Topic- Rate Limiter

what is rate limiter?

→ Control the rate of traffic it could by on basis of anything eg. user_id, ip etc

why do we need it?

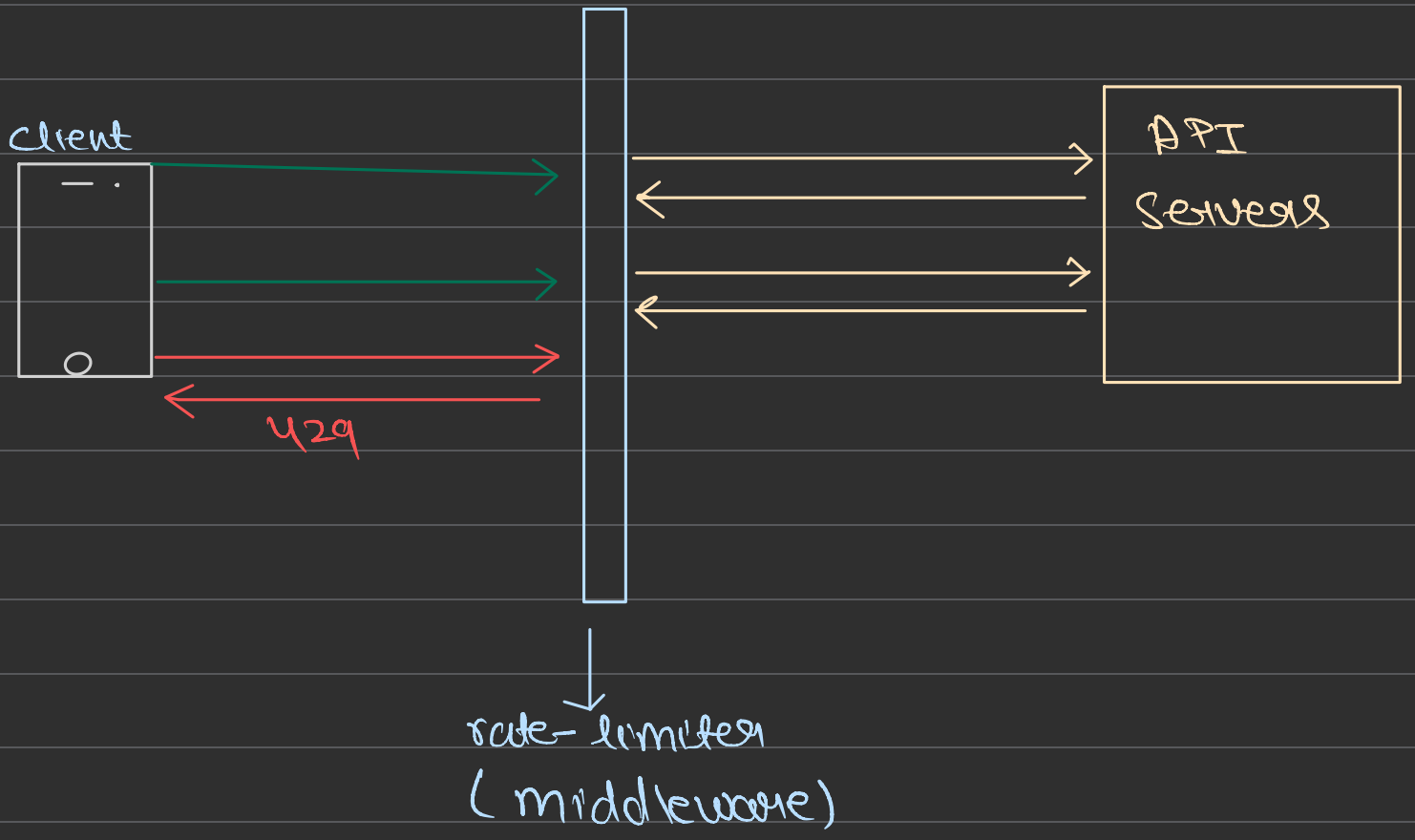
→ Server wont be overloaded.

→ Prevents fro DDOS attack.

