HLD

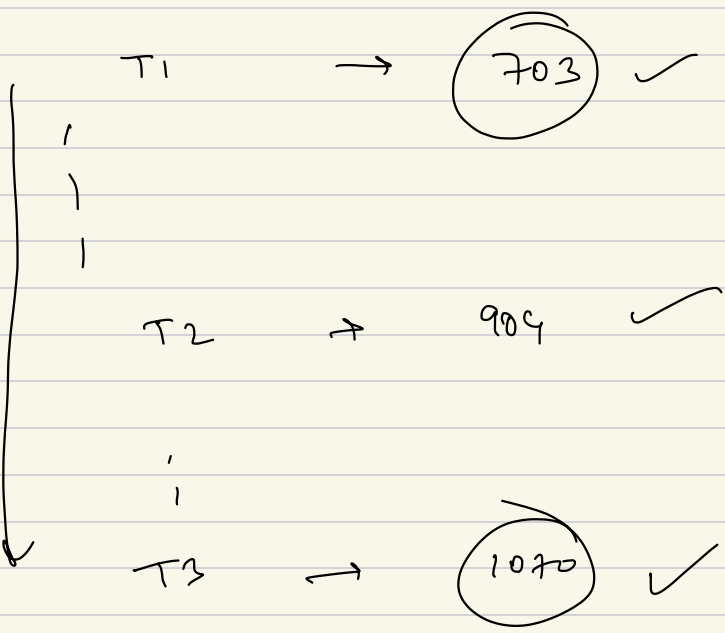
3|Jan|2024

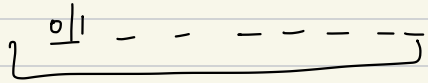
① Design unique Id generator

:) generate unique IDs (numeric ids - 8 bytes long)

→ ① Ids should be unique

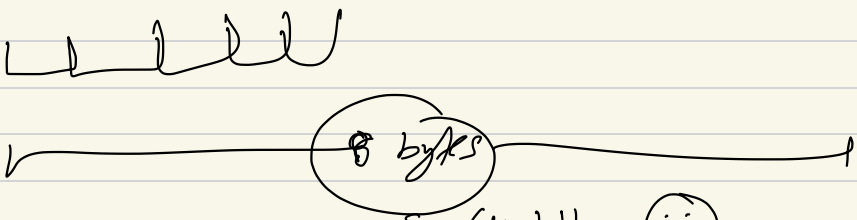
→ ② The ids generated must be incremental





