UDP Programming

RES, Lecture 3

Olivier Liechti

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Lab 02 Feedback

I messed up my branch... what do I do?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Scenario**: you have created a feature branch, made some commits. You would like to submit a Pull Request, but there are "dirty" files around. How do you clean up the situation?

```
$ git checkout -b feature_branch_ABC
$ echo ‘a’ > a.txt
$ git add a.txt
$ git commit -m “This is a commit I want to keep”
$ echo ‘b’ > b.txt
$ git add b.txt
$ git commit -m “This is a commit I want to get rid of”
$ git push -u origin feature_branch_ABC
```

- **Solutions**: there are different ways, and the choice depends on your situation.
- If the "dirty" files are confined in specific commits, you can use `git revert`.
- If it is really messy, then it might be better to **create a new branch**.

I messed up my branch... what do I do?

- **If you create a new branch**, then be careful to create if from master and not from the dirty feature branch.

```
$ git checkout master
$ git checkout -b feature_branch_ABC_clean
```

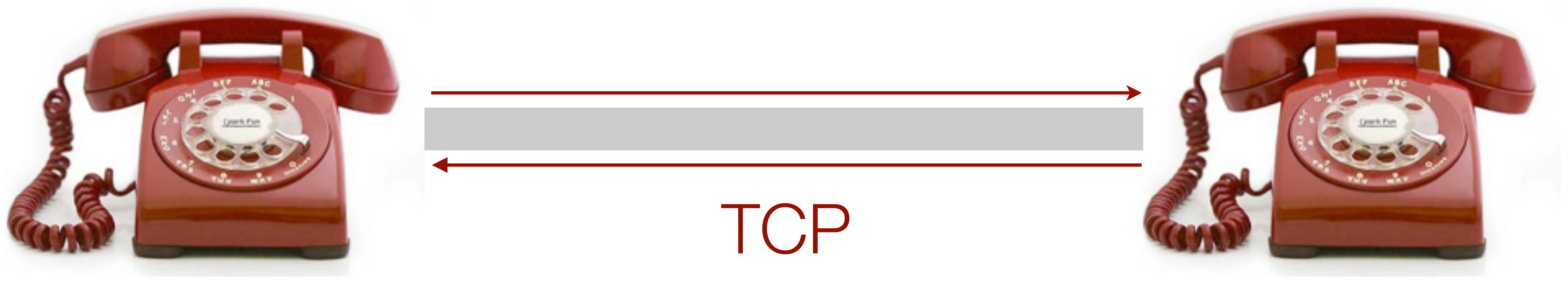
- **With git checkout**, you can get files from other branches

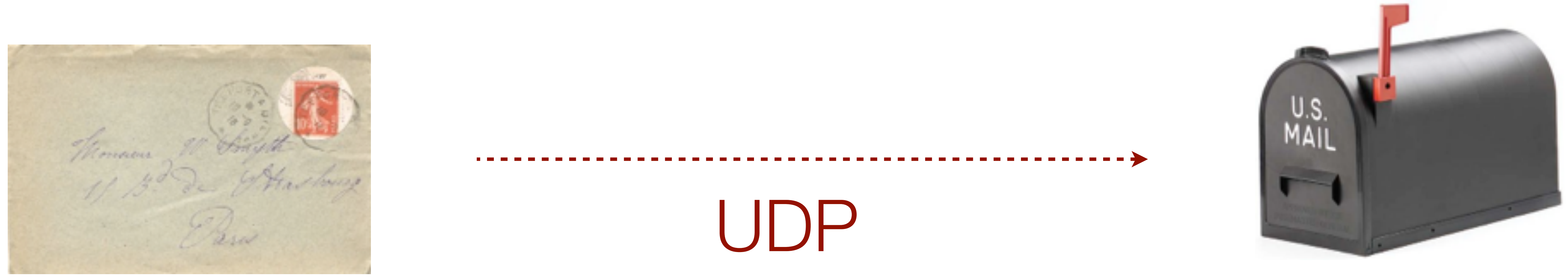
```
$ feature_branch_ABC
$ git checkout -b feature_branch_ABC_clean a.txt
```

