The API Socket (TCP)

TEchnologies Internet (TEI)

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Foundations

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

W1

Introduction to Java IO

Byte and character streams, dealing with files, the decorator pattern, custom reader/writers classes, buffered IOs

W2

Socket API - TCP

Client and server programming, sockets and streams, multi-threaded servers

W3

Application-level Protocol

How to specify your own communication protocol? Implement the client and the server.

W4

Socket API - UDP

Client and server programming, broadcast/multicast, service discovery protocols

This Week

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

- **Monday**
  - **08:30 - 09:00** : Review of the IO Benchmark exercise
  - **09:00 - 09:20** : Introduction to the Socket API (client and server)
  - **09:20 - 10:00** : The Calculator Exercise (Phase 1)
  - **10:30 - 12:00** : The Calculator Exercise (Phase 2)
- **Wednesday**
  - **08:30 - 08:50** : Using Threads for network programming in Java
  - **08:45 - 09:00** : The Multi-Threaded Calculator Exercise

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

Benchmarking buffered IOs

Review

What Do We Need?

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

- We need a way to **record measures** we take during the experiment:
  - Creating a **Measure class** to **represent a single measure**, with one field for every experiment parameter and one field for the measured value is certainly useful.
  - Creating a **MeasuresCollector class** to **record all measures** when we run the experiments is also useful.
  - The Measure and MeasuresCollector should also allow us to **export results in a CSV stream**.

What Do We Need?

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

- We want to **produce test data**:
  - We need the generated data files for measuring read speed. We can also **measure write speed when generating the test files**.
  - We can define a **TestDataProducer** to run the experiment several times, one time for each **combination of the parameters**.
- We want to **consume test data**:
  - We can **measure the read speed** by reading every generated test file.
  - When we generate a test data file, we can **encode the parameter values in the file name** (if the number of parameters is not too large).

What Do We Need?

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

- We can **implement the read and write loops** in dedicated classes:
  - **ReadWorker** has a **constructor**, which allows TestDataConsumer to **pass it an InputStream instance** (with a particular buffering behavior).
  - **ReadWorker** has an **execute** method, that will read all bytes on the InputStream and return the **execution time**.
  - **WriteWorker** has a **constructor**, which allows TestDataProducer to pass it an OutputStream instance (with a particular buffering behavior) and a **number of bytes to write**.
  - **WriteWorker** has an **execute** method, that will write the specified number of bytes on the OutputStream and return the **execution time**.

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

<<class>>
MeasuresCollector

```
List<Measure> collectedMeasures;
void collect(Measure m);
void exportToCSV(PrintWriter out);
```

<<class>>
Measure

```
long sizeInBytes;
boolean useBuffer;
int bufferSizeInBytes;
long processingTimeInMillis;
exportToCSV(PrintWriter out);
```

<<class>>
WriteWorker

```
WriteWorker(OutputStream out, long bytesToRead);
void long execute() {
  // write loop
}
```

<<class>>
TestDataProducer

```
public void produceTestFiles(MeasuresCollector collector);
List<String> getGeneratedFileNames();
```

<<class>>
IOBenchmarkRunner

```
void runExperiments() {
  Collector col = new MeasuresCollector();

  TestDataProducer producer = new TestDataProducer();
  TestDataConsumer consumer = new TestDataConsumer();

  producer.produceTestFiles(collector);
  consumer.consumeTestFiles(producer.getGeneratedFileNames(), collector);

  PrintWriter output = new PrintWriter("data/results.csv");
  collector.exportToCSV(output);
  output.close();
}
```

<<class>>
TestDataConsummer

```
public void consumeTestFiles(List<String> fileNames, MeasuresCollector collector);
List<String> getGeneratedFileNames();
```

<<class>>
ReadWorker

```
ReadWorker(InputStream in, long bytesToRead);
void long execute() {
  // read loop
}
```

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

How would you **describe** and **compare** **TCP** and **UDP**?

Transport Protocols

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

- Both TCP and UDP are **transport protocols**.
- This means that they make it possible for **two programs** (i.e. applications, processes) possibly running on **different machines** to **exchange data**.
- The two protocols also make it possible for several programs to **share the same network interface**. They use the notion of **port** for this purpose.
- TCP and UDP define the **structure of messages**. With TCP, messages are called **segments**. With UDP, messages are used **datagrams**.
- The structure of TCP segments (**number and size of headers**) is more complex than the structure of UDP datagrams.
- Both TCP segments and UDP datagrams can be **encapsulated in IP packets**. In that case, we say that the **payload** of the IP packet is a TCP segment, respectively a UDP datagram.

