Advanced Software Engineering Internet Applications

1. Internet & Web Basics

Internet Basics

1. Hardware components.
2. Internet protocols
3. Messaging.

Internet


PC


server


wireless laptop

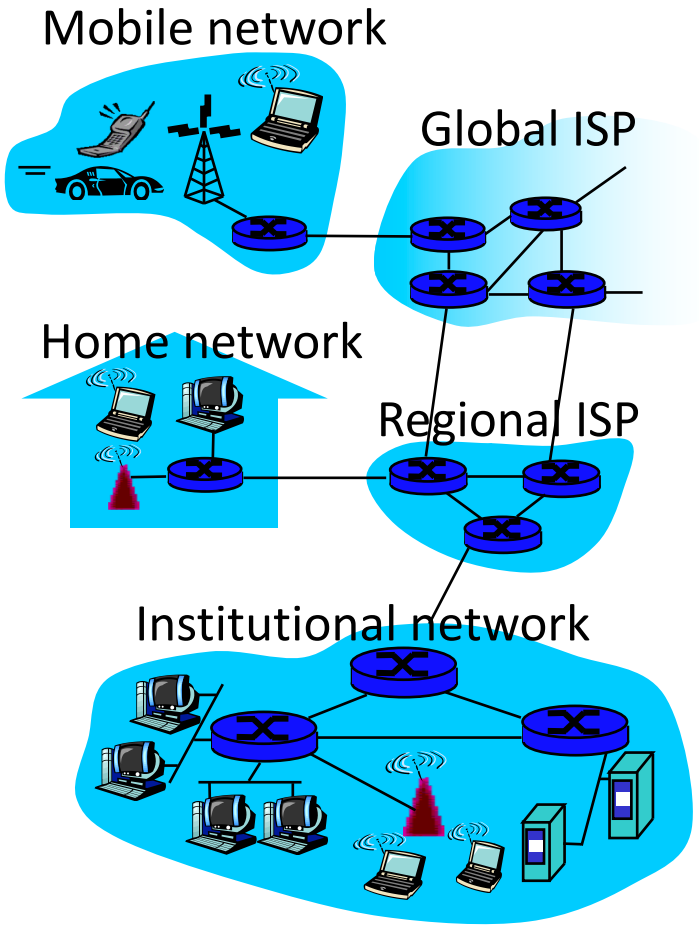

cellular handheld


access points

wired links

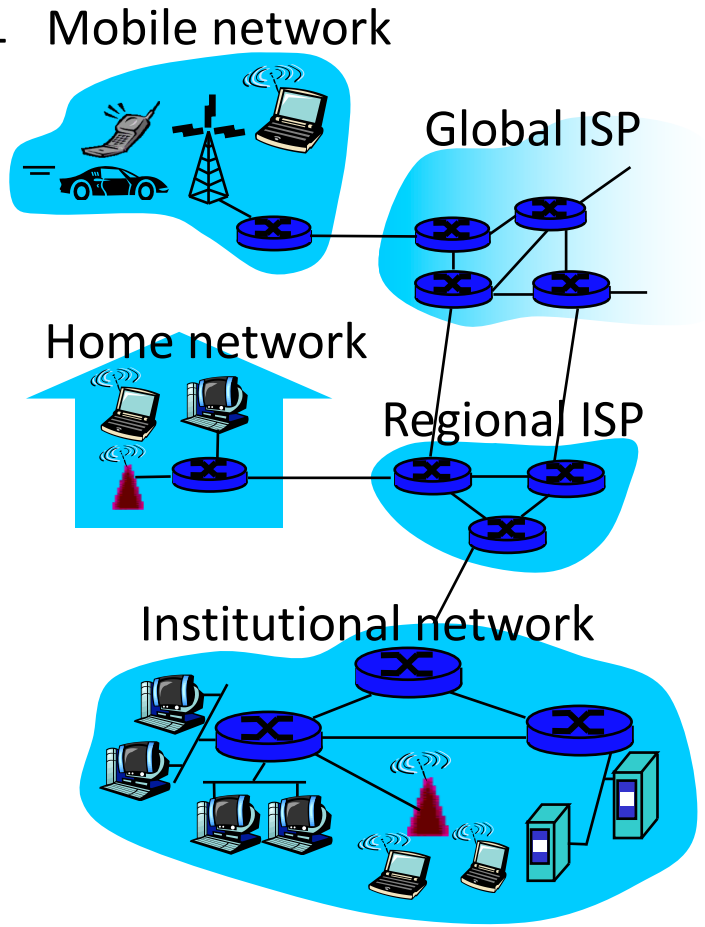

router

- millions of connected computing devices: *hosts = end systems*
  - running *network apps*
- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate = *bandwidth*
- *routers:* forward packets (chunks of data)



Internet

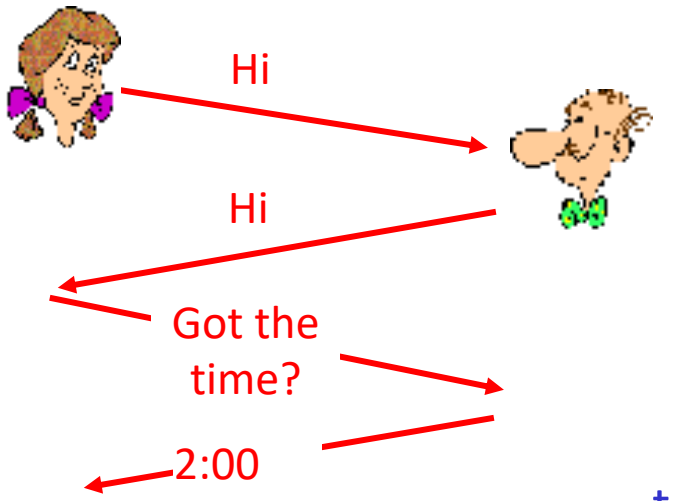
- *protocols* control sending, receiving of msgs
  - e.g., Ethernet, IP, TCP or UDP, HTTP, MQTT
- *Internet:* "network of networks"
  - loosely hierarchical
  - public Internet versus private intranet
- Internet standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



Protocols

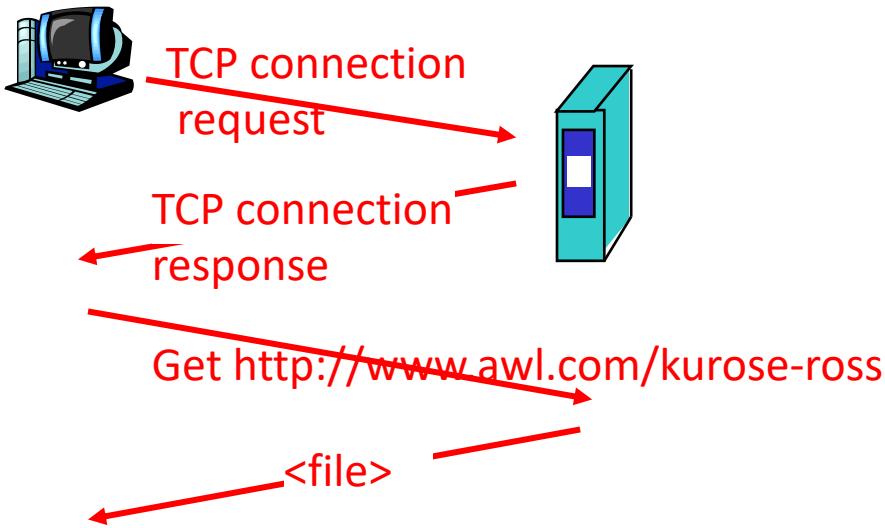
human protocols

- "what's the time?"
- "I have a question"



