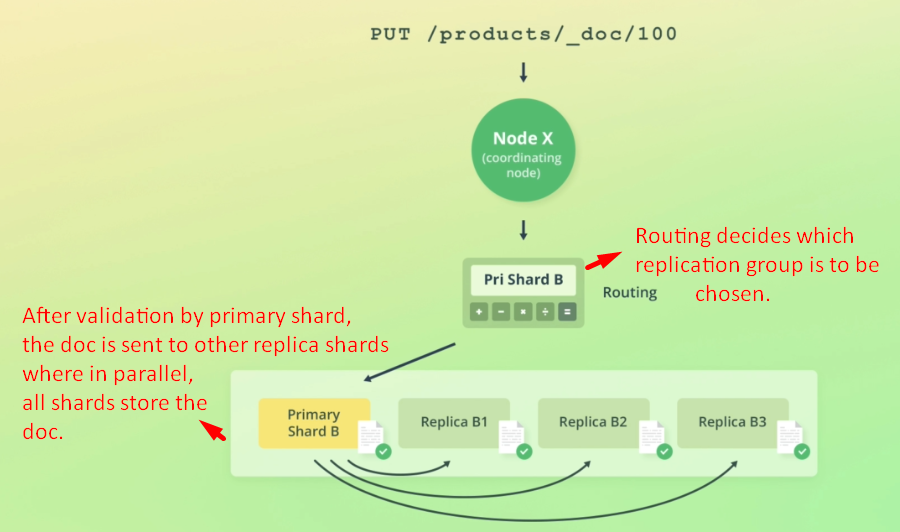
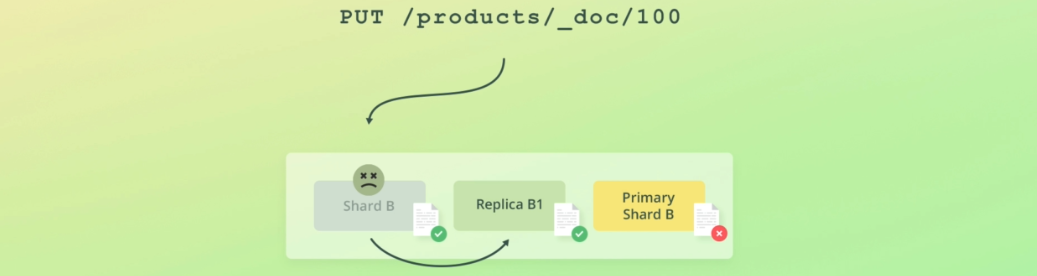
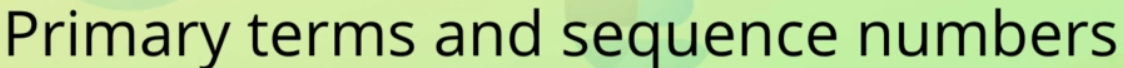
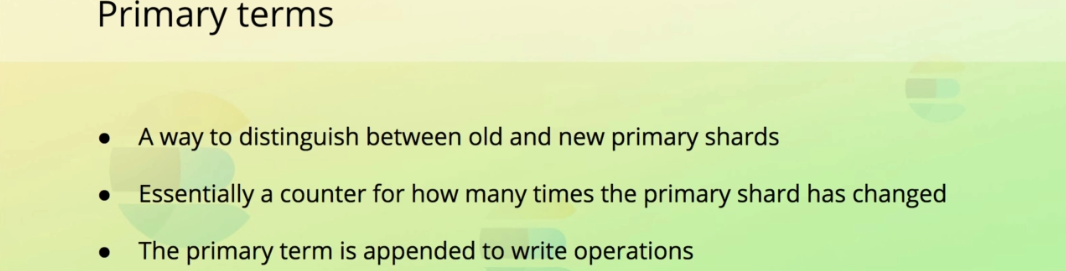
1. 
2. How it writes data?  
   
