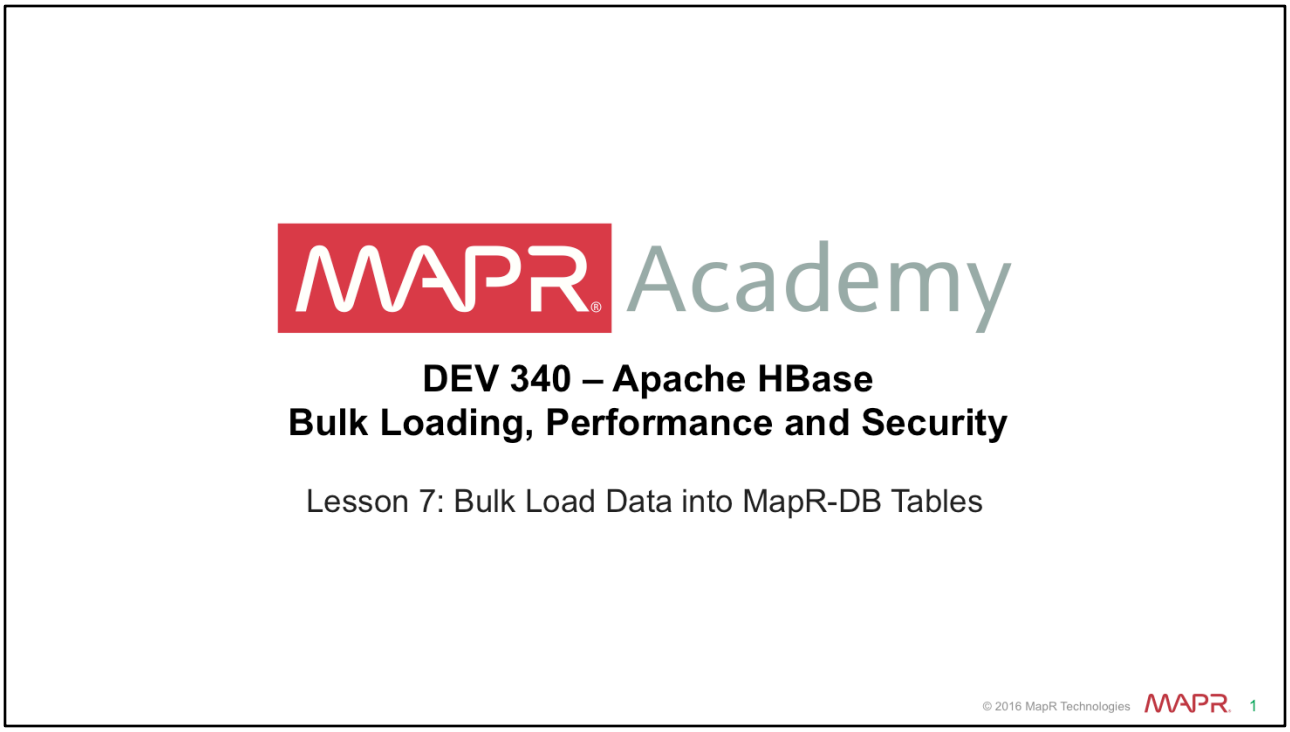
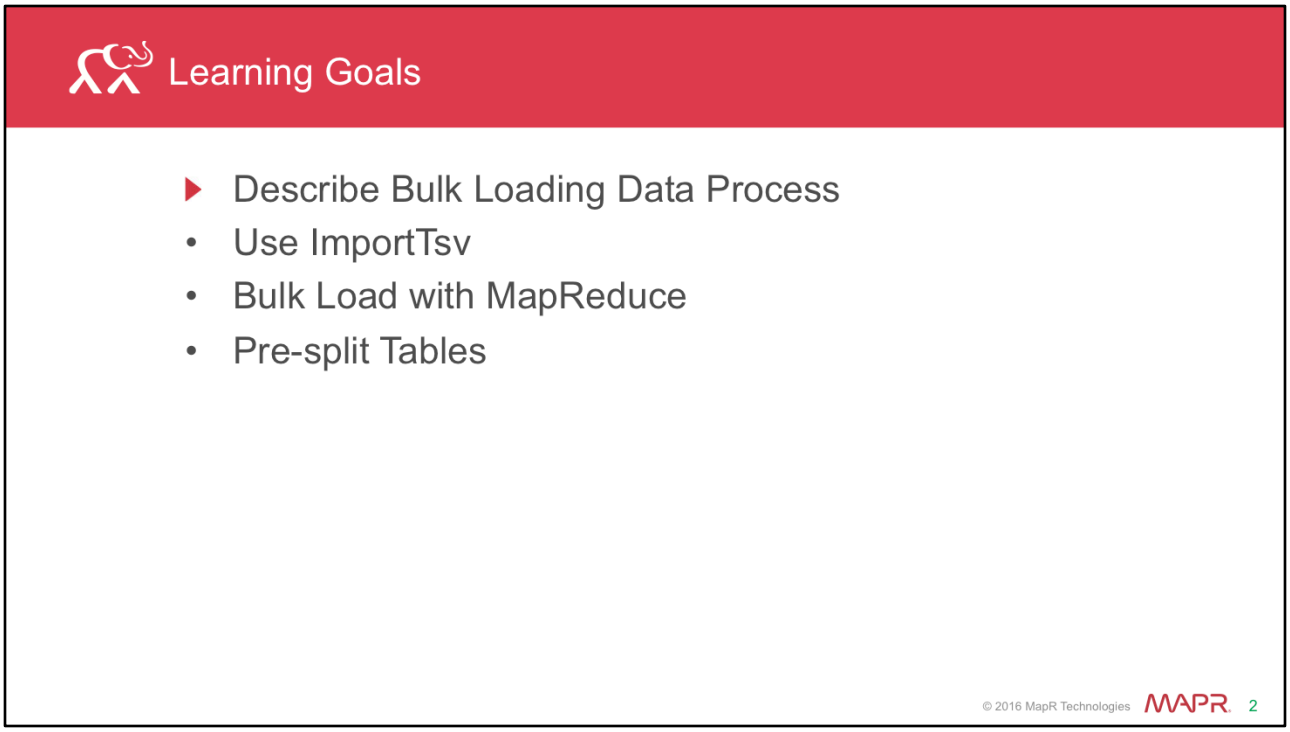
DEV 340 Apache HBase Bulk Load, Performance and Security Slide Guide

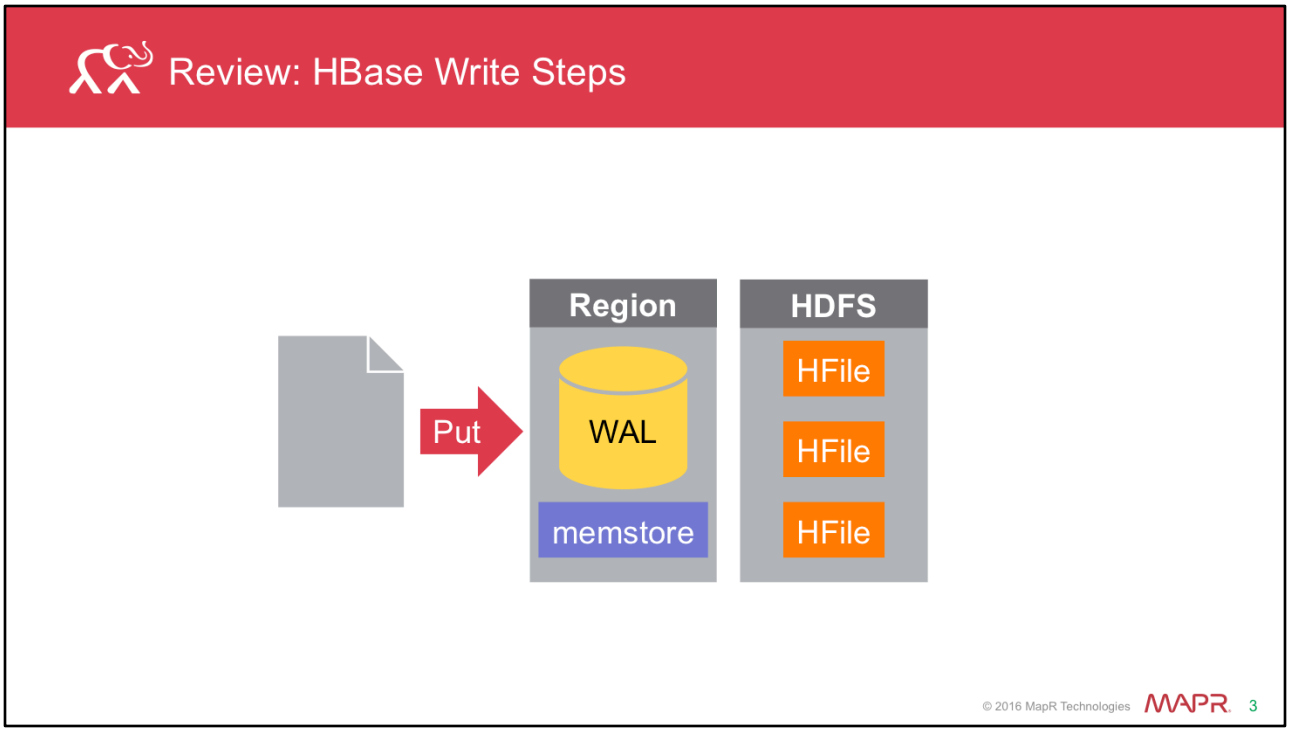
Winter 2016



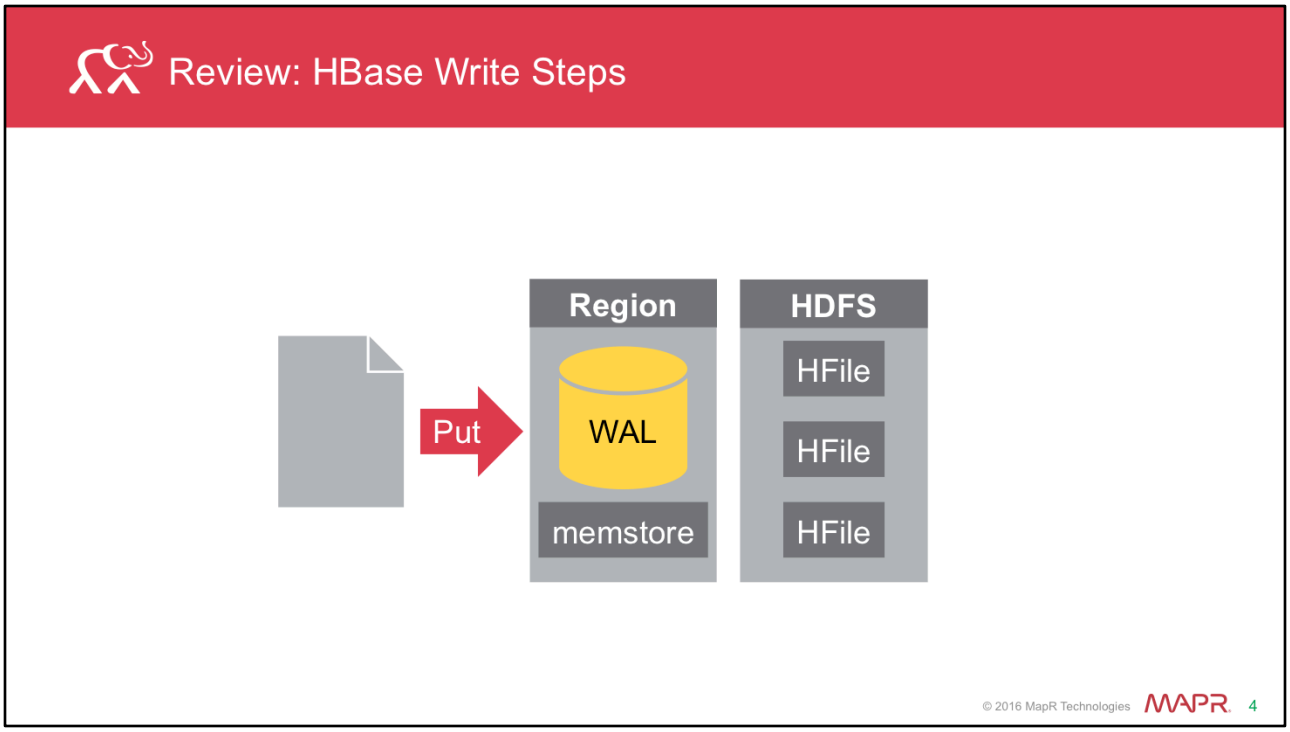
Welcome to DEV 340 Lesson 7, Bulk Loading data into MapR-DB Tables. In this lesson, we are going to discuss several techniques to import data in bulk, with an emphasis on importing from a SQL database.



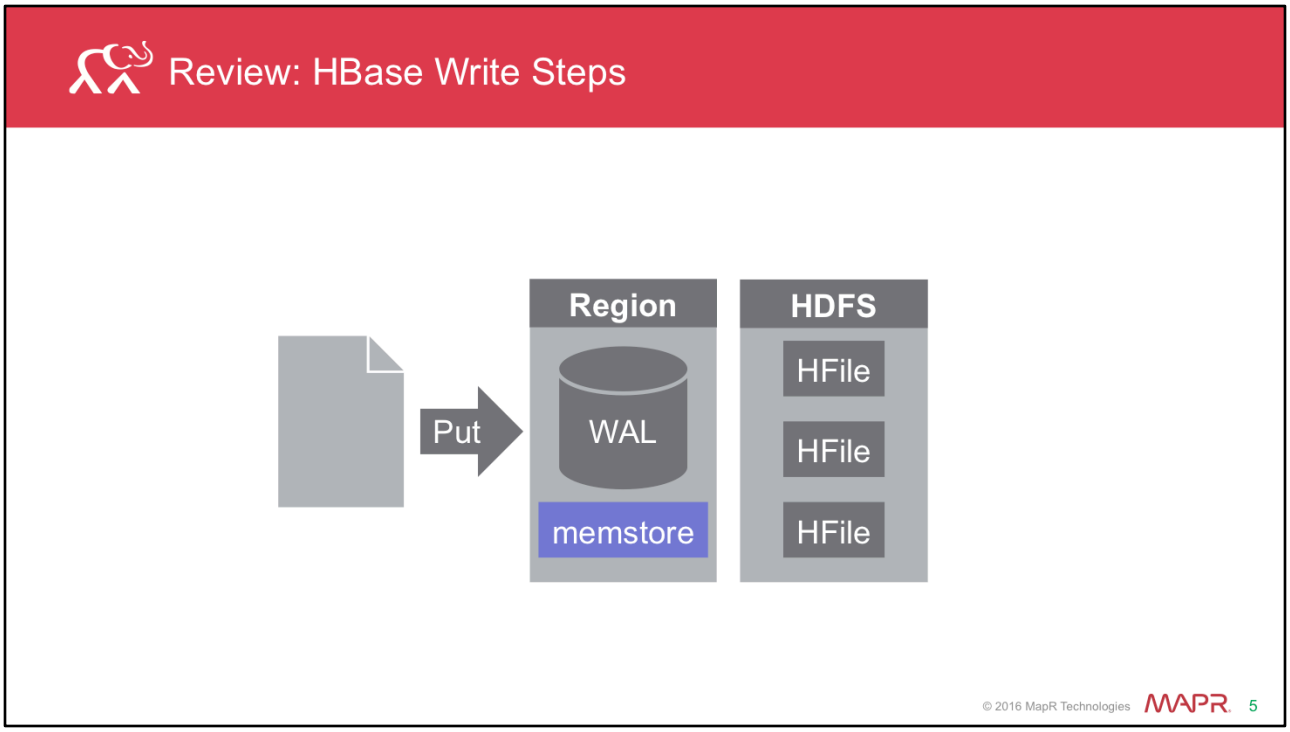
When you have finished with this lesson, you will be able to:
Describe the process to bulk load data into HBase and MapR-DB,
Next we will look at ImportTsv, which is a tool which comes with HBase, and used to import tab separated text files,
We will then revisit MapReduce to look at building custom bulk load tools,
And conclude by looking at when you should consider pre-splitting data before importing it, which is an optimizing technique.



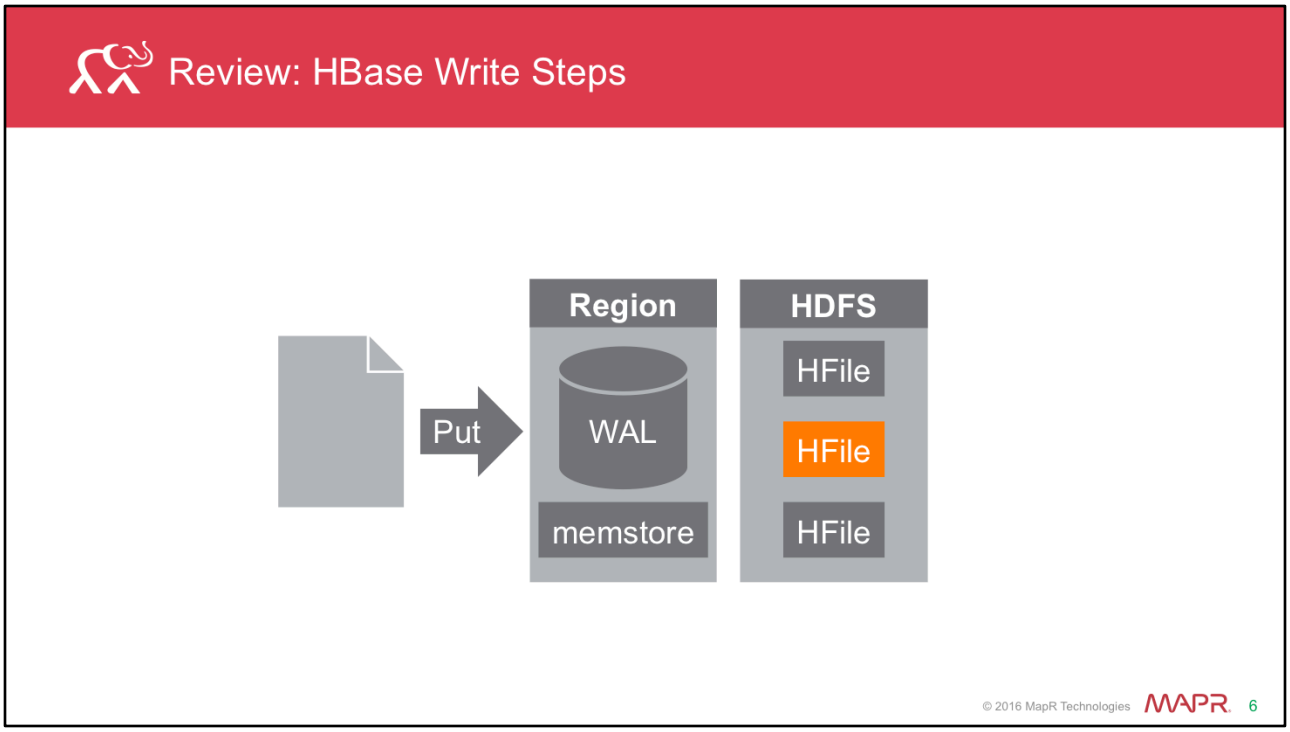
First, let's review the steps to write into an HBase table:



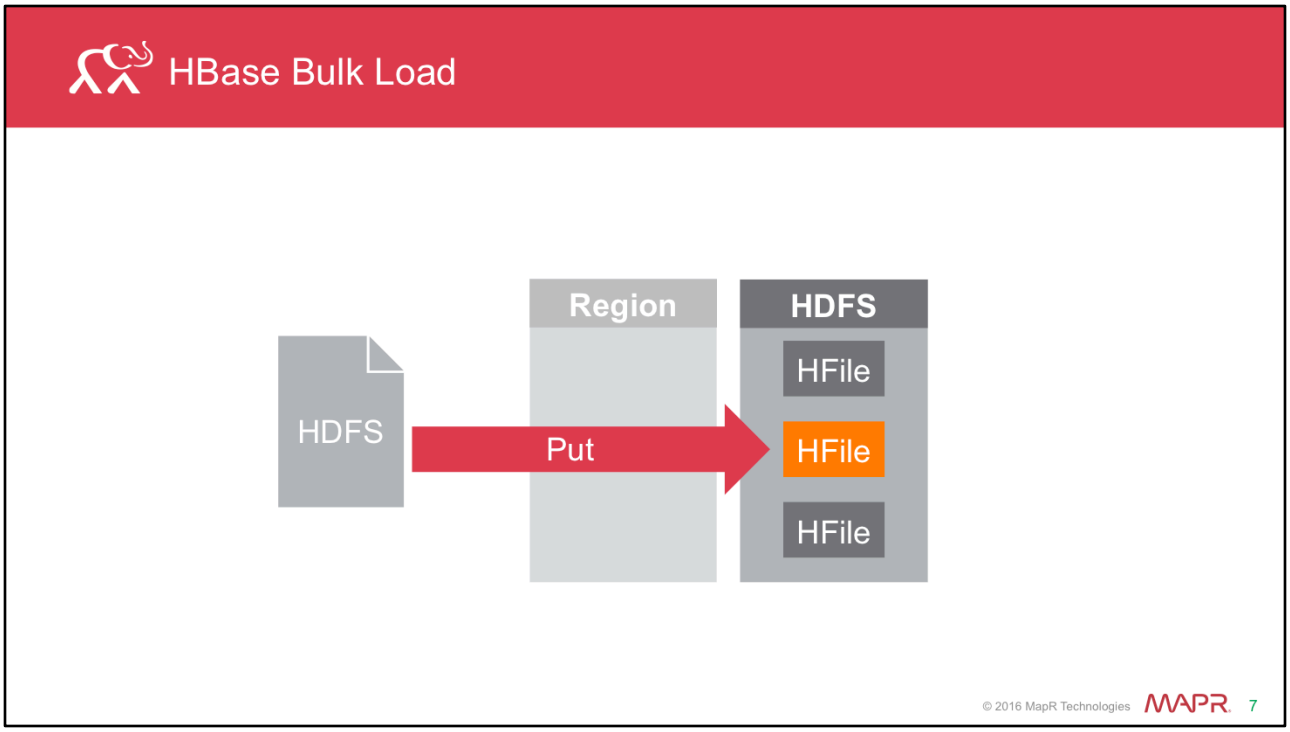
every update writes to the write ahead log on disk, shown here as WAL,



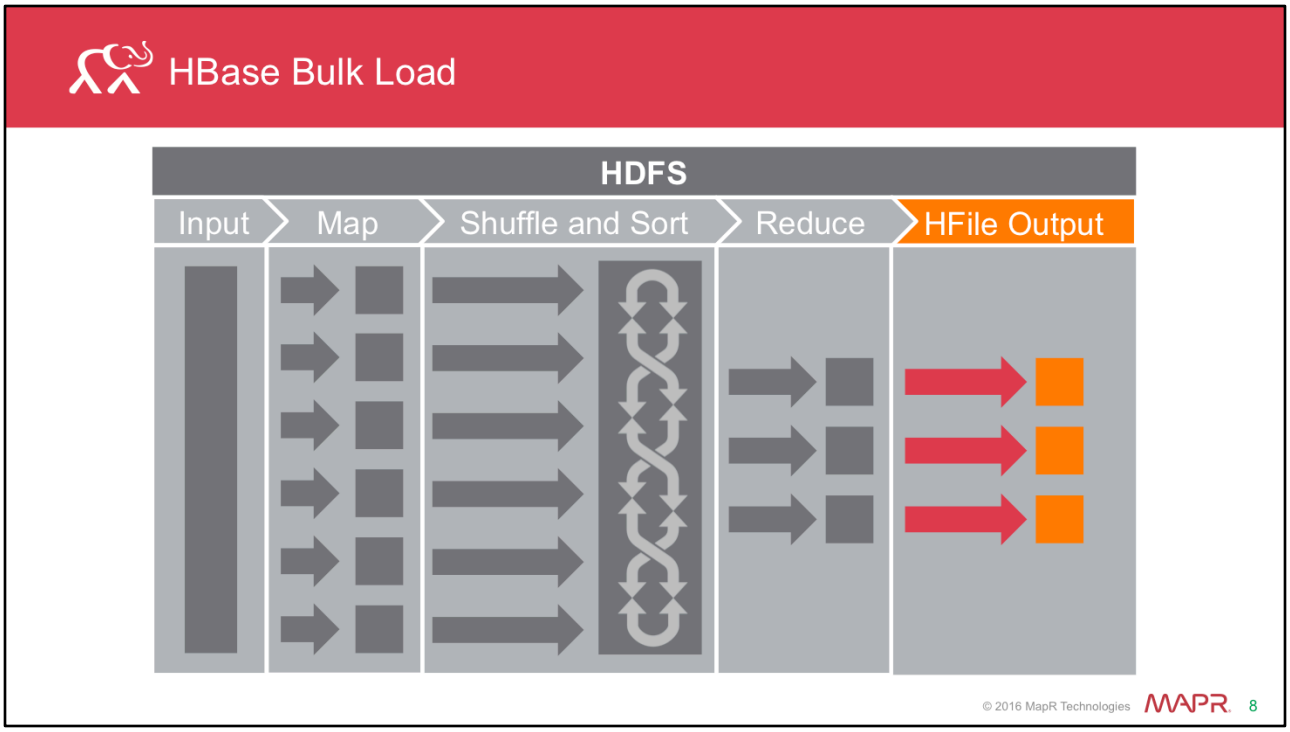
and then to memory, referred to as the memstore.
The memstore stores updates in memory as sorted Key-Value pairs.



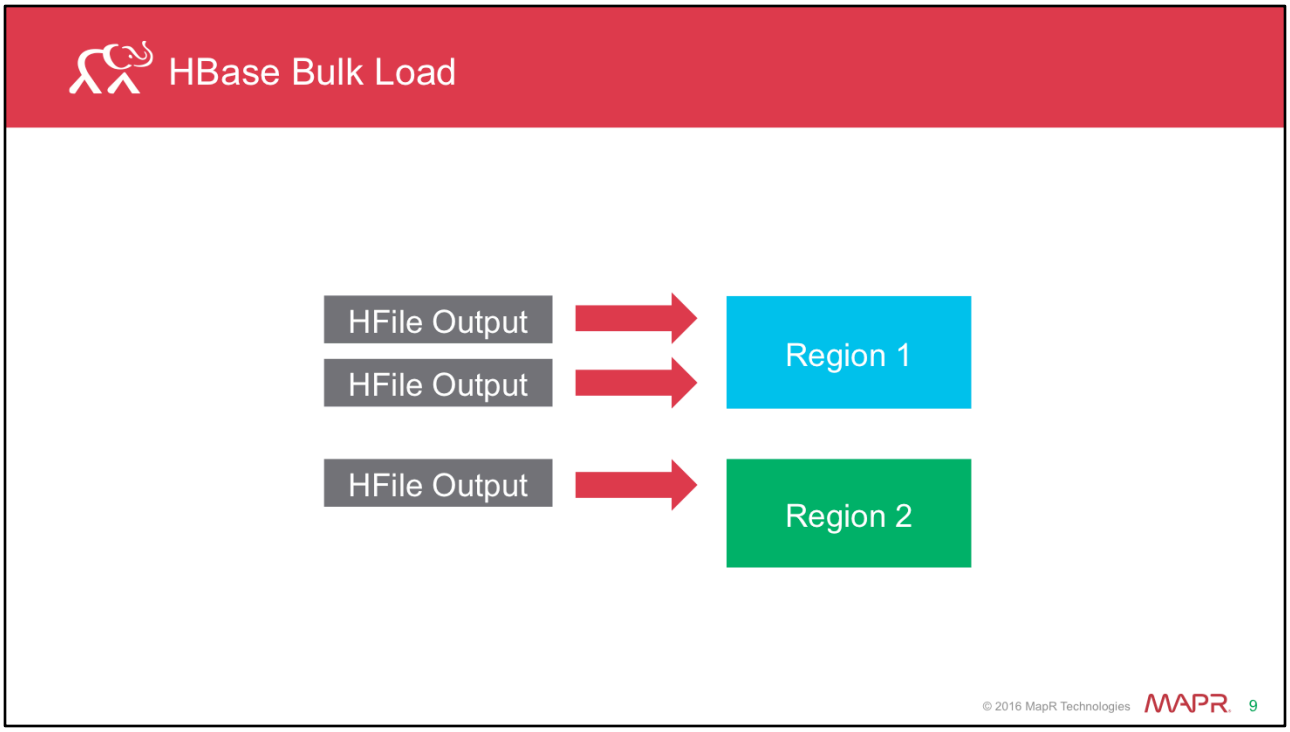
When the memstore is full, the values are flushed to the HDFS file system as an HFile.



You can use HBase bulk load tools such as ImportTsv, or CopyTable to import data from an HDFS file or another HBase table directly into an HBase Hfile, skipping the write ahead log and memstore.



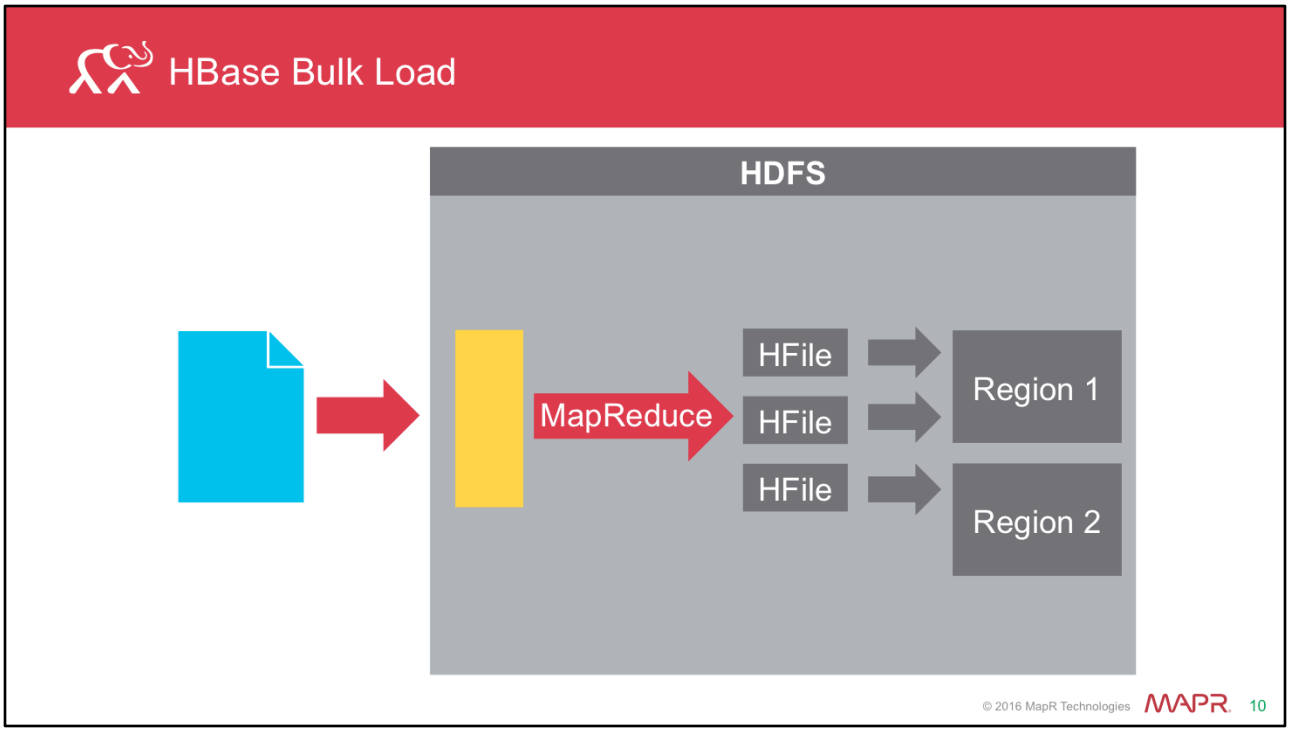
Under the covers these tools use a MapReduce job to convert the raw data into HFile format, which is the HBase internal storage format.



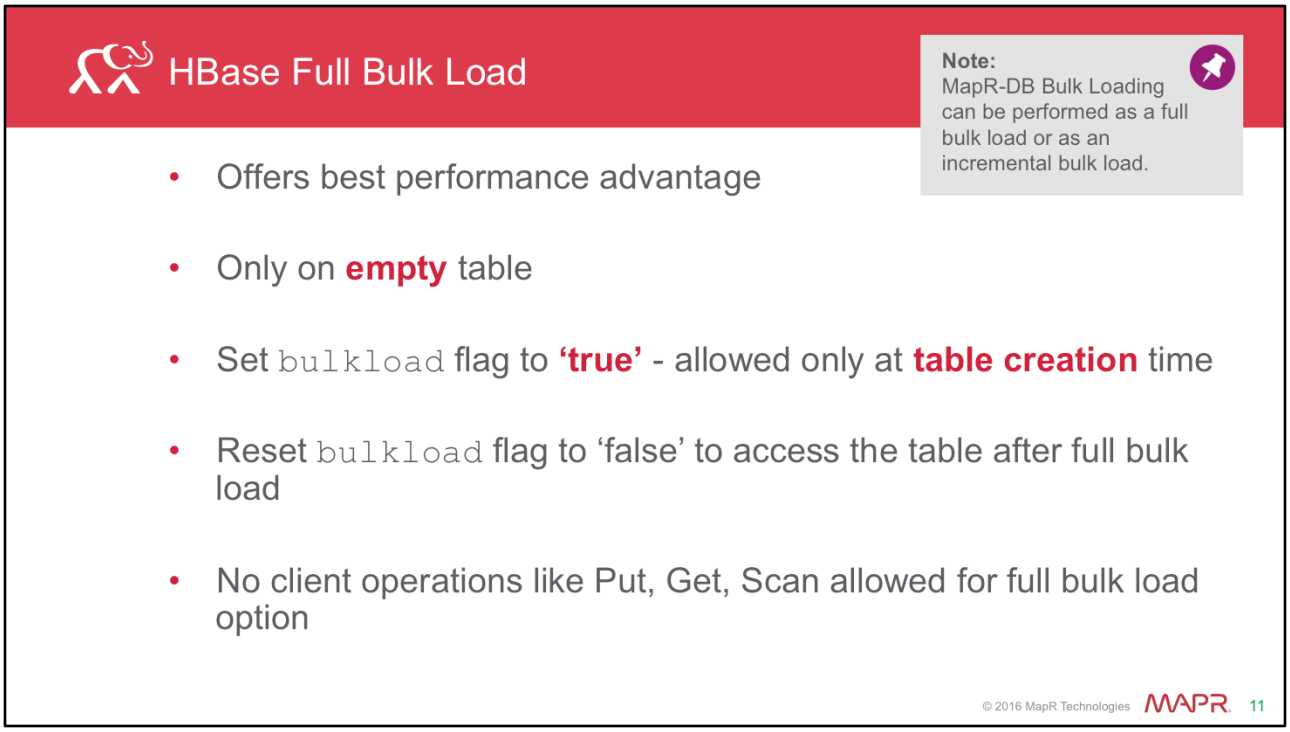
To complete the HBase bulk load, the completebulkload tool is used to load the Hfiles into HBase and tell the Regions Servers where to find them

For more information on the completebulkload toll refer to the HBase documentation.

http://hbase.apache.org/book.html#completebulkload



A full bulk load operation can only be performed on an empty table.
Incremental bulk load operations DO use the write-ahead log.



Bulk loading can be performed as a full bulk load or as an incremental bulk load.
A full bulk load offers the best performance advantage for empty tables.

HBase Incremental Bulk Load

Note:
MapR-DB Bulk Loading can be performed as a full bulk load or as an incremental bulk load.

- Set `bulkload` flag to **'false'**
- Client can access table
- Read existing data and data streamed from incremental bulk load
- Faster than Put API, but slower compared to full bulk load

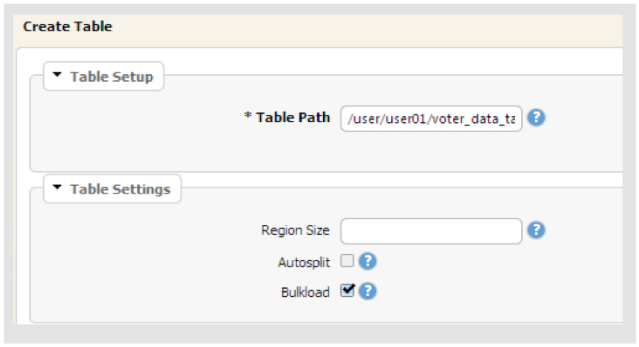
Incremental bulk loads can add data to existing tables concurrently with other table operations.

Create a Table with Bulk Load Support

HBase Shell

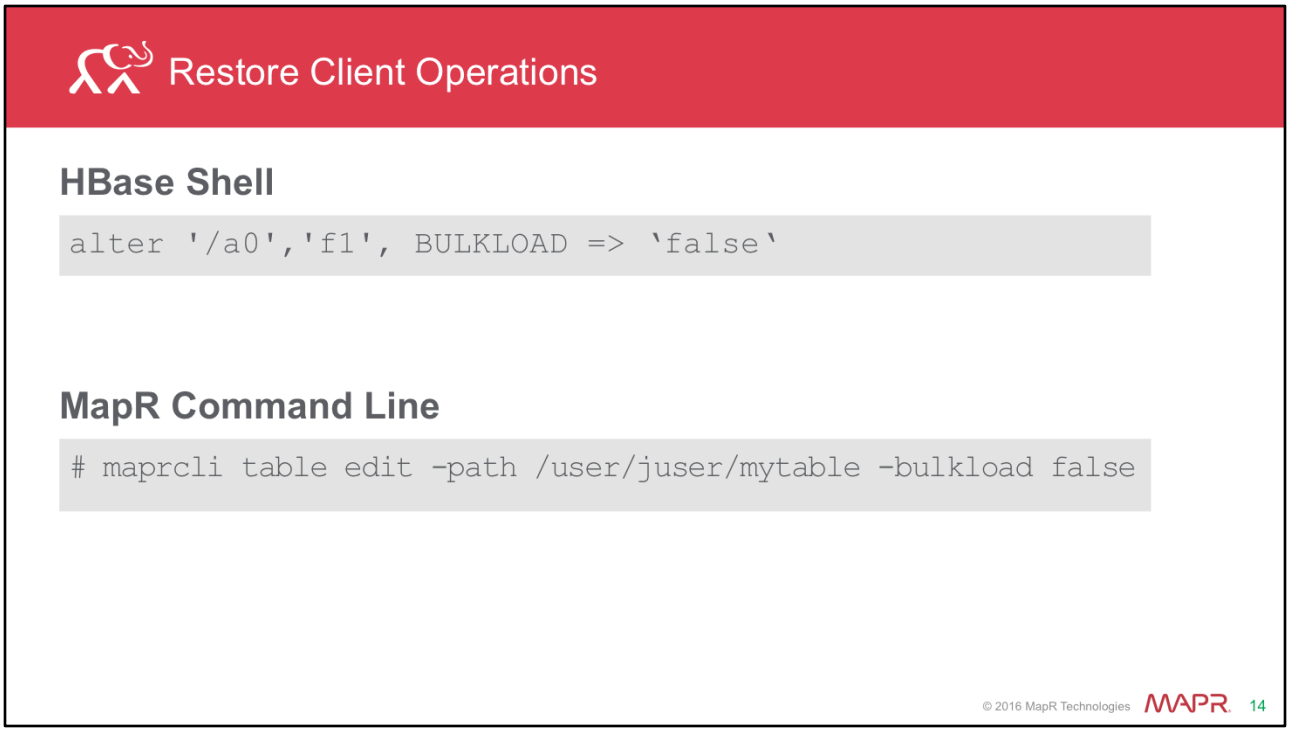
```
hbase>create '/a0','f1', BULKLOAD => 'true'
```

MCS

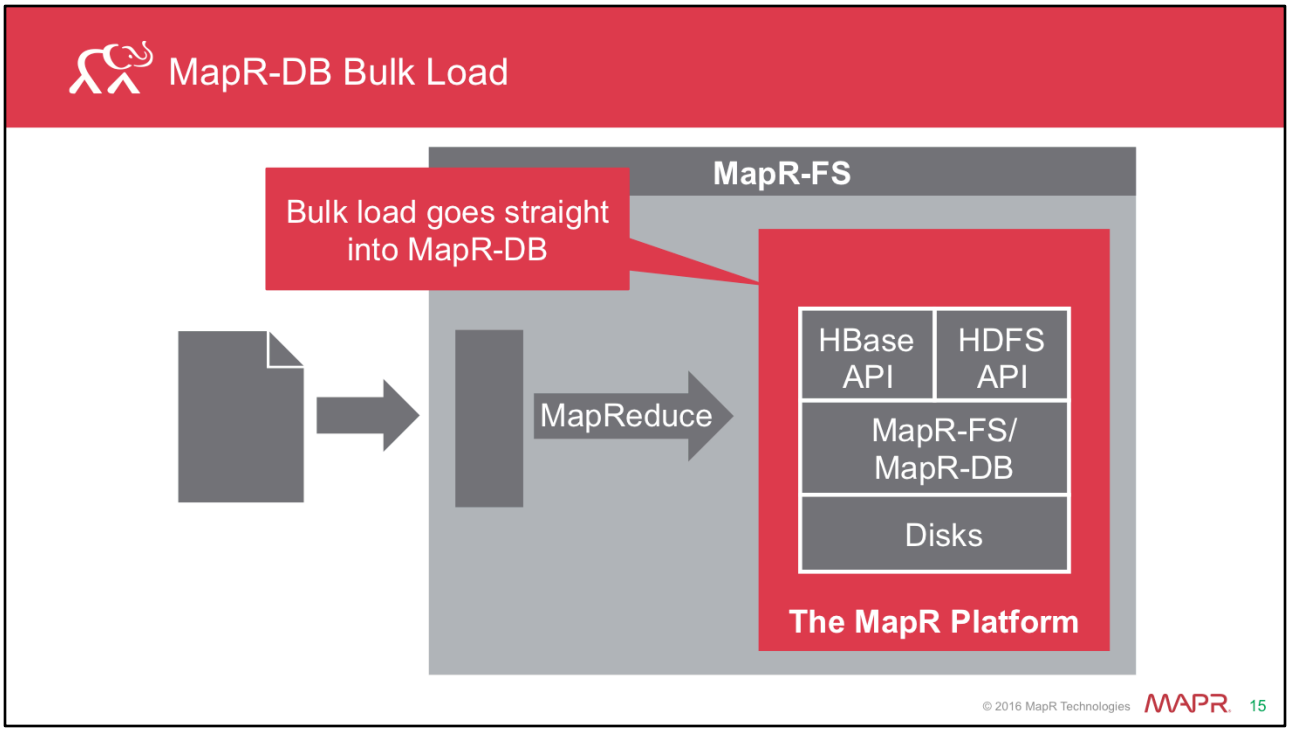


A full bulk load operation can only be performed on an empty table, and you must set the BULKLOAD attribute to true as shown here with the HBase shell. With MapR-DB you can also set the BULKLOAD attribute through the MCS.

For more information about MapR-DB bulk loading with the MCS refer to: http://doc.mapr.com/display/MapR/Bulk+Loading+and+MapR-DB+Tables



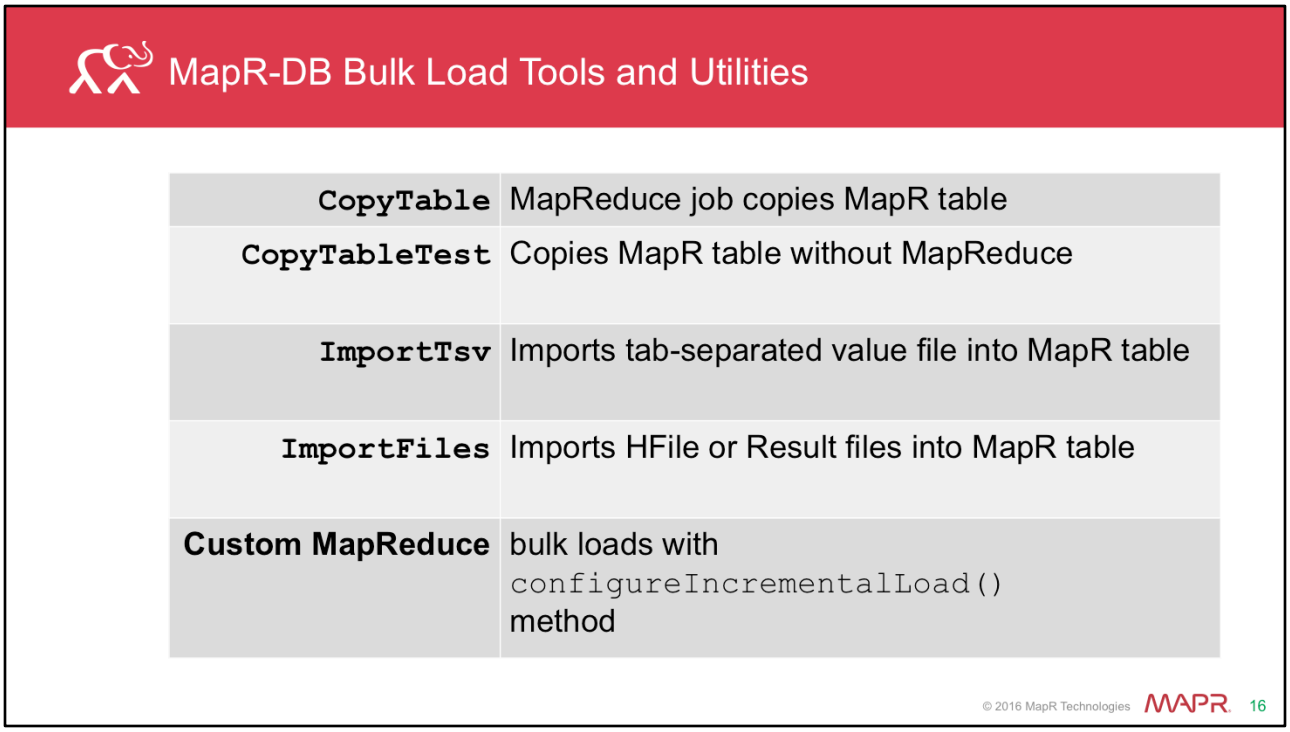
After completing a full bulk load operation, take the table out of bulk load mode to restore normal client operations. You can do this from the HBase shell, the MCS or the maprcli as shown here.



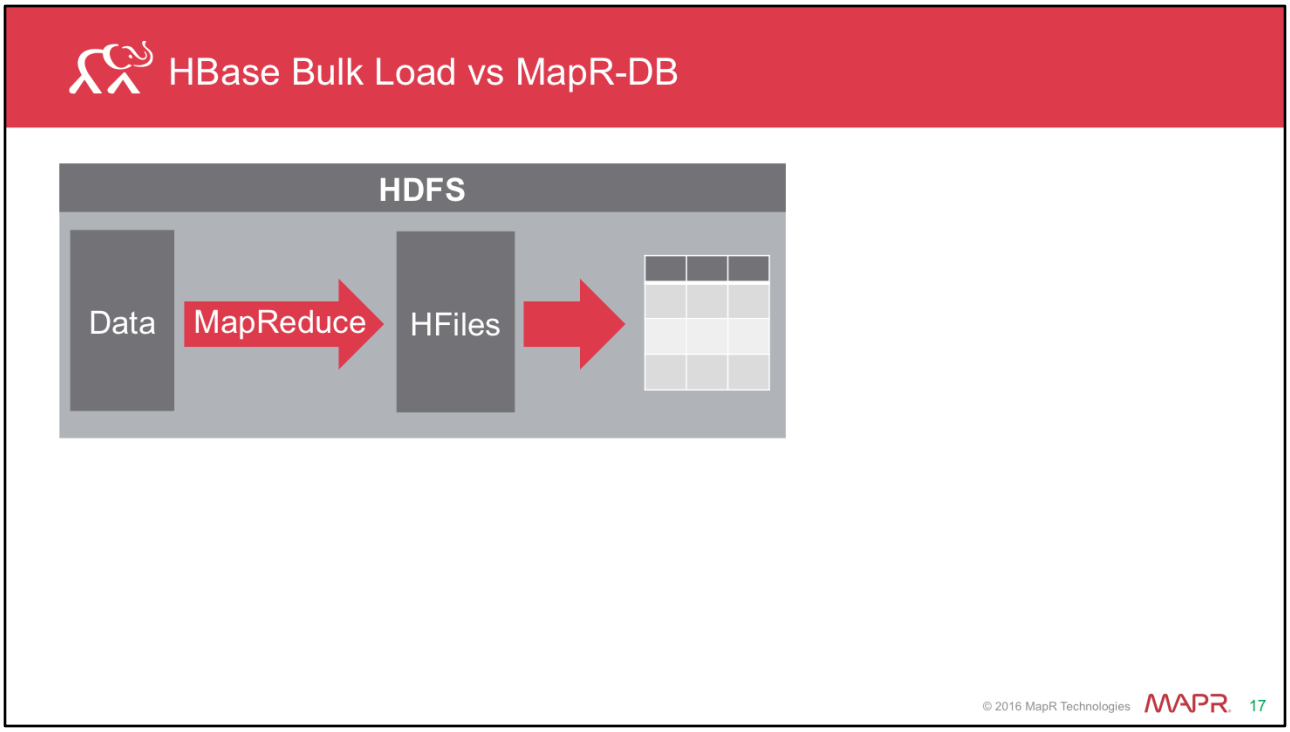
Bulk loading into MapR-DB is similar, but more efficient as it skips the second step in the HBase bulk load process, you do not have to use a tool to load and notify region server of the location of the HFiles.

The MapR-DB implementation integrates table storage into the MapR file system, MapR-DB runs inside of the MapR file system MFS process, which reads from and writes to disks directly.

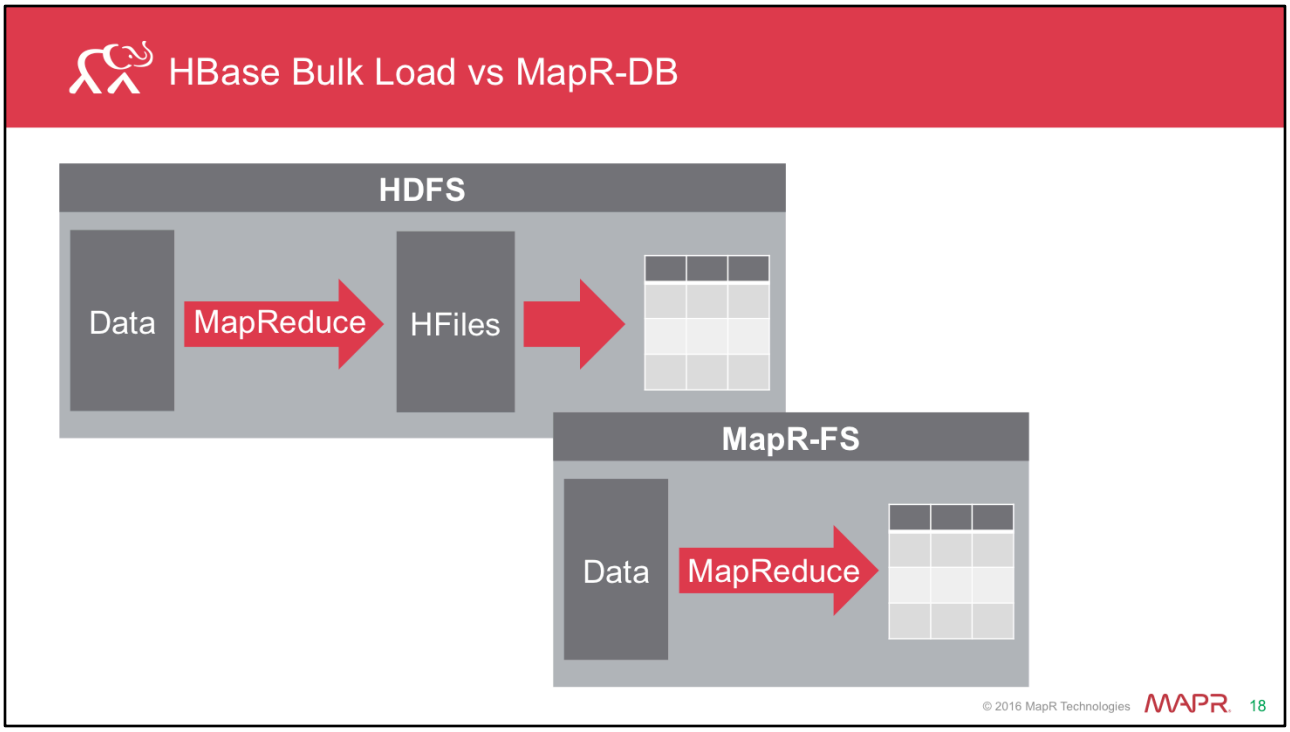
Because MapR-DB tables and files are integrated and guarantee data locality, you can skip the second step. Files do not need to be moved into a storage directory, as with HBase bulk loading.