network protocols

- all communication activity in the Internet governed by protocols

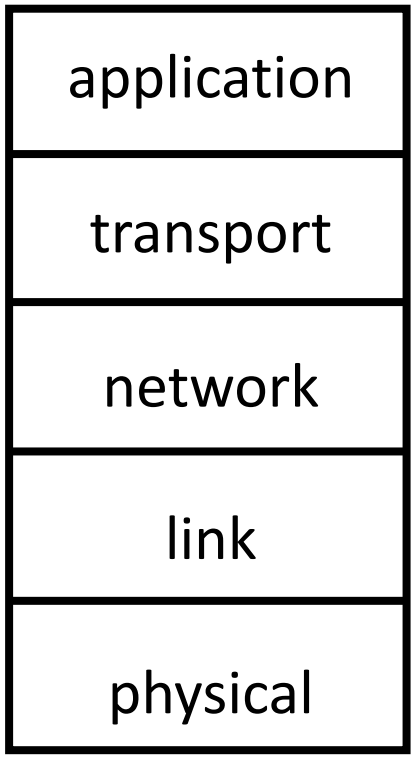


protocols define

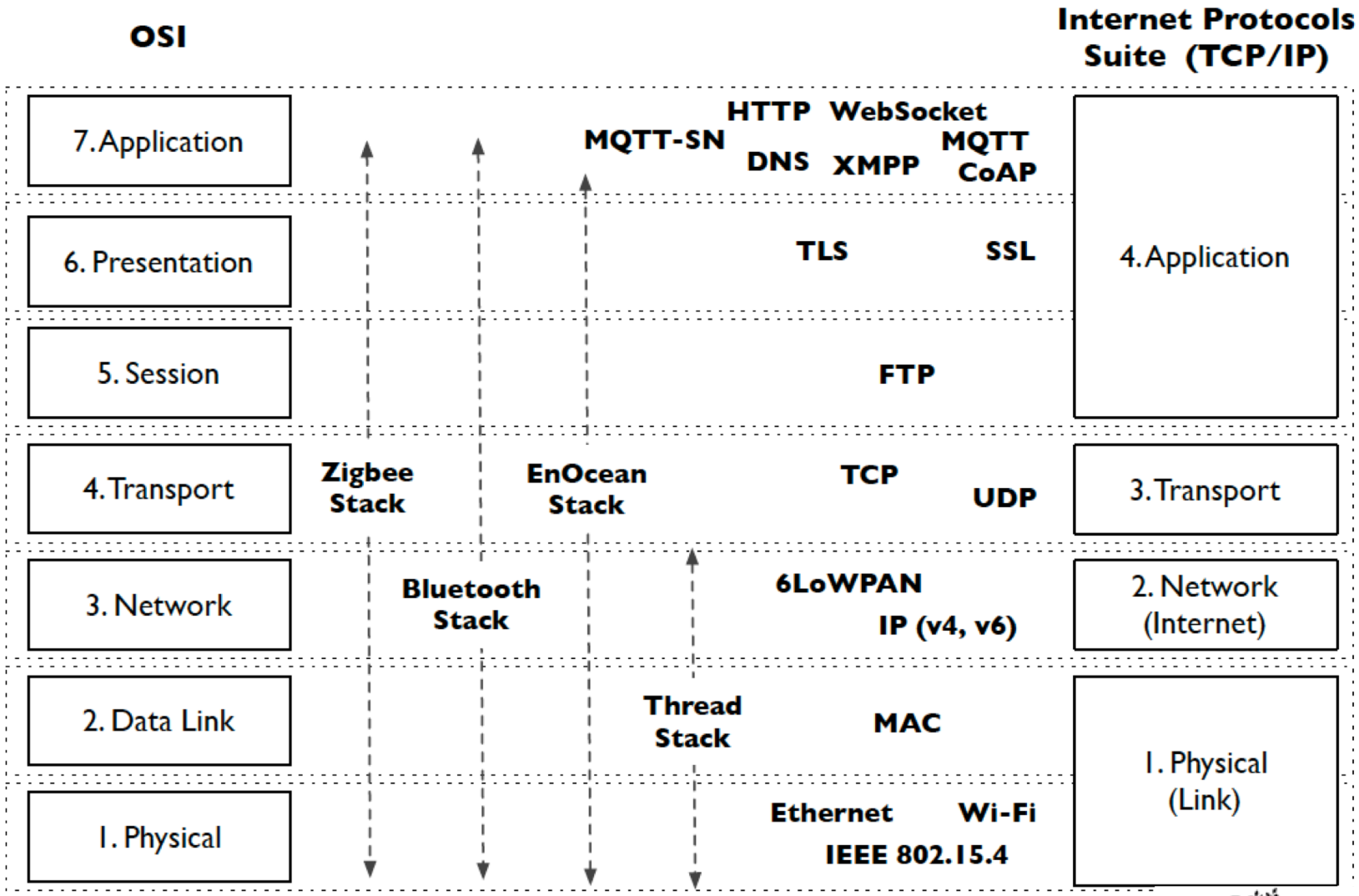
- *format,*
- *order of msgs sent and received among network entities,*
- *and actions taken on msg transmission, receipt*

Internet Protocol Stack

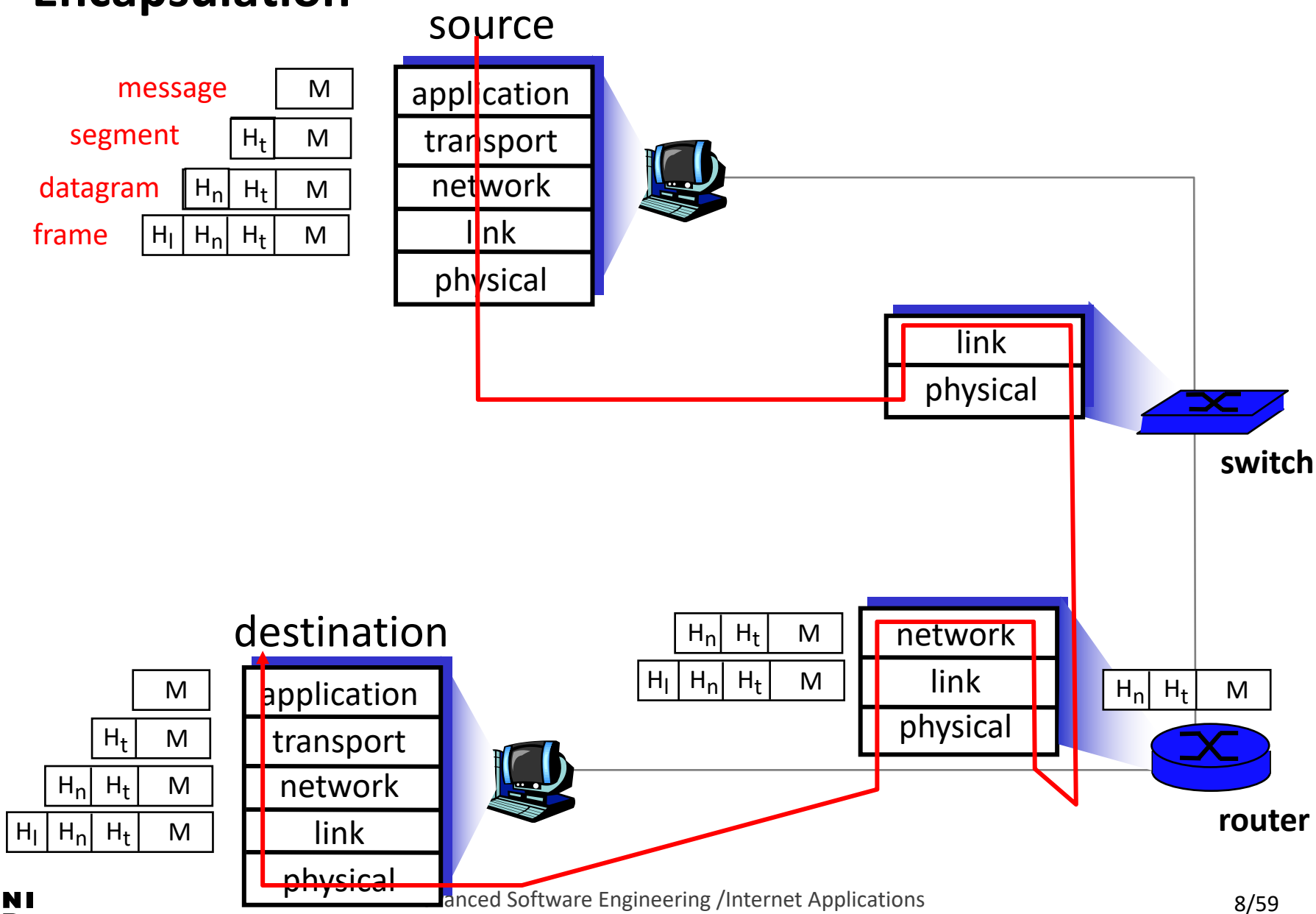
- application: supporting network applications
  - FTP, SMTP, HTTP, CoAP, SOAP, MQTT
- transport: process-process data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - PPP, Ethernet
- physical: bits “on the wire”



Internet Protocol Stack : a Second View from IoT



Encapsulation



Transport Layer

TCP

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded

does not provide: timing, minimum throughput guarantees, security

UDP

- unreliable data transfer between sending and receiving process

does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

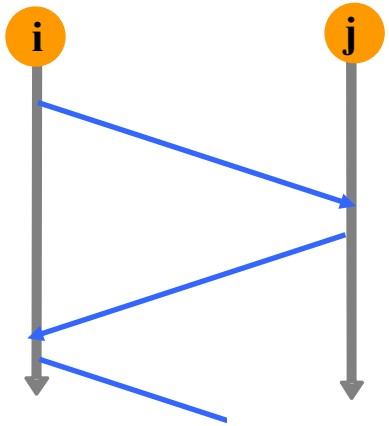
Messaging

- Distributed systems communicate by exchanging messages.
- Message passing between a pair of processes is supported by two message communication operations: send and receive, defined in terms of destinations and messages.
- **Marshalling** (serialization) is the process of taking any form of structured data items and breaking up so that it can be transmitted as a stream of bytes over a communications network in such a way that the original structure can be reconstructed easily on the receiving end.
- **Unmarshalling** (deserialization) is the process of converting the assembled stream of bytes on arrival to produce an equivalent form of structured data at the destination point.

Synchronous and Asynchronous Messaging

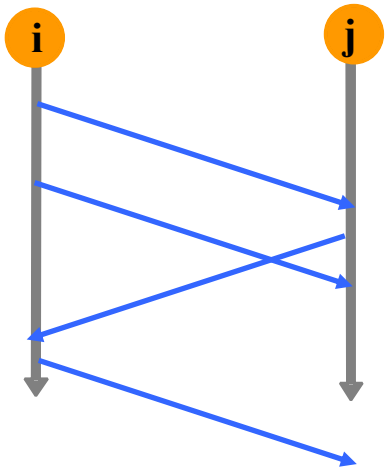
Synchronous communication

- synchronized between two communicating application systems, which must both be up and running.
- Execution flow at the caller's side is interrupted to execute the call.



Asynchronous communication

- the caller employs a send and forget approach that allows it to continue to execute after it sends the message.



