

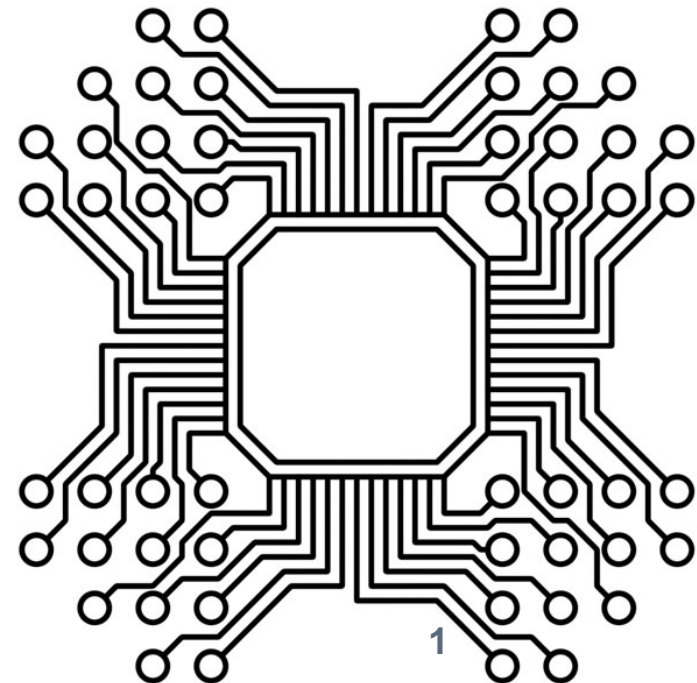
# WEB SERVER

---

Lecturer: Dr. Bui Ha Duc

Dept. of Mechatronics

Email: [ducbh@hcmute.edu.vn](mailto:ducbh@hcmute.edu.vn)

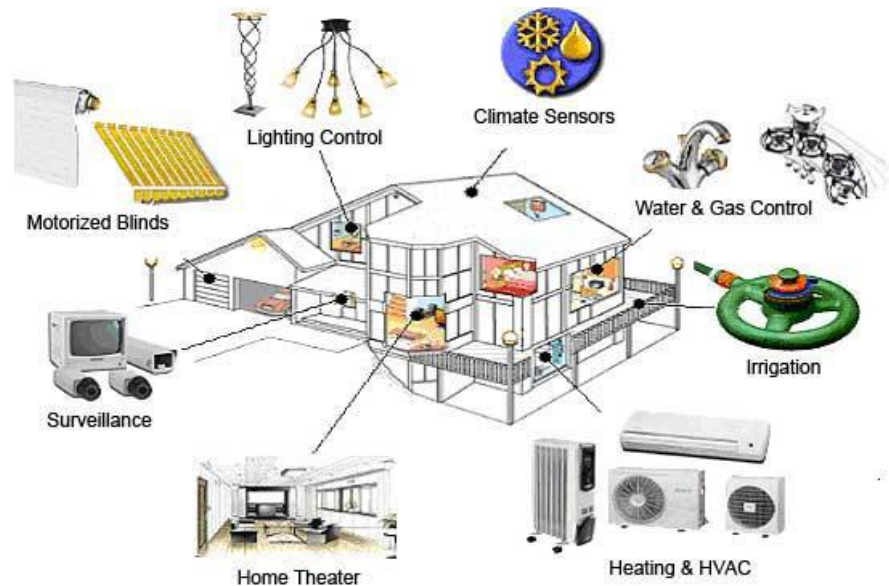


# Problems

- What is web server and its functions?
- How to create a web server?
- How to communicate / transfer data between the web server and the Friendly ARM?
- Why do we need an embedded system?

