

Deploying Fast and Large Scale Web Applications

Thierry Sans

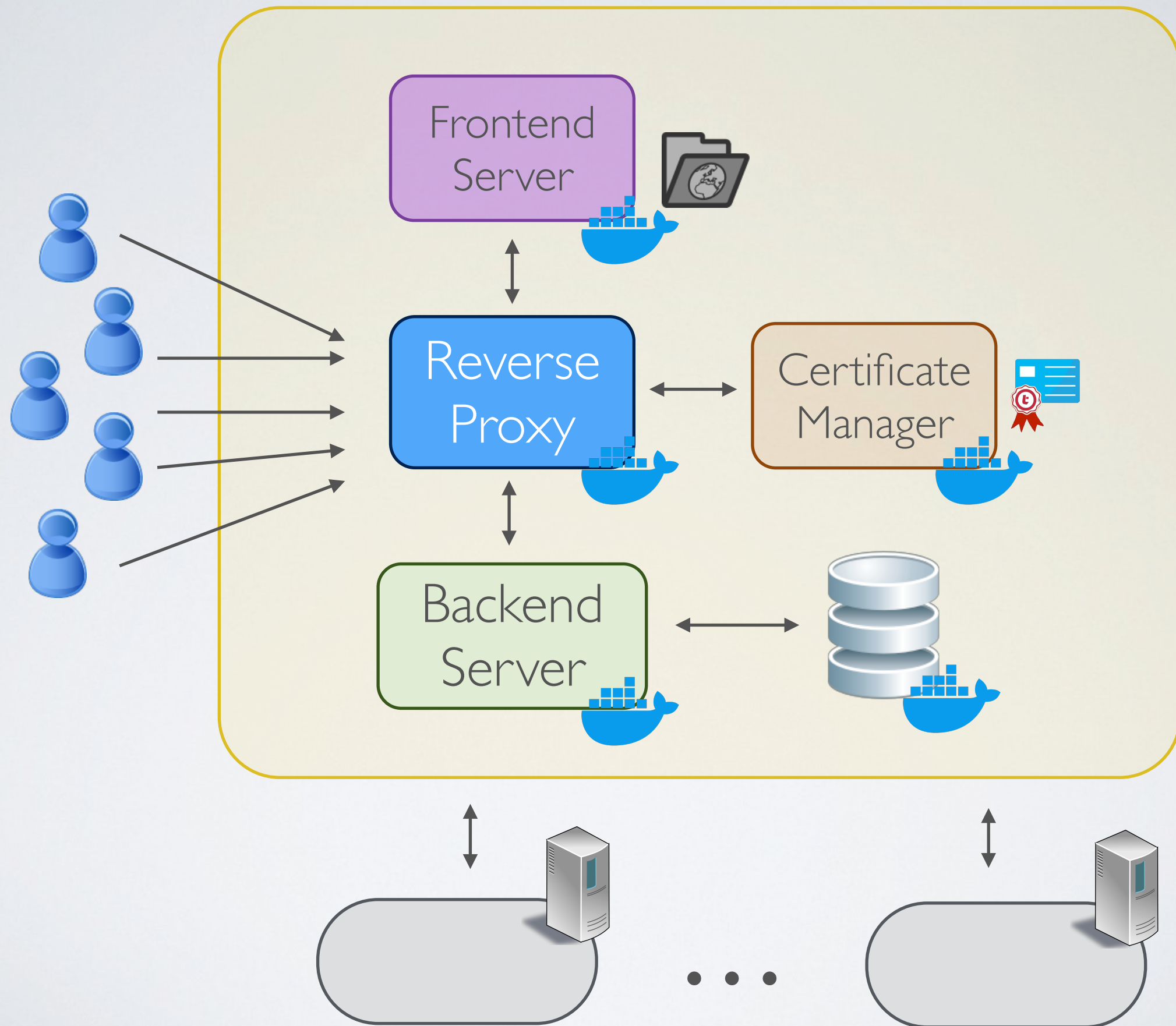
Users respond to speed

“Amazon found every 100ms of latency cost them 1% in sales”

“Google found an extra .5 seconds in search page generation time dropped traffic by 20%”

<http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/>

Our microservice deployment (so far)

