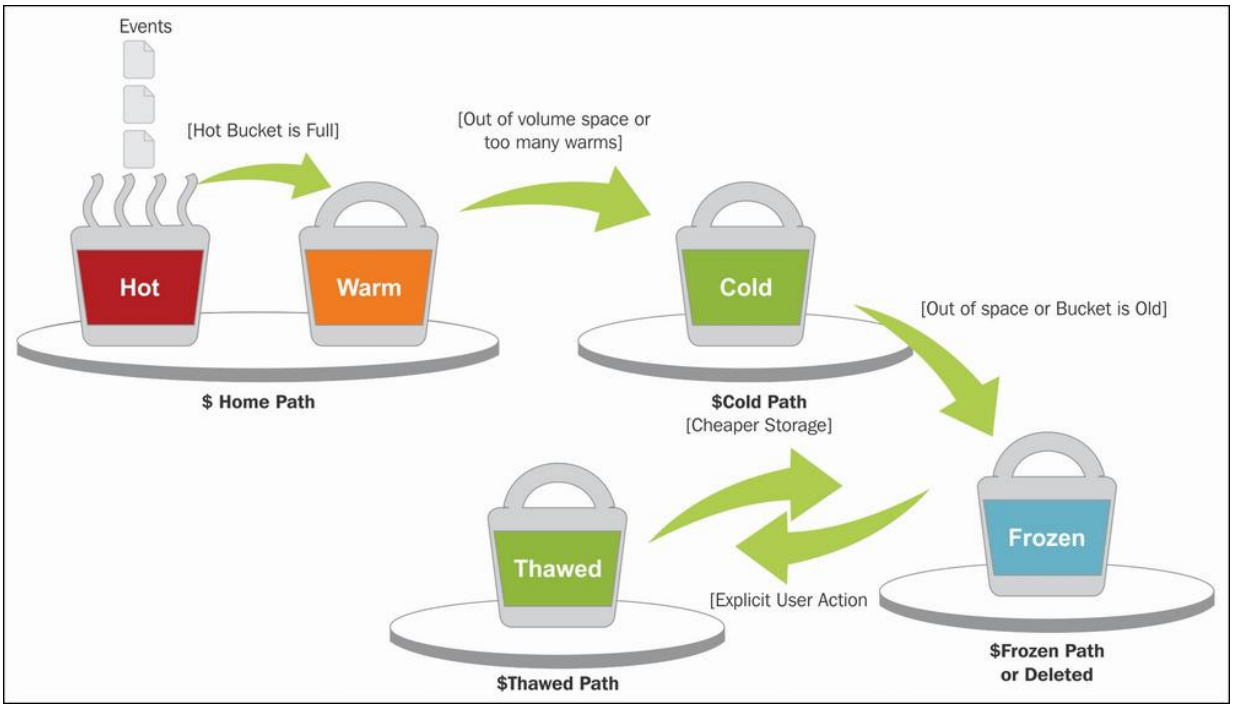
The following image shows the life cycle of Splunk buckets:



Think about our data coming in, to our Splunk environment. The first thing that's going to happen is, whether we're uploading it, or whether it's live streaming data, it's going to be indexed. That index is going to help us, one, be able to search it a little bit better. Splunk's going to put a timestamp on it, and it's going to do some other things to give us some meta data, so that we can simply search through that data a lot quicker in our Splunk environment.

The other thing it's going to do is, it's going to store that data so we can find it. It's going to store those in different buckets. Think of buckets just as the Splunk file system. Just like you have a file system, think of it in a Windows environment. I've got directories and subdirectories. Think of it in the Splunk environment as you have different buckets. I have buckets for different portions of my data. Those are all going to be with a timestamp. Right? Right. As it indexes, that's how Splunk decides where they're going to be in the bucket, and also, there's some other things you can do to decide how long data's going to sit, and sit in each one of your

buckets, but before we jump in and talk about that, let's make sure we understand what those buckets are.

