Assignment, part 2

Statement and concepts

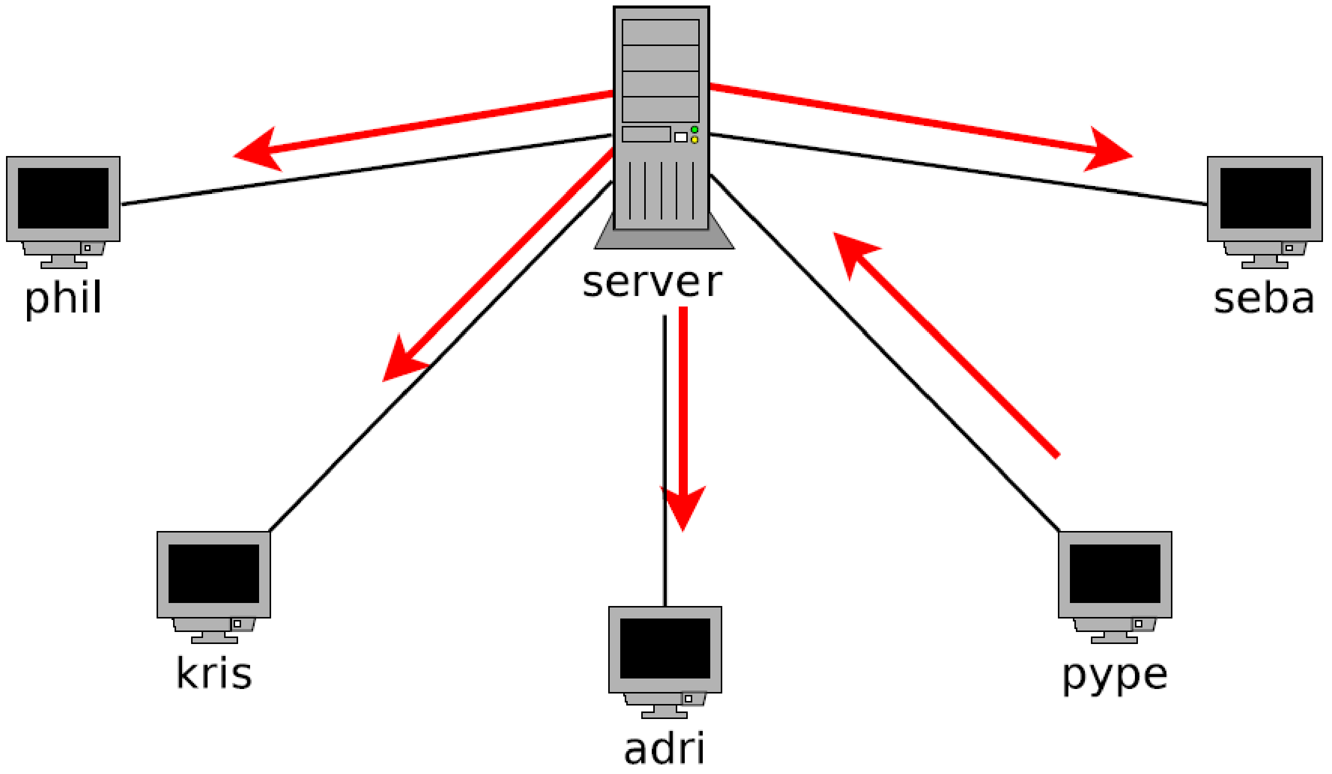
INFO-0010

