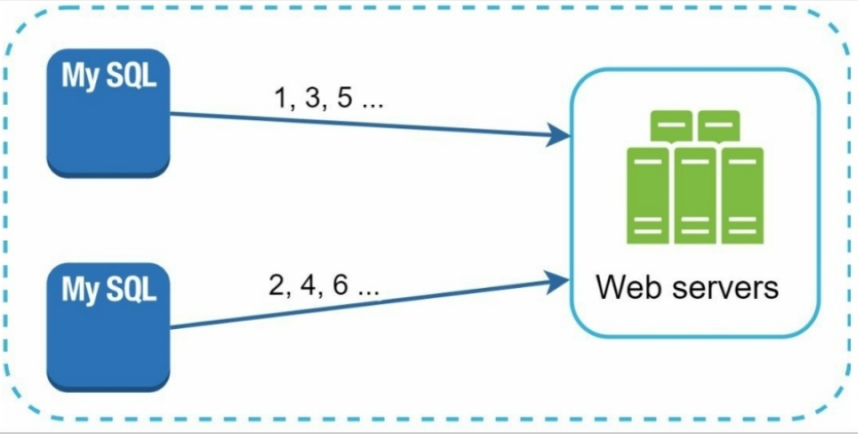
Design a Unique ID Generator in Distributed Systems

- Step 1 - Understand the problem and establish design scope

  - ID must be unique.
  - IDs are numerical value only.
  - IDs fit into 64-bit.
  - IDs are ordered by date.
  - Ability to generate over 10,000 unique per second.

- Step 2 - Propose high-level design and get buy-in
  - Multiple options can be used to generate unique IDs in distributed systems. The options are:
    - Multi-master replication
    - Universally Unique Identifier (UUID)
    - Ticket Servers
    - Twitter Snowflake Approach.

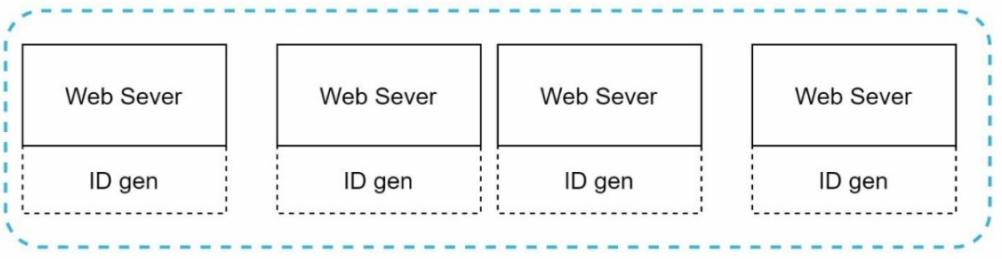
- Multi-Master Replication



- This approach uses the database's auto-increment feature. Instead of increasing the next ID by 1, we increase it by k, where k is the number of database servers in use.

- This strategy has major drawbacks:
  - Hard to scale with multiple data centers.
  - IDs do not go up with time across multiple servers.
  - It does not scale when a server is added or removed.

- UUID
- A UUID is another way to obtain unique IDs. UUID is a 128 bit number used to identify information in computer systems.
- UUID has a very low probability of getting collision.

