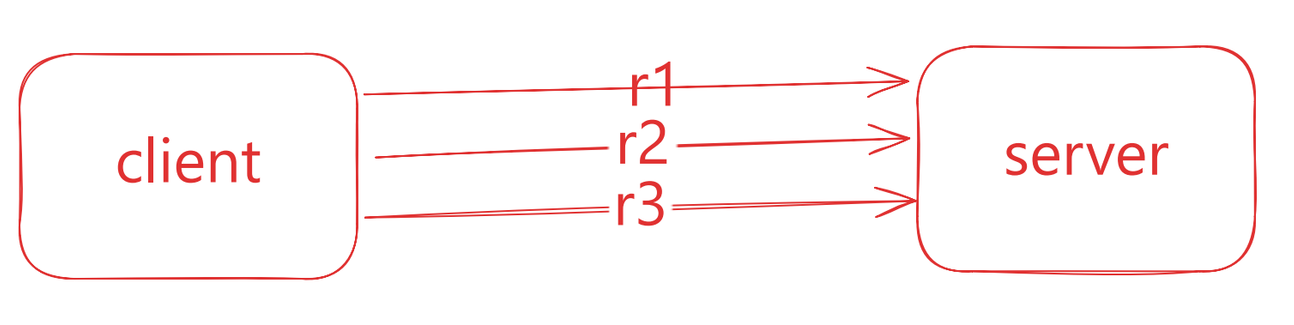
Rate limiting   No.of times that a client can request a server in any timefr
Caching Strategies

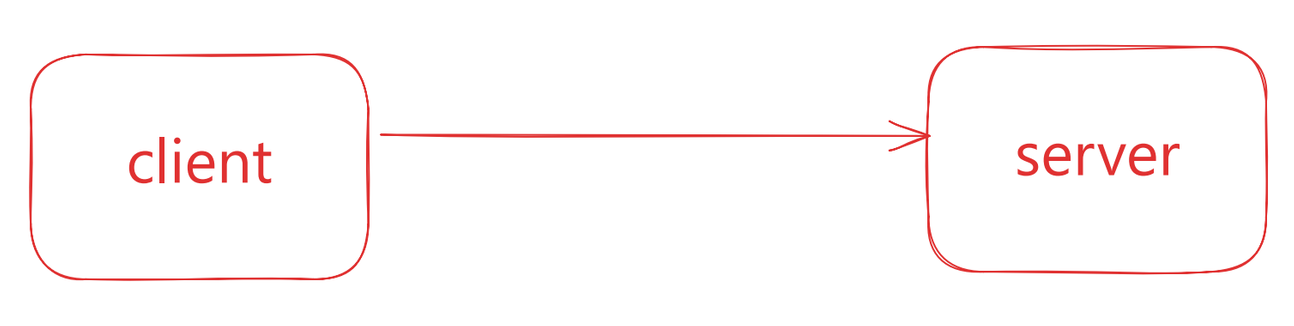
Rate limiting



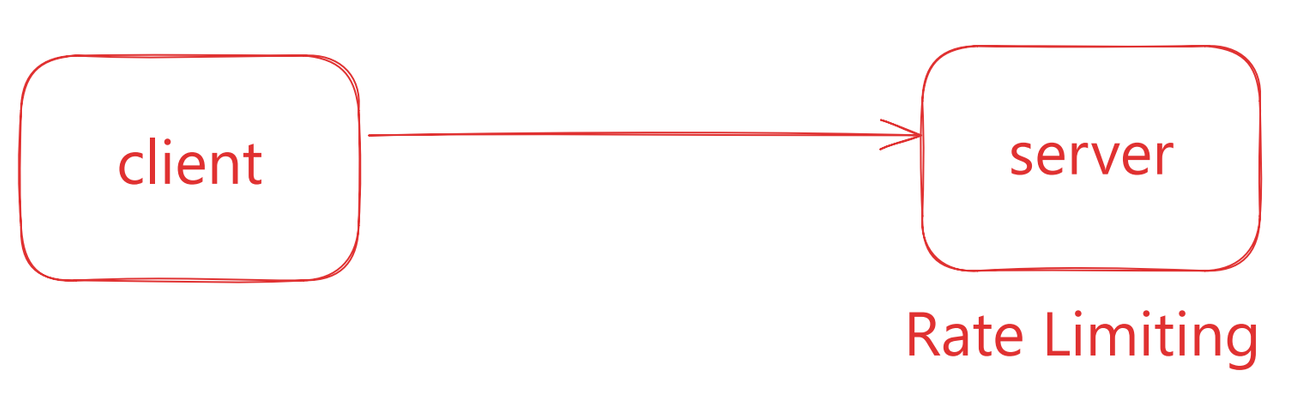
problems:
1. Client can send too many request within a timeframe.

