Advanced Programming 2 - Scalability

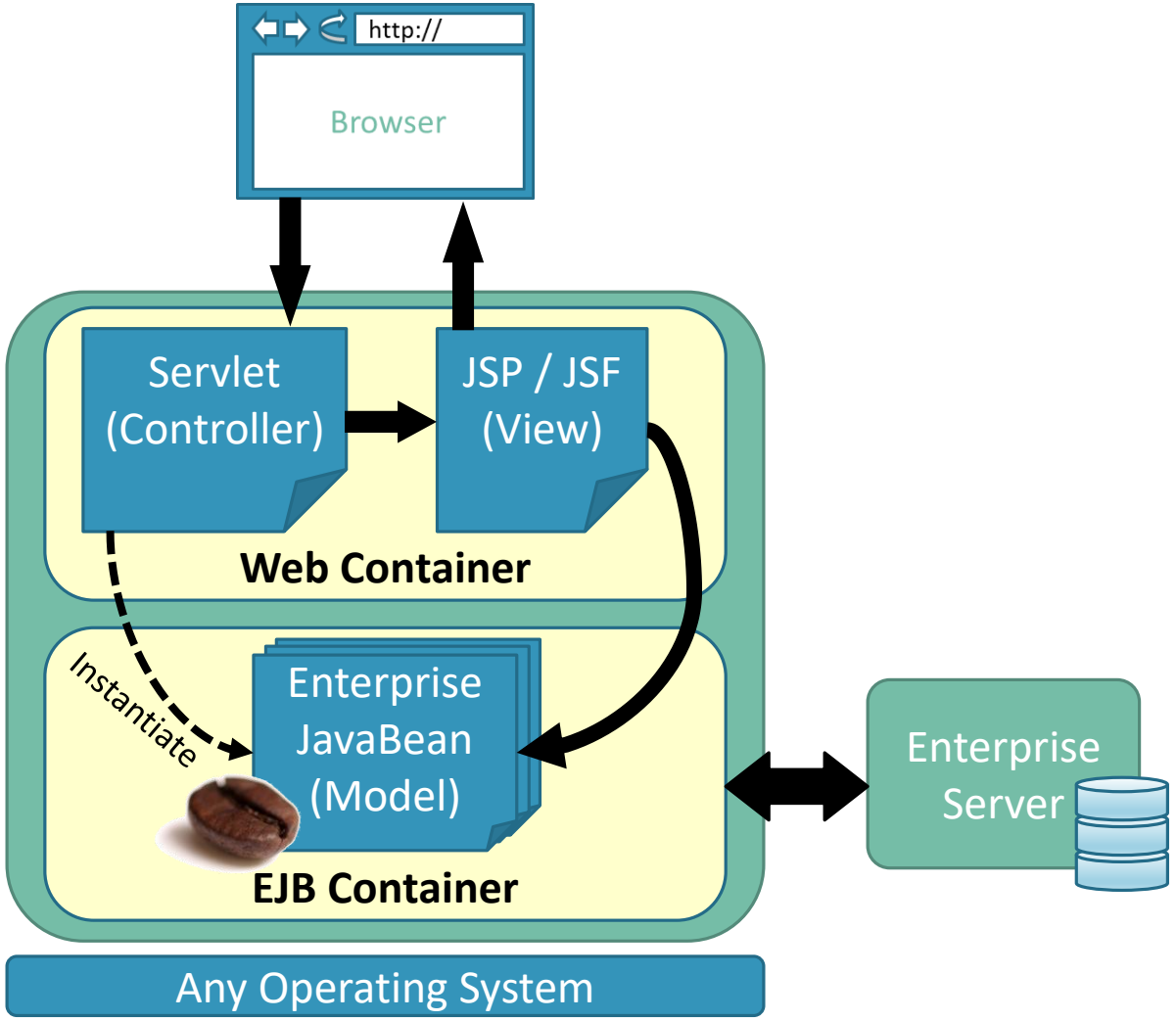
DR. ELIAHU KHALASTCHI

2016

In the last lessons…

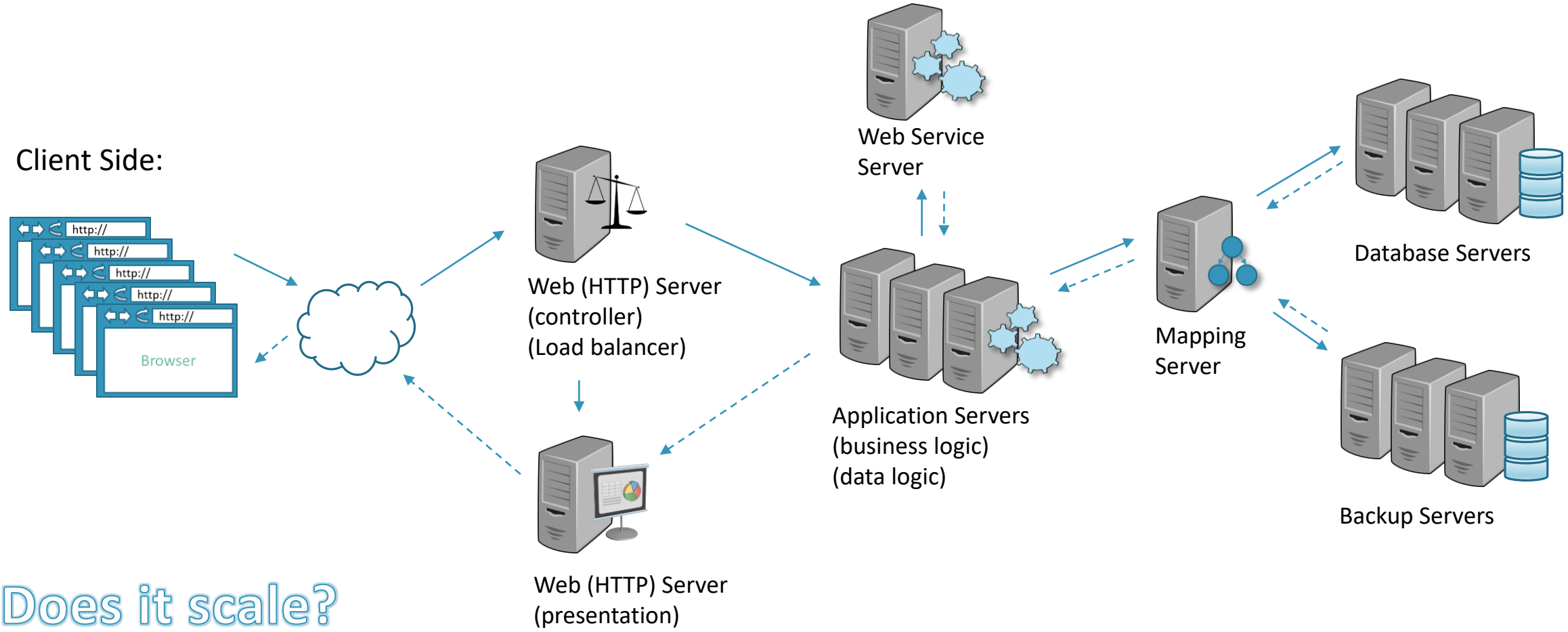
Client Side:
(client tier)

Java2 EE
Application Server
(middle tier)



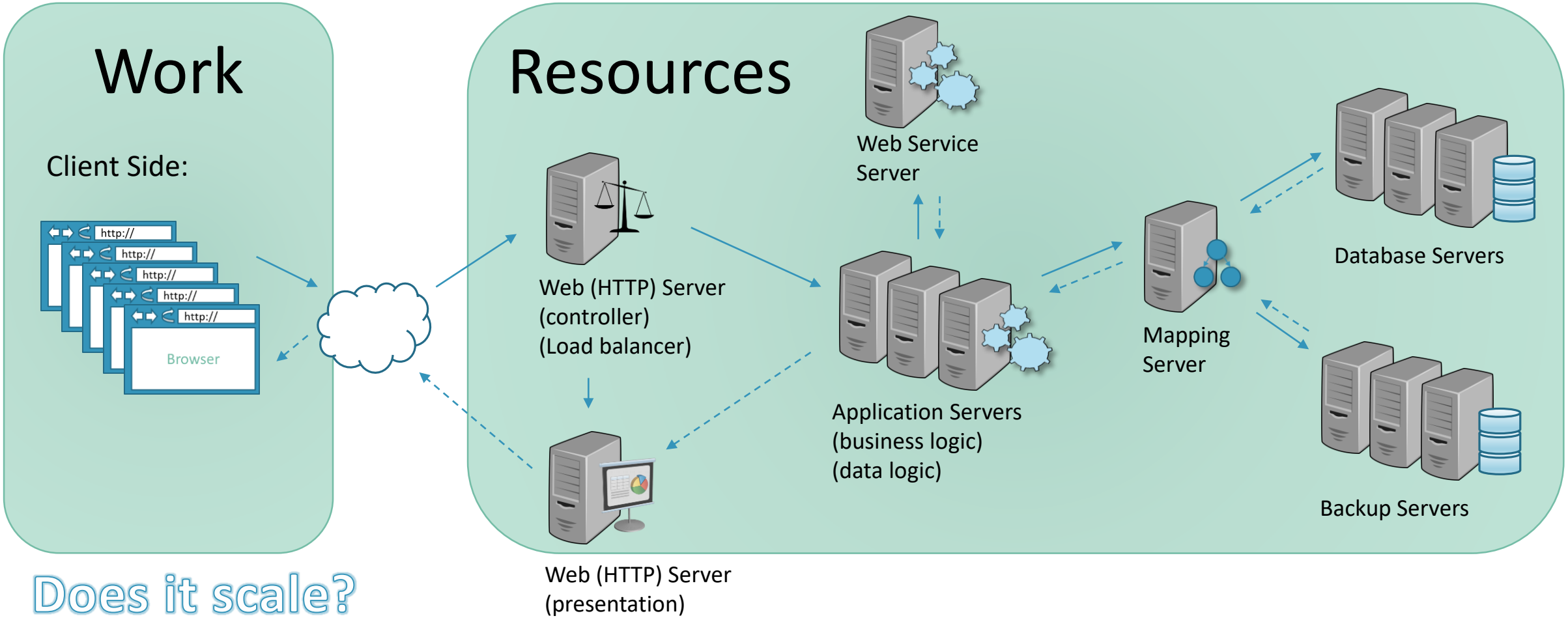
Enterprise
Information
System (EIS)
tier

In reality, it is more complex…



Does it scale?

