Practical Concurrent and Parallel Programming XI

Java Networking & Introduction to Erlang

Raúl Pardo and Jørgen Staunstrup

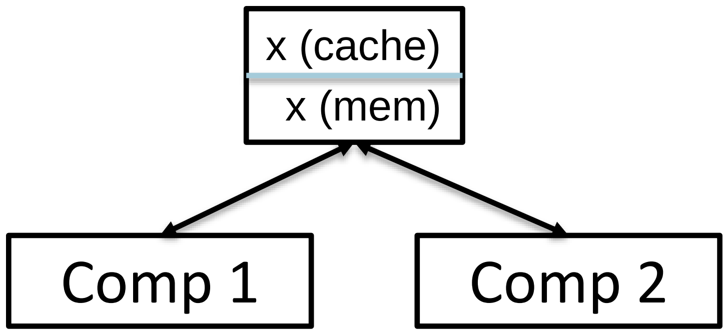
Agenda

- Networking (general)
- Java sockets
- Internet protocols and JSON
- Erlang
- . . .

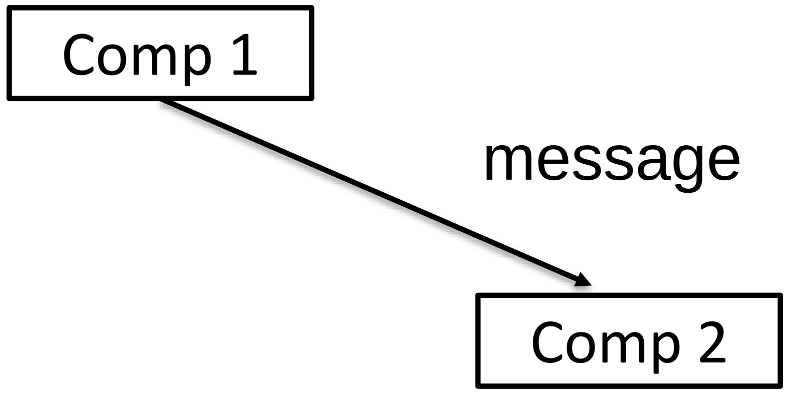
Message passing vs. shared memory

Two mental models for coordinating concurrent computations

Shared memory



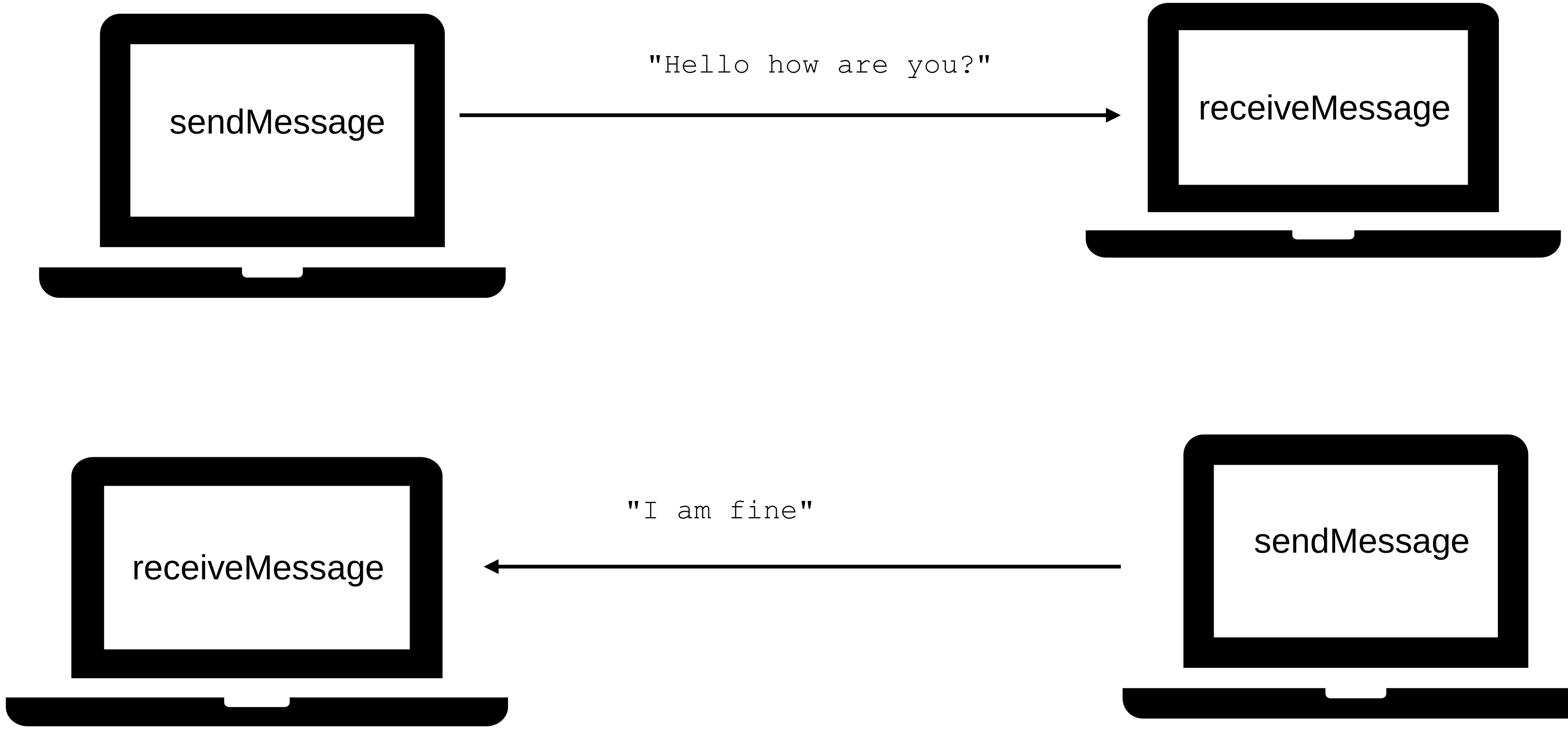
Message passing



Theoretically equally powerful

each can simulate the other

Message passing on the Internet: Sockets (TCP)



Socket addressing

Addressing (IP addresses) like: 192.168.1.204

Each computer has many independent **ports/sockets (e.g. 8080)**

Socket address
192.168.1.204:8080

Addressing local sockets

Referencing sockets on local PC

```
private final static String IP=
  "127.0.0.1";          // this PC
  //"localhost";        // this PC

private Socket clientSocket;