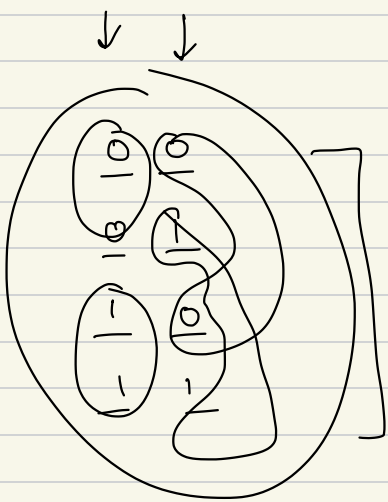
1 Byte = 8 bits ☺

=



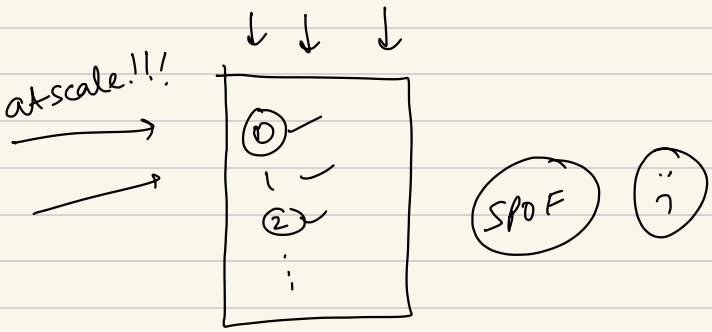
= 64 bits ☺

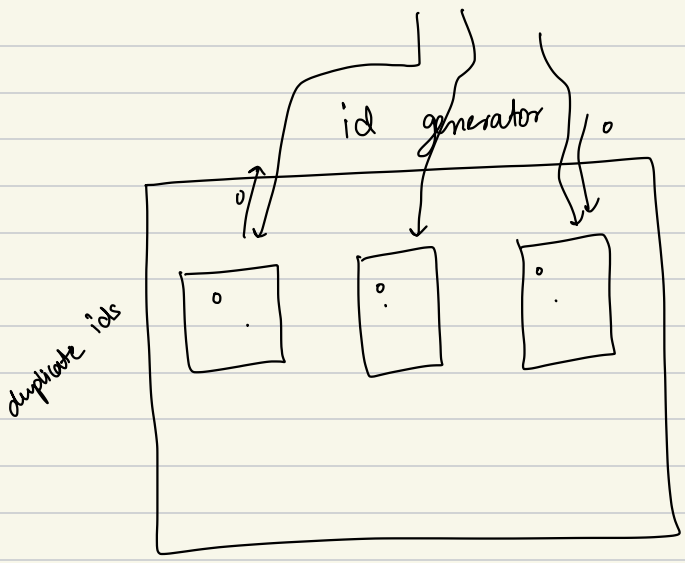
2^{64} = different values of ids ☺

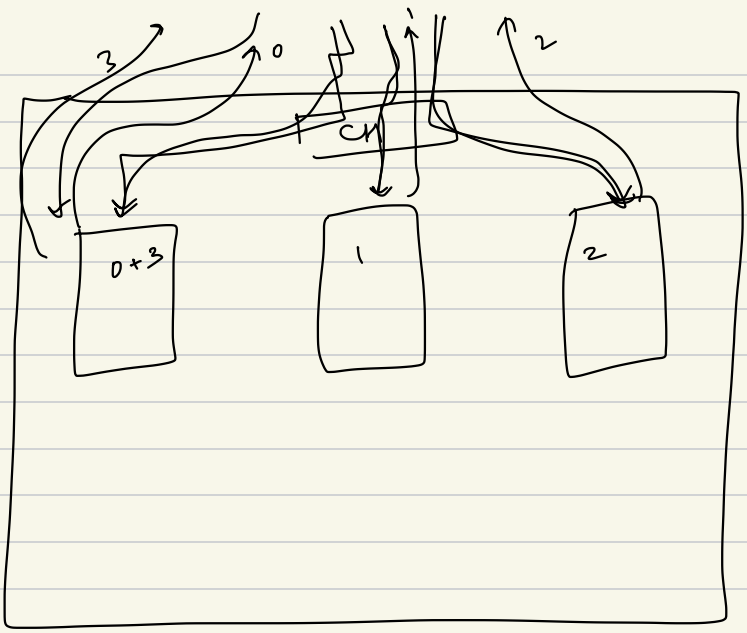


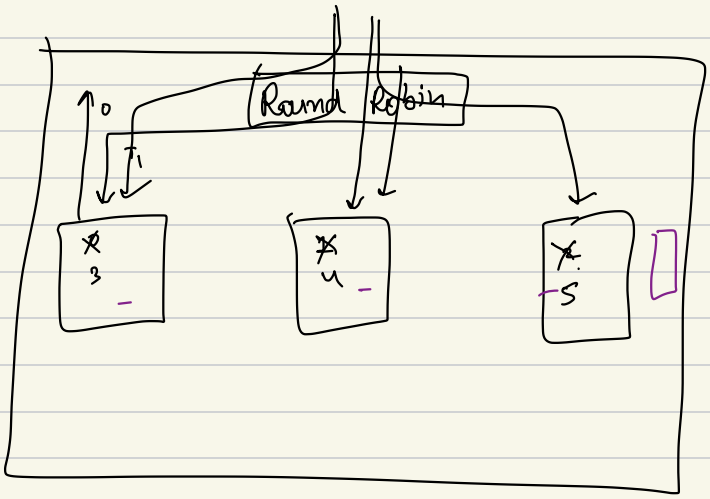
64 bits = 2 x 2 x 2 x 2 x ... (64 times) = 2^{64}

