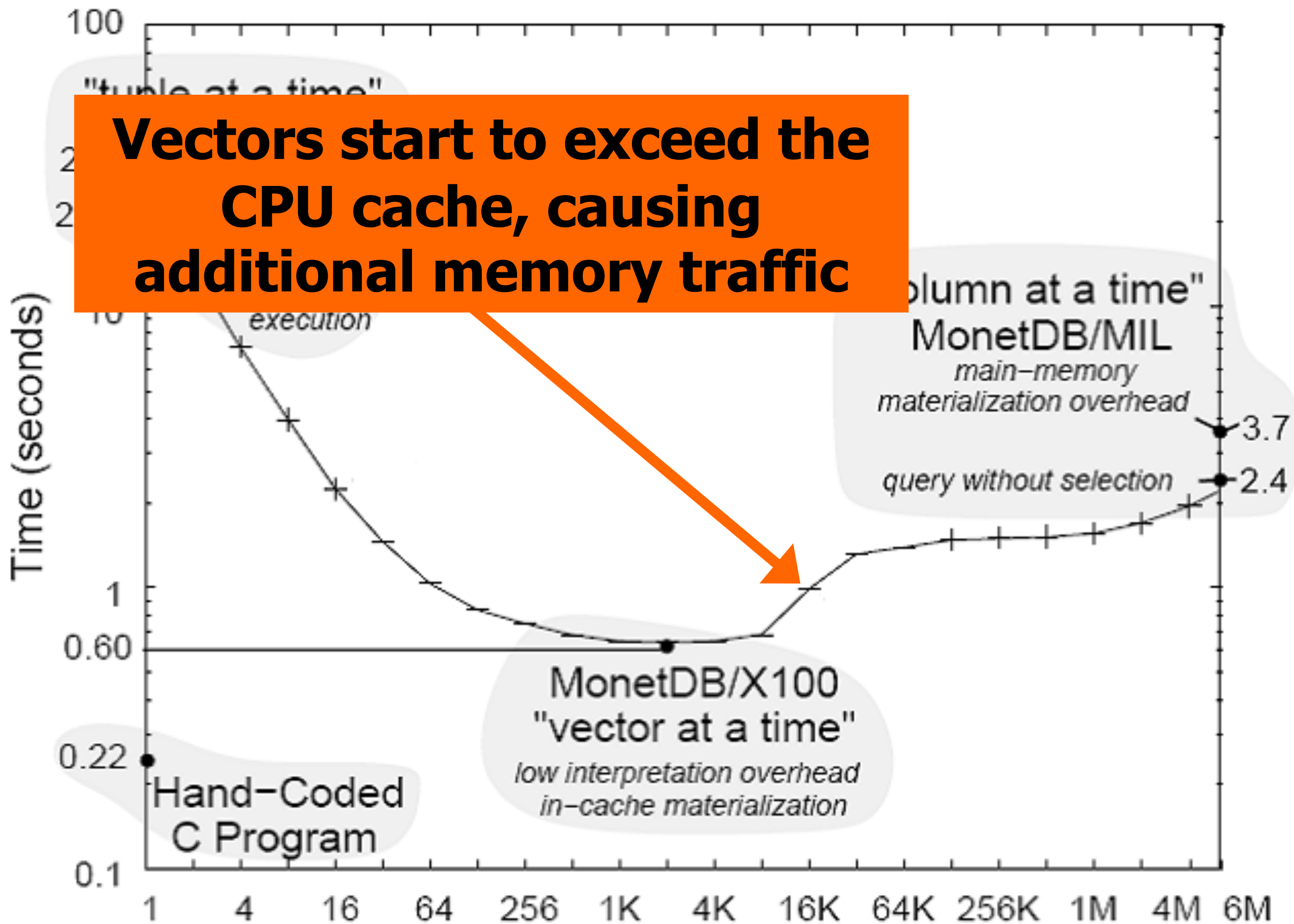
Varying the Vector size



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

Big SQL System Architecture

storage

– columnar storage + compression
– table partitioning / distribution
– clustering and indexing

query-processor

- vectorized or **JIT codegen**
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

cluster

- (meta-) data sharing
- elastic resource provisioning
- continous update infrastructure

Code-Generation based Query Execution

- SQL query gets parsed, normalized & optimized as in any other DB system
- Result is a physical query plan. Then:
  - Cut the plan in pipeline stages. Make a cut at each “blocking” operator
    - Blocking: op must see all data before producing output (eg SORT)
  - Translate each pipeline into a code snippet: single for-loop over the data.
  - Compile the code (“Just-In-Time compilation”) and run on your data

```
SELECT count(*) FROM store_sales WHERE ss_item_sk = 1000
```

becomes in Java:

```
long count = 0;
for(ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
     count += 1;
  }
}
```

Code-Generation based Query Execution

- Query gets parsed, normalized optimized as always
- Result is a physical query plan. Then:
  - Generate a separate program that executes (only) this exact query plan
  - Compile this program (Just-In-Time compilation) and run on your data

- The good:
  - No interpretation needed. You get a program that exactly runs the query, data layouts and types known. Logic hard-coded as tight loops over the data. Very fast.
    - Spark (“tungsten whole-stage codegen”): generates java code
    - Tableau/Hyper: assembly (“LLVM IR” intermediate representation)
- The bad:
  - JIT compilation takes time (query latency). Hard to debug. Hard to get per operator performance info (only per-stage). Cannot change the queryplan at runtime.

Big SQL System Architecture

storage

– columnar storage + compression
– table partitioning / distribution
– clustering and indexing

query-processor

- vectorized or JIT codegen
- fine- & coarse-grained parallelism
- rich SQL (+authorization+..)

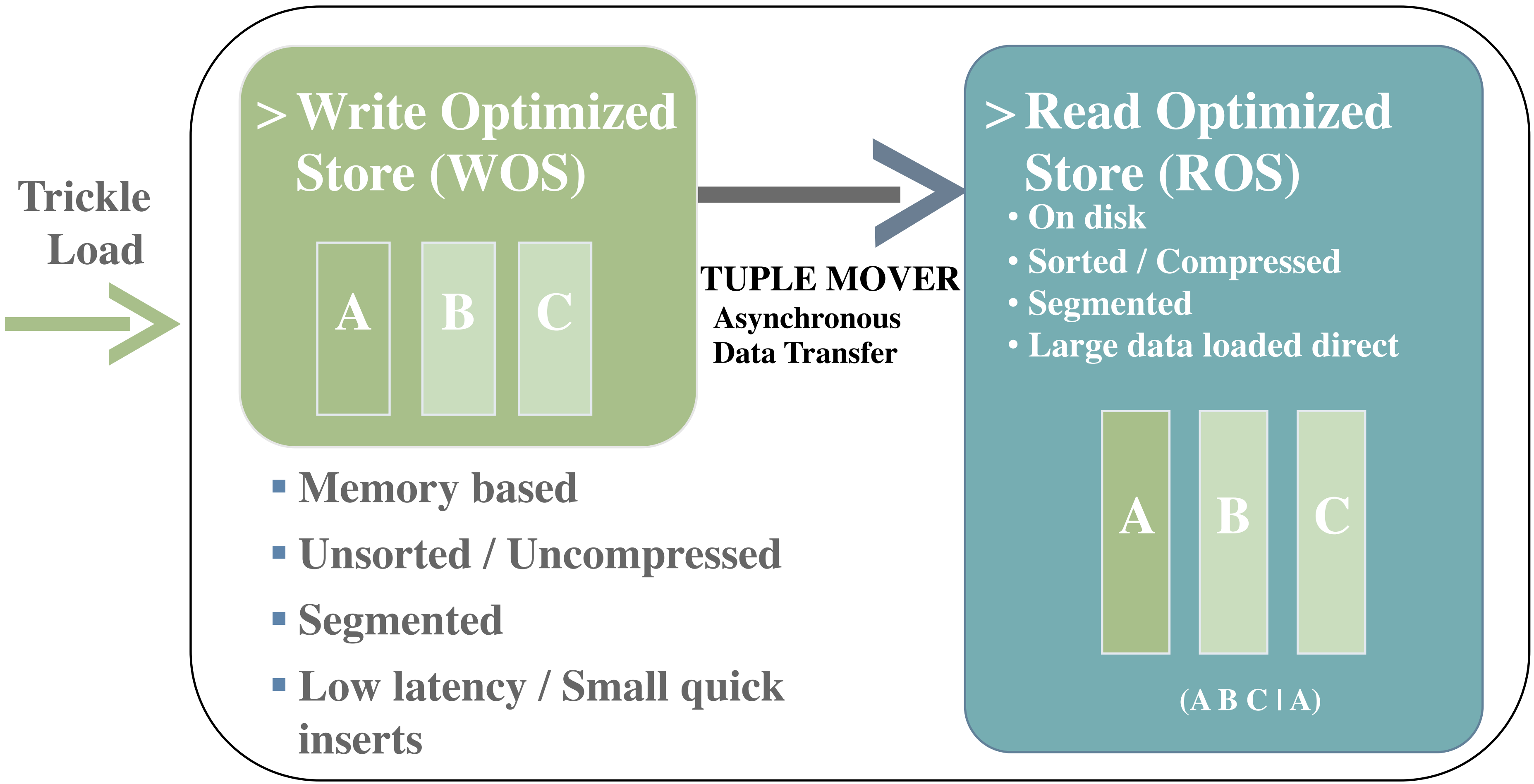
cluster

- (meta-) data sharing
- elastic resource provisioning
- **continous update infrastructure**

