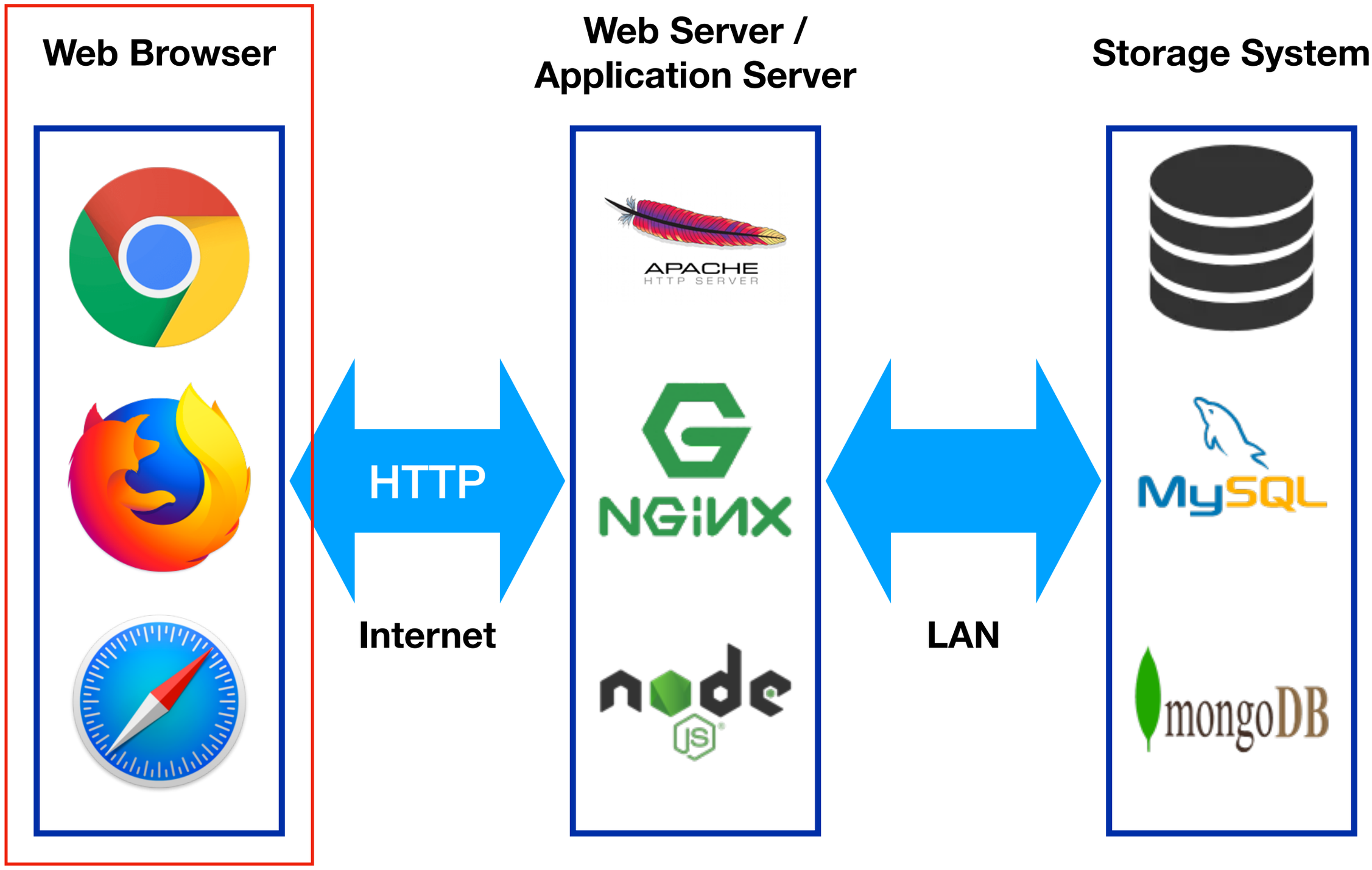
Introduction to Web Applications

CSCI 4140: Open-Source Software Project Development

Prof. Hong Xu

http://course.cse.cuhk.edu.hk/~csci4140/

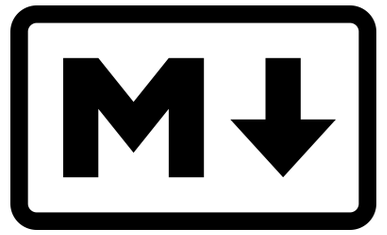
Web Application Architecture



Hypertext Markup Language (HTML)

