Working with the Cassandra Write Path

Apache Cassandra: **Core Concepts, Skills, and Tools**

Leo Schuman, Joe Chu

Oct 20, 2014

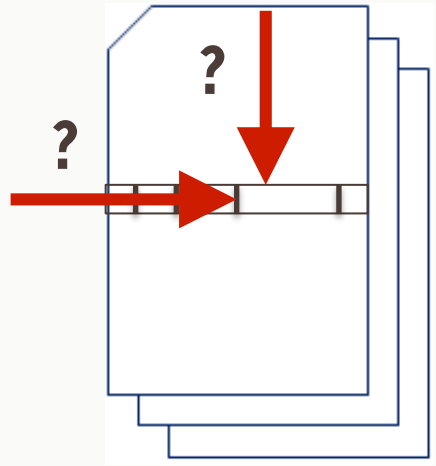
Learning Objectives

- **Understand how data is written to the storage engine**
- Understand the data directories

How does Cassandra write so fast?

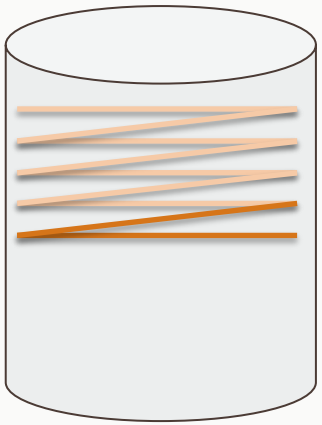
- Cassandra is a log-structured storage engine
  - Data is sequentially appended, not placed in pre-set locations

RDBMS



Seeks and writes values to various pre-set locations

CASSANDRA



Continuously appends to a log

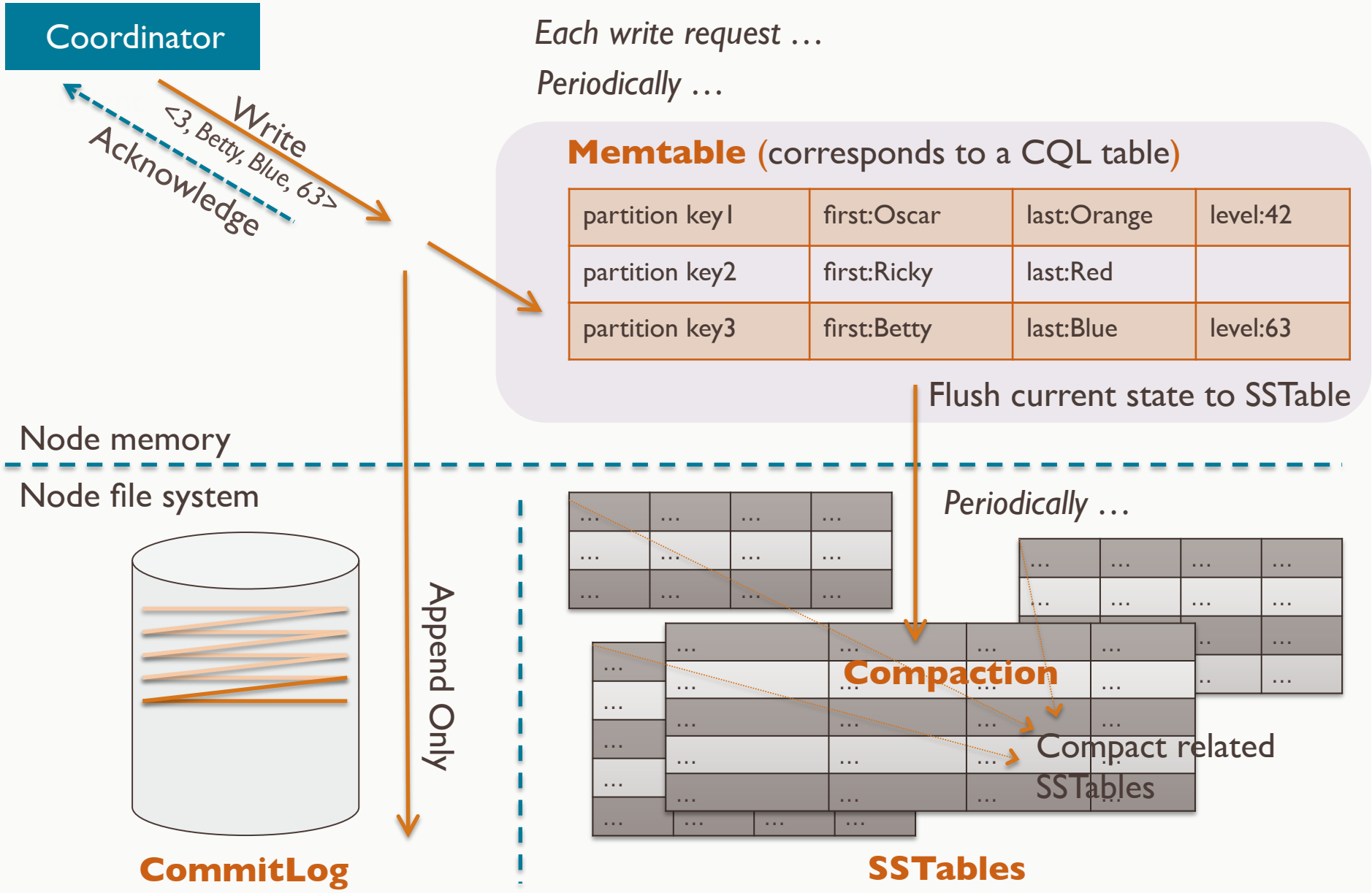
What are the key components of the write path?

