Internet of Things: Protocols and Networks

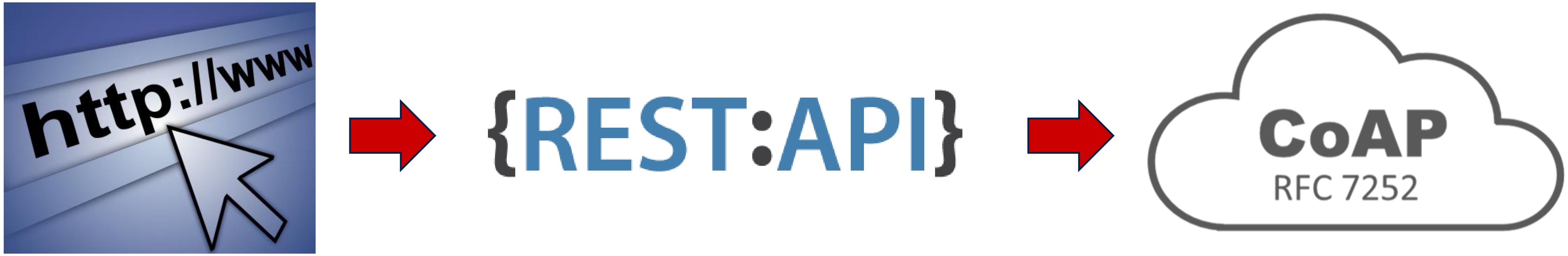
(CSC2106)

# HTTP & REST

## Recent Protocols for IoT

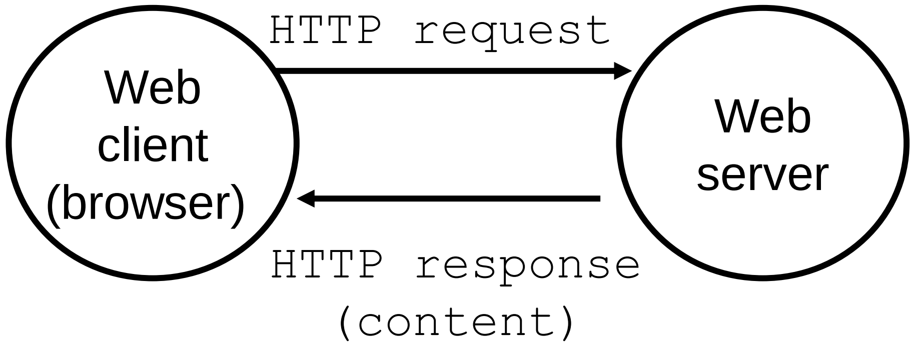
| | | | |
|---|---|---|---|
| Session | **MQTT**, SMQTT, CoRE, DDS, AMQP , XMPP, CoAP, **HTTP**, **REST**, IEC,… | **Security** | **Management** |
| | | | |
| Network | Encapsulation **6LowPAN**, 6TiSCH, 6Lo, Thread… | IEEE 1888.3, TCG, Oath 2.0, SMACK, SASL, EDSA, ace, DTLS, Dice, … | IEEE 1905, IEEE 1451, IEEE 1377, IEEE P1828, IEEE P1856 |
| | Routing RPL, CORPL, CARP | | |
| Datalink | **Wi-Fi**, 802.11ah, **Bluetooth Low Energy**, **Z-Wave, ZigBee Smart**, DECT/ULE, 3G/LTE, NFC, Weightless, HomePlug GP, 802.15.4e, G.9959, WirelessHART, DASH7, ANT+, LTE-A, **LoRaWAN**, ISA100.11a, DigiMesh, WiMAX, **NB-IoT, SigFox** … | | |