In reality, it is more complex…

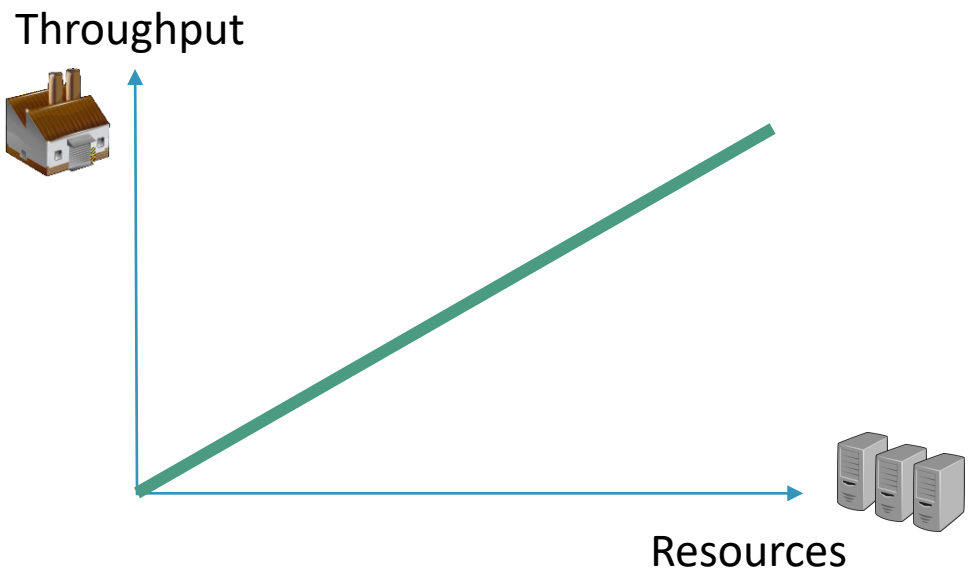


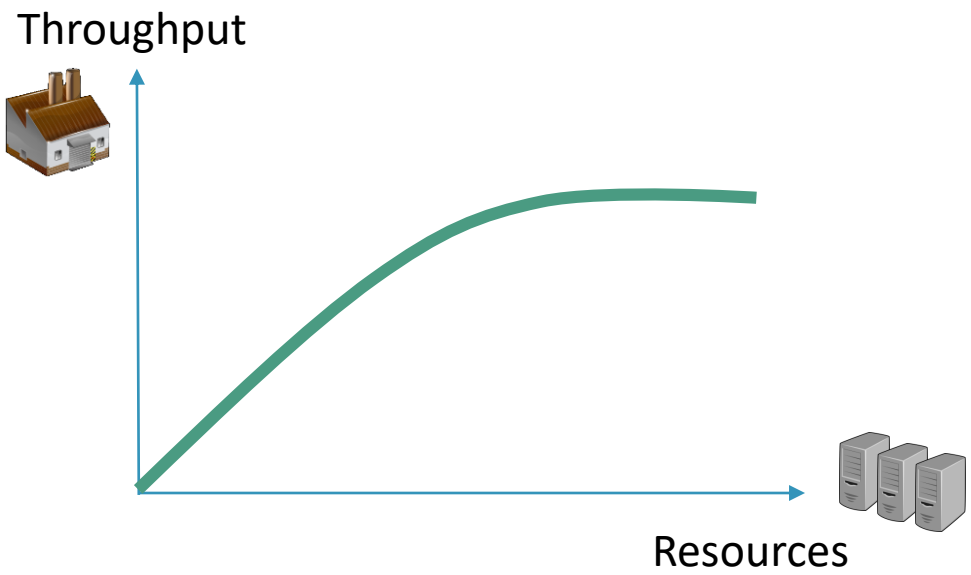
Does it scale?

Definitions

- Scalability
  - The capability of a system to handle a growing amount of work,
  - or its potential to be enlarged in order to accommodate that growth

- Linear Scalability vs. None-Linear Scalability





Definitions (in the Web App context)

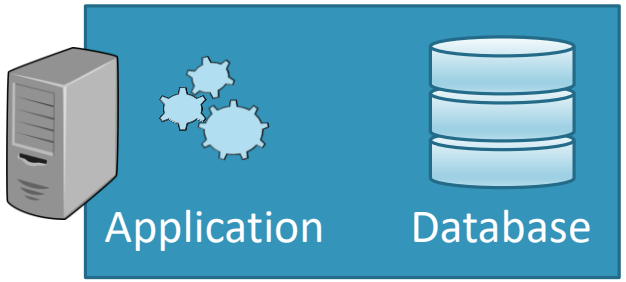
- **Scalability** – the required effort to increase the capacity to handle greater amounts of load
  - How much additional traffic can the system handle?
  - How easy it is to add more storage?
  - How many more transactions can be handled?

- **Performance** – optimized utilization of resources for fast response and low latancy

- **Availability** – the uptime of the application
  - Redundancy for key components
  - Rapid recovery for partial system failure
  - Graceful degradation when a problem occurs

- Other parameters – relatability, manageability, cost, etc.

Let’s start small… one server



A single machine cannot be tuned differently

Application server needs

- More CPU
- Optimized TCP/IP connections
- None blocking I/O, etc.

Database Server needs more RAM

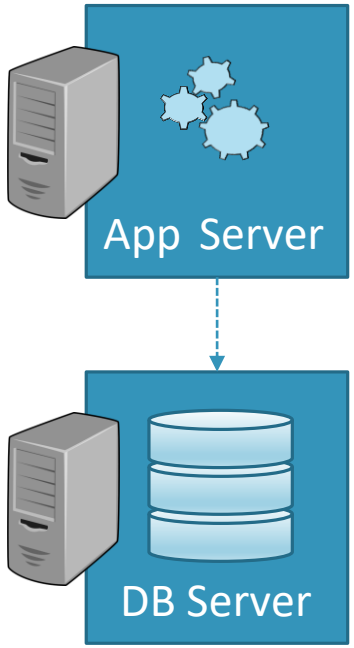
- And different optimizations

After some time… (more users)

Scale-up:

- Put more CPU
- Put more RAM

A Better Idea: s[lit the nodes



We can optimized each node differently

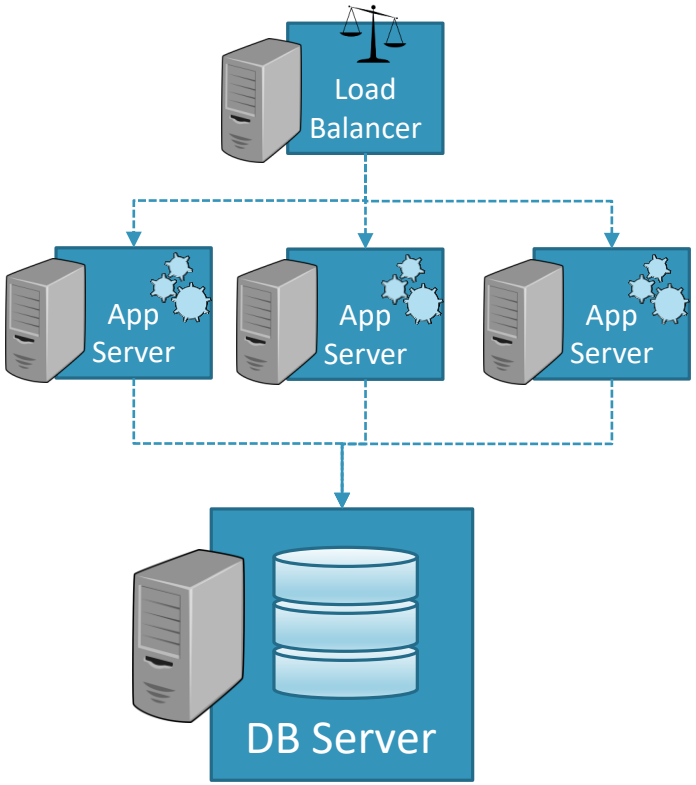
Downsides:

- The availability is not increased
- If one node fails the entire system fails

- Scalability is not increased
- More users →
  - more CPU for the App Server
  - more RAM for the DB Server
- Scale-up…
  - They are still bottlenecks…