clientSocket= new Socket(IP, 8080);
```

Use command
ipconfig
to find IP-addr
of your PC

For this week's exercises both server and client are on the same PC
(in two different windows)

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

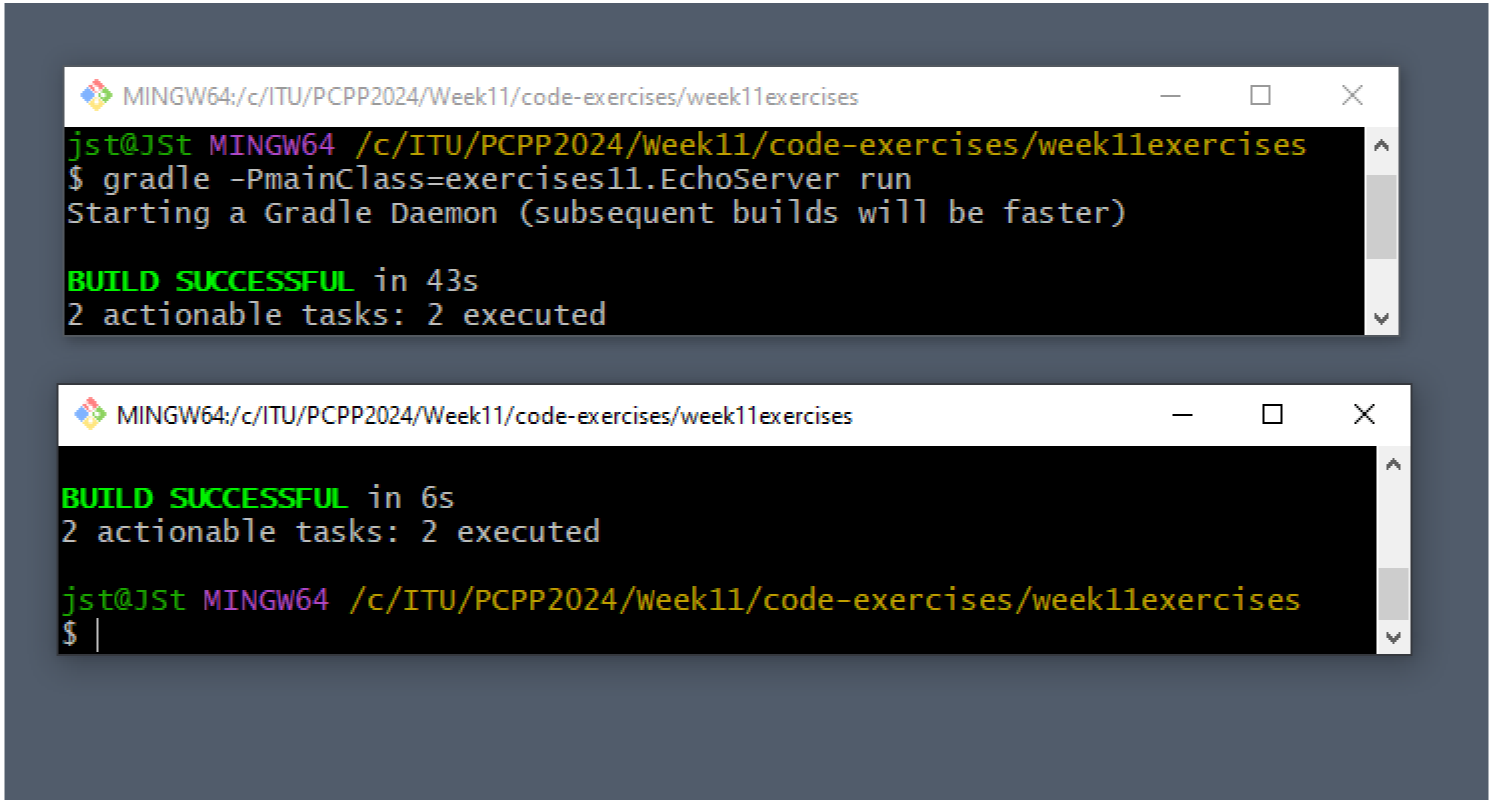
Java Sockets (send)

```
public class Server {
  private ServerSocket serverSocket; // to receive messages
  private BufferedReader in;
  private Socket clientSocket;        // to return responses
  private PrintWriter out;

  public String readMessage(BufferedReader in) {
    try {  return in.readLine();
    } catch (IOException e) { System.out.println(e.getMessage());}
    return null;
  }
...
  serverSocket= new ServerSocket(port);
  clientSocket= serverSocket.accept();
  out= new PrintWriter(clientSocket.getOutputStream(), true);
  in= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

  String inputLine;
  while ((inputLine= readMessage(in)) != null) {

  ... }
```

