Hadoop vs Databases

Hadoop limitations

1. Unstructured data
2. No random access
3. High latency
4. Not ACID compliant

All these limitations make Hadoop unsuited for transaction processing

HBase

is a distributed database management system that's part of the Hadoop ecosystem

HBase

HBase uses HDFS to store it's underlying data

HBase

HBase has the architecture benefits of HDFS

1. Distributed storage
2. Fault tolerance

HBase

It also has many of the properties required for transaction processing

1. Awareness of the structure of data
2. Low latency
3. Random access
4. ACID compliant at some levels

HBase

To understand HBase

it's helpful understand how it's different from a traditional RDBMS

HBase vs RDBMS

In a traditional RDBMS, all operations like creating, inserting, updating rows are done using SQL

HBase does not support SQL

HBase vs RDBMS

Only CRUD operations

HBase only supports a basic set of operations (Create-Read-Update-Delete)

HBase vs RDBMS

(Create-Read-Update-Delete)

Only CRUD operations

All these operations have to be applied at a row level

HBase vs RDBMS

(Create-Read-Update-Delete)

Only CRUD operations

HBase **does not support** any operations across rows (or) across tables

HBase vs RDBMS

(Create-Read-Update-Delete)

Only CRUD operations

This means that you cannot perform operations like

Joins, Group by etc

HBase vs RDBMS

Only CRUD operations

Denormalized

HBase tables are not designed using a relational data model

HBase vs RDBMS

Only CRUD operations

Denormalized

All the data pertaining to an entity is stored in 1 row (ie tables are denormalized)

HBase vs RDBMS

Only CRUD operations

Denormalized

Column oriented storage

HBase has a special kind of data model

HBase vs RDBMS

Only CRUD operations
Denormalized
Column oriented storage

ACID at a row level

HBase is ACID compliant for limited kinds of transactions

HBase vs RDBMS

Column oriented storage

Denormalized

Only CRUD operations

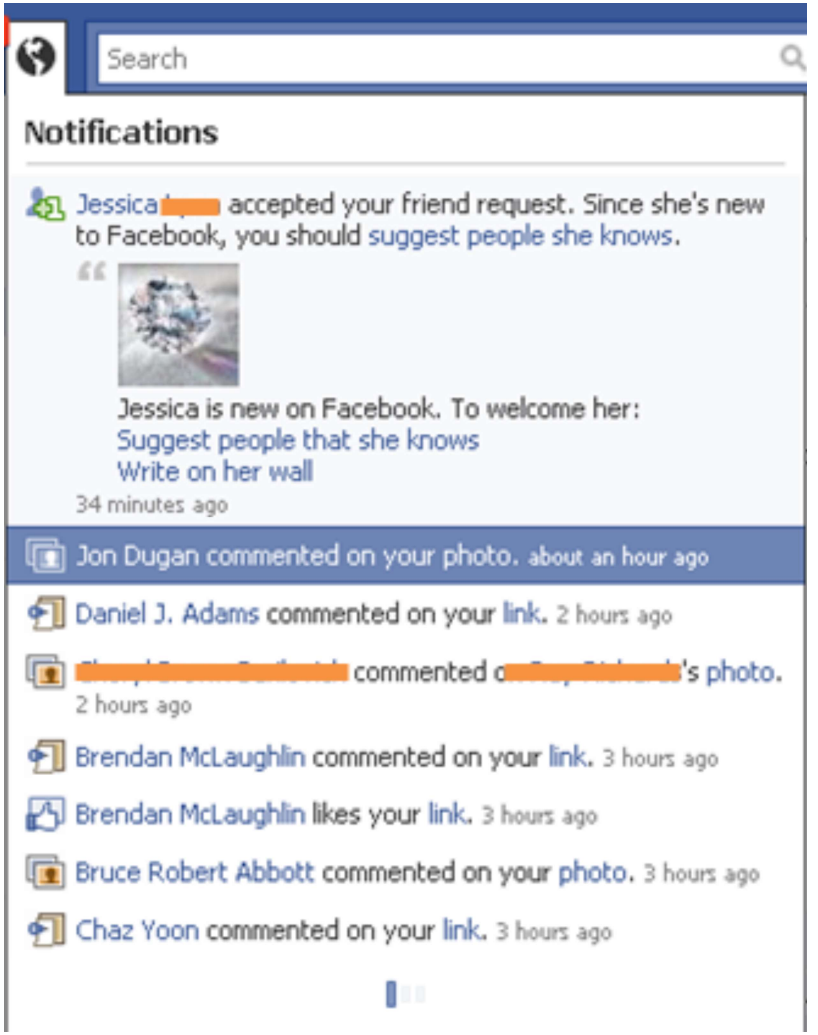
ACID at a row level

Let's understand the implications of each of these

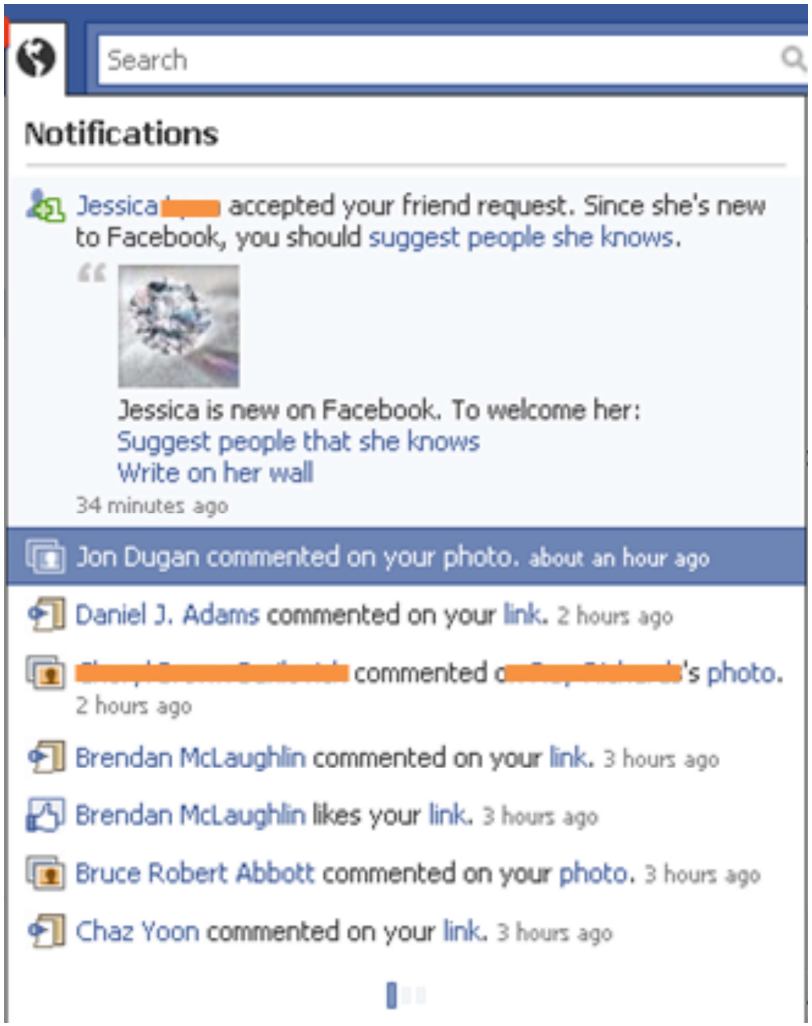
HBase vs RDBMS

Column oriented storage

Say we have an application that **manages notifications** to the users of a social network



HBase vs RDBMS



Column oriented storage

Here is a table that stores some notification related data

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

HBase vs RDBMS

Column oriented storage

This is how data is stored in traditional databases

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

HBase vs RDBMS

Column oriented storage

A table with a fixed schema is defined

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

HBase vs RDBMS

Column oriented storage

Each row represents a data point

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

HBase vs RDBMS

Column oriented storage

In a column oriented store, each cell represents a datapoint

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

Data is stored in a map

Key = <Row id, Col id>

Value = <data>

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

Data is stored in a map

Key = 2, for_user

Value = Chaz

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

row	column	value
1	type	Friend request status
1	for user	Ryan
1	from user	Jessica
1	timestamp	146710201
2	type	Comment
2	for user	Chaz
2	from user	Daniel
2	timestamp	146711200
3	type	Comment
3	for user	Rick
3	from user	Brendan
3	timestamp	1467112205