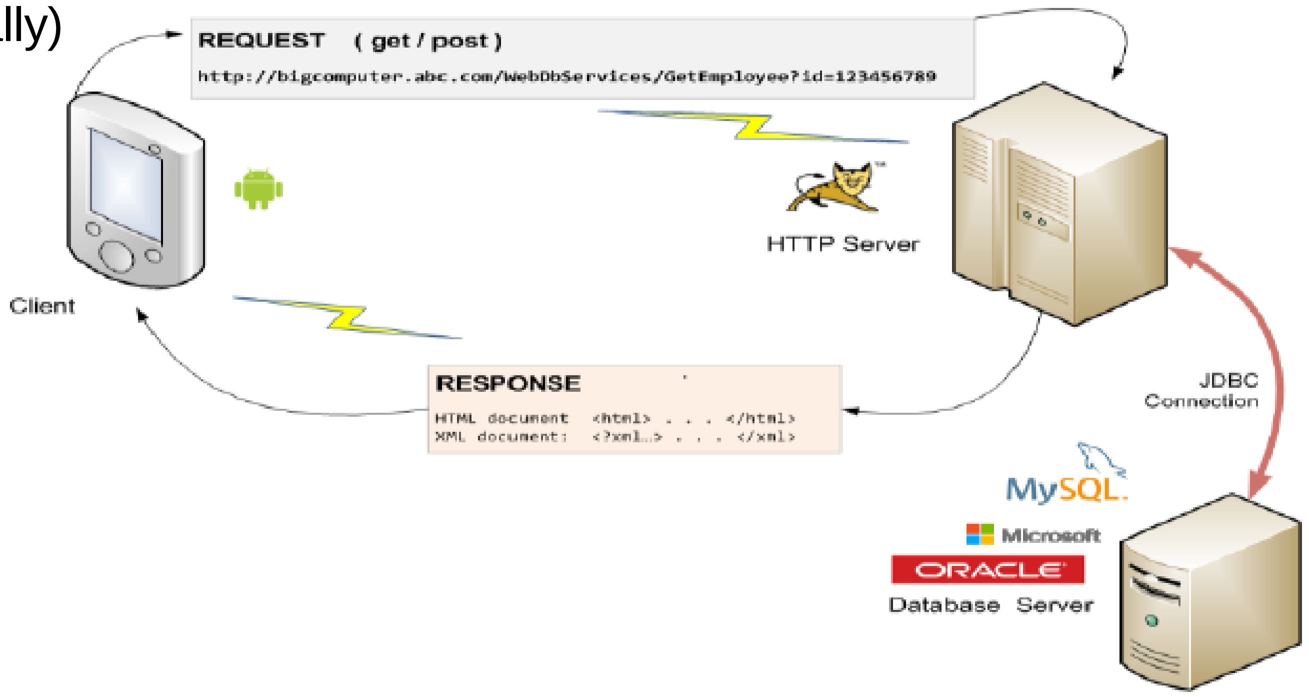
## Overview



# HyperText Transfer Protocol (HTTP)

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (usually)




REQUEST ( get / post )
http://bigcomputer.abc.com/WebDbServices/GetEmployee?id=123456789
HTTP Server
Client
RESPONSE
HTML document <html> . . . </html>
XML document: <?xml..> . . . </xml>
JDBC Connection
MySQL
Microsoft
ORACLE
Database Server

## Web Content

- Web servers return **content** to clients
  - a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type
- Example MIME types

| | |
|---|---|
| `text/html` | HTML document |
| `text/plain` | Unformatted text |
| `application/postscript` | Postcript document |
| `image/gif` | Binary image encoded in GIF format |
| `image/jpeg` | Binary image encoded in JPEG format |

## Static & Dynamic Content

- The content returned in HTTP responses can be either static or dynamic
  - **Static content**: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
  - **Dynamic content**: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
- **Bottom line**: all web content is associated with a *file* that is managed by the server

## Universal Resource Locator (URL)

- Each *file* managed by a server has a unique name called a *URL*
- URLs for static content:
  - `http://www.singaporetech.edu.sg:80/index.html`
  - `http://www.singaporetech.edu.sg/index.html`
  - `http://www.singaporetech.edu.sg`
    - Identifies a *file* called `index.html`, managed by a web server at `www.singaporetech.edu.sg` that is listening on port 80
- URLs for dynamic content:
  - `http://www.singaporetech.edu.sg:8080/cgi-bin/simi?15432&223`
    - Identifies an executable *file* called `simi`, managed by a web server at `www.singaporetech.edu.sg` that is listening on port 8080, that should be called with two argument strings: `15432` and `223`

