LIÈGE université

Faculté des Sciences Appliquées

INFO0010 - Introduction to Computer Networking

THE MQTT BROKER

Guidelines & Complement

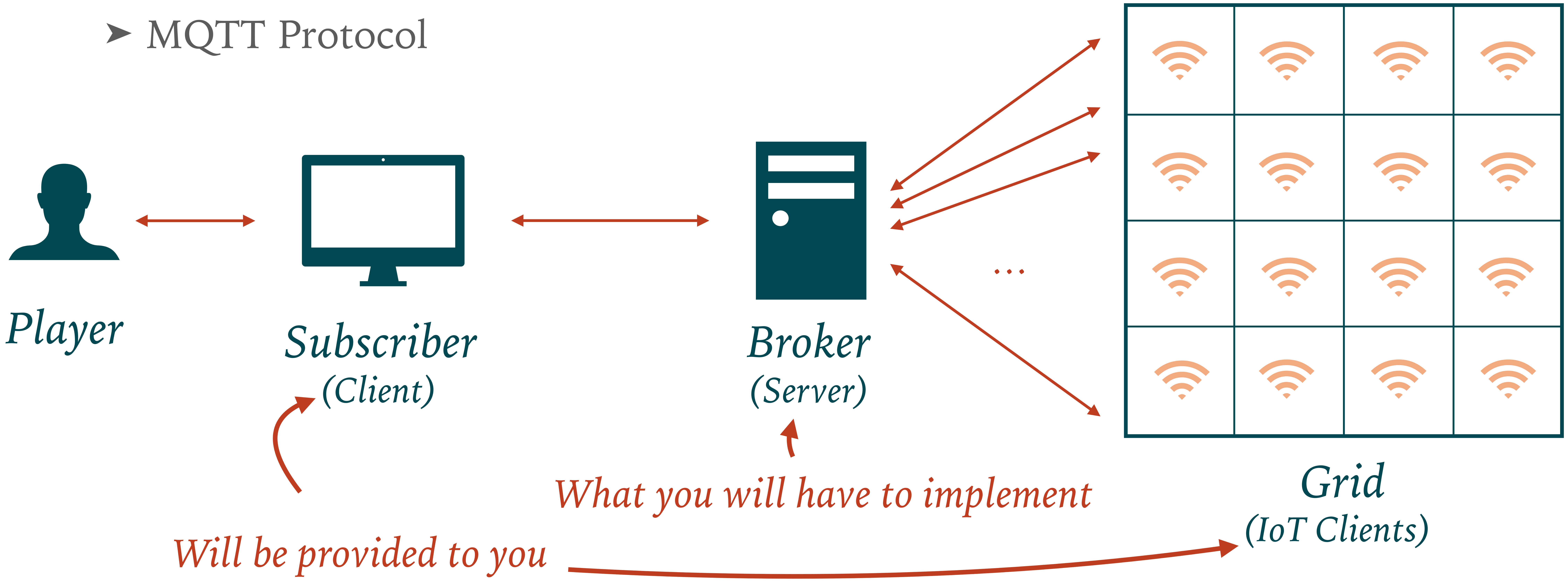
Emeline Marechal *Guy Leduc*

Year 2021-2022

THE ASSIGNMENT

PROJECT IN A NUTSHELL (I)

- *Same application as for the first part the assignment: The Monster Hunting Game.*
  - Broker
  - MQTT Protocol



MQTT: THE STANDARD FOR IOT MESSAGING (I)