3. First of all the request goes through the same routing process we saw in last lecture.
4. Routing resolves the request to a replication group that stores or should store the doc.
5. Instead of routing the request to any of the shards within the replication group, write requests are always routed to the primary shard.
6. The primary shard is first of all responsible for validating the request. This involves validating the structure of the request, as well as validating field values.  
   **For instance**: attempting to add an object for a numeric field would cause a validation error.
7. The primary shard then performs the write operation locally, before forwarding it to the replica shards to keep those up to date as well.
8. To improve the performance, the primary shard forwards the operation to its replica shards in parallel.   
   Note that the operation will succeed even if the operation can’t be replicated to the replica shards.
9. Let’s go deeper.
10. In particular, how Elasticsearch handles failures in regards to data replication.  
    Since Elasticsearch is **distributed** and many operations happen **asynchronously**, a lot of things can go wrong, such as **hardware failures**. If such failures happen at a wrong time, things can get ugly. I guess there is no such thing as a “right” time for failure, but some failures can cause more trouble than others.  
    **Let me give you an example**.  
    
    1. Suppose we index a new document.
    2. The primary shard validates the operation and indexes the document locally.
    3. There are two replica shards and primary shard sends the operation to both.  
       The operation reaches only one of the replica shards, however, before the primary shard goes down, perhaps due to a hardware failure.
    4. When this happens, Elasticsearch goes through a recovery process. (We will not in detail about it now).
    5. One of the replica shards is promoted to primary shard as each replica group must have one primary shard.

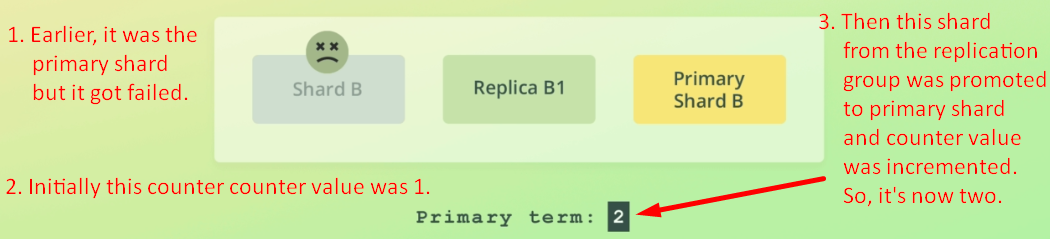
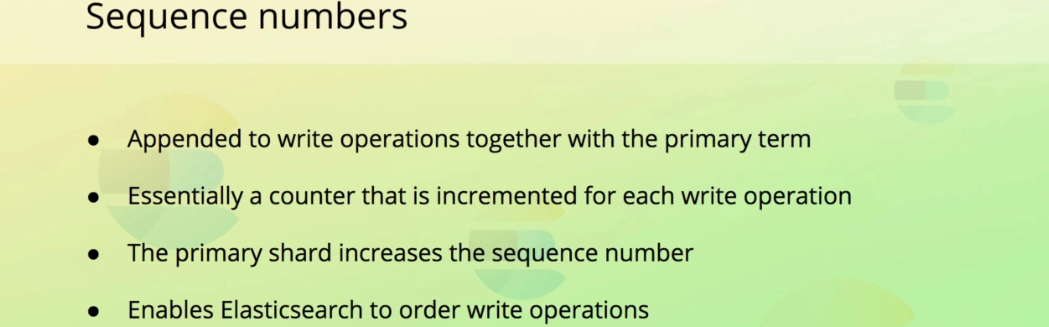
But the problem is that two remaining shards don’t share the same state, because one of them indexed the new document, while the other didn’t.

The replica shard that didn’t receive the index operation thinks it is up to date but that is not the case.  
Now you can image that we may not get the document which even though exists depending on which shard servers the retrieval request.   
If the shard is selected which doesn’t contain the doc serves, we will get empty result.