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

TCP

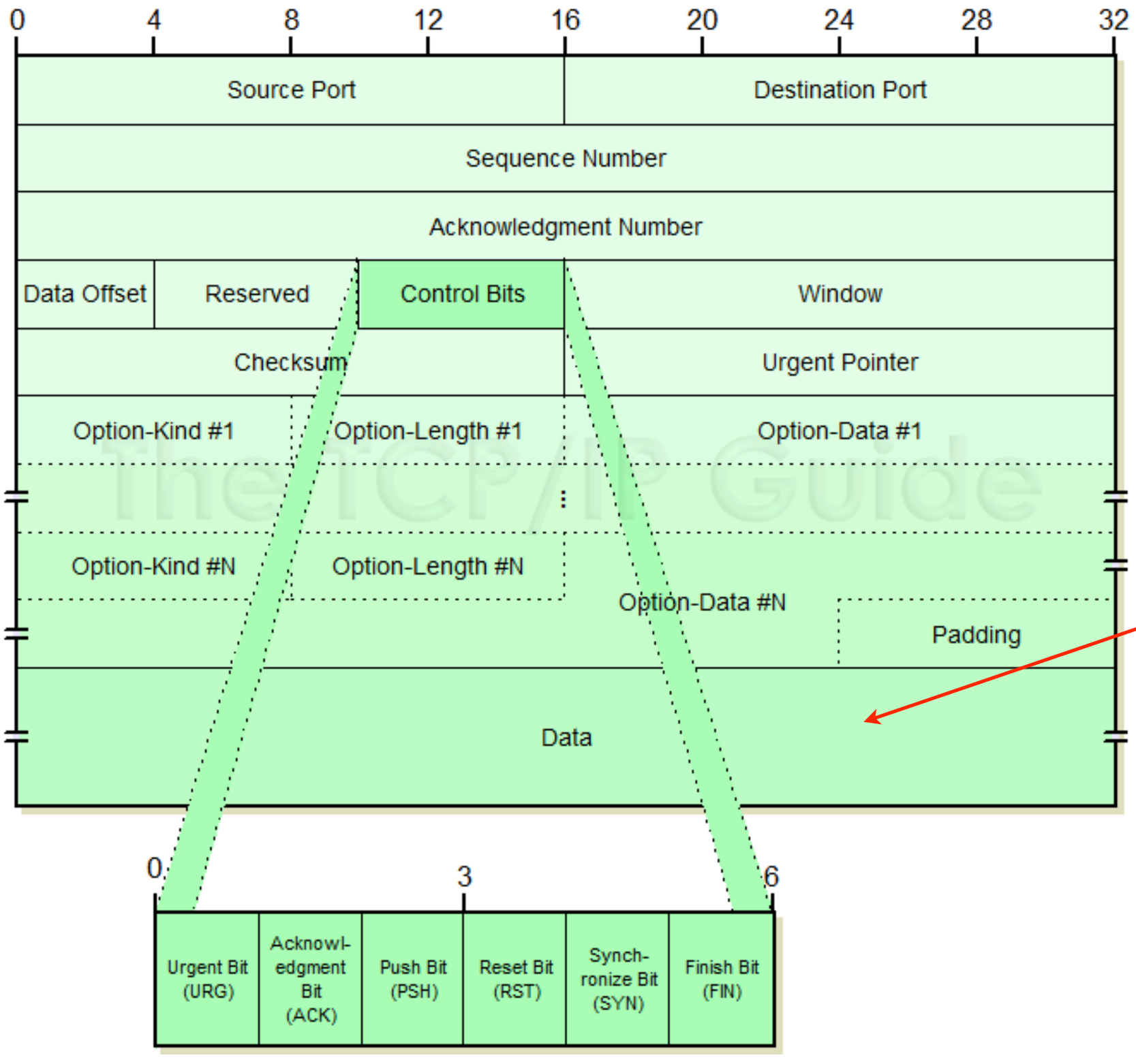
UDP

Transport Protocols

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

- TCP provides a **connection-oriented service**. The client and the server first have to establish a connection. They can then exchange data through a **bi-directional stream of bytes**.
- TCP provides a **reliable data transfer service**. It makes sure that all bytes sent by one program are received by the other. It also preserves the **ordering** of the exchanged bytes.
- UDP provides a **connectionless service**. The client can send information to the server at any time, **even if there is no server listening**. In that case, the information will simply be lost.
- UDP **does not guarantee the delivery** of datagrams. It is possible that a datagram sent by one client will never reach its destination. The ordering is not guaranteed either.
- TCP supports **unicast** communication. UDP supports **unicast**, **broadcast** and **multicast** communication. This is useful for **service discovery**.

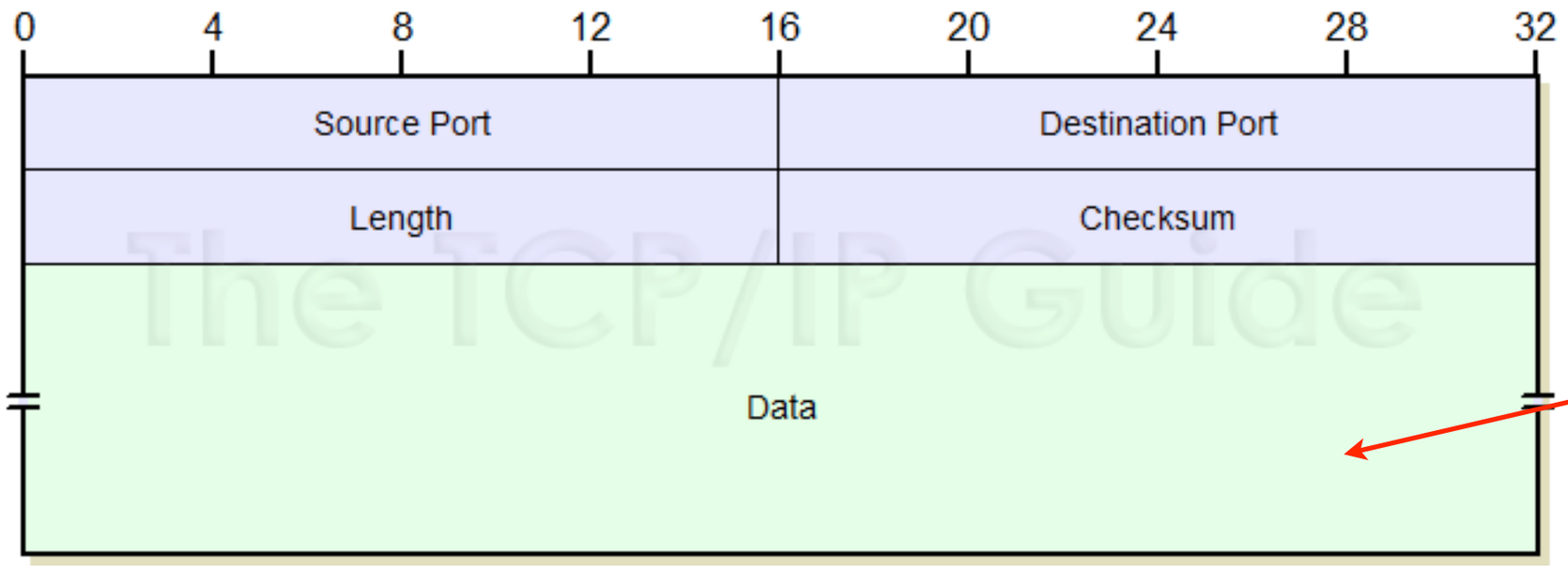
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



The bytes that you write in your java program will be here...