DDOS/DOS

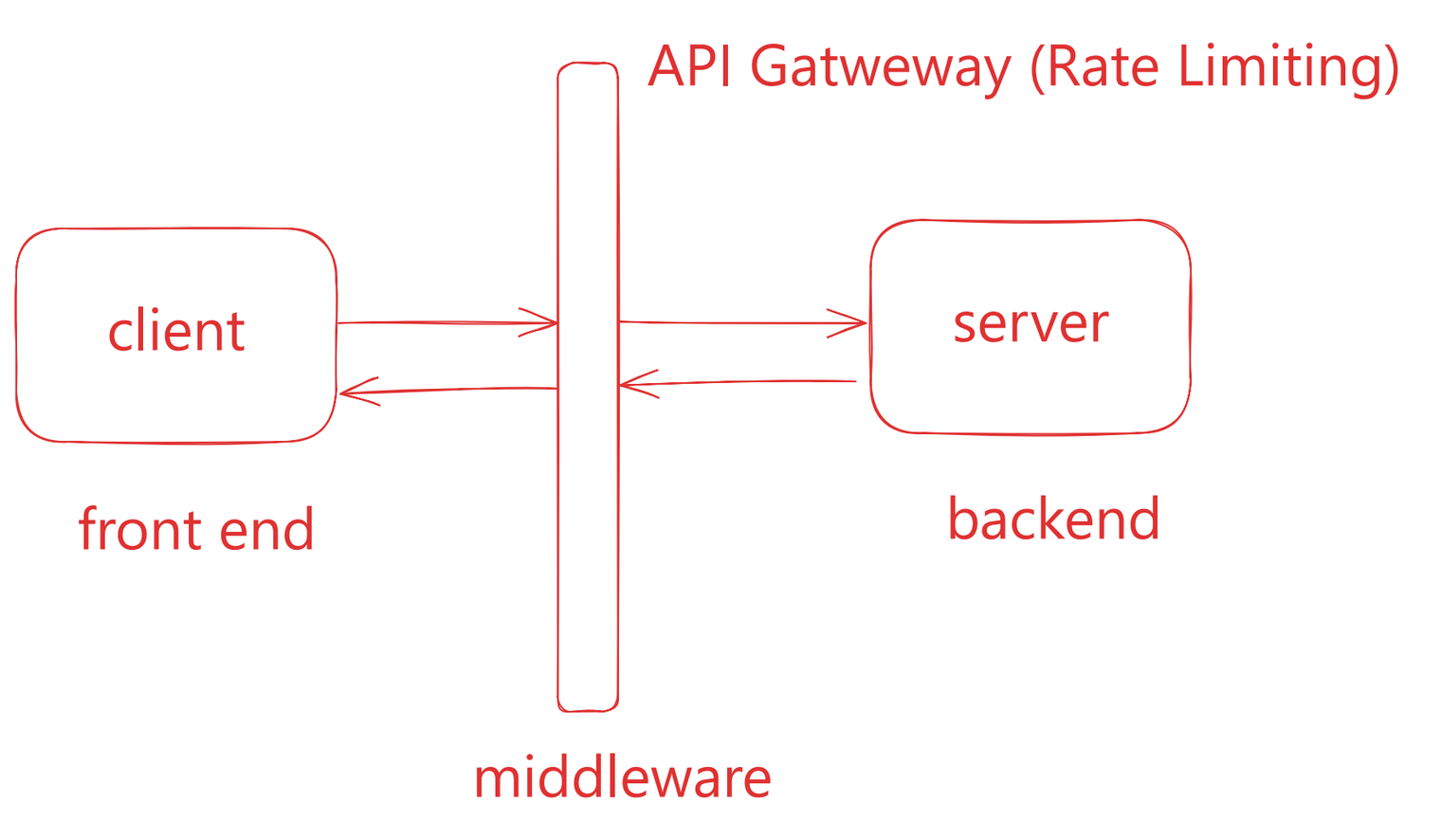
2. Server cost expensive
Each req is expensive.



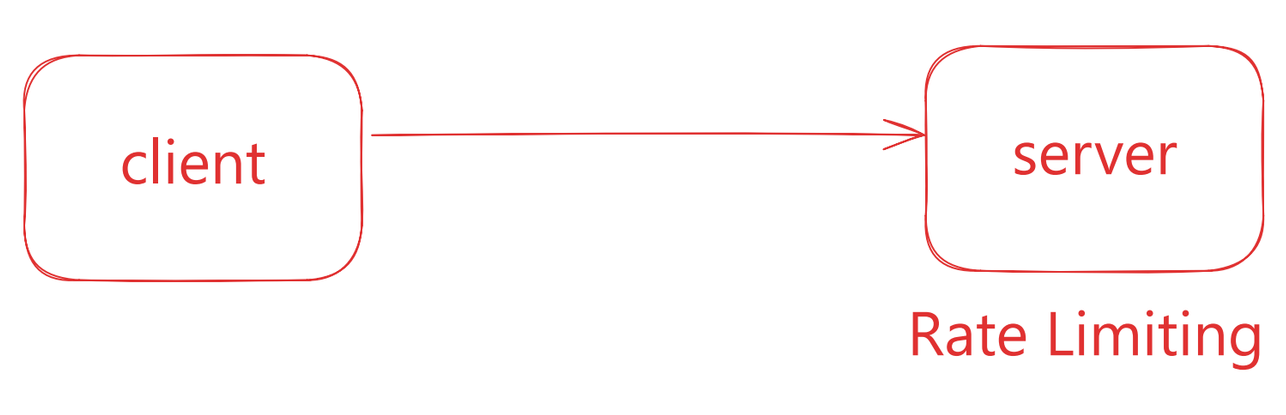
How many req client can send in any timeframe



