Working with the Cassandra Read Path

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

Leo Schuman, Joe Chu

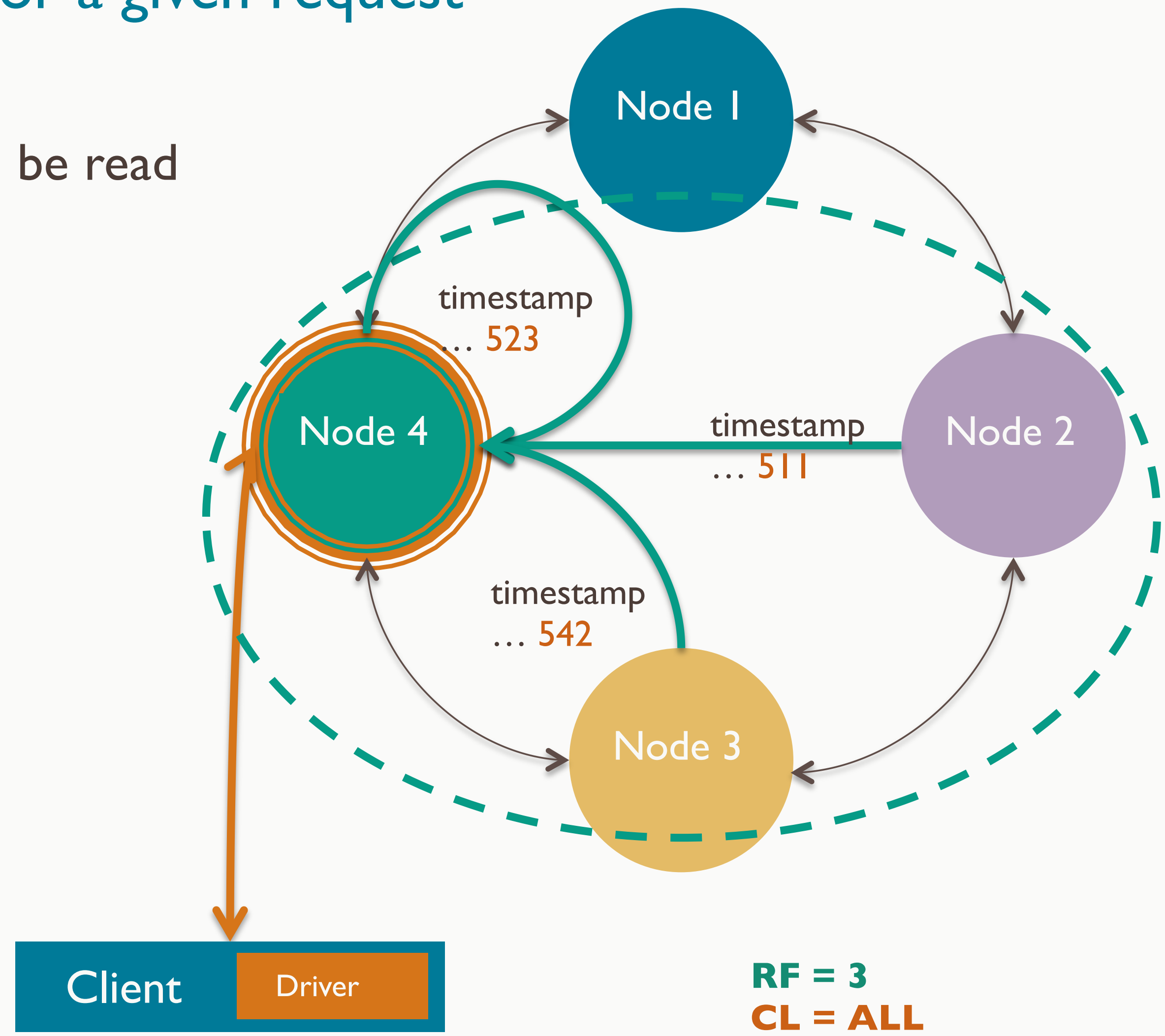
Oct 20, 2014

Learning Objectives

- **Understand how data is read from the storage engine**
- Read data from Cassandra

How does the read path flow among nodes?

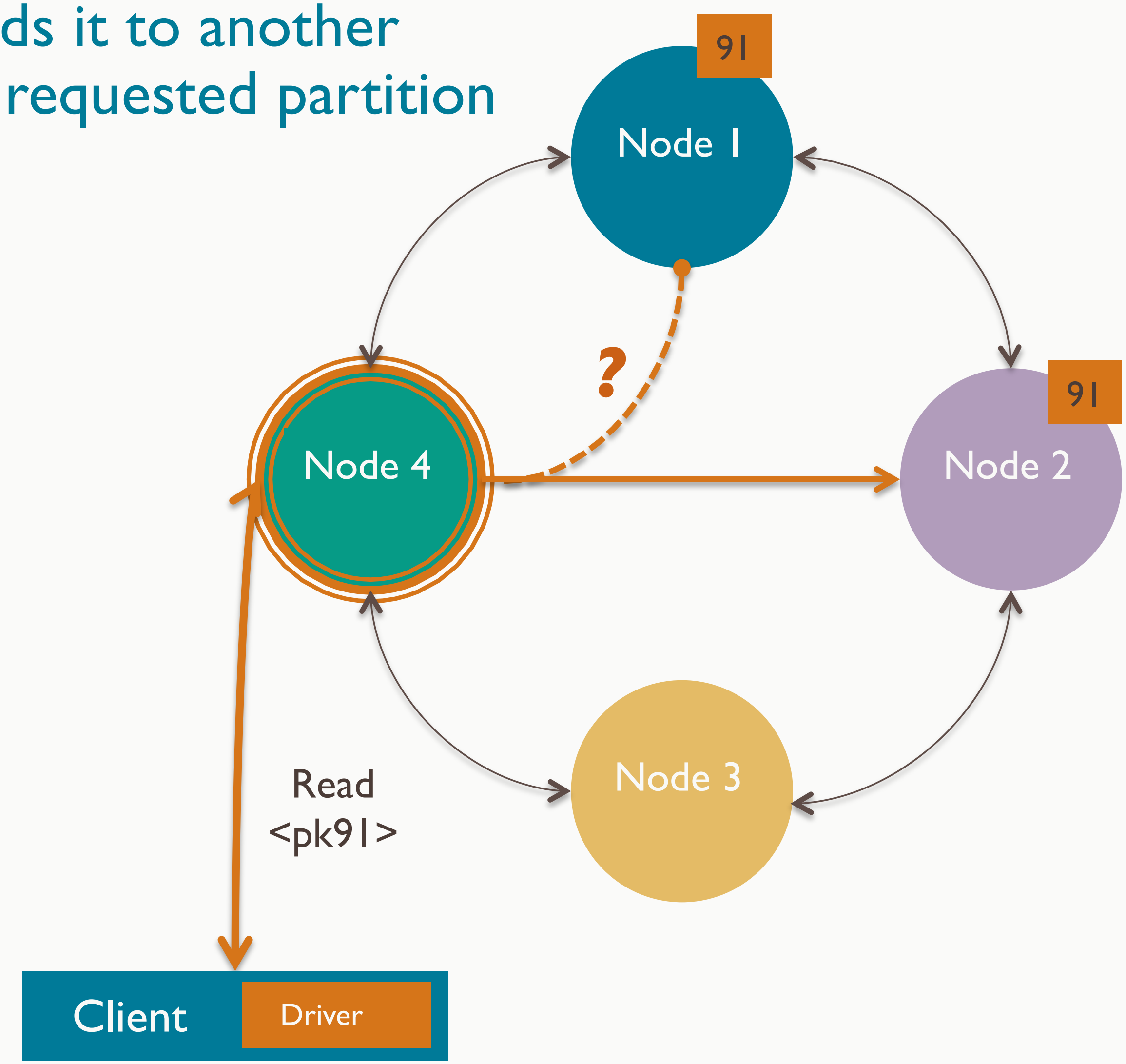
- Cassandra returns the most recent record among the nodes read for a given request
- Consistency Level
  - sets how many nodes will be read for a given request
  - may vary by request



RF = 3
CL = ALL

What are eager retries?

- If a node is slow responding to a request, the coordinator forwards it to another holding a replica of the requested partition
  - New feature in 2.0+
  - Only relevant if RF > 1