http://www.tcpipguide.com/free/t_TCPMessageSegmentFormat-3.htm

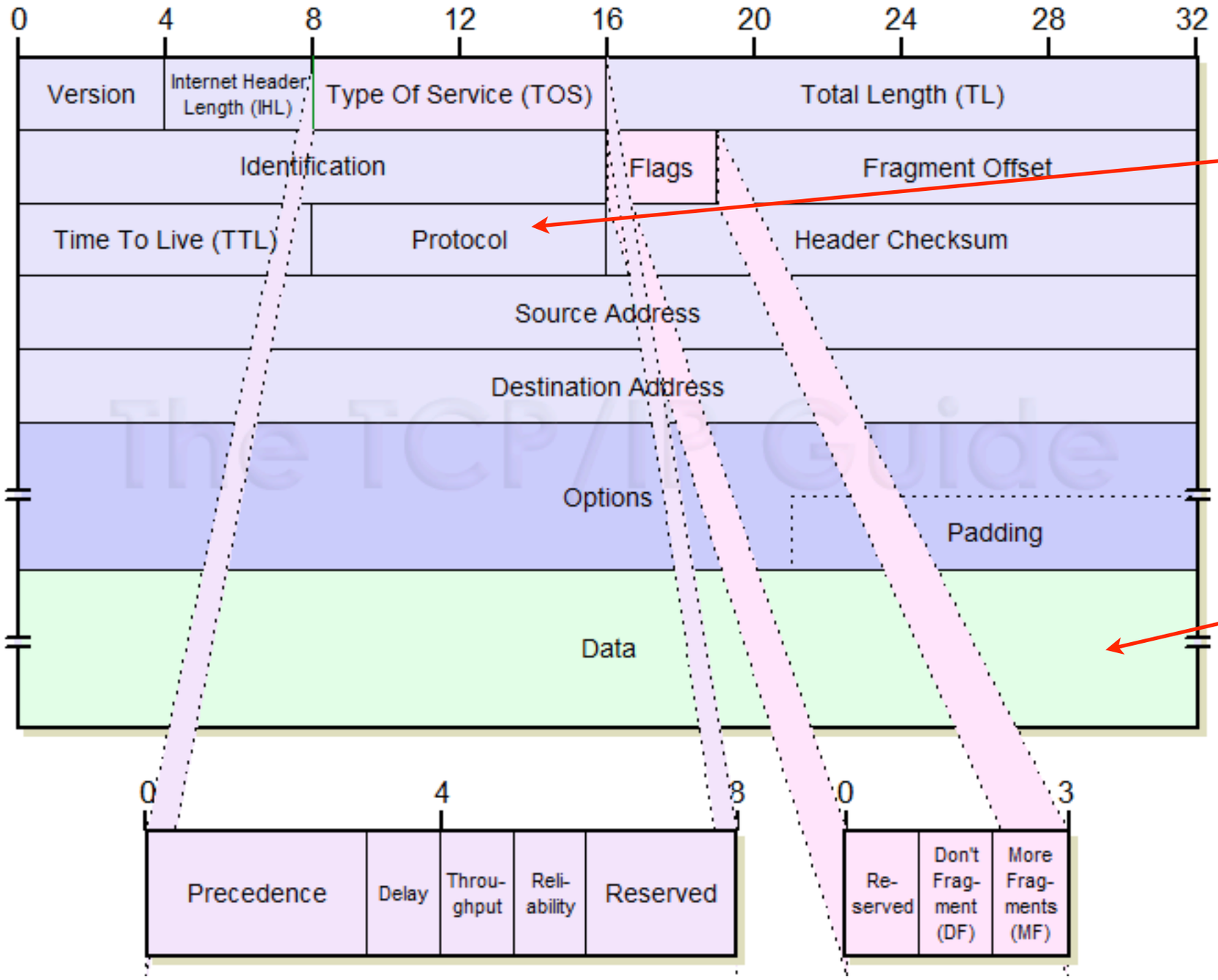
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



The bytes that you write in your java program will be here...

http://www.tcpipguide.com/free/t_UDPMessageFormat.htm

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



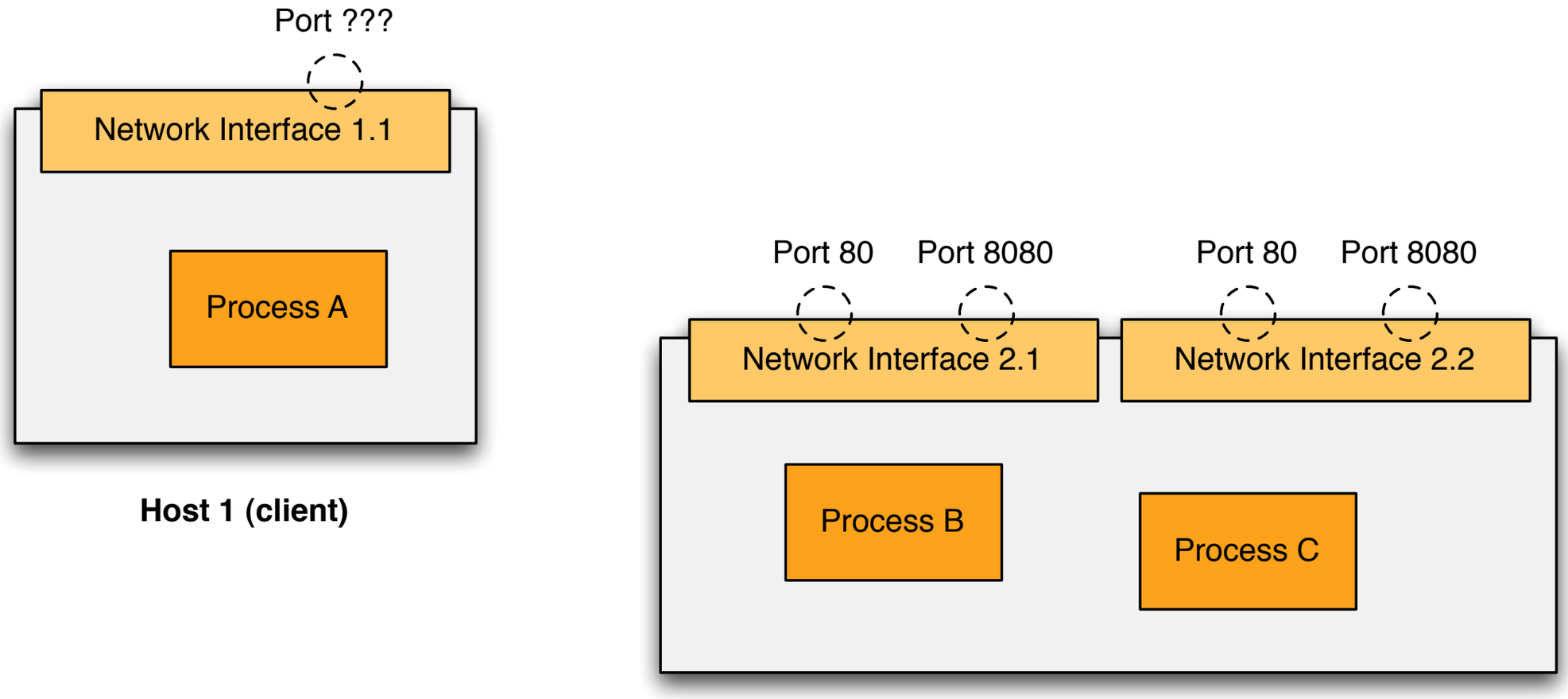
If "Data" is a TCP segment, this field has the decimal value "6". If it is a UDP datagram, this field has the decimal value "17".