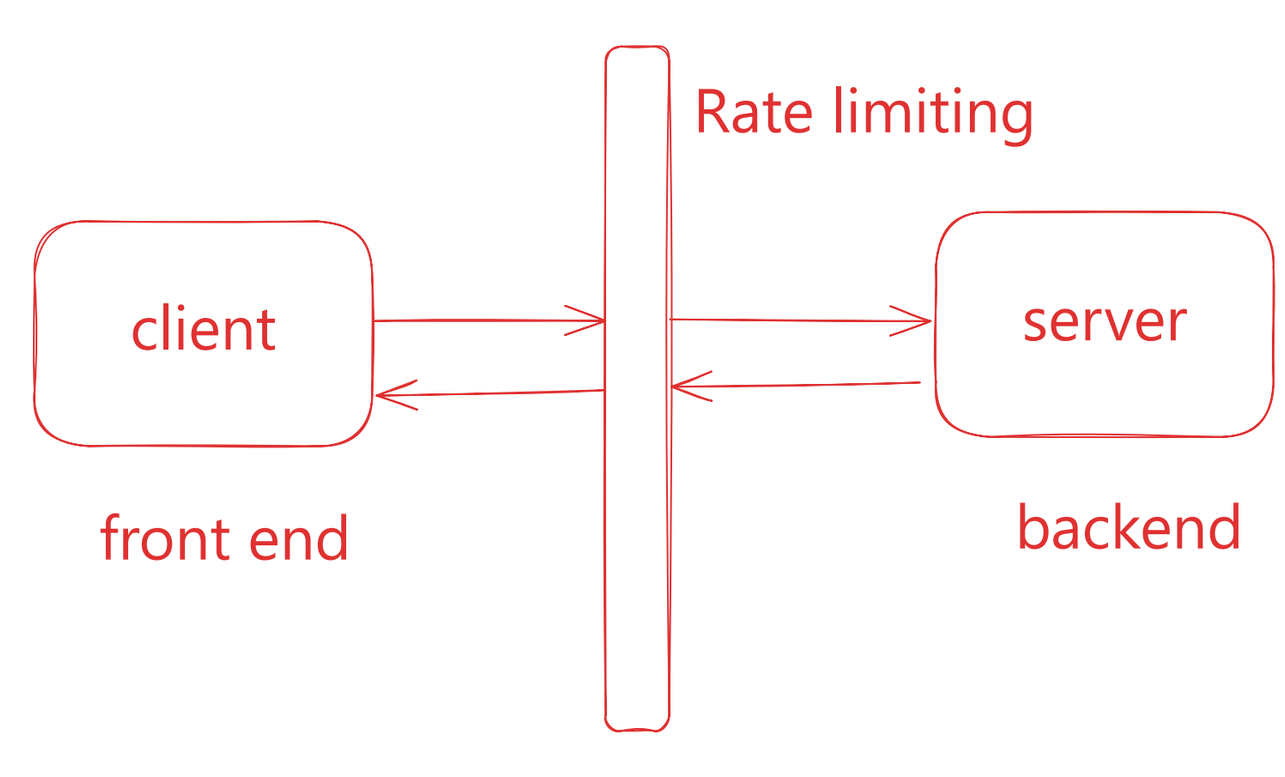
Rate limiting can be implemented
1. Client side : worst   client is publicly availab
2. Server side
3. Middleware



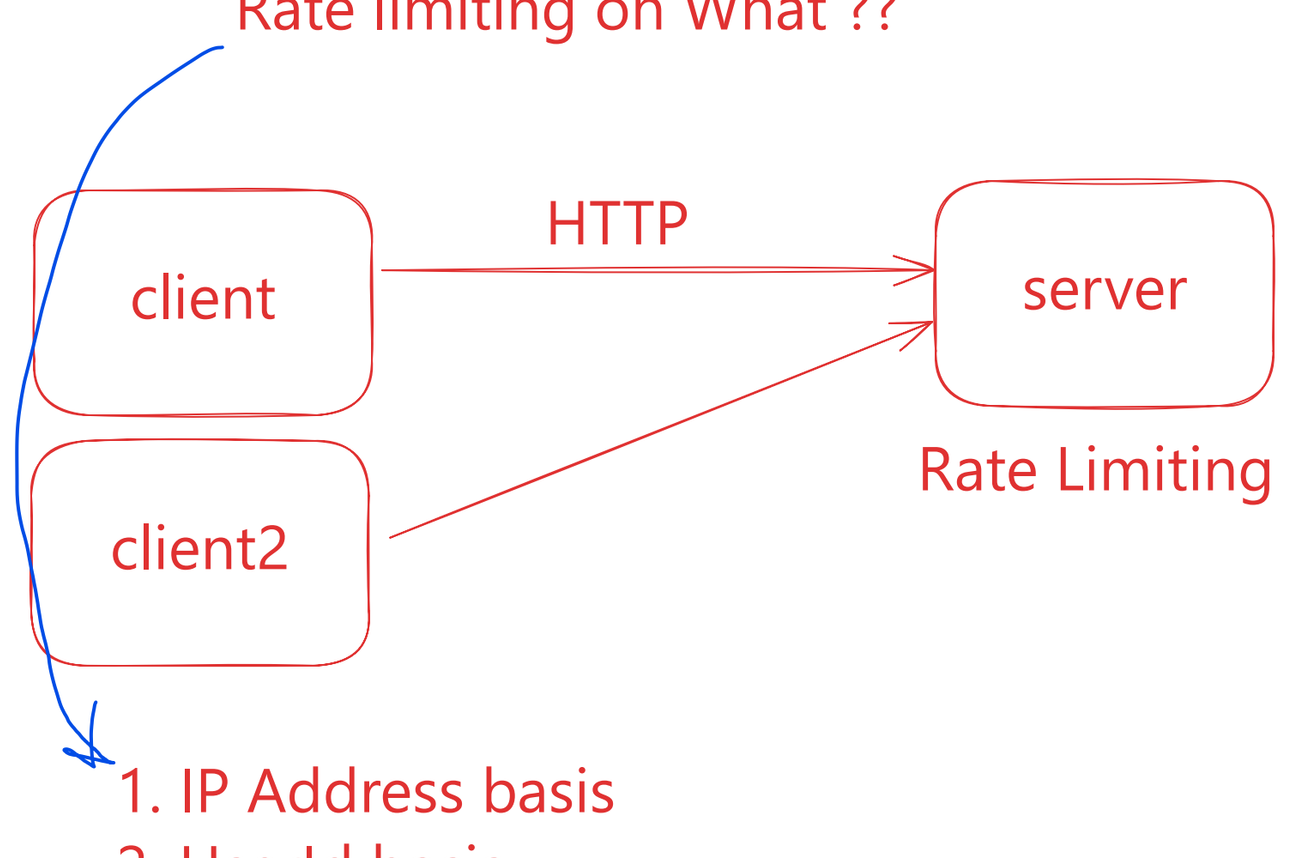
Appilication use case:
1. Server side language(Java, C++, javascript, Python)   should be fast
Server side