- Describes the structure of Web pages using markup
- Markup Language
  - Include directives with content
  - Directives can dictate presentation and describe content, e.g., <title>Title text</title>
- Web pages are built from HTML elements
- HTML elements are represented by tags
  - HTML uses <tag>…</tag> to denote tags

Markup vs Markdown



- Markup is a generic term for any language that describes a document's formatting

- Mardown (md) is a lightweight type of markup language that generates HTML markup

- Very simple syntax, easy to learn, widely used (github)

Markdown	HTML	Rendered Output
`I just love **bold text**.`	`I just love <strong>bold text</strong>.`	I just love **bold text**.
`I just love __bold text__.`	`I just love <strong>bold text</strong>.`	I just love **bold text**.
`Love**is**bold`	`Love<strong>is</strong>bold`	Love**is**bold

https://www.markdownguide.org/basic-syntax/

HTML Tags

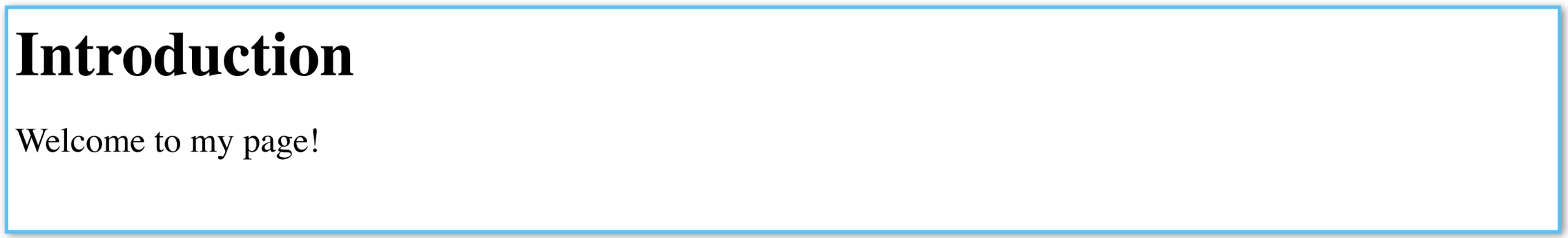
- Web browsers use HTML tags to render a web page
- Tags can provide
  - Formatting information, e.g., <i> for italic
  - Semantics about text, e.g., <title>, <p>
  - Additional description about the content, e.g., <img>
  - Structural information, e.g., <ul> and

Example of HTML Document

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>Introduction</h1>
<p>Welcome to my page!</p>

</body>
</html>
```

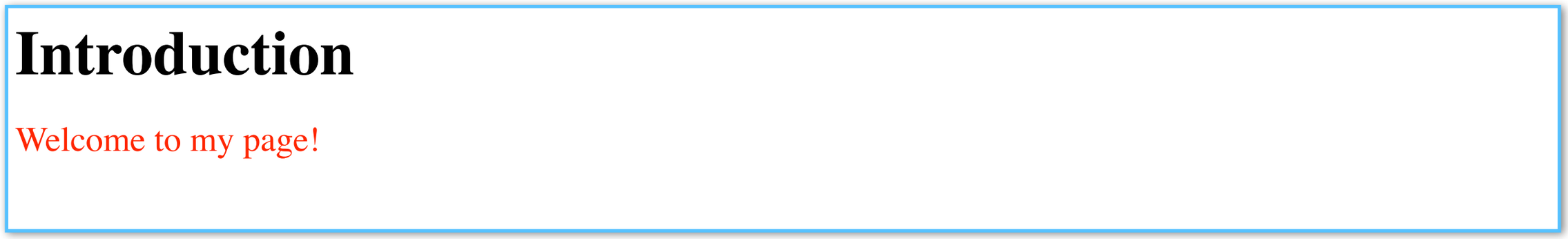


Attributes of HTML Tags

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>Introduction</h1>
<p id="p1" style="color: red">Welcome to my page!</p>

</body>
</html>
```



Cascading Style Sheet (CSS)

- Describes the style of an HTML document
- Describes how HTML elements should be displayed
- Driving problem behind CSS
  - “What font and size does Text generate?”
    - Answer: The browser has default values
  - “How to use something different from the defaults?”
    - Answer: Define inline style, e.g., <p style=“font-size: 16px”>Text
- Style sheets were added to
  - Specify styles of HTML elements rather than browser default
  - Avoid writing inline style for every HTML element