- *http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html*

Lightweight and efficient

Publish/Subscribe pattern

Reliable message delivery

Scale to millions of things

Support for unreliable networks

Agnostic of application data

MQTT

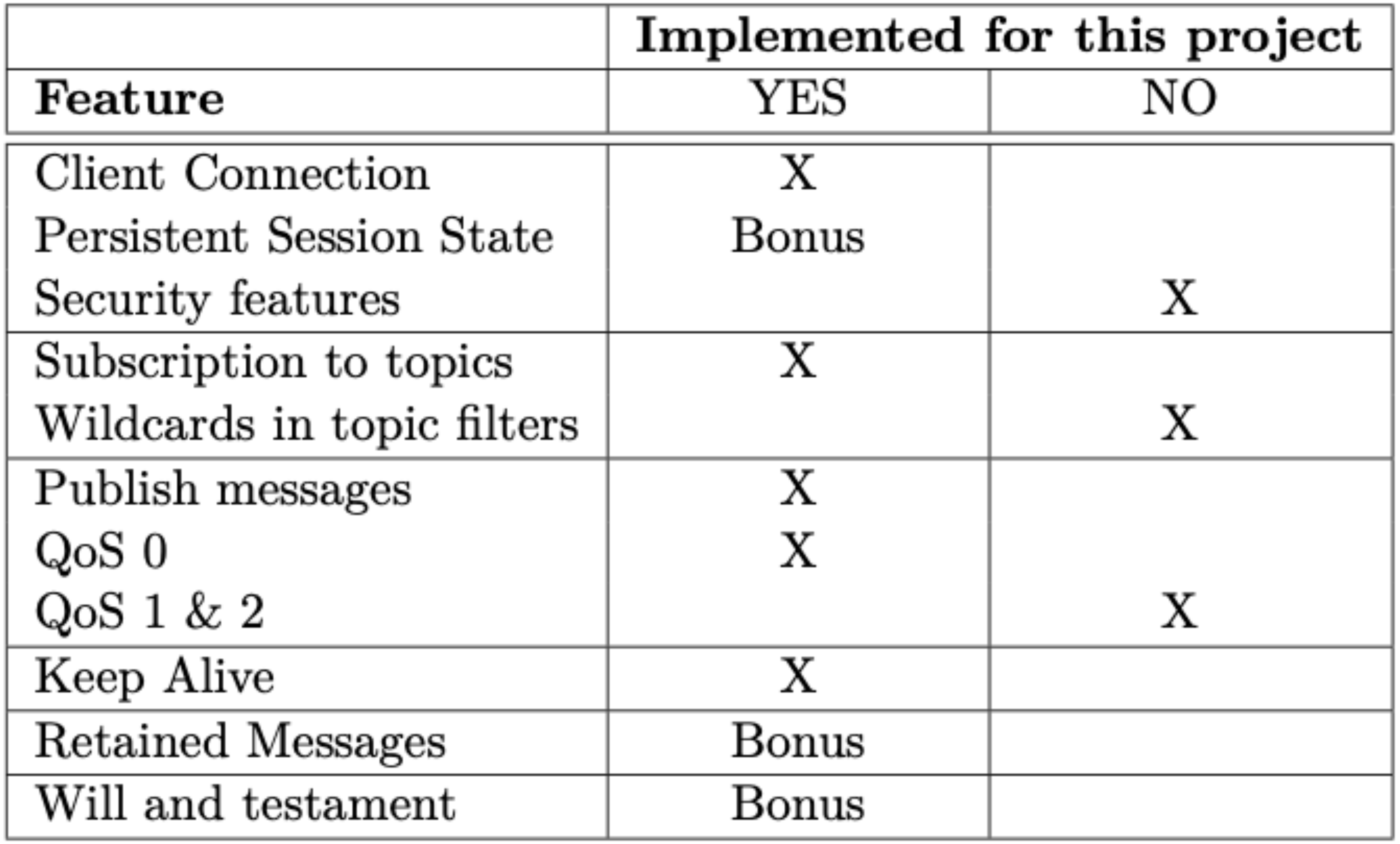
MQTT: THE STANDARD FOR IOT MESSAGING (II)

- *The MQTT Broker is responsible for:*
  - Accepting all incoming connections from publishers/subscribers,
  - Receiving all messages,
  - Filtering those messages according to their topic,
  - Sending those messages to the interested subscribers.

Don't underestimate the time it will take you to master the MQTT protocol.

MQTT: THE STANDARD FOR IOT MESSAGING (III)

➤ *MQTT Features:*



	Implemented for this project	
Feature	YES	NO
Client Connection	X	
Persistent Session State	Bonus	
Security features		X
Subscription to topics	X	
Wildcards in topic filters		X
Publish messages	X	
QoS 0	X	
QoS 1 & 2		X
Keep Alive	X	
Retained Messages	Bonus	
Will and testament	Bonus	