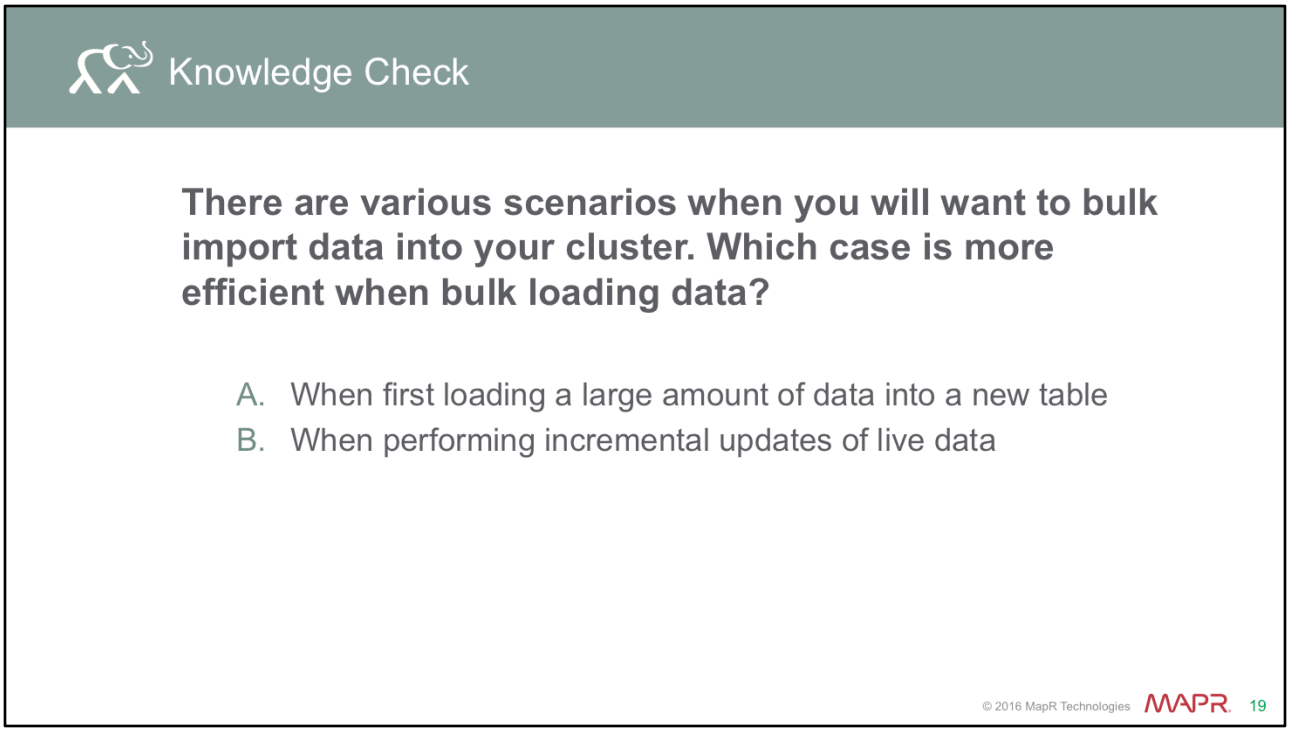
Bulk loading is supported for the following tools, which can be used for both full or incremental bulk load operations:
The CopyTable tool uses a MapReduce job to copy a MapR table.
The CopytableTest tool copies a MapR table without using MapReduce.
The ImportTsv tool imports a tab-separated value file into a MapR table. We will talk more about this tool in the next section.
The ImportFiles tool imports HFile or Result files into a MapR table.
Custom MapReduce jobs can use bulk loads with the configureIncrementalLoad() method from the HFileOutputFormat class.



Apache HBase needs two stages for bulk loading: 1) Generate HFiles and 2) Load HFiles into the table



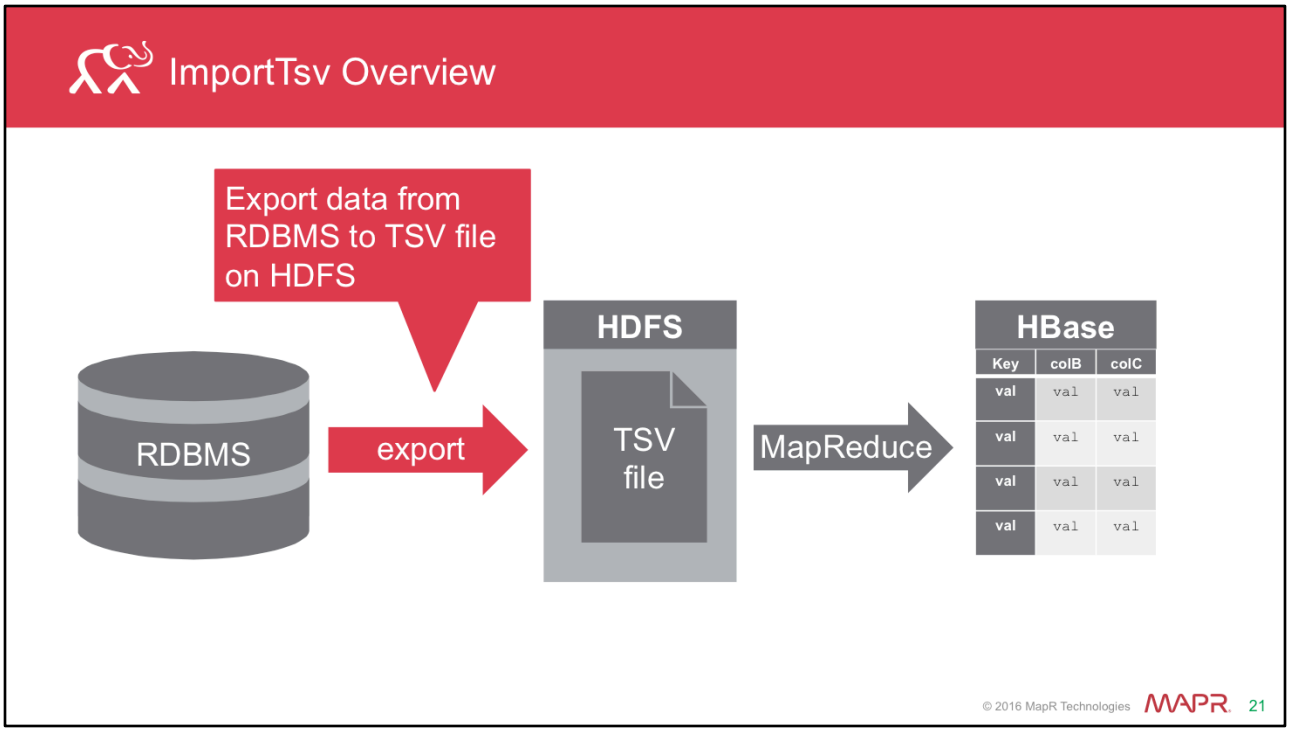
MapR-DB needs only one stage: directly load data into the table.



Bulk loading is more efficient when loading data into a new table. Bulk loading does not write to the WAL, and is therefore faster than a standard put. However, the entire bulk load process must complete before the data is available to process.

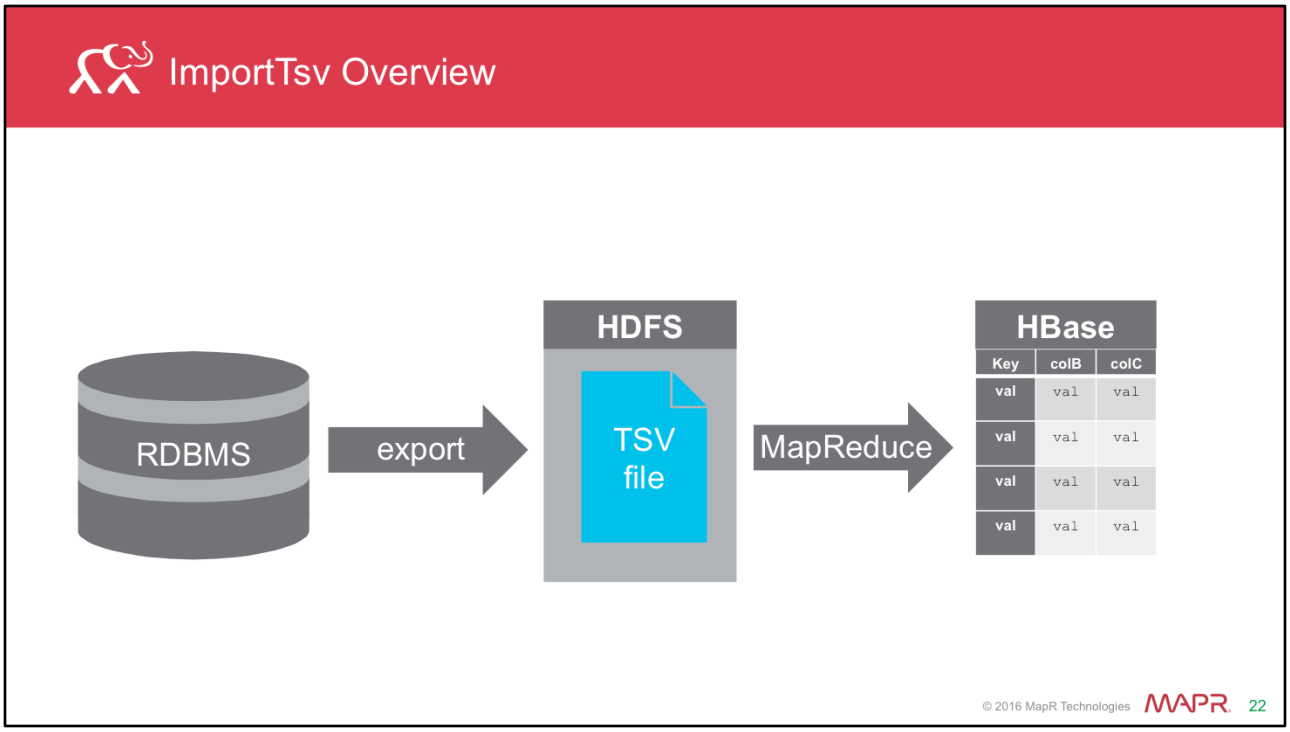
Learning Goals

- Describe Bulk Loading Data Process
- Use ImportTsv
- Bulk Load with MapReduce
- Pre-split Tables

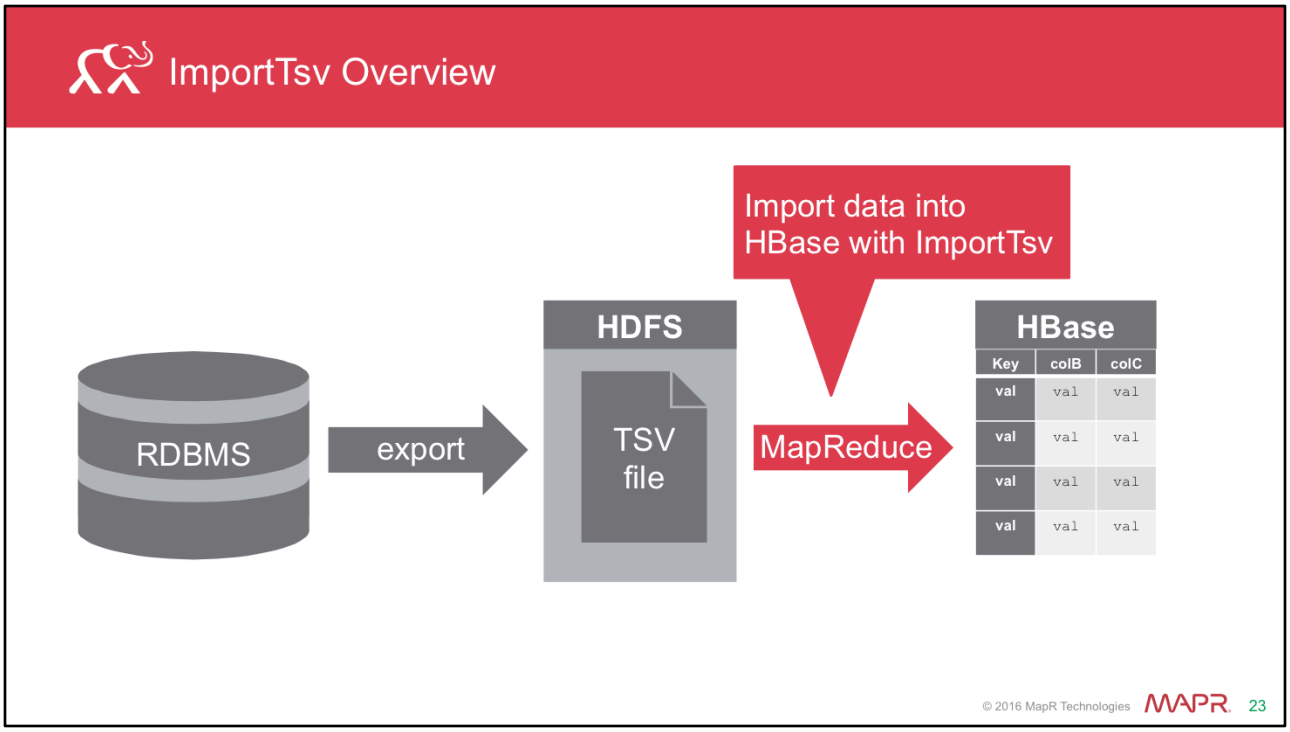


ImportTsv is utility provided by the HBase package, and is a very efficient tool to load text data into MapR-DB or HBase

- First, export the data from the RDBMS to a TSV file, using your standard tools. This is faster than executing SQL on RDBMS for large amounts of data.

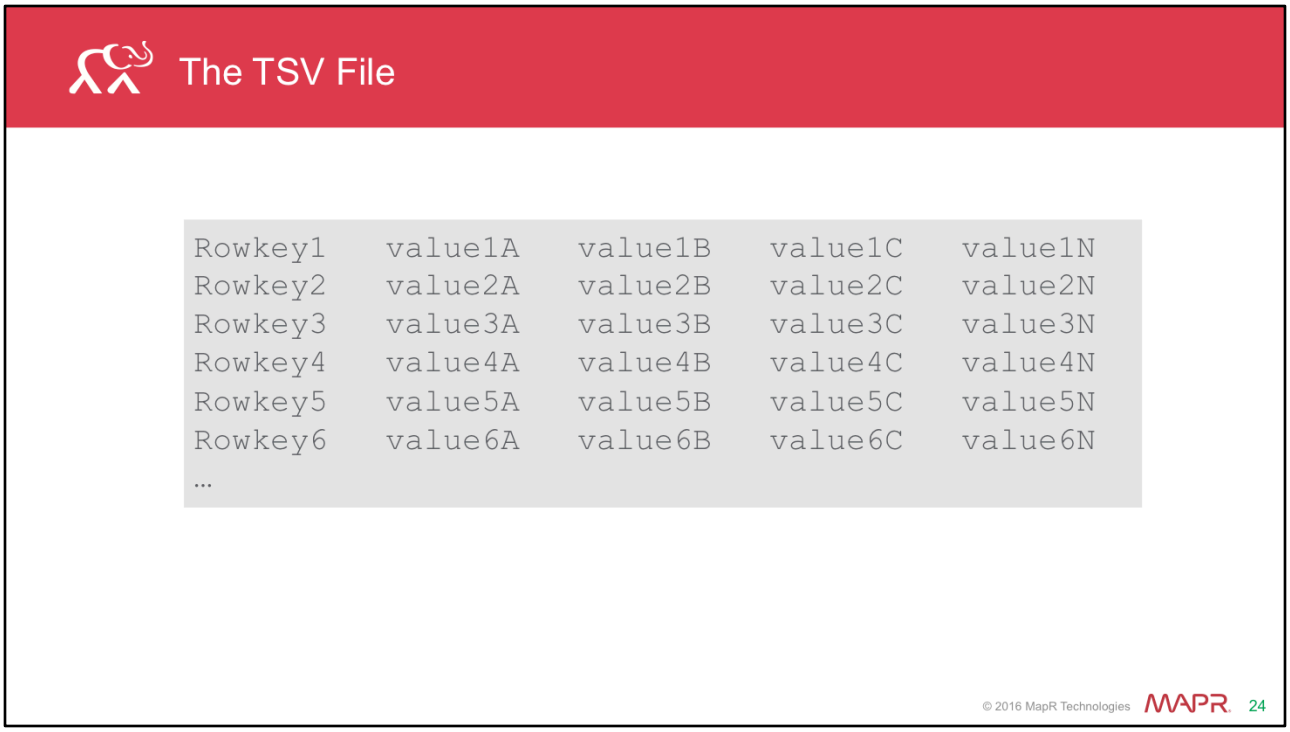


- Load the TSV file into your cluster file system.

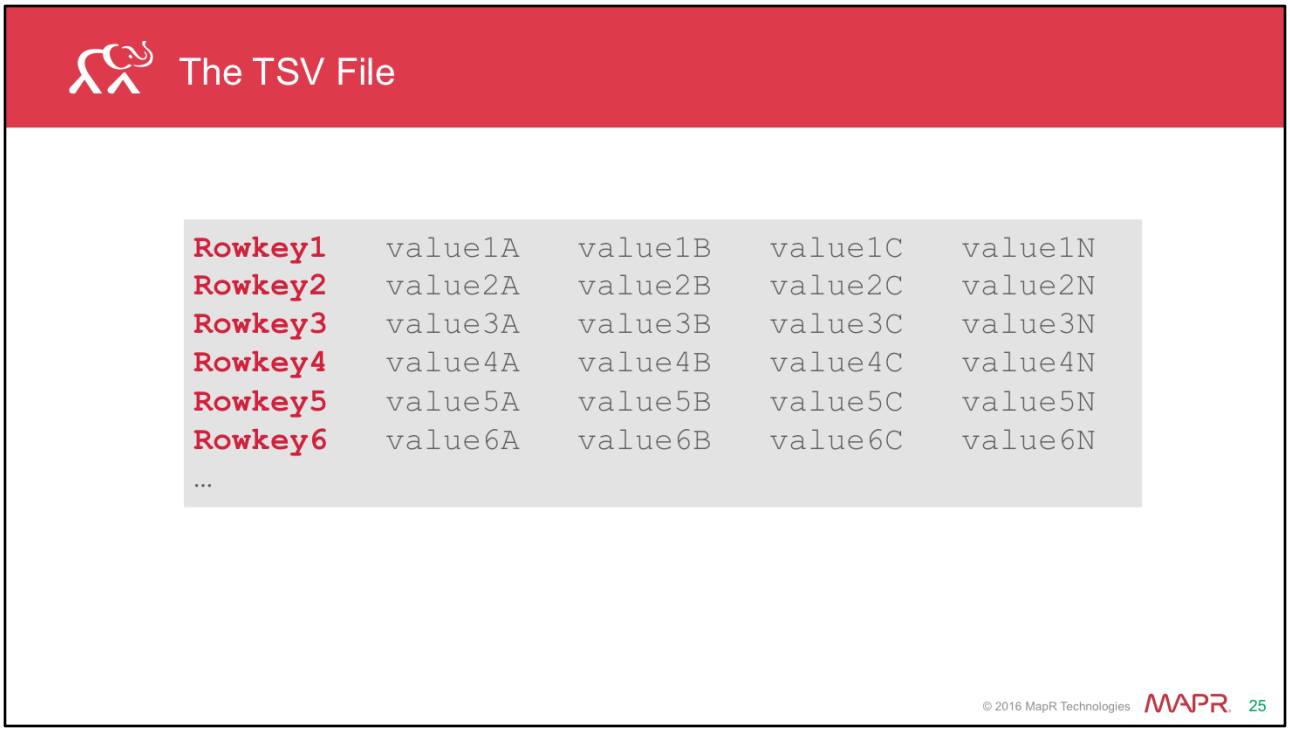


- Then run ImportTsv on the TSV file. ImportTsv will launch a MapReduce job to perform the import.
- Faster than executing SQL on RDBMS, for large amounts of data

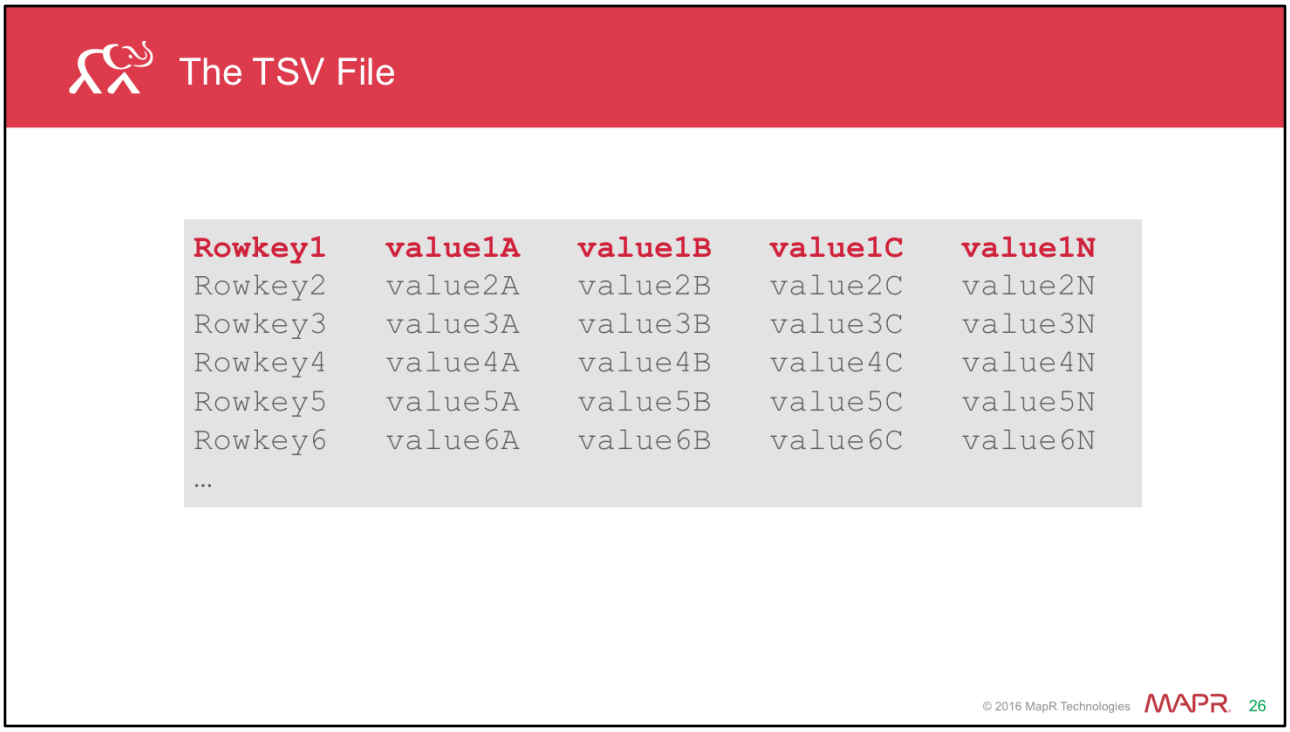
Efficient tool to import TSV files into MapR-DB or HBase.



The TSV file consists of tab-separated data.



The file must contain a field representing the row key of the HBase table row.



Typically, the format of a TSV file is: rowkey \tab value1 \tab value2, and so on, up to valueN.

Running ImportTsv: Set Up

RowKey	Address			Order			
	Street	City	State	Date	ItemNo	Ship Address	Cost

Before we start the ImportTsv MapReduce job, we need to create the table that we want to import the file into.

We will also need to create the column families, and set the column family properties for our schema.

With these set up tasks done, we are ready to run the tool.

Running ImportTsv: Arguments

```
Input data:  rowkey, value1, value2, …  valueN
```

```
$hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \
-Dimporttsv.columns=HBASE_ROW_KEY,family:name1, \
...,family:nameN \
-Dimporttsv.bulk.output=/user/userXX/dummy \
target_table_name \
/user/joe/inputDirectory/data.tsv
```


The row keys and column names corresponding to the TSV data must be specified using the -Dimporttsv.columns option.

You specify this as shown in the format columnfamilyname:columnname separated by columns.

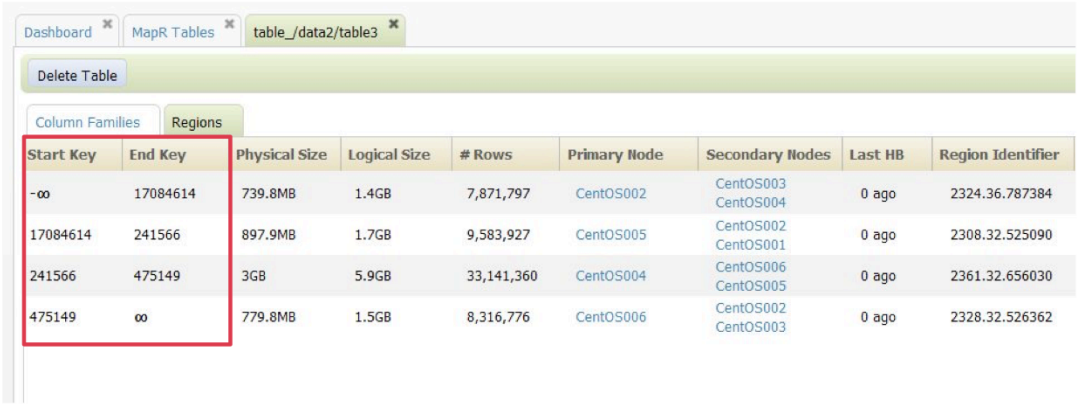
Use HBASE_ROW_KEY to designate which commar separated value should be used as the row key for each imported record.

You must specify one column to be the row key, and specify a column name for every column in the input data.

After the -Dimporttsv.columns parameter, you specify the table name and the input TSV file path

The tool starts a MapReduce job which will process the TSV files under the specified input directory. The MapReduce jobs are executed in parallel on multiple servers, so it is much faster than loading data from a single client.

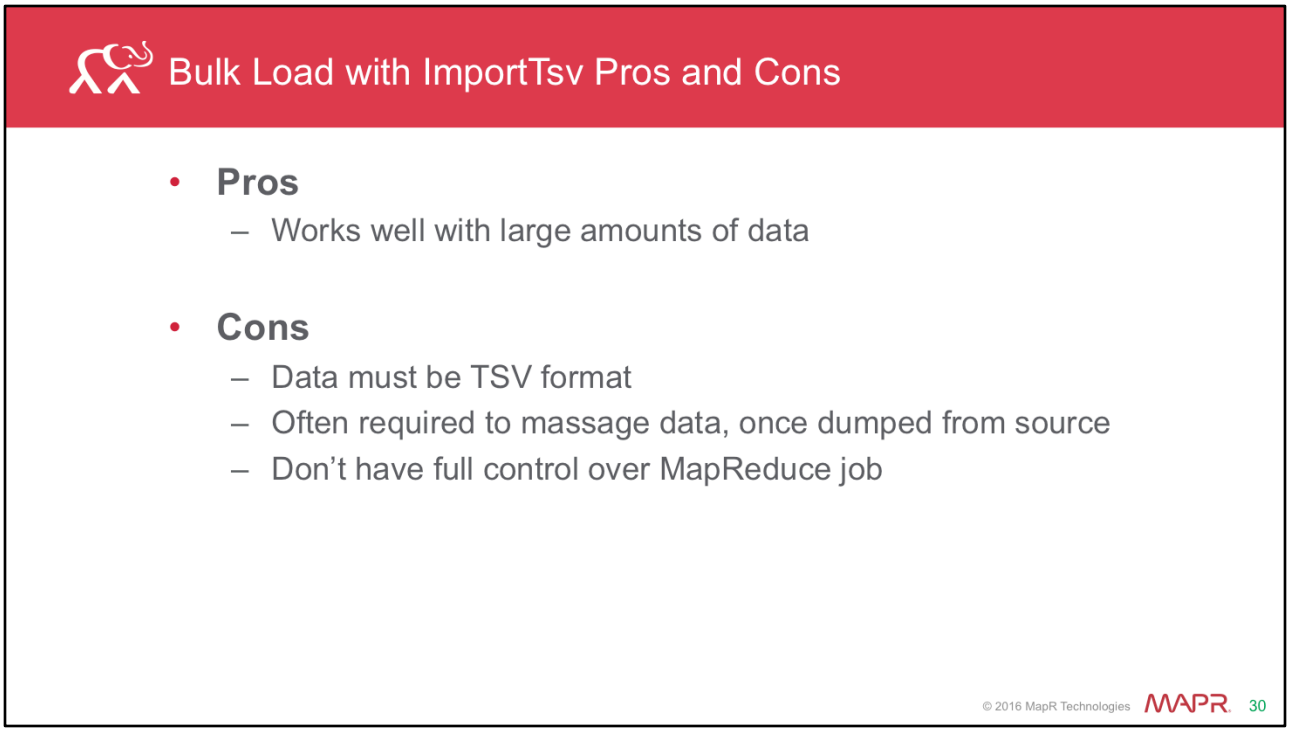
Monitor with the MCS



Start Key	End Key	Physical Size	Logical Size	# Rows	Primary Node	Secondary Nodes	Last HB	Region Identifier
-∞	17084614	739.8MB	1.4GB	7,871,797	CentOS002	CentOS003 CentOS004	0 ago	2324.36.787384
17084614	241566	897.9MB	1.7GB	9,583,927	CentOS005	CentOS002 CentOS001	0 ago	2308.32.525090
241566	475149	3GB	5.9GB	33,141,360	CentOS004	CentOS006 CentOS005	0 ago	2361.32.656030
475149	∞	779.8MB	1.5GB	8,316,776	CentOS006	CentOS002 CentOS003	0 ago	2328.32.526362

Once the job is started, we can monitor it via the web UI.

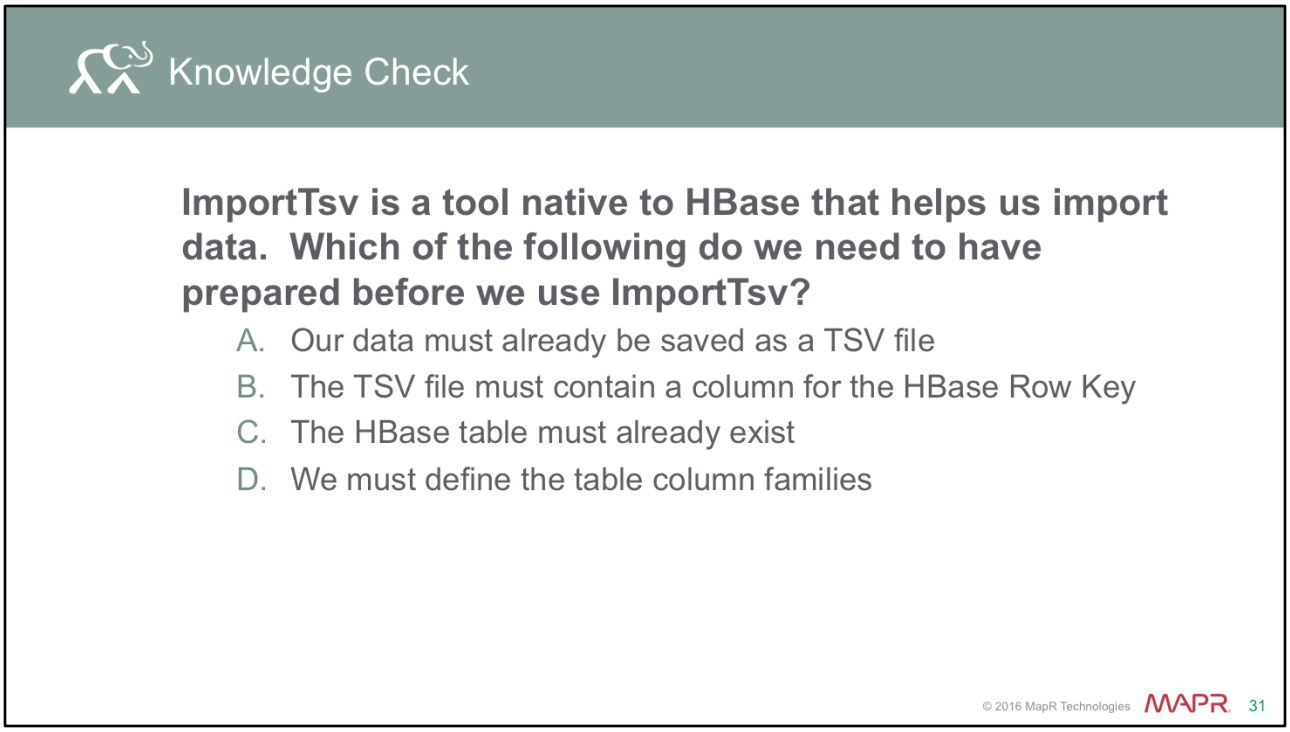
While the import job is running, you can see the regions being created and split in the MCS.



This process of using ImportTsv works well when you are bulk loading large amounts of data, as it is using the MapReduce framework to process the input file into many tasks running in parallel across nodes on your cluster.

Some limits to using ImportTsv, however, are that the data must first be in TSV format. This usually requires massaging the data before it can be imported into HBase.

ImportTsv runs its own, pre-written MapReduce job. The next section will show how to write custom a MapReduce job, for needs beyond what ImportTsv can handle.



All of these are true.

Learning Goals

- Describe Bulk Loading Data Process
- Use ImportTsv
- Bulk Load with MapReduce
- Pre-split Tables

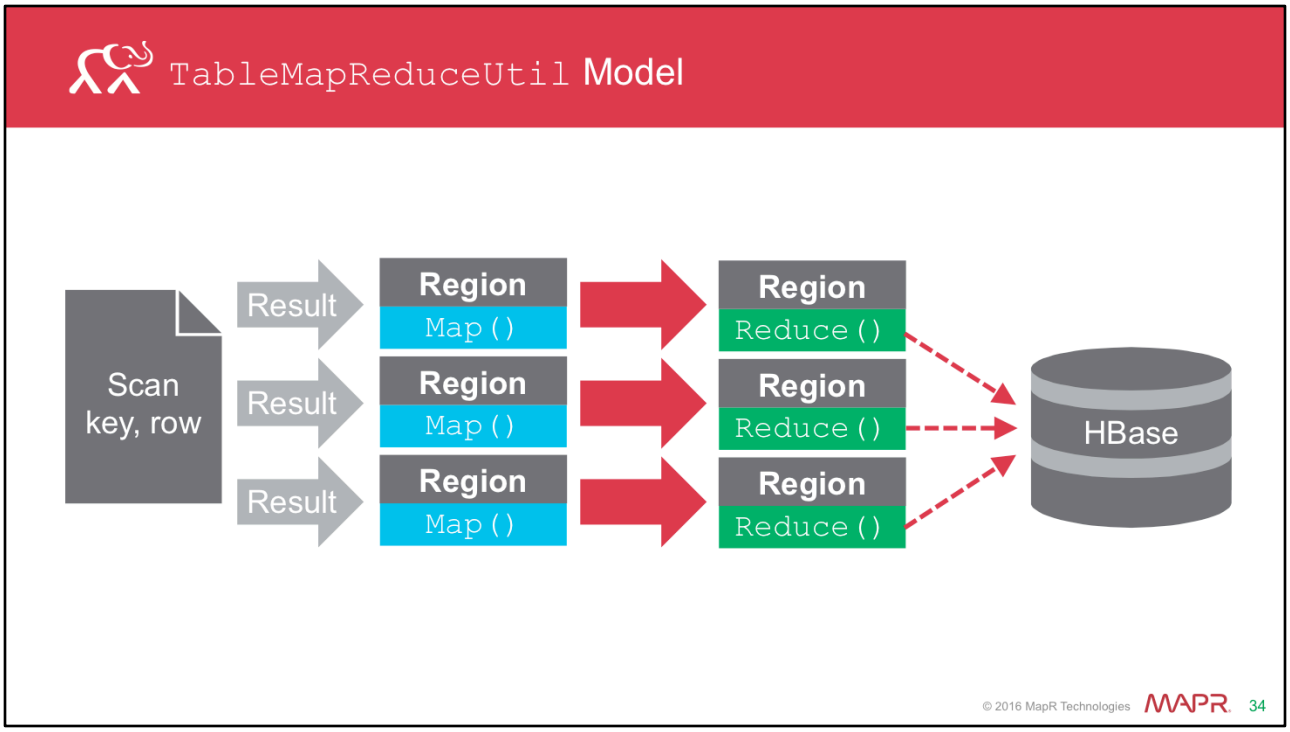
Custom Bulk Load MapReduce

Writing your own MapReduce job helps when importing huge amounts of data.

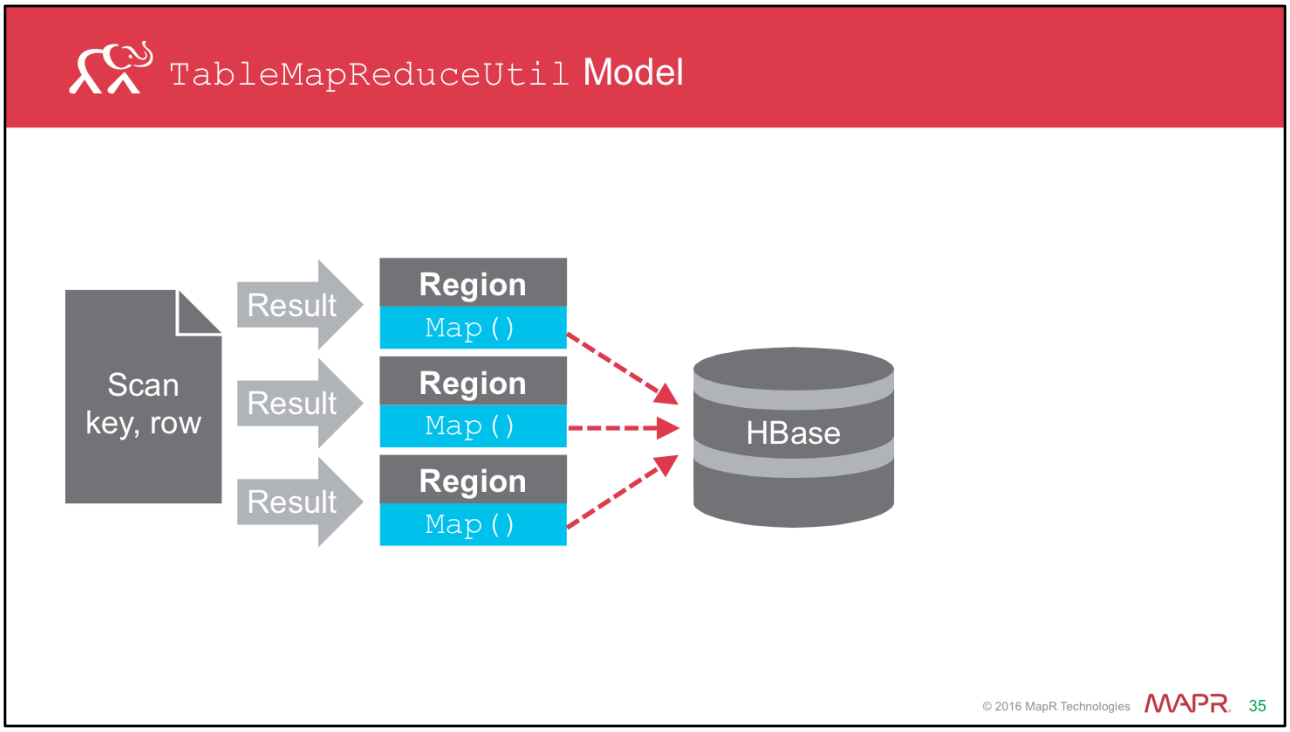
Ex. Ingesting log files to HBase or MapR tables

 33

When we use the ImportTsv utility to bulk load data from our HDFS or MapR-FS file system, it runs a MapReduce job to load the data into the table. If ImportTsv does not meet all of your requirements, we can write a custom MapReduce job to perform bulk loading.



HBase provides the TableOutputFormat class for writing data into an HBase table from a MapReduce job. The TableMapReduceUtil uses the TableOutputFormat as the outputFormat class.



When writing to an HBase table from a MapReducejob with TableOutputFormat you can skip the Reducer step and emit Puts from the Mapper. The graphic shown here illustrates the programming and execution model. Note you only have to write the Mapper code. The rest of it is handled by the TableMapReduceUtil class and the underlying Hadoop framework.

MapReduce Setup

```
public static Job createJob(Configuration conf, String[] args) throws Exception {
    . . .
    Job job = new Job(conf, "mapreduce_import");
    job.setMapperClass(ImportMapper.class);
    . . .

   // Insert into table directly using bulk import
   HTable table = new HTable(jobConf, tableName);
   HFileOutputFormat.configureIncrementalLoad(mrJob, table);
   TableMapReduceUtil.initTableReducerJob(
      tableName,
      null, // reducer class is null
      job);
    return job;
}
```

 36

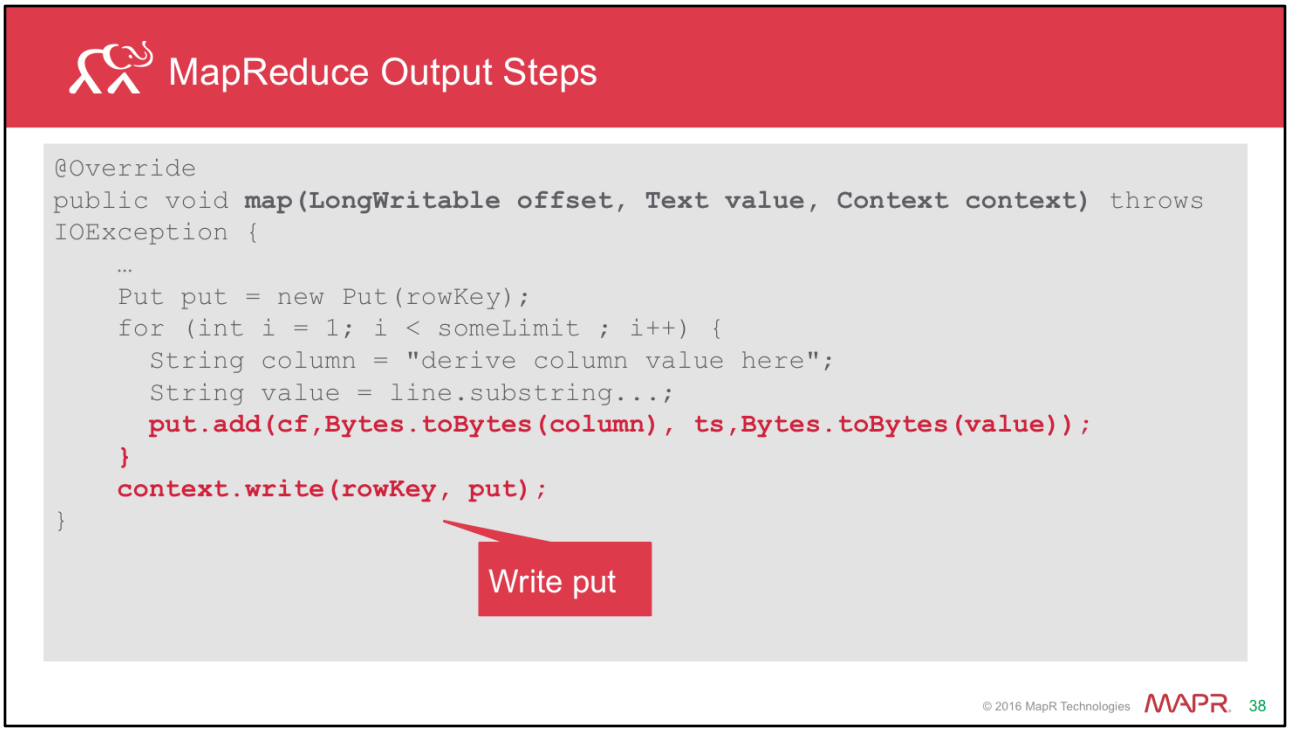
Custom MapReduce jobs can use bulk loads with the configureIncrementalLoad() method from the HFileOutputFormat class.

MapReduce Setup

```
public static Job createJob(Configuration conf, String[] args) throws Exception {
    . . .
    Job job = new Job(conf, "mapreduce_import");
    job.setMapperClass(ImportMapper.class);
    . . .

   // Insert into table directly using bulk import
   HTable table = new HTable(jobConf, tableName);
   HFileOutputFormat.configureIncrementalLoad(mrJob, table);
   TableMapReduceUtil.initTableReducerJob(
      tableName,
      null, // reducer class is null
      job);
    return job;
}
```


In this example, the Reducer class is null because there isn't actually a Reducer step.

Remember when Puts are being emitted from the Mapper, you can skip the Reducer step. Since the framework is already sorting the key values from the Mapper.

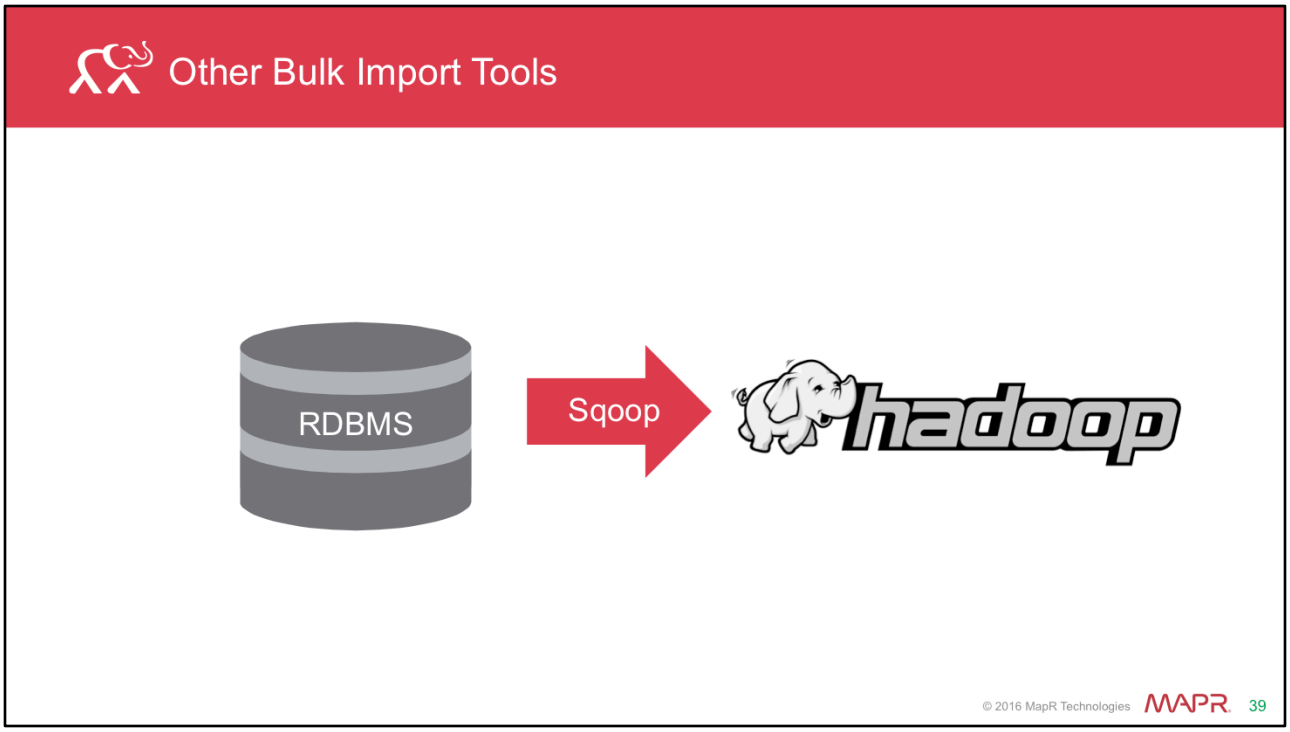


This code is similar to what we covered earlier when we first looked at MapReduce, except the put is being emitted from the Mapper instead of the Reducer.

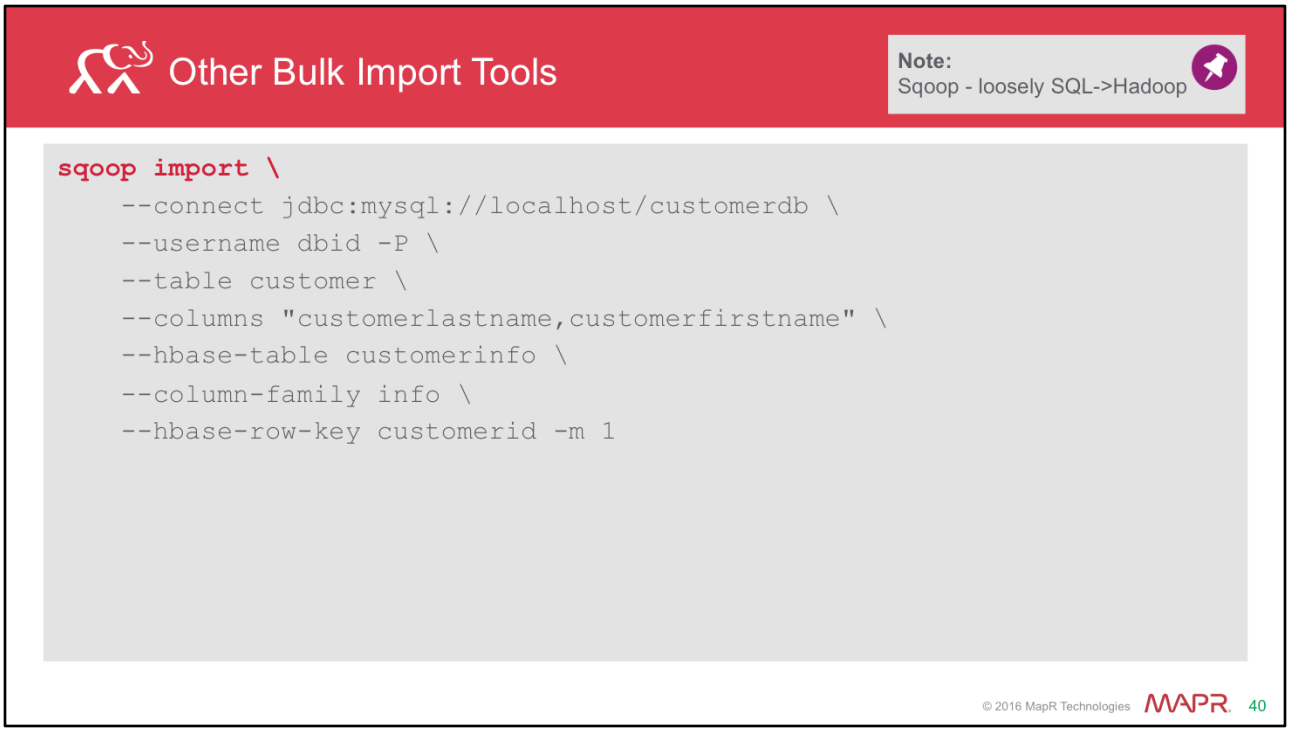
Here we assume that the records are line oriented so extracting the data consists in calling substring on each line to retrieve elements that go into the row key, the column names and values.

The rowkey, column names, and values, constructed from the input, are used to create the put object, then the put object is emitted with the row key.

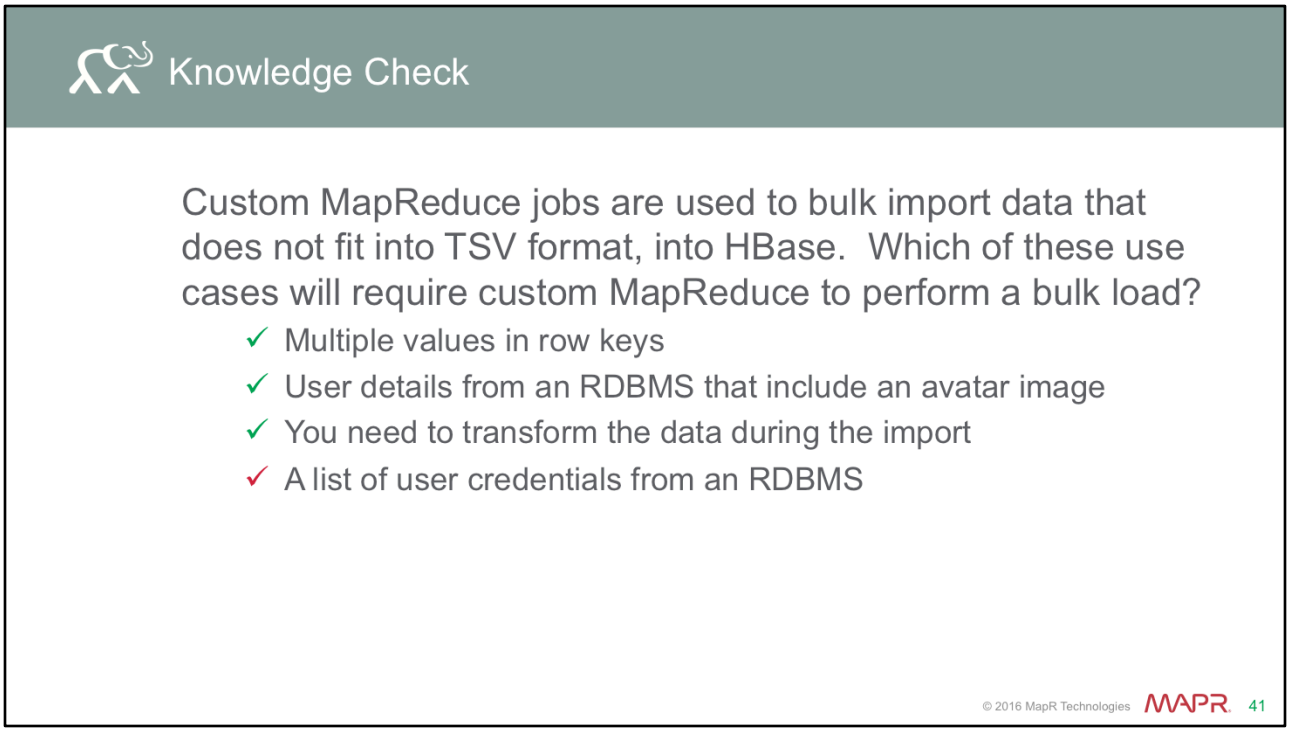
The TableOutputFormat class from the framework will take care of sending the Put to the target table.