We need to scale-out, use more nodes

A Better Idea: Scale-out



The load balancer takes requests

and forward them to the app servers

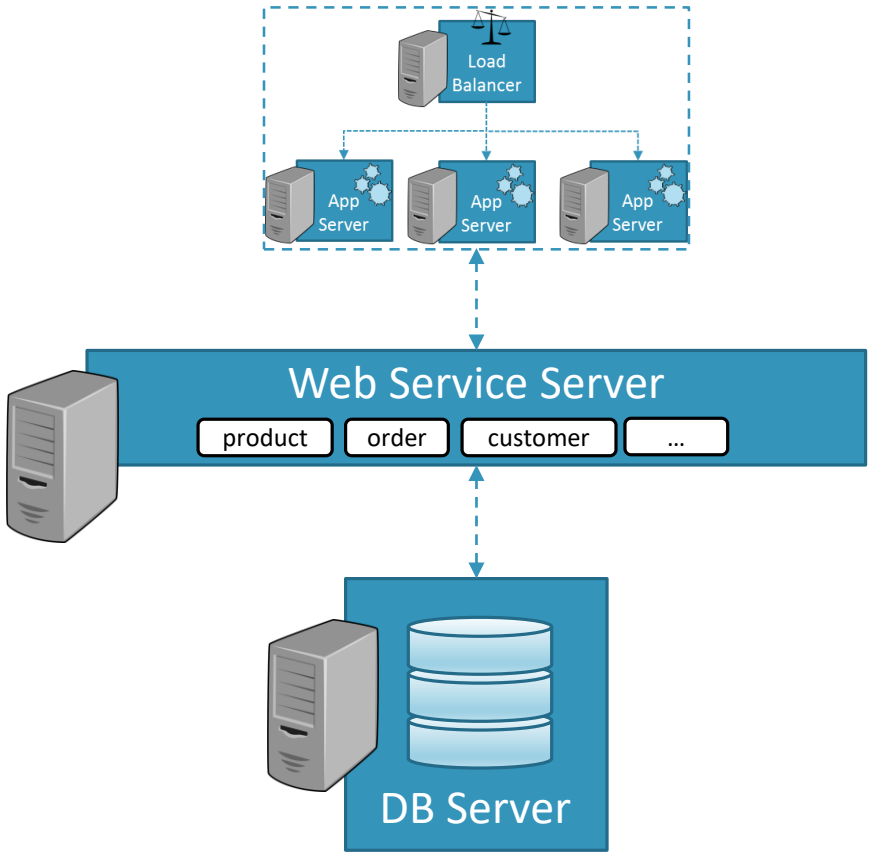
while balancing the load between them

This is scalable,
we can always add more app servers

The load balancer = a single point of failure

The DB Server = a single point of failure

Using Services



Services can be independent

Dedicated teams can work on them in parallel

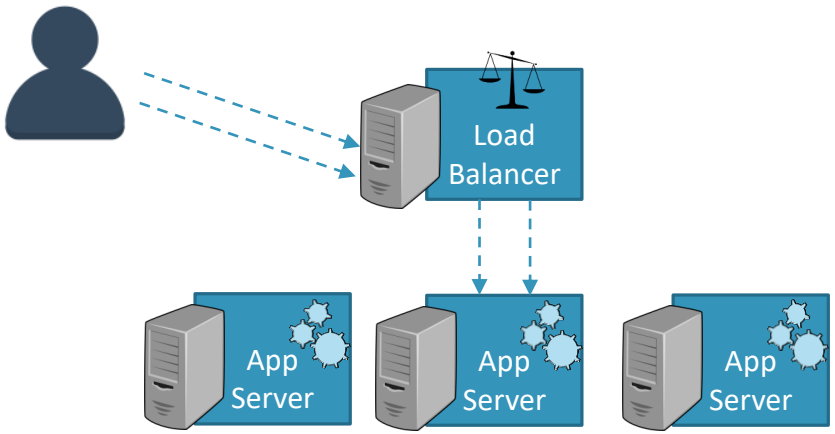
Using more services doesn't increase availability

- Unless we use redundancy…

Performance issue

- The data needs to be cross-service-join

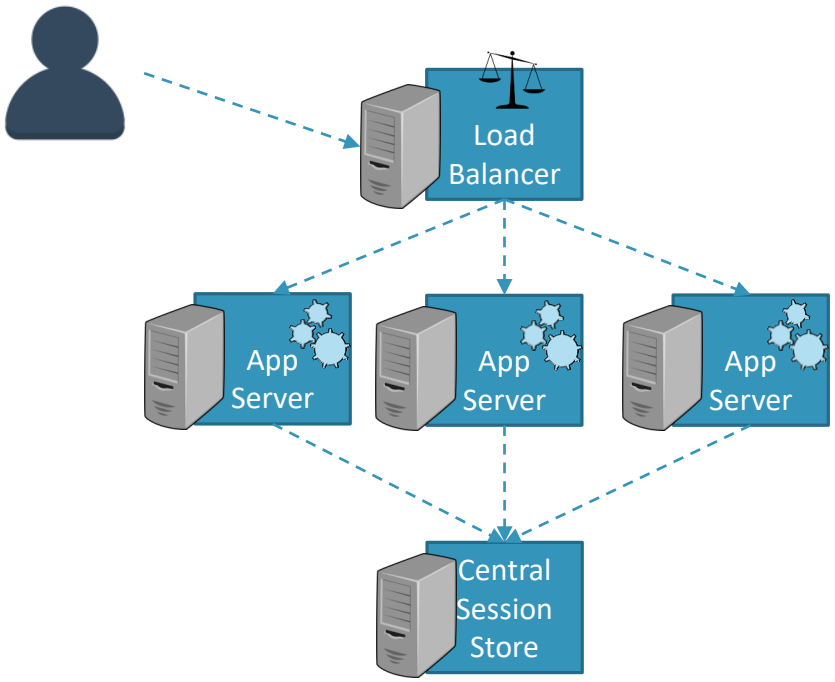
Handling a sticky session…



A sticky session

- A number of related stateful requests
- needs to be handled by the same machine
- Does not scale

Handling a sticky session…



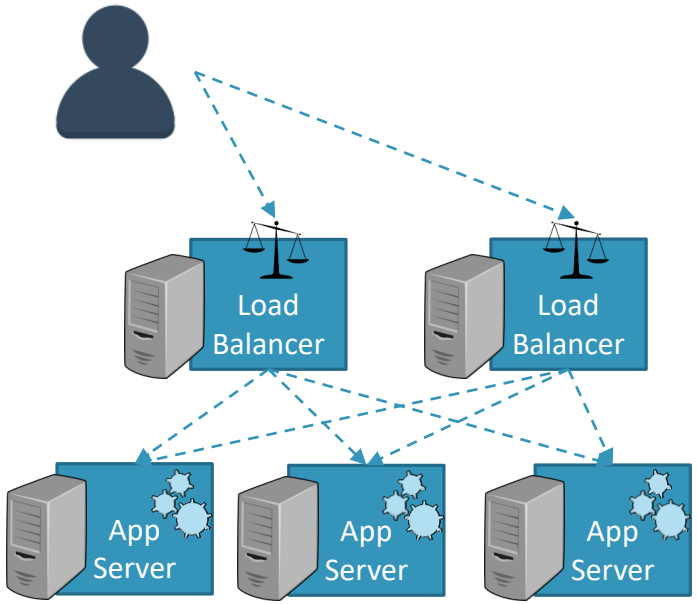
A sticky session

- A number of related stateful requests
- needs to be handled by the same machine
- Does not scale

We can use another server to store the session state

- A database, mem-cache
- Performance issue – another call is made

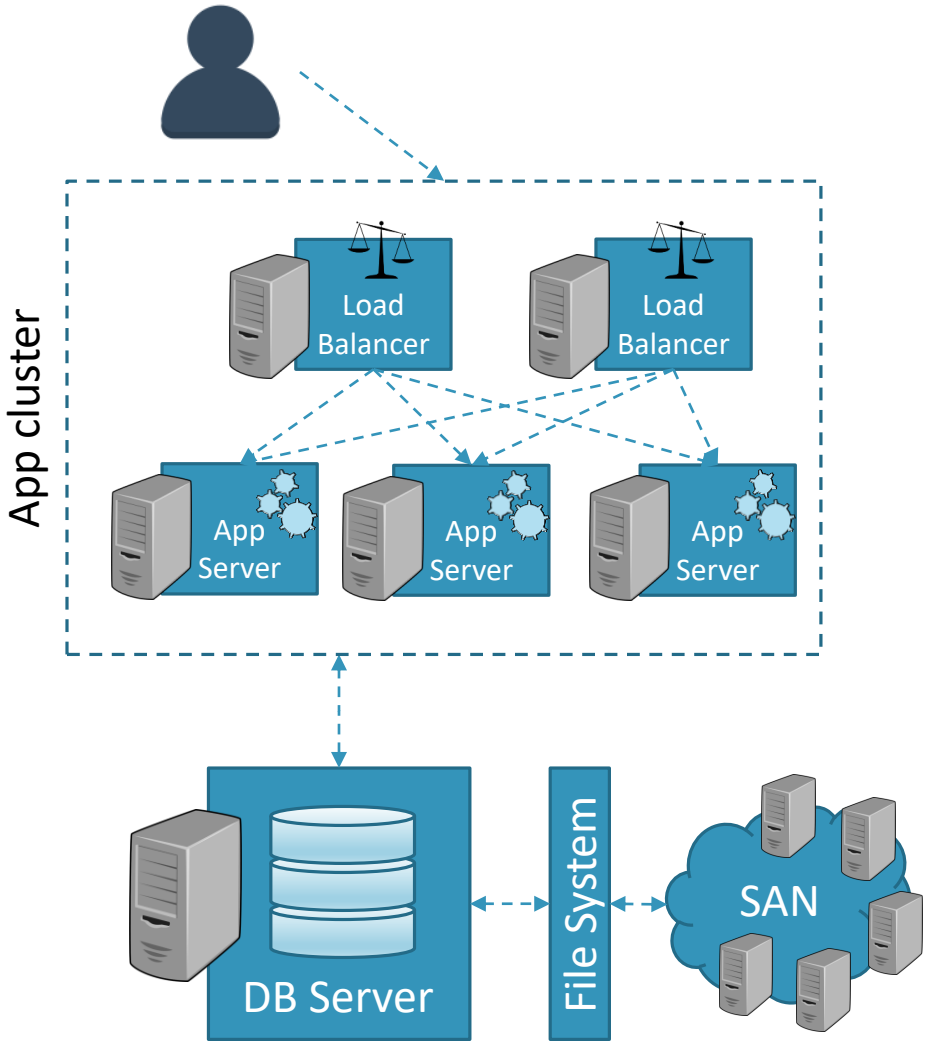
Load Balancer Redundancy



Can be used as a backup

Or be used together

What about the database?



The database can use a SAN

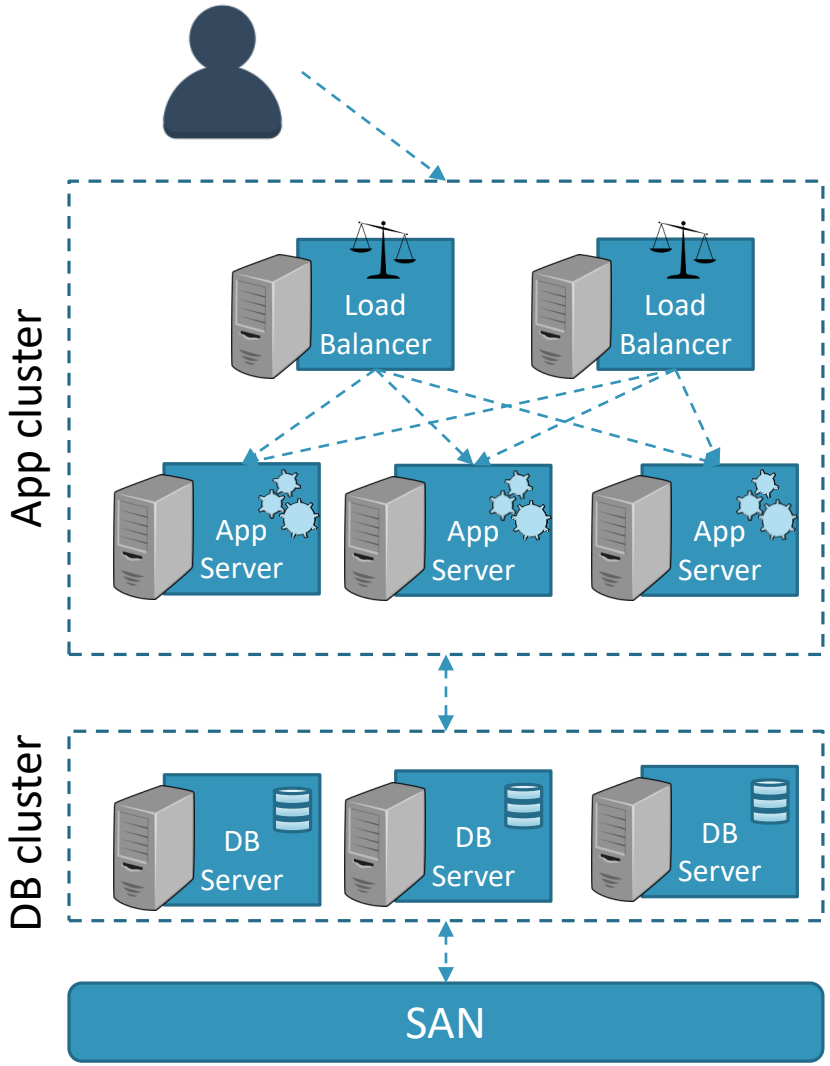
SAN – Storage Area Network

A SAN can be **scaled up**

- More RAM, discs in the network

But still the DB is a single point of failure

Scaling out the database



No single point of failure to the DB

Can retrieve data from either DB server

If we use replicas… (8, 16)