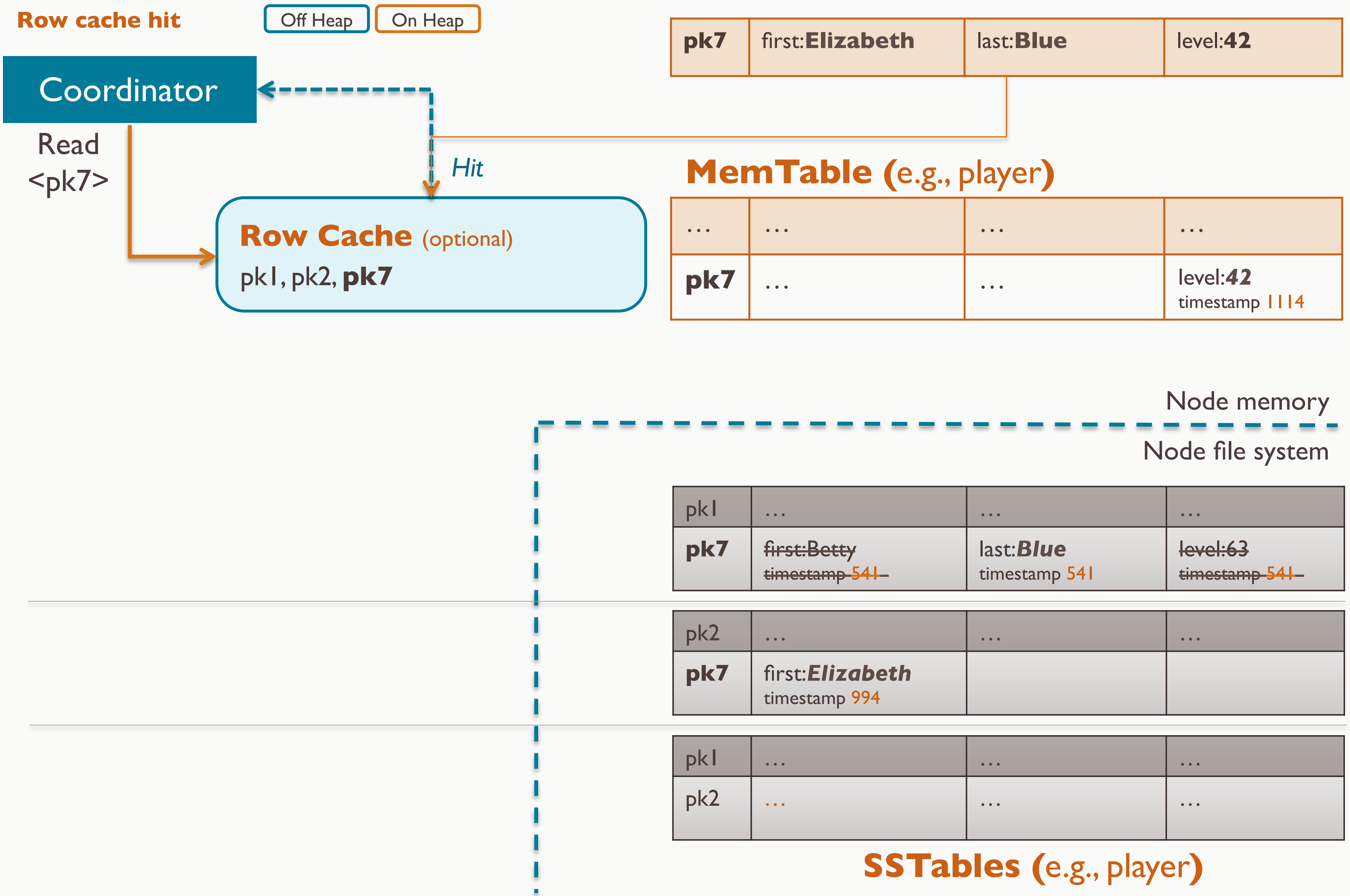


What are the key components of the read path?

- Each node implements in-memory structures for each CQL table
  - MemTable – in-memory table serves data as part of the *merge* process
  - Row Cache – in-memory cache stores recently read rows (optional)
  - Bloom Filters – reports if a *partition key* may be in its corresponding *SSTable*
  - Key Caches – maps recently read *partition keys* to specific *SSTable* offsets
  - Partition Summaries – Sampling from *partition index*
- Each node implements these on disk for each CQL table
  - Partition Indexes – Sorted *partition keys* mapped to their *SSTable* offsets
  - SSTables – static files periodically flushed from a *MemTable*

- Merge – unless served from the *row cache*, a read uses a *partition key* to locate, merge, and return values from a *MemTable* and any related *SSTable* storing values for that key

How does the read path flow on *each* node?



Row cache hit

Row Cache (optional)

pk1, pk2, **pk7**

pk7	first:**Elizabeth**	last:**Blue**	level:**42**

MemTable (e.g., player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

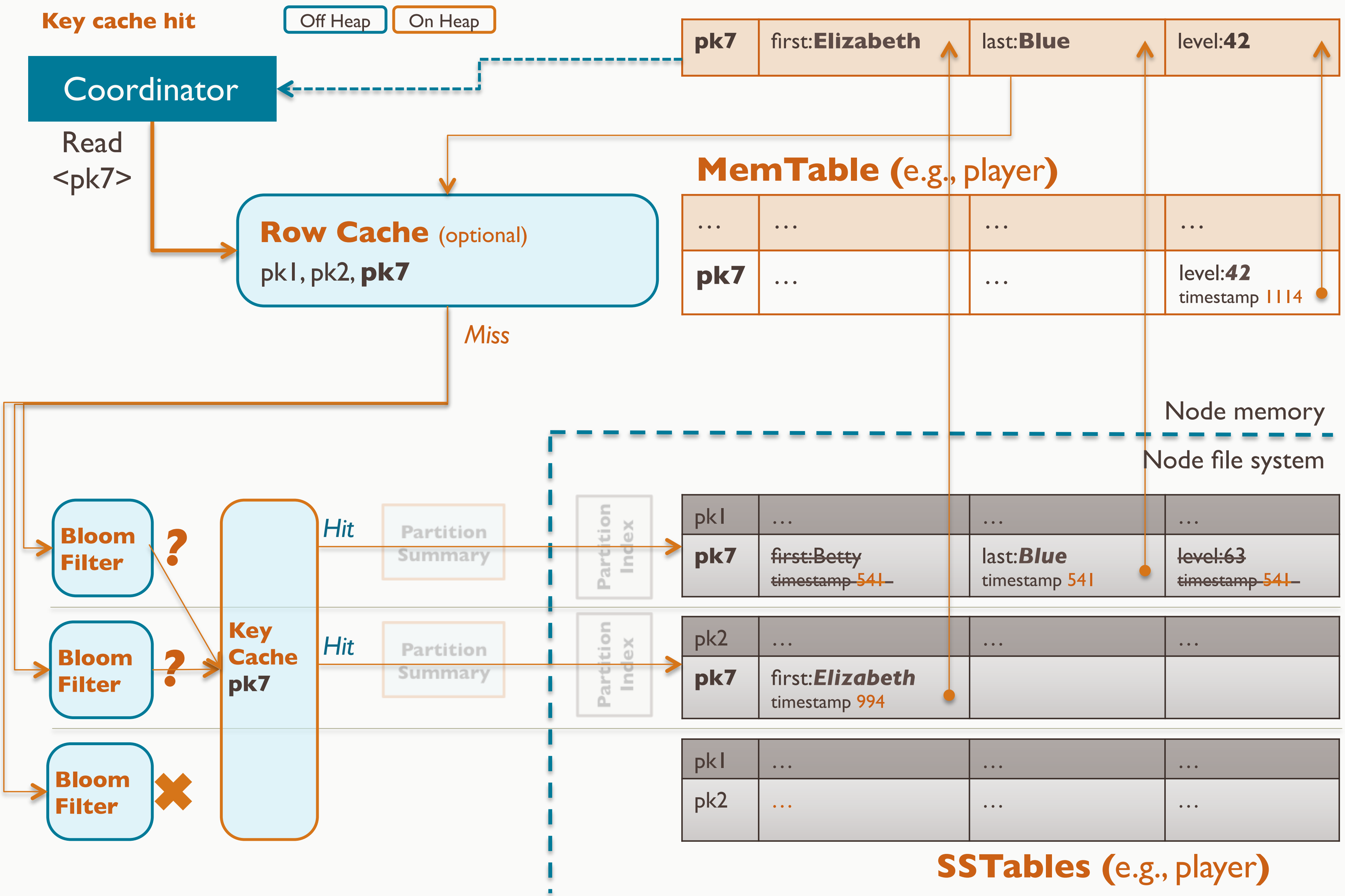
pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

How does the read path flow on *each* node?



pk7	first:**Elizabeth**	last:**Blue**	level:**42**

MemTable (e.g., player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

SSTables (e.g., player)

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

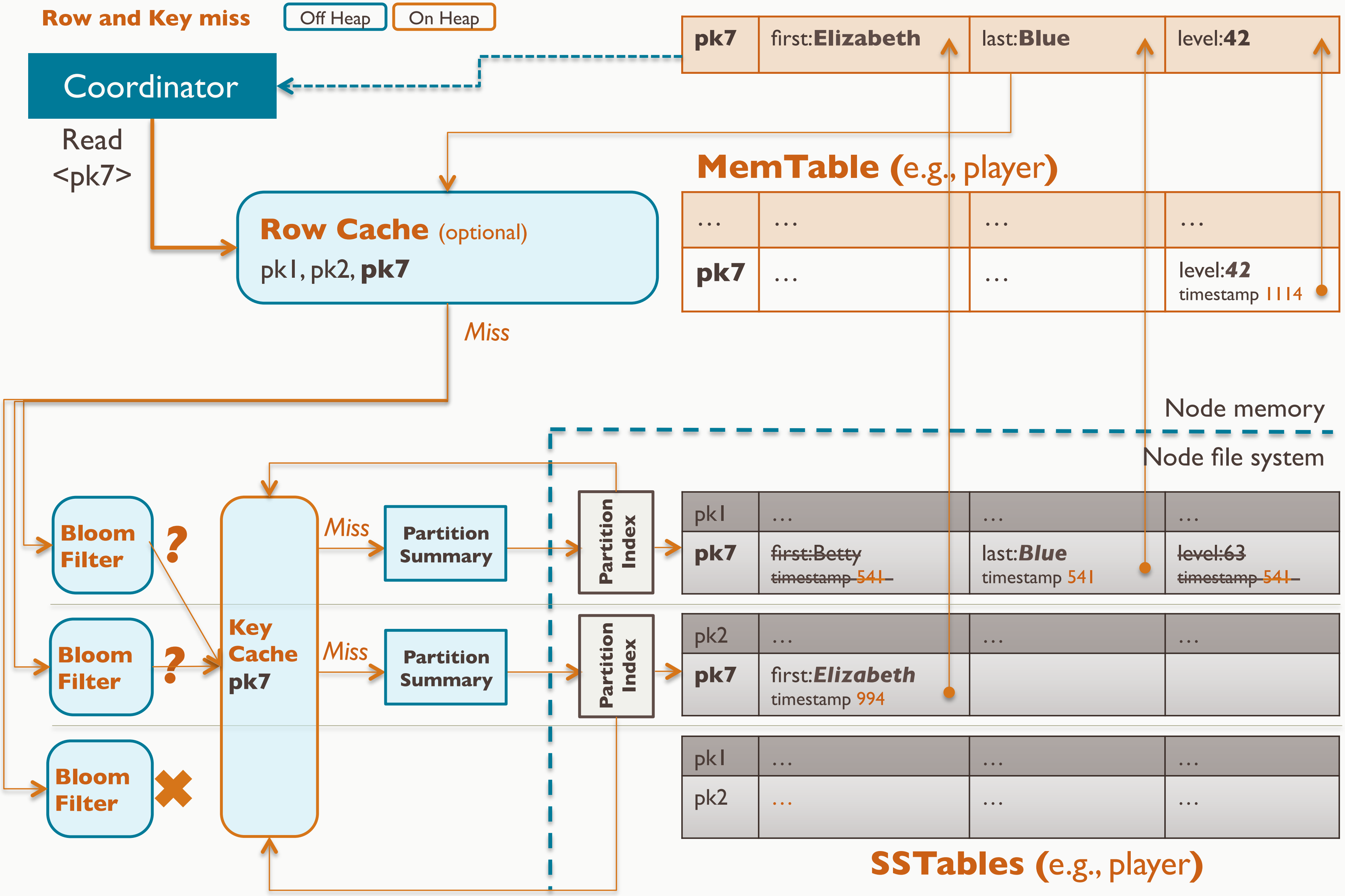
pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

