
STREAM
INTEGRATION
Big Data
Beyond Hadoop
STREAM
INTEGRATION

BIG DATA – BEYOND THE HADOOP-LA?

The forces of explosive data growth, the demand to know more, and the ambitions of the data science community have converged to create the groundbreaking phenomenon known as big data. Hadoop is certainly at the hypocenter of this convergence, and its platform and analytical capabilities continue to evolve.

However, Hadoop isn't the first technology to focus on addressing the challenges of big data. For more than three decades, the technologies of database management systems and massively parallel processing systems have been evolving to keep up with increasing demand for performance, fault tolerance, flexibility, and scalability.

Whether incorporating Hadoop into their architectures or pioneering new technologies and techniques, BI professionals need to understand the approach taken by leading vendors to address the problems and opportunities associated with big data.

You will learn

How big data affects performance, scalability, fault tolerance, and flexibility

The technologies and techniques that address the problems of big data

How Hadoop is evolving to bridge the gap with traditional BI environments

How vendors are integrating Hadoop into their platforms

Emerging big data technologies

Advanced analytics in areas of complex event processing, machine-generated data, text analytics, and advanced visualization

How to navigate the assortment of technologies marketed as big data solutions

Geared To

BI Program Managers

Project Managers

Designers

Architects

Developers

Business Executives

Managers

STREAM INTEGRATION

Paul Flach

VP, ENTERPRISE ANALYTICS
STREAM INTEGRATION

PAUL.FLACH@STREAMINTEGRATION.COM

Big Data Review

MODULE 1

STREAM INTEGRATION

4 Major Trends

1. The amount of data that enterprises want to analyze is exploding.
   - More sources
   - Production rapidly increasing
   - Analysis is required at more granular level
   - More historical data is required
2. The "cloud" is becoming an increasingly popular environment in which to perform analysis
3. Hadoop has commoditized the market for unstructured data processing, transformation, and analysis.
4. A self-sufficient, result-oriented data science community is emerging

http://wikibon.org/blog/wp-content/uploads/2012/05/big-data-infographic.html

Response to These Trends

Shared-nothing MPP analytical platforms designed precisely to overcome the scalability problems

These "second generation analytical platforms" did extremely well in solving many of the scalability problems

However, trends (2) and (3) started to emerge after the wave of second generation analytical platforms were founded

They created stop-gap solutions in their architecture

- Released "cloud" versions of their software
- Many of them released special "connector" code that facilitates the shipping of data back and forth between their platform and Hadoop

STREAM INTEGRATION

Big Data Spans 3 Dimensions

Volume: Enterprises are awash with ever-growing data of all types, easily amassing terabytes—even petabytes—of information.

- Turn 12 terabytes of Tweets created each day into improved product sentiment analysis
- Convert 350 billion annual meter readings to better predict power consumption

Velocity: Sometimes 2 minutes is too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

- Scrutinize 5 million trade events created each day to identify potential fraud
- Analyze 500 million daily call detail records in real-time to predict customer churn faster

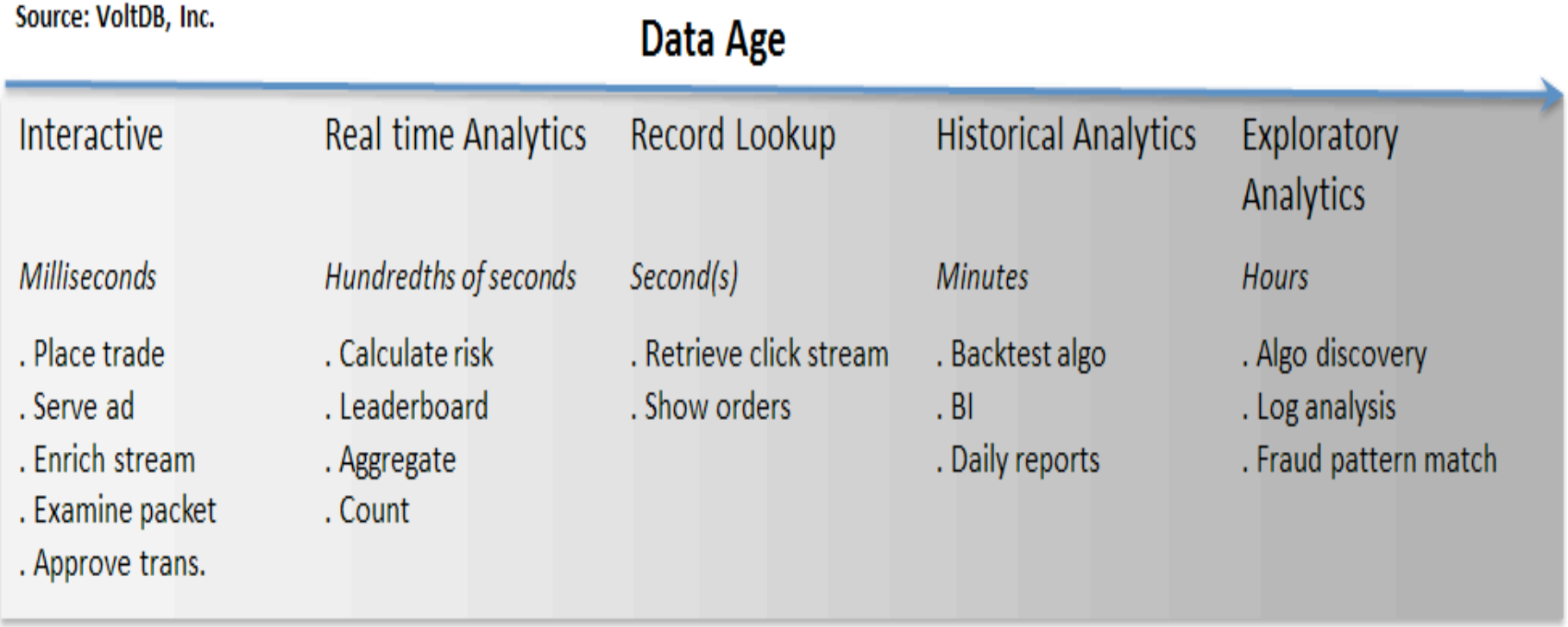
Variety: Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more. New insights are found when analyzing these data types together.

- Monitor 100's of live video feeds from surveillance cameras to target points of interest
- Exploit the 80% data growth in images, video and documents to improve customer satisfaction

STREAM INTEGRATION

Data by Age

Source: VoltDB, Inc.



Questions to Ask

What do I really require of my data store?

What type of data am I using?

Do I need a schema?

What type of analysis am I doing?

What integration is required?

What are my latency requirements?

What is the nature of my transaction processing?

Who will perform the analysis?

STREAM INTEGRATION

DBMS Fundamentals

Module 2

STREAM INTEGRATION

Writing Data to a Database

Rows of data made up of columns are inserted into tables.

```
INSERT INTO Product (ID, Product Name, Price, Type)
VALUES (1,"Bicycle", 159, "Sporting Goods")
```

ID	Product Name	Price	Type
1	Bicycle	159	Sporting Goods

STREAM INTEGRATION

Where are Tables Stored?

Disk Resident Database Systems (DRDB) Why not in Memory?

DBMS stores information on ("hard") disks.

Unlike RAM, time to retrieve a disk page varies depending upon location on disk.

Therefore, relative placement of pages on disk has major impact on DBMS performance!

Transfer Latency

READ: transfer data from disk to main memory (RAM).

WRITE: transfer data from RAM to disk.

Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Costs too much.

Same amount of money will buy you say either 128MB of RAM or 20GB of disk.

Main memory is volatile.

We want data to be saved between runs.

Typical storage hierarchies:

Main memory (RAM) for currently used data (primary storage) .

Disk for the main database (secondary storage).

Tapes for archiving older versions of data (tertiary storage).

STREAM INTEGRATION

Disk Storage Devices

Preferred secondary storage device for high storage capacity and low cost.

Data stored as magnetized areas on magnetic disk surfaces.

A **disk pack** contains several magnetic disks connected to a rotating spindle.

Disks are divided into concentric circular **tracks** on each disk **surface**.

A track is divided into smaller **blocks** or **sectors**

because it usually contains a large amount of information

The division of a track into **sectors** is hard-coded on the disk surface and cannot be changed.

One type of sector organization calls a portion of a track that subtends a fixed angle at the center as a sector.

A track is divided into **blocks**.

The block size B is fixed for each system.

- Typical block sizes range from B=512 bytes to B=4096 bytes.

Whole blocks are transferred between disk and main memory for processing.

Disk Storage Devices (contd.)

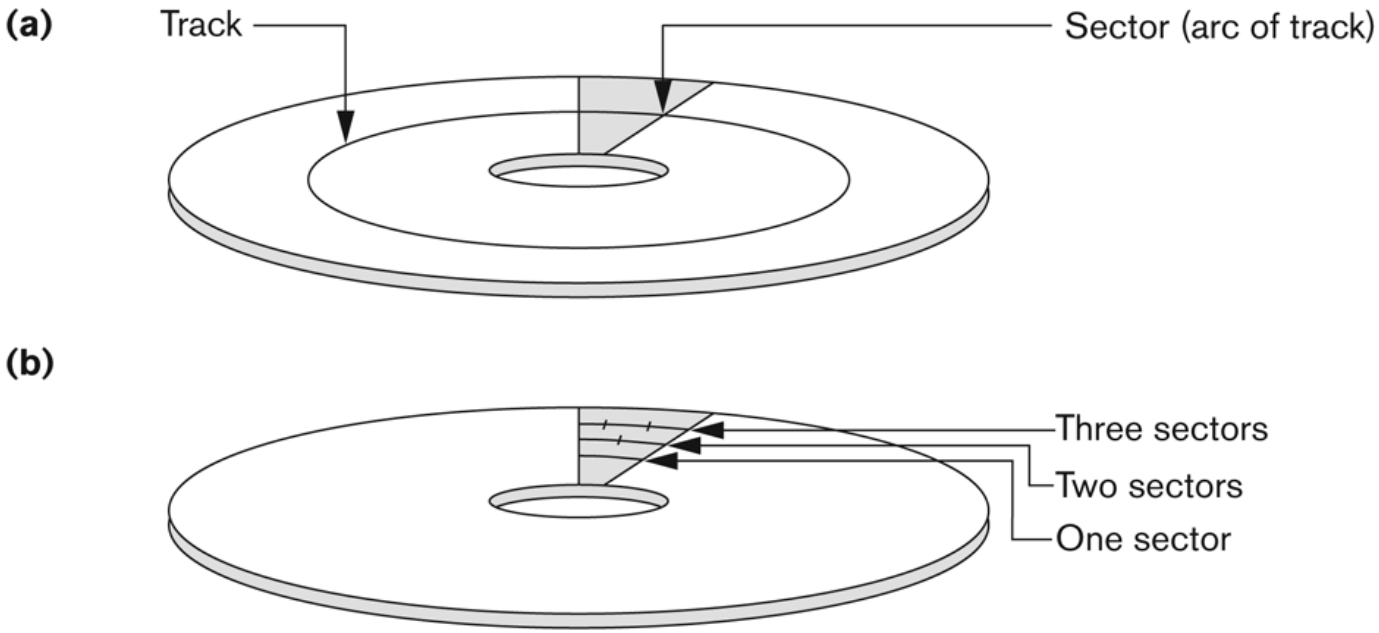


Figure 13.2 Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.

Disk Storage Devices (contd.)

A **read-write head** moves to the track that contains the block to be transferred.

- Disk rotation moves the block under the read-write head for reading or writing.

A physical disk block (hardware) address consists of:

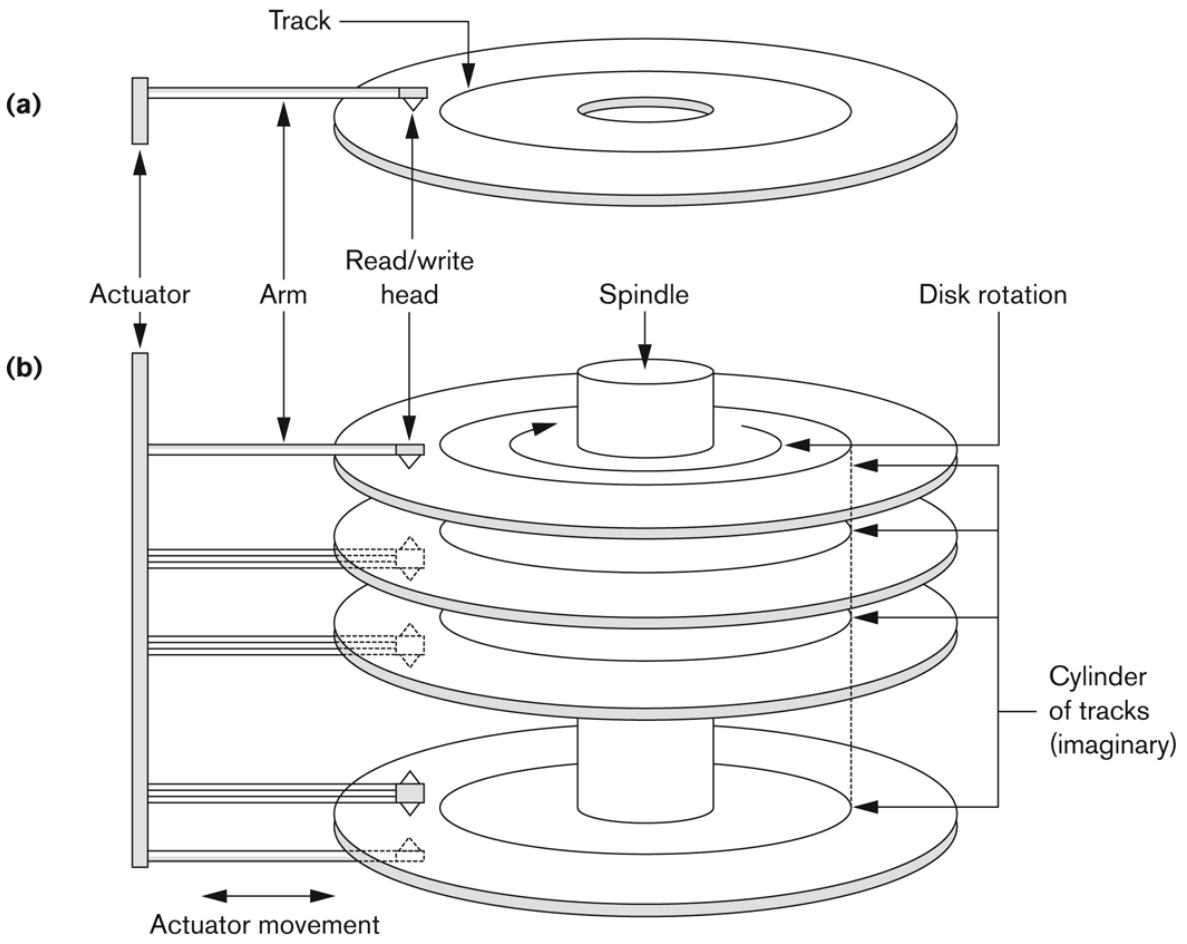
- a cylinder number (imaginary collection of tracks of same radius from all recorded surfaces)
- the track number or surface number (within the cylinder)
- and block number (within track).

Reading or writing a disk block is time consuming because of the seek time s and rotational delay (latency) **rd**.

Double buffering can be used to speed up the transfer of contiguous disk blocks.

Disk Storage Devices (contd.)

Figure 13.1
(a) A single-sided disk with read/write hardware. (b) A disk pack with read/write hardware.



Accessing a Disk Page

Time to access (read/write) a disk block:

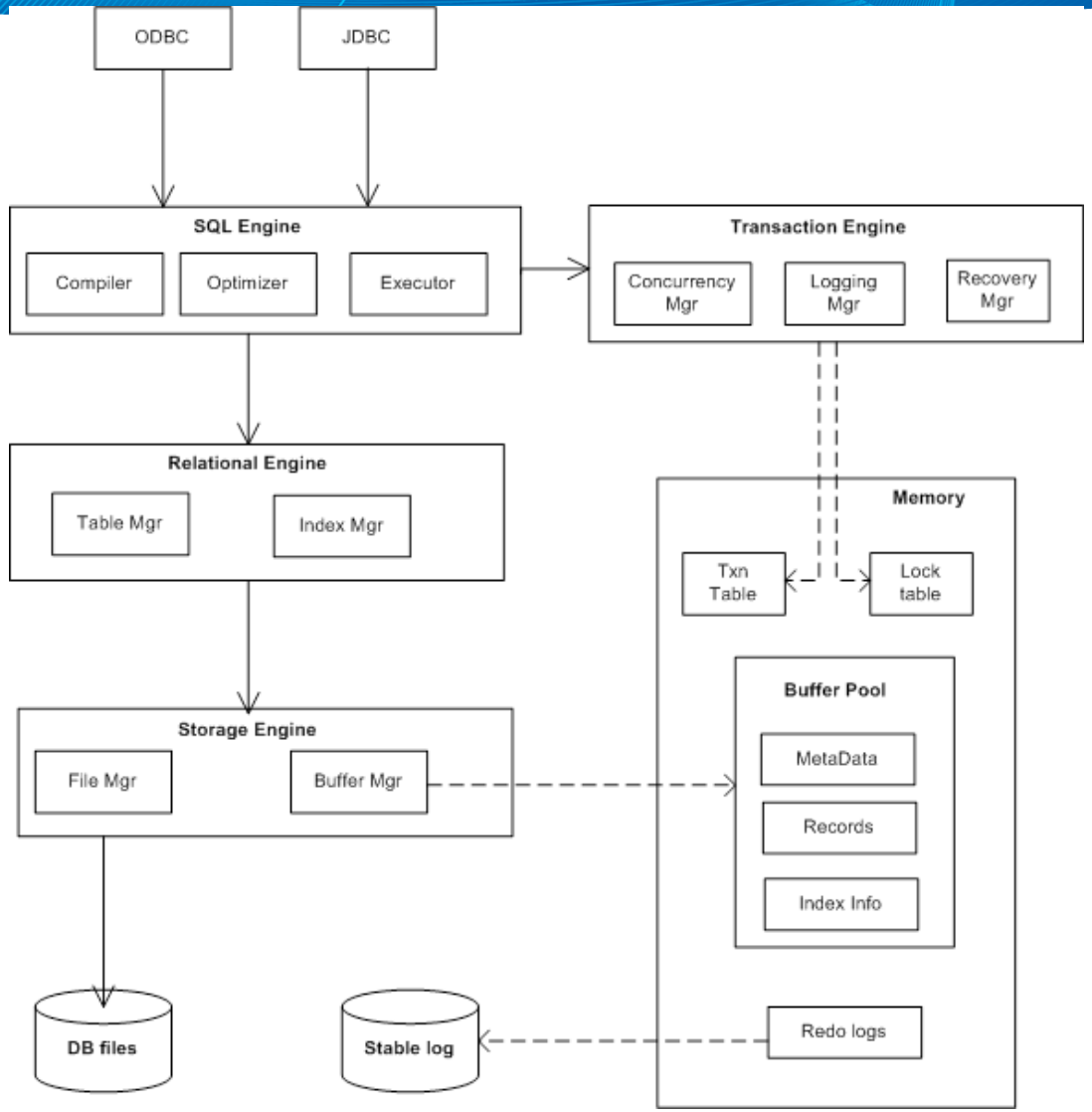
- *seek time* (moving arms to position disk head on track)
- *rotational delay* (waiting for block to rotate under head)
- *transfer time* (actually moving data to/from disk surface)

Seek time and rotational delay dominate.

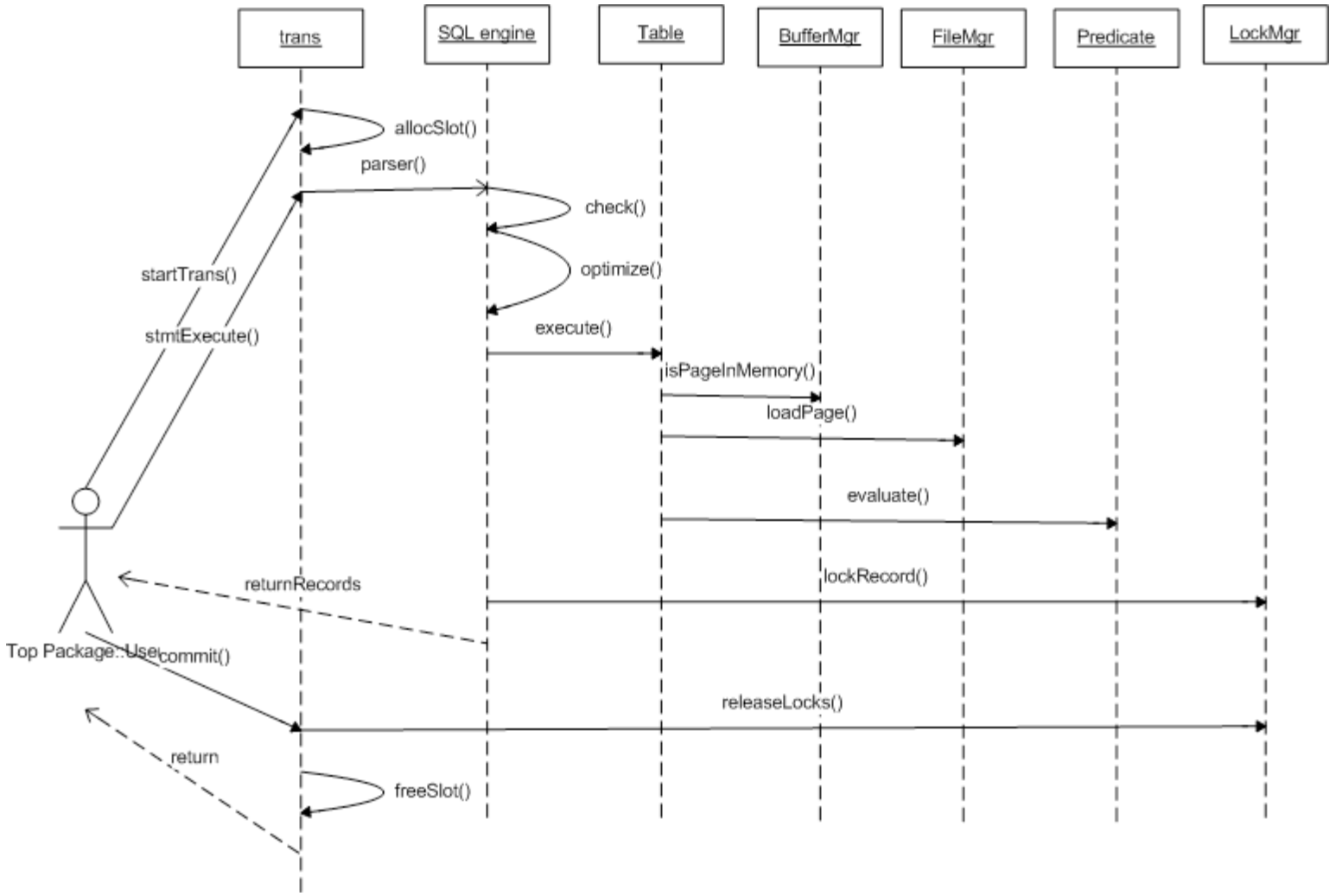
- Seek time varies from about 1 to 20msec
- Rotational delay varies from 0 to 10msec
- Transfer rate is about 1msec per 4KB page

Lower I/O cost: reduce seek/rotation delays

DRDB Architecture



DBMS Select Statement



Index

A data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space

Can be created using one or more columns of a database table

There are many different data structures used for this purpose

- a substantial proportion of the field of Computer Science is devoted to the design and analysis of index data structures

There are complex design trade-offs involving lookup performance, index size, and index update performance

STREAM INTEGRATION

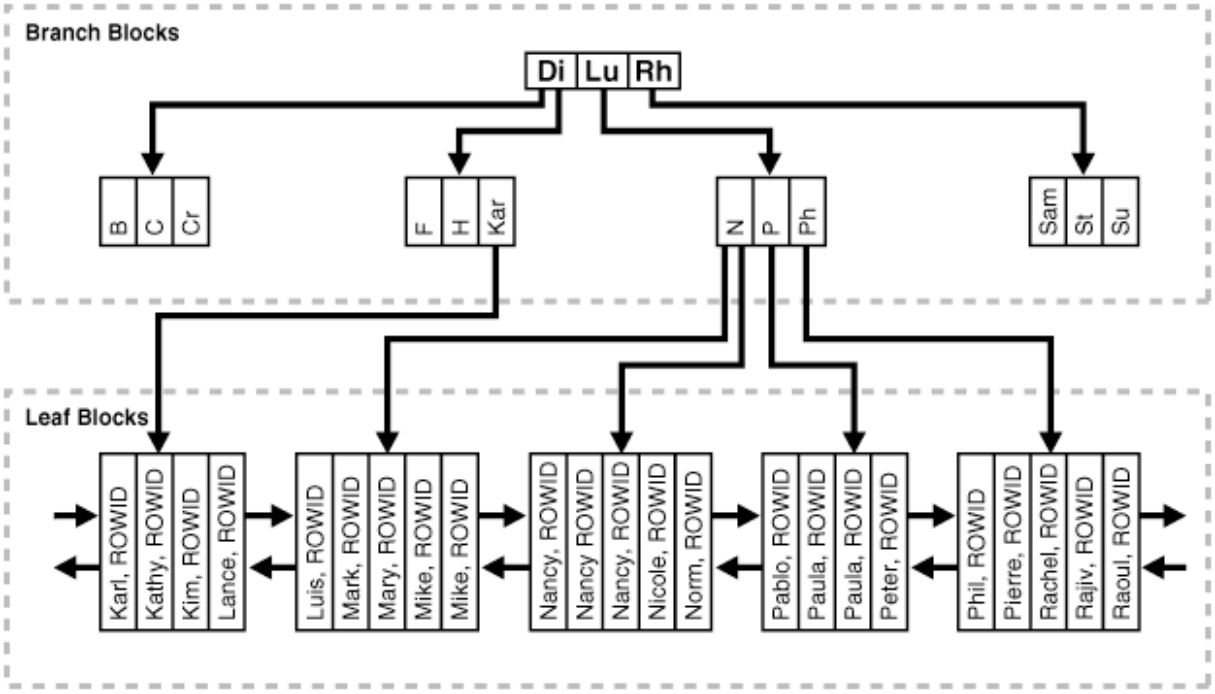
B-Tree index

The upper blocks (branch blocks) of a B-tree index contain index data that points to lower-level index blocks.