Using CSS

- Separate style from content
  - Cleaner HTML document
  - Easier to make changes to both the content and style
- Allow reuse of existing style on
  - Multiple HTML elements
  - Multiple HTML documents

Example of CSS

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>Introduction</h1>
<p>Welcome to my page!</p>

</body>
</html>
```

```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 24px;
}
```

Selector

Declaration Block

Property **Value**

HTML **CSS**

Introduction

Welcome to my page!

CSS Selectors: .class And #id

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1 class="white-
center">Introduction</h1>
<p id="p1">Welcome to my page!</
p>
<p class="white-center">This
paragraph is centered.</p>

</body>
</html>
```

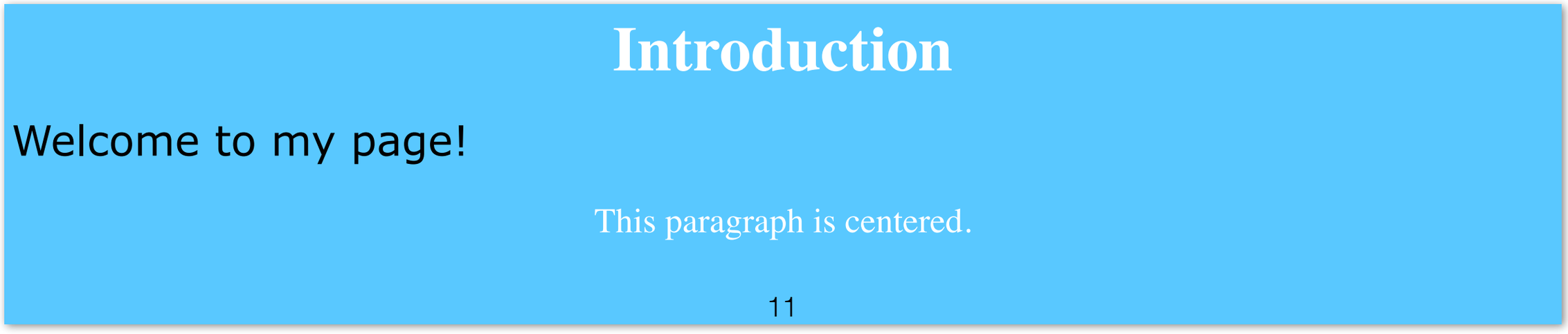
HTML

```
body {
    background-color: lightblue;
}

.white-center {
    color: white;
    text-align: center;
}

#p1 {
    font-family: verdana;
    font-size: 24px;
}
```

