Bulk Loading Your Big Data into Apache HBase, a Full Walkthrough

Jean-Daniel Cryans

Strata + Hadoop World NYC 2014

About me

- Software Engineer at Cloudera, Storage team.
- Apache HBase committer since 2008, PMC member.

Agenda

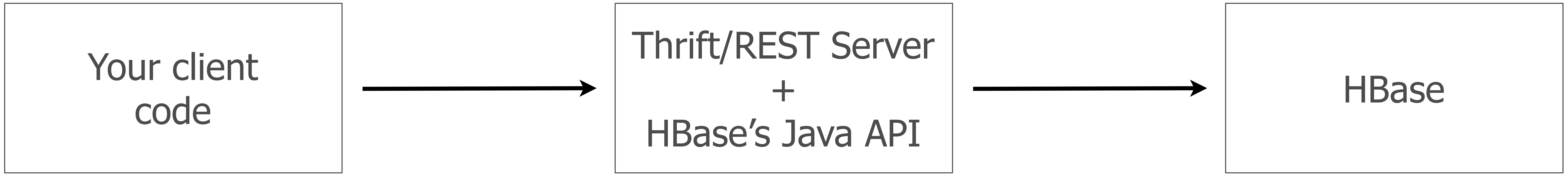
1. **HBase's write path**
2. Bulk loading concepts
3. ETL example
4. Issues and gotchas

Getting your BIG data in HBase

- Thrift/REST
- Java API
- MapReduce

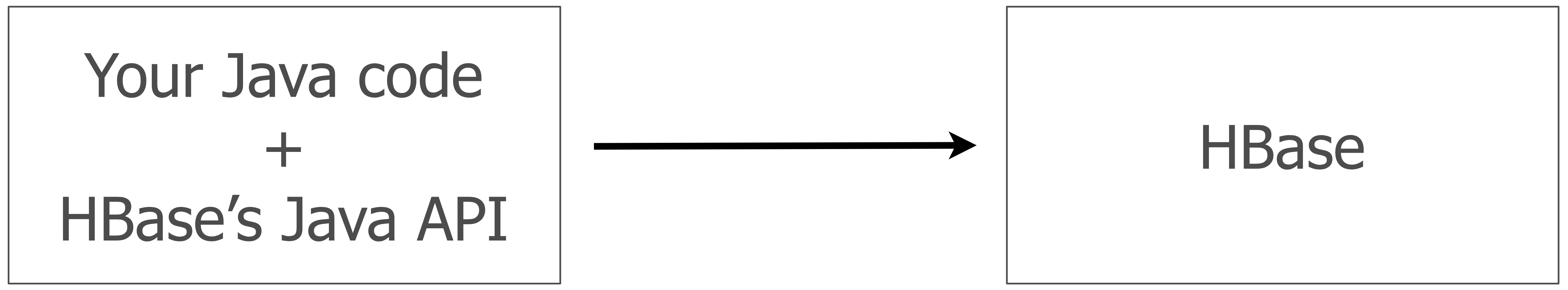
Getting your BIG data in HBase

- Thrift/REST
  - Low throughput due to indirection.
  - Need a way to have many clients.



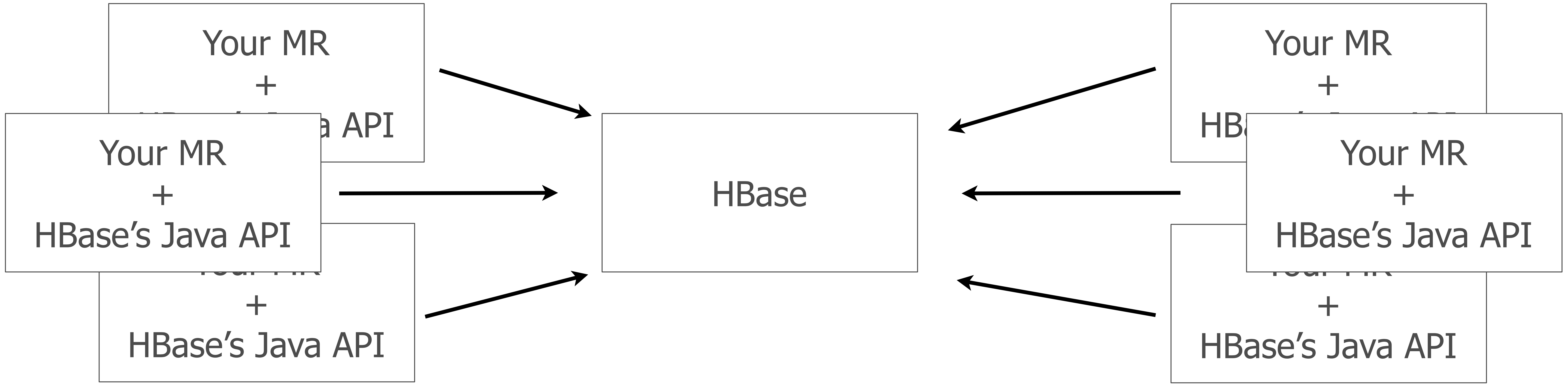
Getting your BIG data in HBase

- Java API
  - Indirection problem is solved.
  - Still need a way to have many clients.



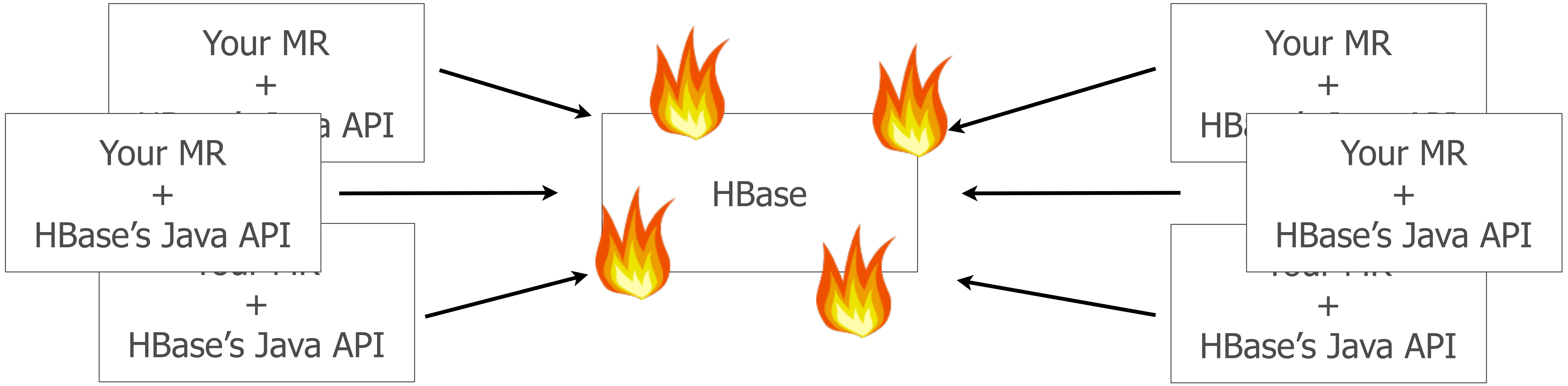
Getting your BIG data in HBase

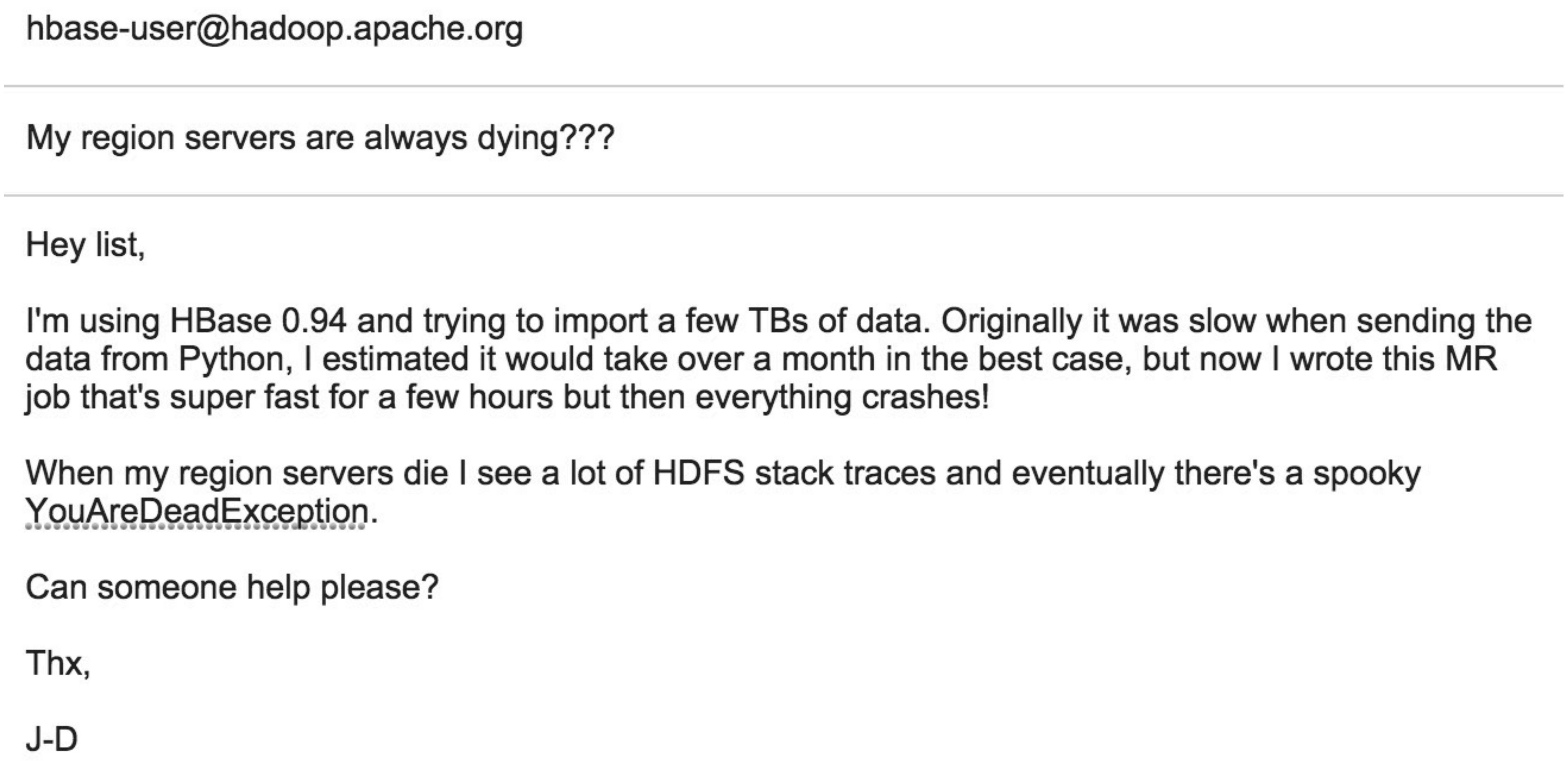
- MapReduce
  - No indirection.
  - No distribution problem, but…



Getting your BIG data in HBase

- MapReduce
  - No indirection.
  - No distribution problem, but...




hbase-user@hadoop.apache.org

My region servers are always dying???

Hey list,

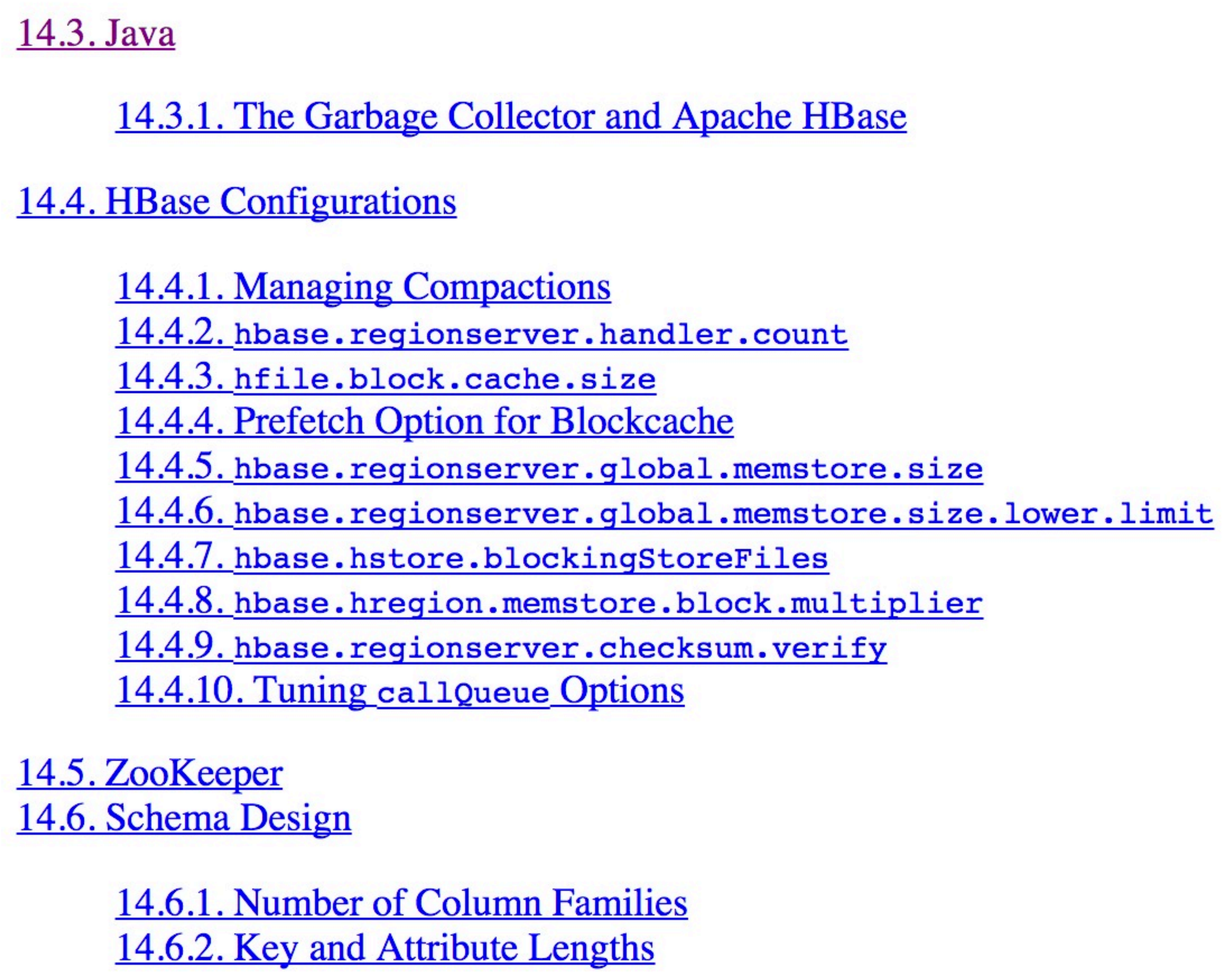
I'm using HBase 0.94 and trying to import a few TBs of data. Originally it was slow when sending the data from Python, I estimated it would take over a month in the best case, but now I wrote this MR job that's super fast for a few hours but then everything crashes!

When my region servers die I see a lot of HDFS stack traces and eventually there's a spooky YouAreDeadException.

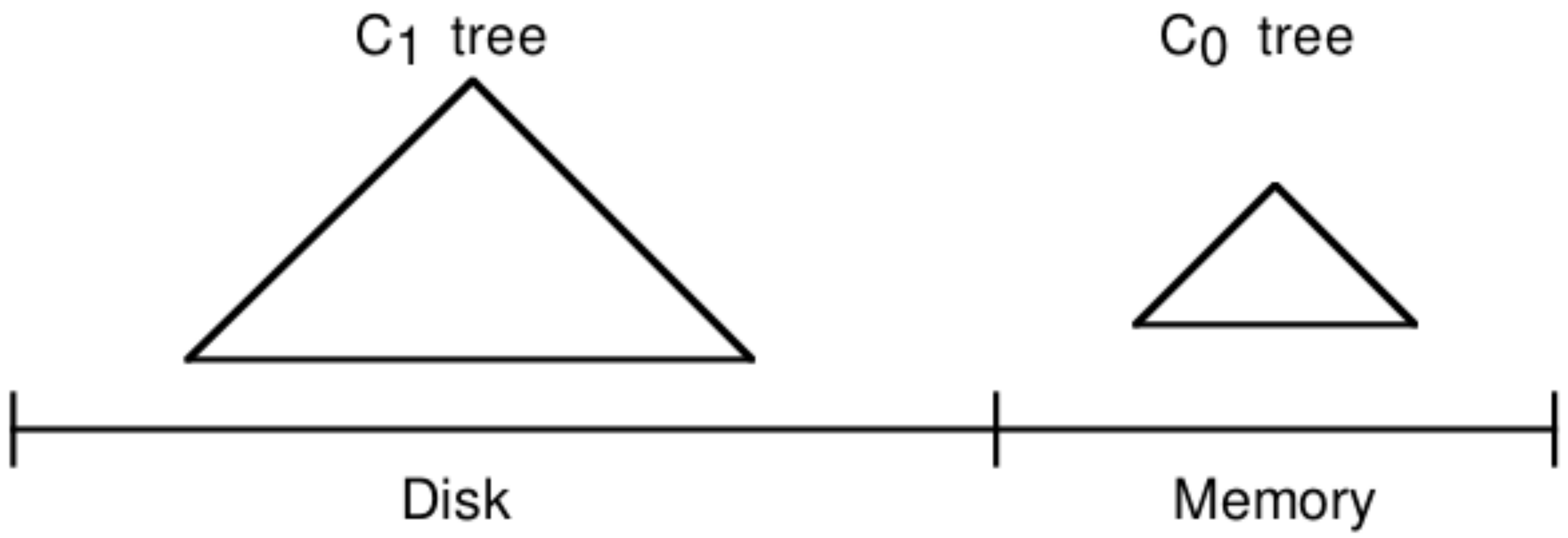
Can someone help please?

Thx,

J-D



Log-structured merge-trees



http://www.cs.umb.edu/~poneil/lsmtree.pdf

A quick intro to LSM trees

- Data is written in memory to C0.
- C0 flushes upon reaching a certain threshold.
- On-disk components C1-Ck are merged in the background.

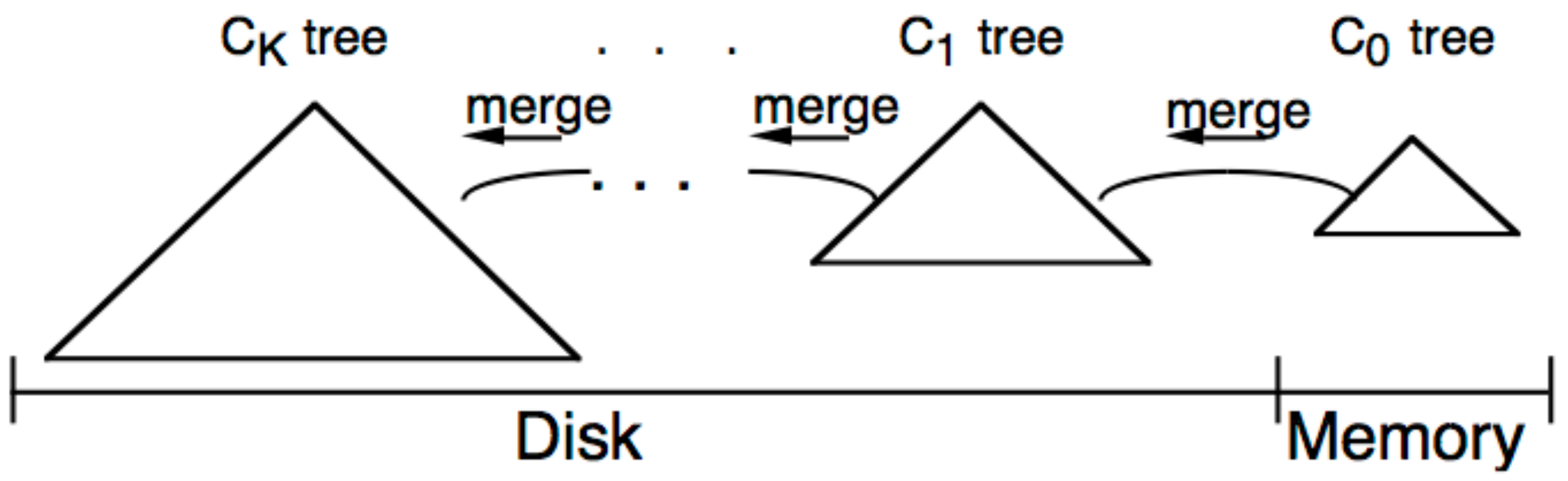
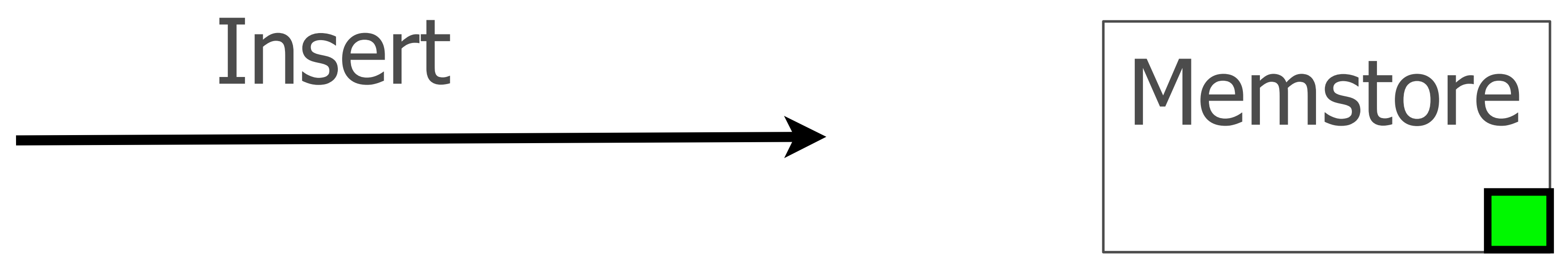


Figure 3.1. An LSM-tree of K+1 components

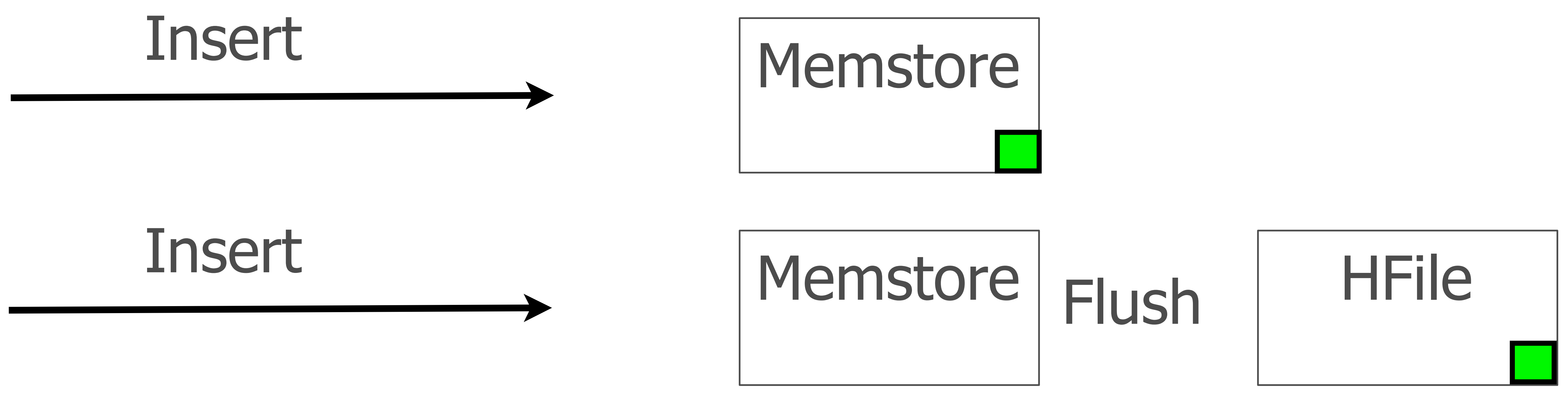
http://www.cs.umb.edu/~poneil/lsmtree.pdf