The first bucket, or really the first two buckets, are going to be your hot and your warm bucket. Your hot and your warm bucket, this is where your most recent data is going to be. Your hot and your warm bucket, they're both going to sit in the same specific area. They're going to be put in there so that you have your data the most current, right? This is where Splunk really puts a lot of performance characteristics around where this data should live in these hot and warm buckets, and specifically really, if you think about it, your hot and your warm bucket, your warm bucket's going to contain some of your most recent data, but your hot bucket is going to be the one that's riding to the new data. As you set up your policies for how long your data is going to exist in your Splunk buckets, your hot and your warm bucket, let's say that arbitrarily you can get 10 events. It's a little bit more complicated than that, but let's just make it simple. Say that you can get 10 events in your buckets. Every time you get to 10 events, your hot bucket is going to become a warm bucket, and you have a brand-new bucket. Think of your hot bucket as where the newest files go, and your warm bucket is where your more recent is. It gives you a life cycle policy.

The third kind of bucket is a cold bucket. This is where our more older data, where our data is kind of aging off. This data can actually live, doesn't have to live with a hot and warm bucket. It can go to maybe a NAS device or some kind of object store where you can actually search on it. It still needs to have some requirements for how it's being searched, because it could be, if we're saying that we have 10 events in each one of our warm buckets, let's say that after a week, those 10 events age off to our cold bucket. Our cold bucket could hold from a week to maybe three months on our policy. That's where our data is going to exist, in that cold bucket. No new data's being written to it, and there's not as much performance requirements just because, probably not searching on it as frequently as we are on the newer events. They're

pulling out for our dashboard, then are stored in our hot and our warm buckets.

Then, we have our fourth bucket, which is going to be our frozen bucket. Think of this as really old, frozen data, hence the frozen bucket. Data that we're holding onto for compliance reasons or we just want to be able to go back and search on it at some point in the past, but this data is actually going off to some kind of long-term retention. The thing about it is, we want to search on it, I'll talk about it in just a second, but this data is not searchable in its current form of a frozen bucket. There's another process to be able to do that, and that'll include another bucket, but think of this as where you're aging off your data. This gives you an opportunity to get a better cost per terabyte for how you're storing the data, and get it out of your Splunk search, so better performance on your Splunk search as well, but still being able to hold on to that data, but think of this as, this is, if we're saying three months is what we're going to hold in our cold bucket. Think of it being more than three months that's going to exist in that frozen bucket.

In our last bucket, number five just talked about it, it's a thawed bucket. Our thawed bucket is how we get that frozen data back into a searchable state. You can go through and being able to thaw that bucket out. Think of it as taking some of the compression out of it, but also putting it in a better place to be able to store it. We talked about performance, and some of the other characteristics that you need to be able to search your data. In those thawed buckets is where you can start and go from that process. It's a full life cycle process. Go from hot, to warm, to cold, to frozen, and then when we want to see your data again, put it in that thawed bucket. That's all we have today. I hope you enjoyed this episode, where we talked about the five different kind of buckets in Splunk. If you have any questions, make sure you put them in the comments section here below. Reach out to me on Big Data Big Questions, and I'll do my best to answer your questions right here.

Data files

There are two key types of files in a bucket:

- The processed external data in compressed form (**rawdata**)
- Indexes that point to the rawdata (**index files**, also referred to as **tsidx files**)

Buckets contain a few other types of files as well, but these are the ones that are most important to understand.

Rawdata is not actually "raw" data, as the term might be defined by a dictionary. Rather, it consists of the external data after it has been processed into events. The processed data is stored in a compressed rawdata journal file. As a journal file, the rawdata file, in addition to containing the **event data**, contains all information necessary to generate the associated index files, if they are missing.

All bucket copies, both **searchable** and **non-searchable**, contain rawdata files. Searchable copies also contain index files.

When a peer node receives a block of data from a forwarder, it processes the data and adds it to the rawdata file in its local hot bucket. It also indexes it, creating the associated index files. In addition, it streams copies of just the processed rawdata to each of its target peers, which then adds it to the rawdata file in its own copy of the bucket. The rawdata in both the original and the replicated bucket copies are identical.

If the cluster has a search factor of 1, the target peers store only the rawdata in the bucket copies. They do not generate index files for the data. By not storing the index files on the target peers, you limit storage requirements. Because the rawdata is stored as a journal file, if the peer maintaining the original, fully indexed data goes down, one of the target peers can step in and generate the indexes from its copy of the rawdata.

If the cluster has a search factor greater than 1, some or all of the target peers also create index files for the data. For example, say you have a replication factor of 3 and a search factor of 2. In that case, the source peer streams its rawdata to two target peers. One of those peers then uses the rawdata to create index files, which it stores in its copy of the bucket. That way, there will be two searchable copies of the data (the original copy and the replicated copy with the index files). As described in "Search factor", this allows the cluster to recover more quickly in case of peer node failure. For more information on searchable bucket copies, see "Bucket searchability" later in this topic.