- **You can then do a clean commit and pull request**

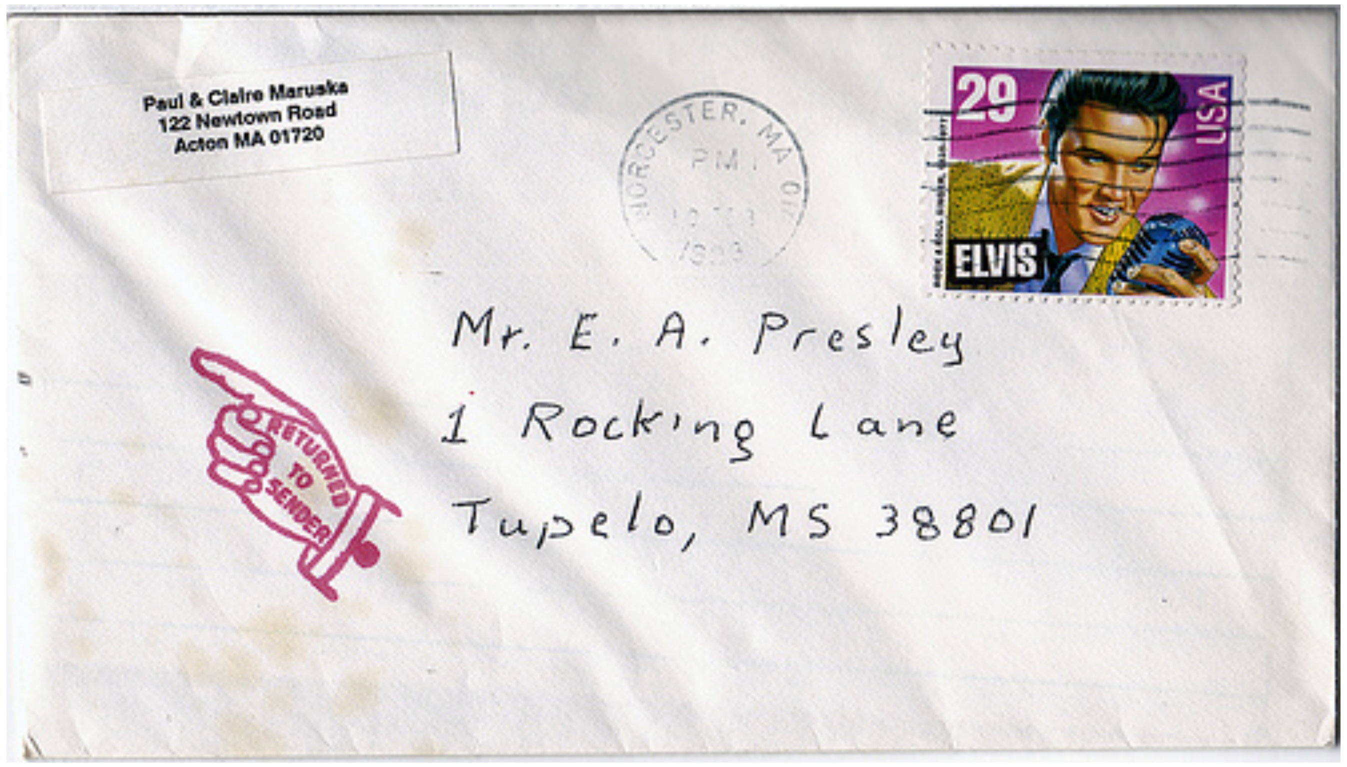
```
$ git add a.txt
$ git commit -m “My feature is complete.”
$ git push -u origin feature_branch_ABC_clean
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud





The UDP Protocol



The UDP Protocol

- When using UDP, **you do not work with the abstraction of an IO stream**.
- Rather, **you work with the abstraction of individual datagrams**, which you can send and receive. Every datagram is independent from the others.
- You **do not have any guarantee**: datagrams can get lost, datagrams can arrive out-of-order, datagrams can be duplicated.
- What do you put "**inside**" the datagram (i.e. what is the payload)?
    - A notification.
    - A request, a query, a command. A reply, a response, a result.
    - A portion of a data stream (managed by the application).
- What do you put "**outside**" of the datagram (i.e. what is the header)?
    - A destination address (IP address in the IP packet header + UDP port)
    - A source address (IP + port)

Unicast, Broadcast, Multicast

Destination Address

192.168.10.2

255.255.255.255

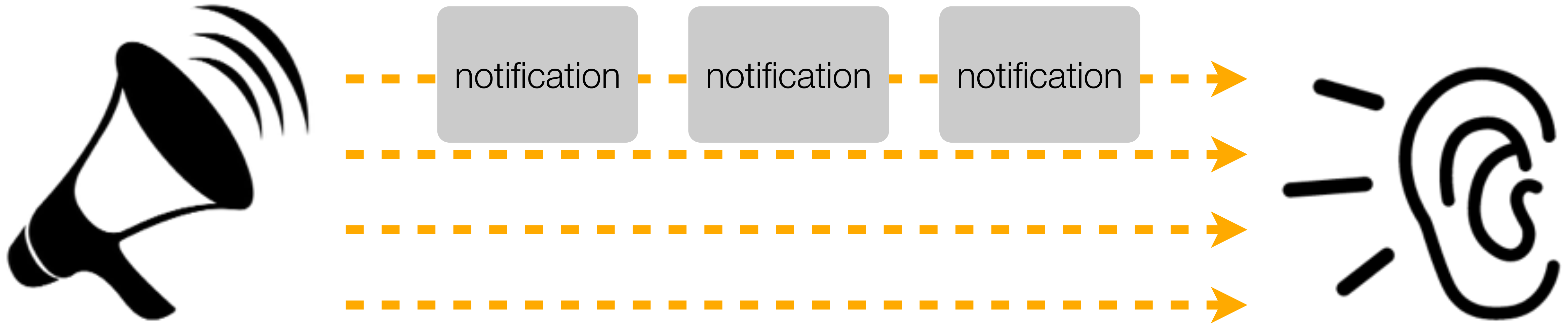
239.255.22.5

IANA Guidelines for IPv4 Multicast Address Assignments
http://tools.ietf.org/html/rfc5771

Messaging Patterns

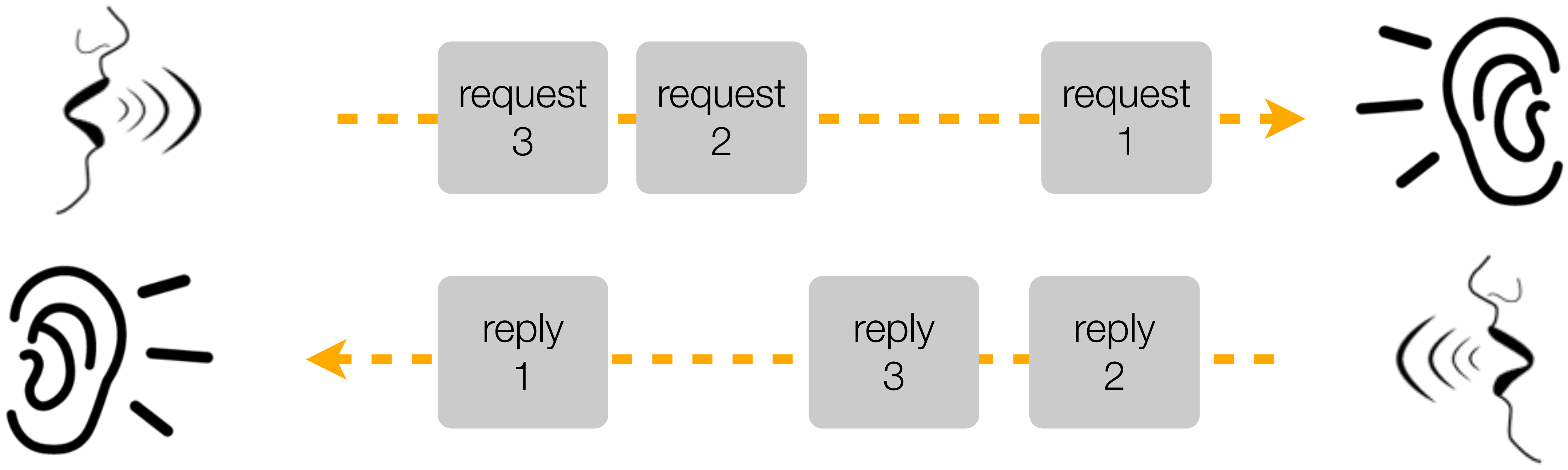
Pattern: Fire-and-Forget

- The sender generates messages, sends them to a single receiver or to a group of receivers.
- The sender may do that on a **periodic** basis.
- The sender does not expect any answer. **He is telling, not asking**.



Pattern: Request-Reply

- This pattern is used to implement the **typical client-server model**. Both the client and the server **produce and consume** datagrams.
- **The client produces requests** (aka queries, commands). The client also listens for incoming replies (aka responses, results). The server listens for requests. **The server also sends replies back to the client**.
- Can the client send a new request, even if he has not received the response to the previous request yet? If yes, and because UDP datagrams can be delivered out of order, how can the client **associate a reply** with the corresponding request?



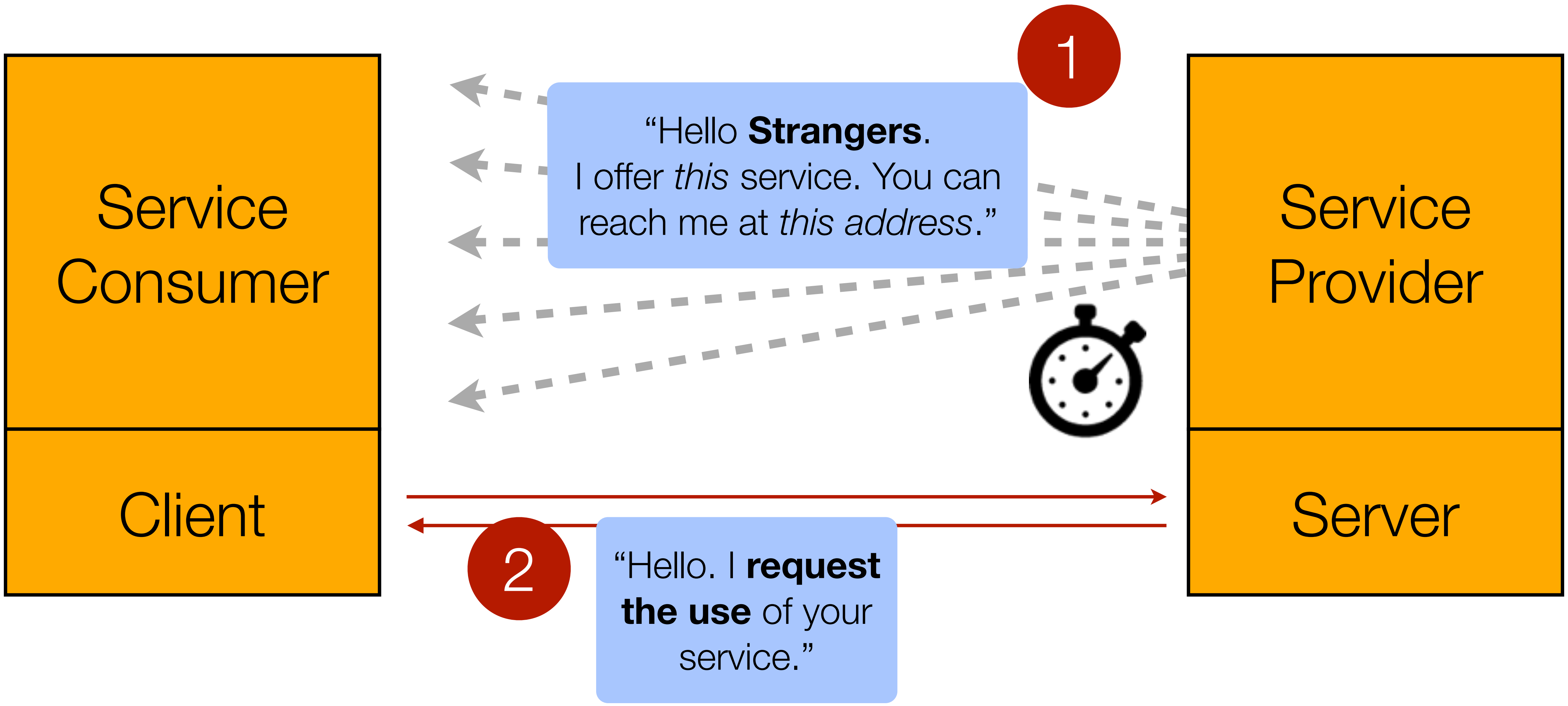
Service Discovery Protocols

Service Discovery Protocols

- With **unicast** message transmission, the sender must know who the receiver is and must know how to reach it (i.e. know its address).
- With **broadcast** and **multicast**, this is not required. The sender knows that nodes that are either *nearby*, or that have *expressed their interest*, will receive a copy of the message.
- This property can be used to **discover the availability of services or resources** in a *dynamic environment*.

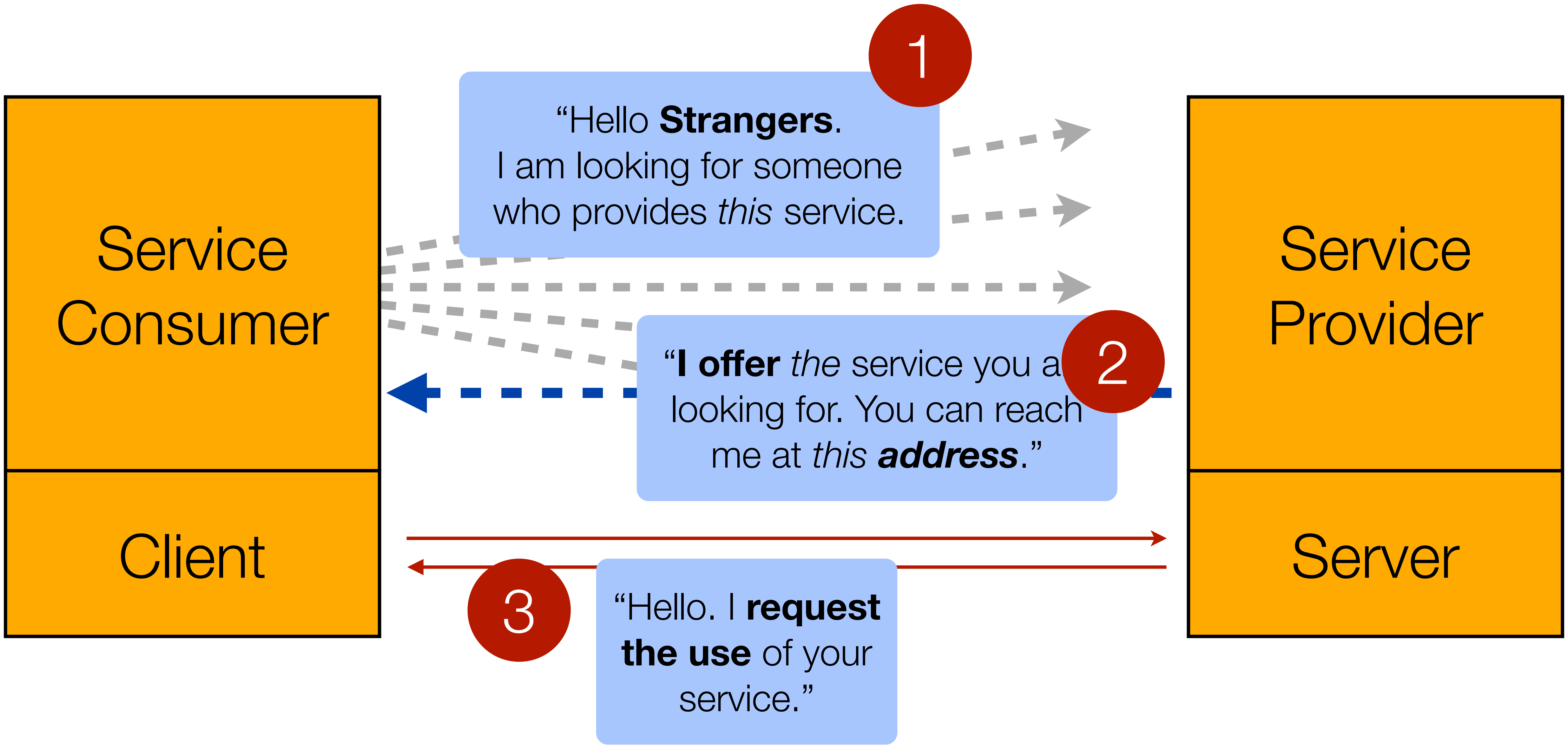
Model 1: Advertise Service Periodically

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud


Service Consumer
Client
Service Provider
Server
1
"Hello **Strangers**. I offer *this* service. You can reach me at *this address*."
2
"Hello. I **request the use** of your service."

Model 2: Query Service Availability

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud


1
"Hello **Strangers**.
I am looking for someone
who provides *this* service.
Service Consumer
Client
2