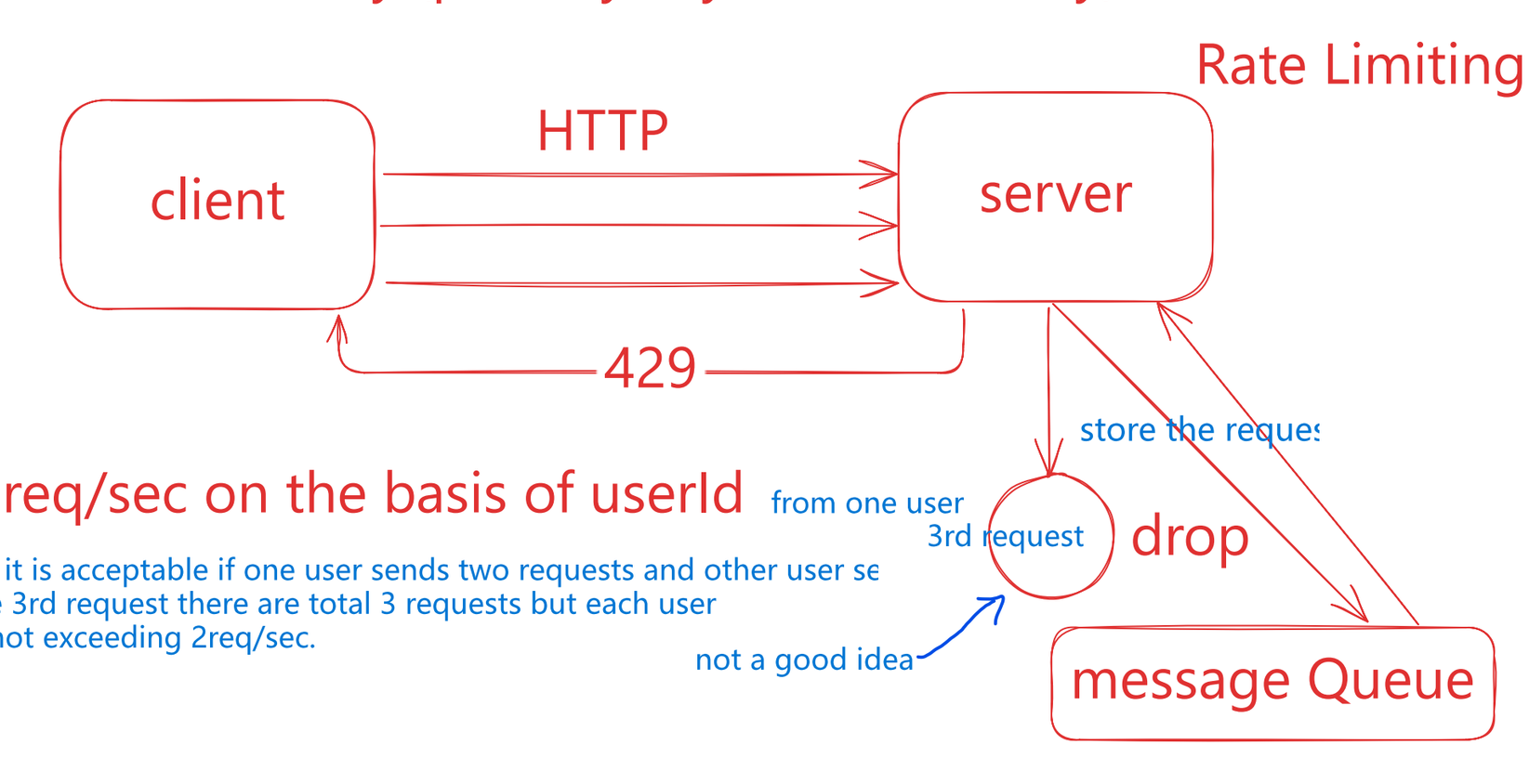
2. Generally 3rd party Rate limiter providers.
Amazon aws. (API Gateway)



Rate limiting on What ??



1. IP Address basis
2. UserId basis
3. other way (primary key, candidate key)



2 req/sec on the basis of userId   from one user
i.e. it is acceptable if one user sends two requests and other user se the 3rd request there are total 3 requests but each user is not exceeding 2req/sec.

