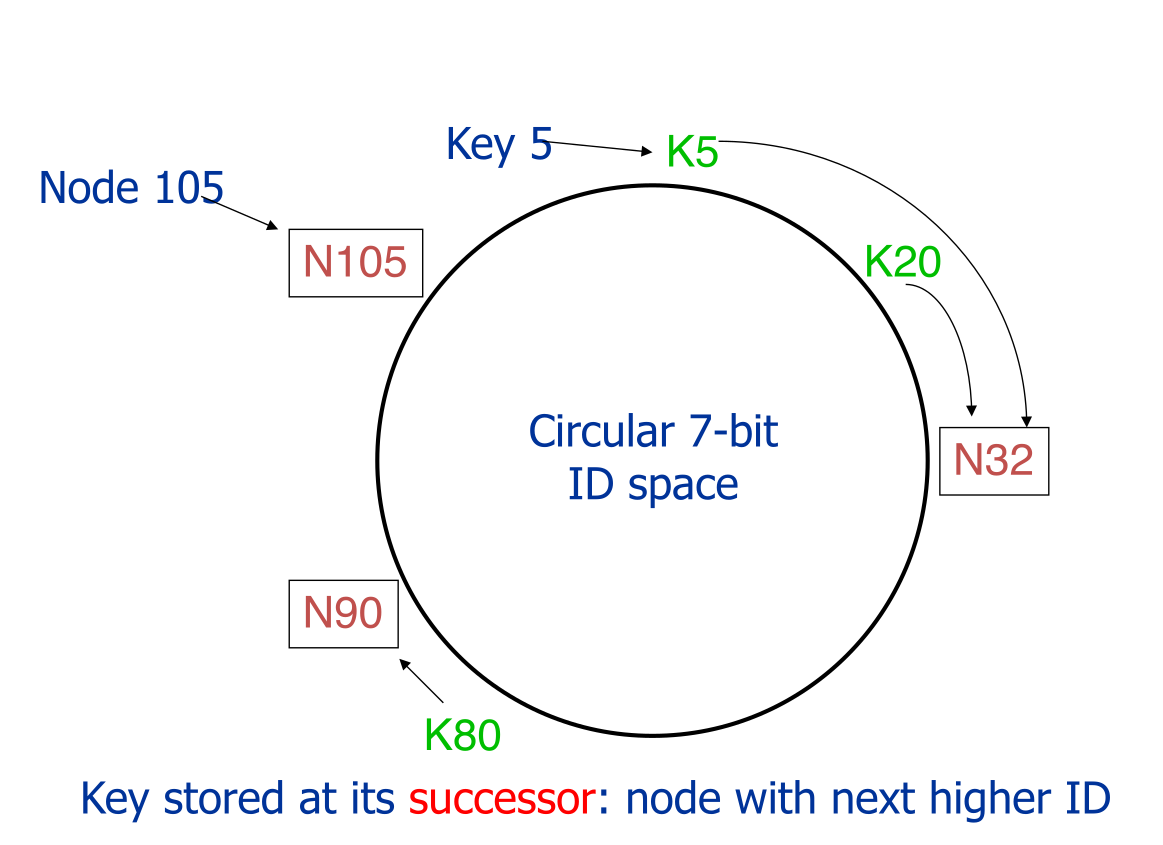
1. tt
   1. 1. An RDD is a read-only collection of data that can be partitioned across a subset of Spark cluster machines and form the main working component that can be operated in parallel with a low-level API that offers transformations and actions.
      2. Scalability is the property of a system to handle a growing amount of work by adding resources to the system.
      3. *At the start of the day, Dave starts a new rumor: he comments to Bob that he believes that Charlie dyes his mustache.*   
         Procedures or processes of computer peer-to-peer communication which disseminates through the system; can be based on new/relevant information received.
      4. Any given response from the server is independent of any sort of state.
   2. 1. “External consistency states that Cloud Spanner executes transactions in a manner that is indistinguishable from a system in which the transactions are executed serially” - From gcp docs.
      2. The TrueTime API provides the now() function which returns the clock uncertainty instead of a set timestamp, by returning it as a tuple [earliest time, latest time]. The API also has provides two functions before(t) and after(t) which returns a boolean if the time stated in t has **definitely** passed or not (I.e. Avoids the period of uncertainty in the now() function call).
      3. The API could look at various time servers from various sources (e.g. GPS, Atomic clocks, other data centers) and return the earliest and latest time in this list of times gathered.
      4. In the event of a network partition, Spanner chooses consistency over availability. The most likely outcome of a partition in practice is that one side has a quorum and will continue just fine, perhaps after electing some new leaders. Thus, the service continues to be available, but users on the minority side have no access.
      5. The side with the new leader has to now carry more load than the other side of the partition???
      6. For the commit time, the best case should be > k ms. Assume the write transaction will take less than k ms, then Spanner will still wait a full duration of commit time before going to the next transaction, which = 2 epsilon = k ms. Assume write transaction will take more than k ms, than the best case will still be > k ms.
2. 1. 1. Location transparency means that a client is unable to tell the difference in system performance no xmatter which location the server sending the response is in. We see this in spanner with its dynamic cross-DC load balancing.
      2. A batch processing system is one that applies operations on collections of data (as opposed to individual transactions on data items). MapReduce is an example of such a system.
      3. It’s what it says on the tin. The pattern by which data is partitioned. Commonly used to determine which machines keep which data in a distributed system
      4. A multi-version store is a data store that maintains multiple versions of the same data. BigTable is a key example of this, allowing the user to specify how many versions are kept.
   2. 1. Dynamo keeps data stored on virtual nodes, accessed in a (single-hop) distributed hash table via consistent hashing. Each node knows the range of keys covered by one another and a request received by a node can be immediately forwarded to the correct one.  
           
           
         This architecture applies for both reads and writes. Dynamo is a multi-version key-value store. In terms of the API, reads are performed through calls to get() and writes are performed through put(). Dynamo uses a sloppy Quorum in order to ensure eventual consistency.
      2. The first challenging aspect of Dynamo’s design I noticed was the way that it handles membership and failure detection. It does this using a gossip-based protocol, which makes the design more complex and requires more communication between the machines.  
           
         The second aspect was handling the failures, in which hinted-handoff is used. Again, this adds complexity and results in more inter-device communication. Ideally, we want to minimize the different tasks that a node storing the data should need to do.
      3. An alternative to both the above aspects would be to have a centralized “master” node that is solely responsible for handling replicas (failures) and also detecting failed nodes. The master would ensure a reasonable number of replicas is maintained as it would know which nodes stored which range of keys. It would also be responsible for periodically pinging nodes, taking some of the load off the individual storage nodes, and it could also take part in request routing.  
           
         Note that this does not reduce the scalability as the “master’s” only responsibility is to store metadata. We are still able to achieve incremental scalability with the hash partitioning and virtual nodes.  
           
         We would avoid master node failures by simply having a pair of master node machines and implementing hinted handoff between the two (as there are only two, they are less likely to fail).
3. I can see why no one answered this question