# RATE LIMITING

# OGRANIČENJE STOPE PRISTUPA



## ŠTA JE RATE LIMITING?

- U računarskim mrežama ograničenje brzine/stope se koristi za kontrolu brzine/stope zahteva poslatih ili primljenih od strane kontrolera mrežnog interfejsa
- Ograničavanje stope/brzine pristupa (*Rate Limiting*) je procedura koja omogućava kontrolu brzine kojom korisnici mogu da šalju zahteve sistemu
- *Rate Limiting* se uglavnom koristi za zaštitu servera od neželjenih ili zlonamernih zahteva

## PRIMERI

- Ograničavanje da korisnik ne može da postavi više od 2 komentara u sekundi
- Omogućavanje korisnicima da se prijave za nagrade za igru ne više od 2 puta dnevno
- Omogućavanje više pretplata na servere sa većim resursima od servera sa manje resursa

# OGRANIČENJE STOPE PRISTUPA



## PREDNOSTI

- Smanjenje troškova klijenata ako plaćaju cenu po pozivu servisa koji koristi u pozadini usluge *3rd party* servisa kao što je plaćanje

- Sprečavanje *Denial of Service* (DOS) napada u slučaju da zlonamerni korisnik šalje ogromnu količinu zahteva za zaguši server

- Sprečavanje preopterećenja servera sa velikim brojem zahteva koji dolaze u određenim periodima dana ili godine

# OGRANIČENJE STOPE PRISTUPA



## TIPOVI

- Rate limiter korisnika omogućava nekim grupama korisnika ograničen pristup sistemu – broj/trajanje zahteva korisnika obično je vezan za njihove ključeve ili IP adrese
- Rate limiter servisa na serverima prati koliko je paralelnih sesija ili veza dozvoljeno za neke grupe korisnika
- Rate limiter lokacije ograničava brzine/stope pristupa za neke regiione, kao i za definisani vremenski pristup – moguće je definisati različite stepene pristupa za različite lokacije

# OGRANIČENJE STOPE PRISTUPA



## RATE LIMITER U SISTEMIMA ZA SKLADIŠTENJE PODATAKA

- Neki sistemi za skladištenje velikih količina podataka su implementirali *Rate Limiting*
- To je nekada ključna razlika u odnosu na druge slične sistema i zato su čest izbor za korišćenje
- RocksDB direktno podržava ovaj mehanizam
- Kod upotrebe ovih sistema korisnici možda žele da priguše maksimalnu brzinu pisanja u okviru nekog ograničenja iz različitih razloga
- Na primer, brzi zapisi izazivaju velike skokove u kašnjenju čitanja ako prekorače definisani prag
- RocksDB ima mogućnost da korisnici podese *Rate limiter* kako njima odgovara
- Pruža mogućnost dinamičkog ograničenja – Auto-tuned Rate Limiter[1]

[1] RocksDB Docs http://rocksdb.org/blog/2017/12/18/17-auto-tuned-rate-limiter.html

# OGRANIČENJE STOPE PRISTUPA



## ALGORITMI

- Token Bucket
- Leaky Bucket
- Fixed Window Counter
- Sliding Window Log
- Sliding Window Counter

# TOKEN BUCKET



## NAJČEŠĆA OPCIJA

- Ovo je najjednostavniji algoritam za ograničavanje brzine pristupa
- Jednostavno se prati broj zahteva napravljenih u zadatom vremenskom intervalu
- Zbog svoje jednostavnosti dosta se koristi
- Google Cloud koristi ovaj algoritam (ili je koristio) za Cloud Tasks queues[1] opciju koja se nudi korisnicima kao usluga, Amazon npr. za EC2[2] instance, Stripe[3] itd.
- Jednostavno se implementira i lako može da se poveže sa velikim brojem različitih slučajeva korišćenja

[1] Cloud Tasks https://cloud.google.com/tasks

[2]Amazon EC2 API https://docs.aws.amazon.com/AWSEC2/latest/APIReference/throttling.html

[3]Stripe Rate Limits https://stripe.com/docs/rate-limits

# TOKEN BUCKET



## IDEJA

- U algoritmu se koristi korpa (bucket) sa fiksnim kapacitetom u jedinicama broja tokena koje može da drži
- Kada zahtev stigne na *Throttling* servis, troši se token iz korpe ako korpa ima najmanje jedan token
- Ako u korpi nema tokena, zahtev se odbacuje ukazujući korisnicima da su njihovi zahtevi premašili ukupan dozvoljeni broj
- Korpa tokena se puni fiksnim brojem tokena po jedinici vremena i dodatni tokeni koji treba da se dodaju u kantu koja je puna se odbacuju

