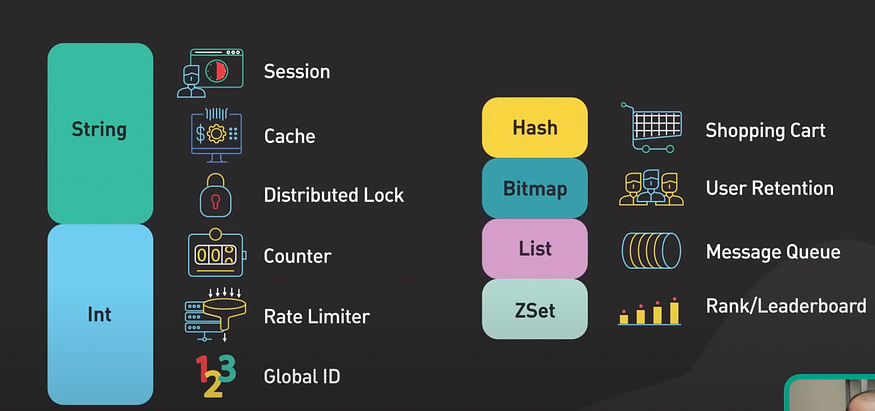
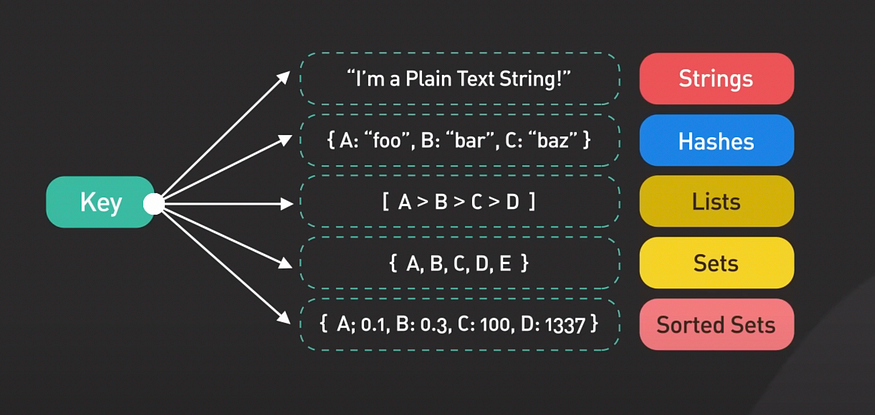
We’ll talk about the top use cases that have been better tested in production at various companies and at various scales, Get insights into how Redis solves interesting scalability challenges for us, and learn why it is a great tool to know well in our system design tool set.



First of all, what is Redis and why do people use it? Redis is an in memory data structure store. It’s most commonly used as a cache. It supports many data structures such as strings, hashes, lists, sets, and sorted sets. Redis is known for its speed.

**1. Caching Objects**

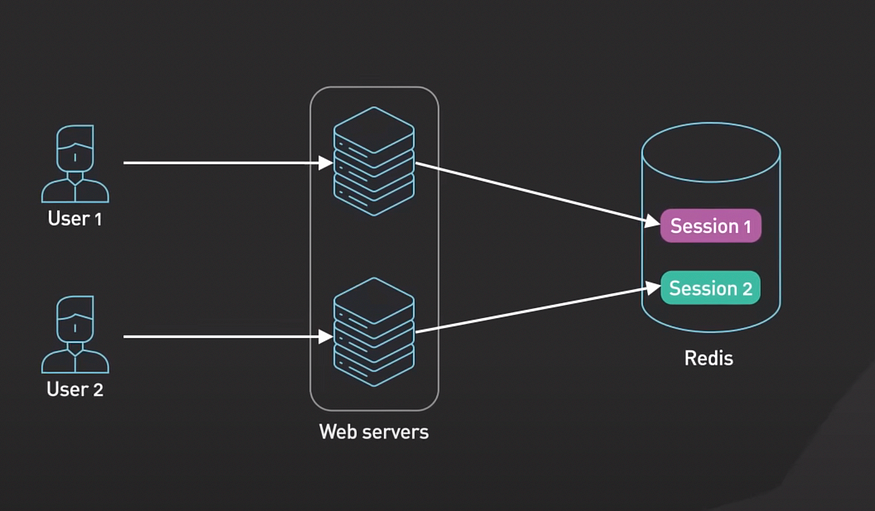
* The number one-use case for Redis is caching objects to speed up web applications.