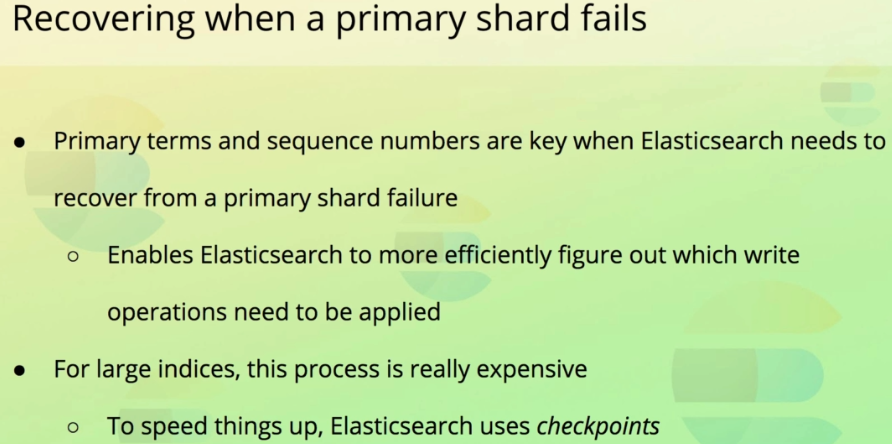
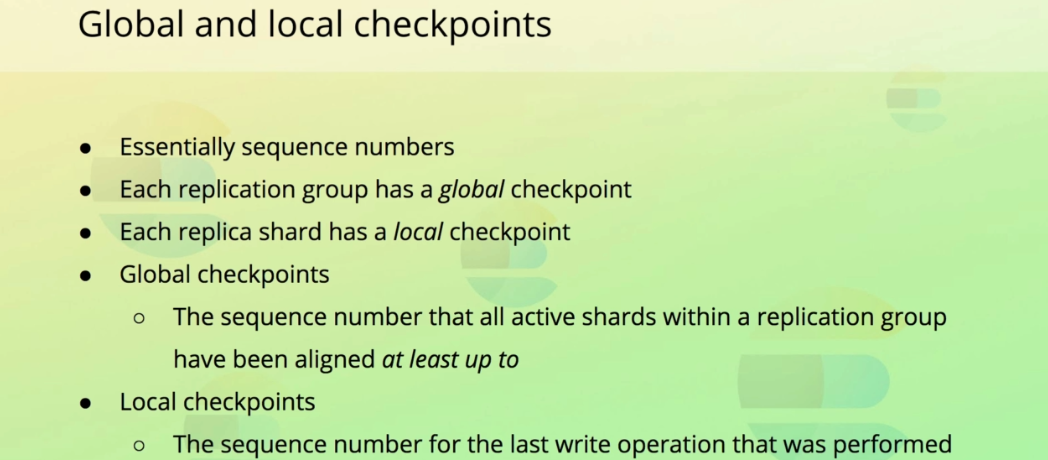
* 1. There may be other situations where things can go. Especially because both the risk and consequences increase when more nodes a cluster contains and the more writes an index needs to handle.

Solution 🡺 Primary Terms and Sequence numbers

1. Elasticsearch solves this problem and many others with something called **“Primary Terms & Sequence Numbers”.**
2. **Primary Terms**: A way for Elasticsearch to distinguish b/w old and new primary shards when the primary shard of a replication group has changed.
3. The primary term for a replication group is essentially just a counter for how many times the primary shard has changed.
4. **Example**:
   1. 
   2. The primary terms for all replication groups are persisted in the **cluster’s state**.
   3. In case of write operations, the current primary term is appended to the operations that are sent to the replica shards.  
      This enables the replica shards to tell whether or not the primary shard has changed since the operation was forward.  
      This by itself enables Elasticsearch to avoid a number of issues, but it’s not quite enough.
   4. A part from associating each operation with a primary term, **a sequence number** is also given to operations.  
      
   5. This sequence number is essentially just a counter that is incremented for each operation at least until the primary shard changes.  
      Primary shard is responsible for increasing this number when it processes a write request.

Sequence numbers enable Elasticsearch to know in which order operations happened on a given primary shard.

* 1. **Recovery  
     **
     1. Primary terms and sequence numbers enable Elasticsearch to recover from situation where a primary shard changes for instance due to a networking error.
     2. Instead of having to compare data on disk, it can use primary terms and sequence numbers to figure out which operations have already been performed, and which are needed to bring a given shard up to date.  
        However, if you have a large index, it’s not feasible to compare millions of operations to figure this out, especially not if data is being indexed and queried at a high rate at the same time.
     3. To speed this process, Elasticsearch maintains global and local checkpoints
  2. **Global and Local Checkpoints  
     **
     1. The global checkpoint is the sequence number that all of the active shards within a replication group have been aligned at least up to.   
        This means that any operation containing a sequence number lower than the global checkpoint have already been performed on all shards whthin the replication group.
     2. If a primary shard fails and rejoins the cluster at a later point, Elasticsearch only needs to compare the operation that are above the global checkpoint that it last knew about.
     3. Likewise, if a replica shard fails, only the operations that have a sequence number higher than its local checkpoint need to be applied when it comes back.  
        This essentially means that to recover, Elasticsearch only needs to compare the operations that happened while the shard was “gone”. Instead of the entire history of the replication group,
     4. Both of the primary term and sequence number are actually available within the response to write requests as well as when retrieving a document by its ID.

1. 