HBase vs RDBMS

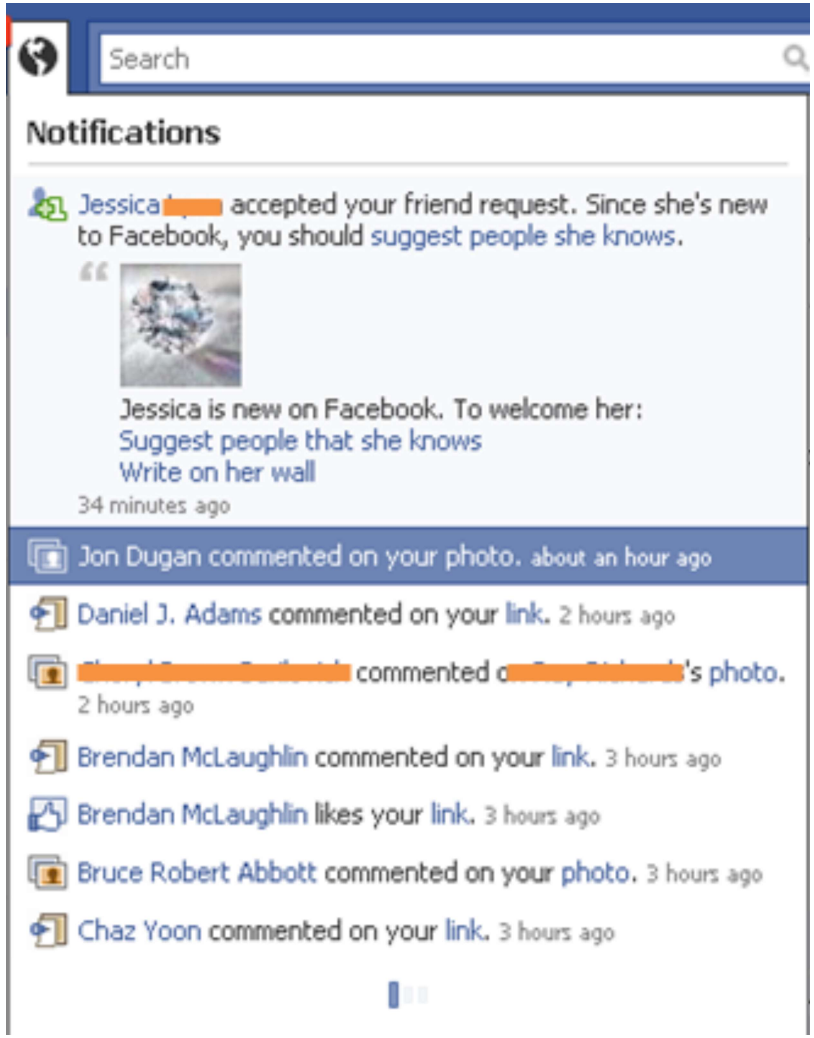
Column oriented storage

An HBase table is in fact a **sorted map**

Keys Values

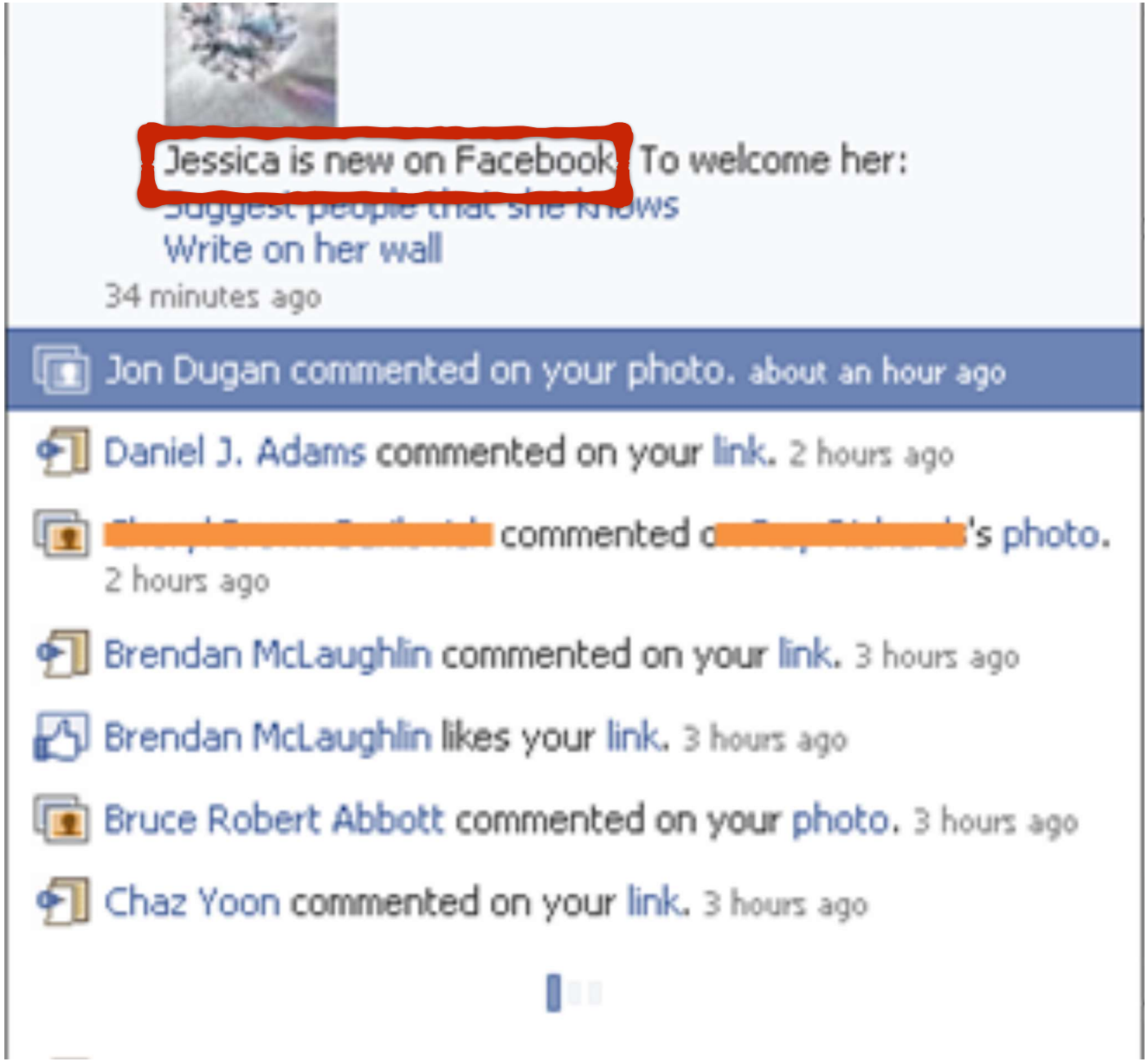
row	column	value
1	type	Friend request status
1	for user	Ryan
1	from user	Jessica
1	timestamp	146710201
2	type	Comment
2	for user	Chaz
2	from user	Daniel
2	timestamp	146711200
3	type	Comment
3	for user	Rick
3	from user	Brendan

HBase vs RDBMS



Column oriented storage

Let's say some notifications have special attributes depending on their type



Column oriented storage

Friend request notifications might have information about the friend

Column oriented storage

Comments and likes have information about a link or photo that prompted them

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request	Ryan	Jessica	146710201	new	-
2	Comment	Chaz	Daniel	146711200	-	link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

In the RDBMS table, each of these attributes becomes a new column

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request	Ryan	Jessica	146710201	new	-
2	Comment	Chaz	Daniel	146711200	-	link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

This results in tables that are very sparse

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request	Ryan	Jessica	146710201	new	-
2	Comment	Chaz	Daniel	146711200	-	link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

In an RDBMS, Sparse tables utilize disk space even for these empty cells

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request	Ryan	Jessica	146710201	new	-
2	Comment	Chaz	Daniel	146711200	-	link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

In a column-oriented store, these cells can be skipped completely

HBase vs RDBMS

Column oriented storage

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request status	Ryan	Jessica	146710201	new	
2	Comment	Chaz	Daniel	146711200	-	link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

row id	column	value
1	type	Friend request status
1	for user	Ryan
1	from user	Jessica
1	timestamp	146710201
1	friend type	new

HBase vs RDBMS

id	type	for user	from user	timestamp	friend type	commented on
1	Friend request status	Ryan	Jessica	146710201	new	
2	Comment	Chaz	Daniel	146711200		link
3	Comment	Rick	Brendan	1467112205	-	photo
4	Like	Rick	Brendan	1467112213	-	-

Column oriented storage

row id	column	value
1	type	Friend request status
1	for user	Ryan
1	from user	Jessica
1	timestamp	146710201
1	friend type	new
2	type	Friend request status
2	for user	Ryan
2	from user	Jessica
2	timestamp	146710201
2	commented on	link