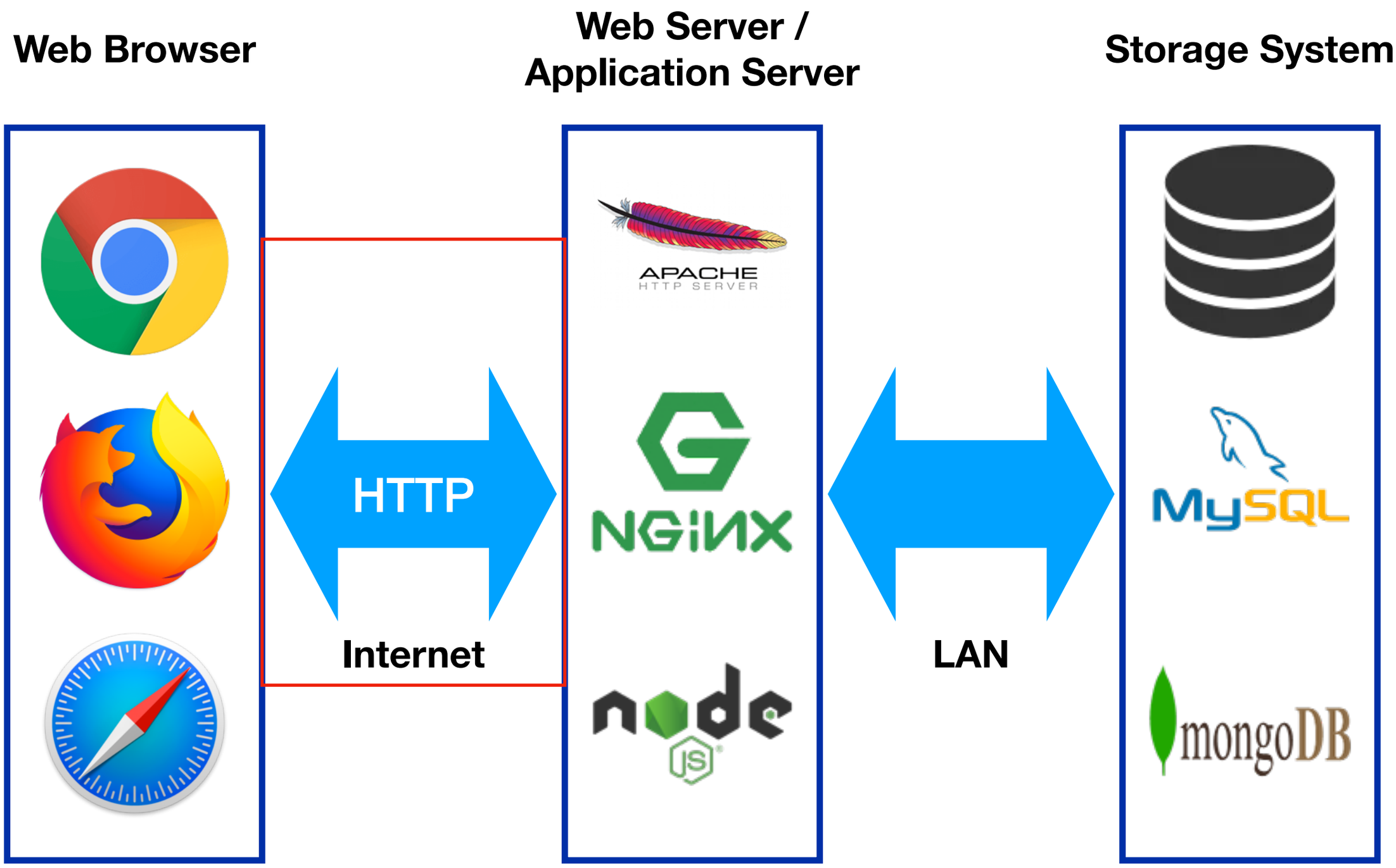
CSS



JavaScript

- The programming language of HTML and the Web
- We will discuss it in later lectures

Web Application Architecture



Hypertext Transport Protocol (HTTP)

http://www.example.com/index.html

- The browser fetches the file index.html from the server using the HTTP protocol