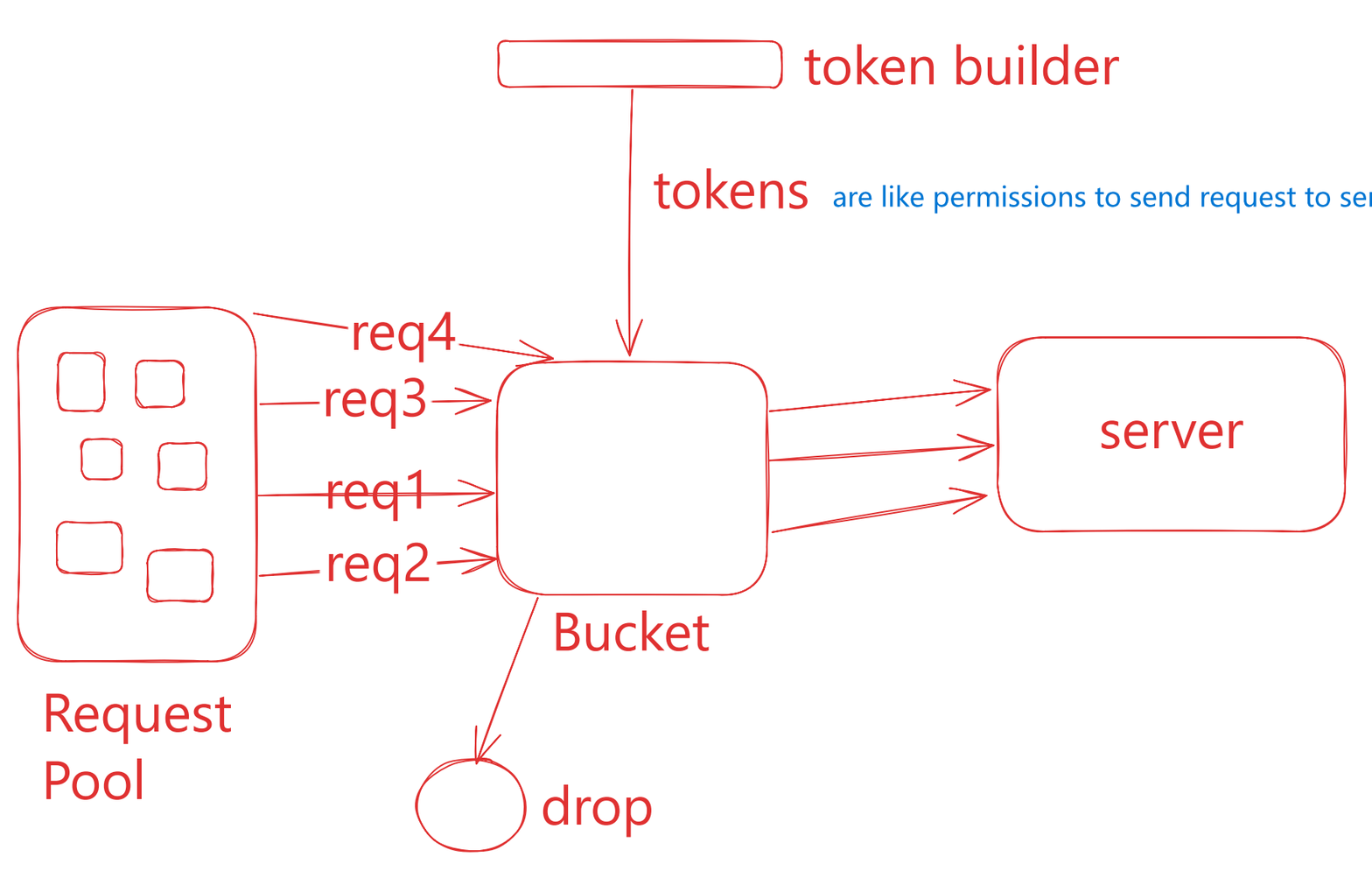
Response codes:
200 : Ok
429 : Too many Request

Headers:
X-api-limit : 2
x-duration : seconds

Algorithms of Rate Limiting

1. Token Bucket
2. Leaky Bucket
3. Fixed window counter
4. Sliding window log
5. sliding window counter

Token Bucket Algorithm



1. Bucket size : 20
2. Inflow : 10 tokens/sec

1. Token builder will push tokens in fixed intervals
2. If bucket size is full all the new tokens will be overthrown (over flow).
3. whenever a req comes it takes a token from bucket if available and go to the server
4. if bucket is empty, req is dropped.

use : Amazon   FMCG

PRos :
1. Simple to implement   DDOS attacks
2. Can handle burst traffic for small duration

COns:
1. To decide the bucket capacity and inflow

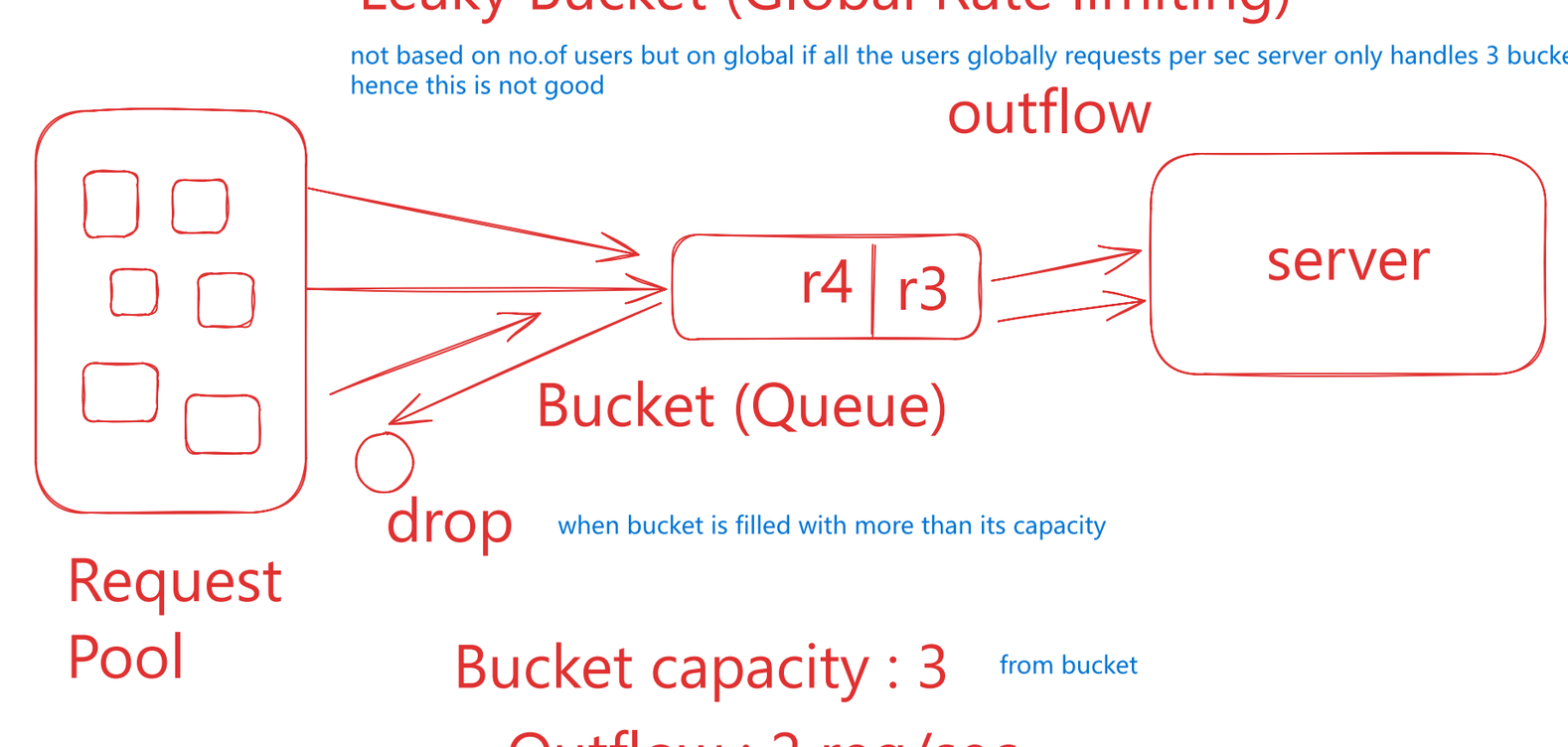
same for token based bucket its token capacity is high hence it can handle burst