Outline

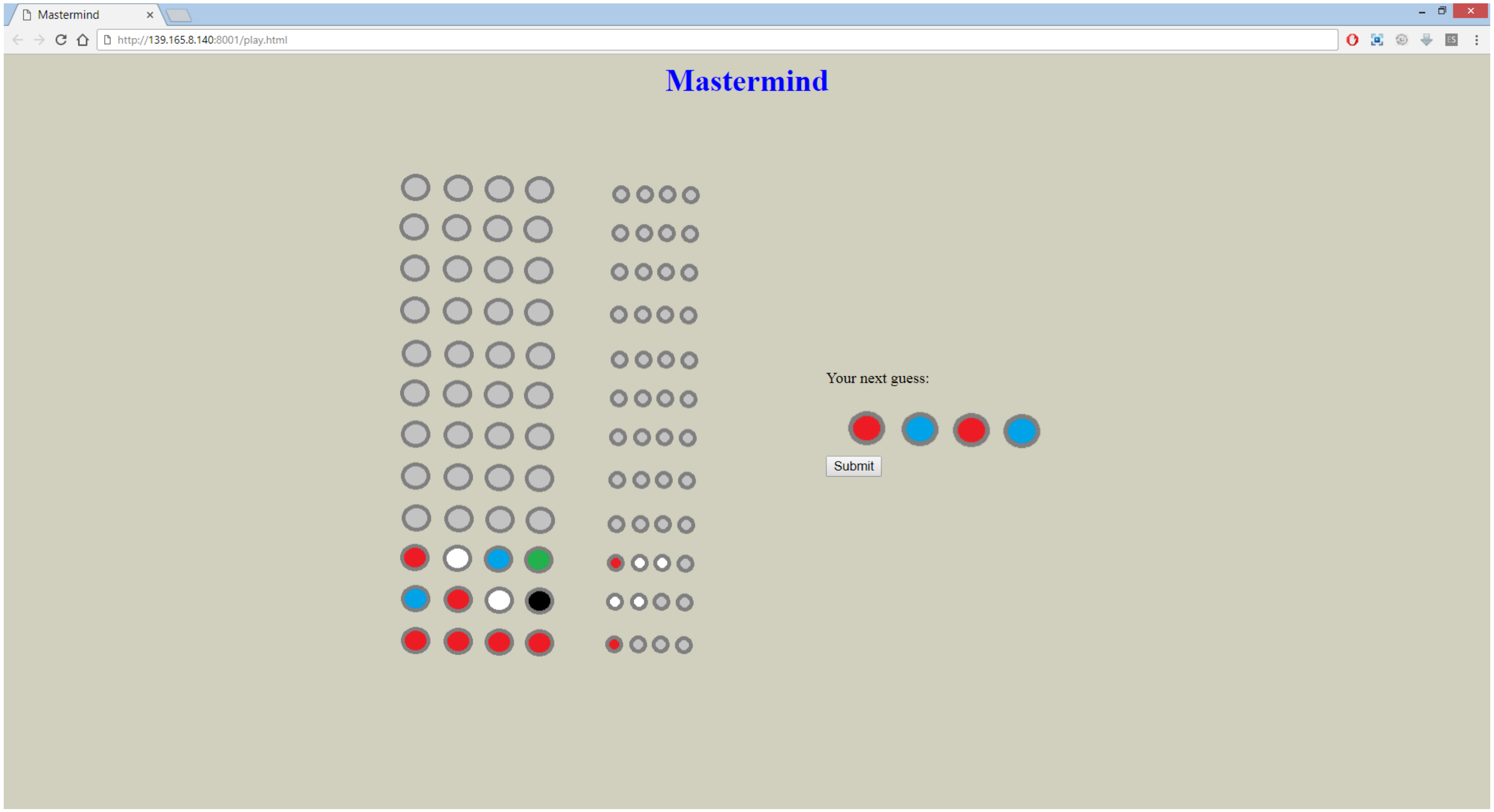
- Statement
- Implementation of concepts