HBase vs RDBMS

Column oriented storage

Column oriented storage has some powerful advantages

1. You can store really sparse tables very efficiently

2. You can accommodate dynamically changing attributes

HBase vs RDBMS

Column oriented storage

Each row id can have a different set of col ids

2. You can accommodate dynamically changing attributes

HBase vs RDBMS

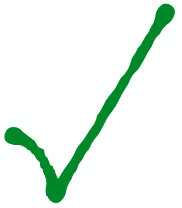
Column oriented storage

The schema for a row id is not fixed, you can keep changing it

ie, Add or remove new col ids

2. You can accommodate dynamically changing attributes

HBase vs RDBMS

Column oriented storage 

Denormalized

Only CRUD operations

ACID at a row level

HBase vs RDBMS

Denormalized

LET'S SAY WE HAVE AN EMPLOYEES DATABASE

WE WANT TO CAPTURE EMPLOYEE NAME, ADDRESS, SUBORDINATES

HBase vs RDBMS

Denormalized

A TRADITIONAL RDBMS WOULD MODEL IT AS 3 TABLES

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

HBase vs RDBMS

Denormalized

A TRADITIONAL RDBMS WOULD MODEL IT AS 3 TABLES

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

THIS KIND OF DESIGN MINIMIZES REDUNDANT STORAGE OF DATA

HBase vs RDBMS

Denormalized

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

THIS KIND OF DESIGN MINIMIZES REDUNDANT STORAGE OF DATA

HBase vs RDBMS

Denormalized

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

THESE STREET AND CITY NAMES ARE ONLY STORED ONCE

AND REFERRED TO BY AN INTEGER ID THEREAFTER

HBase vs RDBMS

Denormalized

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

NORMALIZATION OPTIMIZES FOR STORAGE

HBase vs RDBMS

Denormalized

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

IN A DISTRIBUTED SYSTEM,
STORAGE IS CHEAP

INSTEAD YOU NEED TO
OPTIMIZE DISK SEEKS

HBase vs RDBMS

Denormalized

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

IF YOU STORE DATA ACROSS DIFFERENT TABLES

YOU HAVE TO PERFORM DISK SEEKS FOR EACH TABLE

HBase vs RDBMS

Denormalized

INSTEAD WE CAN EMBED ALL 3 TABLES INTO A SINGLE TABLE

EmpID	EmpName	Address	Subordinates
1	Vitthal	<STRUCT>	<ARRAY>

```
"Street":"Bellandur",
"City":"Bangalore"
```

```
("Anuradha",
"Arun",
"Swetha")
```

HBase vs RDBMS

Denormalized

EmpID	EmpName	Address	Subordinates
1	Vitthal	<STRUCT>	<ARRAY>

THIS IS A DENORMALIZED DESIGN

HBase vs RDBMS

Denormalized

EmpID	EmpName	Address	Subordinates
1	Vitthal	<STRUCT>	<ARRAY>

ALL THE DATA CORRESPONDING TO AN EMPLOYEE IS STORED IN A SINGLE TABLE

HBase vs RDBMS

Denormalized

IN HBASE DATA IS STORED IN A DENORMALIZED MANNER

HBase vs RDBMS

Column oriented storage ✓

Denormalized ✓

Only CRUD operations

ACID at a row level

HBase vs RDBMS

Only CRUD operations

HBase architecture is designed such that you can get random read-write access to a specific row

HBase vs RDBMS

Only CRUD operations

Unlike, traditional RDBMS, HBase does not support SQL

NoSQL

HBase vs RDBMS

Only CRUD operations

HBase only supports a limited set of operations

HBase vs RDBMS

Only CRUD operations

HBase only supports a limited set of operations

Create Add a new value to the table

Read Read the value for a specific row id, col id

Update Update the value for a specific row id, col id

Delete Delete the value for a specific row id, col id

HBase vs RDBMS

Only CRUD operations

Create
Read
Update
Delete

All HBase operations deal with a specific row

HBase vs RDBMS

Only CRUD operations

Create
Read
Update
Delete

HBase does not support any operations across tables

No Joins

No Foreign key constraints

HBase vs RDBMS

Only CRUD operations

Create
Read
Update
Delete

HBase does not support any operations across row ids

No Grouping/Aggregation

HBase vs RDBMS

Only CRUD operations

Create
Read
Update
Delete

This is another reason why denormalization is important in HBase

HBase vs RDBMS

Only CRUD operations

Create
Read
Update
Delete

All the data needed to describe an entity should be self-contained within its row id

HBase vs RDBMS Only CRUD operations

LET'S GO BACK TO THE EMPLOYEE EXAMPLE

A TRADITIONAL RDBMS WOULD MODEL IT AS 3 TABLES

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

EmpID	SubordinateEmpID
1	3
1	4
1	8

HBase vs RDBMS Only CRUD operations

WHEN AN APPLICATION ASKS FOR AN EMPLOYEE'S DETAILS

EmpID	EmpName	AddressId
1	Vitthal	1

AddressId	Street	City
1	Bellandur	Bangalore

YOU WOULD NEED TO JOIN 2 TABLES TO FETCH THE ADDRESS

HBase vs RDBMS Only CRUD operations

WHEN AN APPLICATION ASKS FOR AN EMPLOYEE'S DETAILS

EmpID	EmpName	AddressId
1	Vitthal	1

EmpID	SubordinateEmpID
1	3
1	4
1	8

YOU WOULD NEED TO JOIN THESE 2 TABLES TWICE TO GET THE LIST OF SUBORDINATES FOR AN EMPLOYEE

HBase vs RDBMS

Only CRUD operations

IN AN RDBMS THESE JOINS CAN BE MADE EFFICIENT WITH THE ADDITION OF INDICES

HBase vs RDBMS

Only CRUD operations

IN HBASE, THERE IS NO SUPPORT FOR JOINING TABLES ON THE FLY WHILE FETCHING THE DETAILS FOR 1 ROW

HBase vs RDBMS

Only CRUD operations

YOU COULD USE AN EXTERNAL APPLICATION LIKE MAPREDUCE TO PERFORM JOINS

WHILE THIS IS FINE FOR ANALYTICAL QUERIES, IT WOULD NOT BE SUITABLE FOR TRANSACTION PROCESSING

HBase vs RDBMS Only CRUD operations

IN A DENORMALIZED DESIGN

EmpID	EmpName	Address	Subordinates
1	Vitthal	<STRUCT>	<ARRAY>

YOU CAN USE THE HBASE SUPPORTED READ OPERATION TO READ THE ROW AND FETCH ALL THE DATA

HBase vs RDBMS

Column oriented storage ✓

Denormalized ✓

Only CRUD operations ✓

ACID at a row level

HBase vs RDBMS

ACID at a row level

HBase is ACID compliant, but only at a row id level

For example, let's look at Atomicity

HBase vs RDBMS

ACID at a row level

Atomicity

Transaction 1:
Update values
for 2 col ids
within 1 row

Transaction 2:
Update values
for 2 col ids
for 10 row ids

HBase vs RDBMS

ACID at a row level

Atomic

Transaction 1:
Update values for 2 col ids within 1 row

Atomicity

If one col id update fails, the entire transaction fails

HBase vs RDBMS

If the operation fails after 5 row ids are updated, the row ids which are updated remain updated

ACID at a row level

Atomicity

Not Atomic

Transaction 2:

Update values for 2 col ids for 10 row ids

HBase vs RDBMS

Column oriented storage ✓

Denormalized ✓

Only CRUD operations ✓

ACID at a row level ✓

If you are familiar with the Hadoop ecosystem, you might know of other technologies which seem similar to HBase

HIVE FOR INSTANCE

HBase is a database management system

Used for both transaction processing and analytical processing

HIVE IS A DATA WAREHOUSE

Used only for analytical processing

HBase is a database management system

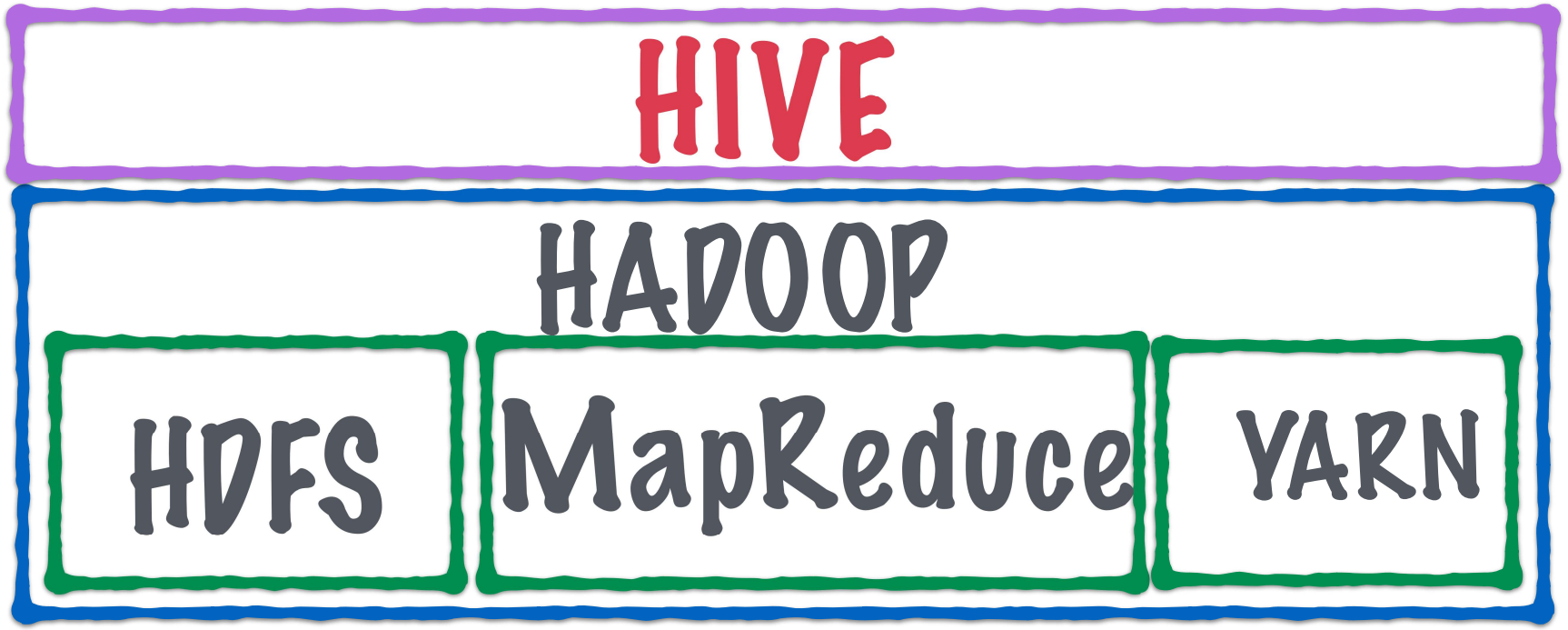
Provides low latency and random access for some supported operations