Options 1 : Auto incrementing ids like SQL



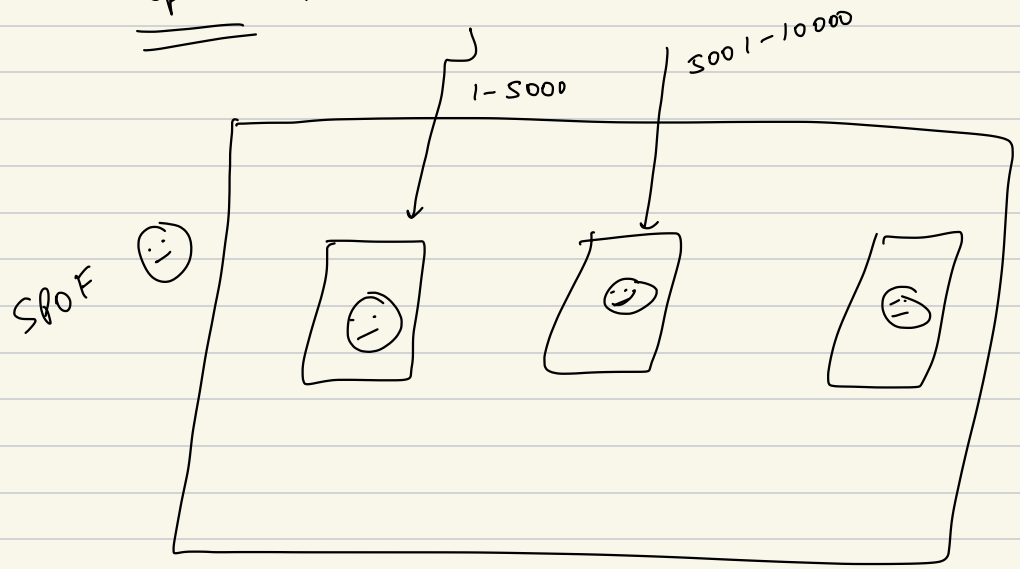




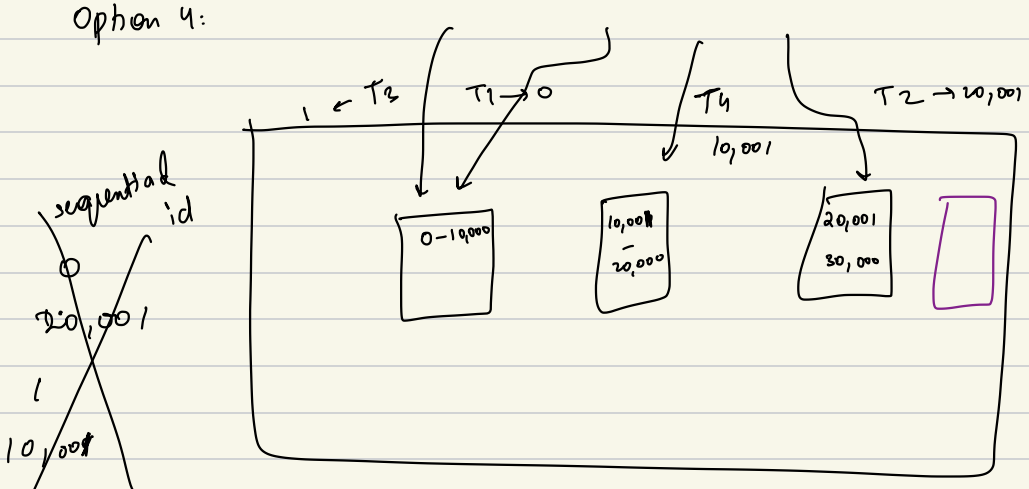


T1 0
T2 ~~1~~
T3 2
T4 3
T5 4
T6 5

Option 3:



Option 4:

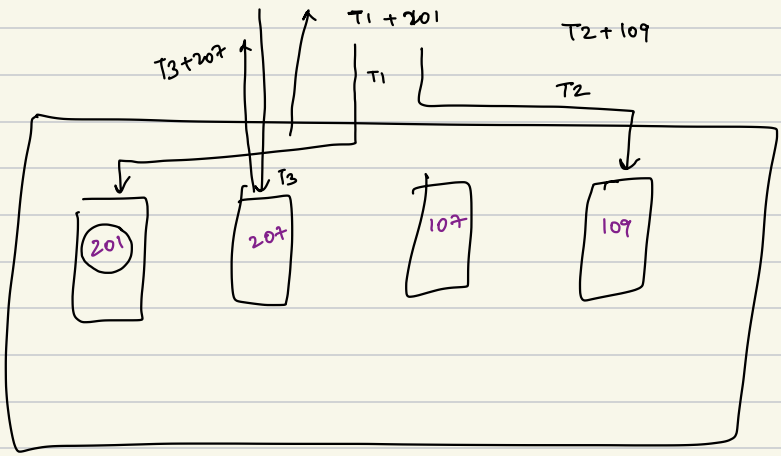


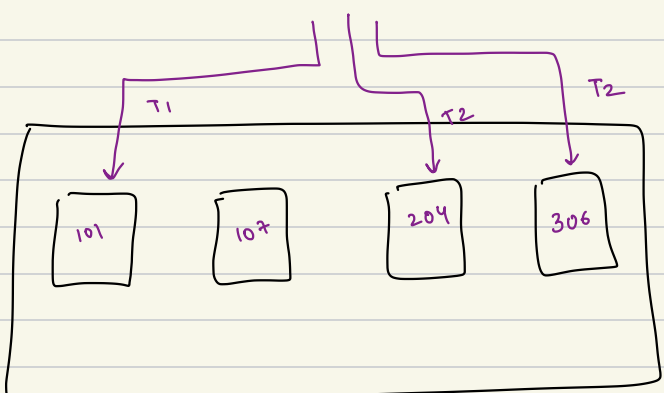
options:

UUID — 128 bits

random → They are NOT auto incremental

(option 6:) [Timestamp + Server Id]



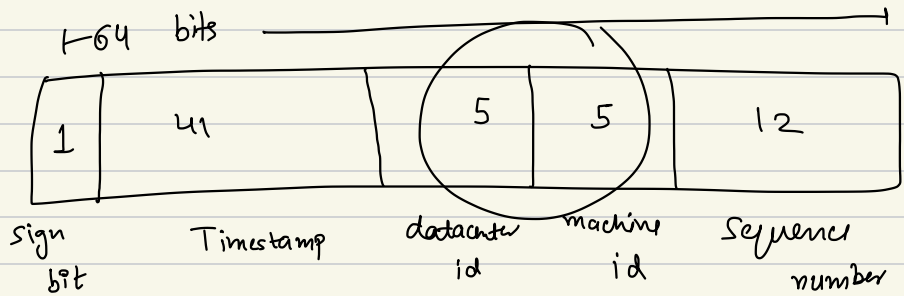


0301 20240905 4600 – 101 ⇒ id ☺

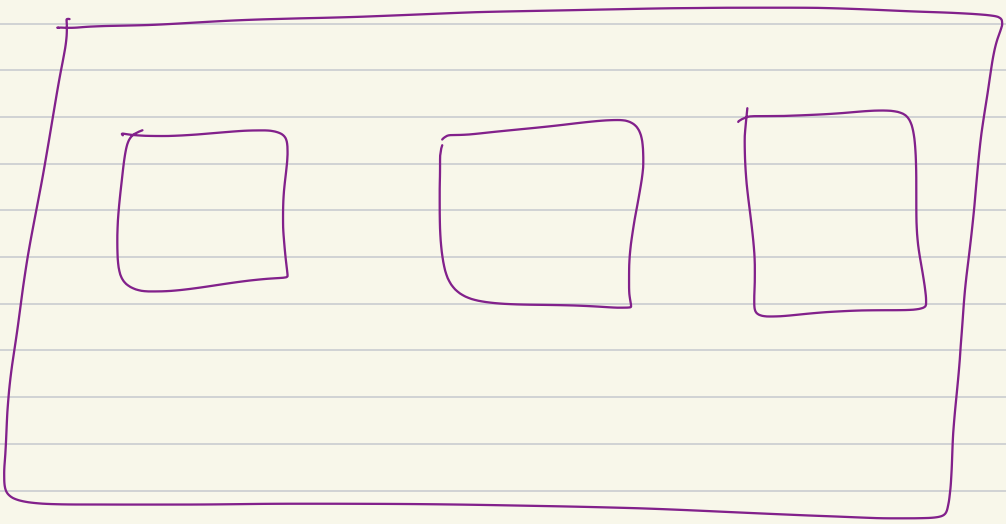
0301 20240956 00 – 306 ⇒ id ☺

0301 20240956 00 – 204 ⇒ id ☺

TWITTER SNOWFLAKES ALGORITHM



1 + 41 + 5 + 5 + 12 = 64 ☺



(1) sign bit (1 bit)

⇒ always 0

(2) Timestamp bits (41 bits)

epoch time

= no of seconds that have passed from

1 Jan 1970

2^{41} seconds

$$\frac{2^{41}}{60} \times \frac{1}{60} \times \frac{1}{24} \times \frac{1}{365}$$

If the sensitivity is in seconds

we can cover 69000 yrs

But if precision sensitivity is kept to milli seconds

2^{41} milliseconds

$$\frac{2^{41}}{1000 \times 60 \times 60 \times 24 \times 365} \simeq 69 \text{ years}$$

Epoch Time → 1 Jan 1970 ☺

+ 69

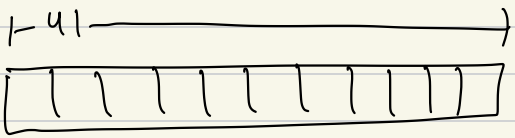
2039 ☹

4th Nov 2010 → base date

+69

2079

2^{41} different values



Timestamp

# of milliseconds that have passed from 4th November 2010

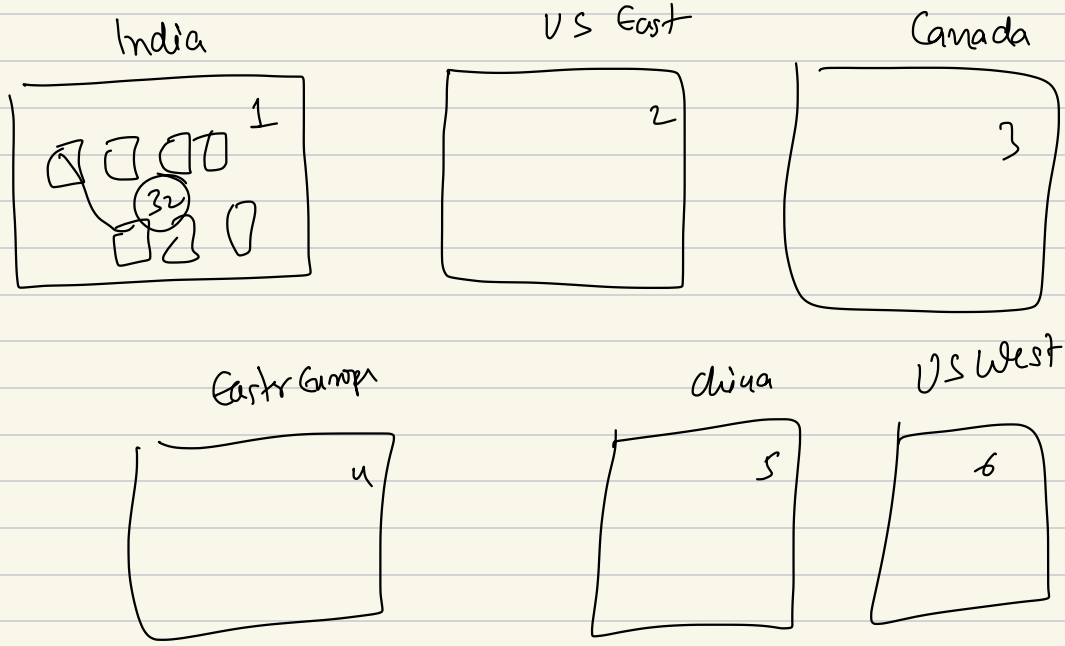
T1 (4/11/2010 00:00 00:000) ≡ 00000000000000000000000000000000000000