Objective

Mastermind game using HTTP GET and HTTP POST methods

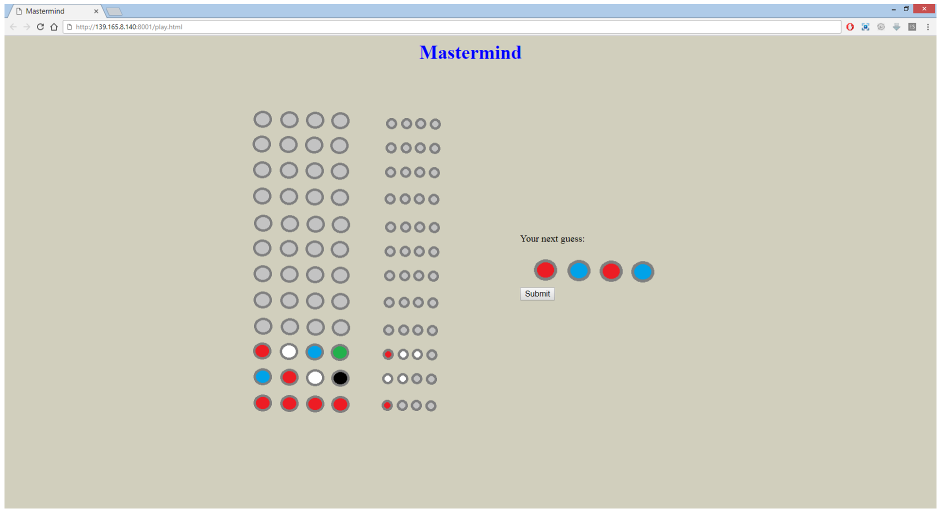


The platform

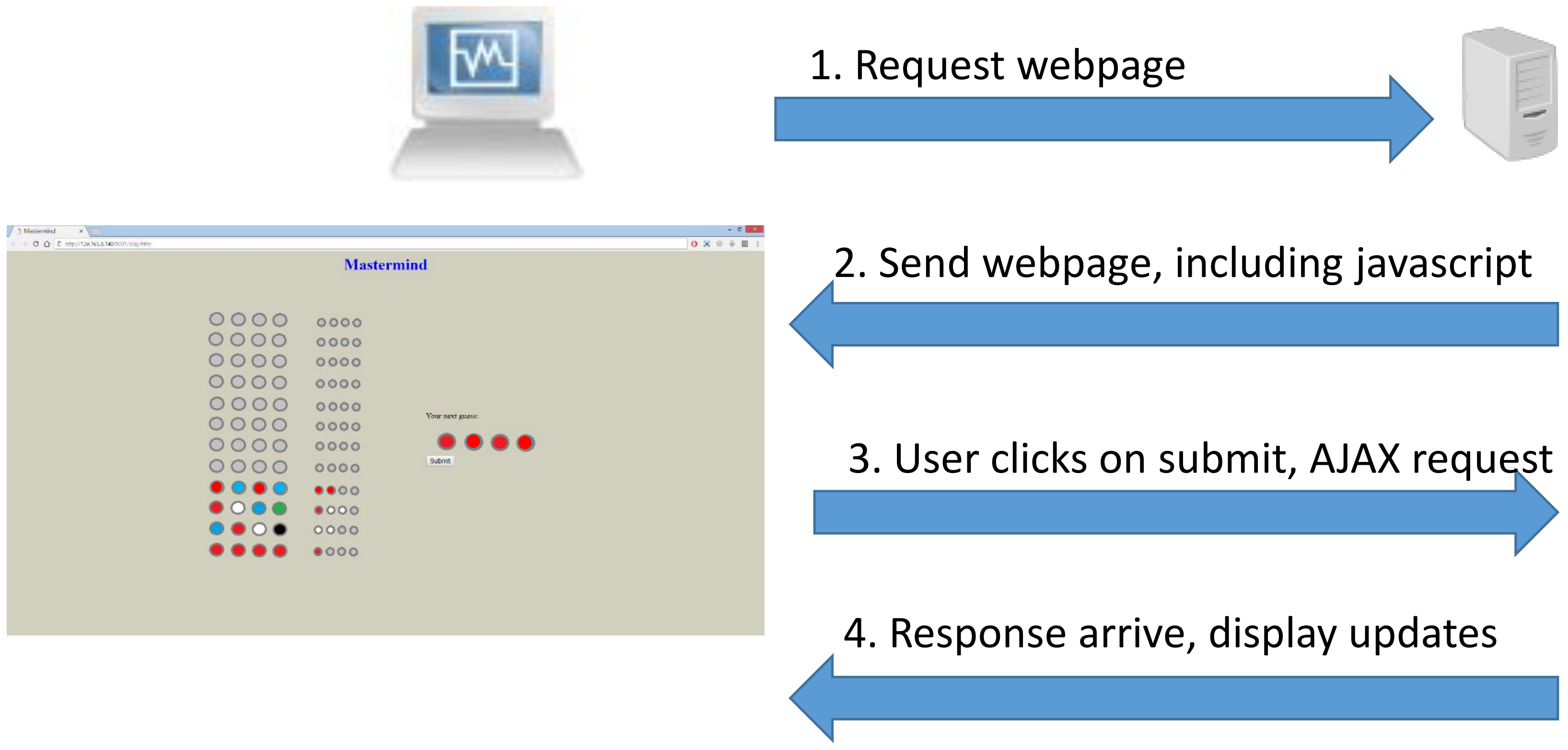


Architecture

$$\begin{tikzcd}\text{Root page ("/")} \arrow[d, "\text{redirection}"] \\ \text{/play.html}\end{tikzcd}$$



Protocol


1. Request webpage
2. Send webpage, including javascript
3. User clicks on submit, AJAX request
4. Response arrive, display updates
Mastermind
Your next guess:
Submit

5. Image displayed. Javascript executing (cycle through colors by clicking), ready for new AJAX requests

Javascript disabled? Replace the form with a traditional POST one (reload entire page).

