Design a Unique Id Generator

Why do we need unique ids?

→ social media posts
→ videos on youtube
→ songs on spotify
→ comments
→ cash transactions
→ messages in event-driven architecture

} unique_id's used everywhere

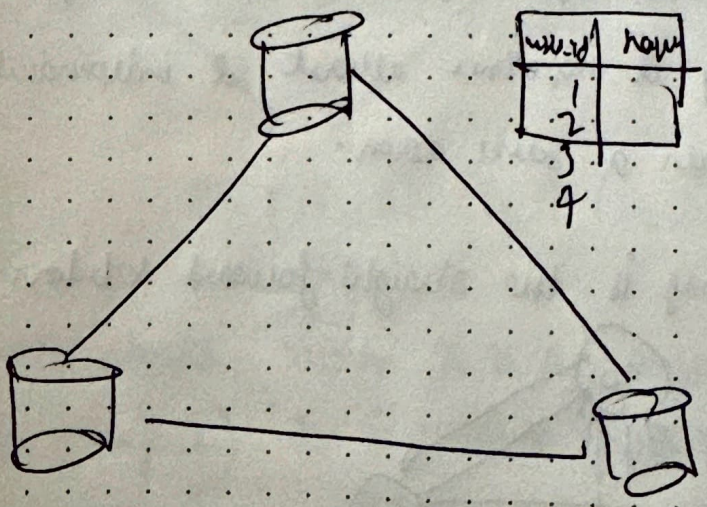
Simple solution → just auto-increment

say, I want to generate user_id for all new users.

* good for single served applications.

user_id	name

But if our application is distributed,



user_id	name
1	
2	
3	
4	

user_id	name
1	
2	
3	
4	

each generating its own id's

say a name in S1 also signed up in S2.

