REST and JSON

Objectives

After completing this lesson, you should be able to:

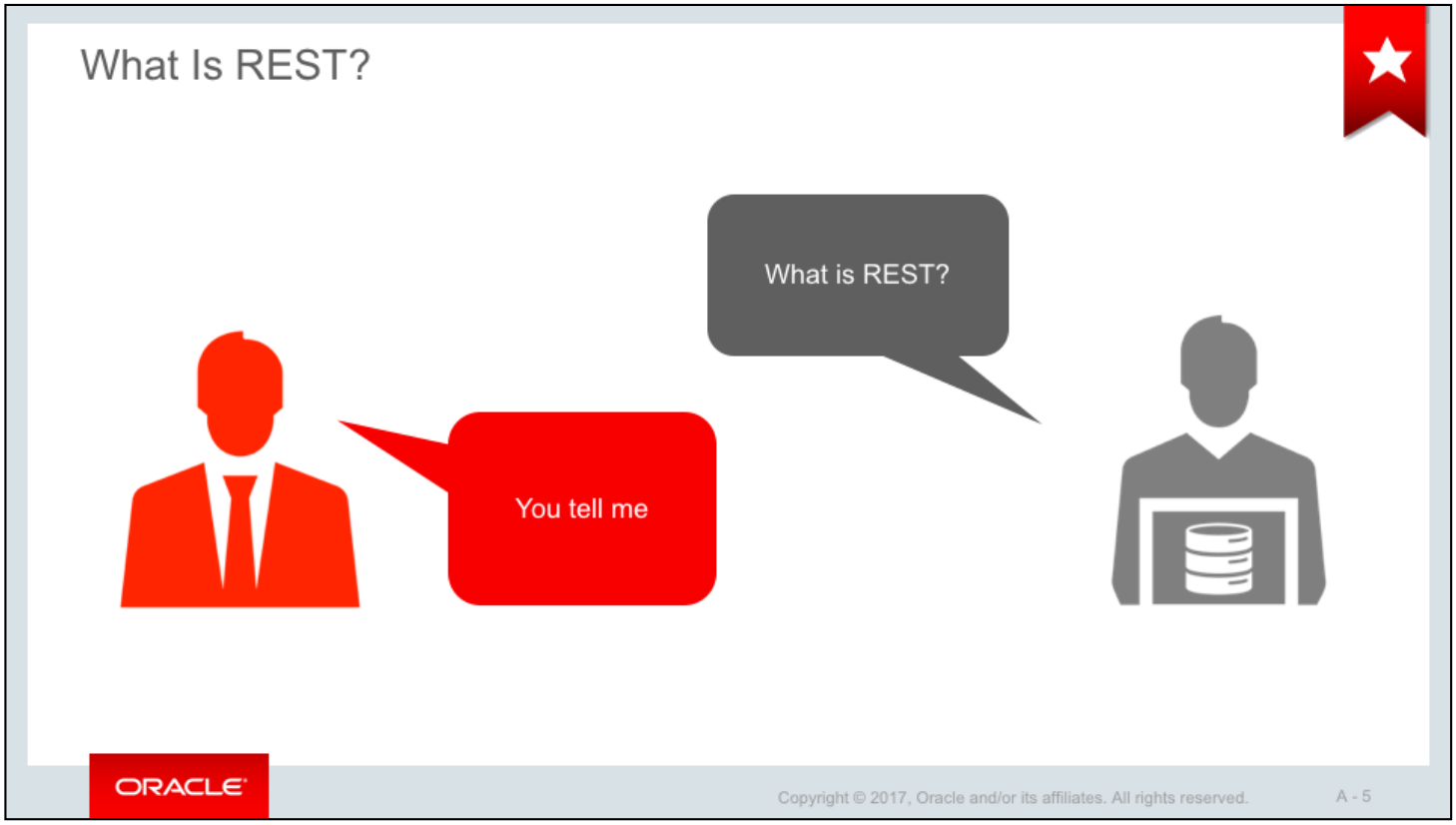
- Identify the format of REST queries
- List the differences between REST and SOAP
- Describe the basic principles of REST
- Describe the basic principles of JSON

Agenda

- REST overview
- JSON overview

Agenda

- REST overview
- JSON overview

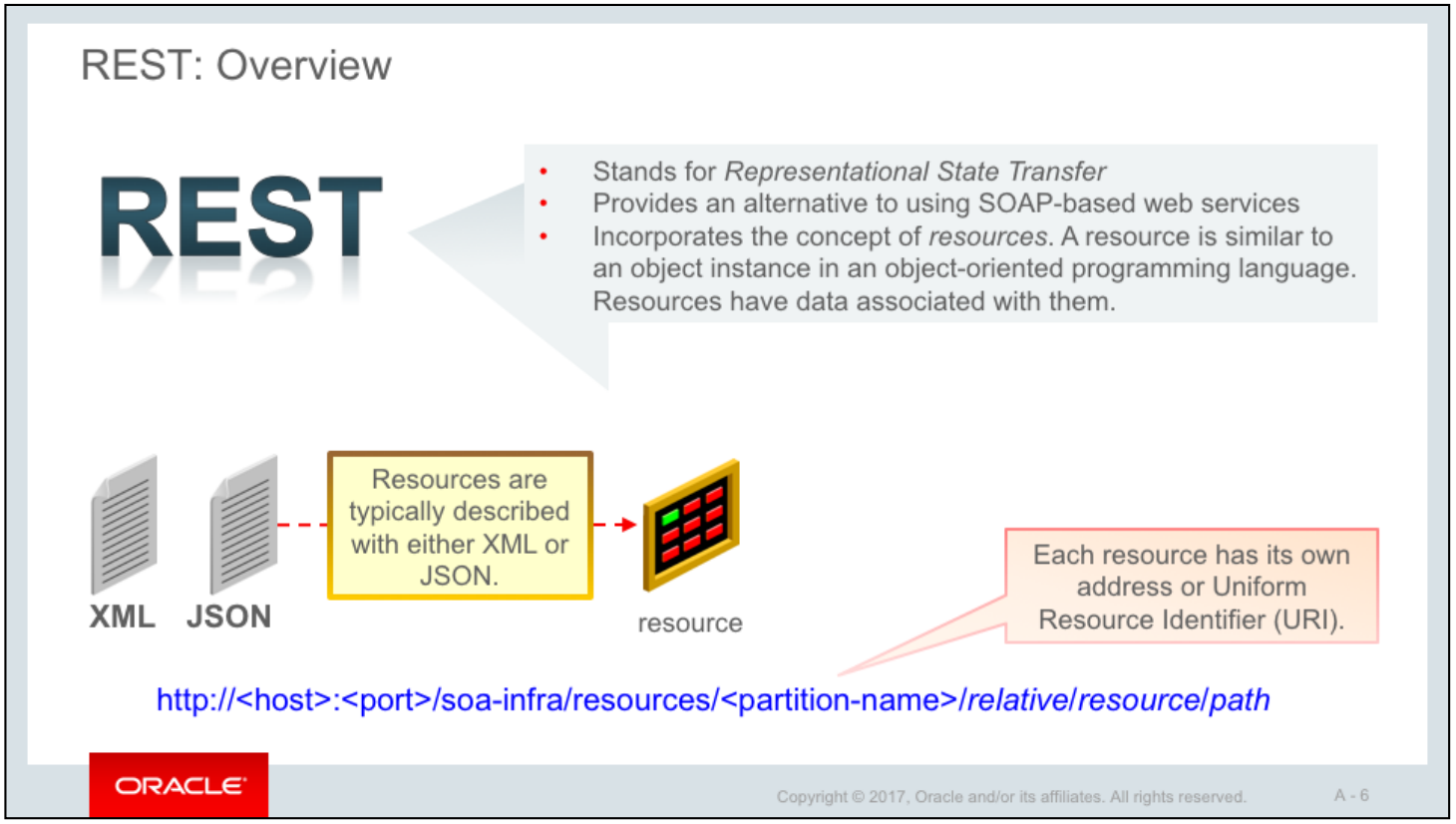


REST is focused on accessing named resources through a single consistent interface.

Many developers found SOAP cumbersome and hard to use. For example, working with SOAP in JavaScript means writing a ton of code to perform extremely simple tasks, because you must create the required XML structure absolutely every time.

REST provides a light-weight alternative. Instead of using XML to make a request, REST relies on a simple URL in many cases. In some situations you must provide additional information in special ways, but most web services using REST rely exclusively on obtaining the needed information using the URL approach. REST can use four different HTTP 1.1 verbs (GET, POST, PUT, and DELETE) to perform tasks.

Unlike SOAP, REST doesn't have to use XML to provide the response. You can find REST-based web services that output the data in comma-separated values (CSV), JavaScript Object Notation (JSON), and Really Simple Syndication (RSS). The point is that you can obtain the output you need in a form that's easy to parse within the language you need for your application.