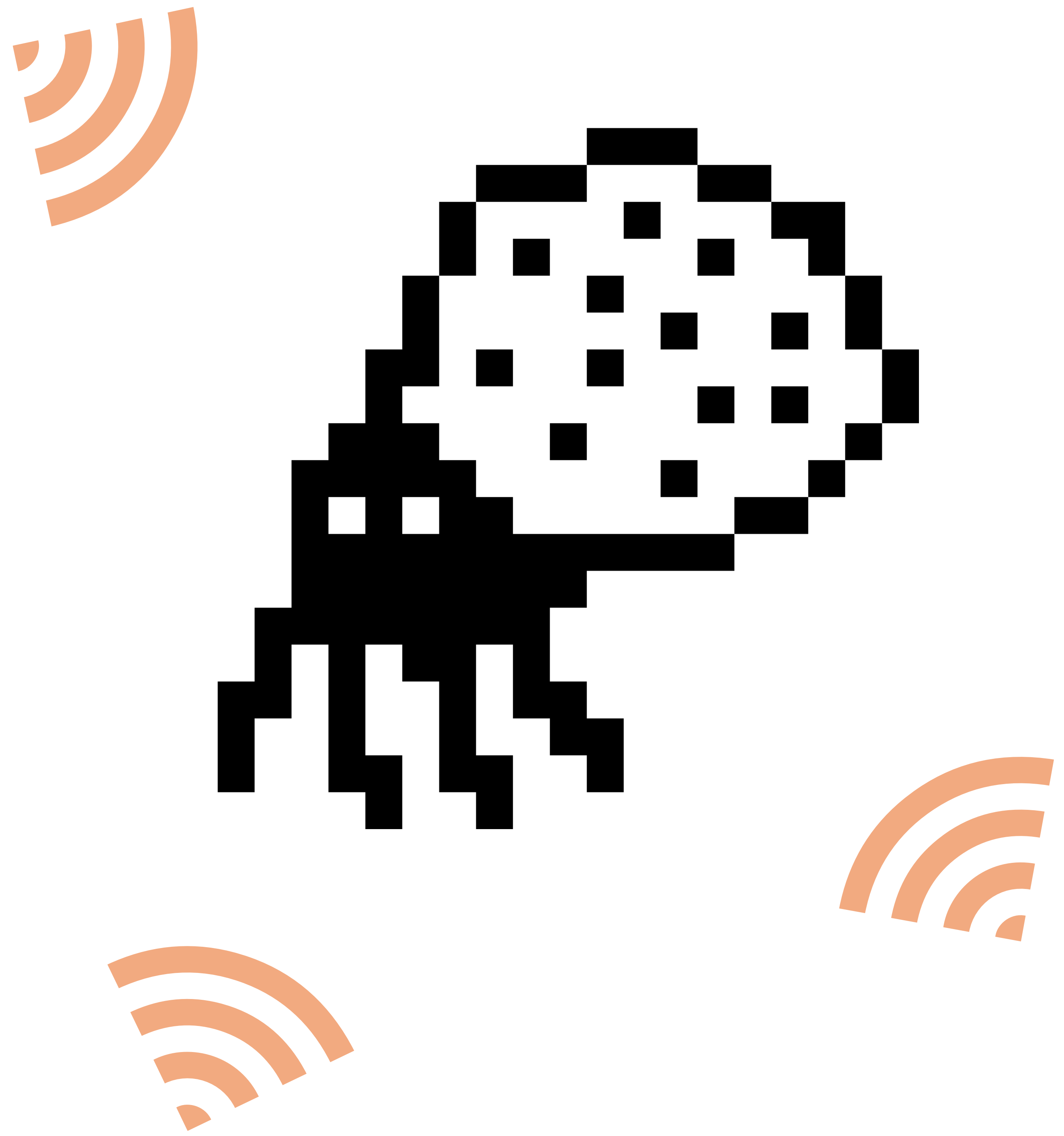
MQTT: THE STANDARD FOR IOT MESSAGING (IV)

- *(Un)intentional malevolence:*
  - Check the validity of the messages you receive,
  - Check the MQTT protocol is not violated,
  - Make sure no one opens a TCP connection and keep it open forever without being active,
  - In case of abnormal behavior/malformed packets, the Broker must close the TCP connection.

Never expect! Always check!

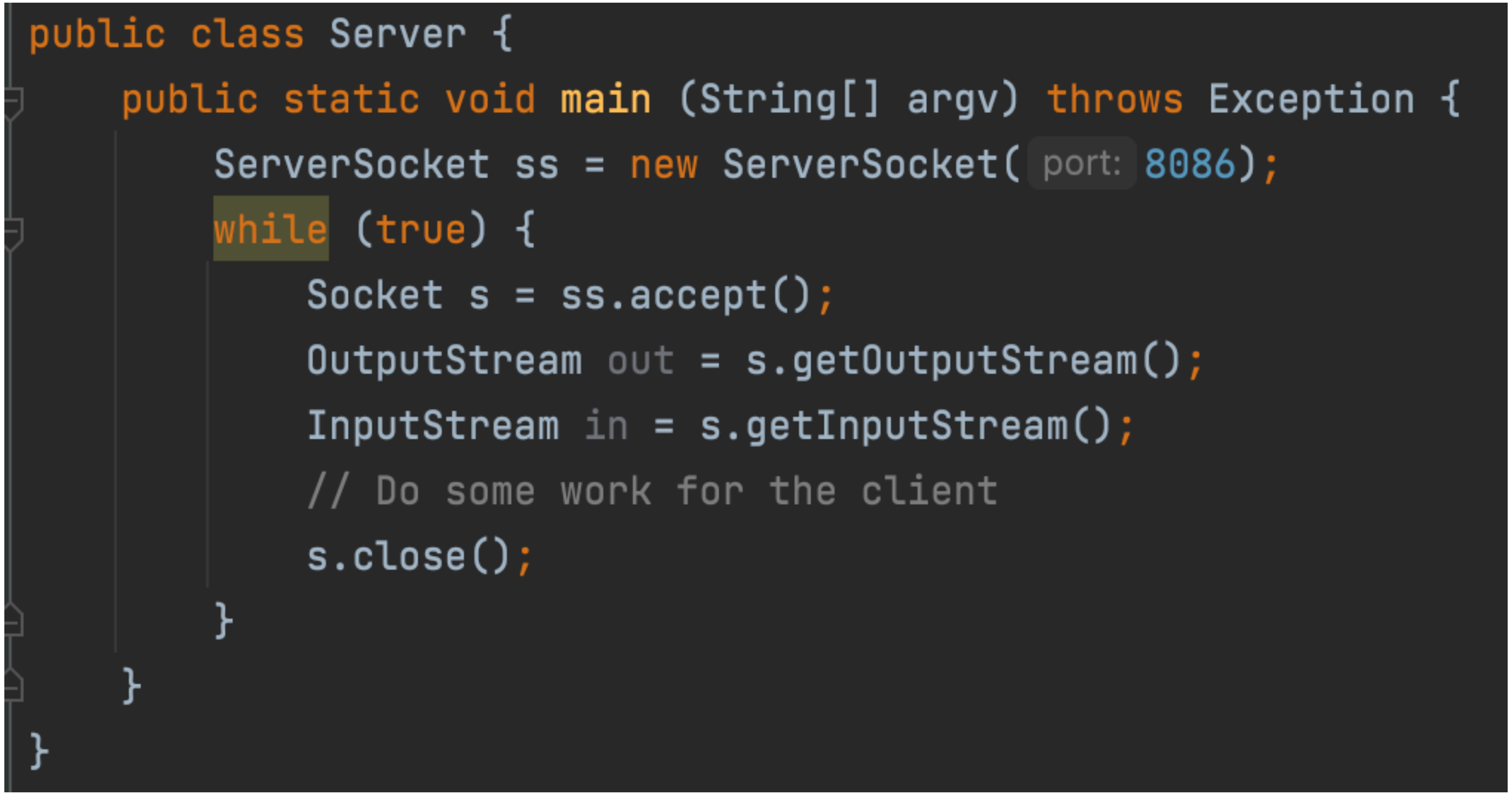
PROJECT IN A NUTSHELL (II)

