Service Oriented Architecture & Web services

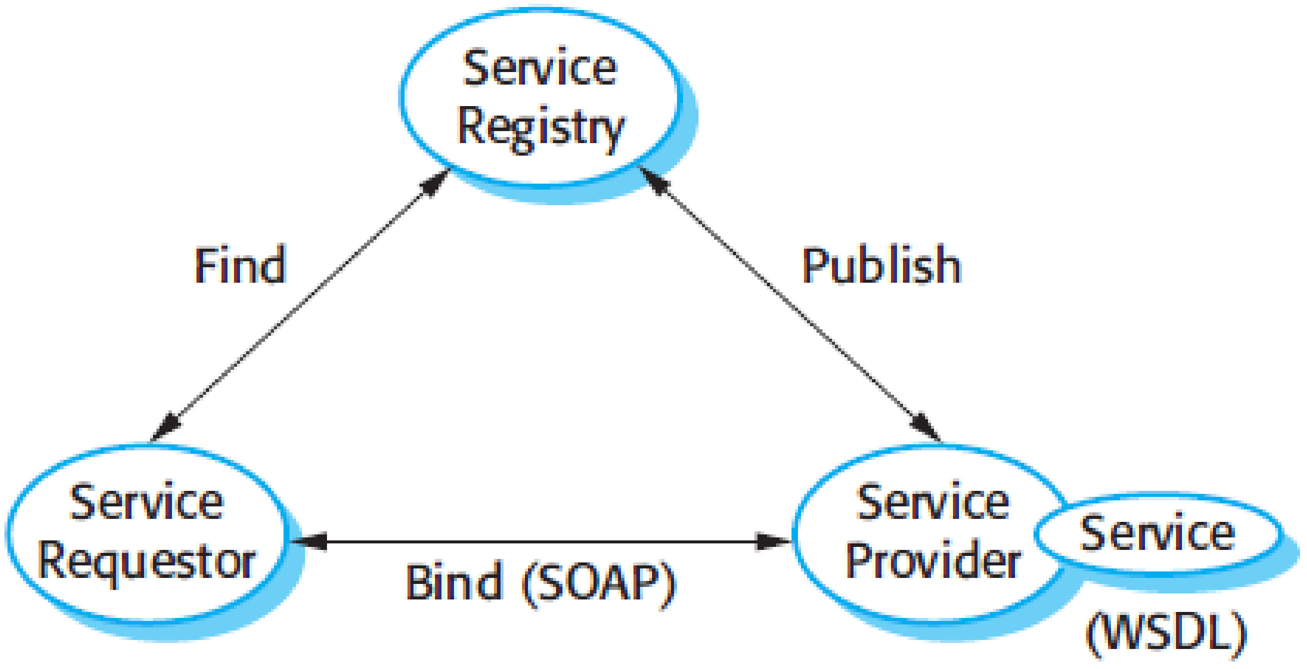
Service-oriented Architecture

- A service is some functionality that is exposed for use by other processes.
- Such exposure is achieved through some interface that can be used by any service requestor.
- A web service is functionality that is exposed and would be accessed through web technologies.
- Service-oriented architecture examines how to build, use, and combine services.
- Instead of creating a large software suite that does everything, we can build and use services, and design an architecture that supports using such services.
- Note: Such definitions are retrieved from UoFA course on service-oriented architecture from Coursera.

Service-oriented Architecture

- Instead, a web service defines such communication through a set of communication protocols and standard data formats.
- Hence, a web application for travelling (e.g., travgo) may take information from services that obtain flight prices, services that obtain hotel prices, car rental services.
- However, some non-functional requirements become very important, like:
  - Response time
  - Availability

Roles of SOA Building Blocks



by Dirk Krafzig, Karl Banke, and Dirk Slama

SOAP versus REST Example

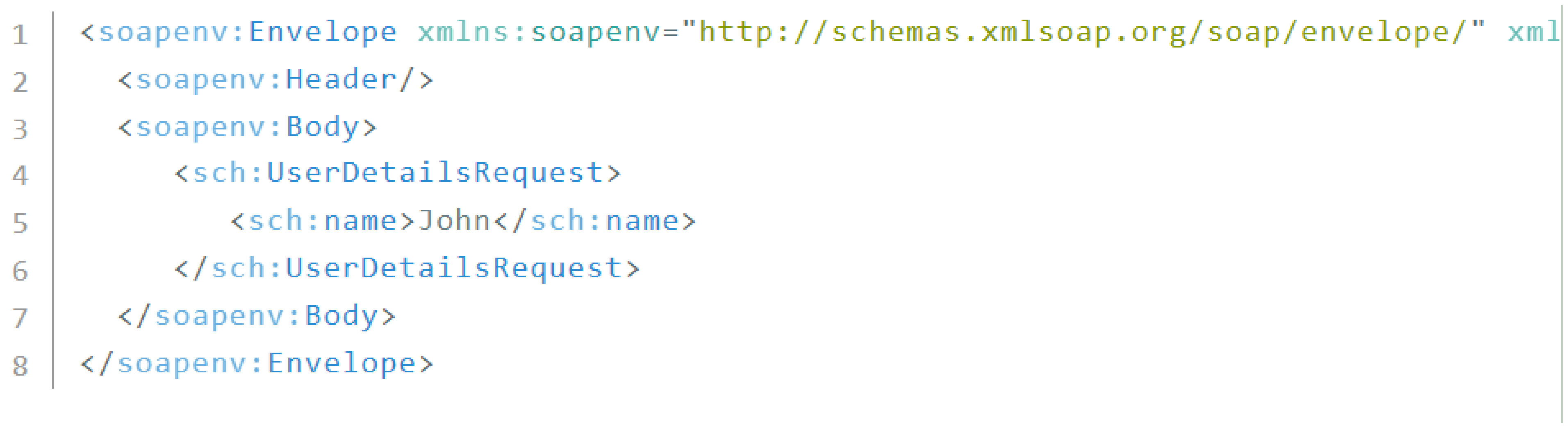
- To better grasp the practical differences between SOAP and REST, we have created an example of how the same operation could be performed using the two technologies.
- In the example, we are making a request for user details.
- Example retrieved from: https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles#soap-vs-rest

SOAP Example

- Using SOAP, the request to the API is an HTTP POST request with an XML request body.
- In this case, we want to fetch the user with the name “John.”

SOAP Example

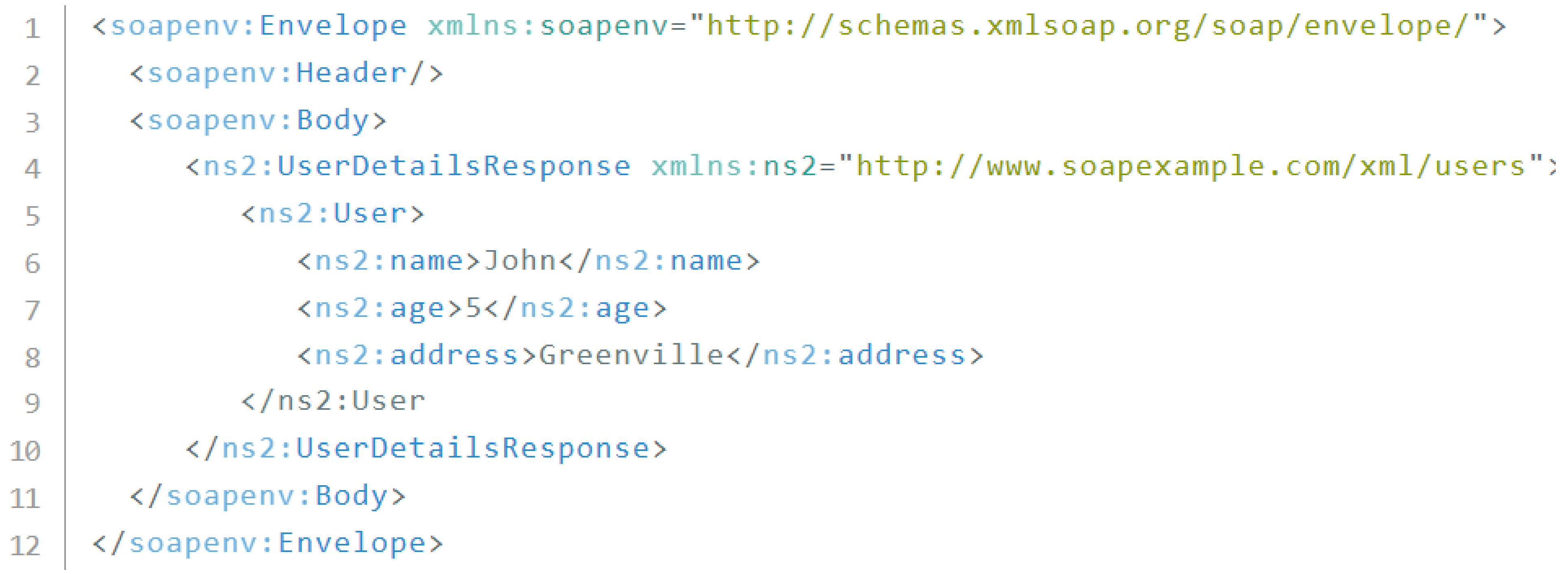
- The request body consists of an envelope which is a type of SOAP wrapper that identifies the requested API, and a SOAP body that holds the request parameters.



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xml
  <soapenv:Header/>
  <soapenv:Body>
     <sch:UserDetailsRequest>
        <sch:name>John</sch:name>
     </sch:UserDetailsRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Example

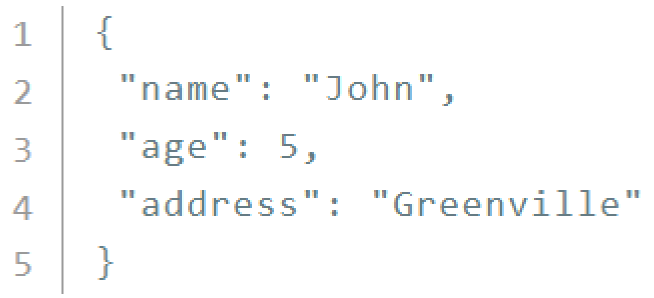
- The response, just like the request, consists of a SOAP envelope and a SOAP body. In this case, the SOAP body represents the requested user data.