How does the read path flow on *each* node?

DATASTAX

Row and Key miss



Off Heap

On Heap

pk7	first:**Elizabeth**	last:**Blue**	level:**42**

Coordinator

Read <pk7>

Row Cache (optional)

pk1, pk2, **pk7**

Miss

MemTable (e.g., player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

Node memory

Node file system

Bloom Filter ?

Bloom Filter ?

Bloom Filter ✖

Key Cache **pk7**

Miss

Partition Summary

Partition Index

Miss

Partition Summary

Partition Index

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

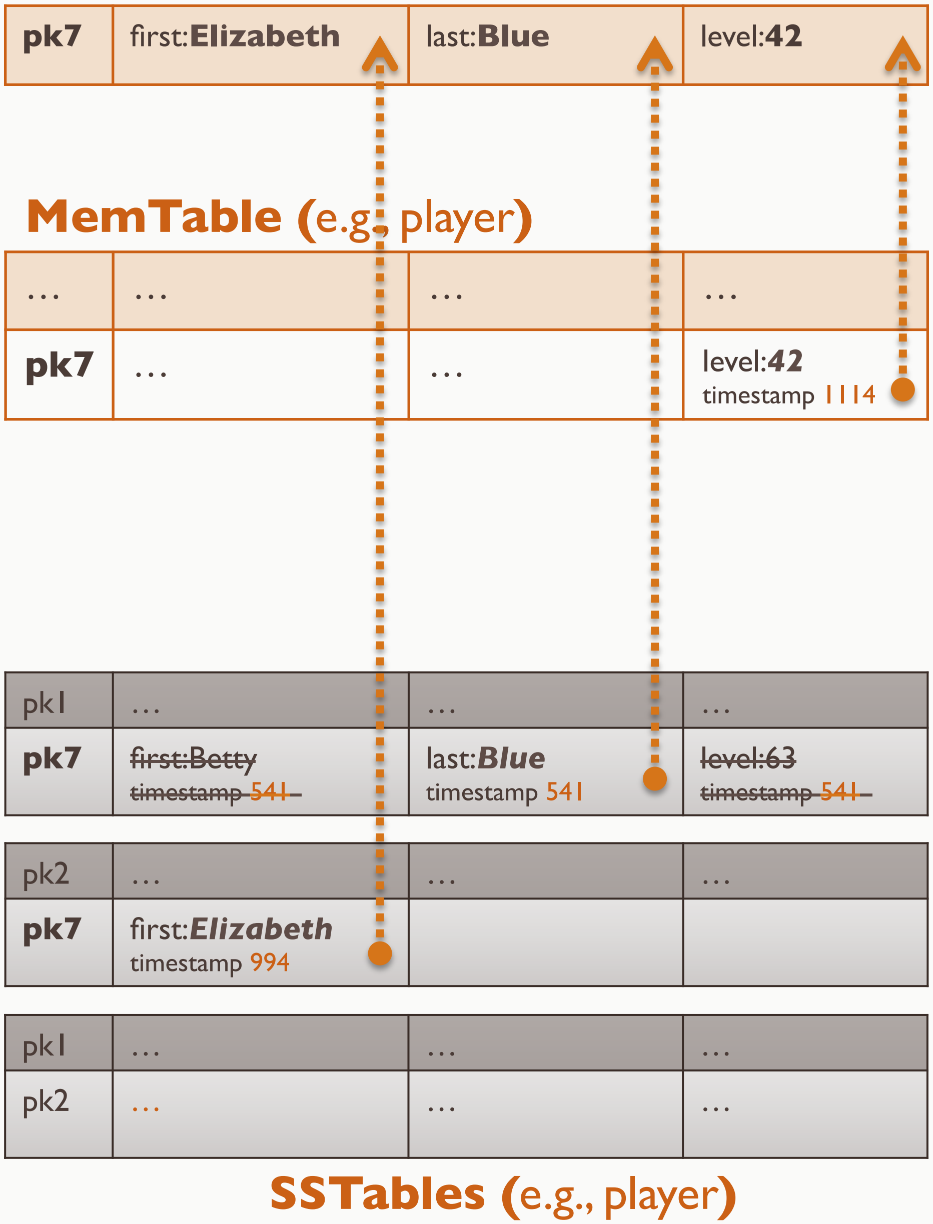
pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

How is a MemTable and its SSTables used during a read?

- Both a *MemTable* and its recent *SSTables* are checked when reading for a partition key
  - the most current column values are combined to form the result



pk7	first:**Elizabeth**	last:**Blue**	level:**42**

MemTable (e.g. player)

…	…	…	…
pk7	…	…	level:**42** timestamp 1114

pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

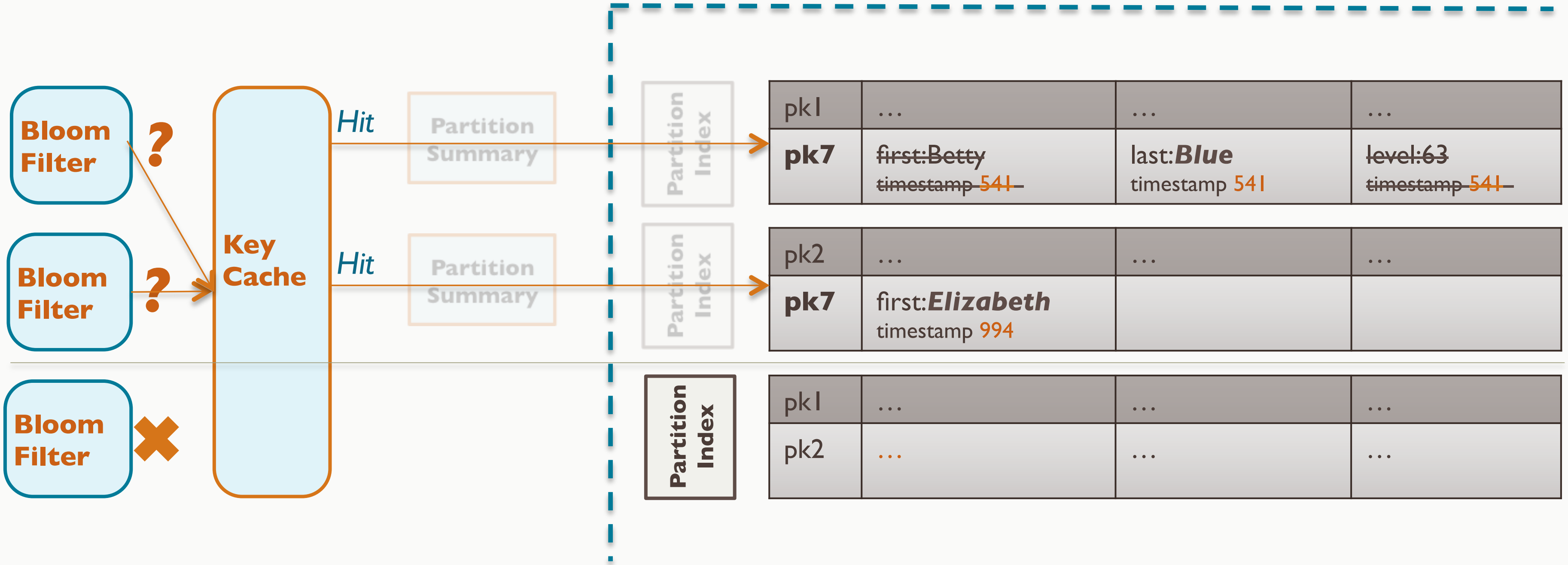
pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

SSTables (e.g., player)

What is a Bloom filter and how does it optimize a read?

