Deploying Fast and Large Scale Web Applications

Thierry Sans

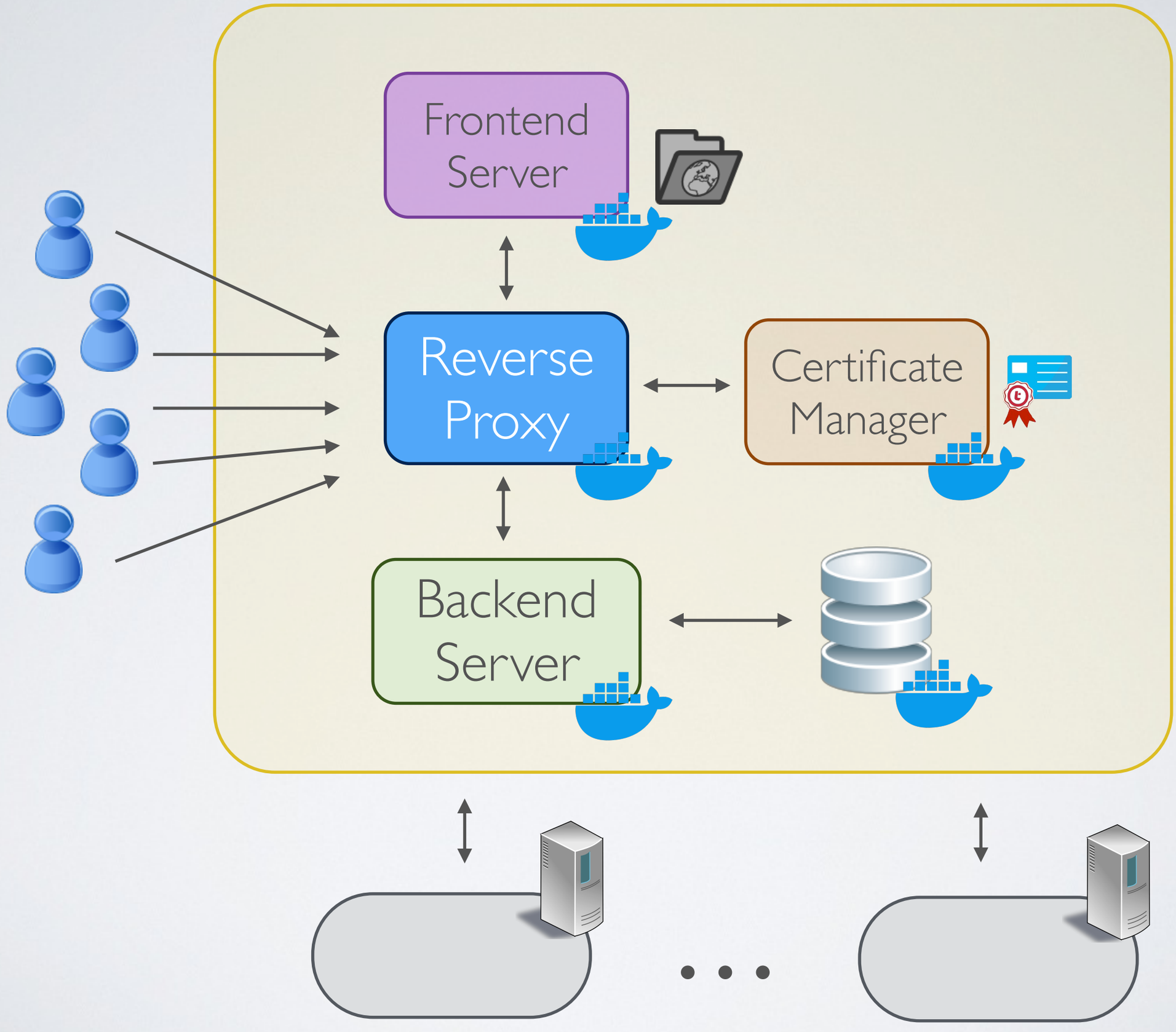
Users respond to speed

"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)