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
     <ns2:UserDetailsResponse xmlns:ns2="http://www.soapexample.com/xml/users">
        <ns2:User>
           <ns2:name>John</ns2:name>
           <ns2:age>5</ns2:age>
           <ns2:address>Greenville</ns2:address>
        </ns2:User
     </ns2:UserDetailsResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

REST Example

- REST APIs can be called with all of the HTTP verbs.
- To get a resource, in this case, a user, a GET request is used.
- While the SOAP request holds the user’s name in the body, a REST API accepts GET parameters from the URI.
- GET https://restexample.com/users?name=John



```
{
  "name": "John",
  "age": 5,
  "address": "Greenville"
}
```

Required Reading

- Chapter 5, from Ian Gorton’s: Essential Software Architecture, Springer Verlag Second Edition, 2011.

References

- Distributed Systems: Concepts and Design, 5th edition. Coulouris, et. Al
- http://en.wikipedia.org/wiki/Service-oriented_architecture
- http://en.wikipedia.org/wiki/Web_service
- Software Engineering (9th Edition), Ian Sommerville
- Cook, William R., and Janel Barfield. "Web services versus distributed objects: A case study of performance and interface design." Web Services, 2006. ICWS'06. International Conference on. IEEE, 2006.

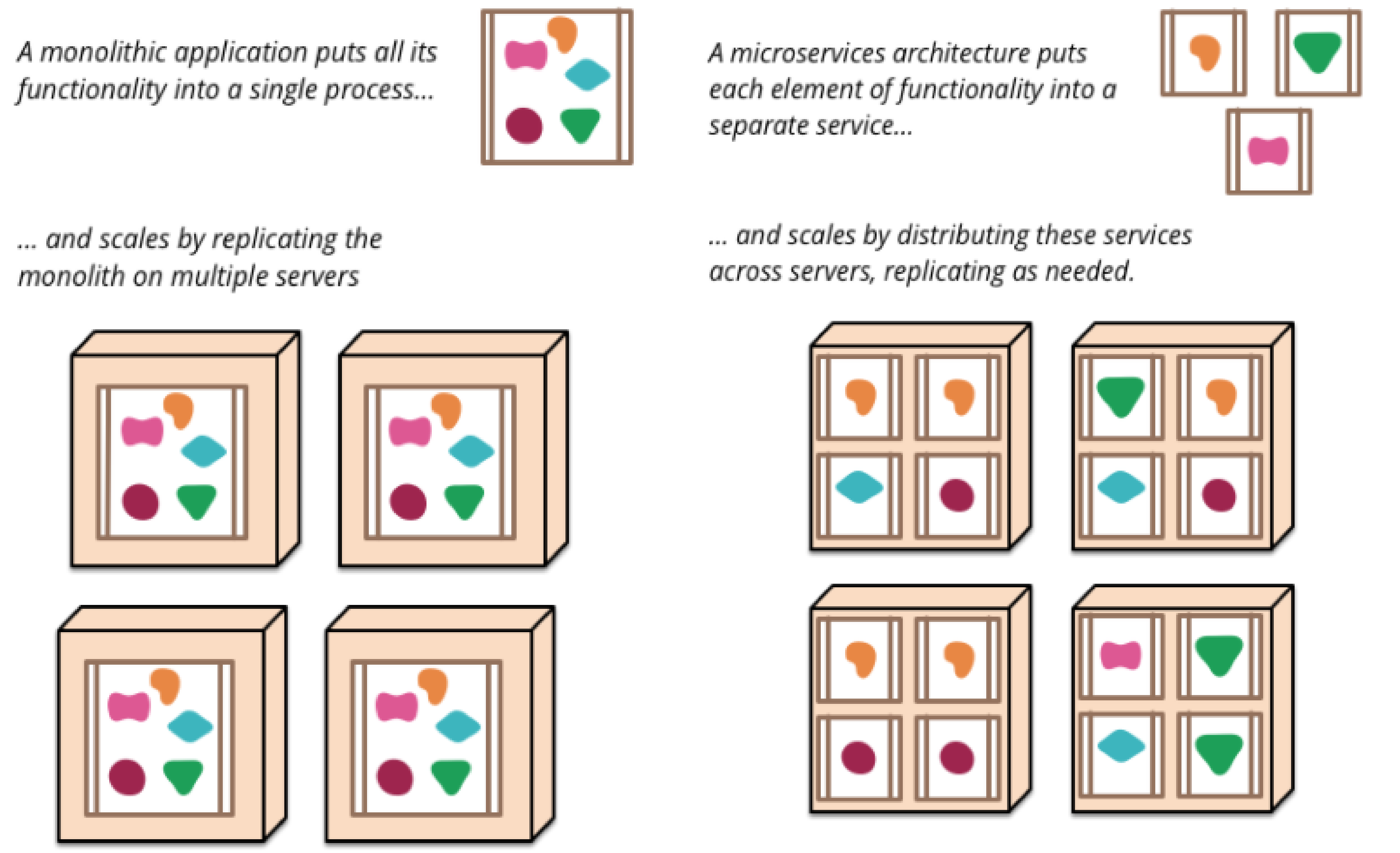
RESTful versus SOAP example: https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles#soap-vs-rest

Microservices

Microservice Architectural Style

- The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.
- These services are built around business capabilities and independently deployable by fully automated deployment machinery.
- There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
- Microservices vs. monolithic style?
- Scaling monolithic applications?

Microservice Architectural Style



A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

- Limitations of monolithic applications?

Microservice Architectural Style

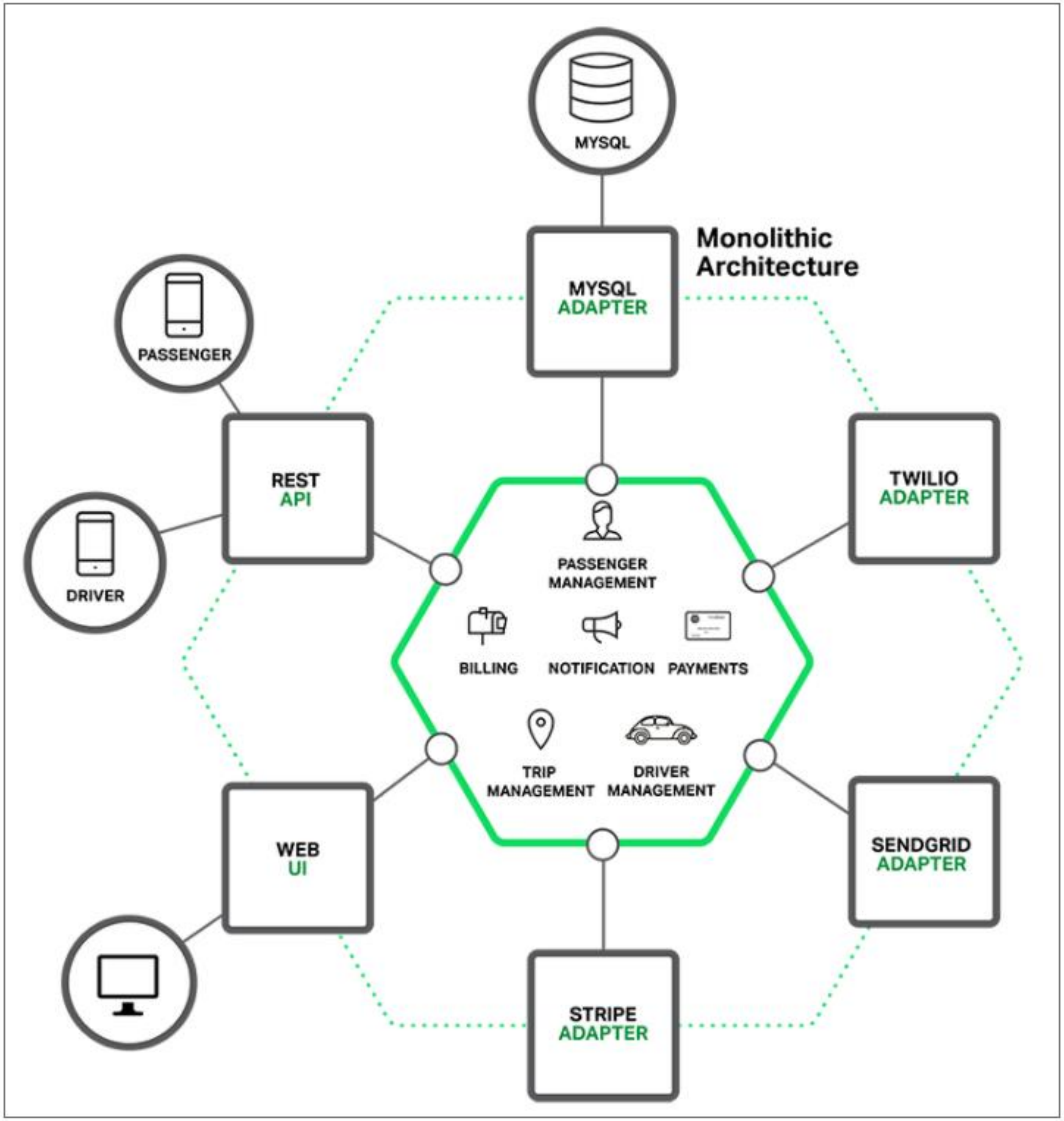
- Many organizations, such as Amazon, eBay, and Netflix, have solved this problem by adopting what is now known as the Microservices Architecture pattern.
- Instead of building a single monstrous, monolithic application, the idea is to split your application into set of smaller, interconnected services.
- A service typically implements a set of distinct features or functionality, such as order management, customer management, etc.