CWI
Centrum Wiskunde & Informatica

Batch Update Infrastructure (Vertica)

Challenge: hard to update columnar compressed data



www.cwi.nl/~boncz/bigdatacourse

CWI
Centrum Wiskunde & Informatica

Batch Update Infrastructure (Hive)

each update writes a separate HDFS file

Base File

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

Challenge: Merge During Query Processing

Update 1

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

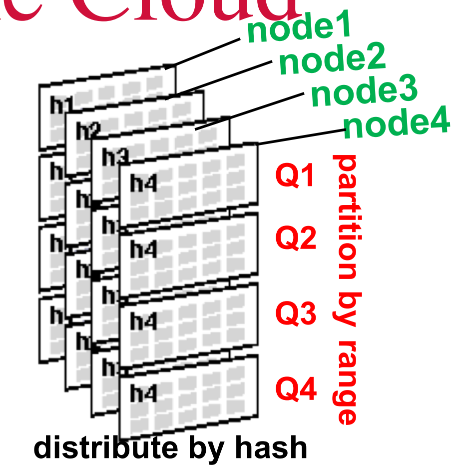
Update 2

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

Batch Update Infrastructure in the Cloud



- Cloud storage (S3, Azure Blob Storage) is **not updatable**
  - Each persistent update must write some new S3 file
- Challenge:
  - Data may arrive all the time, in small quantities
    - This leads to very many small files
    - For S3+Parquet, we need 100MB of data per file to be efficient!
- Solutions:
  1. Batch data in the update pipeline. Only go to the cloud when you have 100MB.
  2. A background compaction process:
     - Once in a while collapse many small S3 files into a big one
     - Either to a minimum threshold – or using exponentially bigger file targets
       - See: Log-Structured Merge-Trees (LSM trees)
     - Compaction is a good time to do partitioning/distribution

SQL ON BIG DATA
- IN THE CLOUD

CWI

Centrum Wiskunde & Informatica

Factors driving Data Systems growth

- Moore's Law
  - $100/TB storage, $1000 servers, commodity networking
- Increasing volumes of "dark" data
  - Data collected but never analyzed
- Widening analysis gap of "traditional" solutions
  - Due to their cost, complexity, scalability, & rigidity

Is it safe to have enterprise data in the Cloud?

2005: No way! Are you crazy?

2012: Don’t think so... But wait, we store our email where?

2018: Of course!

Getting a database in a cloud

Hi! I'm a Data Scientist!
I'm looking for a database for our cloud system

Hello! I am your account manager at X!

Sure thing! Let's install our product, DBMS X for you!

Awesome! It seems to work!

Great. Let me send you that invoice!

Just a sec… How much does the storage cost ?

Hold on, let me check that

And the system is elastic, right?

Wait, what?

And I only pay for what I use, right?

Mommy!!!

Traditional DB systems and the cloud

- Designed for:
  - Small, fixed, optimized clusters of machines
  - Constrained amount of data and resources

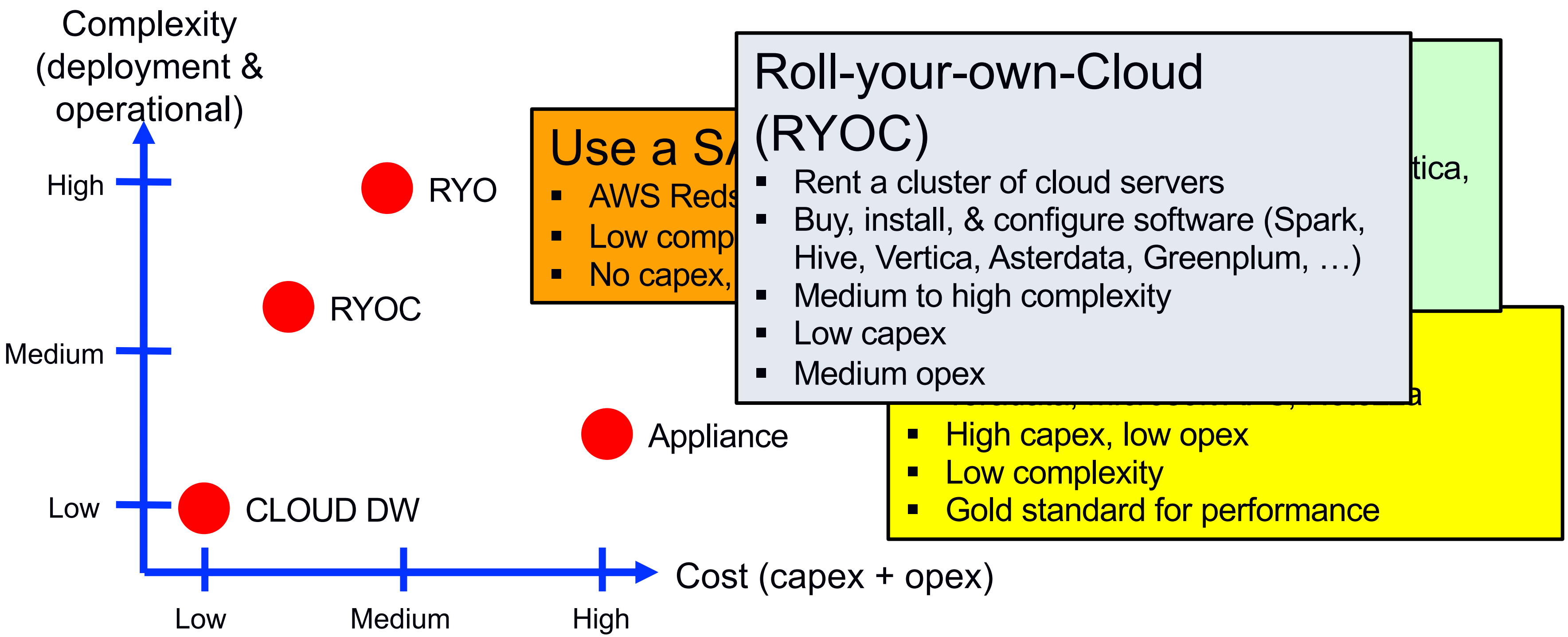
- Can be delivered via the Cloud
  - Reduce the complexity of hardware setup, software installation
  - No elasticity
  - No cheap storage
  - Not designed for cloud's poor stability
  - Not easy to use
  - Not "always on"
  - ...

Data in the Cloud

- Data traditional DW systems are built for
  - Assume predictable, slow-evolving internal data
  - Complex ETL (extract-transform-load) pipelines and physical tuning
  - Limited number of users and use-cases
  - OK to cost $100K per TB

- Data in the cloud
  - Dynamic, external sources: web, logs, mobile devices, sensor data…
  - ELT instead of ETL (data transformation inside the system)
  - Often in semi-structured form (JSON, XML, Avro)
  - Access required by many users, very different use cases
  - 100TBs volume common

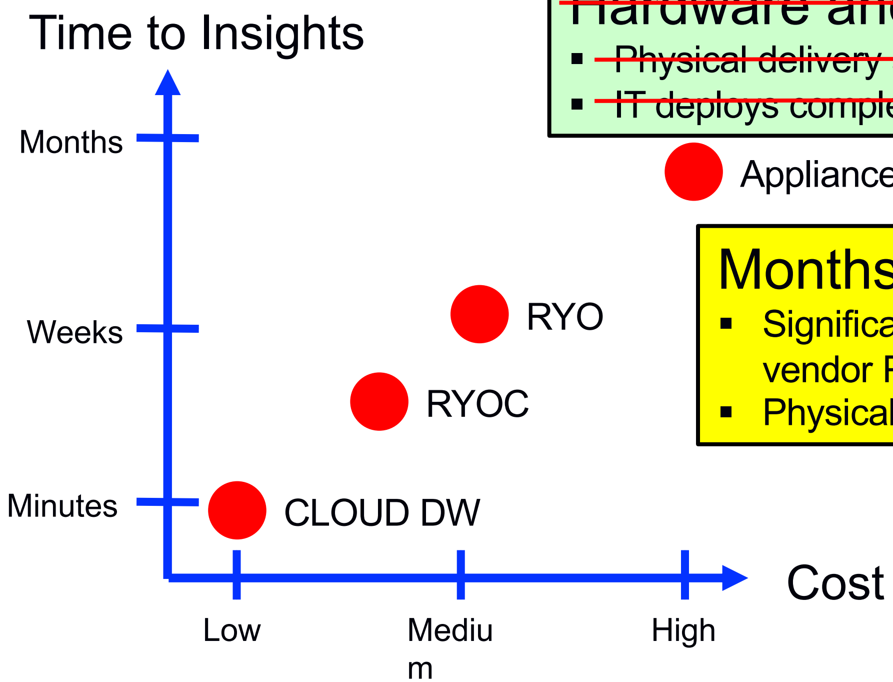
10,000 ft. view: Complexity vs Cost



Instant gratification

~~Hardware and Deployment~~

- ~~Physical delivery and setup of cluster~~
- ~~IT deploys complex parallel system~~