This can contain a TCP segment, a UDP datagram, or something else.

http://www.tcpipguide.com/free/t_IPDatagramGeneralFormat.htm

Using the Socket API in Java

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

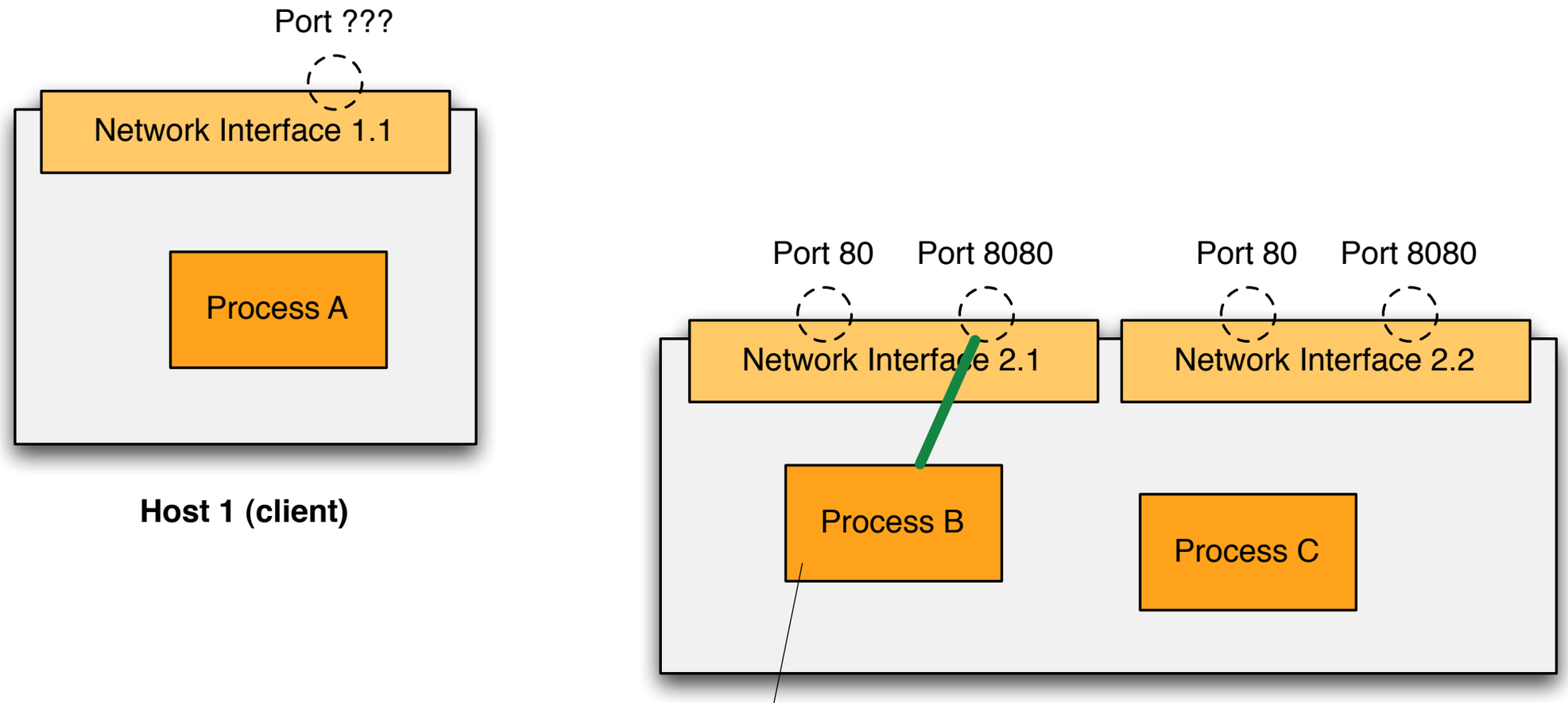


Host 1 (client)

Host 2 (server)

Using the Socket API in Java

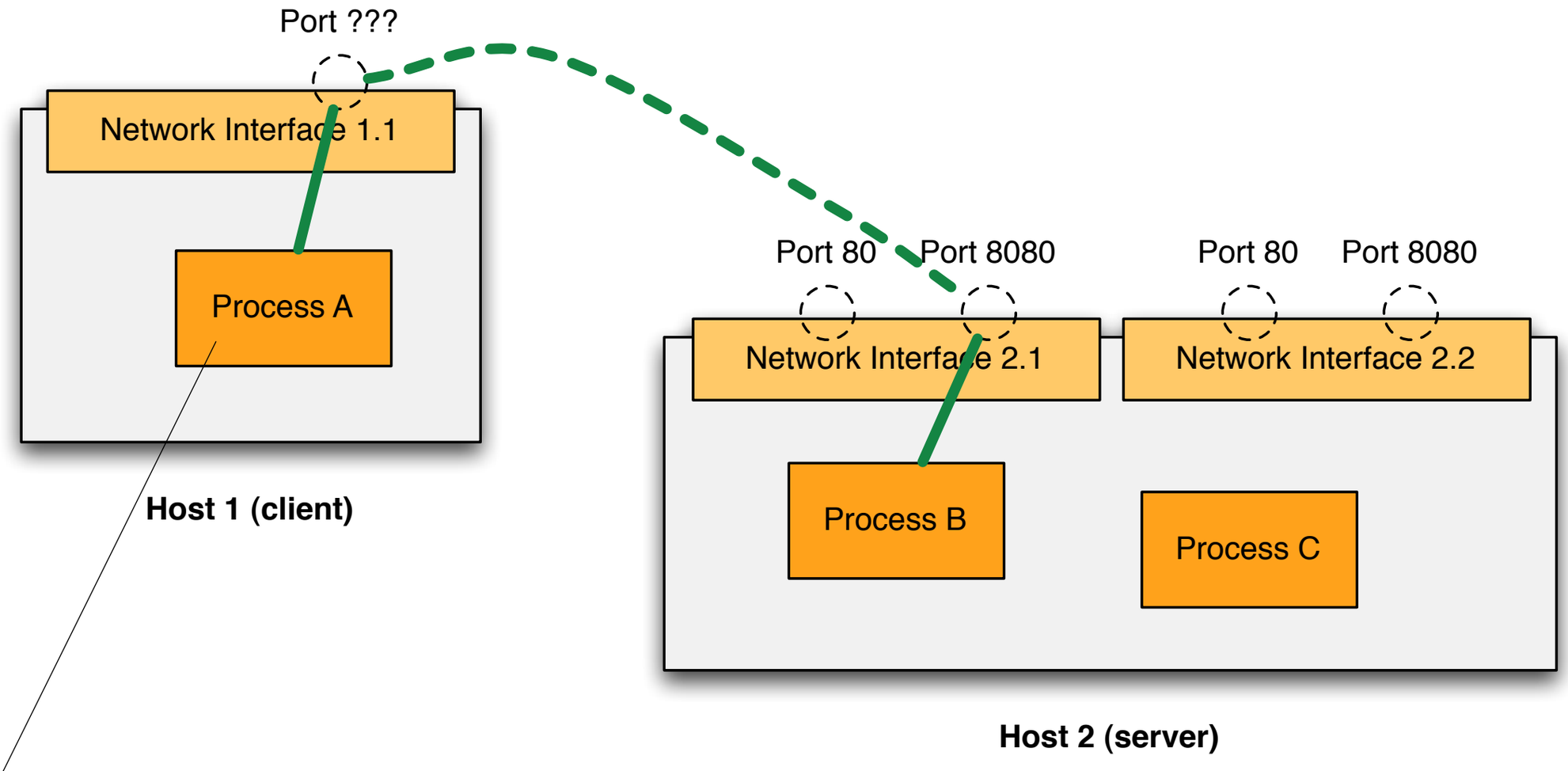
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
// Listen on port 8080
ServerSocket serverSocket = new ServerSocket(8080);