"**I offer** *the* service you are looking for. You can reach me at *this* ***address***."
Service Provider
Server
3
"Hello. I **request the use** of your service."

Reliability

Reliability

- UDP is **not a reliable transport protocol**:
  - Datagrams can arrive in a different order from which they were sent.
  - Datagrams can get lost.
- For some application-level protocols, this is not an issue because they are **tolerant to data loss**. For example, think of a **media streaming** protocol.
- But **if no data loss can be tolerated at the application level** (which is typically the case for file transfer protocols), does it mean that it is impossible to use UDP?

Who Is Responsible for Reliability?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **It is actually possible** to implement reliable application-level protocols on top of UDP, but...
    - It is the **responsibility of the application-level protocol specification** to include appropriate mechanisms to overcome the limitations of UDP.
    - It is the **responsibility of the application developer** to implement these mechanisms.
- In other words, it is up to the application developer to "**do the kind of stuff that TCP usually does**".
- Do you remember about **acknowledgments**, **timers**, **retransmissions**, **stop-and-wait**, **sliding windows**, etc?

A First Example: TFTP

- **Trivial File Transfer Protocol (TFTP)**
  - http://tools.ietf.org/html/rfc1350
  - “TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. **Each nonterminal packet is acknowledged separately.** This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.”
  - “Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet **before** the next packet can be sent.”

A Second Example: CoAP

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Constrained Application Protocol (CoAP)**
  - http://tools.ietf.org/html/draft-ietf-core-coap-18
  - "The Constrained Application Protocol (CoAP) is a **specialized web transfer protocol** for use with constrained nodes and constrained (e.g., low-power, lossy) networks. [...] The protocol is designed for **machine-to-machine (M2M) applications** such as smart energy and building automation."
  - "CoAP provides a **request/response interaction model** between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types."
  - "**2.1. Messaging Model**. The CoAP messaging model is based on the exchange of messages over UDP between endpoints. CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. **Each message contains a Message ID used to detect duplicates and for optional reliability.** (The Message ID is compact; its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters.)
  - "**4.2. Messages Transmitted Reliably**. The reliable transmission of a message is initiated by marking the message as Confirmable in the CoAP header. [...] A recipient MUST **acknowledge** a Confirmable message with an Acknowledgement message [...].

Using the Socket API for a UDP **Sender**

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
1. Create a datagram socket (without giving any address / port)

2. Create a datagram and put some data (bytes) in it

3. Send the datagram via the socket, specifying the destination
address and port

If a reply is expected:

4. Accept incoming datagrams on the socket
```