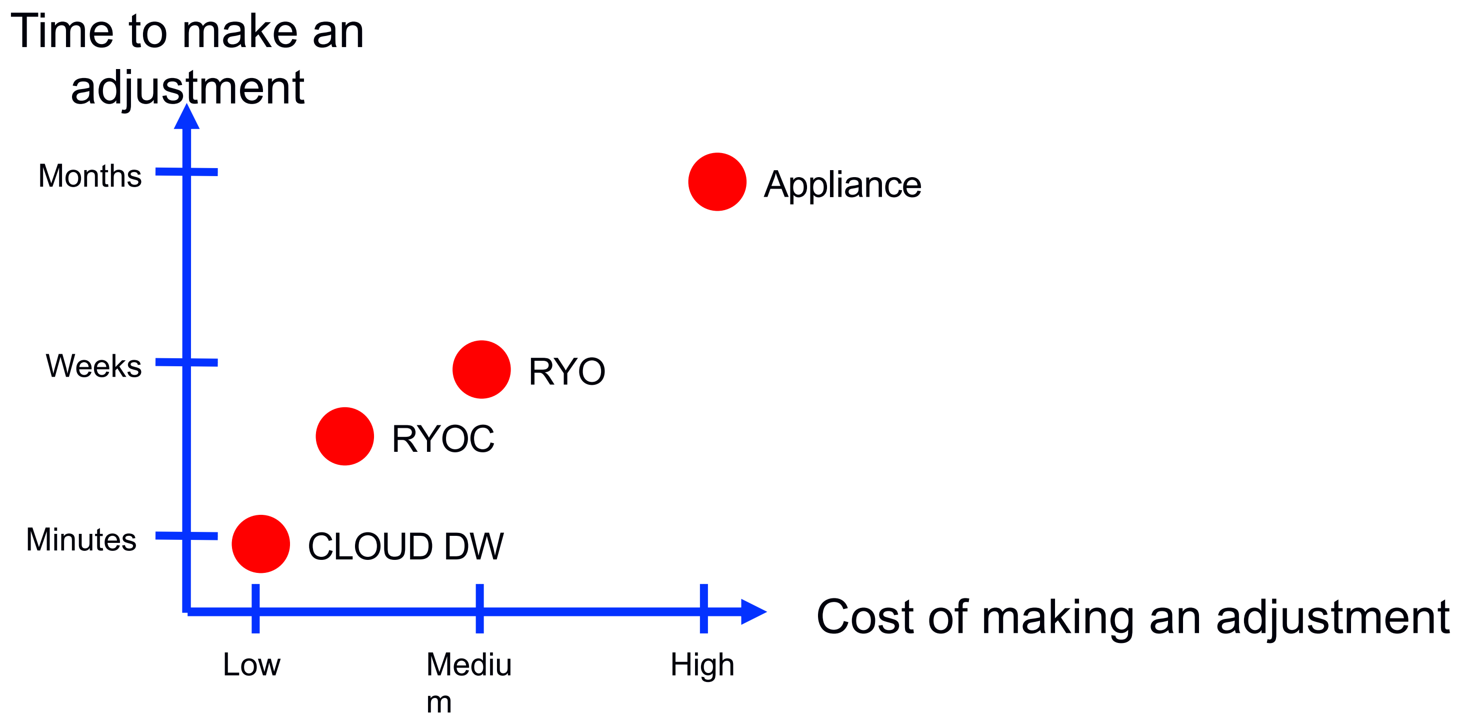


Months due to high capex cost

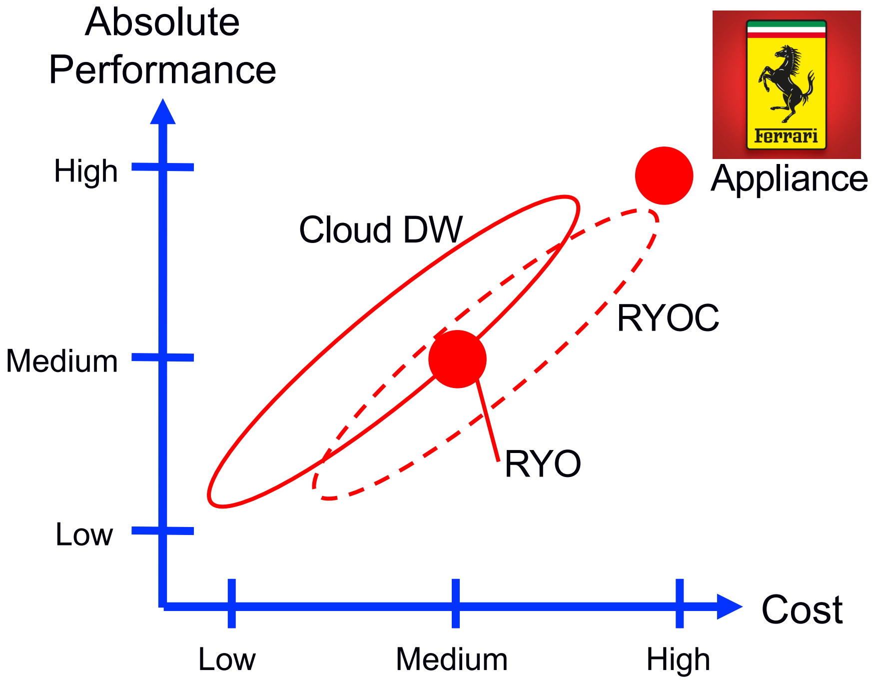
- Significant due diligence through multiple vendor POCs
- Physical delivery and setup of goods

Centrum Wiskunde & Informatica

Scalability and the price of agility



Unfortunately, no “free lunch”



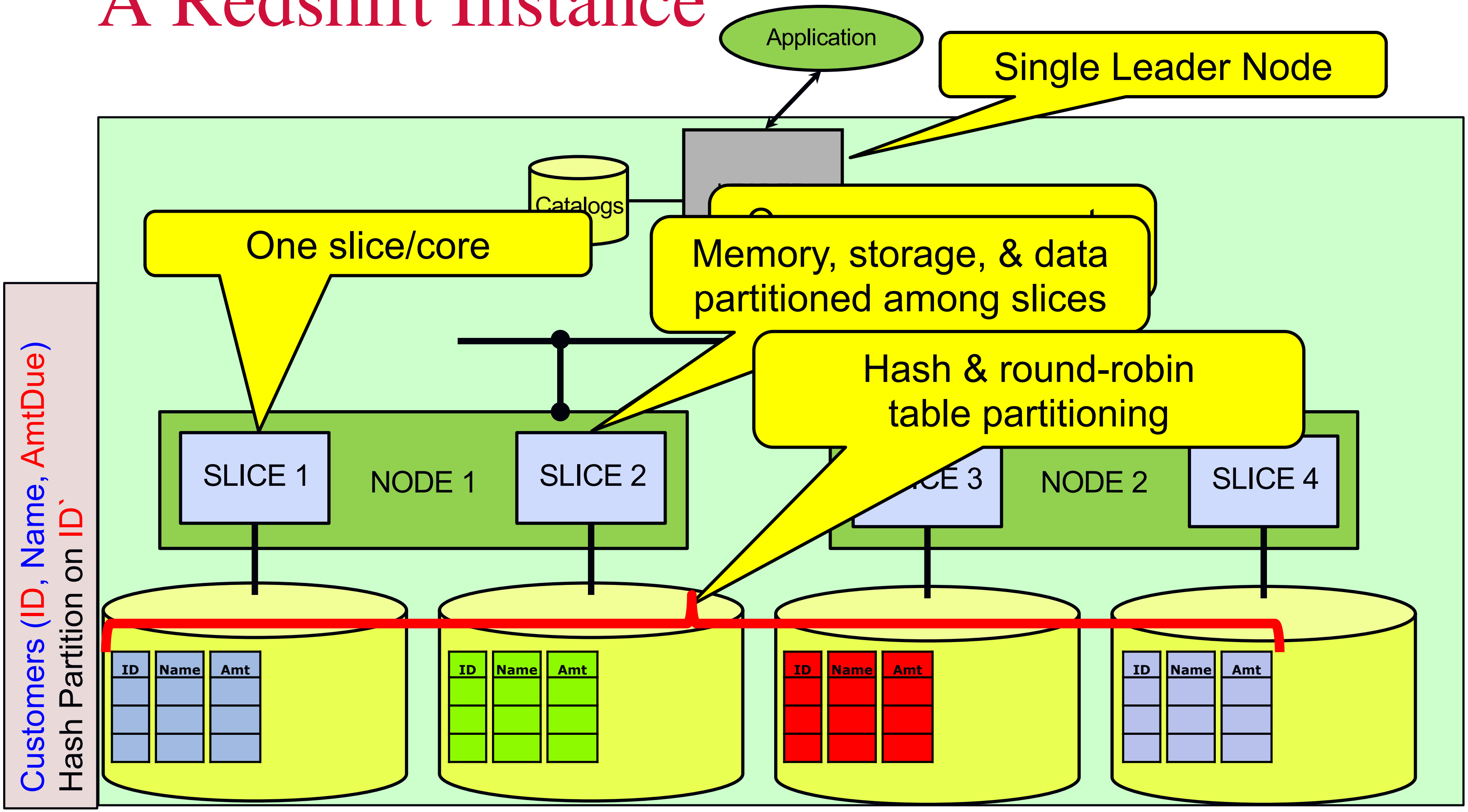
Why Cloud DW?