LSM trees in HBase

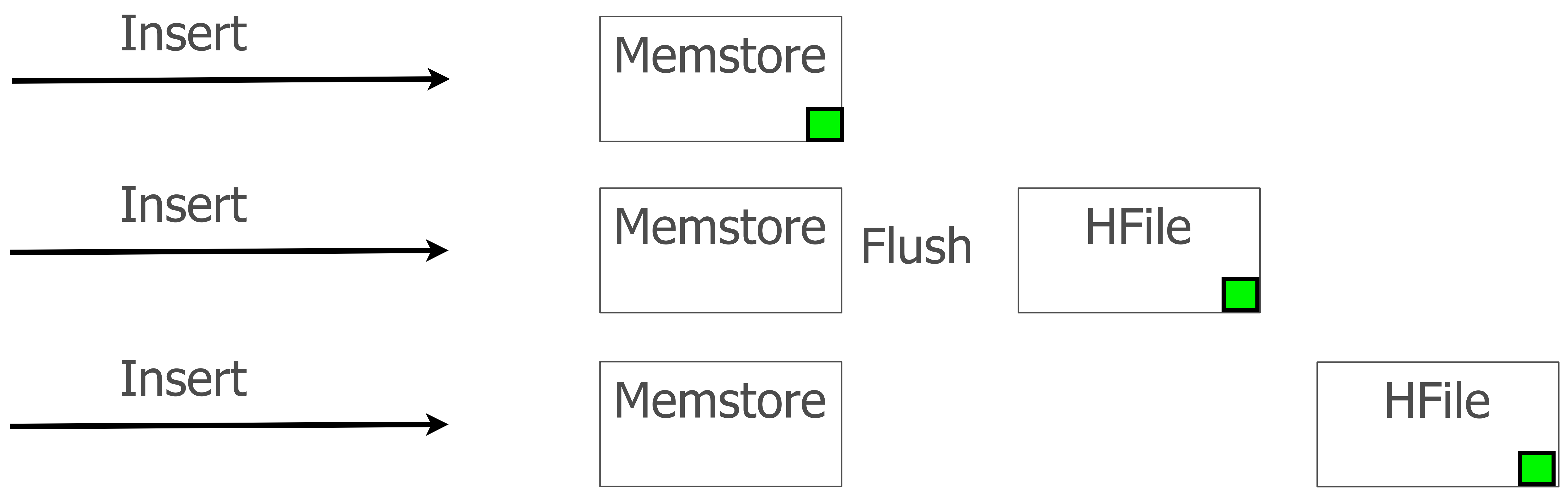
LSM trees in HBase



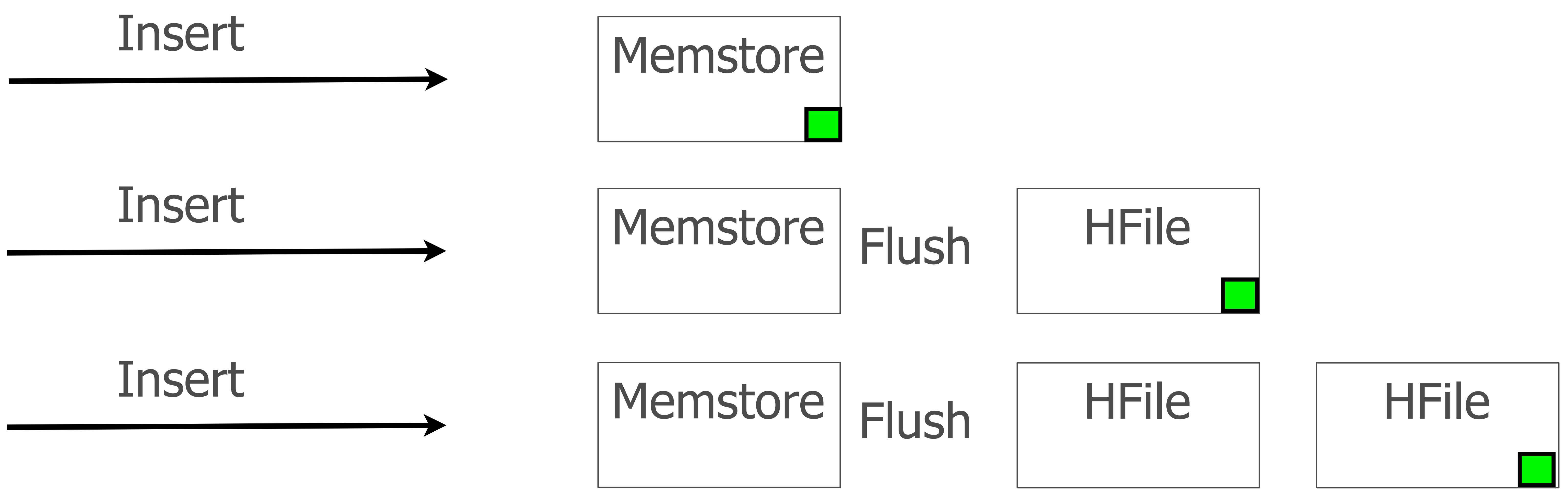
LSM trees in HBase



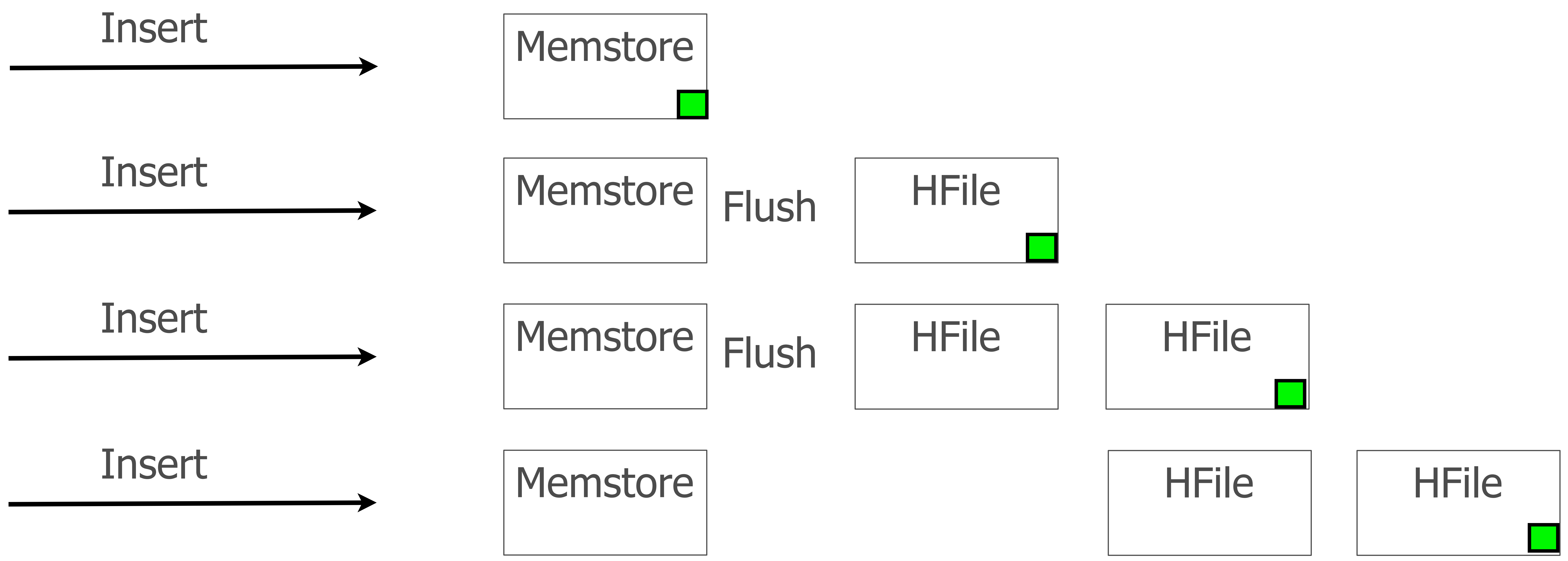
LSM trees in HBase



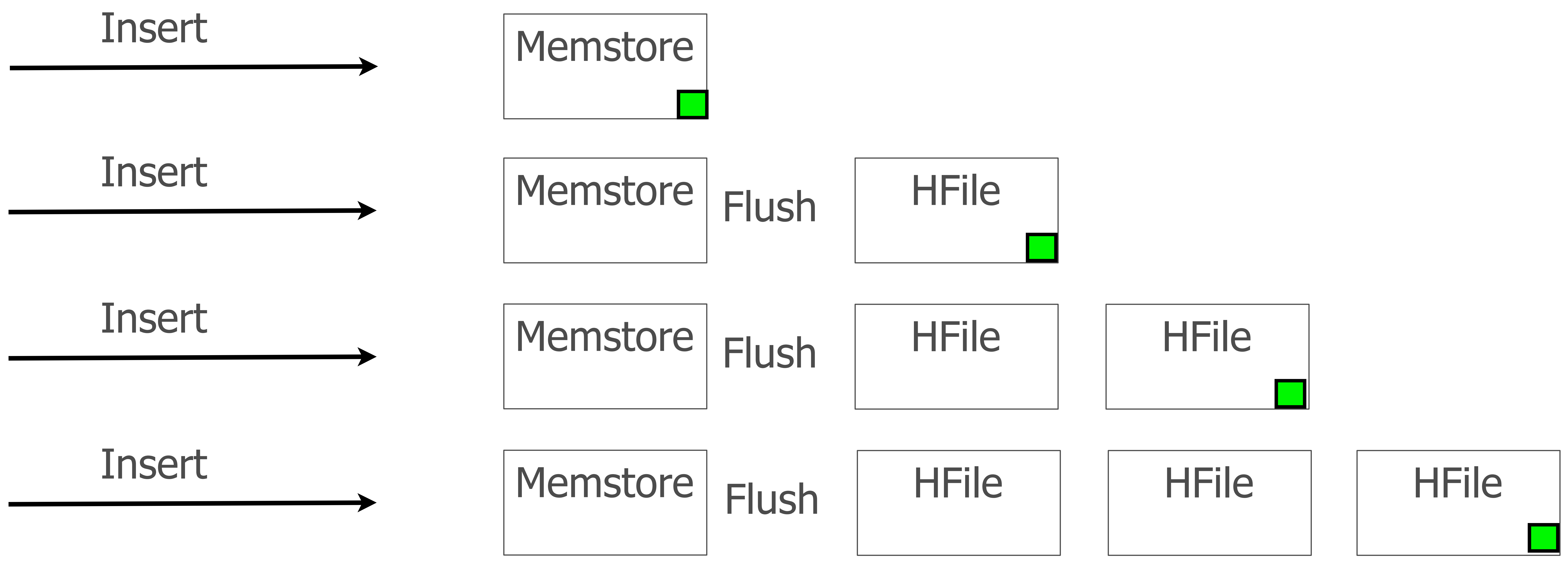
LSM trees in HBase



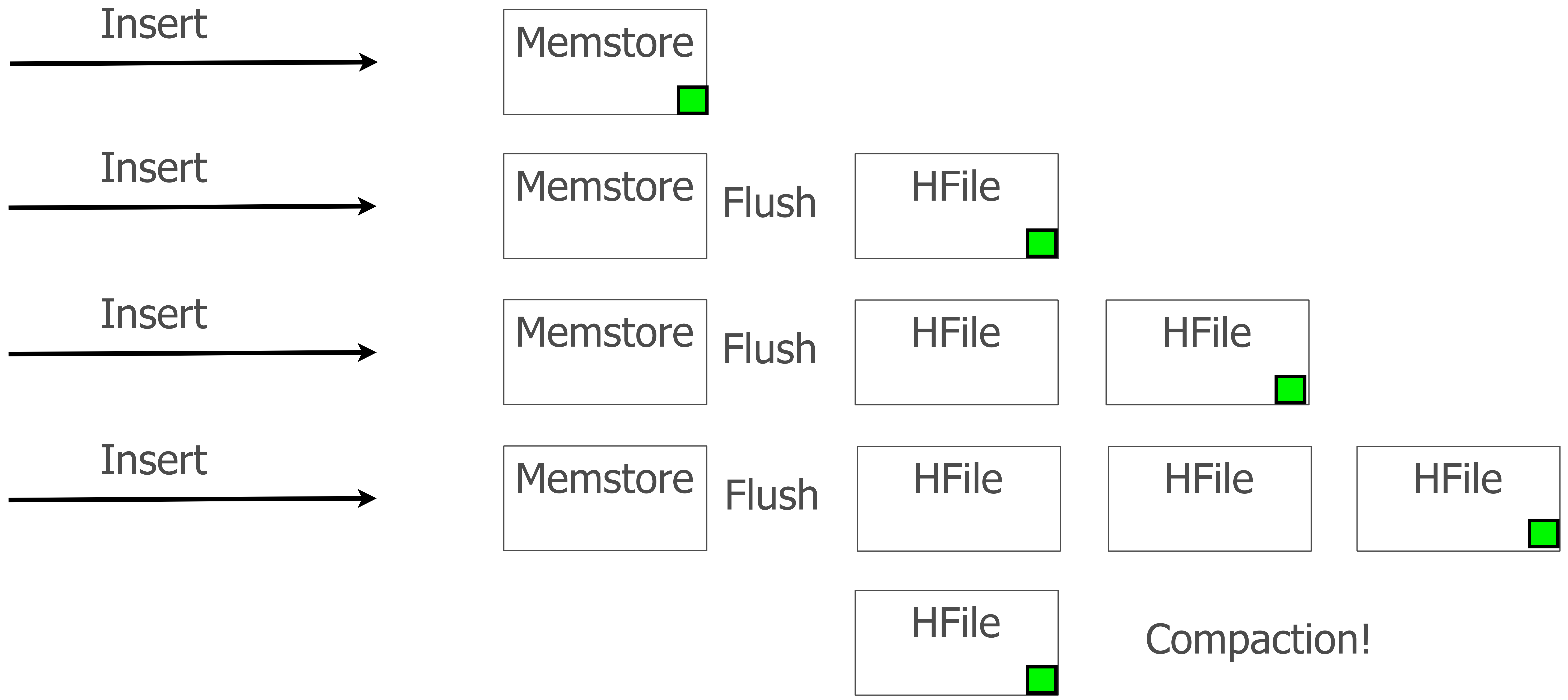
LSM trees in HBase



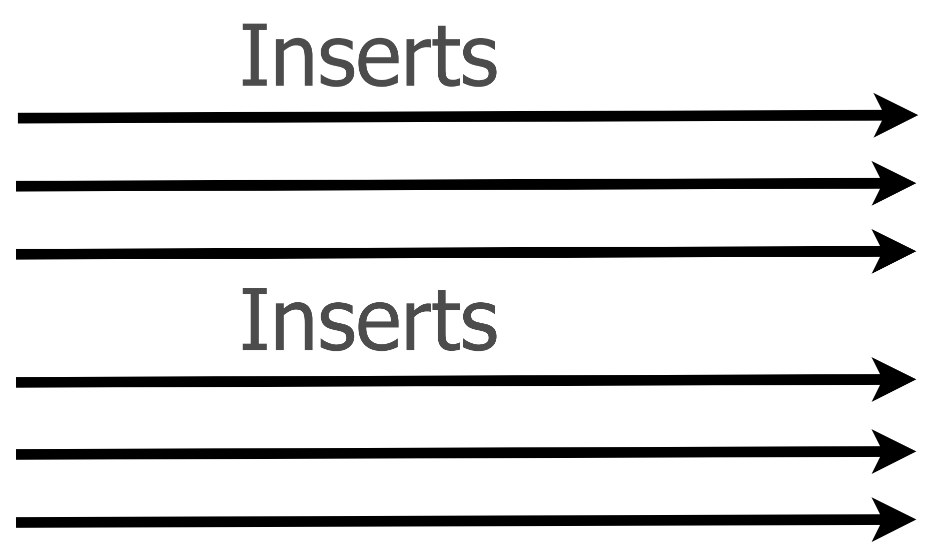
LSM trees in HBase



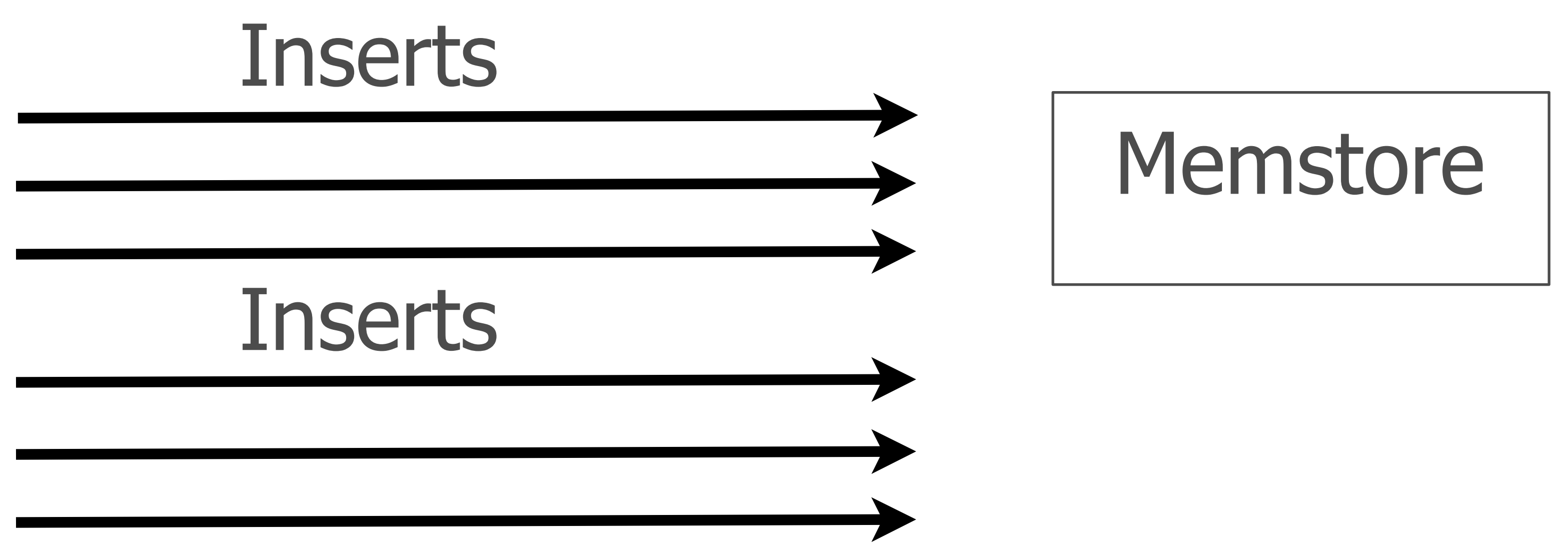
LSM trees in HBase



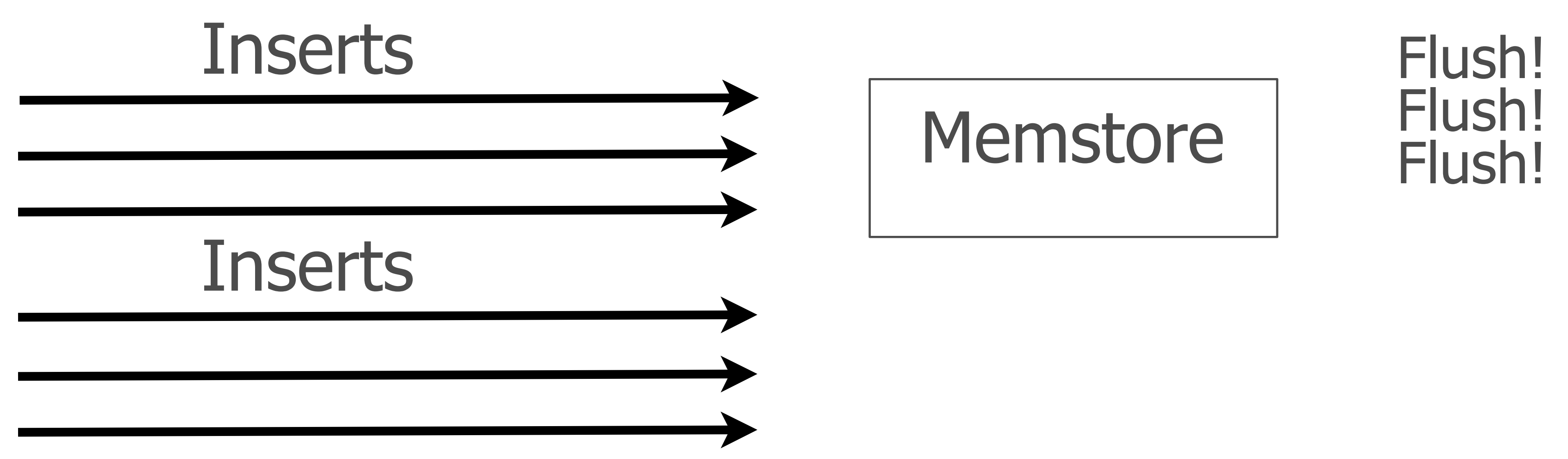
LSM trees in HBase



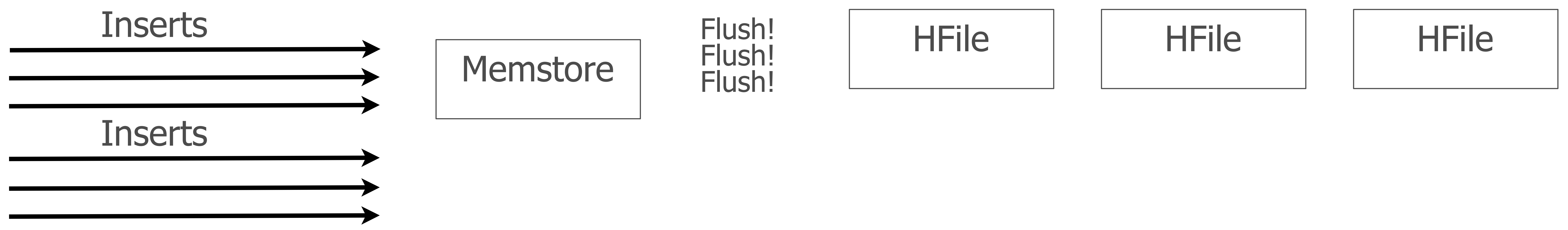
LSM trees in HBase



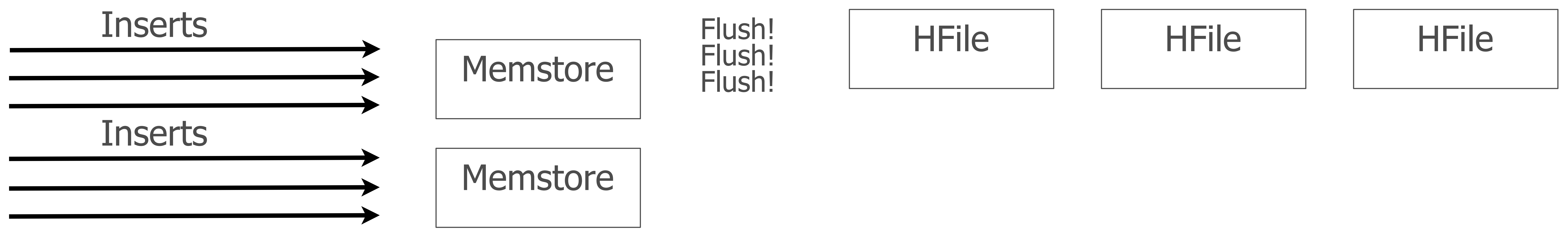
LSM trees in HBase



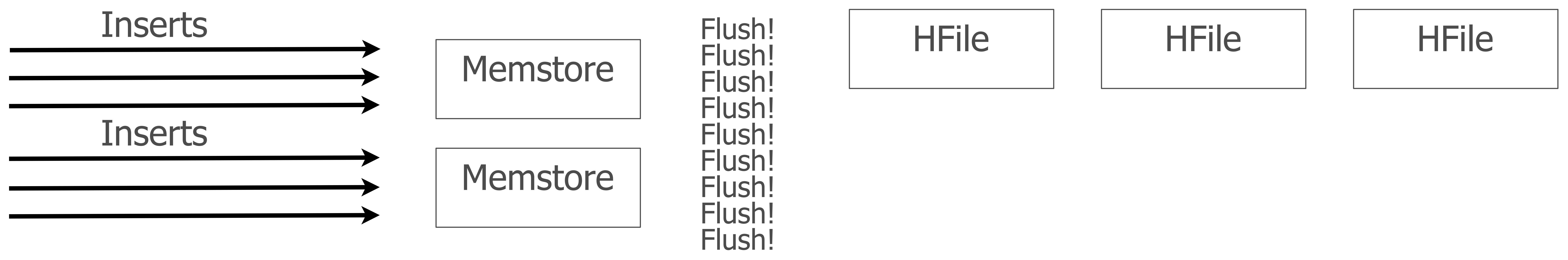
LSM trees in HBase



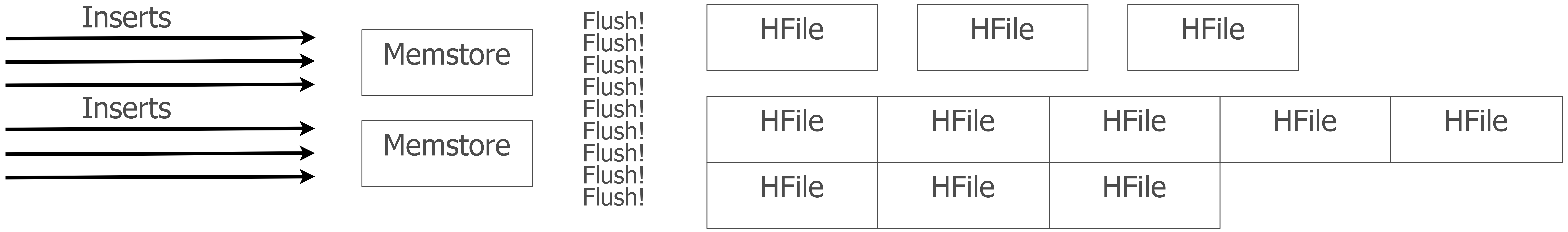
LSM trees in HBase



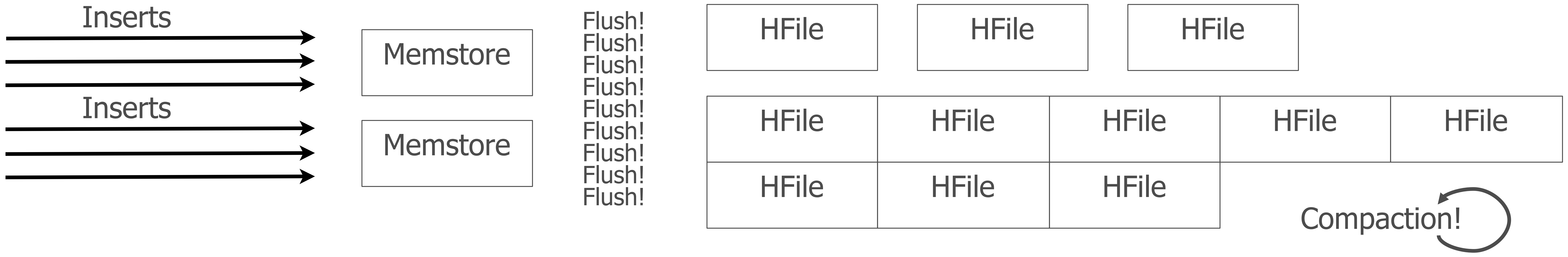
LSM trees in HBase



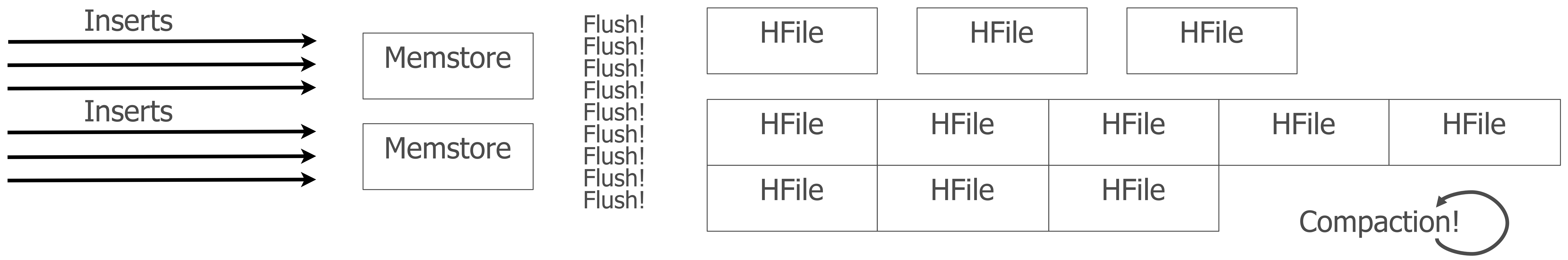
LSM trees in HBase



LSM trees in HBase



LSM trees in HBase



- How many times will data be rewritten?
- What kind of tuning could make this better?
- What about splitting those regions?

LSM trees in HBase

HFile

Or is there a way to just get the final result directly in HBase?

Agenda

1. HBase’s write path
2. **Bulk loading concepts**
3. ETL example
4. Issues and gotchas

Bulk loading overview