# Web server

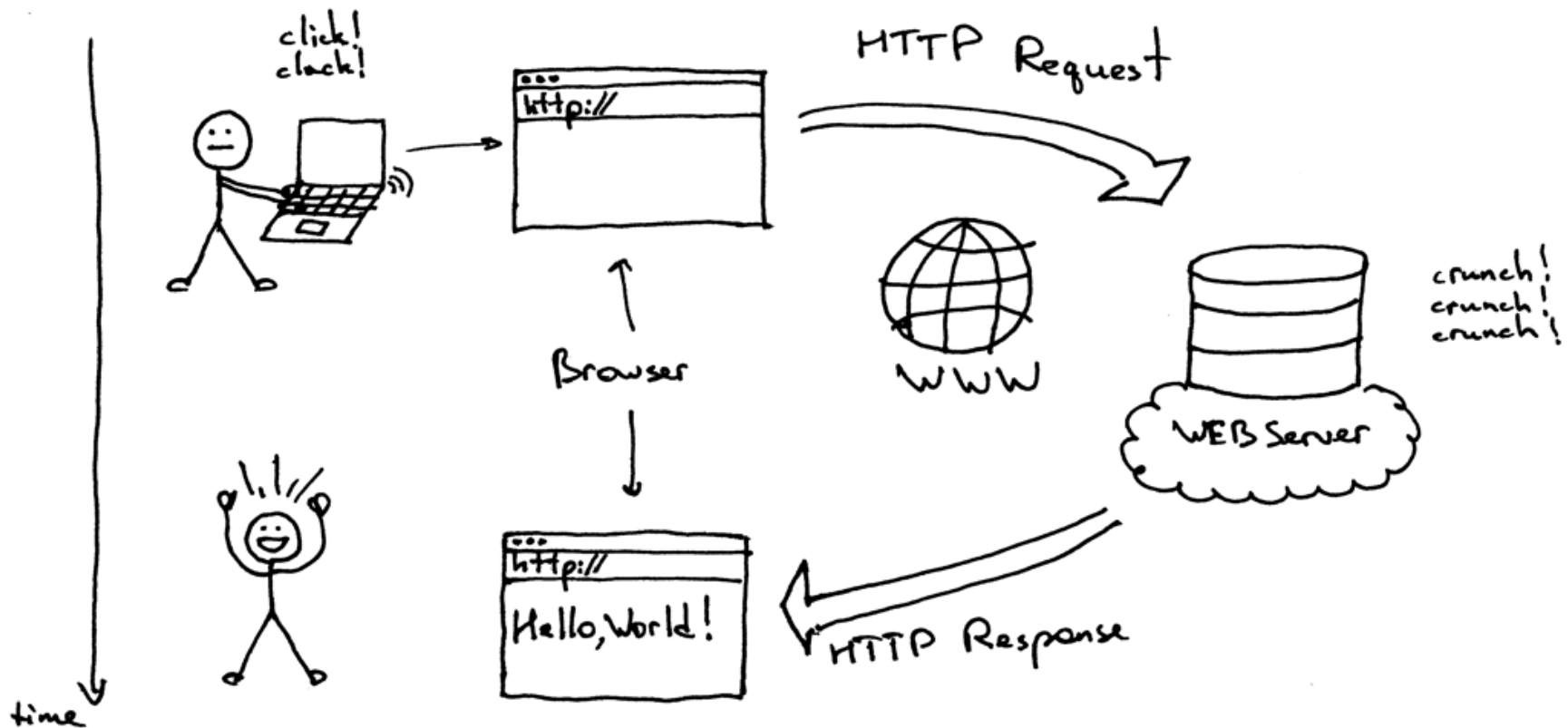
- Web server hosts [a web site](#) and provides [reliable services](#) for any requesting client.
- Web server communicate with clients via different protocols: TCP, UDP, IP...



Open sourced  
webservers for home  
automation:

- Calaos
- Domoticz
- OpenHAB
- OpenMotics
- Home assistant

# Example of a web server



<https://ruslanspivak.com/lbaws-part1/>

# Creating a web server

## Apache Webserver:

- Most popular web server application
- Apache can serve HTML files over HTTP and dynamic web pages using scripting languages such as PHP javascript

## Procedures:

1. Install apache with the command:

```
sudo apt-get install apache2 -y
```

2. Find Raspberry IP with:

```
hostname -I
```

3. Access the website at raspberry IP

# Creating a web server

## Changing the default web page

- The default website is located at:

```
/var/www/html/index.html
```

- /var/www/html is a enclosed folder, owned by root.
- To modify, change the owner with:

```
sudo chown pi: index.html
```

Or 

```
sudo chown -R pi:pi /var/www/html
```

- Replace the index.html with your website

# Creating a website

## Web developing Languages:

- **HTML**: creating Web pages
- **CSS**: create website layout, describes how HTML elements are to be displayed on screen, paper, or in other media
- **Javascript**: makes website do what you want them to do
- **PHP**: making dynamic and interactive Web pages
- **Websocket**: lightweight client-server communication

Visit [www.w3schools.com](http://www.w3schools.com)

# Structure of a website

- <!DOCTYPE html>

<html>

<head>

Elements

<title>Page Title</title>

</head>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>

</html>



# HTML Elements

- HTML element structure:

**<tagname>**Content...**</tagname>**  
Start tagEnd tag

- HTML Elements

Start tag	End tag	Function
<code>&lt;h1&gt;</code>	<code>&lt;/h1&gt;</code>	Heading text
<code>&lt;p&gt;</code>	<code>&lt;/p&gt;</code>	Paragraph
<code>&lt;a link&gt;</code>	<code>&lt;/a&gt;</code>	Link address
<code>&lt;img&gt;</code>		Image
<code>&lt;table&gt;</code>	<code>&lt;/table&gt;</code>	Table
<code>&lt;div&gt;</code>	<code>&lt;/div&gt;</code>	Division, contains other elements
<code>&lt;form&gt;</code>	<code>&lt;/form&gt;</code>	Defines a form that is used to collect user input