# TOKEN BUCKET

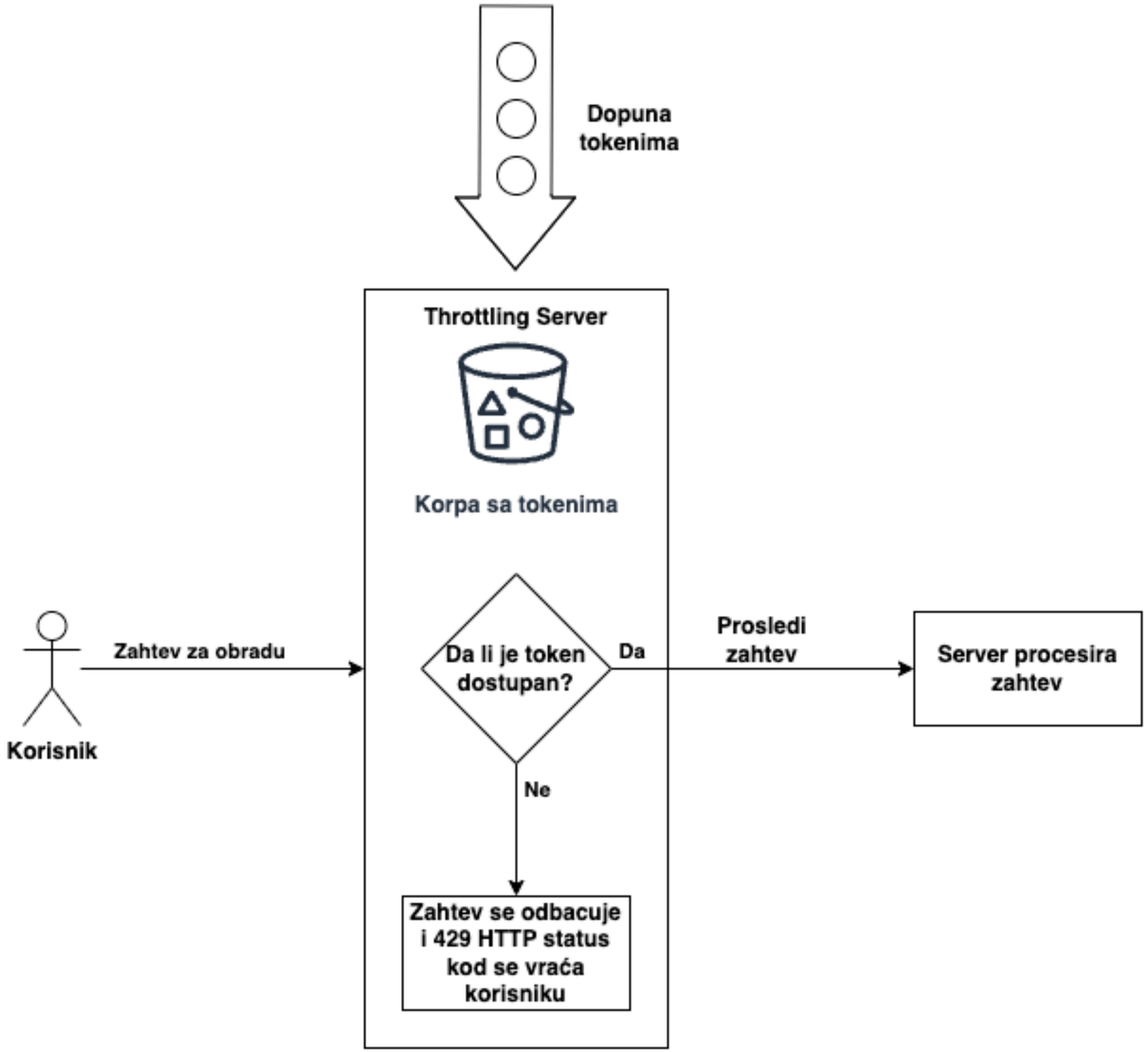


## IDEJA

- Potrebno je da se konfigurišu tri stvari:
    1. Veličina korpe (broj dozvoljenih tokena)
    2. Tempo kojim se dopunjuje korpa
    3. Količina kojom se dopunjuje korpa

# TOKEN BUCKET





# TOKEN BUCKET



## ALGORITAM

- Za svaki zahtev korisnika treba:
  - Proveriti da li je vreme isteklo od poslednjeg resetovanja brojača vremena
  - Ako vreme nije isteklo, treba proveriti da li korisnik ima dovoljno preostalih tokena da obradi dolazni zahtev
  - Ako korisniku nije preostalo slobodnih tokena, trenutni zahtev se odbacuje uz adekvatnu poruku
  - U suprotnom se smanjuje brojač za 1 i vrši obrada dolaznog zahteva
  - Ako je vreme proteklo, tj. razlika resetovanog vremena i trenutnog vremena je veća od definisanog intervala, resetuje se broj dozvoljenih zahteva na unapred definisano ograničenje i definiše se novo vreme resetovanja

# TOKEN BUCKET



## PRIMER

- 3/min:
  - REQ 11:01:20 → BUCKET [11:01:05, 3] → OK
  - REQ 11:01:25 → BUCKET [11:01:05, 2] → OK
  - REQ 11:01:30 → BUCKET [11:01:05, 1] → OK
  - REQ 11:01:35 → BUCKET [11:01:05, 0] → FAIL
  - REQ 11:03:00 → BUCKET [11:03:00, 3] → OK // radi se ažuriranje vremena, broja tokena, pušta se zahtev i smanjuje broj tokena za 1
  - ...

# TOKEN BUCKET



## PREDNOSTI

- Efikasno po pitanju utroška memorije jer postoji korpa fiksne veličine
- Velika količina saobraćaja se može podržati u kratkom vremenskom intervalu dok god ima slobodnih tokena u korpi

## MANE

- Zahtevno za adekvatno podešavanje parametara da se dobiju najbolje performanse

# LEAKY BUCKET



## IDEJA

- Zahtevi se obrađuju po fiksnoj stopi uz pomoć FIFO reda
- Kada zahtev stigne na *Throttling* servis proverava se da li je red popunjen
- Ako je red pun, zahtevi se odbacuju i 429 HTTP statusni kod se vraća klijentu koji ukazuje da je korisnik premašio dozvoljeni broj zahteva
- Zahtevi koji nisu obrađeni ne moraju se odbaciti već se mogu staviti u red za ponovne pokušaje
- Zahtevi se čitaju iz reda u fiksnom intervalu ("*leakuju*") kako bi se osiguralo da serveri nisu preopterećeni naletom zahteva i da bi se smanjila mogućnost za iscrpljivanje resursa servera
- NGINX[1] koristi ovaj algoritam