See these topics for more information on bucket files:

- For information on how bucket files get regenerated when a peer goes down, read "What happens when a peer node goes down".
- For information on the relative sizes of rawdata and index files, read "Storage considerations".

Bucket stages

As a bucket ages, it rolls through several stages:

hot
warm
cold
frozen

For detailed information about these stages, read "How the indexer stores indexes".

For the immediate discussion of cluster architecture, you just need a basic understanding of these bucket stages. A hot bucket is a bucket that's still being written to. When an indexer finishes writing to a hot bucket (for example, because the bucket reaches a maximum size), it rolls the bucket to warm and begins writing to a new hot bucket. Warm buckets are readable (for example, for searching) but the indexer does not write new data to them. Eventually, a bucket rolls to cold and then to frozen, at which point it gets archived or deleted.

There are a couple other details that are important to keep in mind:

- Hot/warm and cold buckets are stored in separately configurable locations.
- The filename of a warm or cold bucket includes the time range of the data in the bucket. For detailed information on bucket naming conventions, read "What the index directories look like".
- Searches occur across hot, warm, and cold buckets.
- The conditions that cause buckets to roll are configurable, as described in "Configure index storage".
- For storage hardware information, such as help on estimating storage requirements, read "Storage considerations".

Bucket searchability and primacy states

A copy of a bucket is either **searchable** or **non-searchable**. Since a cluster can maintain multiple searchable copies of a bucket, the cluster needs a way to identify which copy participates in a search. To handle this, clusters use the concept of **primacy**. A searchable bucket copy is either **primary** or non-primary.

A bucket copy is searchable if it contains index files in addition to the rawdata file. The peer receiving the external data indexes the rawdata and also sends copies of the rawdata to its peers. If the search factor is greater than 1, some or all of those peers will also generate index files for the buckets they're replicating. So, for example, if you have a replication factor of 3 and a search factor of 2 and the cluster is **complete**, the cluster will contain three copies of each bucket. All three copies will contain the rawdata file, and two of the copies (the copy on the source peer and one of the copies on the target peers) will also contain index files and therefore be searchable. The third copy will be non-searchable, but it can be made searchable if necessary. The main reason that a non-searchable copy gets made searchable is because a peer holding a searchable copy of the bucket goes down.

A primary copy of a bucket is the searchable copy that participates in a search. A single-site **valid** cluster has exactly one primary copy of each bucket. That way, one and only one copy of each bucket gets searched. If a node with primary copies goes down, searchable but non-primary copies on other nodes can immediately be designated as primary, thus allowing searches to continue without any need to first wait for new index files to be generated.

Note: In the case of a multisite cluster, a valid cluster is a cluster that has a set of primary copies for each site that supports search affinity. In search affinity, search heads perform searches across the peers on their local site. This requires that each site have its own set of primary buckets.

