MONASH University

# Introduction to mobile and distributed computing

Lecture 1 – Part II

FIT5046: Mobile and Distributed Computing Systems

GROUP OF EIGHT AUSTRALIA

# Today's Lecture

- An Overview of Distributed Systems
- An Overview of Mobile and Distributed Computing
- Web Services

# Distributed Computing

- A computing paradigm where **a number** of **autonomous** entities (most likely **heterogeneous)**

  which are **geographically distributed**

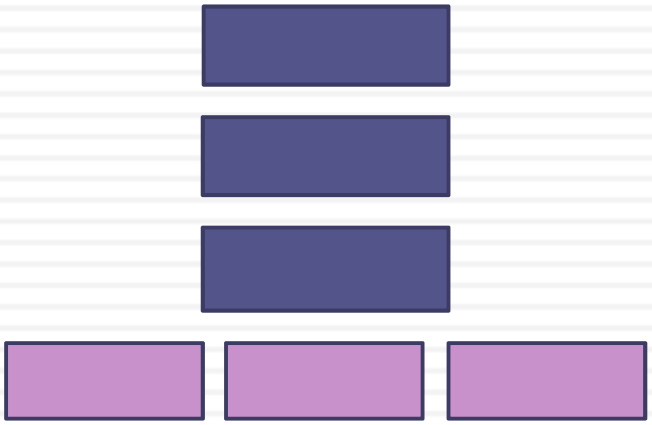
  can **communicate and exchange messages**

  **through a computer network**

  to **achieve certain related tasks** (common goals)

# Vertical and Horizontal Distribution

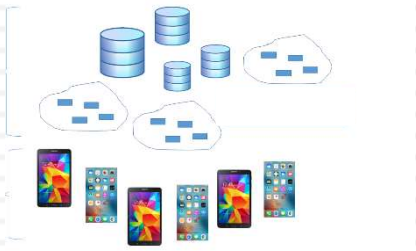
- Vertical distribution: placing logically different layers/components on different machines
  - Each layer on one single machine
- Horizontal distribution: a single logical layer/component is distributed across multiple machines to improve scalability
  - E.g. distributing a database on multiple machines (distributed database)



Source: Tanenbaum & Van Steen (2014) Distributed Systems: Principles and Paradigms

# Mobile and Distributed Computing

- It is a class of distributed computing systems
- It integrates mobile and wireless devices into distributed systems
  - Wireless sensors, wearables, smartwatches, smartphones, tables, and smart things (e.g. smart refrigerator)
- Mobile computing is associated with mobility of hardware, users, data, applications and network in computer applications
- Mobile computing is possible because of wireless communication technologies:
  - Cellular networks (4G, 5G), WiFi, WiMAX, Bluetooth, ZigBee, NFC, RFID, …

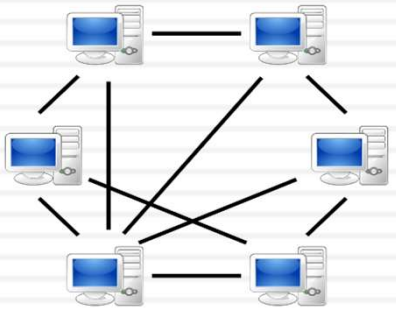


# Emerging Computing Paradigms

- Ubiquitous computing and IoT (week 8 lecture)
  - Context-aware computing, situation-aware computing
  - IoT (Internet of Things) and smart cities
  - Cloud computing
  - Edge or fog computing
- Mobile Sensing (week 9 lecture)
  - Wireless sensors
  - Wireless sensor networks (WSNs)
- Location-aware systems (week 10 lecture)

# Distributed Computing Models

- The client/server model
  - Server processes offer services to clients processes
  - Usually there is a data storage at the backend
- Peer-to-peer
  - Each process logically equal to each other
  - Data flows between the processes

Source: Wikipedia

# FLUX Quiz

- Go to this link: https://flux.qa/C32HCB
- Sign up/in with Monash email as shown below
- Click on FIT5046 Lecture 1

# Quiz Question 1

# Distributed Computing and SOA (Service-Oriented Architecture)

- Service-oriented architecture was introduced as a paradigm for distributed systems
  - Application functionalities (software components) are provided as **services** (independent modules)
  - Exposed to public (clients) using a **standard interface** protocol, aka an application programming interface (API)
  - **Message based** interactions through these interfaces
  - **Reuse** of services and **composition** of services
  - **Interoperability** to support different platforms

# Web Services



- SOA is implemented by creating web services
- "A Web service is a piece of software/code designed to support interoperable machine-to-machine interaction over a network" (W3C)
- Web services provide a standard interface to make the functionalities available to the public (clients)
- Web services provide access to business logic, data and processes or other services
- Web services were originally implemented as SOAP web services and later evolved into RESTful web services (RESTful Web APIs)

# REST (REpresentational State Transfer)

- RESTful web service were emerged based on the REST architecture's concept (introduced by Roy Fielding)

REST slides are adopted from:

- **Roy Fielding's PhD thesis**
  http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

