Oct 20, 2014

Learning Objectives

- **Understand tombstones and deletion**
- Understand compaction and its necessity
- Choose and implement compaction strategies

What are tombstones and how are they used?

DATASTAX

- Deleted columns are not immediately removed, just marked
  - immediate removal would require a time-wasting seek
- When a CQL query deletes a partition column, or its TTL is found to be expired during a read
  1. a tombstone (deletion marker) is applied to this column in its Memtable
  2. subsequent queries treat this column as deleted
  3. at the next Memtable flush, the tombstone passes to the new SSTable
  4. at each compaction, tombstoned columns older than *gc_grace_seconds* are evicted from the newly compacted SSTables

What are tombstones and how are they used?

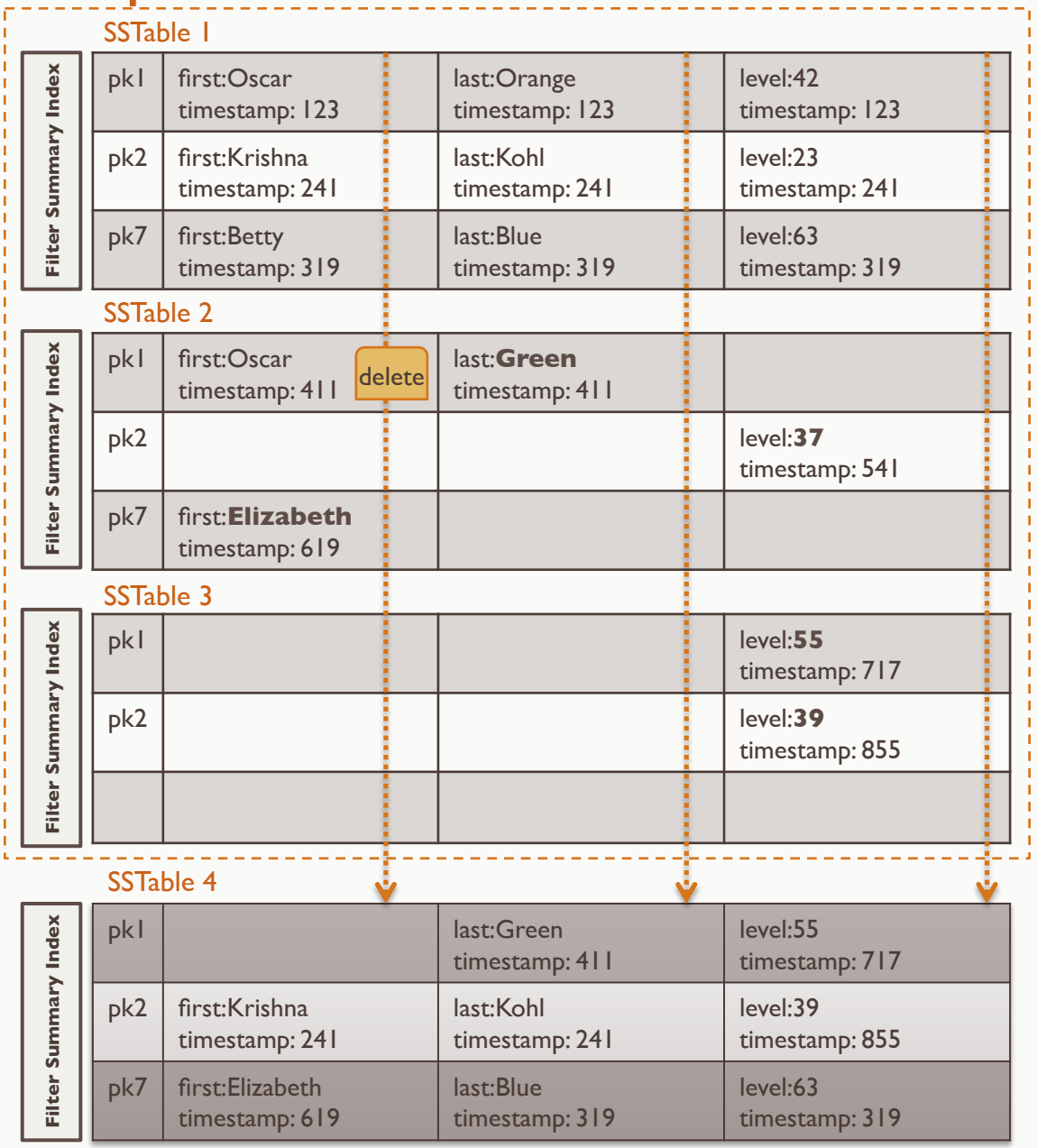
- gc_grace_seconds – table property defining how long tombstones will be retained before eviction in the next compaction (default: 864000, 10 days)
  - "zombie columns" – if a node fails before a replicated tombstone arrives, then is restored more than *gc_grace_seconds* later, the otherwise deleted column will reappear, as all other nodes will have evicted the tombstone
  - The cure: use *nodetool repair* when restoring failed nodes, to ensure all its partitions are consistent, including any pending deletions