Microservice Architectural Style

- Each microservice is a mini-application that has its own hexagonal architecture consisting of business logic along with various adapters.
- Some microservices would expose an API that’s consumed by other microservices or by the application’s clients. Other microservices might implement a web UI. At runtime, each instance is often a cloud VM or a Docker container.
- Example:
  - Consider that you are building a competitor application to Uber
  - How would the architecture look?

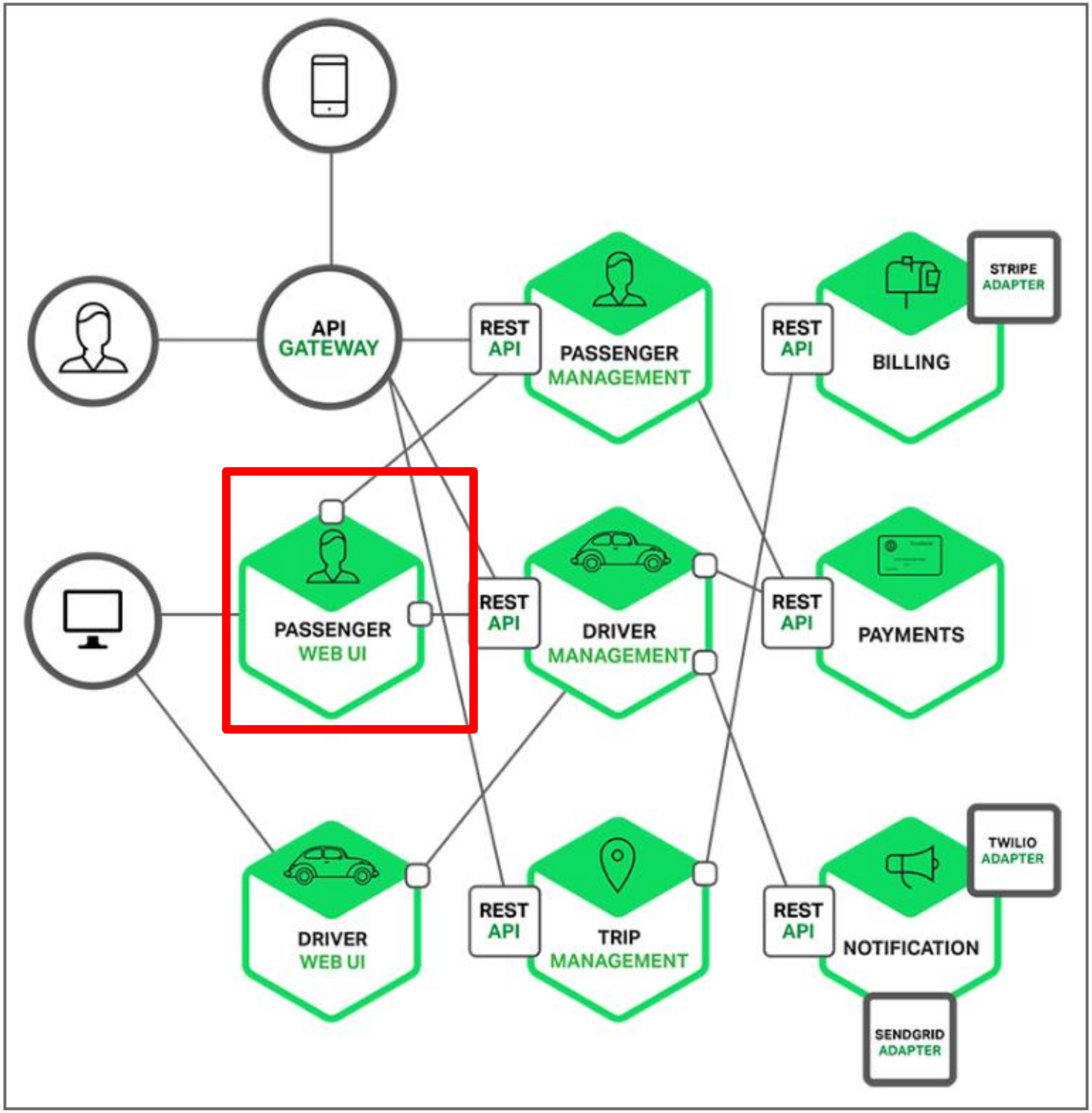
Uber competitor monolithic architecture

Pros?



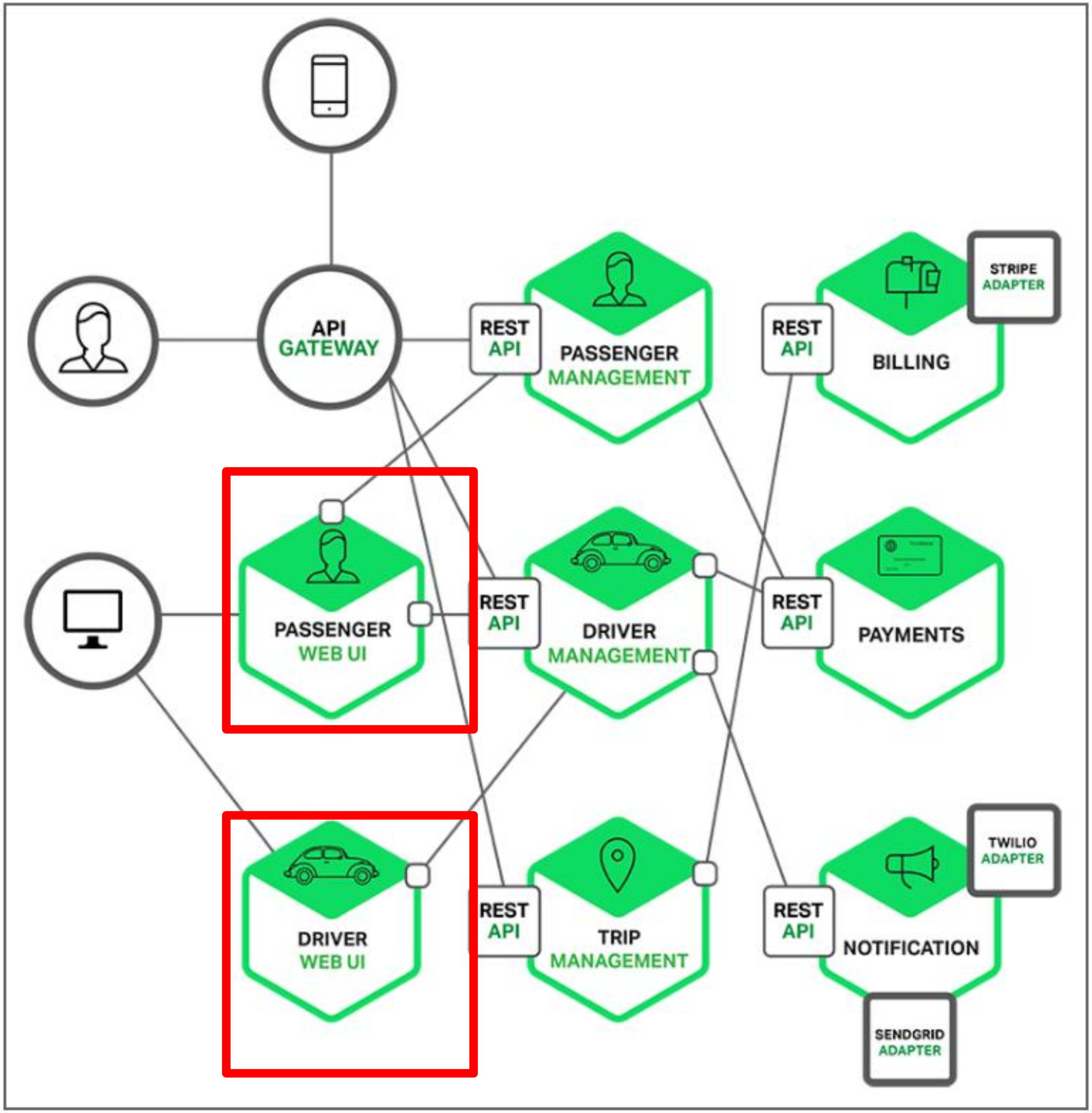
Uber competitor microservice architecture

Pros?