- A probabilistic data structure testing if a key may be in a SSTable
  - each SSTable has a Bloom filter on disk, used from off-heap memory
  - false positives are possible, false negatives are not
  - larger tables have a higher possibility of false positives
    - 1gb to 2gb per billion partitions in a SSTable
- Eliminates seeking a partition key in any SSTable without it



pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

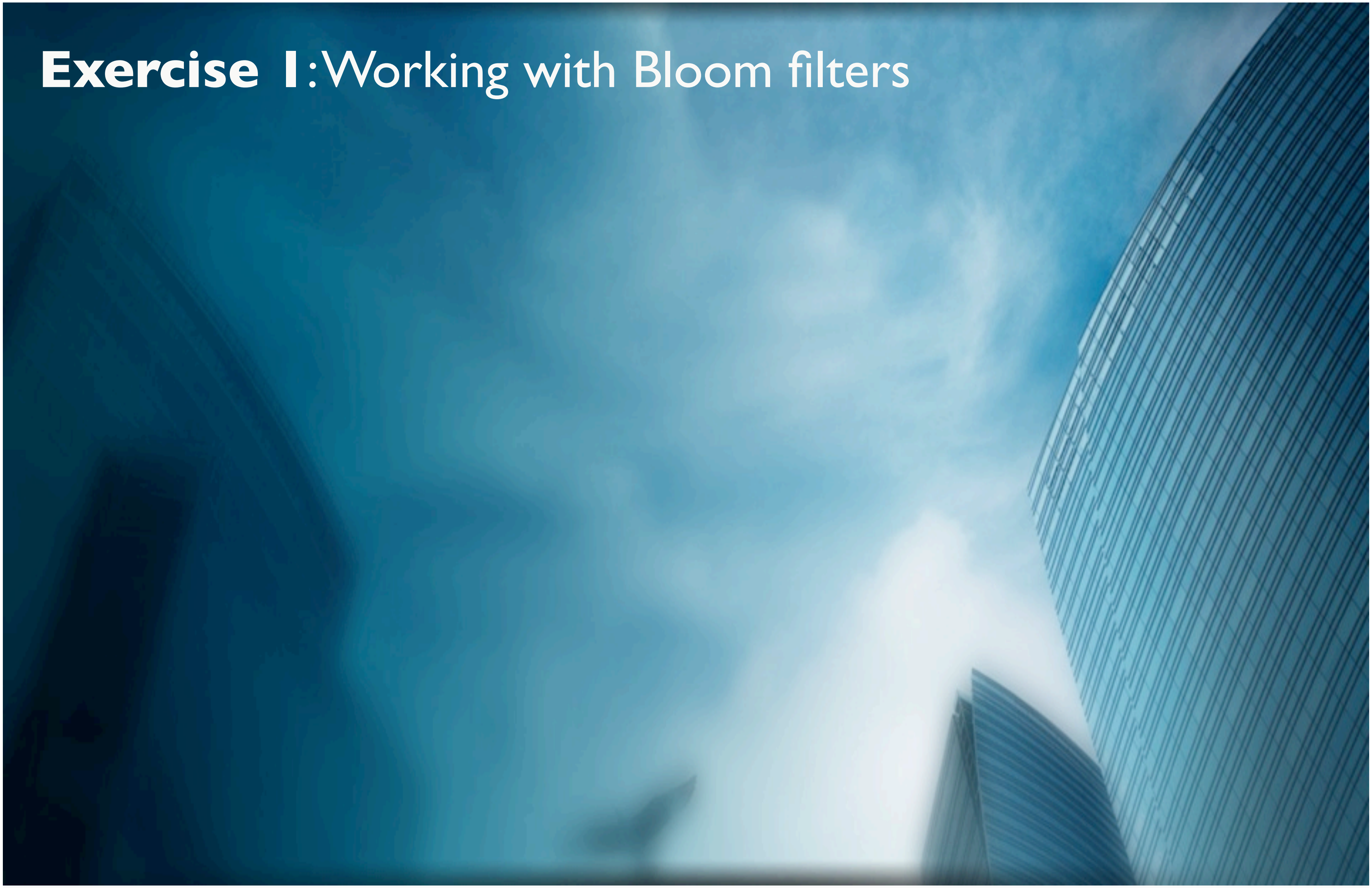
pk1	…	…	…
pk2	…	…	…

What is the bloom_filter_fp_chance table setting?

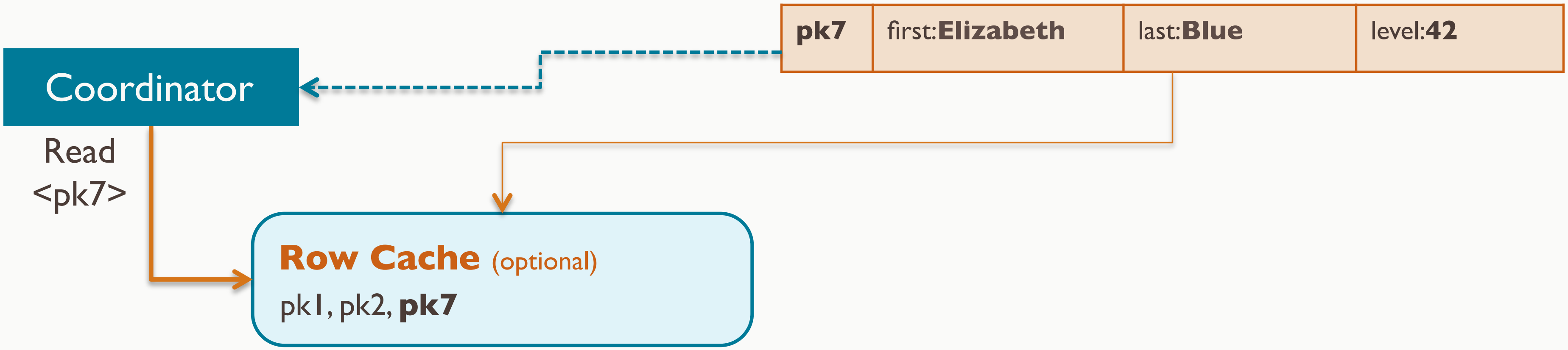
- Controls the percentage chance of false positive results from the Bloom filters for SSTables flushed for a specified table
- Values range from 0.0 to 1.0
  - 0.0 no false positives, greatest memory use
  - 0.1 maximum recommended setting, diminishing returns if higher
  - 1.0 Bloom filtering disabled for this table
- Default setting depends on compaction strategy
  - 0.01 Size-tiered compaction (STC)
  - 0.1 Leveled compaction (LCS)

```
ALTER TABLE player
WITH bloom_filter_fp_chance = 0.1;
```

Exercise 1: Working with Bloom filters



What is the row cache and how is it configured?



Coordinator

Read <pk7>

Row Cache (optional)

pk1, pk2, **pk7**

pk7	first:**Elizabeth**	last:**Blue**	level:**42**

- The *merged* row(s) for a *partition key* is saved in off-heap memory
- Row caching is enabled in CQL with the *caching* and *rows_per_partition* properties
  - ALL – cache all rows for a partition key
  - n – cache the first *n* rows for a partition key
  - NONE – (default) disable row caching for this table

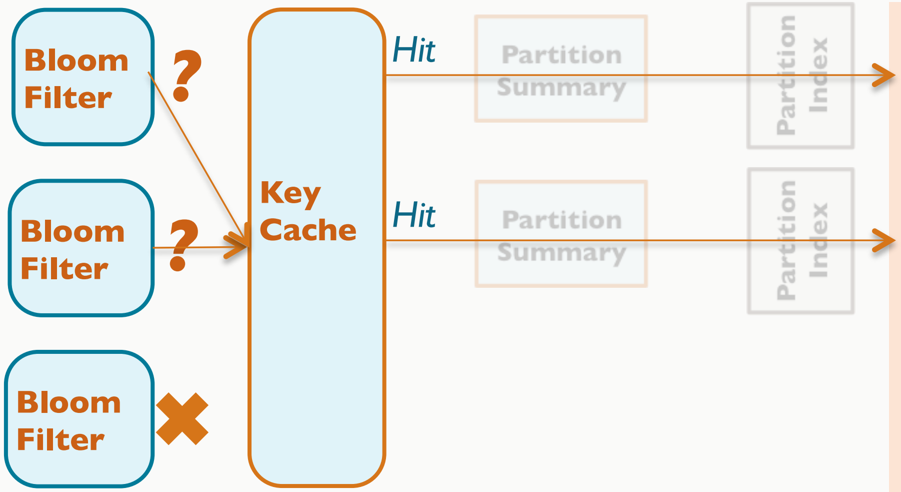
```
CREATE TABLE player (
  first text PRIMARY KEY,
  last text,
  level text
)
WITH caching = {'keys': 'ALL',
'rows_per_partition': '1'};
```

What is the row cache and how is it configured?

- Caches can periodically save to disk improving node restart speed
- Row cache size and save period are set globally for all tables on a node in *cassandra.yaml*
  - row_cache_size_in_mb – maximum row cache size, set to 0 to disable
  - row_cache_save_period – periodicity in seconds at which row cache should be saved to the *saved_caches_directory*, improves cache usage following a node restart, set to 0 to disable row cache saving
  - row_cache_keys_to_save – max number of cached rows to save each period, if disabled all cached rows are saved
  - saved_caches_directory – location to save row, key, and counter caches
    - default: /var/lib/cassandra/saved_caches

What is the key cache and how is it configured?