Learning Objectives

- Understand tombstones and deletion
- **Understand compaction and its necessity**
- Choose and implement compaction strategies

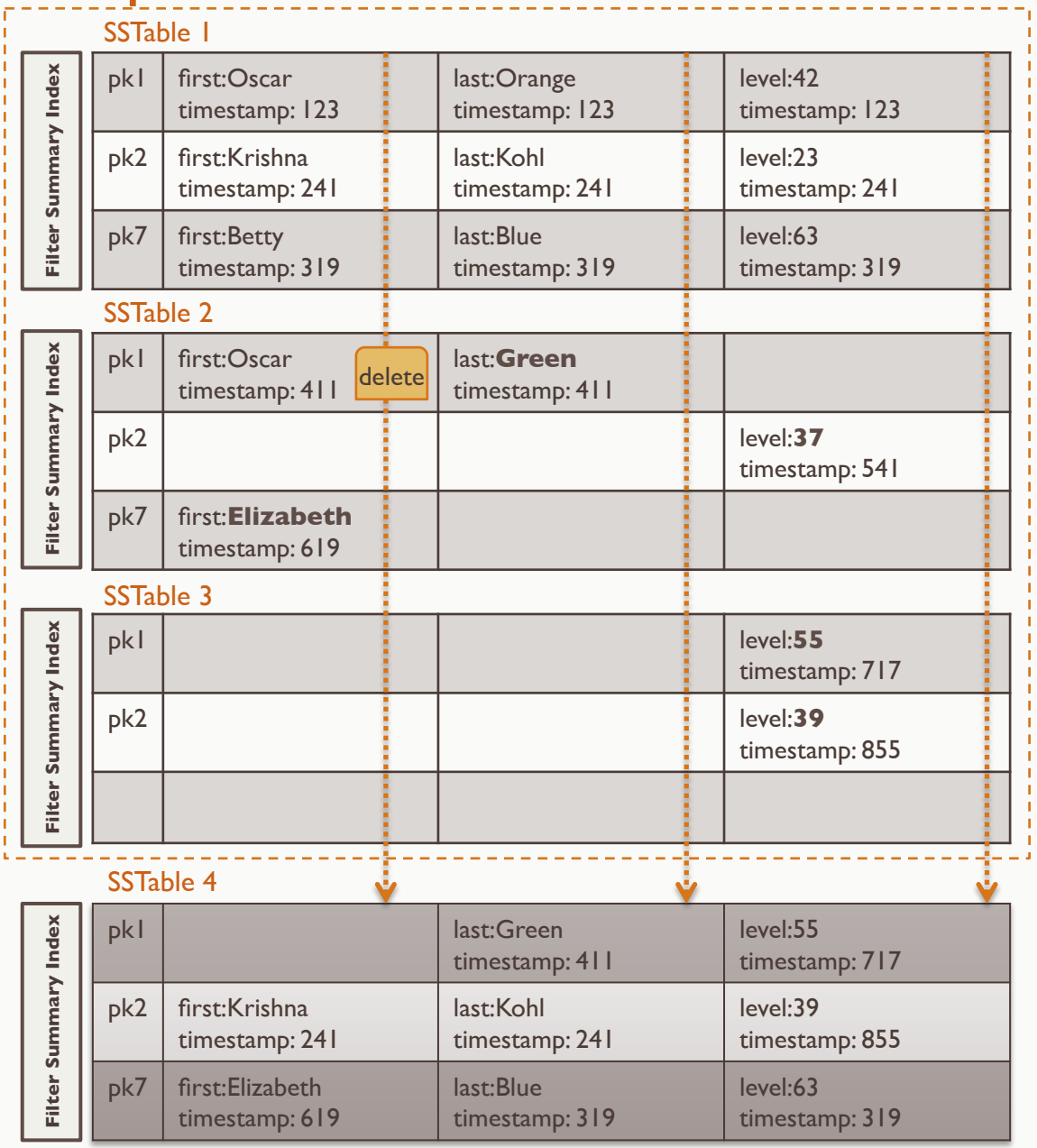
What is storage engine compaction?