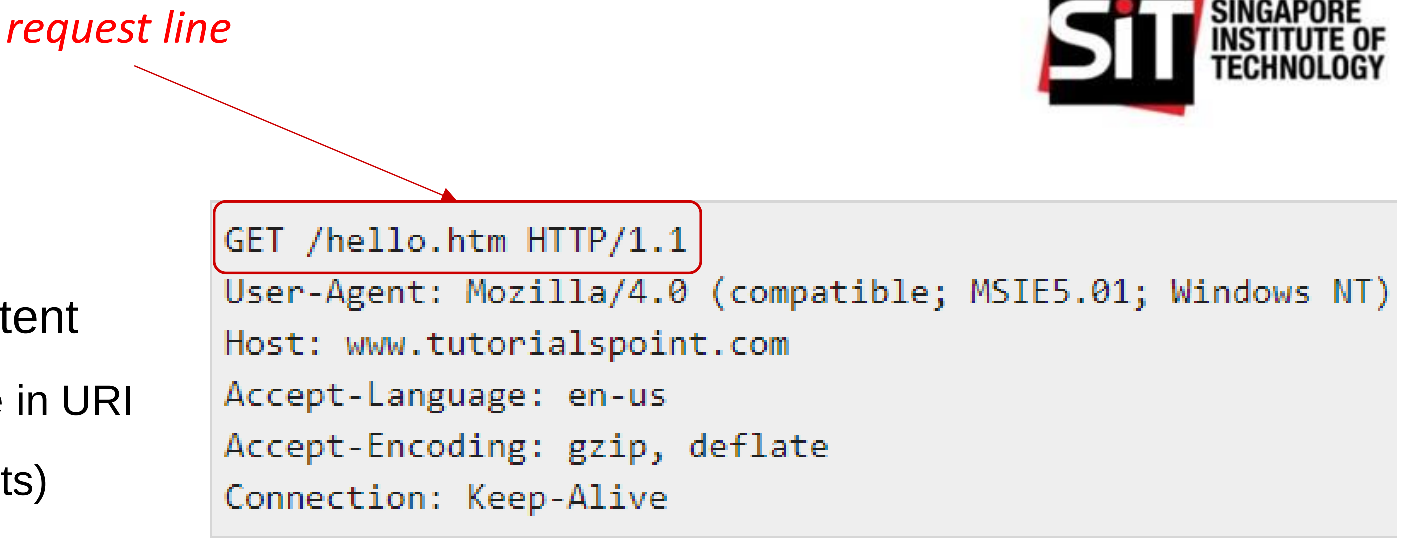
## HTTP Requests

- HTTP request is a *request line*, followed by zero or more *request headers*
- Request line: `<method> <uri> <version>`
  - `<version>` is HTTP version of request (`HTTP/1.0` or `HTTP/1.1`)
  - `<uri>` is typically URL for proxies and servers
  - `<method>` is either `GET`, `POST`, `OPTIONS`, `HEAD`, `PUT`, `DELETE`, or `TRACE`

A **URI** is an **identifier** of a specific resource. Like a page, or book, or a document. A **URL** is **special type of identifier** that also tells you how to access it, such as HTTPs , FTP , etc. .

## HTTP Requests

- HTTP methods:
  - `GET`: Retrieve static or dynamic content
    - Arguments for dynamic content are in URI
    - Workhorse method (99% of requests)
  - `POST`: Retrieve dynamic content
    - Arguments for dynamic content are in the request body
  - `OPTIONS`: Get server or file attributes
  - `HEAD`: Like `GET` but no data in response body
  - `PUT`: Write a file to the server
  - `DELETE`: Delete a file on the server
  - `TRACE`: Echo request in response body
    - Useful for debugging