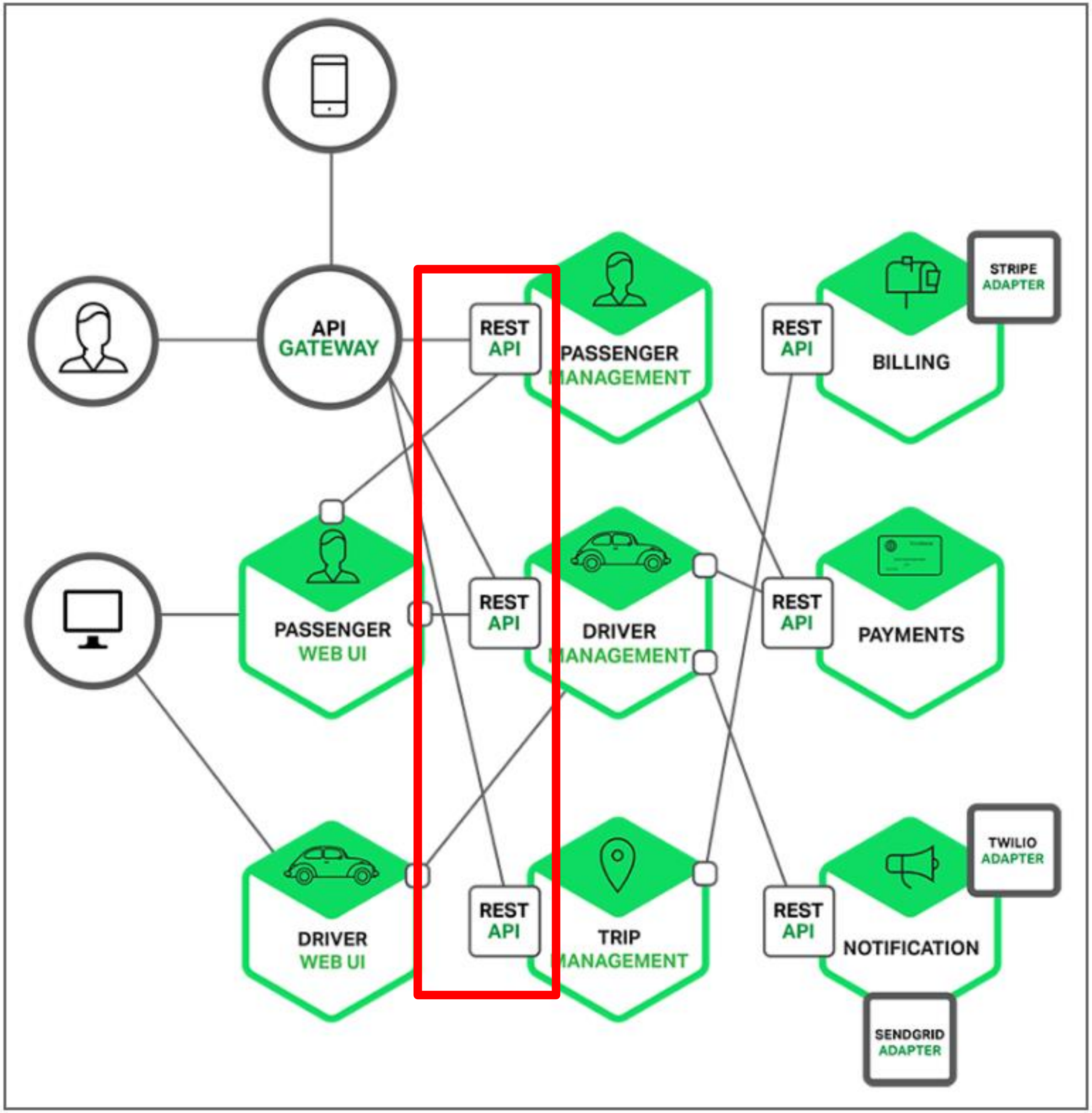
Uber competitor microservice architecture

Pros?



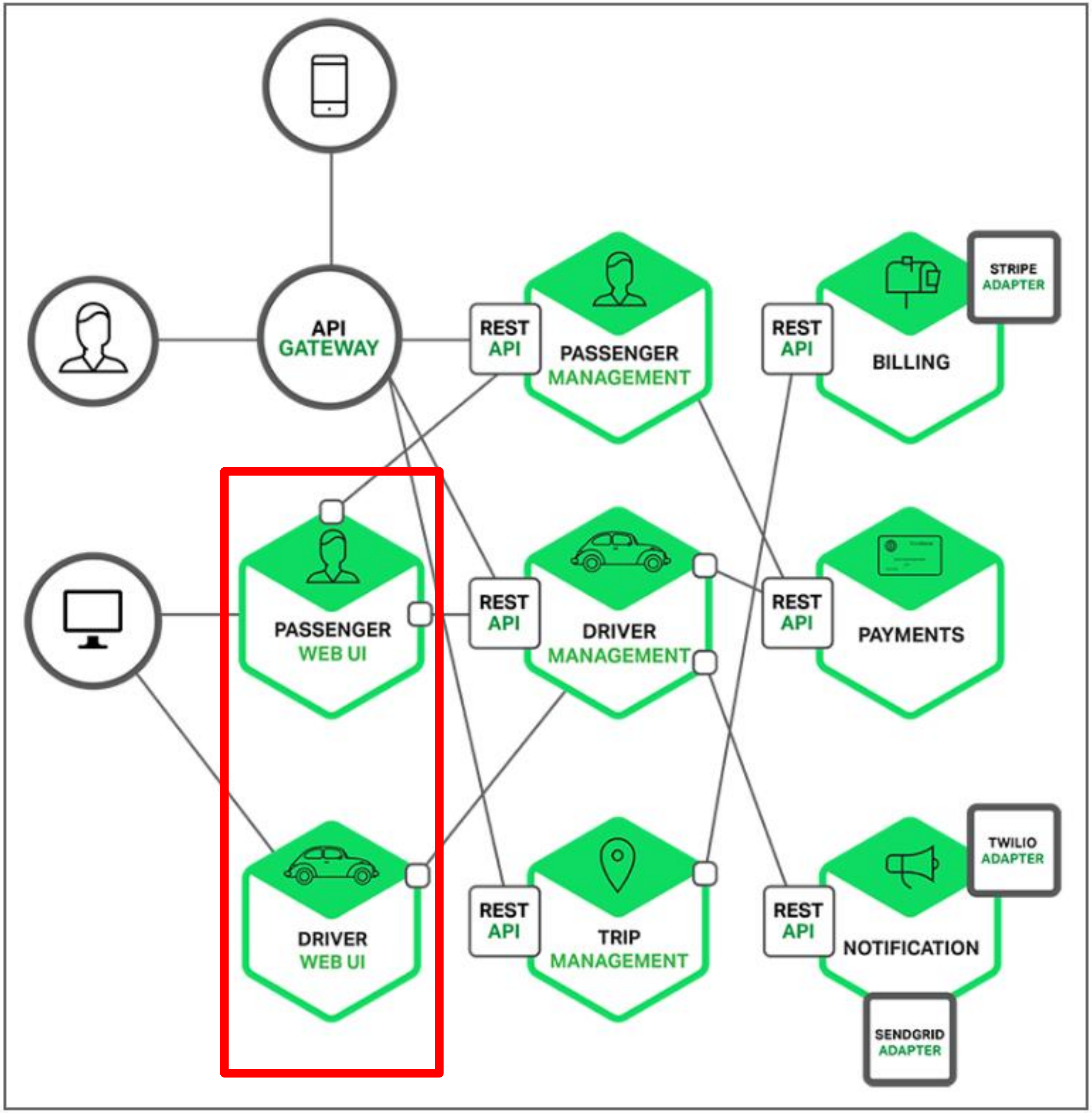
Uber competitor microservice architecture

Pros?



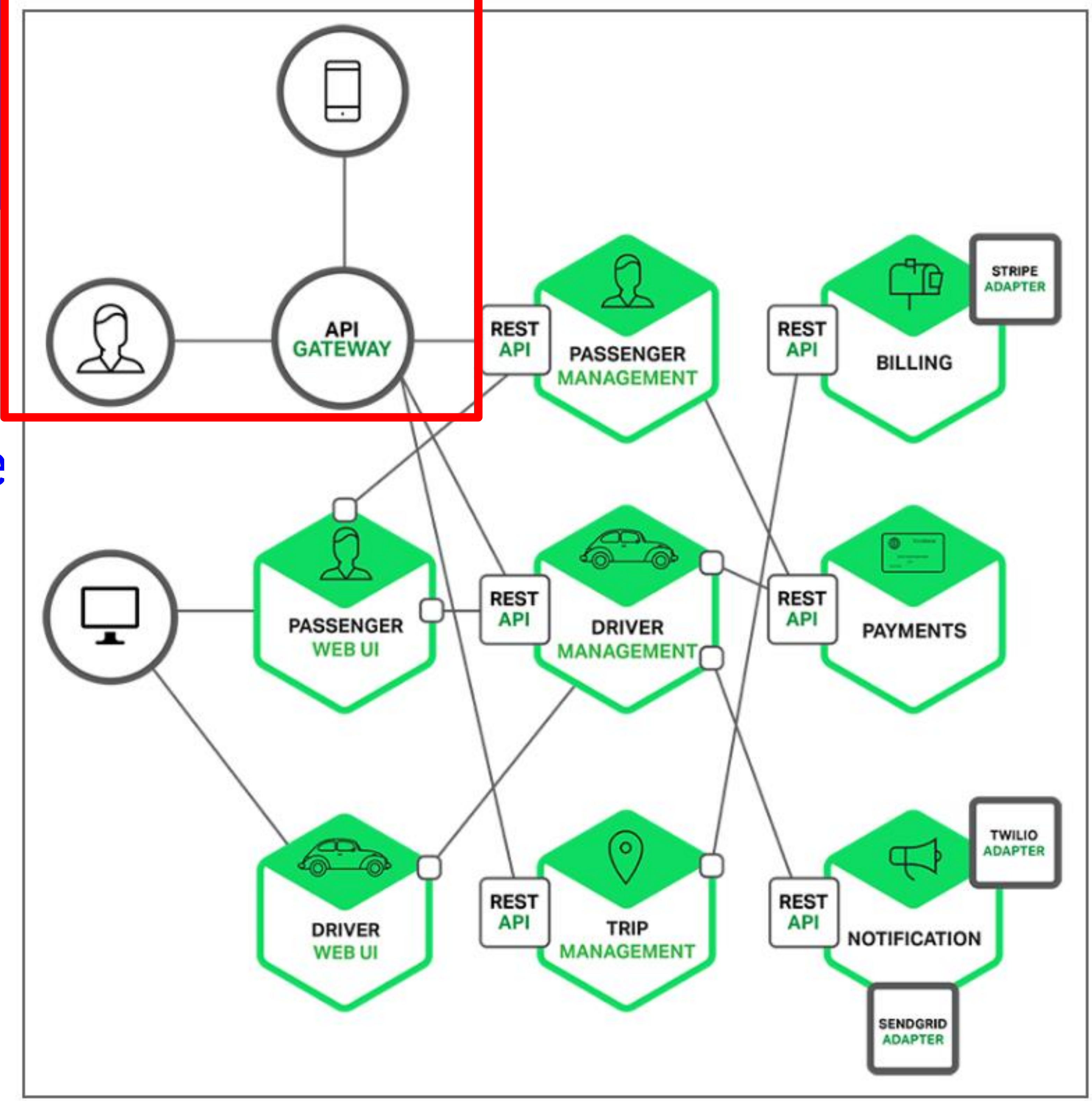
Uber competitor microservice architecture

Pros?