- HTTP - A simple request-response protocol on top of TCP/IP

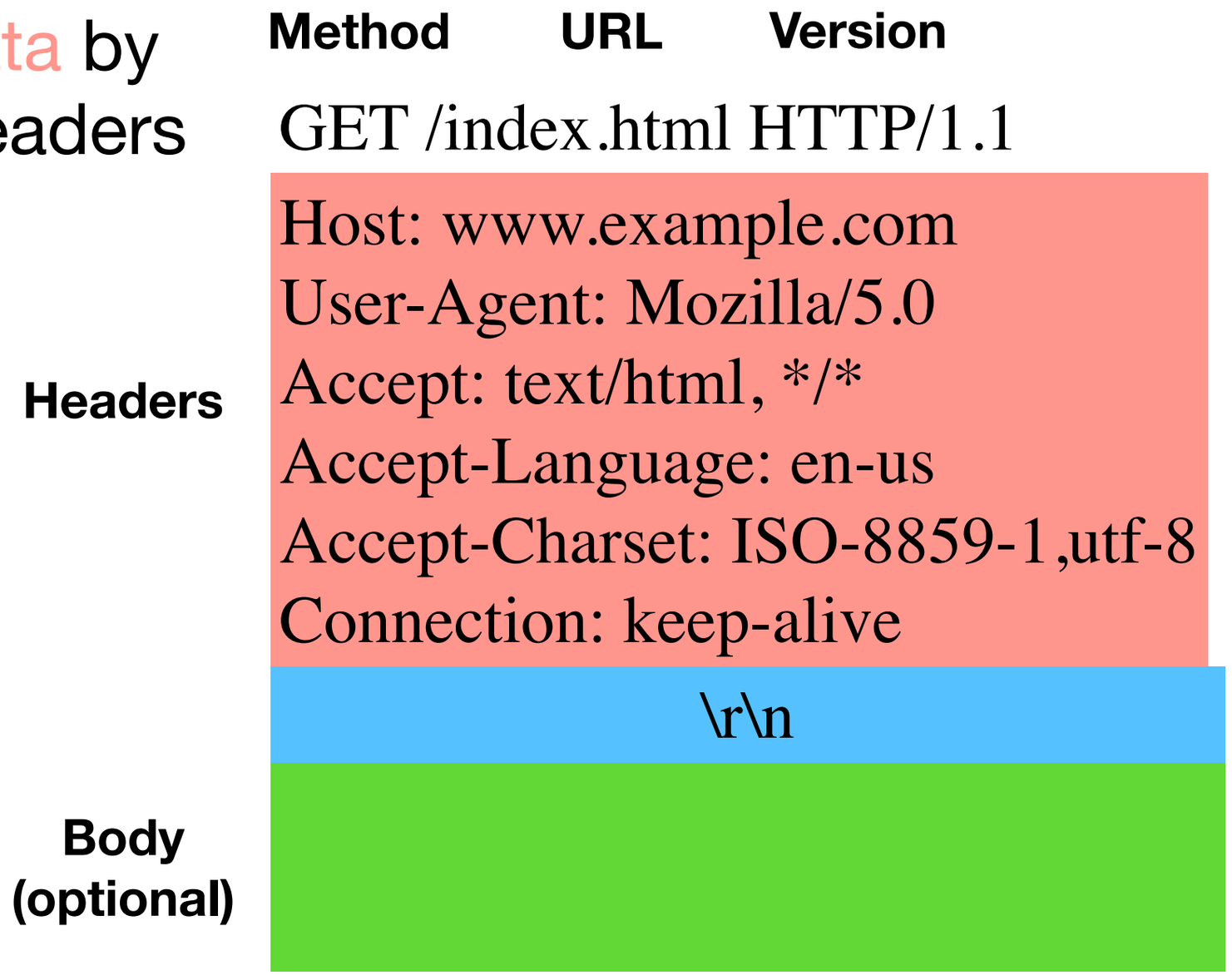
1. Establish a TCP/IP connection with www.example.com