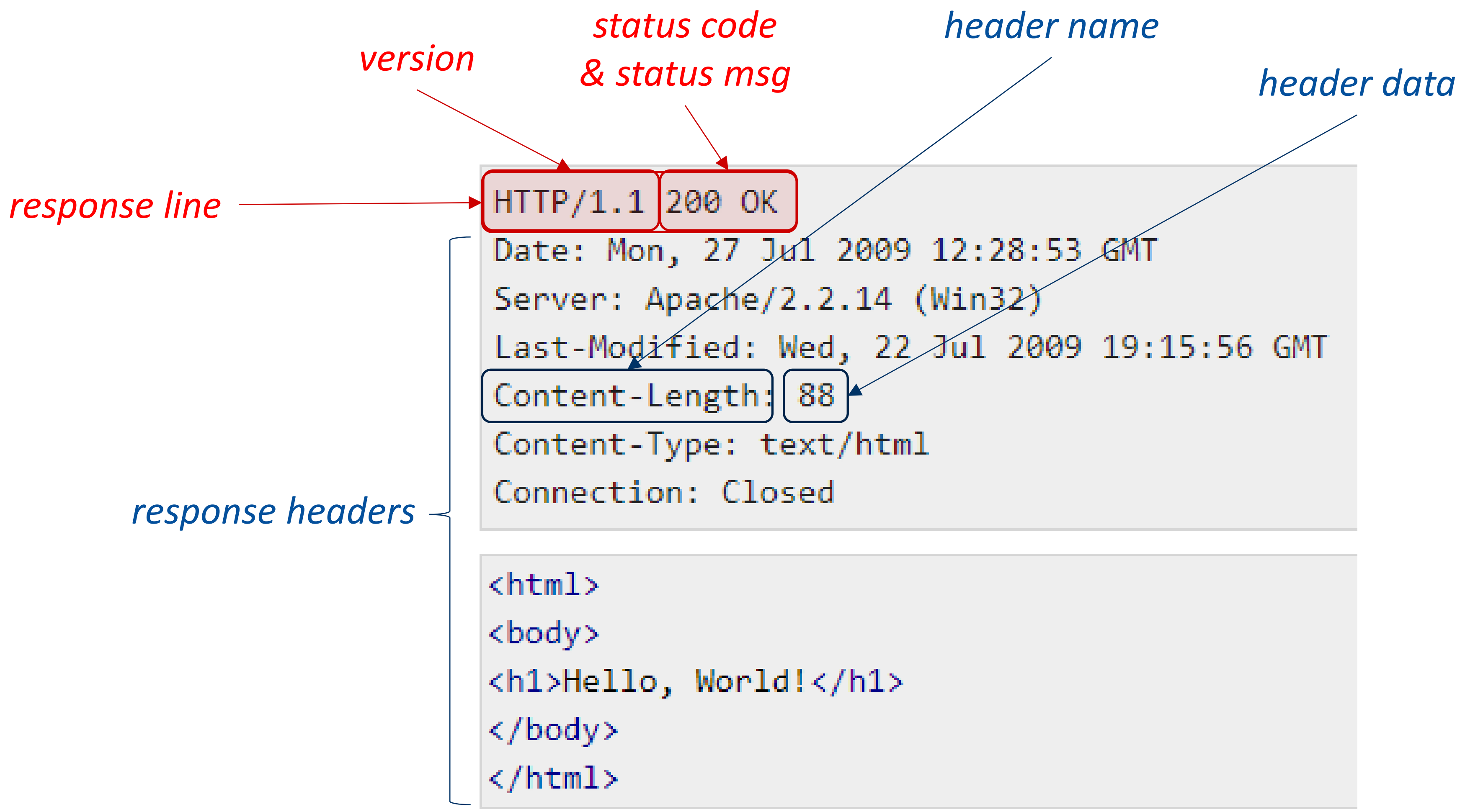


```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

## HTTP Responses

- HTTP response is a *response line* followed by zero or more *response headers*
- Response line:
- `<version> <status code> <status msg>`
  - <version> is HTTP version of the response
  - <status code> is numeric status
  - <status msg> is corresponding English text
    - 200 OK - Request was handled without error
    - 403 ForbiddenServer lacks permission to access file
    - 404 Not found Server couldn't find the file
- Response headers: `<header name>: <header data>`
  - Provide additional information about response
  - `Content-Type:` MIME type of content in response body
  - `Content-Length:` Length of content in response body

## HTTP Responses (example)



```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

# Example of an HTTP Transaction (via telnet)

```
unix> telnet www.aol.com 80          Client: open connection to server
Trying 205.188.146.23...             Telnet prints 3 lines to the terminal
Connected to aol.com.
Escape character is '^]'.
GET / HTTP/1.1                       Client: request line
host: www.aol.com                    Client: required HTTP/1.1 HOST header
                                     Client: empty line terminates headers.
HTTP/1.0 200 OK                      Server: response line
MIME-Version: 1.0                    Server: followed by five response headers
Date: Mon, 08 Jan 2001 04:59:42 GMT
Server: NaviServer/2.0 AOLserver/2.3.3
Content-Type: text/html              Server: expect HTML in the response body
Content-Length: 42092                Server: expect 42,092 bytes in the resp body
                                               Server: empty line (“\r\n”) terminates hdrs
<html>                               Server: first HTML line in response body
...                                  Server: 766 lines of HTML not shown.
</html>                              Server: last HTML line in response body
Connection closed by foreign host.   Server: closes connection
unix>                                Client: closes connection and terminates
```