- *Guidelines:*
  - Java (1.8) Sockets,
  - Imposed binary protocol to follow: MQTT, the standard for IoT messaging,
  - To be realized in teams of 2 students,
  - Must be **fully operational** on the `ms8xx.montefiore.ulg.ac.be` machines. See http://www.student.montefiore.ulg.ac.be/accounts.php to create an account if not already done,
  - **Hard deadline: 17th of December 2021.**



SERVER ARCHITECTURE

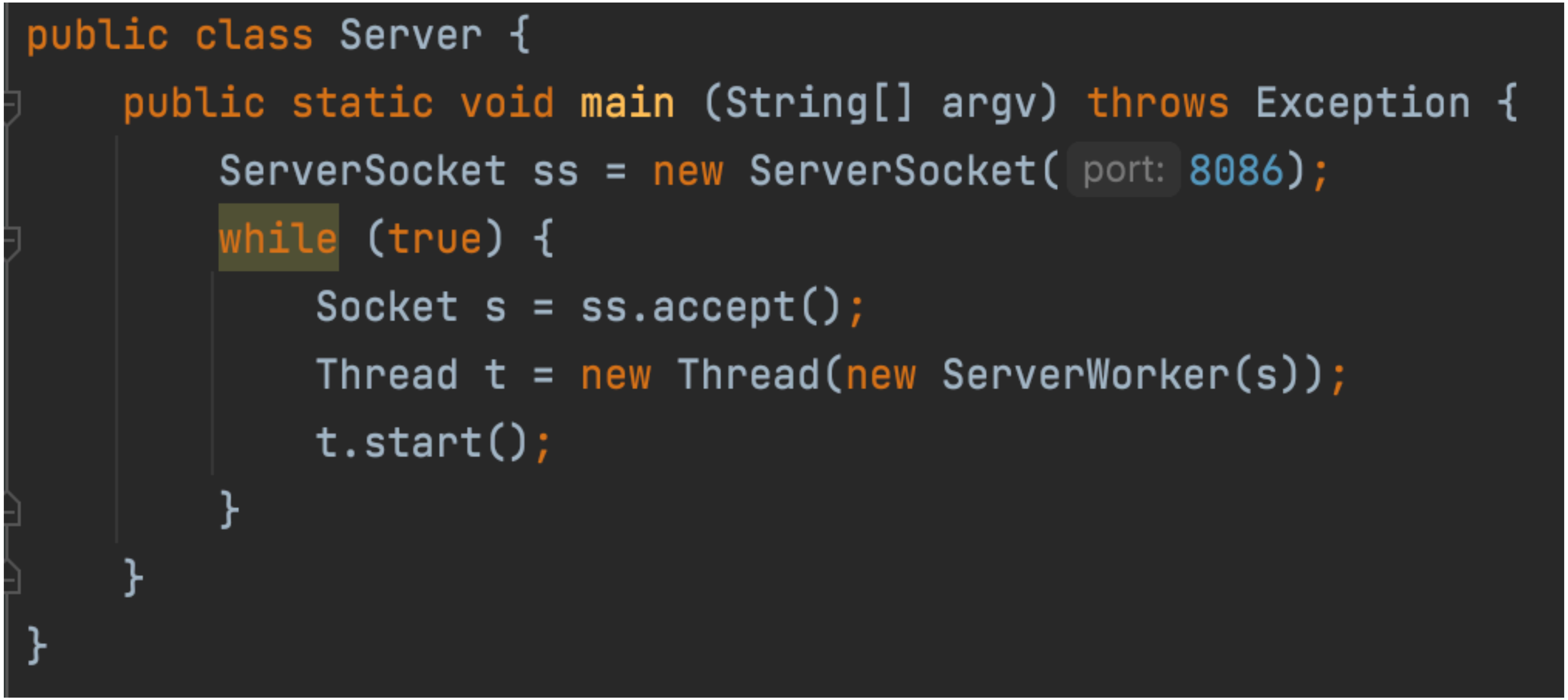
NAIVE SERVER ARCHITECTURE



```
public class Server {
    public static void main (String[] argv) throws Exception {
        ServerSocket ss = new ServerSocket( port: 8086);
        while (true) {
            Socket s = ss.accept();
            OutputStream out = s.getOutputStream();
            InputStream in = s.getInputStream();
            // Do some work for the client
            s.close();
        }
    }
}
```