- Goal: generate data files in HBase's own format, respecting the region boundaries, and give them to the region servers.
- Use cases:

Bulk loading overview

- Goal: generate data files in HBase's own format, respecting the region boundaries, and give them to the region servers.
- Use cases:

Initial Data Import
Example:

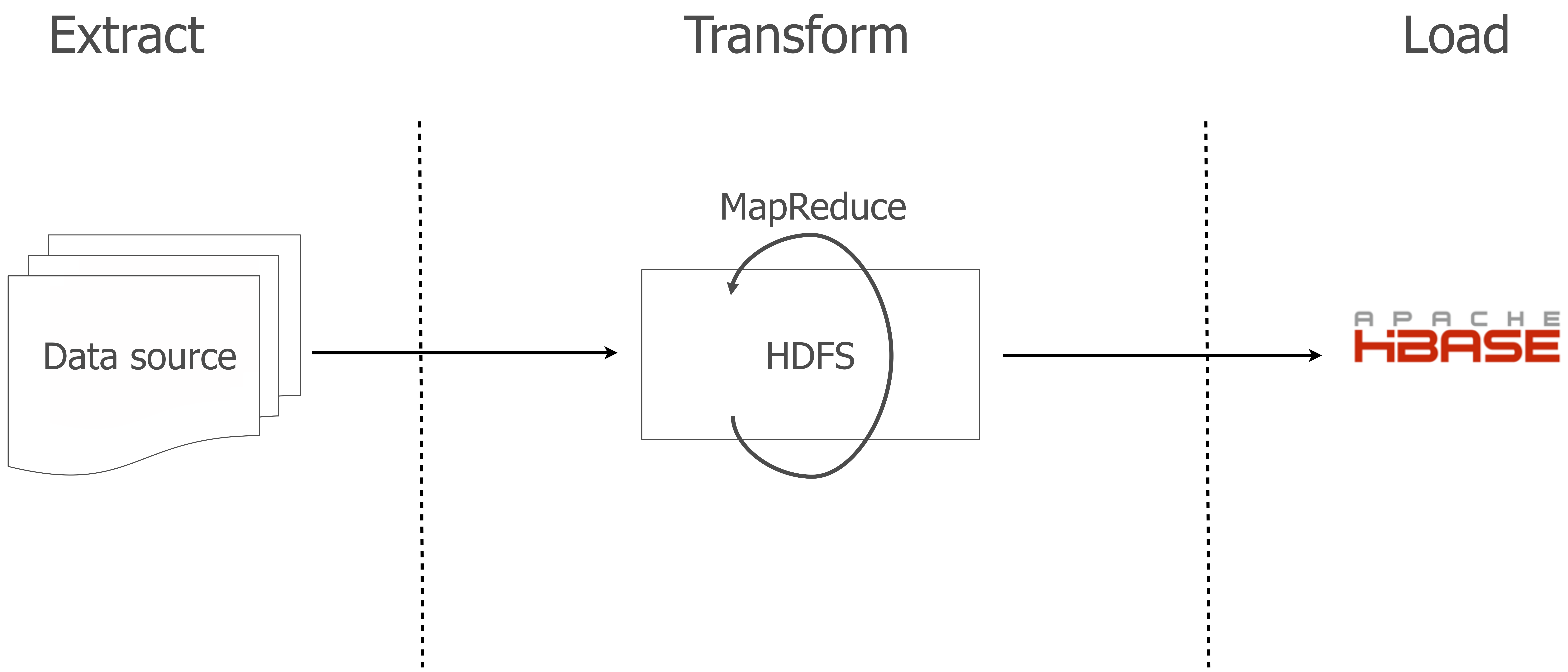
Bulk loading overview

- Goal: generate data files in HBase’s own format, respecting the region boundaries, and give them to the region servers.
- Use cases:

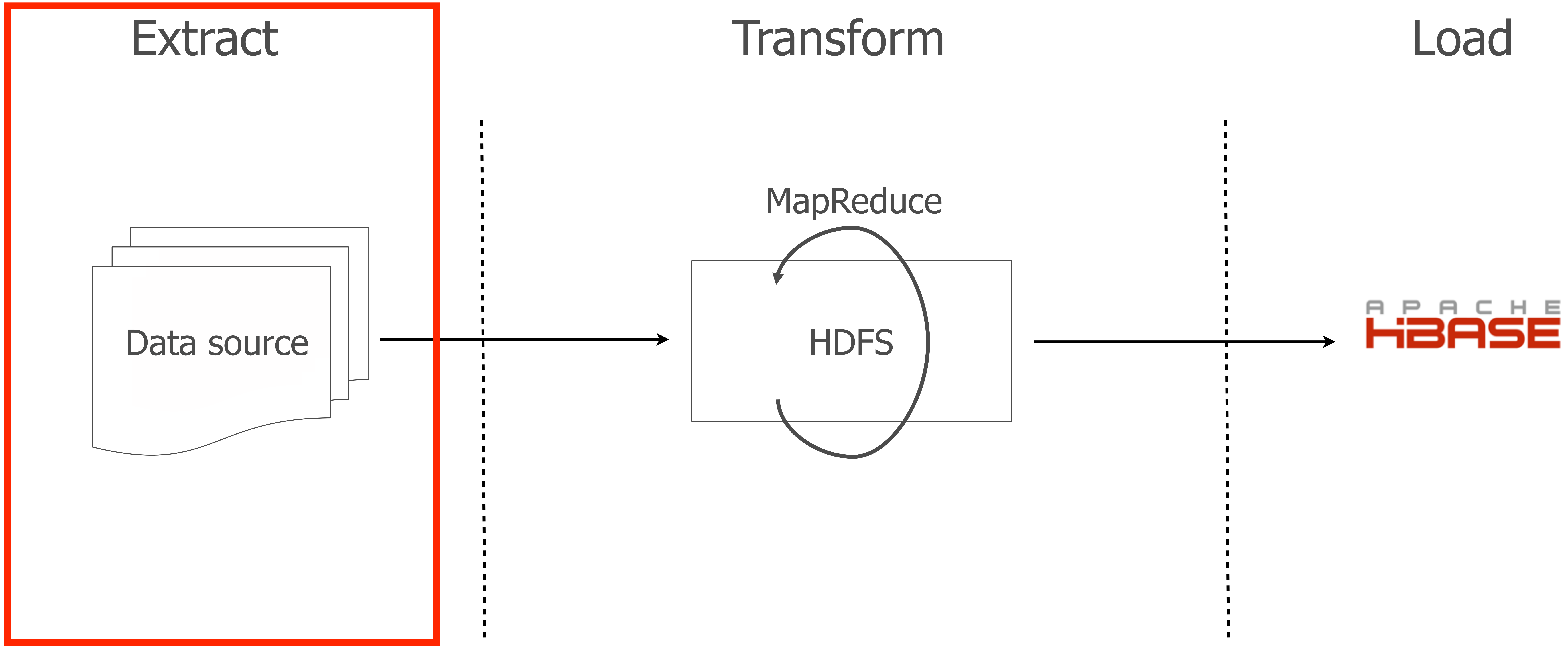
Initial Data Import
Example:

Regular Imports
Example:

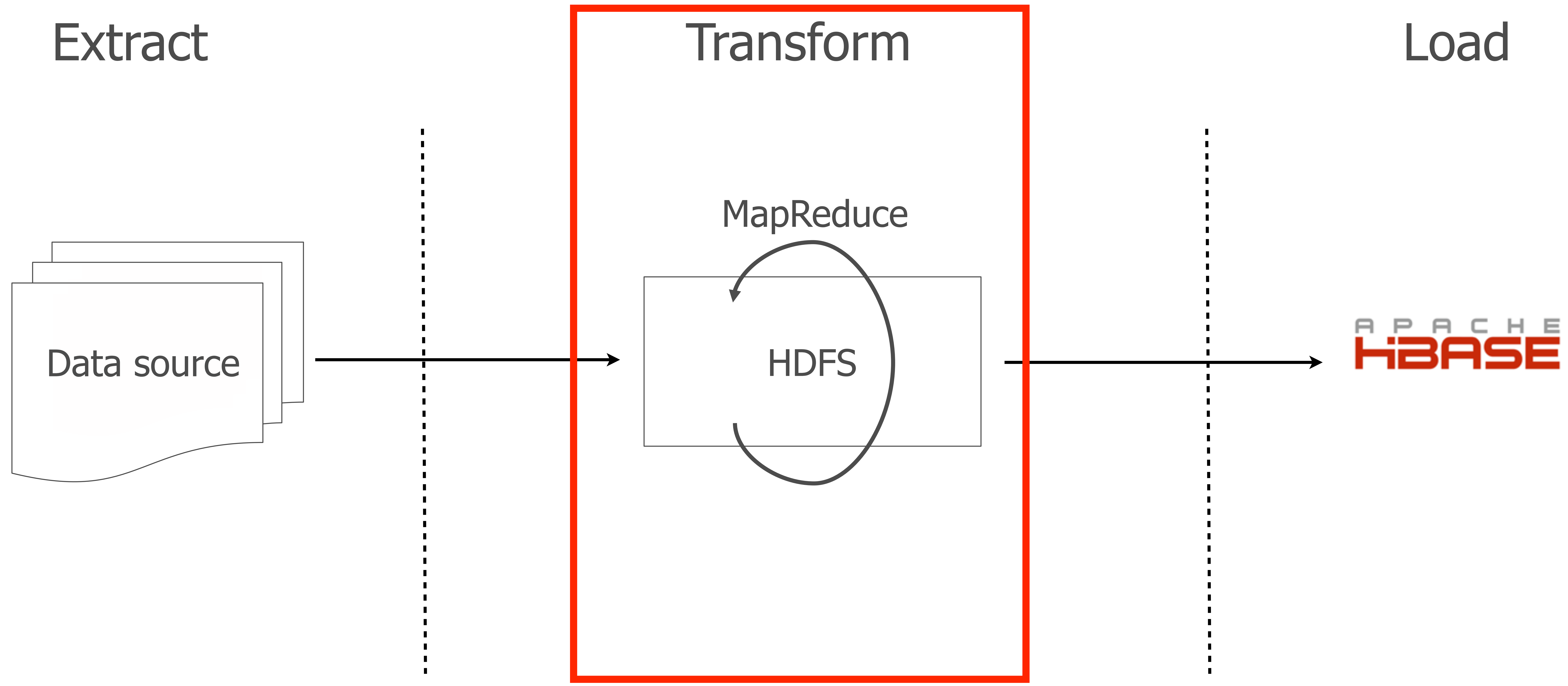
Bulk loading data flow



Bulk loading data flow



Bulk loading data flow



Transforming data into HFiles

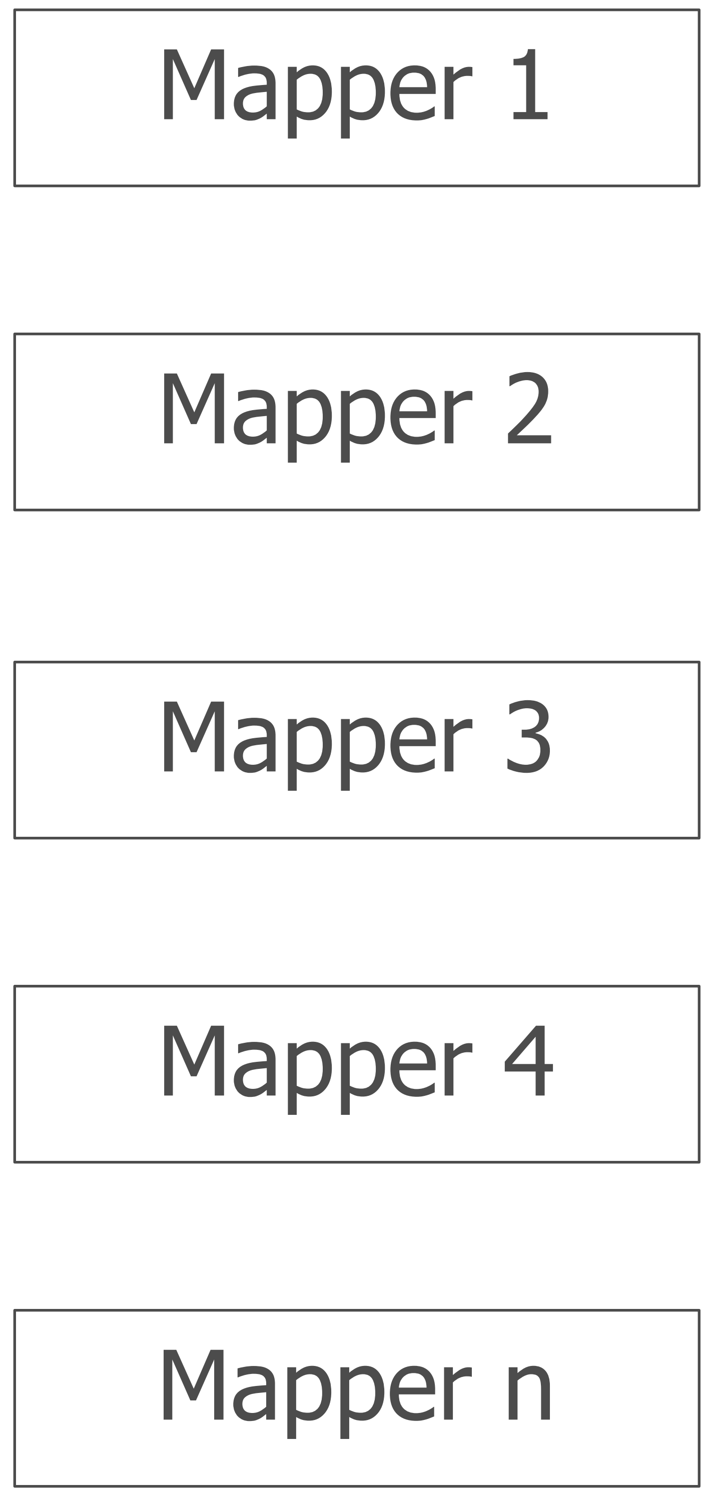
```
HTable table = new HTable(conf, tableName);
job.setReducerClass(PutSortReducer.class);
Path outputDir = new Path(hfileOutPath);
FileOutputFormat.setOutputPath(job, outputDir);
job.setMapOutputKeyClass(ImmutableBytesWritable.class);
job.setMapOutputValueClass(Put.class);
HFileOutputFormat.configureIncrementalLoad(job, table);
```

Transforming data into HFiles

```
HFileOutputFormat.configureIncrementalLoad(job, table);
```

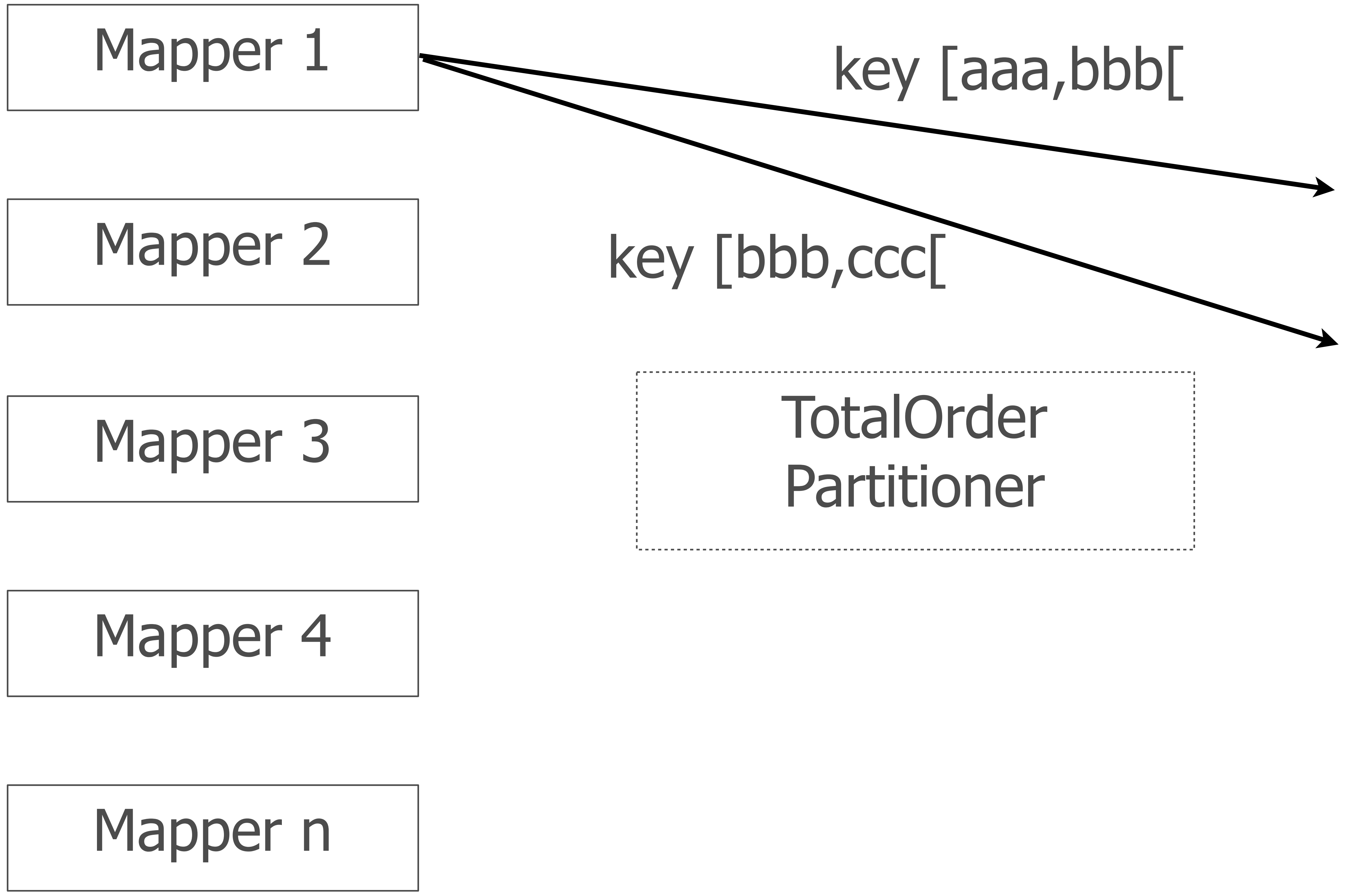
Transforming data into HFiles

```
HFileOutputFormat.configureIncrementalLoad(job, table);
```