# HTML Attributes

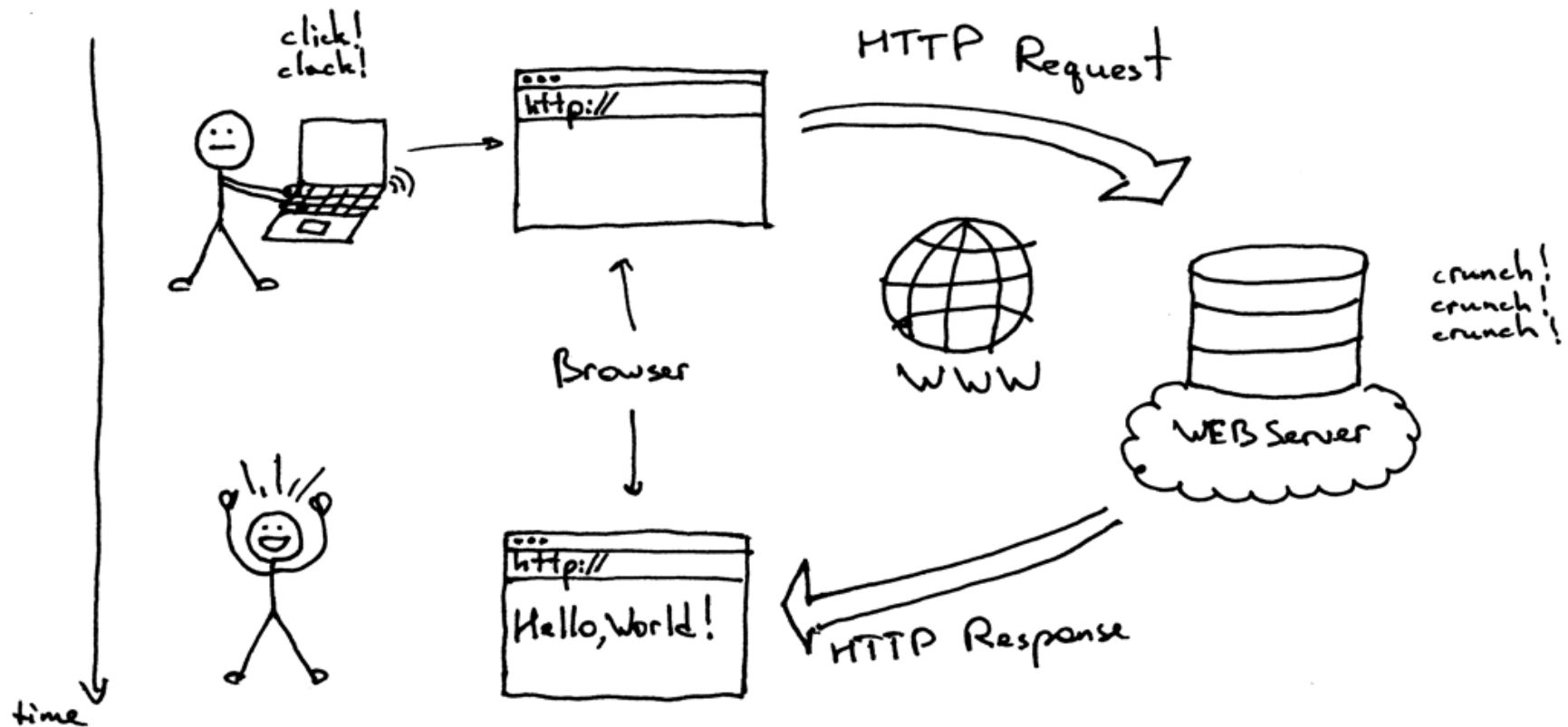
- Attributes provide **additional information** about an element.
- Attributes are always specified in **the start tag**

E.g. `<a href="https://www.w3schools.com">This is a link</a>`  
``

- HTML Attributes:

Attribute	Function
disabled	Specifies that an input element should be disabled
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the source (web address) for an image
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element