The server can only deal with one client at a time!

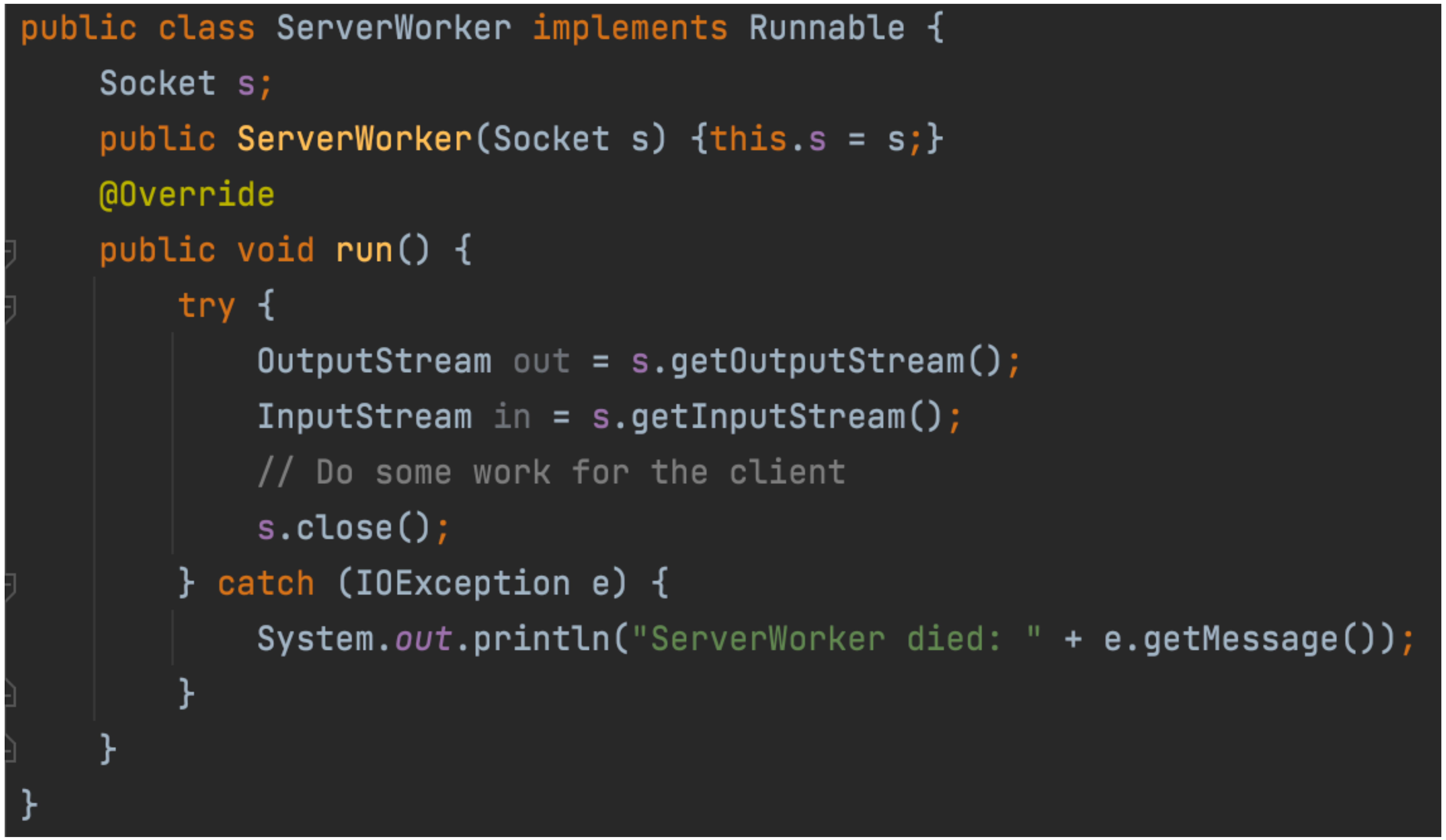
MULTI-THREADED SERVER ARCHITECTURE



```
public class Server {
    public static void main (String[] argv) throws Exception {
        ServerSocket ss = new ServerSocket( port: 8086);
        while (true) {
            Socket s = ss.accept();
            Thread t = new Thread(new ServerWorker(s));
            t.start();
        }
    }
}
```