Example-

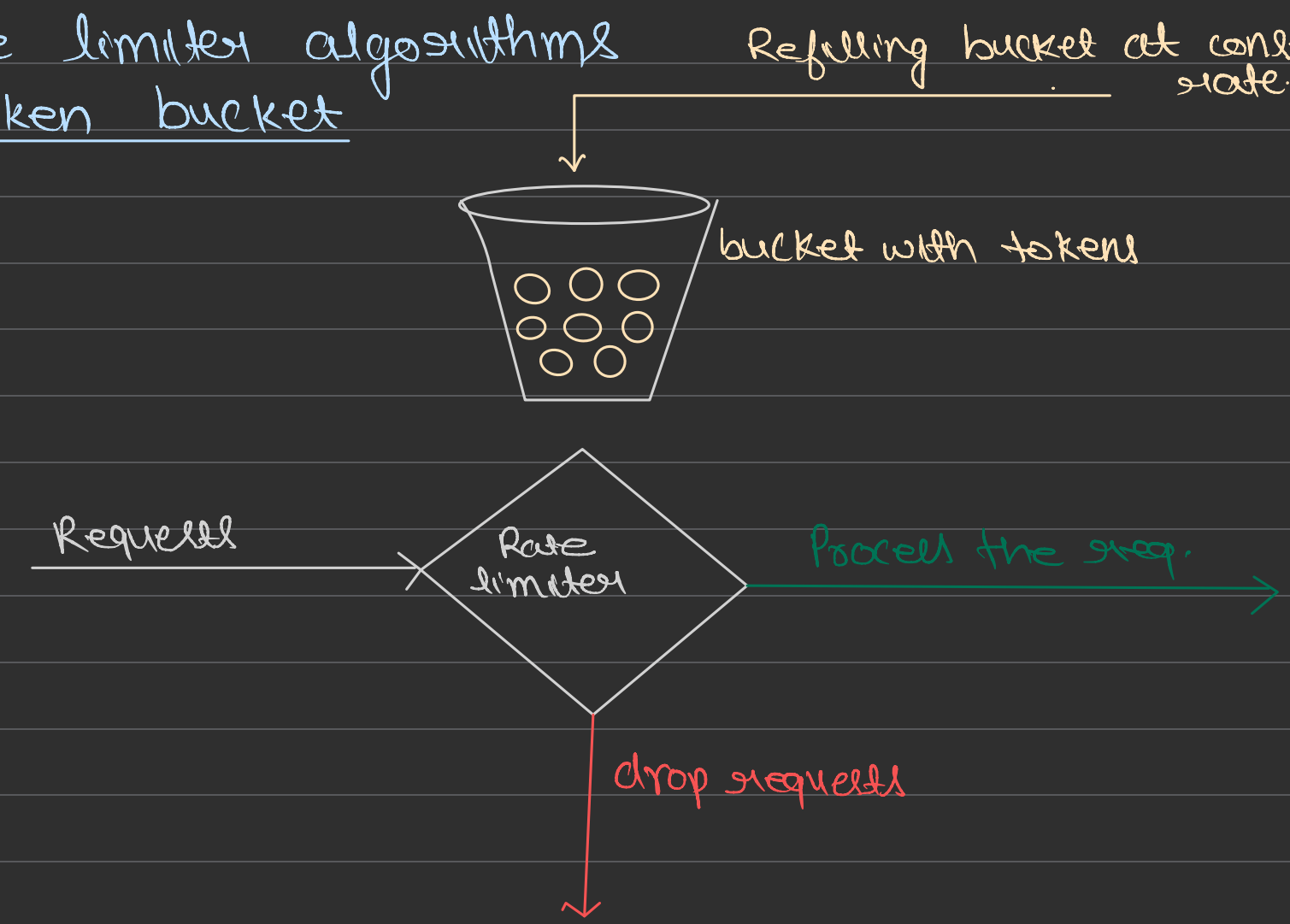
every user can't more than 10 requests / sec

API rate limiter.