HIVE IS A DATA WAREHOUSE

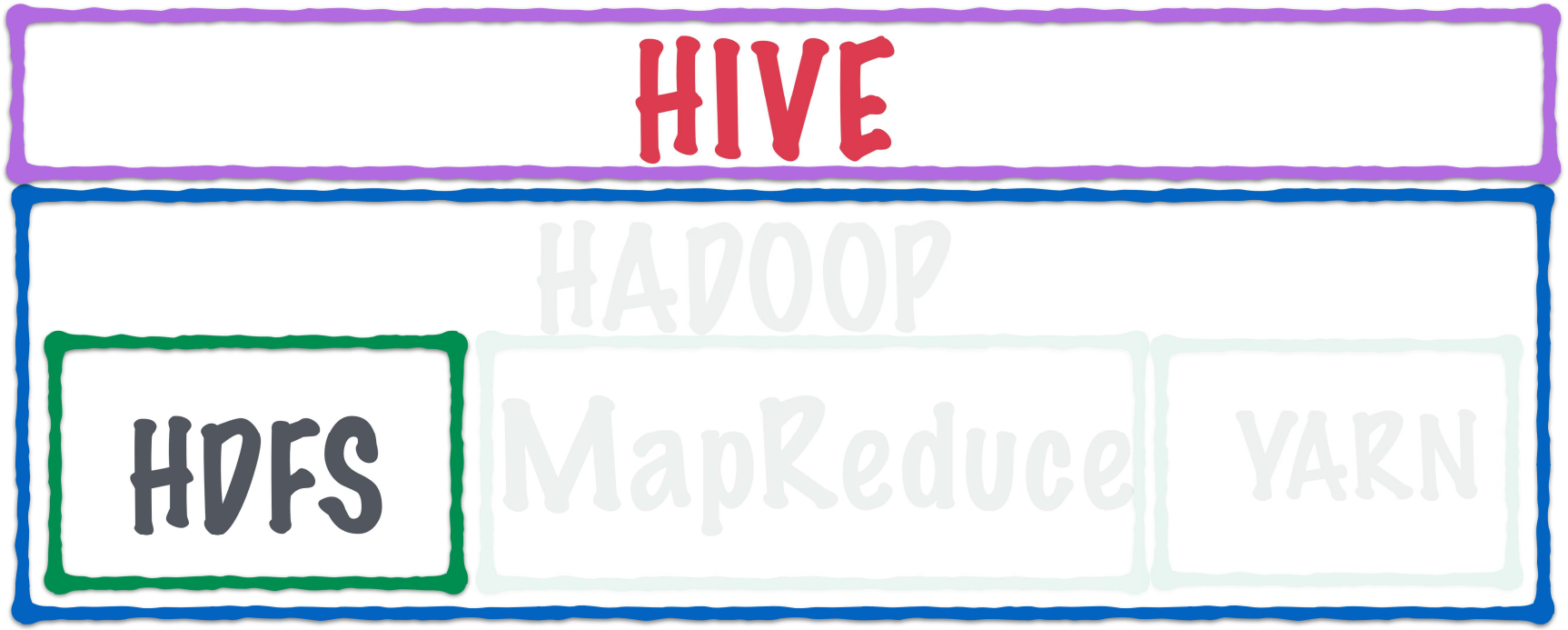
Only suitable for batch processing jobs that can tolerate high latency

HBase does not provide
any SQL interface

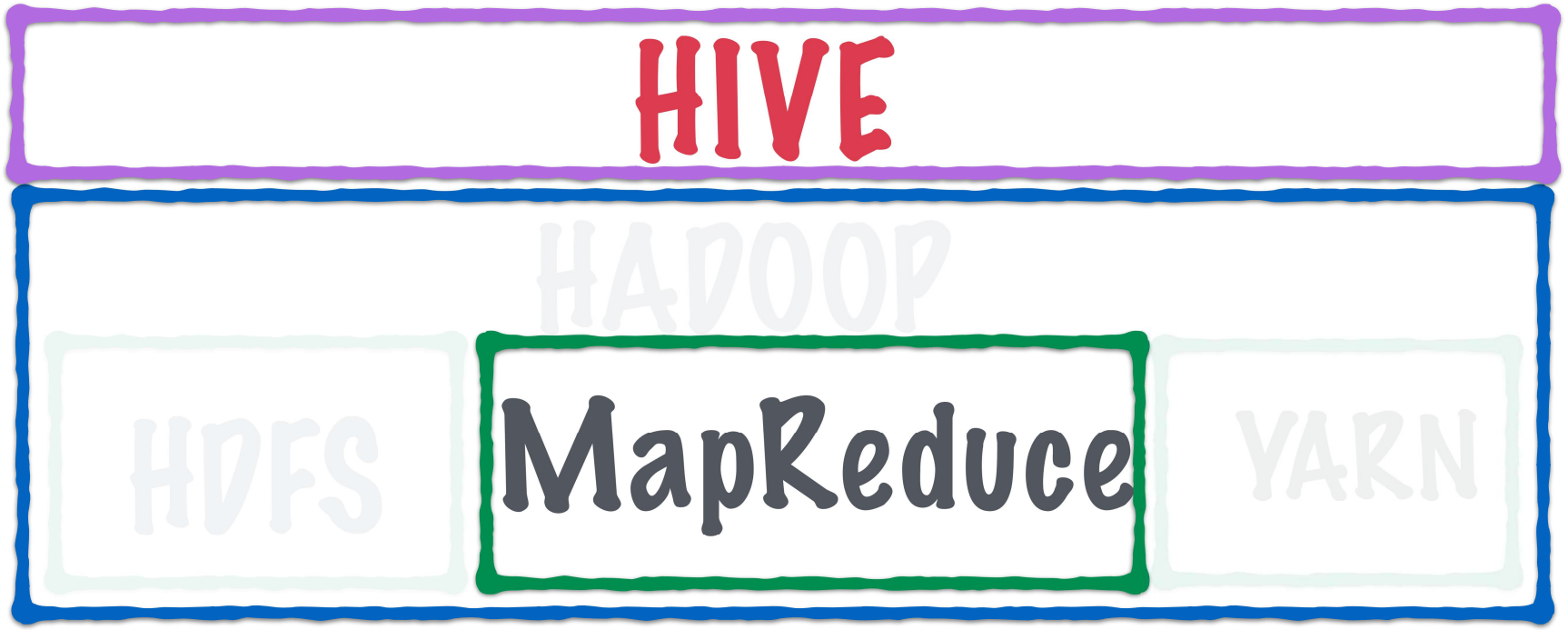
Hive does!