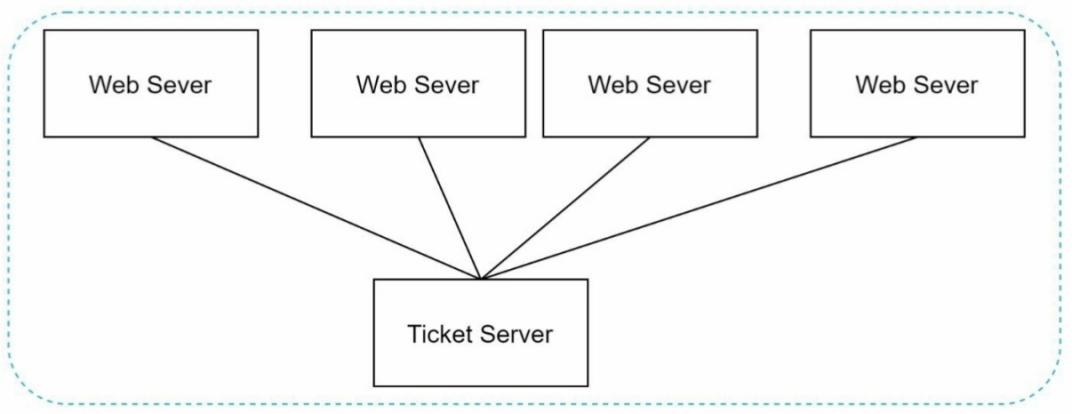


- In this design, each web server contains an ID Generator, and a web server is responsible for generating IDs independently.

- Pros:
  - Generating UUID is simple. No coordination between servers is needed so there will not be any synchronization issues.
  - The system is easy to scale because each web server is responsible for generating IDs they consume. ID generator can easily scale with web servers.

- Cons:
  - IDs are 128 bits long, but our requirement is 64 bits.
  - IDs do not go up with time.
  - IDs could be non-numeric.

- Ticket Server
  - Tickets servers are another interesting way to generate unique IDs.
  - Flicker developed ticket servers to generate distributed primary keys.

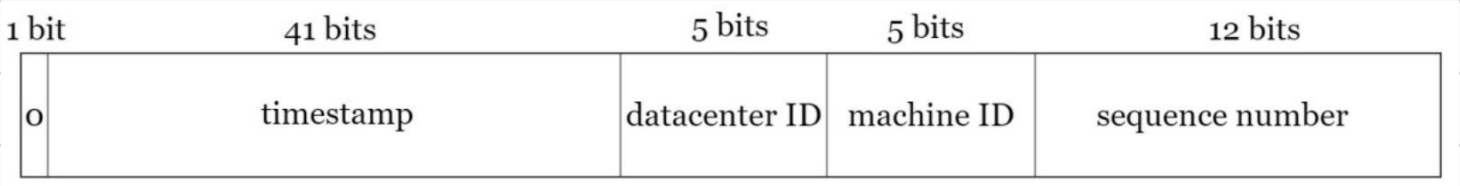


  - The idea is to use a centralized auto_increment feature in a single database server.

- Pros:
  - Numeric IDs.
  - It is easy to implement, and it works for small to medium-scale applications.

- Cons:
  - Single Point of Failure. Single ticket server means if the ticket server goes down, all systems that depend on it will face issues. To avoid a SPOF, we can set up multiple ticket servers. However, this will introduce new challenges such as data synchronization.

- Twitter Snowflake Approach
  - Instead of generating an ID directly, we divide an ID into different sections.