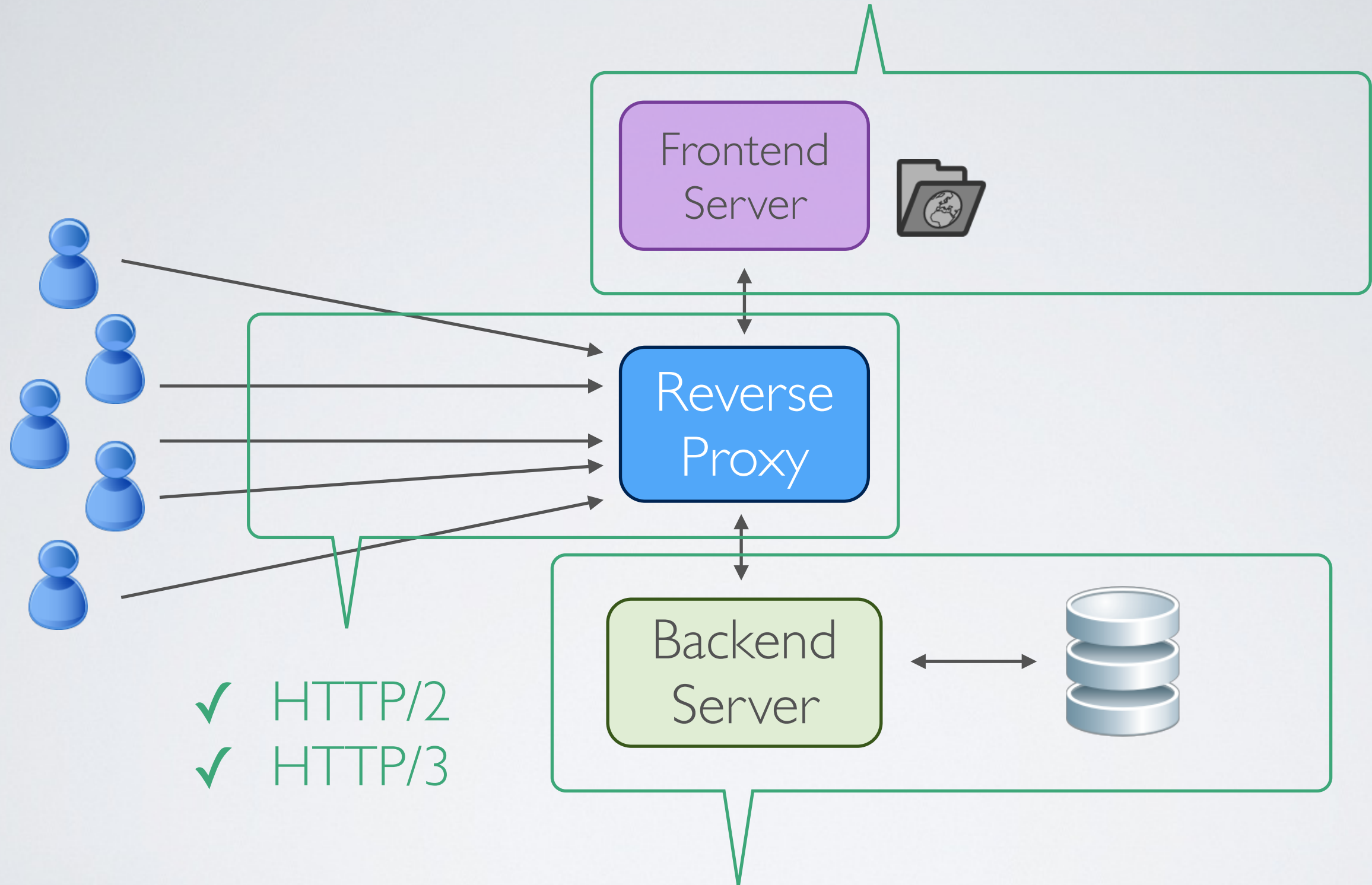


Problems

- How to increase the throughput?
- How to scale to serve millions of users?

Solutions

- ✓ Web Packing
- ✓ Progressive Web Applications (PWA)