like 1 → Rahul 1 → Chetan

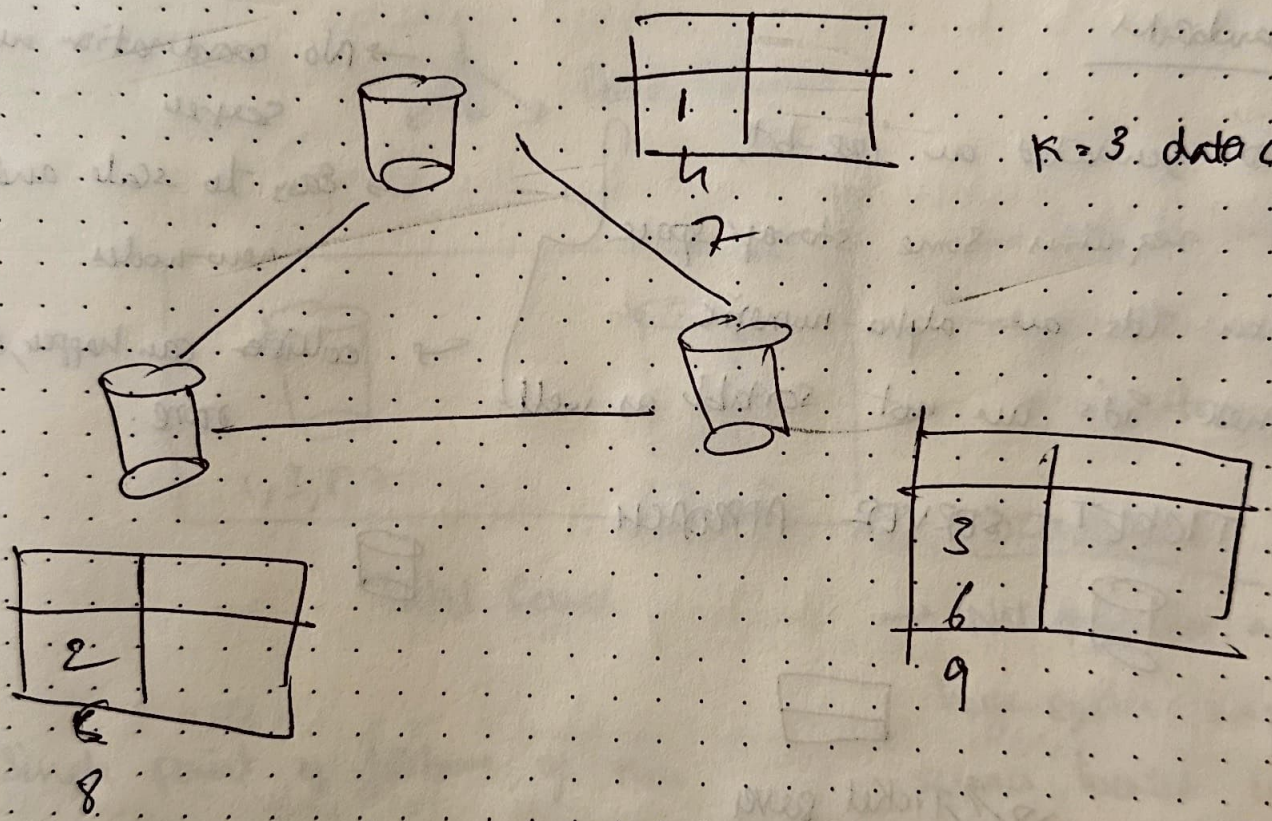
S1 ✓ S2 ✓

same id but 2 different users so the issues in this are

→ Not good for distributed systems and a single id can get assigned to multiple resources.

→ Different requirements can be there for unique-ids.

① Multi-master Replication method



K = 3 data centers.

So, their tables starts with these values and the offset is say K.

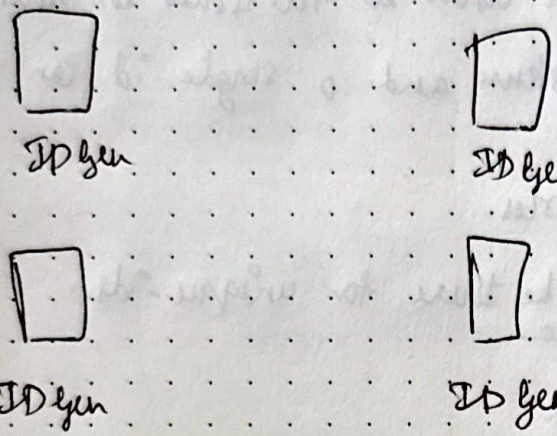
→ Easy to implement

→ But not easy to scale as all of this can get disrupted

2nd solution is

(2) Using UUIDs

universally unique identifier ids → 128 bit



Each server to have its own unique ID generator.

→ Simple way to generate unique IDs.

→ No coordination needed b/w servers.

→ Easy to scale and add new nodes.

→ collision can happen, but rare.

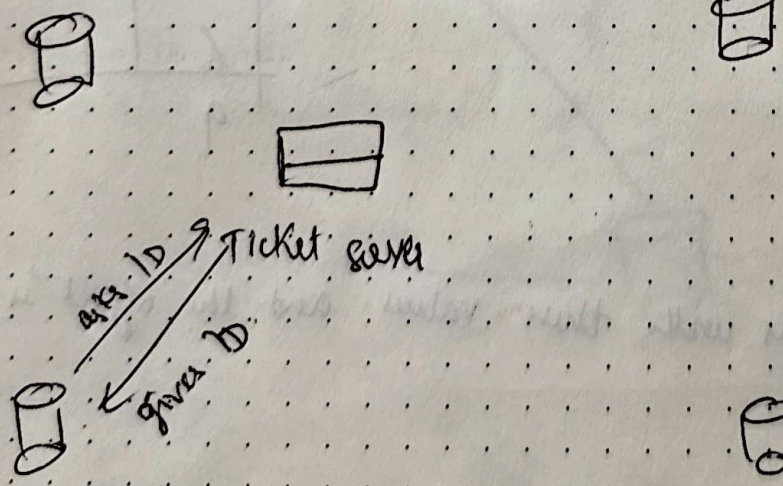
Drawbacks

→ IDs generated are 128-bit, requires some storage space.

→ These Ids are alpha-numeric.

→ These ids are not sortable as well.