[1] NGINX Rate Limiting https://www.nginx.com/blog/rate-limiting-nginx/

# LEAKY BUCKET

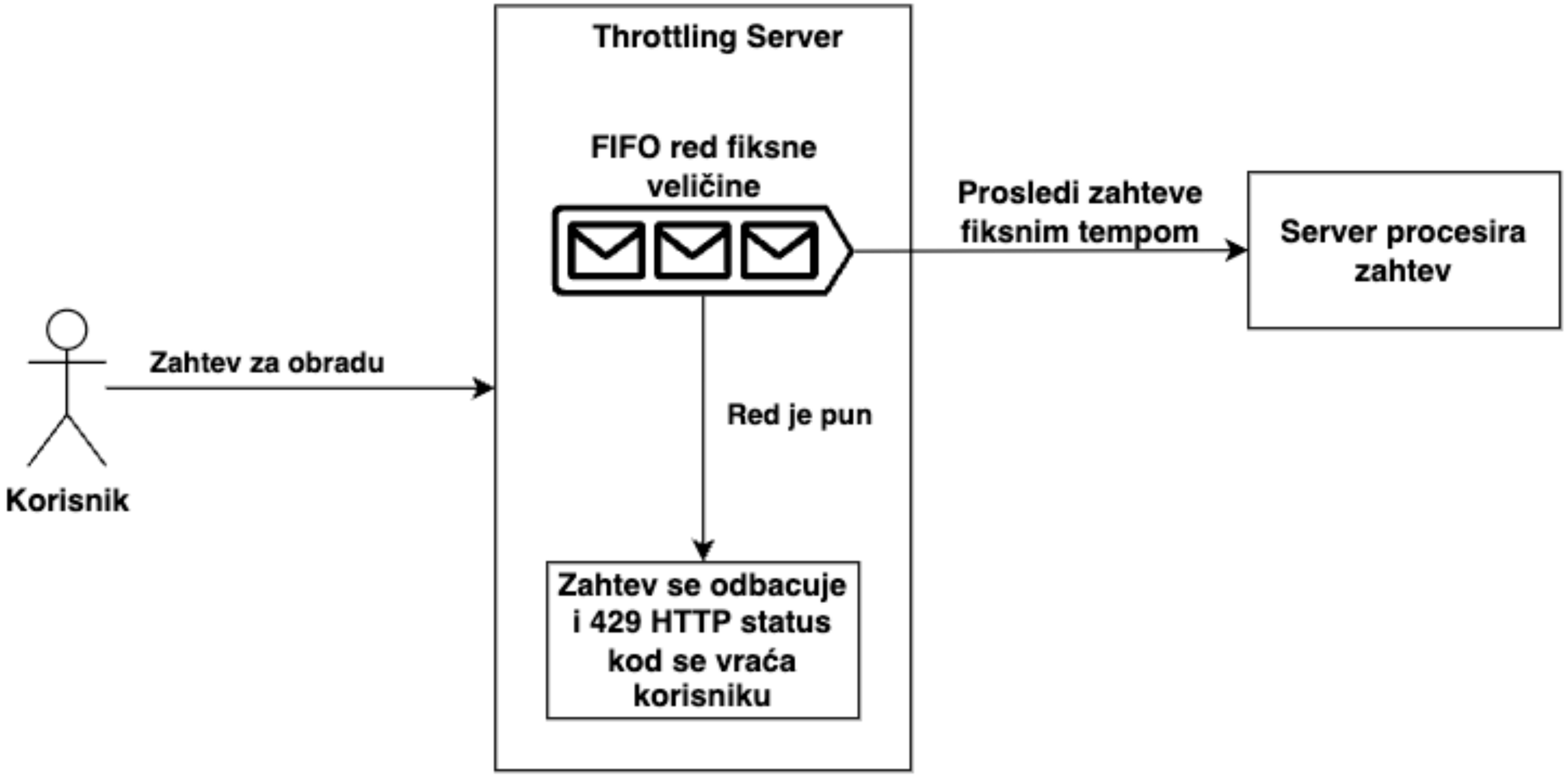


## IDEJA

- Potrebno je da se konfigurišu dve stvari:
    1. Veličina reda (broj dozvoljenih tokena)
    2. Tempo kojim se uzimaju zahtevi iz reda

# LEAKY BUCKET





# LEAKY BUCKET



## PREDNOSTI

- Efikasno po pitanju utroška memorije jer postoji red fiksne veličine
- Fiksan tempo kojim se uzimaju zahtevi iz reda kojim se reguliše koliko zahteva može da se obrađuje

## MANE

- Zahtevno za adekvatno podešavanje parametara da se dobiju najbolje performanse
- Bazira se na statičkom redu te postoji šansa da noviji zahtevi koji pristignu u red nikad ne budu obrađeni (*request starvation*)

