* MongoDB allows distribution of its contained data set across several replicas which collectively forms a replica set
  + The means there are several daemon instances or background processes ("mongod") which all manage the same set of data
    - Each set of data is ideally on a separate machine but should at least refer to a physically separate location on disk
    - Benefits
      * Fault tolerance
      * Greater throughput as disk accesses can be handled by different machines in the cluster
      * Lower latency
      * Recovery upon failure
  + Any application accesses these instances through an interface exposed by an mongos object
  + Can elect to shard any collection in a database
    - This means dividing the collection into discrete chunks
      * The division is done by choosing a field or a series of fields that would exist in every document in the collection
      * This becomes the shard key which defines a lower and upper bound of key values
    - and placing each chunk onto an ideally separate cluster of machines
    - Different from full replication in that each cluster only holds a portion of the data
    - Within each shard/replica set, there is a primary replica that handles all the writes for the replica set and by default handles the reads if available to do so
    - Chunk migration
  + The choice of the shard key is important
  + Each replica will have its own non-volatile memory
    - Note that within the "mongod" daemon process controlling the replica is a storage engine that manages that replica's interaction with storage
    - There are several storage engines that are available in MongoDB
      * The choice of storage engine will basically affect how the replica indexes the records and in what manner it interacts with volatile and non-volatile memory
        + Indexing - how a database uses a key or a composite key to find a desired document which is arranged in storage in a particular manner that doesn't involve scanning an entire document
    - The choices include
      * MMAPv1
      * Wired Tiger
      * In-memory storage engine
    - The base arrangement of the records across these engines is a B-Tree data structure arranged by the index
      * B-Tree is a balanced tree structure where there are:
        + 0 to many internal nodes containing no actual data themselves but point to other internal nodes or eventually point to the leaf nodes which actually contain the information
        + Functions similar to a BST
        + Log base M complexity on reads
        + Rebalances on insertion/deletion (potentially makes a write expensive if it necessitates a rebalance)
      * Moving from MMAPv1 (deprecated) to WiredTiger -> hold the B-Tree partially in memory and batch write to disk when tree grows to certain size
        + More importantly, moving from collection locking single-writer multiple reader protocol to document locking multiple writer multiple reader (meaning, asynchronous writes are now possible)

Configuring MongoDB to allow reading from secondaries means there is no strict consistency guarantee between replicas but eventually the changes will propagate

* + - * + In-memory storage engine -> holding basically everything in volatile memory and writing nothing to non-volatile memory
  + Writes by a primary are logged
    - This means the primary will conduct it's write to the disk, then log whatever it did to it's oplog
    - This log is propagated to the secondaries who will then apply the same operations to their own data sets
    - Note that the primary does not send an image to the secondaries
    - Practice of write ahead logging: favour write speed as opposed to fast recovery with images
  + The choice of the number of replicas is important
    - The documentation explicitly asks that each replica set contain an odd number of replicas
      * For fault tolerance purposes.
      * MongoDB uses heartbeat messages to determine whether a replica has failed
        + If the primary doesn't send a heartbeat message within a timeout threshold (settable), then the secondaries conclude that the primary has failed
      * The remaining secondaries will conduct an election to determine the next primary
      * Winning an election requires that the winning replica attain a simple majority (>50% of the votes)
      * Basically this means the more replicas in a shard, the more tolerant that shard is towards replicas failing
      * Each replica has a priority value ranging from 0 to 1000, with the highest number being the intended next primary
      * The elected primary will then propagate a log of operations to the other remaining secondaries, who can then bring their state up to date.
      * MongoDB specifies that each shard can have at most 7 voting replicas and 43 extra non-voting members
        + The non-voting members cannot become the primary and are there just for performance reasons
        + Show math behind fault tolerance
    - Note that primary failure is not the only thing that can cause a change in primary
      * Situations like adding a new node or calling a method that reconfigures a replica will trigger a new election