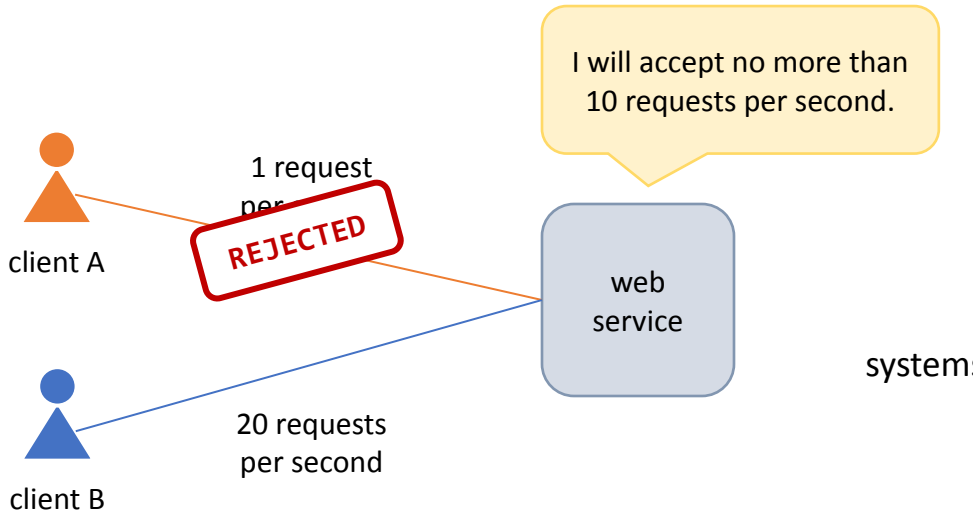
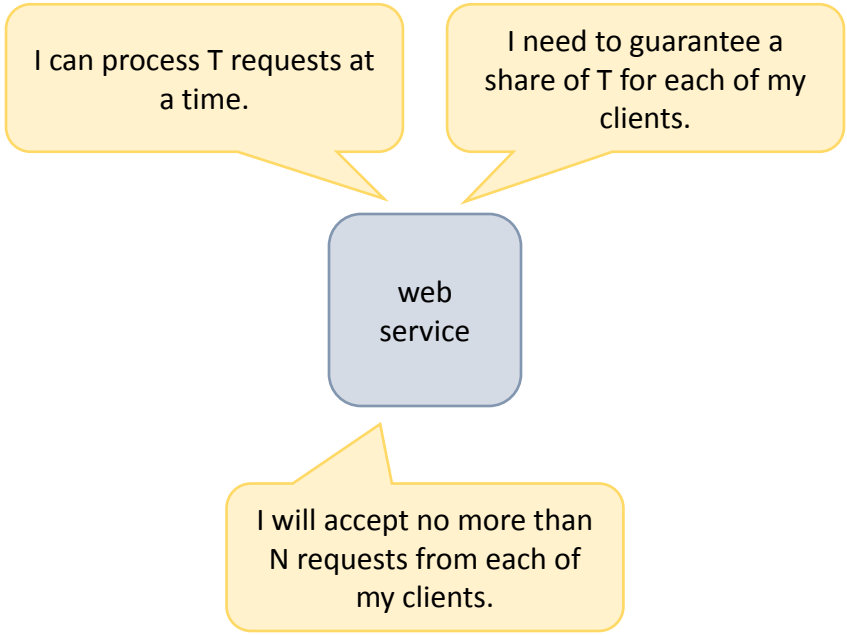
# Rate limiting




**noisy neighbor** problem

can happen in **multi-tenant systems**,
systems where common resources are shared by **multiple clients**