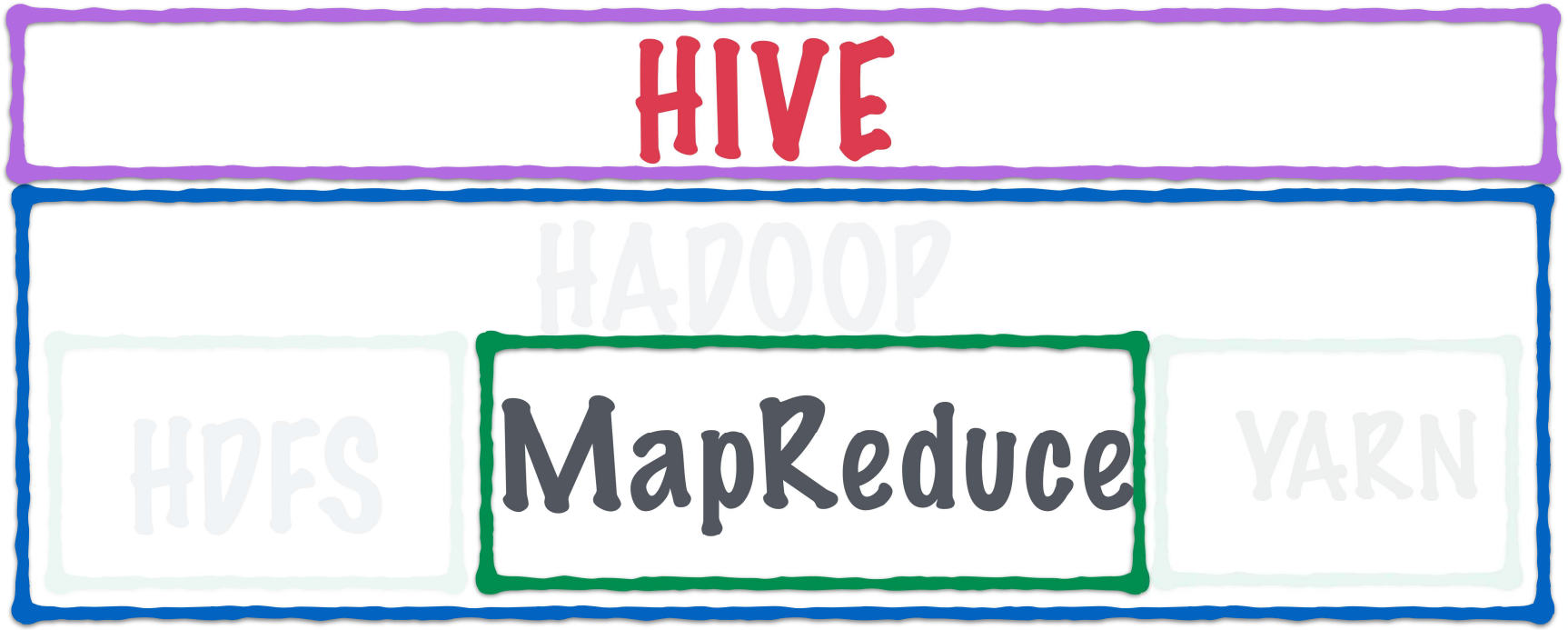
HIVE IS A DATAWAREHOUSE
BUILT ON TOP OF HADOOP



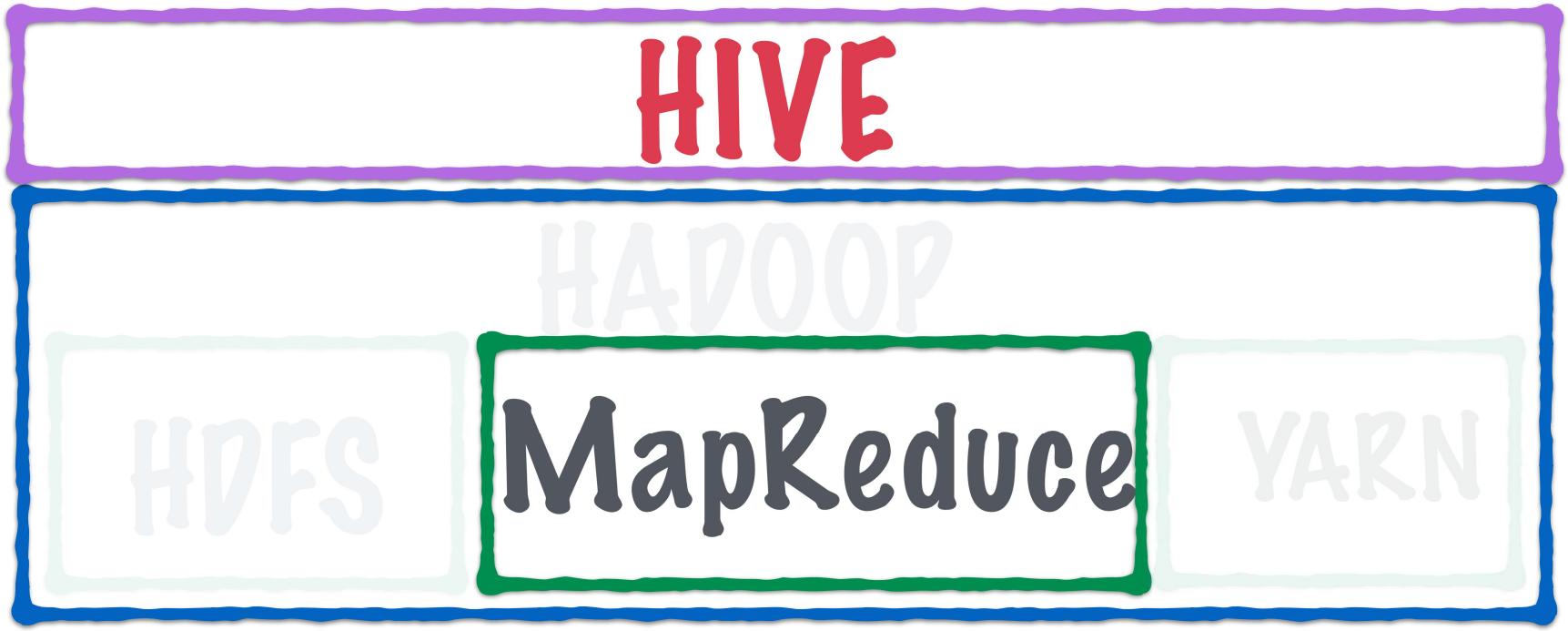
HIVE STORES IT'S DATA
AS FILES IN HDFS



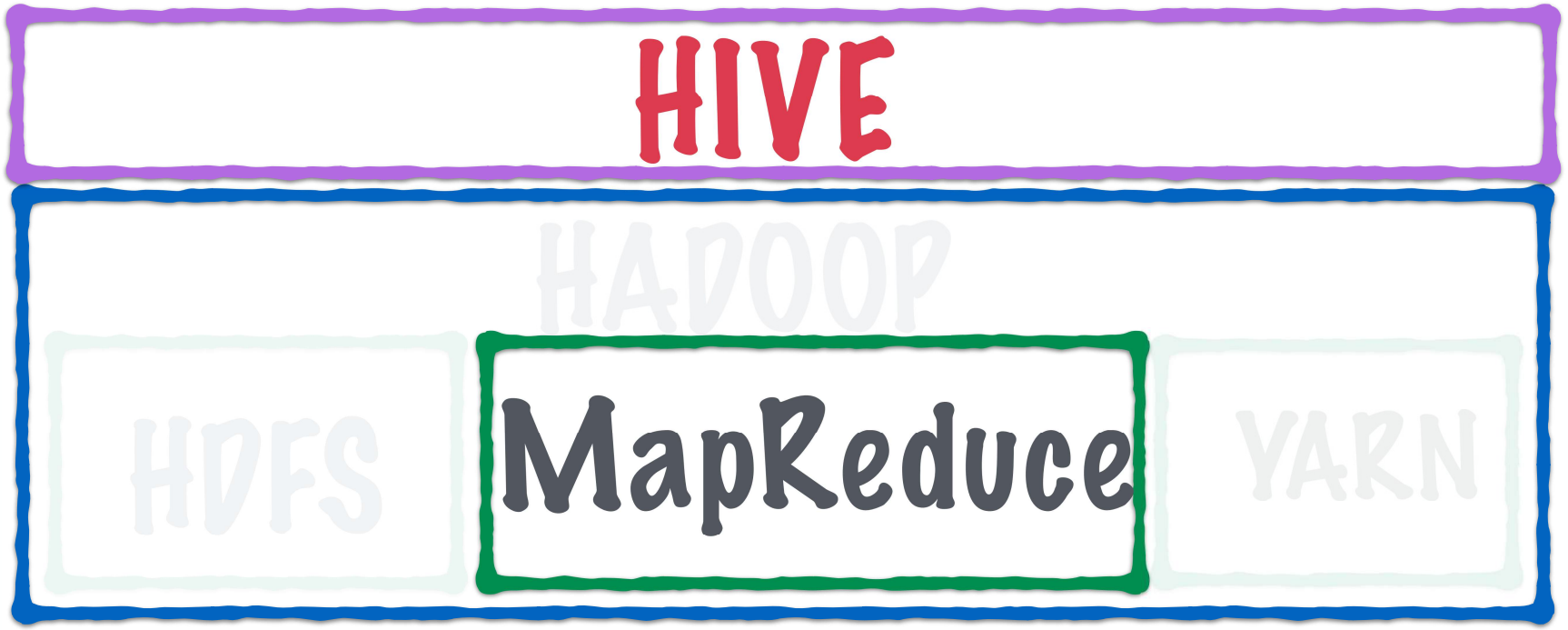
All processing tasks in Hadoop are run using MapReduce tasks