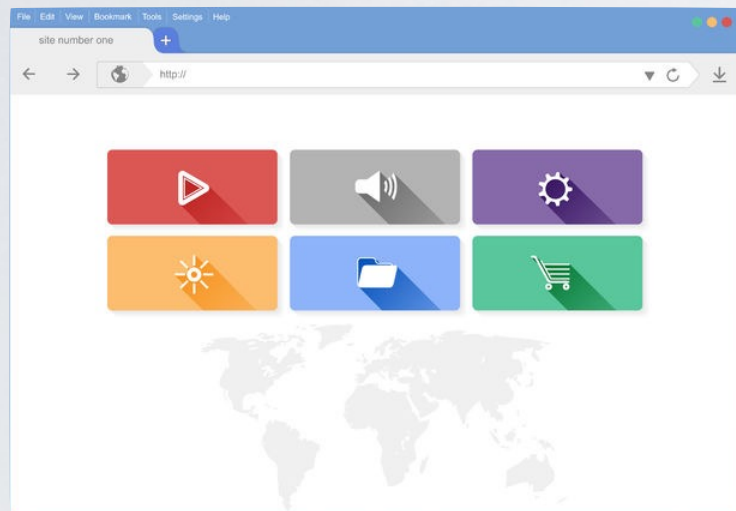


- ✓ Faster web caching
- ✓ Better scalability with load balancer and CDN

Frontend packing

The problem

Browser



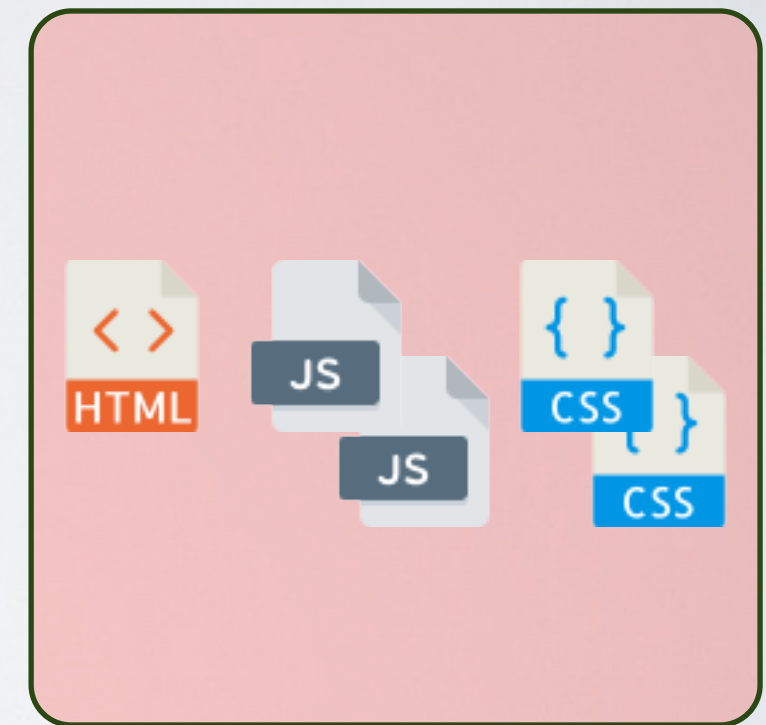
GET /

GET /js/lib.js

GET /js/index.js

GET /style/index.css

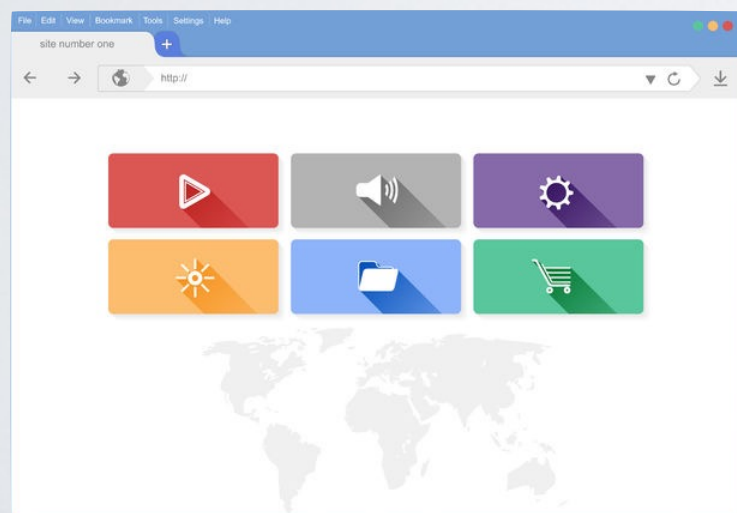
GET /style/generic.css



Frontend Server

The solution - using a frontend packer

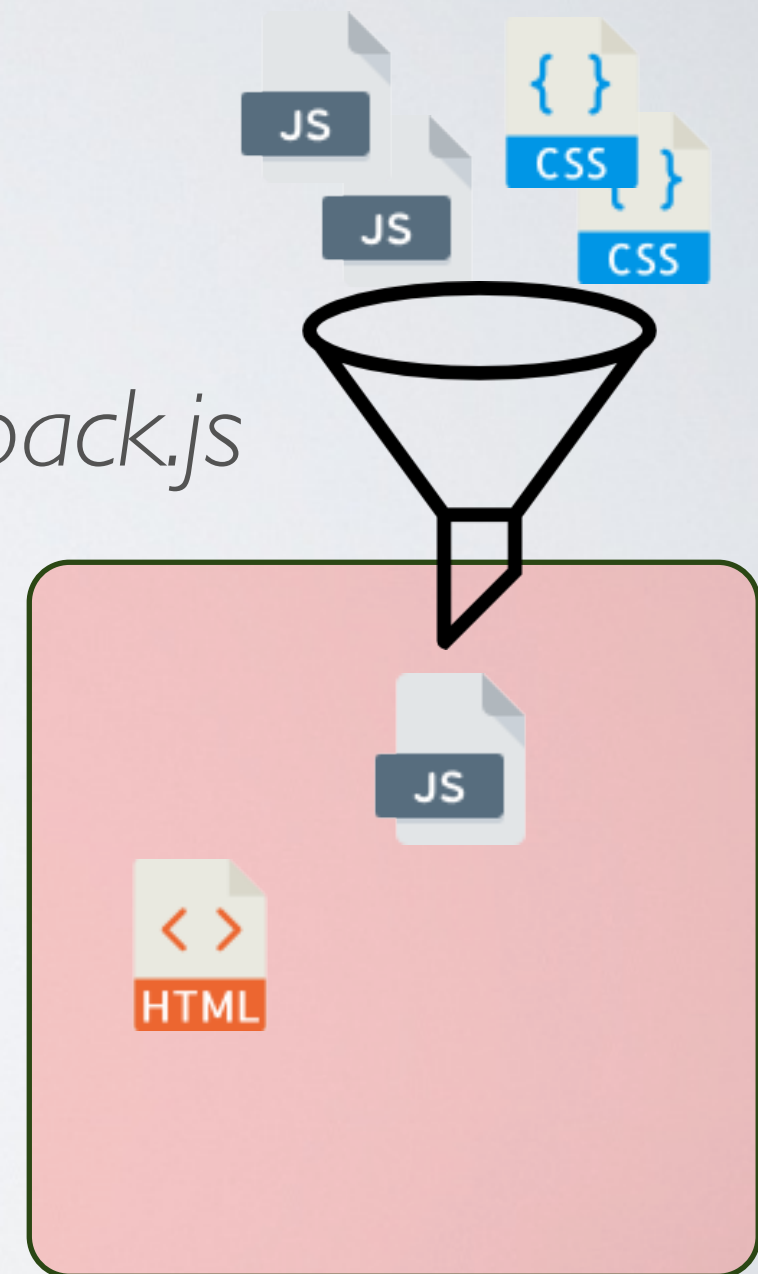
Browser



GET /

GET /js/bundle.js

e.g. *webpack.js*



Frontend Server

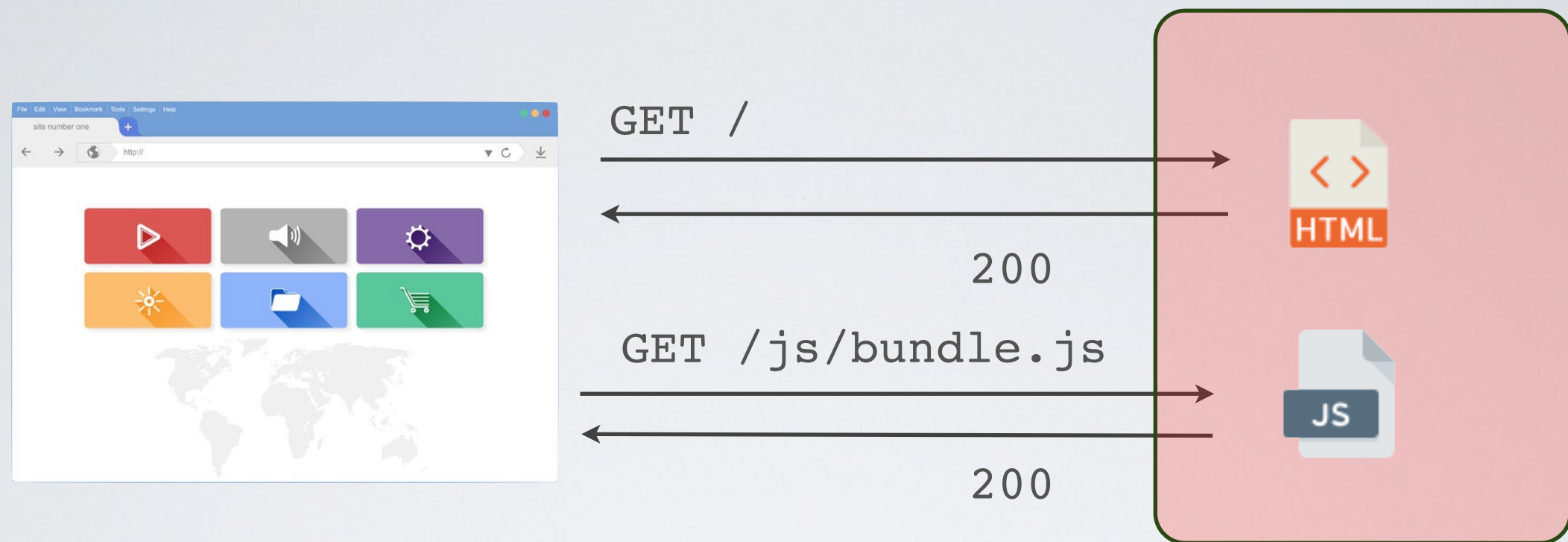
HTTP/2

HTTP/2

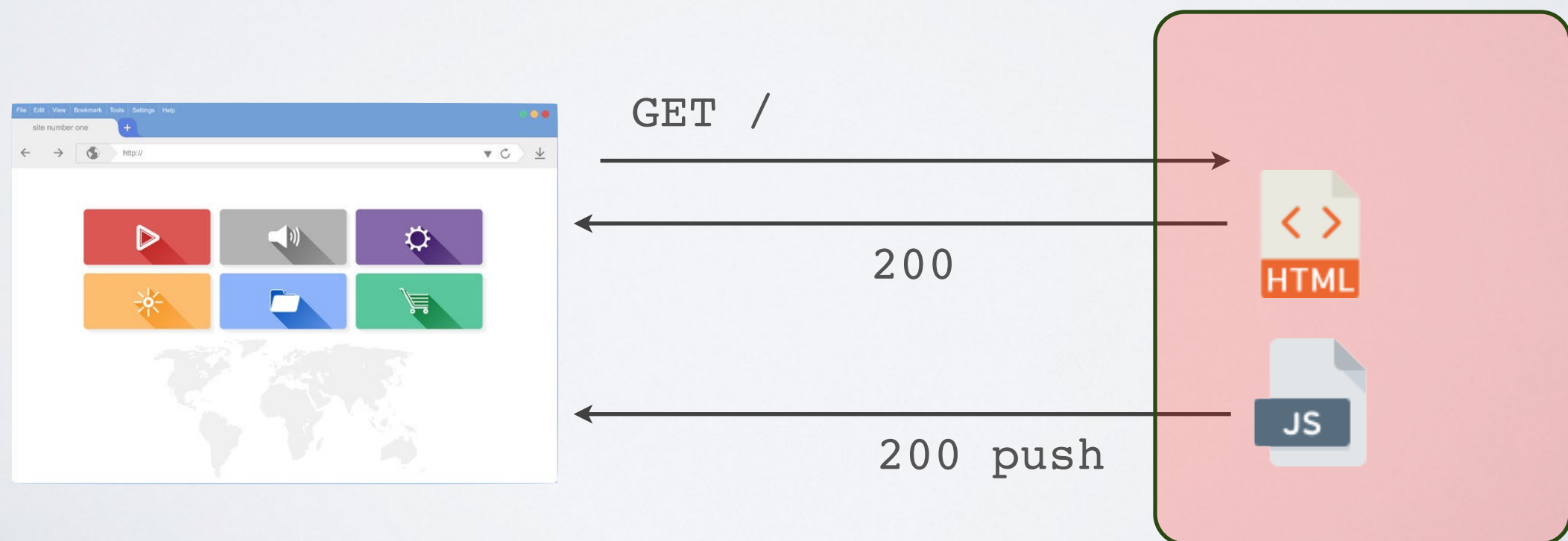
HTTP/2 enables multiplexing

- ➡ send multiple HTTP responses for a given request (a.k.a push)
 - Proposed by Google (called SPDY)
 - Adopted as an standard in 2015 (RFC 7540)
 - HTTP/2 is compatible with HTTP/1 (same protocol)

HTTP 1.1



HTTP 2.0



Great technology ... but nobody uses it!

