Anatomy of Cassandra's Read Operation

Let’s explore Cassandra’s read path.

We'll cover the following

Let’s dig deeper into the components involved in Cassandra’s read path.

Caching#

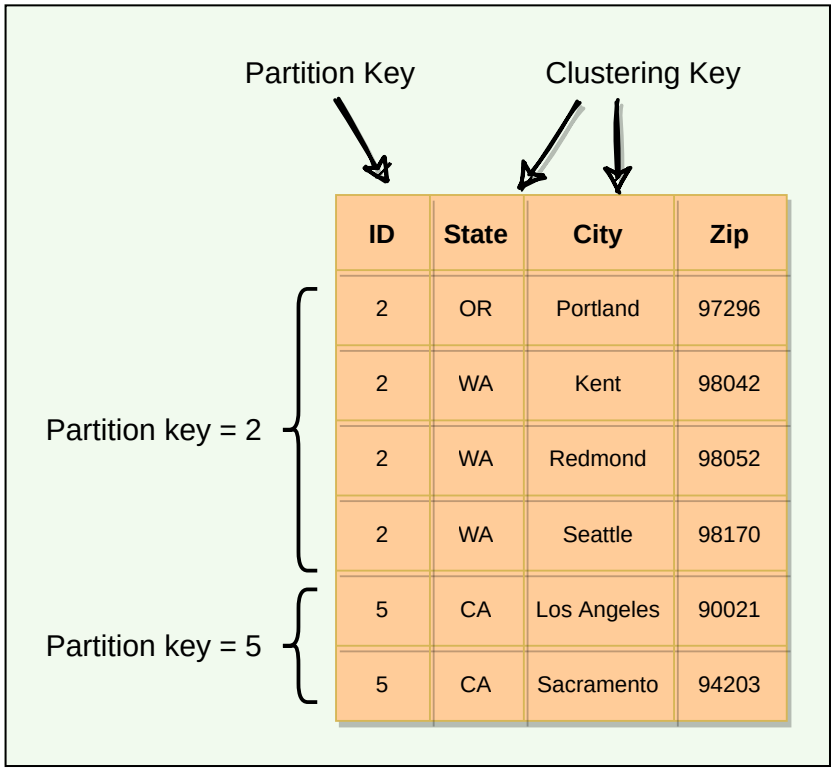
To boost read performance, Cassandra provides three optional forms of caching:

1. **Row cache:** The row cache, caches frequently read (or hot) rows. It stores a complete data row, which can be returned directly to the client if requested by a read operation. This can significantly speed up read access for frequently accessed rows, at the cost of more memory usage.

2. **Key cache:** Key cache stores a map of recently read partition keys to their SSTable offsets. This facilitates faster read access into SSTables stored on disk and improves the read performance. Key cache takes little memory compared to row cache (where the whole row is stored in memory) and provides a considerable improvement for read operations.
3. **Chunk cache:** Chunk cache is used to store uncompressed chunks of data read from SSTable files that are accessed frequently.