when you want your application to not handle burst traffic then you can use leaky

Leaky Bucket (Global Rate limiting)
not based on no.of users but on global if all the users globally requests per sec server only handles 3 buckets per sec hence this is not good

i.e. you want server to serve the requests at its own server outflow ca

App : 1000 users   if based on userIds then you have to have bucket for each u which can be a lot of buckets (same for token based buckets
IP level : every unique limit



Bucket capacity : 3   from bucket
Outflow : 2 req/sec   bucket outflow

Shopify

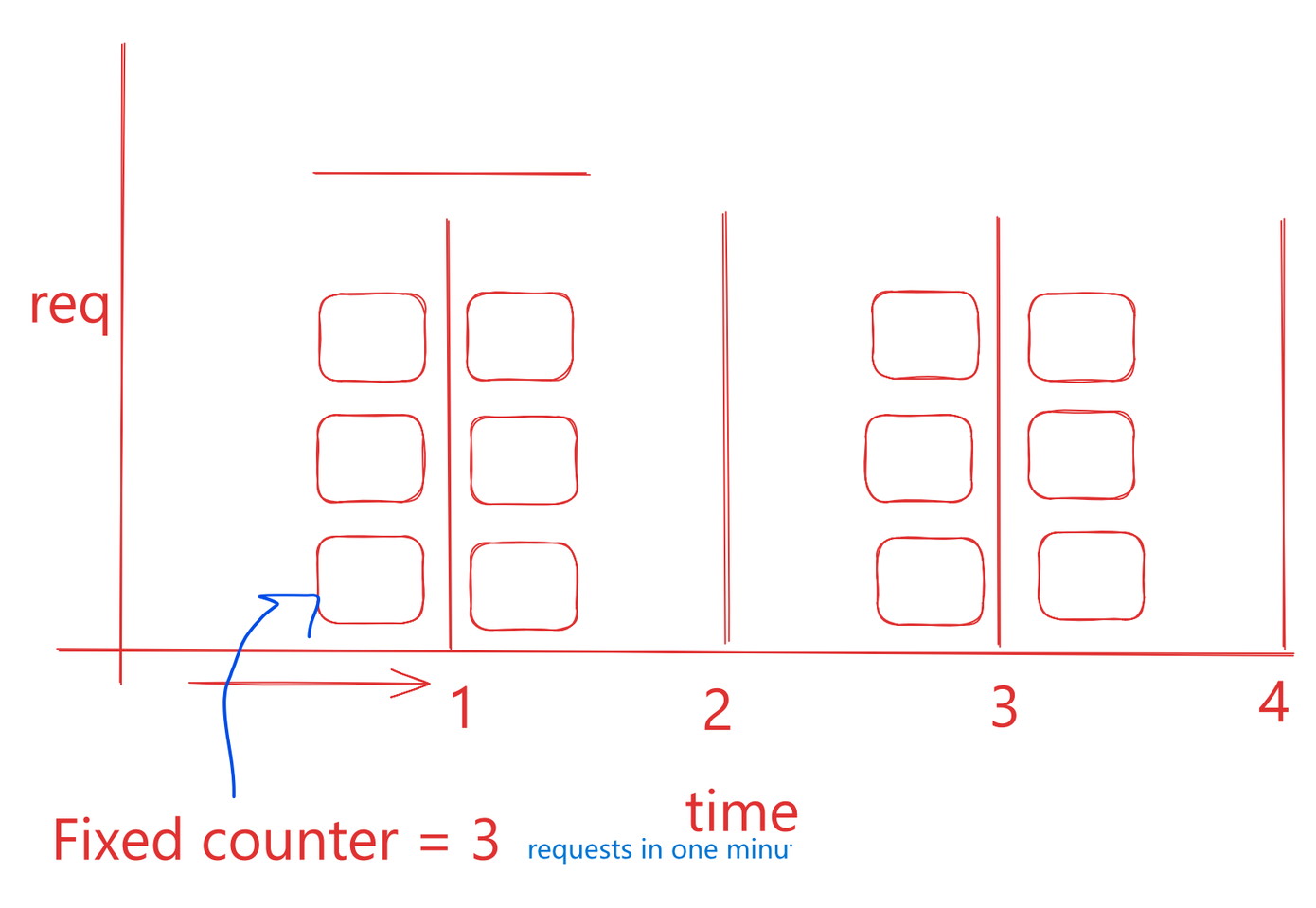
Pros :
Simple to implement
Even though burst traffic is there.
our server will not crash

Cons: unwanted requests in the queue by DDOS attacks
If DDOS/DOS happens, our server misses valuable Request (Main req --> starve)
high latency

APPlICATION :
100 APIS --> API wise   100 different buckets

FIXED WINDOW COUNTER

1. In a fixed interval (Say a sec, or a minute)
Only a spcific no. of req can come.



Fixed counter = 3   requests in one minu

Cons :
If burst traffic comes at edges of window, it may lead to server crash, high latency.