Google is planning to remove the push feature from Chrome!

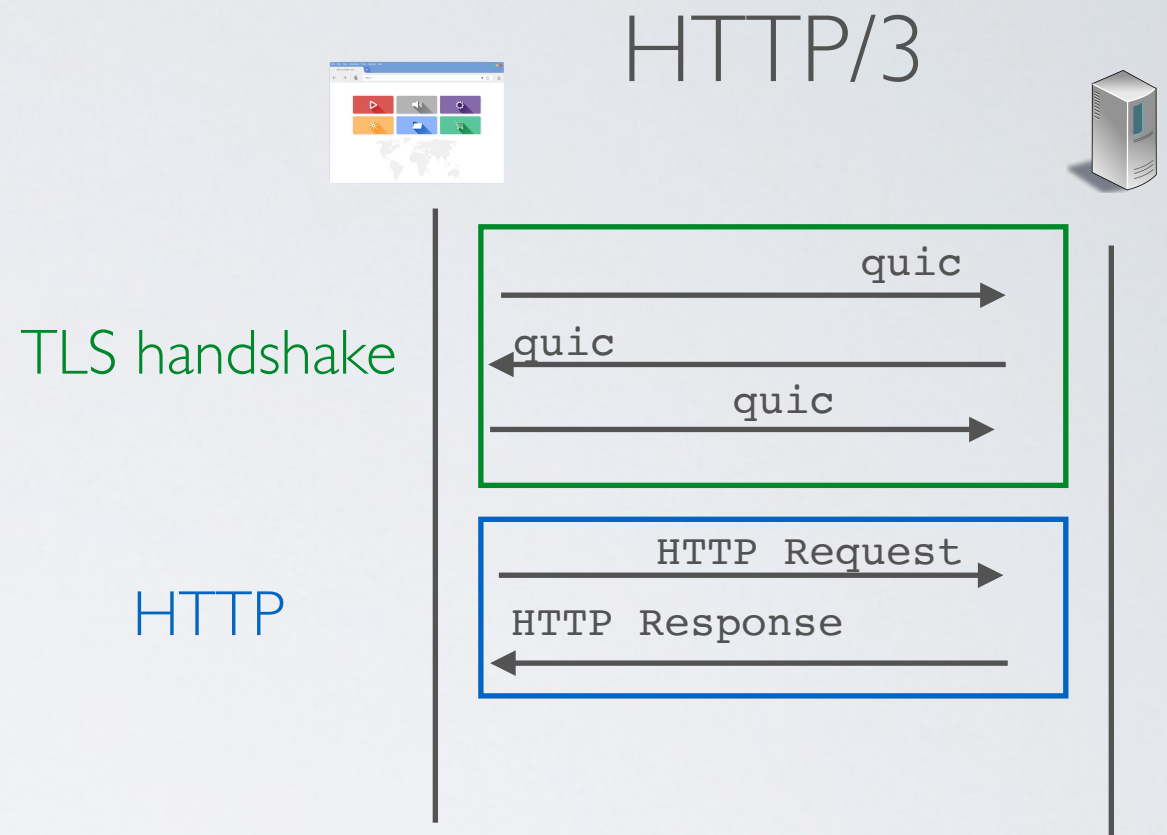
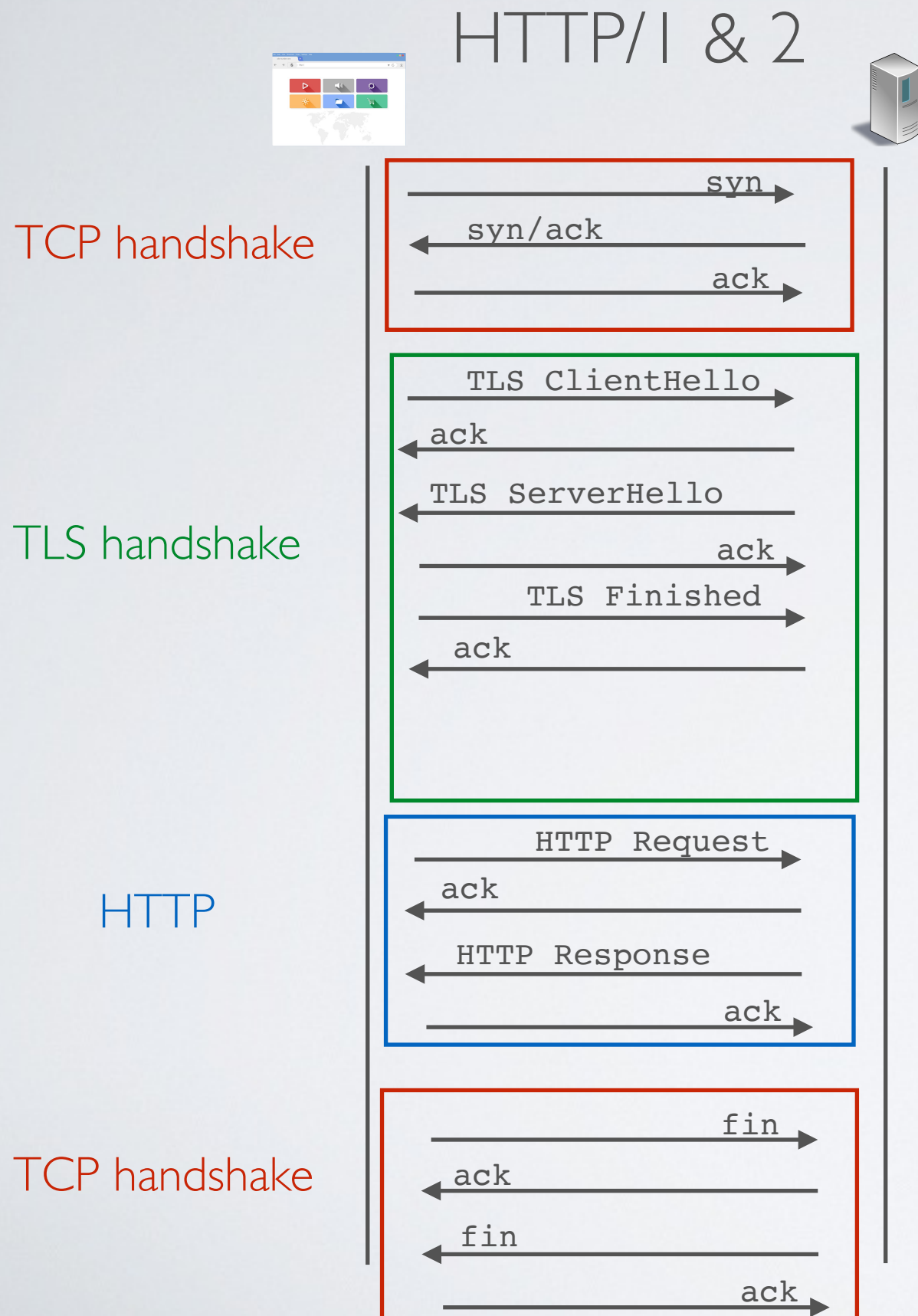
"Almost five and a half years after the publication of the HTTP/2 RFC, server push is still extremely rarely used. Over the past 28 days, 99.95% of HTTP/2 connections created by Chrome never received a pushed stream, and 99.97% of connections never received a pushed stream that got matched with a request. These numbers are exactly the same as in June 2019"