- Critical, periodic SSTable maintenance process
  - merges most recent partition keys and columns
  - evicts deleted and TTL-expired partition columns
  - creates new SSTable
  - rebuilds partition index and partition summary
  - deletes the old SSTables



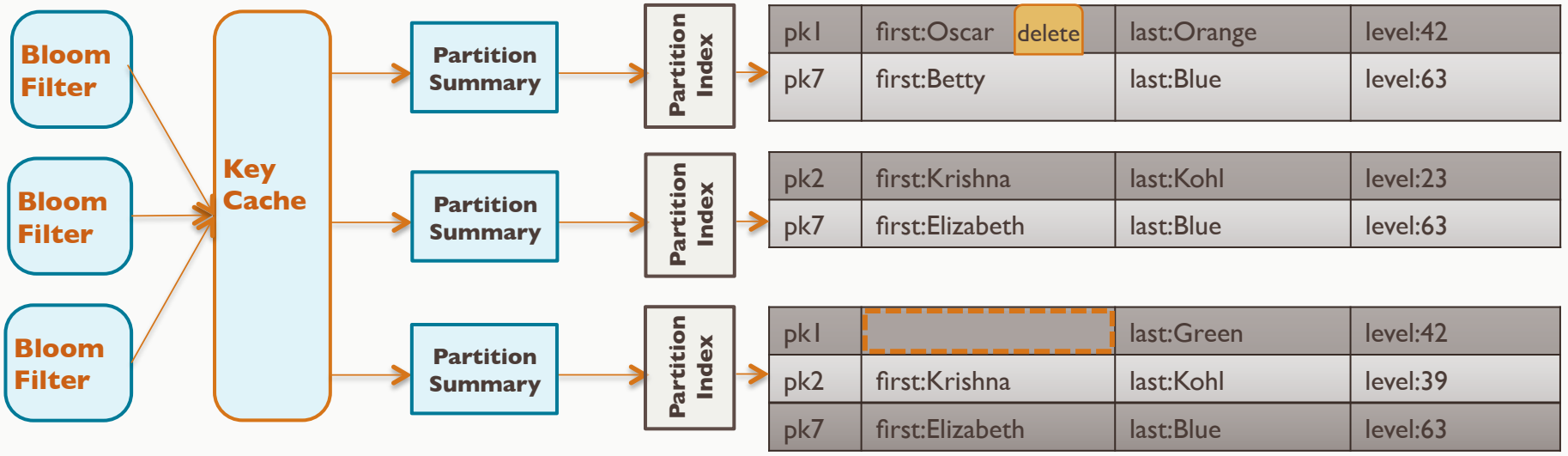
What is storage engine compaction?

- Efficient because
  - SSTables are inherently sorted by partition key
  - no random I/O required
- Necessary because
  - SSTables are immutable, so updates tend to fragment data over time
  - deletes are writes and must be periodically cleared



How does compaction affect reads and disk space?

- During compaction
  - disk I/O and utilization increase
  - off-cache read performance may be impacted
- After compaction
  - read performance increases as less SSTables are read for off-cache reads
  - disk utilization drops as old SSTables are deleted
- Performance tuning is discussed in detail in the *Apache Cassandra: Operations and Performance Tuning* course



Demo 1: Open and show SSTables with TTLs and Deletes

Learning Objectives

- Understand tombstones and deletion
- Understand compaction and its necessity
- **Choose and implement compaction strategies**

What compaction strategies are available?