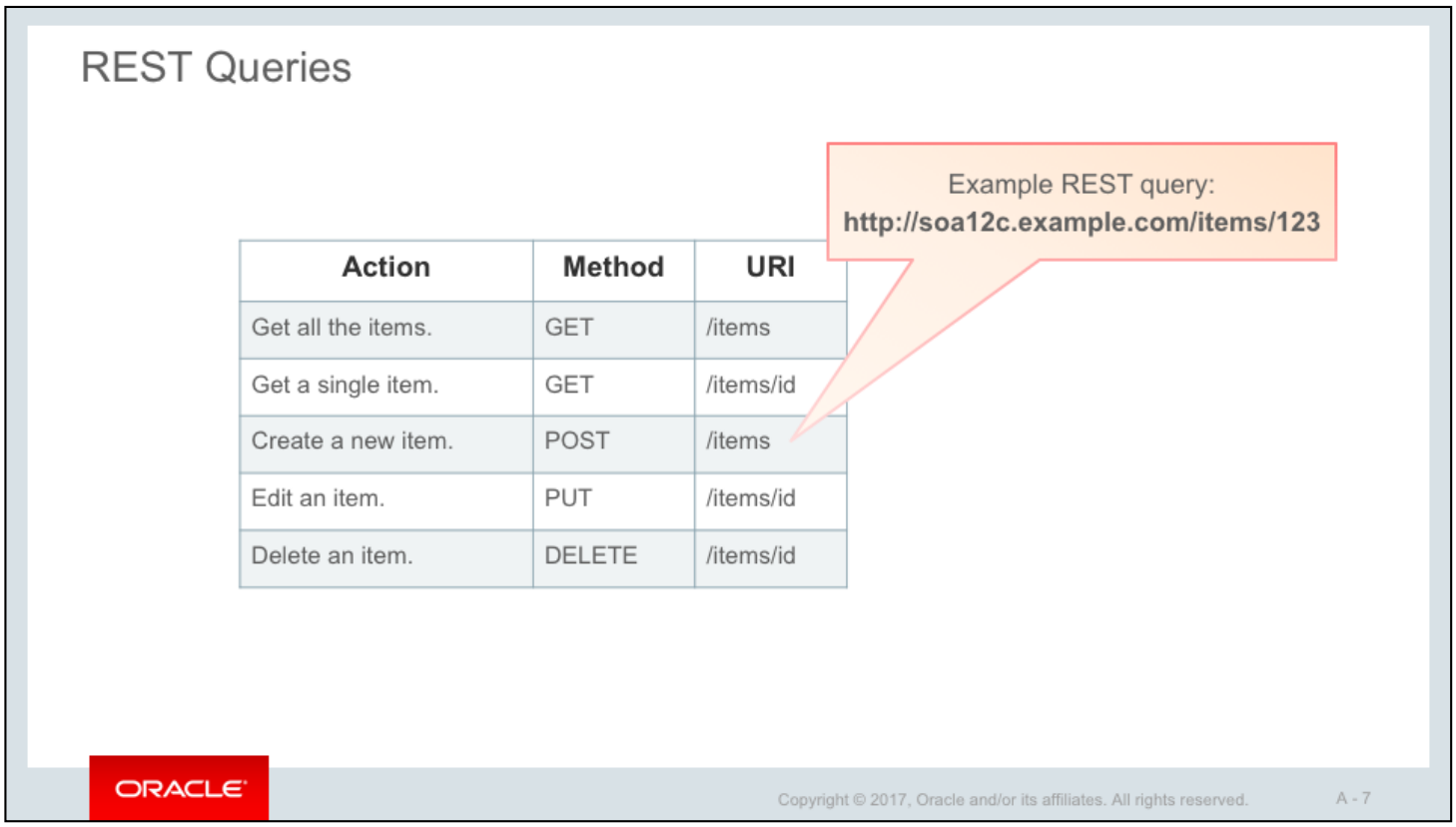
REST stands for Representational State Transfer. REST provides an alternative to using SOAP-based web services.

It incorporates the concept of *resources*. A resource is similar to an object instance in an object-oriented programming language. Resources have data associated with them. The underlying concept is that when you send a request for a resource, the resource that is returned shows you the options for the next step. This model is very flexible, and does not require that the client know in advance what services and service calls are available.

REST services rely on a stateless, client/server, cacheable communications protocol—and in virtually all cases, the HTTP protocol. RESTful applications use the HTTP verbs GET, PUT, POST, and DELETE to create, update, read, and delete data.

RESTful services do not impose constraints on the data format that is exchanged. Data can be formatted as XML, HTML, JavaScript Object Notation (JSON), and so on.

REST services themselves are often described with a Web Application Description Language (WADL) document.



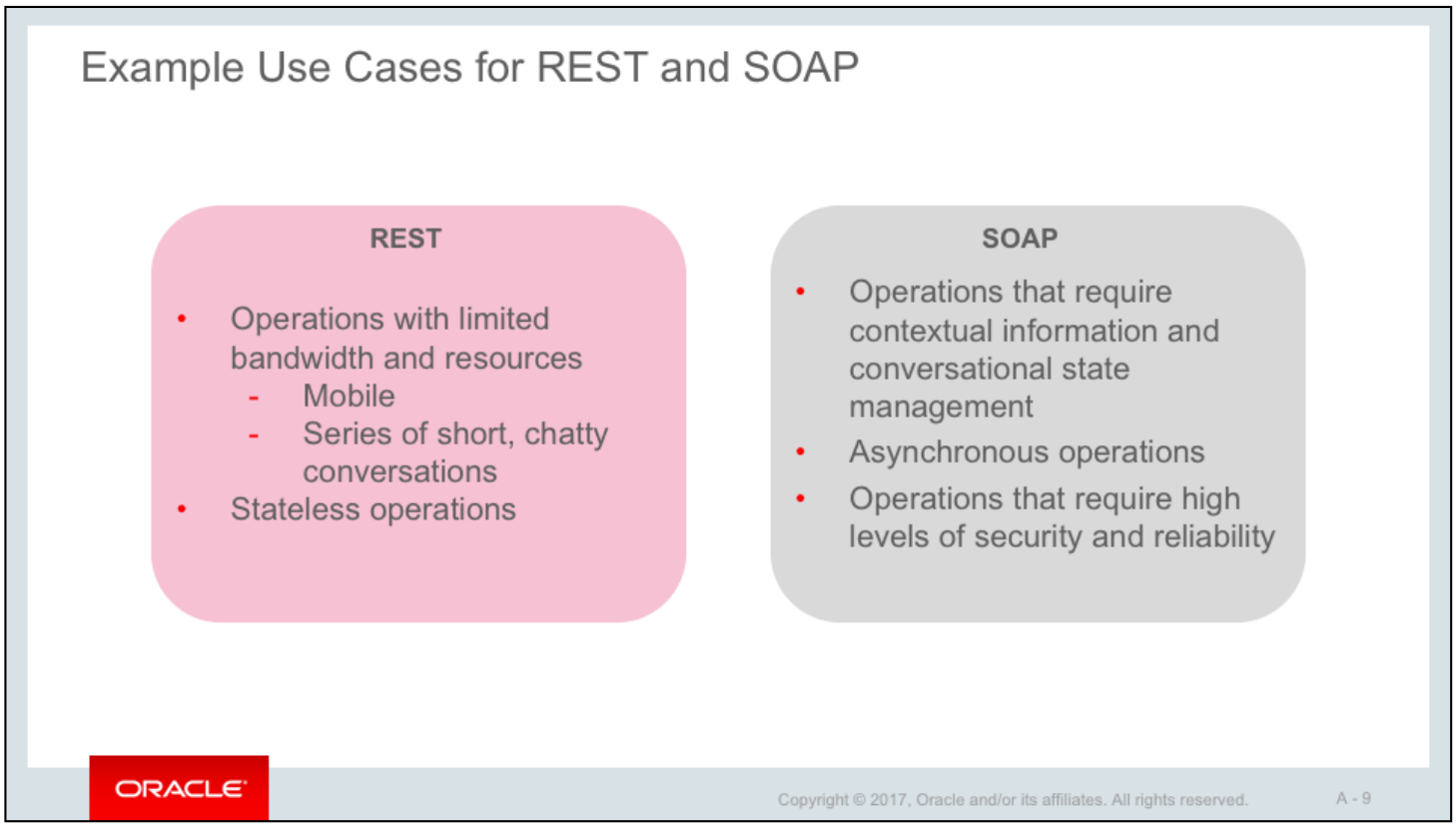
Recall that REST describes resources (reference to data) in terms of URIs and that REST services nearly always use the HTTP protocol. HTTP messages include a header and a body (although the body can be empty). HTTP request messages include methods in the header to specify an action. With this combination, REST queries use HTTP verbs and URIs to specify actions on resources (or data).

The table in the slide lists some typical actions, and the HTTP methods and URIs that might be used to accomplish them.