source <https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/vOWBKZGoAQAJ?pli=1>

HTTP/3

(work in progress)

HTTP/3 (standard draft)



➔ Use UDP instead of TCP

Chrome in Dec'19
Firefox in Jan'20

PWA

Progressive Web Applications

The idea

- ➔ A web application that can be installed on your system
 - Relies on browser local storage to store the frontend (and checks for update with the server)
 - Relies on Web-Workers for caching and communication

Backend Web Caching

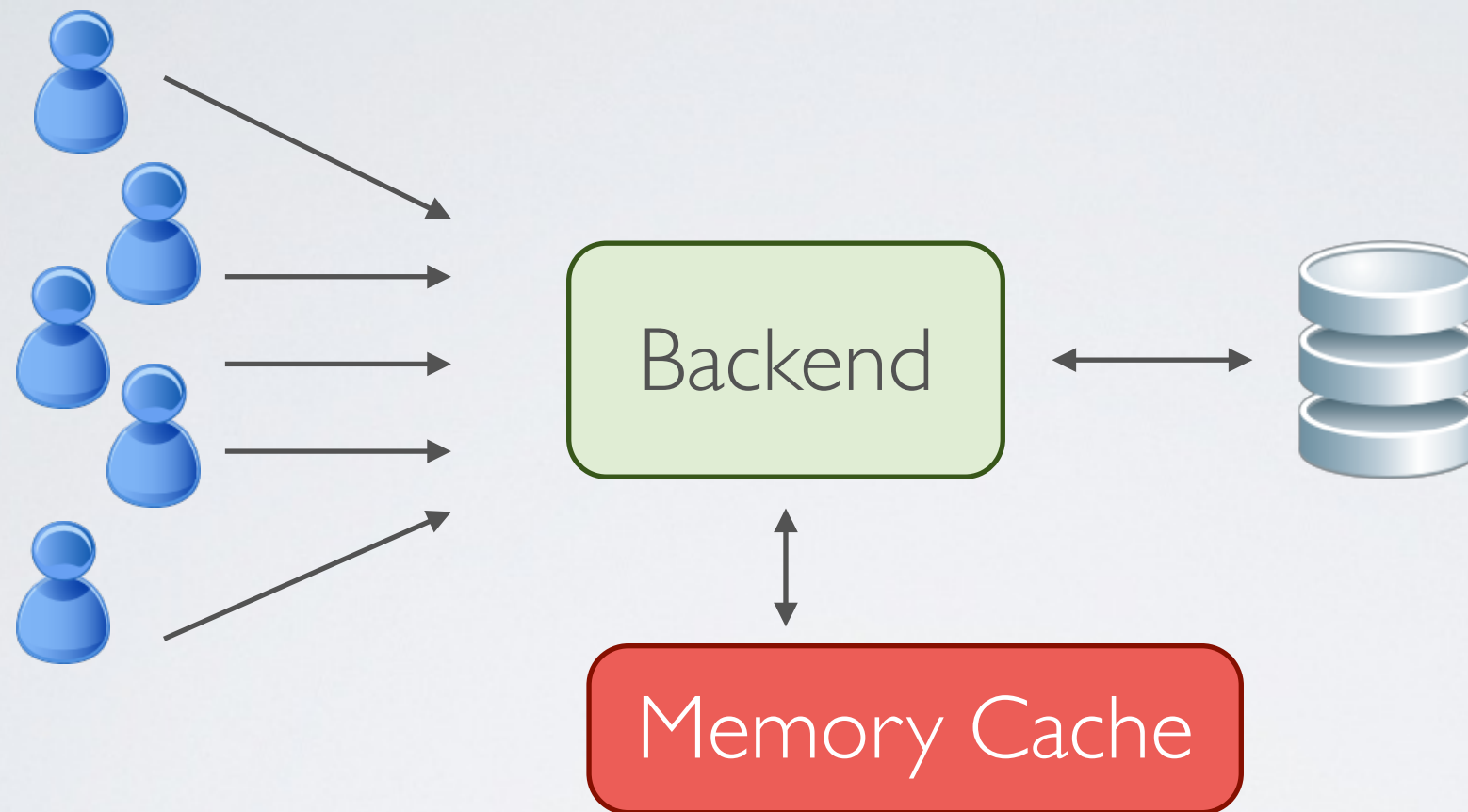
How to improve response time?

Processing the request means:

1. Parse the HTTP request
2. Map the URL to the handler
3. Query the database or third-party API
4. Compute the HTTP response

DB and API accesses are expensive (time and money when your host charges you each access)