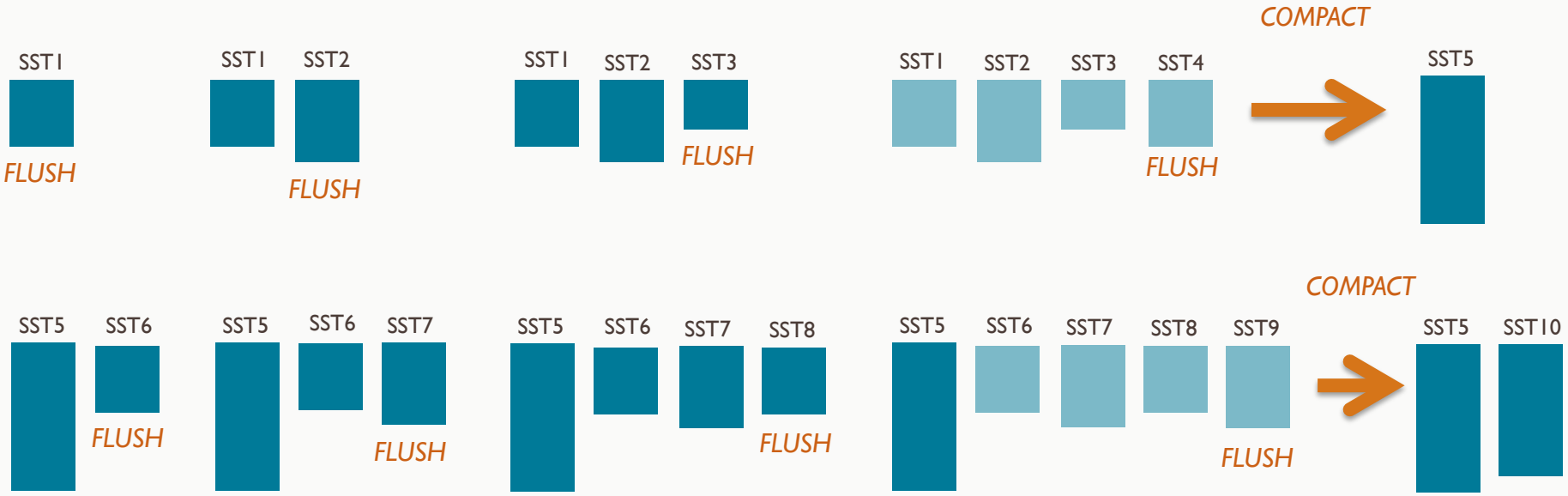
DATASTAX

- compaction – table property defining when and how compaction occurs for this table
- Available compaction strategies are
  - Size-Tiered (default) – compaction triggered as number of SSTables reach a threshold
  - Leveled – uniform-size SSTables organized and compacted by successive levels
  - Date-Tiered – data written within a certain time window is saved together

```
CREATE TABLE performer (
  first text PRIMARY KEY,
  last text,
  level text
)
WITH compaction = {'class' : '<strategy>', <params>};
```

What is size-tiered compaction, and when is it used?

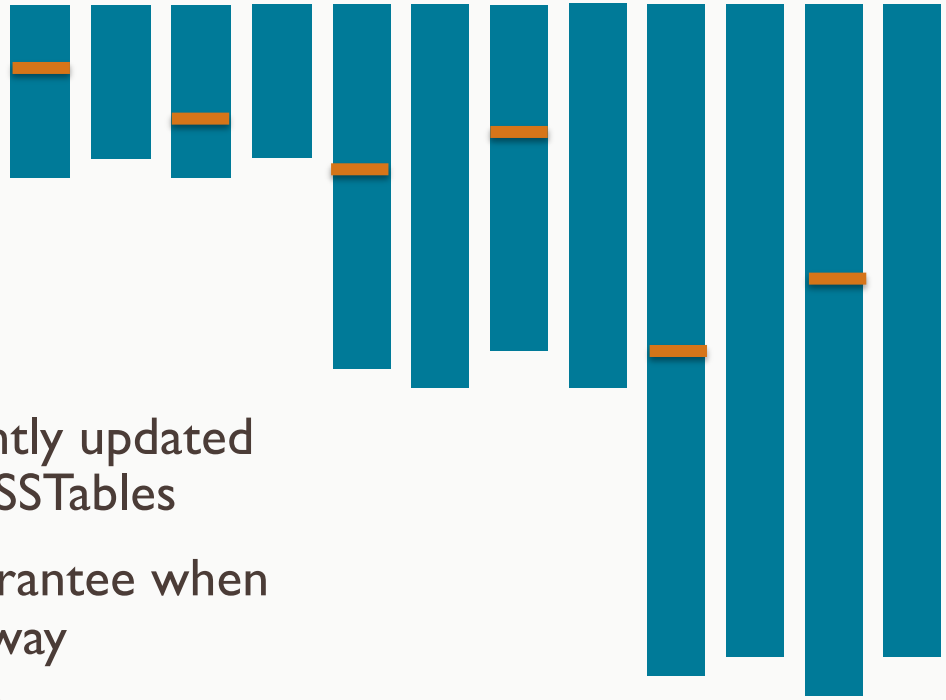
- Compacts set number of similarly sized SSTables to a larger SSTable




```
ALTER TABLE performer
WITH compaction =
 {'class' : 'SizeTieredCompactionStrategy', <params>};
```