- Wait for the backup – consistent, performance issue
- Don't wait – nonconsistency, better performance

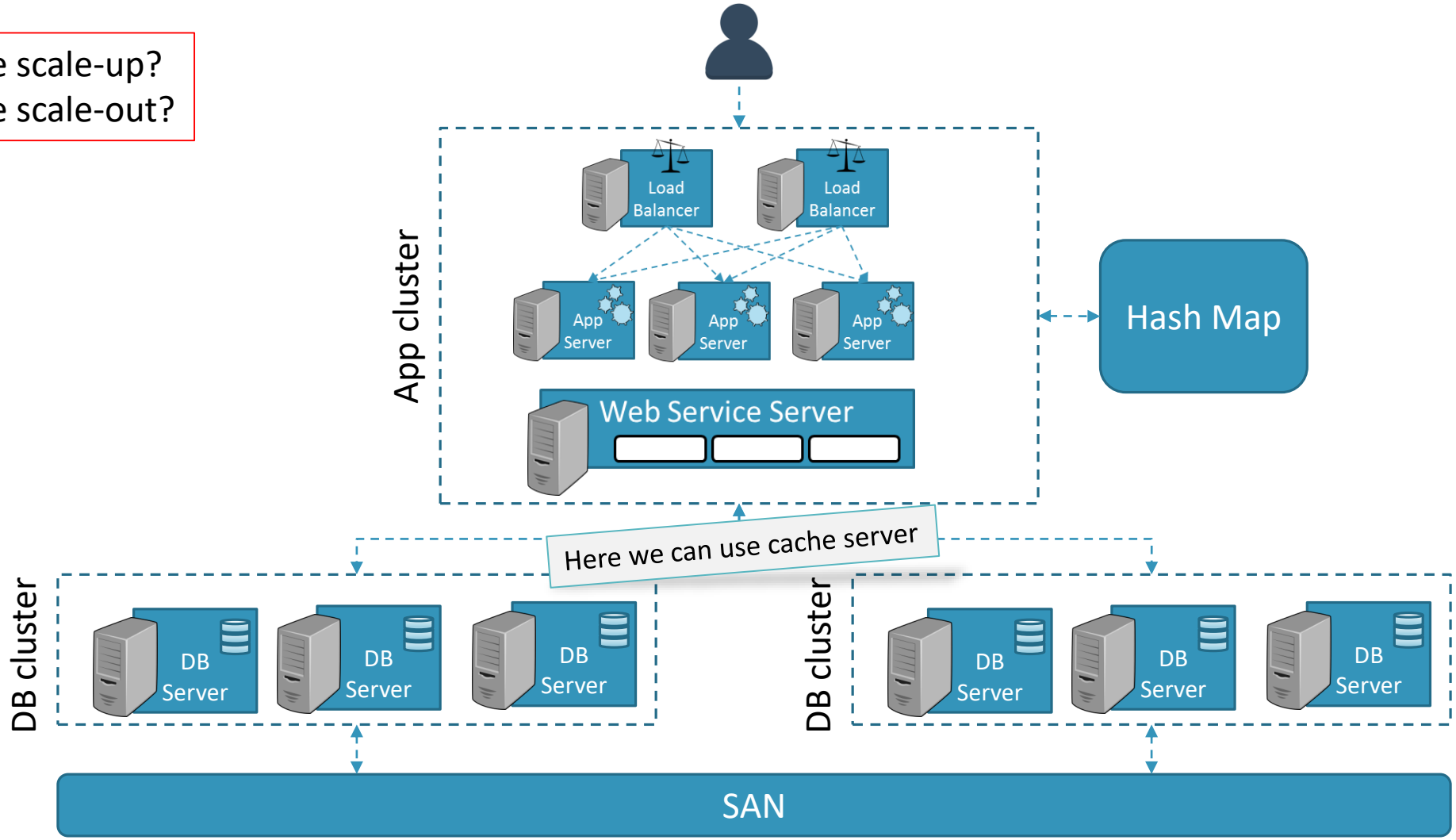
Scaling out a DB is difficult (horizontal partitioning)

- We loose the advantage of ACID transactions
- Atomicity, Consistency, Isolation, Durability

We can set different responsibilities for different DBs (vertical partitioning)

Overview

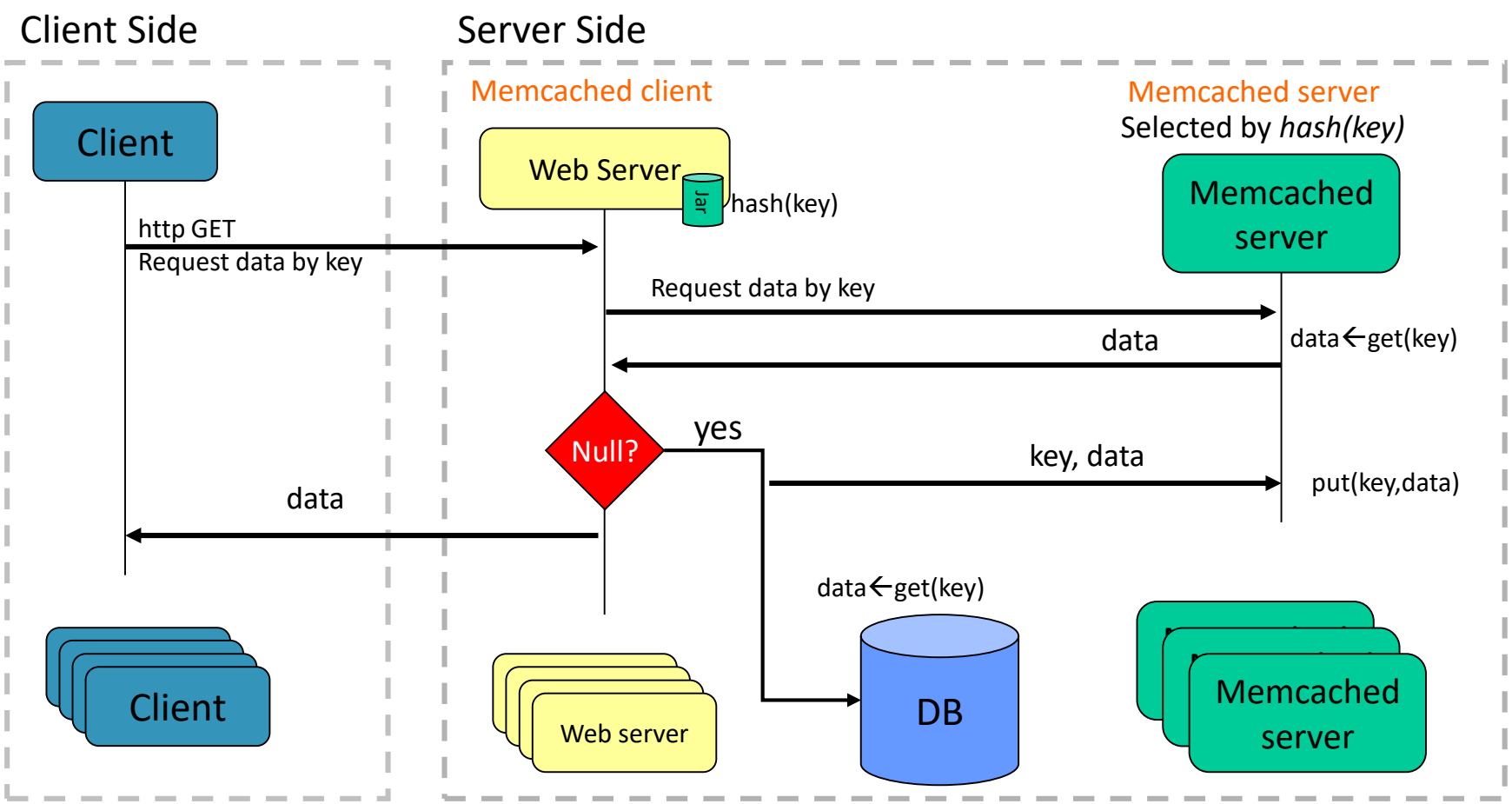
What can we scale-up?
What can we scale-out?