If we do not specify the port when creating the socket, the operating system will **automatically assign a free one** for us.

This port will be in the “source port” field of the UDP header. The receiver of our datagram will extract this value and will use it to send us the reply (it is a part of the ***return address***.

Using the Socket API for a UDP **Receiver**

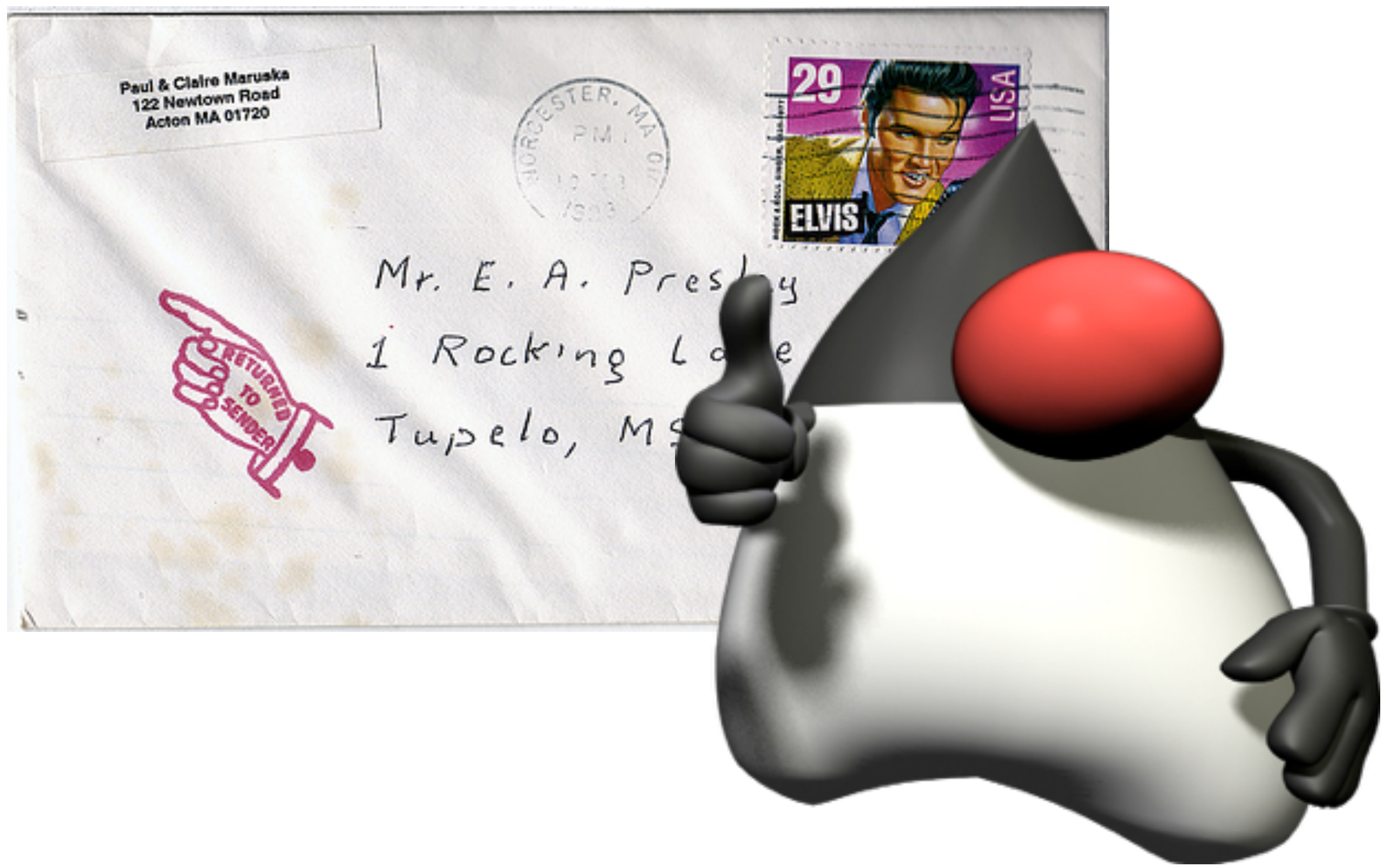
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Application-level protocol specifications often define **a standard UDP port**, where the clients can send their requests. When this is the case, we have to specify the port number when creating the socket.

```
1. Create a datagram socket, specifying a particular port

2. Loop
   2.1. Accept an incoming datagram via the socket
   2.2. Process the datagram
   2.3. If a reply is expected by the sender
        2.3.1. Extract the return address from the request datagram
        2.3.2. Prepare a reply datagram
        2.3.3. Send the reply datagram via the socket
```

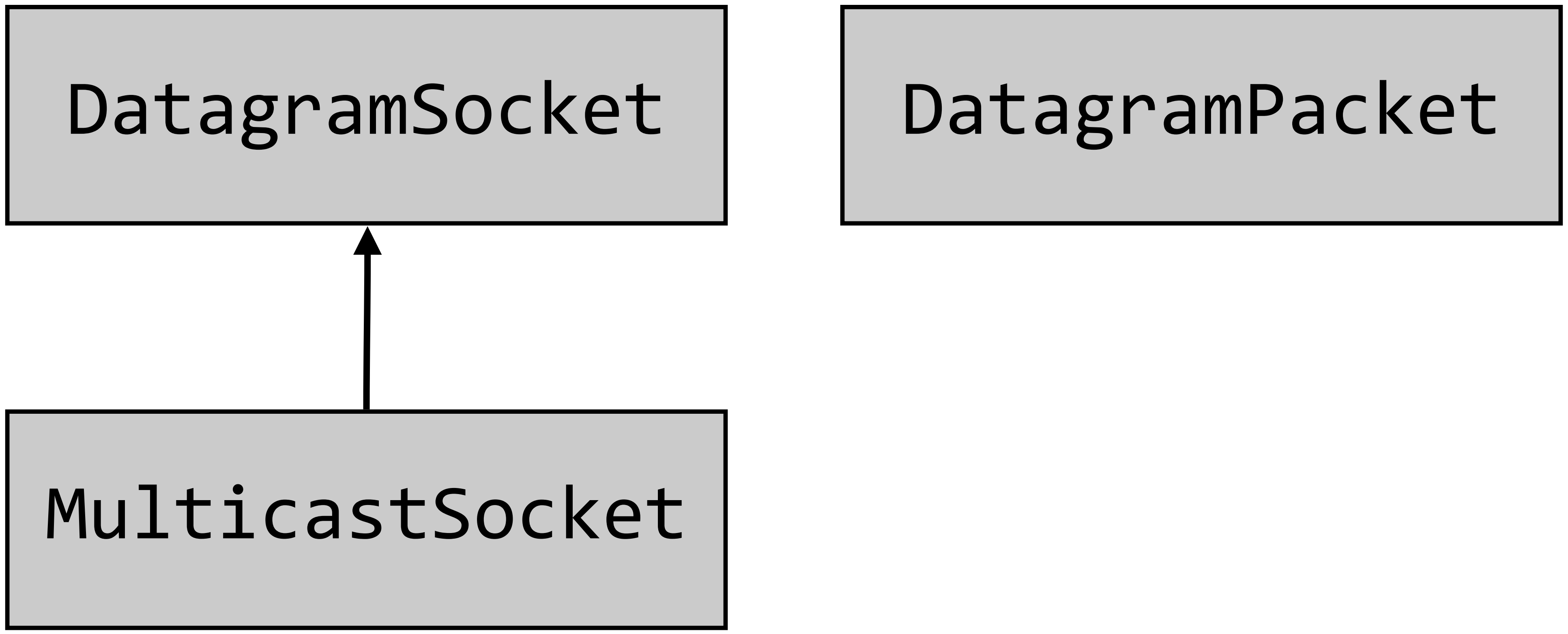
It is at this stage that we extract the source port and source IP address from the incoming datagram. We have found the ***return address***, so we know where to the send the response.