Uber competitor microservice architecture

Pros?



Required Readings

- https://martinfowler.com/articles/microservice-trade-offs.html
- https://martinfowler.com/articles/microservices.html
- https://www.nginx.com/blog/introduction-to-microservices/
- https://www.nginx.com/blog/building-microservices-using-an-api-gateway/
- https://www.nginx.com/blog/building-microservices-inter-process-communication/

Application Services Design Considerations

- **State Management**
- Application Servers
- Load Balancing

State Management

- Service implementations that need to scale should avoid storing conversational state.
- Let’s check an example using HTTP stateless protocol.
- Within HTTP, each request is executed independently, without any knowledge of the requests that were executed before it from the same client.
- What does statelessness imply?
  - Every request needs to be self-contained, with sufficient information provided by the client for the web server to satisfy the request regardless of previous activity from that client.

State Management (Cont.)

- What does conversational state mean?
  - *Conversational state* represents any information that is retained between requests such that a subsequent request can assume the service has retained knowledge about the previous interactions.
- Let us see an example…

State Management (Cont.) Example: A skier service API

- A user may request their profile by submitting a GET request as follows: GET /skico.com/skiers/768934
- A user may need to modify their city attribute and send a PUT request to update the resource:

```
PUT /skico.com/skiers/
{
"username": "Ian123",
"email": "i.gorton@somewhere.com"
"city": "Wenatchee"
}
```

State Management (Cont.)
Example: A skier service API

- But…Does the service know the unique ID of that resource when using such PUT request ? (i.e., 768934)
- For this update operation to succeed, the service must have retained conversational state from the previous GET request.
- How to retain conversational state?

State Management (Cont.)
Example: A skier service API

- How to retain conversational state?
  - Session state objects!
  - When the service receives the initial GET request, it creates a session state object that uniquely identifies the client connection.
  - When the subsequent PUT request arrives from the client, it uses the session state object to look up the skierID associated with this session and uses that to update the skier’s home city.
- Services that maintain conversational state are known as stateful services. (Pros?)
- What if you decide to scale such service in terms of users, for a single service instance?
  - Amount of client session states (memory?)
  - Session timeout period. (short versus long time interval)

State Management (Cont.)
Example: A skier service API

- How about a stateless service?
  - A stateless service does not assume any conversational state from previous calls
  - The service should not maintain any knowledge from earlier requests, so that each request can be processed individually.
  - Hence the skier API would be defined as follows:

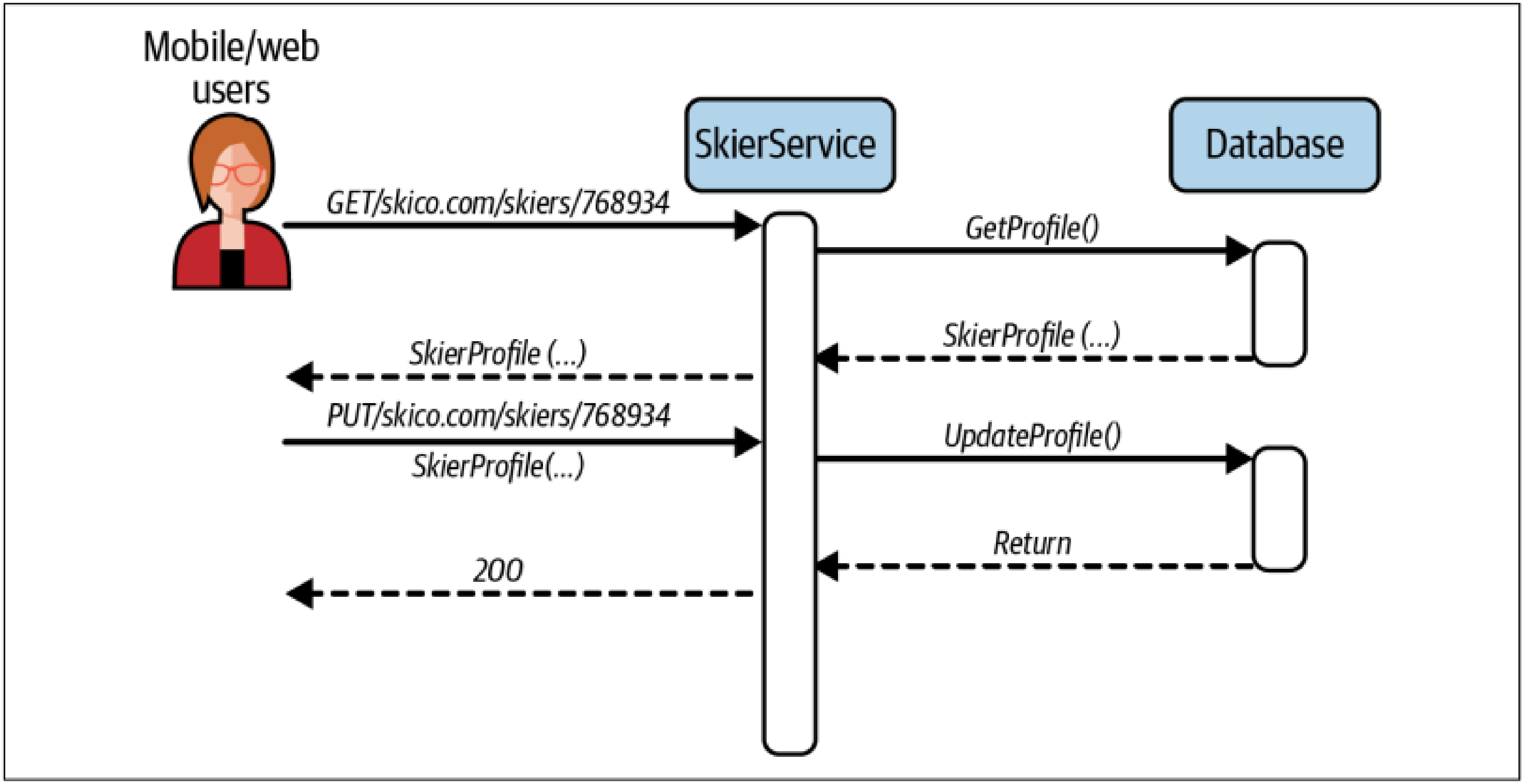
```
PUT /skico.com/skiers/768934
{
"username": "Ian123",
"email": "i.gorton@somewhere.com"
"city": "Wenatchee"
}
```

31

  - sss

State Management (Cont.)
Example: A skier service API

A sequence diagram illustrating this stateless design is shown in Figure 5-2.



- Any scalable service will need stateless APIs. (Why?)
- With every request, you need to send the complete information and access the DB

Application Services Design Considerations

- State Management
- **Application Servers**
- Load Balancing

Application Servers

- The basic role of application servers is to accept requests from clients, apply application logic to the requests, and reply to the client with the request results
- The technological landscape of application servers is broad and complex, depending on the language you want to use and the specific capabilities that each offers.
- In Java, the Java Enterprise Edition (JEE) defines a comprehensive, feature rich, standards based platform for application servers, with multiple different vendor and open source implementations.

Application Servers

- Tomcat is an open source implementation of a subset of the JEE platform, namely the Java servlet, JavaServer Pages (JSP), Java Expression Language (EL), and Java WebSocket technologies.
-