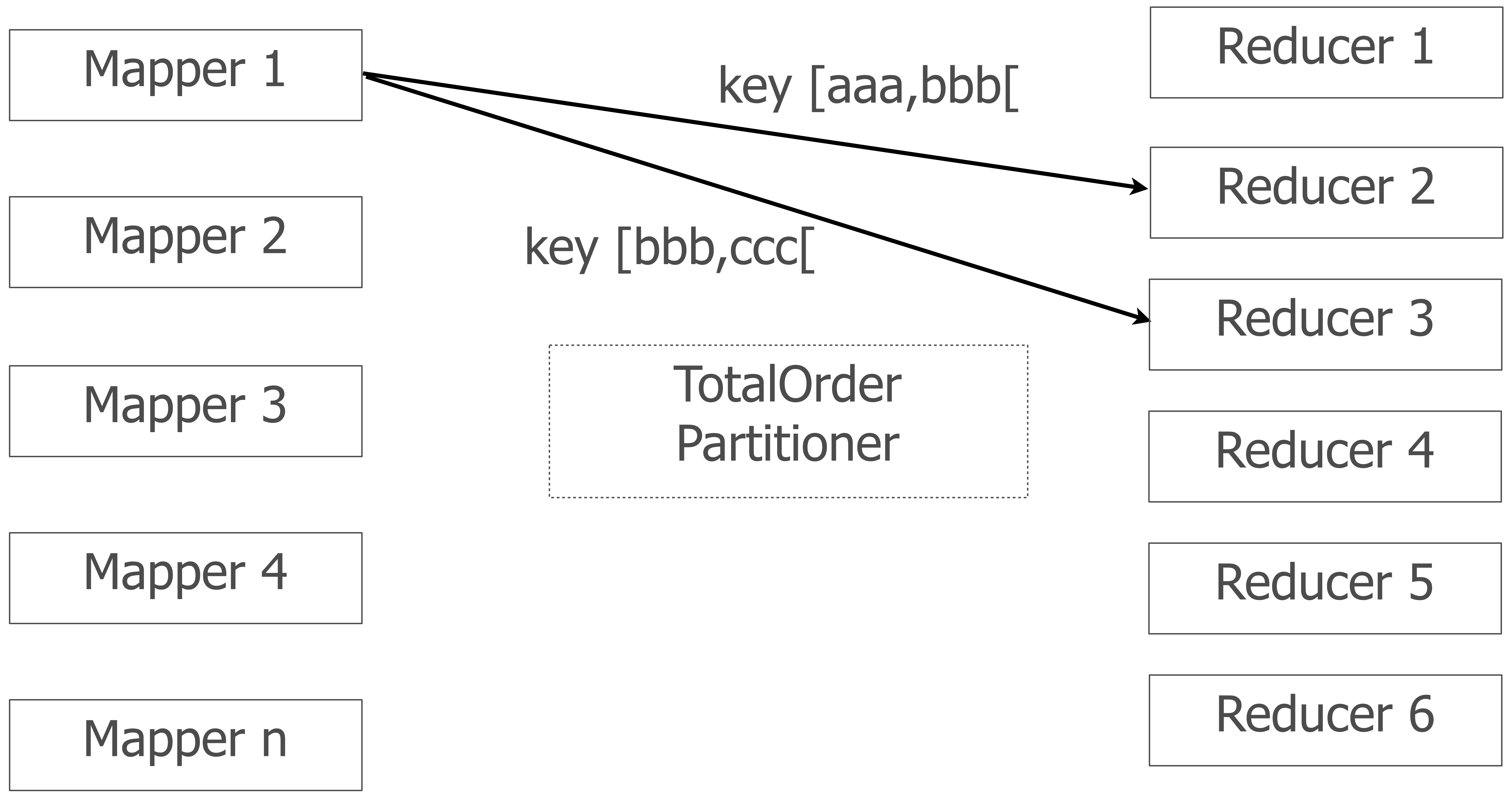
Transforming data into HFiles

```
HFileOutputFormat.configureIncrementalLoad(job, table);
```



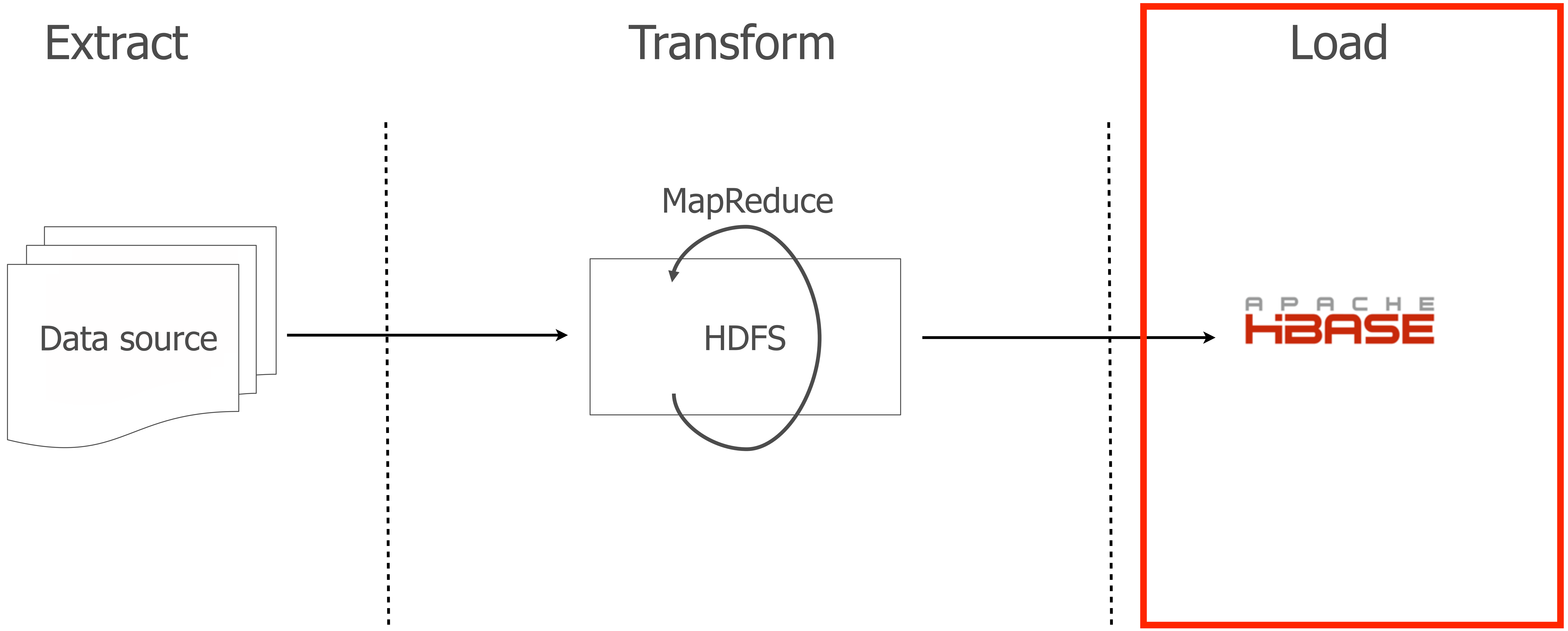
Transforming data into HFiles

```
HFileOutputFormat.configureIncrementalLoad(job, table);
```



Each reducer outputs one file per region.

Bulk loading data flow



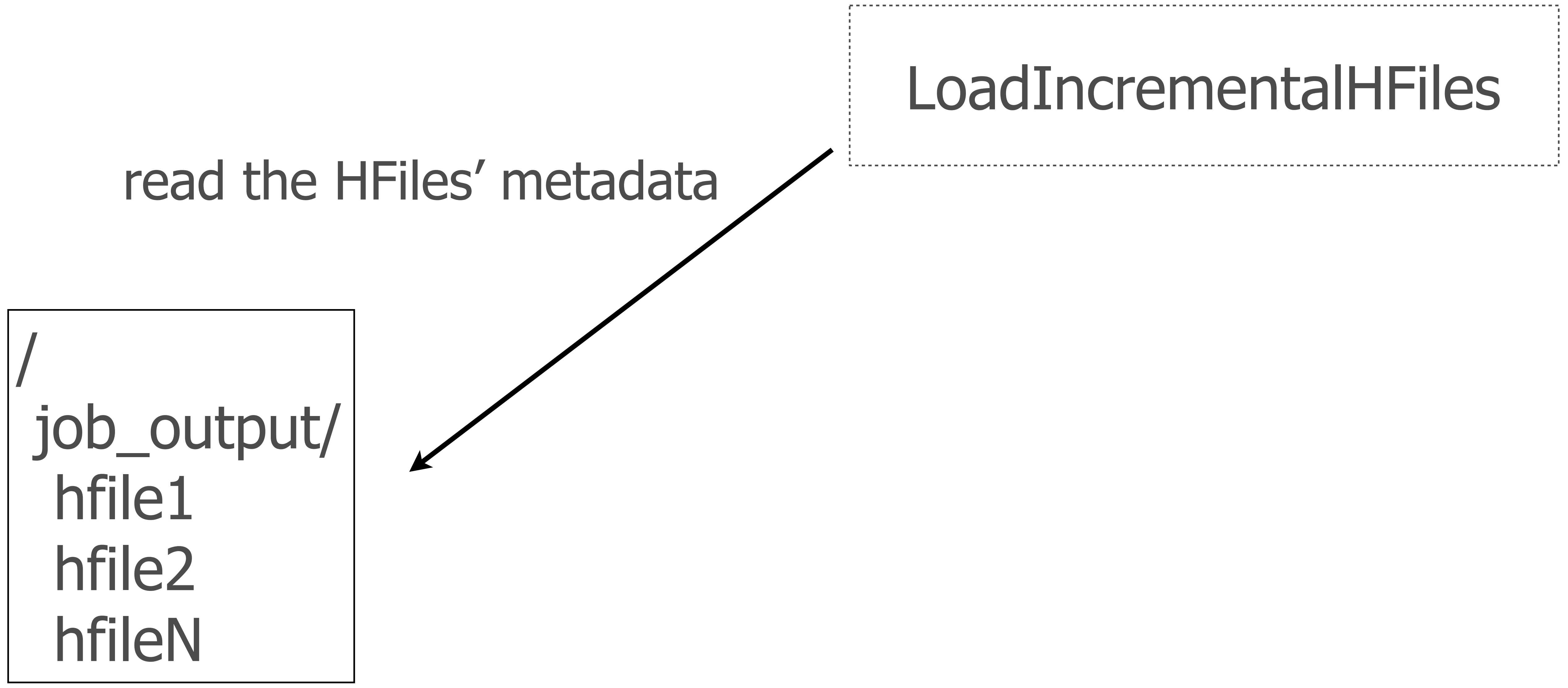
Loading HFiles

```
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>
```

```
/
 job_output/
  hfile1
  hfile2
  hfileN
```

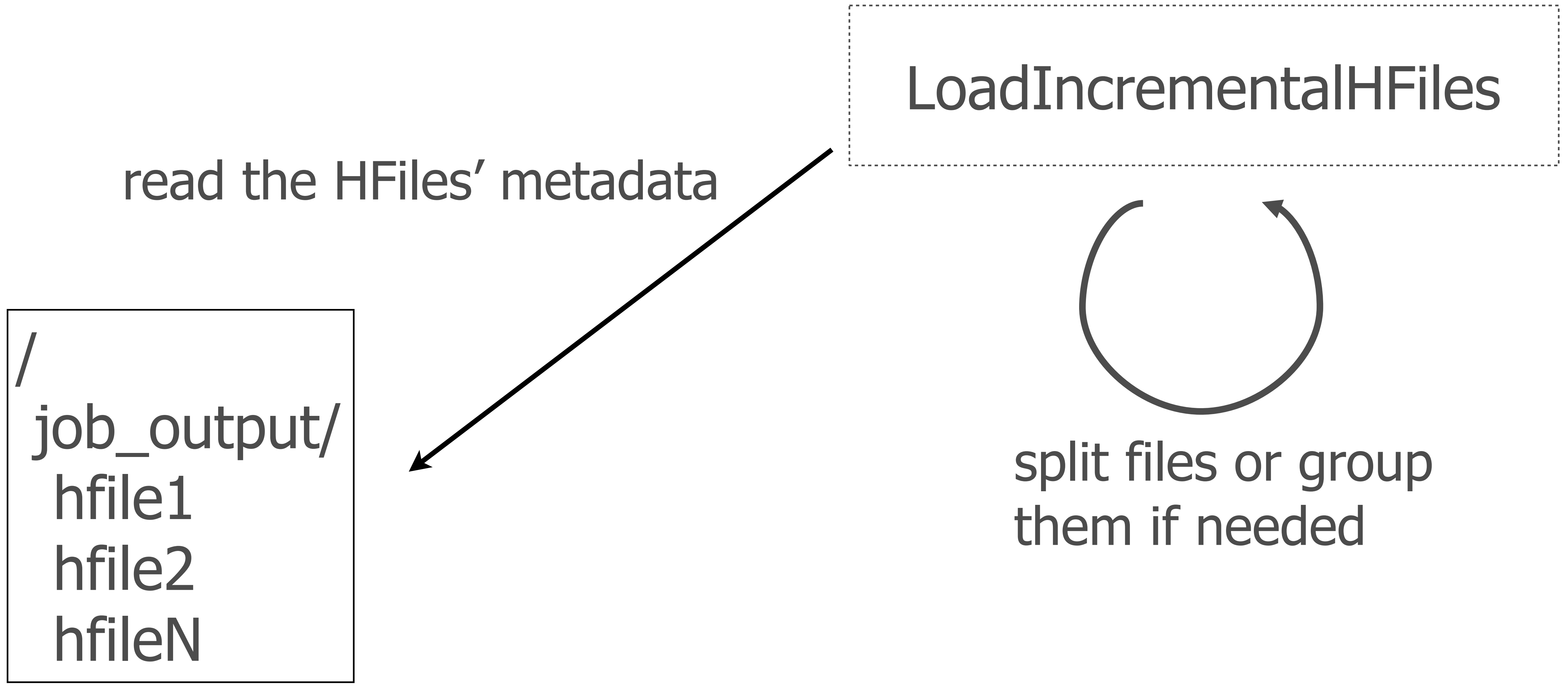
Loading HFiles

```
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>
```



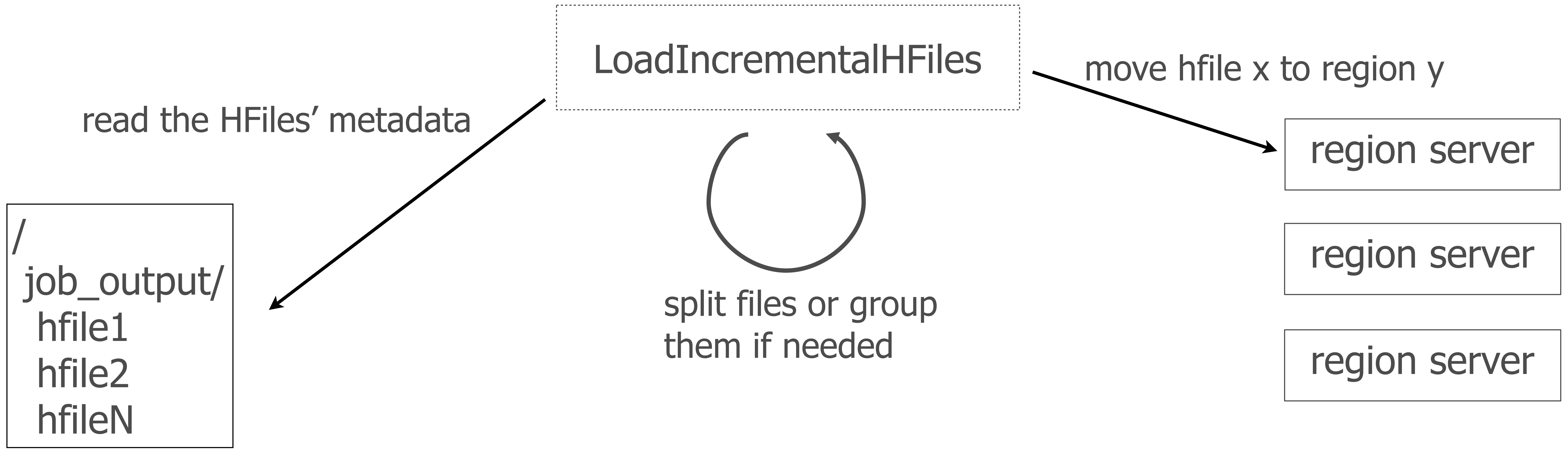
Loading HFiles

```
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>
```


LoadIncrementalHFiles
read the HFiles' metadata
/
job_output/
hfile1
hfile2
hfileN
split files or group them if needed

Loading HFiles

```
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>
```



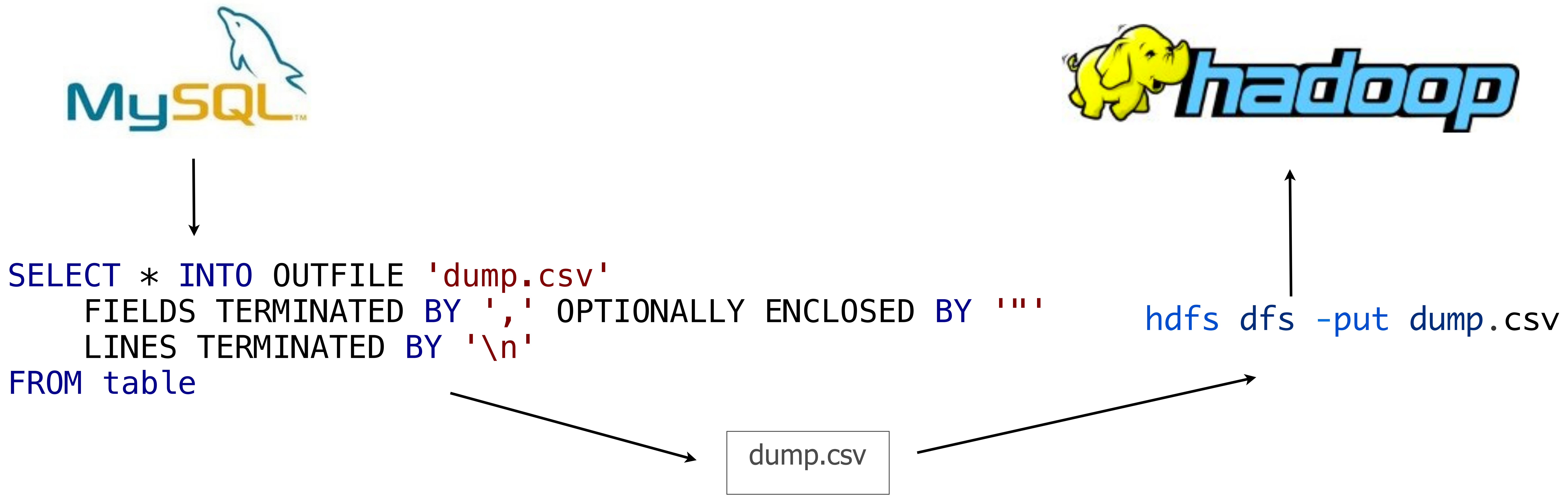
Agenda

1. HBase’s write path
2. Bulk loading concepts
3. **ETL example**
4. Issues and gotchas

MySQL Import

- Extract
  - CSV dump into file.
- Transform
  - Map columns, create HFiles.
- Load
  - Use LoadIncrementalHFiles.

Extract



```
SELECT * INTO OUTFILE 'dump.csv'
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
FROM table
```

```
hdfs dfs -put dump.csv
```

Transform

Map dump.csv

Reduce to output/

```
hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.separator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2 table-name dump.csv
```

Transform

Map dump.csv

Reduce to output/

```
hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.separator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2 table-name dump.csv
```

Transform

Map dump.csv
Reduce to output/

```
hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.separator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2 table-name dump.csv
```