T2(4/11/2010 00:00 00:050) ≡ 00000000000000000000000000110010

110010

T3 (3/01/2024 22:25 00:000)

(3) Datacenter bits ((5) bits) $\Rightarrow 2^5 = 32$



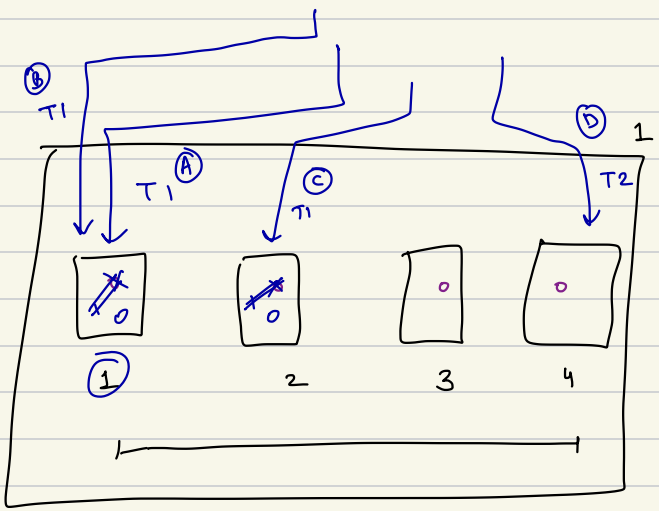
(4) Machine bits (5) $= 2^5 = 32$

(5) Sequence Number (12 bits) → counter

are reserved for generating auto incremented sequence numbers

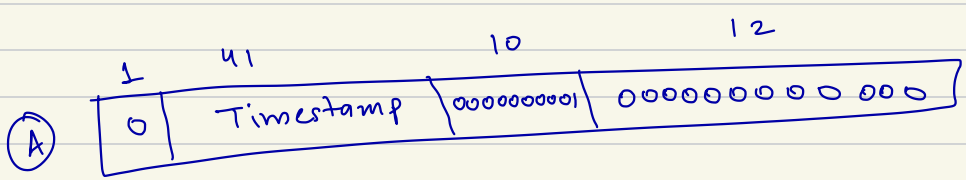
$$2^{12} = 4096$$

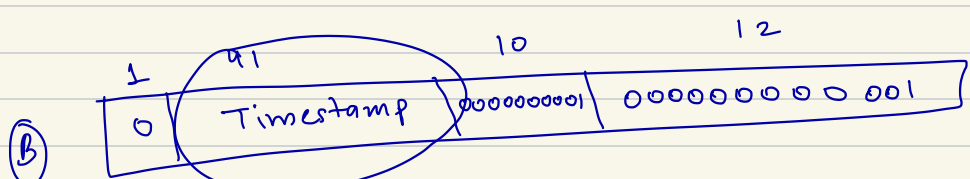
reset to 0 at every millisecond

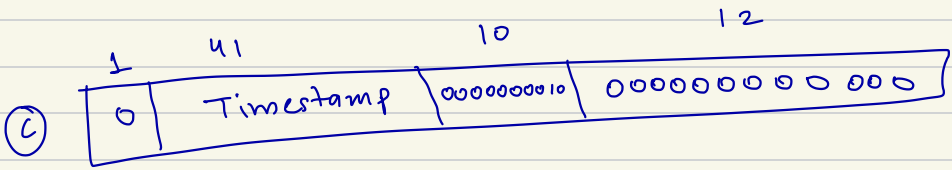


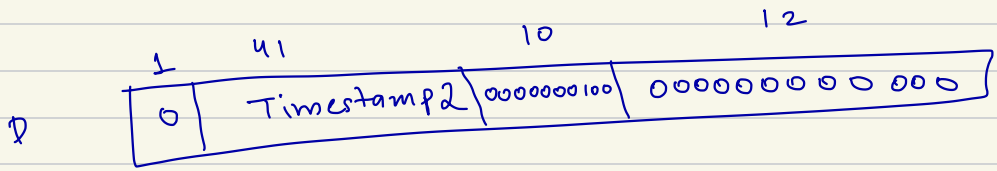
- ms

10 bits ≡ machine ids









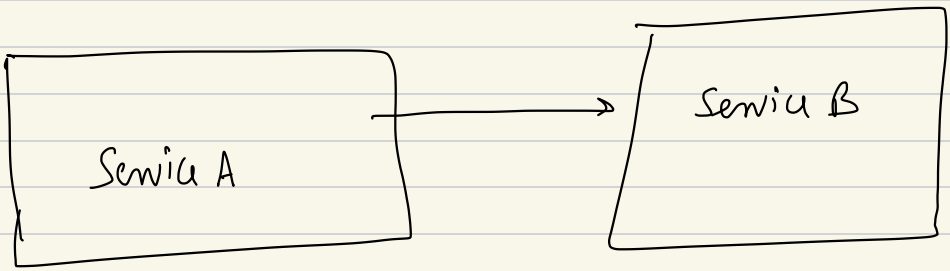
---

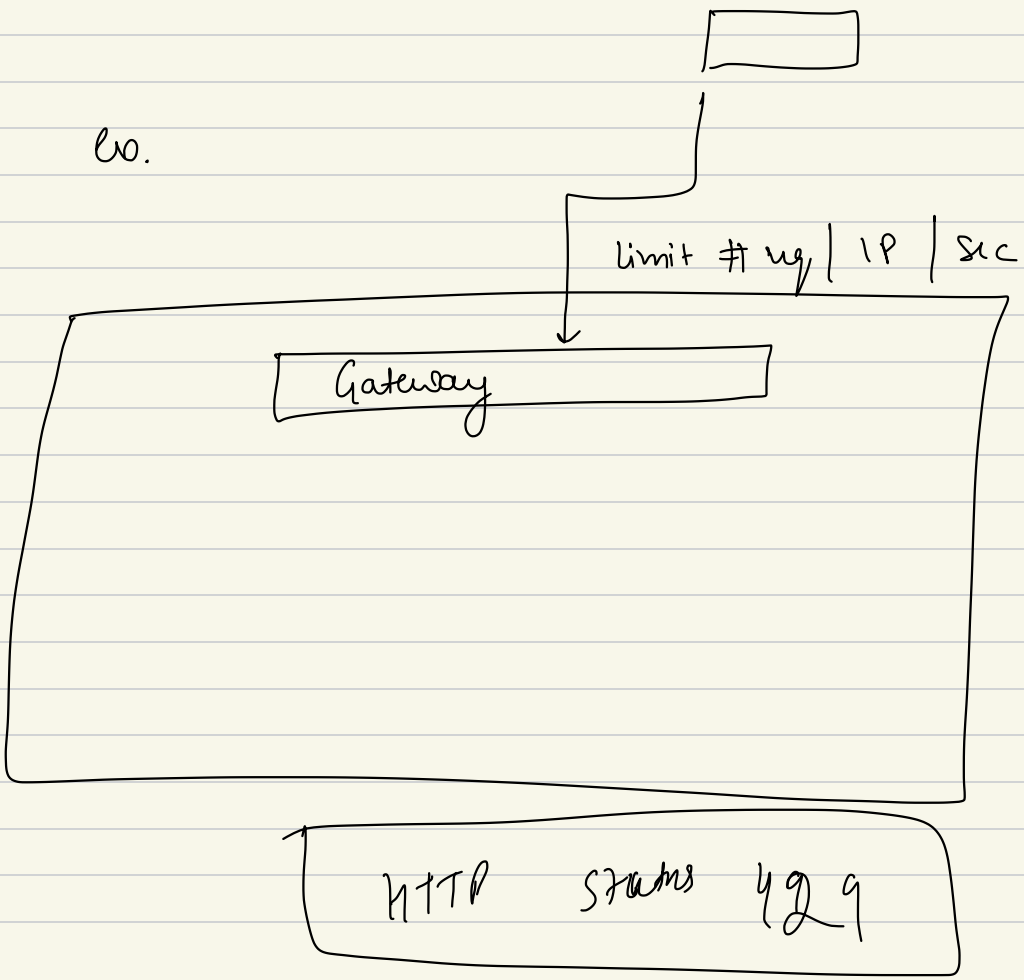
① always unique Ids

[a machine can atmost handle 4096 id generation reqs per millisecond]

