ITEC 414
Network Programming

Dr. N. B. Gyan

Central University, Miotso. Ghana

Client/Server Model

Client Server Defined

- The **client-server** programming model is a distributed computing architecture that segregates information users (clients) from information providers (servers).
- A **client** is an application that needs something like a web page or IP address from a server.
- Clients may contact a server for this information at any time. Clients are information users.

Client Server Defined

- A **server** is an application that provides information or resources to clients.
- It needs to be always up and running, waiting for requests from clients.
- Client applications communicate only with server applications and vice versa. Clients do not communicate directly with other clients.

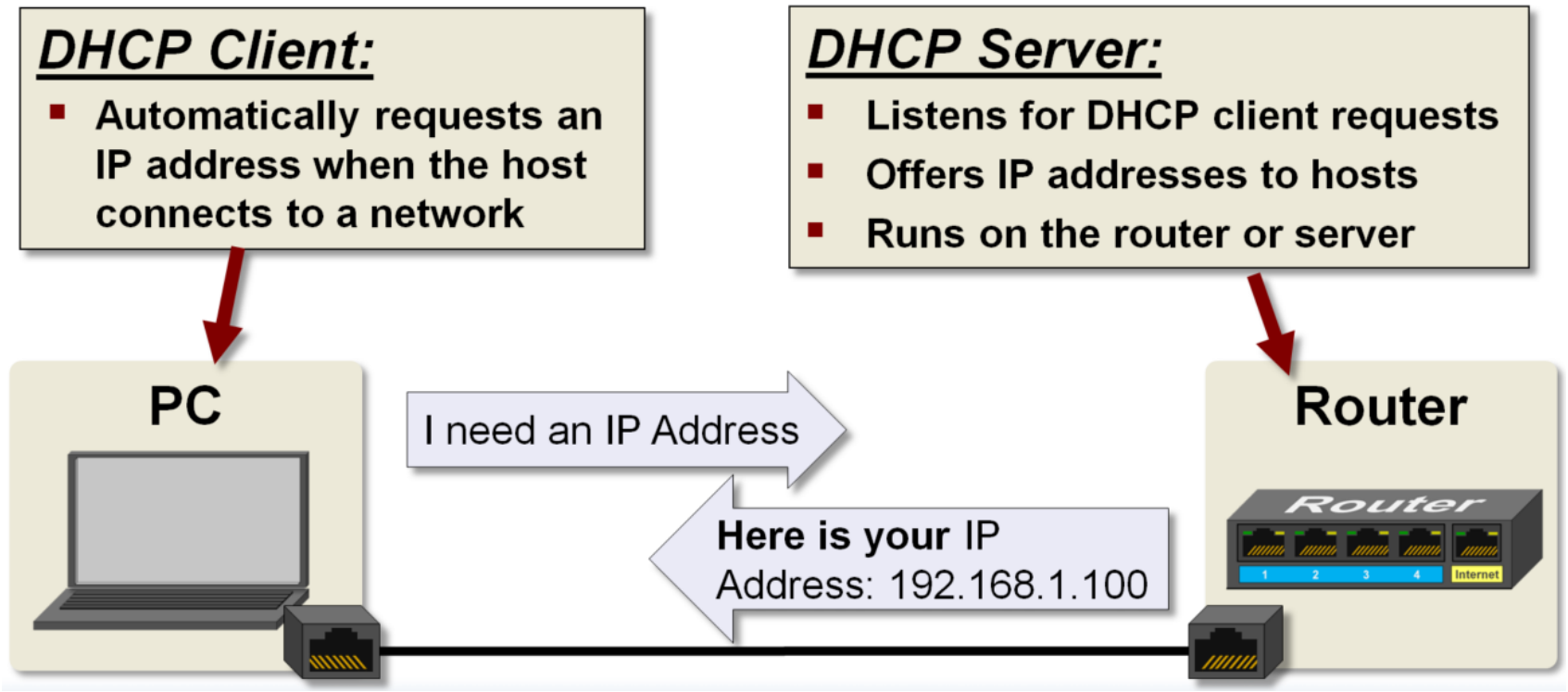
Example: DHCP Client Server

- Here is a very common example of the client-server programming model.
- The **dynamic host configuration protocol** (DHCP) is the application responsible for requesting and offering IP addresses.
- A DHCP client automatically requests an IP address from a DHCP server when a network is detected.
- A DHCP client could request a new IP address at any time, so the DHCP server must always be active and ready to respond to client requests.

Example: DHCP Client Server

- The DHCP server application typically exists in a router, but may also be found running on a network server for larger networks.

Example: DHCP Client Server



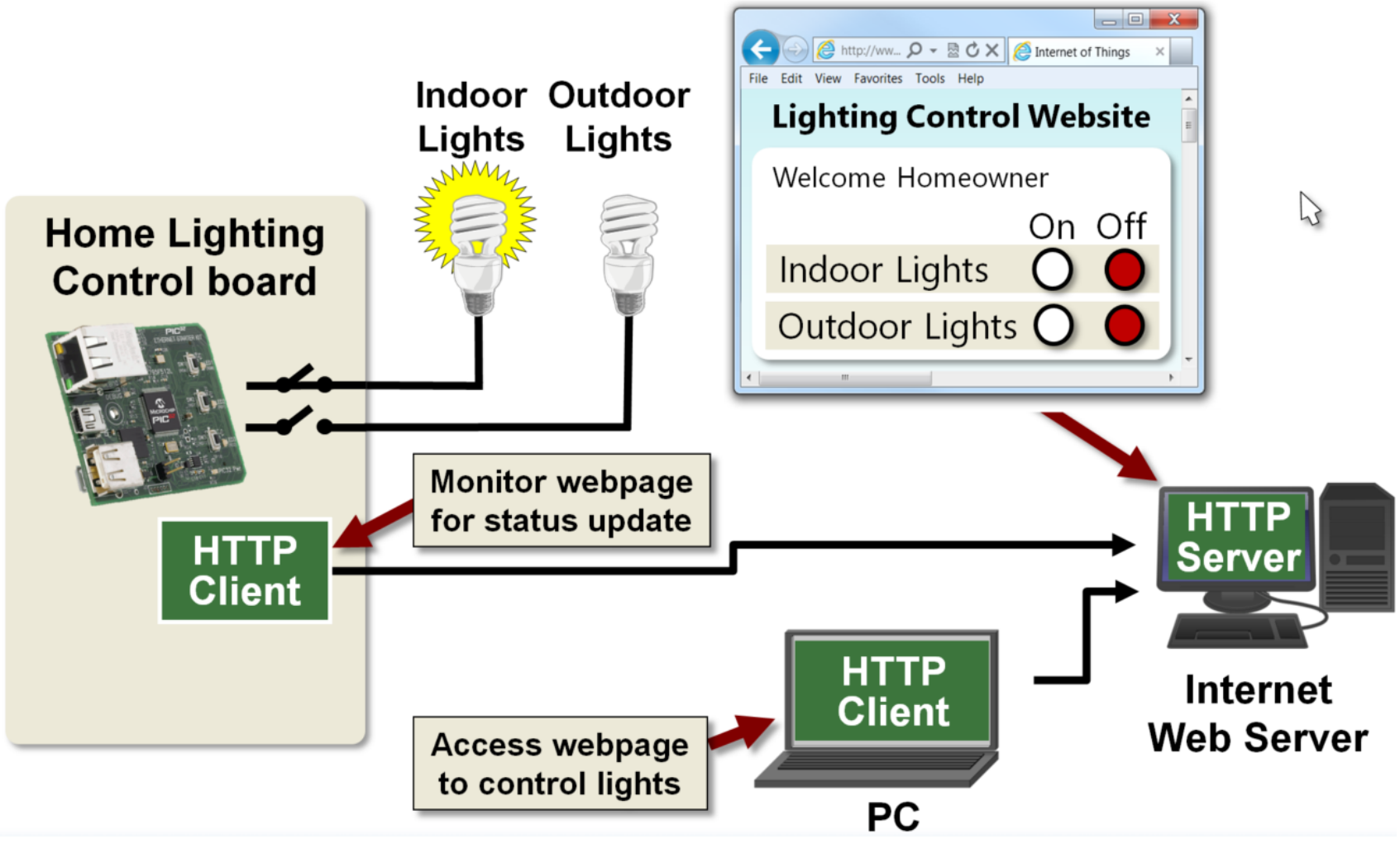
Example: HTTP Client Server

- An HTTP client running on a PC to control the lights at home.
- The following example shows an HTTP client running on a home lighting control board, which has been configured to monitor a lighting control website running on an Internet web server to determine if lights should be on or off.