The lowest level index blocks (leaf blocks) contain every indexed data value and a corresponding rowid used to locate the actual row.

As you insert new rows into the table, new rows are inserted into index leaf blocks.

When a leaf block is full, another insert will cause the block to be split into two blocks, which means an entry for the new block must be added to the parent branch-block



How are Indexes Used?

To quickly find specific rows by avoiding a Full Table Scan

A range scan will find the first row in the range using the same technique as the Unique Scan, but will then keep reading the index up to the end of the range.

To avoid a table access altogether

If the information is in the index, then it doesn't bother to read the table.

To avoid a sort

If a sort operation requires rows in the same order as the index, the DBMS may read the table rows via the index. A sort operation is not necessary since the rows are returned in sorted order.

Full Table Scans are Not Bad

An index can make a query *slower*

Particularly common in batch systems that process large quantities of data

A Full Table Scan selects rows in no particular order, except that they are nearest to hand

When a block is read from disk, the DBMS caches the next few blocks with the expectation that it will be asked for them very soon

Full Table Scan is the faster way to pick up *every* record

Index is the faster way to pick up *just a particular record*, or even 100 particular records

The main reasons for this are:

Reading a table in indexed order means more movement for the disk head

A DBMS cannot read single rows

- the entire block must be read with all but one row discarded
- an index scan of 100 rows would read 100 blocks, but a FTS might read 100 rows in a single block

Accessing the entire index *and* the entire table is still more IO than just accessing the table

STREAM INTEGRATION

Data Partitioning

Vertical Partitioning

Involves creating tables with fewer columns and using additional tables to store the remaining columns. Techniques:

- Different physical storage might be used to realize vertical partitioning
- Storing infrequently used or very wide columns on a different device
- Split dynamic data (slow to find) from static data (fast to find)

Horizontal Partitioning

Involves putting different rows into different tables

- Eg. customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest
- The two partition tables become CustomersEast and CustomersWest, while a view with a union might be created over both of them to provide a complete view of all customers

STREAM INTEGRATION

Horizontal Partitioning

Partitioning Strategies

Hash - as each row is loaded, a hash function is applied to one or more attributes of each row to determine the target node and disk where the row should be stored.

Range - Selects a partition by determining if the partitioning key is inside a certain range. An example could be a partition for all rows where the column zipcode has a value between 70000 and 79999.

Round-Robin - the first record of an input data set goes to the first processing node, the second to the second processing node, and so on. When you reach the last processing node in the system, start over

Partitioning across nodes provides scalable query performance

Partitioned Query Execution

Partitioning across nodes provides scalable query performance

Consider:

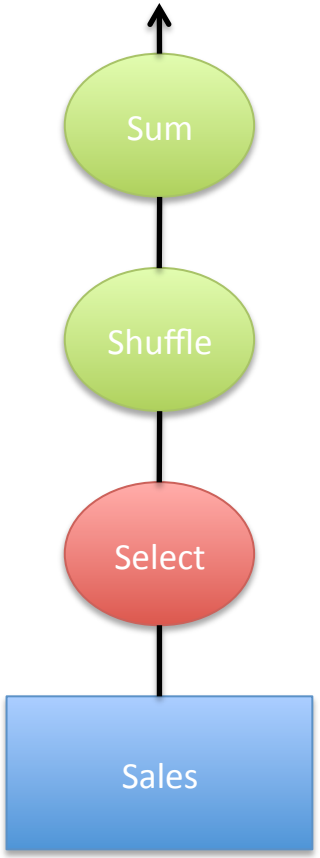
```
SELECT custId, amount FROM Sales
  WHERE date BETWEEN
  "12/1/2009" AND "12/25/2009";
```

- Query can be executed in parallel by executing a SELECT operator against the Sales records with the specified date predicate on each node of the cluster.
- The intermediate results from each node are then sent to a single node that per- forms a MERGE operation

Partitioned Execution of Aggregation

Suppose we would like to know the total sales amount for each custId within the same date range. This is done through the following query:

```
SELECT custId, SUM(amount)
FROM Sales
  WHERE date BETWEEN
  "12/1/2009" AND
    "12/25/2009"
  GROUP BY custId;
```

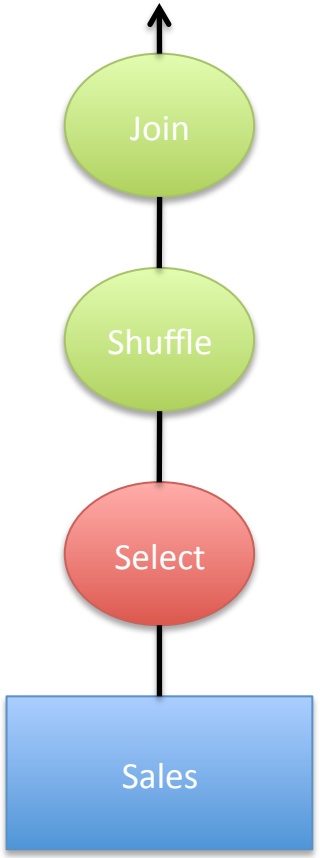


STREAM INTEGRATION

Partitioned Execution of a Join

finding the names and email addresses of customers who purchased an item costing more than $1,000 during the holiday shopping period:

```
SELECT C.name, C.email
  FROM Customers C, Sales S
WHERE C.custId = S.custId
  AND S.amount > 1000
AND S.date BETWEEN
  "12/1/2009" AND
  "12/25/2009";
```



STREAM INTEGRATION

Compression

The task of compression consists of two components:

An *encoding* algorithm that takes a message and generates a "compressed" representation

A *decoding* algorithm that reconstructs the original message or some approximation of it from the compressed representation

Lossless vs lossy algorithms

Compression is all about probability

Model and Coder

The *model* component somehow captures the probability distribution of the messages by knowing or discovering something about the structure of the input.

The *coder* component then takes advantage of the probability biases generated in the model to generate codes.

Entropy - is a measure of predictability

STREAM INTEGRATION

Brewer's CAP Theorem (or PACELC)

Of the three desirable properties you want in your distributed system (consistency, availability, and tolerance of network partition), you can only choose two

Consistency - all nodes see the same data at the same time

Availability - a guarantee that every request receives a response about whether it was successful or failed

Partition tolerance - the system continues to operate despite arbitrary message loss or failure of part of the system

STREAM INTEGRATION

ACID

A set of properties that guarantee that database transactions are processed reliably

Atomicity: Either all or none of the effect should appear in database after transaction completes.

Consistency: Constraints should always keep the database in consistent state

Isolation: Transaction should run as though no other transaction is running.

Durability: Once the transaction completes, effect of the transaction on the database must never be lost.

STREAM INTEGRATION

Survey the Big Data Landscape

MODULE 3

Next Generation Data Warehousing

- Massively parallel processing, or MPP, capabilities
- Shared-nothing architectures
- Columnar architectures
- Advanced data compression capabilities
- Commodity hardware

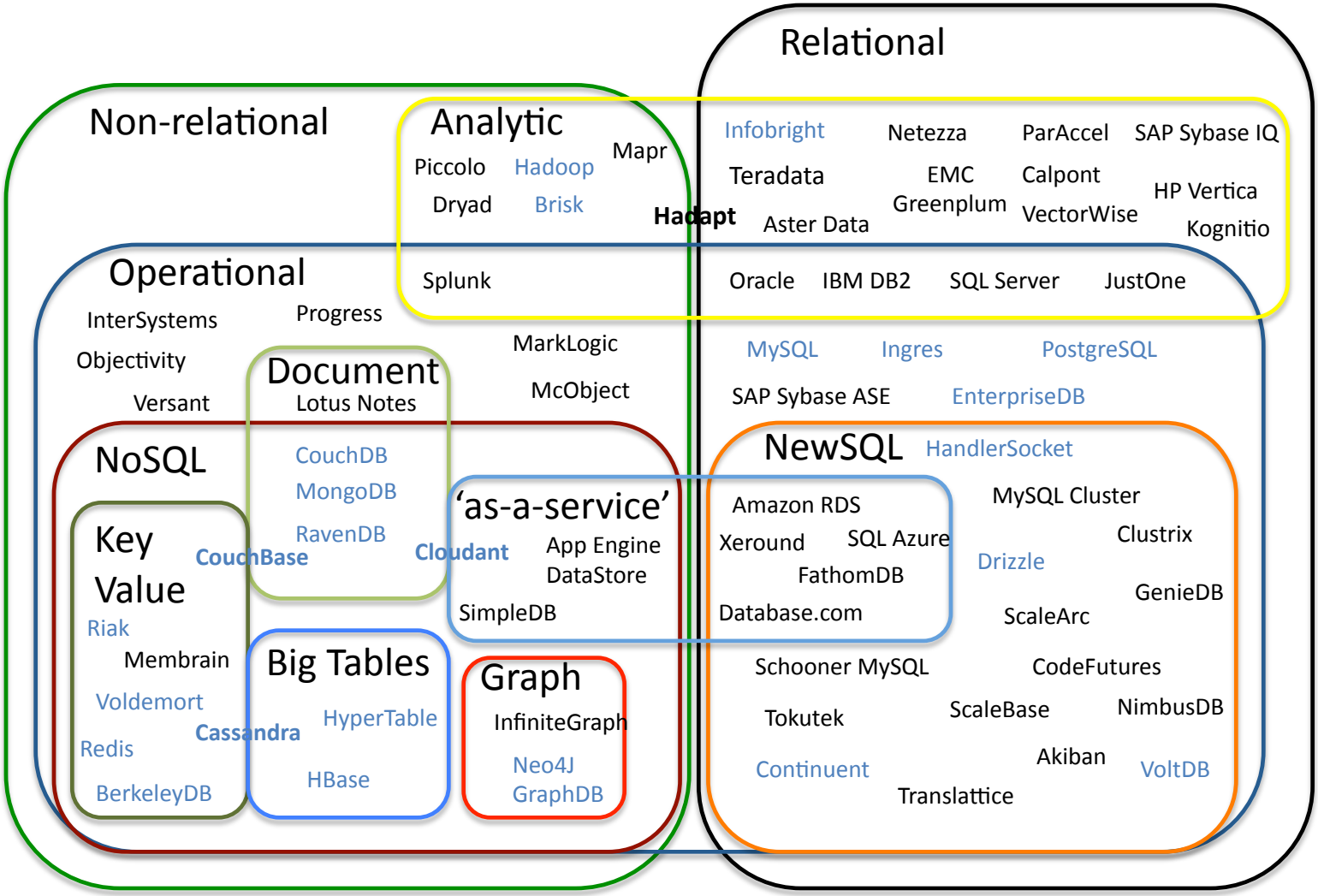
Market Segments

BIG DATA Market Segments

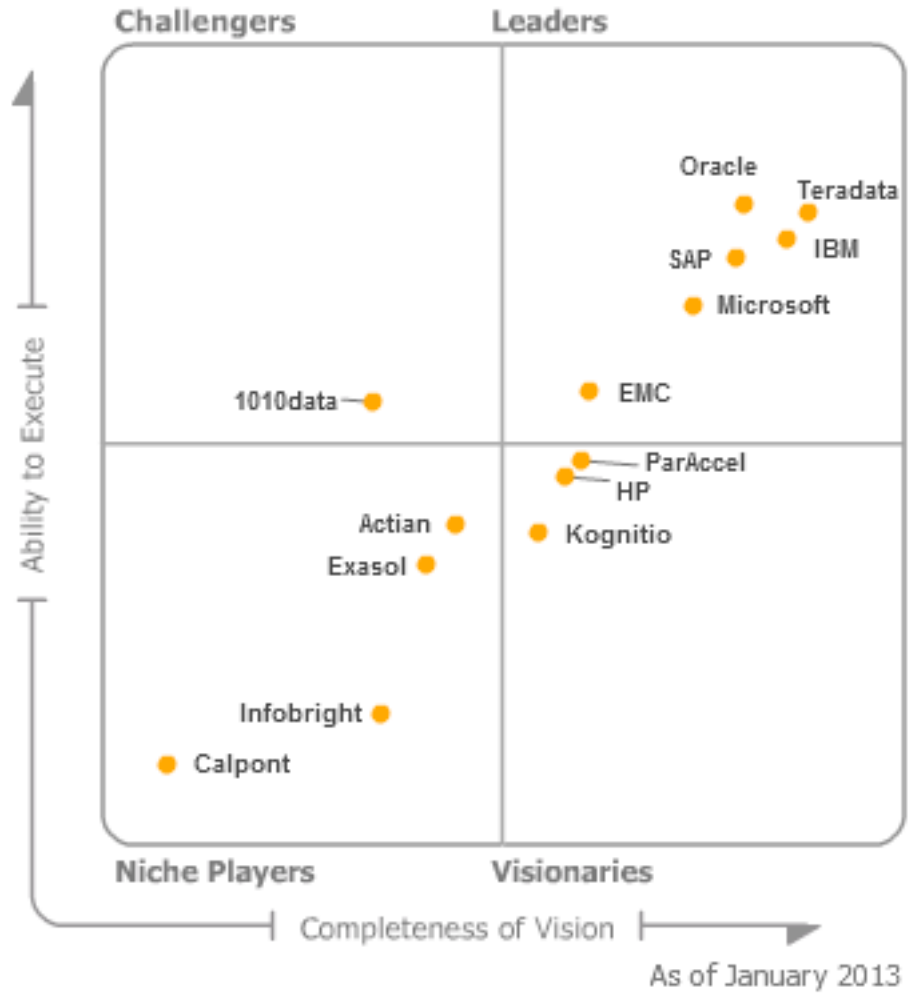
Hardware	Big Data Distributions	Data Management Components	Analytics Layer	Applications Layer	Services
• Storage • Servers • Networking Vendors include Dell, HP, Arista, IBM, Cisco, EMC, NetApp.	• Open source Hadoop distributions • Enterprise Hadoop distributions • Non-Hadoop Big Data frameworks Vendors/ providers include Apache, Cloudera, Hortonworks, IBM, EMC, MapR, LexisNexis.	• Distributed file stores • NoSQL databases • Hadoop-optimized data warehousing • Data integration • Data quality and governance Vendors/providers include Apache, DataStax, Pervasive Software, Couchbase, IBM, Oracle, Informatica, Syncsort, Talend.	• Analytic application development platforms • Advanced analytics applications Vendors/providers include Apache, Karmasphere, Hadapt, Attivio, 1010data, EMC, SAS Institute, Digital Reasoning, Revolution Analytics.	• Data visualization tools • Business intelligence applications Vendors include Datameer, ClickFox, Platfora, Tableau Software, Tresata, IBM, SAP, Microstrategy, Pentaho, QlikTech, Japersoft.	• Consulting • Training • Technical support • Software maintenance • Hardware maintenance • Hosting/Big-Data-as-a-Service/cloud Vendors include Tresata, Tidemark, Think Big Analytics, Amazon Web Services, Accenture, Cloudera, Hortonworks.
Next Generation Data Warehouse Appliances					
• MPP, columnar data warehouse appliances. • In-memory analytics engines • Fast data loading Vendors include EMC Greenplum, HP Vertica, Teradata Aster, IBM Netezza, Kognitio, ParAccel.					

STREAM INTEGRATION

Data management Categorization



DW DBMS Magic Quadrant



BI and Analytics Platforms

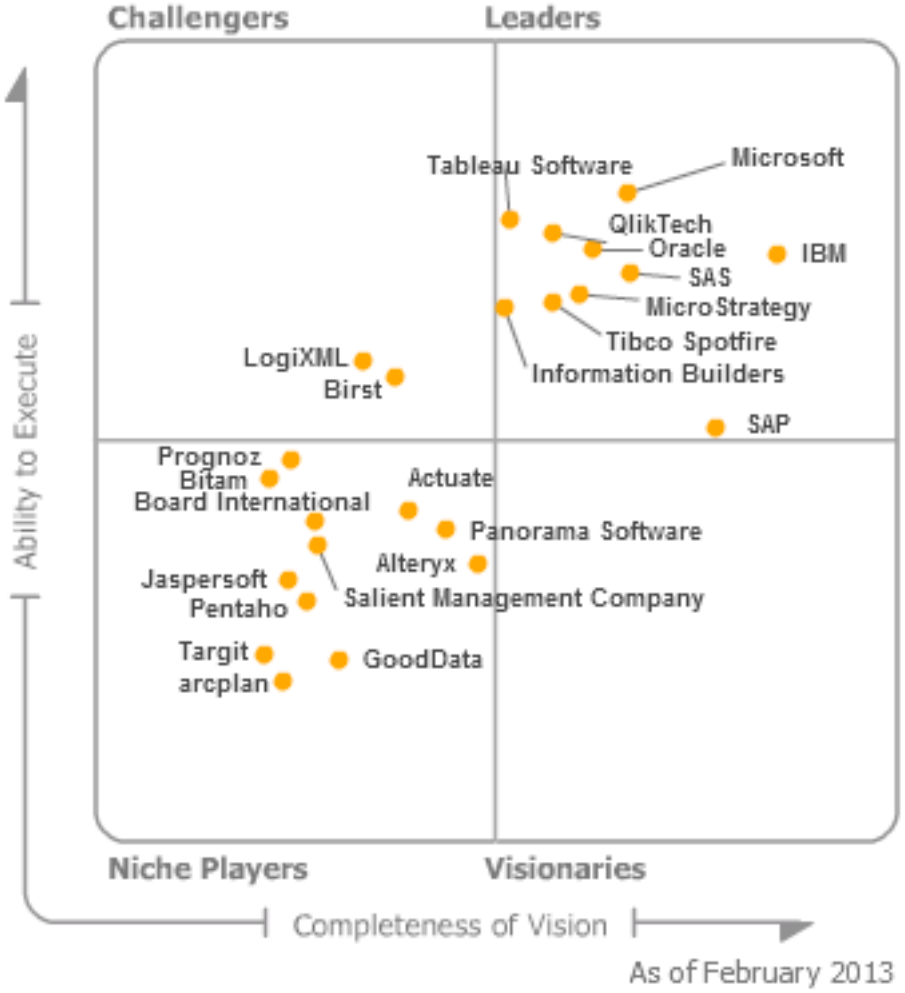
How big is your database? How big is your budget?

How do you feel about appliances?

How do you feel about the cloud?

What are the size and shape of your workload?

How fresh does the data need to be?