Java Sockets (receive)

```
public class client {
  private Socket clientSocket;
  private PrintWriter out;
  private BufferedReader in;

 public void startConnection(String ip, int port) {
   try {
     clientSocket= new Socket(ip, port);
     out= new PrintWriter(clientSocket.getOutputStream(), true);
     in= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
   } catch (IOException e) {  System.out.println(e.getMessage());    }
 }

 public String sendMessage(String msg) {
   try {
     out.println(msg);
     return in.readLine();
   } catch (Exception e) {  return null;    }
 }
...

startConnection("127.0.0.1", 8080);
sendMessage("get")
```

Running client and server



MINGW64:/c/ITU/PCPP2024/Week11/code-exercises/week11exercises

```
jst@JSt MINGW64 /c/ITU/PCPP2024/Week11/code-exercises/week11exercises
$ gradle -PmainClass=exercises11.EchoServer run
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 43s
2 actionable tasks: 2 executed
```

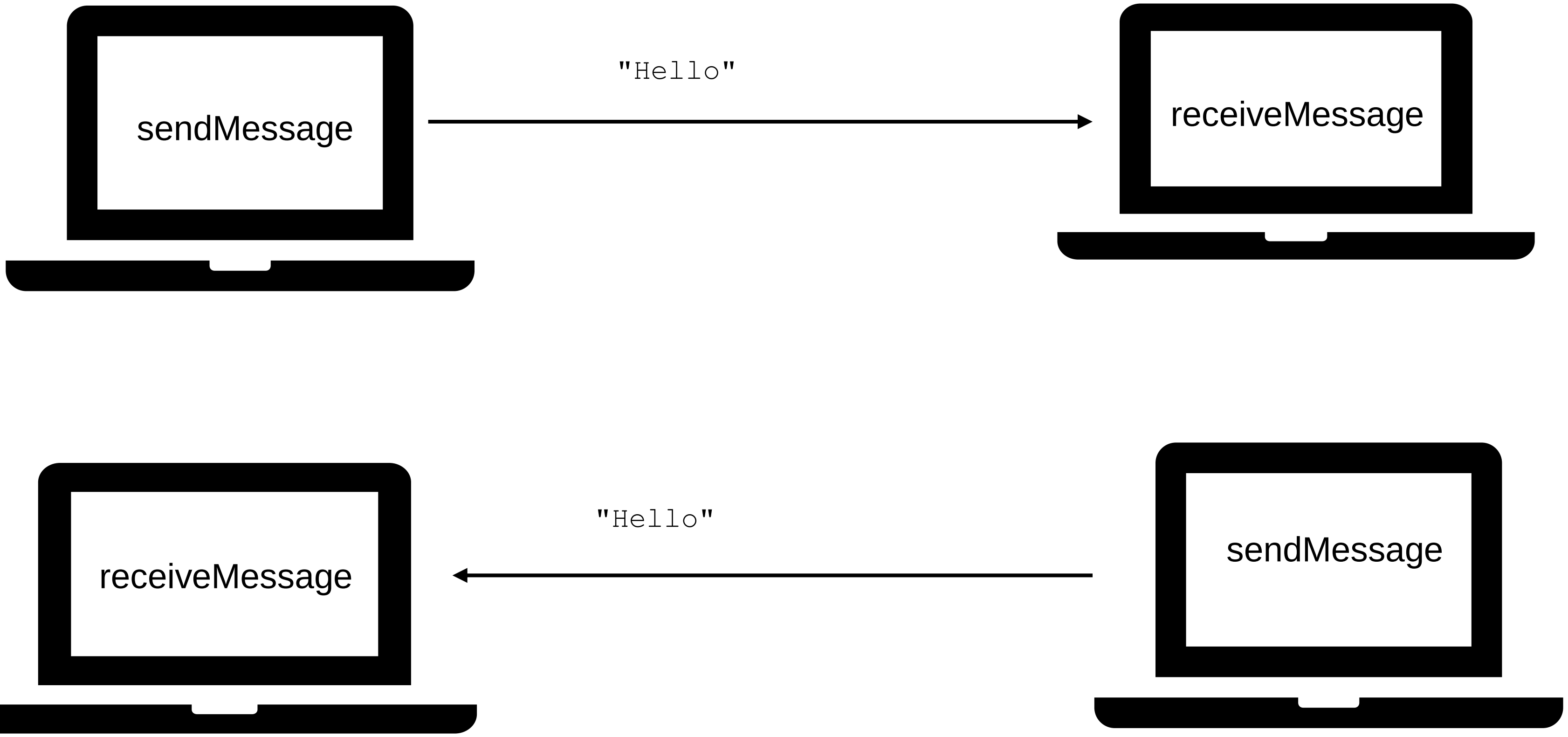
MINGW64:/c/ITU/PCPP2024/Week11/code-exercises/week11exercises

```

BUILD SUCCESSFUL in 6s
2 actionable tasks: 2 executed

jst@JSt MINGW64 /c/ITU/PCPP2024/Week11/code-exercises/week11exercises
$ |
```

You need **two** terminal windows to run both server and client

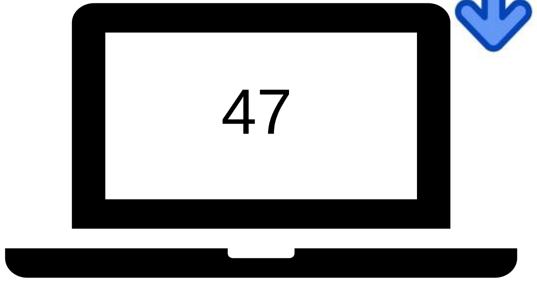
Example: EchoServer and EchoClient

complete code in: code-exercises/ …/EchoServer.java and /EchoClient.java



Java example: Number Server

```
public class NumberServer {
  private int count= 0;


  /*messages

    get       returns the current value of the server's number
    incr      increments the server's number by 1