2. Send a HTTP GET request along the connection

3. Read the response from the server through the connection

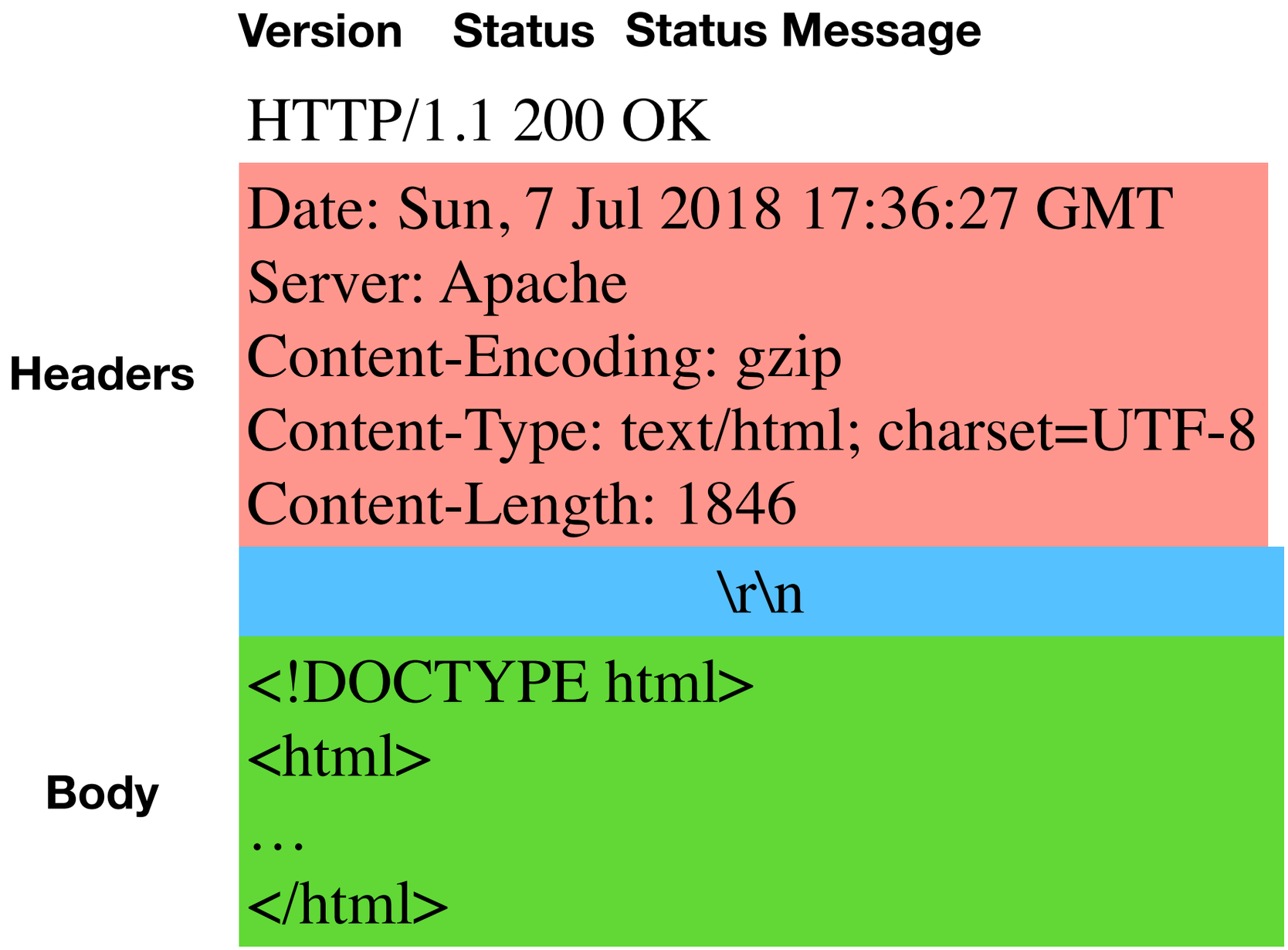
HTTP Request

- Writes data into socket
- Embeds input data by the user in the headers and the body



HTTP Response

- Reads data from socket



Common HTTP Response Status Codes

Code	Status	Description
200	OK	Success
301	Moved Permanently	Permanent redirection
307	Temporary Redirect	Try URI in the Location field this time
400	Bad Request	Malformed request
401	Unauthorized	Require user authentication
403	Forbidden	Refuse to fulfill the request
404	Not Found	
500	Internal Server Error	Something unexpected (Error) happened
501	Not Implemented	Does not support the requested functionality
503	Service Unavailable	Unable to handle the request due to overloading or maintenance

HTTP Methods

- GET - Reads data from a specified resource
- HEAD - Fetches only the information (HTTP header) about data at a specified resource
- POST - Submits data to be processed to a specified resource and get a response back
- PUT - Uploads data to a specified resource
- DELETE - Deletes the specified resource

GET and **POST** are commonly used

HTTP Is Stateless

- The HTTP server is not required to retain information or status about each user for the duration of multiple requests.
- A web application can track state over multiple requests if it wants to.
  - HTTP cookies
  - Server side sessions
  - Hidden variables within web forms

Web Application Architecture

Web Browser

Web Server / Application Server

Storage System

HTTP

Internet

LAN

Web Servers

- Both Web Browsers and Web Servers speak HTTP
- Web Servers: get HTTP requests and send HTTP responses
- A web server loops forever doing
  - Accept TCP connection from a client (browser)
  - Read HTTP request from the connection
  - Process the request
  - Write HTTP response to the connection
  - Close the TCP connection

Serving Static Documents

- Process “GET /index.html HTTP/1.1”

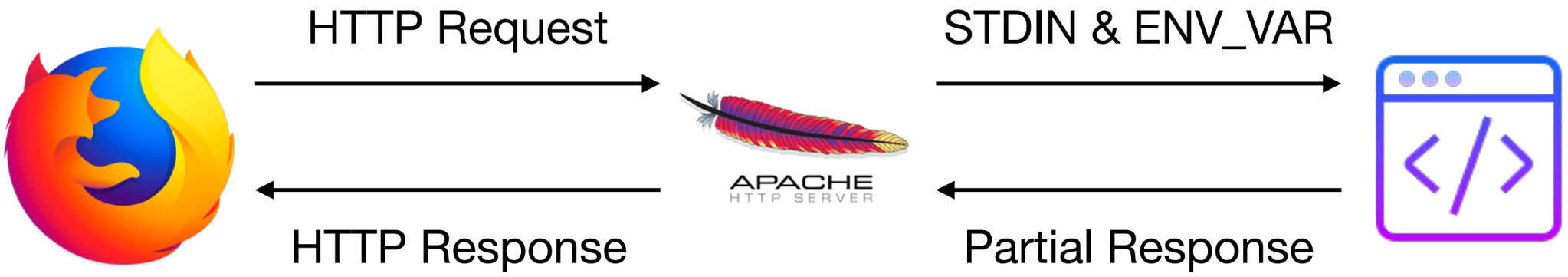
```
int fd = open(“index.html”);
int len = read(fd, buffer, sizeOfFile(fd));
write(tcpConnection, httpResponseHeader, headerSize);
write(tcpConnection, buffer, len);
close(fd);
```