- Key caching saves a *partition key* and its *offset position(s)* in the SSTable(s) for a MemTable
  - one key cache entry for each *SSTable* holding a *replica* of this partition
  - reduces a read to a single seek per recent replica
- Key caching is enabled in CQL with the *caching* and *keys* properties
  - ALL – (default) enable key caching for this table
  - NONE – disable key caching for this table



```
CREATE TABLE player (
  first text PRIMARY KEY,
  last text,
  level text
)
WITH caching = {'keys': 'ALL',
'rows_per_partition': '1'};
```

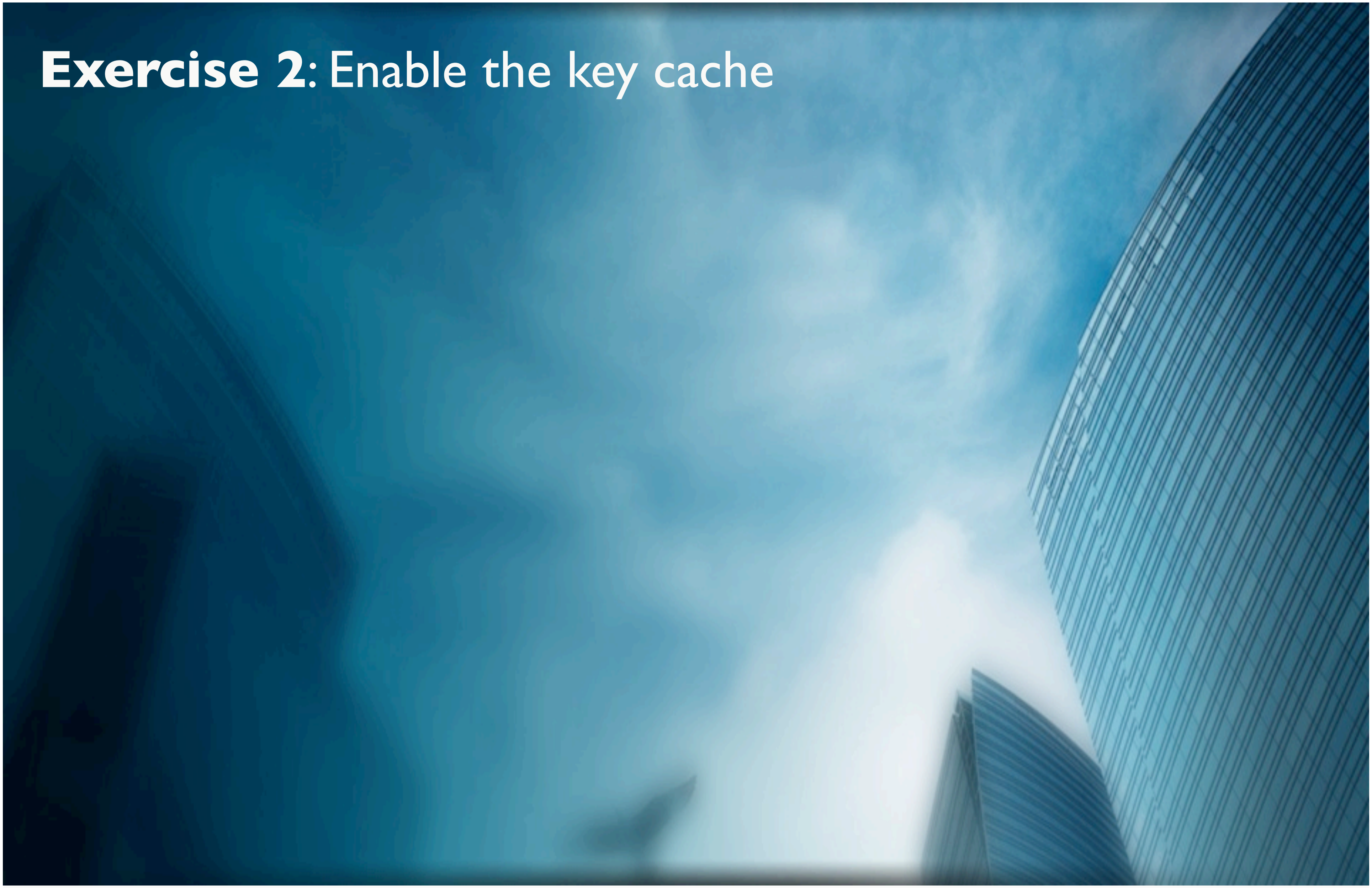
What is the key cache and how is it configured?

- Caches can periodically save to disk, to improve node restart speed
- Key cache size and save period are set globally for all tables on a node in *cassandra.yaml*
  - key_cache_size_in_mb – maximum key cache size, set to 0 to disable
    - default: 5% of available heap or 100mb, whichever is smaller
  - key_cache_save_period – periodicity in seconds at which key cache should be saved to the *saved_caches_directory*, improves cache usage following a node restart, set to 0 to disable key cache saving
  - key_cache_keys_to_save – max number of cached keys to save each period, if disabled all cached keys are saved
  - saved_caches_directory – location to save row, key, and counter caches
    - default: /var/lib/cassandra/saved_caches
- Enabling key caching is commonly termed "pre-heating"

What is the counter cache and how is it configured?

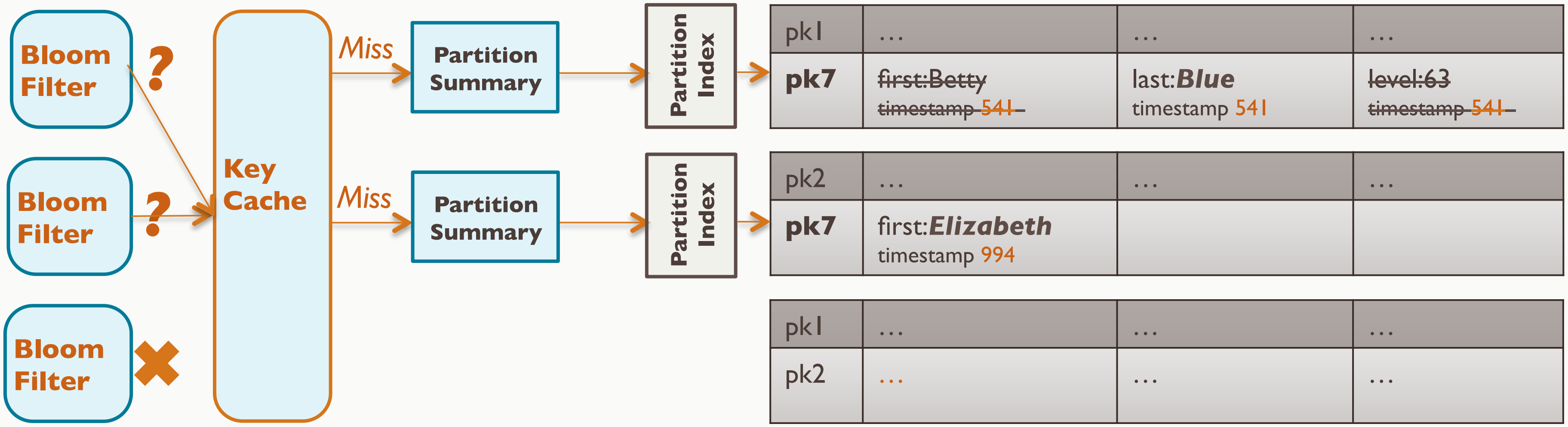
- Counter caching saves the clock and count of a counter in memory
  - helps reduce lock contention for the read-before-write counter updates
  - only the local tuple for node is saved in the counter cache
- Counter cache size and save period are set globally for all counter tables on a node in *cassandra.yaml*
  - counter_cache_size_in_mb – maximum counter cache size, set to 0 to disable
    - default: 2.5% of available heap or 50mb, whichever is smaller
  - counter_cache_save_period – periodicity in seconds at which the counter cache should be saved to the *saved_caches_directory*, improves cache usage following a node restart, set to 0 to disable counter cache saving
  - counter_cache_keys_to_save – max number of cached keys to save each period, if disabled all cached keys are saved
  - saved_caches_directory – location to save row, key, and counter caches
    - default: /var/lib/cassandra/saved_caches

Exercise 2: Enable the key cache



What are partition summaries and how are they used?

- If a partition key's location is not in the *key cache*, the read must seek the requested partition on disk
- The *partition summary* in an in-memory sampling from a *partition index*, used to locate a key's approximate location in the full index
  - default sample ratio is 1 per 128 partition keys in the index
  - configured with the table property min_index_interval (default: 128) and max_index_interval (default: 2048)
  - held in off-heap memory



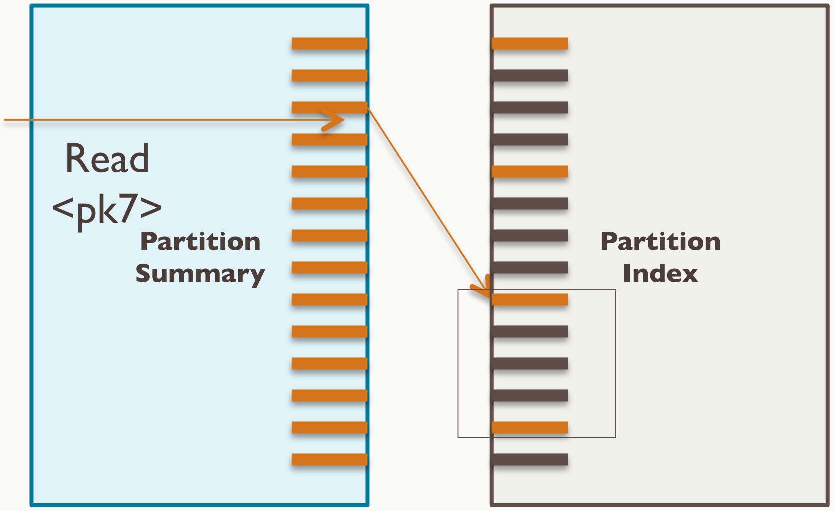
pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

pk1	…	…	…
pk2	…	…	…

What are partition summaries and how are they used?