Fine-grained caching with the web application



Cache controlled by the program

- Specific for each app

- ✓ Good for caching database requests and storing sessions

- ➔ Popular memory cache : *Memcached*

Distributed Shared Cache : Memcached

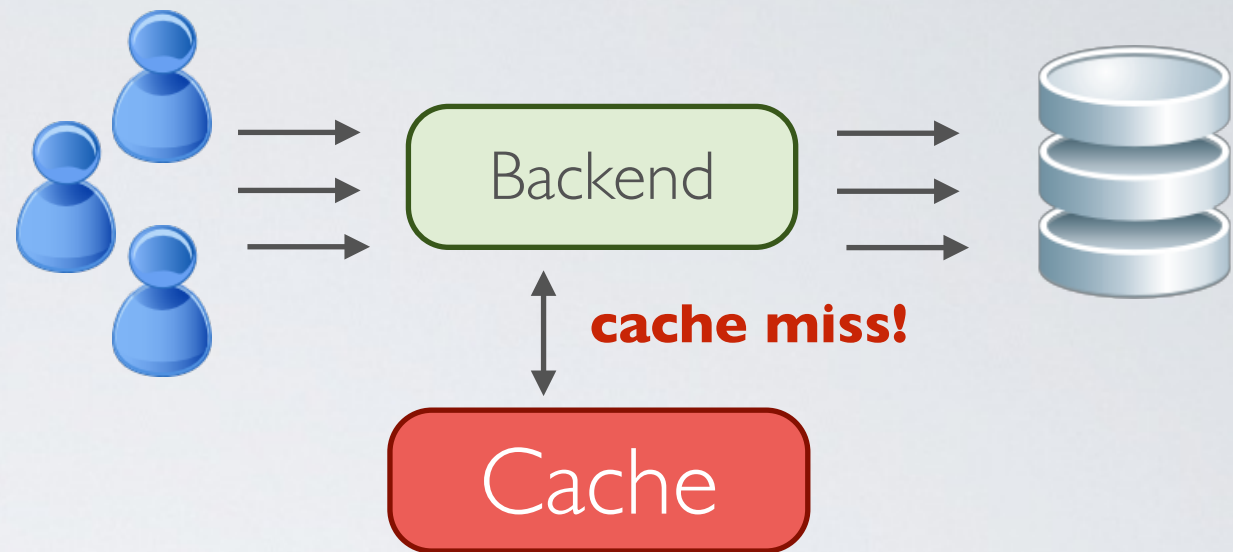
<http://memcached.org/>

- Store key/value pairs in memory
- Throw away data that is the least recently used

A typical cache algorithm

```
retrieve from cache
if data not in cache:
    # cache miss
    query the database or API
    update the cache
return result
```

Cache Stampede (a.k.a dog piling)



Problem:

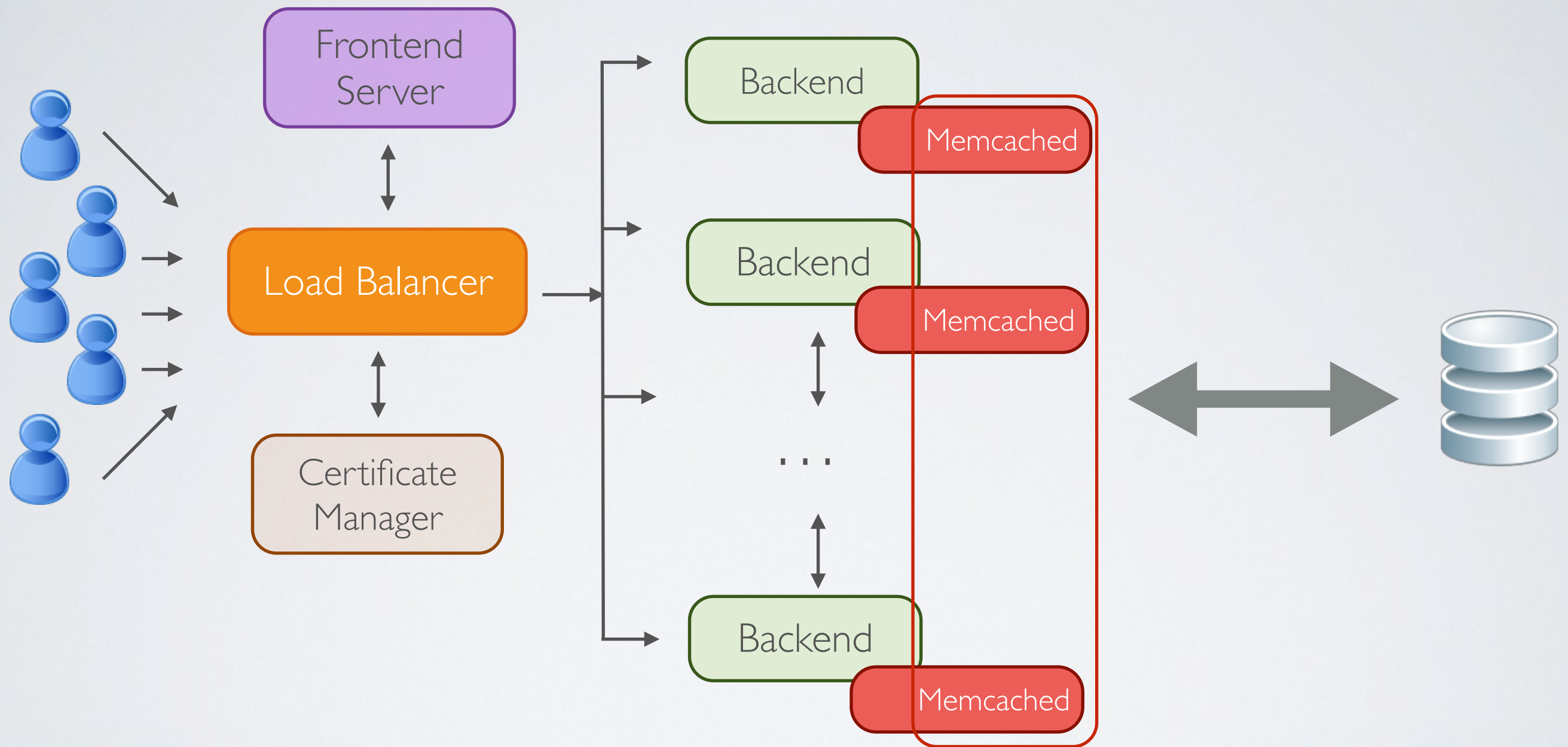
Multiple concurrent requests doing the same request because cache was cleared

Solution:

- update the cache instead of clearing it after an insert
 - a page view will never query the database
- ➡ Requires cache warming