Memcached

A CASE STUDY

Memcached Overview



Pseudo code example

```
function get_foo(int userid) {
   data = db_select("SELECT * FROM users WHERE userid = ?", userid);
   return data;
}
```

```
function get_foo(int userid) {
   /* first try the cache */
   data = memcached_fetch("userrow:" + userid);
   if (!data) {
      /* not found : request database */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* then store in cache until next get */
      memcached_add("userrow:" + userid, data);
   }
   return data;
}
```

```
function update_foo(int userid, string dbUpdateString) {
  /* first update database */
   result = db_execute(dbUpdateString);
   if (result) {
      /* database update successful : fetch data to be stored in cache */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* the previous line could also look like data = createDataFromDBString(dbUpdateString); */
      /* then store in cache until next get */
      memcached_set("userrow:" + userid, data);
   }
}
```

Memcached scalability

- The scalability is **linear**
- As the number of keys increases
- We can add a linear number of memcahed server

- Data is retrieved from the RAM
  - Faster than from the DB
  - Less load on the DB

If a Memcached Server crashes…

- One less server keys can be referred to…
- The hash map size is reduced from n to $n-1$

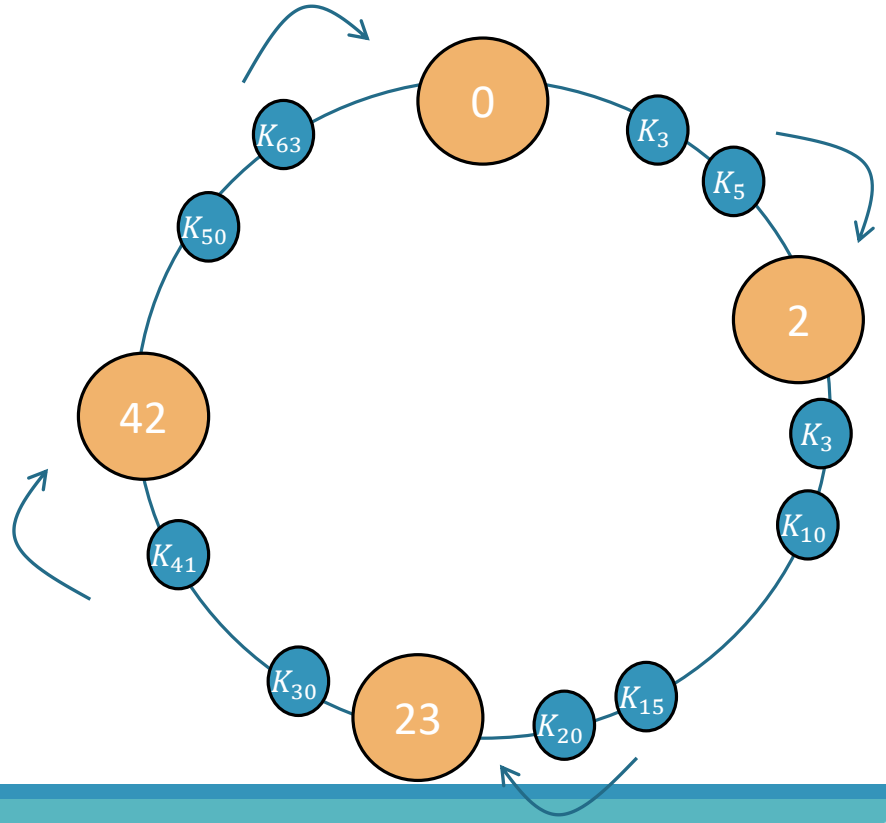
- Most mapping algorithms require the **remapping of all keys**
  - E.g., put object o in machine number $hash(o)\ mod\ n$

- This is a catastrophe…
  - All the data from the DB needs to be reloaded to the memcahed servers
- The same problem exists when adding a server… (scale-out)

- Unless we use **consistent hashing**

Consistent Hashing

- If a node is removed (or added) – only K/n keys need to be remapped
  - Where K is the number of keys, and n is the number of slots
- Main idea: keys and nodes are mapped to the same (ring-like) ID space



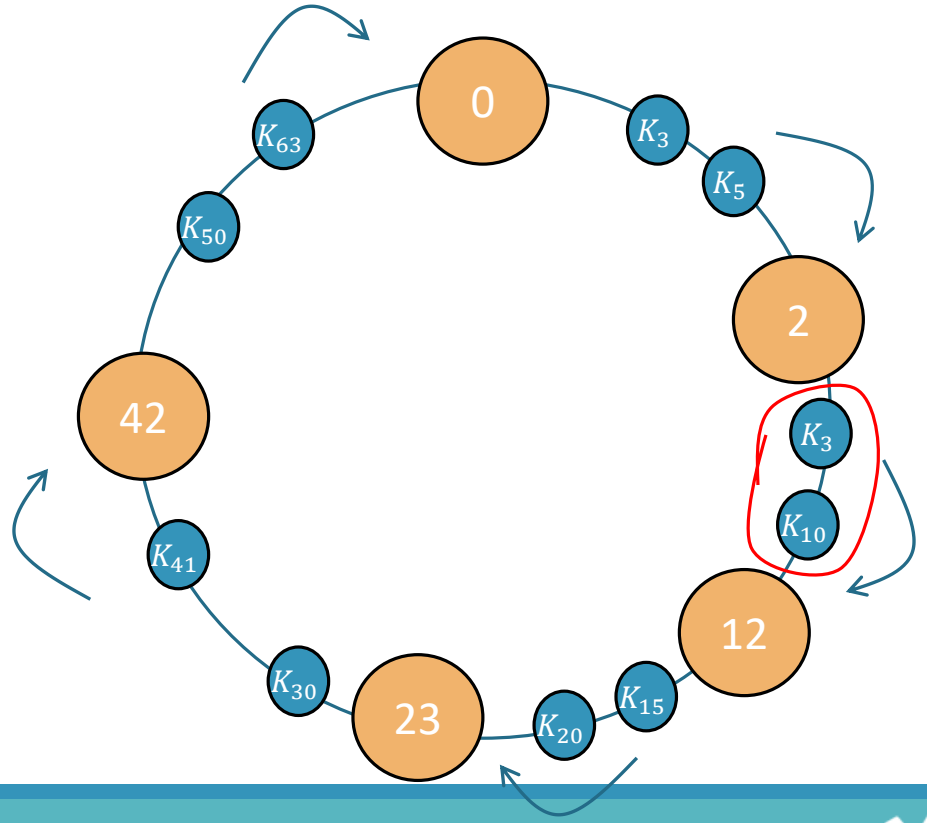
using $hash(IP)$ a node is mapped to a place on the ring – its ID

$hash(Key)$ also points to a place on the ring – the key ID

We need to map a key ID to a node ID
- We go clockwise and assign the key to the next node

Consistent Hashing

- If a node is removed (or added) – only K/n keys need to be remapped
  - Where K is the number of keys, and n is the number of slots
- Main idea: keys and nodes are mapped to the same (ring-like) ID space



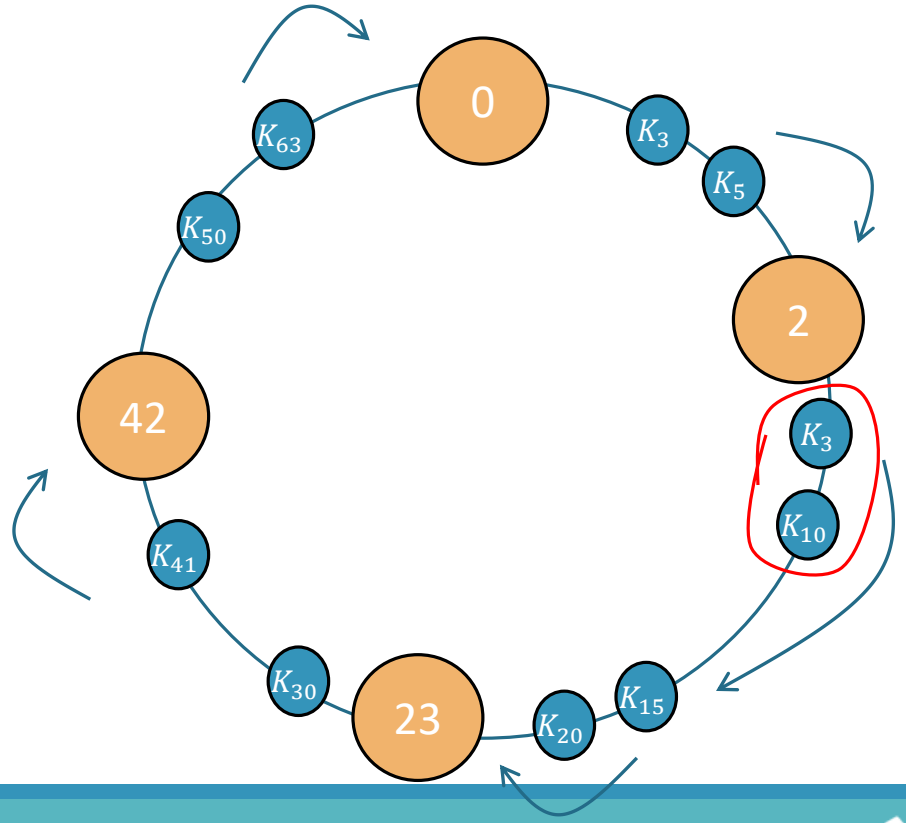
using $hash(IP)$ a node is mapped to a place on the ring – its ID

$hash(Key)$ also points to a place on the ring – the key ID

We need to map a key ID to a node ID
- We go clockwise and assign the key to the next node

Consistent Hashing

- If a node is removed (or added) – only K/n keys need to be remapped
  - Where K is the number of keys, and n is the number of slots
- Main idea: keys and nodes are mapped to the same (ring-like) ID space