Sessions

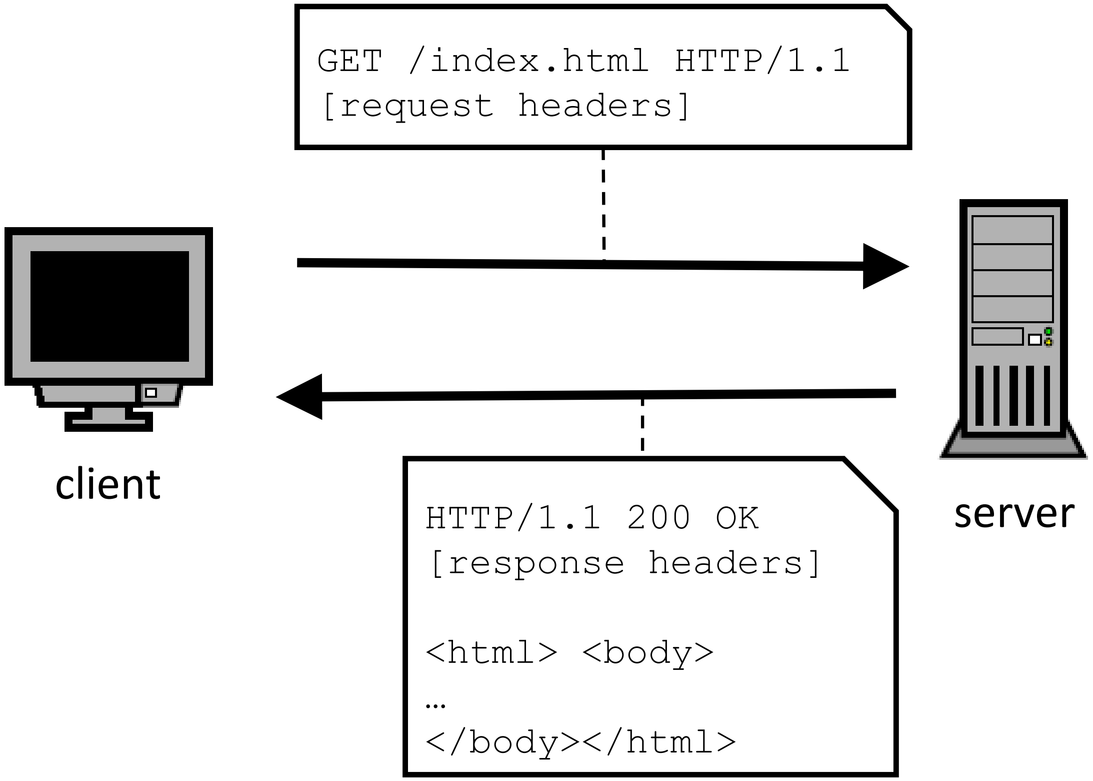
- HTTP is a *stateless* protocol.
  - For each request, the browser usually* opens a new TCP connection, send the HTTP request, receives the response, displays the content, then closes the TCP connection.
- But the user wants the server to keep a track of where he left off during the game.
- Solution? Sessions.
- On the first connection, the server generates a *cookie* that is sent to the client. This cookie is sent back by the browser at each HTTP request, such that the server can keep a trace of the session state.

*Unless specified in the options, see later

Outline

- Statement
- Implementation of concepts

The HTTP protocol

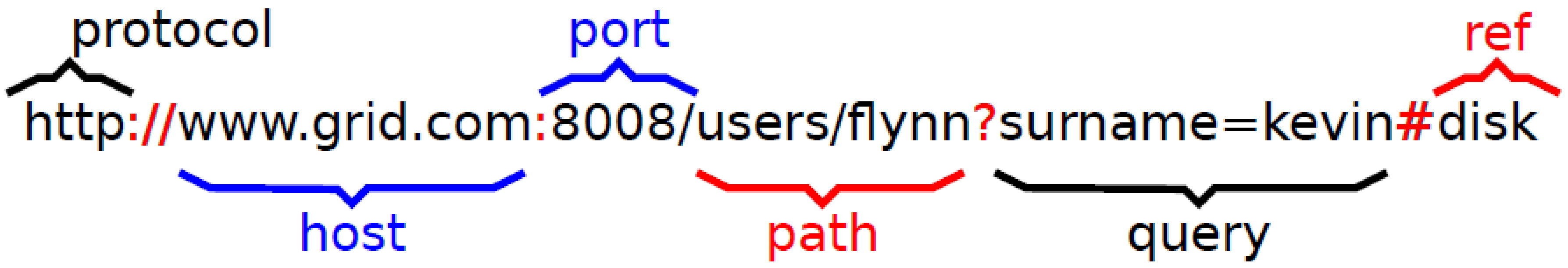


```
GET /index.html HTTP/1.1
[request headers]
```

```
HTTP/1.1 200 OK
[response headers]

<html> <body>
…
</body></html>
```

- Applicative protocol over TCP.
- Retrieve resources (files, web pages) on a machine
- Text-oriented protocol (vs. e.g. binary BitTorrent)
- Options (content-type, length, etc) : cf RFC2616

Java.net.URL



- Internet resources and services are typically identified and located through Uniform Resource Identifier/Locator.
- beware : parsing them properly is harder than you might think.