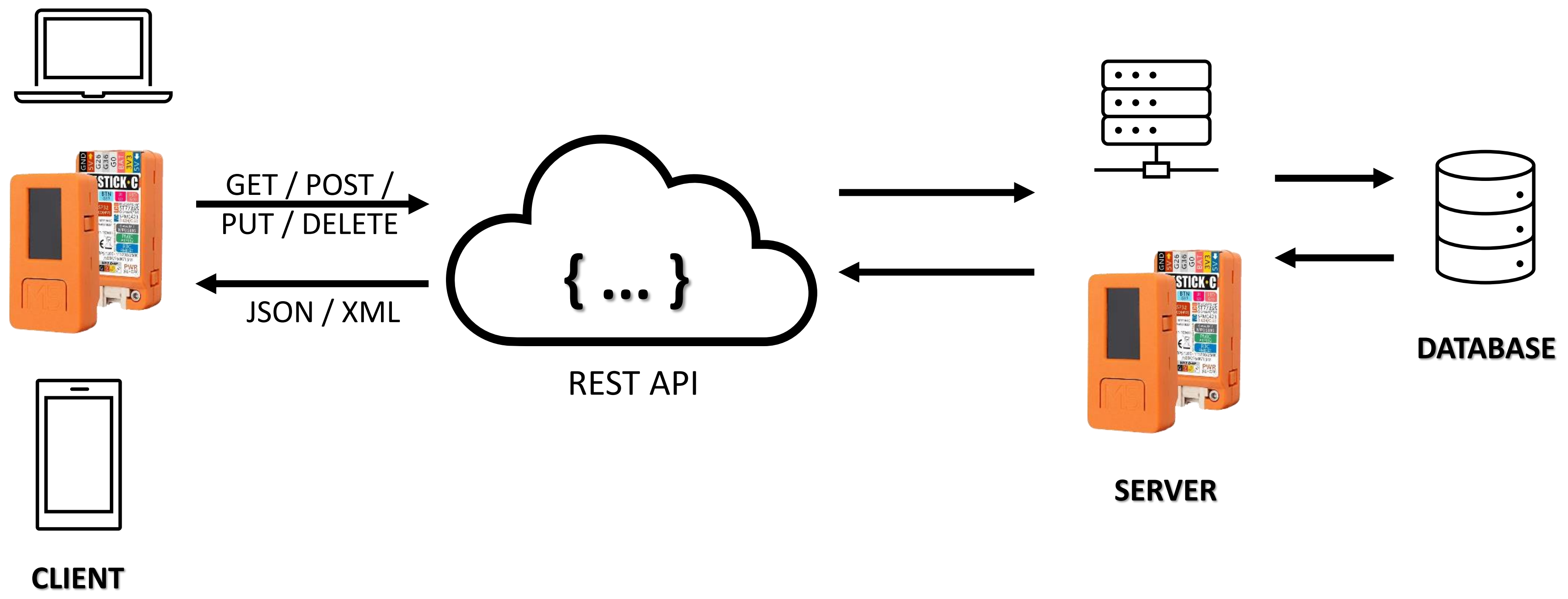
## {REST:API}

## REpresentational State Transfer (REST)

- REST was first introduced by Roy Fielding in 2000.
- REST is web standards-based architecture and uses HTTP Protocol
- REST is not a standard, but an architectural style
- Revolves around resource where every component is a resource
- A resource is accessed by a common interface using HTTP standard methods
- REST can adopt several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - text/xml, text/html, image/gif, image/jpeg, etc  (Resource Types, MIME Types)