DATASTAX

- Each node implements four key components to handle its writes
  - Memtables – in-memory tables corresponding to CQL tables, with indexes
  - CommitLog – append-only log, replayed to restore downed node's *Memtables*
  - SSTables – *Memtable* snapshots periodically flushed to disk, clearing heap
  - Compaction – periodic process to merge and streamline *SSTables*

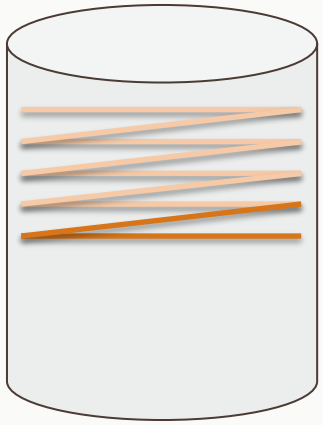
- When any node receive any write request
  1. The record appends to the *CommitLog*, and
  2. The record appends to the *Memtable* for this record's target CQL table
  3. Periodically, *Memtables* flush to *SSTables*, clearing JVM heap and *CommitLog*
  4. Periodically, *Compaction* runs to merge and streamline *SSTables*

How does the write path flow on a node?



What is the CommitLog and how is it configured?

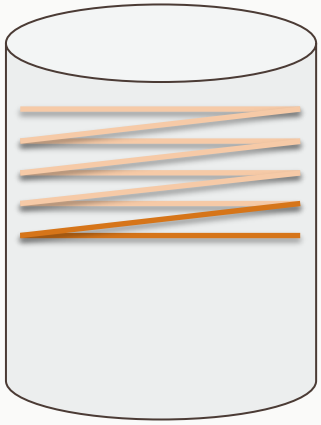
- An append-only log used to automatically rebuild *Memtables* on restart of a downed node, configured in *conf/cassandra.yaml*
- *Memtables* flush to disk when *CommitLog* size reaches total allowed space
  - commitlog_total_space_in_mb – size at which oldest *Memtable* log segment will be flushed to disk (default: *1024* for 64bit JVMs)
  - commitlog_segment_size_in_mb – max size of individual log segments (default: *32*)
- Entries are marked as flushed, as corresponding *Memtable* entries flush to disk as an *SSTable*
  - Flushed CommitLog segments are periodically recycled
- Best practice is to locate *CommitLog* on its own disk to minimize write head movement, or on SSD
  - commitlog_directory – default is */var/lib/cassandra/commitlog* (package install) or *install_location/data/commitlog* (binary tarball)



CommitLog

What is the CommitLog and how is it configured?

- Entries accrue in memory, and are synced to disk in either a *batch* or *periodic* manner
  - commitlog_sync – either *periodic* or *batch* (default: periodic)
- batch – writes are not acknowledged until the log syncs to disk
  - commitlog_sync_batch_window_in_ms – how long to wait for more writes before fsync (default: *50*)
- periodic – writes are acknowledged immediately, while sync happens periodically
  - commitlog_sync_period_in_ms – how long to wait between fsync of log to disk (default: *10000*)



CommitLog

What are Memtables and how are they flushed to disk?