Application Servers

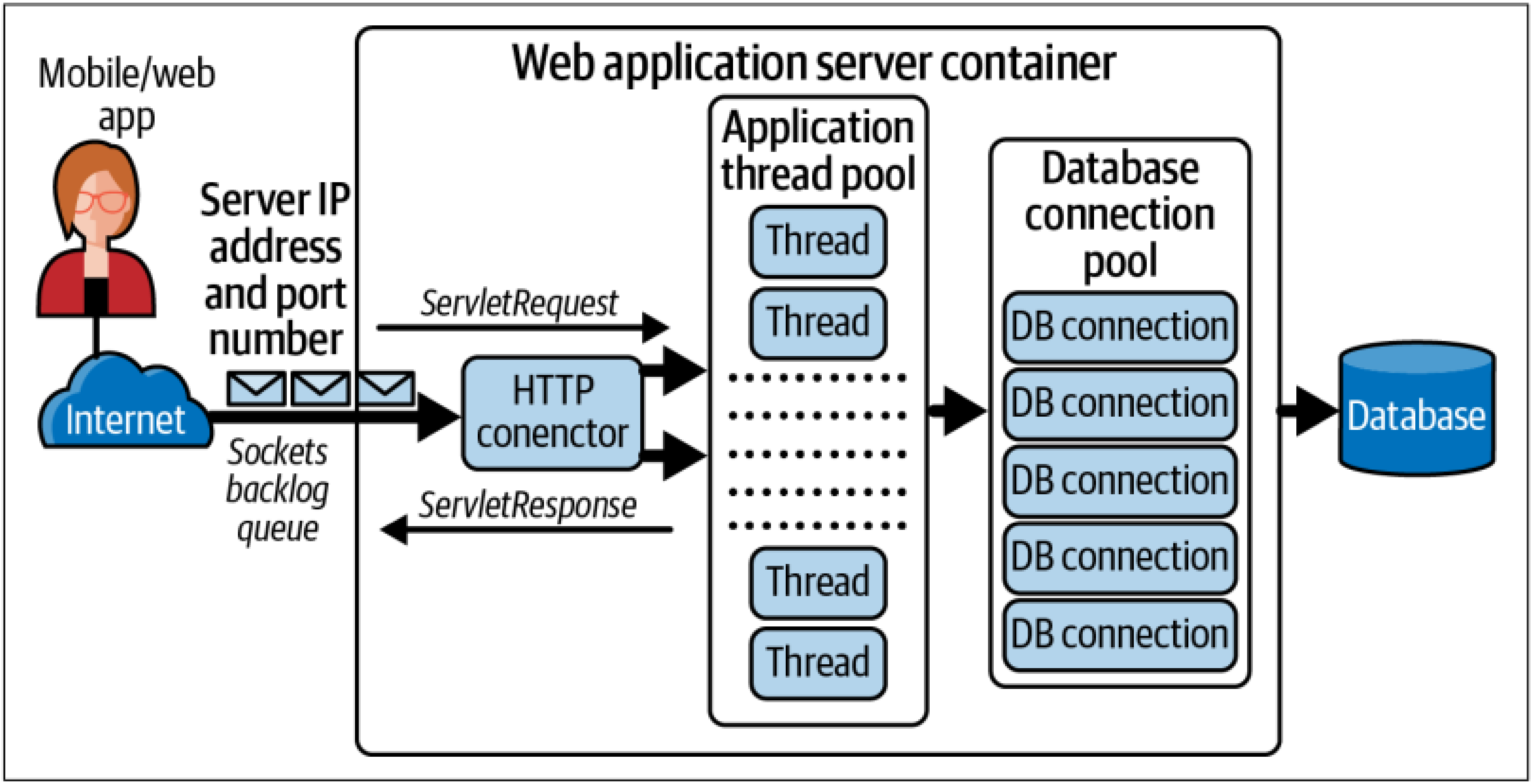


Figure 5-3. Anatomy of a web application server

A *servlet container* is an execution environment for application-defined servlets. Servlets are dynamically loaded into this container

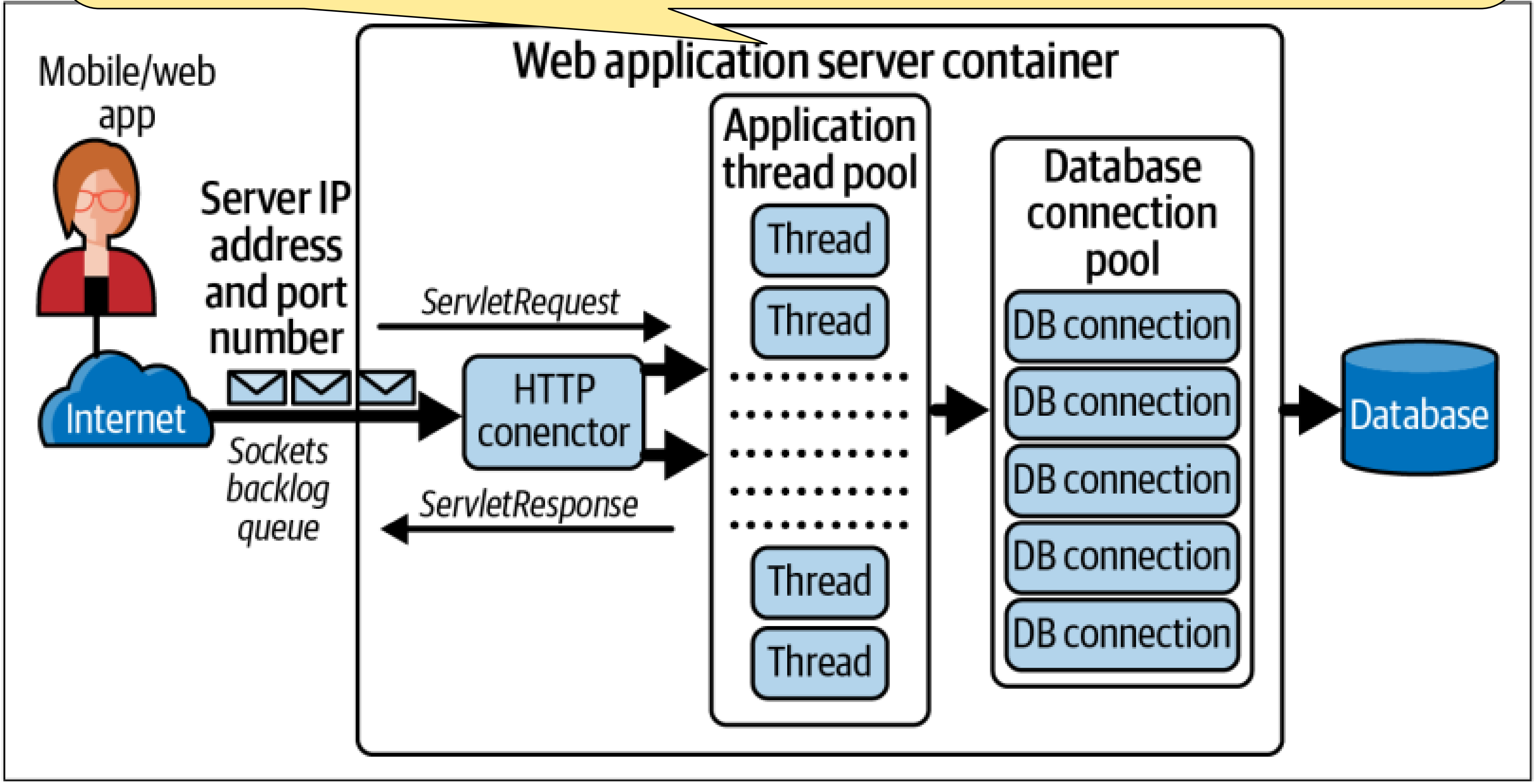


Figure 5-3. Anatomy of a web application server

Requests arrive at the IP address of the server, which is listening for traffic on specific ports.

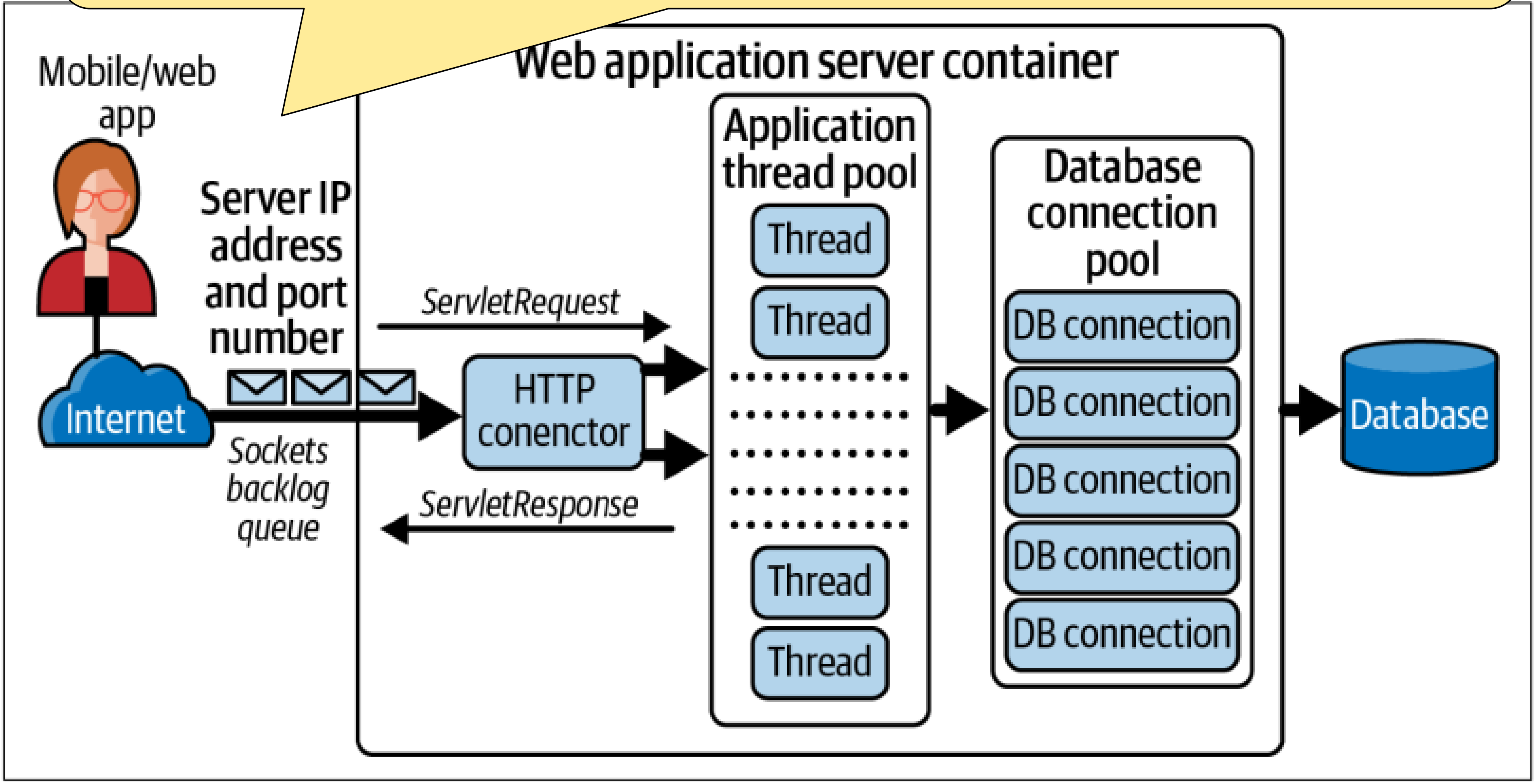
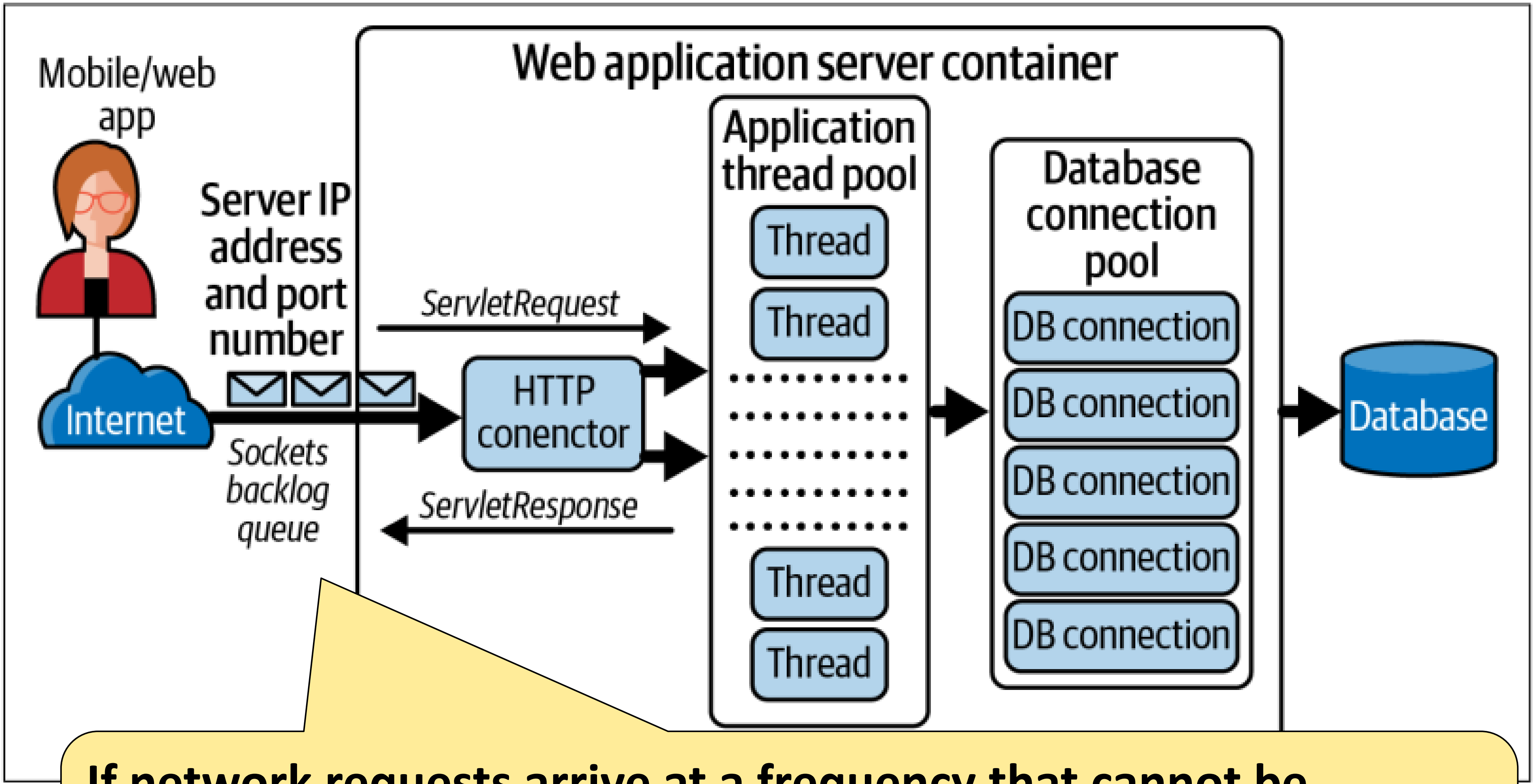