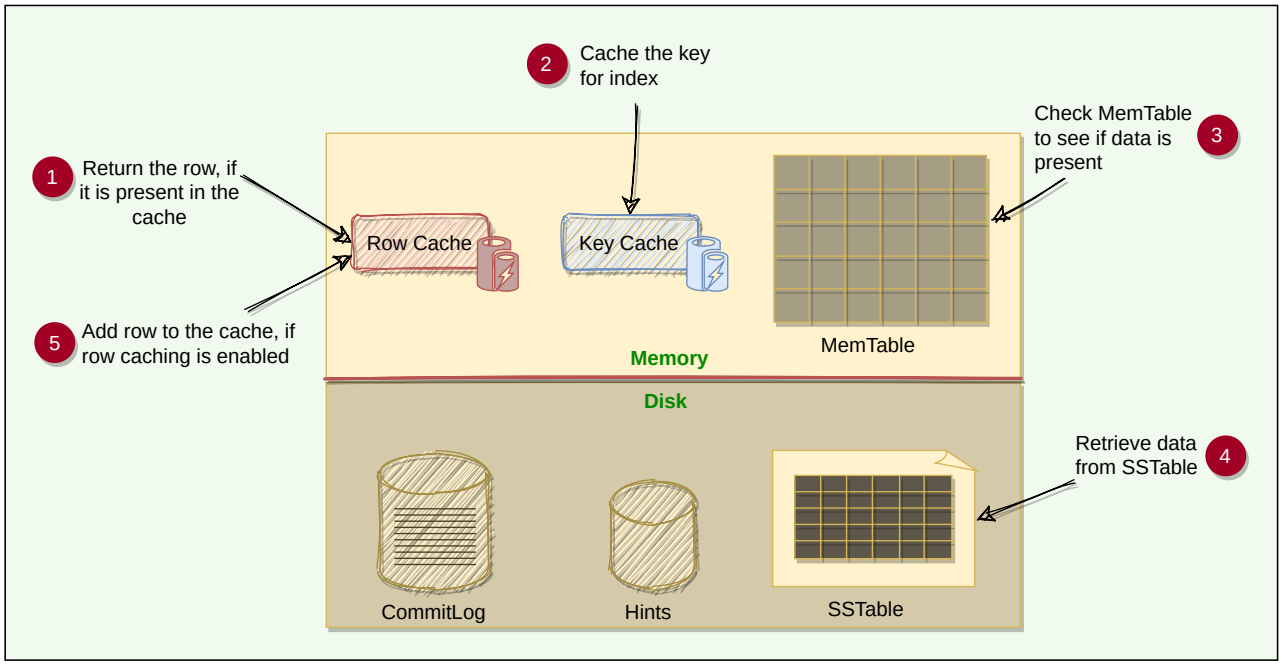
Reading from MemTable#

As we know, data is sorted by the partition key and the clustering columns. Let's take an example. Here we have two partitions of a table with partition keys '2' and '5'. The clustering columns are the state and city names. When a read request comes in, the node performs a binary search on the partition key to find the required partition and then return the row.



Reading data from MemTable

Here is the summary of Cassandra's read path:



Anatomy of Cassandra's read path

Reading from SSTable#

Bloom filters#

Each SStable has a Bloom filter associated with it, which tells if a particular key is present in it or not. Bloom filters are used to boost the performance of read operations. Bloom filters are very fast, non-deterministic algorithms for testing whether an element is a member of a set. They are non-deterministic because it is possible to get a false-positive read from a Bloom filter, but false-negative is not possible. Bloom filters work by mapping the values in a data set into a bit array and condensing a larger data set into a digest string using a hash function. The digest, by definition, uses a much smaller amount of memory than the original data would. The filters are stored in memory and are used to improve

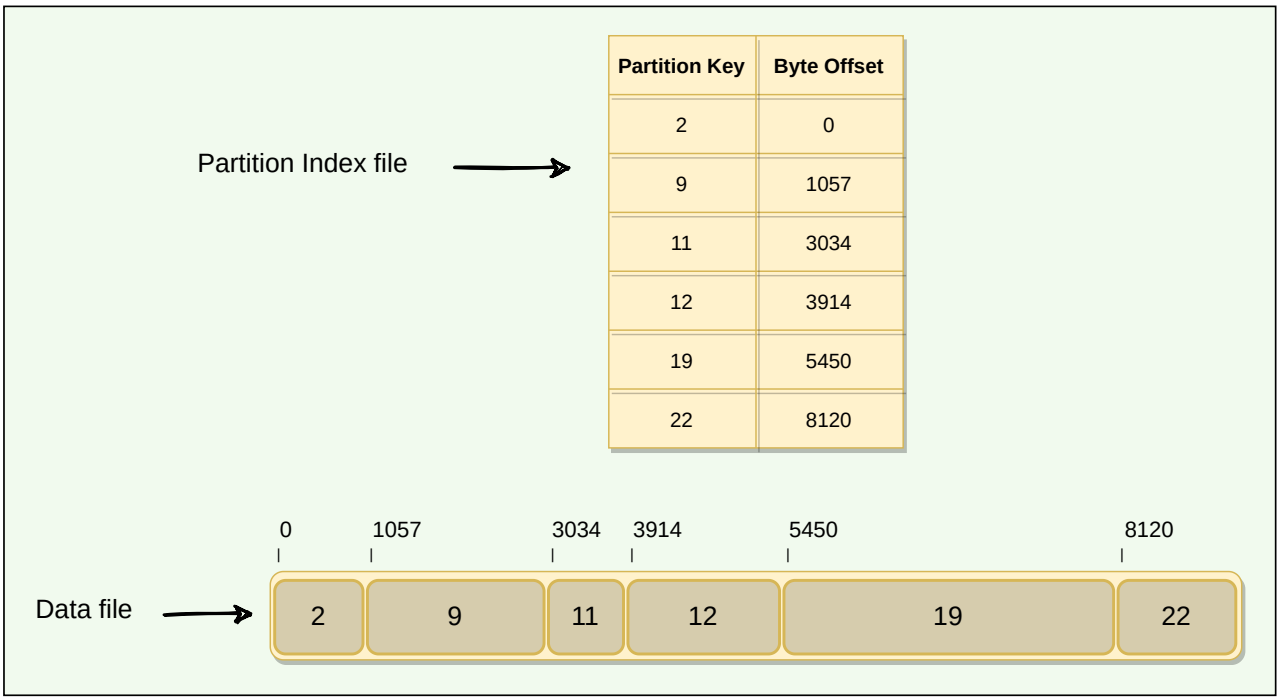
performance by reducing the need for disk access on key lookups. Disk access is typically much slower than memory access. So, in a way, a Bloom filter is a special kind of key cache.