# FIXED WINDOW COUNTER

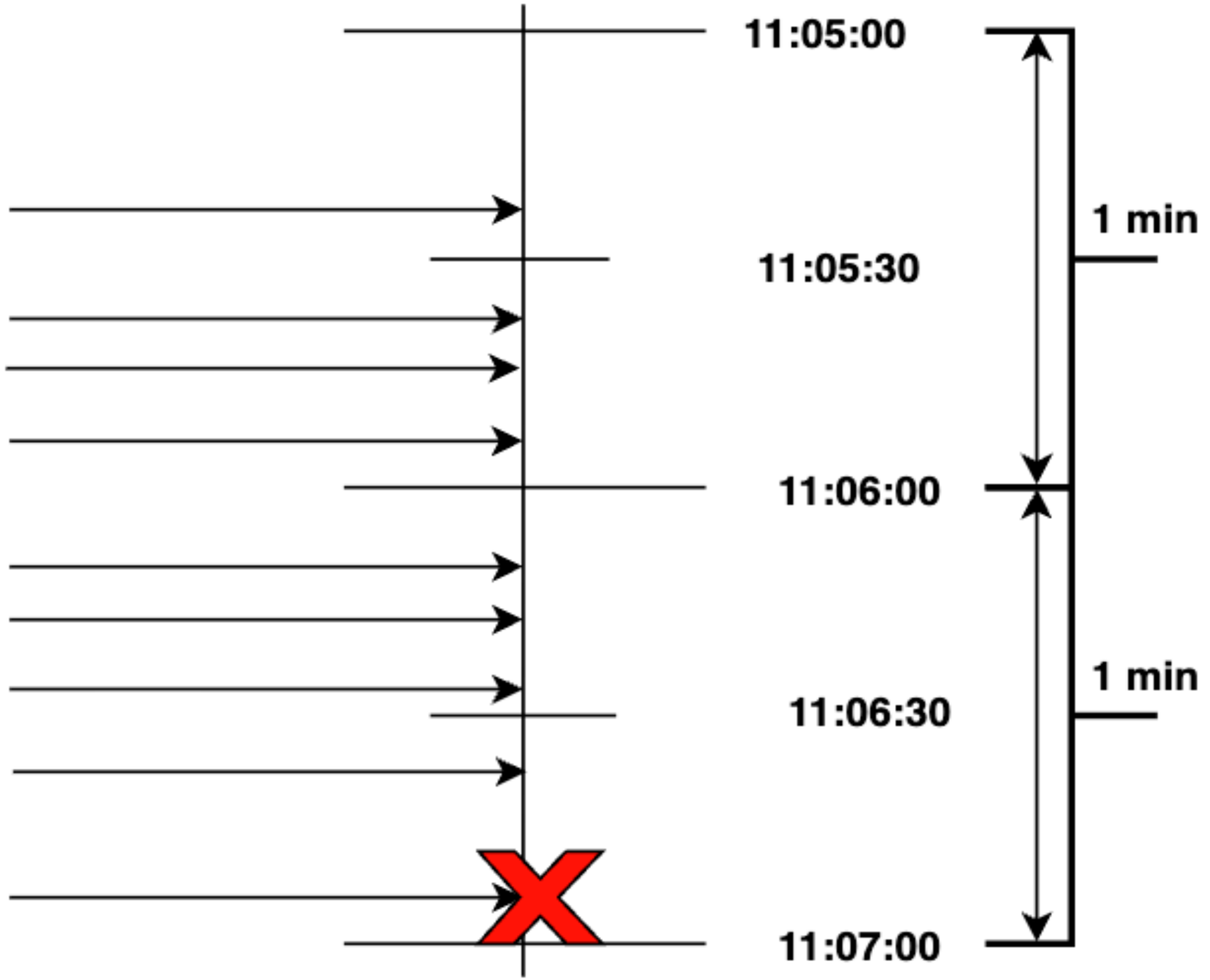


## IDEJA

- Algoritam dozvoljava specificirani broj API poziva od korisnika u određenom periodu
- Kada zahtev stigne povećava se vrednost brojača za 1
- Ako brojač dostigne vrednost praga za određeni period naredni zahtevi se odbacuju
- Brojač će resetuje nakon isteklog intervala

# FIXED WINDOW COUNTER





# FIXED WINDOW COUNTER



## PREDNOSTI

- Efikasno po pitanju utroška memorije
- Lak za razumevanje
- Jednostavna implementacija
- Kako se broj zahteva obnavlja na početku svakog vremenskog prozora, algoritam ne uzrokuje izgladnjivanje novijih zahteva (*request starvation*)

## MANE

- Može da dođe do naglog povećanja saobraćaja pogotovo na početku intervala
- Ako brojač dostigne limit već na početku prozora, svi klijenti će morati da sačekaju dugi prozor za resetovanje brojača
- To se dešava jer ne postoji ograničenje na minimalni period koji treba da prođe između dva zahteva

# SLIDING WINDOW LOG

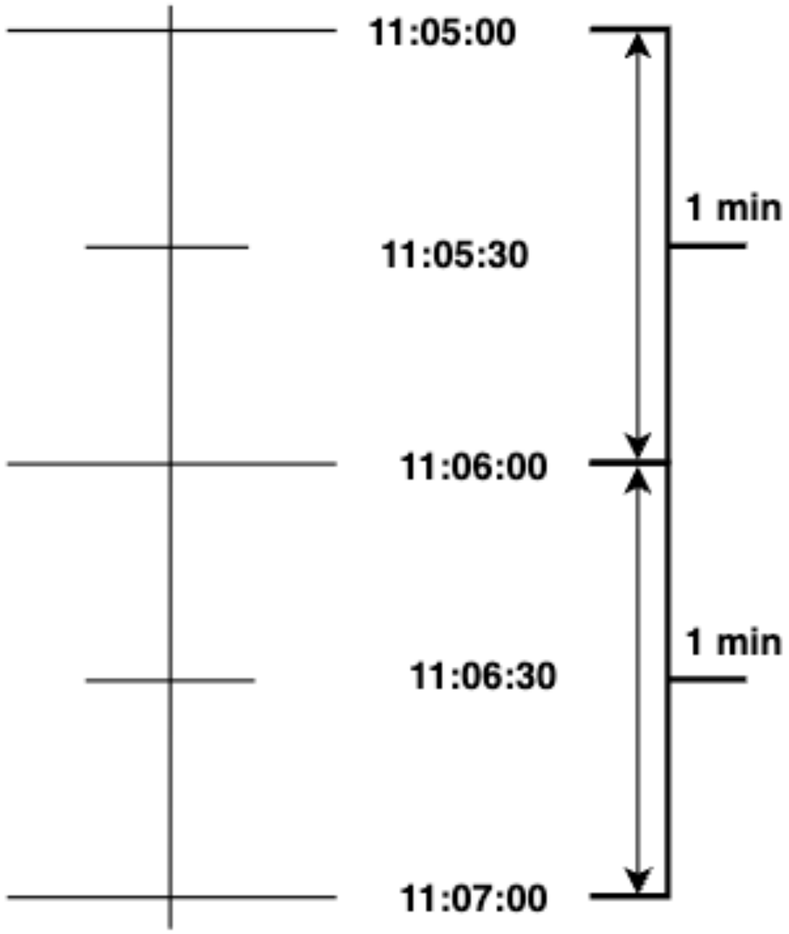


## IDEJA

- Sličan algoritmu *Fixed Window Counter* s tim da rešava glavni problem tog algoritma tj. omogućava da se više zahteva obradi u krajnjim granicama intervala
- Algoritam vodi evidenciju o svakom pristiglom zahtevu (čuva *timestamp* prihvaćenih i odbijenih zahteva)
- Kada stigne novi zahtev, uklanjaju se svi *timestamp*-ovi koji su stariji od trenutnog vremenskog intervala počevši od *timestamp*-a trenutnog zahteva
- Dodaje se novi *timestamp* zahteva u log i proverava da li je ukupan broj *timestamp*-ova premašio dozvoljeni prag
- Ako jeste, zahtev se odbacuje, ako nije, zahtev se prihvata za obradu