What is size-tiered compaction, and when is it used?

- Some implications of this compaction approach
  - fast to complete each compaction because relatively few SSTables are compacted at once
  - successively larger SSTable sets
  - inconsistent read latency, as frequently updated partitions may spread across many SSTables
  - may waste space, as there is no guarantee when obsolete columns will be merged away
  - requires significant disk space (2 x free disk space as largest CQL table)
- May be preferable for write-heavy and time-series data applications



What is full compaction, and when is it used?

DATASTAX

- A maintenance operation, not a strategy
  - usable only on tables set to *SizeTieredCompactionStrategy*
  - compacts all or specified SSTables into a single table
  - also referred to as "major compaction"
  - consumes considerable disk I/O and disk space

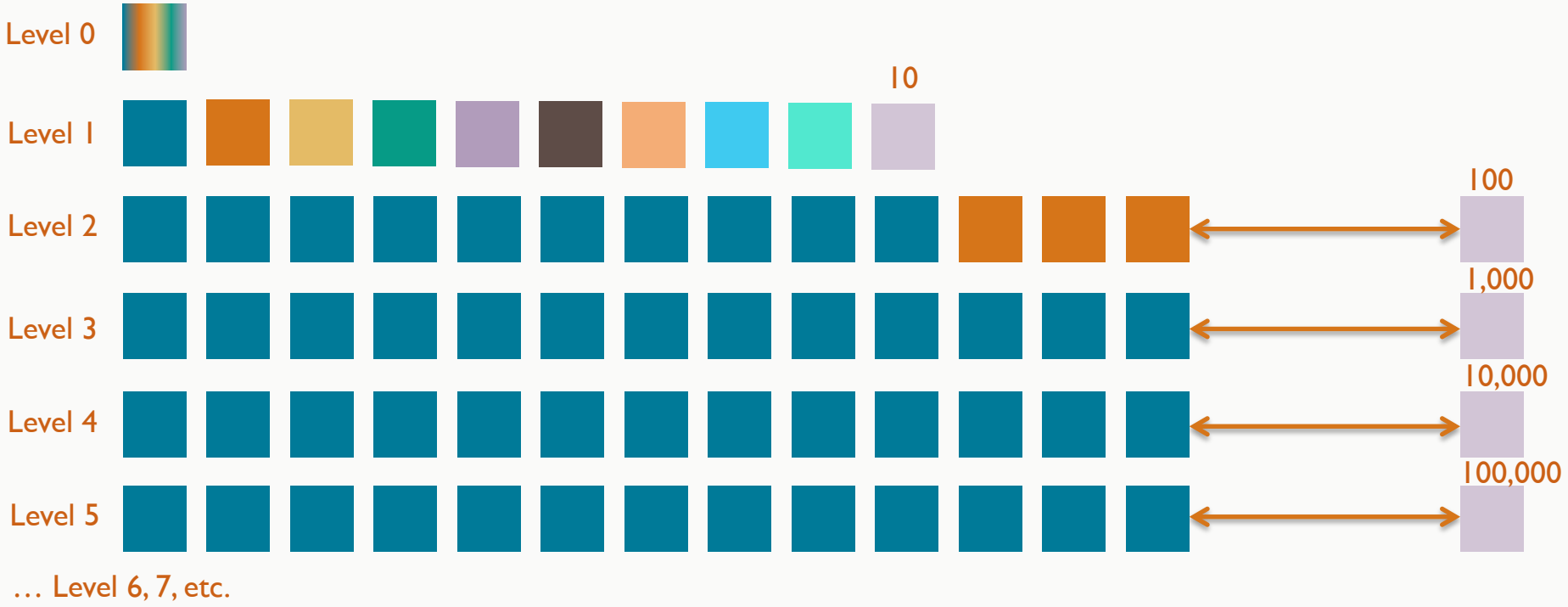
```
bin/nodetool compact [keyspace] [table]
```

  - if no table is specified, all tables are compacted
- Not recommended for production use

Demo 2: Show *size-tiered* compaction in use

What is leveled compaction, and when is it used?