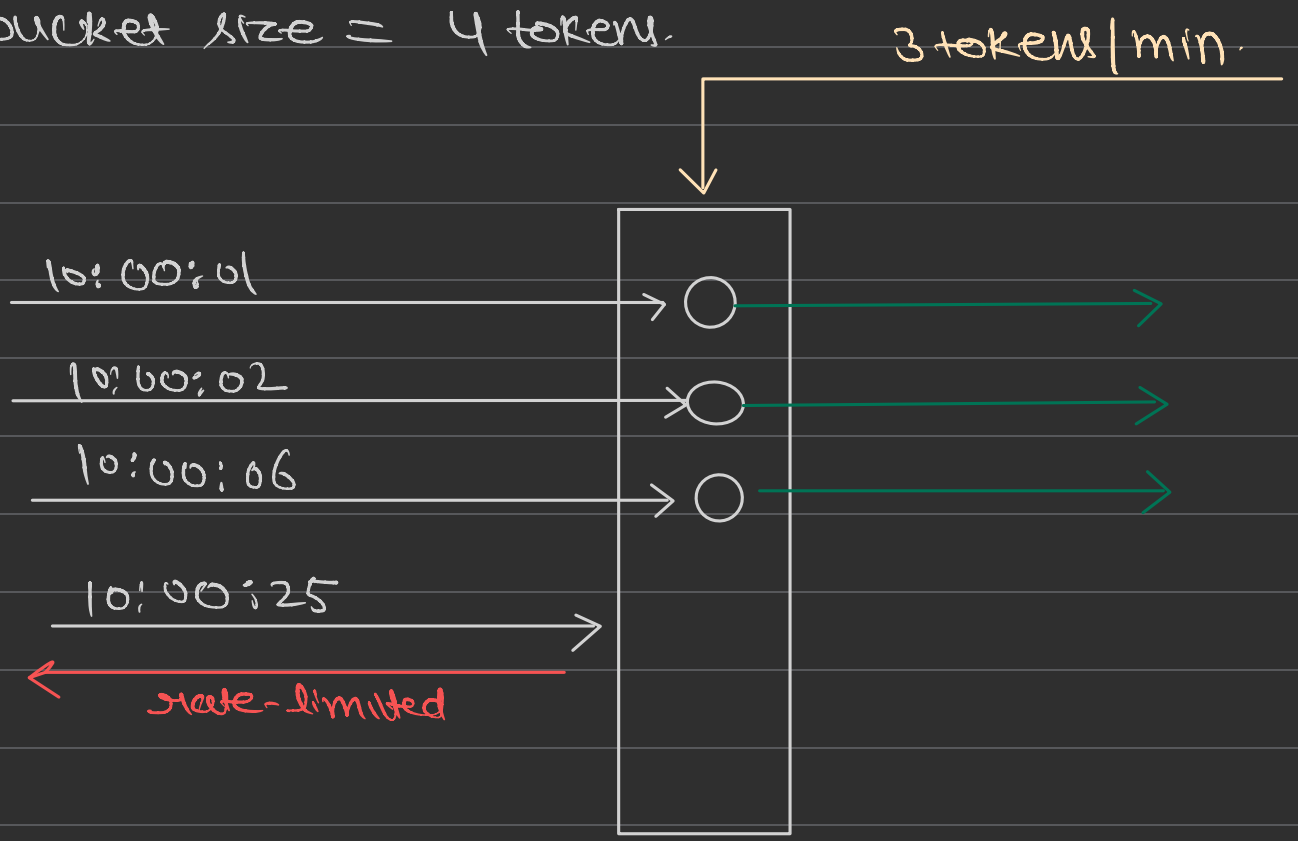
Rate limiter algorithms

① Token bucket



Eg. refill rate = 3 tokes | minute

bucket size = 4 tokens.



