**DESIGN A UNIQUE ID GENERATOR IN DISTRIBUTED SYSTEMS**

**SIMPLEST APPROACH**

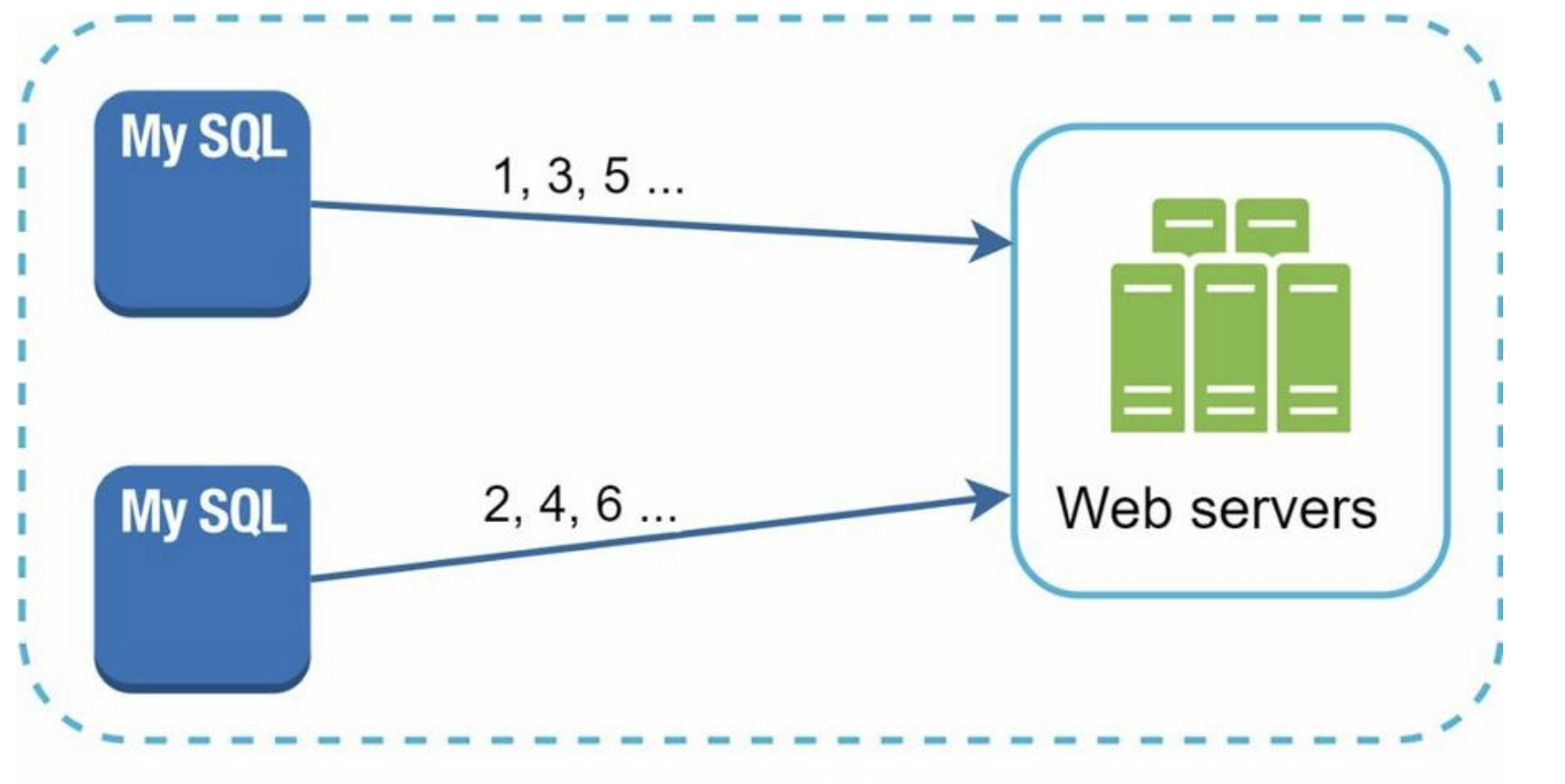
Use a primary key with the auto\_increment attribute in a traditional database

Problem with the above

auto\_increment does not work in a distributed environment because a single database server is not large enough and generating unique IDs across multiple databases with minimal delay is challenging