// Wait until a client makes a connection request...
Socket commSocket = serverSocket.accept();
```

Using the Socket API in Java

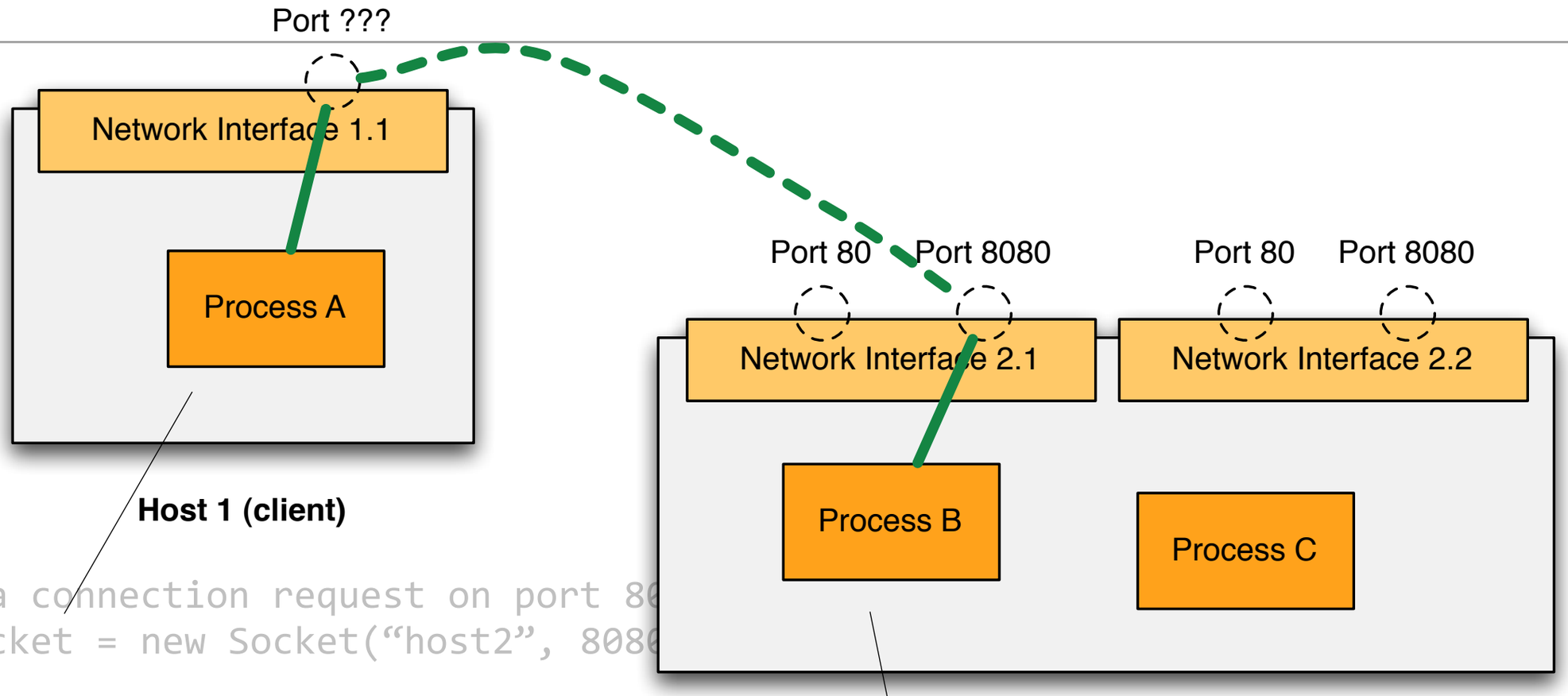
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
// Makes a connection request on port 8080
Socket serverSocket = new Socket(“host2”, 8080);
```

Using the Socket API in Java

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




```
// Makes a connection request on port 80
Socket socket = new Socket("host2", 8080);

InputStream fromServer = socket.getInputStream();
OutputStream toServer = socket.getOutputStream();
```

```
// Listen on port 8080
ServerSocket serverSocket = new ServerSocket(8080);

// Wait until a client makes a connection request...
Socket commSocket = serverSocket.accept();

InputStream fromClient = commSocket.getInputStream();
OutputStream toClient = commSocket.getOutputStream();
```

The Calculator Exercise

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

- **Phase 1 : Implement a simple TCP server and a simple TCP client**
  - The server must **accept connections** on a specified port.
  - The server implements a single **ABOUT** command that returns basic information about the program (e.g. “This program was written at HEIG-VD, etc.”).
  - To use this command, the client can make a connection request to the server. As soon as the connection is established, the client sends the following bytes: [**ABOUT\n**] through the socket.
  - **When the server receives this sequence of bytes**, it sends the command output to the client and closes the connection.
  - If the client sends **any other sequence of bytes**, the server **sends an error** message.

Read Sections of the Java Tutorial

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

- **Lesson: All About Sockets**
  http://docs.oracle.com/javase/tutorial/networking/sockets/index.html
- **What Is a Socket?**
  http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
- **Reading from and Writing to a Socket**
  http://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html
- **Writing the Server Side of a Socket**
  http://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

The Calculator Exercise

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

- **Phase 2 : Add a COMPUTE command**
  - In addition to the ABOUT command, the client can send a COMPUTE command with the following syntax: [**COMPUTE *OPERAND1* *OPERATOR* *OPERAND2*\n**], where
    - ***OPERAND1*** and ***OPERAND2*** are numeric values (double)
    - ***OPERATOR*** is either * or +
- **When the server receives a command from the client**, it first needs to determine whether it is a ABOUT or a COMPUTE command. If it is a COMPUTE command, then it needs to parse the command, compute the result (by multiplying, respectively adding the operands).
- The server has to **do proper error handling** (invalid command, syntax error, etc.).