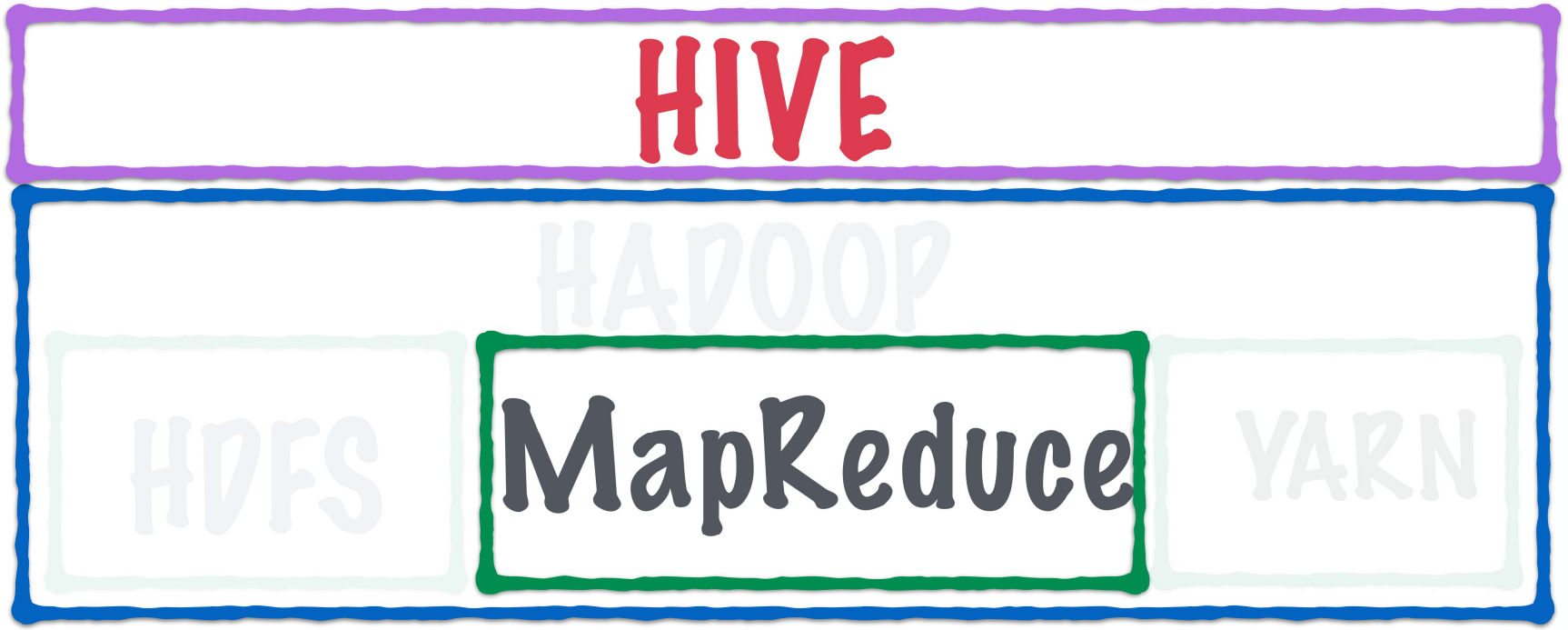
MapReduce tasks are usually written using a Java Framework



Writing these MapReduce tasks can be pretty daunting



Traditional databases/closed-source datawarehouses normally use SQL



SQL = Structured Query Language

SQL = Structured Query Language

SQL is really much easier to use and understand :)

SQL = Structured Query Language

It's widely used by analysts and programmers to work with databases/data warehouses

SQL = Structured Query Language

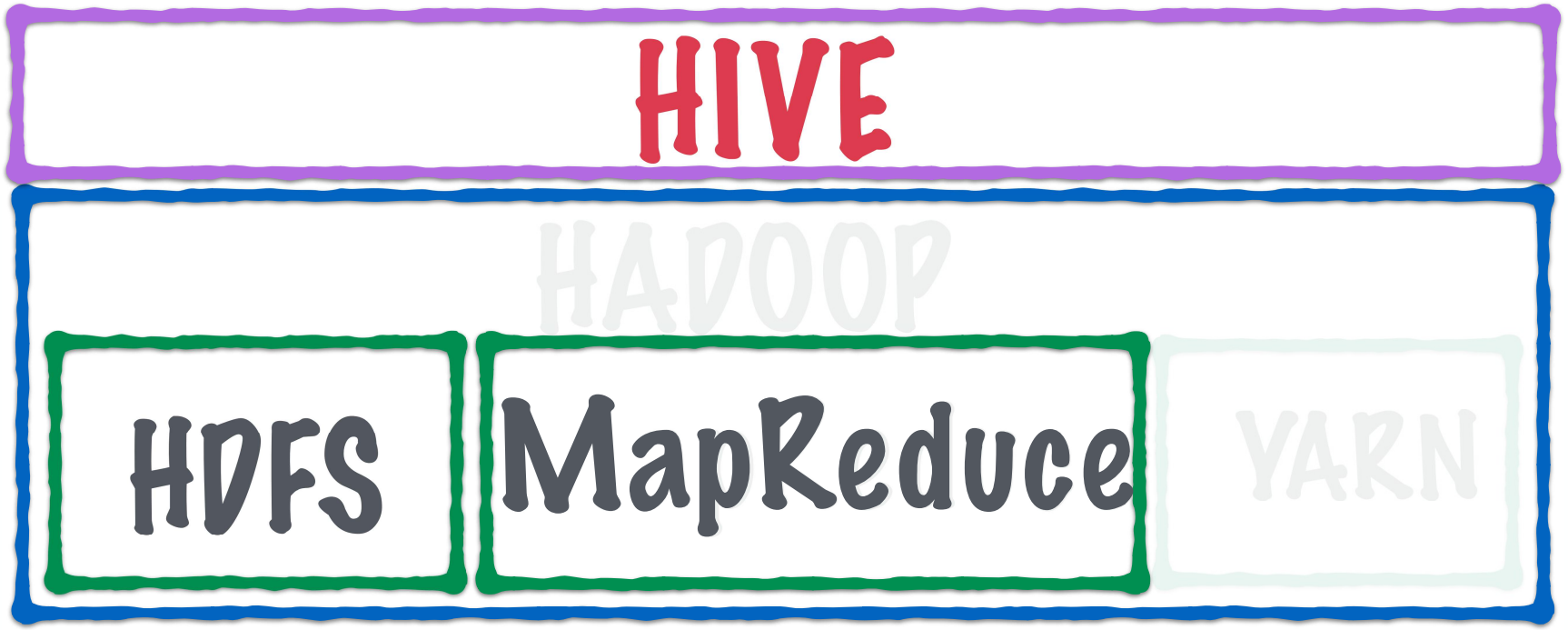
SQL has a few easy to understand constructs