  Many fields are optionals,
- `u=new URL(<string>)` creates an internal object from an URL, ready to use
- accessor methods extract parts you need, e.g. `u.getPort()`
- make sure you use what is expected by the protocol

  (e.g. `GET /path?query HTTP/1.1`, no host, no ref, no port here !)

HTTP example

```
GET /pixs/bilourun3b.gif HTTP/1.0CRLF
User-Agent : Wget/1.12 (linux-gnu)CRLF
Accept : */*CRLF
Host : dsgametools.sf.net[1]CRLF
Connection : Keep-Alive[2]CRLF
CRLF
```

```
HTTP/1.1 200 OKCRLF
Server : Apache/2.2.3 (CentOS)CRLF
Last-Modified : Wed, 17 Feb 2010
09 :14 :23 GMTCRLF
ETag : "3a6-47fc846b079c0"CRLF
Cache-Control : max-age=259200CRLF
Expires : Sun, 06 Mar 2011
21 :15 :26 GMTCRLF
Content-Type : image/gifCRLF
Content-Length : 936CRLF
Date : Thu, 03 Mar 2011 21 :15 :26
GMTCRLF
Connection : keep-aliveCRLF
CRLF
GIF89a[3]...........`.@h.8..X.0x.8.85
@...P....................... !.......
!..NETSCAPE2.0...
```

1. Mandatory
2. Better use `Connection: close` for your tests
3. Binary content. Charset settings are important in that case.

HTTP response codes

- The first digit of the response code classifies the category of the response:

  - 1xx : Information
  - 2xx : Successful
  - 3xx : Redirection
  - 4xx : Client Error
  - 5xx : Server Error

HTTP response codes

- HTTP uses a few dozens of codes, and considering all of them is outside the scope of this assignment.
- However, I expect you to use the following codes:

  - 200 : OK
  - 303 : See Other
  - 400 : Bad Request
  - 404 : Not Found
  - 405 : Method Not Allowed
  - 411 : Length Required
  - 501 : Not Implemented
  - 505 : HTTP Version Not Supported

POST methods

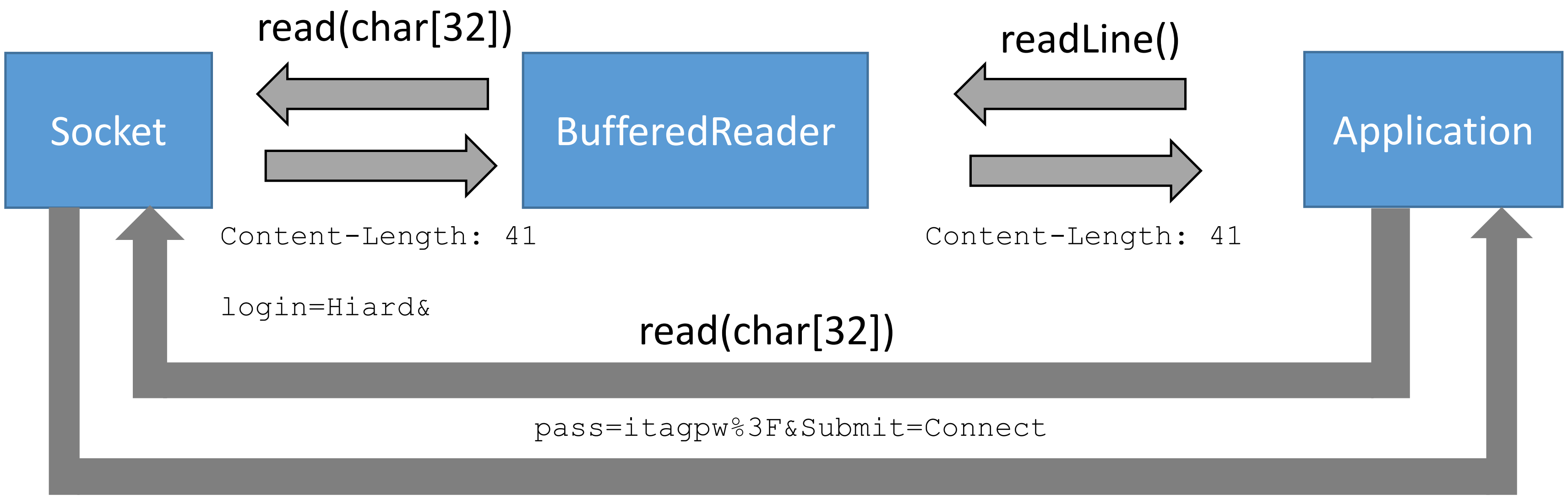
- Content follows the header
- `Content-Length` field

```
POST /identification.HTML HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

login=Hiard&pass=itagpw%3F&Submit=Connect
```