Sqoop is a tool which can be used to import data from an RDBMS into HBase.



Here is an example importing from a customer table in a relational database into a customer table in HBase



Multiple values in row keys :: This is true! Bulk importing data that has multiple values in the row keys will require custom MapReduce
User details that include an avatar image :: This is true! Bulk importing data that is not in a string format requires custom MapReduce
You need to transform the data during the import :: This is true! Bulk importing data that needs to be transformed during the import will require custom MapReduce
A list of user credentials from an RDBMS :: This is false. Text data from an RDBMS can be exported as a TSV file, and bulk imported with the ImportTsv utility.

Discussion: When might you use custom bulk loading?

Some common uses for making a custom bulk load MapReduce program are when:

- You want multiple values in row keys
- Your data does not use a string format
- You need to transform the data during the import

Explain how these circumstances might be important to you?

What other uses might you have for writing a custom bulk load application?

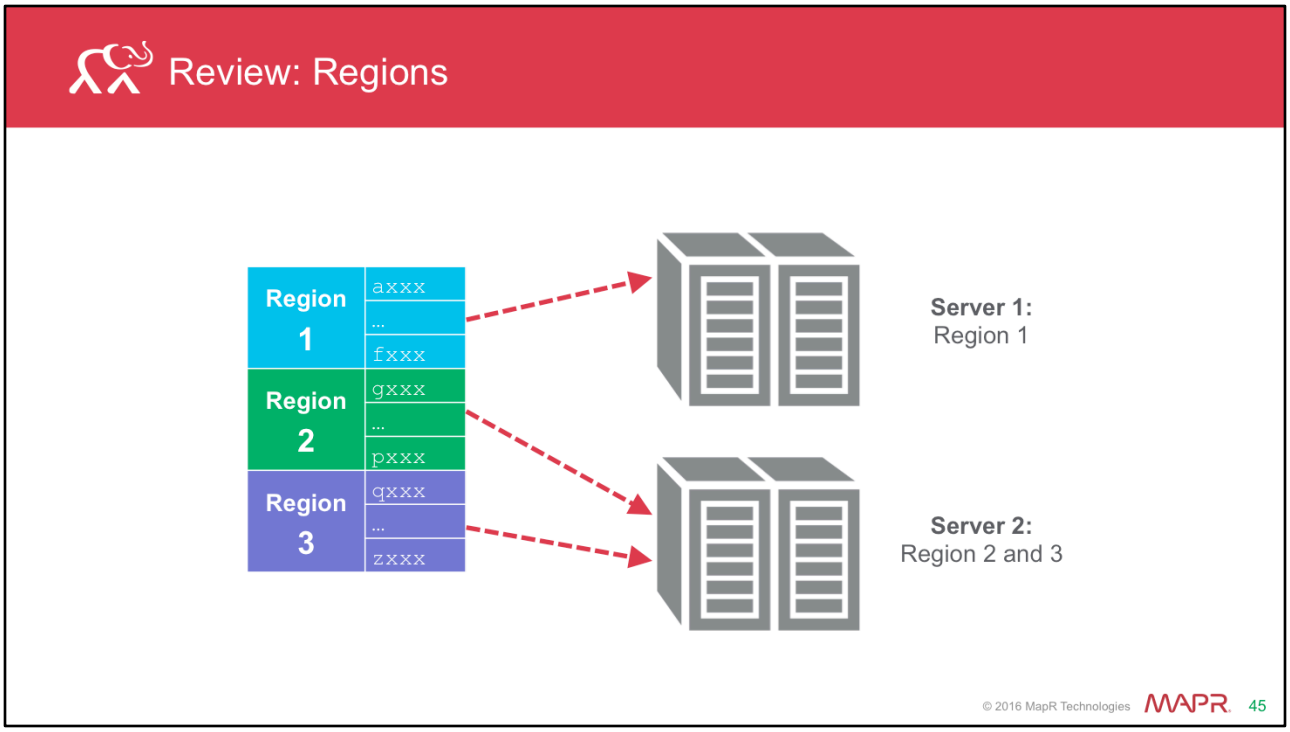
Learning Goals

- Describe Bulk Loading Data Process
- Use ImportTsv
- Bulk Load with MapReduce
- Pre-split Tables

Review: Regions

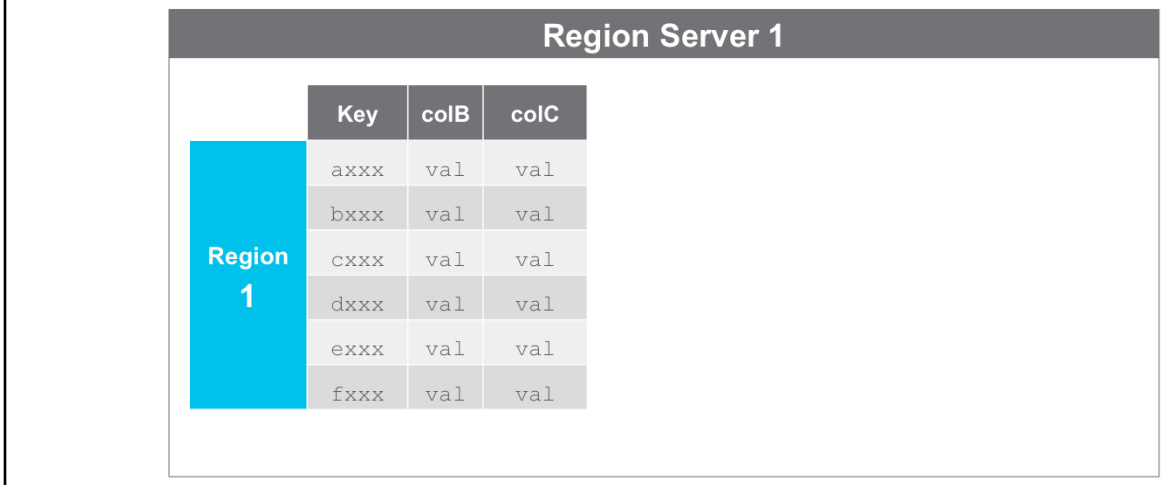
	RowKey	Address			Order	
		Name	Address	Email	Date	ItemNo
Region 1	axxx	val		val	val	
	...					
	fxxx	val			val	val
Region 2	gxxx					
	...	val	val	val	val	val
	pxxx					
Region 3	qxxx	val				
	...	val		val	val	
	zxxx					

Looking back on our previous courses, HBase tables are split into sequences of rows, by key range, called regions.



This is done to help spread out the load of HBase operations across the cluster.

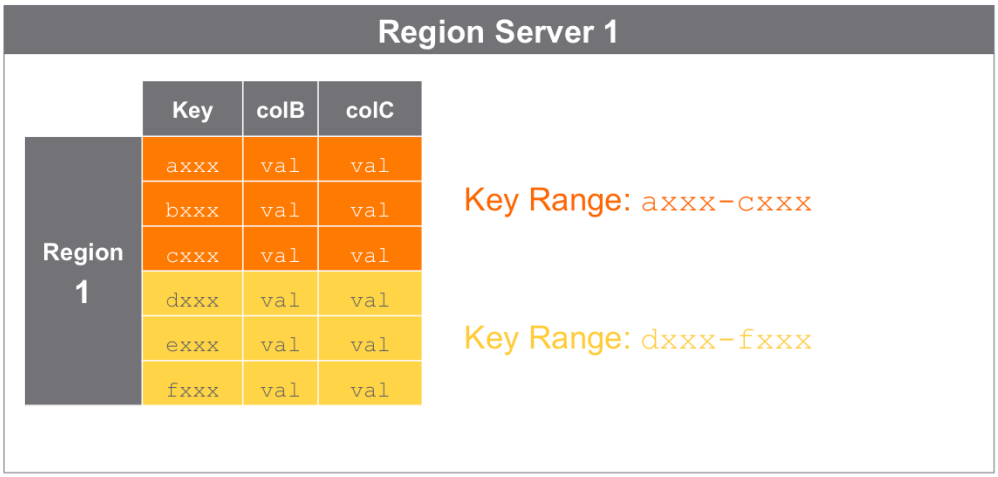
Region Split



 46

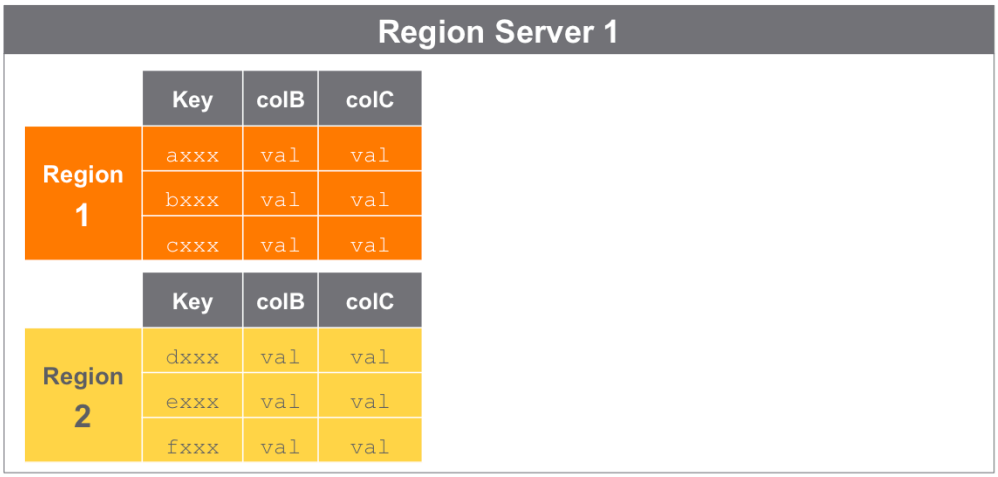
When a region grows too large

Region Split



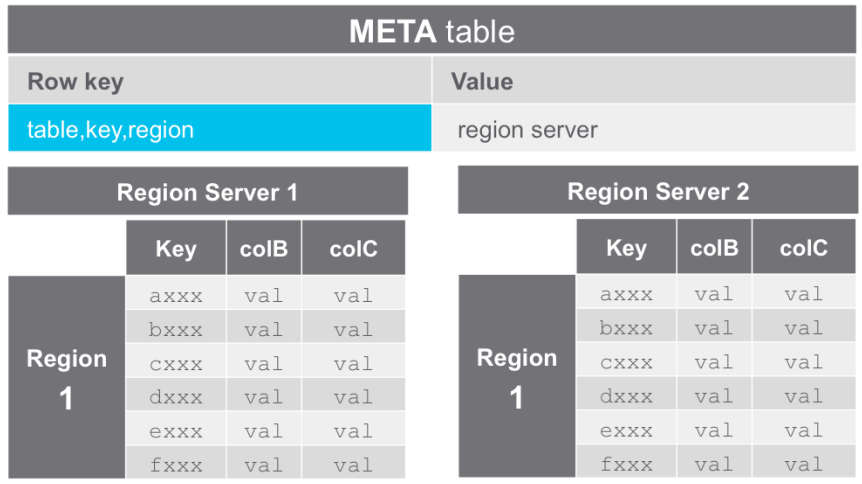
it is automatically split by key range

Region Split



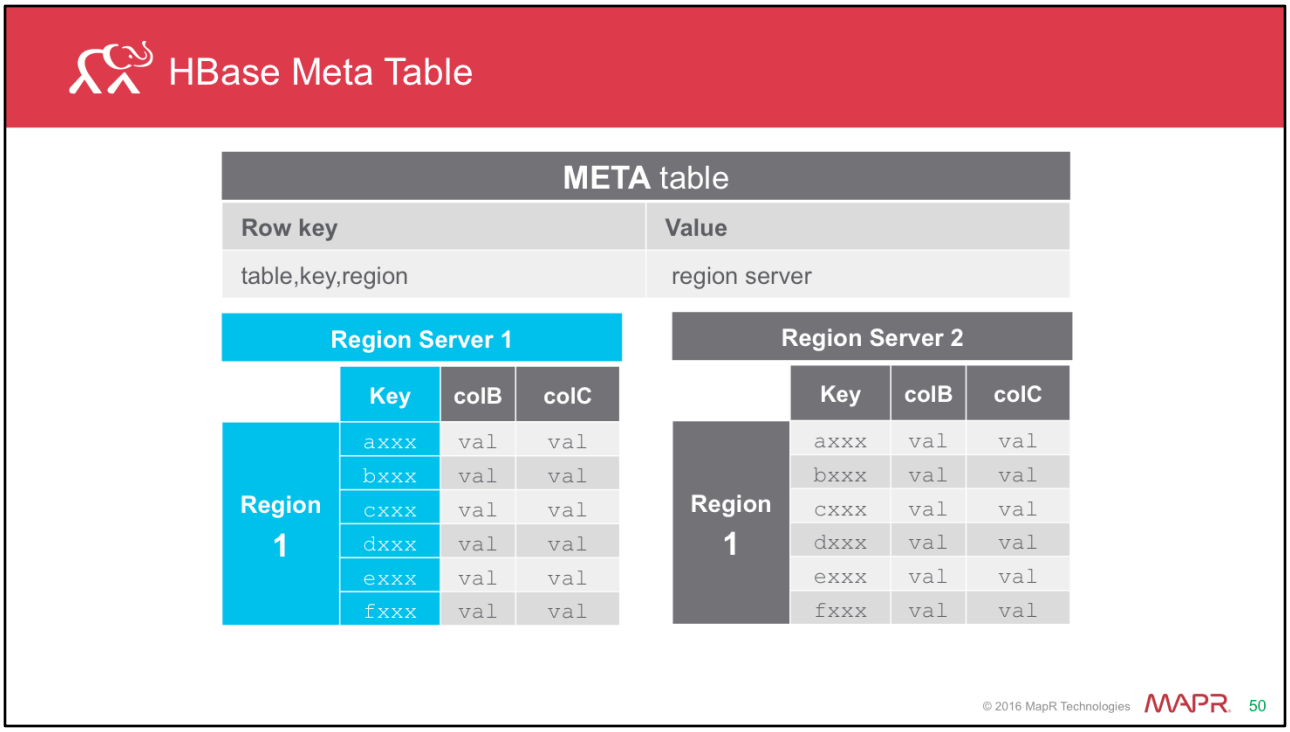
Creating two child regions.

HBase Meta Table



 49

The META table is an HBase table that keeps a list of all regions in the system. The META Table allows HBase clients under the cover.



The META table is used to find the region for a given table key to look up a region by key.

HBase Meta Table

META table

Row key	Value
table,key,region	region server

Region Server 1

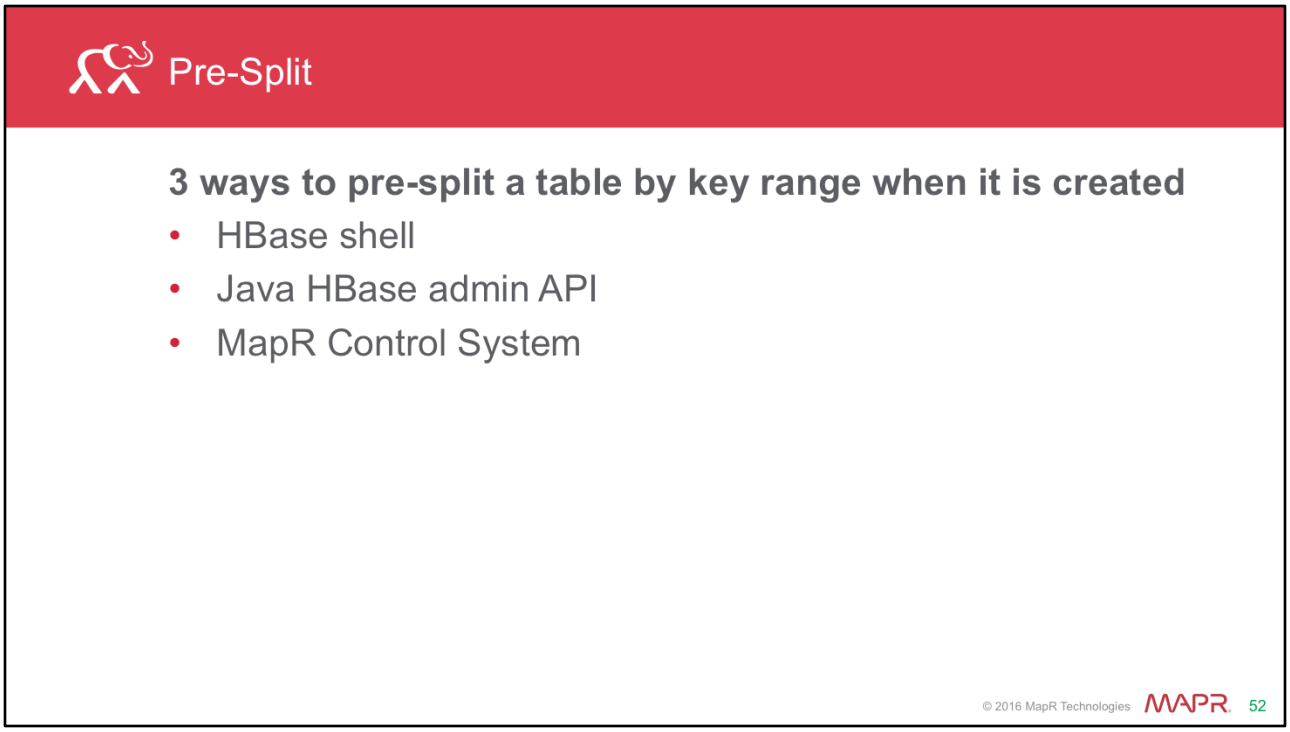
	Key	colB	colC
Region 1	axxx	val	val
	bxxx	val	val
	cxxx	val	val
Region 2	dxxx	val	val
	exxx	val	val
	fxxx	val	val

Region Server 2

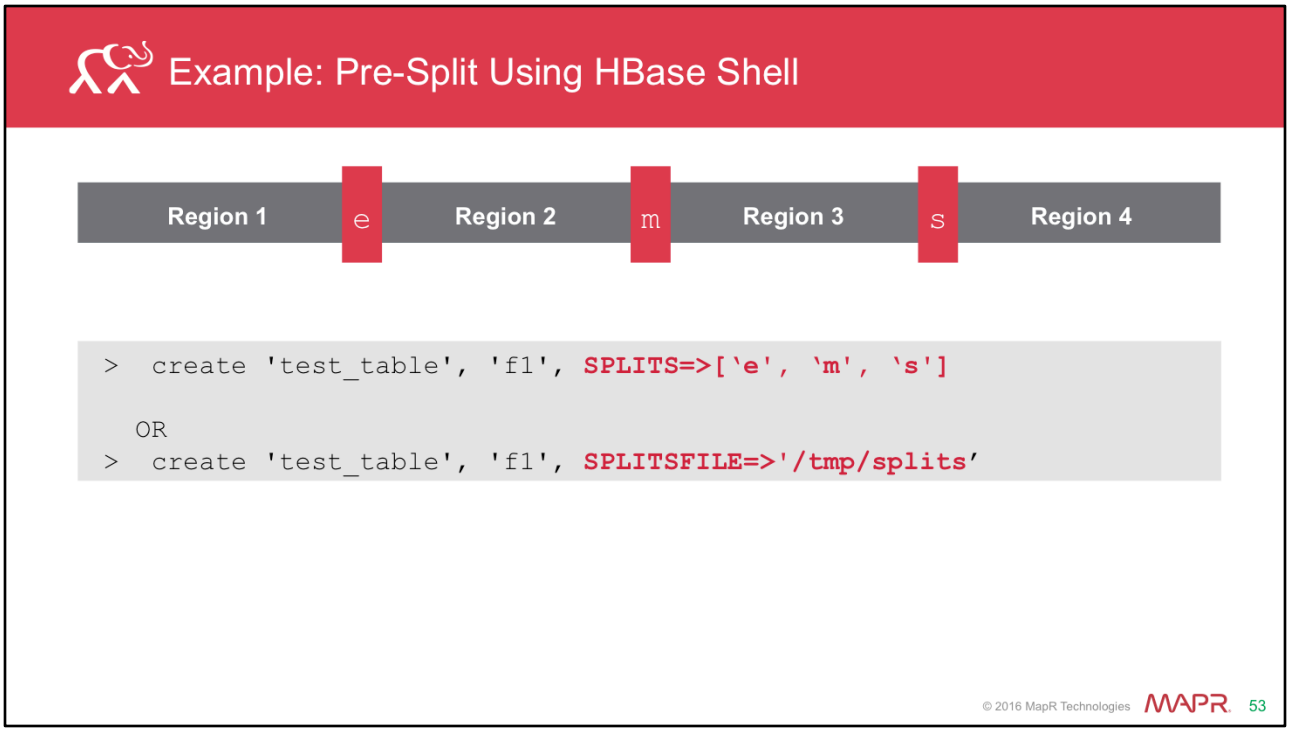
	Key	colB	colC
Region 1	axxx	val	val
	bxxx	val	val
	cxxx	val	val
	dxxx	val	val
	exxx	val	val
	fxxx	val	val

 51

The META table is updated when regions split.



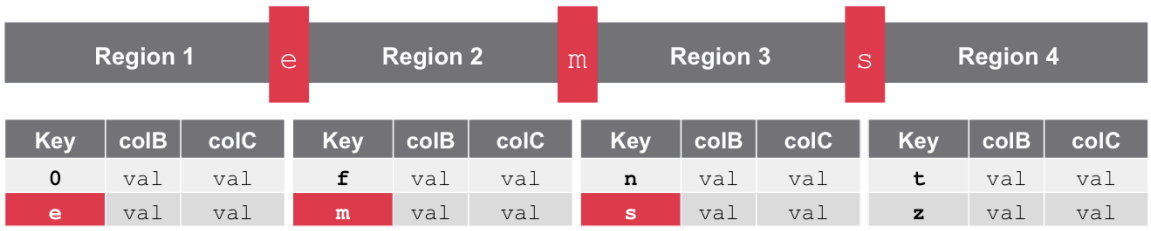
It is possible to pre-split a table by key range when it is created.
There are 3 ways you can do this:
You can use the HBase shell, which we will explore further
The Java HBase admin API,
or MapR Control System



When you presplit a table using the HBase shell, you can specify the split keys in an array, or you can read the split keys from a file.

This creates and pre-splits a table with an initial set of empty regions defined by the specified split keys, which form the start and end keys of the regions created.
Therefore, the total number of regions created will be the number of split keys plus one.

Example: Pre-Split



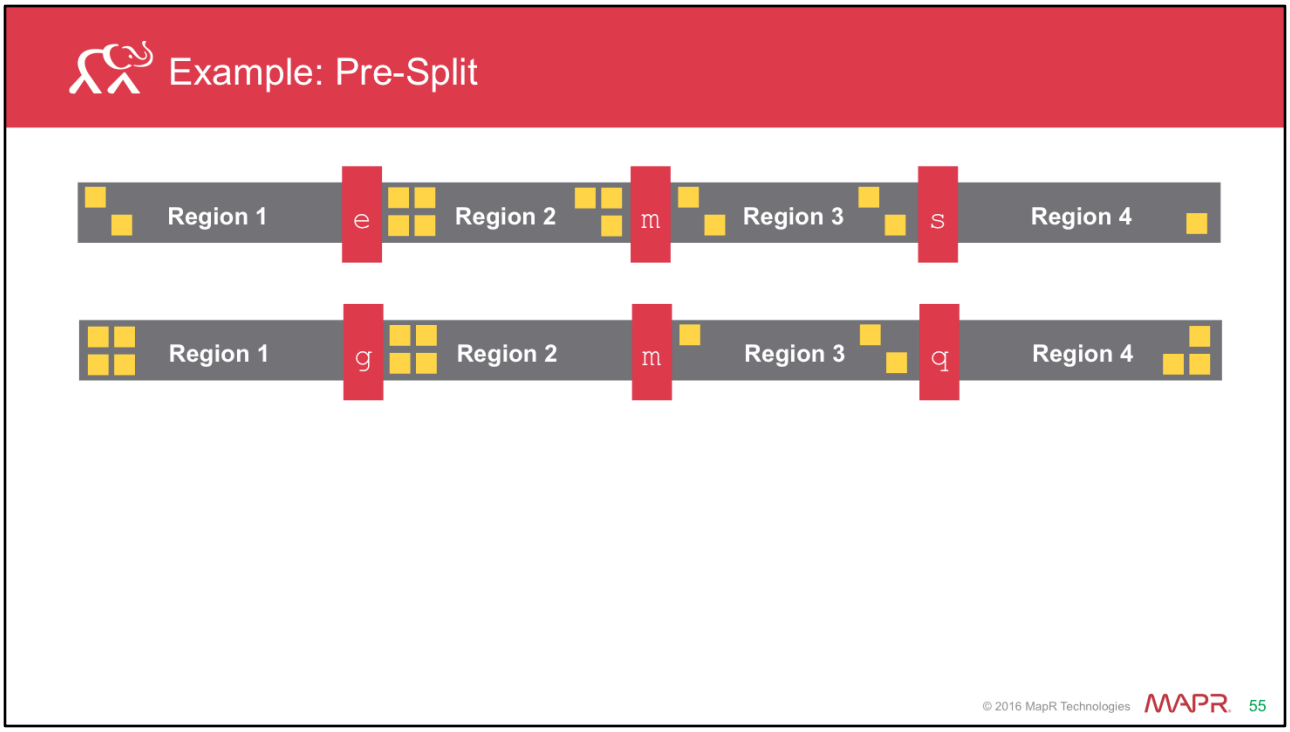
Key	colB	colC
0	val	val
e	val	val

Key	colB	colC
f	val	val
m	val	val

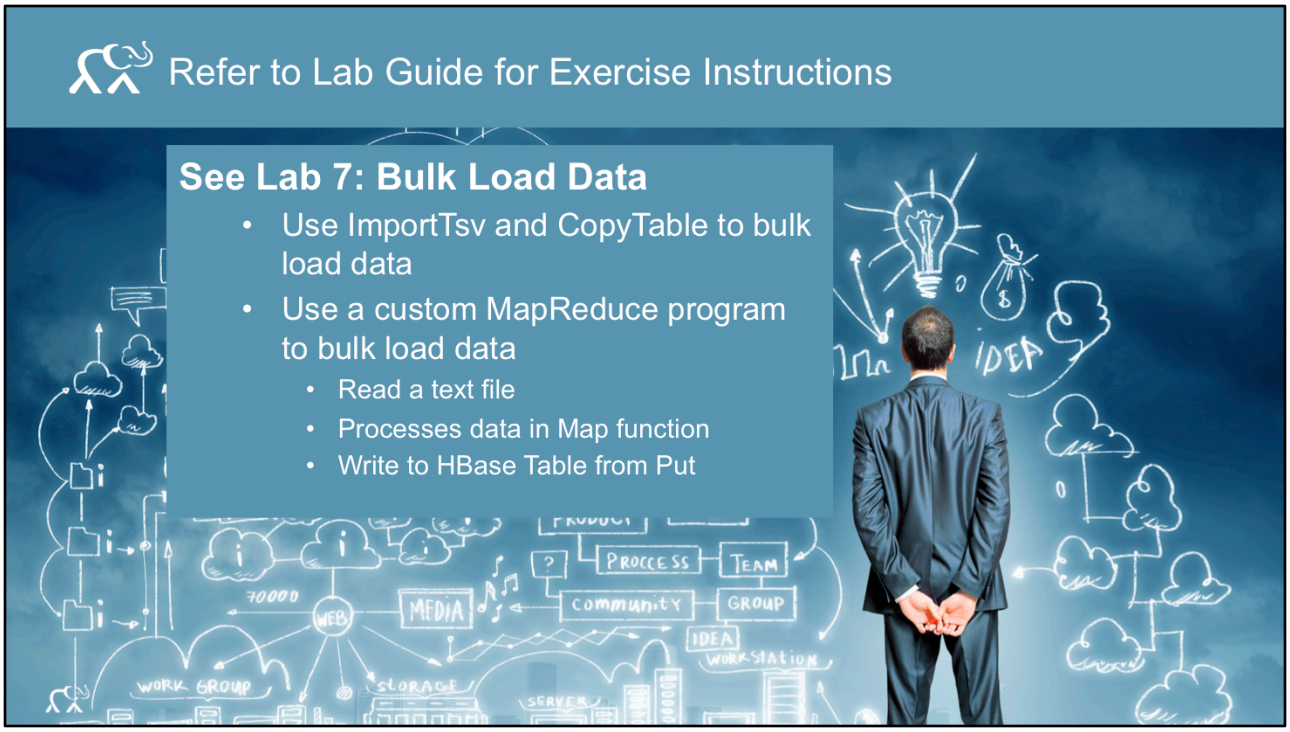
Key	colB	colC
n	val	val
s	val	val

Key	colB	colC
t	val	val
z	val	val

In this example, the split keys are e,m,s, which creates 4 regions. e is the end key for region 1, and s is the start key for region 4.



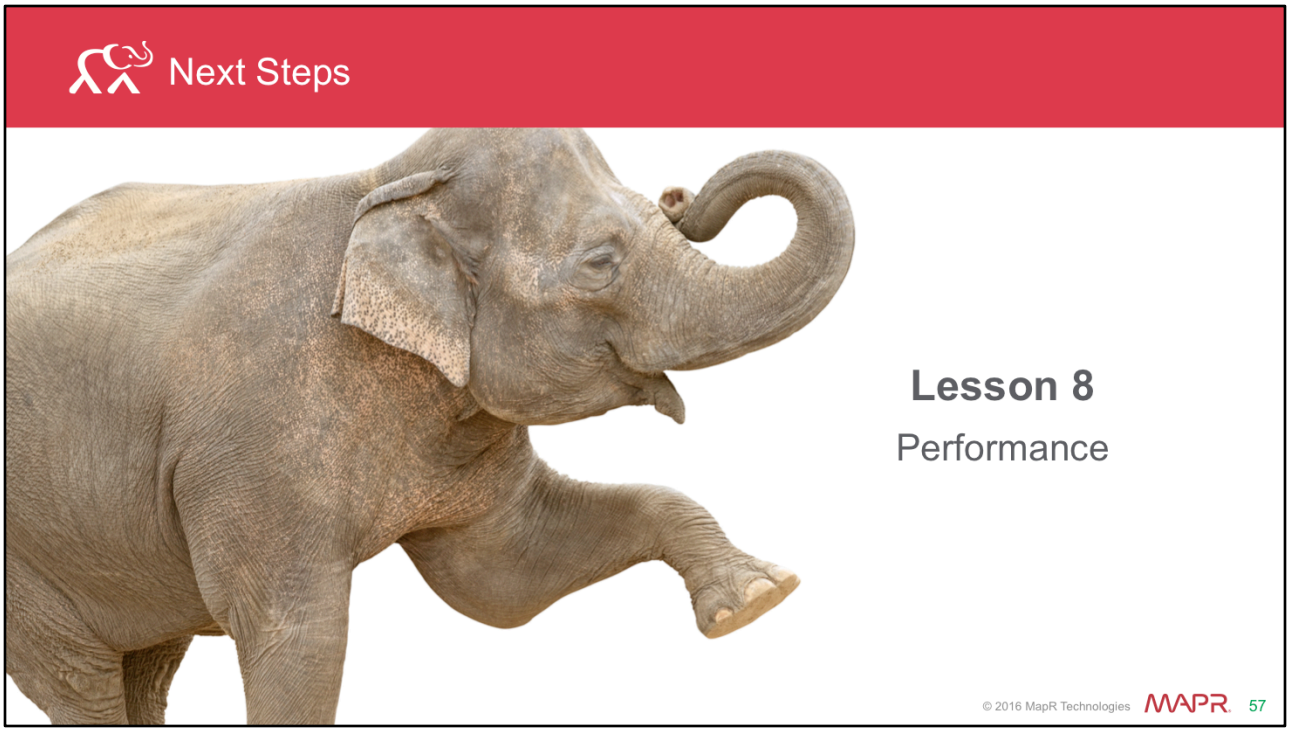
When presplitting tables, it is important to plan where the key splits will be based on the distribution of your data. Presplitting the tables will make loading faster, but if the region splits are not planned well the data load will be unbalanced.



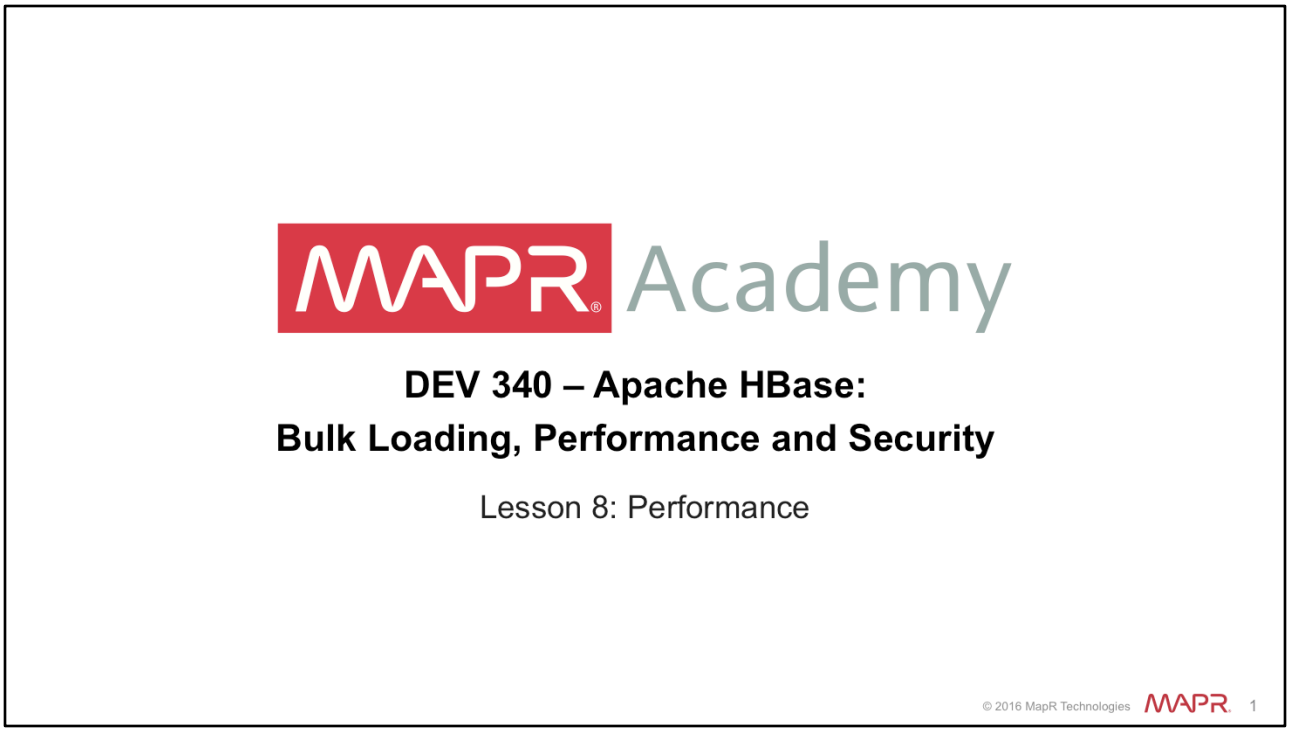
We have discussed several techniques to import data in bulk with an emphasis on importing from a SQL db.
1) We looked at a simple client using the HBase API and the java JDBC API to transfer data from a SQL db to MapR-DB
2) We looked at using importsv a built-in HBase tool to import text data
3) We saw how we can use a MapReduce task
4) Some other options/clients like thrift, hive, pig, rest and CLI
5) Finally was saw how to optimize by pre-splitting table before importing data into it.

Next, lets work on the lab exercises:
1) Using the importtsv tool & Copytable
2) Using MapReduce program that reads a text file, processes the data in Map function and the uses Put to write to the Hbase Table.

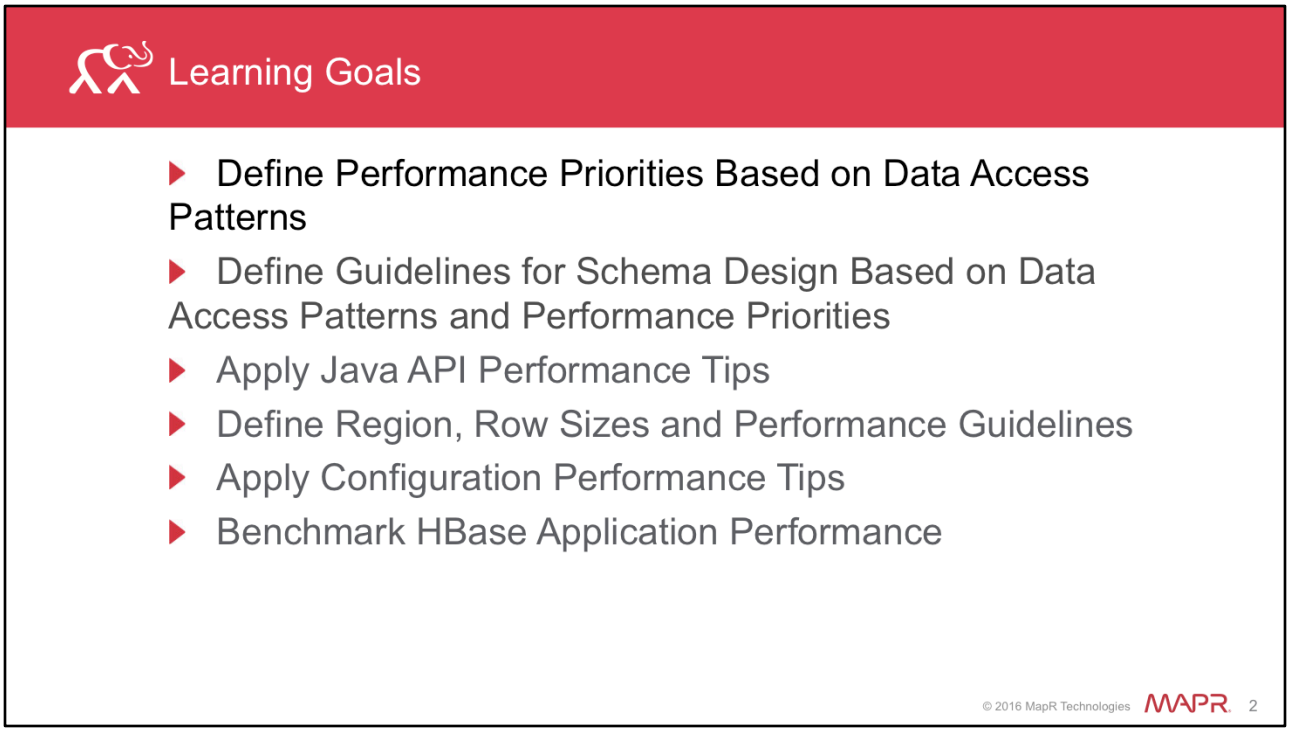
Lab Documents:
Lab_11_working_importtsv_copytable.pdf
working with importtsv and copytable
Lab_11_BulkLoad.pdf
BulkLoad
lab-exercises in eclipse



Congratulations, you have finished DEV 340 Lesson 7, Bulk Loading Data into MapR Tables. Continue on to lesson 8 to learning about improving the performance of your HBase and MapR-DB tables.

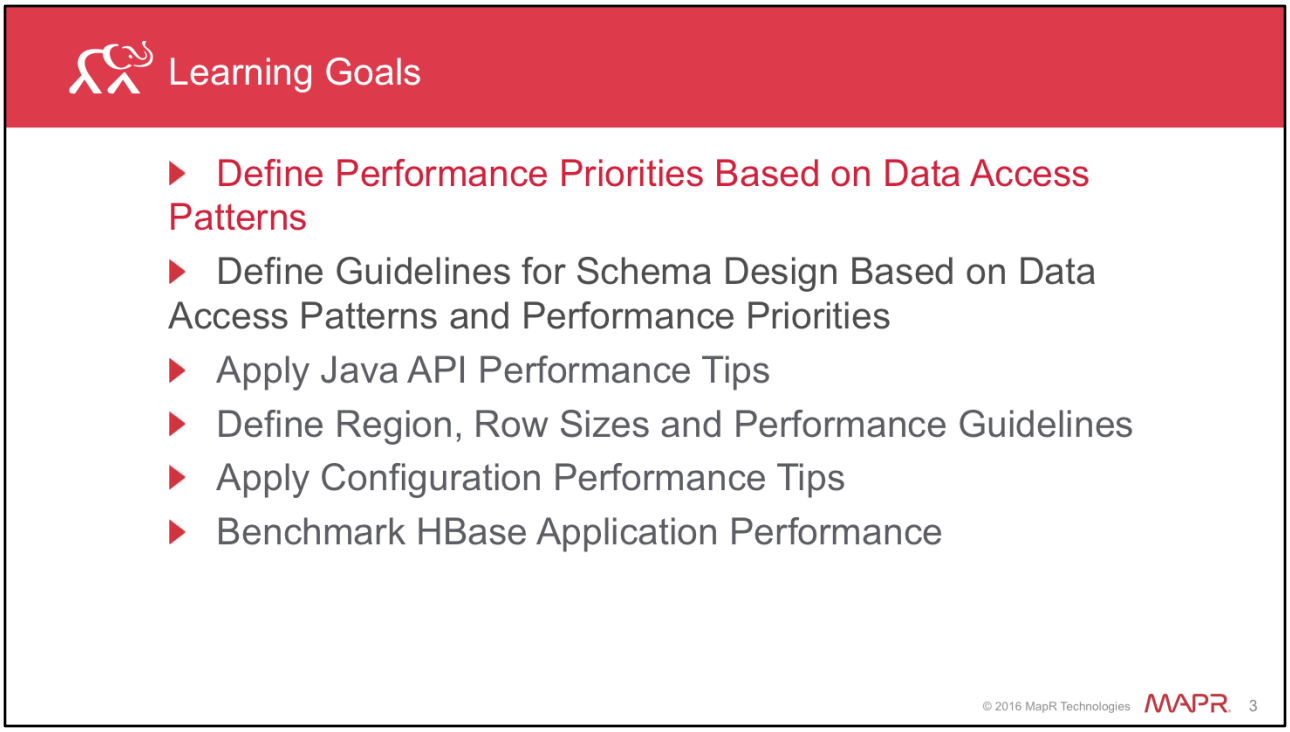


Welcome to DEV 340 Lesson 8, Apache HBase bulk loading, performance and security.

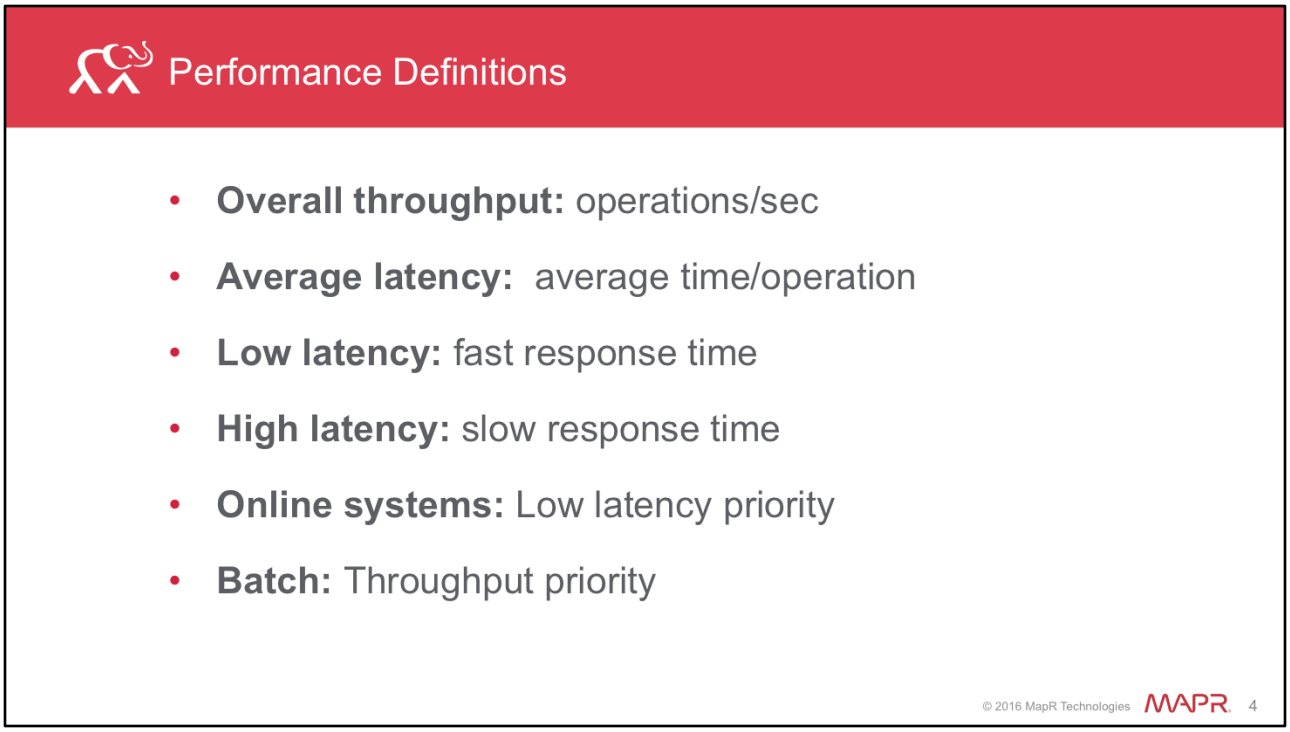


When you have finished with this lesson, you will be able to:

Understand How Data Access Patterns affect Performance Priorities
Define Guidelines for HBase Schema Design based on Data Access Patterns and performance Priorities
Apply Performance Tips when using the Java API
Define the Performance Guidelines for HBase Region and Row Sizes
Apply performance tips when Configuring HBase settings
And Benchmark HBase application performance



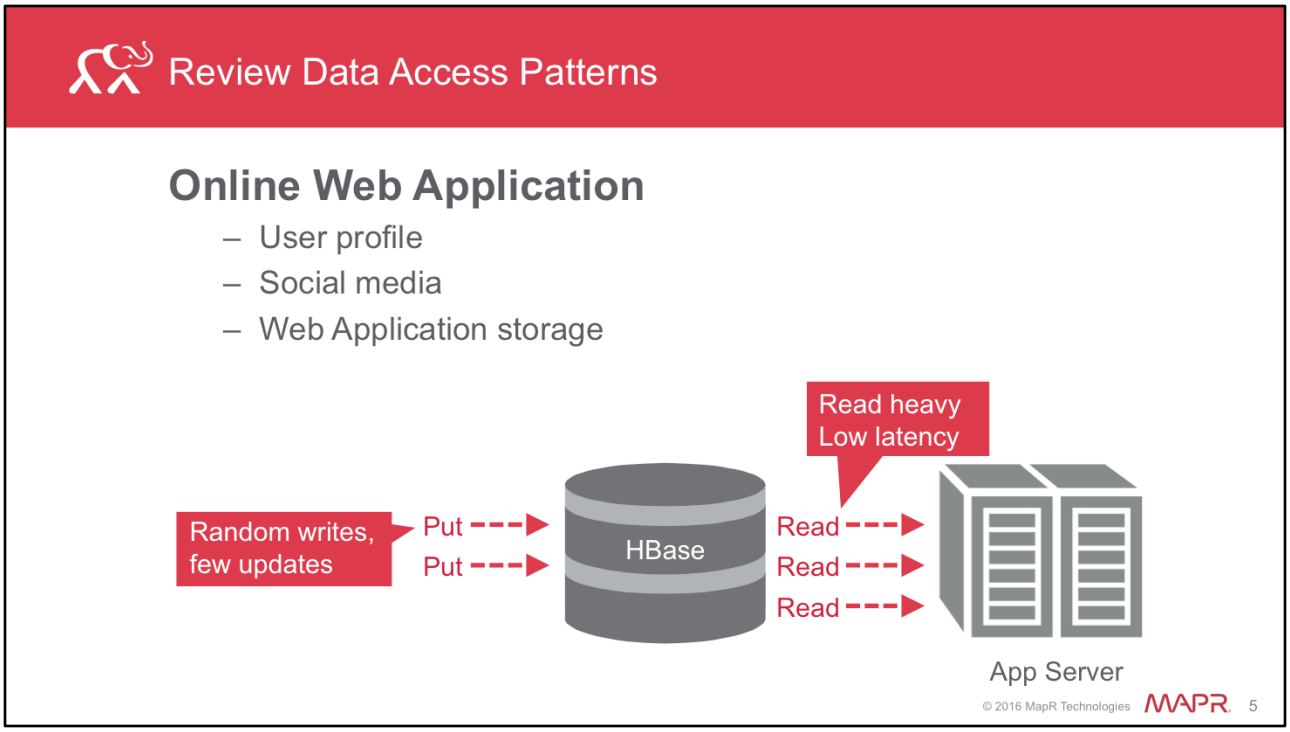
In the first section, we will define some HBase access patterns based on performance priorities.



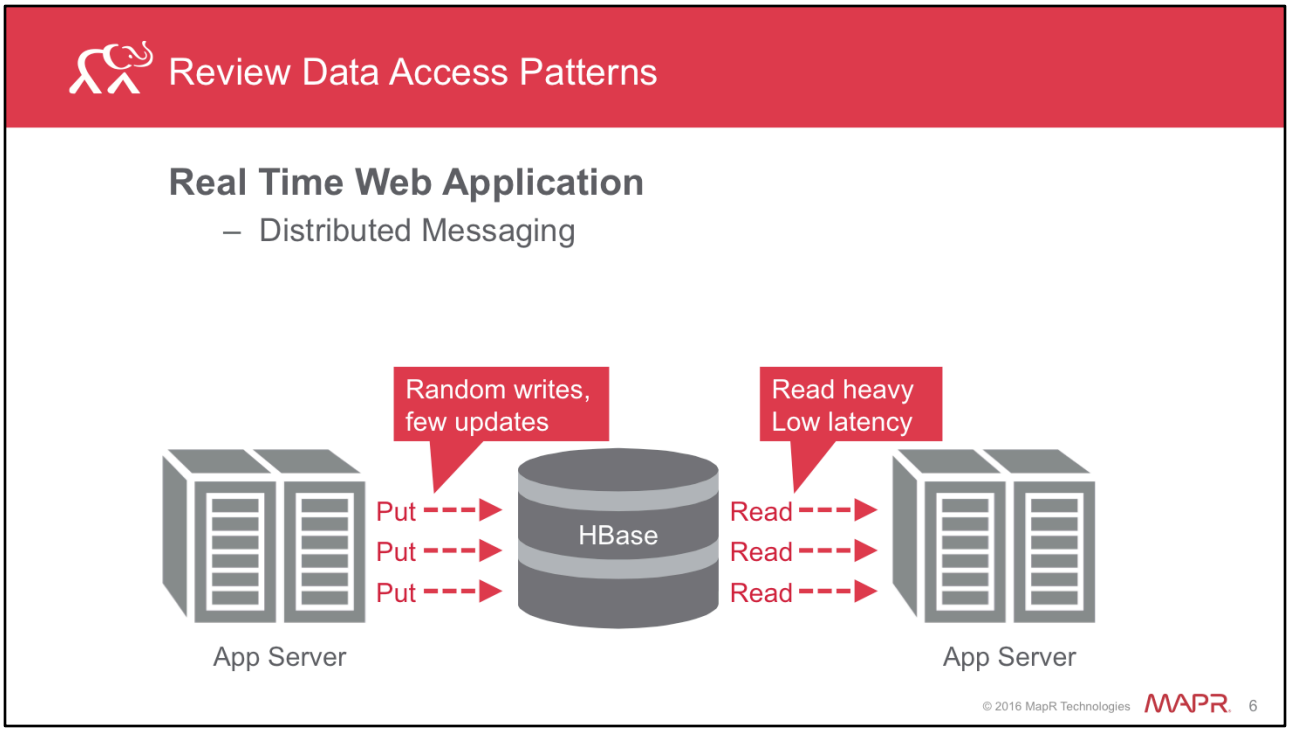
Before we talk about performance tips and techniques, here is a review of some terms we will be using.

Throughput is the number of operations per second, while Latency is the average time per operation.