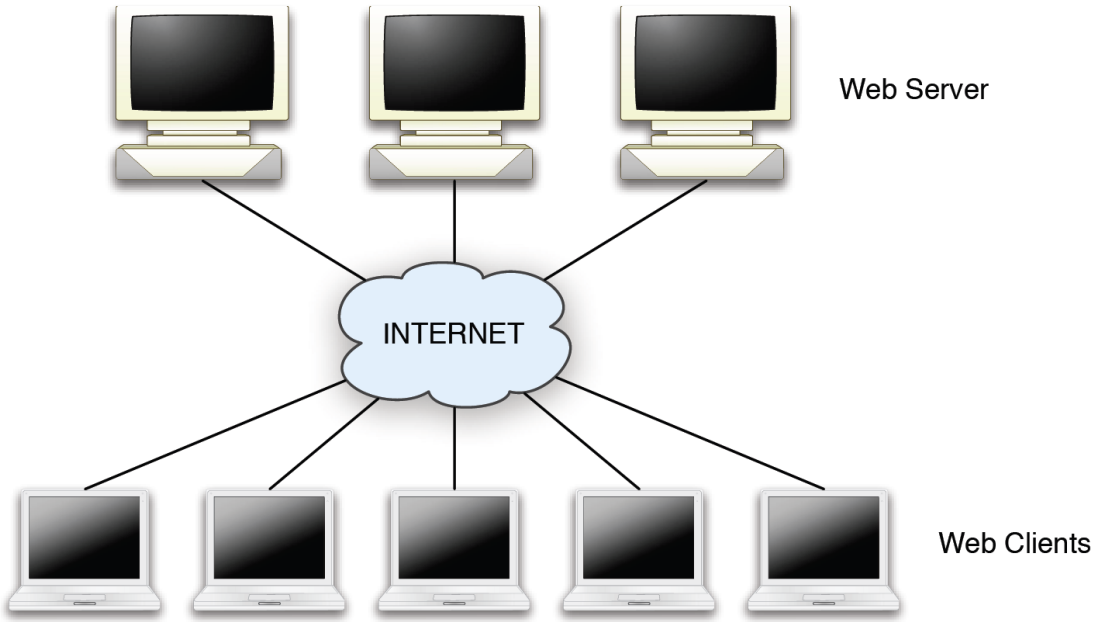
Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

Application Layer: Web

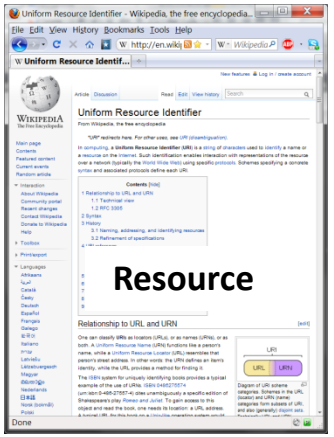
The World Wide Web, abbreviated as WWW and known as the Web is a system of interlinked hypertext documents accessed via the Internet.

- Web client: Web browser to view Web pages and navigate between them by using hyperlinks.
- Web server: provide Web pages that may contain text, images, videos, and other multimedia



Uniform Resource Identifiers (URIs)

http://en.wikipedia.org/wiki/Uniform_Resource_Identifier →



- **URI** is a string of characters used to identify a name or a resource on the Internet.
- URI can be a URL or a URN.
- Uniform Resource Name (**URN**) defines an item's identity :
  - the URN `urn:isbn:0-395-36341-1` is a URI that specifies the identifier system, i.e. International Standard Book Number (ISBN), as well as the unique reference within that system and allows to talk about a book, but doesn't suggest where and how to obtain an actual copy of it.
- Uniform Resource Locator (**URL**) also provides a method for finding it :
  - the URL above identifies a resource (Wikipedia page) and implies that a representation of that resource (such as the page's current HTML code, as encoded characters) is obtainable via HTTP from a host named `en.wikipedia.org`.

Hyper-Text Markup Language (HTML)

- HTML
  - a subset of Standardized General Markup Language (SGML)
  - facilitates a hyper-media environment
- Documents use elements to “mark up” or identify sections of text for different purposes or display characteristics
- HTML markup consists of several types of entities
  - including: elements, attributes, data types and character references
- Documents are rendered by browsers
  - markup elements are not seen by the user when page is displayed

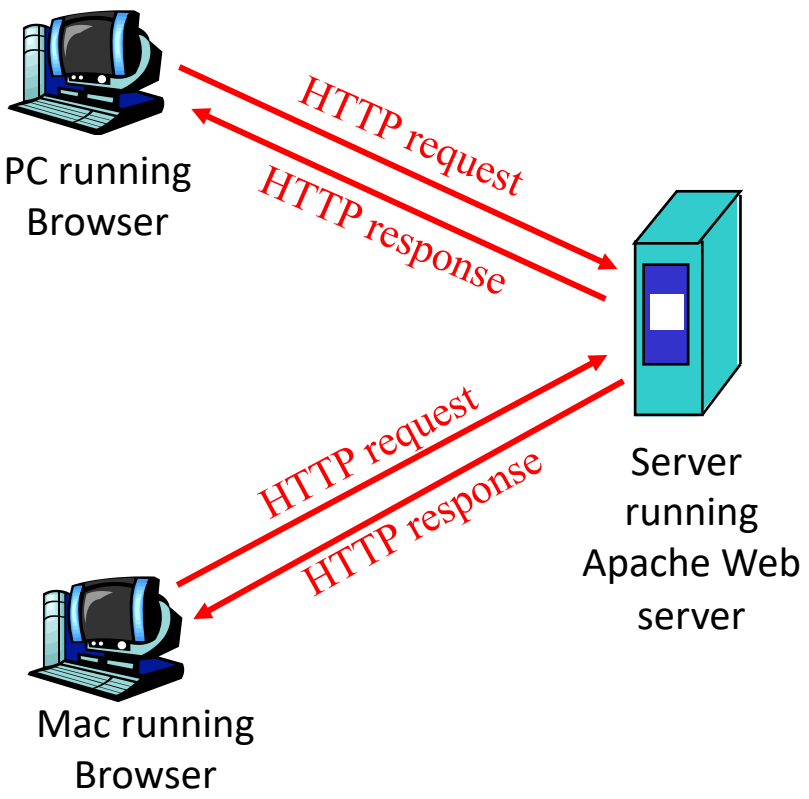
```
<!doctype html>
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

Hyper-Text Transfer Protocol (HTTP)

- employs client/server model
  - *client:* browser that requests, receives, “displays” Web objects
  - *server:* Web server sends objects in response to requests
- Uses TCP:
  - client initiates TCP connection (creates socket) to server
  - server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- HTTP is “stateless”
  - server maintains no information about past client requests



HTTP Request Methods – 1 (the five CRUD ones)

HTTP request methods indicate the desired action to be performed on the identified resource :

- **POST** (Create)
  - Creates a new (subordinate) element. The data to be processed (e.g., from an HTML form) are included in the body of the request.
- **GET** (Read)
  - Requests a representation of the specified resource.
- **PUT** (Update/Replace)
  - Replace an "entire" given resource. If it does not exist, it is created.
- **PATCH** (Update/Modify)
  - Modifies "partially" a given resource.
- **DELETE** (Delete)
  - Deletes the specified resource.

HTTP Request Methods – 2 (others)

- **HEAD**
  - Requests a resource. The server only sends back the associated header information without the actual content.
- **OPTIONS**
  - Checks the server communication options.
- **UPGRADE**
  - Is used for switching to another protocol (e.g. WebSocket).
- **TRACE**
  - Debugging use : allows the client to see what the server on the other side is receiving.
- **CONNECT**
  - Is used for dynamically switching to a tunneled connection.

HTTP Status Codes

- **1xx**
  - denotes general information like 101 Continue.
- **2xx**
  - denotes accepted and successfully executed requests.
- **3xx**
  - denotes all types of redirections. To fulfill the request, the client has to take further actions.
- **4xx**
  - denotes client errors.
- **5xx**
  - denotes server errors.

HTTP Request Syntax

```
METHOD URI PROTOCOL
HEADER_1: VALUE
HEADER_2: VALUE
…
HEADER_n: VALUE

BODY OF THE REQUEST
…
```

empty line!

often empty

- Available methods: `GET`, `POST`, `PUT`, `PATCH`, `DELETE`
- Protocols: `HTTP/1.0`, `HTTP/1.1`

HTTP Request Example – 1 : A minimal Get

```
> GET /utc/now HTTP/1.1
> User-Agent: curl/7.35.0
> Host: www.timeapi.org
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Fri, 16 Sep 2016 16:17:58 GMT
< Connection: keep-alive
< Content-Type: text/plain;charset=utf-8
< Content-Length: 25
<
2016-09-16T17:17:59+01:00
```

This server does not respond anymore. A more complex example would be

curl --verbose http://worldtimeapi.org/api/timezone/Europe/Zurich

HTTP Request Example – 2 : Get Pokemon Trainers List in application/json

```
> GET /api/v1/trainers/ HTTP/1.1
> User-Agent: curl/7.35.0
> Host: pokemon.thing.zone
> Accept: application/json
>
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 234
< Date: Fri, 16 Sep 2016 16:06:29 GMT
<
[
    {
        "country_code": "JPN",
        "gender": "male",
        "id": 1,
        "name": "Ash Ketchum"
    },
    {
        "country_code": "JPN",
        "gender": "female",
        "id": 2,
        "name": "Serena"
    }
```

HTTP Request Example – 3 : Get Pokemon Trainers List in application/xml

```
> GET /api/v1/trainers/ HTTP/1.1
> User-Agent: curl/7.35.0
> Host: pokemon.thing.zone
> Accept: application/xml
>
< HTTP/1.1 200 OK
< Content-Type: application/xml
< Content-Length: 299
< Date: Fri, 16 Sep 2016 16:06:29 GMT
<
<?xml version="1.0" encoding="UTF-8"?>
<trainers>
    <trainer id="1">
        <country_code>JPN</country_code>
        <gender>male</gender>
        <name>Ash Ketchum</name>
    </trainer>
    <trainer id="2">
        <country_code>JPN</country_code>
        <gender>female</gender>
        <name>Serena</name>
    </trainer>
</trainers>
```

HTTP Request Example – 4 : POST a new Pokemon Trainer in application/json