Memtable

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63

- *Memtables* are in-memory representations of a CQL table
  - Each node has a *Memtable* for each CQL table in the keyspace
  - Each *Memtable* accrues writes and provides reads for data not yet flushed
  - Updates to *Memtables* mutate the in-memory partition
- When a *Memtable* flushes to disk
  1. Current *Memtable* data is written to a new immutable *SSTable* on disk
  2. JVM heap space is reclaimed from the flushed data
  3. Corresponding *CommitLog* entries are marked as flushed

What are Memtables and how are they flushed to disk?

Memtable

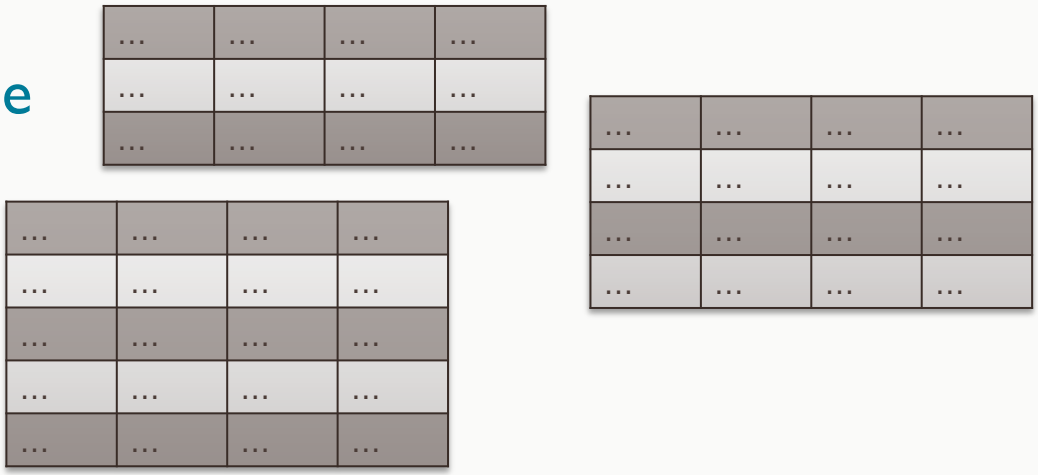
partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63

- A *Memtable* flushes the oldest *CommitLog* segments to a new corresponding *SSTable* on disk when
  - memtable_total_space_in_mb is reached (default: 25% of JVM heap)
  - commitlog_total_space_in_mb is reached
  - `nodetool flush` command is issued
- The *nodetool flush* command force-flushes designated *Memtables*

```
./nodetool flush [keyspace] [table(s)]
```

What is an SSTable and what are its characteristics?

- An *SSTable* ("sorted string table") is
  - an immutable file of sorted partitions
  - written to disk through fast, sequential i/o
  - contains the state of a *Memtable* when flushed
- The current data state of a CQL table is comprised of
  - its corresponding *Memtable* plus
  - all current *SSTables* flushed from that *Memtable*
- *SSTables* are periodically compacted from many to one