Relational – Analytic Databases

Module 4

STREAM INTEGRATION

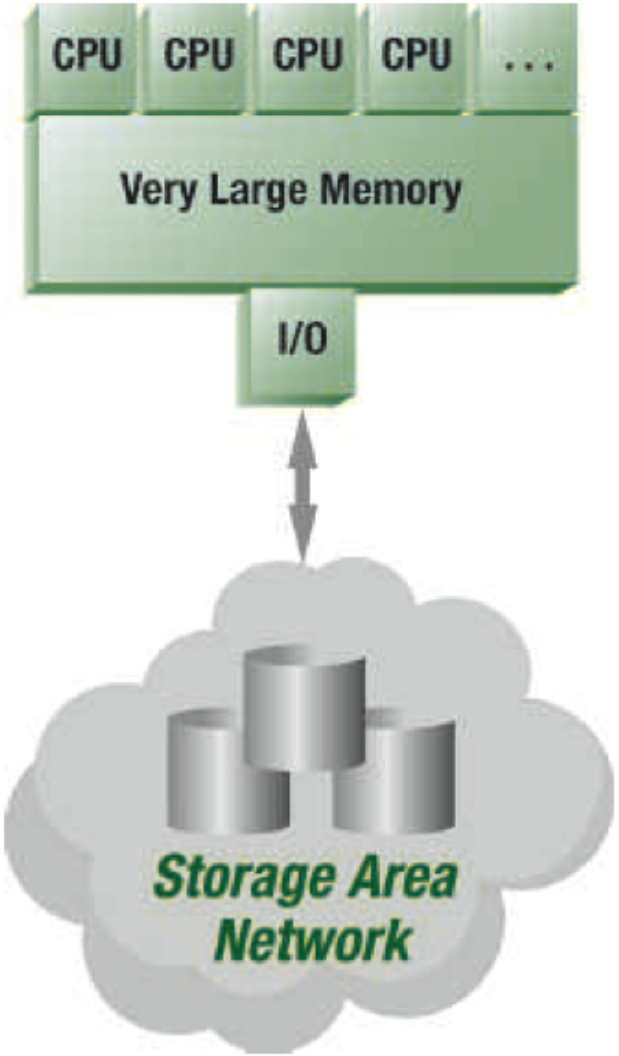
Symmetric Multiprocessing (SMP)

Characteristics

- Several processors, each with its own memory cache
- Threads of code are distributed across processors by the OS
- Load is balanced across processors
- Single bank of memory shares data among tasks

Limitations

- Not able to move large amounts of data
- Scaling is not linear due to memory and I/O resource bottlenecks



STREAM INTEGRATION

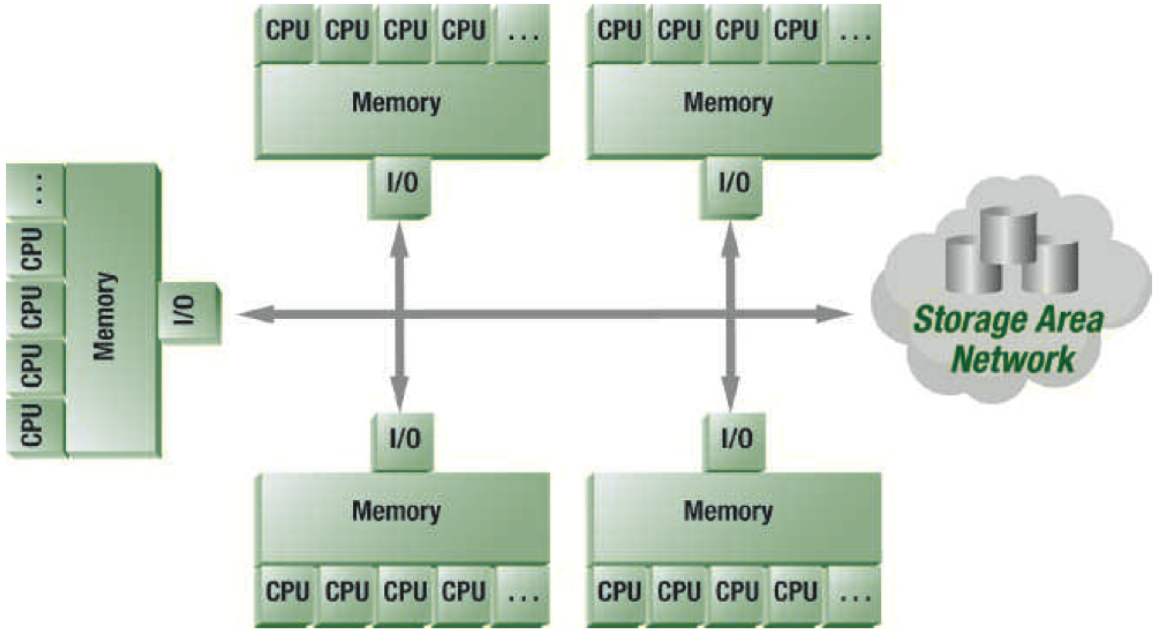
Massively Parallel Processing (MPP)

Characteristics

- Each CPU within an SMP node shares RAM with its neighbors
- Shared access to the storage network over a shared I/O bus

Limitations

- Imposes a bottleneck that limits performance and scalability
- Nodes are contending for access to storage over a common I/O bus



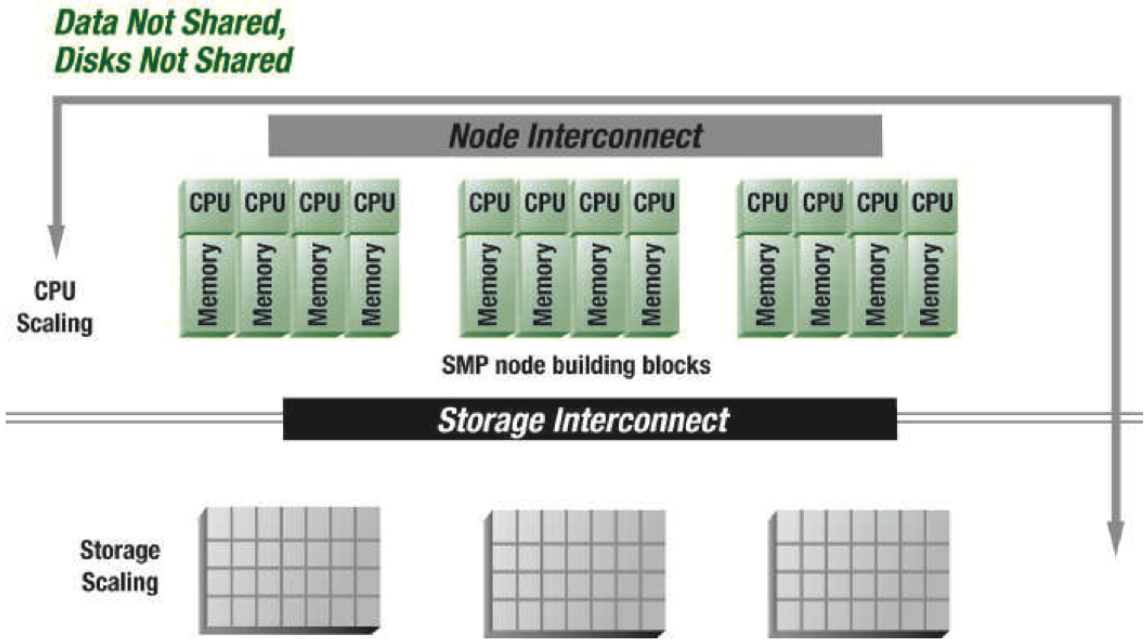
Shared Nothing Architecture

Characteristics

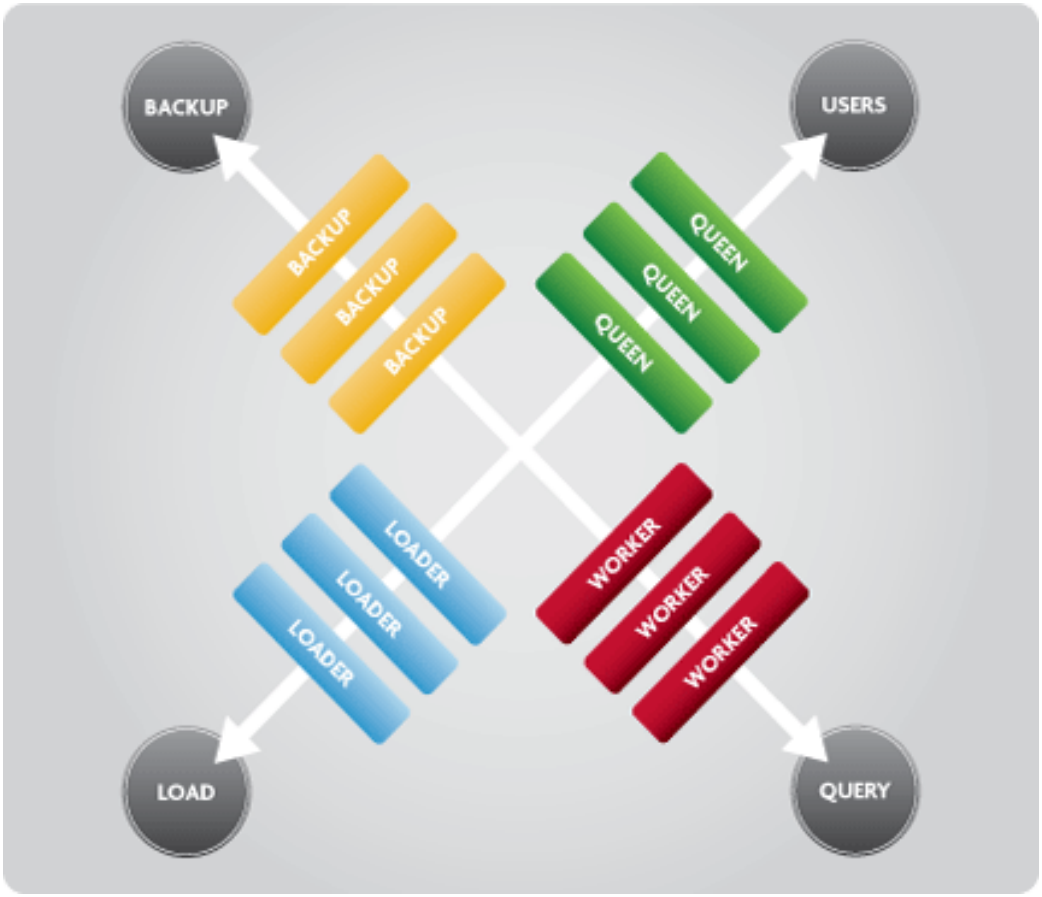
- Each processor communicates with its associated disk drive
- One processor collects the intermediate results and assembles the query response
- No contention for resources

Limitations

- Requires significant movement of data



Aster Data



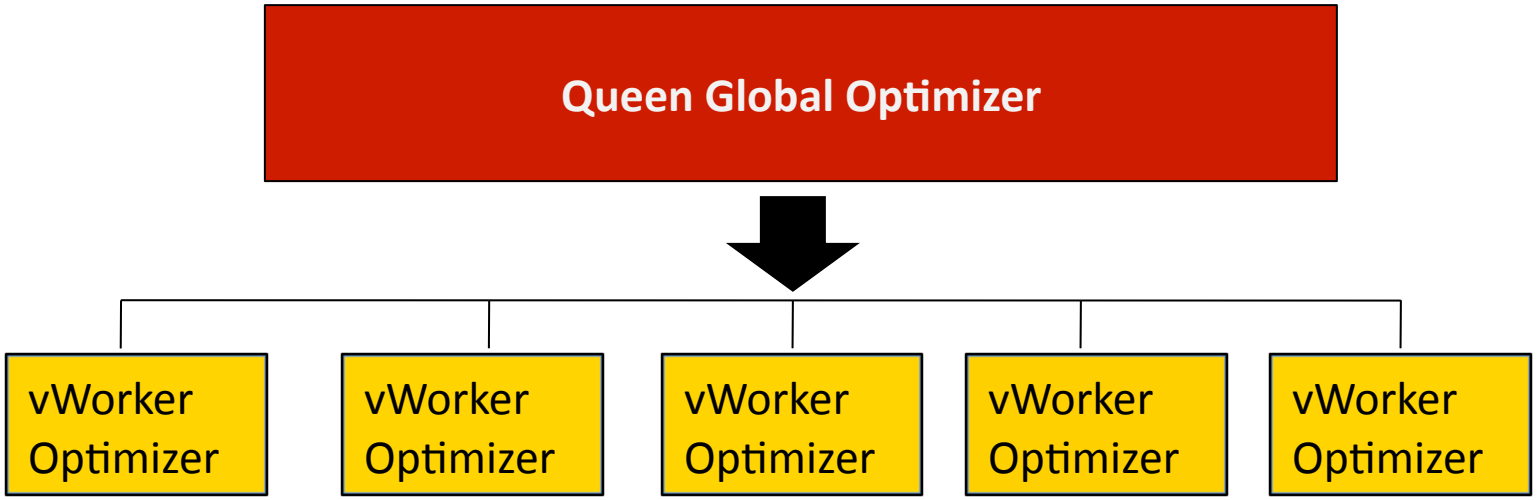
Incremental Scaling

- add Workers for more data storage/processing
- add Loaders for more data load per hour/day
- add a redundant Queen for fast system recovery
- add a Backup nCluster for data archiving

All scaling done through the Aster Management Console

STREAM INTEGRATION

nCluster Query Execution



Queen

Admission control, session management

Global query optimization – rule based, not cost based. Maximize computation at vWorkers and minimize network transfer.

Assemble final query result, including global operations like ORDER BY

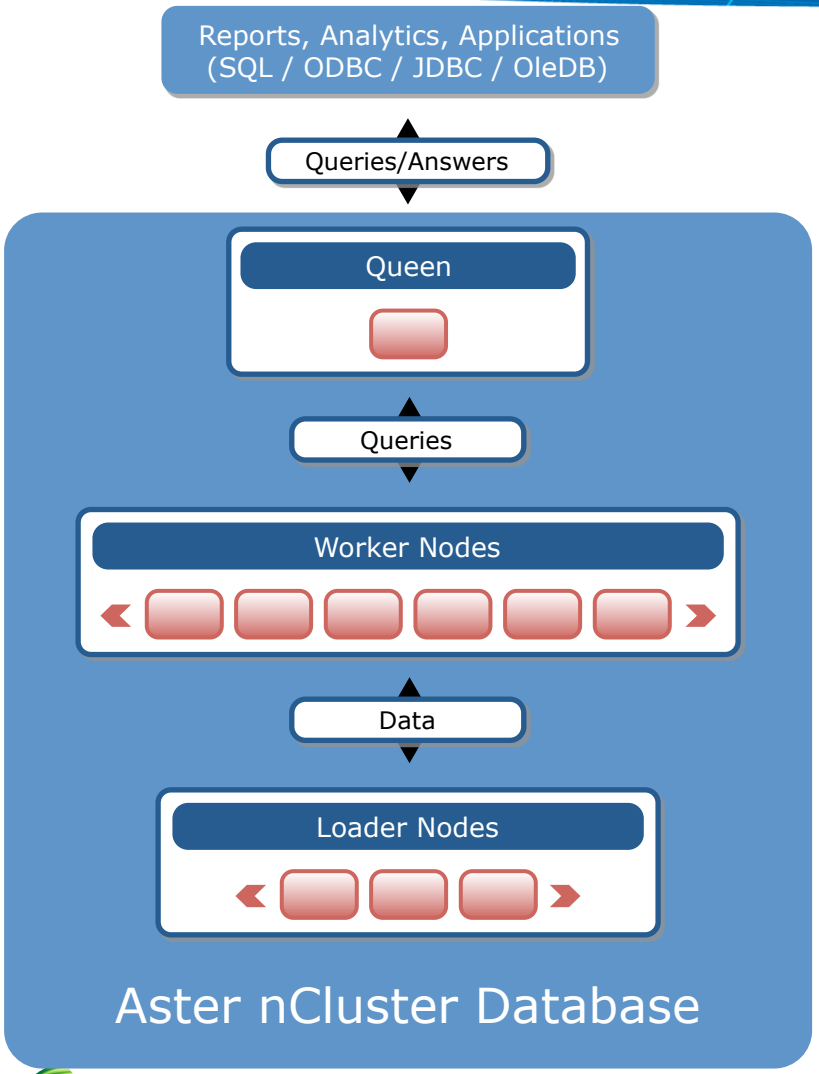
vWorker

Parallel query execution

Cost-based query optimization on local data

STREAM INTEGRATION

nCluster Architecture



Key Architectural Elements:

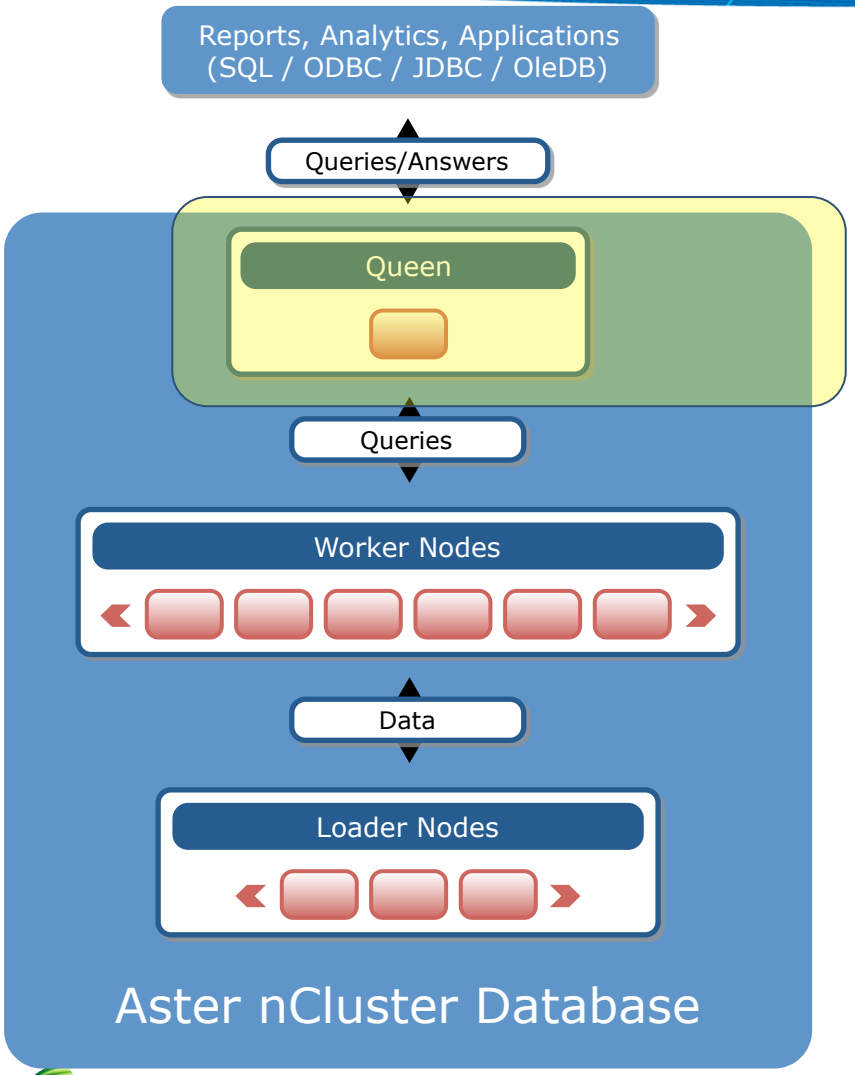
Cluster of tens to hundreds of commodity hardware boxes managed as a single database.

4 independent (share nothing) tiers for management, query processing, loading, & backup.

In-database SQL-MapReduce enables parallelization of query and analytics processing.

STREAM INTEGRATION

nCluster: The Queen



Queen

Manages system, configurations, schema, and error handling

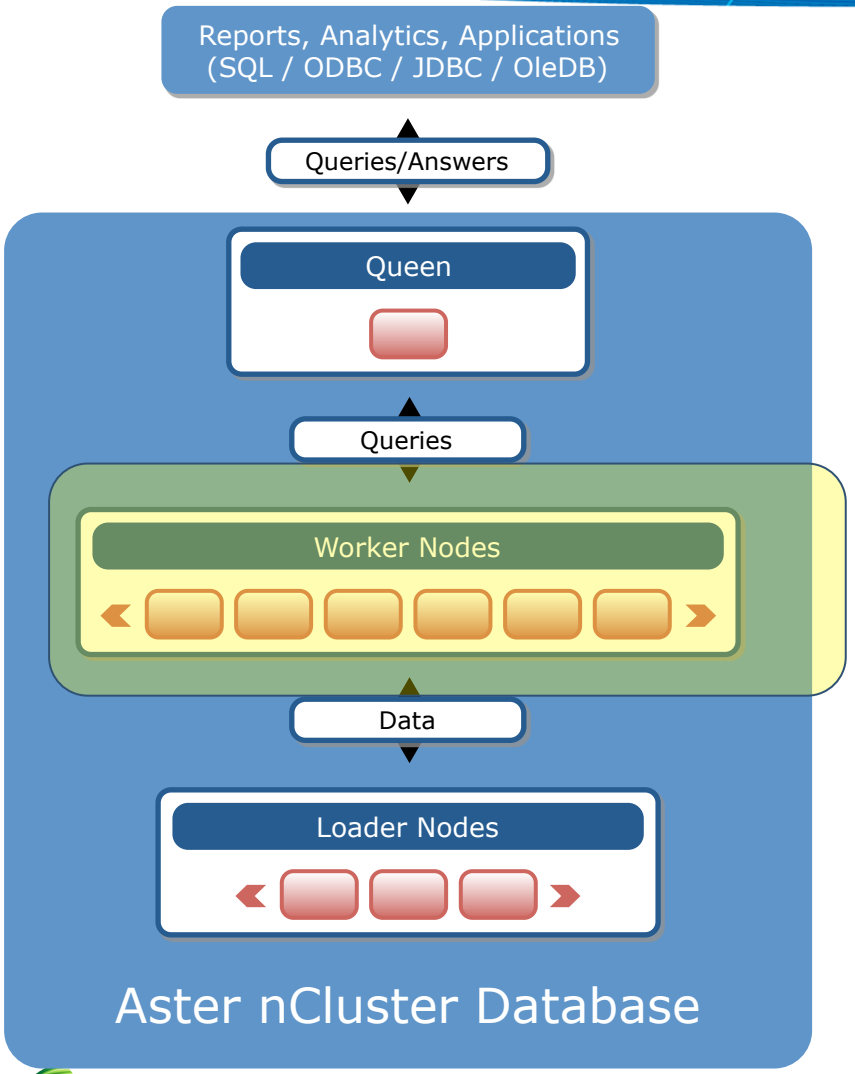
Presents query interface, SQL/ODBC/ JDBC/OleDB "The Face of nCluster"

Global query optimizer coordinates queries in 3 steps:

- Parses SQL statements and hands off to Planner
- Planner develops a set of sub-queries to be executed
- Executor will execute sub-queries and aggregate results

STREAM INTEGRATION

nCluster: Worker Nodes



Worker Node

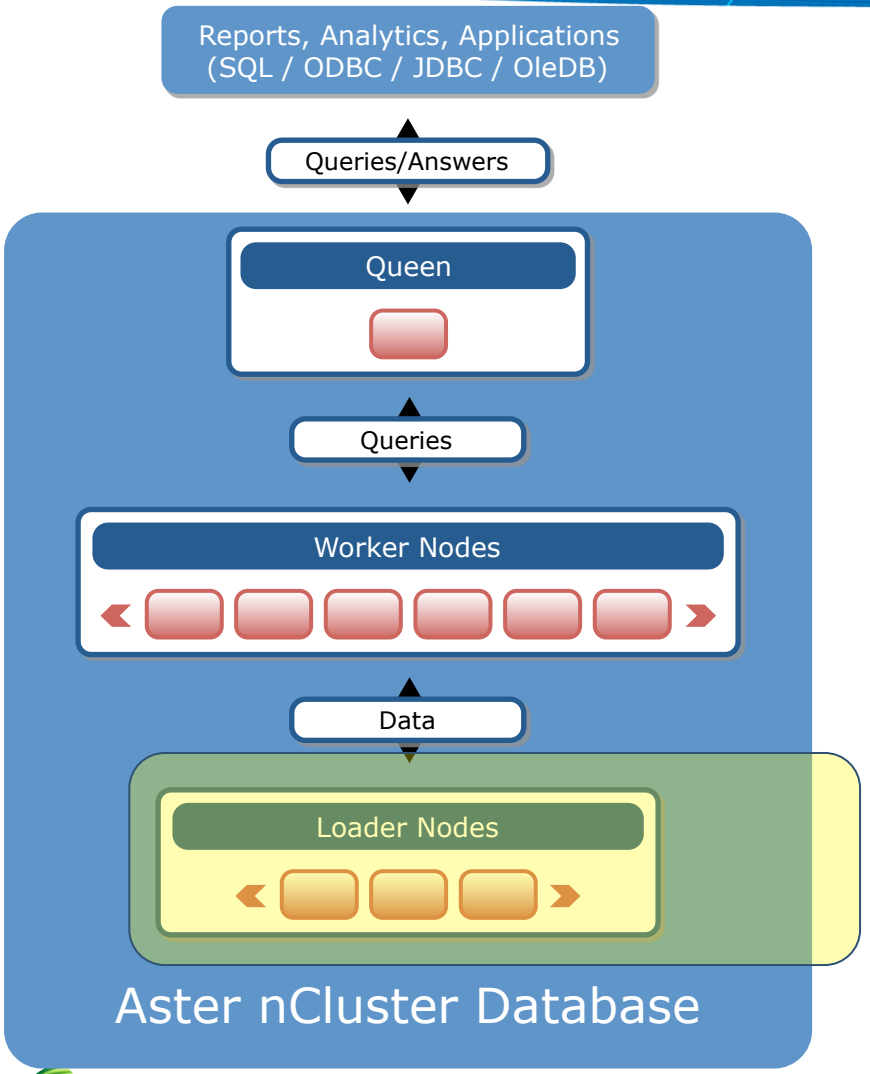
Stores data and interacts with the queen and other workers.