Example: HTTP Client Server

- I browse to the same lighting control webpage being monitored by the lighting control board, enter my username and password, and now have the ability to change the webpage.
- The next time the control board checks this webpage it will see the change and control the lights appropriately.

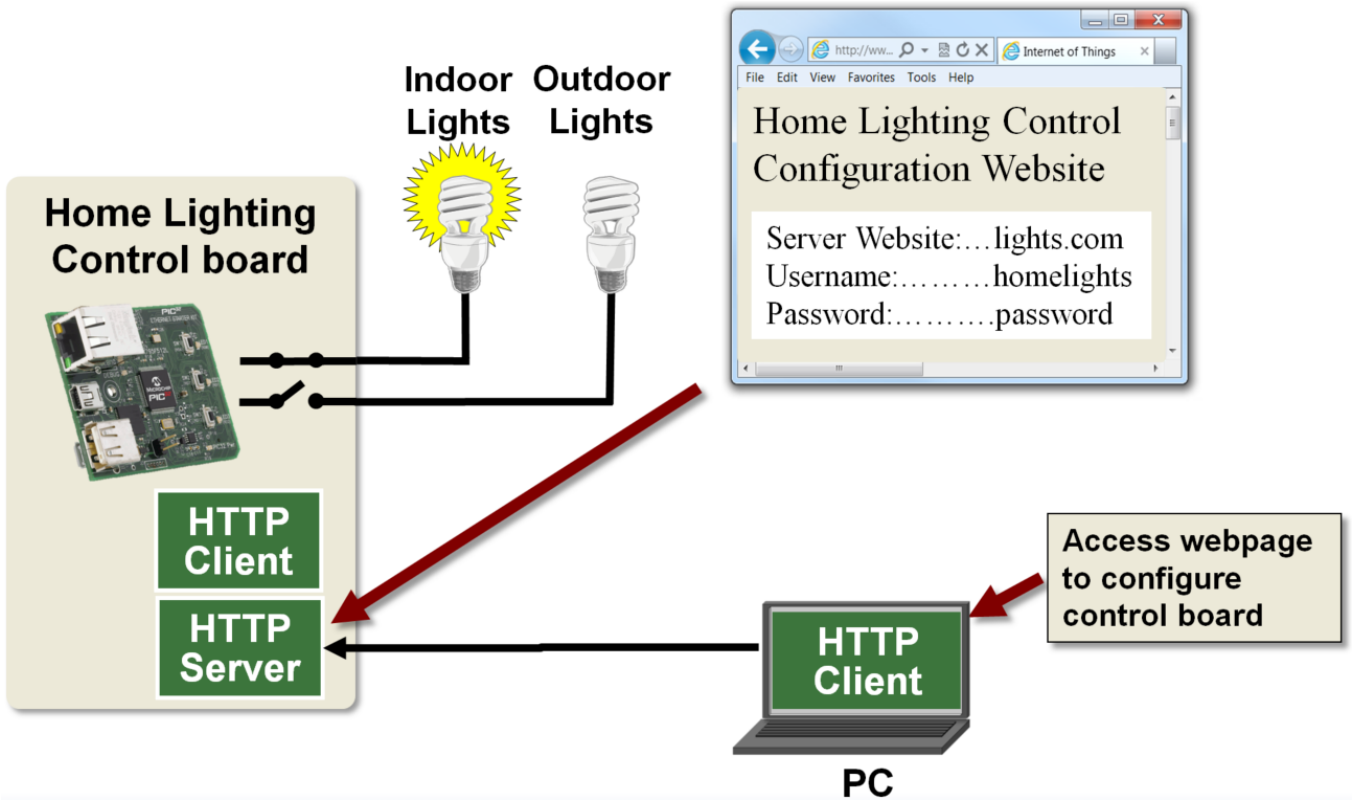
Example: HTTP Client Server



Example: HTTP Client and Server in the Same Local Host

- A network host is usually either a client or a server, but it is possible for a host to be both. Let's see an example of this.
- My control board may also have an HTTP Server running concurrently with the client.
- This could be used to serve a simple setup and configuration web page, which would allow me to change the website and log-in information the HTTP client uses to check for lighting control updates.

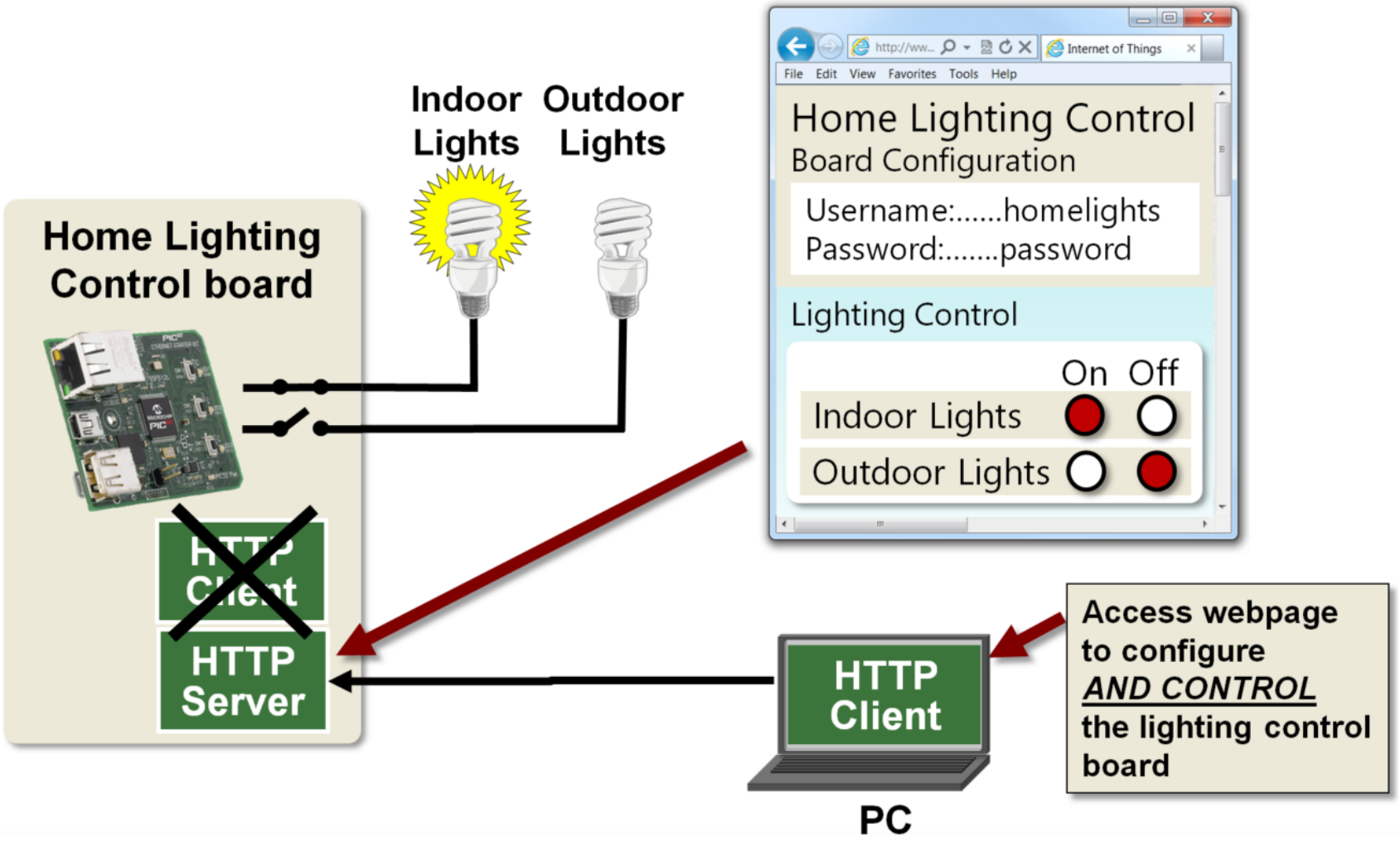
Example: HTTP Client and Server in the Same Local Host