- No CapEx and low OpEx
- Go from conception to insight in hours
- Rock bottom storage prices (Azure, AWS S3, GFS)
- Flexibility to scale up/down compute capacity
- Simple upgrade process

Amazon (AWS) Redshift

- Classic shared-nothing design w. locally attached storage
  - Engine is ParAccel database system (classic MPP, JIT C++)
- Leverages AWS services
  - EC2 compute instances
  - S3 storage system
  - Virtual Private Cloud (VPC)
- Leader in market adoption

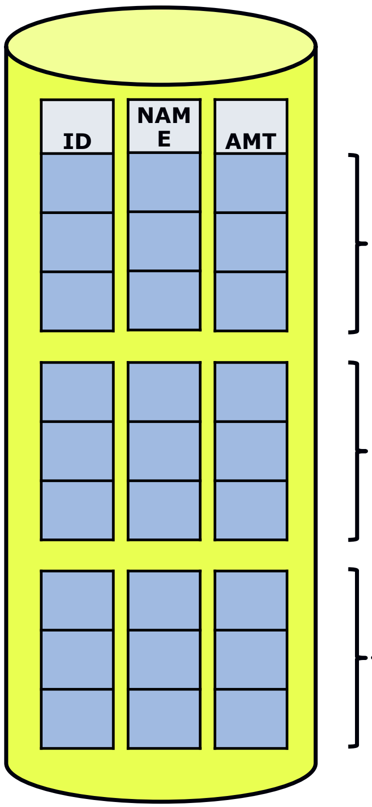
A Redshift Instance



Within a slice

Two sort options:

1) Compound sort key
2) "Interleaved" sort key (multidimensional sorting)

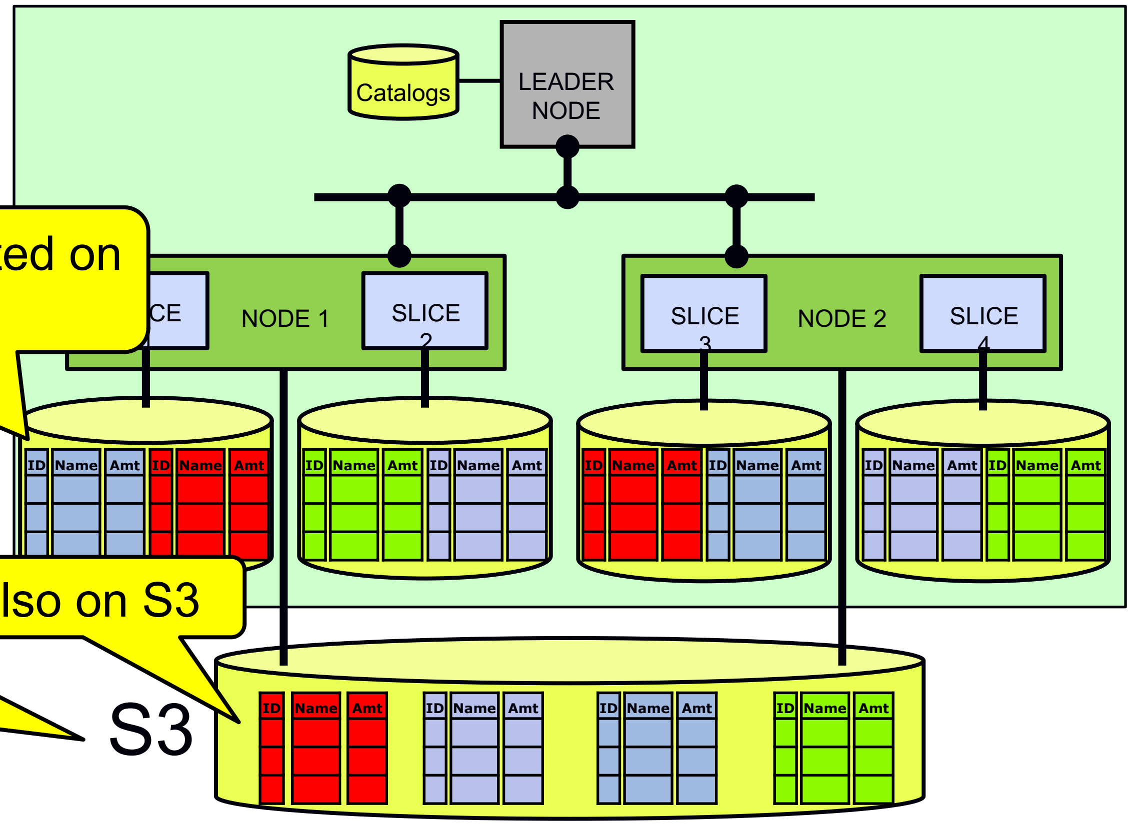


Columns stored in 1MB blocks

Min and Max value of each block retained in a "zone" map

Rich collection of compression options (RLE, dictionary, gzip, …)