# Rate limiting



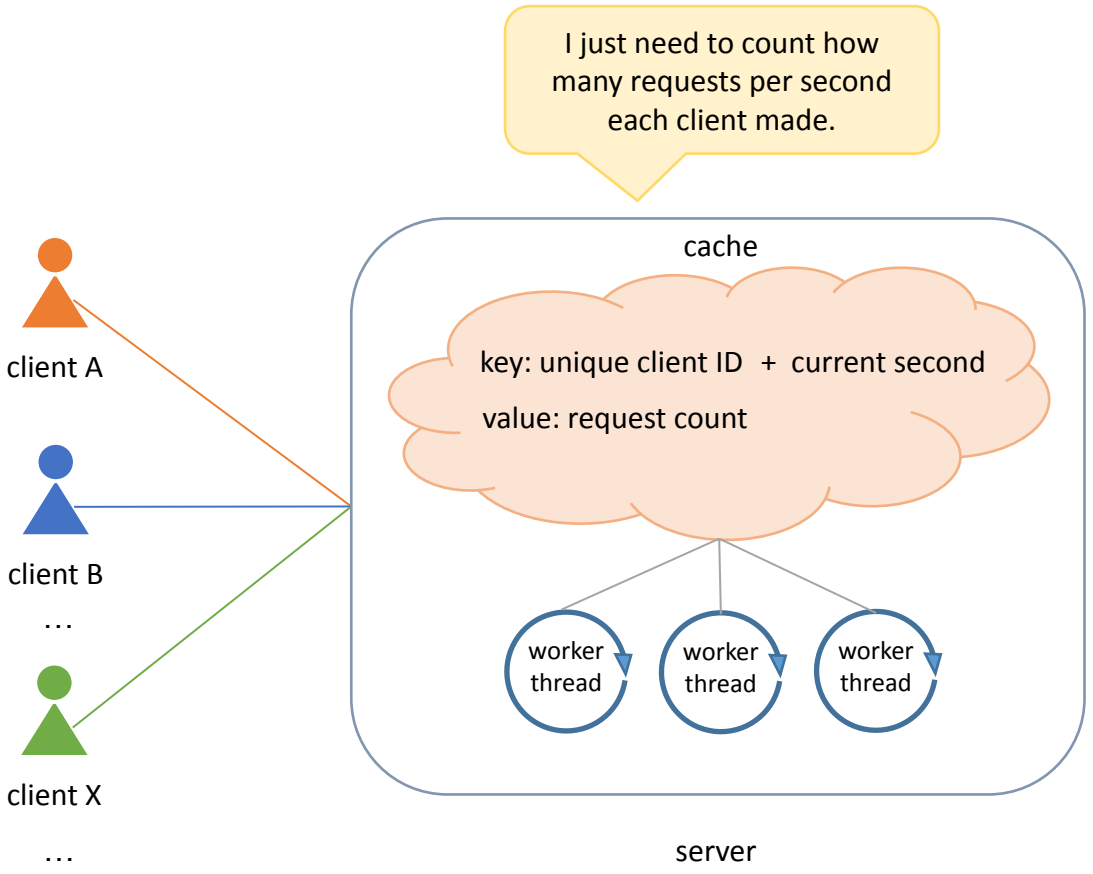

## that's what I don't know yet ...

- how long the time interval should be
- how to account for heterogeneity of requests

## so I'll make assumptions ...

- 1 second (ideally configurable)
- all requests are equal (for now)

# Rate limiting




## Where do we store request counters?

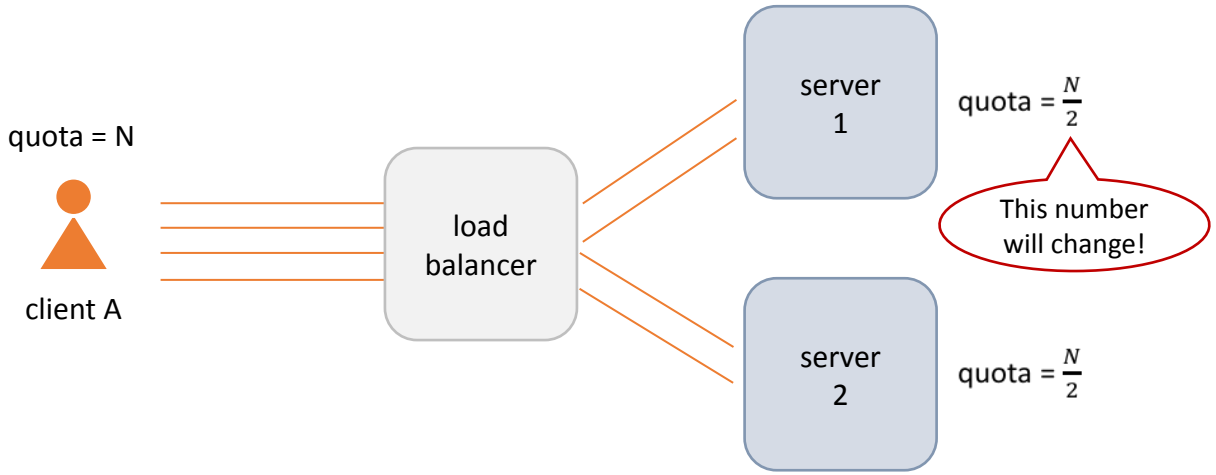
| memory (local cache) | disk (embedded database ) |
|---|---|

## Where do we keep track of time?

| in cache key | in cache value |
|---|---|

| problems | solution |
|---|---|
| • concurrency | locks, atomic variables |
| • cache size | eviction policy (LRU), time-based expiration (active) |

# Rate limiting

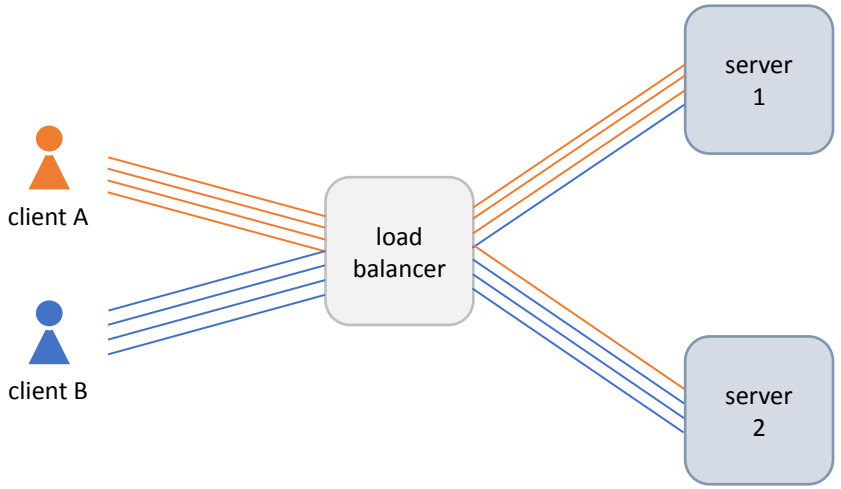



How to share static information (e.g. configuration changes) with all servers in the cluster in a timely manner?