Example: Local Network HTTP Server

- If you do have an HTTP server running on your embedded device, it could also be used to actually control the device.
- This would allow you to eliminate the HTTP client application and Internet web server. At first, this may appear to be the best solution, but looks can be deceiving.
- This is probably the easiest solution if the HTTP client is running on the same local network as the lighting control board. Unfortunately, this is not very common.

Example: Local Network HTTP Server



Example: Local Network HTTP Server

- The ability to control the lights or anything else from a remote location over the internet is a more likely and useful scenario.
- Accessing a web server on a local network from the Internet can be done, but it’s not a trivial task.
- Deciding where to locate a web server must be carefully considered.

Internet Server vs. Local Network Server

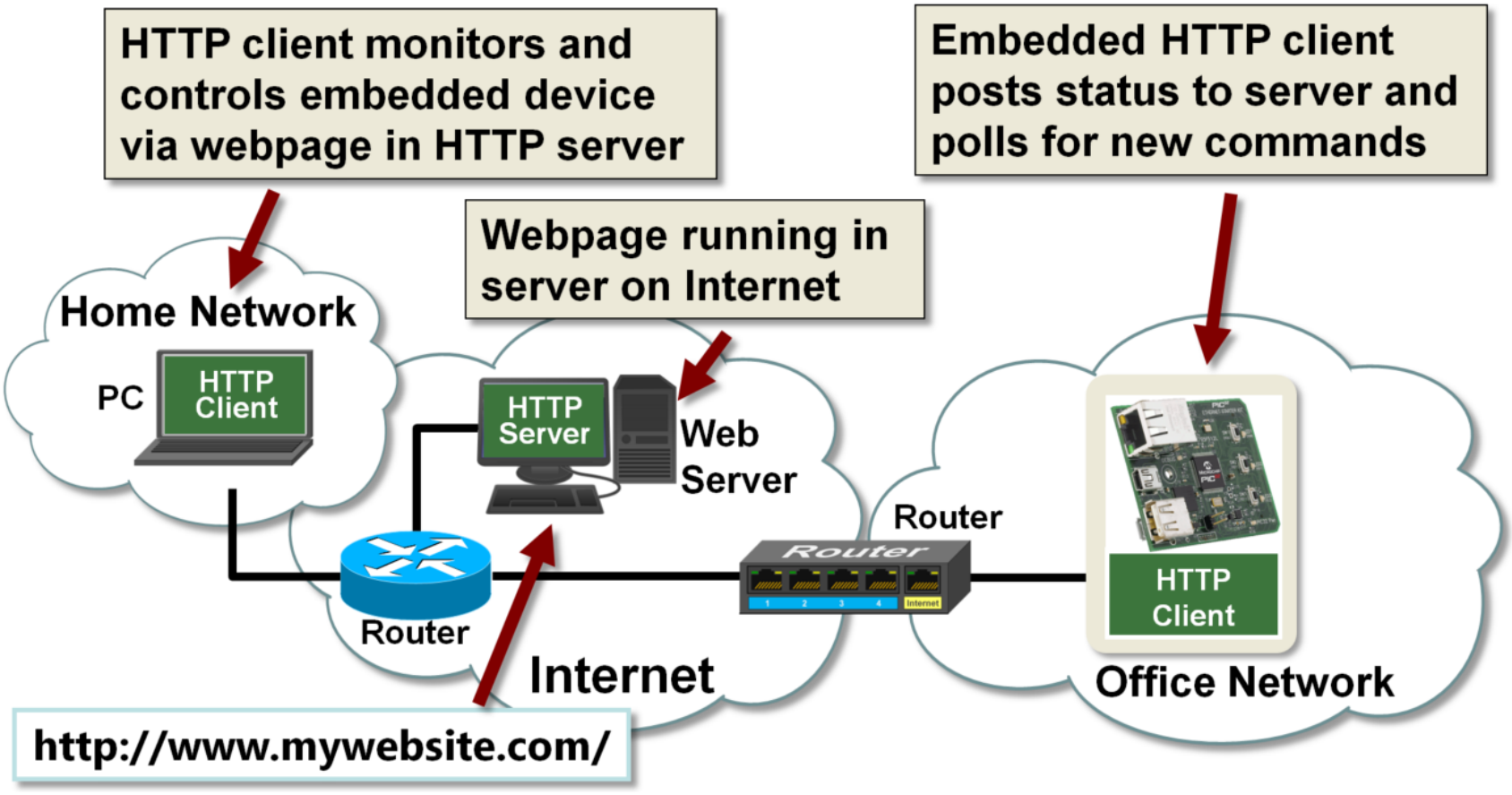
Internet Server

- Accessing an HTTP server on the Internet is effortless.
- You may need to enter a username or password for some sites, but it really couldn’t be easier.
- In this example, the webpage that controls and displays the current state of the board runs on an HTTP server on the Internet.
- The embedded HTTP client posts its current status to the server and polls for new commands.

Internet Server

- A web browser on a PC or smartphone can monitor and control the embedded device via the webpage in the HTTP server.
- The server may be implemented on a shared web hosting service from a company like godaddy.com or networksolutions.com, or may be implemented in the cloud, which is a service of decentralized servers from companies like Amazon, Google, or ioBridge.
- These service providers will allow you to choose a website name, so your webpage can be easily accessed.