Concurrent Programming

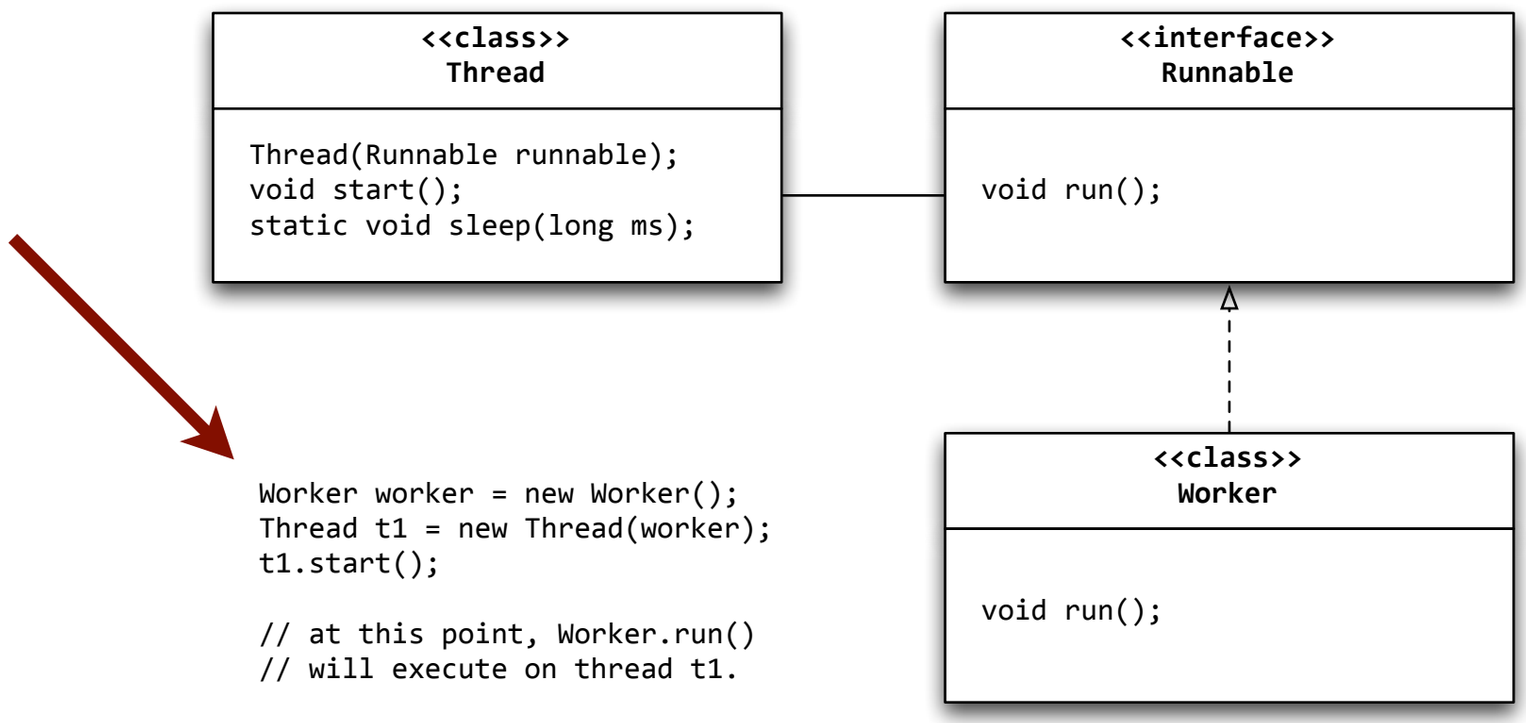
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- On top of the **operating system**, it is possible launch the Java Virtual Machine (**JVM**) several times (by invoking the java command). In this scenario, there is **one process (program) for every JVM instance**.
- If you don't do anything special, there is a **single execution thread** within each JVM. This means that all instructions in your code are executed **sequentially**.
- Very often, you write software where you want to **perform several tasks at the same time** (concurrently). For instance:
  - Manage a UI **while** fetching data from the network,
  - Talking to one HTTP client **while** talking to another HTTP client,
  - Have a worker do complex calculations on a subset of the data, **while** having another worker do the same calculations on another subset.
- You can use **threads** (also called **lightweight processes**) for this purpose.

Concurrent Programming in Java

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

- In Java, there are two main types
  - The **Thread class**, which *could be extended* to implement the behavior you want to run in parallel.
  - The **Runnable interface**, which *is implemented* for the same purpose and is passed as an argument to the Thread constructor.



Threads and Memory

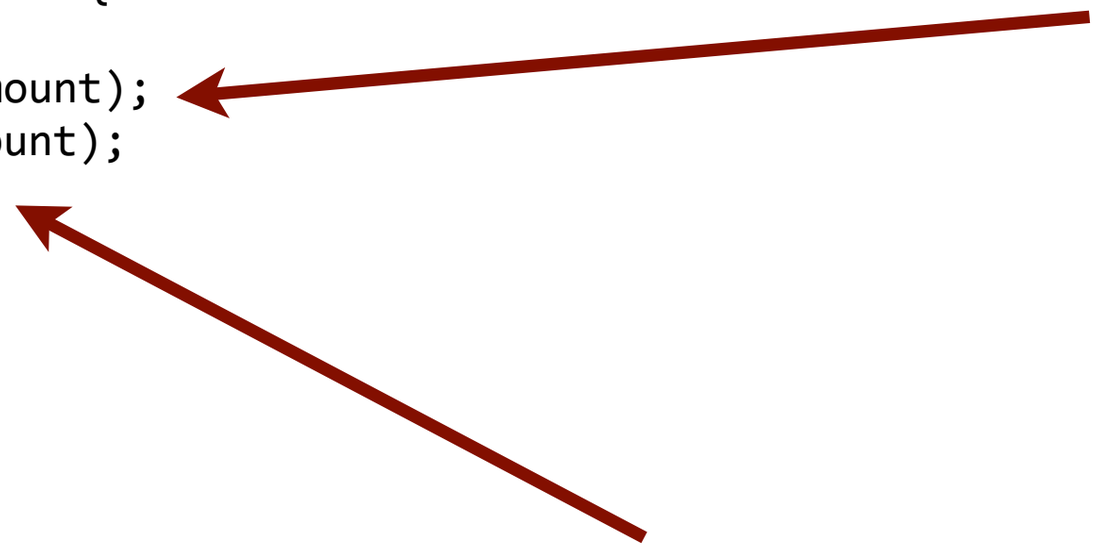
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- When you use **threads**, you have to be careful and be aware of what can happen when **several threads use the same objects**.
- If you are not careful, you will run into **concurrency issues**. You will introduce **bugs** that are not always easy to reproduce and that can be tricky to troubleshoot.