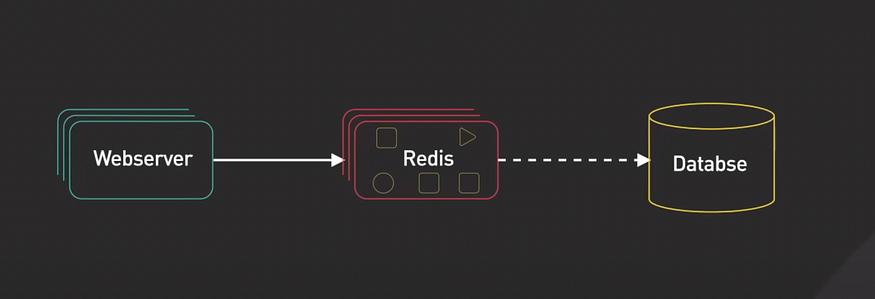


* In this use case, Redis stores frequently requested data in memory. It allows the web servers to return frequently accessed data quickly.
* This reduces the load on the database and improves the response time for the application.
* At scale, the cache is distributed among a cluster of Redis servers. Sharding is a common technique to distribute the caching load evenly across the cluster.

**2. Session Store**

* To share session data among stateless servers. When a user logs into a web application, the session data is stored in Redis along with a unique session ID that is returned to the client as a cookie.

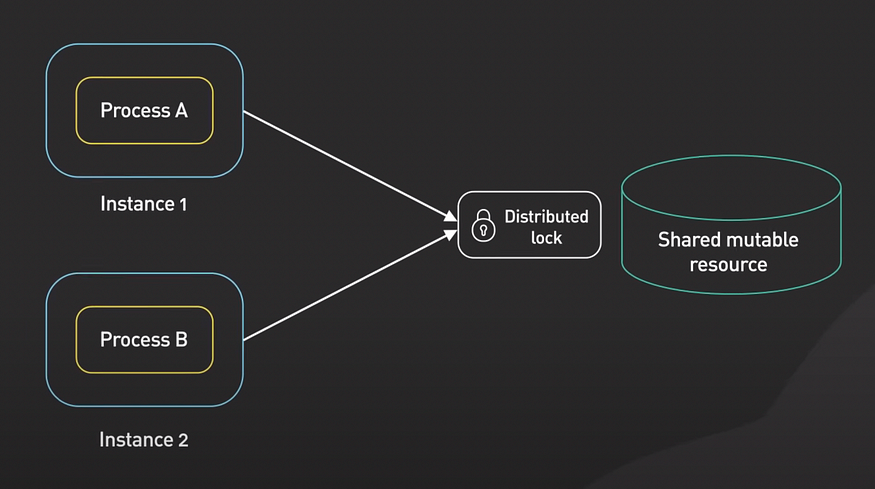




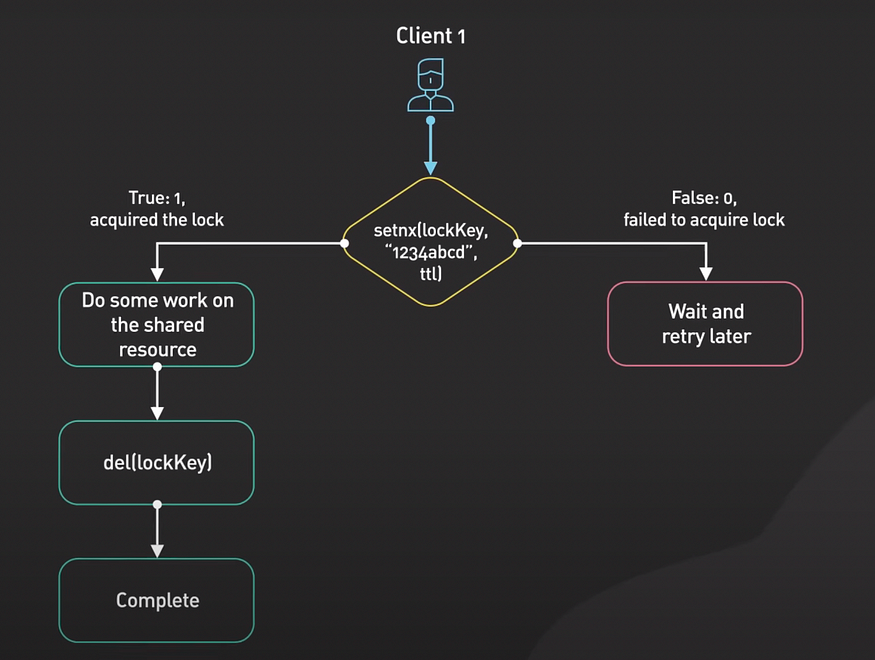
* When the user makes a request to the application, the session ID is included in the request and the stateless web server retrieves the session data from Redis using the ID.
* The session data stored in Redis would be lost if the Redis servers restarts. Even though Redis provides persistent options by a snapshot and AOF or append only file, they allow session data to be saved to disk and reloaded into memory in the event of a restart.
* These options often takes too long to load to be practical in production. Replication is usually used instead in this case. Data is replicated to a backup instance.
* In the event of a crash of the main instance, the backup instance is quickly promoted to take over the traffic.