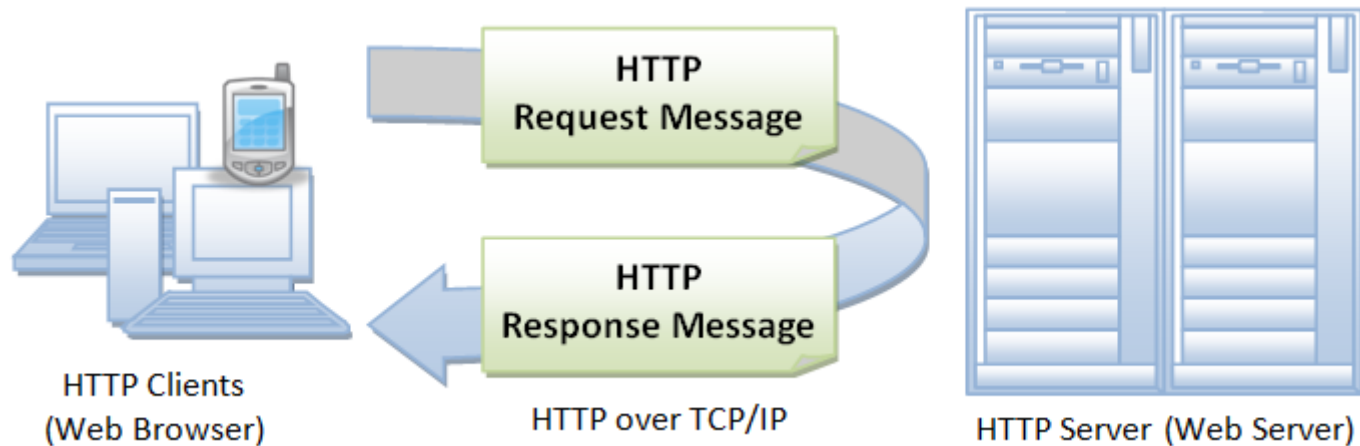
# Example of a web server



<https://ruslanspivak.com/lbaws-part1/>

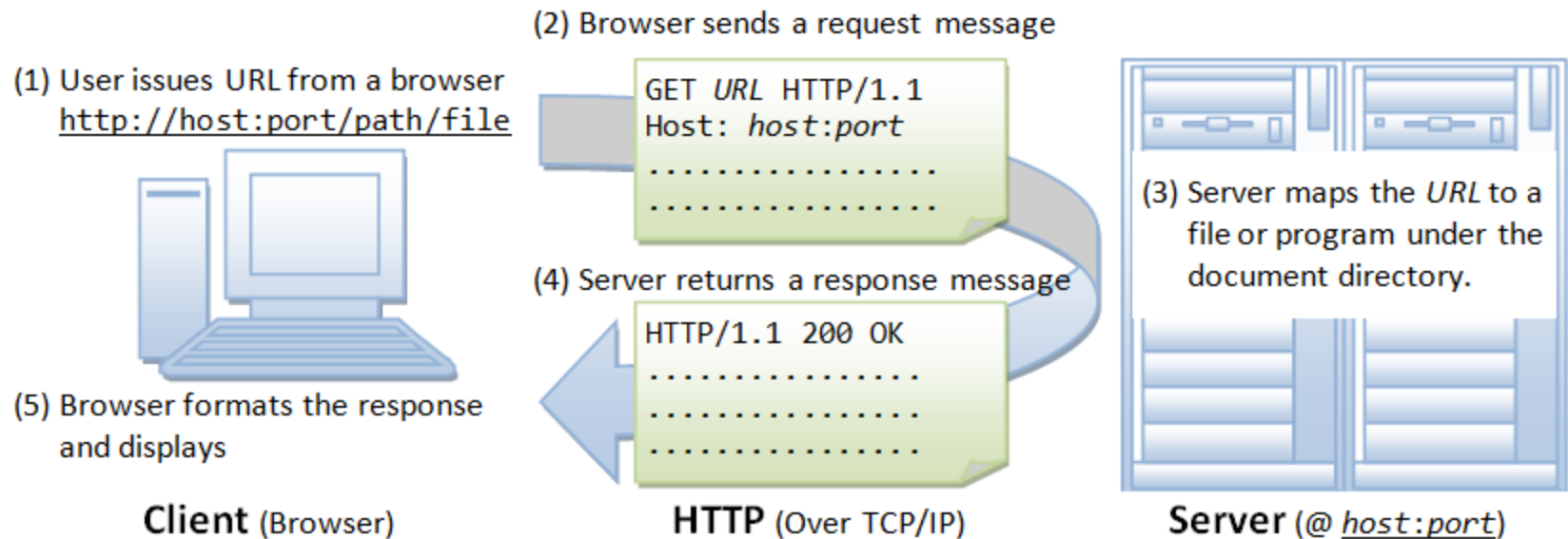
# HyperText Transfer Protocol (HTTP)

- HTTP is an *asymmetric request-response client-server* protocol
- HTTP is a stateless protocol



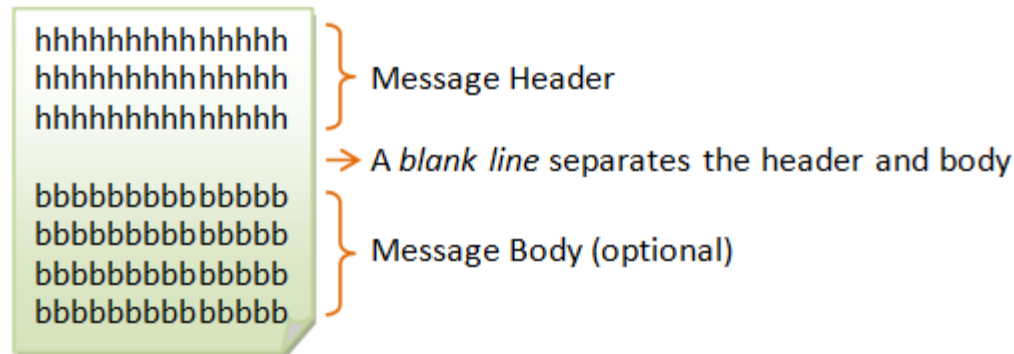
[http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http\\_basics.html](http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html)