- *We spawn a new thread every time a connection arrives*
- *The main thread can go back to welcoming incoming clients*

DEFINING A WORKER

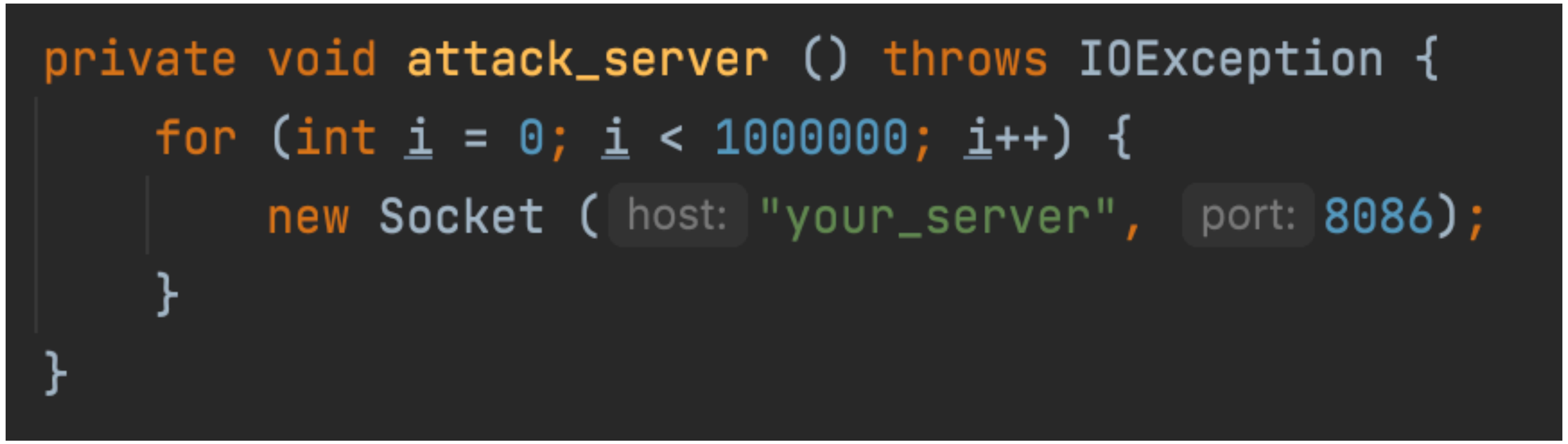


```java
public class ServerWorker implements Runnable {
    Socket s;
    public ServerWorker(Socket s) {this.s = s;}
    @Override
    public void run() {
        try {
            OutputStream out = s.getOutputStream();
            InputStream in = s.getInputStream();
            // Do some work for the client
            s.close();
        } catch (IOException e) {
            System.out.println("ServerWorker died: " + e.getMessage());
        }
    }
}
```

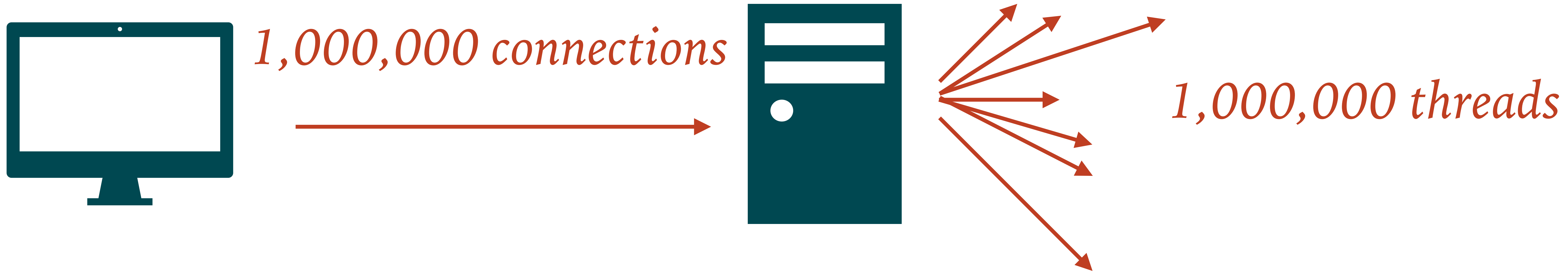
- *Implements the Runnable Interface*
- *Overrides the run method*