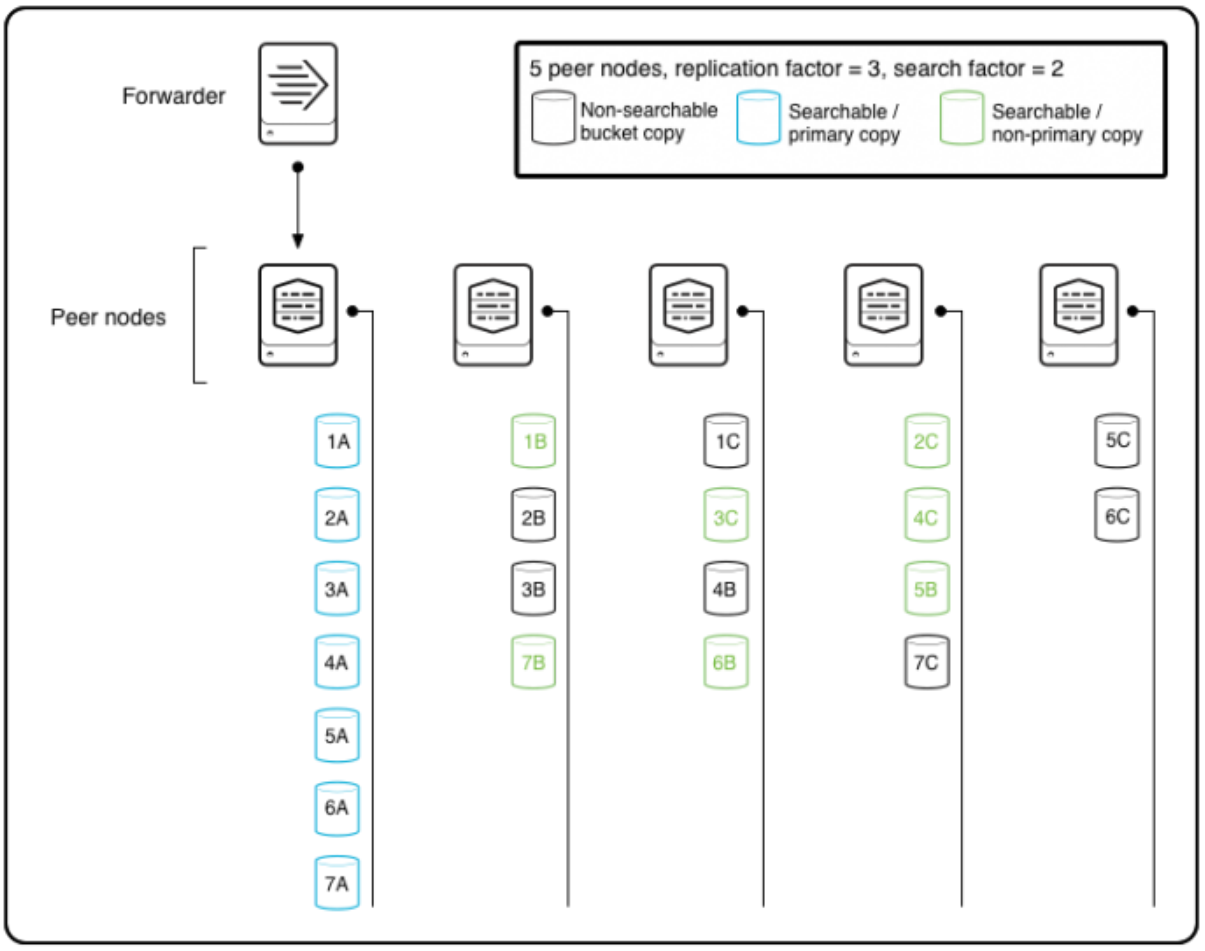
Initially, the copy of the bucket on the peer originating the data is the primary copy, but this can change over time. For example, if the peer goes down, the manager node reassigns primacy from any primary copies on the downed peer to corresponding searchable copies on remaining peers. For more information on this process, read "What happens when a peer node goes down".

Primacy reassignment also occurs when the manager rebalances the cluster, in an attempt to achieve a more even distribution of primary copies across the set of peers. Rebalancing occurs under these circumstances:

- A peer joins or rejoins a cluster.
- A manager node rejoins the cluster.
- You manually invoke the `rebalance_primaries` REST endpoint on the manager node.

See "Rebalance the indexer cluster primary buckets" for details.

The following diagram shows buckets spread across all the peers, as in the previous diagram. The cluster has a replication factor of 3 and a search factor of 2, which means that the cluster maintains two searchable copies of each bucket. Here, the copies of the buckets on the source peer are all primary (and therefore also searchable). The buckets' second searchable (but non-primary) copies are spread among most of the remaining peers in the cluster.



The set of primary bucket copies define a cluster's generation, as described in the next section.

Generations

A **generation** identifies which copies of a cluster's buckets are primary and therefore will participate in a search.

Note: The actual set of buckets that get searched also depends on other factors such as the search time range. This is true for any indexer, clustered or not.

The generation changes over time, as peers leave and join the cluster. When a peer goes down, its primary bucket copies get reassigned to other peers. The manager also reassigns primaries under certain other circumstances, in a process known as "cluster rebalancing".

Here is another way of defining a generation: A generation is a snapshot of a **valid** state of the cluster; "valid" in the sense that every bucket on the cluster has exactly one primary copy.

All peers that are currently registered with the manager participate in the current generation. When a peer joins or leaves the cluster, the manager creates a new generation.

Note: Since the process of reassigning primary to new bucket copies is not instantaneous, the cluster might quickly go through a number of generations while reassigning primacy due to an

event such as a downed peer, particularly in the case where numerous primaries were residing on the downed peer.