Using UDP in Java



Using UDP in Java

`java.net`



```java
// Sending a message to a multicast group

socket = new DatagramSocket();
byte[] payload = "Java is cool, everybody should know!!".getBytes();

DatagramPacket datagram = new DatagramPacket(payload, payload.length,
InetAddress.getByName("239.255.3.5"), 4411);

socket.send(datagram);
```

Publisher (multicast)

Subscriber (multicast)

```java
// Listening for broadcasted messages on the local network

MulticastSocket socket = new MulticastSocket(port);
InetAddress multicastGroup = InetAddress.getByName("239.255.3.5");
socket.joinGroup(multicastGroup);

while (true) {
    byte[] buffer = new byte[2048];
    DatagramPacket datagram = new DatagramPacket(buffer, buffer.length);
    try {
        socket.receive(datagram);
        String msg = new String(datagram.getData(), datagram.getOffset(), datagram.getLength());
        LOG.log(Level.INFO, "Received a datagram with this message: " + msg);
    } catch (IOException ex) {
        LOG.log(Level.SEVERE, ex.getMessage(), ex);
    }
}
socket.leaveGroup(multicastGroup);
```

```java
// Broadcasting a message to all nodes on the local network

socket = new DatagramSocket();
socket.setBroadcast(true);

byte[] payload = "Java is cool, everybody should know!!".getBytes();

DatagramPacket datagram = new DatagramPacket(payload, payload.length,
InetAddress.getByName("255.255.255.255"), 4411);

socket.send(datagram);
```

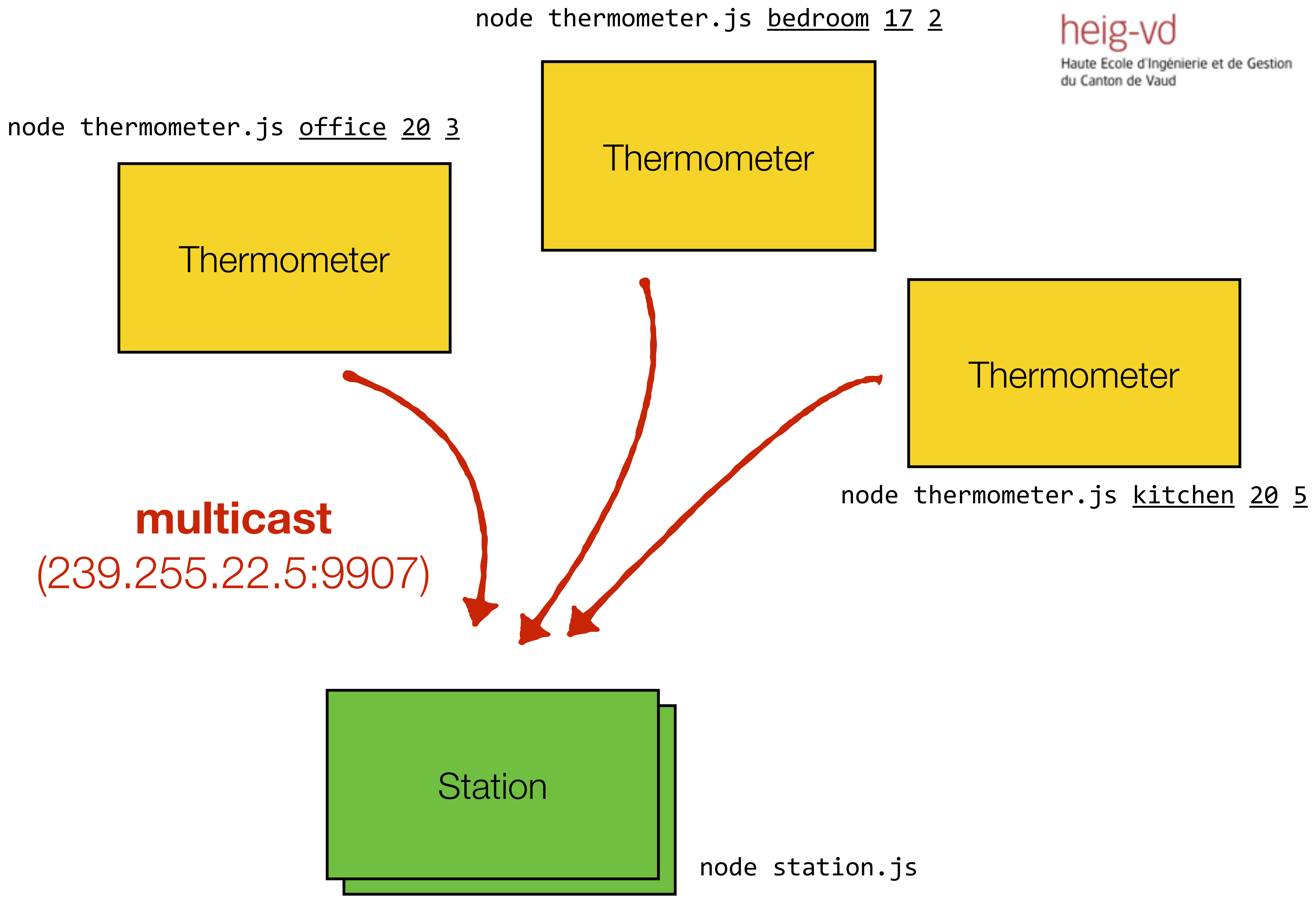
Publisher

Subscriber

```java
// Listening for broadcasted messages on the local network

DatagramSocket socket = new DatagramSocket(port);

while (true) {
    byte[] buffer = new byte[2048];
    DatagramPacket datagram = new DatagramPacket(buffer, buffer.length);
    try {
        socket.receive(datagram);
        String msg = new String(datagram.getData(), datagram.getOffset(), datagram.getLength());
        LOG.log(Level.INFO, "Received a datagram with this message: " + msg);
    } catch (IOException ex) {
        LOG.log(Level.SEVERE, ex.getMessage(), ex);
    }
}
```

Example: **09-thermometers**

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



```
node thermometer.js bedroom 17 2
node thermometer.js office 20 3
node thermometer.js kitchen 20 5
node station.js
```

thermometer.js (fragment)

```
// We use a standard Node.js module to work with UDP
var dgram = require('dgram');

// Let's create a datagram socket. We will use it to send our UDP datagrams
var s = dgram.createSocket(‘udp4');

// Create a measure object and serialize it to JSON
var measure = new Object();
measure.timestamp = Date.now();
measure.location = that.location;
measure.temperature = that.temperature;
var payload = JSON.stringify(measure);

// Send the payload via UDP (multicast)
message = new Buffer(payload);
s.send(message, 0, message.length, protocol.PROTOCOL_PORT, protocol.PROTOCOL_MULTICAST_ADDRESS,
function(err, bytes) {
console.log("Sending payload: " + payload + " via port " + s.address().port);
});
```

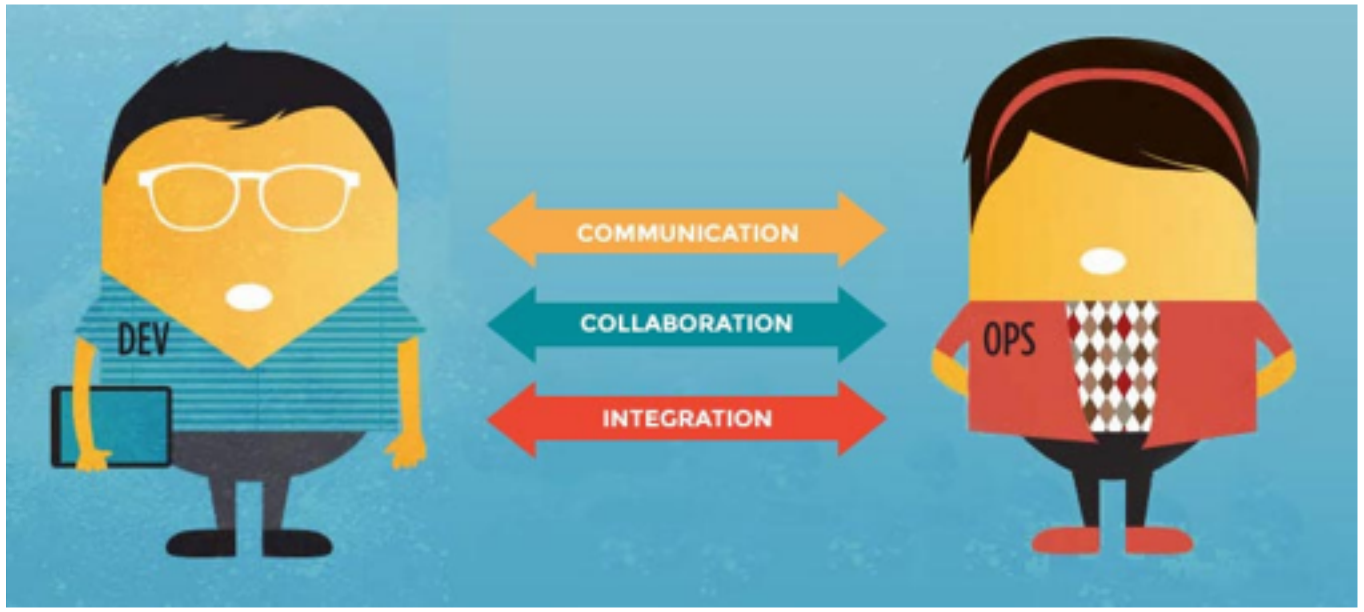
station.js (fragment)