Generating Dynamic Content

- Common Gateway Interface (CGI)
  - A protocol for web servers to execute programs like calling Command-line Interface (CLI) programs (in another process) to generate web pages dynamically
- Process “GET /index.cgi HTTP/1.1”

runProgram(index.cgi, tcpConnection, env, …);

How CGI Programs Work?




```perl
#!/usr/bin/perl

=head1 DESCRIPTION

env - a CGI program that just prints its environment

=cut
print "Content-type: text/plain\n\n";

for my $var ( sort keys %ENV ) {
 printf "%s = \"%s\"\n", $var, $ENV{$var};
}
```

https://csci4140.cse.cuhk.edu.hk/cgi-bin/env.pl

Partial Response Generated by CGI Program

Version **Status** **Status Message**

HTTP/1.1 200 OK

Headers

Date: Sun, 7 Jul 2018 17:36:27 GMT
Server: Apache
Content-Encoding: gzip

Content-type: text/plain

\r\n

Body

CONTENT_LENGTH = ""
CONTENT_TYPE = ""
DAEMON_OPTS = "-f"
DOCUMENT_ROOT = "/var/www/html"
…

Server Output

CGI Program Output

CGI Program should generate

- The content-type
- The body

Discussion

How can you return a *zip file* so that the browser knows that it is a zip file?

Dynamic Content

- With CGI programs, we can fulfill many tasks
  - Form submission
  - Search
  - …
- How to generate customized output?
  - Generate content based on user input
  - Generate content based on application state

User Input - GET Request

```
<form method="GET" action="form.cgi">
   Username:
   <input type="text" name="user"></input><br>
   Language:
   <input type="text" name="lang"></input><br>
   <input type="submit"></input>
</form>
```

Username: alex
Language: enUS
Submit

http://www.example.com/form.cgi?user=alex&lang=enUS **Query String**

Method URL Version

GET /form.cgi?**user=alex&lang=enUS** HTTP/1.1

Headers

Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive

\r\n

Body

User Input - POST Request

```
<form method="POST" action="form.cgi">
   Username:
   <input type="text" name="user"></input><br>
   Language:
   <input type="text" name="lang"></input><br>
   <input type="submit"></input>
</form>
```

Username: alex
Language: enUS
Submit

http://www.example.com/form.cgi

	Method URL Version
	POST /form.cgi HTTP/1.1
Headers	Host: www.example.com User-Agent: Mozilla/5.0 Accept: text/html, */* Accept-Language: en-us **Content-Type**: application/x-www-form-urlencoded **Content-length**: 19
	\r\n
Body	**user=alex&lang=enUS**

GET Request vs. POST Request

	GET	POST
Visibility	Data is visible to everyone in URL	Data is not visible
History	Parameters remain in browser history	Parameters are not saved
Data length	Limited by URL length (maximum: 2048 characters)	No restriction
Data type	Only ASCII characters	No restriction Binary data is allowed
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded multipart/form-data
Security	Data is visible in the URL Never send sensitive data using GET!	Safer compared to GET as parameters are not stored in browser history or web server log

Problems of CGI

- High performance overhead
  - Invocation of a newly created process to call a command (CGI program) per request
  - The time and resource to create new process can be much higher than the actual work of generating the output
  - The overhead can be even higher if the CGI program needs to be interpreted

Reducing Overhead

- FastCGI
  - Pre-fork multiple persistent processes to handle requests
- Run application code within the web server process
  - Extensions modules, e.g., mod_php
- Use precompiled CGI programs
  - Write C/C++ programs
  - Avoid using interpreted languages such as Perl or PHP