Load

List the files
under output/

Tell each RS
to move them.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles output table-name
```

Agenda

1. HBase's write path
2. Bulk loading concepts
3. ETL example
4. **Issues and gotchas**

Planning the bulk load; gotchas

- Initial import
  - Tables must still be created, pre-split.

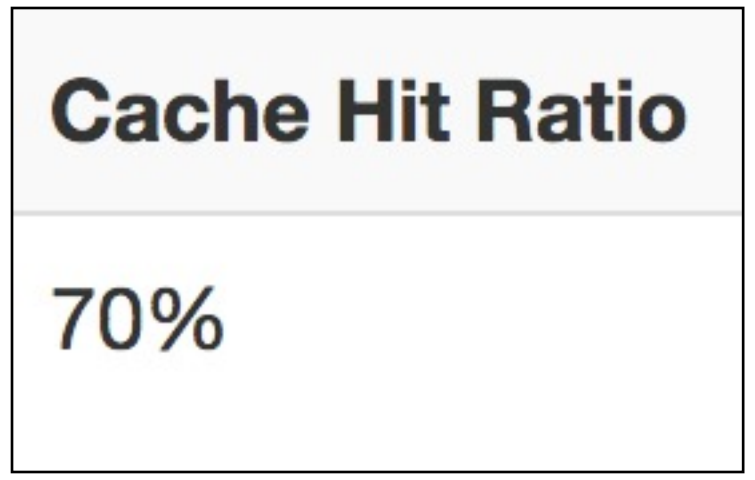
```
create 'table-name', {NAME => 'f'},   {SPLITS => ['a', 'b', 'c', 'd']}
```

  - Plan for the files to fit in the regions else it will split.

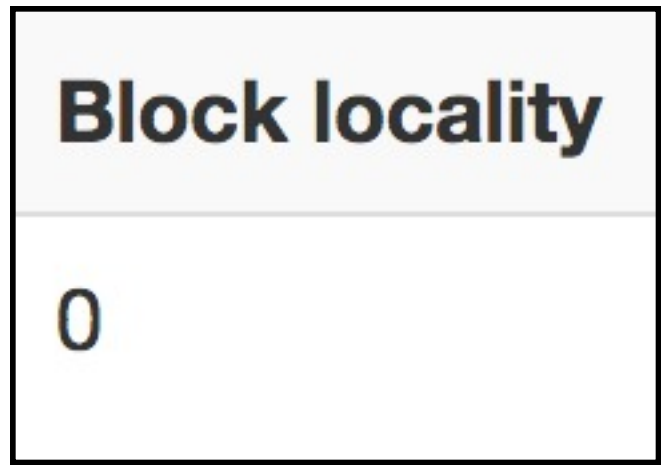
```
alter 'table-name', {MAX_FILESIZE => 10737418240}
```

Planning the bulk load; gotchas

- Regular import
  - Loading data on HDFS still not free, IO-wise.
    - Especially the Transform phase.
  - Data won't be in the block cache once Loaded.
  - Block locality isn't guaranteed.



Cache Hit Ratio
70%



Block locality
0

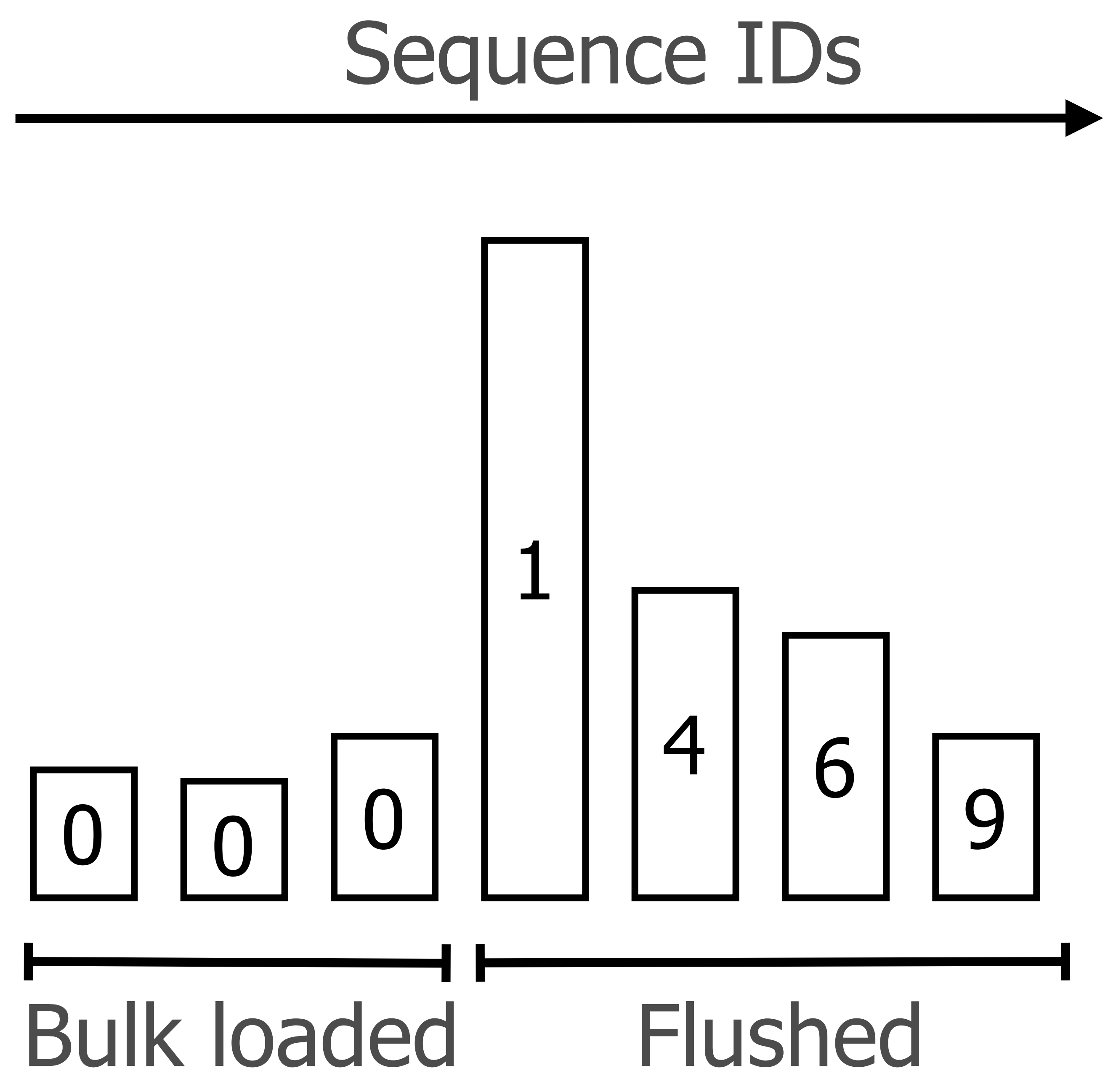
Gotchas: Security

- Problem:
  - The user “hbase” must move files it doesn’t have access to.

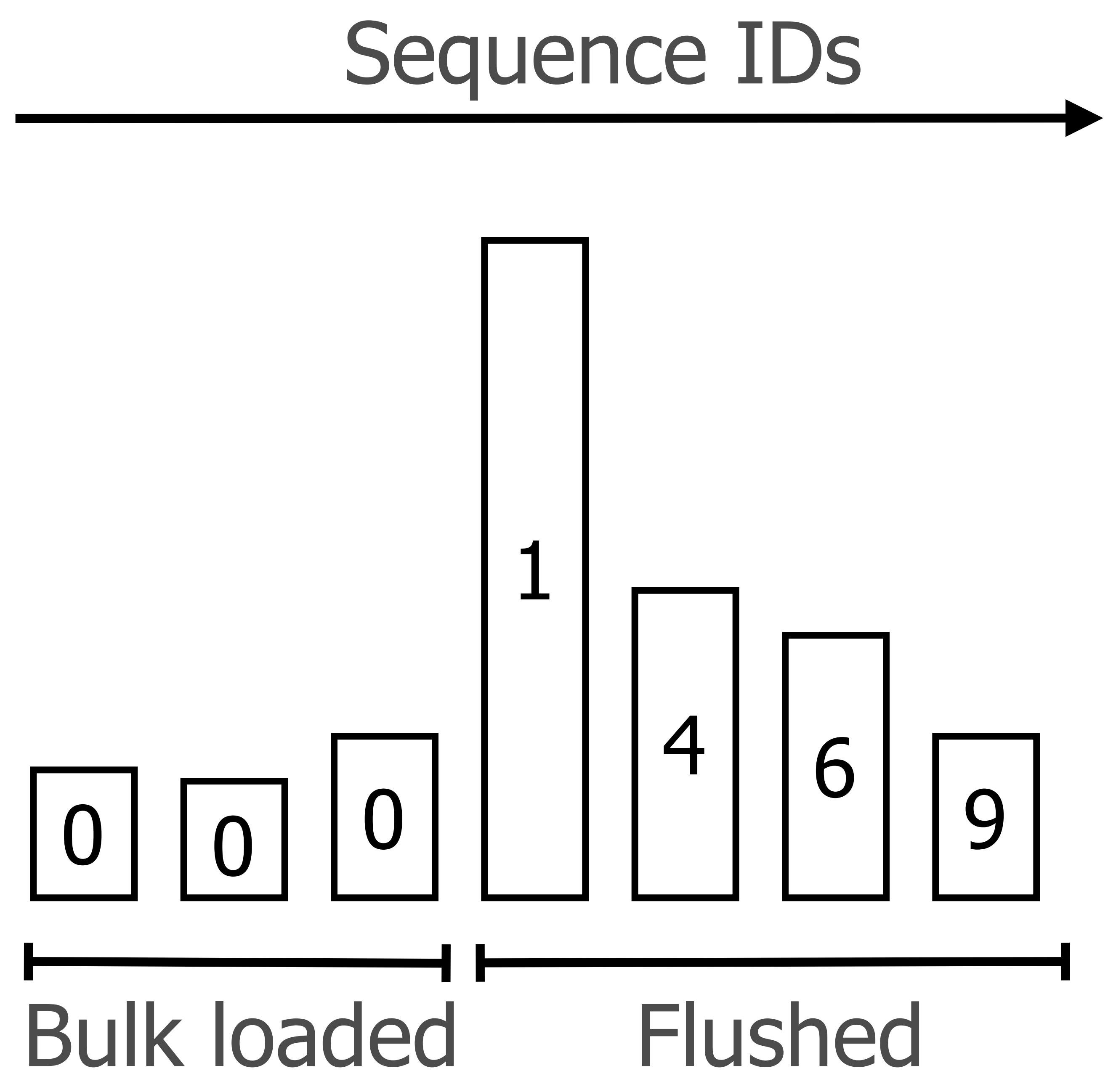
Gotchas: Security

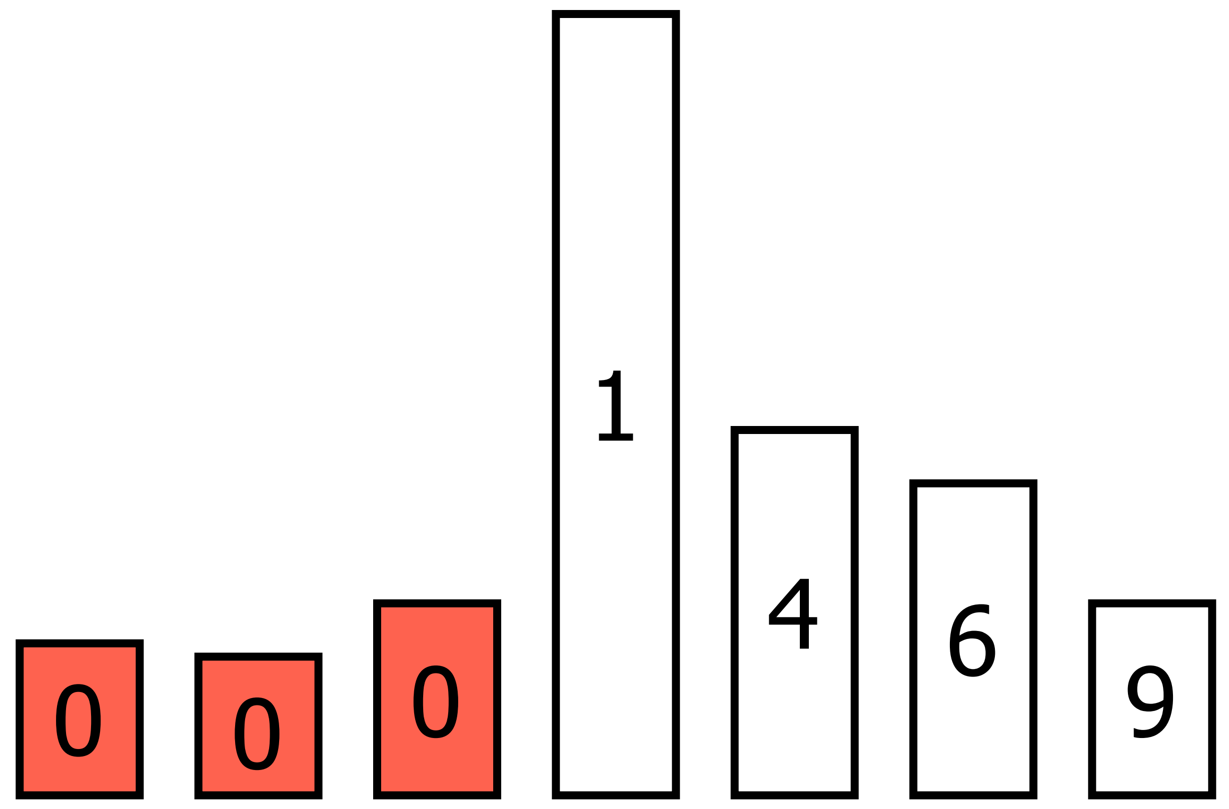
- Problem:
  - The user “hbase” must move files it doesn’t have access to.
- SecureBulkLoadEndpoint
  - Must be installed as part of enabling security.
  - A secret staging directory with 777 perms is used.
  - LoadIncrementalHFiles moves files there and then the RS moves it into its regions’ directories.

Gotchas: HBASE-8521 (fixed 0.94.13+)