Cassandra maintains a Bloom filter for each SSTable. When a query is performed, the Bloom filter is checked first before accessing the disk. Because false negatives are not possible, if the filter indicates that the element does not exist in the set, it certainly does not; but if the filter thinks that the element is in the set, the disk is accessed to make sure.

How are SSTables stored on the disk?#

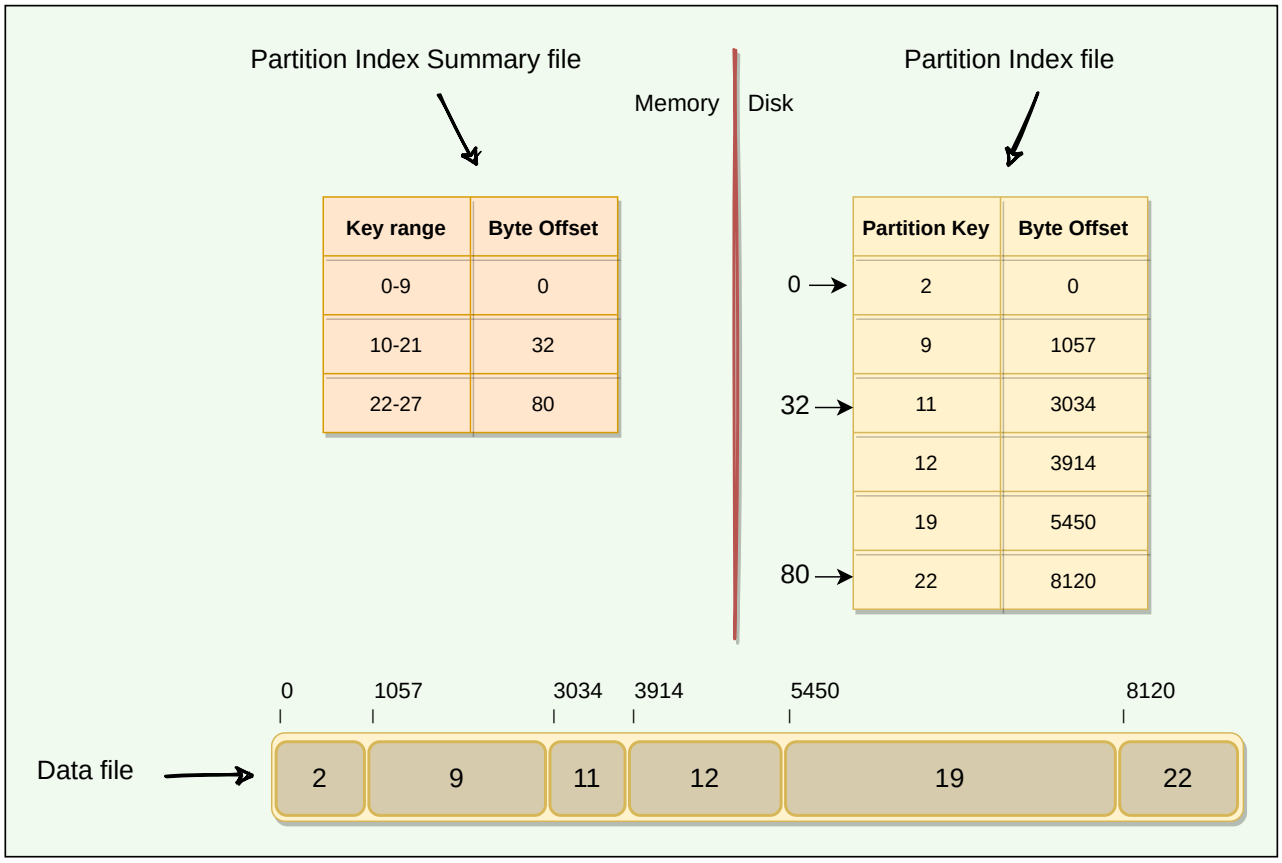
Each SSTable consists of two files:

1. **Data File:** Actual data is stored in a data file. It has partitions and rows associated with those partitions. The partitions are in sorted order.
2. **Partition Index file:** Stored on disk, partition index file stores the sorted partition keys mapped to their SSTable offsets. It enables locating a partition exactly in an SSTable rather than scanning data.