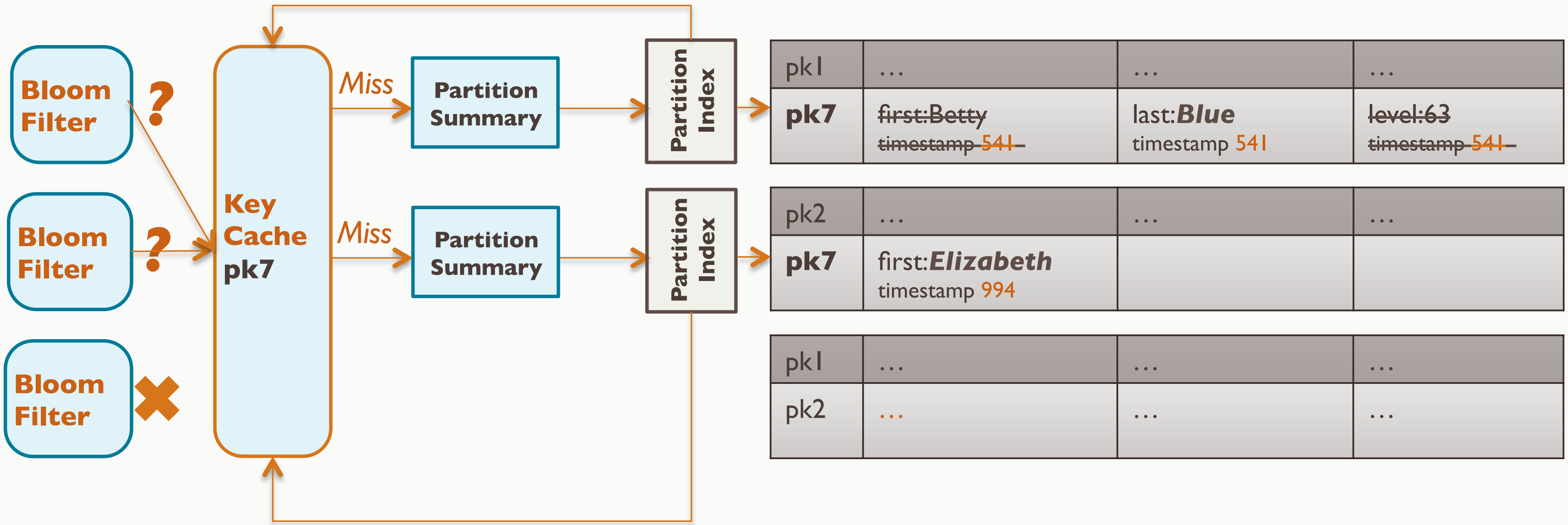
- If a partition key's location is not in the *key cache*, the read must seek the requested partition on disk
- The *partition summary* in an in-memory sampling from a *partition index*, used to locate a key's approximate location in the full index
  - default sample ratio is 1 per 128 partition keys in the index
  - configured with the table property min_index_interval (default: 128) and max_index_interval (default: 2048)
  - held in off-heap memory



```
CREATE TABLE player (
  first text PRIMARY KEY,
  last text,
  level text
)
WITH min_index_interval = 256
AND max_index_interval = 2048;
```

What are partition indexes and how are they used?

- The *partition index* of each SSTable provides the physical offset locations for each of its partitions, sorted by *partition key*
- Starting with the approximate location from the *partition summary*, the *partition index* is read to find the physical position of a partition
  - Once found, the location of this partition key is added to the *key cache*



pk1	…	…	…
pk7	~~first:Betty~~ ~~timestamp 541~~	last:***Blue*** timestamp 541	~~level:63~~ ~~timestamp 541~~

pk2	…	…	…
pk7	first:***Elizabeth*** timestamp 994		

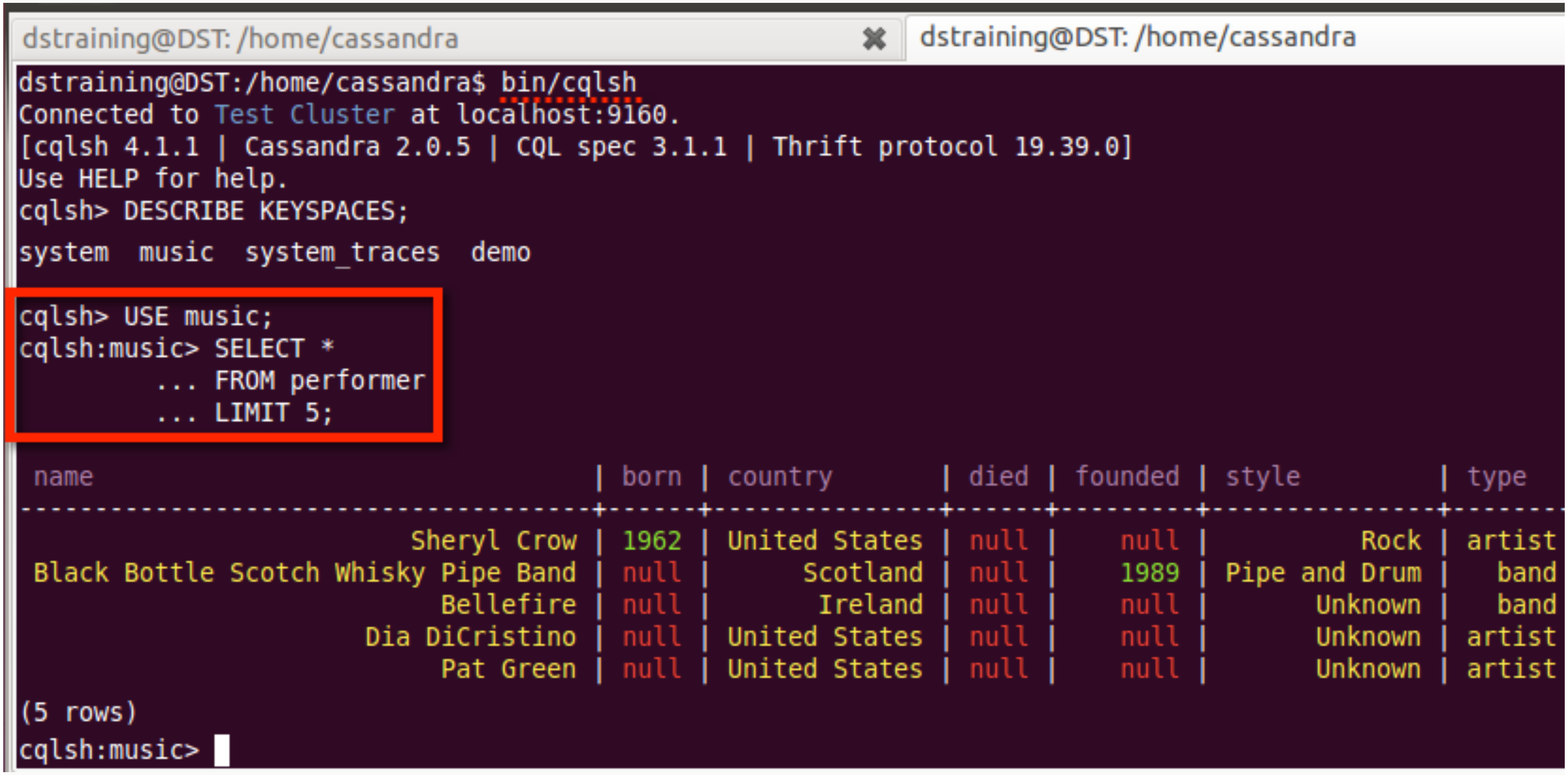
pk1	…	…	…
pk2	…	…	…

Learning Objectives

- Understand how data is read from the storage engine
- **Read data from Cassandra**

How do you execute CQL queries in cqlsh?

- As learned earlier, *cqlsh* enables command line CQL execution



dstraining@DST: /home/cassandra
dstraining@DST: /home/cassandra

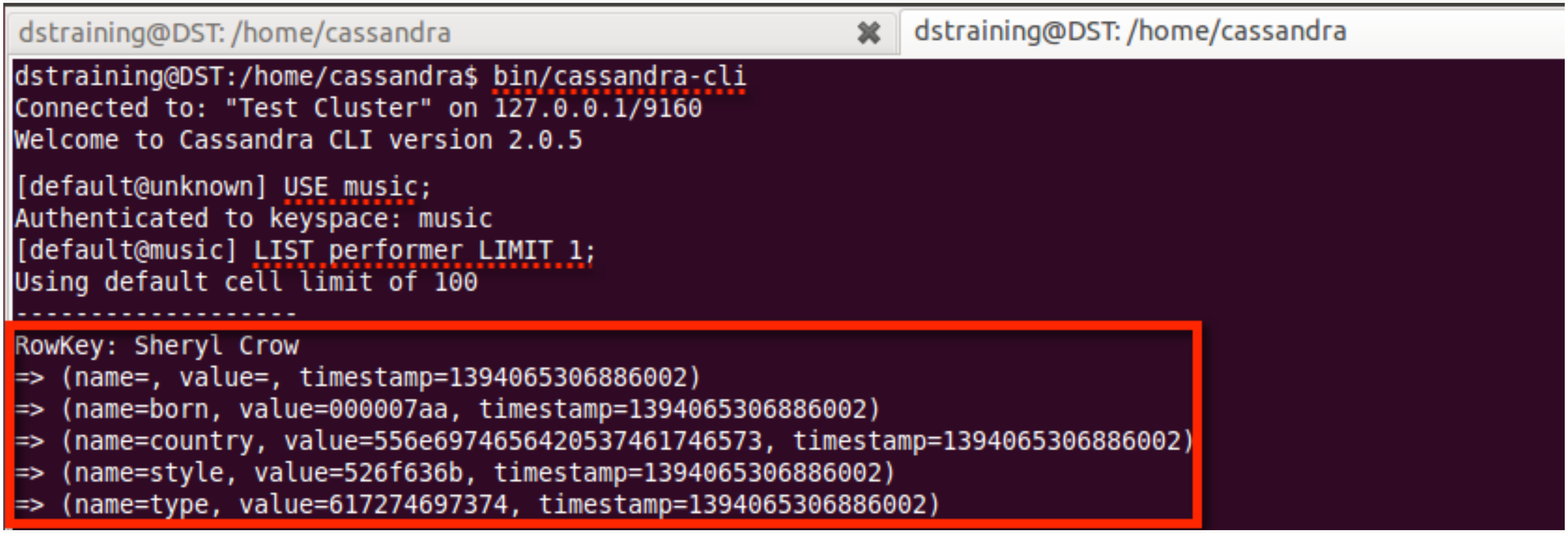
```
dstraining@DST:/home/cassandra$ bin/cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.5 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh> DESCRIBE KEYSPACES;

system  music  system_traces  demo

cqlsh> USE music;
cqlsh:music> SELECT *
         ... FROM performer
         ... LIMIT 5;

 name                                 | born | country       | died | founded | style         | type
--------------------------------------+------+---------------+------+---------+---------------+-------
                          Sheryl Crow | 1962 | United States | null |    null |          Rock | artist
 Black Bottle Scotch Whisky Pipe Band | null |      Scotland | null |    1989 | Pipe and Drum |   band
                            Bellefire | null |       Ireland | null |    null |       Unknown |   band
                       Dia DiCristino | null | United States | null |    null |       Unknown | artist
                            Pat Green | null | United States | null |    null |       Unknown | artist

(5 rows)
cqlsh:music>
```