    put dd    changes the server's number to dd
    stop      stops the server

  */

  public static void main(String[] args) {

  }
}
```



NumberServer (functionality)

```
String inputLine;
while ((inputLine= readMessage(in)) != null) {
  if ("incr".equals(inputLine)) {
     count= count+1;
     out.println(count);
  } else if ("get".equals(inputLine)) {
     out.println(count);
  } else if ("put".equals(inputLine.substring(0, 3))) {
     count= Integer.parseInt(inputLine.substring(4, inputLine.length()));
     out.println(count);
  } else if ("stop".equals(inputLine)) {
     out.println("good bye "+ count);
     stop();
     break;
  }
}
```

NumberServer (communication)

```
public class NumberServer {
  private ServerSocket serverSocket;
  private Socket clientSocket;
  private PrintWriter out;
  private BufferedReader in;
  private int count= 0;

  public String readMessage(BufferedReader in) {
    try {
      return in.readLine();
    } catch (IOException e) { System.out.println(e.getMessage());}
    return null;
  }

  public void start(int port) {
    try {
      serverSocket= new ServerSocket(port);
      clientSocket= serverSocket.accept();
      out= new PrintWriter(clientSocket.getOutputStream(), true);
      in= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
      ...  // functionality --- see previous slide

    } catch (IOException e) { System.out.println(e.getMessage());}

  }
}
```

Complete code is in:
code-exercises/ …
NumberServer.java

Ports

```
public static void main(String[] args) {
    new NumberServer().start(8080);
}
```

The server will read messages one at a time from a specific port.