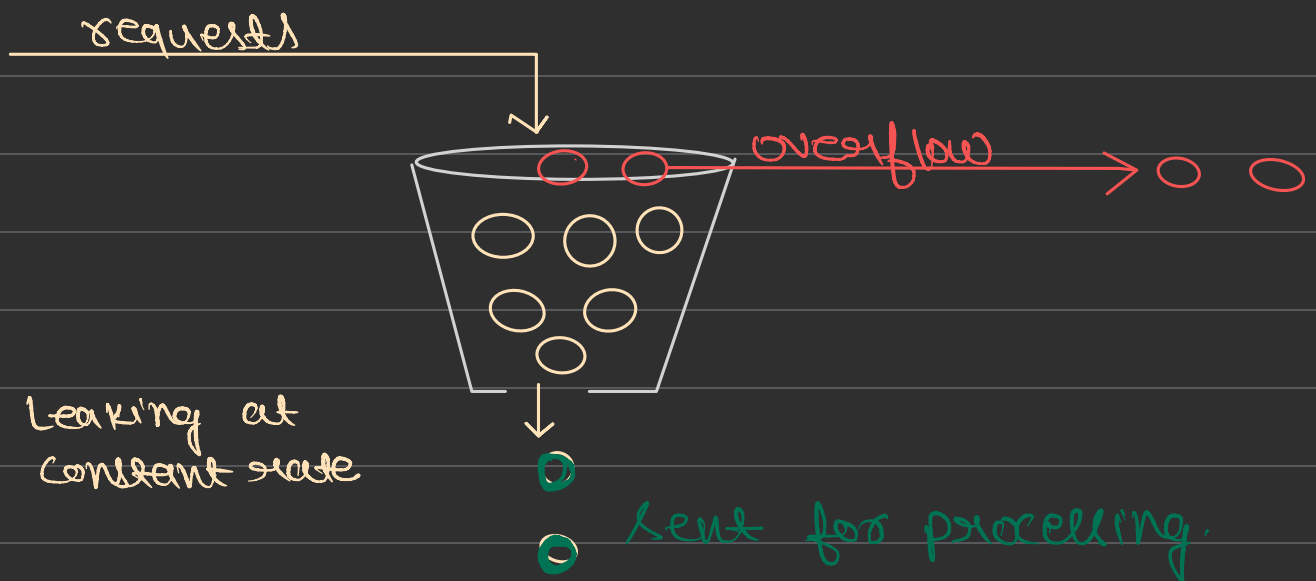
ISSUE: At the time unit boundaries, we may twice the traffic.

① Refil rate decides Avg traffic to your server.
② Bucket size signify Peak traffic in a unit of time.

eg. refil rate = 4 token / min.
Bucket size = 16 token.

② Leaking bucket.

Bucket with a given capacity that is leaking at a constant rate. i.e.



Can be implement via queue, whenever a req. comes —

→ If queue isn't full, add them to the queue.
else drop the request.

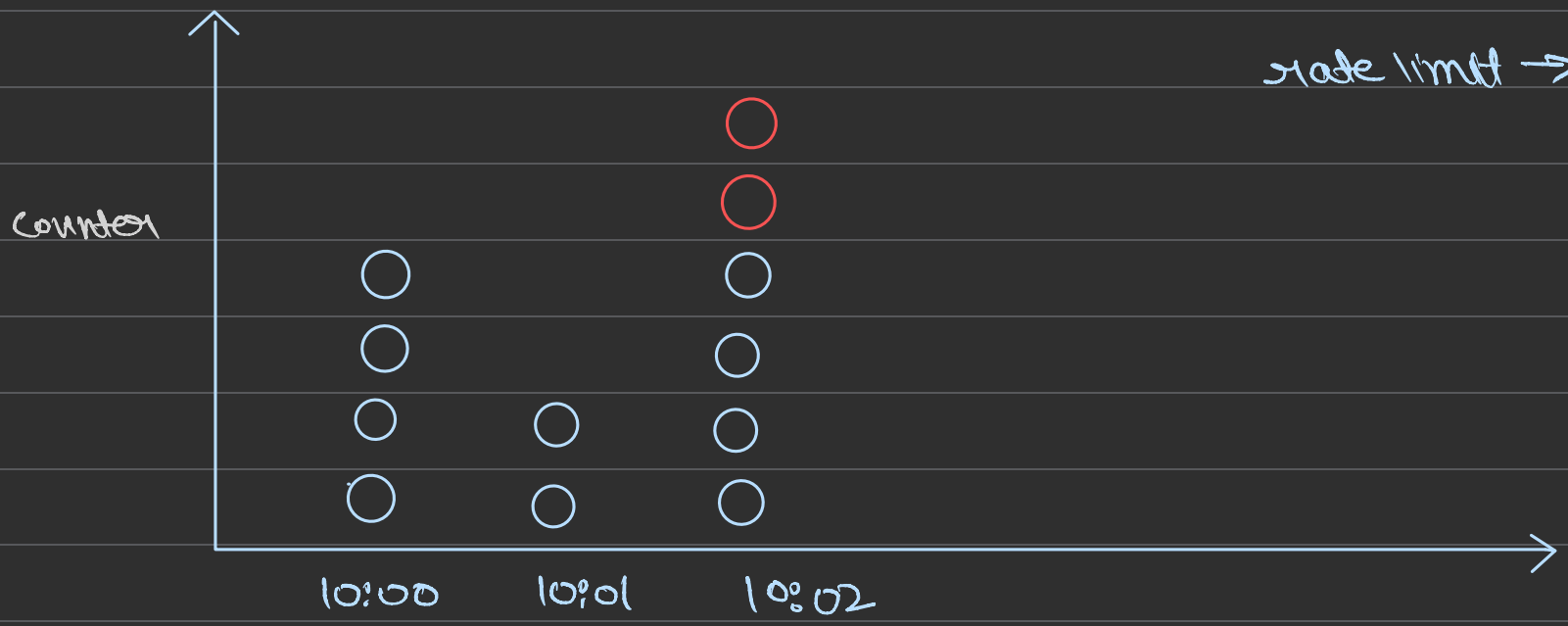
Also server will keep consuming requests from the queue at a constant rate.