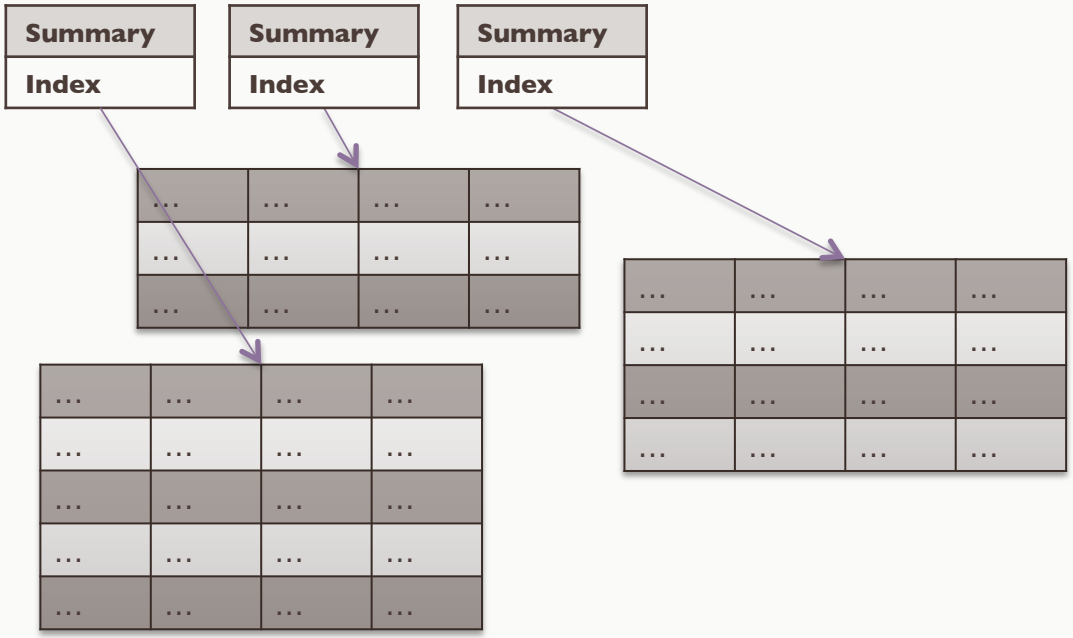


What is an SSTable and what are its characteristics?

- For each *SSTable*, two structures are created
  - Partition index – list of its primary keys and row start positions
  - Partition summary – in-memory sample of its *partition index* (default: *1 partition key of 128*)

Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63



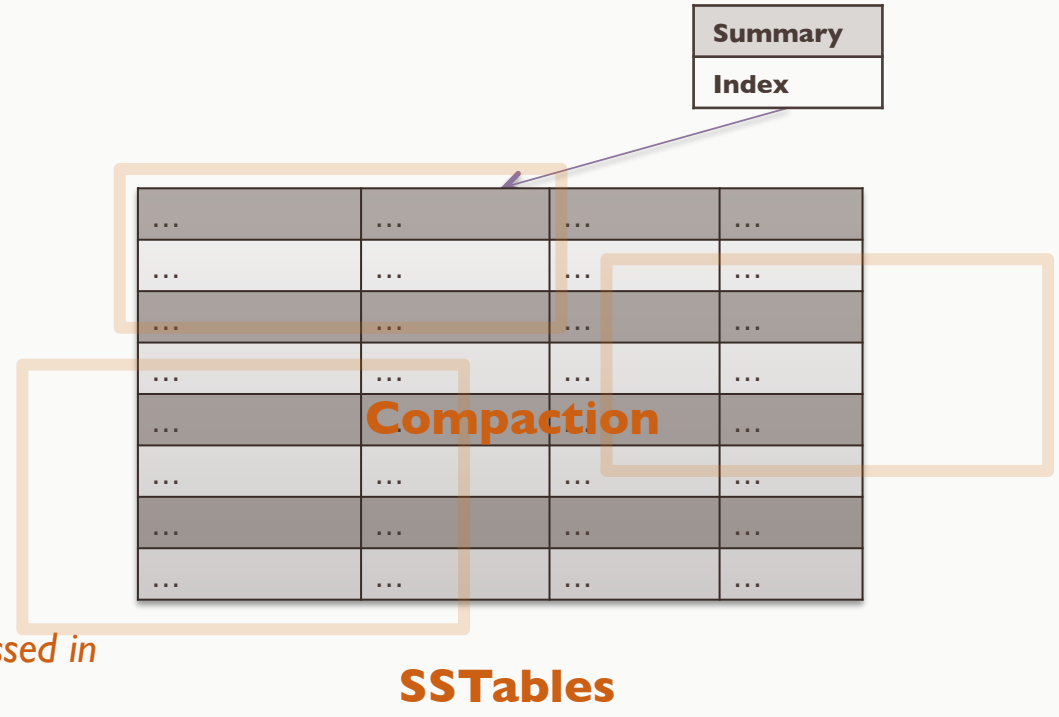
SSTables

What is compaction?