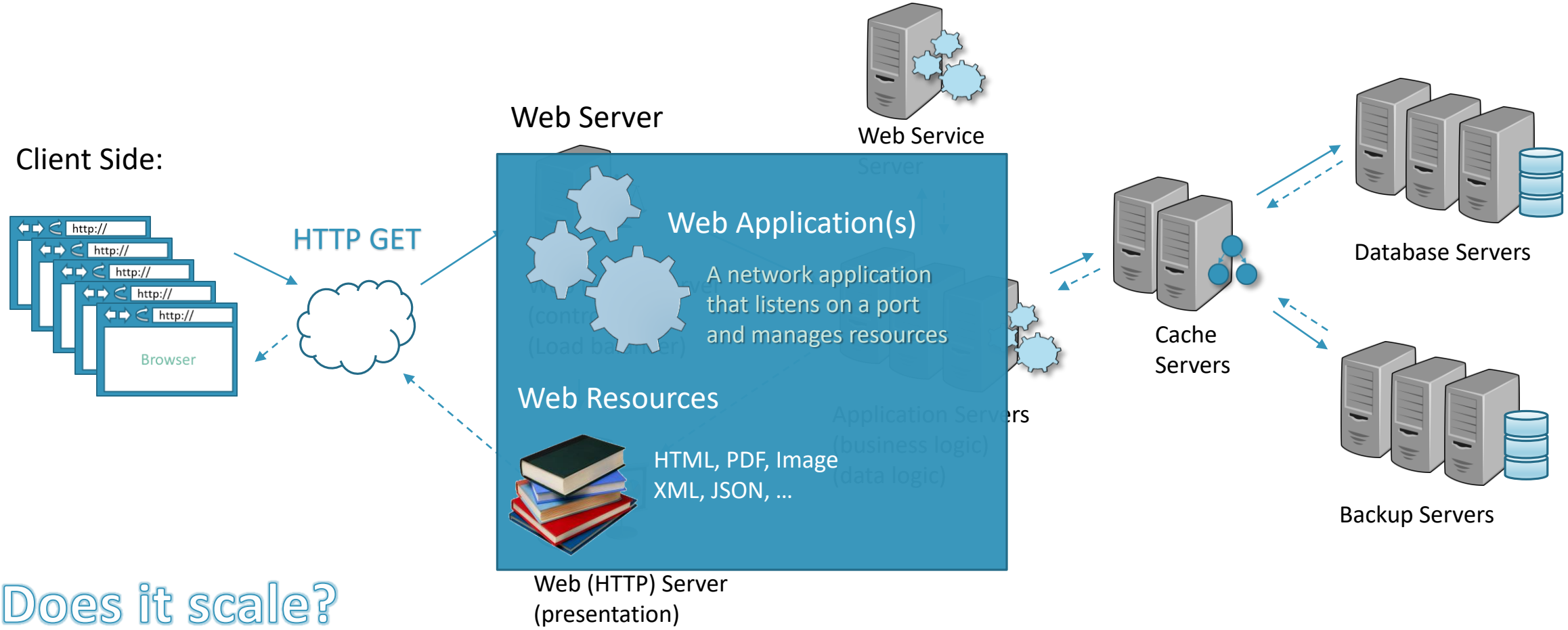
using $hash(IP)$ a node is mapped to a place on the ring – its ID

$hash(Key)$ also points to a place on the ring – the key ID

We need to map a key ID to a node ID
- We go clockwise and assign the key to the next node

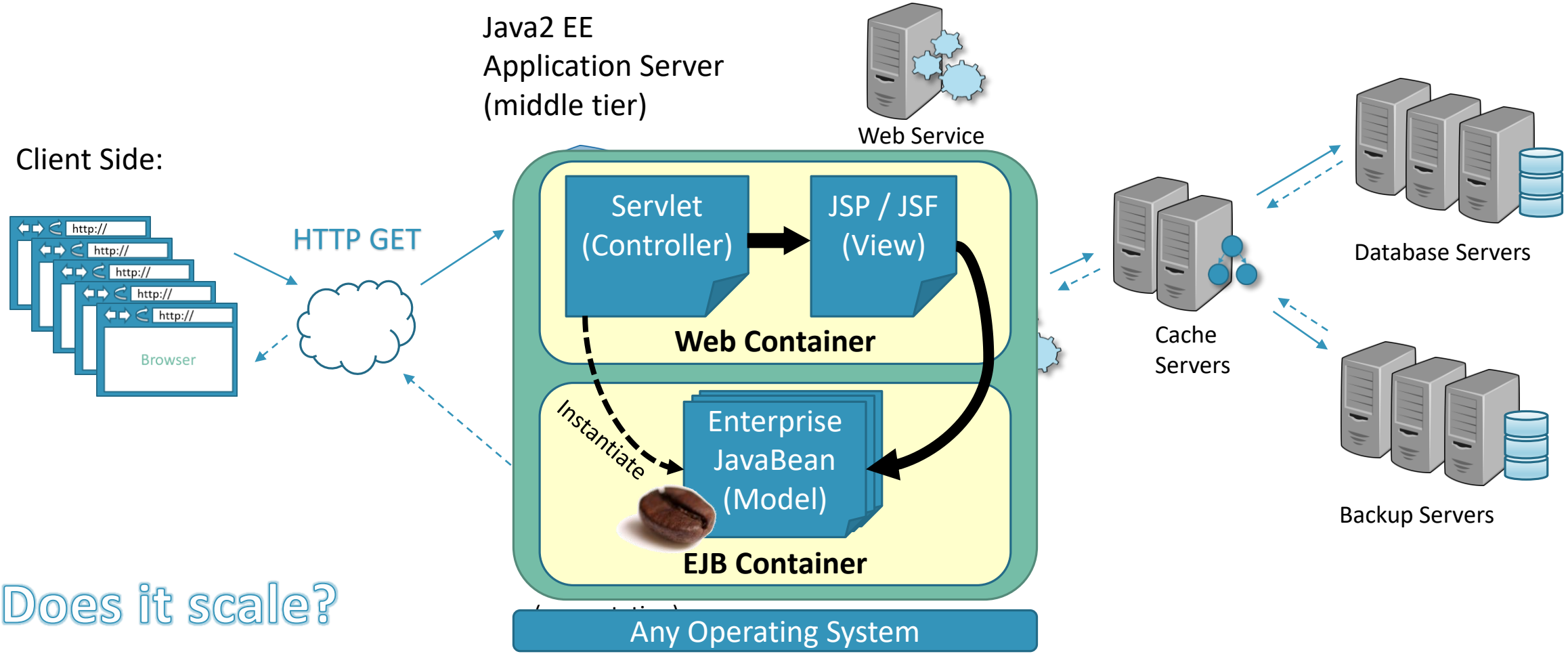
Summary

Our Enterprise



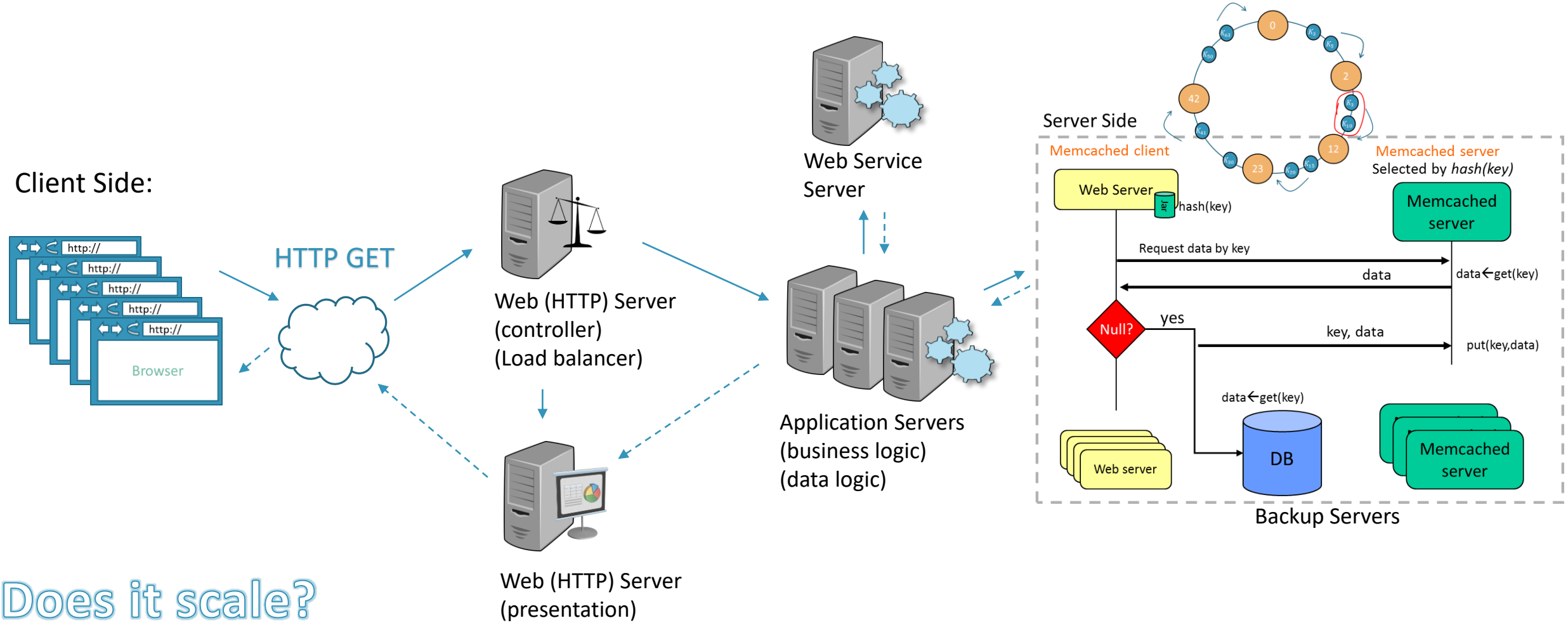
Does it scale?

Our Enterprise



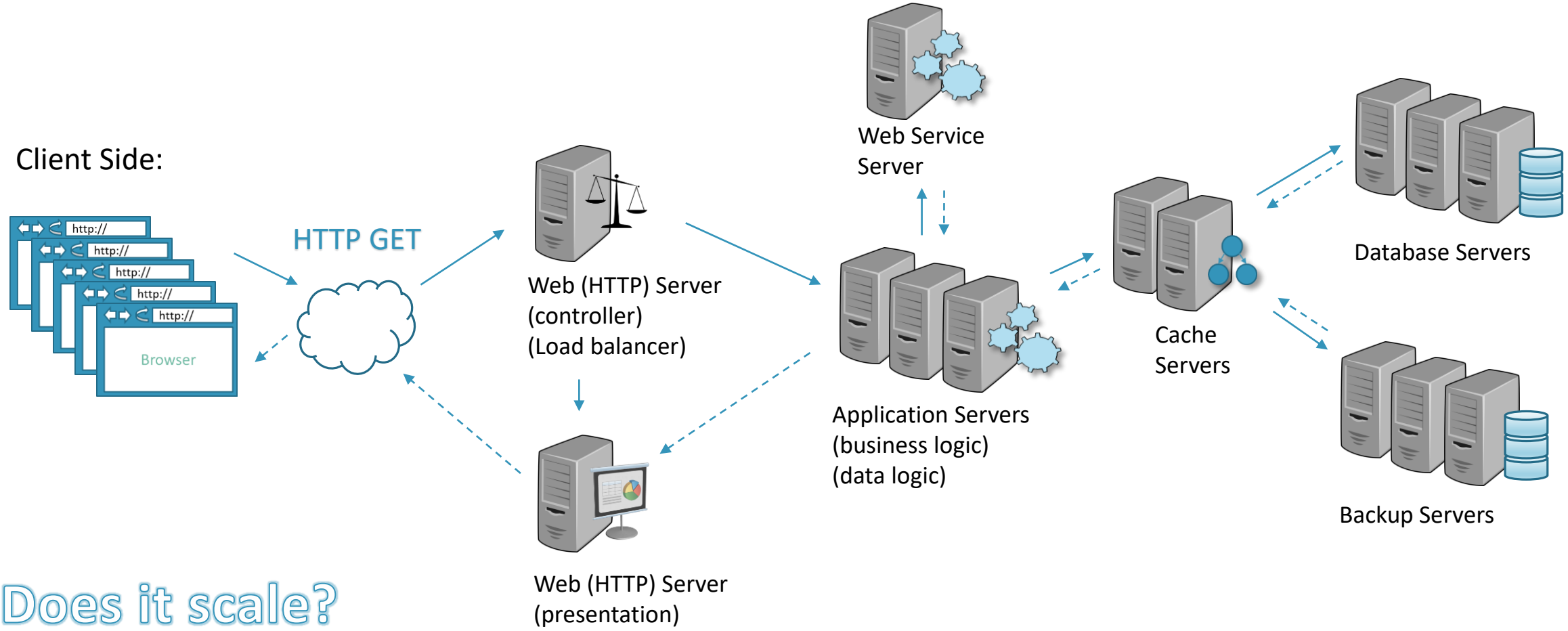
Does it scale?