Unique Fault Tolerance Approach

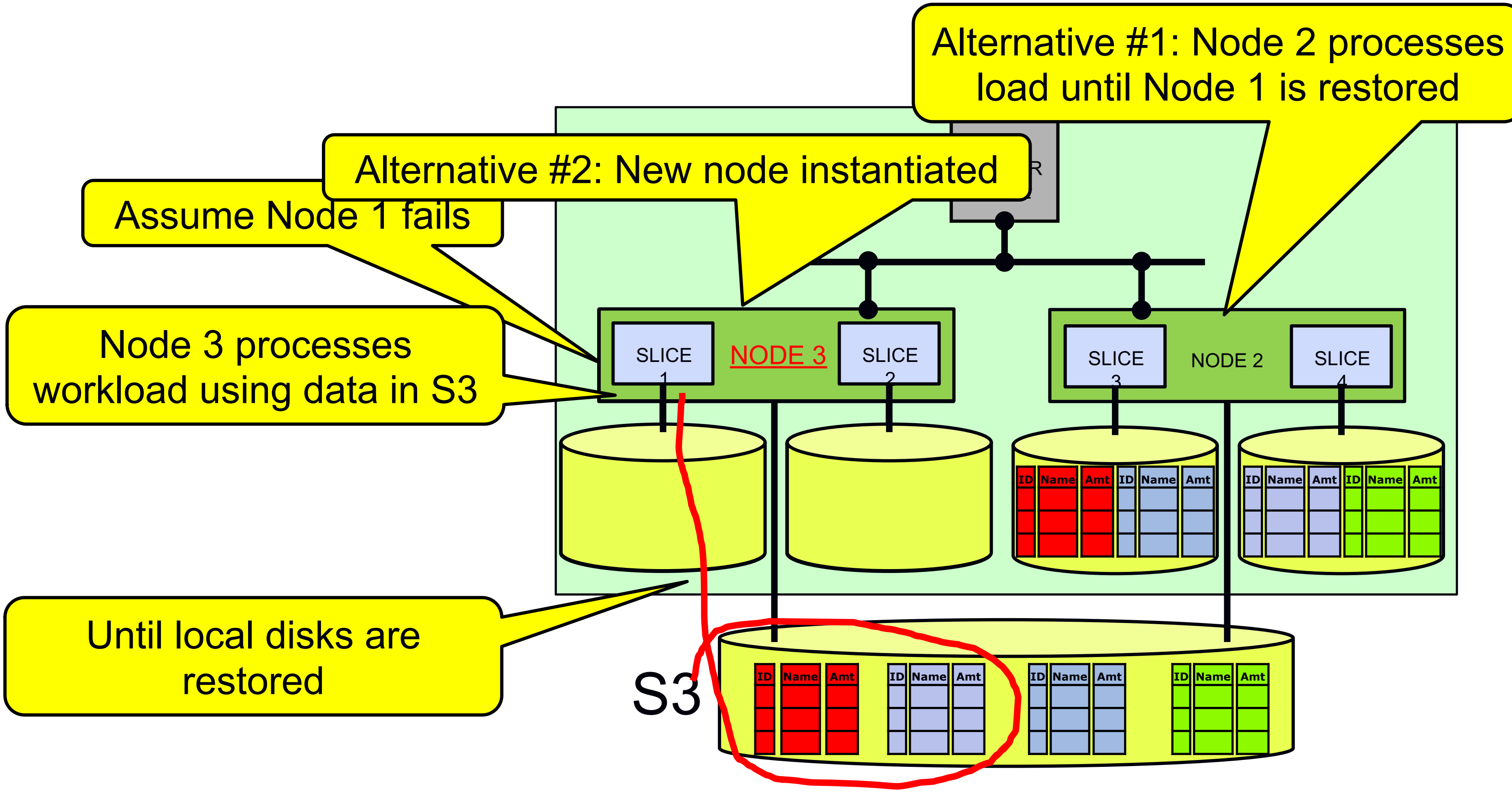


Each 1MB block gets replicated on a different compute node

And also on S3

S3, in turn, triply replicates each block

Handling Node Failures



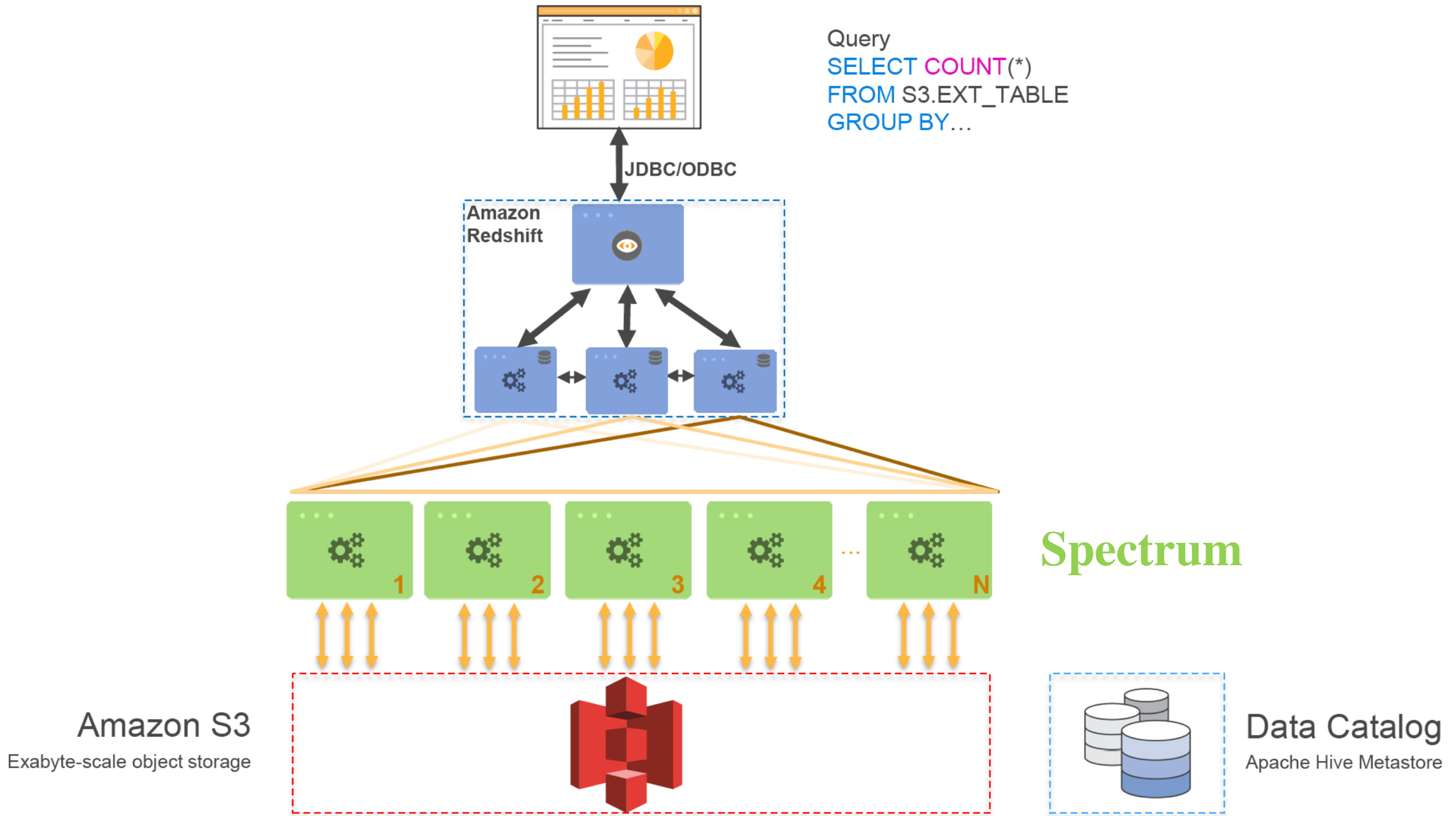
Redshift Summary

- Highly successful cloud SAAS DW service
- Classic shared-nothing design
- Leverages S3 to handle node and disk failures
- Key strength: performance through use of local storage
- Key weaknesses: compute cannot be scaled independent of storage (and vice versa)

Redshift Spectrum

- **Serverless** extension to Redshift
  - Spectrum automagically runs on many nodes you do not need to start or stop. Pay per query
- Can access large datasets in S3
  - Parquet, ORC, CSV, json, …
- Streams query sub-results into a Redshift cluster
  - Redshift cluster handles the rest of the query
  - Spectrum can filter and pre-aggregate massive data
  - Spectrum-Redshift highly compatible

Redshift Spectrum



www.cwi.nl/~boncz/bigdatacourse

Snowflake Elastic DW

- Shared-storage design
  - Compute decoupled from storage
  - Highly elastic
- Leverages AWS
  - Tables stored in S3 but dynamically cached on local storage Clusters of EC2 instances used to execute queries
- Rich data model
  - Schema-less ingestion of JSON documents

Snowflake Architecture

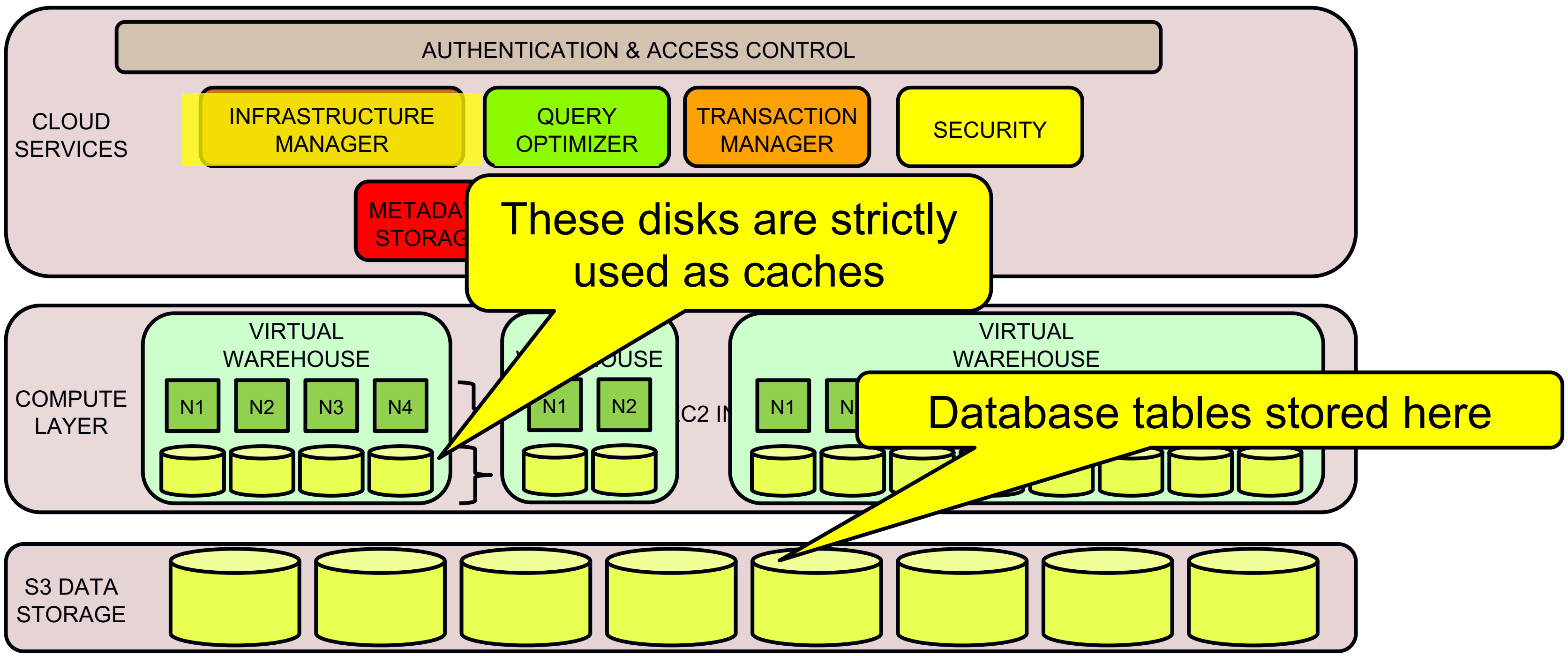
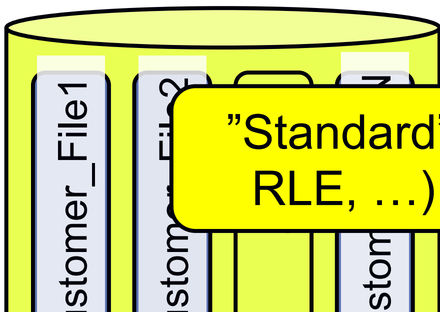


Table Storage

- Rows of each table are stor
  S3 files:

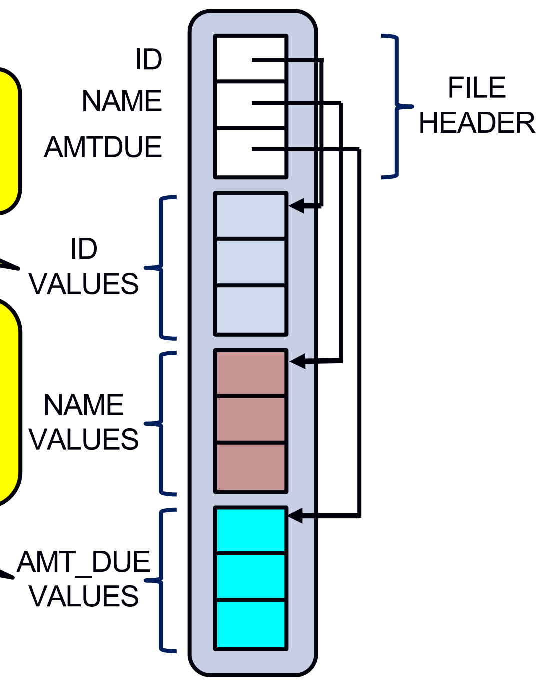
Not able to support hash or RR partitioning as files are created strictly as rows are inserted into table