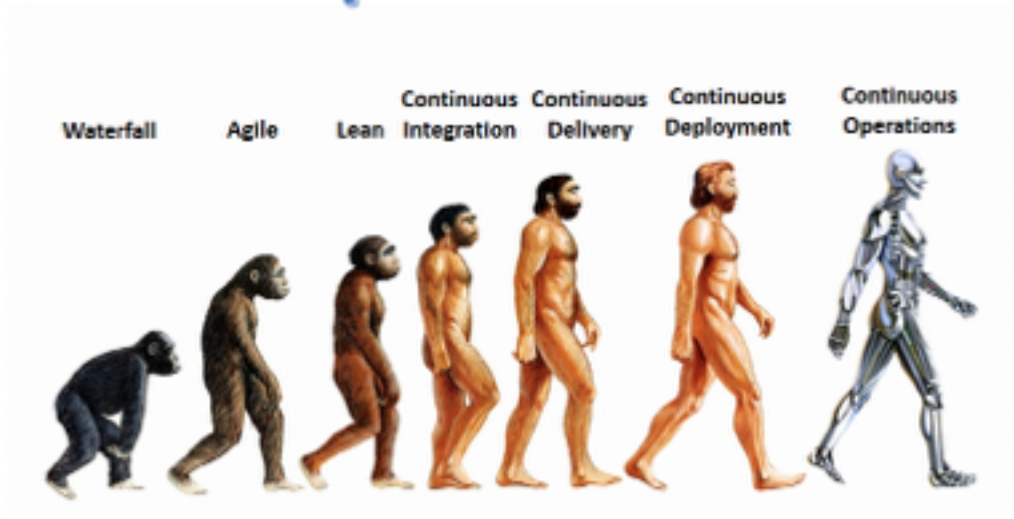
```
// We use a standard Node.js module to work with UDP
var dgram = require('dgram');

// Let's create a datagram socket. We will use it to listen for datagrams published in the
// multicast group by thermometers and containing measures
var s = dgram.createSocket('udp4');
s.bind(protocol.PROTOCOL_PORT, function() {
  console.log("Joining multicast group");
  s.addMembership(protocol.PROTOCOL_MULTICAST_ADDRESS);
});

// This call back is invoked when a new datagram has arrived.
s.on('message', function(msg, source) {
        console.log("Data has arrived: " + msg + ". Source IP: " + source.address + ". Source
port: " + source.port);
});
```

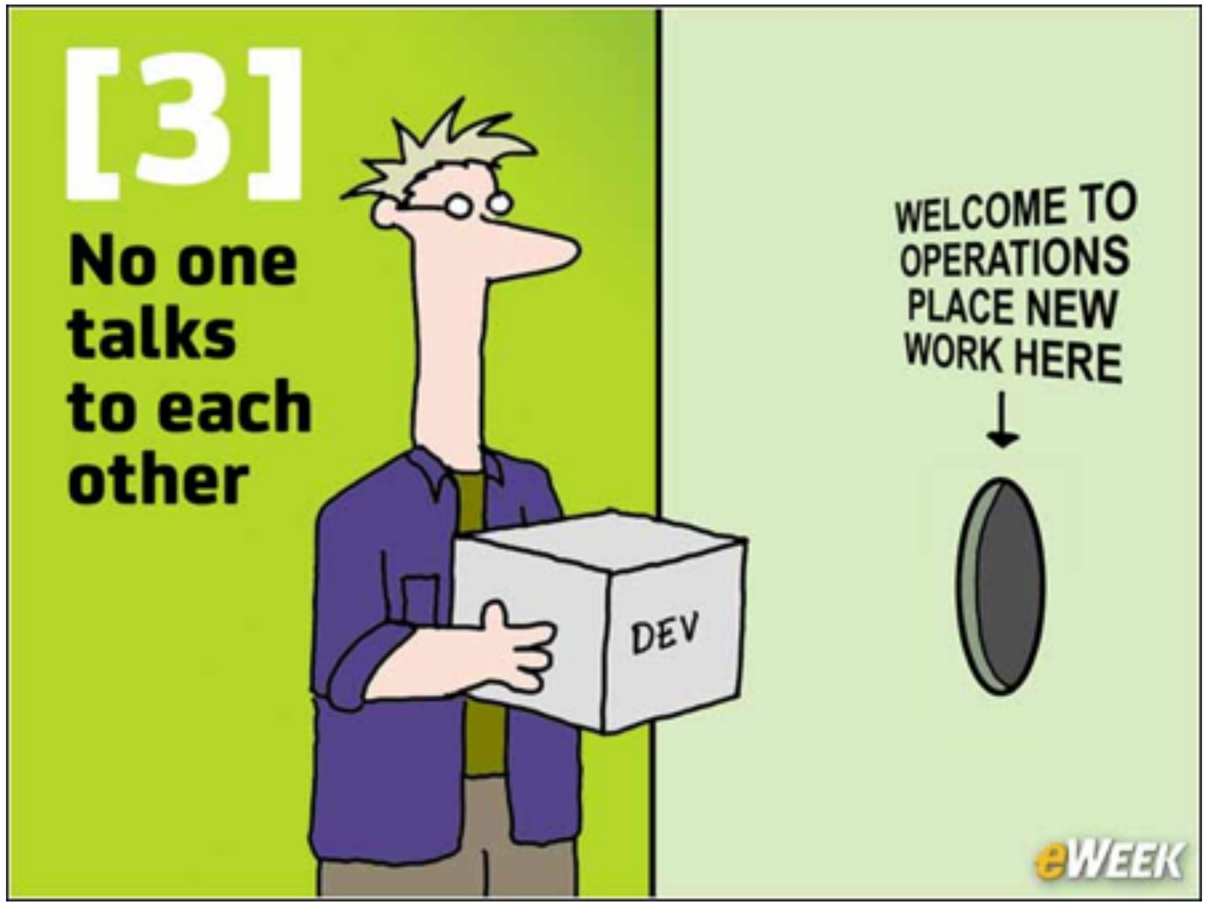
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