# HyperText Transfer Protocol



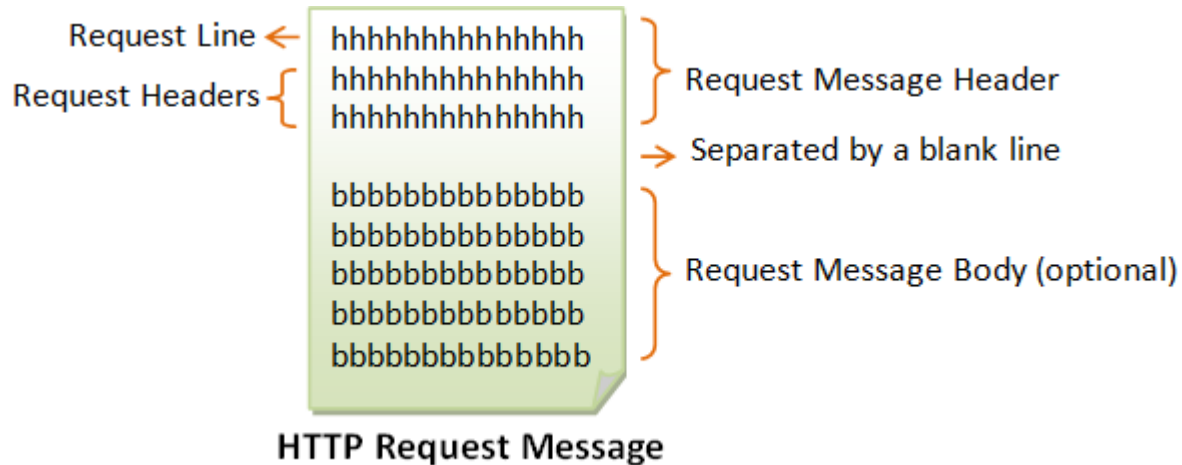
# HTTP Messages

- HTTP client and server communicate by sending **text messages**
- An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*.



HTTP Messages

# HTTP Request Message



- The request line has the following syntax:

*request-method-name request-URI HTTP-version*

- *request-method-name*: request methods, e.g., GET, POST, HEAD, and OPTIONS.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

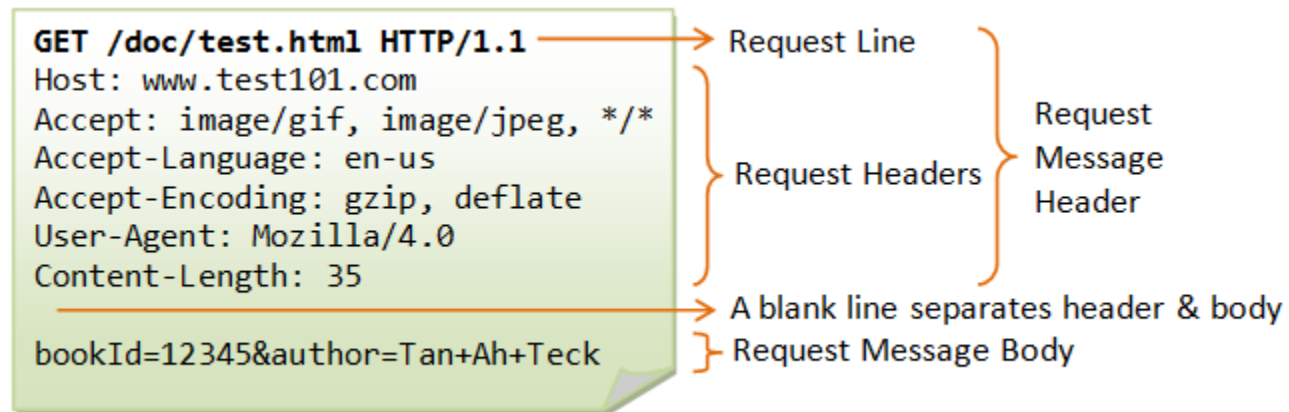
# HTTP Request Message

- The request headers Syntax:

*request-header-name: request-header-value1, request-header-value2, ...*

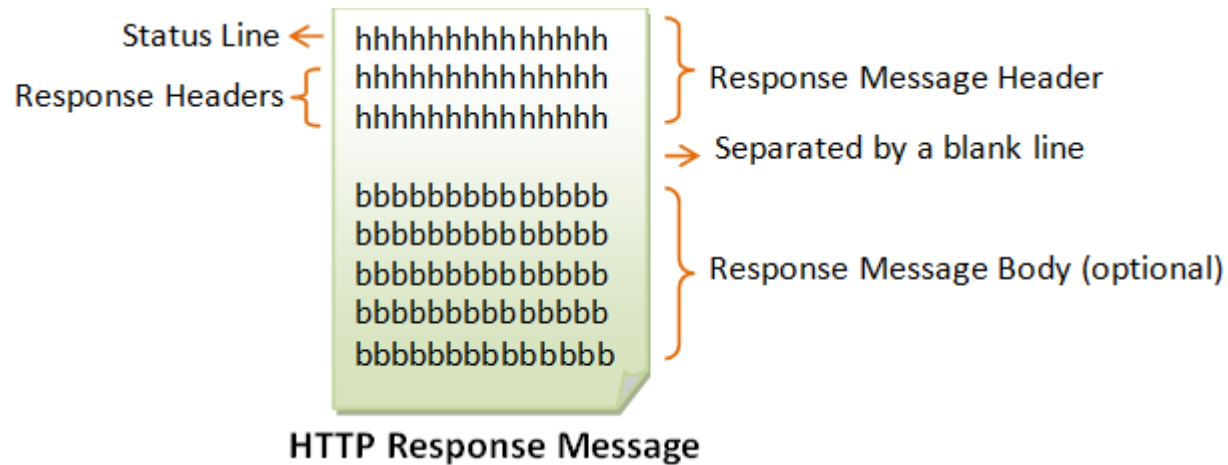
- request message body: contain user inputs

name1=value1&name2=value2...





# HTTP Response Message



- The status line has the following syntax:

*HTTP-version status-code reason-phrase*

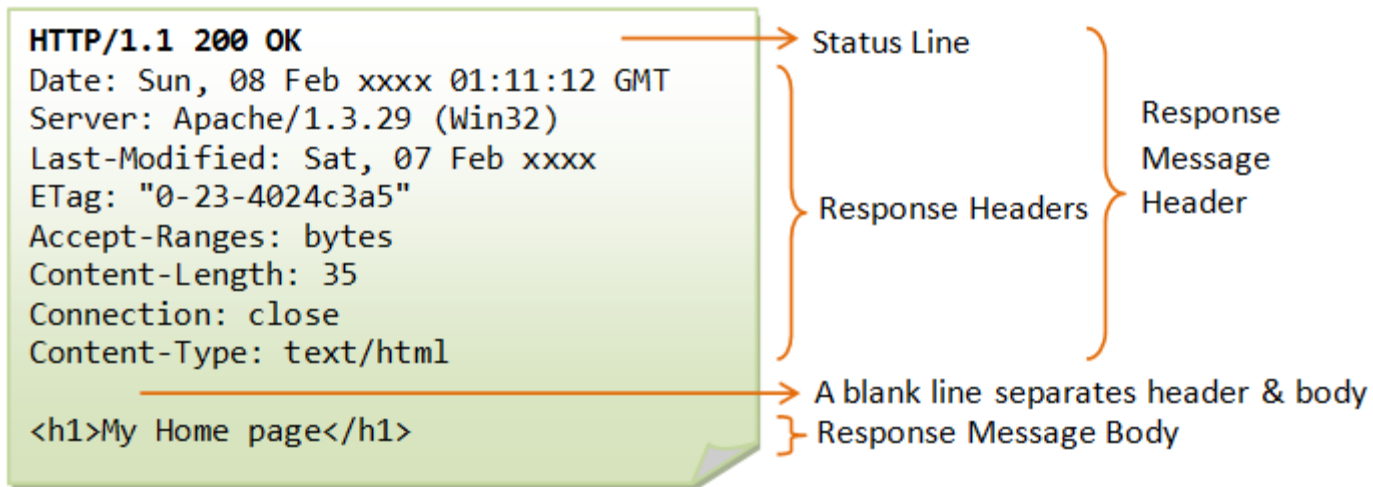
- *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- *status-code*: a 3-digit number generated by the server to reflect the outcome of the request. (200, 403, 404...)
- *reason-phrase*: gives a short explanation to the status code