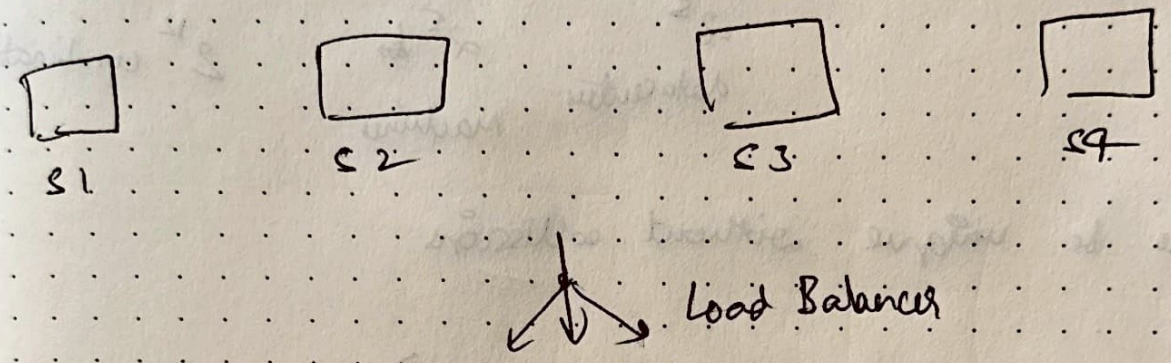
(3) TICKET - SERVER APPROACH



→ Unique ids generated by this central system called Ticket server.

→ So, say if we need a unique ID for some resource then it contacts the ticket server and get back the unique-id ~~based on~~

→ Ticket server is responsible for say just numeric id's (or) alphanumeric id's (or) different id's for say users and different kind of ids for posts / images etc.



→ Easy to implement

→ generates unique_id of any format.

DRAWBACKS

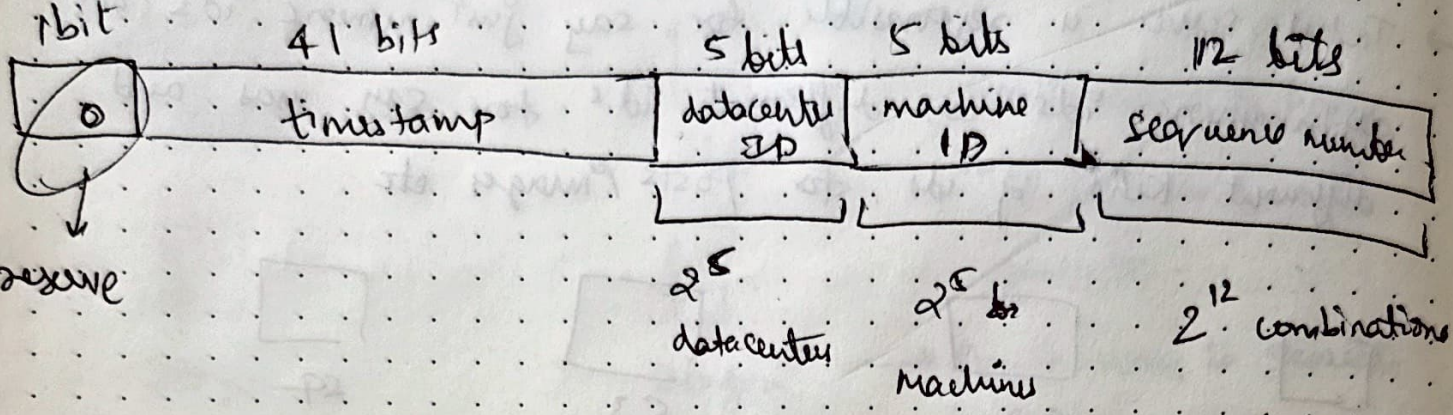
→ Not good for a huge system say servers located in different region, always need to have that n/w hop to the centralized system

→ Single point of failure, if this ticket server goes down, then anything goes down.

Twitter Snowflake approach

Worked on divide and conquer approach.

They used a 64-bit id, where



→ Guaranteed to be unique without collisions

→ Super fast

→ can give sortable ids (also has time-stamp)

Hence, the most robust approach

Drawbacks

→ Time synchronisation across servers is imp.

→ Implementation can be tricky.

→ If this approach fails some machines? (✗ No SPOF)

→ other machines continue generating IDs without any issue.

→ If a machine restarts, it continues to generate IDs based on the latest time-stamp.

→ time-stamp component ensures older IDs are never duplicated.