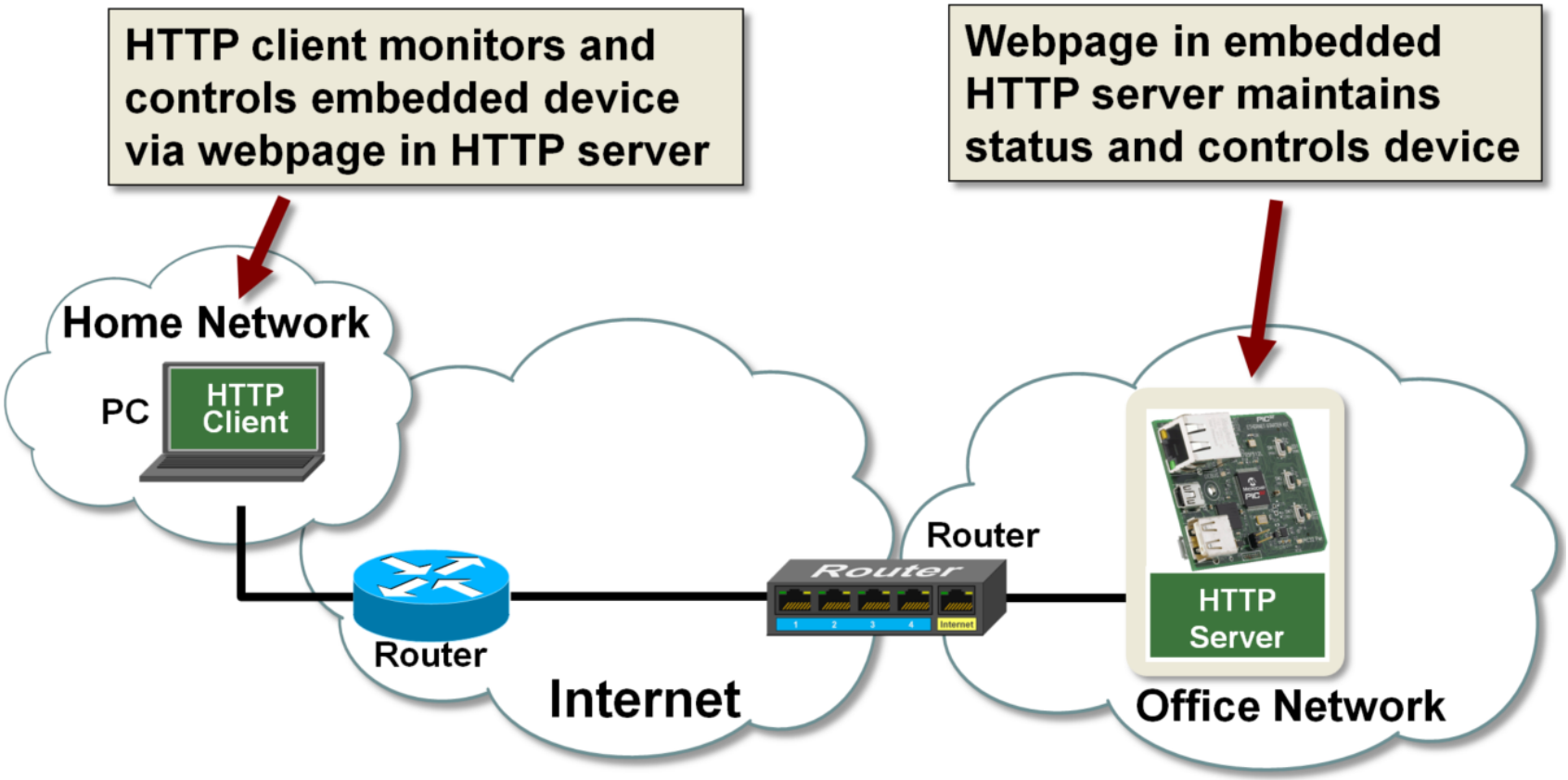
Example: Local Network HTTP Server



Local Network Server

- Instead of locating our HTTP server on the Internet, we could instead locate it on a local network.
- In this case, the webpage that controls and monitors the embedded device is actually running on the embedded device.
- Just as in the previous case, a web browser on a PC or smartphone can monitor and control the embedded device via the webpage running in the embedded HTTP server.

Example: Local Network HTTP Server

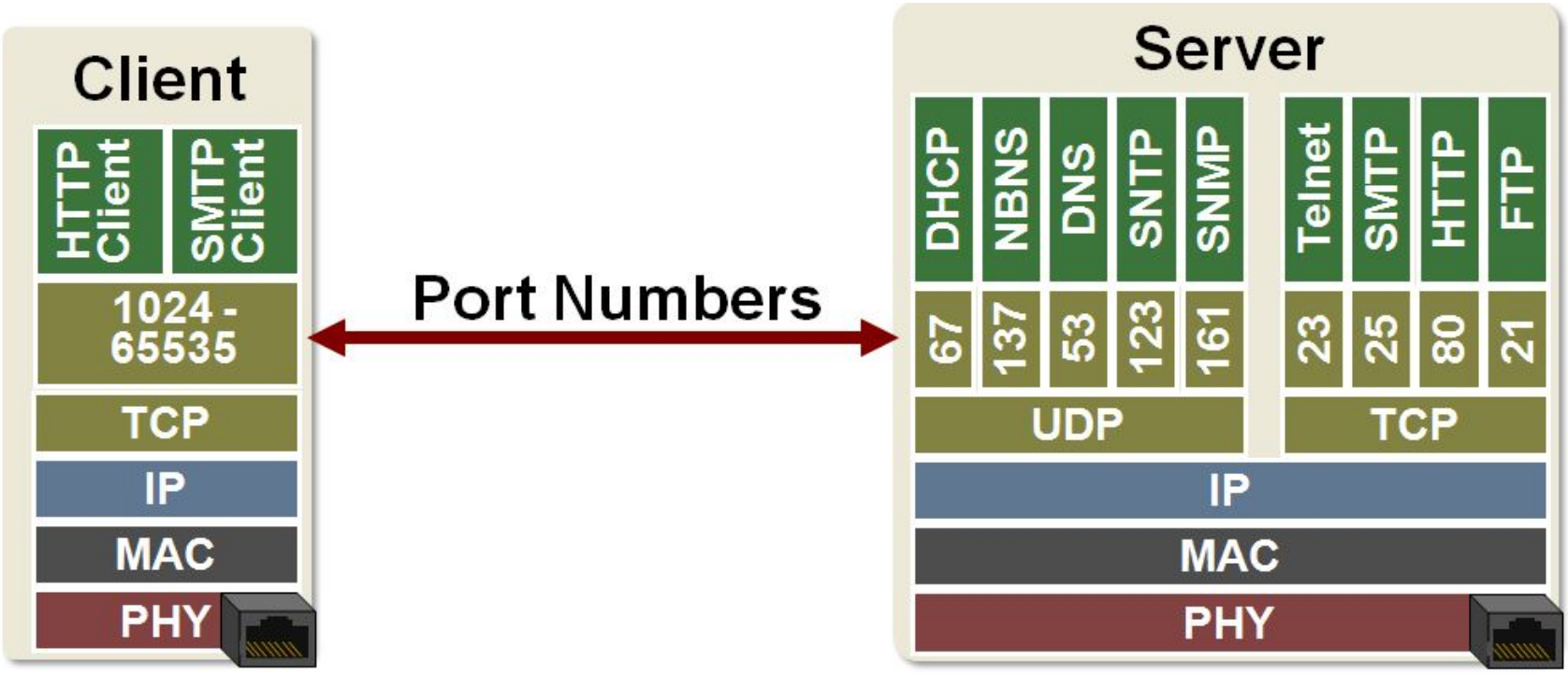


Sockets & Ports

Sockets & Ports

- **Ports** are used to identify processes running in the applications on a host.
- Let's assume we have two applications running on one PC that require TCP/IP communications. Assume one is a web browser and the other is an email client.
- Both applications send and receive packets with the same IP address, so how does the Transport layer differentiate a web browser packet from an email packet?
- The answer is port numbers.