- Updates do mutate *Memtable* partitions, but its *SSTables* are immutable
  - no SSTable seeks/overwrites
  - SSTables just accrue new timestamped updates
- So, *SSTables* must be periodically *compacted*
  - related *SSTables* are merged
  - most recent version of each column is compiled to one partition in one new SSTable
  - partitions marked for deletion are evicted
  - old *SSTables* are deleted

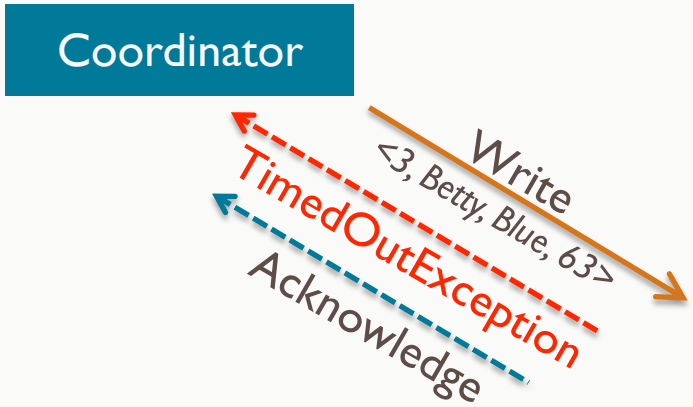
Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty	last:Blue	level:63



Note, Compaction and the Read Path are discussed in further detail later in this course.

What is the significance of idempotency?



Memtable (corresponds to a CQL table)

partition key1	first:Oscar	last:Orange	level:42
partition key2	first:Ricky	last:Red	
partition key3	first:Betty *timestamp 541*	last:Blue *timestamp 541*	level:63 *timestamp 541*
partition key3	first:Betty *timestamp 583*	last:Blue *timestamp 583*	level:63 *timestamp 583*

- Due to the high per-operation overhead, Cassandra does not support transactional rollback ("two phase commit")
  - As a result, a Cassandra client could receive an exception from a successful insert/update operation (e.g., *TimedOutException* due to network latency)
- Idempotent operation – always causes the same result
  - Insert/updates are effectively idempotent when run with identical values
  - Operations involving COUNTER columns are not idempotent
- Each column of any write is time-stamped, and only the most recent are read and compacted

Exercise 1: Insert data and observe the write path flow

Learning Objectives

- Understand how data is written to the storage engine
- **Understand the data directories**

Where are the data directories located?

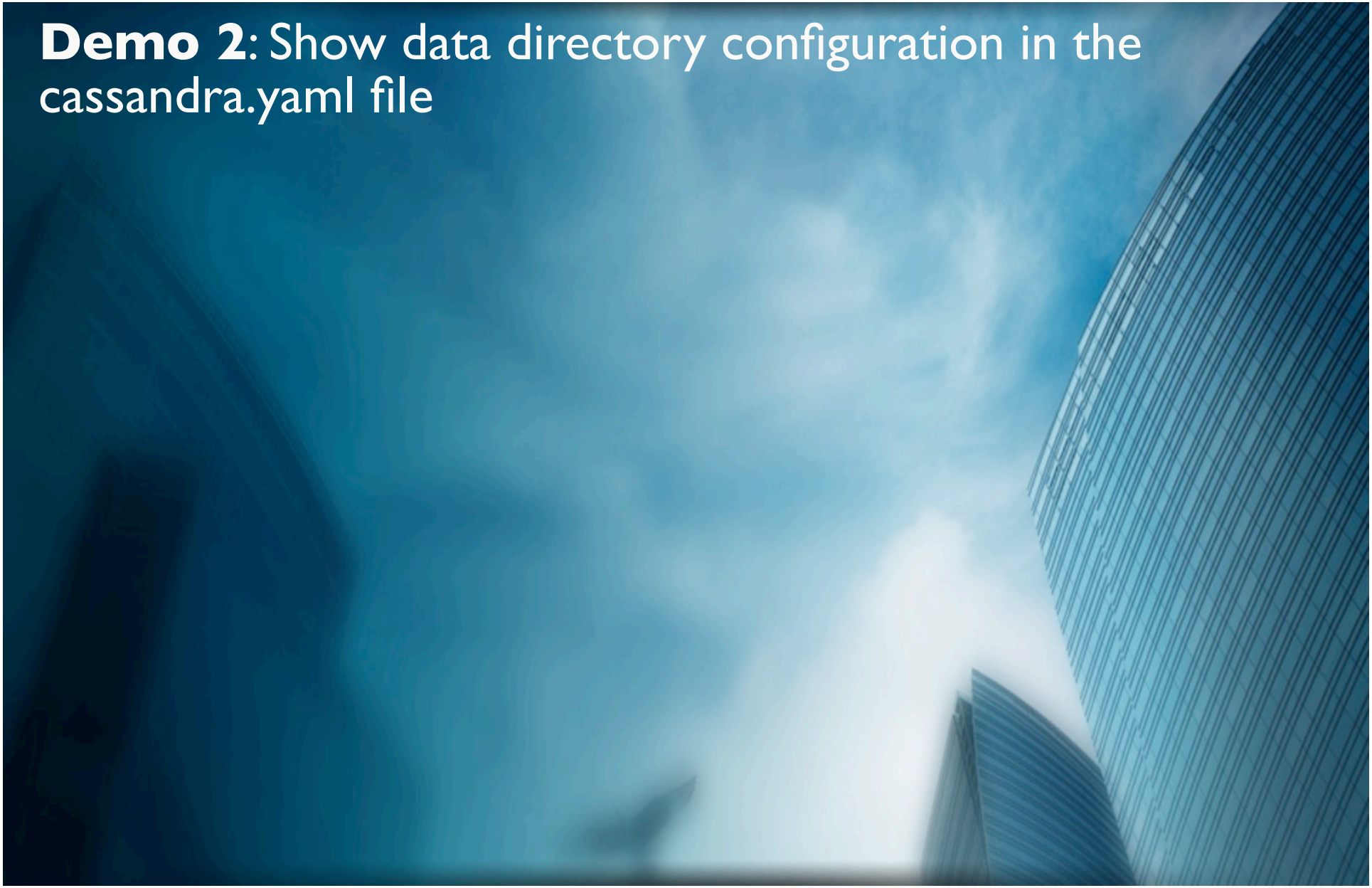
- The *SSTable* and *CommitLog* directory locations are configured in *conf/cassandra.yaml*
  - data_file_directories – if multiple locations, distribution is balanced
  - commitlog_directory – best practice to place on separate disk
  - By default, the files are all placed in */var/lib/cassandra* or in *install_location/data*