```
> POST /api/v1/trainers/ HTTP/1.1
> User-Agent: curl/7.35.0
> Host: Host: pokemon.thing.zone
> Content-Type: application/json
> Accept: application/json
> Content-Length: 55
>
  {"country_code":"CHE","gender":"male","name":"Jacques"}
< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 140
< Date: Fri, 16 Sep 2016 16:11:02 GMT
<
{
    "country_code": "CHE",
    "created": "2016-09-16T16:11:02.919007+00:00",
    "gender": "male",
    "id": 20,
    "name": "Jacques",
}
```

Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

A Word about RESTful Web Services – 1

- Definition (valid for all types of WS)

  A **web service** is a software component supporting interoperable machine-to-machine interactions over a network. Each component is deployed in a supporting container, giving it a life-cycle and ensuring communication with the outer world. It has a well defined API and produces and consumes standardized messages over a well defined protocol.

A Word about RESTful Web Services – 2

- In the **REST architectural style**, data and functionality are considered resources and are accessed using URIs, typically links on the Web.
- The resources are acted upon by using a Uniform interface.
- The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP.
- Clients and servers exchange representations of resources.
- Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

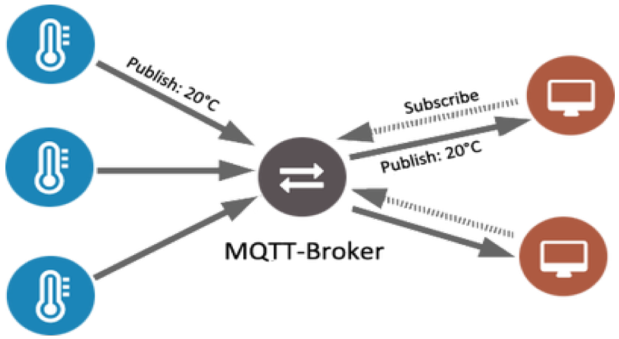
Publish-Subscribe Pattern

- Communication through **channels**
  - Publisher is not aware of subscribers
  - Subscriber is not aware of publishers
- Topic-based system
  - Use named channels e.g. */temperature*
- Content-based system
  - Subscribers filter from content

MQTT

ISO/IEC PRF 20922

- Pub/sub **topic-based** lightweight protocol on top of TCP
- Publisher and subscribers are clients
- **MQTT broker** distributes messages
  - Acts as server—stable point in the network



source: https://pagefault.blog/2017/03/02/using-local-mqtt-broker-for-cloud-and-interprocess-communication/

- MQTT over WebSocket enables direct communication between browsers and broker
  - Used by webbluetooth-proxy to exchange messages between Bluetooth devices and broker

MQTT

Topics

- A topic is a string referring the channel
  - *house/room1/temperature*
    - separator: /
    - topic level: house
    - topic level: room1
- Can have multiple levels with the / separator
- Wildcards
  - Single level: **+**
    - *house/+/temperature*
  - Multi level: **#**
    - *house/#*
- Publishing is always to a single topic = no wildcard

MQTT

Extra features

- Quality of Service (QoS)—3 levels
  - At most once (0)
  - At least once (1)
  - Exactly once (2)
- Last will testament
  - Publisher tells the broker to publish a message if it detects a **connection break**
  - Based on keep-alive interval
- Retained messages
  - Publisher tells broker to keep message even if no subscriber
  - Only **one (last) message** retained **per topic**

Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

XML

Markup Language

- method of distinguishing text from instructions
- special characters are used to show where a markup instruction starts and stops

```
<centre on> This is a <italics on> very serious <italics off>
matter.<centre off>
```

This is a *very serious* matter.

- “Tags” are usually readable by humans

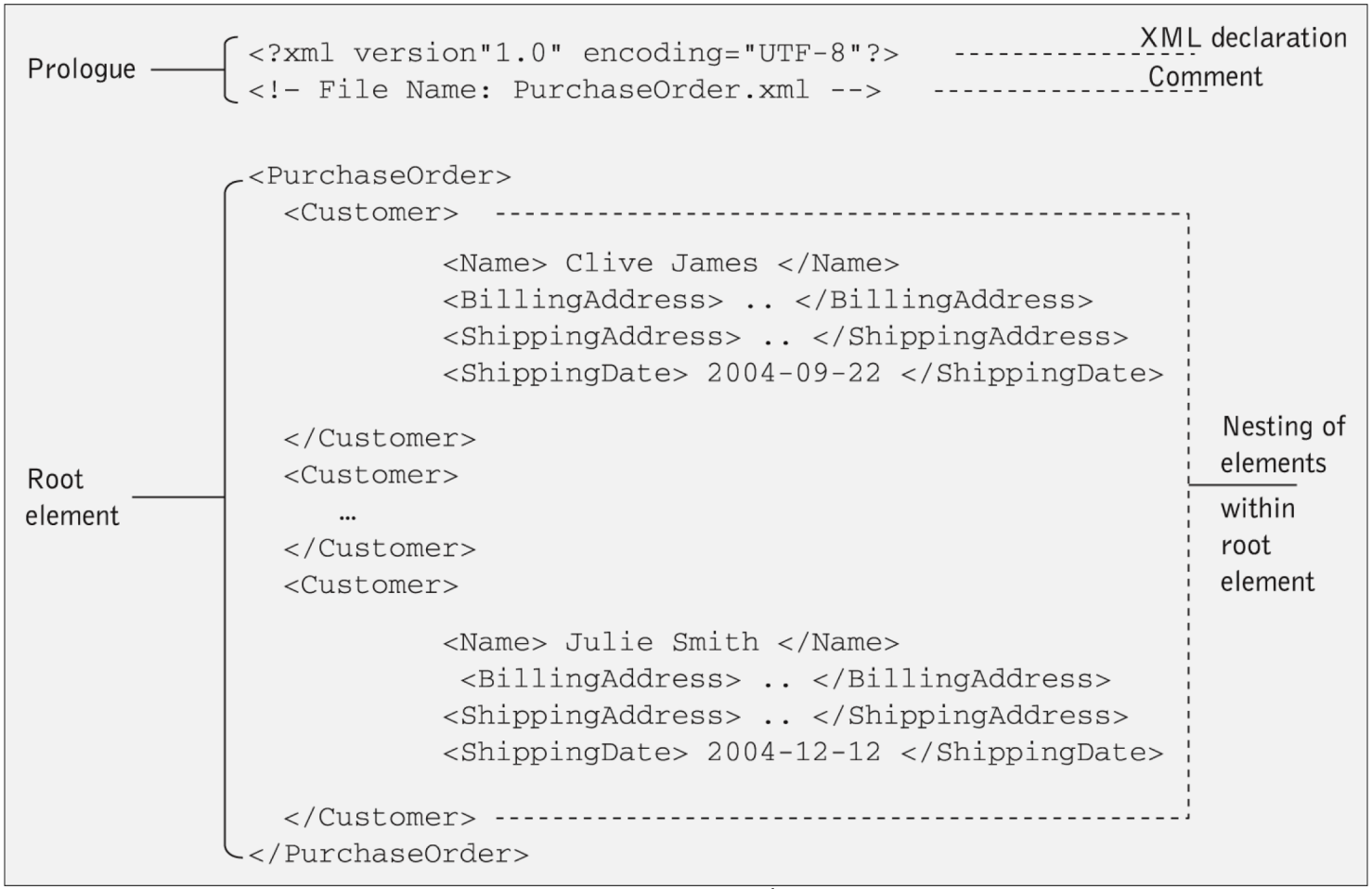
XML

- Extensible Markup Language (XML) is not a set of tags itself but a meta-language that allows you to create and use tag sets in a standard way.
- Provides fixed rules about markup structures (elements, attributes, and entities), about their notation, and their function.

XML Structure

XML Document is composed of named **containers** and their data values.

- containers are represented as declarations, elements, and attributes.



XML Schema

- **XML schema** refers to a document that defines the content of and structure (grammar) for expressing XML documents.
- Schemas provide support for additional meta-data characteristics such as structural relationships, cardinality, valid values, and data types.
- Each type of schema acts as a method of describing data characteristics and applying rules and constraints to a referencing XML document.
- An XML schema describes the elements and attributes that may be contained in a schema conforming document and the ways that the elements may be arranged within a document structure.
- Its syntax is based on XML itself.
- Reuse and refinement of schemas.
- An XML schema is an element with an opening tag like

```
<schema "http://www.w3.org/2000/10/XMLSchema" version="1.0">
```

- Structure of schema elements
  - Element and attribute types using data types

XML Schema and XML Instance

File “Person.xml”

```
<?xml version="1.0"
encoding="UTF-8"?>

<Person
xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
Person.xsd">

      <First>Sophie</First>

      <Last>Jones</Last>

      <Age>34</Age>

</Person>
```

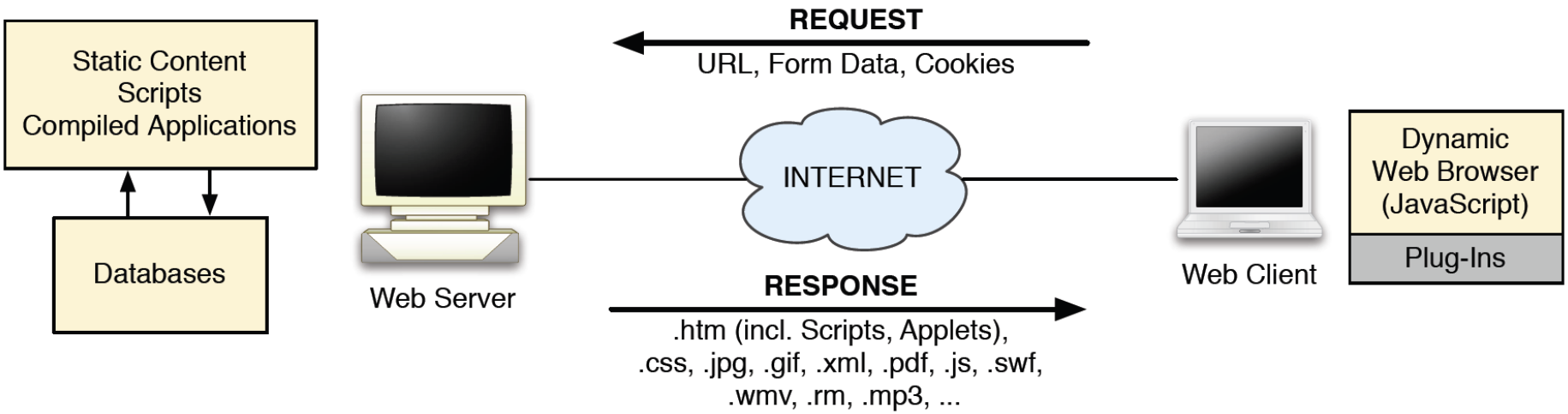
File “Person.xsd”

```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/
             XMLSchema">
 <xs:element name="Person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="First"
                type="xs:string"/>
      <xs:element name="Middle"
                type="xs:string"
                minOccurs="0"/>
      <xs:element name="Last"
                type="xs:string"/>
      <xs:element name="Age"
                type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Web Basics

1. The Web data model: URI and HTML.
2. HTTP Request Syntax :
   - Examples of headers, as well as payloads and answers in HTML, JSON and XML.
3. An introductory word about RESTful web services.
4. Pub / Sub Pattern, MQTT Quickstart
5. More about XML.
6. Web programming and Web application architectures.