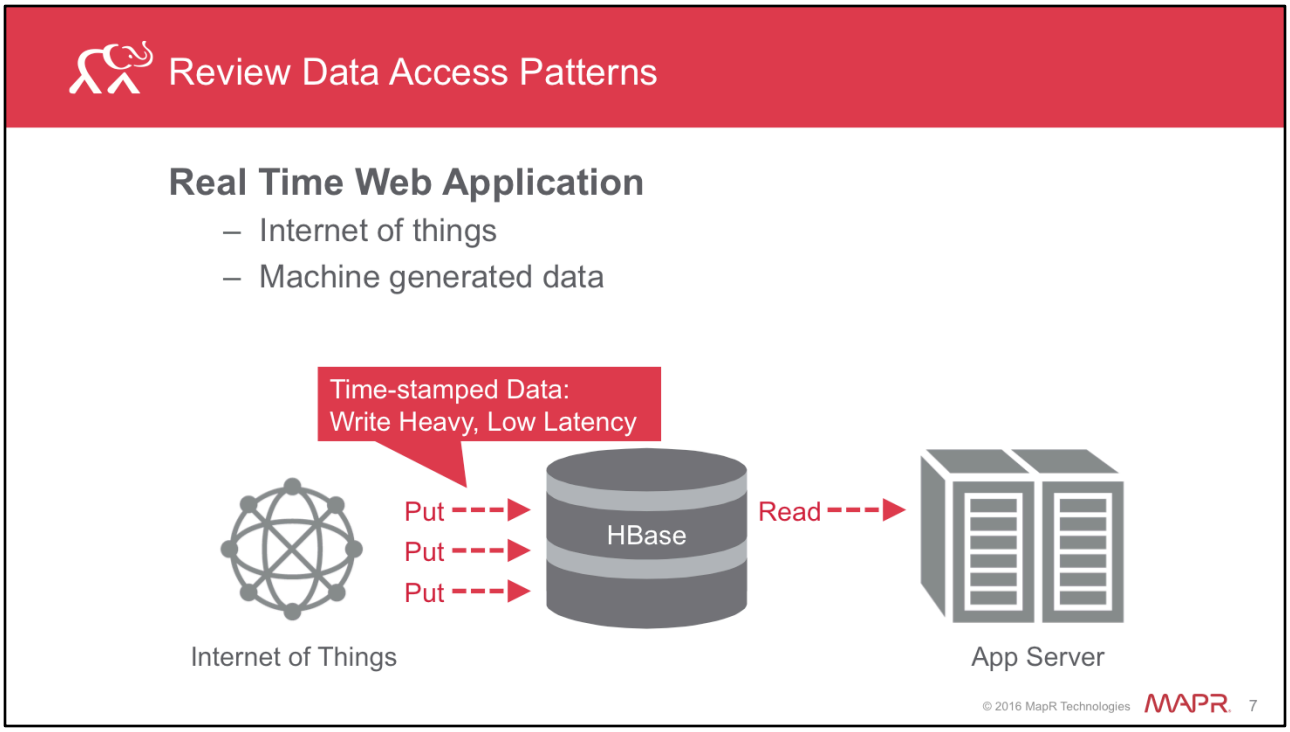
Online systems have low-latency requirements, since a person is waiting for a response.
Batch systems have high throughput as the priority.



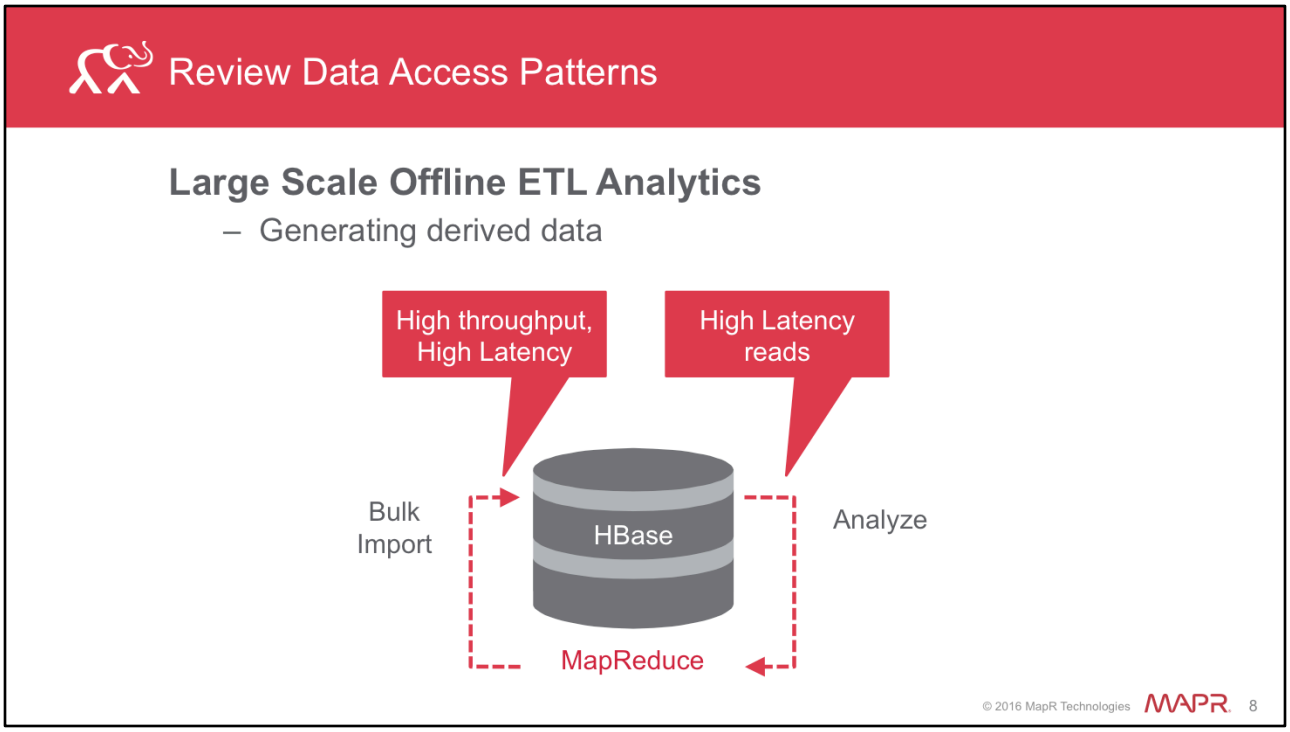
When HBase is used as the back end data store for an Online Web Application, the data access pattern is read heavy, and the priority is low latency reads.



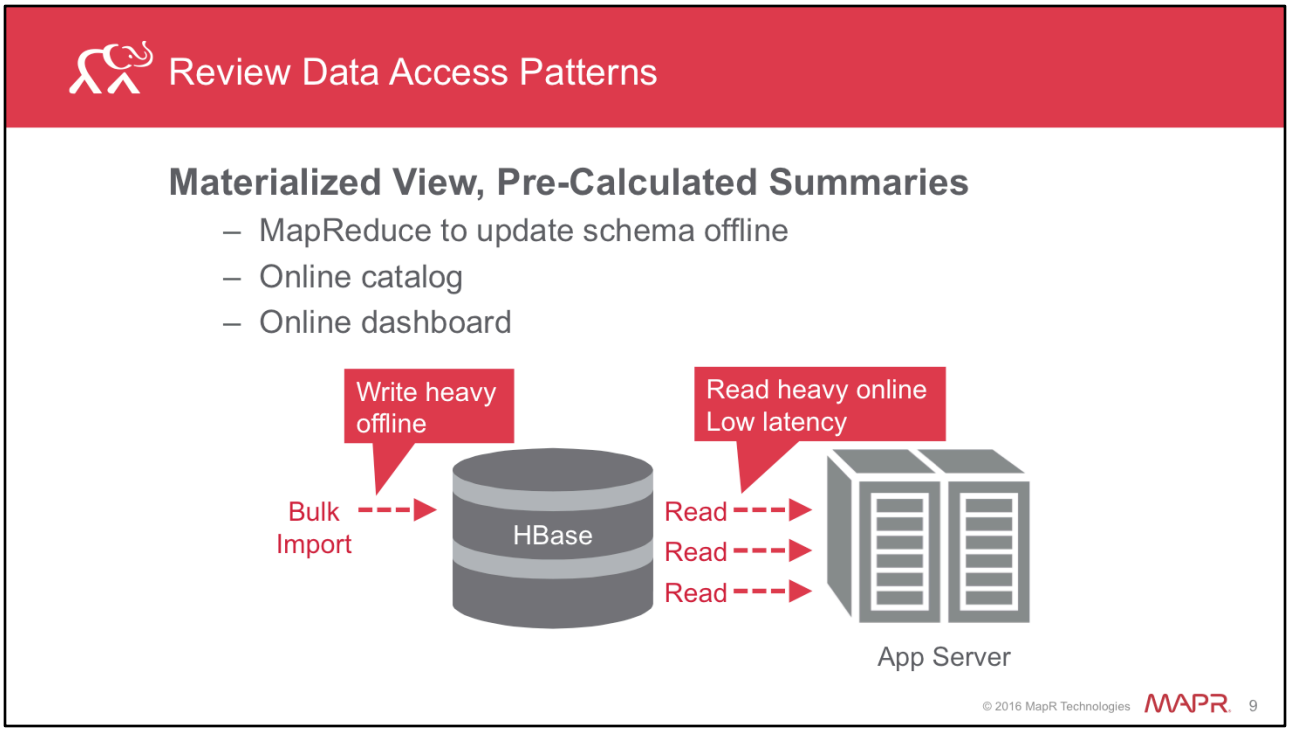
A messaging application which is using HBase as the data store, such as Facebook's messaging application, will be read and write heavy. Since this is read/write online, the priority will be low latency for both reads and writes.



For use cases such as the Internet of things, where sensors are generating a lot of data to be stored in HBase, the data access pattern will be write heavy, and the priority will be low latency writes.



Batch ETL jobs using Hive or MapReduce do not expect a quick response time, the priority is throughput.



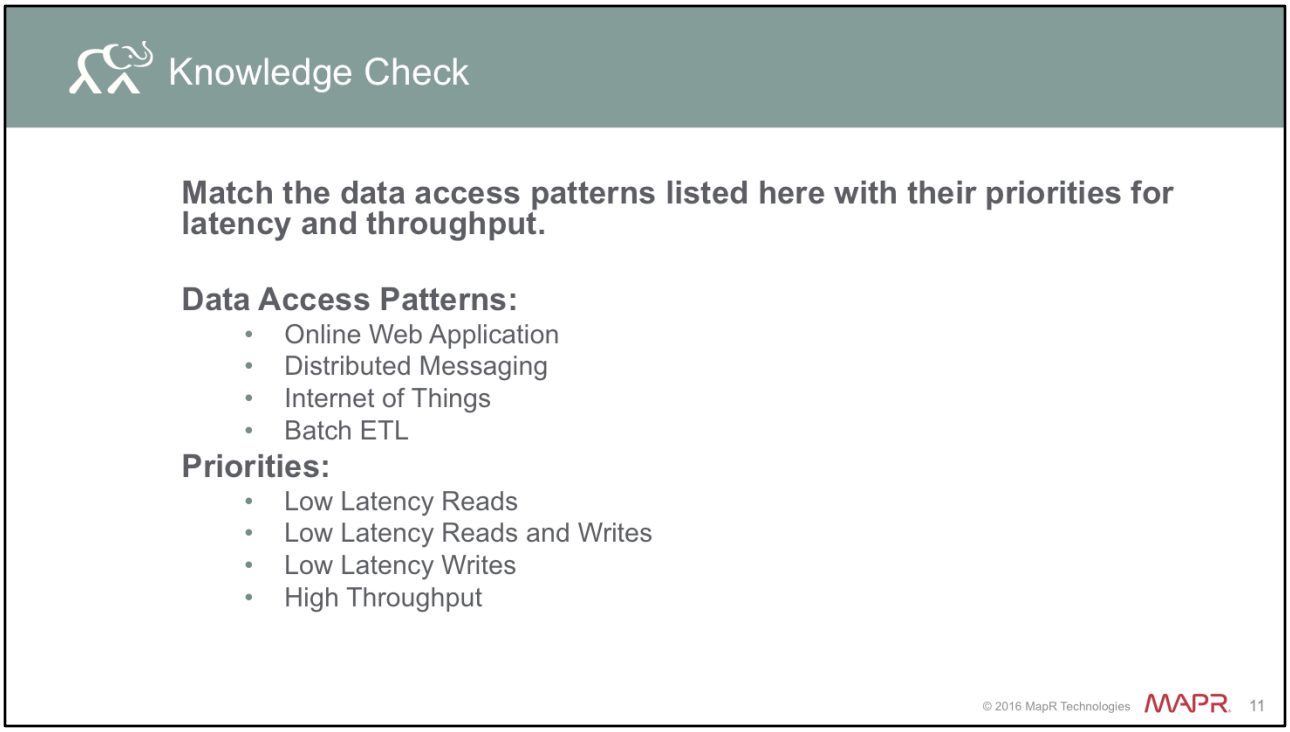
In order to provide for fast reads, as discussed in schema design, you should de-normalize your schema so that the data that is read together is stored together. MapReduce jobs can be used to bulkload your data into de-normalized, pre-computed, or materialized views, grouping the data together for faster reads based on your query patterns.

MapReduce jobs are batch jobs, and therefore throughput is the priority.

Class Discussion

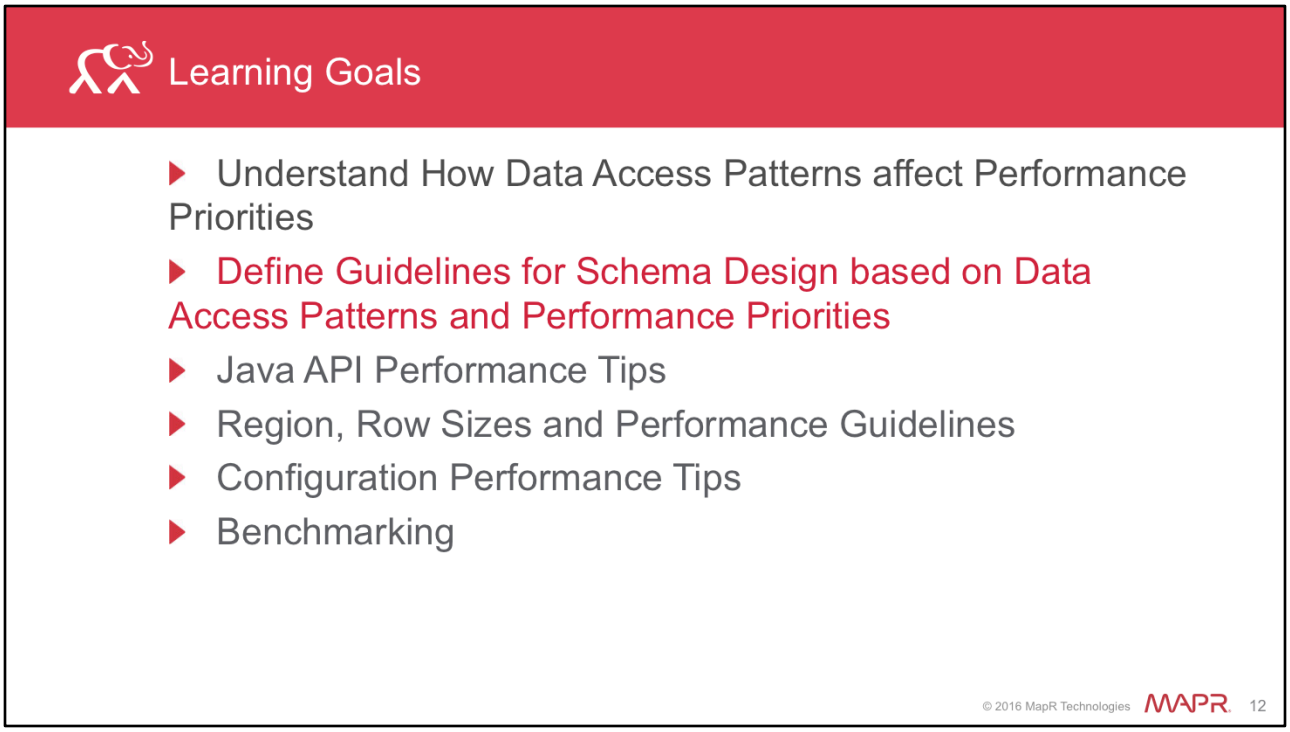
Think about the Data Access Patterns that we just reviewed.

- Which ones of these apply to your company, and what other data access patterns have you used?
- In what context?

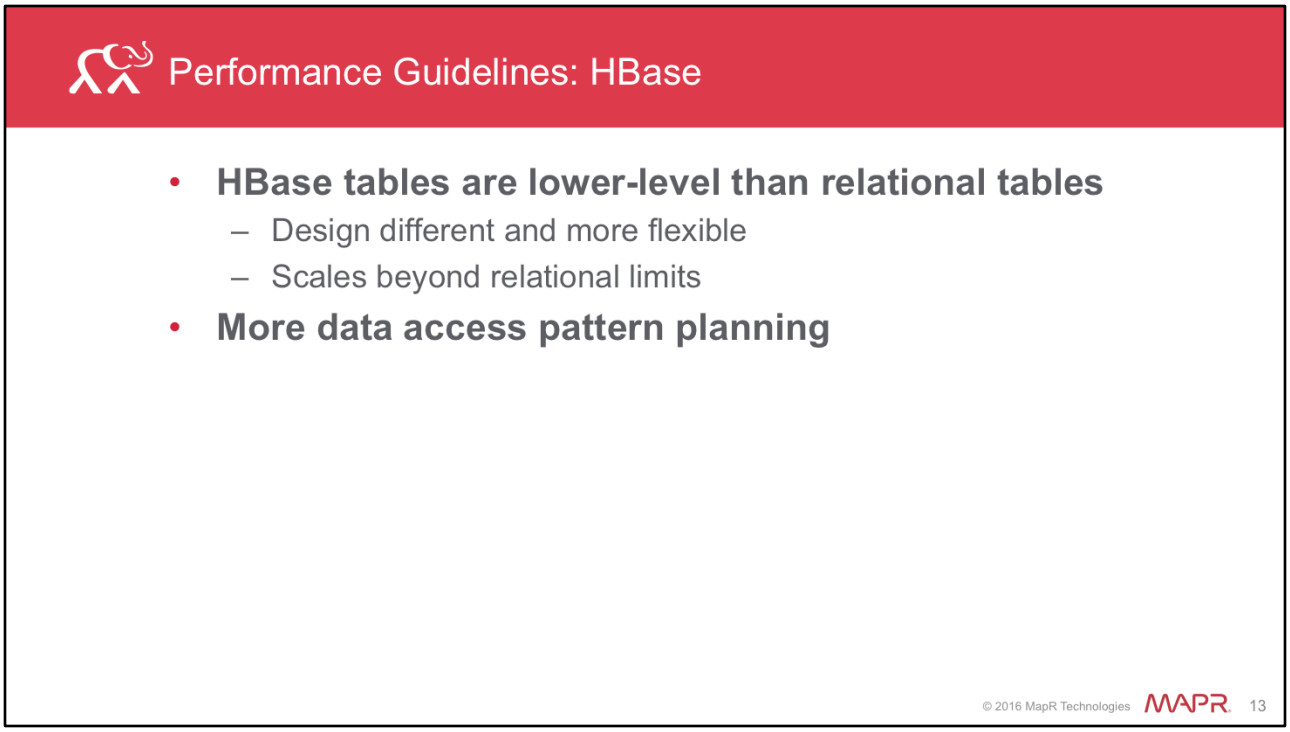


Match the data access patterns listed here with their priorities for latency and throughput.

Online Web Application::Low Latency Reads
Distributed Messaging:: Low Latency Reads and Writes
Internet of Things::Low Latency Writes
Batch ETL::High Throughput

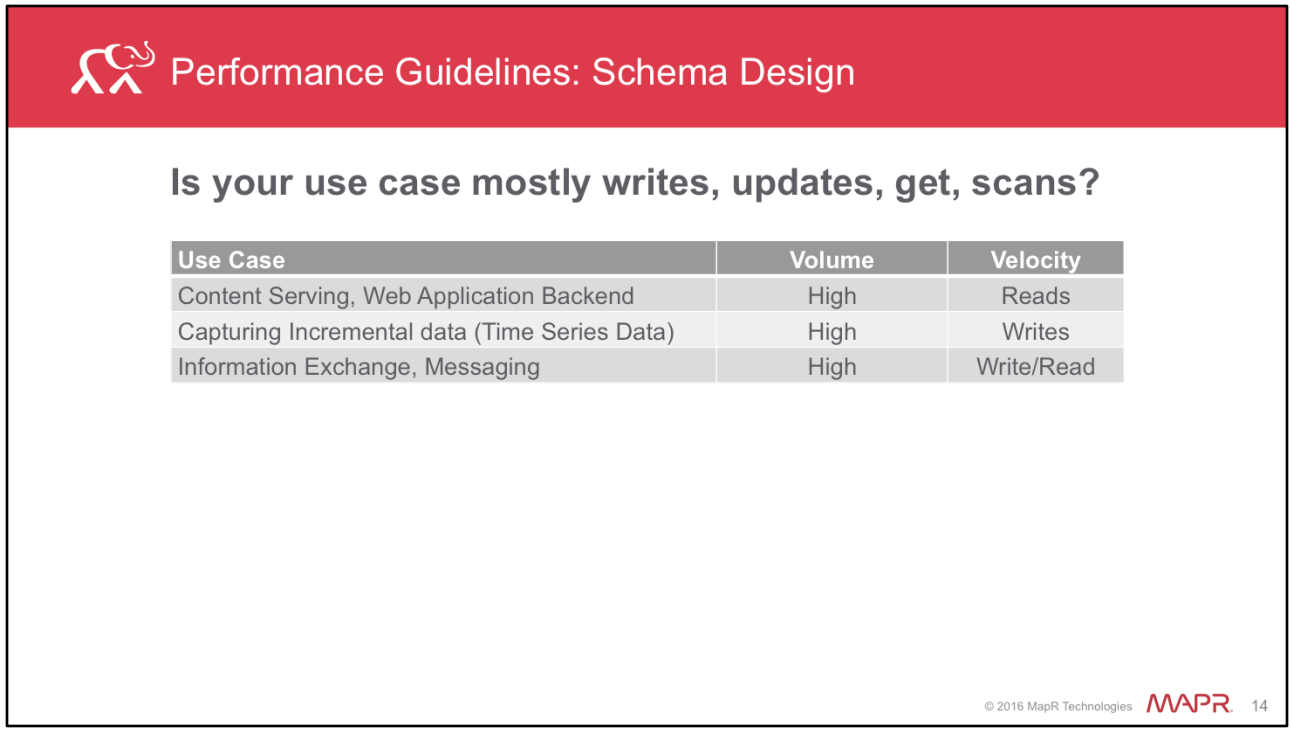


In this next section, we will discuss some schema design tips that will improve application performance based on data access patterns and priorities.



We have seen that with HBase you have to think about how your data will be read when designing your application.

You have to design your schema based on your application's data access patterns, since you can not rely on adding indexes and joins like with a relational database.
This means more planning for your data access patterns is needed, but HBase can scale across a cluster, beyond relational database limits.



Your schema design will impact performance directly when you read and write data.

Performance Guidelines: Schema Design

Is your use case mostly writes, updates, get, scans?

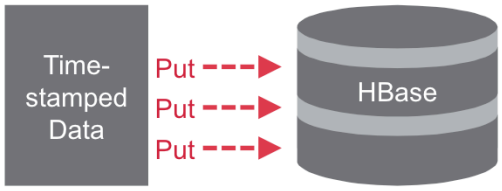
Use Case	Volume	Velocity
Content Serving, Web Application Backend	High	Reads
Capturing Incremental data (Time Series Data)	High	Writes
Information Exchange, Messaging	High	Write/Read

You should think about your use cases and data access patterns compared to the ones we just discussed. Are your priorities low latency reads?

Performance Guidelines: Schema Design

Is your use case mostly writes, updates, get, scans?

Use Case	Volume	Velocity
Content Serving, Web Application Backend	High	Reads
Capturing Incremental data (Time Series Data)	High	Writes
Information Exchange, Messaging	High	Write/Read

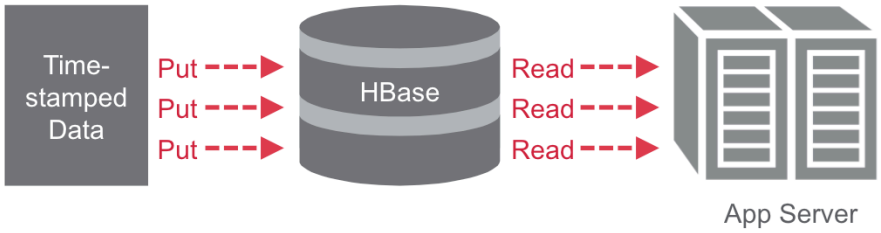


or writes?

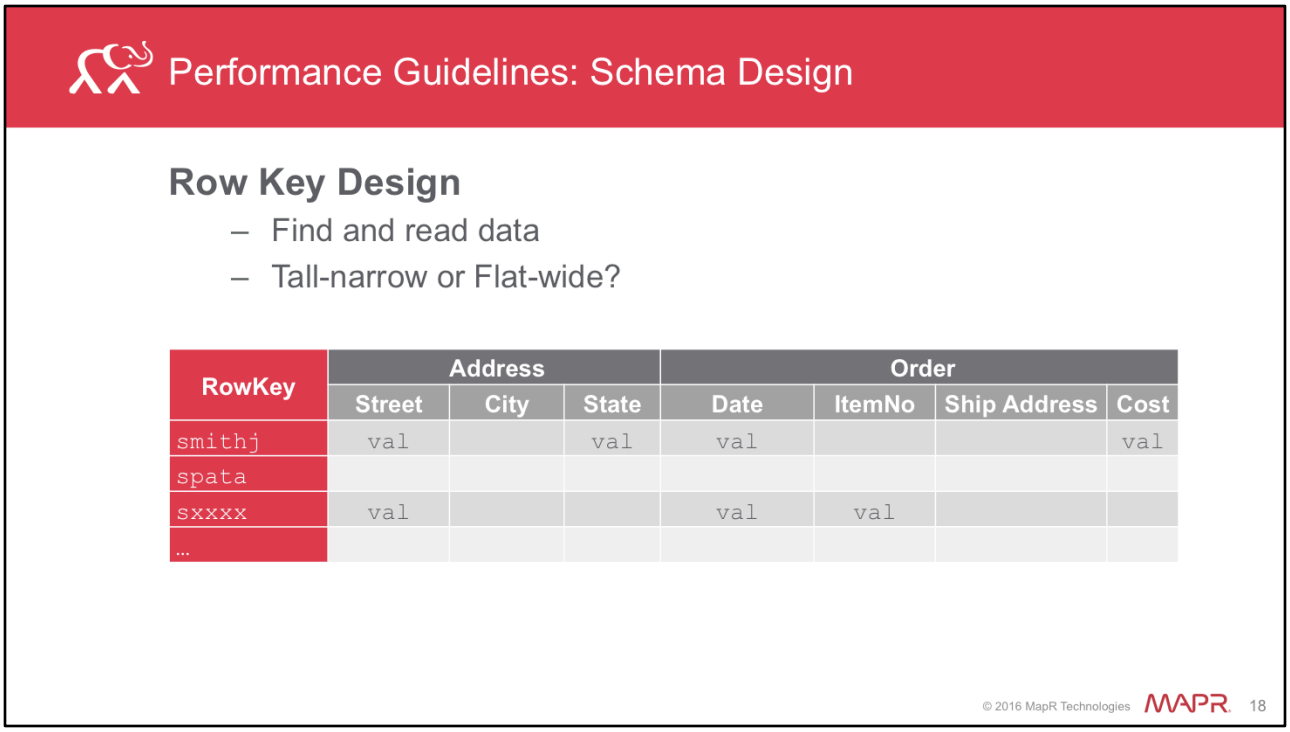
Performance Guidelines: Schema Design

Is your use case mostly writes, updates, get, scans?

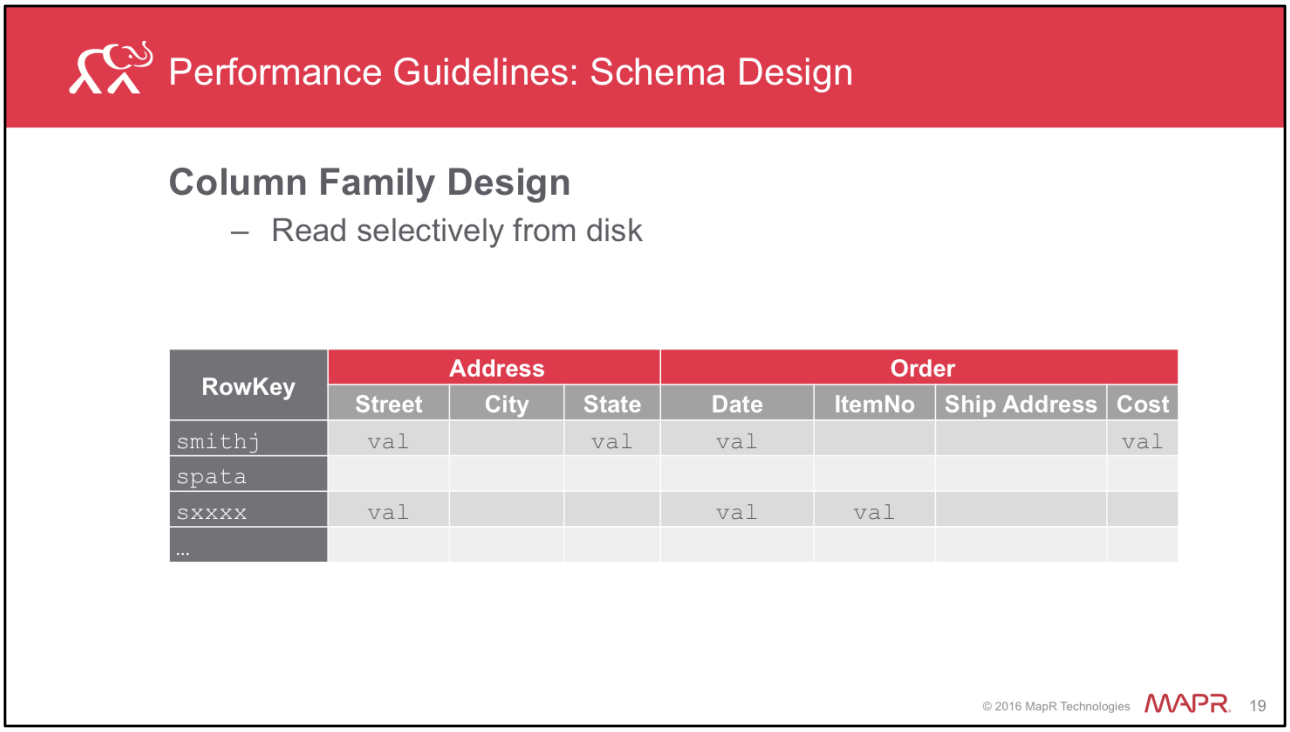
Use Case	Volume	Velocity
Content Serving, Web Application Backend	High	Reads
Capturing Incremental data (Time Series Data)	High	Writes
Information Exchange, Messaging	High	Write/Read



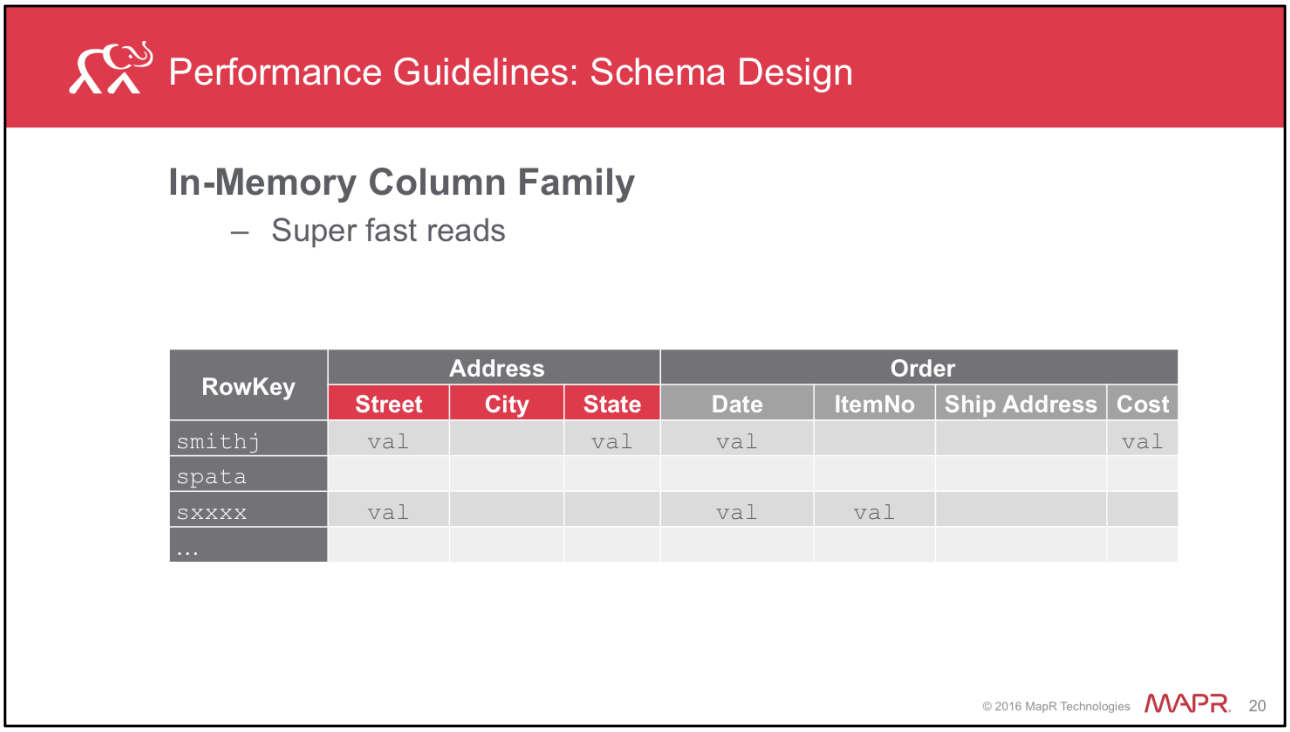
or high throughput?



When you are designing your HBase application for performance there are three significant areas of schema design: The Row Key design is critical in order to find and read data efficiently.



The Column Family design allows you to group the columns together that most typically will be read together.

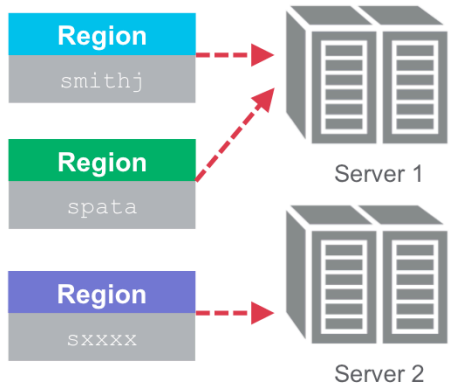


In memory column families allow for very fast reads, for data that is frequently read together and not too big to cache.

Performance Guidelines: Row Keys

Row keys determine data locality

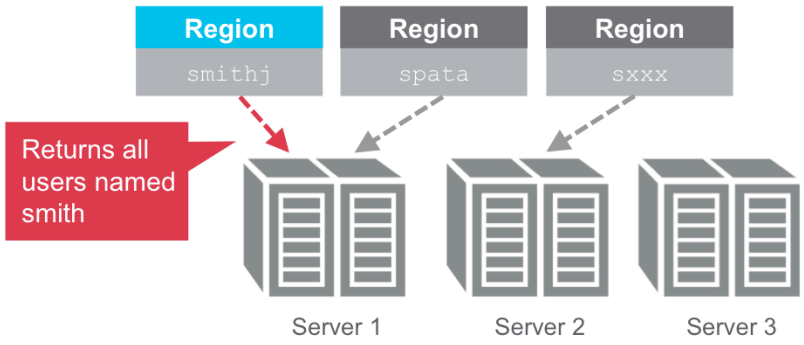
RowKey	Address		
	Street	City	State
smithj	val		val
spata			
sxxxx	val		
…			



Remember row keys determine data locality since HBase data is stored in files sorted by the Key.

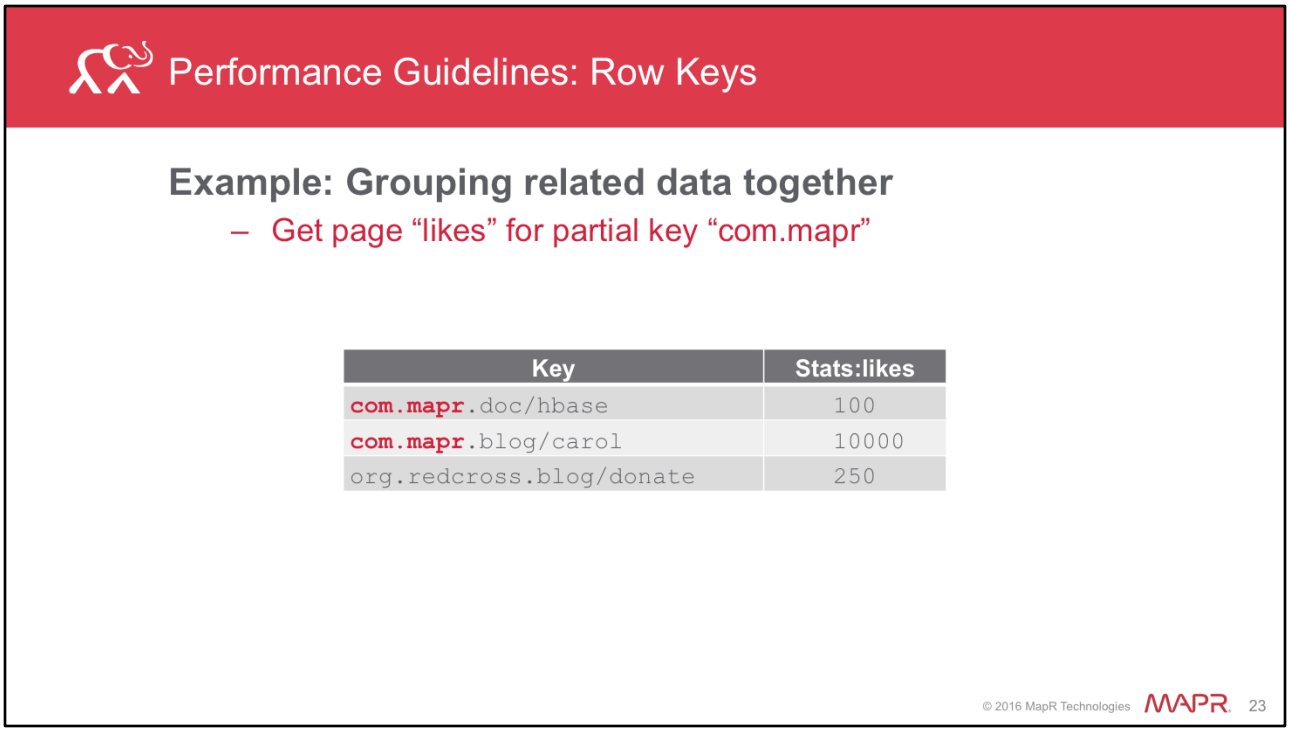
Performance Guidelines: Row Keys

- Row key significant for read performance
- Should have unique and useful meaning

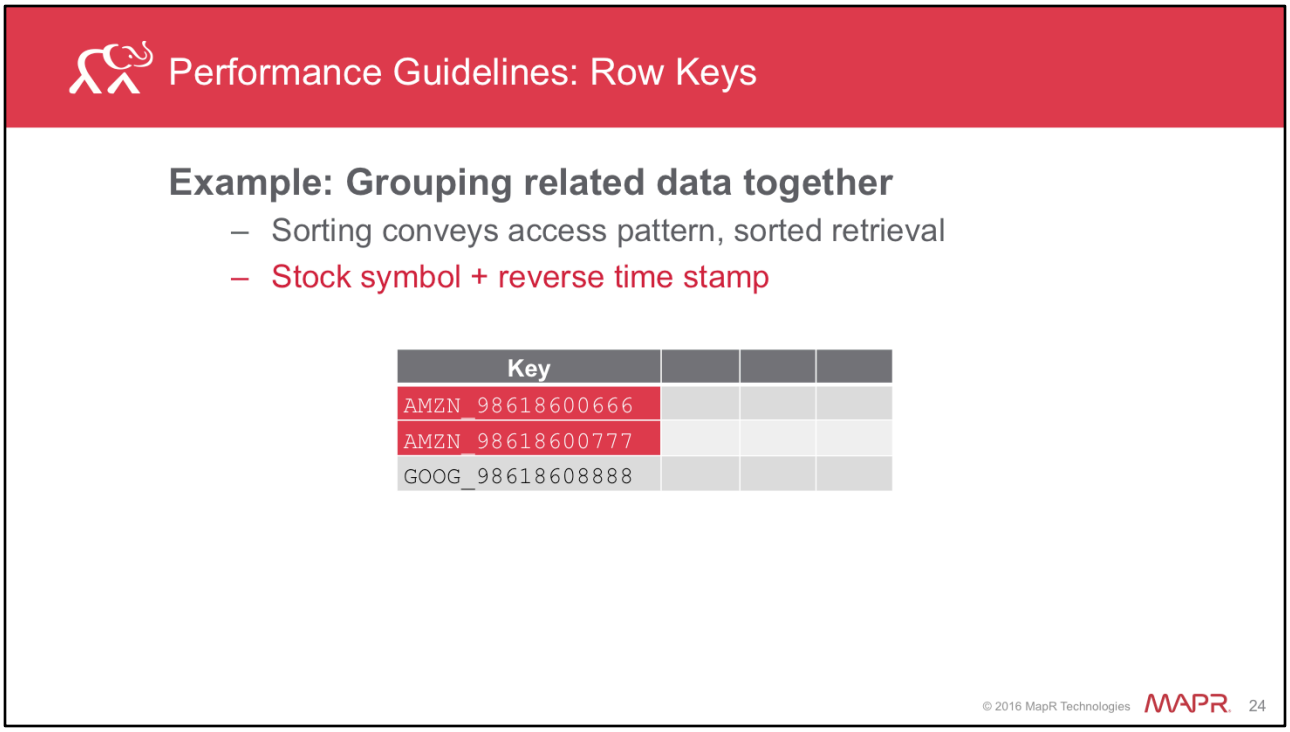


 22

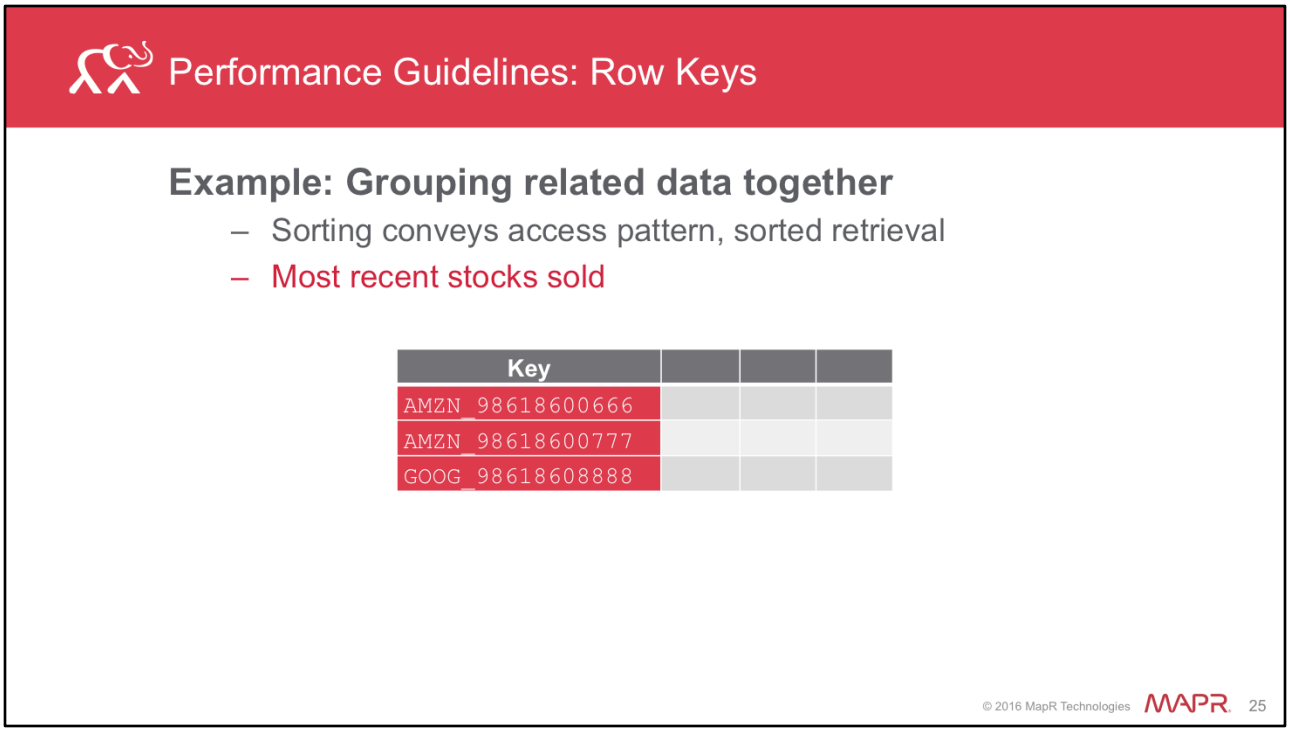
The row key is very significant for read performance, since reading data that is stored together minimizes disk seeks. Your row key should have useful information for querying your data.



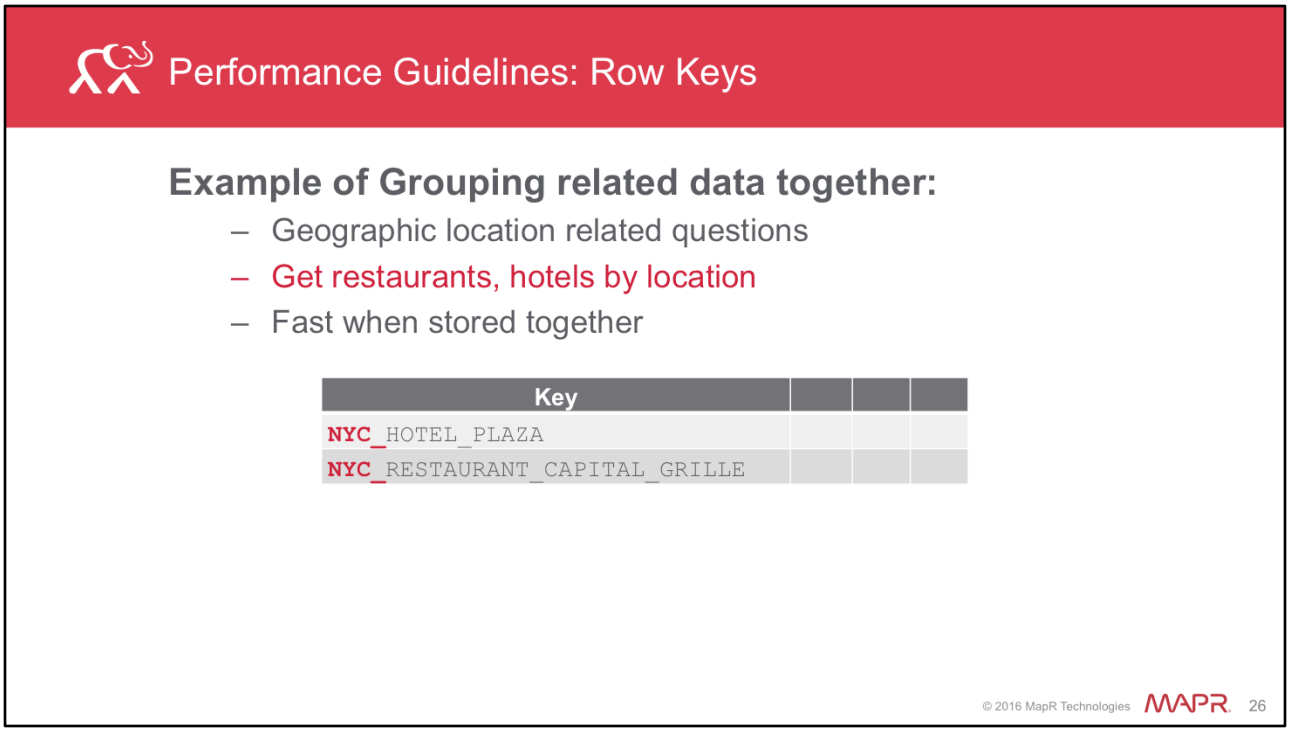
Grouping related data together is fast for data that should be read together. An example of a good way to group your data together by row key is when you want to scan by a partial key of related web URLs, like com.mapr, in order to read a counter of likes per page.



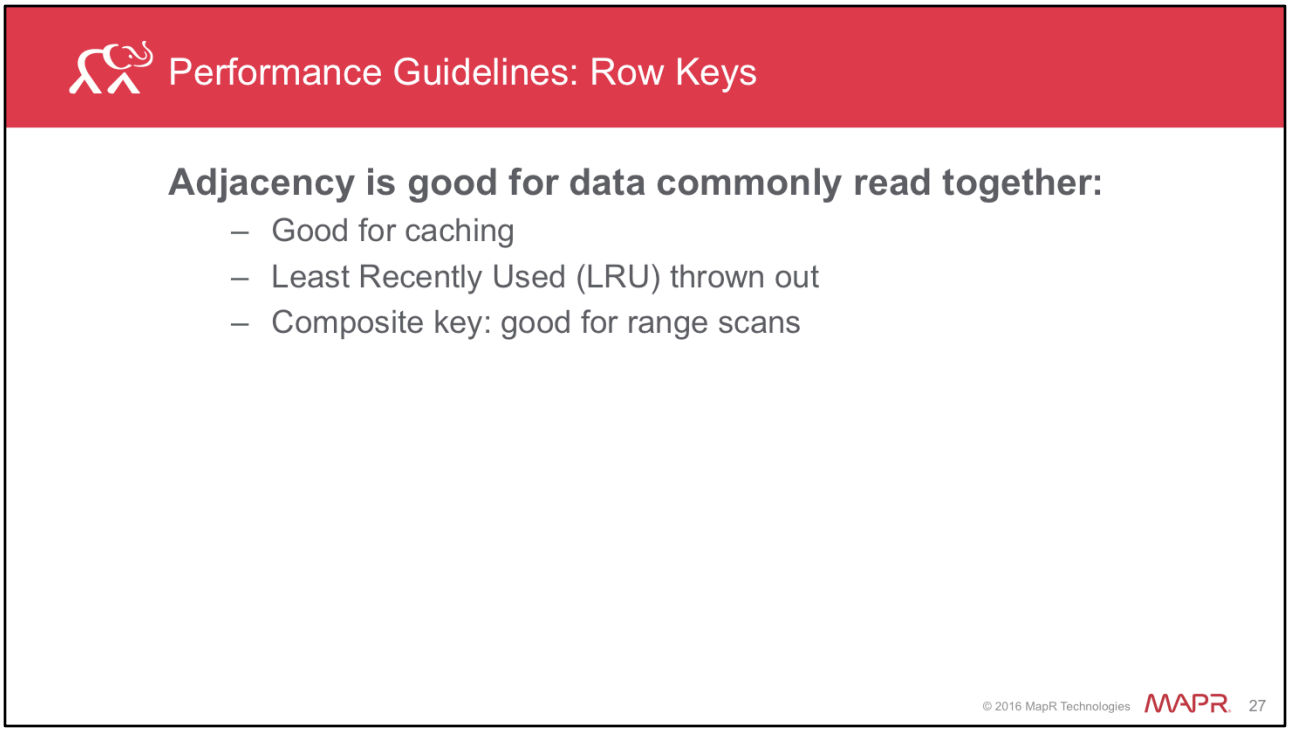
Another example of a good way to group your data together by row key is when you want a grouped and sorted retrieval.



In this example, you want to read your data sorted by stock symbol and most recent transaction time.