Our Enterprise



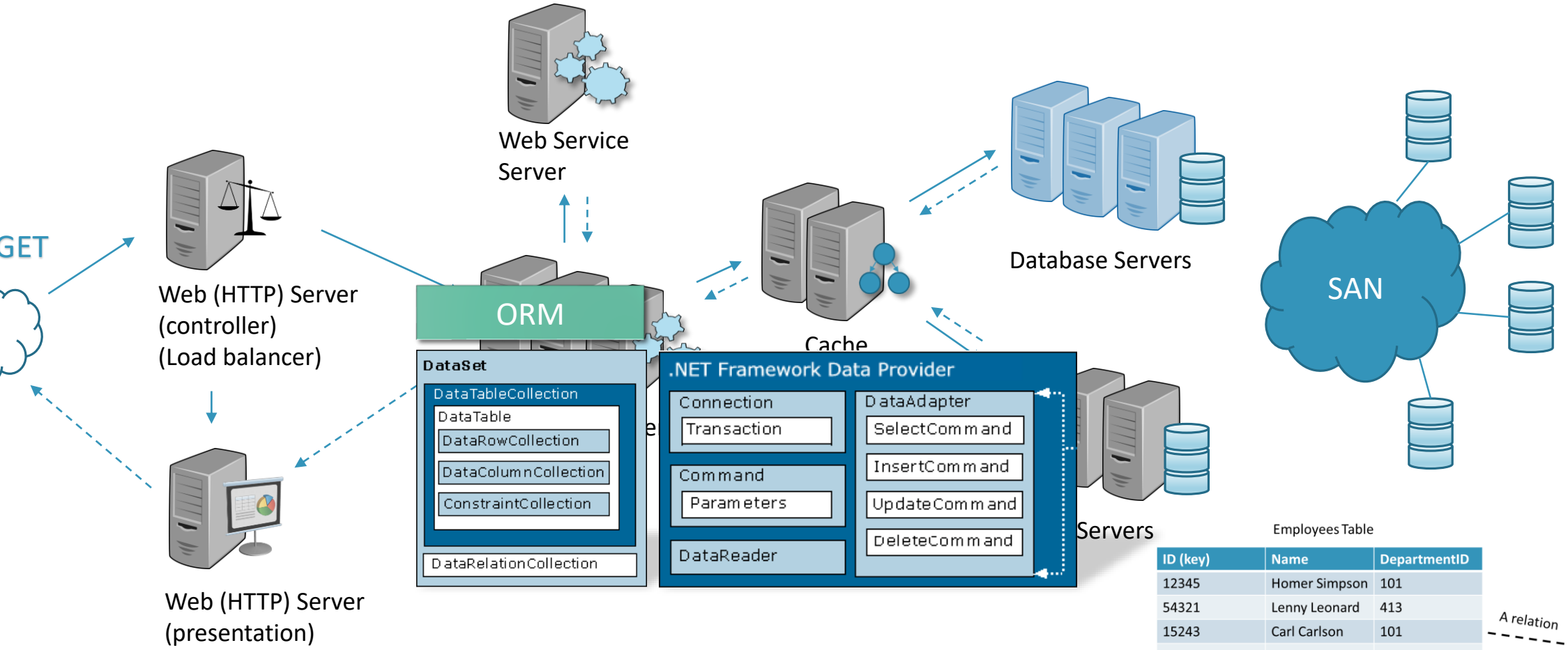
Does it scale?

Our Enterprise



Does it scale?

Our Enterprise



Employees Table

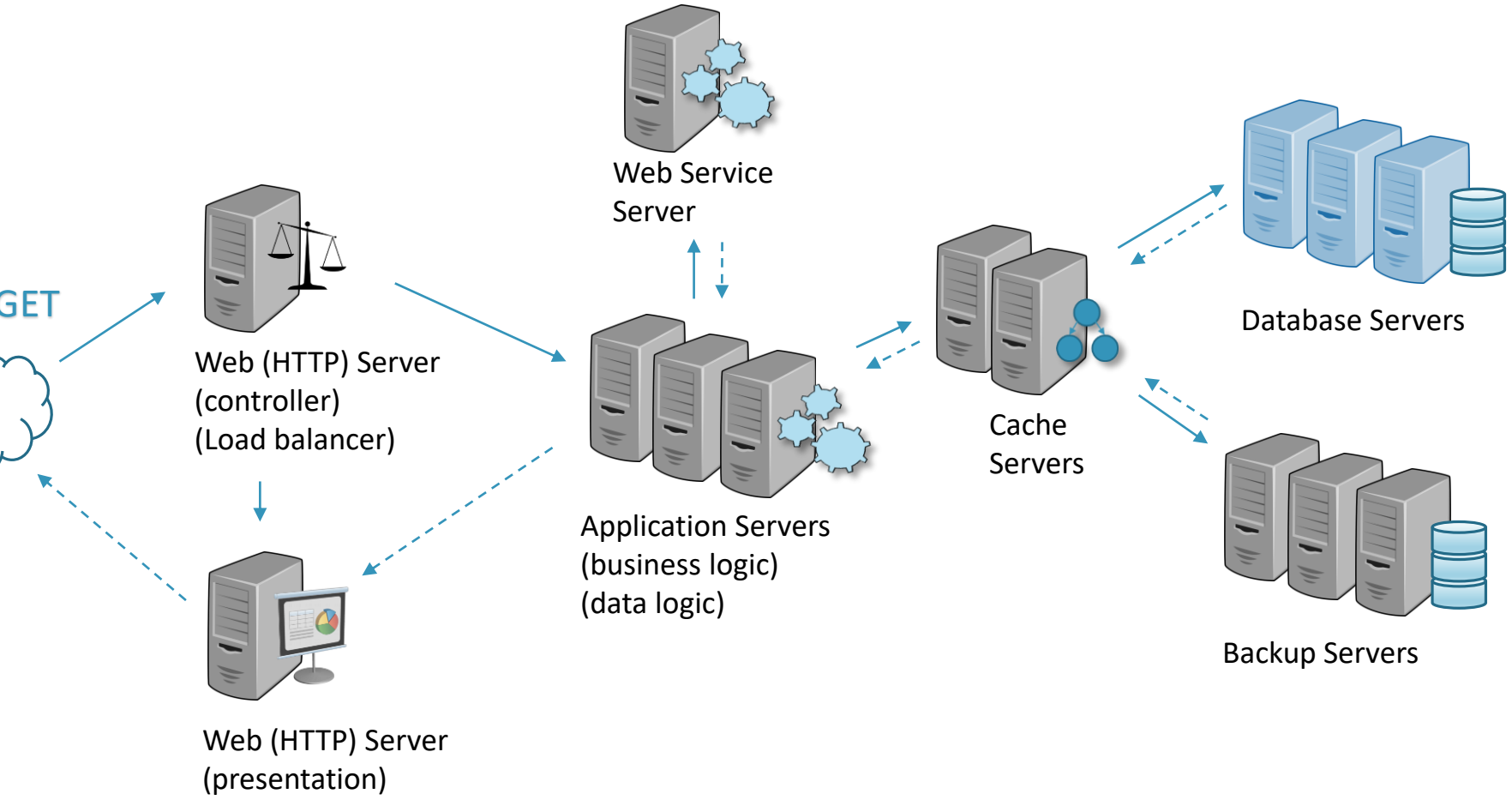
ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
...	...	...