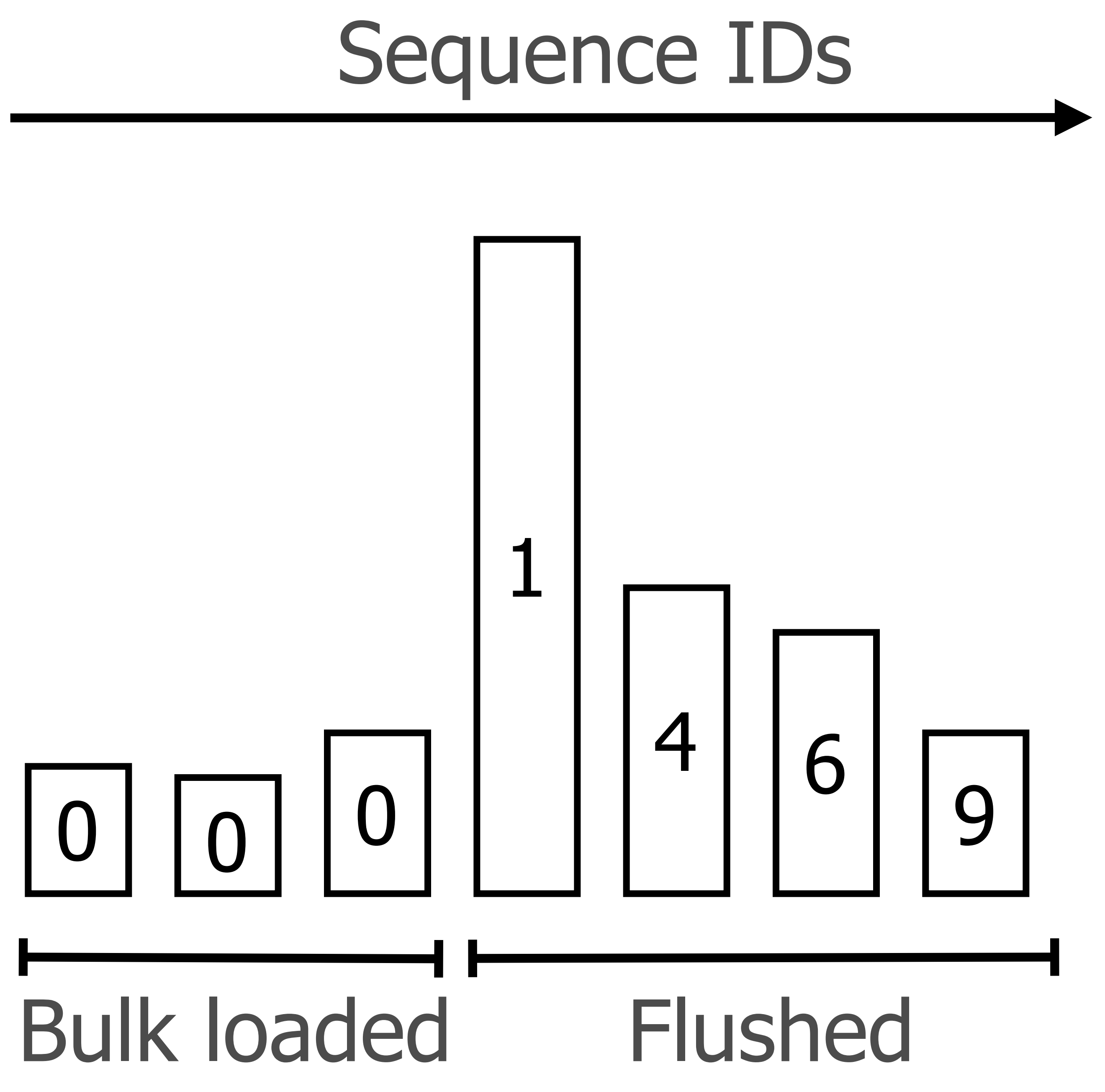
Gotchas: HBASE-8521 (fixed 0.94.13+)

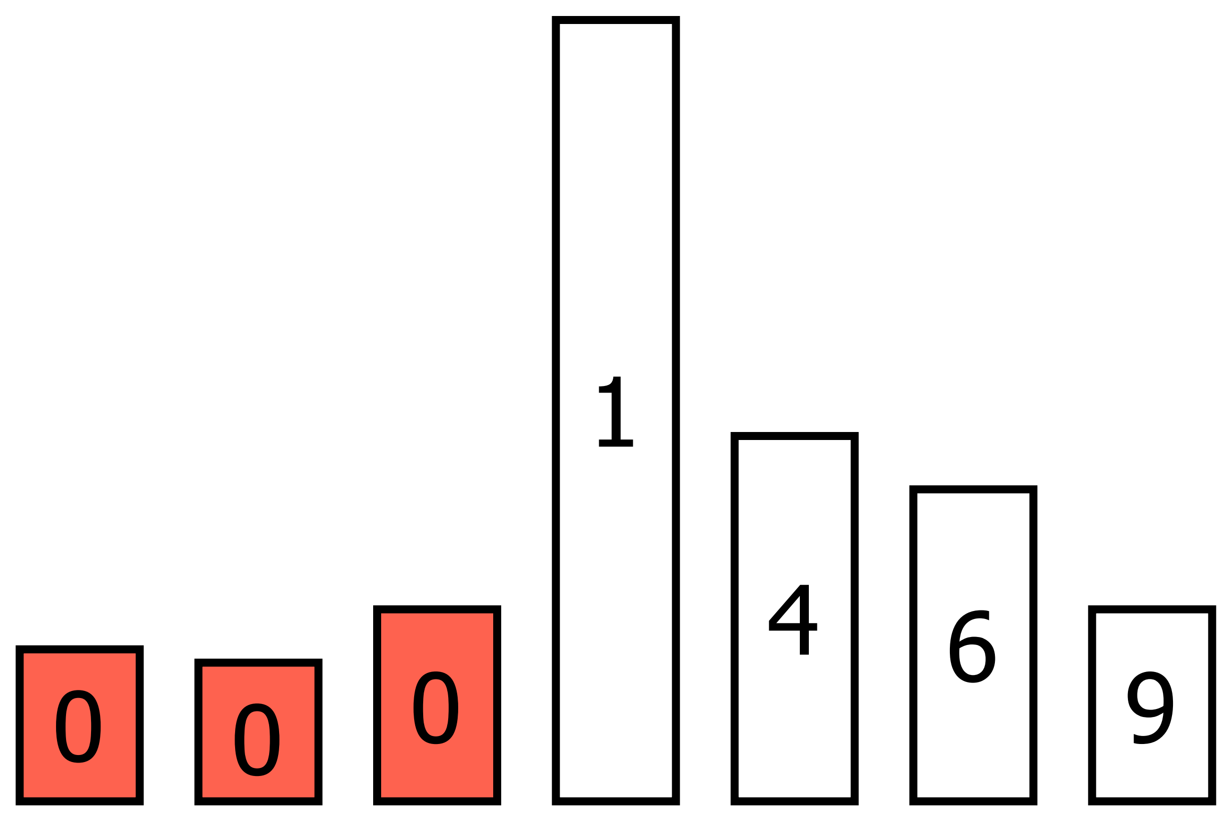




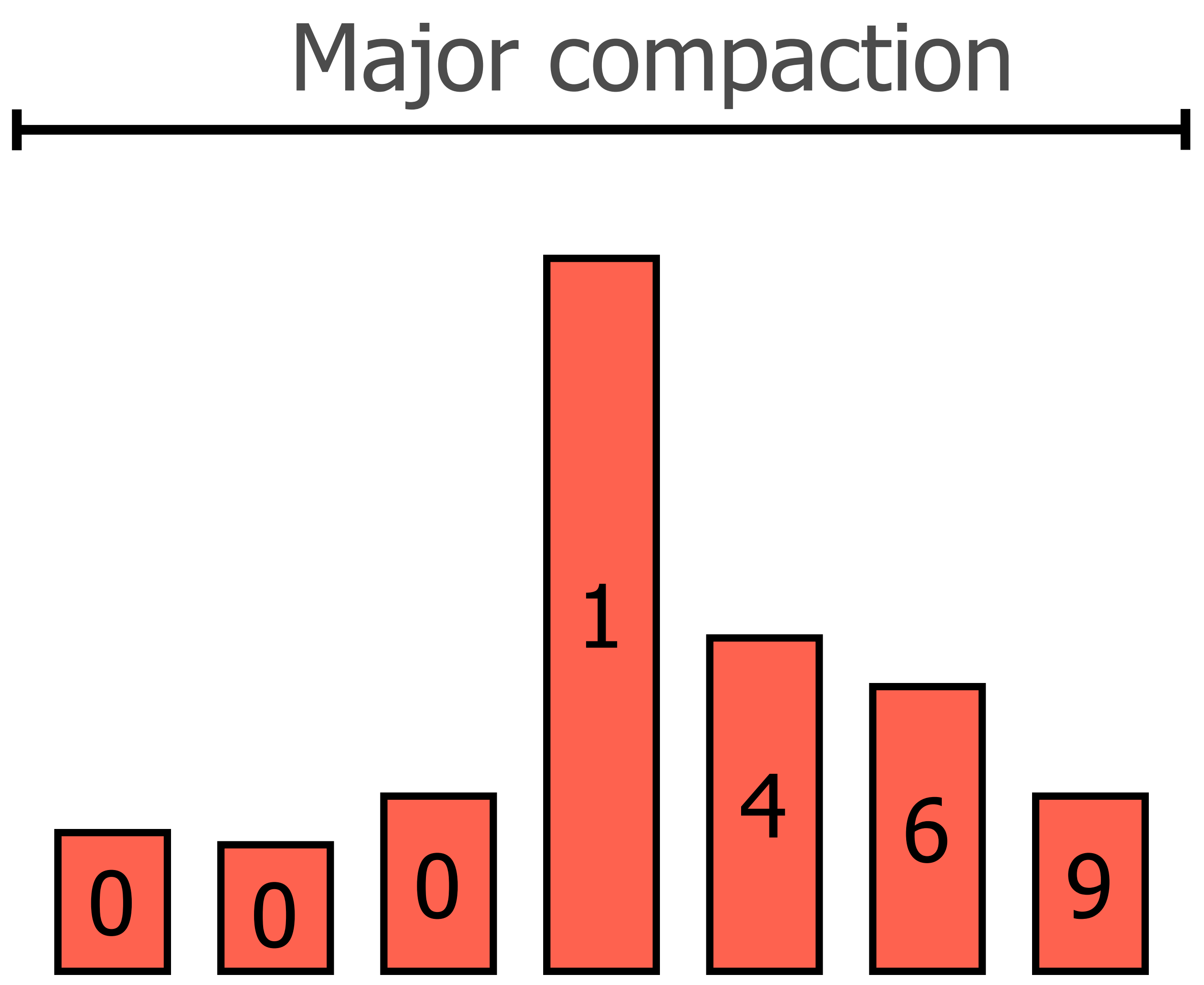
3 HFiles fit the compaction selection criteria...

Gotchas: HBASE-8521 (fixed 0.94.13+)



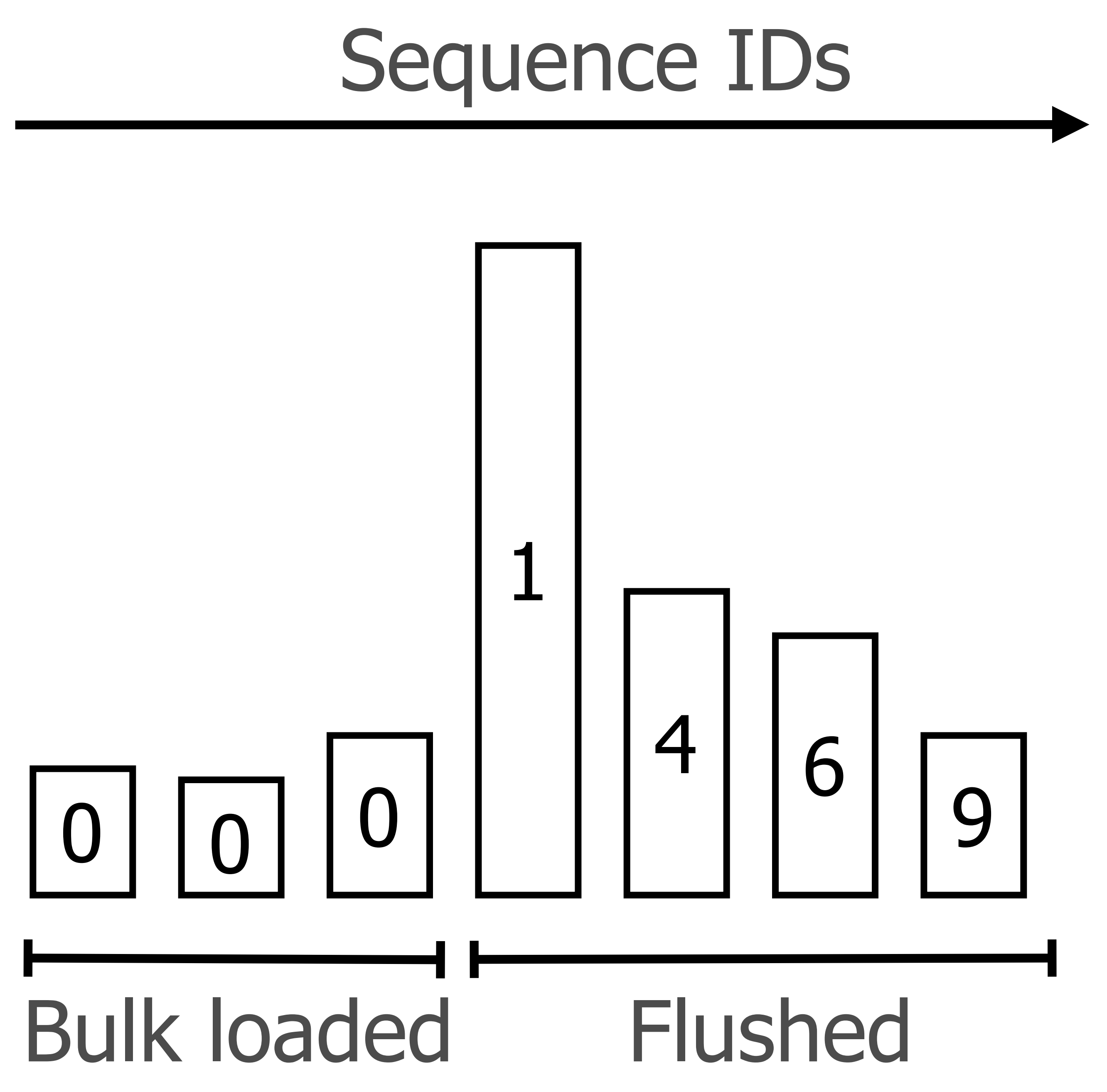


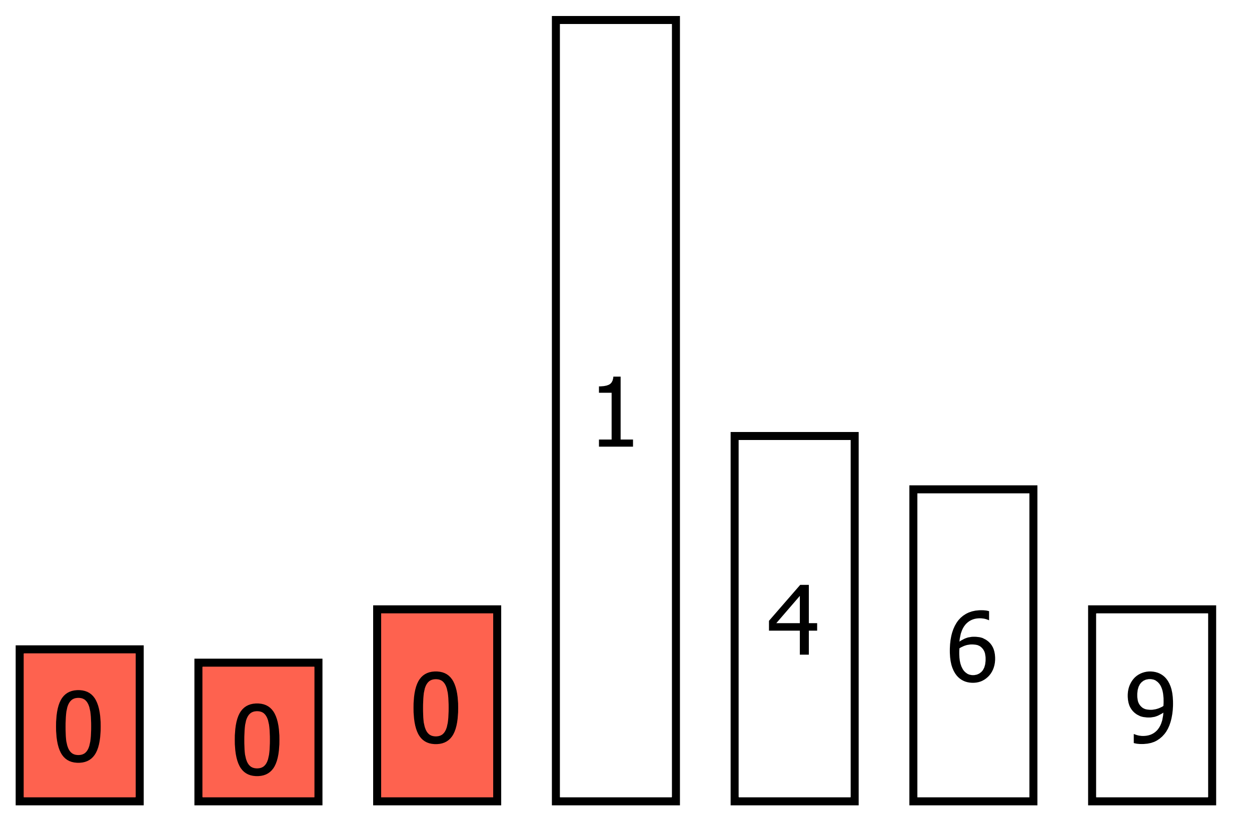
3 HFiles fit the compaction selection criteria...



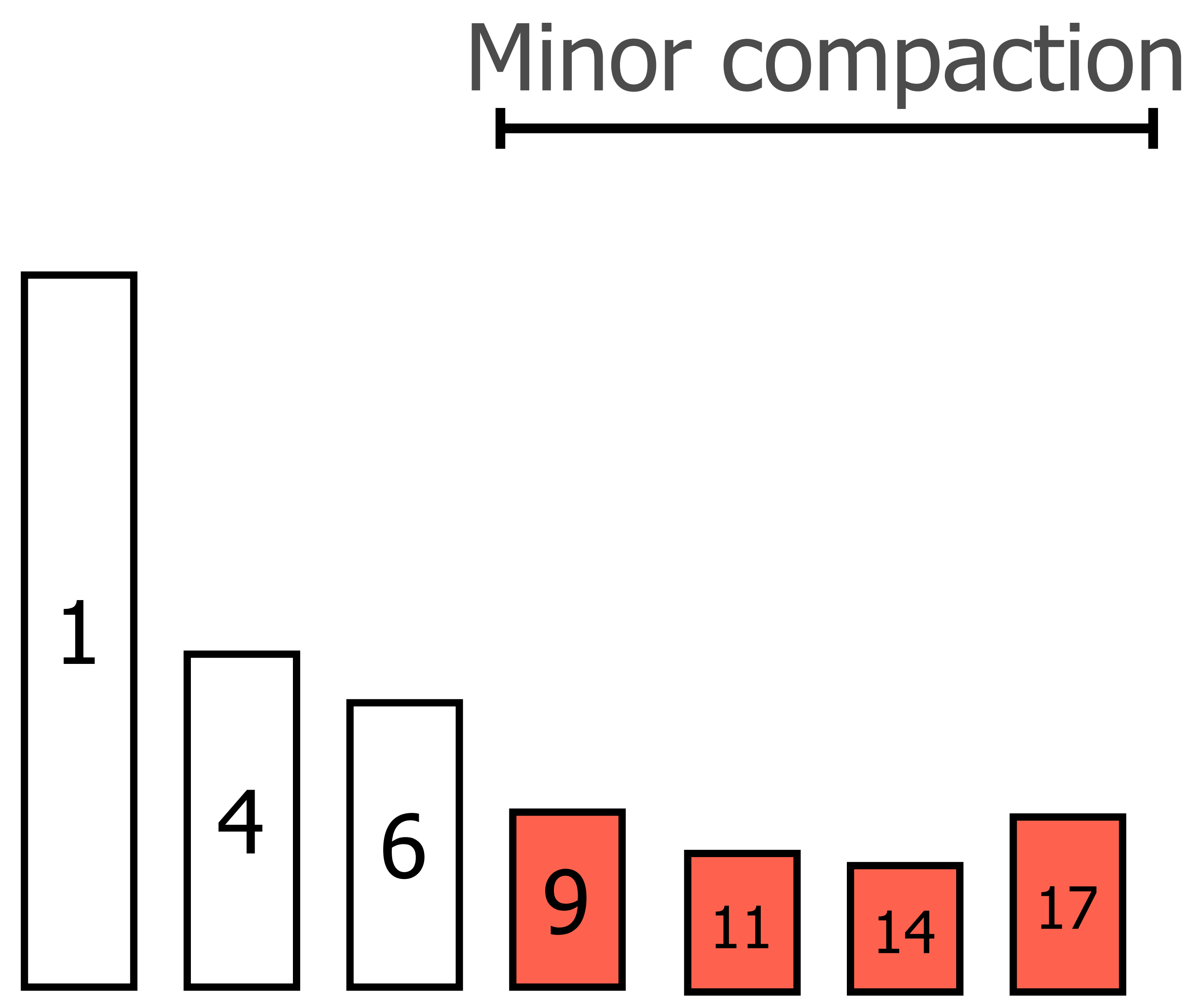
It means we also select all the files that come after!

Gotchas: HBASE-8521 (fixed 0.94.13+)



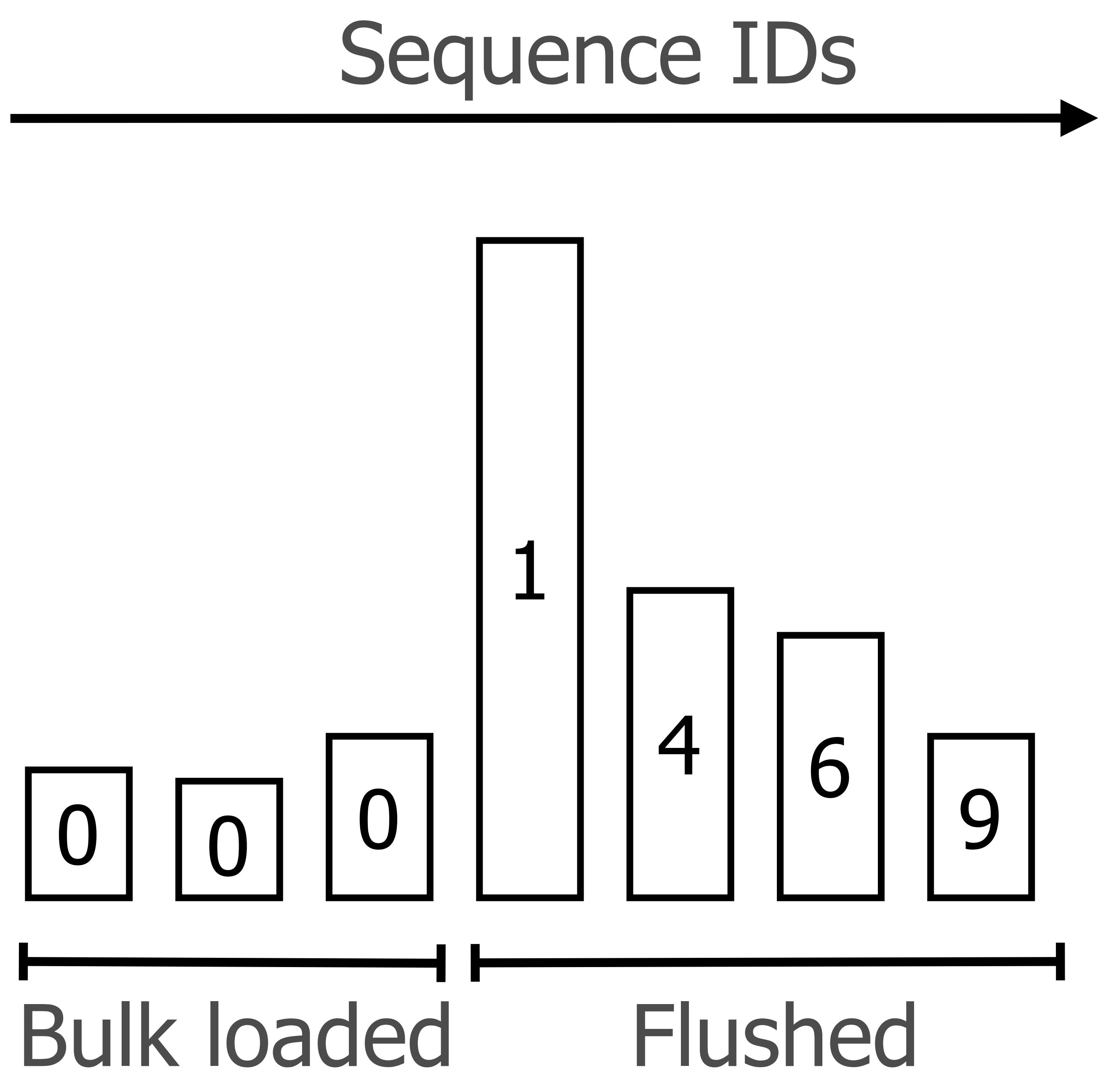


3 HFiles fit the compaction selection criteria...

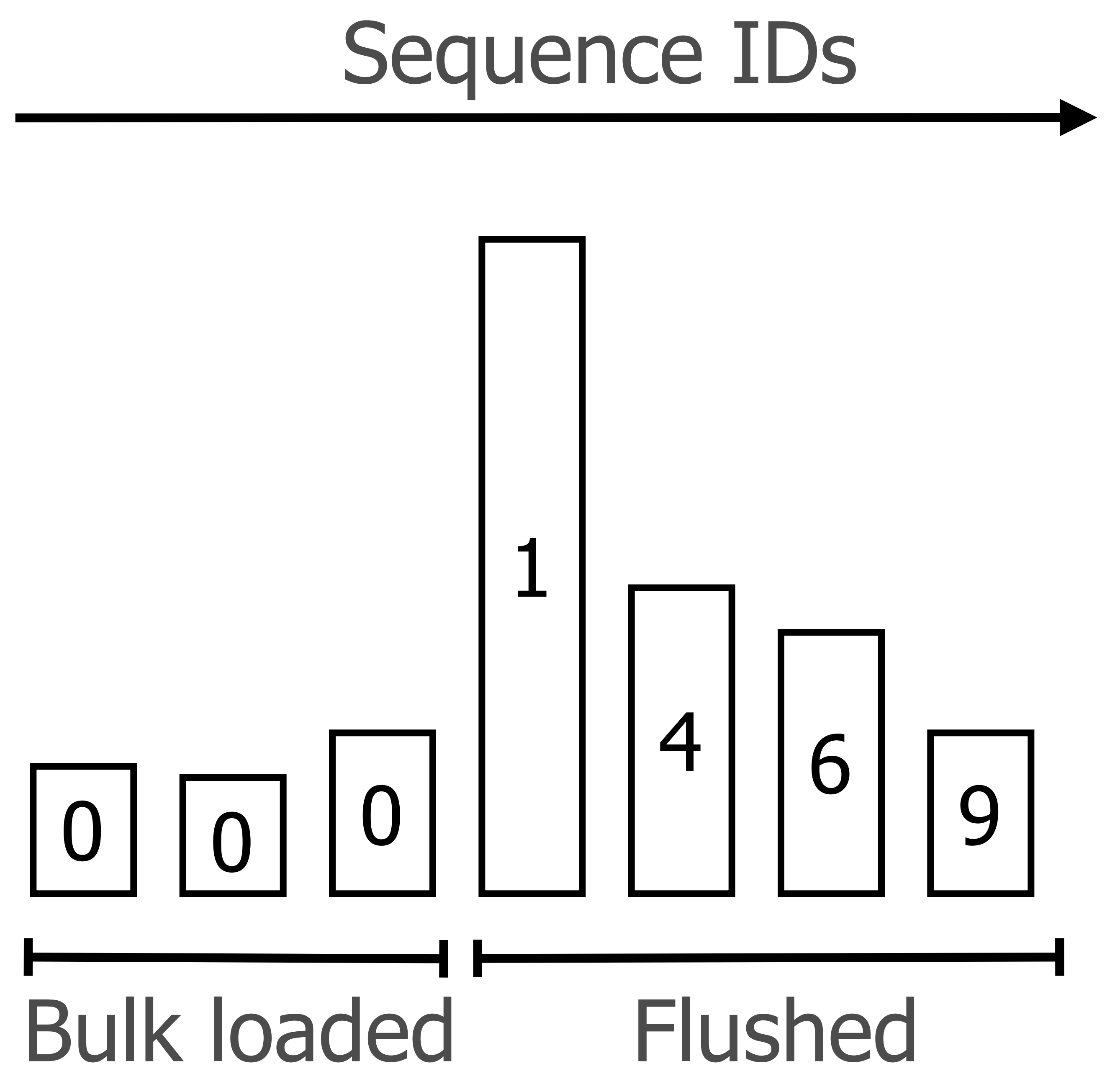


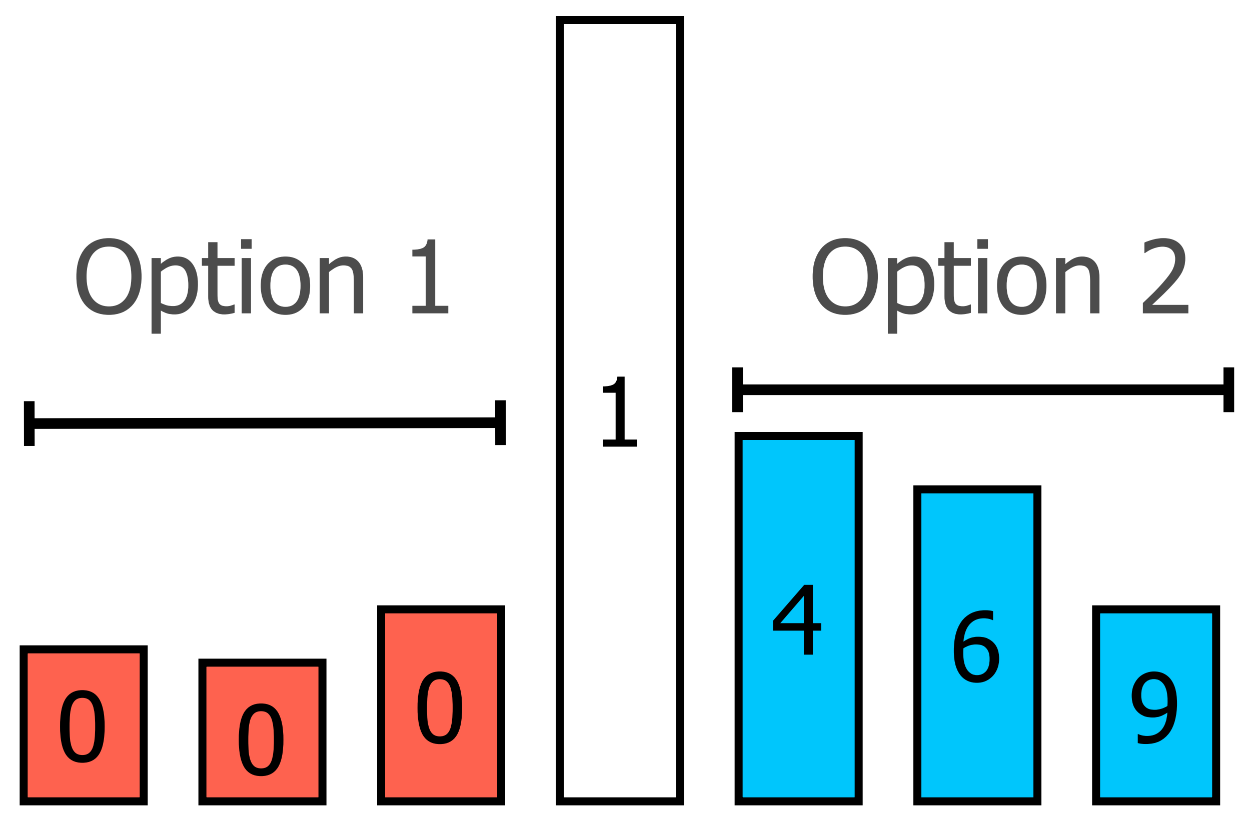
Solution: assign sequence IDs to the bulk loaded files. Optional in 0.94, on by default in 0.96+.

Gotchas: HBASE-8283 (fixed 0.94.9+)



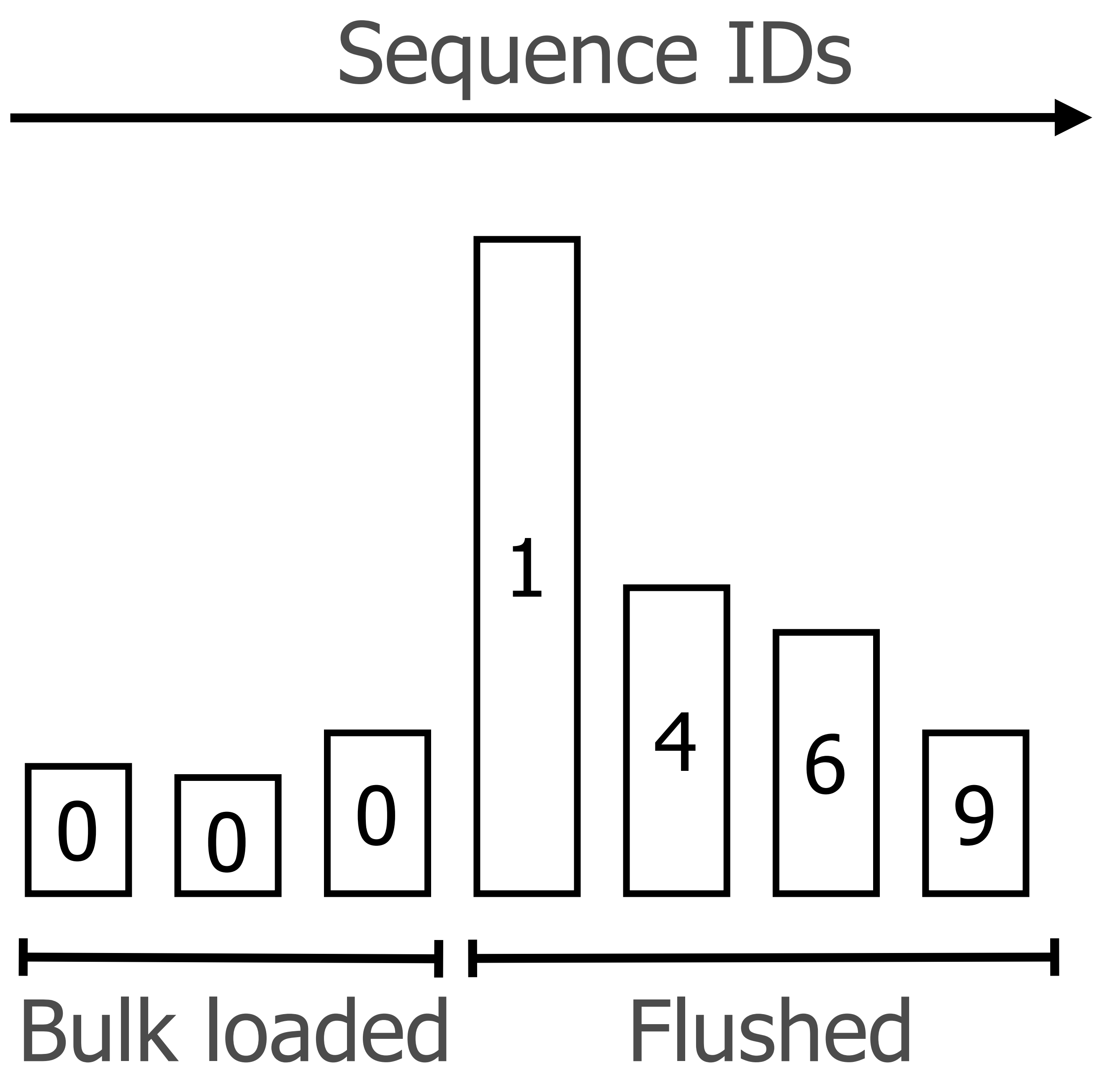
Gotchas: HBASE-8283 (fixed 0.94.9+)

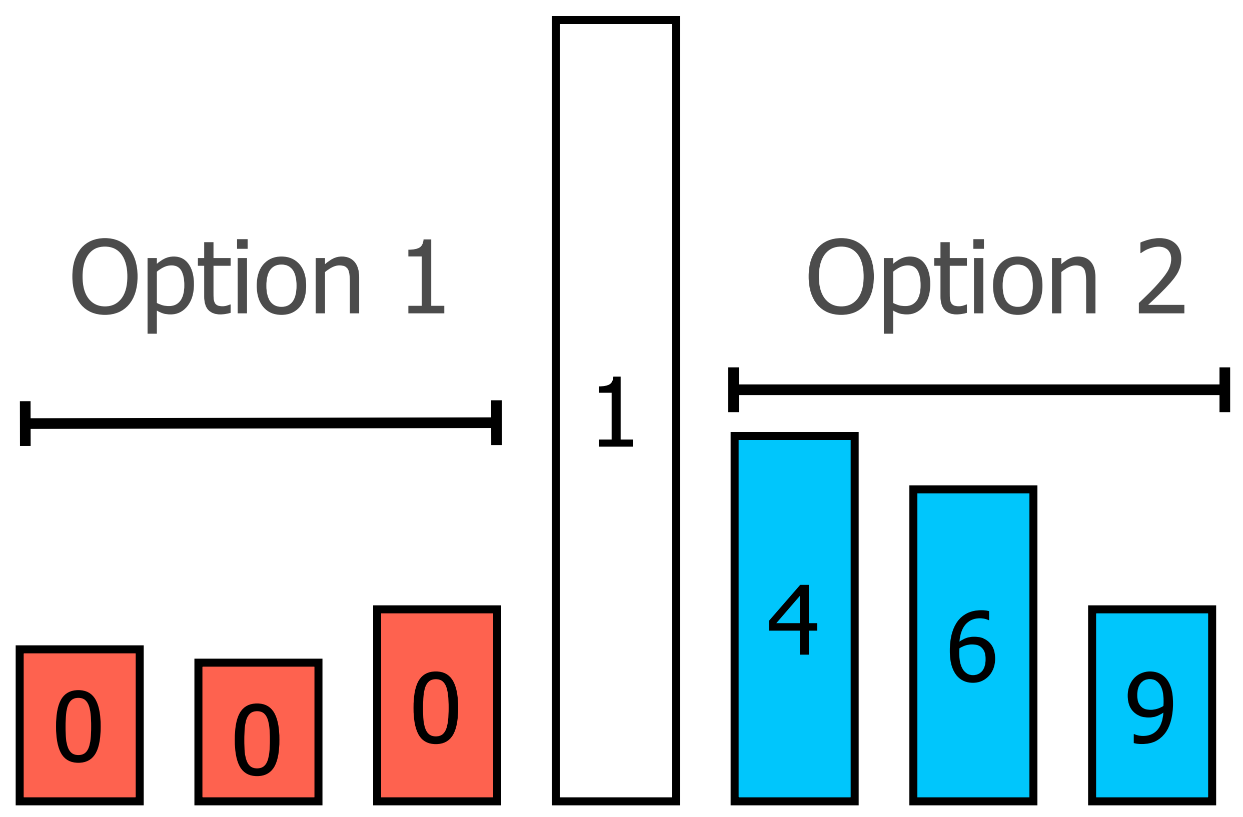




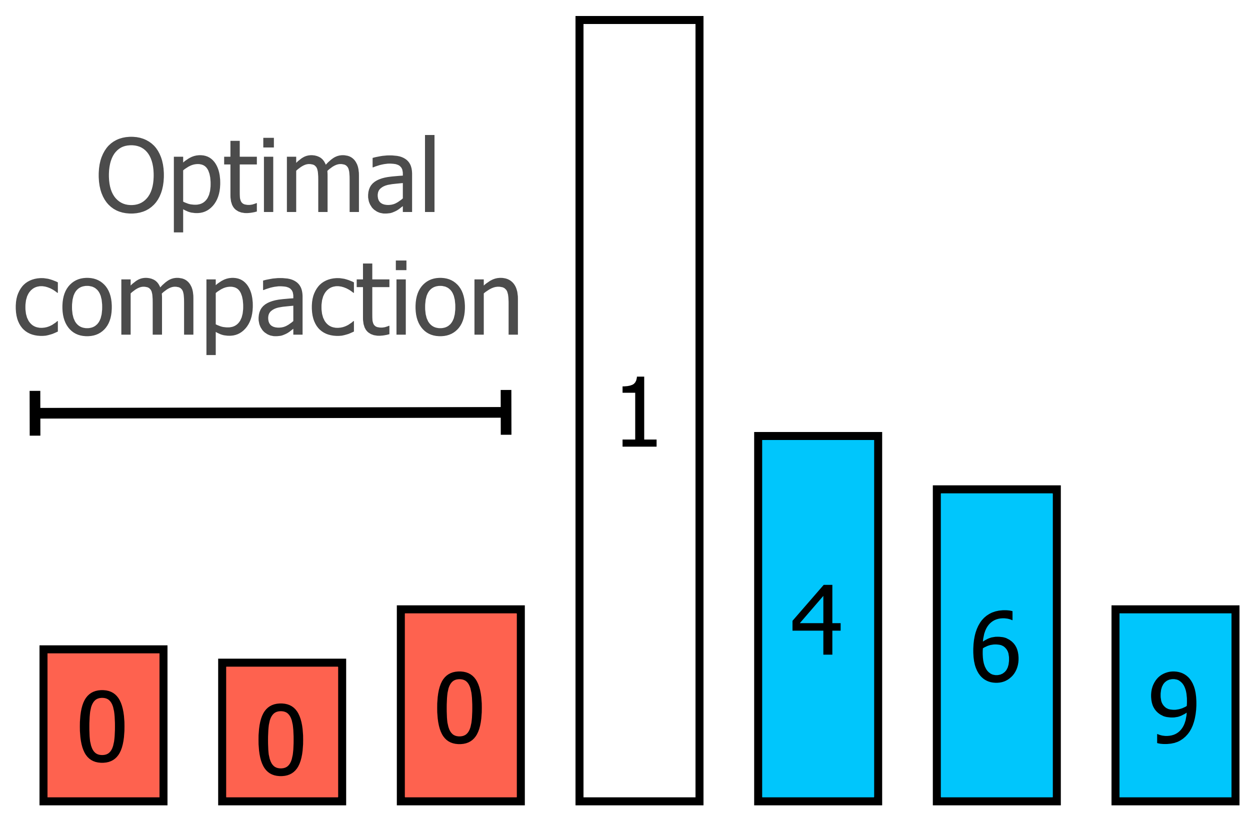
New selection algorithm considers multiple alternatives and doesn't work in only one direction

Gotchas: HBASE-8283 (fixed 0.94.9+)



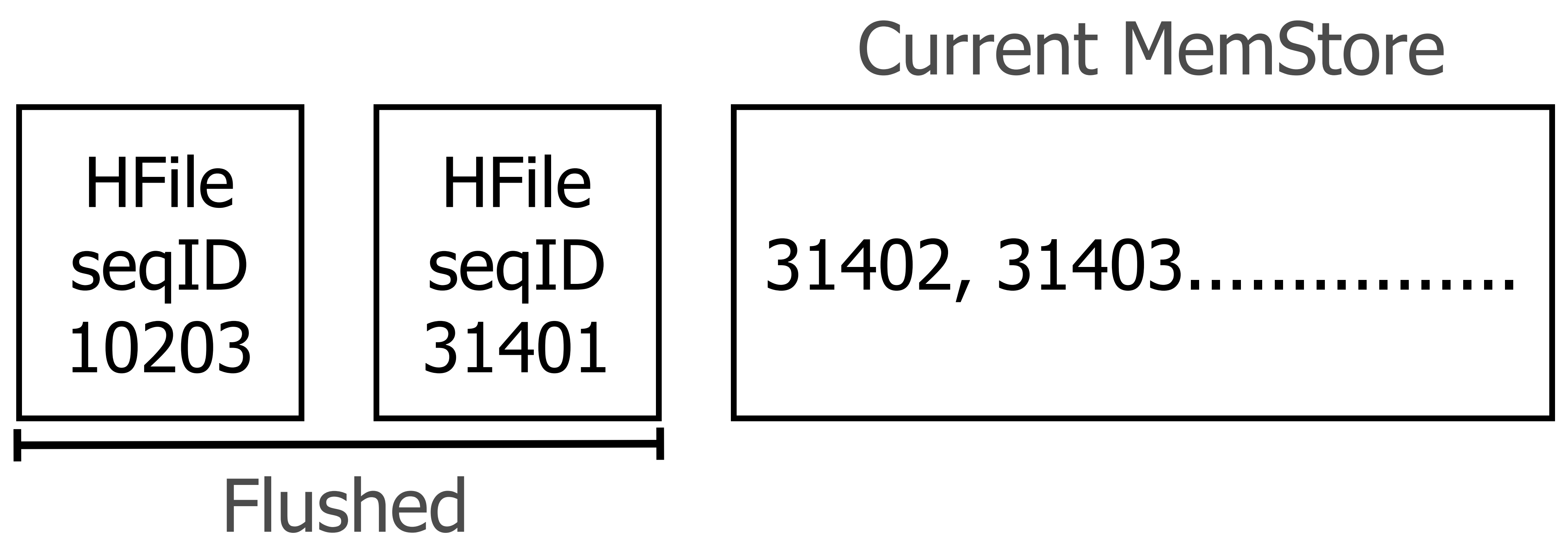


New selection algorithm considers multiple alternatives and doesn't work in only one direction

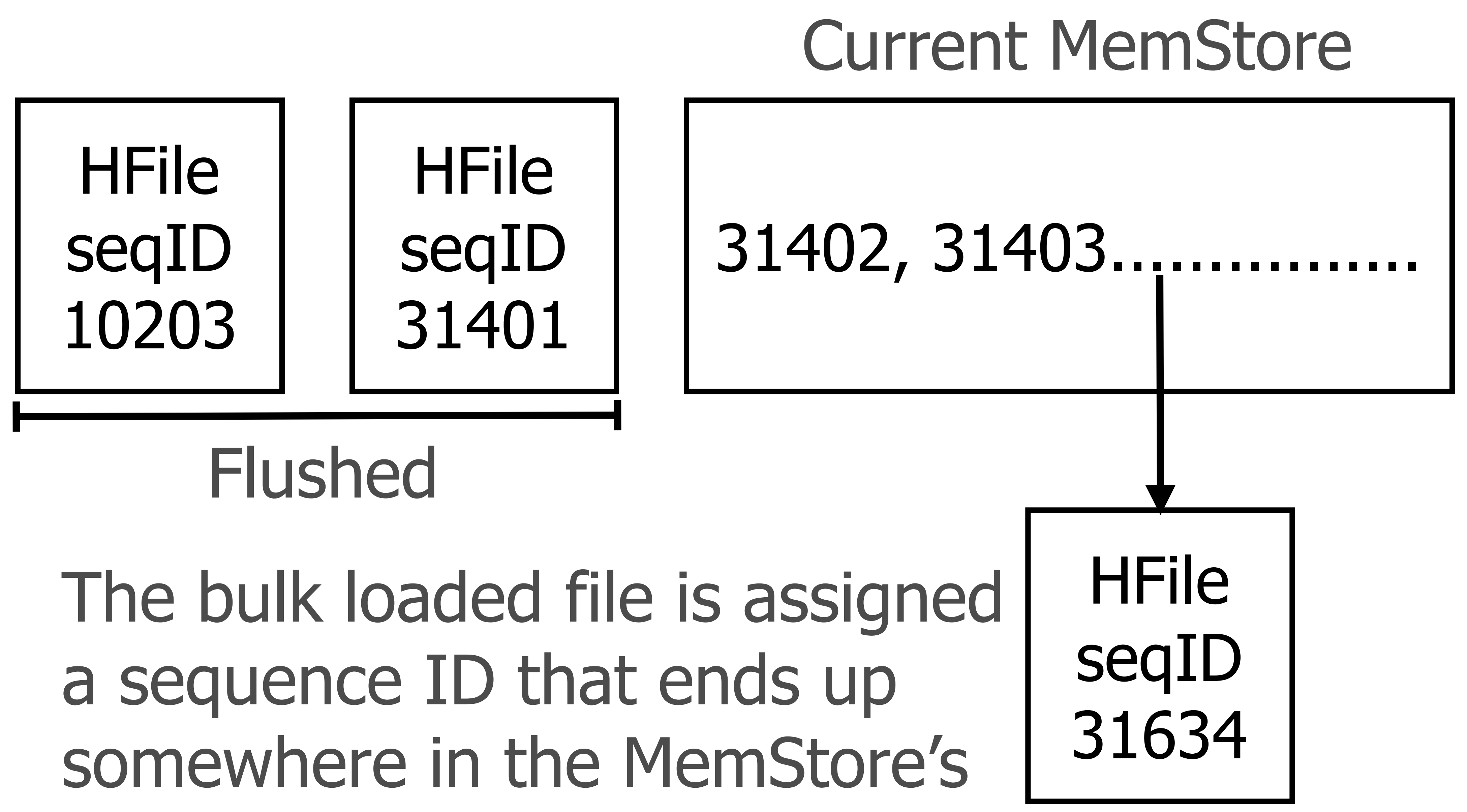


The selection that lowers the amount of seeks while compacting the least amount of data is chosen

HBASE-10958 AKA Blindspot

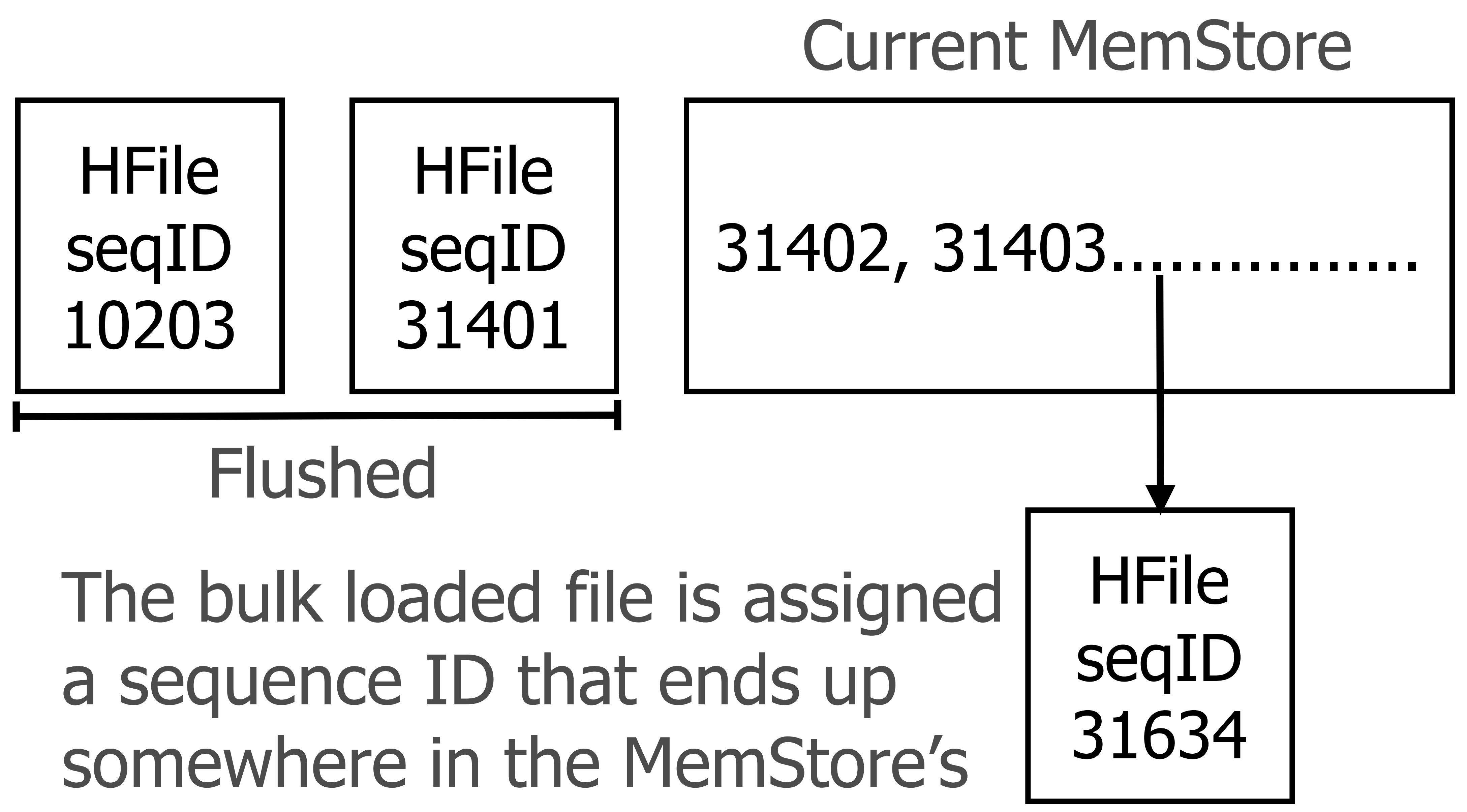