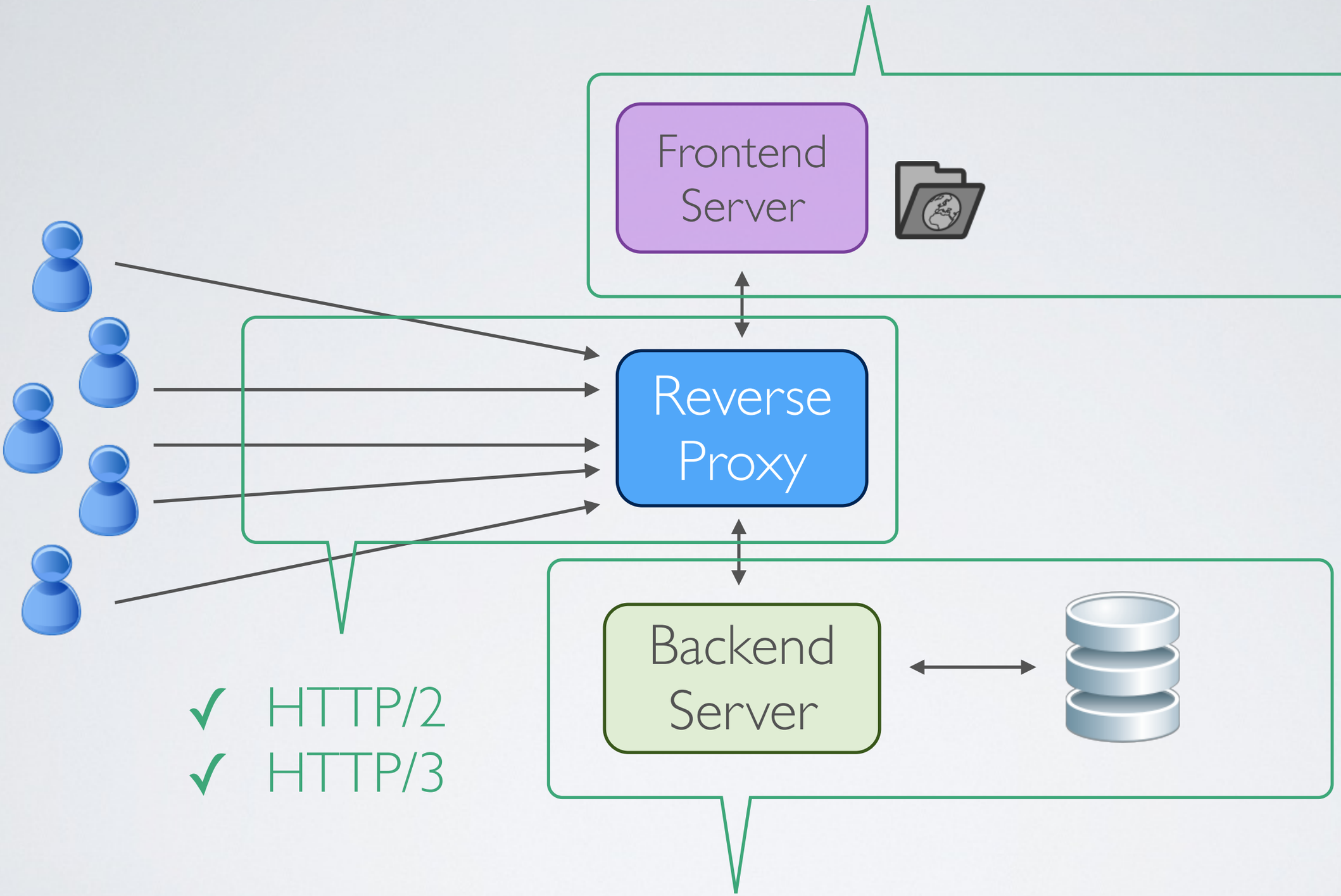


Problems

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

Solutions

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



- ✓ HTTP/2
- ✓ HTTP/3

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

Backend Web Caching

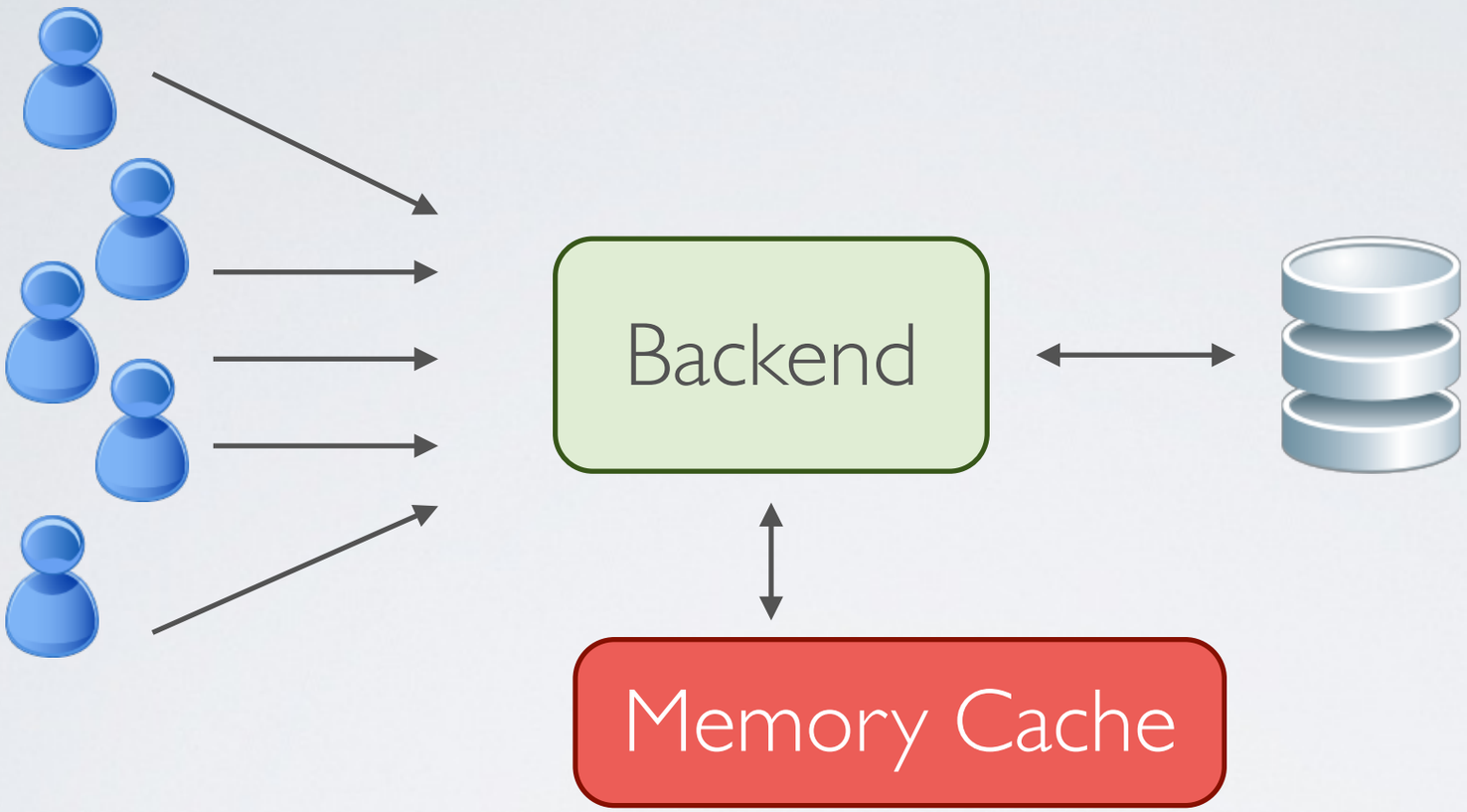
How to improve response time?