"Standard" compression (gzip, RLE, …) schemes available

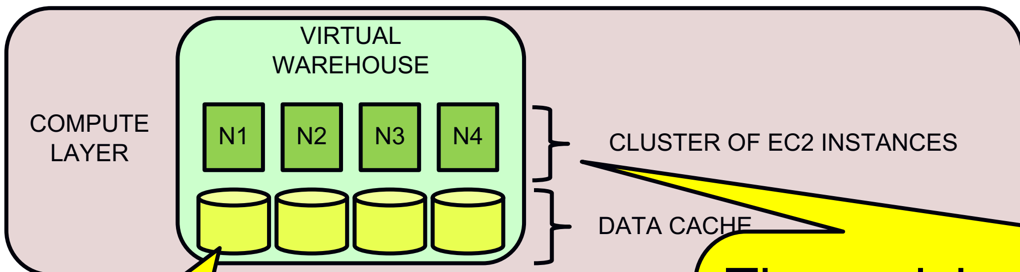
- Each … ows stored in columnar fashion

Min & max value of <u>each</u> column of <u>each</u> file of <u>each</u> table are kept in catalog. Used for pruning at run time.



Virtual Warehouses

Dynamically created cluster of EC2 instances

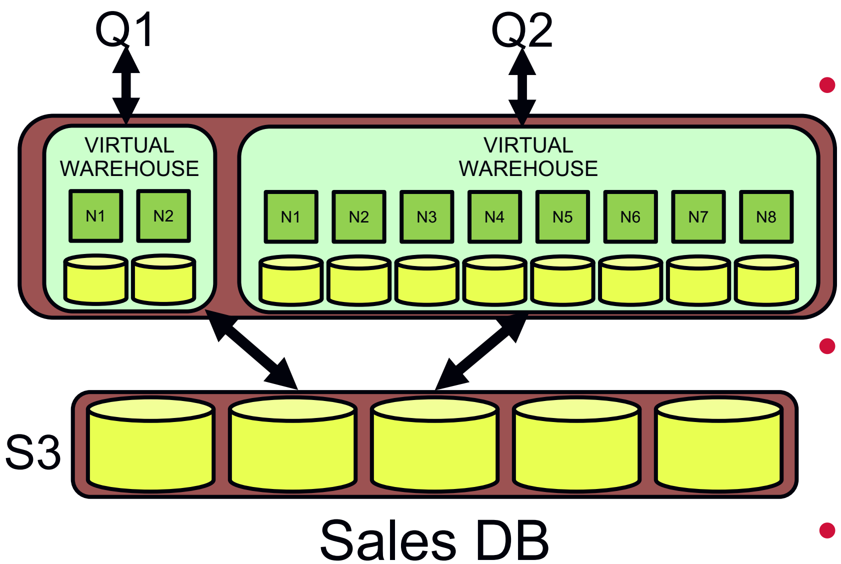


Three sizing mechanisms:

1) Number of EC2 instances
2) "Size" of each instance (# cores, I/O capacity)
3) Auto-scaling of one virtual warehouse

Local disks cache file headers & table columns

Separate Compute & Storage.



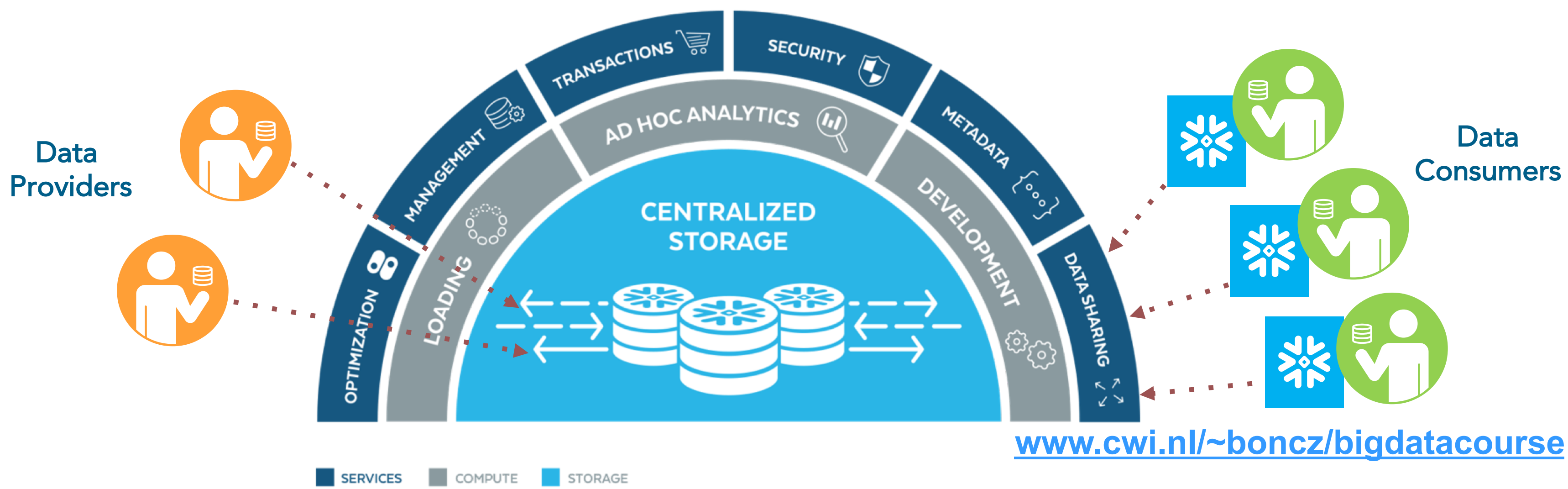
- Queries against the same DB can be given the resources to meet their needs – truly unique idea
- DBA can dynamically adjust number & types of nodes
- This flexibility is simply not feasible with a shared-nothing approach such as RedShift.

Data sharing

- Enabled by Snowflake's unique cloud architecture

Providers
- Secure and integrated Snowflake's access control model
- Only pay normal storage costs for shared data
- No limit to the number of consumer accounts with which a dataset may be shared

Consumers
- Get access to the data without any need to move or transform it.
- Query and combine shared data with existing data or join together data from multiple publishers



www.cwi.nl/~boncz/bigdatacourse

Snowflake Summary

- Designed for the cloud from conception
- Can directly query unstructured data (Json) w/o loading
- Compute and storage independently scalable
  - AWS S3 for table storage, uses its own closed formal (you need to load)
  - Virtual warehouses composed of clusters of AWS EC2 instances
  - Not ”serverless”
  - Queries can be given exactly the compute resources they need
- No management knobs
  - No indices, no create/update stats, no distribution keys, …

Google BigQuery

- Separate storage and compute
- Leverages Google's internal storage & execution stacks
  - Collosus distributed file system
  - DremelX query executor
  - Jupiter networking stack
  - Borg resource allocator
- No knobs, no indices, …