Partition index summary file#

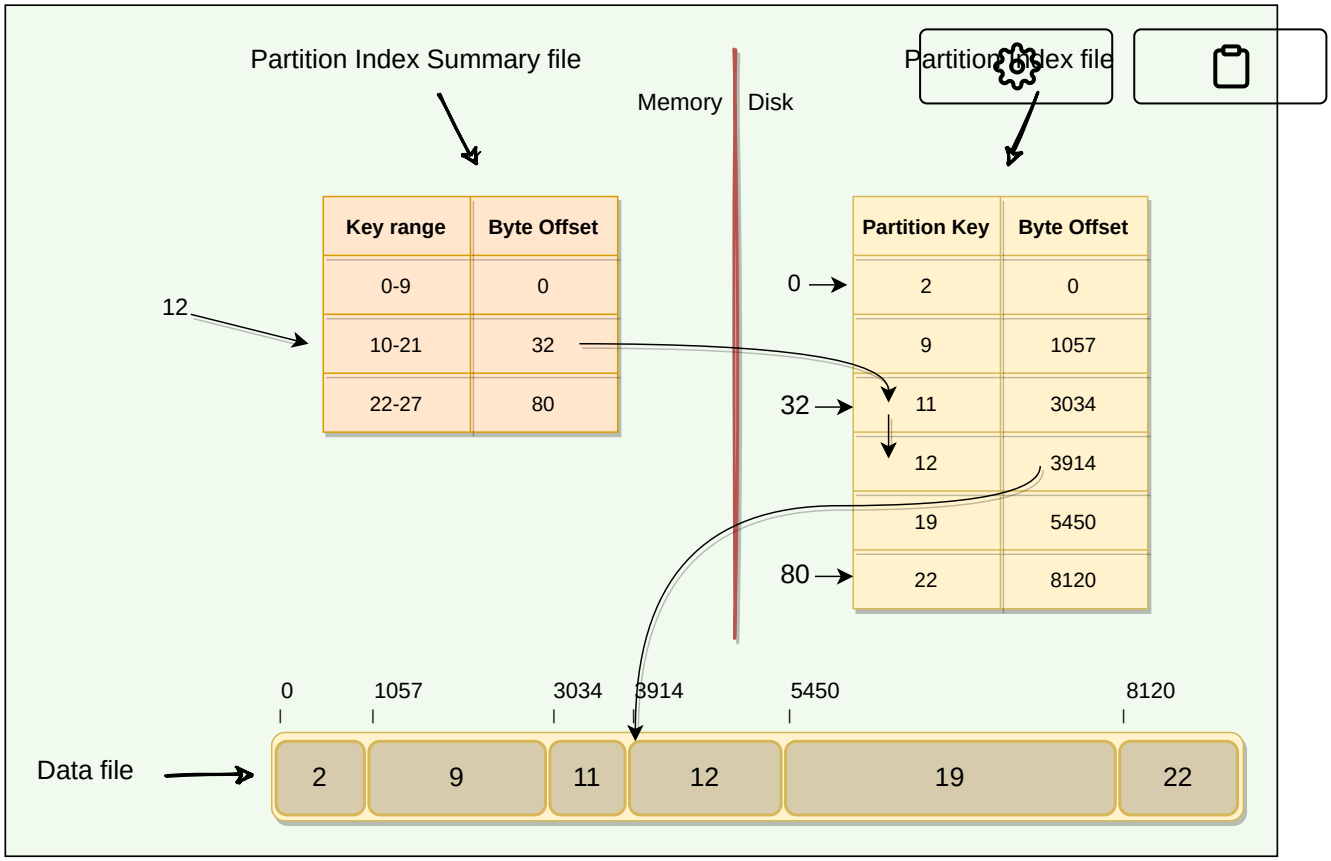
Stored in memory, the Partition Index Summary file stores the summary of the Partition Index file. This is done for performance improvement.