**3. Distributed Locks**

* Distributed locks are used when multiple nodes in an application need to coordinate access to some shared resource. Redis is used as a distributed lock with its atomic commands like SETNX or SET if not exist.



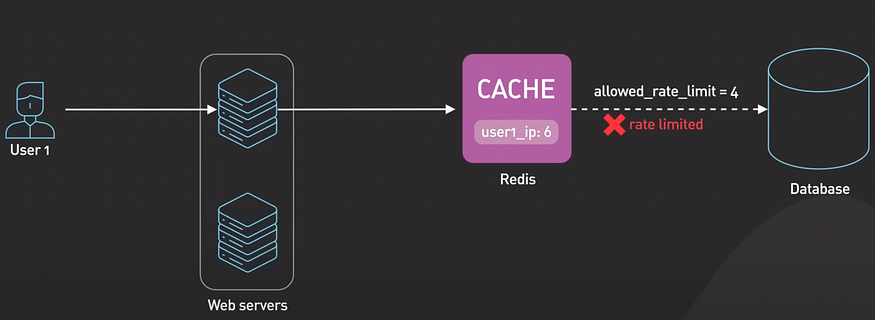
* It allows a caller to set a key only if it does not already exist.
* Here’s how it works at a high level. Client One tries to acquire the lock by setting a key with a unique value and a timeout using the SETNX command. If the key was not already set, the SETNX command returns one indicating that the log has been acquired by Client one. Client one finishes its work.



* And releases the log by deleting the key. Now if the key was already set, the SETNX command returns 0, indicating that the log has already been held by another client. In this case client one waits and retries the SETNX operation until the log is released by the other client.
* Note that this simple implementation might be good enough for many use cases, but it’s not completely fault tolerant for production use.
* There are many Redis client libraries that provide high quality distributed lock implementation built right out-of-the-box.

**4. Rate Limiter**

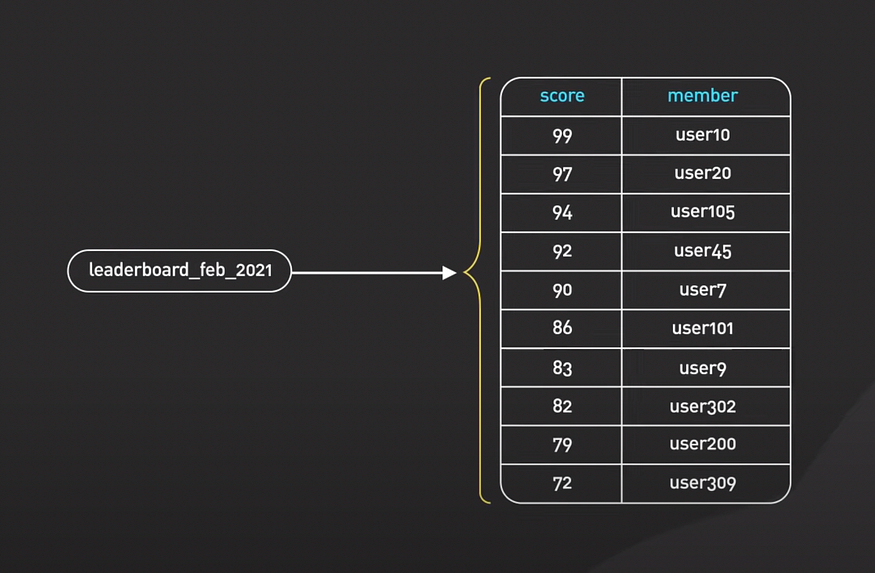
* Redis can be used as a rate limiter by using its increment command on some counters and setting expiration times on those counters.



* A very basic rate limiting algorithm works this way. For each incoming request, the request IP or user ID is used as a key.
* The number of requests for the key is incremented using the increment command in Redis. The current count is compared to the allow rate limit. If the count is within the rate limit, the request is processed. If the count is over the limit, the request is rejected. The keys are set to expire after a specific time window, for example a minute, to reset the counts for the next time window.
* More sophisticated rate limiters, Like the leaky bucket, algorithm can also be implemented using Redis.

**5. Gaming Leaderboard**

* Redis is a delightful way to implement various types of gaming leaderboards. Sorted Sets are the fundamental data structure that enables this.



* Sorted Set is similar to the Set data structure in Redis. Members can be a list of non-repeating strings. The only difference is that each member is associated with a score, a floating-point number that provides a sorting order for the Sorted Set. Members are always sorted from the smallest to the greatest score.
* The elements are sorted by score. This allows for quick retrieval of the elements by a score in logarithmic time.