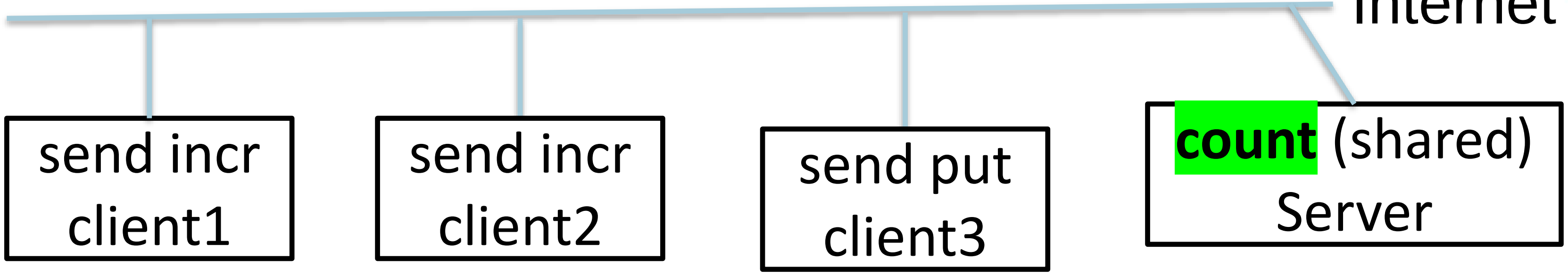
Different ports can be used to differentiate different message types

https://www.techtarget.com/searchnetworking/definition/port-number

Number client

```
public class NumberClient {
  private Socket clientSocket;
  private PrintWriter out;
  private BufferedReader in;

 public String sendMessage(String msg) {
   try {
     out.println(msg);
     return in.readLine();
   } catch (Exception e) {  return null;   }
 }
  ...
    sendMessage("get")

    sendMessage("put&"+1);

    sendmessage("incr");
```

Counting server



The count is similar to a volatile int

SocketCountingThreads

Experimenting with the shared counter:

1. Clients increments locally

```
int c=
  Integer.parseInt(sendMessage("get"));
c= c+1;
sendMessage("put&"+c);
```

2. Server locking (~volatile)

```
sendMessage(incr);
```

3. Clients and server on same PC

4. Clients and server on different PCs (local network)

Various observations

Clients increment locally using two messages (no locking)

Run-time: ~ 41 mS Lots of increments lost on server

Synchronized increment counter on client (client locking)

Run-time: ~ 4.5 mS No increments lost

Increment counter on server (server locking)

Run-time: ~ 22 mS No increments lost

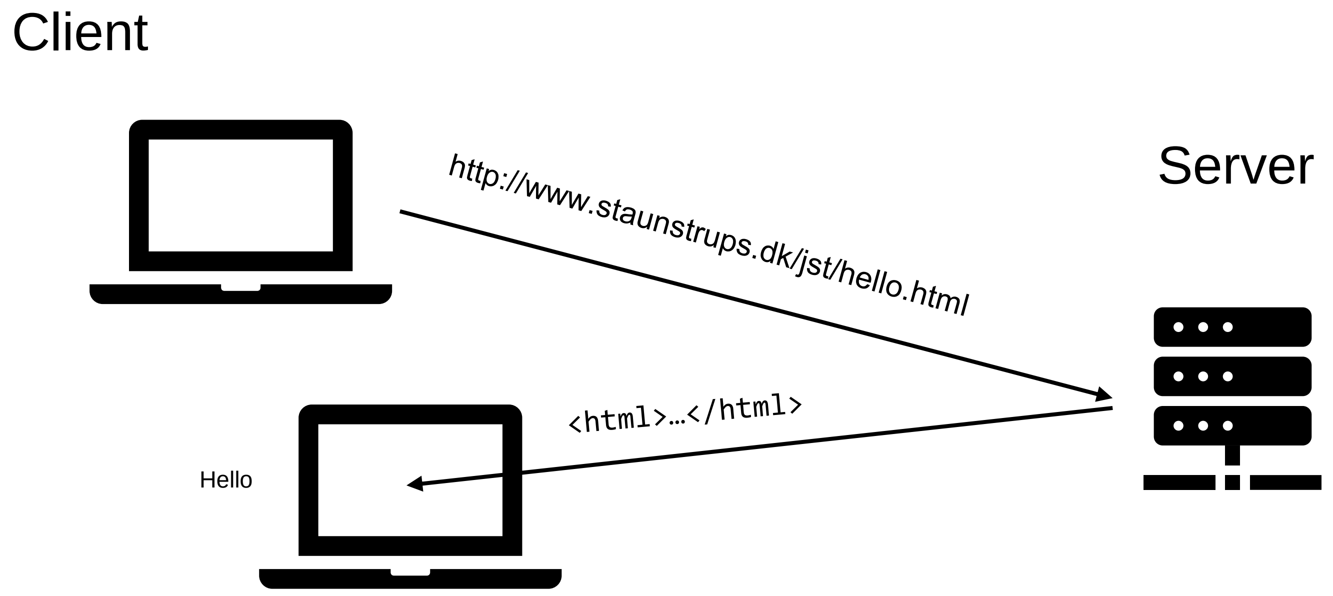
Increment a local counter (sort of non-volatile) ~ 0.8 mS