Select, group by, join etc

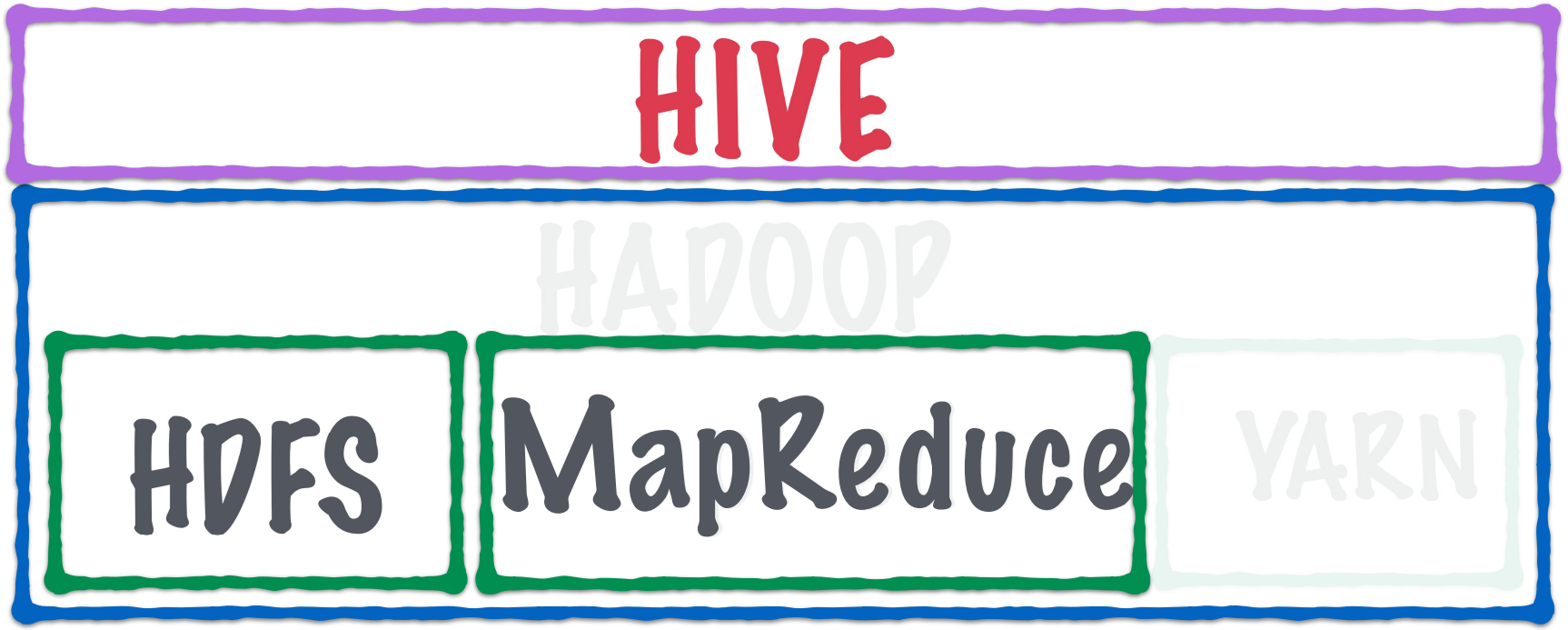
SQL = Structured Query Language

Most data processing tasks are defined using a combination of these constructs

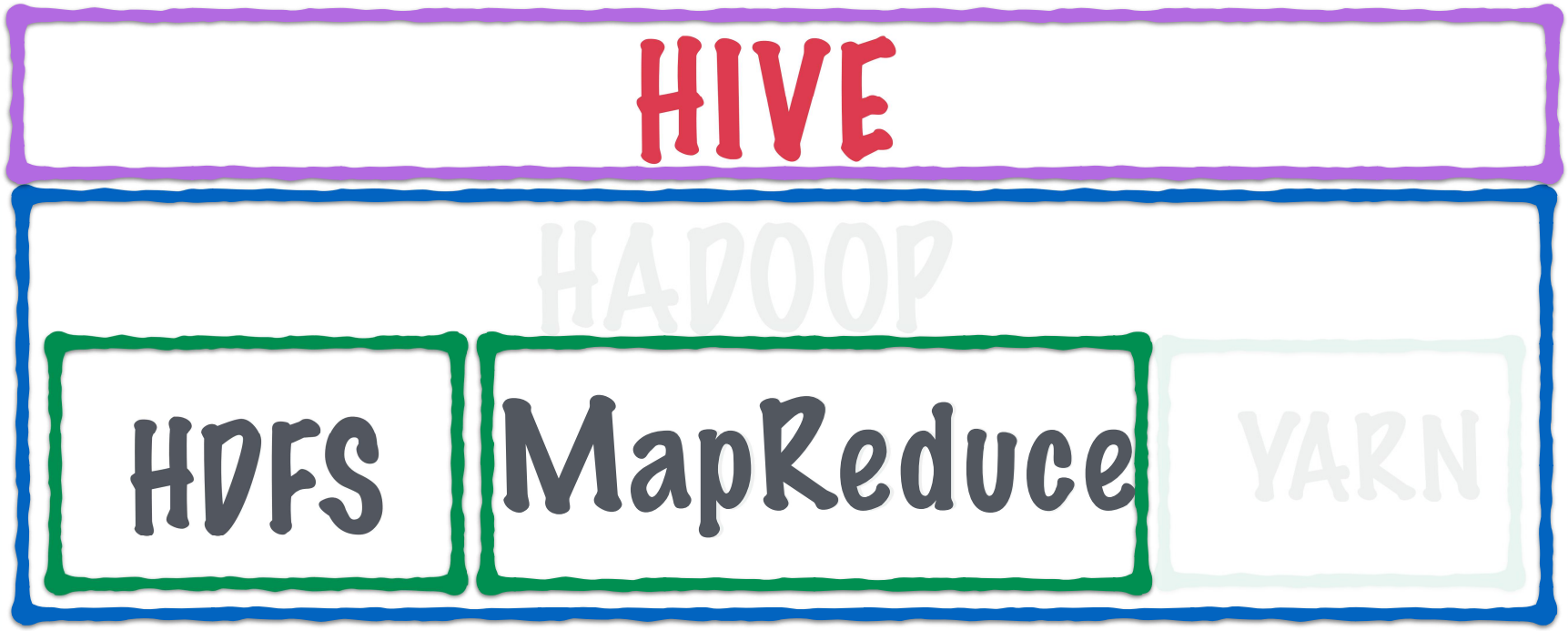
Select, group by, join etc



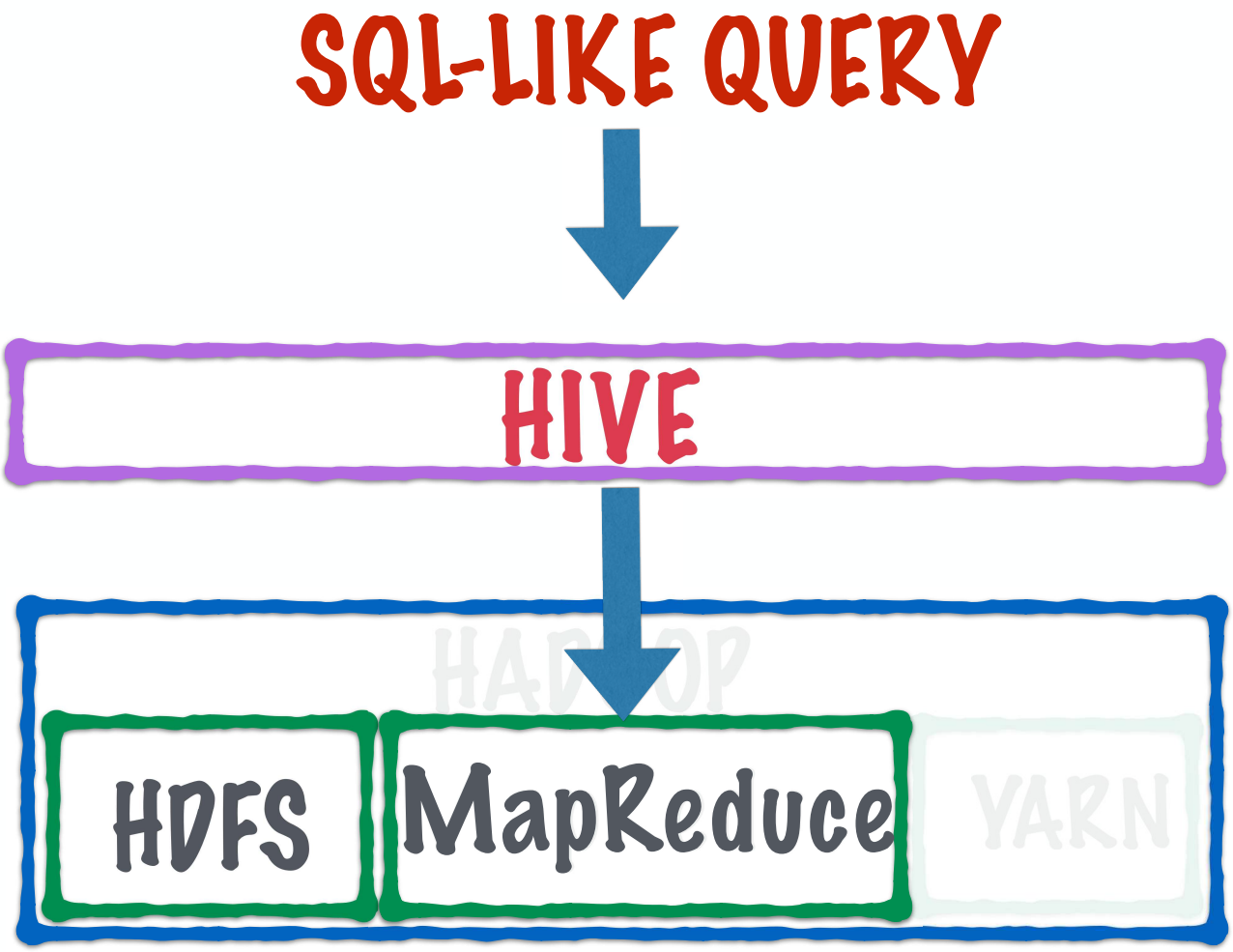
HIVE PROVIDES AN SQL LIKE INTERFACE TO DATA IN HDFS



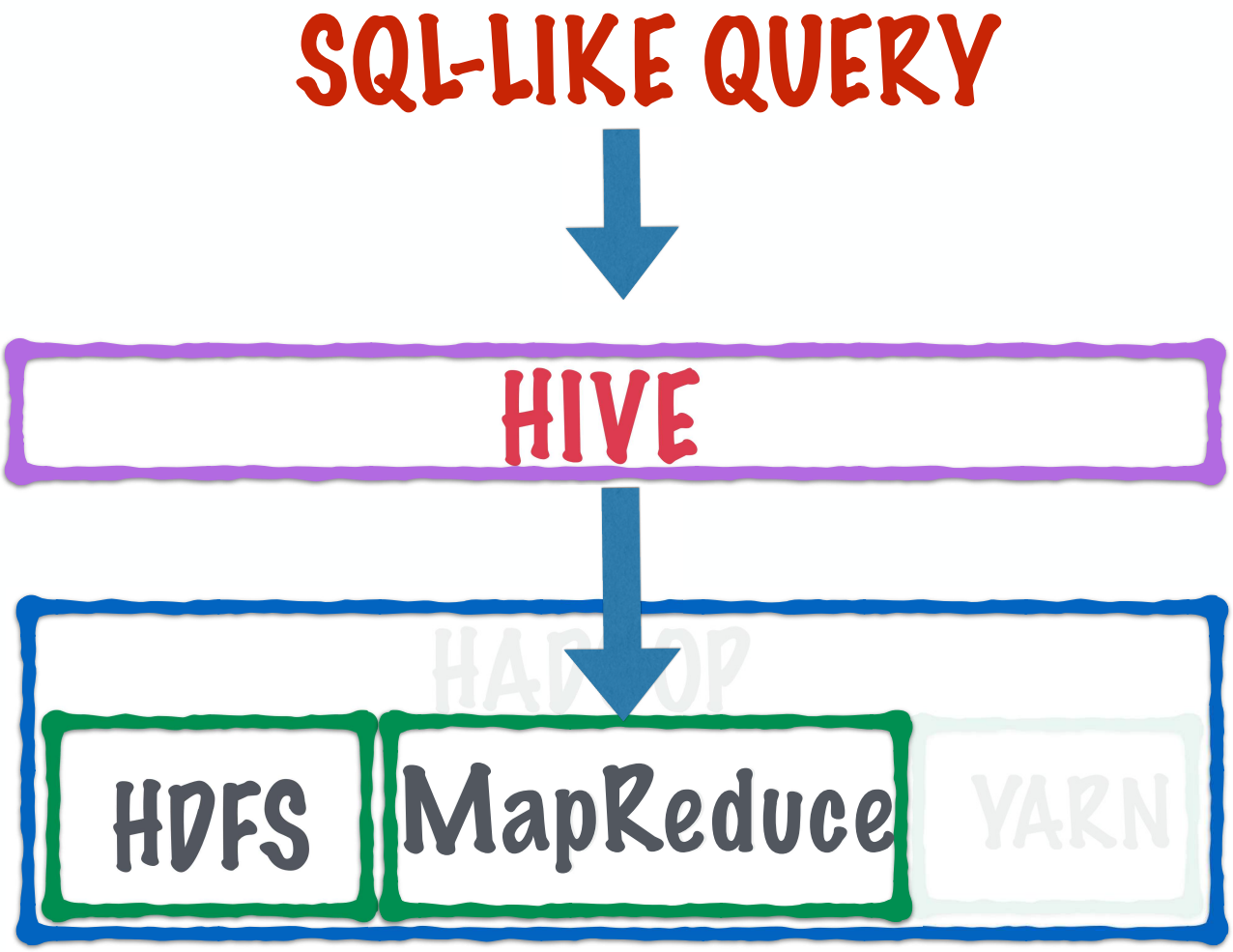
THE FILES IN HDFS ARE EXPOSED TO THE USER IN THE FORM OF TABLES



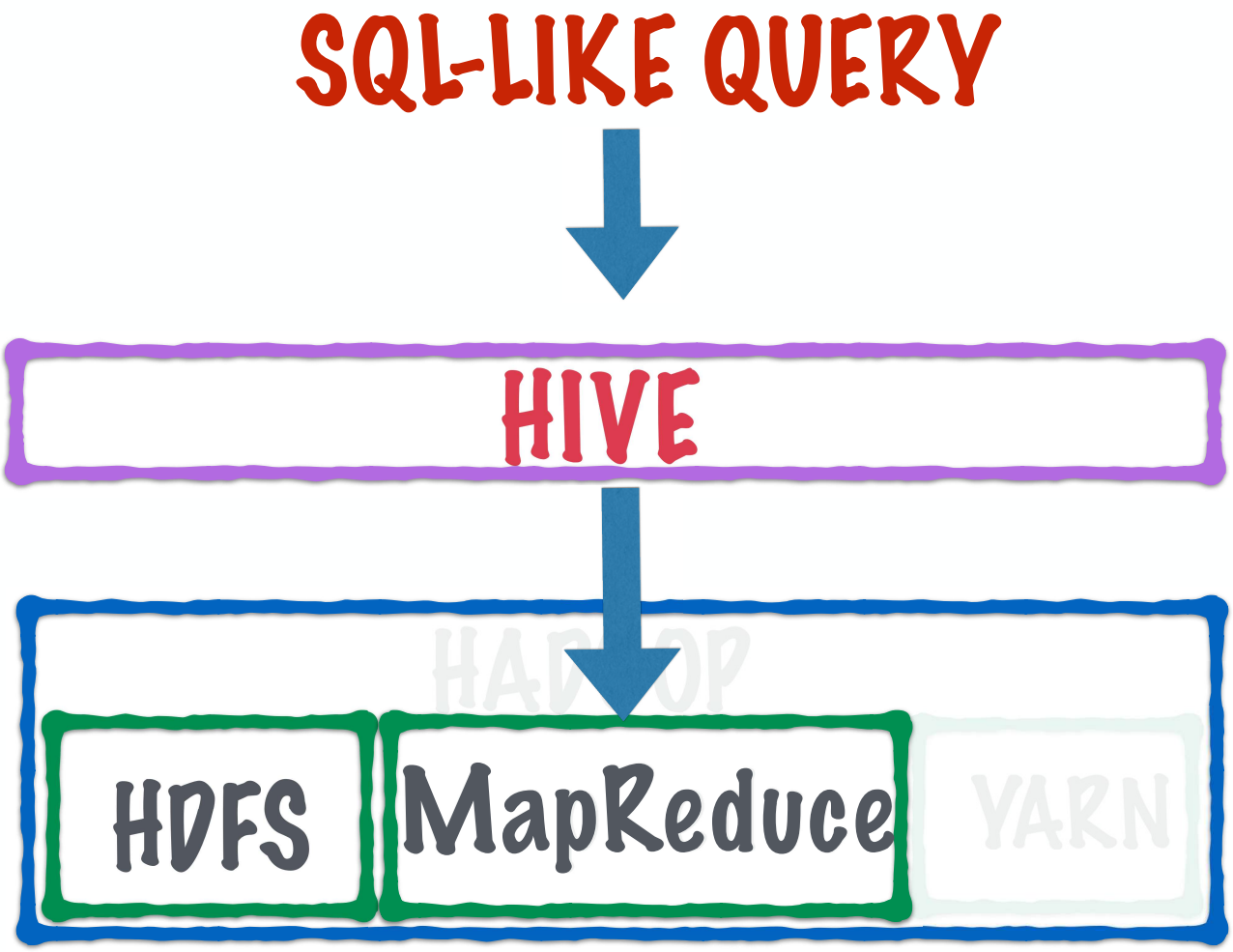
THE USER CAN WRITE SQL-LIKE QUERIES TO WORK WITH THESE TABLES



HIVE WILL TRANSLATE THE QUERY INTO 1/MORE MAPREDUCE TASKS



THE MAPREDUCE TASKS WILL PROCESS THE DATA IN HDFS AND RETURN ANY RESULTS TO HIVE



THE QUERIES ARE WRITTEN IN A SQL LIKE LANGUAGE CALLED HIVEQL