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

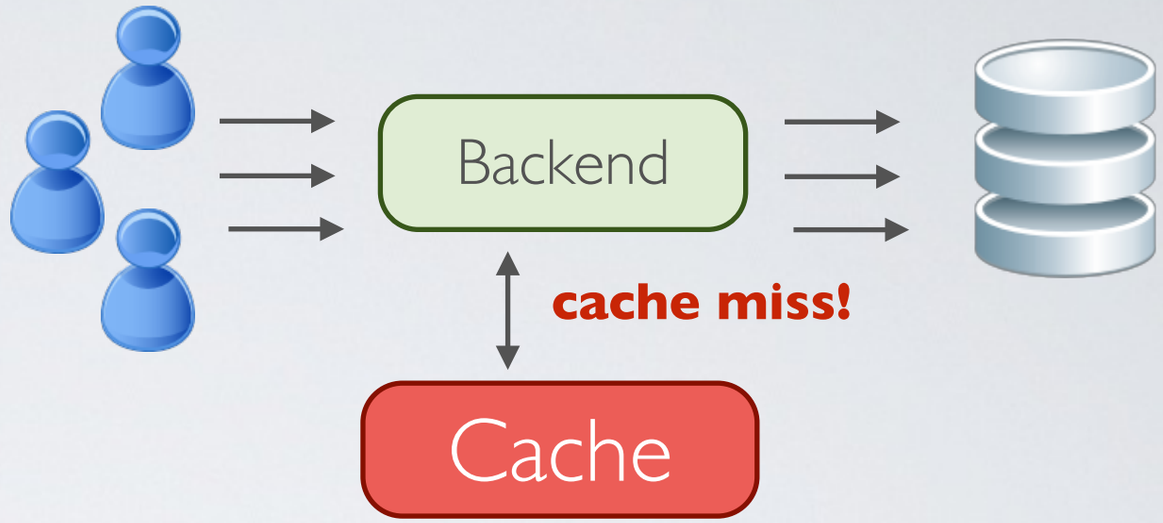
http://memcached.org/

- 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

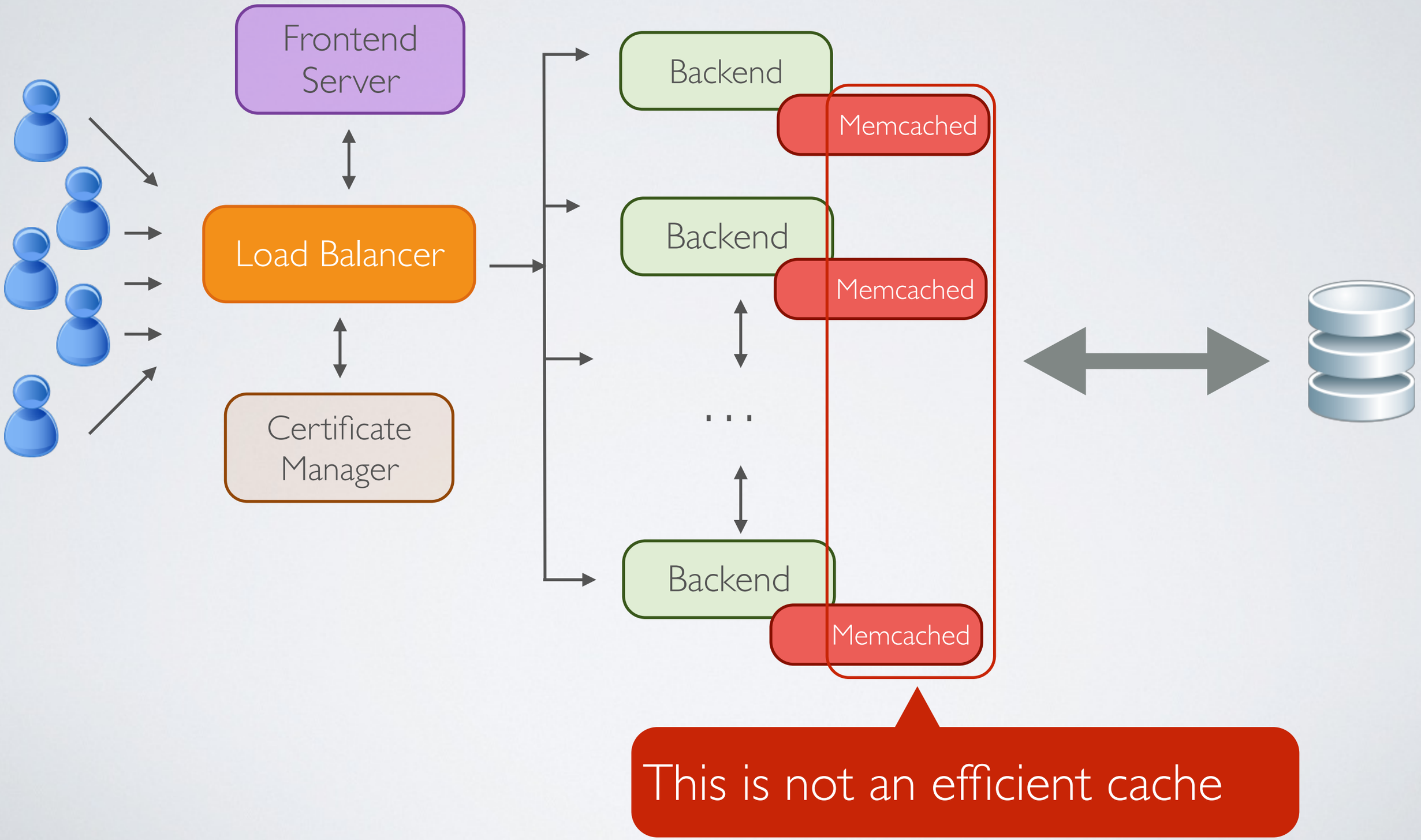
Solution:

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

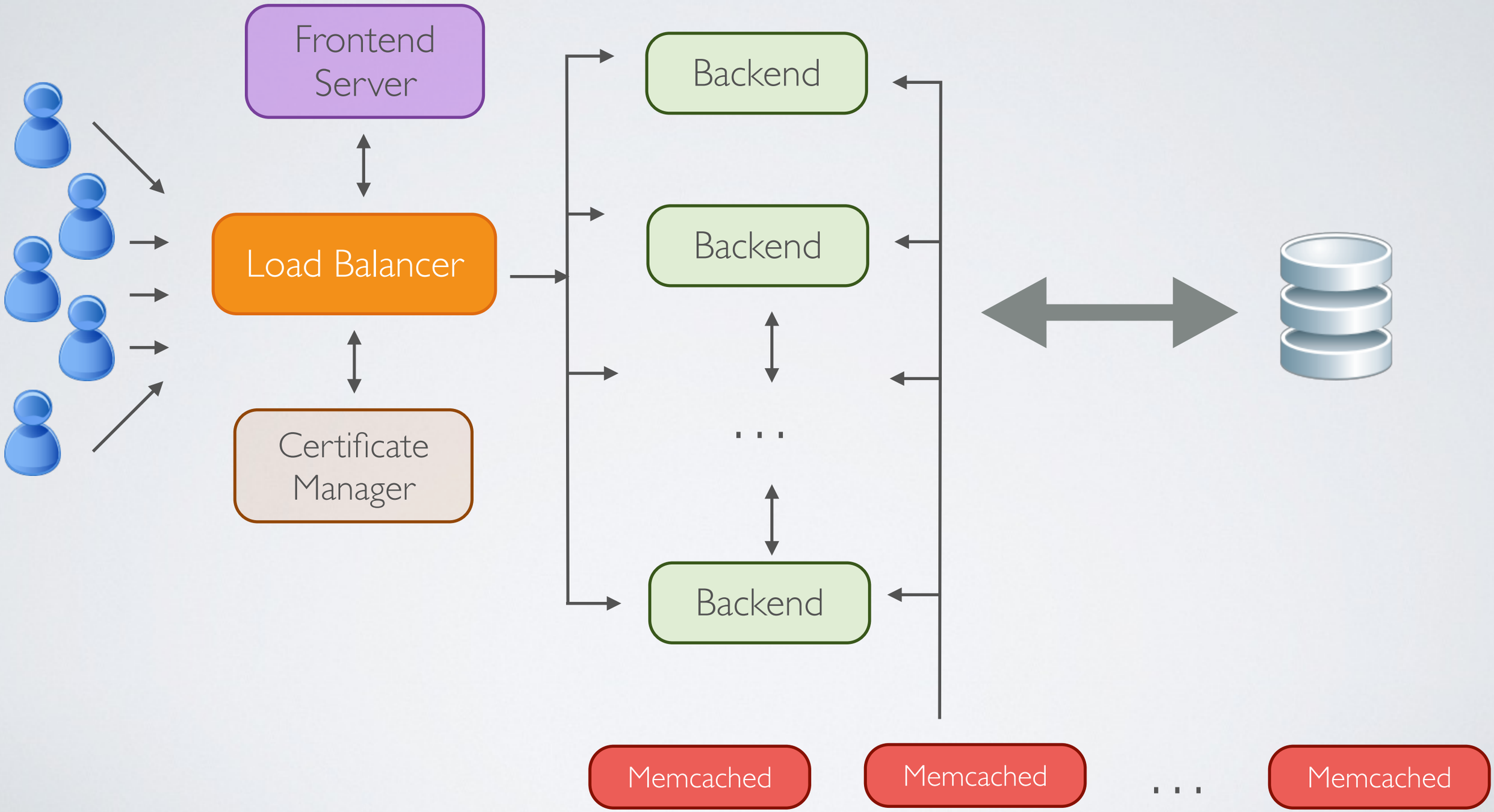
➡ Requires cache warming

Scaling The Backend

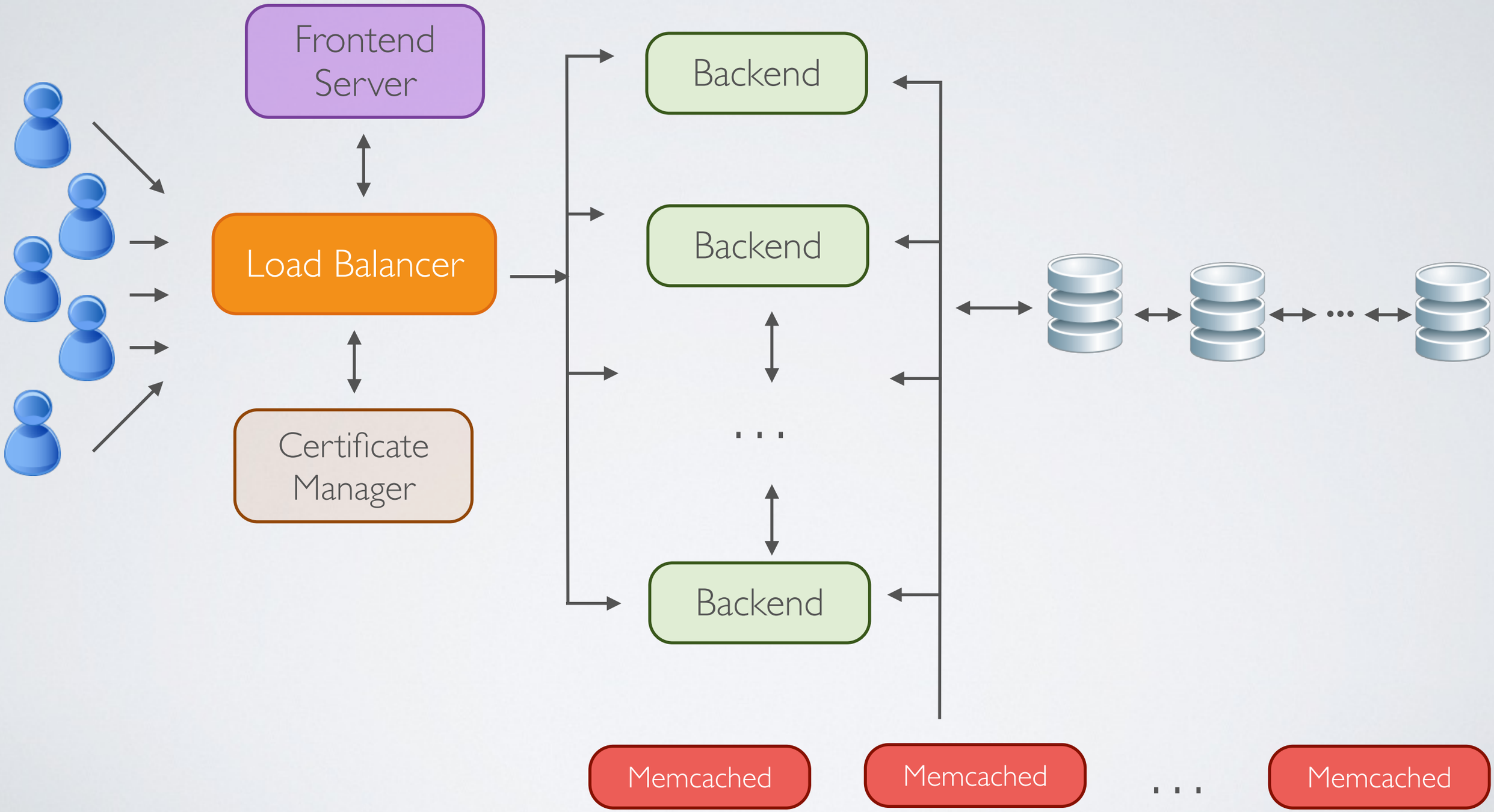
Serving multiple apps with a load balancer