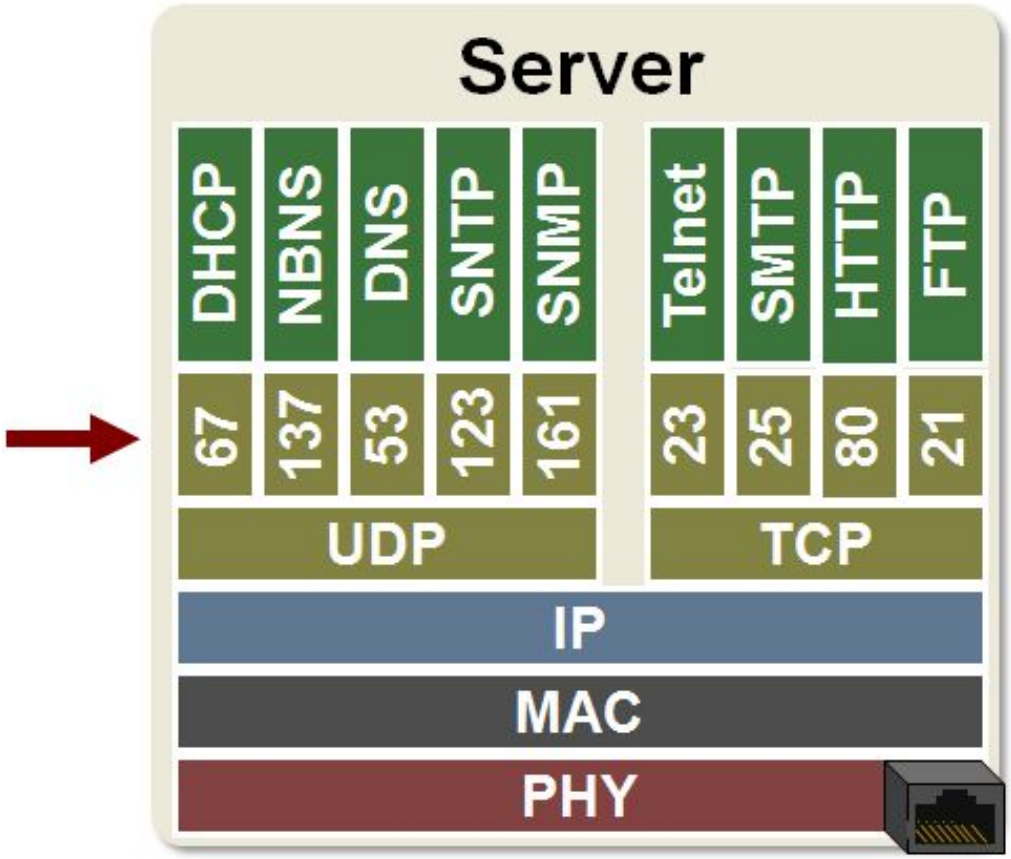
TCP/IP Ports


Client
HTTP Client
SMTP Client
1024 - 65535
TCP
IP
MAC
PHY
Port Numbers
Server
DHCP
NBNS
DNS
SNTP
SNMP
Telnet
SMTP
HTTP
FTP
67
137
53
123
161
23
25
80
21
UDP
TCP
IP
MAC
PHY

TCP/IP “Well-Known” Ports

- “Well-Known” ports are port numbers that have been reserved for common applications, typically **server** applications.
- The port numbers assigned to these server applications have to be known by the client’s Transport layer, so they can add the correct destination port number to messages.
- Clients know that servers will be listening for their requests at these reserved port numbers.

“Well-known” TCP/IP Ports

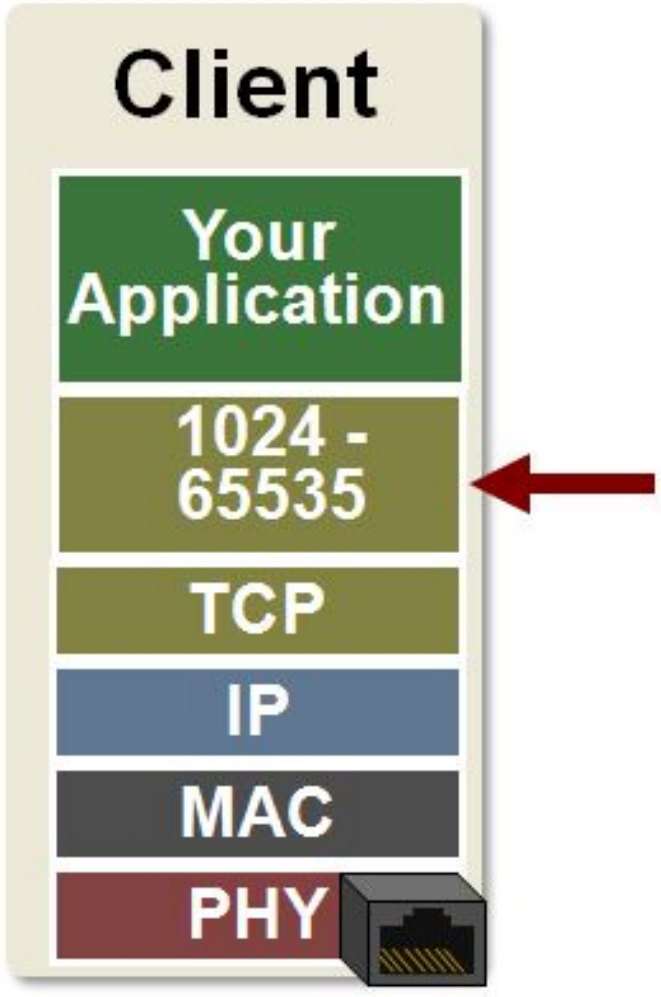


The well known port numbers are assigned by IANA which is the **Internet Assigned Numbers Authority**. IANA is the same group that manages the DNS Root and IP addresses.

Client-side TCP/IP Ports

- **Client** side port numbers are generated and assigned by the Transport layer.
- They could be any number from 1024 to 65535. These port numbers are typically allocated for short term use and are referred to as “Ephemeral or Dynamic Ports”.

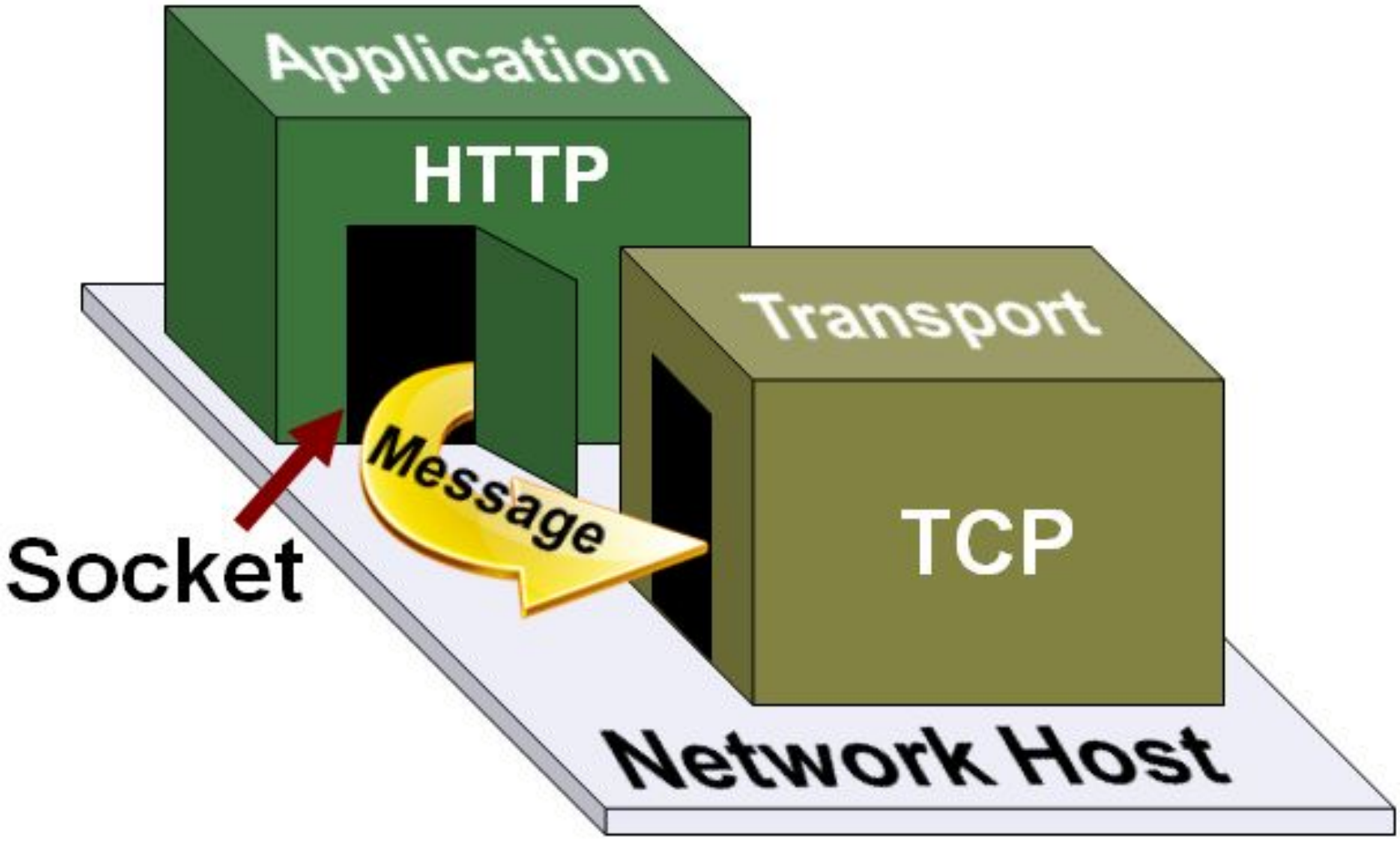
Client-side TCP/IP Ports



Sockets

- A **socket** is a software concept for a connection.
- Sockets enable applications to connect to a Transmission Control Protocol/Internet Protocol (TCP/IP) network.
- An application running on a host creates a socket or doorway to connect with an application on another host.
- Messages pass through this socket or doorway.