Beware of BufferedReader

- BufferedReader makes buffered reads!
  - i.e. maybe it already read more on the Socket than what you requested

```
...
Content-Length: 41

login=Hiard&pass=itagpw%3F&Submit=Connect
```



Redirections

- Automatically makes the browser request another page without the need for the user to do anything.
- At least two ways:

```
HTTP/1.1 303 See Other
Location: http://localhost:8088/viewImages.html
```

or

```
HTTP/1.1 200 OK
Content-Length: 53
(other header fields)

<script>document.location="viewImages.html";</script>
```

- In this work:
  - Redirection to `/play.html` when `/` is requested

(Session) cookies

- Setting a cookie:

```
Set-Cookie: SESSID=rk64vvmhlbt6rsdfv4f02kc5g0; path=/
```

The browser will automatically include

```
Cookie: SESSID=rk64vvmhlbt6rsdfv4f02kc5g0
```

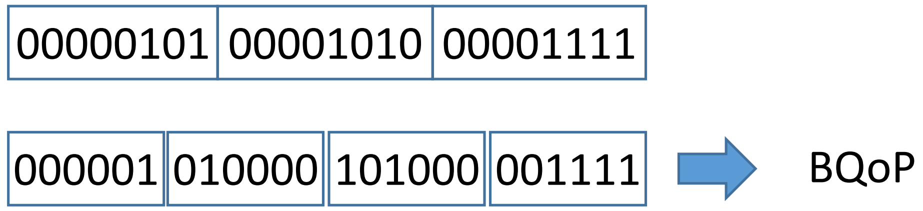
in all of its requests.

- Invalidating a cookie:

```
Set-Cookie: SESSID=deleted; path=/; expires=Thu, 01 Jan 1970 00:00:00 GMT
```

Base64

- Convert binary data into ASCII characters (e.g. for transferring images)
- 3 bytes → 24 bits → 4 groups of 6 bits
    - If less than 3 bytes, use padding (fill with 0's)
- Each of the 64 possible values is assigned to an ASCII character
- Used to encode an image in this assignment
- Example: 5 10 15 (decimal)



- Display a base64 image inside HTML

<img src="data:image/png;base64,iVBORw0K..." />

HTML

- A HTTP response usually contains a HTML webpage.
- HTML is a tag-oriented programming language.
- A tag is a character string starting with "<" and ending with ">", e.g. <b>,<em>, <div>, <p>, …
- Tags allow to structure the document, change the display of the content, add non-textual elements, …
- When a tag is opened, it should be closed (adding "/" as first character of the tag), e.g. "</div>".
  - OK : <b><em>This text in bold and italic.</em></b>
  - KO : <b><em>This text in bold and italic.</html>
  - KO : <b><em>This text in blod and italic.</b></em>
- A good HTML tutorial : https://www.w3schools.com/html/

Javascript

- Generated by the server (inside the returned webpage)
- Executed by the client
- Inside <script> tags
- Can be disabled (use <noscript>)
- Example:

```
<HTML>
  <HEAD>
    <TITLE> Welcome to my site</TITLE></HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JAVASCRIPT">
      <!--
         alert("Welcome to my site!");
      // -->
    </SCRIPT>
    <NOSCRIPT>This browser does not support javascript</NOSCRIPT>
  </BODY>
</HTML>
```

Monitoring events

- Javascript can help you monitor what's happening on the page (e.g. mouse click, key pressed, …)
- Attach a function to an event
  - on<eventname> : DOM0
  - addEventListener : DOM2
- Example :

```
<script>
  function WhichButton(event) {
    alert("You pressed button: " + event.button)
  }
</script>

<div onmousedown="WhichButton(event);">Click this text </div>
```

AJAX requests

- Create a (short) request to the server, such that the entire reloading of the page is not necessary
- Use XmlHTTPRequest to create your request
- Use document.getElementByID to locate the tag you want to update
- Use innerHTML to update the content inside a tag, or specify an attribute to change inside a tag

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
      // Action to be performed when the document is read;
      document.getElementByID("mytext").value = 'Ready';
   }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```

Chunked encoding

- With "regular" transfers (i.e. HTTP/1.0), the server has to know the last byte to send (therefore, the size of the entire message) before being able to start the transmission of the first byte (since `Content-Length` is in the header).
- HTTP/1.1 offered a new way of transmitting data : chunked encoding.
- The message is split into chunks of data, each chunk being sent as soon as available, preceded with its size, in hexadecimal.
- E.g.:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

18
This is a first chunk.

1A
And this is another one.