- use configuration management tools
- use a daemon process to fetch data periodically from a shared storage (database, object storage)
- use gossip protocol to solve the membership problem

problems

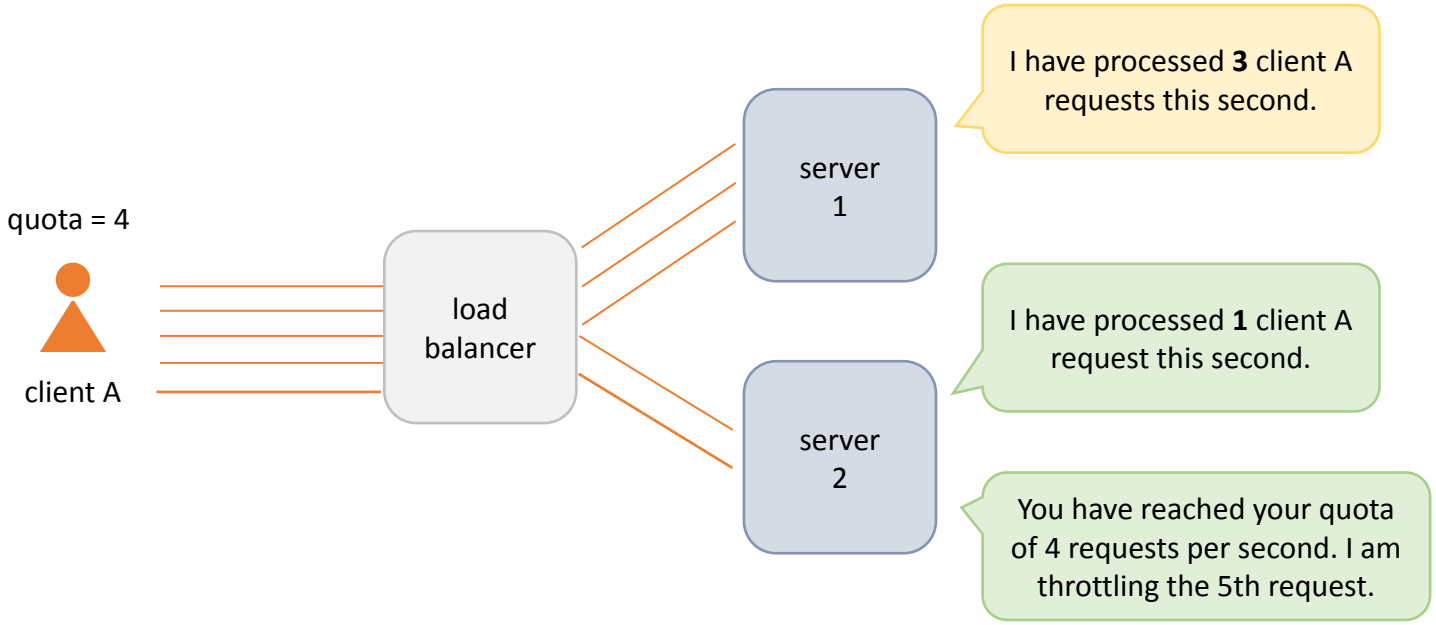
- request load balancer does not guarantee uniform distribution for each individual client
- uniformly distributed connections ≠ uniformly distributed requests
  (L4 vs L7 load balancers, which we will discuss later in the course)



# Rate limiting

## What should clients do with throttled requests?

- retry
  (only with exponential backoff and jitter)
- fallback to an alternative solution
  (send to queue and process with lower priority on the side ,
  buffer in memory or save on disk and replay later)
- batch requests
  (to reduce the number of requests)




## How to share dynamic information (e.g. frequently updated counters) with all servers in the cluster in a timely manner?

- use gossip protocol to exchange data
- discover peers using a service registry or seed nodes advertised by DNS
- use either TCP or UDP network protocol

### problems

- peer-to-peer communication will eventually become the bottleneck for large clusters with many clients

- use distributed cache
- use hash partitioning and consistent hashing to build scalable and reliable cache

# Rate limiting

reliability

scalability

performance

network protocols

how to choose a network protocol

thread per request thread model

local cache and data eviction

embedded database

DNS

peer discovery

service registry

gossip protocol

persistent connections

retries with exponential backoff and jitter

fallback

batching

compression

partitioning

consistent hashing

request routing

load shedding

backpressure

load balancer

reverse proxy

API gateway

service mesh

token bucket algorithm