Addressing (server)

```
private final static String URL=
  //"127.0.0.1";      // this PC
  //"localhost";      // this PC
  //"192.168.1.204"; // other PC onlocal network
  "XPS-13";           // other PC onlocal network (hostname)
```

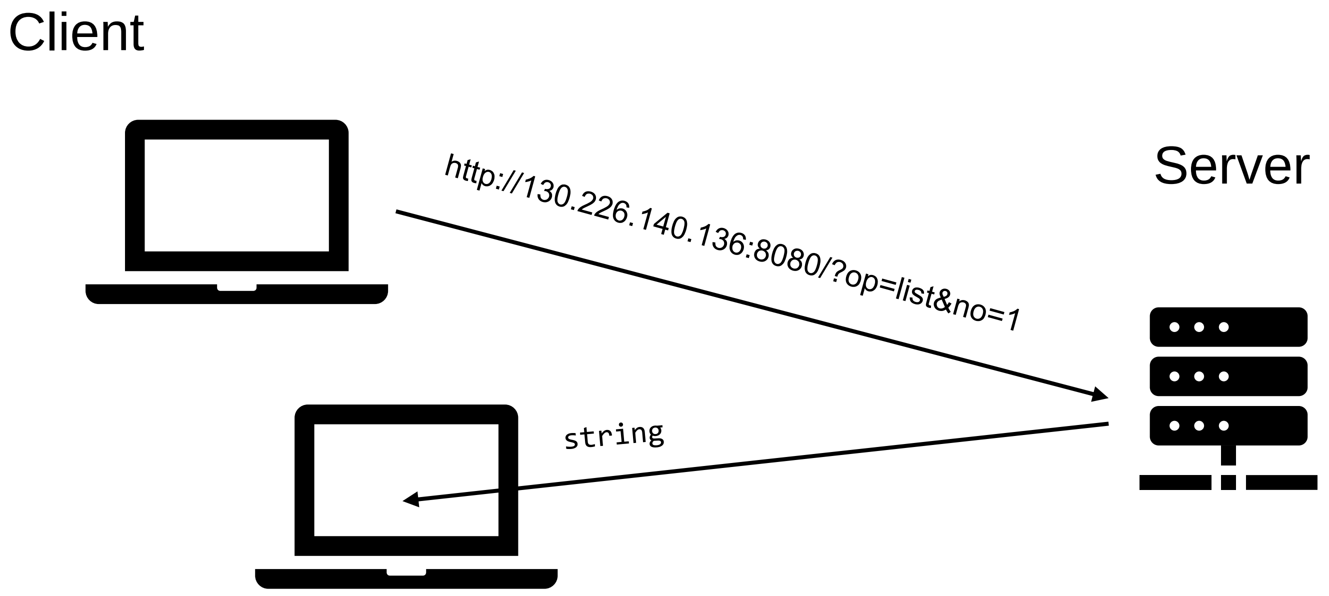
```
Increment counter on server (server locking)

Run-time (localhost):    ~  22 mS          No increments lost
Run-time (local wifi):   ~ 245 mS          No increments lost
```