# HTTP Response Message

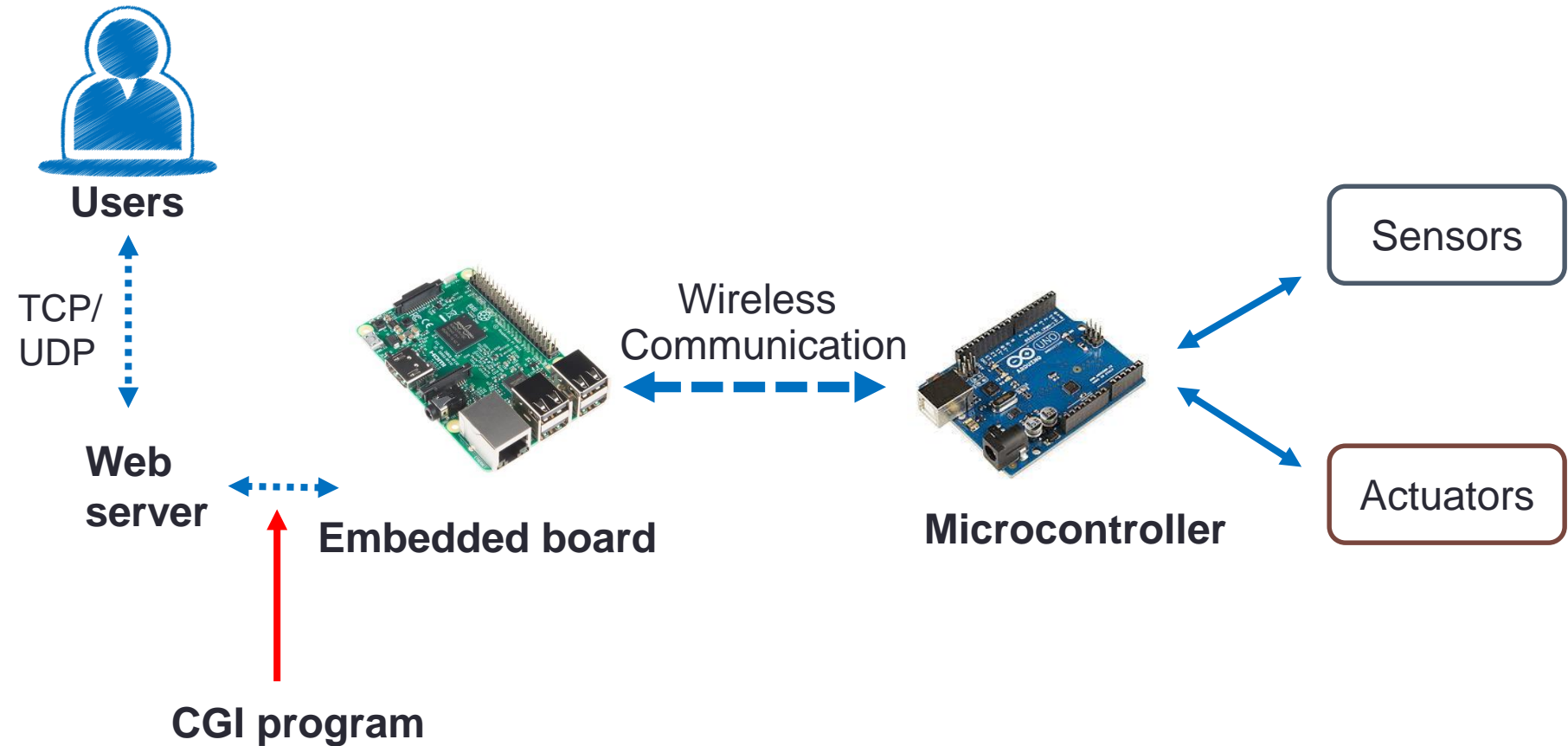
- The response headers syntax:

```
response-header-name: response-header-value1,  
response-header-value2, ...
```

- Response message body: return value/ website



# System Diagram



# What is a CGI

- CGI (Common Gateway Interface) is a program that runs on a web server.
- CGIs are typically called from HTML forms, and usually are designed to run quickly and response to the requests of clients.
- CGIs can be written in many languages: C, C++, Perl, Bash
- CGI program allows data to be sent to/ receive from a web server and processed them.
- The CGI script could actually return any type of response, but 99% of the time, the result of any CGI script is going to be plain HTML

- Add permission to access wiringPi

```
sudo adduser www-data i2c
sudo adduser www-data spi
sudo adduser www-data gpio
```

# CGI

```
#include <stdio.h>
#include <stdlib.h>
#define DATAFILE "../data/data.txt"
int main(void)
{
    FILE *f = fopen(DATAFILE, "r");
    int ch;
    if(f == NULL) {
        printf("%s%c%c\n",
            "Content-Type:text/html;charset=iso-8859-1", 13, 10);
        printf("<TITLE>Failure</TITLE>\n");
        printf("<P><EM>Unable to open data file, sorry!</EM>"); }
    else {
        printf("%s%c%c\n",
            "Content-Type:text/plain;charset=iso-8859-1", 13, 10);
        while((ch=getc(f)) != EOF)
            putchar(ch);
        fclose(f); }
    return 0;
}
```

**CGI program in C**

```
#!/bin/sh

type=0
period=1

case $QUERY_STRING in
    *ping*)
        type=0
        ;;
    *counter*)
        type=1
        ;;
    *stop*)
        type=2
        ;;
    *)
        type=0
        ;;
esac
```

**CGI program in BASH**

# References

- CGI program in C tutorial

<https://www.cs.tut.fi/~jkorpela/forms/cgic.html>

- CGI program in Bash

<http://www.team2053.org/docs/bashcgi/index.html>

# CGI program in C

## Steps to create a CGI using C language

1. Write a C program using notepad, IDE...  
e.g. `led_control.c`
2. Compile \*.c files to \*.cgi  
`gcc -o led_control.cgi led_control.c`
3. Copy the CGI file to webserver folder `/usr/lib/cgi-bin`
4. Change permission to execute cgi files with  
`chmod a+x led_control.cgi`