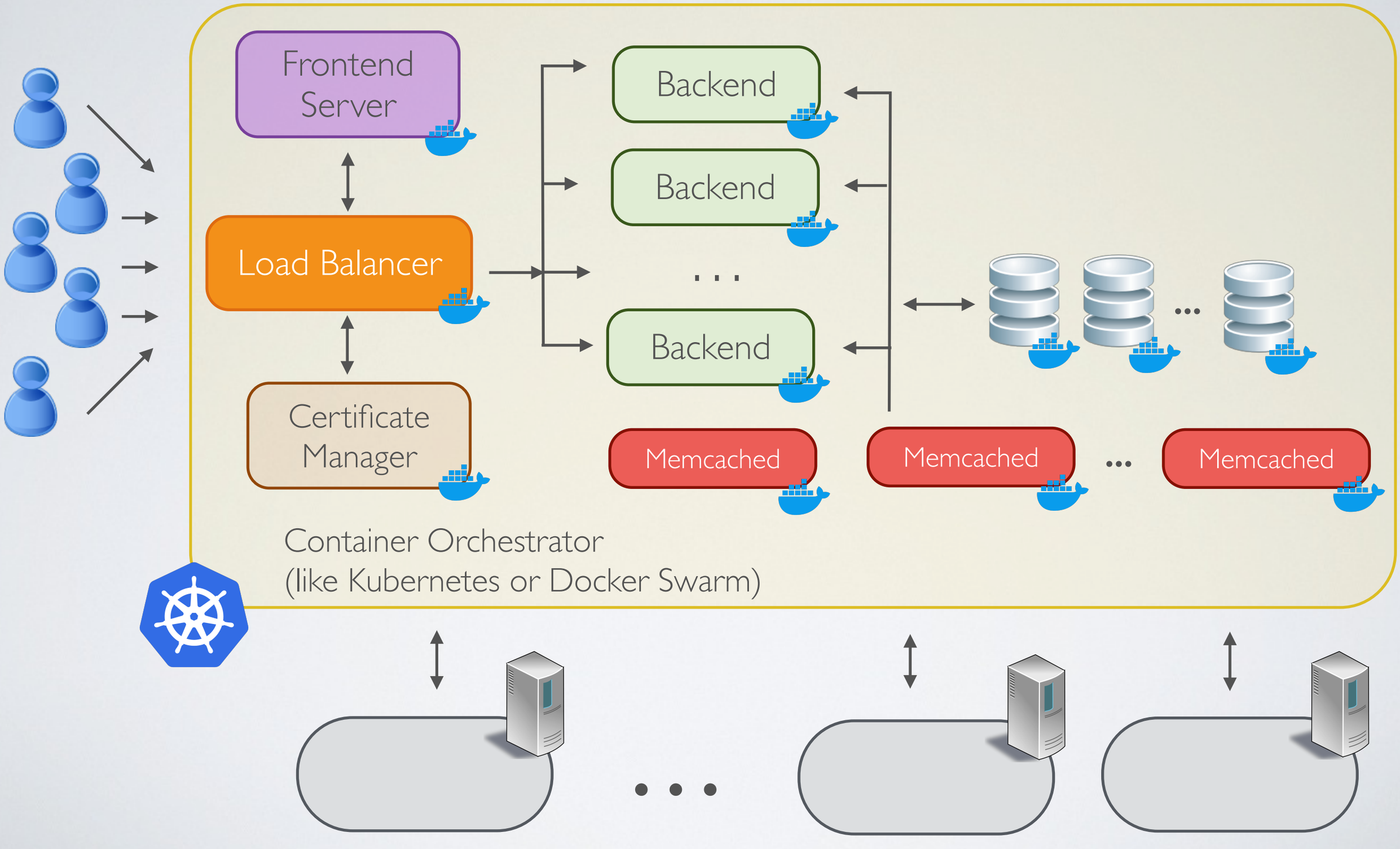
Distributed Shared Cache



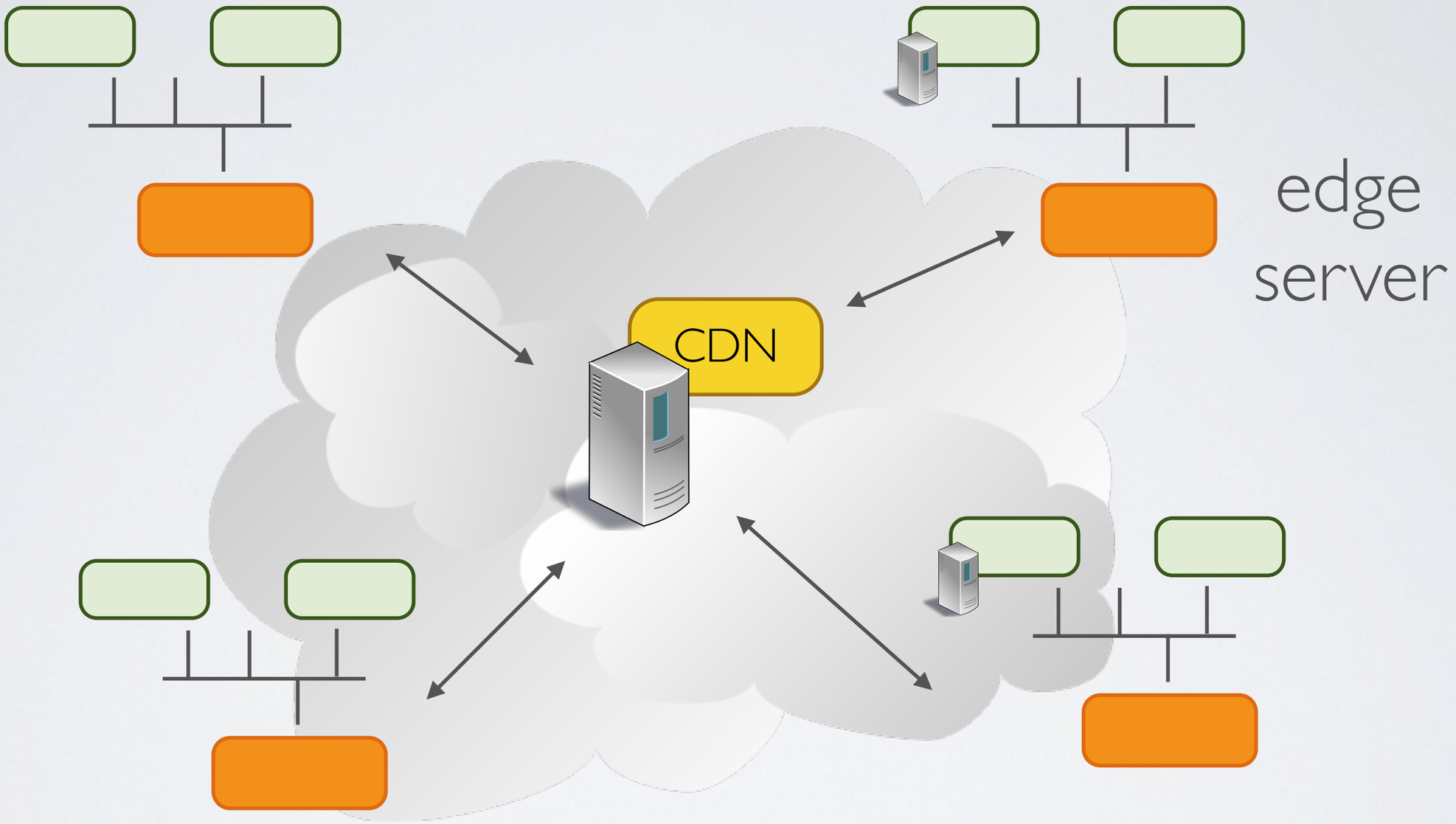
Database Sharding



Automatic Scaling with container Orchestration



CDN : Content Distribution Network

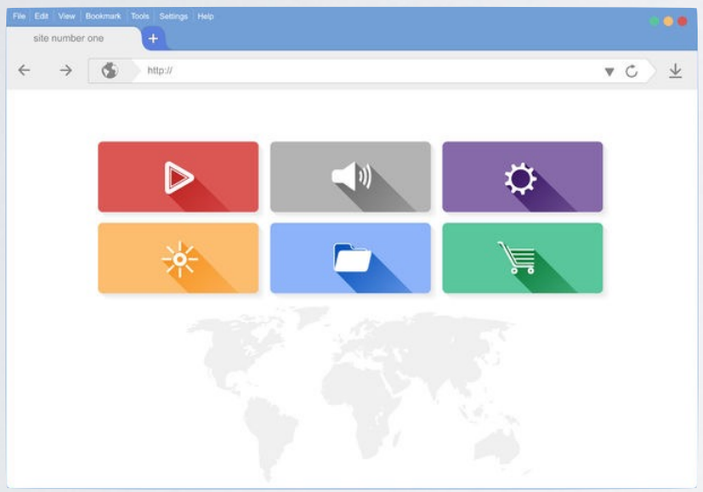


Example : Akamai, Cloudflare

Frontend packing

The problem

Browser



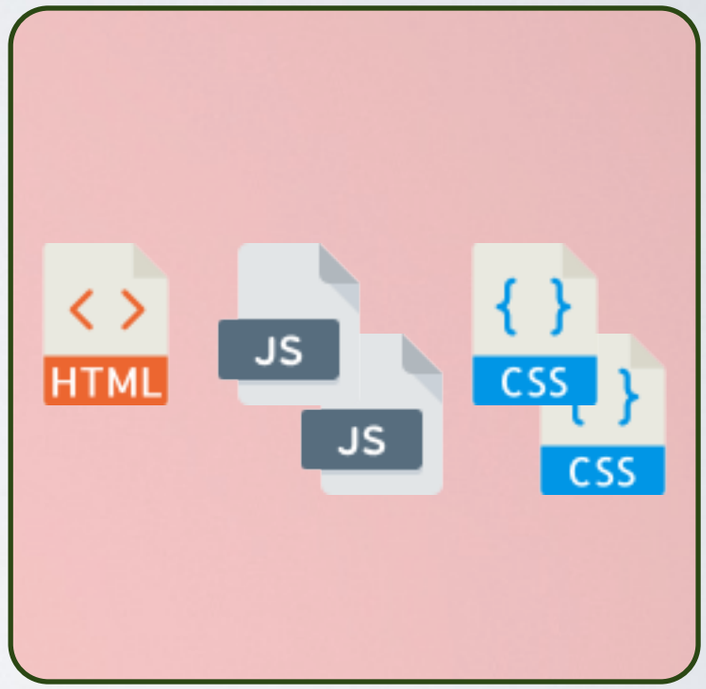
GET /

GET /js/lib.js

GET /js/index.js