Executes queen's orders:

- Run queries
- Replicate
- Balance storage
- Balance processing

Local query optimizer

nCluster: Loader Nodes



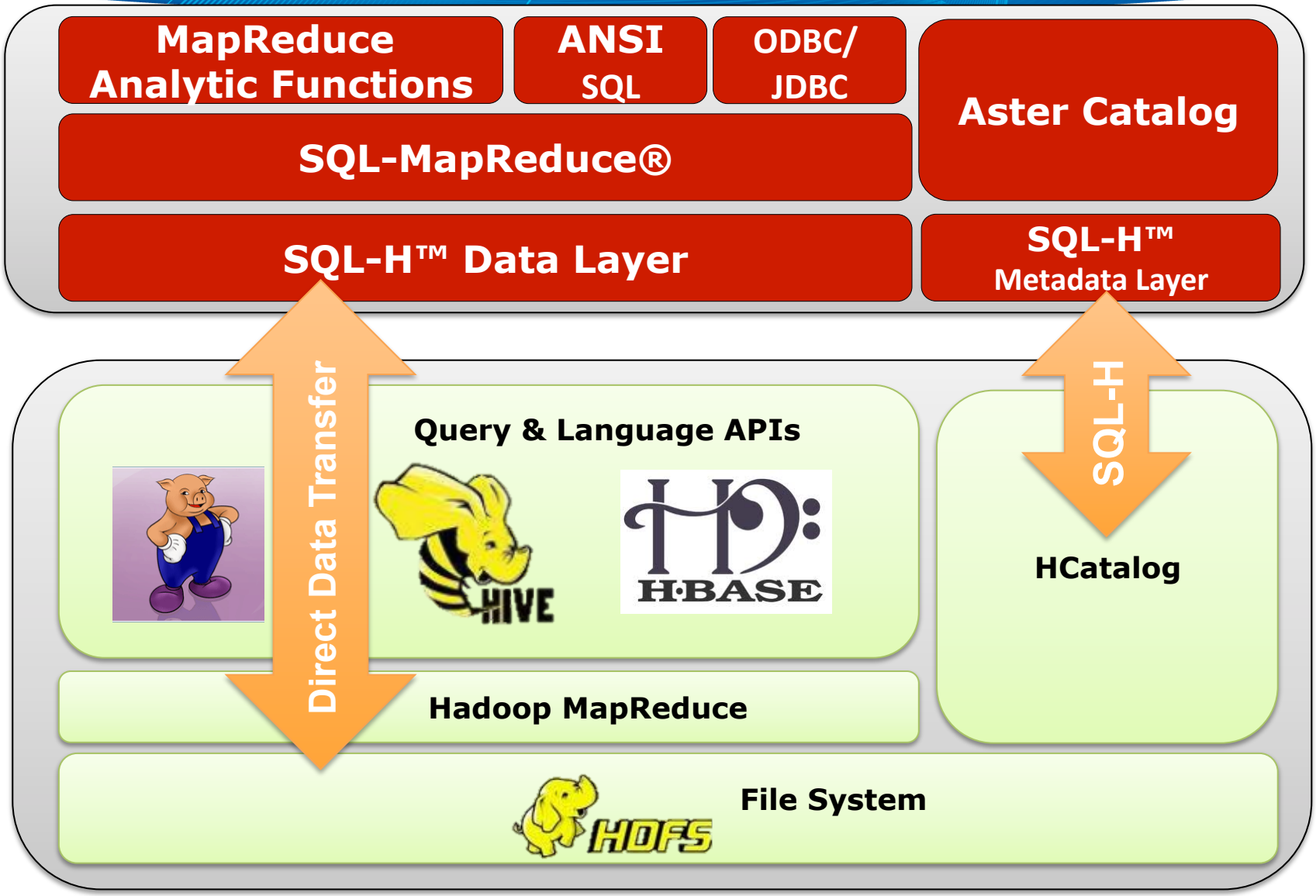
Loader Node

Receives new data from bulk loading clients (for example, nCluster_loader, Informatica, similar tools)

Partitions this data into appropriate segments

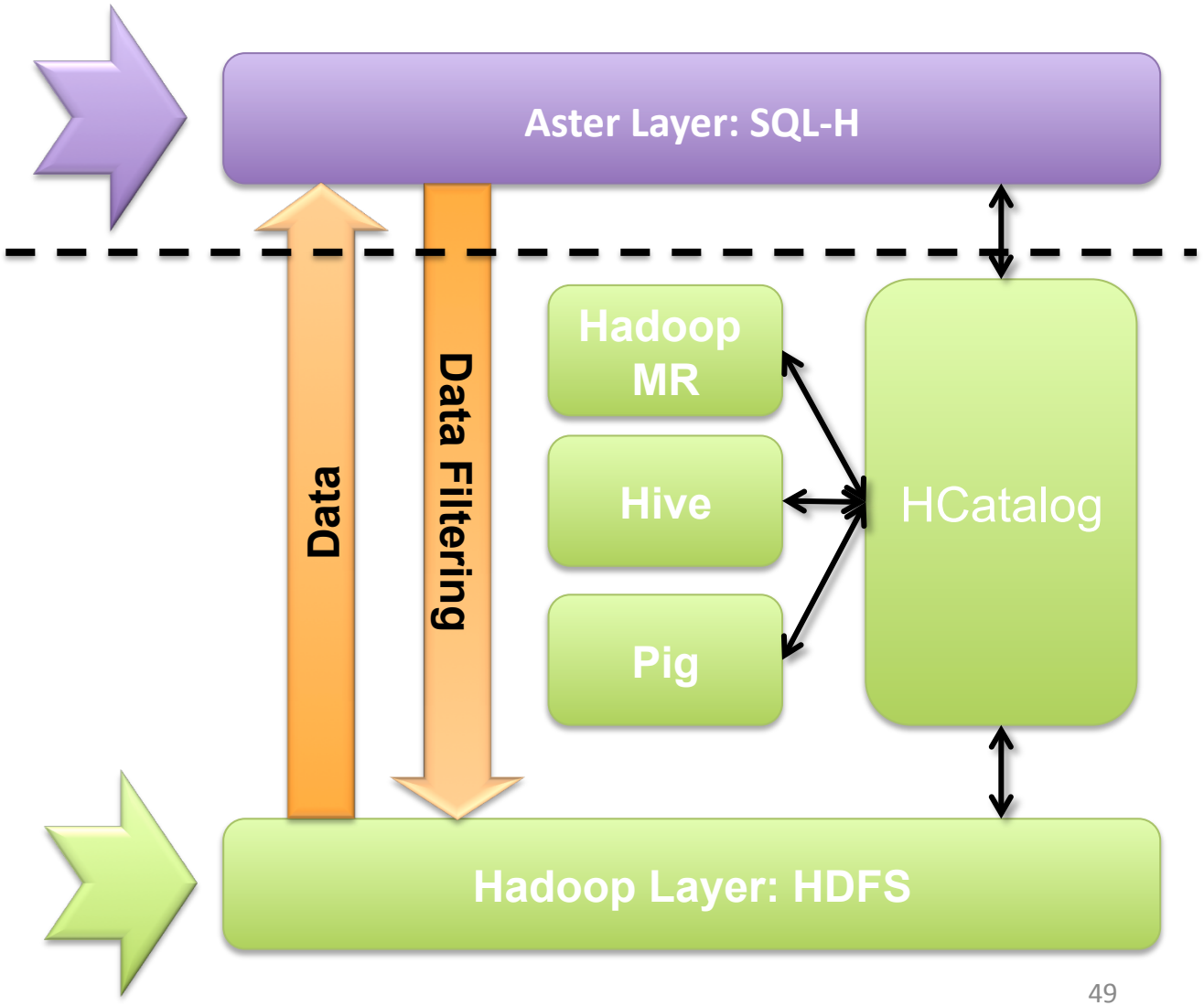
Distributes the segmented data across vworkers

SQL-H™ Architecture



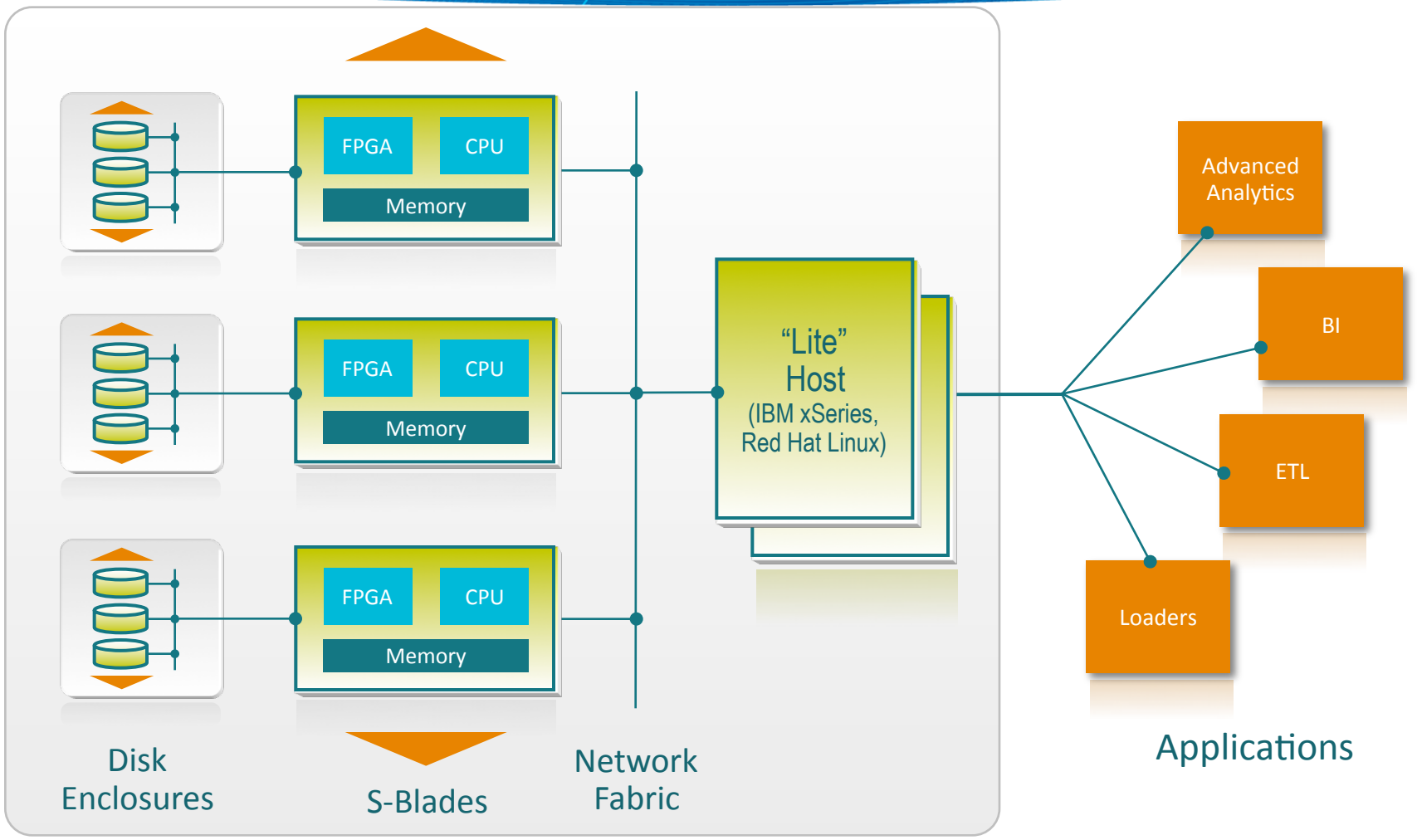
Data & Processing Locality in SQL-H

- SQL & SQL-MapReduce execution
- Intermediate data persistence
- Optional: HDFS data subset persistence for maximum performance



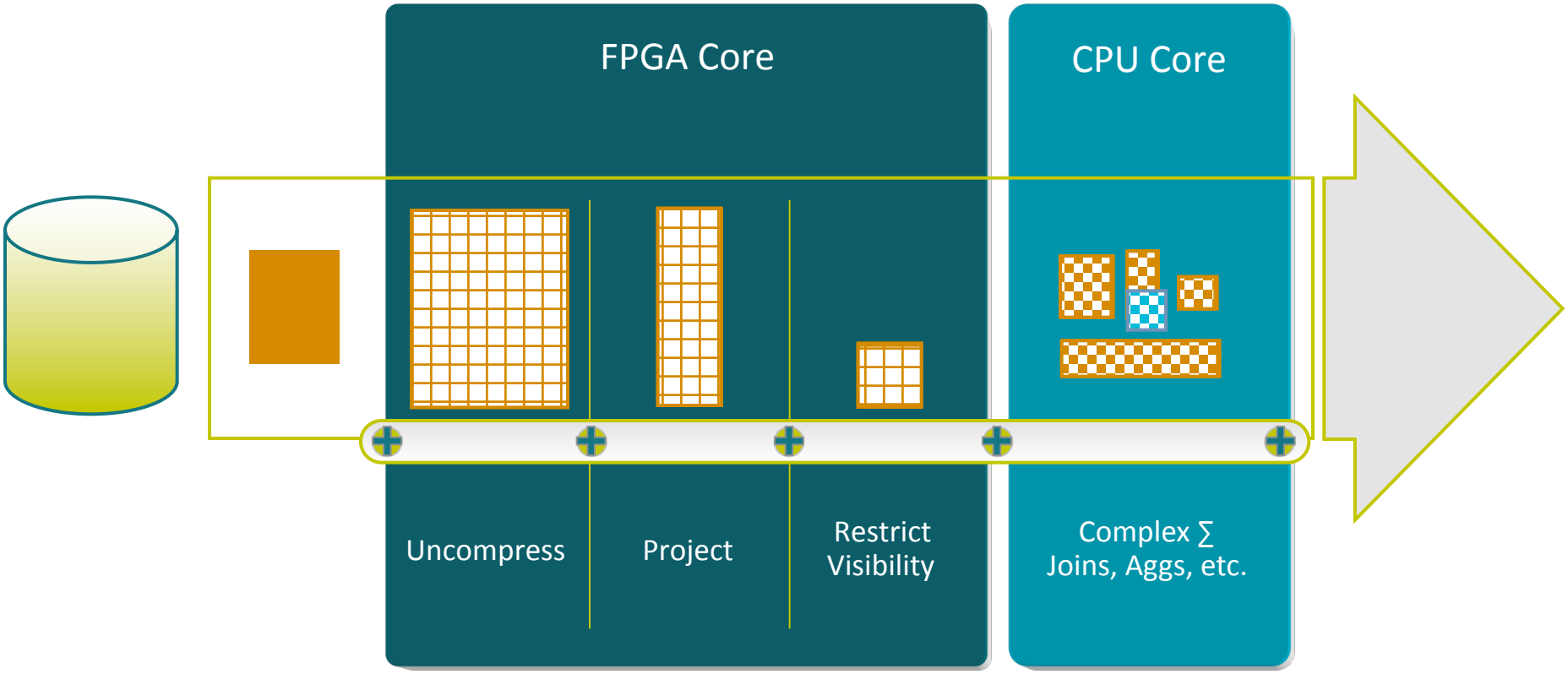
- Select appropriate subset of data based on tables & partitions in Hcatalog
- Directly & in parallel move data from HDFS to Teradata Aster

Netezza Architecture



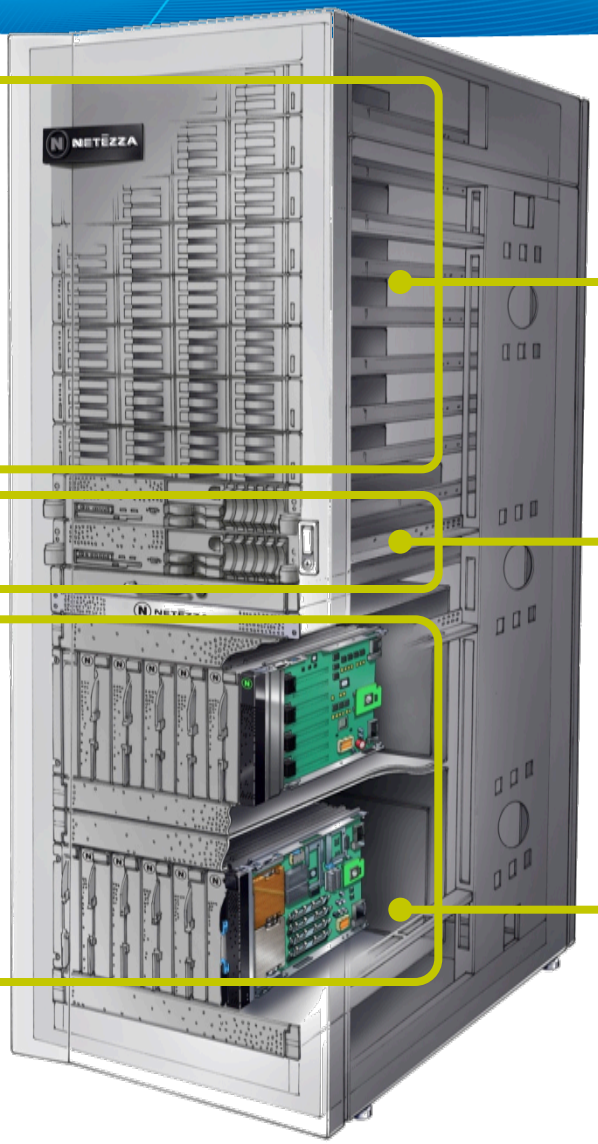
Netezza Appliance

Data Stream Processing



There are 96 Disk Drives, 96 FPGA Cores and 96 CPU Cores per rack

Inside the Netezza TwinFin™ Appliance



Disk Enclosures

Slice of User Data
Swap and Mirror partitions
High speed data streaming

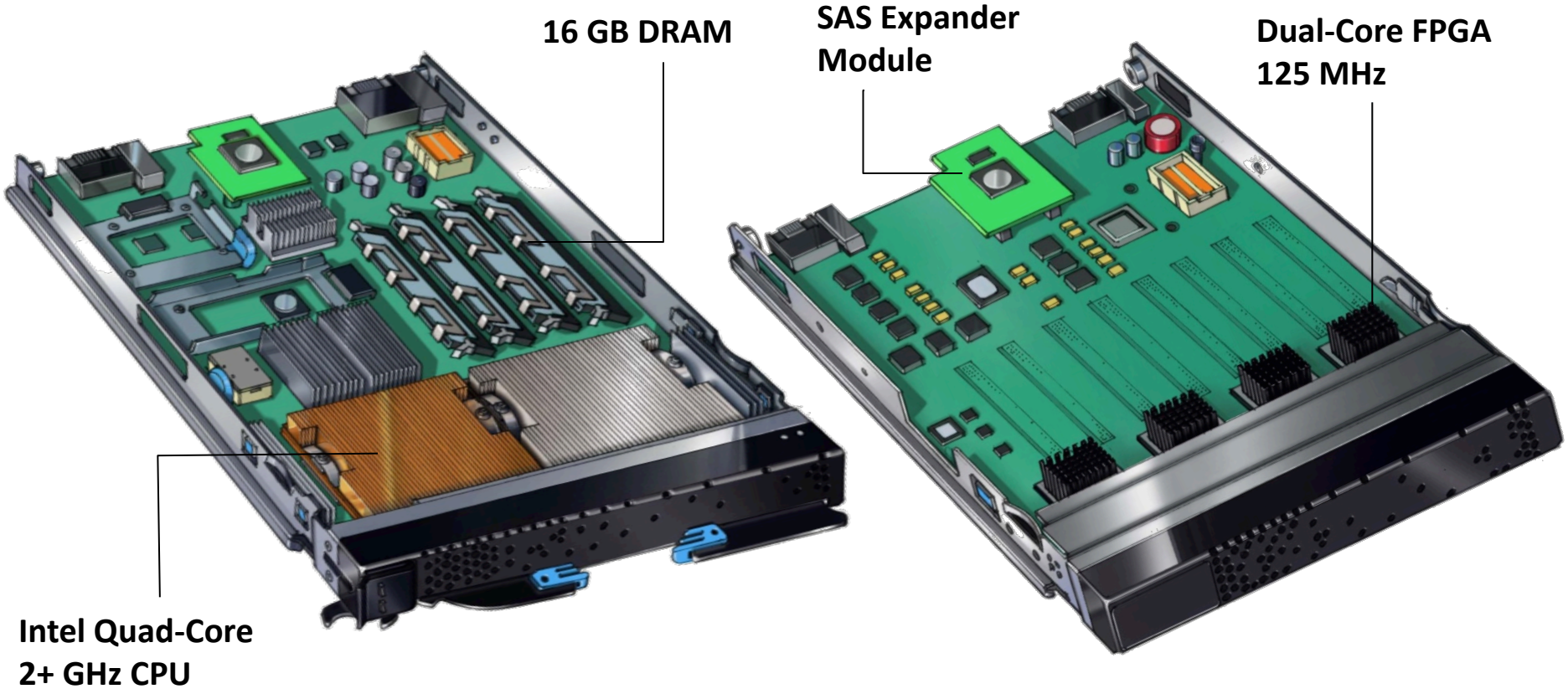
SMP Hosts

SQL Compiler
Query Plan
Optimize
Admin

S-Blades™
(with FPGA-based
Database Accelerator)

Processor &
streaming DB logic
High-performance database engine streaming joins, aggregations, sorts, etc.