Issue → burst traffic can quickly fill the queue & all subsequent req. will be rejected.

③ fixed window counter

① Create a counter for time window.

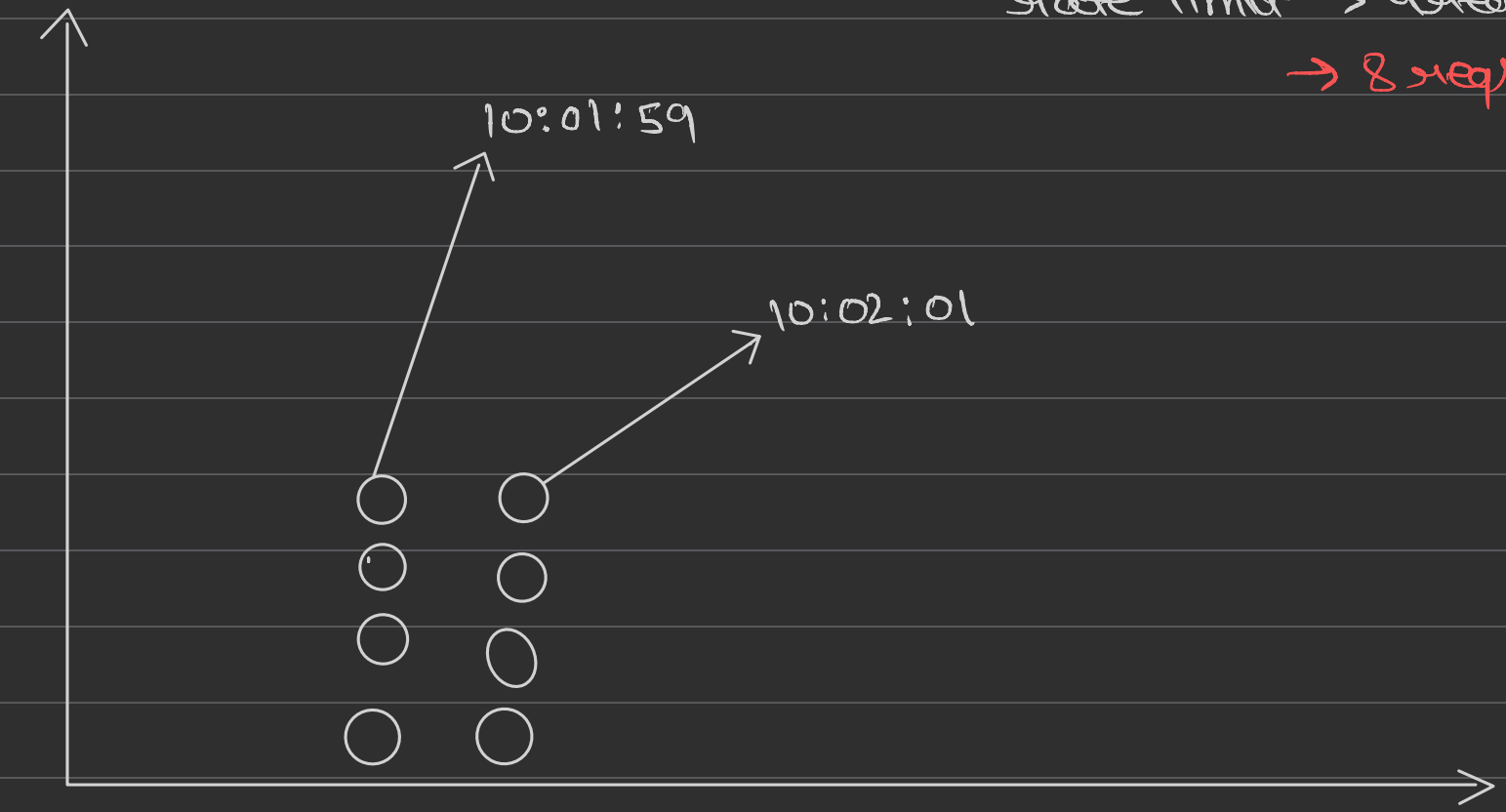
② When req comes, identify the counter and try to increase the counter, if its less than threshold.