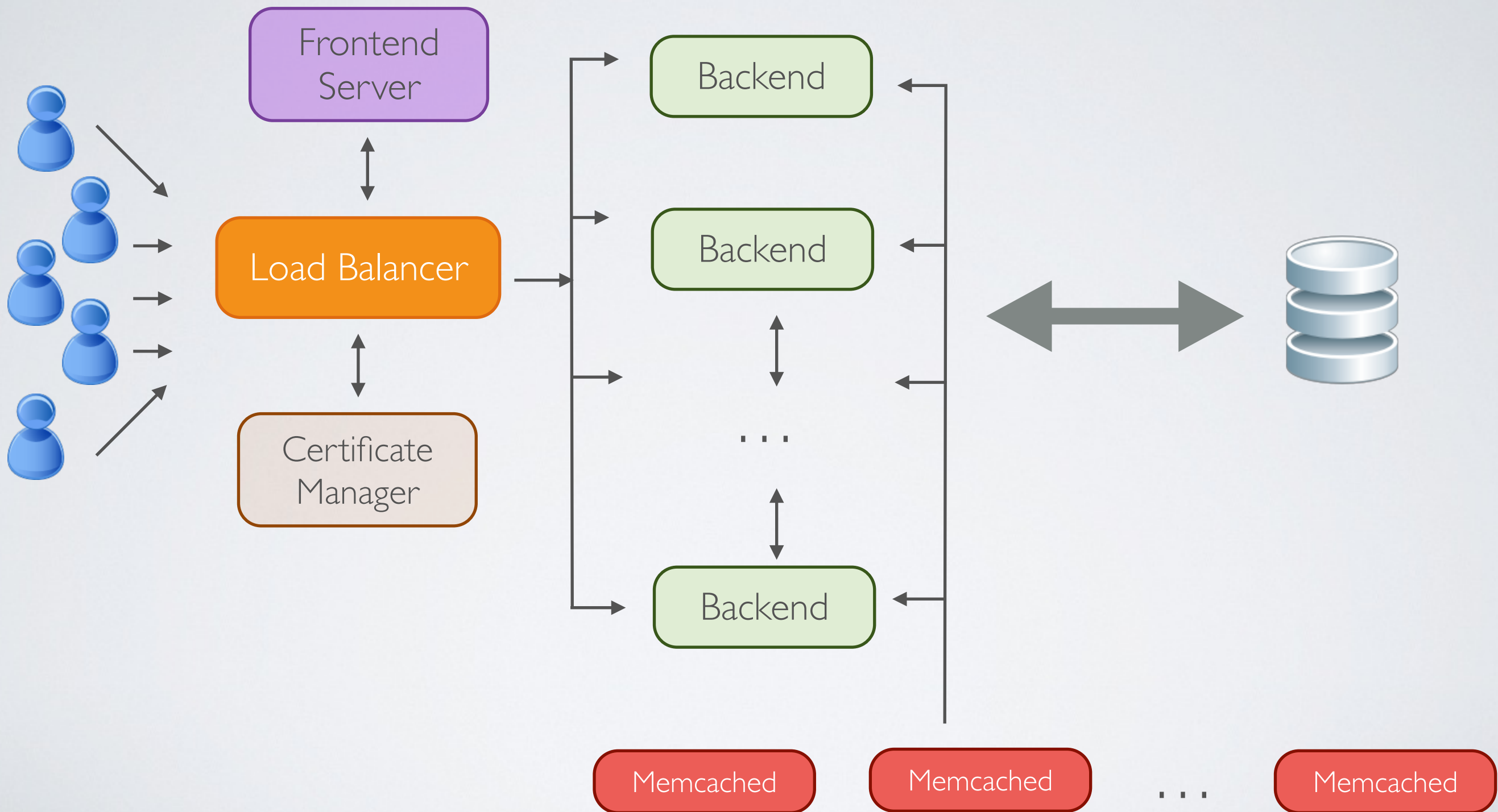
Scaling The Backend

Serving multiple apps with a load balancer

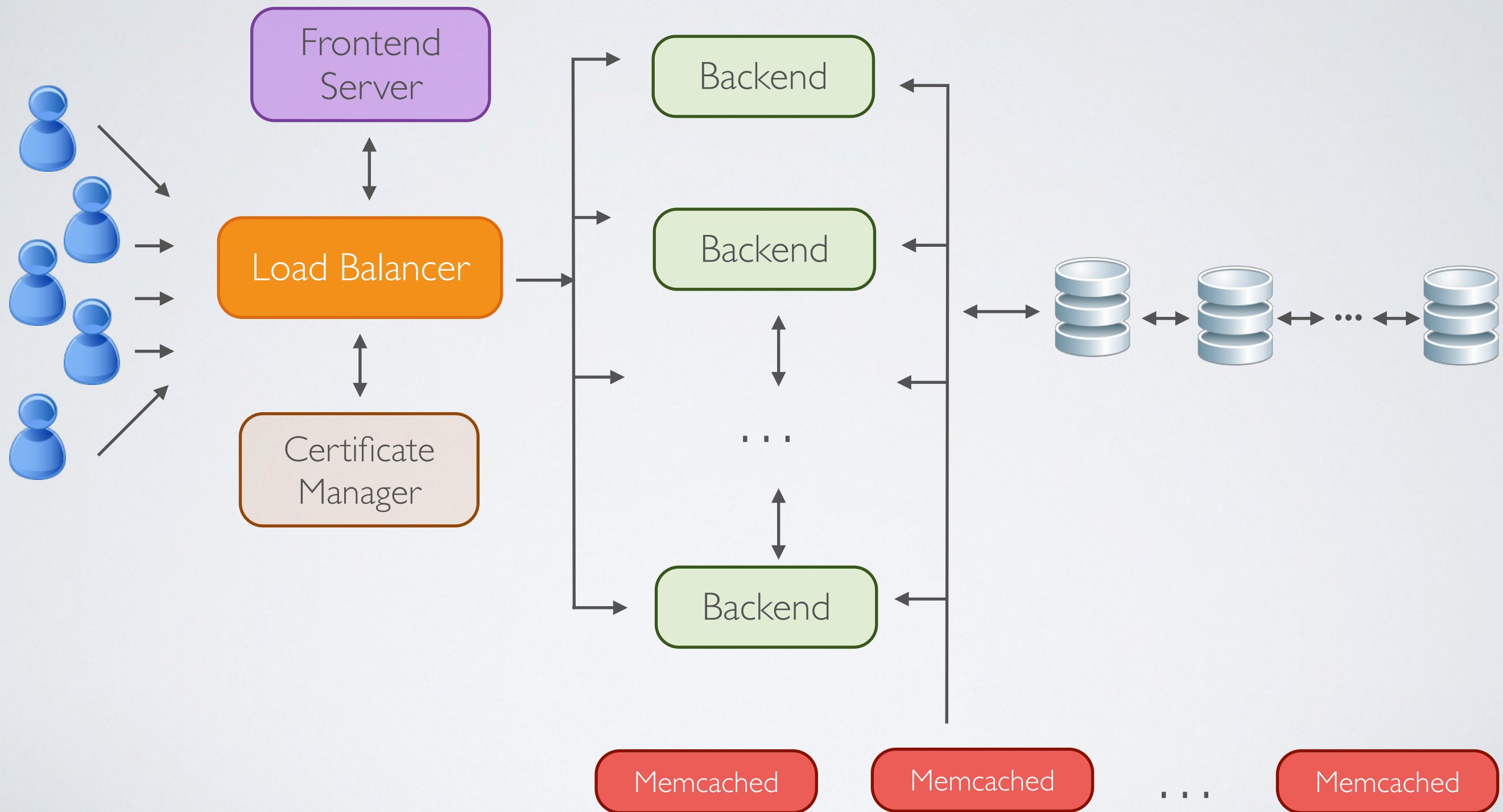


This is not an efficient cache

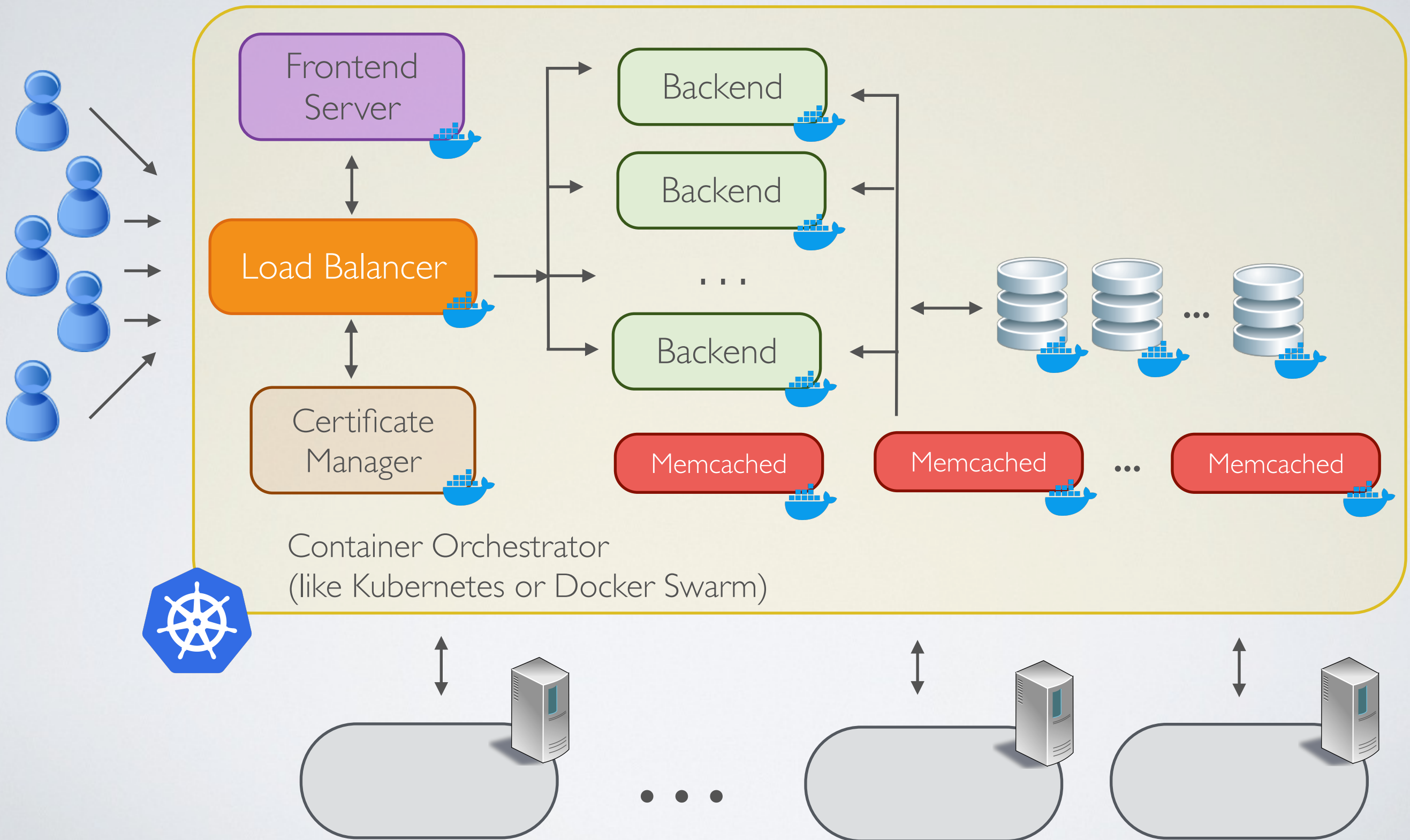