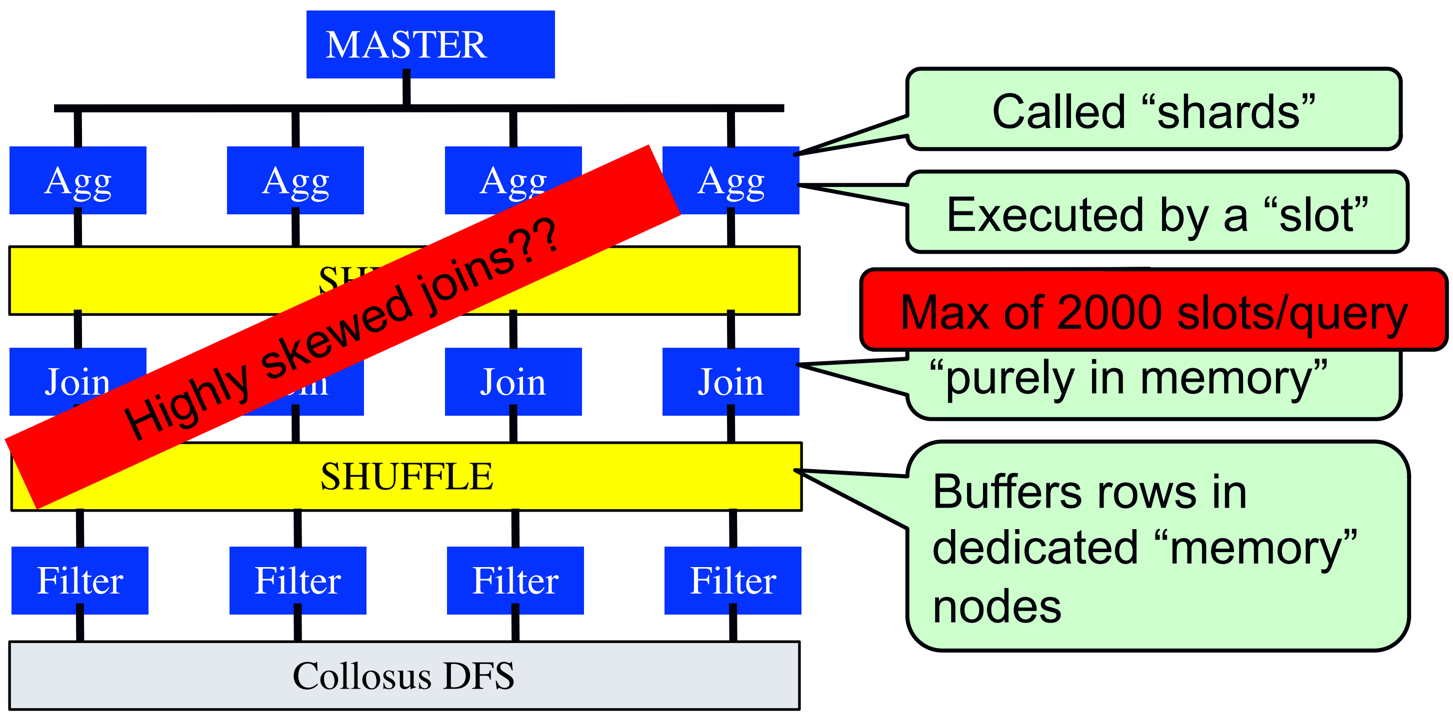
BigQuery Tables

- Stored in Collosus FS
  - Partitioned by day (optionally)
- Columnar storage (Capacitor)
  - RLE compression
  - Sampling used to pick sort order
  - Columns partitioned across multiple disks
- Also "external" tables
  - JSON, CSV & Avro formats
  - Google Drive and Cloud Storage

CWI
Centrum Wiskunde & Informatica

Query Execution

SQL queries compiled into a tree of DremelX operators



CPU Resource Allocation

- Serverless, which usually implies..
- compute resources not dedicated!
  - Shared among other internal and external customers
  - No apparent way to control computational resources used for a query
- # of shards/slots assigned to an operator function of:
  - Estimated amount of data to be processed
  - Cluster capacity and current load

BigQuery Pricing

- Storage: $0.02/GB/month

  (AWS is about $0.023/GB/month)

- Query options

  1) Pay-as-you-go: $5/TB "processed"

     - calculated after column is uncompressed

  (AWS is about $1.60/TB using M4.4Xlarge EC2 instance)

  2) Flat rate: $40,000/month for 2,000 dedicated slots

Amazon Athena

- Similar to Google BigQuery:
  - serverless analytical SQL
- Works straight on S3
  - Parquet, ORC, CSV, JSON
  - Pay by the data accessed (only)
- Presto in-the-cloud
  - plus Hive for table creation
  - plus “Glue” for bulk loading

Databricks Spark

- Spark-as-a-service in the cloud (“the best Spark”)
  - All data stored in S3
- Clusters run in the **user** account
  - Control plane runs in Databricks account
- User can dynamically power up and down clusters
  - Clusters can be grown and shrunk

DBIO Caching Layer

- cloud instances have fast local disks
  - AWS: NVMe 3TB drives, 500MB/s per core (125MB/s S3)
  - Azure: even bigger difference (slower network)
- DBIO caches Parquet pages
  - compressed or uncompressed
  - **Spark scheduler** schedules jobs with affinity (node that likely caches data becomes executor of queries on ot)

Big Data was the Missing Link for AI

BIG DATA

Customer Data

Emails/Web pages

Click Streams

Sensor data (IoT)

Video/Speech

GREAT RESULTS

Big Data was the Missing Link for AI

BIG DATA

Customer Data

Emails/Web pages

Click Streams

Sensor data (IoT)

Video/Speech

GREAT RESULTS

Hardest part of AI isn't AI

"Hidden Technical Debt in Machine Learning Systems", Google NIPS 2015

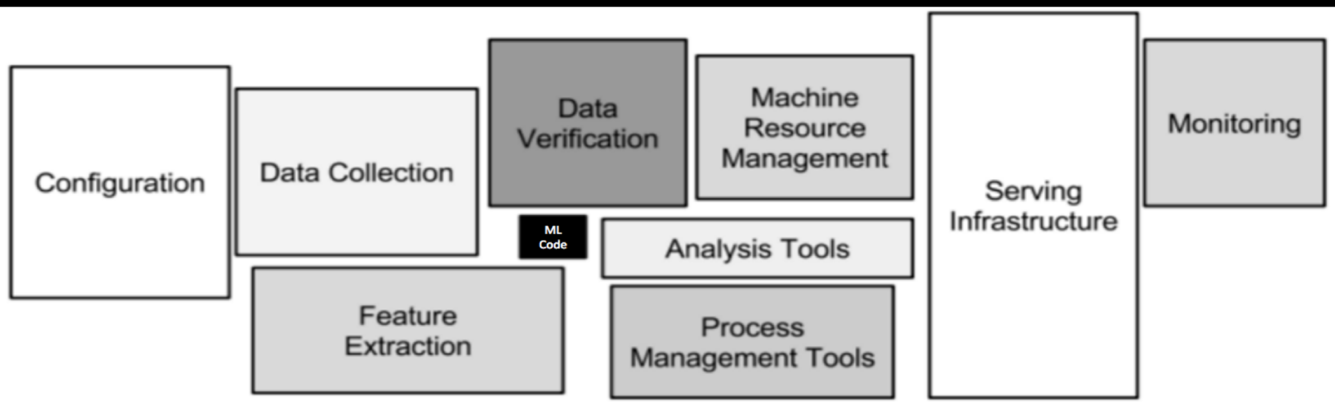


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Databricks Delta

THE GOOD
OF DATA LAKES

- Massive scale on Amazon S3
- Open Formats (Parquet, ORC)
- Predictions (ML) & Real Time Streaming

THE GOOD
OF DATA WAREHOUSES

- Pristine Data
- Transactional Reliability
- Fast Queries (10-100x)

Databricks Delta

THE GOOD OF DATA LAKES

- Massive scale on Amazon S3
- Open Formats (Parquet, ORC)
- Predictions (ML) & Real Time Streaming

THE GOOD OF DATA WAREHOUSES

- Pristine Data
- Transactional Reliability
- Fast Queries (10-100x)