- Compacts small, fixed-size SSTables to successively 10x larger levels
  - when using leveled compaction, each SSTable is 160mb by default
    - configured by the sstable_size_in_mb table attribute
  - partition keys sorted and grouped in each level's SSTables (similar to nodes)
    - as a result, each read touches, at most, one SSTable per level



What is leveled compaction, and when is it used?

DATASTAX

- Some implications of this compaction approach
  - assuming near-uniform row size within each SSTable, 90% of all reads will be satisfied from a single SSTable
    - even massive rows touch, at most, one SSTable per level
    - 10 terabytes of (theoretical) data would occupy 7 levels
  - no more than 10% of space wasted on obsolete rows
  - requires only 10 x *sstable_size_in_mb* free disk space to run compaction
    - but, requires roughly double the I/O of size-tiered compaction
- May be preferable for read-heavy applications

```
ALTER TABLE performer
WITH compaction =
 {'class' : 'LeveledCompactionStrategy', <params>};
```

What is date-tiered compaction, and when is it used?

DATASTAX

- Ideally suited for time-series data
- Implications of this compaction approach
  - compaction is triggered when a number of SSTables meet the min_threshold (default: 4) that were written around the same time period.
  - the table sub-property *base_time_seconds* (default: 1 hour) determines the size of the first time window. SSTables written within the time window can be selected for compaction.
  - worst case compaction still temporarily doubles disk space usage of the table
  - when used with TTL data, this strategy compacts data that expire around the same time together, and drops the SSTable after expiration without the need for a compaction

```
ALTER TABLE performer
WITH compaction =
 {'class' : 'DateTieredCompactionStrategy', <params>};
```

What compaction strategy should you use?

DATASTAX

- It depends …

SizeTieredCompaction – Pros	SizeTieredCompaction – Cons
• relatively low overhead per compaction due to small number of SSTables involved • *may be optimal for write heavy applications*	• up to 2 x largest table free disk space needed for compaction
LeveledCompaction – Pros	**LeveledCompaction – Cons**
• reduces total potential SSTables to be touched for each read • less disk space needed for compaction • better tombstone eviction • *may be optimal for read heavy applications*	• compaction more frequent – higher throughput required – so may be an issue for older, slower hardware • no advantage if rows are write-once (e.g., time-series data)
DateTieredCompaction – Pros	**DateTieredCompaction – Cons**
• best for time-series data applications	• strategy doesn't provide any benefit if queries or workload is not time-based

Exercise 1: Configure and run other compaction strategies

Summary

- Deleted columns are marked by tombstones, not immediately removed
- *gc_grace_seconds* sets how long tombstones survive before eviction during compaction
- Nodes down longer than *gc_grace_seconds* should have their data rebuilt from the other replicas
- Compaction is an ongoing SSTable maintenance process which
  - merges most recent columns
  - evicts tombstones
  - rebuilds partition indexes and summaries
  - creates a new SSTable
  - deletes old SSTables

Summary

- Necessary because updates fragment partitions over time and tombstones must be evicted
- Efficient because SSTables are sorted so there is no random I/O
- Improves read speed and reduces disk utilization
- Tuned by *cassandra.yaml* settings including
  - *compaction_throughput_mb_per_sec*
- Compaction strategy is set by the *compaction* table property
- *Size-tiered* compacts a set number of SSTables to larger SSTables
- *Full* compaction can be triggered with *nodetool compact*
  - not recommended
- *Leveled* compacts like-sized SSTables into successively 10 x larger levels, and spreads partitions across levels
- *Date-tiered is ideal for time-series data*

Review Questions

- What are zombie columns, and how do you prevent them?
- How do SSTables change during compaction?
- What are some benefits of Size-tiered compaction?
- What are some benefits of Leveled compaction?
- When might you use *nodetool compact*?