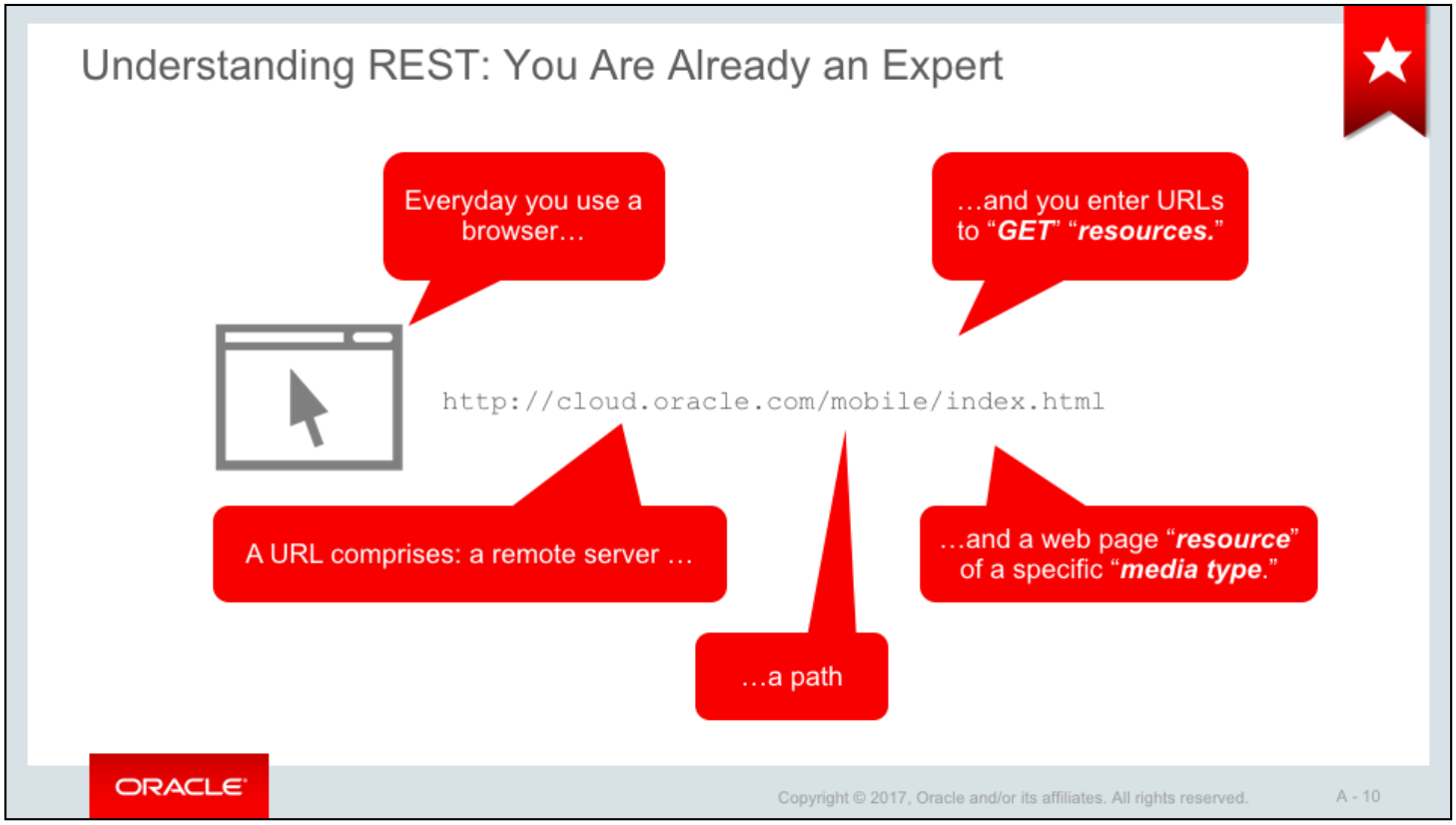
Comparing REST and SOAP: Two Ways to Access Web Services

REST	SOAP
Is an architectural style that leverages web standards	Is a formal standard for message exchange
Uses HTTP	Is protocol-independent
Permits many data formats	Uses XML
Uses URI and HTTP verbs to access *resources* (data)	Uses a WSDL document to access *operations* (business logic)
Is stateless	WS* standards provide support for stateful transactions.

REST is an architectural style, whereas SOAP is a message exchange standard. Comparing them is much like comparing apples and oranges, but in practical terms, they describe ways to access web services. As such, the table in the slide lists some features of each. These differences in features mean that in some situations, REST offers advantages, and in others, SOAP is the better choice.



For web services that require robust security, transactions, and reliability, SOAP can leverage the WS* standards, and would typically be a better choice. However, because of its use of URIs and light-weight message formats such as JSON, REST can offer better performance and scalability. REST is well-suited for situations such as mobile computing, which are constrained by limited bandwidth and resources.



Resources are linked together (hypermedia).

Design a tree of resources with a single base resource. It is comparable to an object graph in Java or hierarchy of elements in an XML document.

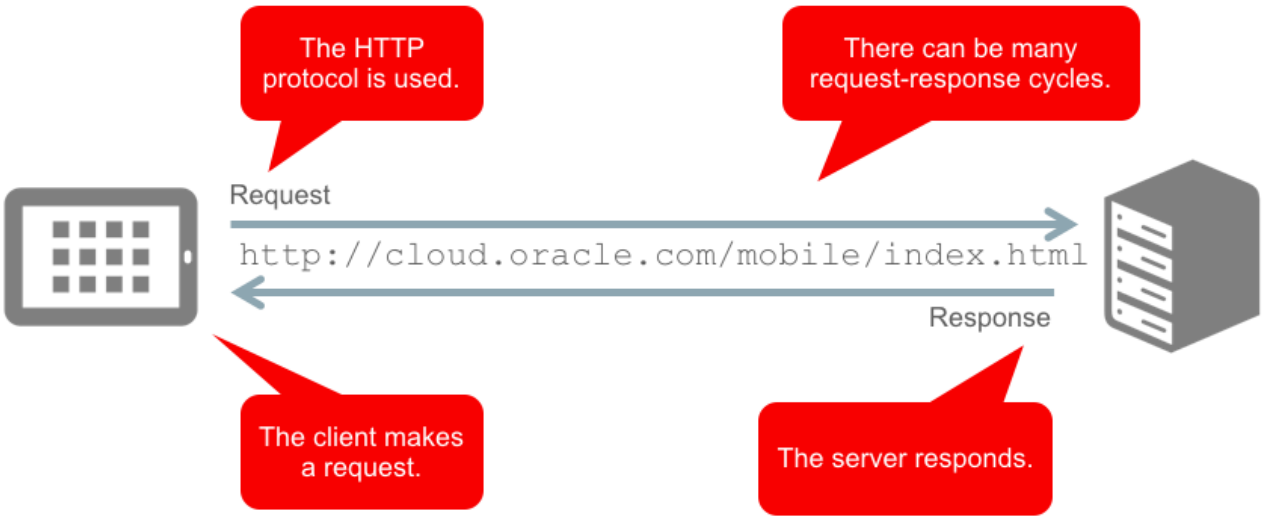
Resources are (usually) nouns or things. Each resource has a limited number of general-purpose operations that it may support (GET, PUT, POST, DELETE). A resource is uniquely identified by a URL.

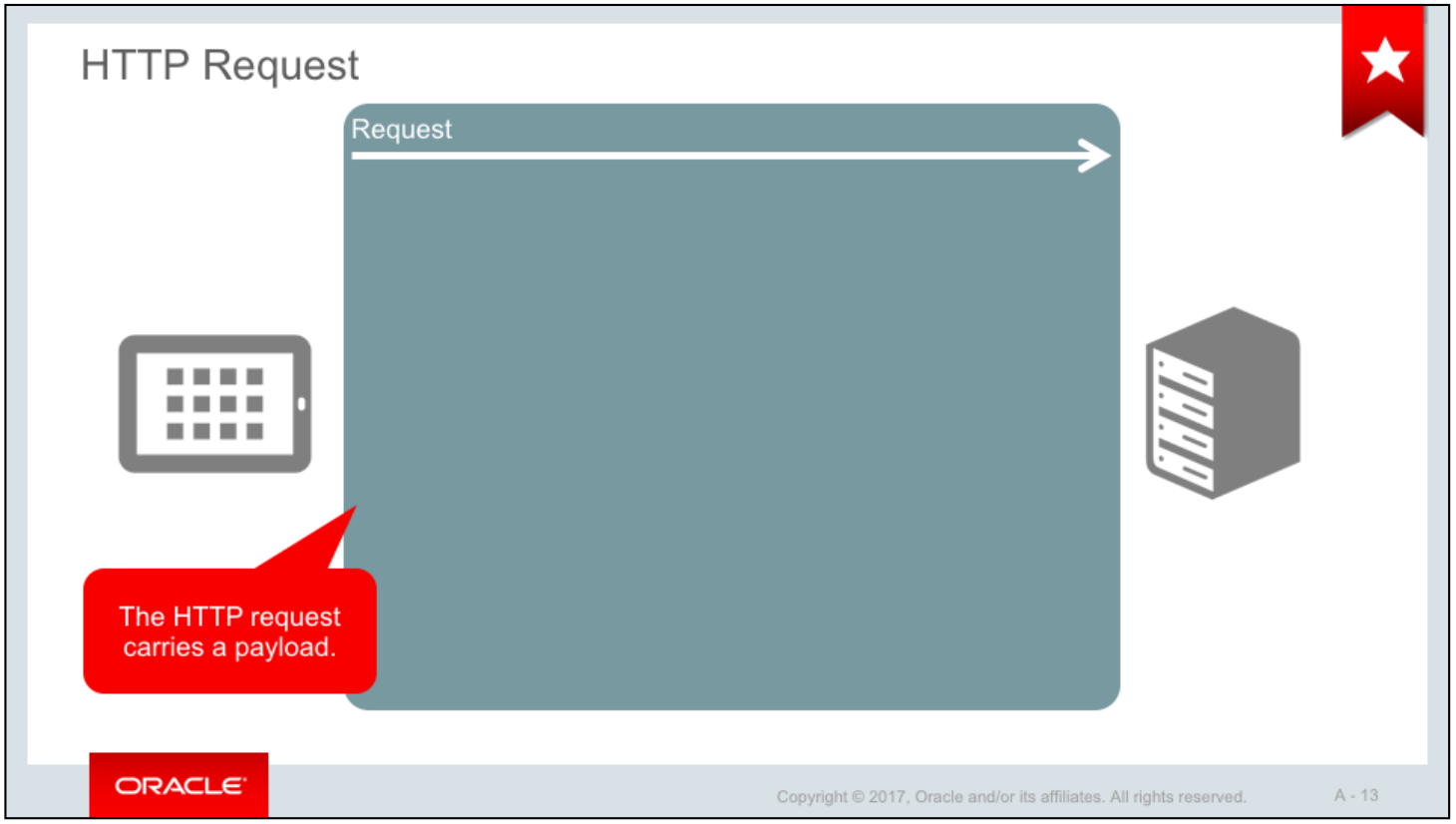
http://llocalhost:7001/myapp/resources/users