S-Blade™ Components



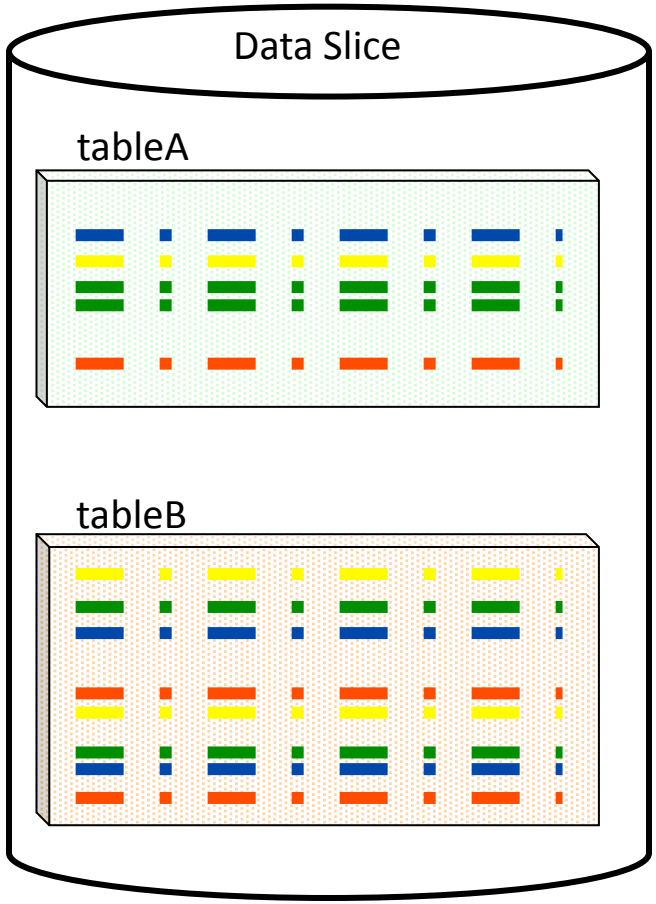
IBM BladeCenter Server

Netezza DB Accelerator

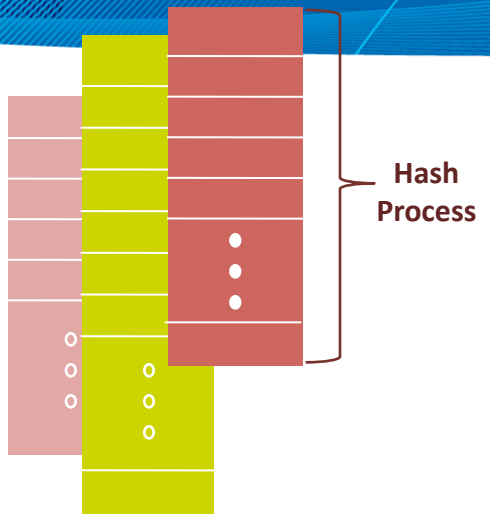
STREAM INTEGRATION

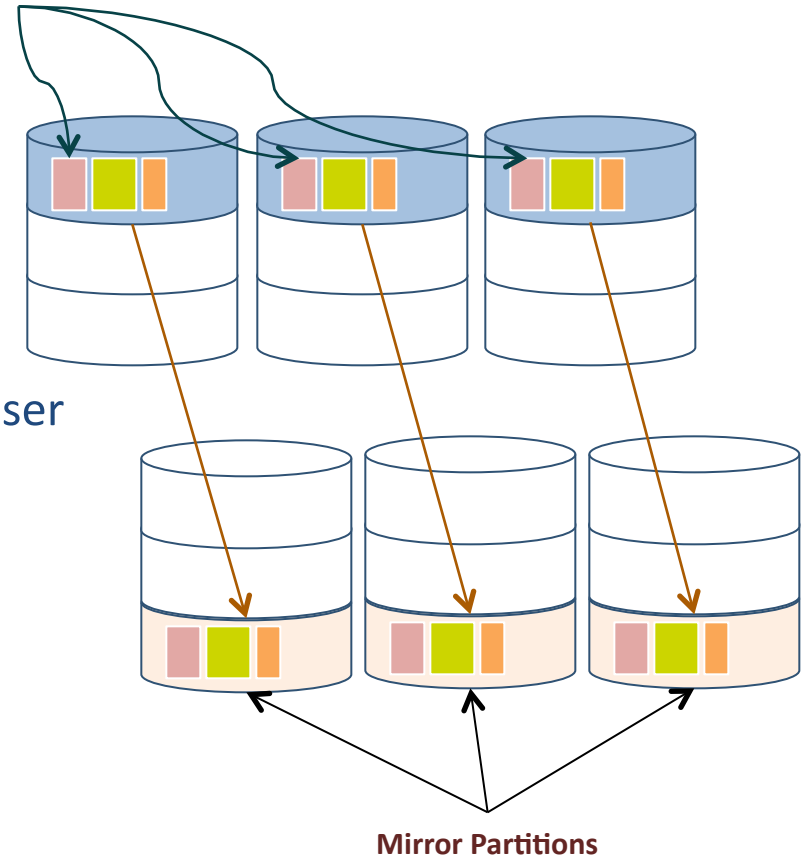
Data slices: How the Netezza Appliance Stores Data on disk

- Each Snippet Processor has a dedicated hard drive
- Data on this drive is called a **data slice**.
- Tables are split across data slices
- Netezza stores data on disk in groups according to row
- Data is compressed according to identical columnar values (columnar compression)



Data Distribution and Mirroring





A hashing algorithm is used to distribute user data to primary partitions on each disk.

Data is also mirrored

- Disk failures transparent to queries and transactions
- Failed drives automatically regenerated
- Bad sectors automatically rewritten or relocated

Data Distribution: Overview

How it works…

Tables in a Netezza Data Warehouse are distributed across SPUs and data slices

The distribution is determined by the **distribution key**, in most cases it consists of one or more columns of the table

When the system creates records, it assigns them to a logical data slice based on their distribution key value.

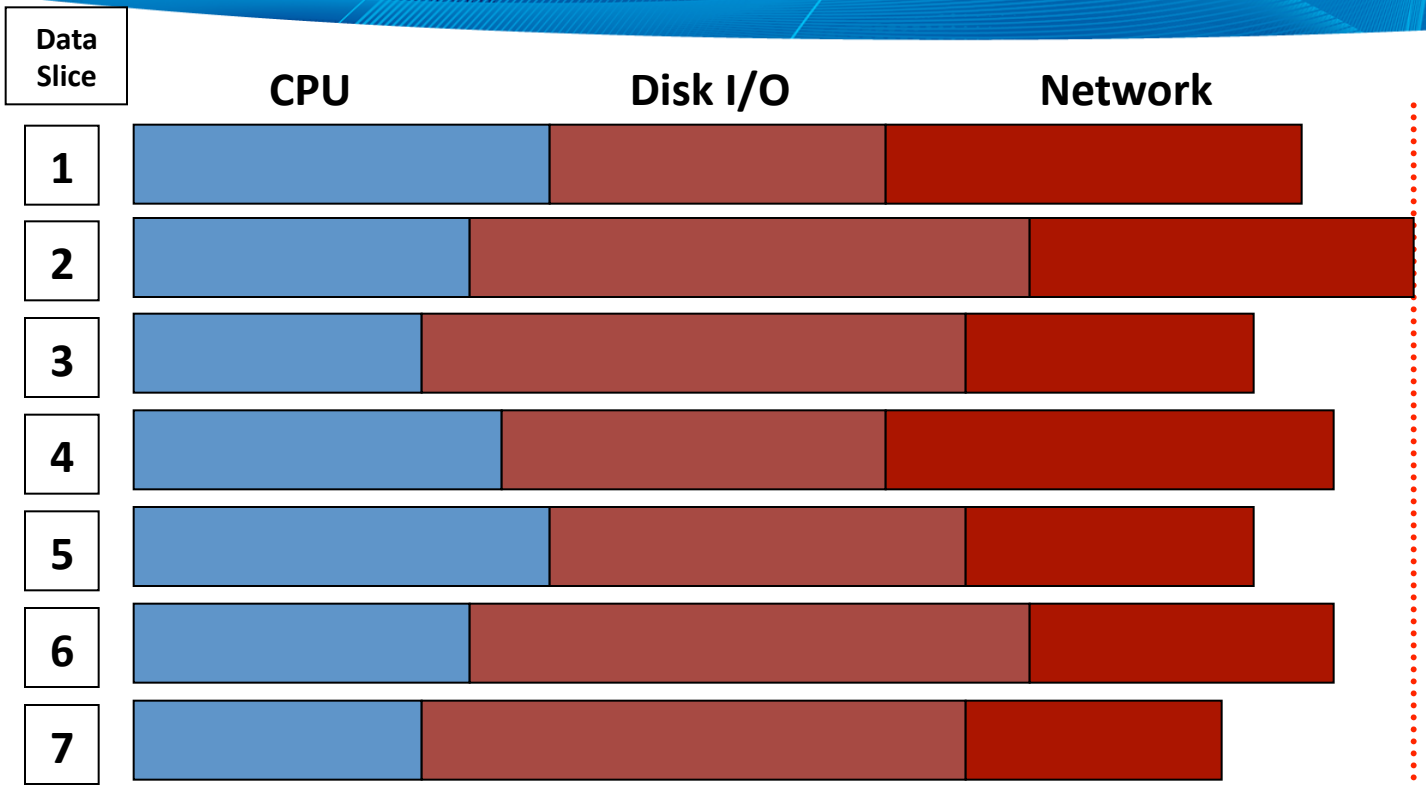
There are two distribution methods, **hash** and **random** (round robin method).

Impact on performance…

The performance of the system is directly linked to uniform distribution of the user data across all of the data slices in the system.

Bad distribution methods and keys can result in uneven distribution of a table across data slices and SPUs (skew) which results in bottlenecks and bad performance.

Distributions and Performance

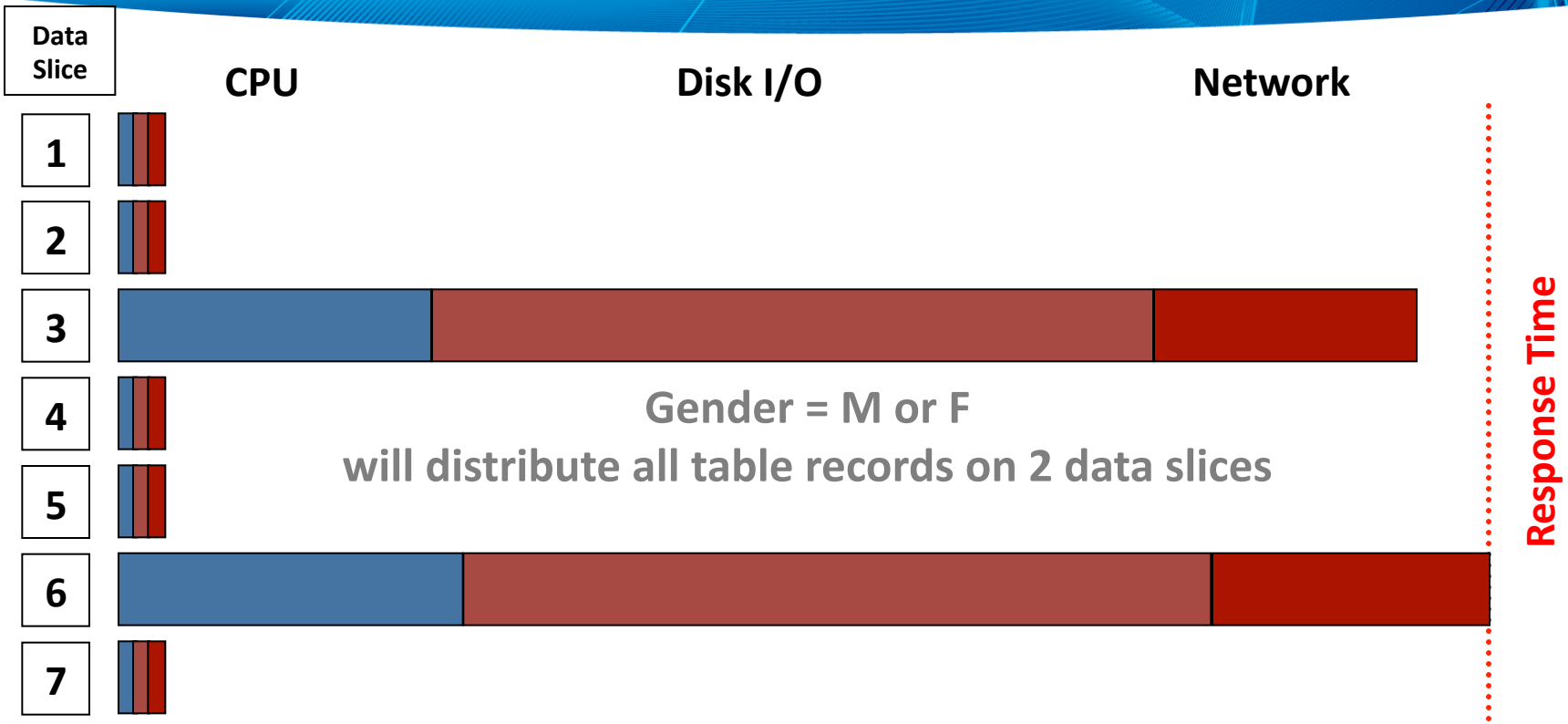


Response time is affected by the completion time for all of the data slices in the MPP array.

A distribution method that distributes data *evenly* across all data slices is the single most important factor that can influence overall performance!

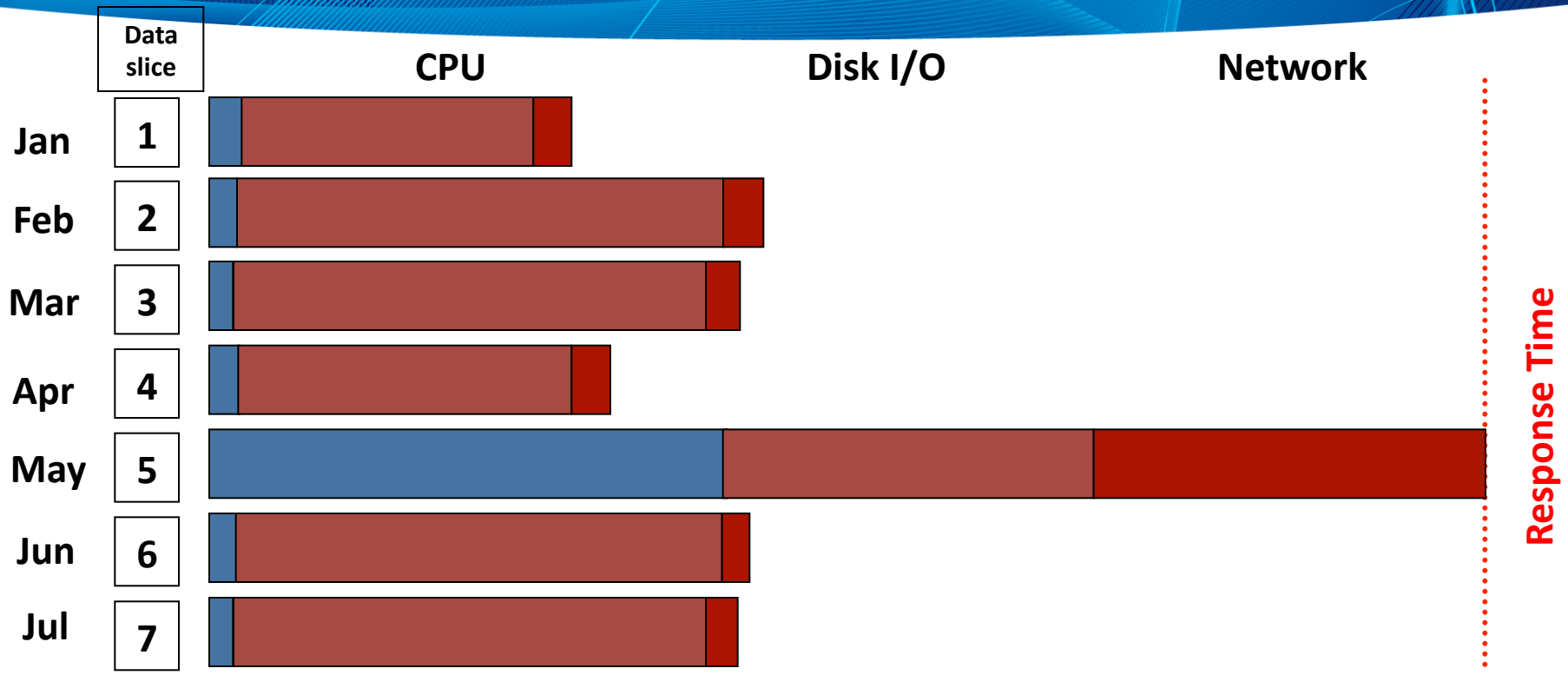
STREAM INTEGRATION

Hash Distributions and Data Skew



Select a distribution key with unique values and high cardinality

Hash Distributions and Processing Skew



Using a DATE column as the distribution key may distribute rows evenly across all data slices. However, most analysis (queries) is performed on a date range. Massively parallel processing won't be achieved when all of the records to be processed for a given date range are located on a single data slice (or a few data slices) .

STREAM INTEGRATION

Extents and Zone Maps

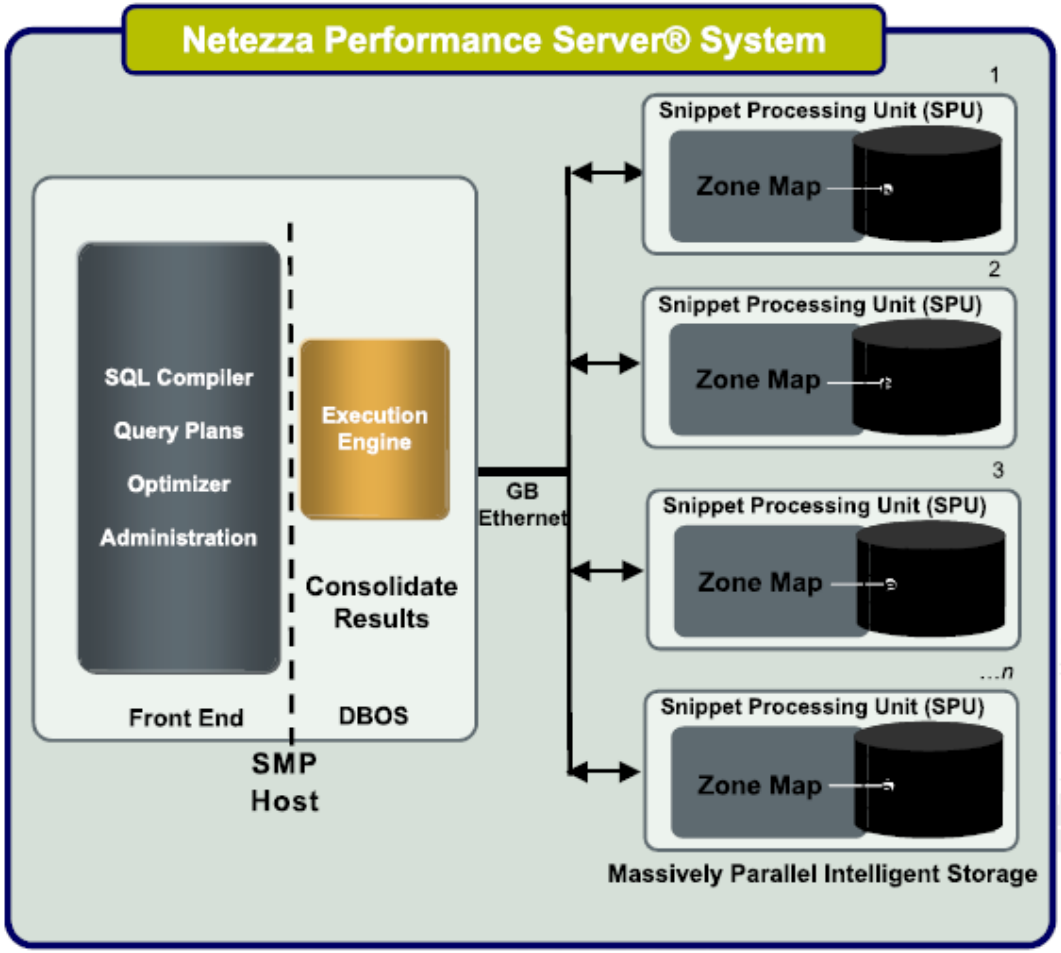
An extent is the smallest unit of disk allocation

3 MB of disk space

A zone map is an internal mapping structure to show the range (min and max) of values within **each extent**

During scans, zone maps are used to reduce IO by skipping extents that didn't qualify the query parameters

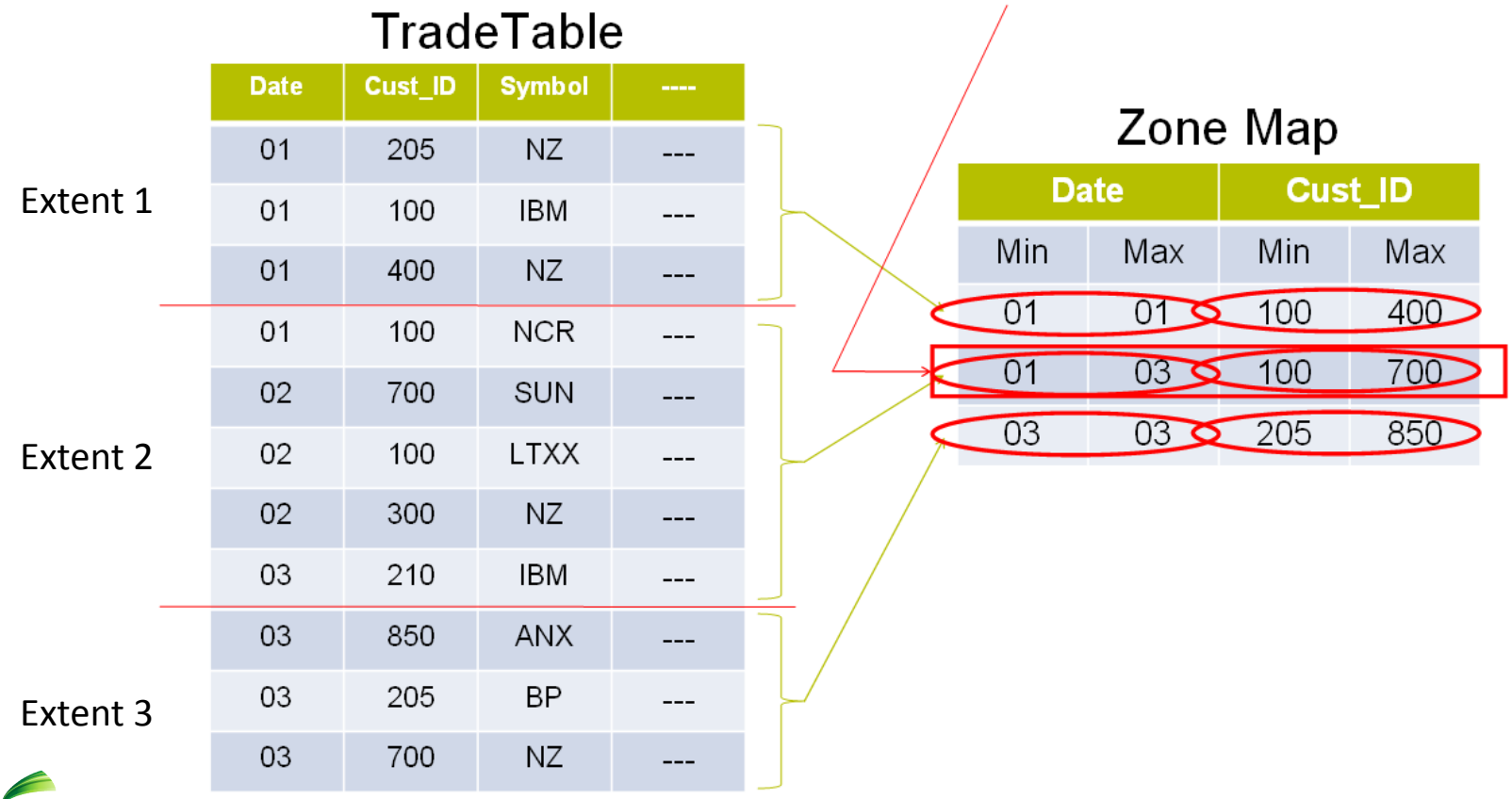
Zone maps are internal to the system thus no administration involved



Zone Maps: Avoid IO

- In the example below only the second extent will be scanned

```
SELECT * FROM TradeTable WHERE date=02
```

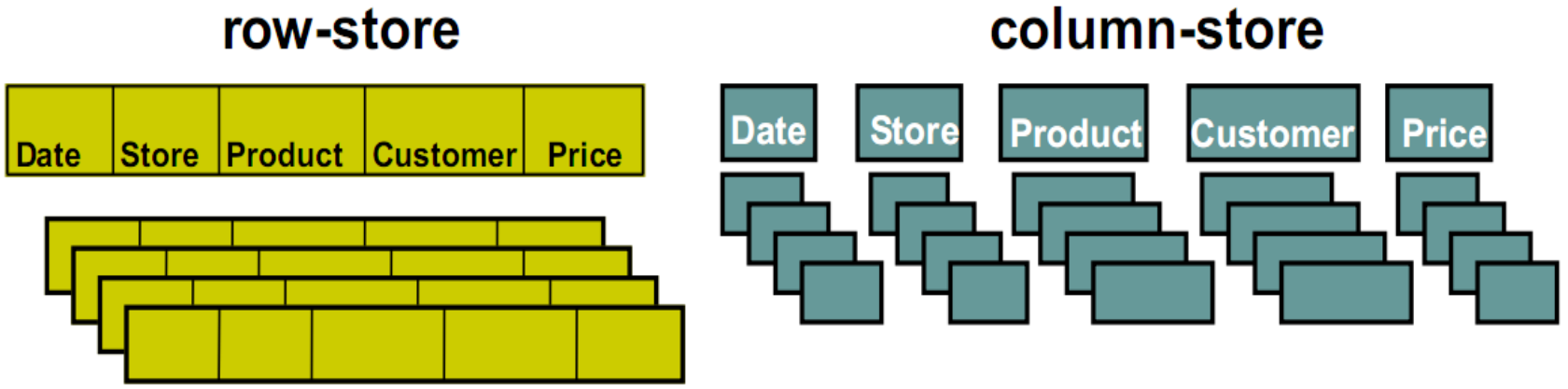


Column Stores

Module 5

STREAM INTEGRATION

Column Stores



In a row store data is stored in the disk tuple by tuple.