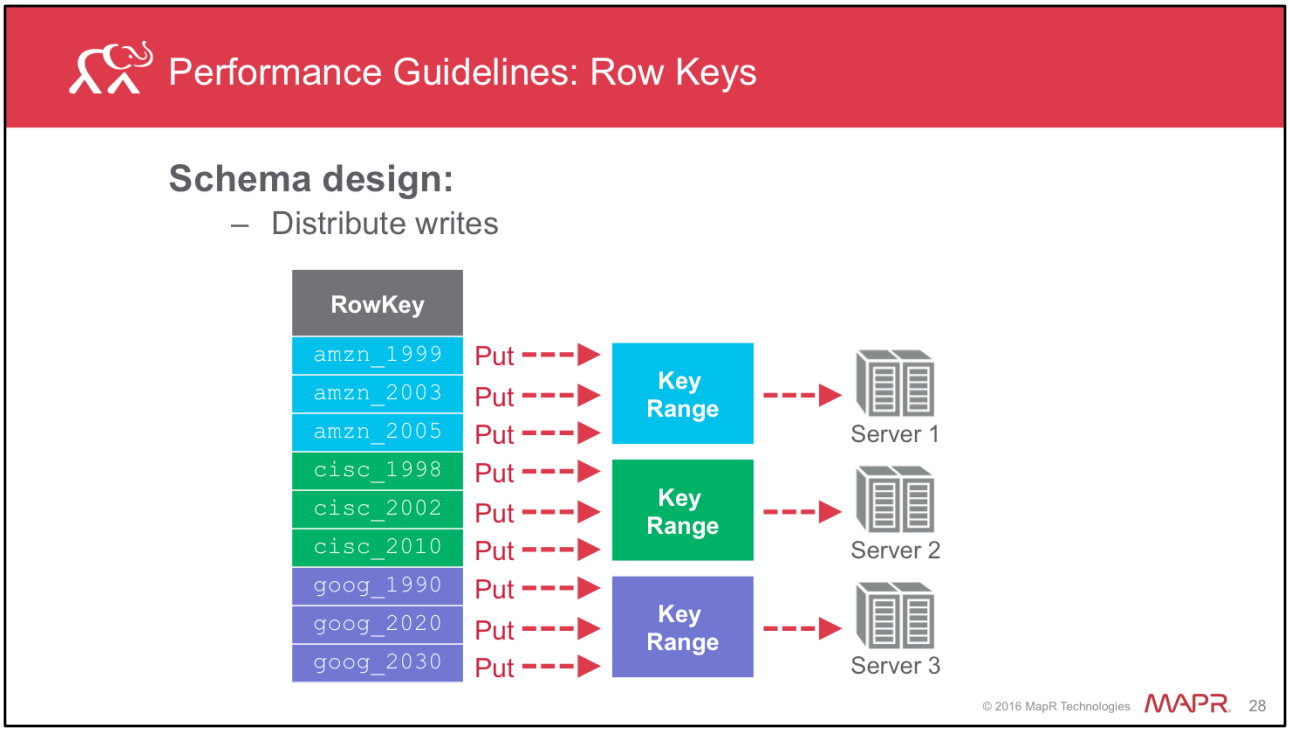
Another example of a good way to group your data together by row key is for geographic related questions, such as what restaurants or hotels are near a particular city. In this example, travel related data is grouped by city making queries for hotels or restaurants by city fast.



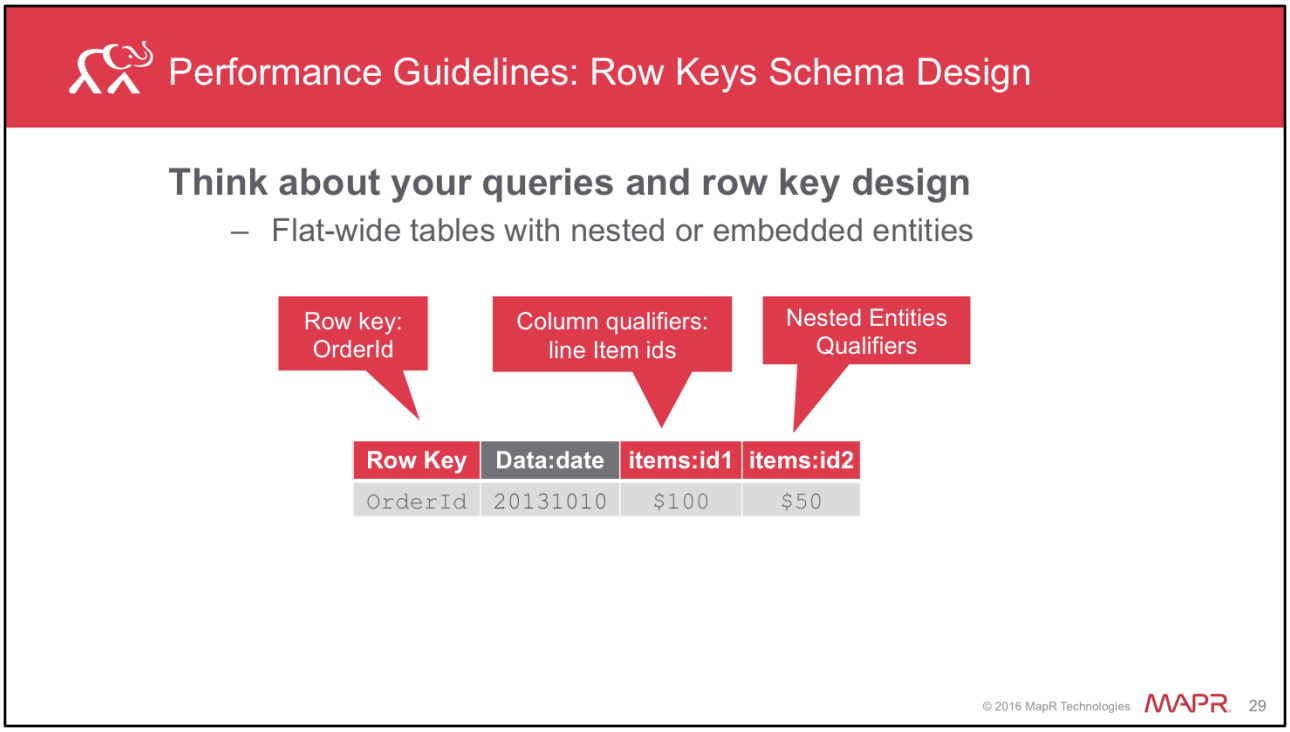
Grouping related data that is frequently read together is fast, especially for caching. Reading from memory is always faster than from disk.

Memory column families will remain in memory, with the Least Recently Used being taken out when the cache is full.

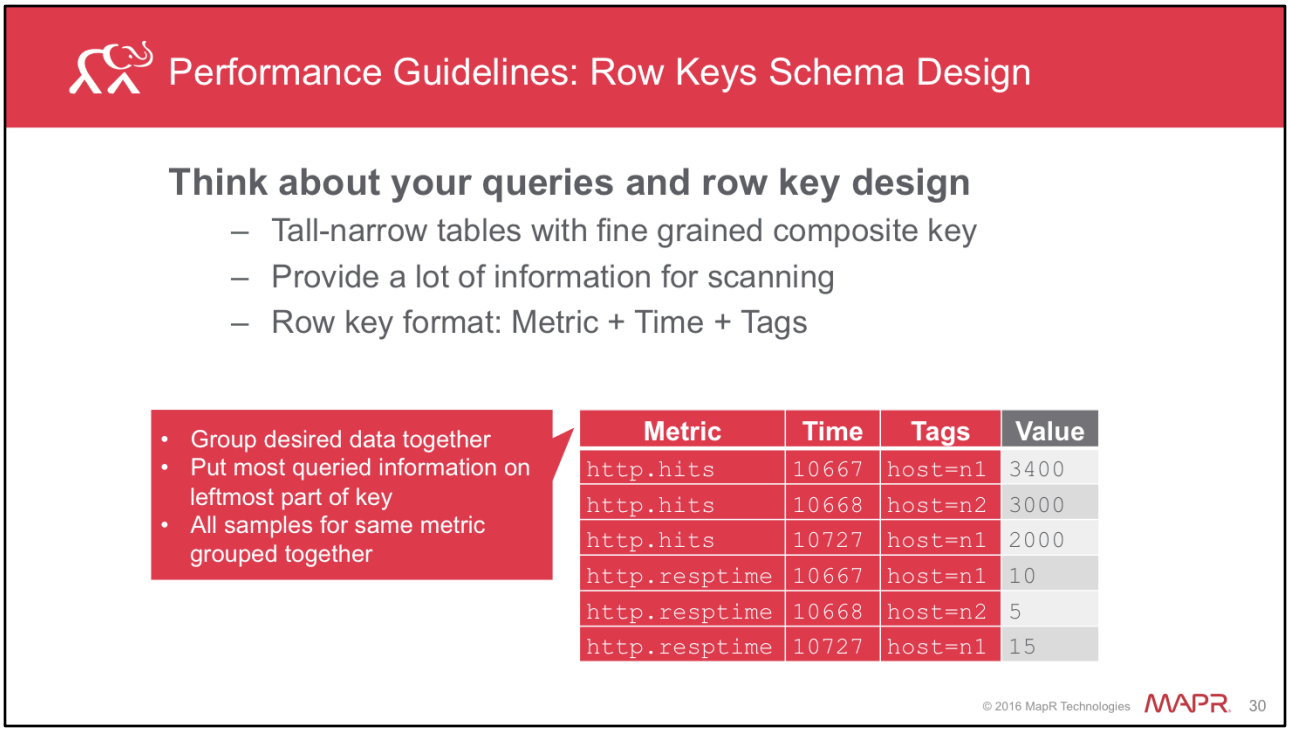
Composite keys are good for fast range scans, as full table scans will fill memory quickly, and will not benefit from caching.



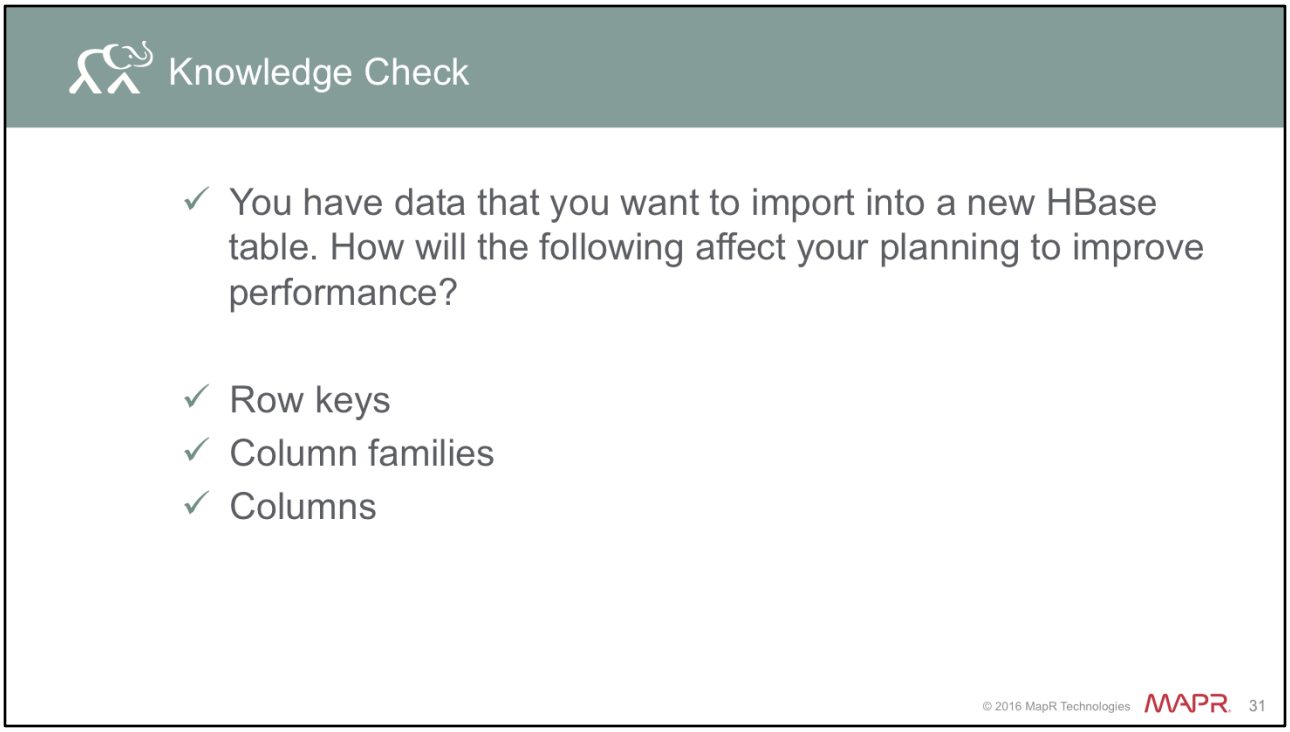
When you are grouping data together by row key that you should make sure your row keys distribute writes to avoid hot spotting. The example shown here uses the stock symbol as a row key prefix to avoid sequential writes.



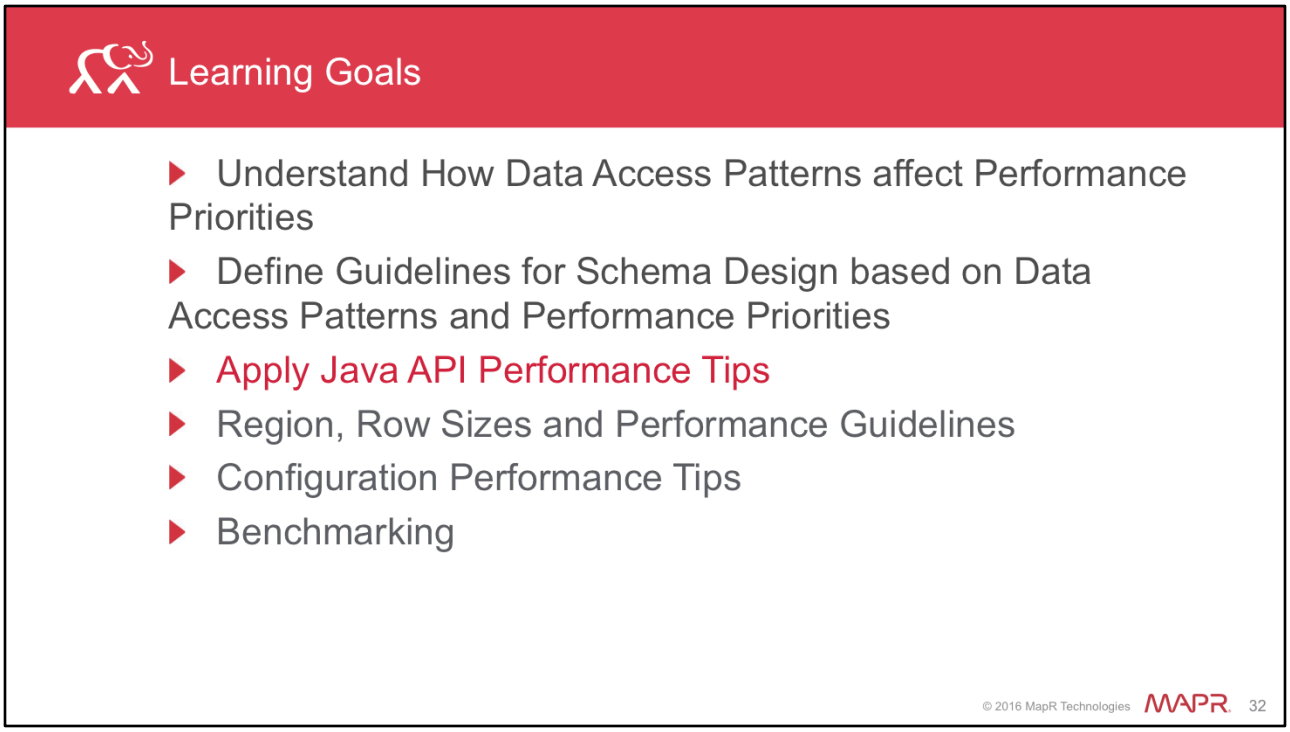
For fast gets, it's possible to model one-to-many relationships in HBase as a single row. This is referred to as nested or embedded entities.
In this example, the order and related line items are stored together and can be read together with a get on the row key.



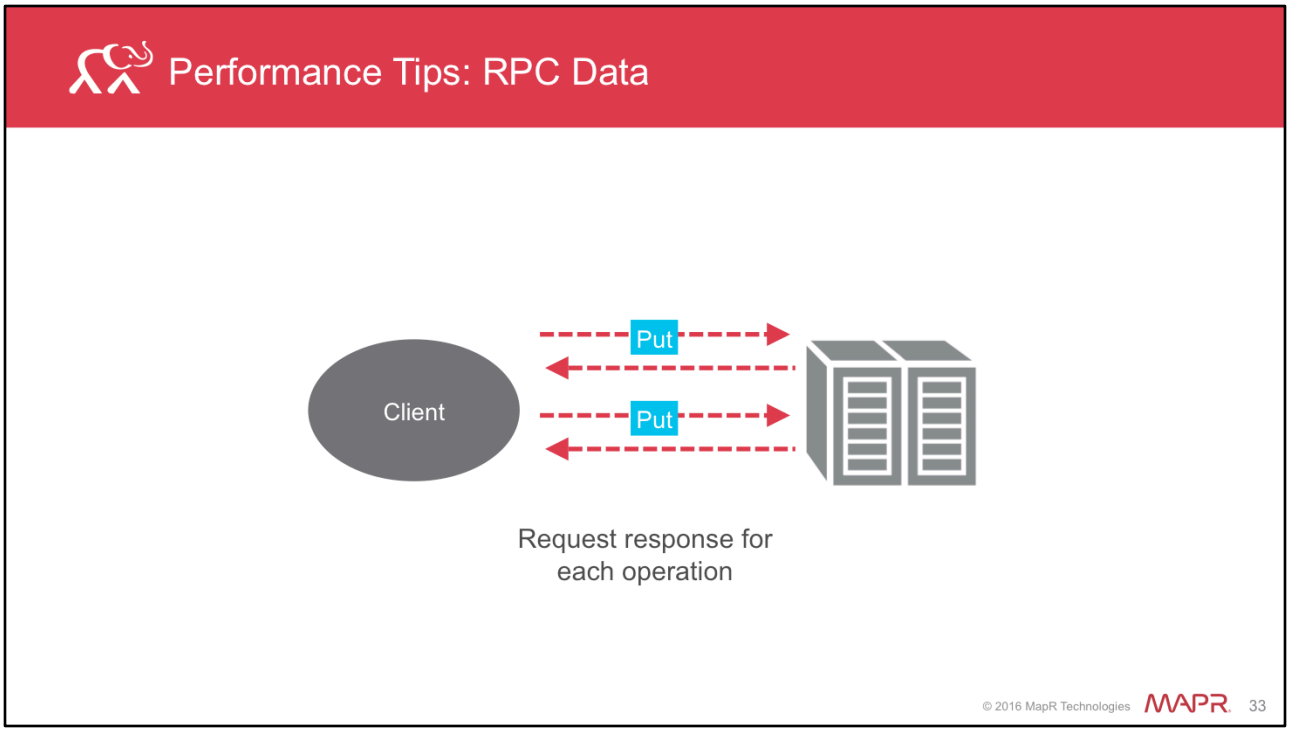
For fine grained scanning you can put a lot of information in the row key in a tall table, like in the OpenTSDB example shown here, which we discussed in the schema design lecture.



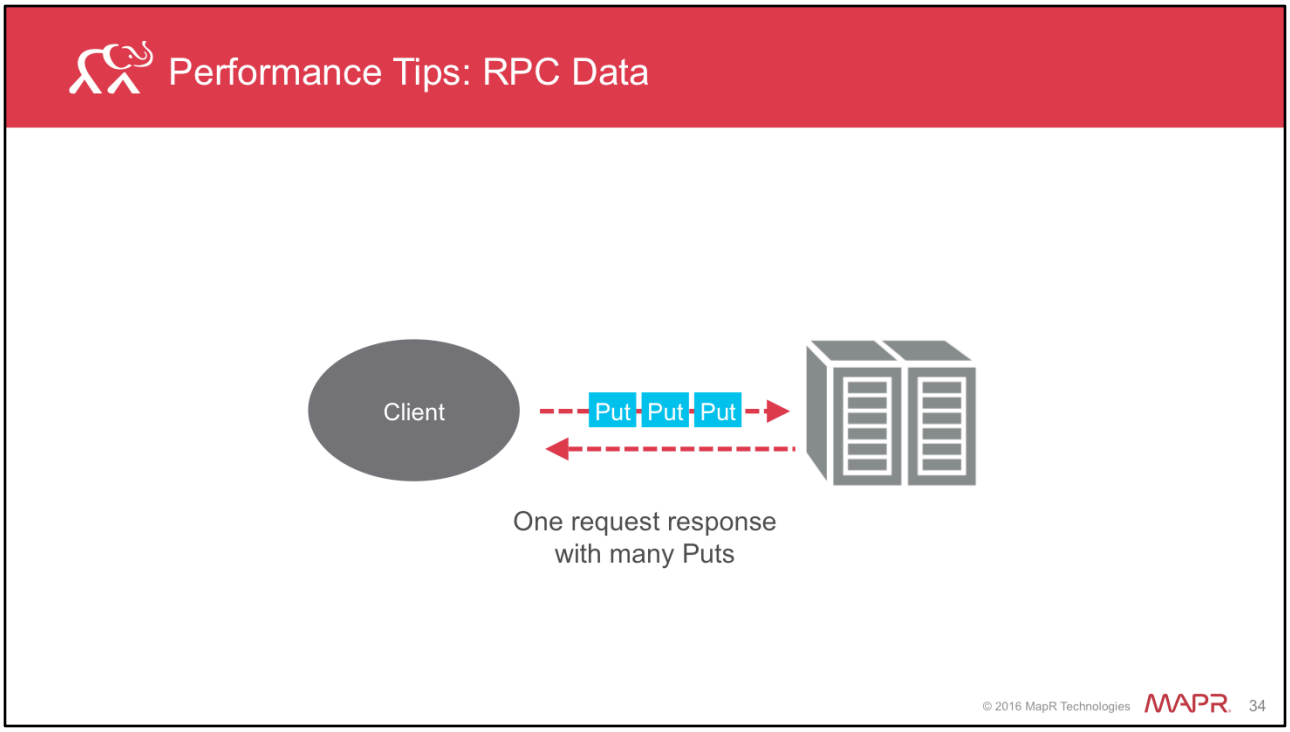
Row keys – Design your row keys to do two things; distribute the data throughout the cluster to avoid hot spotting, and use row keys to group common searches in the same location for faster retrieval.
Column families – group commonly searched content into column families to reduce the amount of data that is transferred.
Columns – keep your commonly searched columns in memory for faster searches.



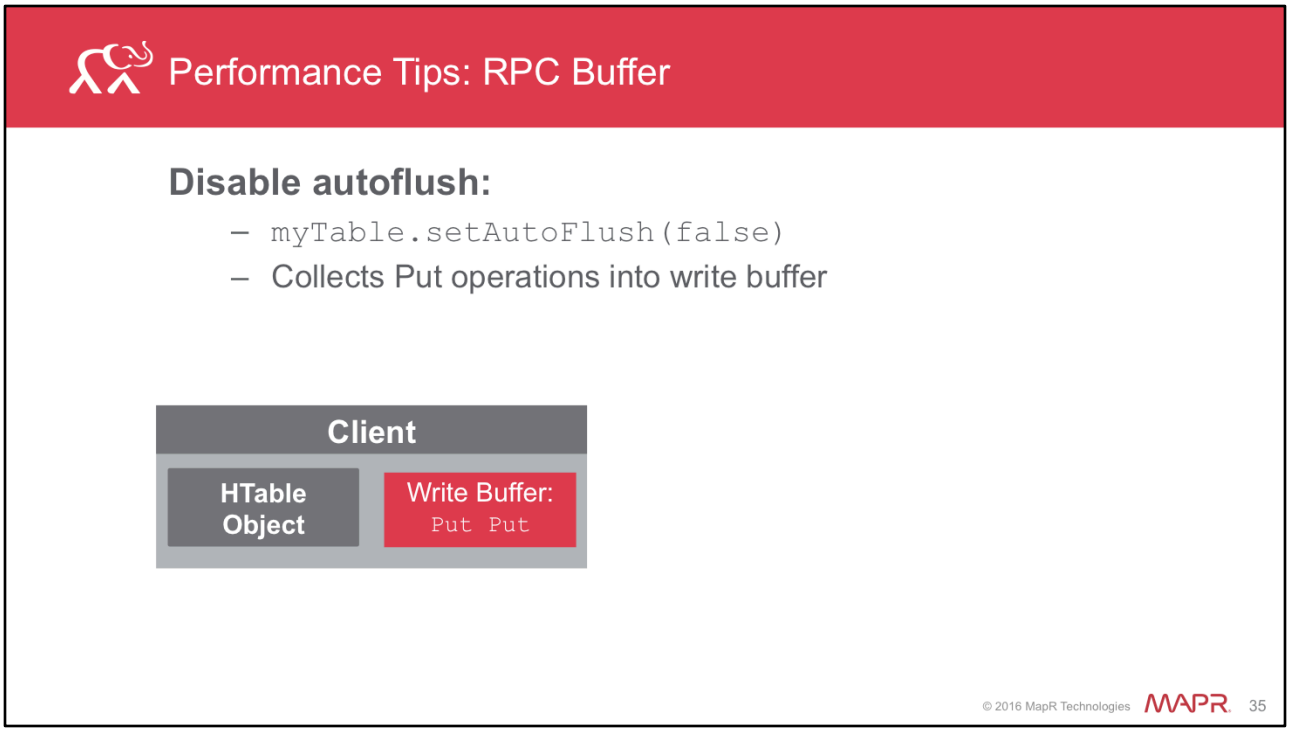
In this next section, we will discuss how to apply some performance tips when using the Java API. We will be reviewing content from DEV 330 and 335. You can refer back to these courses for more detail.



Network round-trips are expensive. Each single operation means a trip from the client to the server and back.

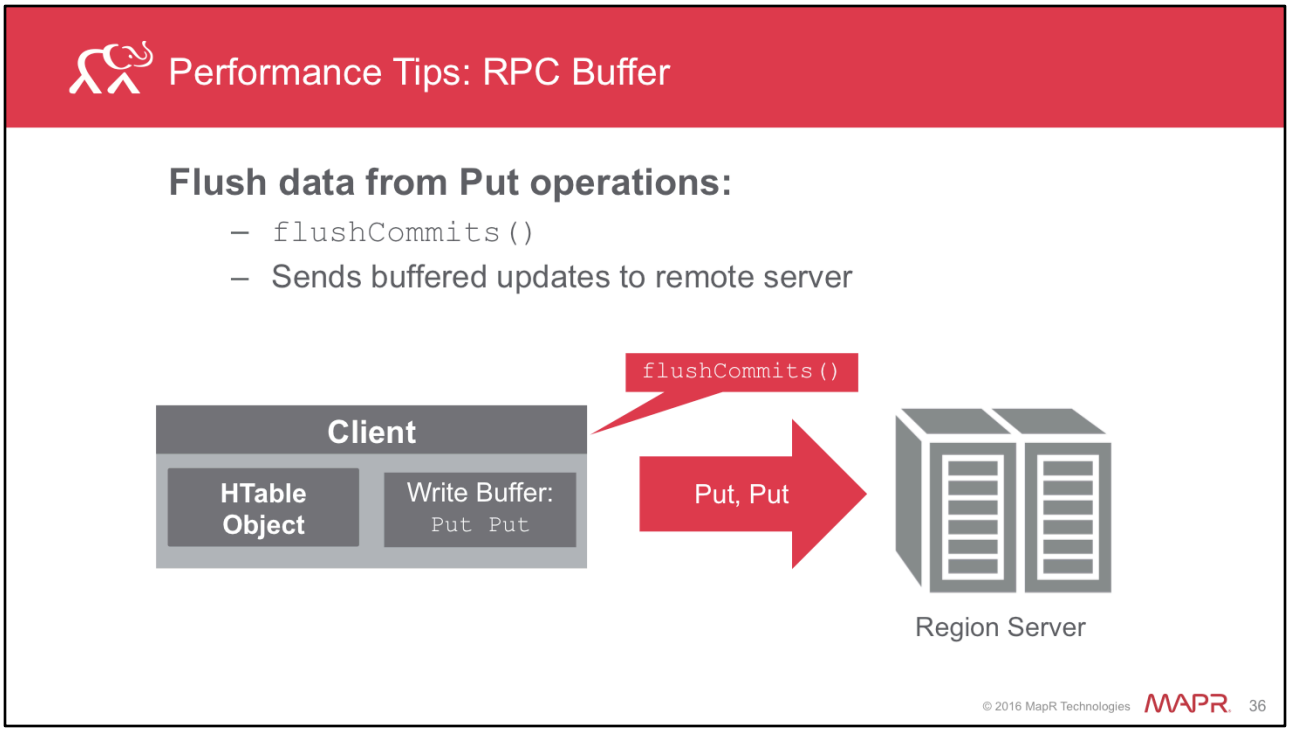


When updating small records you will get better performance by grouping puts into fewer RPCs. For data throughput, it is better to have fewer RPC calls with more data compared to lots of RPC calls with small amounts of data.

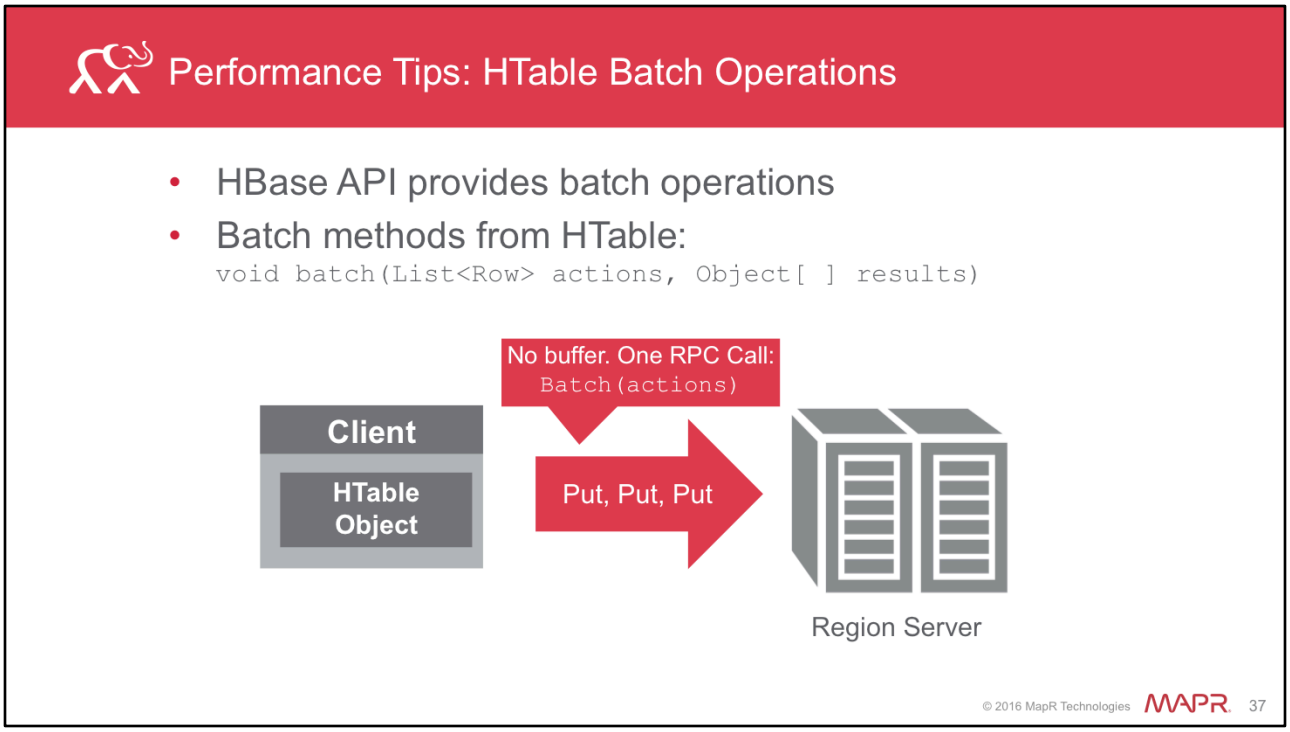


Remember for HBase puts, by default, each call will be sent one at a time because autoflush is on by default.

Disabling autoflush with the `HTable setAutoFlush method`, will cause put objects to be collected in the write buffer.



The puts from the write buffer will be sent either when flush commits is called, or when the write buffer is full.



Another way to get better performance with fewer RPCs is to use the HTable Batch methods. These methods let you pass a list of objects for updating or reading, which will be sent in one RPC call.

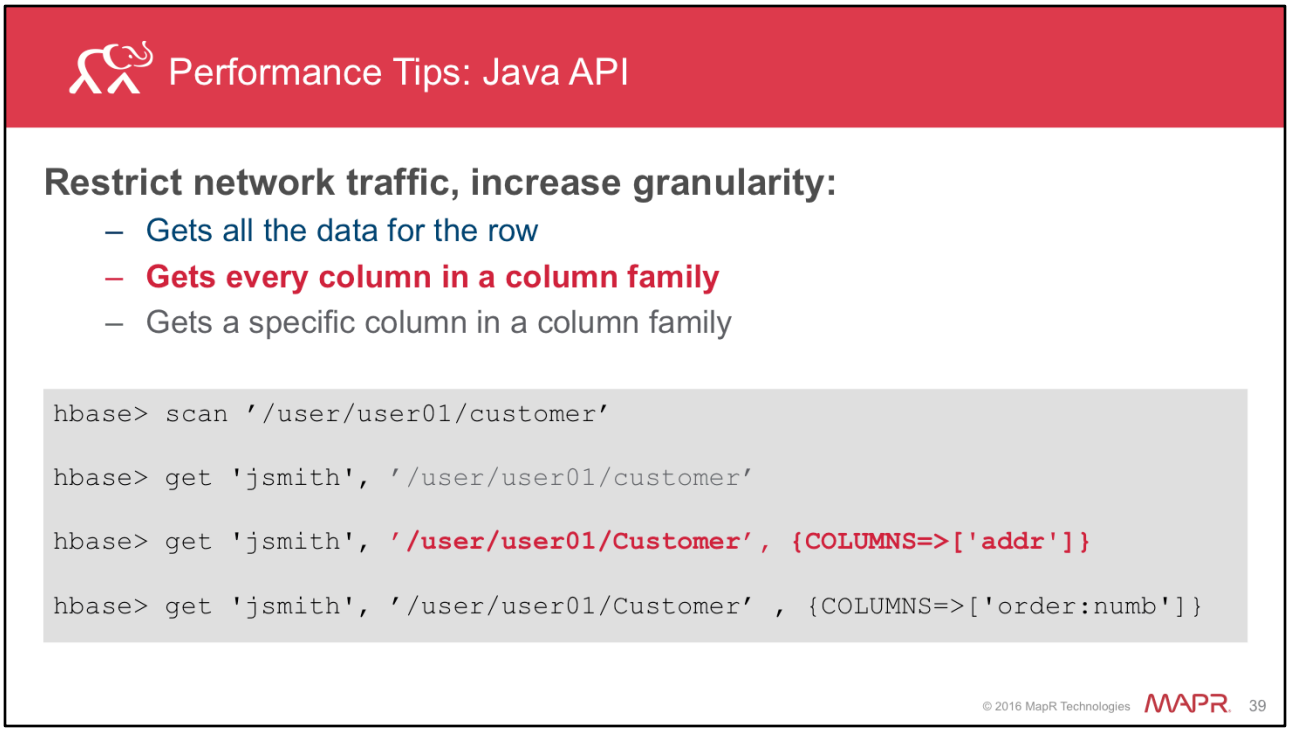
Performance Tips: Java API

Restrict network traffic, increase granularity:

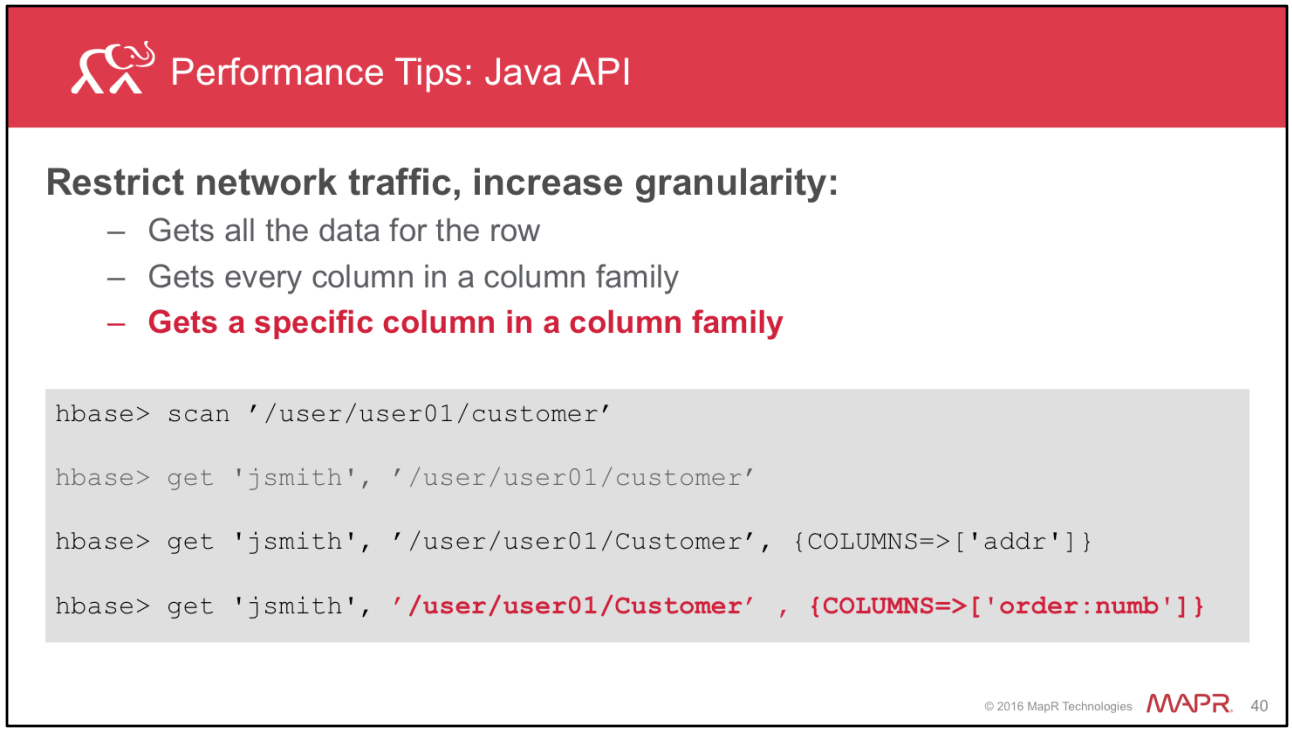
- **Gets all the data for the row**
- Gets every column in a column family
- Gets a specific column in a column family

```
hbase> scan '/user/user01/customer'

hbase> get 'jsmith', '/user/user01/customer'

hbase> get 'jsmith', '/user/user01/Customer', {COLUMNS=>['addr']}

hbase> get 'jsmith', '/user/user01/Customer' , {COLUMNS=>['order:numb']}
```

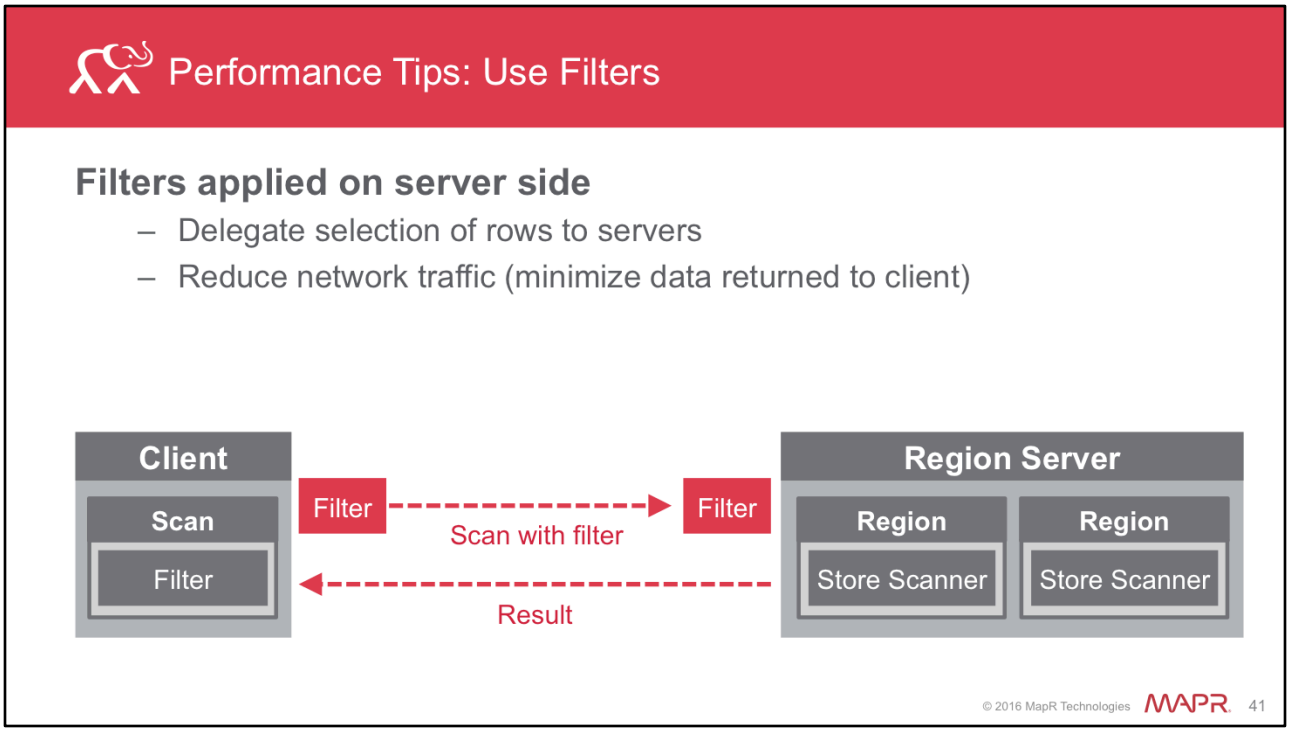

Remember with a Get or Scan you should narrow down the data returned, otherwise all of the rows will be returned.



If you know the data that you are interested in, you can specify the column family that you want. This will reduce the data transferred by returning just that column family.

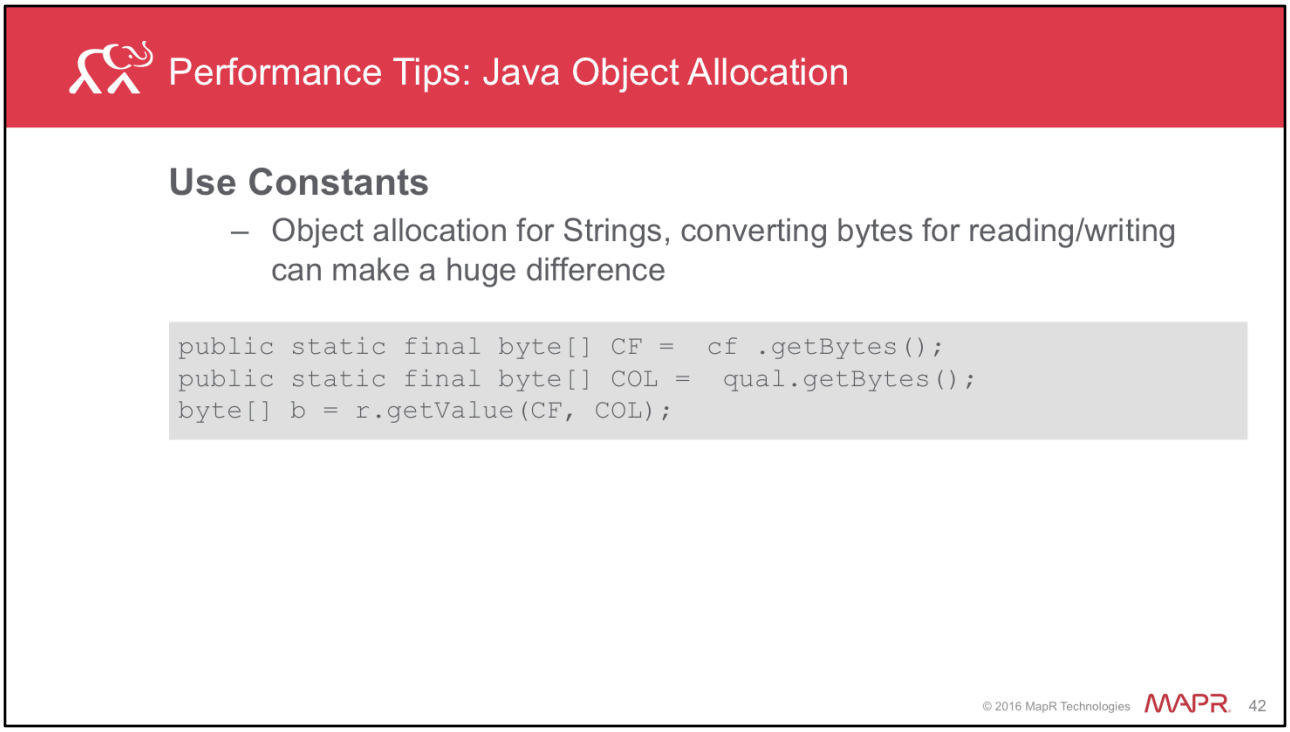


And you can further reduce the data, if you specify the columns you want. You will get better performance by defining the columns you want with addColumn(), transferring less data to the client.

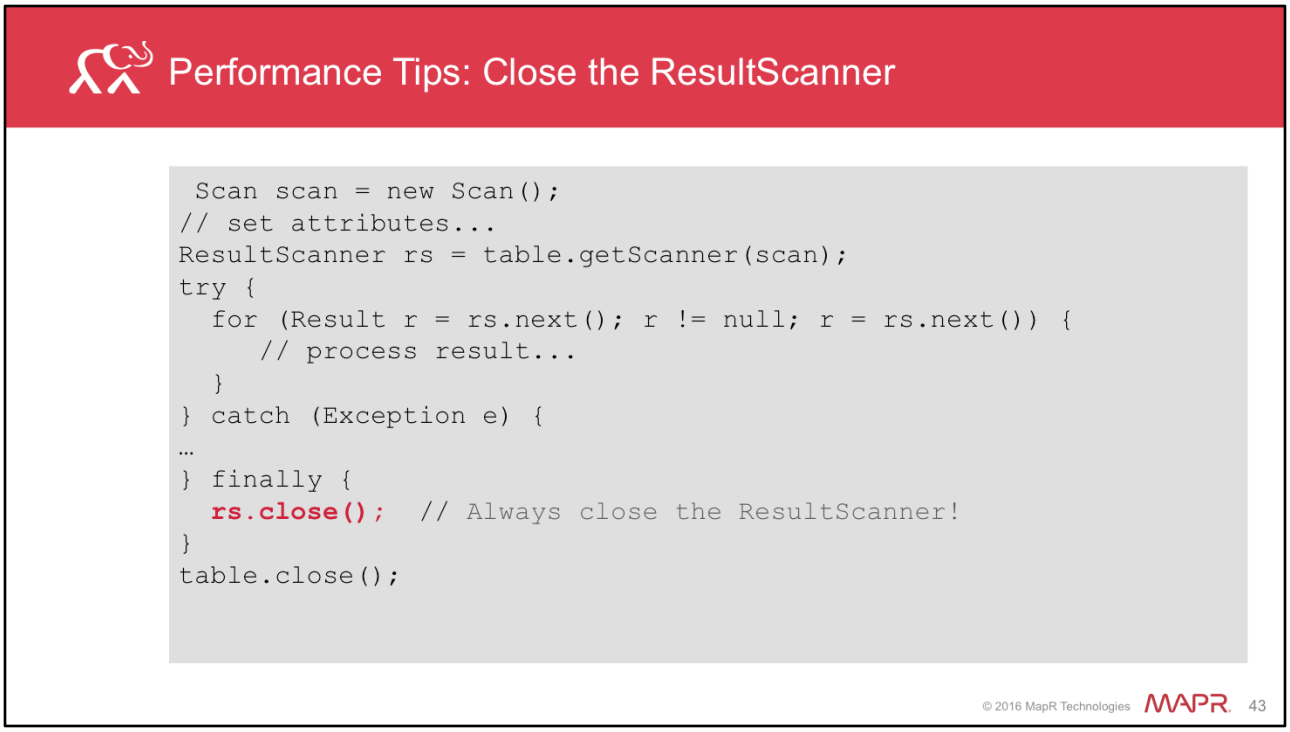


You can use filters to reduce the amount of data transferred to the client.

On the server side, the filter is used to determine whether a record should be returned back to the client side.

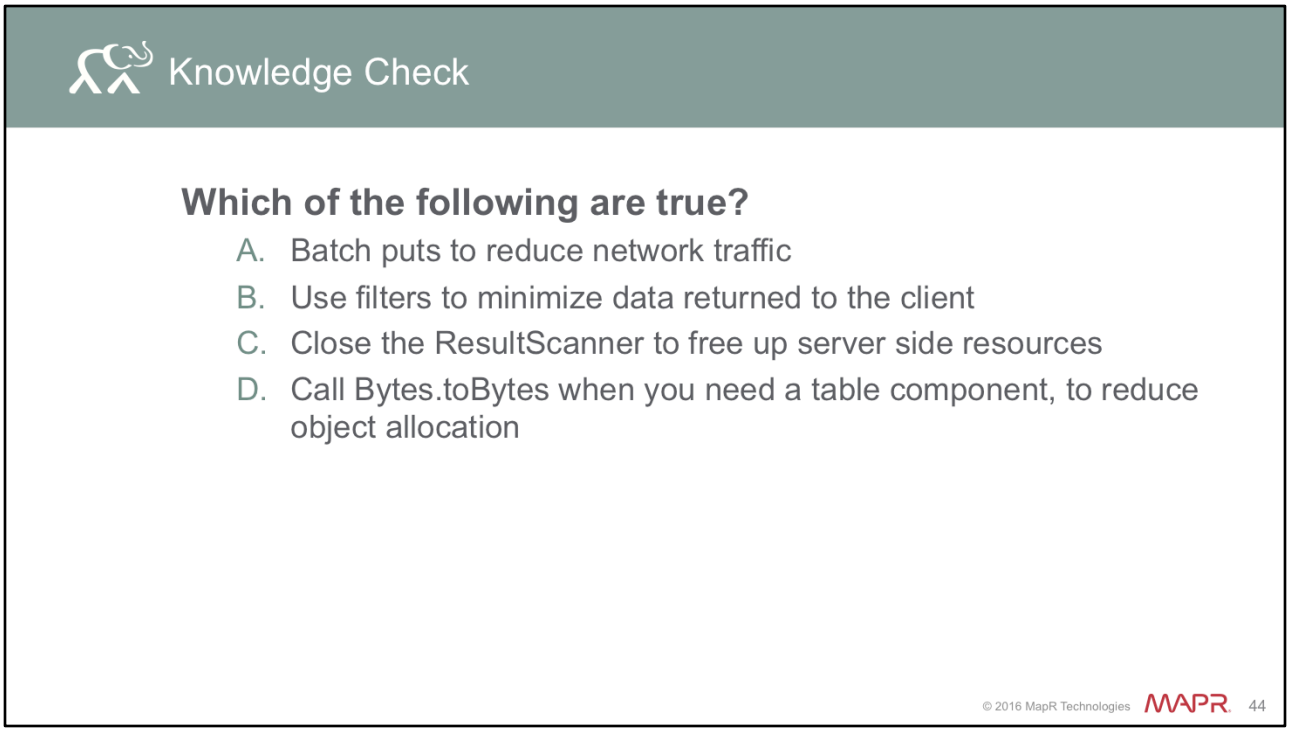


Reducing object allocation for converting bytes for reading/ writing to HBase can make a huge difference in performance. You can reduce object allocation by setting up such variables once with “public static final“ instead of calling Bytes.toBytes every time you need to specify a table component. This saves resources and is more efficient.