# Enable CGI on Apache

- By default, CGI is disabled.
- To enable CGI, type these commands:  

```
cd /etc/apache2/mods-enabled  
sudo ln -s ../mods-available/cgi.load
```
- Reload apache service:  

```
sudo service apache2 reload
```

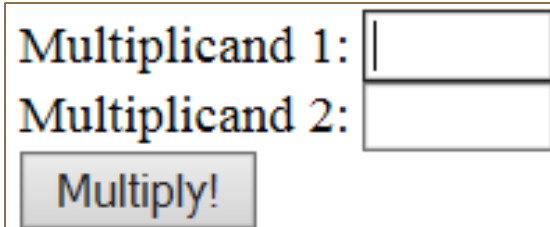
# CGI program in C

## First program

```
#include <stdio.h>
int main(void) {
    printf("Content-Type: text/plain;charset=us-ascii\n\n");
    printf("Hello world\n\n");
    return 0;
}
```

# CGI program in C

## Process a simple form



Multiplicand 1:

Multiplicand 2:

The library function **getenv** (defined in the standard library **stdlib**) is used to access the value in **QUERY\_STRING** as a string

Data is passed to the script or program in an environment variable called **QUERY\_STRING**.

```
<body>                                default METHOD="GET"
  <form action="mult.cgi" method="GET">
    <div><label>Multiplicand 1: <input name="m" size="5"></label></div>
    <div><label>Multiplicand 2: <input name="n" size="5"></label></div>
    <div><input type="submit" value="Multiply!"></div>
  </form>
</body>
```

# CGI program in C

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char *data;
    long m,n;
    printf("%s%c%c\n",
        "Content-Type:text/html;charset=iso-8859-1",13,10);
    printf("<TITLE>Multiplication results</TITLE>\n");
    printf("<H3>Multiplication results</H3>\n");
    data = getenv("QUERY_STRING");
    if(data == NULL)
        printf("<P>Error! Error in passing data from form to script.");
    else if(sscanf(data,"m=%ld&n=%ld",&m,&n)!=2)
        printf("<P>Error! Invalid data. Data must be numeric.");
    else
        printf("<P>The product of %ld and %ld is %ld.",m,n,m*n);
    return 0;
}
```

# The Action Attribute

## Example

```
<form action=".../mult.cgi">
```

- The **action** attribute defines the action to be performed when the form is submitted.
- In the example above, the form data is sent to a program on the server called “mult.cgi”. This program contains a server-side script that handles the form data.
- If the action attribute is omitted, the action is set to the current page.

# The Method Attribute

- The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data.

e.g. `<form action=“.../mult.cgi” method = “GET”>`

`<form action=“.../mult.cgi” method = “POST”>`

- If **GET** is used, the data received will be

`?m=20&n=3`

(data will be show on the url bar)

- **GET** is best suited for short, non-sensitive, amounts of data, because it has size limitations.

# The Method Attribute

- If **POST** is used, the data received will be  
 $m=20\&n=3$
- **POST** has no size limitations, and can be used to send large amounts of data.

- *Reference*

([https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Sending\\_and\\_retrieving\\_form\\_data](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Sending_and_retrieving_form_data))

# Reading input

- Using **POST**, data is passed to the script or program in the standard input stream (stdin), and the length (in bytes, i.e. characters) of the data is passed in an environment variable called `CONTENT_LENGTH`.
- When reading the input, the program must not try to read more than `CONTENT_LENGTH` characters.

(Refer to these website for more detail:

[https://www.w3schools.com/php/php\\_forms.asp](https://www.w3schools.com/php/php_forms.asp)

<https://www.ostraining.com/blog/coding/retrieve-html-form-data-with-php/>

)



# Reading input from POST

Name:

E-mail:

```
<!DOCTYPE HTML>
<html>
<body>
  <form action="welcome.php" method="post">
    Name: <input type="text" name="name"><br>
    E-mail: <input type="text" name="email"><br>
    <input type="submit">
  </form>
</body>
</html>
```

The data received will be ?

# Reading input from POST with PHP

```
// welcome.php
<?php
// retrieve the form data by using the element's name attributes value as
key
$name = $_POST["name"];
$email = $_POST["email"];
// display the results
echo '<h3>Your name is</h3>' . $name;
echo "<br>";
echo '<h3>Your email is</h3>' . $email;

?>
```

# Example

- How does this website look like?

```
<h2>PHP Form Validation Example</h2>
<form method="post" action="form.php">
  Name: <input type="text" name="name">
  <br><br>
  E-mail: <input type="text" name="email">
  <br><br>
  Website: <input type="text" name="website">
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>
```

# Example

## PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other

**Your Input:**

**The data received will be ?**

**Write a php file to read the user input?**

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ( isset( $_POST['submit'] ) ) {
    $name = $_POST["name"];
    $email = $_POST["email"];
    $website = $_POST["website"];
    $comment = $_POST["comment"];
    $gender = $_POST["gender"];
}

?>
```