A collection is also a resource. A specific resource in a collection is located with its unique value ("roy" in this example: http://llocalhost:7001/myapp/resources/users/roy).

Understanding REST: You Are Already an Expert

Understanding REST: You Are Already an Expert



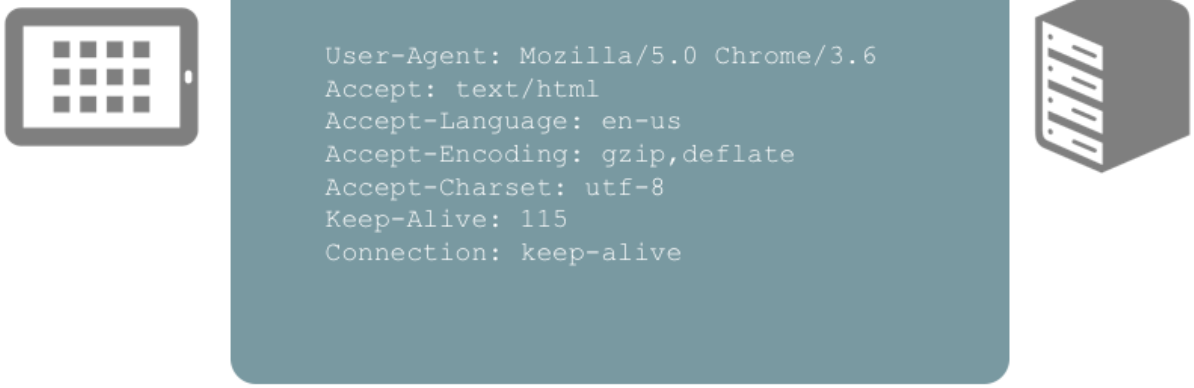

HTTP Request
Request
The HTTP request carries a payload.
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 13