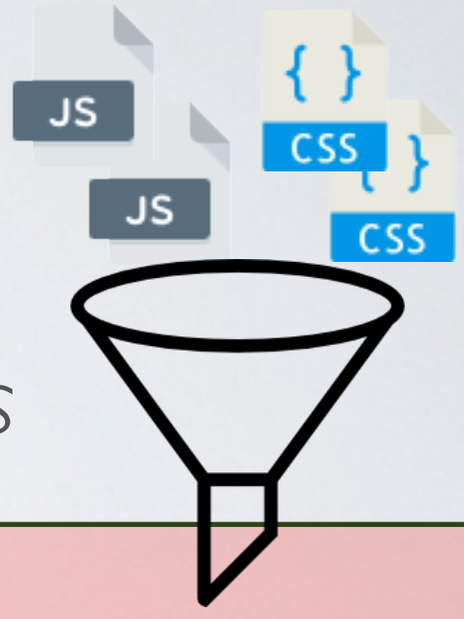
GET /style/index.css

GET /style/generic.css



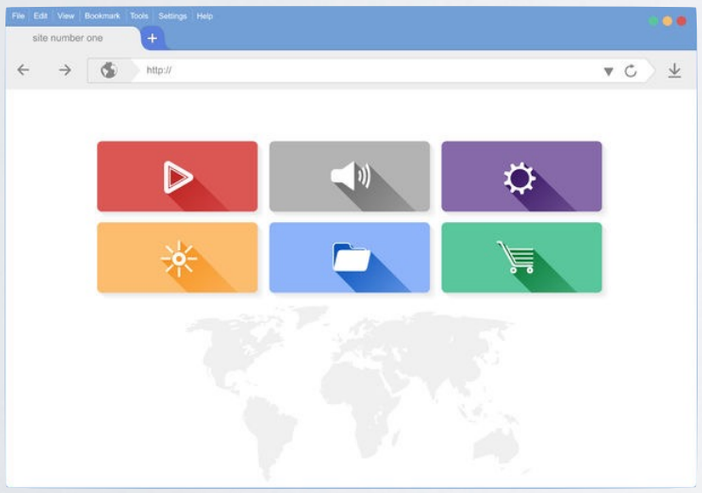
Frontend Server

The solution - using a frontend packer



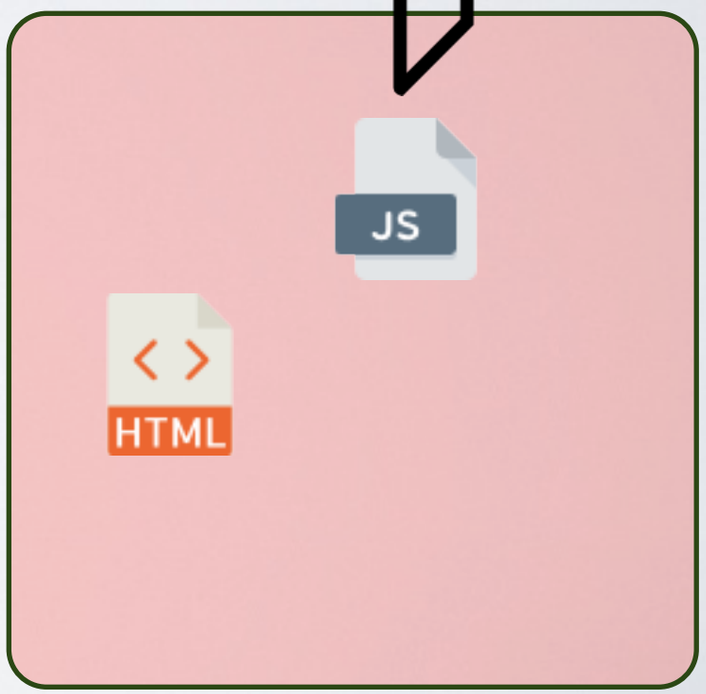
e.g. *webpack.js*

Browser




```
GET /
```

```
GET /js/bundle.js
```



Frontend Server

HTTP/2

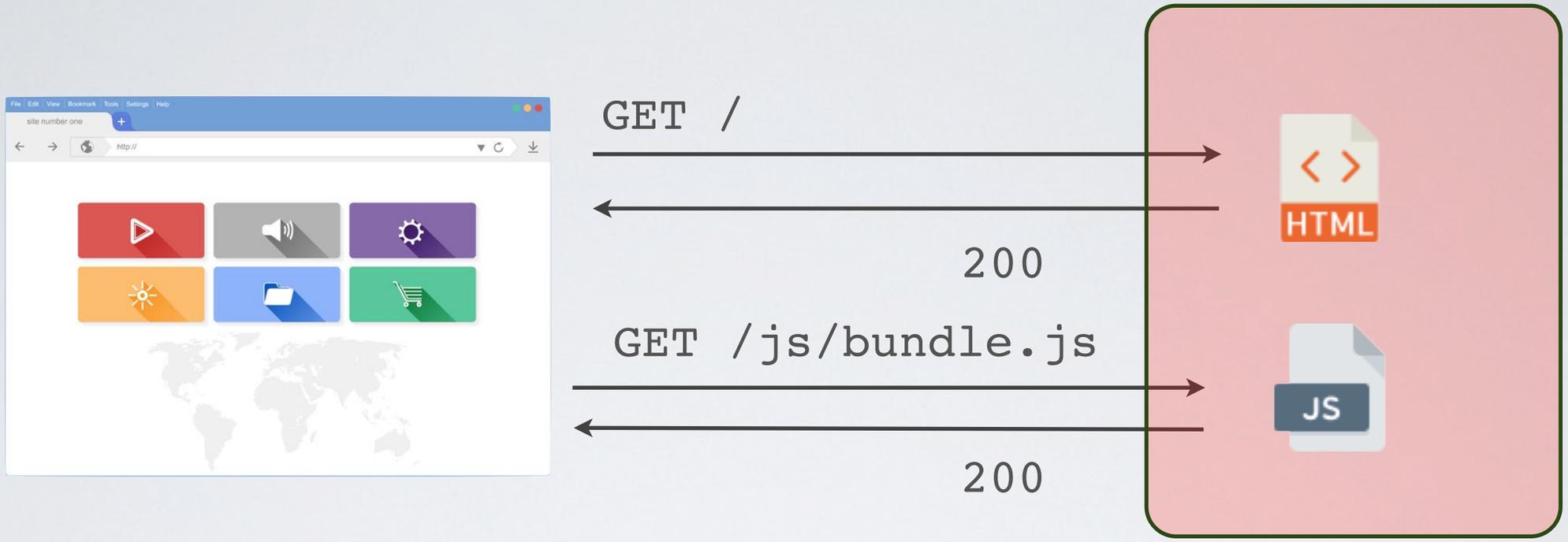
HTTP/2

HTTP/2 enables multiplexing