cassandra.yaml

```
# Directories where Cassandra should store data on disk.  Cassandra
# will spread data evenly across them, subject to the granularity of
# the configured compaction strategy.
data_file_directories:
    - /var/lib/cassandra/data

# commit log
commitlog_directory: /var/lib/cassandra/commitlog
```

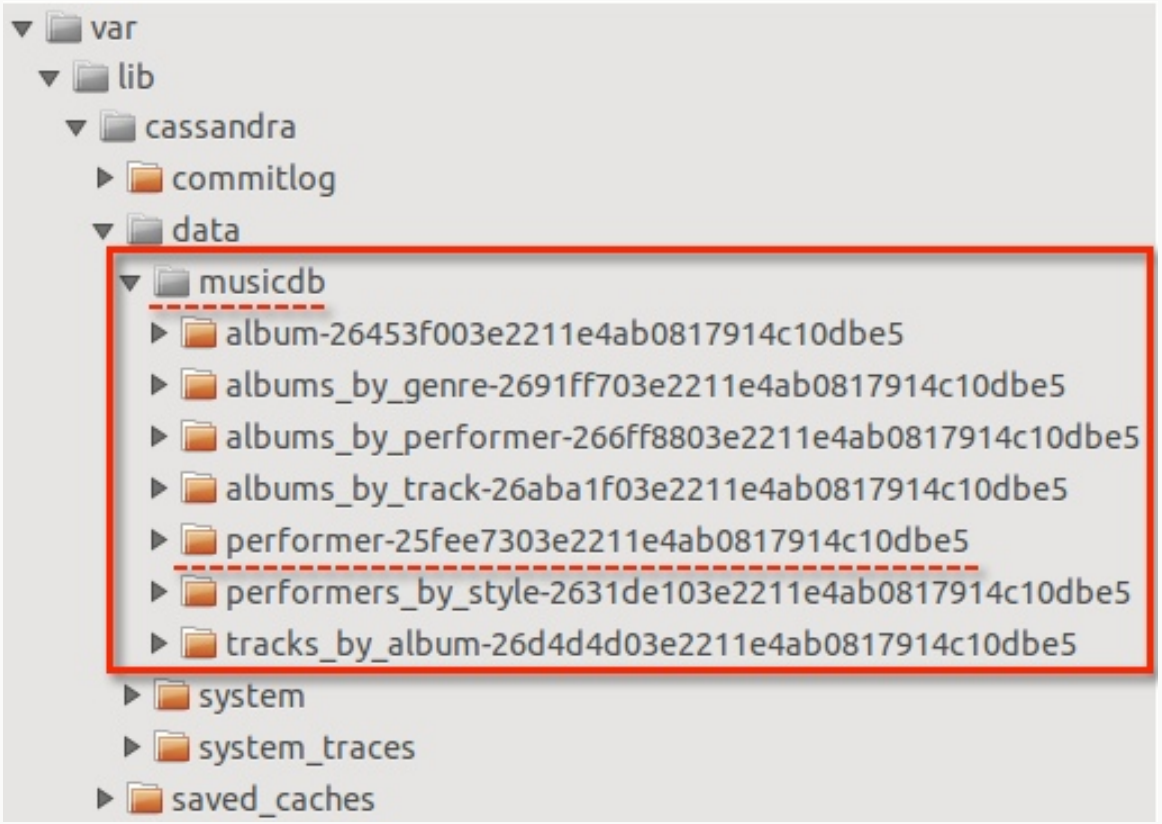
Demo 2: Show data directory configuration in the cassandra.yaml file