HBASE-10958 AKA Blindspot



The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's

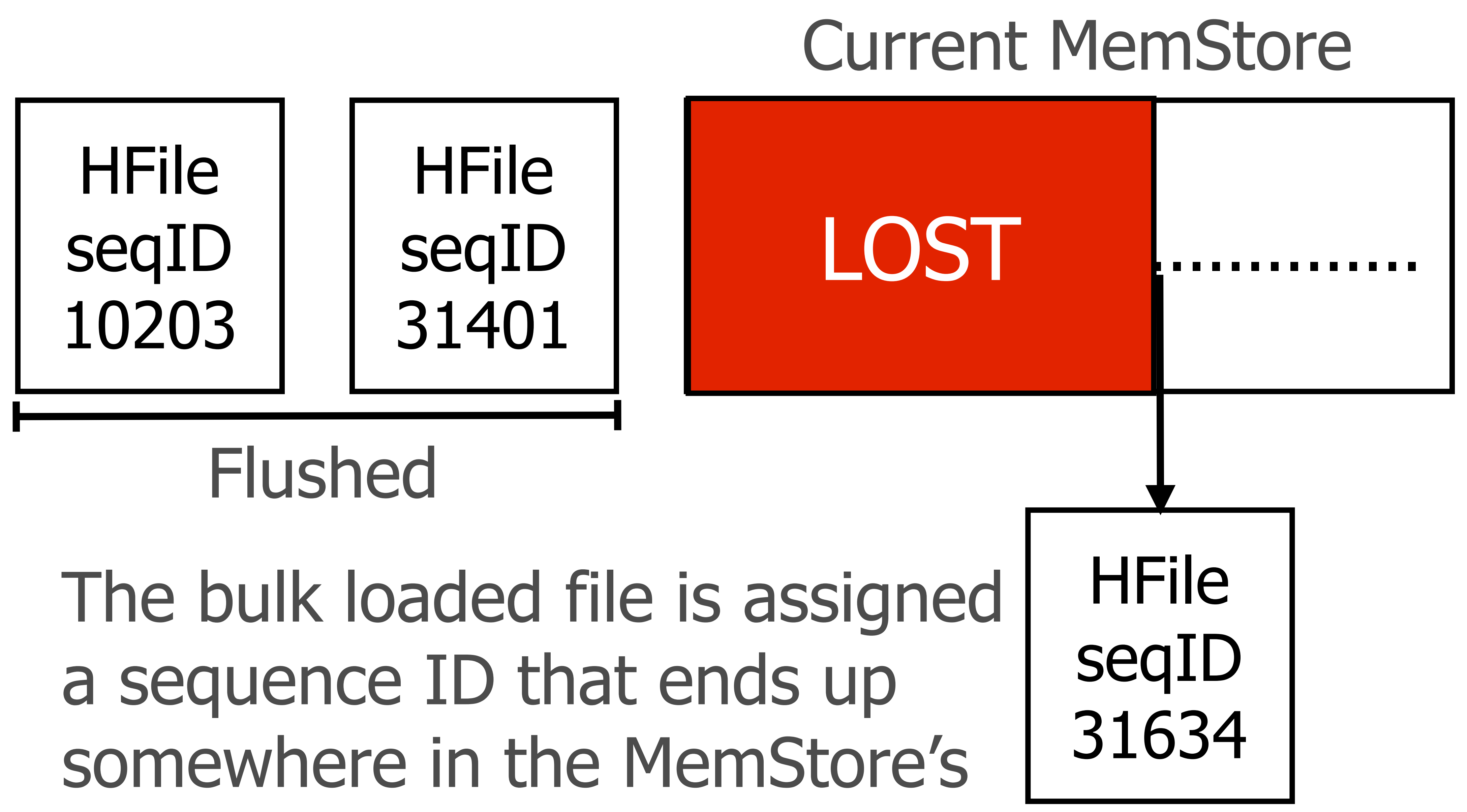
HBASE-10958 AKA Blindspot



The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's

When loading edits from a failed RS's log, replay only the edits coming after the HFile's highest sequence ID.

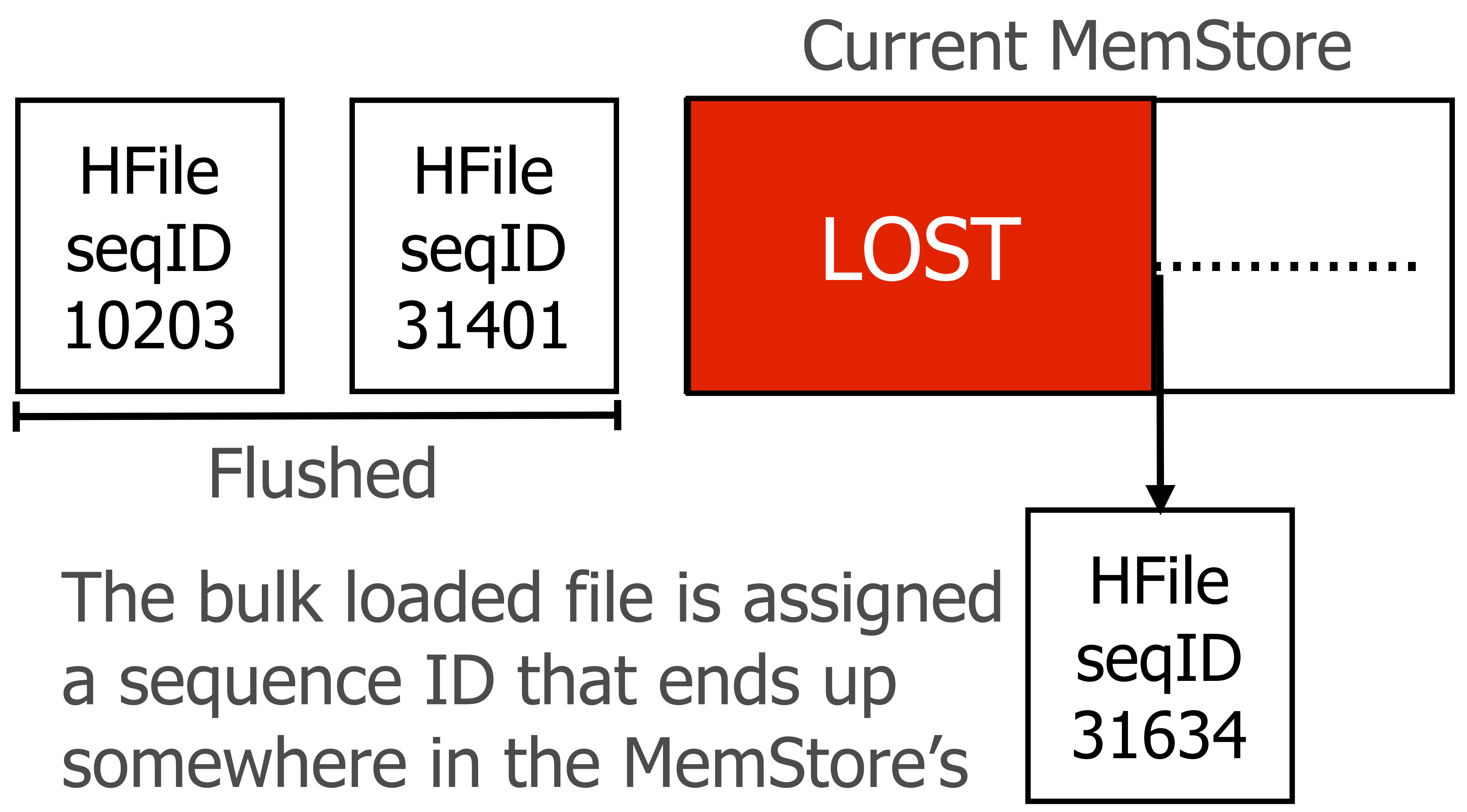
HBASE-10958 AKA Blindspot



The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's

When loading edits from a failed RS's log, replay only the edits coming after the HFile's highest sequence ID.

HBASE-10958 AKA Blindspot

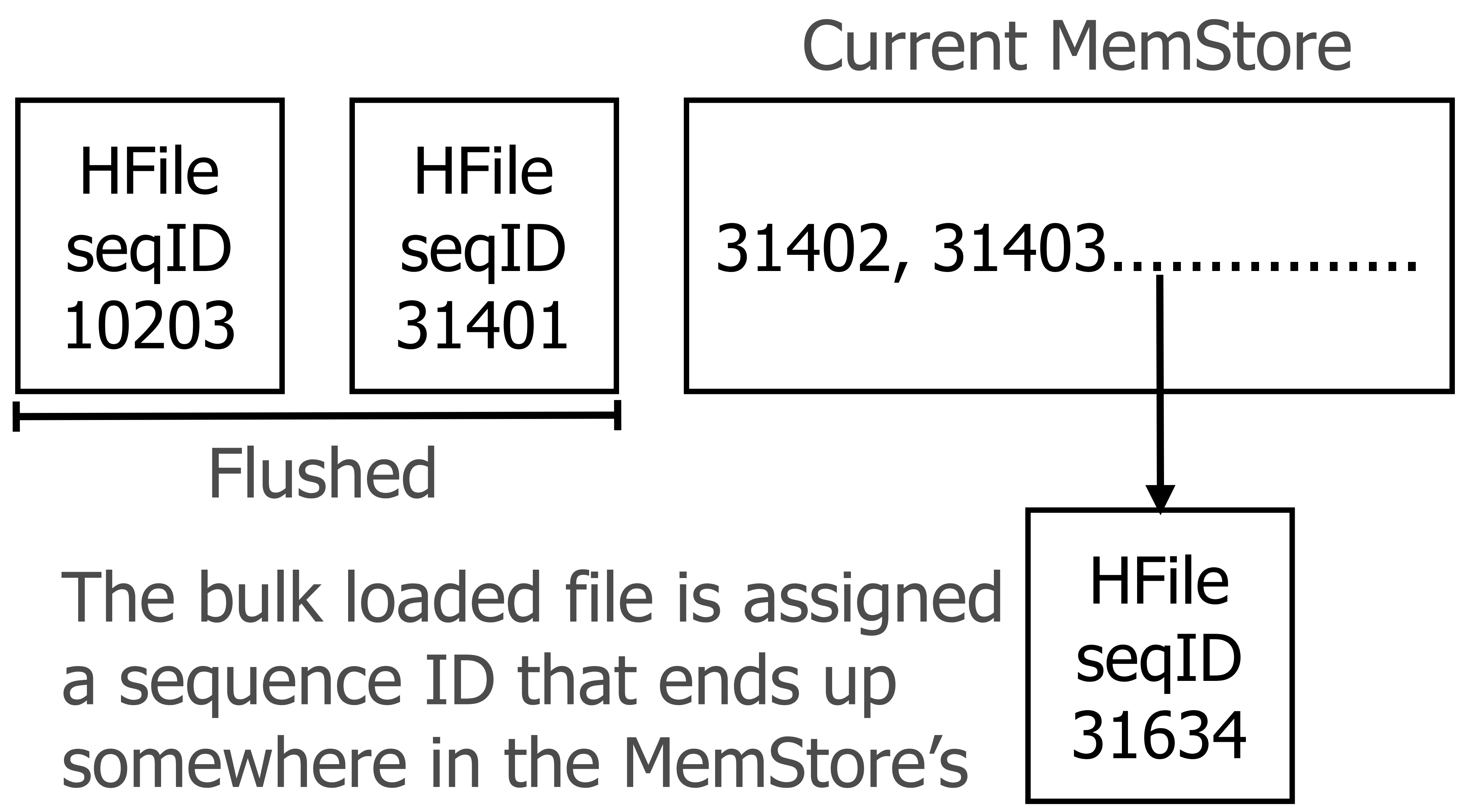


The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's

When loading edits from a failed RS's log, replay only the edits coming after the HFile's highest sequence ID.

Thankfully, we can recognize when a file is a bulk loaded one...

HBASE-10958 AKA Blindspot

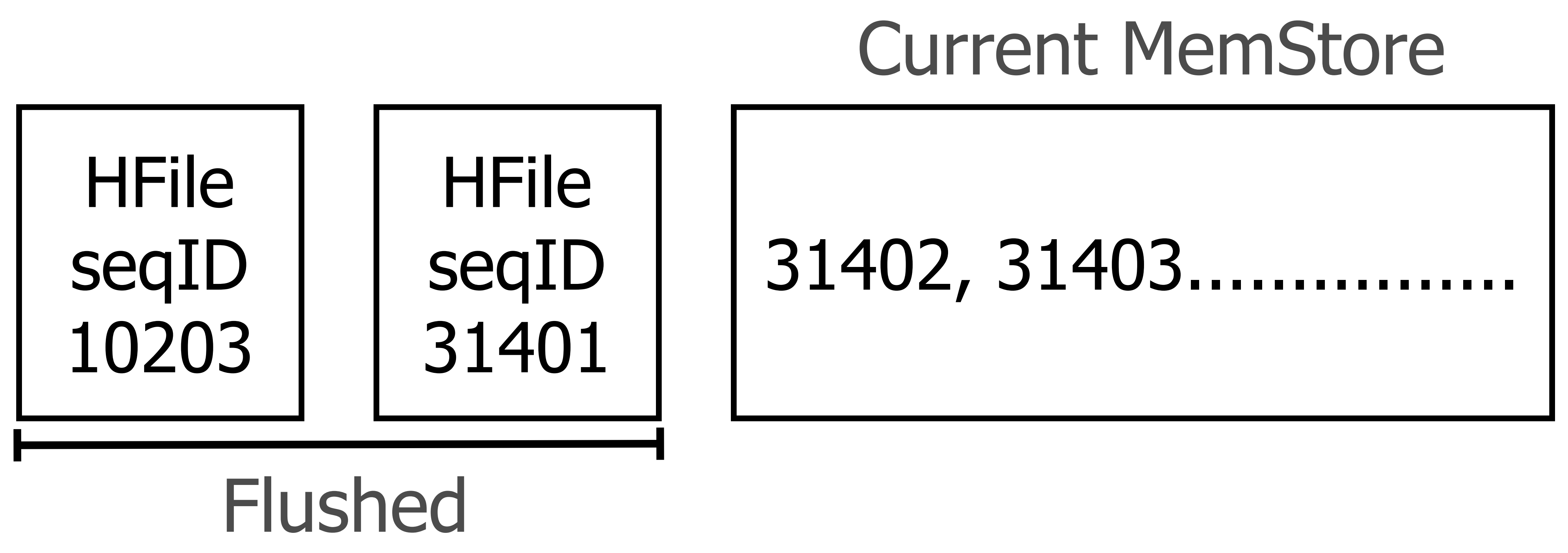


The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's

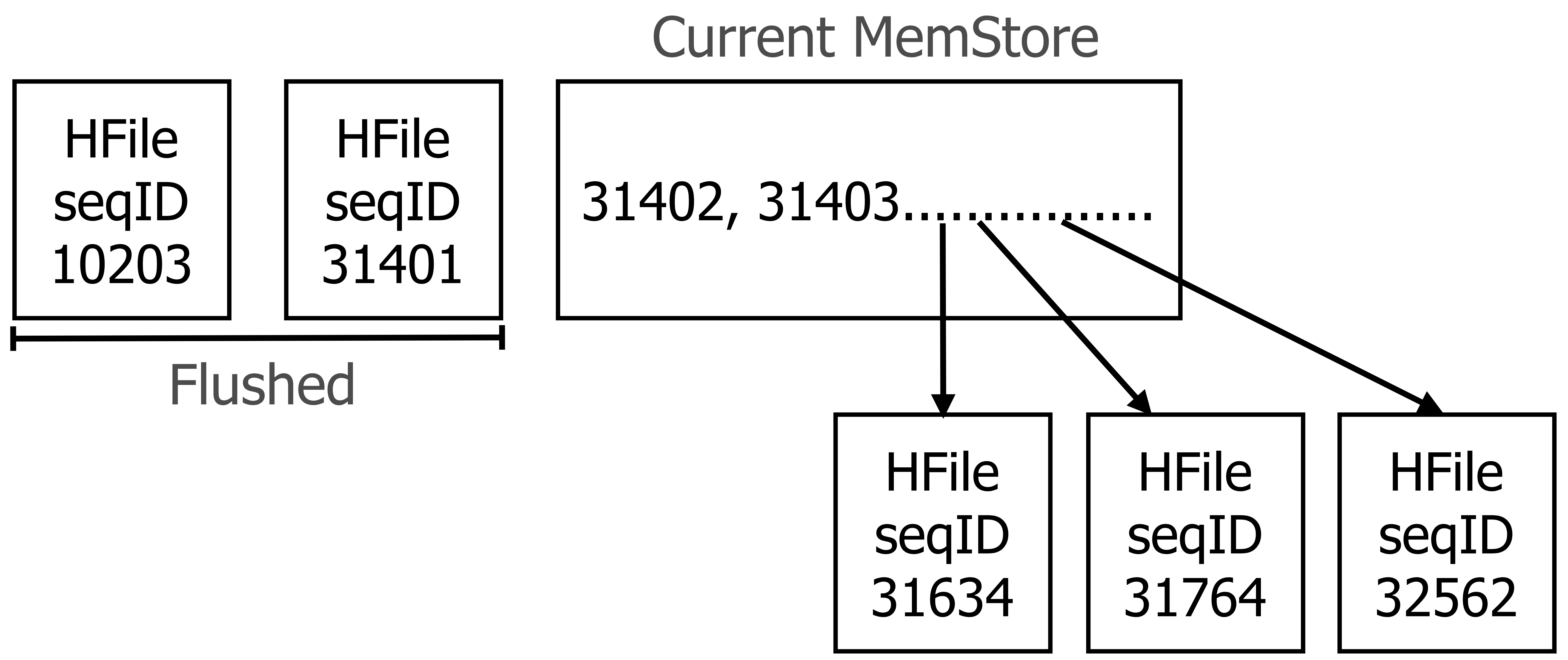
When loading edits from a failed RS's log, replay only the edits coming after the HFile's highest sequence ID.

Thankfully, we can recognize when a file is a bulk loaded one...

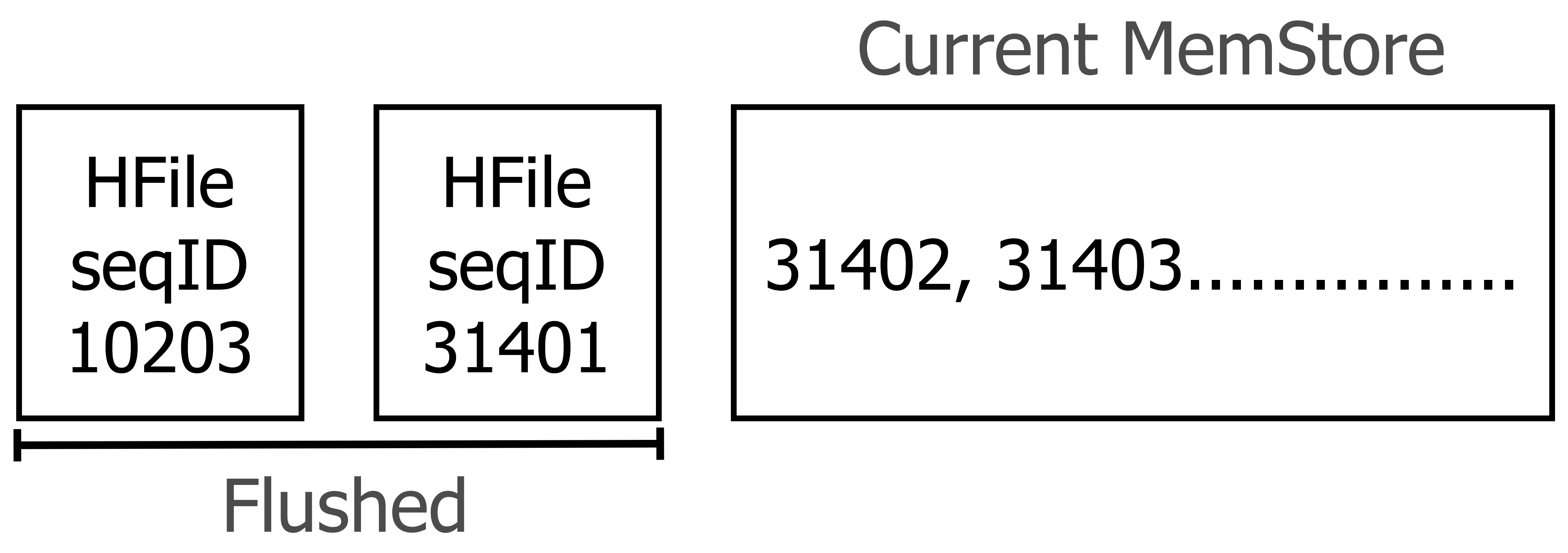
HBASE-10958 AKA Blindspot



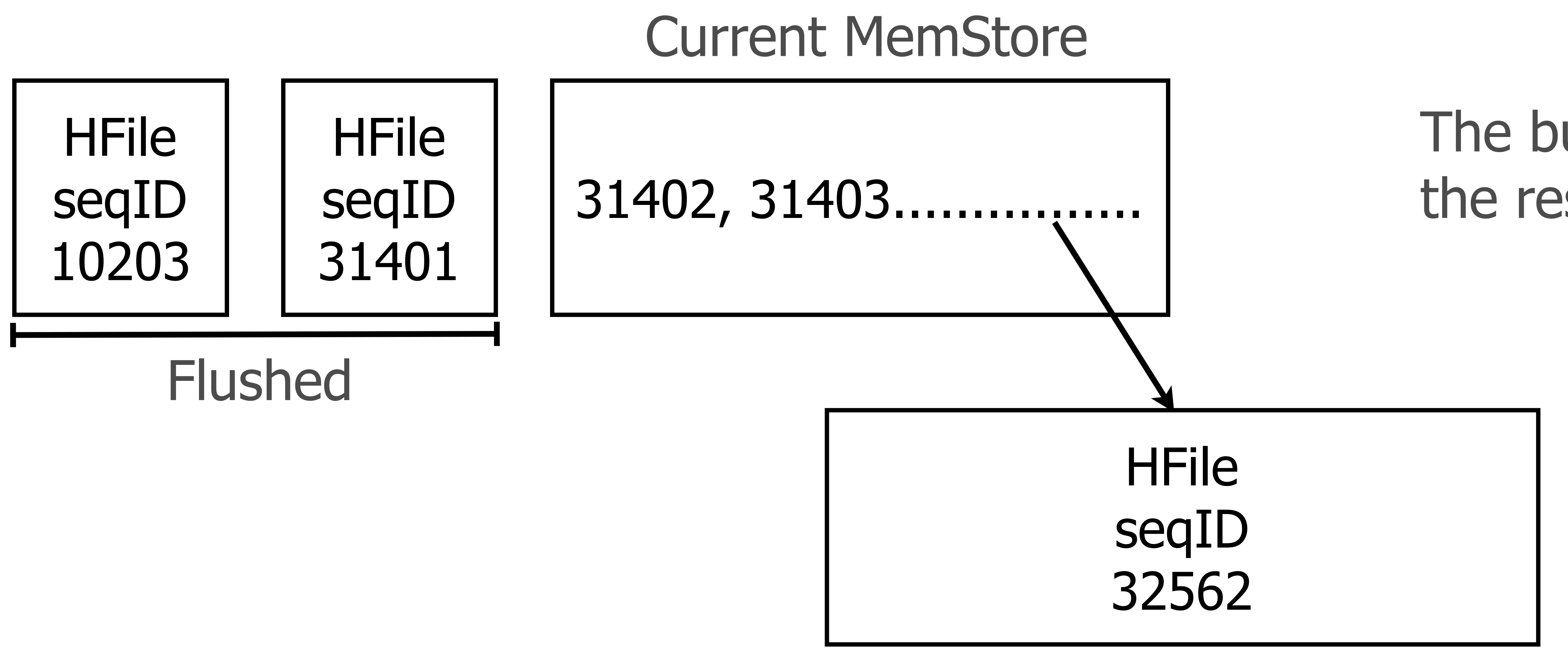
HBASE-10958 AKA Blindspot



HBASE-10958 AKA Blindspot

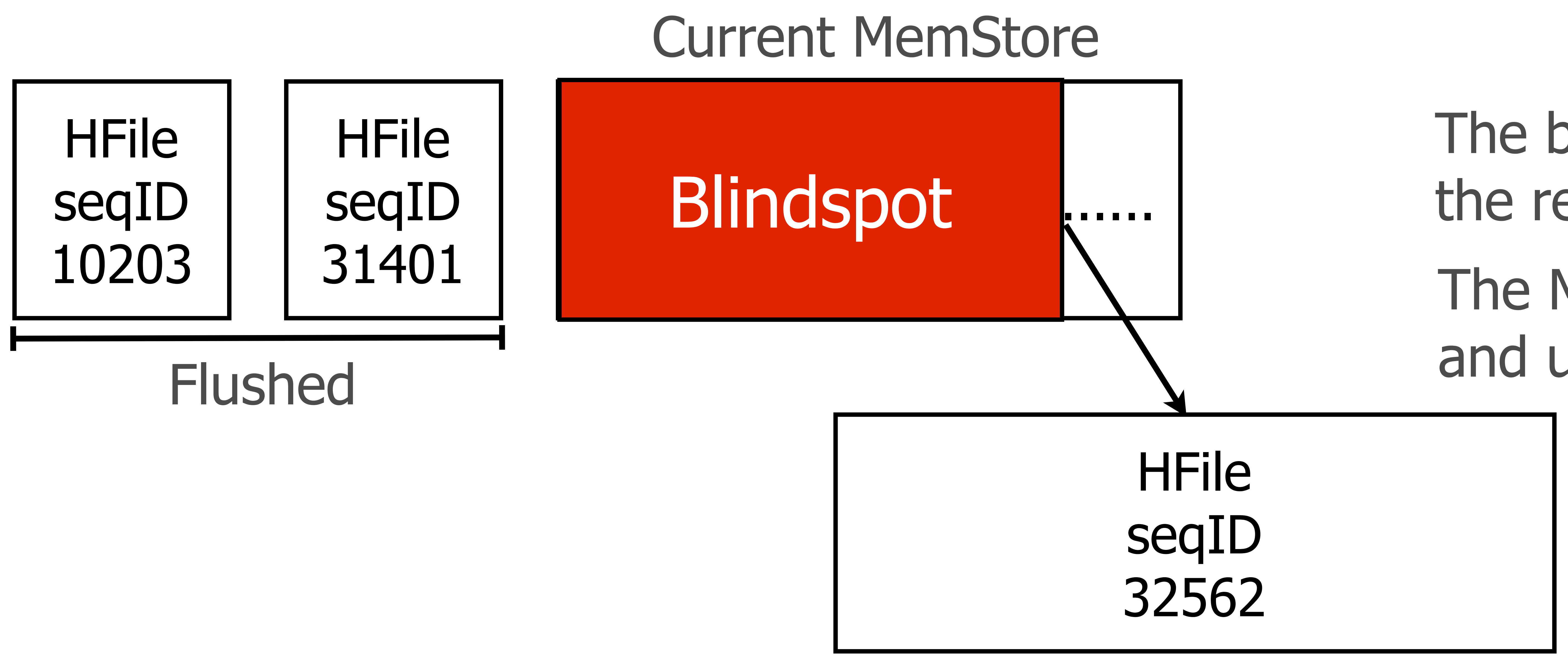


HBASE-10958 AKA Blindspot