Sample Code (1/4)

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

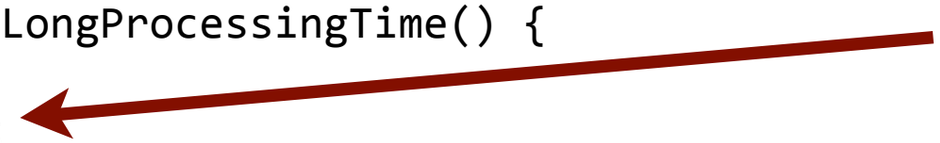
```
public interface ICustomerAccount {

    public void credit(long amount);
    public void debit(long amount);
    public long getBalance();

}
```



Sample Code (1/4)

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

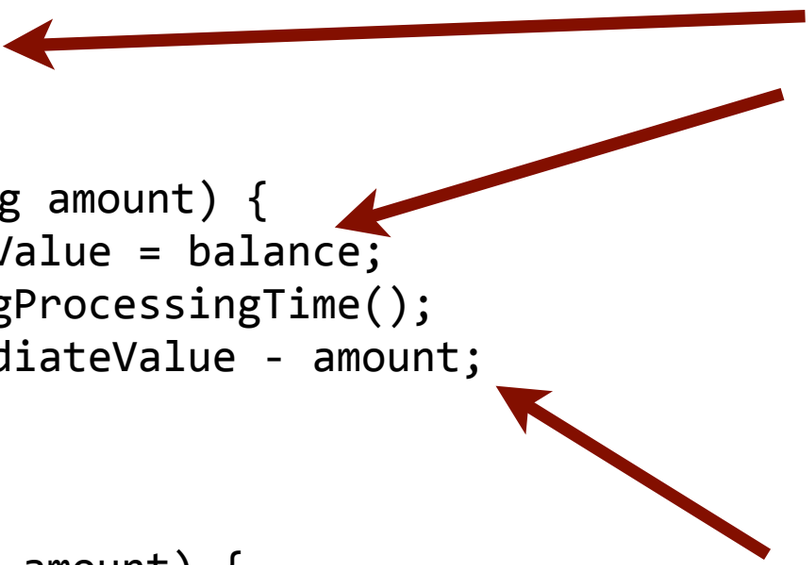
```
public class Utils {

    public static void simulateLongProcessingTime() {
        try {
            Thread.sleep(200);
        } catch (InterruptedException ex) {
            Logger.getLogger(Utils.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```



Sample Code (1/4)

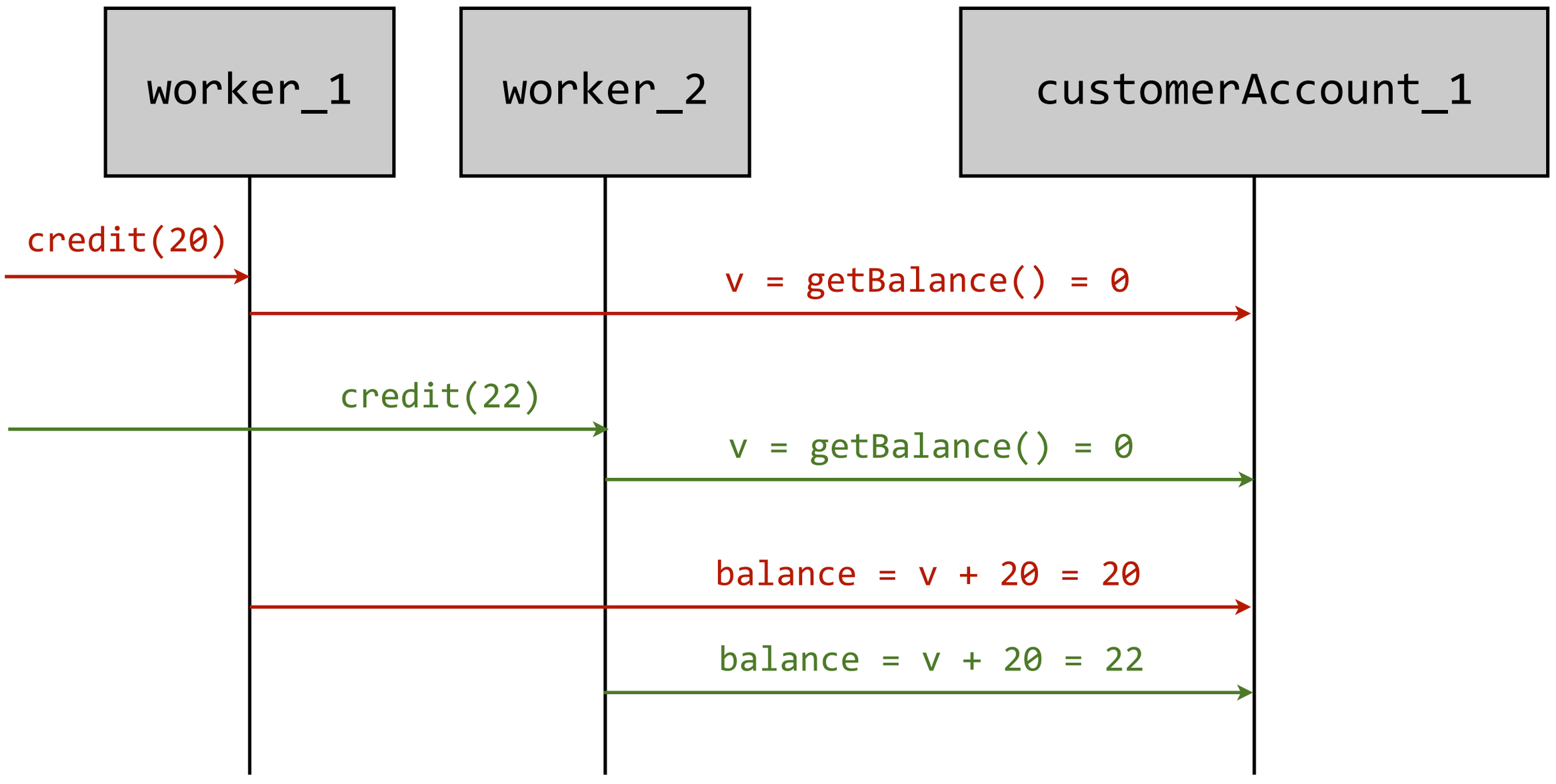
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
public class NonThreadSafeCustomerAccount implements ICustomerAccount {

     private long balance;

     @Override
     public void credit(long amount) {
          long intermediateValue = balance;
          Utils.simulateLongProcessingTime();
          balance = intermediateValue - amount;
     }

     @Override
     public void debit(long amount) {
          credit(-amount);
     }

     @Override
     public long getBalance() {
          return balance;
     }

}
```