Issue → burst traffic

rate limit → 4 req | min.

→ 8 reqs | min.



④ Sliding window log → fixed the issue of static window counter.

① keep log of requests timestamp.
② whenever new req. comes, simply add it in the log.
③ remove expired timestamp.
④ if size of log is within threshold, then accepts the req. else reject.

Let's take an example.
→ log threshold 3
→ time unit is minute

$r_1 \rightarrow$ 10:00:05

$r_2 \rightarrow$ 10:00:09

$r_3 \rightarrow$ 10:00:50

let say, r_4 comes at 10:01:06

~~$r_1 \rightarrow$ 10:00:05~~ expire.

$r_2 \rightarrow$ 10:00:09

$r_3 \rightarrow$ 10:00:50

$r_4 \rightarrow$ 10:01:06

let say, r_5 comes

~~$r_1 \rightarrow$ 10:00:05~~ expire.

$r_2 \rightarrow$ 10:00:09

$r_3 \rightarrow$ 10:00:50

$r_4 \rightarrow$ 10:01:06

$r_5 \rightarrow$ 10:01:07 $\rightarrow$ drop

⑤ Sliding window counter

Combination of sliding window log + fixed window counter.

① It always keep track of no. of req. in prev min.

② When a req. comes, let say at x^{th} fraction of minute. It finds Approximate traffic of sliding window.

no. of req. in prev minute X fraction of prev. minute while considering sliding window
+ no. of req. in current minute.

Let say - rate limit = 10 req. / min.

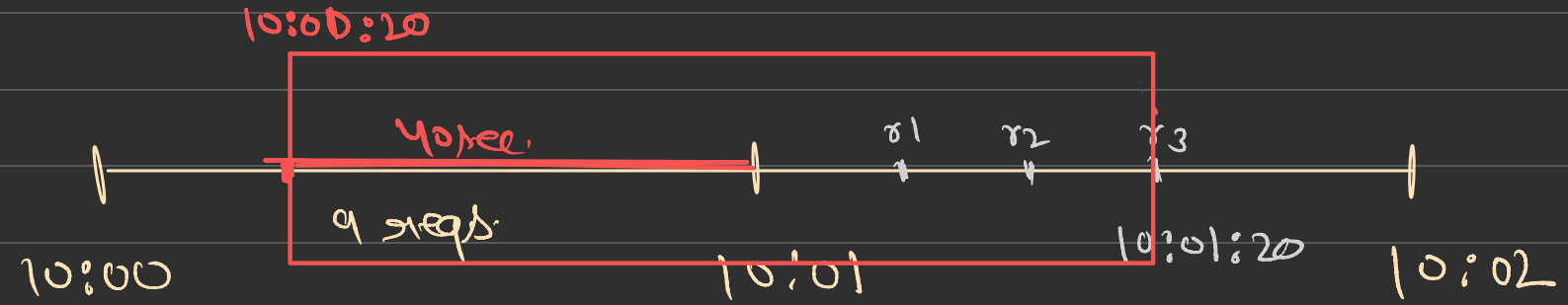
no. of req. in prev min (10:01) → 9 req.