*https://blog.ndepend.com/rest-vs-restful/*

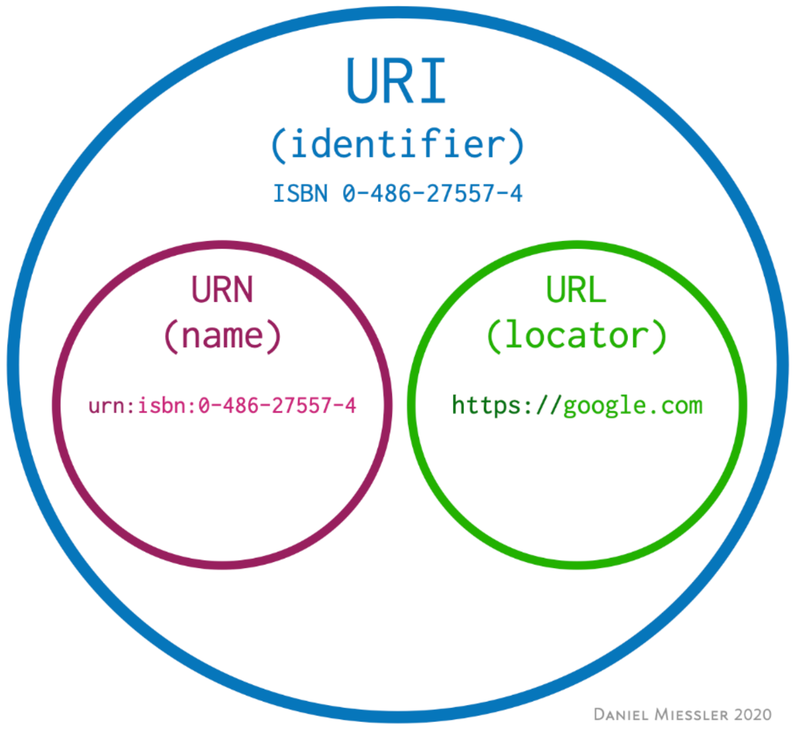
## Software Architecture for REST



## REST & HTTP

- The motivation for REST was to take advantage of the success of the Web
  - URI Addressable resources → URL
  - HTTP Protocol
  - Request → Response → Display Response
- Make use of the HTTP protocol beyond POST & GET
  - HTTP PUT, HTTP DELETE

A URI is an identifier.
A URL is an identifier that tells you how to get to it.



## Principles of REST

1. **Client-Server Architecture** - The client and server act independently.
2. **Statelessness** - The server does not record the state of the client.
3. **Cacheable** - The server marks whether data is cacheable.
4. **Layered System** - The application behaves the same regardless of any intermediaries between the client and server.
5. **Uniform Interface** - The client and server interact in a uniform and predictable way. An important aspect of this is that the server exposes resources.
6. **Code on Demand (optional)** – It allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. Allowing features to be downloaded after deployment improves system extensibility.

## Resources

- The key abstraction of information in REST is a resource
- A resource is a conceptual mapping to a set of entities
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g., “today's weather in Berlin”), a collection of other resources, a non-virtual object (e.g., a person), etc.
- Represented with a global identifier (URI in HTTP)
  - http://www.boeing.com/aircraft/747

## Naming Resources

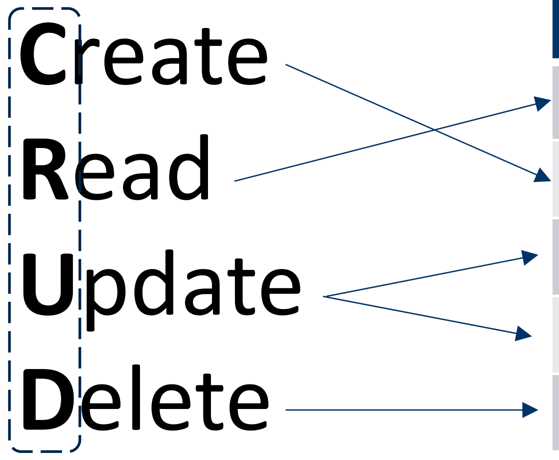
- REST uses URI to identify resources

  - http://localhost/books/
  - http://localhost/books/ISBN-0011
  - http://localhost/books/ISBN-0011/authors

  - http://localhost/classes
  - http://localhost/classes/cs2650
  - http://localhost/classes/cs2650/students

- As you traverse the path from more generic to more specific, you are navigating the data

# REST Verbs



| Verb | Objective | Usage | Multiple Request | Cache |
|---|---|---|---|---|
| GET | Retrieve items from resource | Links | Yes | Yes |
| POST | Create new item in resource | Forms | No | No |
| PUT | Replace existing item in resource | Forms | Yes | No |
| PATCH | Update existing item in resource | Forms | No/Yes | No |
| DELETE | Delete existing item in resource | Forms/Links | Yes | No |