How are data directories created for a keyspace?

- Data directories are created by *keyspace* and *table* name / id

  …/data/keyspace/tablename-tableid

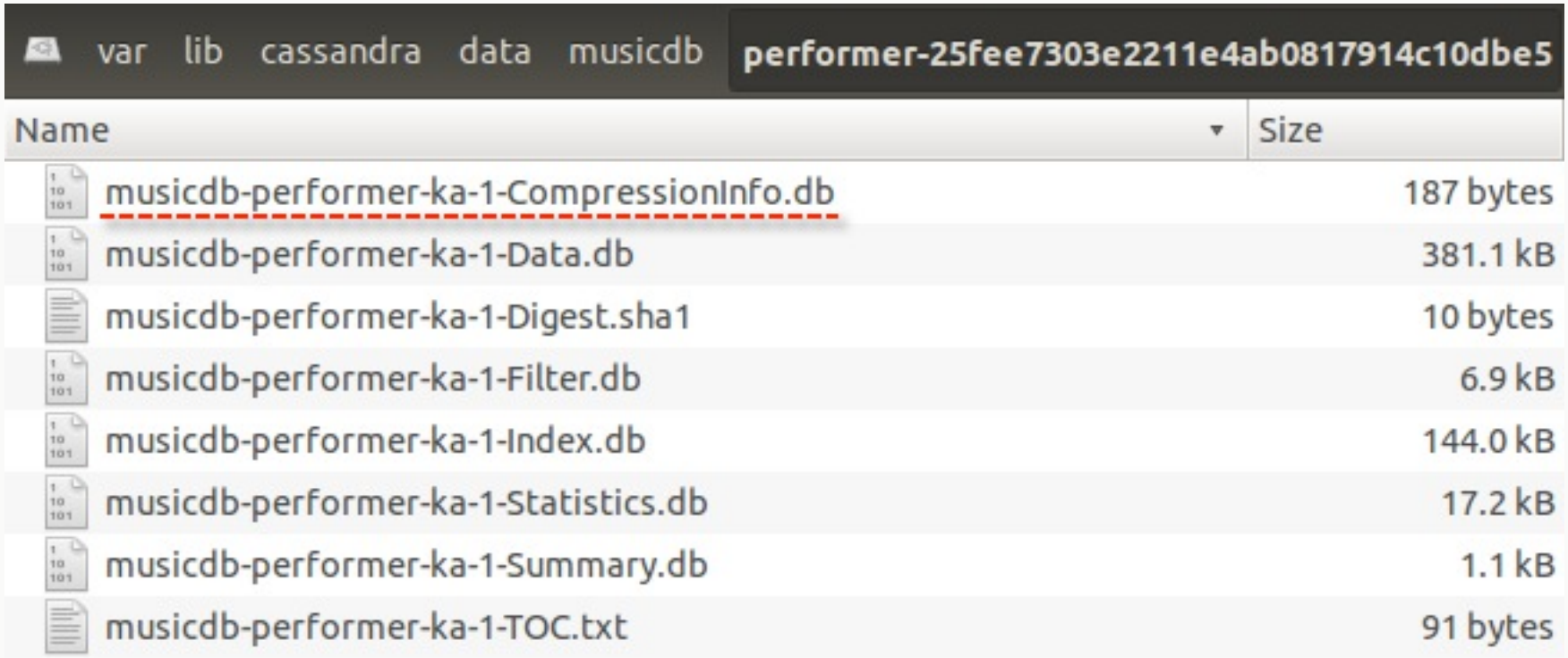



```
CREATE KEYSPACE musicdb
WITH replication = {
 'class' : 'SimpleStrategy',
 'replication_factor' : 1
};

CREATE TABLE performer (
  name VARCHAR,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT,
  PRIMARY KEY (name)
);
```

What files result from Memtable flush or compaction?