Web Programming – 1



Requests and responses are still handled by HTTP (or HTTPS)

Web Programming – 2

Web programming = creating dynamic Web pages

- dynamic: depending on parameters (user input, change in data set, time of day, …)
  - also known as RIA (Rich Internet Application)
- goal: generate HTML for client (Web browser)
  - or something else a Web browser can make use of/display: CSS, JavaScript, PDF, XML, video, audio, ...
  - alternatively delivered via Web browser plugins (e.g., Adobe Flash, Microsoft Silverlight)
- means to write different types of executable code
  - client-side: application logic executed by the client's Web browser
  - server-side: application logic executed by the Web server
- state of the art is a mixture of client-side and server-side logic

Web Programming – 3

- code can be
  - interpreted: script language
    - client: JavaScript, Adobe Flash, Microsoft Silverlight, …
    - server: PHP, Python, JS with Node.js, …
  - compiled: programming language
    - client: Java Applet, JavaFX, …
    - server: Java, .NET, …
- communication between client and server may be based on Web services
  - RESTful or SOAP
  - on top of HTTP/HTTPS
  - data encoded as XML (REST and SOAP) or JSON (REST)

Client or Server? – 1

- Advantages of client-side programs:
  - Fast (exploit the client's computing power)
  - Allow immediate responses to user interactions
  - No special server configuration required
  - Work offline (in the absence of a server)
  - Greater access to the information available client-side

- Disadvantages of client-side programs:
  - Long initial loading time
  - Client can see the code
  - Browser support not guaranteed or not fully compatible
  - Number of supported languages is limited

Client or Server? – 2

- Advantages of server-side programs:
  - Code is hidden from the client
  - Work on all browsers
  - Can produce browser-customized output
  - Enable access to central data bases and files
  - Wide range of supported languages

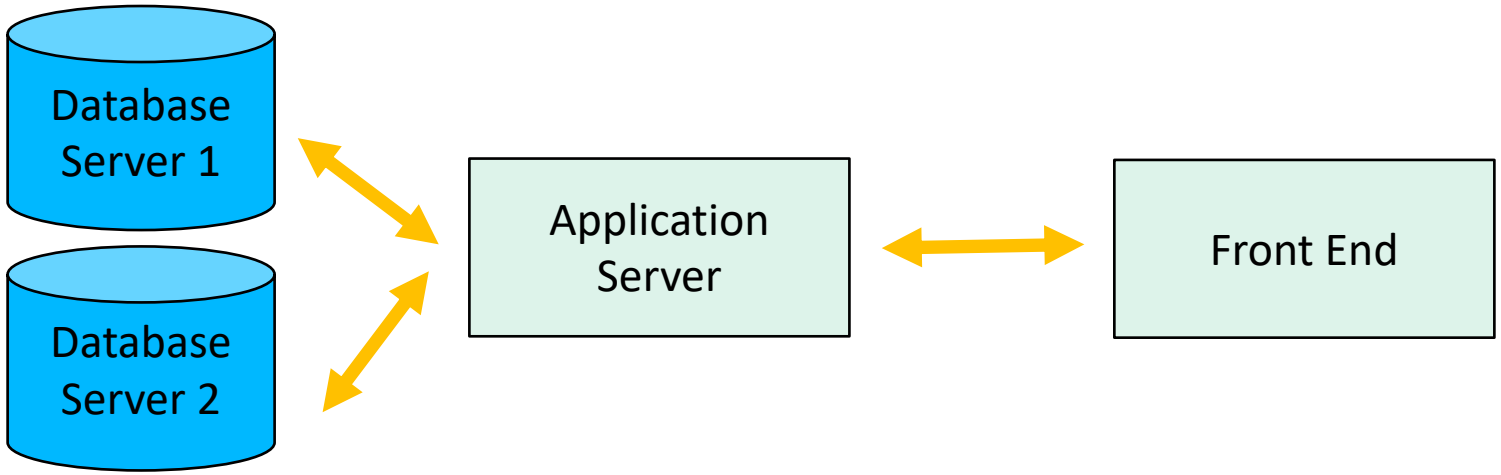
- Disadvantages of server-side programs:
  - Slow page refreshing
  - Server configuration must support the chosen language
  - Require high-performance Web servers

Software Architecture

- Software architecture
  - sub-systems making up a system
  - the framework for sub-system control and communication
- Architectural design process
  - early stage of the system design process
    - often carried out in parallel with some specification activities.
  - represents the link between specification and design processes
    - based on (or in parallel with) context and domain model
- Architecture may concern different levels of abstractions
  - individual programs (and their internal components)
  - complex enterprise systems that are composed of other systems, programs, and program components
    - distributed over different computers, which may be owned and managed by different companies

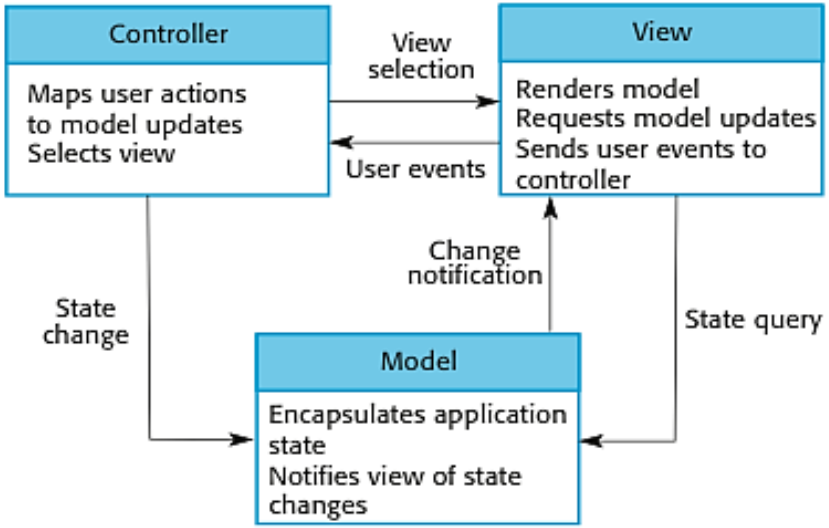
Distributed Systems

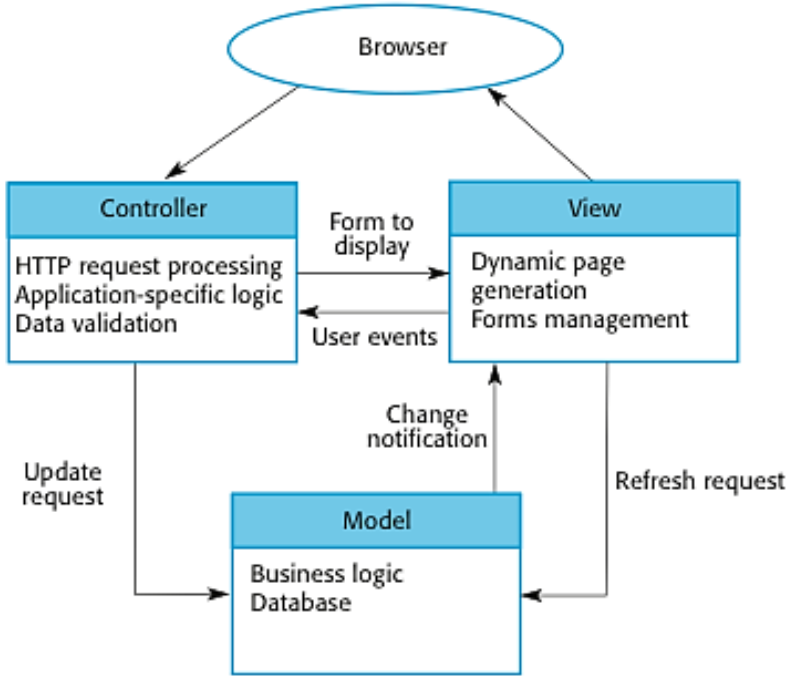
- A distributed system is
  - a collection of networked computers
  - where each computer has one or more tasks and
  - where computers communicate and coordinate their actions by passing messages
- Distribution is transparent to the user so that the system appears as a single integrated facility.