Figure 5-3. Anatomy of a web application server

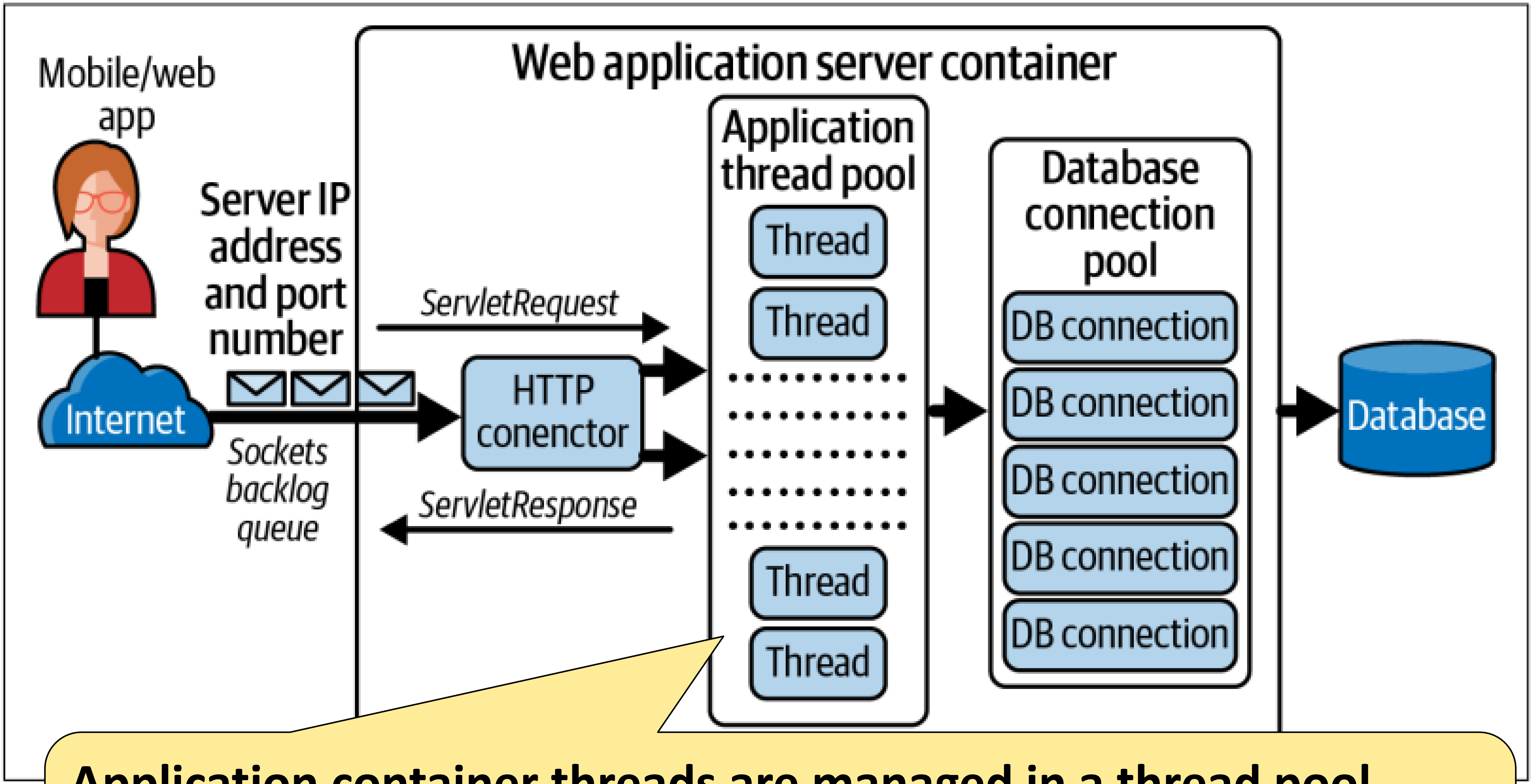
Application Servers



Fig

If network requests arrive at a frequency that cannot be processed by the TCP listener, pending requests are queued up in the *Sockets Backlog*.