Reading from partition index summary file

If we want to read data for `key=12`, here are the steps we need to follow (also shown in the figure below):

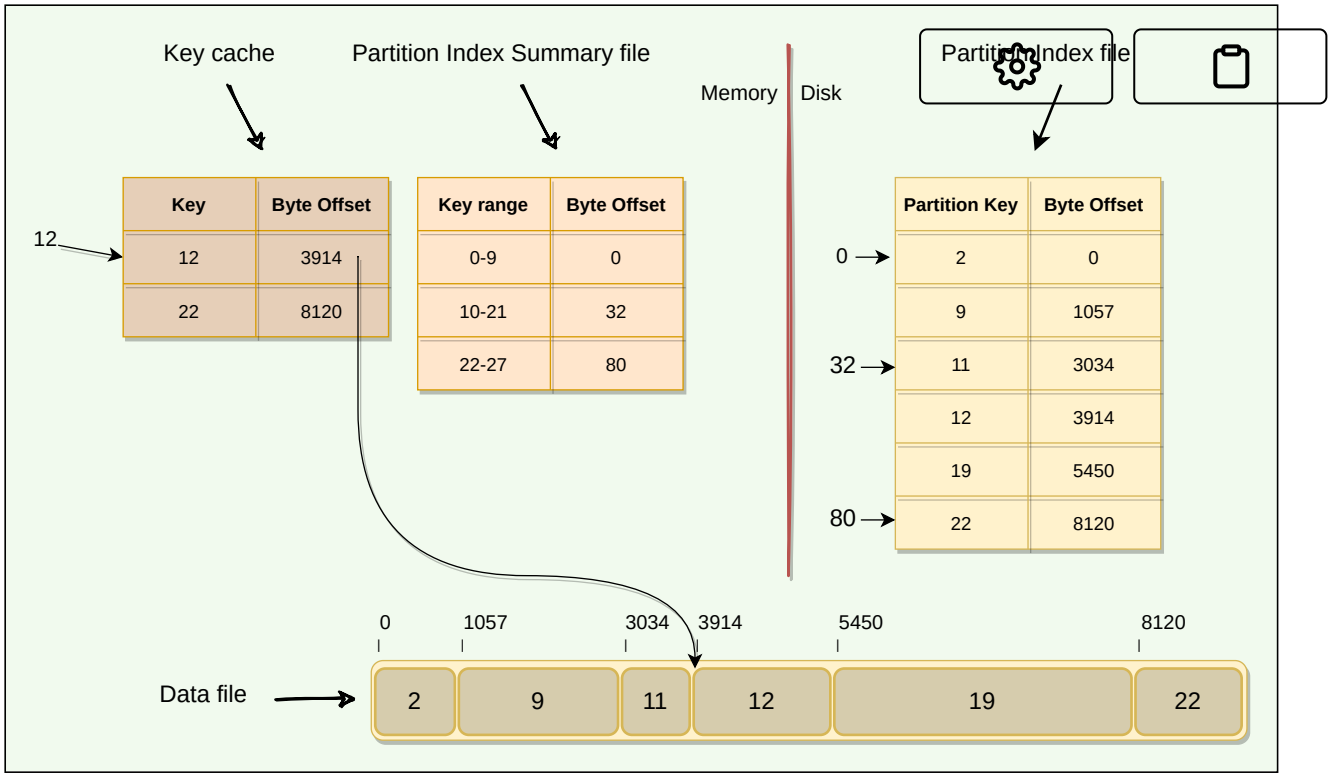
1. In the Partition Index Summary file, find the key range in which the `key=12` lies. This will give us offset (=32) into the Partition Index file.
2. Jump to offset 32 in the Partition Index file to search for the offset of `key=12`. This will give us offset (=3914) into the SSTable file.
3. Jump to SSTable at offset 3914 to read the data for `key=12`