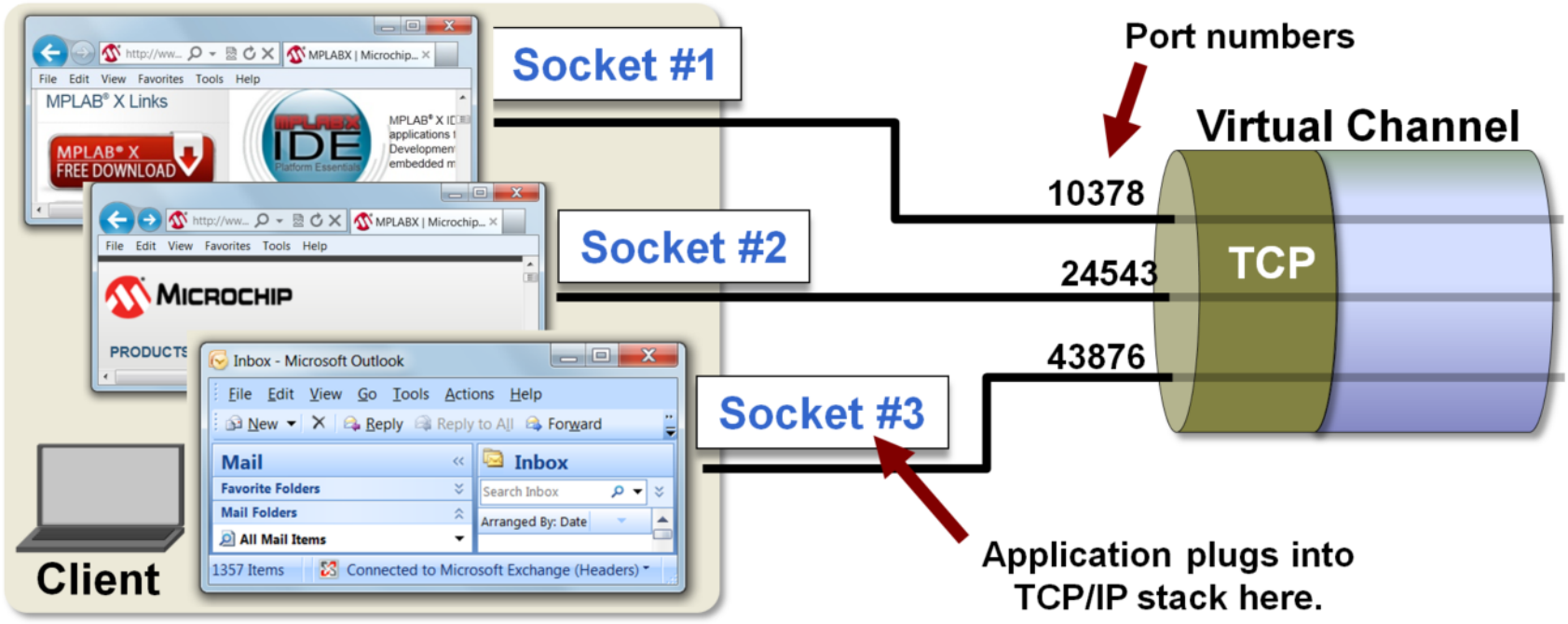
Sockets


Application
HTTP
Transport
Message
TCP
Socket
Network Host

Sockets Enable Virtual TCP or UDP Connections Between Hosts

- Sockets enable virtual TCP or UDP communication channels between hosts.
- When an application starts on a host, a port number is assigned to a process or a function running in it.
- When that application wants to communicate with another host, (go to a website for example) a socket is created.

Sockets Enable Virtual TCP or UDP Connections Between Hosts



This example shows three applications requiring three TCP communication channels: Two channels for each of the two web browsers acting as HTTP clients, and one for the email application acting as an SMTP client.

Example: Use Sockets to Create a TCP Connection

The following steps describe a TCP connection process using sockets.

1. Server Creates Socket and Listens
2. Client Creates a Socket and Connects
3. Transport Layer Delivers Message to Server
4. Server Creates Socket & Process
5. Transport Layer Delivers Message to Client
6. Sockets Closed

1. Server Creates Socket and Listens

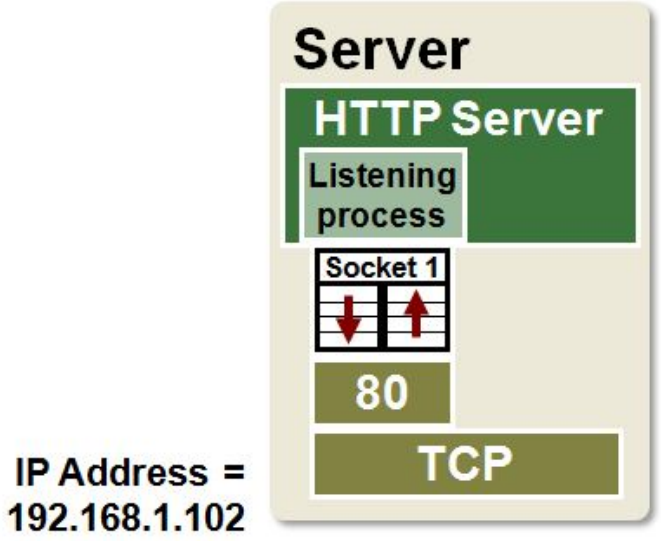
- A web server creates a socket dedicated to listening for client requests.
- After the socket exists, the server goes into “listening” mode and waits for a client’s request. It periodically checks for messages received in this socket.

1. Server Creates Socket and Listens

- This type of socket is referred to as a *connectionless* socket.
- A connectionless socket is used to establish a TCP connection with the HTTP server. There is no destination IP address or port number defined for this type of socket.

1. Server Creates Socket and Listens

Server Sockets	Socket 1	
Transport	TCP	
Source Port	80	
Source IP Addr	192.168.1.102	
Destination Port	n/a	
Destination IP Addr	n/a	

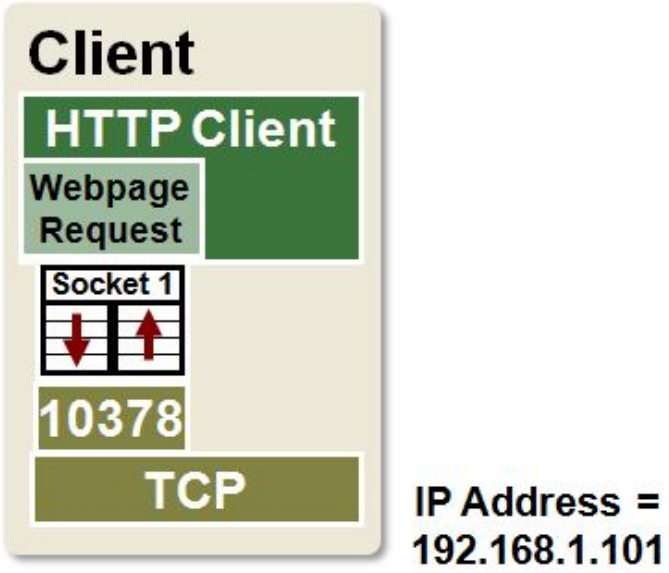


2. Client Creates a Socket and Connects

When a client wants to download a web page it creates a socket then sends the web page download request to the socket.