## GET

- How clients ask for the information they seek

- Issuing a GET request transfers the data from the server to the client in some representation

- GET http://localhost/books
  - Retrieve all books

- GET http://localhost/books/ISBN-0011021
  - Retrieve book identified with ISBN-0011021

- GET http://localhost/books/ISBN-0011021/authors
  - Retrieve authors for book identified with ISBN-0011021

## PUT & POST

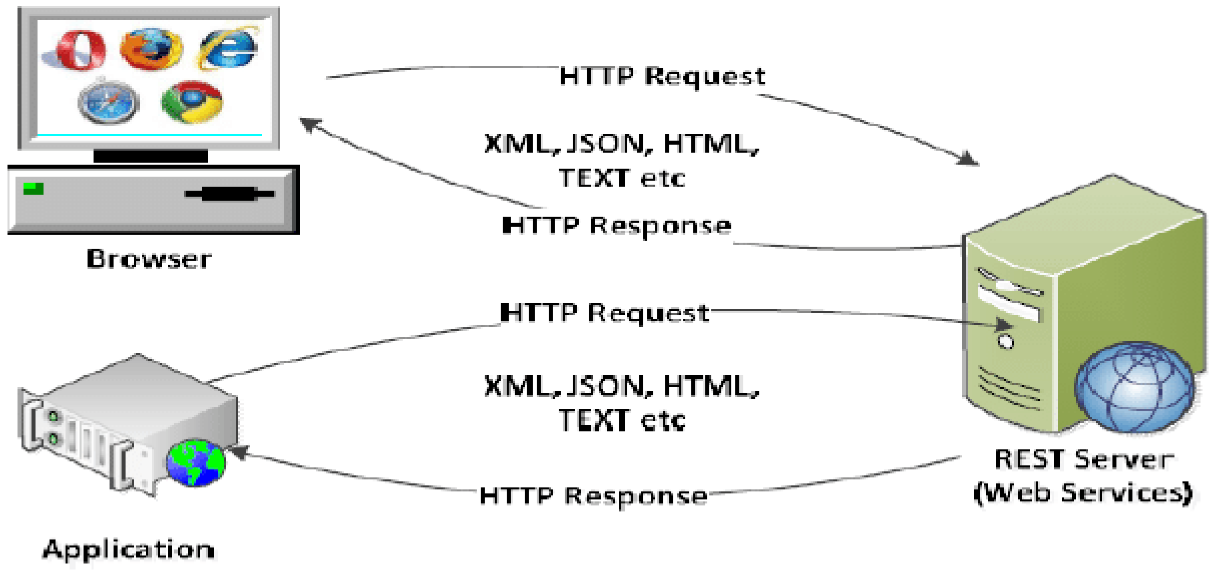
- HTTP POST creates a resource
- HTTP PUT updates the (entire) resource

- POST http://localhost/books/
  - Content: {title, authors[], …}
  - Creates a new book with given properties

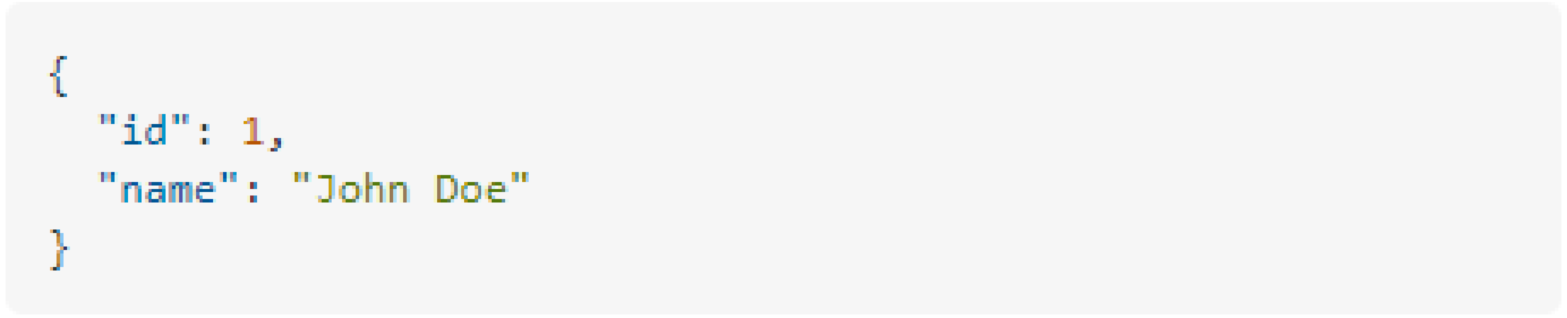
- PUT http://localhost/books/isbn-111
  - Content: {isbn, title, authors[], …}
  - Update book identified by isbn-111 with submitted properties