third req of current min comes at → 10:02:20

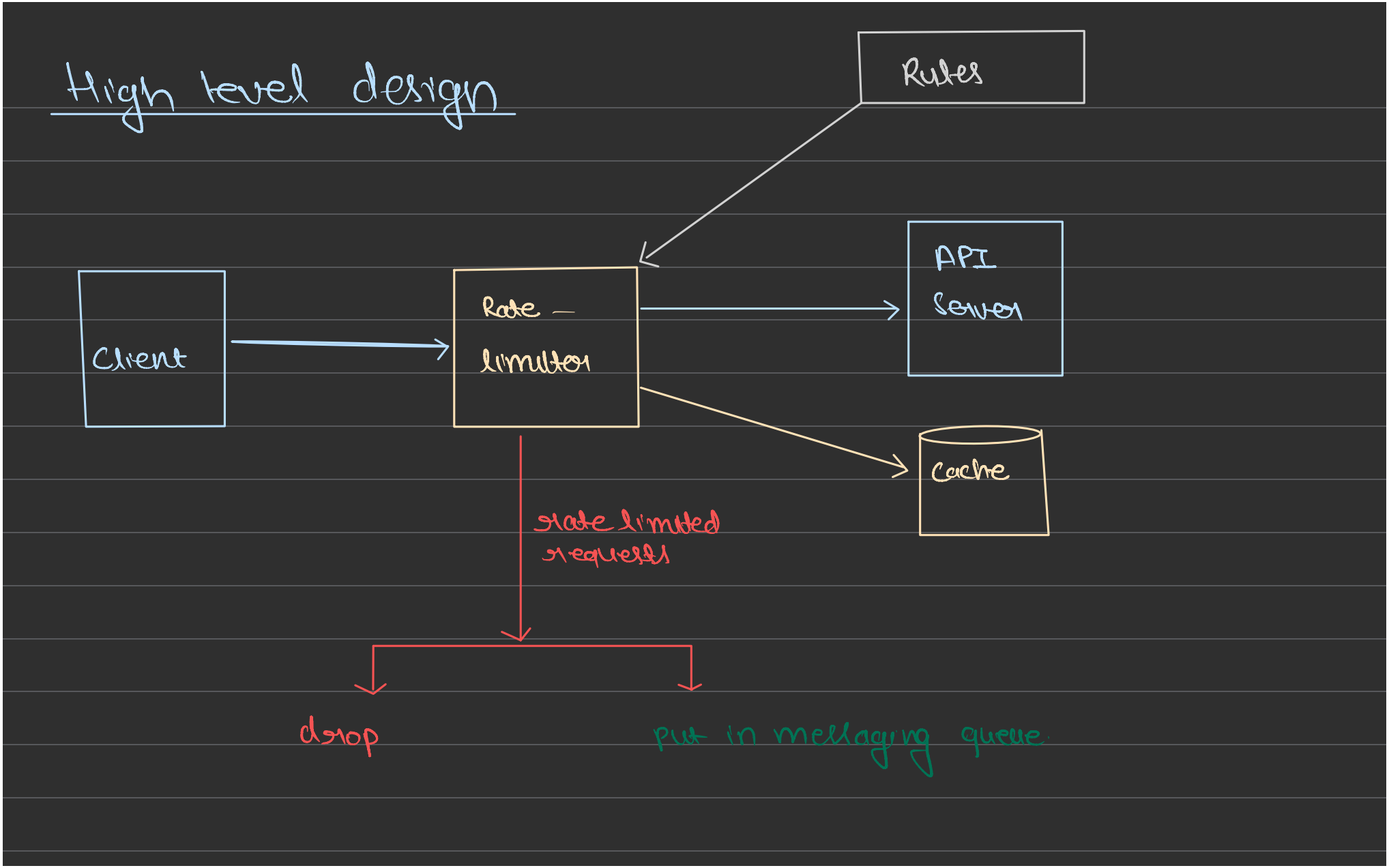
fraction of prev min in current sliding window = $\frac{2}{3}$

so ~~total~~ approximate req in current sliding window

$= 9 \times \frac{2}{3} + 2 + 1 = 9$



High level design



Challenges with distributed rate-limiter.

① Race condition → Take locks while inc the counter.

let say $x = 3$, two concurrent thread execute $x = x + 1$

we may end up with $x = 4$, instead of 5

⑤ Synchronization

Let say if we have many instance of rate limiter running. challenge is to have a same data shared on each instance.

→ Centralized cache (Redis)

10/min.

r1 — 10:00:57 c=1

r2 — c=2

r3 — c=3

r4 —

10:01:57 = c=0 c=1

10:02:56 → 9 c=10

10:02:57 → c=1

every rate-limiter host have its own cache