In a column store data is stored in the disk column by column

Key Distinctions

Row-store architectures are write-optimized

A single disk write pushes all of the fields of a single record out to disk

High performance writes are achieved

Column-store architectures are read-optimized

Read-optimized for ad-hoc querying of large amounts of data

Data warehouses periodically perform a bulk load followed by a relatively long period of ad-hoc queries

Column stores only read the values of columns required for processing a given query - avoids bringing irrelevant attributes into memory

Suited for warehouse environment where typical queries involve aggregates performed over large numbers of data items

- Column stores have a sizeable performance advantage.

Suitable for read-mostly, read-intensive, large data repositories

STREAM INTEGRATION

Other Distinctions

Row Store	Column Store
(-) Processes all attributes in a row	(+) Processes only required attributes
(-) I/O – reads all attributes in a row	(+) I/O – reads only required attributes
(+) Easy to add/modify a record	(+) Only need to read in relevant data
(-) Might read in unnecessary data	(-) Tuple writes require multiple accesses
(-) Column-store type partitioning can be emulated but performance can't be matched	(+) Can be significantly faster for most analytics
(+) Well-suited for OLTP	(-) Slower for OLTP

STREAM INTEGRATION

Vertica Implementation of C-Store

Organizes data on disk as columns of values from the same attribute, as opposed to storing it as rows of tabular records

Employs high compression of data on disk, as well as a query execution engine that is able to keep data compressed while it is operated on

Allows sufficient space to store multiple copies of the data to ensure fault tolerance and to improve concurrent and ad- hoc query performance

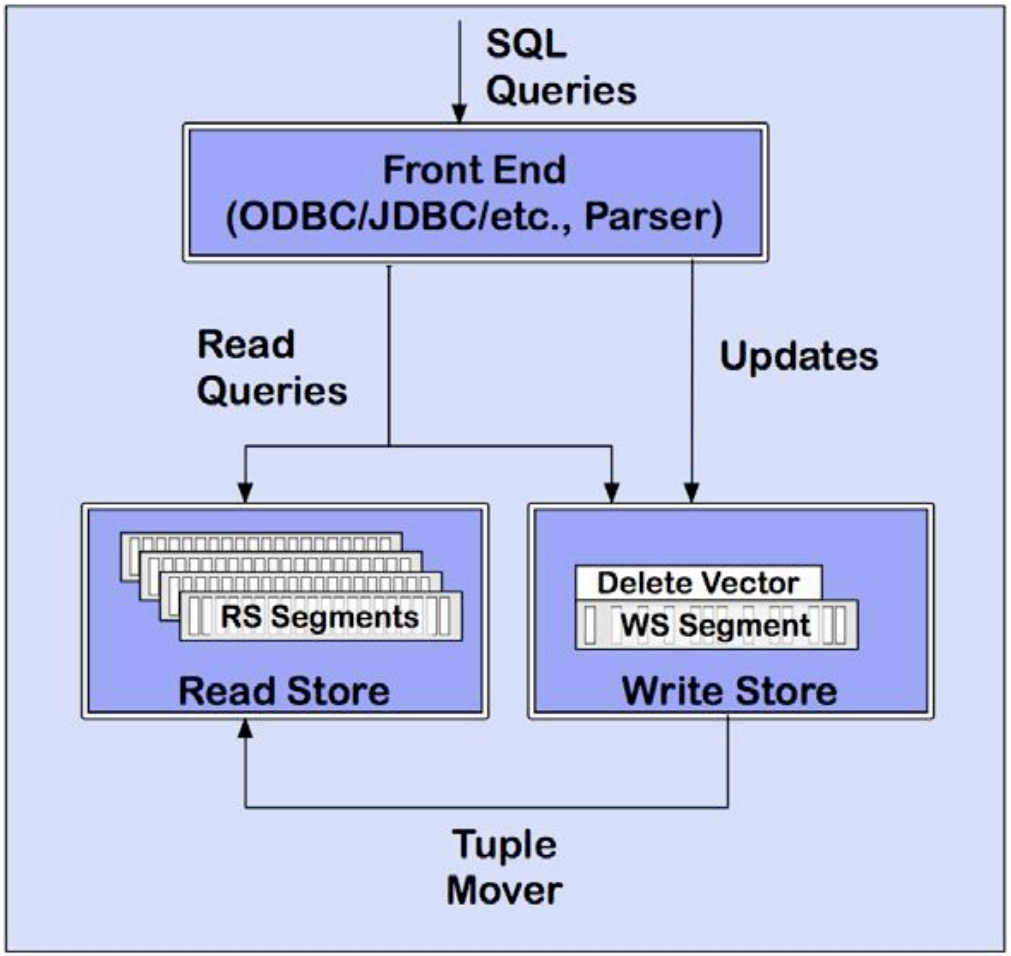
Compression, column-orientation, and table decomposition are transparent to the end- user

Provides a standard SQL interface to users

A hybrid data store, where newly inserted records are added to a write-optimized portion of the database to allow continuous, high-performance insert operations concurrently with querying to enable real-time analytics

Vertica Architecture

- SQL queries are parsed and optimized
- Write-Optimized Store (WOS) fits into memory and supports insert update operations
- Read-Optimized Store (ROS) contains the bulk of the data and is sorted and compressed – optimized for read and query
- Tuple Mover moves data out of the WOS and into ROS



Logical to Physical

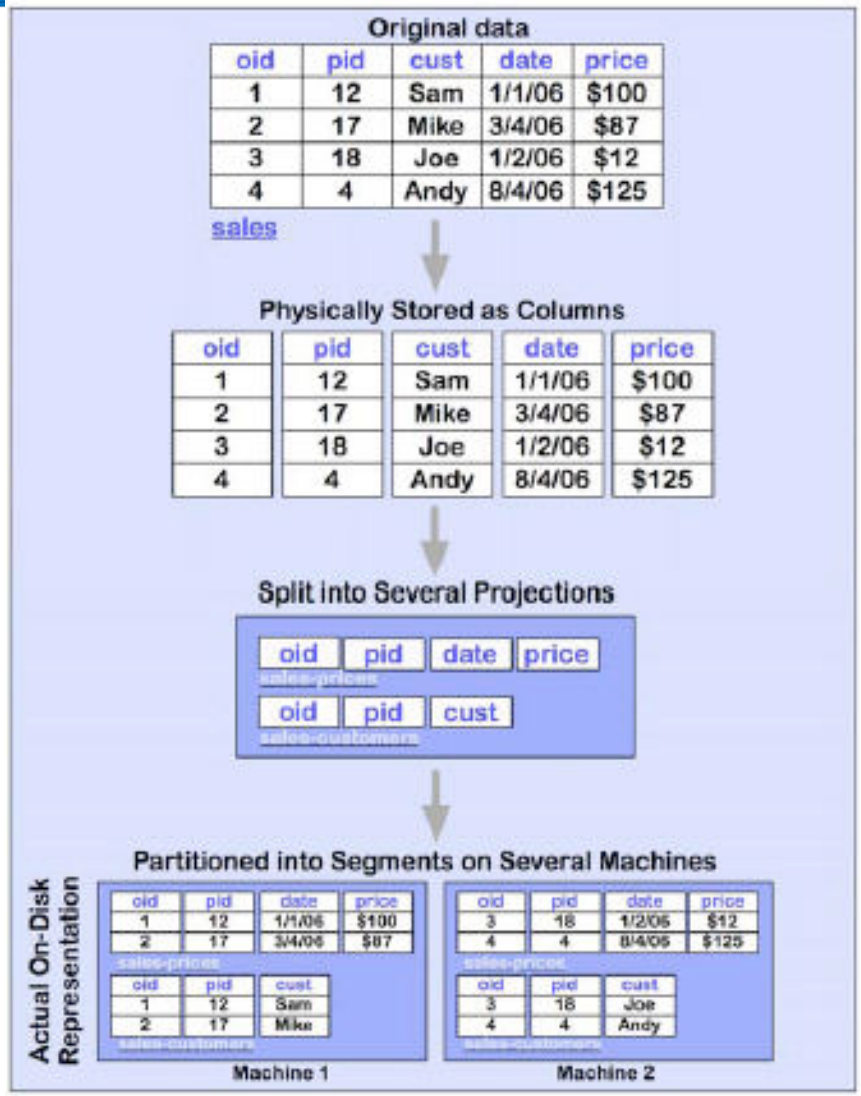
Each column may be stored in one or more ***Projections*** that represent partially redundant copies of the data in the database

Example - Sales table stored as two projections

> *sales-prices with* the schema (order-id, product-id, sales- date, sale-price)
>
> *sales- customers* with the schema (order-id, product-id, customer-id)

Each has a sort order that specifies how the data in the projection is arranged on disk

Storing several overlapping projections of a table in different sort orders allows flexibility to choose most efficient means to answer many different types of queries



Storage Architecture Enables Parallelization

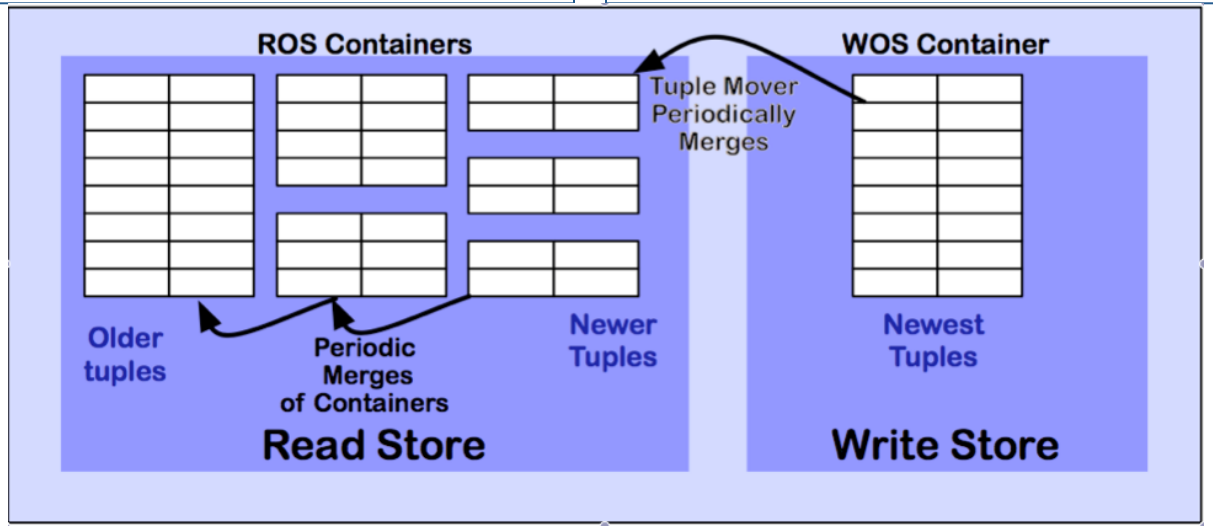
On a single machine, the ROS is horizontally partitioned into several *Storage Containers*

The Tuple Mover inserts new data into the ROS without having to merge together and rewrite all old containers

To keep the total number of containers small, from time to time, the tuple mover merges these ROS containers together in the background

Each projection is horizontally partitioned into *segments,* each of which is stored on a different machine in the cluster based on the value of one attribute in the table

This makes it possible to parallelize queries, allowing many applications to scale linearly with the number of nodes in the database



Performance Features

Column-Orientation

Resolves disk-bound queries by reading required columns only

Aggressive Compression

Entropy on columnar data as opposed to row data allows for significant compression

Read-Optimized Storage

Bulk data is stored in ROS which is optimized for read operations

Data is dense packed – no empty space is left at the end of disk pages

Multiple Sort Orders

Replicas of data in different sort orders

Enables high availability and recovery

Parallel Shared Nothing Architecture

Designed to run on off-the-shelf, Linux-based servers

STREAM INTEGRATION

Breakdown of Column-Store Advantages

Block processing improves the performance by a factor of 5% to 50%

Compression improves the performance by almost a factor of two on average

Late materialization improves performance by almost a factor of three

Invisible join improves the performance by 50-75%

Hybrid Approach

MODULE 6

STREAM INTEGRATION

Hybrid Approaches

MapReduce is increasingly being used to analyze structured data because of the tendency to unify data management platforms

Significant amount of research and commercial activity has focused on integrating MapReduce and relational database technology.

There are two approaches to this problem:

1. Starting with a parallel database system and adding some MapReduce features

2. Starting with MapReduce and adding database system technology.

Shared Nothing Architectures

Most analytical database vendors deploy their DBMS on a shared-nothing architecture

This architecture is widely believed to scale the best

These architectures are optimized for common data analysis workloads that tend to consist of:

- Many large scan operations
- Multidimensional aggregations
- Star schema joins

These operations are fairly easy to parallelize across nodes in a shared-nothing network

Scaling MPP vs MapReduce

MPP

MPP databases scale really well into the tens of nodes

- Near linear scalability is not uncommon

Few known parallel databases deployments consist of more than one hundred nodes

Problems when scaling into hundreds of nodes

- Failures become increasingly common as one adds more nodes to a system
  - Designed with the assumption that failures are a rare event
- Design is based on a homogeneous array of machines
  - Nearly impossible to achieve pure homogeneity at scale
- Parallel databases have not been tested at larger scales
  - Unforeseen engineering hurdles await

MapReduce

MapReduce-based systems were designed from the beginning to scale to thousands of nodes in a shared-nothing architecture

Success has been proven in Google's internal operations

Can be used to process structured data at tremendous scale

- Hadoop is being used to manage Facebook's 2.5 petabyte data warehouse

STREAM INTEGRATION

MapReduce Limitations

Lacks many of the features that have proven invaluable for structured data analysis workloads

- Data schema
- Separation of schema and application
- High-level access language
- Indexing
- Updates
- Integrity constraints and referential integrity
- Views
- Compatibility with SQL-based BI tools

Hadoop has a suboptimal storage layer

- DFS lacks many of the optimization techniques implemented in RDBMS systems including careful layout of data on disk, indexing, sorting, shared I/O, buffer management, compression, and query optimization

STREAM INTEGRATION

Benchmark Performance

MR Grep tasks

For the task, each system must scan through a data set of 100B records looking for a three-character pattern.

Web log task

conventional SQL aggregation with a GROUP BY clause on a table of user visits in a Web server log

Join task

fairly complex join operation over two tables requiring an additional aggregation and filtering operation

	Grep	Web Log	Join
Hadoop	284s	1,146s	1,158s
DBMS-X	194s	740s	32s
Vertica	108x	268s	55s
Hadoop /DBMS-X	1.5x	1.6x	36.3x
Hadoop /Vertica	2.6x	4.3x	21.0x

STREAM INTEGRATION

Architectural Differences

Repetitive record parsing

Compression

Pipelining

Scheduling

Column-oriented storage

STREAM INTEGRATION

Solution

Combine the scalability advantages of MapReduce with the performance and efficiency advantages of parallel databases

Design a hybrid system that is well suited for the an analytical DBMS market and can handle the future demands of data intensive applications.

Use MapReduce as the communication layer above multiple nodes running single-node DBMS instances.

- Express queries are in SQL and translated into MapReduce
- Push as much work as possible into the higher performing single node databases

Combine the job scheduler, task coordination, and parallelization layer of Hadoop, with the storage layer of a DBMS to retain the best features of both systems.

- Achieve the performance on structured data analysis comparable with parallel database systems,
- Maintain Hadoop's fault tolerance, scalability, ability to handle heterogeneous node performance, and query interface flexibility.

STREAM INTEGRATION

Desired Properties

Performance.

A factor of ten can make a big difference in the amount, quality, and depth of analysis a system can do.

High performance systems can delay a costly hardware upgrade, or avoid buying additional compute nodes as an application continues to scale.

Public cloud computing costs scale linearly with the requisite storage, network bandwidth, and compute power.

Fault Tolerance.

A fault tolerant analytical DBMS is simply one that does not have to restart a query if one of the nodes fails

Cheap, unreliable commodity hardware used to build a shared-nothing cluster of machines is more prone to failures

Google reports an average of 1.2 failures per analysis job

Ability to run in a heterogeneous environment.

It is nearly impossible to get homogeneous performance across hundreds or thousands of compute nodes

- Part failures that do not cause complete node failure, result in degraded hardware performance
- Individual node disk fragmentation and software configuration errors
- Concurrent queries or processes

Flexible query interface.

Business Intelligence tools typically connect to databases using ODBC or JDBC, so databases that want to work with these tools must accept SQL queries through these interfaces.

Revisiting MPP

Top Marks

Support standard relational tables and SQL, and implement many of the performance enhancing techniques developed by the research community over the past few decades

Indexing, compression, materialized views, result caching, and I/O sharing.

Can achieve especially high performance when administered by a highly skilled DBA

Automating of DBA tasks has been bundled into appliances

Flexible query interface

Implementation of SQL and ODBC

Many parallel databases allow UDFs

Needs Improvement

Fault tolerance and ability to operate in a heterogeneous environments

Assumptions that failures are rare events and "large" clusters mean dozens of nodes have resulted in engineering decisions that make it difficult to achieve these properties.

Often there is a clear tradeoff between fault tolerance and performance

MPP databases tend to choose the performance extreme of these tradeoffs.

- Frequent check-pointing of completed sub-tasks increase the fault tolerance of long-running read queries, yet this check-pointing reduces performance
- Pipelining intermediate results between query operators can improve performance, but can result in a large amount of work being lost upon a failure.

STREAM INTEGRATION

Revisiting MapReduce

Top Marks

Meets the fault tolerance and ability to operate in heterogeneous environment properties.

- Achieves fault tolerance by detecting and reassigning Map tasks of failed nodes to other nodes in the cluster
- Achieves the ability to operate in a heterogeneous environment via redundant task execution.

Has a flexible query interface

- Map and Reduce functions are jus arbitrary computations written in a general-purpose language
- MapReduce-based systems do not generally accept declarative SQL

Needs Improvement

Performance is the biggest issue

- No prior modeling and loading of data before processing
- Traditional analytics are based on repeated queries, and are poorly suited for the one-time query processing model of MapReduce

STREAM INTEGRATION

Are the MapReduce Semantics Unique?

The filtering and transformation of individual data items (tuples in tables) can be executed by a modern parallel DBMS using SQL

For Map operations not easily expressed in SQL, many DBMSs support user-defined functions

- UDF extensibility provides the equivalent functionality of a Map operation
- SQL aggregates augmented with UDFs and user-defined aggregates provide DBMS users the same MR-style reduce functionality.