HTTP protocol



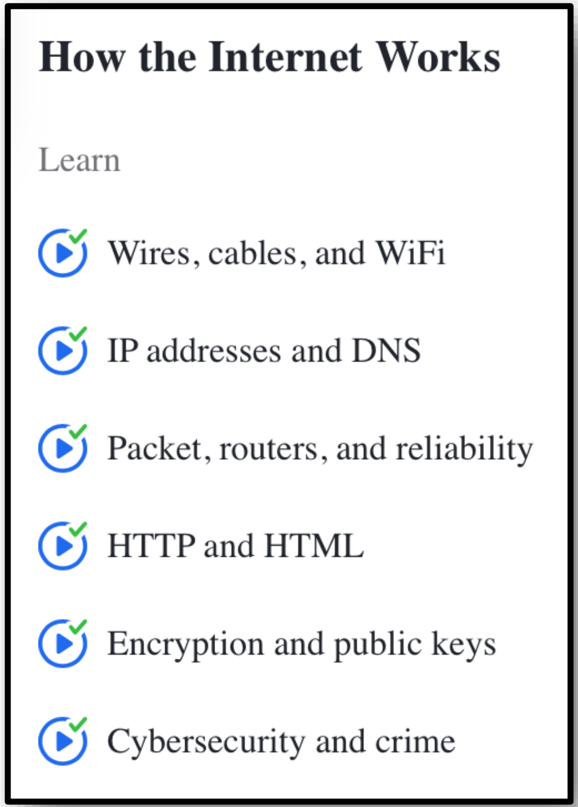
HTTP is asymmetric: **only the client can initiate communication** **and** the server forgets the request when the answer has been sent

HTTP (AnswerServer)



The server returns a plain list

Khan Academy (code.org)


How the Internet Works

Learn

- Wires, cables, and WiFi
- IP addresses and DNS
- Packet, routers, and reliability
- HTTP and HTML
- Encryption and public keys
- Cybersecurity and crime

Excellent videoes explaining how the internet works

https://www.khanacademy.org/partner-content/code-org/internet-works

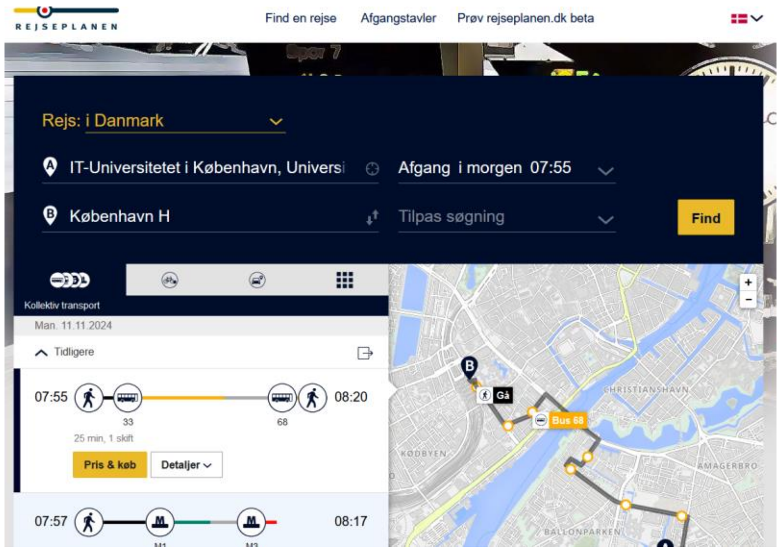
NetworkFetcher

Fetching an HTTP page

```
public class NetworkFetcherT {
  private static final String TAG= "NetworkFetchr";
  public byte[] getUrlBytes(String urlSpec) throws IOException {
    URL url= new URL(urlSpec);
    HttpURLConnection connection= (HttpURLConnection)url.openConnection();
    try {
      ByteArrayOutputStream out= new ByteArrayOutputStream();
      InputStream in= connection.getInputStream();
      if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
        throw new IOException(connection.getResponseMessage() +
            ": with " +  urlSpec);     }
      int bytesRead= 0;
      byte[] buffer= new byte[1024];
      while ((bytesRead = in.read(buffer)) > 0) {
        out.write(buffer, 0, bytesRead);
      }
      out.close();
      return out.toByteArray();
    } finally {
      connection.disconnect();
    }
  }
```

code-exercises/…/NetworkFetcher

Your personal "Rejseplan"



or

Simple Java program

```
Bus 33: 11:59 mod Rådhuspladsen St. (H.C. Andersens Boulevard)
Bus 33: 12:02 mod Nøragersmindevej (Kongelundsvej)
Bus 33: 12:14 mod Rådhuspladsen St. (H.C. Andersens Boulevard)
Bus 33: 12:17 mod Dragør Stationsplads
```

Finding your bus stop

https://xmlopen.rejseplanen.dk/bin/rest.exe/departureBoard?offsetTime=0&format=json&id=xxx

Replace xxx with a string e.g.