Break → 10:49 – 10:55 PM ☺

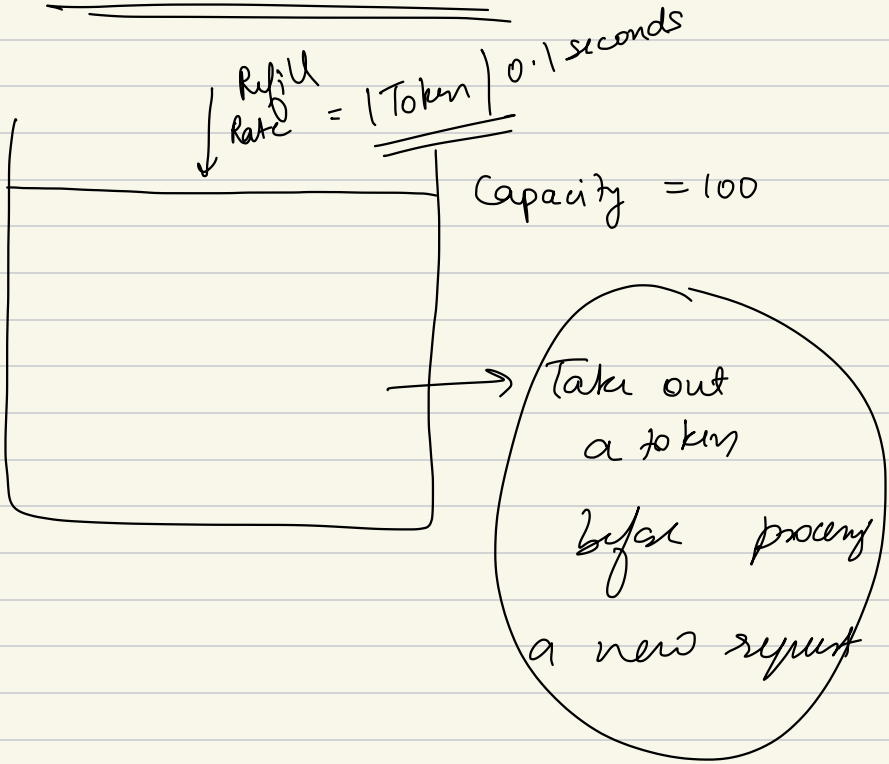
(2) Design a Rate Limiter





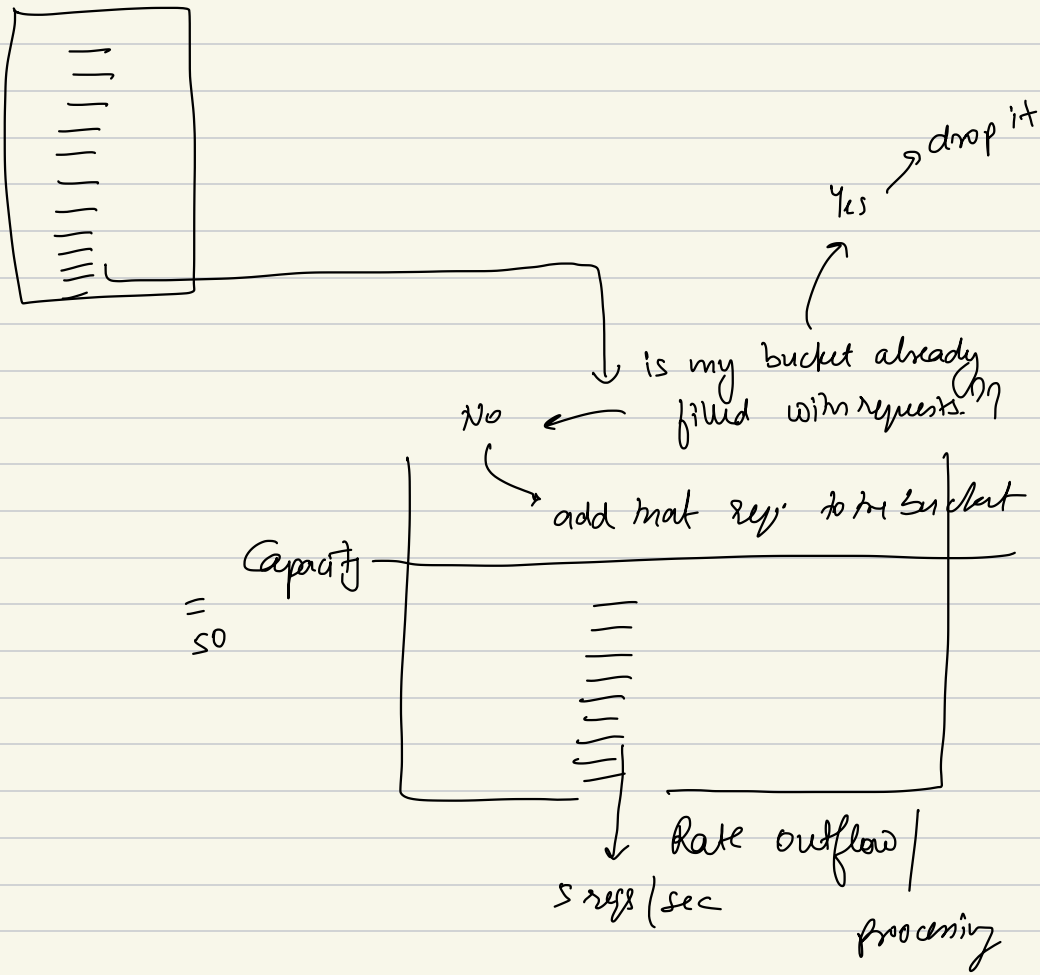
Algorithms for Rate Limiting

(1) Token Bucket Algorithm

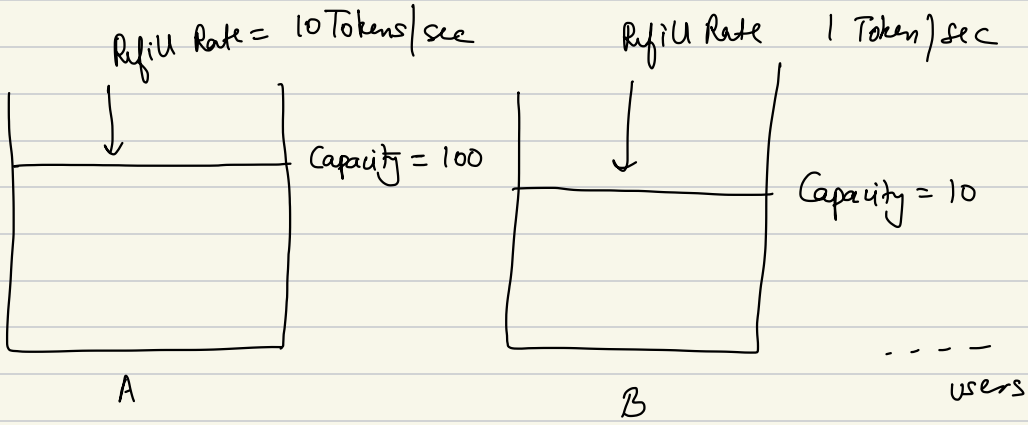


Smooth way of handling Rate Limiting

② Leaky Bucket Algorithm



	A	B
ChatGPT	10$ month	Free Tier



1

10

2

20

30

40

~~50~~

→ A uses 9 questions ☺

38

1

2

3

4

5

B asks 12 questions

5 questions dropped ☺

A	B
38	0
+ 10	+ 1
+ 10	+ 1

98

+ 10 => 100

→ 100
→
→

100

100

Burst

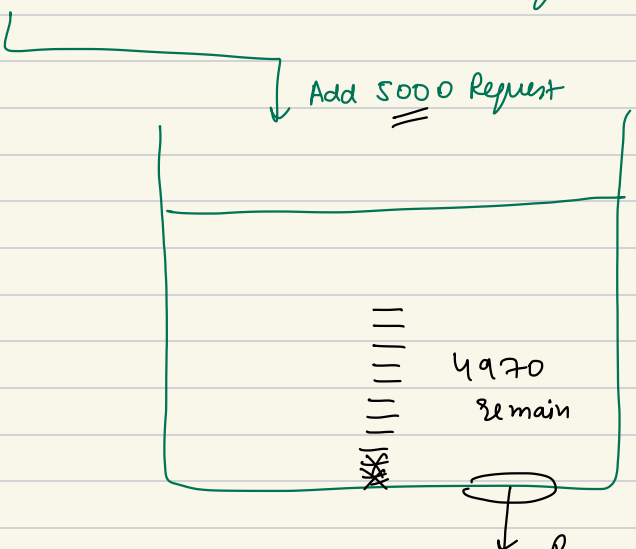
no concept of a requests waiting at the server.

② Leaky Bucket Algo

Email delivery system / Marketing

A = 100$ month

- Capacity = 10,000
- Request Processing Rate = 10 emails/second



5,000

1sec

2sec

3sec

10

10

Add 20,000 Email Request

Successfully add 5030/20,000

The remaining 14,970 are dropped.