The reshuffle that occurs between the Map and Reduce tasks in MR is equivalent to a GROUP BY operation in SQL

Given this, parallel DBMSs provide the same computing model as MR, with the added benefit of using a declarative language (SQL)

Hybrid

Ideally, the fault tolerance and ability to operate in heterogeneous environment properties of MapReduce could be combined with the performance of parallel databases systems

STREAM INTEGRATION

HADAPT

Start-up company currently commercializing the Yale University research project called HadoopDB led by Dr. Daniel Abadi

Combines the benefits of Apache Hadoop and relational DBMS technology into a single system for applications that rely on multi-structured data analytics

Designed for the cloud, and is optimized for virtualized environments

Architected to leverage clusters of industry standard (commodity) machines

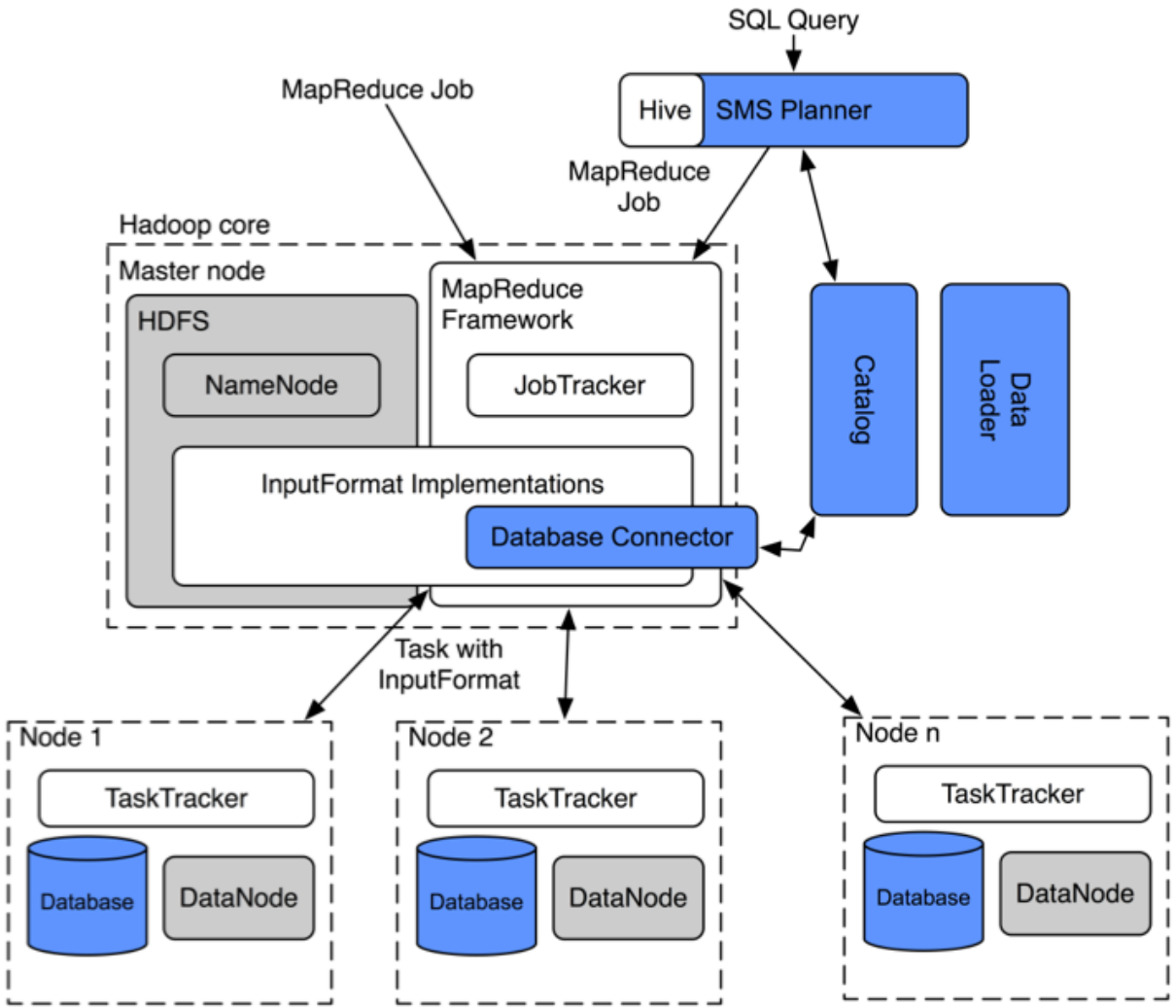
Provides the full power of MapReduce as well as SQL support and the ability to work with data within a single platform

Based on findings from the HadoopDB project it aims to achieve:

- Performance and efficiency of MPP databases
- Scalability, fault tolerance, and flexibility of MapReduce-based systems

STREAM INTEGRATION

Hadapt Architecture



Main Components

Flexible Query Interface

- Data can be queried using both SQL and MapReduce
- SQL can be embedded within MapReduce or vice versa
- JDBC/ODBC drivers for connectivity with customer-facing BI tools

Query Planner

- Queries are analyzed to consider data partitioning and distribution, indexes, and statistics to determine a query plan
- Split query execution ensures optimal use of the DBMS layer before pushing operations into Hadoop

Adaptive Query Execution

- The time to complete the query will be approximately equal to the time it takes the slowest compute node to complete its assigned task
- This dynamic is especially problematic in a cloud environment
- Query plans are adjusted dynamically based on cloud worker node performance

Data Loader

- Data is loaded using all machines in parallel
- Data is partitioned into small chunks and replicated across the cluster
- Optimizes query performance and fault tolerance

Data manager

- Stores metadata on the schema, data, and chunk distribution
- Handles data replication, backups, recovery, and rebalancing of chunks across the cluster

Hybrid Storage Engine

- A DBMS engine is stored on each node in addition to a standard distributed file system (HDFS)
- DBMS layer is optimized for structured data and HDFS handles unstructured data

Optimization Techniques

Guiding Heuristics

- Database systems can process data at a faster rate than Hadoop
- Each MapReduce job typically involves many I/O operations and network transfers

Referential Partitioning

- Distributed joins are expensive in Hadoop relative to DBMS
- Preference to computing joins within a single database engine
- Aggressive hash-partitioning considers many foreign/primary key references at some cost to loading performance

Split MR/DB Joins

- Joins are performed by MapReduce for tables that are not co-partitioned
- Split execution environments enable the implementation of a variety of joins in the Map phase

Post-join Aggregation

- hash-based partial aggregation is done at the end of each Reduce task
- The grouping attribute extracted from each result of the Reduce task is used to probe a hash table in order to update the appropriate running aggregation
- Significant I/O savings

Pre-join Aggregation

- Used for joins that cannot be pushed into the DBMS
- Partial aggregation is pushed to the DBMS
- The performance of the subsequent join in Hadoop is improved due to savings in I/O and network traffic

In-Memory Databases (IMDB)

Module 7

In-Memory Database (IMDB)

Primarily relies on main memory for computer data storage.

Faster than disk-optimized databases since the internal optimization algorithms are simpler and execute fewer CPU instructions

Accessing data in memory reduces the I/O reading activity when querying the data which provides faster and more predictable performance than disk

These devices lose all stored information when the device loses power or is reset

- In this case, IMDBs can be said to lack support for the durability portion of the ACID

STREAM INTEGRATION

Storage Trends

HDD

Most database systems were designed to operate from HDD

The electromechanical elements to position a read/write unit over the physical media present challenges

- The time to place the read/write head in the right place has not improved at the same rate as disk transfer and bandwidth improvements
- The sequential bandwidth of reading from disk has experienced exponential growth, while the ability to randomly access small amounts of data on disk has not

Dynamic RAM (DRAM)

It has gotten cheaper and its capacity has increased significantly

The bandwidth to and from DRAM has been exponentially increased but latency, the ability to randomly access a single byte, hasn't been able to keep pace.

STREAM INTEGRATION

NewSQL

Combines the speed and scale of NoSQL with the proven capabilities of OldSQL (Traditional SQL)

Drivers

The Internet: The internet allows end users to interact directly with databases, dramatically increasing the velocity and volume of database transactions

Smart Devices: Transactions now originate from multiple sources, including tablets, laptop computers and smart phones

Sensor-generated Data: Everything worth anything will soon be sensor tagged

OldSQL and NoSQL

OldSQL

Focuses processing time on four overhead components (buffer management, locking, latching, logging)

can't meet the throughput requirements of NewOLTP.

NoSQL

Neglects both SQL and ACID to improve scalability and performance.

Replaces SQL with low-level programming interfaces

May be appropriate for non-transactional systems and single-record, commutative transactions,

Is not a good fit for new OLTP, which requires throughput, scale *and* ACID.

STREAM INTEGRATION

VoltDB

Designed by several well-known database system researchers

- Michael Stonebraker (who was involved in Ingres and PostgreSQL),
- Sam Madden
- Daniel Abadi (Vertica and Hadapt)

In-memory storage to maximize throughput, avoiding costly disk accesses

A NewSQL relational database that supports SQL access from within pre-compiled Java stored procedures

Serializes all data access, thus avoiding overheads of traditional databases such as locking, latching and buffer management

Performance and scale through horizontal partitioning

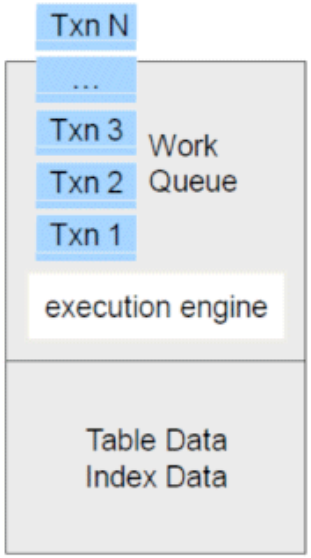
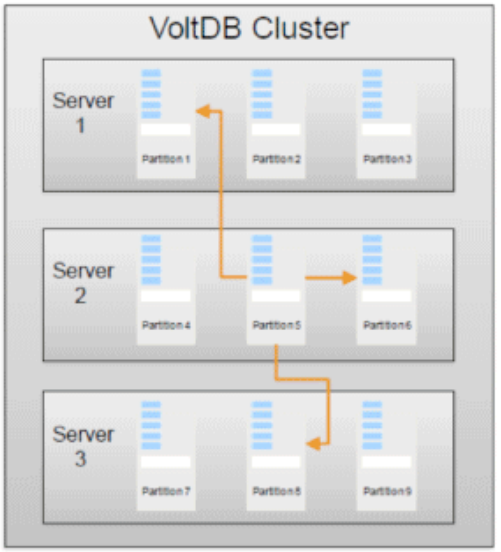
High availability through synchronous, multi-master replication (in VoltDB parlance, “K-safety”)

Durability through an innovative combination of database snapshots and command logs that store recoverable state information on persistent devices (i.e., spinning disks and/or SSDs)

Key Characteristics

High-throughput, low-latency SQL database operations	In-memory operation, stored procedure interface, asynchronous/synchronous stored procedure execution and serial transaction processing
VoltDB scaling architecture	Partitioning, reference table replication, cluster transparency
High Availability (HA)	K-Safety, network fault detection, live node rejoin, snapshots, command logging
Durability	Snapshots and command logging for database crash recovery
Database Replication	Cluster-wide replication for disaster recovery, hot stand-by and workload optimization
Realtime analytics	Materialized views and distributed read optimizations
Provisioning, management & monitoring	VoltDB Enterprise Manager and REST management API
VoltDB Studio for rapid application development	Companion tool for popular IDEs that supports rapid iteration of dev/test/ tuning cycle
Data Integration	Streaming export (including buffering and buffer overflow), and Hadoop integration
Developing applications with VoltDB	VoltDB- and community-created client libraries, JDBC and sample reference implementations

VoltDB Architecture

Single Partition Transactions	Multiple Partition Transactions
	
Each VoltDB partition operates autonomously, freeing the rest of the cluster to handle other requests in parallel.	When a procedure requires data from multiple partitions, one node acts as a coordinator and distributes the necessary work to the other nodes, collects the results and completes the task. Transactional integrity is maintained and the architecture of multiple parallel partitions ensures throughput is kept at a maximum.

HBASE

MODULE 8

NoSQL

A general term meaning that the database is not an RDBMS which supports SQL

It may not give full ACID guarantees

It has a distributed, fault-tolerant architecture

Useful when working with a huge quantity of data and the data's nature does not require a relational model for the data structure

While the data could be structured its primary focus is the ability to store and retrieve great quantities of data, and not the relationships between the elements

Particularly useful for statistical or real-time analyses for growing list of elements

NoSQL vs SQL

HBase

What Is it?

A NoSQL database

More of a "Data Store" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc.

When to use HBase

Is not suitable for every problem

Not suitable for use with fewer than billions of rows

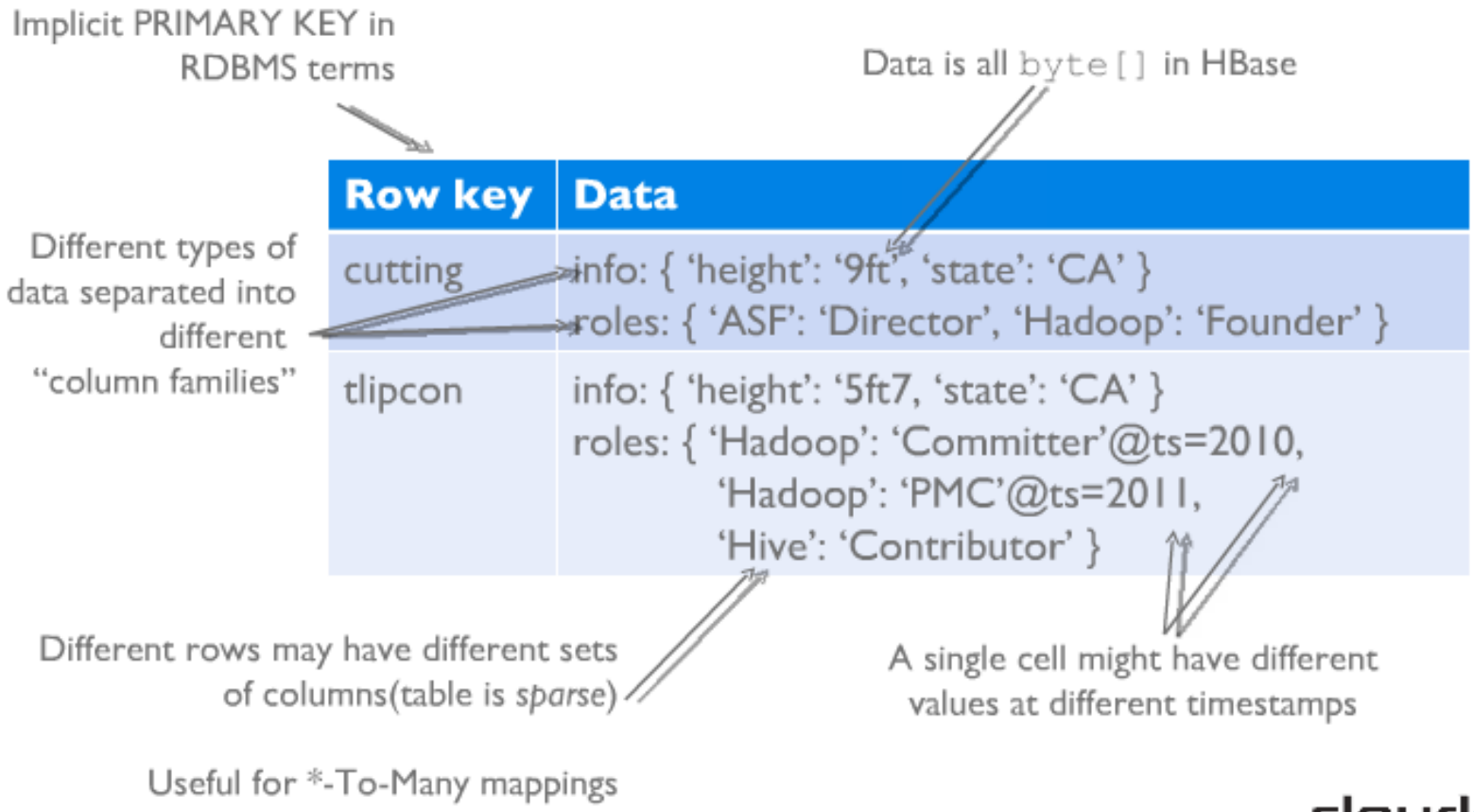
When all the features of a DBMS are not required

Can't port an application written for RDBMS by simply changing the JDBC driver

Must have sufficient number of nodes in your cluster to achieve benefit

STREAM INTEGRATION

Sorted Map Datastore



cloudera

STREAM INTEGRATION

`info` Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

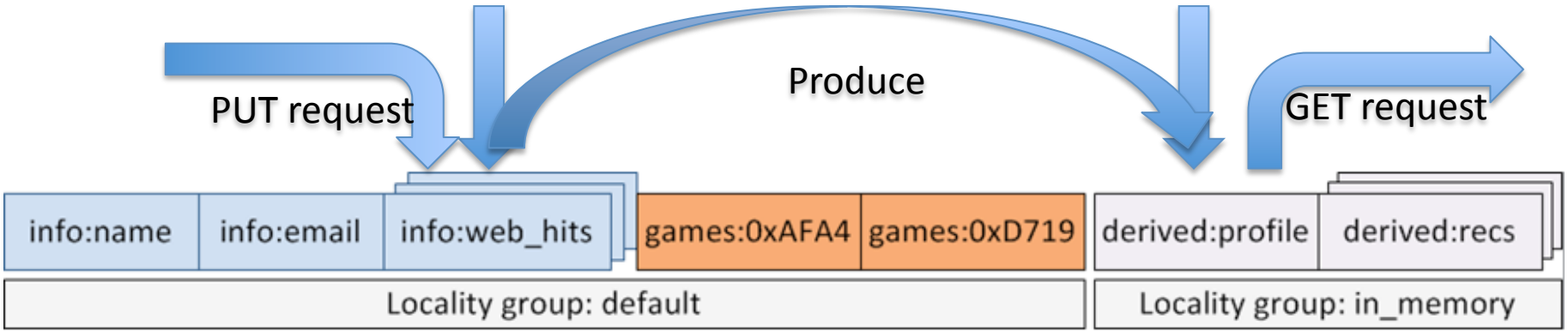
cloudera

STREAM INTEGRATION

WibiData

- Designed for investigative and operational analysis of consumer internet data
- Core technology is a data management and analytic execution layer built on top of HBase and Hadoop
- REST API's for interactive access
- Import/export tools including JDBC access
- Management tools
- Analytic libraries for data mining, predictive analytics, machine learning, etc.

WibiData Operators



Each row is represents a user

Individual columns can be multi-valued

Operators allow you to think about your data in a row-oriented data model.

Producers are computation functions that allow you to update a row

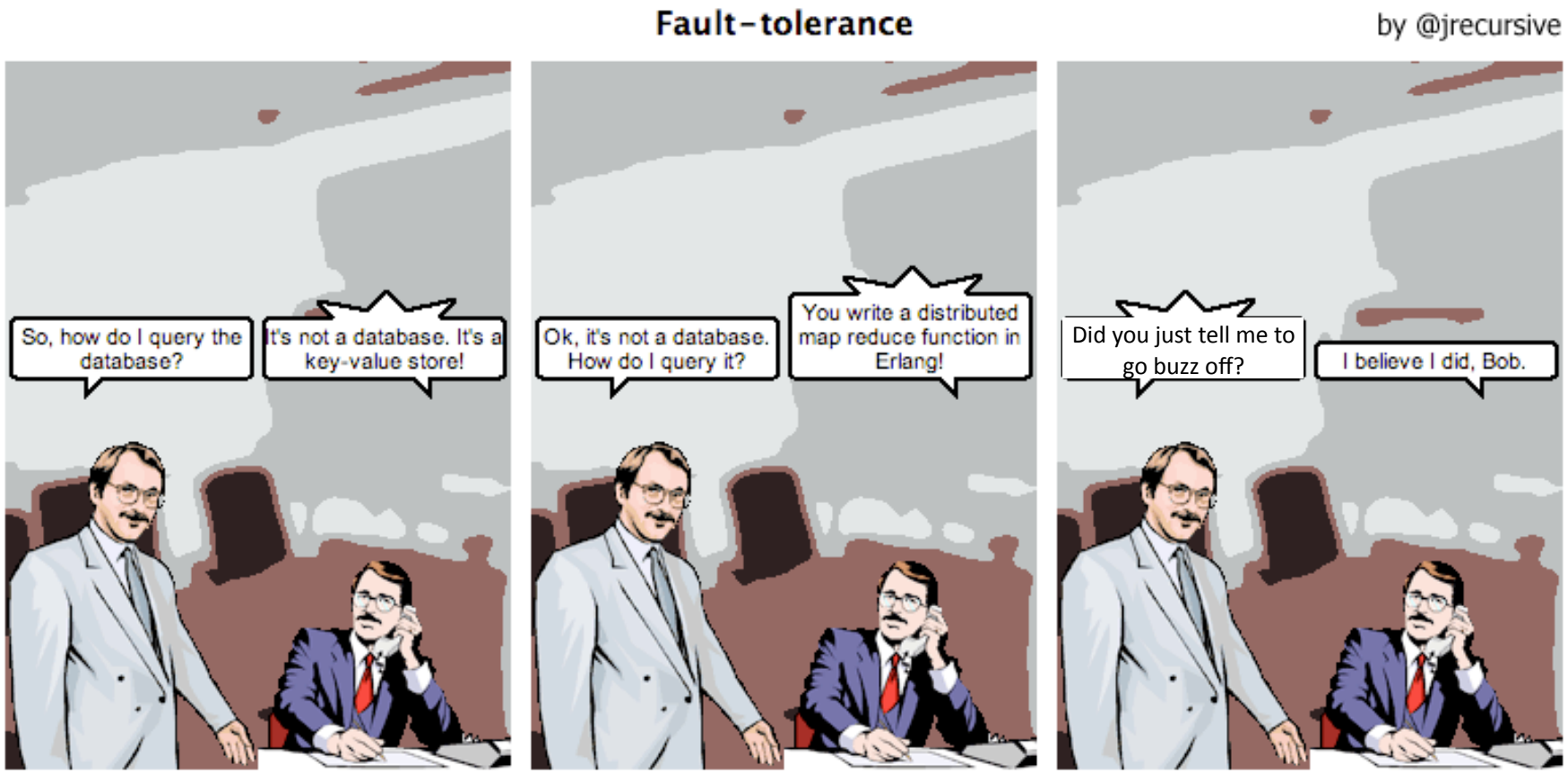
Gatherers extract a projection for use in a more conventional MapReduce pipeline. The result can be applied on a row-by-row basis with a producer

Producers are not tied to MapReduce and can be executed in batch or interactively

WibiData – Key Characteristics

- The DML (Data Manipulation Language) is a lot less rich than SQL
  - does have ways to simulate joins, foreign keys, etc. but does not enforce referential integrity or foreign key constraints
- Schemas are much more dynamic than SQL schemas
  - different rows in the same table have different associated columns (no null values required)
  - schemas can change over the life of a column
- Schemas can have nested or recursive data structures, such as array-valued cells
  - where you'd have a single value in a relational table, you might have the equivalent of a whole relational table

It must be time for a break...



HBase vs HDFS

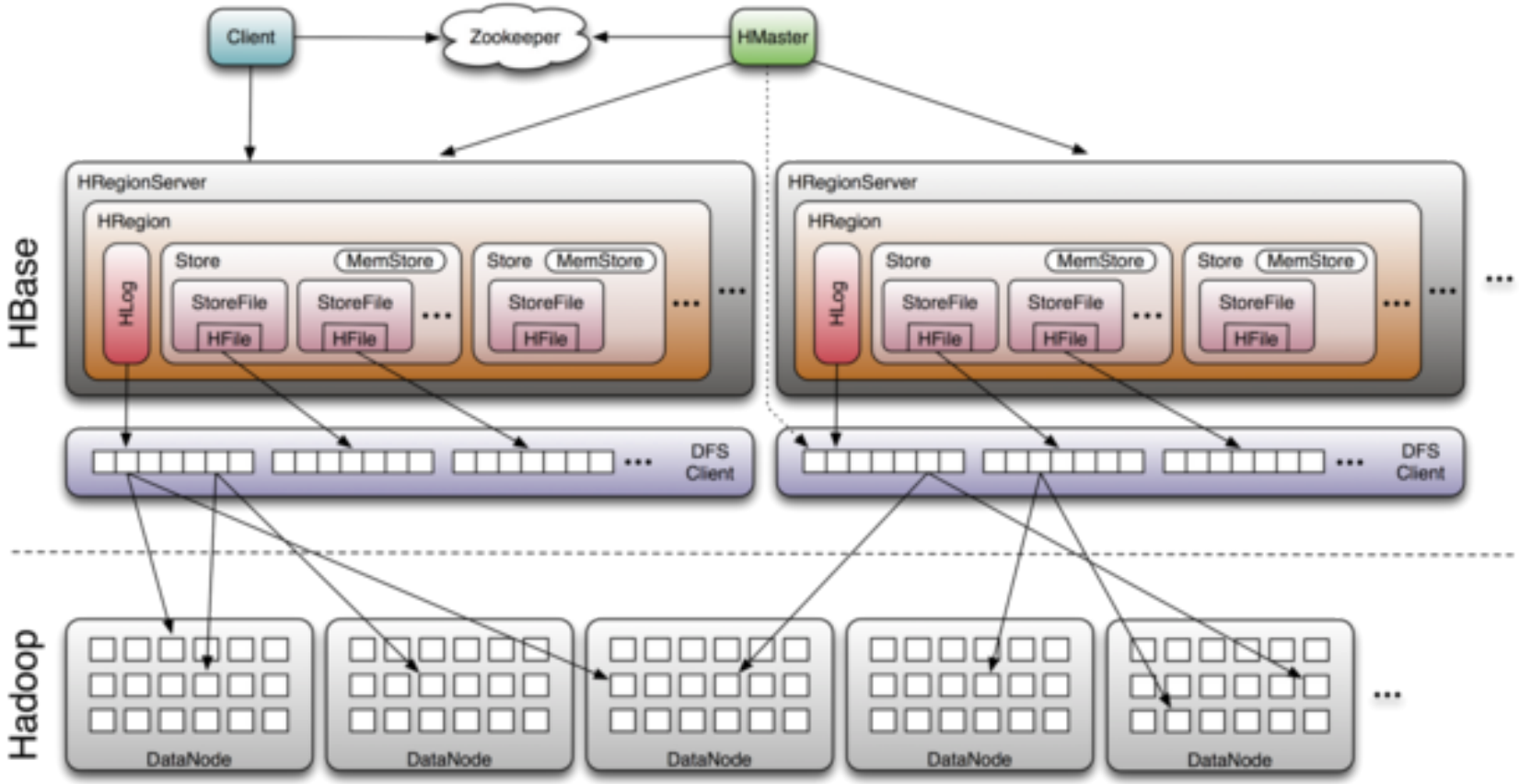
	HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured Storage	Requires complex programming	Sparse column-family data model
Max data size	30+ PB	Up to 1 PB

If you do not require random access, stick to HDFS

HBase vs RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query Language	SQL	Get/put/scan/etc
Security	Authentication/ Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	Up to 1 PB
Read/write throughput limits	1000s queries/second	Millions of queries / second

HBase Architecture



NoSQL BigTable - MongoDB

Document-oriented

Documents (objects) map nicely to programming language data types

Embedded documents and arrays reduce need for joins

Dynamically-typed (schemaless) for easy schema evolution

No joins and no multi-document transactions for high performance and easy scalability

High performance

No joins and embedding makes reads and writes fast

Indexes including indexing of keys from embedded documents and arrays

Optional streaming writes (no acknowledgements)

High availability

Replicated servers with automatic master failover

Easy scalability

Automatic sharding (auto-partitioning of data across servers)

- Reads and writes are distributed over shards
- No joins or multi-document transactions make distributed queries easy and fast

Eventually-consistent reads can be distributed over replicated servers

Rich query language

STREAM INTEGRATION

MongoDB Data Model

Mongo data model

A Mongo system holds a set of databases

A **database** holds a set of collections

A **collection** holds a set of documents

A **document** is a set of fields

A **field** is a key-value pair

A **key** is a name (string)

A **value** is a

- Basic type like string, integer, float, timestamp, binary, etc.,
- A document, or
- An array of values

STREAM INTEGRATION

NoSQL 'as-a-service' - Amazon SimpleDB

Working in a cloud

A key concept with Amazon and many of its Web services is cloud computing

takes advantage of Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2).