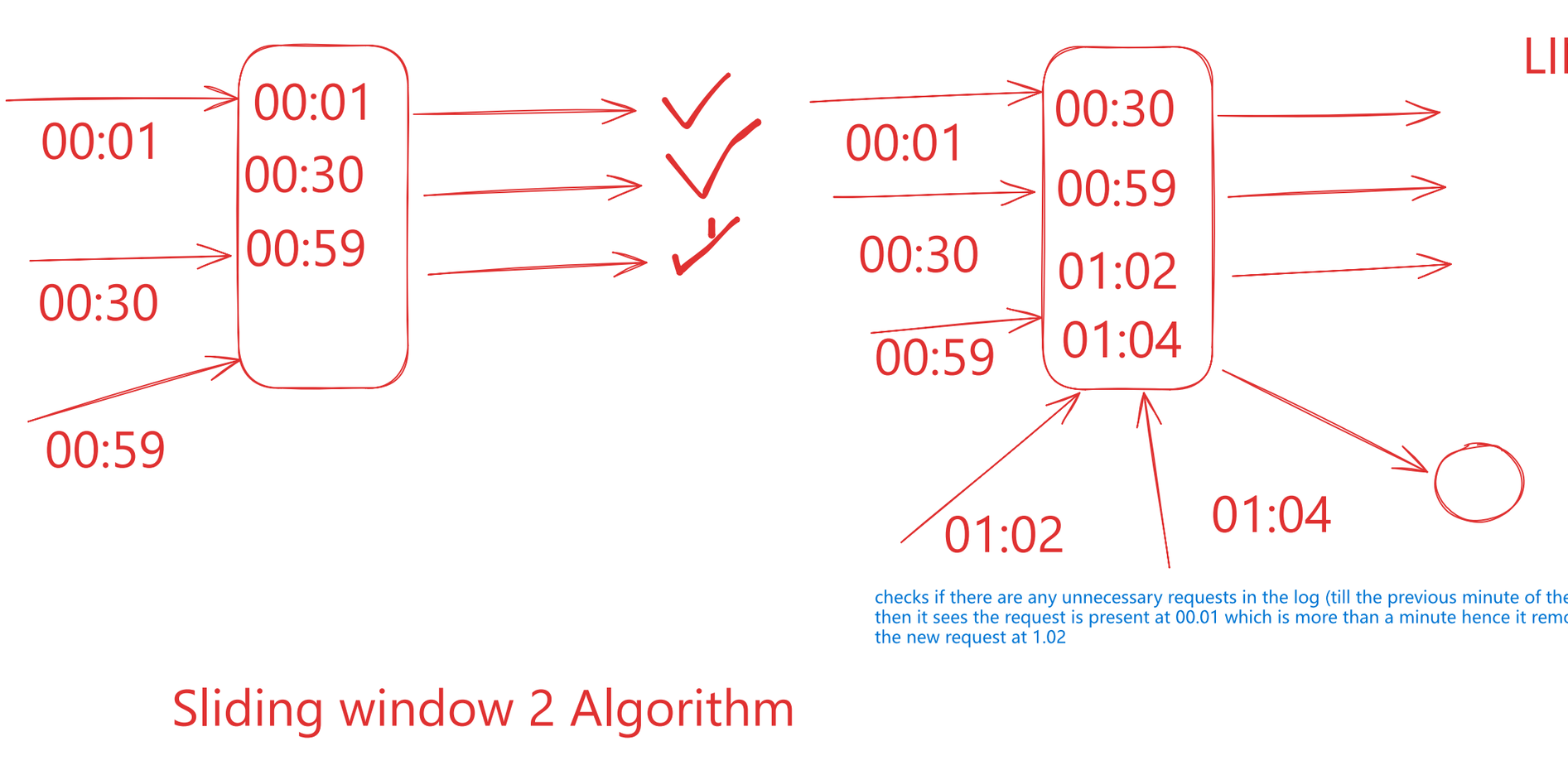
Sliding window Log Algorithm

Best Algorithm for Rate Limiting
Strict
cons   Slow & memory consuming

Algorithm:
1. It will store each req in a log file.
2. Whenever a req comes, it will first remove all the outdated req from the file.
3. THen it will log the new req and check, if the counter limit reached, drop the req else send it to server.