- Data files are created by keyspace name, table name, plus
  - Version – SSTable format version (e.g., 'ka' is Cassandra 2.1)
  - Generation – incremented each time *SSTables* flush from a *Memtable*
  - Component – describes the type of file content
- *<keyspace>-<table>-<version>-<generation>-<component>*



What files result from Memtable flush or compaction?

- `-CompressionInfo.db` – metadata for Data file compression
- `-Data.db` – base *SSTable* data including
  - row key, data size, columns index, row level tombstone info, column count, and column list in sorted order by name
- `-Filter.db` – *SSTable* partition keys *Bloom filter*, to optimize reads
- `-Index.db` – index for this *SSTable*, used to optimize reads
  - sorted row keys mapped to offsets in Data file; newer versions also include column index, tombstone, and bloom filter info
- `-Statistics.db` – statistics for this *SSTable*
  - row size and column count estimate, generation numbers of files from which this SSTable was compacted, more
- `-Summary.db` – sampling from *Index* file, used to optimize reads
  - sample size determined by index_interval (default: 1 of each 128)
- `-TOC.txt` – component list for this *SSTable*

What is sstable2json?

DATASTAX

- tools/bin/sstable2json is a utility which exports an *SSTable* in JSON format, for testing and debugging
  - -k display only the partitions for the specified set of keys (limit: 500)
  - -x exclude a specified set of keys (limit: 500)
  - -e enumerate keys only

```
./sstable2json [full_path_to_SSTable_Data_file] | more
```

```
student@cascor:~/cassandra/bin$ sstable2json ~/cassandra/data/data/musicdb/
performer-40f07fd06a3d11e49732d16f287d9d64/musicdb-performer-ka-1-Data.db
[
{"key": "Sheryl Crow",
 "cells": [["","",1415777208220432],
           ["born","1962",1415777208220432],
           ["country","United States",1415777208220432],
           ["died",1415777208,1415777208220432,"d"],
           ["founded",1415777208,1415777208220432,"d"],
           ["style","Rock",1415777208220432],
           ["type","artist",1415777208220432]]},
{"key": "Black Bottle Scotch Whisky Pipe Band",
 "cells": [["","",1415777208220560],
           ["born",1415777208,1415777208220560,"d"],
           ["country","Scotland",1415777208220560],
           ["died",1415777208,1415777208220560,"d"],
```

Exercise 3: Insert data and observe SSTables created

Summary

- Cassandra writes fast because it sequentially appends to a log, without seeking
- A *Memtable* is an in-memory structure corresponding to a CQL table and its indexes
- The *CommitLog* is an append-only log, replayed to restore a downed node's Memtables
- *SSTables* are *Memtable* snapshots periodically flushed to disk
- *Compaction* is a periodic process to merge and optimize *SSTables*
- *CommitLog* accrues in memory and syncs to disk in a batch or periodic manner
- When *Memtables* flush to *SSTables*, heap memory is cleared and the *CommitLog* is truncated
- Flush happens at *Memtable_total_space_in_mb*, *commitlog_total_space_in_mb*, or *nodetool flush*

Summary

- Total table data is the current state of a *Memtable* plus its *SSTables*
- Writes should be idempotent, they persist if acknowledgment fails
- Each column in any write is time-stamped; only the most current are read and compacted
- *data_file_directories* and *commitlog_directory* are set in *cassandra.yaml*
- Each CQL table in a keyspace has a corresponding keyspace name/ table name/table id folder
- Data file names are comprised of keyspace-table-version-generation-component.db
- Data file components are: *Data*, *Index*, *Summary*, *Filter*, *CompressionInfo*, *Statistics*, *TOC*
- *sstable2json* converts a SSTable to JSON for debug/testing

Review Questions

- What happens when a *Memtable* is flushed?
- What causes a *Memtable* to flush?
- What is the relationship of a CQL table to *Memtables* and *SSTables?*
- Do disk seeks happen during writes?
- How are data files organized?