# SLIDING WINDOW LOG

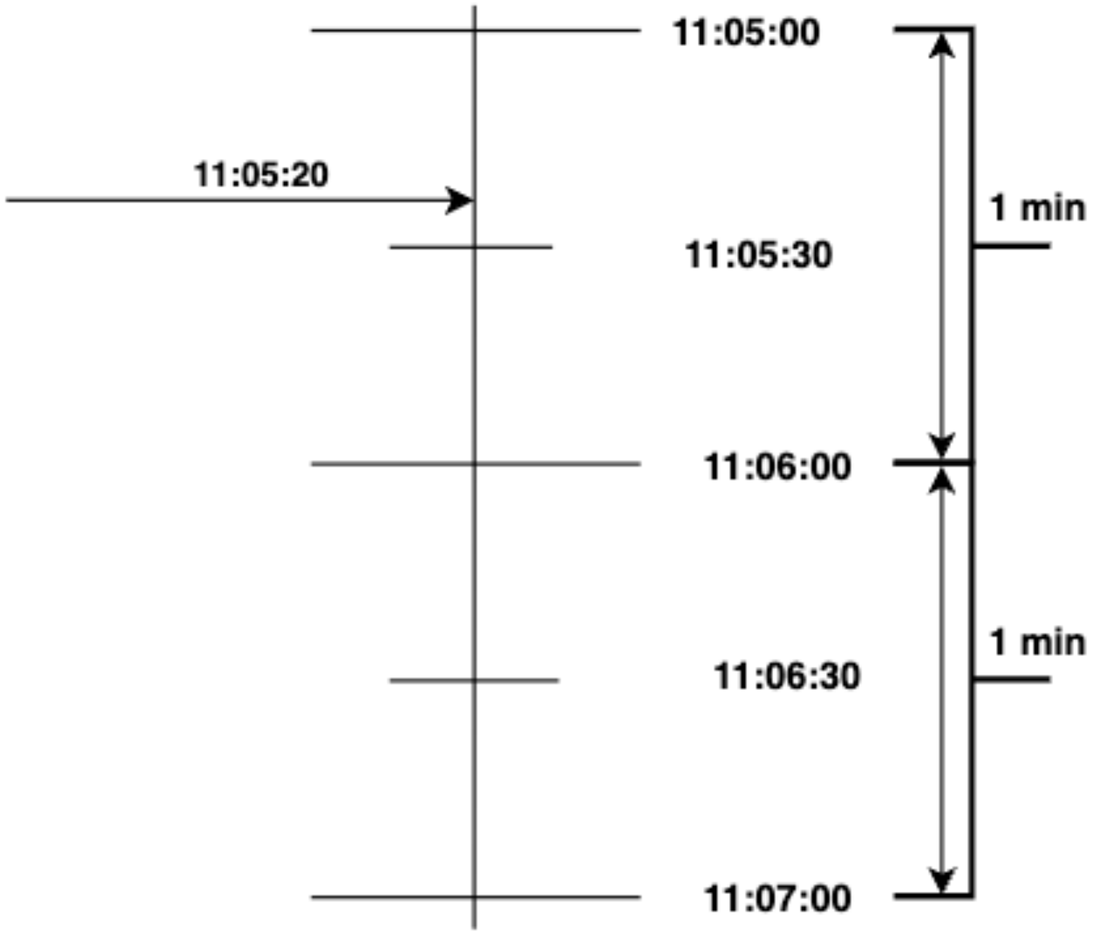





Timestamp
Log

# SLIDING WINDOW LOG



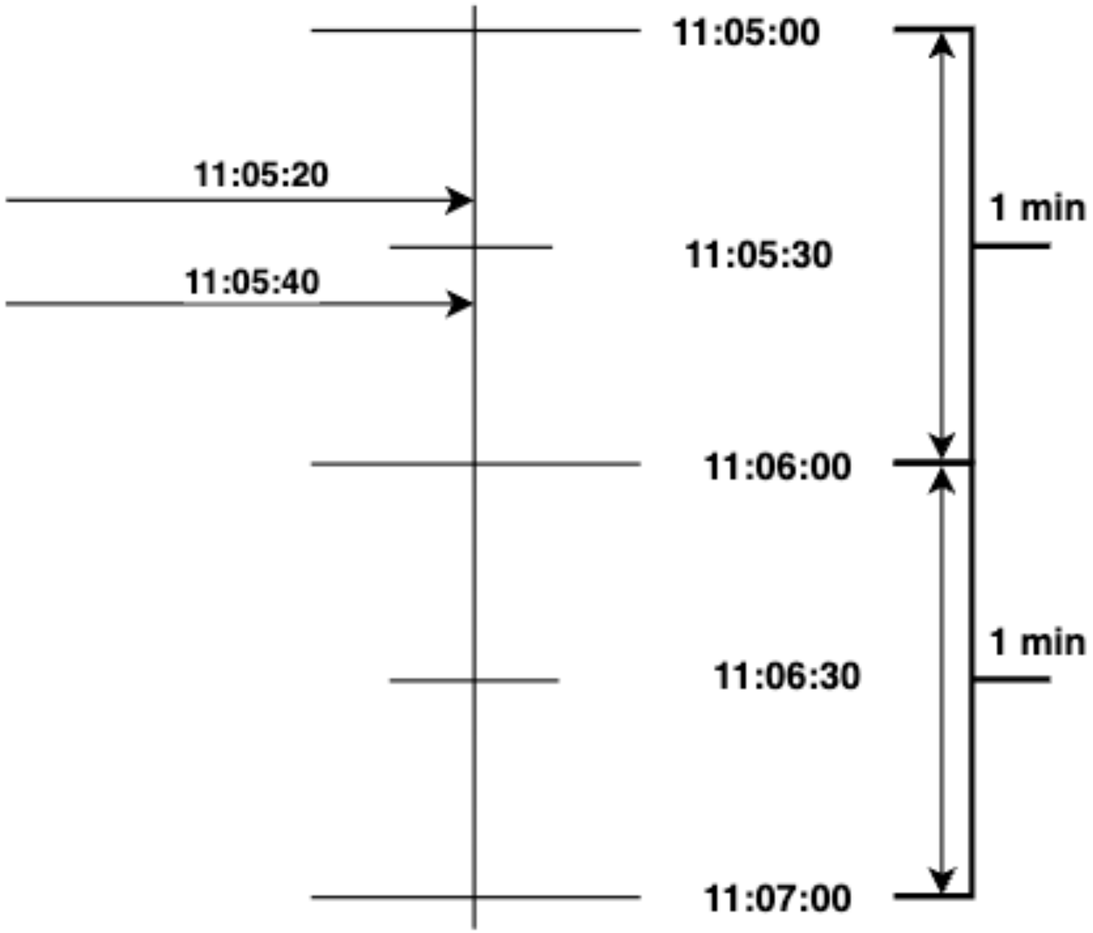



Timestamp
Log

11:05:20

# SLIDING WINDOW LOG



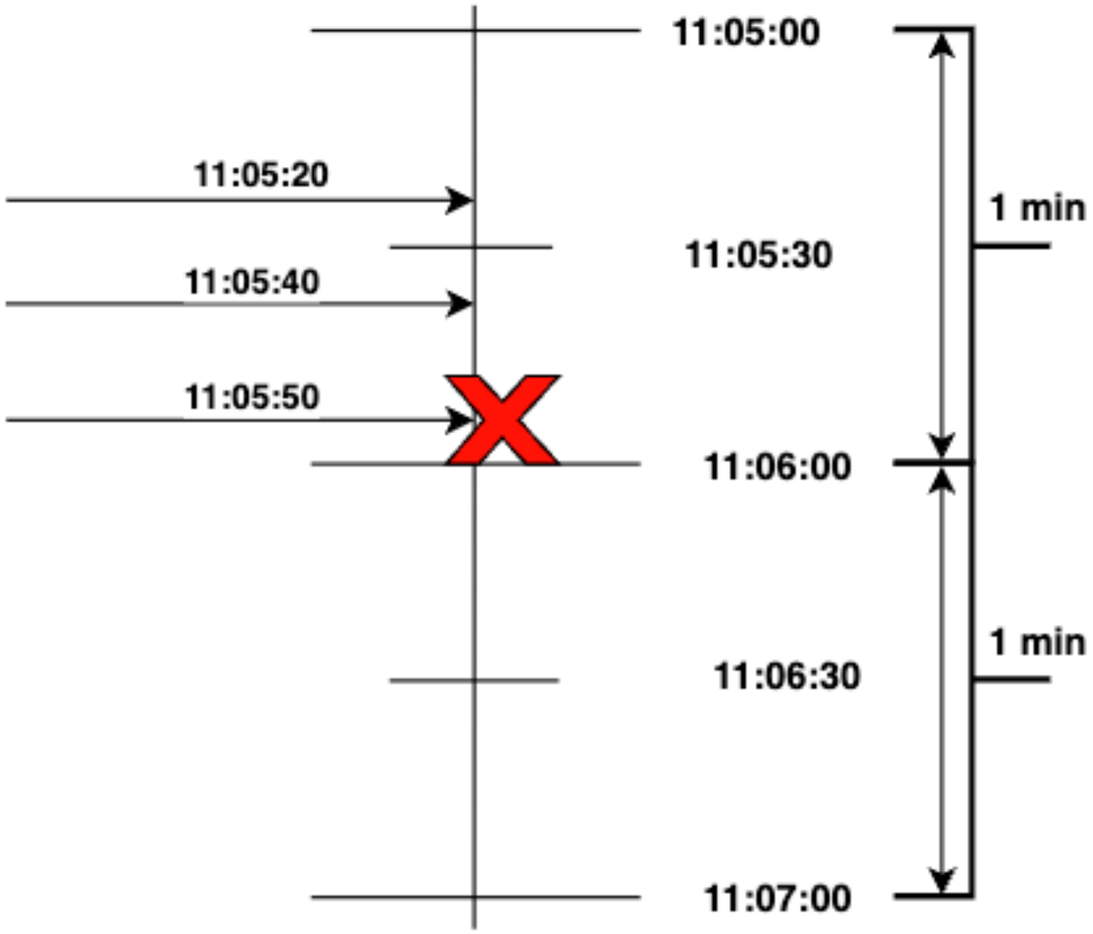


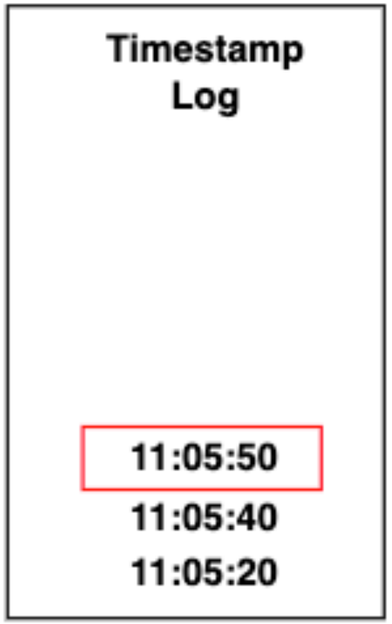

Timestamp
Log

11:05:40
11:05:20

# SLIDING WINDOW LOG

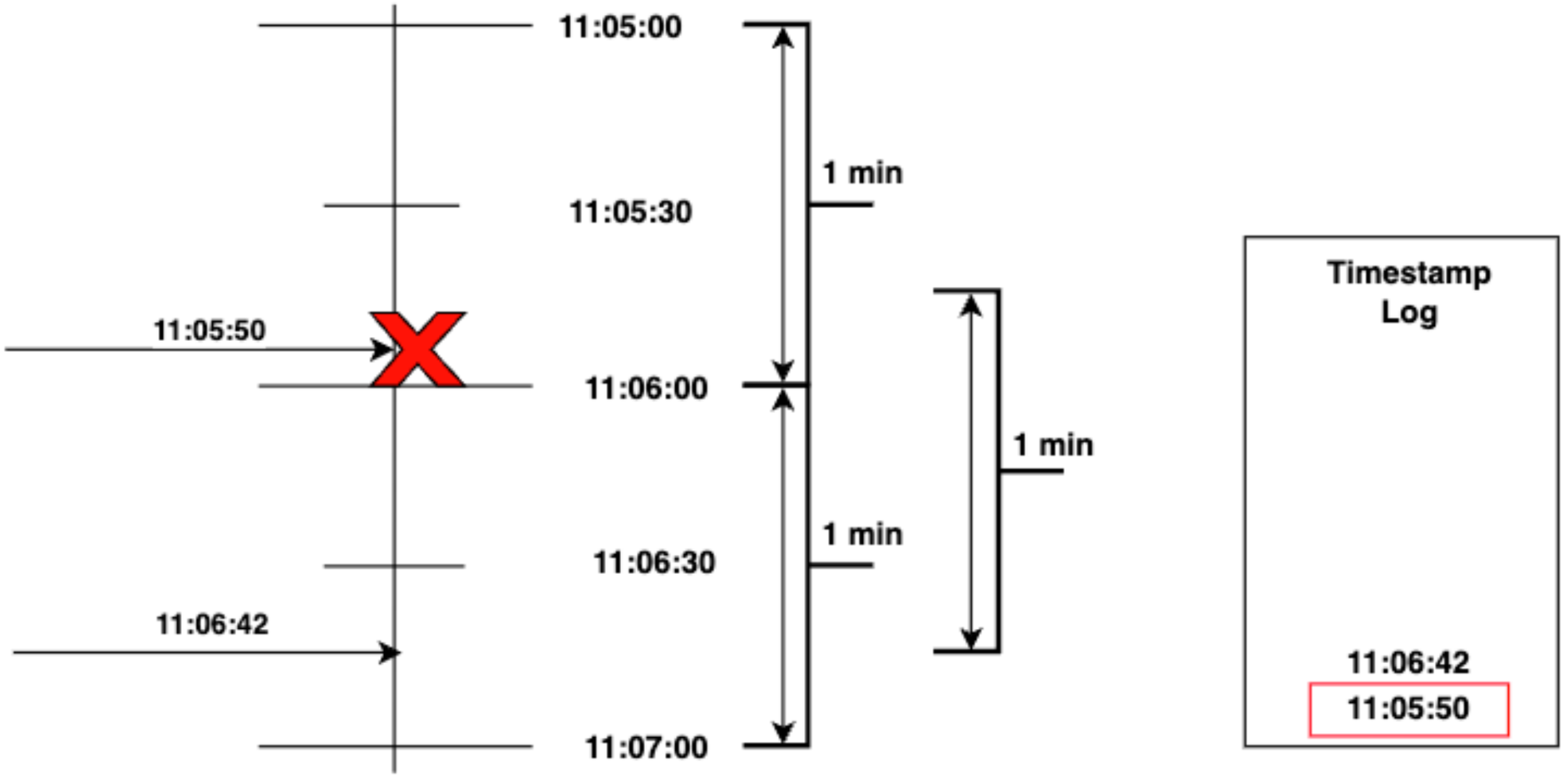