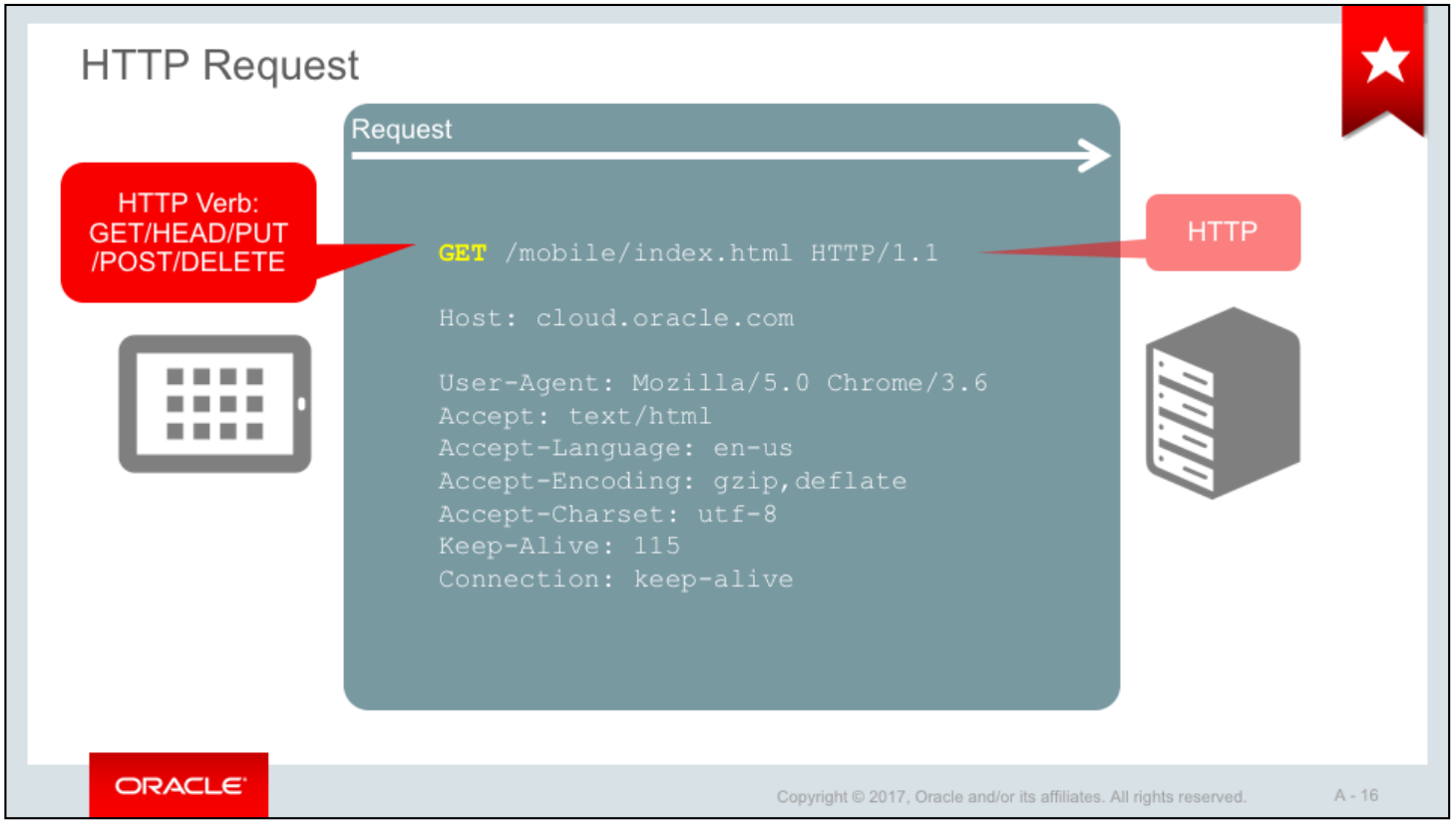
HTTP Request

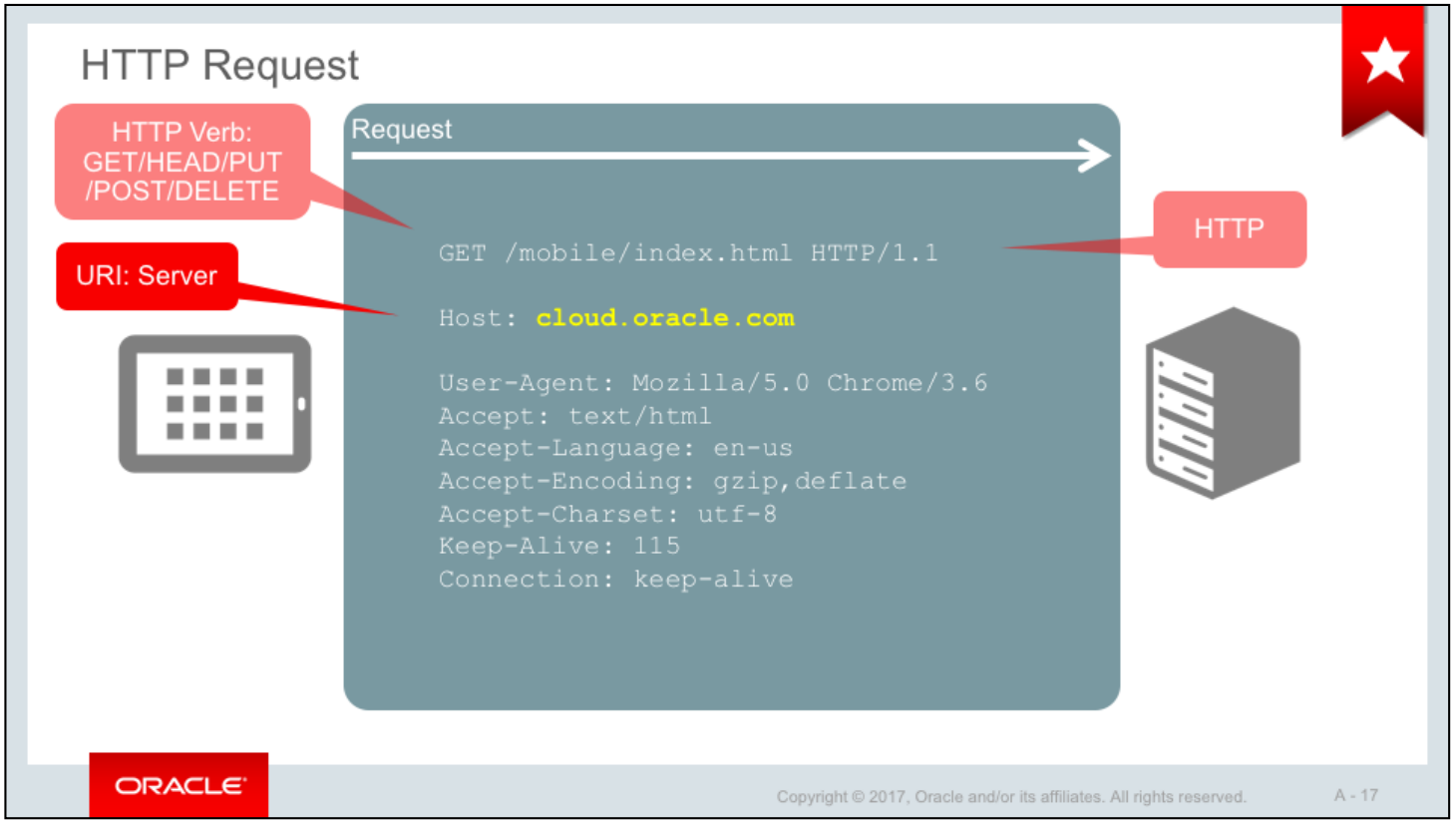
Request

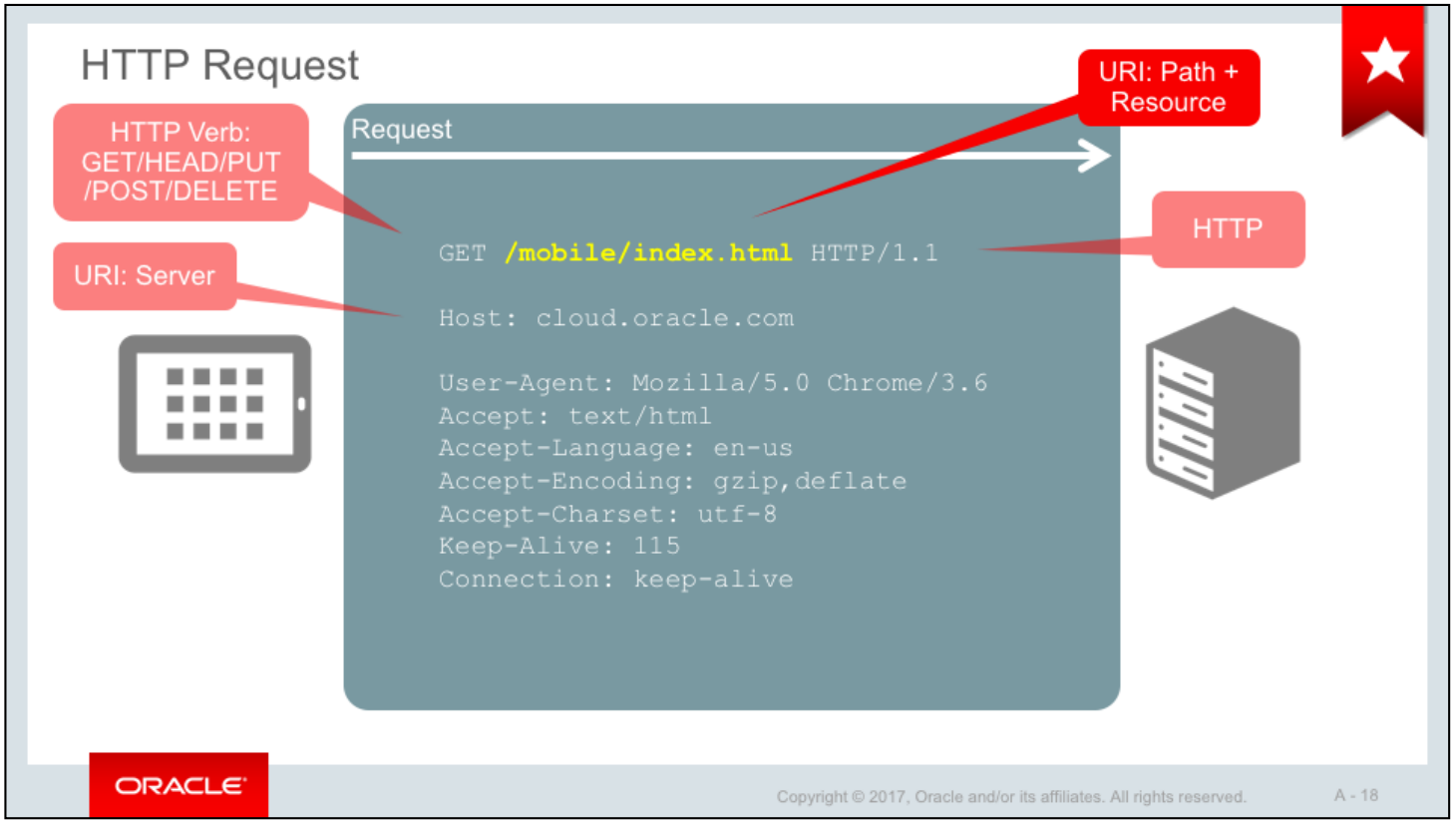
```
GET /mobile/index.html HTTP/1.1

Host: cloud.oracle.com

User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
```