MVC

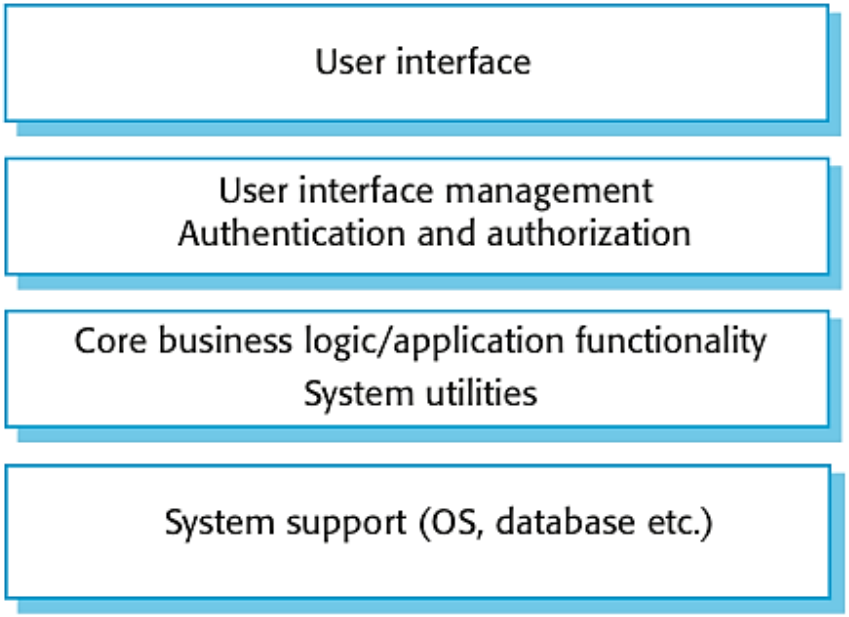
Separates presentation and interaction from the system data.





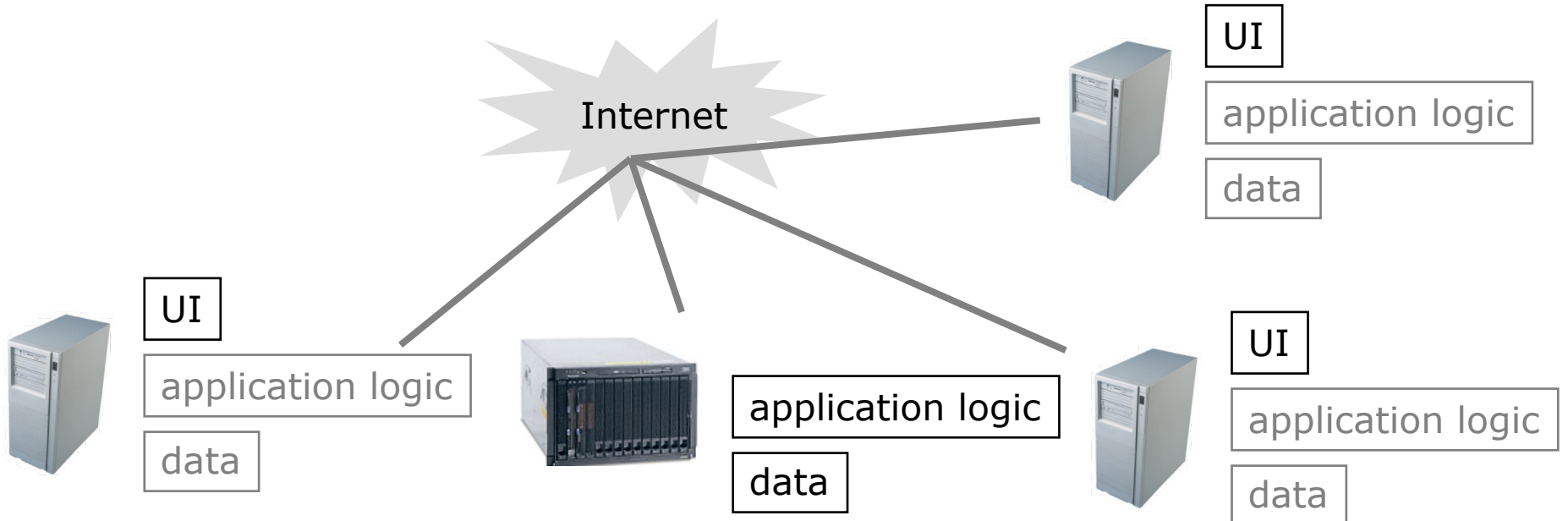
Layers Architecture Pattern

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services
  - layering is a very coarse-grained grouping of subsystem, packages, and classes.
  - layers are organized such that “higher” layers call upon services of “lower” layers, but not normally vice versa.
  - In a strict layered architecture, a layer only calls upon the services of the layer directly below it.
  - Coupling between layers should be minimal.

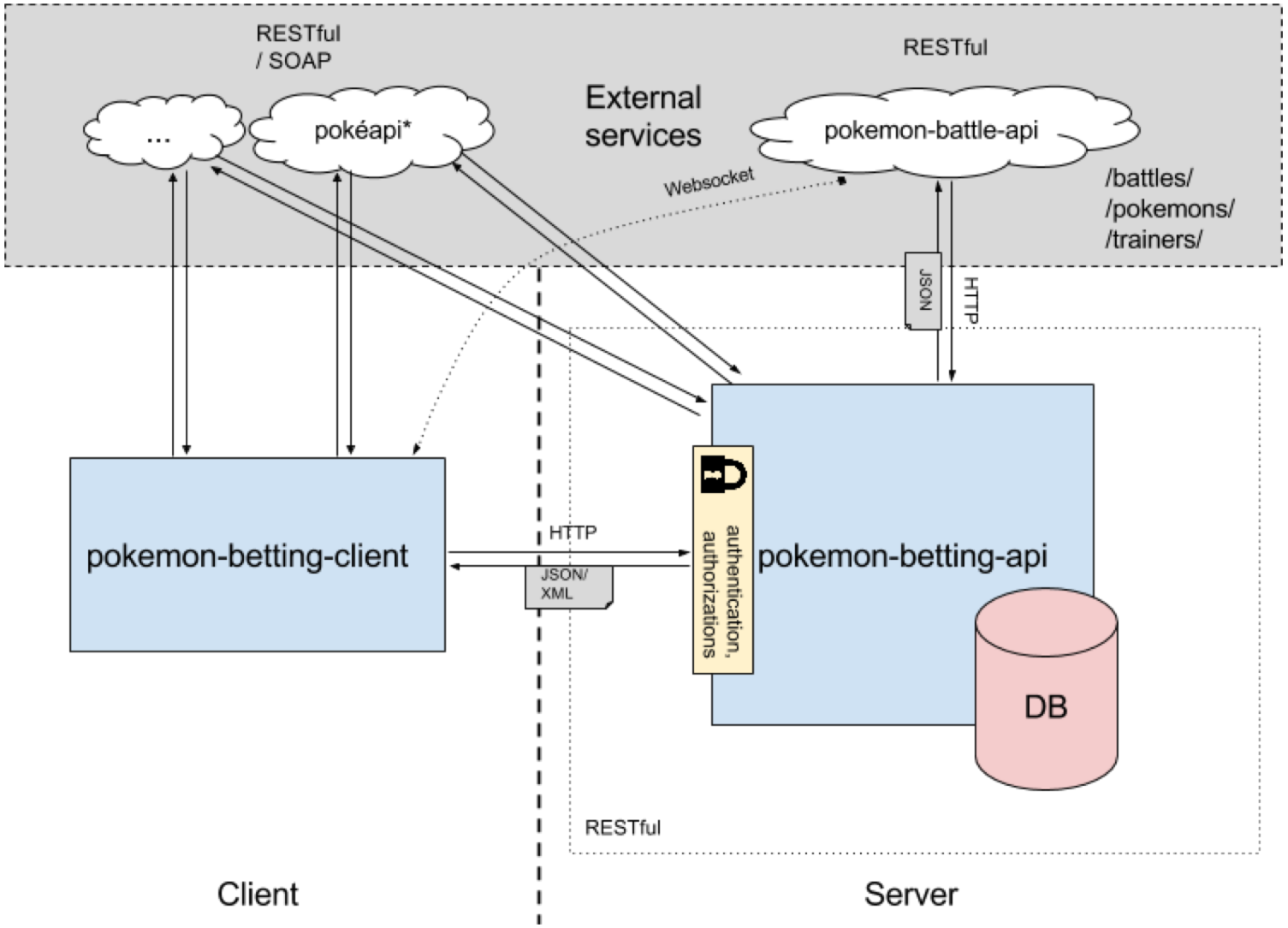


Client-Server Architecture (C/S)

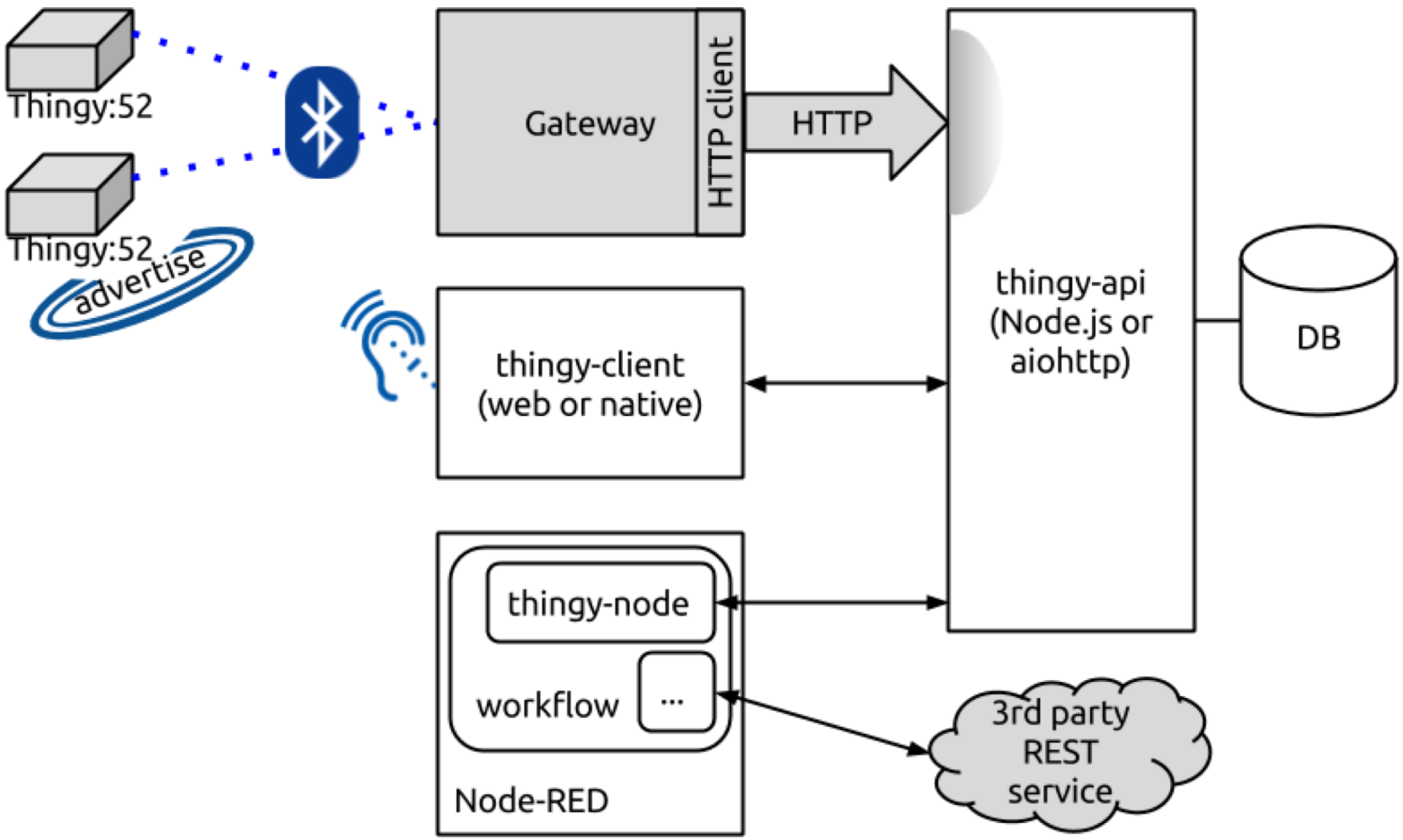
- Server
  - responsible for observing application logic and managing data
  - collects and distributes messages from clients
- Client
  - connects to the server
  - responsible for UI
  - some application logic / data management