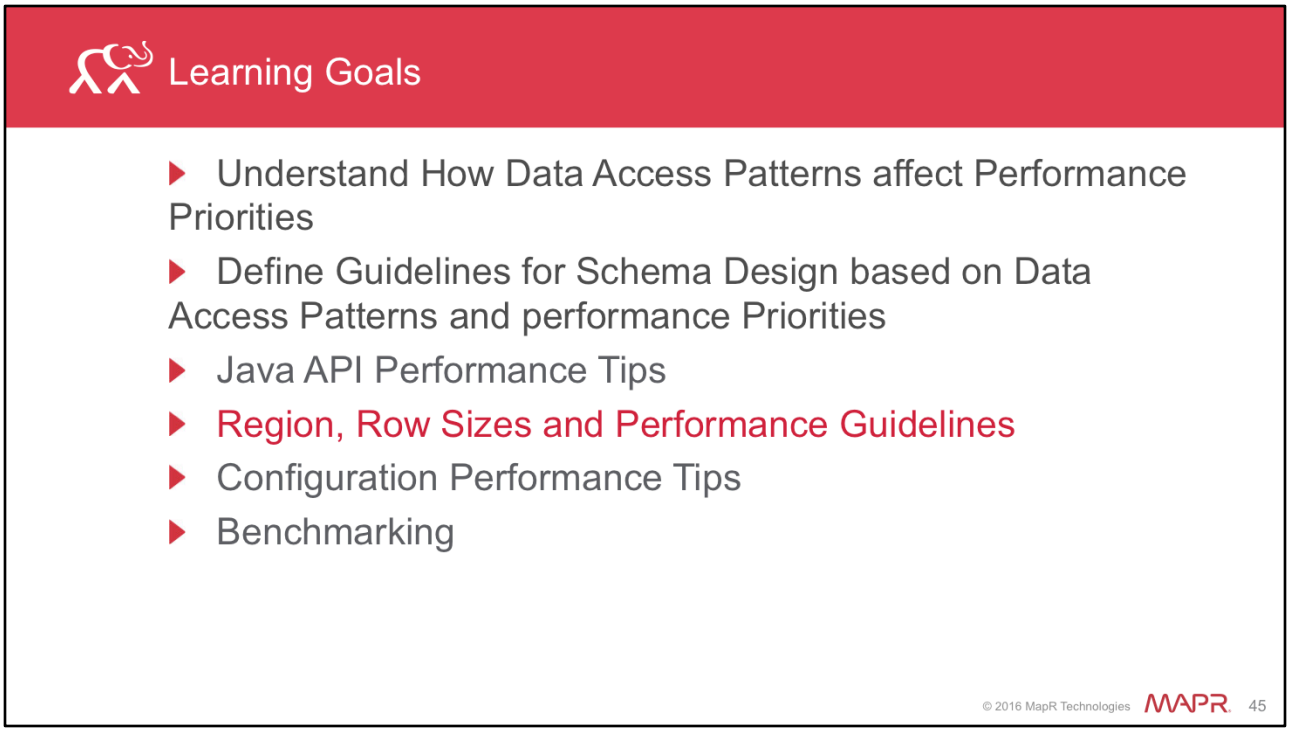
You should close the ResultScanner to release resources which are no longer needed. You should do this so that you don't create performance issues on the server side.

To avoid performance problems always have ResultScanner processing enclosed in the finally clause of try/catch blocks. This means it will always be called, if an Exception is thrown or not.

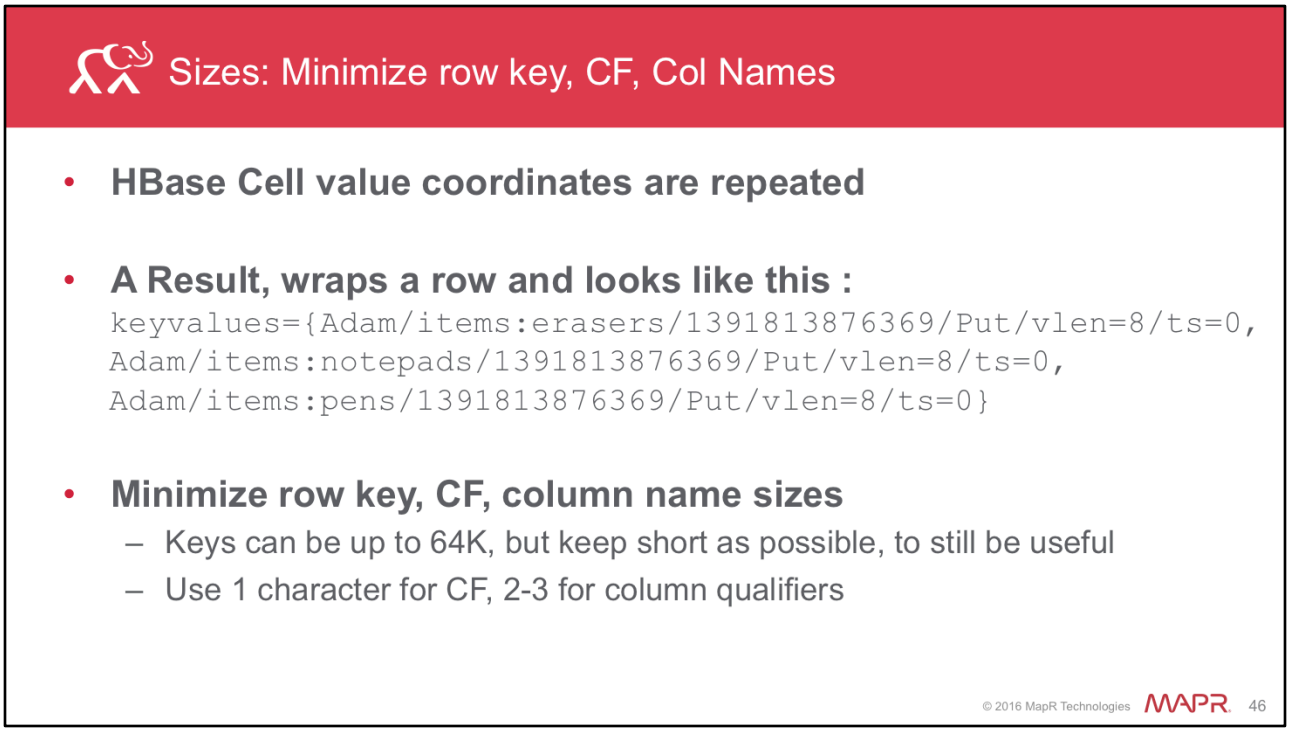


A. Batch puts to reduce network traffic – true
B. Refine gets to reduce amount of data returned - true
C. Close the ResultScanner to free up server side resources - true
D. Call Bytes.toBytes when you need a table component, to reduce object allocation – false

Batching RPC calls and refining the data returned to the client are ways to reduce network traffic, and speed up your applications. Make sure to close the ResultScanner in the 'finally' clause of a try/catch block, to release resources which are no longer needed. Reduce object allocation by setting up such variables once with "public static final" instead of calling Bytes.toBytes every time you need to specify a table component.



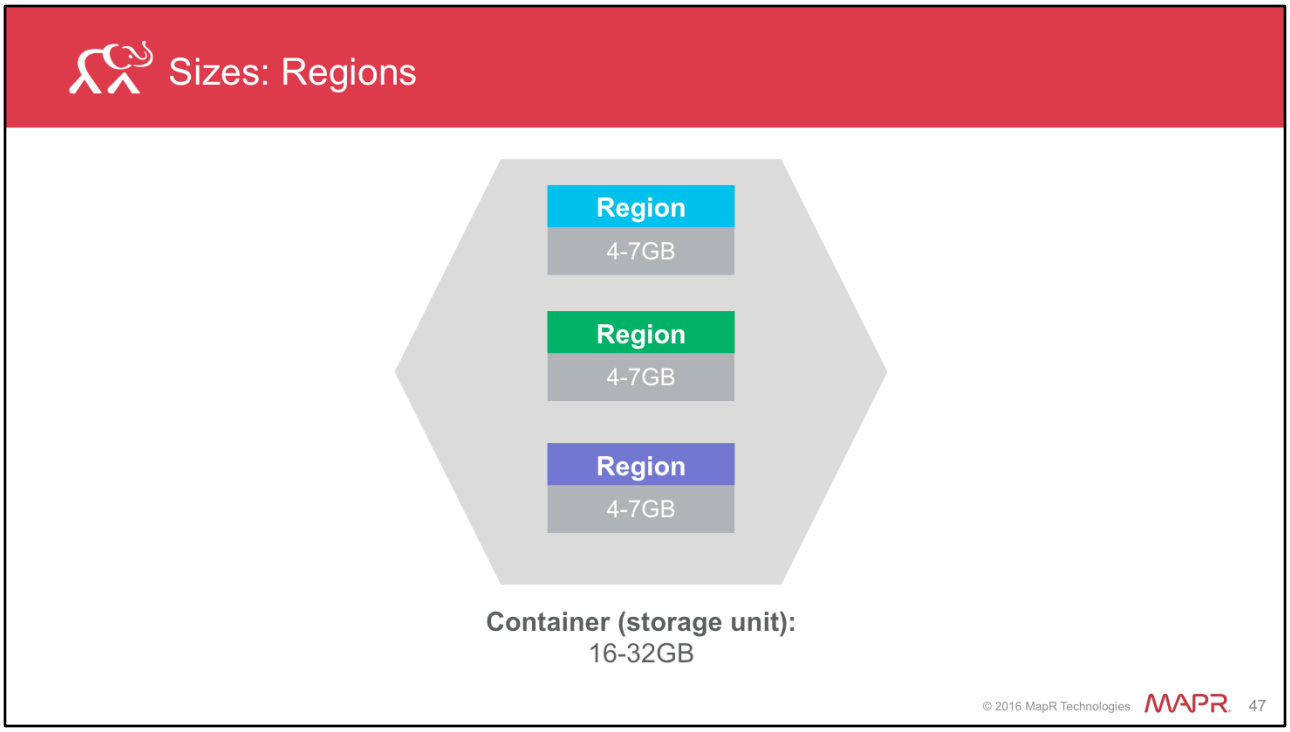
In this next section, we will discuss the different sizes and performance guidelines for HBase components.



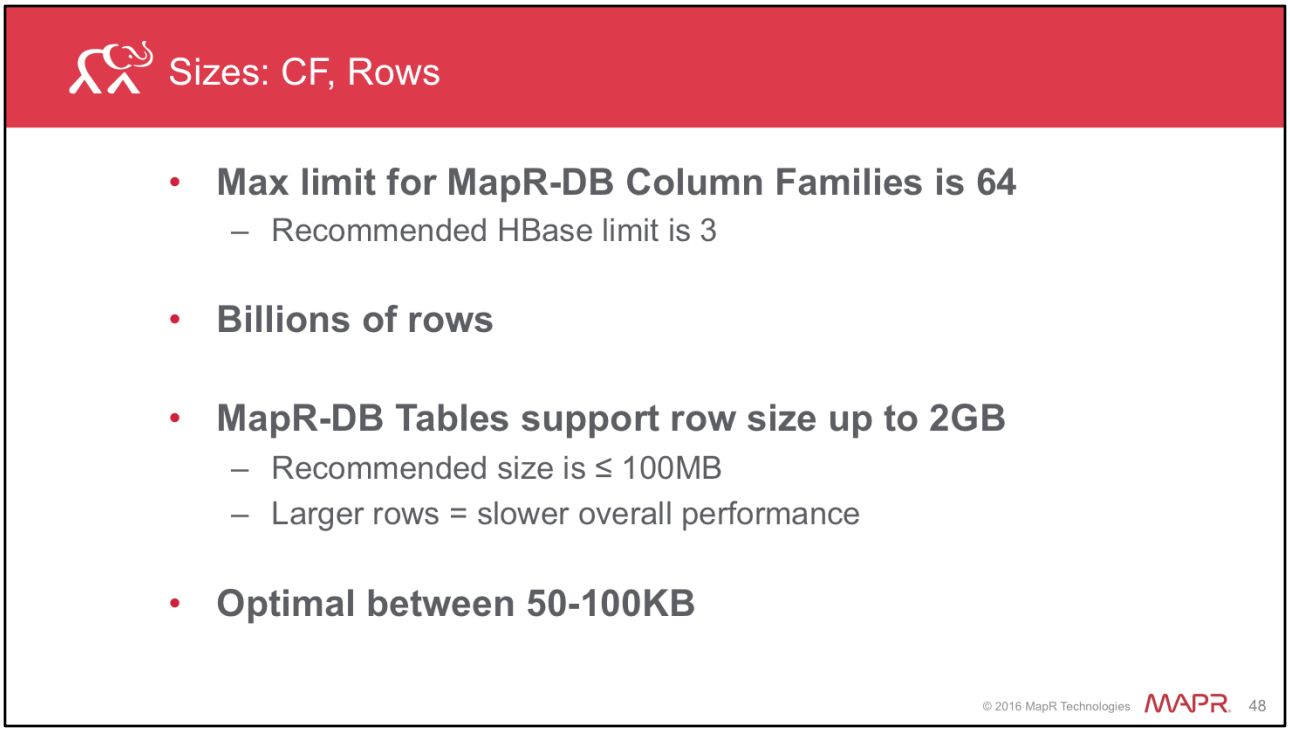
Cell value coordinates are repeated take up space in the RPC calls, memory, indexes, and on disk. Therefore, you should make the row key, column family names and column names as short as possible.

- You can use a lookup table to keep names short and meaningful, like in the opentsdb example discussed in schema design

Note MapR-DB does not store the full coordinates for each cell on disk like in HBase, but size still matters. MapR-DB does not repeat the key or the CF details, while HBase does. So, the number of bytes transferred from disk to find a subset of columns in a row is far less compared to HBase. Consequently the in-memory footprint in MapR-DB is much smaller, leading to more efficient use of the memory. MapR-DB does store the column qualifier as part of data.



A default MapR container has a maximum size of 32 GB when full. MapR-DB regions are 4 to 7 GB, which leads to more than one region in a container.



The limit for the number of Column Families for a MapR-DB Table is 64, with HBase this recommendation is 3. You can have billions of rows.

- MapR-DB Tables support a rowsize of up to 2GB
- The row size limit is enforced by the amount of available memory and how much data can fit in an RPC call
- The Sweet Spot for row size is between 50-100KB

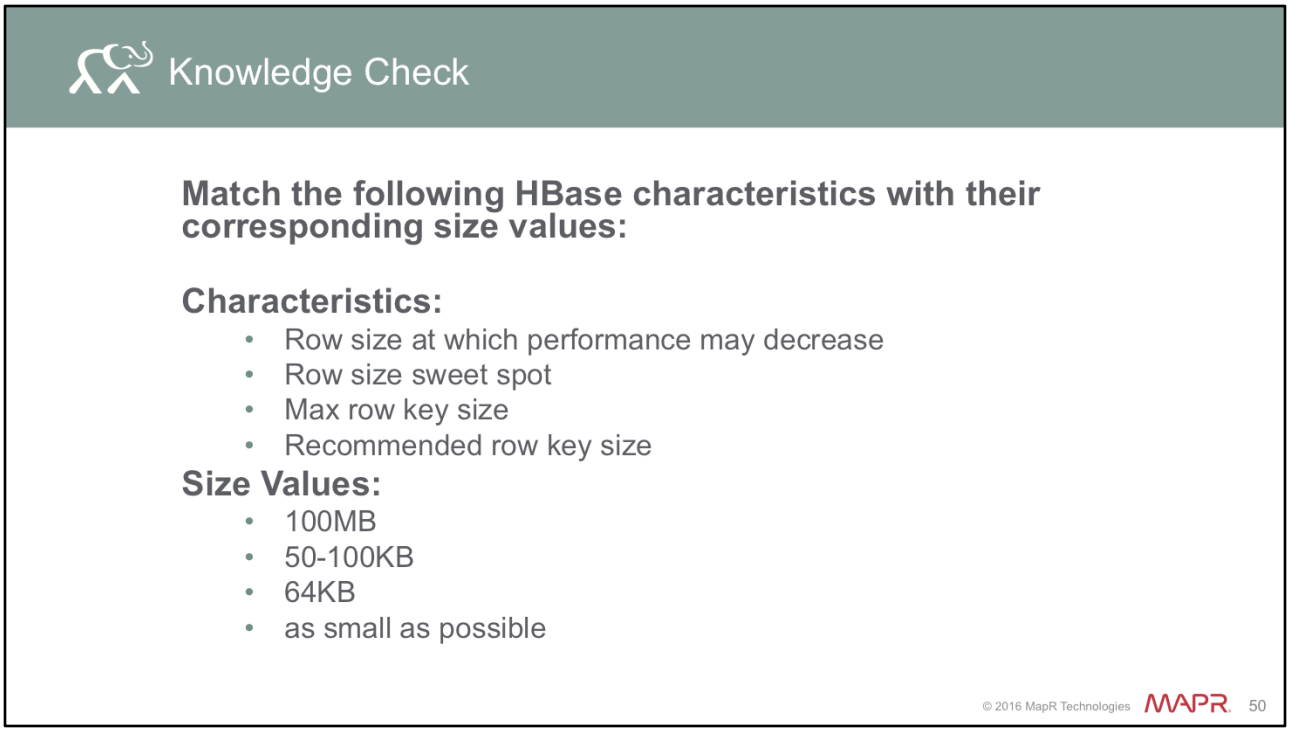
Marshalling/unmarshalling giant objects is not a good idea, otherwise Performance will drop dramatically.

MapR-DB Large Row Support

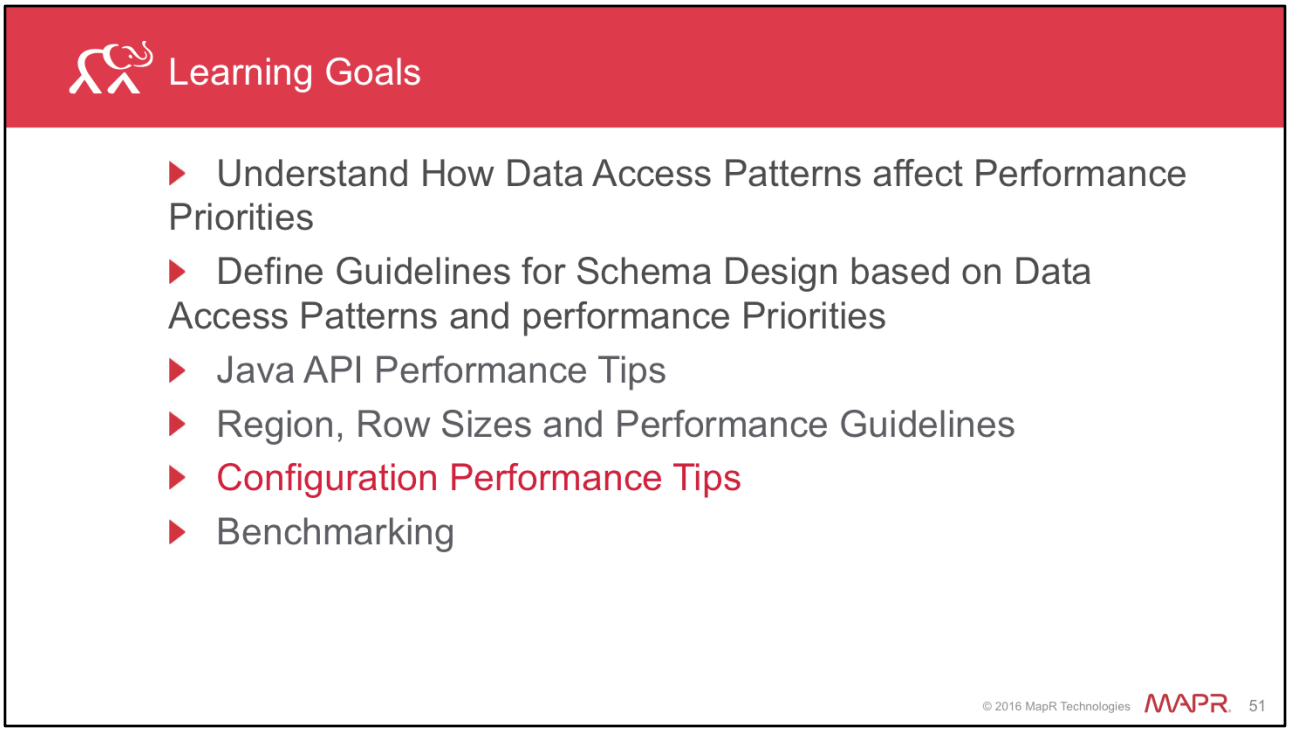
- **Configure maximum row size**
  - `mfs.db.max.rowsize.kb` set to 100MB row size:
  - `maprcli config save -values {"mfs.db.max.rowsize.kb" :"102400"}`
- **To view the current setting:**
  - `maprcli config load -json | grep mfs.db.max.rowsize.kb`

Rows in excess of 100MB may show decreased performance. You can configure this maximum by changing the value of the mfs.db.max.rowsize.kb value.

http://doc.mapr.com/display/MapR/Setting+Up+MapR-FS+to+Use+Tables#SettingUpMapR-FStoUseTables-ConfiguringMaximumRowSizesforMapRTables



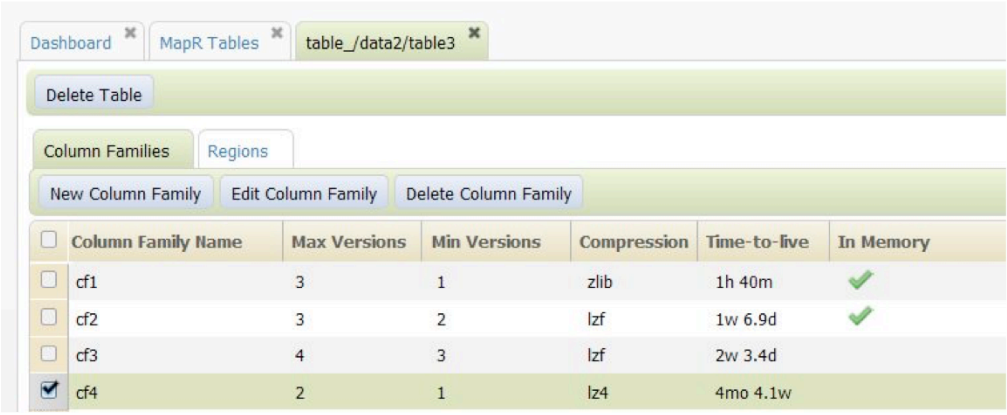
Row size at which performance may decrease::100MB
Row size sweet spot::50-100KB
Max row key size::64K
Recommended row key size::as small as possible



In this next section, we will discuss tip for configuring HBase for better performance.

Performance Tips: Column Families

- You can set column family attributes using MCS
- Compression, MaxVersions, Time-to-live, In-memory

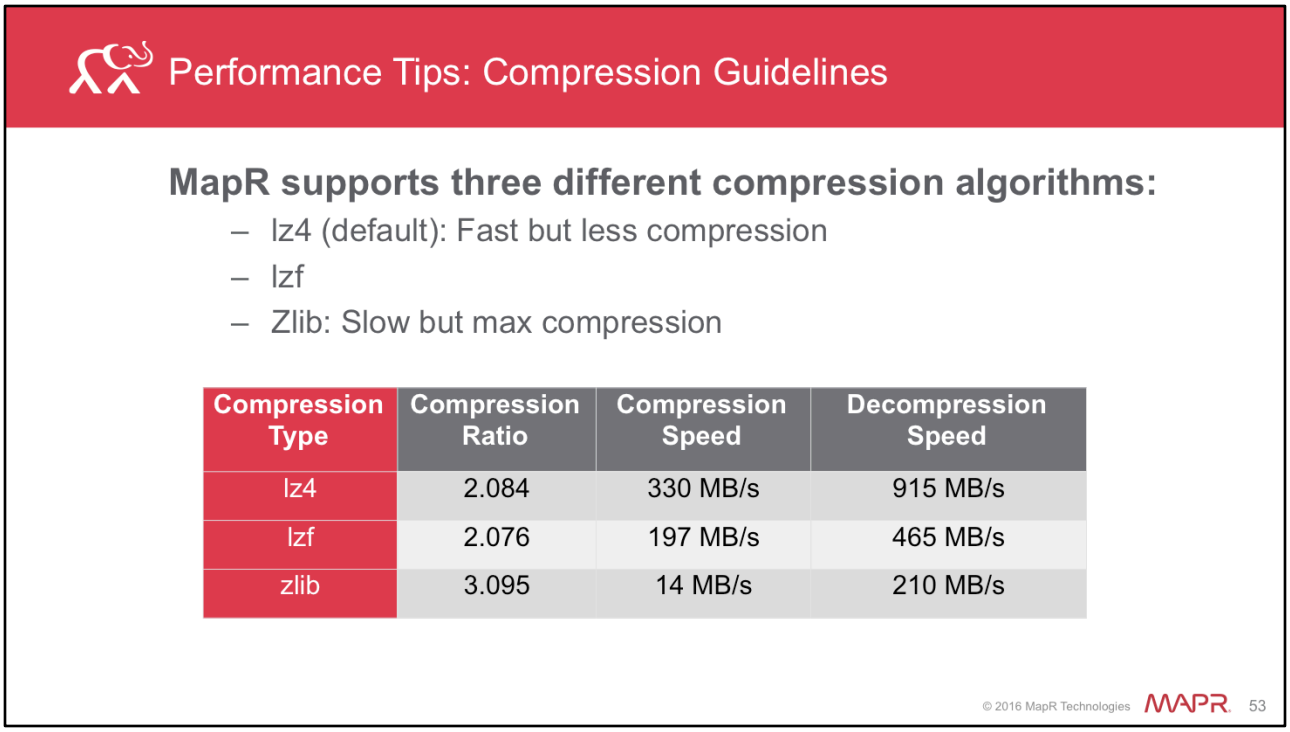


 52

You can use the MapR control system, the HBase shell, or the Java API to set properties for column families.
You can set the Max Min versions, Compression type, time to live, and the in memory setting, whether the column family is kept in memory or not.
Refer to DEV 330 for more details.

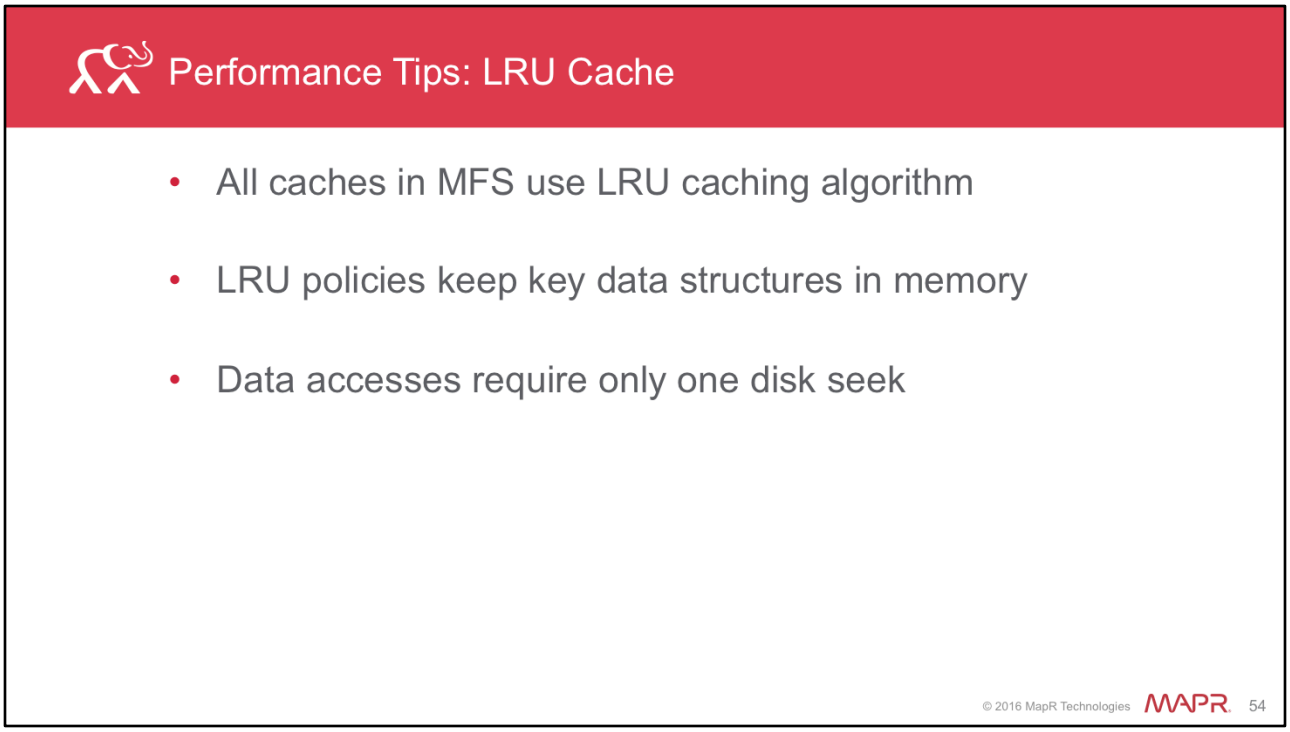
Time-to-live TTL
is specified in seconds and is, by default, set to Integer.MAX_VALUE or 2,147,483,647 seconds that's 2 billion 147 million etc 2^31
If a value exceeds its TTL it is dropped and this is based on the timestamp. This is in addition to the number of versions that get kept.

In-memory
Defaults to false
Setting it to true is not a guarantee that values are loaded into memory nor that they stay there. It's more like an elevated priority. This setting is useful for small column families with few



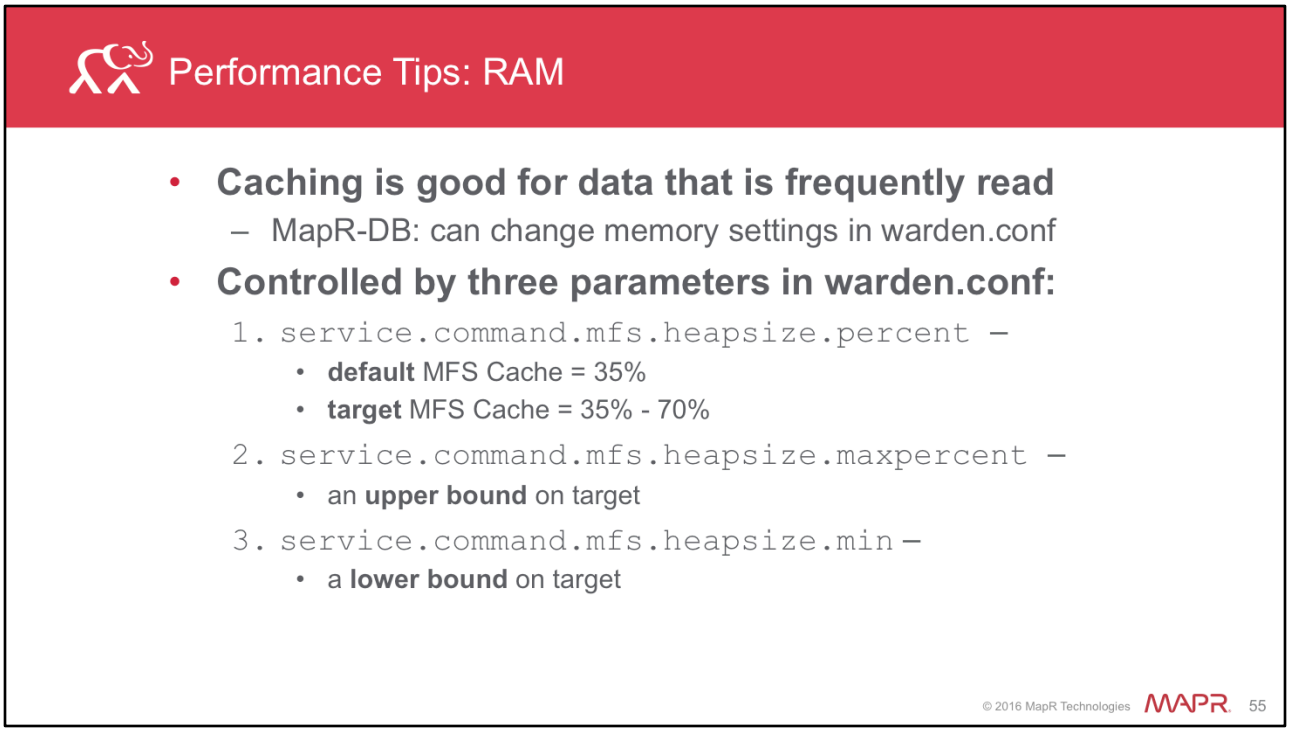
You can set the compression type on the column family. Lz4 is the default, and is faster but gives less compression, Zlib is slower but gives max compression.

There is a tradeoff between compression ratio and speed. Compressed data uses less disk space and less bandwidth on the network than uncompressed data, however for scanning files it is slower for reads.



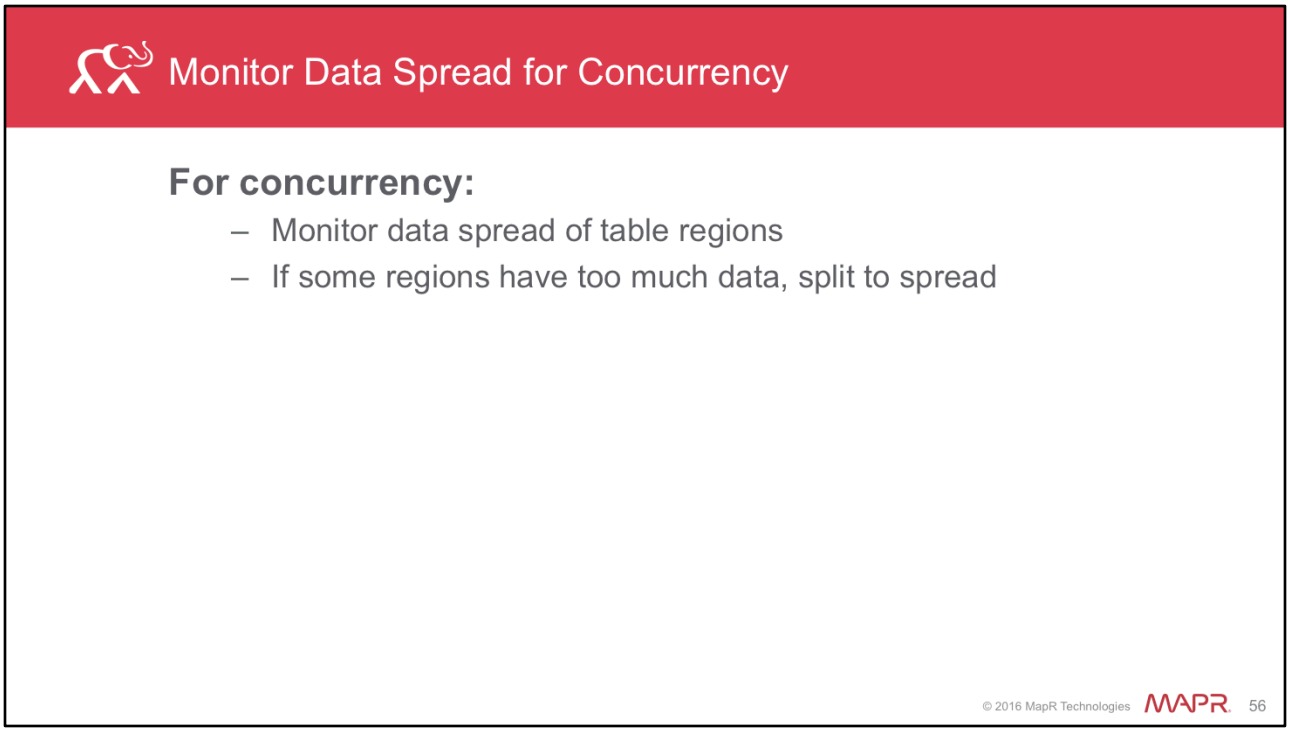
MapR uses very efficient caching techniques for all of its file system and MapR-DB data :

- There is no duplicate caching like in HBase - data is cached in the filesystem, and shared with MapR-DB
- Paging of cached data happens with Least Recently Used data being evicted. Key data structures such as indexes are kept in memory
- Using these key data structure in memory means out of memory data accesses requires only one disk seek.
- Since all reads happen from the container master node, the replicas do minimal caching of MapR-DB data



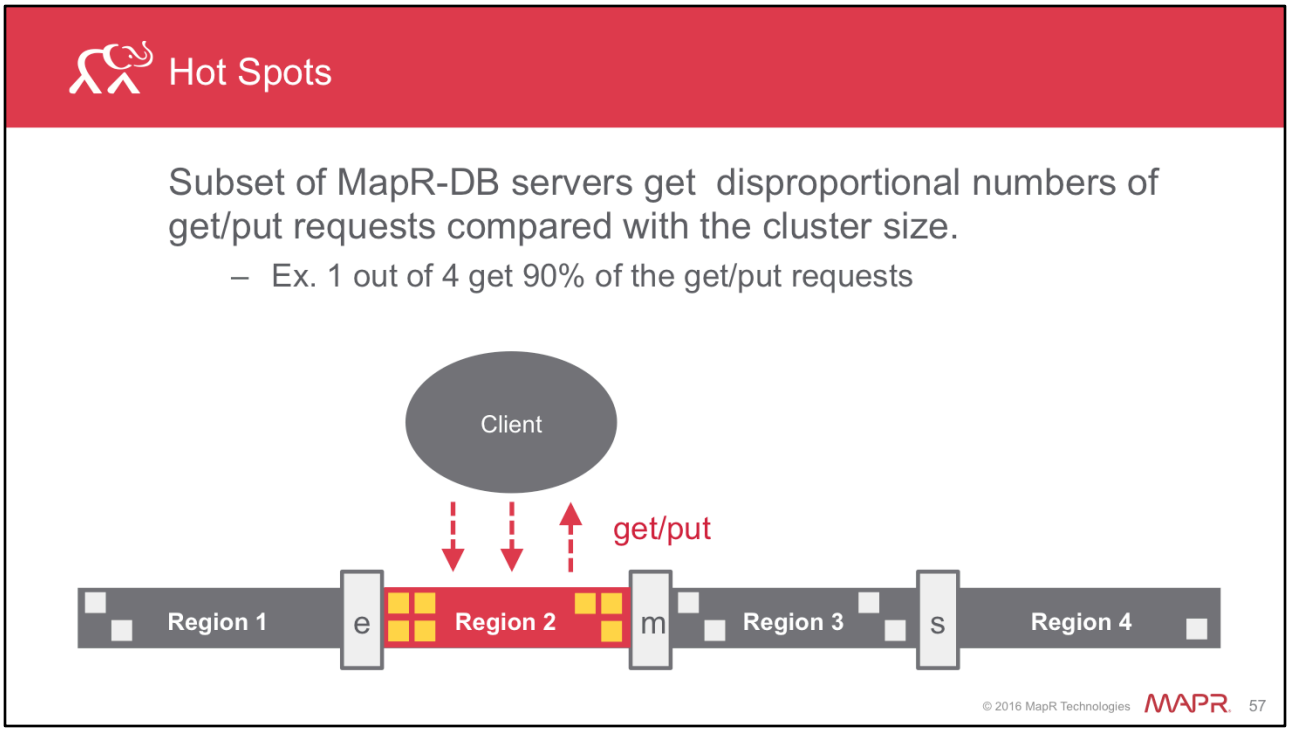
You can Increase MapR-FS memory settings in warden.conf. This is controlled by the three parameters in the warden.conf file shown here.

The target for MFS Cache is typically 35%, but for MapR-DB specific usages, especially on nodes that have very large memory and with very high performance requirements, increasing memory cache allocation as high as 70% or more may be beneficial.

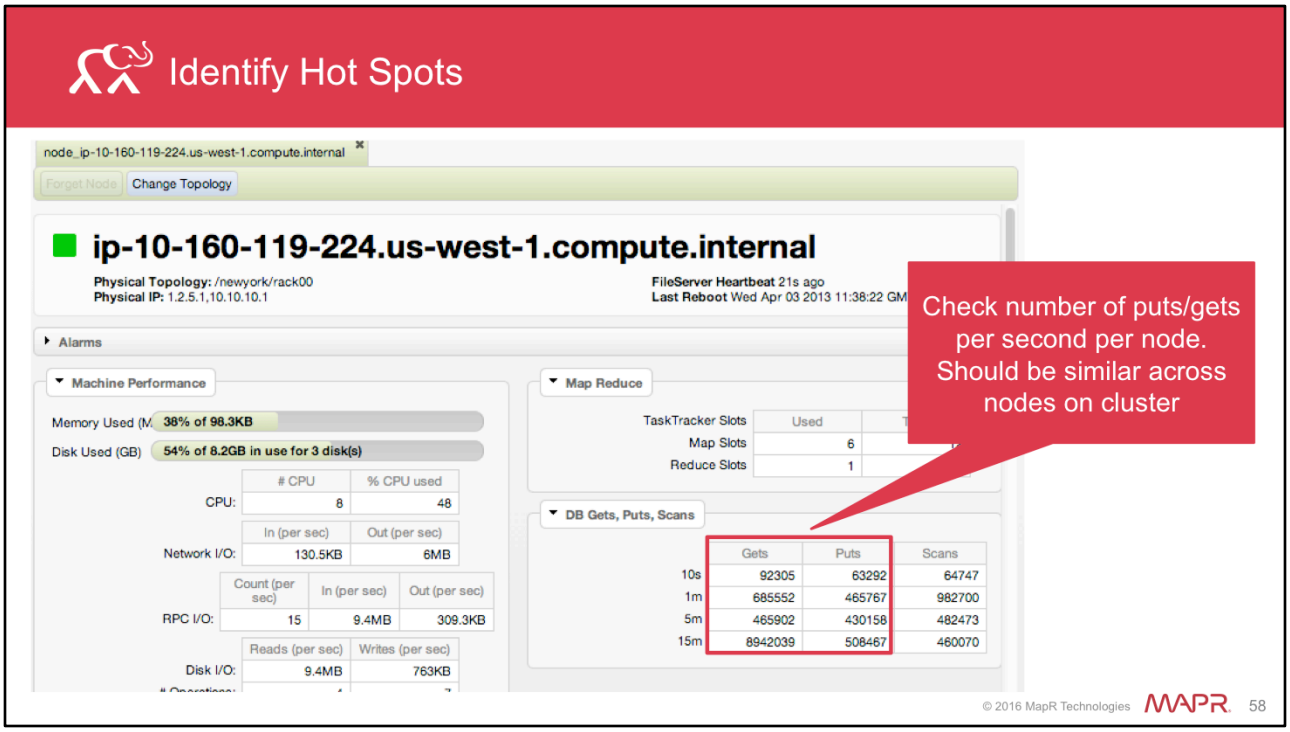


You should monitor that your data is spread out over regions for good concurrency.

If some regions have too much data, you could split those regions to spread out your data for better concurrency.

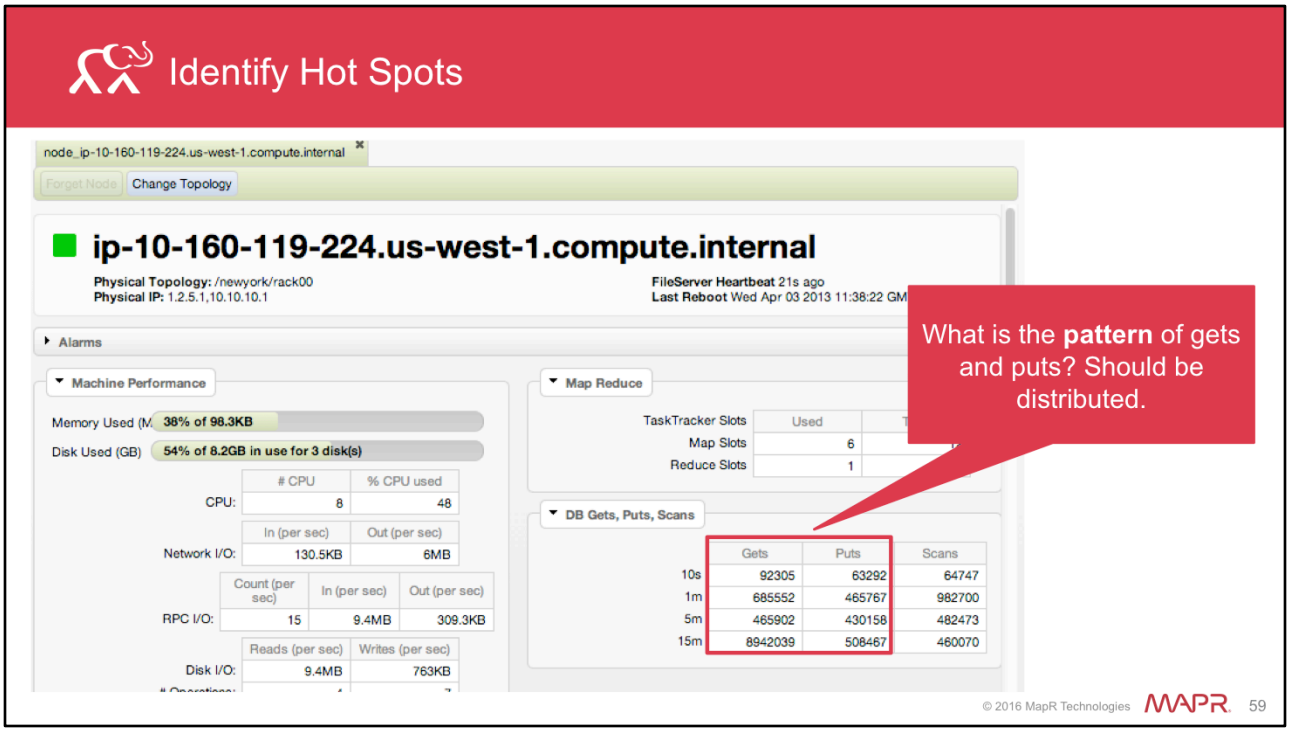


When a subset of MapR-DB servers get disproportional numbers of get/put requests compared with the cluster size, it is called “Hot Spotting”. Refer to DEV 325 for more information on hot spotting.

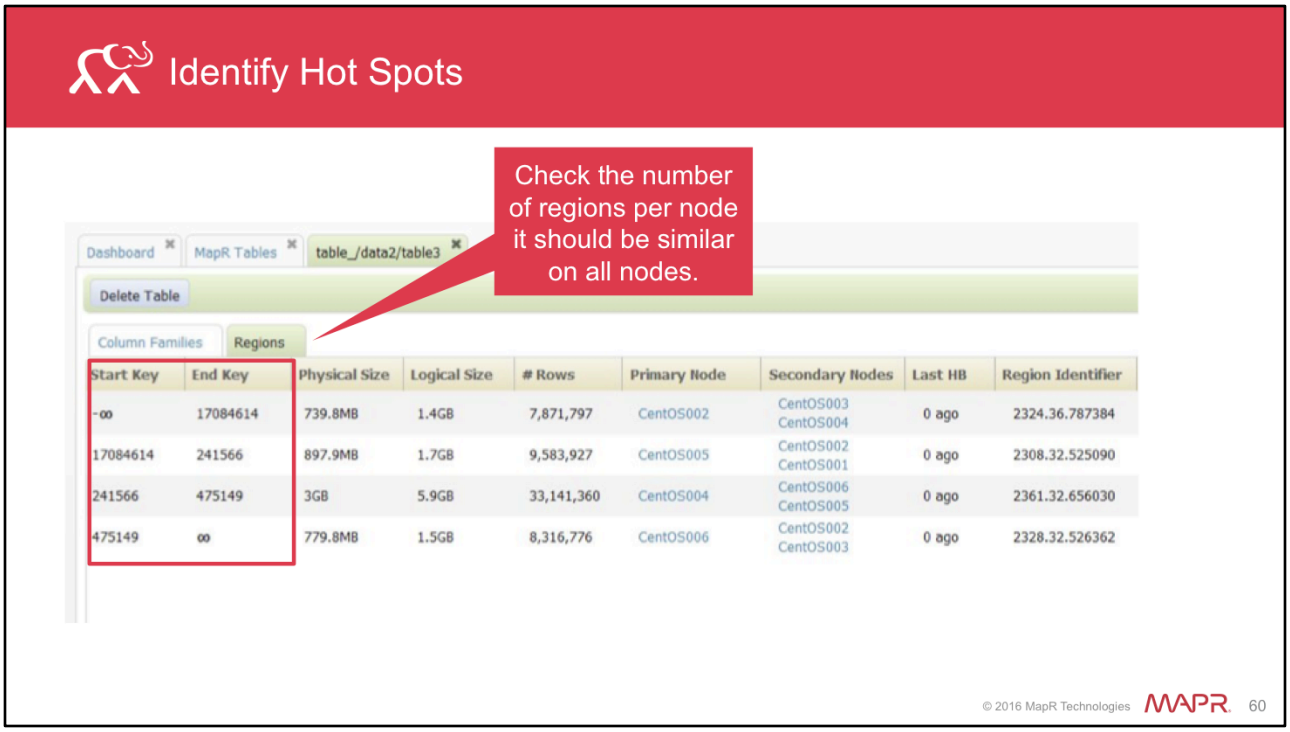


You can use the MCS Machine Performance pane to display the number of gets, puts, and scan operations performed during various time intervals, to check for hot zones.

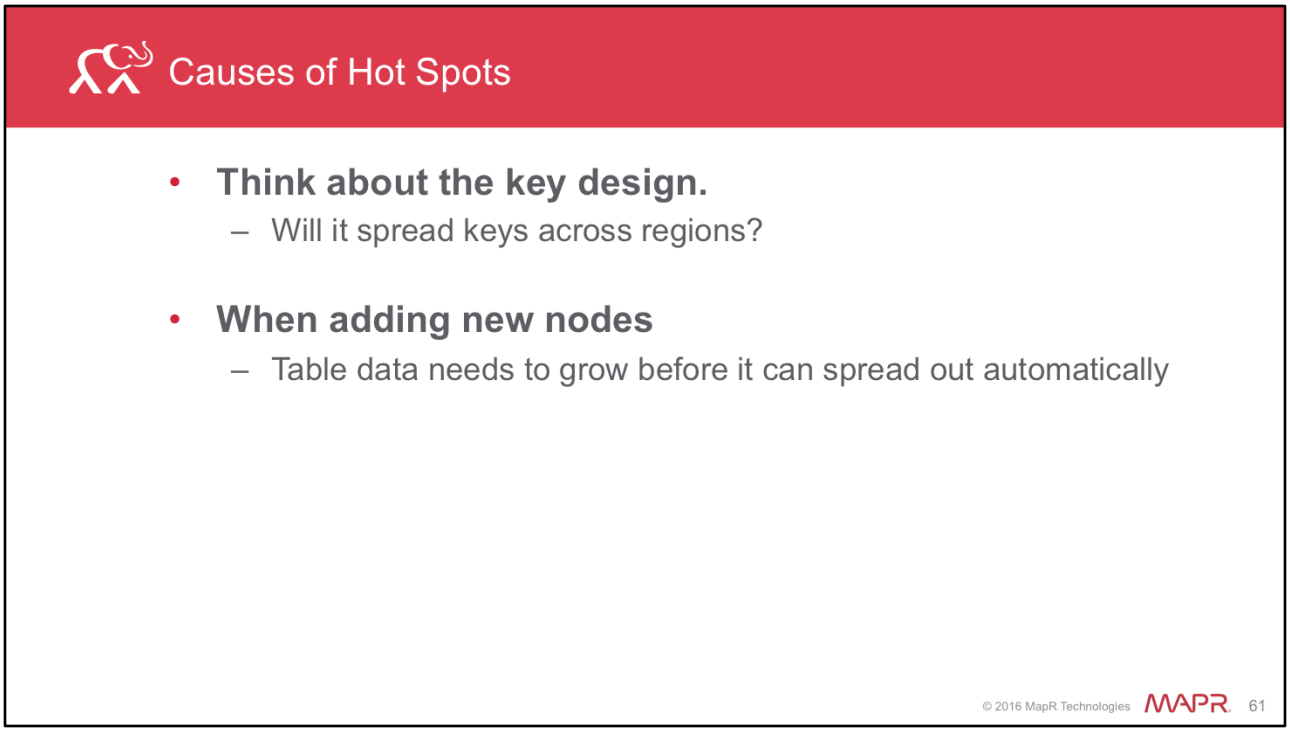
Check the number of puts/gets per second per node. They should be similar across nodes on the cluster.



Also, check the pattern of gets and puts. These should also be evenly distributed across nodes.

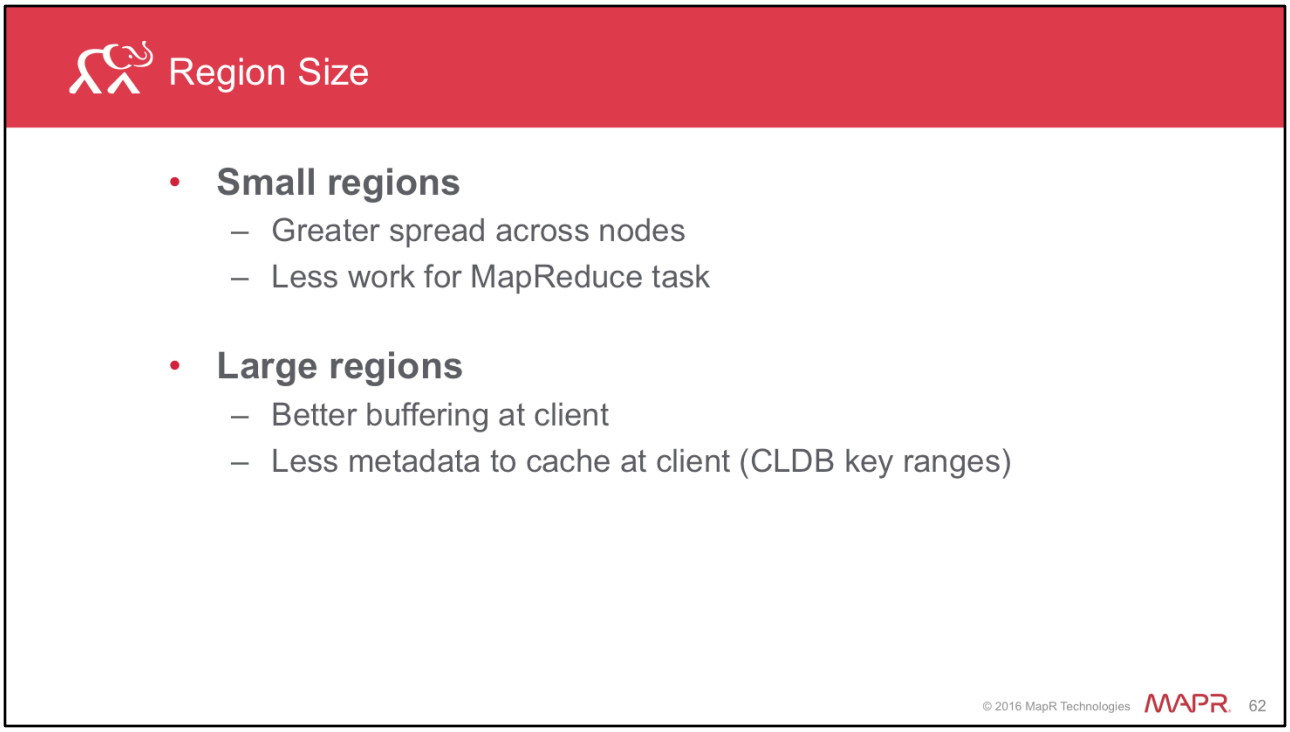


Check the number of regions per node, this should be similar on all nodes.