10,000
4970
5030

20,000

add

reject

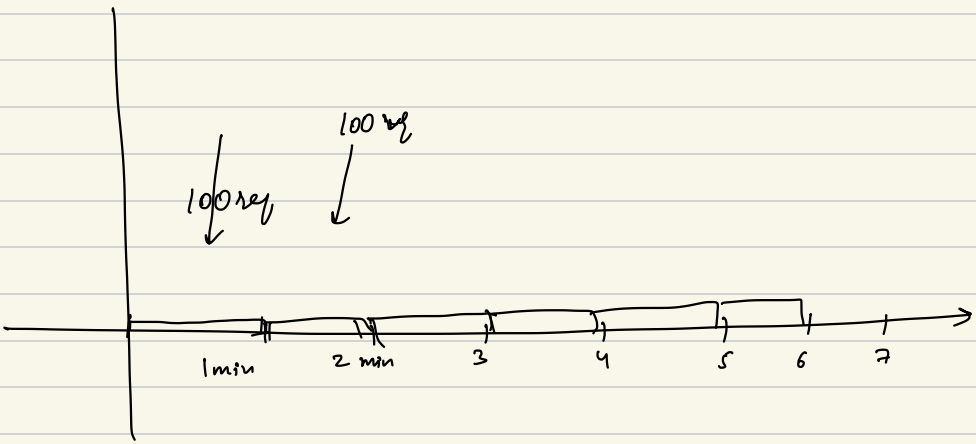
5030

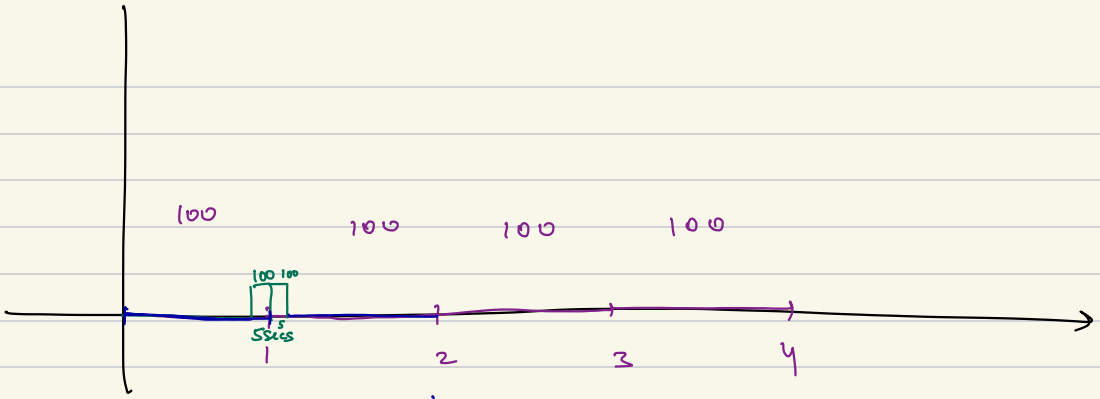
14970

(3) Fixed Window Counter

Most intuitive

gets defeated





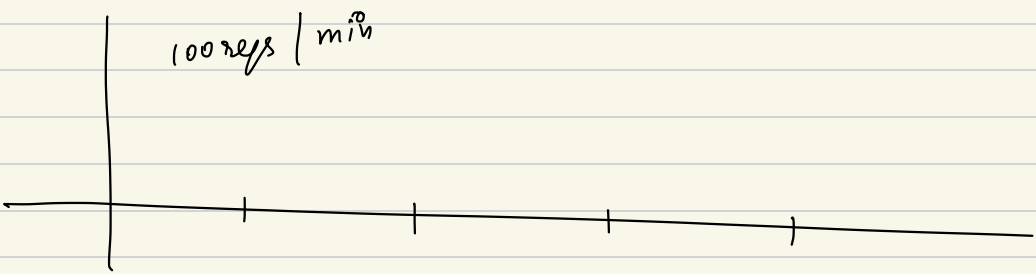
✓ anticipated load = 100 reqs / min

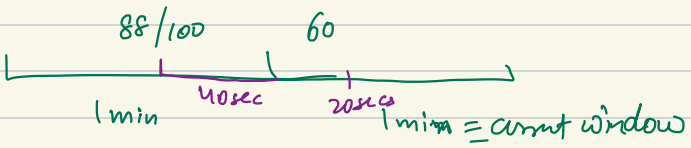
✓ actual load = 200 reqs / 10secs

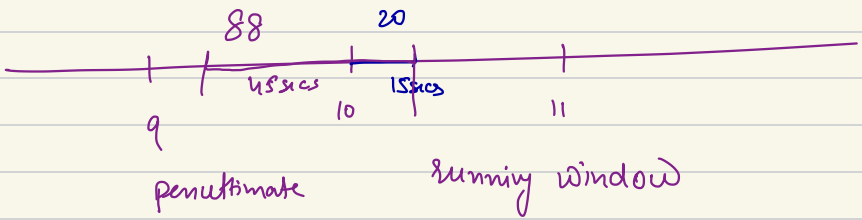
---

④ Sliding Window Counter → is a better version of Algo 3 = Fixed window

Approximate Algo







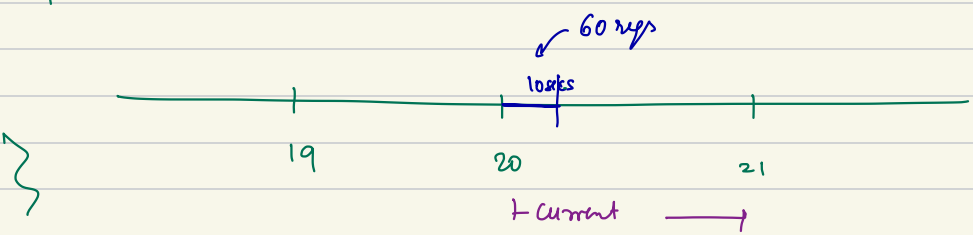
$$\frac{45}{60} \quad \frac{3}{4} \times \overset{22}{\cancel{88}}$$

66 34

---

deeper example

penultimate_window_count = 90



Approximate Calculation

$\frac{10}{60} = 60$

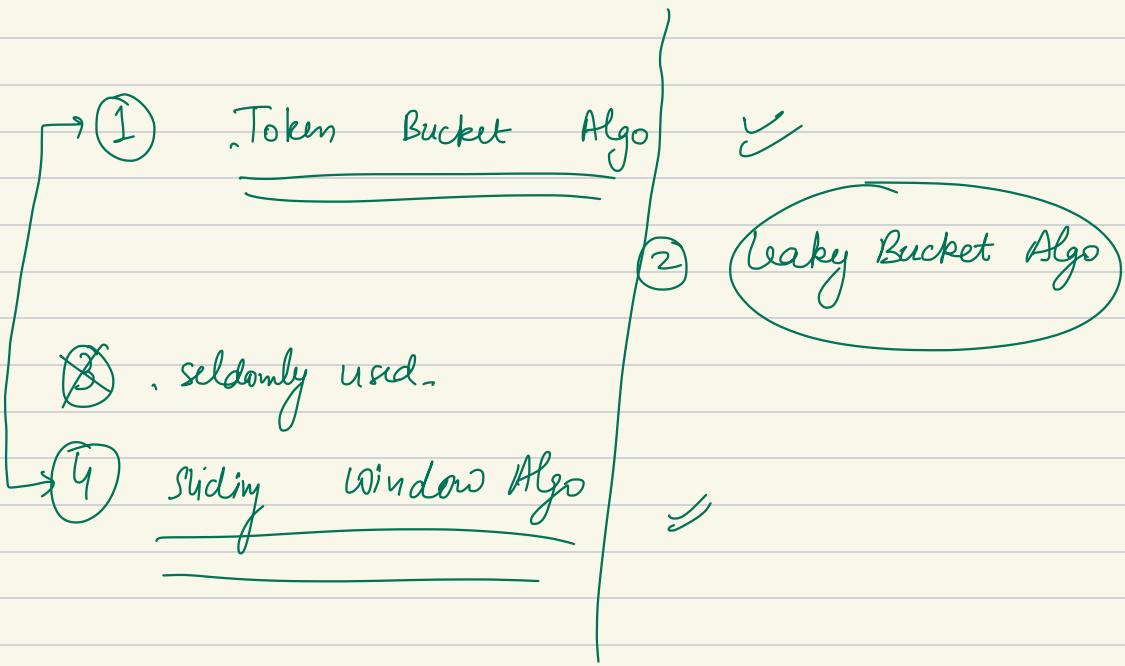
Prev window = $\frac{5}{60} \times \overset{15}{\cancel{90}}$ approx