ATTACK ON THE SERVER

➤ *What happens if I run this piece of code?*

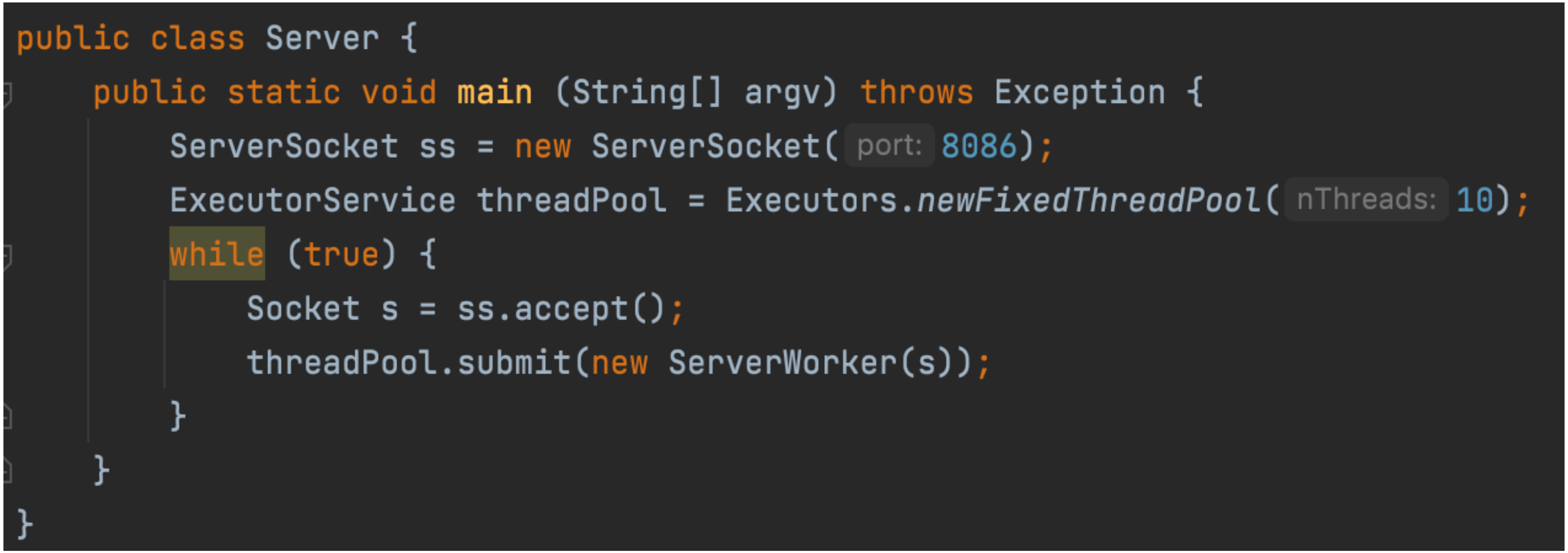


```
private void attack_server () throws IOException {
    for (int i = 0; i < 1000000; i++) {
        new Socket ( host: "your_server",  port: 8086);
    }
}
```



The server will die from exhaustion of resources!

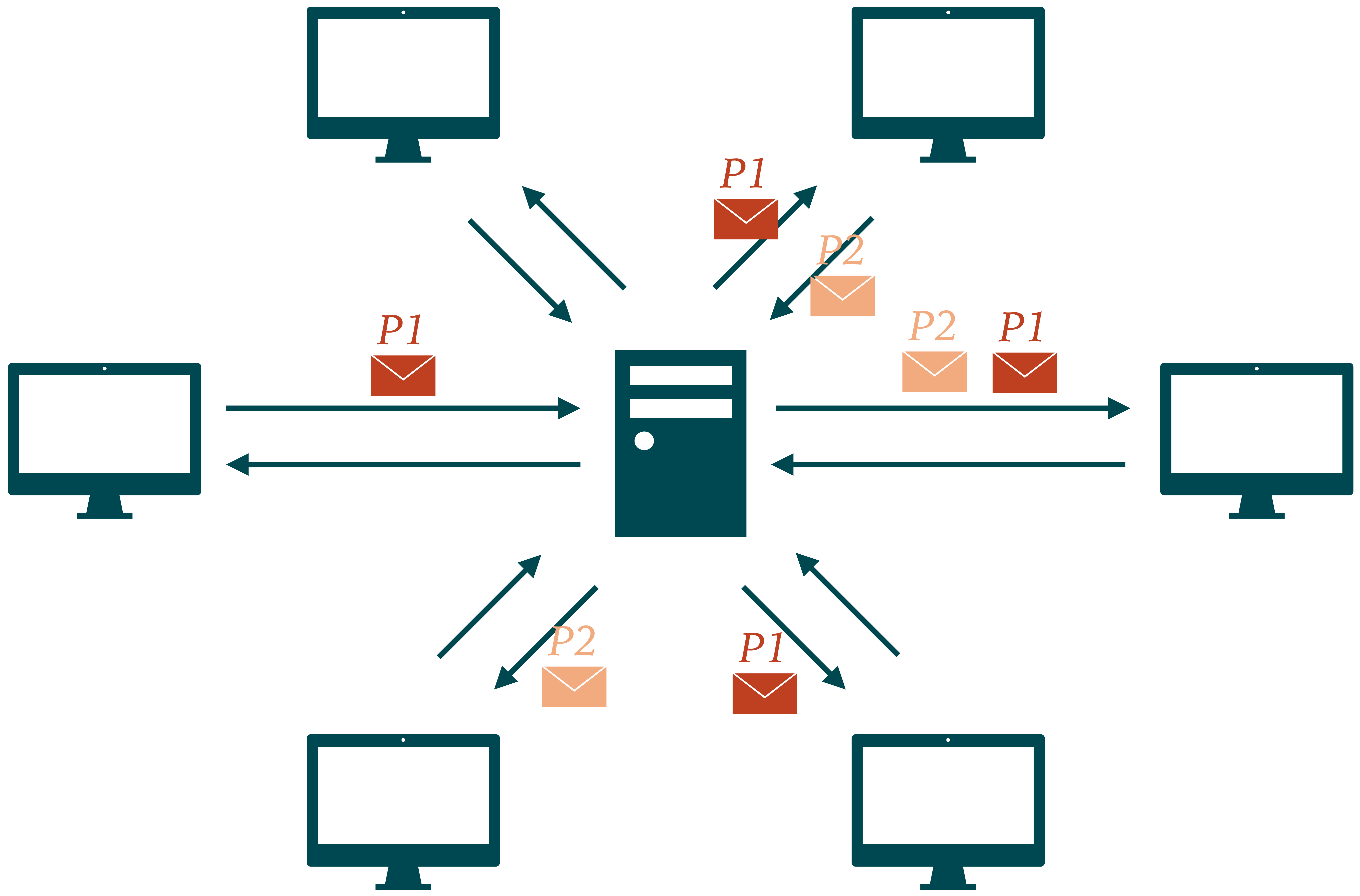
SOLUTION: A THREAD POOL



```
public class Server {
    public static void main (String[] argv) throws Exception {
        ServerSocket ss = new ServerSocket( port: 8086);
        ExecutorService threadPool = Executors.newFixedThreadPool( nThreads: 10);
        while (true) {
            Socket s = ss.accept();
            threadPool.submit(new ServerWorker(s));
        }
    }
}
```