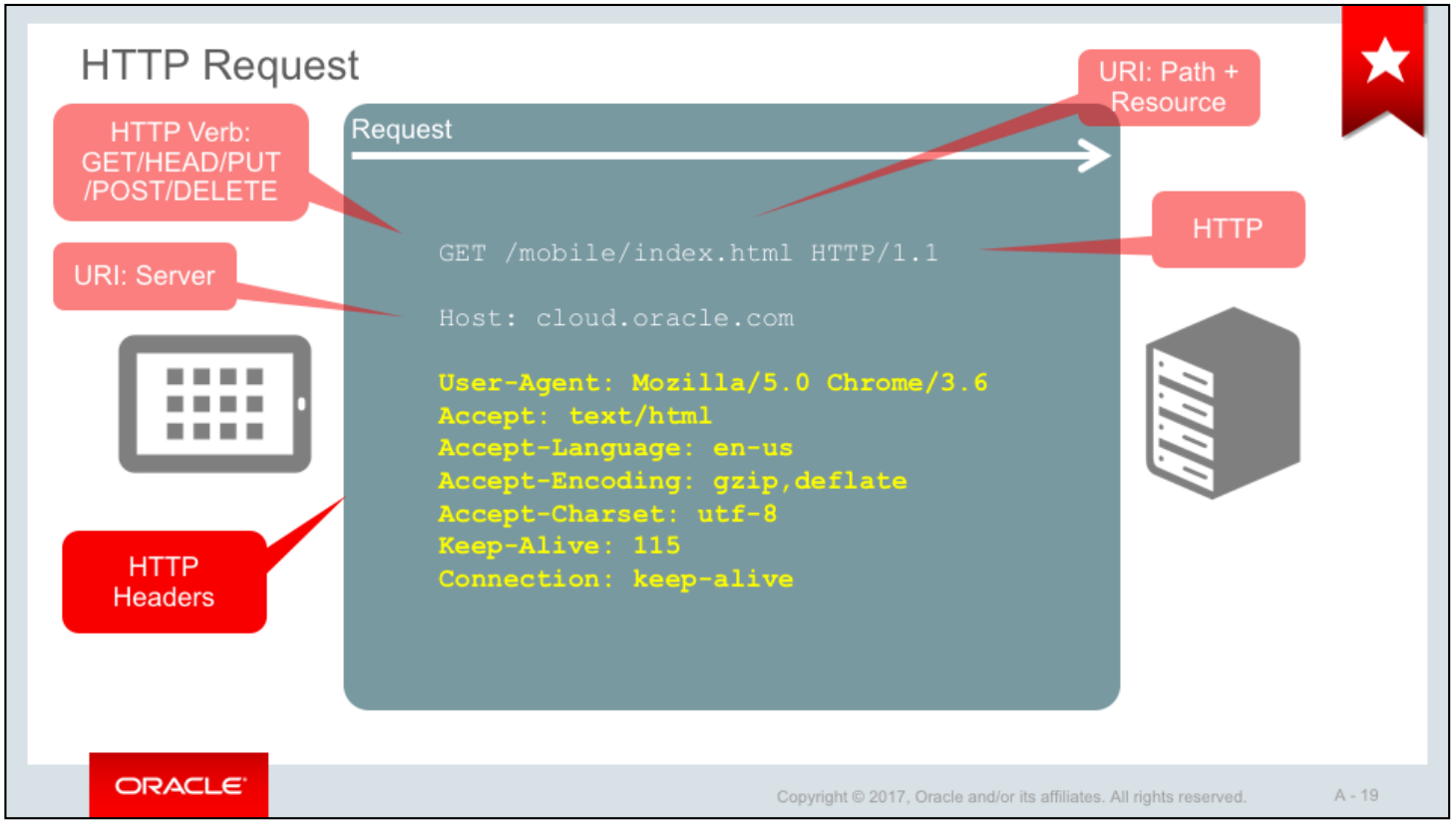
HTTP Request

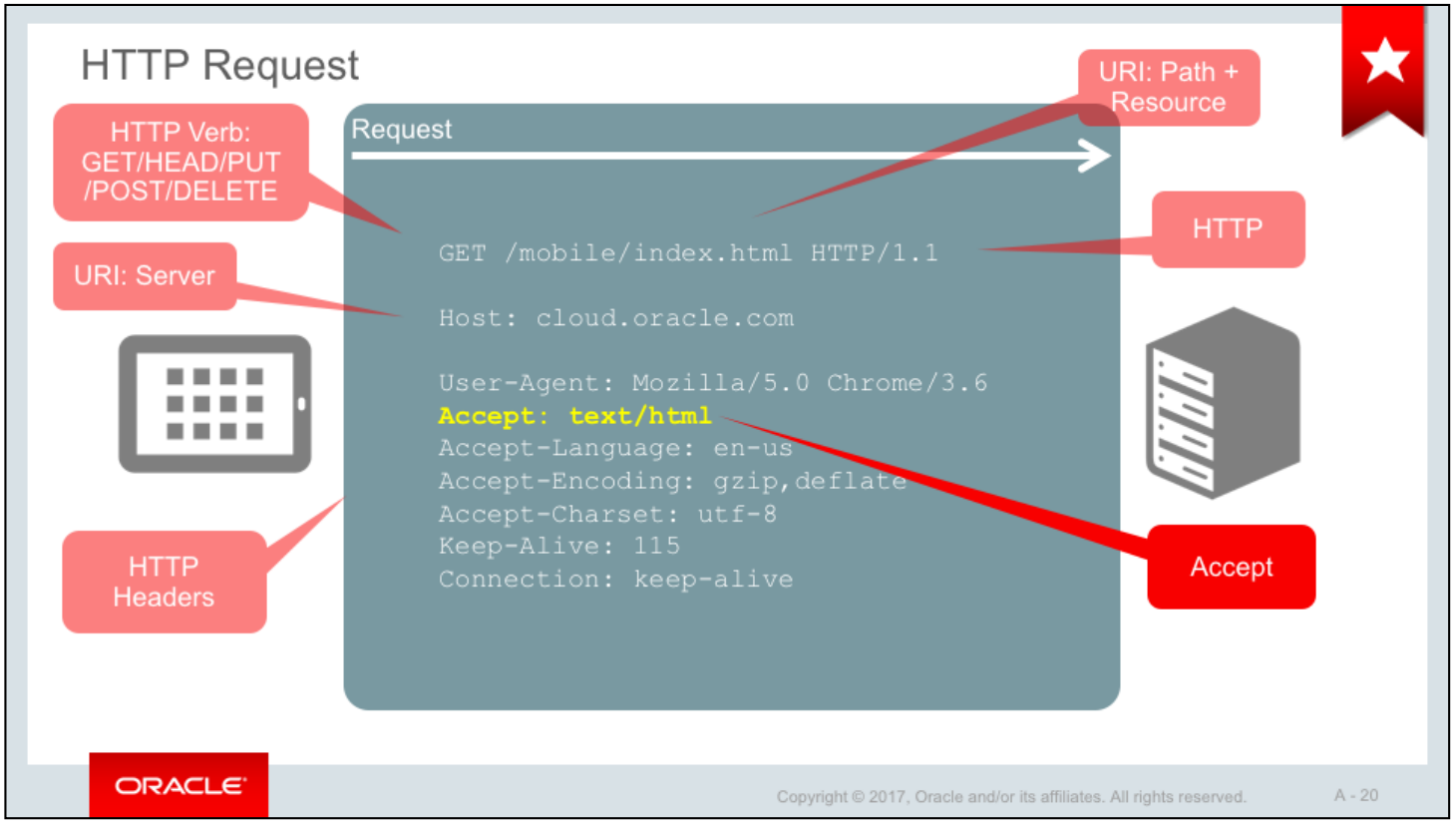


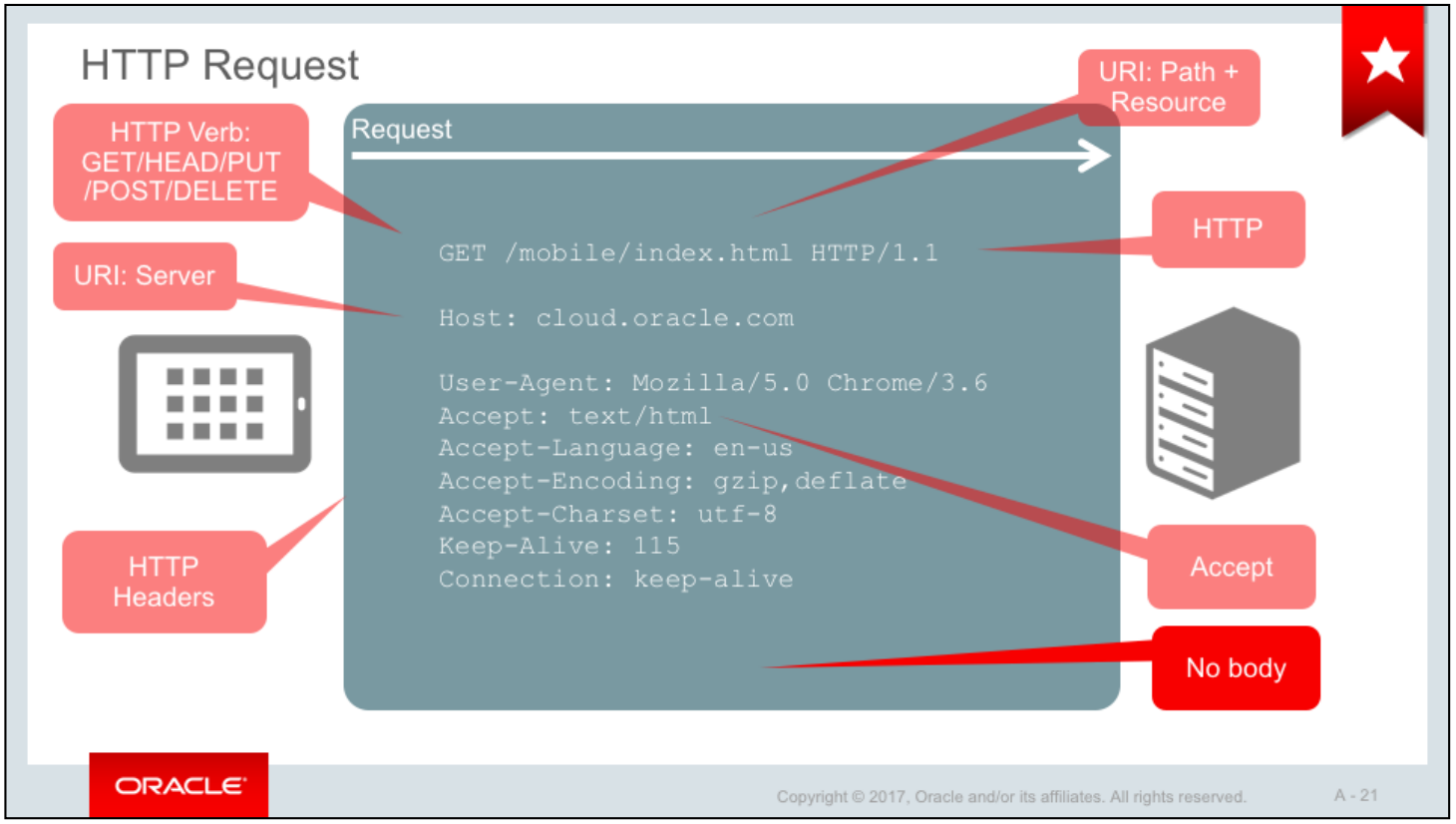

HTTP Request
Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
HTTP
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 16


HTTP Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
URI: Server
Request
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
HTTP
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 17


HTTP Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
URI: Path + Resource
Request
HTTP
URI: Server
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 18


HTTP Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
URI: Path + Resource
Request
HTTP
URI: Server
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
HTTP Headers
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 19


HTTP Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
URI: Path + Resource
Request
HTTP
URI: Server
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
HTTP Headers
Accept
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 20


HTTP Request
HTTP Verb: GET/HEAD/PUT /POST/DELETE
URI: Path + Resource
Request
HTTP
URI: Server
GET /mobile/index.html HTTP/1.1
Host: cloud.oracle.com
User-Agent: Mozilla/5.0 Chrome/3.6
Accept: text/html
Accept-Language: en-us
Accept-Encoding: gzip,deflate
Accept-Charset: utf-8
Keep-Alive: 115
Connection: keep-alive
HTTP Headers
Accept
No body
ORACLE
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.
A - 21