Note, cqlsh and CQL are taught in detail in Module 4 – Introducing the Cassandra Data Model and CQL

How do you examine data storage using CLI?

- The *cassandra-cli* utility is
  - useful for examining and learning the internal storage engine structure
  - deprecated and less functional than CQL and *cqlsh*, which are fully backwards-compatible with column families and data created using Thrift

`use`	`create`	`set`	`get`	`list`
`limit`	`help`	`assume`	`quit`	

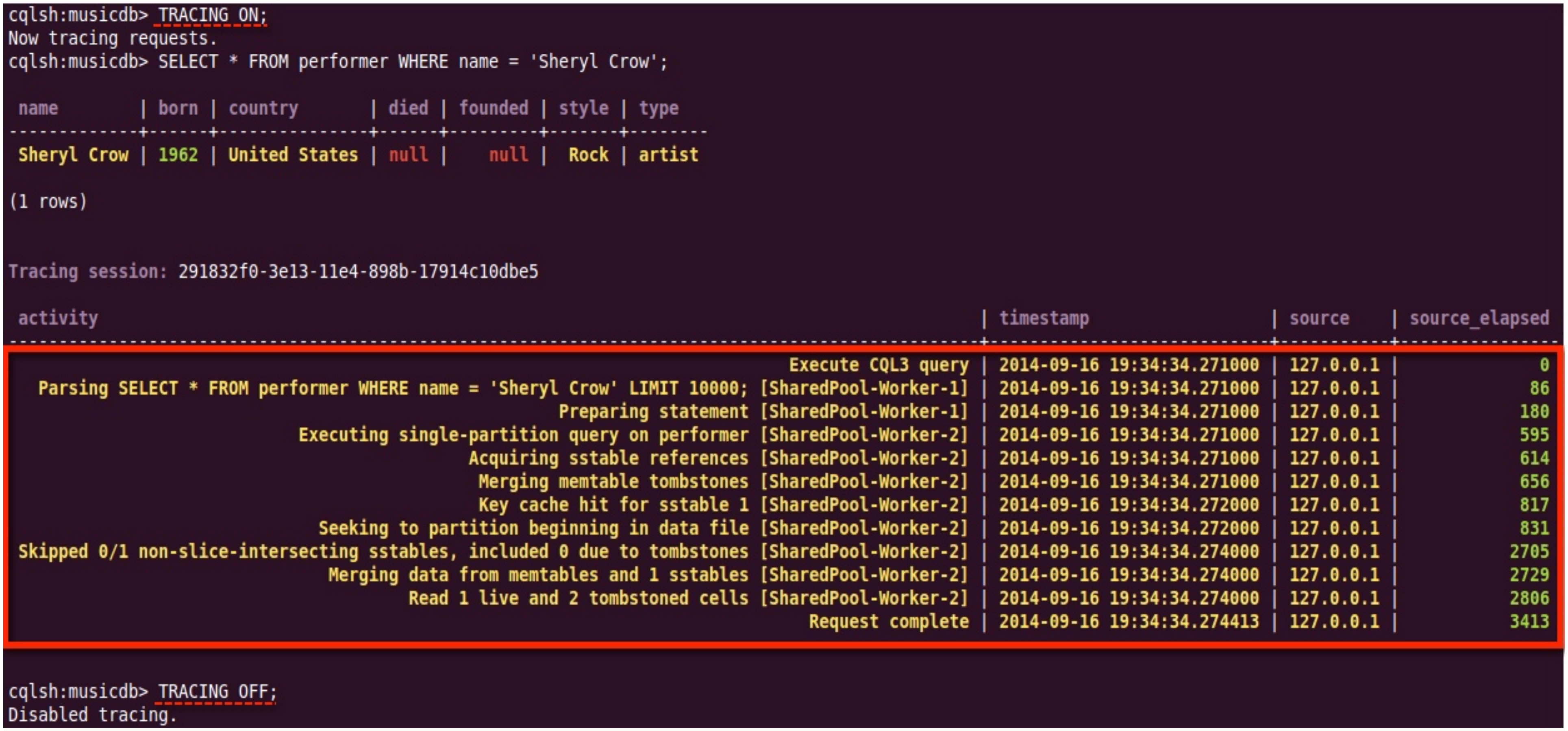

dstraining@DST: /home/cassandra
dstraining@DST: /home/cassandra

```
dstraining@DST:/home/cassandra$ bin/cassandra-cli
Connected to: "Test Cluster" on 127.0.0.1/9160
Welcome to Cassandra CLI version 2.0.5

[default@unknown] USE music;
Authenticated to keyspace: music
[default@music] LIST performer LIMIT 1;
Using default cell limit of 100
-------------------
RowKey: Sheryl Crow
=> (name=, value=, timestamp=1394065306886002)
=> (name=born, value=000007aa, timestamp=1394065306886002)
=> (name=country, value=556e6974656420537461746573, timestamp=1394065306886002)
=> (name=style, value=526f636b, timestamp=1394065306886002)
=> (name=type, value=617274697374, timestamp=1394065306886002)
```

Demo 3: Use the CLI to examine data storage

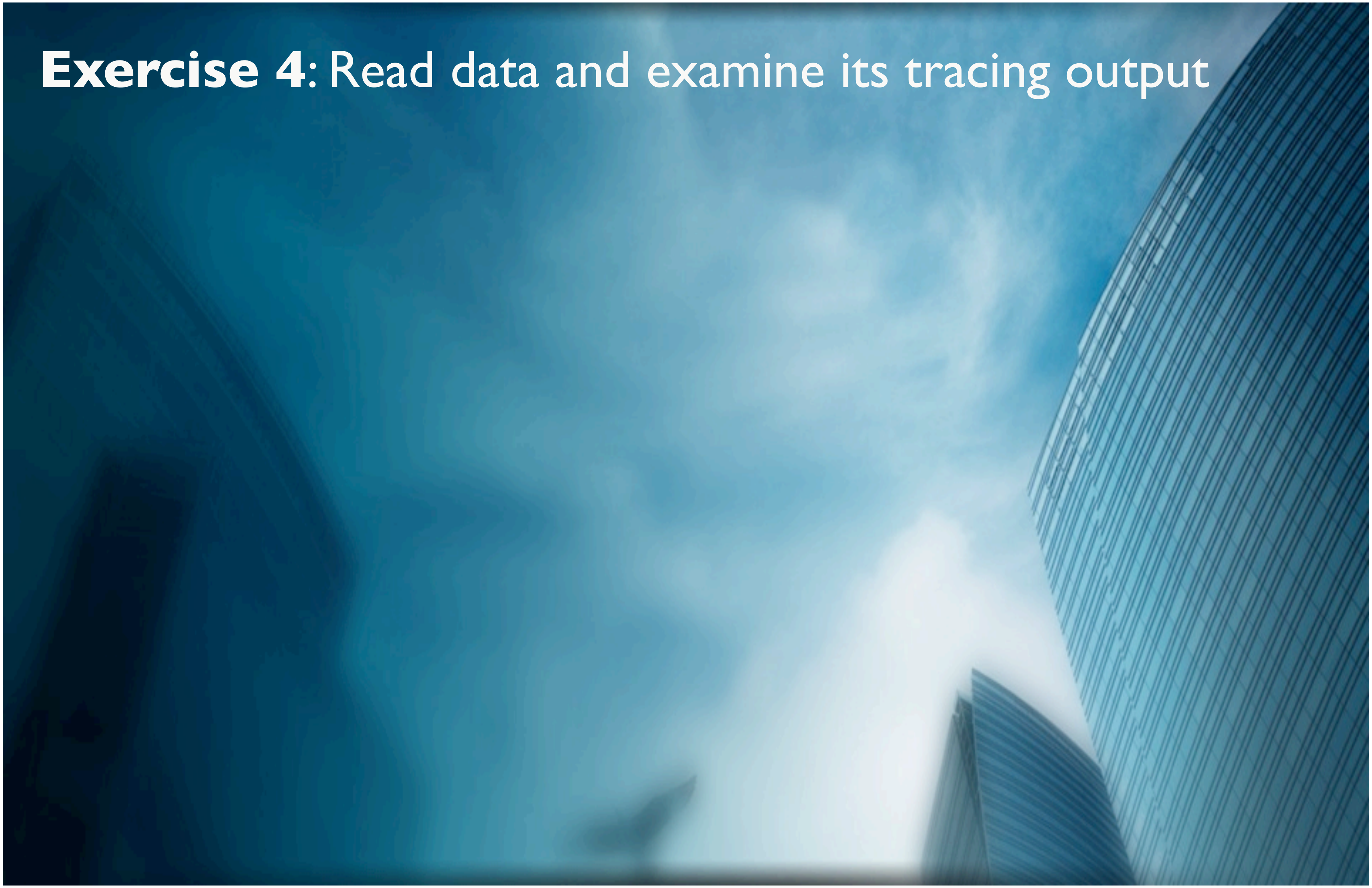
How is CQL tracing enabled and used?