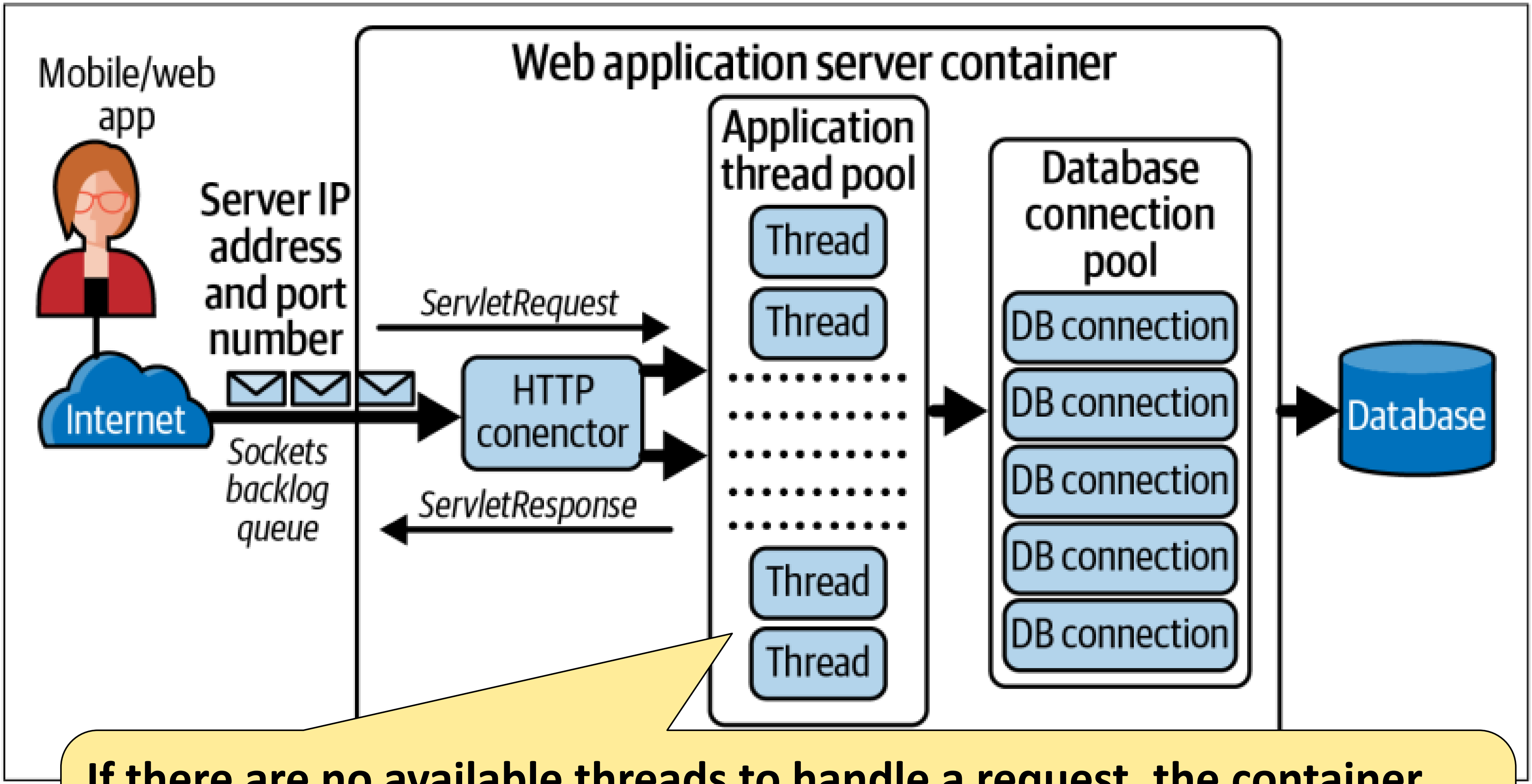
Application Servers



Fi

Application container threads are managed in a thread pool, essentially a Java Executor, which by default in Tomcat is a minimum size of 25 threads and a maximum of 200.

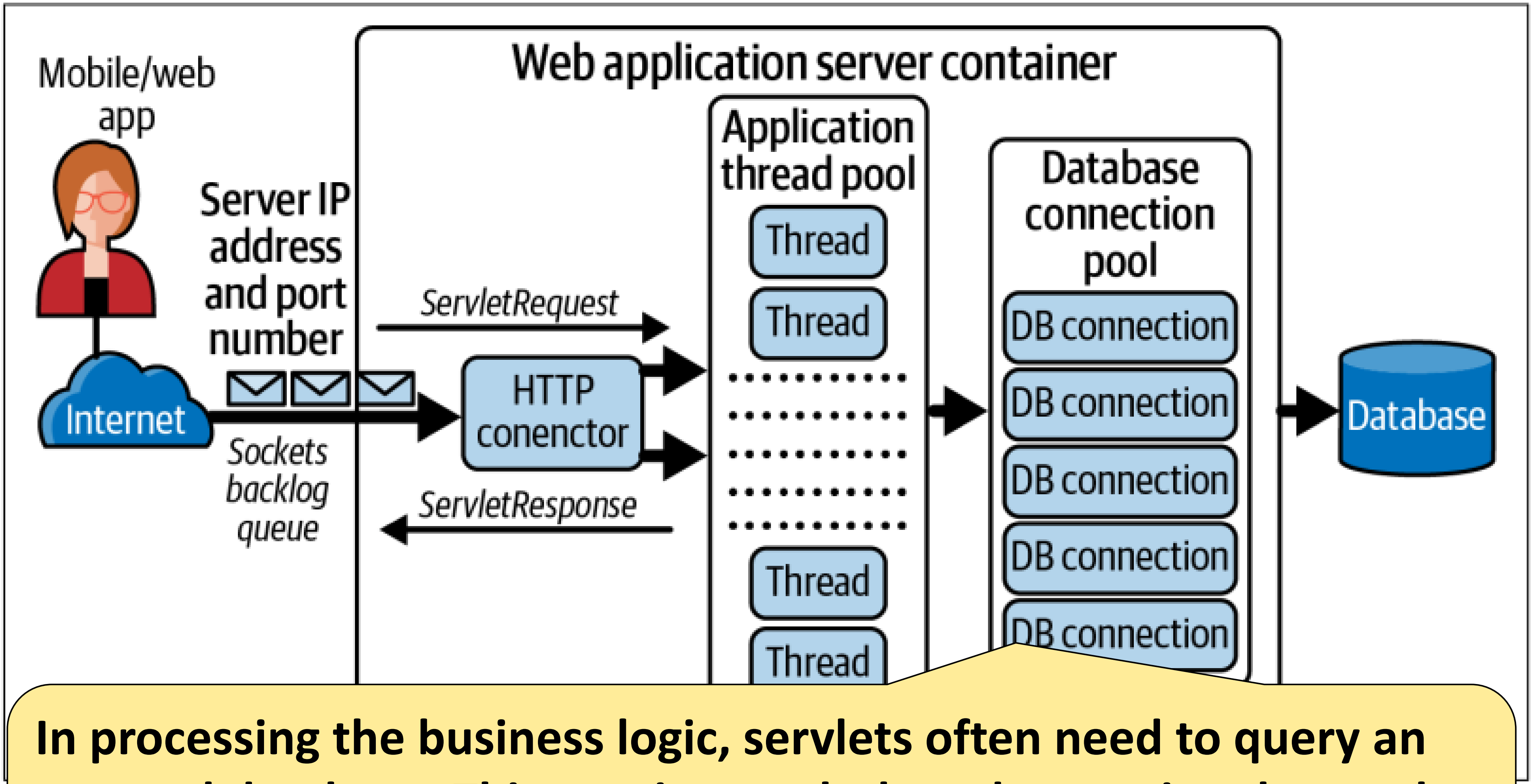
Application Servers



Fig

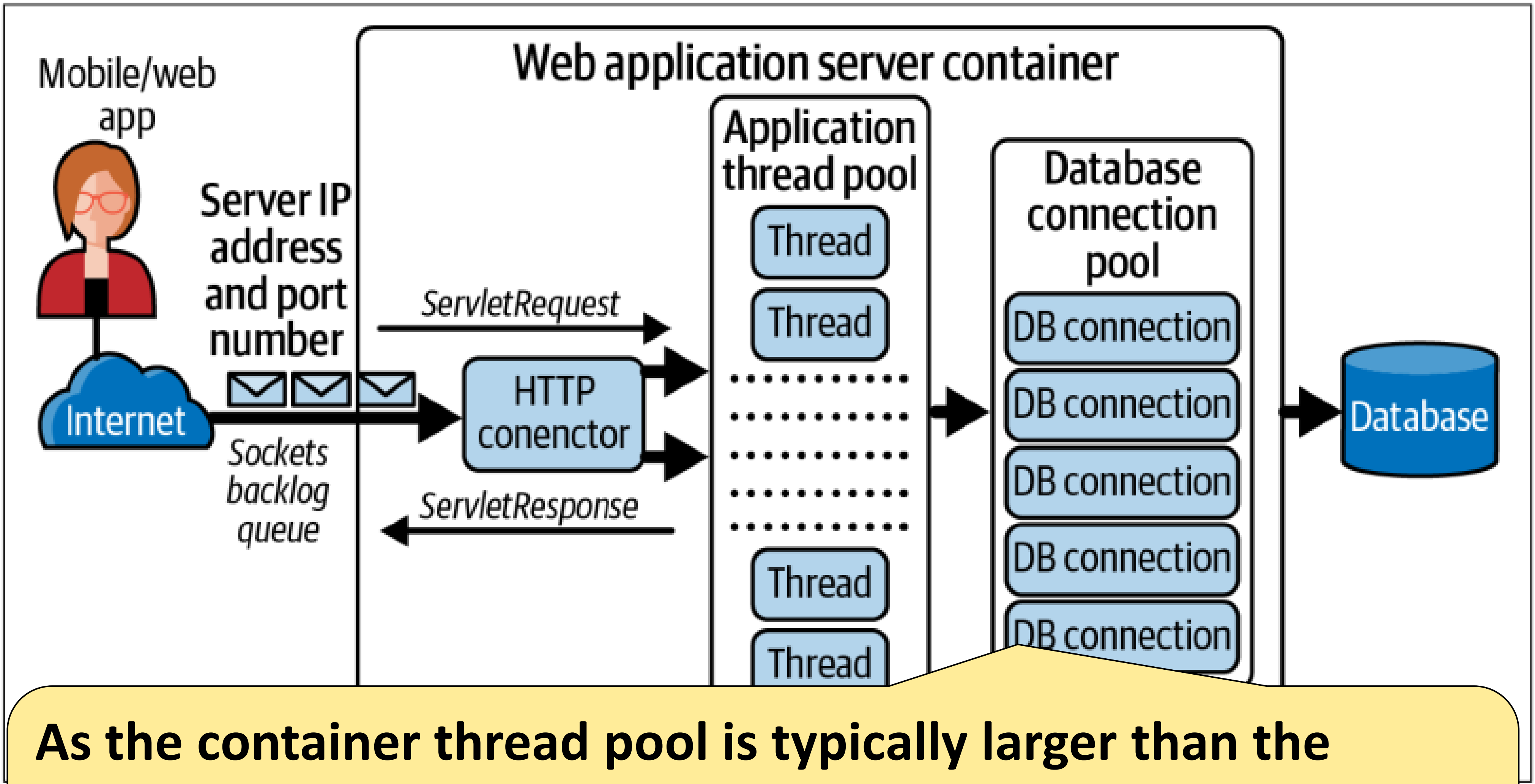
If there are no available threads to handle a request, the container maintains them in a queue of runnable tasks and dispatches these as soon as a thread becomes available

Application Servers



In processing the business logic, servlets often need to query an external database. This requires each thread executing the servlet methods to obtain a database connection and execute database queries.

Application Servers



As the container thread pool is typically larger than the database connection pool, a servlet may request a connection when none are available

Required Readings

- Chapter 5: Application Services, from the textbook: “Foundations of Scalable Systems”, Ian Gorton, O’reilly Media Inc., 2022.