The generation is a cluster-wide attribute. Its value is the same across all sites in a multisite cluster.

How cluster nodes use the generation

Here is how the various cluster nodes use generation information:

- The manager creates each new generation, and assigns a **generation ID** to it. When necessary, it communicates the current generation ID to the peers and the search head. It also keeps track of the primary bucket copies for each generation and on which peers they are located.
- The peers keep track of which of their bucket copies are primary for each generation. The peers retain primary information across multiple generations.
- For each search, the search head uses the generation ID that it gets from the manager to determine which peers to search across.

When the generation changes

The generation changes under these circumstances:

- The manager comes online.
- A peer joins the cluster.
- A peer goes down, either intentionally (through the CLI `offline` command) or unintentionally (by crashing). When a peer goes down, the manager reassigns primacy from bucket copies on the downed node to searchable copies of the same buckets on the remaining nodes and creates a new generation.
- Whenever rebalancing of the primary copies occurs, such as when you manually hit the `rebalance_primaries` REST endpoint on the manager. For information on rebalancing, see "Rebalance the indexer cluster primary buckets".
- When the manager resolves certain bucket anomalies.

The manager does not create a new generation merely when a bucket rolls from hot to warm, thus causing a new hot bucket to get created (unless the bucket rolled for one of the reasons listed above). In that situation, the set of peers doesn't change. The search head only needs to know which peers are part of the generation; that is, which peers are currently participating in the cluster. It does not need to know which bucket copies on a particular peer are primary; the peer itself keeps track of that information.

How the generation is used in searches

The search heads poll the manager for the latest generation information at regular intervals. When the generation changes, the manager gives the search heads the new generation ID and a list of the peers that belong to that generation. Each search head, in turn, gives the peers the ID whenever it initiates a search. The peers use the ID to identify which of their buckets are primary for that search.

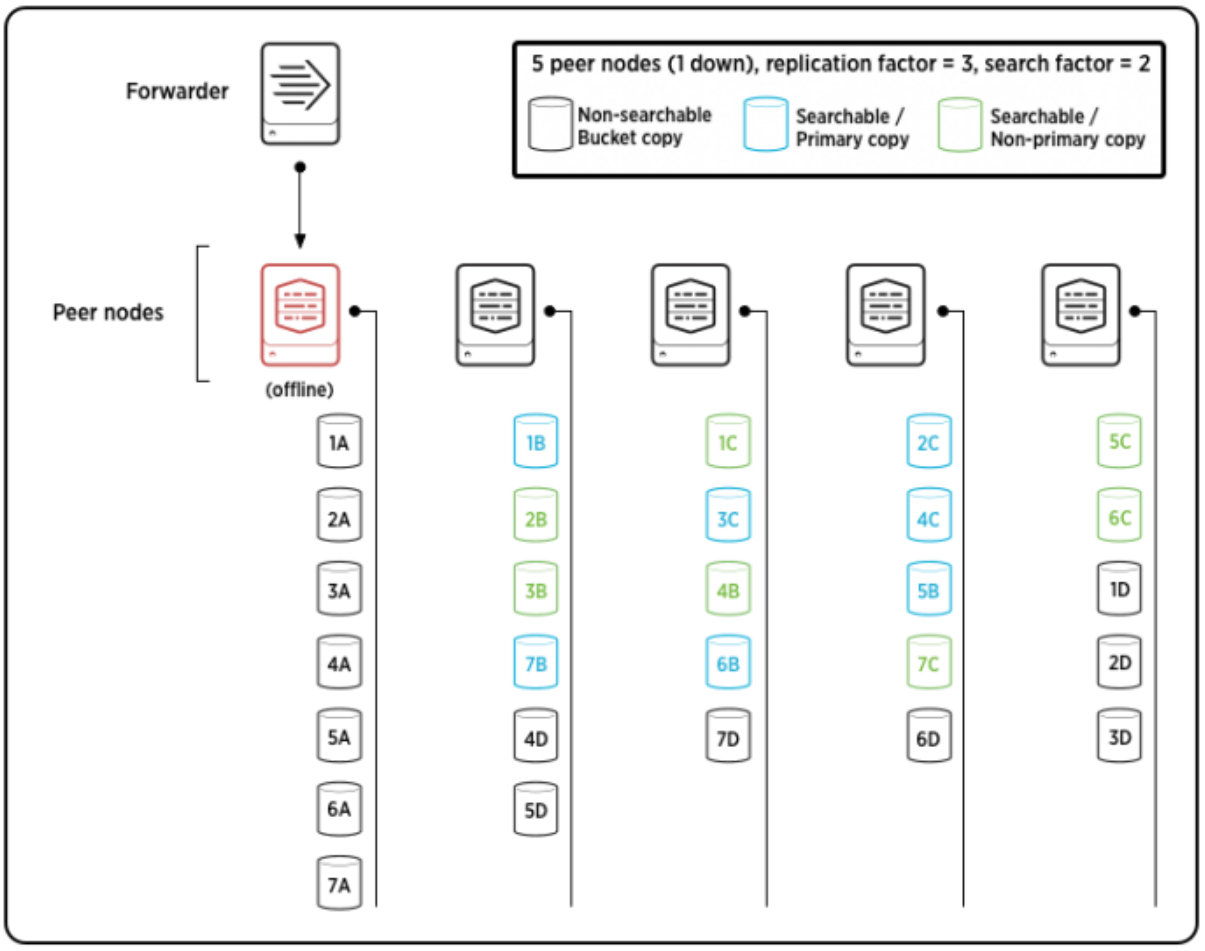
Usually, a search occurs over the most recent generation of primary bucket copies. In the case of long-running searches, however, it is possible that a search could be running across an earlier generation. This situation typically occurs because a peer went down in the middle of the search. This allows the long-running search to complete, even though some data might be missing due to the downed peer node. The alternative would be to start the search over again, which you can always do manually if necessary.