- The CQL TRACING command enables and disables request tracing
  - results displayed and saved to *sessions* and *events* in *system_traces* keyspace



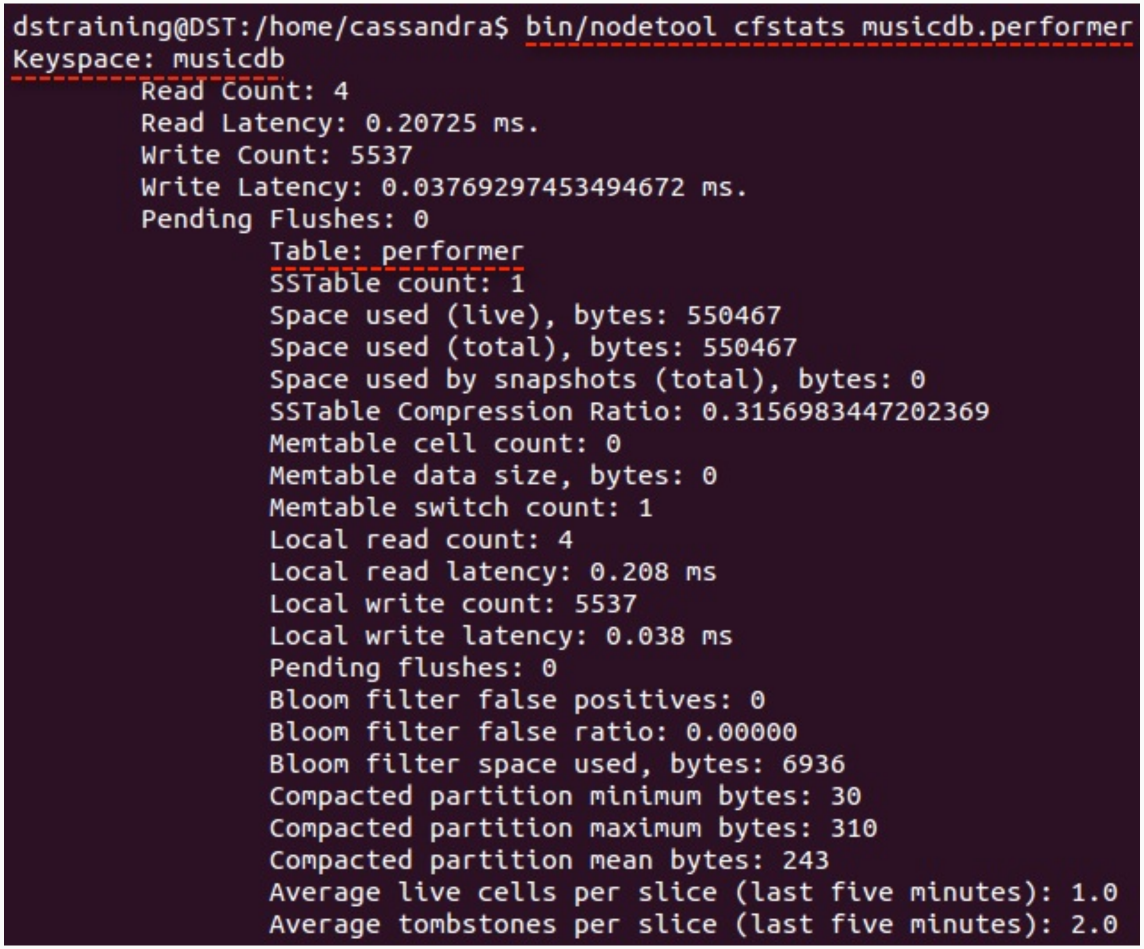
```
cqlsh:musicdb> TRACING ON;
Now tracing requests.
cqlsh:musicdb> SELECT * FROM performer WHERE name = 'Sheryl Crow';

 name        | born | country       | died | founded | style | type
-------------+------+---------------+------+---------+-------+--------
 Sheryl Crow | 1962 | United States | null |    null |  Rock | artist

(1 rows)


Tracing session: 291832f0-3e13-11e4-898b-17914c10dbe5

 activity                                                                                        | timestamp                  | source    | source_elapsed
-------------------------------------------------------------------------------------------------+----------------------------+-----------+---------------
                                                                              Execute CQL3 query | 2014-09-16 19:34:34.271000 | 127.0.0.1 |              0
   Parsing SELECT * FROM performer WHERE name = 'Sheryl Crow' LIMIT 10000; [SharedPool-Worker-1] | 2014-09-16 19:34:34.271000 | 127.0.0.1 |             86
                                                       Preparing statement [SharedPool-Worker-1] | 2014-09-16 19:34:34.271000 | 127.0.0.1 |            180
                             Executing single-partition query on performer [SharedPool-Worker-2] | 2014-09-16 19:34:34.271000 | 127.0.0.1 |            595
                                              Acquiring sstable references [SharedPool-Worker-2] | 2014-09-16 19:34:34.271000 | 127.0.0.1 |            614
                                               Merging memtable tombstones [SharedPool-Worker-2] | 2014-09-16 19:34:34.271000 | 127.0.0.1 |            656
                                               Key cache hit for sstable 1 [SharedPool-Worker-2] | 2014-09-16 19:34:34.272000 | 127.0.0.1 |            817
                               Seeking to partition beginning in data file [SharedPool-Worker-2] | 2014-09-16 19:34:34.272000 | 127.0.0.1 |            831
 Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones [SharedPool-Worker-2] | 2014-09-16 19:34:34.274000 | 127.0.0.1 |           2705
                                Merging data from memtables and 1 sstables [SharedPool-Worker-2] | 2014-09-16 19:34:34.274000 | 127.0.0.1 |           2729
                                        Read 1 live and 2 tombstoned cells [SharedPool-Worker-2] | 2014-09-16 19:34:34.274000 | 127.0.0.1 |           2806
                                                                                Request complete | 2014-09-16 19:34:34.274413 | 127.0.0.1 |           3413


cqlsh:musicdb> TRACING OFF;
Disabled tracing.
```

Exercise 4: Read data and examine its tracing output



How do you obtain performance data using cfstats?

- *nodetool cfstats* command provides statistics for a specified table ("column family"), including
  - read latency
  - write latency
  - SSTable count
  - space used



```
dstraining@DST:/home/cassandra$ bin/nodetool cfstats musicdb.performer
Keyspace: musicdb
        Read Count: 4
        Read Latency: 0.20725 ms.
        Write Count: 5537
        Write Latency: 0.03769297453494672 ms.
        Pending Flushes: 0
                Table: performer
                SSTable count: 1
                Space used (live), bytes: 550467
                Space used (total), bytes: 550467
                Space used by snapshots (total), bytes: 0
                SSTable Compression Ratio: 0.3156983447202369
                Memtable cell count: 0
                Memtable data size, bytes: 0
                Memtable switch count: 1
                Local read count: 4
                Local read latency: 0.208 ms
                Local write count: 5537
                Local write latency: 0.038 ms
                Pending flushes: 0
                Bloom filter false positives: 0
                Bloom filter false ratio: 0.00000
                Bloom filter space used, bytes: 6936
                Compacted partition minimum bytes: 30
                Compacted partition maximum bytes: 310
                Compacted partition mean bytes: 243
                Average live cells per slice (last five minutes): 1.0
                Average tombstones per slice (last five minutes): 2.0
```

```
bin/nodetool -h [host] -p [port] cfstats <keyspace>.<table>
```

Exercise 5: Use *cfstats* to measure performance

Summary

- If a node responds slowly to a request, the request is forwarded to another replica node
- The *row cache* is an optional mechanism to cache recently requested partitions
- Each SSTable has a *Bloom filter*, *partition summary*, and *partition index*
- A *Bloom filter* reduces disk seeks by ruling out SSTables which do not contain a partition
- The *key cache*, shared by all SSTables for a MemTable, caches the location of recently requested partition keys
- A *partition summary* is an evenly distributed in-memory sampling from a partition index, used to reduce index seek time
- A *partition index* provides the specific data file offset location for each partition key in an SSTable

Summary

- A *partition key* found in the *partition index* is added to the *key cache*
- Cassandra *merges* the most recent columns of data from a MemTable, and its SSTables, for a given request
- If *row cache* is in use, the merged CQL row for a partition key is updated when the requested row is returned
- Row and key caching is controlled using the *caching* table property
- *cqlsh* enables command line CQL queries and shell commands
- *cassandra-cli* enables command line Thrift API commands
- The *nodetool cfstats* command provides statistical information about a keyspace and table

Review Questions

- What benefit do Bloom filters provide to the read process?
- Is the partition summary read for partition keys in the key cache?
- What is the relationship between the partition summary and index?
- How many key caches are maintained for a MemTable?