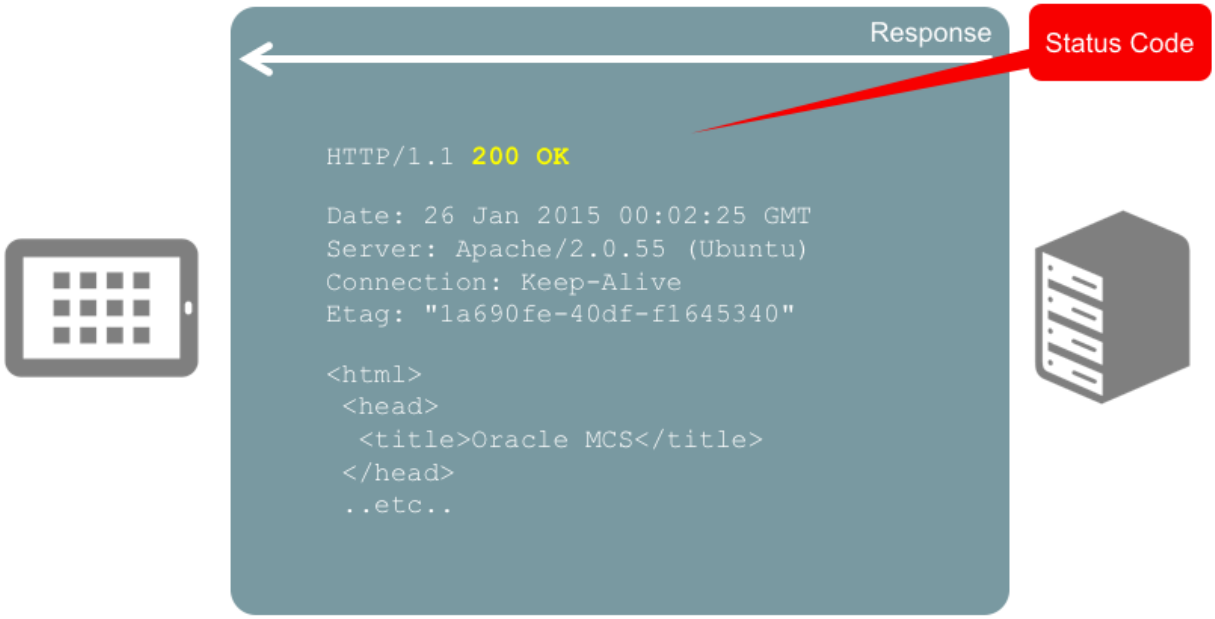
HTTP Response

HTTP Response

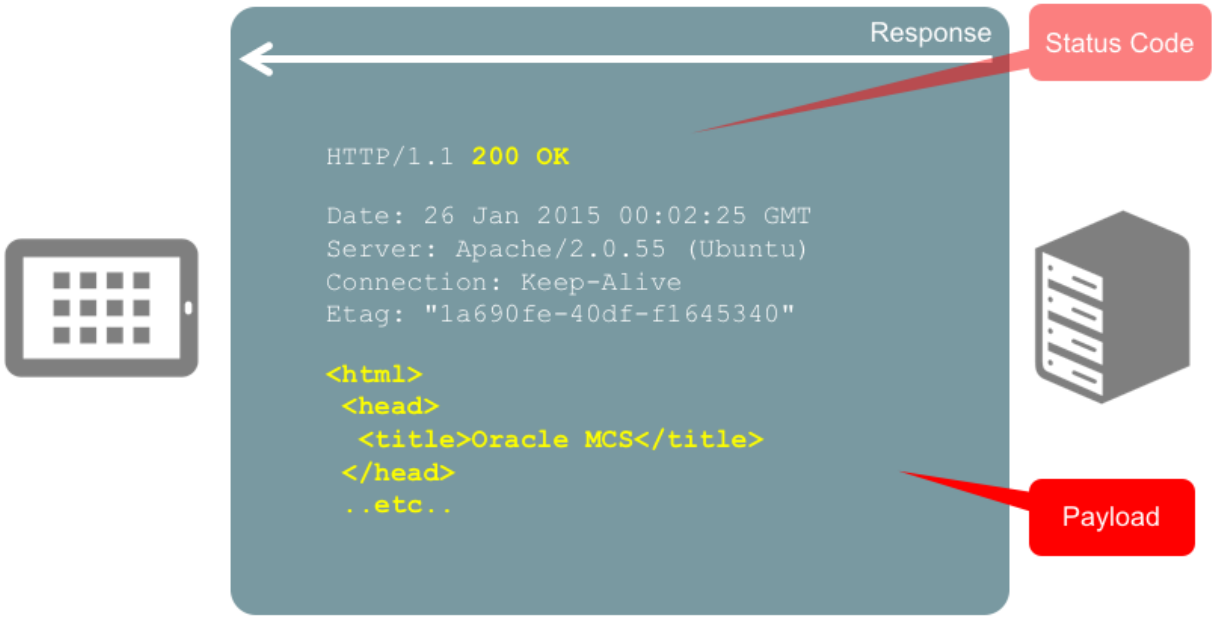
Response

```
HTTP/1.1 200 OK

Date: 26 Jan 2015 00:02:25 GMT
Server: Apache/2.0.55 (Ubuntu)
Connection: Keep-Alive
Etag: "1a690fe-40df-f1645340"

<html>
 <head>
  <title>Oracle MCS</title>
 </head>
 ..etc..
```

HTTP Response



HTTP Response



Understanding REST: The **Four** Slide Introduction

Resources

HTTP verbs

Status codes

Media types

Understanding REST: The **Four** Slide Introduction

Resources

HTTP verbs

Status codes

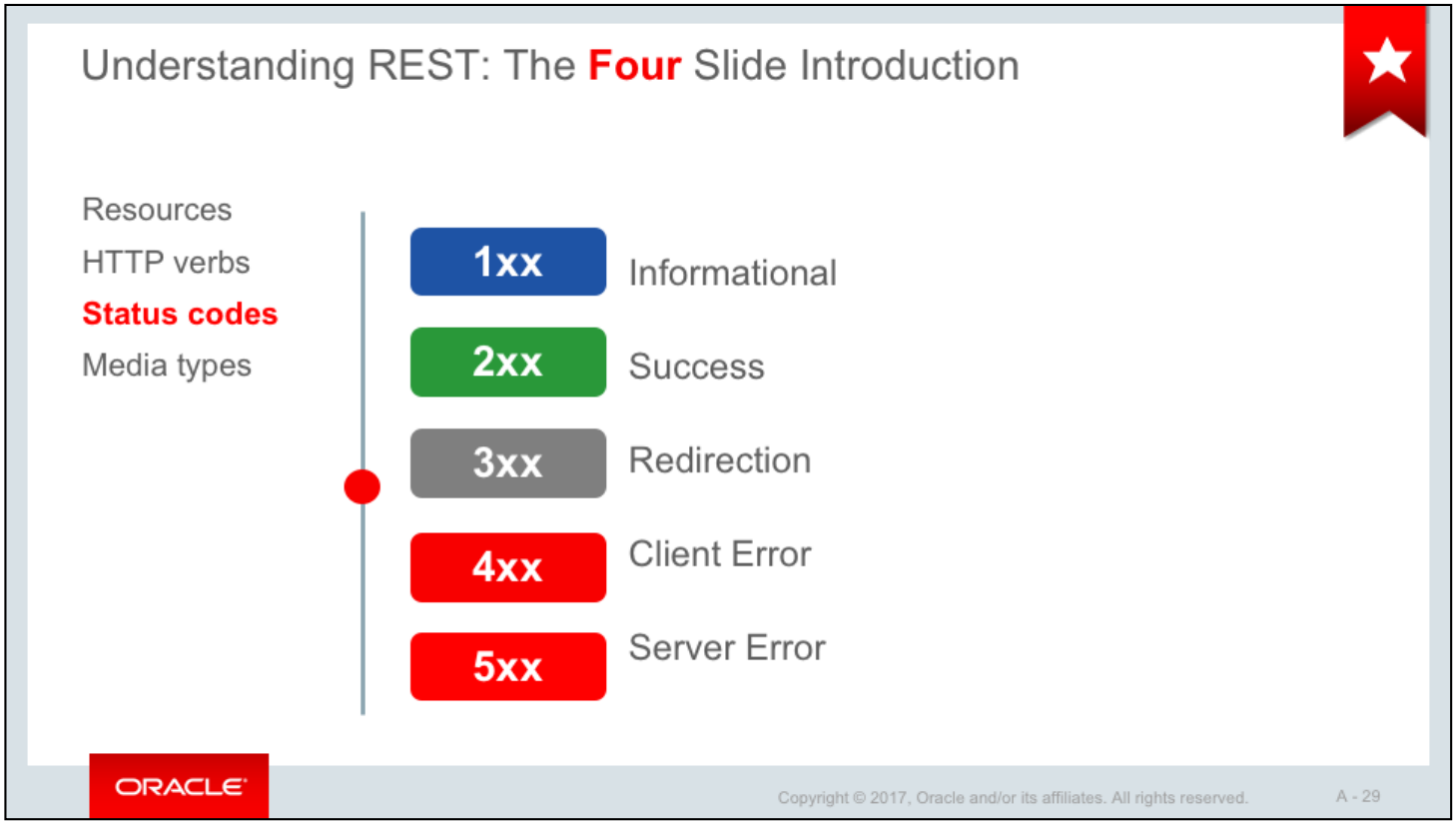
Media types

GET	http://host/path/departments
GET	http://host/path/employees
GET	http://host/path/departments/hr
GET	http://host/path/employees/101
GET	http://host/path/departments/hr/employees