DIFFERENCES BETWEEN HIVE AND HBASE

HIVE

USED FOR BATCH PROCESSING

HBASE

USED FOR BOTH BATCH AND TRANSACTION PROCESSING

HIVE

USED FOR BATCH PROCESSING

PROVIDES AN SQL SKIN FOR HADOOP

HBASE

USED FOR BOTH BATCH AND TRANSACTION PROCESSING

NO SQL INTERFACE

HIVE

USED FOR BATCH PROCESSING

PROVIDES AN SQL SKIN FOR HADOOP

USES BOTH HDFS AND THE MAPREDUCE ENGINE

HBASE

USED FOR BOTH BATCH AND TRANSACTION PROCESSING

NO SQL INTERFACE

USES HDFS BUT HAS IT'S OWN ARCHITECTURE

HIVE

USED FOR BATCH PROCESSING

PROVIDES AN SQL SKIN FOR HADOOP

USES BOTH HDFS AND THE MAPREDUCE ENGINE

DATA MODEL IS SIMILAR TO DATABASES (TABLES WITH FIXED SCHEMA)

HBASE

USED FOR BOTH BATCH AND TRANSACTION PROCESSING

NO SQL INTERFACE

USES HDFS BUT HAS IT'S OWN ARCHITECTURE

DATA MODEL IS COLUMN ORIENTED STORAGE