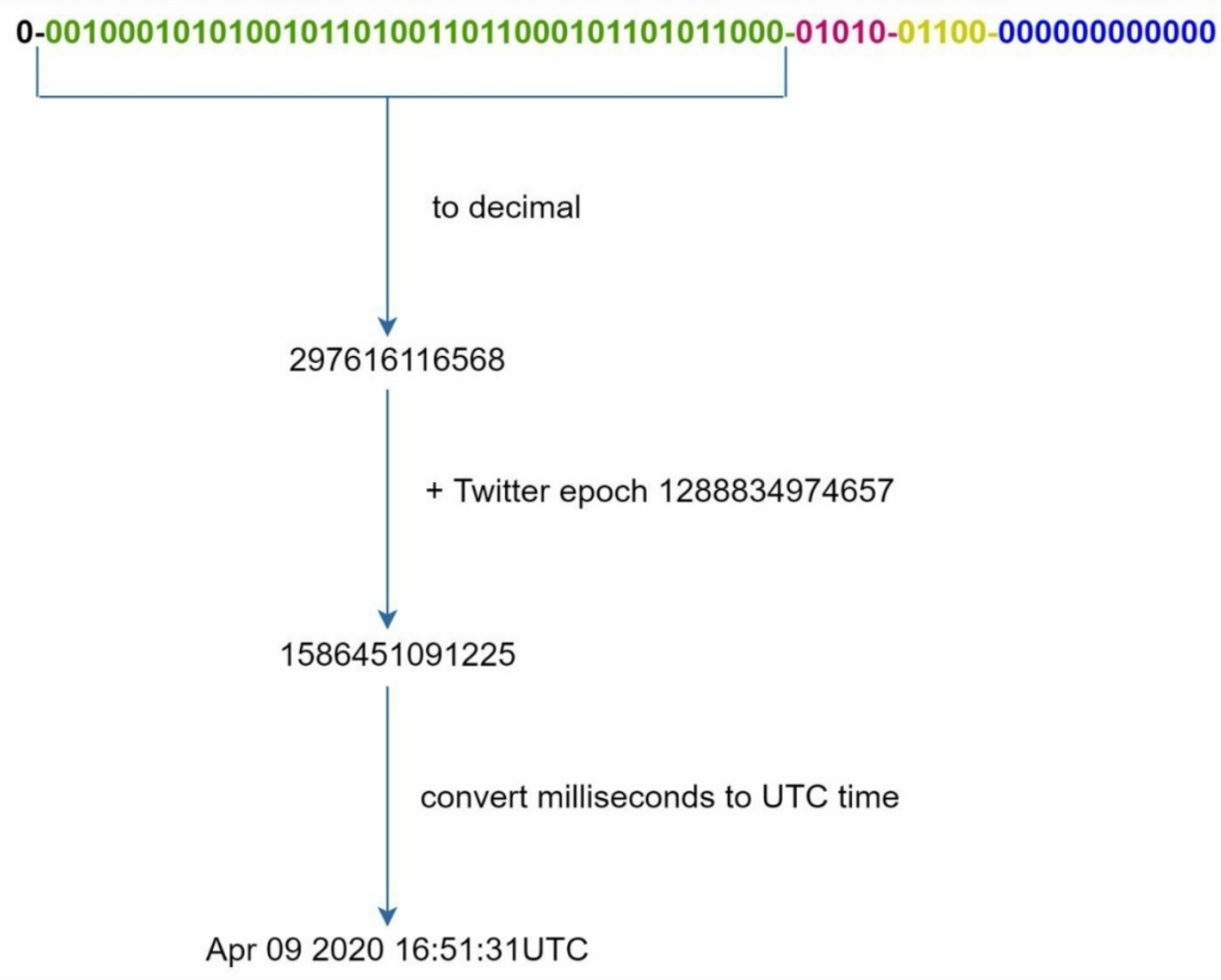


1 bit	41 bits	5 bits	5 bits	12 bits
0	timestamp	datacenter ID	machine ID	sequence number

  - Each section is explained below:
    - Sign bit: 1 bit. It will always be 0. This is reserved for future uses. It can potentially be used to distinguish between signed and unsigned numbers.
    - Timestamp: 41 bits. Milliseconds since the epoch or custom epoch.
    - Datacenter ID: 5 bits, which gives us $2^5 = 32$ datacenters.
    - Machine ID: 5 bits, which gives $2^5 = 32$ machines per datacenter.
    - Sequence number: 12 bits. For every ID generated on that machine, the sequence number is incremented by 1. The number is reset to 0 every millisecond.

- Step 3 - Design Deep Dive
  - Twitter snowflake ID generator approach works best for the given requirements.
  - Datacenter IDs and machine IDs are chosen at the startup time, generally fixed when the system is running.

  - Timestamp
    - The most important bits make up the timestamp section. As timestamps grow with time, IDs are sortable with time.



- The maximum timestamp that can be represented in 41 bits is $2^{41} - 1 = 2199023255551$ ms $\simeq$ 69 years. This means the ID generator will work for 69 years.

- Sequence Number
  - Sequence number is 12 bits, which gives 4096 combinations. This field is 0 unless more than one ID is generated in a ms. In theory, a machine can support a maximum of 4096 new IDs per ms.

- Step 4 - Wrap Up
  - Few additional talking points:
    - Clock Synchronization: In our design, we assume ID generation servers have the same clock. This assumption might not be true when a server is running on multiple cores or in multi-machine scenario. Network Time Protocol is the most popular solution to this problem.
    - Section length tuning: Fewer sequence numbers but more timestamp bits are effective for low concurrency and long-term applications.
    - High Availability: Since ID generator is a mission critical system, it must be highly available.