2. Client Creates a Socket and Connects

Client Sockets	Socket 1
Transport	TCP
Source Port	10378
Source IP Addr	192.168.1.101
Destination Port	80
Destination IP Addr	192.168.1.102



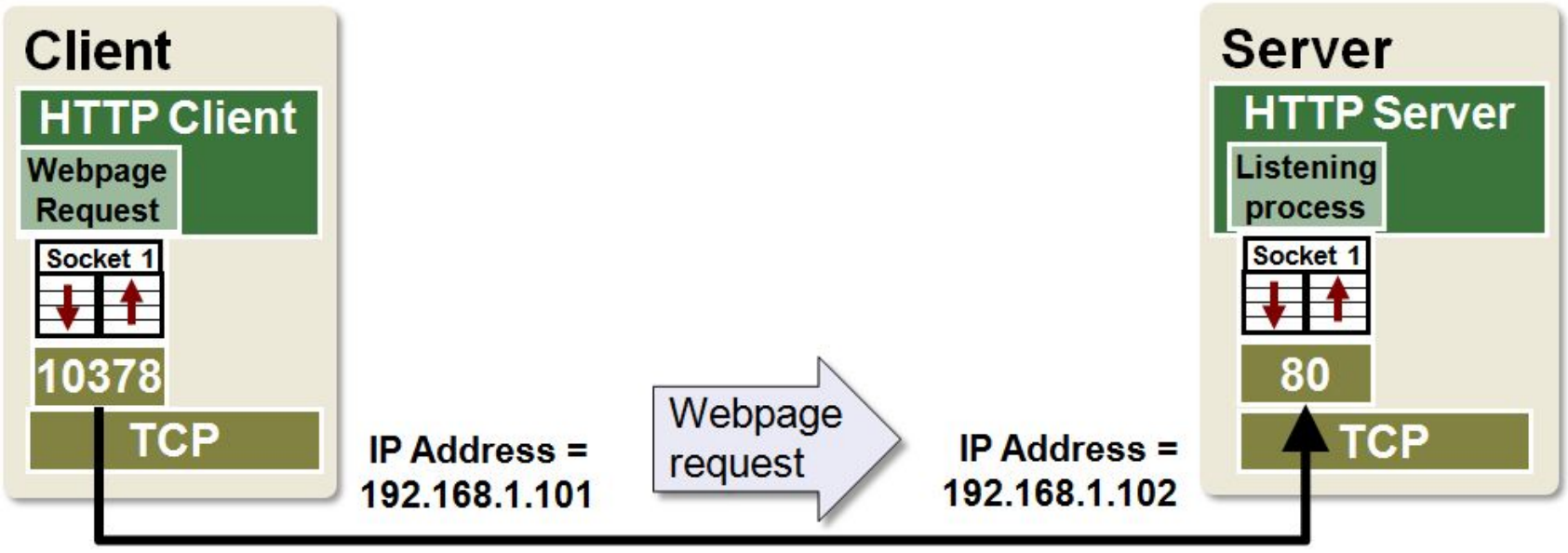
3. Transport Layer Delivers Message to Server

- The client's Transport layer periodically checks its transmit buffers to determine if a message needs to be sent.
- When a message is found it is forwarded to the destination address.

3. Transport Layer Delivers Message to Server

Client Sockets	Socket 1
Transport	TCP
Source Port	10378
Source IP Addr	192.168.1.101
Destination Port	80
Destination IP Addr	192.168.1.102

Server Sockets	Socket 1	
Transport	TCP	
Source Port	80	
Source IP Addr	192.168.1.102	
Destination Port	n/a	
Destination IP Addr	n/a	



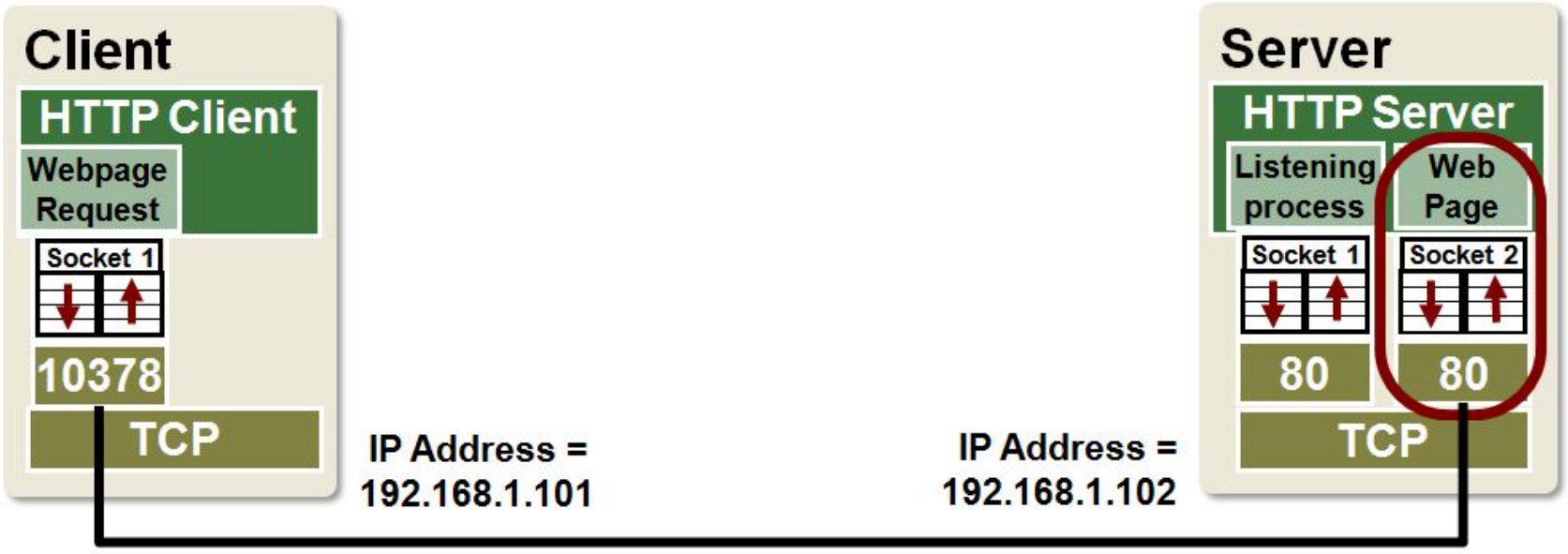
4. Server Creates Socket & Process

- When the server receives the client's request, it creates a new dedicated socket and process.
- It then creates a message for the client and sends it to the socket.
- Note this socket uses the client's destination IP address and port number.
- This virtual TCP connection is now referred to as "established".

4. Server Creates Socket & Process

Client Sockets	Socket 1
Transport	TCP
Source Port	10378
Source IP Addr	192.168.1.101
Destination Port	80
Destination IP Addr	192.168.1.102

Server Sockets	Socket 1	Socket 2
Transport	TCP	TCP
Source Port	80	80
Source IP Addr	192.168.1.102	192.168.1.102
Destination Port	n/a	10378
Destination IP Addr	n/a	192.168.1.101



The message sent by the server is the HTML file for the requested webpage.