## Representations

- How data is represented or returned to the client for presentation.
- Typical formats:
  - JavaScript Object Notation (JSON)
  - Extensible Markup Language (XML)

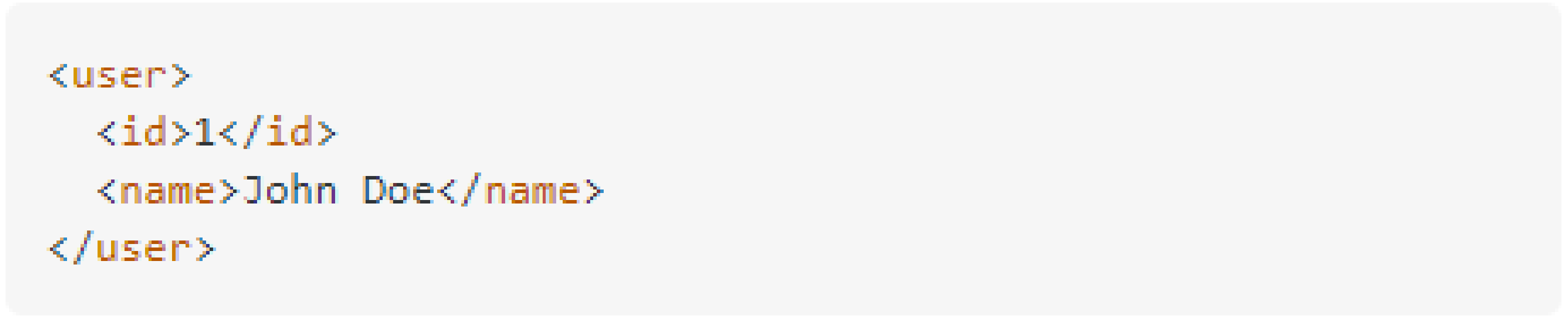

HTTP Request
XML, JSON, HTML, TEXT etc
HTTP Response
Browser
HTTP Request
XML, JSON, HTML, TEXT etc
HTTP Response
REST Server (Web Services)
Application

## Representations Examples



```
{
  "id": 1,
  "name": "John Doe"
}
```

JSON

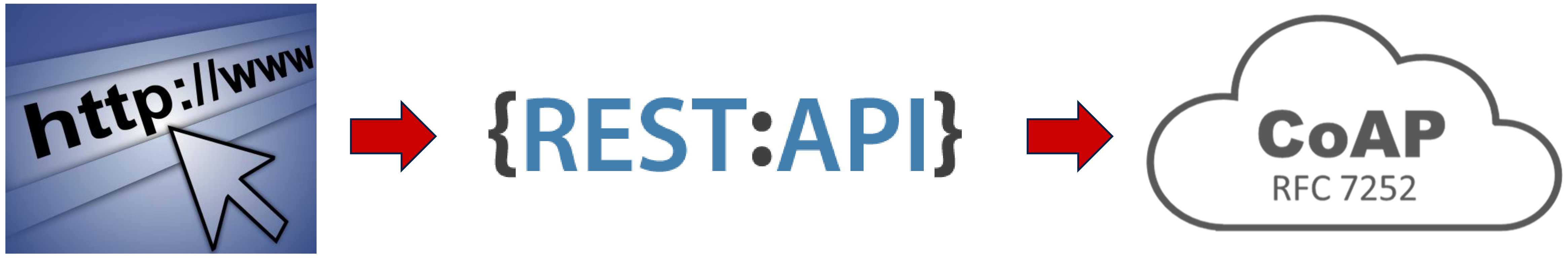


```
<user>
  <id>1</id>
  <name>John Doe</name>
</user>
```

XML

- The above shown documents are not the resource itself.
- They are just a way to represent the resource which is stored somehow in your server.

## ~~Overview~~ Summary


http://www
{REST:API}
CoAP
RFC 7252