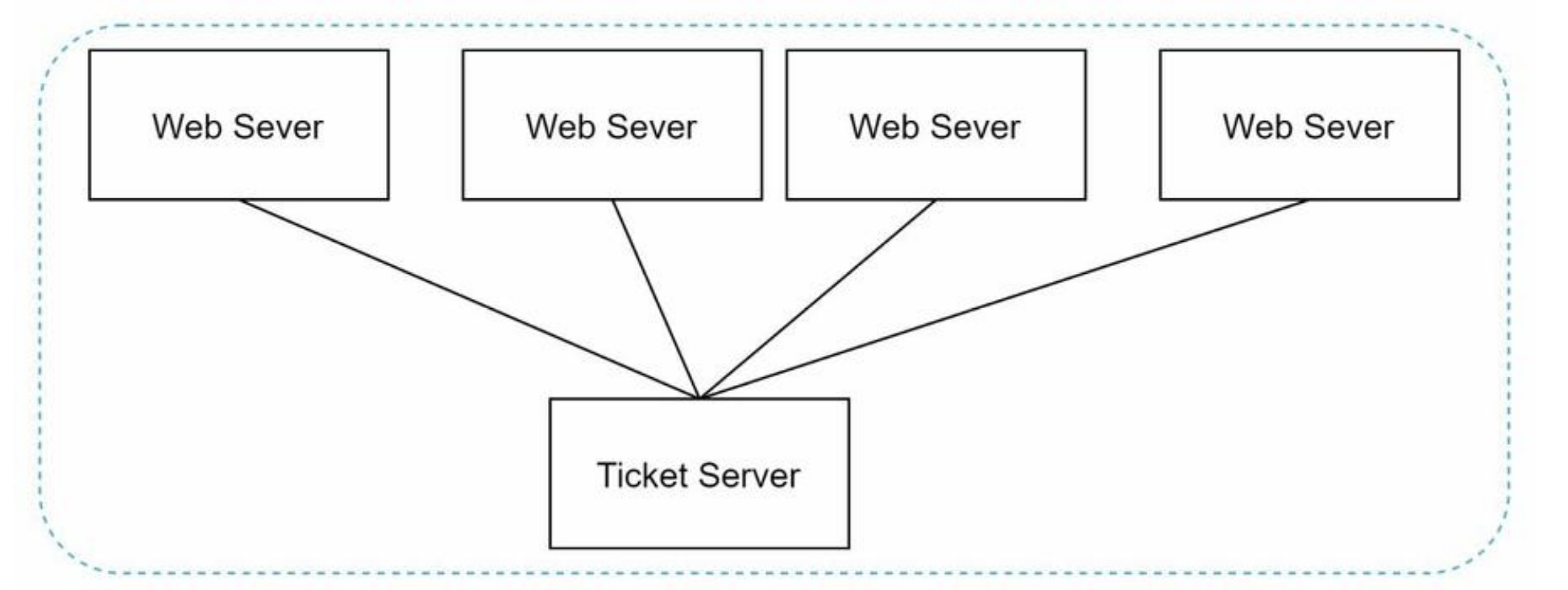
**High-level design**

1. **Multi-master replication**



* This approach uses the databases’ 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
* It does not scale well when a server is added or removed (need to update the counter for each server, otherwise will lead to collision)

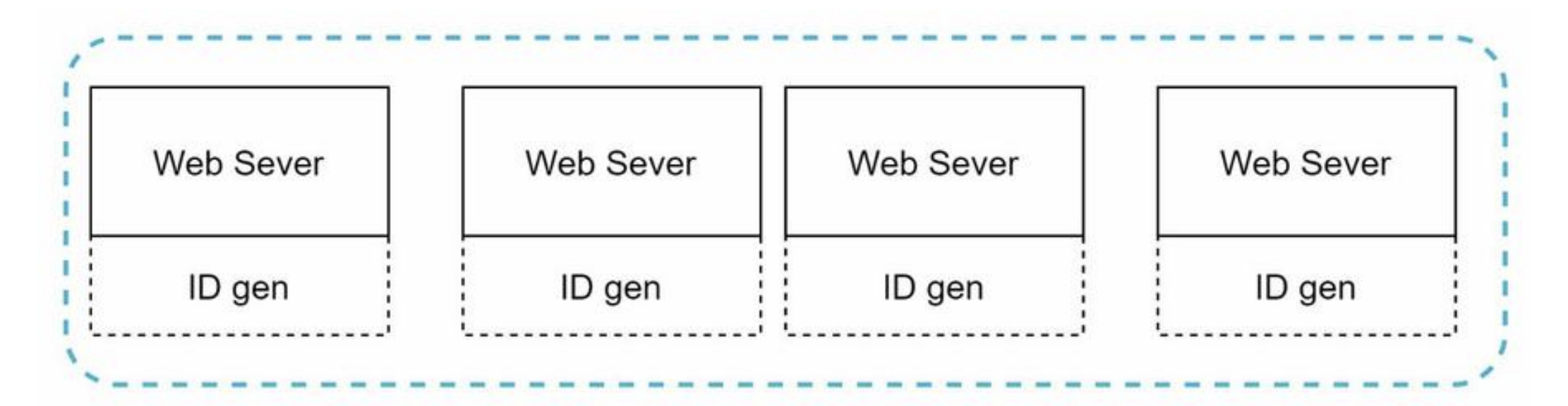
1. **Ticket Server**



* Ticket server acts as single point of failure
* To mitigate single point of failure, can have multiple ticket server – then will require to synchronise these servers

1. **UUID**

* UUID is a 128-bit number used to identify information in computer systems. UUID has a very low probability of getting collusion, by using a library
* OS have inbuilt functions to support this.
* E.g UUID: 09c93e62-50b4-468d-bf8a-c07e1040bfb2



* Generated ids are alphanumeric, if requirement is for numeric this will not solve the problem, sorting can take long time.

1. **Twitter snowflake approach**

* Let’s talk about Divide and conquer technique
* Instead of considering ID as a single entity, lets divide the id into multiple parts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 Bit | 41 Bits | 5 bits | 5 bits | 12 bits |
| Sign Bit | Timestamp - Epoch | Datacenter Id | Machine Id | Sequence number |

Sign bit 🡪 It will always be 0, can be used for the future

Timestamp 🡪 Milliseconds since the epoch

Datacenter ID 🡪 2 ^ 5 = 32 datacenters

Machine ID 🡪 2 ^ 5 = 32 machines per datacenter

Sequence number 🡪 For every ID generated on that machine/process, the sequence number is incremented by 1. The number is reset to 0 every second.

Datacenter IDs and machine IDs are chosen at the startup time, generally fixed once the system is up running

Timestamp and sequence numbers are generated when the ID generator is running.

As timestamp is dependent on the epoch, this is the most important number and will also make it easier to sort the IDs

One **Problem** of the above system

**Clock synchronization**

As there are multiple servers running, there time might be skewed, one of the best ways to solve this is to use NTP (Network time protocol).