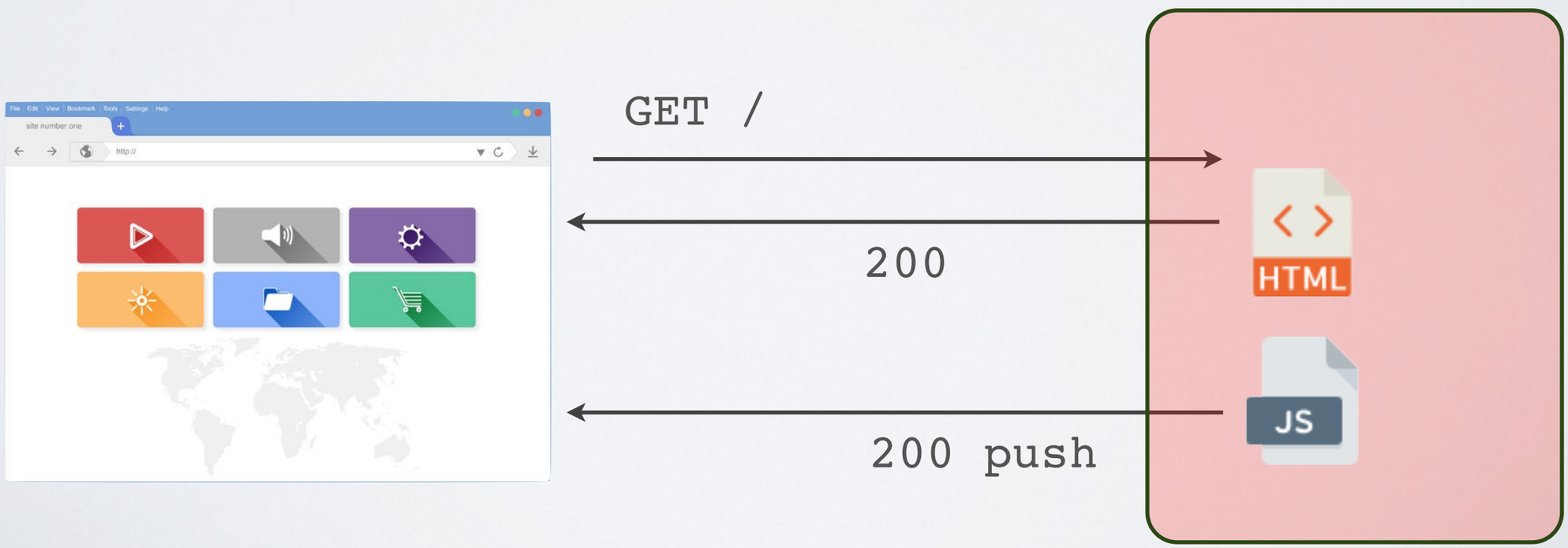
➡ send multiple HTTP responses for a given request (a.ka `push`)

- 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



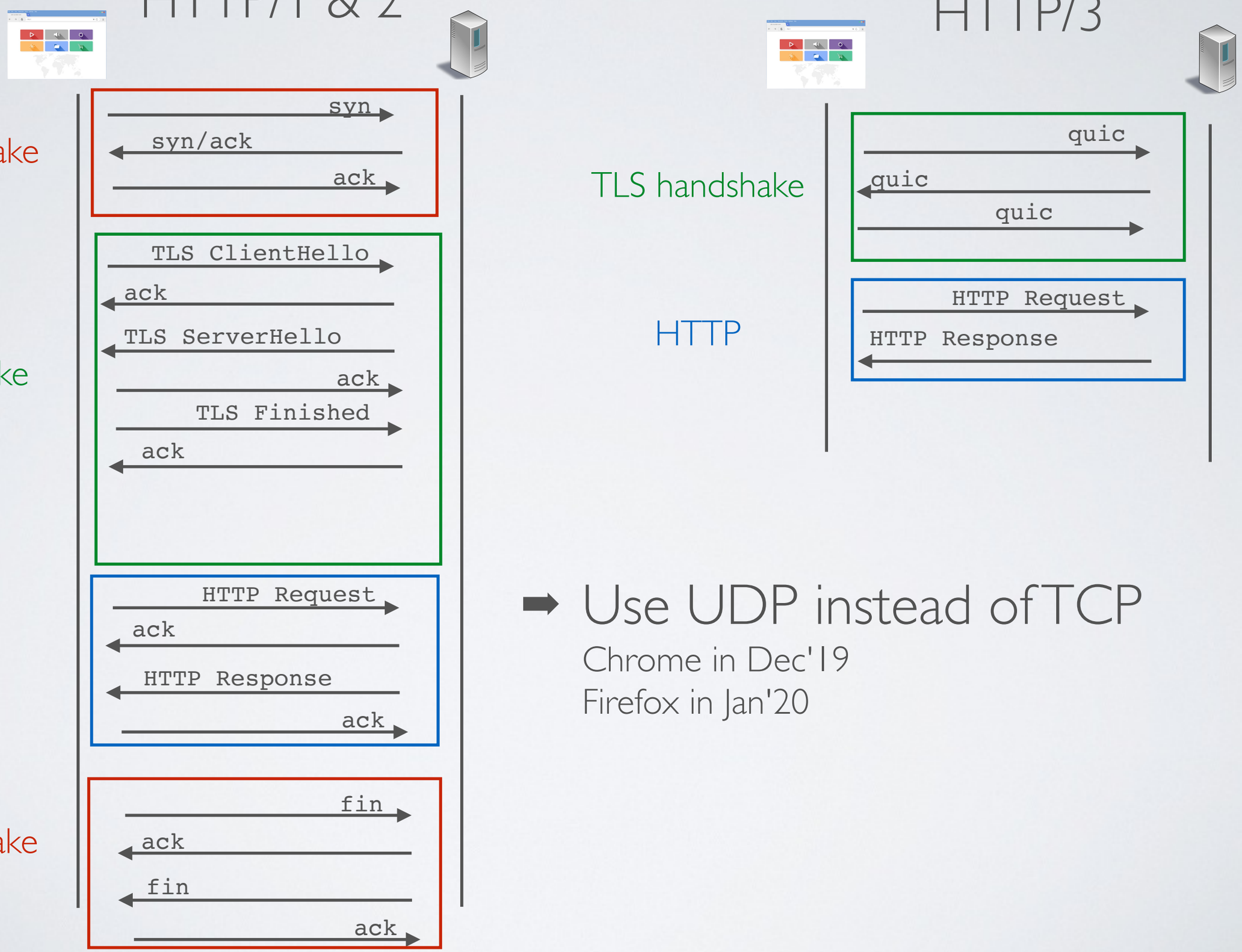
HTTP 2.0



HTTP/3

(work in progress)

HTTP/3 (standard draft)



➡ Use UDP instead of TCP

Chrome in Dec'19

Firefox in Jan'20

(Bonus) Long Polling

Short Polling vs Long Polling

Short Polling

- The frontend request an update from the backend every few seconds
- The backend replies right away regardless if there is an update or not

◉ Many request/responses are wasted

Long Polling

- The frontend request an update from the backend and wait for the response
- The backend replies to the update request only when there is an update

✓ No request/response wasted

✓ Updates are processed as soon as they arrived

Long Polling