3 req/sec

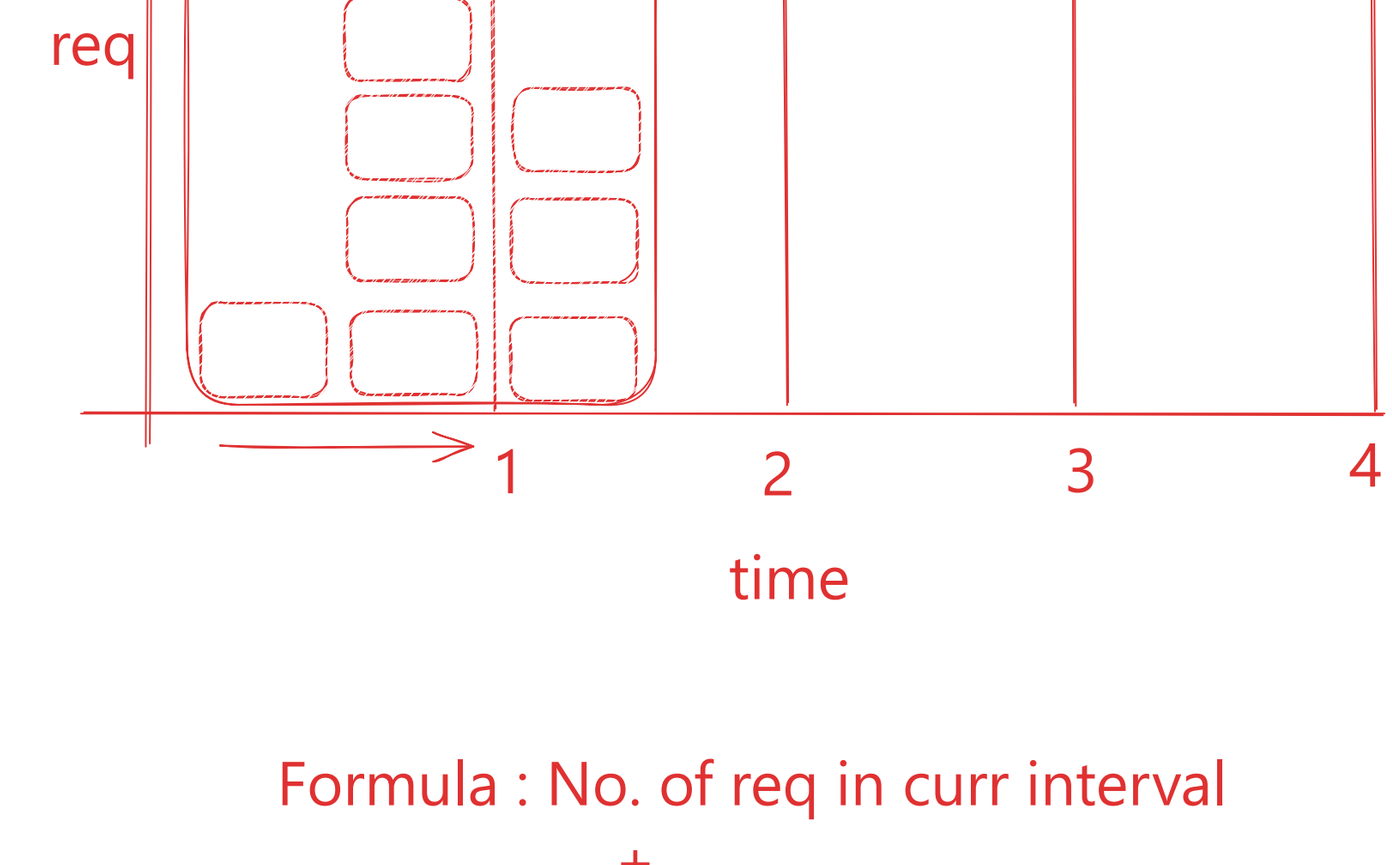
LINUX --> EPOCH timestamp



checks if there are any unnecessary requests in the log (till the previous minute of the request insertion i.e. 00.02) then it sees the request is present at 00.01 which is more than a minute hence it removes the 0.01 request from the log and insert the new request at 1.02

Sliding window 2 Algorithm

Hybrid approach -->   7 req/ minute



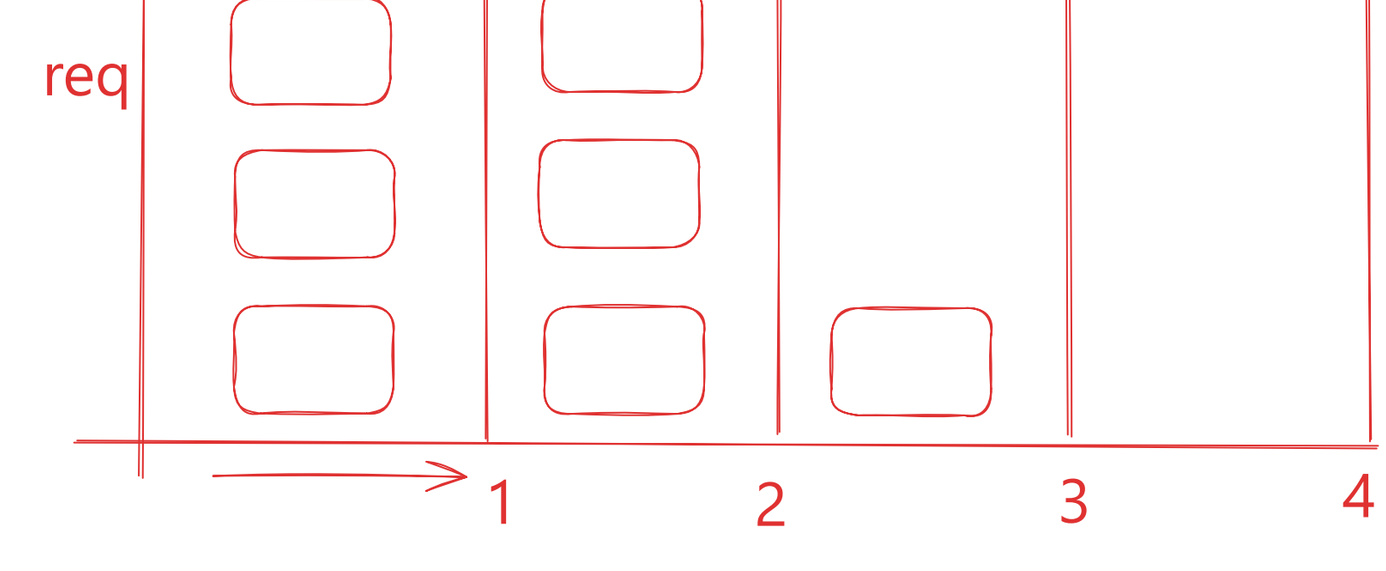
window size and window capacity depends on the server

Formula : No. of req in curr interval
+
No. of req in prev interval
*
70%   assumption as per the example

= 4 + 5 * 0.7 = 7.5   > 7.5 (hybrid approach) then drops the requ

Create our own Rate Limiter

Which algo to implement ?
--> Fixed window counter



Excepted Counter = 3
current counter = 3

Db ? SQL / NoSQl

Db : It is slow.

Cache : In-Memory Cache : Redis