MASSIVE SCALE

- Decouple Compute & Storage

RELIABILITY

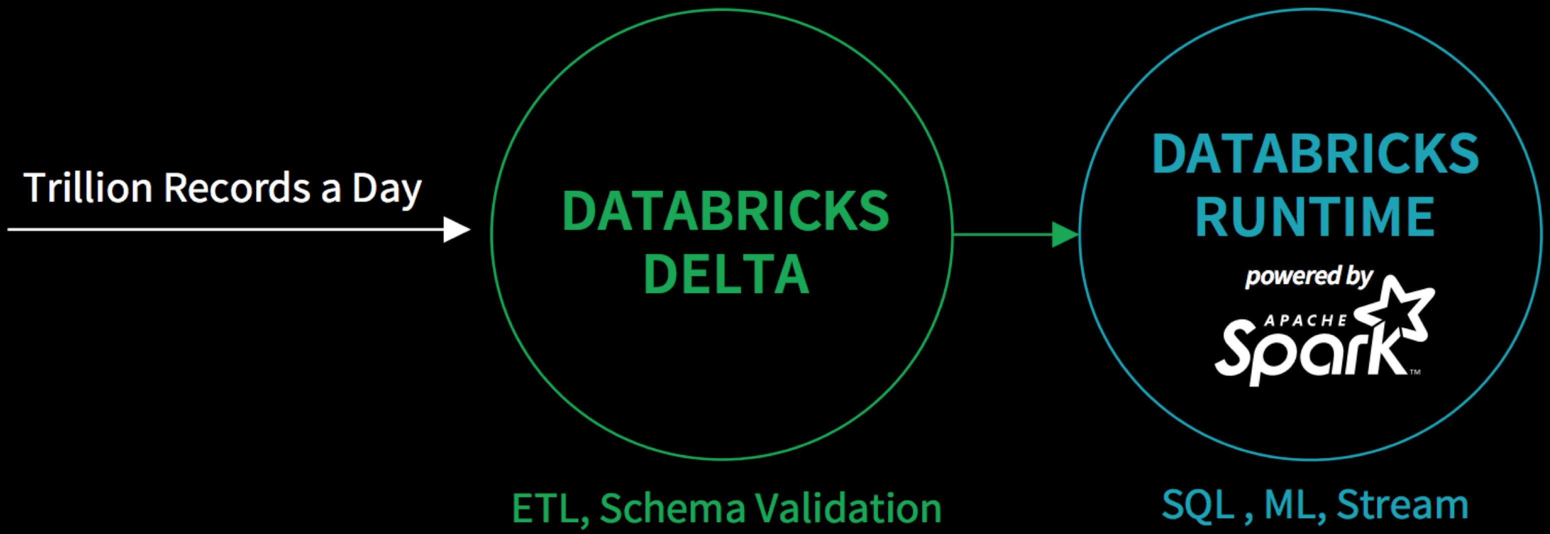
- ACID Transactions & Data Validation

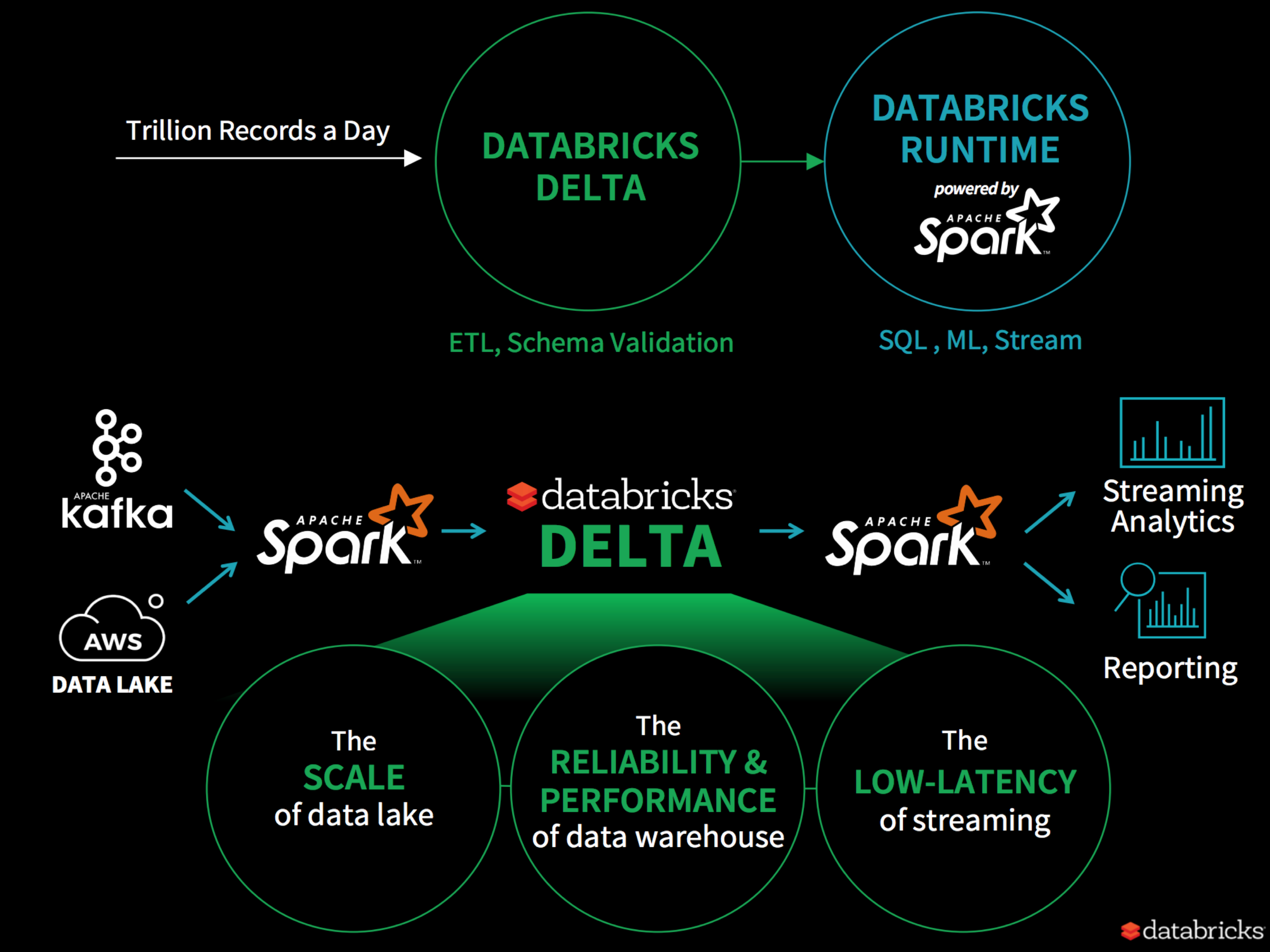
PERFORMANCE

- Data Indexing & Caching (10-100x)

LOW-LATENCY

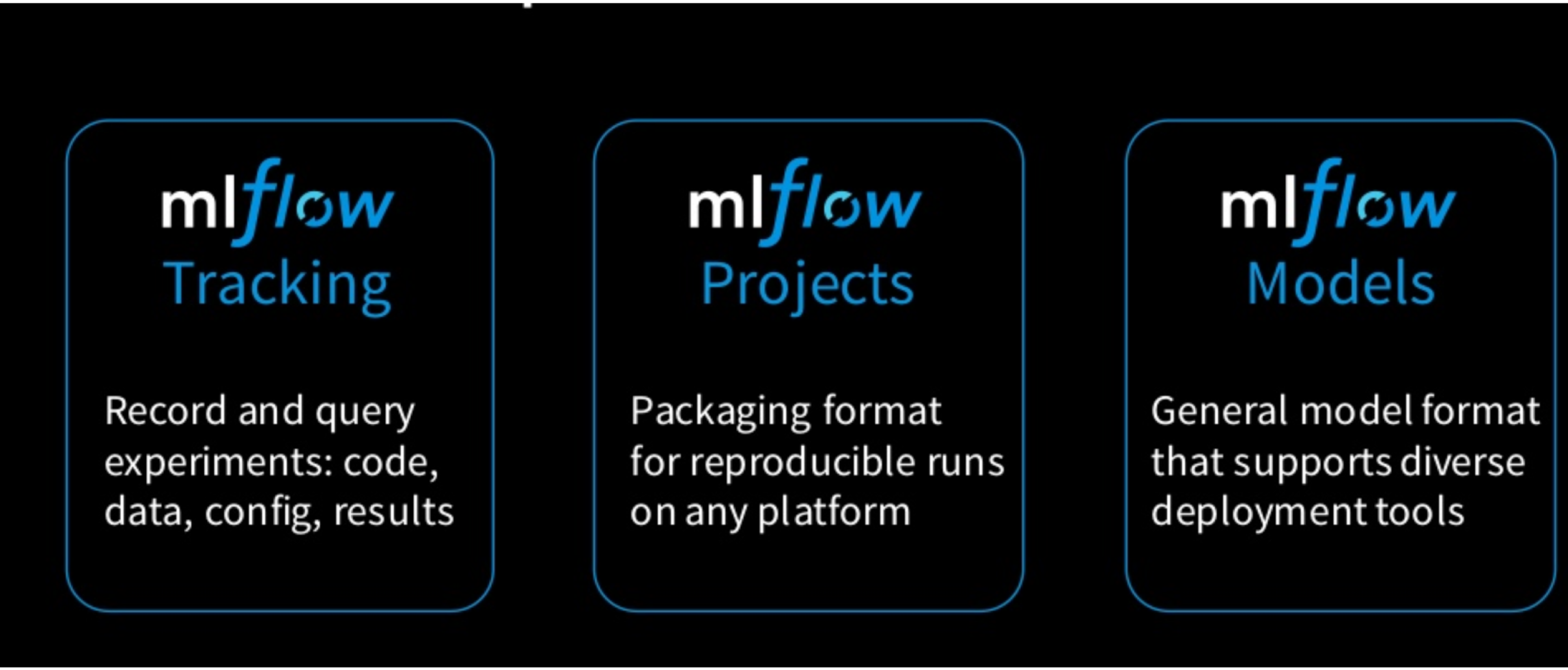
- Real-Time Streaming Ingest


Trillion Records a Day
DATABRICKS DELTA
DATABRICKS RUNTIME
powered by
APACHE Spark
ETL, Schema Validation
SQL , ML, Stream


Trillion Records a Day
DATABRICKS DELTA
DATABRICKS RUNTIME
powered by APACHE Spark
ETL, Schema Validation
SQL , ML, Stream
APACHE kafka
AWS
DATA LAKE
APACHE Spark
databricks
DELTA
APACHE Spark
Streaming Analytics
Reporting
The SCALE of data lake
The RELIABILITY & PERFORMANCE of data warehouse
The LOW-LATENCY of streaming
databricks

Databricks MLflow

- System to make ML experiments reproducible



Pay For What You Use

- Amazon Redshift
  - More storage requires buying more compute. Rather expensive.
- Snowflake
  - Charged separately for S3 storage and EC2 usage
  - Data resides in Snowflake account
  - works in AWS, Azure, and soon Google cloud
- BigQuery serverless
  - Charged separately for GFS storage and TBs "processed"
  - Data resides at Google.
- Amazon Athena serverless
  - Presto-in-the-cloud. Pay per data accessed.
- Databricks
  - Charged separately for S3 storage and EC2 usage (user account)
  - plus DBUs to Databricks (~EC2 usage)
  - works in AWS & Azure

Centrum Wiskunde & Informatica

Elasticity

- Redshift
  - Co-located storage and compute constrains elasticity
- Snowflake
  - Query-level control through Virtual Warehouse mechanism
- BigQuery
  - Google decides for you based on input table sizes
- Athena
  - Amazon decides for you based on input table sizes
- Databricks Spark
  - DB-level adjustment (cluster size) – dynamically changeable

CWI Centrum Wiskunde & Informatica

Summary

- The MapReduce vs Database debate
  - Big Data technologies are adopting database ideas increasingly
    - Schema, Storage Techniques, Query Execution, …
- Architecture of Analytical Database Systems
  - Understand the basic design areas (storage, query processing, system)
    - Column storage, compression, vectorization/JIT, MinMax pushdown, clustering, partitioning/distribution, update infrastructure, …
- Cloud Database Systems
  - Motivation, Characteristics – differences with on-premise
    - CapEx vs OpEx, time to deployment, elasticity, human factors
    - Absence of data locality
  - Overview of some of the popular systems
    - Redshift, Snowflake, Databricks, BigQuery & Athena
    - How is it charged? How does it scale? Who holds the Data

www.cwi.nl/~boncz/bigdatacourse