A relation

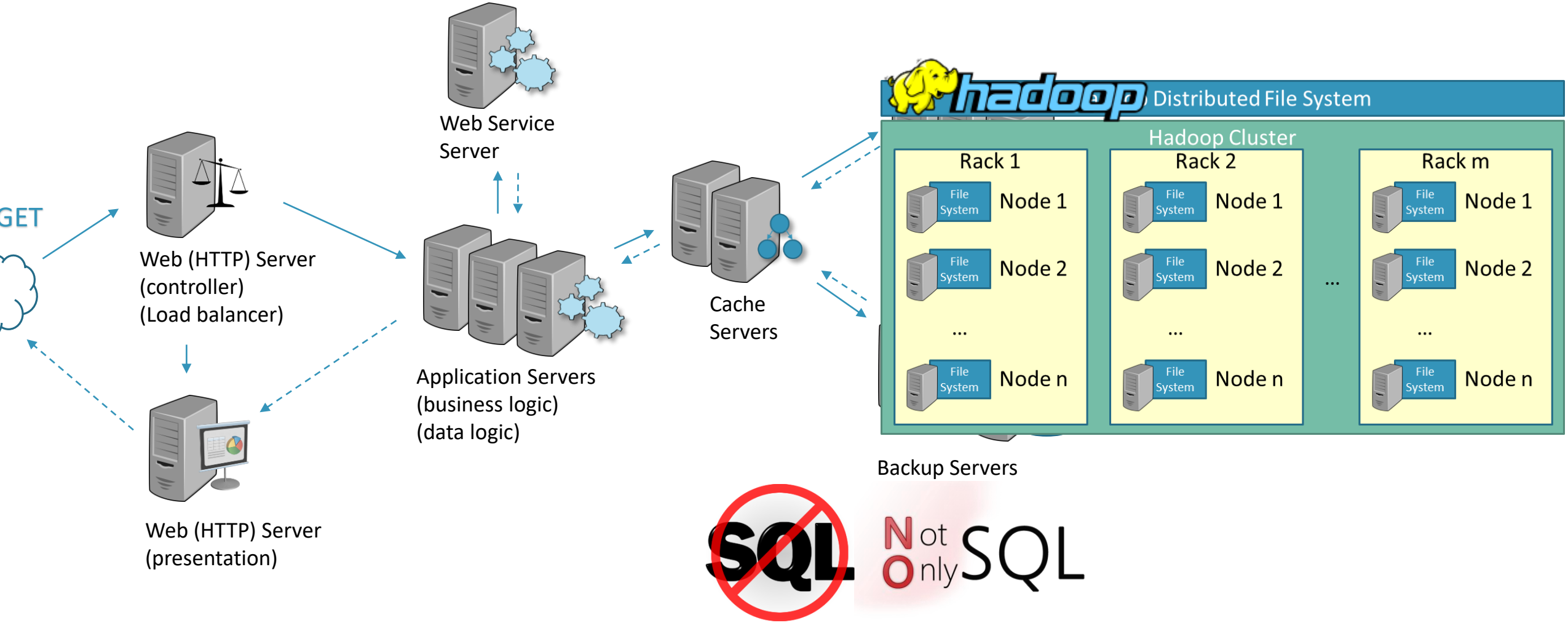
Department Table

Number	Name	manager
101	control	12345
413	R&D	96534
...	...	...

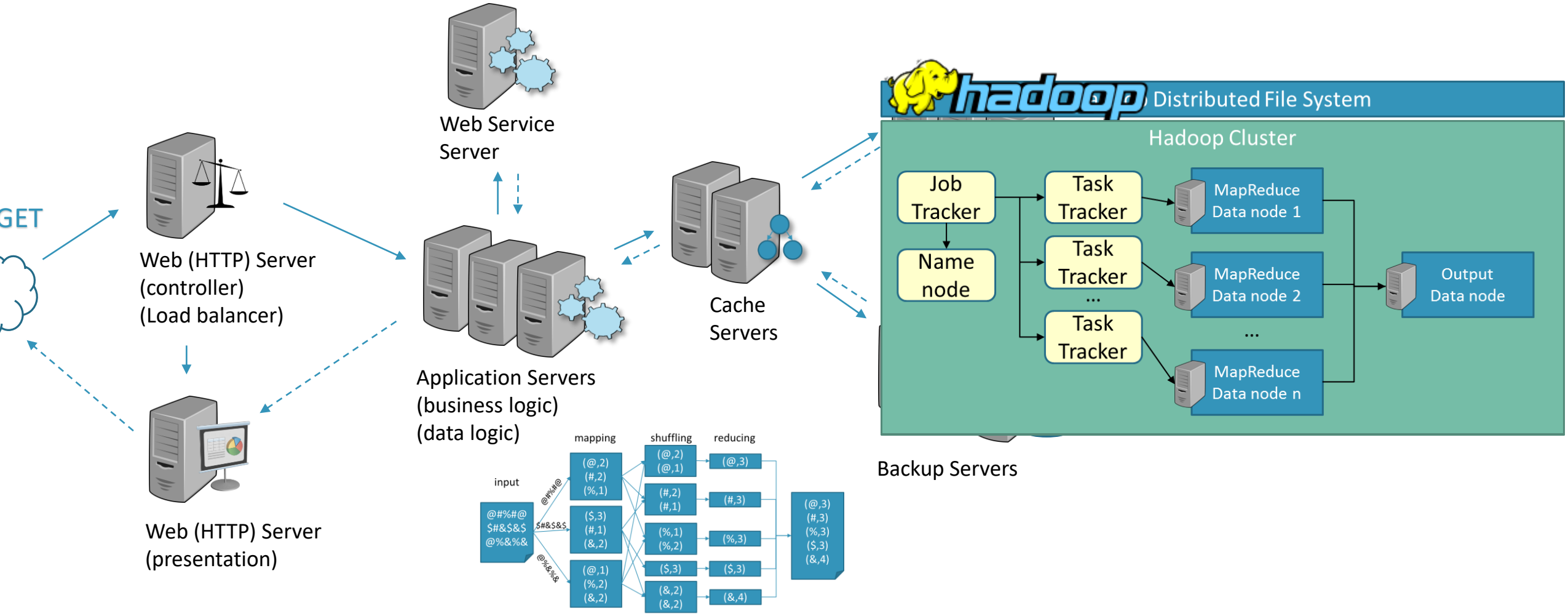
Our Enterprise



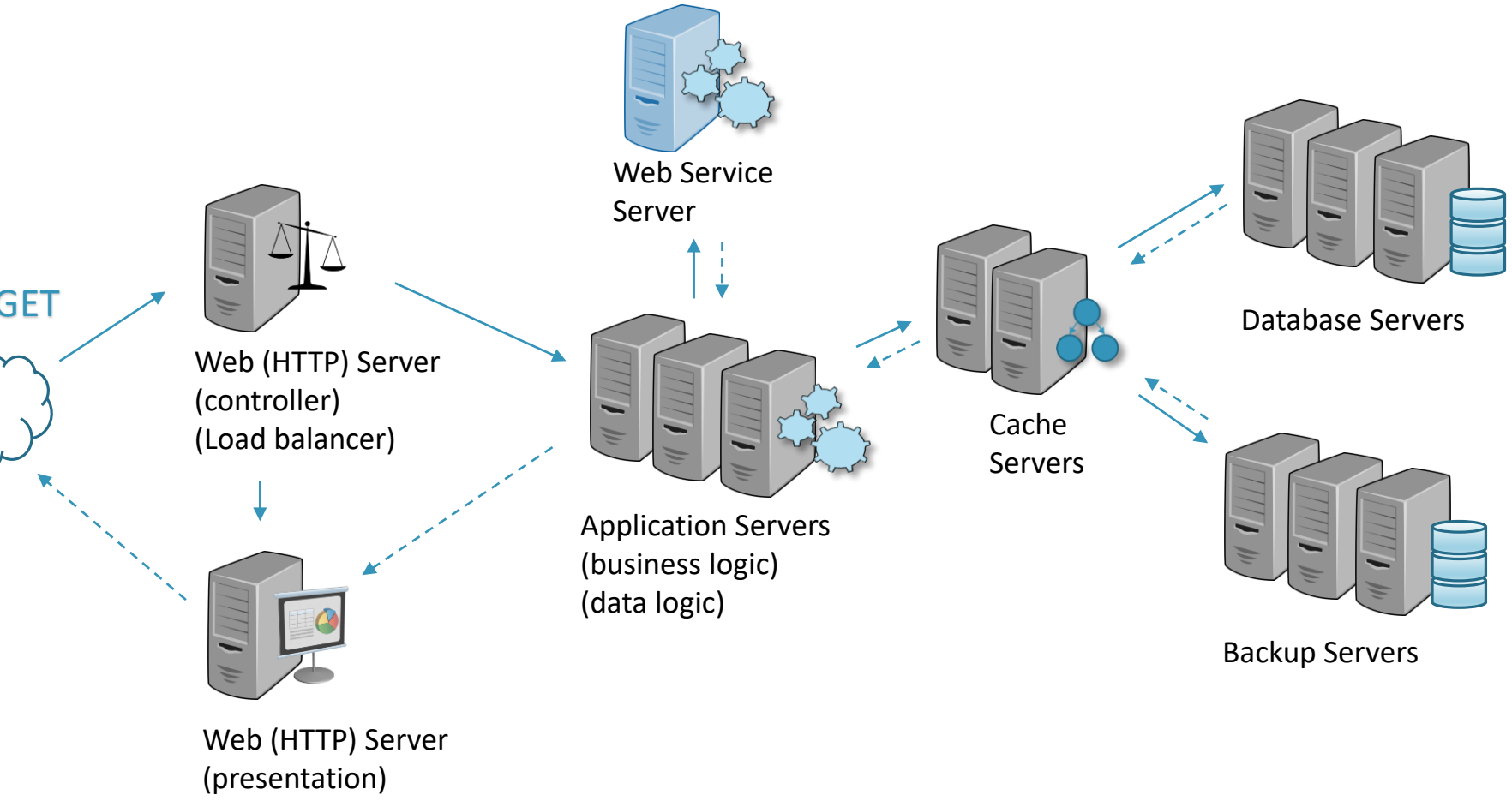
Our Enterprise



Our Enterprise



Our Enterprise



Our Enerprise

UDDI - Universal Description, Discovery and Integration

lists what services are available

Service Description

Service Broker

WSDL - Web Services Description Language

```
▼<definitions xml
 xmlns:wsam="http
 targetNamespace=
  ▼<types>
    ▼<xsd:schema>
       <xsd:impor
     </xsd:schema
   </types>
  ▼<message name=
     <part name="
     <part name="
   </message>
```

WSDL

```
<soap:envelope>
  <soap:body>
    <generate>
      <rows>10</rows>
      <cols>5</cols>
    </generate >
  </soap:body>
</soap:envelope>
```

SOAP - Request

ess Protocol, transfers XML (data)

esponse

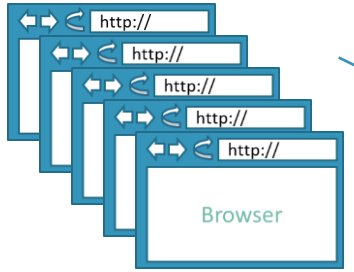
Service Requester

(Web) Service Provider

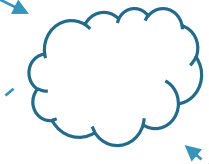
SOA – Service Oriented Architecture

Service – a well defined function that does not depend on the state of other services

Client Side:



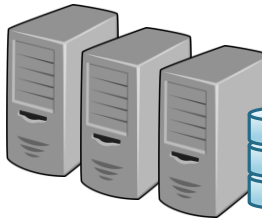
HTTP GET



Web (HTTP) Server (controller) (Load balancer)

Web Service Server

Cache Servers



Database Servers

Our Enterprise

SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

REST

```
http://www.acme.com/phonebook/UserDetails/12345
```