Think about the key design. Will it spread keys across regions? Review DEV 325, if you are having problems.

If you are adding new nodes then existing Table's data needs to grow before it can spread out automatically.



There are tradeoffs with region size:
Smaller regions will give you a better spread across nodes which can be better for parallel processing across nodes.
Larger regions will give you less key range location meta data to cache on the client side .

Manually Tune MapR-DB Region Size

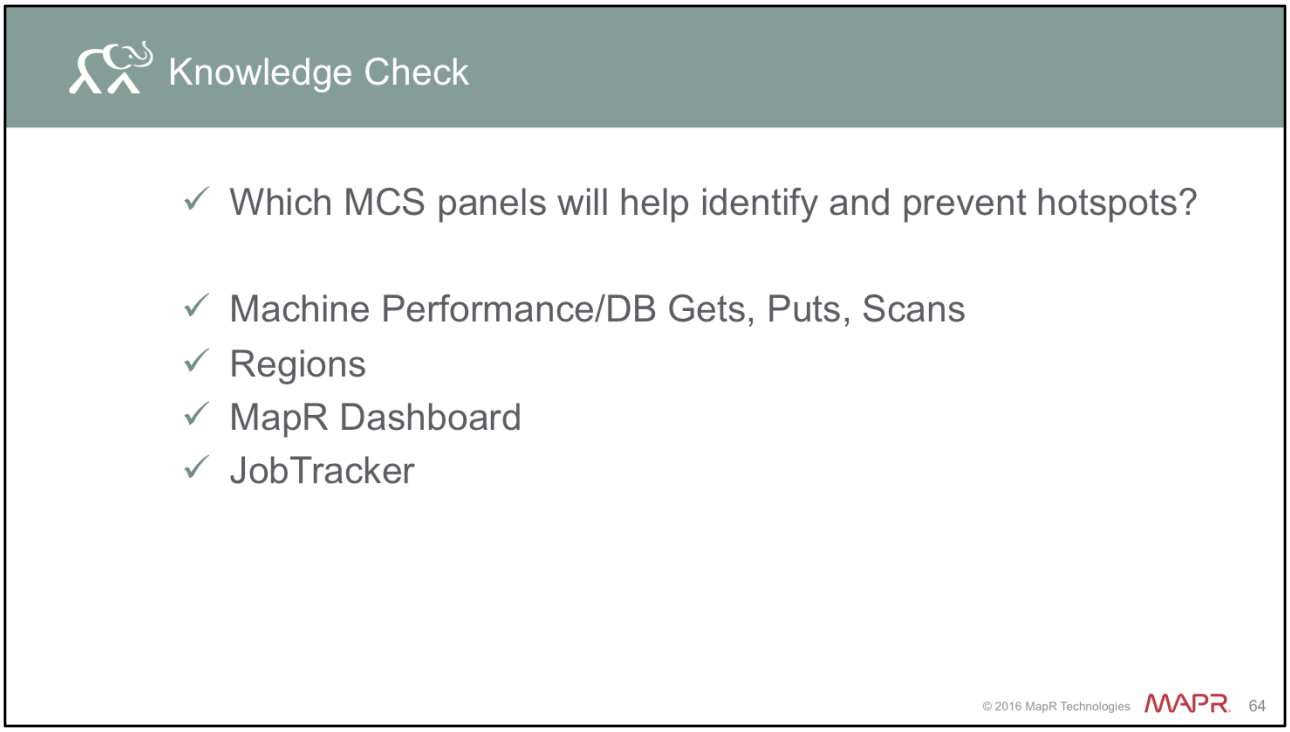
- **Enable/disable autosplit:**
  - `maprcli table edit -autosplit true/false -path tablePath`
- **Set default region size for a table:**
  - `maprcli table edit -regionSizeMB regionSize -path tablePath`
- **Manually split a region:**
  - `maprcli table region split -path tablePath -fid regionFid`
- **Manually merge regions:**
  - `maprcli table region merge -path tablePath -fid regionFid`

http://doc.mapr.com/display/MapR/table+region

 63

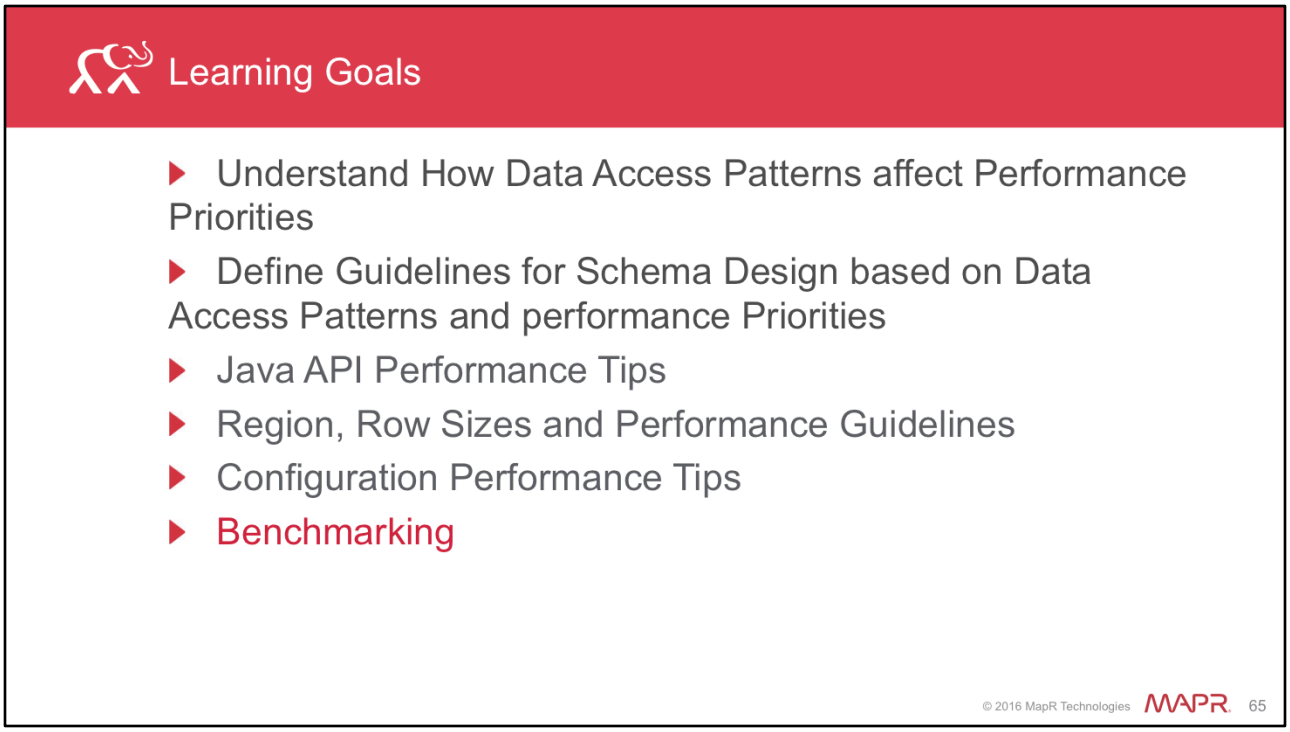
These are the commands which you can use to manually tune MapR-DB region size.

See the mapr documentation for more details: http://doc.mapr.com/display/MapR/table+region.

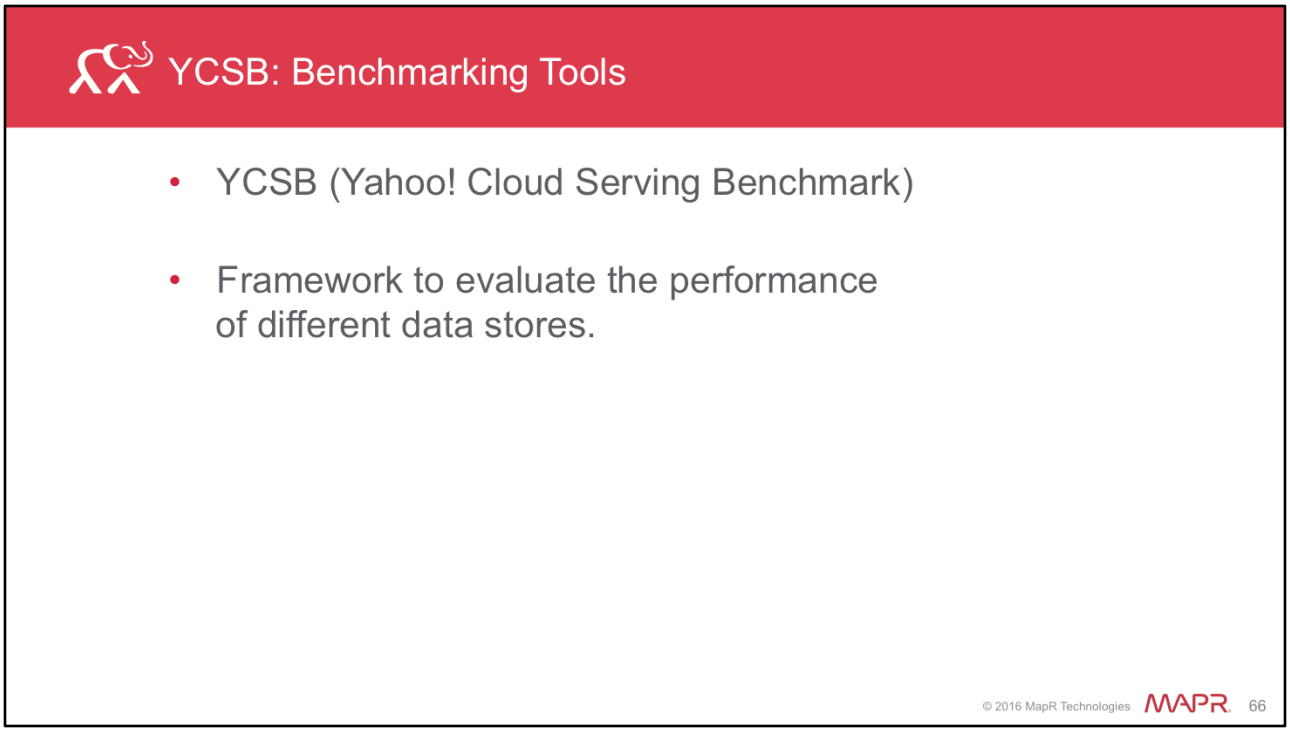


Which MCS panels will help identify and prevent hotspots?

Machine Performance/DB Gets, Puts, Scans – TRUE. You can use this panel to view the number and pattern of Gets, Puts and Scans, to make sure they are even spread out across the cluster.
Regions – TRUE. Make sure the number of regions per node is similar across the cluster.
MapR Dashboard – False. The MapR dashboard does not contain information about HBase or MapR-DB tables.
JobTracker – False. The JobTracker panel contains information about MapReduce jobs, but not the data stored in HBase or MapR-DB tables.



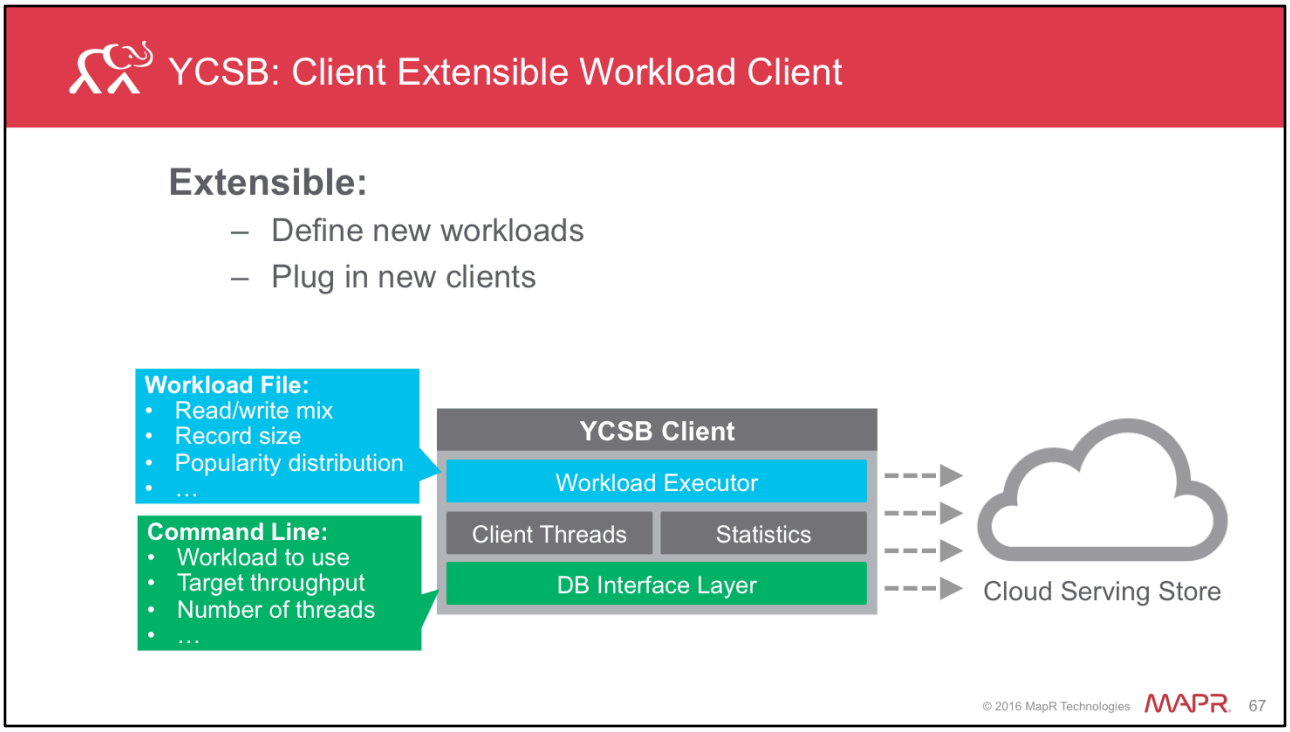
In this next section, we will look at how you can monitor performance with the MCS.



A popular tool for measuring HBase performance is the Yahoo! Cloud Serving Benchmark, or YCSB. We will use this tool in the lab exercises.

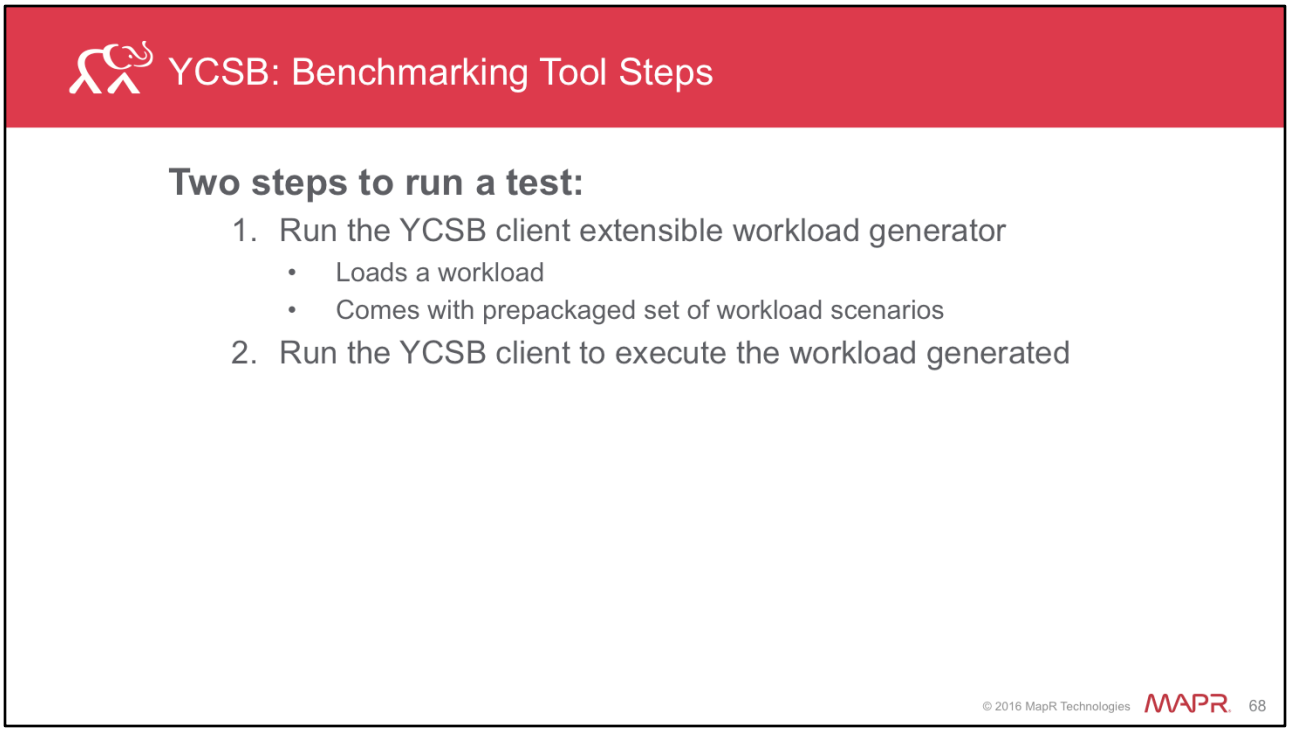
YCSB is a standard performance-testing tool that can be used to compare different databases.

--
https://github.com/brianfrankcooper/YCSB/



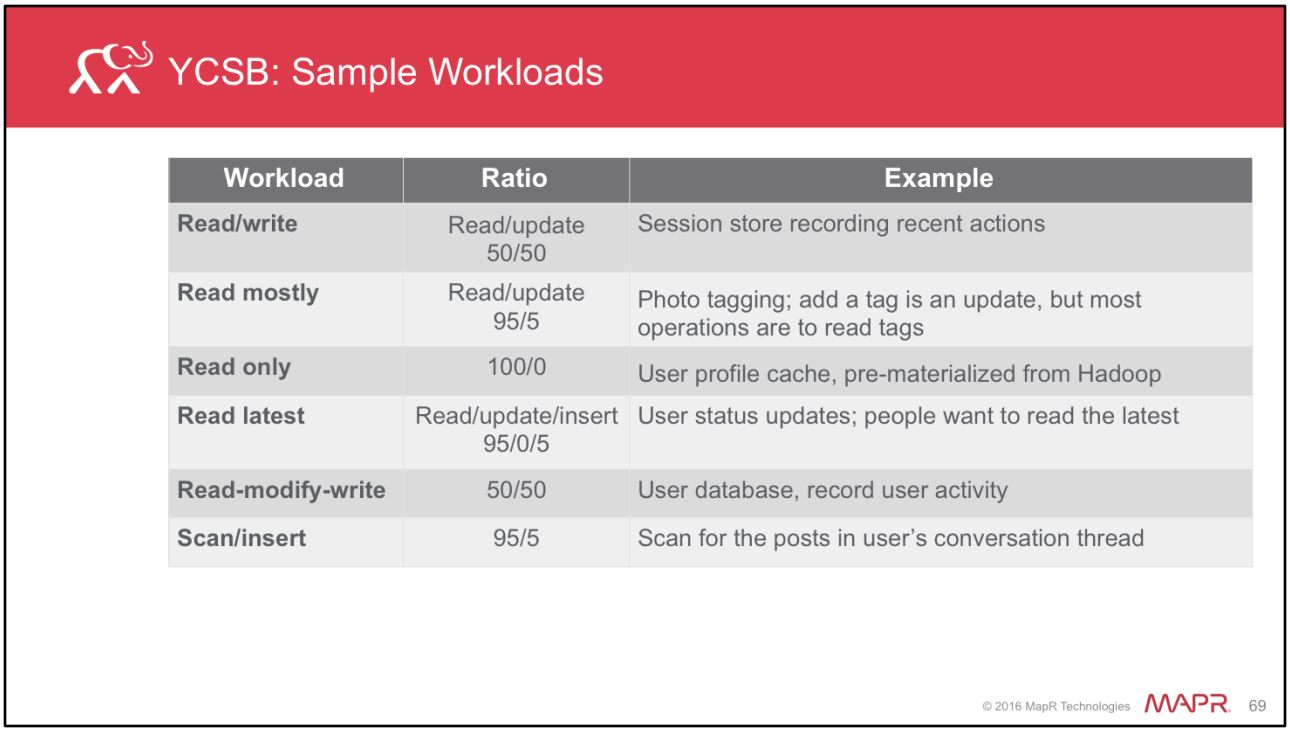
YCSB consists of the YCSB client, which is an extensible workload generator, and the core workloads, which are a set of workloads that comes prepackaged and can be generated by the YCSB client.

The Client is extensible so that you can define workloads.

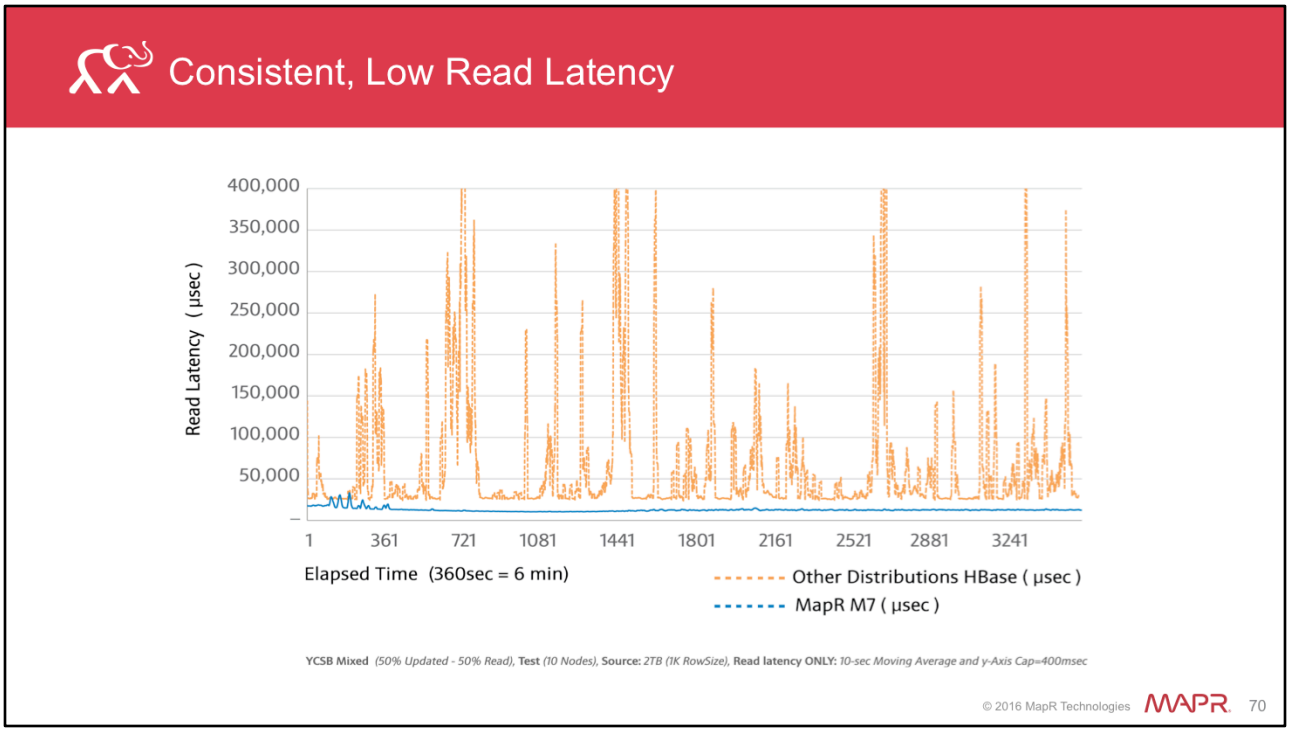


There are 2 steps to run a YCSB test:

- You first run the client to generate a workload
- then you run the client to execute the workload you just generated



This shows the different prepackaged workloads that you can run with YCSB. You can test for workloads from read only to read/write 50/50.



This graph shows read latency over time, a smaller read latency means faster response time, so smaller is better. MapR-DB is shown in blue and Cloudera HBase in Orange. The spikes for Cloudera HBase are because of the impact of garbage collection and compactions on performance. MapR-DB is written in C, there is no garbage collection. MapR-DB also does not have to do compactions.

Avoid Latency Spikes in Production

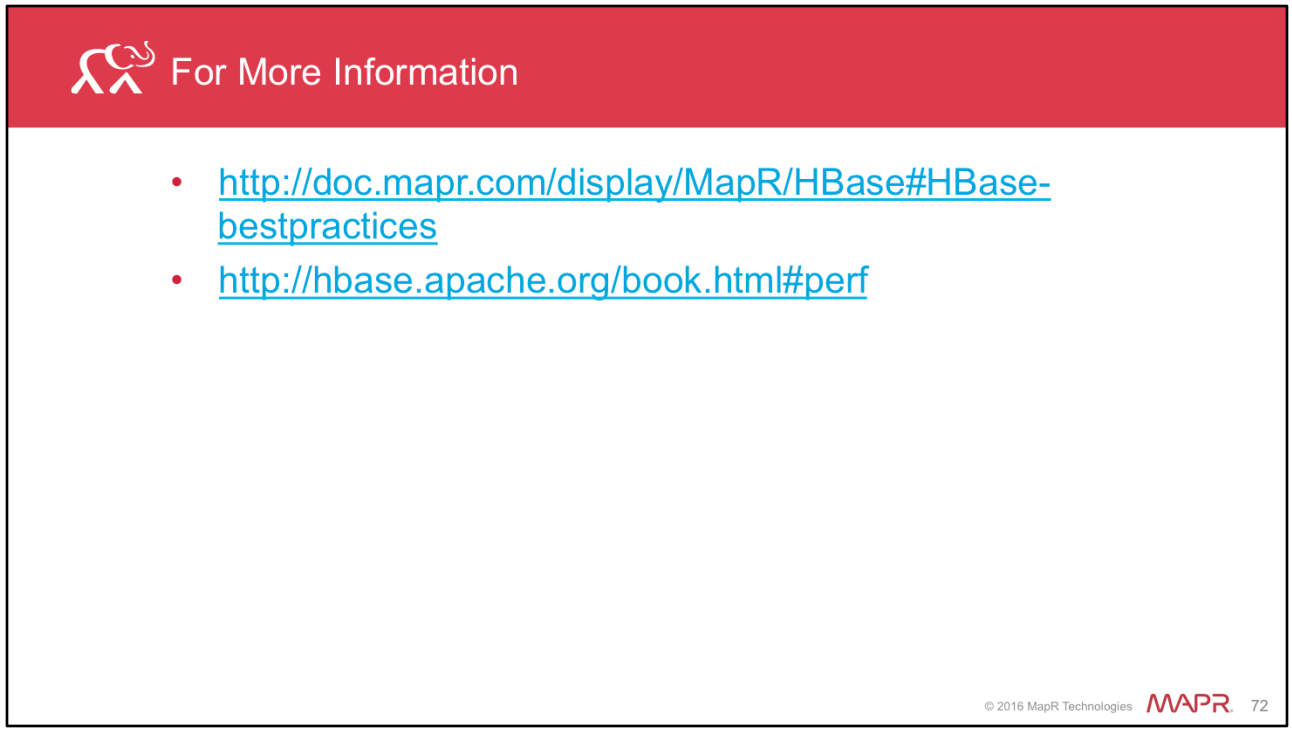
HBase suffers from large compaction and garbage collection delays

	Average read latency (ms)		
Benchmark	**MapR-DB 5.x**	**CDH 5.4.4**	**MapR-DB Improvement**
50% read, 50% update	12.5	28.4	2.3x
95% read, 5% update	15.6	44.0	2.8x
Reads	12.3	51.3	4.2x
Scans (50 rows)	32.3	174.9	5.4x

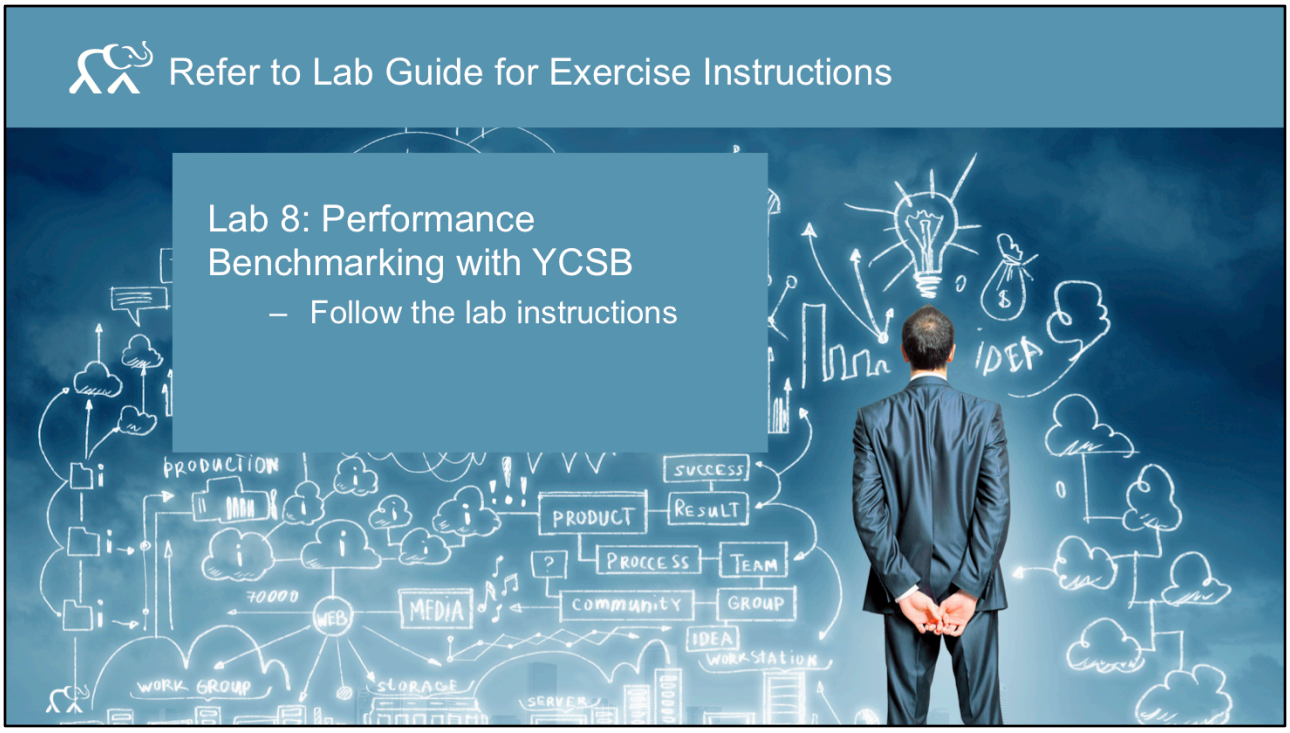
Test configuration: 10 Node Cluster
16x2 cores (2.6 GHz), 128G RAM, 1x10GE, jumbo frames, 11x1TB disk (7200 RPM, 1 SP)
1K record size, 2TB data size, SC reads enabled, CentOS Release 6.2 (Final)

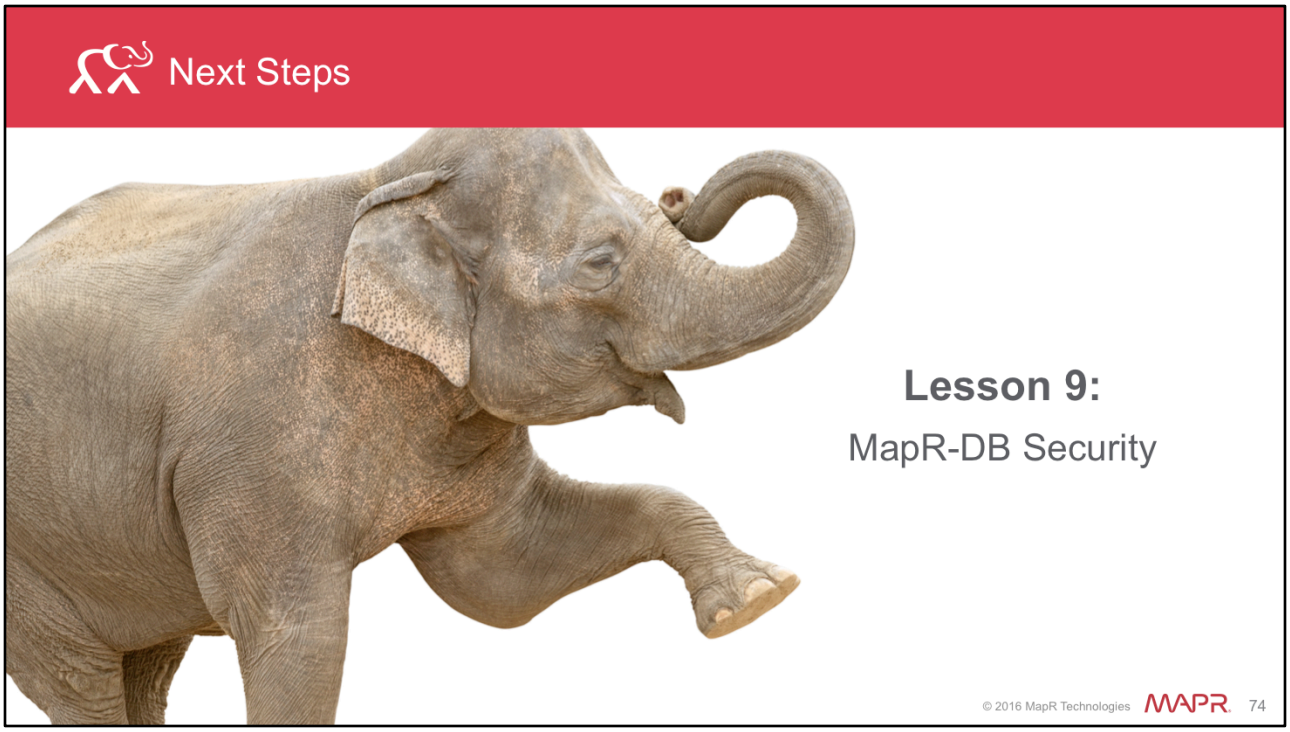
 71

This table shows a YCSB benchmark comparing MapR-DB to Cloudera HBase. The last column show the observed MapR-DB advantage in performance.

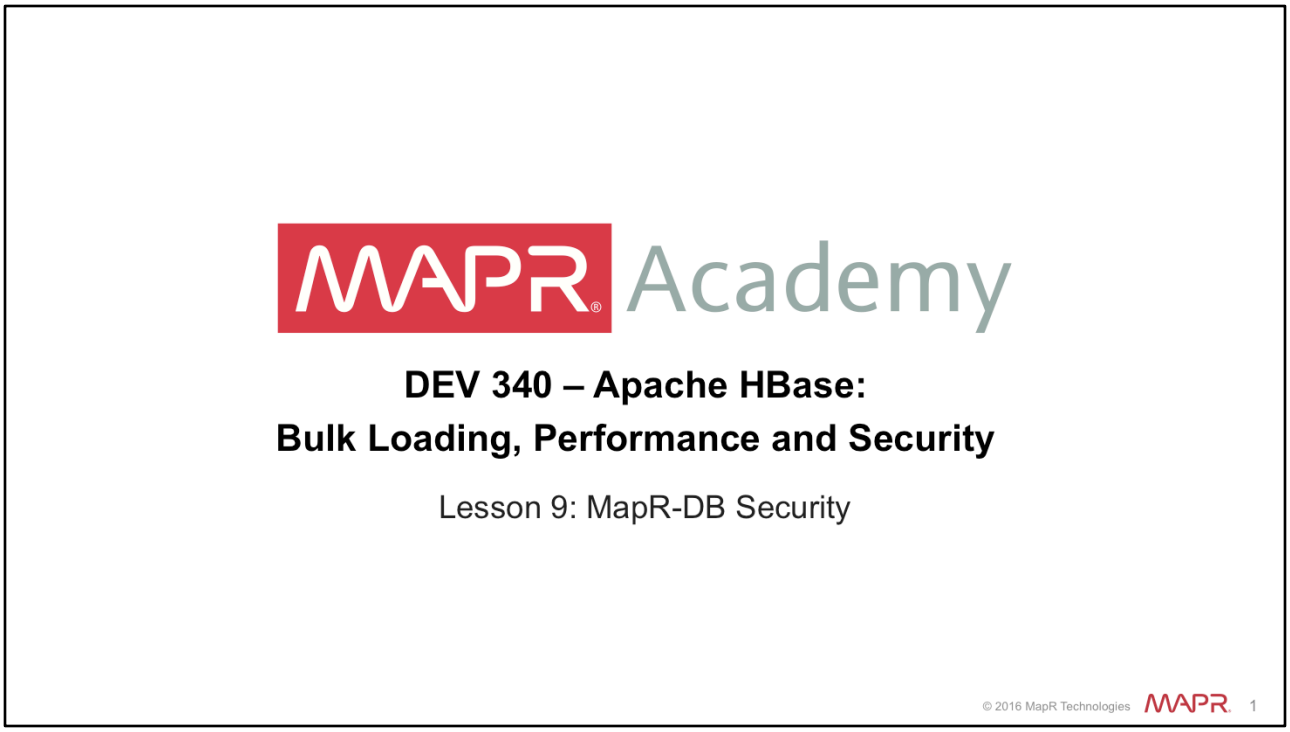


http://doc.mapr.com/display/MapR/HBase#HBase-bestpractices


Refer to Lab Guide for Exercise Instructions
Lab 8: Performance Benchmarking with YCSB
– Follow the lab instructions
PRODUCTION
PRODUCT
RESULT
PROCCESS
TEAM
COMMUNITY
GROUP
MEDIA
WEB
IDEA
WORK GROUP
STORAGE
SERVER
WORKSTATION



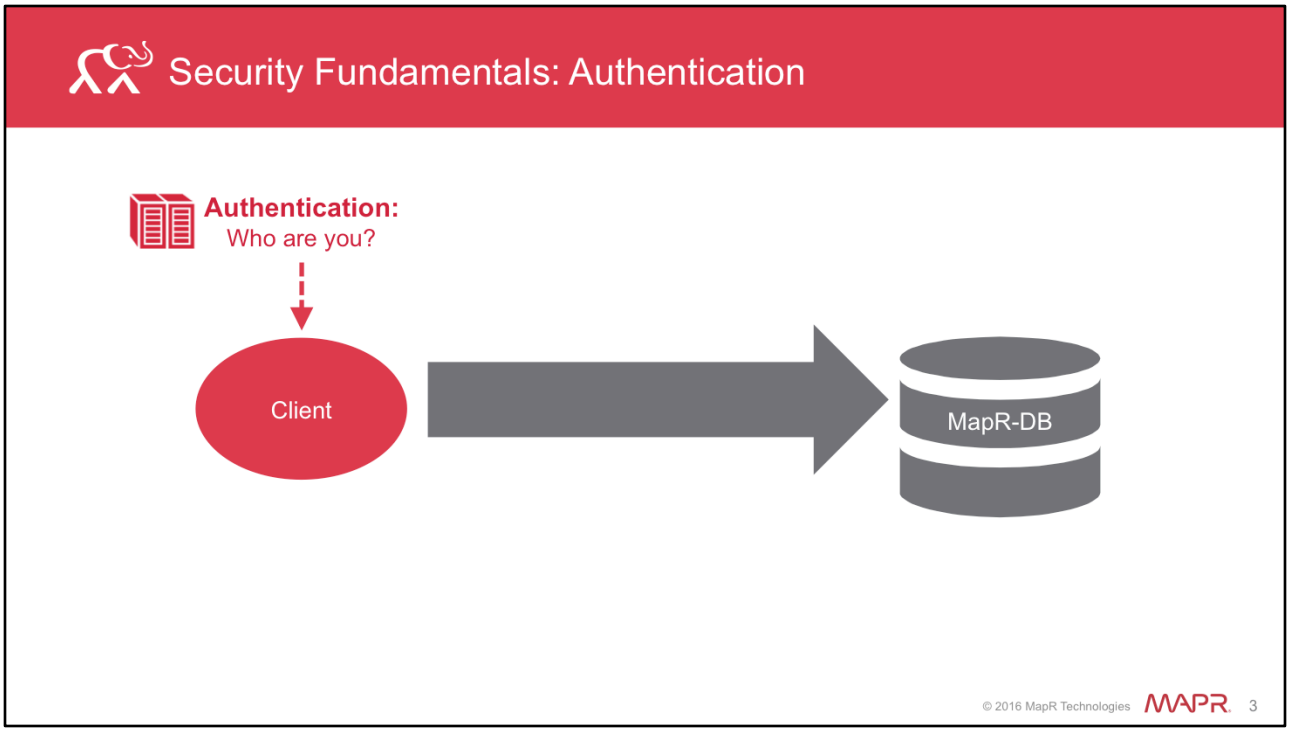
Congratulations, you have finished DEV 340 lesson 8. Continue to lesson 9 to learn about MapR-DB security.



Welcome to DEV 340, Lesson 9: MapR-DB Security.

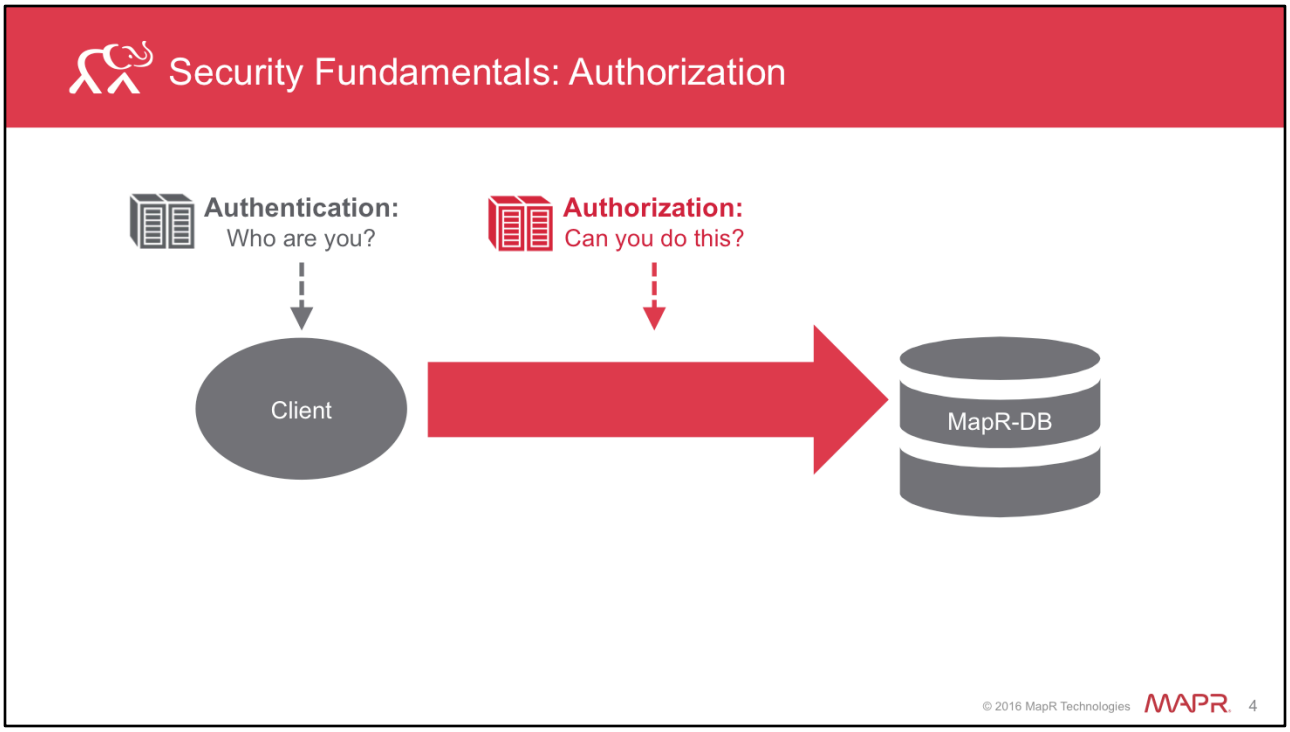
When you have finished with this lesson, you will be able to Define the fundamental concepts of computer systems security, And properly secure data held in your MapR-DB

To start, let's review the overall concepts of computer systems security.

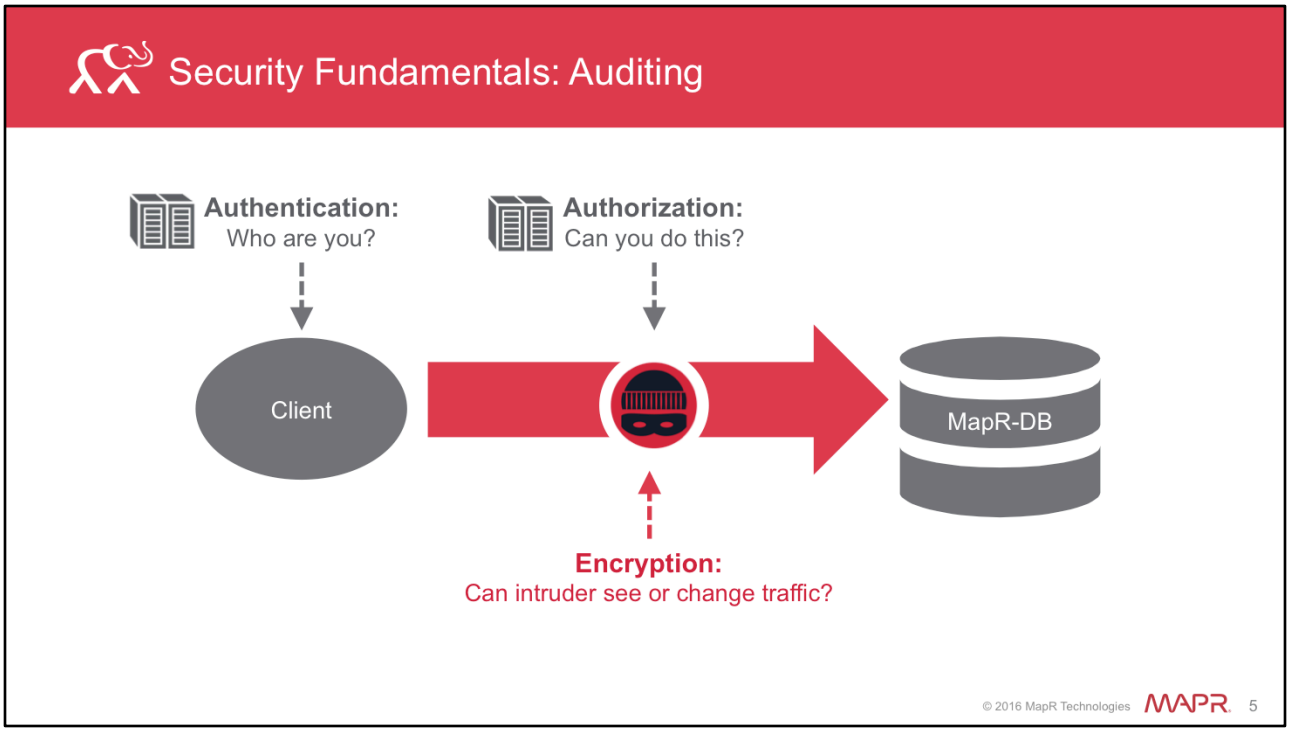


A secure environment is predicated on the following capabilities:

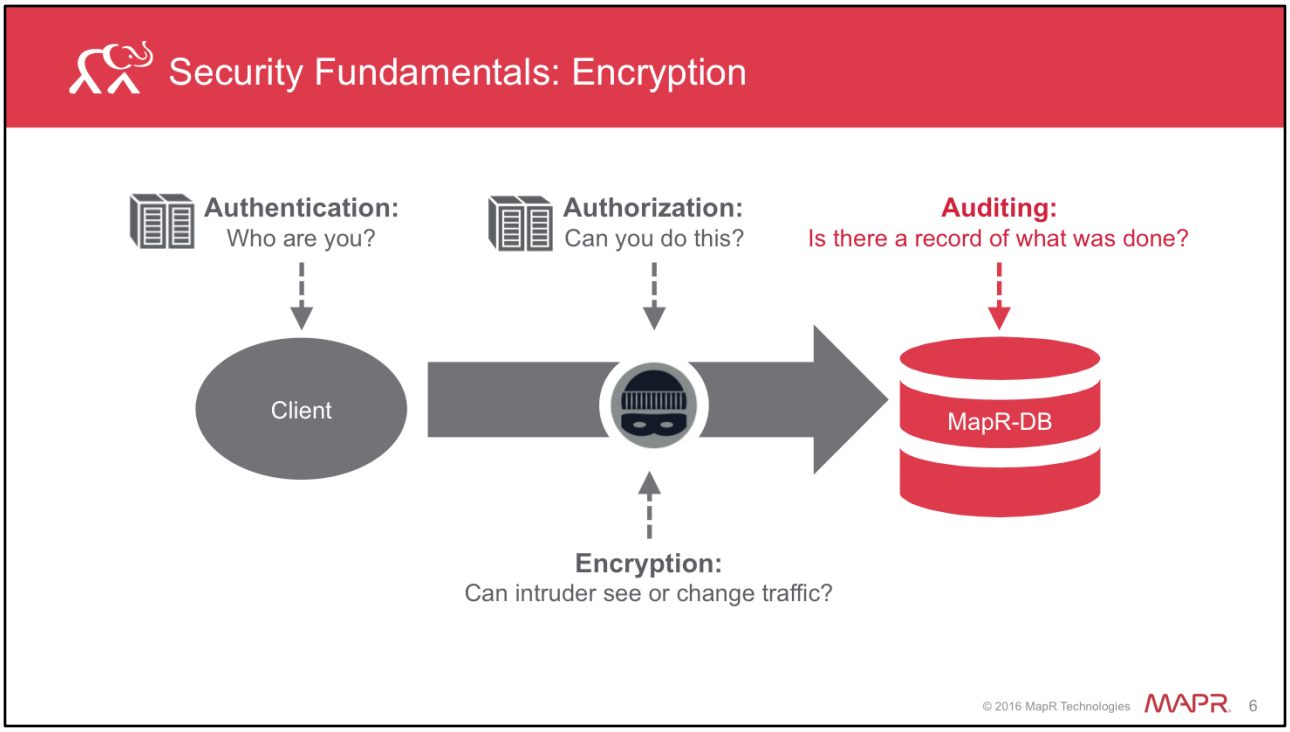
Authentication, is restricting access to a specified set of users. Robust authentication prevents third parties from representing themselves as legitimate users.



Authorization, is restricting an authenticated user's capabilities on the system. Flexible authorization enables a system to grant a set of capabilities for a user to perform desired tasks, but prevents the use of any capabilities outside of that scope.

