Dynamo DB Notes:

- Sacrificing consistency for availability - eventual consistency
- Highly reliable/scalable KV store. ACID data stores have poor availability, and traditional replicated RDBMS systems focus on guaranteeing strong consistency to replicated data. Hence, they are limited in scalability and reliability.
- Decentralized, loosely coupled, hundreds of services.
- Customers should always be able top view and add items to shopping cart, so service always has to be able to write to and read from data store + data needs to be available across data centers.
- Consistent hashing to partition and replicate data, consistency facilitated by obj versioning.
- Quorums + decentralized replica synchronization protocol for consistency among replicas.
- Gossip based failure detection
- Most microservices @ Amazon retrieve data by primary key, and do not need querying of RDBMS
- Query Model: State is stored as binary objects (blobs) identified by unique keys. No isolation guarantees, single key updates.
- Clients and services engage in SLAs (agreement between client + service, client's expected request rate distribution for an API + service latency).
    - Ex: Service guaranteeing to provide a response within 300ms for 99.9% of requests for peak client load of 500 requests per second
- Services are stateless
- Data replication algorithms in commercial systems perform synchronous replica coordination to provide a strongly consistent data access interface. These are forced to tradeoff the availability of data.

- When dealing with network failures, strong consistency and high availability cannot be achieved simultaneously.
- Dynamo ⇒ always writeable data store.
    - conflict resolution is pushed to reads during server failures, to ensure writes are never rejected.
- Last write wins methodology during conflicts
- Incremental scalability: should be able to scale out one storage host(node) at a time with minimal impact on operators of the system and system itself
- Symmetry: Every node in Dynamo should have the same set of responsibilities as its peers.
- Decentralization: Favors peer-to-peer techniques over centralized control (something like using Zookeeper). More scalable/available
- Heterogeneity: work distribution must be proportional to capabilities of individual servers
- Structured P2P networks use a globally consistent protocol to ensure any node can efficiently route a search query to some peer that has the desired data. Routing mechanisms ensure queries can be answered within a bounded # of hops.
    - To reduce latency from multi-hop routing, some P2P systems use O(1) routing where each peer maintains enough routing info locally to route requests to appropriate peer within constant # of hops.
- Distributed file systems support hierarchical namespaces. GFS uses a single master server for hosting metadata, data is split into chunks and stored in chunk servers.
- Similarly, Dynamo allows read and write operations to continue during network.
    - KV store is intended to store smaller obj (size < 1M)
    - easier to configure on per-app basis.
- Dynamo does not focus on problem of data integrity, and is built for a trusted env, unlike Antiquity (distributed storage system) which uses a secure log to

preserve data integrity, and replicates each log on multiple servers, and uses Byzantine fault tolerance protocols to ensure consistency.

- BigTable has a sparse multi-dimensional sorted map to allow apps to access data using multiple attributes. Conversely, dynamo targets apps that require only K/V access with a focus on high availability.
- 4 rules:
    - always writeable
    - all nodes assumed to be trusted
    - apps that use dynamo do not require support for hierarchical namespaces
    - built for latency sensitive apps that require 99.9% of read and write operations to be performed within a few milliseconds.
- To meet these latency requirements, routing requests thru multiple nodes should be avoided. Dynamo uses zero-hop DHT, where each node maintains enough info locally to route a request to appropriate node.
- system needs to have scalable and robust solutions for load balancing, membership and failure detection, failure recovery, replica synchronization, overload handling, state transfer, concurrency and job scheduling, request marshalling, request routing, system monitoring and alarming, and configuration management.
- core distributed systems techniques used in Dynamo: partitioning, replication, versioning, membership, failure handling and scaling.
- Utilizes a key-value interface for object storage, supporting `get()` and `put()` operations, employs MD5 hashing for key-based data placement and replication, and manages data versioning and validity through contextual metadata.
- To scale incrementally, data must be **partitioned** over set of nodes.
    - Consistent hashing to distribute load across multiple storage hosts:
        - output range of hash function treated as fixed circular ring
        - each node assigned random value, representing position

        - each data item identified by key is assigned to node, by hashing data item's key to get position on ring + walking rings clockwise to find first node with larger position than cur.
        - each node becomes responsible for the region in the ring between it and predecessor node on the ring.
            - departure or arrival of node only affects immediate neighbors and other nodes remain unaffected.
        - Dynamo addresses consistent hashing challenges by introducing virtual nodes, enabling uniform data distribution and load balancing
            - instead of mapping a node to a single point in the circle, each node gets assigned to multiple points in the ring
            - if a node becomes unavailable, load handled by this node is evenly dispersed across remaining available nodes.
            - when node becomes available again or a new node is added to the system, the newly available node accepts an equivalent amount of load from other available nodes.
- Dynamo **replicates** data on multiple hosts:
    - Data is replicated across N hosts, with each key assigned to a coordinator node for replication to N-1 clockwise successor nodes, ensuring each node covers a segment of the ring and maintains a "preference list" of nodes responsible for each key.
    - The preference list extends beyond N to compensate for node failures and virtual nodes, ensuring it includes distinct physical nodes by skipping over successive virtual positions owned by the same physical node to maintain diversity in data storage responsibility.
- Dynamo provides eventual consistency which allows for updates to be propagated to all replicas asynchronously.
    - Dynamo allows applications like shopping carts (adding items to cart and deleting are put requests) to function with eventual consistency by maintaining immutable versions of data, enabling both concurrent updates

and reconciliation of divergent data versions to ensure no operation (add/remove) is lost despite failures or network issues.

- Certain failure modes can result in system having several versions of the same data. Must design apps that acknowledge the possibility of multiple versions of the same data.
- Dynamo uses **vector clocks** to capture causality between different versions of the same object.
  - A vector clock is a list of (node, counter) pairs. One vector clock is associated with every version of every object. Can determine whether two versions of an object are on parallel branches or have casual ordering by observing vector clocks.
  - If counters on 1st obj's clock ≤ all nodes of 2nd clock, then 1st is an ancestor of 2nd, and can't be forgotten. Otherwise, the two changes are a conflict and require reconciliation.
  -