- *Server can handle up to 10 (in this case) concurrent connections.*
- *Once the work is done, the thread is back in the pool and ready for a new task.*

MQTT BROKER ARCHITECTURE: ASYNCHRONOUS



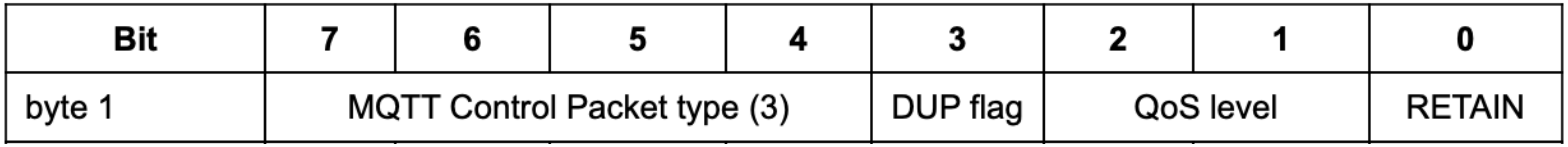
- MQTT is asynchronous by design
- Two threads per client (a *reading* and a *writing* thread)
- Mechanism for coordination between the threads (have a look at `java.util.concurrent.BlockingQueue`)

SHARED OBJECTS

- *What if some objects need to be manipulated by different threads?*
  - The execution is concurrent and non-atomic, the object consistency is not ensured,
  - Need to maintain the object consistency,
  - Solution: key word `synchronized`,
  - Mutual exclusion: the code inside a `synchronized` block cannot be run at the same time by different threads.

BYTE PARSING

- *Parsing of a byte to extract all the fields:*



Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN

- *Behavior specific to Java:*
  - For bits arithmetic, a byte cannot be manipulated alone. It will first undergo *promotion* (transformation of a byte into an int) by the JVM:
    - 00101011 ⟶ 00000000000000000000000000101011
    - 11010100 ⟶ 11111111111111111111111111010100

SOME COMMAND LINES

- To compile:

```
javac *.java
```

- To launch Java Program:

```
java MyMain
```

- To launch a jar archive:

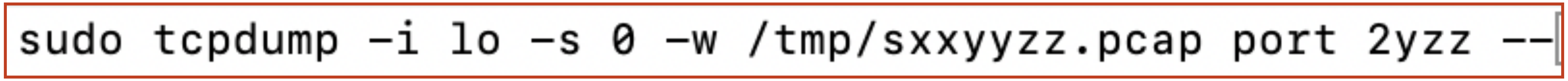
```
java -jar MyJar.jar
```

- To track system calls issued by your program:

```
strace -e trace=network -f java MyMain
```

NOTES ON TESTING ON THE MS8XX

- *To launch tcpdump on the ms8xx:*



```
sudo tcpdump -i lo -s 0 -w /tmp/sxxyyzz.pcap port 2yzz --
```

  - Use your ULiège ID
  - Output files must be situated in directory `/tmp`
  - The ms8xx machines are shared :
  - Don’t step on each other’s feet: respect the guidelines for port and pcap numbering!

LIVE DEMO