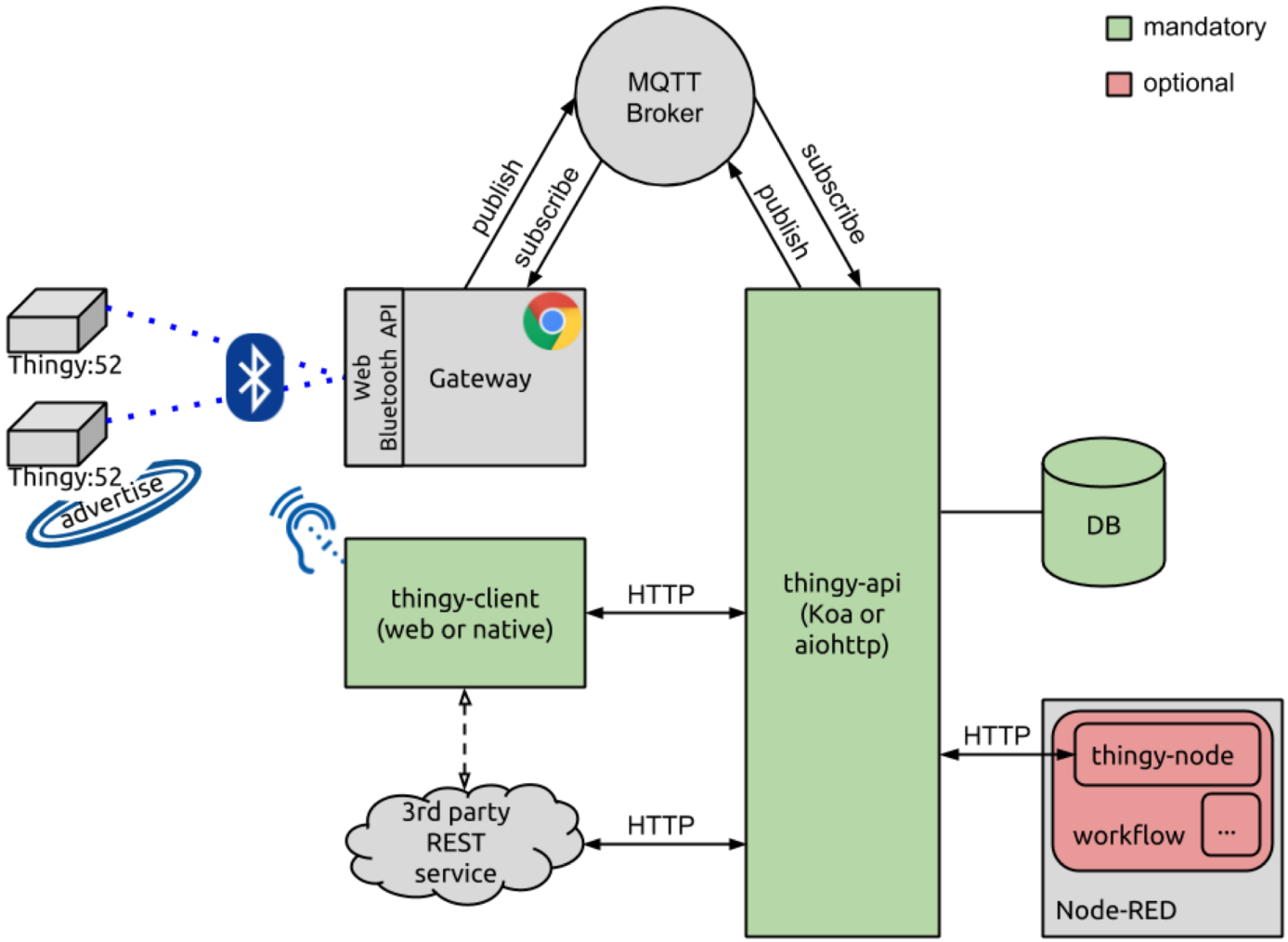
Pokemon Project = Example - 1 of Client-Server System



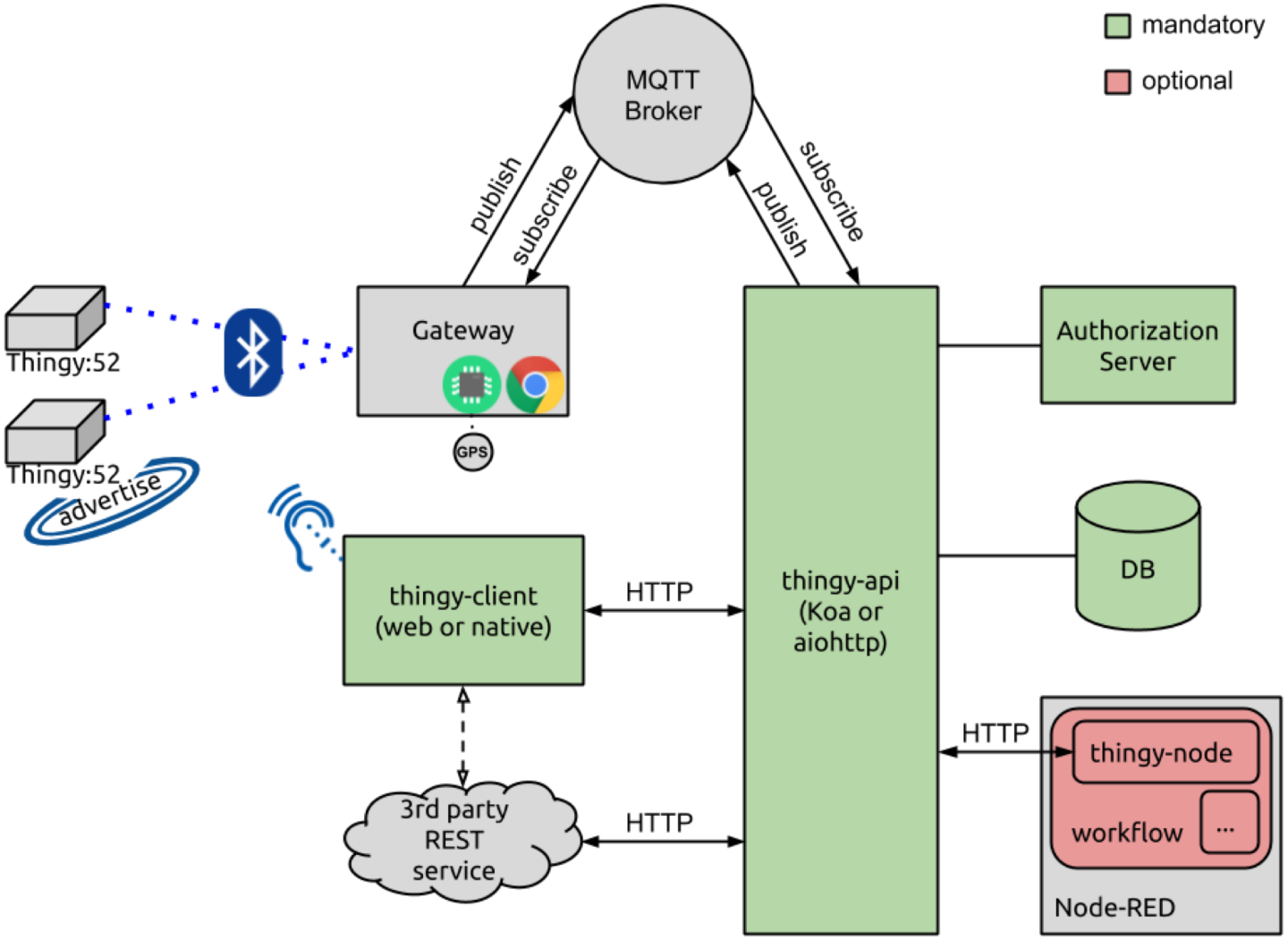
Thingy Project 17 = Example - 2 of Client-Server System