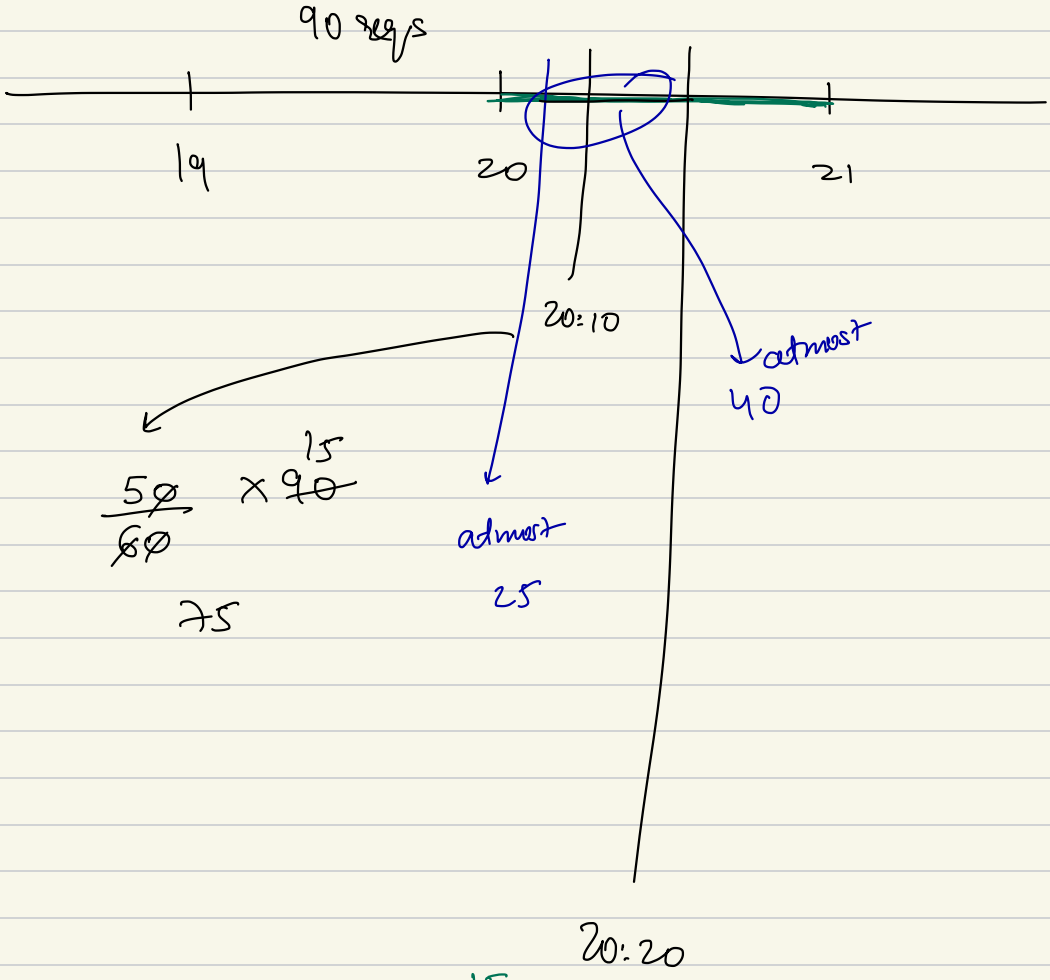
75 approx

[Atmost 75 reqs can be allowed in first 10 secs of curr window]

25/60 allowed

35/60 dropped





$\frac{50}{60} \times \overset{15}{\cancel{90}}$

75

almost

25

$\frac{40}{60} \times \overset{15}{\cancel{90}}$

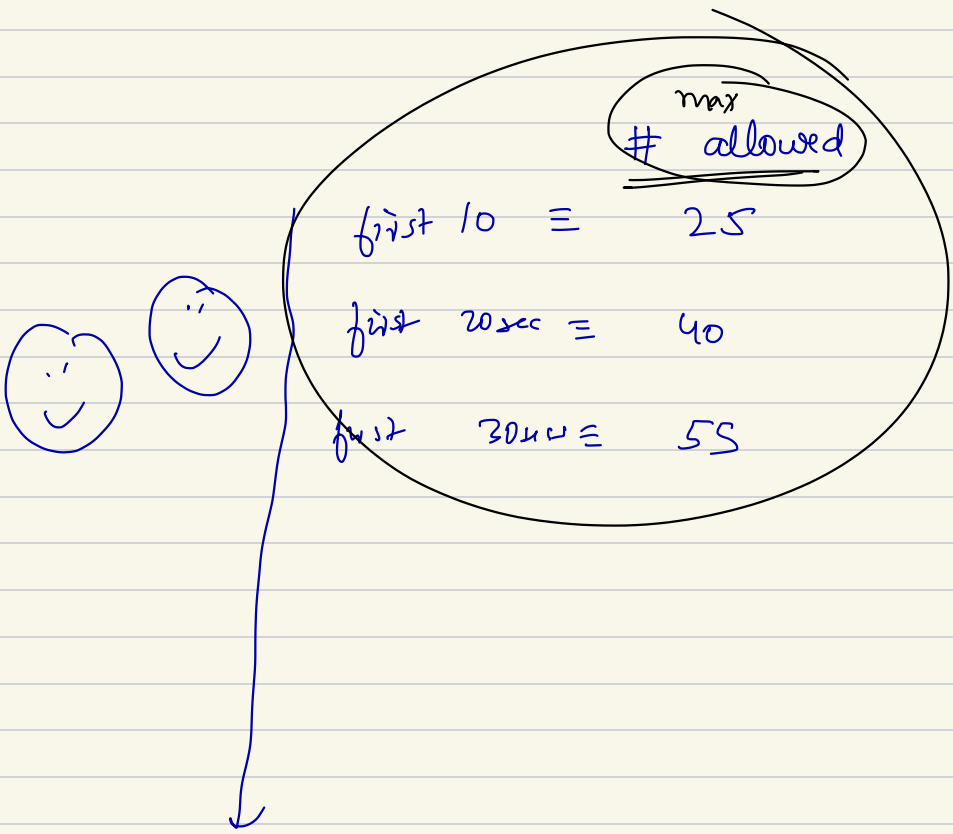
60

20:30

$\frac{30}{60} \times \frac{15}{90}$

45

55 req / first 30 secs



Questions

Machine # 4