1C
But this one is cut in half
17
without carriage return
0
```

```
This is a first chunk.
And this is another one.
But this one is cut in half without carriage return
```

Gzip compression

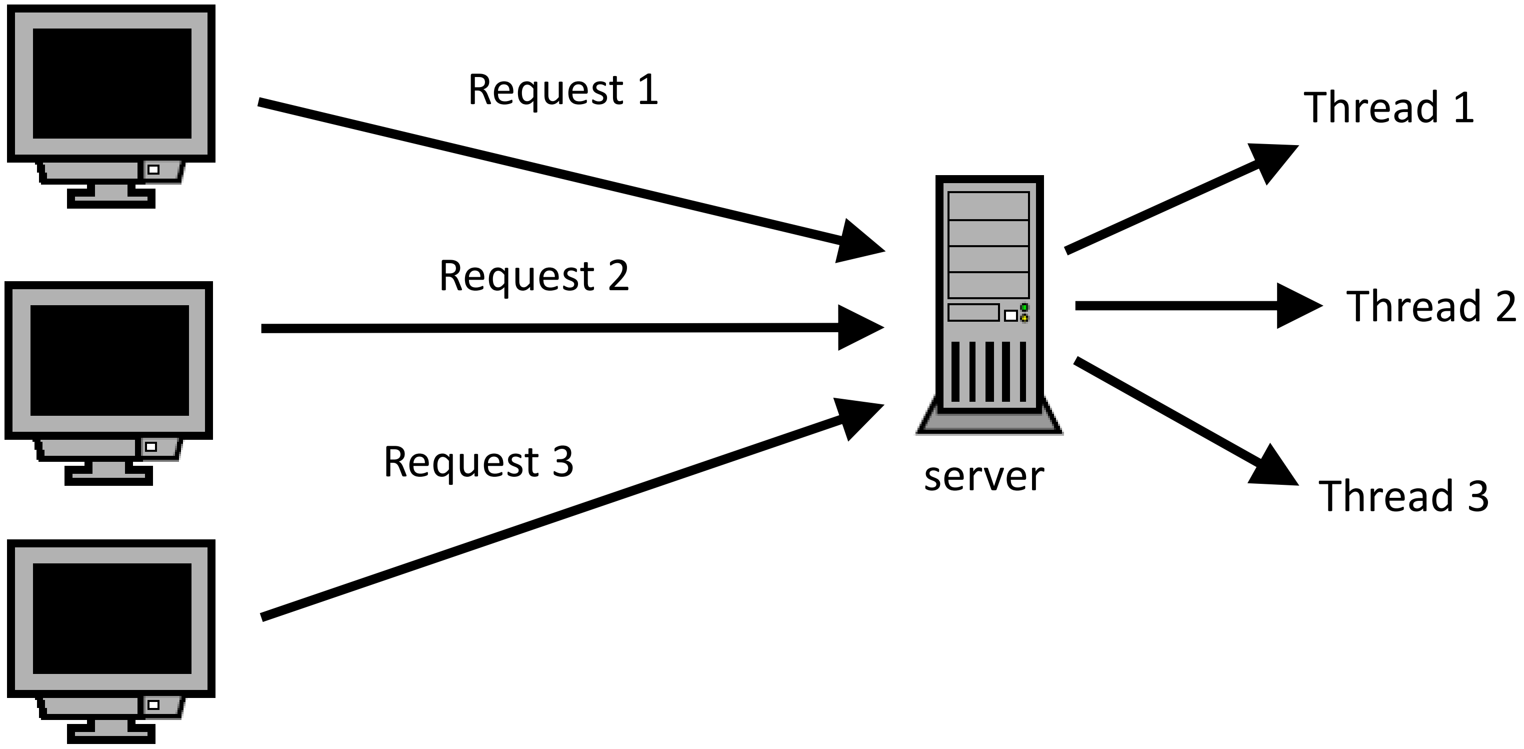
- As webpages were tending to become larger over time, the need for compression soon arose (otherwise, the ratio goodput/throughput could be quite low).

- Compression can be achived by passing data into a `GZipOutputStream`, and transmit the output of that stream.
  - However, this is now binary data, not text (so, ensuring that both sides use the same `charset` is important)
  - Compression is done firsly, then performing chunked encoding on the compressed data is done secondly.

- For bonus points, you can implement Gzip compression.

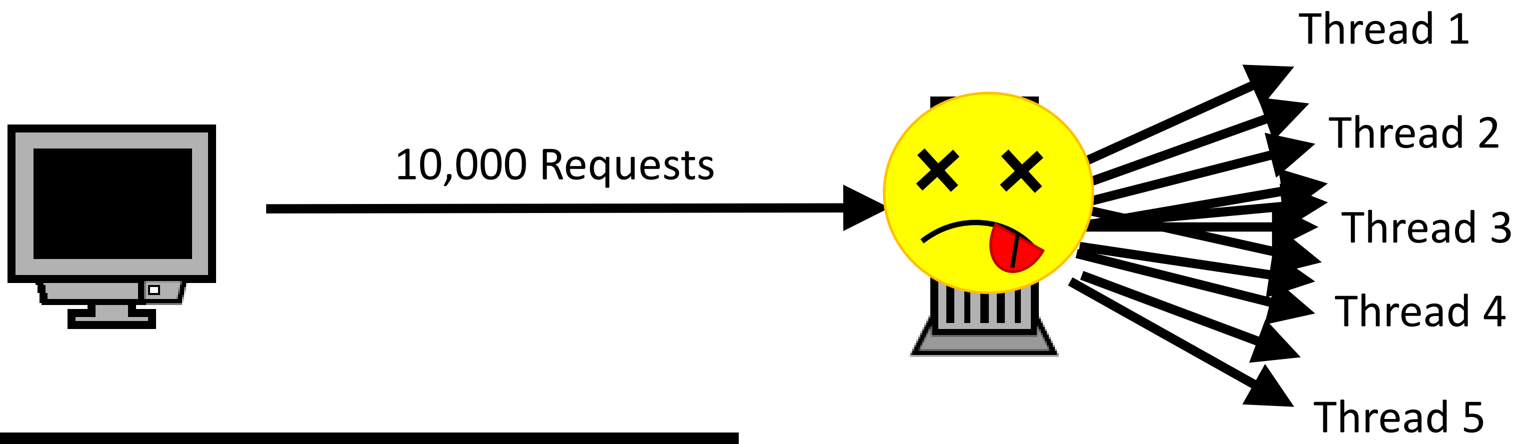
Thread pools

- Previously:



Thread pools

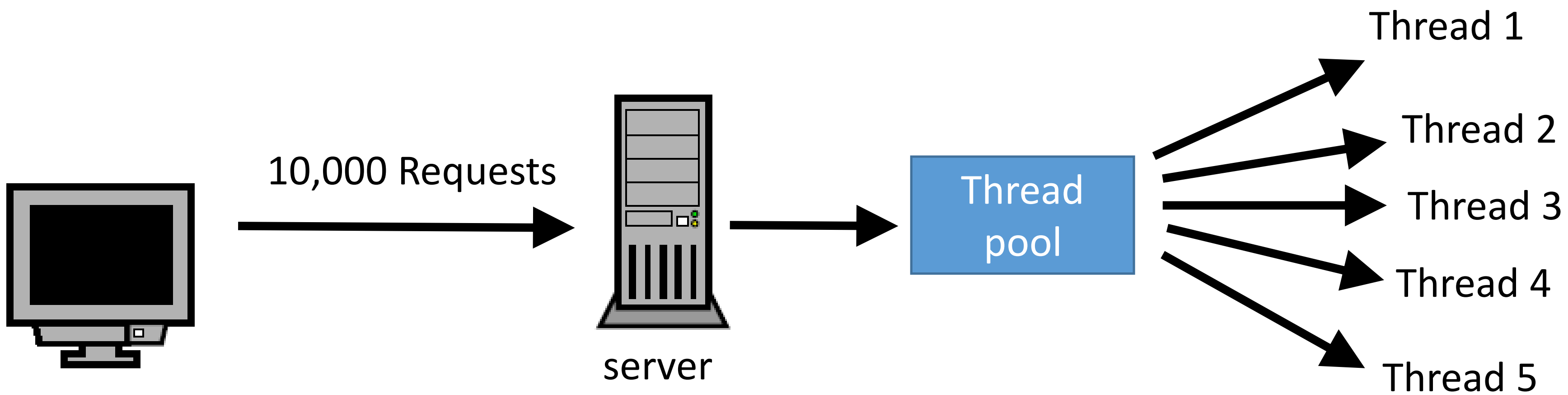
- Imagine the following attack:



```
Socket[] s = new Socket[10000];
for(int i = 0; i < 10000; i++)
{
     s[i] = new Socket("100.100.100.100",80);
}
```

Thread pools

- With a thread pool:



1. Handle the first 5 requests (9,995 remaining)
2. As soon as a thread finishes, it returns to the pool and receives a new task
3. When all tasks are done, each thread is back in the pool, ready for new tasks

Possible loss of speed performance, but increased robustness

Guidelines

- Deadline : 13th of May.
- Work by groups of two students.
- Check that your program works on ms8** machines with Firefox (Don't have an account? Contact Marc Frédéric). Launch the server on a given machine, and try to access it from a different machine.
- Guidelines of the first part still apply (Java 1.8, no package instructions, no file manipulation, don't intercept CTRL-C, …).