# REST is an architecture

- REST is not a protocol, a technology, a standard, or a specification
- The architecture consists of elements and relationships between these elements
- The REST architecture's constraints that control the roles the roles/features of these elements and also their allowed relationships
  - Architectural Constraints
  - Interface Constraints
- While REST is not a standard, it uses standards:
  - HTTP
  - URL
  - XML/HTML/JSON/etc
  - MIME Types or Media types such as text/xml, image/gif, application/json, audio/mpeg

# Quiz Question 2

# Architectural Constraints



- Client/Server
- Stateless
- Cache
- Uniform Interface
- Layered Systems
- Code-On-Demand

# Architectural Constraints - 1

## Client-Server

- the client-server architectural style
  - separates the user interface concerns of clients from the data storage concerns of servers
  - improves the portability of the user interface across multiple platforms
  - improves scalability by simplifying the server components (not concerned about the user interface)

# Architectural Constraints - 2



**Stateless**

- "communication must be stateless in nature"
  - each request from the client to the server must contain all the information necessary to understand the request
  - The server does not store the client's context*
  - Improves scalability and reliability

# Architectural Constraints - 3



## Cache

- "…the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable."
  - If a response is cacheable, the response can be reused for equivalent requests later
  - + Improves network efficiency and performance
  - - decreases reliability (possibility of stale data)

# Architectural Constraints - 4



## Uniform Interface

- all resources are accessed with a generic interface (e.g., HTTP **GET, POST, PUT, DELETE**)
  - PATCH is an HTTP method that enables updating part of the resources
  - HEAD is similar to GET but with no response body
  - OPTIONS is a request for information about the communication options
- the overall system architecture is simplified

# Architectural Constraints - 5



**Layered System**

- allows an architecture to be composed of hierarchical layers
  - each component cannot "see" beyond the immediate layer with which they are interacting
  - Clients have no knowledge that services they invoke may also invoke other services

# Architectural Constraints - 6



**Code-On-Demand**

- an optional constraint
- “allows client functionality to be extended by downloading and executing code in the form of applets or scripts.”

# REST and Resources



## Identification of resources

- A **resource:**
  - Any information that can be named can be a resource
    - E.g. a person, an object, a service
    - Nouns instead of verbs
- A **resource identifier**
  - Each resource becomes accessible via a URI/URL

| ID | URI |
|---|---|
| 1 | customers/1/ (http://localhost:8080/CustomerDB/resources/customers/1/) |
| 2 | customers/1/discountCode/ (http://localhost:8080/CustomerDB/resources/customers/1/discountCode/) |
| 3 | customers/2/ (http://localhost:8080/CustomerDB/resources/customers/2/) |

# **RE**presentational **S**tate **T**ransfer



- A **representation:**
  - It is a document capturing the current state of a resource
  - A resource can have different representations (e.g. JSON or XML)

- **REST** (**RE**presentational **S**tate **T**ransfer):
  - each resource state has a representation, and this representation can be updated and transferred from the server to the client application

# Quiz Question 3

# JSON



- JSON stands for JavaScript Object Notation
- JSON is lightweight text-data interchange format
- JSON is "self-describing" and easy to understand
- JSON supports two structures:
  - Objects: a collection of name/value pairs
    - {"firstName": "John"}
  - Arrays: an ordered list of values

```
{"phoneNumber": [
    {
      "type": "home",  "number": "212 555-1234"
    },
    {
      "type": "fax",  "number": "646 555-4567"
    }  ] }
```

# JSON (cont'd)

- Objects in name/value pairs , each name is followed by a colon
- A value can be a string, a number, true/false or null, an object or an array
- Data is separated by commas
- Curly braces hold objects and square brackets hold arrays

```
{   "firstName": "John",    "lastName": "Smith",   "age": 25,    "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": 10021
   },
   "phoneNumber": [
      {
         "type": "home",  "number": "212 555-1234"
      },
      {
         "type": "fax",  "number": "646 555-4567"
      }   ]
}
```

# JSON Data Types

- a string { "name":"John" }
- a number { "age":30 }
- an object (JSON object) {

```
"address": {
      "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021    },
```

- an array

```
{"phoneNumber": [
        {
          "type": "home",  "number": "212 555-1234"
        },
        {
          "type": "fax",  "number": "646 555-4567"
        }  ] }
```

- a Boolean { "sale":true }
- null { "middlename":null }

https://www.w3schools.com/js/js_json_datatypes.asp

# Parsing JSON



- JSON parsing online
  - http://json.parser.online.fr/
  - https://jsoneditoronline.org/

{ "firstName": "John", "lastName": "Smith", "age": 25, "address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021 },"phoneNumbers": [ {"type": "home", "number": "212 555-1234" }, {"type": "fax", "number": "646 555-4567" } ] }

# Parsing JSON (cont'd)

- There are libraries to create and parse JSON such as Google Gson libraries
- In Android, we will use org.json libraries
  - *import org.json.JSONObject;*
  - The JSONObject class is used to create or parse JSON

```
JSONObject jsonObject = new JSONObject(result);
 JSONArray jsonArray = jsonObject.getJSONArray("items");
 if(jsonArray != null && jsonArray.length() > 0) {
     snippet =jsonArray.getJSONObject(0).getString("snippet");
}
```

*Check the Google response structure here:*
*https://developers.google.com/custom-search/v1/reference/rest/v1/Search*
*https://developers.google.com/custom-search/json-api/v1/reference/cse/list#response*

# Quiz Question 4