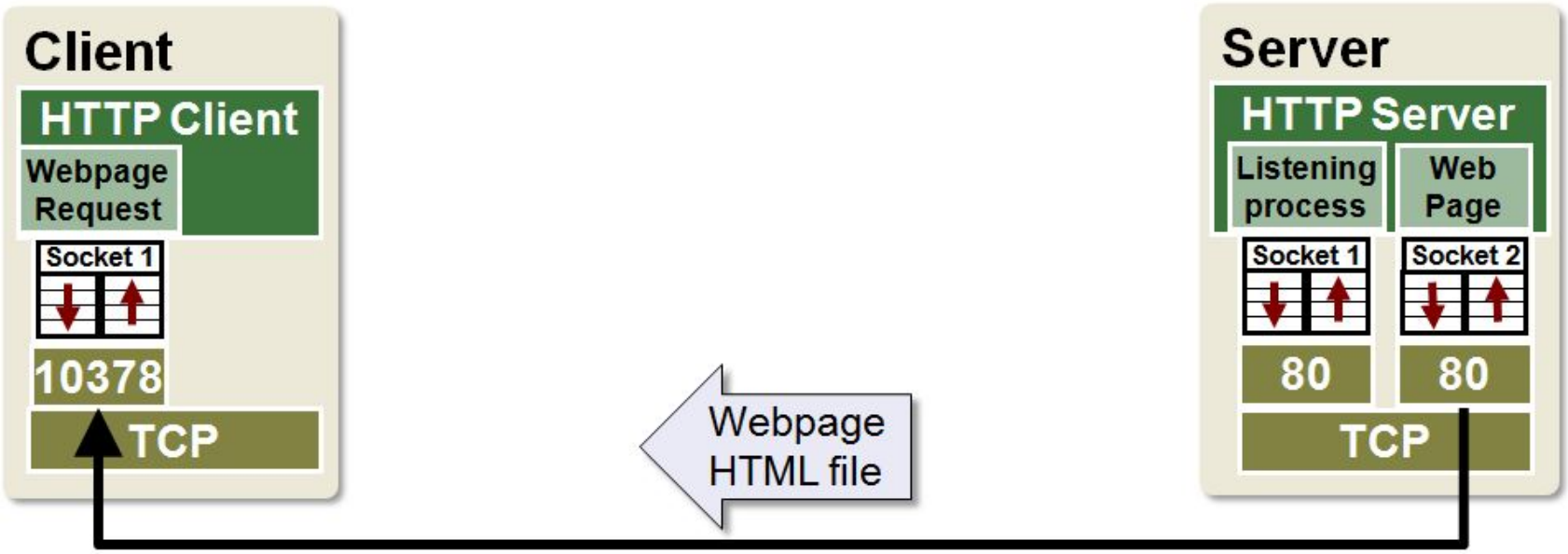
5. Transport Layer Delivers Message to Client

- The server's Transport layer periodically checks its transmit buffers to determine if a message needs to be sent.
- When a message is found it is forwarded to the destination address.

5. Transport Layer Delivers Message to Client

Client Sockets	Socket 1
Transport	TCP
Source Port	10378
Source IP Addr	192.168.1.101
Destination Port	80
Destination IP Addr	192.168.1.102

Server Sockets	Socket 1	Socket 2
Transport	TCP	TCP
Source Port	80	80
Source IP Addr	192.168.1.102	192.168.1.102
Destination Port	n/a	10378
Destination IP Addr	n/a	192.168.1.101



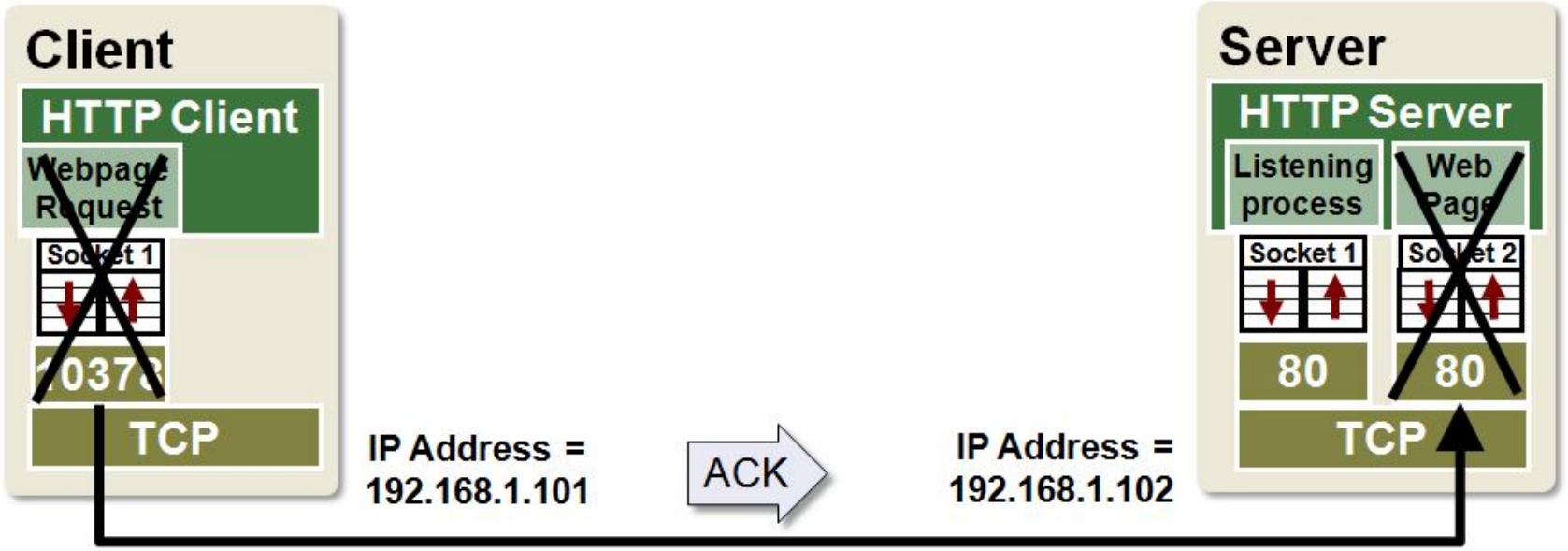
6. Sockets Closed

- After the client receives the web page it requested, it sends an acknowledge to the server then closes its socket.
- The server receives the client's acknowledge then closes its socket.

6. Sockets Closed

Client Sockets	Socket 1
Transport	TCP
Source Port	10378
Source IP Addr	192.168.1.101
Destination Port	80
Destination IP Addr	192.168.1.102

Server Sockets	Socket 1	Socket 2
Transport	TCP	TCP
Source Port	80	80
Source IP Addr	192.168.1.102	192.168.1.102
Destination Port	n/a	10378
Destination IP Addr	n/a	192.168.1.101



Berkeley Sockets

- **Berkeley sockets** is an industry standard **Application Programming Interface** (API) to create and use sockets.
- It was initially used as an API for the Unix operating system and was later adopted by TCP/IP.
- Berkeley defines 18 standard function names for this purpose. This graphic shows a few examples.

Berkeley Sockets

Function Name	Description
socket()	Create a socket
bind(), connect()	Assign a socket to an IP address & port #
send(), recv(), or write(), read()	Tx & Rx to and from the socket

Berkeley sockets are also sometimes referred to as BSD sockets (Berkeley Software Distribution), named for work done at the University of California, Berkeley, in the 1980's.

Berkeley Sockets

- The **socket()** function creates a socket on the host.
- The **bind()** function is typically used on the server side and assigns a socket to its local IP address and port number.
- **connect()** is typically used on the client side. It creates a socket and also attempts to establish a TCP or UDP connection with a server.
- **send()**, **recv()** and **write()**, **read()** are used to send and receive the messages to and from the socket.

Questions

True / false?

1. A client always needs to be running and listening for requests from servers.
2. Server applications are assigned standardized “well known” port numbers.
3. Sockets use source and destination MAC addresses to define a virtual channel between two hosts.