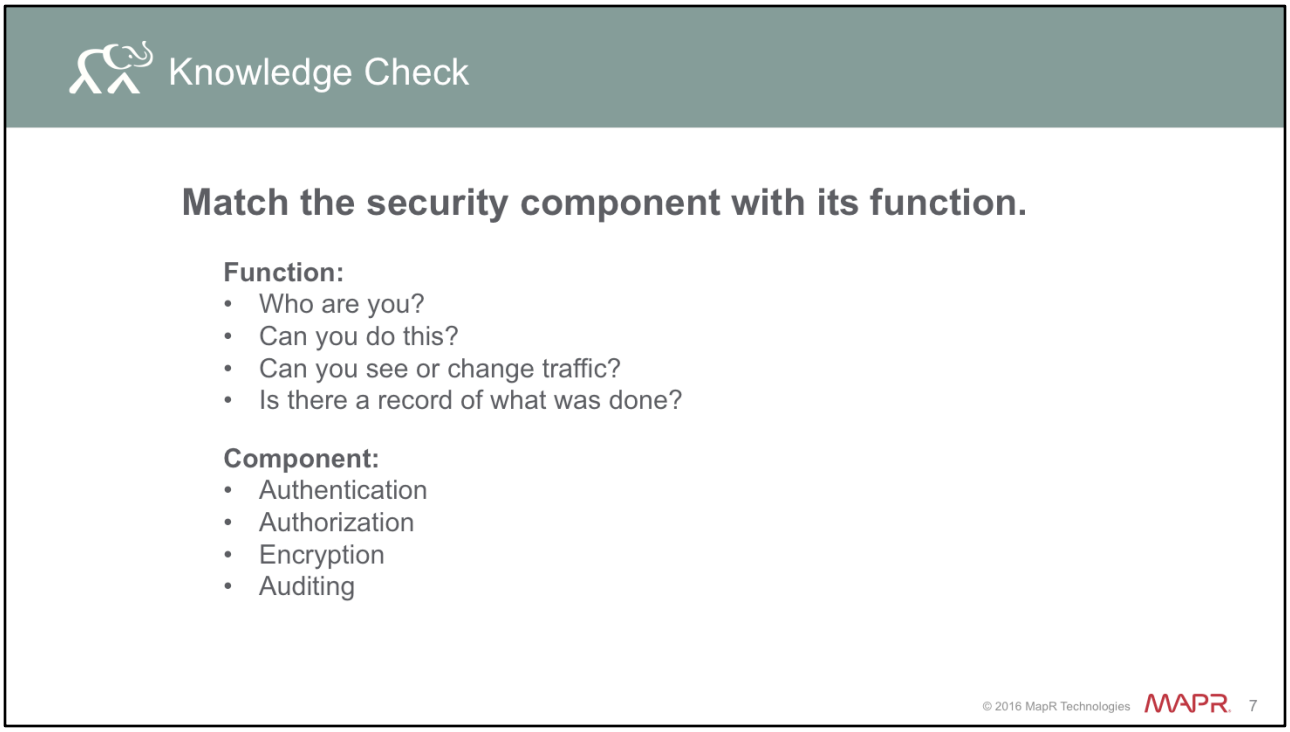


Encryption, is restricting an external party's ability to read data. Data transmission between nodes in a secure MapR cluster is encrypted. This prevents an attacker with access to that communication from viewing or changing the contents of the transmission.



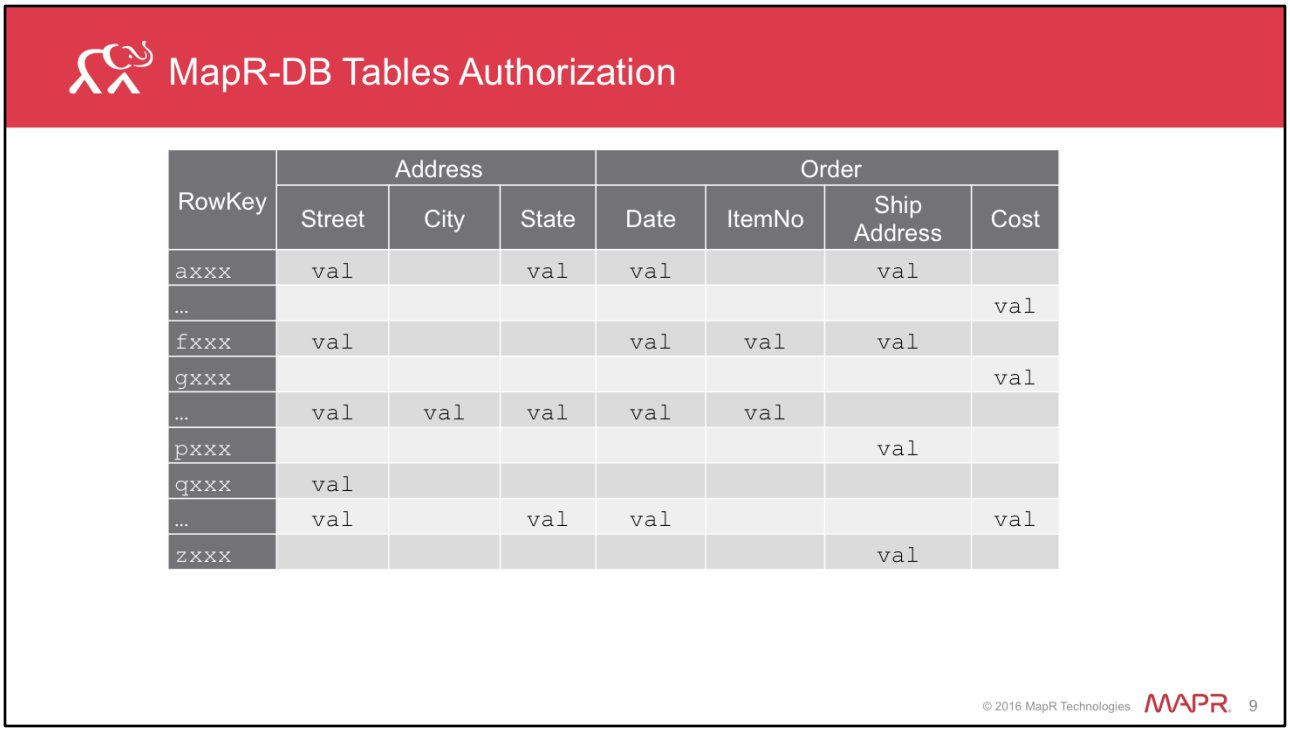
Meanwhile, *Auditing* keeps a record of all actions performed in your system.

Commonly, systems will use one or more of these components to create a secure environment.



Authentication::Who are you?
Authorization::Can you do this?
Encryption::Can you see or change traffic?
Auditing::Is there a record of what was done?

Now that we know how computer systems security works, let's look at authorization into a MapR-DB.



Permissions for MapR-DB tables, column families, and columns are defined by Access Control Expressions, or ACEs.

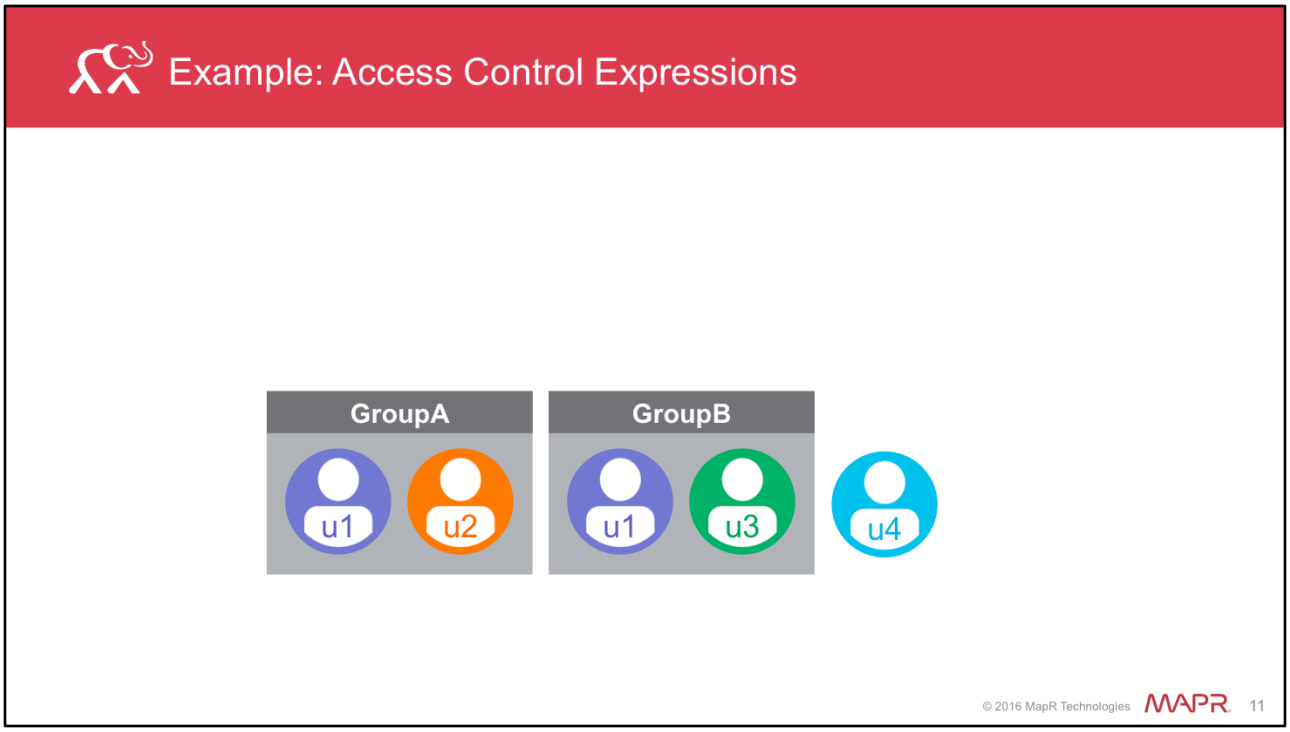
Access Control Expressions

Tokens	Description
u	Username or user ID, as they appear in /etc/passwd
g	Group name or group ID, as they appear in /etc/group
r	Name of a specific role
p	Public. Specifies that this operation is available to the public without restriction.
Operators	**Description**
!	Negation operator
&	AND operation
|	OR operation
()	Delimiters for subexpressions
“”	The empty string indicates that no user has the specified permission

 10

An ACE is a boolean expression of roles, users, groups and the boolean operators for and, or and not.

When a user, group, or role requests to read data from, or write data to a column, MapR-DB checks whether that user, group, or role has read or write permission for the column family AND read or write permission for the column.



For example, here we have 4 users and 2 groups

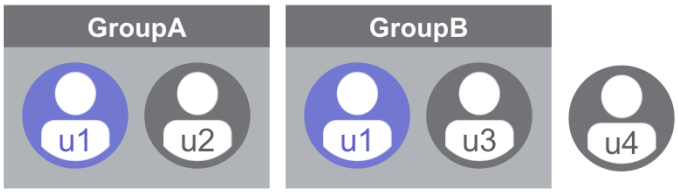
- GroupB includes users u1 and u3
- And GroupA contains users u2 and u1

User u4 in no group.

Example: Access Control Expressions

ACE Example:

```
g:groupB & g:groupA
```



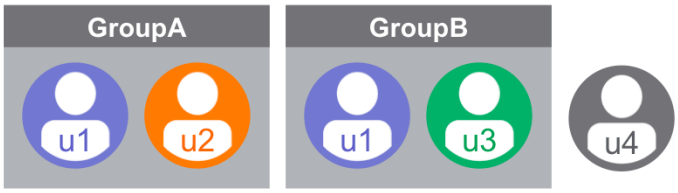
 12

An ACE of GroupB and GroupA would be only user u1.

Example: Access Control Expressions

ACE Example:

```
g:groupB | g:groupA
```

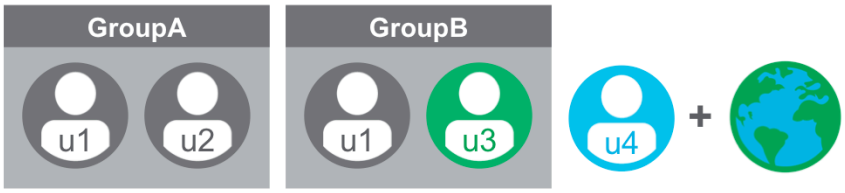


While the ACE of GroupB or GroupA would be users u1, u3, and u2.

Example: Access Control Expressions

ACE Example:

`!g:groupA` WARNING ! WHY ?



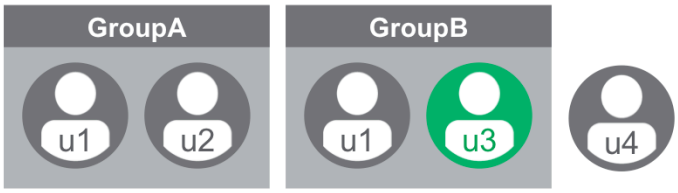
A warning about the NOT operator: NOT implies a very large universe. For example “!groupA” is everyone who is not in GroupA, which is a lot of people.

Example: Access Control Expressions

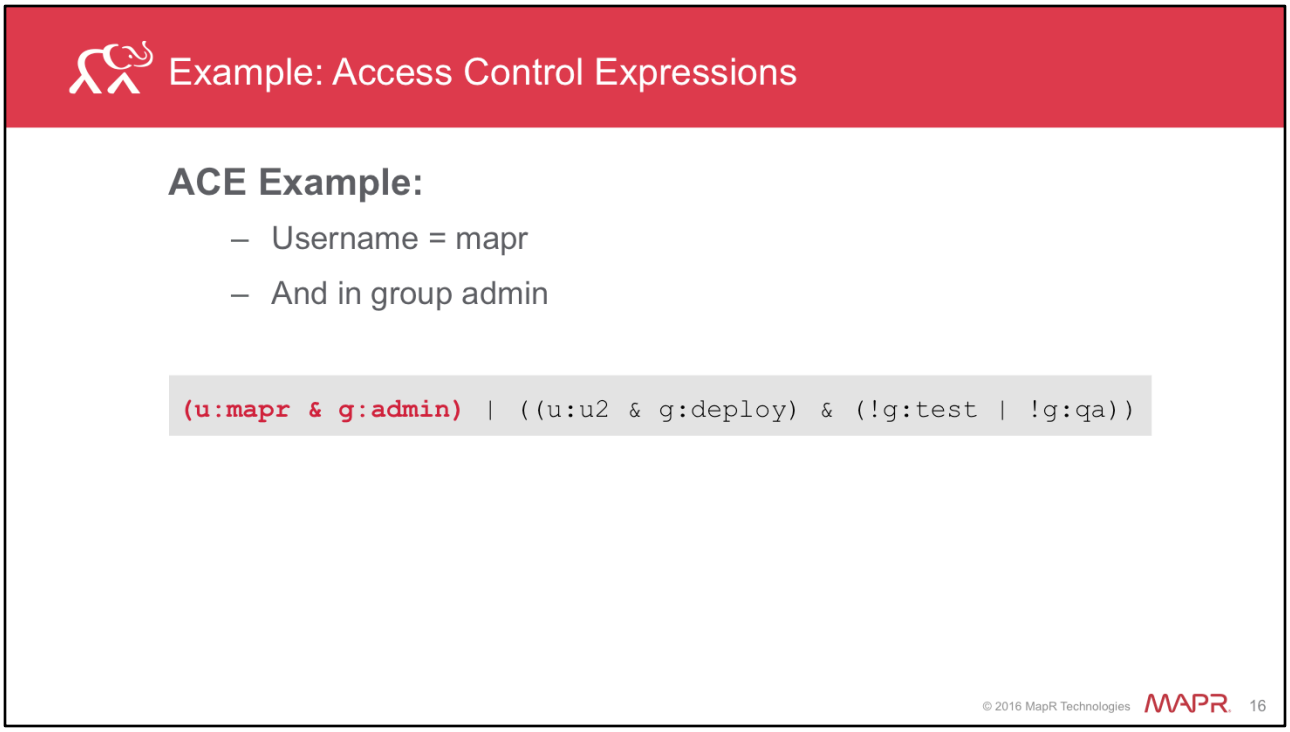
ACE Example:

`g:groupB & !g:groupA`

- Subset of GroupB

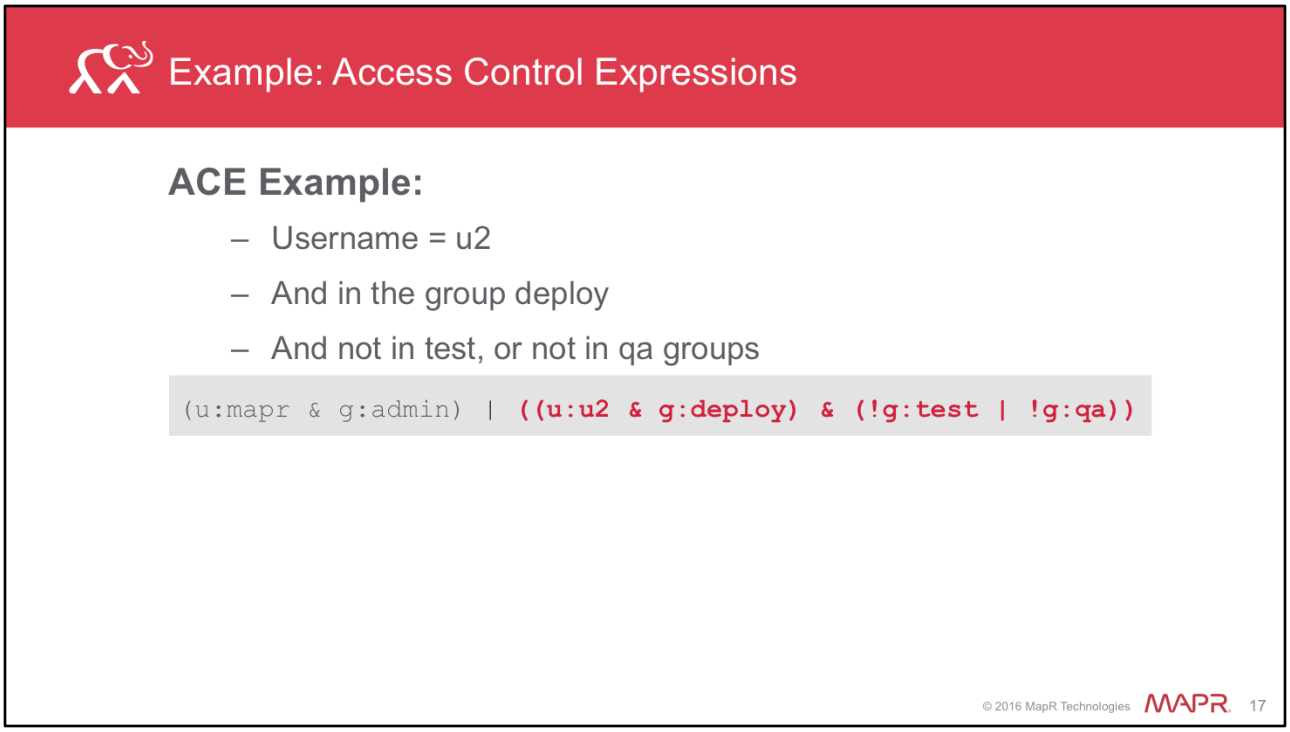


NOT is best used to limit some other constraint. For example “groupB&!groupA” which would be user just u3 in this example.



ACEs can be simple, or quite complex. In this example, we are giving access to user who have:

The username mapr and they are in the admin group.



OR

Their usersname is u2 and they are in the deploy group, and are not in the test group, or not in the qa group.

Role Based Access Control

A role is a label that defines a common task or set of behaviors related to permissions for an application

Admin

 18

A role is a name or label that defines a common task or set of behaviors related to permissions for an application, for example admin or staff.

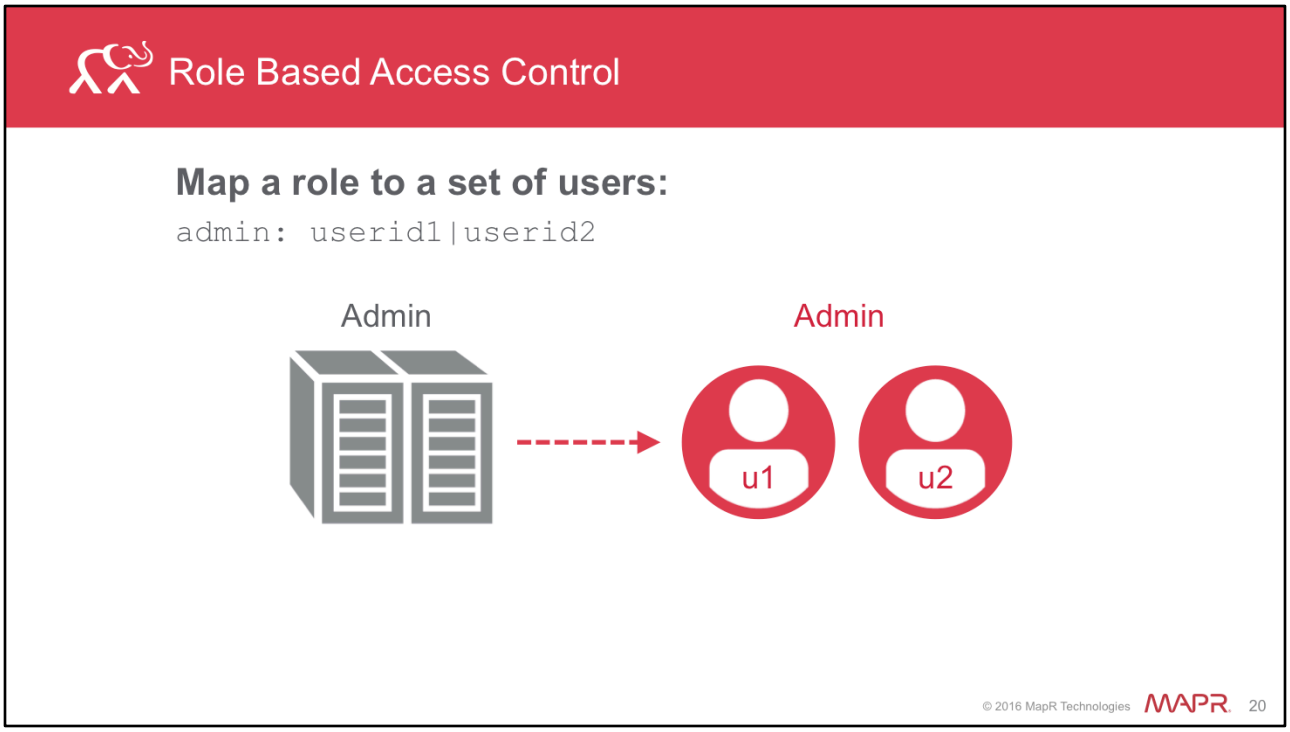
Role Based Access Control

An ACE can give permissions to that role:

```
Read Data u:mapr | r:admin
```

 19

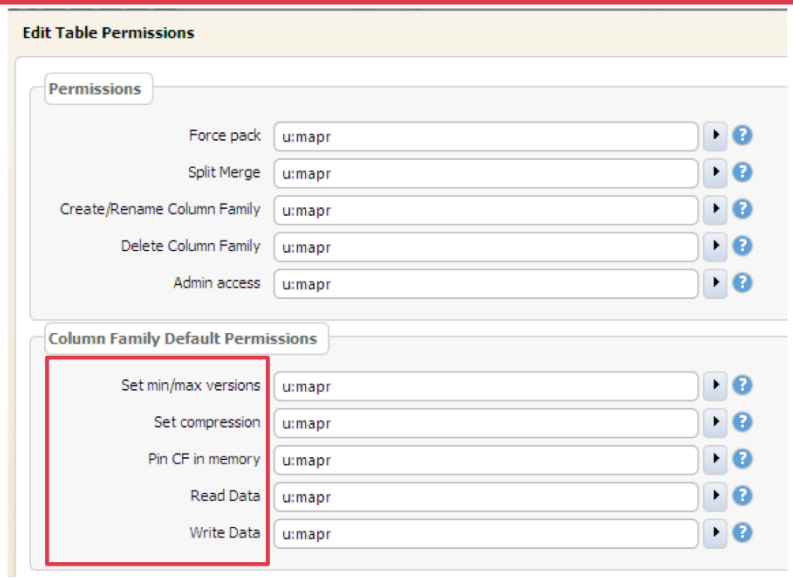
You can define a logical role in an ACE, such as Admin, and give permissions for that role.



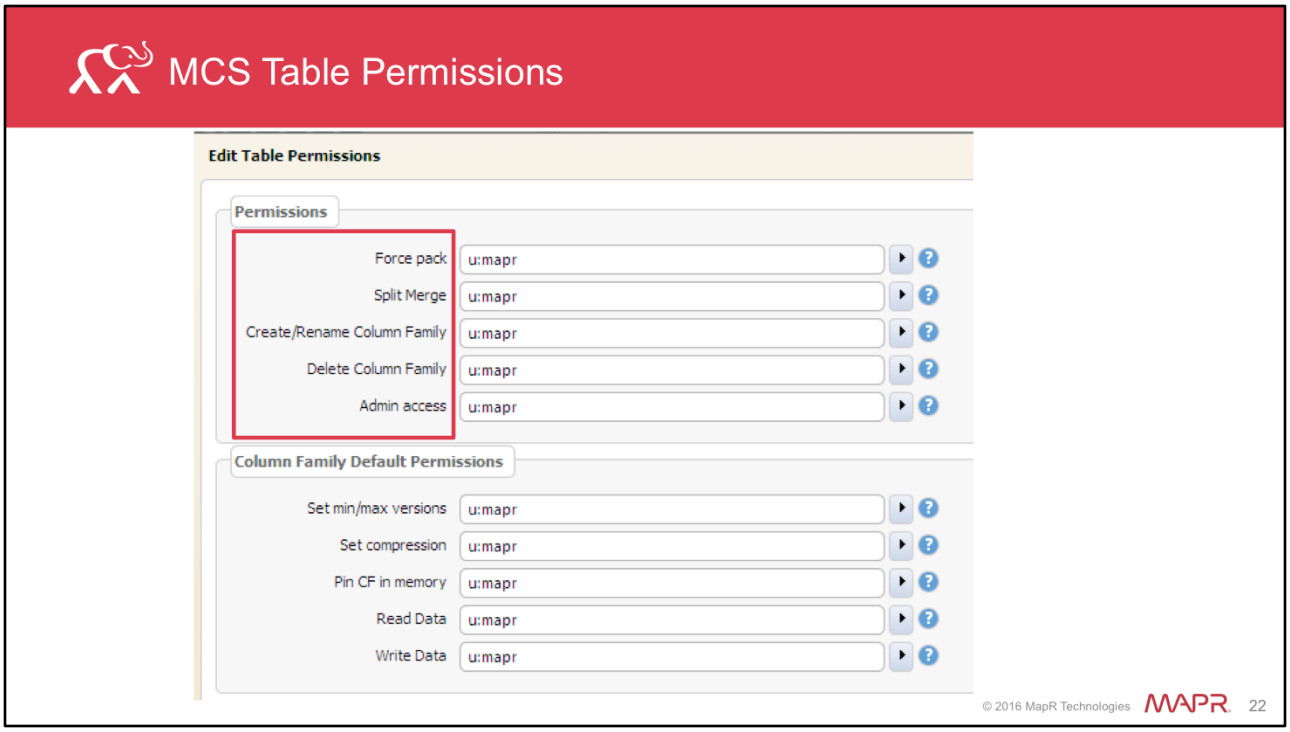
Then you can map this role to a set of user's userids, giving those users the permissions for that role, in this case Admin role permissions.

Roles are not Unix groups, they are internal to the MapR system. They enable you to use functionality similar to Unix groups for your users, without requiring you to alter your system's existing group hierarchy.

MCS Table Permissions



Access Control Expressions can be defined on MapR-DB table data from the table to the column level…

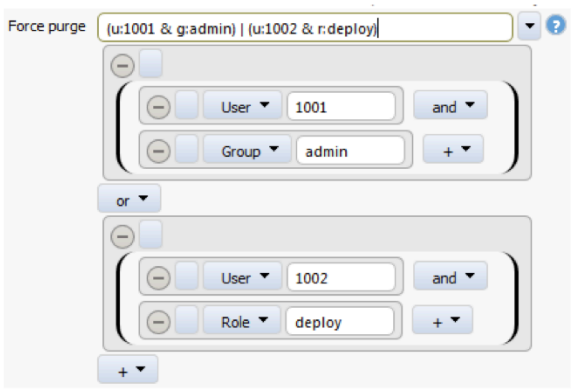


and on MapR-DB table operations such as add/delete column family, split/merge and pack operations.

This screen shot shows the MCS Edit Table permissions screen, which allows you to set permissions when you create or edit tables.

MCS ACE Expression Builder

Use the + button to add a condition to the expression. Note that you cannot mix AND and OR without using sub-expressions.



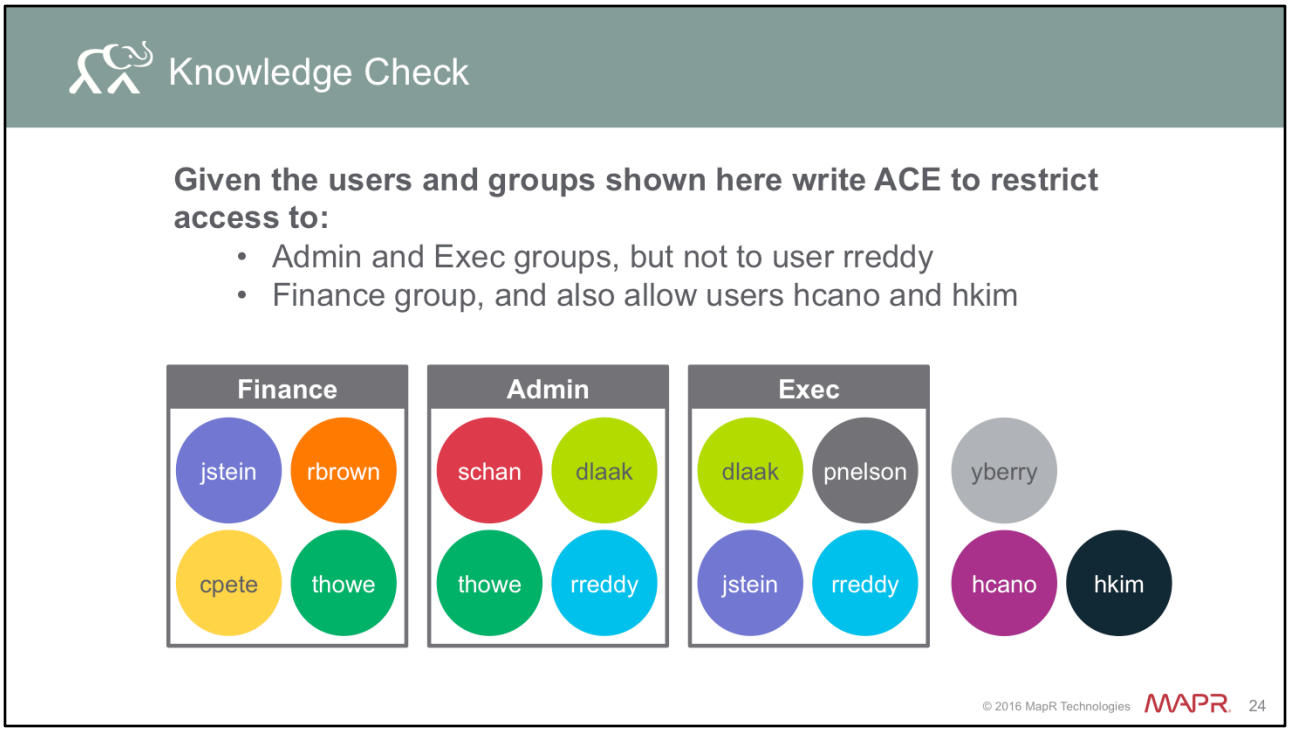
 23

The MCS GUI provides an expression builder that validates the correctness of the settings in real-time.

For more information refer to the mapr documentation.

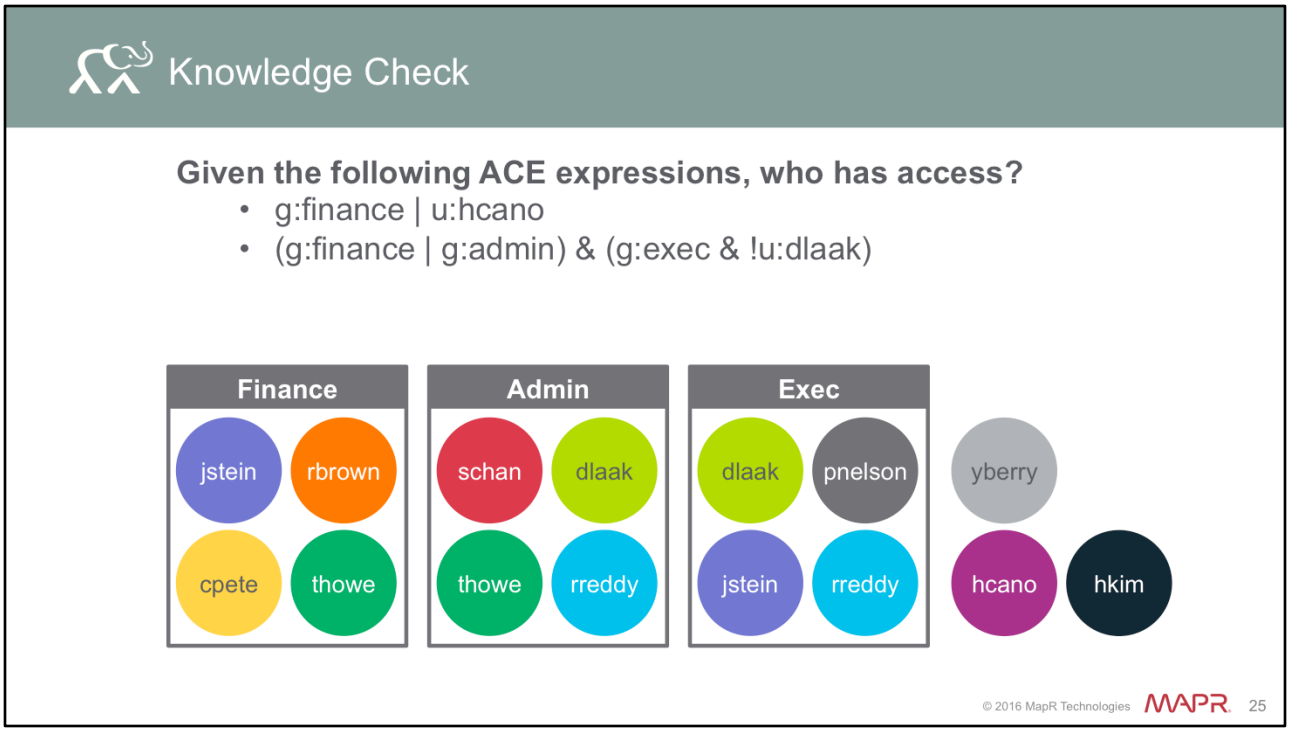
---

http://doc.mapr.com/display/MapR/Enabling+Table+Authorizations+with+Access+Control+Expressions



(g:admin | g:finance) & !u:rreddy
(g:finance | u:hcano | u:hkim)

If the user is in the group finance, or is the user hcano. Therefore, jstein, rbrown, cpete, thowe and hcano all have access.
If the user is in the group finance or has the role admin, and if they are in the group exec, except for user dlaak. Therefore, rreddy and jstein have access.



(g:admin & !u:rreddy) & g:exec
g:finance & u:hcano & u:hkim

If the user is in the group finance, or is the user hcano. Therefore, jstein, rbrown, cpete, thowe and hcano all have access.
If the user is in the group finance or has the group admin, and if they are in the group exec, except for user dlaak. Therefore, rreddy and jstein have access.

MapR-DB Tables Authorization

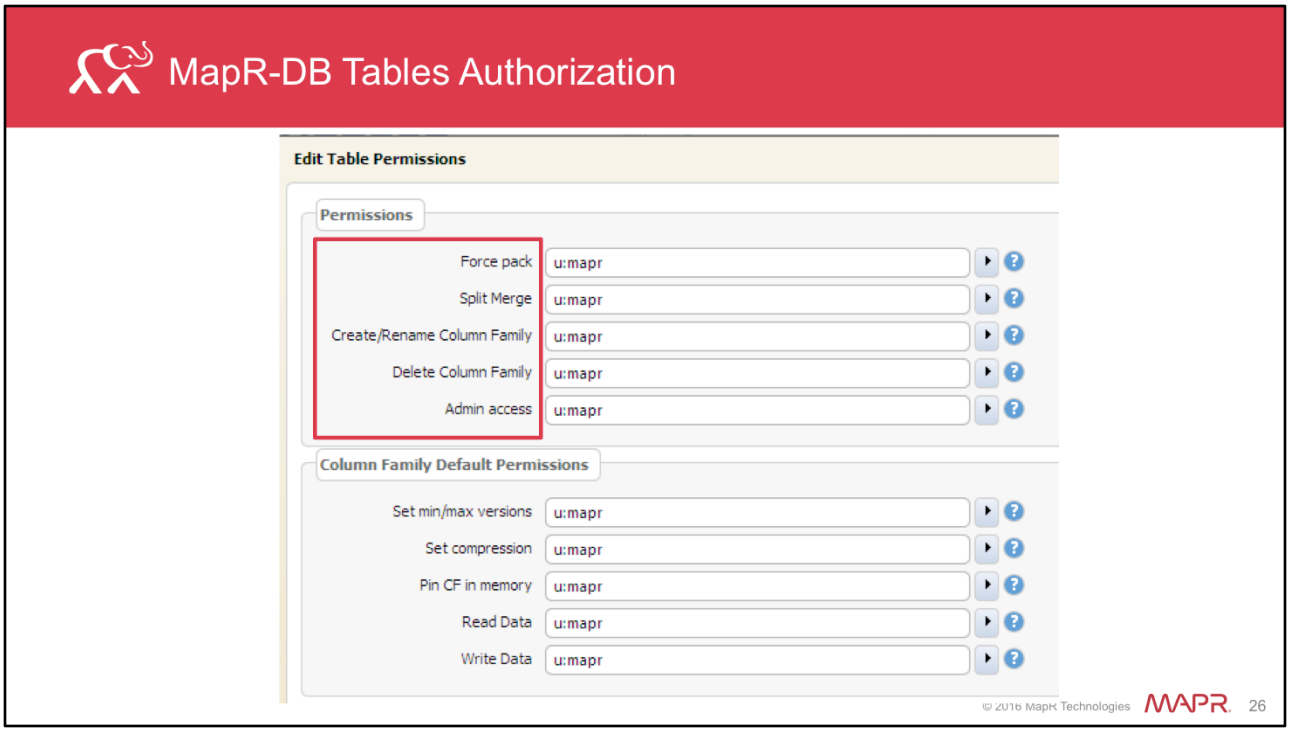
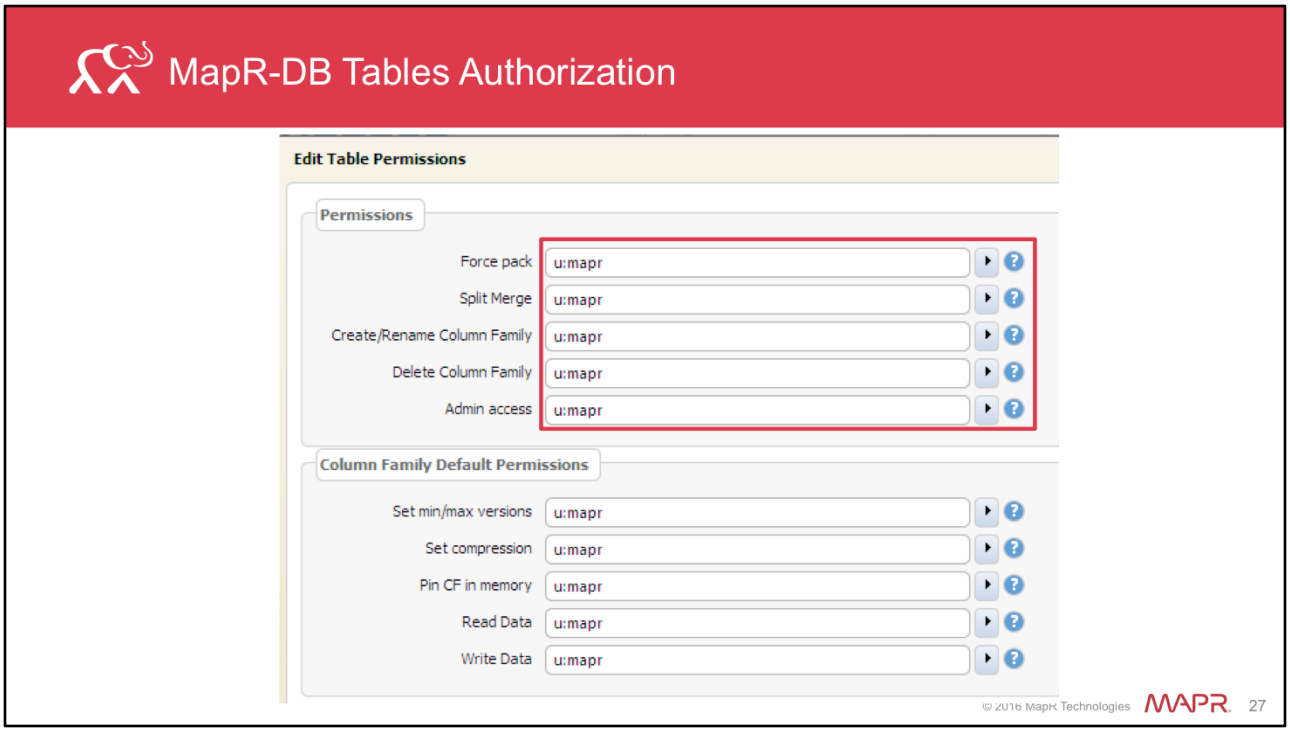
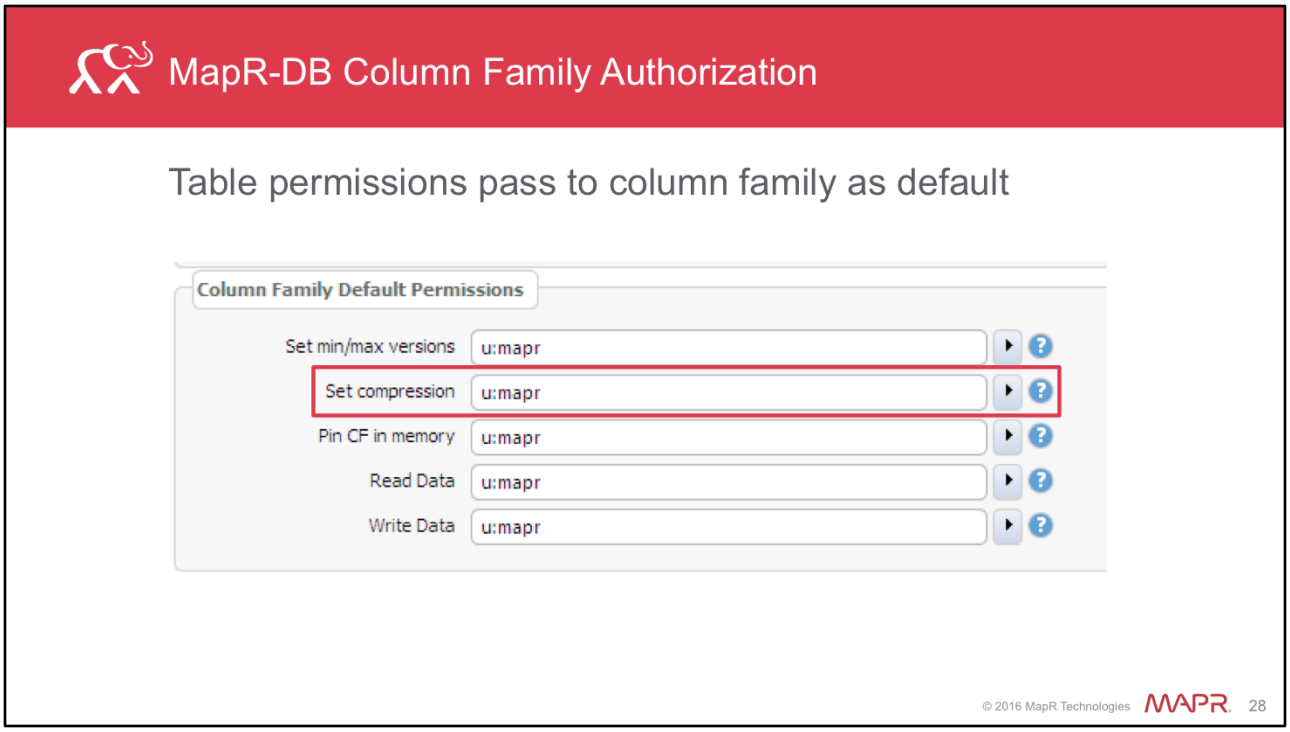


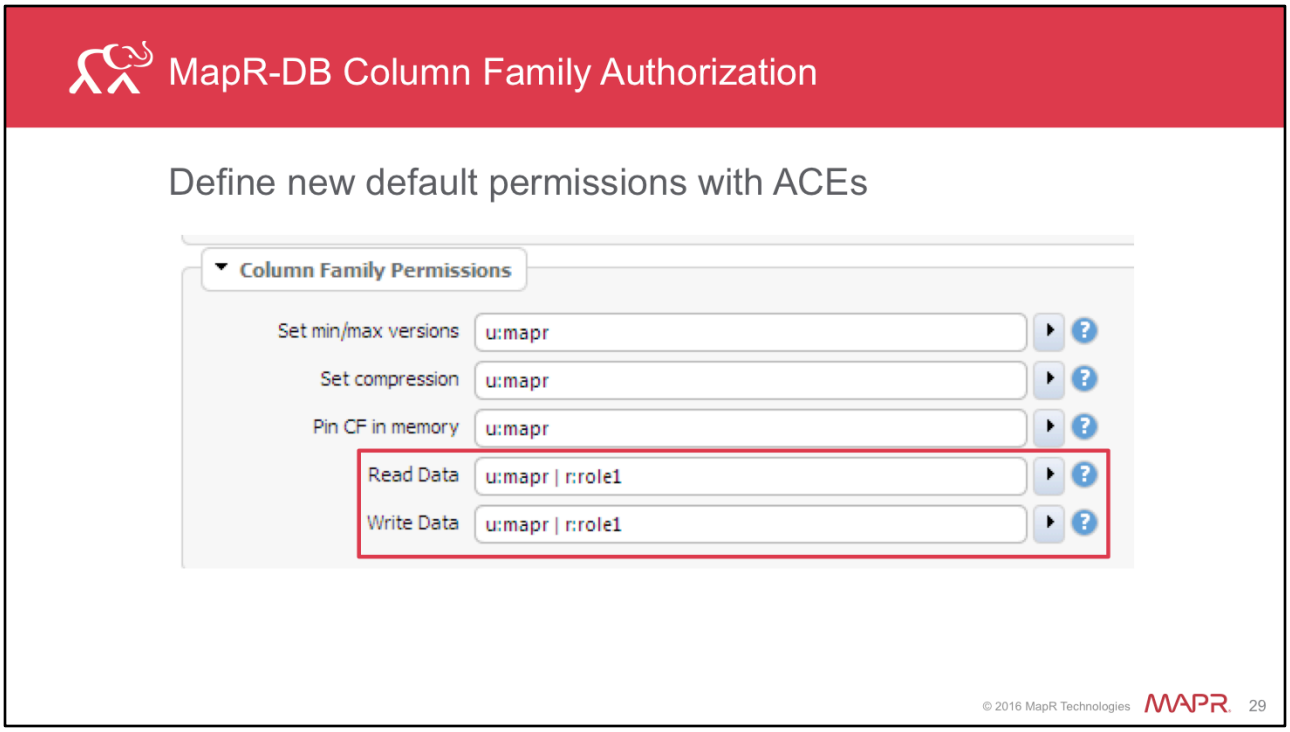
Table level ACEs mainly deal with restricting users from modifying table attributes, that is adding, renaming or removing column families.



The permissions of a new table default to the UID of the user creating the table.



Column families inherit table permissions as their default permissions.



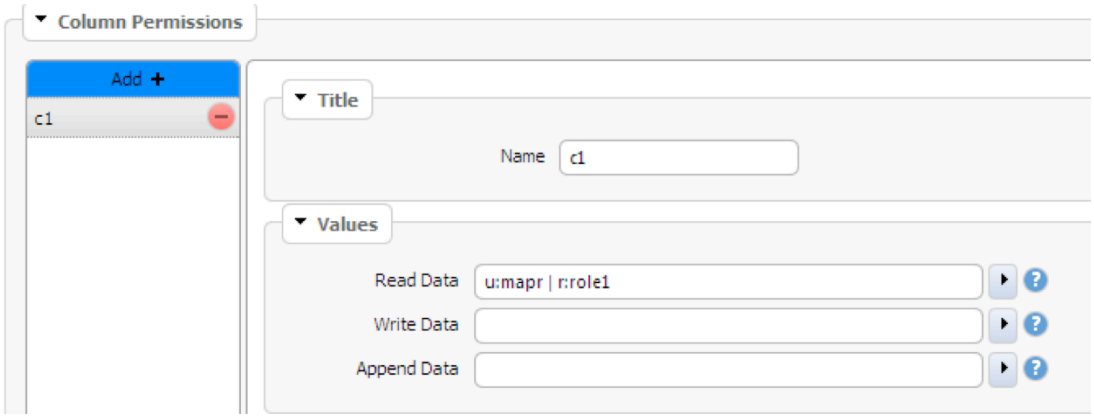
You can set default permissions for column families when you create or edit tables, and you can override these defaults when you create a new column family.

These ACEs get inherited by column-families when a new table is created.

These permissions default to the creating user, shown here as user mapr, but this screen allows you to set the default to another ACE.

MapR-DB Column Authorization

Column Access = Column Family & Column permissions



Unlike column families, columns are not predefined since users can dynamically add any column during a put operation.

Column access is an AND of the permissions on the column family AND column. If a user wants to get or put data from or to a particular column, he needs to pass access tests of both ColumnFamily and Column.

Example: MapR-DB Column Authorization

```
maprcli table cf edit -path /tables/customer -cfname Address -writeperm "u:mapr"
maprcli table cf colperm set -path /tables/customer -cfname Address -name City -writeperm "u:mapr"
```

RowKey	Address			Order			
	Street	City	State	Date	ItemNo	Ship Address	Cost
axxx	val		val	val		val	
…							val
fxxx	val			val	val	val	
gxxx							val
…	val	val	val	val	val		
pxxx						val	

 31

For example, suppose user mapr tries to write data to the columns 'city' in column family 'address'. MapR-DB checks whether user mapr has write permission on 'address' AND 'city'. If user mapr does not have all permissions, MapR-DB returns an error that says access for the write is denied.

Shown here are the MapR command line interface commands for setting write permissions on the customer table column family Address and column City for the user mapr.

Example: MapR-DB Column Authorization

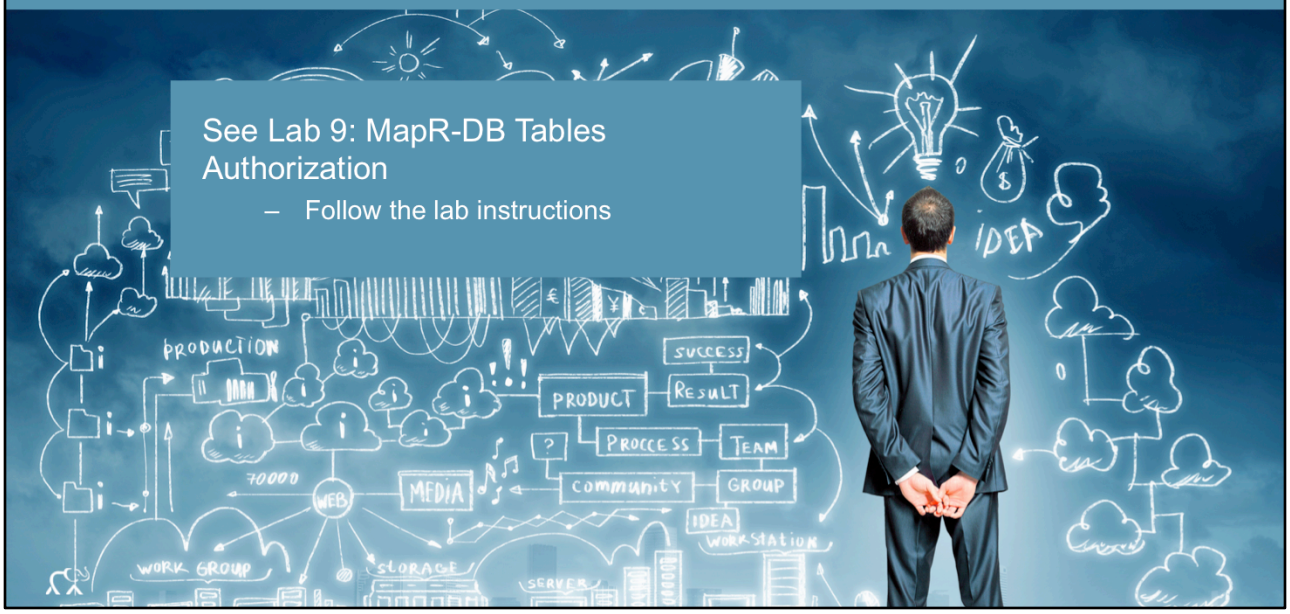
```
maprcli table cf colperm set -path /tables/customer -cfname Order -name Cost -readperm "u:mapr"
```

RowKey	Address			Order			
	Street	City	State	Date	ItemNo	Ship Address	Cost
axxx	val		val	val		val	
…							val
fxxx	val			val	val	val	
gxxx							val
…	val	val	val	val	val		
pxxx						val	

If this user mapr were to try to read from the City column, MapR-DB would simply not return the data. If the user mapr tried to read from this column and additional columns on which he does have read permissions, the results would contain the data for those additional columns but exclude the data for 'city'.

Shown here is the mapr command line interface command for setting read permissions on the customer table column family Order, column Cost for the user mapr.

Refer to Lab Guide for Exercise Instructions



Congratulations! You have completed DEV 340. Visit mapr.com/training to learn about additional courses available on the Hadoop ecosystem.