Reading from partition index summary file

Reading SSTable through key cache#

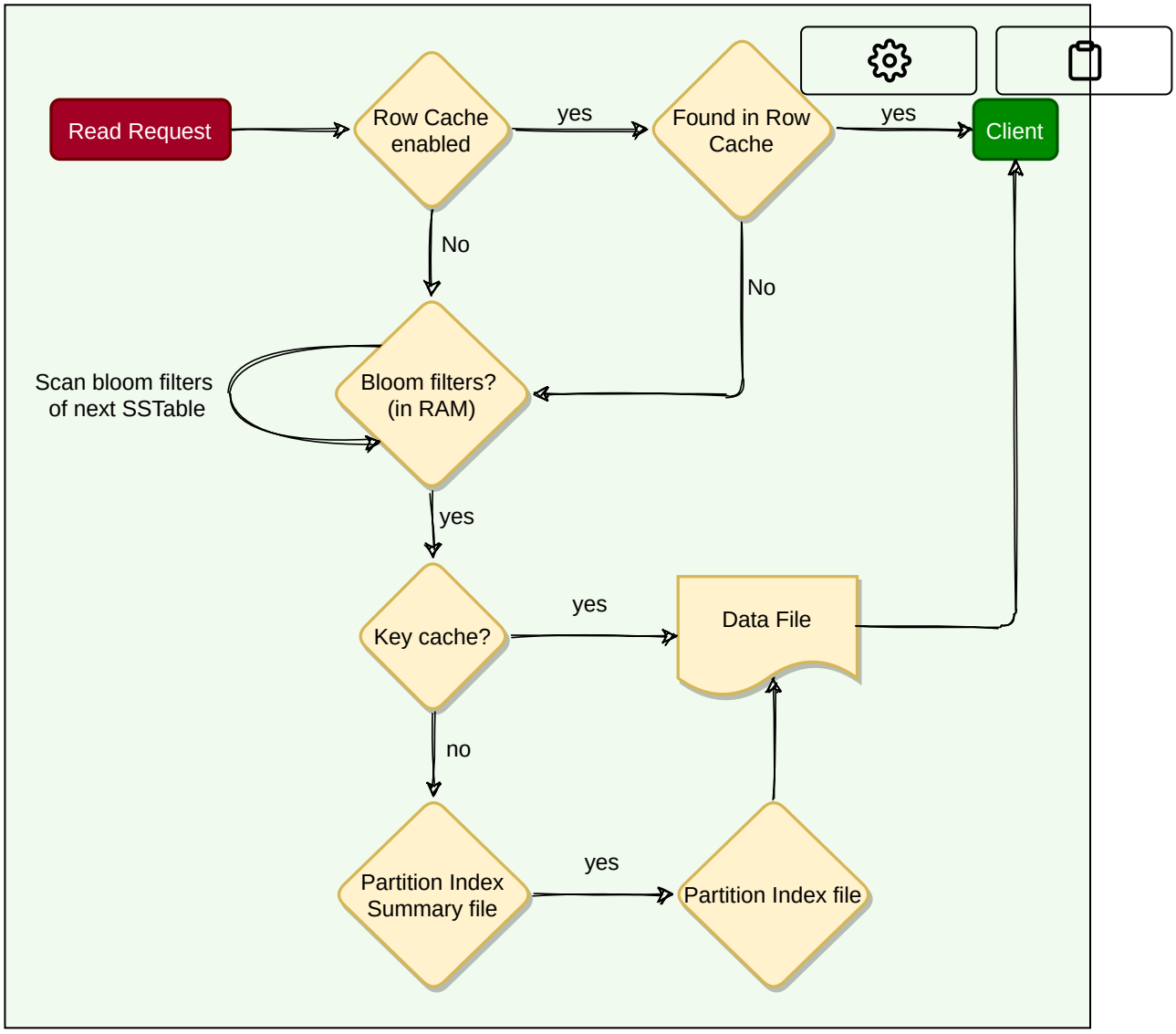
As the Key Cache stores a map of recently read partition keys to their SSTable offsets, it is the fastest way to find the required row in the SSTable.



Reading SSTable through key cache

If data is not present in MemTable, we have to look it up in SSTables or other data structures like partition index, etc. Here is the summary of Cassandra's read operation:

1. First, Cassandra checks if the row is present in the Row Cache. If present, the data is returned, and the request ends.
2. If the row is not present in the Row Cache, bloom filters are checked. If a bloom filter indicates that the data is present in an SSTable, Cassandra looks for the required partition in that SSTable.
3. The key cache is checked for the partition key presence. A cache hit provides an offset for the partition in SSTable. This offset is then used to retrieve the partition, and the request completes.
4. Cassandra continues to seek the partition in the partition summary and partition index. These structures also provide the partition offset in an SSTable which is then used to retrieve the partition and return. The caches are updated if present with the latest data read.



Cassandra's read operation workflow

← Back

Next →

Anatomy of Cassandra's Write Operat...

Compaction

Mark as Completed

Report an Issue