The basic premise

A Web service for running queries on structured data in real time

APIs allow developers to easily create SimpleDB-based data and manipulate this data

The concepts

Removes some of the constraints associated with relational database systems by organizing data into domains

Domains are comprised of items, and items are described by attribute-value pairs

Customer Account: thought of as a spreadsheet; it may contain multiple individual sheets

Domains: individual sheets within the spreadsheet container are called domains (analogous to tables)

Items: The individual rows within domains are called items and may contain one or more attribute name-value pairs

Attributes: The individual columns within a domain or sheet are called attributes; they represent categories of data that can be assigned to items

Values: The individual cells within a domain or sheet are called values. Values are instances of attributes or the actual values

NoSQL Graph Database – Neo4J

An intuitive graph-oriented model for data representation

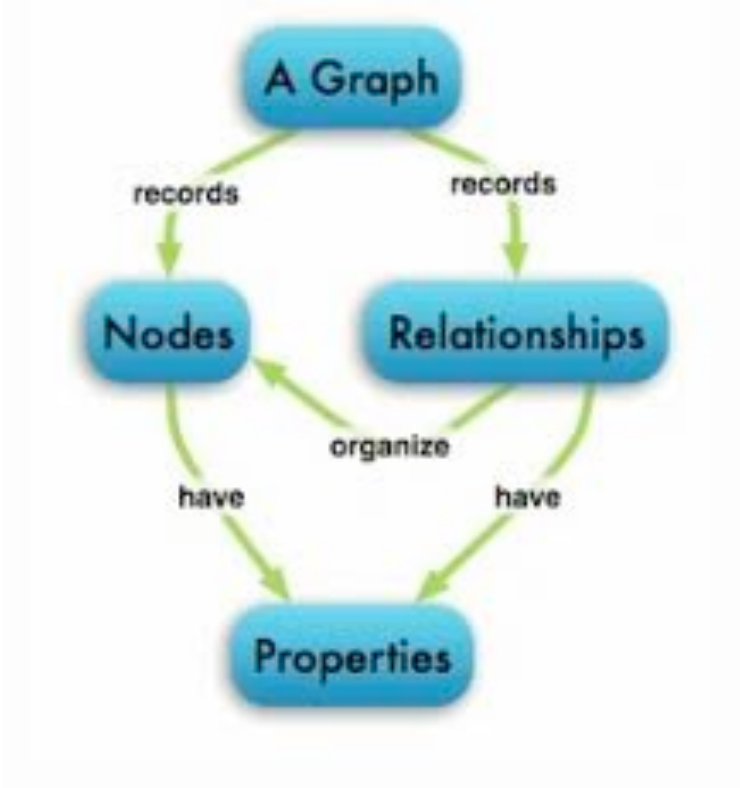
A disk-based, native storage manager completely optimized for storing graph structures for maximum performance and scalability

Massive scalability. Neo4j can handle graphs of several billion nodes/relationships/properties on a single machine and can scale out across multiple machines

A powerful traversal framework for high-speed traversals in the node space

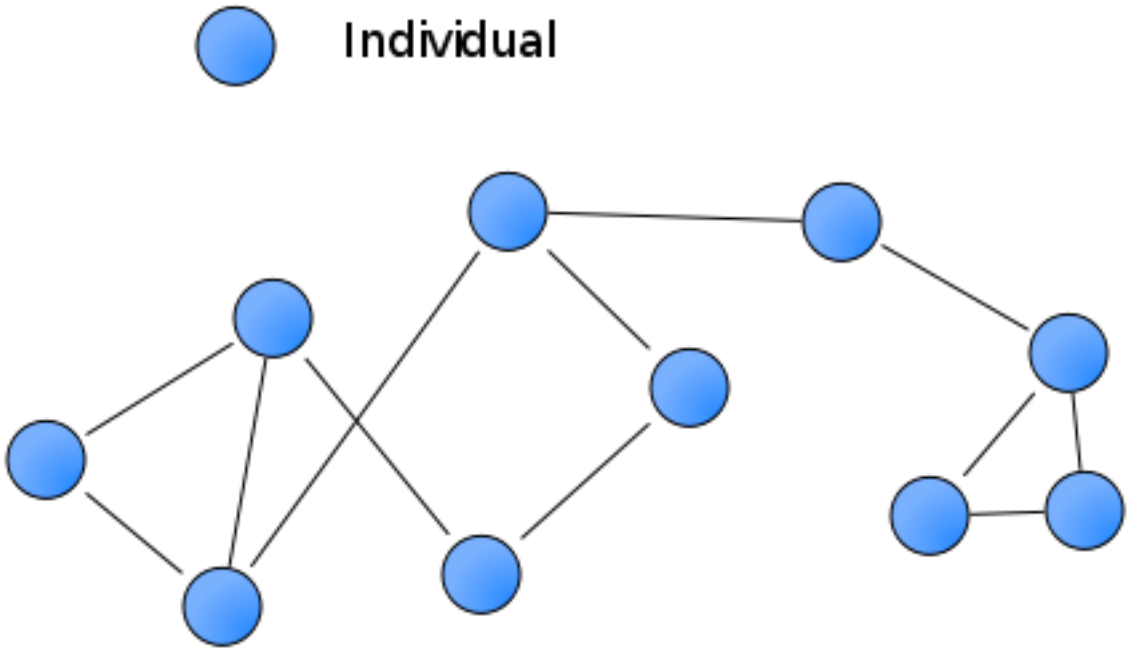
Can be deployed as a full server or a very slim database with a small footprint

A simple and convenient object-oriented API

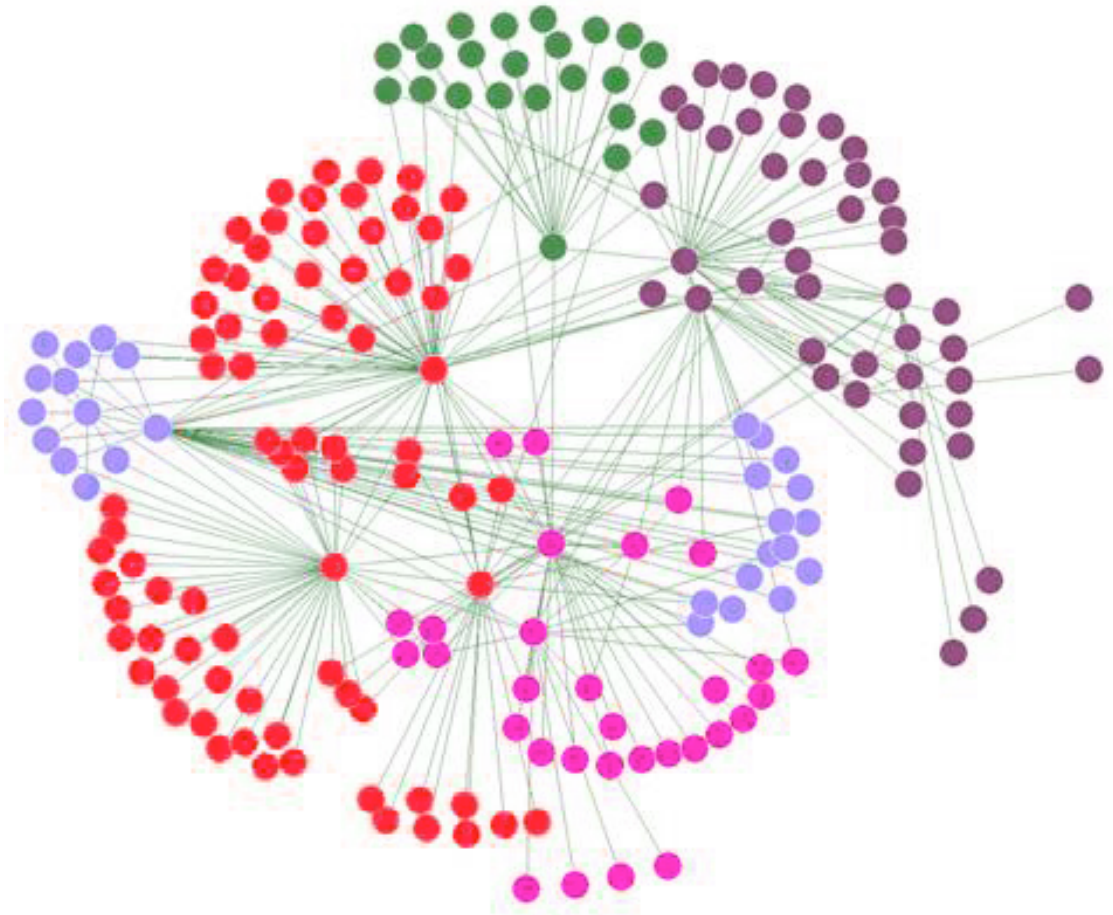


STREAM INTEGRATION

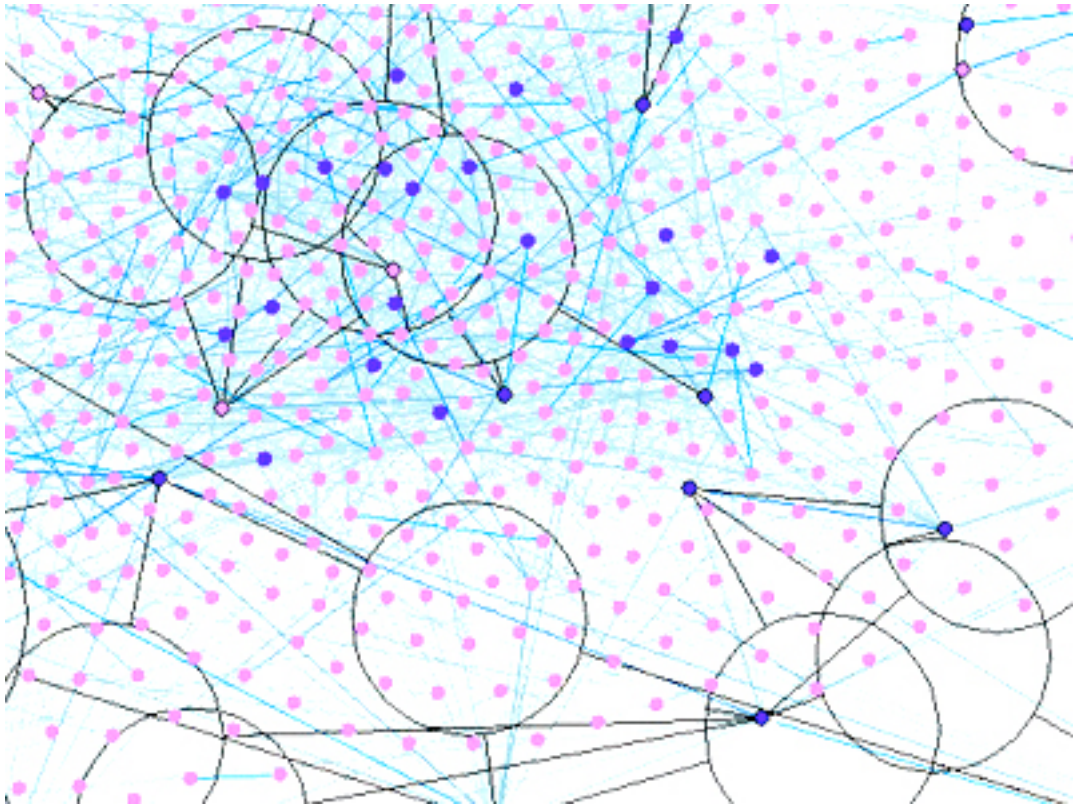
Micro Level



Mesa Level



Macro Level



Friend of a Friend (FOAF)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#JW>
   a foaf:Person ;
   foaf:name "Jimmy Wales" ;
   foaf:mbox <mailto:jwales@bomis.com> ;
   foaf:homepage <http://www.jimmywales.com/> ;
   foaf:nick "Jimbo" ;
   foaf:depiction <http://www.jimmywales.com/aus_img_small.jpg> ;
   foaf:interest <http://www.wikimedia.org> ;
   foaf:knows [
      a foaf:Person ;
      foaf:name "Angela Beesley"
   ] .

<http://www.wikimedia.org>
   rdfs:label "Wikipedia" .
```

Non-Relational Operational - Splunk

Collects and indexes all types of machine data.

Application Management

enables operational visibility across the applications to better manage downtime

Security and Compliance

real-time security monitoring, historical analysis and visualization of massive data sets, providing security intelligence for both known and unknown threats

data exploration of incidents in real time to perform incident investigations, maintain a proactive defense and support the creation of ad hoc reports

Infrastructure and IT Operations Management

ability to detect and investigate network, server and storage issues—located within physical, virtual or cloud infrastructures

Business and Web Analytics

real-time analysis of digital asset views/usage, site performance, web sessions—including analysis of pages viewed, content in shopping baskets, value of dropped baskets etc

Machine Data

Application Logs

Web Access Logs

Web Proxy Logs

Call Detail Records

ClickStream Data

Message Queues

Packet Data

Configuration Files

Database Audit Logs

Filesystem Audit Logs

Management and Logging APIs

OS Metrics, Status and Diagnostic Commands

Syslog, WMI and more….

STREAM INTEGRATION

Data Discovery

BigSheets

IBM Data Content CCI etc

Oracle Endeca

- Making it as intuitive as using google…
- indexed search on social media, websites, content systems, email, and database text - providing unprecedented visibility into data and business processes, saving time and cost, and leading to better business decisions

STREAM INTEGRATION

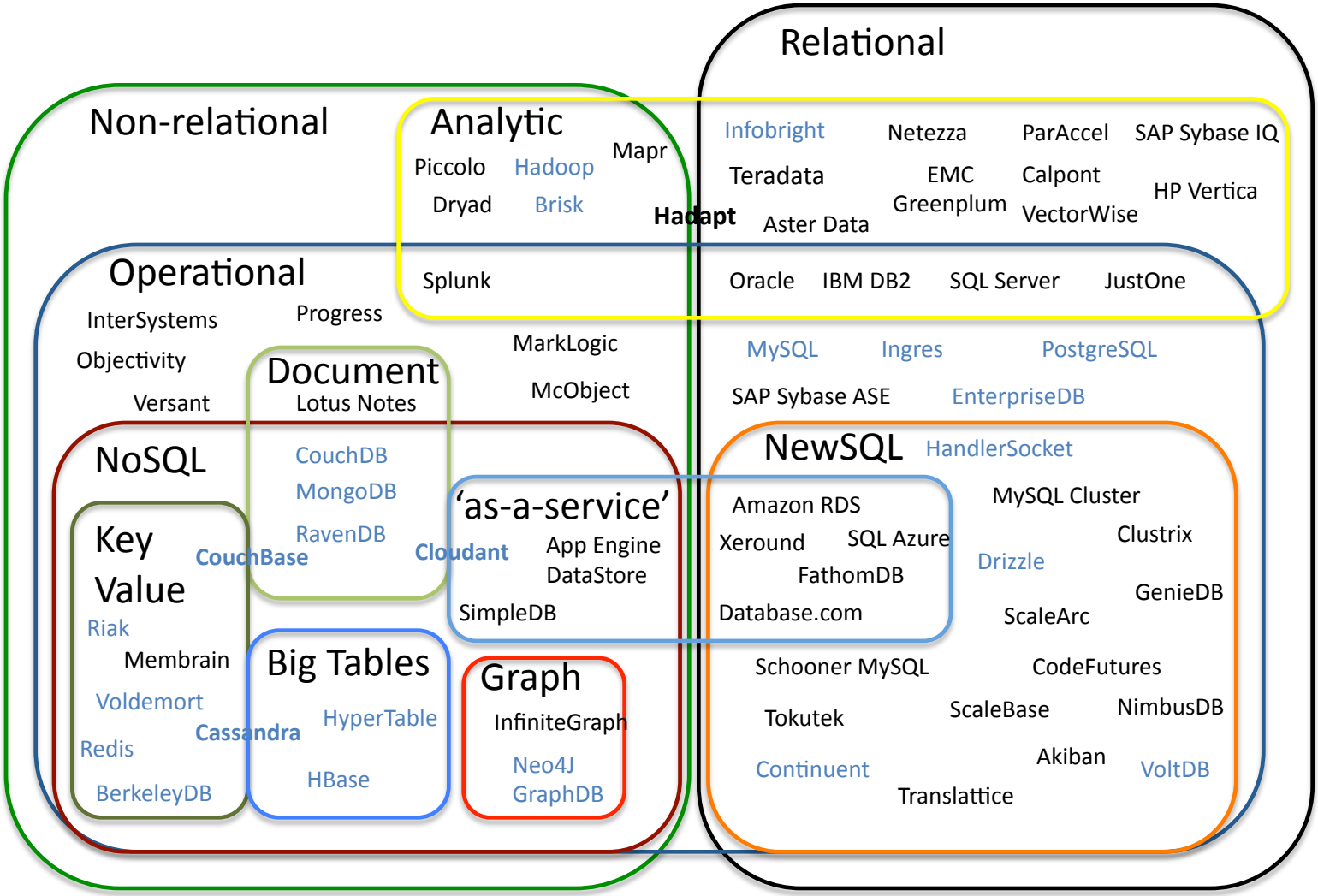
Oracle Endeca

Easily combine structured and unstructured data
and metadata to understand key metrics in their
relevant context and evaluate new business situations.

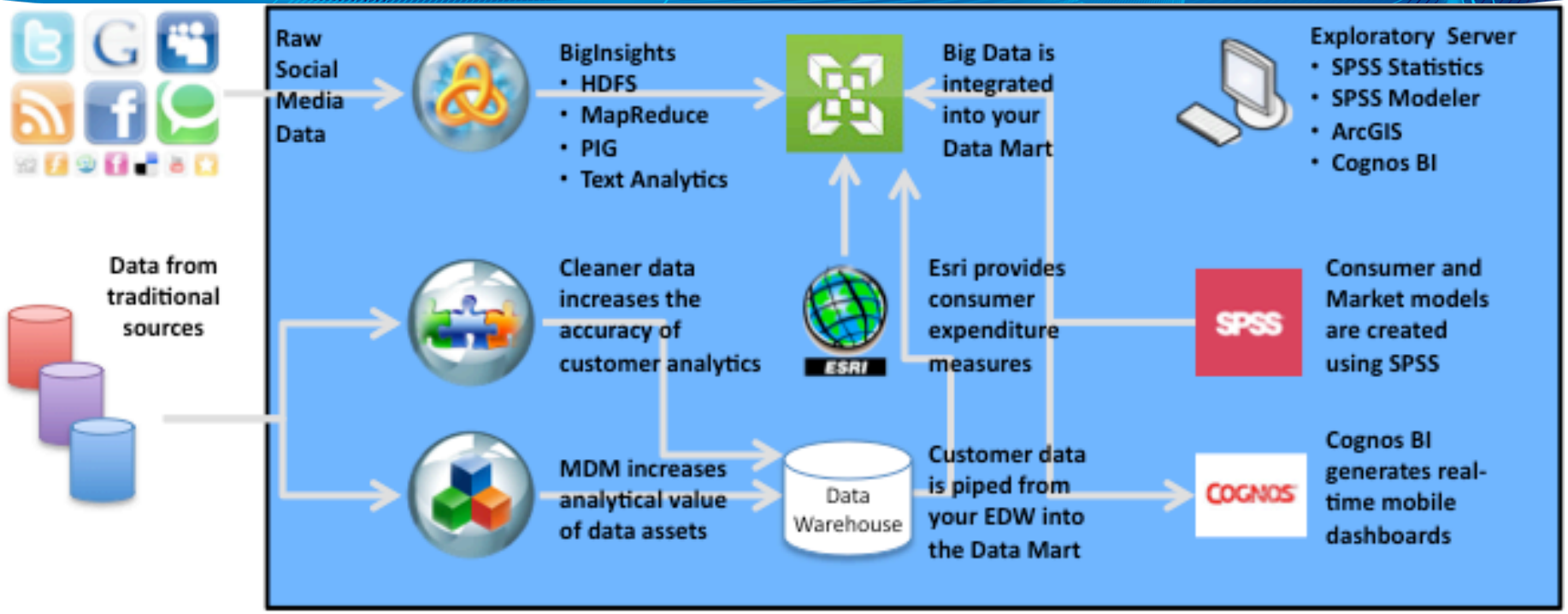
- Ask unanticipated questions of any data through in
tuitive, flexible, and highly interactive online
discovery applications.
- Mash up and recombine data and visualizat
ions to compose discovery applications.
- Leverage all of the rich metadata of existing Bu
siness Intelligence data models and semantic layers as
a basis upon which to build discovery applications.
- Unburden IT from the constant chasing of ne
w requirements and non-traditional data sources for
incorporation into the data warehouse, enabling them
to deliver fast access to relevant data and self-
service to business users while maintaining security, governance, and quality.

Summary

DBMS Categorization



Integrated Big Data Architecture



Stream Solutions and Accelerators

STREAM Stream Media Connector	STREAM Social Media Analytics	STREAM PureData System Integration	STREAM Stream Analytic Maps
	STREAM StreamClean Quality Service	STREAM Stream Spatial Binding	STREAM Stream Consumer Analytics
	STREAM MDM Framework	STREAM Integration Framework	STREAM Social CHATboard™

For more information

Email: paul.flach@streamintegration.com

Thank you!