Lyngby

Vesterport

Personalized rejseplan

Rejseplanen has an open API, see file ReST_documentation_Rejseplanen_Latest.pdf

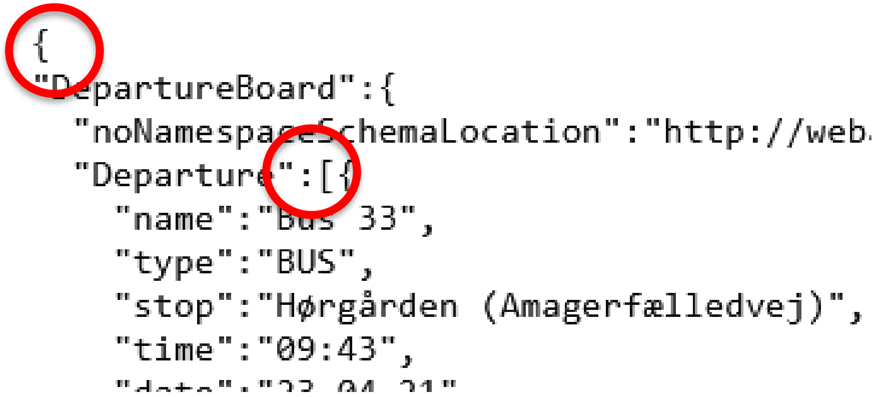
```
public class BusDepart {

  final static String RejseplanURL =
    "https://xmlopen.rejseplanen.dk/bin/rest.exe/departureBoard?offsetTime=0&format=json&id=";
  final static String ITU = "000000900";

  NetworkFetcher nf= new NetworkFetcher();

  public BusDepart(){
    byte[] res= null;
    try { res= nf.getUrlBytes(RejseplanURL+ITU);
    } catch (IOException e) {  System.out.println(e.getMessage());   }
    System.out.println(new String(res, StandardCharsets.UTF_8));
  }
  public static void main(String[] args) {  new BusDepart();  }
}
```

code-exercises/…/NetworkFetcher

https://xmlopen.rejseplanen.dk/bin/rest.exe/departureBoard?offsetTime=0&format=json&id=000000900

JSON version of "rejseplanen"

See GitHub week11: ReST_documentation_Rejseplanen_Latest.pdf



```
{
"DepartureBoard":{
  "noNamespaceSchemaLocation":"http://web
  "Departure":[{
    "name":"Bus 33",
    "type":"BUS",
    "stop":"Hørgården (Amagerfælledvej)",
    "time":"09:43",
    "date":"23.04.21"
```

```
JSONObject depBoard= jsonBody.getJSONObject("DepartureBoard");
JSONArray depArray= depBoard.getJSONArray("Departure");
if (depArray.length()>0) {
for (int i=0; ((i<depArray.length() && (found<4))); i++) {

   String bName= depArray.getJSONObject(i).getString("name");
   ...

}
```

JSON

JSON:

lightweight data interchange format

Java**S**cript **O**bject **N**otation

JavaScript object

```
var item= {
  what: "can",
  whereC: "metal"
};
```

JSON (String):

```
{"what":"can", "whereC":"metal"}
```

JSON String is a serialized version of a JavaScript object

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

Fetching elements from a JSONSTRING

Object

```
o: {"what":"can", "whereC":"metal"}
```

```
o.getString("what")
o.getString("whereC")
```

Array

```
       0         i
a: [   ...   { }    ...          ]
```

```
a.getJSONObject(i)
```

JSON library

```
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
```

Tutorial: https://www.w3schools.com/js/js_json.asp

build.gradle

```
...
dependencies {
    // Use JUnit test framework.
    testImplementation 'junit:junit:4.13.2'

    // This dependency is used by the application.
    implementation 'com.google.guava:guava:30.1.1-jre'

    implementation 'org.json:json:20240303'
...
}
```

Rejseplanen info in Java

BusDepart.java

and

NetworkFetcher.java

Both in exercises directory

```
Bus 33: 11:59 mod Rådhuspladsen St. (H.C. Andersens Boulevard)
Bus 33: 12:02 mod Nøragersmindevej (Kongelundsvej)
Bus 33: 12:14 mod Rådhuspladsen St. (H.C. Andersens Boulevard)
Bus 33: 12:17 mod Dragør Stationsplads
```