The bulk loaded status is lost through compaction, the resulting HFile looks like any other!

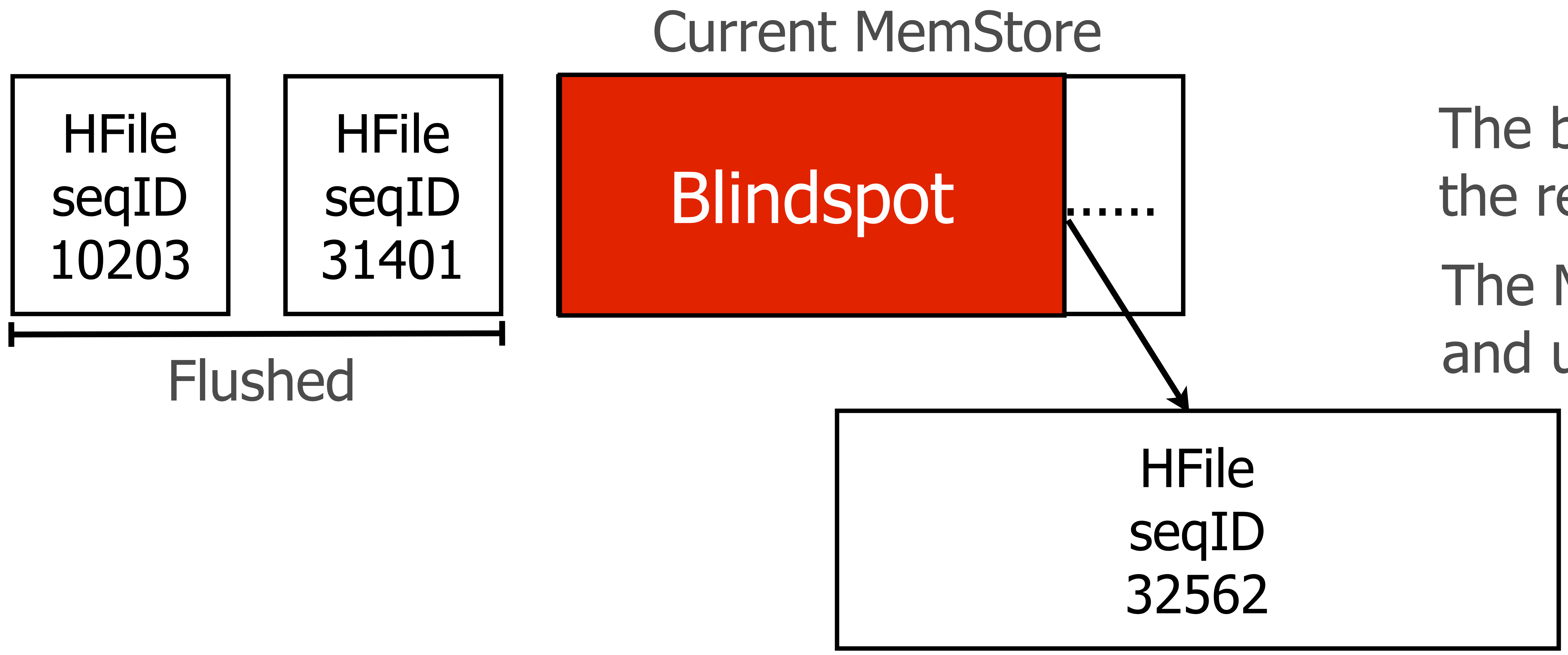
HBASE-10958 AKA Blindspot



The bulk loaded status is lost through compaction, the resulting HFile looks like any other!

The MemStore's data, starting from the beginning and up to 32562, will be lost even if it was logged.

HBASE-10958 AKA Blindspot



The bulk loaded status is lost through compaction, the resulting HFile looks like any other!

The MemStore's data, starting from the beginning and up to 32562, will be lost even if it was logged.

The proposed solution is to force flush when bulk loading with sequence IDs, since the way it currently works goes against log replay's assumptions.

In conclusion

- Loading via the “normal” APIs can be slow and/or disrupt the cluster.

In conclusion

- Loading via the “normal” APIs can be slow and/or disrupt the cluster.
- Bulk loading can create files that HBase can directly use.

In conclusion

- Loading via the “normal” APIs can be slow and/or disrupt the cluster.
- Bulk loading can create files that HBase can directly use.
- Useful for your original data import or incremental ones.

In conclusion

- Loading via the “normal” APIs can be slow and/or disrupt the cluster.
- Bulk loading can create files that HBase can directly use.
- Useful for your original data import or incremental ones.
- Recommended to use HBase versions released during the past year.

@jdcryans