ORACLE

Understanding REST: The **Four** Slide Introduction

Understanding REST: The **Four** Slide Introduction



Understanding REST: The **Four** Slide Introduction

Resources

HTTP verbs

Status codes

Media types

Payload type “requested” by client

- Via HTTP Parameter “accept”
- Defines MIME types
  - Examples: **application/json**, application/xml, image/gif

```
{"departments: [
  {"id":"MAN","name":"Manfacturing"},
  {"id":"HR","name":"Human Resources"},
  {"id":"FIN","name":"Finance"},
  ...etc...
 ]}
```

Agenda

- REST overview
- JSON overview

Understanding JSON: The **Three** Slide Introduction

```
{
  "firstName":"John",
  "lastName":"Smith",
  "isAlive":true,
  "age":25,
  "address": {
    "streetAddress":"21 2nd Street",
    "city":"New York",
    "state":"NY",
    "postalCode":"10021-3100"
  },
  "phoneNumbers": [
    {"type":"home",   "number":"1234"},
    {"type":"office","number":"4567"}
  ],
  "children":[],
  "spouse":null
}
```

"JSON is a standard using **human-readable text** to transmit data objects of attribute-value pairs. It is typically **used in machine-to-machine communications** and is a contemporary replacement for the older XML standard."

- Wikipedia

Understanding JSON: The **Three** Slide Introduction

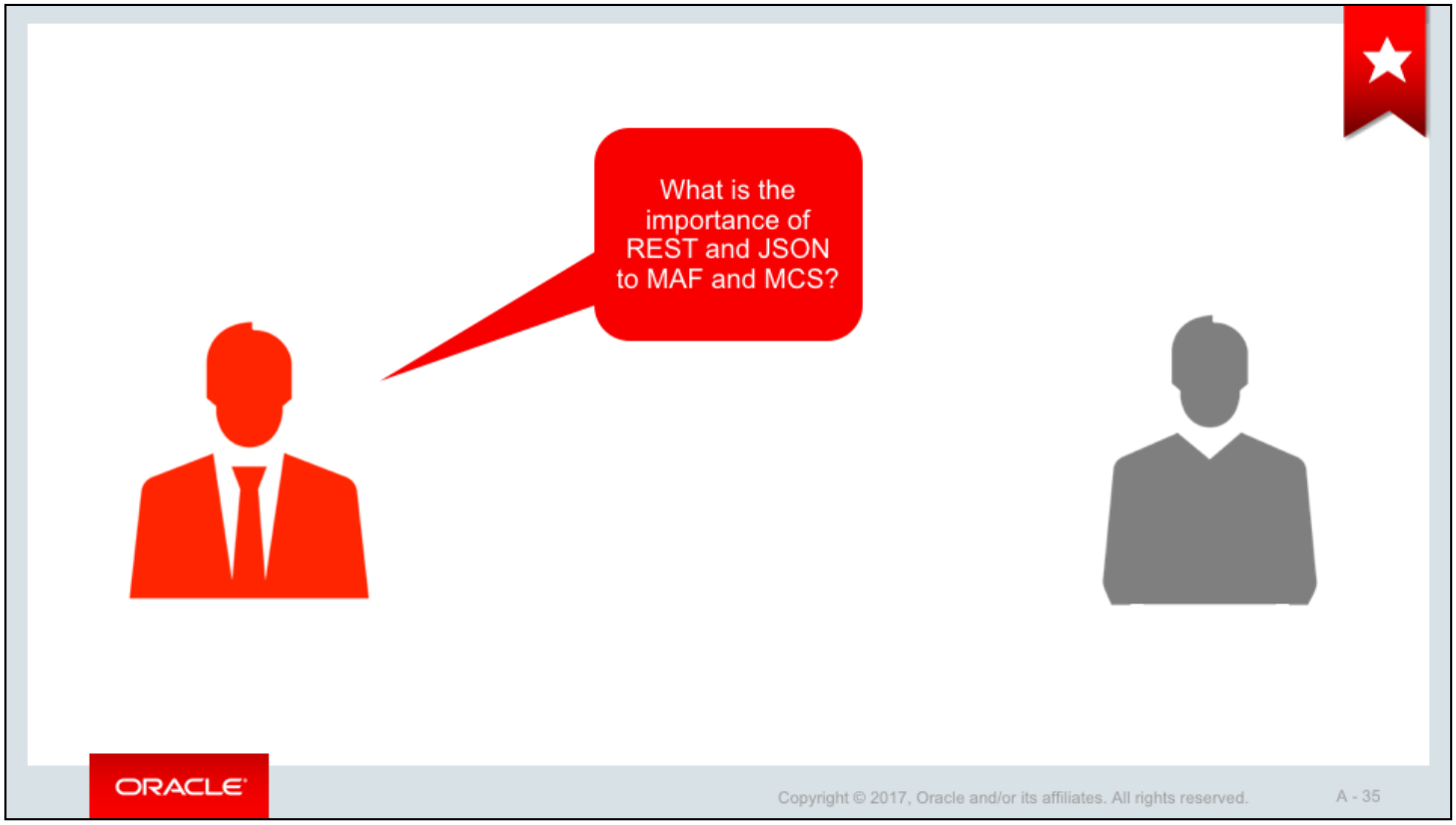
```
{
  "firstName":"John",
  "lastName":"Smith",
  "isAlive":true,
  "age":25,
  "address": {
    "streetAddress":"21 2nd Street",
    "city":"New York",
    "state":"NY",
    "postalCode":"10021-3100"
  },
  "phoneNumbers": [
    {"type":"home",   "number":"1234"},
    {"type":"office","number":"4567"}
  ],
  "children":[],
  "spouse":null
}
```

- Textual data payloads
- Human readable
- Supports validation and schemas
- “Fat free” alternative to XML
- Compact mobile friendly payloads
- JavaScript has inbuilt support.

Understanding JSON: The **Three** Slide Introduction

```
{
  "firstName":"John",
  "lastName":"Smith",
  "isAlive":true,
  "age":25,
  "address": {
    "streetAddress":"21 2nd Street",
    "city":"New York",
    "state":"NY",
    "postalCode":"10021-3100"
  },
  "phoneNumbers": [
    {"type":"home",   "number":"1234"},
    {"type":"office","number":"4567"}
  ],
  "children":[],
  "spouse":null
}
```

- Blocks delineated by ellipses
- Key-value pairs, separated by colon
  - Key with quotes
  - Strings and dates with quotes
  - Boolean, integers, null without quotes
  - Supports nesting of records
  - Supports arrays/collections
  - Empty array
  - Null values
- Elements are comma delimited.



A mobile back end is a logical grouping of custom APIs, users, storage collections, and other resources that serves as a cloud-based companion to one or more related mobile apps. Within a mobile back end, you organize and develop resources that will be used by your apps (which they access as REST web services).

Especially if your company is new to MCS, one of the first things you'll need to do is start creating your own set of REST APIs to provide building blocks for your apps. REST is an architectural style in which data and functionality are represented as resources and are referenced using Uniform Resource Identifiers (URIs). The operations for acting on those resources are standard HTTP methods (such as GET, PUT, POST, and DELETE).

API creation is divided into two parts: designing and implementing. Let's talk about designing first.

If you are a mobile app developer or a service developer, you will likely be tasked with designing APIs. When you design a REST API, you express the functionality that you expect in terms of resources, along with the HTTP methods they accept, and media types for the request and response bodies. In other words, you essentially define the formats for making a request on the API and for what kind of data is returned in the response. This definition is stored in a RAML (RESTful API Modeling Language) document. You don't actually fill in the details of how the data is produced and where it comes from. Those details are worked out later in the implementation.

You define a connector API by filling in info on the target resource, creating rules for the call parameters to "shape" the returned data so that it works well in a mobile context, and specifying security policies. The result is a reusable service that’s exposed as a straightforward REST API that you can view in the Custom Code API Catalog. Service developers can call this connector from their custom code just like they would any other API and do not have to worry about tricky specifics like security policies and identity propagation.

In addition to custom APIs, you can use MCS platform REST APIs in your apps. You can call these APIs directly from your apps and/or via the implementation code of custom APIs. You can also access many of them through MCS’s SDKs for the iOS, Android, Windows, Cordova, and JavaScript platforms.

After you have selected the custom APIs to use in your mobile backend, you can call their REST endpoints from your mobile app code. Platform APIs are automatically available for all mobile backends, but calling them works the same way as calling custom APIs.

Summary

In this lesson, you should have learned how to:

- Identify the format of REST queries
- List the differences between REST and SOAP
- Describe the basic principles of REST
- Describe the basic principles of JSON