Why a downed peer causes the generation to change

The reason that a downed peer causes the manager to create a new generation is because, when a peer goes down, the manager reassigns the downed peer's primary copies to copies on other peers. A copy that was not primary for a previous generation becomes primary in the new generation. By knowing the generation ID associated with a search, a peer is able to determine which of its buckets are primary for that search.

For example, the diagram that follows shows the same simplified version of a cluster as earlier, after the source node holding all the primary copies has gone down and the manager has directed the remaining peers in fixing the buckets. First, the manager reassigned primacy to the remaining searchable copy of each bucket. Next, it directed the peers to make their non-searchable copies searchable, to make up for the missing set of searchable copies. Finally, it directed the replication of a new set of non-searchable copies (1D, 2D, etc.), spread among the remaining peers.

Even though the source node went down, the cluster was able to fully recover both its **complete** and **valid** states, with replication factor number (3) of total bucket copies, search factor number (2) of searchable bucket copies, and exactly one primary copy of each bucket. This represents a different generation from the previous diagram, because primary copies have moved to different peers.



Note: This diagram only shows the buckets originating from one of the peers. A more complete version of this diagram would show buckets originating from several peers as they have migrated around the cluster.

How the cluster handles frozen buckets

In the case of a standalone indexer, when a bucket rolls to frozen, the indexer deletes it from its `colddb` directory. Depending on its retirement policy, the indexer might copy it to an archive directory before deleting it. See "Archive indexed data."

In the case of an indexer cluster, when a peer freezes a copy of a bucket, it notifies the manager. The manager then stops doing fix-ups on that bucket. It operates under the assumption that the other peers will eventually freeze their copies of that bucket as well. If the freezing behavior is determined by the `maxTotalDataSizeMB` attribute, which limits the maximum size of an index, it can take some time for all copies of the bucket to freeze, as an index will typically be a different size on each peer. Therefore, the index can reach its maximum size on one peer, causing the oldest bucket to freeze, even though the index is still under the limit on the other peers.

Note: In 6.3, a change was made in how the cluster responds to frozen primary bucket copies, in order to prolong the time that a bucket remains available for searching:

In a pre-6.3 cluster, when a primary copy freezes, the cluster does not attempt to reassign the primary to any other remaining searchable copy. Searching on a bucket ceases once the primary is frozen.
In 6.3 and later, when a primary copy freezes, the cluster reassigns the primary to another searchable copy, if one exists. Searching then continues on that bucket with the new primary copy. When that primary also freezes, the cluster attempts to reassign the primary yet again to another searchable copy. Once all searchable copies of the bucket have been frozen, searching ceases on that bucket.

In both pre-6.3 and post-6.3, when a copy freezes, the cluster does not perform fix-up on the bucket; that is, it does not attempt to create a new copy,or to convert a non-searchable copy to searchable, to meet replication and search factors for the bucket.