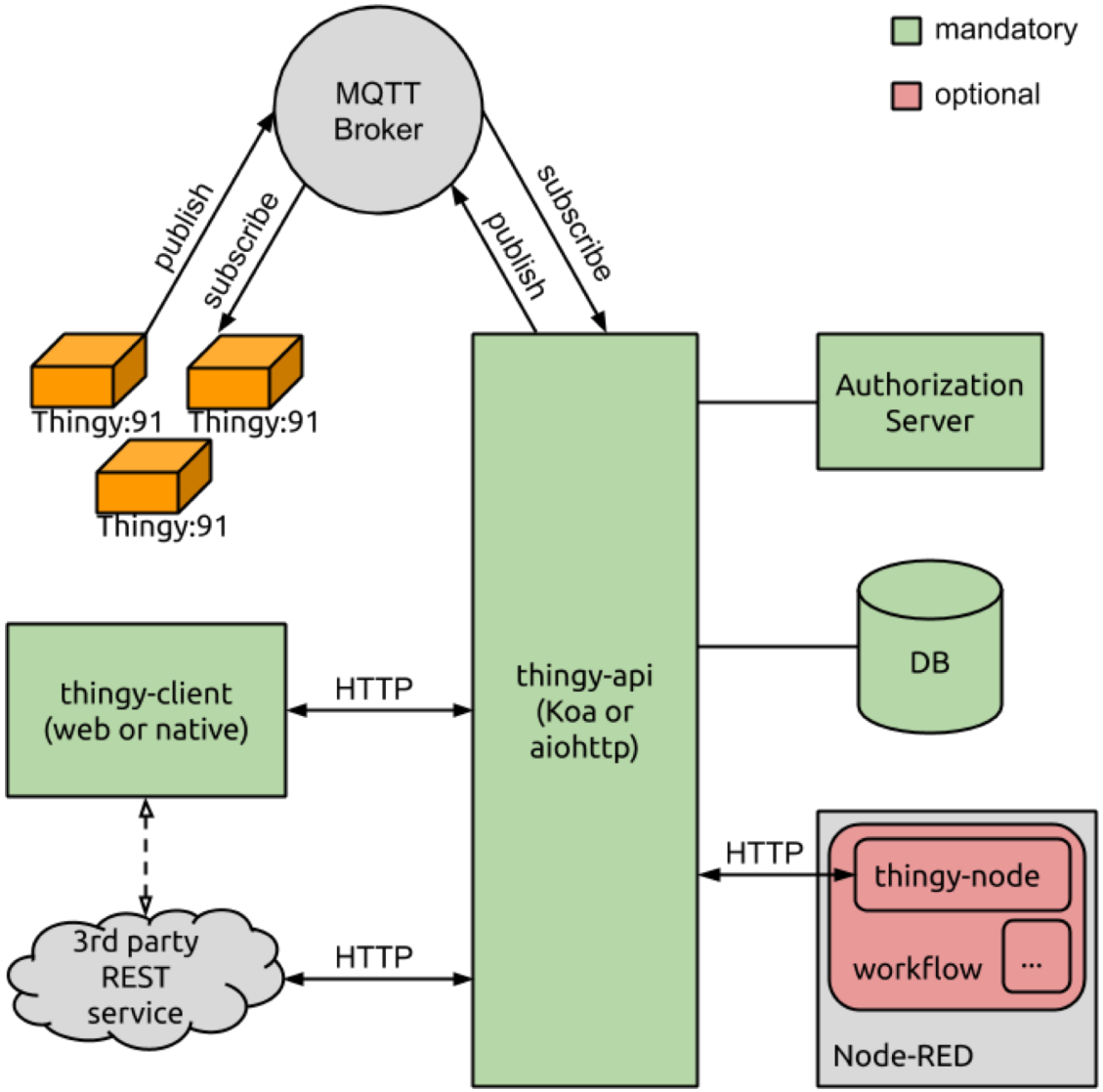
Thingy Project 18 = Example - 3 of Client-Server System



Thingy Project 19 = Example - 4 of Client-Server System

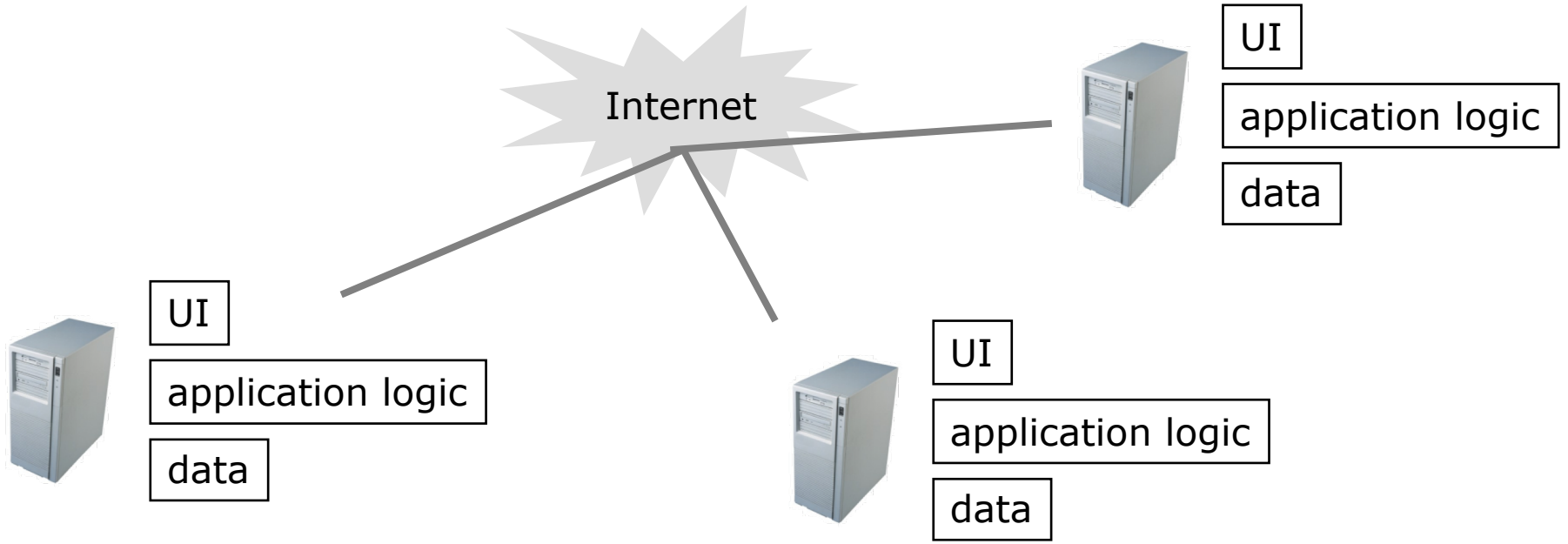


Thingy Project 20 = Example - 5 of Client-Server System



Peer-to-Peer Architecture (P2P)

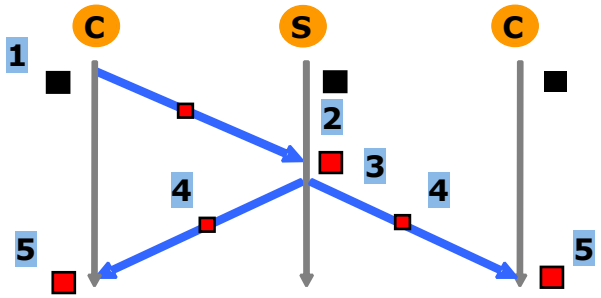
- Peer
  - the user's machine (site) runs a complete application
    - complete application logic
    - responsible for data management
  - directly communicates with all other peers



Data Management: C/S vs. P2P

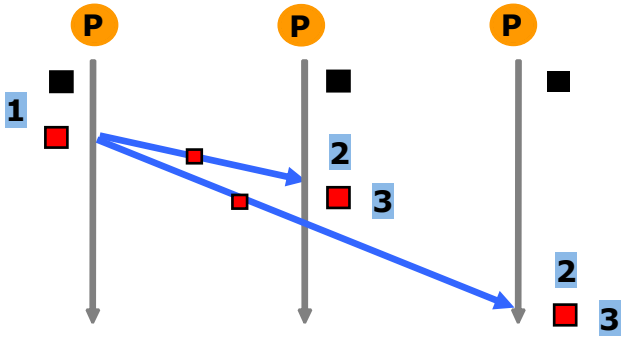
Centralized (C/S)

- server manages master copy of state S
- state changes
  1. client sends operation to server
  2. server collects and serializes all operations
  3. server executes state change
  4. server sends state update to all clients
  5. clients display updated state



Distributed (P2P)

- each site manages a copy of state S
- state changes
  1. site executes state change, displays updated state and sends operation to all other sites
  2. each site collects and serializes all operations
  3. each site executes state change and displays updated state



Client-Server vs. Peer-to-Peer

Client-Server
+ simple control, data management, and communication
+ central administration of hardware and software
+ debugging and updates
+ run costly processes / store large data at a powerful machine
+ fairness / cheat control
+ accounting
- performance bottleneck w.r.t. processing load, storage, and traffic
- single point of failure
- administration overhead

"A distributed system is one that stops you from getting any work done when a machine you've never heard of crashes", L. Lamport

Peer-to-Peer
- inverse to client-server

Hybrid Architecture
- C/S often have partially distributed functionality or data
  - e.g., server farm for load-balancing

Hybrid of Client-Server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging

- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Literature

- IETF RFC Editor (http://www.rfc-editor.org/rfc.html)
- James F. Kurose, Computer Networking: A Top-Down Approach, Addison-Wesley, 2010
- Also the 5-6 first chapters of Andreas Ruppen thesis : A Model-Driven, Component Generation Approach for the Web of Things, Thesis no 1905, 2015, UniFr
  Provides a good background about the Internet, the HTTP protocol and RESTful web services, with a lot of references.
  Available under : http://diuf.unifr.ch/drupal/softeng/publications
- Another really great book, which goes far beyond this course is the one of: Dominique D. Guinard and Vlad M. Trifa, Building the Web of Things, Maning, 2016.