# SLIDING WINDOW LOG





# SLIDING WINDOW LOG



## PREDNOSTI

- Rešava problem *Fixed Window Counter*-a koji ne dozvoljava veći broj zahteva za obradu oko granica intervala
- Precizan

## MANE

- Troši više resursa za računanje jer je potrebno da se čuvaju sva vremena zahteva (prihvaćenih i odbijenih) i uklone zastareli *timestamp*-ovi kako pristižu novi zahtevi

# SLIDING WINDOW COUNTER



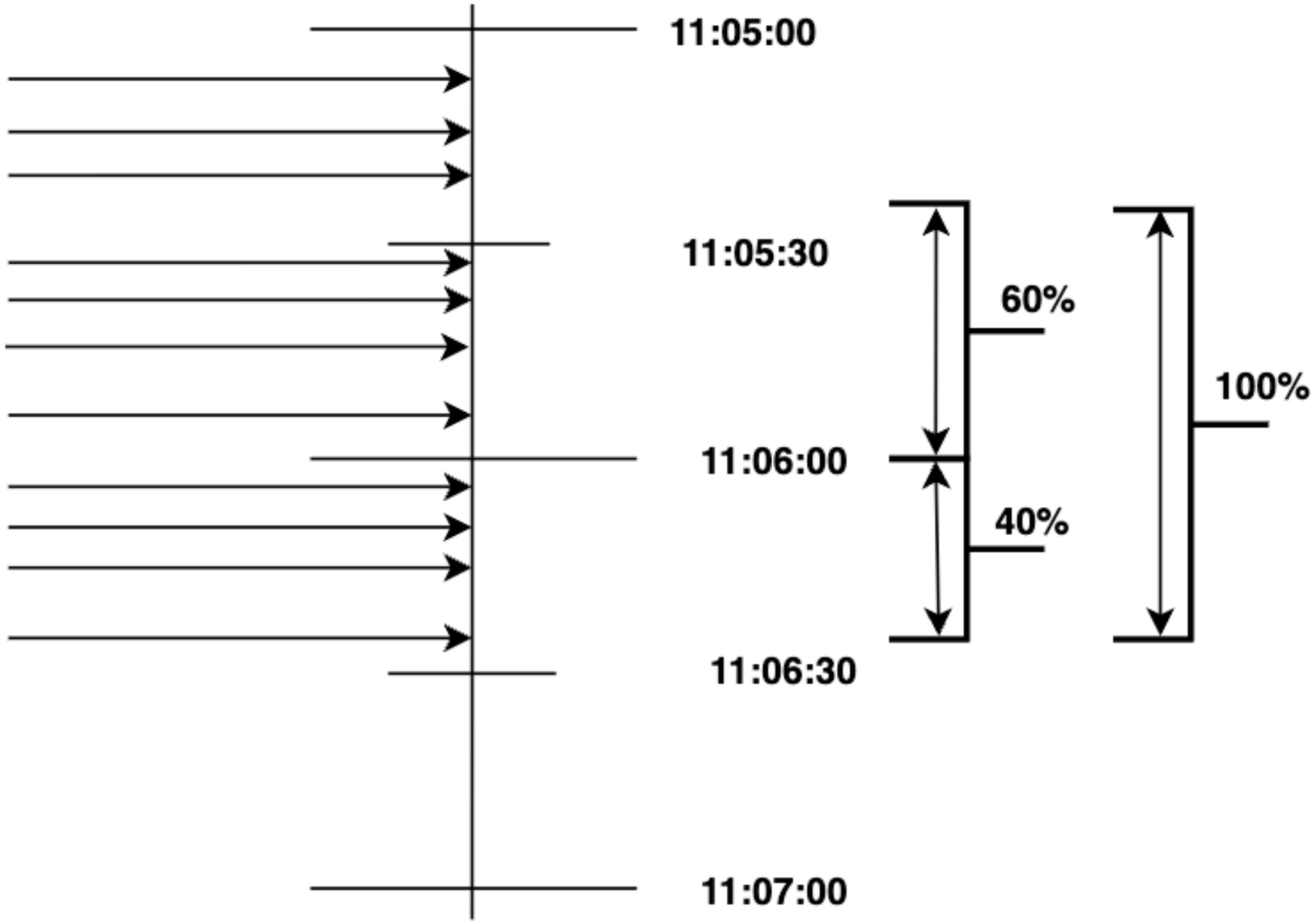
## IDEJA

- Hibridni algoritam između *Fixed Window Counter* i *Sliding Window Log* algoritama
- Donekle rešava probleme velike upotrebe resursa za izračunavanje i problem većeg broja zahteva oko granica prozora
- Formula za računanje broja trenutnih zahteva:

  *zahtevi u trenutnom prozoru + zahtevi u prethodnom prozoru * procenat preklapanja intervala*

# SLIDING WINDOW COUNTER





# SLIDING WINDOW COUNTER



## PRIMER

- Konfiguracija dozvoljava maksimalno 10 zahteva u minuti
- Već je dobijeno 7 zahteva u prethodnom prozoru i 4 u trenutnom prozoru
- Poslednji zahtev u trenutnom prozoru je došao na isteku 40% trenutnog prozora
- Prema prethodno navedenoj formuli izračunava se ukupan broj zahteva za trenutni prozor kao:
    - 4+7*0,6 = 8,2 (zaokruženo na 8 zahteva)
- Još uvek je ispod dozvoljenog praga, tako da je novi zahtev prihvaćen

# SLIDING WINDOW COUNTER



## PREDNOSTI

- Rešava problem većeg broja zahteva za obradu oko granica intervala
- Precizan

## MANE

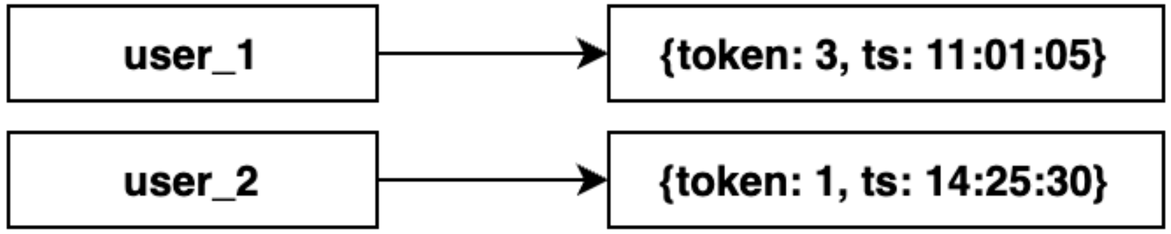
- I dalje troši dosta resursa za računanje jer je potrebno da se čuvaju sva vremena zahteva (prihvaćenih i odbijenih) i uklone zastarela vremena tokom pristizanja novih zahteva

# IZAZOVI



## ČUVANJE PODATAKA O BROJAČIMA

- Ove informacije se mogu čuvati za svakog korisnika posebno u memoriji ili nekom sistemu koji je namenjen za to
- Redis je dobro rešenje za čuvanje ovih podataka
- Svaki korisnik može da bude ključ dok vrednost može da sadrži vremensku odrednicu i broj tokena



# IZAZOVI

## AUTOMATIZACIJA SA STRANE KORISNIKA

- Klijenti mogu da saznaju da li se približavaju graničnim vrednostima tako što će proveriti vrednosti HTTP zaglavlja
- Twitter koristi sledeća zaglavlja:
  - X-Rate-Limit-Limit – maksimalni broj zahteva za endpoint
  - X-Rate-Limit-Remaining – broj preostalih zahteva za prozor od 15 minuta
  - X-Rate-Limit-Reset – preostali prozor pre nego što se limit resetuje u sekundama UTC epohe
- Github koristi sledeća zaglavlja:
  - X-RateLimit-Limit – maksimalni broj zahteva koji je dozvoljen po satu
  - X-RateLimit-Remaining – preostali broj zahteva u trenutnom prozoru
  - X-RateLimit-Reset - vreme kada će se prozor resetovati u sekundama UTC epohe

## TRKA ZA RESURSIMA

- Uobičajeni scenariji u distribuiranim sistemima
- Na primer, može da se desi scenario gde 2 različita zahteva dolaze na dva različita servera za *Throttling* a oba su dobila istu vrednost za brojač zahteva
- Ovo se može rešiti implementacijom zaključavanja ali će to usporiti system
- Drugi način je da se npr. koriste Redis sorted setovi[1]

[1] Redis sorted sets https://redis.io/docs/data-types/sorted-sets/

# REFERENCE




- SIMIĆ M. NAPREDNI ALGORITMI I STRUKTURE PODATAKA KROZ NOSQL I BIGDATA SISTEME. FAKLUTET TEHNIČKIH NAUKA. OGRANIČENJE STOPE PRISTUPA (UDŽBENIK U PROCESU OBJAVLJIVANJA)
- DORAL L. P. AN INTRODUCTION TO COMPUTER NETWORKS. TOKEN BUCKET RATE LIMITING http://intronetworks.cs.luc.edu/current2/html/tokenbucket.html
- LANG L. ET AL. AN IMPROVED TOKEN BUCKET ALGORITHM FOR SERVICE GATEWAY TRAFFIC LIMITING https://www.scitepress.org/Papers/2019/80988/80988.pdf
- WONG C. ET AL. EMPIRICAL ANALYSIS OF RATE LIMITING MECHANISMS https://www.pdl.cmu.edu/PDL-FTP/associated/RAID05.pdf
- GHOBADI M. ET AL. TRICKLE: RATE LIMITING YOUTUBE VIDEO STREAMING https://people.csail.mit.edu/ghobadi/papers/trickle_atc_2012.pdf
- ALIBABA CLOUD. THROTTLING SOLUTIONS IN STANDALONE AND DISTRIBUTED SCENARIOS https://bit.ly/3jgxxbC

# KOJA SU VAŠA PITANJA?