docker

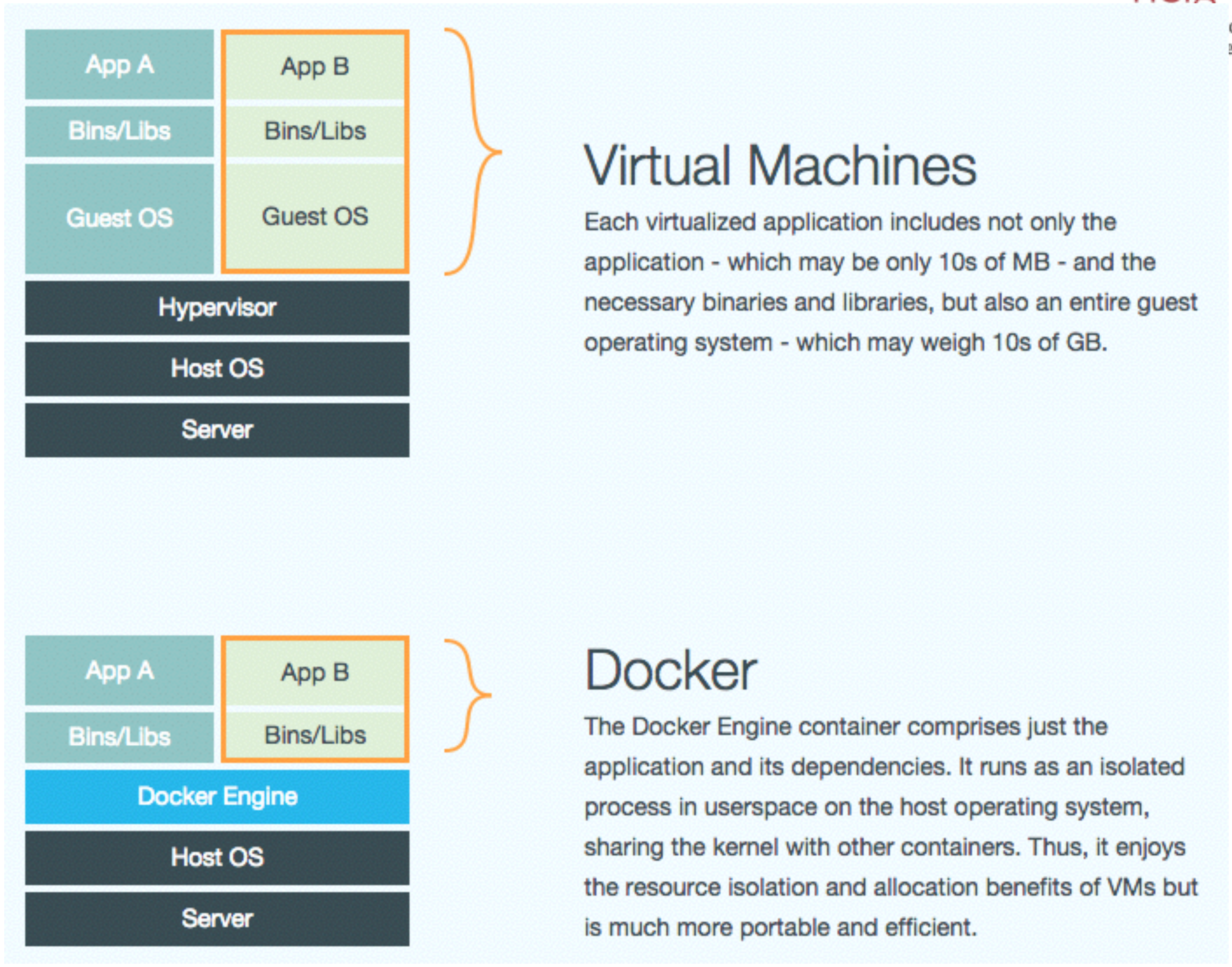
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud






[3]
No one talks to each other
DEV
WELCOME TO OPERATIONS PLACE NEW WORK HERE
eWEEK

I want change!
Wall of Confusion
I want stability!
Development
Operations



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.

Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

https://www.docker.com/whatisdocker/

Docker containers (1)

- You can think of a container as a **lightweight** virtual machine. Each container is isolated from the others and has its own IP address.
- Each container is created from a docker image (one could say that a container is a **running instance of an image**).
- There are **commands** to start, list, stop and delete containers.

```
# Start a container (more on this later)
$ docker run

# List running containers
$ docker ps

# List all containers
$ docker ps -a

# Delete a container
$ docker rm

# Display logs produced by a container
$ docker logs
```

Docker containers (2)

- With Docker, the philosophy is to have **one application-level service per container** (it is not a strict rule).
- With Docker, the philosophy is also to **run several (many) containers on the same machine**.
- If you think of a typical **Web Application infrastructure**, you would have one or more containers for the apache web server, one container for the database, one container for the reverse proxy, etc.
- With Docker, containers tend also to be **short-lived**. Each container has an **entry point**, i.e. a command that is executed at startup. When this command returns, the container is stopped (and will typically be removed).
- **If a container dies, it should not be a big deal**. Instead of trying to fix it, one will create a new one (from the same image).

Docker images (1)

- **A Docker image is a template**, which is used to create containers.
- Every image is **built from a base image** and adds its own configuration and content.
- With Vagrant, we use a file named Vagrantfile to configure and provision a Vagrant box. With Docker, we use a file name **Dockerfile** to create an image. The file contains statements (FROM, RUN, COPY, ADD, EXPOSE, CMD, VOLUME, etc.)
- Just like the community is sharing Vagrant boxes, **the community is sharing Docker images.** This happens on the Docker Hub registry (https://registry.hub.docker.com/).

Docker images (2)