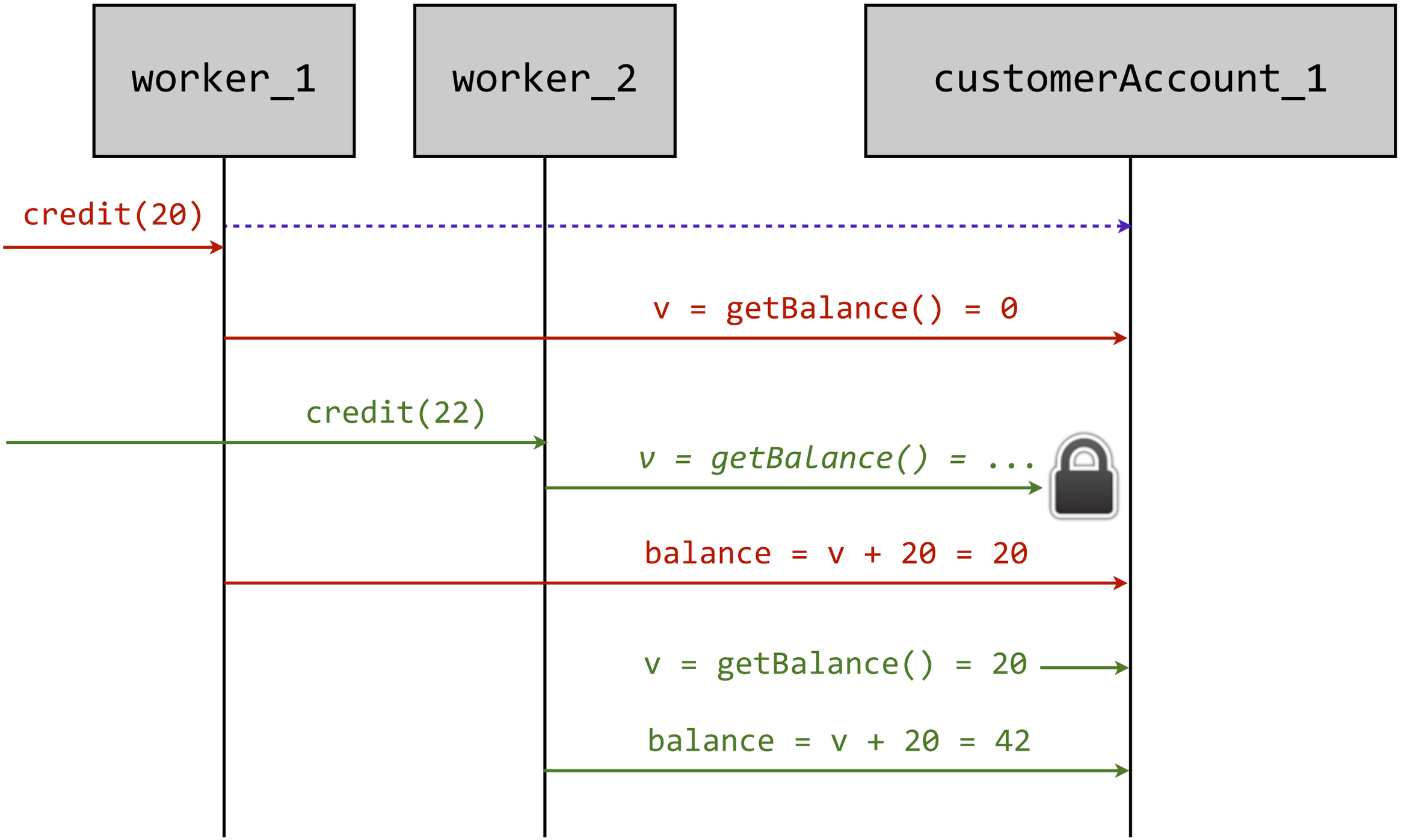
Concurrency Issue (Data Corruption)

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



Preventing the Issue With a Lock

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



Sample Code (1/4)

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

```
public class ThreadSafeCustomerAccount implements ICustomerAccount {

     private long balance;

     @Override
     public synchronized void credit(long amount) {
          long intermediateValue = balance;
          Utils.simulateLongProcessingTime();
          balance = intermediateValue + amount;
     }

     @Override
     public synchronized void debit(long amount) {
          credit(-amount);
     }

     @Override
     public synchronized long getBalance() {
          return balance;
     }
}
```

Threads & Network Programming

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

- Threads are useful both when implementing **servers** and **clients**:
  - A **TCP server** may want to **listen for connection requests**, **while** talking to **several clients** at the same time.
  - A **client application** may want to **fetch data in the background**, executing dealing with a TCP socket in a different thread than the main UI thread (doing otherwise would freeze the UI).

Sample Code (1/4)

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

```
public class MultiThreadedServer {

    private static final int DEFAULT_LISTEN_PORT = 1446;

    public void start() {
        Thread acceptThread = new Thread(new AcceptWorker(DEFAULT_LISTEN_PORT));
        acceptThread.start();
    }

    public static void main(String[] args) {
        MultiThreadedServer server = new MultiThreadedServer();
        server.start();
    }
}
```

Sample Code (2/4)

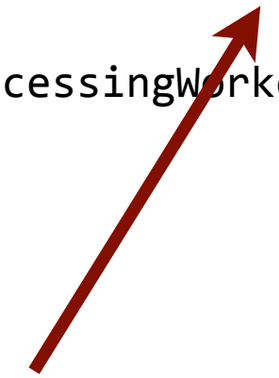
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
public class AcceptWorker implements Runnable {
     private int listenPort;

     AcceptWorker(int listenPort) {
          this.listenPort = listenPort;
     }

     public void run() {
          System.out.println("AcceptWorker starts his task...");
          try {
               System.out.println("-> creating new server socket on port " + listenPort);
               ServerSocket serverSocket = new ServerSocket(listenPort);
               while (true) {
                    System.out.println("-> waiting for client...");
                    Socket clientSocket = serverSocket.accept();
                    System.out.println("-> client has arrived, spawning new thread to process his
                                        request.");
                    Thread t = new Thread(new RequestProcessingWorker(clientSocket));
                    t.start();
               }
          } catch (IOException ex) {
               Logger.getLogger(AcceptWorker.class.getName()).log(Level.SEVERE, null, ex);
          }
     }
}
```

Sample Code (3/4)

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

```
public class RequestProcessingWorker implements Runnable {

    private Socket clientSocket;
    private BufferedReader in;
    private PrintWriter out;

    RequestProcessingWorker(Socket clientSocket) {
        this.clientSocket = clientSocket;
        try {
            this.in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            this.out = new PrintWriter(clientSocket.getOutputStream());
        } catch (IOException ex) {
            Logger.getLogger(RequestProcessingWorker.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }

...
```



Sample Code (4/4)

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

```
@Override
public void run() {
     System.out.println("RequestProcessingWorker starts his task...");
     try {
          String input;
          while ((input = in.readLine()) != null) {
               System.out.println(">> Client sent : " + input);
               out.println("OK");
               out.flush();
          }
     } catch (IOException ex) {
          Logger.getLogger(RequestProcessingWorker.class.getName()).log(Level.SEVERE, null,
ex);
     } finally {
          System.out.println("Client has closed connection, closing streams and socket...");
          try {
               in.close();
               out.close();
               clientSocket.close();
          } catch (IOException ex) {
               Logger.getLogger(RequestProcessingWorker.class.getName()).log(Level.SEVERE,
null, ex);
          }
     }
}
```