Distributed Shared Cache



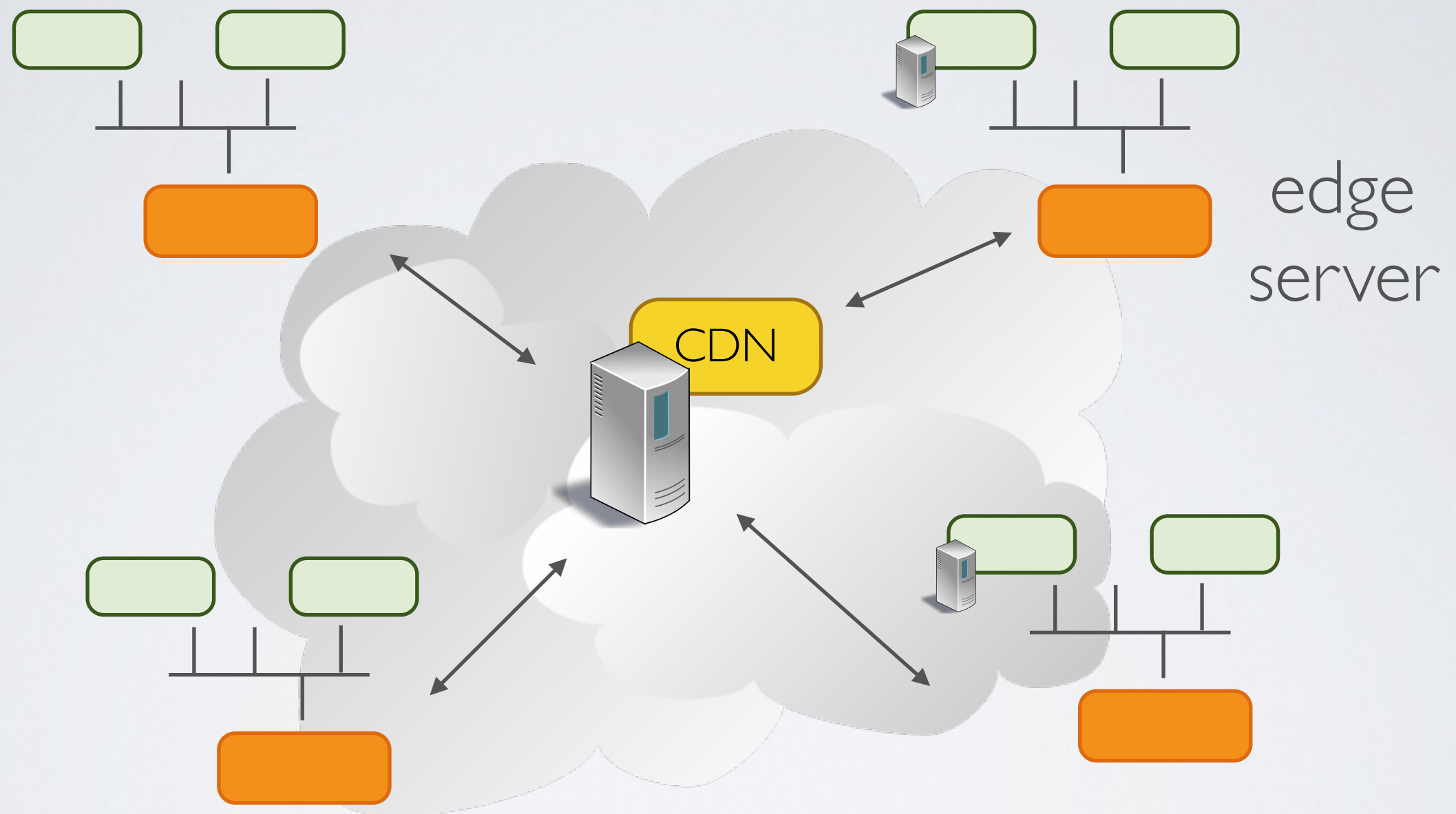
Database Sharding



Automatic Scaling with container Orchestration



CDN : Content Distribution Network



Example : Akamai, Cloudflare