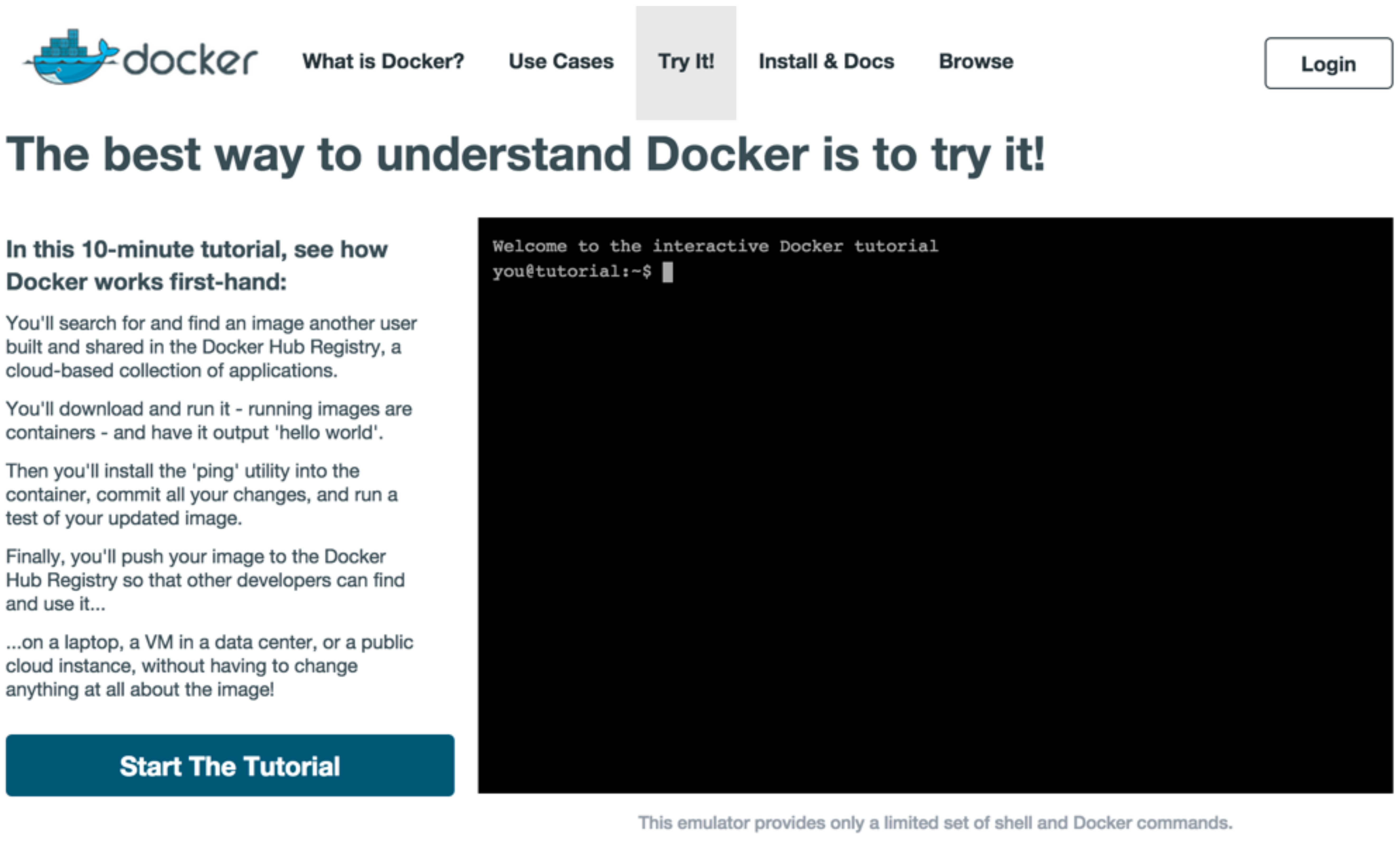
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Here is an example for a Dockerfile (used for first experiments, does not

```
# This image is based on another image

FROM dockerfile/nodejs:latest

# For information: who maintains this Dockerfile?

MAINTAINER Olivier Liechti

# When we create the image, we copy files from the host into
# the image file system. This is NOT a shared folder!

COPY file_system /opt/res/

# With RUN, we can execute commands when we create the image. Here,
# we install the PM2 process manager

RUN npm install -g pm2@0.12.9
```

```
# Create an image from this Dockerfile
$ docker build -t heigvd/res-demo .

# Execute /bin/bash in a new container, created from the image
$ docker run -i -t heigvd/res-demo /bin/bash
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



The best way to understand Docker is to try it!

In this 10-minute tutorial, see how Docker works first-hand:

You'll search for and find an image another user built and shared in the Docker Hub Registry, a cloud-based collection of applications.

You'll download and run it - running images are containers - and have it output 'hello world'.

Then you'll install the 'ping' utility into the container, commit all your changes, and run a test of your updated image.

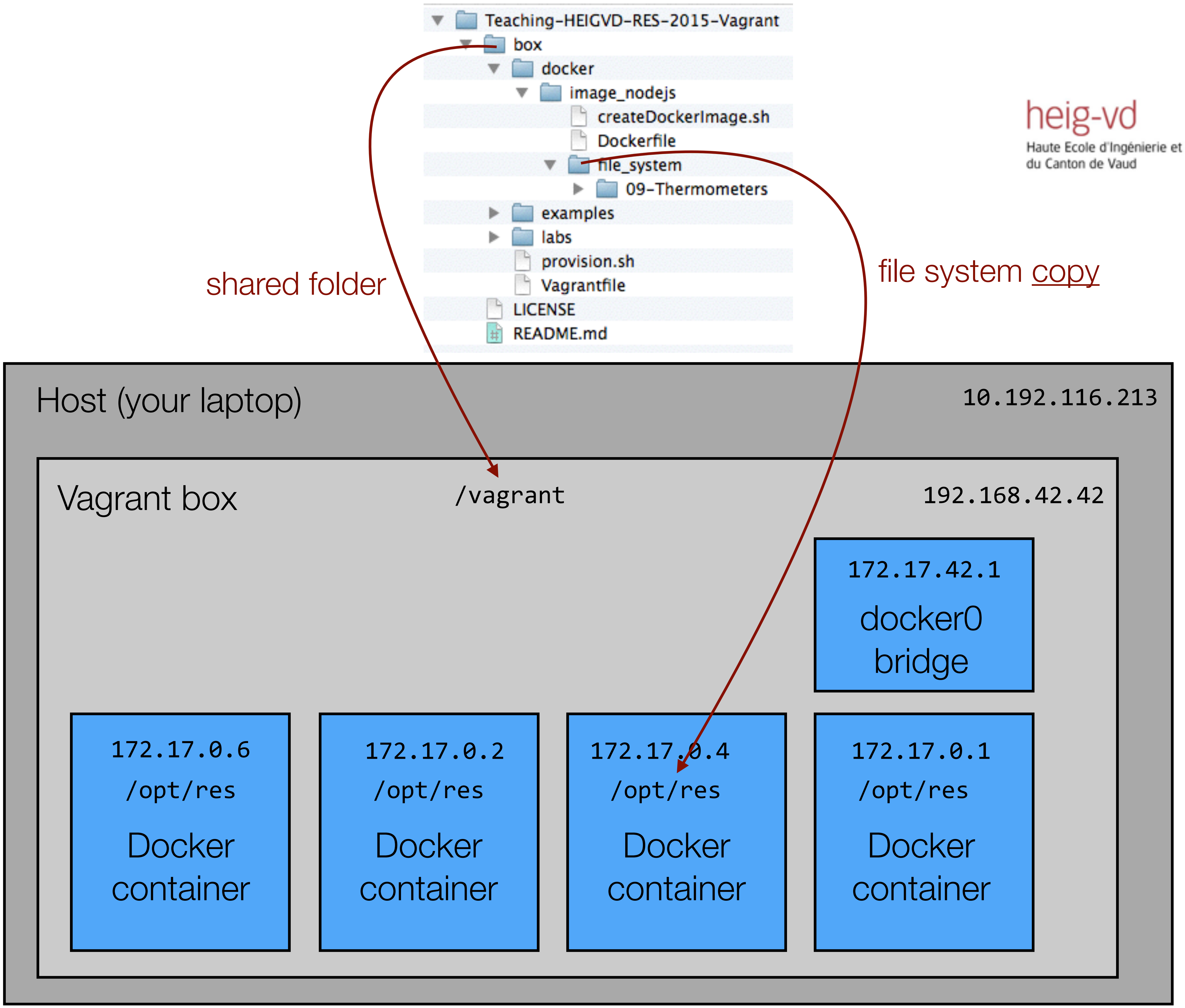
Finally, you'll push your image to the Docker Hub Registry so that other developers can find and use it...

...on a laptop, a VM in a data center, or a public cloud instance, without having to change anything at all about the image!

```
Welcome to the interactive Docker tutorial
you@tutorial:~$ 
```

This emulator provides only a limited set of shell and Docker commands.

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Things that we will not talk about today

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **How to write a Dockerfile** (we give it to you)
- **Data persistence** (we don't have to manage data for now)
- **Links between containers** (we use "standard" TCP/IP between containers)

Get Docker on your Vagrant box

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

First, let's grab the Docker material from GitHub:

```
$ cd Teaching-HEIGVD-RES-2015-Vagrant
$ git fetch upstream
$ git checkout fb-docker-setup
```

We have updated **`provision.sh`** to also install Docker. So we need to re-provision our vagrant box:

```
$ vagrant halt
$ vagrant up --provision
```

*FYI, here is the addition we made to **`provision.sh`**:*

```
# Install Docker
echo "*************************  install docker  *************************"
wget -qO- https://get.docker.com/ | sh
sudo usermod -aG docker vagrant
```

Create a Docker image

We have prepared a script for you:

```
$ cd /vagrant/docker/image_nodejs/
$ ./createDockerImage.sh
```

Content of ***createDockerImage.sh***

```
#!/bin/bash
docker build -t heigvd/res-sensors .
docker images
```

Content of ***Dockerfile***

```
FROM dockerfile/nodejs:latest
MAINTAINER Olivier Liechti
COPY file_system /opt/res/
RUN npm install -g pm2@0.12.9
```

Launch 3 containers

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

We have prepared another script for you:

```
$ cd /vagrant/docker/image_nodejs/
$ ./startContainers.sh
```

Content of ***`startContainers.sh`***

```
#!/bin/bash

docker run -d --name container_office_sensor heigvd/res-sensors node /opt/res/09-Thermometers/thermometer.js office 20 3
docker run -d --name container_bedroom_sensor heigvd/res-sensors node /opt/res/09-Thermometers/thermometer.js bedroom 17 2
docker run -d --name container_station heigvd/res-sensors node /opt/res/09-Thermometers/station.js
```

Once you have running containers, you can play around:

```
$ docker logs container_station
$ docker logs -f container_station
$ docker stop container_station
$ docker rm container_station
$ ./removeNonRunningContainers.sh
```

Sniffing UDP traffic

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Using tcpdump on the Vagrant box

```
$ sudo tcpdump -i docker0 udp -A
```

```
E..J..@...E..........q&..6.e{"timestamp":1427800010160,"temperature":null}
11:06:50.362733 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800010362,"temperature":null}
11:06:50.565816 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800010565,"temperature":null}
11:06:50.768966 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800010768,"temperature":null}
11:06:50.970691 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800010970,"temperature":null}
11:06:51.172537 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800011172,"temperature":null}
11:06:51.374546 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800011374,"temperature":null}
11:06:51.578663 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E..........q&..6.e{"timestamp":1427800011578,"temperature":null}
```